

# 软件分析

南京大学

计算机科学与技术系

程序设计语言与

静态分析研究组

李棣 谭添

# Static Program Analysis

## Pointer Analysis Context Sensitivity (I)

Nanjing University

Tian Tan

2021

# Problem of Context-Insensitive Pointer Analysis

```
void main() {  
    Number n1, n2, x, y;  
    n1 = new One(); // o1  
    n2 = new Two(); // o2  
    x = id(n1);  
    y = id(n2);  
    int i = x.get();  
}  
Number id(Number n) {  
    return n;  
}  
  
interface Number {  
    int get(); }  
class One implements Number {  
    public int get() { return 1; }}  
class Two implements Number {  
    public int get() { return 2; }}
```

# Problem of Context-Insensitive Pointer Analysis

```
void main() {  
    Number n1, n2, x, y;  
    n1 = new One(); //  $o_1$   
    n2 = new Two(); //  $o_2$   
    x = id(n1);  
    y = id(n2);  
    → int i = x.get();  
}  
Number id(Number n) {  
    return n;  
}  
  
interface Number {  
    int get(); }  
class One implements Number {  
    public int get() { return 1; }}  
class Two implements Number {  
    public int get() { return 2; }}
```

# Problem of Context-Insensitive Pointer Analysis

```
void main() {  
    Number n1, n2, x, y;  
    n1 = new One(); // o1  
    n2 = new Two(); // o2  
    x = id(n1);  
    y = id(n2);  
    → int i = x.get(); Constant propagation: i = ?  
}  
Number id(Number n) {  
    return n;  
}  
  
interface Number {  
    int get(); }  
class One implements Number {  
    public int get() { return 1; }}  
class Two implements Number {  
    public int get() { return 2; }}
```

# Problem of Context-Insensitive Pointer Analysis

Context insensitivity, `x.get()`:

```
void main() {
    Number n1, n2, x, y;
    n1 = new One(); // o1
    n2 = new Two(); // o2
    x = id(n1);
    y = id(n2);
    → int i = x.get();
}

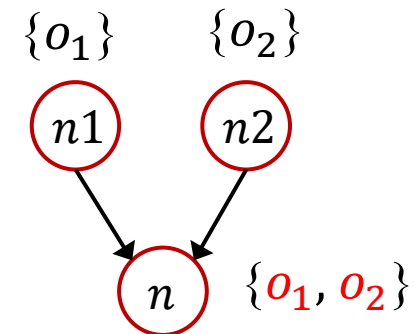
Number id(Number n) {
    return n;
}

interface Number {
    int get(); }
class One implements Number {
    public int get() { return 1; }}
class Two implements Number {
    public int get() { return 2; }}
```

# Problem of Context-Insensitive Pointer Analysis

Context insensitivity, `x.get()`:

```
void main() {  
    Number n1, n2, x, y;  
    n1 = new One(); // o1  
    n2 = new Two(); // o2  
    → x = id(n1);  
    y = id(n2);  
    int i = x.get();  
}  
Number id(Number n) {  
    return n;  
}  
  
interface Number {  
    int get(); }  
class One implements Number {  
    public int get() { return 1; }}  
class Two implements Number {  
    public int get() { return 2; }}
```

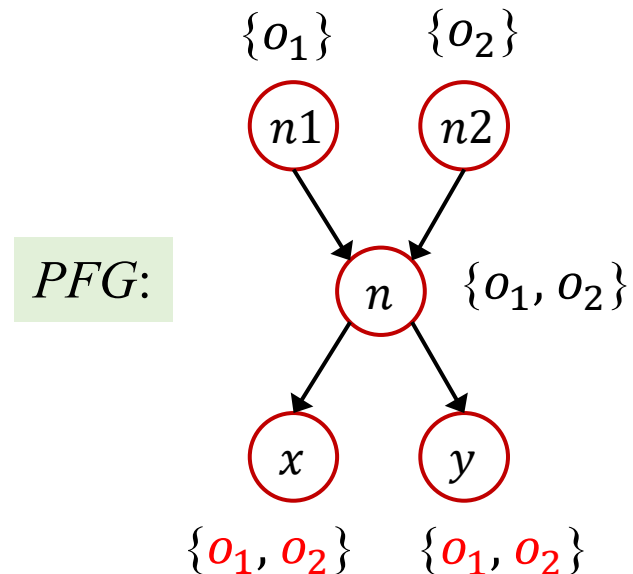


PFG:

# Problem of Context-Insensitive Pointer Analysis

Context insensitivity, `x.get()`:

```
void main() {  
    Number n1, n2, x, y;  
    n1 = new One(); // o1  
    n2 = new Two(); // o2  
    → x = id(n1);  
    y = id(n2);  
    int i = x.get();  
}  
Number id(Number n) {  
    return n;  
}  
  
interface Number {  
    int get(); }  
class One implements Number {  
    public int get() { return 1; }}  
class Two implements Number {  
    public int get() { return 2; }}
```





# Problem of Context-Insensitive Pointer Analysis

```
void main() {  
    Number n1, n2, x, y;  
    n1 = new One(); // o1  
    n2 = new Two(); // o2  
    x = id(n1);  
    y = id(n2);  
    → int i = x.get();  
}  
  
Number id(Number n) {  
    return n;  
}  
  
interface Number {  
    int get();  
}  
class One implements Number {  
    public int get() { return 1; }  
}  
class Two implements Number {  
    public int get() { return 2; }  
}
```

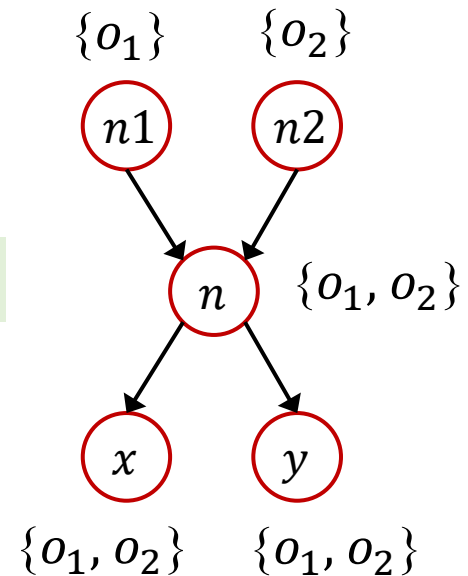
**Context insensitivity, `x.get()`:**

- 2 call targets

Constant propagation

- `i` = NAC

*PFG:*

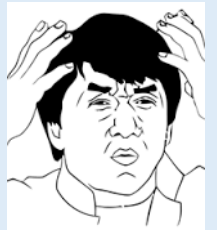


# Problem of Context-Insensitive Pointer Analysis

```
void main() {  
    Number n1, n2, x, y;  
    n1 = new One(); // o1  
    n2 = new Two(); // o2  
    x = id(n1);  
    y = id(n2);  
    → int i = x.get();  
}  
Number id(Number n) {  
    return n;  
}  
  
interface Number {  
    int get();  
}  
class One implements Number {  
    public int get() { return 1; }  
class Two implements Number {  
    public int get() { return 2; }  
}
```

**Context insensitivity, `x.get()`:**

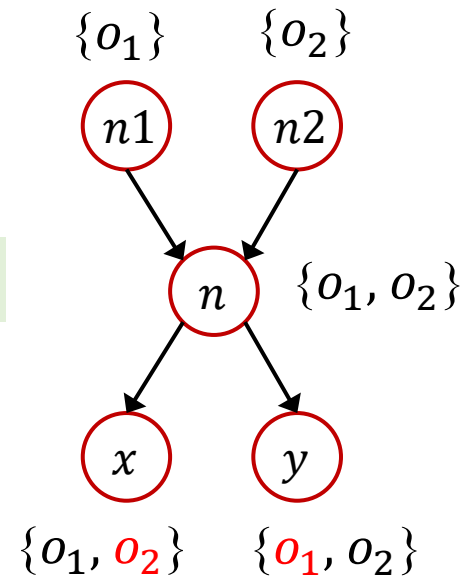
- 2 call targets
- 1 false positive



**Constant propagation**

- `i` = NAC

*PFG:*



# Via Context-Sensitive Pointer Analysis

Context sensitivity, `x.get()`:

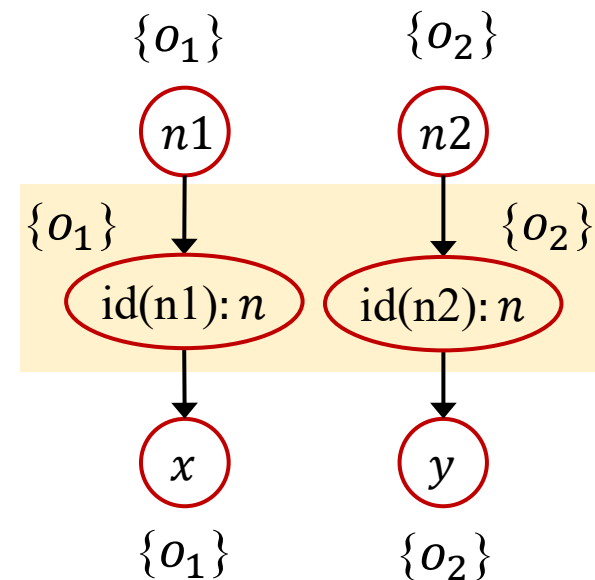
```
void main() {  
    Number n1, n2, x, y;  
    n1 = new One(); // o1  
    n2 = new Two(); // o2  
    x = id(n1);  
    y = id(n2);  
    → int i = x.get();  
}  
Number id(Number n) {  
    return n;  
}  
  
interface Number {  
    int get(); }  
class One implements Number {  
    public int get() { return 1; }}  
class Two implements Number {  
    public int get() { return 2; }}
```

# Via Context-Sensitive Pointer Analysis

Context sensitivity, `x.get()`:

```
void main() {  
    Number n1, n2, x, y;  
    n1 = new One(); // o1  
    n2 = new Two(); // o2  
    → x = id(n1);  
    y = id(n2);  
    int i = x.get();  
}  
  
Number id(Number n) {  
    return n;  
}  
  
interface Number {  
    int get(); }  
class One implements Number {  
    public int get() { return 1; }}  
class Two implements Number {  
    public int get() { return 2; }}
```

*PFG:*



# Via Context-Sensitive Pointer Analysis

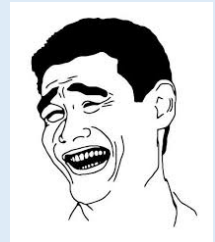
```
void main() {  
    Number n1, n2, x, y;  
    n1 = new One(); // o1  
    n2 = new Two(); // o2  
    x = id(n1);  
    y = id(n2);  
    int i = x.get();  
}  
  
Number id(Number n) {  
    return n;  
}  
  
interface Number {  
    int get();  
}  
class One implements Number {  
    public int get() { return 1; }}  
class Two implements Number {  
    public int get() { return 2; }}
```

**Context sensitivity, `x.get()`:**

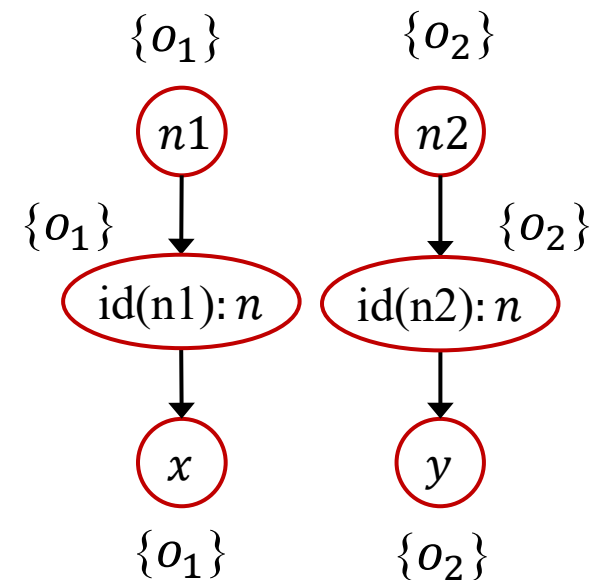
- 1 call targets
- 0 false positive

**Constant propagation**

- `i = 1`



*PFG:*



# Contents

1. Introduction
2. Context Sensitive Pointer Analysis: Rules
3. Context Sensitive Pointer Analysis: Algorithms
4. Context Sensitivity Variants



# Contents

1. **Introduction**
2. Context Sensitive Pointer Analysis: Rules
3. Context Sensitive Pointer Analysis: Algorithms
4. Context Sensitivity Variants

# Imprecision of Context Insensitivity (C.I.)

- In dynamic execution, a method may be called multiple times under **different calling contexts**

```
x = id(n1);  
y = id(n2);  
int i = x.get();  
  
Number id(Number n) {  
    return n;  
}
```



# Imprecision of Context Insensitivity (C.I.)

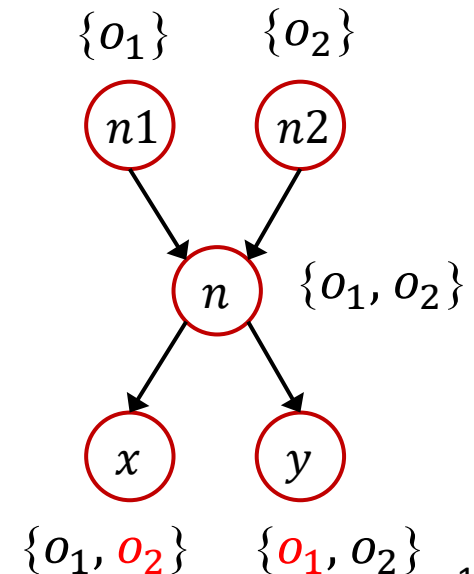
- In dynamic execution, a method may be called multiple times under **different calling contexts**
- Under different calling contexts, the variables of the method may point to **different objects**

```
x = id(n1);  
y = id(n2);  
int i = x.get();  
  
Number id(Number n) {  
    return n;  
}
```

# Imprecision of Context Insensitivity (C.I.)

- In dynamic execution, a method may be called multiple times under **different calling contexts**
- Under different calling contexts, the variables of the method may point to **different objects**
- In C.I. pointer analysis, objects under different contexts are **mixed** and **propagated** to other parts of program (through return values or side-effects), causing **spurious data flows**

```
x = id(n1);  
y = id(n2);  
int i = x.get();  
  
Number id(Number n) {  
    return n;  
}
```



# Context Sensitivity (C.S.)

- Context sensitivity models calling contexts by **distinguishing** different data flows of different contexts to improve precision

# Context Sensitivity (C.S.)

- Context sensitivity models calling contexts by **distinguishing** different data flows of different contexts to improve precision
- The oldest and best-known context sensitivity strategy is **call-site sensitivity** (call-string)
  - Which represents **each context** of a method as **a chain of call sites**, i.e.,
    - a call site of the method,
    - a call site of the caller,
    - a call site of caller of caller, etc.(abstract call stacks in dynamic execution)

# Context Sensitivity (C.S.)

- Context sensitivity models calling contexts by **distinguishing** different data flows of different contexts to improve precision
- The oldest and best-known context sensitivity strategy is **call-site sensitivity** (call-string)
  - Which represents **each context** of a method as **a chain of call sites**, i.e.,  
a call site of the method,  
a call site of the caller,  
a call site of caller of caller, etc.  
(abstract call stacks in dynamic execution)

```
1 x = id(n1);  
2 y = id(n2);  
3 int i = x.get();  
4  
5 Number id(Number n) {  
6     return n;  
7 }
```

In call-site sensitivity, what are the contexts for method `id(Number)`?

# Context Sensitivity (C.S.)

- Context sensitivity models calling contexts by **distinguishing** different data flows of different contexts to improve precision
- The oldest and best-known context sensitivity strategy is **call-site sensitivity** (call-string)
  - Which represents **each context** of a method as **a chain of call sites**, i.e.,
    - a call site of the method,
    - a call site of the caller,
    - a call site of caller of caller, etc.(abstract call stacks in dynamic execution)

```
1 x = id(n1);
2 y = id(n2);
3 int i = x.get();
4
5 Number id(Number n) {
6     return n;
7 }
```

In call-site sensitivity, method `id(Number)` has two contexts: `[1]` and `[2]`

# Context Sensitivity (C.S.)

- Context sensitivity models calling contexts by **distinguishing** different data flows of different contexts to improve precision
- The oldest and best-known context sensitivity strategy is **call-site sensitivity** (call-string)
  - Which represents **each context** of a method as **a chain of call sites**, i.e.,  
a call site of the method,  
a call site of the caller,  
a call site of caller of caller, etc.

You will see other variants of context sensitivity in the next lecture

```
1 x = id(n1);
2 y = id(n2);
3 int i = x.get();
4
5 Number id(Number n) {
6     return n;
7 }
```

In call-site sensitivity, method `id(Number)` has two contexts: `[1]` and `[2]`

# Cloning-Based Context Sensitivity

The most straightforward approach to implement context sensitivity

- In cloning-based context-sensitive pointer analysis, each **method** is **qualified** by one or more **contexts**



# Cloning-Based Context Sensitivity

The most straightforward approach to implement context sensitivity

- In cloning-based context-sensitive pointer analysis, each **method** is **qualified** by one or more **contexts**
- The **variables** are also **qualified** by **contexts** (inherited from the method they are declared in)

# Cloning-Based Context Sensitivity

The most straightforward approach to implement context sensitivity

- In cloning-based context-sensitive pointer analysis, each **method** is **qualified** by one or more **contexts**
- The **variables** are also **qualified** by **contexts** (inherited from the method they are declared in)
- Essentially each method and its variables are cloned, **one clone per context**

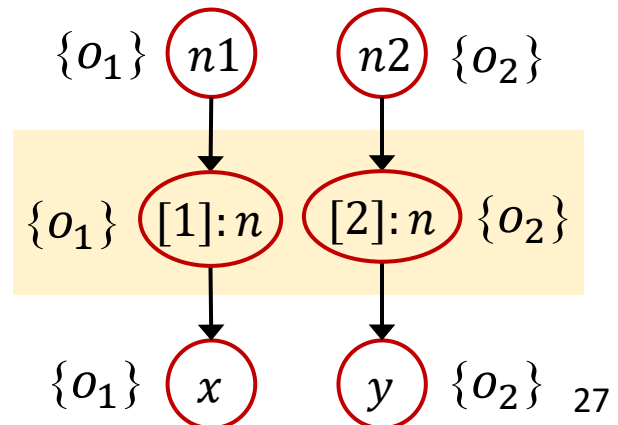
# Cloning-Based Context Sensitivity

The most straightforward approach to implement context sensitivity

- In cloning-based context-sensitive pointer analysis, each **method** is **qualified** by one or more **contexts**
- The **variables** are also **qualified** by **contexts** (inherited from the method they are declared in)
- Essentially each method and its variables are cloned, **one clone per context**

```
1 x = id(n1);
2 y = id(n2);
3 int i = x.get();
4
5 Number id(Number n) {
6     return n;
7 }
```

In call-site sensitivity, method `id(Number)` has two contexts: `[1]` and `[2]`



# Context-Sensitive Heap

- OO programs (e.g., Java) are typically heap-intensive
- In practice, to improve precision, context sensitivity should also be applied to heap abstraction
  - The abstract **objects** are also **qualified** by **contexts** (called **heap contexts**)

# Context-Sensitive Heap

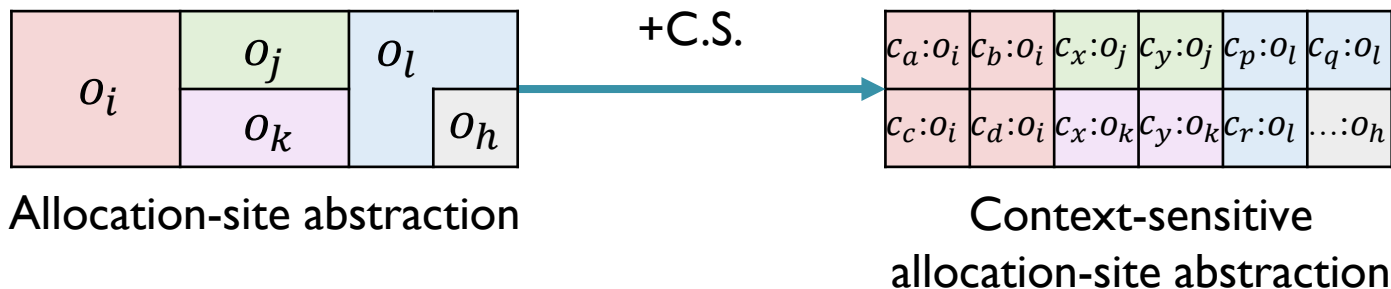
- OO programs (e.g., Java) are typically heap-intensive
- In practice, to improve precision, context sensitivity should also be applied to heap abstraction
  - The abstract **objects** are also **qualified** by **contexts** (called **heap contexts**)

The most common choice is to **inherit contexts from the method** where the object is allocated

# Context-Sensitive Heap

- OO programs (e.g., Java) are typically heap-intensive
- In practice, to improve precision, context sensitivity should also be applied to heap abstraction
  - The abstract **objects** are also **qualified** by **contexts** (called **heap contexts**)

The most common choice is to **inherit contexts from the method** where the object is allocated
  - Context-sensitive heap abstraction provides a **finer-grained** heap model over allocation-site abstraction



# Why Context-Sensitive Heap Improve Precision?

- In dynamic execution, **an allocation site** can create **multiple objects** under different calling **contexts**

```
X newX(...) {  
  X x = new X();  
  ...  
  return x;  
}
```

# Why Context-Sensitive Heap Improve Precision?

- In dynamic execution, **an allocation site** can create **multiple objects** under different calling **contexts**

```
X newX(Y y) {  
  X x = new X();  
  x.f = y;  
  return x;  
}
```

- **Different objects** (allocated by the same site) may be manipulated with different data flows, e.g., **stored different values** to their fields



# Why Context-Sensitive Heap Improve Precision?

- In dynamic execution, **an allocation site** can create **multiple objects** under different calling **contexts**

```
X newX(Y y) {  
  X x = new X();  
  x.f = y;  
  return x;  
}
```

- **Different objects** (allocated by the same site) may be manipulated with different data flows, e.g., **stored different values** to their fields

- In pointer analysis, analyzing such code without heap contexts may **lose precision** by **merging the data flows** of different contexts to one abstract object
- In contrast, **distinguishing** different objects from the same allocation site by heap contexts **gains precision**

