



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



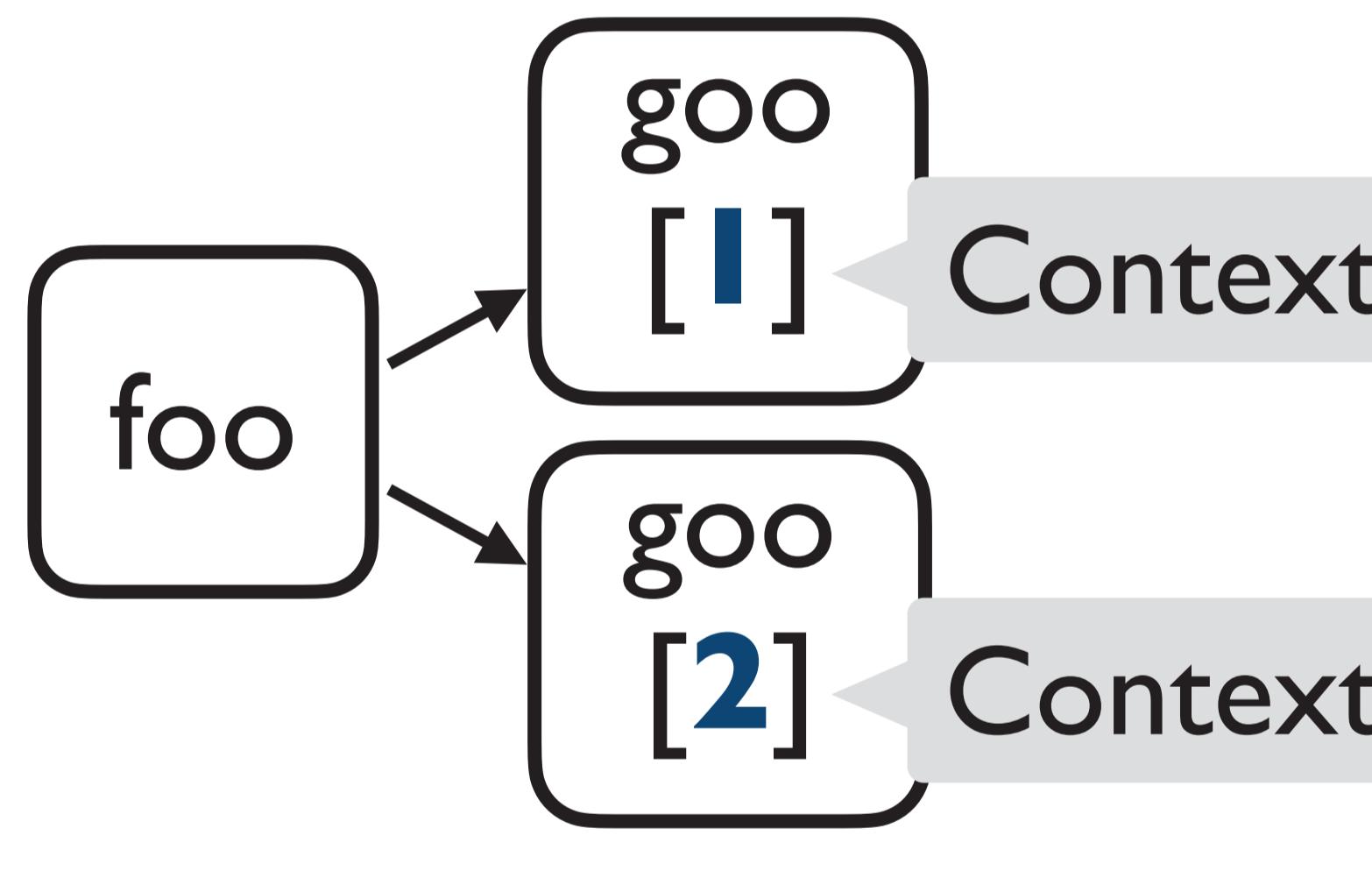
SW재난연구센터 workshop @ Jeju, Korea

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: }
```



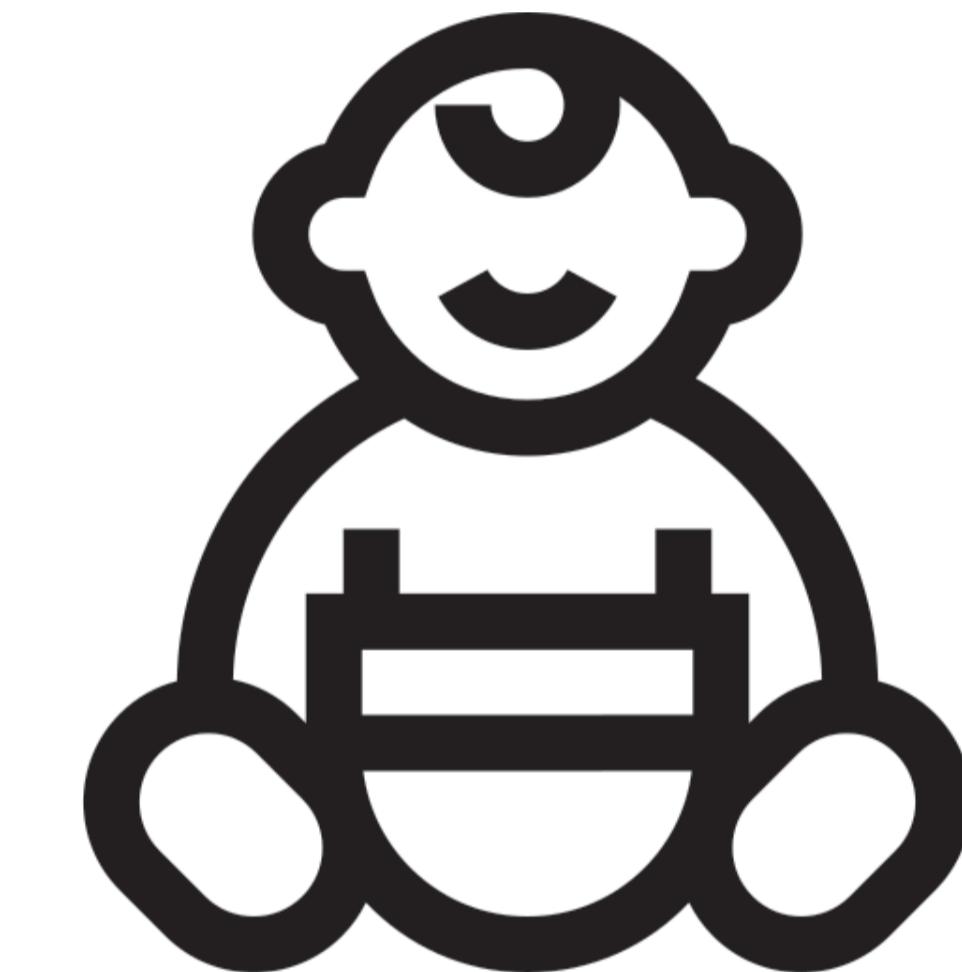
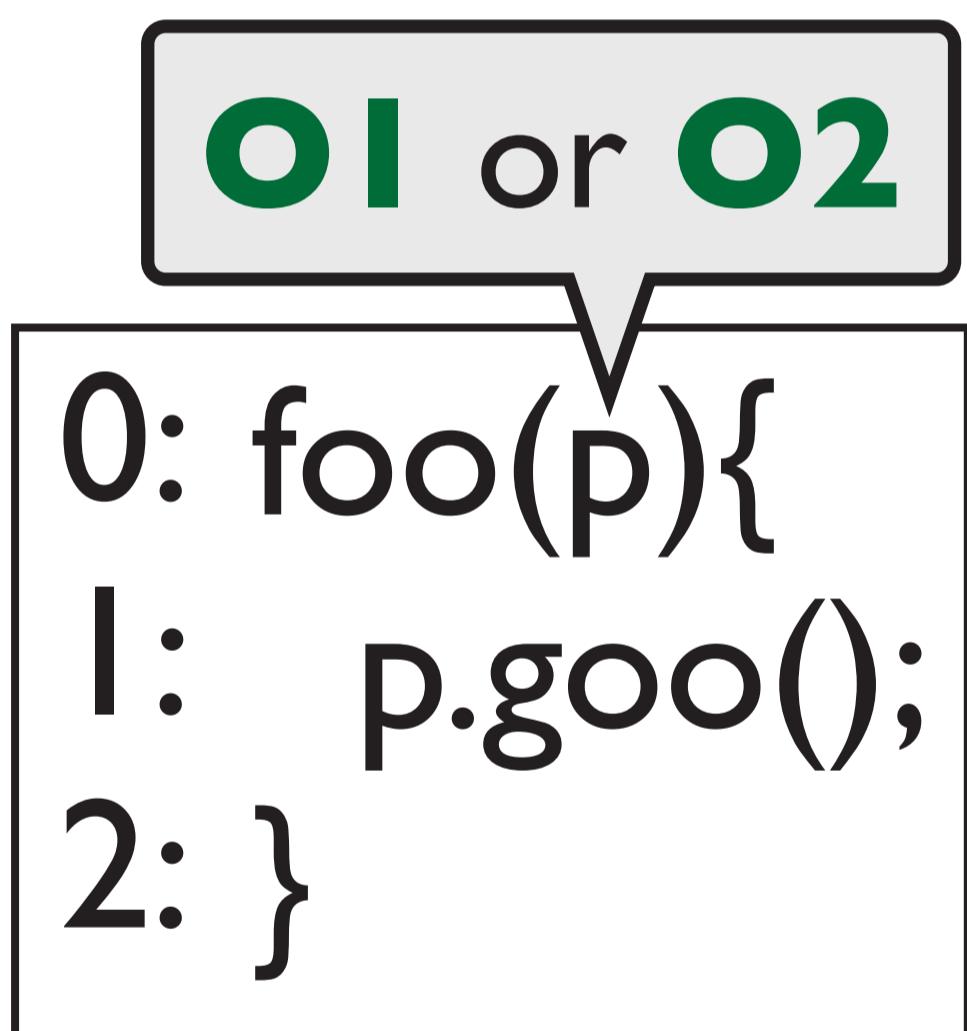
Call-site sensitivity



Call-site Sensitivity vs Object Sensitivity

Object sensitivity appeared in 2002

- Considers “**What**”

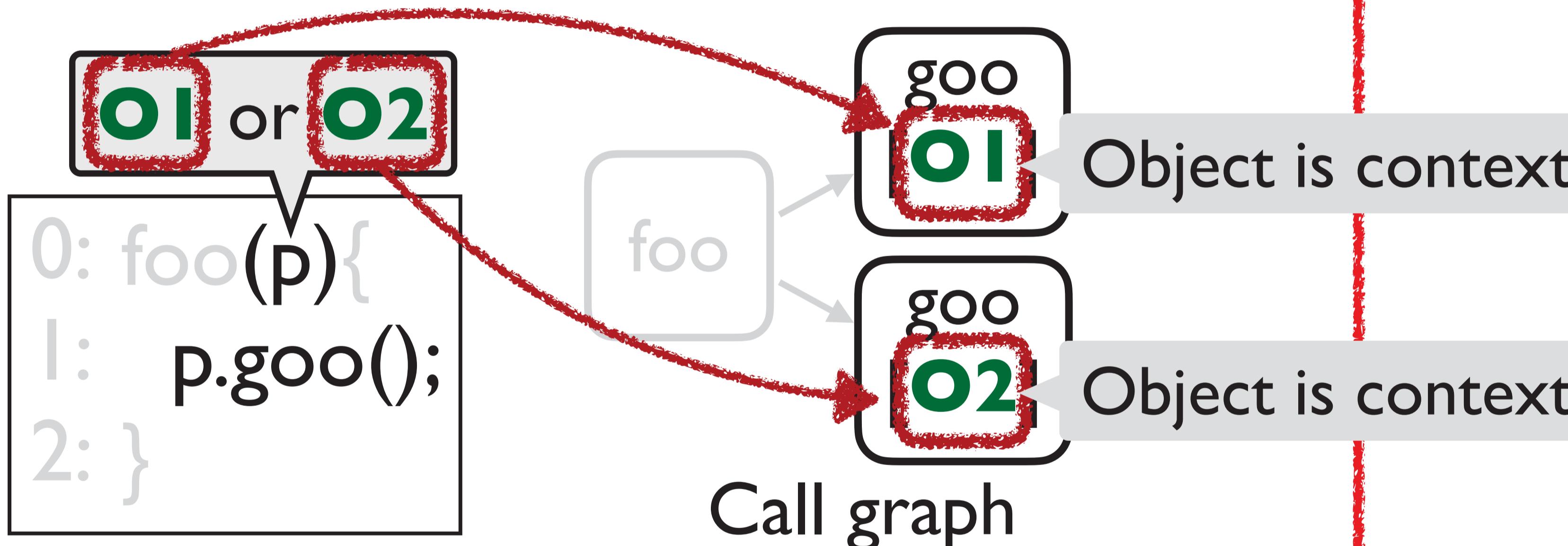


Object sensitivity

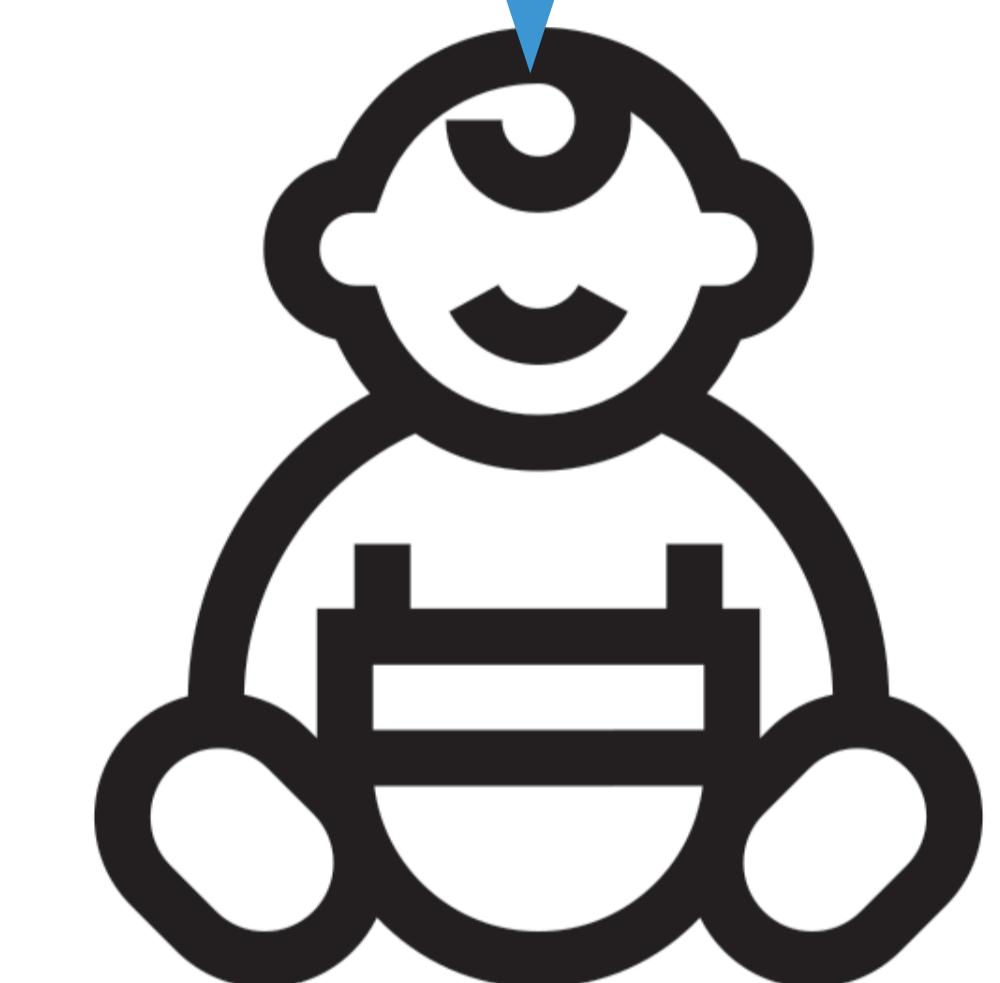
Call-site Sensitivity vs Object Sensitivity

Object sensitivity appeared in 2002

- Considers “**What**”



What is it called with?



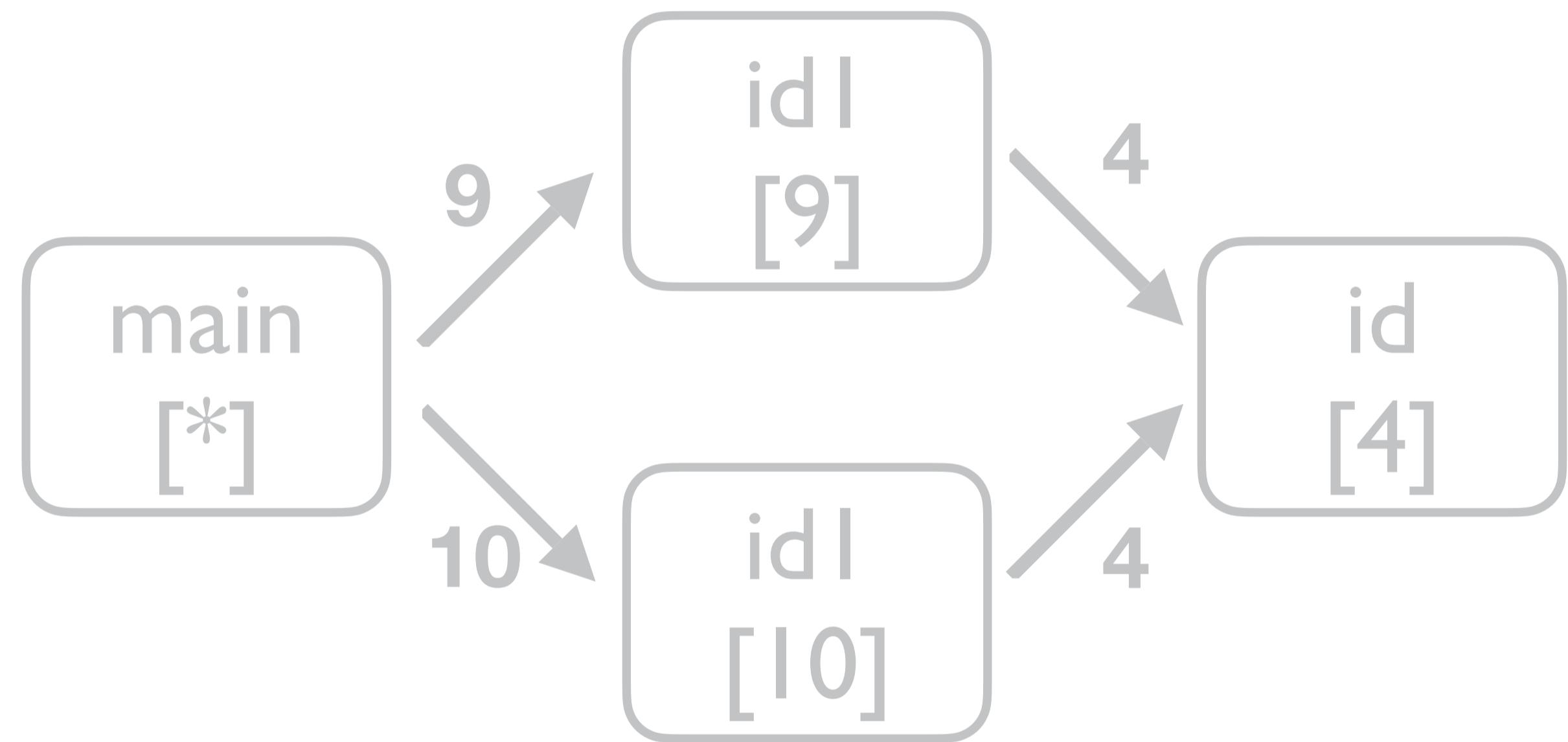
Object sensitivity



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(); //C1  
8:     c2 = new C(); //C2  
9:     a = (A) c1.id1(new A()); //query1  
10:    b = (B) c2.id1(new B()); //query2  
11: }
```



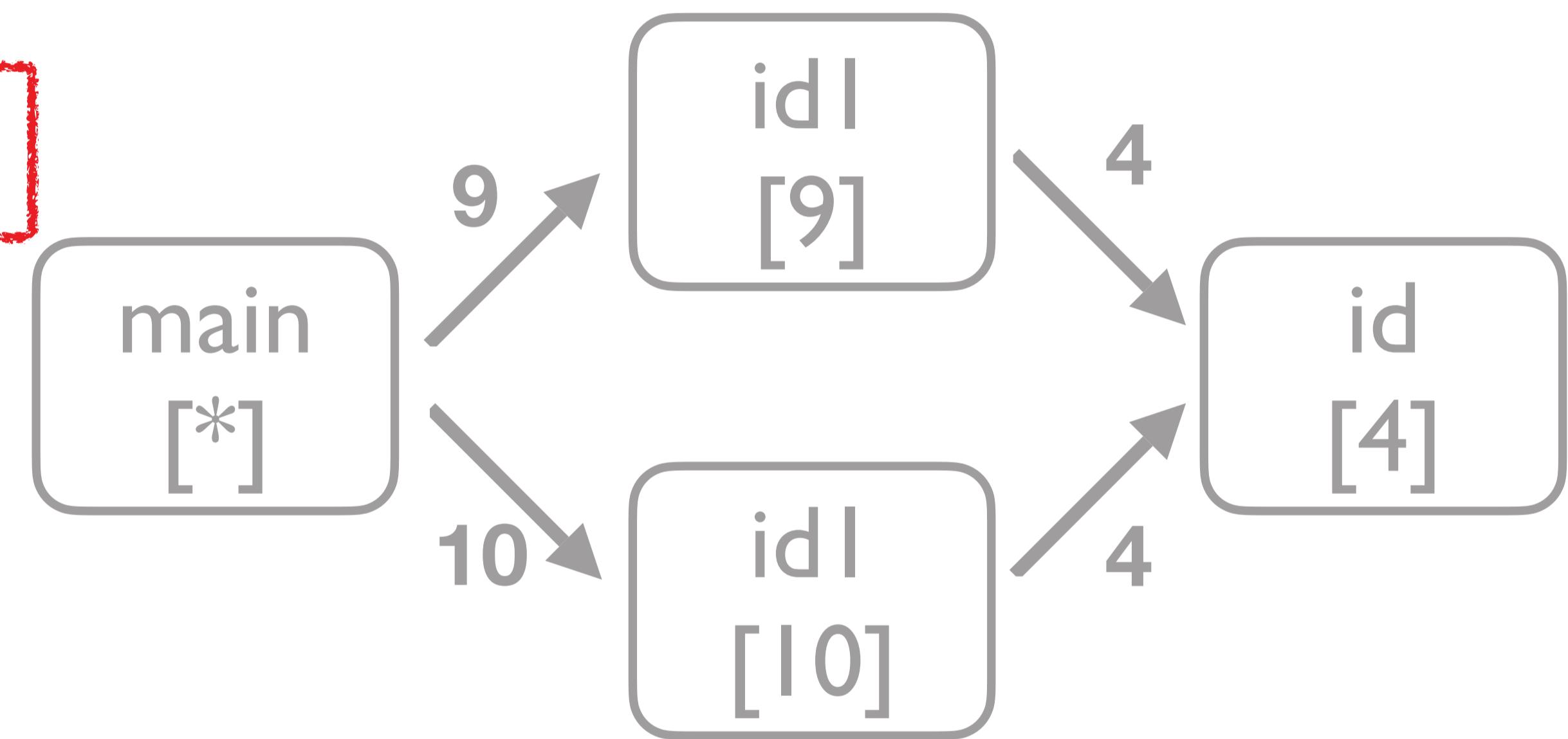
Call-graph of I-CFA

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:     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: }
```

Identity function



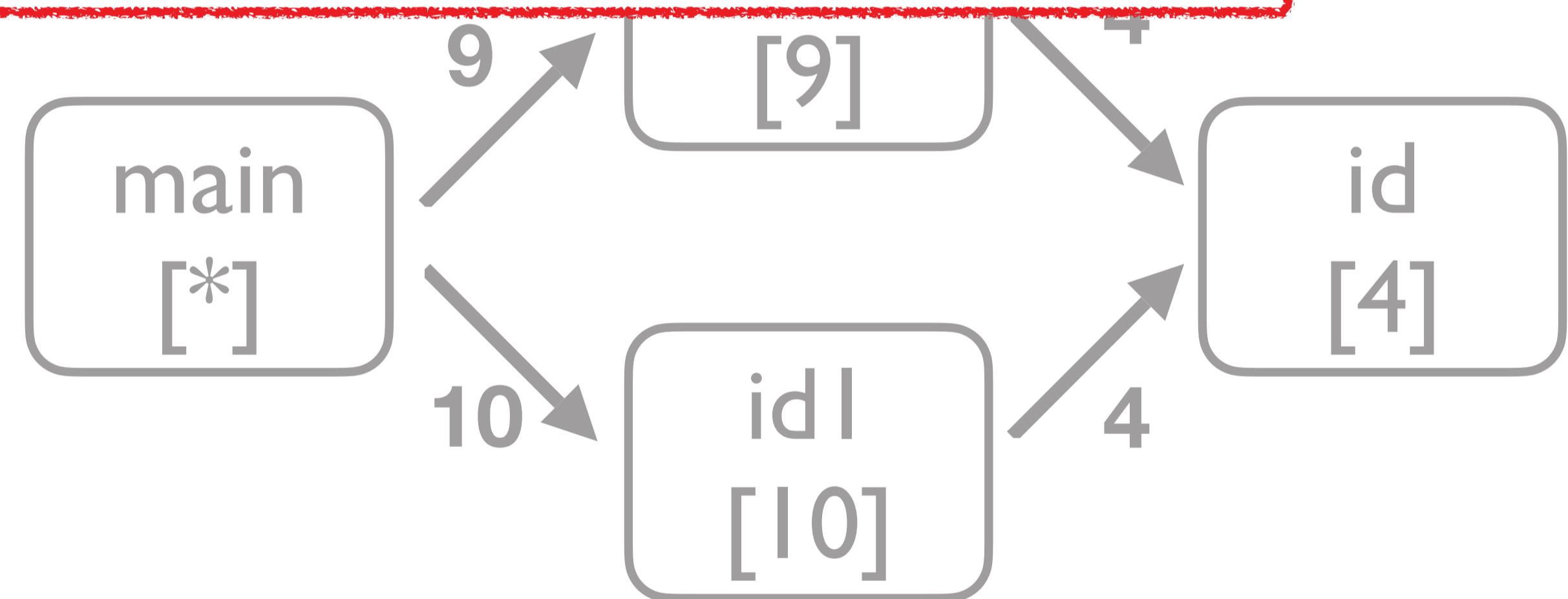
Call-graph of I-CFA

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



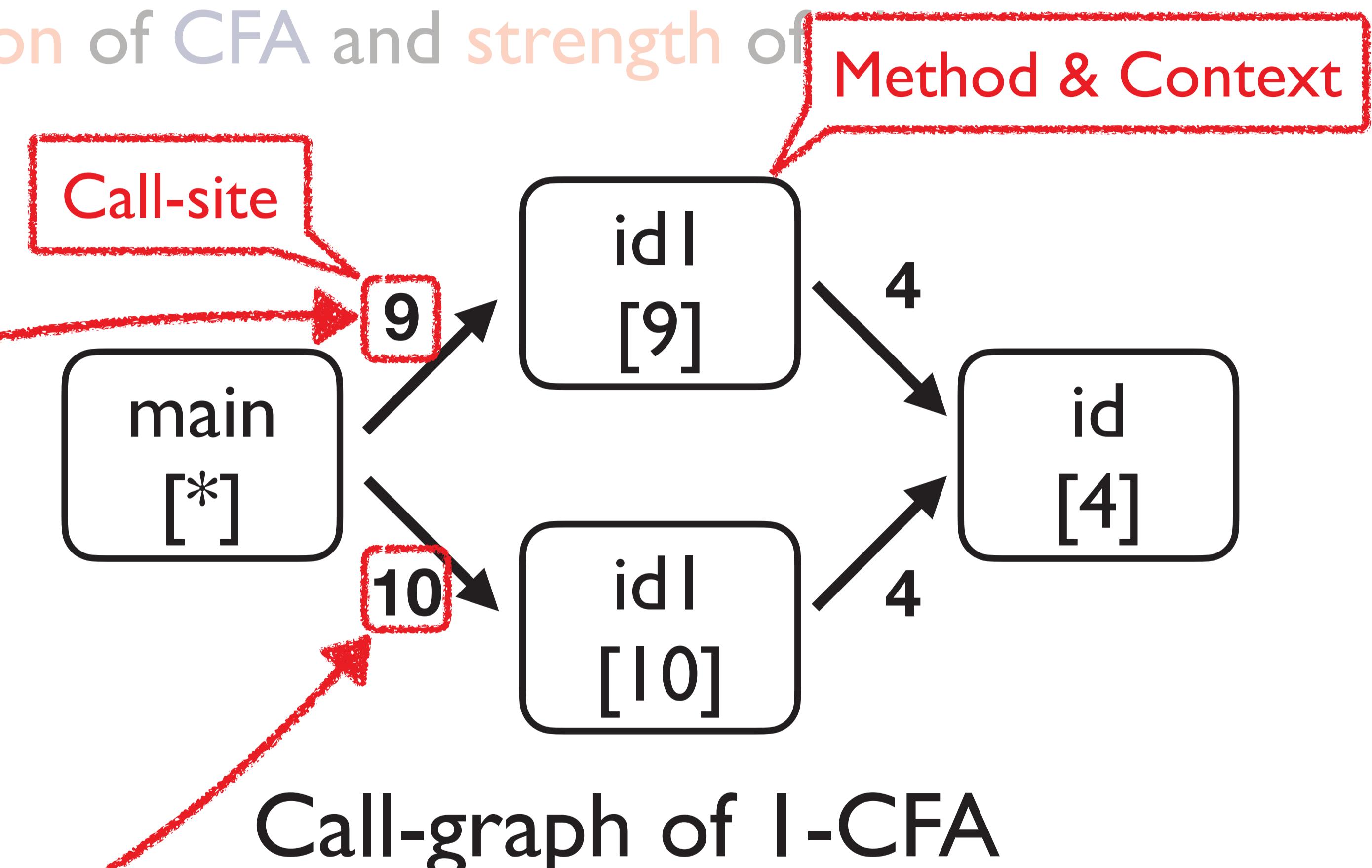
Call-graph of I-CFA

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: }
```

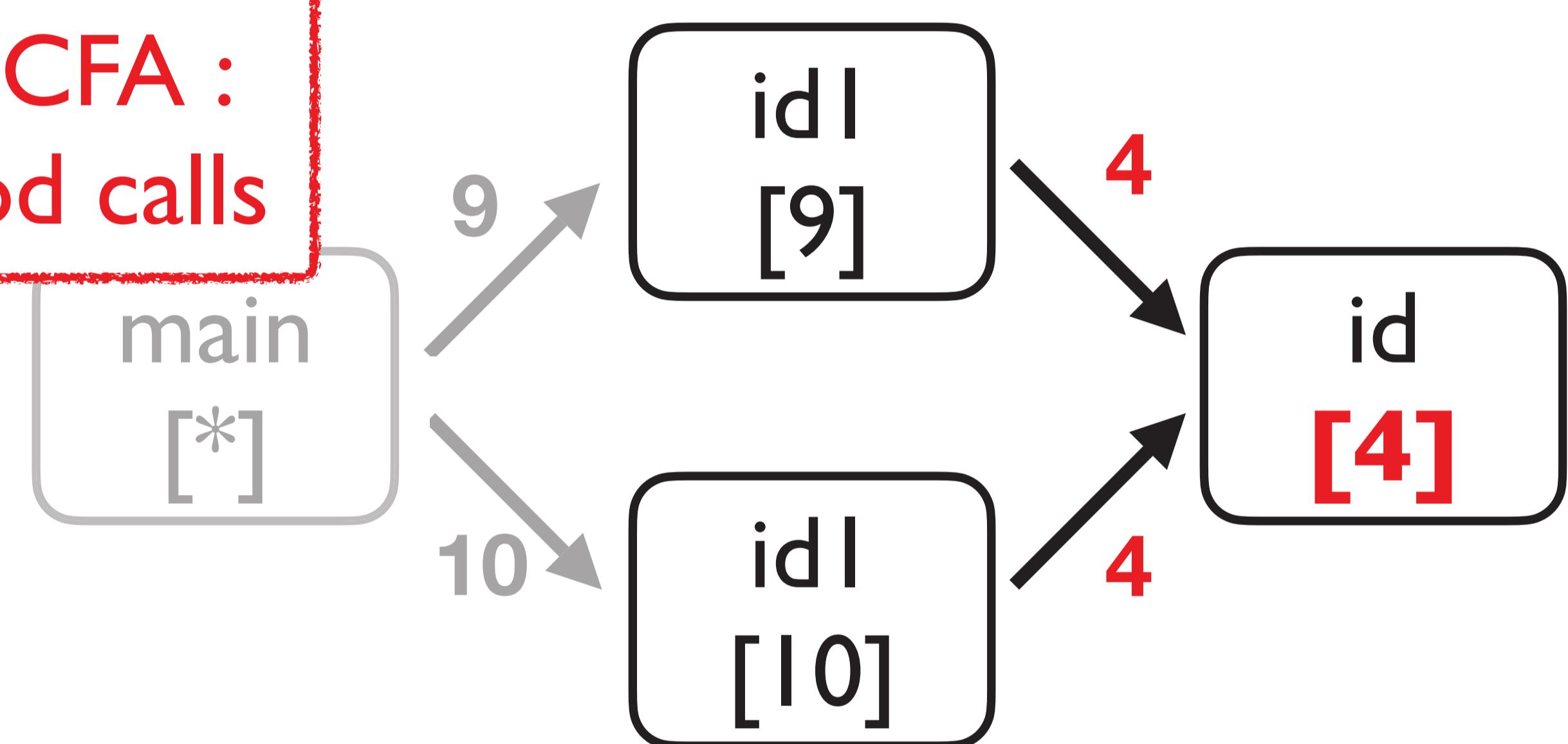


Call-site Sensitivity vs Object Sensitivity

- 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

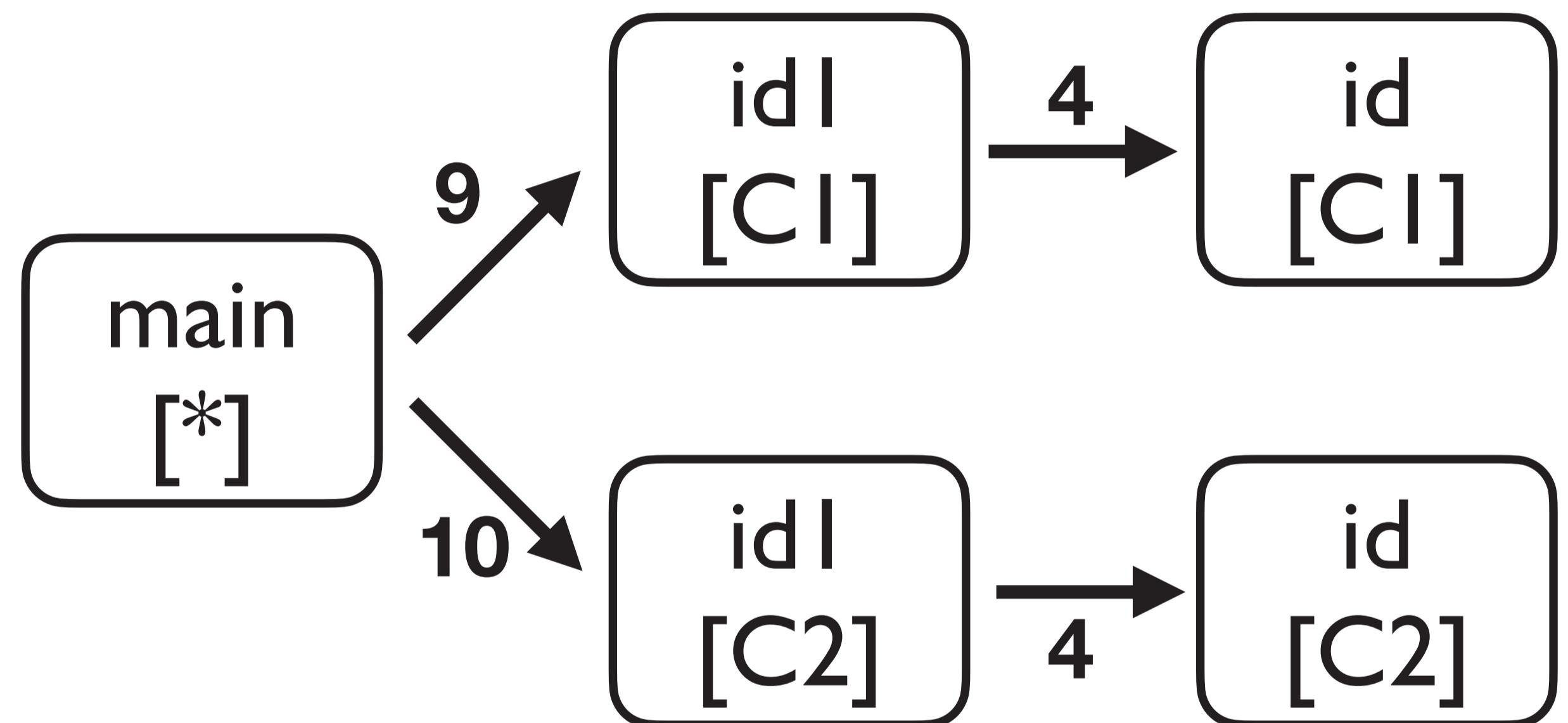


Call-graph of I-CFA

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: }
```

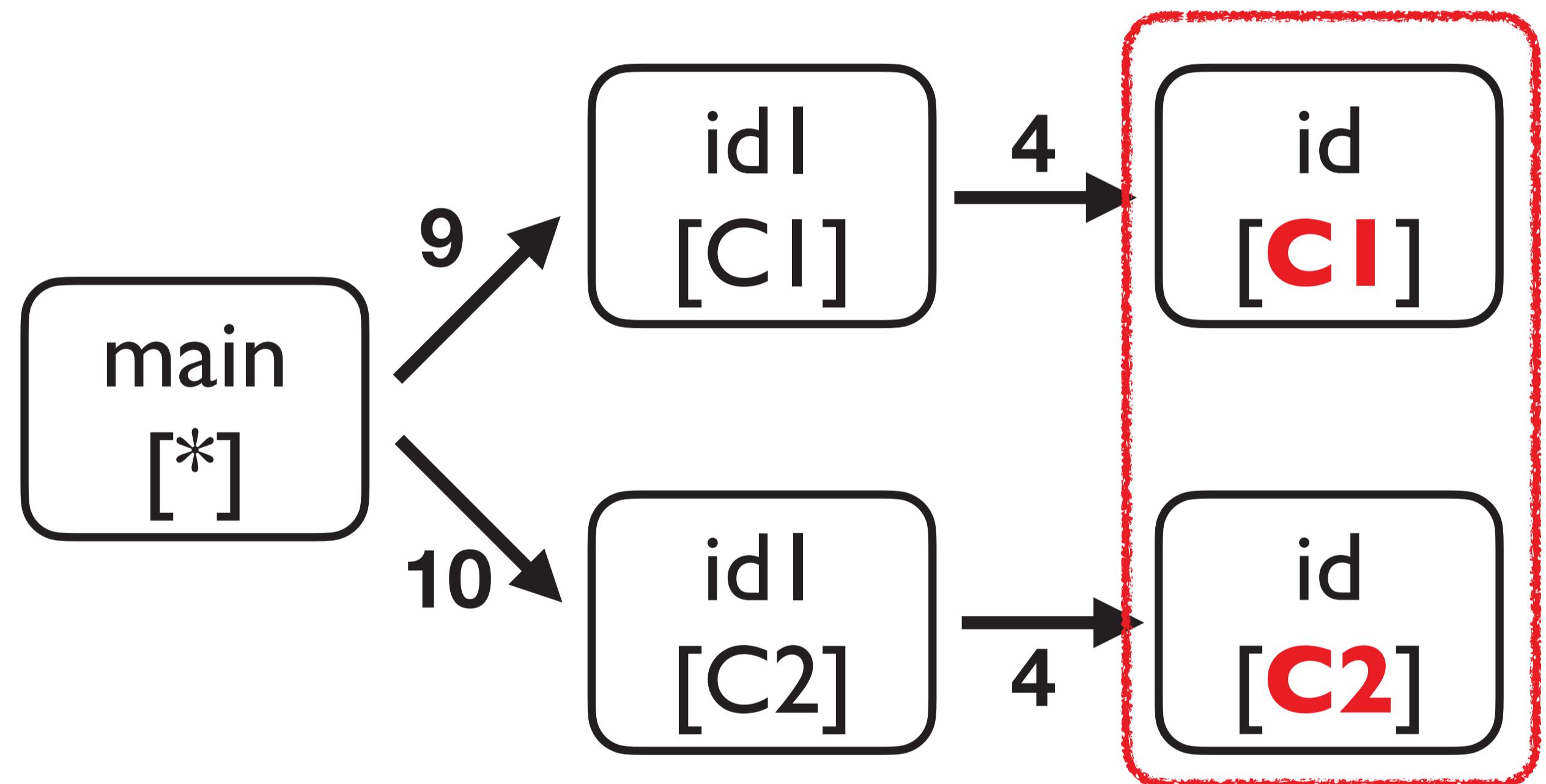


Call-graph of I-Obj

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: }
```

