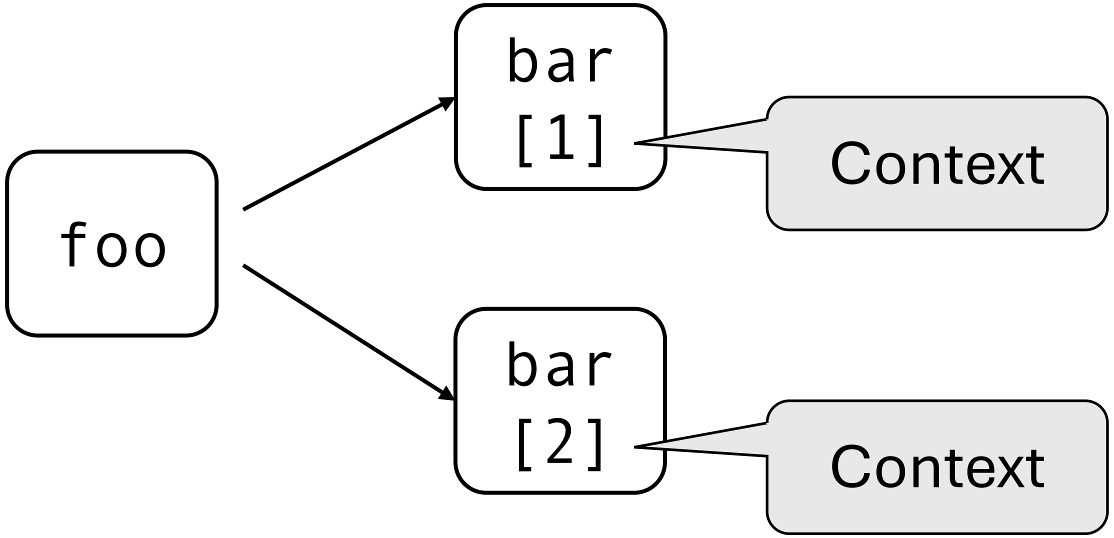
Return of CFA: **Call-Site Sensitivity Can Be Superior to Object Sensitivity** Even for Object-Oriented Programs

Call-Site Sensitivity vs Object Sensitivity

Call-Site Sensitivity vs Object Sensitivity

Call-Site Sensitivity considers "where"

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

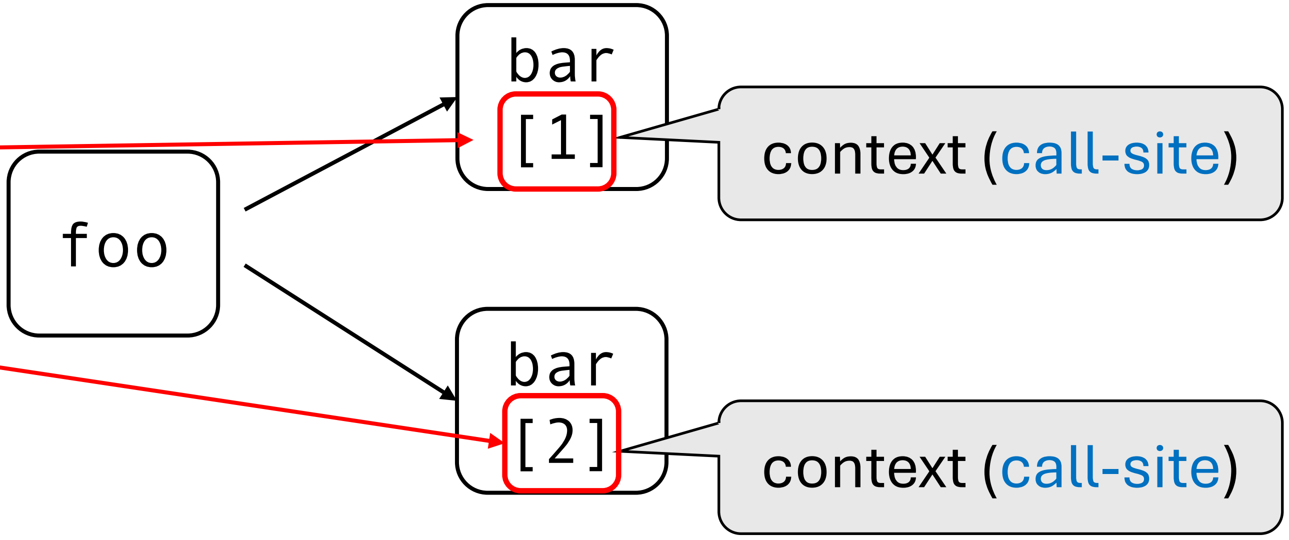


Call graph

Call-Site Sensitivity vs Object Sensitivity

Call-Site Sensitivity considers "where"

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



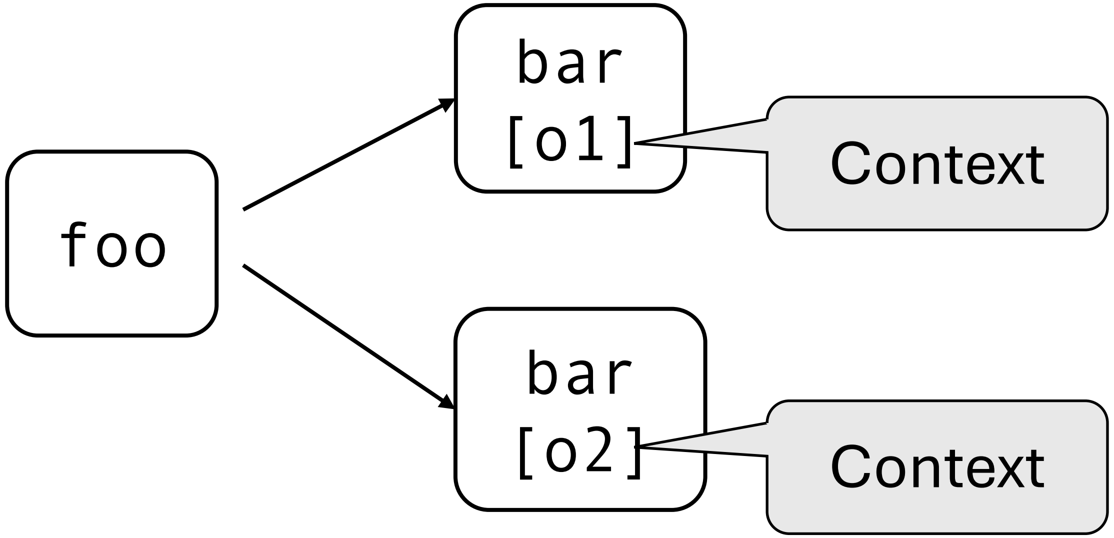
Call graph

Call-Site Sensitivity vs Object Sensitivity

Object Sensitivity considers “what”

o1 or o2

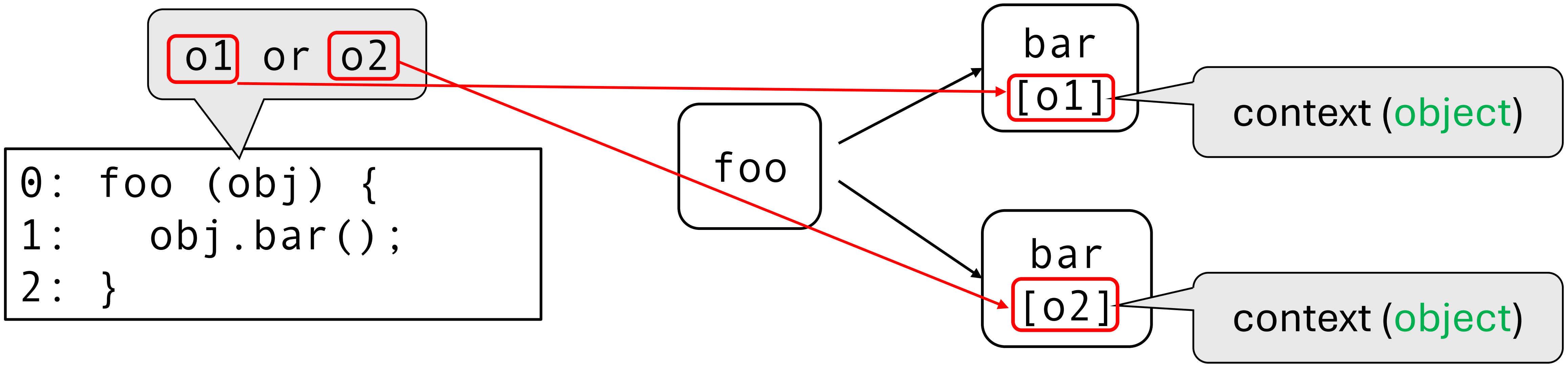
```
0: foo (obj) {
1:    obj.bar();
2: }
```



Call graph

Call-Site Sensitivity vs Object Sensitivity

Object Sensitivity considers "what"

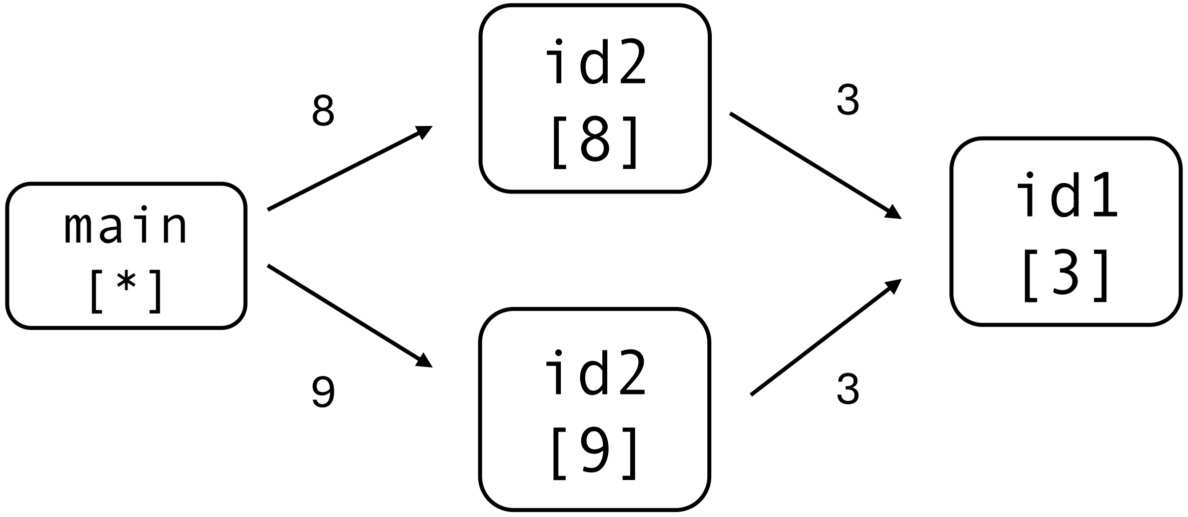


Call graph

Call-Site Sensitivity vs Object Sensitivity

Example: weakness of call-site sensitivity and strength of object sensitivity

```
1 : class C {
2 :    id1(v) { return v };
3 :    id2(v) { return id1(v) };
4 : }
5 : main() {
6 :    c1 = new C();
7 :    c2 = new C();
8 :    A a = (A) c1.id2(new A());
9 :    B b = (B) c2.id2(new B());
10: }
```

