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

Two major camps

A: Call-Site Sensitivity Can

Object Sensitivity Even for

Object-Oriented Programs

Minseok Jeon and Hakjoo Oh

KOREA UNIVERSITY

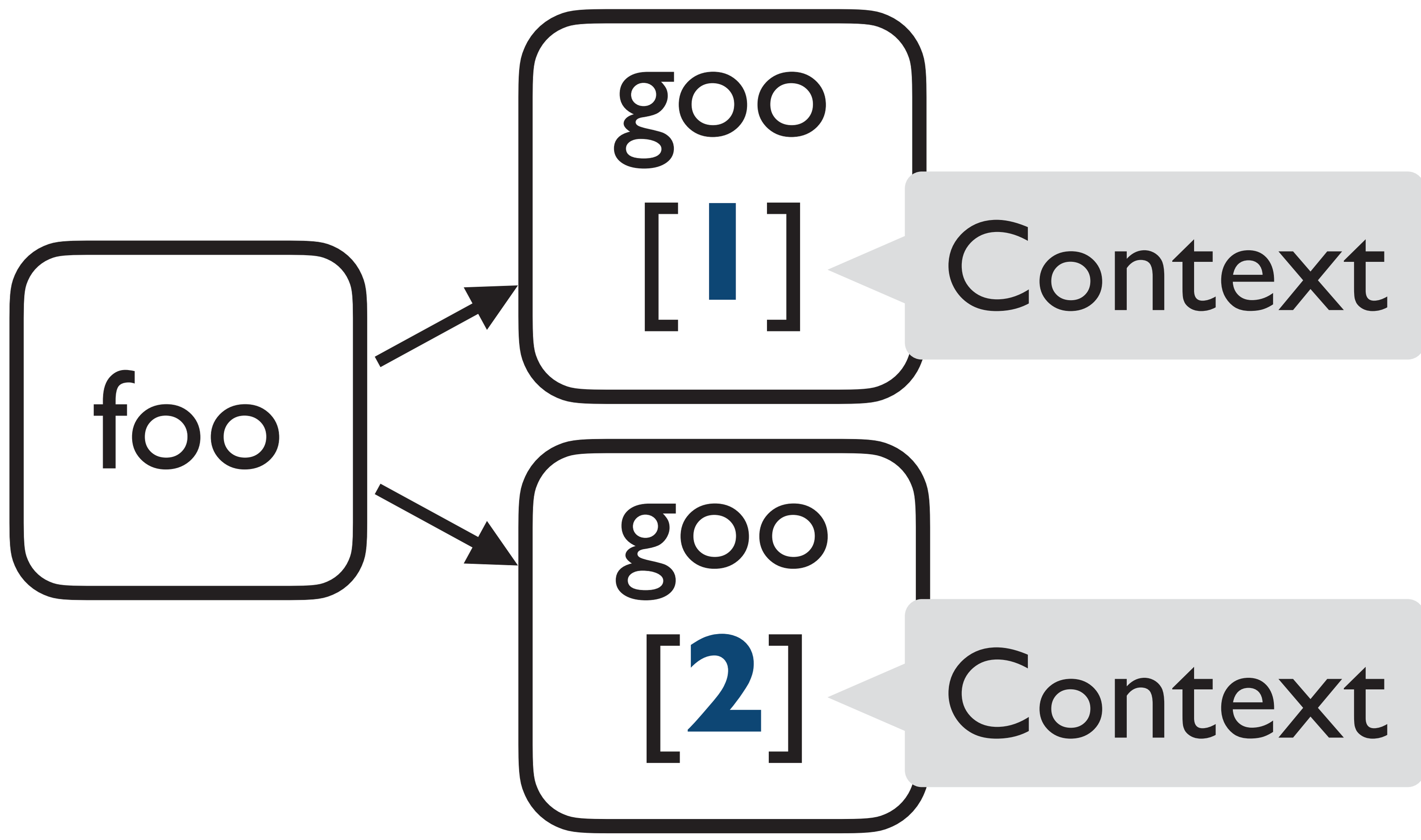
SW재난연구센터 workshop @ Jeju, Korea

Call-site Sensitivity vs Object Sensitivity

Call-site sensitivity was born in 1981

- Considers "**Where**"

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



Call graph

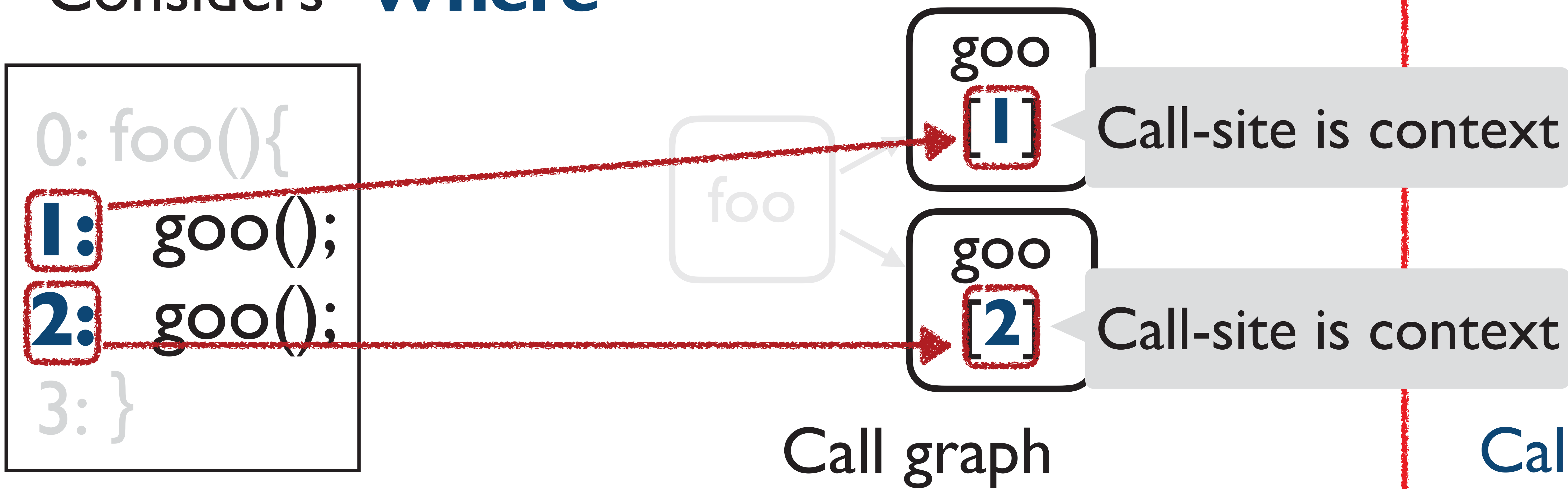
Call-site sensitivity

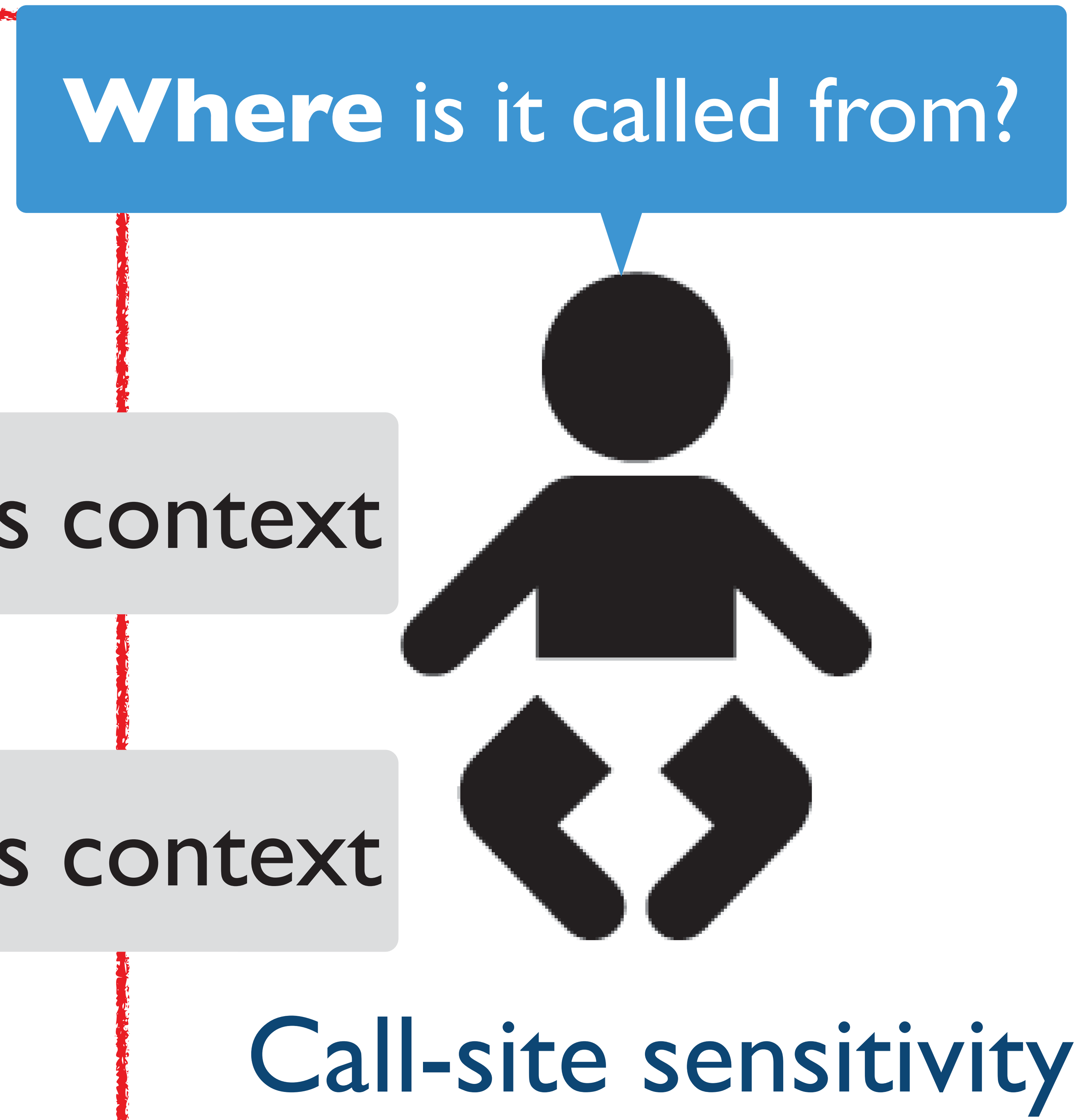
1981 2002 2010 2022

Call-site Sensitivity vs Object Sensitivity

Call-site sensitivity was born in 1981

- Considers "**Where**"





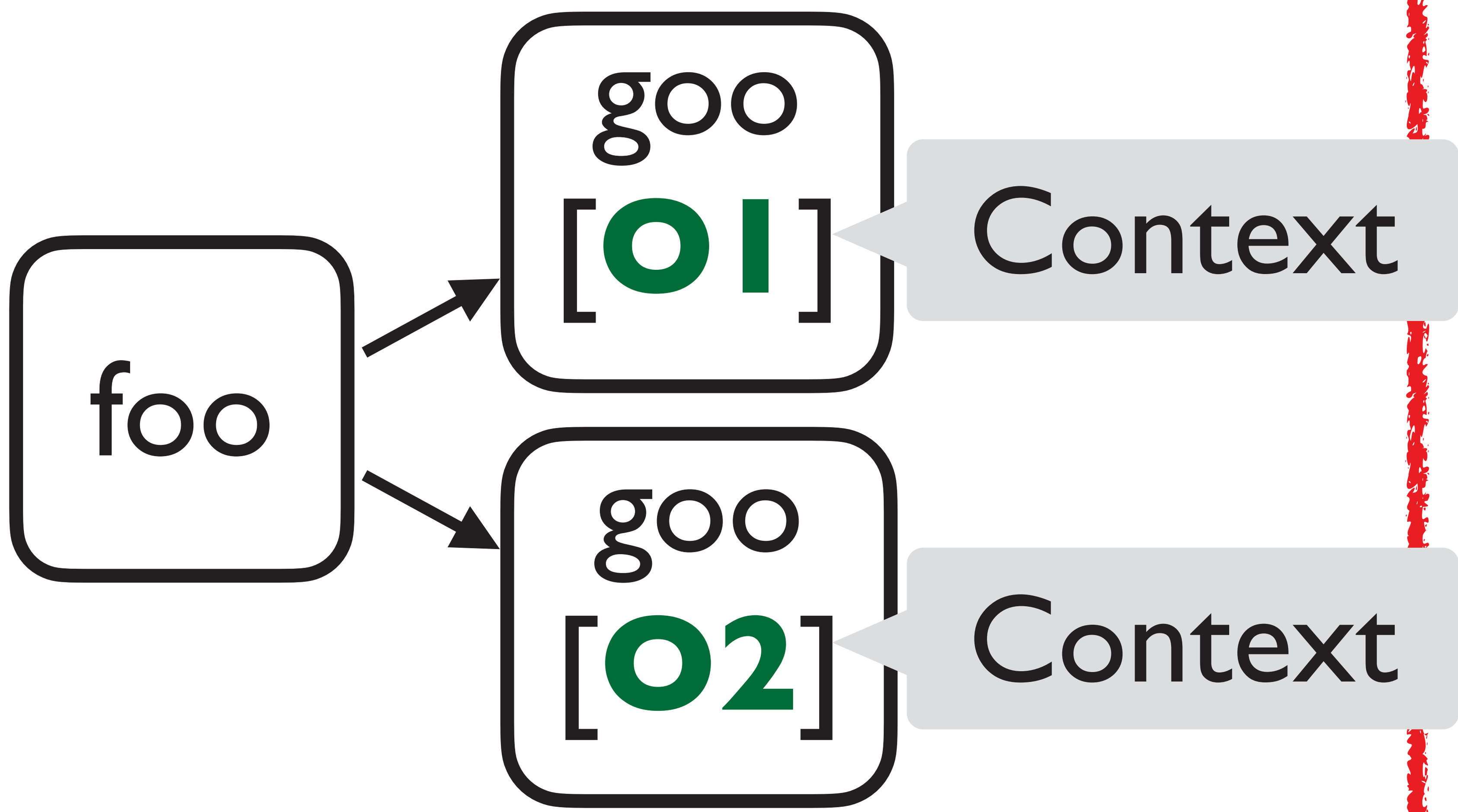
Call-site Sensitivity vs Object Sensitivity

Object sensitivity appeared in 2002

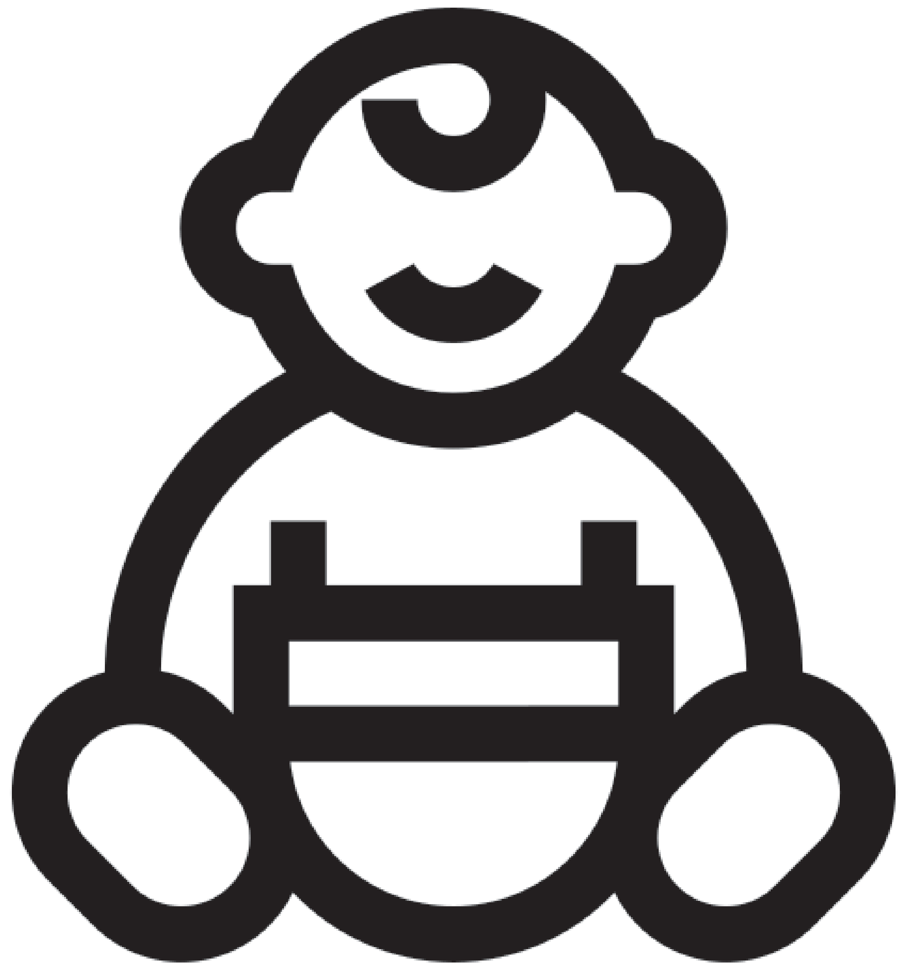
- Considers "**What**"

O1 or **O2**

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



Call graph

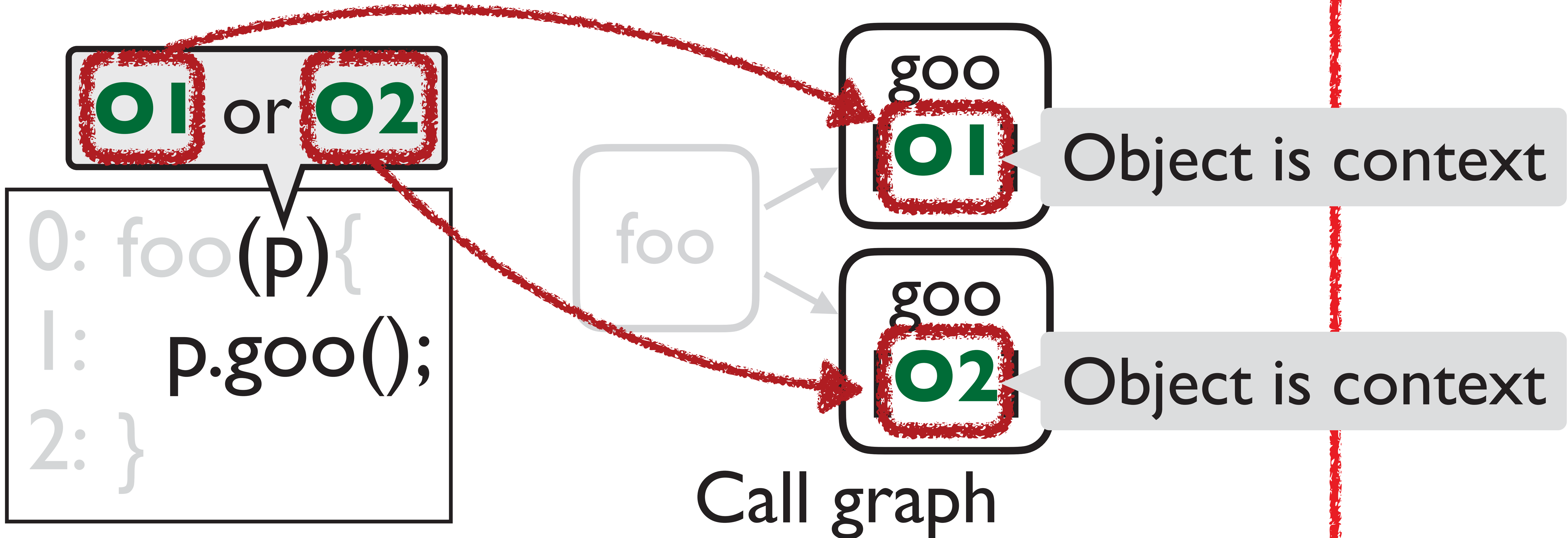


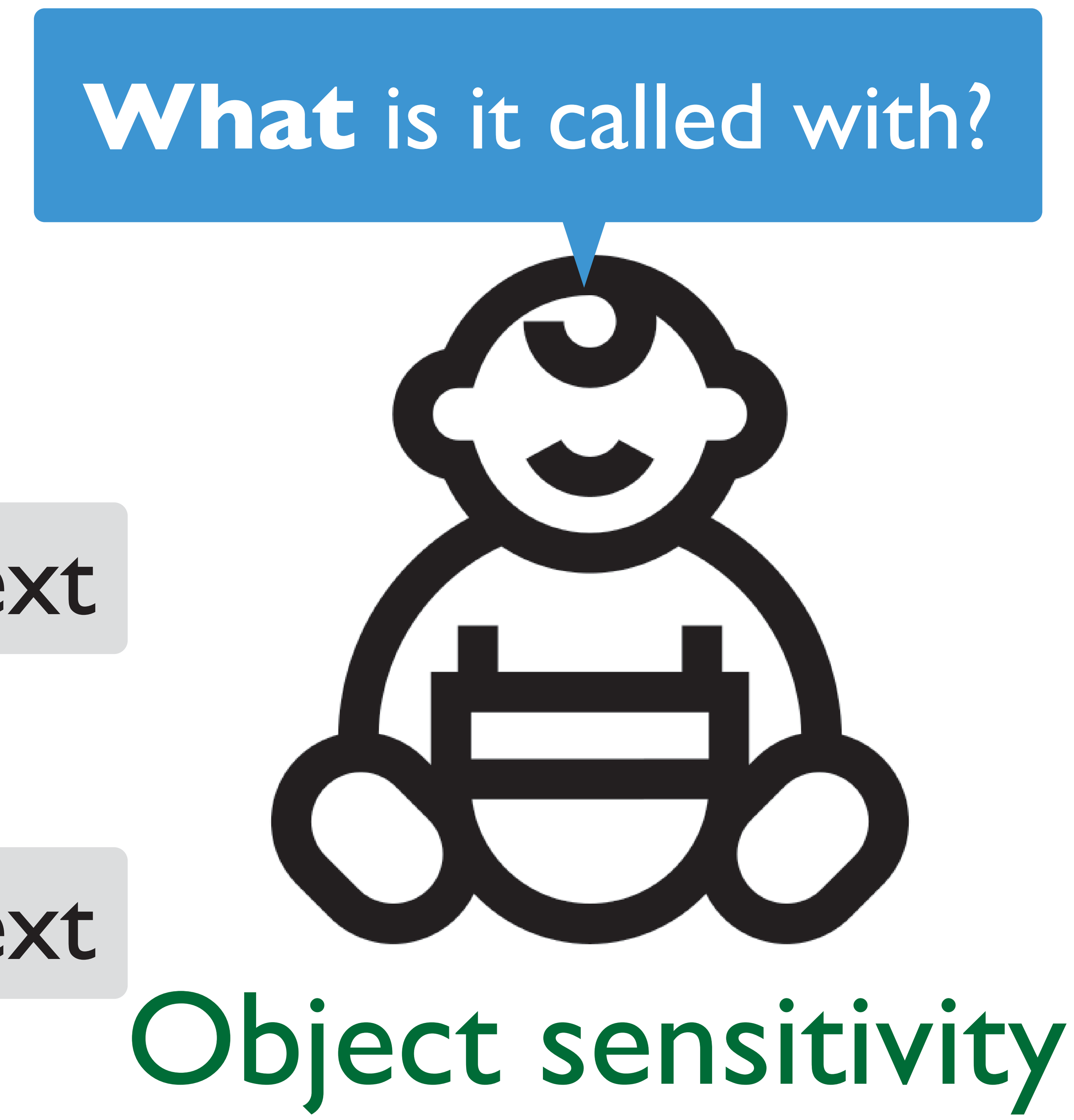
Object sensitivity

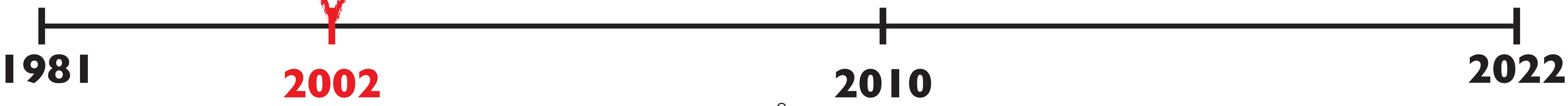
Call-site Sensitivity vs Object Sensitivity

Object sensitivity appeared in 2002

- Considers "**What**"



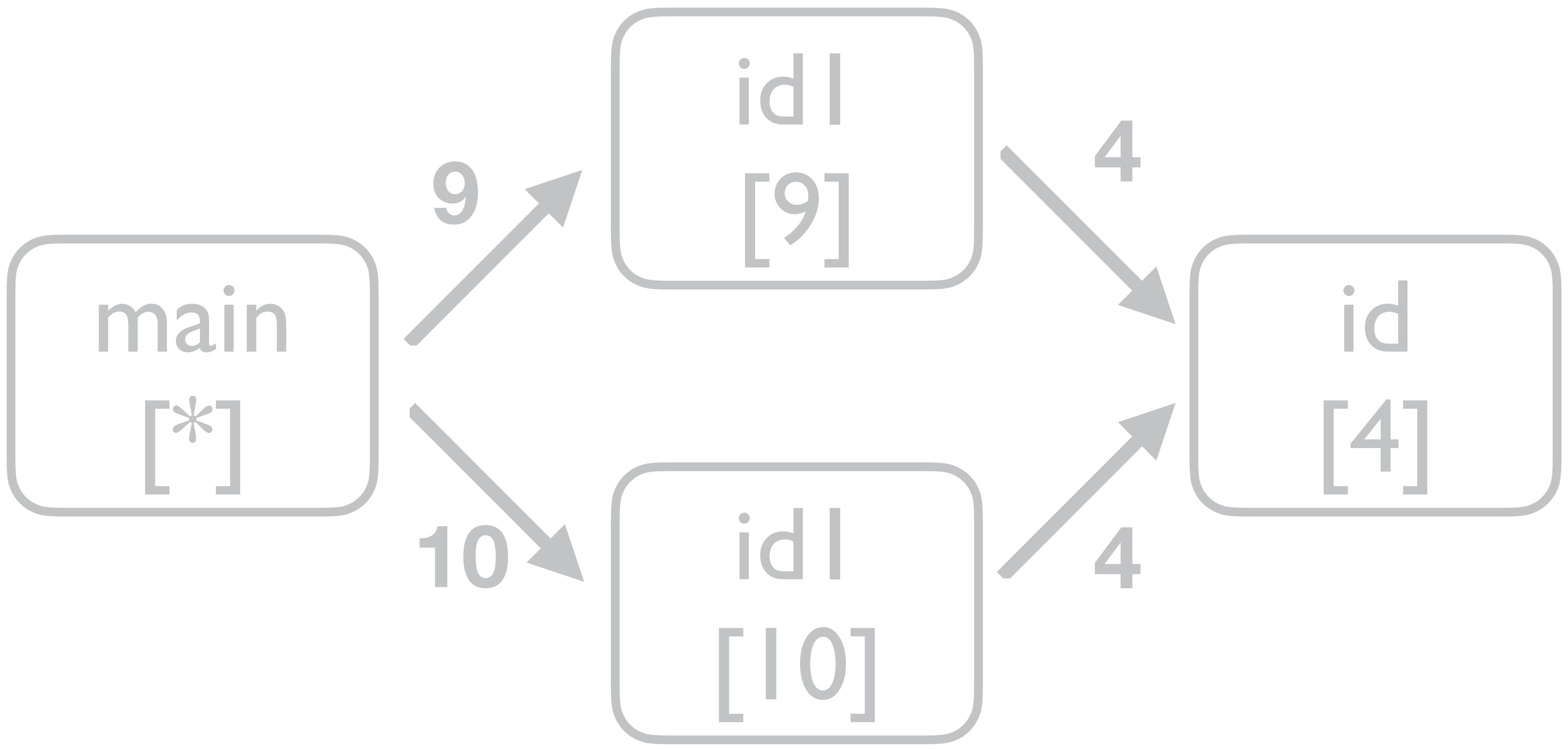




Call-site Sensitivity vs Object Sensitivity

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

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

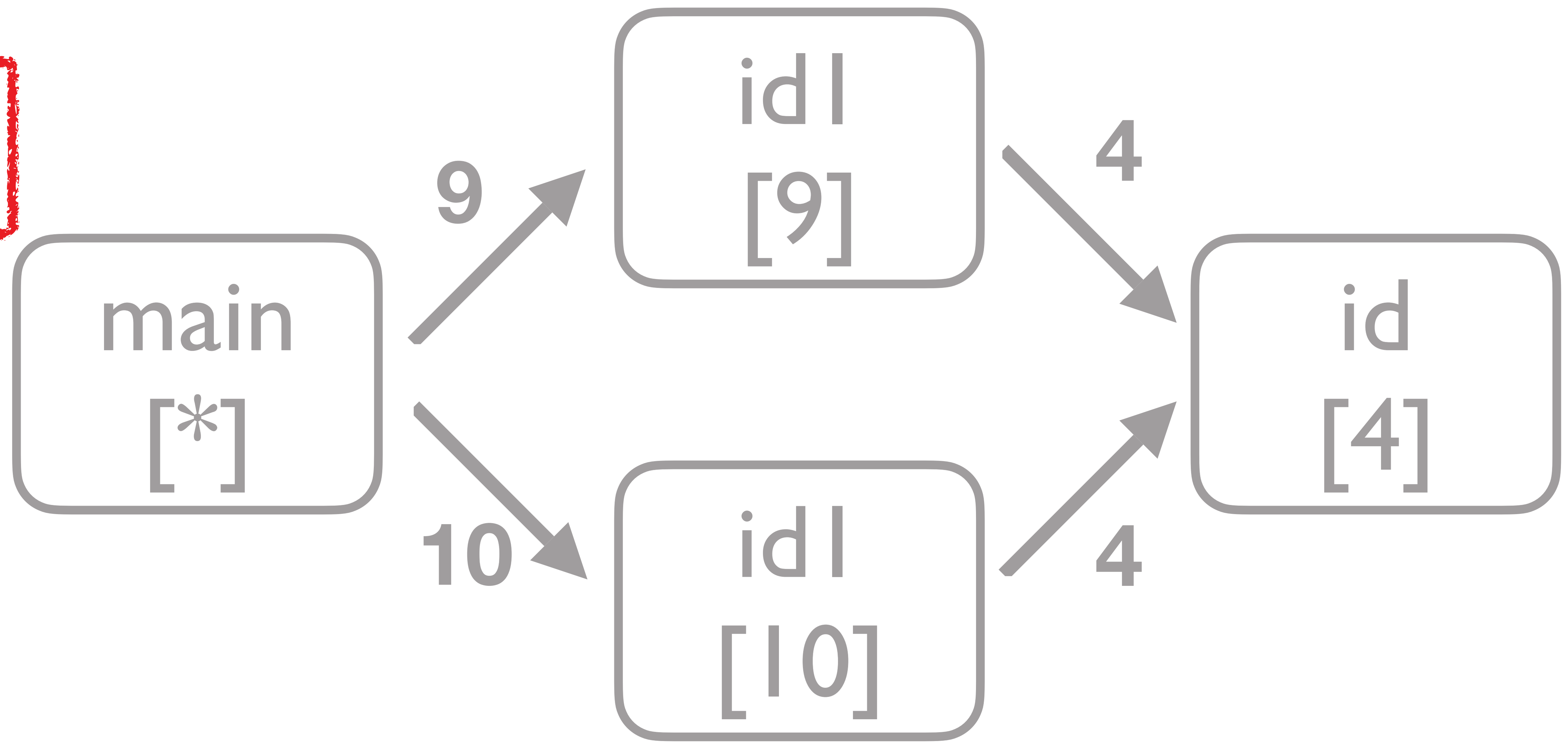


Call-graph of 1-CFA

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;}      <- Identity function
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: }
```



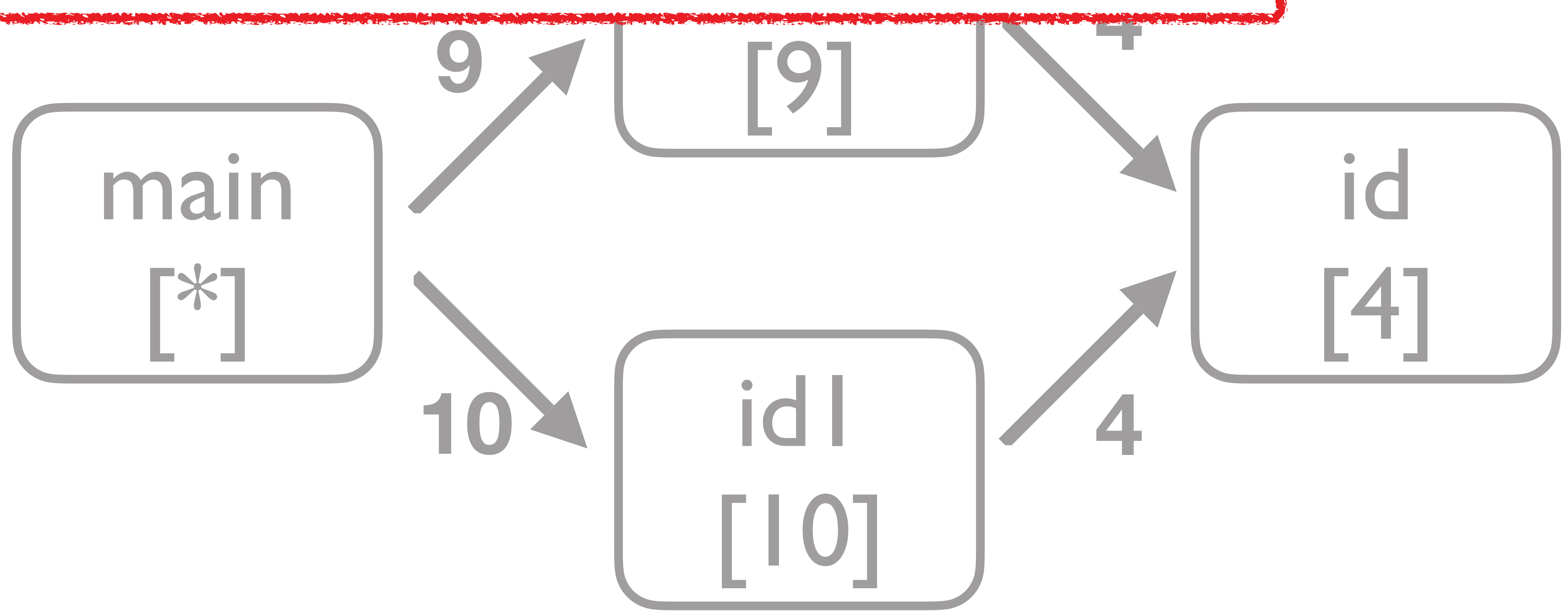
Call-graph of 1-CFA

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: }
```