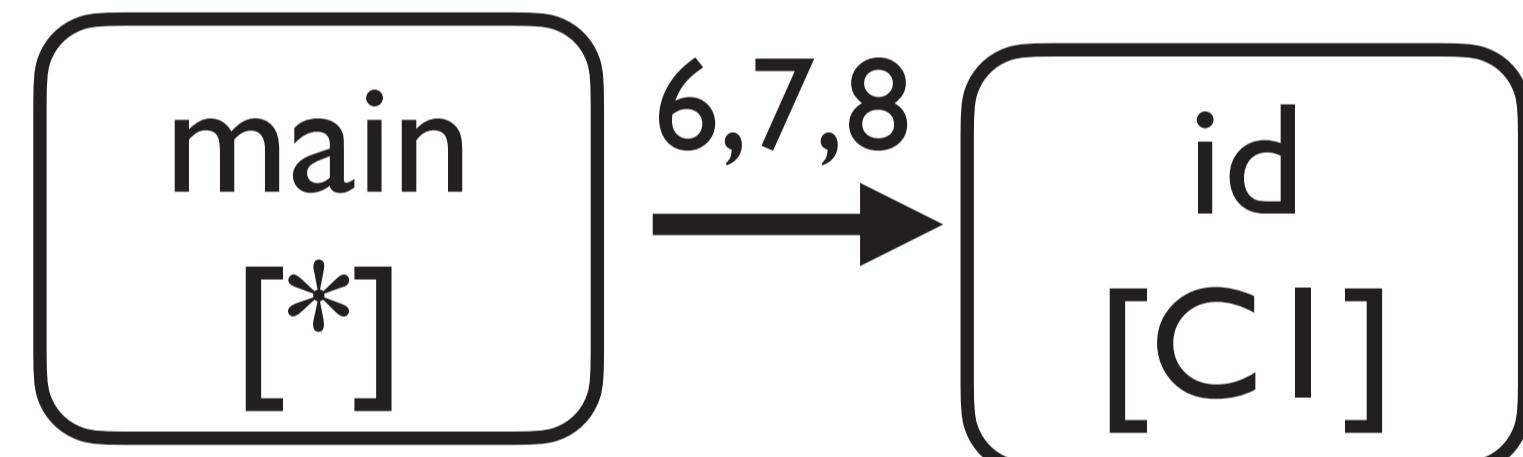


Call-graph of I-Obj

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();//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: }
```

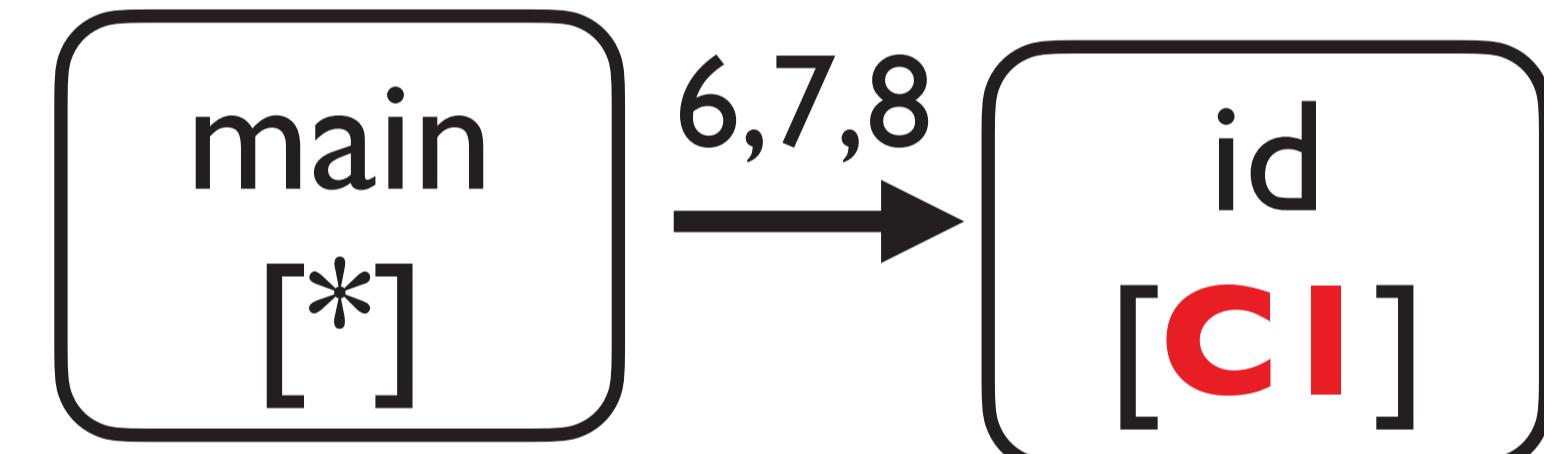


Call-graph of I-Obj

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: }
```



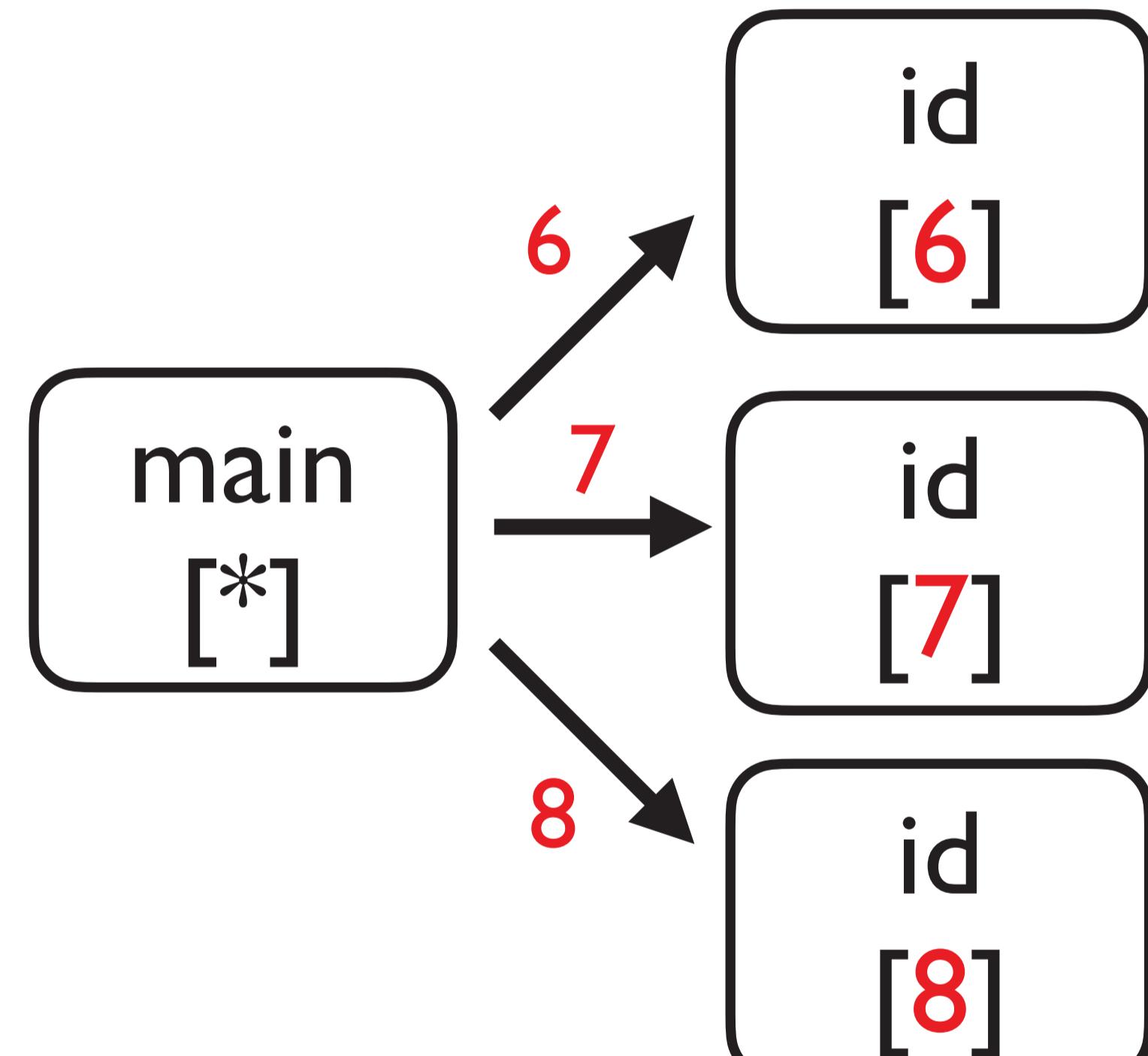
Call-graph of I-Obj

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: }
```

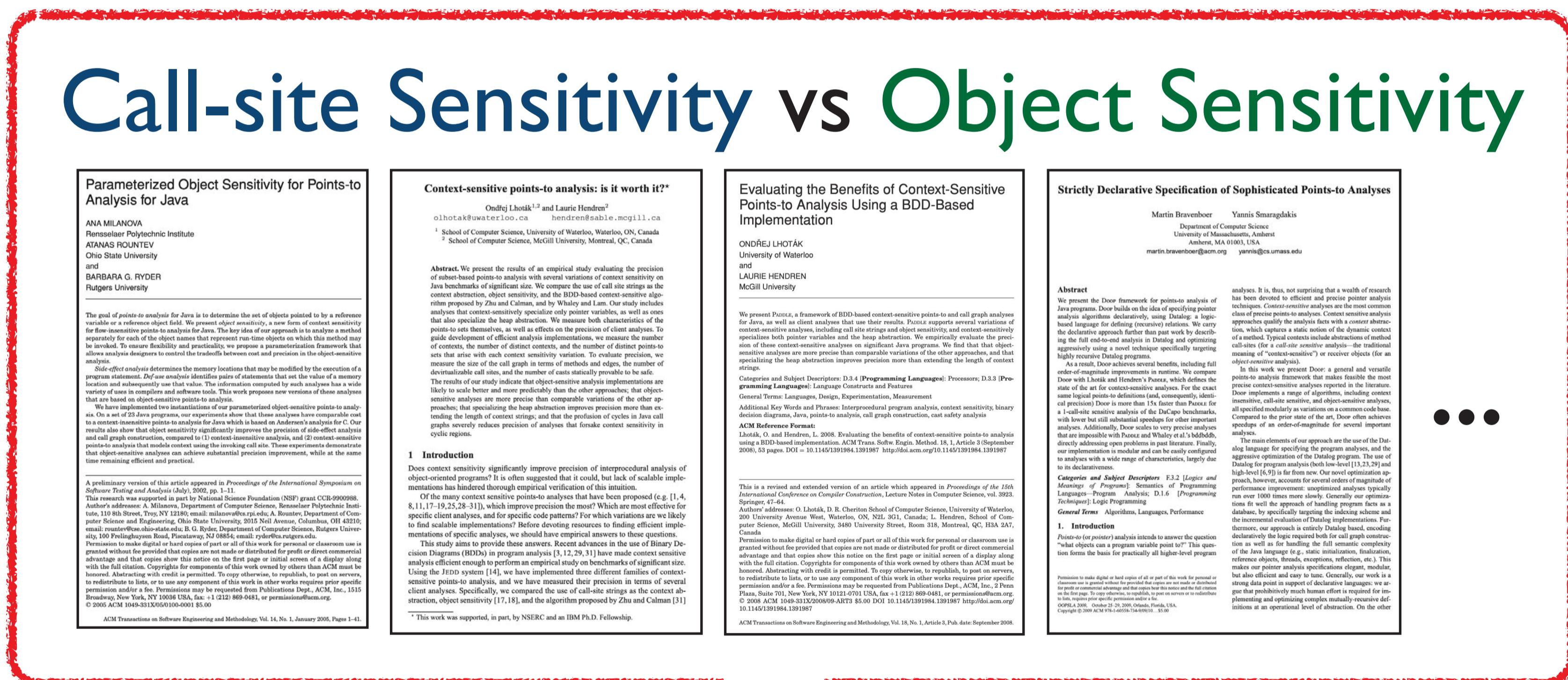


Call-graph of I-CFA

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



Obj vs CFA

A horizontal timeline graphic. It features a thick black line with vertical tick marks at both ends. A red, hand-drawn style wavy line is drawn across the black line, starting near the left tick mark and ending near the right tick mark. The years 1981, 2002, 2010, and 2022 are printed in large, bold, black, sans-serif font below the timeline.

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 given 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 flexibility and practicality, we propose a parameterized framework that allows users to choose the context abstraction and the precision of the analysis.

Side-effect analysis determines the memory locations that may be modified by the execution of a program statement. *Door* use analysis identifies pairs of statements that set the value of a memory location and subsequently use that value. This information computed by such analyses has a wide variety of applications, ranging from security analysis to work preserving versions of those analyses that are based on object-sensitive points-to analysis.

We have implemented two instantiations of our parameterized object sensitivity framework for Java analysis. Our first instantiation is *Door*, which shows that these analyses have comparable cost to 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. Our second instantiation is *CFA*, which shows that object sensitivity for 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 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 Liu. Our study includes analysis of context abstraction, object sensitivity, and context-sensitive analysis that specialize the heap abstraction. We empirically evaluate the precision of context abstraction, object sensitivity, and context-sensitive analysis that specialize the heap abstraction. We find that object-sensitive analyses are more precise than comparable variations of the other approaches, and that specializing the heap abstraction improves precision more than extending the length of context abstraction.

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

We present *Door*, a framework for BDD-based context-sensitive points-to and call graph analysis for Java, as well as client analysis that use their results. *Door* supports several variations of context-sensitive analysis, including call site strings and object sensitivity, and context-sensitivity specializes both pointer variables and the heap abstraction. We empirically evaluate the precision of context abstraction, object sensitivity, and context-sensitive analysis that specialize the heap abstraction. We find that object-sensitive analyses are more precise than comparable variations of the other approaches, and that specializing the heap abstraction improves precision more than extending the length of context abstraction.

Strictly Declarative Specification of Sophisticated Points-to Analyses
Martin Bravenboer Yannis Smaragdakis
Department of Computer Science
University of Massachusetts, Amherst
Amherst, MA 01003, USA
martin.bravenboer@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 *Datalog*: a logic-based language for defining (recursive) relations. We carry out a formal analysis of pointer analysis algorithms with a context abstraction, which captures a static notion of the dynamic context of a method. Typical contexts include abstractions of method calls (e.g., a context representing all possible contexts of a “vector-access” or receiver object) or receiver objects (as for an *object-sensitive* analysis).

In this work we present *Door*, a general and versatile points-to analysis framework that makes feasible the most precise context-sensitive analyses reported in the literature. Datalog implementations of algorithms, including context abstraction, 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, *Door* often achieves orders of magnitude better precision, even for very large 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 pointer analysis clearly distinguishes *Door* from state-of-the-art pointer analysis tools (e.g., [13, 23, 29]) and highlights its strengths. Our novel approach is an approach, however, accounts for several orders of magnitude of performance improvement: unoptimized analyses typically take minutes to hours to complete, whereas our optimizations fit well the approach of handling program facts as a database, by specifically targeting the indexing scheme and the query rewriting mechanism of *Datalog*. Furthermore, our approach is entirely *Datalog* based, encoding declaratively the logic required both for call graph construction and for pointer analysis, and thus requires no knowledge of the Java language (e.g., static initialization, finalization, reference objects, threads, exceptions, reflection, etc.).

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, pointer analysis, context abstraction, 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>

1 Introduction
Does context sensitivity significantly improve precision of interprocedural analysis of object-to-object pointers? It is often suggested that it could but has not been implemented in practice. This is hindered through empirical verification of this intuition.

Of the many context sensitive points-to analyses that have been proposed (e.g. [1, 4, 8, 11, 17–19, 25, 28–31]), which improve precision the most? Which are most effective for specific client analyses, and for specific code patterns? For which variations are we likely to find significant improvements in precision? What are the costs of these improvements? In the case of specific analyses, we should have empirical answers to these questions.

This study aims to provide these answers. Recent advances in the use of Binary Decision Diagrams (BDDs) in program analysis [3, 12, 29, 31] have made context sensitive analysis efficient enough to perform an empirical study on benchmarks of significant size.

Obj wins **Obj wins** **Obj wins** **Obj wins**



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

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

Hakjoo Oh AAA616 2019 Fall, Lecture 8

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);
    }
}
```

Yannis Smaragdakis University of Athens

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 for Java
 - 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 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 what instruction 3 `p` points to `z`. If this information is available we can perform a simple flow analysis of the function as follows:
 $f_{CP}[\ast p := y](\sigma) = [z \rightarrow \sigma(y)]\sigma$ where `must-point-to` is a relation between variables and pointers.
When we know exactly what a variable `z` points to, we say that `z` has `must-point-to` information, and we can perform a *strong update* of variable `z`, because we know with confidence that assigning to `z` will not result in any pointer aliasing over all the locations that `z` points to. How is this possible? It is not possible in C or Java, which are languages with pass-by-value, for example C++, it is possible to have the same location point to different objects at the same time. Of course, it is also possible that we are uncertain to which of the two distinct locations `p` points. For example:

now the essence of knowledge

KOREA UNIVERSITY

National and Kapodistrian University of Athens

HARVARD

Carnegie Mellon University

Obj

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 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

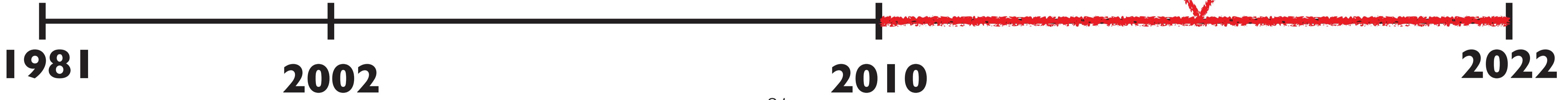
Call-site Sensitivity vs Object Sensitivity

- Researches focused on improving Object Sensitivity

Researches on Object Sensitivity

The red-bordered box contains five research papers:

- Making k -Object-Sensitive Pointer Analysis More Precise with Still k -Limiting**
Tian Tan¹, Yannis Smaragdakis²
1 School of Computer Science, University of Aarhus, Denmark
2 Advanced Innovation Center, University of Athens, Greece
Abstract: Object-sensitive pointer analysis is vital for precision and good performance. The state-of-the-art k -object-sensitive pointer analysis (k-OAPA) combines both flavors of context sensitivity but is limited to $k=1$. We propose a new generation of call-site- and object-sensitive pointer analysis that can handle $k>1$ without significantly impacting precision or performance. We show that our approach only requires a small modification to the existing k-OAPA framework.
- Efficient and Scalable Context-Sensitive Pointer Analysis for Object-Oriented Languages**
Tian Tan¹, Anders Møller², Yannis Smaragdakis³
1 School of Computer Science, University of Aarhus, Denmark
2 Department of Informatics, University of Aarhus, Denmark
3 Department of Computer Science, University of Massachusetts Amherst, USA
Abstract: Context-sensitive pointer analysis has emerged as a key technique for analyzing object-oriented programs. The state-of-the-art, however, object-sensitive pointer analysis (OSPA) implementations deviate significantly from the theoretical guarantees of OSPA, resulting in a loss of precision and a reduction in degrees of freedom, relating to at every method call and object creation. We present a spectrum of analyses of object-sensitive pointer analysis. The results show that past implementations contract to the severe detriment of precision. Our work shows that higher precision and often better performance can be achieved by approximating of object-sensitive pointer analysis at substantially reduced cost. Our work also shows that OSPA tools are not dynamic types of objects and therefore do not support object-sensitive pointer analysis. Our results expose the influence of pointer analysis and demonstrate its importance in improving the quality of a spectrum of analyses that range from static analysis to memory safety analysis (comparable to analyses that measure the reachability of memory).
- Precision-Guided Context-Sensitive Pointer Analysis**
YUE LI¹, TIAN TAN², ANDERS MØLLER², YANNIS SMARAGDAKIS³
1 Aarhus University, Denmark
2 Aarhus University, Denmark
3 University of Athens, Greece
Abstract: Mainstream pointer-to-analysis techniques for languages rely predominantly on the analysis of pointers to model heap objects. We propose a more general form of pointer-to-analysis that is specifically designed to meet the needs of an important class of type-dependent pointer analysis, namely, object-oriented pointer analysis (OOPA). OOPA is a challenging problem, e.g., a context-insensitive graph (CIG) for a Java program is often too large to be analyzed in full. We propose a new form of pointer-to-analysis that is based on a graph abstraction of the CIG. This abstraction is called a type-consistent object that is created by tree abstraction. MAHONG enables an algorithmic abstraction of pointer-to-analysis that is nearly the same precision for type-dependent pointer analysis as the original pointer-to-analysis. We present the SCALAR framework that addresses this problem. SCALAR efficiently estimates the amount of pointers to information that are needed to analyze each method so that the total number of pointers is minimized. We evaluate the precision of SCALAR against state-of-the-art pointer-to-analysis approaches. Our experimental results demonstrate that SCALAR achieves the same precision as state-of-the-art pointer-to-analysis approaches but with a substantially lower computational cost.
- Scalability-First Partial Context-Sensitive Pointer Analysis**
Tian Tan¹, Yannis Smaragdakis³
1 Aarhus University, Denmark
3 University of Athens, Greece
Abstract: Context-sensitive pointer analysis is an essential technique for analyzing object-oriented programs. The state-of-the-art pointer-to-analysis approaches for Java programs are not able to handle programs with millions of objects. We propose a new form of pointer-to-analysis that is called scalability-first partial context-sensitive pointer analysis (S-PCPA). S-PCPA is based on a graph abstraction of pointer-to-analysis. It is able to handle programs with millions of objects. We present the SCALAR framework that addresses this problem. SCALAR efficiently estimates the amount of pointers to information that are needed to analyze each method so that the total number of pointers is minimized. We evaluate the precision of SCALAR against state-of-the-art pointer-to-analysis approaches. Our experimental results demonstrate that SCALAR achieves the same precision as state-of-the-art pointer-to-analysis approaches but with a substantially lower computational cost.
- Data-Driven Context-Sensitive Pointer Analysis**
SEHUN JEONG¹, MINSEOK JEON², YANNIS SMARAGDAKIS³
1 Korea University, Republic of Korea
2 Korea University, Republic of Korea
3 University of Athens, Greece
Abstract: Context-sensitive pointer analysis is an important technique for analyzing object-oriented programs. The state-of-the-art pointer-to-analysis approaches for Java programs are not able to handle programs with millions of objects. We propose a new form of pointer-to-analysis that is called data-driven context-sensitive pointer analysis (D-CPA). D-CPA is based on a graph abstraction of pointer-to-analysis. It is able to handle programs with millions of objects. We present the SCALAR framework that addresses this problem. SCALAR efficiently estimates the amount of pointers to information that are needed to analyze each method so that the total number of pointers is minimized. We evaluate the precision of SCALAR against state-of-the-art pointer-to-analysis approaches. Our experimental results demonstrate that SCALAR achieves the same precision as state-of-the-art pointer-to-analysis approaches but with a substantially lower computational cost.



Call-site Sensitivity vs Object Sensitivity

- Call-site Sensitivity has been ignored

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

The box contains five academic papers:

- A Machine-Learning Algorithm with Disjunctive Models for Data-driven Program Analysis** (1981)
- Making k-Object-Sensitive Pointer Analysis More Precise with Still k-Limiting** (1981)
- Scalability-First Pointer Analysis: Self-Tuning Context-Sensitivity** (1992)
- Pick Your Contexts Well: Understanding Object-Sensitivity** (1998)
- Hybrid Context-Sensitivity for Points-To Analysis** (2002)
- Precision-Guided Context Sensitivity for Point-to Analysis** (2008)
- Introspective Analysis: Context-Sensitivity, Across the Board** (2018)

1981

2002

2010

2022

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]

A Machine-Learning Algorithm with Disjunctive Models for Data-driven Program Analysis
MINSEOK JON, SEHUN JEONG, SUNGDEOK CHA, and HAKJOON OH¹, a
Republic of Korea

We present a new machine-learning algorithm with disjunctive models for data-driven program analysis. Our key findings in static pointer analysis is a substantial amount of manual effort required for the analysis performance. Recently, data-driven program analysis has emerged to address this problem by automatically adjusting the analysis based on data through a learning algorithm. All previous work on data-driven program analysis is limited to the analysis of static pointer analysis to simple-minded learning models and algorithms that are unable to capture sophisticated dynamic behaviors. In this paper, we propose a learning-based approach for data-driven program analysis as well as a learning algorithm to find the model parameters based on various pointer features and therefore is able to express nonlinear combined programs. Key techniques is to employ a learning-based approach to find the model parameters that would simply be impractical. We present a stepwise and greedy algorithm that handles the learning of the model parameters and the analysis of the learned model. The proposed context-sensitive pointer-to analysis for Java and flow-sensitive interval analysis for C. Experimental results demonstrate that our automated technique significantly improves the performance of the state-of-the-art pointer-to analysis.

CSCS Concepts: Theory of computation → Program analysis; Computing method → machine learning approaches.

Additional Key Words and Phrases: Data-driven program analysis; Static analysis; Context sensitivity

ACM Reference Format:
Minseok Jon, Sehun Jeong, Sungdeok Cha, and Hakjoon Oh. 2017. A Machine-Learning Algorithm with Disjunctive Models for Data-Driven Program Analysis. *ACM Trans. Program. Lang. Syst.* December 2017, 42 pages. <https://doi.org/10.1145/3000000.3000001>

1 INTRODUCTION
One major challenge in static program analysis is a substantial amount of manual effort for the analysis performance for real-world applications. Practical static analysis and variety of heuristics to optimize their performance. For example, context-sensitive analysis for optimizing object-sensitivity for real-world static analyzers apply context-sensitivity methods determined by some local rules [Smaragdakis et al. 2014]. Another solution is to use pointer-to analysis with Octree-based [2008]. Because it is impacted of all variable relationships in the program, static analyzers employ variable-clause

Making k-Object-Sensitive Pointer Analysis More Precise with Still k-Limiting
Tian Tan¹, Yu Li¹, and Jingling Xue^{1,2*}
¹ School of Computer Science and Engineering, UNSW Australia Advanced Innovation Center for Imaging Technology, CNF, CSIRO Data61, Australia
² Department of Mathematics, University of California, Berkeley, CA, USA

Abstract Object-sensitivity is regarded as equally the best choice for pointer analysis in object-oriented languages. However, existing pointer analysis tools for object-sensitivity are often too conservative to achieve one that leads to reasonable analysis time and obtain high precision. In this paper, we introduce BRAS, a general approach improving the precision of object-sensitivity pointer analysis for data-driven program analysis as well as a learning algorithm to find the model parameters based on various pointer features and therefore is able to express nonlinear combined programs. Key techniques is to employ a learning-based approach to find the model parameters that would simply be impractical. We present a stepwise and greedy algorithm that handles the learning of the model parameters and the analysis of the learned model. The proposed context-sensitive pointer-to analysis for Java and flow-sensitive interval analysis for C. Experimental results demonstrate that our automated technique significantly improves the performance of the state-of-the-art pointer-to analysis.

CSCS Concepts: Theory of computation → Program analysis; Computing method → machine learning approaches.

Additional Key Words and Phrases: Data-driven program analysis; Static analysis; Context sensitivity

ACM Reference Format:
Tian Tan, Yu Li, and Jingling Xue. 2016. Making k-Object-Sensitive Pointer Analysis More Precise with Still k-Limiting. *ACM Trans. Program. Lang. Syst.* December 2016, 42 pages. <https://doi.org/10.1145/3000000.3000001>

Scalability-First Pointer Analysis: Self-Tuning Context-Sensitivity
Yu Li¹, Tian Tan¹, and Jingling Xue^{1,2*}
¹ School of Computer Science and Engineering, UNSW Australia Advanced Innovation Center for Imaging Technology, CNF, CSIRO Data61, Australia
² Department of Mathematics, University of California, Berkeley, CA, USA

Abstract Context-sensitivity is important in pointer analysis to ensure high precision, but existing techniques suffer from unpredictable high cost. In this paper, we propose a self-tuning context-sensitivity to achieve one that leads to reasonable analysis time and obtain high precision. In this paper, we introduce BRAS, a general approach improving the precision of object-sensitivity pointer analysis for data-driven program analysis as well as a learning algorithm to find the model parameters based on various pointer features and therefore is able to express nonlinear combined programs. Key techniques is to employ a learning-based approach to find the model parameters that would simply be impractical. We present a stepwise and greedy algorithm that handles the learning of the model parameters and the analysis of the learned model. The proposed context-sensitive pointer-to analysis for Java and flow-sensitive interval analysis for C. Experimental results demonstrate that our automated technique significantly improves the performance of the state-of-the-art pointer-to analysis.

CSCS Concepts: Theory of computation → Program analysis; Computing method → machine learning approaches.

Additional Key Words and Phrases: Data-driven program analysis; Static analysis; Context sensitivity

ACM Reference Format:
Yu Li, Tian Tan, and Jingling Xue. 2016. Scalability-First Pointer Analysis: Self-Tuning Context-Sensitivity. *ACM Trans. Program. Lang. Syst.* December 2016, 42 pages. <https://doi.org/10.1145/3000000.3000001>

Pick Your Contexts Well: Understanding Object-Sensitivity
Yannis Smaragdakis¹, Martin Breuer², David R. Chelius³, and George Karitsis⁴
¹ Department of Informatics, University of Crete, Greece
² Lehigh University, Bethlehem, PA, USA
³ Two-Millennia Plus, Andover, MA, USA
⁴ Department of Informatics, University of Crete, Greece

Abstract Object-sensitivity has emerged as an increasingly dominant paradigm in pointer analysis for object-oriented languages. However, existing pointer analysis tools for object-sensitivity are often too conservative to achieve one that leads to reasonable analysis time and obtain high precision. In this paper, we introduce BRAS, a general approach improving the precision of object-sensitivity pointer analysis for data-driven program analysis as well as a learning algorithm to find the model parameters based on various pointer features and therefore is able to express nonlinear combined programs. Key techniques is to employ a learning-based approach to find the model parameters that would simply be impractical. We present a stepwise and greedy algorithm that handles the learning of the model parameters and the analysis of the learned model. The proposed context-sensitive pointer-to analysis for Java and flow-sensitive interval analysis for C. Experimental results demonstrate that our automated technique significantly improves the performance of the state-of-the-art pointer-to analysis.

CSCS Concepts: Theory of computation → Program analysis; Computing method → machine learning approaches.

Additional Key Words and Phrases: Data-driven program analysis; Static analysis; Context sensitivity

ACM Reference Format:
Yannis Smaragdakis, Martin Breuer, David R. Chelius, and George Karitsis. 2016. Pick Your Contexts Well: Understanding Object-Sensitivity. *ACM Trans. Program. Lang. Syst.* December 2016, 42 pages. <https://doi.org/10.1145/3000000.3000001>

Hybrid Context-Sensitivity for Points-To Analysis
George Karitsis¹, Yannis Smaragdakis², and Andrei Balanouras³
¹ Department of Informatics, University of Crete, Greece
² Department of Informatics, University of Crete, Greece
³ Department of Informatics, University of Crete, Greece

Abstract Context-sensitivity is an invaluable approach for achieving high precision with good performance. However, existing context-sensitivity approaches are often too conservative to achieve one that leads to reasonable analysis time and obtain high precision. In this paper, we introduce BRAS, a general approach improving the precision of object-sensitivity pointer analysis for data-driven program analysis as well as a learning algorithm to find the model parameters based on various pointer features and therefore is able to express nonlinear combined programs. Key techniques is to employ a learning-based approach to find the model parameters that would simply be impractical. We present a stepwise and greedy algorithm that handles the learning of the model parameters and the analysis of the learned model. The proposed context-sensitive pointer-to analysis for Java and flow-sensitive interval analysis for C. Experimental results demonstrate that our automated technique significantly improves the performance of the state-of-the-art pointer-to analysis.

CSCS Concepts: Theory of computation → Program analysis; Computing method → machine learning approaches.

Additional Key Words and Phrases: Data-driven program analysis; Static analysis; Context sensitivity

ACM Reference Format:
George Karitsis, Yannis Smaragdakis, and Andrei Balanouras. 2016. Hybrid Context-Sensitivity for Points-To Analysis. *ACM Trans. Program. Lang. Syst.* December 2016, 42 pages. <https://doi.org/10.1145/3000000.3000001>

Precision-Guided Context Sensitivity for Point-to Analysis
Tania Smaragdaki¹, George Karitsis², and George Balousros³
¹ Department of Informatics, University of Crete, Greece
² Department of Informatics, University of Crete, Greece
³ Department of Informatics, University of Crete, Greece

Abstract Context-sensitivity is the primary approach for adding more precision without paying a heavy price. However, existing context-sensitivity approaches are often too conservative to achieve one that leads to reasonable analysis time and obtain high precision. In this paper, we introduce BRAS, a general approach improving the precision of object-sensitivity pointer analysis for data-driven program analysis as well as a learning algorithm to find the model parameters based on various pointer features and therefore is able to express nonlinear combined programs. Key techniques is to employ a learning-based approach to find the model parameters that would simply be impractical. We present a stepwise and greedy algorithm that handles the learning of the model parameters and the analysis of the learned model. The proposed context-sensitive pointer-to analysis for Java and flow-sensitive interval analysis for C. Experimental results demonstrate that our automated technique significantly improves the performance of the state-of-the-art pointer-to analysis.

CSCS Concepts: Theory of computation → Program analysis; Computing method → machine learning approaches.