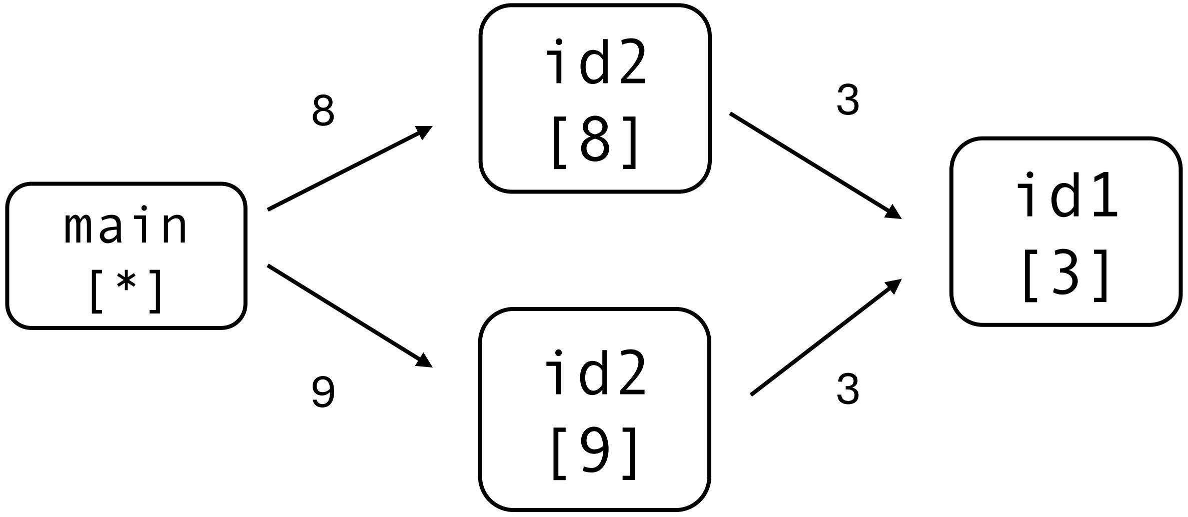


Call graph of 1-call-site

Call-Site Sensitivity vs Object Sensitivity

Example: weakness of call-site sensitivity and strength of object sensitivity

```
1 : class C {
2 :    id1(v) { return v };
3 :    id2(v) { return id1(v) };
4 : }
5 : main() {
6 :    c1 = new C();
7 :    c2 = new C();
8 :    A a = (A) c1.id2(new A());
9 :    B b = (B) c2.id2(new B());
10: }
```

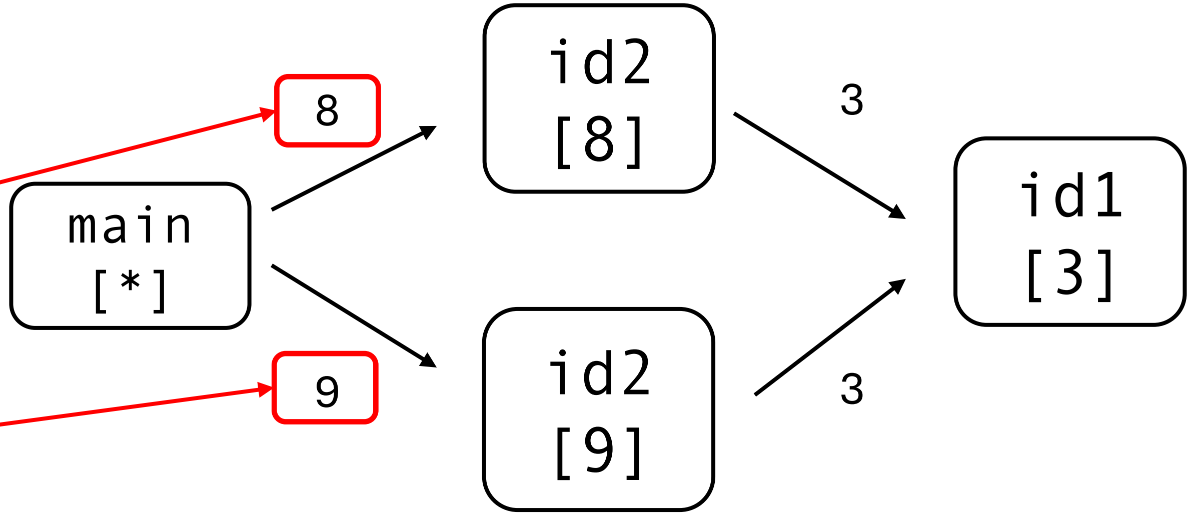


Call graph of 1-call-site

Call-Site Sensitivity vs Object Sensitivity

Example: weakness of call-site sensitivity and strength of object sensitivity

```
class C {
   id1(v) { return v };
   id2(v) { return id1(v) };
}
main() {
   c1 = new C();
   c2 = new C();
   A a = (A) c1.id2(new A());
   B b = (B) c2.id2(new B());
}
```



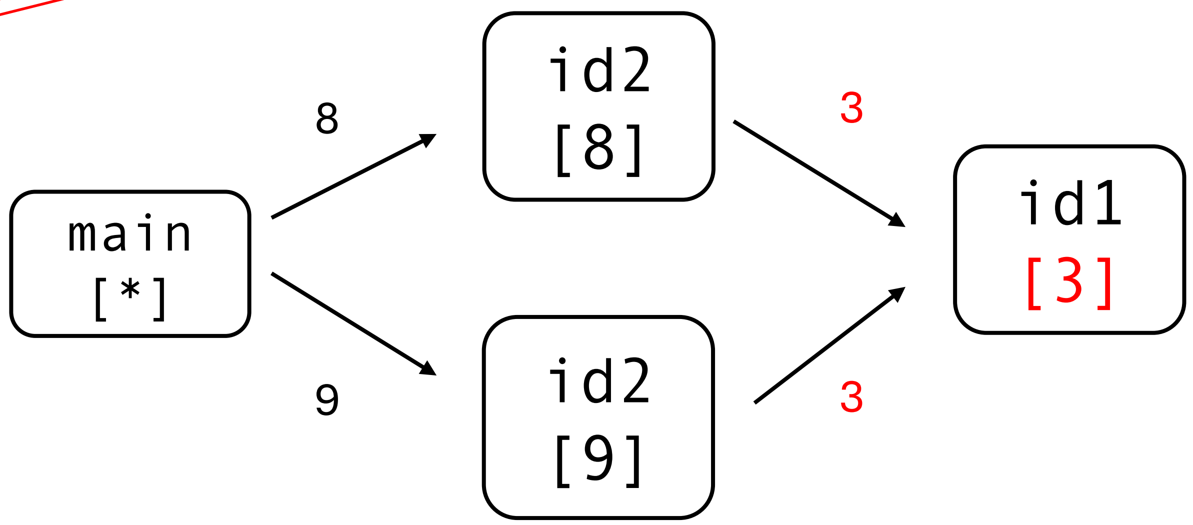
Call graph of 1-call-site

Call-Site Sensitivity vs Object Sensitivity

Example: weakness of call-site sensitivity and … vity

Weakness:
nested method calls

```
class C {
   id1(v) { return v };
   id2(v) { return id1(v) };
}
main() {
   c1 = new C();
   c2 = new C();
   A a = (A) c1.id2(new A());
   B b = (B) c2.id2(new B());
}
```

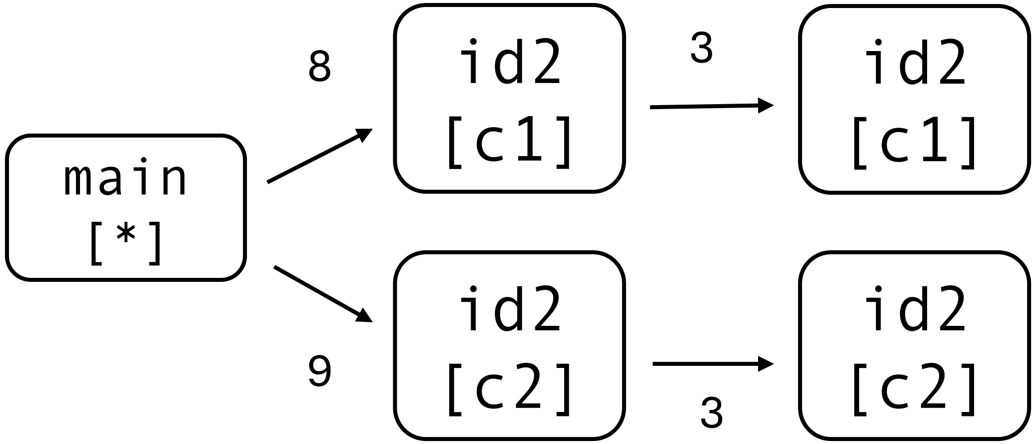


Call graph of 1-call-site

Call-Site Sensitivity vs Object Sensitivity

Example: weakness of call-site sensitivity and strength of object sensitivity

```
class C {
   id1(v) { return v };
   id2(v) { return id1(v) };
}
main() {
   c1 = new C();
   c2 = new C();
   A a = (A) c1.id2(new A());
   B b = (B) c2.id2(new B());
}
```



