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

Minseok Jeon and Hakjoo Oh

KOREA UNIVERSITY

SW재난연구센터 workshop @ Jeju, Korea

Artifacts Available acm V1.1

Artifacts Evaluated acm Reusable V1.1

Two major camps

A: Call-Site Sensitivity Can

Object Sensitivity Even for

Object-Oriented Programs

Minseok Jeon and Hakjoo Oh

KOREA UNIVERSITY

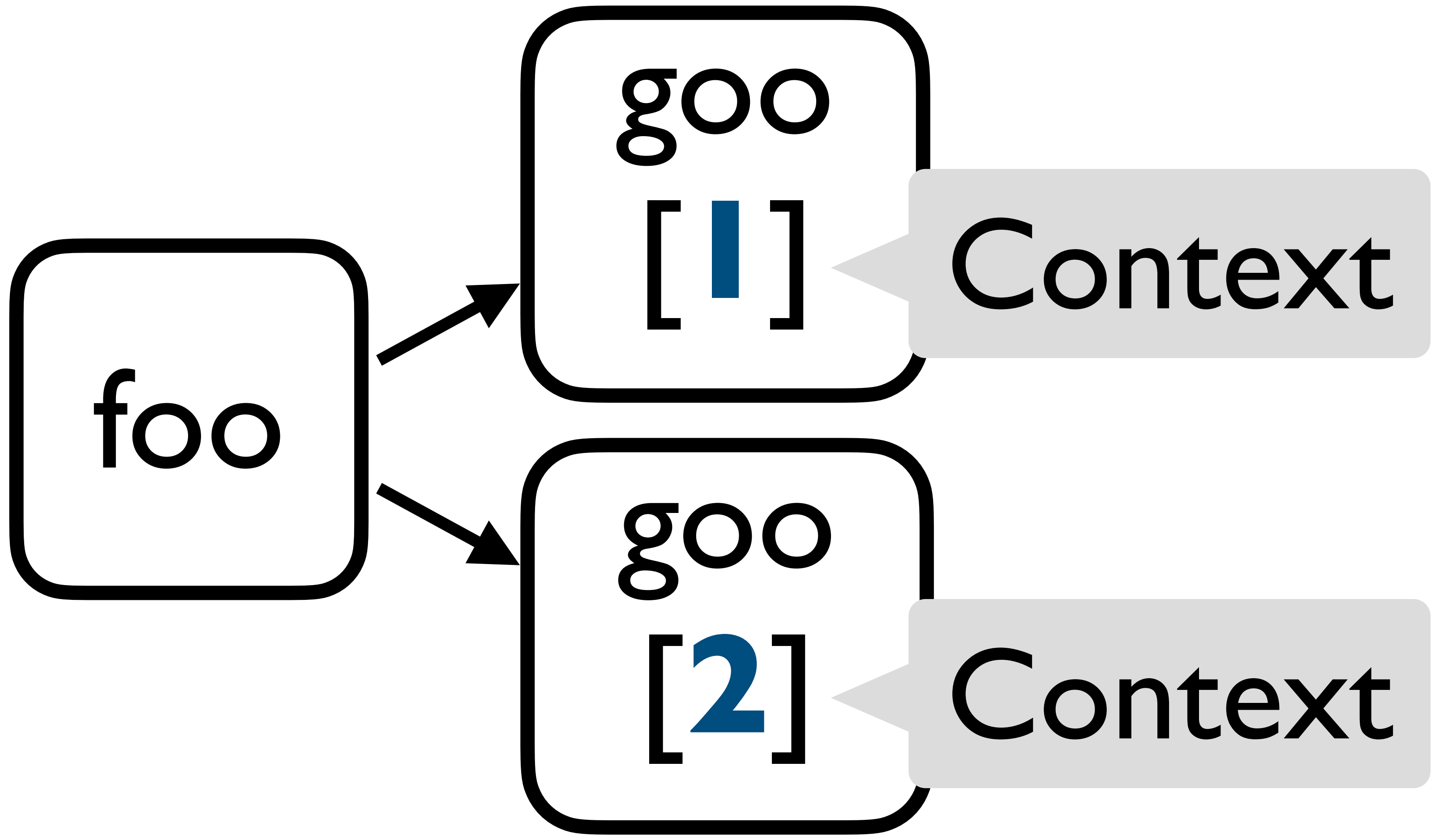
SW재난연구센터 workshop @ Jeju, Korea

Call-site Sensitivity vs Object Sensitivity

Call-site sensitivity was born in 1981

- Considers "**Where**"

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



Call graph

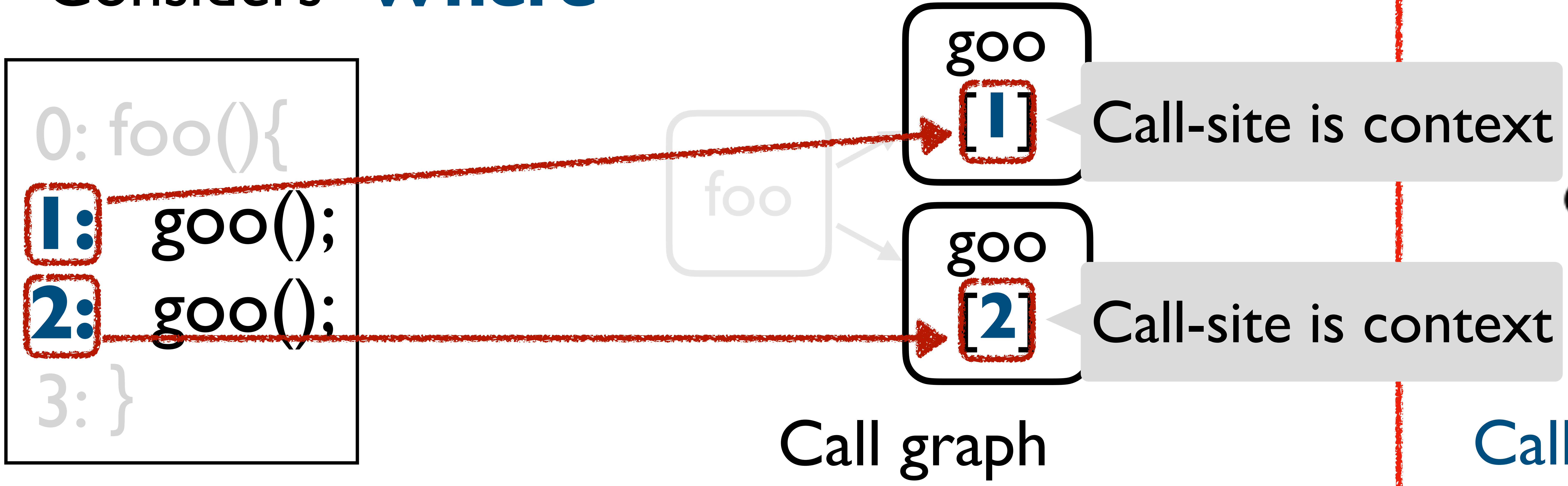
Call-site sensitivity

1981
2002
2010
2022

Call-site Sensitivity vs Object Sensitivity

Call-site sensitivity was born in 1981

- Considers "**Where**"



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

Call graph

Where is it called from?

Call-site sensitivity

1981 2002 2010 2022

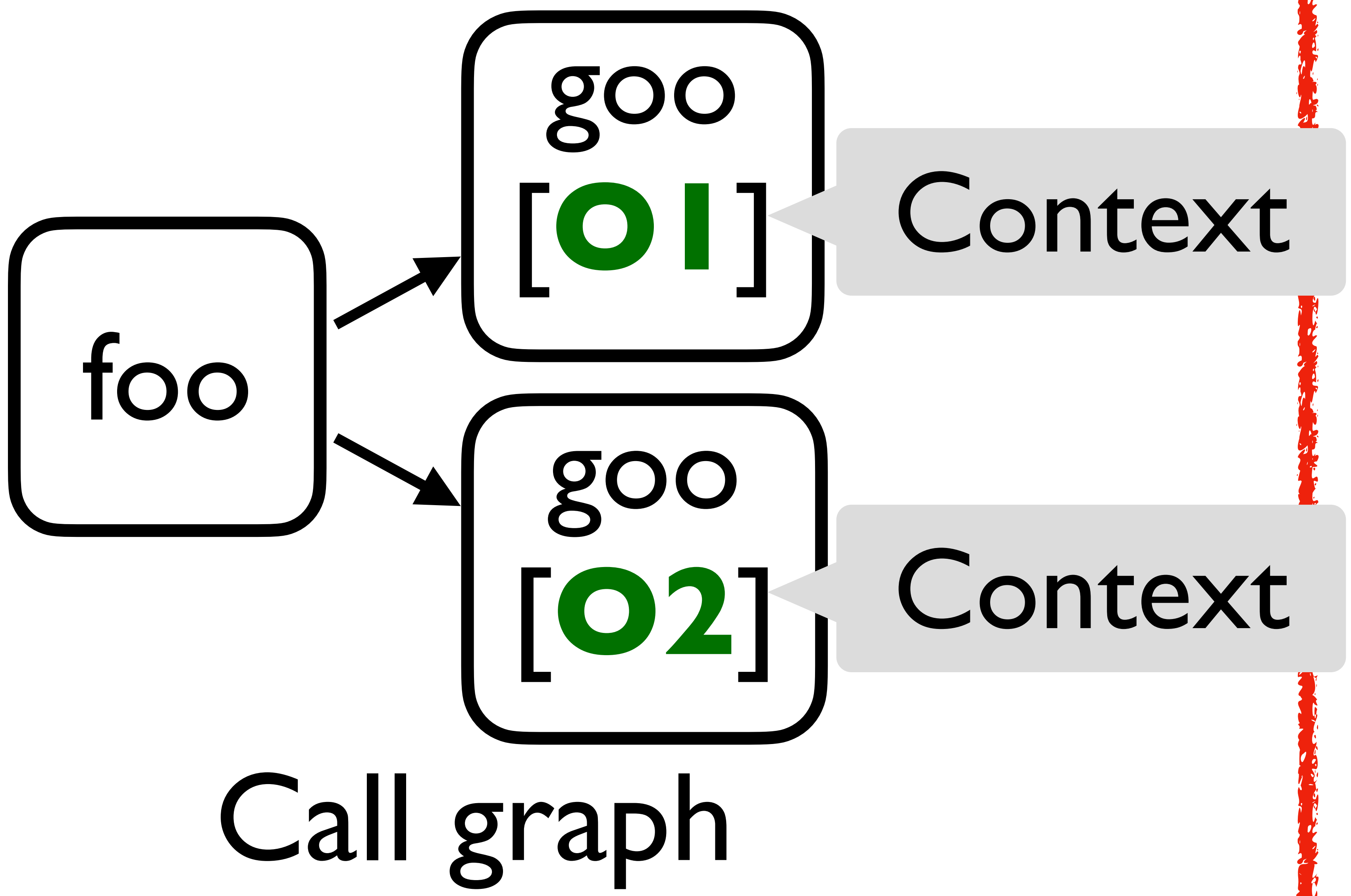
Call-site Sensitivity vs Object Sensitivity

Object sensitivity appeared in 2002

- Considers “**What**”

O1 or **O2**

```
foo(p){
   p.goo();
}
```



Call graph

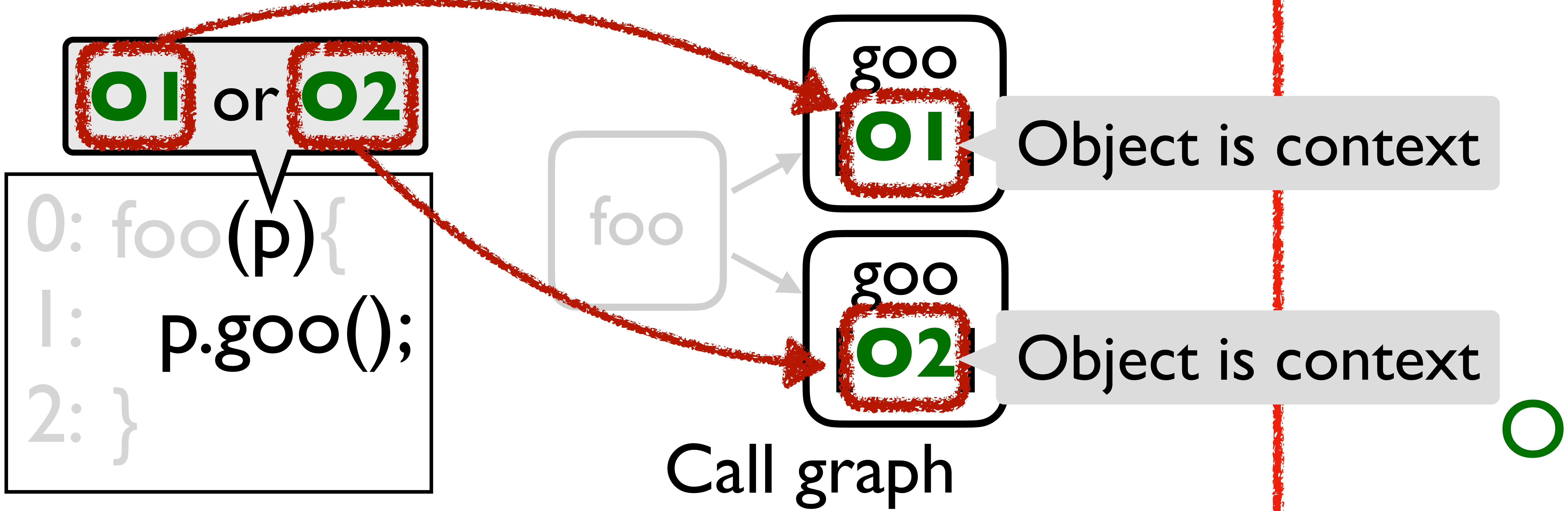
Object sensitivity

1981
2002
2010
2022

Call-site Sensitivity vs Object Sensitivity

Object sensitivity appeared in 2002

- Considers "**What**"



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

What is it called with?

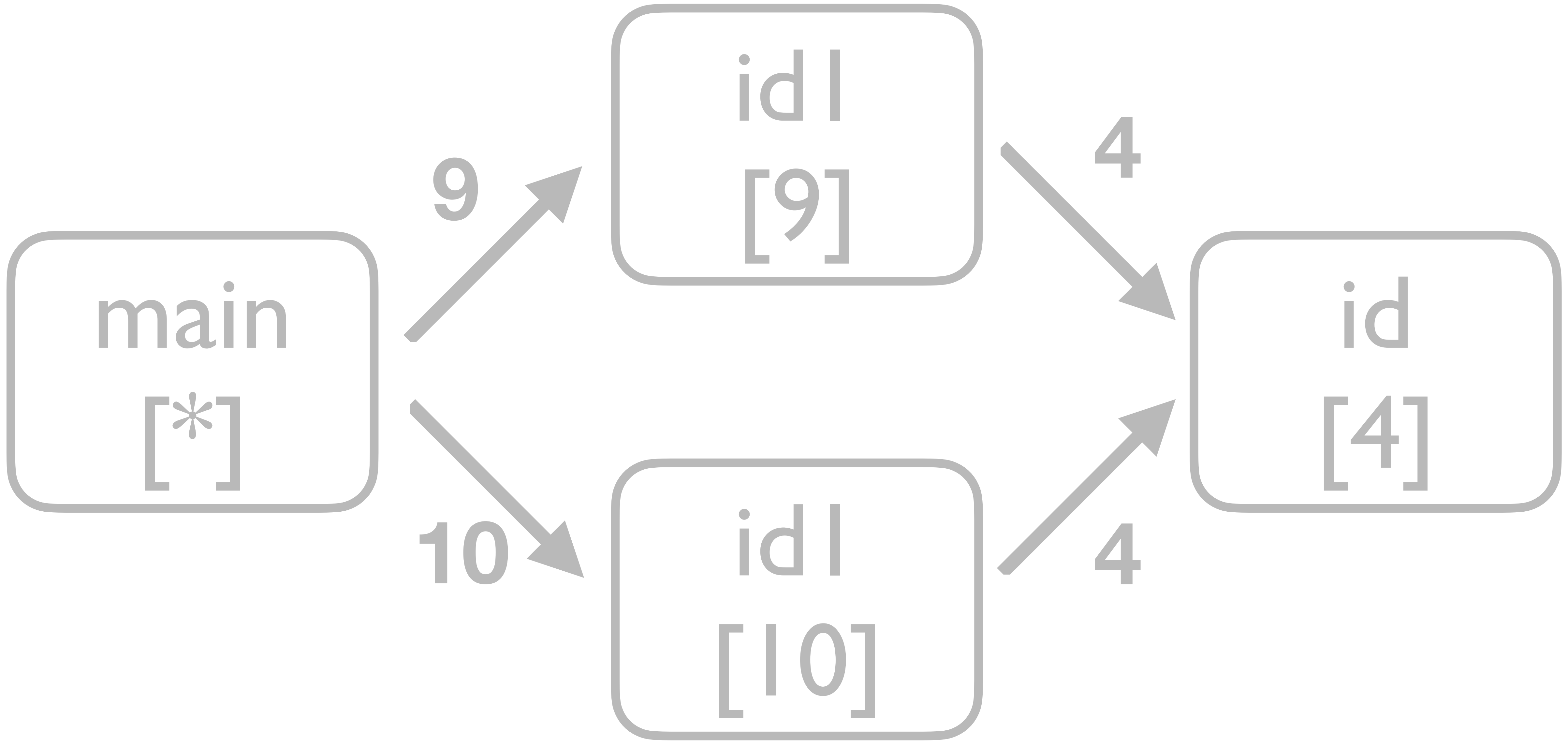
Object sensitivity

1981
2002
2010
2022

Call-site Sensitivity vs Object Sensitivity

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

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



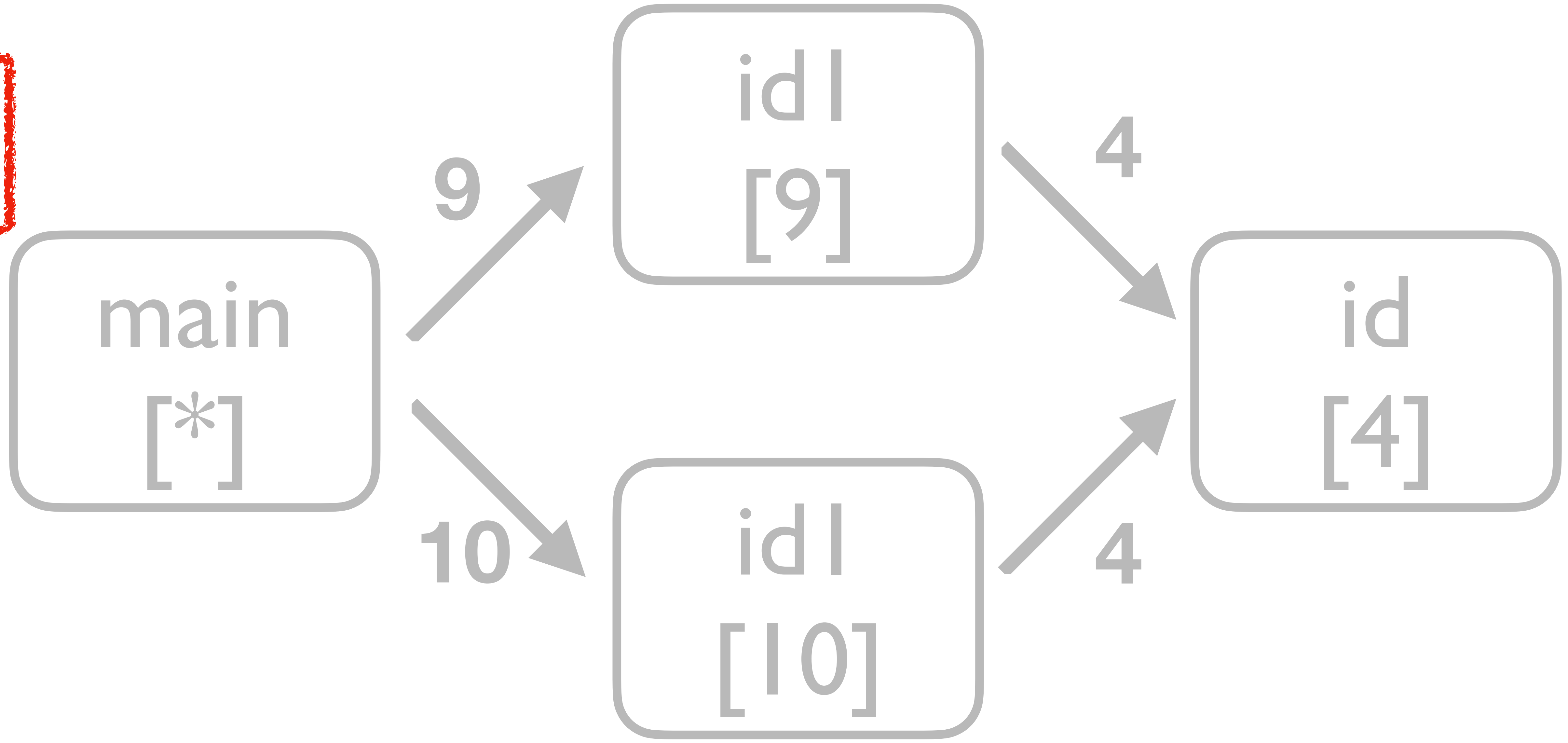
Call-graph of 1-CFA

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



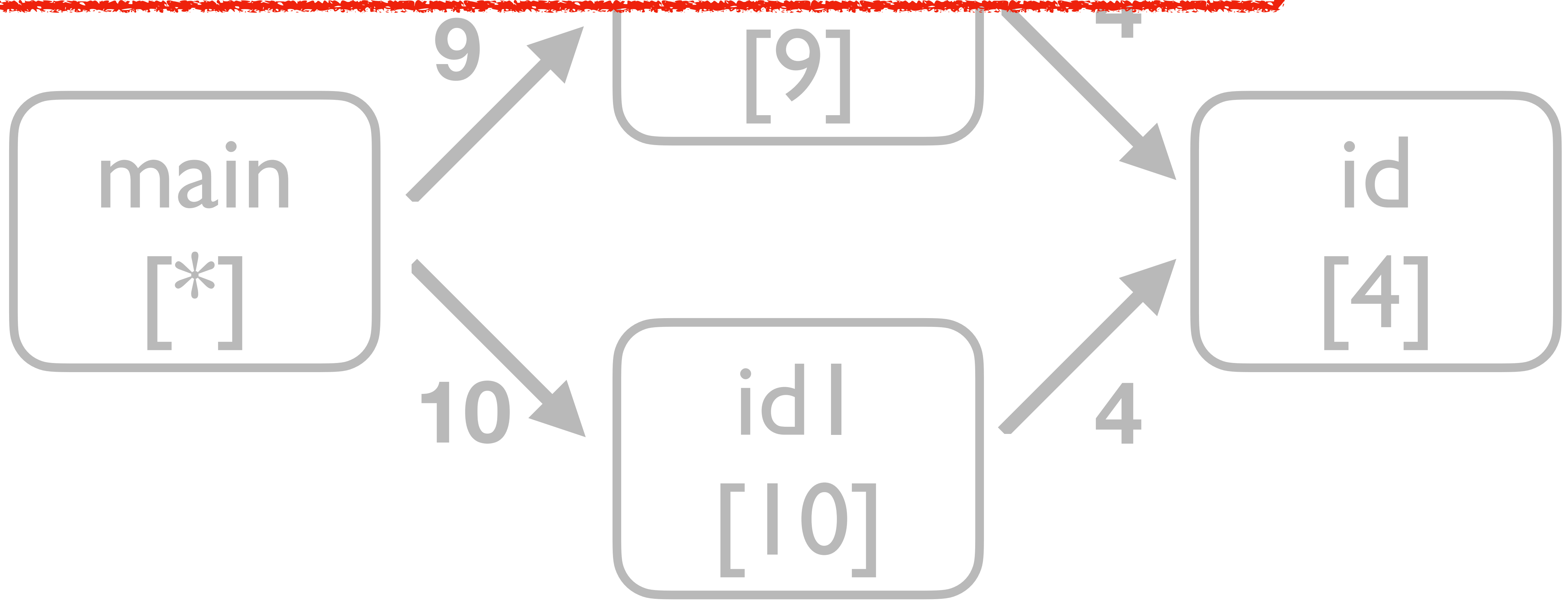
Call-graph of 1-CFA

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: }
```

Also an identity function implemented with id

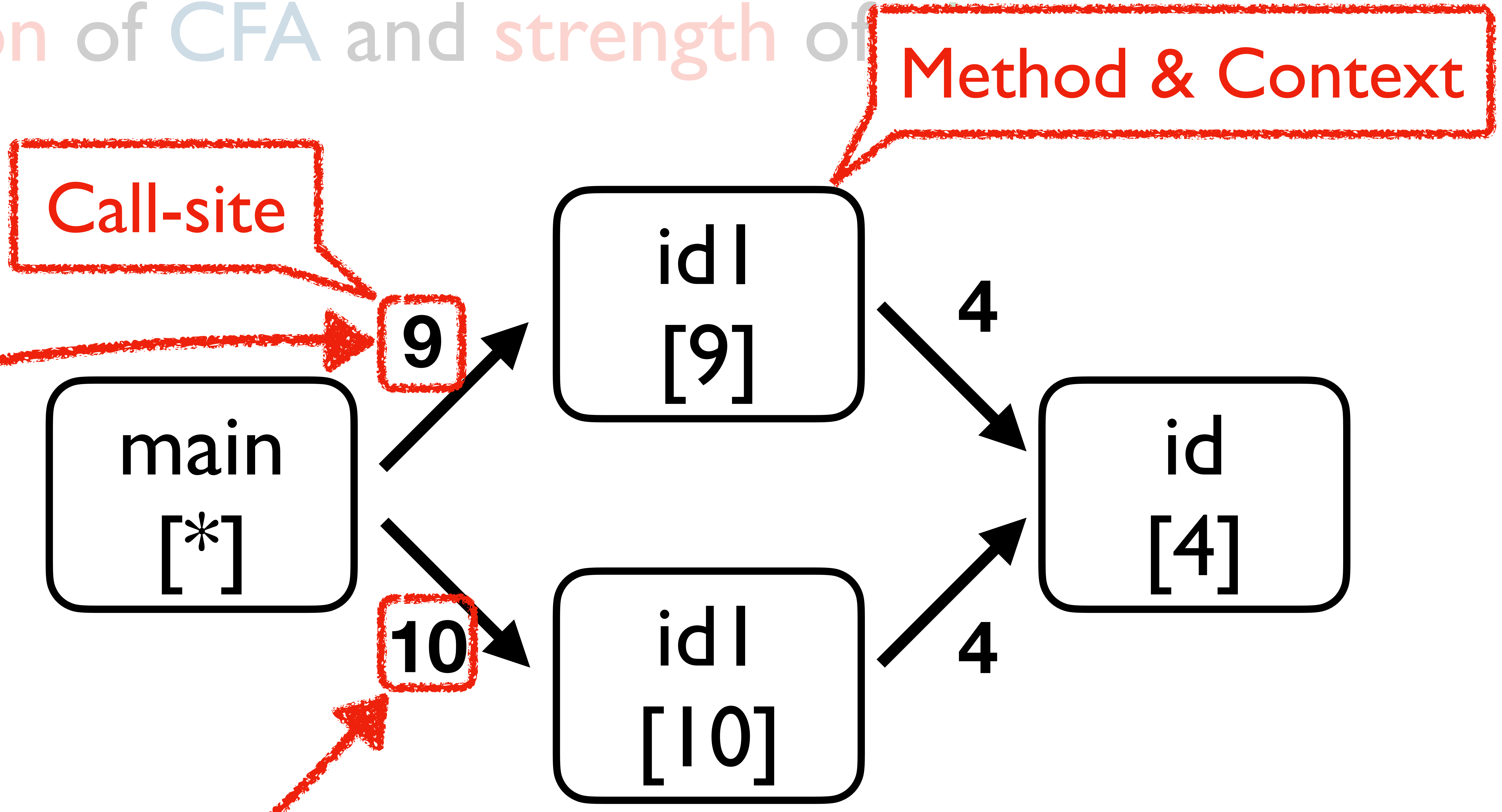


Call-graph of 1-CFA

Call-site Sensitivity vs Object Sensitivity

- An example shows the limitation of CFA and strength of

```
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: }
```



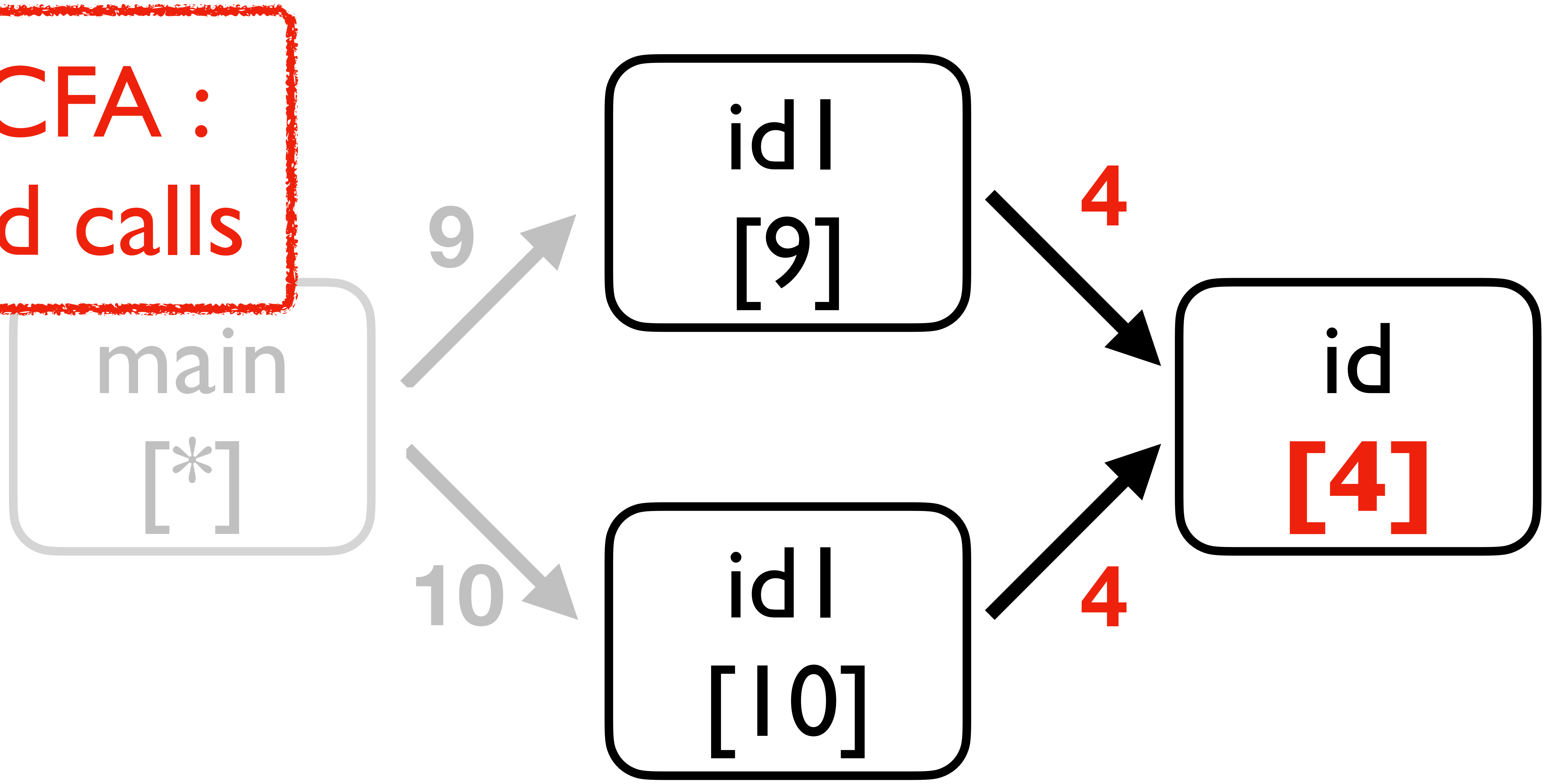
Call-graph of 1-CFA

Call-site Sensitivity vs Object Sensitivity

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

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

Limitation of CFA :
Nested method calls

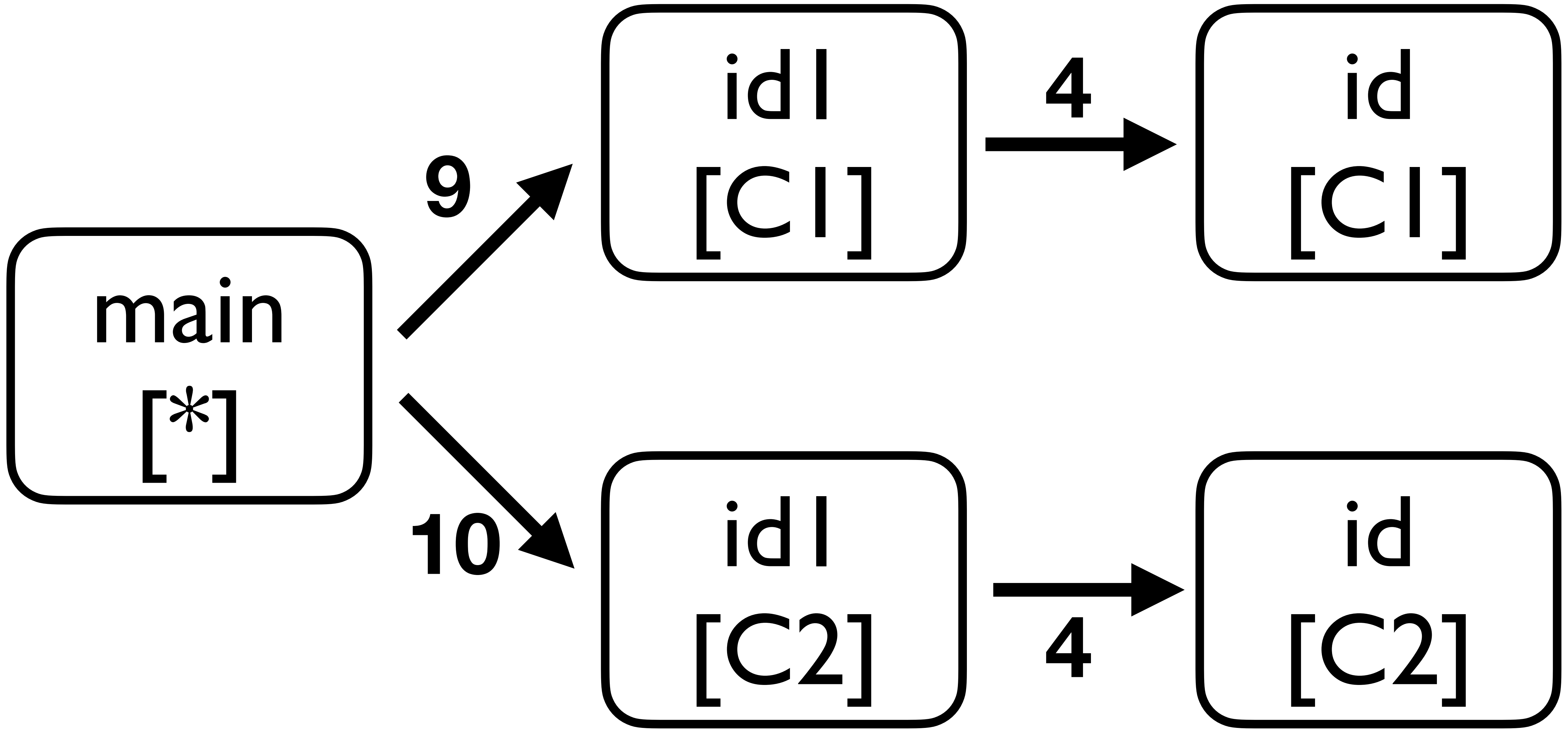


Call-graph of 1-CFA

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: }
```



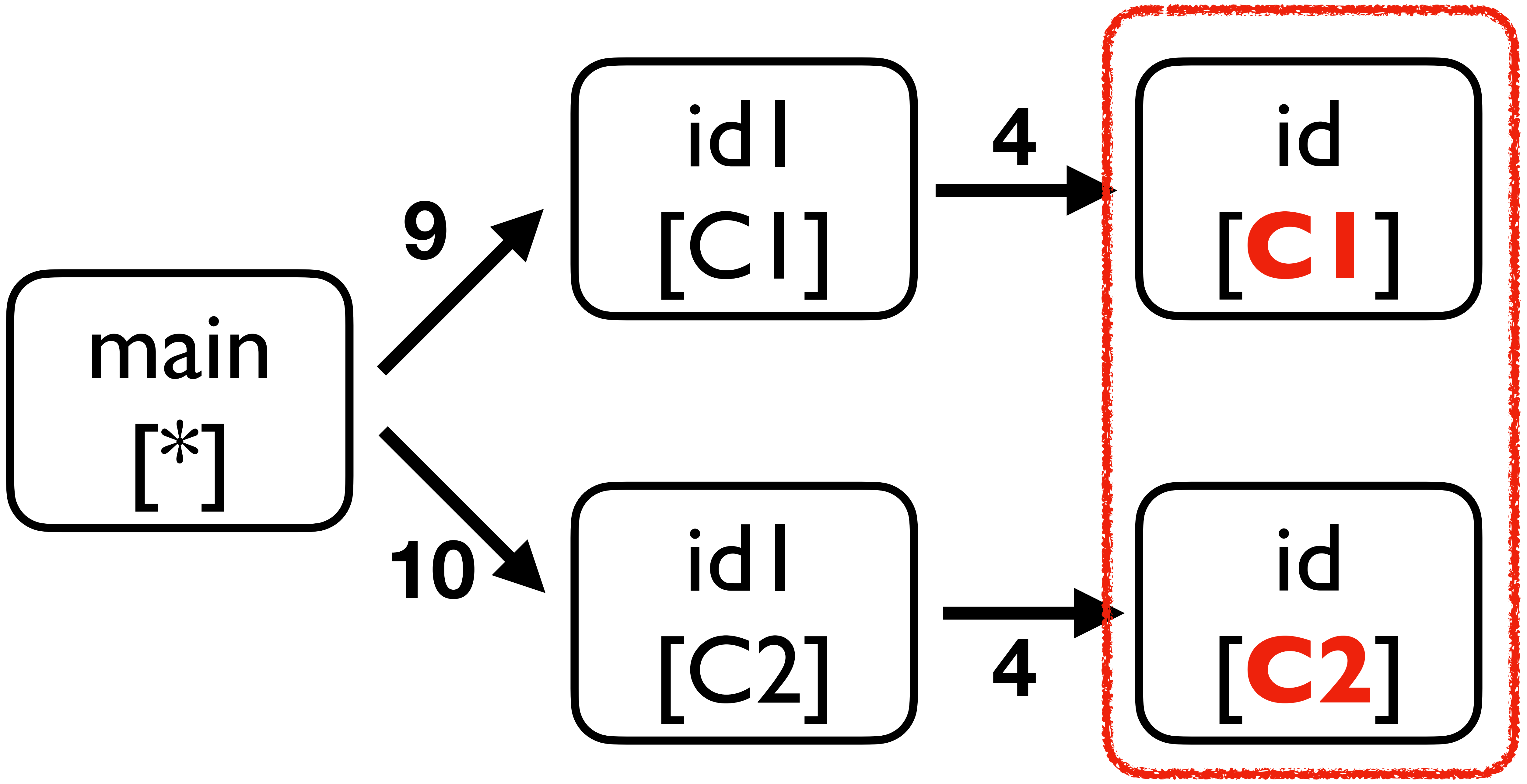
Call-graph of 1-Obj

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: }
```

C1 or C2

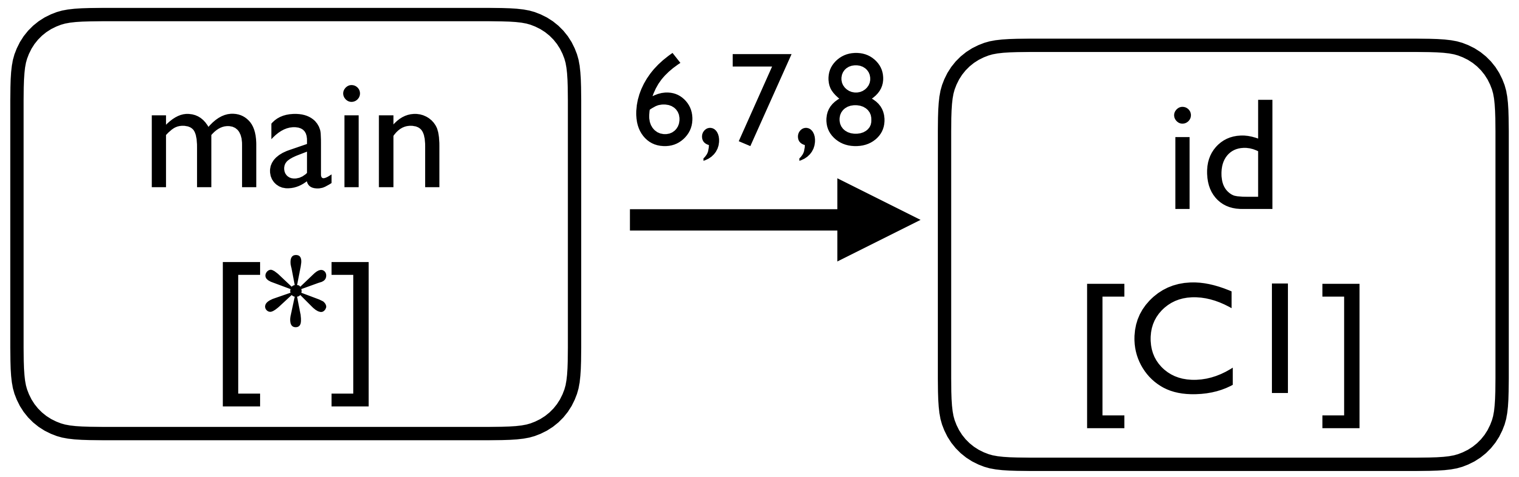


Call-graph of 1-Obj

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:   c1 = new C();//C1
6:   a = (A) c1.id(new A());//query1
7:   b = (B) c1.id(new B());//query2
8:   c = (B) c1.id(new C());//query3
9: }
```