# Context-Sensitive Heap: Example

```
1  n1 = new One();
2  n2 = new Two();
3  x1 = newX(n1);
4  x2 = newX(n2);
5  n = x1.f;
6
7  X newX(Number p) {
8      X x = new X();
9      x.f = p;
10     return x;
11 }
12 class X {
13     Number f;
14 }
```

# Context-Sensitive Heap: Example

C.S., no C.S. heap

Variable	Object
n1	$o_1$
n2	$o_2$

```
1  n1 = new One();
2  n2 = new Two();
3  x1 = newX(n1);
4  x2 = newX(n2);
5  n = x1.f;
6
7  X newX(Number p) {
8      X x = new X();
9      x.f = p;
10     return x;
11 }
12 class X {
13     Number f;
14 }
```

# Context-Sensitive Heap: Example

C.S., no C.S. heap

Variable	Object
n1	$o_1$
n2	$o_2$
3:p	$o_1$

```
1  n1 = new One();
2  n2 = new Two();
→ 3  x1 = newX(n1);
4  x2 = newX(n2);
5  n = x1.f;
6
→ 7  X newX(Number p) {
8      X x = new X();
9      x.f = p;
10     return x;
11 }
12 class X {
13     Number f;
14 }
```

# Context-Sensitive Heap: Example

C.S., no C.S. heap

Variable	Object
n1	$o_1$
n2	$o_2$
3:p	$o_1$

?

```
1  n1 = new One();
2  n2 = new Two();
→ 3  x1 = newX(n1);
4  x2 = newX(n2);
5  n = x1.f;
6
7  X newX(Number p) {
→ 8    X x = new X();
9    x.f = p;
10   return x;
11 }
12 class X {
13   Number f;
14 }
```

# Context-Sensitive Heap: Example

C.S., no C.S. heap

Variable	Object
n1	$o_1$
n2	$o_2$
3:p	$o_1$
3:x	$o_8$

```
1  n1 = new One();
2  n2 = new Two();
→ 3  x1 = newX(n1);
4  x2 = newX(n2);
5  n = x1.f;
6
7  X newX(Number p) {
→ 8      X x = new X();
9      x.f = p;
10     return x;
11 }
12 class X {
13     Number f;
14 }
```

# Context-Sensitive Heap: Example

```
1  n1 = new One();
2  n2 = new Two();
→ 3  x1 = newX(n1);
4  x2 = newX(n2);
5  n = x1.f;
6
7  X newX(Number p) {
8      X x = new X();
→ 9      x.f = p;
→10     return x;
11 }
12 class X {
13     Number f;
14 }
```

C.S., no C.S. heap

Variable	Object
n1	$o_1$
n2	$o_2$
3:p	$o_1$
3:x	$o_8$
x1	$o_8$
Field	Object
$o_8.f$	$o_1$

# Context-Sensitive Heap: Example

```
1  n1 = new One();
2  n2 = new Two();
3  x1 = newX(n1);
4  x2 = newX(n2);
5  n = x1.f;
6
7  X newX(Number p) {
8      X x = new X();
9      x.f = p;
10     return x;
11 }
12 class X {
13     Number f;
14 }
```

C.S., no C.S. heap

Variable	Object
n1	$o_1$
n2	$o_2$
3:p	$o_1$
3:x	$o_8$
x1	$o_8$
4:p	$o_2$
4:x	$o_8$
Field	Object
$o_8.f$	$o_1$



# Context-Sensitive Heap: Example

```
1  n1 = new One();
2  n2 = new Two();
3  x1 = newX(n1);
4  x2 = newX(n2);
5  n = x1.f;
6
7  X newX(Number p) {
8      X x = new X();
9      x.f = p;
10     return x;
11 }
12 class X {
13     Number f;
14 }
```

C.S., no C.S. heap

Variable	Object
n1	$o_1$
n2	$o_2$
3:p	$o_1$
3:x	$o_8$
x1	$o_8$
4:p	$o_2$
4:x	$o_8$
Field	Object
$o_8.f$	$o_1$



# Context-Sensitive Heap: Example

```
1  n1 = new One();
2  n2 = new Two();
3  x1 = newX(n1);
4  x2 = newX(n2);
5  n = x1.f;
6
7  X newX(Number p) {
8      X x = new X();
9      x.f = p;
10     return x;
11 }
12 class X {
13     Number f;
14 }
```

C.S., no C.S. heap

Variable	Object
n1	$o_1$
n2	$o_2$
3:p	$o_1$
3:x	$o_8$
x1	$o_8$
4:p	$o_2$
4:x	$o_8$
x2	$o_8$
Field	Object
$o_8.f$	$o_1, o_2$

# Context-Sensitive Heap: Example

```
1  n1 = new One();
2  n2 = new Two();
3  x1 = newX(n1);
4  x2 = newX(n2);
→ 5  n = x1.f;
6
7  X newX(Number p) {
8      X x = new X();
9      x.f = p;
10     return x;
11 }
12 class X {
13     Number f;
14 }
```

C.S., no C.S. heap

Variable	Object
n1	$o_1$
n2	$o_2$
3:p	$o_1$
3:x	$o_8$
x1	$o_8$
4:p	$o_2$
4:x	$o_8$
x2	$o_8$
n	?
Field	Object
$o_8.f$	$o_1, o_2$

# Context-Sensitive Heap: Example

```
1  n1 = new One();
2  n2 = new Two();
3  x1 = newX(n1);
4  x2 = newX(n2);
→ 5  n = x1.f;
6
7  X newX(Number p) {
8      X x = new X();
9      x.f = p;
10     return x;
11 }
12 class X {
13     Number f;
14 }
```

C.S., no C.S. heap

Variable	Object
n1	$o_1$
n2	$o_2$
3:p	$o_1$
3:x	$o_8$
x1	$o_8$
4:p	$o_2$
4:x	$o_8$
x2	$o_8$
n	$o_1, o_2$
Field	Object
$o_8.f$	$o_1, o_2$

# Context-Sensitive Heap: Example

```
1  n1 = new One();
2  n2 = new Two();
3  x1 = newX(n1);
4  x2 = newX(n2);
5  n = x1.f;
6
7  X newX(Number p) {
8      X x = new X();
9      x.f = p;
10     return x;
11 }
12 class X {
13     Number f;
14 }
```

C.S., no C.S. heap

Variable	Object
n1	$o_1$
n2	$o_2$
3:p	$o_1$
3:x	$o_8$
x1	$o_8$
4:p	$o_2$
4:x	$o_8$
x2	$o_8$
n	$o_1, o_2$
Field	Object
$o_8.f$	$o_1, o_2$

Spurious data flow  
due to lack of C.S. heap

# Context-Sensitive Heap: Example

```

1  n1 = new One();
2  n2 = new Two();
→ 3  x1 = newX(n1);
4  x2 = newX(n2);
5  n = x1.f;
6
→ 7  X newX(Number p) {
8      X x = new X();
9      x.f = p;
10     return x;
11 }
12 class X {
13     Number f;
14 }

```

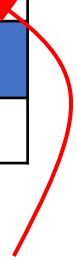
C.S., no C.S. heap

Variable	Object
n1	$o_1$
n2	$o_2$
3:p	$o_1$
3:x	$o_8$
x1	$o_8$
4:p	$o_2$
4:x	$o_8$
x2	$o_8$
n	$o_1, o_2$
Field	Object
$o_8.f$	$o_1, o_2$

C.S. + C.S heap

Variable	Object
n1	$o_1$
n2	$o_2$
3:p	$o_1$

Spurious data flow  
due to lack of C.S. heap



# Context-Sensitive Heap: Example

```

1  n1 = new One();
2  n2 = new Two();
→ 3  x1 = newX(n1);
4  x2 = newX(n2);
5  n = x1.f;
6
7  X newX(Number p) {
→ 8    X x = new X();
9    x.f = p;
10   return x;
11 }
12 class X {
13   Number f;
14 }
    
```

C.S., no C.S. heap

Variable	Object
n1	$o_1$
n2	$o_2$
3:p	$o_1$
3:x	$o_8$
x1	$o_8$
4:p	$o_2$
4:x	$o_8$
x2	$o_8$
n	$o_1, o_2$
Field	Object
$o_8.f$	$o_1, o_2$

Spurious data flow  
due to lack of C.S. heap

C.S. + C.S. heap

Variable	Object
n1	$o_1$
n2	$o_2$
3:p	$o_1$

?

# Context-Sensitive Heap: Example

```

1  n1 = new One();
2  n2 = new Two();
3  x1 = newX(n1);
4  x2 = newX(n2);
5  n = x1.f;
6
7  X newX(Number p) {
8    X x = new X();
9    x.f = p;
10   return x;
11 }
12 class X {
13   Number f;
14 }

```

C.S., no C.S. heap

Variable	Object
n1	$o_1$
n2	$o_2$
3:p	$o_1$
3:x	$o_8$
x1	$o_8$
4:p	$o_2$
4:x	$o_8$
x2	$o_8$
n	$o_1, o_2$
Field	Object
$o_8.f$	$o_1, o_2$

C.S. + C.S heap

Variable	Object
n1	$o_1$
n2	$o_2$
3:p	$o_1$
3:x	$3:o_8$

Object with heap context

Spurious data flow  
due to lack of C.S. heap



# Context-Sensitive Heap: Example

```

1  n1 = new One();
2  n2 = new Two();
3  x1 = newX(n1);
4  x2 = newX(n2);
5  n = x1.f;
6
7  X newX(Number p) {
8      X x = new X();
9      x.f = p;
10     return x;
11 }
12 class X {
13     Number f;
14 }

```

C.S., no C.S. heap

Variable	Object
n1	$o_1$
n2	$o_2$
3:p	$o_1$
3:x	$o_8$
x1	$o_8$
4:p	$o_2$
4:x	$o_8$
x2	$o_8$
n	$o_1, o_2$
Field	Object
$o_8.f$	$o_1, o_2$

C.S. + C.S. heap

Variable	Object
n1	$o_1$
n2	$o_2$
3:p	$o_1$
3:x	3: $o_8$

?

Spurious data flow  
due to lack of C.S. heap

# Context-Sensitive Heap: Example

```

1  n1 = new One();
2  n2 = new Two();
→ 3  x1 = newX(n1);
4  x2 = newX(n2);
5  n = x1.f;
6
7  X newX(Number p) {
8      X x = new X();
→ 9      x.f = p;
→10     return x;
11 }
12 class X {
13     Number f;
14 }

```

C.S., no C.S. heap

Variable	Object
n1	$o_1$
n2	$o_2$
3:p	$o_1$
3:x	$o_8$
x1	$o_8$
4:p	$o_2$
4:x	$o_8$
x2	$o_8$
n	$o_1, o_2$
Field	Object
$o_8.f$	$o_1, o_2$

C.S. + C.S heap

Variable	Object
n1	$o_1$
n2	$o_2$
3:p	$o_1$
3:x	3: $o_8$
x1	3: $o_8$
Field	Object
3: $o_8.f$	$o_1$

Spurious data flow  
due to lack of C.S. heap

# Context-Sensitive Heap: Example

```

1  n1 = new One();
2  n2 = new Two();
3  x1 = newX(n1);
4  x2 = newX(n2);
5  n = x1.f;
6
7  X newX(Number p) {
8    X x = new X();
9    x.f = p;
10   return x;
11 }
12 class X {
13   Number f;
14 }

```

C.S., no C.S. heap

Variable	Object
n1	$o_1$
n2	$o_2$
3:p	$o_1$
3:x	$o_8$
x1	$o_8$
4:p	$o_2$
4:x	$o_8$
x2	$o_8$
n	$o_1, o_2$
Field	Object
$o_8.f$	$o_1, o_2$

C.S. + C.S. heap

Variable	Object
n1	$o_1$
n2	$o_2$
3:p	$o_1$
3:x	3: $o_8$
x1	3: $o_8$
4:p	$o_2$
4:x	4: $o_8$
Field	Object
3: $o_8.f$	$o_1$

Spurious data flow  
due to lack of C.S. heap

# Context-Sensitive Heap: Example

```

1  n1 = new One();
2  n2 = new Two();
3  x1 = newX(n1);
4  x2 = newX(n2);
5  n = x1.f;
6
7  X newX(Number p) {
8      X x = new X();
9      x.f = p;
10     return x;
11 }
12 class X {
13     Number f;
14 }

```

C.S., no C.S. heap

Variable	Object
n1	$o_1$
n2	$o_2$
3:p	$o_1$
3:x	$o_8$
x1	$o_8$
4:p	$o_2$
4:x	$o_8$
x2	$o_8$
n	$o_1, o_2$
Field	Object
$o_8.f$	$o_1, o_2$

C.S. + C.S heap

Variable	Object
n1	$o_1$
n2	$o_2$
3:p	$o_1$
3:x	3: $o_8$
x1	3: $o_8$
4:p	$o_2$
4:x	4: $o_8$
x2	4: $o_8$
Field	Object
3: $o_8.f$	$o_1$
4: $o_8.f$	$o_2$

Spurious data flow  
due to lack of C.S. heap

# Context-Sensitive Heap: Example

```

1  n1 = new One();
2  n2 = new Two();
3  x1 = newX(n1);
4  x2 = newX(n2);
5  n = x1.f;
6
7  X newX(Number p) {
8      X x = new X();
9      x.f = p;
10     return x;
11 }
12 class X {
13     Number f;
14 }

```

C.S., no C.S. heap

Variable	Object
n1	$o_1$
n2	$o_2$
3:p	$o_1$
3:x	$o_8$
x1	$o_8$
4:p	$o_2$
4:x	$o_8$
x2	$o_8$
n	$o_1, o_2$
Field	Object
$o_8.f$	$o_1, o_2$

Spurious data flow  
due to lack of C.S. heap

C.S. + C.S heap

Variable	Object
n1	$o_1$
n2	$o_2$
3:p	$o_1$
3:x	3: $o_8$
x1	3: $o_8$
4:p	$o_2$
4:x	4: $o_8$
x2	4: $o_8$
n	?
Field	Object
3: $o_8.f$	$o_1$
4: $o_8.f$	$o_2$

# Context-Sensitive Heap: Example

```

1  n1 = new One();
2  n2 = new Two();
3  x1 = newX(n1);
4  x2 = newX(n2);
5  n = x1.f;
6
7  X newX(Number p) {
8      X x = new X();
9      x.f = p;
10     return x;
11 }
12 class X {
13     Number f;
14 }

```

C.S., no C.S. heap

Variable	Object
n1	$o_1$
n2	$o_2$
3:p	$o_1$
3:x	$o_8$
x1	$o_8$
4:p	$o_2$
4:x	$o_8$
x2	$o_8$
n	$o_1, o_2$
Field	Object
$o_8.f$	$o_1, o_2$

C.S. + C.S heap

Variable	Object
n1	$o_1$
n2	$o_2$
3:p	$o_1$
3:x	3: $o_8$
x1	3: $o_8$
4:p	$o_2$
4:x	4: $o_8$
x2	4: $o_8$
n	$o_1$
Field	Object
3: $o_8.f$	$o_1$
4: $o_8.f$	$o_2$

Spurious data flow  
due to lack of C.S. heap

# Context-Sensitive Heap: Example

```

1  n1 = new One();
2  n2 = new Two();
3  x1 = newX(n1);
4  x2 = newX(n2);
5  n = x1.f;
6
7  X newX(Number p) {
8      X x = new X();
9      x.f = p;
10     return x;
11 }
12 class X {
13     Number f;
14 }
    
```

C.S., no C.S. heap

Variable	Object
n1	$o_1$
n2	$o_2$
3:p	$o_1$
3:x	$o_8$
x1	$o_8$
4:p	$o_2$
4:x	$o_8$
x2	$o_8$
n	$o_1, o_2$
Field	Object
$o_8.f$	$o_1, o_2$

Spurious data flow  
due to lack of C.S. heap

C.S. + C.S. heap

Variable	Object
n1	$o_1$
n2	$o_2$
3:p	$o_1$
3:x	3: $o_8$
x1	3: $o_8$
4:p	$o_2$
4:x	4: $o_8$
x2	4: $o_8$
n	$o_1$
Field	Object
3: $o_8.f$	$o_1$
4: $o_8.f$	$o_2$

Context-sensitive heap  
improves precision

# Context-Sensitive Heap: Example

```

1  n1 = new One();
2  n2 = new Two();
3  x1 = newX(n1);
4  x2 = newX(n2);
5  n = x1.f;
6
7  X newX(Number p) {
8      X x = new X();
9      x.f = p;
10     return x;
11 }
12 class X {
13     Number f;
14 }

```

C.I. + C.S. heap

Variable	Object
n1	$o_1$
n2	$o_2$
p	$o_1, o_2$
x	$3:o_8, 4:o_8$
x1	$3:o_8, 4:o_8$
x2	$3:o_8, 4:o_8$
n	$o_1, o_2$
Field	Object
$3:o_8.f$	$o_1, o_2$
$4:o_8.f$	$o_1, o_2$

Without C.S., C.S. heap  
cannot improve precision

C.S. + C.S heap

Variable	Object
n1	$o_1$
n2	$o_2$
3:p	$o_1$
3:x	$3:o_8$
x1	$3:o_8$
4:p	$o_2$
4:x	$4:o_8$
x2	$4:o_8$
n	$o_1$
Field	Object
$3:o_8.f$	$o_1$
$4:o_8.f$	$o_2$

Context-sensitive heap  
improves precision



# Contents

1. Introduction
- 2. Context Sensitive Pointer Analysis: Rules**
3. Context Sensitive Pointer Analysis: Algorithms
4. Context Sensitivity Variants

# Domain and Notations

In context-sensitive analysis,  
program elements are qualified by **contexts**

Context:  $c, c', c'' \in C$

Context-sensitive methods:  $c:m \in C \times M$

Context-sensitive variables:  $c:x, c':y \in C \times V$

Context-sensitive objects:  $c:o_i, c':o_j \in C \times O$

Fields:  $f, g \in F$

Instance fields:  $c:o_i.f, c':o_j.g \in C \times O \times F$

Context-sensitive pointers:  $\text{CSPtr} = (C \times V) \cup (C \times O \times F)$

Points-to relations:  $pt : \text{CSPtr} \rightarrow \mathcal{P}(C \times O)$

# Rules

Kind	Statement	Rule (under context $c$ )
New	$i: x = \text{new } T()$	$\overline{c: o_i \in pt(c: x)}$
Assign	$x = y$	$\frac{c': o_i \in pt(c: y)}{c': o_i \in pt(c: x)}$
Store	$x.f = y$	$\frac{c': o_i \in pt(c: x), c'': o_j \in pt(c: y)}{c'': o_j \in pt(c': o_i.f)}$
Load	$y = x.f$	$\frac{c': o_i \in pt(c: x), c'': o_j \in pt(c': o_i.f)}{c'': o_j \in pt(c: y)}$

# Rules

Kind	Statement	Rule
New	$i: x = \text{new } T()$	$\frac{}{\Box o_i \in pt(\Box x)}$
Assign	$x = y$	$\frac{\Box o_i \in pt(\Box y)}{\Box o_i \in pt(\Box x)}$
Store	$x.f = y$	$\frac{\Box o_i \in pt(\Box x), \Box o_j \in pt(\Box y)}{\Box o_j \in pt(\Box o_i.f)}$
Load	$y = x.f$	$\frac{\Box o_i \in pt(\Box x), \Box o_j \in pt(\Box o_i.f)}{\Box o_j \in pt(\Box y)}$

# Rules

Kind	Statement	Rule ( <b>under context <math>c</math></b> )
New	$i: x = \text{new } T()$	$\overline{c: o_i \in pt(c: x)}$
Assign	$x = y$	$\frac{c': o_i \in pt(c: y)}{c': o_i \in pt(c: x)}$
Store	$x.f = y$	$\frac{c': o_i \in pt(c: x), c'': o_j \in pt(c: y)}{c'': o_j \in pt(c': o_i.f)}$
Load	$y = x.f$	$\frac{c': o_i \in pt(c: x), c'': o_j \in pt(c': o_i.f)}{c'': o_j \in pt(c: y)}$

# Rule: New

$$\overline{c: o_i \in pt(c: x)}$$

→ Conclusion

$c: o_i$

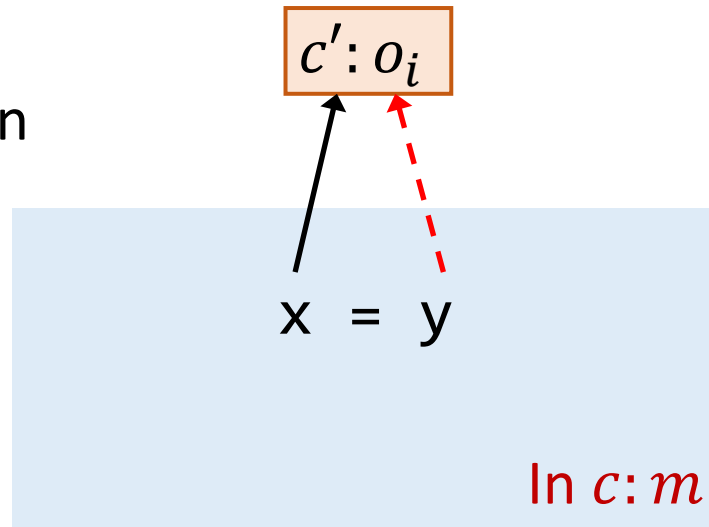
$i: x = \text{new } T()$

In  $c: m$

# Rule: Assign

$$\frac{c': o_i \in pt(c: y)}{c': o_i \in pt(c: x)}$$

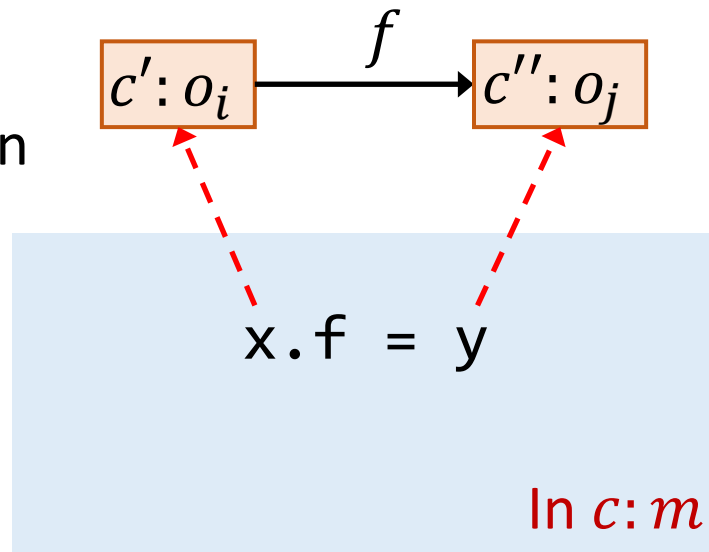
$\dashrightarrow$  Premises  
 $\longrightarrow$  Conclusion