Additional Key Words and Phrases: Data-driven program analysis; Static analysis; Context sensitivity

ACM Reference Format:
Tania Smaragdaki, George Karitsis, and George Balousros. 2016. Precision-Guided Context Sensitivity for Point-to Analysis. *ACM Trans. Program. Lang. Syst.* December 2016, 42 pages. <https://doi.org/10.1145/3000000.3000001>

Introspective Analysis: Context-Sensitivity, Across the Board
YUE LI, Aarhus University, Denmark
ANDERS MØLLER, Aarhus University, Denmark
YANNIS SMAKRGADAKIS, Department of Informatics, University of Crete, Greece

Abstract Context-sensitivity is the primary approach for adding more precision without paying a heavy price. However, existing context-sensitivity approaches are often too conservative to achieve one that leads to reasonable analysis time and obtain high precision. In this paper, we introduce BRAS, a general approach improving the precision of object-sensitivity pointer analysis for data-driven program analysis as well as a learning algorithm to find the model parameters based on various pointer features and therefore is able to express nonlinear combined programs. Key techniques is to employ a learning-based approach to find the model parameters that would simply be impractical. We present a stepwise and greedy algorithm that handles the learning of the model parameters and the analysis of the learned model. The proposed context-sensitive pointer-to analysis for Java and flow-sensitive interval analysis for C. Experimental results demonstrate that our automated technique significantly improves the performance of the state-of-the-art pointer-to analysis.

CSCS Concepts: Theory of computation → Program analysis; Computing method → machine learning approaches.

Additional Key Words and Phrases: Data-driven program analysis; Static analysis; Context sensitivity

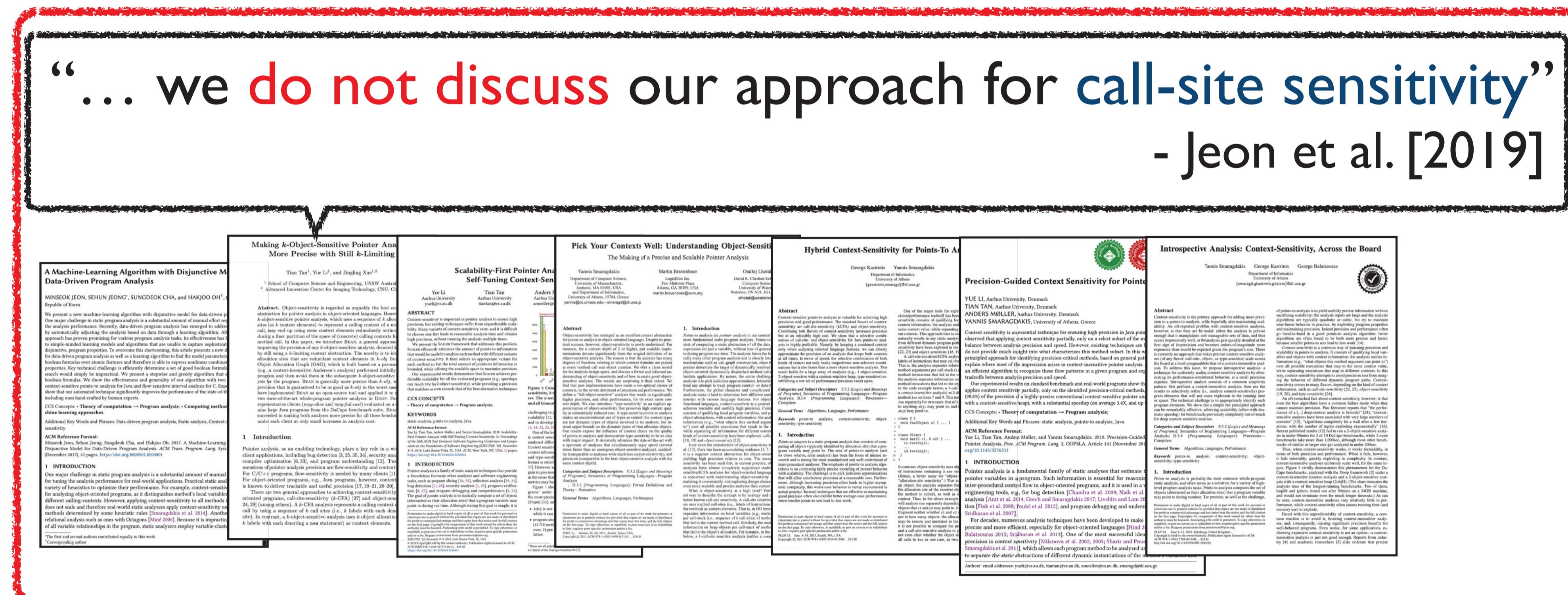
ACM Reference Format:
YUE LI, Anders Møller, and Yannis Smaragdakis. 2016. Introspective Analysis: Context-Sensitivity, Across the Board. *ACM Trans. Program. Lang. Syst.* December 2016, 42 pages. <https://doi.org/10.1145/3000000.3000001>



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]



1981

2002

2010

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



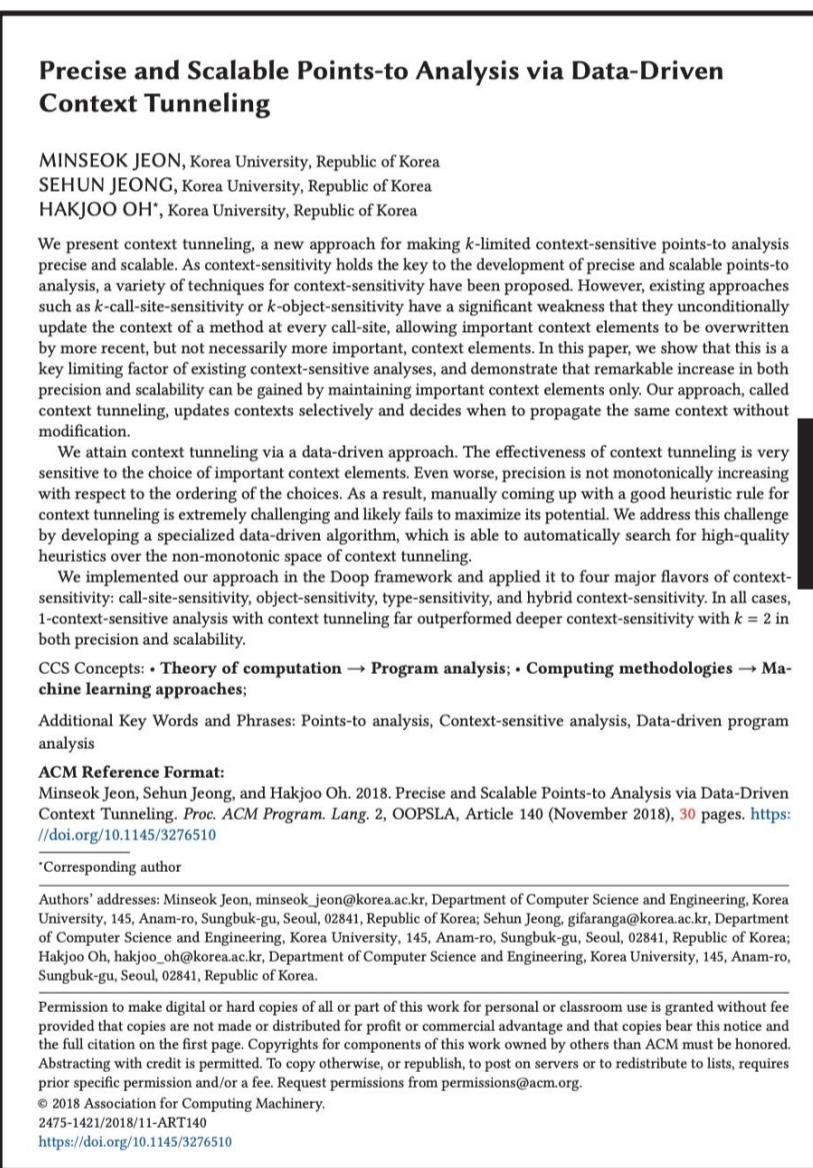
Call-site Sensitivity vs Object Sensitivity

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

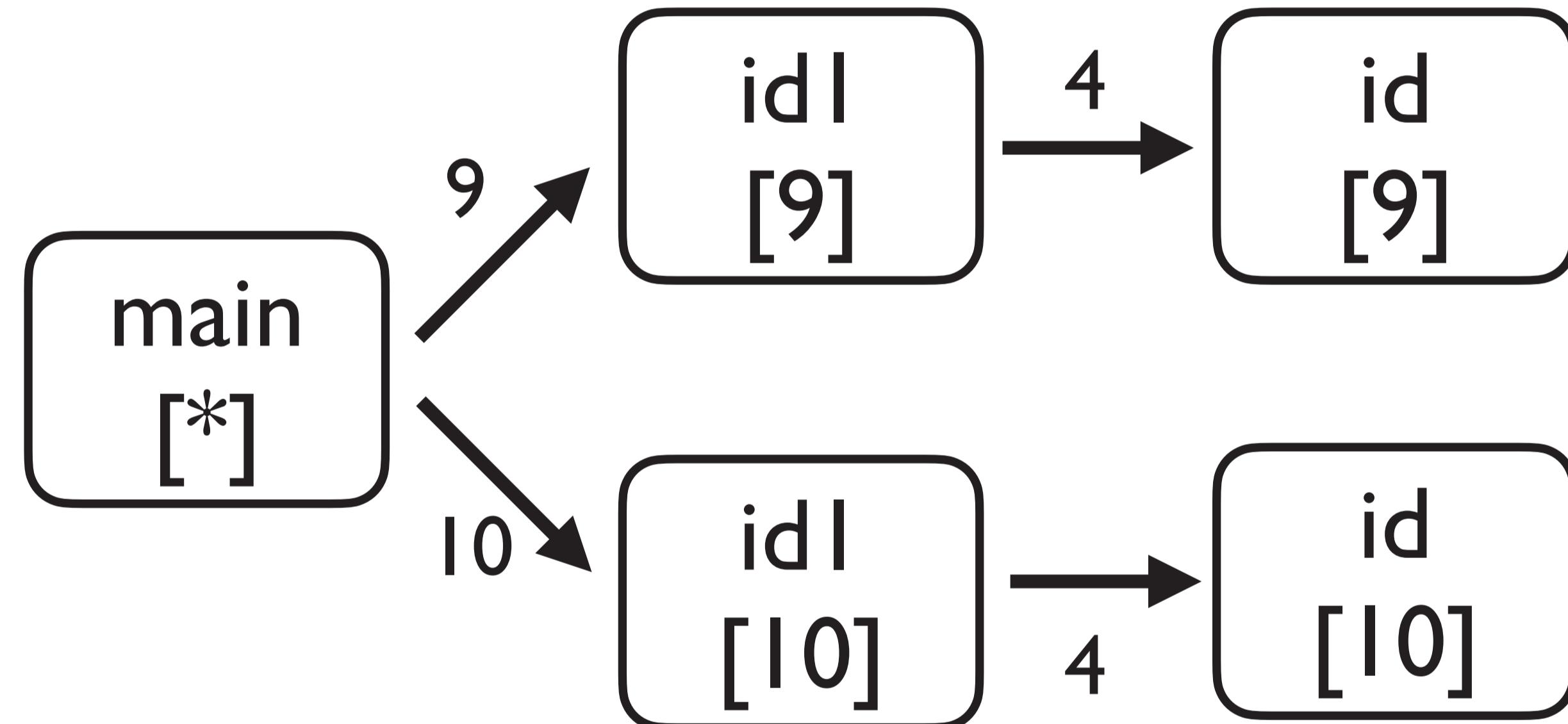
Jeon et al. [2018]



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();//C1  
8:   c2 = new C();//C2  
9:   a = (A) c1.id1(new A());//query1  
10:  b = (B) c2.id1(new B());//query2  
11: }
```

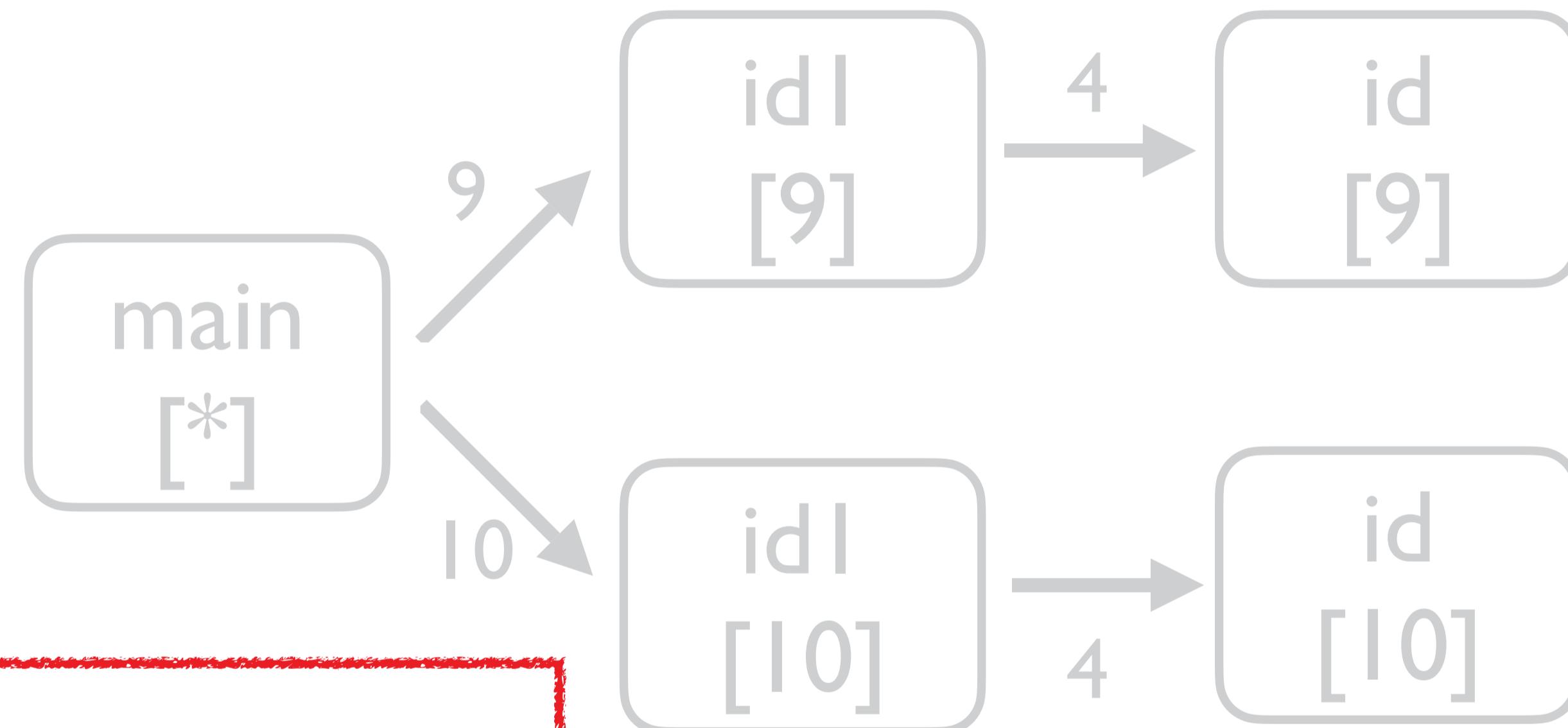


I-CFA with context tunneling
(T= {4})

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

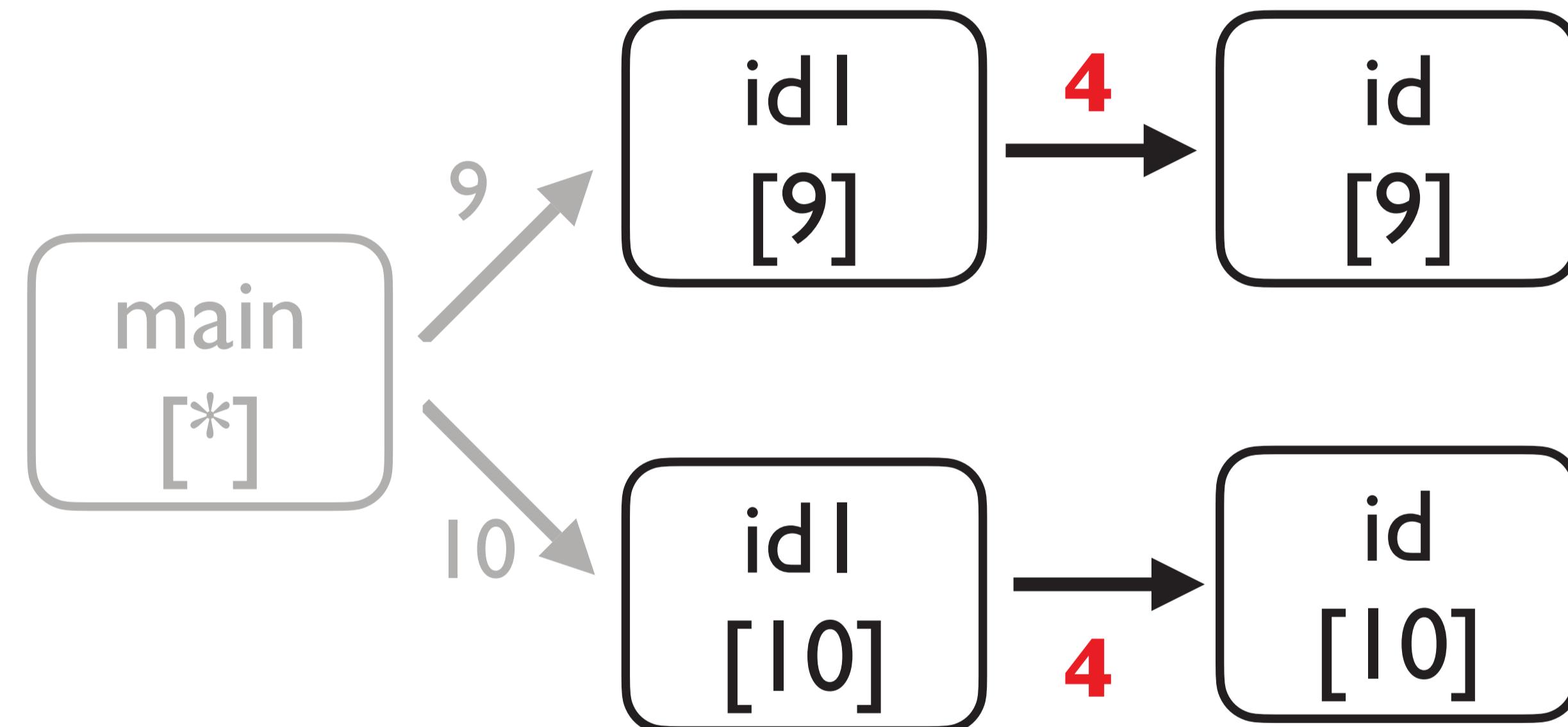


Tunneling abstraction:
Determines where to apply context tunneling
with context tunneling ($T = \{4\}$)

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  
}
```



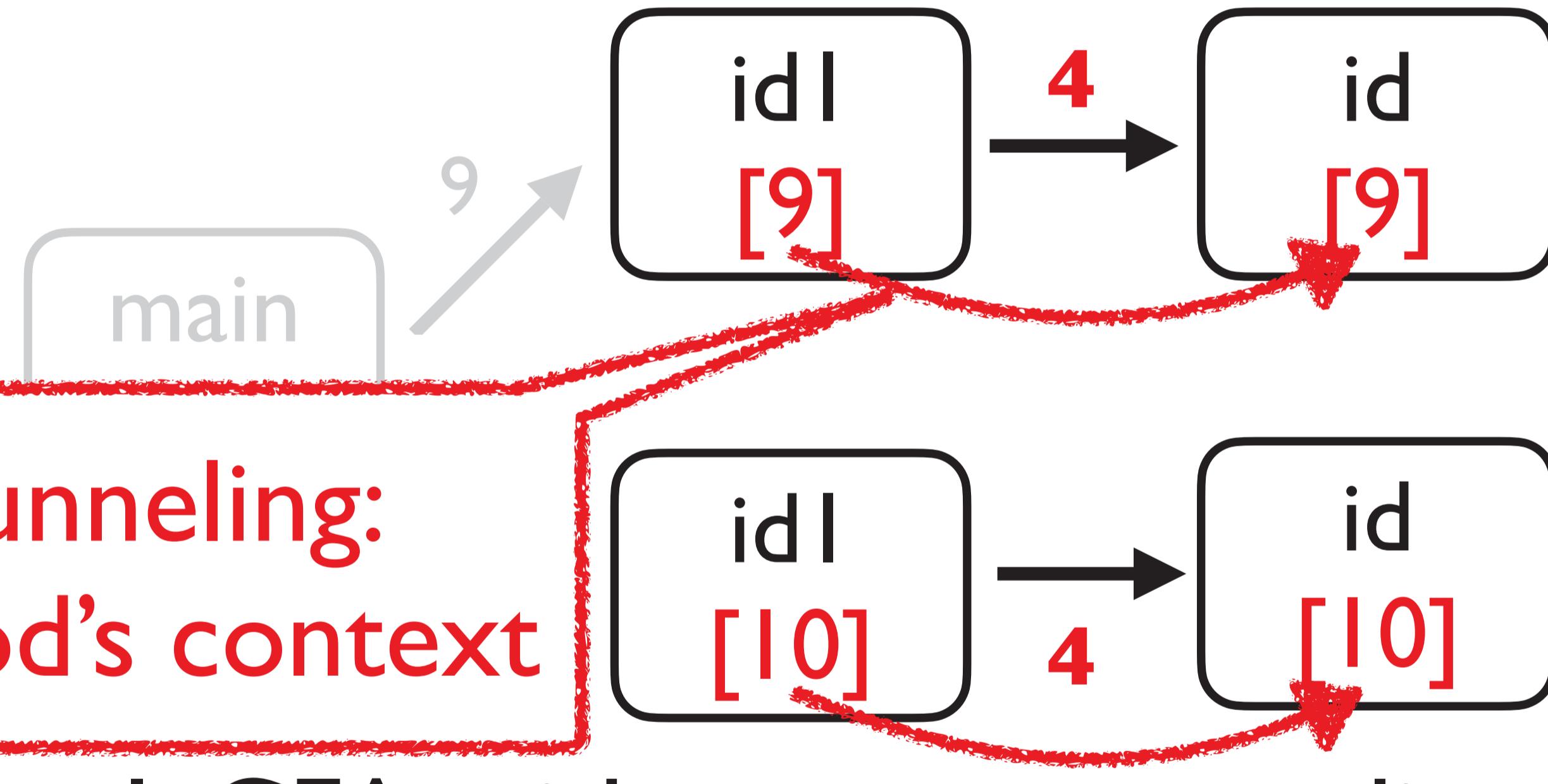
I-CFA with context tunneling
(T= {4})

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

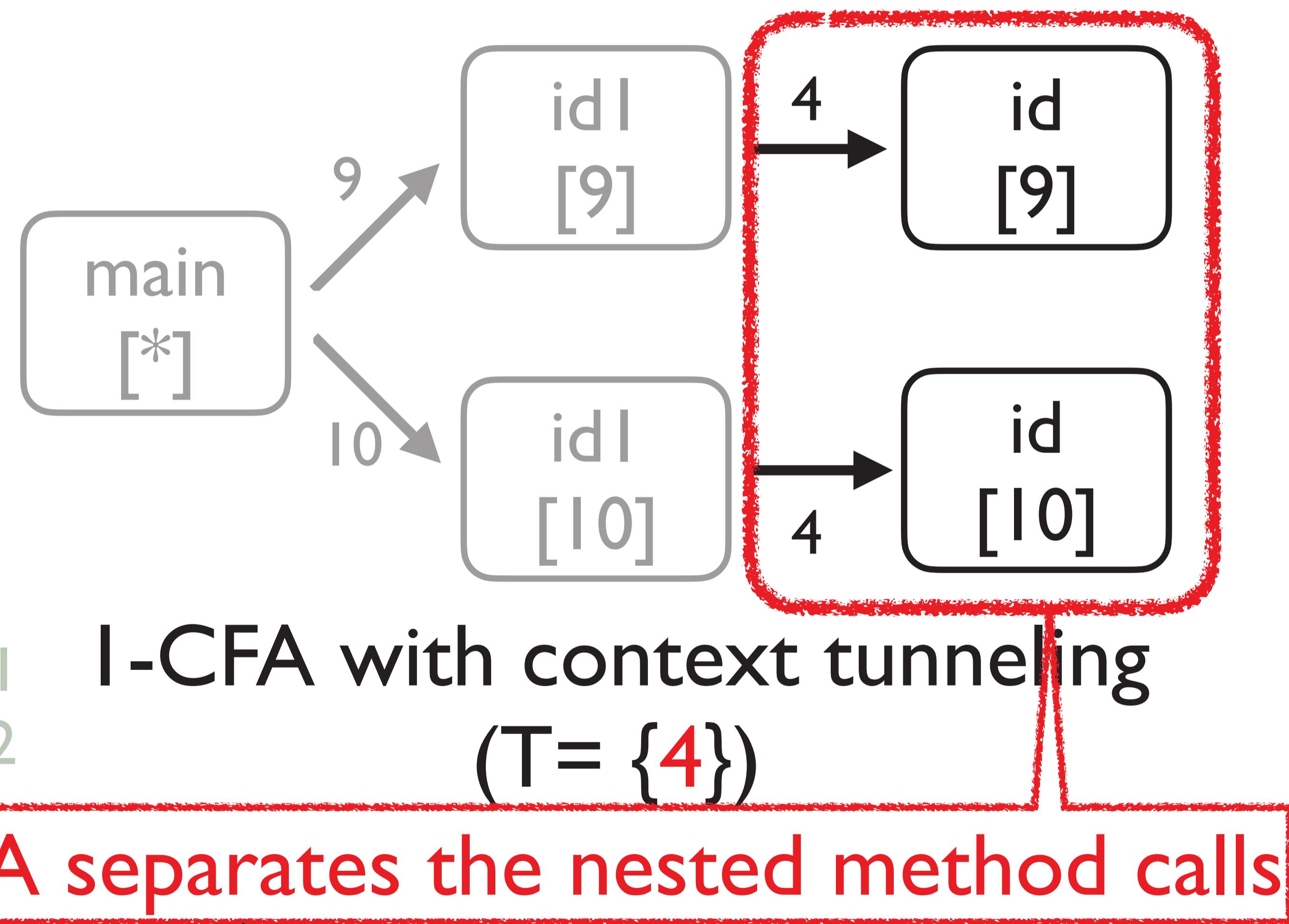
```
0: class C{  
1:   id(v){  
2:     return v;}  
3:   id1(v){  
4:     return id0(v);}  
5: }  
6: main()  
7: c1 = new C();  
8: c2 = new C();  
9: a = (A) c1.id1(new A());//query1  
10: b = (B) c2.id1(new B());//query2  
11: }
```



Call-site Sensitivity vs Object Sensitivity

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

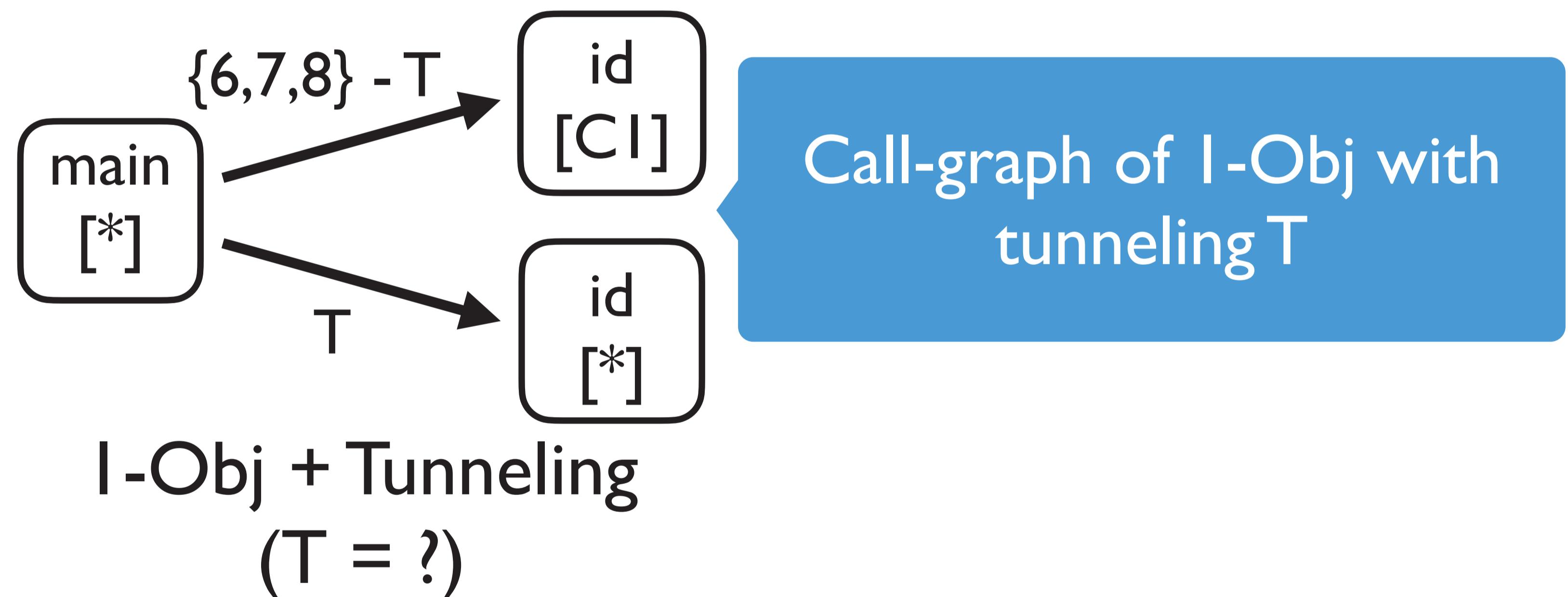
```
0: class C{  
1:   id(v){  
2:     return v;  
3:   id1(v){  
4:     return id0(v);  
5:   }  
6: main(){  
7:   c1 = new C();//C1  
8:   c2 = new C();//C2  
9:   a = (A) c1.id1(new A());//query1  
10:  b = (B) c2.id1(new B());//query2  
11: }
```



Call-site Sensitivity vs Object Sensitivity

- Object sensitivity still suffers from its limitation

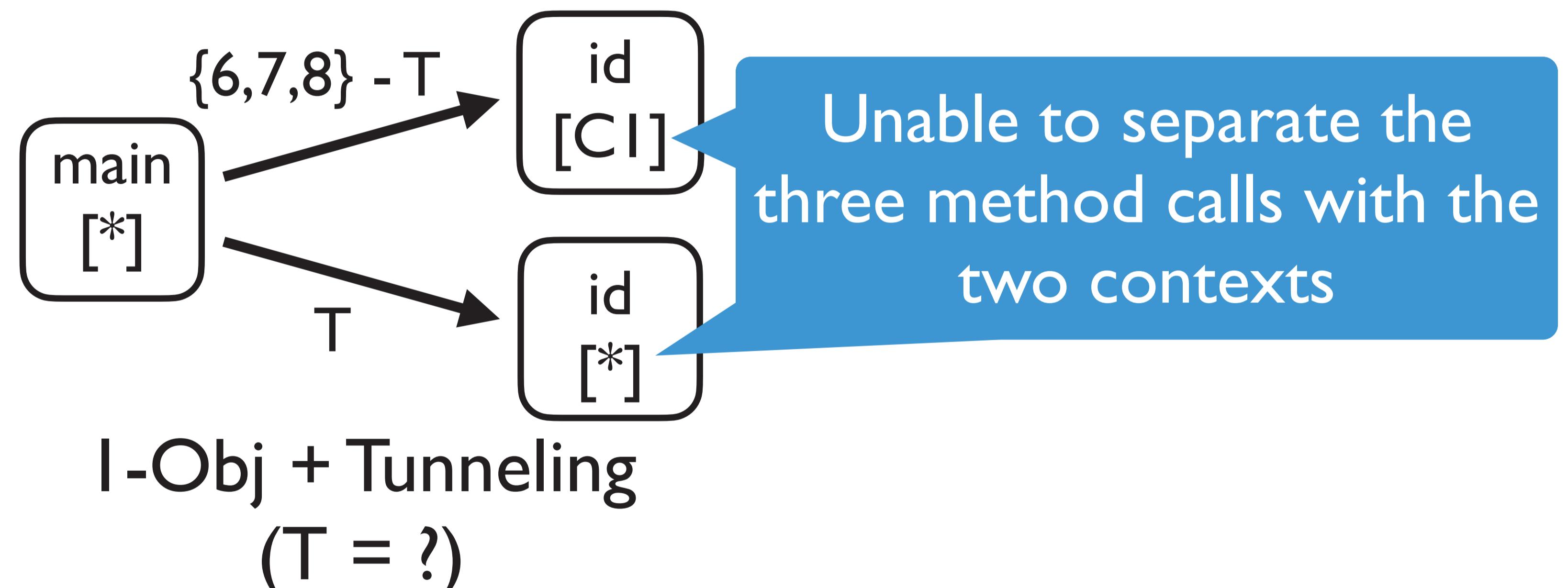
```
0: class C{  
1:   id(v){  
2:     return v;  
3:   }  
4: main(){  
5:   cl = new C(); //CI  
6:   a = (A) cl.id(new A());  
7:   b = (B) cl.id(new B());  
8:   c = (C) cl.id(new C());  
9: }
```



Call-site Sensitivity vs Object Sensitivity

- Object sensitivity still suffers from its limitation

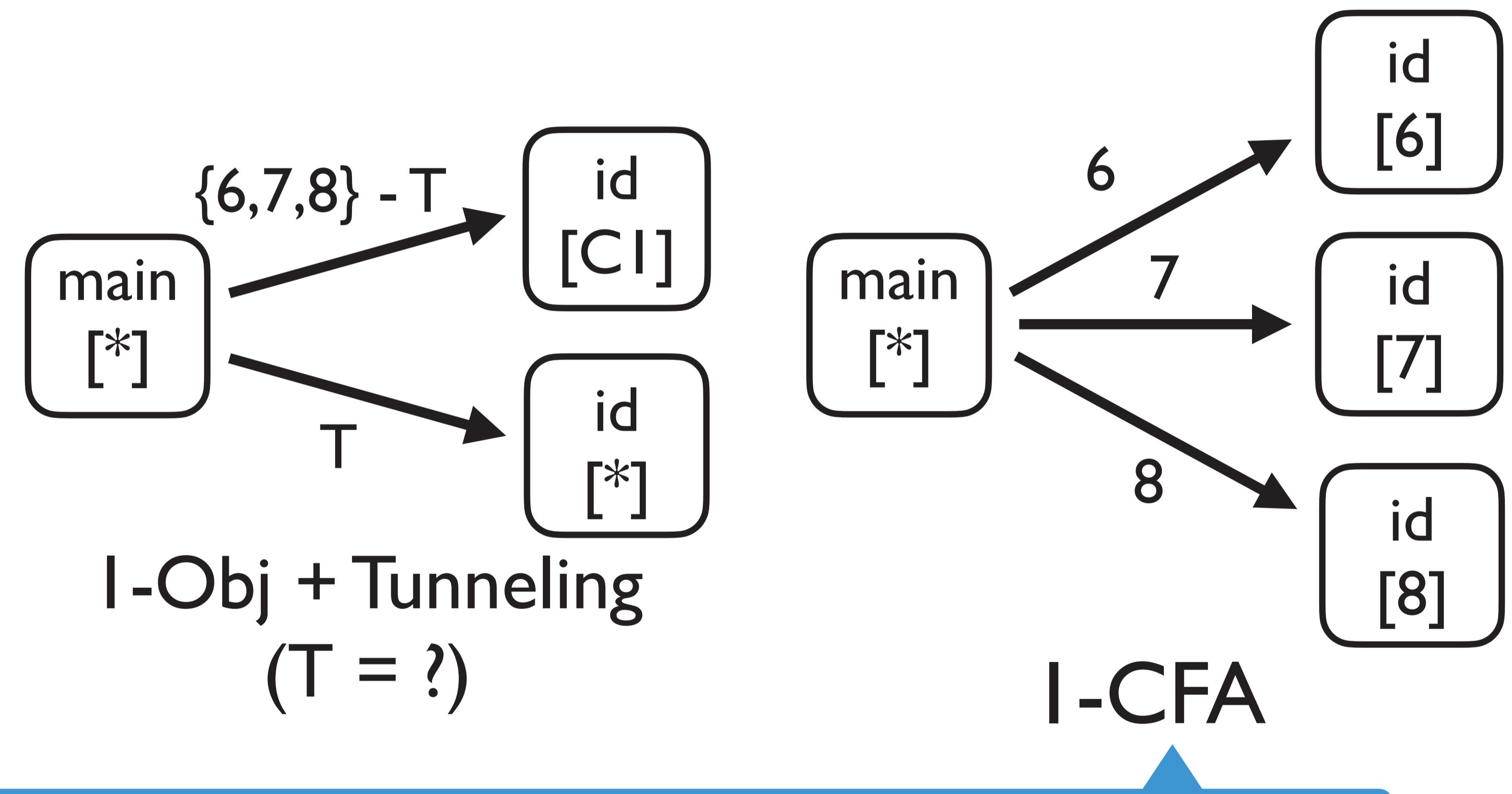
```
0: class C{  
1:   id(v){  
2:     return v;  
3: }  
4: main(){  
5:   cl = new C(); //CI  
6:   a = (A) cl.id(new A());  
7:   b = (B) cl.id(new B());  
8:   c = (C) cl.id(new C());  
9: }
```