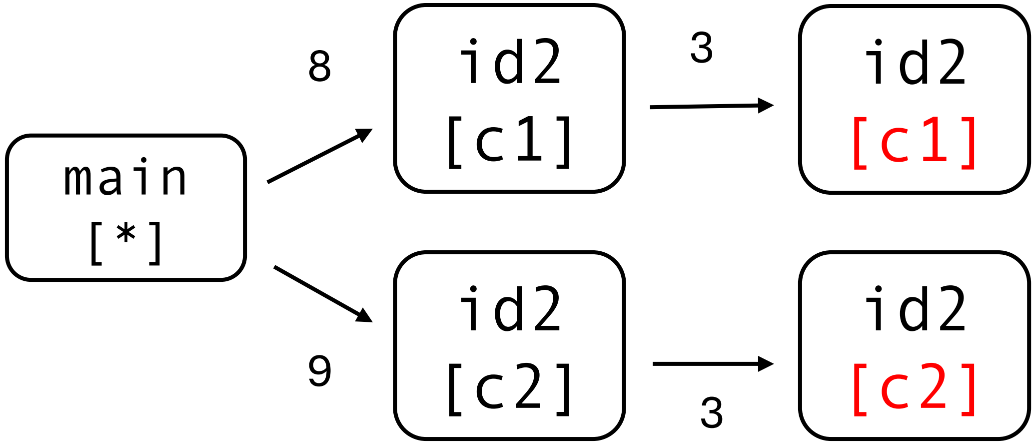
Call graph of 1-object

Call-Site Sensitivity vs Object Sensitivity

Example: weal ty and strength of object sensitivity

Strength:
c1 or c2

```
class C {
   id1(v) { return v };
   id2(v) { return id1(v) };
}
main() {
   c1 = new C();
   c2 = new C();
   A a = (A) c1.id2(new A());
   B b = (B) c2.id2(new B());
}
```

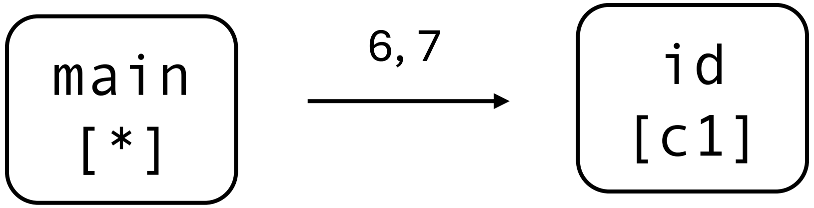


Call graph of 1-object

Call-Site Sensitivity vs Object Sensitivity

Example: weakness of object sensitivity and strength of call-site sensitivity

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

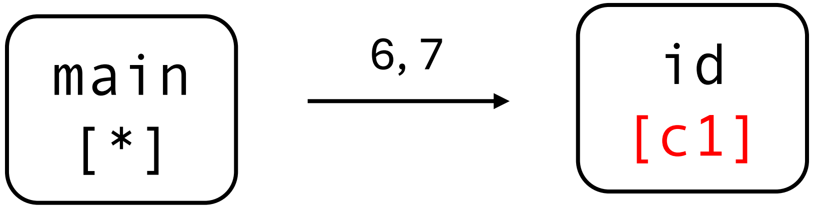


Call graph of 1-object

Call-Site Sensitivity vs Object Sensitivity

Example: weakness of object sensitivity and strength of call-site sensitivity

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



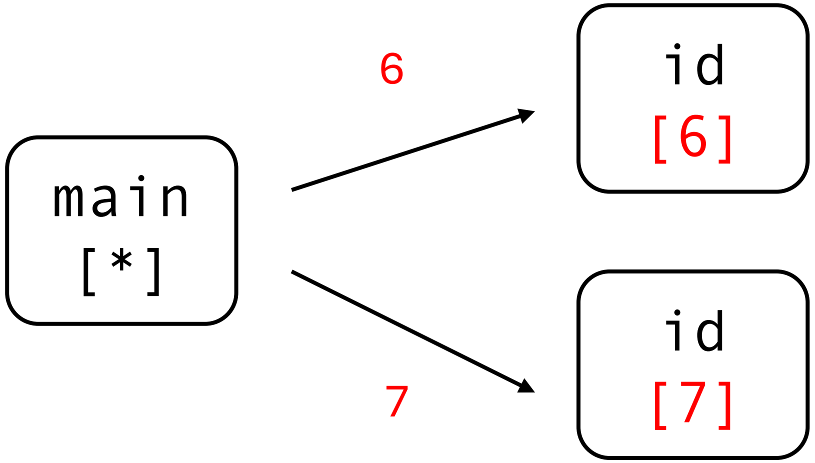
Call graph of 1-object

Weakness:
The two method calls share the same object c1

Call-Site Sensitivity vs Object Sensitivity

Example: weakness of object sensitivity and strength of call-site sensitivity

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



Call graph of 1-call-site

Strength:
Eeasily seperates the two method calls

Call-Site Sensitivity vs Object Sensitivity

The Status Quo: Object sensitivity outperforms call-site sensitivity

1981: Call-Site sensitivity proposed

2002: Object sensitivity proposed

2002 ~ 2010: Object vs Call-site

2010 ~ 2022: Object win

- Researchers focused on improving Object sensitivity approach

Call-Site Sensitivity vs Object Sensitivity

The Status Quo: Object sensitivity outperforms call-site sensitivity

1981: Call-Site sensitivity proposed

2002: Object sensitivity proposed

2002 ~ 2010: Object vs Call-site

2018: Context tunneling proposed

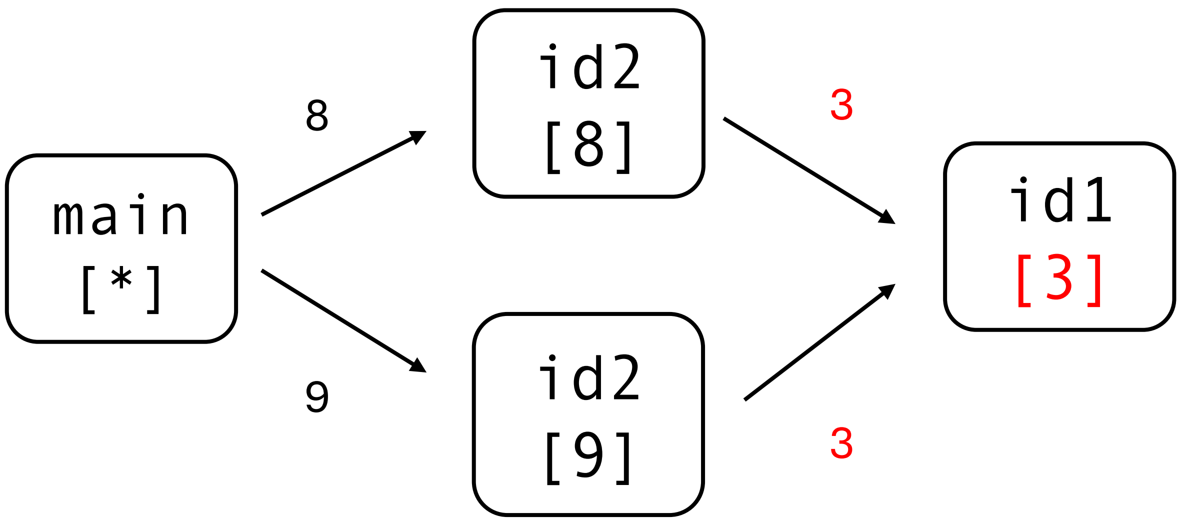
2010 ~ 2022: Object win

- Researchers focused on improving Object sensitivity approach

Call-Site Sensitivity vs Object Sensitivity

Context tunneling can remove the weakness of call-site sensitivity

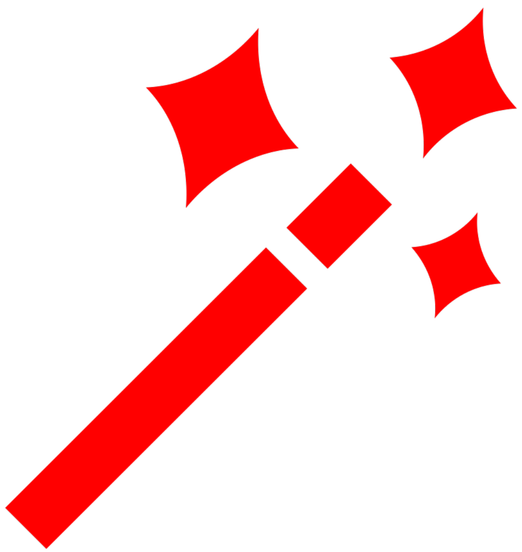
```
1 : class C {
2 :    id1(v) { return v };
3 :    id2(v) { return id1(v) };
4 : }
5 : main() {
6 :    c1 = new C();
7 :    c2 = new C();
8 :    A a = (A) c1.id2(new A());
9 :    B b = (B) c2.id2(new B());
10: }
```