# Rule: Store

$$\frac{c': o_i \in pt(c: x), c'': o_j \in pt(c: y)}{c'': o_j \in pt(c': o_i . f)}$$

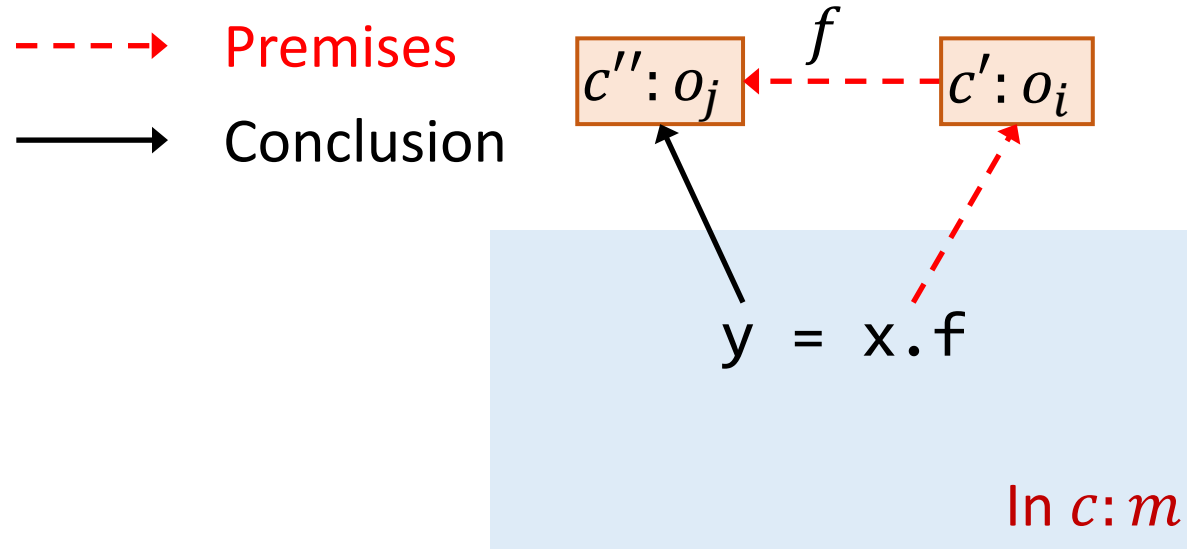
$\dashrightarrow$  Premises  
 $\longrightarrow$  Conclusion





# Rule: Load

$$\frac{c':o_i \in pt(c:x), c'':o_j \in pt(c':o_i.f)}{c'':o_j \in pt(c:y)}$$



# Rules

-----> Premises  
 -----> Conclusion

Kind	Rule	Illustration
New	$\frac{}{c: o_i \in pt(c: x)}$	<p><math>c: o_i</math>  <math>i: x = \text{new } T()</math></p>
Assign	$\frac{c': o_i \in pt(c: y)}{c': o_i \in pt(c: x)}$	<p><math>c': o_i</math>  <math>x = y</math></p>
Store	$\frac{c': o_i \in pt(c: x), c'': o_j \in pt(c: y)}{c'': o_j \in pt(c': o_i. f)}$	<p><math>c': o_i \xrightarrow{f} c'': o_j</math>  <math>x.f = y</math></p>
Load	$\frac{c': o_i \in pt(c: x), c'': o_j \in pt(c': o_i. f)}{c'': o_j \in pt(c: y)}$	<p><math>c'': o_j \xleftarrow{f} c': o_i</math>  <math>y = x.f</math></p>

# Rule: Call

Kind	Statement	Rule (under context $c$ )
Call	$l: r = x.k(a_1, \dots, a_n)$	$  \begin{array}{c}  c': o_i \in pt(c: x), \\  m = \text{Dispatch}(o_i, k), c^t = \text{Select}(c, l, c': o_i) \\  c'': o_u \in pt(c: a_j), 1 \leq j \leq n \\  c''': o_v \in pt(c^t: m_{ret}) \\  \hline  c': o_i \in pt(c^t: m_{this}) \\  c'': o_u \in pt(c^t: m_{pj}), 1 \leq j \leq n \\  c''': o_v \in pt(c: r)  \end{array}  $

# Rule: Call

Kind	Statement	Rule (under context $c$ )
Call	$l: r = x.k(a_1, \dots, a_n)$	$  \begin{array}{c}  \longrightarrow c': o_i \in pt(\textcolor{red}{c}: x), \\  m = \textcolor{red}{Dispatch}(o_i, k), c^t = \textcolor{red}{Select}(c, l, c': o_i) \\  c'': o_u \in pt(c: a_j), 1 \leq j \leq n \\  c''': o_v \in pt(c^t: m_{ret}) \\  \hline  c': o_i \in pt(c^t: m_{this}) \\  c'': o_u \in pt(c^t: m_{pj}), 1 \leq j \leq n \\  c''': o_v \in pt(c: r)  \end{array}  $

- $\textcolor{red}{Dispatch}(o_i, k)$ : resolves the virtual dispatch of  $k$  on  $o_i$  to a **target method** (based on type of  $o_i$ )

# Rule: Call

Caller context:  $c$   
Callee context:  $c^t$

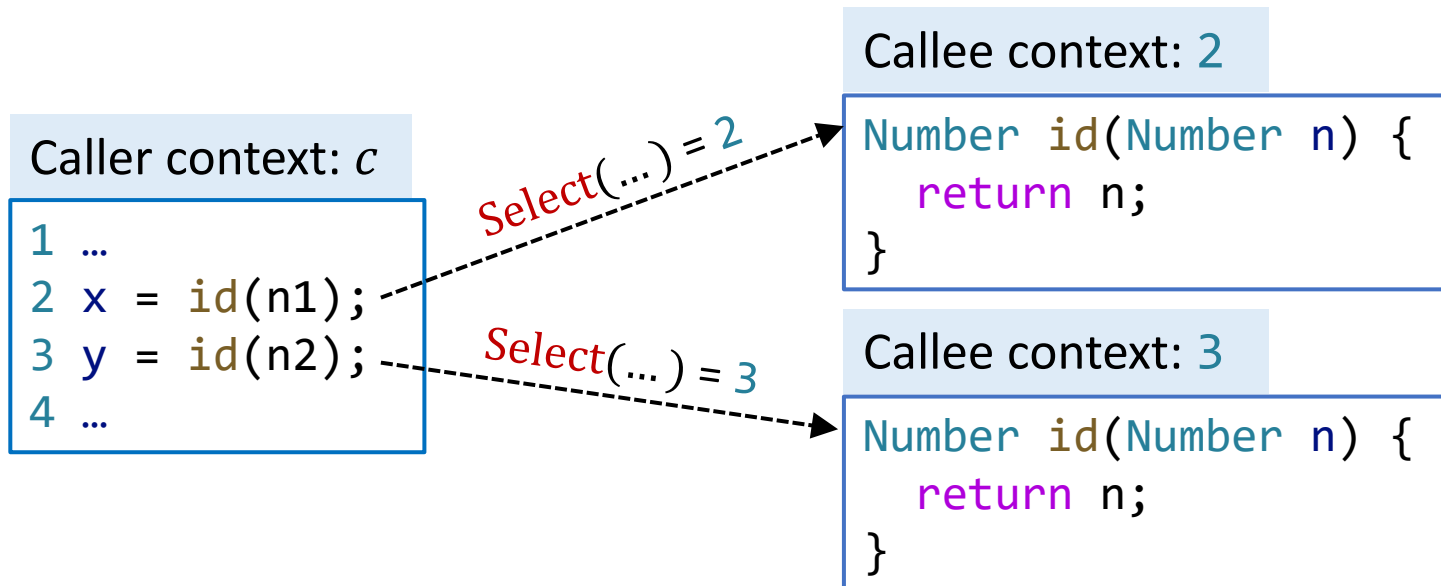
Kind	Statement	Rule (under context $c$ )
Call	$l: r = x.k(a_1, \dots, a_n)$	$  \begin{array}{c}  \longrightarrow c': o_i \in pt(c: x), \\  m = \text{Dispatch}(o_i, k), c^t = \text{Select}(c, l, c': o_i) \\  c'': o_u \in pt(c: a_j), 1 \leq j \leq n \\  c''': o_v \in pt(c^t: m_{ret}) \\  \hline  c': o_i \in pt(c^t: m_{this}) \\  c'': o_u \in pt(c^t: m_{pj}), 1 \leq j \leq n \\  c''': o_v \in pt(c: r)  \end{array}  $

- **Dispatch**( $o_i, k$ ): resolves the virtual dispatch of  $k$  on  $o_i$  to a **target method** (based on type of  $o_i$ )
- **Select**( $c, l, c': o_i$ ): selects **context** for target method  $m$ , based on the information available at call site  $l$

# Rule: Call

Caller context:  $c$   
Callee context:  $c^t$

Kind	Statement	Rule (under context $c$ )
Call	$l: r = x.k(a_1, \dots, a_n)$	$  \begin{array}{c}  \longrightarrow c': o_i \in pt(c: x), \\  m = \text{Dispatch}(o_i, k), c^t = \text{Select}(c, l, c': o_i) \\  c'': o_u \in pt(c: a_j), 1 \leq j \leq n \\  c''': o_v \in pt(c^t: m_{ret}) \\  \hline  c': o_i \in pt(c^t: m_{this}) \\  c'': o_u \in pt(c^t: m_{pj}), 1 \leq j \leq n \\  c''': o_v \in pt(c: r)  \end{array}  $



# Rule: Call

Caller context:  $c$   
Callee context:  $c^t$

Kind	Statement	Rule (under context $c$ )
Call	$l: r = x.k(a_1, \dots, a_n)$	$  \begin{array}{l}  \longrightarrow c': o_i \in pt(\textcolor{red}{c}: x), \\  m = \textcolor{red}{Dispatch}(o_i, k), c^t = \textcolor{red}{Select}(c, l, c': o_i) \\  c'': o_u \in pt(c: a_j), 1 \leq j \leq n \\  c''': o_v \in pt(c^t: m_{ret}) \\  \hline  \longrightarrow c': o_i \in pt(\textcolor{red}{c}^t: m_{this}) \\  c'': o_u \in pt(c^t: m_{pj}), 1 \leq j \leq n \\  c''': o_v \in pt(c: r)  \end{array}  $

- $\textcolor{red}{Dispatch}(o_i, k)$ : resolves the virtual dispatch of  $k$  on  $o_i$  to a **target method** (based on type of  $o_i$ )
- $\textcolor{red}{Select}(c, l, c': o_i)$ : selects **context** for target method  $m$ , based on the information available at call site  $l$
- $c^t: m_{this}$ : **this variable** of  $c^t: m$

# Rule: Call

Caller context:  $c$   
Callee context:  $c^t$

Kind	Statement	Rule (under context $c$ )
Call	$l : r = x.k(a_1, \dots, a_n)$	$  \begin{array}{c}  c' : o_i \in pt(c : x), \\  m = \text{Dispatch}(o_i, k), c^t = \text{Select}(c, l, c' : o_i) \\  \longrightarrow c'' : o_u \in pt(\textcolor{red}{c} : a_j), 1 \leq j \leq n \\  \quad c''' : o_v \in pt(c^t : m_{ret}) \\  \hline  c' : o_i \in pt(c^t : m_{this}) \\  \longrightarrow c'' : o_u \in pt(\textcolor{red}{c}^t : m_{pj}), 1 \leq j \leq n \\  \quad c''' : o_v \in pt(c : r)  \end{array}  $

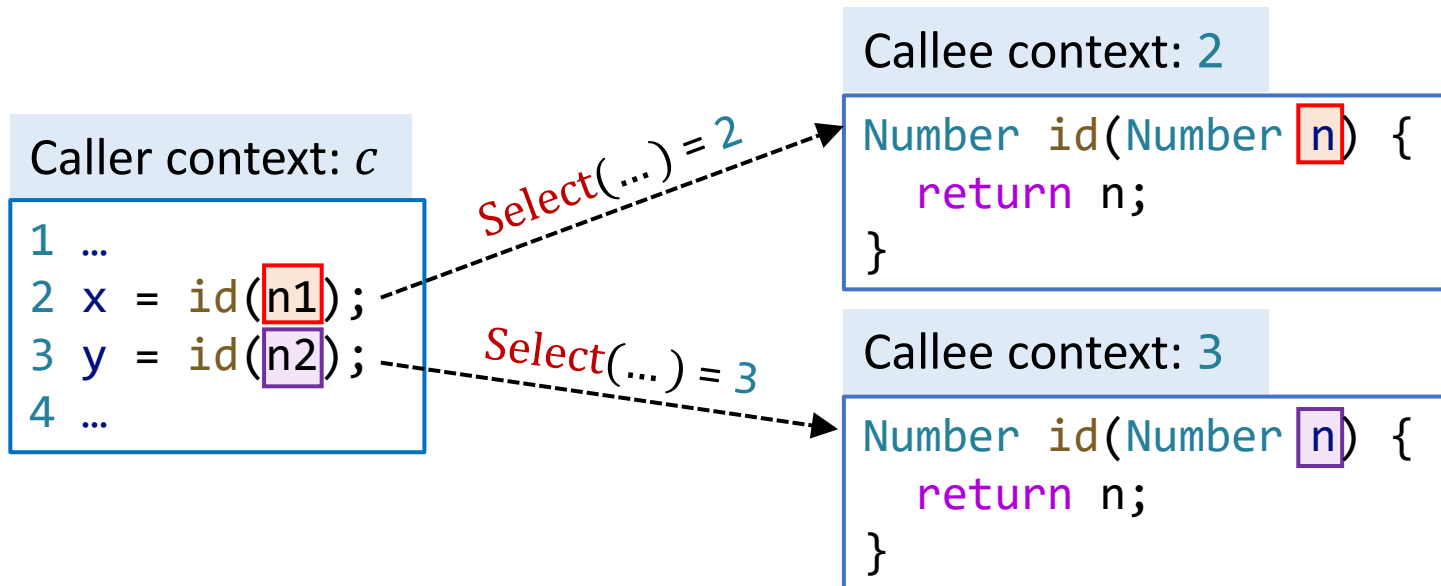
- $\text{Dispatch}(o_i, k)$ : resolves the virtual dispatch of  $k$  on  $o_i$  to a **target method** (based on type of  $o_i$ )
- $\text{Select}(c, l, c' : o_i)$ : selects **context** for target method  $m$ , based on the information available at call site  $l$
- $c^t : m_{this}$ : **this variable** of  $c^t : m$
- $c^t : m_{pj}$ : the  **$j$ -th parameter** of  $c^t : m$



# Rule: Call

Caller context:  $c$   
Callee context:  $c^t$

Kind	Statement	Rule (under context $c$ )
Call	$l : r = x.k(a_1, \dots, a_n)$	$  \begin{array}{c}  c' : o_i \in pt(c : x), \\  m = \text{Dispatch}(o_i, k), c^t = \text{Select}(c, l, c' : o_i) \\  \longrightarrow c'' : o_u \in pt(\text{c} : aj), 1 \leq j \leq n \\  c''' : o_v \in pt(c^t : m_{ret}) \\  \hline  c' : o_i \in pt(c^t : m_{this}) \\  \longrightarrow c'' : o_u \in pt(\text{c}^t : m_{pj}), 1 \leq j \leq n \\  c''' : o_v \in pt(c : r)  \end{array}  $



# Rule: Call

Caller context:  $c$   
Callee context:  $c^t$

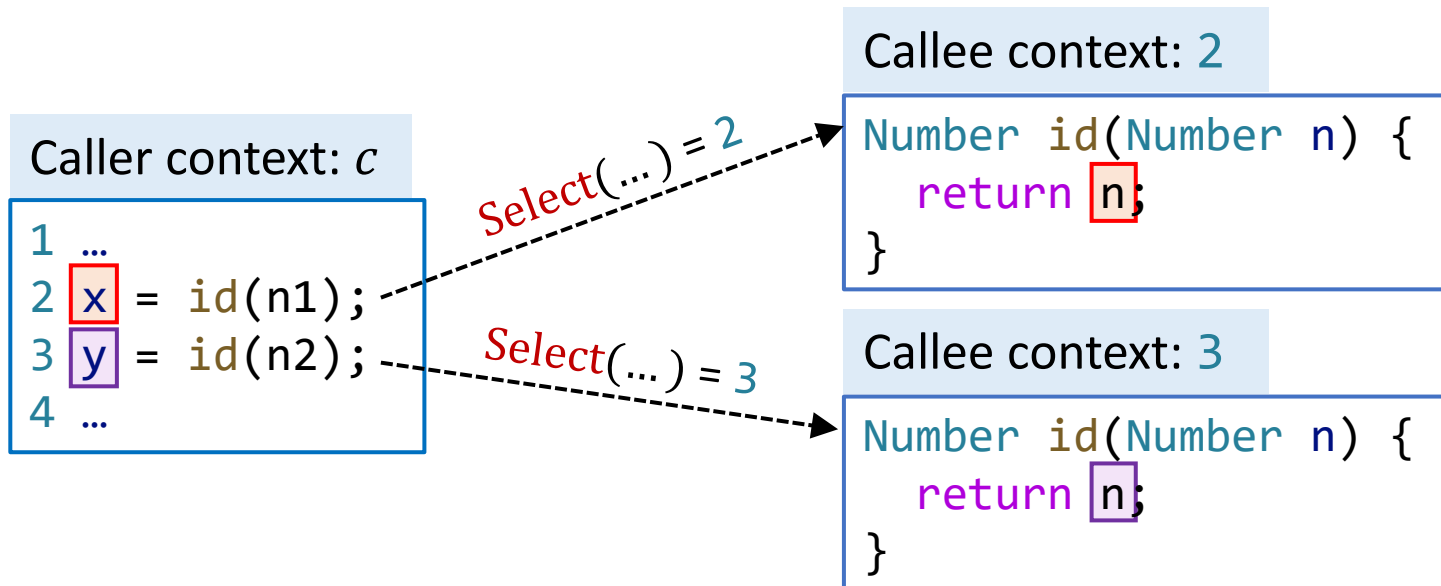
Kind	Statement	Rule (under context $c$ )
Call	$l : r = x.k(a_1, \dots, a_n)$	$  \begin{array}{c}  c' : o_i \in pt(c : x), \\  m = \text{Dispatch}(o_i, k), c^t = \text{Select}(c, l, c' : o_i) \\  c'' : o_u \in pt(c : a_j), 1 \leq j \leq n \\  \xrightarrow{\quad} c''' : o_v \in pt(\textcolor{red}{c}^t : m_{ret}) \\  \hline  c' : o_i \in pt(c^t : m_{this}) \\  c'' : o_u \in pt(c^t : m_{pj}), 1 \leq j \leq n \\  \xrightarrow{\quad} c''' : o_v \in pt(\textcolor{red}{c} : r)  \end{array}  $

- $\text{Dispatch}(o_i, k)$ : resolves the virtual dispatch of  $k$  on  $o_i$  to a **target method** (based on type of  $o_i$ )
- $\text{Select}(c, l, c' : o_i)$ : selects **context** for target method  $m$ , based on the information available at call site  $l$
- $c^t : m_{this}$ : **this variable** of  $c^t : m$
- $c^t : m_{pj}$ : the  **$j$ -th parameter** of  $c^t : m$
- $c^t : m_{ret}$ : the variable that holds the **return value** of  $c^t : m$

# Rule: Call

Caller context:  $c$   
Callee context:  $c^t$

Kind	Statement	Rule (under context $c$ )
Call	$l: r = x.k(a_1, \dots, a_n)$	$  \begin{array}{l}  c': o_i \in pt(c: x), \\  m = \text{Dispatch}(o_i, k), c^t = \text{Select}(c, l, c': o_i) \\  c'': o_u \in pt(c: a_j), 1 \leq j \leq n \\  \longrightarrow c''': o_v \in pt(c^t: m_{ret}) \\  \hline  c': o_i \in pt(c^t: m_{this}) \\  c'': o_u \in pt(c^t: m_{pj}), 1 \leq j \leq n \\  \longrightarrow c''': o_v \in pt(c: r)  \end{array}  $



# The X You Need To Understand in This Lecture

- Concept of context sensitivity (C.S.)
- Concept of context-sensitive heap (C.S. heap)
- Why C.S. and C.S. heap improve precision
- Context-sensitive pointer analysis rules

注意注意！  
划重点了！



# Static Program Analysis

## Pointer Analysis Context Sensitivity (II)

Nanjing University

Tian Tan

2021

# Review: C.S. Pointer Analysis Rules

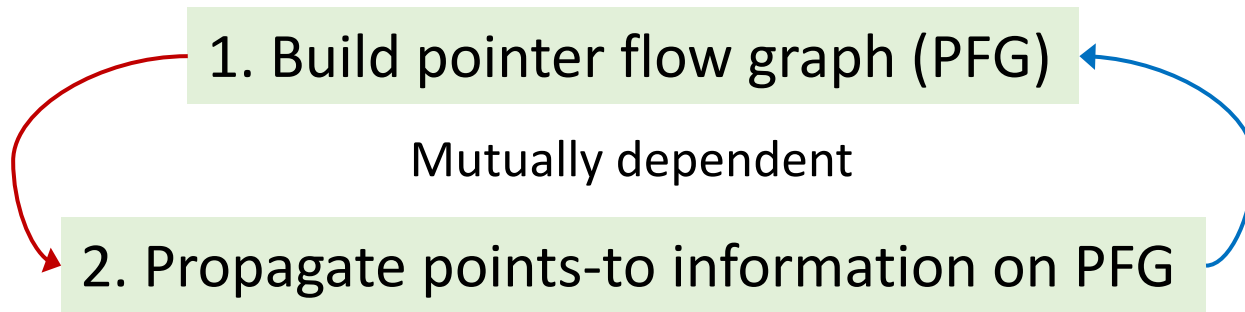
Kind	Statement	Rule (under context $c$ )
New	$i: x = \text{new } T()$	$\overline{c: o_i \in pt(c: x)}$
Assign	$x = y$	$\frac{c': o_i \in pt(c: y)}{c': o_i \in pt(c: x)}$
Store	$x.f = y$	$\frac{c': o_i \in pt(c: x), c'': o_j \in pt(c: y)}{c'': o_j \in pt(c': o_i.f)}$
Load	$y = x.f$	$\frac{c': o_i \in pt(c: x), c'': o_j \in pt(c': o_i.f)}{c'': o_j \in pt(c: y)}$
Call	$l: r = x.k(a_1, \dots, a_n)$	$\frac{\begin{array}{l} c': o_i \in pt(c: x), \\ m = \text{Dispatch}(o_i, k), c^t = \text{Select}(c, l, c': o_i) \\ c'': o_u \in pt(c: a_j), 1 \leq j \leq n \\ c''': o_v \in pt(c^t: m_{ret}) \end{array}}{\begin{array}{l} c': o_i \in pt(c^t: m_{this}) \\ c'': o_u \in pt(c^t: m_{pj}), 1 \leq j \leq n \\ c''': o_v \in pt(c: r) \end{array}}$