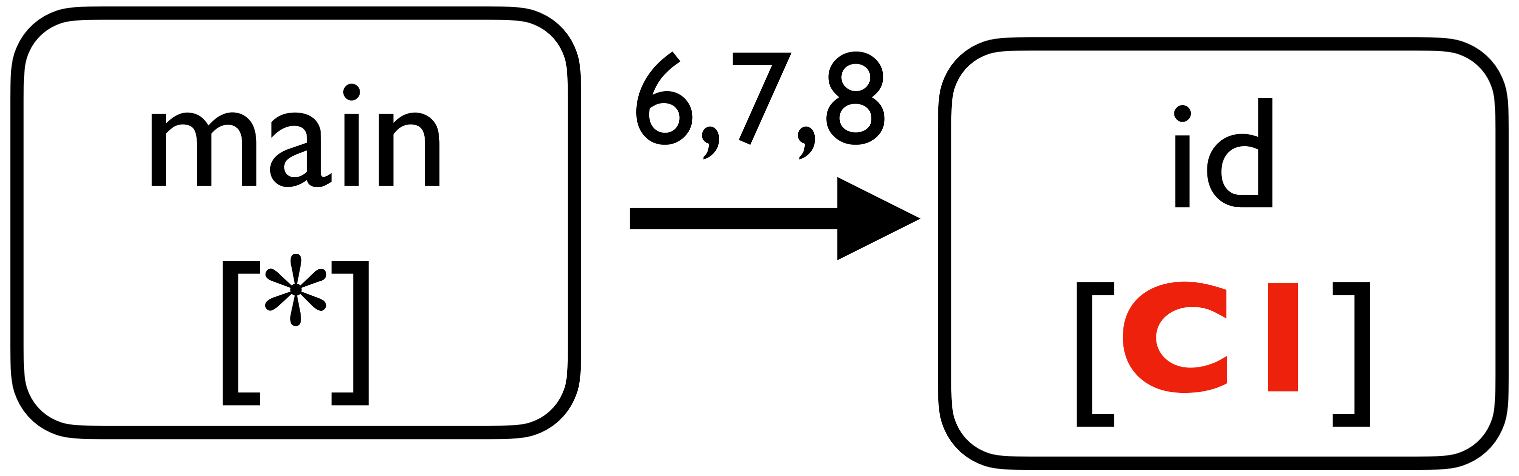


Call-graph of 1-Obj

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:   c1 = new C();//C1
6:   a = (A) c1.id(new A());//query1
7:   b = (B) c1.id(new B());//query2
8:   c = (B) c1.id(new C());//query3
9: }
```



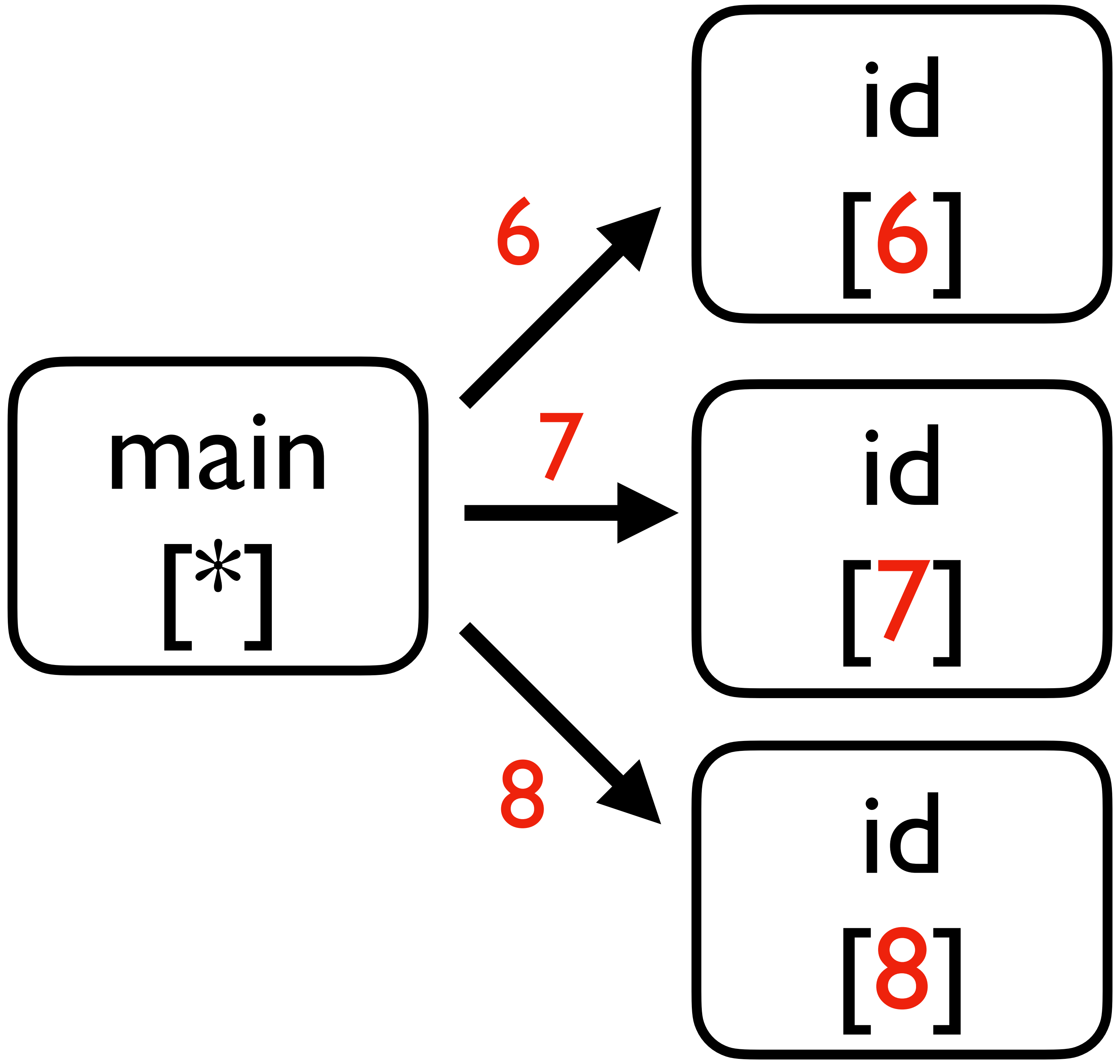
Call-graph of 1-Obj

The three method calls share the same receiver object C1

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:   c1 = new C();//C1
6:   a = (A) c1.id(new A());//query1
7:   b = (B) c1.id(new B());//query2
8:   c = (C) c1.id(new C());//query3
9: }
```



Call-graph of 1-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

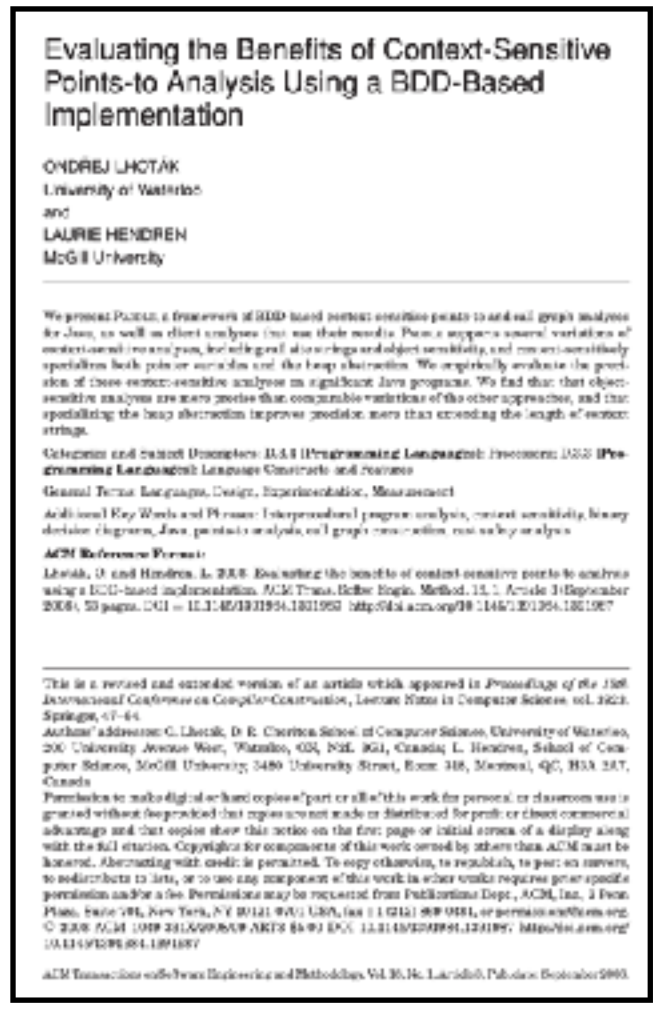
Parameterized Object Sensitivity for Points-to Analysis for Java

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

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

Ondřej Lhoták[1,2] and Laurie Hendren[2]

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



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

ONDŘEJ LHOTÁK
University of Waterloo
and
LAURIE HENDREN
McGill University

Strictly Declarative Specification of Sophisticated Points-to Analyses

Martin Bravenboer   Yannis Smaragdakis
Department of Computer Science
University of Massachusetts, Amherst
Amherst, MA 01003, USA

...

Obj vs CFA

1981
2002
2010
2022

Call-site Sensitivity vs Object Sensitivity

- Object Sensitivity outperformed call-site sensitivity

Call-site Sensitivity vs Object Sensitivity

Parameterized Object Sensitivity for Points-to Analysis for Java

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

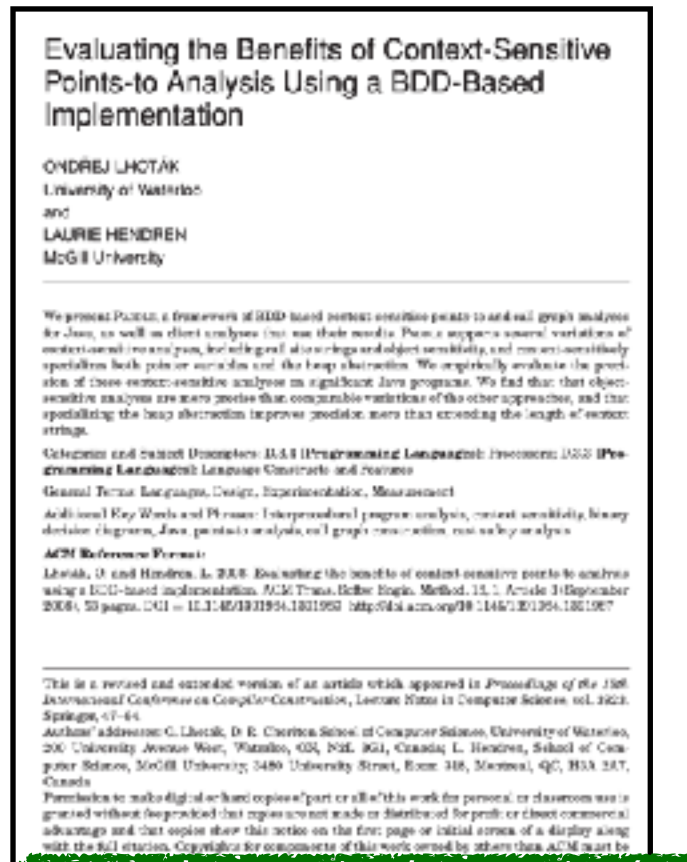
Obj wins

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

Ondřej Lhoták[1, 2] and Laurie Hendren[2]

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

Obj wins


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

ONDŘEJ LHOTÁK
University of Waterloo
and
LAURIE HENDREN
McGill University

Obj wins

Strictly Declarative Specification of Sophisticated Points-to Analyses

Martin Bravenboer
Yannis Smaragdakis
Department of Computer Science
University of Massachusetts, Amherst
Amherst, MA 01003, USA

Obj wins

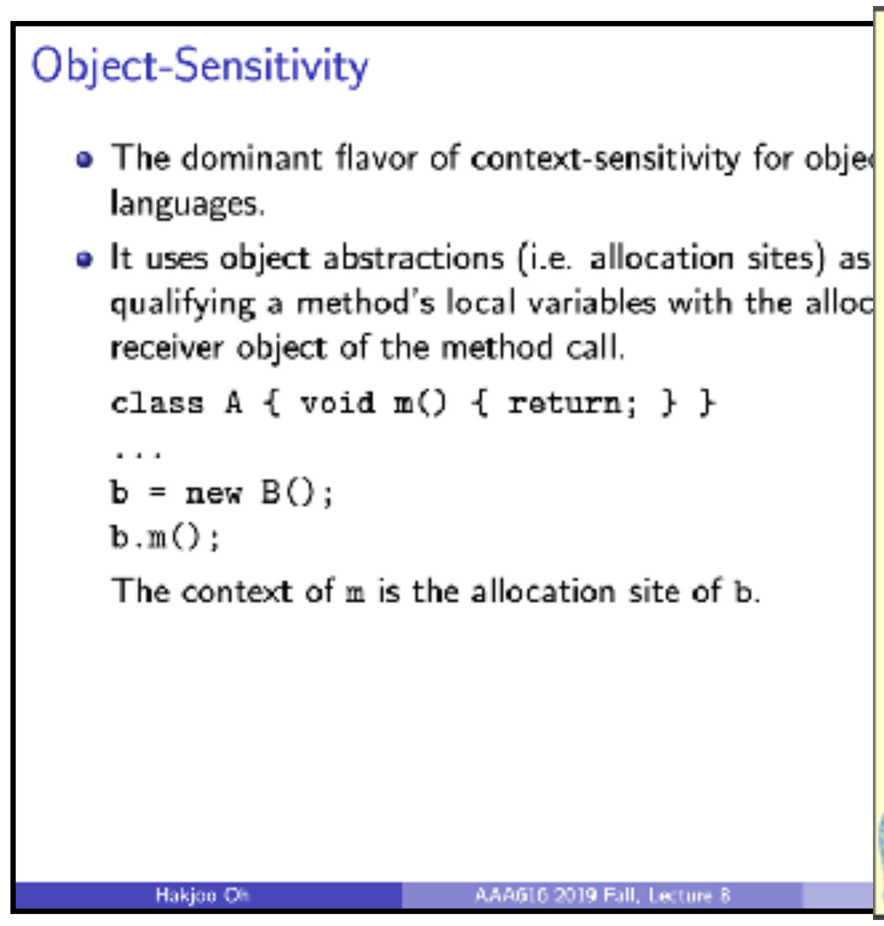
…

Obj **CFA**

1981
2002
2010
2022

Call-site Sensitivity vs Object Sensitivity

- Lectures have taught the superiority of object sensitivity



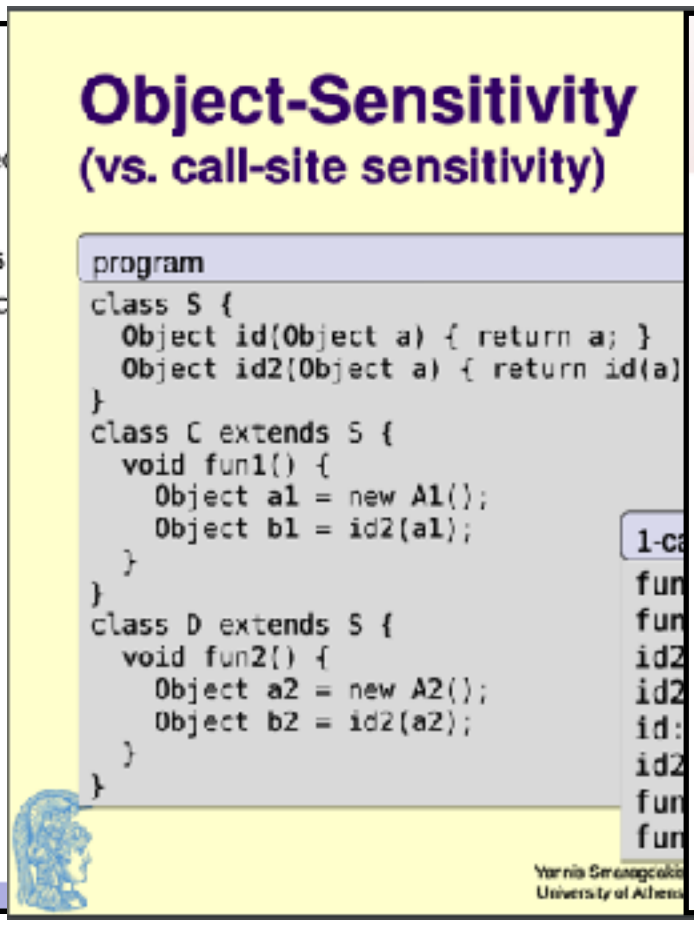
Object-Sensitivity

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

Hakjoo Oh AAA616 2019 Fall, Lecture 8



Object-Sensitivity
(vs. call-site sensitivity)

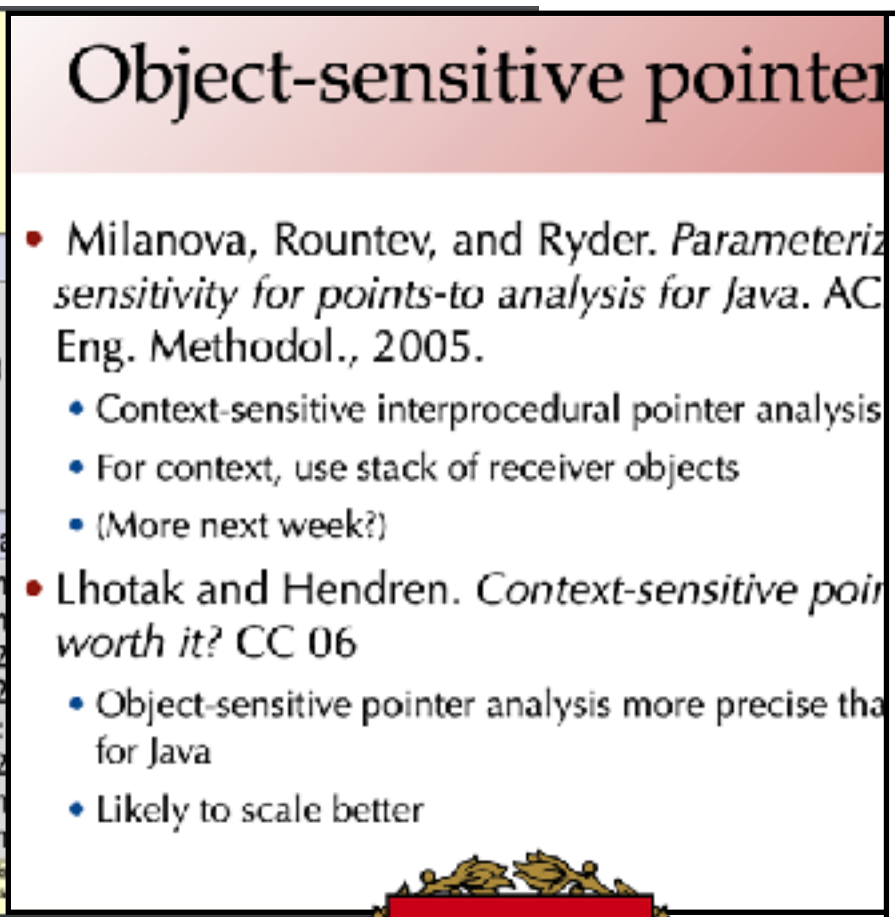
program

```
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);
  }
}
```

1-ca

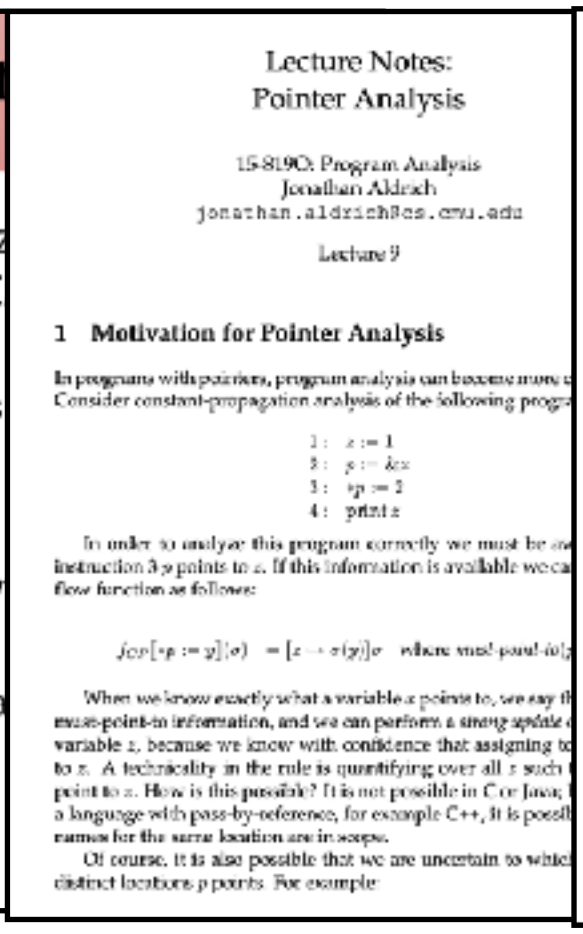
```
fun
fun
id2
id2
id:
id2
fun
fun
```

Yannis Smaragdakis
University of Athens



Object-sensitive pointer

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



Lecture Notes:
Pointer Analysis

15-819O: Program Analysis
Jonathan Aldrich
jonathan.aldrich@cs.cmu.edu

Lecture 9

1 Motivation for Pointer Analysis

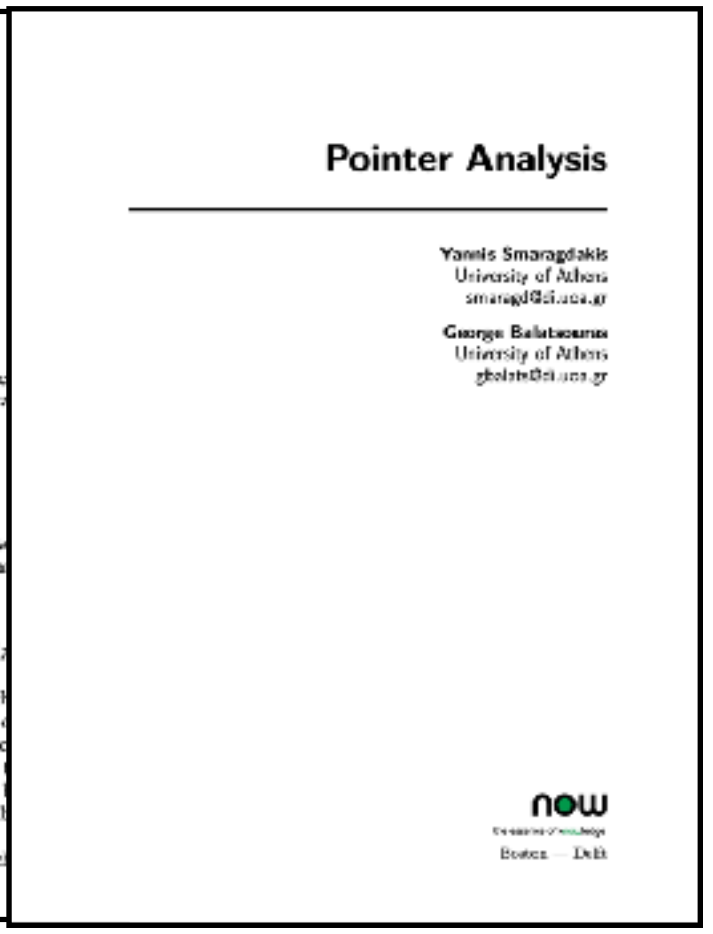
In programs with pointers, program analysis can become more c
Consider constant-propagation analysis of the following progra

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

In order to analyze this program correctly we must be aw
instruction 3 p points to z. If this information is available we ca
flow function as follows:

$f_{CP}[\![*p := y]\!](\sigma) = [z \mapsto \sigma(y)]\sigma$ where must-point-to(

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



Pointer Analysis

Yannis Smaragdakis
University of Athens
smaragd@di.uoa.gr

George Balatsouras
University of Athens
gbalats@di.uoa.gr

now
the essence of knowledge
Boston — Delft

...

KOREA UNIVERSITY

National and Kapodistrian University of Athens

VE RI TAS HARVARD

Carnegie Mellon University

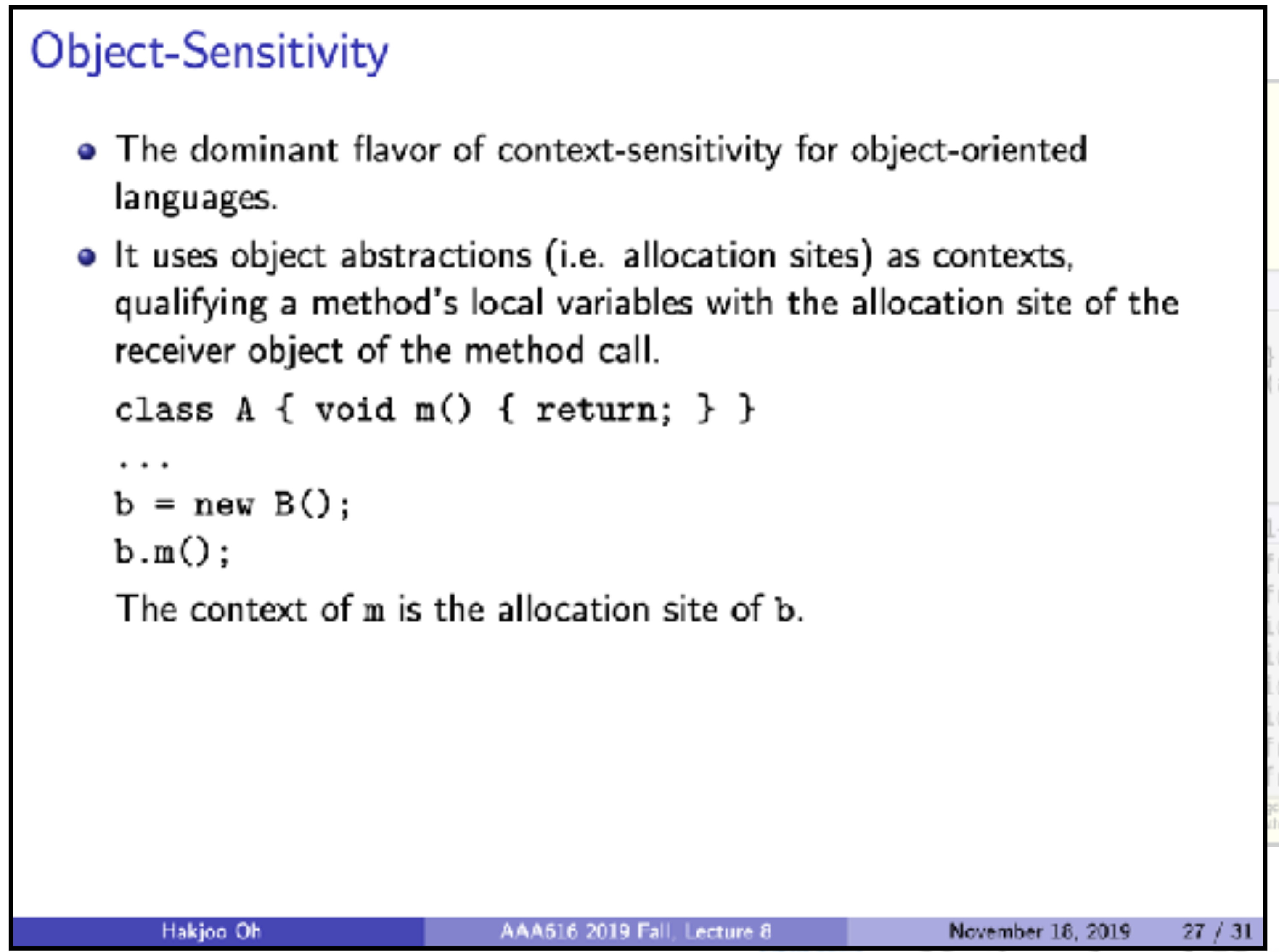
now the essence of knowledge

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.

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

KOREA UNIVERSITY

Object-sensitive pointer analysis
Milanova, Rountev, and Ryder. Parameterized object sensitivity for points-to analysis for Java. ACM Trans. Softw. Eng.
Lecture Notes: Pointer Analysis
15-819O: Program Analysis
Jonathan Aldrich
Lecture 9
Pointer Analysis
Yannis Smaragdakis
University of Athens
National and Kapodistrian University of Athens
HARVARD
Carnegie Mellon University
now the essence of knowledge

I was also taught like that

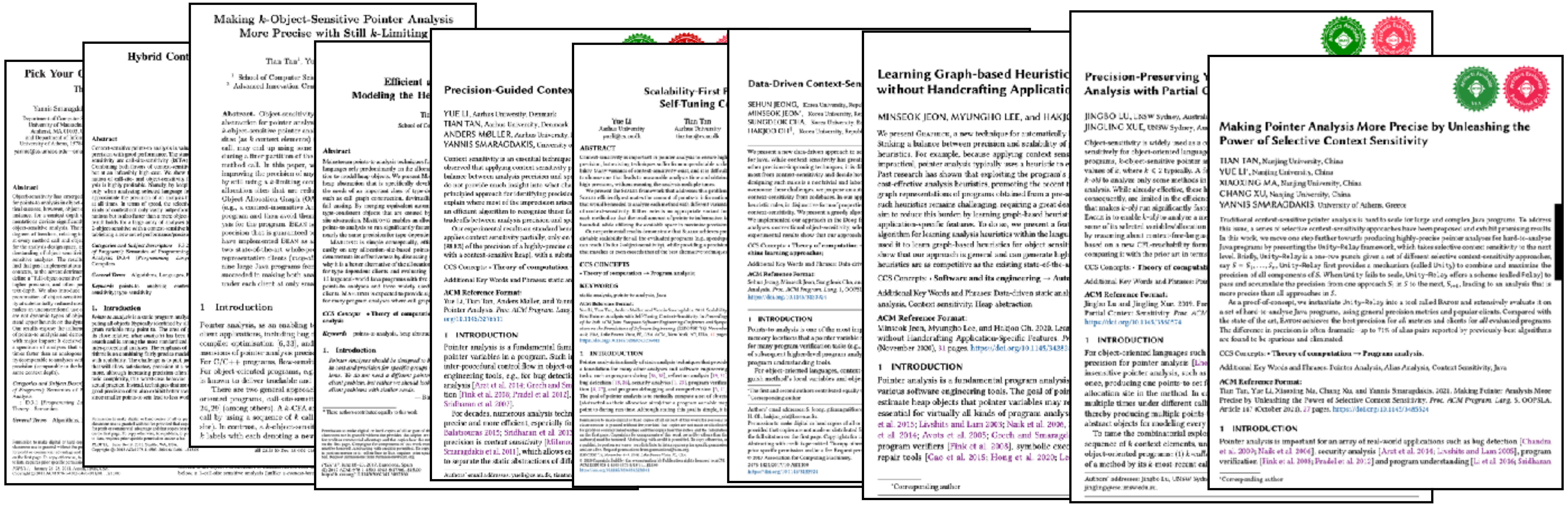
Obj

1981
2002
2010
2022

Call-site Sensitivity vs Object Sensitivity

- Researches focused on improving Object Sensitivity

Researches on Object Sensitivity



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

Precision-Guided Contex

Scalability-First P Self-Tuning C

Data-Driven Context-Sen

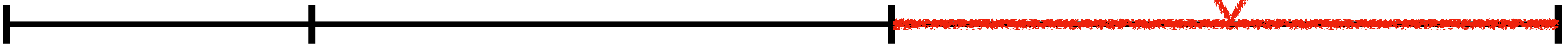
Learning Graph-based Heuristic without Handcrafting Applicatio

Precision-Preserving Y Analysis with Partial C

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

...

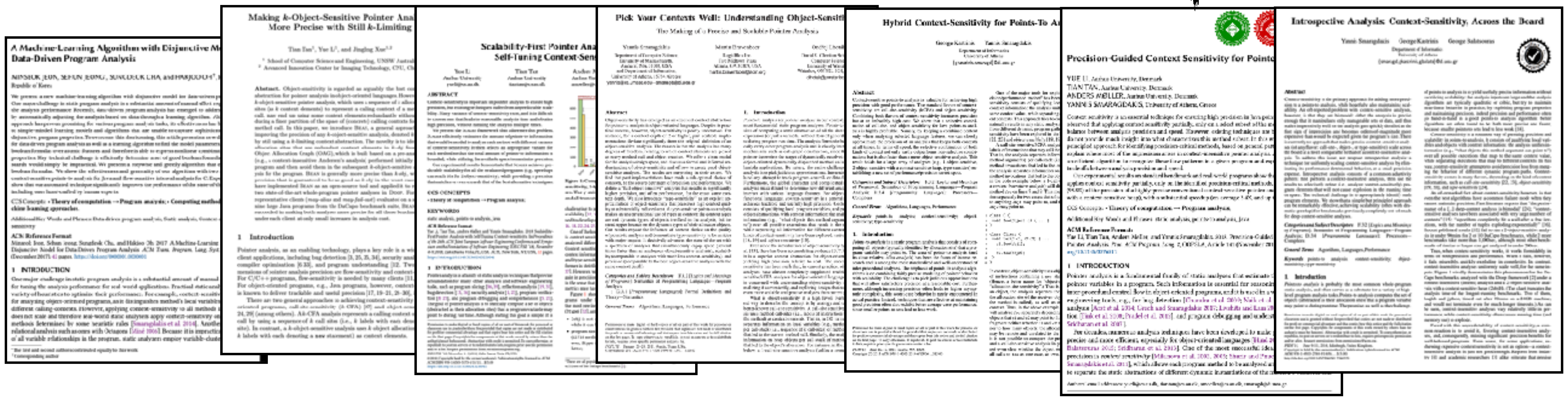
Obj



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 M... Data-Driven Program Analysis
Making k-Object-Sensitive Pointer Ana... More Precise with Still k-Limiting
Scalability-First Pointer Ana... Self-Tuning Context-Sen...
Pick Your Contexts Well: Understanding Object-Sensiti... The Making of a Precise and Scalable Pointer Analysis
Hybrid Context-Sensitivity for Points-To A...
Precision-Guided Context Sensitivity for Pointe...
Introspective Analysis: Context-Sensitivity, Across the Board

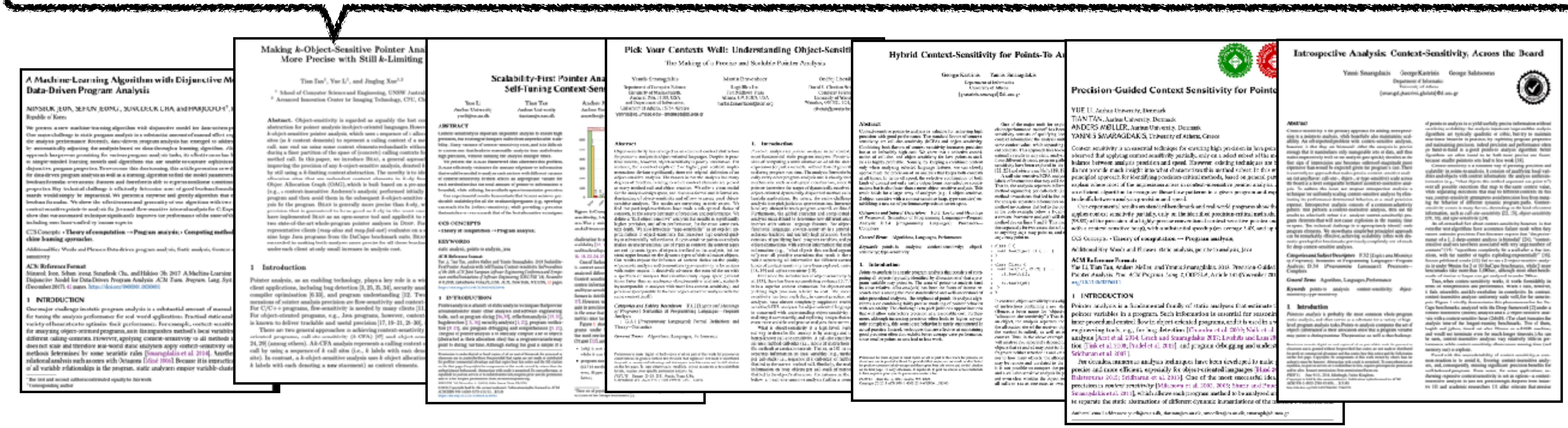
1981
2002
2010
2022

CFA

Call-site Sensitivity vs Object Sensitivity

- Call-site Sensitivity has been ignored

> “We have included 2cs+h to demonstrate the superiority of object sensitivity over call-site sensitivity”
>
> - Tan et al. [2016]


A Machine-Learning Algorithm with Disjunctive M
Data-Driven Program Analysis
Making k-Object-Sensitive Pointer Ana
More Precise with Still k-Limiting
Scalability-First Pointer Ana
Self-Tuning Context-Sen
Pick Your Contexts Well: Understanding Object-Sensiti
The Making of a Precise and Scalable Pointer Analysis
Hybrid Context-Sensitivity for Points-To A
Precision-Guided Context Sensitivity for Pointe
Introspective Analysis: Context-Sensitivity, Across the Board

1981
2002
2010
2022

CFA

Call-site Sensitivity vs Object Sensitivity

- Call-site Sensitivity has been ignored

"… we do not discuss our approach for call-site sensitivity"

- Jeon et al. [2019]

A Machine-Learning Algorithm with Disjunctive M
Data-Driven Program Analysis

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

Scalability-First Pointer Ana
Self-Tuning Context-Sen

Pick Your Contexts Well: Understanding Object-Sensiti

Hybrid Context-Sensitivity for Points-To A

Precision-Guided Context Sensitivity for Pointe

Introspective Analysis: Context-Sensitivity, Across the Board

CFA

1981
2002
2010
2022

Call-site Sensitivity vs Object Sensitivity

- Call-site Sensitivity has been ignored

> "… we do not discuss our approach for call-site sensitivity"
>
> \- Jeon et al. [2019]

I also strongly dismissed call-site sensitivity

CFA

1981
2002
2010
2022

Call-site Sensitivity vs Object Sensitivity

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

1981

2002

2010

2022

Call-site Sensitivity vs Object Sensitivity

A technique context tunneling is proposed

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

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

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

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

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

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

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

ACM Reference Format:
Minseok Jeon, Sehun Jeong, and Hakjoo Oh. 2018. Precise and Scalable Points-to Analysis via Data-Driven Context Tunneling. *Proc. ACM Program. Lang.* 2, OOPSLA, Article 140 (November 2018), 30 pages. https://doi.org/10.1145/3276510

*Corresponding author

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

2475-1421/2018/11-ART140
https://doi.org/10.1145/3276510

Jeon et al. [2018]