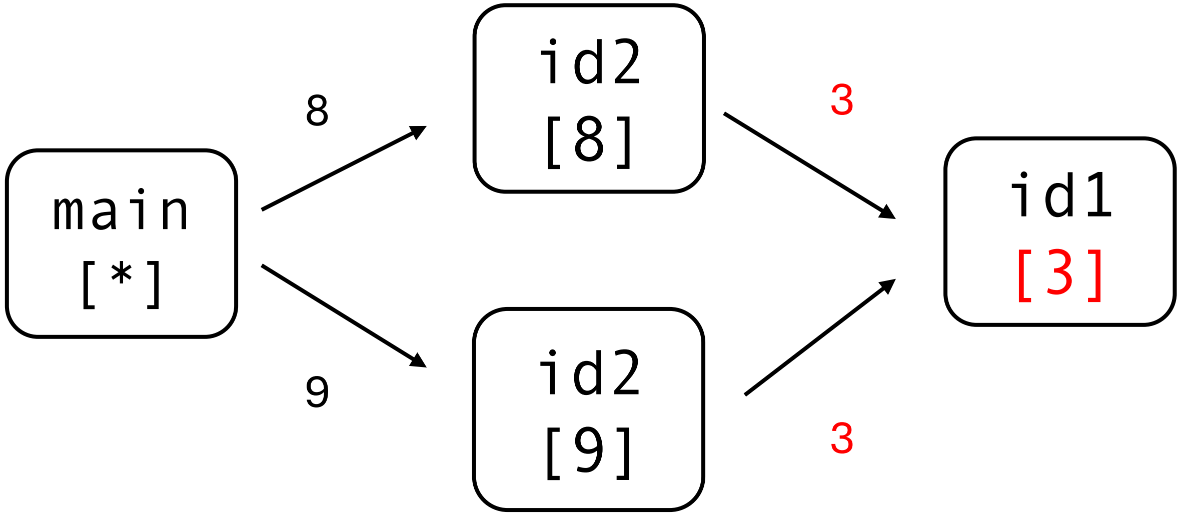
Call graph of 1-call-site

Call-Site Sensitivity vs Object Sensitivity

Context tunneling can remove the weakness of call-site sensitivity



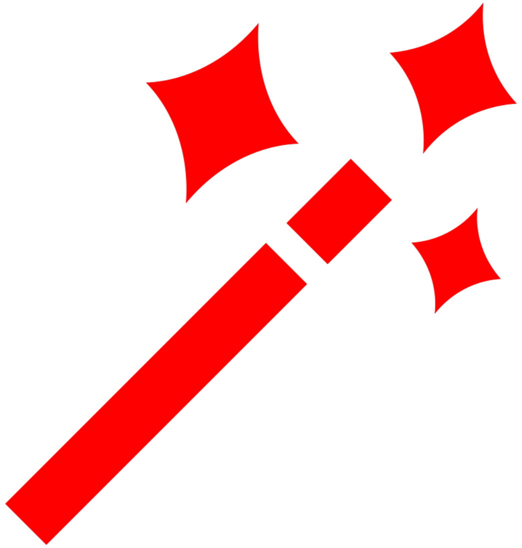
Context Tunneling



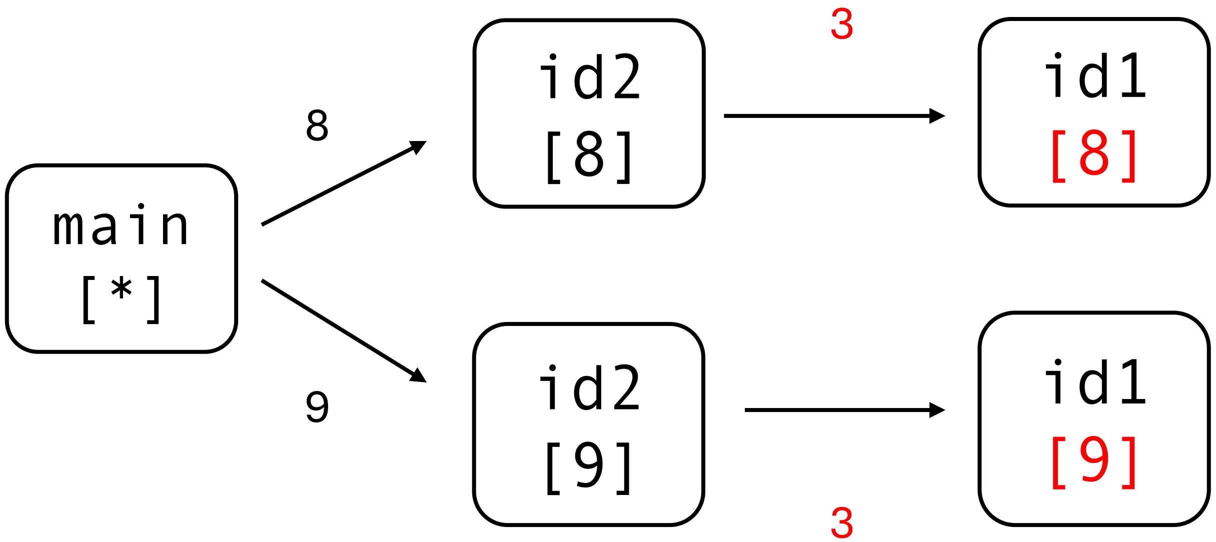
Call graph of 1-call-site

Call-Site Sensitivity vs Object Sensitivity

Context tunneling can remove the weakness of call-site sensitivity



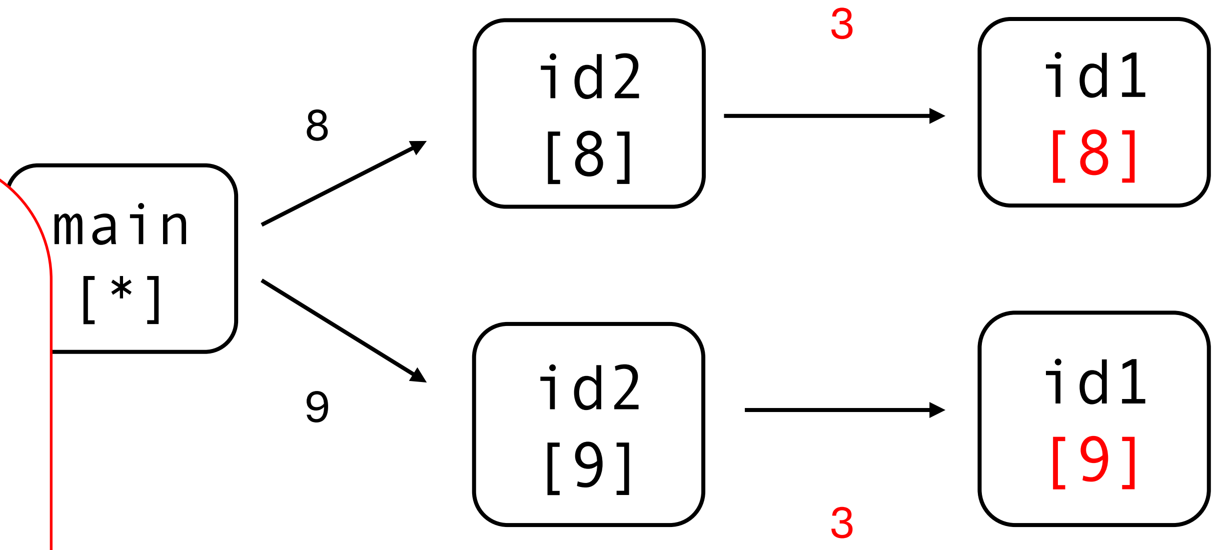
Context Tunneling



Call graph of 1-call-site
with context tunneling
(T = {3})

Call-Site Sensitivity vs Object Sensitivity

Context tunneling can remove the weakness of call-site sensitivity



Tunneling abstraction:

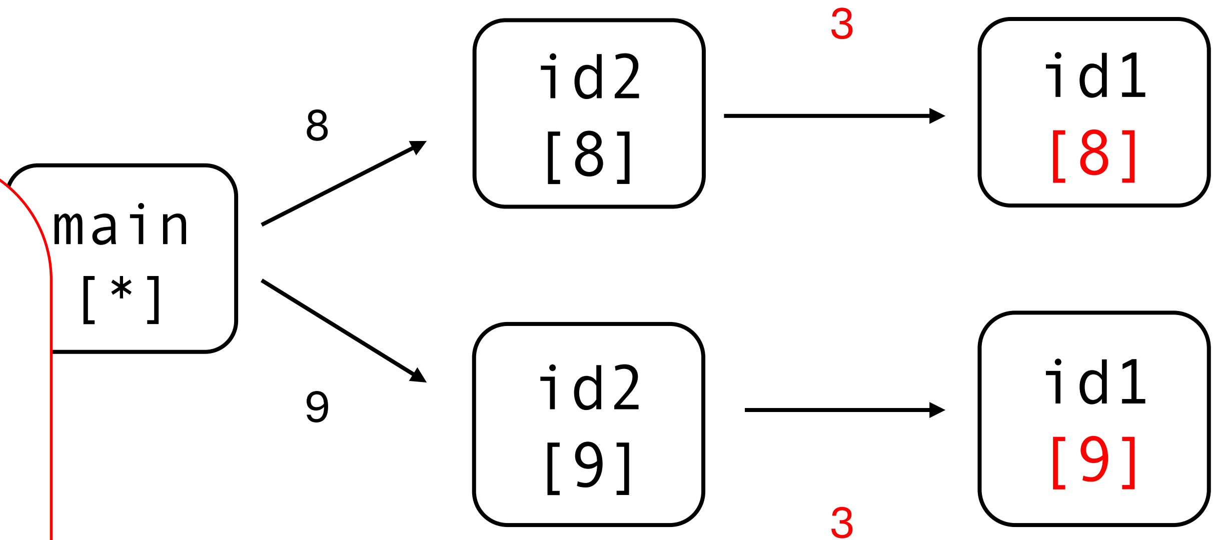
- Determines where to apply context tunneling

Call graph of 1-call-site
with context tunneling
(**T = {3}**)

Call-Site Sensitivity vs Object Sensitivity

Context tunneling can remove the weakness of call-site sensitivity

Tunneling abstraction:
- Determines where to apply context tunneling
- Refers a set of unimportant call-sites that should not be used as a context elemtents



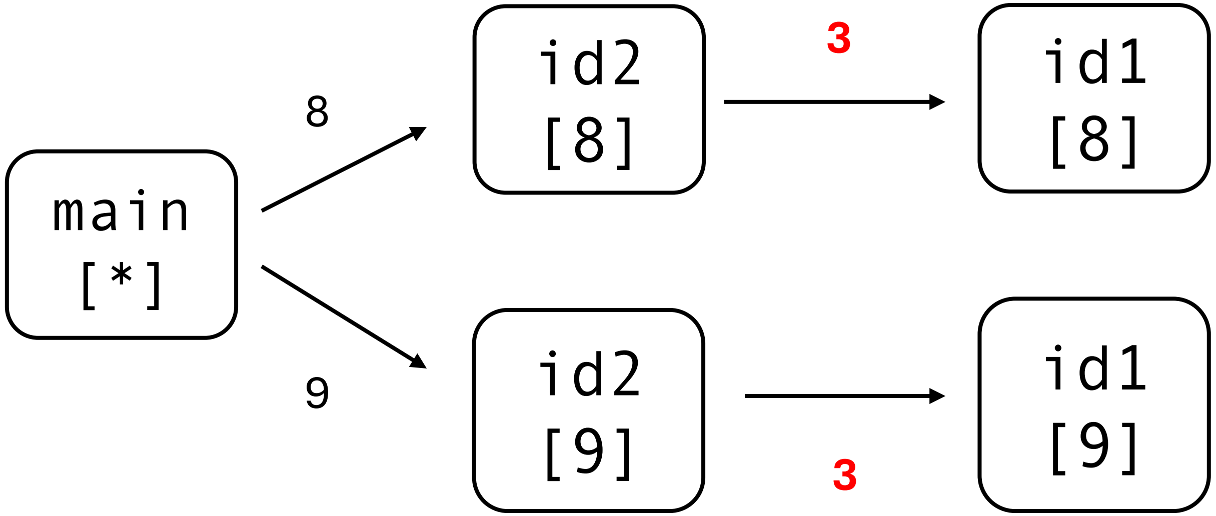
Call graph of 1-call-site with context tunneling (**T = {3}**)

Call-Site Sensitivity vs Object Sensitivity

Context tunneling can remove the weakness of call-site sensitivity

Apply context tunneling:

1. **Detect tunneling abstraction (T = {3})**



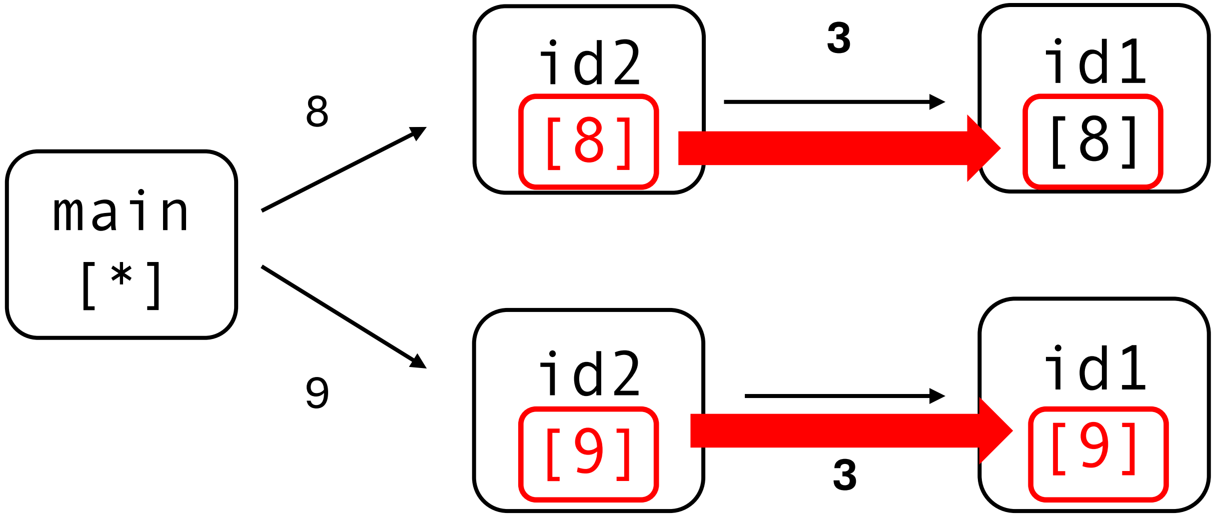
Call graph of 1-call-site with context tunneling (**T = {3}**)

Call-Site Sensitivity vs Object Sensitivity

Context tunneling can remove the weakness of call-site sensitivity

Apply context tunneling:

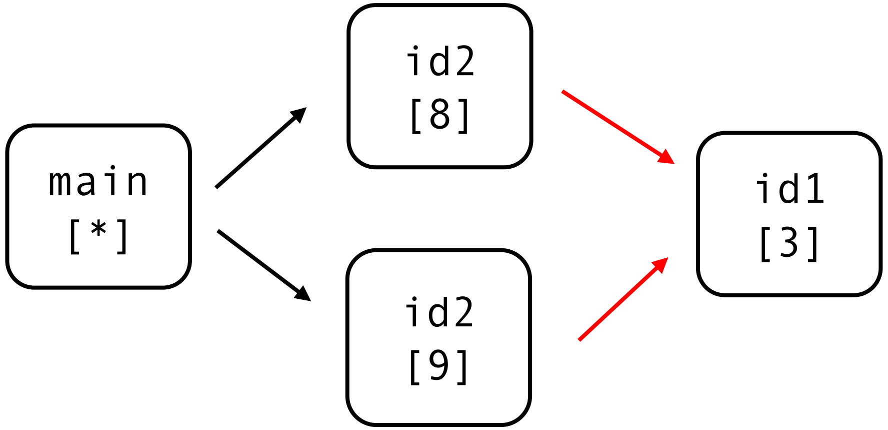
1. **Detect tunneling abstraction (T = {3})**
2. **Inherit caller method's context**



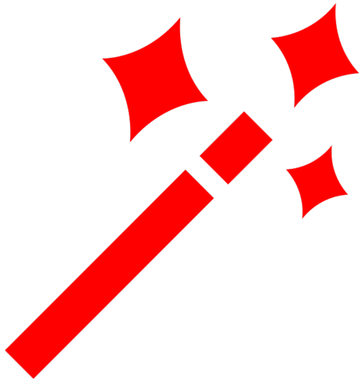
Call graph of 1-call-site
with context tunneling
(**T = {3}**)

Call-Site Sensitivity vs Object Sensitivity

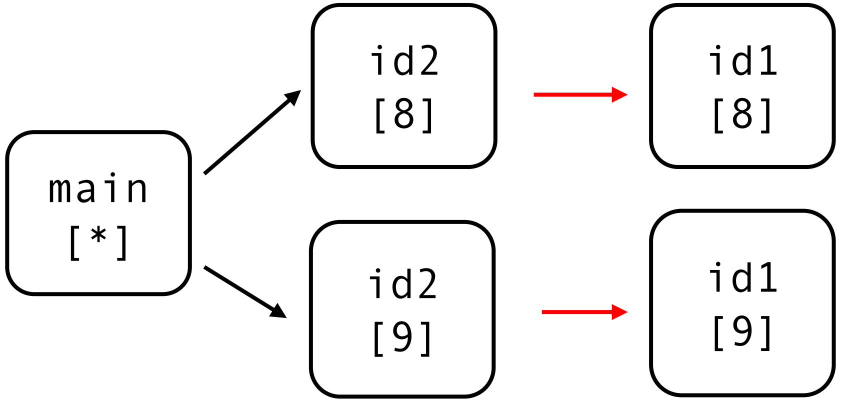
Context tunneling can remove the weakness of call-site sensitivity



Call graph of 1-call-site



Context Tunneling



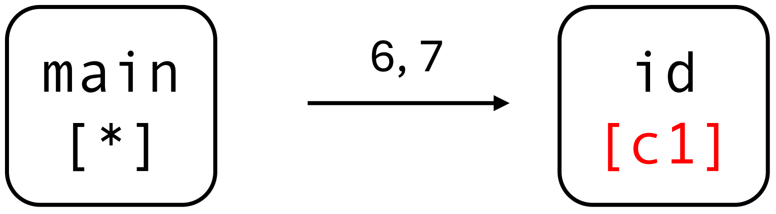
Call graph of 1-call-site
with context tunneling
(T = {3})

Call-Site Sensitivity vs Object Sensitivity

Context tunneling can't remove the weakness of object sensitivity

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

Weakness:
The two method calls share the same object c1

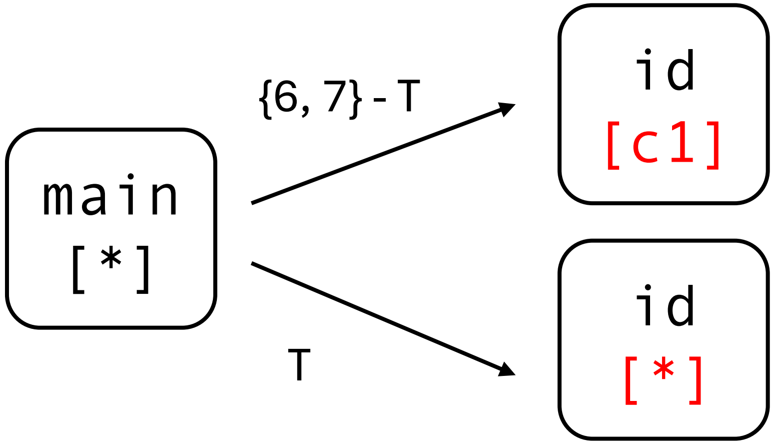


Call graph of 1-object

Call-Site Sensitivity vs Object Sensitivity

Context tunneling can't remove the weakness of object sensitivity

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



Call graph of 1-call-site with context tunneling (**T = {?}**)

Unable to separate the two method calls with **two context (c1 and *)**

Call-Site Sensitivity vs Object Sensitivity

Obervation

When context tunneling is included,

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

Call-Site Sensitivity vs Object Sensitivity

Obervation

When context tunneling is included,

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

Claim

When context tunneling is included,

- **Call-site sensitivity** is **more precise** than **obejct sensitivity**

Call-Site Sensitivity vs Object Sensitivity

Obervation

When context tunneling is included,

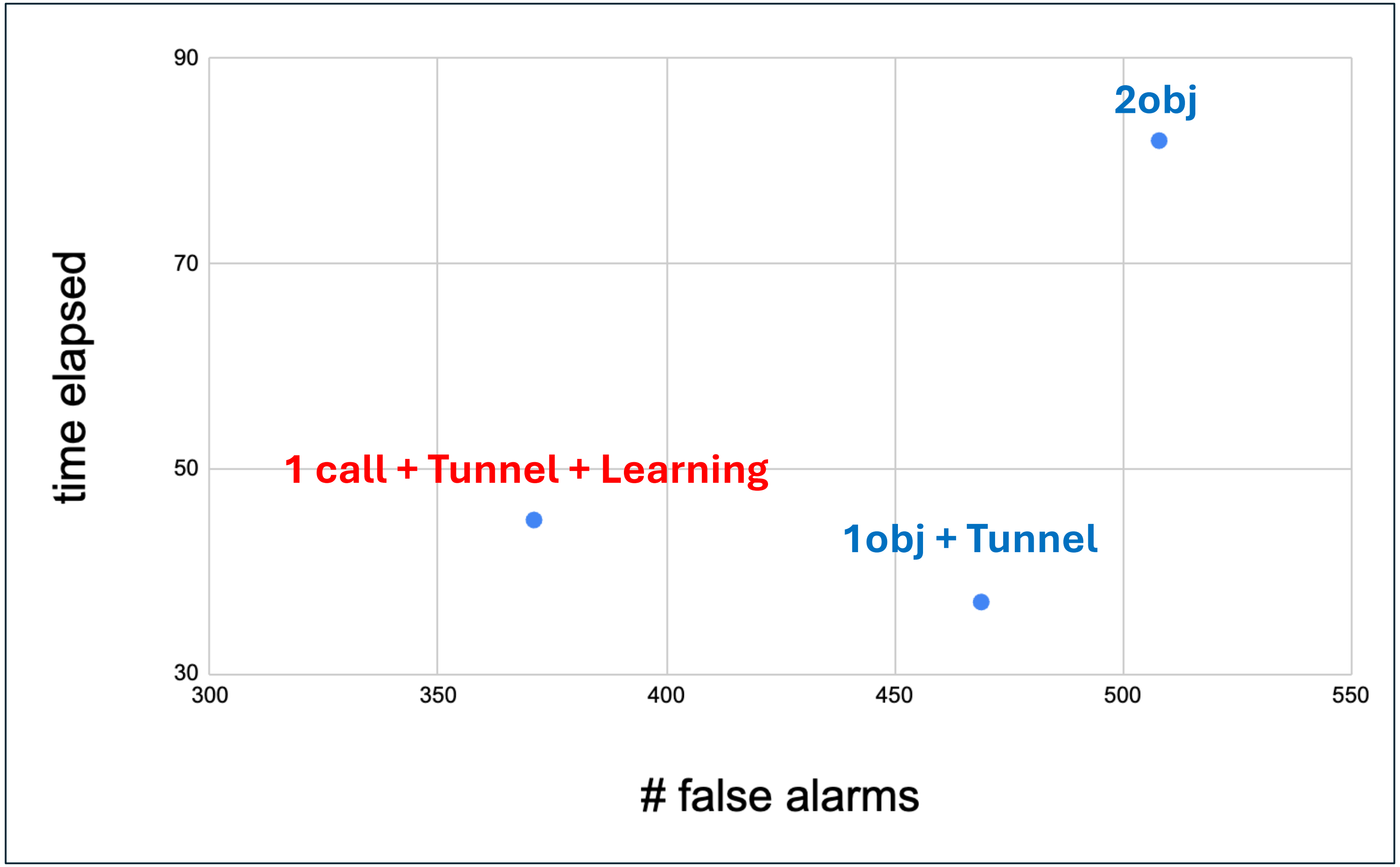
- Weakness of **call-site sensitivity** is **removed**
- Weakness of **object sensitivity** is not removed