# Contents

1. Introduction
2. Context Sensitive Pointer Analysis: Rules
- 3. Context Sensitive Pointer Analysis: Algorithms**
4. Context Sensitivity Variants

# How to Implement Context-Sensitive Pointer Analysis

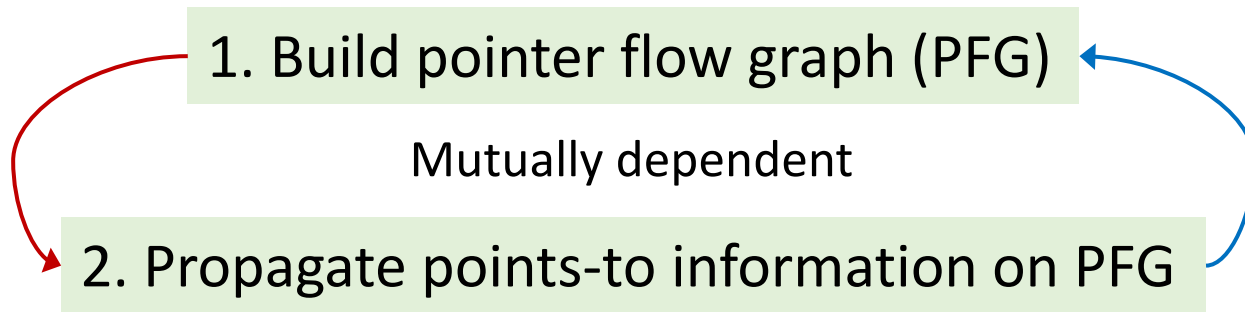
- Recall context-insensitive pointer analysis



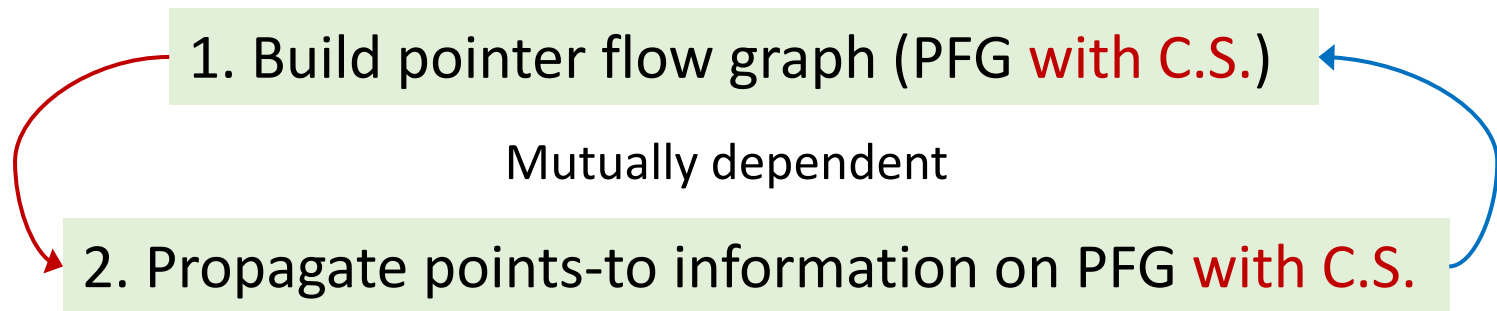


# How to Implement Context-Sensitive Pointer Analysis

- Recall context-insensitive pointer analysis



- Context-sensitive pointer analysis



# Pointer Flow Graph with C.S.

Pointer flow graph of a program is a *directed graph* that expresses how objects flow among the pointers in the program

# Pointer Flow Graph with C.S.

Pointer flow graph of a program is a *directed graph* that expresses how objects flow among the pointers in the program

- Nodes:  $CSPointer = (C \times V) \cup (C \times O \times F)$

A node  $n$  represents *a context-sensitive variable* or *a field of a context-sensitive abstract object*

With C.S., the nodes (pointers) are qualified by contexts

# Pointer Flow Graph with C.S.

Pointer flow graph of a program is a *directed graph* that expresses how objects flow among the pointers in the program

- Nodes:  $\text{CSPtr} = (C \times V) \cup (C \times O \times F)$

A node  $n$  represents *a context-sensitive variable* or *a field of a context-sensitive abstract object*

With C.S., the nodes (pointers) are qualified by contexts

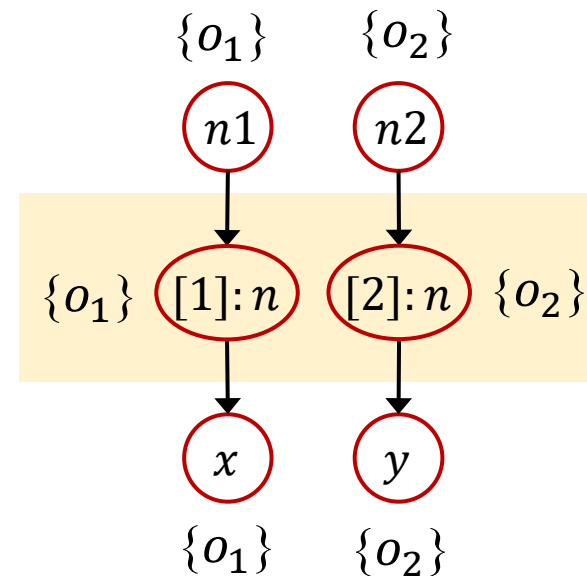
- Edges:  $\text{CSPtr} \times \text{CSPtr}$

An edge  $x \rightarrow y$  means that the objects pointed by pointer  $x$  *may flow to* (and also be pointed to by) pointer  $y$

e.g., edge  $c: a \rightarrow c': b$  means that objects in  $pt(c: a)$  may flow to  $pt(c': b)$

# An Example

```
1 x = id(n1);  
2 y = id(n2);  
3 int i = x.get();  
4  
5 Number id(Number n) {  
6     return n;  
7 }
```



The PFG contains two nodes for variable  $n$  in method `id()`, one node per context

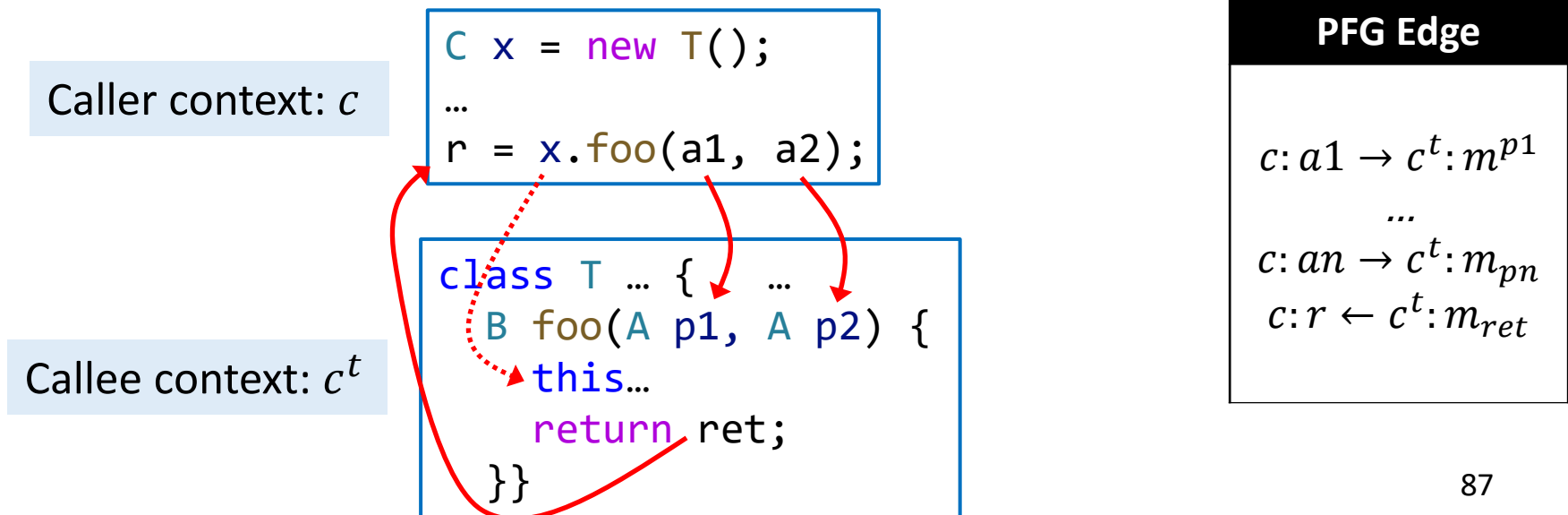
# Pointer Flow Graph: Edges

- PFG edges are added according to the statements of the program and the corresponding rules

Kind	Statement	Rule (under context $c$ )	PFG Edge
New	$i: x = \text{new } T()$	$\overline{c: o_i \in pt(c: x)}$	N/A
Assign	$x = y$	$\frac{c': o_i \in pt(\mathbf{c: y})}{c': o_i \in pt(\mathbf{c: x})}$	$c: x \leftarrow c: y$
Store	$x.f = y$	$\frac{c': o_i \in pt(c: x), c'': o_j \in pt(\mathbf{c: y})}{c'': o_j \in pt(\mathbf{c': o_i.f})}$	$c': o_i.f \leftarrow c: y$
Load	$y = x.f$	$\frac{c': o_i \in pt(c: x), c'': o_j \in pt(\mathbf{c': o_i.f})}{c'': o_j \in pt(\mathbf{c: y})}$	$c: y \leftarrow c': o_i.f$

# Pointer Flow Graph: Call

Kind	Statement	Rule (under context $c$ )
Call	$l: r = x.k(a_1, \dots, a_n)$	$  \begin{array}{c}  c': o_i \in pt(c: x), \\  m = \text{Dispatch}(o_i, k), c^t = \text{Select}(c, l, c': o_i) \\  c'': o_u \in pt(\mathbf{c: a_j}), 1 \leq j \leq n \\  c''': o_v \in pt(\mathbf{c^t: m_{ret}}) \\  \hline  c': o_i \in pt(c^t: m_{this}) \\  c'': o_u \in pt(\mathbf{c^t: m_{pj}}), 1 \leq j \leq n \\  c''': o_v \in pt(\mathbf{c: r})  \end{array}  $



# C.S. Pointer Analysis: Algorithm

```

Solve( $m^{entry}$ )
   $WL = [ ], PFG = \{ \}, S = \{ \}, RM = \{ \}, CG = \{ \}$ 
  AddReachable( $[ ]: m^{entry}$ )
  while  $WL$  is not empty do
    remove  $\langle n, pts \rangle$  from  $WL$ 
     $\Delta = pts - pt(n)$ 
    Propagate( $n, \Delta$ )
    if  $n$  represents a variable  $c: x$  then
      foreach  $c': o_i \in \Delta$  do
        foreach  $x.f = y \in S$  do
          AddEdge( $c: y, c': o_i.f$ )
        foreach  $y = x.f \in S$  do
          AddEdge( $c': o_i.f, c: y$ )
        ProcessCall( $c: x, c': o_i$ )

```

$S$	Set of <b>reachable</b> statements
$S_m$	Set of <b>statements in method <math>m</math></b>
$RM$	Set of <b>C.S. reachable</b> methods
$CG$	<b>C.S. call graph edges</b>

```

AddReachable( $c: m$ )
  if  $c: m \notin RM$  then
    add  $c: m$  to  $RM$ 
     $S \cup = S_m$ 
    foreach  $i: x = \text{new } T() \in S_m$  do
      add  $\langle c: x, \{c: o_i\} \rangle$  to  $WL$ 
    foreach  $x = y \in S_m$  do
      AddEdge( $c: y, c: x$ )

ProcessCall( $c: x, c': o_i$ )
  foreach  $l: r = x.k(a_1, \dots, a_n) \in S$  do
     $m = \text{Dispatch}(o_i, k)$ 
     $c^t = \text{Select}(c, l, c': o_i)$ 
    add  $\langle c^t: m_{this}, \{c': o_i\} \rangle$  to  $WL$ 
    if  $c: l \rightarrow c^t: m \notin CG$  then
      add  $c: l \rightarrow c^t: m$  to  $CG$ 
      AddReachable( $c^t: m$ )
      foreach parameter  $p_i$  of  $m$  do
        AddEdge( $c: a_i, c^t: p_i$ )
      AddEdge( $c^t: m_{ret}, c: r$ )

```



# C.I. Pointer Analysis: Algorithm

```

Solve( $m^{entry}$ )
   $WL = [ ], PFG = \{ \}, S = \{ \}, RM = \{ \}, CG = \{ \}$ 
  AddReachable( $m^{entry}$ )
  while  $WL$  is not empty do
    remove  $\langle n, pts \rangle$  from  $WL$ 
     $\Delta = pts - pt(n)$ 
    Propagate( $n, \Delta$ )
    if  $n$  represents a variable  $x$  then
      foreach  $o_i \in \Delta$  do
        foreach  $x.f = y \in S$  do
          AddEdge( $y, o_i.f$ )
        foreach  $y = x.f \in S$  do
          AddEdge( $o_i.f, y$ )
        ProcessCall( $x, o_i$ )

```

$S$	Set of <b>reachable</b> statements
$S_m$	Set of <b>statements in method <math>m</math></b>
$RM$	Set of <b>C.I. reachable</b> methods
$CG$	<b>C.I.</b> call graph <b>edges</b>

```

AddReachable( $m$ )
  if  $m \notin RM$  then
    add  $m$  to  $RM$ 
     $S \cup = S_m$ 
    foreach  $i: x = \text{new } T() \in S_m$  do
      add  $\langle x, \{ o_i \} \rangle$  to  $WL$ 
    foreach  $x = y \in S_m$  do
      AddEdge( $y, x$ )

ProcessCall( $x, o_i$ )
  foreach  $l: r = x.k(a_1, \dots, a_n) \in S$  do
     $m = \text{Dispatch}(o_i, k)$ 
    add  $\langle m_{this}, \{ o_i \} \rangle$  to  $WL$ 
    if  $l \rightarrow m \notin CG$  then
      add  $l \rightarrow m$  to  $CG$ 
      AddReachable( $m$ )
      foreach parameter  $p_i$  of  $m$  do
        AddEdge( $a_i, p_i$ )
      AddEdge( $m_{ret}, r$ )

```

# C.S. Pointer Analysis: Algorithm

```
Solve( $m^{entry}$ )  
   $WL = [ ], PFG = \{ \}, S = \{ \}, RM = \{ \}, CG = \{ \}$   
  AddReachable( $[ ]: m^{entry}$ )  
  while  $WL$  is not empty do  
    remove  $\langle n, pts \rangle$  from  $WL$   
     $\Delta = pts - pt(n)$   
    Propagate( $n, \Delta$ )  
    if  $n$  represents a variable  $c: x$  then  
      foreach  $c': o_i \in \Delta$  do  
        foreach  $x.f = y \in S$  do  
          AddEdge( $c: y, c': o_i.f$ )  
        foreach  $y = x.f \in S$  do  
          AddEdge( $c': o_i.f, c: y$ )  
      ProcessCall( $c: x, c': o_i$ )
```

$S$	Set of <b>reachable</b> statements
$S_m$	Set of <b>statements in method <math>m</math></b>
$RM$	Set of <b>C.S. reachable</b> methods
$CG$	<b>C.S.</b> call graph <b>edges</b>

# C.S. Pointer Analysis: Algorithm

```
Solve( $m^{entry}$ )  
   $WL = []$ ,  $PF\mathcal{G} = \{\}$ ,  $S = \{\}$ ,  $RM = \{\}$ ,  $CG = \{\}$   
  AddReachable( $[\ ]: m^{entry}$ )  
  while  $WL$  is not empty do  
    remove  $\langle n, pts \rangle$  from  $WL$   
     $\Delta = pts - pt(n)$   
    Propagate( $n, \Delta$ )  
    if  $n$  represents a variable  $c: x$  then  
      foreach  $c': o_i \in \Delta$  do  
        foreach  $x.f = y \in S$  do  
          AddEdge( $c: y, c': o_i.f$ )  
        foreach  $y = x.f \in S$  do  
          AddEdge( $c': o_i.f, c: y$ )  
        ProcessCall( $c: x, c': o_i$ )
```

Callee context:  $c^t$

```
class T ... {  
  ...  
  B foo(A p1, A p2) {  
    this..  
    return ret;  
  }  
}
```

$c^t: T.foo(A, A) \in RM$

$S$	Set of <b>reachable</b> statements
$S_m$	Set of <b>statements in method <math>m</math></b>
$RM$	Set of <b>C.S. reachable</b> methods
$CG$	<b>C.S.</b> call graph <b>edges</b>

# C.S. Pointer Analysis: Algorithm

```

Solve( $m^{entry}$ )
   $WL = [], PFG = \{\}, S = \{\}, RM = \{\}, CG = \{\}$ 
  AddReachable( $[\ ]: m^{entry}$ )
  while  $WL$  is not empty do
    remove  $\langle n, pts \rangle$  from  $WL$ 
     $\Delta = pts - pt(n)$ 
    Propagate( $n, \Delta$ )
    if  $n$  represents a variable  $c: x$  then
      foreach  $c': o_i \in \Delta$  do
        foreach  $x.f = y \in S$  do
          AddEdge( $c: y, c': o_i.f$ )
        foreach  $y = x.f \in S$  do
          AddEdge( $c': o_i.f, c: y$ )
        ProcessCall( $c: x, c': o_i$ )
  
```

Caller context:  $c$

```

1 C x = new T();
2 r = x.foo(a1, a2);
  
```

Callee context:  $c^t$

```

class T ... {
  B foo(A p1, A p2) {
    this...
    return ret;
  }
}
  
```

$c^t: T.foo(A, A) \in RM$

$c: 2 \rightarrow c^t: T.foo(A, A) \in CG$

$S$  Set of **reachable** statements  
 $S_m$  Set of **statements in method  $m$**   
 $RM$  Set of **C.S. reachable** methods  
 $CG$  **C.S. call graph edges**

# C.S. Pointer Analysis: Algorithm

```

Solve( $m^{entry}$ )
   $WL = [], PFG = \{\}, S = \{\}, RM = \{\}, CG = \{\}$ 
  AddReachable( $[\ ]: m^{entry}$ )
  while  $WL$  is not empty do
    remove  $\langle n, pts \rangle$  from  $WL$ 
     $\Delta = pts - pt(n)$ 
    Propagate( $n, \Delta$ )
    if  $n$  represents a variable  $c: x$  then
      foreach  $c': o_i \in \Delta$  do
        foreach  $x.f = y \in S$  do
          AddEdge( $c: y, c': o_i.f$ )
        foreach  $y = x.f \in S$  do
          AddEdge( $c': o_i.f, c: y$ )
    ProcessCall( $c: x, c': o_i$ )
  
```

```

AddReachable( $c: m$ )
  if  $c: m \notin RM$  then
    add  $c: m$  to  $RM$ 
     $S \cup = S_m$ 
    foreach  $i: x = \text{new } T() \in S_m$  do
      add  $\langle c: x, \{c: o_i\} \rangle$  to  $WL$ 
    foreach  $x = y \in S_m$  do
      AddEdge( $c: y, c: x$ )
  
```

Kind	Statement	Rule (under context $c$ )	PFG Edge
New	$i: x = \text{new } T()$	$\frac{}{c: o_i \in pt(c: x)}$	N/A
Assign	$x = y$	$\frac{c': o_i \in pt(\textcolor{red}{c}: y)}{c': o_i \in pt(\textcolor{red}{c}: x)}$	$c: x \leftarrow c: y$

# C.S. Pointer Analysis: Algorithm

```

Solve( $m^{entry}$ )
   $WL = [], PFG = \{\}, S = \{\}, RM = \{\}, CG = \{\}$ 
  AddReachable( $[\ ]: m^{entry}$ )
  while  $WL$  is not empty do
    remove  $\langle n, pts \rangle$  from  $WL$ 
     $\Delta = pts - pt(n)$ 
    Propagate( $n, \Delta$ )
    if  $n$  represents a variable  $c: x$  then
      foreach  $c': o_i \in \Delta$  do
        foreach  $x.f = y \in S$  do
          AddEdge( $c: y, c': o_i.f$ )
        foreach  $y = x.f \in S$  do
          AddEdge( $c': o_i.f, c: y$ )
        ProcessCall( $c: x, c': o_i$ )
  
```

```

AddEdge( $s, t$ )
  if  $s \rightarrow t \notin PFG$  then
    add  $s \rightarrow t$  to  $PFG$ 
    if  $pt(s)$  is not empty then
      add  $\langle t, pt(s) \rangle$  to  $WL$ 

Propagate( $n, pts$ )
  if  $pts$  is not empty then
     $pt(n) \cup = pts$ 
    foreach  $n \rightarrow s \in PFG$  do
      add  $\langle s, pts \rangle$  to  $WL$ 
  
```

Exactly same as in C.I. analysis

$S$  Set of **reachable** statements  
 $S_m$  Set of **statements in method  $m$**   
 $RM$  Set of **C.S. reachable** methods  
 $CG$  **C.S. call graph edges**

# C.S. Pointer Analysis: Algorithm

```

Solve( $m^{entry}$ )
   $WL = [], PFG = \{\}, S = \{\}, RM = \{\}, CG = \{\}$ 
  AddReachable( $[\ ]: m^{entry}$ )
  while  $WL$  is not empty do
    remove  $\langle n, pts \rangle$  from  $WL$ 
     $\Delta = pts - pt(n)$ 
    Propagate( $n, \Delta$ )
    if  $n$  represents a variable  $c: x$  then
      foreach  $c': o_i \in \Delta$  do
        foreach  $x.f = y \in S$  do
          AddEdge( $c: y, c': o_i.f$ )
        foreach  $y = x.f \in S$  do
          AddEdge( $c': o_i.f, c: y$ )
    ProcessCall( $c: x, c': o_i$ )
  