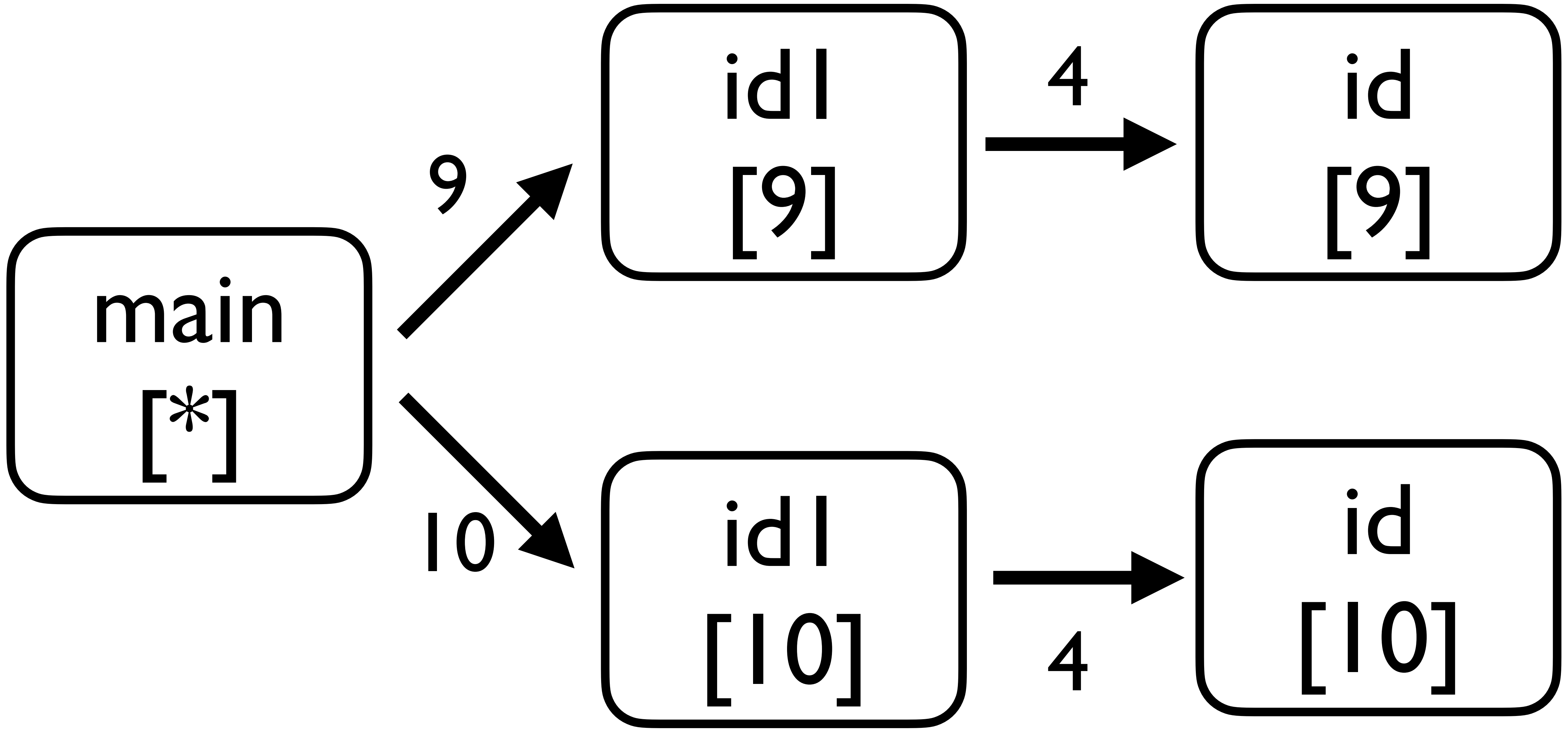
Context tunneling can improve both call-site sensitivity and object sensitivity

1981
2002
2010
2018
2022

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: }
```

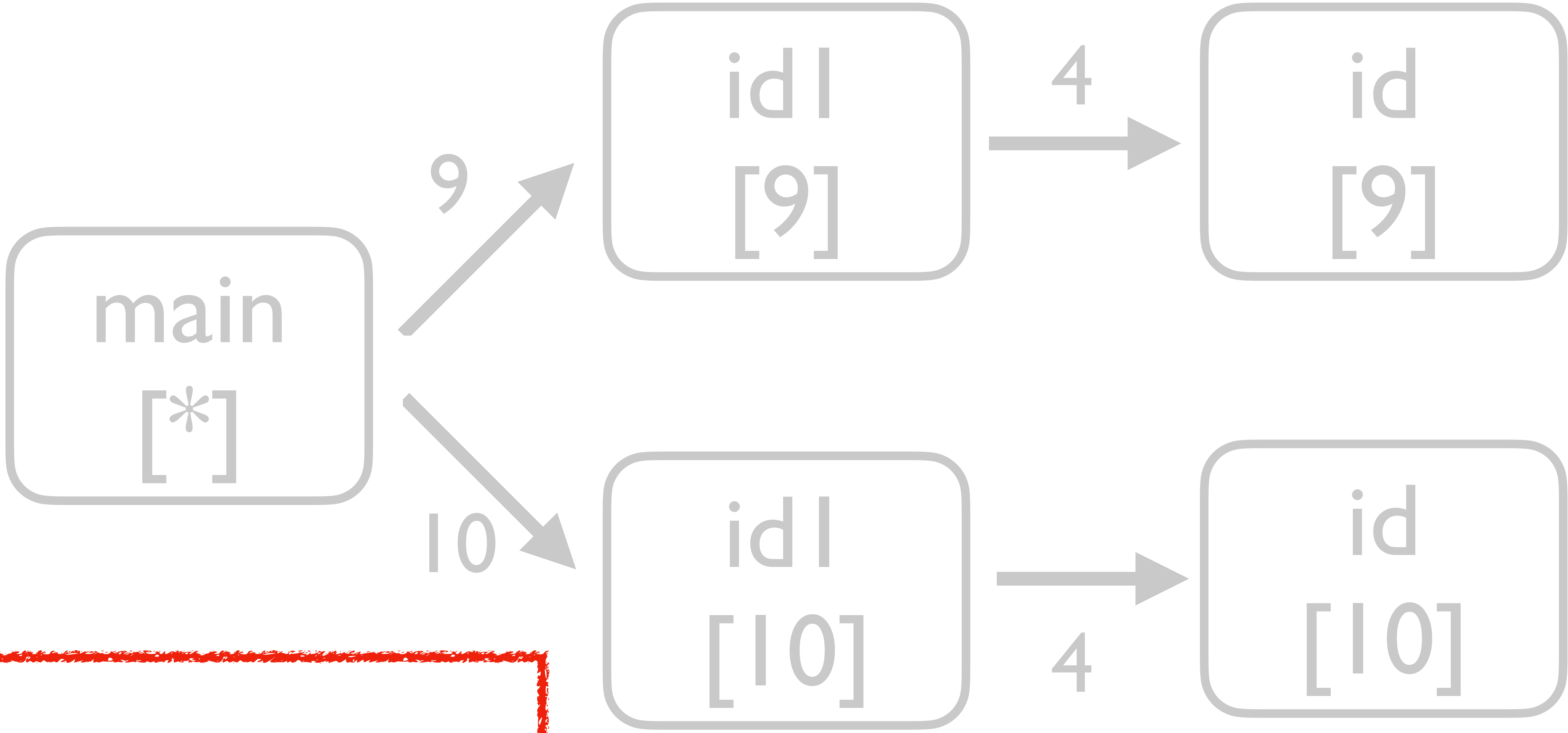


1-CFA with context tunneling
(T= {4})

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
```



th context tunneling

(T= {4})

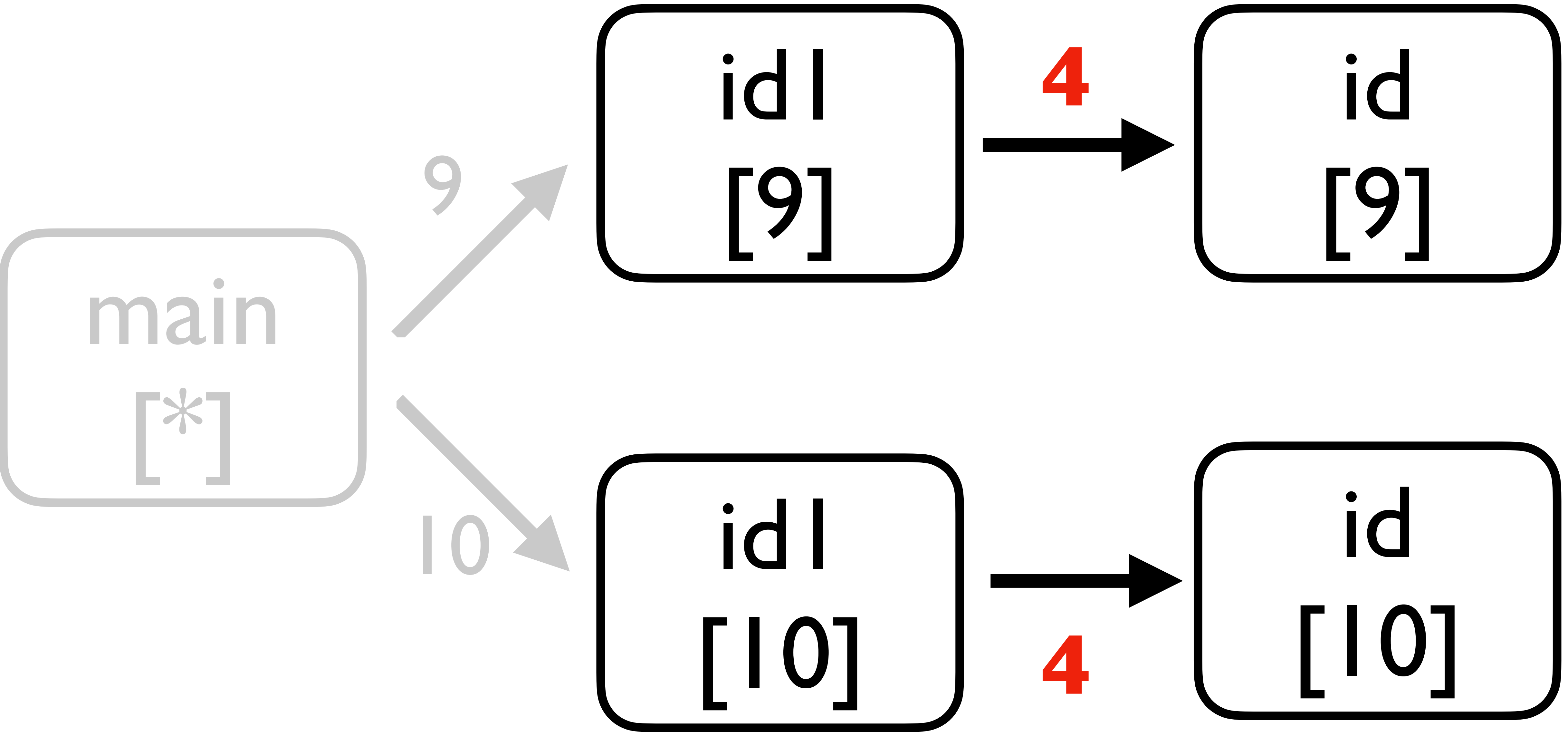
Tunneling abstraction:

Determines where to apply context tunneling

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: }
```



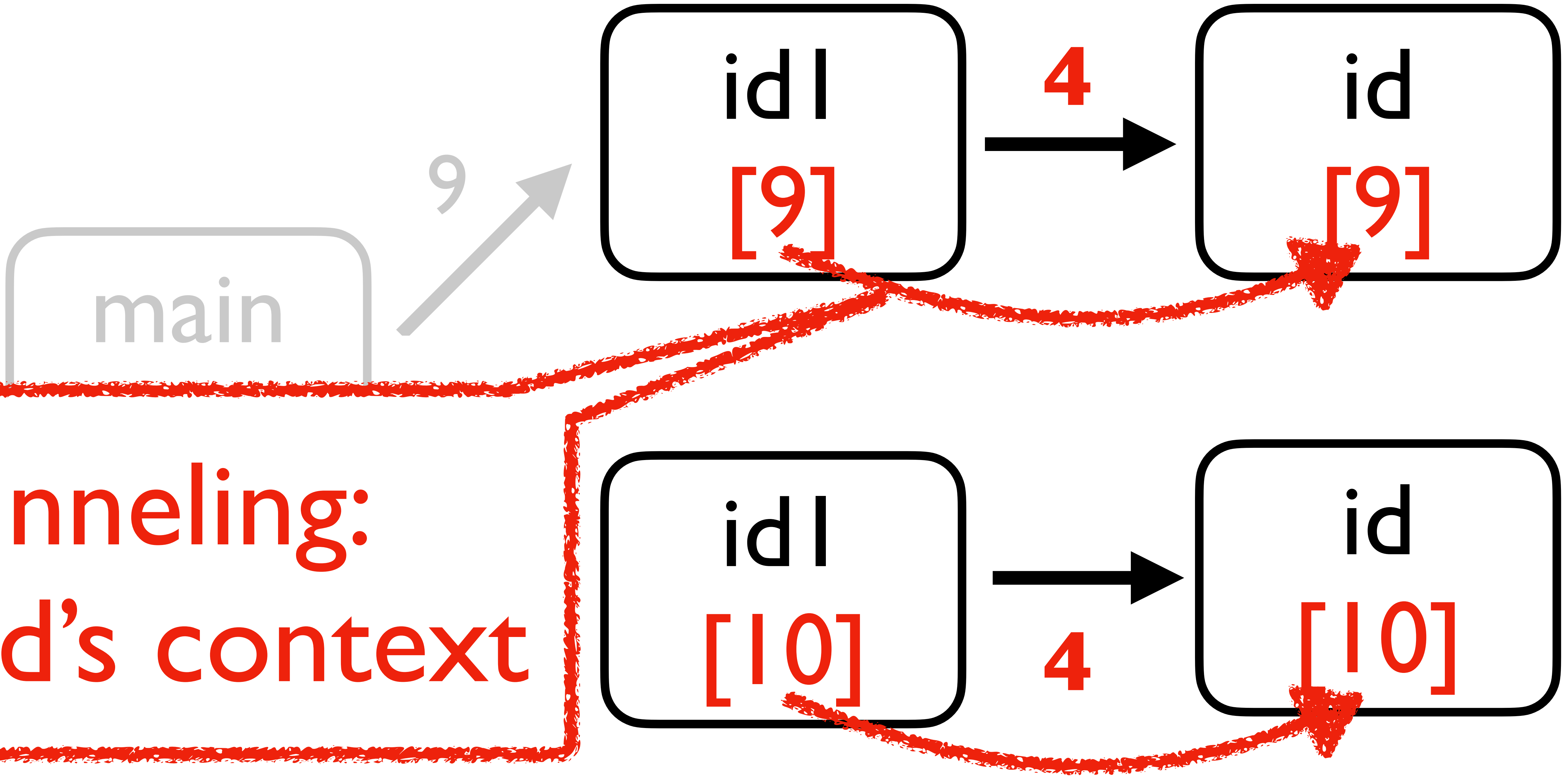
1-CFA with context tunneling
(T= {4})

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



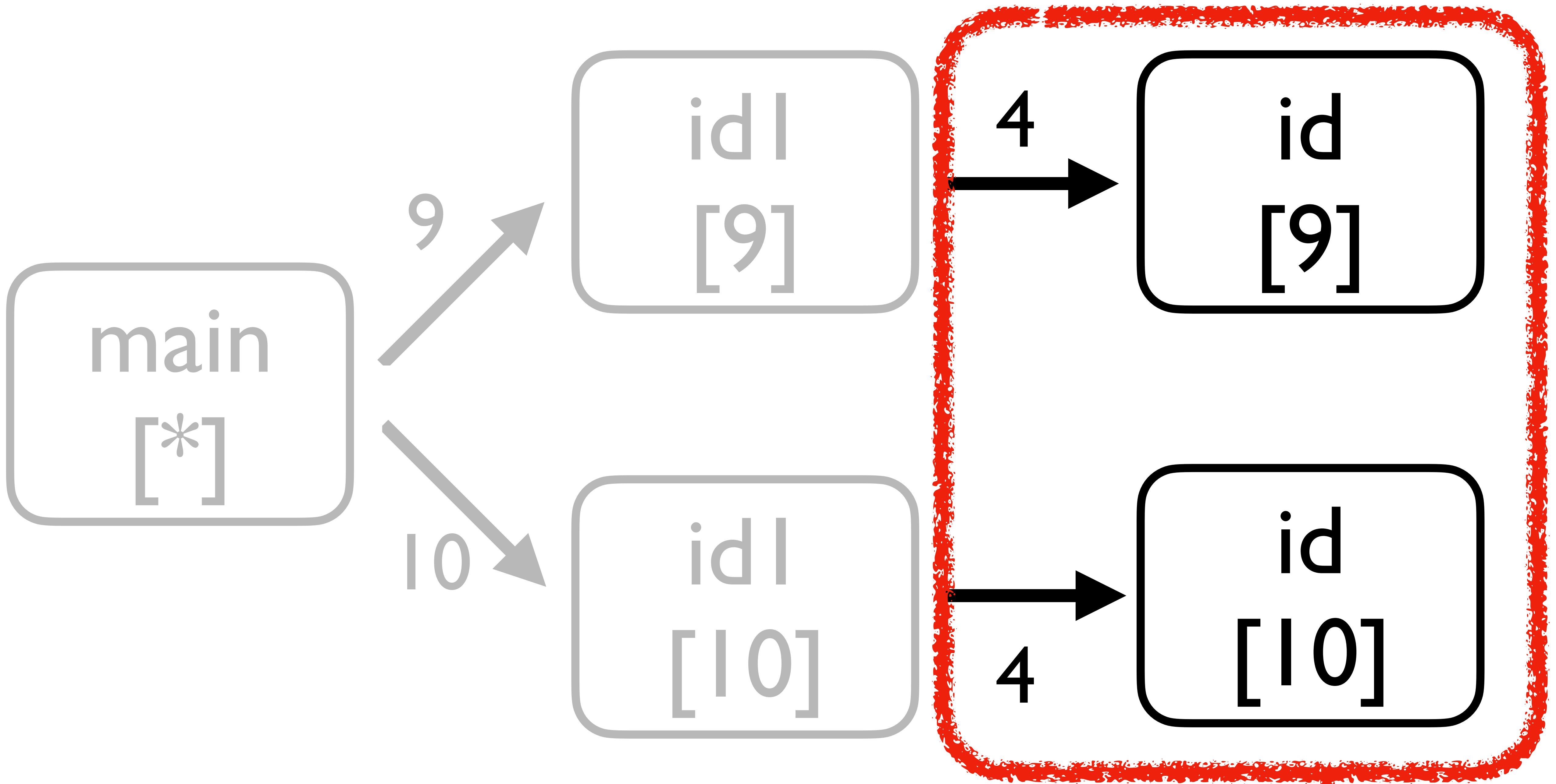
Apply context tunneling:
Inherit caller method's context

1-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: }
```



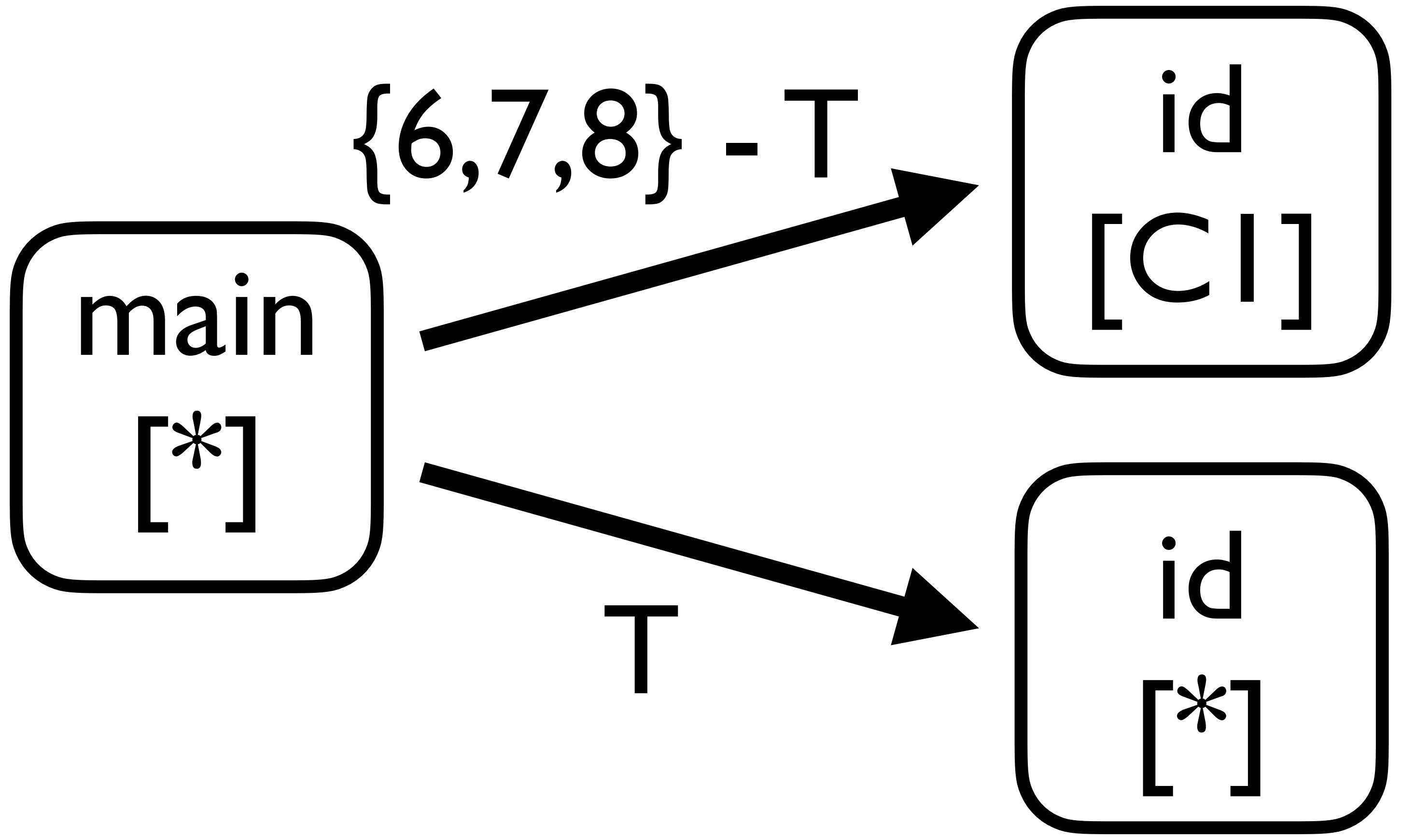
1-CFA with context tunneling
(T= {4})

With tunneling, 1-CFA separates the nested method calls

Call-site Sensitivity vs Object Sensitivity

- Object sensitivity still suffers from its limitation

```
0:  class C{
1:    id(v){
2:      return v;}
3:  }
4:  main(){
5:    c1 = new C();//C1
6:    a = (A) c1.id(new A());
7:    b = (B) c1.id(new B());
8:    c = (C) c1.id(new C());
9:  }
```



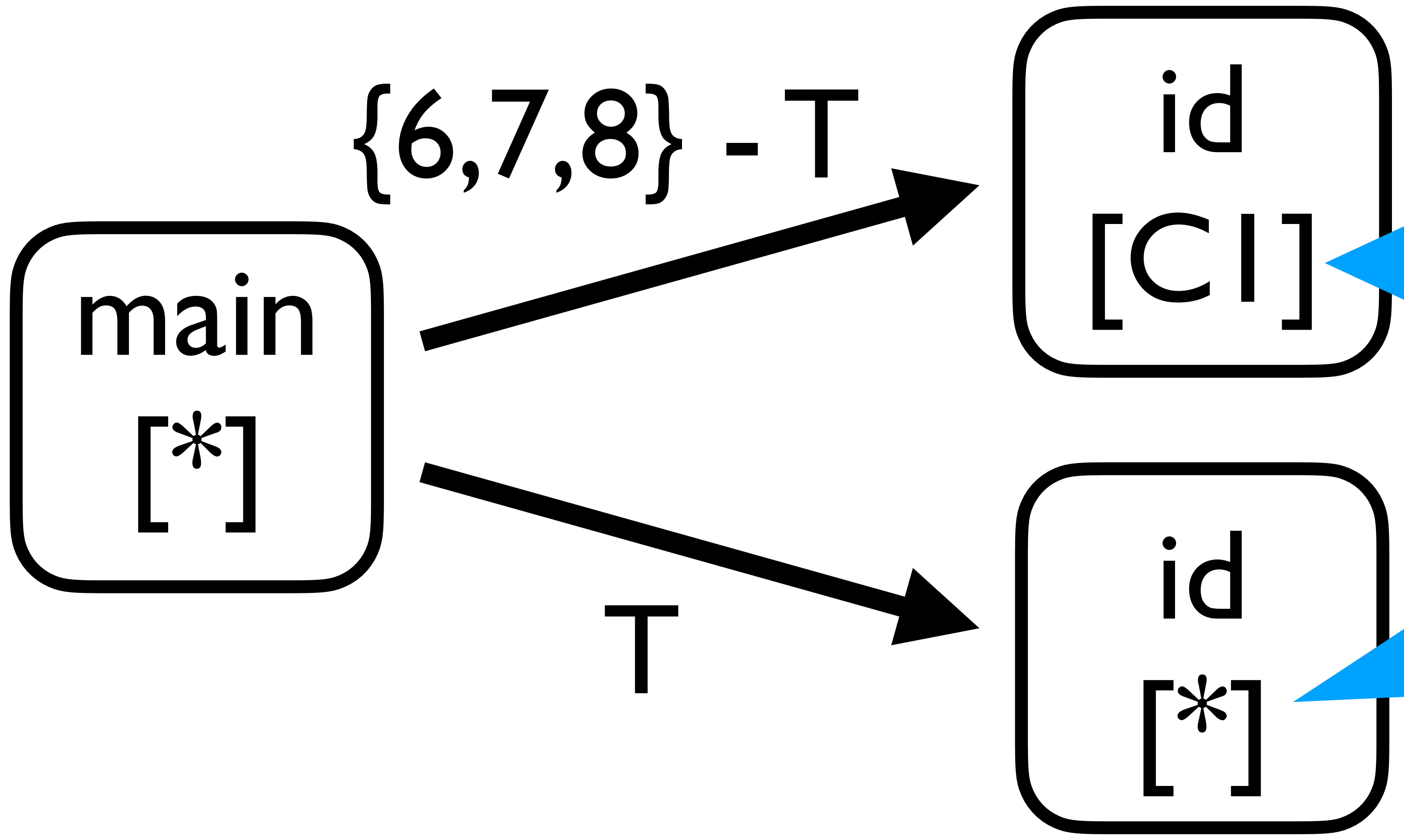
1-Obj + Tunneling
(T = ?)

Call-graph of 1-Obj with tunneling T

Call-site Sensitivity vs Object Sensitivity

- Object sensitivity still suffers from its limitation

```
0:  class C{
1:    id(v){
2:       return v;}
3:  }
4:  main(){
5:    c1 = new C();//C1
6:    a = (A) c1.id(new A());
7:    b = (B) c1.id(new B());
8:    c = (C) c1.id(new C());
9:  }
```



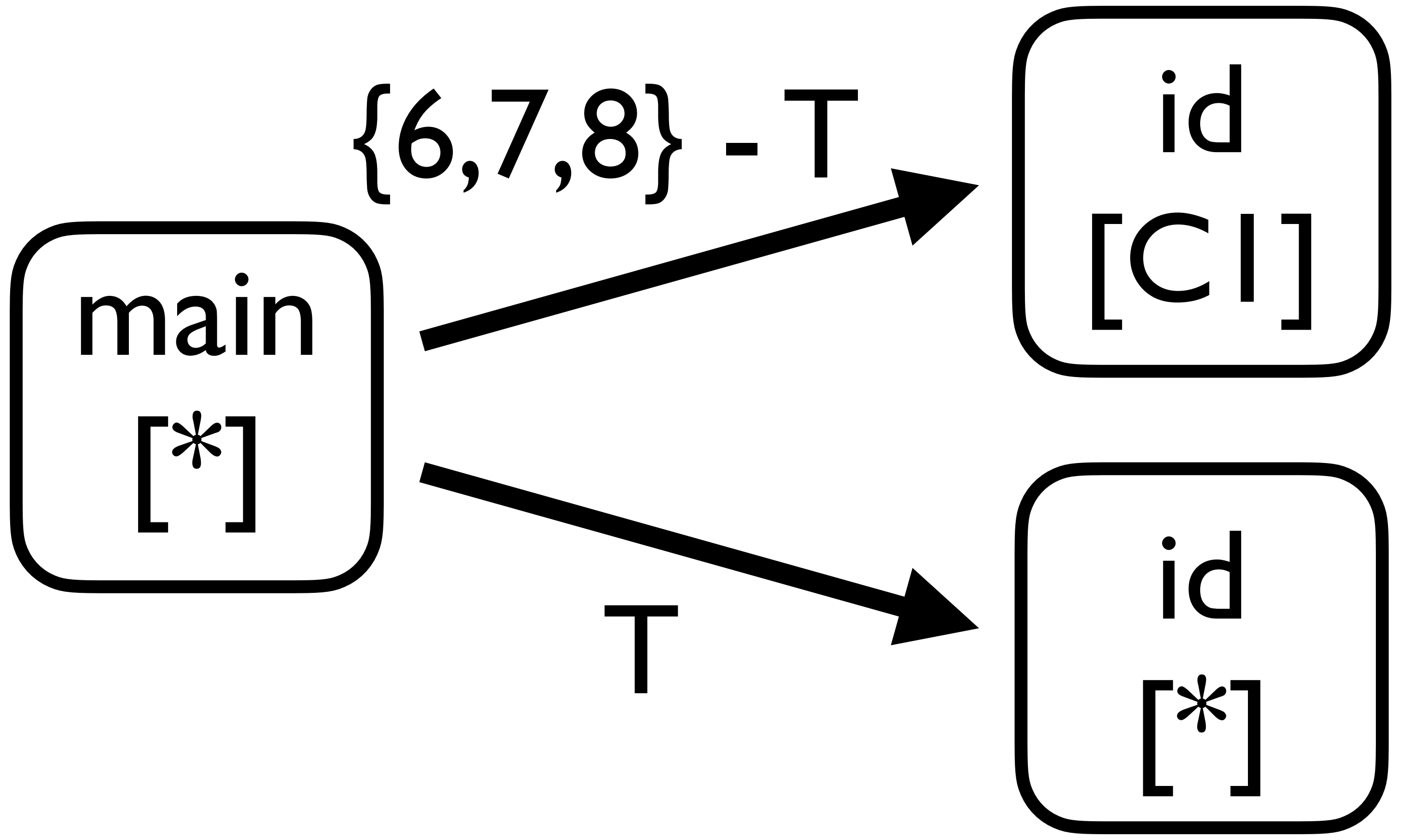
Unable to separate the three method calls with the two contexts

1-Obj + Tunneling
(T = ?)

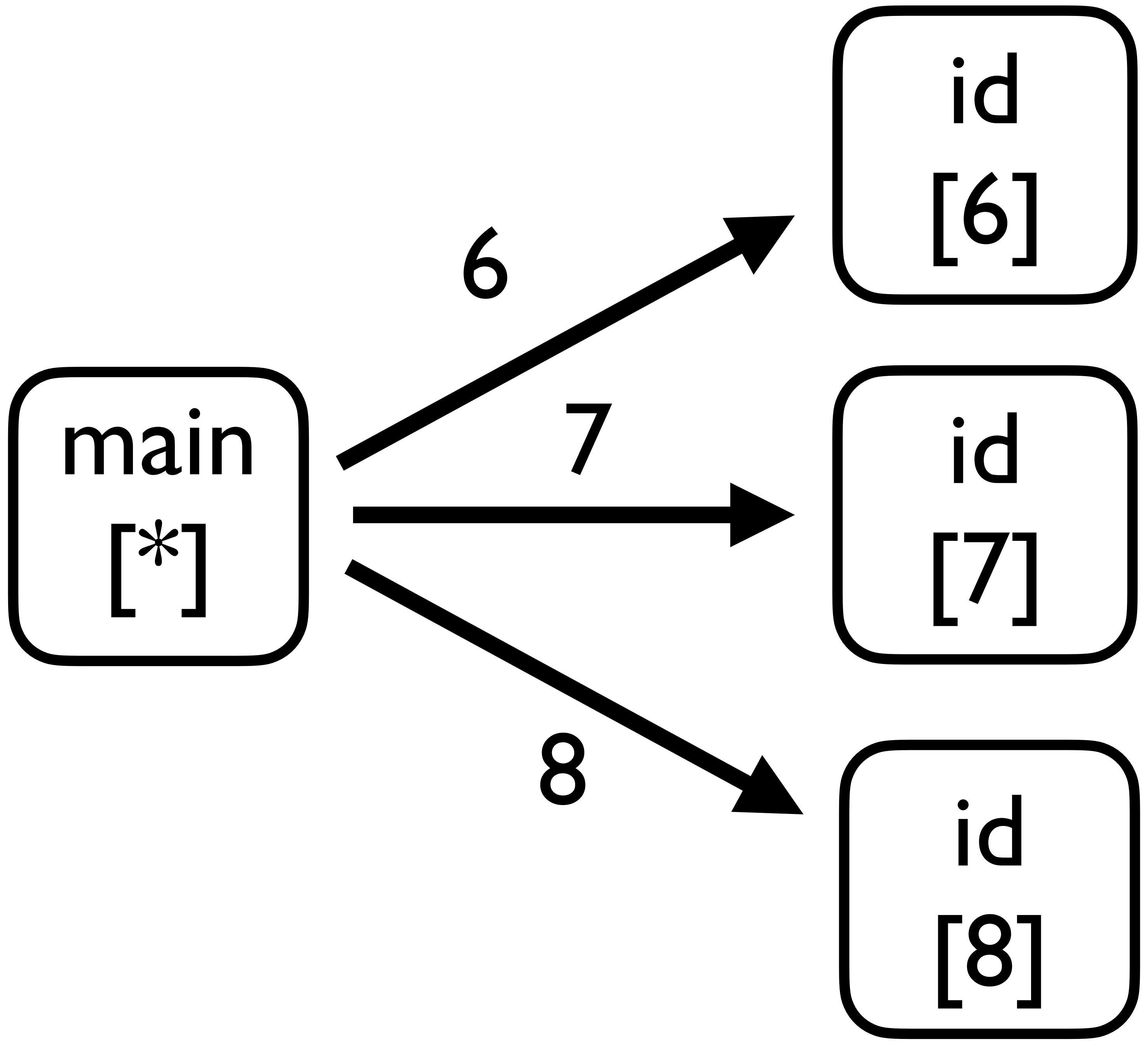
Call-site Sensitivity vs Object Sensitivity

- Object sensitivity still suffers from its limitation

```
0:  class C{
1:    id(v){
2:      return v;}
3:  }
4:  main(){
5:    c1 = new C();//C1
6:    a = (A) c1.id(new A());
7:    b = (B) c1.id(new B());
8:    c = (C) c1.id(new C());
9:  }
```



1-Obj + Tunneling
(T = ?)