Also an identity function implemented with id

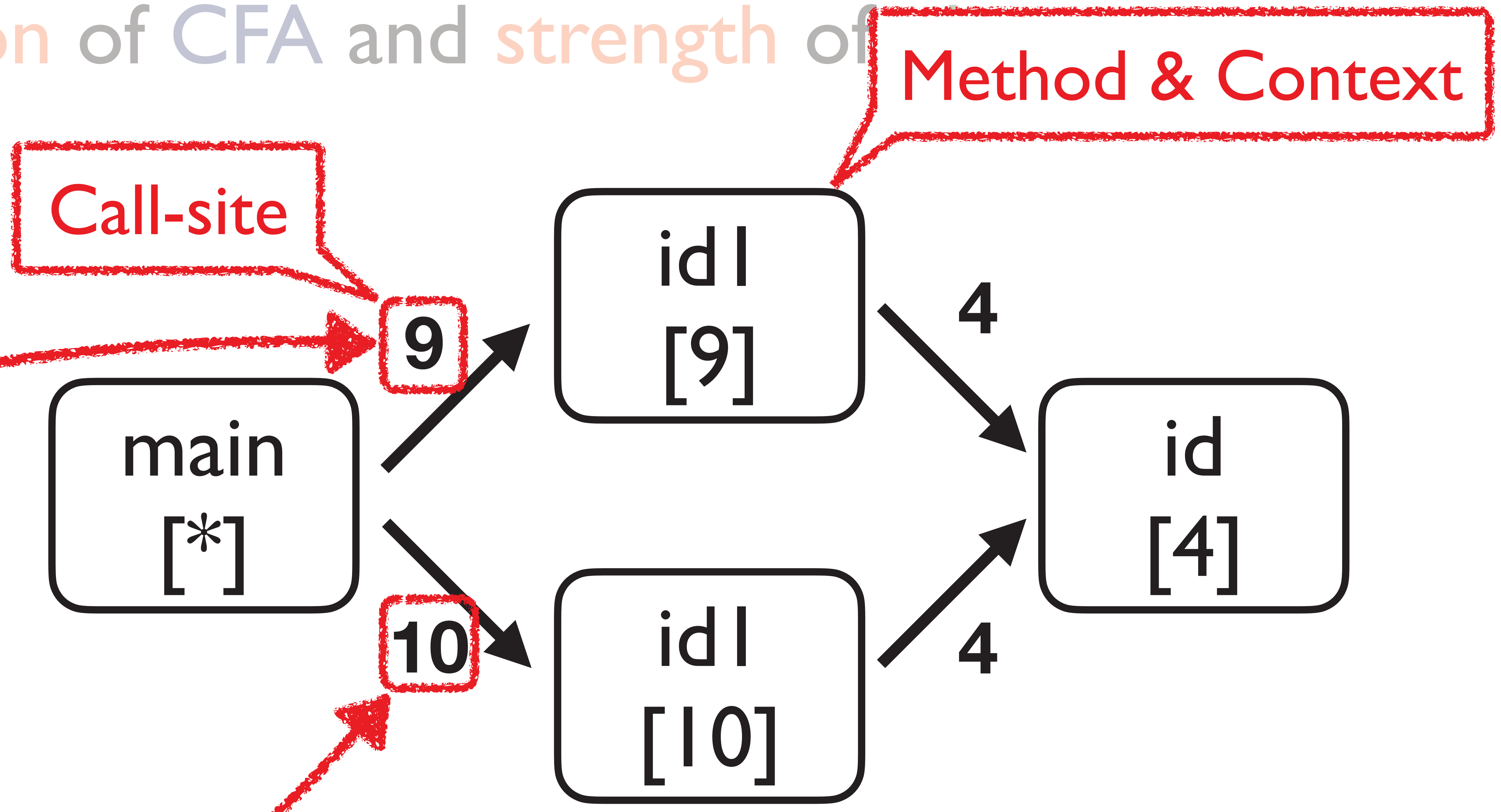


Call-graph of 1-CFA

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: }
```



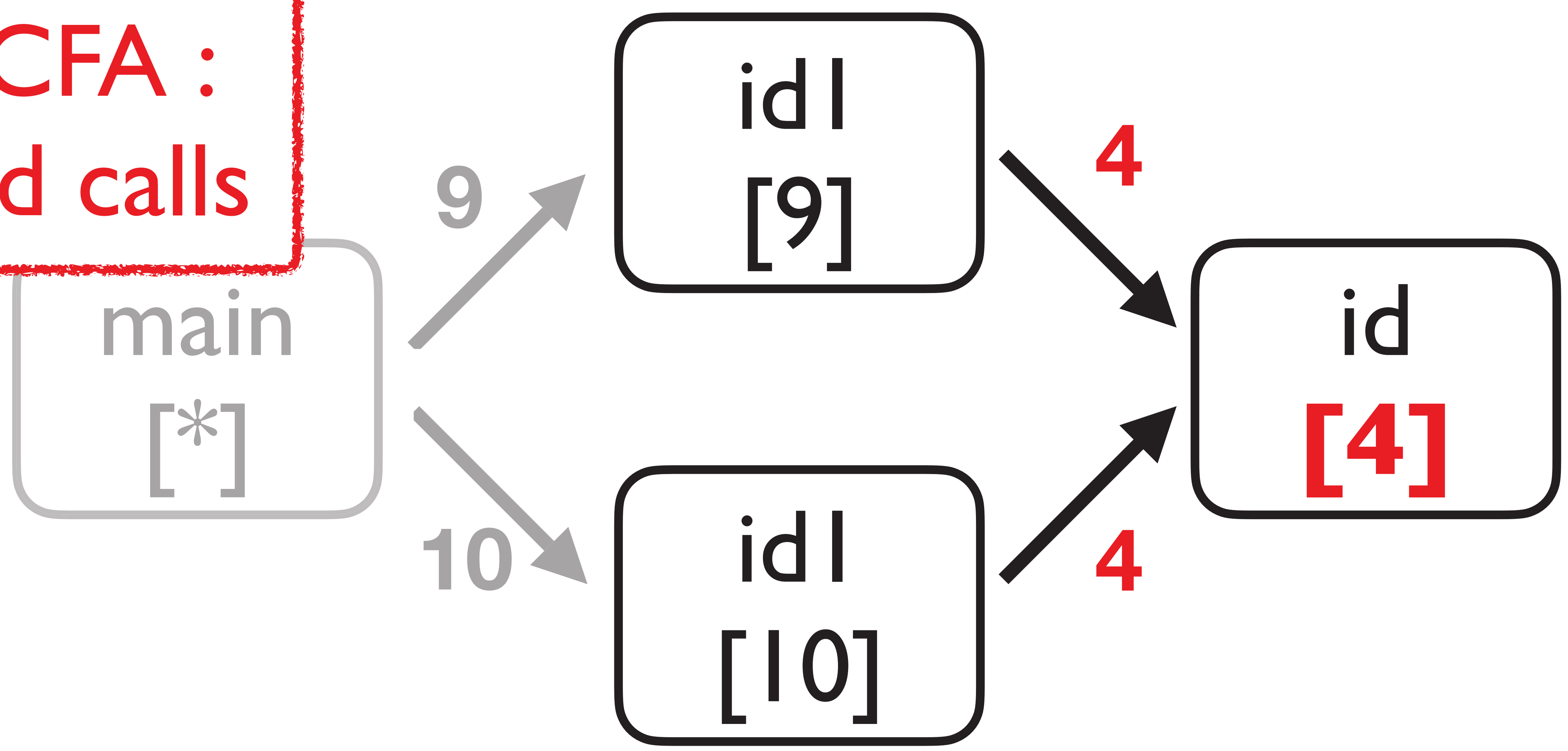
Call-graph of 1-CFA

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

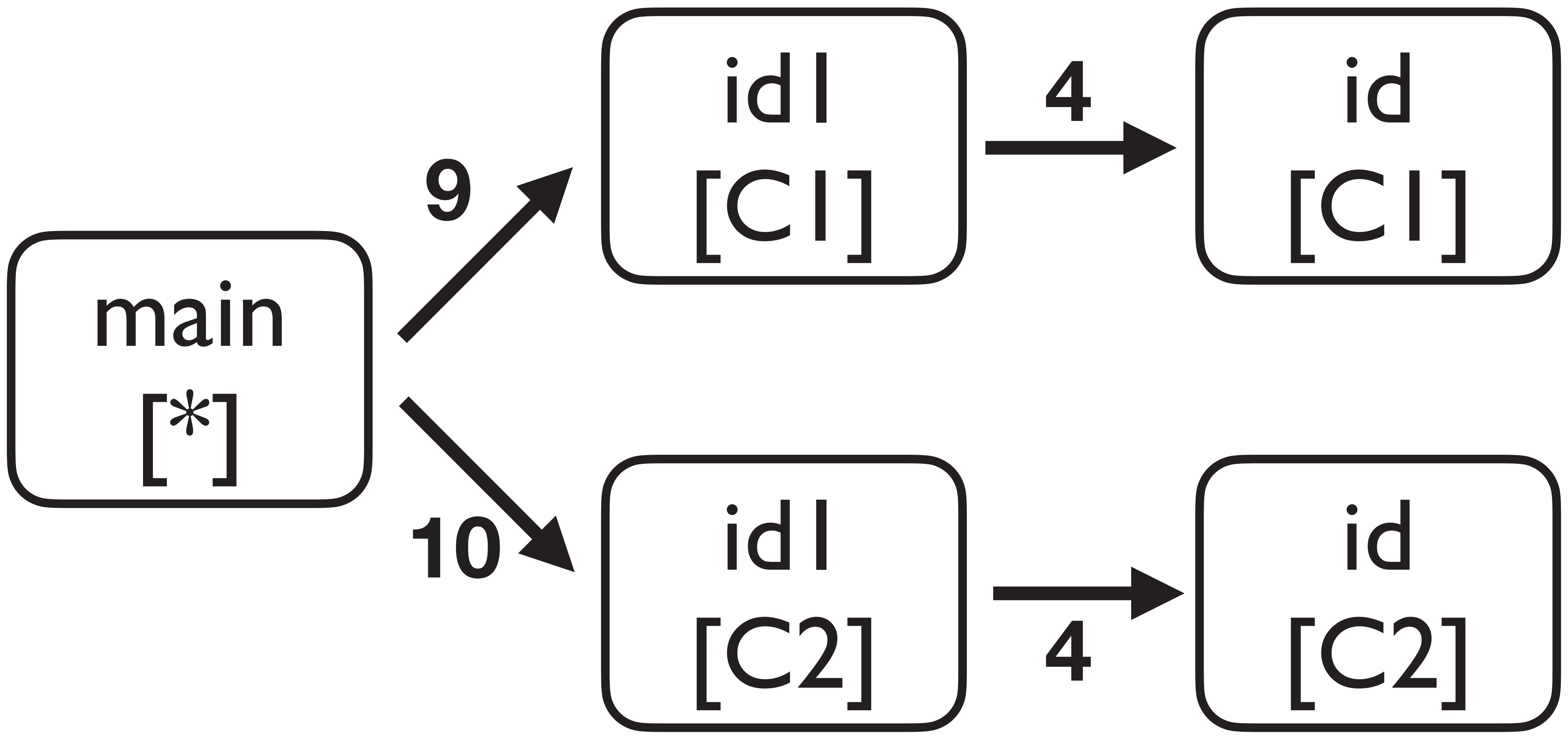


Call-graph of 1-CFA

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: }
```



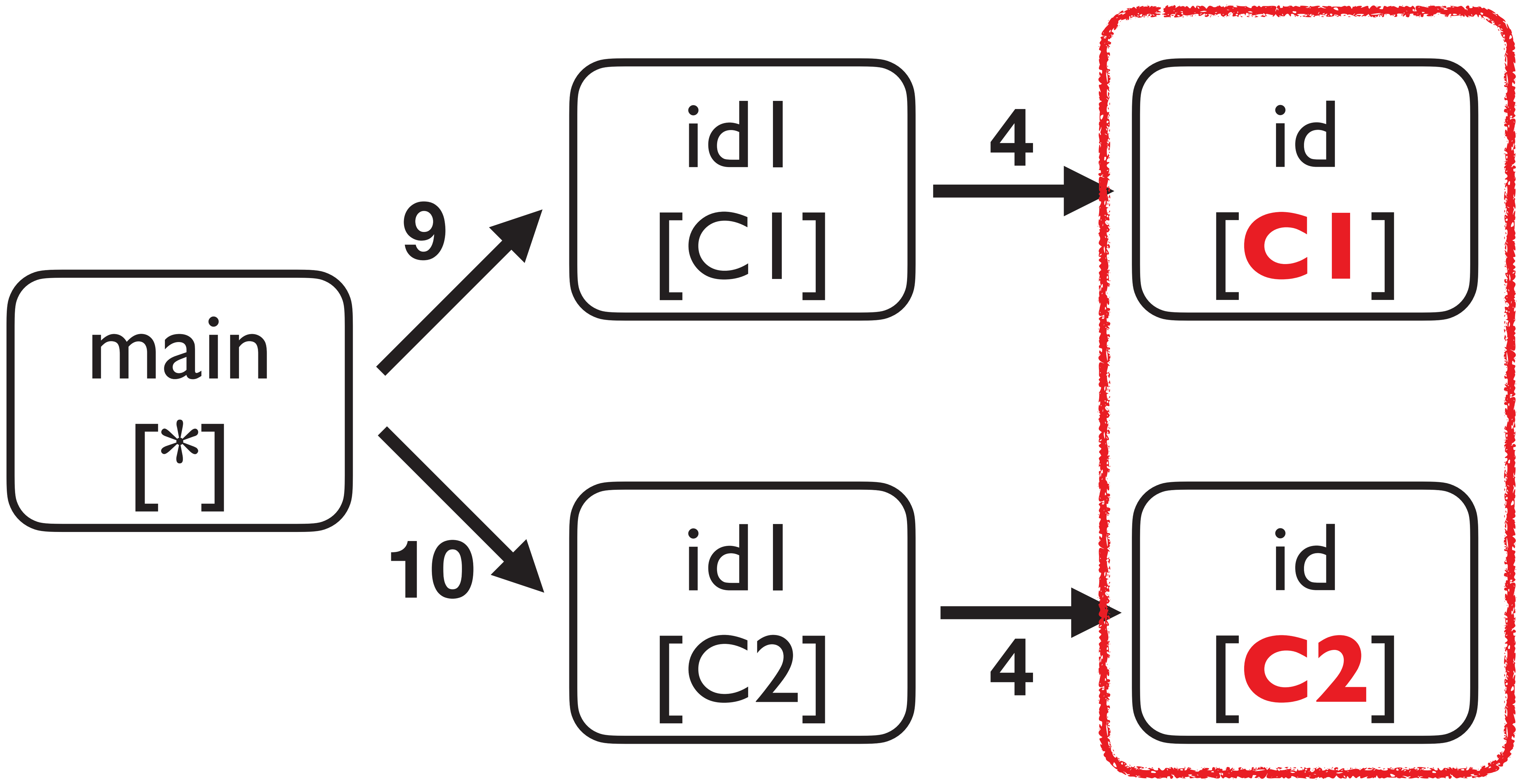
Call-graph of 1-Obj

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

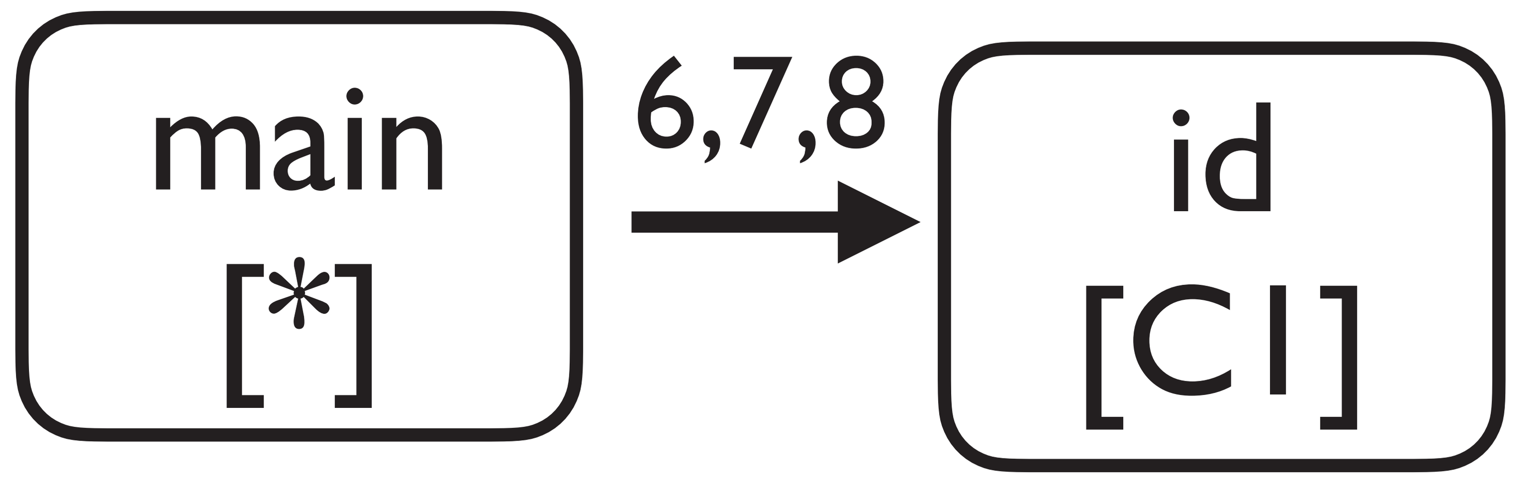


Call-graph of 1-Obj

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: }
```



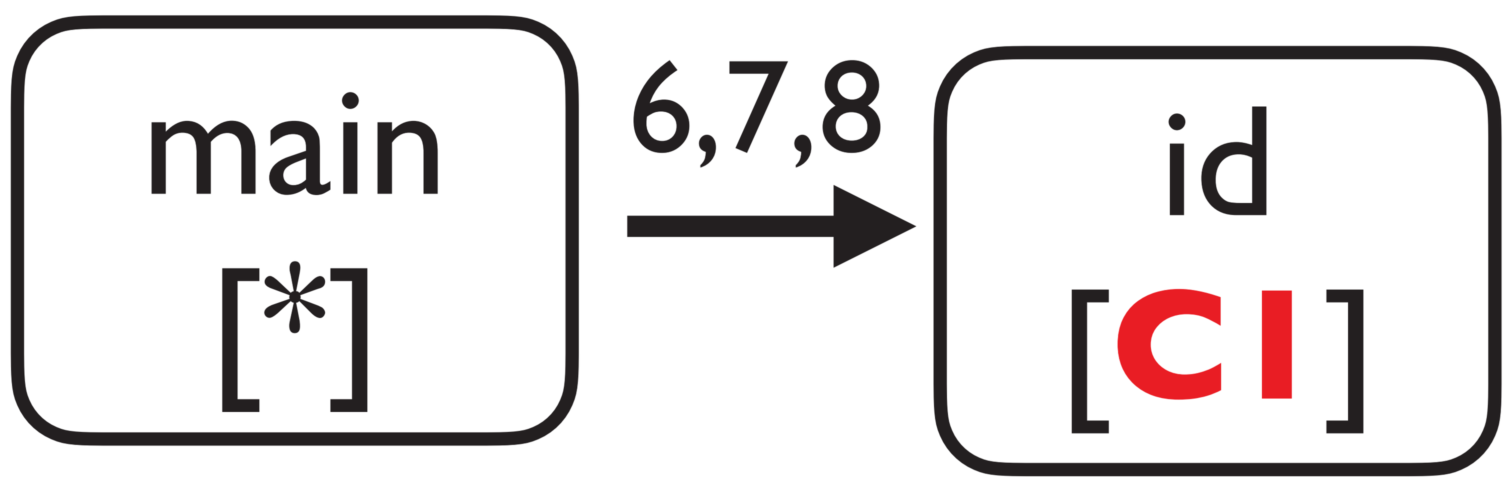
Call-graph of 1-Obj

Call-site Sensitivity vs Object Sensitivity

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

```
class C{
  id(v){
    return v;}
}
main(){
  c1 = new C();//C1
  a = (A) c1.id(new A());//query1
  b = (B) c1.id(new B());//query2
  c = (B) c1.id(new C());//query3
}
```

The three method calls share the same receiver object C1

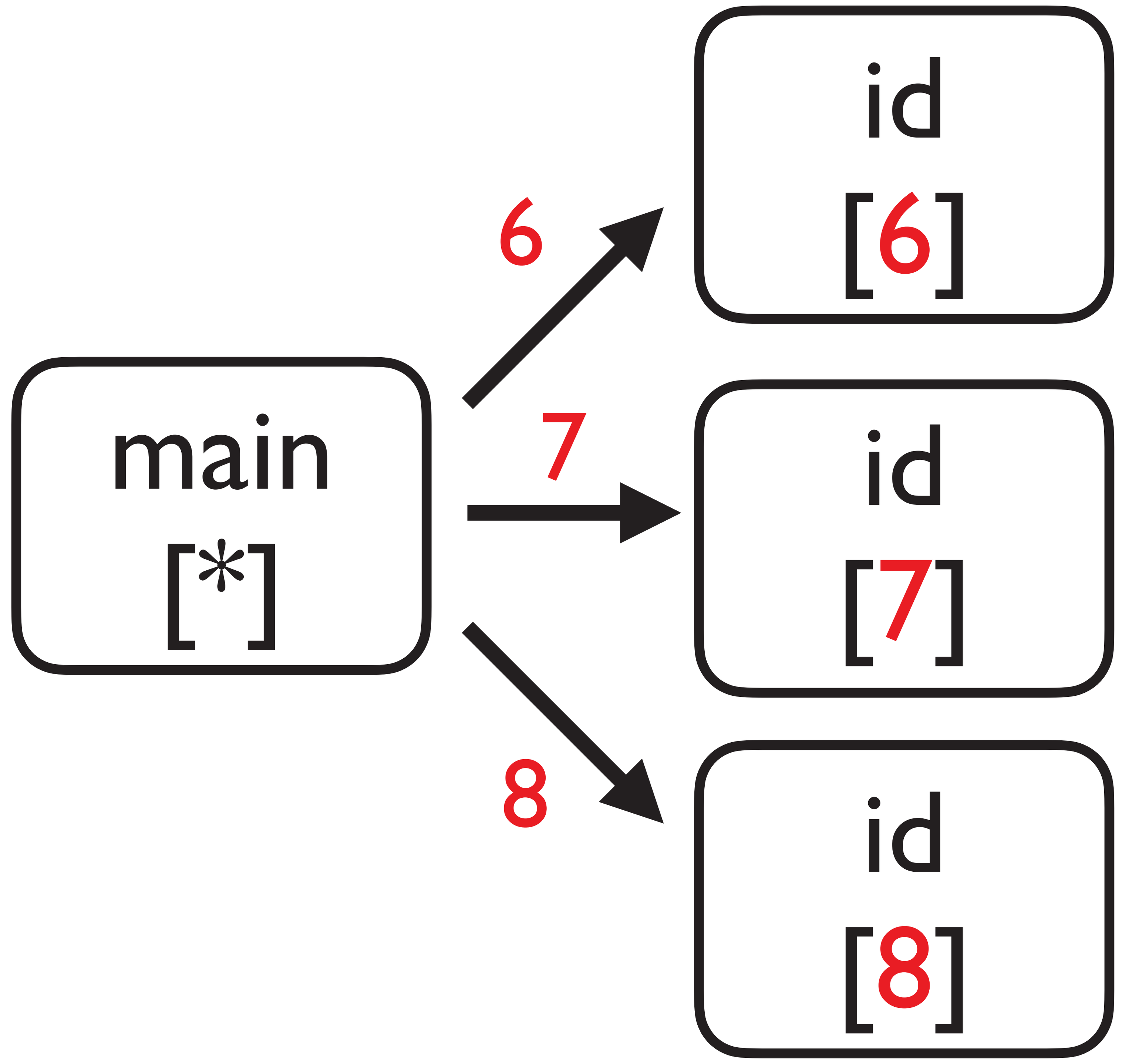


Call-graph of 1-Obj

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 = (C) c1.id(new C());//query3
9: }
```



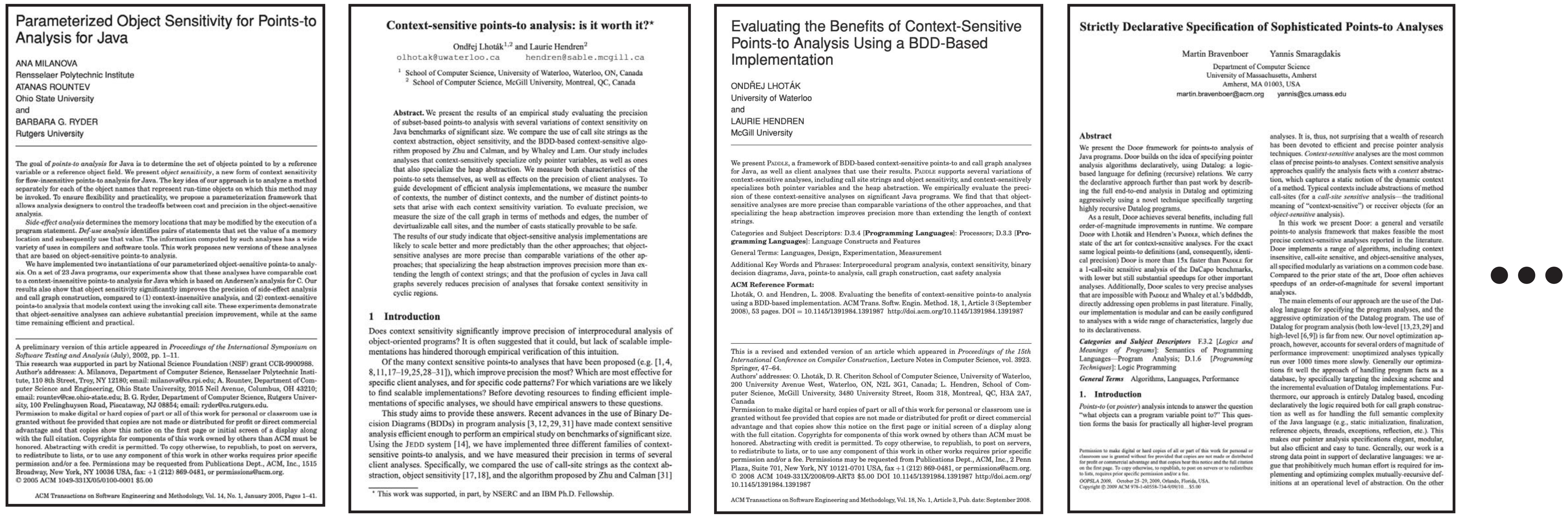
Call-graph of 1-CFA

Call-site sensitivity easily separates the three method calls

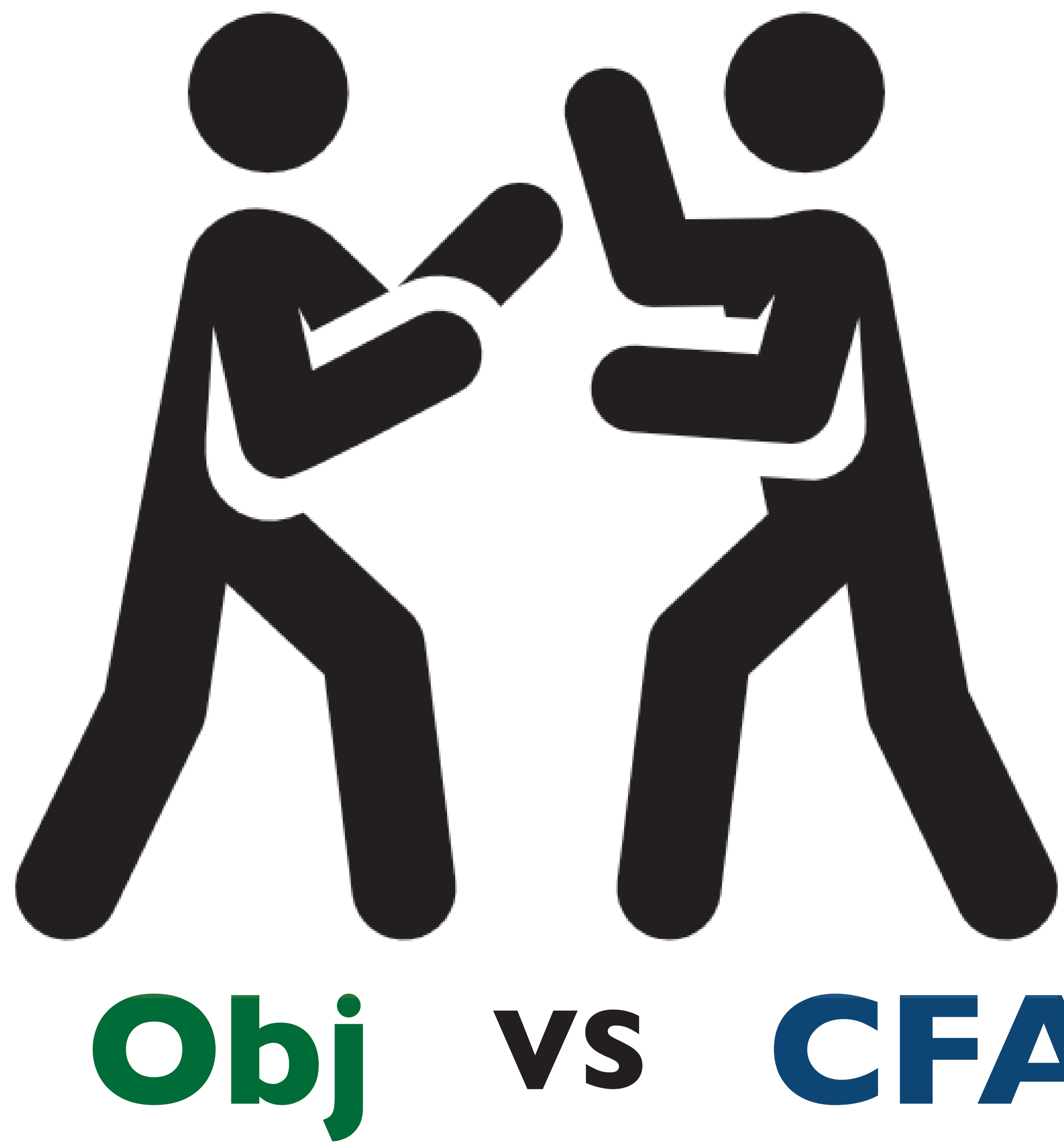
Call-site Sensitivity vs Object Sensitivity

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

Call-site Sensitivity vs Object Sensitivity



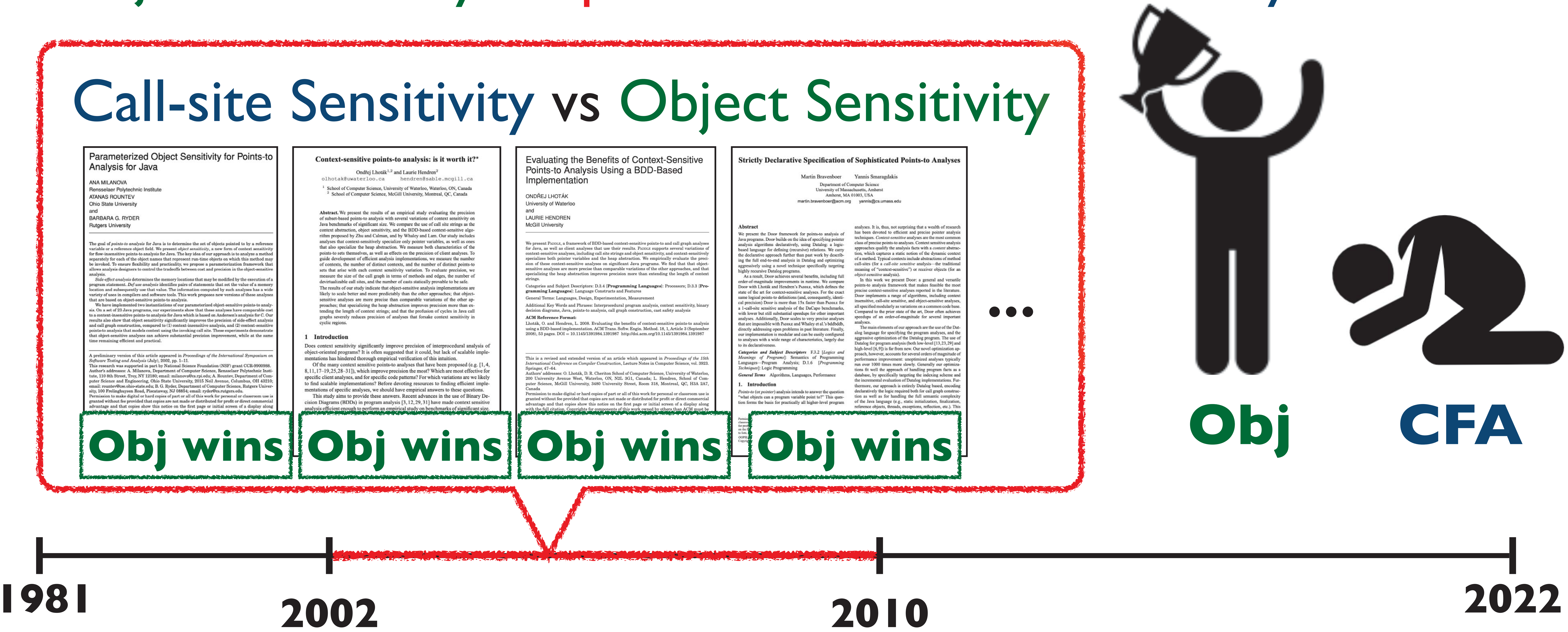
...



1981 — 2002 — 2010 — 2022

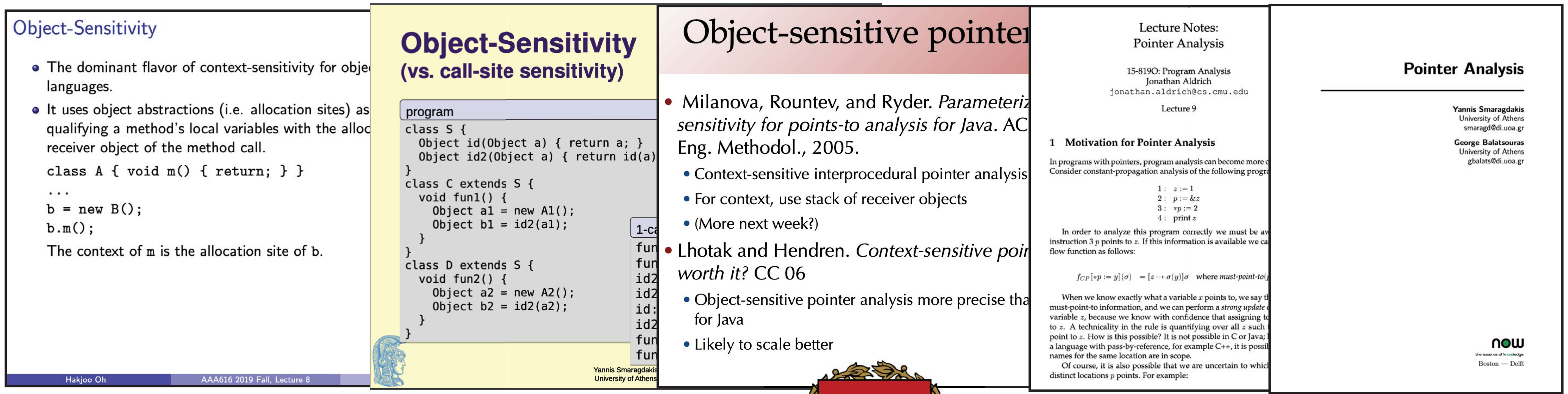
Call-site Sensitivity vs Object Sensitivity

- Object Sensitivity outperformed call-site sensitivity



Call-site Sensitivity vs Object Sensitivity

- Lectures have taught the superiority of object sensitivity



KOREA UNIVERSITY · National and Kapodistrian University of Athens · 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

Call-site Sensitivity vs Object Sensitivity

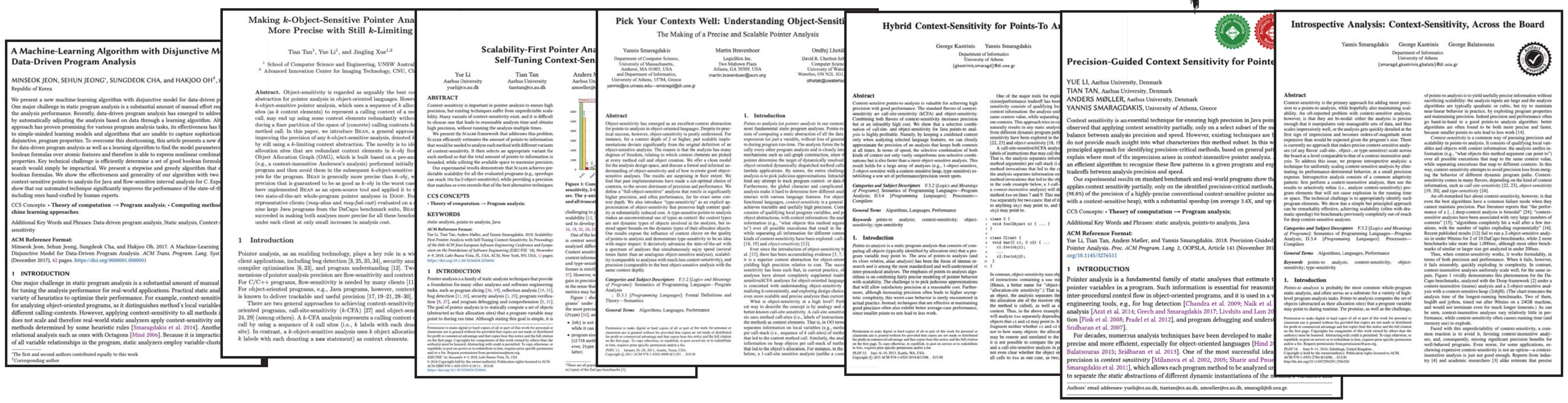
- Researches focused on improving Object Sensitivity

Call-site Sensitivity vs Object Sensitivity

- Call-site Sensitivity has been ignored

"We do not consider call-site sensitive analyses …"

\- Li et al. [2018]



CFA

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]

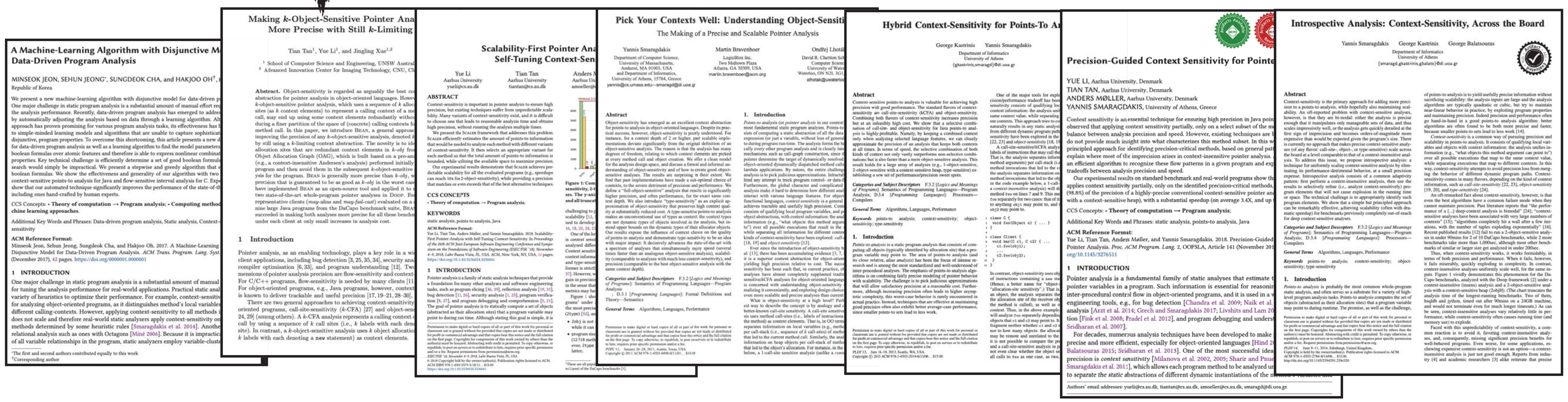
CFA

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]



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]