```

```

AddEdge( $s, t$ )
  if  $s \rightarrow t \notin PFG$  then
    add  $s \rightarrow t$  to  $PFG$ 
  if  $pt(s)$  is not empty then
    add  $\langle t, pt(s) \rangle$  to  $WL$ 

Propagate( $n, pts$ )
  if  $pts$  is not empty then
     $pt(n) \cup= pts$ 
    foreach  $n \rightarrow s \in PFG$  do
      add  $\langle s, pts \rangle$  to  $WL$ 
  
```

Kind	Statement	Rule (under context $c$ )	PFG Edge
Store	$x.f = y$	$\frac{c': o_i \in pt(c: x), c'': o_j \in pt(\mathbf{c: y})}{c'': o_j \in pt(\mathbf{c': o_i.f})}$	$c': o_i.f \leftarrow c: y$
Load	$y = x.f$	$\frac{c': o_i \in pt(c: x), c'': o_j \in pt(\mathbf{c': o_i.f})}{c'': o_j \in pt(\mathbf{c: y})}$	$c: y \leftarrow c': o_i.f$

# C.S. Pointer Analysis: Algorithm

```

Solve( $m^{entry}$ )
   $WL = [], PFG = \{\}, S = \{\}, RM = \{\}, CG = \{\}$ 
  AddReachable( $[\ ]: m^{entry}$ )
  while  $WL$  is not empty do
    remove  $\langle n, pts \rangle$  from  $WL$ 
     $\Delta = pts - pt(n)$ 
    Propagate( $n, \Delta$ )
    if  $n$  represents a variable  $c: x$  then
      foreach  $c': o_i \in \Delta$  do
        foreach  $x.f = y \in S$  do
          AddEdge( $c: y, c': o_i.f$ )
        foreach  $y = x.f \in S$  do
          AddEdge( $c': o_i.f, c: y$ )
      ProcessCall( $c: x, c': o_i$ )

```

```

ProcessCall( $c: x, c': o_i$ )
  foreach  $l: r = x.k(a1, \dots, a_n) \in S$  do
     $m = \text{Dispatch}(o_i, k)$ 
     $c^t = \text{Select}(c, l, c': o_i)$ 
    add  $\langle c^t: m_{this}, \{c': o_i\} \rangle$  to  $WL$ 
    if  $c: l \rightarrow c^t: m \notin CG$  then
      add  $c: l \rightarrow c^t: m$  to  $CG$ 
      AddReachable( $c^t: m$ )
      foreach parameter  $p_i$  of  $m$  do
        AddEdge( $c: a_i, c^t: p_i$ )
      AddEdge( $c^t: m_{ret}, c: r$ )

```

Kind	Statement	Rule (under context $c$ )	PFG Edge
Call	$l: r = x.k(a1, \dots, a_n)$	$  \begin{array}{c}  c': o_i \in pt(c: x), \\  m = \text{Dispatch}(o_i, k), c^t = \text{Select}(c, l, c': o_i) \\  c'': o_u \in pt(\mathbf{c: a_j}), 1 \leq j \leq n \\  c''': o_v \in pt(\mathbf{c^t: m_{ret}}) \\  \hline  c': o_i \in pt(c^t: m_{this}) \\  c'': o_u \in pt(\mathbf{c^t: m_{pj}}), 1 \leq j \leq n \\  c''': o_v \in pt(\mathbf{c: r})  \end{array}  $	$  \begin{array}{l}  c: a1 \rightarrow c^t: m^{p1} \\  \dots \\  c: a_n \rightarrow c^t: m_{pn} \\  c: r \leftarrow c^t: m_{ret}  \end{array}  $



# C.S. Pointer Analysis: Algorithm

```

Solve( $m^{entry}$ )
   $WL = [], PFG = \{\}, S = \{\}, RM = \{\}, CG = \{\}$ 
  AddReachable( $[\ ]: m^{entry}$ )
  while  $WL$  is not empty do
    remove  $\langle n, pts \rangle$  from  $WL$ 
     $\Delta = pts - pt(n)$ 
    Propagate( $n, \Delta$ )
    if  $n$  represents a variable  $c: x$  then
      foreach  $c': o_i \in \Delta$  do
        foreach  $x.f = y \in S$  do
          AddEdge( $c: y, c': o_i.f$ )
        foreach  $y = x.f \in S$  do
          AddEdge( $c': o_i.f, c: y$ )
        ProcessCall( $c: x, c': o_i$ )

```

$S$	Set of <b>reachable</b> statements
$S_m$	Set of <b>statements in method <math>m</math></b>
$RM$	Set of <b>C.S. reachable</b> methods
$CG$	<b>C.S. call graph edges</b>

```

AddReachable( $c: m$ )
  if  $c: m \notin RM$  then
    add  $c: m$  to  $RM$ 
     $S \cup = S_m$ 
    foreach  $i: x = \text{new } T() \in S_m$  do
      add  $\langle c: x, \{c: o_i\} \rangle$  to  $WL$ 
    foreach  $x = y \in S_m$  do
      AddEdge( $c: y, c: x$ )

ProcessCall( $c: x, c': o_i$ )
  foreach  $l: r = x.k(a_1, \dots, a_n) \in S$  do
     $m = \text{Dispatch}(o_i, k)$ 
     $c^t = \text{Select}(c, l, c': o_i)$ 
    add  $\langle c^t: m_{this}, \{c': o_i\} \rangle$  to  $WL$ 
    if  $c: l \rightarrow c^t: m \notin CG$  then
      add  $c: l \rightarrow c^t: m$  to  $CG$ 
      AddReachable( $c^t: m$ )
      foreach parameter  $p_i$  of  $m$  do
        AddEdge( $c: a_i, c^t: p_i$ )
      AddEdge( $c^t: m_{ret}, c: r$ )

```

# Contents

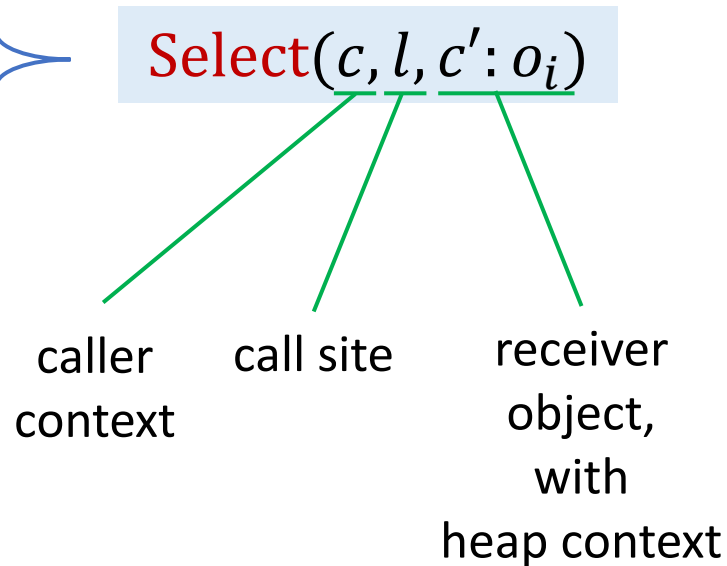
1. Introduction
2. Context Sensitive Pointer Analysis: Rules
3. Context Sensitive Pointer Analysis: Algorithms
4. **Context Sensitivity Variants**

# Context Sensitivity Variants

- Call-site sensitivity
- Object sensitivity
- Type sensitivity
- .....

# Context Sensitivity Variants

- Call-site sensitivity
- Object sensitivity
- Type sensitivity
- .....



# Context Sensitivity Variants

- Call-site sensitivity
- Object sensitivity
- Type sensitivity
- .....

Caller context:  $c$

```
1 C x = new T();  
2 r = x.foo(a1, a2);
```

Callee context:  $c^t$

```
class T ... {  
  B foo(A p1, A p2) {  
    this...  
    return ret;  
  }}  
  }
```

**Select**( $c, l, c': o_i$ )

caller  
context  
 $c$

call site  
2

receiver  
object,  
with  
heap context  
 $c: o_1$

# Context Insensitivity

- Can be seen as a special case of context sensitivity in C.S. analysis framework

**Select**(\_, \_, \_) = []

# Call-Site Sensitivity\*

- Each context consists of a list of call sites (call chain)
  - At a method call, append the **call site** to the **caller context** as callee context
  - Essentially the abstraction of call stacks

$$\text{Select}(c, l, \_) = [l', \dots, l'', l]$$

where  $c = [l', \dots, l'']$

Also called call-string sensitivity, or  $k$ -CFA

\* Olin Shivers, 1991. “*Control-Flow Analysis of Higher-Order Languages*”. Ph.D. Dissertation. Carnegie Mellon University.

# Call-Site Sensitivity: Example

```
1  void main() {  
2      ...  
3      a.foo();  
4      ...  
5  }  
6  
7  void foo() {  
8      ...  
9      b.bar();  
10     ...  
11 }  
12  
13 void bar() {  
14     ...  
15     ...  
16     ...  
17 }
```

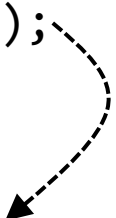


# Call-Site Sensitivity: Example

```
1 void main() { Context: [ ]
2     ...
3     a.foo();
4     ...
5 }
6
7 void foo() {
8     ...
9     b.bar();
10    ...
11 }
12
13 void bar() {
14     ...
15     ...
16     ...
17 }
```

# Call-Site Sensitivity: Example

```
1 void main() { Context: []  
2   ...  
3   a.foo();  
4   ...  
5 }  
6  
7 void foo() { Context: [3]  
8   ...  
9   b.bar();  
10  ...  
11 }  
12  
13 void bar() {  
14   ...  
15   ...  
16   ...  
17 }
```



# Call-Site Sensitivity: Example

```
1 void main() { Context: []  
2   ...  
3   a.foo();  
4   ...  
5 }  
6  
7 void foo() { Context: [3]  
8   ...  
9   b.bar();  
10  ...  
11 }  
12  
13 void bar() { Context: [3,9]  
14   ...  
15   ...  
16   ...  
17 }
```

# Call-Site Sensitivity: Example

```
1 void main() { Context: []  
2   ...  
3   a.foo();  
4   ...  
5 }  
6  
7 void foo() { Context: [3]  
8   ...  
9   b.bar();  
10  ...  
11 }  
12  
13 void bar() { Context: [3,9]  
14   ...  
15   bar(); ?  
16   ...  
17 }
```

# Call-Site Sensitivity: Example

```
1 void main() { Context: [ ]
2   ...
3   a.foo();
4   ...
5 }
6
7 void foo() { Context: [3]
8   ...
9   b.bar();
10  ...
11 }
12
13 void bar() { Context: [3,9]
14   ...
15   bar();
16   ...
17 }
```

The diagram illustrates call-site sensitivity by showing the context of each function call. Dashed arrows indicate the call flow: from line 3 to line 9, and from line 15 to line 15. The context lists show the sequence of call sites (line numbers) that lead to the current function call.

Context: [ ]

Context: [3]

Context: [3,9]

[3,9,15]

[3,9,15,15]

[3,9,15,15,15]

[3,9,15,15,15, ... ]

...

# $k$ -Limiting Context Abstraction

- Motivation
  - Ensure termination of pointer analysis
  - Avoid too many contexts (long call chains) in real-world programs blow up the pointer analysis

# $k$ -Limiting Context Abstraction

- Motivation
  - Ensure termination of pointer analysis
  - Avoid too many contexts (long call chains) in real-world programs blow up the pointer analysis
- Approach: set an upper bound for length of contexts, denoted by  $k$ 
  - For call-site sensitivity, each context consists of the last  $k$  call sites of the call chains
  - In practice,  $k$  is a small number (usually  $\leq 3$ )
  - Method contexts and heap contexts may use different  $k$ 
    - e.g.,  $k=2$  for method contexts,  $k=1$  for heap contexts

# $k$ -Call-Site Sensitivity/ $k$ -CFA

- 1-call-site/1-CFA

$\text{Select}(\_, l, \_) = [l]$

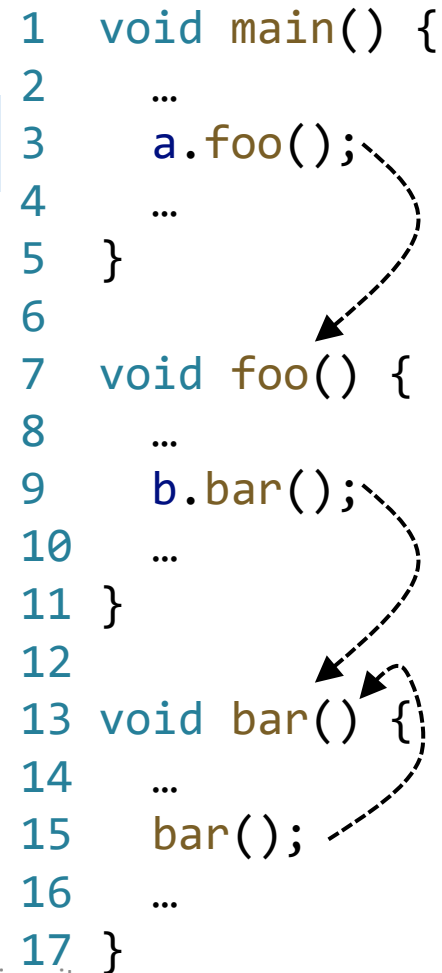


# $k$ -Call-Site Sensitivity/ $k$ -CFA

- 1-call-site/1-CFA

**Select**( $\_, l, \_$ ) =  $[l]$

```
1 void main() {  
2   ...  
3   a.foo();  
4   ...  
5 }  
6  
7 void foo() {  
8   ...  
9   b.bar();  
10  ...  
11 }  
12  
13 void bar() {  
14   ...  
15   bar();  
16   ...  
17 }
```



Context: [ ]

Context: [3]

Context: ???

# $k$ -Call-Site Sensitivity/ $k$ -CFA

- 1-call-site/1-CFA

$\text{Select}(\_, l, \_) = [l]$

```
1 void main() {  
2   ...  
3   a.foo();  
4   ...  
5 }  
6  
7 void foo() {  
8   ...  
9   b.bar();  
10  ...  
11 }  
12  
13 void bar() {  
14   ...  
15   bar();  
16   ...  
17 }
```

Context: [ ]

Context: [3]

Context: [9]  
[15]

# $k$ -Call-Site Sensitivity/ $k$ -CFA

- 1-call-site/1-CFA

$$\text{Select}(\_, l, \_) = [l]$$

- 2-call-site/2-CFA

$$\begin{aligned} \text{Select}(c, l, \_) &= [l'', l] \\ \text{where } c &= [l', l''] \end{aligned}$$

# 1-Call-Site: Example

**Solve**( $m^{entry}$ )

$WL = [], PFG = \{\}, S = \{\}, RM = \{\}, CG = \{\}$

**AddReachable**( $[\ ]: m^{entry}$ )

**while**  $WL$  is not empty **do**

    remove  $\langle n, pts \rangle$  from  $WL$

$\Delta = pts - pt(n)$

**Propagate**( $n, \Delta$ )

**if**  $n$  represents a variable  $c: x$  **then**

**foreach**  $c': o_i \in \Delta$  **do**

**foreach**  $x.f = y \in S$  **do**

**AddEdge**( $c: y, c': o_i.f$ )

**foreach**  $y = x.f \in S$  **do**

**AddEdge**( $c': o_i.f, c: y$ )

**ProcessCall**( $c: x, c': o_i$ )

```
1 class C {
2     static void main() {
3         C c = new C();
4         c.m();
5     }
6
7     Number id(Number n) {
8         return n;
9     }
10    void m() {
11        Number n1, n2, x, y;
12        n1 = new One();
13        n2 = new Two();
14        x = this.id(n1);
15        y = this.id(n2);
16        x.get();
17    }
18 }
```

# 1-Call-Site: Example

**Solve**( $m^{entry}$ )

$WL = [], PFG = \{\}, S = \{\}, RM = \{\}, CG = \{\}$

**AddReachable**( $[\ ]: m^{entry}$ )

**while**  $WL$  is not empty **do**

    remove  $\langle n, pts \rangle$  from  $WL$

$\Delta = pts - pt(n)$

**Propagate**( $n, \Delta$ )

**if**  $n$  represents a variable  $c: x$  **then**

**foreach**  $c': o_i \in \Delta$  **do**

**foreach**  $x.f = y \in S$  **do**

**AddEdge**( $c: y, c': o_i.f$ )

**foreach**  $y = x.f \in S$  **do**

**AddEdge**( $c': o_i.f, c: y$ )

**ProcessCall**( $c: x, c': o_i$ )

```
interface Number {
    int get(); }
class One implements Number {
    public int get() { return 1; }}
class Two implements Number {
    public int get() { return 2; }}
```

```
1 class C {
2     static void main() {
3         C c = new C();
4         c.m();
5     }
6
7     Number id(Number n) {
8         return n;
9     }
10    void m() {
11        Number n1, n2, x, y;
12        n1 = new One();
13        n2 = new Two();
14        x = this.id(n1);
15        y = this.id(n2);
16        x.get();
17    }
18 }
```

# 1-Call-Site: Example

**Solve**( $m^{entry}$ )

$WL = [], PFG = \{\}, S = \{\}, RM = \{\}, CG = \{\}$

**AddReachable**( $[\ ]: m^{entry}$ )

**while**  $WL$  is not empty **do**

    remove  $\langle n, pts \rangle$  from  $WL$

$\Delta = pts - pt(n)$

**Propagate**( $n, \Delta$ )

**if**  $n$  represents a variable  $c: x$  **then**

**foreach**  $c': o_i \in \Delta$  **do**

**foreach**  $x.f = y \in S$  **do**

**AddEdge**( $c: y, c': o_i.f$ )

**foreach**  $y = x.f \in S$  **do**

**AddEdge**( $c': o_i.f, c: y$ )

**ProcessCall**( $c: x, c': o_i$ )

```
interface Number {
    int get(); }
class One implements Number {
    public int get() { return 1; }}
class Two implements Number {
    public int get() { return 2; }}
```

```
1 class C {
2     static void main() {
3         C c = new C();
4         c.m();
5     }
6
7     Number id(Number n) {
8         return n;
9     }
10    void m() {
11        Number n1, n2, x, y;
12        n1 = new One();
13        n2 = new Two();
14        x = this.id(n1);
15        y = this.id(n2);
16        x.get(); ?
17    }
18 }
```

# 1-Call-Site: Example

**Solve**( $m^{entry}$ )

$WL = [], PFG = \{\}, S = \{\}, RM = \{\}, CG = \{\}$

**AddReachable**( $[\ ]: m^{entry}$ )

**while**  $WL$  is not empty **do**

    remove  $\langle n, pts \rangle$  from  $WL$

$\Delta = pts - pt(n)$

**Propagate**( $n, \Delta$ )

**if**  $n$  represents a variable  $c: x$  **then**

**foreach**  $c': o_i \in \Delta$  **do**

**foreach**  $x.f = y \in S$  **do**

**AddEdge**( $c: y, c': o_i.f$ )

**foreach**  $y = x.f \in S$  **do**

**AddEdge**( $c': o_i.f, c: y$ )

**ProcessCall**( $c: x, c': o_i$ )

```
interface Number {
    int get(); }
class One implements Number {
    public int get() { return 1; }}
class Two implements Number {
    public int get() { return 2; }}
```

```
1 class C {
2     static void main() {
3         C c = new C();
4         c.m();
5     }
6
7     Number id(Number n) {
8         return n;
9     }
10    void m() {
11        Number n1, n2, x, y;
12        n1 = new One();
13        n2 = new Two();
14        x = this.id(n1);
15        y = this.id(n2);
16        x.get();
17    }
18 }
```

For simplicity, we do not apply C.S. heap and omit **this** variable of **C.id(Number)**

# 1-Call-Site: Example

*WL*: []

**Solve**( $m^{entry}$ )

→  $WL = [], PFG = \{\}, S = \{\}, RM = \{\}, CG = \{\}$

**AddReachable**([]:  $m^{entry}$ )

**while**  $WL$  is not empty **do**

  remove  $\langle n, pts \rangle$  from  $WL$

$\Delta = pts - pt(n)$

**Propagate**( $n, \Delta$ )

**if**  $n$  represents a variable  $c: x$  **then**

**foreach**  $c': o_i \in \Delta$  **do**

**foreach**  $x.f = y \in S$  **do**

**AddEdge**( $c: y, c': o_i.f$ )

**foreach**  $y = x.f \in S$  **do**

**AddEdge**( $c': o_i.f, c: y$ )

**ProcessCall**( $c: x, c': o_i$ )

**Processing:**

*PFG*:

*CG*: {}

*RM*: {}

```
1 class C {
2   static void main() {
3     C c = new C();
4     c.m();
5   }
6
7   Number id(Number n) {
8     return n;
9   }
10  void m() {
11    Number n1, n2, x, y;
12    n1 = new One();
13    n2 = new Two();
14    x = this.id(n1);
15    y = this.id(n2);
16    x.get();
17  }
18 }
```



# 1-Call-Site: Example

*WL:* []

**Solve**( $m^{entry}$ )

$WL=[]$ ,  $PFG=\{\}$ ,  $S=\{\}$ ,  $RM=\{\}$ ,  $CG=\{\}$

→ **AddReachable**([]:  $m^{entry}$ )

**Processing:**

**AddReachable**( $c:m$ )

**if**  $c:m \notin RM$  **then**

add  $c:m$  to  $RM$

$S \cup = S_m$

**foreach**  $i: x = \text{new } T() \in S_m$  **do**

add  $\langle c:x, \{c:o_i\} \rangle$  to  $WL$

**foreach**  $x = y \in S_m$  **do**

**AddEdge**( $c:y, c:x$ )

*PFG:*

*CG:* {}

*RM:* {}

```
1 class C {
2     static void main() {
3         C c = new C();
4         c.m();
5     }
6
7     Number id(Number n) {
8         return n;
9     }
10    void m() {
11        Number n1,n2,x,y;
12        n1 = new One();
13        n2 = new Two();
14        x = this.id(n1);
15        y = this.id(n2);
16        x.get();
17    }
18 }
```

# 1-Call-Site: Example

*WL:* []