Claim

When context tunneling is included,

- **Call-site sensitivity** is **more precise** than **obejct sensitivity**

Results



90
70
50
30
time elapsed
2obj
1 call + Tunnel + Learning
1obj + Tunnel
300
350
400
450
500
550
false alarms

More details about Obj2CFA

- Obj2CFA uses simulation technique
  1. Runs the object-sensitive analysis to obtain it's **call graph**
  2. Infers a tunneling abstraction
- Simulation: Finding more precise call-site sensitivity, but expensive
  - Apply Simulation-guided Learning to Improve scalability

Obj2CFA - Simulation

Simulation takes a call-graph and infers a tunneling abstraction

```
class C {
  id1(v) { return v };
  id2(v) { return id1(v)};
  foo() {
    A a = (A)this.id1(new A());
    B b = (B)this.id1(new B());}
}
main() {
  c1 = new C();
  c2 = new C();
  c3 = new C();
  A a = (A) c1.id2(new A());
  B b = (B) c2.id2(new B());
  c3.foo();
}
```

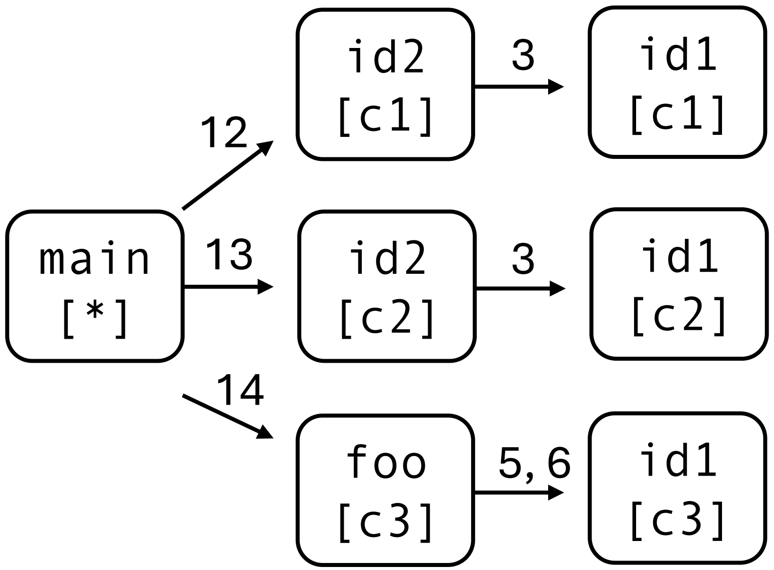
Weakness of call-site sensitivity

Weakness of object sensitivity

Obj2CFA - Simulation

Simulation takes a call-graph and infers a tunneling abstraction

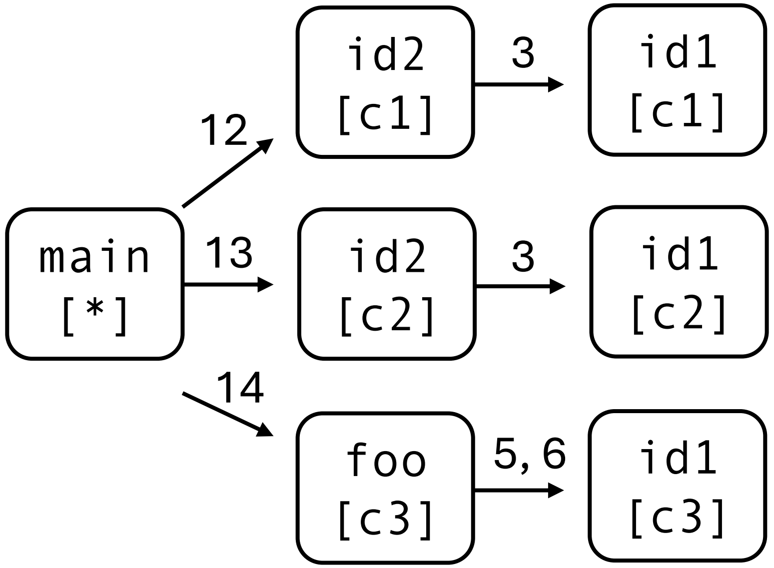
```
1 : class C {
2 :   id1(v) { return v };
3 :   id2(v) { return id1(v)};
4 :   foo() {
5 :     A a = (A)this.id1(new A());
6 :     B b = (B)this.id1(new B());}
7 : }
8 : main() {
9 :   c1 = new C();
10:   c2 = new C();
11:   c3 = new C();
12:   A a = (A) c1.id2(new A());
13:   B b = (B) c2.id2(new B());
14:   c3.foo();
15: }
```



1-Obj + Tunneling
(T = Φ)

Obj2CFA - Simulation

Simulation takes a call-graph and infers a tunneling abstraction



1-Obj + Tunneling
(T = Φ)

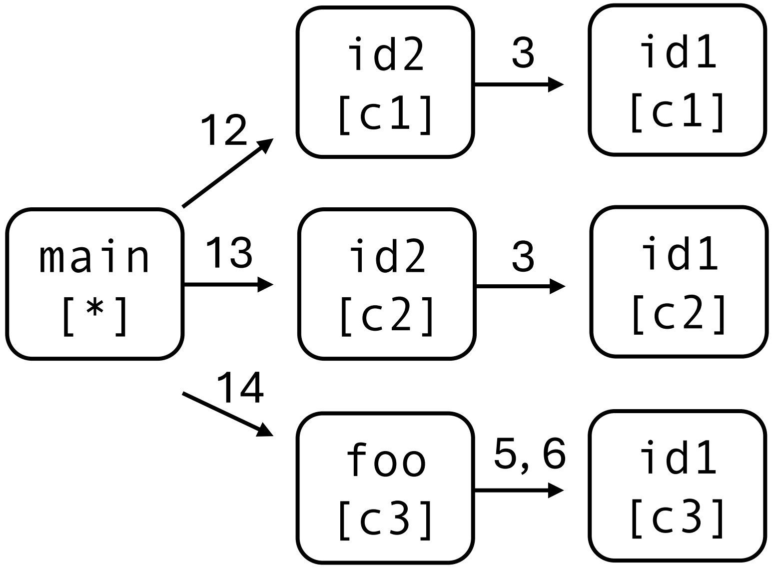
Simulation

T' = {3}

Tunneling abstraction
for 1-call-site sensitivity

Obj2CFA - Simulation

Simulation takes a call-graph and infers a tunneling abstraction



1-Obj + Tunneling
(T = Φ)

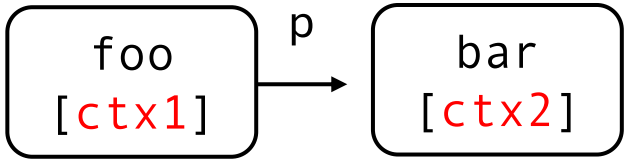
Simulation → **T' = {3}**

T' = (P1 ∪ P2) – P3

(P1 ∪ P2) : Interesting context to simulate
P3: Not interesting context to simulate

Obj2CFA - Simulation

Simulation takes a call-graph and infers a tunneling abstraction

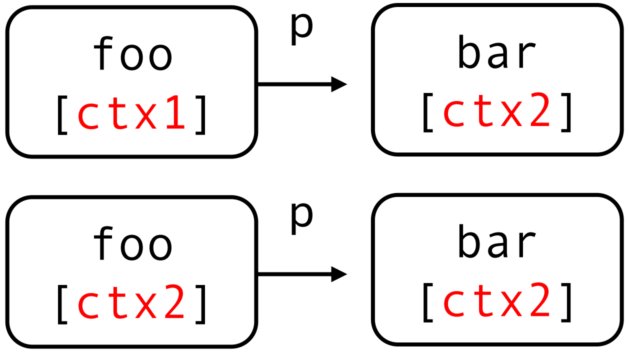


T' = (P1 ∪ P2) – P3

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

Obj2CFA - Simulation

Simulation takes a call-graph and infers a tunneling abstraction

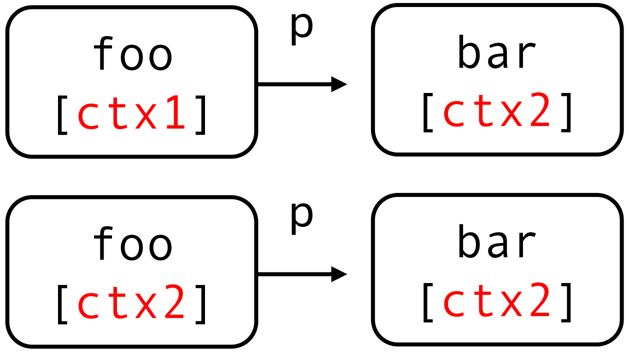


T’ = (P1 ∪ P2) – P3

- Property 1: caller and callee methods have the same context
- Property 2: different caller contexts imply different callee contexts