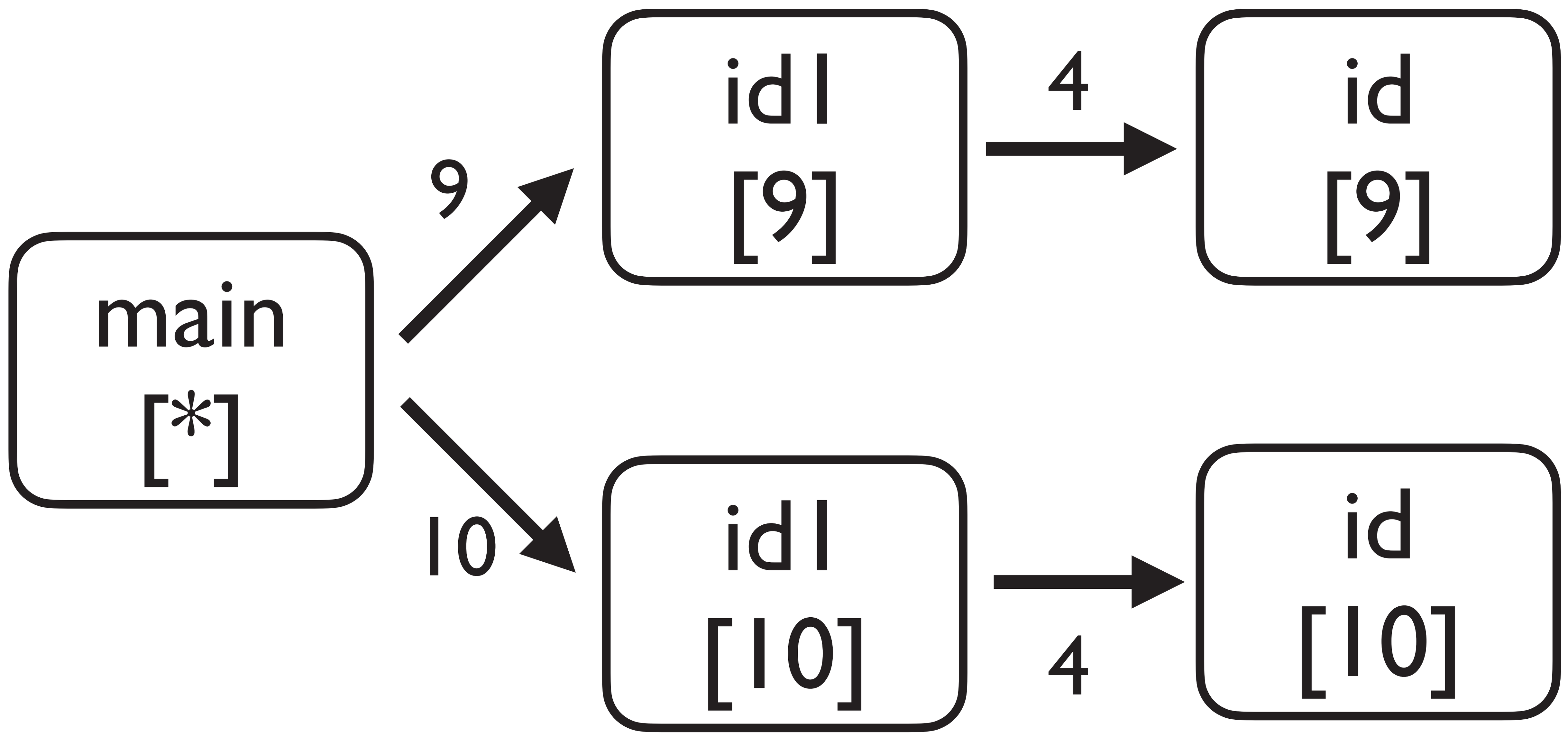
Context tunneling can improve both call-site sensitivity and object sensitivity

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: }
```

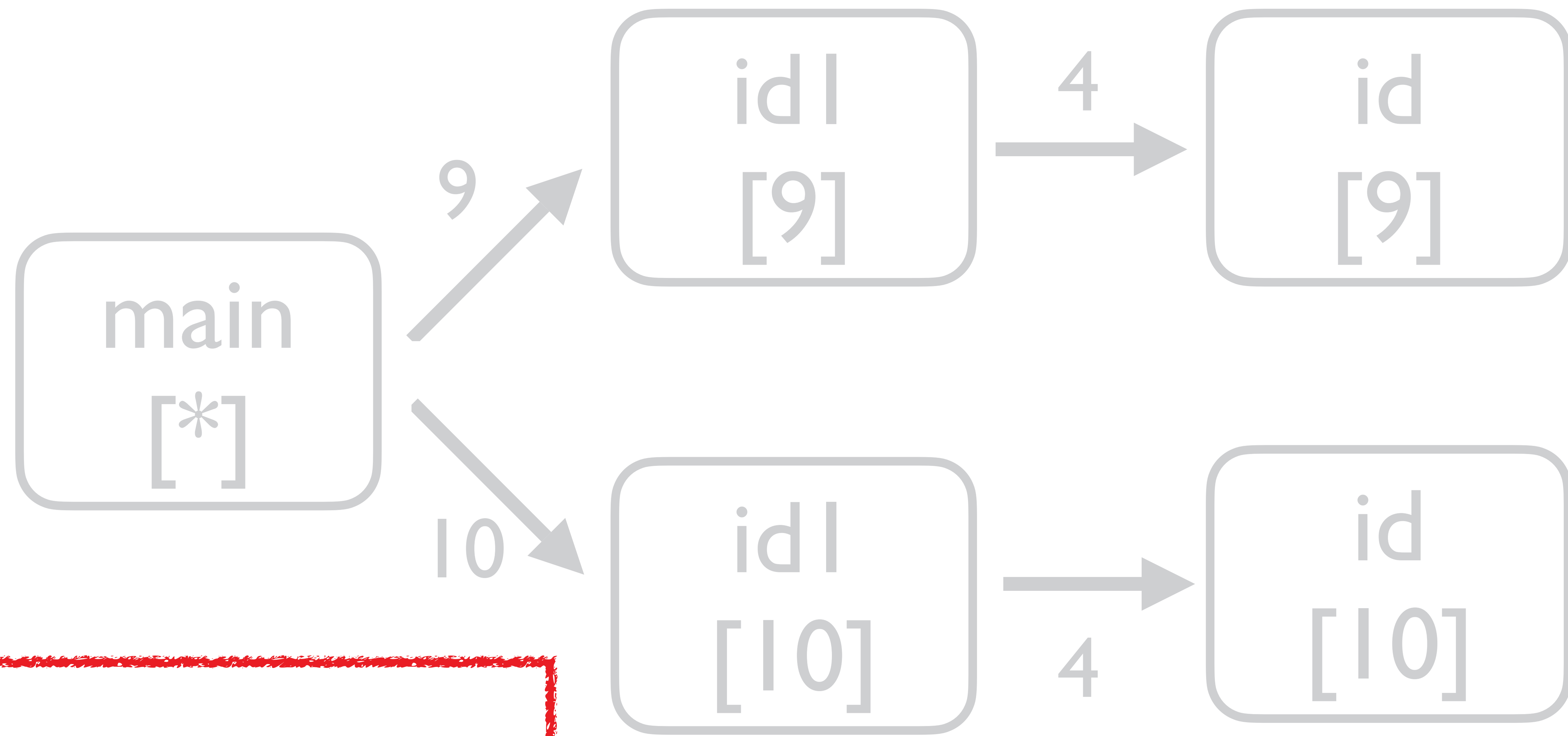


1-CFA with context tunneling
(T= {4})

Call-site Sensitivity vs Object Sensitivity

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

```
class C{
  id(v){
    return v;}
  id1(v){
    return id0(v);}
}
main(){
```



Tunneling abstraction:

Determines where to apply context tunneling

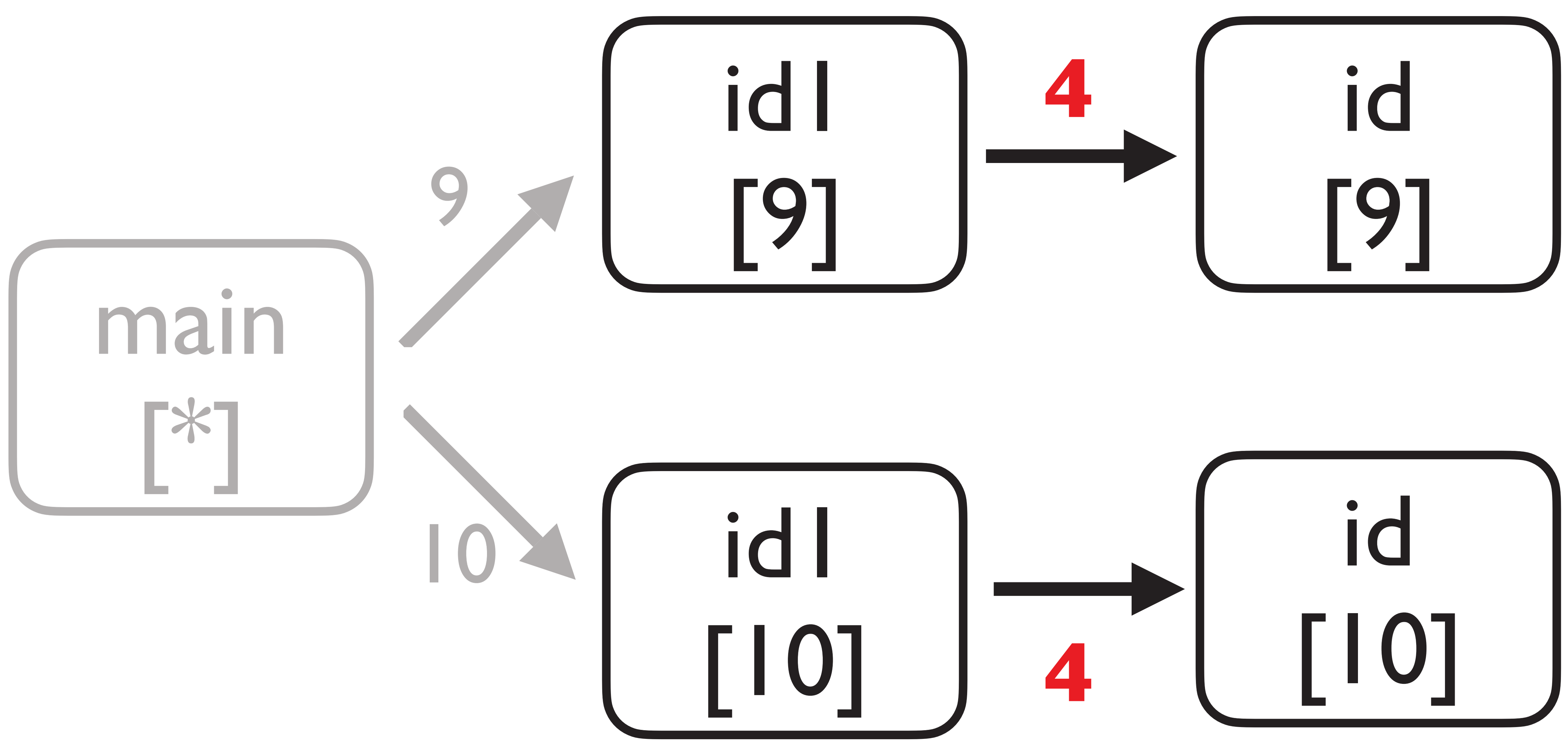
th context tunneling

(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})

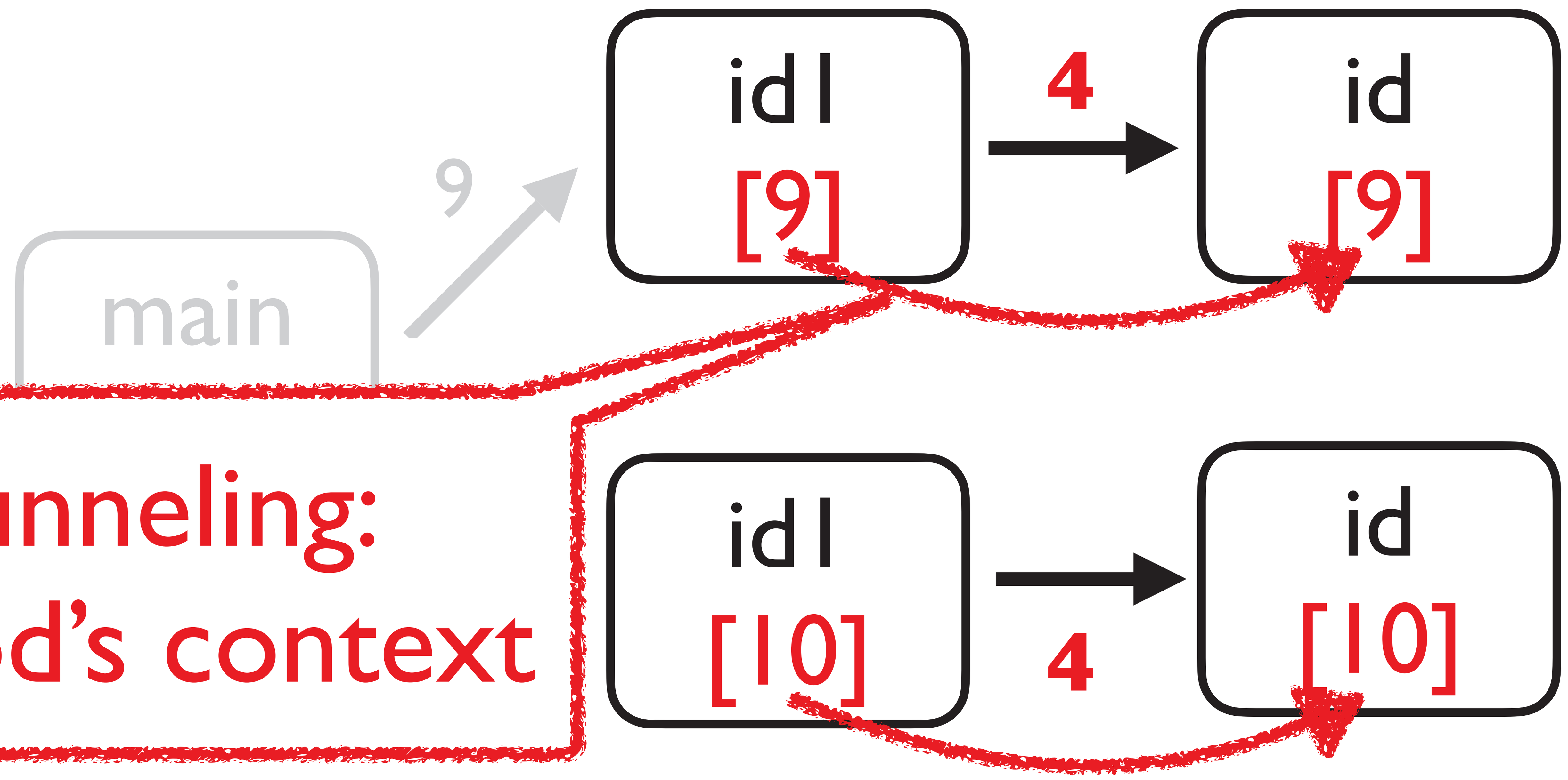
Unimportant call-sites that should not be used as context elements

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
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

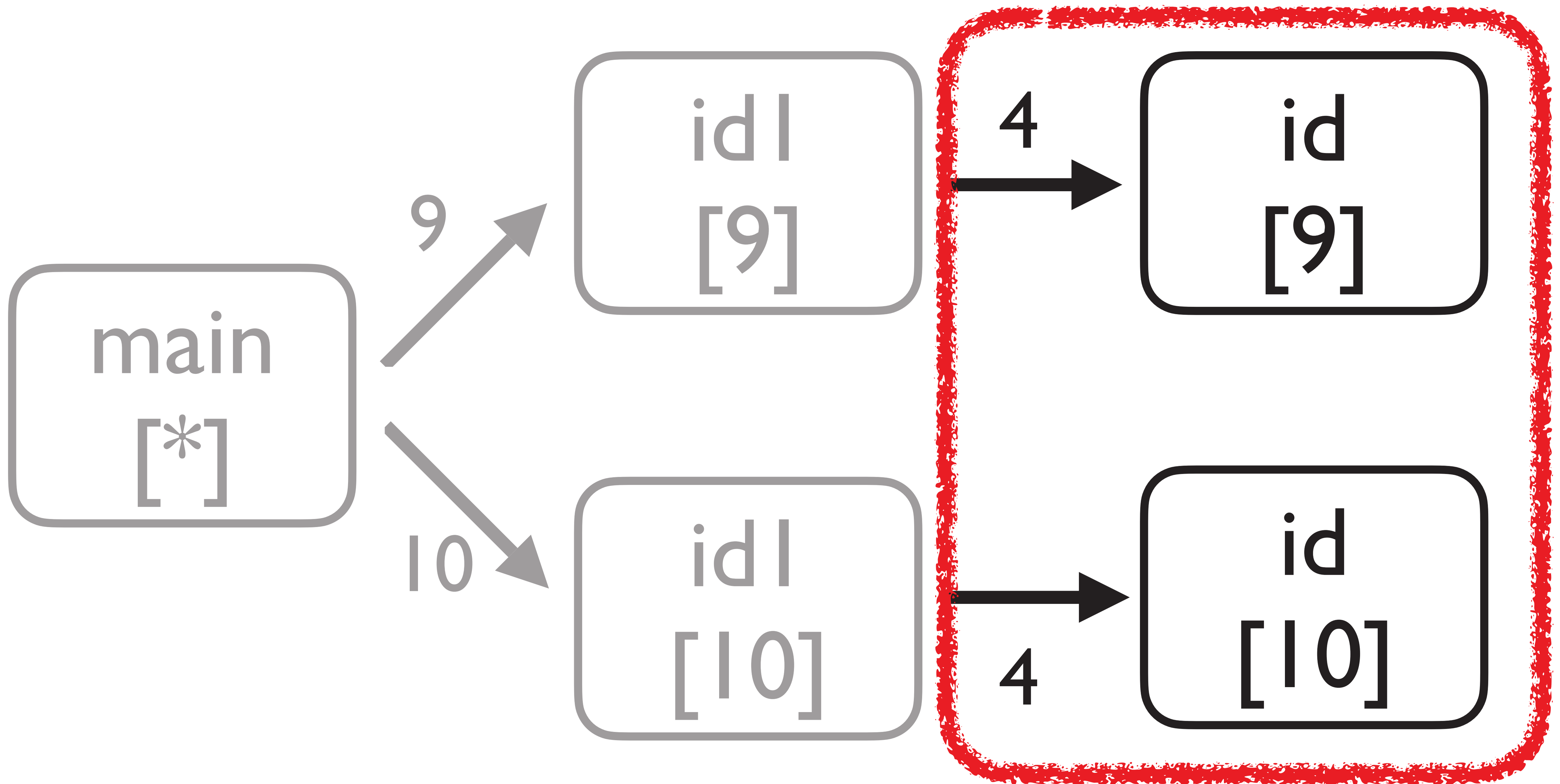


1-CFA with context tunneling
(T= {**4**})

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: }
```



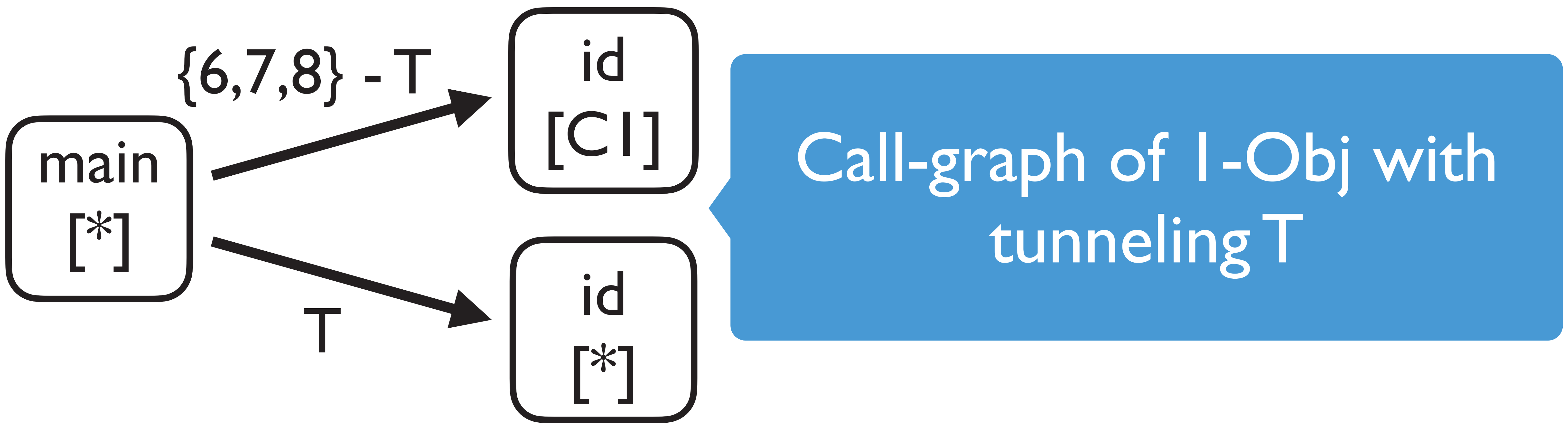
1-CFA with context tunneling (T= {4})

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:  }
```

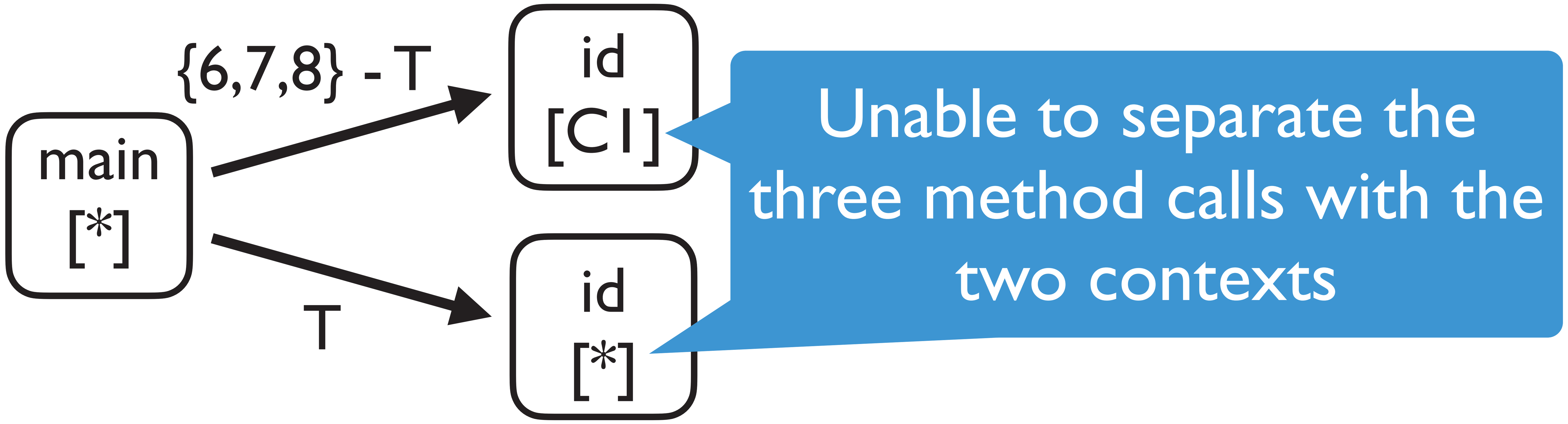


1-Obj + Tunneling
(T = ?)

Call-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:  }
```

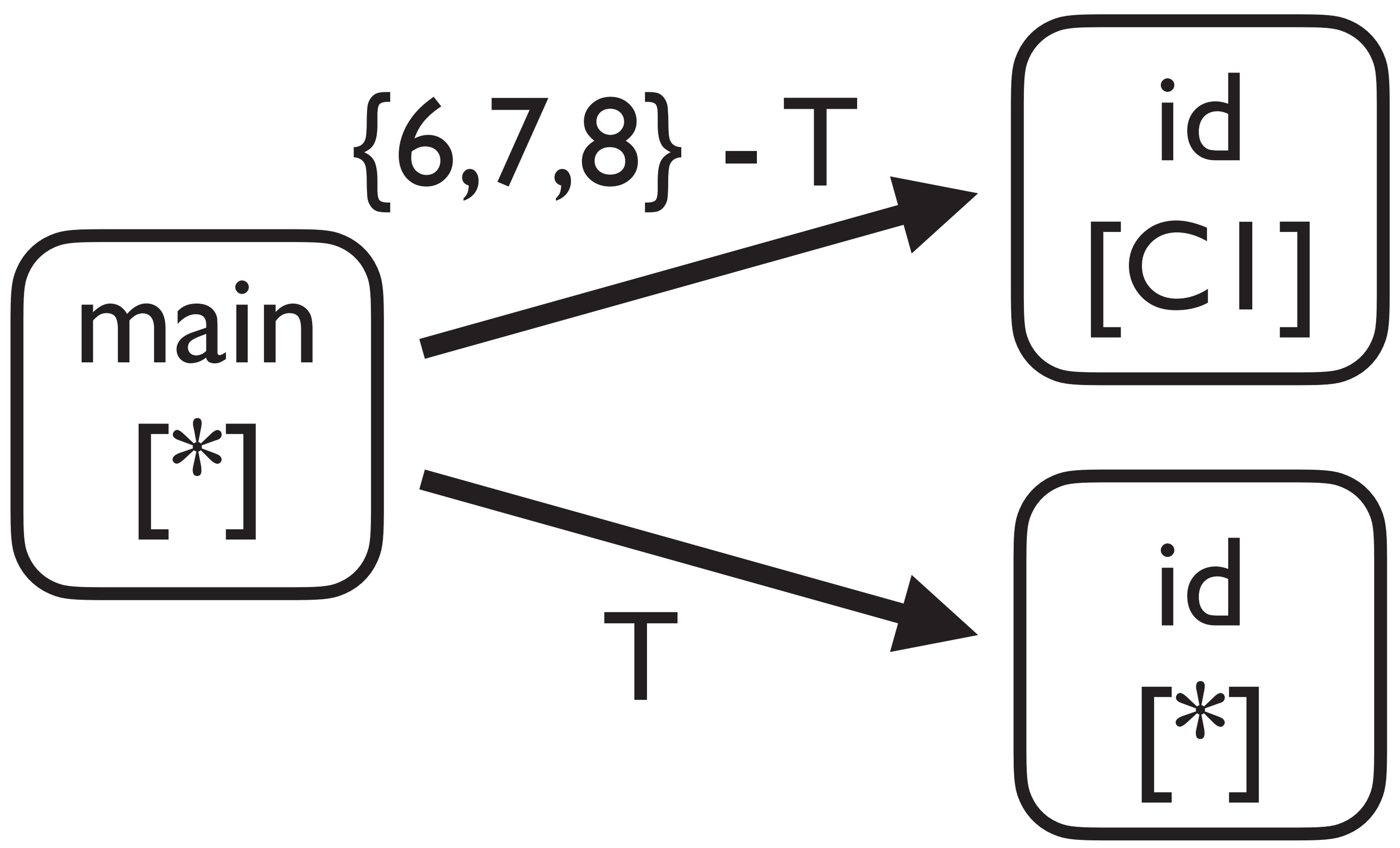


1-Obj + Tunneling
(T = ?)

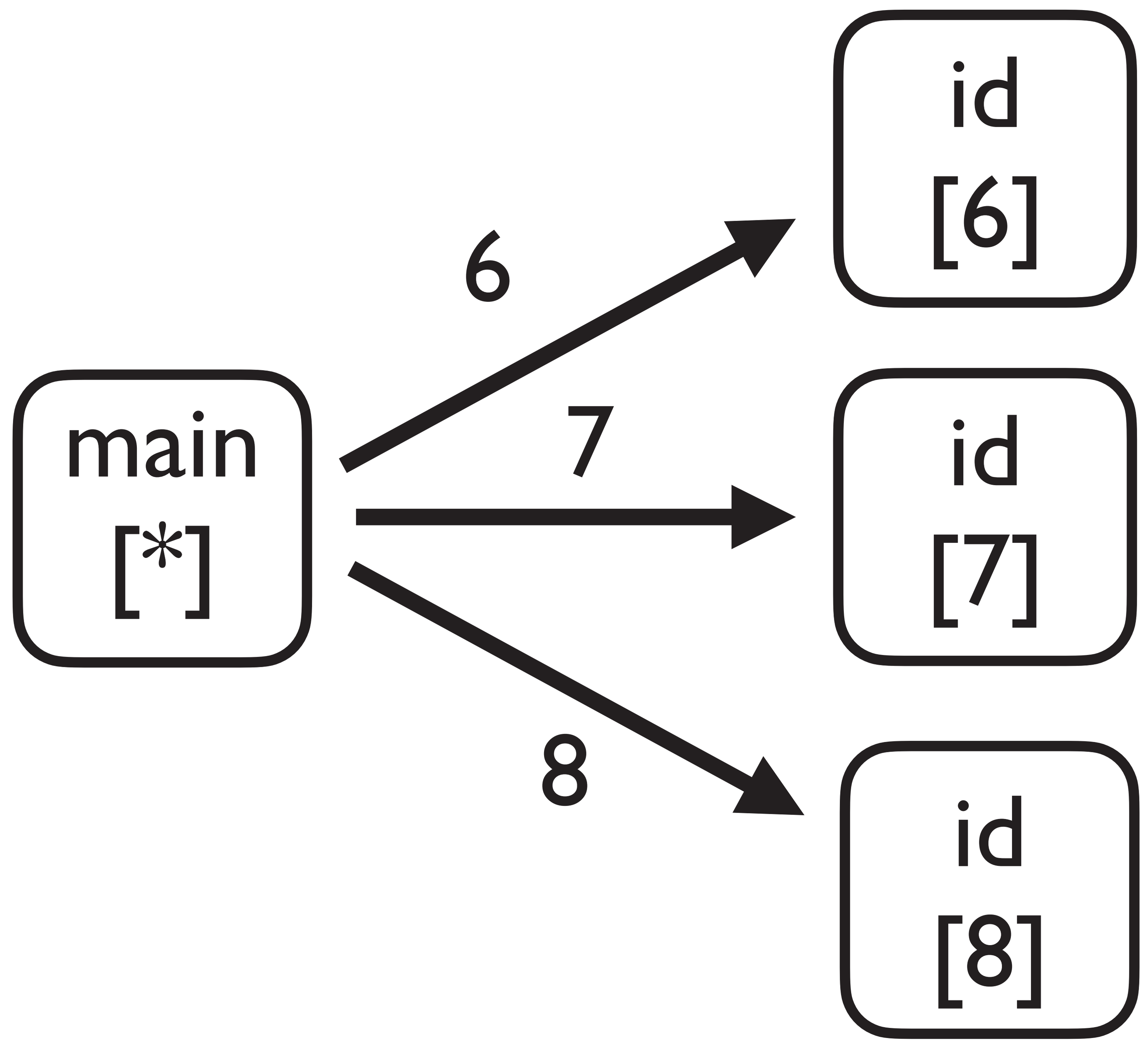
Call-site Sensitivity vs Object Sensitivity

- 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

Observation

When context tunneling is included

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

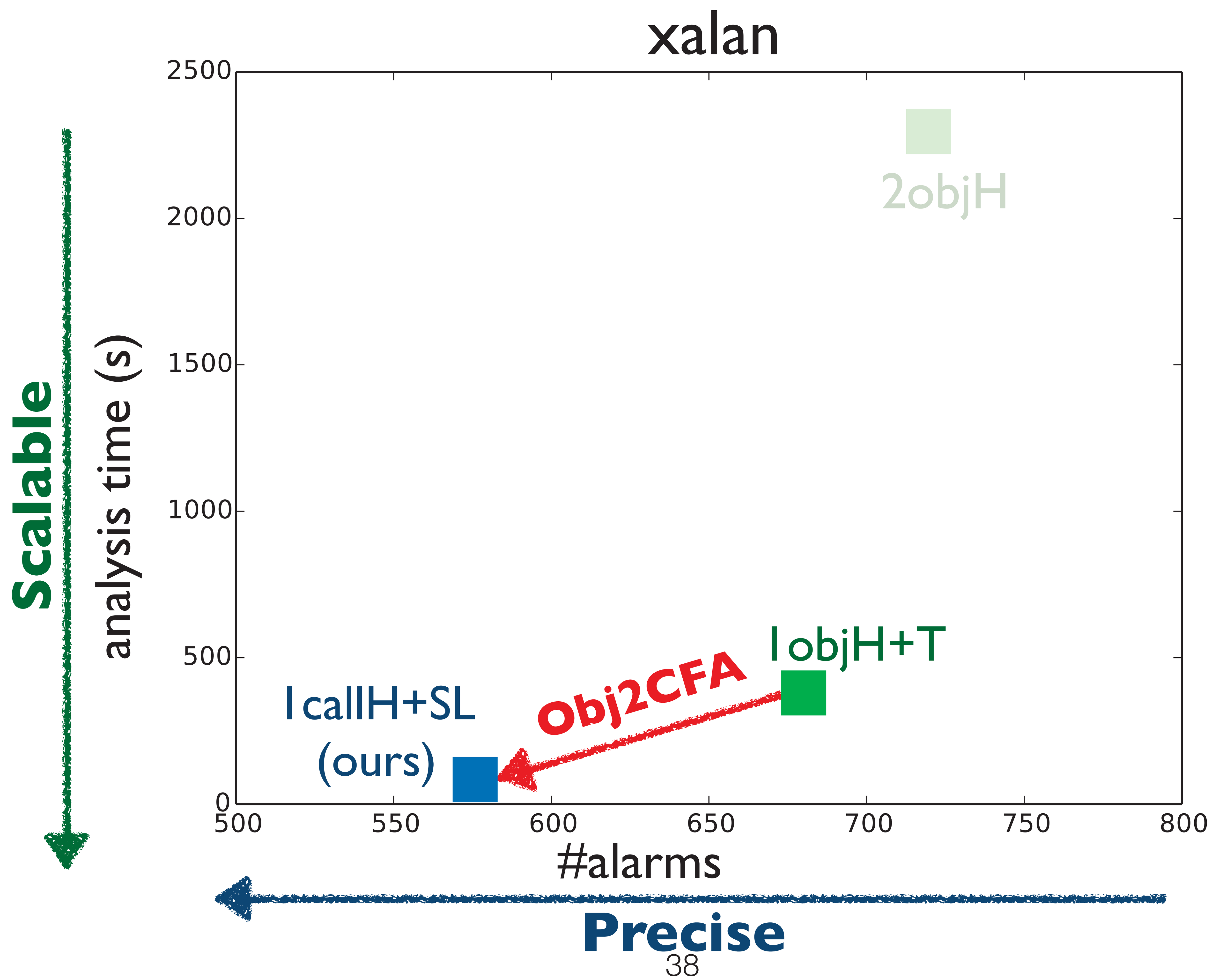
```
0: c
1:
2:
3: }
4: m
5:
6:   a
7:   b
8:   c
9: }
```