**Solve**( $m^{entry}$ )

$WL=[]$ ,  $PFG=\{\}$ ,  $S=\{\}$ ,  $RM=\{\}$ ,  $CG=\{\}$

**AddReachable**([]:  $m^{entry}$ )

**Processing:**

**AddReachable**( $c:m$ )

**if**  $c:m \notin RM$  **then**

→ add  $c:m$  to  $RM$

$S \cup = S_m$

**foreach**  $i: x = \text{new } T() \in S_m$  **do**

add  $\langle c:x, \{c:o_i\} \rangle$  to  $WL$

**foreach**  $x = y \in S_m$  **do**

**AddEdge**( $c:y, c:x$ )

*PFG:*

*CG:* {}

```
1 class C {
2     static void main() {
3         C c = new C();
4         c.m();
5     }
6
7     Number id(Number n) {
8         return n;
9     }
10    void m() {
11        Number n1,n2,x,y;
12        n1 = new One();
13        n2 = new Two();
14        x = this.id(n1);
15        y = this.id(n2);
16        x.get();
17    }
18 }
```

*RM:* { []:C.main() }

# 1-Call-Site: Example

*WL:*  $[\langle []:c, \{o_3\} \rangle]$

**Solve**( $m^{entry}$ )

$WL=[], PFG=\{\}, S=\{\}, RM=\{\}, CG=\{\}$

**AddReachable**( $[]: m^{entry}$ )

**Processing:**

**AddReachable**( $c: m$ )

**if**  $c: m \notin RM$  **then**

add  $c: m$  to  $RM$

$S \cup = S_m$

**foreach**  $i: x = \text{new } T() \in S_m$  **do**

→ add  $\langle c: x, \{c: o_i\} \rangle$  to  $WL$

**foreach**  $x = y \in S_m$  **do**

**AddEdge**( $c: y, c: x$ )

*PFG:*

*CG:*  $\{\}$

```
1 class C {
2     static void main() {
3         C c = new C();
4         c.m();
5     }
6
7     Number id(Number n) {
8         return n;
9     }
10    void m() {
11        Number n1,n2,x,y;
12        n1 = new One();
13        n2 = new Two();
14        x = this.id(n1);
15        y = this.id(n2);
16        x.get();
17    }
18 }
```

*RM:*  $\{ []:C.main() \}$

# 1-Call-Site: Example

*WL:* []

**Solve**( $m^{entry}$ )

$WL = [], PFG = \{\}, S = \{\}, RM = \{\}, CG = \{\}$

**AddReachable**([]:  $m^{entry}$ )

**while**  $WL$  is not empty **do**

→ **remove**  $\langle n, pts \rangle$  from  $WL$

$\Delta = pts - pt(n)$

**Propagate**( $n, \Delta$ )

**if**  $n$  represents a variable  $c: x$  **then**

**foreach**  $c': o_i \in \Delta$  **do**

**foreach**  $x.f = y \in S$  **do**

**AddEdge**( $c: y, c': o_i.f$ )

**foreach**  $y = x.f \in S$  **do**

**AddEdge**( $c': o_i.f, c: y$ )

**ProcessCall**( $c: x, c': o_i$ )

**Processing:**  $\langle []: c, \{o_3\} \rangle$

*PFG:*

*CG:* {}

*RM:* { []:C.main() }

```
1 class C {
2   static void main() {
3     C c = new C();
4     c.m();
5   }
6
7   Number id(Number n) {
8     return n;
9   }
10  void m() {
11    Number n1,n2,x,y;
12    n1 = new One();
13    n2 = new Two();
14    x = this.id(n1);
15    y = this.id(n2);
16    x.get();
17  }
18 }
```

# 1-Call-Site: Example

*WL*: []

**Solve**( $m^{entry}$ )

$WL = [], PFG = \{\}, S = \{\}, RM = \{\}, CG = \{\}$

**AddReachable**([]:  $m^{entry}$ )

**while** *WL* is not empty **do**

  remove  $\langle n, pts \rangle$  from *WL*

$\Delta = pts - pt(n)$

→ **Propagate**( $n, \Delta$ )

**if**  $n$  represents a variable  $c: x$  **then**

**foreach**  $c': o_i \in \Delta$  **do**

**foreach**  $x.f = y \in S$  **do**

**AddEdge**( $c: y, c': o_i.f$ )

**foreach**  $y = x.f \in S$  **do**

**AddEdge**( $c': o_i.f, c: y$ )

**ProcessCall**( $c: x, c': o_i$ )

**Processing**:  $\langle []:c, \{o_3\} \rangle$

$\{o_3\}$

*PFG*:

$[]:c$

*CG*: {}

*RM*: {  $[]:C.main()$  }

```

1 class C {
2   static void main() {
3     C c = new C();
4     c.m();
5   }
6
7   Number id(Number n) {
8     return n;
9   }
10  void m() {
11    Number n1,n2,x,y;
12    n1 = new One();
13    n2 = new Two();
14    x = this.id(n1);
15    y = this.id(n2);
16    x.get();
17  }
18 }

```

# 1-Call-Site: Example

*WL:* []

```
ProcessCall(c: x, c': oi)  
  foreach l: r = x.k(a1,...,an) ∈ S do  
    m = Dispatch(oi, k)  
    ct = Select(c, l, c': oi)  
    add ⟨ct: mthis, {c': oi}⟩ to WL  
    if c: l → ct: m ∉ CG then  
      add c: l → ct: m to CG  
      AddReachable(ct: m)  
      foreach parameter pi of m do  
        AddEdge(c: ai, ct: pi)  
        AddEdge(ct: mret, c: r)
```

Processing: ⟨[]: *c*, {*o<sub>3</sub>*}⟩

{*o<sub>3</sub>*}

*PFG*: []: *c*

*CG*: {}

→ ProcessCall(*c*: *x*, *c'*: *o<sub>i</sub>*)

```
1 class C {  
2   static void main() {  
3     C c = new C();  
4     c.m();  
5   }  
6  
7   Number id(Number n) {  
8     return n;  
9   }  
10  void m() {  
11    Number n1,n2,x,y;  
12    n1 = new One();  
13    n2 = new Two();  
14    x = this.id(n1);  
15    y = this.id(n2);  
16    x.get();  
17  }  
18 }
```

*RM*: { []:C.main() }

# 1-Call-Site: Example

*WL:* []

**ProcessCall**( $c: x, c': o_i$ )

**foreach**  $l: r = x.k(a_1, \dots, a_n) \in S$  **do**

$m = \text{Dispatch}(o_i, k)$   $m = C.m()$

$\rightarrow c^t = \text{Select}(c, l, c': o_i)$   $c^t = ?$

add  $\langle c^t: m_{this}, \{c': o_i\} \rangle$  to *WL*

**if**  $c: l \rightarrow c^t: m \notin CG$  **then**

add  $c: l \rightarrow c^t: m$  to *CG*

**AddReachable**( $c^t: m$ )

**foreach** parameter  $p_i$  of  $m$  **do**

**AddEdge**( $c: a_i, c^t: p_i$ )

**AddEdge**( $c^t: m_{ret}, c: r$ )

**ProcessCall**( $c: x, c': o_i$ )

**Processing:**  $\langle []:c, \{o_3\} \rangle$

$\{o_3\}$

*PFG:*

[]:c

*CG:* {}

```

1 class C {
2     static void main() {
3         C c = new C();
4         c.m();
5     }
6
7     Number id(Number n) {
8         return n;
9     }
10    void m() {
11        Number n1, n2, x, y;
12        n1 = new One();
13        n2 = new Two();
14        x = this.id(n1);
15        y = this.id(n2);
16        x.get();
17    }
18 }
    
```

*RM:* { []:C.main() }

# 1-Call-Site: Example

*WL:* []

**ProcessCall**( $c: x, c': o_i$ )

**foreach**  $l: r = x.k(a_1, \dots, a_n) \in S$  **do**

$m = \text{Dispatch}(o_i, k)$   $m = C.m()$

$\rightarrow c^t = \text{Select}(c, l, c': o_i)$   $c^t = [4]$

add  $\langle c^t: m_{this}, \{c': o_i\} \rangle$  to *WL*

**if**  $c: l \rightarrow c^t: m \notin CG$  **then**

add  $c: l \rightarrow c^t: m$  to *CG*

**AddReachable**( $c^t: m$ )

**foreach** parameter  $p_i$  of  $m$  **do**

**AddEdge**( $c: a_i, c^t: p_i$ )

**AddEdge**( $c^t: m_{ret}, c: r$ )

**ProcessCall**( $c: x, c': o_i$ )

**Processing:**  $\langle [ ]: c, \{o_3\} \rangle$

$\{o_3\}$

*PFG:*

[ ]:c

*CG:* { }

```

1 class C {
2     static void main() {
3         C c = new C();
4         c.m();
5     }
6
7     Number id(Number n) {
8         return n;
9     }
10    void m() {
11        Number n1, n2, x, y;
12        n1 = new One();
13        n2 = new Two();
14        x = this.id(n1);
15        y = this.id(n2);
16        x.get();
17    }
18 }
    
```

*RM:* { [ ]:C.main() }



# 1-Call-Site: Example

*WL:*  $[\langle [4]:C.m_{this}, \{o_3\} \rangle]$

**ProcessCall**( $c: x, c': o_i$ )

**foreach**  $l: r = x.k(a_1, \dots, a_n) \in S$  **do**

$m = \text{Dispatch}(o_i, k)$   $m = C.m()$

$c^t = \text{Select}(c, l, c': o_i)$   $c^t = [4]$

→ **add**  $\langle c^t: m_{this}, \{c': o_i\} \rangle$  **to**  $WL$

**if**  $c: l \rightarrow c^t: m \notin CG$  **then**

**add**  $c: l \rightarrow c^t: m$  **to**  $CG$

**AddReachable**( $c^t: m$ )

**foreach** parameter  $p_i$  of  $m$  **do**

**AddEdge**( $c: a_i, c^t: p_i$ )

**AddEdge**( $c^t: m_{ret}, c: r$ )

**ProcessCall**( $c: x, c': o_i$ )

**Processing:**  $\langle [ ]:c, \{o_3\} \rangle$

$\{o_3\}$

*PFG:*

$[ ]:c$

*CG:*  $\{ \}$

*RM:*  $\{ [ ]:C.main() \}$

```

1 class C {
2     static void main() {
3         C c = new C();
4         c.m();
5     }
6
7     Number id(Number n) {
8         return n;
9     }
10    void m() {
11        Number n1, n2, x, y;
12        n1 = new One();
13        n2 = new Two();
14        x = this.id(n1);
15        y = this.id(n2);
16        x.get();
17    }
18 }

```

# 1-Call-Site: Example

*WL:*  $\{ \langle [4]:C.m_{this}, \{o_3\} \rangle \}$

**ProcessCall**( $c: x, c': o_i$ )

**foreach**  $l: r = x.k(a_1, \dots, a_n) \in S$  **do**

$m = \text{Dispatch}(o_i, k)$   $m = C.m()$

$c^t = \text{Select}(c, l, c': o_i)$   $c^t = [4]$

add  $\langle c^t: m_{this}, \{c': o_i\} \rangle$  to *WL*

**if**  $c: l \rightarrow c^t: m \notin CG$  **then**

→ add  $c: l \rightarrow c^t: m$  to *CG*

**AddReachable**( $c^t: m$ )

**foreach** parameter  $p_i$  of  $m$  **do**

**AddEdge**( $c: a_i, c^t: p_i$ )

**AddEdge**( $c^t: m_{ret}, c: r$ )

**ProcessCall**( $c: x, c': o_i$ )

**Processing:**  $\langle [ ]:c, \{o_3\} \rangle$

$\{o_3\}$

*PFG:*

$[ ]:c$

*CG:*  $\{ [ ]:4 \rightarrow [4]:C.m() \}$

*RM:*  $\{ [ ]:C.main() \}$

```

1 class C {
2     static void main() {
3         C c = new C();
4         c.m();
5     }
6
7     Number id(Number n) {
8         return n;
9     }
10    void m() {
11        Number n1, n2, x, y;
12        n1 = new One();
13        n2 = new Two();
14        x = this.id(n1);
15        y = this.id(n2);
16        x.get();
17    }
18 }

```

# 1-Call-Site: Example

**ProcessCall**( $c: x, c': o_i$ )

**foreach**  $l: r = x.k(a_1, \dots, a_n) \in S$  **do**

$m = \text{Dispatch}(o_i, k)$   $m = C.m()$

$c^t = \text{Select}(c, l, c': o_i)$   $c^t = [4]$

add  $\langle c^t: m_{this}, \{c': o_i\} \rangle$  to  $WL$

**if**  $c: l \rightarrow c^t: m \notin CG$  **then**

add  $c: l \rightarrow c^t: m$  to  $CG$

➔ **AddReachable**( $c^t: m$ )

**foreach** parameter  $p_i$  of  $m$  **do**

**AddEdge**( $c: a_i, c^t: p_i$ )

**AddEdge**( $c^t: m_{ret}, c: r$ )

**ProcessCall**( $c: x, c': o_i$ )

**WL:**  $[\langle [4]: C.m_{this}, \{o_3\} \rangle,$   
 $\langle [4]: n1, \{o_{12}\} \rangle, \langle [4]: n2, \{o_{13}\} \rangle]$

**Processing:**  $\langle [ ]: c, \{o_3\} \rangle$

**AddReachable**( $c: m$ )

**if**  $c: m \notin RM$  **then**

➔ add  $c: m$  to  $RM$

$S \cup = S_m$

**foreach**  $i: x = \text{new } T() \in S_m$  **do**

➔ add  $\langle c: x, \{c: o_i\} \rangle$  to  $WL$

**foreach**  $x = y \in S_m$  **do**

**AddEdge**( $c: y, c: x$ )

**PFG:**

```

1 class C {
2   static void main() {
3     C c = new C();
4   ➔ c.m();
5   }
6
7   Number id(Number n) {
8     return n;
9   }
10
11   void m() {
12     ➔ n1 = new One();
13     ➔ n2 = new Two();
14     x = this.id(n1);
15     y = this.id(n2);
16     x.get();
17   }
18 }

```

**RM:**  $\{ [ ]: C.main(), [4]: C.m() \}$

# 1-Call-Site: Example

**Solve**( $m^{entry}$ )

$WL=[ ], PFG=\{\}, S=\{\}, RM=\{\}, CG=\{\}$

**AddReachable**( $[ ]: m^{entry}$ )

**while**  $WL$  is not empty **do**

    remove  $\langle n, pts \rangle$  from  $WL$

$\Delta = pts - pt(n)$

→ **Propagate**( $n, \Delta$ )

**if**  $n$  represents a variable  $c: x$  **then**

**foreach**  $c': o_i \in \Delta$  **do**

**foreach**  $x.f = y \in S$  **do**

**AddEdge**( $c: y, c': o_i.f$ )

**foreach**  $y = x.f \in S$  **do**

**AddEdge**( $c': o_i.f, c: y$ )

**ProcessCall**( $c: x, c': o_i$ )

$WL: \quad [ \langle [4]:n1, \{o_{12}\} \rangle, \langle [4]:n2, \{o_{13}\} \rangle ]$

**Processing:**  $\langle [4]:C.m_{this}, \{o_3\} \rangle$

$PFG: \quad \begin{matrix} \{o_3\} \\ [ ]:c \end{matrix}$

$\begin{matrix} \{o_3\} \\ [4]:C.m_{this} \end{matrix}$

$CG: \quad \{ [ ]:4 \rightarrow [4]:C.m() \}$

$RM: \quad \{ [ ]:C.main(), [4]:C.m() \}$

```

1 class C {
2     static void main() {
3         C c = new C();
4         c.m();
5     }
6
7     Number id(Number n) {
8         return n;
9     }
10    void m() {
11        Number n1,n2,x,y;
12        n1 = new One();
13        n2 = new Two();
14        x = this.id(n1);
15        y = this.id(n2);
16        x.get();
17    }
18 }

```

# 1-Call-Site: Example

```

ProcessCall( $c: x, c': o_i$ )
  foreach  $l: r = x.k(a_1, \dots, a_n) \in S$  do
     $m = \text{Dispatch}(o_i, k)$ 
     $c^t = \text{Select}(c, l, c': o_i)$ 
    add  $\langle c^t: m_{this}, \{c': o_i\} \rangle$  to  $WL$ 
    if  $c: l \rightarrow c^t: m \notin CG$  then
      add  $c: l \rightarrow c^t: m$  to  $CG$ 
      AddReachable( $c^t: m$ )
      foreach parameter  $p_i$  of  $m$  do
        AddEdge( $c: a_i, c^t: p_i$ )
      AddEdge( $c^t: m_{ret}, c: r$ )
  ProcessCall( $c: x, c': o_i$ )
  
```

$WL: [\![4]\!]:n1, \{o_{12}\}, [\![4]\!]:n2, \{o_{13}\}\}$

**Processing:**  $\langle [\![4]\!]:C.m_{this}, \{o_3\} \rangle$

$PFG: \{o_3\}$   
 $[\![]\!]:c$

$\{o_3\}$   
 $[\![4]\!]:C.m_{this}$

$CG: \{ [\![]\!]:4 \rightarrow [\![4]\!]:C.m() \}$

$RM: \{ [\![]\!]:C.main(), [\![4]\!]:C.m() \}$

```

1 class C {
2   static void main() {
3     C c = new C();
4     c.m();
5   }
6   Number id(Number n) {
7     return n;
8   }
9
10  void m() {
11    Number n1, n2, x, y;
12    n1 = new One();
13    n2 = new Two();
14    x = this.id(n1);
15    y = this.id(n2);
16    x.get();
17  }
18 }
  
```

# 1-Call-Site: Example

**WL:**  $[\langle [4]:n1, \{o_{12}\} \rangle, \langle [4]:n2, \{o_{13}\} \rangle]$

**ProcessCall**( $c: x, c': o_i$ )

**foreach**  $l: r = x.k(a1, \dots, a_n) \in S$  **do**

$m = \text{Dispatch}(o_i, k)$

$\rightarrow c^t = \text{Select}(c, l, c': o_i)$

$m = C.\text{id}(\text{Number})$

$c^t = [14]$

add  $\langle c^t: m_{\text{this}}, \{c': o_i\} \rangle$  to  $WL$

**if**  $c: l \rightarrow c^t: m \notin CG$  **then**

add  $c: l \rightarrow c^t: m$  to  $CG$

**AddReachable**( $c^t: m$ )

**foreach** parameter  $p_i$  of  $m$  **do**

**AddEdge**( $c: a_i, c^t: p_i$ )

**AddEdge**( $c^t: m_{\text{ret}}, c: r$ )

**ProcessCall**( $c: x, c': o_i$ )

**Processing:**  $\langle [4]:C.m_{\text{this}}, \{o_3\} \rangle$

$\{o_3\}$

**PFG:**  $[ ]:c$

$\{o_3\}$

$[4]:C.m_{\text{this}}$

**CG:**  $\{ [ ]:4 \rightarrow [4]:C.m() \}$

**RM:**  $\{ [ ]:C.\text{main}(), [4]:C.m() \}$

```

1 class C {
2     static void main() {
3         C c = new C();
4         c.m();
5     }
6
7     Number id(Number n) {
8         return n;
9     }
10    void m() {
11        Number n1, n2, x, y;
12        n1 = new One();
13        n2 = new Two();
14        x = this.id(n1);
15        y = this.id(n2);
16        x.get();
17    }
18 }

```

# 1-Call-Site: Example

**WL:**  $[\langle [4]:n1, \{o_{12}\} \rangle, \langle [4]:n2, \{o_{13}\} \rangle]$

**ProcessCall**( $c: x, c': o_i$ )

**foreach**  $l: r = x.k(a1, \dots, a_n) \in S$  **do**

$m = \text{Dispatch}(o_i, k)$

$c^t = \text{Select}(c, l, c': o_i)$

add  $\langle c^t: m_{this}, \{c': o_i\} \rangle$  to  $WL$

**if**  $c: l \rightarrow c^t: m \notin CG$  **then**

→ add  $c: l \rightarrow c^t: m$  to  $CG$

→ **AddReachable**( $c^t: m$ )

**foreach** parameter  $p_i$  of  $m$  **do**

**AddEdge**( $c: a_i, c^t: p_i$ )

**AddEdge**( $c^t: m_{ret}, c: r$ )

**ProcessCall**( $c: x, c': o_i$ )

**Processing:**  $\langle [4]:C.m_{this}, \{o_3\} \rangle$

$m = C.id(\text{Number})$

$c^t = [14]$

$\{o_3\}$

**PFG:**

$[ ]:c$

$\{o_3\}$

$[4]:C.m_{this}$

**CG:**  $\{ [ ]:4 \rightarrow [4]:C.m(),$   
 $[4]:14 \rightarrow [14]:C.id(\text{Number}) \}$

**RM:**  $\{ [ ]:C.main(), [4]:C.m(),$   
 $[14]:C.id(\text{Number}) \}$

```

1 class C {
2     static void main() {
3         C c = new C();
4         c.m();
5     }
6     Number id(Number n) {
7         return n;
8     }
9
10    void m() {
11        Number n1, n2, x, y;
12        n1 = new One();
13        n2 = new Two();
14        x = this.id(n1);
15        y = this.id(n2);
16        x.get();
17    }
18 }

```

# 1-Call-Site: Example

**WL:**  $[\langle [4]:n1, \{o_{12}\} \rangle, \langle [4]:n2, \{o_{13}\} \rangle]$

**ProcessCall**( $c: x, c': o_i$ )

**foreach**  $l: r = x.k(a1, \dots, a_n) \in S$  **do**

$m = \text{Dispatch}(o_i, k)$

$c^t = \text{Select}(c, l, c': o_i)$

add  $\langle c^t: m_{this}, \{c': o_i\} \rangle$  to  $WL$

**if**  $c: l \rightarrow c^t: m \notin CG$  **then**

add  $c: l \rightarrow c^t: m$  to  $CG$

**AddReachable**( $c^t: m$ )

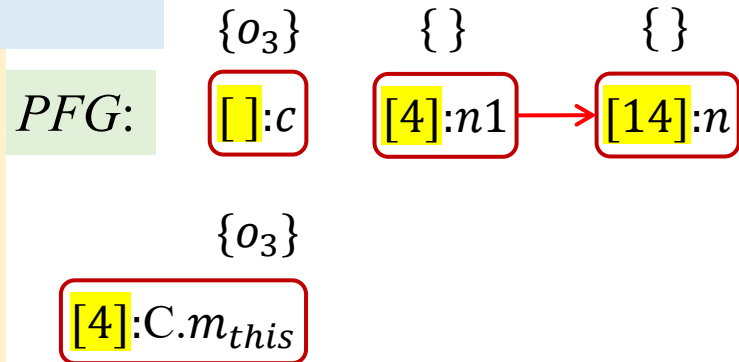
**foreach** parameter  $p_i$  of  $m$  **do**

$\rightarrow$  **AddEdge**( $c: a_i, c^t: p_i$ )