1-CFA

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**

```
c


}
m

  a
  b = (B) c1.id(new B());
  c = (B) c1.id(new C());
}
```

Call-site Sensitivity vs Object Sensitivity

- Object sensitivity still suffers from its limitation

```
c
 
 
}
m
 
  a
  b
  c
}
```

Observation

When context tunneling is included

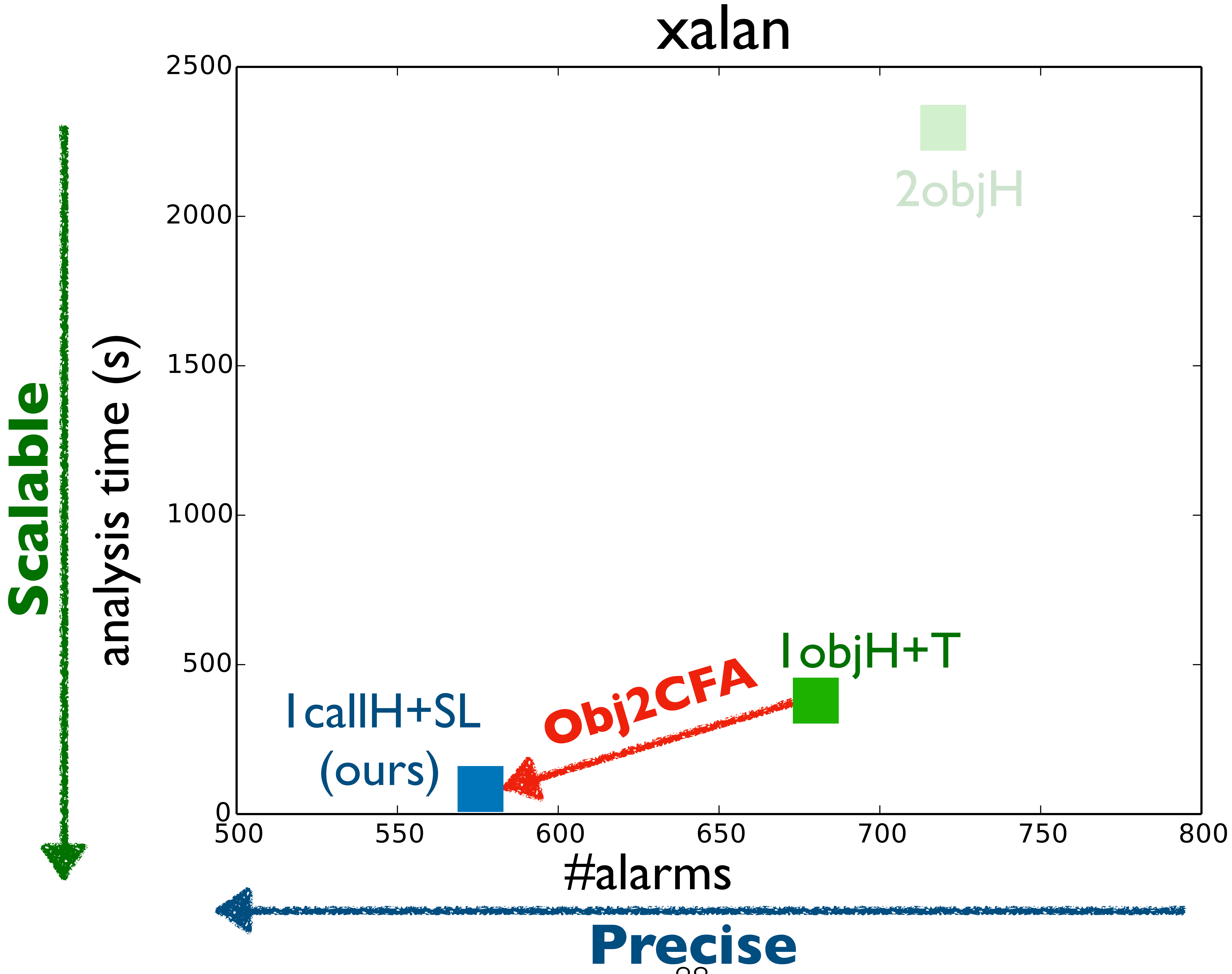
- Limitation of call-site sensitivity is **removed**
- Limitation of object sensitivity is **not removed**

Our claim

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

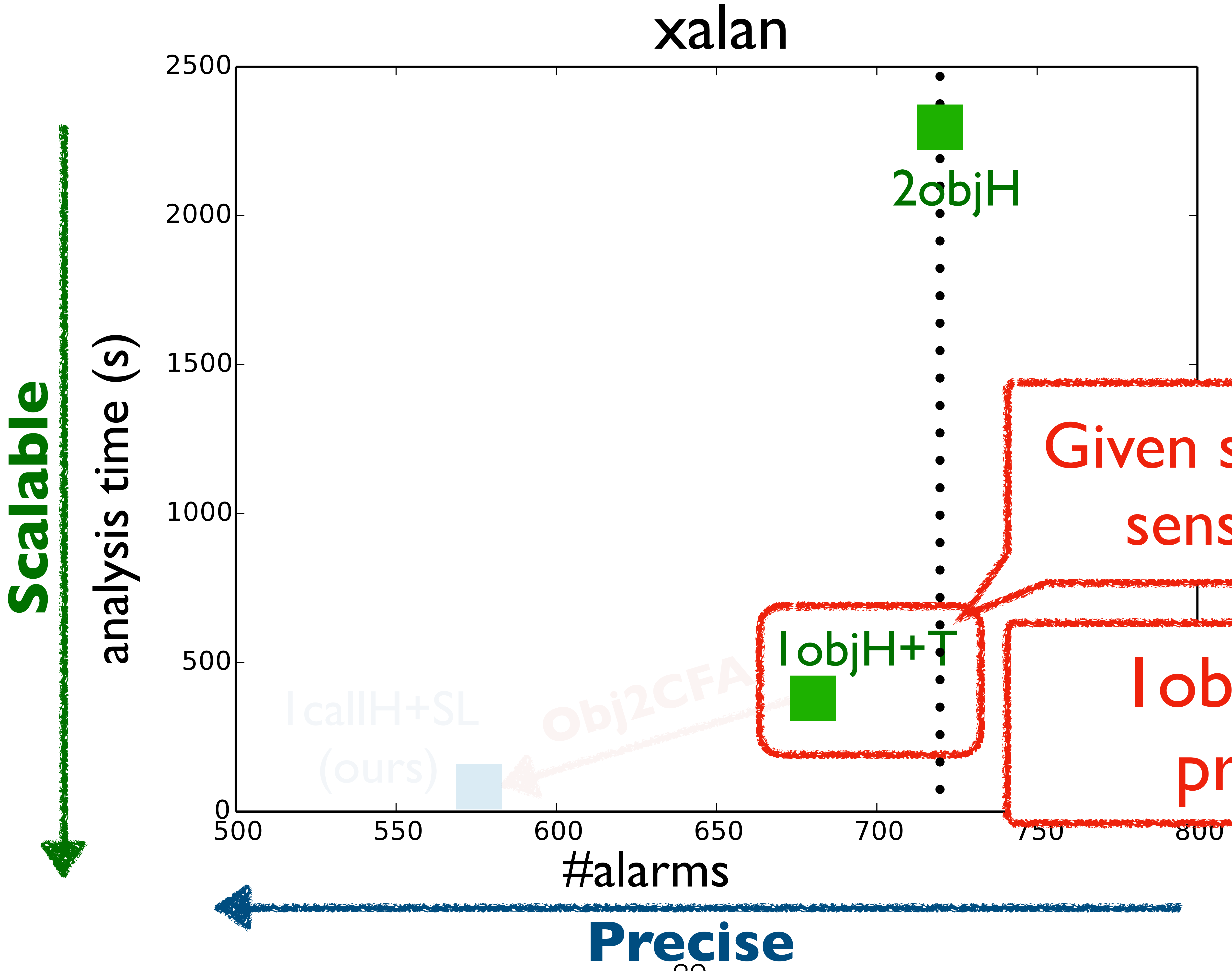
Our Technique : **Obj2CFA**

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



Our Technique : **Obj2CFA**

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

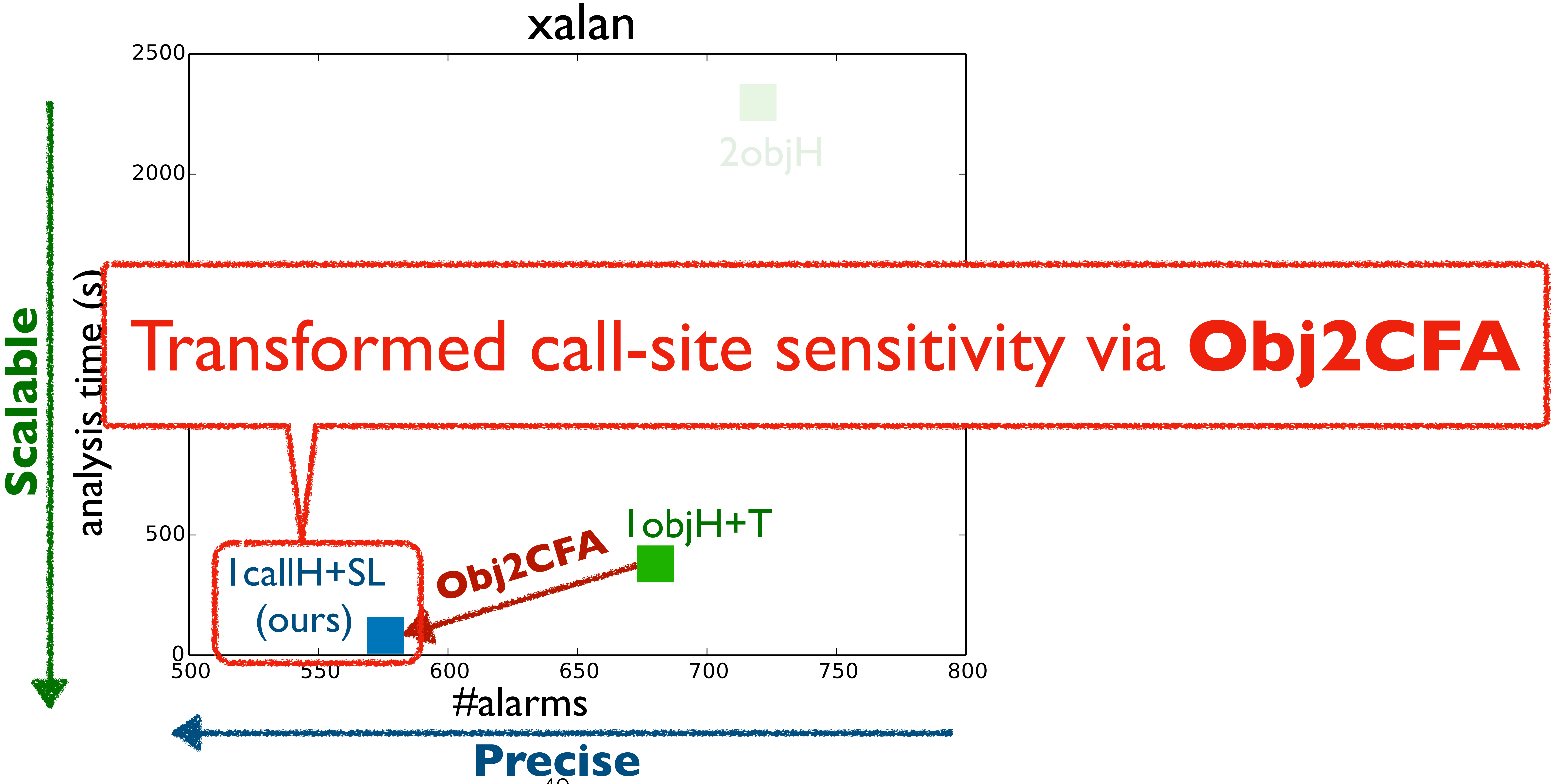


Given state-of-the art 1-object sensitivity with tunneling

1objH+T is even more precise than 2objH

Our Technique : **Obj2CFA**

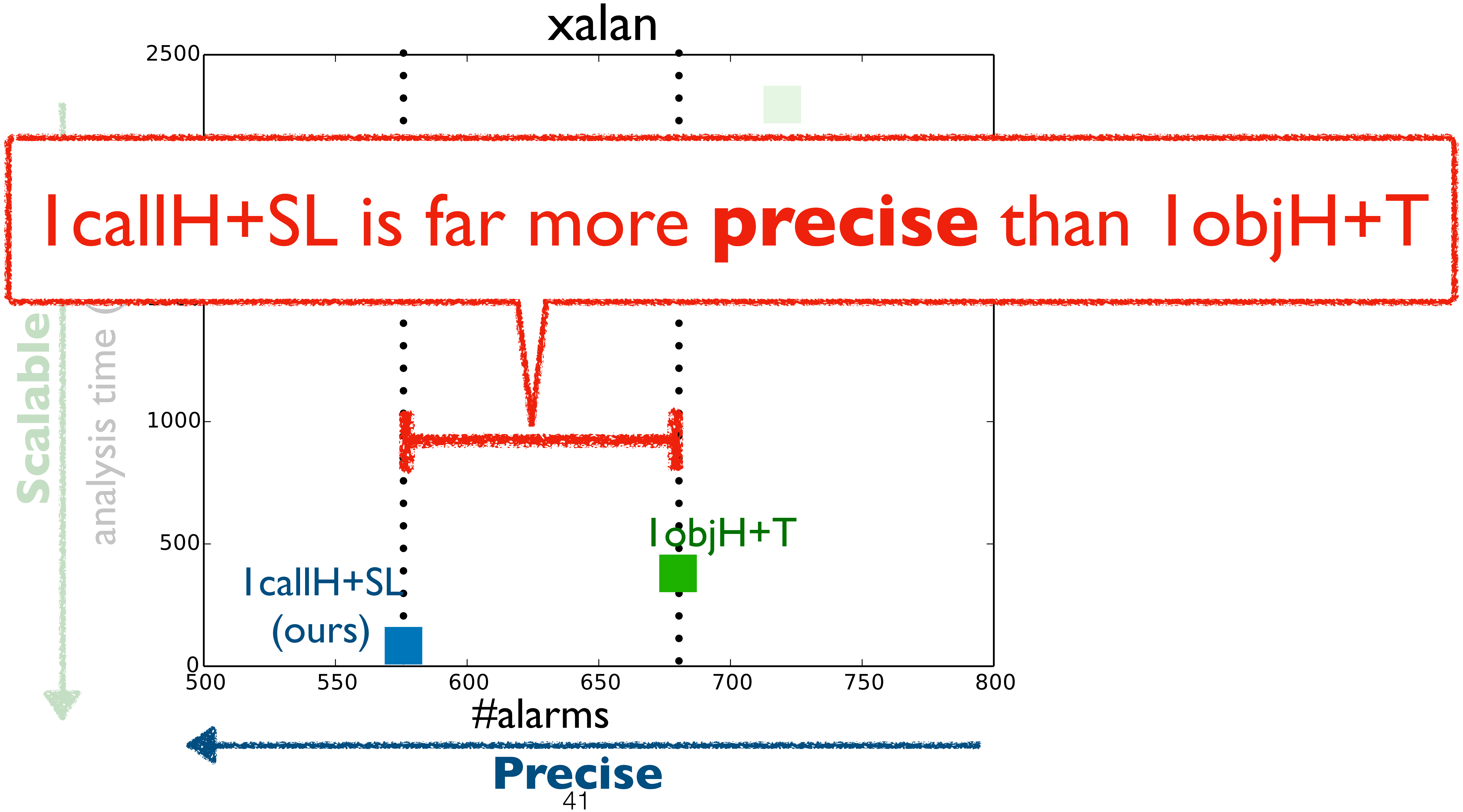
- **Obj2CFA** transforms a given object sensitivity into a more precise CFA



Transformed call-site sensitivity via **Obj2CFA**

Our Technique : **Obj2CFA**

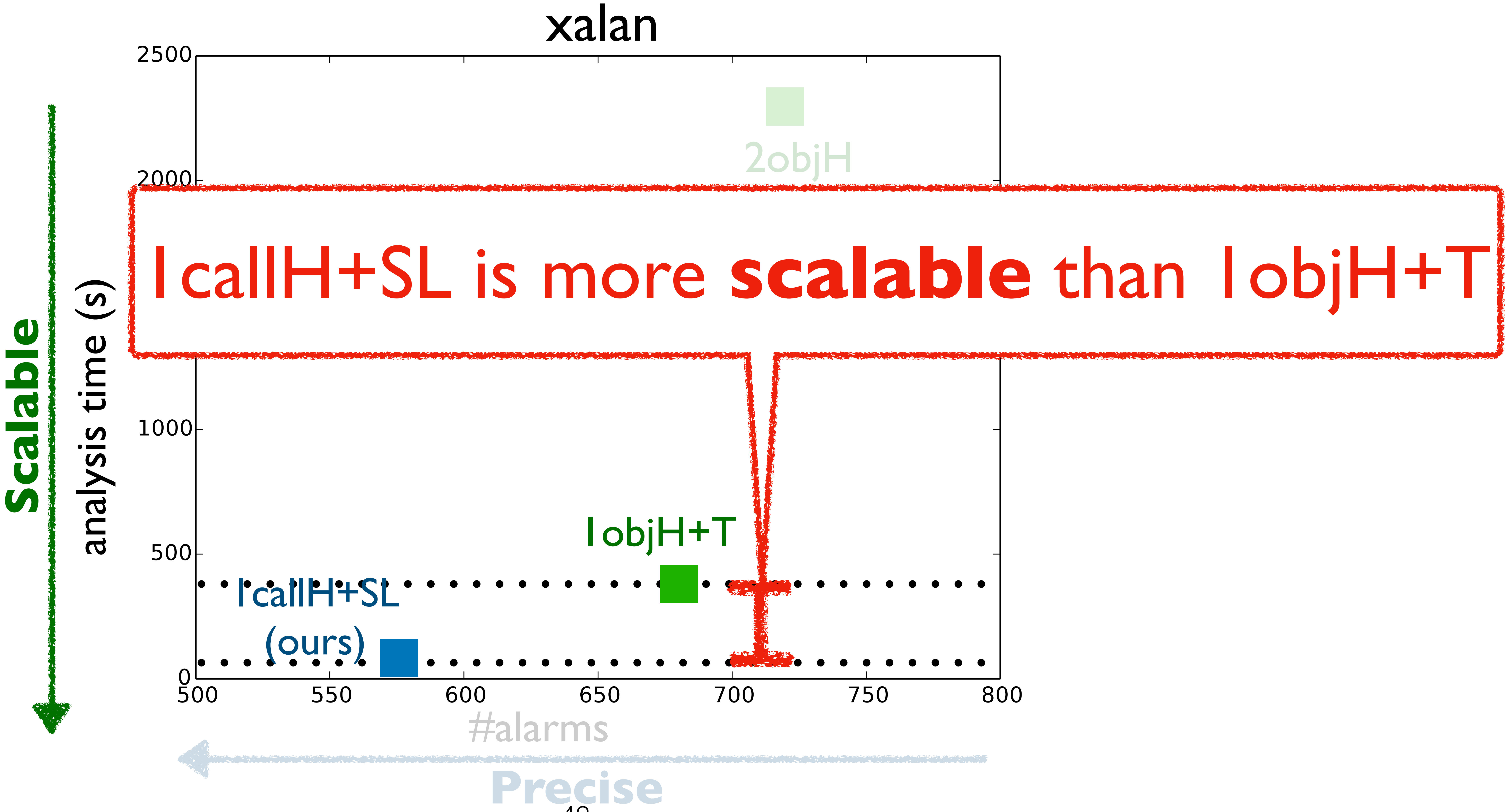
- **Obj2CFA** transforms a given object sensitivity into a more precise CFA



1callH+SL is far more **precise** than 1objH+T

Our Technique : **Obj2CFA**

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

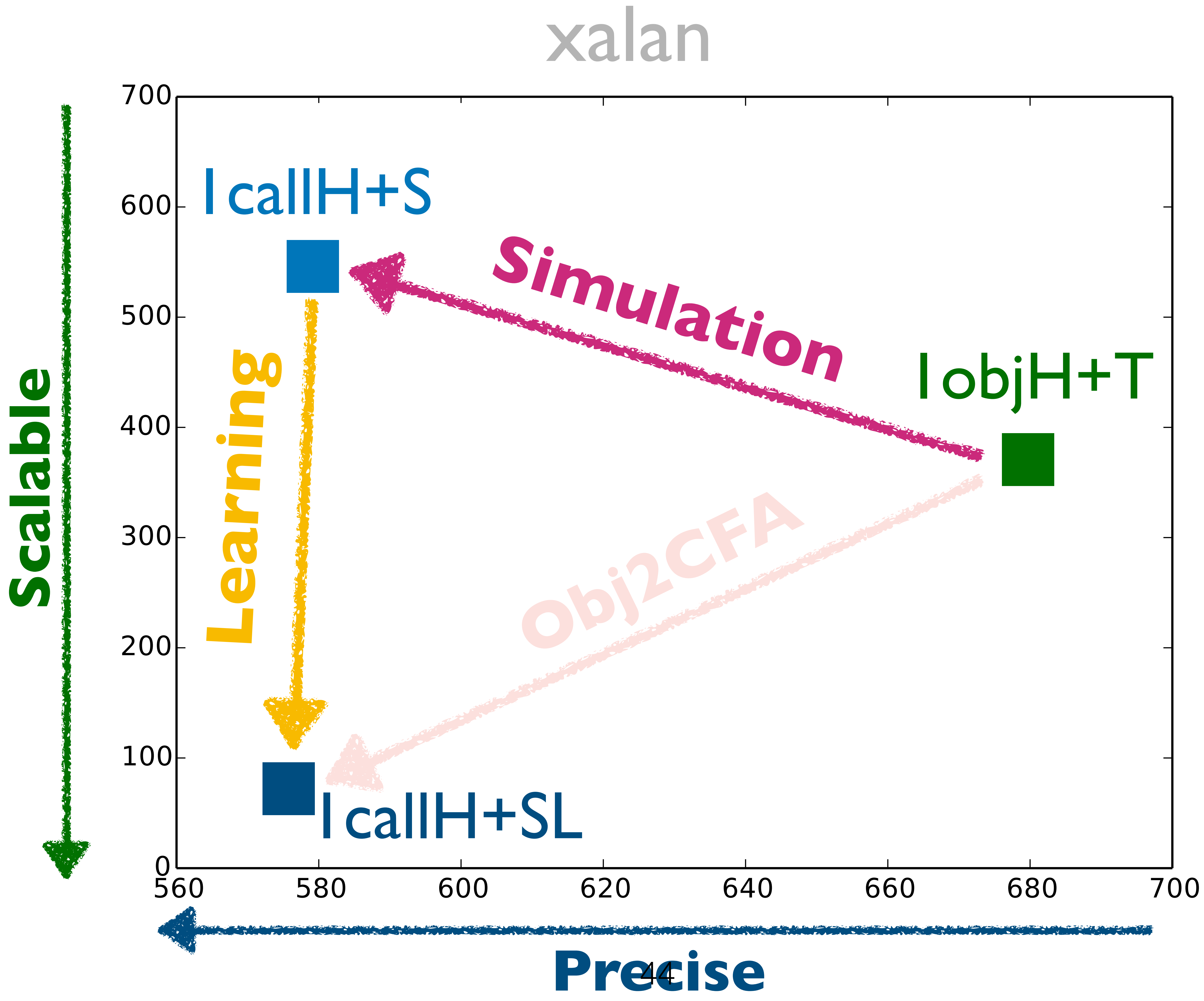


1callH+SL is more **scalable** than 1objH+T

Detail of Obj2CFA

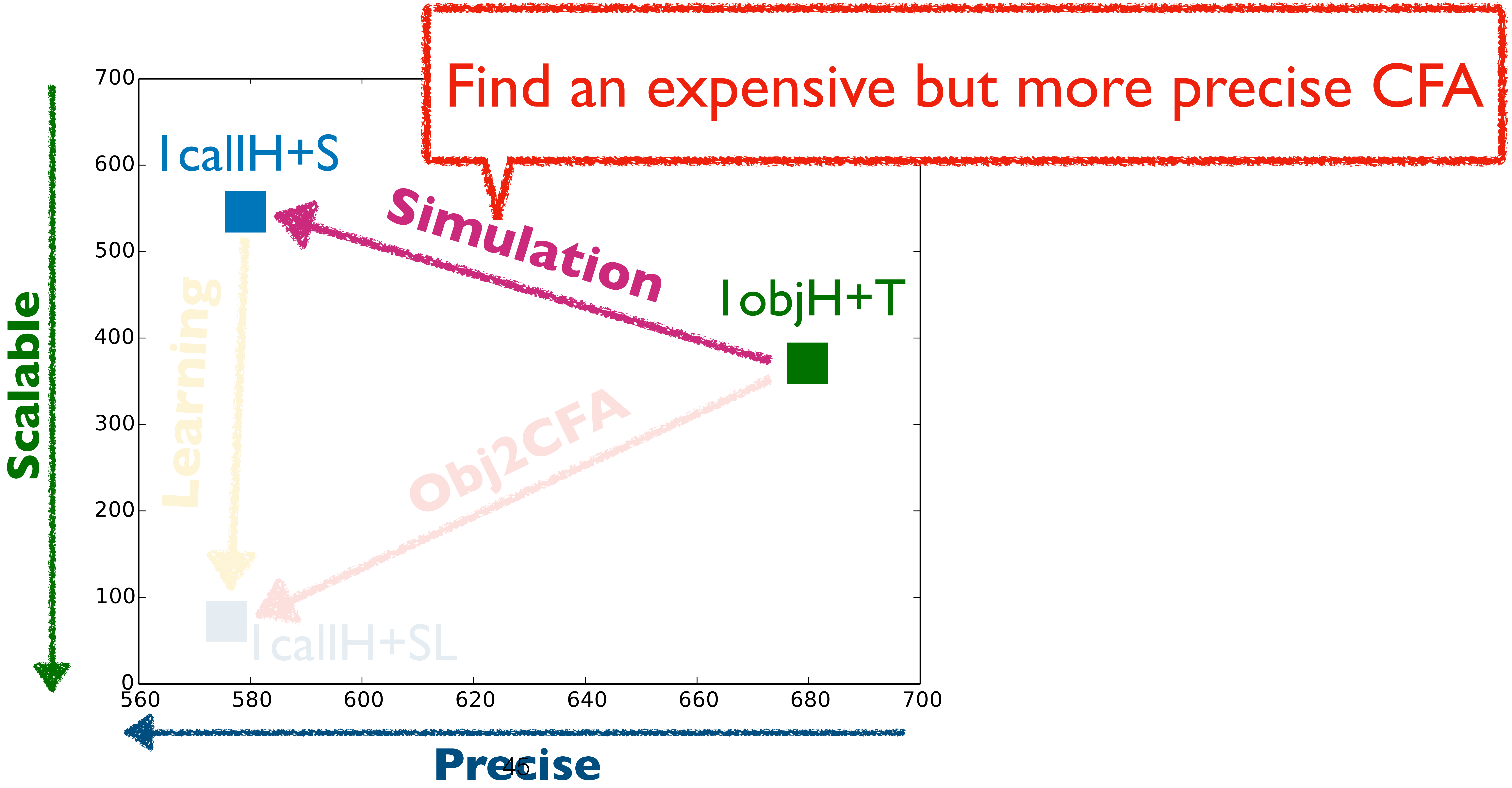
Our Technique : **Obj2CFA**

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



Our Technique : **Obj2CFA**

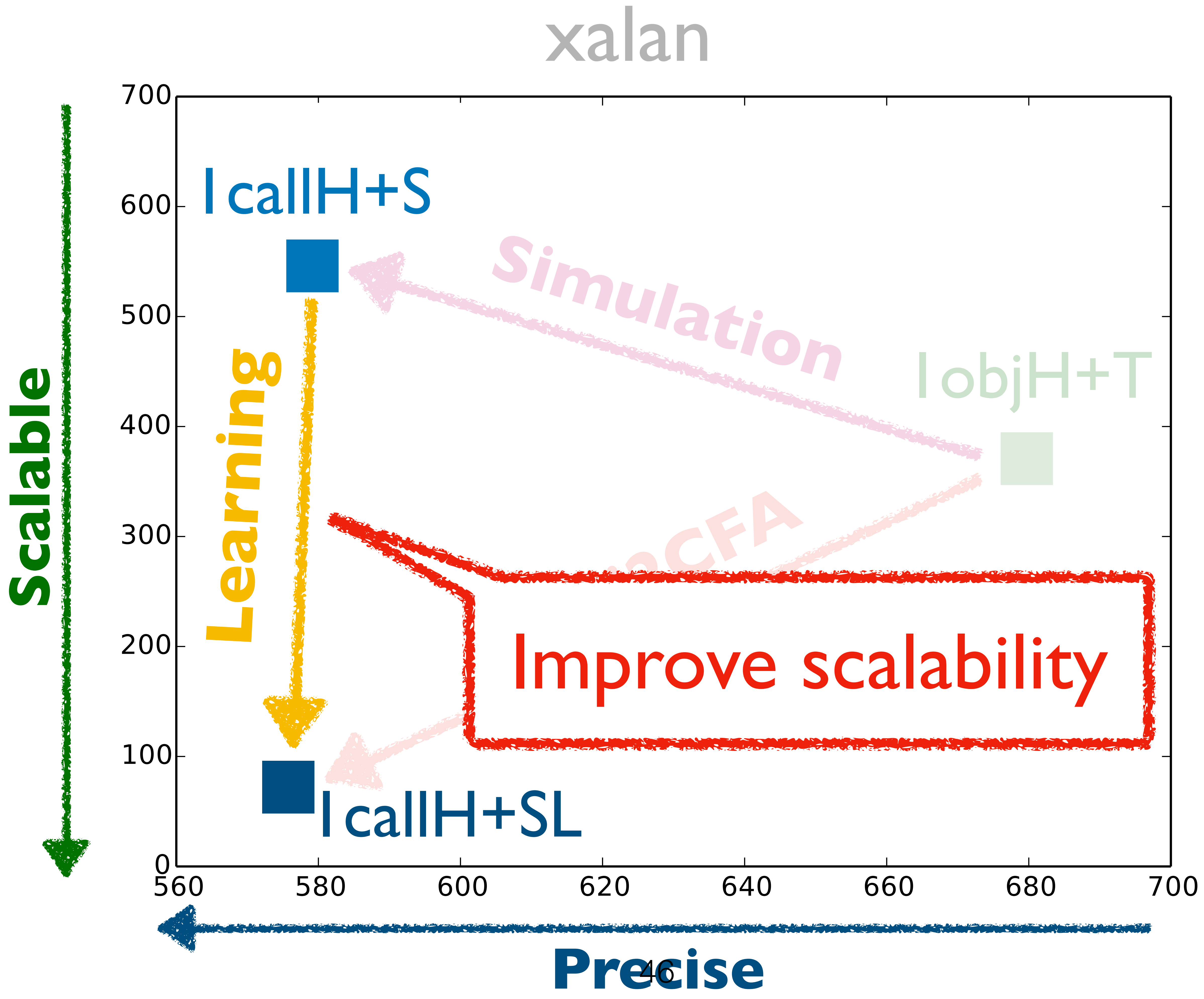
- **Obj2CFA** consists of **simulation** and simulation-guided **learning**



Find an expensive but more precise CFA

Our Technique : **Obj2CFA**

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



Technique 1: Simulation

- Running example to illustrate Simulation

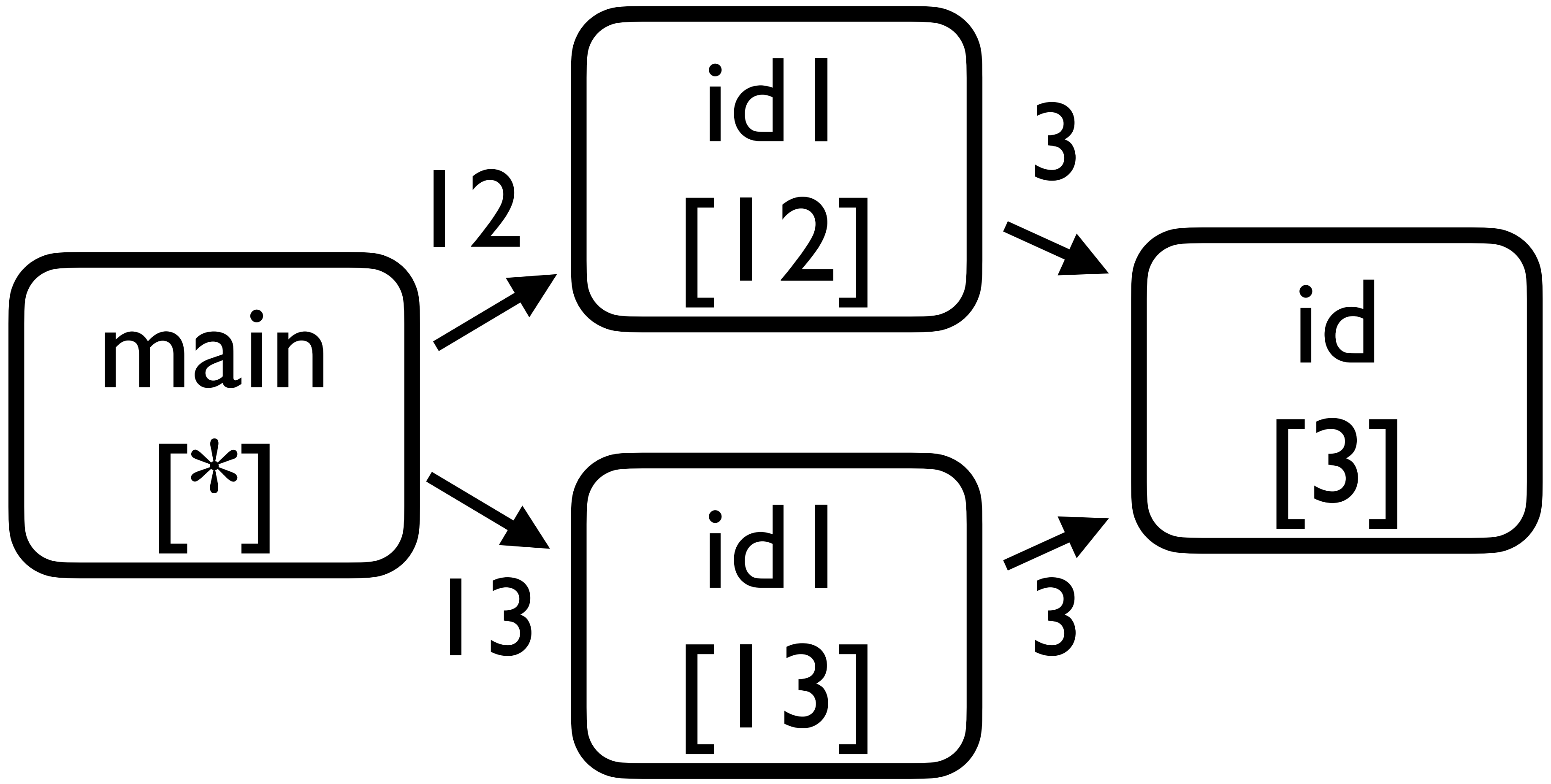
```
class C{
  id(v){return v;}
  id1(v){return id(v);}
  foo(){
    A a = (A) this.id(new A());}//query1
    B b = (B) this.id(new B());}//query2
}
main(){
  c1 = new C(); //C1
  c2 = new C(); //C2
  c3 = new C(); //C3
  A a = (A) c1.id1(new A());//query3
  B b = (B) c2.id1(new B());//query4
  c3.foo();
}
```

Technique 1: Simulation

- Running example to illustrate Simulation

```
1:  class C{
2:    id(v){return v;}
3:    id1(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.id1(new A());//query3
13:   B b = (B) c2.id1(new B());//query4
14:   c3.foo();
15: }
```

Limitation of conventional 1-CFA

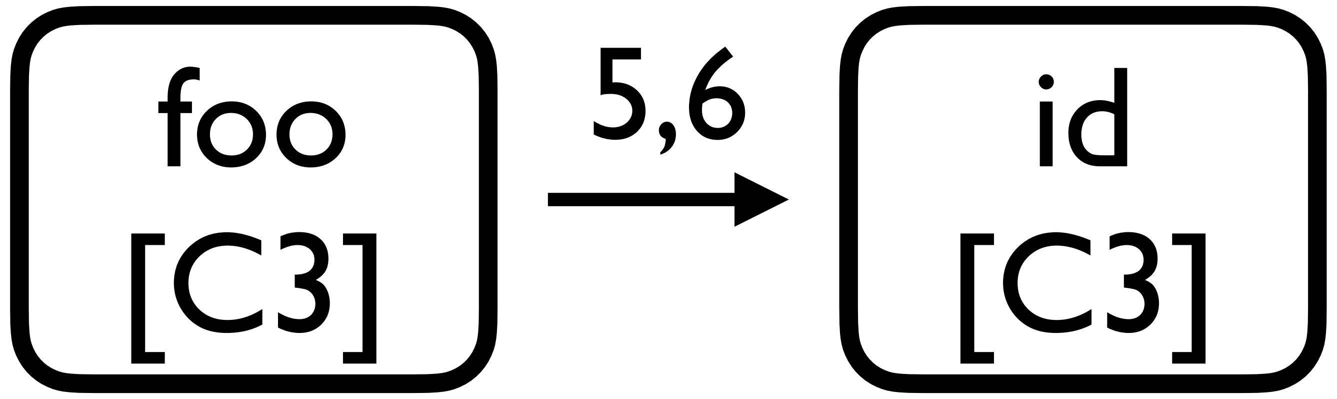


Technique 1: Simulation

- Running example to illustrate Simulation

```
1:  class C{
2:    id(v){return v;}
3:    id1(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.id1(new A());//query3
13:   B b = (B) c2.id1(new B());//query4
14:   c3.foo();
15: }
```

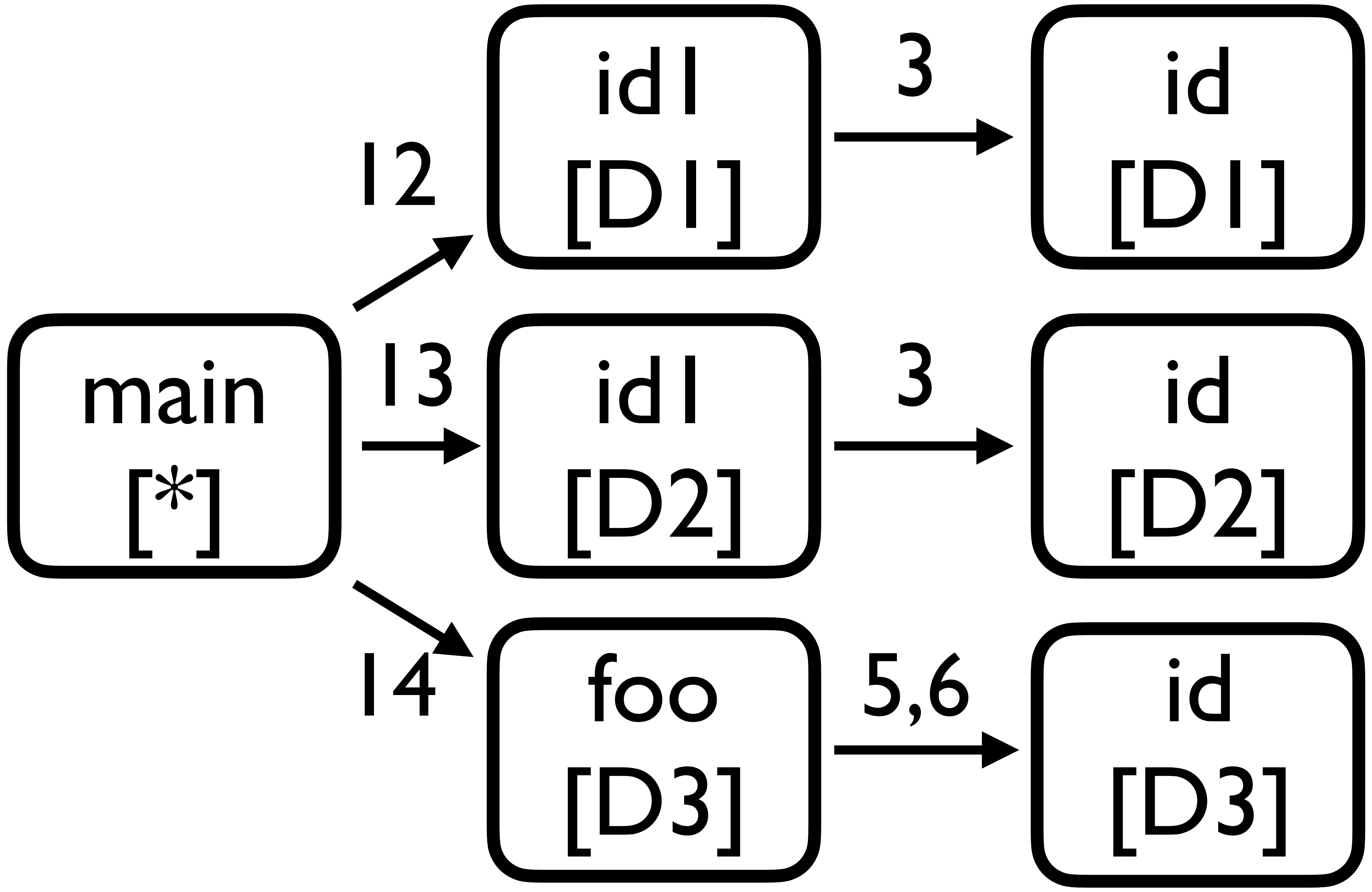
Limitation of object sensitivity



Technique 1: Simulation

- Given object sensitivity is conventional 1-object sensitivity (e.g., T = ∅)

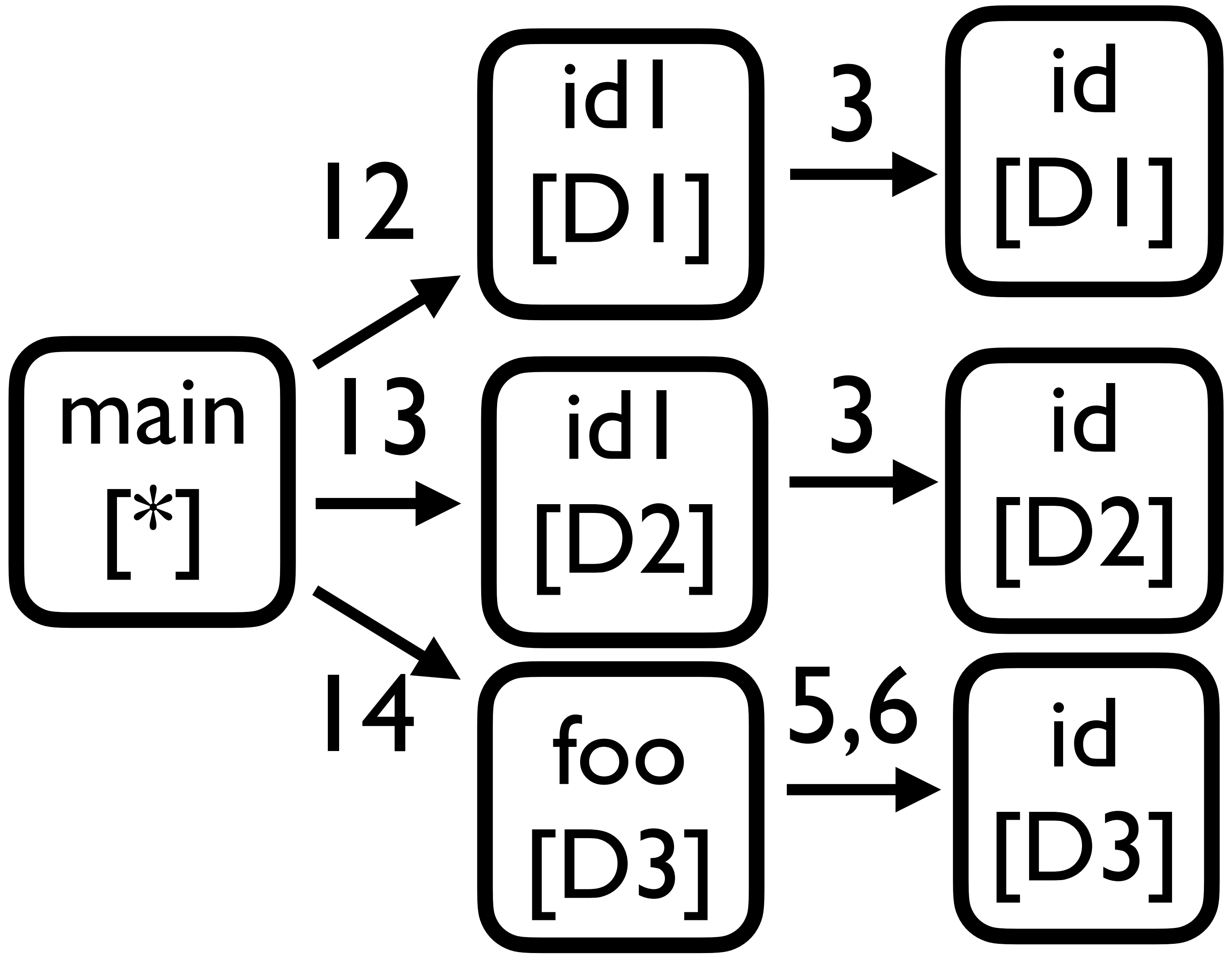
```
1:  class C{
2:    id(v){return v;}
3:    id1(v){return id(v);}
4:    foo(){
5:      A a = (A) this.id(new A());}//query1
6:       B b = (B) this.id(new B());}//query2
7:  }
8:  main(){
9:    c1 = new C(); //C1
10:   c2 = new C(); //C2
11:   c3 = new C(); //C3
12:  A a = (A) c1.id1(new A());//query3
13:   B b = (B) c2.id1(new B());//query4
14:   c3.foo();
15: }
```



1objH+T (T = ∅)