Obj2CFA - Simulation

Simulation takes a call-graph and infers a tunneling abstraction

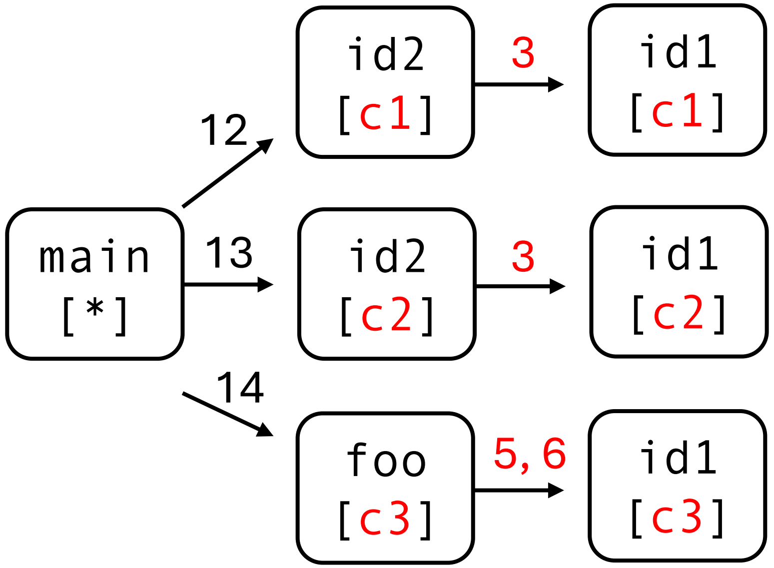


T' = (P1 ∪ P2) – P3

- Property 1: caller and callee methods have the same context
- Property 2: different caller contexts imply different callee contexts
- Property 3: given object sensitivity produced only one context

Obj2CFA - Simulation

Simulation takes a call-graph and infers a tunneling abstraction

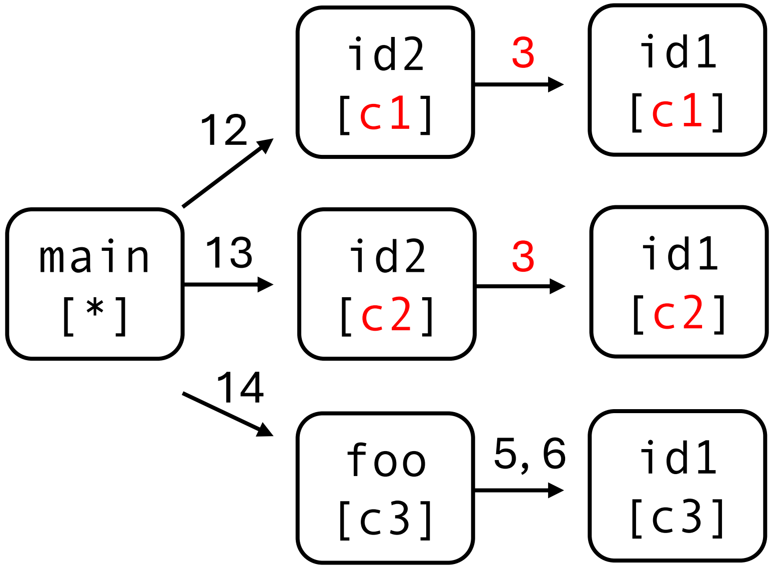


1-Obj + Tunneling
(T = Φ)

- P1: caller and calle methods have the same context
  P1 = {3, 5, 6}

Obj2CFA - Simulation

Simulation takes a call-graph and infers a tunneling abstraction

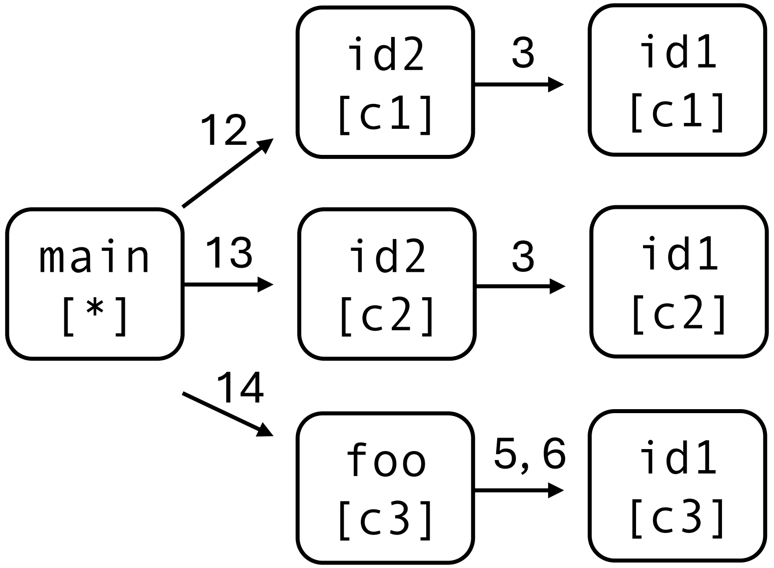


1-Obj + Tunneling
(T = Φ)

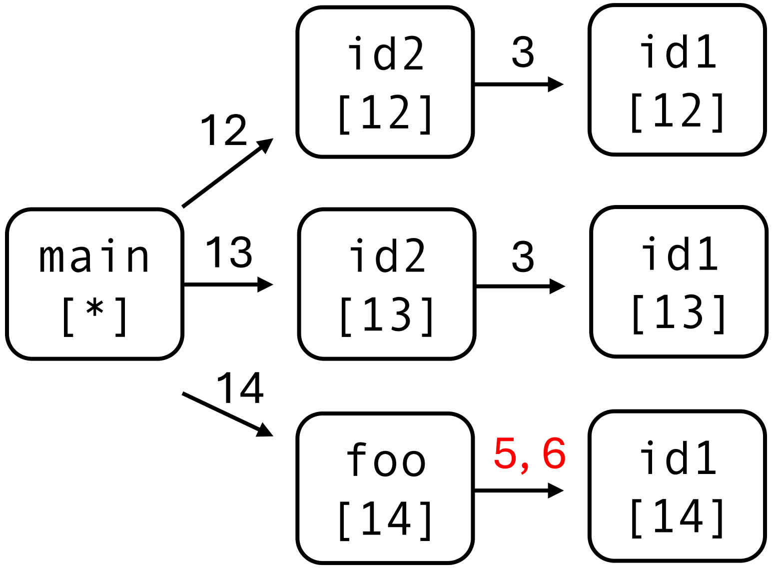
- P1: caller and calle methods have the same context
  P1 = {3, 5, 6}
- P2: different caller context imply different callee context
  P2 = {3}

Obj2CFA - Simulation

Simulation takes a call-graph and infers a tunneling abstraction



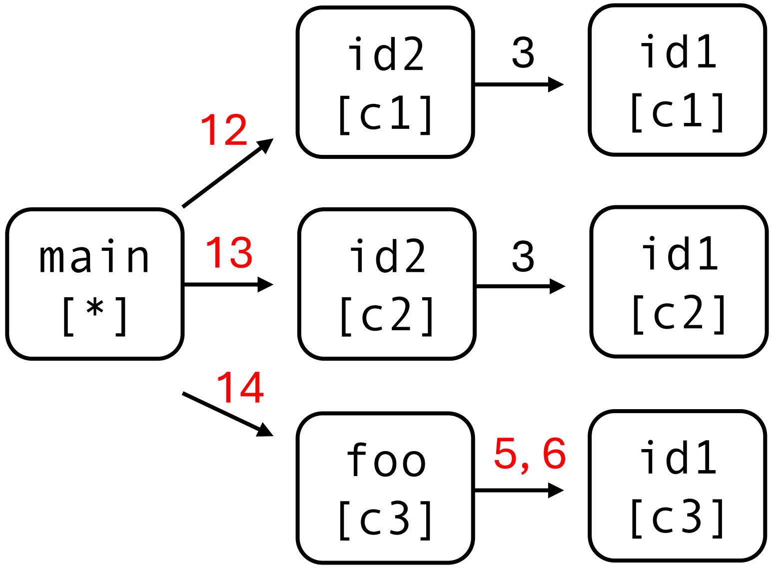
1-Obj + Tunneling
(T = Φ)



1-call-site + Tunneling
(T = P1 ∪ P2 = {3, 5, 6})

Obj2CFA - Simulation

Simulation takes a call-graph and infers a tunneling abstraction

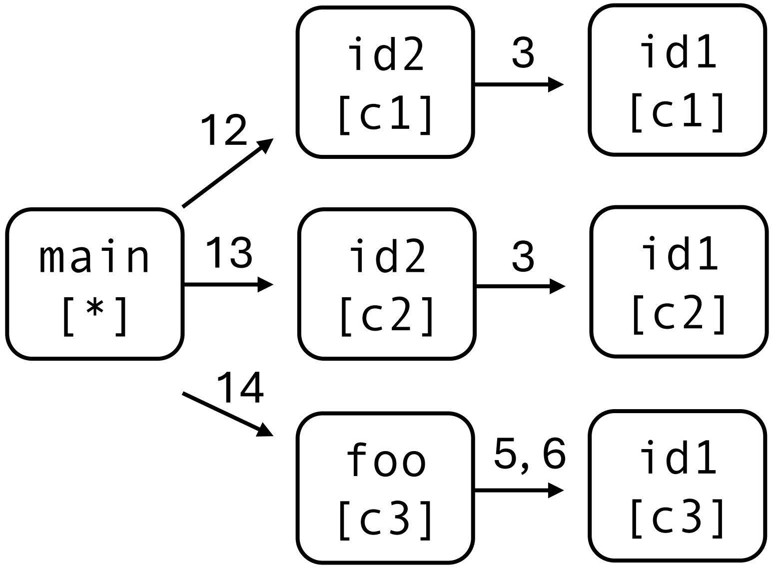


1-Obj + Tunneling
(T = Φ)

- P1: caller and calle methods have the same context
  P1 = {3, 5, 6}
- P2: different caller context imply different callee context
  P2 = {3}
- P3: given object sensitivity produced only one context
  P3 = {5, 6, 12, 13, 14}