**AddEdge**( $c^t: m_{ret}, c: r$ )

**ProcessCall**( $c: x, c': o_i$ )

**Processing:**  $\langle [4]:C.m_{this}, \{o_3\} \rangle$



**CG:**  $\{ [ ]:4 \rightarrow [4]:C.m(),$   
 $[4]:14 \rightarrow [14]:C.id(Number) \}$

**RM:**  $\{ [ ]:C.main(), [4]:C.m(),$   
 $[14]:C.id(Number) \}$

```

1 class C {
2   static void main() {
3     C c = new C();
4     c.m();
5   }
6   Number id(Number n) {
7     return n;
8   }
9
10  void m() {
11    Number n1, n2, x, y;
12    n1 = new One();
13    n2 = new Two();
14    x = this.id(n1);
15    y = this.id(n2);
16    x.get();
17  }
18 }
  
```



# 1-Call-Site: Example

**WL:**  $[\langle [4]:n1, \{o_{12}\} \rangle, \langle [4]:n2, \{o_{13}\} \rangle]$

**ProcessCall**( $c: x, c': o_i$ )

**foreach**  $l: r = x.k(a1, \dots, a_n) \in S$  **do**

$m = \text{Dispatch}(o_i, k)$

$c^t = \text{Select}(c, l, c': o_i)$

add  $\langle c^t: m_{this}, \{c': o_i\} \rangle$  to  $WL$

**if**  $c: l \rightarrow c^t: m \notin CG$  **then**

add  $c: l \rightarrow c^t: m$  to  $CG$

**AddReachable**( $c^t: m$ )

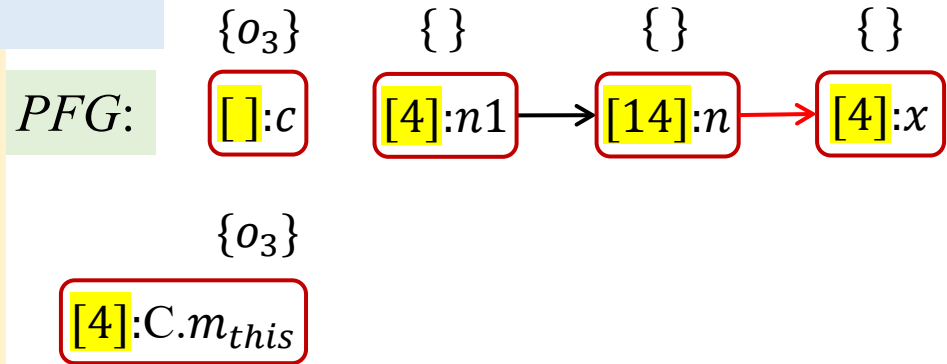
**foreach** parameter  $p_i$  of  $m$  **do**

**AddEdge**( $c: a_i, c^t: p_i$ )

**AddEdge**( $c^t: m_{ret}, c: r$ )

**ProcessCall**( $c: x, c': o_i$ )

**Processing:**  $\langle [4]:C.m_{this}, \{o_3\} \rangle$



**CG:**  $\{ [ ]:4 \rightarrow [4]:C.m(), [4]:14 \rightarrow [14]:C.id(Number) \}$

**RM:**  $\{ [ ]:C.main(), [4]:C.m(), [14]:C.id(Number) \}$

```

1 class C {
2   static void main() {
3     C c = new C();
4     c.m();
5   }
6   Number id(Number n) {
7     return n;
8   }
9   void m() {
10    Number n1, n2, x, y;
11    n1 = new One();
12    n2 = new Two();
13    x = this.id(n1);
14    y = this.id(n2);
15    x.get();
16  }
17 }
18 }
  
```

# 1-Call-Site: Example

**WL:**  $[\langle [4]:n1, \{o_{12}\} \rangle, \langle [4]:n2, \{o_{13}\} \rangle]$

**ProcessCall**( $c: x, c': o_i$ )

**foreach**  $l: r = x.k(a1, \dots, a_n) \in S$  **do**

$m = \text{Dispatch}(o_i, k)$

$c^t = \text{Select}(c, l, c': o_i)$

add  $\langle c^t: m_{this}, \{c': o_i\} \rangle$  to  $WL$

**if**  $c: l \rightarrow c^t: m \notin CG$  **then**

add  $c: l \rightarrow c^t: m$  to  $CG$

**AddReachable**( $c^t: m$ )

**foreach** parameter  $p_i$  of  $m$  **do**

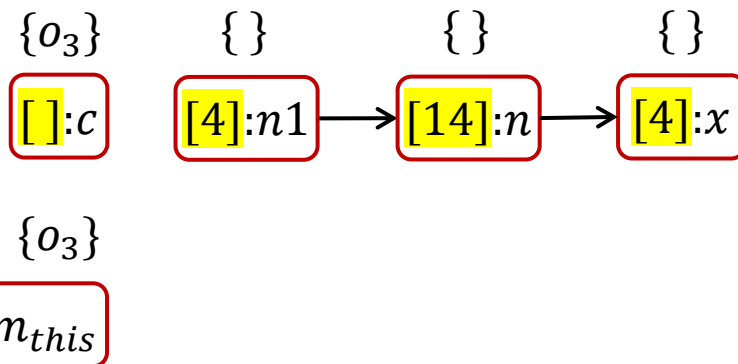
**AddEdge**( $c: a_i, c^t: p_i$ )

**AddEdge**( $c^t: m_{ret}, c: r$ )

**ProcessCall**( $c: x, c': o_i$ )

**Processing:**  $\langle [4]:C.m_{this}, \{o_3\} \rangle$

**PFG:**



**CG:**  $\{ [ ]:4 \rightarrow [4]:C.m(),$   
 $[4]:14 \rightarrow [14]:C.id(Number),$   
 $[4]:15 \rightarrow [15]:C.id(Number) \}$

**RM:**  $\{ [ ]:C.main(), [4]:C.m(),$   
 $[14]:C.id(Number),$   
 $[15]:C.id(Number) \}$

```

1 class C {
2   static void main() {
3     C c = new C();
4     c.m();
5   }
6   Number id(Number n) {
7     return n;
8   }
9 }
10 void m() {
11   Number n1, n2, x, y;
12   n1 = new One();
13   n2 = new Two();
14   x = this.id(n1);
15   y = this.id(n2);
16   x.get();
17 }
18 }

```

# 1-Call-Site: Example

**WL:**  $[\langle [4]:n1, \{o_{12}\} \rangle, \langle [4]:n2, \{o_{13}\} \rangle]$

**ProcessCall**( $c: x, c': o_i$ )

**foreach**  $l: r = x.k(a1, \dots, a_n) \in S$  **do**

$m = \text{Dispatch}(o_i, k)$

$c^t = \text{Select}(c, l, c': o_i)$

add  $\langle c^t: m_{this}, \{c': o_i\} \rangle$  to  $WL$

**if**  $c: l \rightarrow c^t: m \notin CG$  **then**

add  $c: l \rightarrow c^t: m$  to  $CG$

**AddReachable**( $c^t: m$ )

**foreach** parameter  $p_i$  of  $m$  **do**

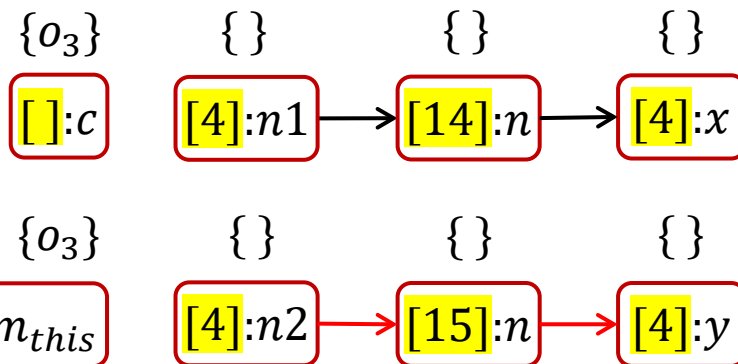
**AddEdge**( $c: a_i, c^t: p_i$ )

**AddEdge**( $c^t: m_{ret}, c: r$ )

**ProcessCall**( $c: x, c': o_i$ )

**Processing:**  $\langle [4]:C.m_{this}, \{o_3\} \rangle$

**PFG:**



**CG:**

$\{ [ ]:4 \rightarrow [4]:C.m(),$   
 $[4]:14 \rightarrow [14]:C.id(Number),$   
 $[4]:15 \rightarrow [15]:C.id(Number) \}$

**RM:**

$\{ [ ]:C.main(), [4]:C.m(),$   
 $[14]:C.id(Number),$   
 $[15]:C.id(Number) \}$

```

1 class C {
2     static void main() {
3         C c = new C();
4         c.m();
5     }
6     Number id(Number n) {
7         return n;
8     }
9 }
10 void m() {
11     Number n1, n2, x, y;
12     n1 = new One();
13     n2 = new Two();
14     x = this.id(n1);
15     y = this.id(n2);
16     x.get();
17 }
18 }

```

# 1-Call-Site: Example

**Solve**( $m^{entry}$ )

$WL = [], PFG = \{\}, S = \{\}, RM = \{\}, CG = \{\}$

**AddReachable**( $[\ ]: m^{entry}$ )

**while**  $WL$  is not empty **do**

  remove  $\langle n, pts \rangle$  from  $WL$

$\Delta = pts - pt(n)$

→ **Propagate**( $n, \Delta$ )

**if**  $n$  represents a variable  $c: x$  **then**

**foreach**  $c': o_i \in \Delta$  **do**

**foreach**  $x.f = y \in S$  **do**

**AddEdge**( $c: y, c': o_i.f$ )

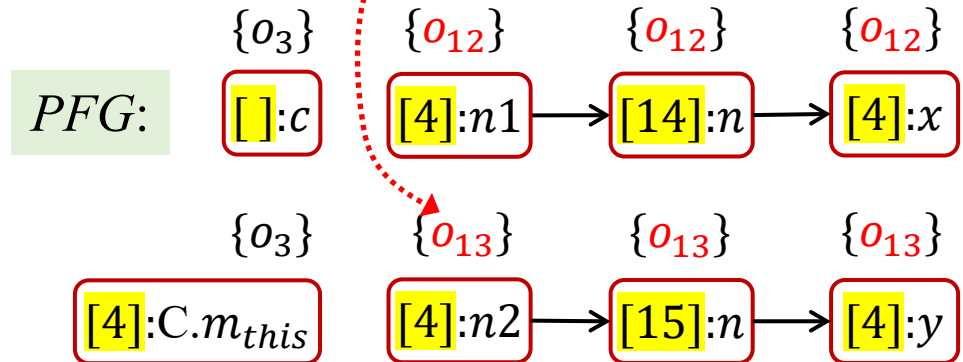
**foreach**  $y = x.f \in S$  **do**

**AddEdge**( $c': o_i.f, c: y$ )

**ProcessCall**( $c: x, c': o_i$ )

**WL:**  $\{ \langle [4]:n1, \{o_{12}\} \rangle, \langle [4]:n2, \{o_{13}\} \rangle \}$

**Processing: ...**



**CG:**  $\{ [ ]:4 \rightarrow [4]:C.m(),$   
 $[4]:14 \rightarrow [14]:C.id(Number),$   
 $[4]:15 \rightarrow [15]:C.id(Number) \}$

**RM:**  $\{ [ ]:C.main(), [4]:C.m(),$   
 $[14]:C.id(Number),$   
 $[15]:C.id(Number) \}$

```

1 class C {
2   static void main() {
3     C c = new C();
4     c.m();
5   }
6   Number id(Number n) {
7     return n;
8   }
9
10  void m() {
11    Number n1, n2, x, y;
12    n1 = new One();
13    n2 = new Two();
14    x = this.id(n1);
15    y = this.id(n2);
16    x.get();
17  }
18 }

```

# 1-Call-Site: Example

*WL:* []

**Solve**( $m^{entry}$ )

$WL = [], PFG = \{\}, S = \{\}, RM = \{\}, CG = \{\}$

**AddReachable**([]:  $m^{entry}$ )

**while**  $WL$  is not empty **do**

  remove  $\langle n, pts \rangle$  from  $WL$

$\Delta = pts - pt(n)$

**Propagate**( $n, \Delta$ )

**if**  $n$  represents a variable  $c: x$  **then**

**foreach**  $c': o_i \in \Delta$  **do**

**foreach**  $x.f = y \in S$  **do**

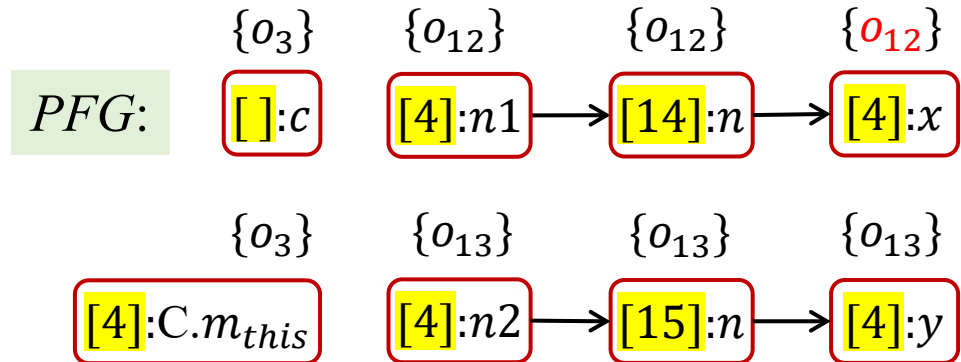
**AddEdge**( $c: y, c': o_i.f$ )

**foreach**  $y = x.f \in S$  **do**

**AddEdge**( $c': o_i.f, c: y$ )

**ProcessCall**( $c: x, c': o_i$ )

**Processing:**  $\langle [4]: x, \{o_{12}\} \rangle$



*CG:* { [ ]:4  $\rightarrow$  [4]:C.m(),  
 [4]:14  $\rightarrow$  [14]:C.id(Number),  
 [4]:15  $\rightarrow$  [15]:C.id(Number),  
 [4]:16  $\rightarrow$  [16]:One.get() }

*RM:* { [ ]:C.main(), [4]:C.m(),  
 [14]:C.id(Number),  
 [15]:C.id(Number),  
 [16]:One.get() } 141

```

1 class C {
2   static void main() {
3     C c = new C();
4     c.m();
5   }
6
7   Number id(Number n) {
8     return n;
9   }
10  void m() {
11    Number n1, n2, x, y;
12    n1 = new One();
13    n2 = new Two();
14    x = this.id(n1);
15    y = this.id(n2);
16    x.get();
17  }
18 }
  
```

# 1-Call-Site: Example

*WL*: []

**Solve**( $m^{entry}$ )

$WL = [], PFG = \{\}, S = \{\}, RM = \{\}, CG = \{\}$

**AddReachable**([]:  $m^{entry}$ )

**while**  $WL$  is not empty **do**

  remove  $\langle n, pts \rangle$  from  $WL$

$\Delta = pts - pt(n)$

**Propagate**( $n, \Delta$ )

**if**  $n$  represents a variable  $c: x$  **then**

**foreach**  $c': o_i \in \Delta$  **do**

**foreach**  $x.f = y \in S$  **do**

**AddEdge**( $c: y, c': o_i.f$ )

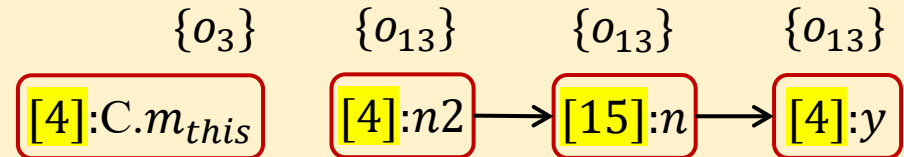
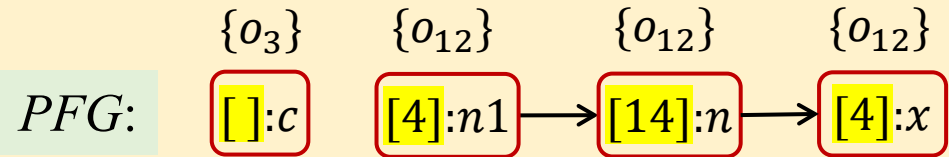
**foreach**  $y = x.f \in S$  **do**

**AddEdge**( $c': o_i.f, c: y$ )

**ProcessCall**( $c: x, c': o_i$ )

Algorithm finishes

Final results



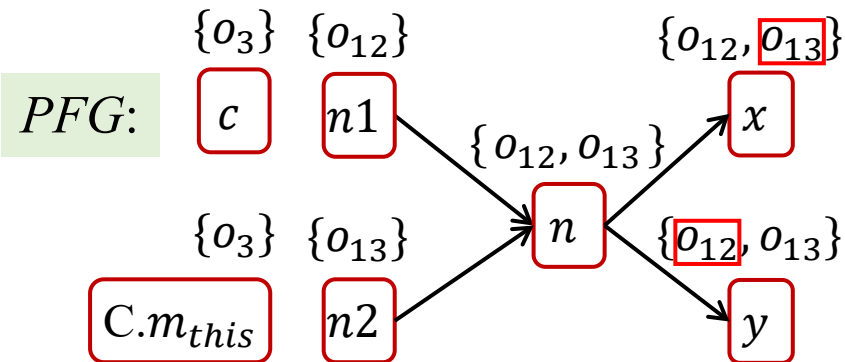
*CG*: { [ ]:4  $\rightarrow$  [4]:C.m(),  
 [4]:14  $\rightarrow$  [14]:C.id(Number),  
 [4]:15  $\rightarrow$  [15]:C.id(Number),  
 [4]:16  $\rightarrow$  [16]:One.get() }

*RM*: { [ ]:C.main(), [4]:C.m(),  
 [14]:C.id(Number),  
 [15]:C.id(Number),  
 [16]:One.get() } 142

```

1 class C {
2   static void main() {
3     C c = new C();
4     c.m();
5   }
6   Number id(Number n) {
7     return n;
8   }
9
10  void m() {
11    Number n1, n2, x, y;
12    n1 = new One();
13    n2 = new Two();
14    x = this.id(n1);
15    y = this.id(n2);
16    x.get();
17  }
18 }
  
```

# C.I. vs. C.S. (1-Call-Site)



**CG:**

- $4 \rightarrow C.m()$ ,
- $14 \rightarrow C.id(\text{Number})$ ,
- $15 \rightarrow C.id(\text{Number})$ ,
- $16 \rightarrow \text{One.get}()$ ,
- $16 \rightarrow \text{Two.get}()$

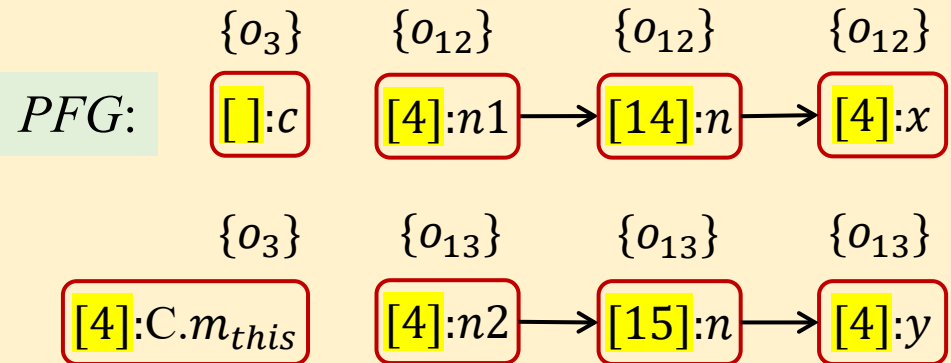
**Spurious**

```

1 class C {
2     static void main() {
3         C c = new C();
4         c.m();
5     }
6
7     Number id(Number n) {
8         return n;
9     }
10    void m() {
11        Number n1, n2, x, y;
12        n1 = new One();
13        n2 = new Two();
14        x = this.id(n1);
15        y = this.id(n2);
16        x.get();
17    }
18 }

```

## Final results



**CG:**

- $[ ]:4 \rightarrow [4]:C.m()$ ,
- $[4]:14 \rightarrow [14]:C.id(\text{Number})$ ,
- $[4]:15 \rightarrow [15]:C.id(\text{Number})$ ,
- $[4]:16 \rightarrow [16]:\text{One.get}()$

# Object Sensitivity\*

- Each context consists of a list of abstract objects (represented by their allocation sites)
  - At a method call, use the **receiver object** with its **heap context** as callee context
  - Distinguish the operations of data flow on **different objects**

$$\text{Select}(\_, \_, c' : o_i) = [o_j, \dots, o_k, o_i]$$

where  $c' = [o_j, \dots, o_k]$

Essentially “allocation-site sensitivity”

\* Ana Milanova, Atanas Rountev, and Barbara G. Ryder. “*Parameterized Object Sensitivity for Points-to and Side-Effect Analyses for Java*”. ISSTA 2002.



# Object Sensitivity: Example

```
1  a1 = new A();
2  a2 = new A();
3  b1 = new B();
4  b2 = new B();
5  a1.set(b1);
6  a2.set(b2);
7  x = a1.get();
8
9  class A {
10     B f;
11     void set(B b) {
12         this.doSet(b);
13     }
14     void doSet(B p) {
15         this.f = p;
16     }
17     B get() {
18         return this.f;
19     }
20 }
```

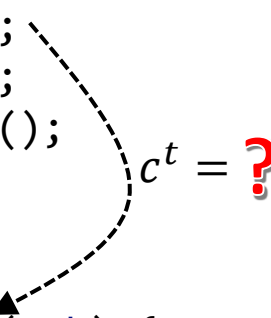


# Object Sensitivity: Example

```
1  a1 = new A(); a1 → o1
2  a2 = new A(); a2 → o2
3  b1 = new B(); b1 → o3
4  b2 = new B(); b2 → o4
5  a1.set(b1);
6  a2.set(b2);
7  x = a1.get();
8
9  class A {
10     B f;
11     void set(B b) {
12         this.doSet(b);
13     }
14     void doSet(B p) {
15         this.f = p;
16     }
17     B get() {
18         return this.f;
19     }
20 }
```

# Object Sensitivity: Example

```
1  a1 = new A(); a1 → o1
2  a2 = new A(); a2 → o2
3  b1 = new B(); b1 → o3
4  b2 = new B(); b2 → o4
5  a1.set(b1);
6  a2.set(b2);
7  x = a1.get();
8
9  class A {
10     B f;
11     void set(B b) {
12         this.doSet(b);
13     }
14     void doSet(B p) {
15         this.f = p;
16     }
17     B get() {
18         return this.f;
19     }
20 }
```



$c^t = ?$

# Object Sensitivity: Example

```

1  a1 = new A(); a1 → o1
2  a2 = new A(); a2 → o2
3  b1 = new B(); b1 → o3
4  b2 = new B(); b2 → o4
5  a1.set(b1);
6  a2.set(b2);
7  x = a1.get();
8
9  class A {
10     B f;
11     void set(B b) {
12         this.doSet(b);
13     }
14     void doSet(B p) {
15         this.f = p;
16     }
17     B get() {
18         return this.f;
19     }
20 }

```

$c^t = [o_1]$

Variable	Object
$[o_1]: set_{this}$	$o_1$
$[o_1]: b$	$o_3$

# Object Sensitivity: Example

```

1  a1 = new A(); a1 → o1
2  a2 = new A(); a2 → o2
3  b1 = new B(); b1 → o3
4  b2 = new B(); b2 → o4
5  a1.set(b1);
6  a2.set(b2);
7  x = a1.get();
8
9  class A {
10     B f;
11     void set(B b) {
12         this.doSet(b);
13     }
14     void doSet(B p) {
15         this.f = p;
16     }
17     B get() {
18         return this.f;
19     }
20 }

```

Diagram illustrating object sensitivity analysis:

- A dashed arrow points from line 5 (`a1.set(b1);`) to line 12 (`this.doSet(b);`), labeled  $c^t = [o_1]$ .
- A dashed arrow points from line 12 (`this.doSet(b);`) to line 15 (`this.f = p;`), labeled  $c^t = ?$ .