Technique 1: Simulation

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



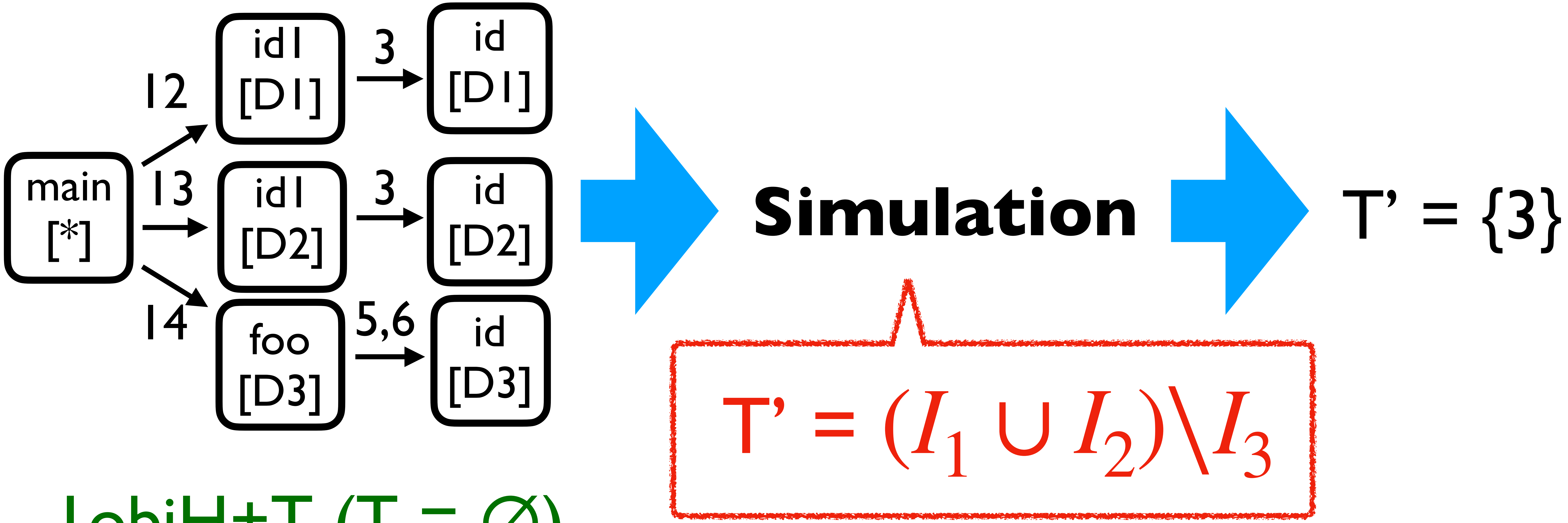
1objH+T (T = ∅)

→ **Simulation** → T’ = {3}

Tunneling abstraction for 1-CFA

Technique 1: Simulation

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



main [*]

12

13

14

id1 [D1]

id1 [D2]

foo [D3]

3

3

5,6

id [D1]

id [D2]

id [D3]

Simulation

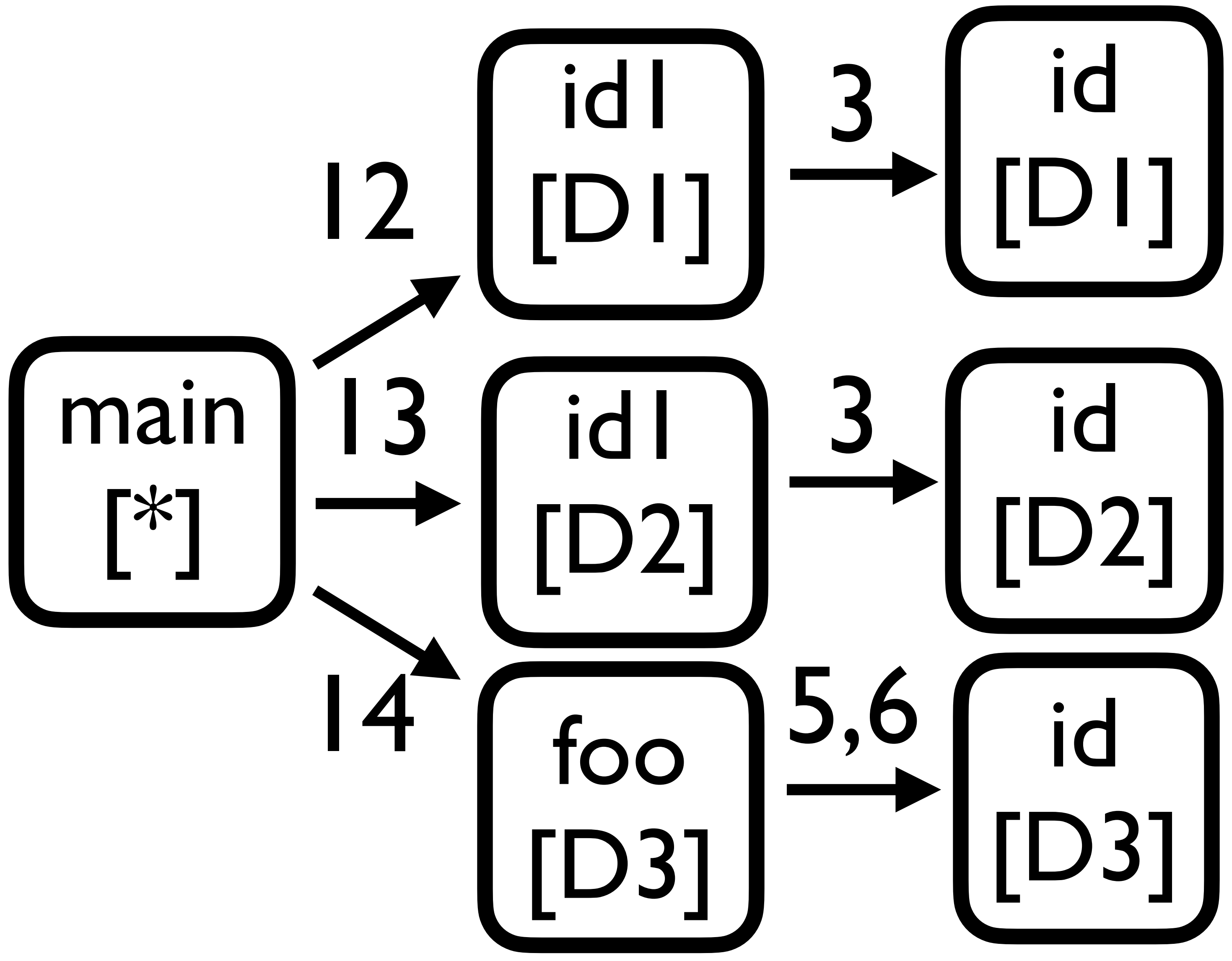
T' = {3}

$$T' = (I_1 \cup I_2) \backslash I_3$$

1objH+T (T = ∅)

Technique 1: Simulation

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



1objH+T (T = ∅)

Simulation

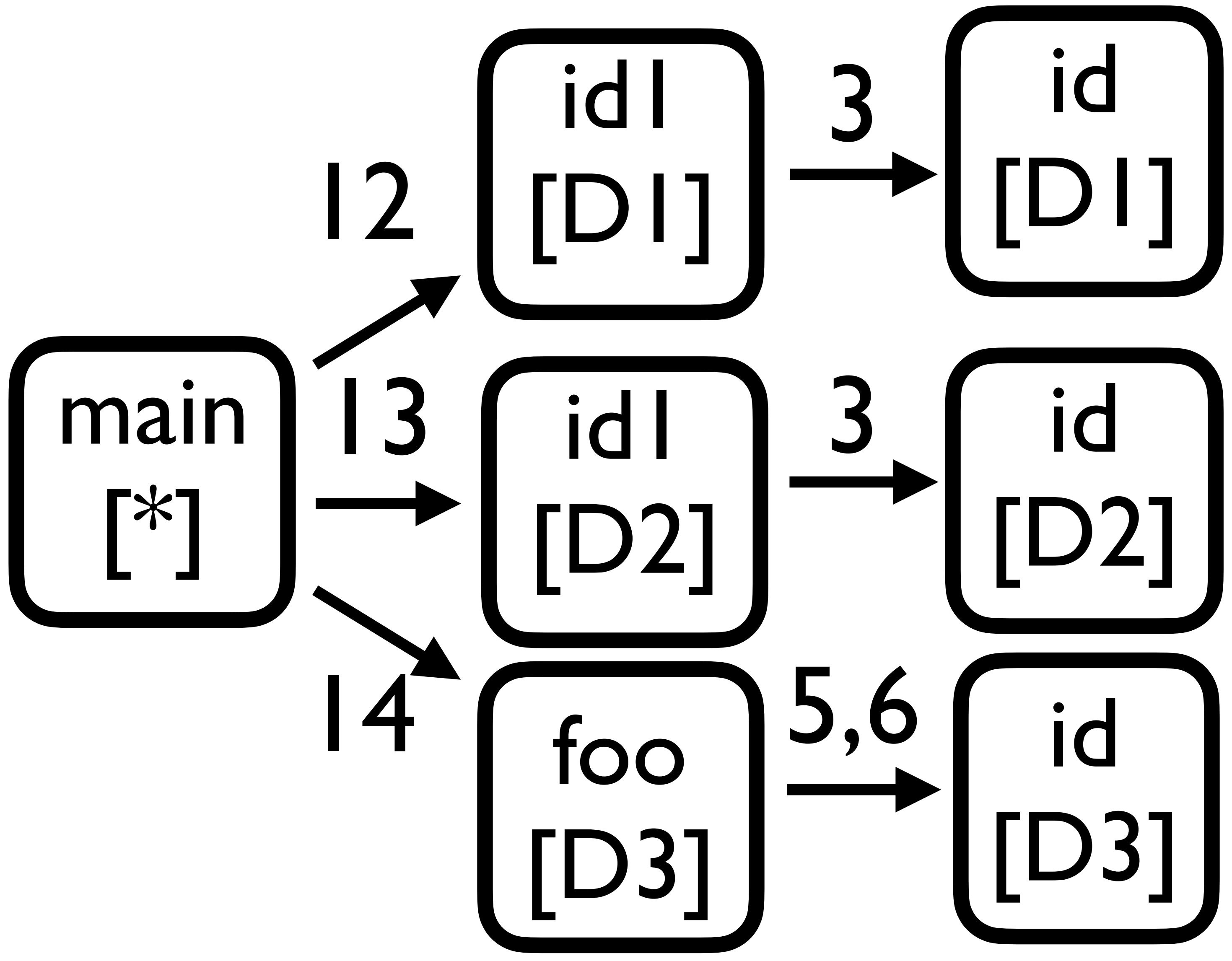
T' = {3}

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

Need tunneling to simulate the given object sensitivity

Technique 1: Simulation

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



1objH+T (T = ∅)

Simulation

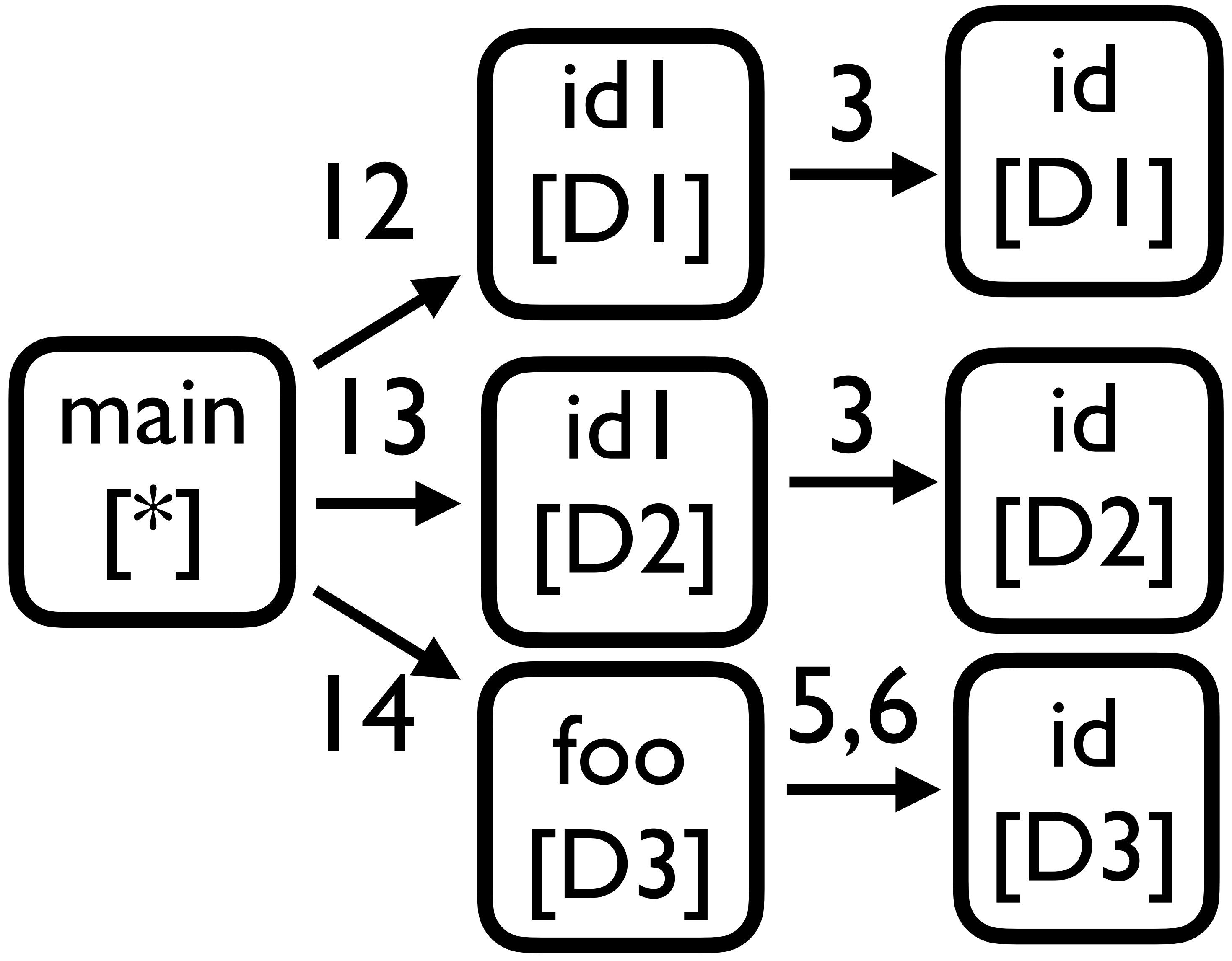
T’ = {3}

T’ = $(I_1 \cup I_2)\boxed{\setminus I_3}$

Tunneling should be avoided for improving precision

Technique 1: 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'

~~1objH+T (T = ∅)~~

1callH+T'

What is T'?

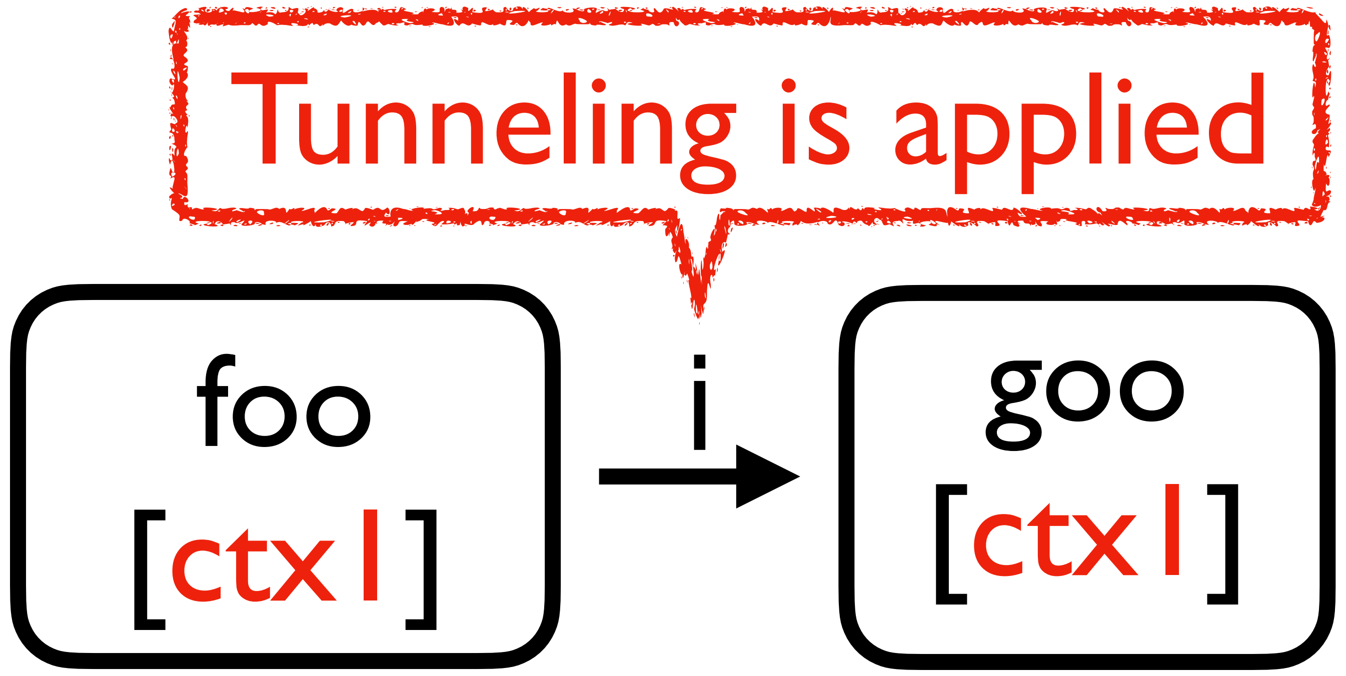
Intuition Behind Simulation ($I_1 \cup I_2$)

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

We track the two properties to find the T’

Intuition Behind Simulation $(I_1 \cup I_2)$

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



Property of context tunneled call-sites

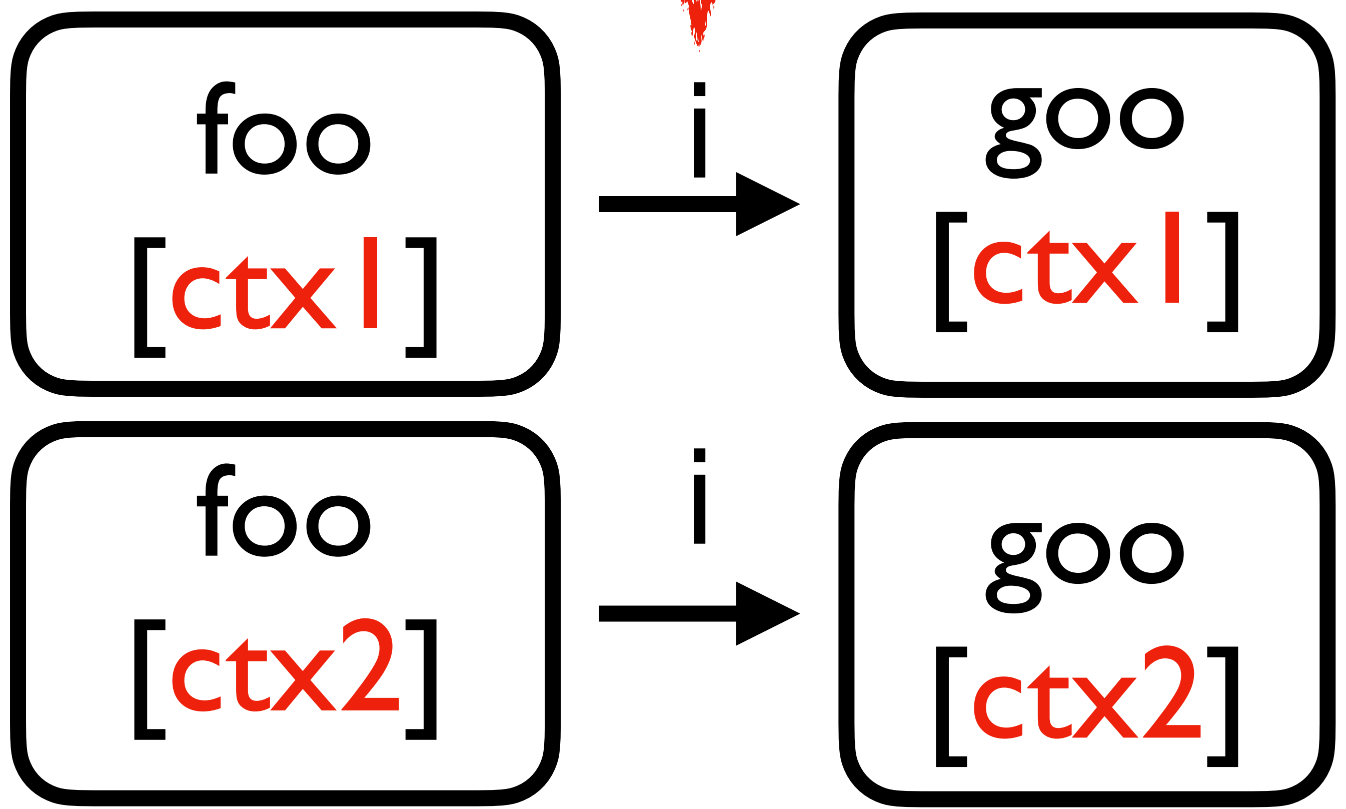
I_1

- Property 1: caller and callee methods have the same context

Intuition Behind Simulation ($I_1 \cup I_2$)

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

Tunneling is applied



I_1

Property of context tunneled invocations

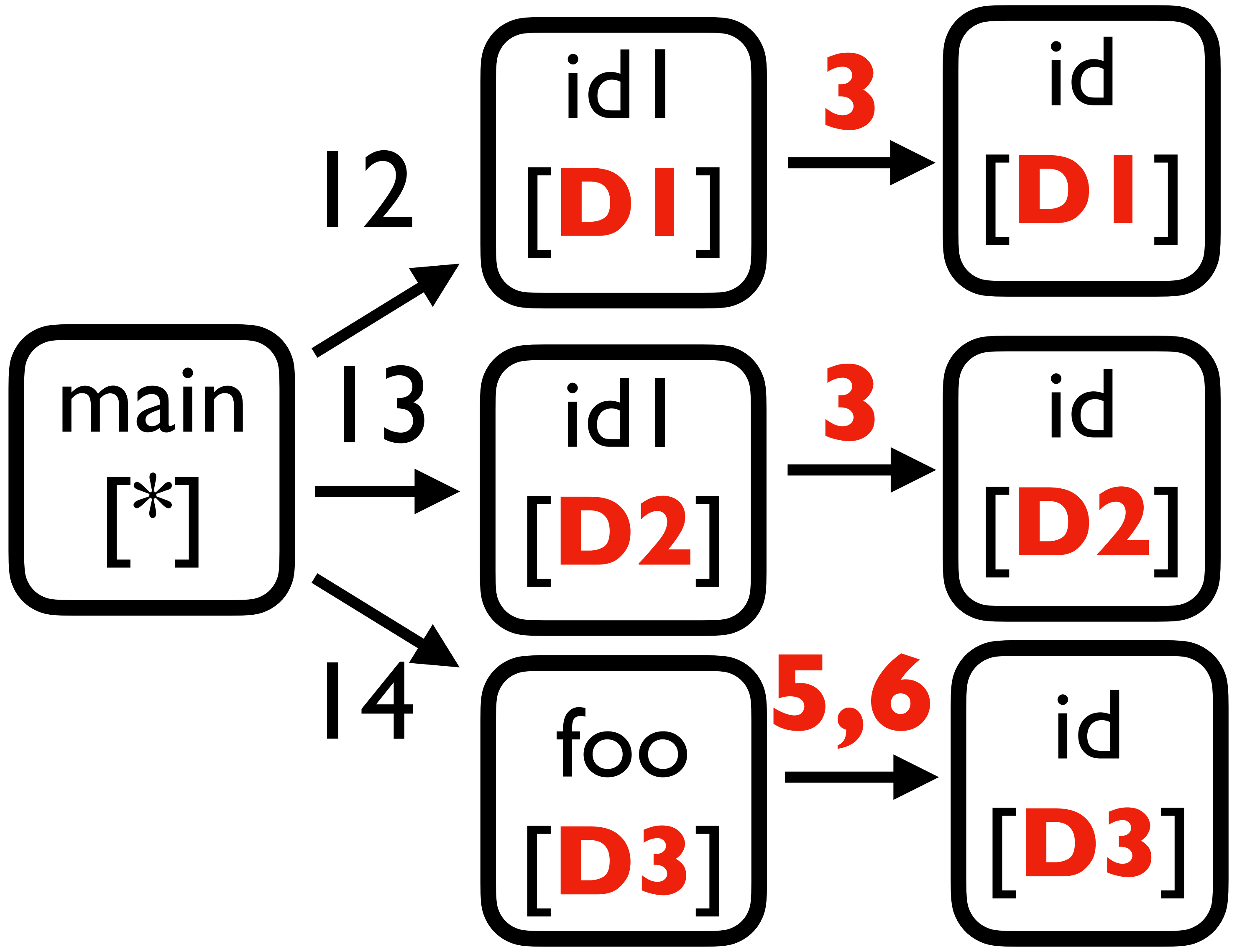
e context

I_2

- Property 2: different caller contexts imply different callee contexts

Intuition Behind Simulation ($I_1 \cup I_2$)

- Suppose given call-graph is produced from 1callH+T' and infer what T' is



- I_1: caller and callee methods have the same context

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

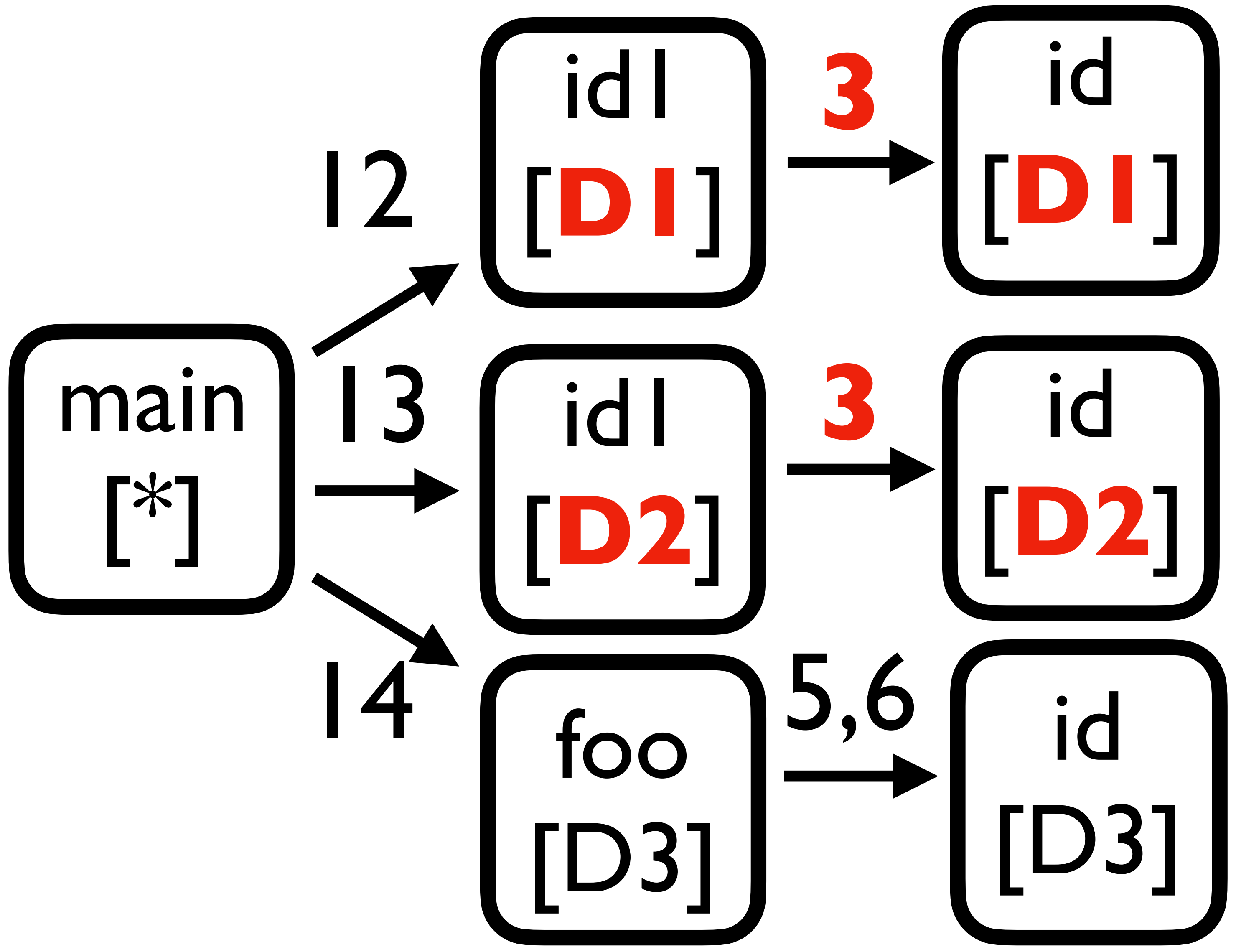
~~1objH+T (T = ∅)~~

1callH+T'

What is T'?

Intuition Behind Simulation ($I_1 \cup I_2$)

- Suppose given call-graph is produced from 1callH+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}

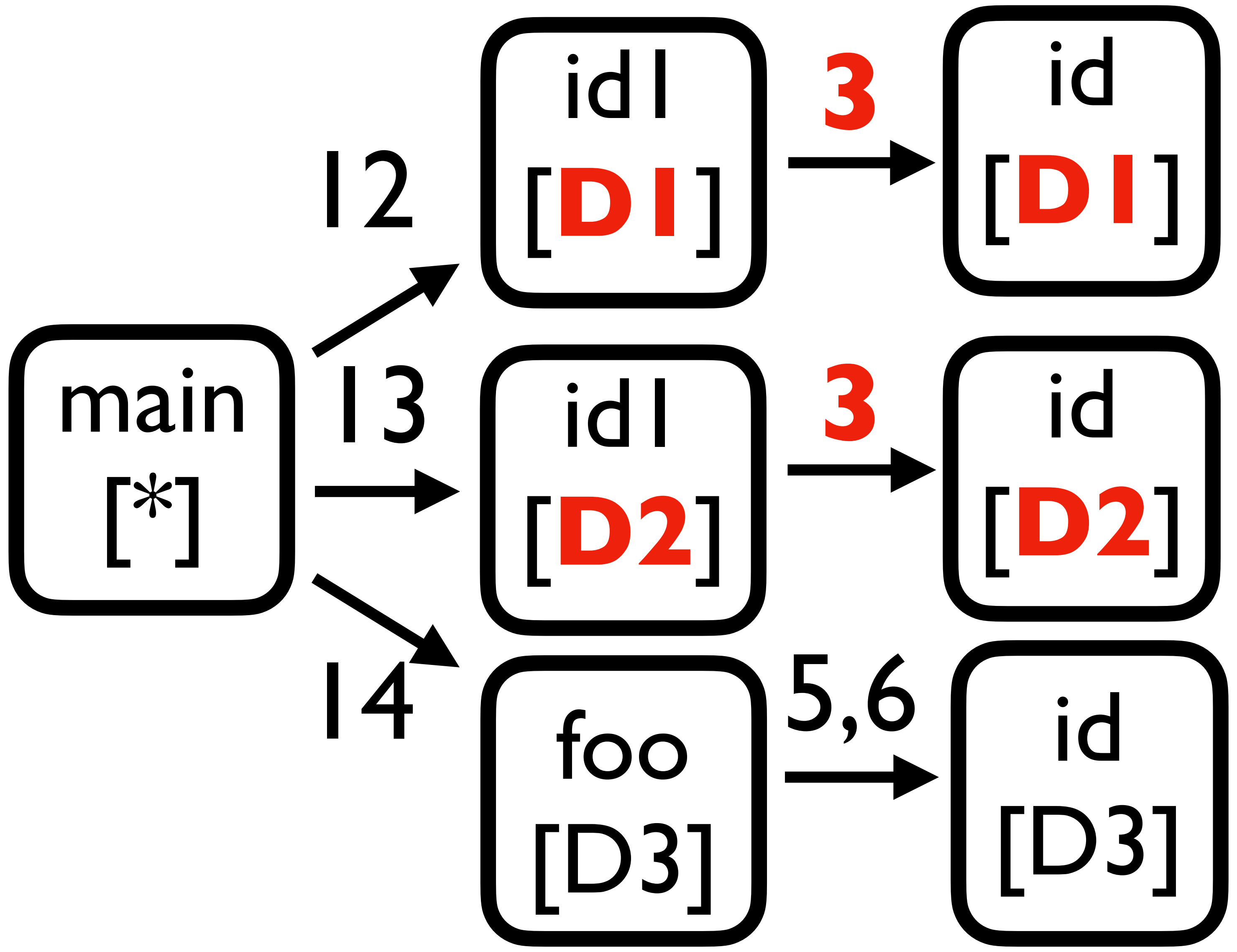
~~1objH+T (T = ∅)~~

1callH+T'

What is T'?

Intuition Behind Simulation ($I_1 \cup I_2$)

- Suppose given call-graph is produced from 1callH+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

$$I_2=\{3\}$$

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

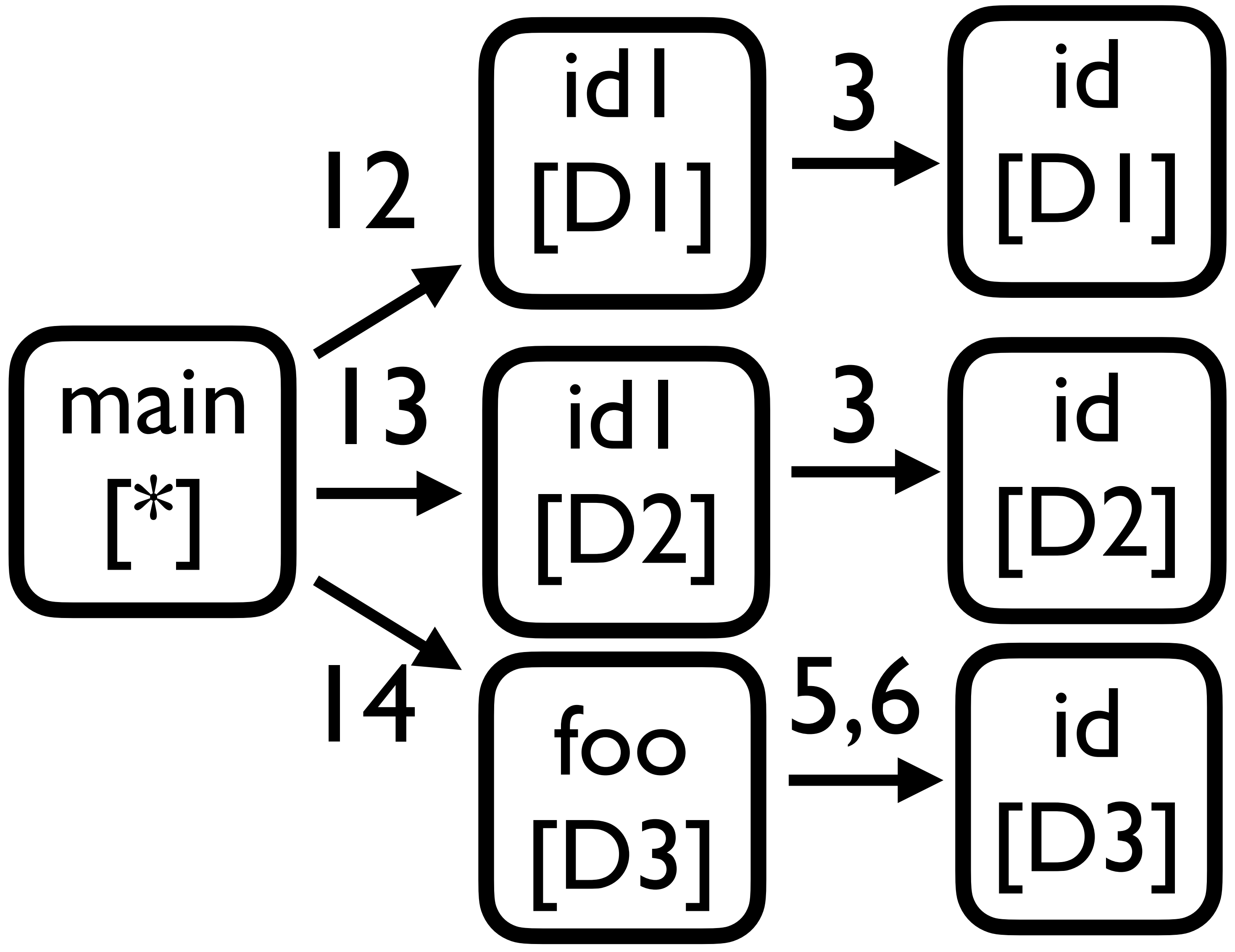
~~1objH+T (T = ∅)~~

1callH+T'

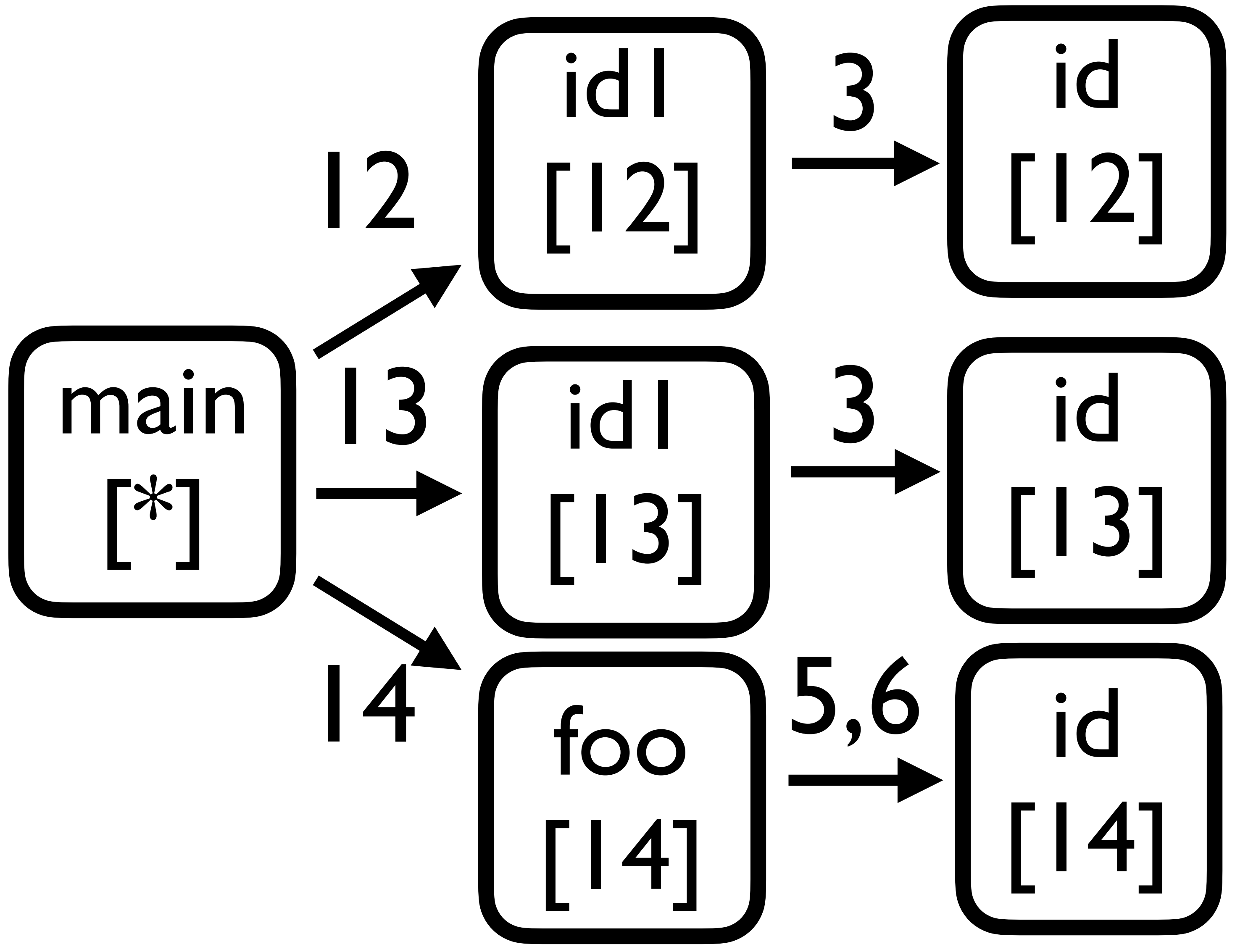
What is T'?