Obj2CFA - Simulation

Simulation takes a call-graph and infers a tunneling abstraction



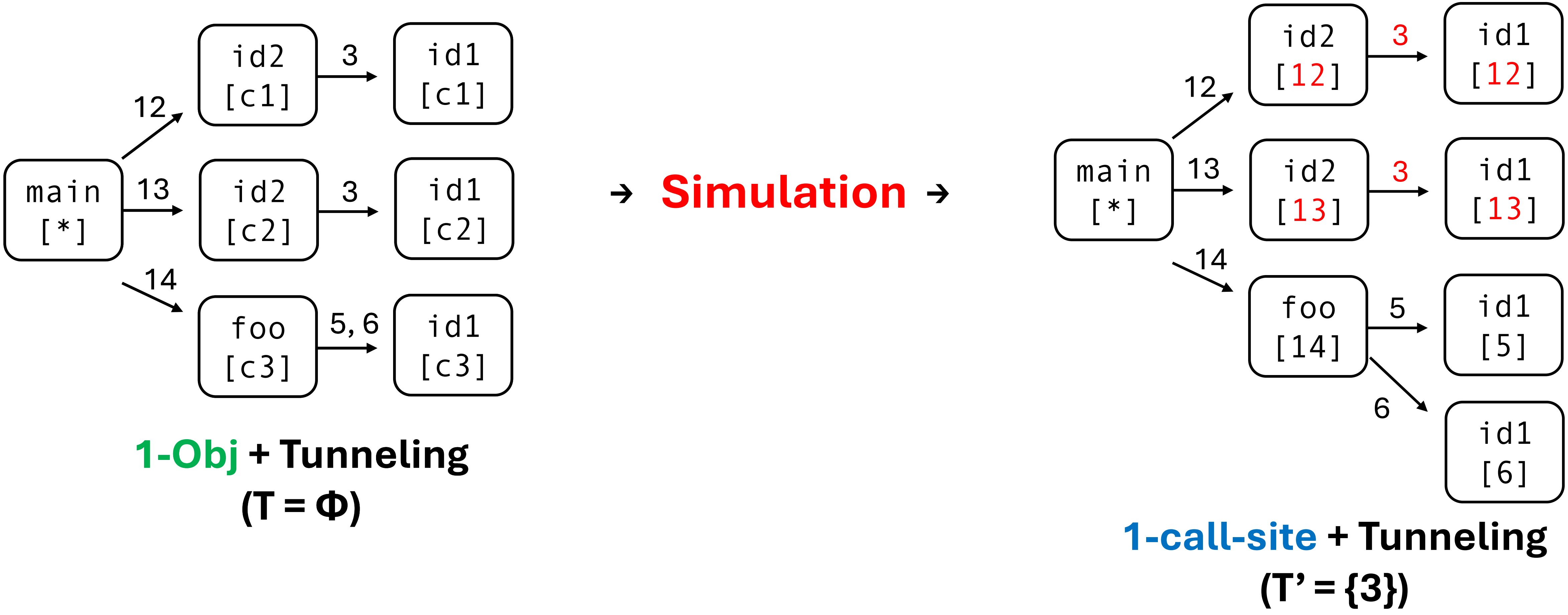
1-Obj + Tunneling
(T = Φ)

- P1: caller and calle methods have the same context
  P1 = {3, 5, 6}
- P2: different caller context imply different callee context
  P2 = {3}
- P3: given object sensitivity produced only one context
  P3 = {5, 6, 12, 13, 14}

T' = (P1 ∪ P2) – P3 = {3}

Obj2CFA - Simulation

Simulation takes a call-graph and infers a tunneling abstraction



Obj2CFA – Simulation-guided Learning

- Simulation is exepnsive
  - simulation requires object sensitivity to obtain call graph
- Learning
  - Given training programs and simulated tunneling abstractions,
  - Find a model that produces similar tunneling abstractions
- The authors modified the features of the context tunneling paper [1] to capture the specific simulated behavior

[1] Jeong et al. 2017. Data-driven Context-sensitivity for Points-to Analysis. Proc. ACM Program. Lang. 1, OOPSLA

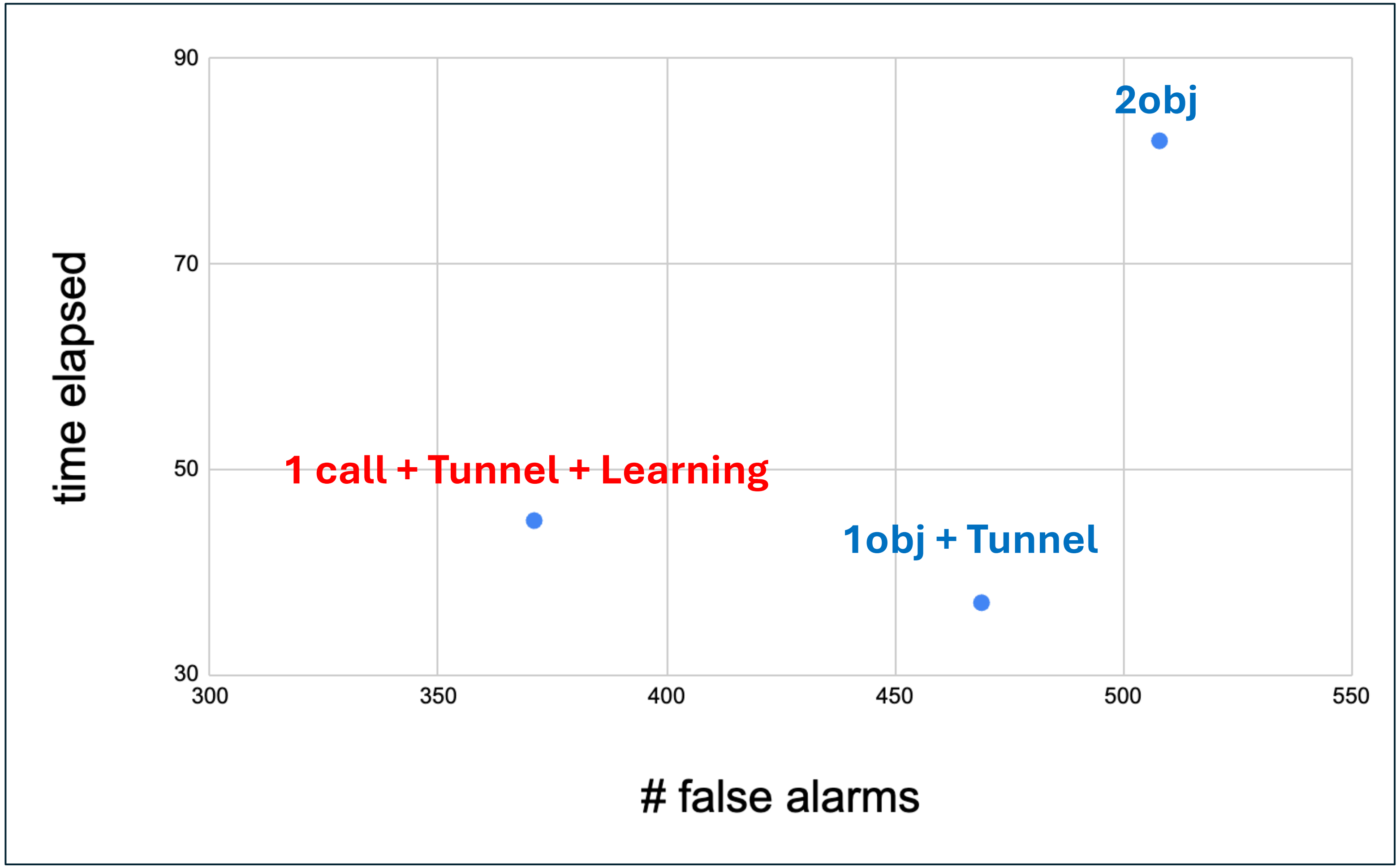
Evaluation

- **Doop**
  - Pointer analysis framework for JAVA
- Dataset
  - 12 Java programs (10 DaCapo 2006 programs + 2 real-world open sourec programs)
- Baselines
  - 1-object-sensitive with tunneling [1]
  - 1-call-site sensitivity with tunneling [1]
  - 2-object-sensitive [2]
  - 2-object sensitivity [3]

[1] Jeong et al. 2017. Data-driven Context-sensitivity for Points-to Analysis. Proc. ACM Program. Lang. 1, OOPSLA
[2] Smaragdakis et al. 2014. Introspective Analysis: Context-sensitivity, Across the Board. In Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation
[3] Li et al. 2018a. Precision-guided Context Sensitivity for Pointer Analysis. Proc. ACM Program. Lang. 2, OOPSLA

Results

Obj2CFA outperformed the baselines in terms of precision and scalability (time elapesed)



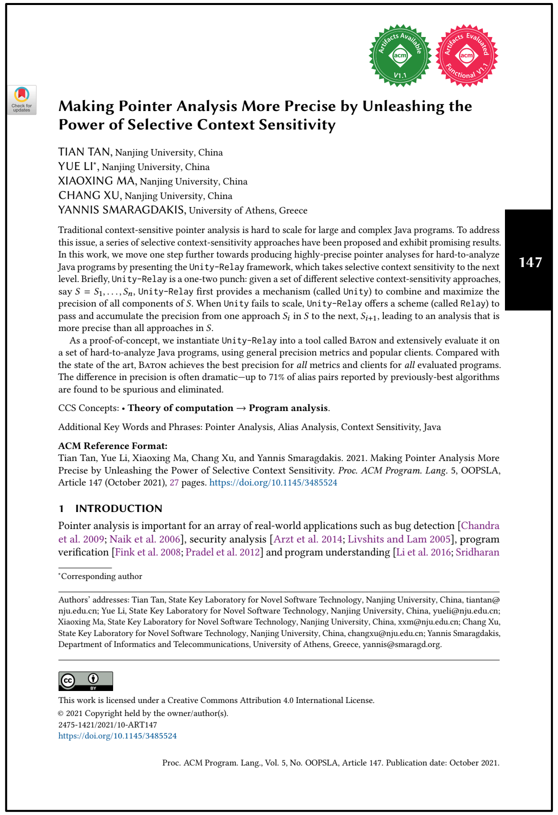
Summary

- Until recently, call-site sensitivity is known as a worse context than object sensitivity
- However context tunneling can make call-site sensitivity analysis

  to a good context flavor
- Cotext tunneling can transform object sensitivity to call-site sensitivity
- Cotext tunneling can improve the precision of call-site sensitivity

Appendix: **Call-Site Sensitivity vs Object Sensitivity**

Object Sensitivity wins!



[1] Ana Milanova, Atanas Rountev, and Barbara G. Ryder. 2005. Parameterized object sensitivity for points-to analysis for Java. ACM Trans. Softw. Eng. Methodol. 14, 1 (January 2005), 1–41. https://doi.org/10.1145/1044834.1044835
[2] Martin Bravenboer and Yannis Smaragdakis. 2009. Strictly declarative specification of sophisticated points-to analyses. SIGPLAN Not. 44, 10 (October 2009), 243–262. https://doi.org/10.1145/1639949.1640108
[3] Tian Tan, Yue Li, Xiaoxing Ma, Chang Xu, and Yannis Smaragdakis. 2021. Making pointer analysis more precise by unleashing the power of selective context sensitivity. Proc. ACM Program. Lang. 5, OOPSLA, Article 147 (October 2021), 27 pages. https://doi.org/10.1145/3485524