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(); //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: r
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: When context tunneling is included
2: 
3: }
4: m
5: 
6: a
7: b
8: c
9: }
```

When context tunneling is included

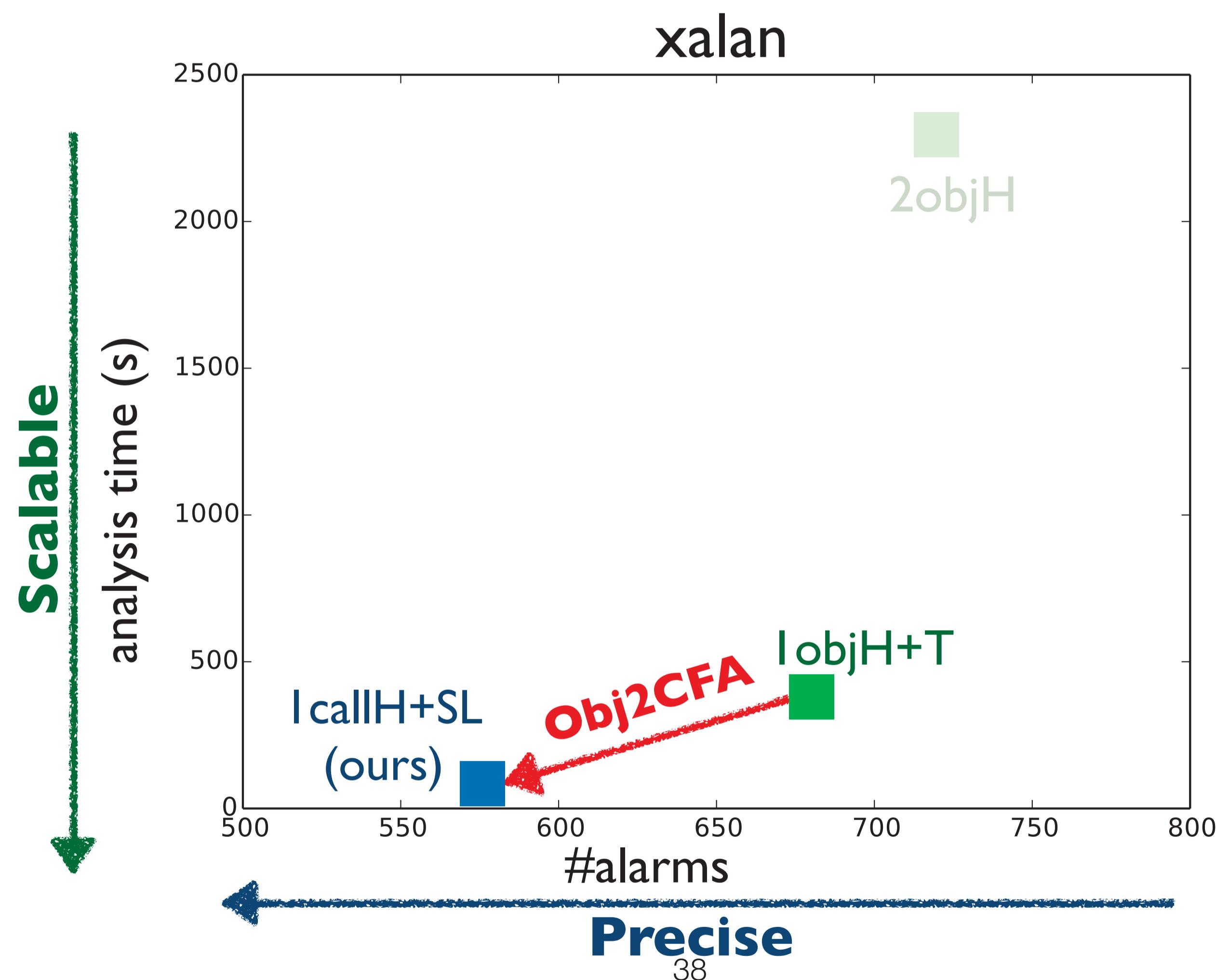
- Limitation of call-site sensitivity is **removed**
- Limitation of object sensitivity is **not removed**

Our claim

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

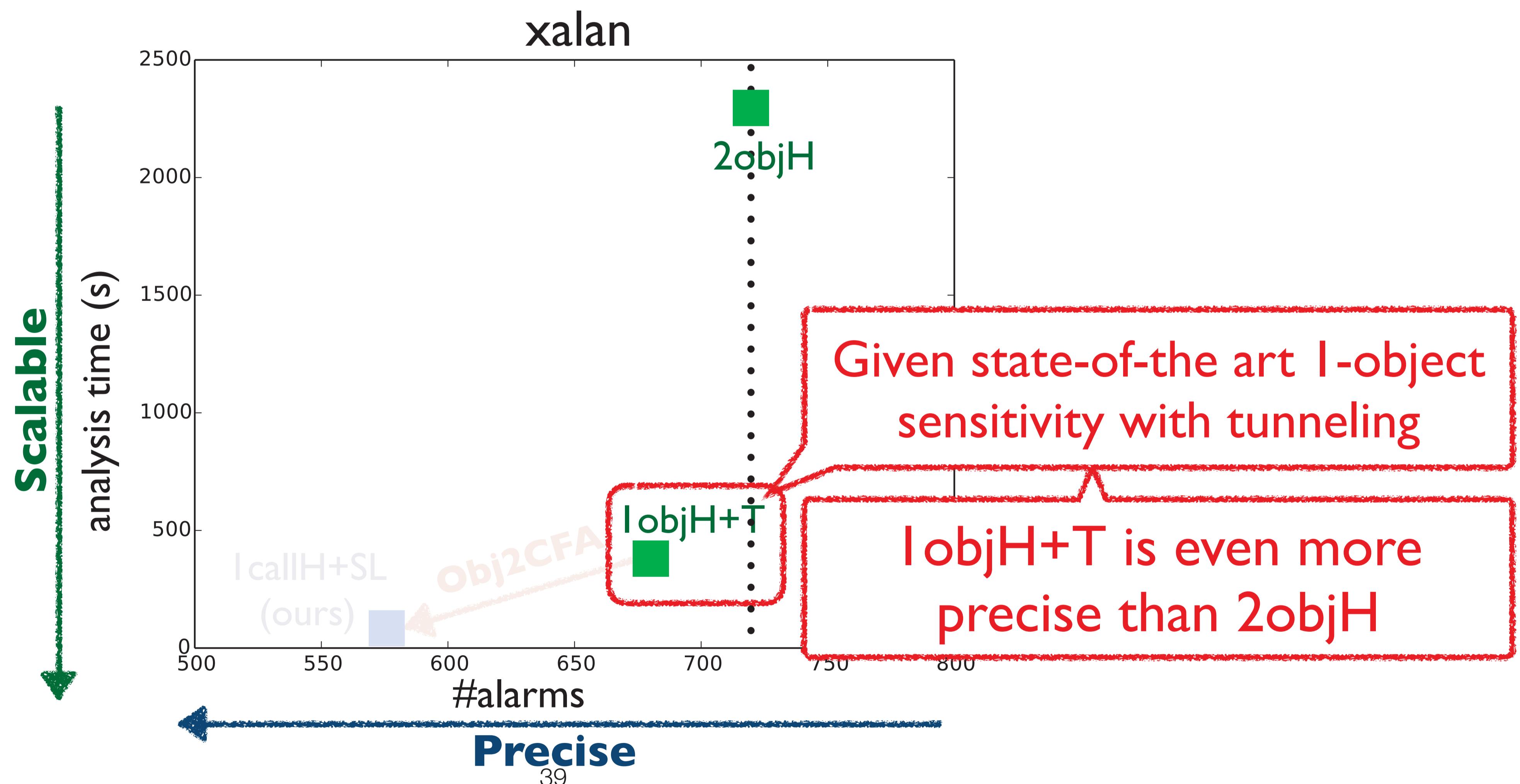
Our Technique : **Obj2CFA**

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



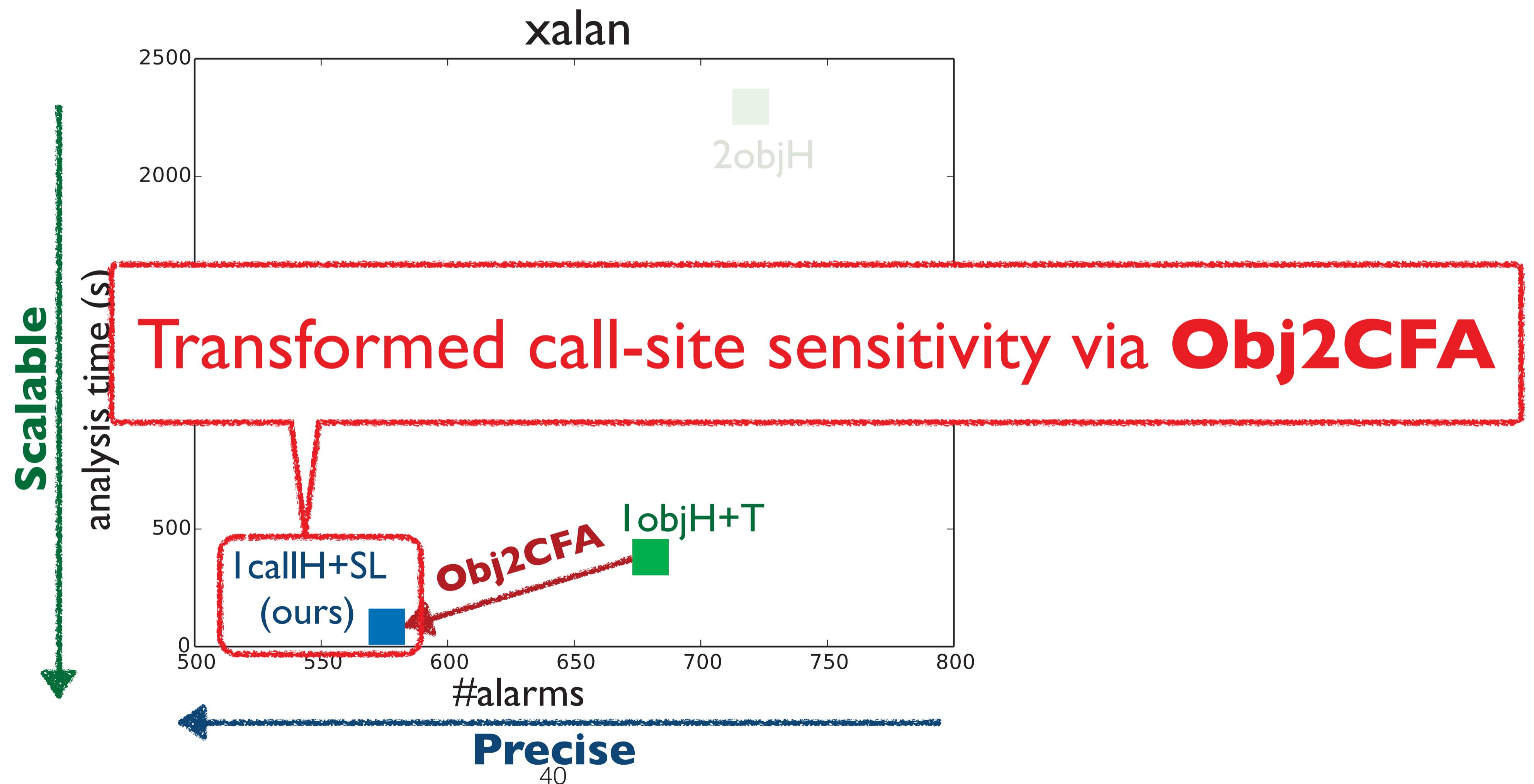
Our Technique : **Obj2CFA**

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



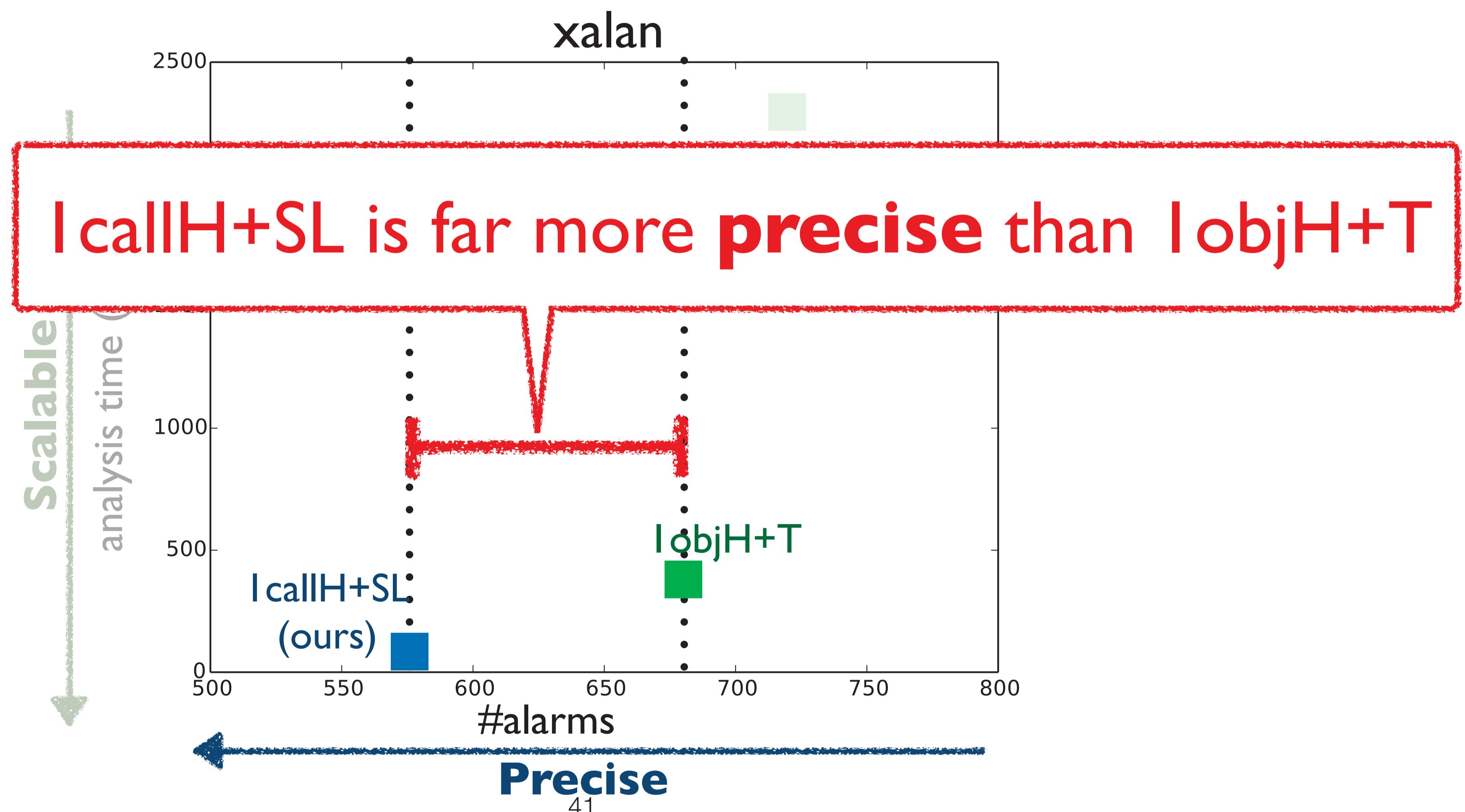
Our Technique : **Obj2CFA**

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



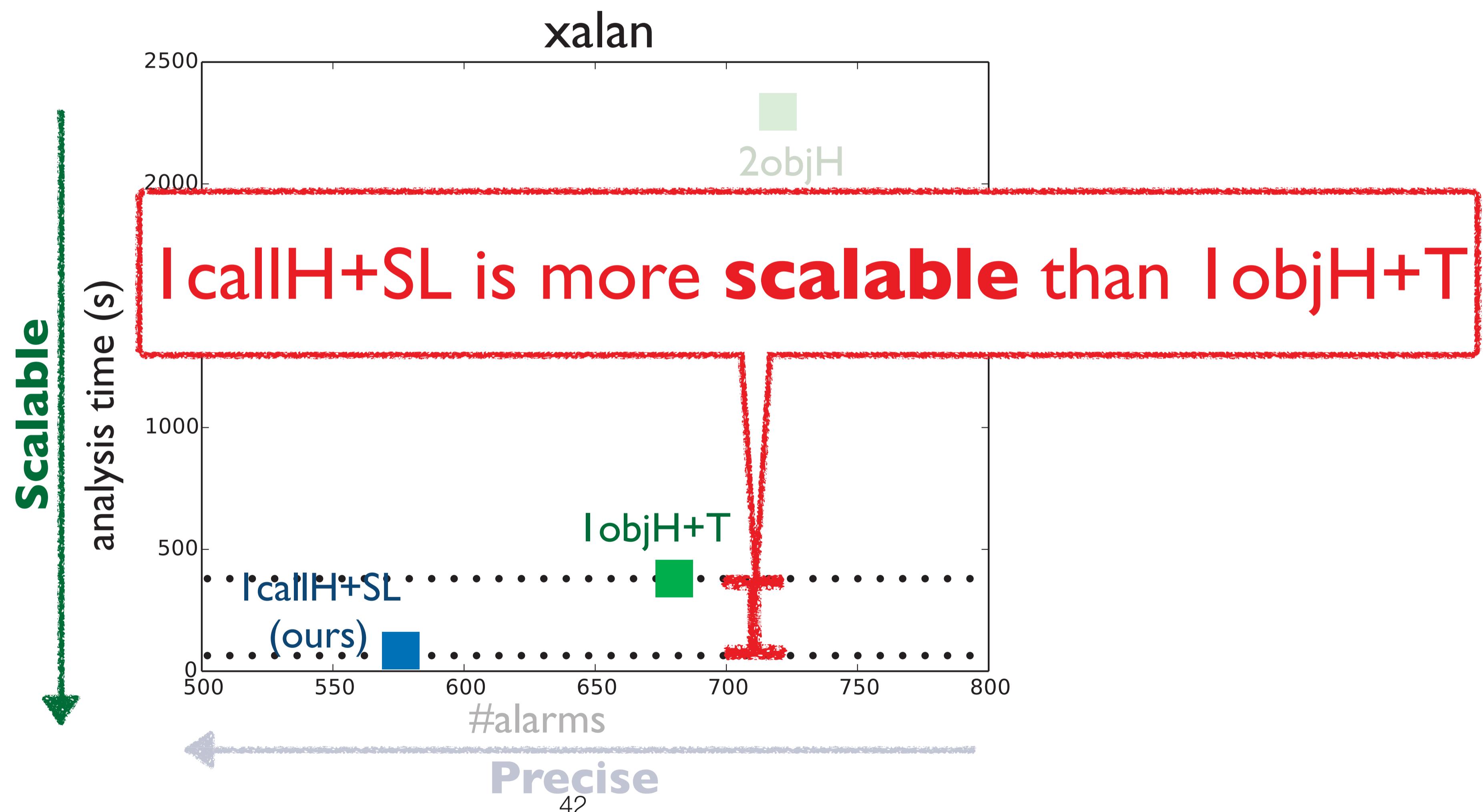
Our Technique : **Obj2CFA**

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



Our Technique : **Obj2CFA**

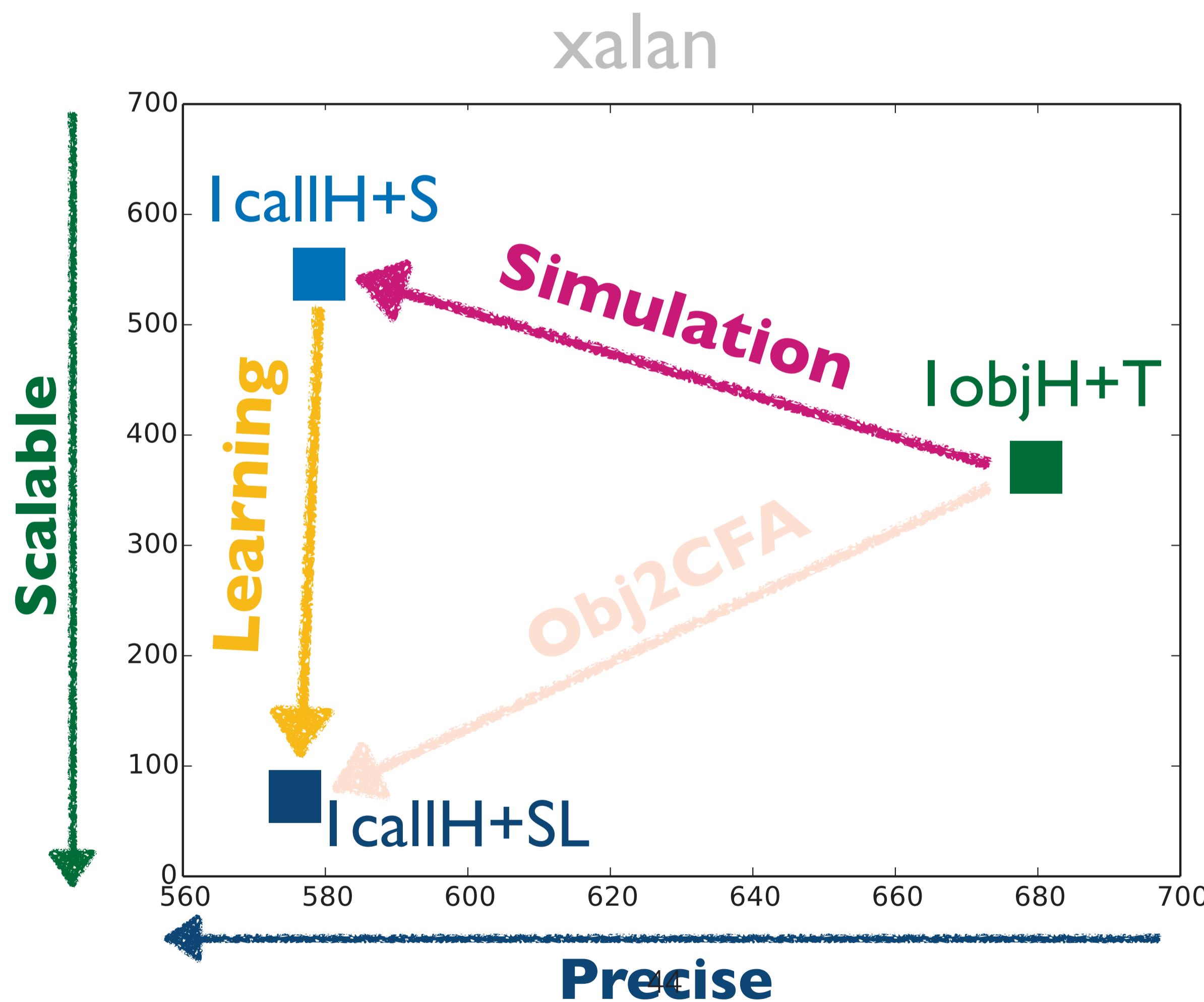
- **Obj2CFA** transforms a given **object sensitivity** into a more precise **CFA**



Detail of Obj2CFA

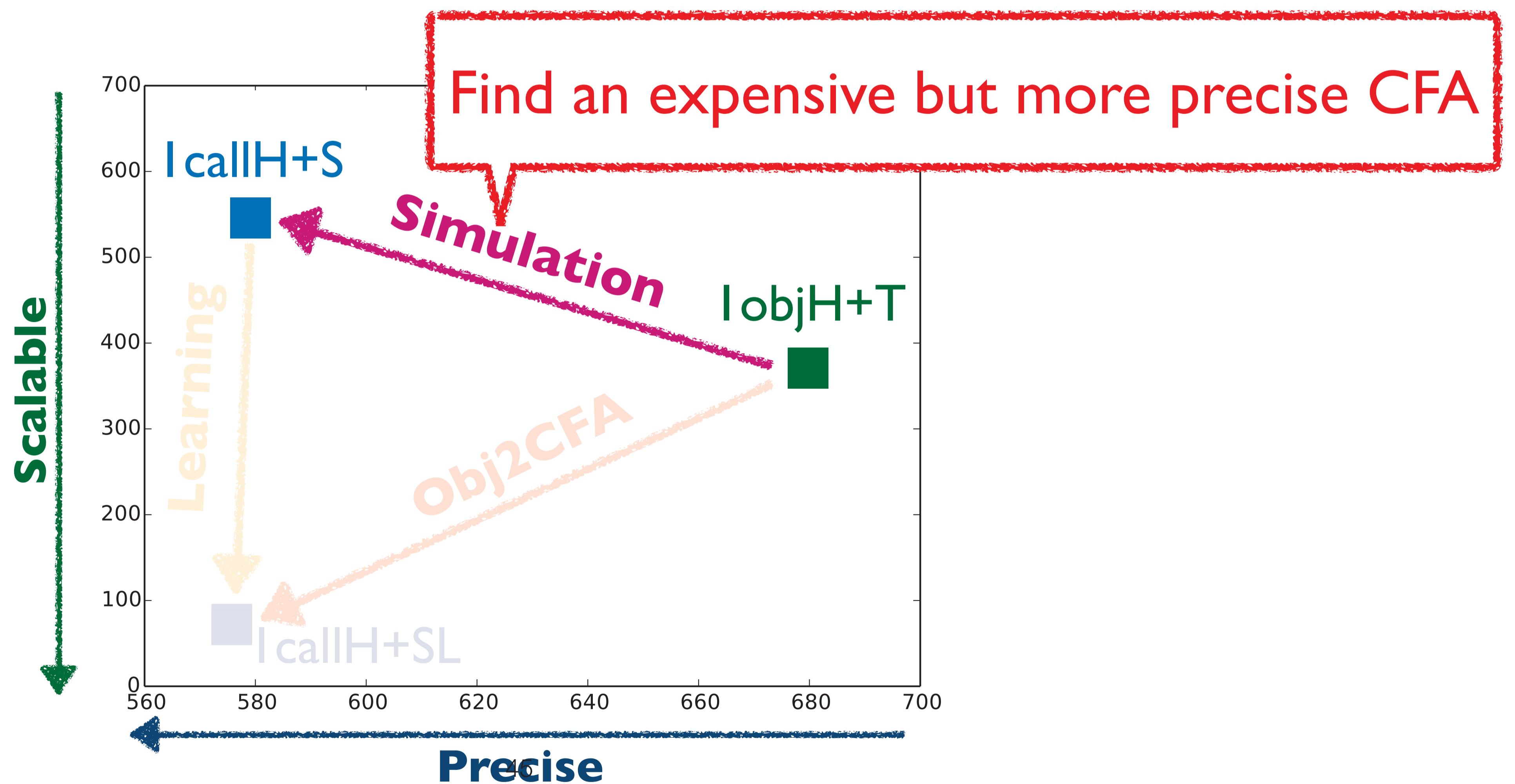
Our Technique : **Obj2CFA**

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



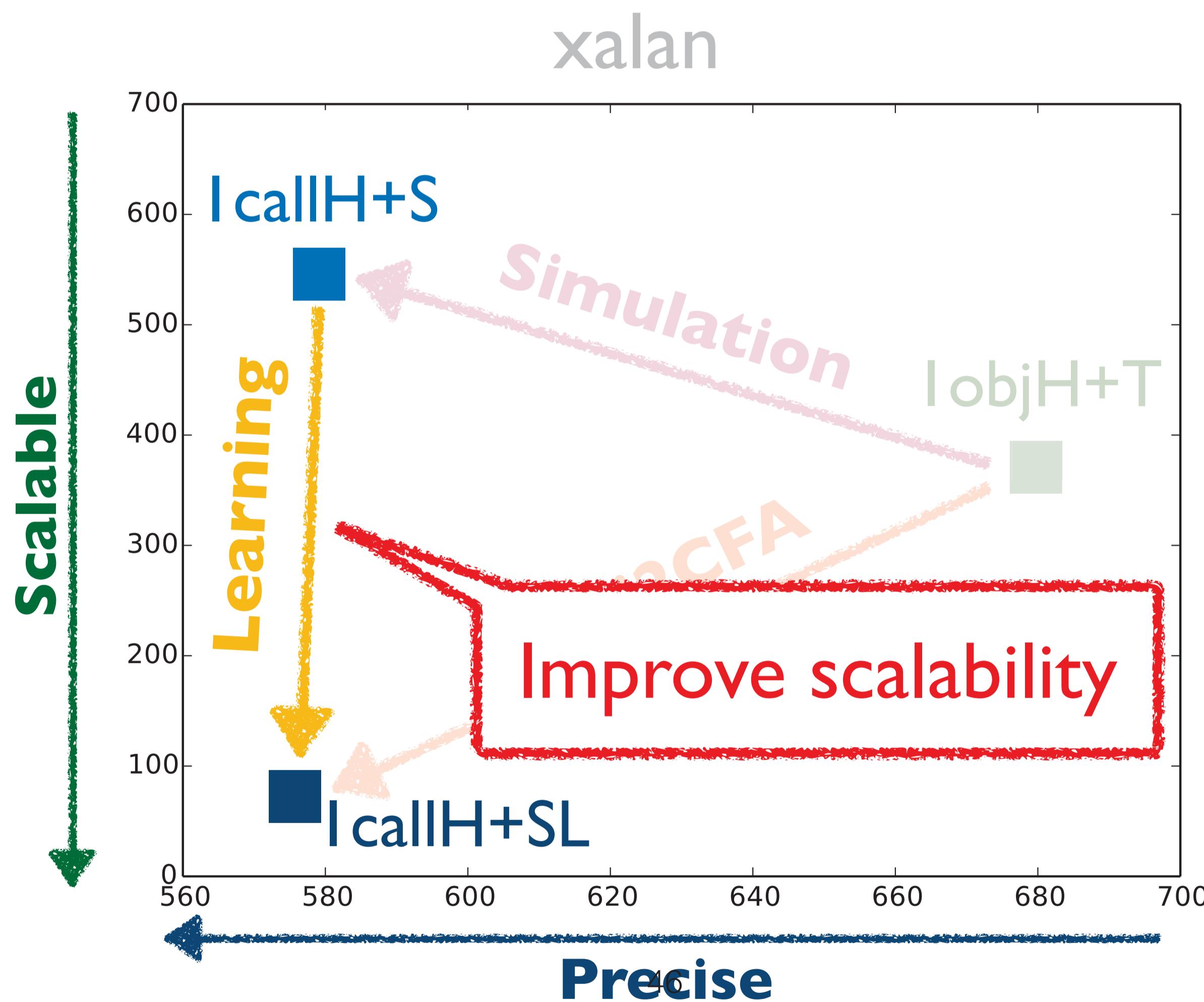
Our Technique : **Obj2CFA**

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



Our Technique : **Obj2CFA**

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



Technique I: Simulation

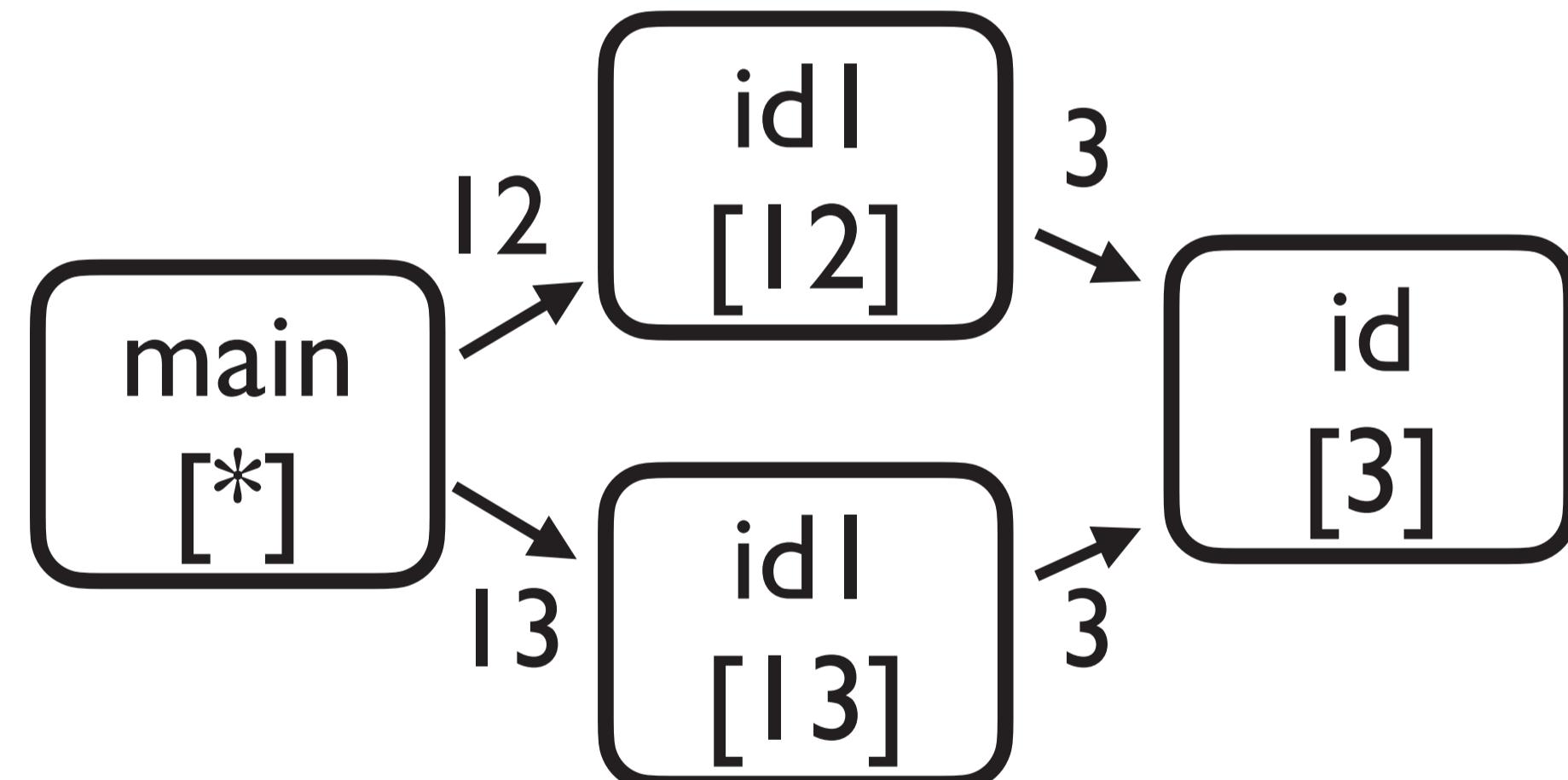
- Running example to illustrate Simulation

```
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

```
I: 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: }
```