Intuition Behind Simulation $(I_1 \cup I_2)$

- Suppose given call-graph is produced from 1callH+T’ and infer what T’ is



1objH+T (T = ∅)

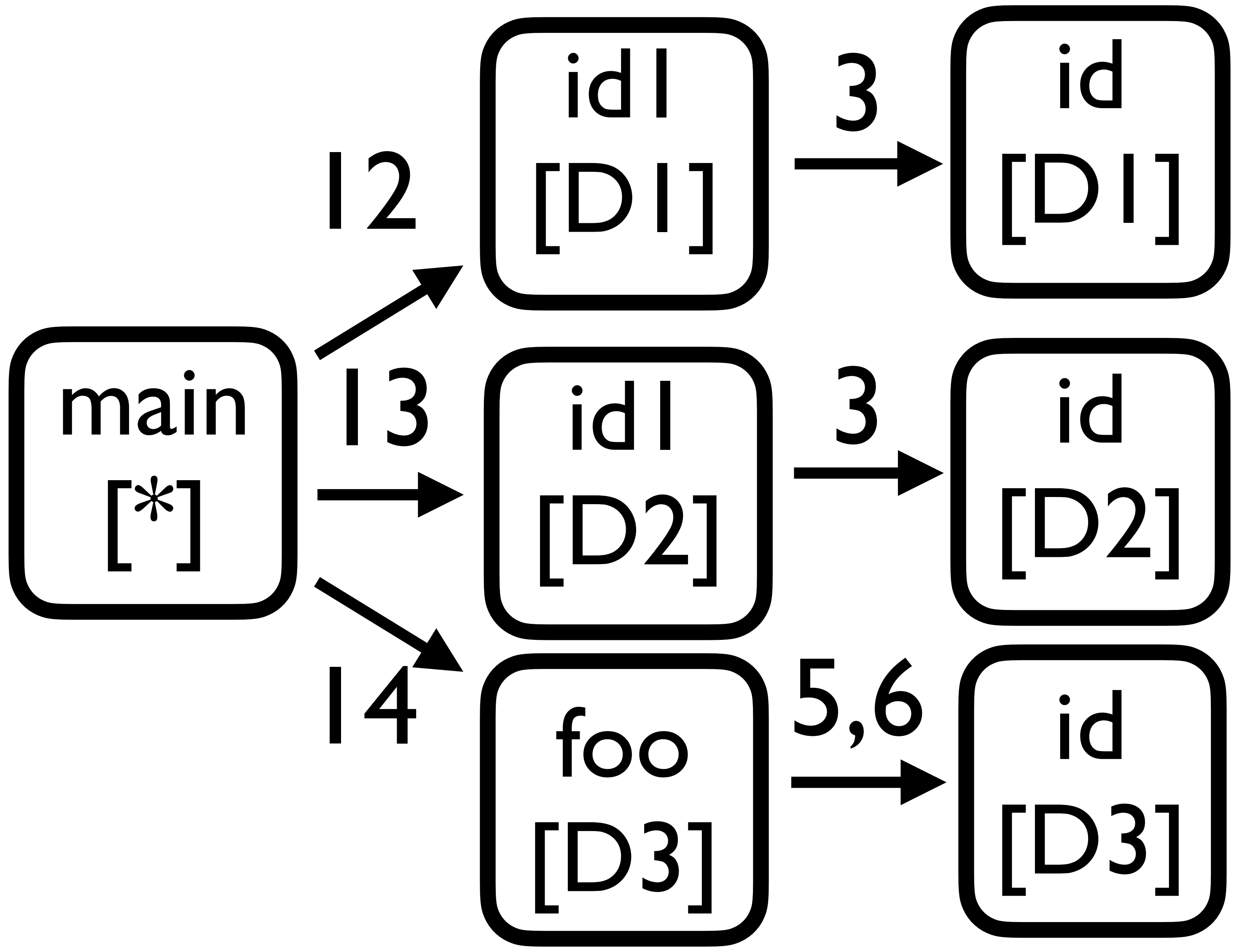


1callH+T’ (T’ = {3,5,6})

Intuition Behind Simulation ($I_1 \cup I_2$)

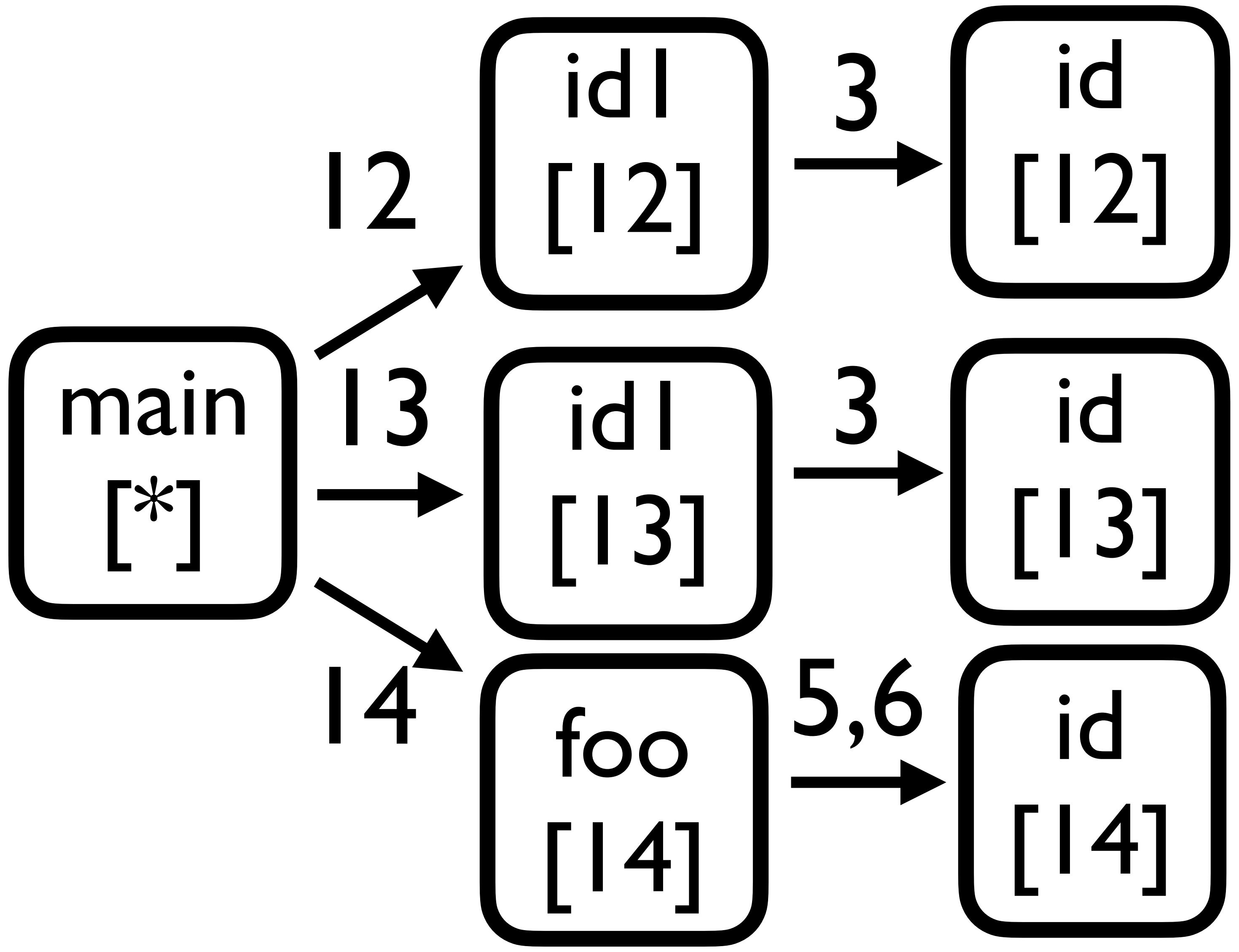
- Suppose given call-g d infer what T' is

Exactly the same analyses



1objH+T (T = ∅)

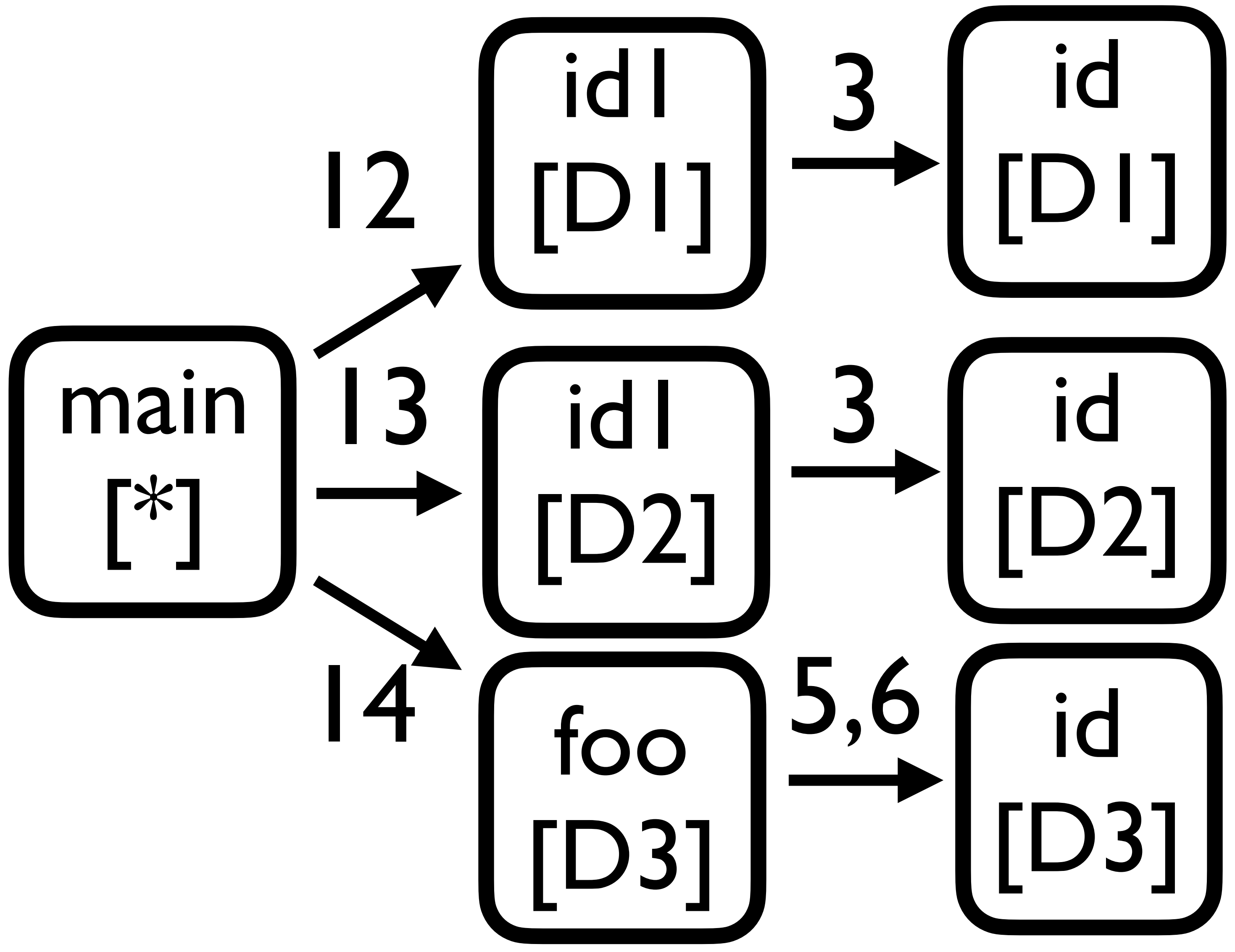
=



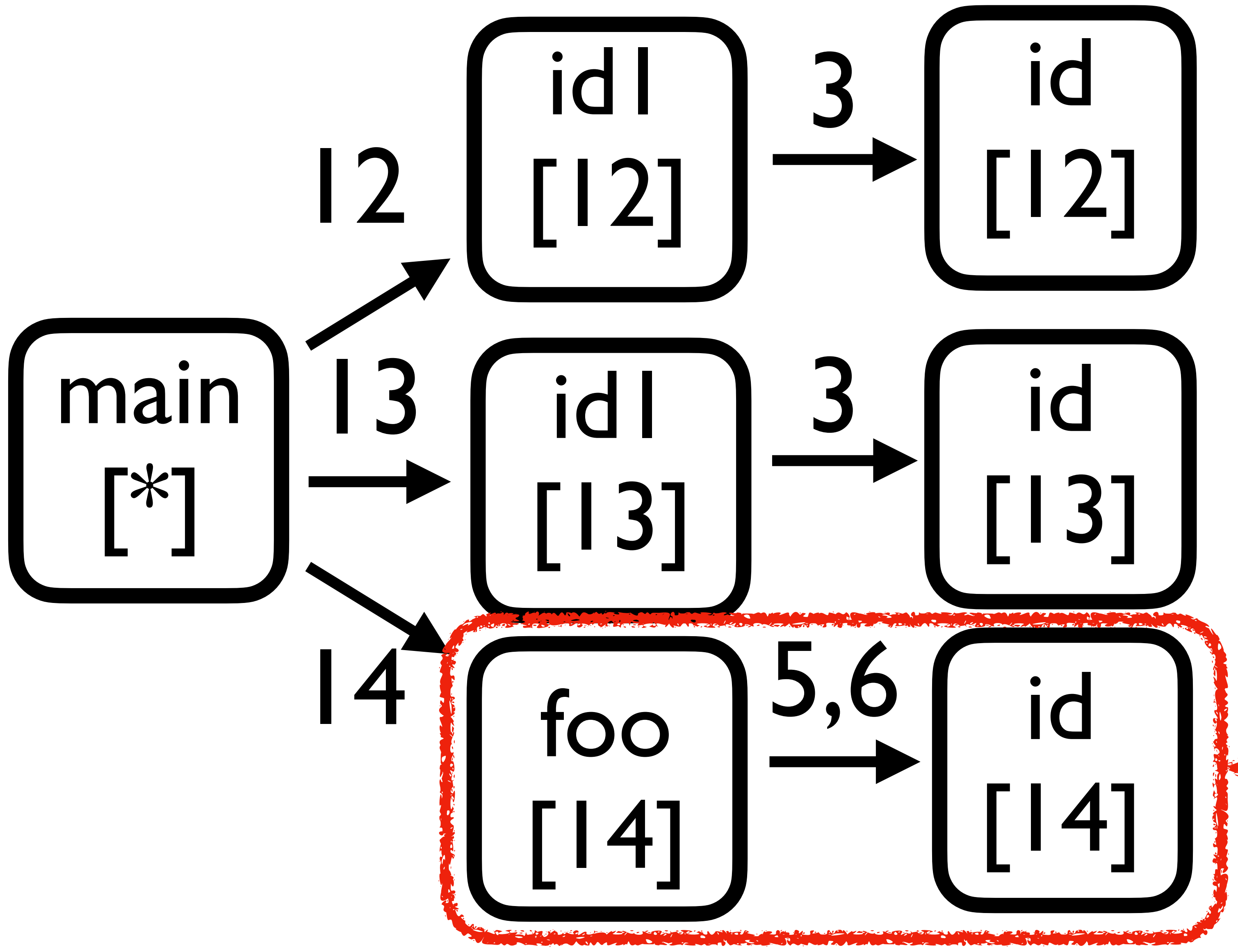
1callH+T' (T' = {3,5,6})

Intuition Behind Simulation ($I_1 \cup I_2$)

- Suppose given call-graph is produced from 1callH+T' and infer what T' is



1objH+T (T = ∅)

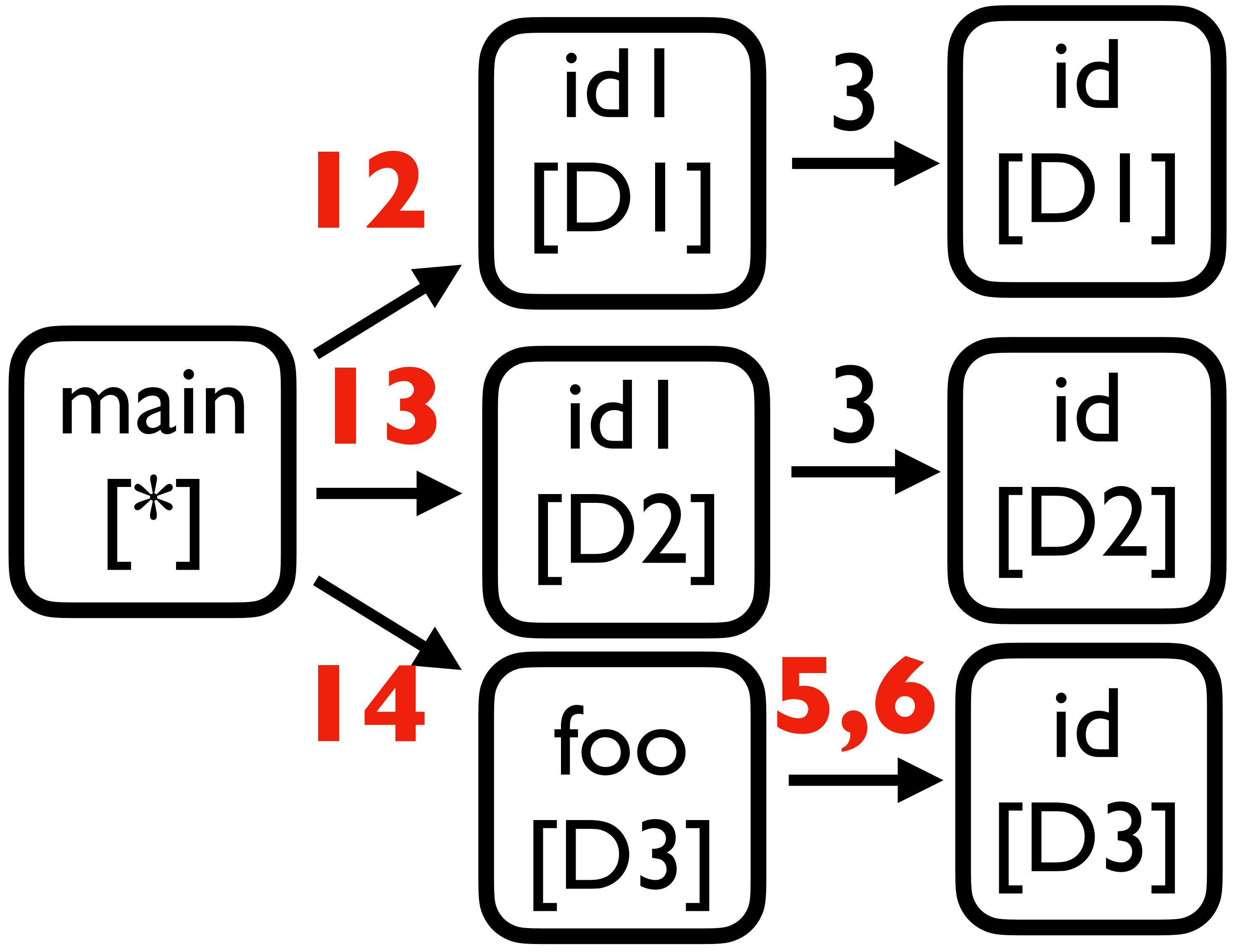


Necessity of I_3

1callH+T' (T' = {3,5,6})

Intuition Behind Simulation (I_3)

- I_3 : Tunneling should be avoided for improving precision



1objH+T (T = ∅)

- I_1: caller and callee methods have the same context

I_1={3,5,6}

- I_2: different caller ctx imply different callee ctx

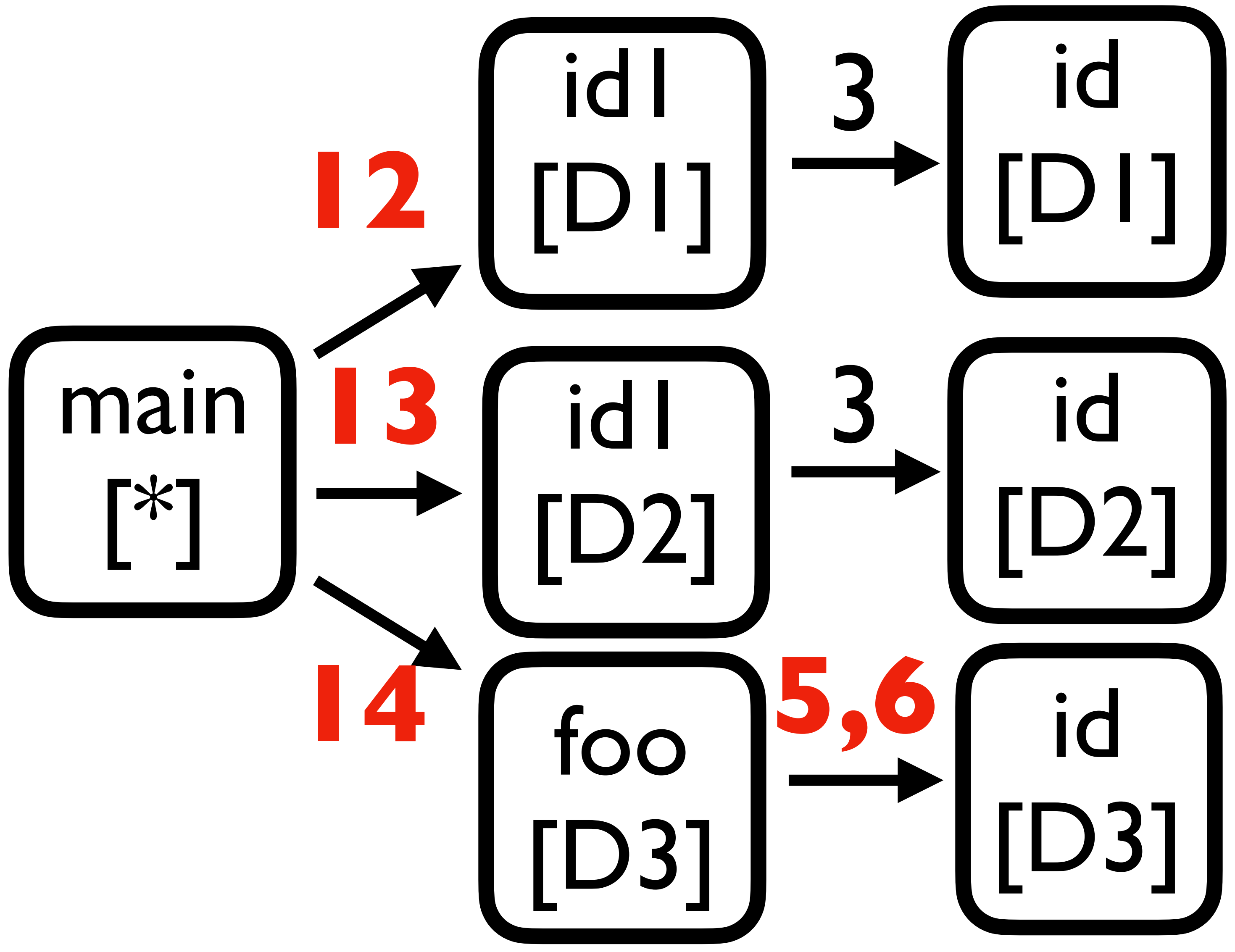
I_2={3}

- I_3: given object sensitivity produced only one context

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

Intuition Behind Simulation

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



1objH+T (T = ∅)

- 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

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

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

Technique 1: Simulation

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

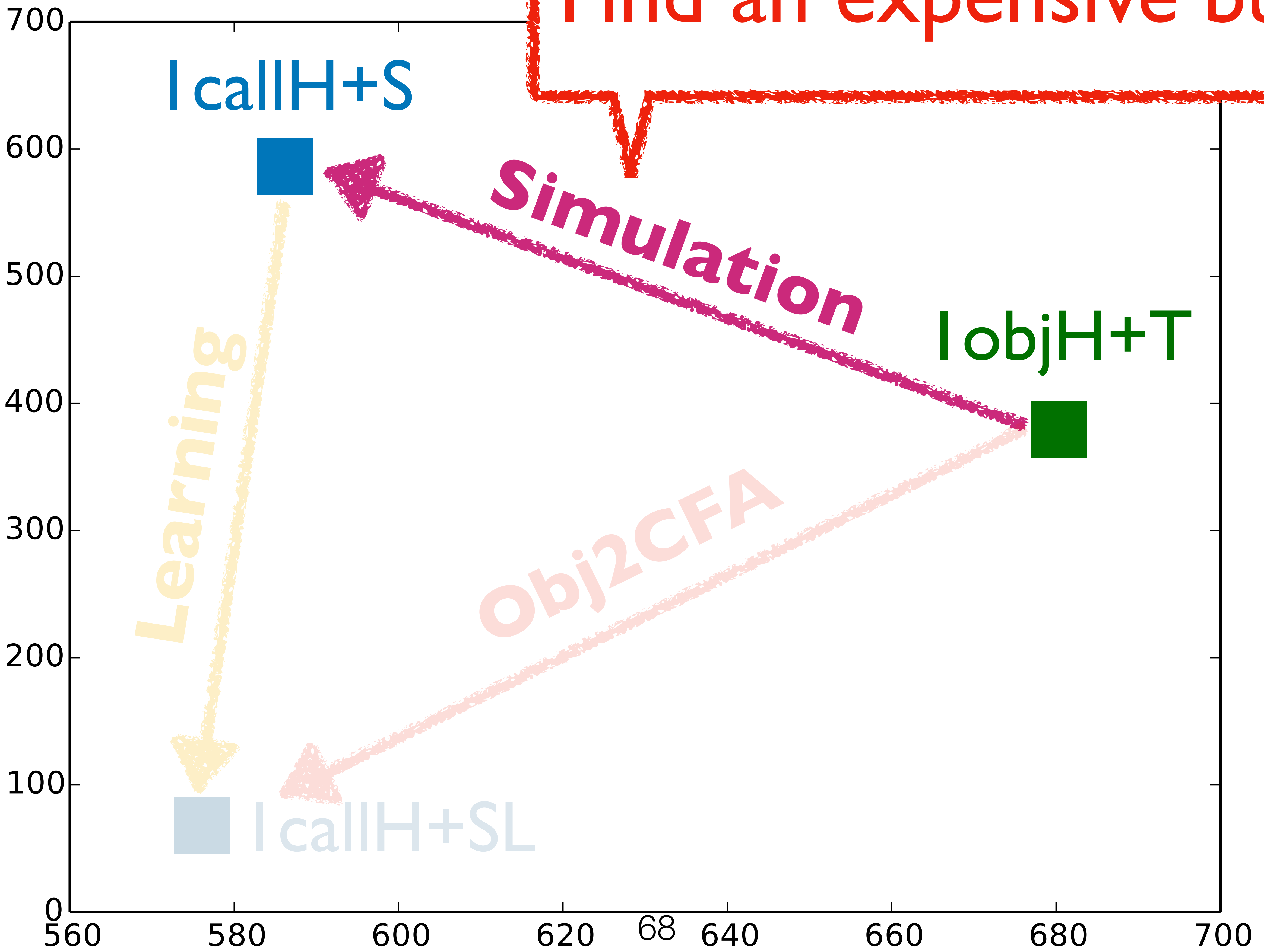
main
[*]
12
13
14
id1
[D1]
3
id
[D1]
id1
[D2]
3
id
[D2]
foo
[D3]
5,6
id
[D3]
Simulation
main
[*]
12
13
14
id1
[12]
3
id
[12]
id1
[13]
3
id
[13]
foo
[14]
6
id
[5]
5
id
[6]

1objH+T (T = ∅)

1callH+T’ (T’ = {3})

Our Technique : **Obj2CFA**

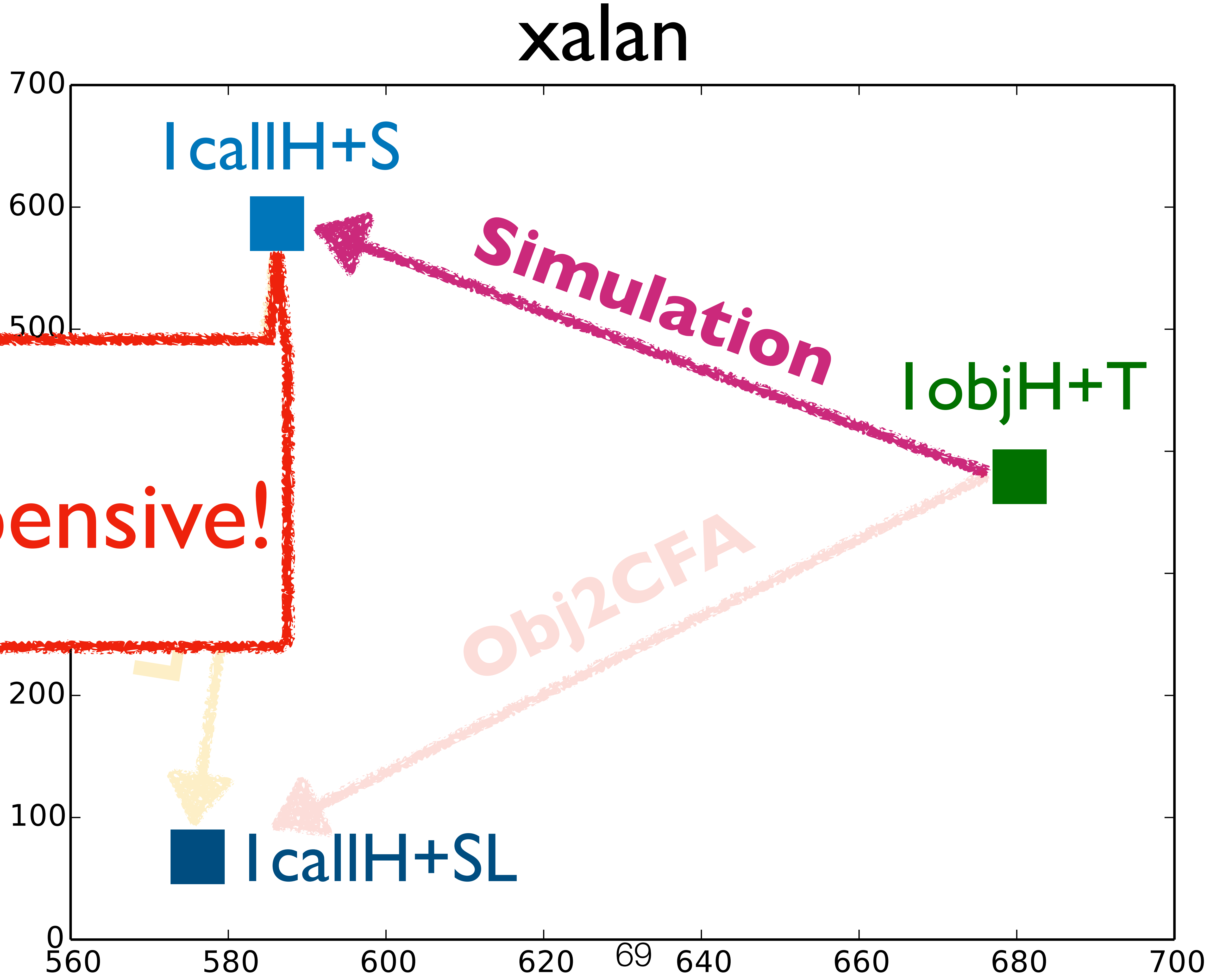
- **Obj2CFA** consists of **simulation** and simulation-guided **learning**



Find an expensive but more precise CFA

Our Technique : **Obj2CFA**

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

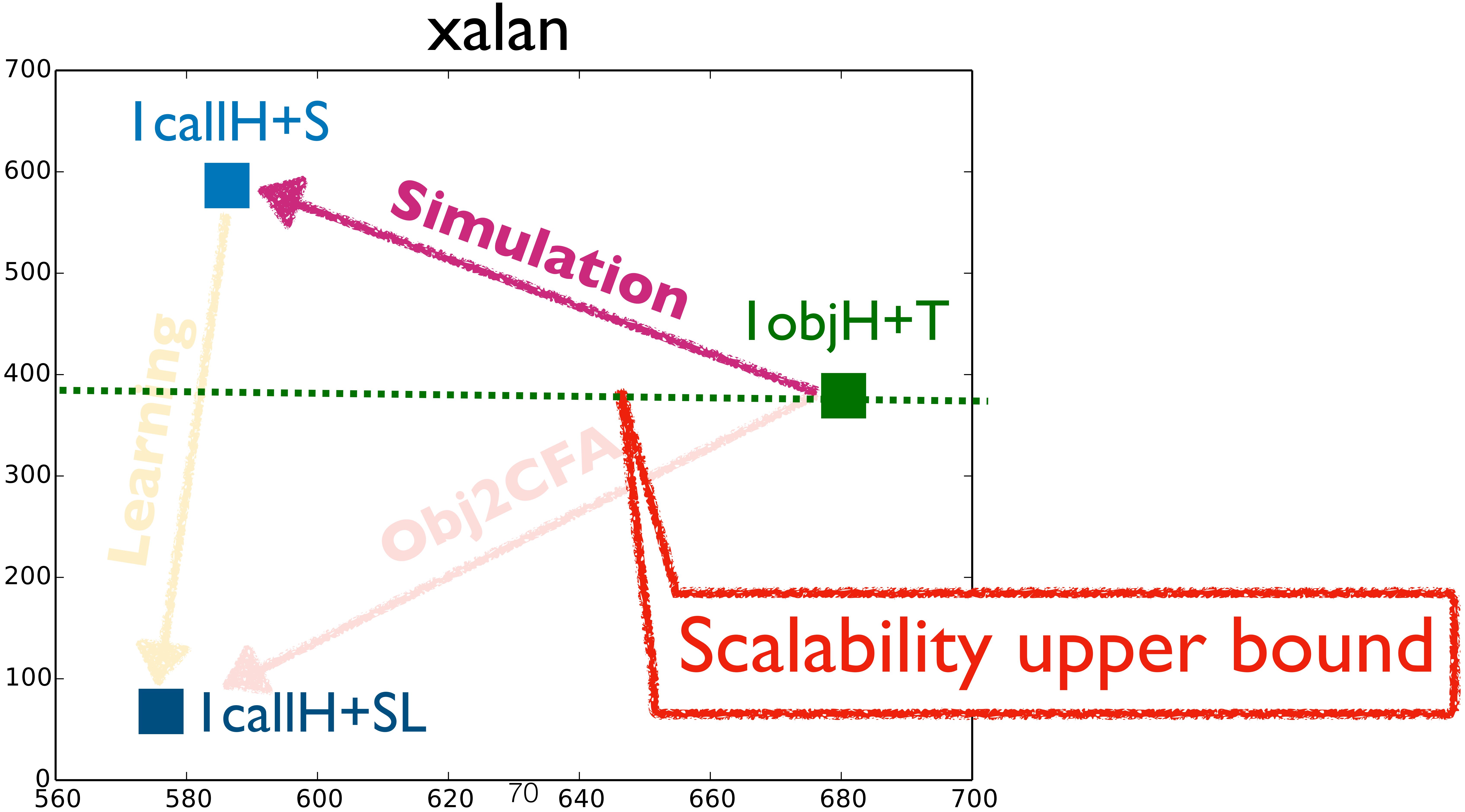


Limitation

Simulation is expensive!

Our Technique : **Obj2CFA**

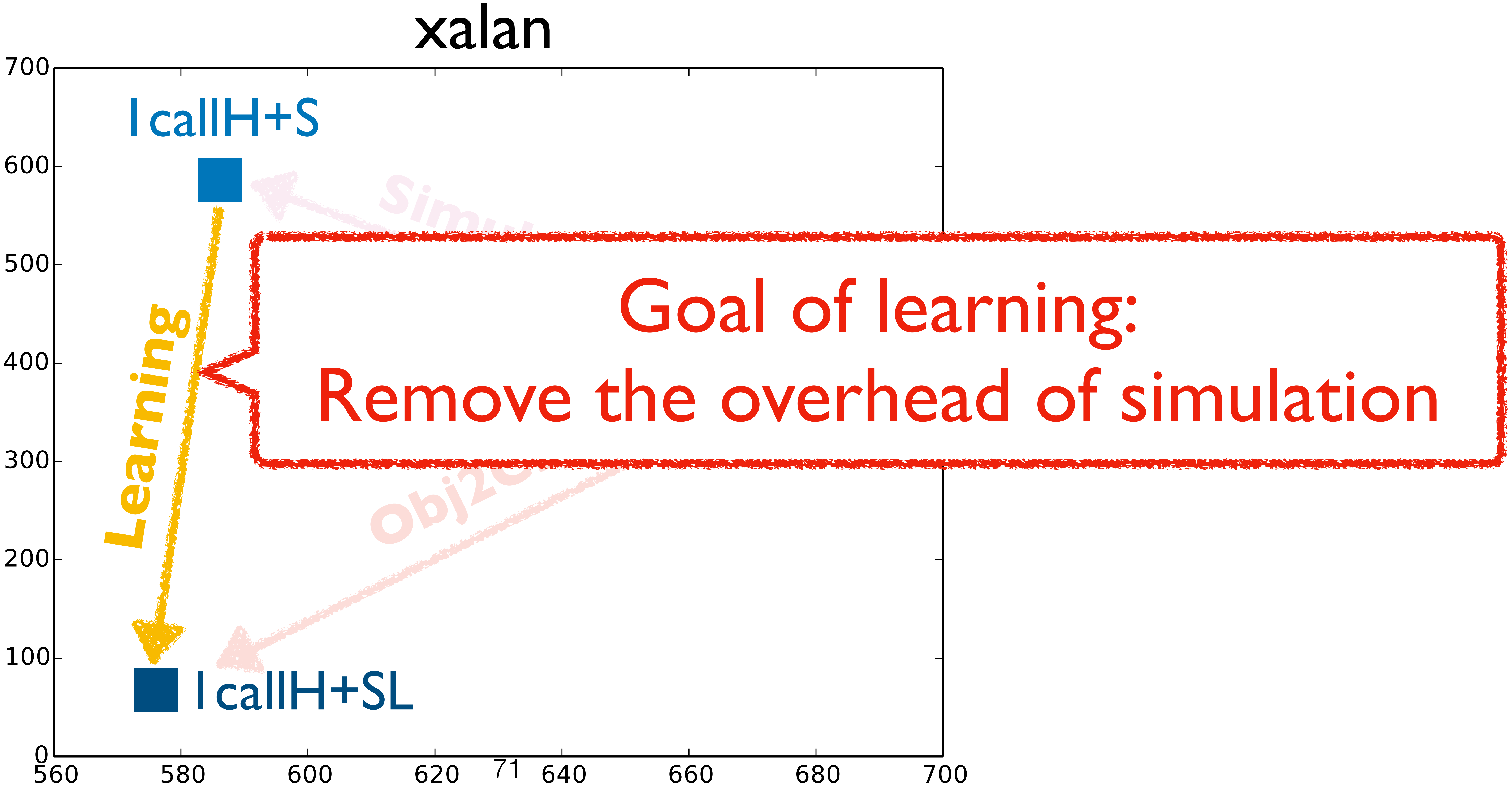
- **Obj2CFA** consists of **simulation** and simulation-guided **learning**



Scalability upper bound

Our Technique : **Obj2CFA**

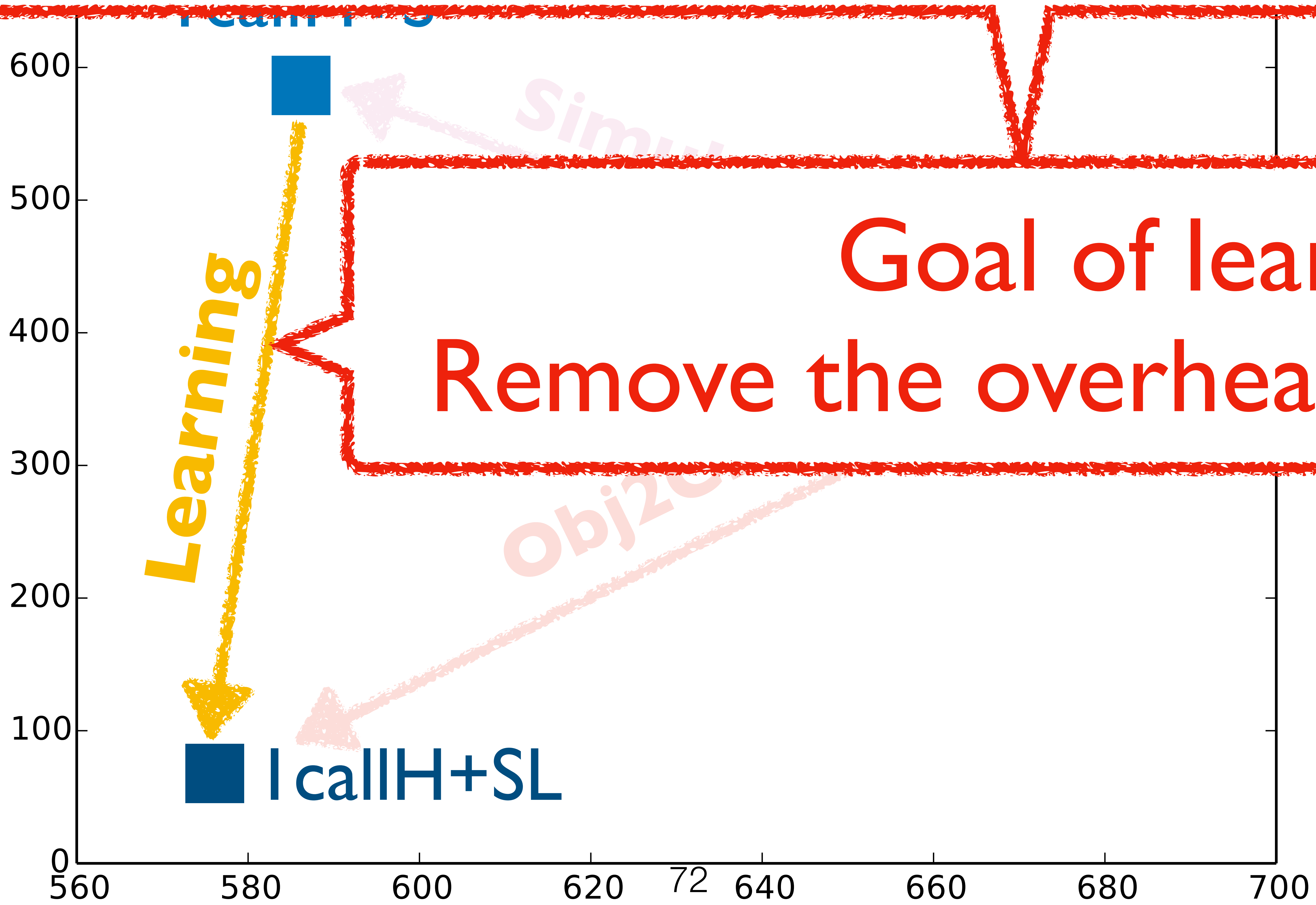
- **Obj2CFA** consists of **simulation** and simulation-guided **learning**