Our claim

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

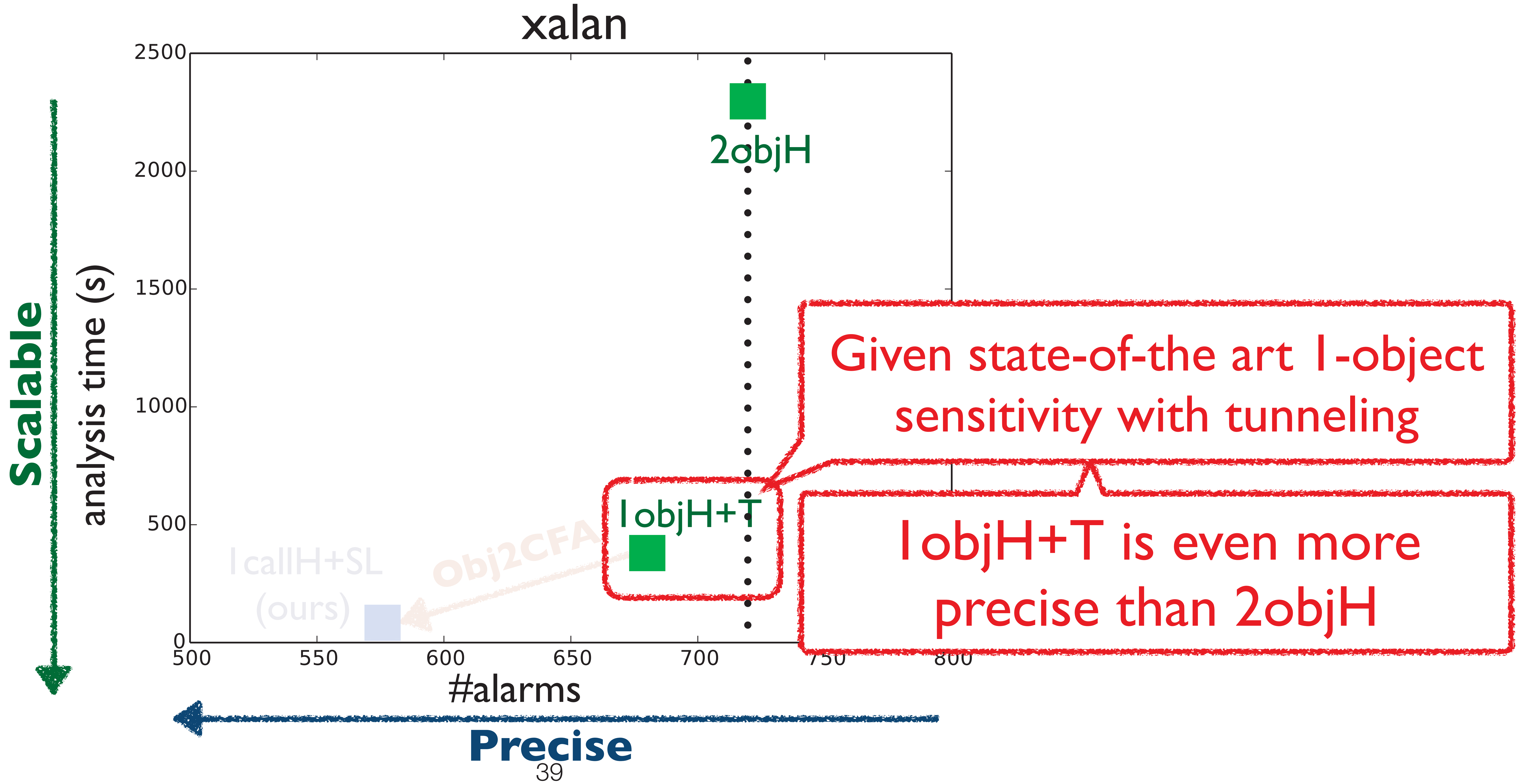
Our Technique : Obj2CFA

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



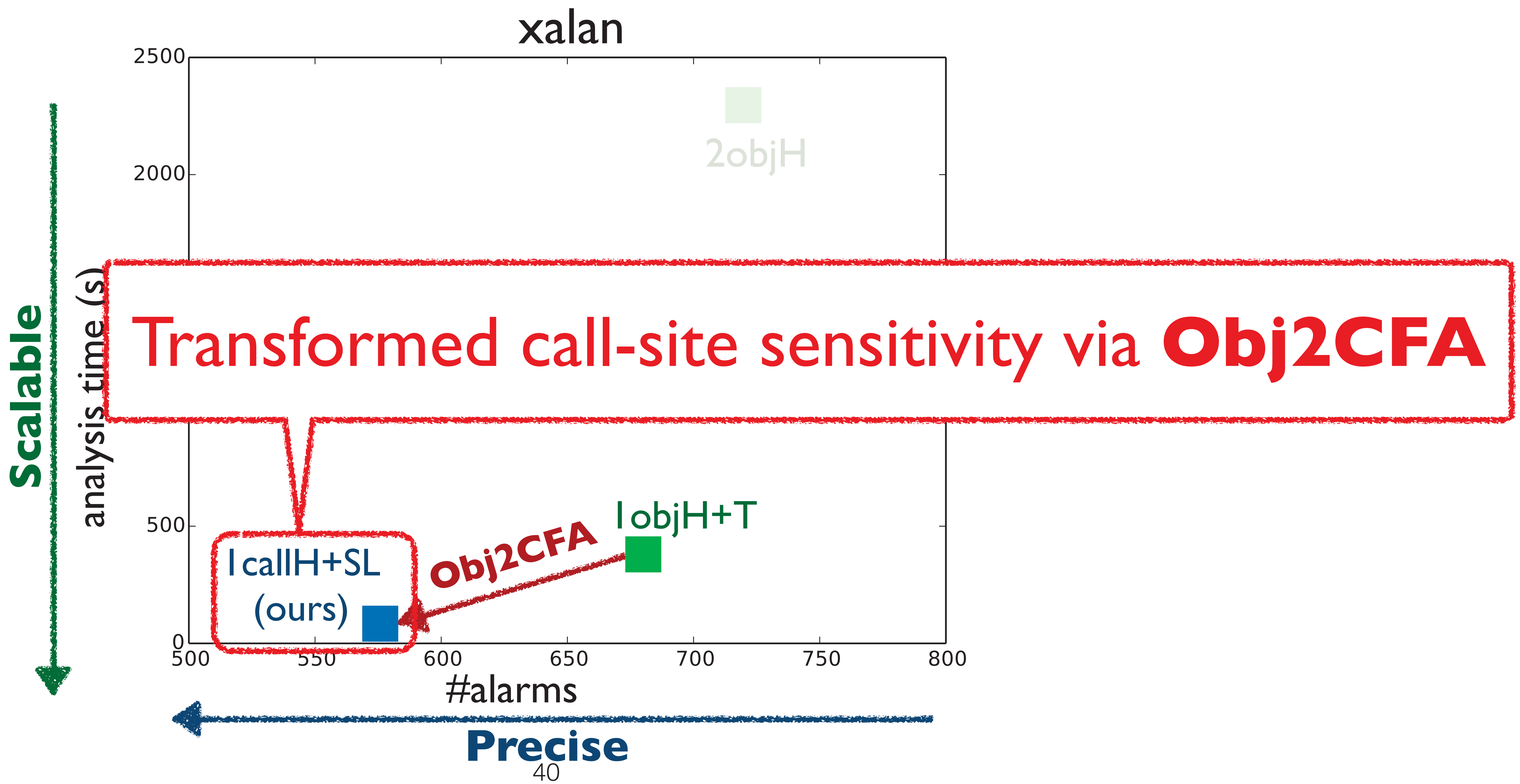
Our Technique : **Obj2CFA**

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



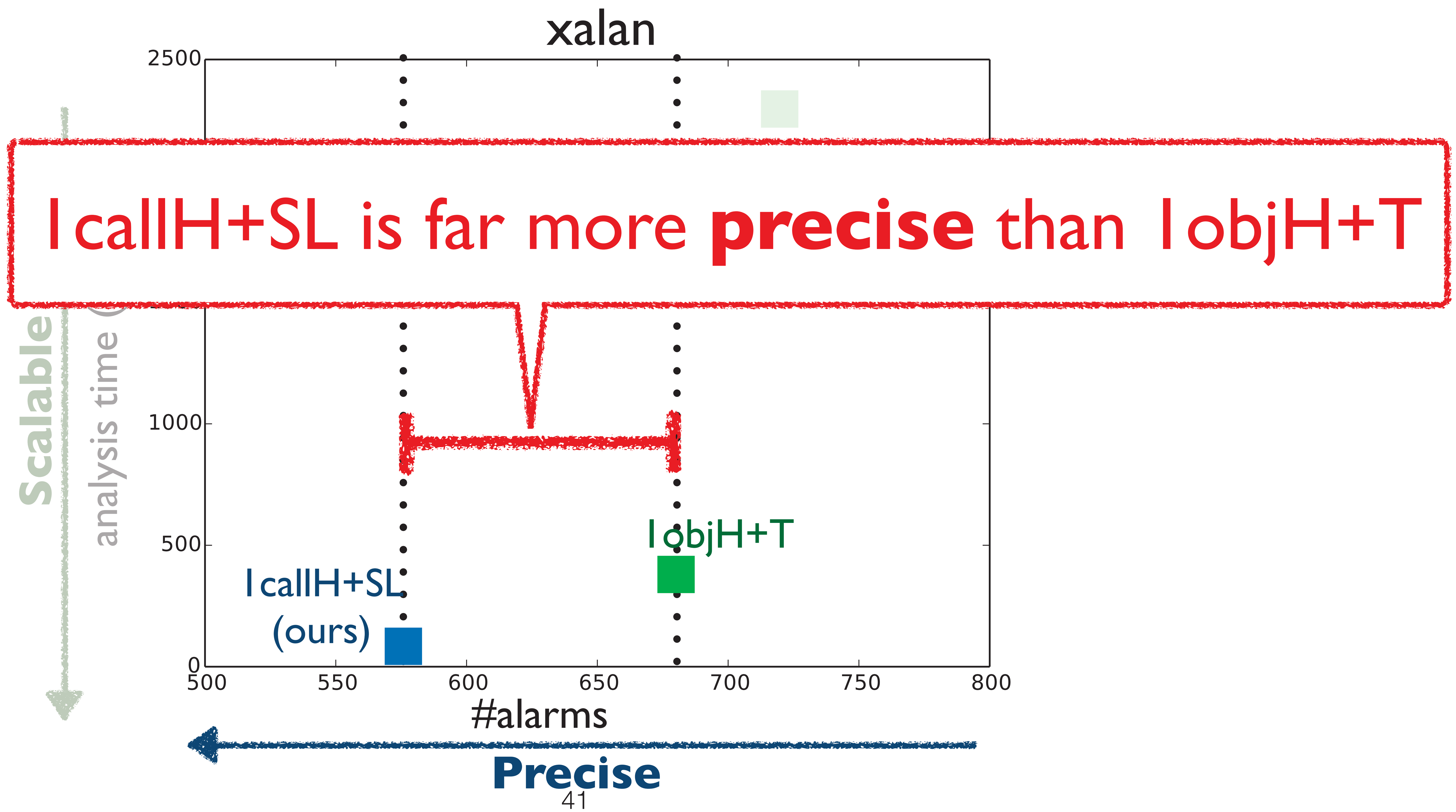
Our Technique : **Obj2CFA**

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



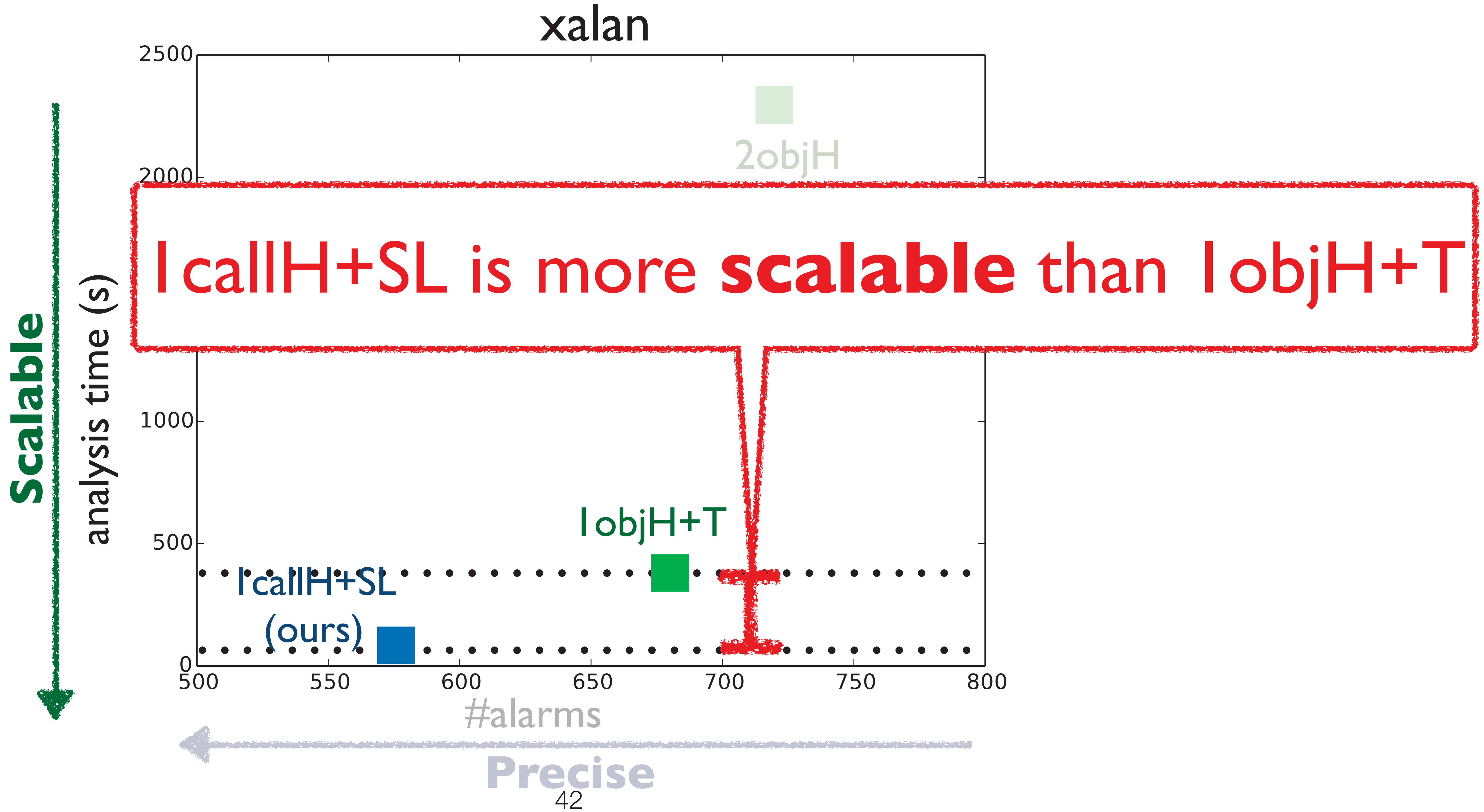
Our Technique : **Obj2CFA**

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



Our Technique : **Obj2CFA**

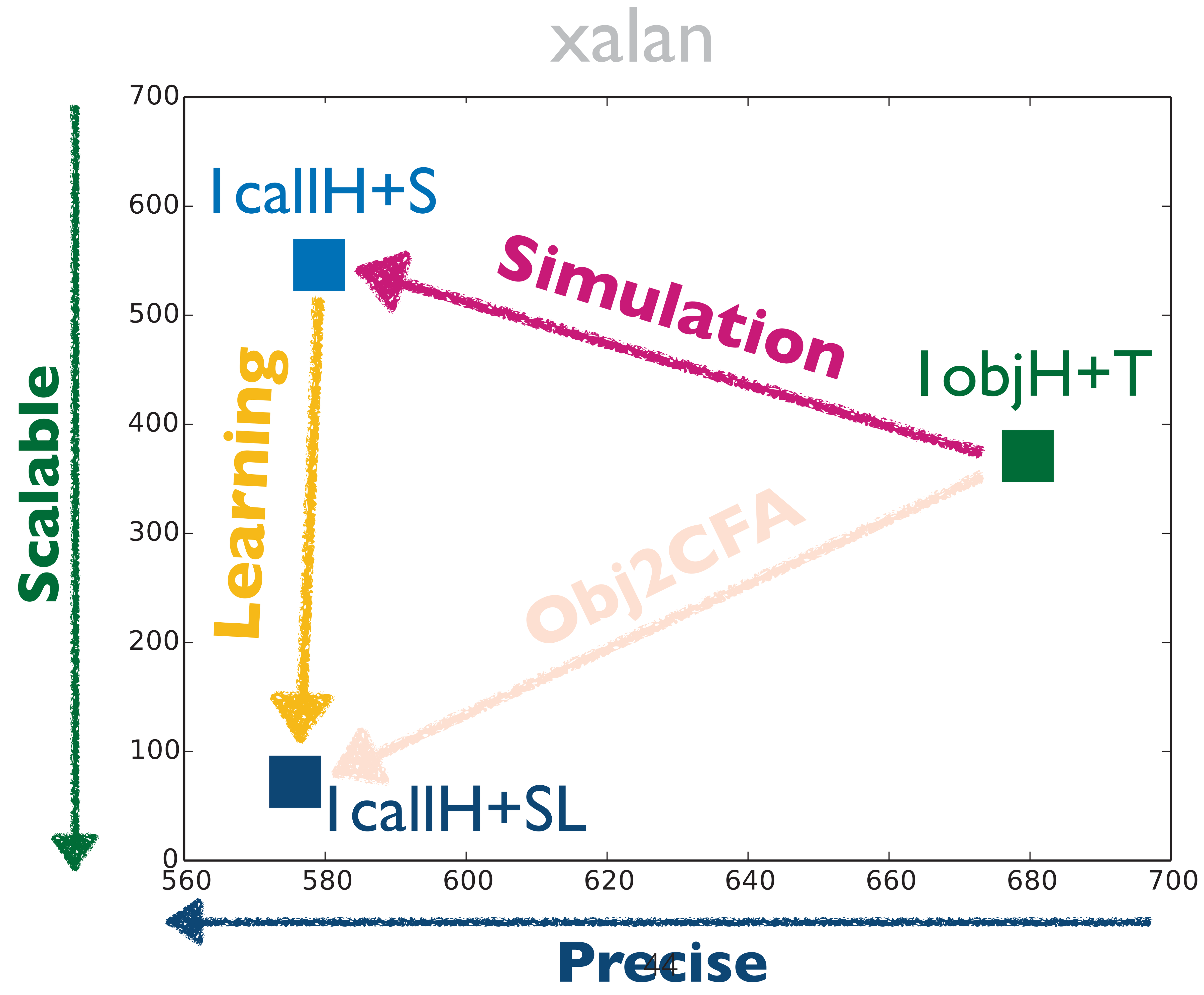
- **Obj2CFA** transforms a given object sensitivity into a more precise CFA



Detail of Obj2CFA

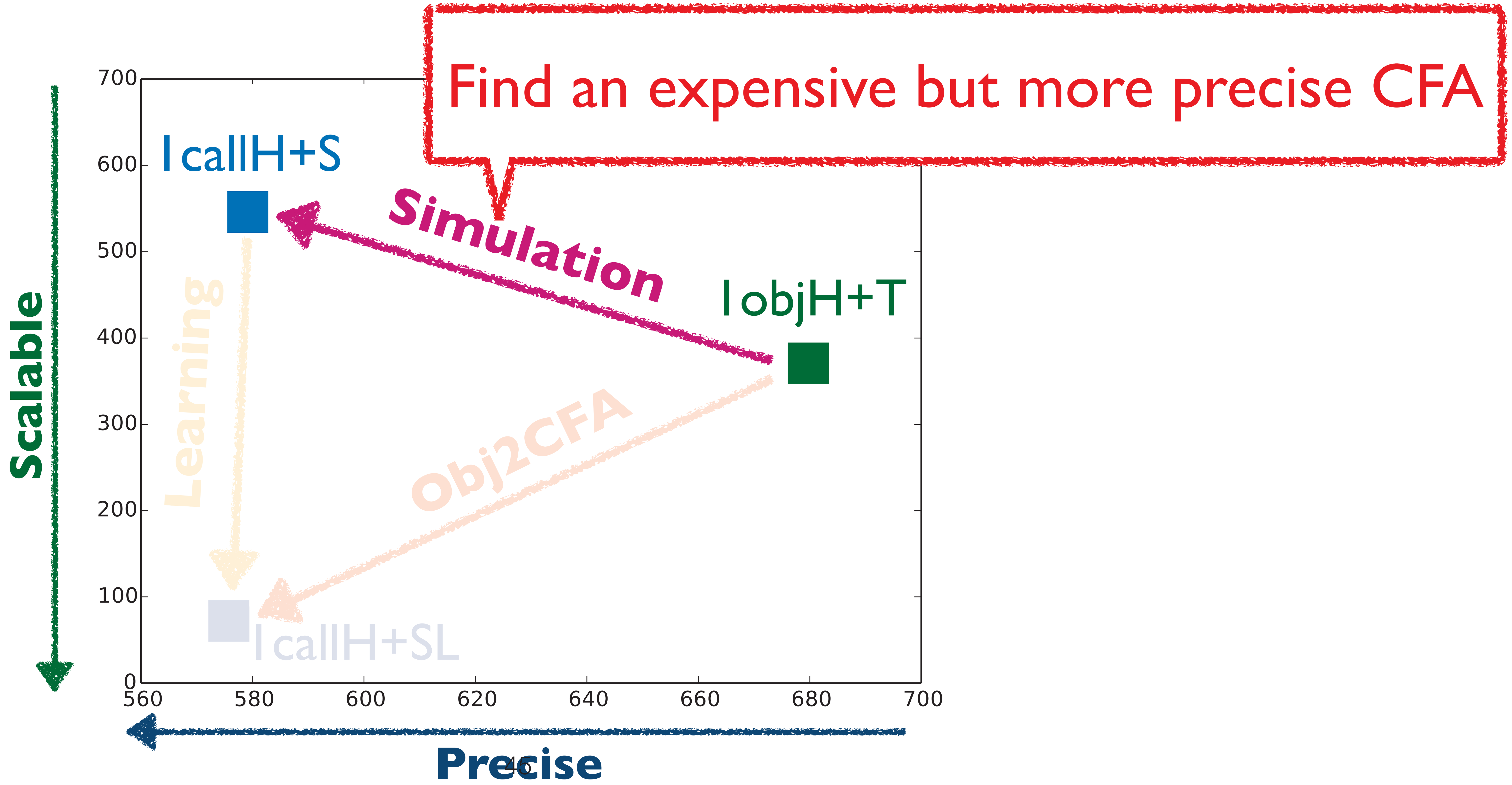
Our Technique : Obj2CFA

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



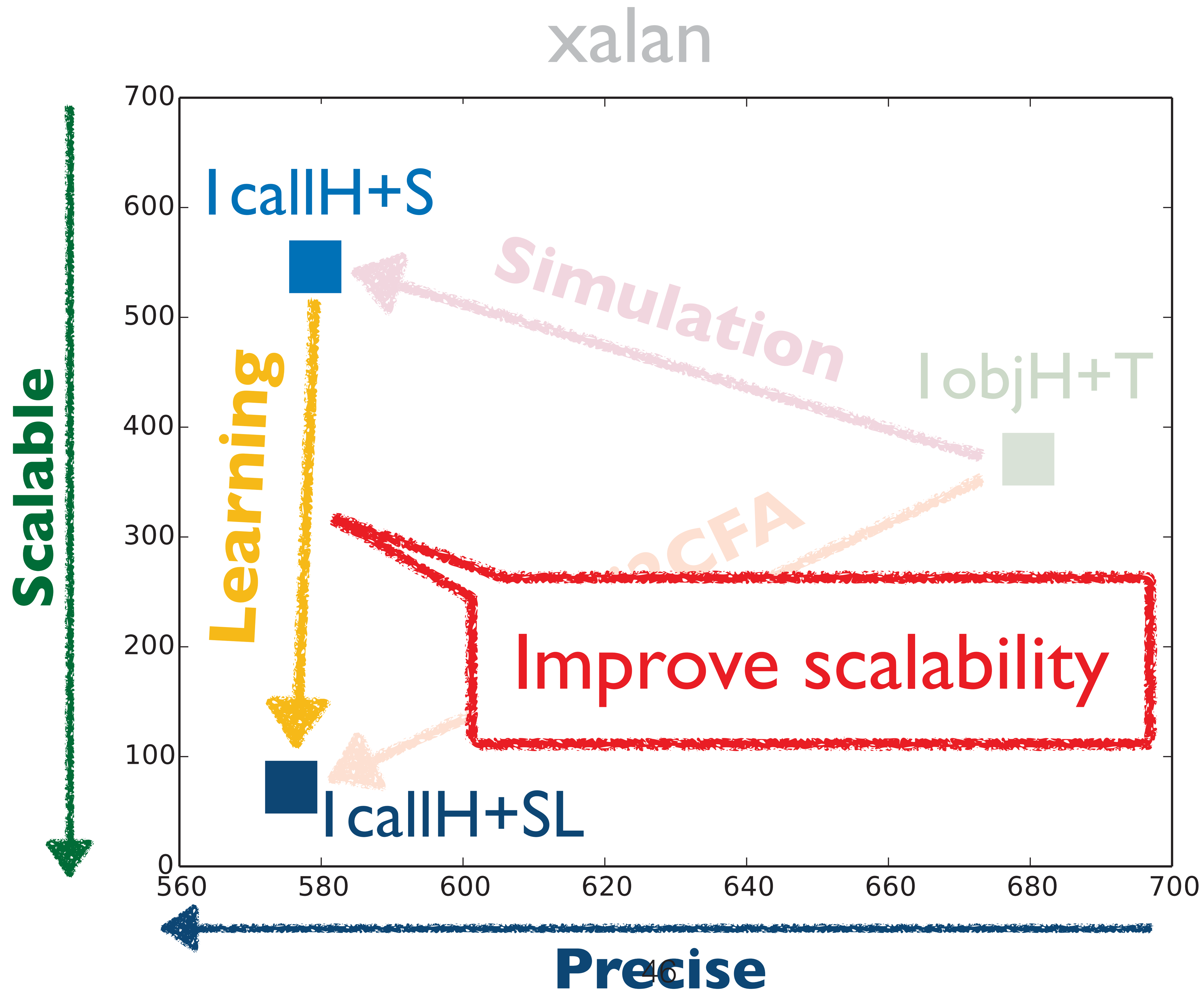
Our Technique : Obj2CFA

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



Our Technique : Obj2CFA

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



Technique 1: Simulation

- Running example to illustrate Simulation

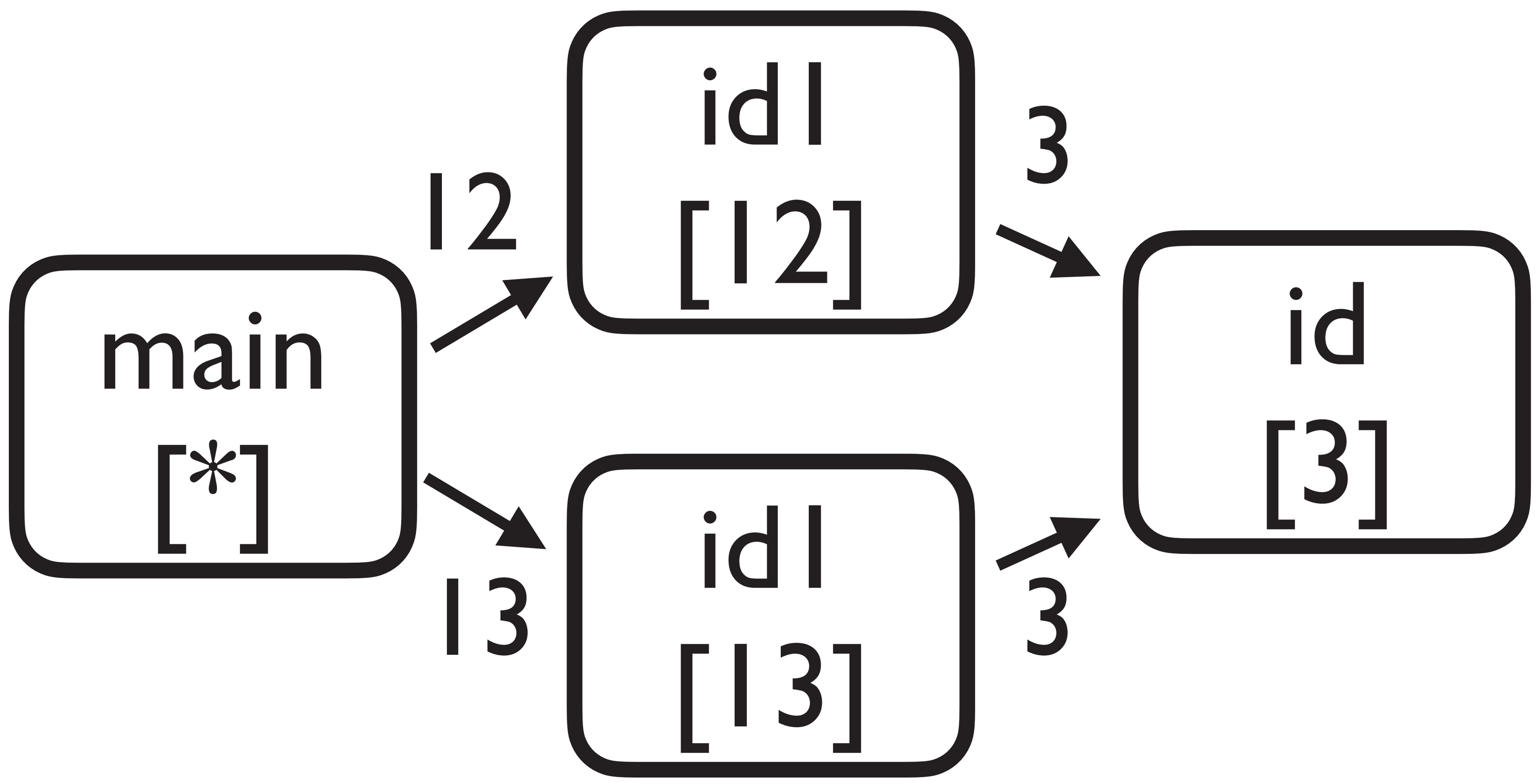
```
class C{
  id(v){return v;}
  id1(v){return id(v);}
  foo(){
    A a = (A) this.id(new A());}//query1
    B b = (B) this.id(new B());}//query2
}
main(){
  c1 = new C(); //C1
  c2 = new C(); //C2
  c3 = new C(); //C3
  A a = (A) c1.id1(new A());//query3
  B b = (B) c2.id1(new B());//query4
  c3.foo();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

Technique 1: Simulation

- Running example to illustrate Simulation

```
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();
}
```

Limitation of conventional 1-CFA

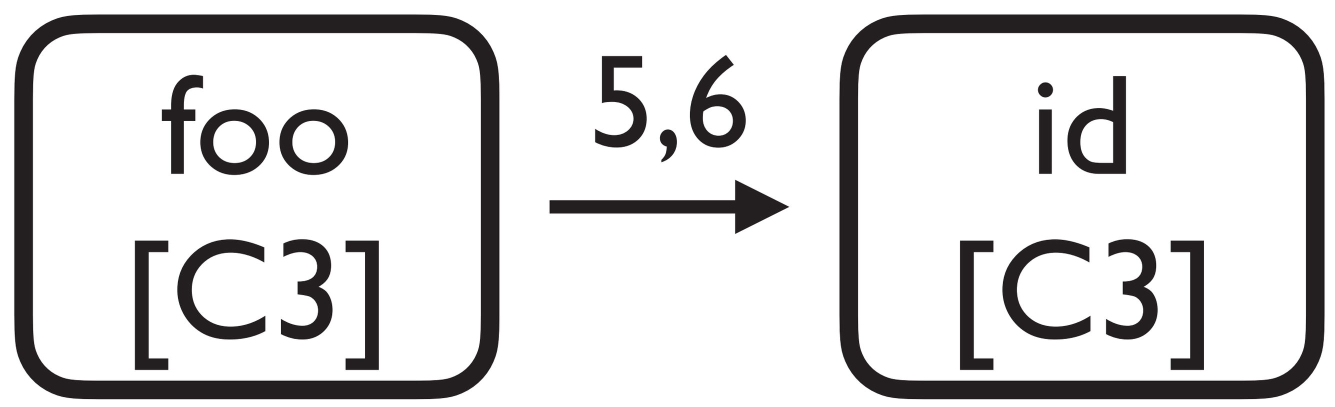


Technique 1: Simulation

- 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: }
```