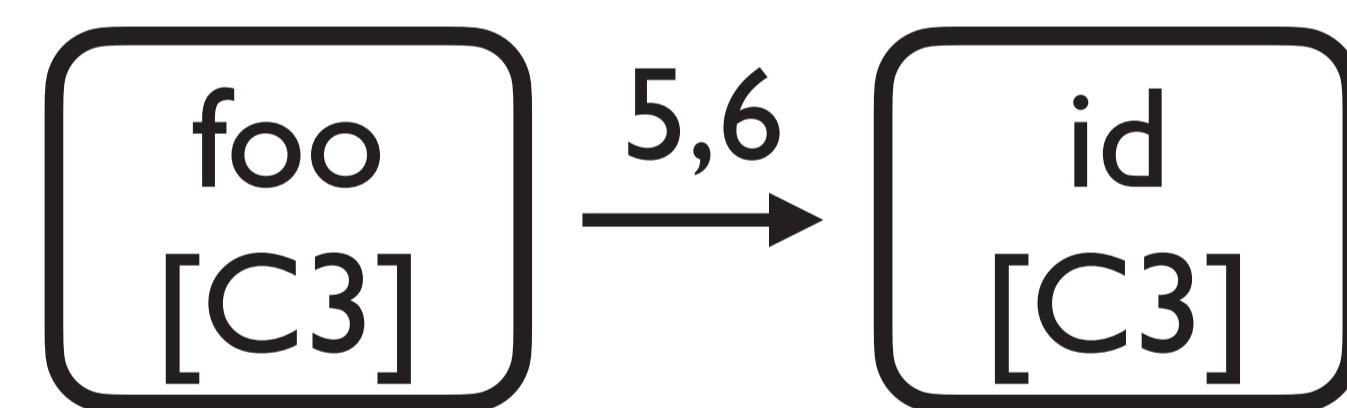
Technique I: Simulation

- 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: }
```



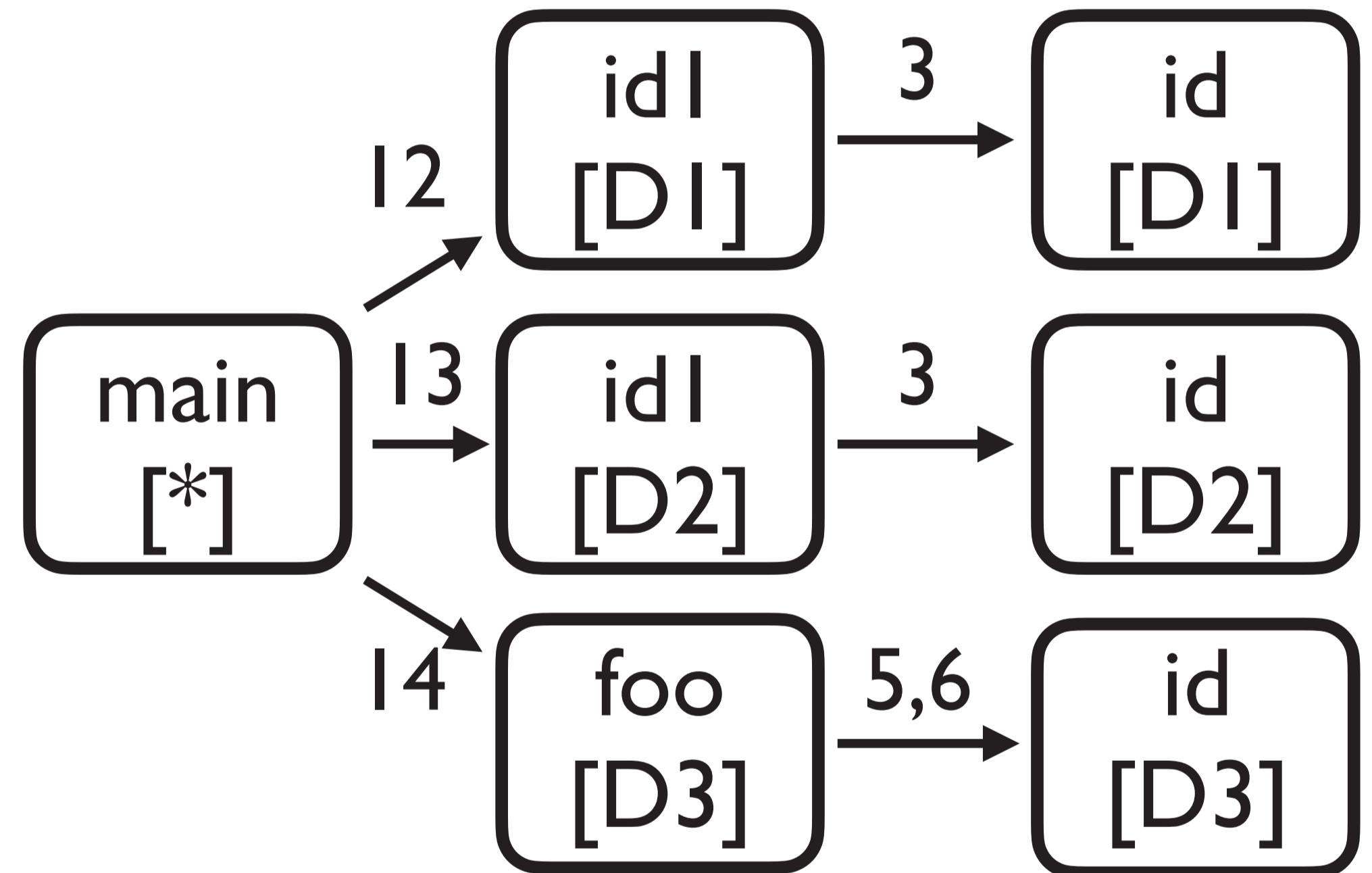
Limitation of object sensitivity



Technique I: Simulation

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

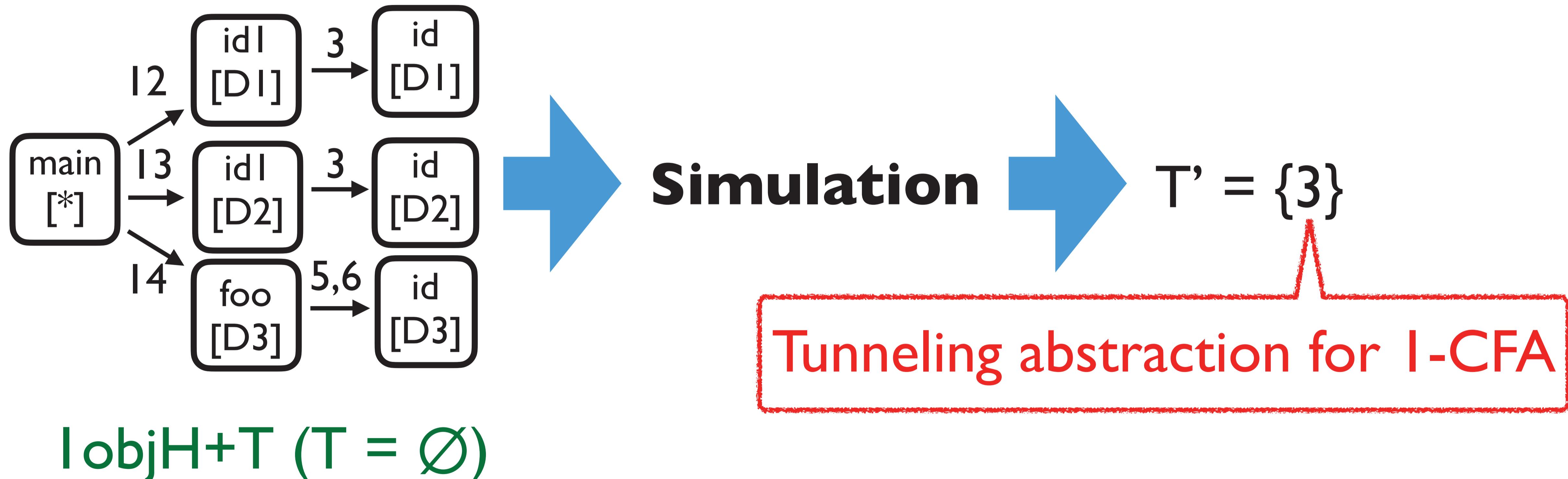
```
1: class C{  
2:   id(v){return v;}  
3:   idI(v){return id(v);}  
4:   foo(){  
5:     A a = (A) this.id(new A());}//query1  
6:     B b = (B) this.id(new B());}//query2  
7: }  
8: main(){  
9:   c1 = new C();//C1  
10:  c2 = new C();//C2  
11:  c3 = new C();//C3  
12:  A a = (A) c1.idI(new A());//query3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



lobjH+T ($T = \emptyset$)

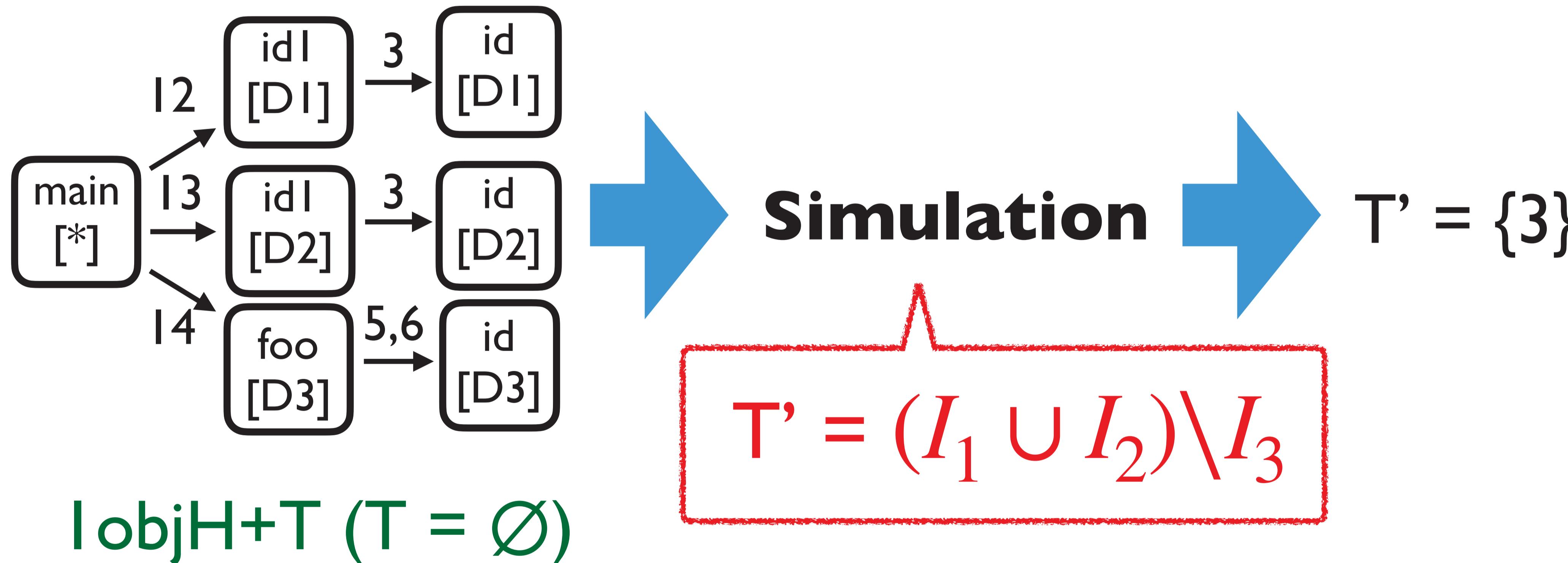
Technique I: Simulation

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



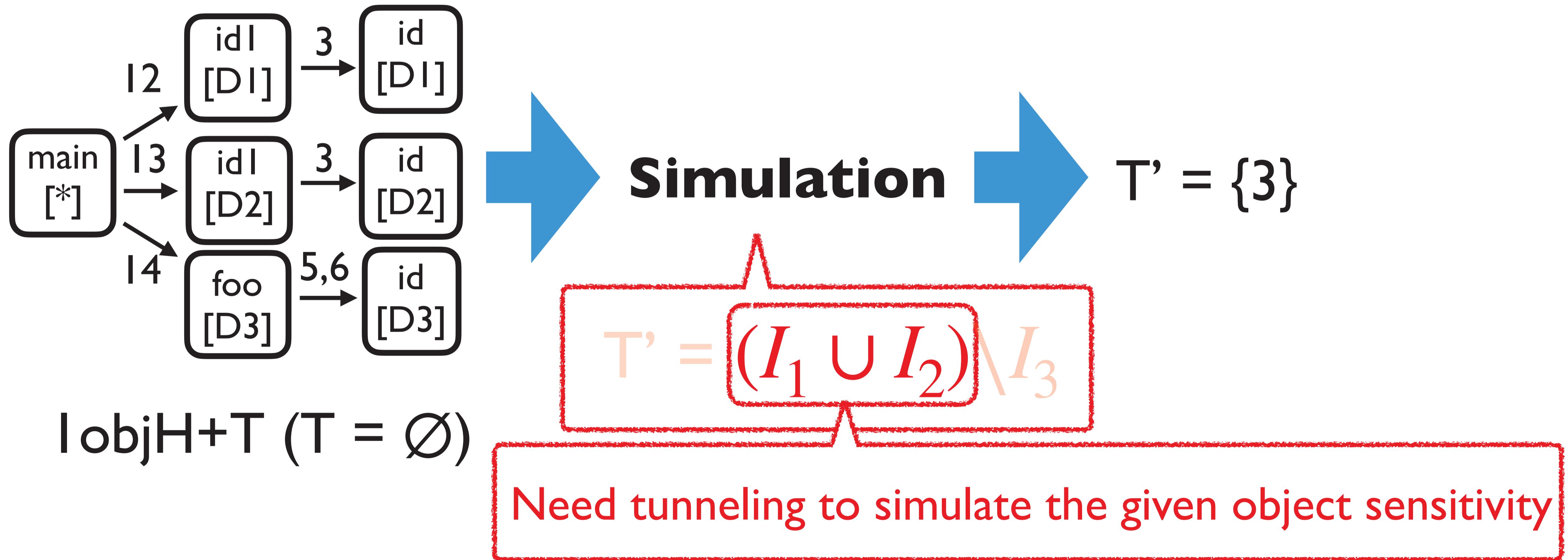
Technique I: Simulation

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



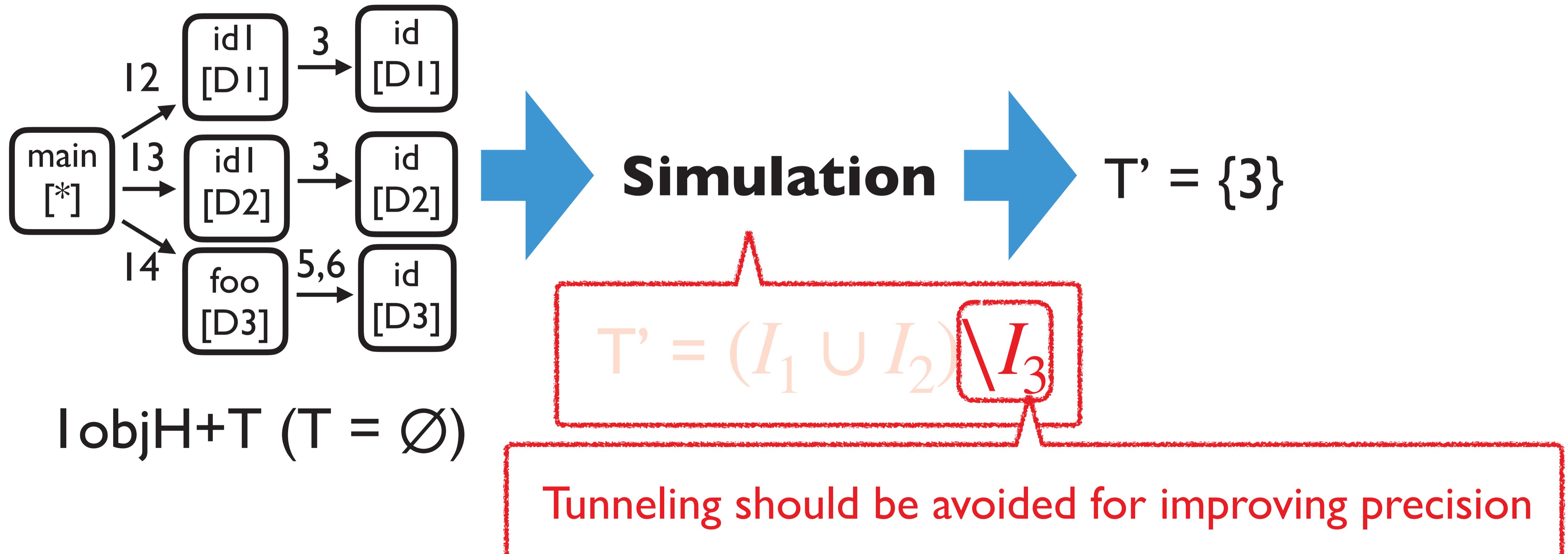
Technique I: Simulation

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



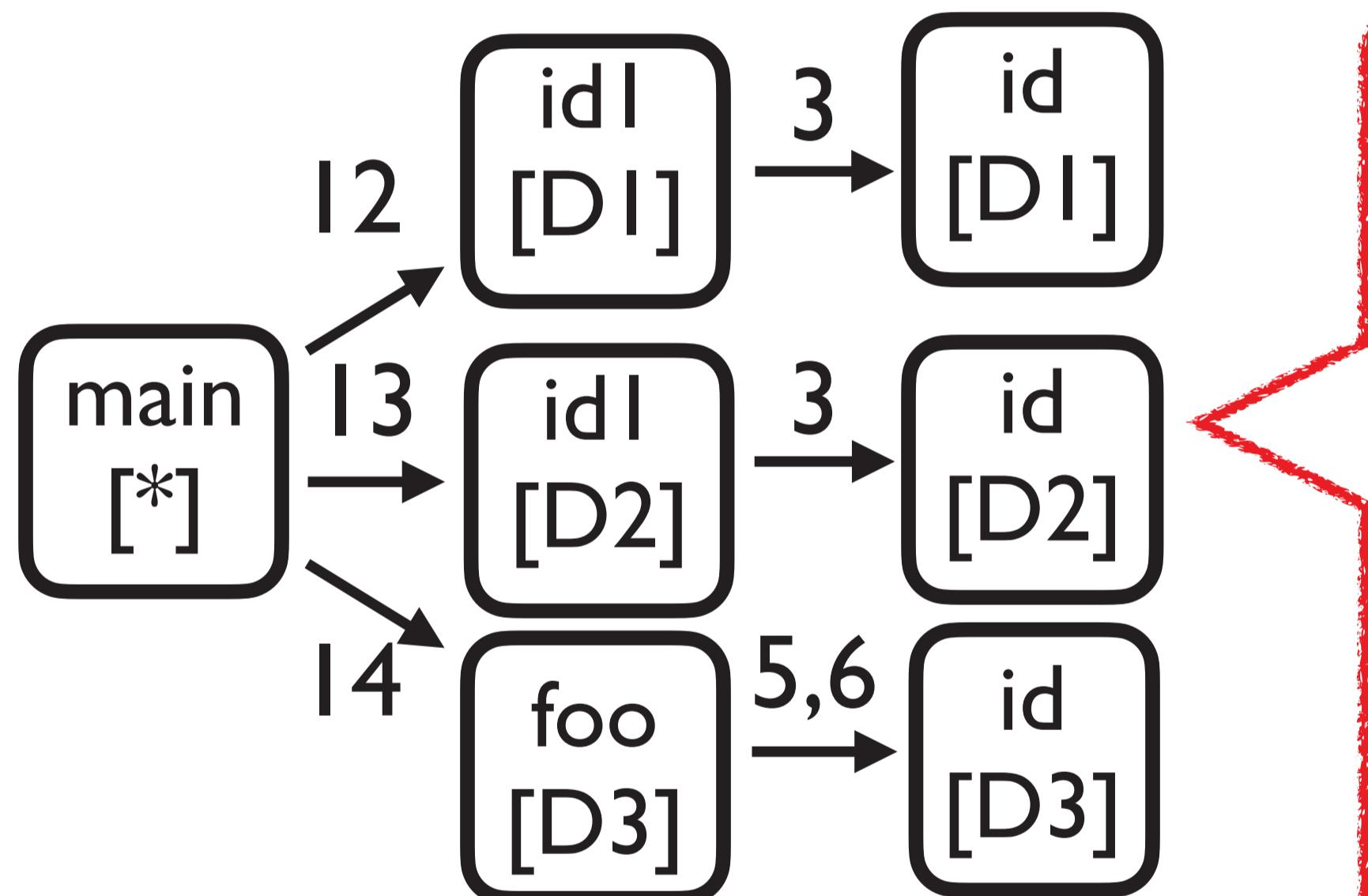
Technique I: Simulation

- **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'

~~IobjH+T (T = Ø)~~

IcallH+T'

What is T'?

Intuition Behind Simulation ($I_1 \cup I_2$)

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

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



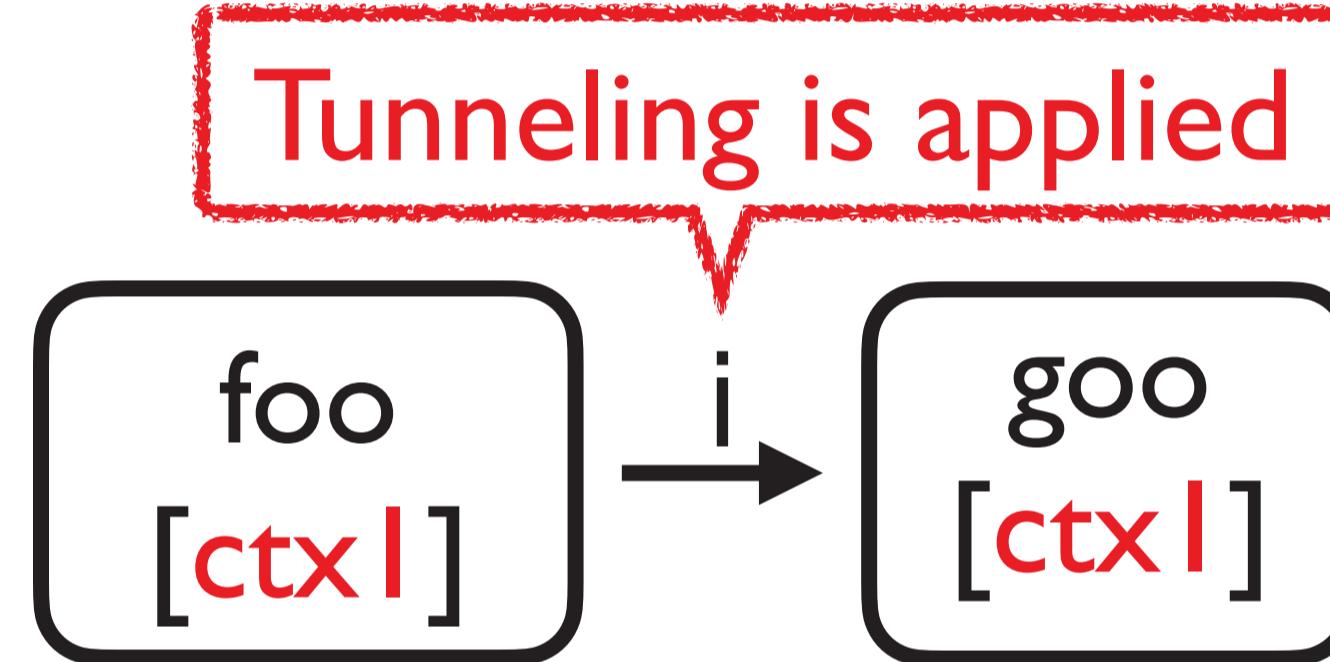
We track the two properties to find the T'

main
[*]

Intuition Behind Simulation ($I_1 \cup I_2$)

- Suppose we have a call-site i in I_1 that tunnels context $\text{ctx } I$ to I_2 .

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



Property of context tunneled call-sites

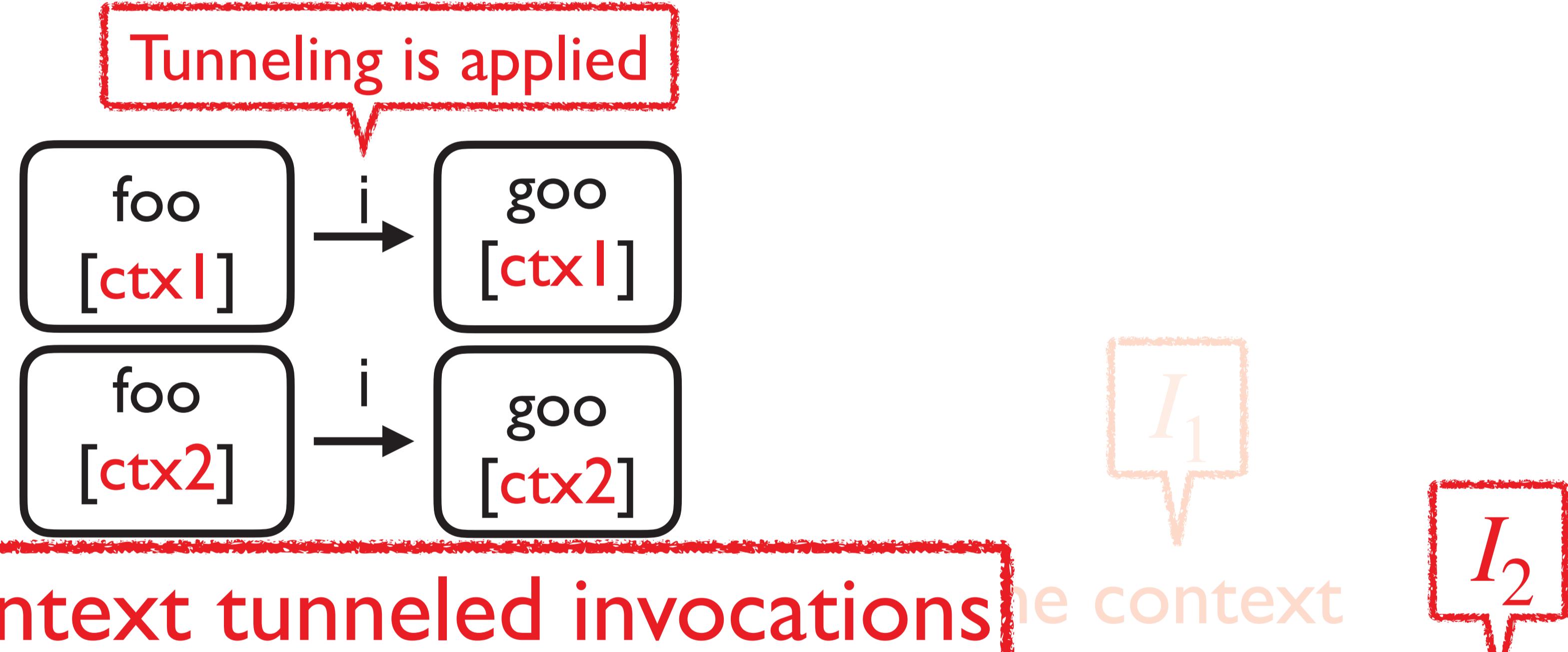
I_1

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

Intuition Behind Simulation ($I_1 \cup I_2$)

- Suppose we have two contexts I_1 and I_2 .
• If tunneling is applied to i , two properties inevitably appear at i

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

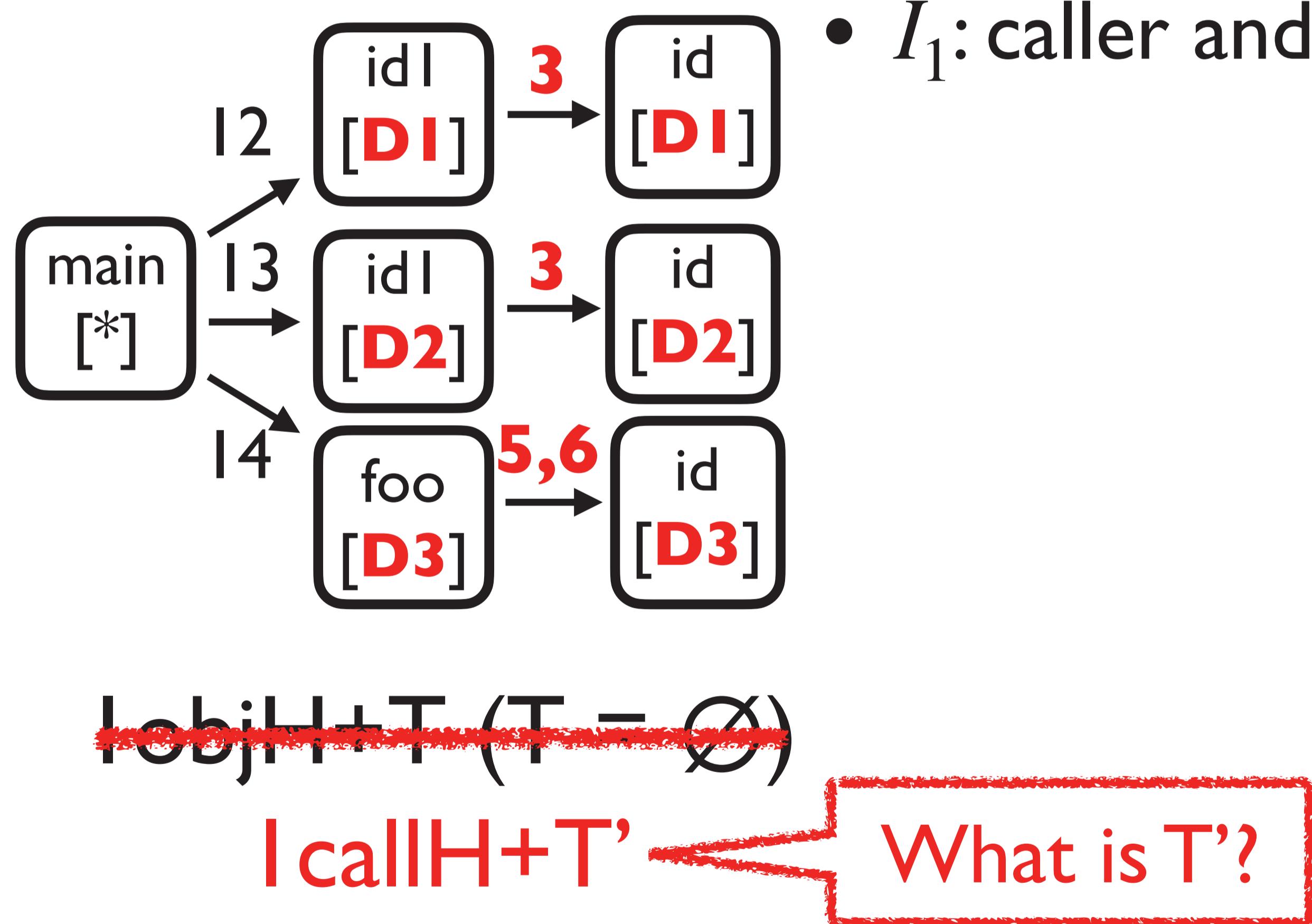


- Property of context tunneled invocations

- 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

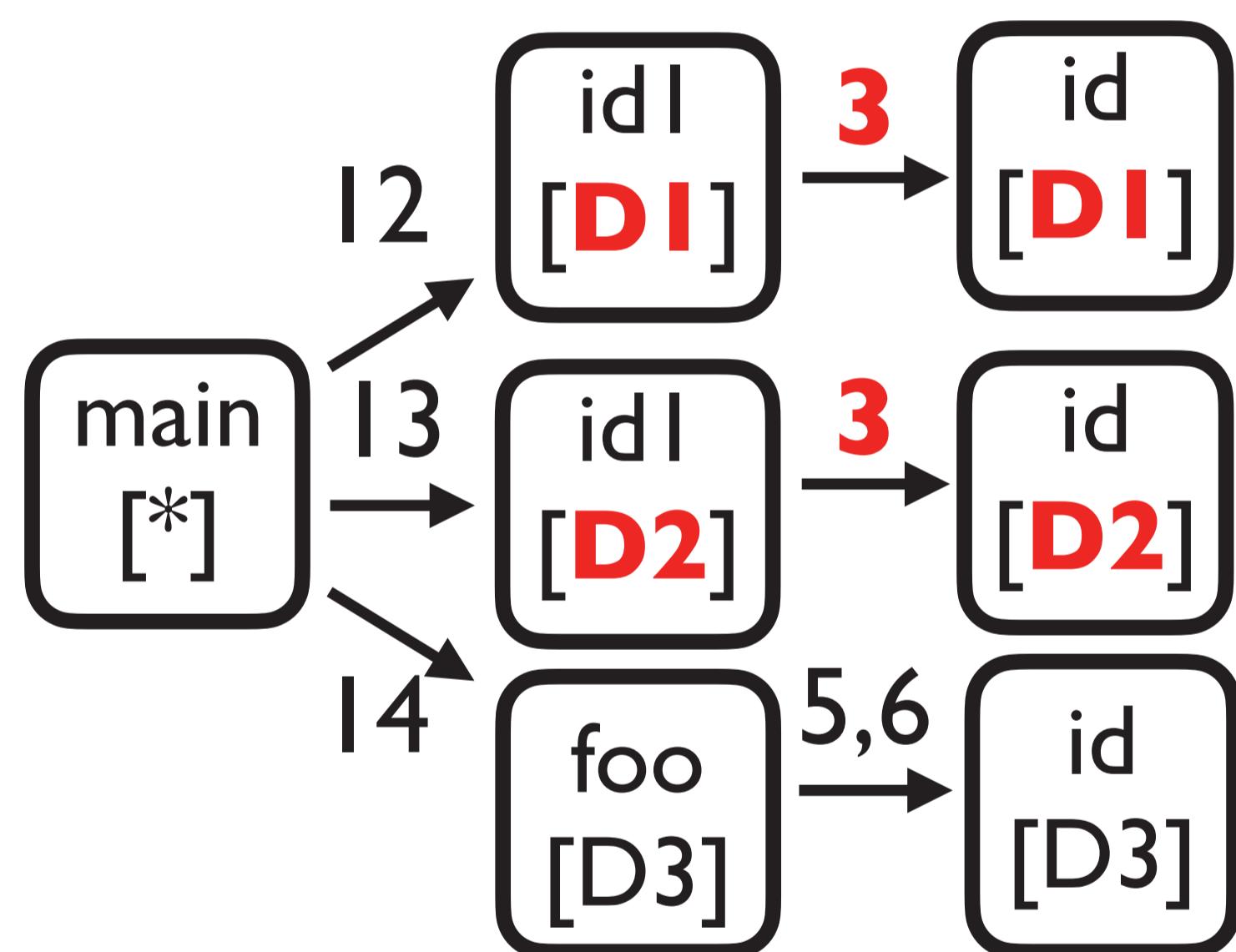


- I_1 : caller and callee methods have the **same context**

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

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
 $I_2 = \{3\}$

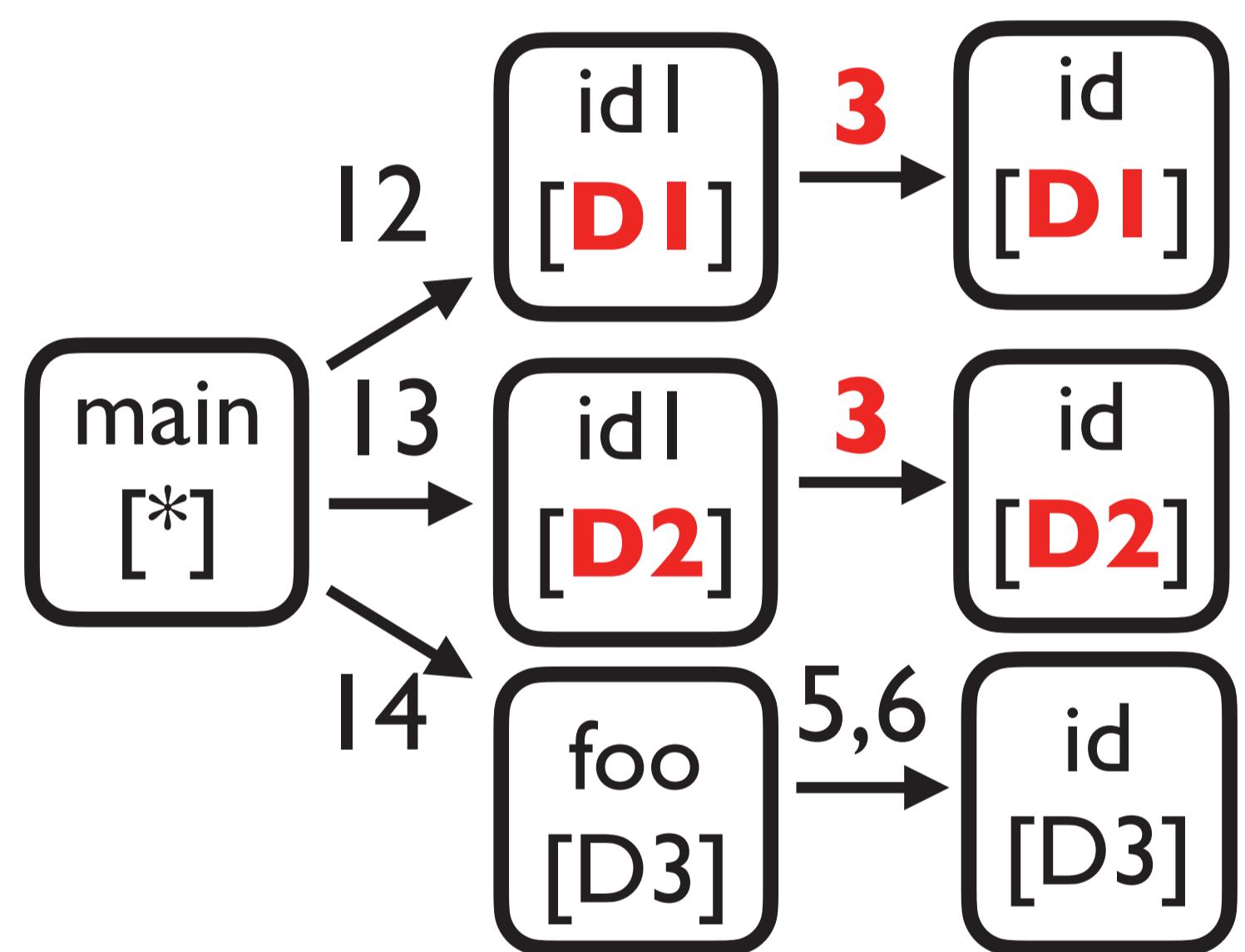
~~$I_{objH+T} (T = \emptyset)$~~

$I_{callH+T'}$

What is T' ?

Intuition Behind Simulation ($I_1 \cup I_2$)

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

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

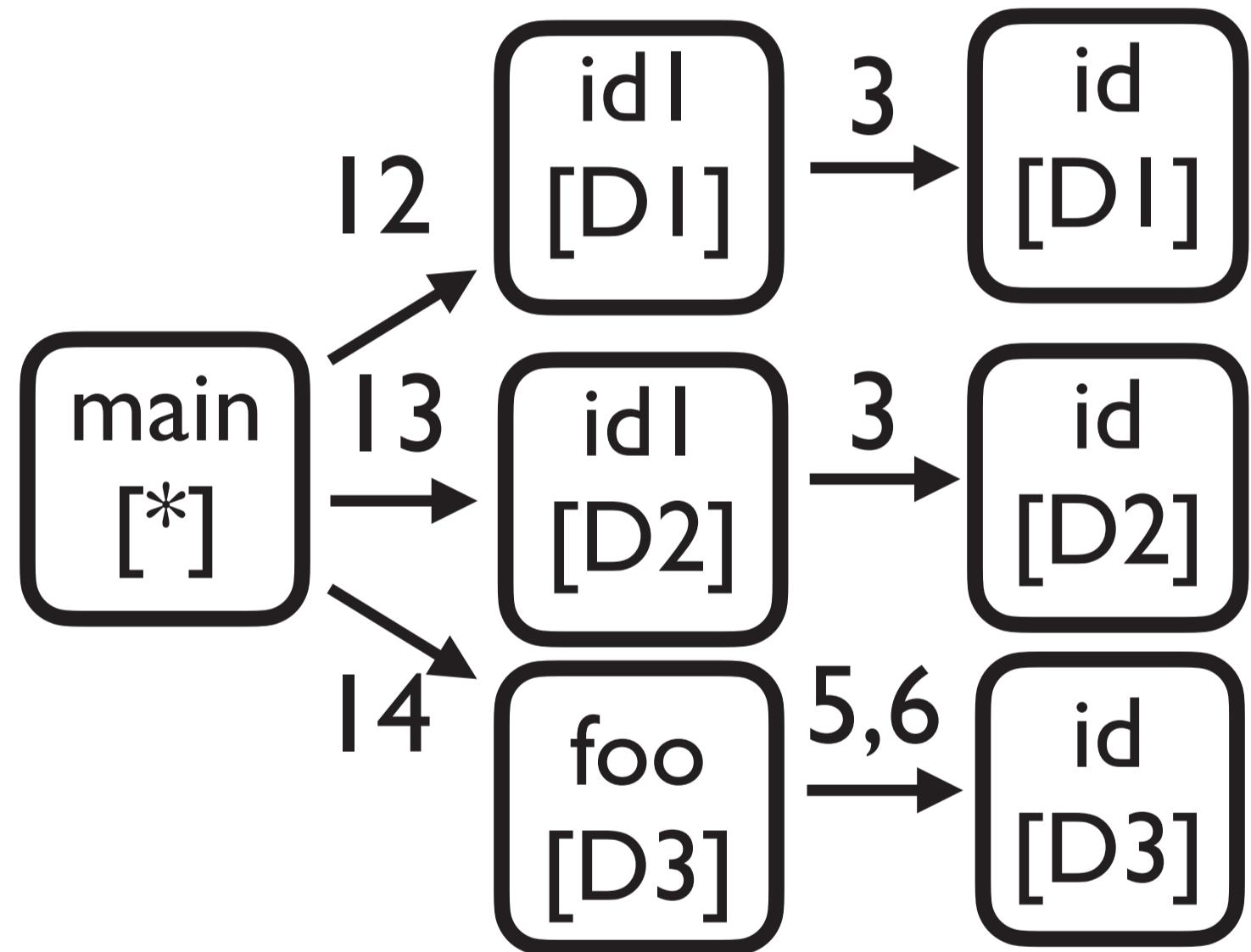
$I_{\text{callH+T'}}$

What is T' ?