Goal of learning:
Remove the overhead of simulation

Our Technique : Obj2CFA

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

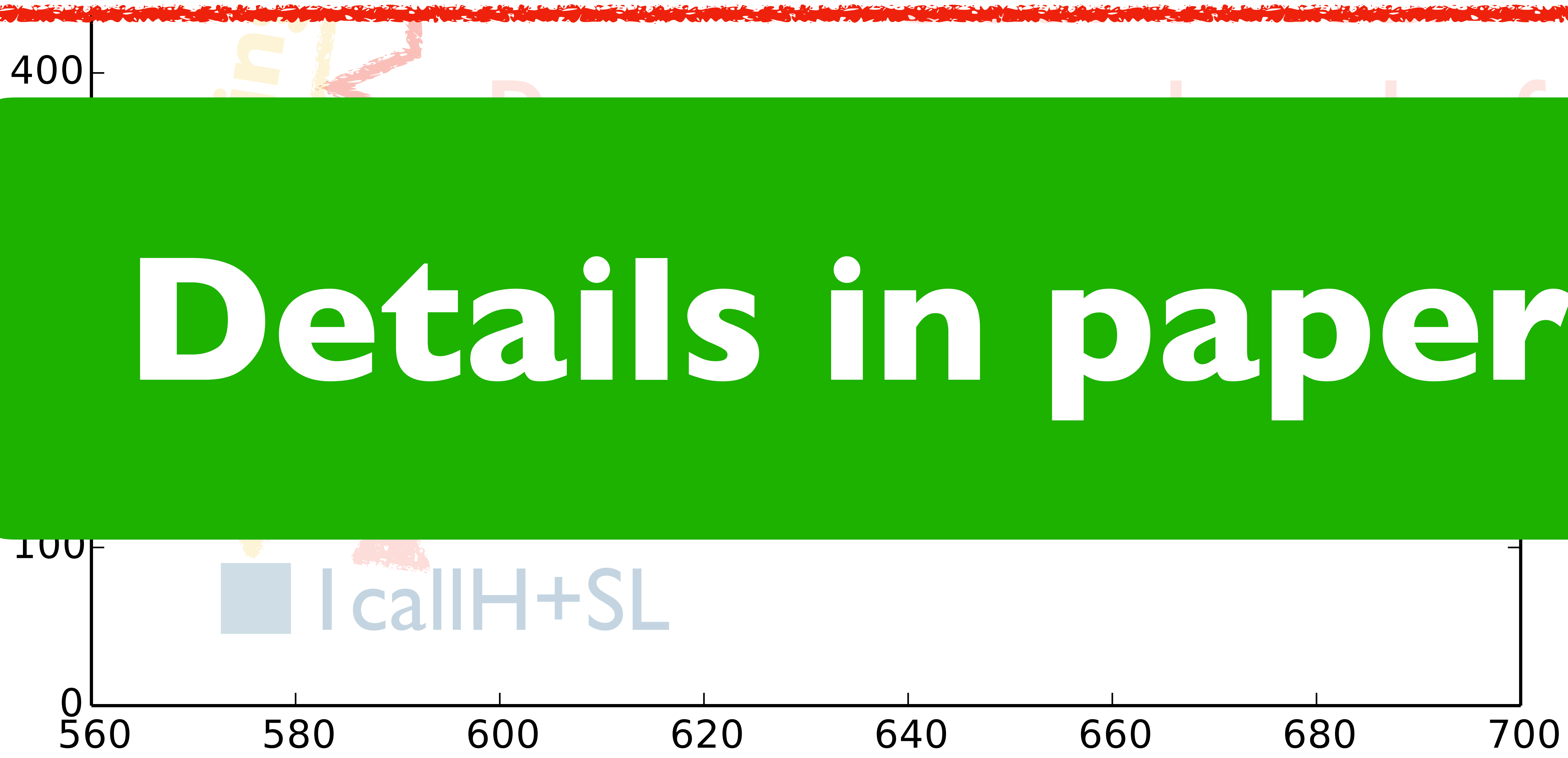


Goal of learning:
Remove the overhead of simulation

Our Technique : Obj2CFA

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

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

2009 (OOPSLA)

Pick Your Contexts Well: Understanding O

The Making of a Precise and Scalable Pointer

Yannis Smaragdakis Martin Bravenboer

2011 (POPL)

Hybrid Context-Sensitivity for Points-To A

George Kastrinis Yannis Smaragdakis

Department of Informatics, University of Athens

2013 (PLDI)

Introspective Analysis: Context-Sensitivity, Across th

Yannis Smaragdakis George Kastrinis George Balatsouras

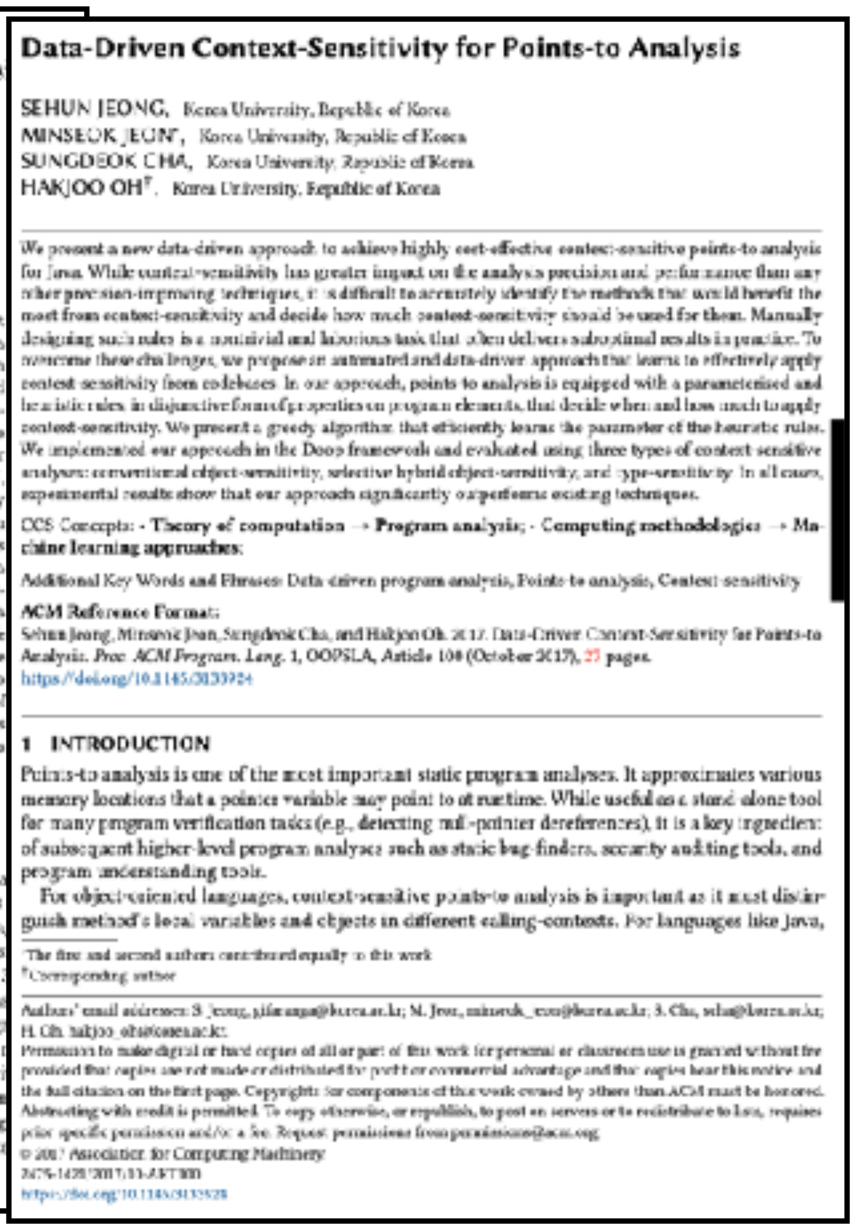
Department of Informatics, University of Athens

2014 (PLDI)

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

Tian Tan, Yue Li, and Jingling Xue

2016 (SAS)


Data-Driven Context-Sensitivity for Points-to Analysis

SEHUN JEONG, Korea University, Republic of Korea
MINSEOK JEON, Korea University, Republic of Korea
SUNGDEOK CHA, Korea University, Republic of Korea
HAKJOO OH, Korea University, Republic of Korea

2017 (OOPSLA)

...

Setting

- Doop
  - Pointer analysis framework for Java

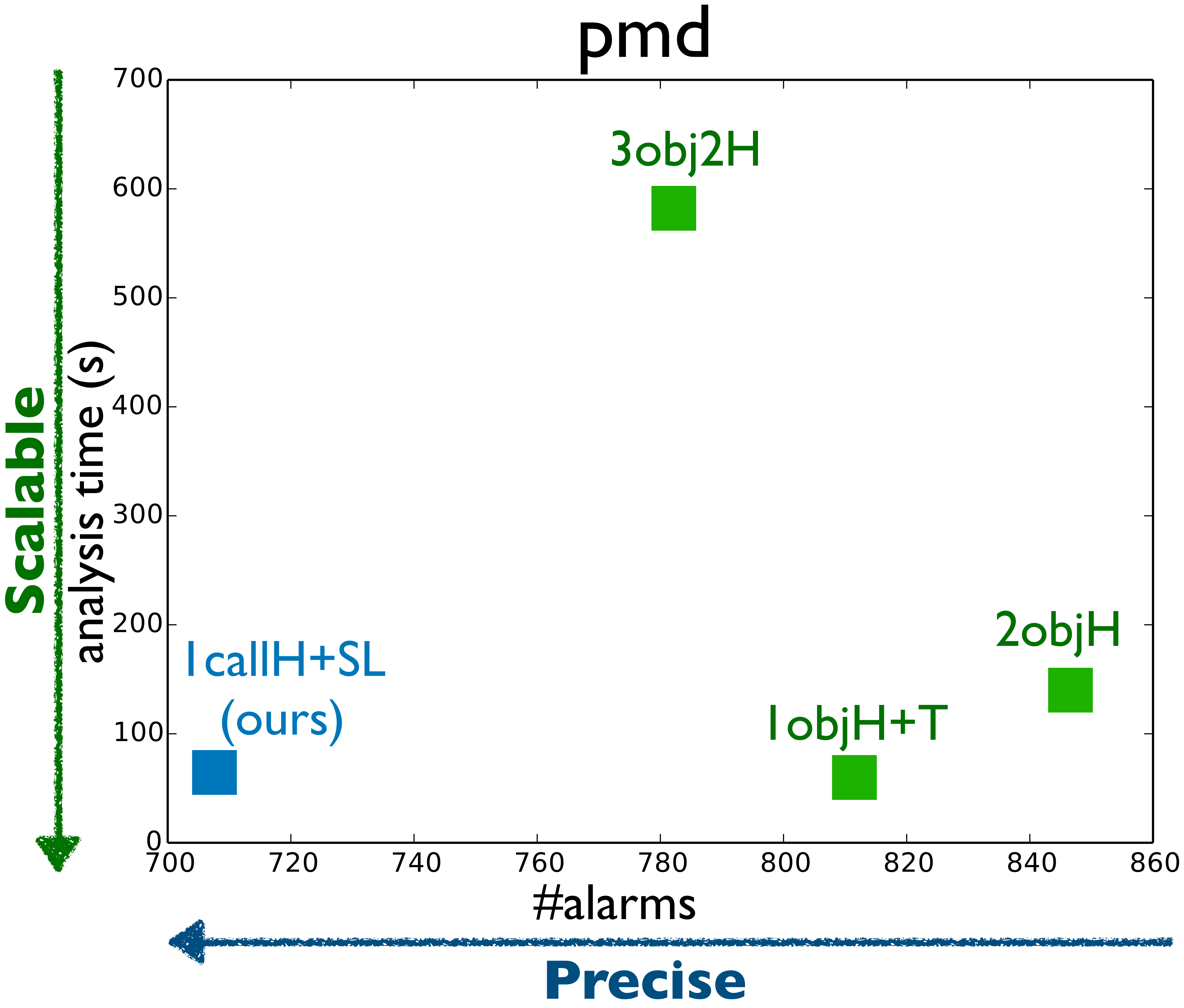
- Research Question: which one is better?

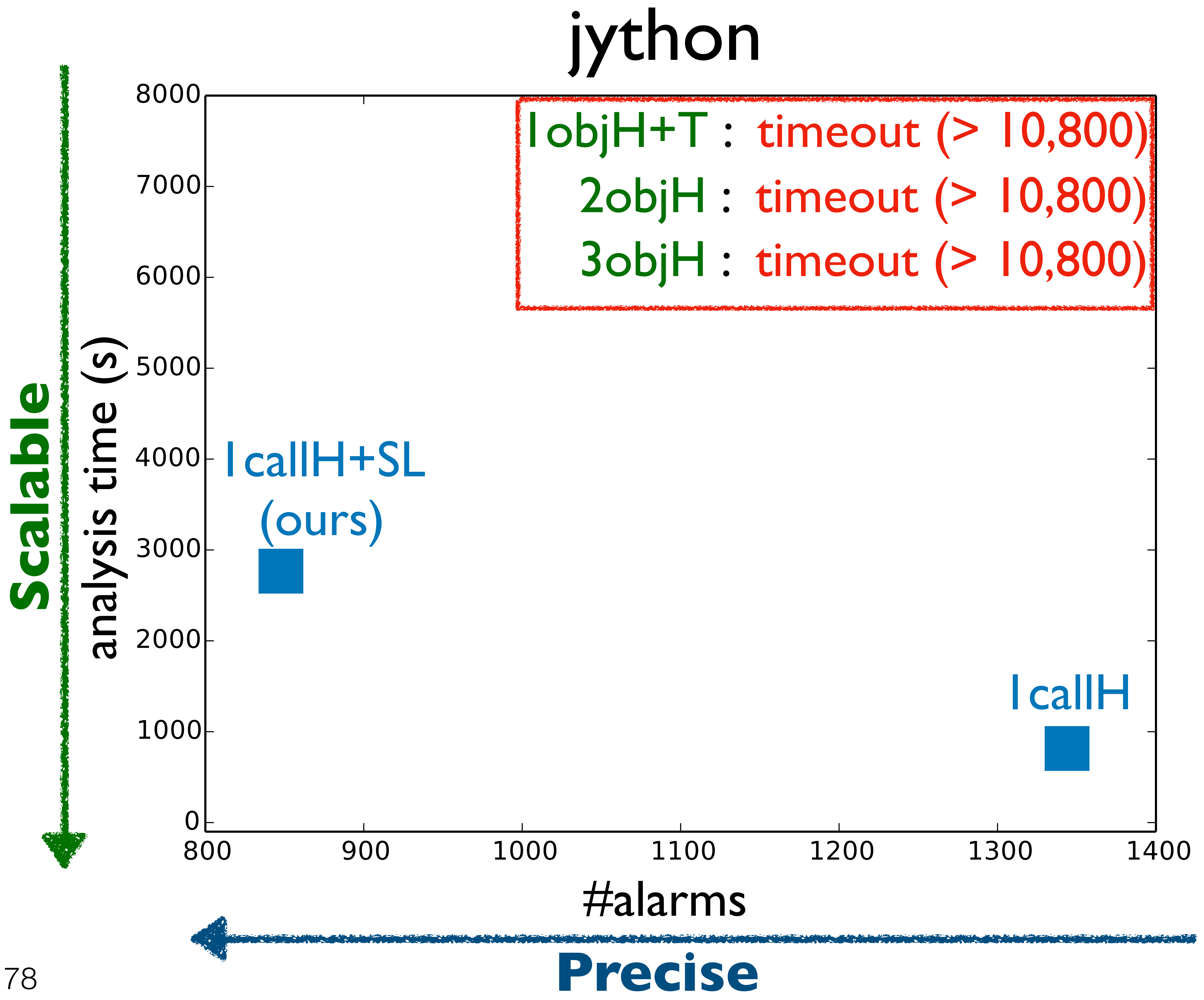
Call-site sensitivity vs Object sensitivity

Context tunneling is included

Call-site Sensitivity vs Object Sensitivity

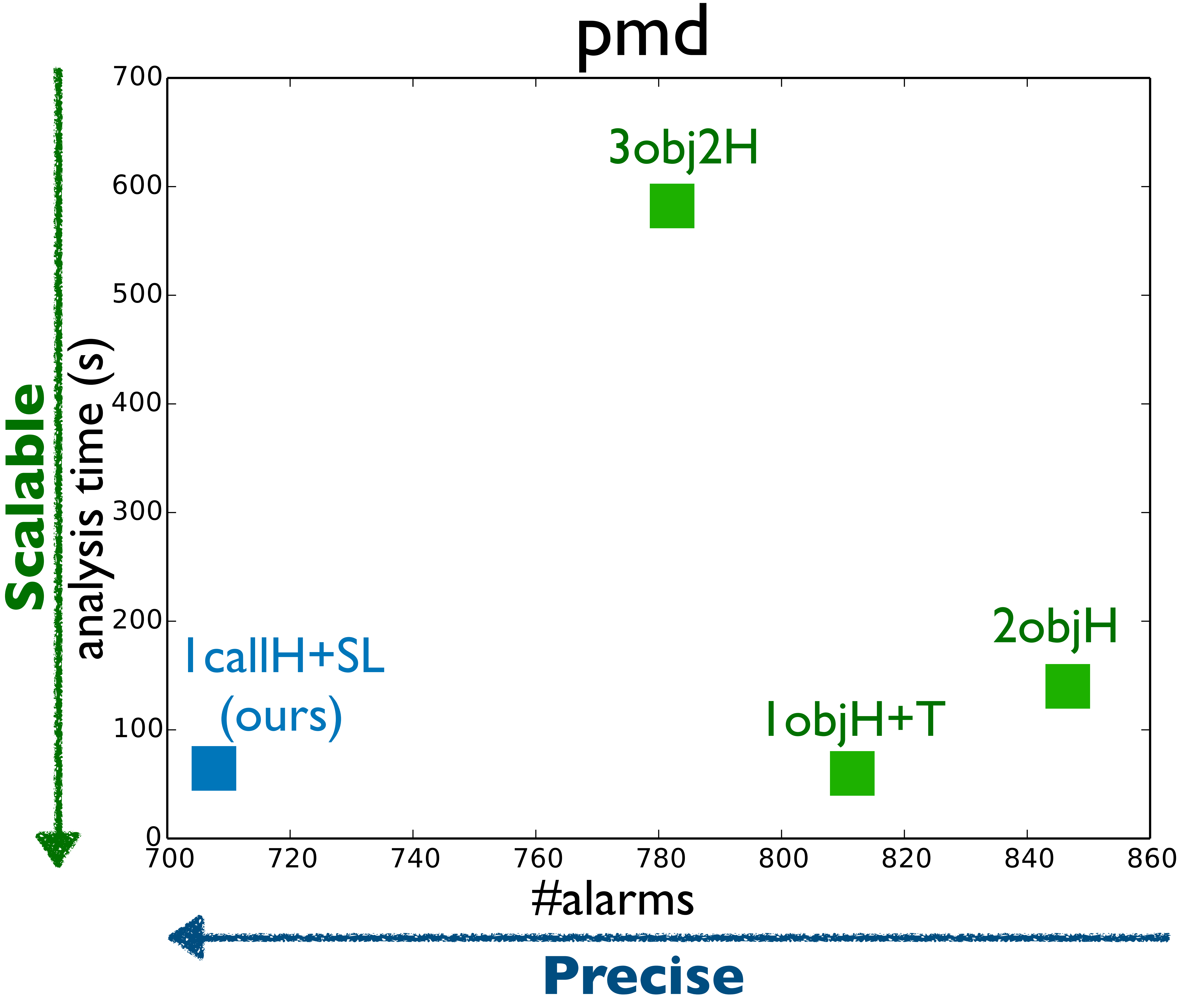
- 1callH+SL (ours) is more precise and scalable than the existing object sensitivities

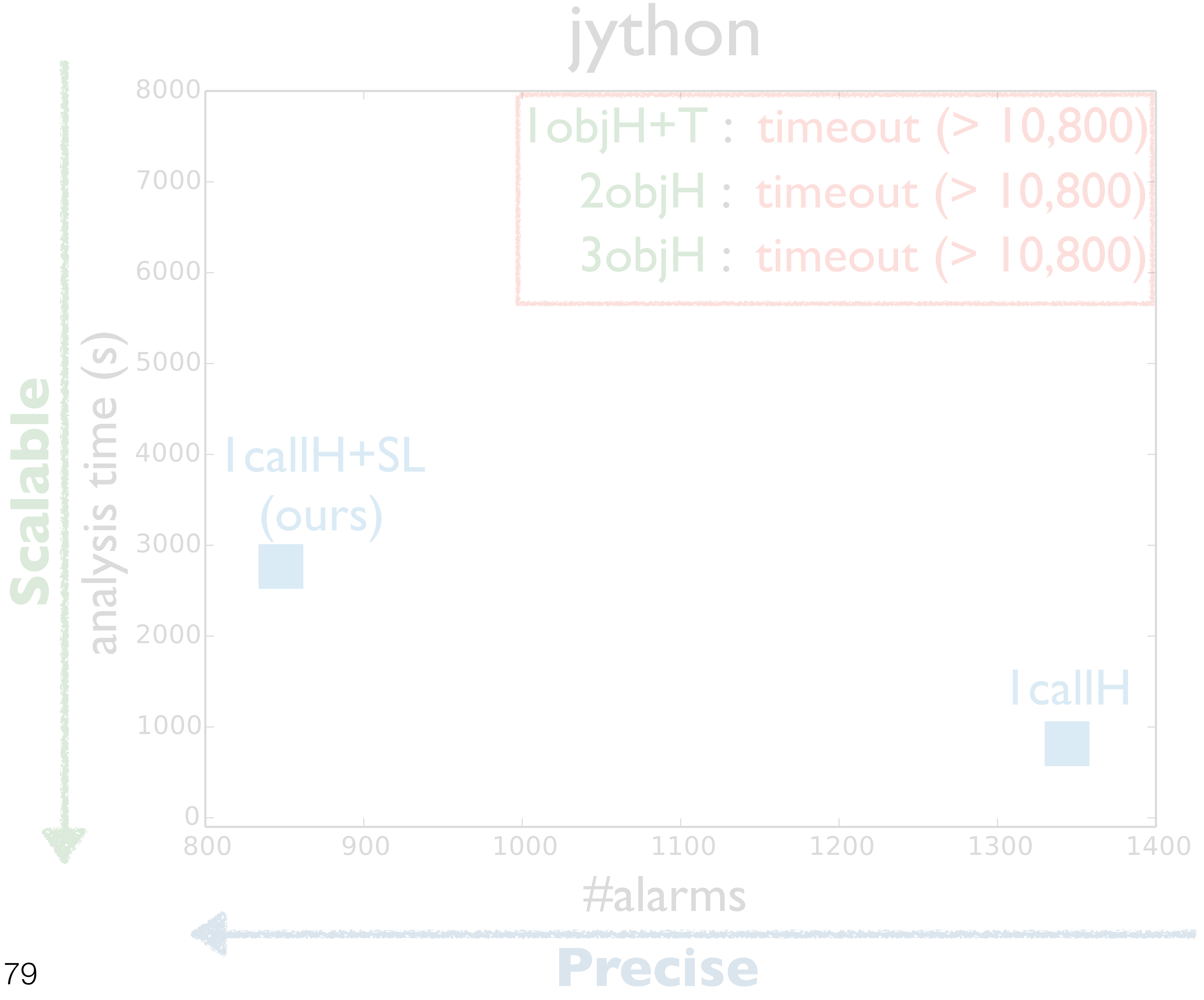




Call-site Sensitivity vs Object Sensitivity

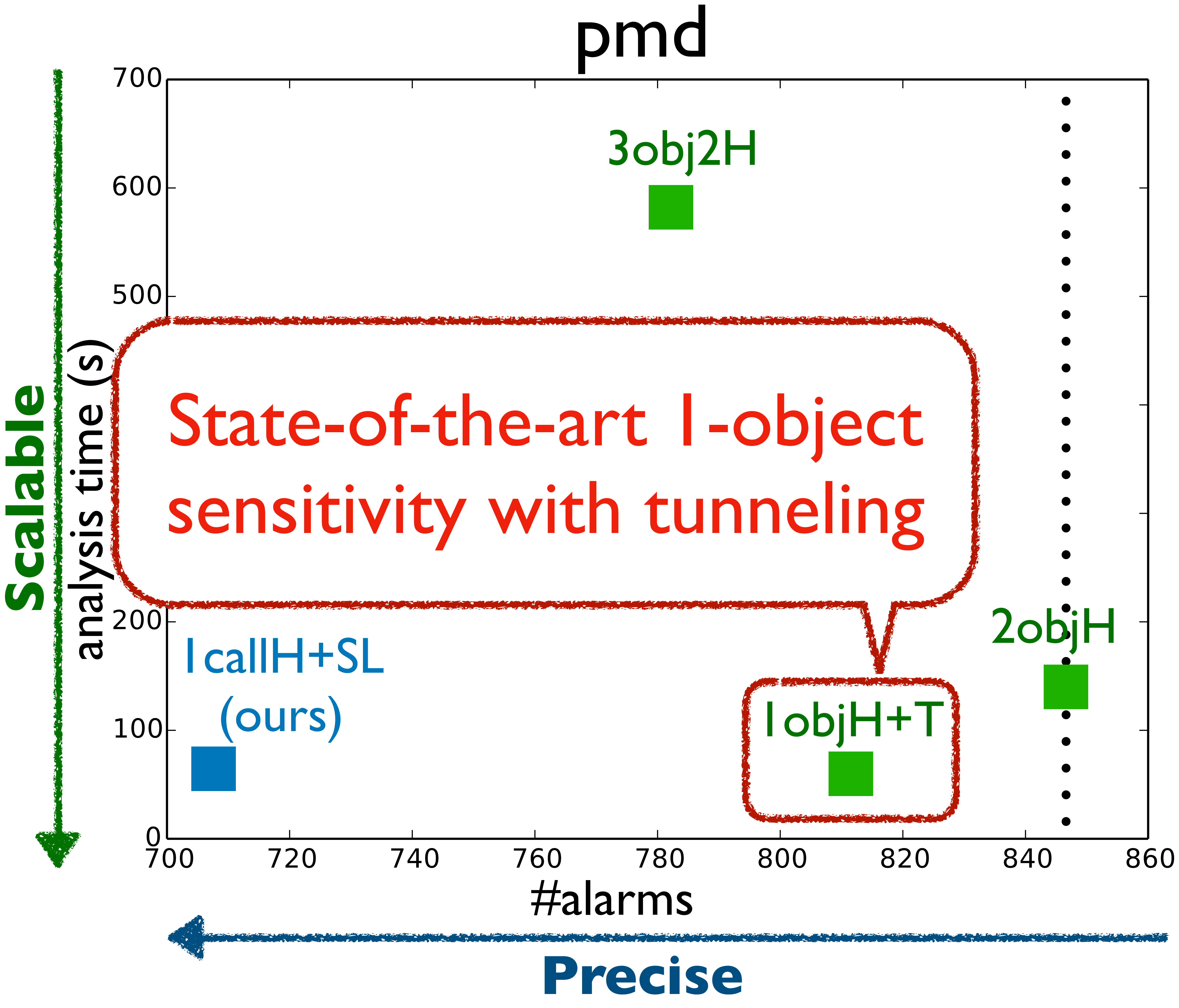
- 1callH+SL (ours) is more precise and scalable than the existing object sensitivities



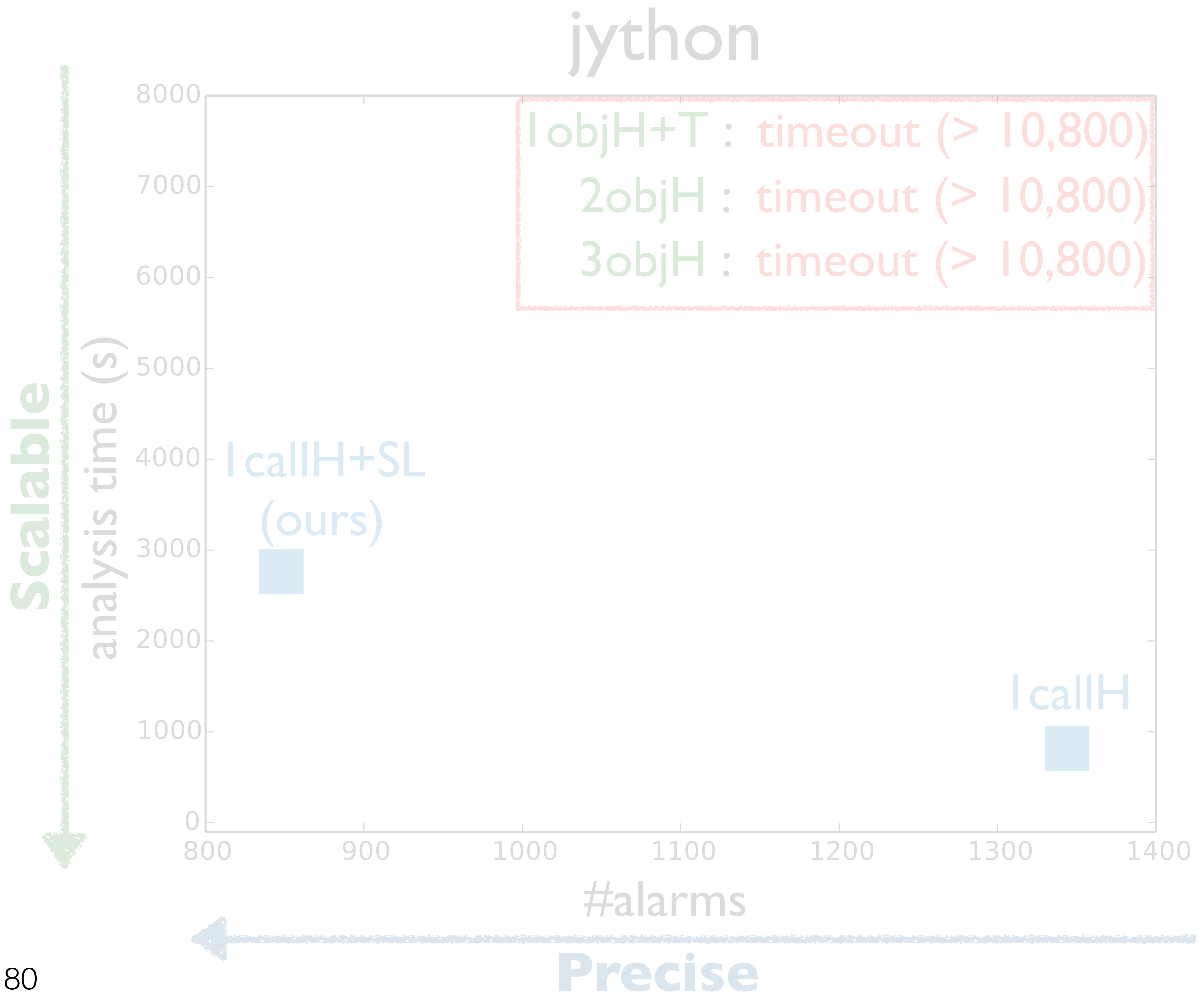


Call-site Sensitivity vs Object Sensitivity

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

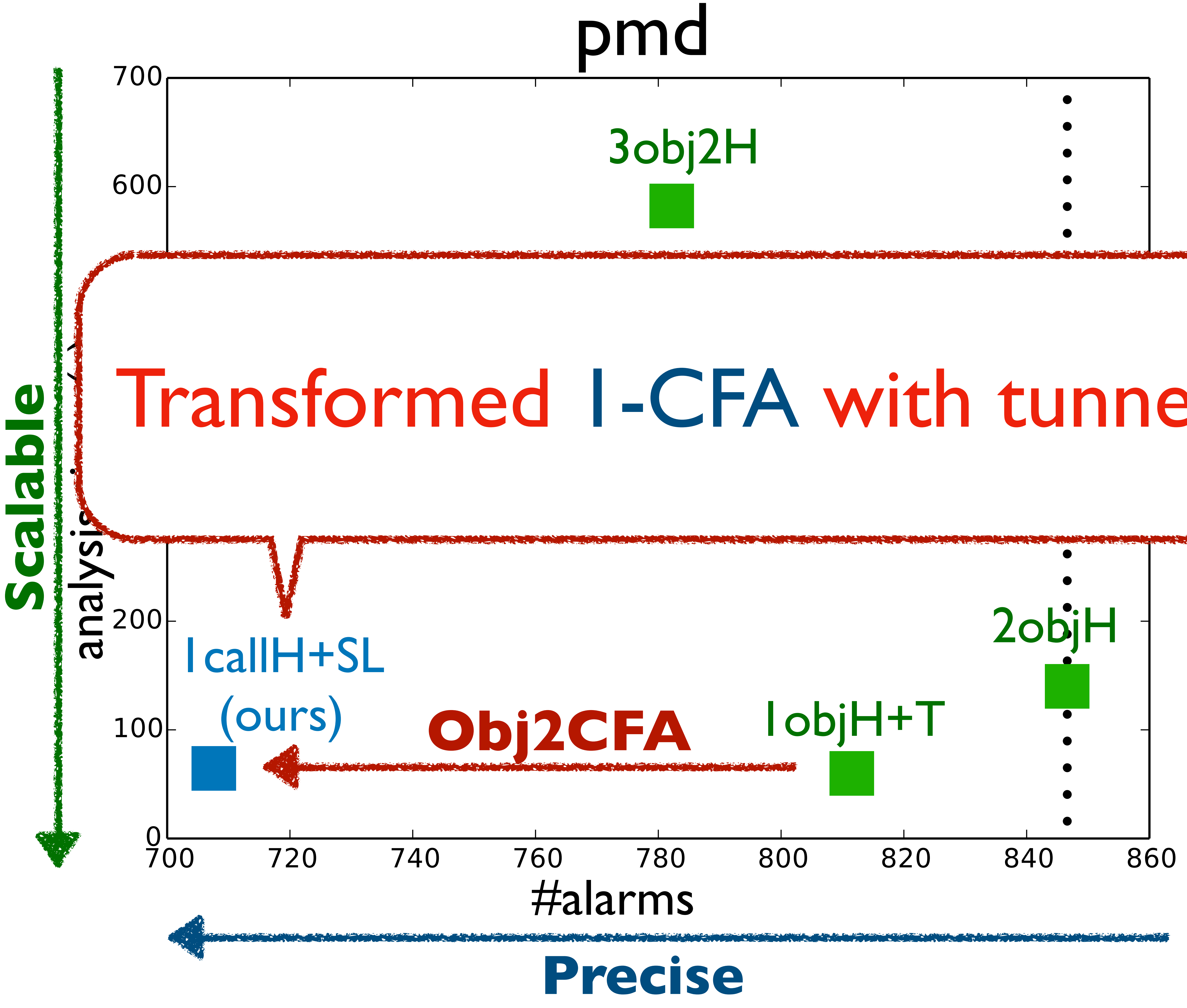


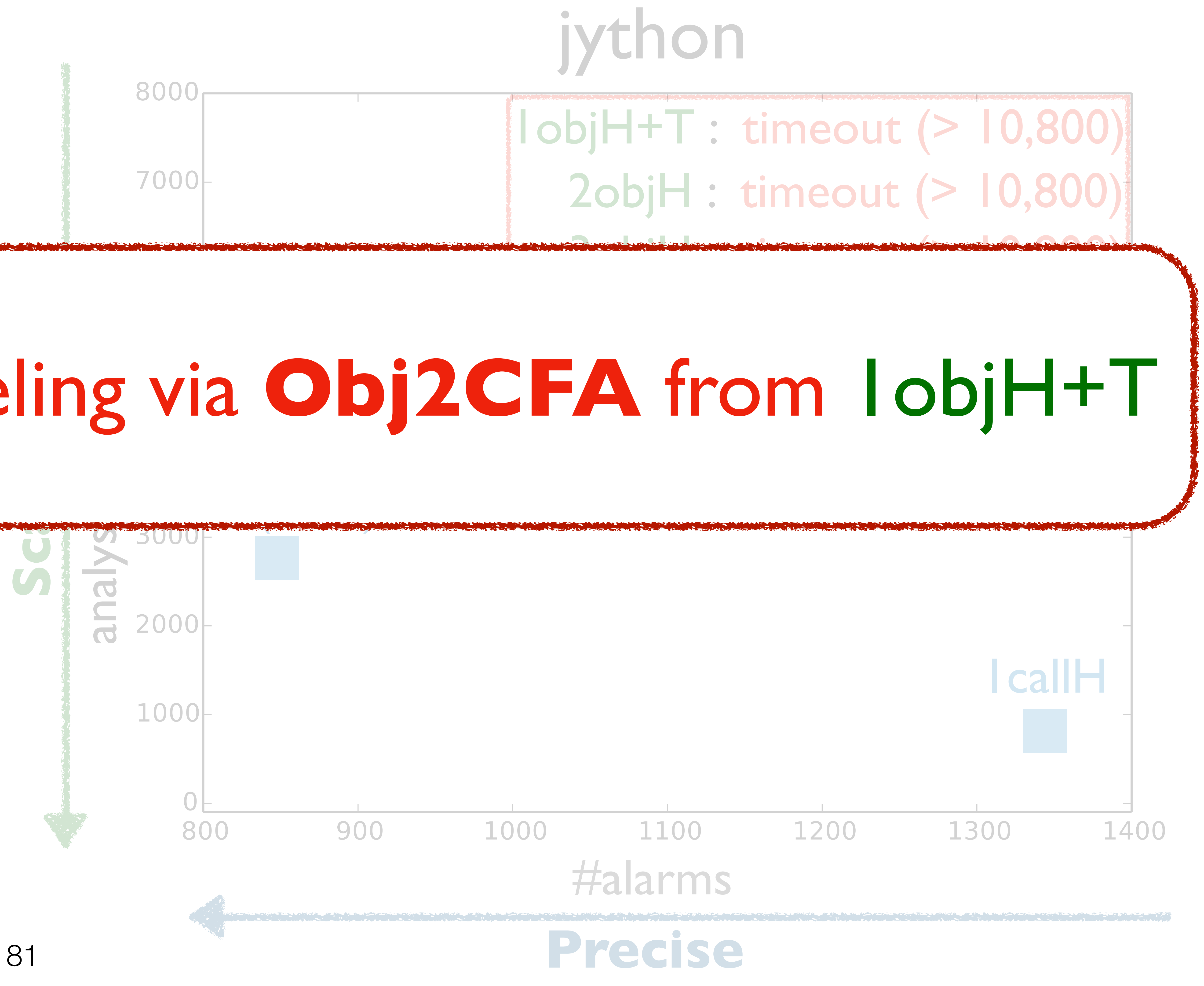
State-of-the-art 1-object sensitivity with tunneling