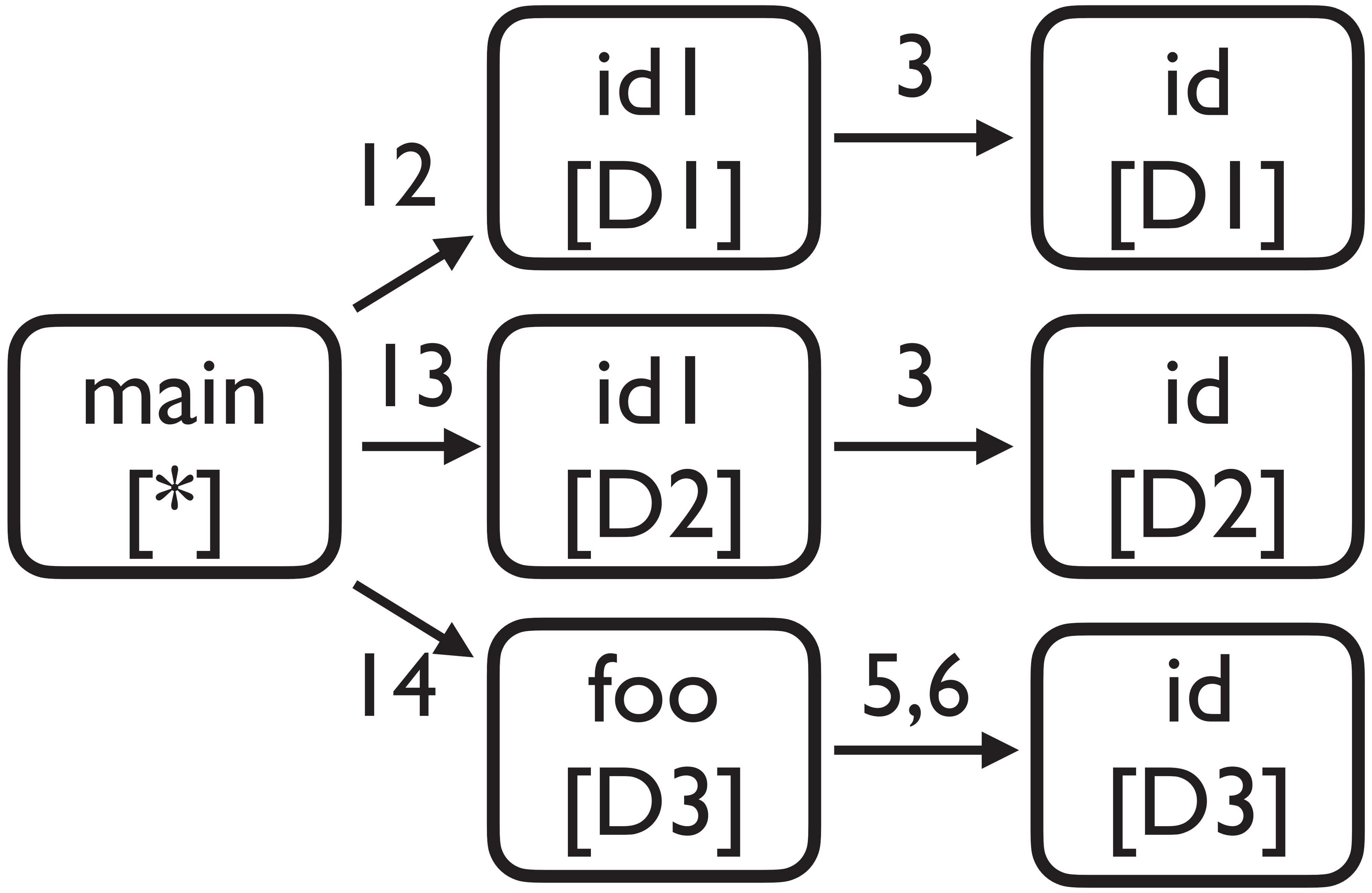
Limitation of object sensitivity



Technique 1: Simulation

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

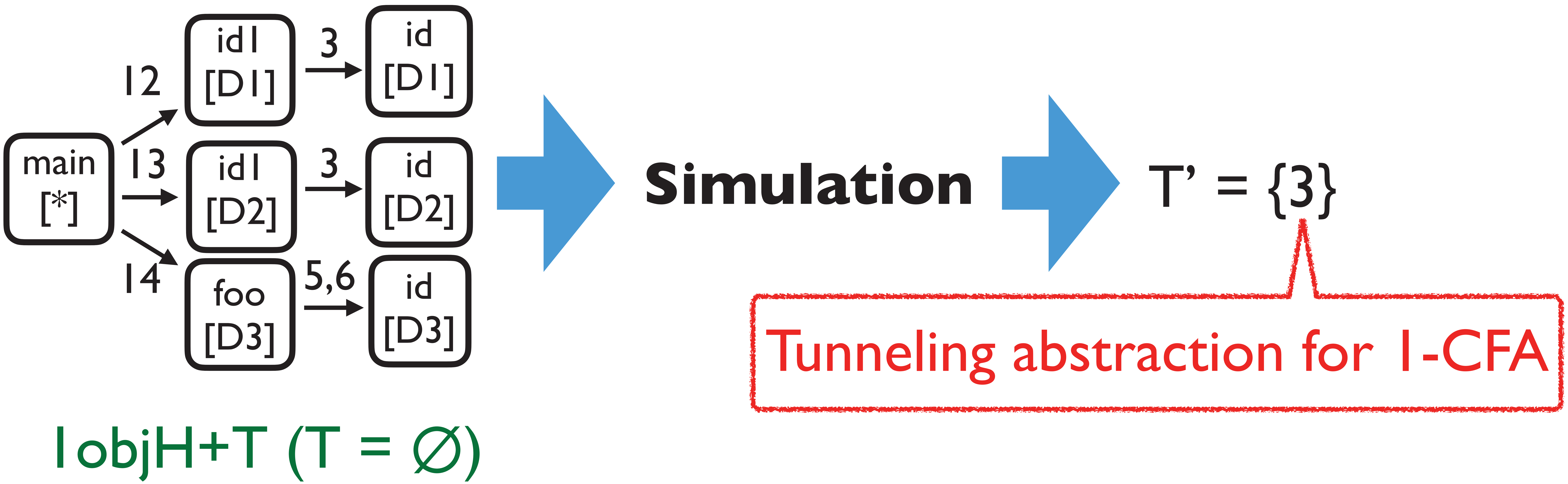
```
1:  class C{
2:    id(v){return v;}
3:    id1(v){return id(v);}
4:    foo(){
5:      A a = (A) this.id(new A());}//query1
6:      B b = (B) this.id(new B());}//query2
7:  }
8:  main(){
9:    c1 = new C(); //C1
10:   c2 = new C(); //C2
11:   c3 = new C(); //C3
12:   A a = (A) c1.id1(new A());//query3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



1objH+T (T = ∅)

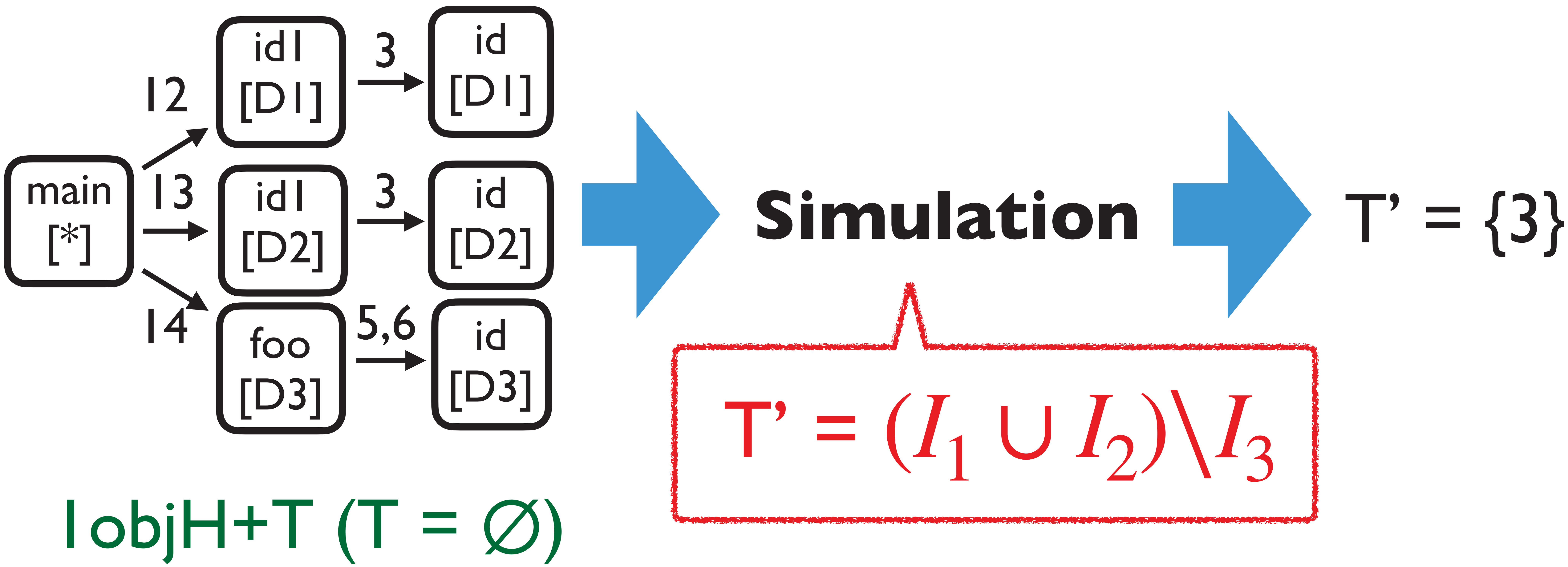
Technique 1: Simulation

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



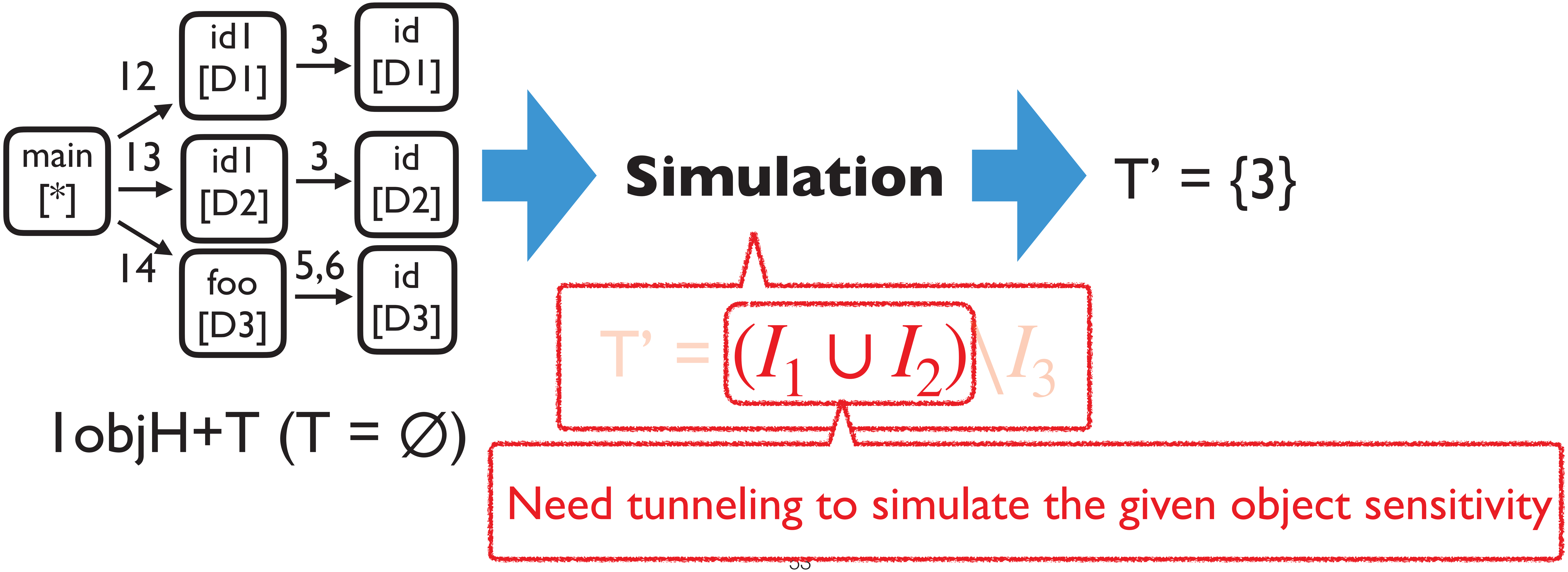
Technique 1: Simulation

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



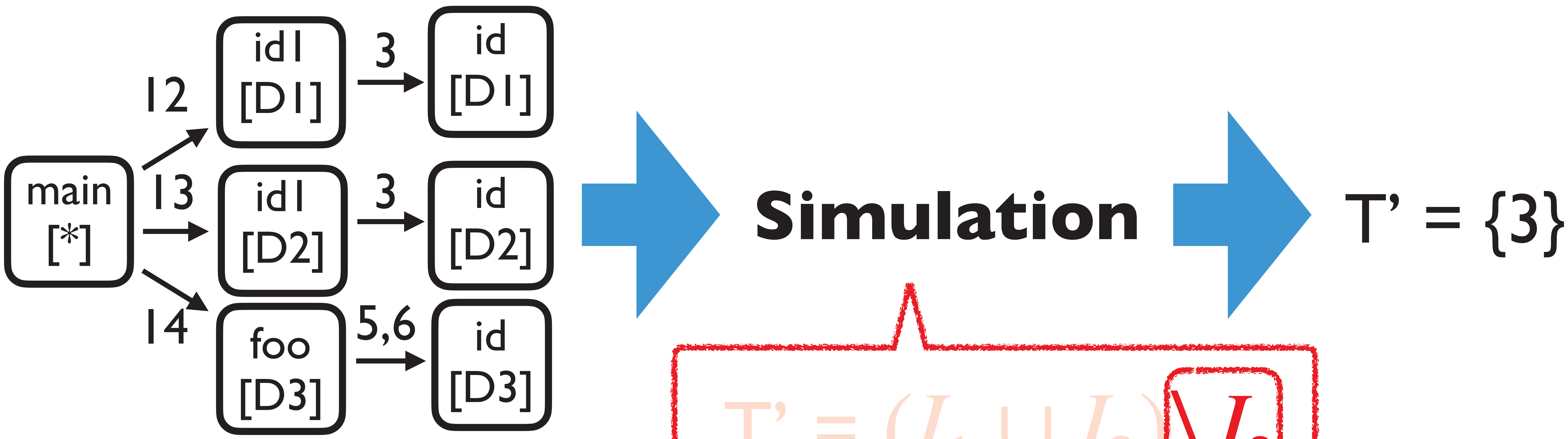
Technique 1: Simulation

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



Technique 1: Simulation

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



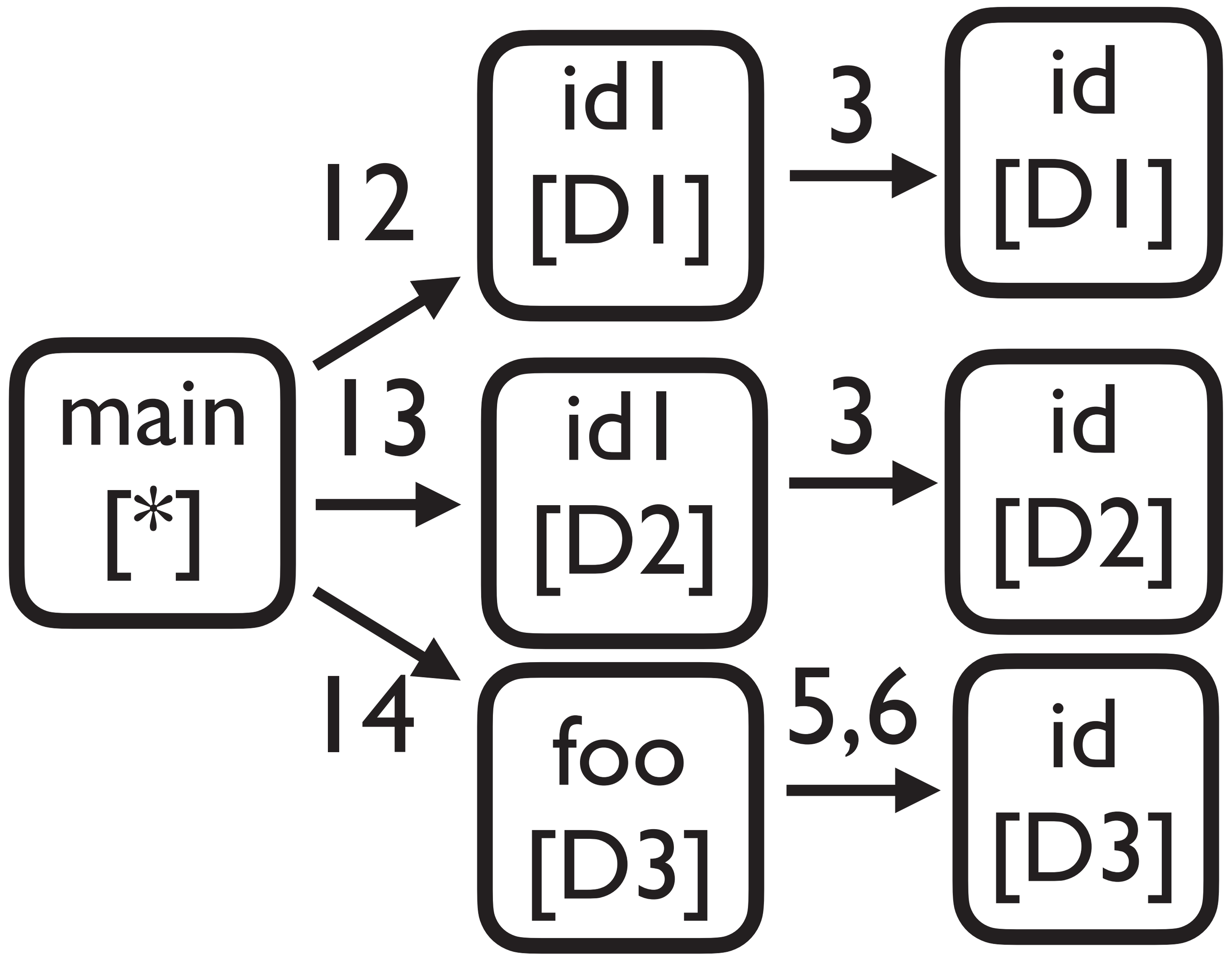
1objH+T (T = ∅)

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

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'?

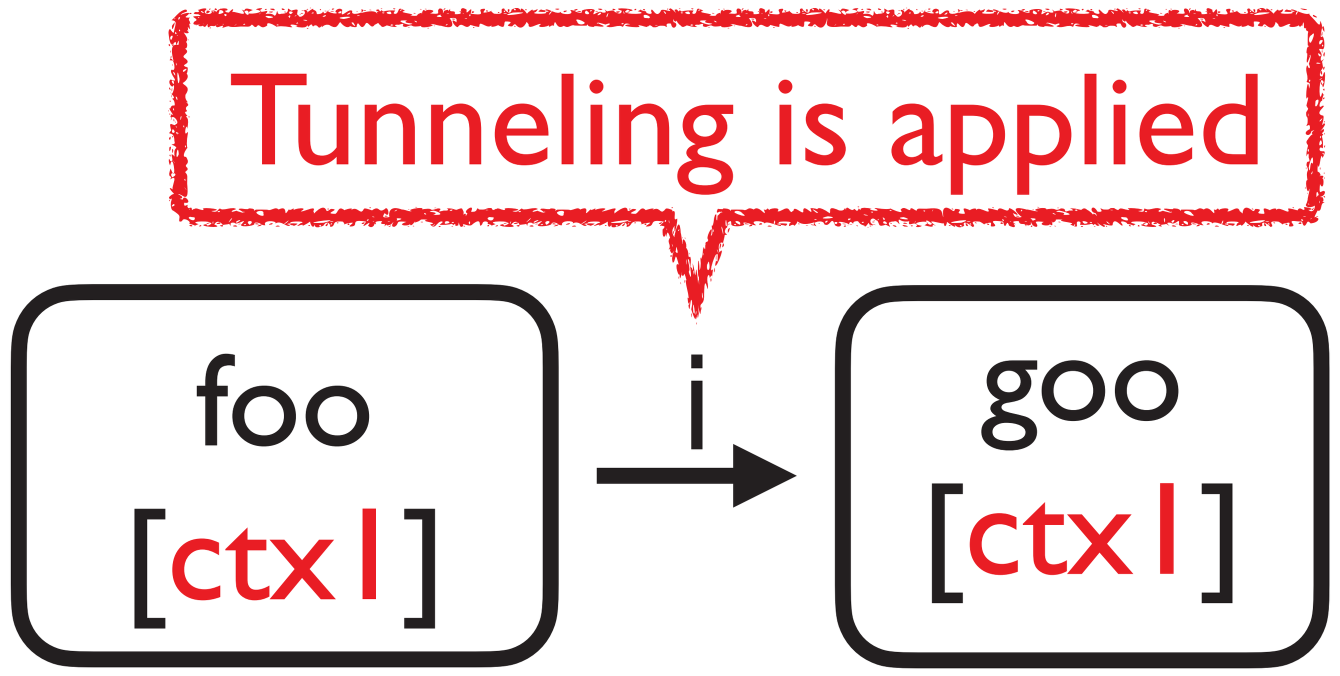
Intuition Behind Simulation $(I_1 \cup I_2)$

• 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



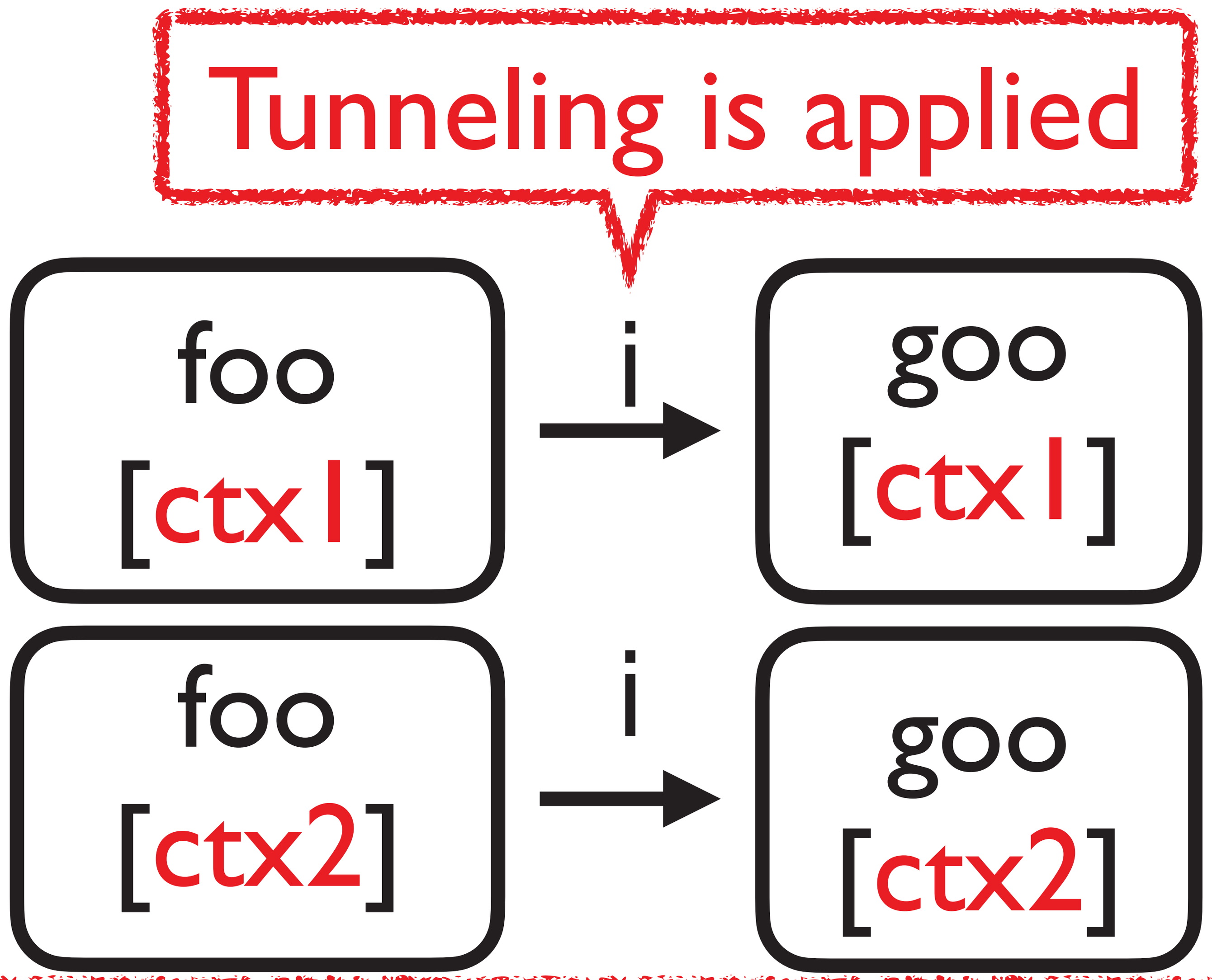
Property of context tunneled call-sites

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



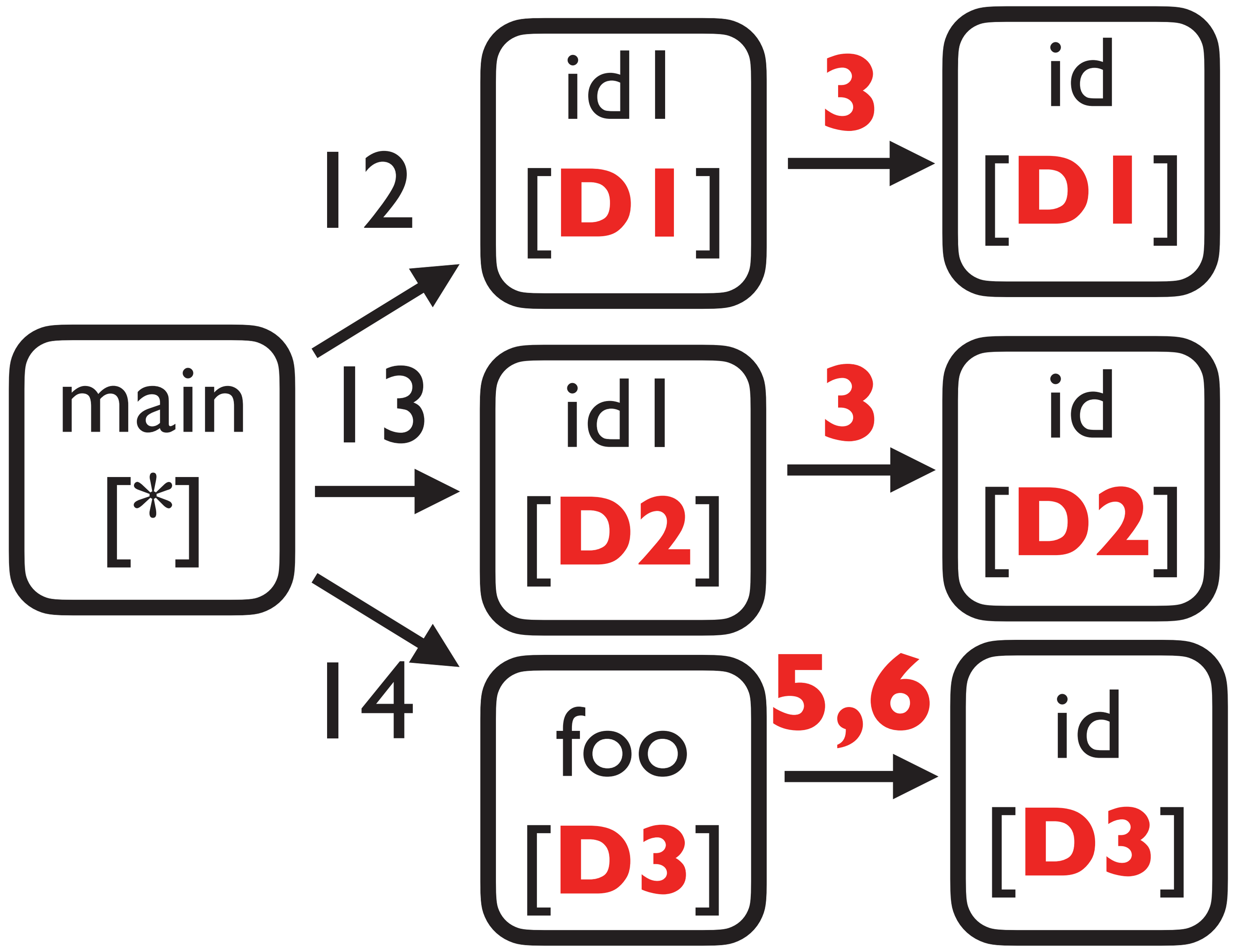
Property of context tunneled invocations

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}

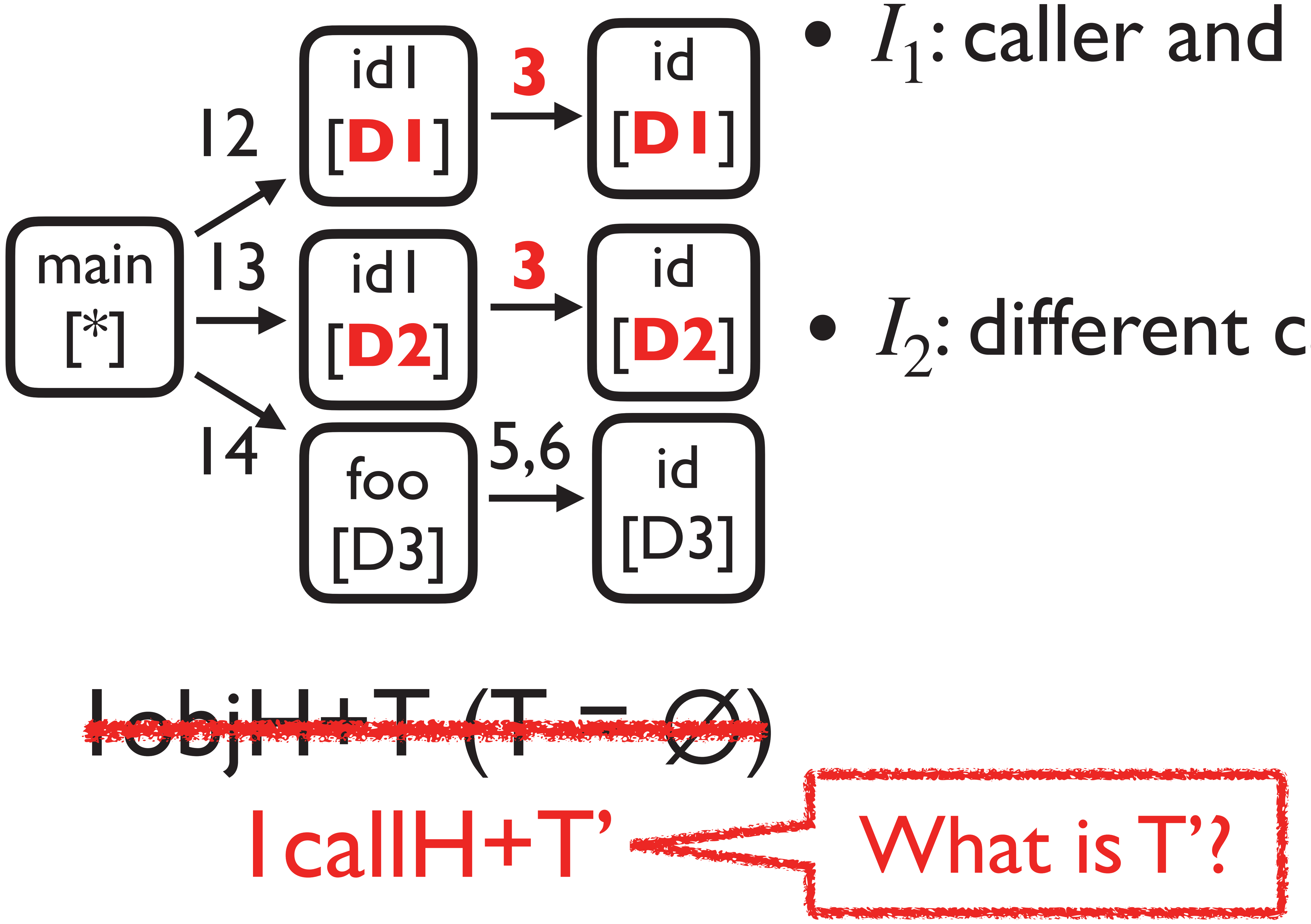
~~1objH+T (T = ∅)~~