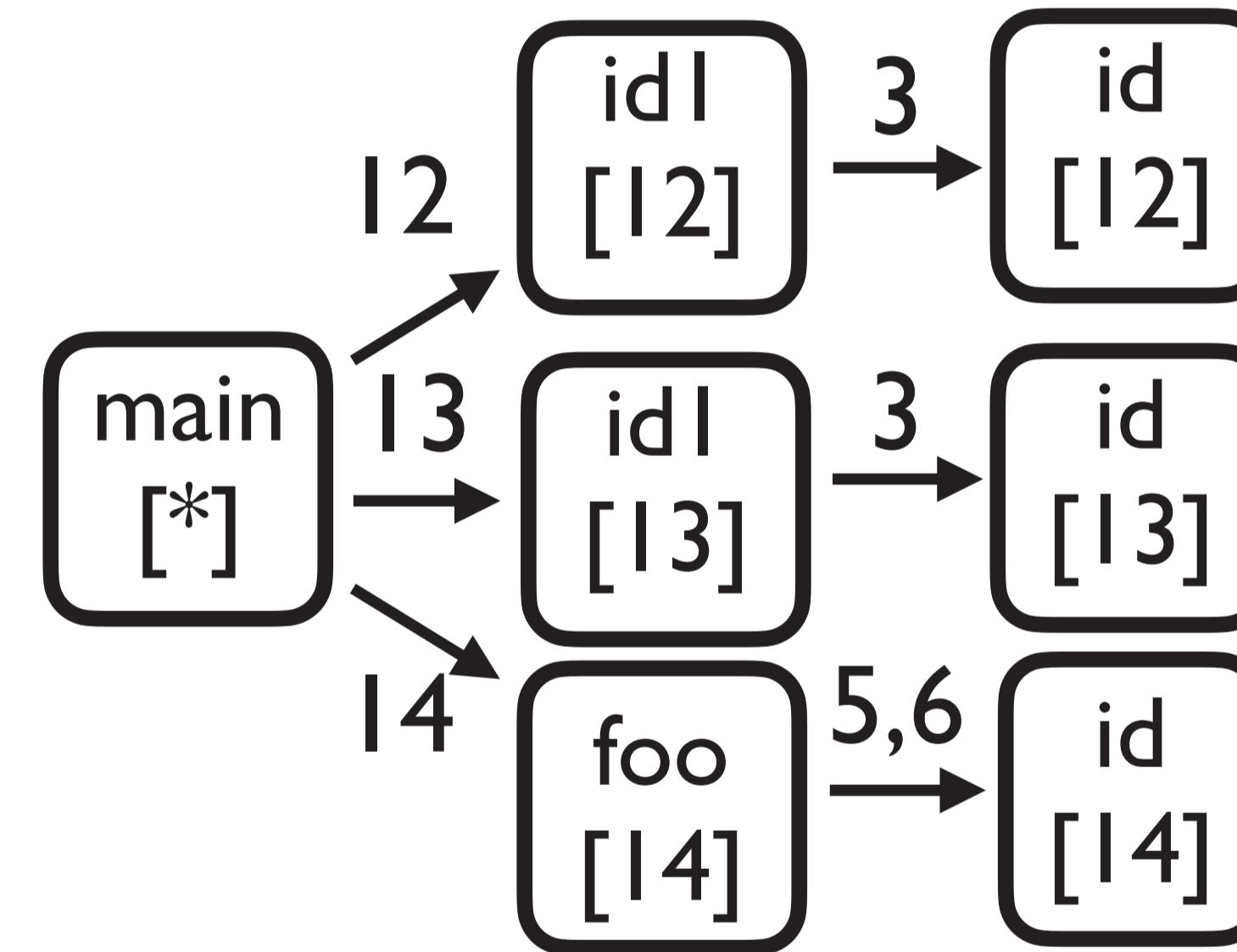
$$T' = I_1 \cup I_2 = \{3, 5, 6\}$$

Intuition Behind Simulation ($I_1 \cup I_2$)

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



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

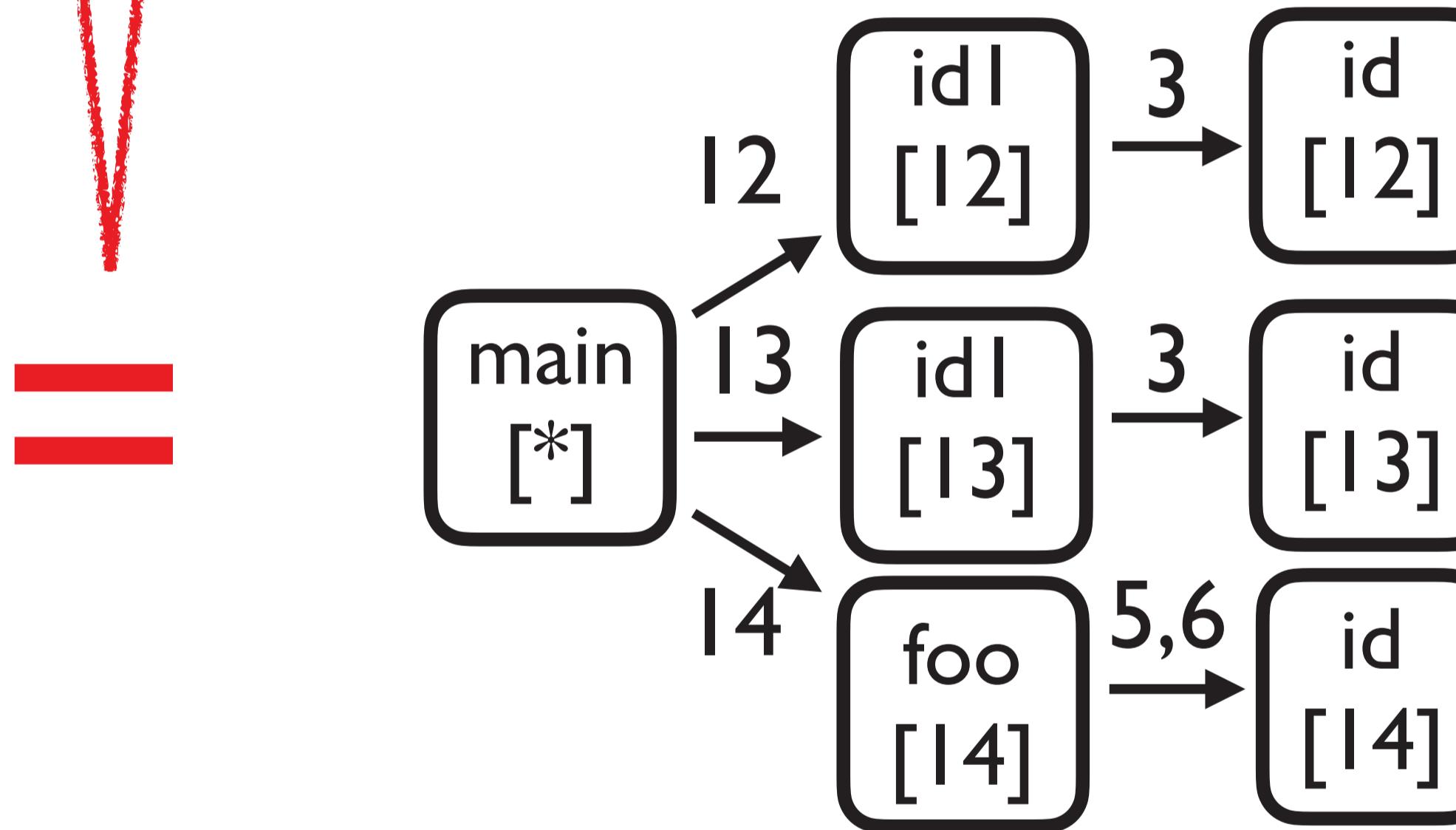
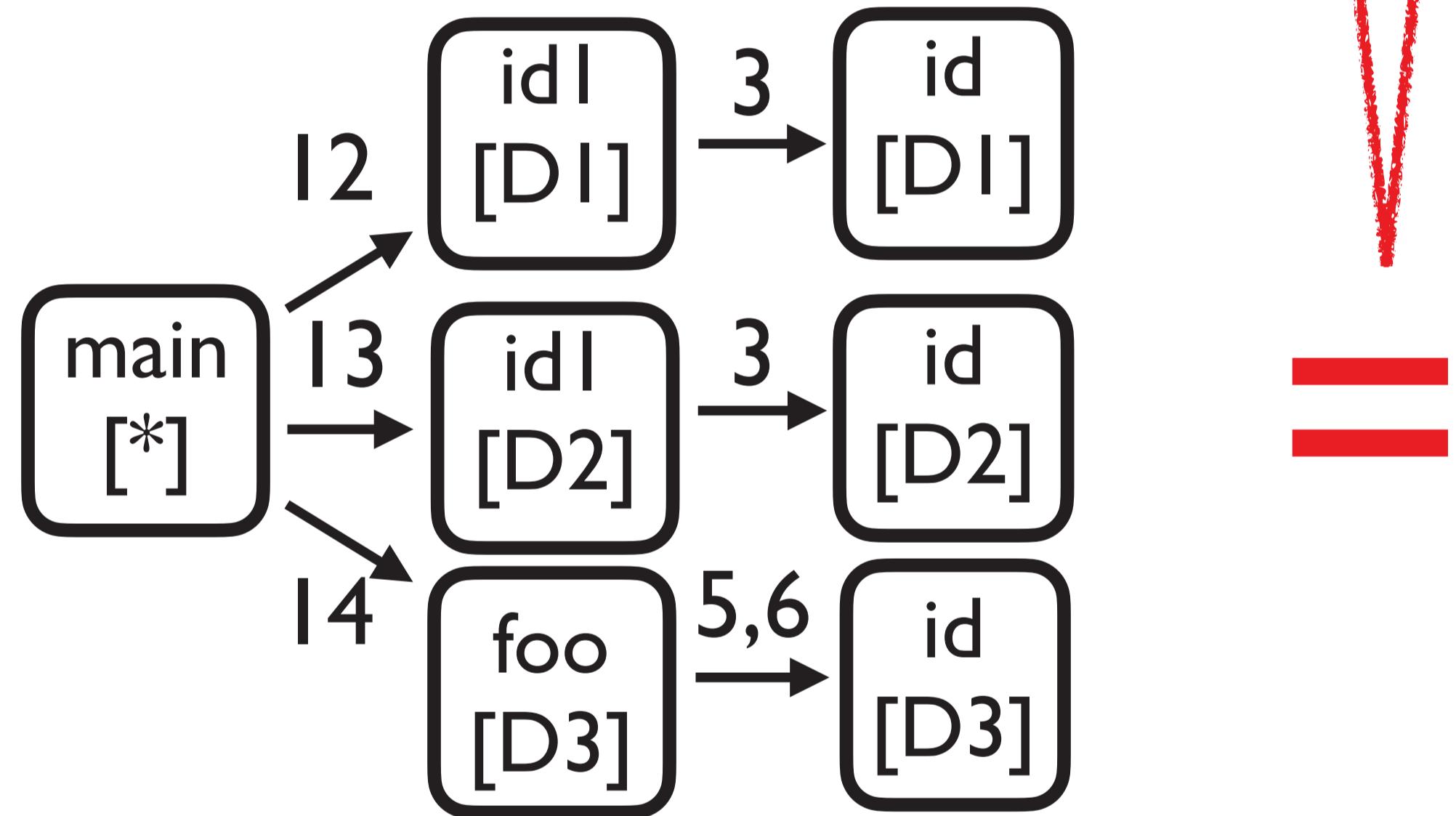


$I_{\text{callH+T'}} (T' = \{3,5,6\})$

Intuition Behind Simulation ($I_1 \cup I_2$)

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

Exactly the same analyses

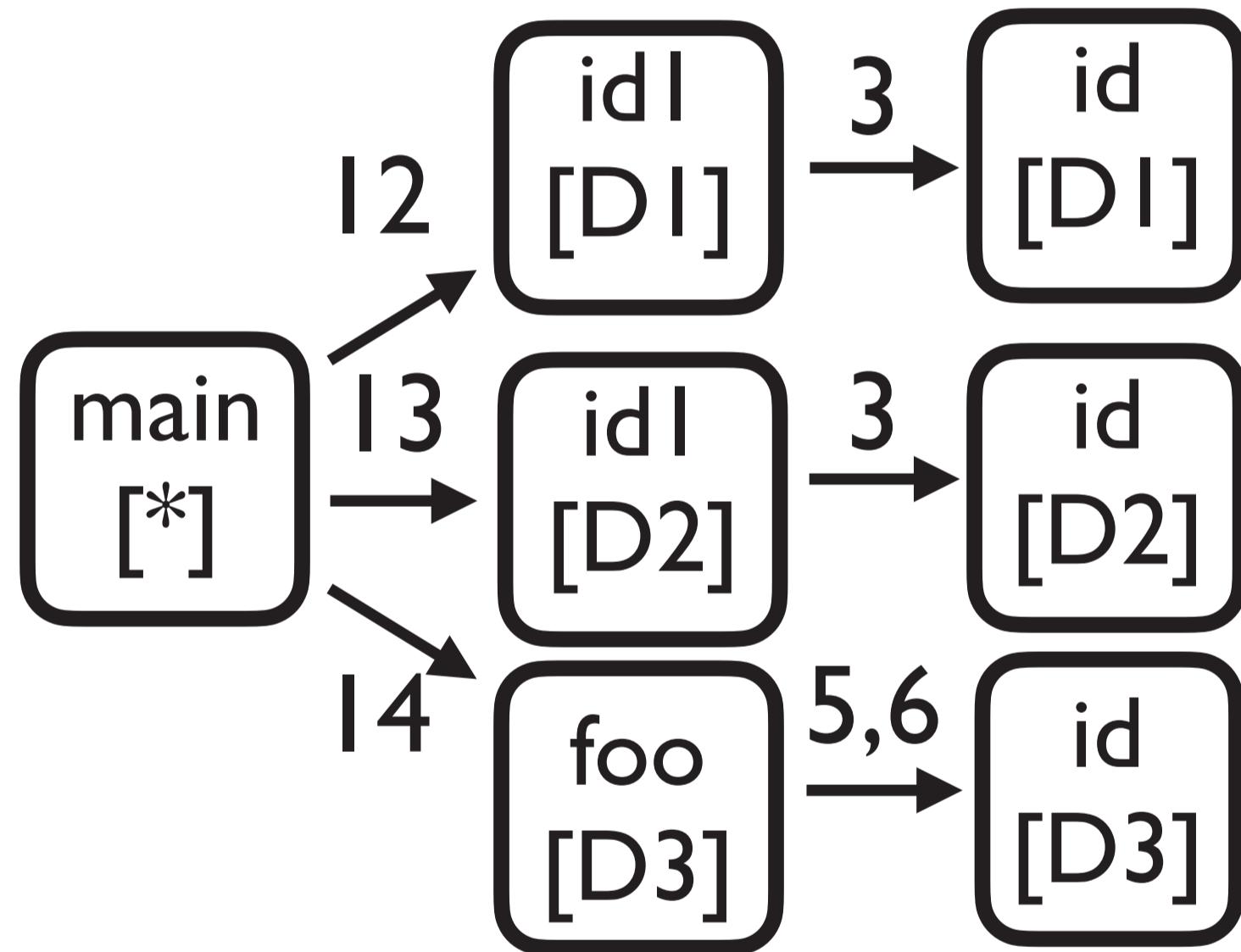


$I_{\text{obj}}H+T$ ($T = \emptyset$)

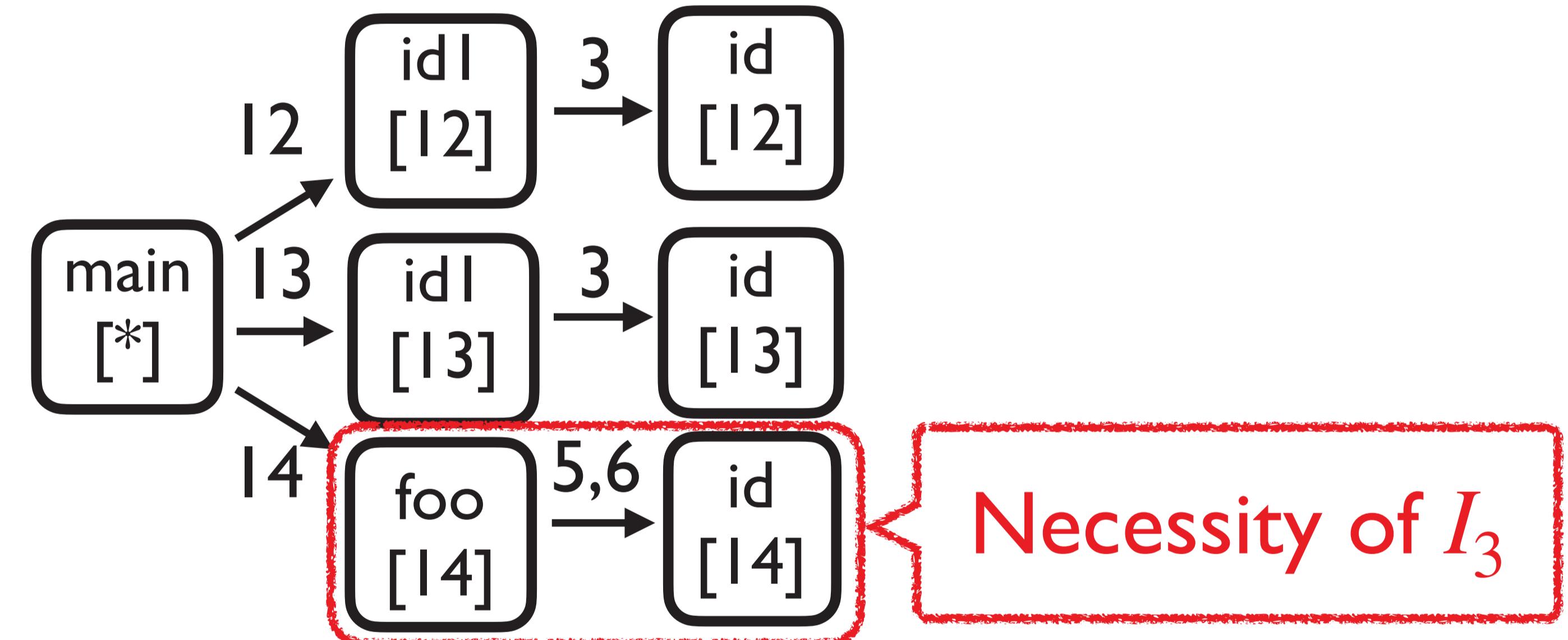
$I_{\text{call}}H+T'$ ($T' = \{3,5,6\}$)

Intuition Behind Simulation ($I_1 \cup I_2$)

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



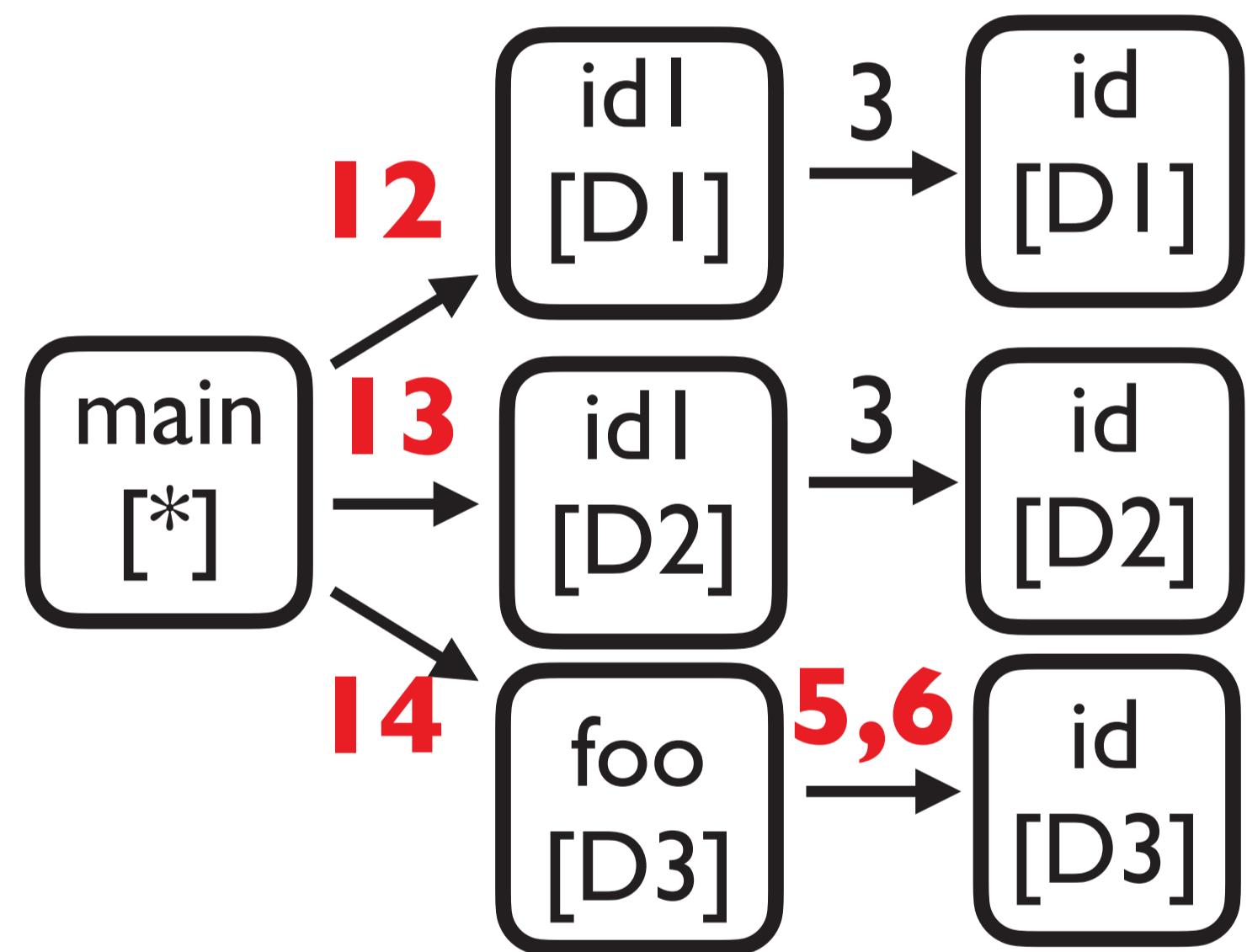
$I_{\text{objH+T}} (T = \emptyset)$



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

Intuition Behind Simulation (I_3)

- I_3 : Tunneling should be avoided for improving precision



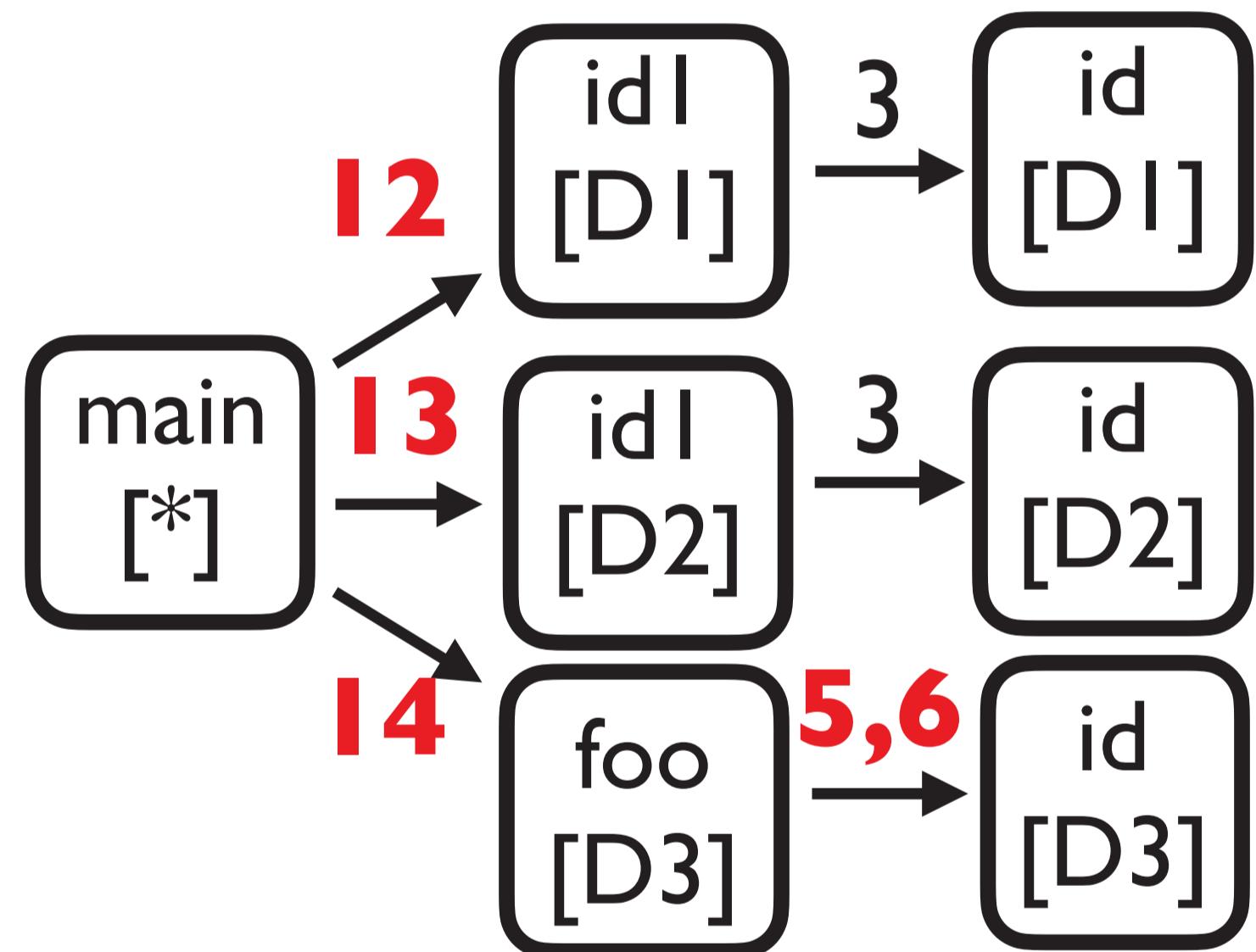
- I_1 : caller and callee methods have the **same context**
 $I_1 = \{3, 5, 6\}$
- I_2 : different caller ctx imply different callee ctx
 $I_2 = \{3\}$
- I_3 : given object sensitivity produced only one context