Call-site Sensitivity vs Object Sensitivity

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

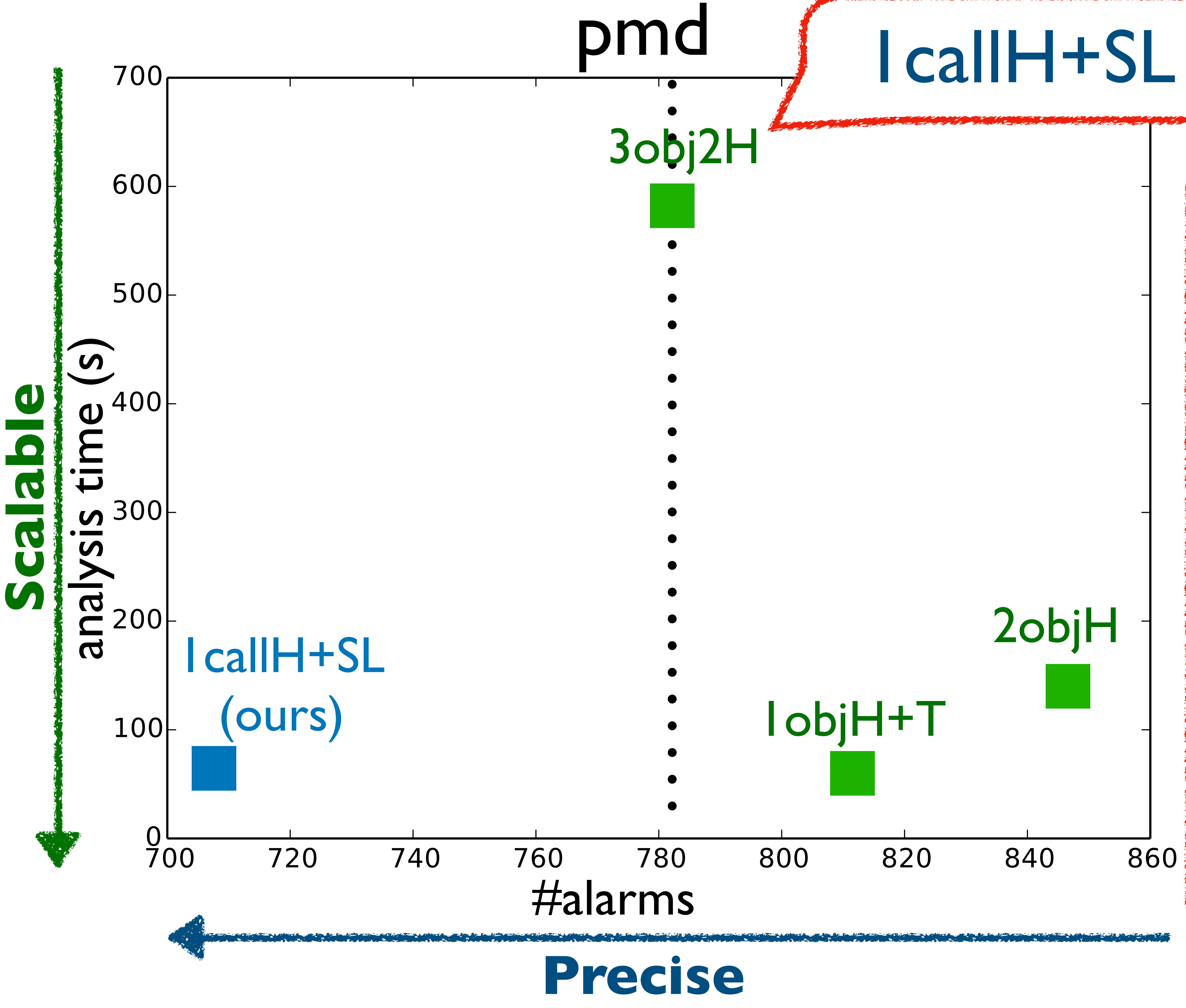




Transformed 1-CFA with tunneling via **Obj2CFA** from 1objH+T

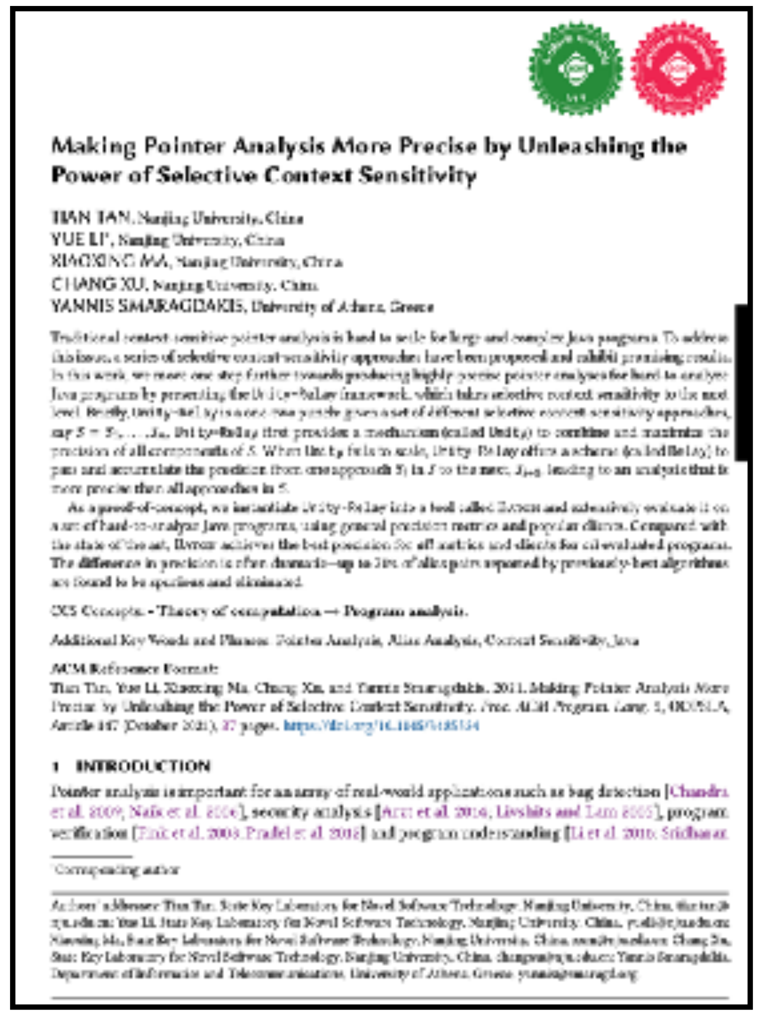
Call-site Sensitivity vs Object Sensitivity

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



1callH+SL is even more precise than 3obj2H

Precision upper bound of
recent researches on object sensitivity



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

TIAN TAN, Nanjing University, China
YUE LI, Nanjing University, China
XIAOXING MA, Nanjing University, China
CHANG XU, Nanjing University, China
YANNIS SMARAGDAKIS, University of Athens, Greece

OOPLSA 2021

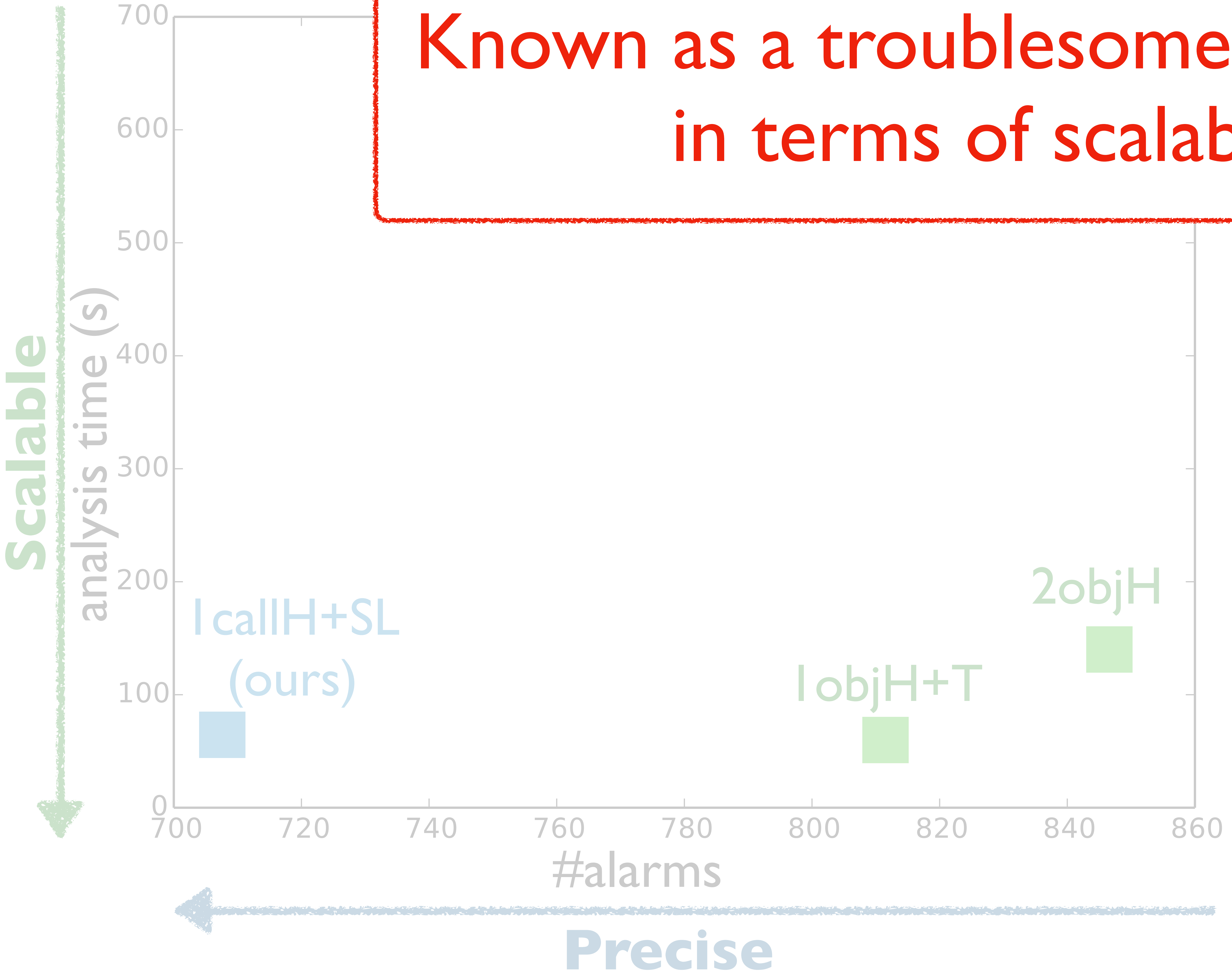
Precision-Preserving Yet Fast Object-Sensitive Pointer Analysis with Partial Context Sensitivity

JINGBO LU, UNSW Sydney, Australia
JINGLING XUE, UNSW Sydney, Australia

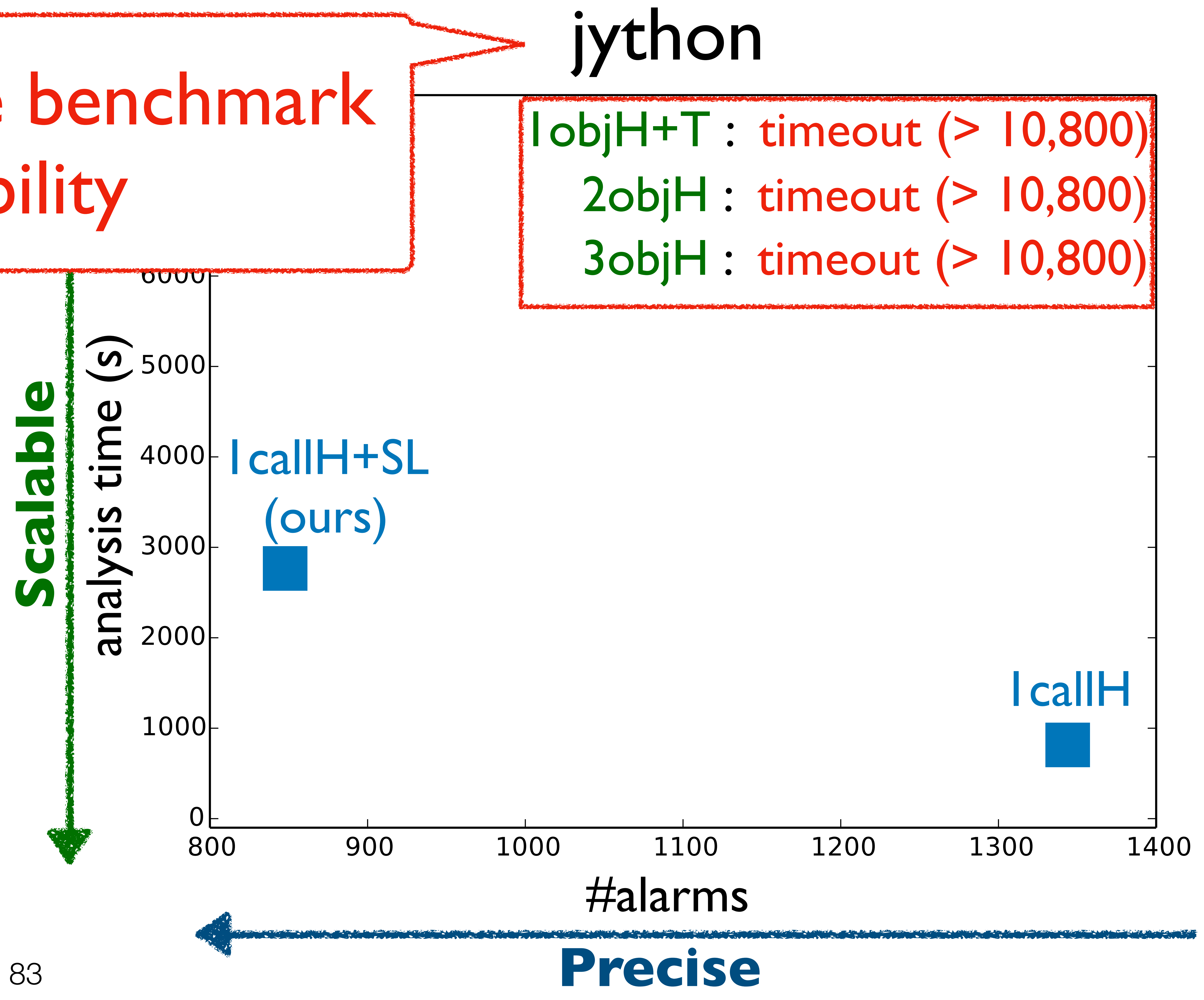
OOPLSA 2019

Call-site Sensitivity vs Object Sensitivity

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



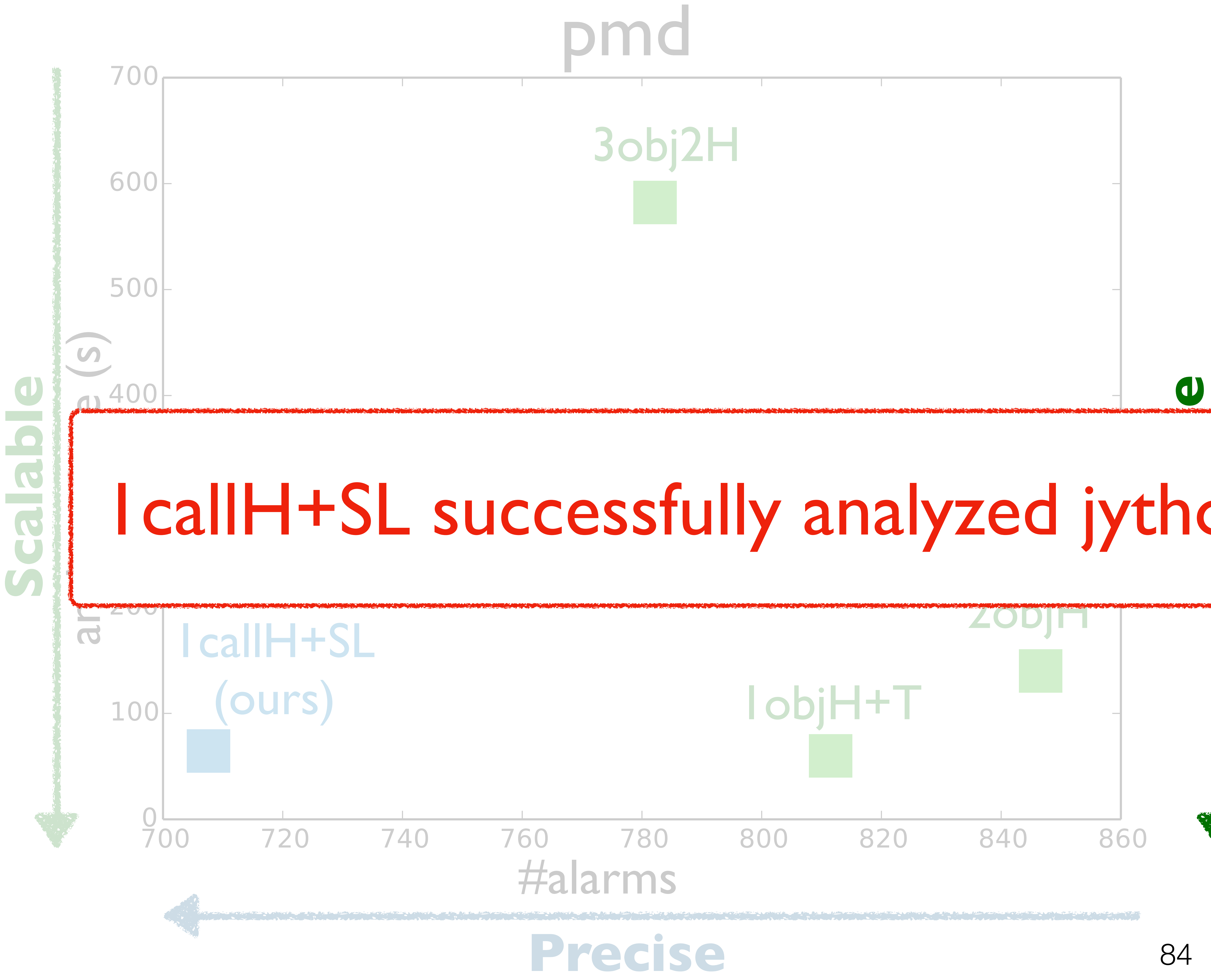
Known as a troublesome benchmark in terms of scalability



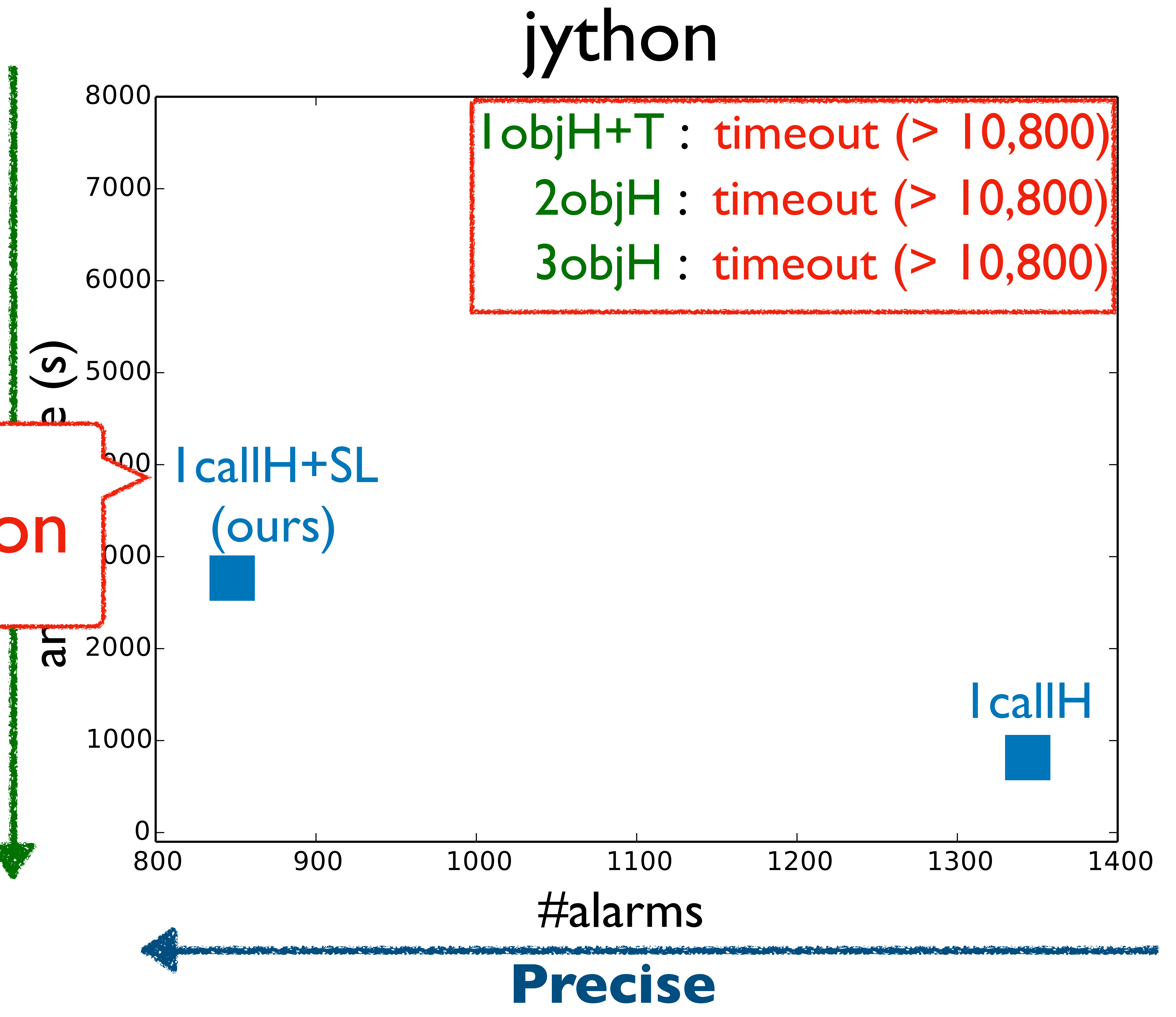
1objH+T : timeout (> 10,800)
2objH : timeout (> 10,800)
3objH : timeout (> 10,800)

Call-site Sensitivity vs Object Sensitivity

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



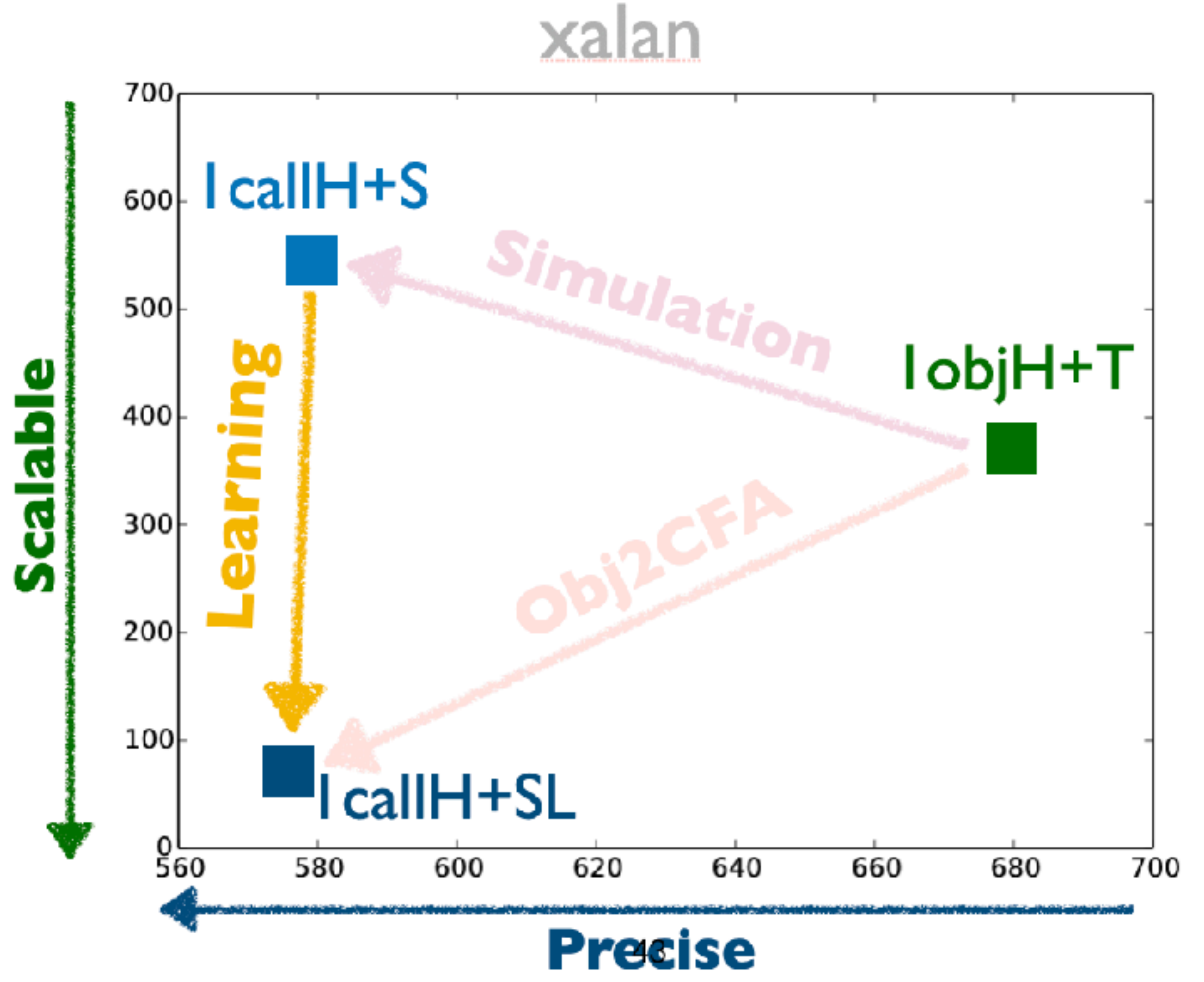
1callH+SL successfully analyzed jython

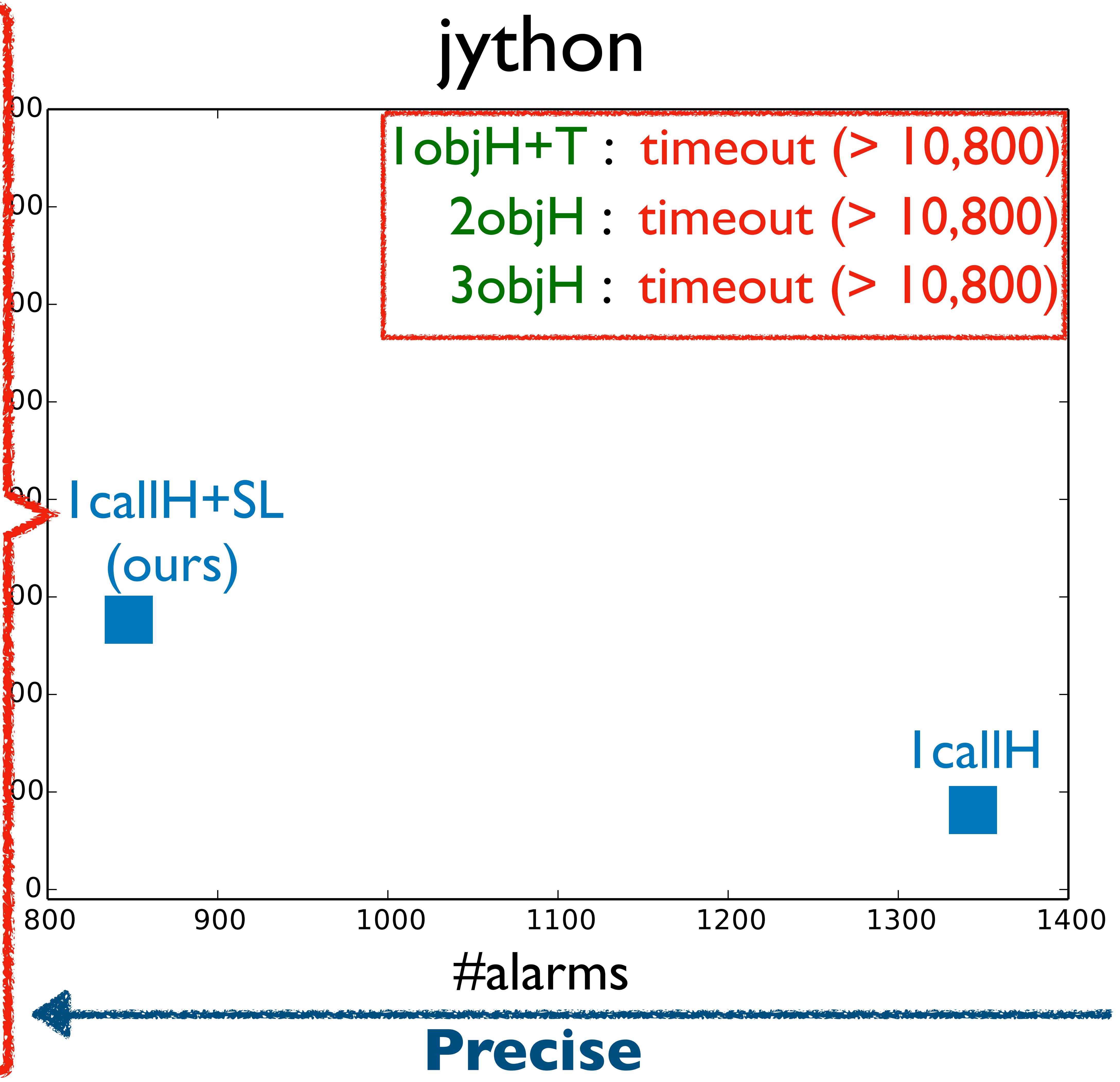


Call-site Sensitivity vs Object Sensitivity

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

- Necessity of learning
- 1callH+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

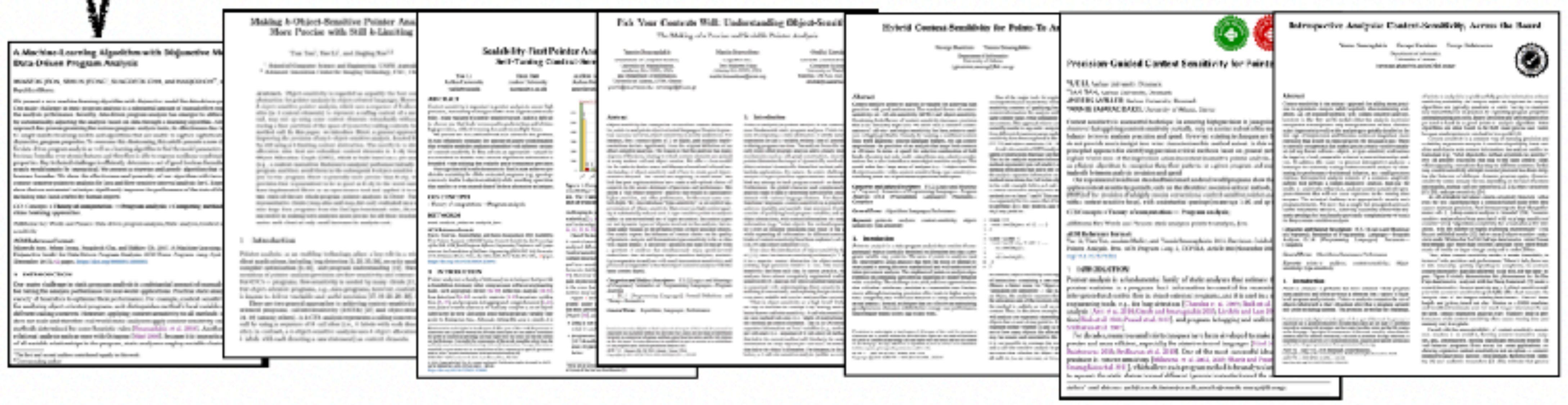
Thank you

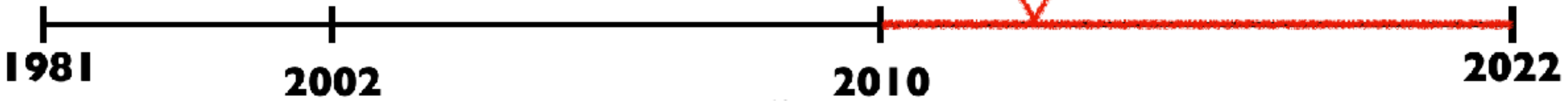
Summary

- 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]

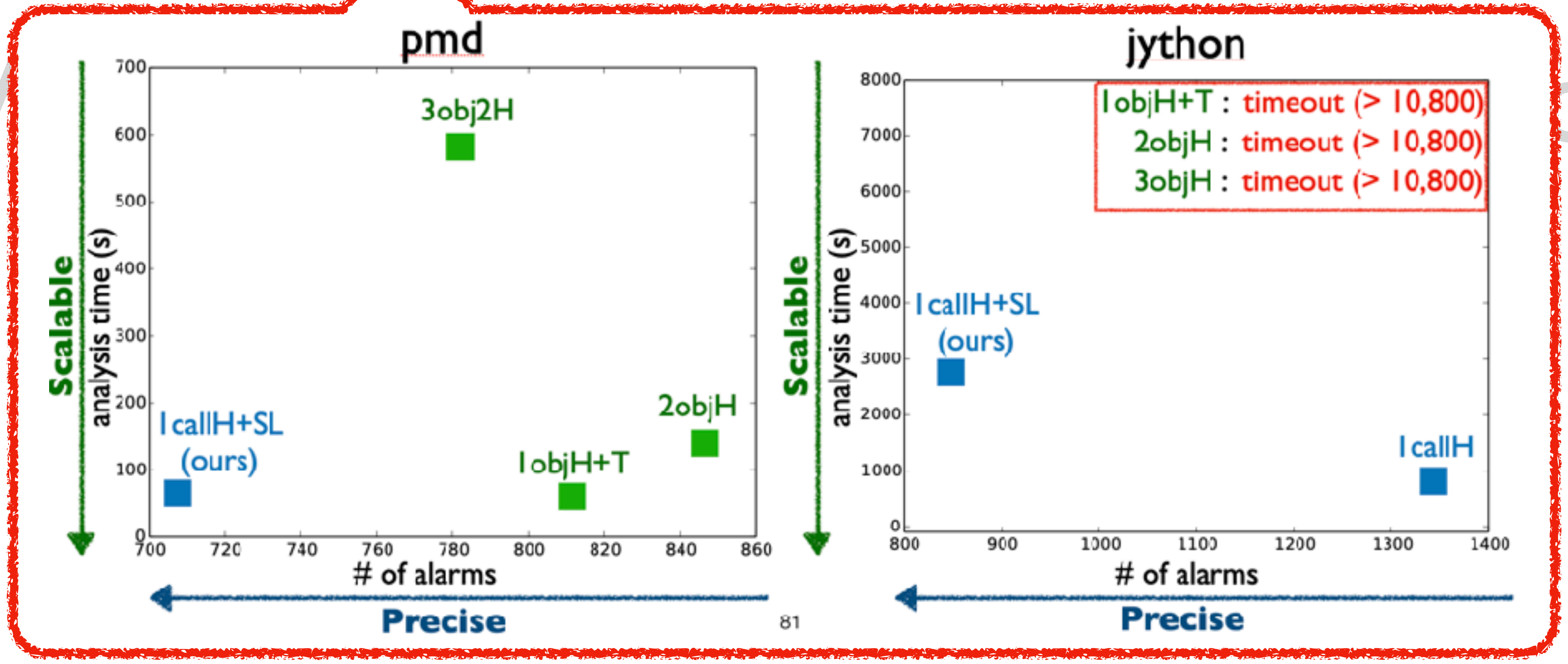




Summary

- Currently, CFA is known as a bad context

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



- W om now on

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

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

In this paper, we challenge the commonly-accepted wisdom in static analysis that object sensitivity is superior to call-site sensitivity for object-oriented programs. In static analysis of object-oriented programs, object sensitivity has been established as the dominant flavor of context sensitivity thanks to its outstanding precision. On the other hand, call-site sensitivity has been regarded as unsuitable and its use in practice has been constantly discouraged for object-oriented programs. In this paper, however, we claim that call-site sensitivity is generally a superior context abstraction because it is practically possible to transform object sensitivity into more precise call-site sensitivity. Our key insight is that the previously known superiority of object sensitivity holds only in the traditional *k*-limited setting, where the analysis is enforced to keep the most recent *k* context elements. However, it no longer holds in a recently-proposed, more general setting with context tunneling. With context tunneling, where the analysis is free to choose an arbitrary *k*-length subsequence of context strings, we show that call-site sensitivity can simulate object sensitivity almost completely, but not vice versa. To support the claim, we present a technique, called Obj2Cfa, for transforming arbitrary context-tunneled object sensitivity into more precise, context-tunneled call-site-sensitivity. We implemented Obj2Cfa in Doop and used it to derive a new call-site-sensitive analysis from a state-of-the-art object-sensitive pointer analysis. Experimental results confirm that the resulting call-site sensitivity outperforms object sensitivity in precision and scalability for real-world Java programs. Remarkably, our results show that even 1-call-site sensitivity can be more precise than the conventional 3-object-sensitive analysis.

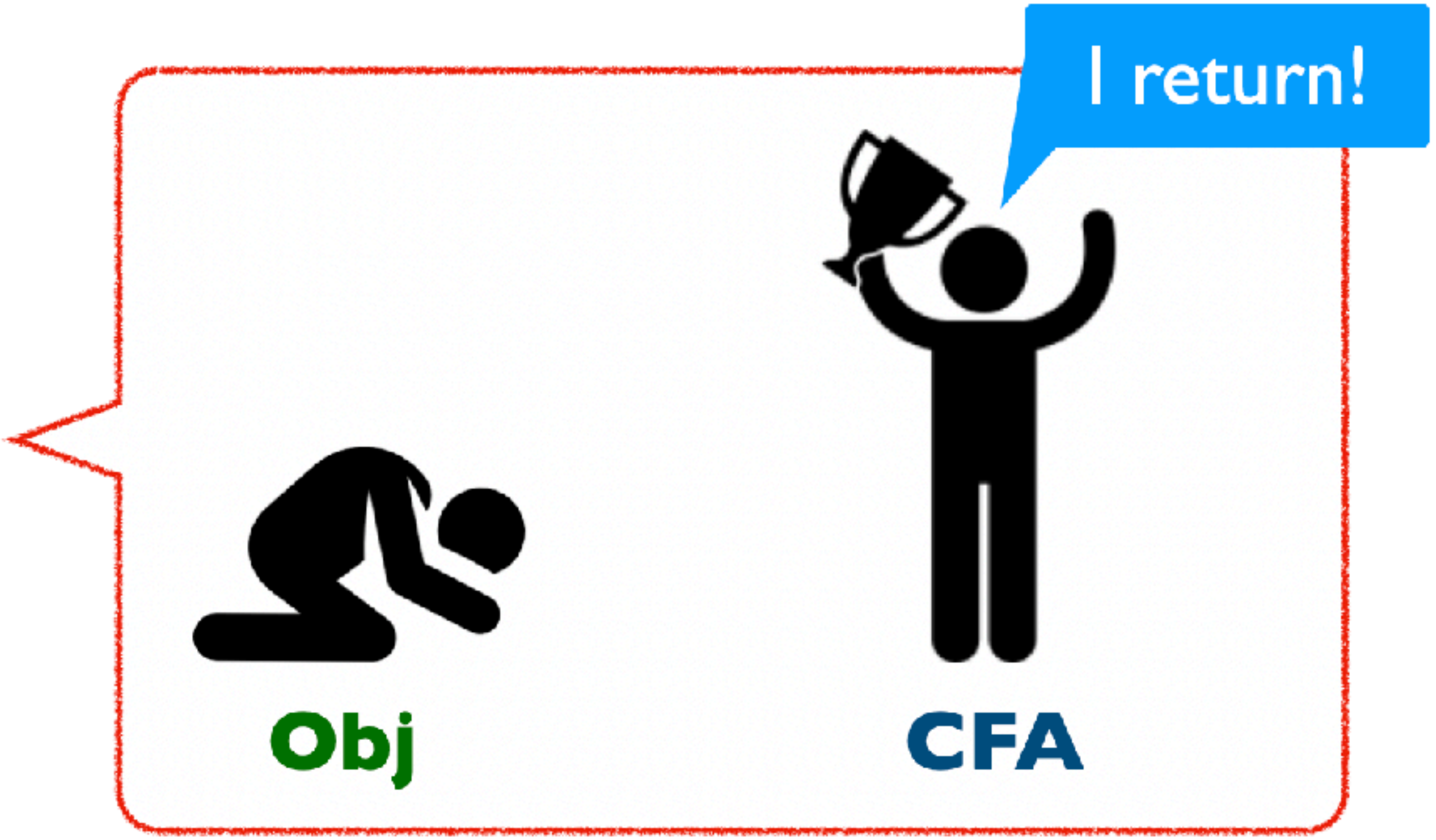
1 INTRODUCTION

> *"Since its introduction, object sensitivity has emerged as the dominant flavor of context sensitivity for object-oriented languages."*
>
> —Smaragdakis and Balatsouras [2015]

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

CFA wins!

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



- We need to reconsider CFA from now on

Thank you