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

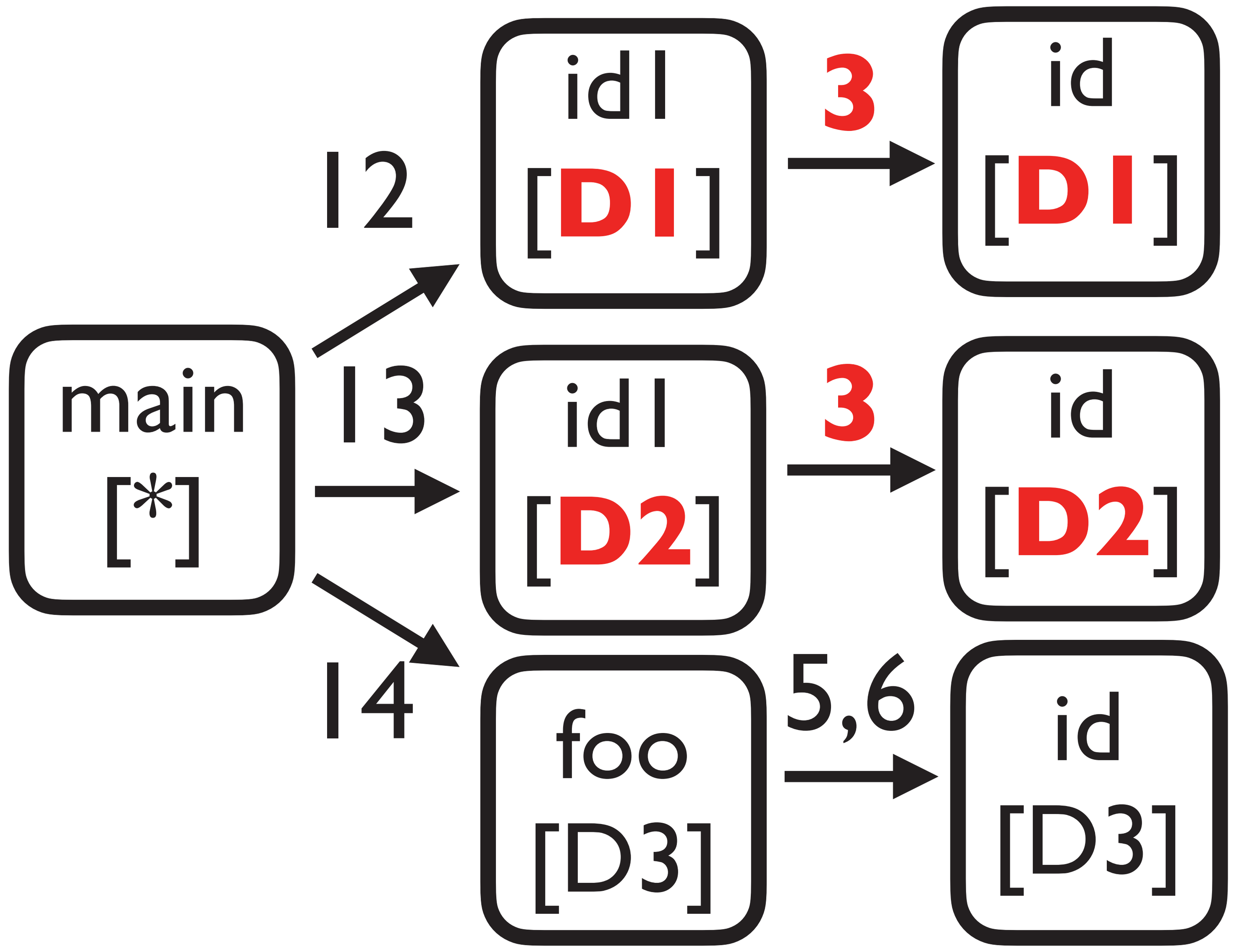
I_1={3,5,6}

- I_2: different caller ctx imply different callee ctx

I_2={3}

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

$$I_2=\{3\}$$

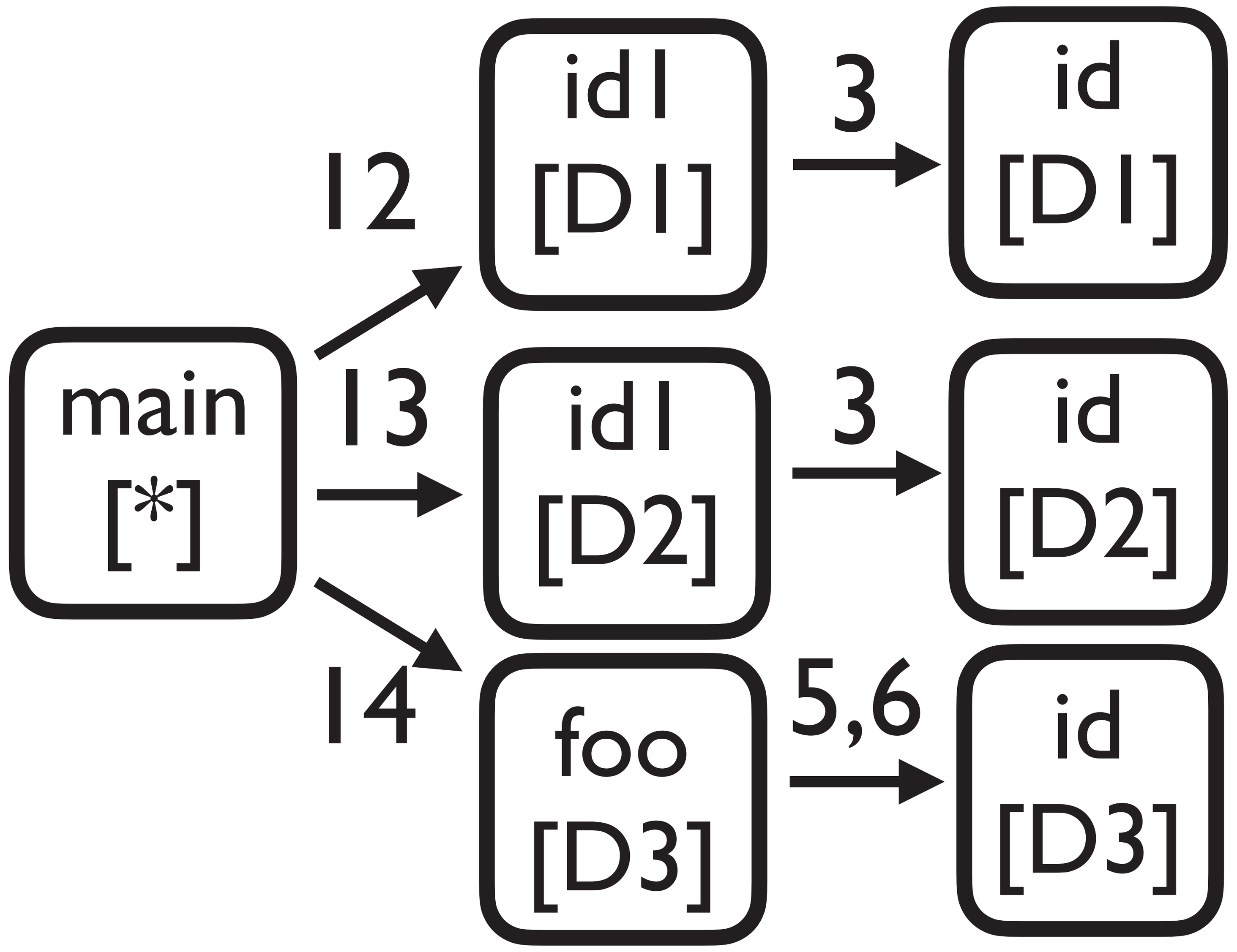
~~1objH+T (T = ∅)~~

1callH+T' ← What is T'?

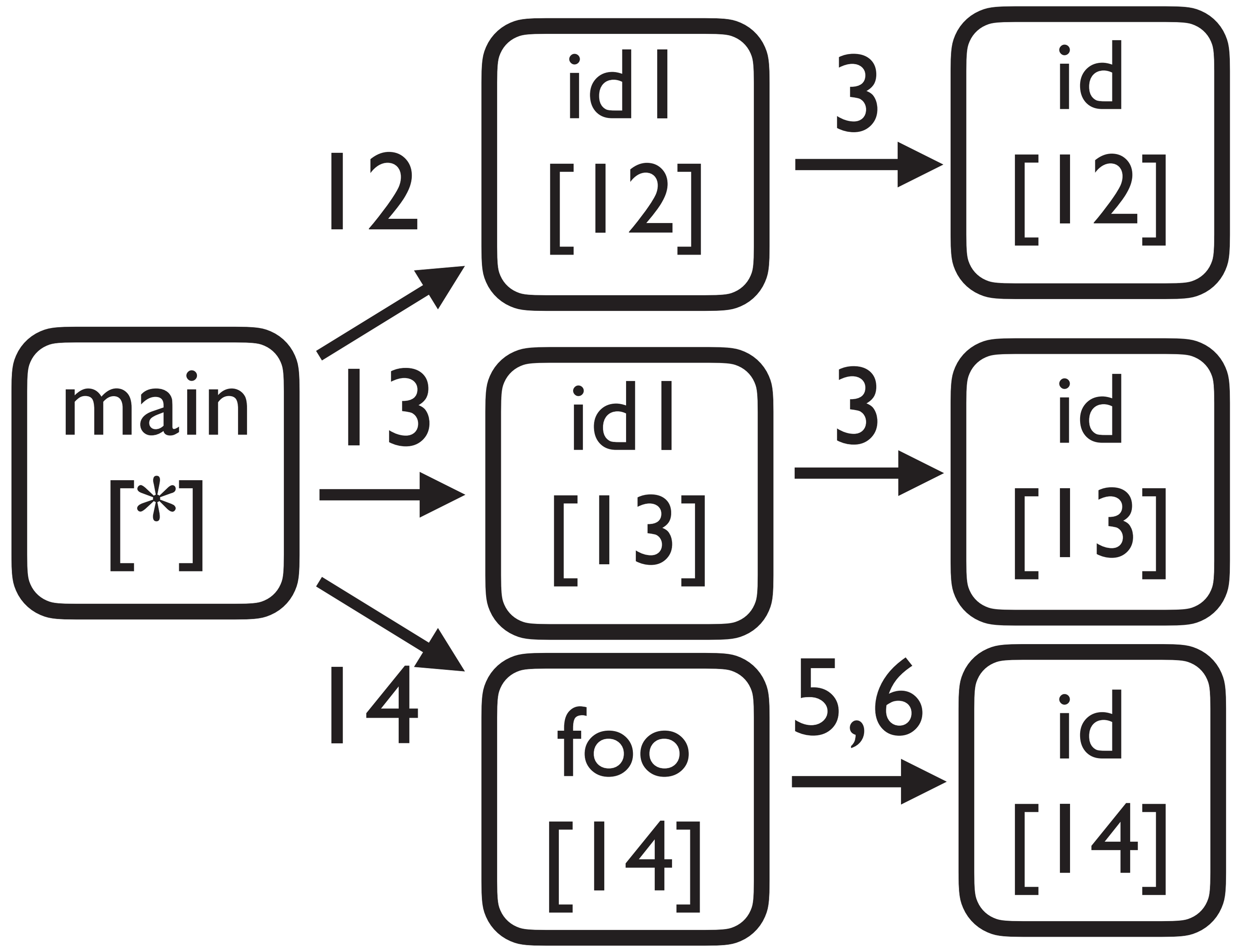
$$T' = I_1 \cup I_2 = \{3,5,6\}$$

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 = ∅)

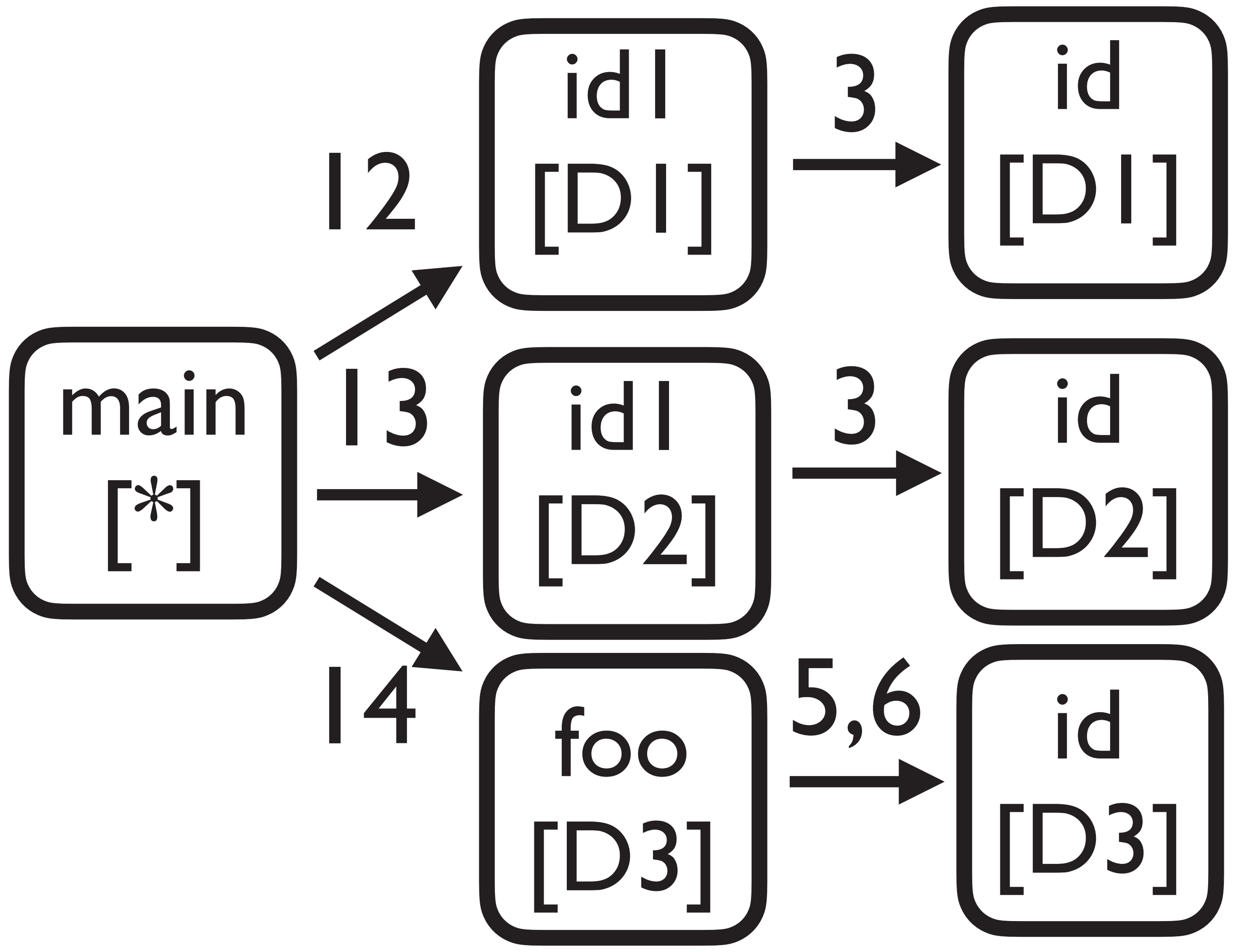


1callH+T' (T' = {3,5,6})

Intuition Behind Simulation $(I_1 \cup I_2)$

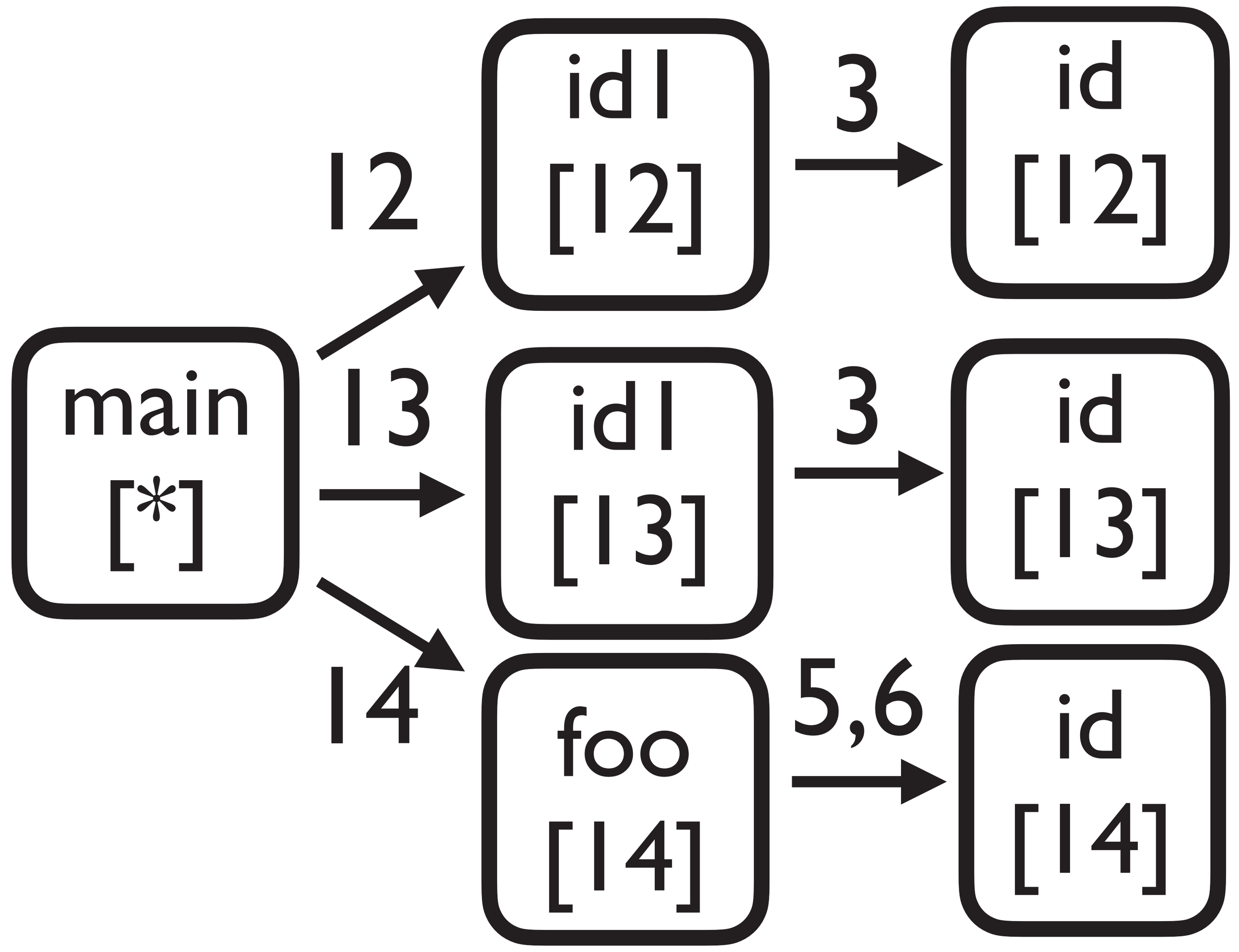
- Suppose given call-g … d infer what T’ is

Exactly the same analyses



1objH+T (T = ∅)

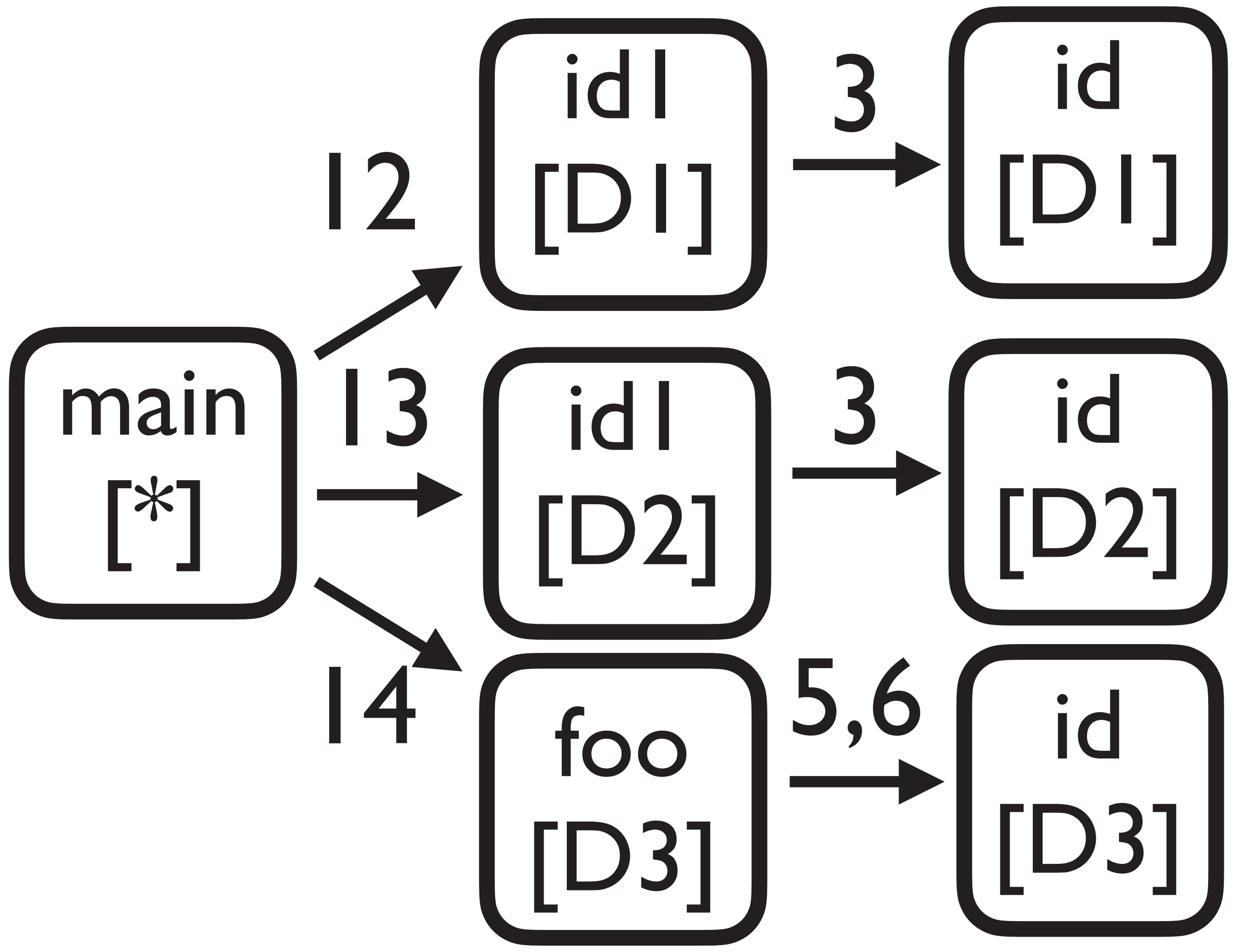
=



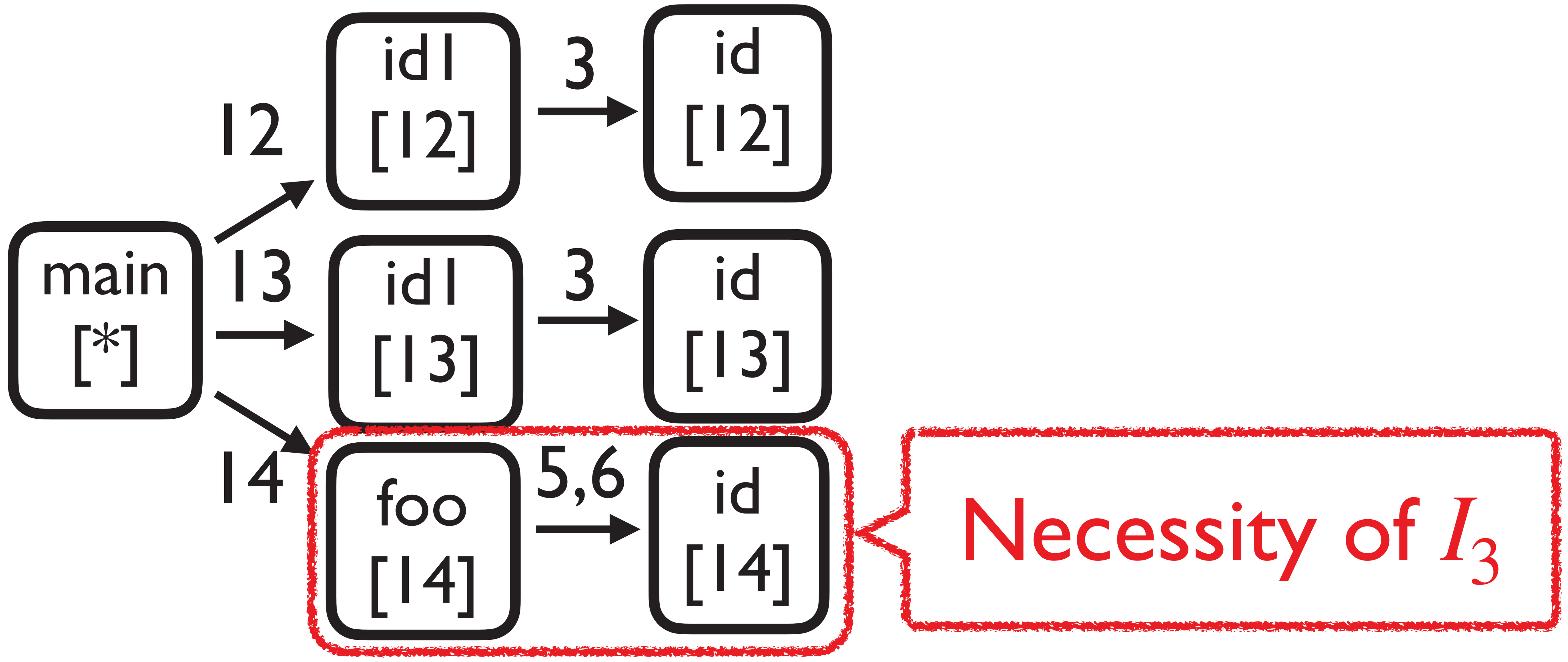
1callH+T’ (T’ = {3,5,6})

Intuition Behind Simulation $(I_1 \cup I_2)$

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



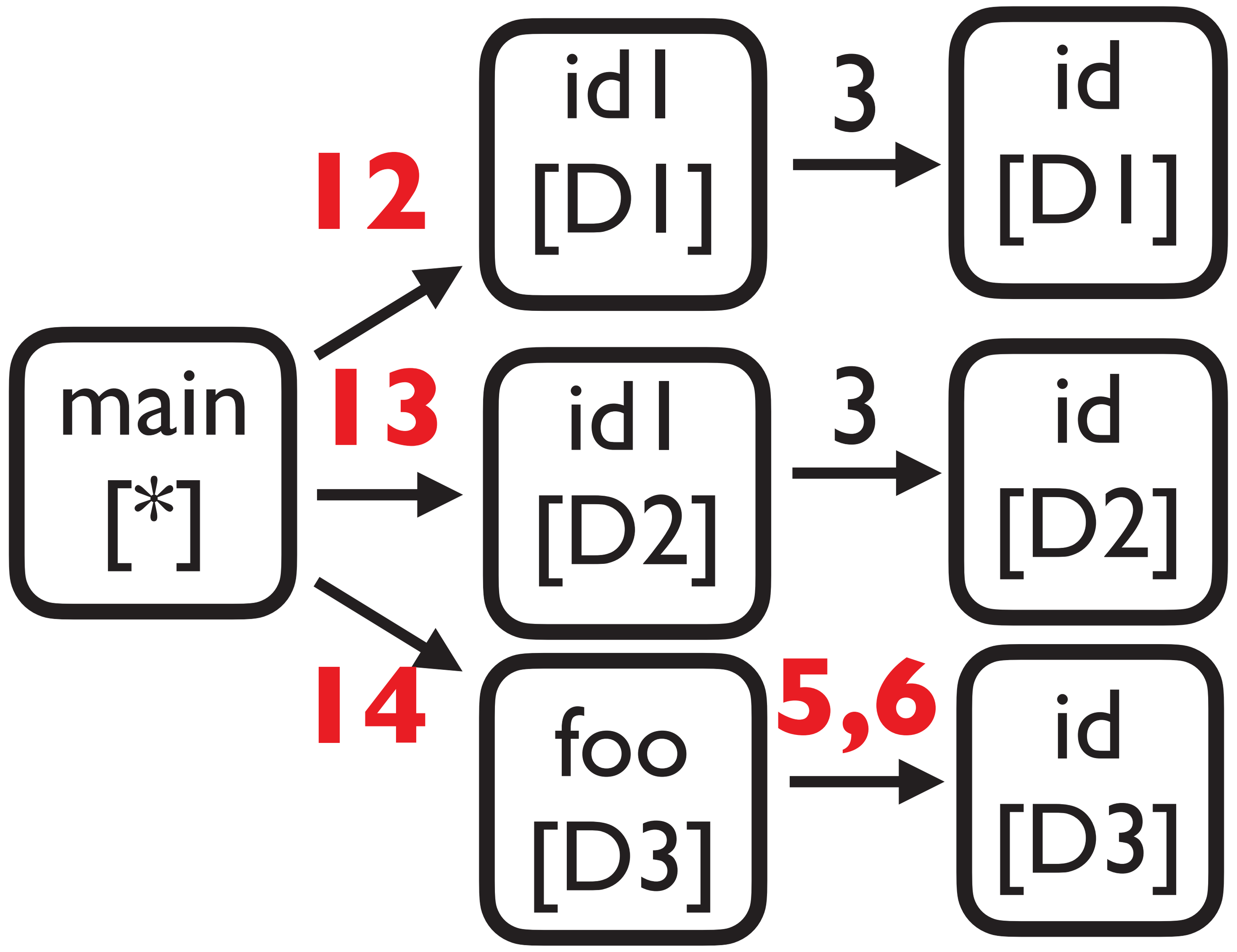
1objH+T (T = ∅)



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

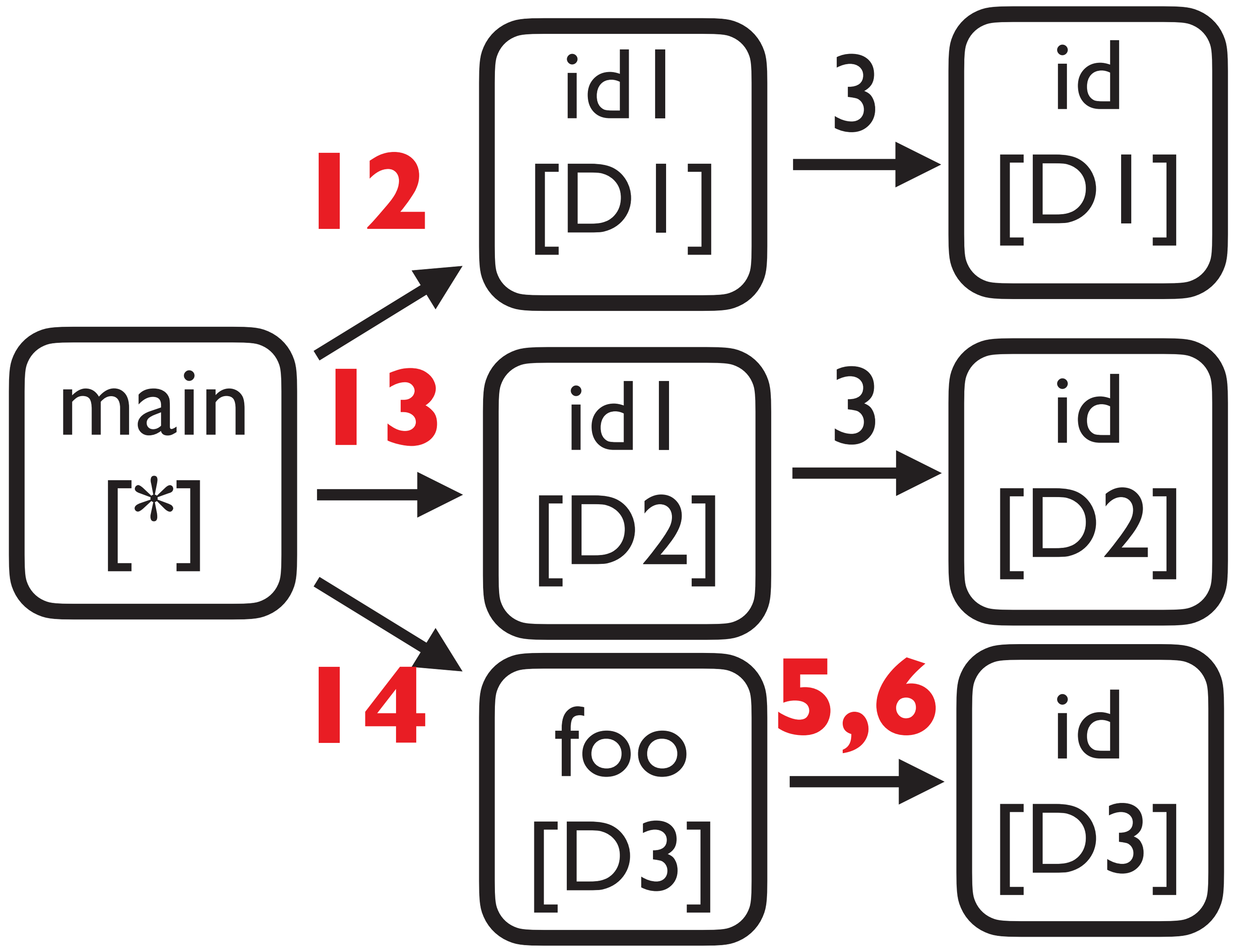
I_2={3}

- 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}

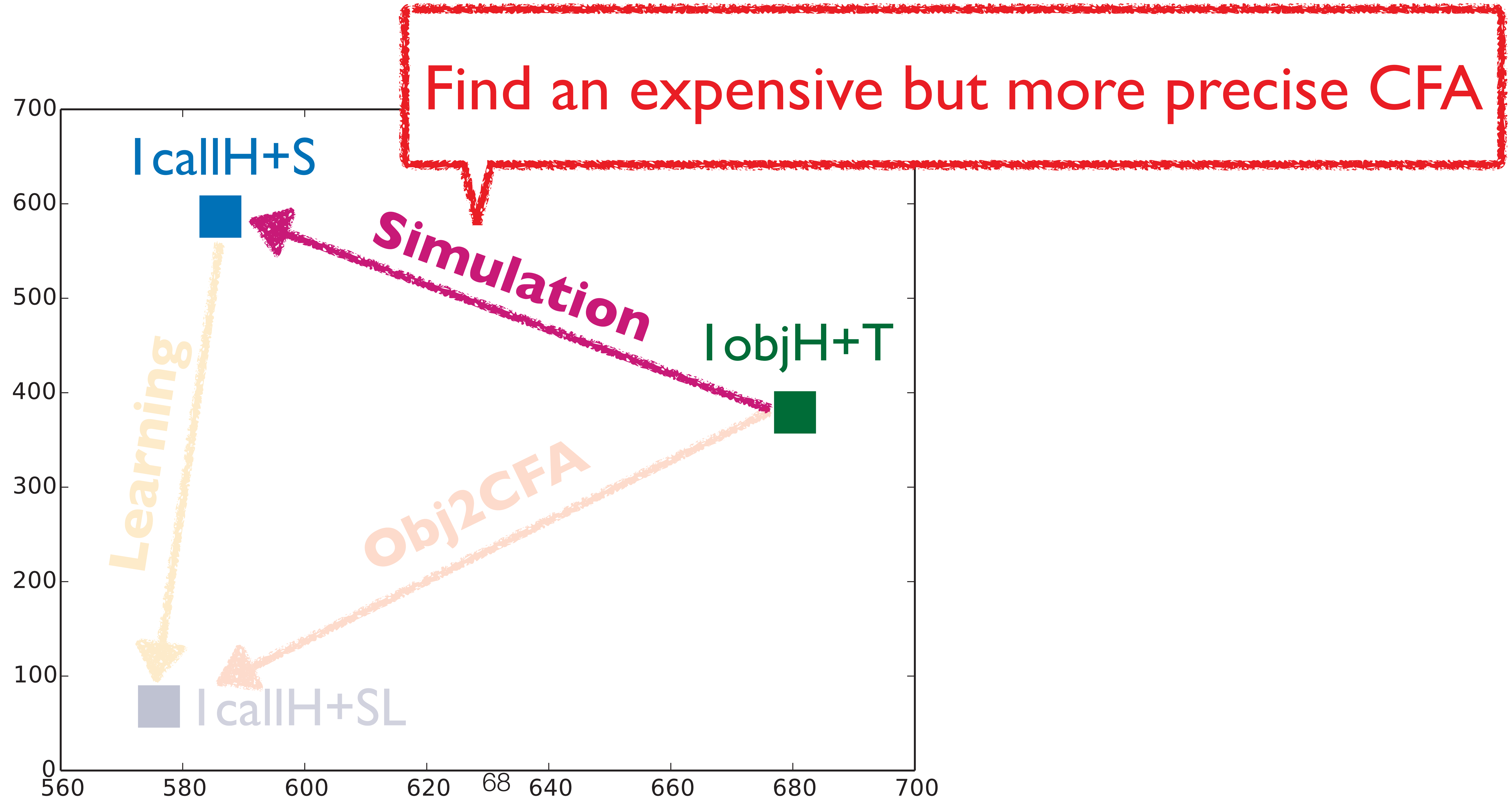
T'= $(I_1 \cup I_2) \backslash I_3$ = {3}

Technique 1: Simulation

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

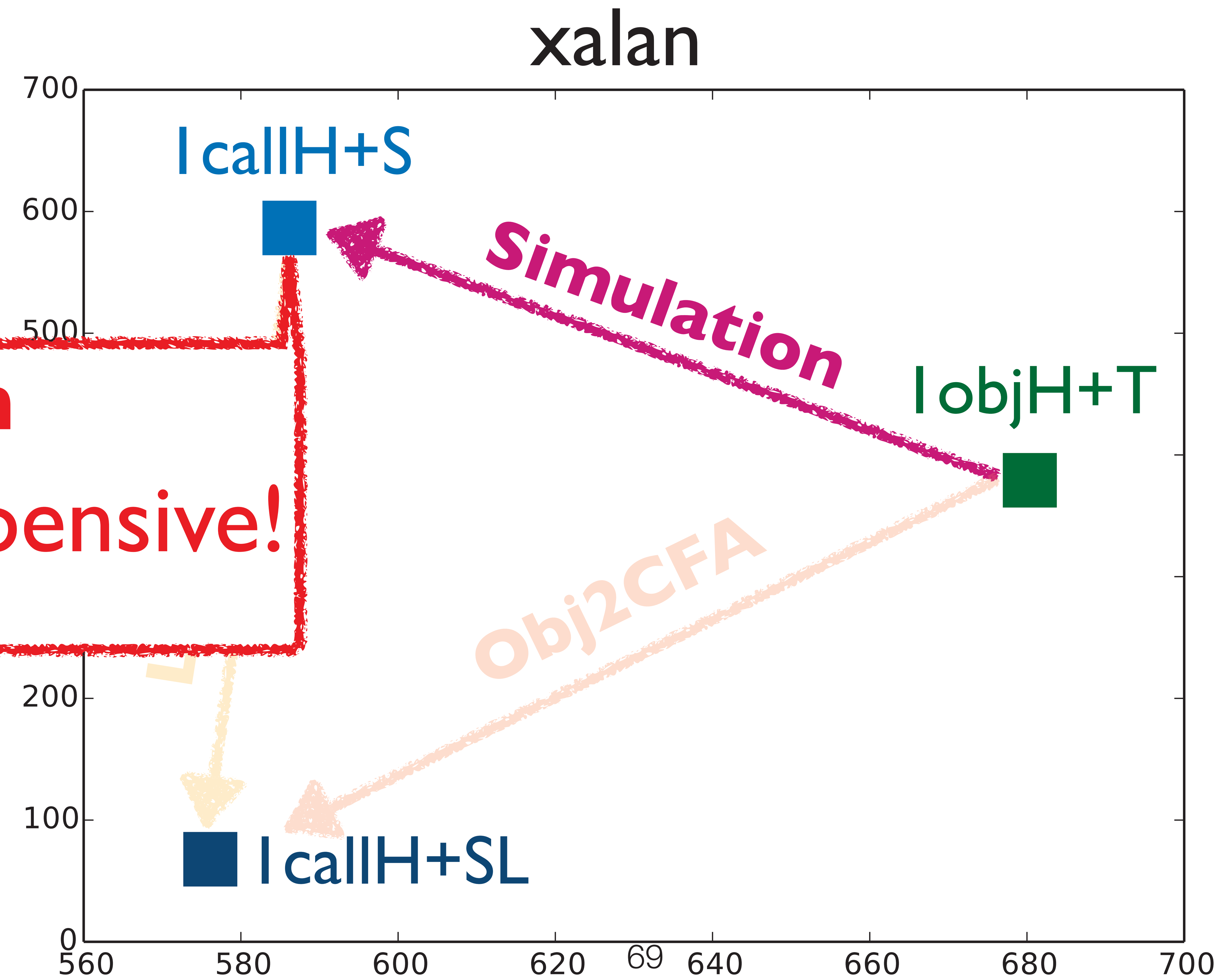
Our Technique : **Obj2CFA**

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



Our Technique : Obj2CFA

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

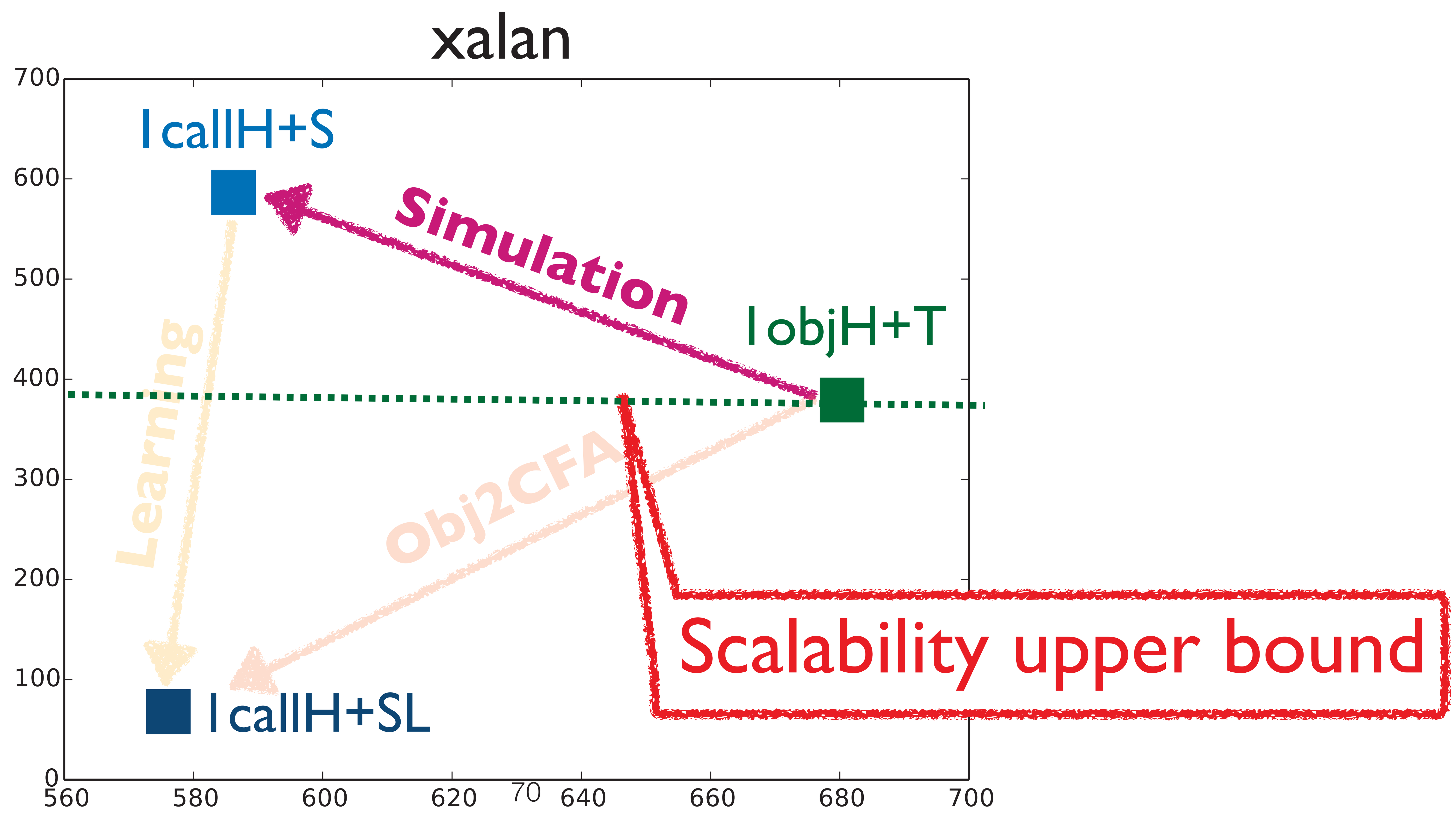


Limitation

Simulation is expensive!

Our Technique : **Obj2CFA**

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