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

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

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

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

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

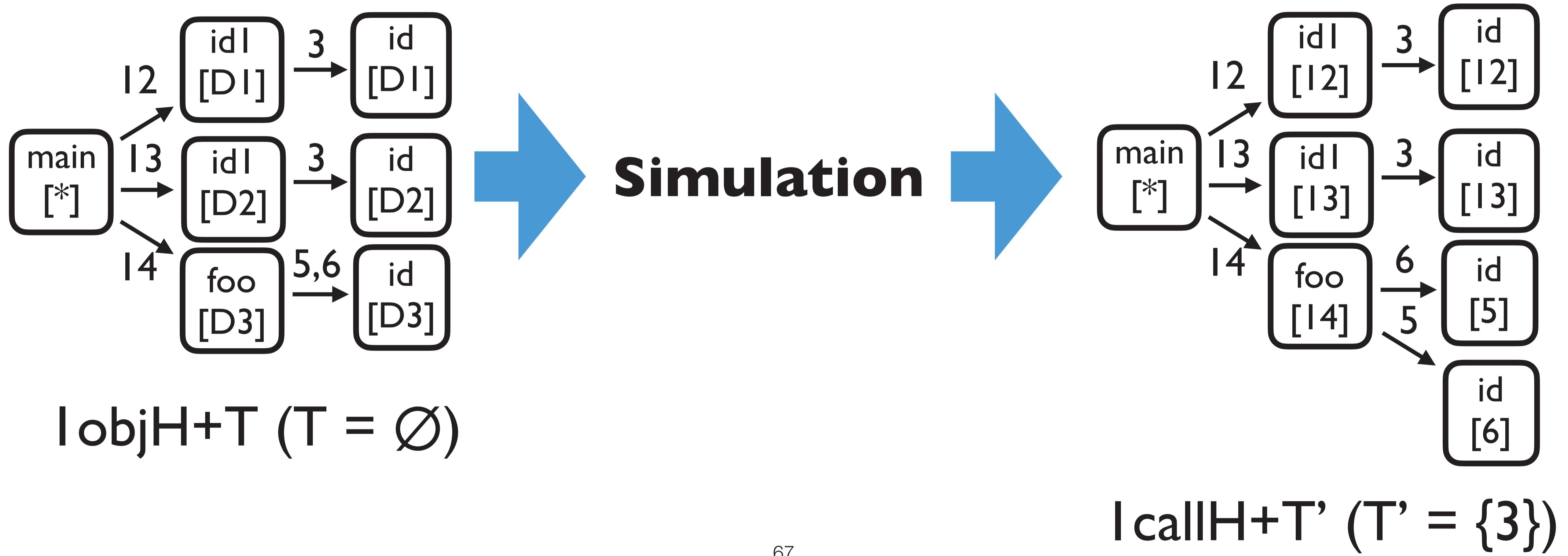
$$I_2 = \{3\}$$

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

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

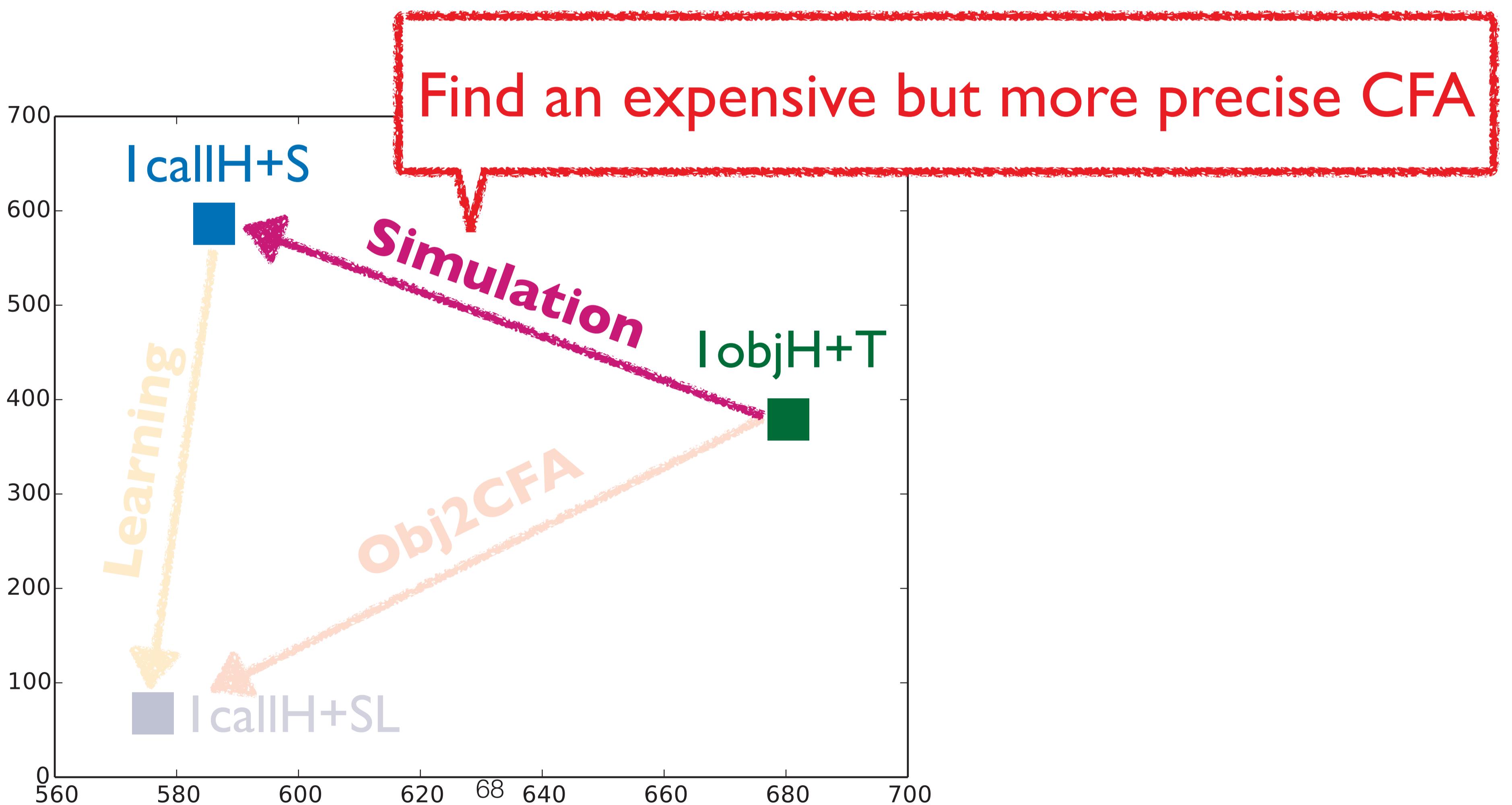
Technique I: Simulation

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



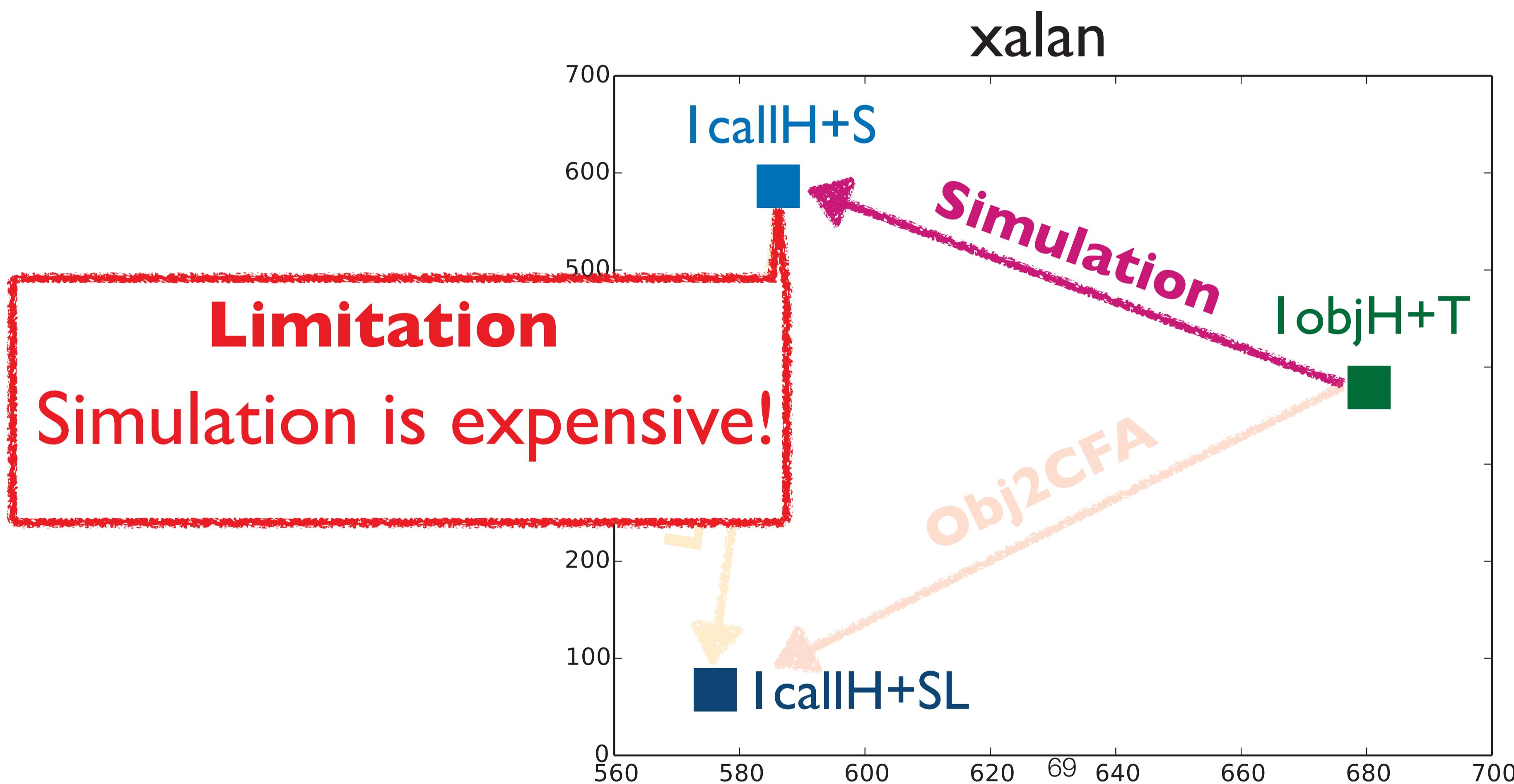
Our Technique : **Obj2CFA**

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



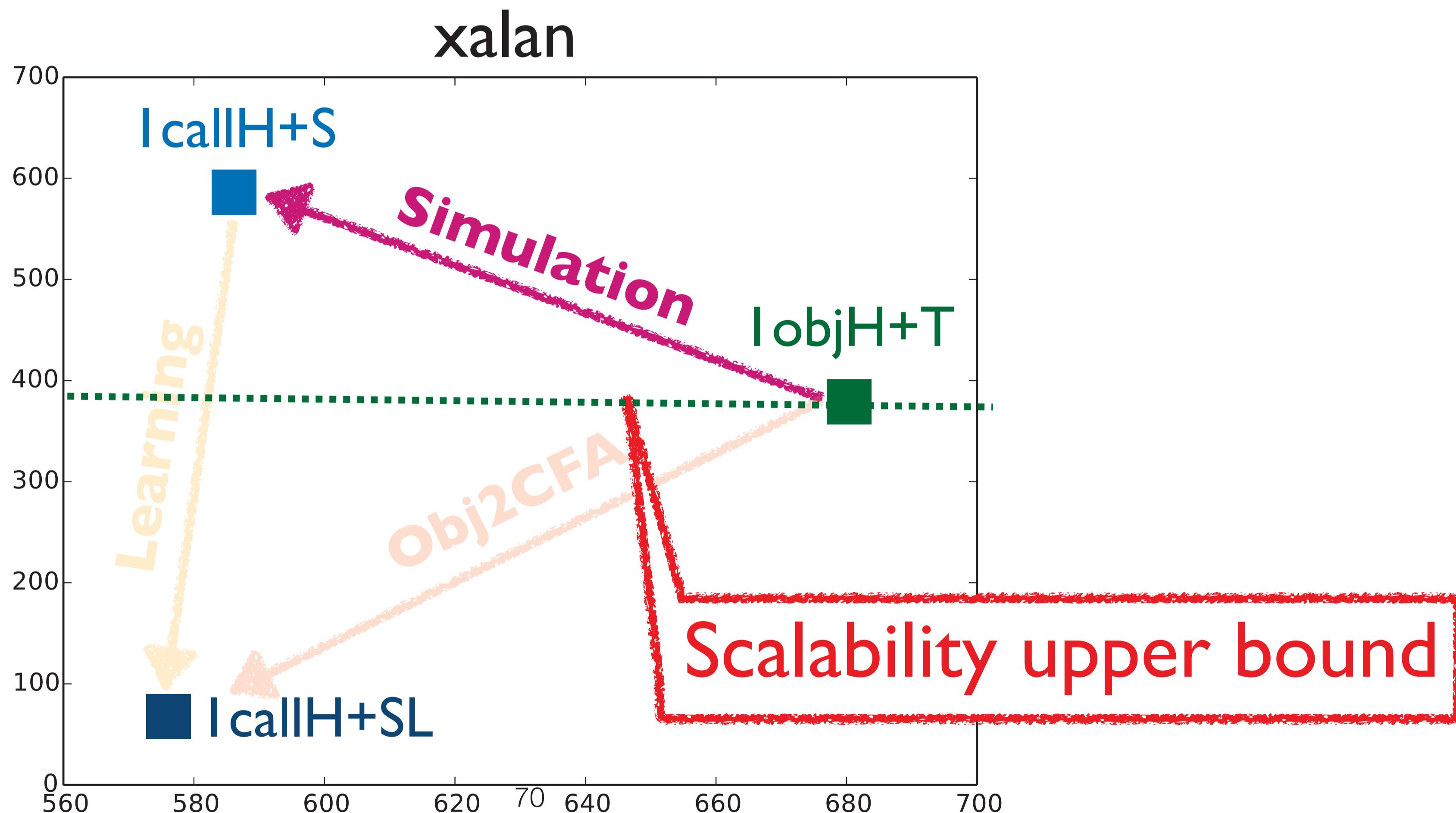
Our Technique : **Obj2CFA**

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



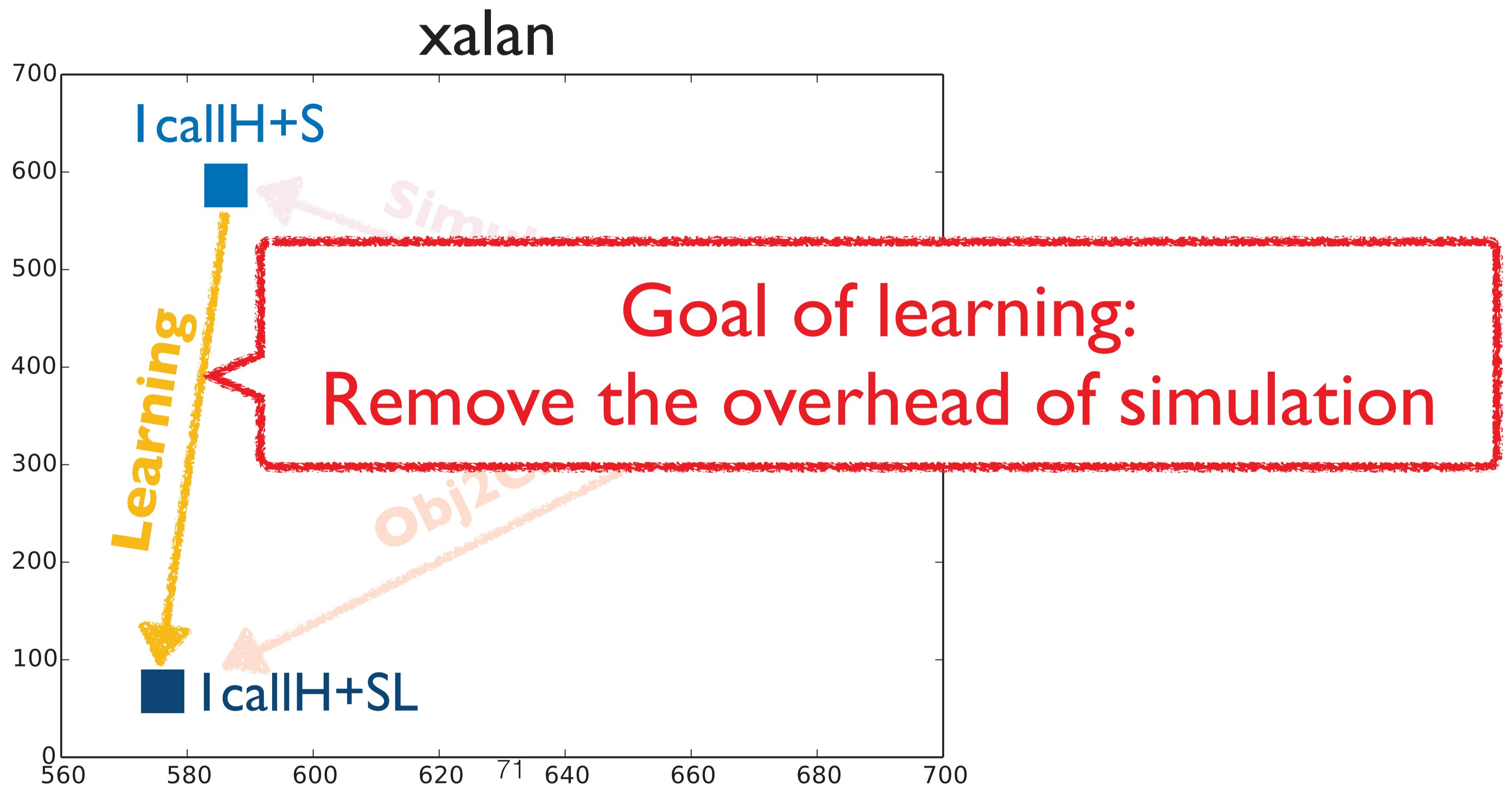
Our Technique : **Obj2CFA**

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



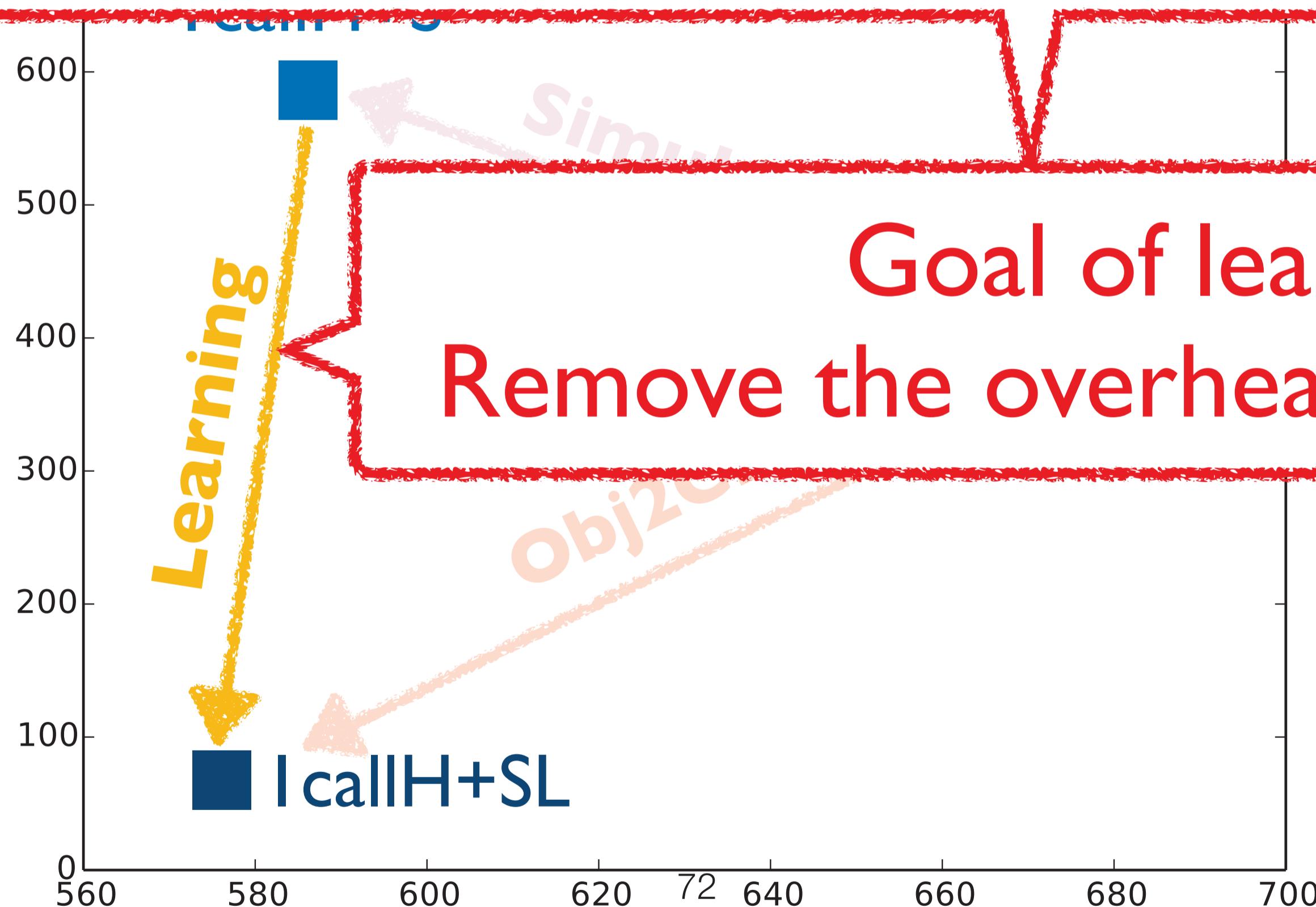
Our Technique : **Obj2CFA**

- **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



Goal of learning:

Remove the overhead of simulation

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

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

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@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 Datalog: a logic-based language for defining (recursive) 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, Door achieves several benefits, including full order-of-magnitude improvements in runtime. We compare Door with Lhoták and Hendren's PADDLE, which defines the state of the art for context-sensitive analyses. For the exact same logical points-to definitions (and, consequently, identical precision) Door is more than 15x faster than PADDLE for a 1-call-site sensitive analysis of the DaCapo benchmarks, with lower but still substantial speedups for other important analyses. Additionally, Door scales to very precise analyses that are impossible with PADDLE and Whaley et al.'s bddbdb, directly addressing open problems in past literature. Finally, our implementation is modular and can be easily configured to analyses with a wide range of characteristics, largely due to its declarativeness.

Categories and Subject Descriptors F.3.2 [Logics and Meanings of Programs]: Semantics of Programming Languages—Program Analysis; D.1.6 [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 has been devoted to efficient and precise techniques. Context-sensitive analyses are a class of precise points-to analyses. Context approaches qualify the analysis facts with tion, which captures a static notion of the of a method. Typical contexts include abst call-sites (for a *call-site sensitive* analysis meaning of “context-sensitive”) or receiving object-sensitive analysis).

In this work we present Door: a general points-to analysis framework that makes precise context-sensitive analyses report. Door implements a range of algorithms, insensitive, call-site sensitive, and object-all specified modularly as variations on a c. Compared to the prior state of the art, Door speeds up an order-of-magnitude for analyses.

The main elements of our approach are a logic language for specifying the program aggressive optimization of the Datalog program. Datalog for program analysis (both low-level and high-level [6, 9]) is far from new. Our novel approach, however, accounts for several orders of magnitude performance improvement: unoptimized runs over 1000 times more slowly. Generations fit well the approach of handling a database, by specifically targeting the incremental evaluation of Datalog programs. Furthermore, our approach is entirely declarative: the logic required both for compilation as well as for handling the full semantics of the Java language (e.g., static initializations, reference objects, threads, exceptions, ref makes our pointer analysis specifications but also efficient and easy to tune. Generations strong data point in support of declarative programming that prohibitively much human effort implementing and optimizing complex mutations at an operational level of abstraction.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

OOPSLA 2009, October 25–29, 2009, Orlando, Florida, USA.
Copyright © 2009 ACM 978-1-60558-734-9/09/10...\$5.00

Hybrid Context-Sensitivity for Points-To Analysis

George Kastrinis Yannis Smaragdakis

Department of Informatics

University of Athens

{gkastrinis,smaragd}@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 (kCFA) and object-sensitivity. Combining both flavors of context-sensitivity increases precision but at an infeasibly 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 only when analyzing selected language features, we can closely approximate the precision of an analysis that keeps both contexts at all times. In terms of speed, the selective combination of both kinds of context not only vastly outperforms non-selective combinations but is also faster than a mere object-sensitive analysis. This result holds for a large array of analyses (e.g., 1-object-sensitive, 2-object-sensitive with a context-sensitive heap, type-sensitive) establishing a new set of performance/precision sweet spots.

Categories and Subject Descriptors F.3.2 [Logics and Meanings of Programs]: Semantics of Programming Languages—Program Analysis; D.3.4 [Programming Languages]: Processors—Compilers

General Terms Algorithms, Languages, Performance

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

1. Introduction

Points-to analysis is a static program analysis that consists of computing all objects (typically identified by allocation site) that a program variable may point to. The area of points-to analysis (and its close relative, *alias analysis*) has been the focus of intense research and is among the most standardized and well-understood of inter-procedural analyses. The emphasis of points-to analysis algorithms is on combining fairly precise modeling of pointer behavior with scalability. The challenge is to pick judicious approximations that will allow satisfactory precision at a reasonable cost. Furthermore, although increasing precision often leads to higher asymptotic complexity, this worst-case behavior is rarely encountered in actual practice. Instead, techniques that are effective at maintaining good precision often also exhibit better average-case performance, since smaller points-to sets lead to less work.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

PLDI'13, June 16–19, 2013, Seattle, WA, USA.

Copyright © 2013 ACM 978-1-4503-2014-6/13/06... \$15.00

One of the major tools for exploring the precision/performance tradeoff has been context-sensitivity. Qualifying local context information: the analysis unit uses the same context value, while separating different contexts. This approach tries to naturally result in any static analysis from different dynamic program paths. Sensitivity have been explored in the [22, 23] and *object-sensitivity* [18, 19].

A call-site-sensitive/kCFA analyzes labels of instructions that may call the method. That is, the analysis separates information about method arguments per call-stack (i.e., method invocations that led to the current context). The analysis separates information on method invocations that led to the object in the code example below, a 1-call-to-a-context-insensitive analysis will disregard method `foo` on lines 7 and 9. This means `foo` separately for two cases: that of it pointing to anything `obj1` may point to, and `obj2` may point to.

```
1 class C {
2     void foo(Objet o) { ... }
3 }
4
5 class Client {
6     void bar(C c1, C c2) { ...
7         c1.foo(obj1);
8         ...
9         c2.foo(obj2);
10    }
11 }
```

In contrast, object-sensitivity uses objects as the analysis unit. (Hence, a better name for “object-allocation-site sensitivity”.) Thus, if an object, the analysis separates the allocation site of the receiver object (the method is called), as well as of the context. Thus, in the above example, the analysis will analyze `foo` separately depending on whether `c1` and `c2` may point to. It will fragment neither whether `c1` and `c2` may point to nor how many objects: the allocation site may be remote and unrelated to the context. Thus, it is not possible to compare the precision of a context-sensitive analysis and a call-site-sensitive analysis in practice. It is not even clear whether the object semantics of all calls to `foo` as one case, as two,

Yannis Smaragdakis George Kastrinis George Balatsouras
Department of Informatics University of Athens
{smaragd,gkastrinis,gbalats}@di.uoa.gr

Abstract

Context-sensitivity is the primary approach for adding more precision to points-to analysis, while hopefully also maintaining scalability. An oft-reported problem with context-sensitive analyses, however, is that they are bi-modal: either the analysis is precise enough that it manipulates only manageable sets of data, and thus scales impressively well, or the analysis gets quickly derailed at the first sign of imprecision and becomes orders-of-magnitude more expensive than would be expected given the program's size. There is currently no approach that makes precise context-sensitive analyses (of any flavor: call-site-, object-, or type-sensitive) scale across the board at a level comparable to that of a context-insensitive analysis. To address this issue, we propose introspective analysis: a technique for uniformly scaling context-sensitive analysis by eliminating its performance-detrimental behavior, at a small precision expense. Introspective analysis consists of a common adaptivity pattern: first perform a context-insensitive analysis, then use the results to selectively refine (i.e., analyze context-sensitively) program elements that will not cause explosion in the running time or space. The technical challenge is to appropriately identify such program elements. We show that a simple but principled approach can be remarkably effective, achieving scalability (often with dramatic speedup) for benchmarks previously completely out-of-reach for deep context-sensitive analyses.

Categories and Subject Descriptors F.3.2 [Logics and Meanings of Programs]; Semantics of Programming Languages—Program Analysis; D.3.4 [Programming Languages]: Processors—Compilers

General Terms Algorithms, Languages, Performance

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

1. Introduction

Points-to analysis is probably the most common whole-program static analysis, and often serves as a substrate for a variety of high-level program analysis tasks. Points-to analysis computes the set of objects (abstracted as their allocation sites) that a program variable may point to during runtime. The promise, as well as the challenge,

of points-to analysis is to yield usefully precise results while sacrificing scalability: the analysis inputs are algorithms are typically quadratic or cubic near-linear behavior in practice, by exploiting and maintaining precision. Indeed precision and scalability go hand-in-hand in a good points-to analysis: algorithms are often found to be both more accurate and smaller points-to sets lead to less memory usage.

Context-sensitivity is a common way of sacrificing scalability in points-to analysis. It consists of tables and objects with context information: formation (e.g., "what objects this method analyzes over all possible executions that map to this context"). While separating executions that map to different contexts, context-sensitivity attempts to avoid predicting the behavior of different dynamic programs. Context-sensitivity comes in many flavors, depending on the information used, such as call-site-sensitivity [22, 19, 20], and type-sensitivity [24].

An oft-remarked fact about context-sensitivity is that even the best algorithms have a common failing: they cannot maintain precision. Past literature remarks of a [...] deep-context analysis is bisimulation analysis has been associated with contexts" [15]: "algorithms completely hit a precision limit, with the number of tuples exploding exponentially". Recent published results [12] fail to run a 22ms in under 90mins for 2 of 10 DaCapo benchmarks take more than 1,000sec, although benchmarks of similar or larger size get analyzed in under 100ms.

Thus, when context-sensitivity works, it terms of both precision and performance. When it fails miserably, quickly exploding in context-insensitive analyses uniformly scale up. Figure 1 vividly demonstrates this phenomenon on the DaCapo benchmarks, analyzed with the Doop context-insensitive (insens) analysis and a 20bjH context-sensitive heap (20bjH). The analysis time of the longest-running benchmark, hsqldb and jython, timed out after 90mins and would not terminate even for much longer. Interestingly, context-insensitive analyses vary in performance, while context-sensitivity often causes memory use to explode.

Faced with this unpredictability of context-sensitivity, the common reaction is to avoid it, favoring conservative, and, consequently, missing significant well-behaved programs. Even worse, for some, chewing expensive context-sensitivity is not a good idea: context-insensitive analysis is just not good enough [4] and academic researchers [3] alike.

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

HUN JEONG, Korea University, Republic of Korea
NSEOK JEON*, Korea University, Republic of Korea
NGDEOK CHA, Korea University, Republic of Korea
AKJOO OH†, Korea University, Republic of Korea

present a new data-driven approach to achieve highly cost-effective context-sensitive points-to analysis in Java. While context-sensitivity has greater impact on the analysis precision and performance than any other precision-improving techniques, it is 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 assigning 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 heuristic rules, in disjunctive form of 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, implemented our approach in the Doop framework and evaluated using three types of context-sensitive analyses: conventional object-sensitivity, selective hybrid object-sensitivity, and type-sensitivity. In all cases, experimental results show that our approach significantly outperforms existing techniques.

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

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

M Reference Format:

Jun Jeong, Minseok Jeon, Sungdeok Cha, and Hakjoo Oh. 2017. Data-Driven Context-Sensitivity for Points-to Analysis. *Proc. ACM Program. Lang.* 1, OOPSLA, Article 100 (October 2017), 27 pages.
<https://doi.org/10.1145/3133924>

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.

For object-oriented languages, context-sensitive points-to analysis is important as it must distinguish method's local variables and objects in different calling-contexts. For languages like Java,

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

*Authors' email addresses: S. Jeong, gifaranga@korea.ac.kr; M. Jeon, minseok_jeon@korea.ac.kr; S. Cha, scha@korea.ac.kr;
†Hakjoo_Oh@korea.ac.kr

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

© 2017 Association for Computing Machinery.
978-1-4503-4963-5/17/10-ART100
<https://doi.org/10.1145/3133924>

2009 (OOPSLA)

2011 (POPL)

2013 (PLDI)

2014
(PLDI)

2016 (SAS)

2017 (OOPSLA)

Setting

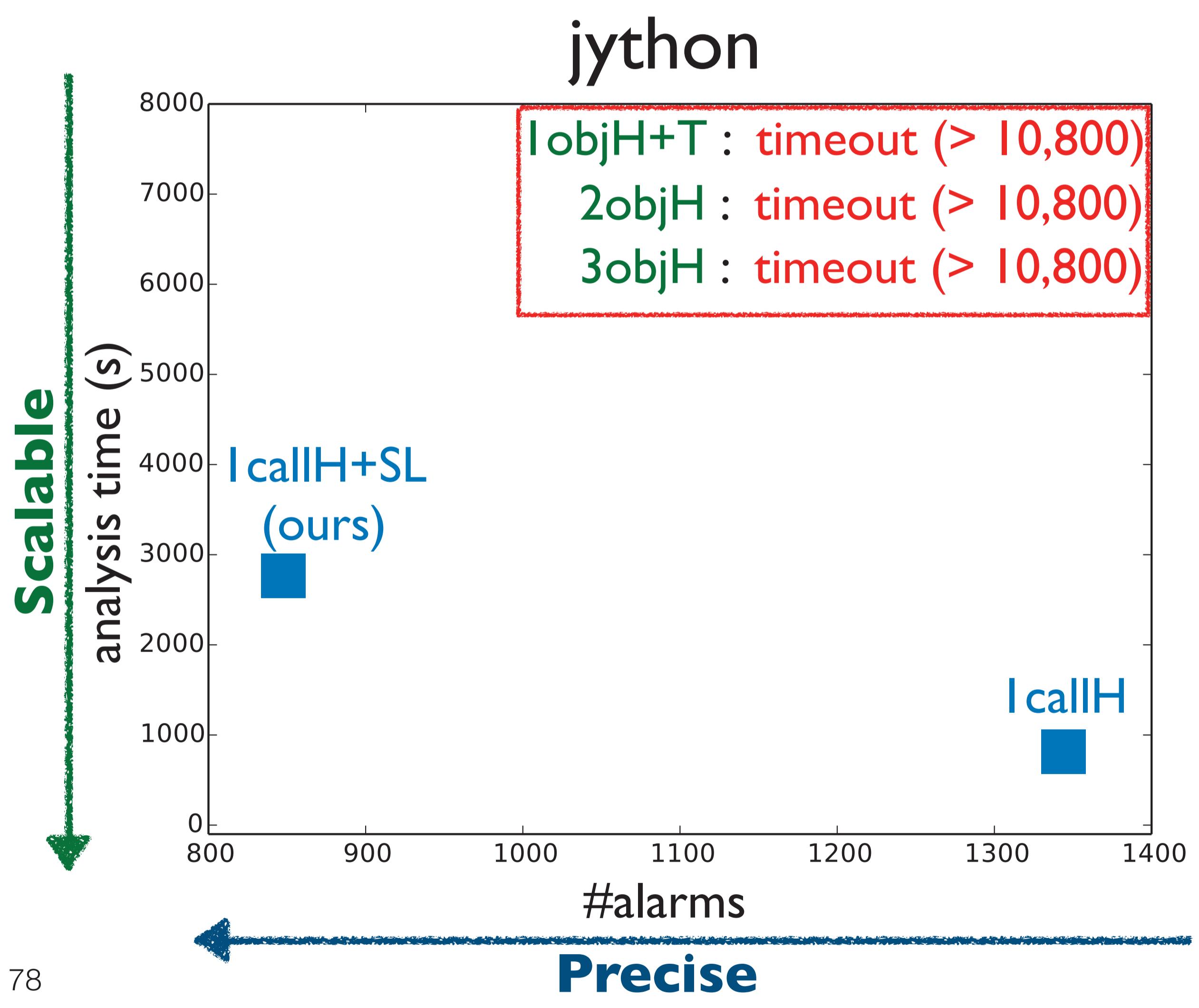
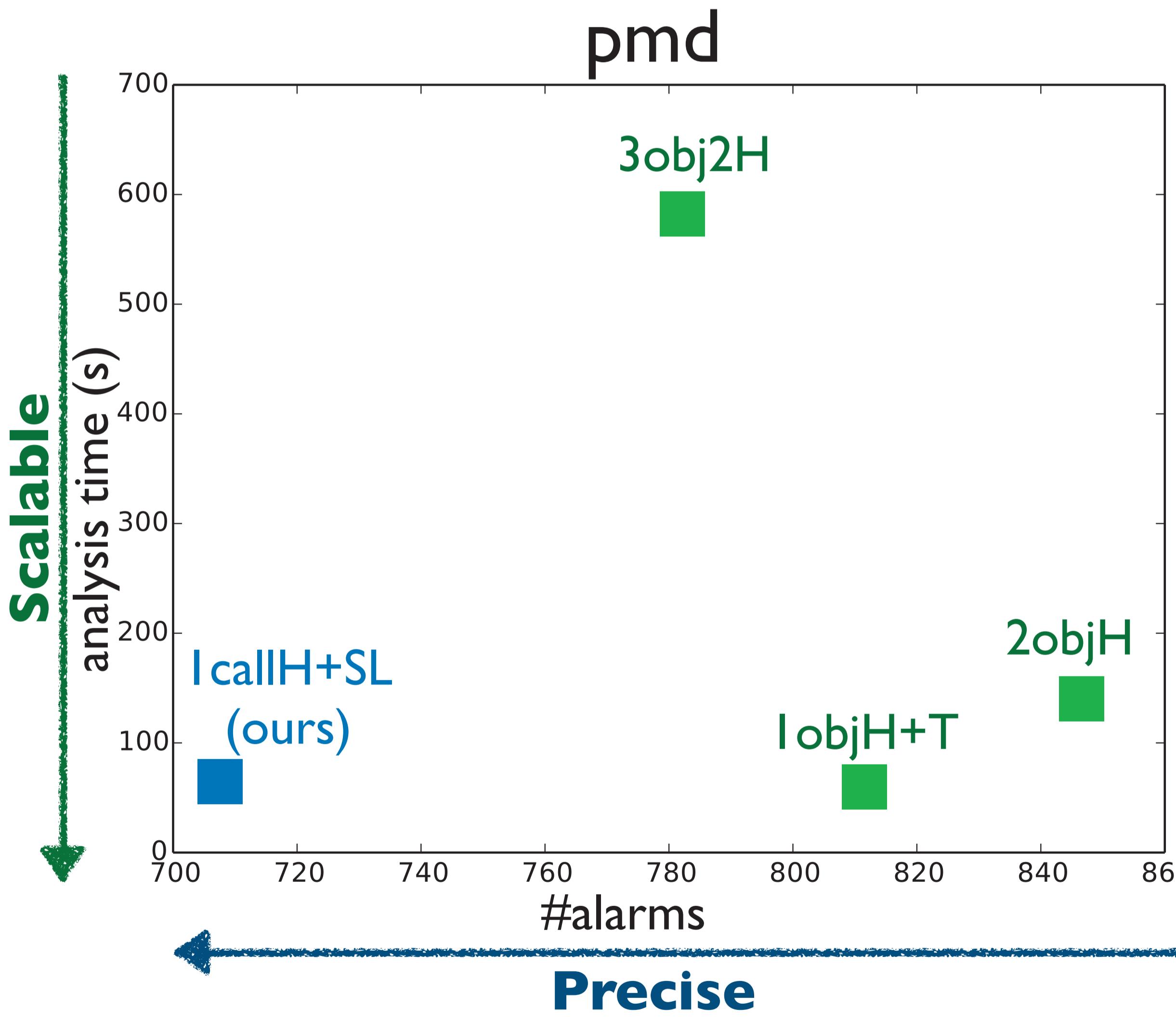
- Doop
 - Pointer analysis framework for Java
- Research Question: which one is better?

Call-site sensitivity vs Object sensitivity

Context tunneling is included

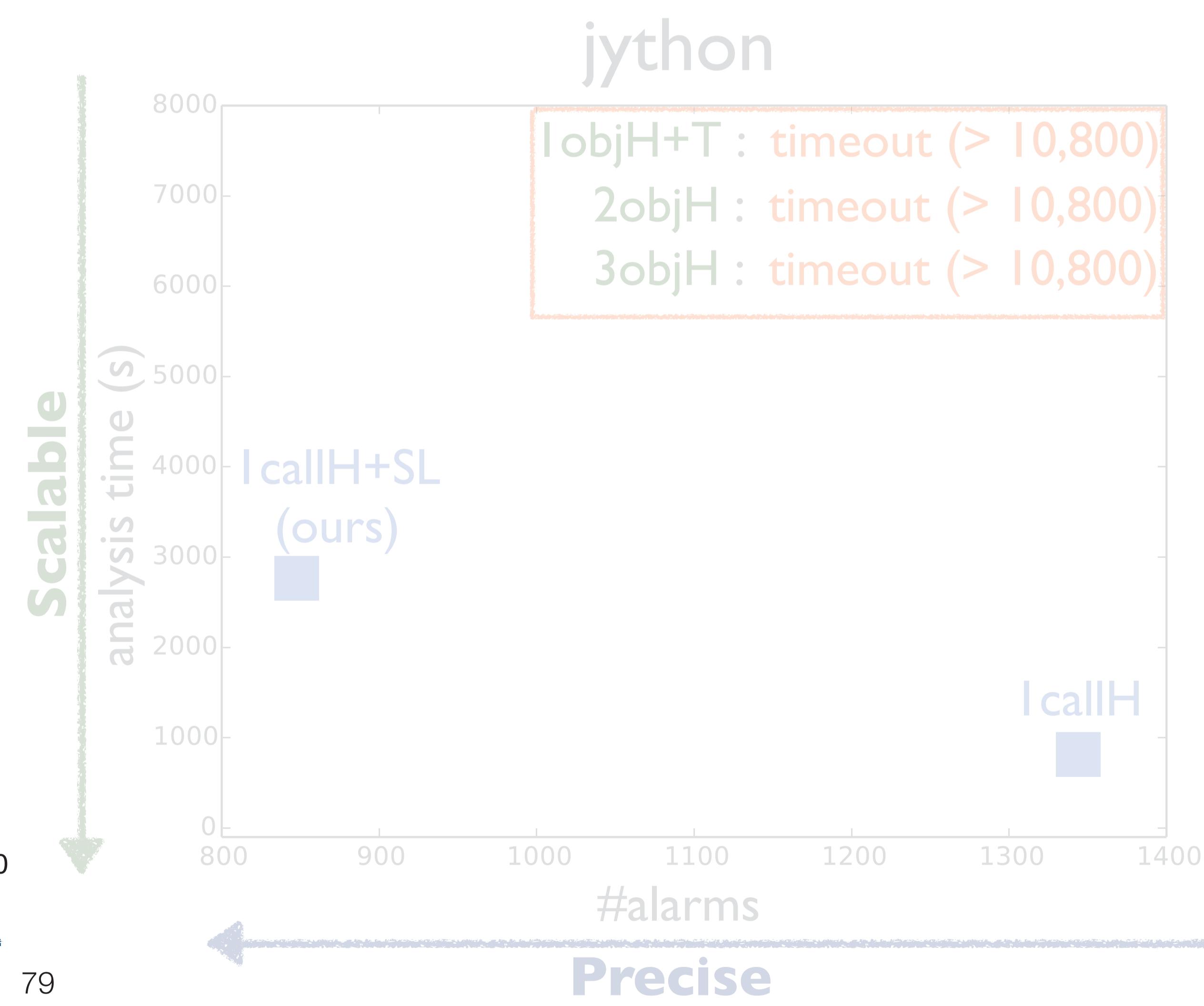
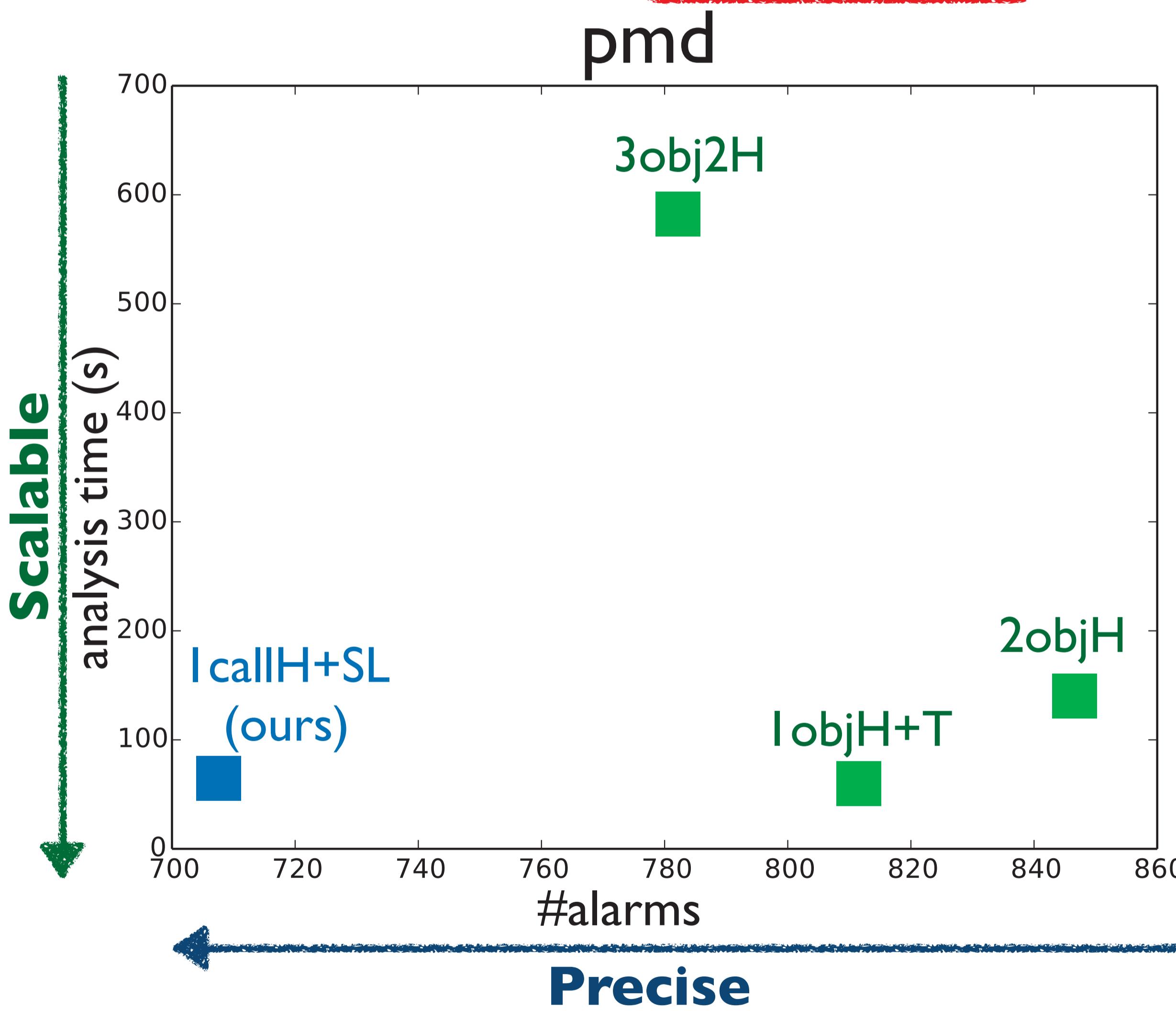
Call-site Sensitivity vs Object Sensitivity

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



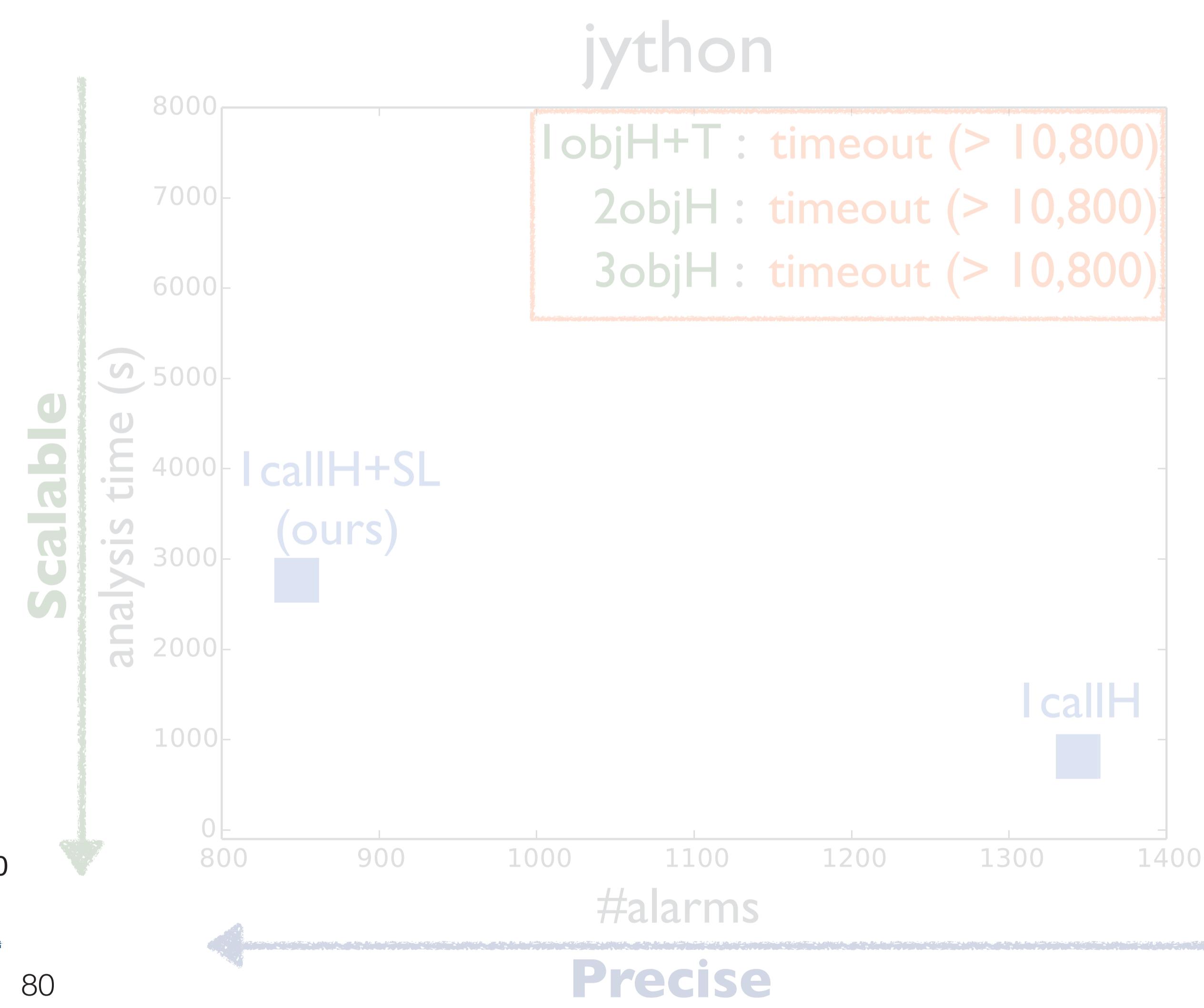
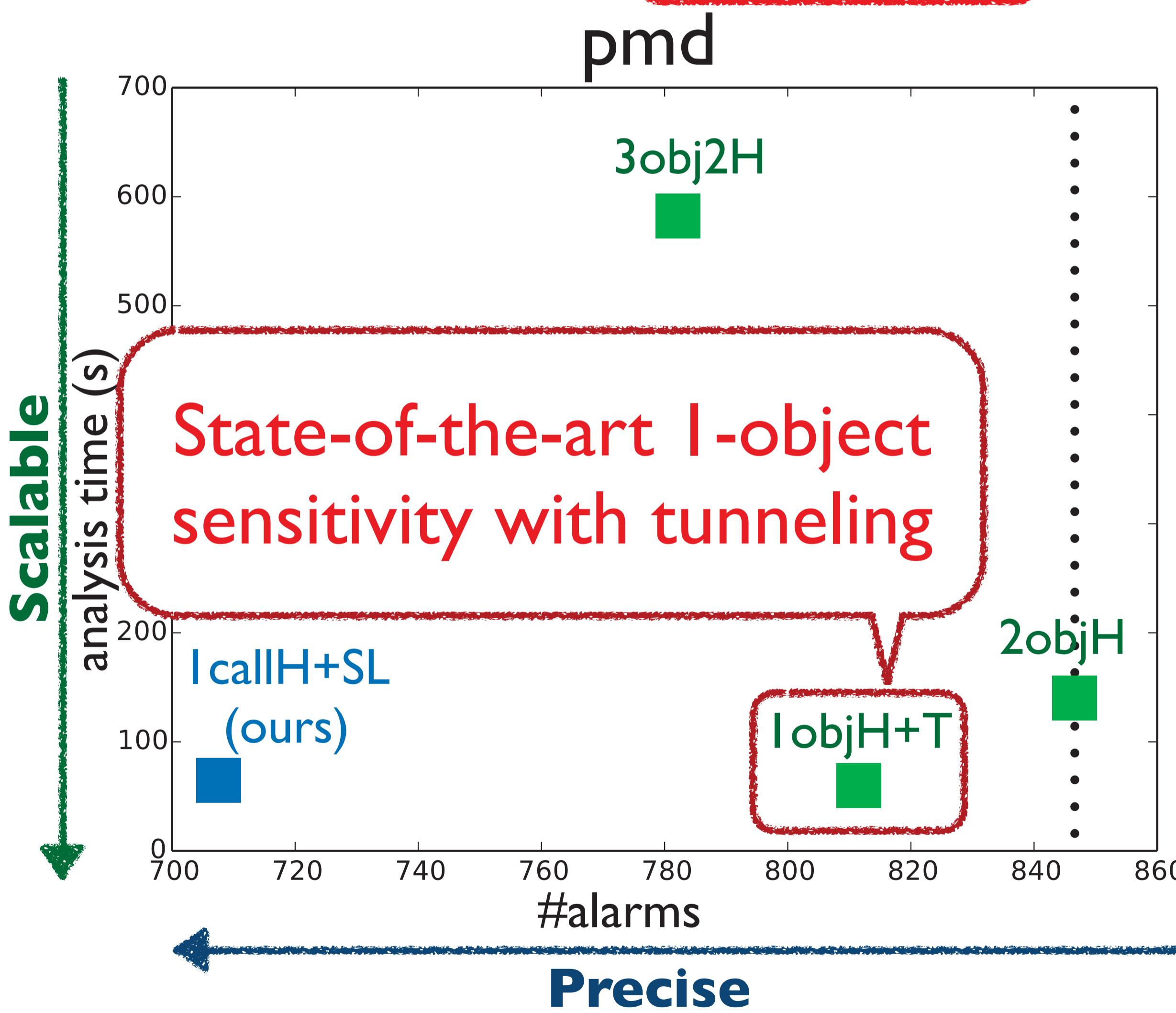
Call-site Sensitivity vs Object Sensitivity

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



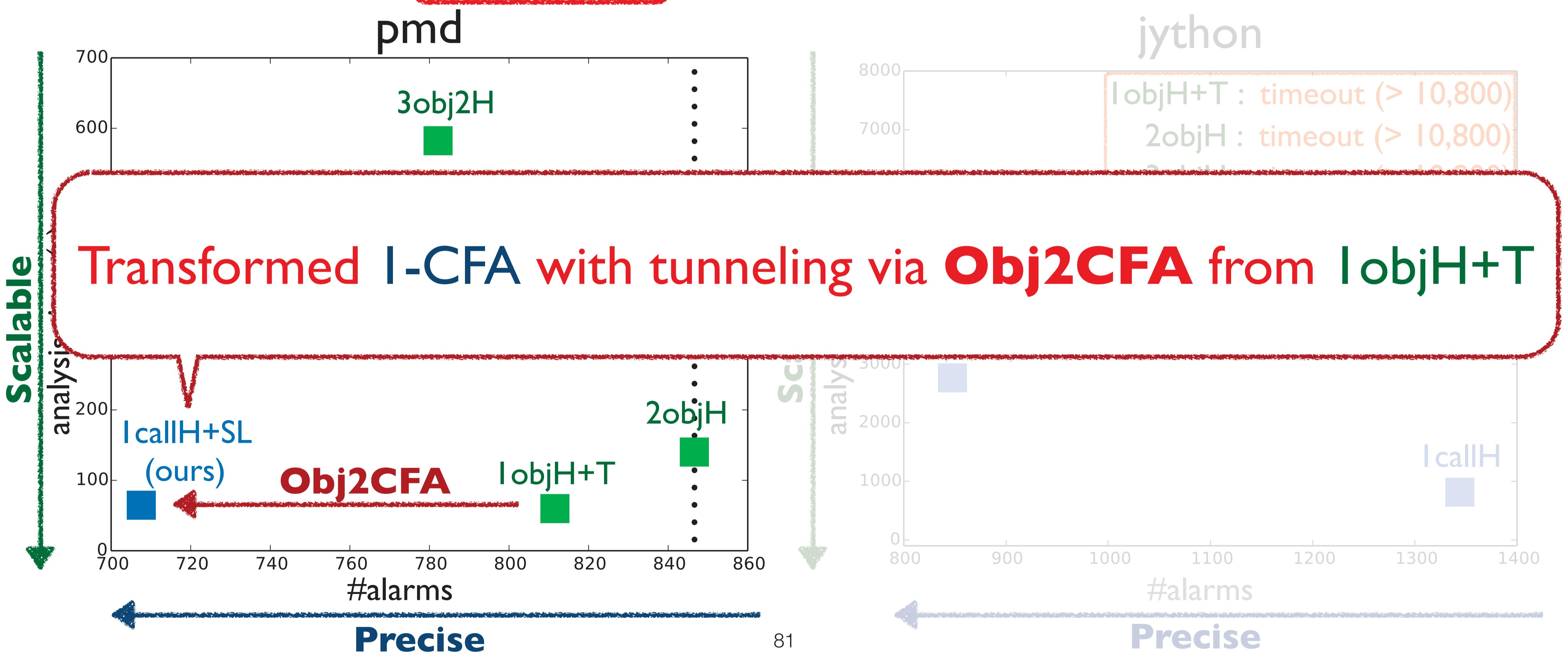
Call-site Sensitivity vs Object Sensitivity

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



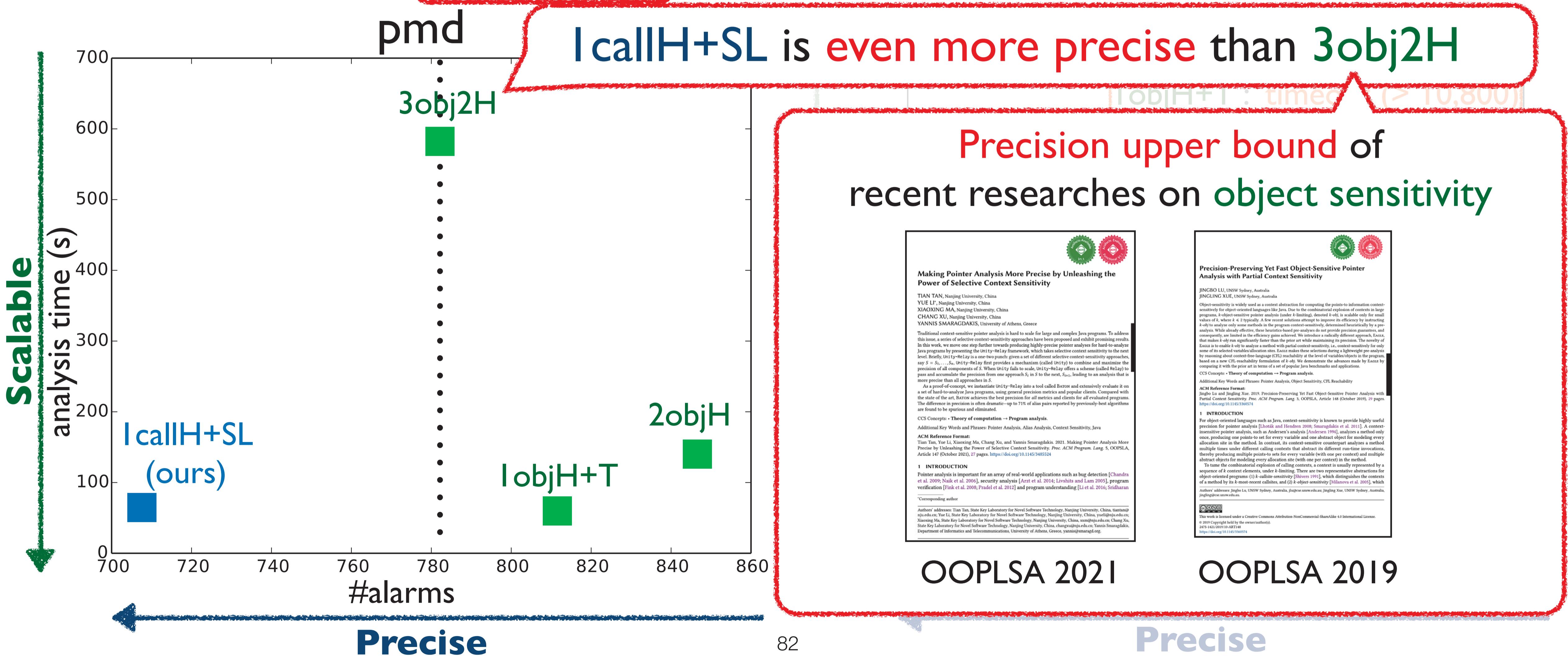
Call-site Sensitivity vs Object Sensitivity

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



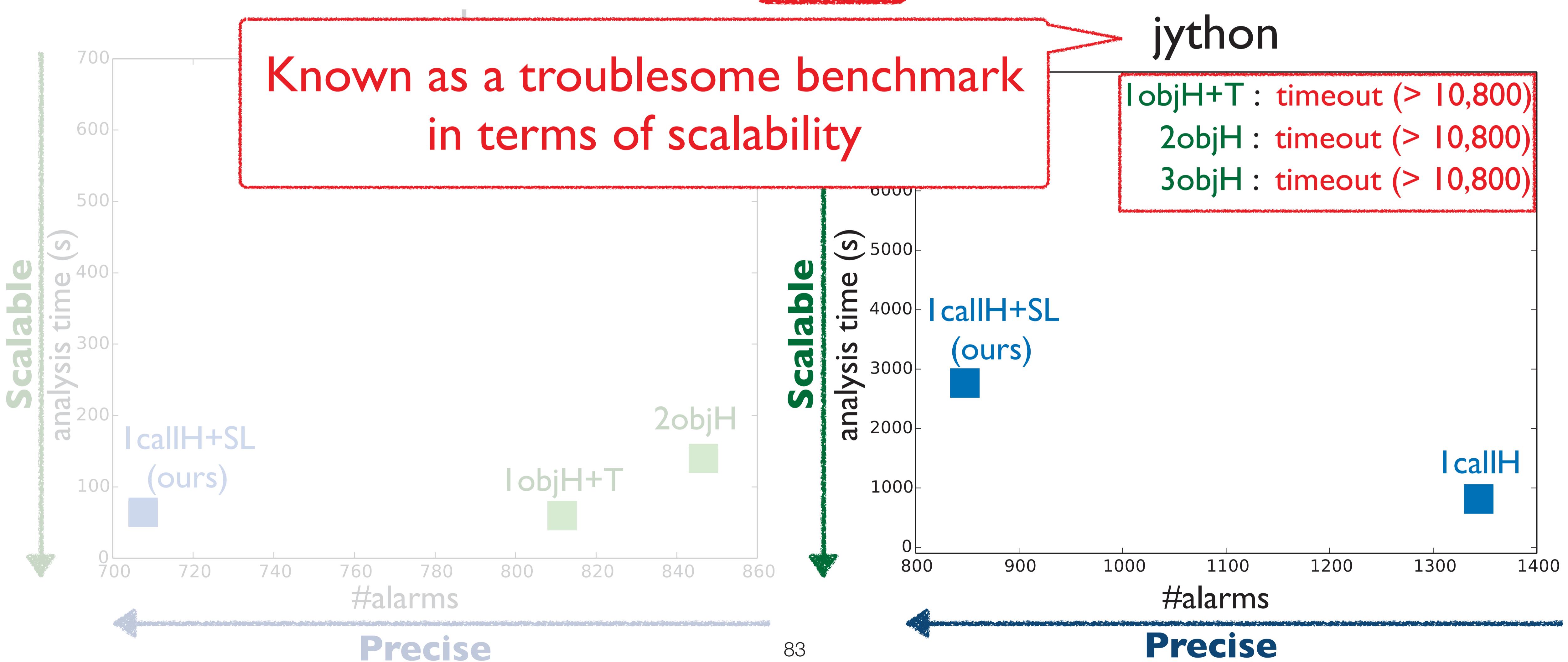
Call-site Sensitivity vs Object Sensitivity

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



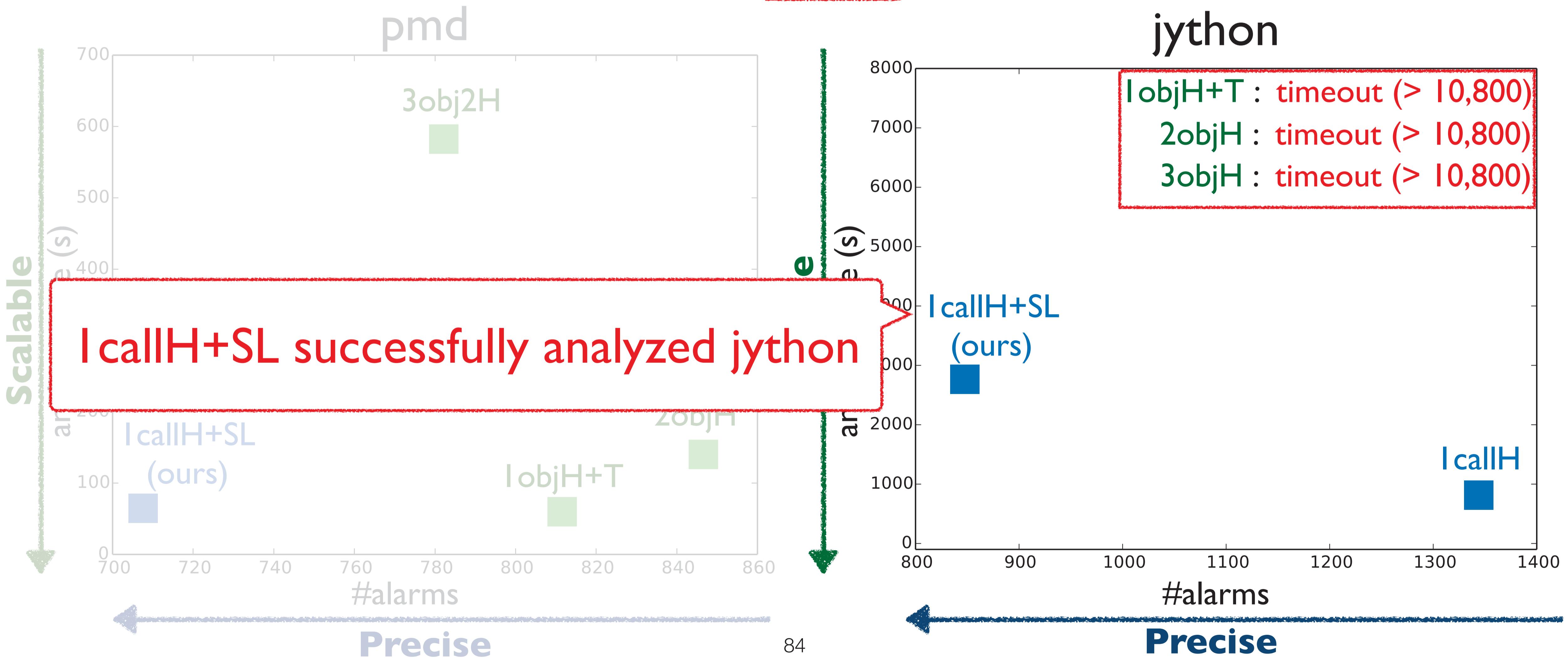
Call-site Sensitivity vs Object Sensitivity

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



Call-site Sensitivity vs Object Sensitivity

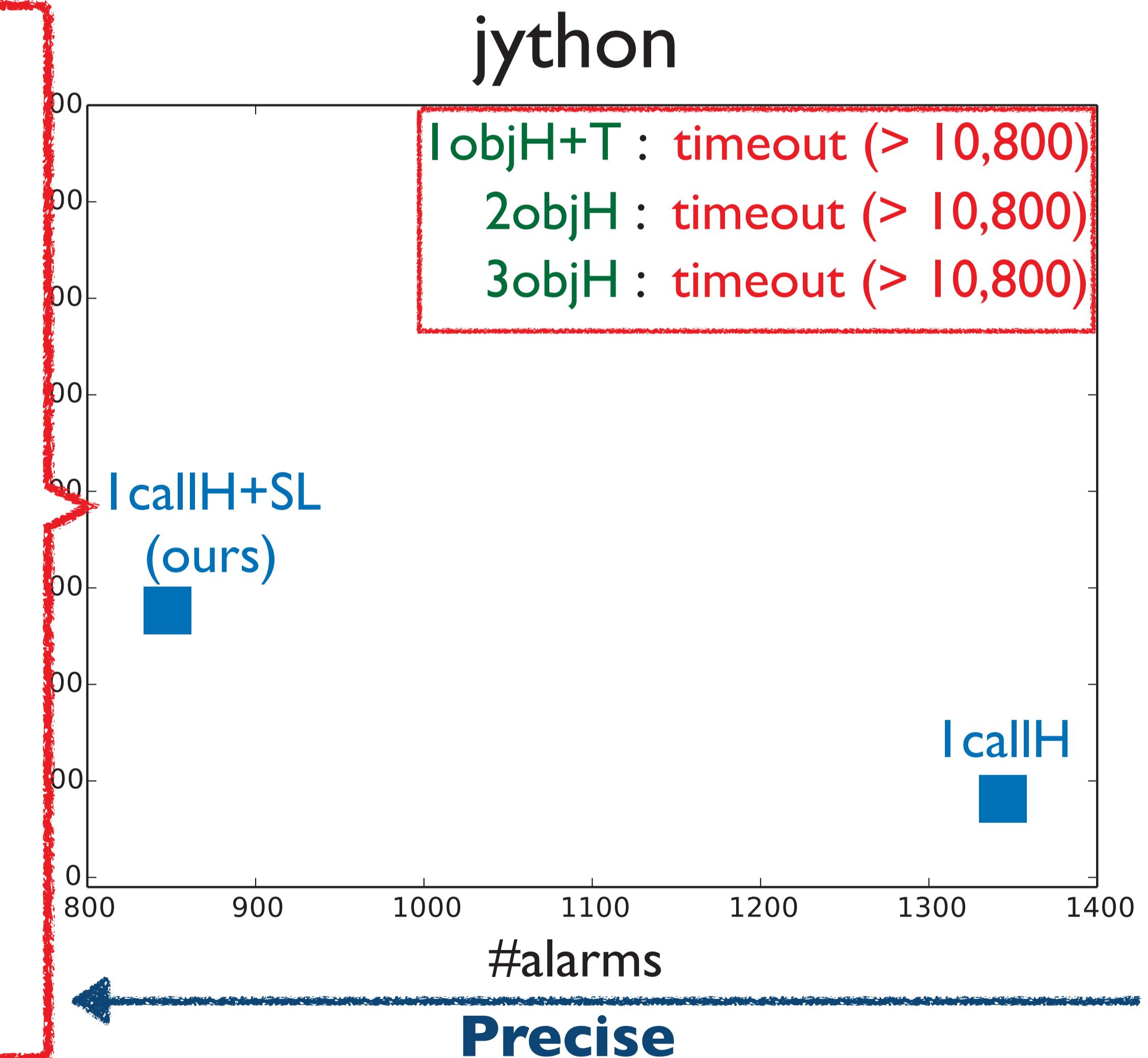
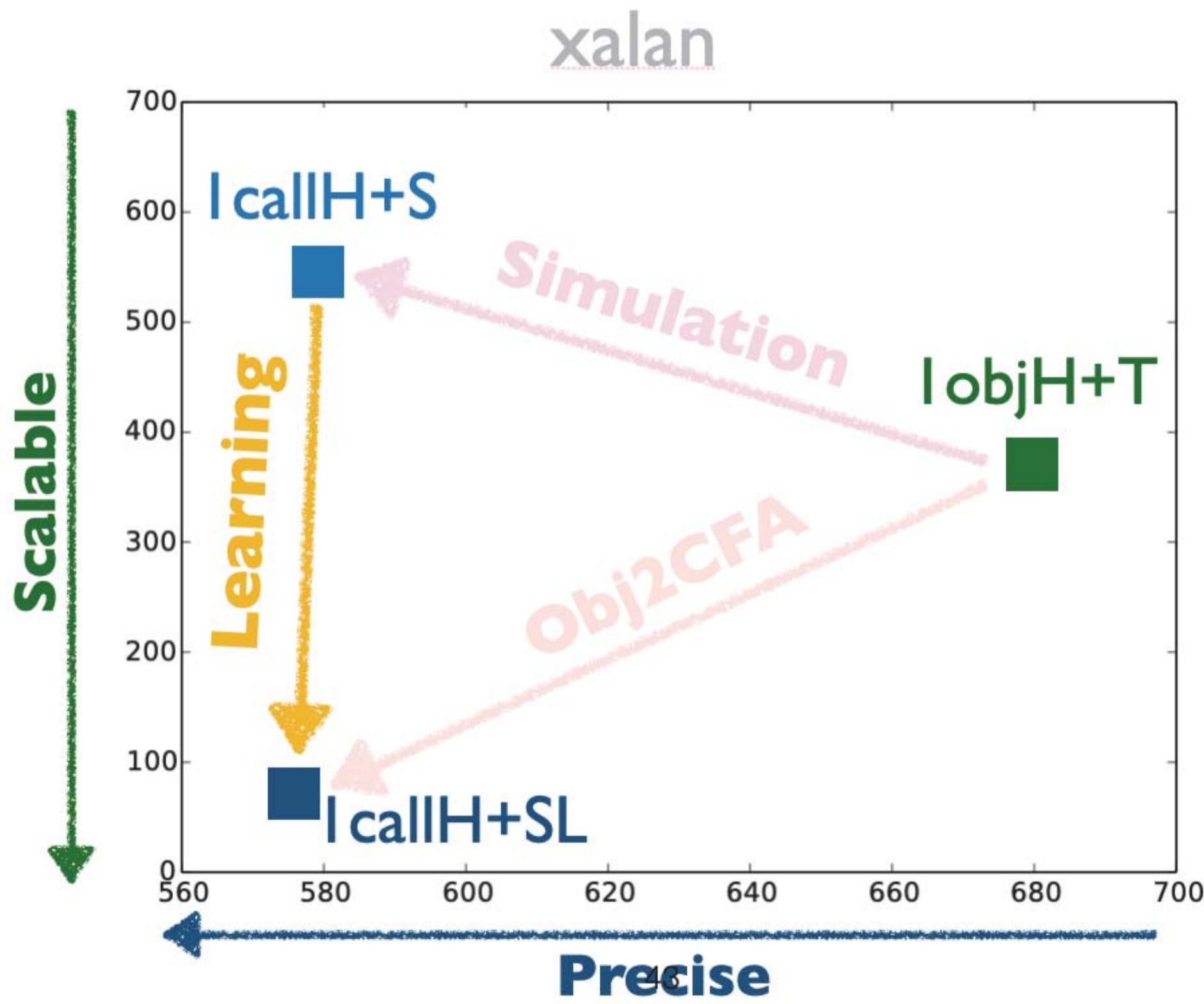
- IcallH+SL (ours) is more precise and scalable than the existing object sensitivities



Call-site Sensitivity vs Object Sensitivity

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

- 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

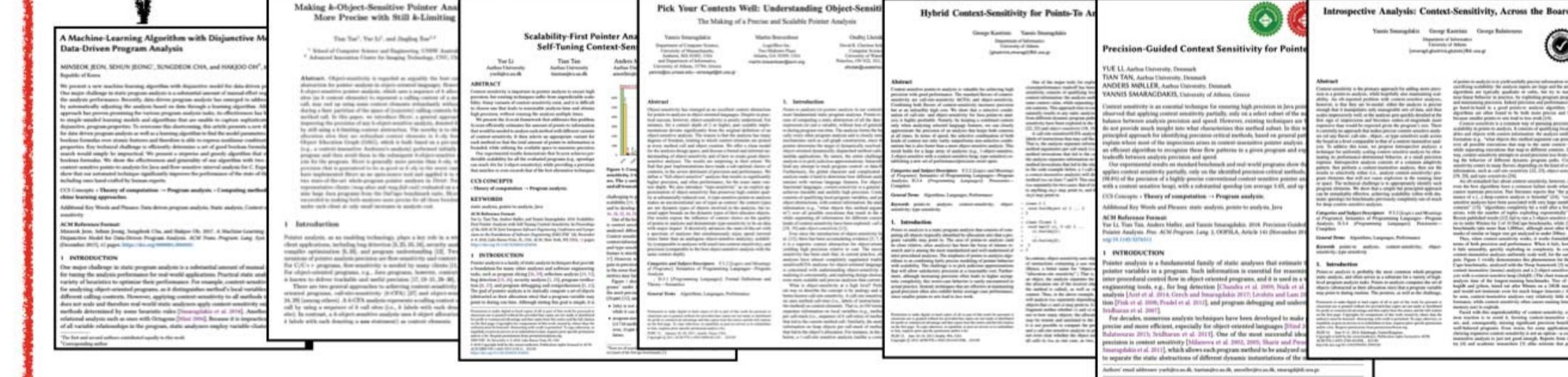
Thank you

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

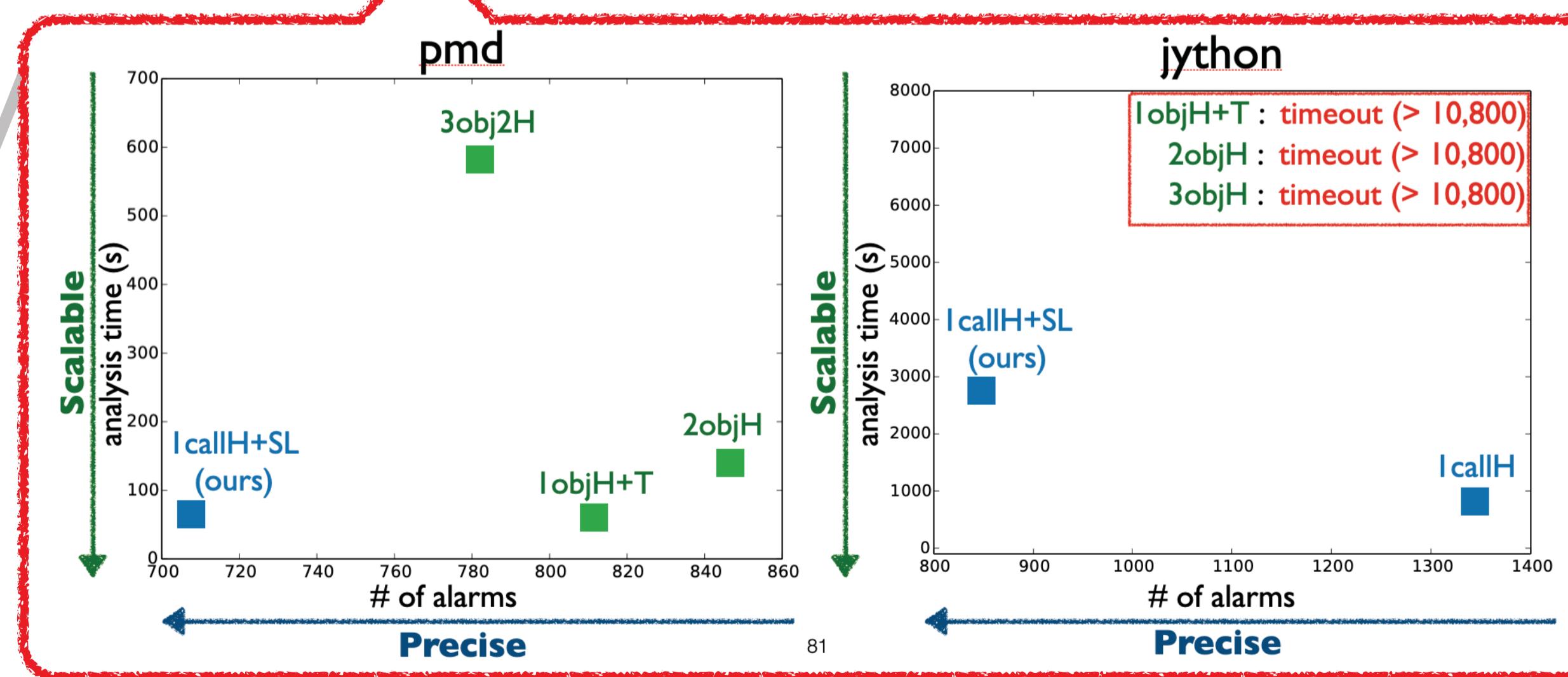
2010

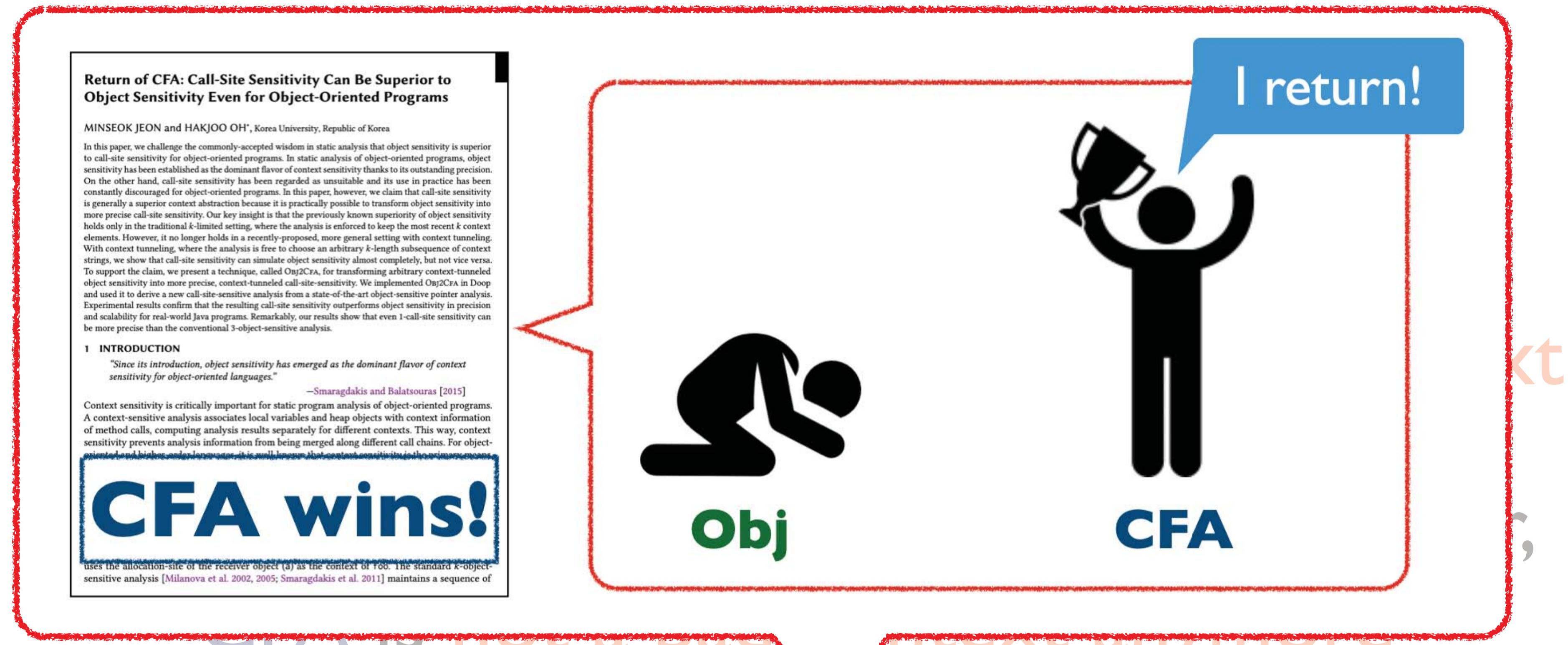
2022

Summary

- Currently, CFA is known as a bad context
- However, if context tunneling is included, CFA is not a bad context anymore

- We can do better from now on





- We need to reconsider CFA from now on

Thank you