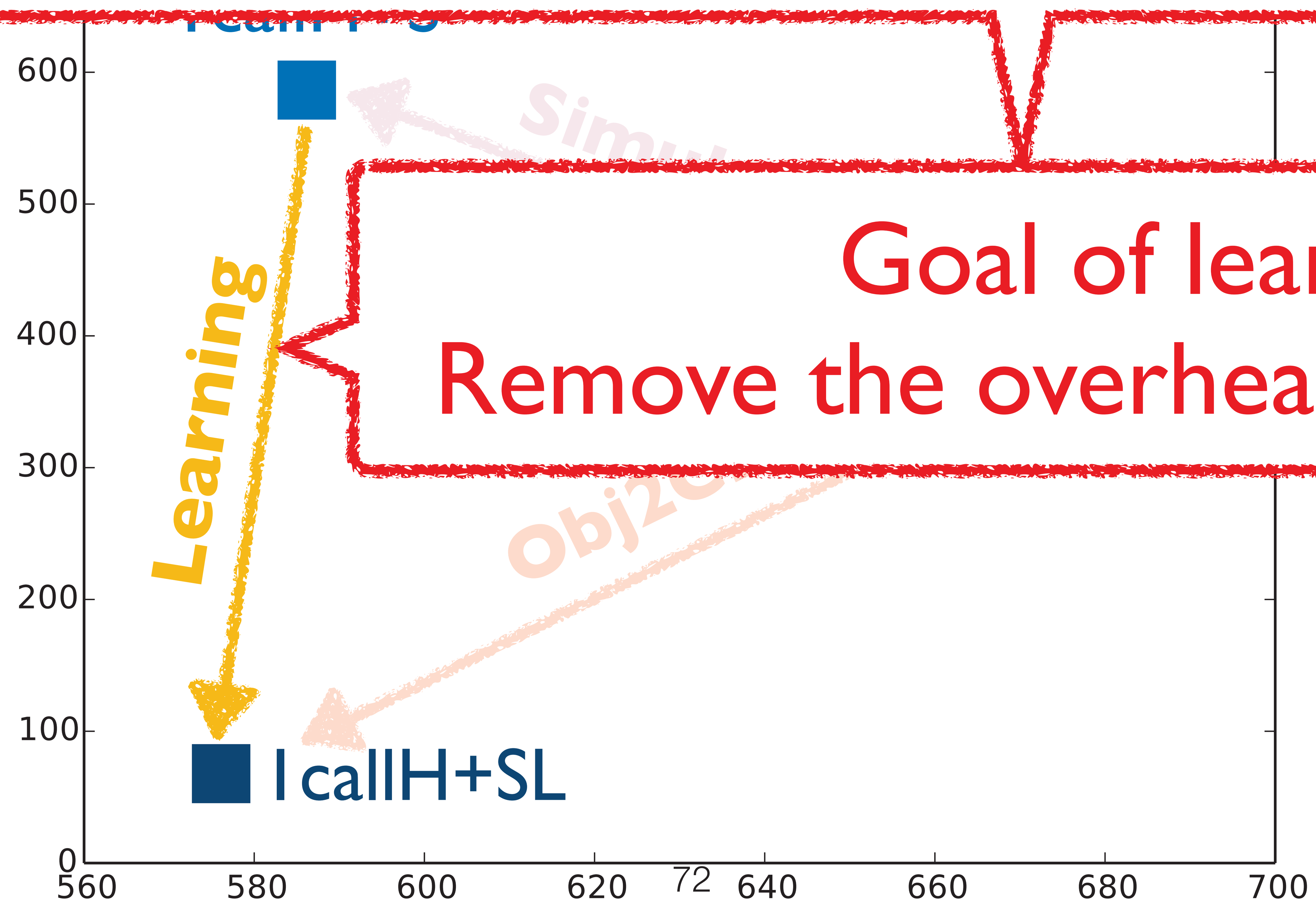


Our Technique : **Obj2CFA**

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

Our Technique : Obj2CFA

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

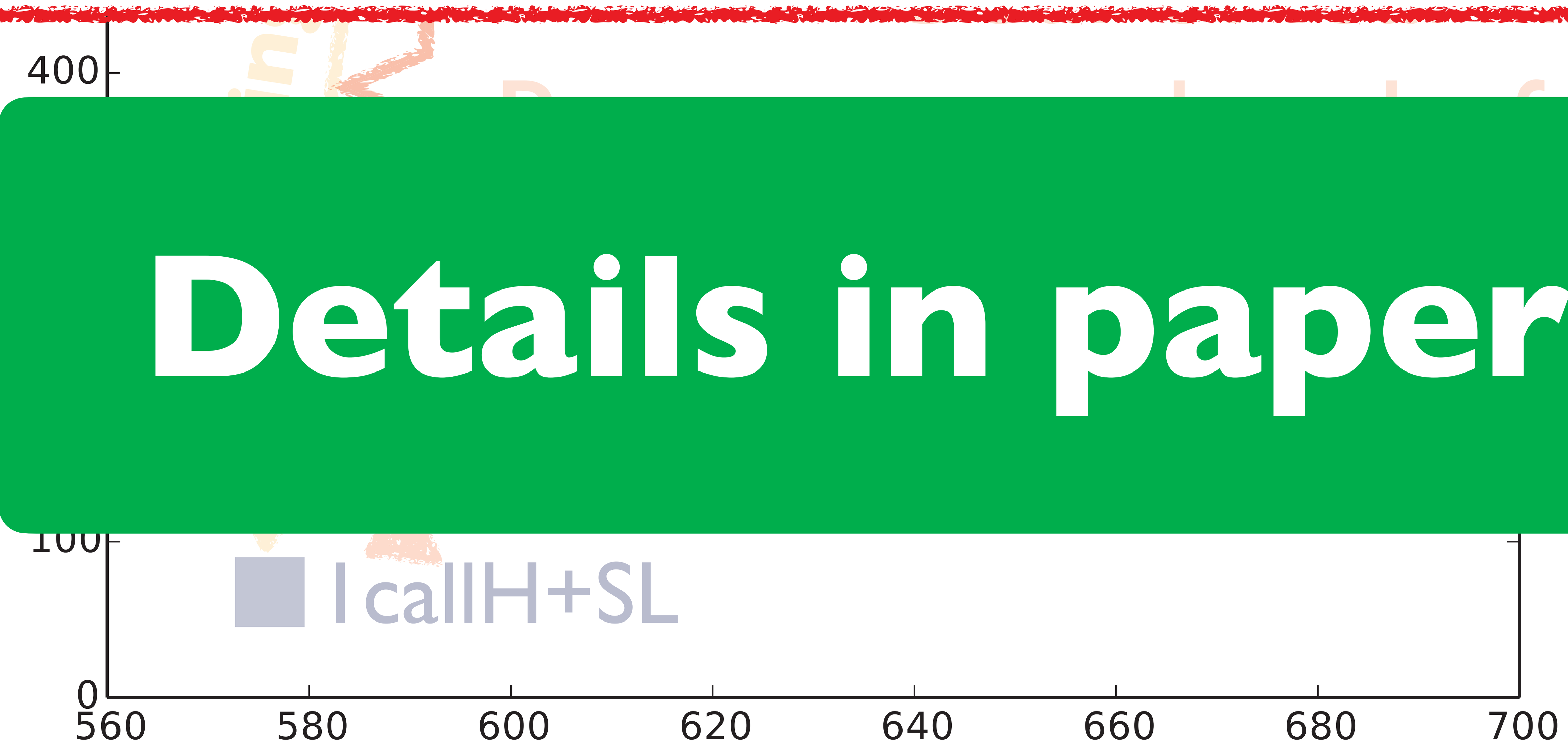


Goal of learning:
Remove the overhead of simulation

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?

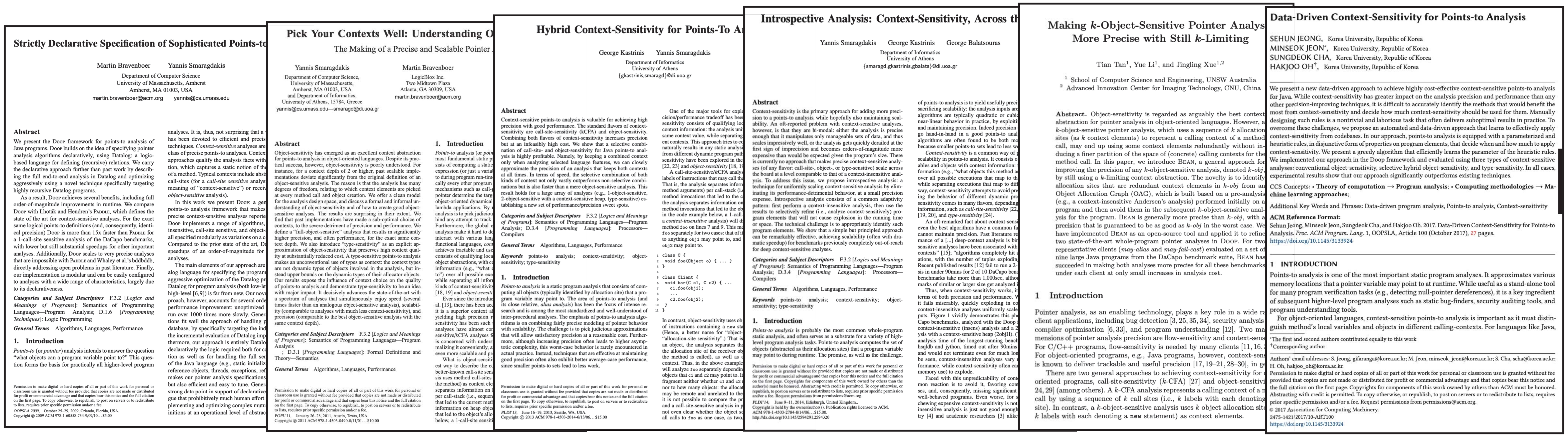
Call-site sensitivity vs Object sensitivity

Context tunneling is included

Setting

- Doop

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



2009 (OOPSLA)	2011 (POPL)	2013 (PLDI)	2014 (PLDI)	2016 (SAS)	2017 (OOPSLA)	…

Setting

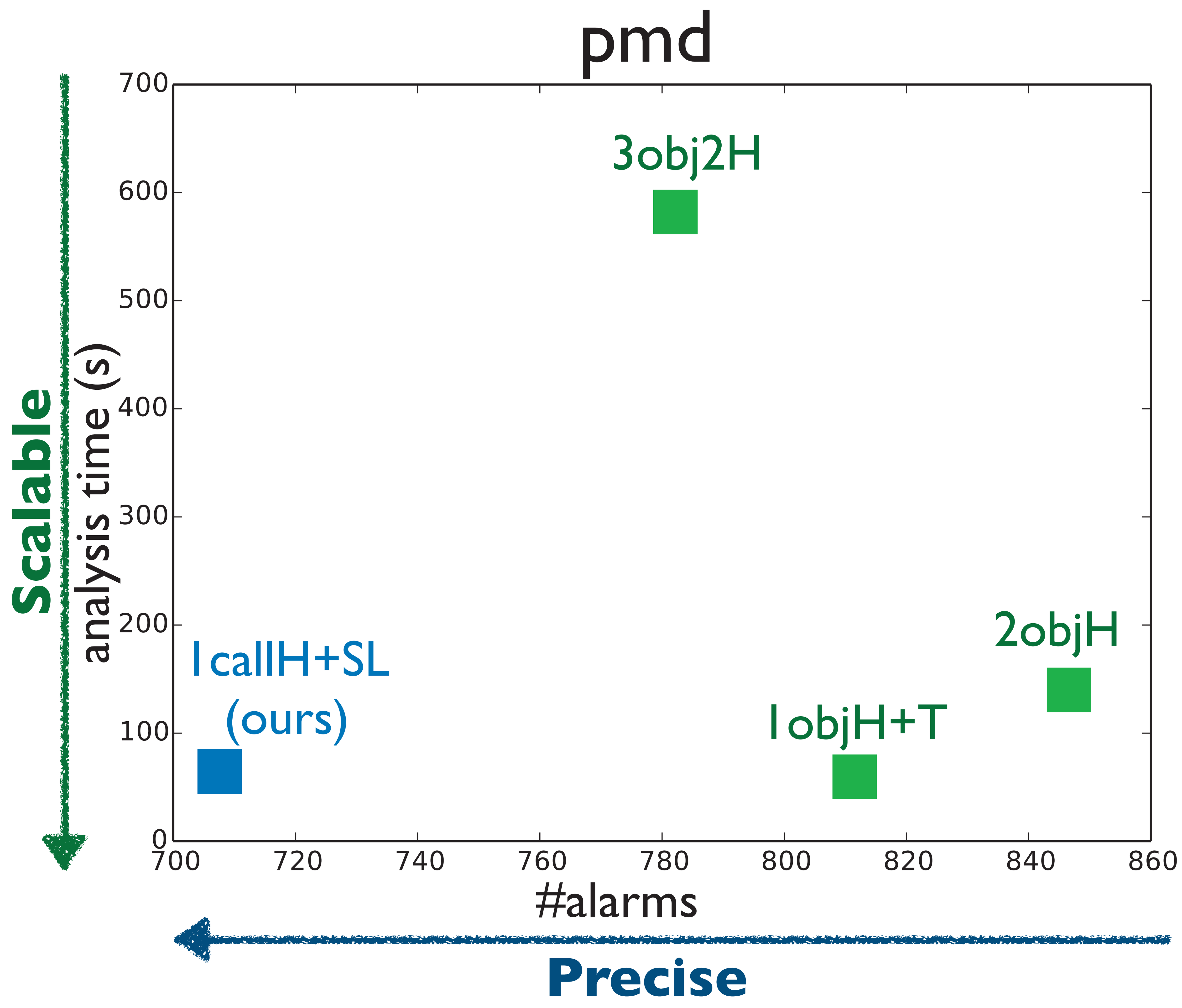
- Doop
    - Pointer analysis framework for Java
- Research Question: which one is better?

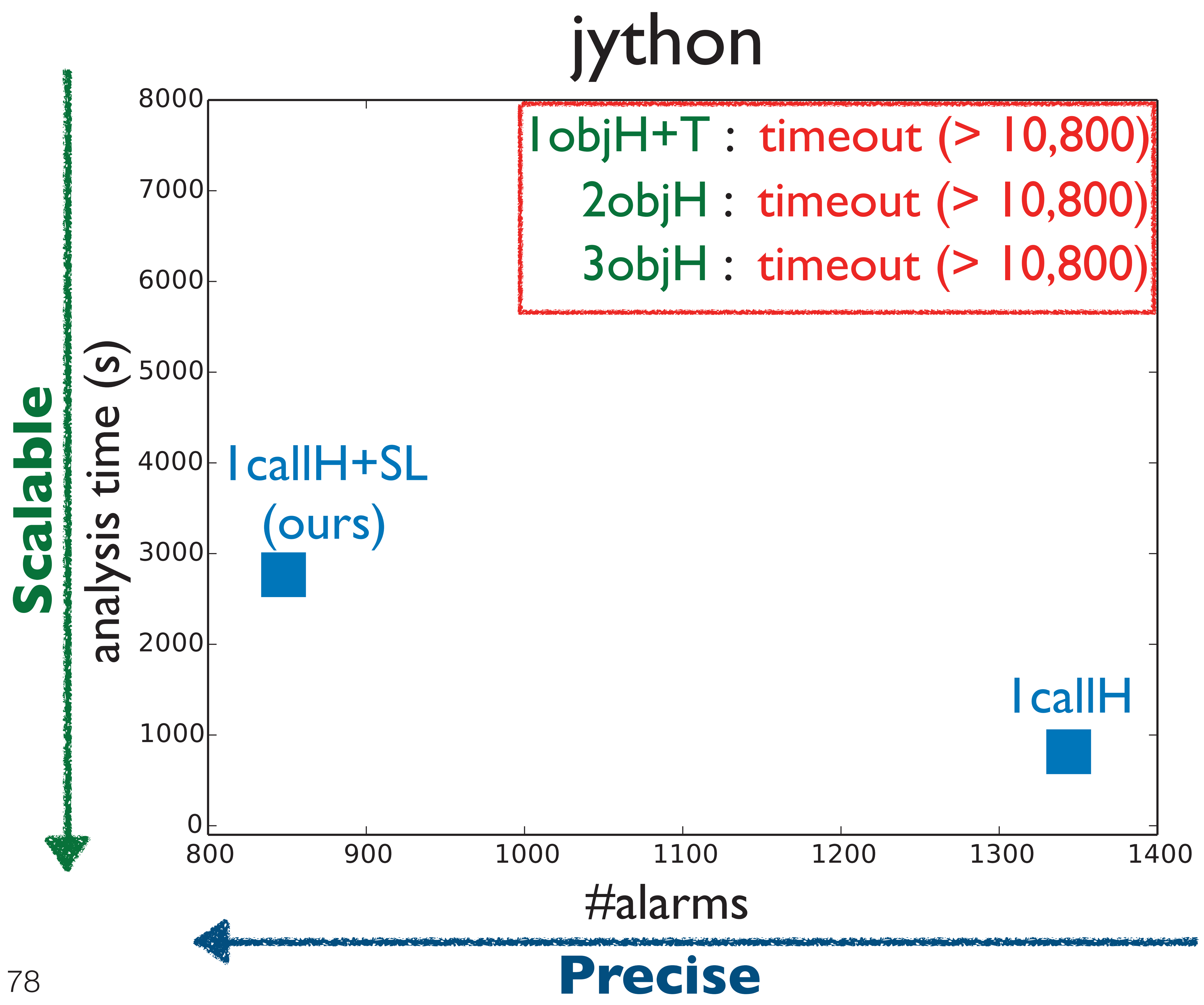
Call-site sensitivity vs **Object sensitivity**

Context tunneling is included

Call-site Sensitivity vs Object Sensitivity

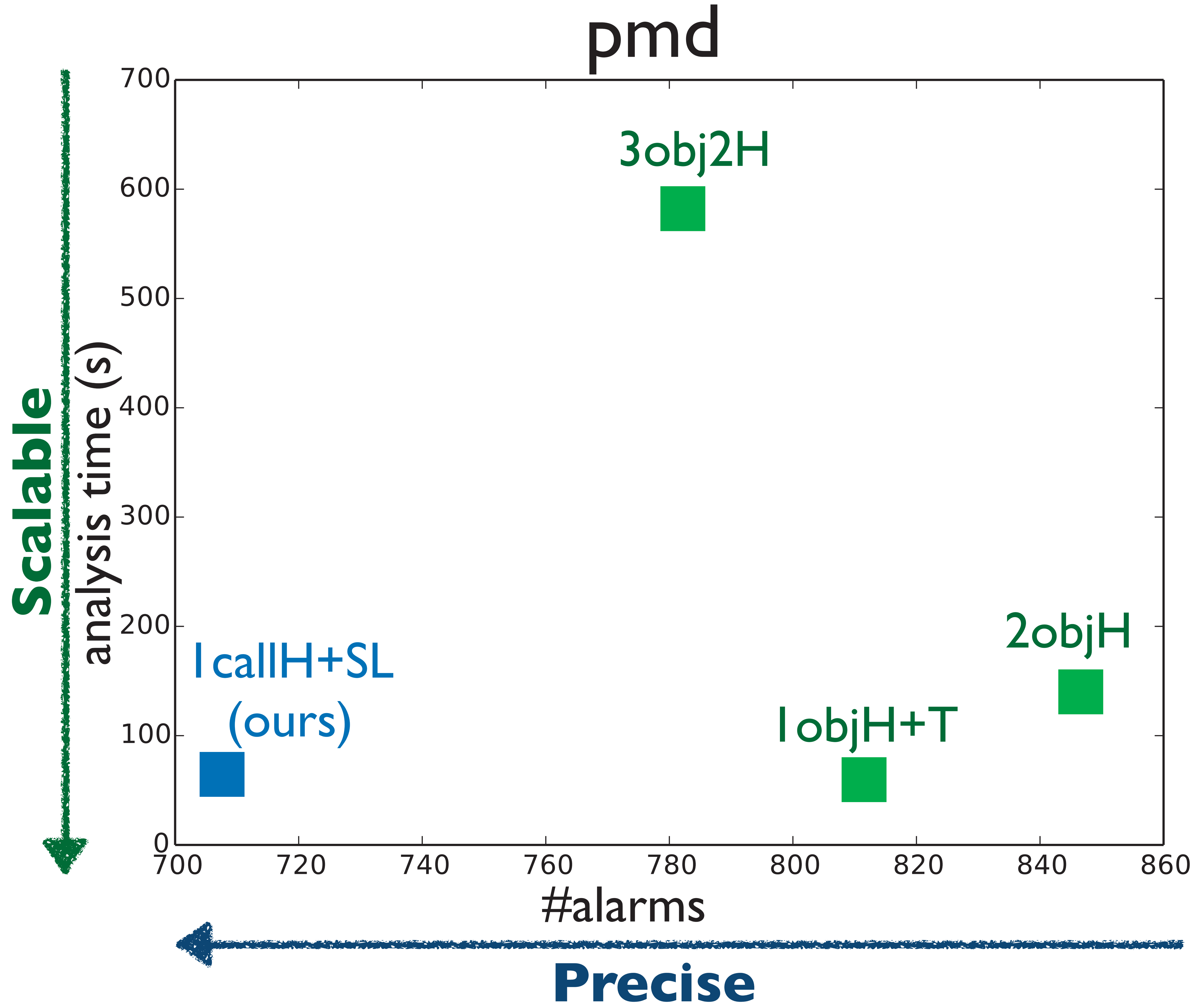
- 1callH+SL (ours) is more precise and scalable than the existing object sensitivities

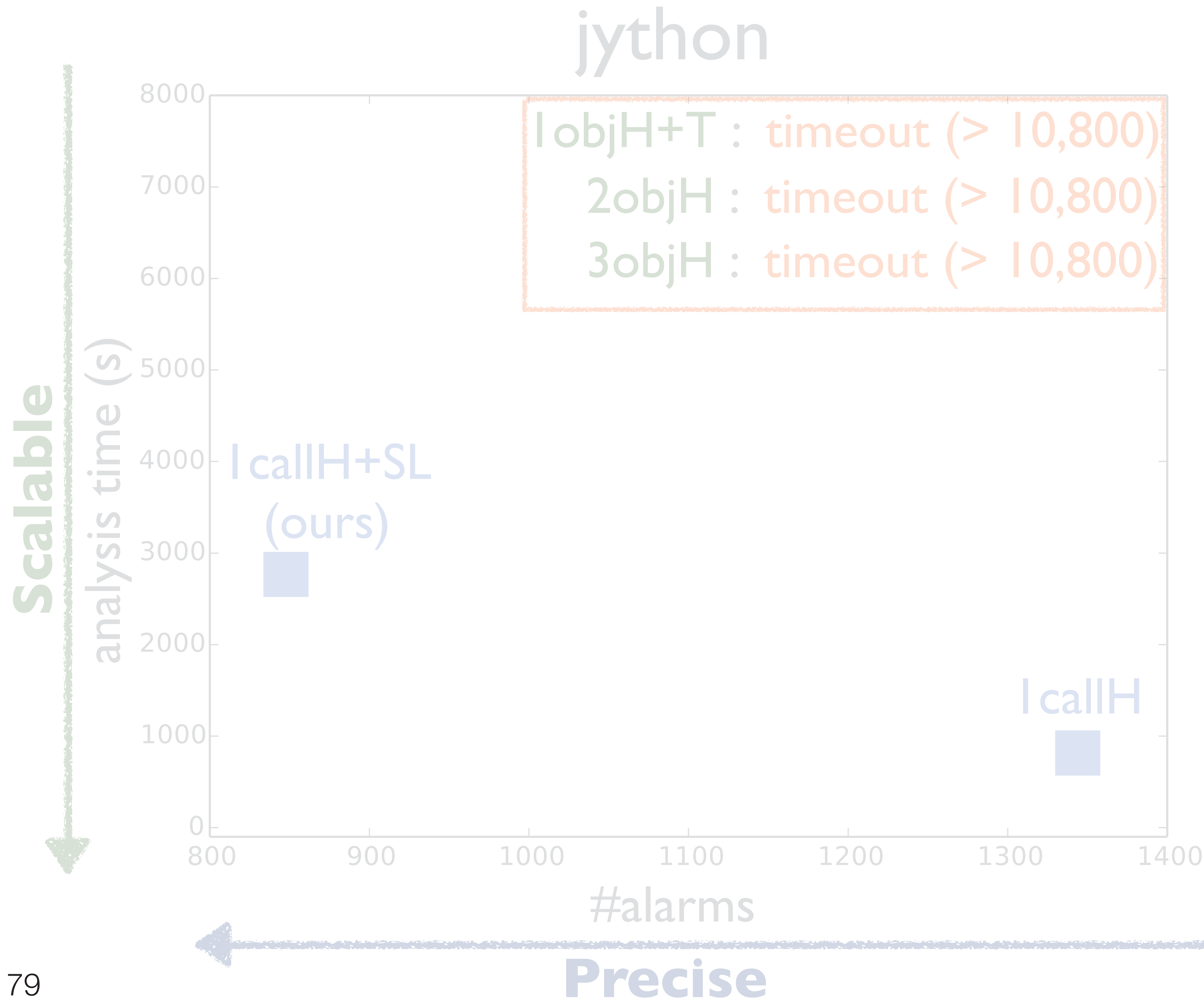




Call-site Sensitivity vs Object Sensitivity

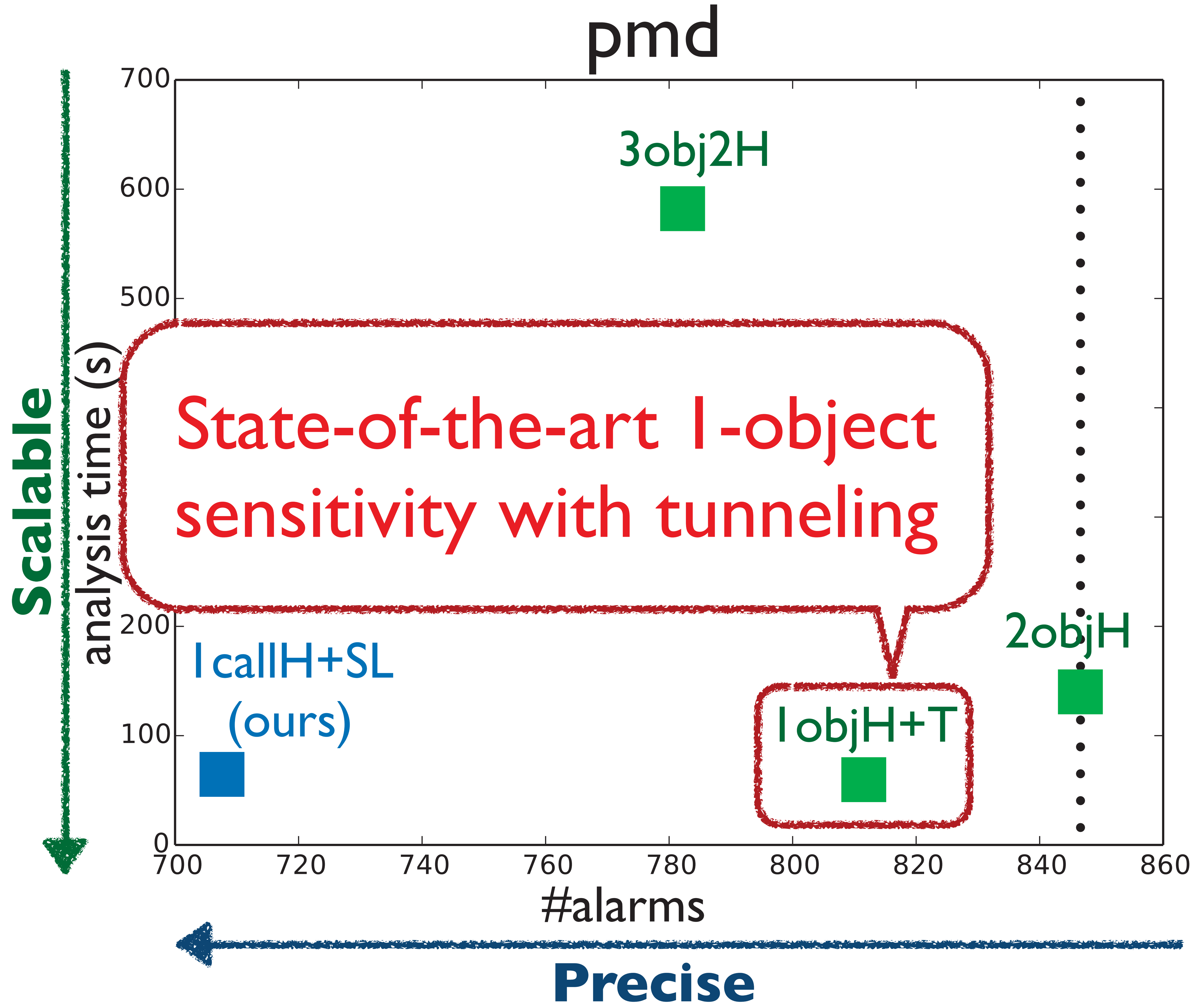
- 1callH+SL (ours) is more precise and scalable than the existing object sensitivities

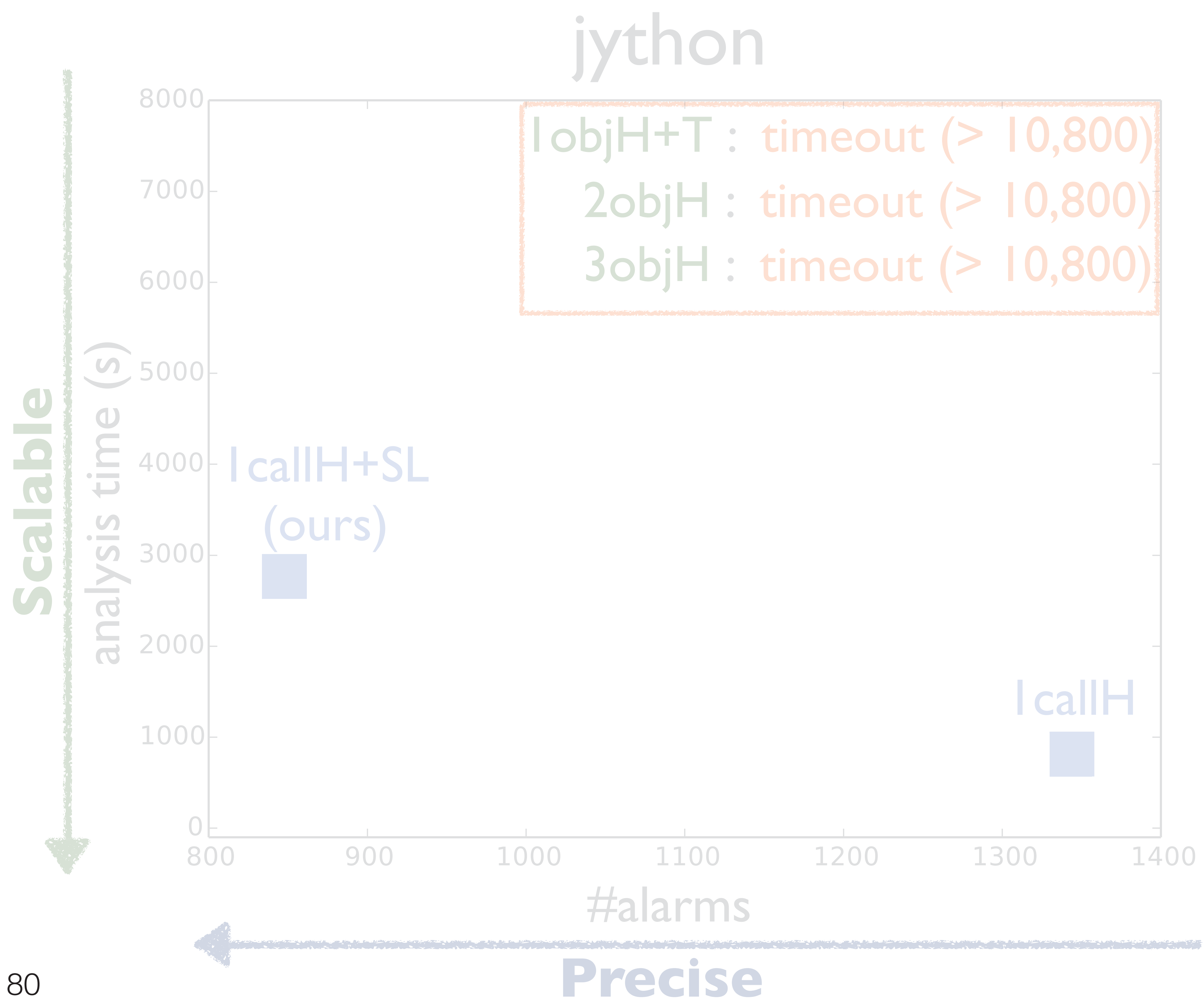




Call-site Sensitivity vs Object Sensitivity

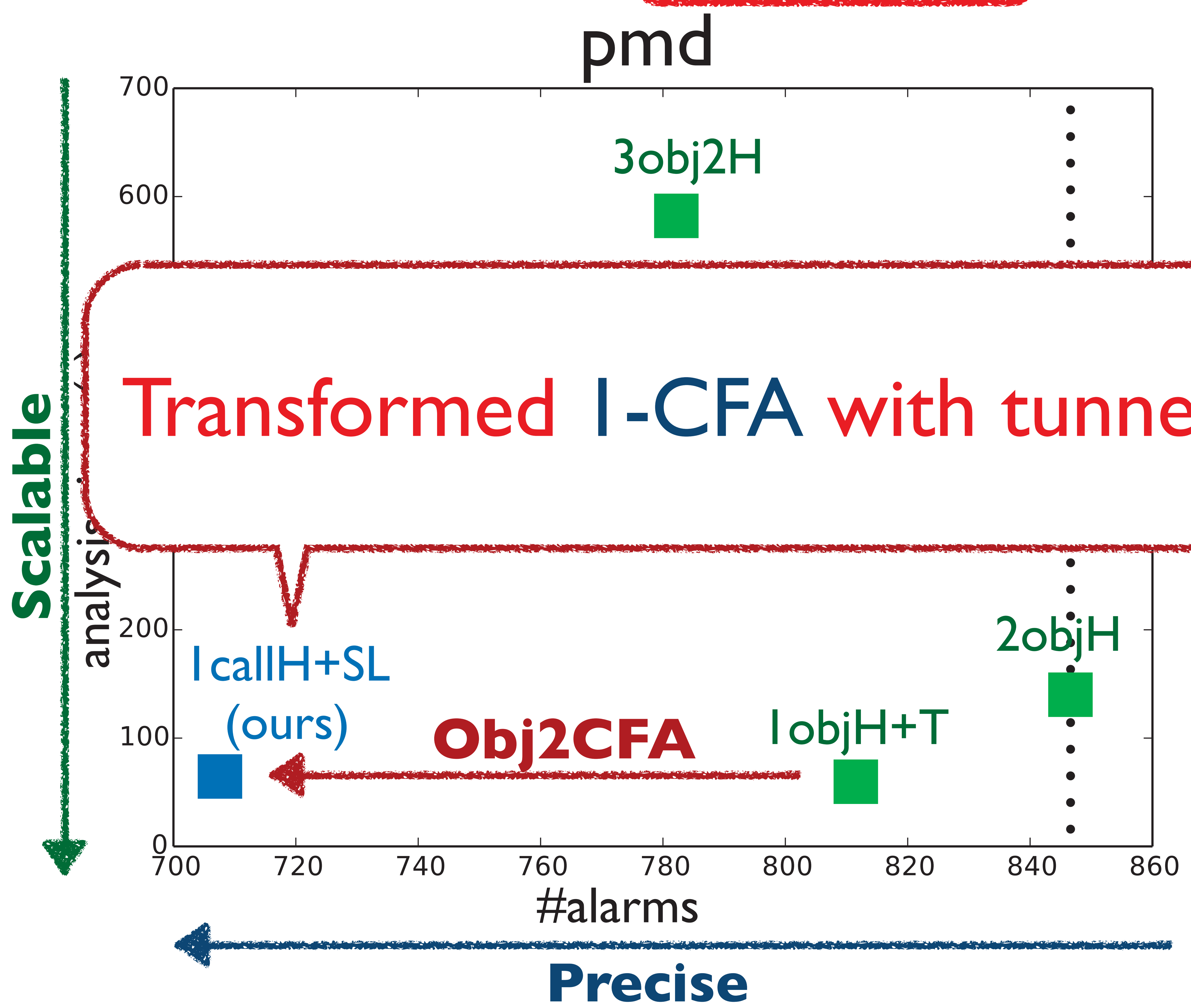
- 1callH+SL (ours) is more precise and scalable than the existing object sensitivities

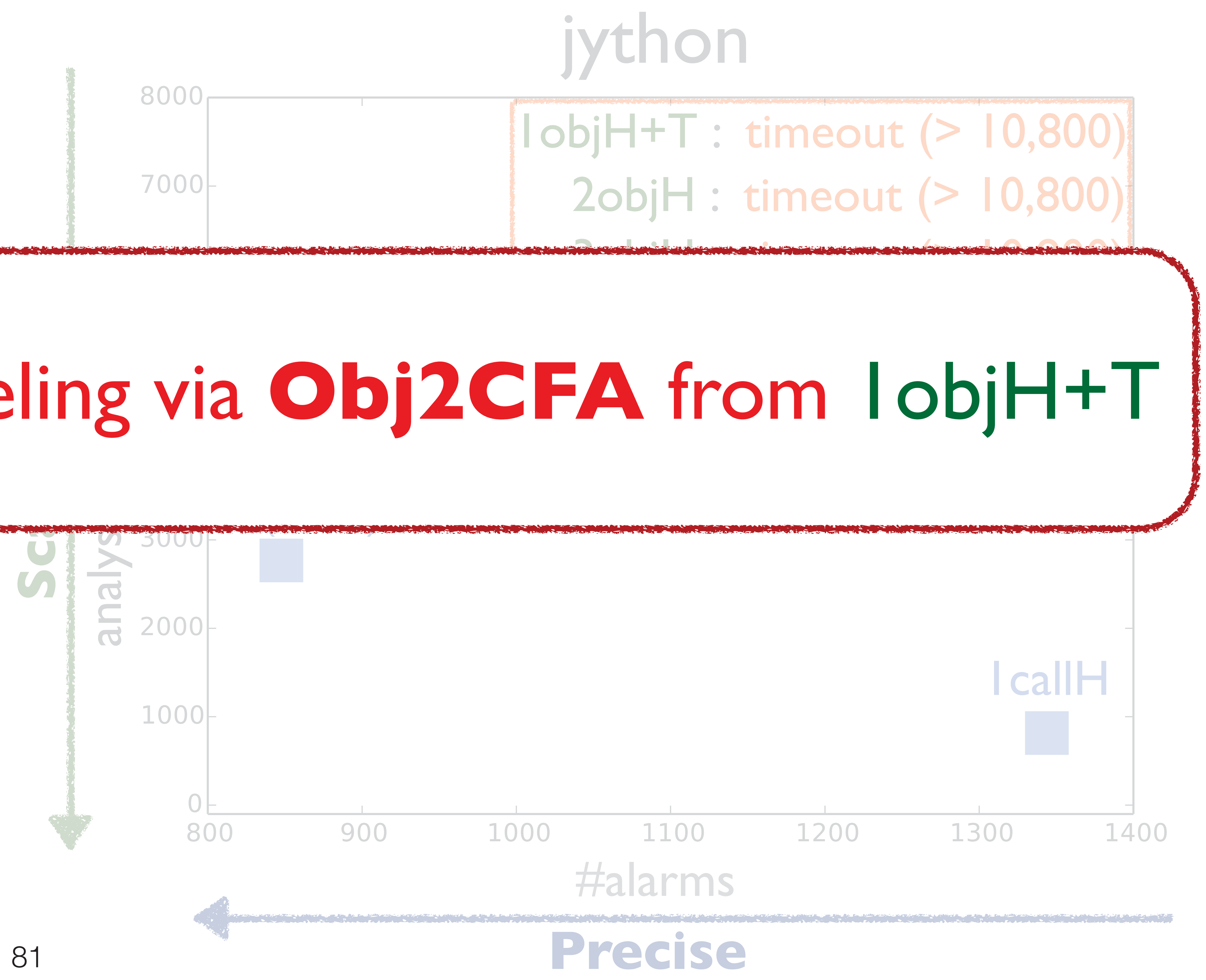




Call-site Sensitivity vs Object Sensitivity

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



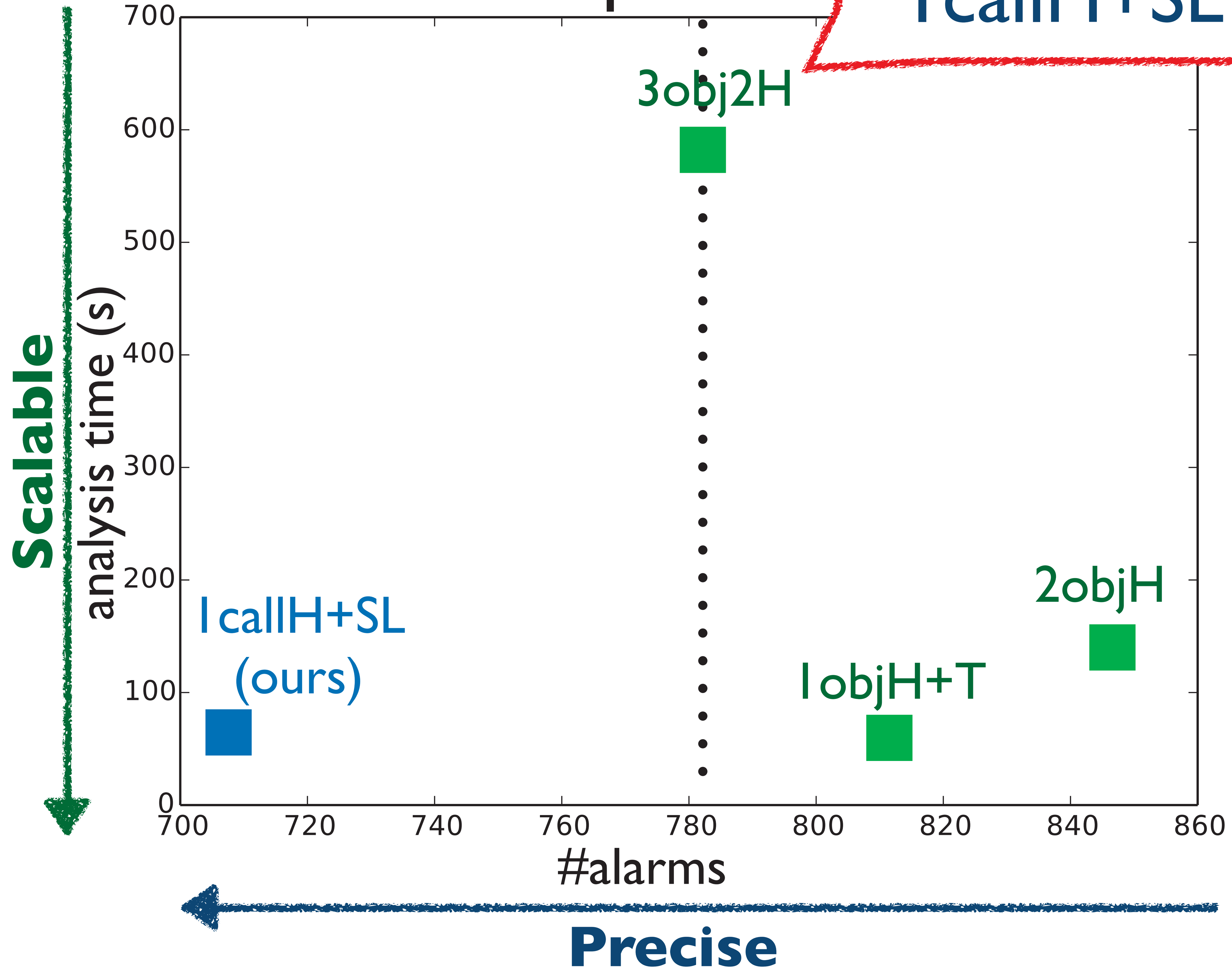


Call-site Sensitivity vs Object Sensitivity

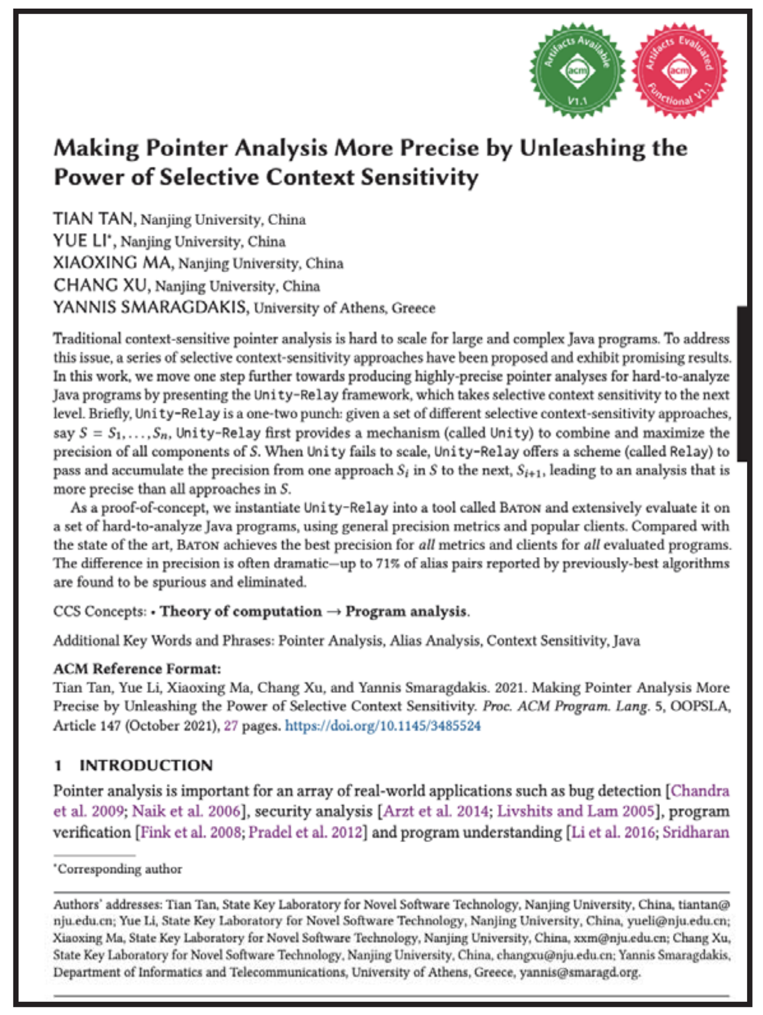
- 1callH+SL (ours) is more precise and scalable than the existing object sensitivities

1callH+SL is even more precise than 3obj2H

pmd



Precision upper bound of recent researches on object sensitivity



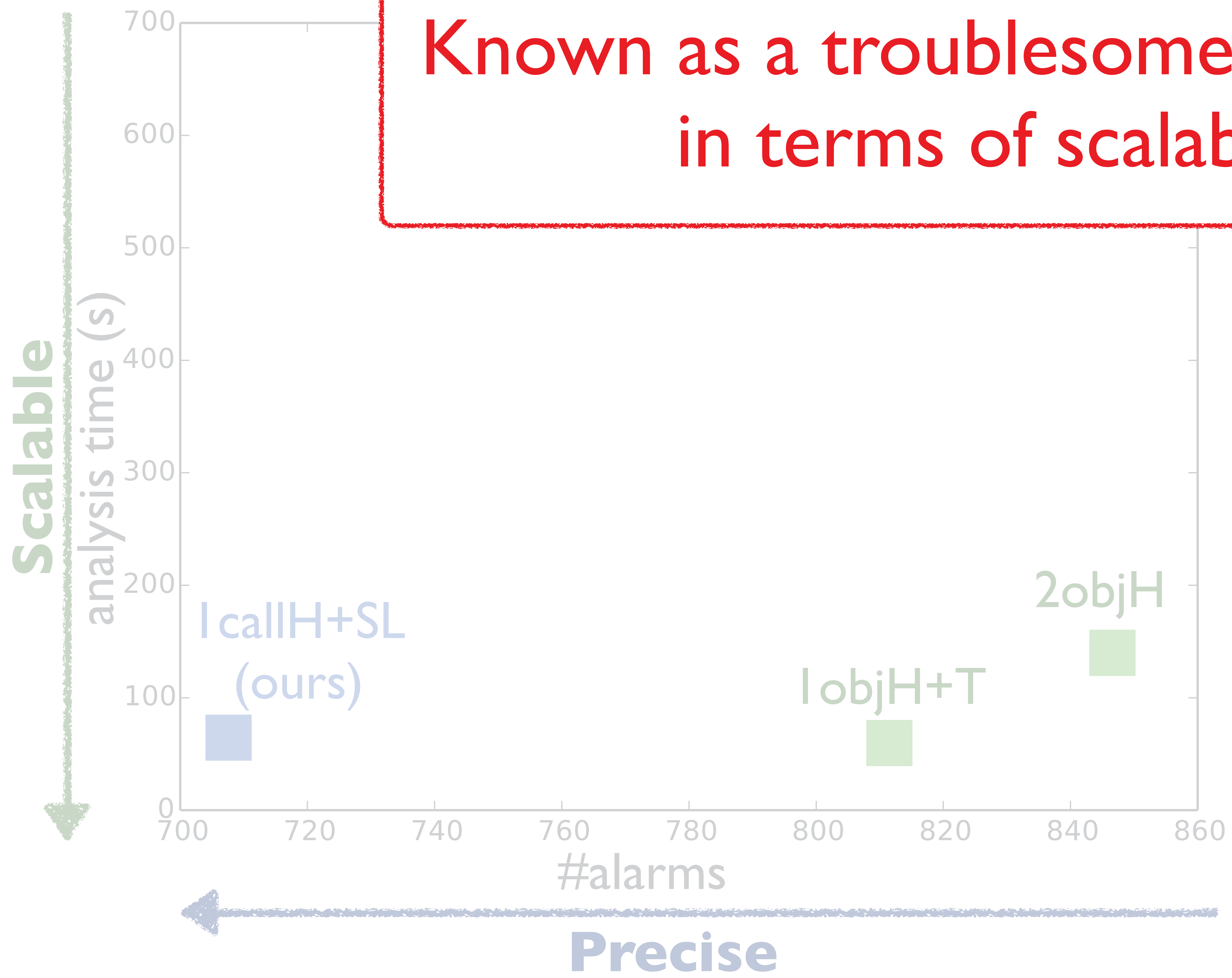
OOPLSA 2021

OOPLSA 2019

Call-site Sensitivity vs Object Sensitivity

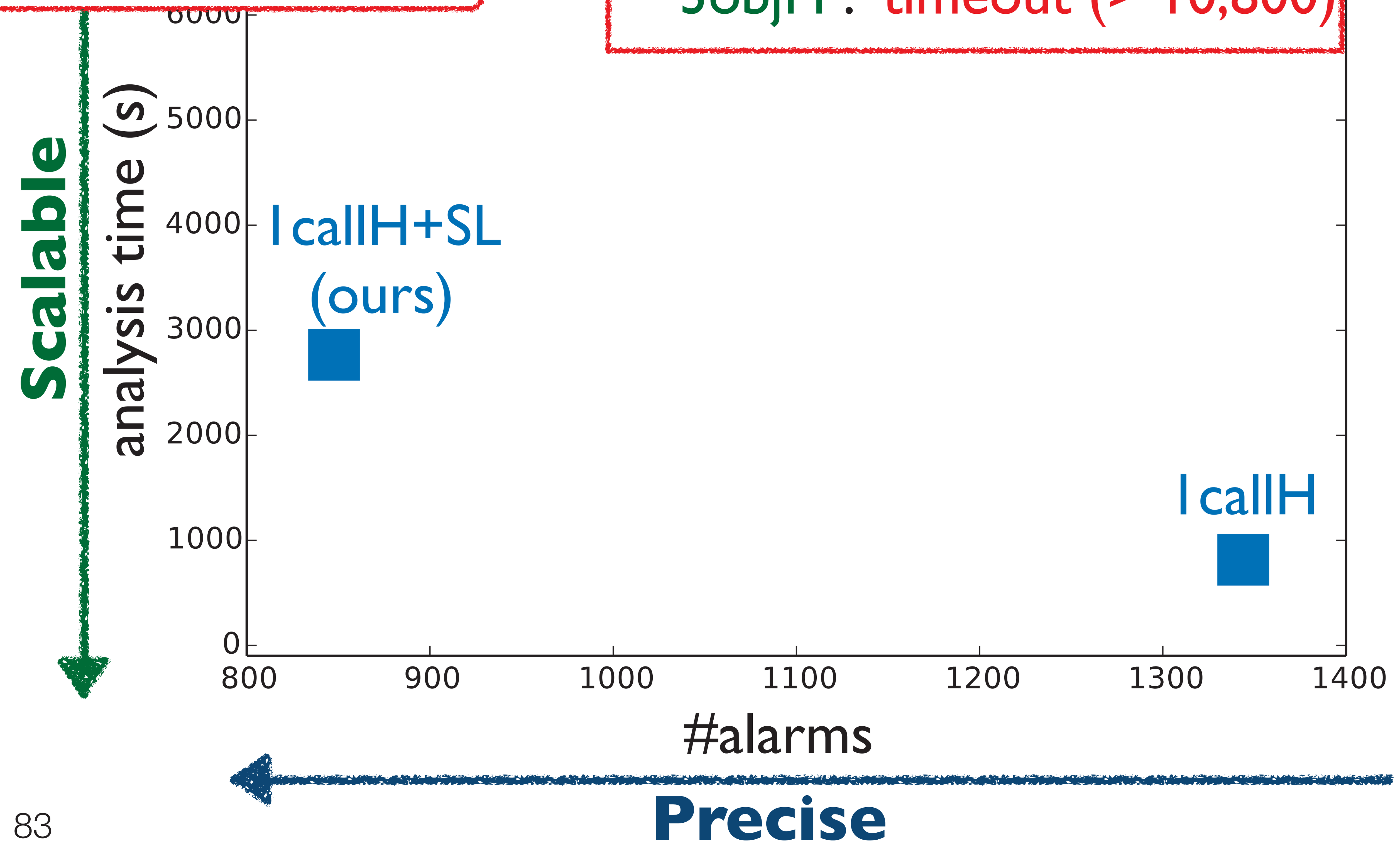
- 1callH+SL (ours) is more precise and scalable than the existing object sensitivities

Known as a troublesome benchmark in terms of scalability



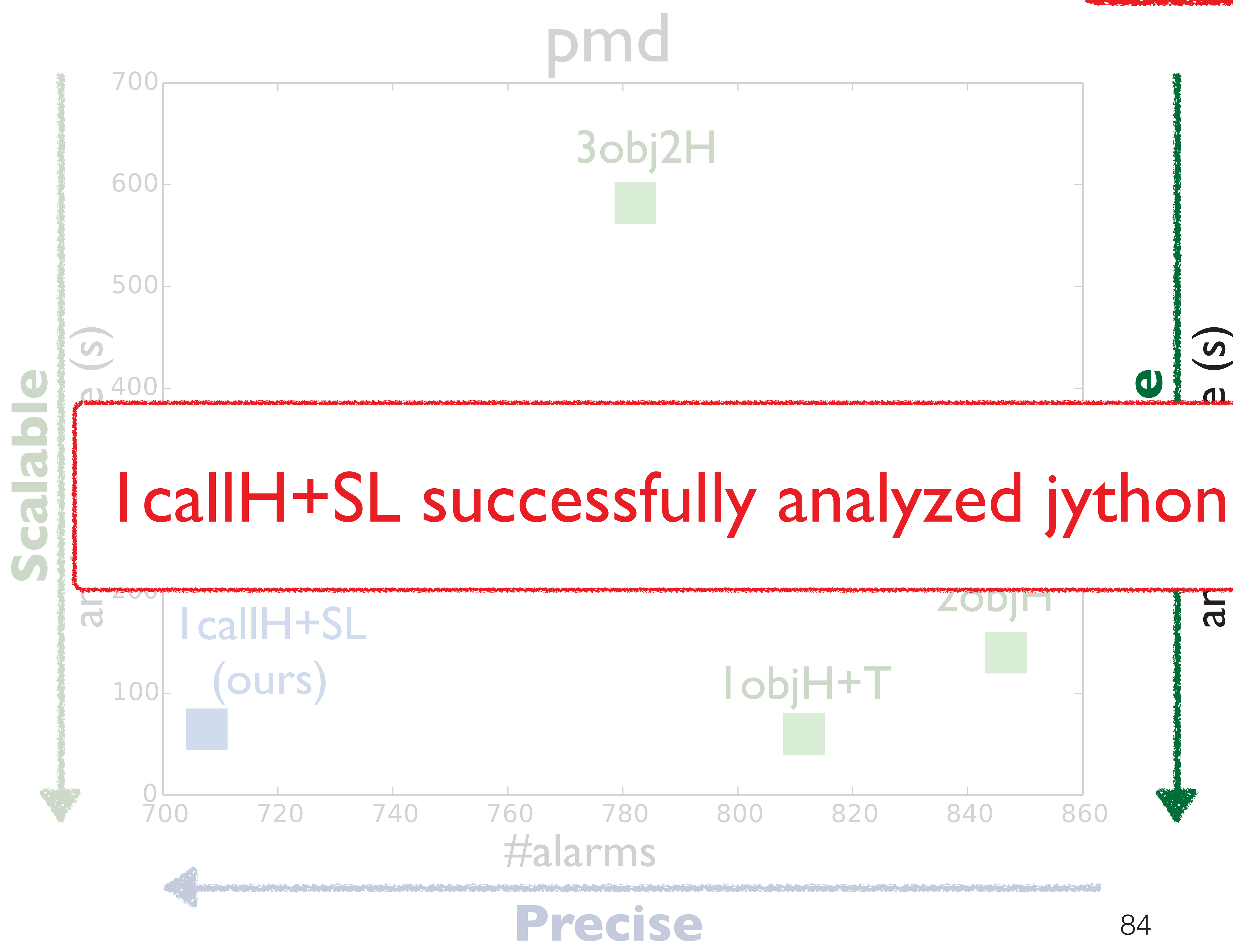
jython

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

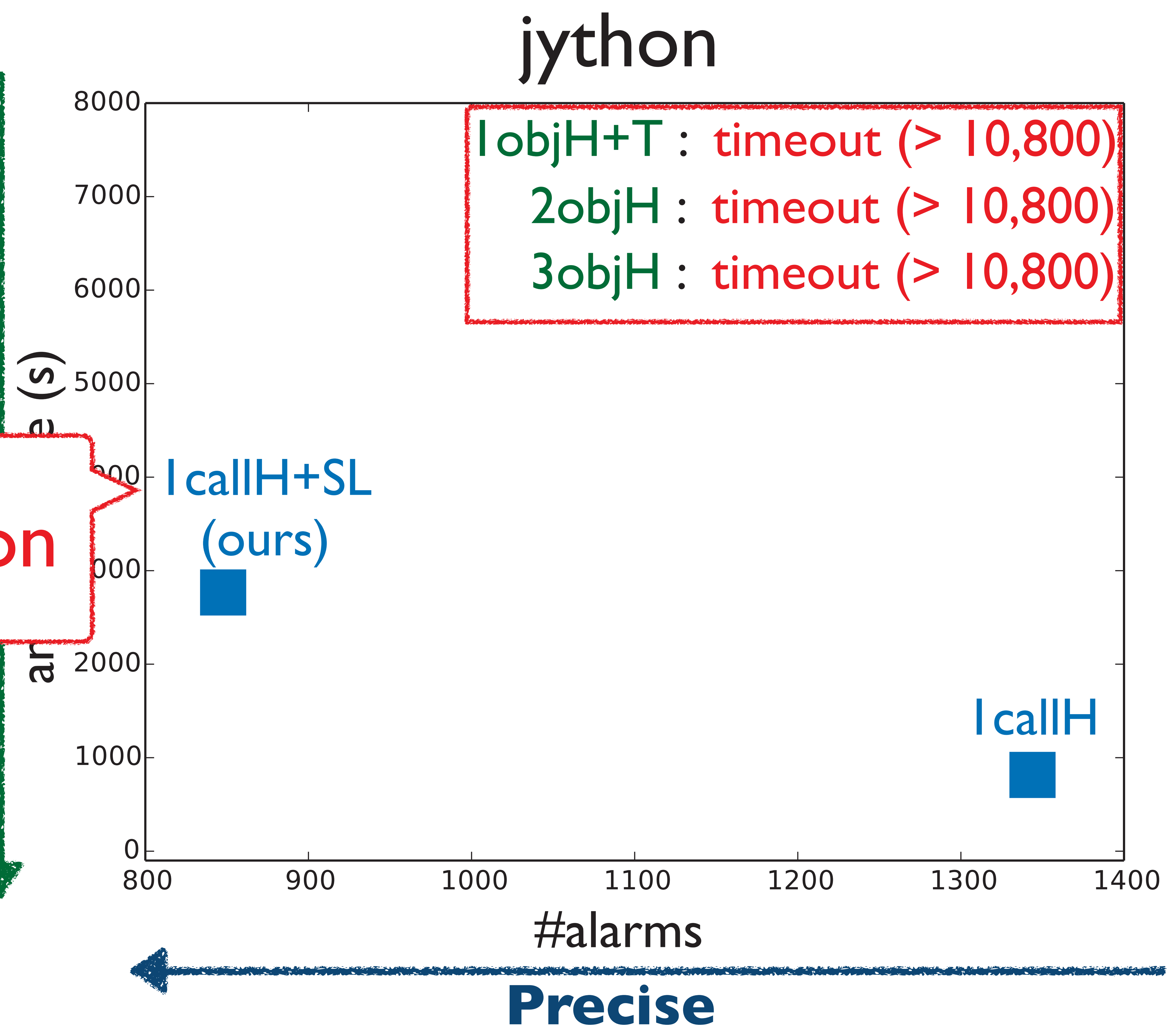


Call-site Sensitivity vs Object Sensitivity

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



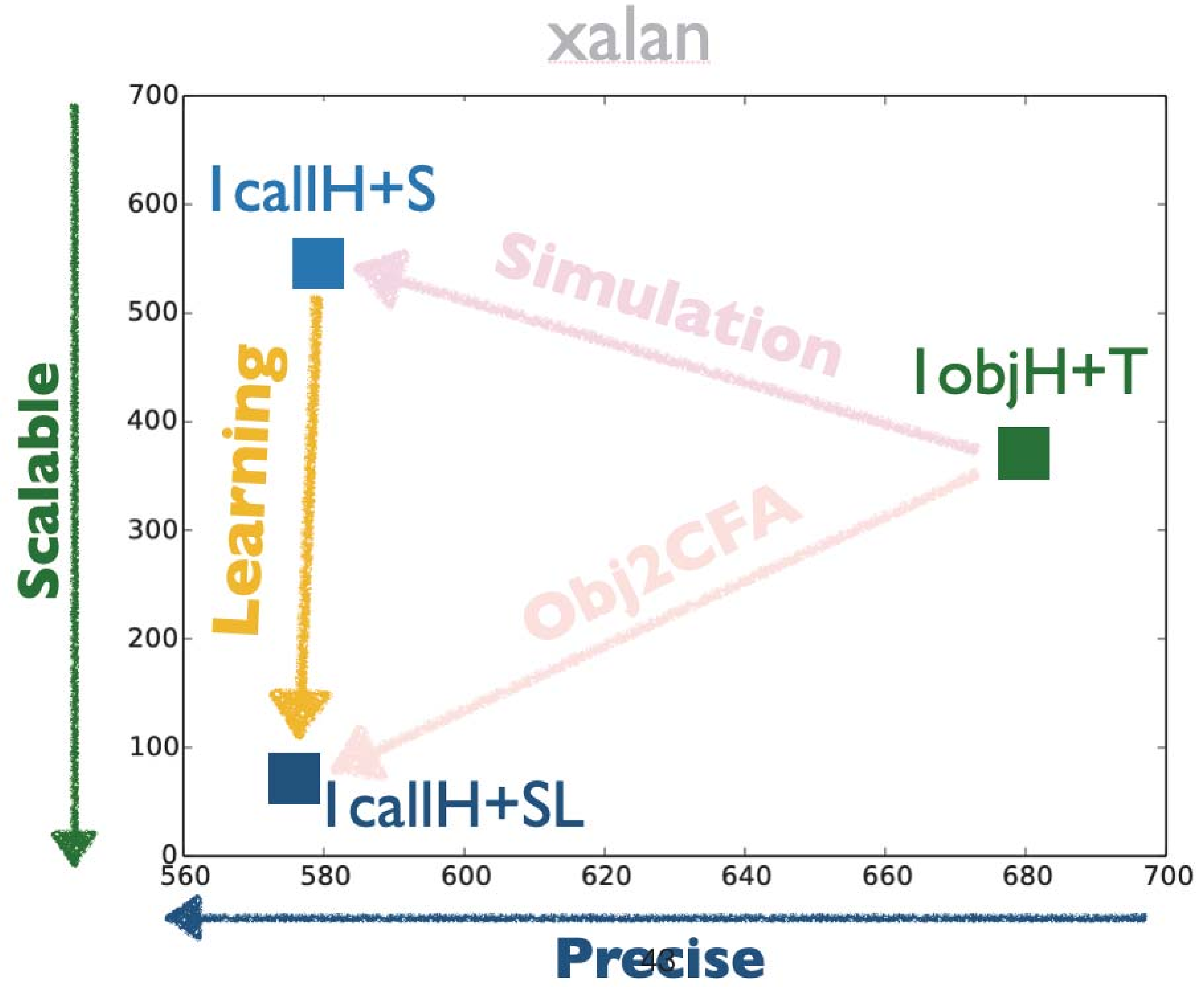
1callH+SL successfully analyzed jython

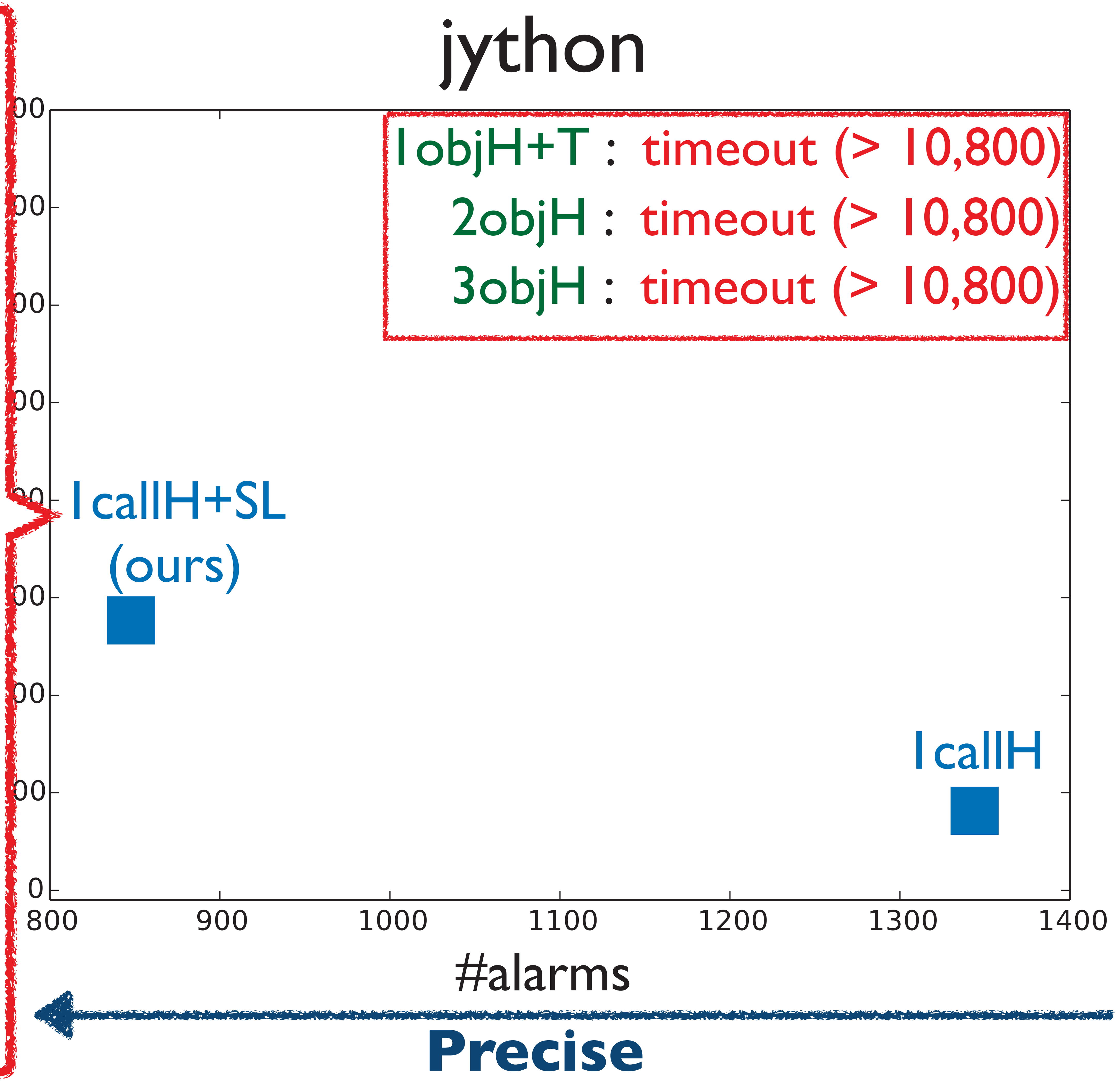


Call-site Sensitivity vs Object Sensitivity

- 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

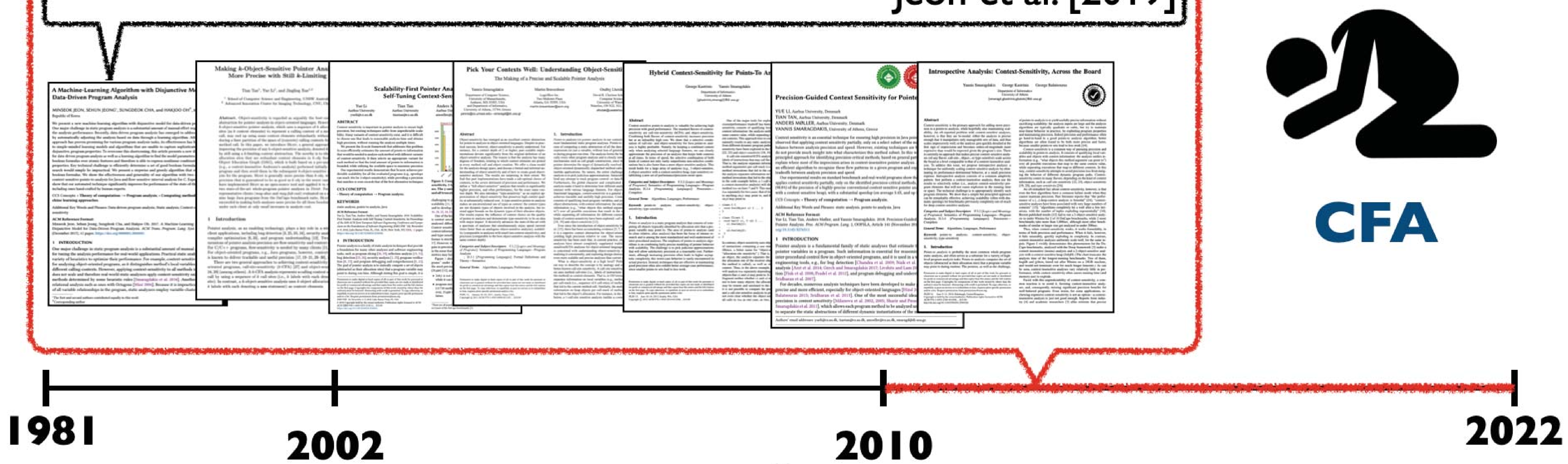
Thank you

Summary

- Currently, CFA is known as a bad context

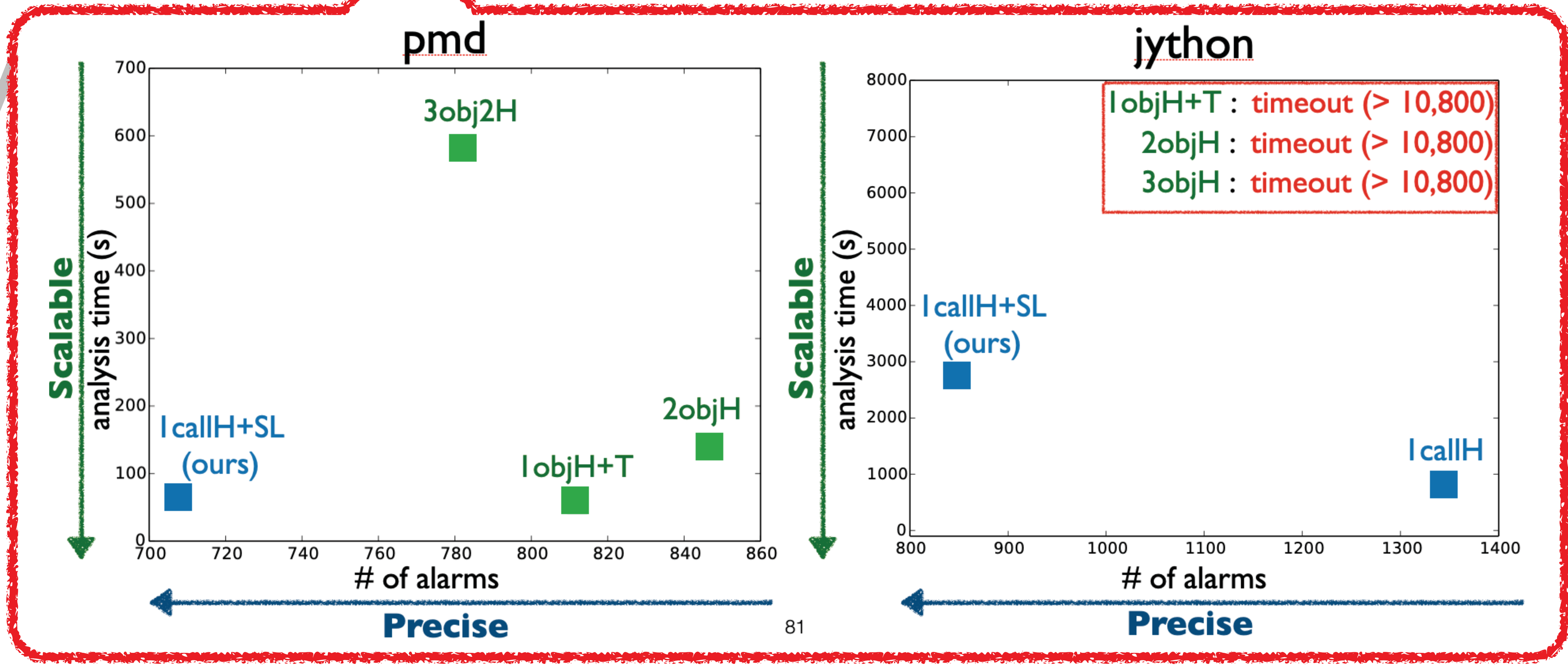
  - Call-site Sensitivity has been ignored

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

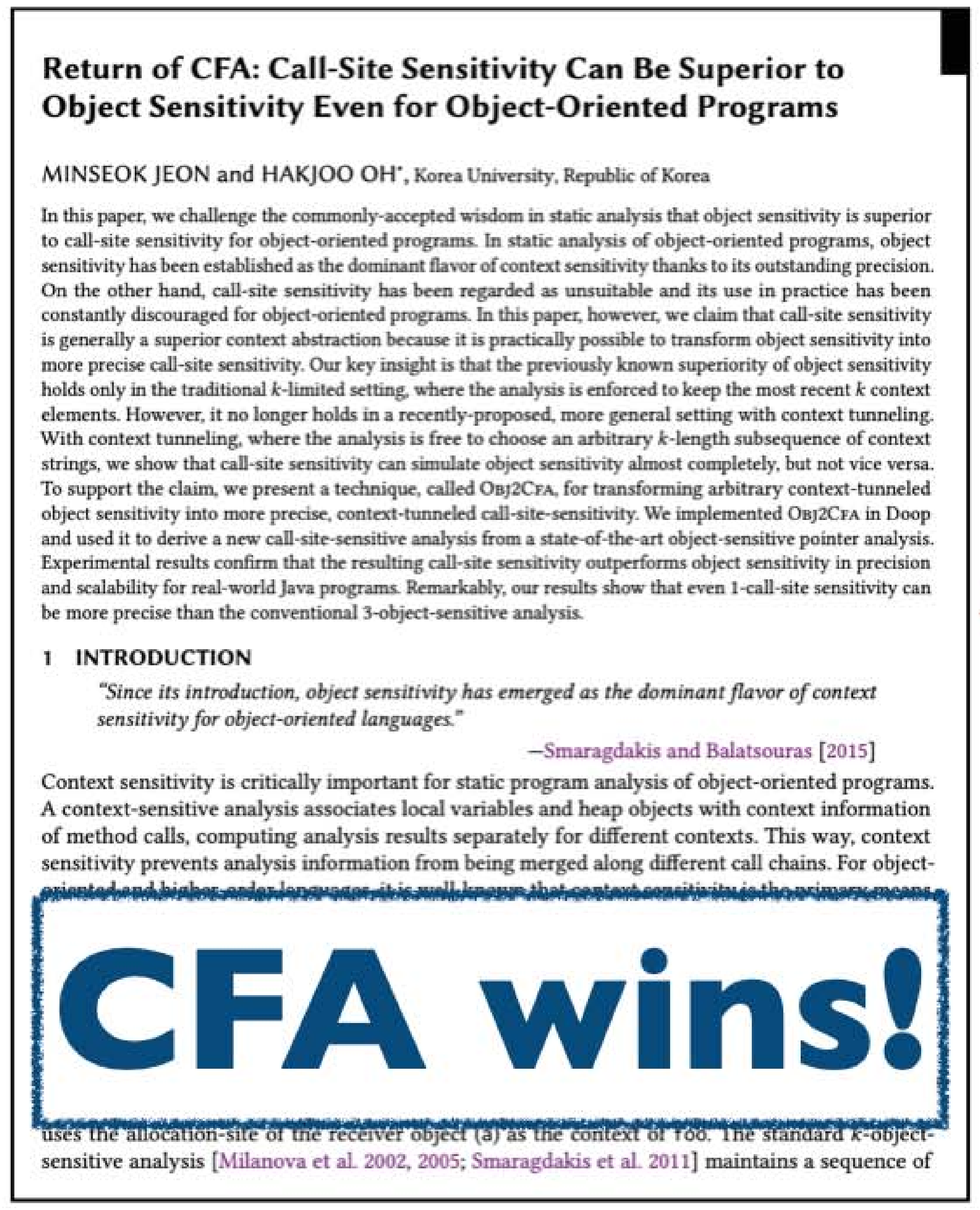


Summary

- 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

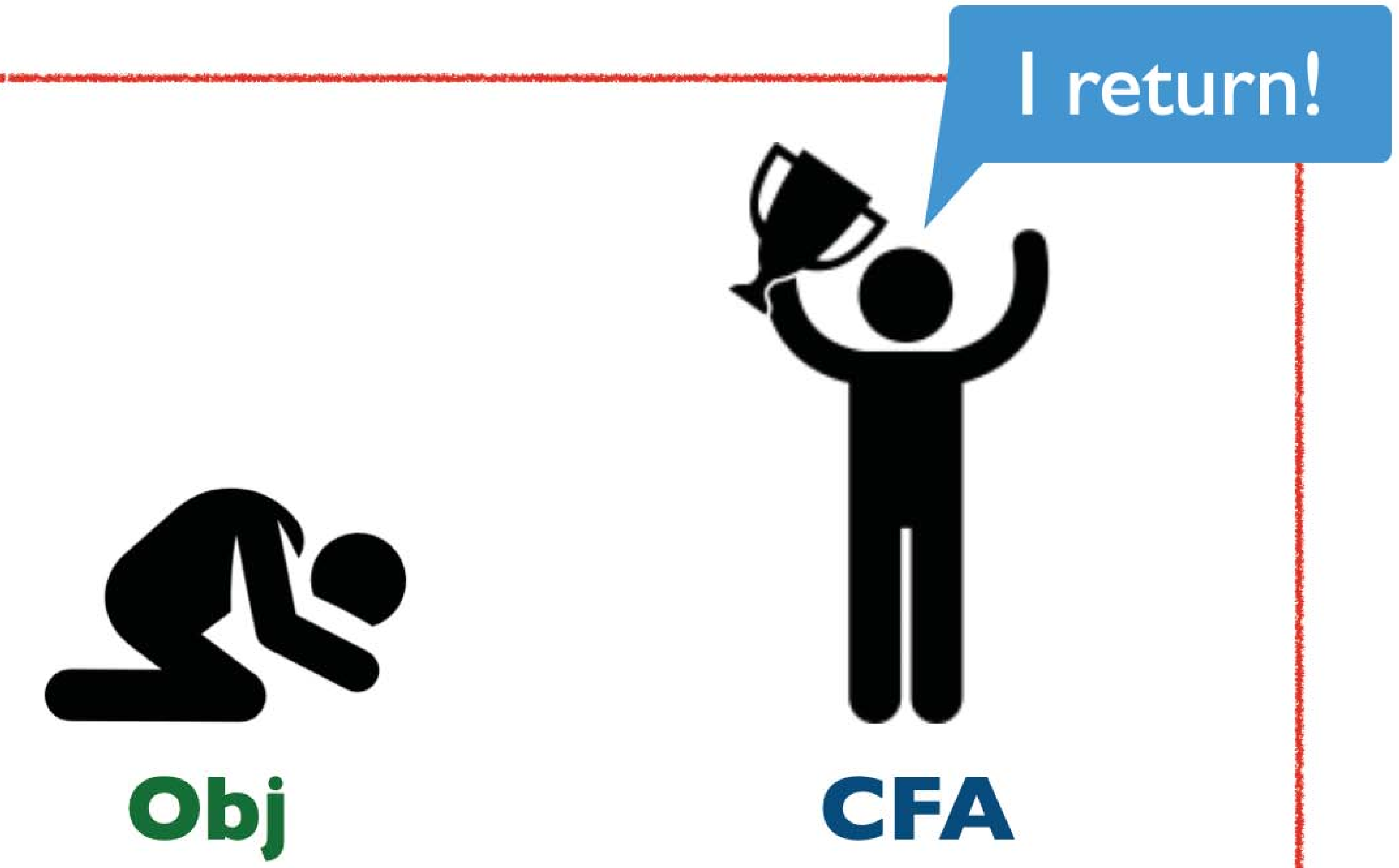
"Since its introduction, object sensitivity has emerged as the dominant flavor of context sensitivity for object-oriented languages."

—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