Variable	Object
$[o_1]: set_{this}$	$o_1$
$[o_1]: b$	$o_3$

# Object Sensitivity: Example

```

1  a1 = new A(); a1 → o1
2  a2 = new A(); a2 → o2
3  b1 = new B(); b1 → o3
4  b2 = new B(); b2 → o4
5  a1.set(b1);
6  a2.set(b2);
7  x = a1.get();
8
9  class A {
10     B f;
11     void set(B b) {
12         this.doSet(b);
13     }
14     void doSet(B p) {
15         this.f = p;
16     }
17     B get() {
18         return this.f;
19     }
20 }

```

Diagram illustrating object sensitivity analysis:

- A dashed arrow from line 5 (`a1.set(b1);`) points to the `set` method in the class `A` definition (line 11).
- A dashed arrow from line 12 (`this.doSet(b);`) points to the `doSet` method in the class `A` definition (line 14).
- Annotations  $c^t = [o_1]$  are placed near the arrows, indicating the context of the call.

Variable	Object
$[o_1]: set_{this}$	$o_1$
$[o_1]: b$	$o_3$
$[o_1]: doSet_{this}$	$o_1$
$[o_1]: p$	$o_3$
Field	Object
$o_1.f$	$o_3$

# Object Sensitivity: Example

```

1  a1 = new A(); a1 → o1
2  a2 = new A(); a2 → o2
3  b1 = new B(); b1 → o3
4  b2 = new B(); b2 → o4
5  a1.set(b1);
6  a2.set(b2);
7  x = a1.get();
8
ct = [o2] class A {
10     B f;
11     void set(B b) {
12         this.doSet(b); ct = [o1]
13     }
14     void doSet(B p) {
15         this.f = p;
16     }
17     B get() {
18         return this.f;
19     }
20 }

```

Diagram illustrating object sensitivity analysis. The code defines a class `A` with a field `B f` and methods `set(B b)`, `doSet(B p)`, and `get()`. The execution flow is shown with arrows and context-sensitive frames  $c^t$ .

- At line 6, `a2.set(b2)`, the context is  $c^t = [o_2]$  (indicated by a red dashed arrow from line 2).
- At line 12, `this.doSet(b)`, the context is  $c^t = [o_1]$  (indicated by a black dashed arrow from line 6).
- At line 14, `void doSet(B p)`, the context is  $c^t = [o_2]$  (indicated by a red dashed arrow from line 12).
- At line 15, `this.f = p`, the context is  $c^t = [o_1]$  (indicated by a black dashed arrow from line 12).

Variable	Object
$[o_1]: set_{this}$	$o_1$
$[o_1]: b$	$o_3$
$[o_1]: doSet_{this}$	$o_1$
$[o_1]: p$	$o_3$
$[o_2]: set_{this}$	$o_2$
$[o_2]: b$	$o_4$
$[o_2]: doSet_{this}$	$o_2$
$[o_2]: p$	$o_4$
Field	Object
$o_1.f$	$o_3$
$o_2.f$	$o_4$

# Object Sensitivity: Example

```

1  a1 = new A(); a1 → o1
2  a2 = new A(); a2 → o2
3  b1 = new B(); b1 → o3
4  b2 = new B(); b2 → o4
5  a1.set(b1);
6  a2.set(b2);
7  x = a1.get();
8
ct = [o2] class A {
10    B f;
11    void set(B b) {
12        this.doSet(b);
13    }
14    void doSet(B p) {
15        this.f = p;
16    }
17    B get() {
18        return this.f;
19    }
20 }

```

Diagram illustrating object sensitivity analysis. The code defines a class `A` with a field `f` of type `B`. The execution flow is as follows:

- Line 1: `a1 = new A();` creates object `a1` pointing to `o1`.
- Line 2: `a2 = new A();` creates object `a2` pointing to `o2`.
- Line 3: `b1 = new B();` creates object `b1` pointing to `o3`.
- Line 4: `b2 = new B();` creates object `b2` pointing to `o4`.
- Line 5: `a1.set(b1);` calls `set` on `a1` with argument `b1`. The context `ct = [o1]` is established.
- Line 6: `a2.set(b2);` calls `set` on `a2` with argument `b2`. The context `ct = [o2]` is established.
- Line 7: `x = a1.get();` calls `get` on `a1`. The context `ct = [o1]` is established.
- Line 8: The class definition for `A` is shown. The context `ct = [o2]` is established for the `set` method.
- Line 11: The `set` method body is shown. The context `ct = [o1]` is established for the `doSet` method.
- Line 14: The `doSet` method body is shown. The context `ct = [o2]` is established for the `get` method.
- Line 17: The `get` method body is shown. The context `ct = [o1]` is established for the `return` statement.

Variable	Object
[o <sub>1</sub> ]: set <sub>this</sub>	o <sub>1</sub>
[o <sub>1</sub> ]: b	o <sub>3</sub>
[o <sub>1</sub> ]: doSet <sub>this</sub>	o <sub>1</sub>
[o <sub>1</sub> ]: p	o <sub>3</sub>
[o <sub>2</sub> ]: set <sub>this</sub>	o <sub>2</sub>
[o <sub>2</sub> ]: b	o <sub>4</sub>
[o <sub>2</sub> ]: doSet <sub>this</sub>	o <sub>2</sub>
[o <sub>2</sub> ]: p	o <sub>4</sub>
x	o <sub>3</sub>
Field	Object
o <sub>1</sub> .f	o <sub>3</sub>
o <sub>2</sub> .f	o <sub>4</sub>





# Call-Site vs. Object Sensitivity

1-call-site

1-object

```

1  a1 = new A(); a1 → o1
2  a2 = new A(); a2 → o2
3  b1 = new B(); b1 → o3
4  b2 = new B(); b2 → o4
5  a1.set(b1);
6  a2.set(b2);
7  x = a1.get();
8
9  class A {
10     B f;
11     void set(B b) {
12         this.doSet(b);
13     }
14     void doSet(B p) {
15         this.f = p;
16     }
17     B get() {
18         return this.f;
19     }
20 }

```

Variable	Object
[o <sub>1</sub> ]: set <sub>this</sub>	o <sub>1</sub>
[o <sub>1</sub> ]: b	o <sub>3</sub>
[o <sub>1</sub> ]: doSet <sub>this</sub>	o <sub>1</sub>
[o <sub>1</sub> ]: p	o <sub>3</sub>
[o <sub>2</sub> ]: set <sub>this</sub>	o <sub>2</sub>
[o <sub>2</sub> ]: b	o <sub>4</sub>
[o <sub>2</sub> ]: doSet <sub>this</sub>	o <sub>2</sub>
[o <sub>2</sub> ]: p	o <sub>4</sub>
x	o <sub>3</sub>
Field	Object
o <sub>1</sub> .f	o <sub>3</sub>
o <sub>2</sub> .f	o <sub>4</sub>

# Call-Site vs. Object Sensitivity

1-call-site

Variable	Object
[5]: $set_{this}$	$o_1$
[5]: $b$	$o_3$

1-object

Variable	Object
[ $o_1$ ]: $set_{this}$	$o_1$
[ $o_1$ ]: $b$	$o_3$
[ $o_1$ ]: $doSet_{this}$	$o_1$
[ $o_1$ ]: $p$	$o_3$
[ $o_2$ ]: $set_{this}$	$o_2$
[ $o_2$ ]: $b$	$o_4$
[ $o_2$ ]: $doSet_{this}$	$o_2$
[ $o_2$ ]: $p$	$o_4$
$x$	$o_3$
Field	Object
$o_1.f$	$o_3$
$o_2.f$	$o_4$

```

1  a1 = new A(); a1 → o1
2  a2 = new A(); a2 → o2
3  b1 = new B(); b1 → o3
4  b2 = new B(); b2 → o4
5  a1.set(b1);
6  a2.set(b2);
7  x = a1.get();
8
9  class A {
10     B f;
11     void set(B b) {
12         this.doSet(b);
13     }
14     void doSet(B p) {
15         this.f = p;
16     }
17     B get() {
18         return this.f;
19     }
20 }

```

$c^t = [5]$

# Call-Site vs. Object Sensitivity

1-call-site

Variable	Object
[5]: $set_{this}$	$o_1$
[5]: $b$	$o_3$
[12]: $doSet_{this}$	$o_1$
[12]: $p$	$o_3$
Field	Object
$o_1.f$	$o_3$

1-object

Variable	Object
[ $o_1$ ]: $set_{this}$	$o_1$
[ $o_1$ ]: $b$	$o_3$
[ $o_1$ ]: $doSet_{this}$	$o_1$
[ $o_1$ ]: $p$	$o_3$
[ $o_2$ ]: $set_{this}$	$o_2$
[ $o_2$ ]: $b$	$o_4$
[ $o_2$ ]: $doSet_{this}$	$o_2$
[ $o_2$ ]: $p$	$o_4$
$x$	$o_3$
Field	Object
$o_1.f$	$o_3$
$o_2.f$	$o_4$

```

1  a1 = new A(); a1 → o1
2  a2 = new A(); a2 → o2
3  b1 = new B(); b1 → o3
4  b2 = new B(); b2 → o4
5  a1.set(b1);
6  a2.set(b2);
7  x = a1.get();
8
9  class A {
10     B f;
11     void set(B b) {
12         this.doSet(b);
13     }
14     void doSet(B p) {
15         this.f = p;
16     }
17     B get() {
18         return this.f;
19     }
20 }

```

Diagram illustrating call-site sensitivity:

- Call-site  $c^t = [5]$  points to the `set` method call on line 5.
- Call-site  $c^t = [12]$  points to the `doSet` method call on line 12.

# Call-Site vs. Object Sensitivity

1-call-site

Variable	Object
[5]: <i>set<sub>this</sub></i>	$o_1$
[5]: <i>b</i>	$o_3$
[6]: <i>set<sub>this</sub></i>	$o_2$
[6]: <i>b</i>	$o_4$
[12]: <i>doSet<sub>this</sub></i>	$o_1$
[12]: <i>p</i>	$o_3$
Field	Object
$o_1.f$	$o_3$

1-object

Variable	Object
[ $o_1$ ]: <i>set<sub>this</sub></i>	$o_1$
[ $o_1$ ]: <i>b</i>	$o_3$
[ $o_1$ ]: <i>doSet<sub>this</sub></i>	$o_1$
[ $o_1$ ]: <i>p</i>	$o_3$
[ $o_2$ ]: <i>set<sub>this</sub></i>	$o_2$
[ $o_2$ ]: <i>b</i>	$o_4$
[ $o_2$ ]: <i>doSet<sub>this</sub></i>	$o_2$
[ $o_2$ ]: <i>p</i>	$o_4$
$x$	$o_3$
Field	Object
$o_1.f$	$o_3$
$o_2.f$	$o_4$

```

1  a1 = new A(); a1 → o1
2  a2 = new A(); a2 → o2
3  b1 = new B(); b1 → o3
4  b2 = new B(); b2 → o4
5  a1.set(b1);
6  a2.set(b2);
7  x = a1.get();
8
ct = [6] class A {
10     B f;
11     void set(B b) {
12         this.doSet(b);
13     }
14     void doSet(B p) {
15         this.f = p;
16     }
17     B get() {
18         return this.f;
19     }
20 }

```

Diagram illustrating call-site sensitivity with dashed arrows and call-site labels:

- Call-site  $c^t = [5]$  points to the `set` method call on line 5.
- Call-site  $c^t = [6]$  points to the `set` method call on line 6.
- Call-site  $c^t = [12]$  points to the `doSet` method call on line 12.

# Call-Site vs. Object Sensitivity

1-call-site

Variable	Object
[5]: $set_{this}$	$o_1$
[5]: $b$	$o_3$
[6]: $set_{this}$	$o_2$
[6]: $b$	$o_4$
[12]: $doSet_{this}$	$o_1$
[12]: $p$	$o_3$
Field	Object
$o_1.f$	$o_3$

1-object

Variable	Object
[ $o_1$ ]: $set_{this}$	$o_1$
[ $o_1$ ]: $b$	$o_3$
[ $o_1$ ]: $doSet_{this}$	$o_1$
[ $o_1$ ]: $p$	$o_3$
[ $o_2$ ]: $set_{this}$	$o_2$
[ $o_2$ ]: $b$	$o_4$
[ $o_2$ ]: $doSet_{this}$	$o_2$
[ $o_2$ ]: $p$	$o_4$
$x$	$o_3$
Field	Object
$o_1.f$	$o_3$
$o_2.f$	$o_4$

```

1  a1 = new A(); a1 → o1
2  a2 = new A(); a2 → o2
3  b1 = new B(); b1 → o3
4  b2 = new B(); b2 → o4
5  a1.set(b1);
6  a2.set(b2);
7  x = a1.get();
8
ct = [6] class A {
10     B f;
11     void set(B b) {
12         this.doSet(b);
13     }
14     void doSet(B p) {
15         this.f = p;
16     }
17     B get() {
18         return this.f;
19     }
20 }

```

Diagram illustrating call-site sensitivity with annotations:

- $c^t = [5]$  points to the `set` method call on line 5.
- $c^t = [6]$  points to the `set` method call on line 6.
- $c^t = [12]$  points to the `doSet` method call on line 12.
- $c^t = ?$  points to the `doSet` method call on line 14.

# Call-Site vs. Object Sensitivity

1-call-site

Variable	Object
[5]: $set_{this}$	$o_1$
[5]: $b$	$o_3$
[6]: $set_{this}$	$o_2$
[6]: $b$	$o_4$
[12]: $doSet_{this}$	$o_1, o_2$
[12]: $p$	$o_3, o_4$
Field	Object
$o_1.f$	$o_3, o_4$
$o_2.f$	$o_3, o_4$

1-object

Variable	Object
[ $o_1$ ]: $set_{this}$	$o_1$
[ $o_1$ ]: $b$	$o_3$
[ $o_1$ ]: $doSet_{this}$	$o_1$
[ $o_1$ ]: $p$	$o_3$
[ $o_2$ ]: $set_{this}$	$o_2$
[ $o_2$ ]: $b$	$o_4$
[ $o_2$ ]: $doSet_{this}$	$o_2$
[ $o_2$ ]: $p$	$o_4$
$x$	$o_3$
Field	Object
$o_1.f$	$o_3$
$o_2.f$	$o_4$

```

1  a1 = new A(); a1 → o1
2  a2 = new A(); a2 → o2
3  b1 = new B(); b1 → o3
4  b2 = new B(); b2 → o4
5  a1.set(b1);
6  a2.set(b2);
7  x = a1.get();
8
ct = [6] class A {
10     B f;
11     void set(B b) {
12         this.doSet(b);
13     }
14     void doSet(B p) {
15         this.f = p;
16     }
17     B get() {
18         return this.f;
19     }
20 }

```

Diagram illustrating call-site sensitivity. The code defines a class `A` with a field `f` of type `B`. The `set` method calls `doSet`, which updates `this.f`. The `get` method returns `this.f`. The call-site `ct` is shown for the `set` method call at line 6, pointing to the `doSet` method. The call-site `ct` is also shown for the `doSet` method call at line 12, pointing to the `set` method.

# Call-Site vs. Object Sensitivity

1-call-site

Variable	Object
[5]: $set_{this}$	$o_1$
[5]: $b$	$o_3$
[6]: $set_{this}$	$o_2$
[6]: $b$	$o_4$
[12]: $doSet_{this}$	$o_1, o_2$
[12]: $p$	$o_3, o_4$
$x$	$o_3, o_4$
Field	Object
$o_1.f$	$o_3, o_4$
$o_2.f$	$o_3, o_4$

1-object

Variable	Object
[ $o_1$ ]: $set_{this}$	$o_1$
[ $o_1$ ]: $b$	$o_3$
[ $o_1$ ]: $doSet_{this}$	$o_1$
[ $o_1$ ]: $p$	$o_3$
[ $o_2$ ]: $set_{this}$	$o_2$
[ $o_2$ ]: $b$	$o_4$
[ $o_2$ ]: $doSet_{this}$	$o_2$
[ $o_2$ ]: $p$	$o_4$
$x$	$o_3$
Field	Object
$o_1.f$	$o_3$
$o_2.f$	$o_4$

```

1  a1 = new A(); a1 → o1
2  a2 = new A(); a2 → o2
3  b1 = new B(); b1 → o3
4  b2 = new B(); b2 → o4
5  a1.set(b1);
6  a2.set(b2);
7  x = a1.get();
8
ct = [6] class A {
10     B f;
11     void set(B b) {
12         this.doSet(b);
13     }
14     void doSet(B p) {
15         this.f = p;
16     }
17     B get() {
18         return this.f;
19     }
20 }
  
```

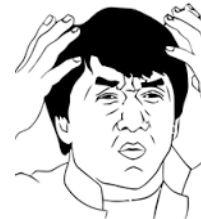
Diagram illustrating call-site sensitivity analysis. The code shows two objects,  $o_1$  and  $o_2$ , each having a `set` method. The `set` method calls `doSet`, which updates the `f` field. The analysis tracks the call site ( $c^t$ ) for each method call. For the first `set` call (line 5),  $c^t = [5]$ . For the second `set` call (line 6),  $c^t = [6]$ . The `doSet` method is called from both `set` methods, so its call site is  $c^t = [12]$ . The `get` method is called from line 7, so its call site is  $c^t = [7]$ .

# Call-Site vs. Object Sensitivity

1-call-site

Variable	Object
[5]: $set_{this}$	$o_1$
[5]: $b$	$o_3$
[6]: $set_{this}$	$o_2$
[6]: $b$	$o_4$
[12]: $doSet_{this}$	$o_1, o_2$
[12]: $p$	$o_3, o_4$
$x$	$o_3, o_4$
Field	Object
$o_1.f$	$o_3, o_4$
$o_2.f$	$o_3, o_4$

Spurious



1-object

Variable	Object
[ $o_1$ ]: $set_{this}$	$o_1$
[ $o_1$ ]: $b$	$o_3$
[ $o_1$ ]: $doSet_{this}$	$o_1$
[ $o_1$ ]: $p$	$o_3$
[ $o_2$ ]: $set_{this}$	$o_2$
[ $o_2$ ]: $b$	$o_4$
[ $o_2$ ]: $doSet_{this}$	$o_2$
[ $o_2$ ]: $p$	$o_4$
$x$	$o_3$
Field	Object
$o_1.f$	$o_3$
$o_2.f$	$o_4$

```

1  a1 = new A(); a1 → o1
2  a2 = new A(); a2 → o2
3  b1 = new B(); b1 → o3
4  b2 = new B(); b2 → o4
5  a1.set(b1);
6  a2.set(b2);
7  x = a1.get();
8
ct = [6] class A {
10     B f;
11     void set(B b) {
12         this.doSet(b);
13     }
14     void doSet(B p) {
15         this.f = p;
16     }
17     B get() {
18         return this.f;
19     }
20 }
  
```

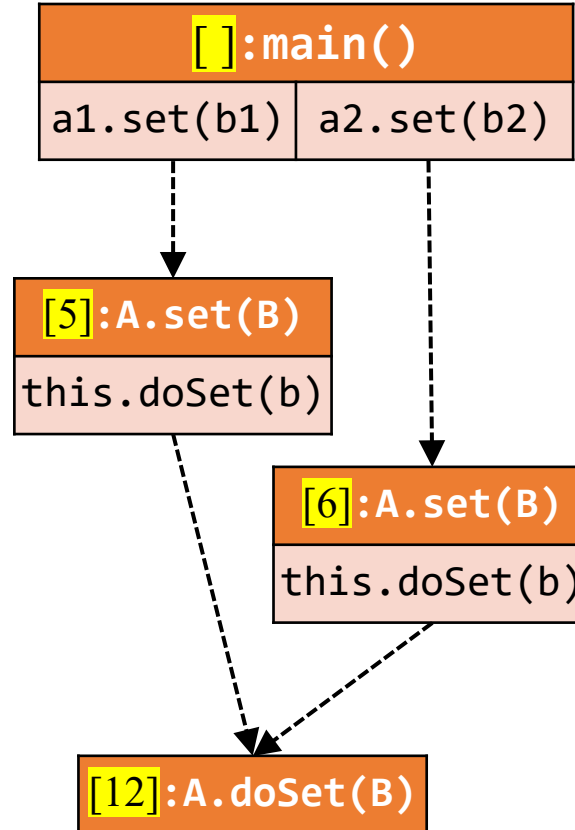
Diagram illustrating call-site sensitivity analysis. The code shows two objects,  $o_1$  and  $o_2$ , each having a `set` method. The `set` method calls `doSet`, which updates the `f` field. The analysis tracks the call site ( $c^t$ ) for each method call. For `a1.set(b1)`,  $c^t = [5]$ . For `a2.set(b2)`,  $c^t = [6]$ . For `a1.get()`,  $c^t = [12]$ . The analysis shows that the `set` method is called from two different call sites, and the `doSet` method is called from two different call sites. The analysis also shows that the `doSet` method updates the `f` field, which is shared by both objects. This leads to a spurious result where the `f` field of  $o_1$  is updated by the `doSet` method called from  $o_2$ .



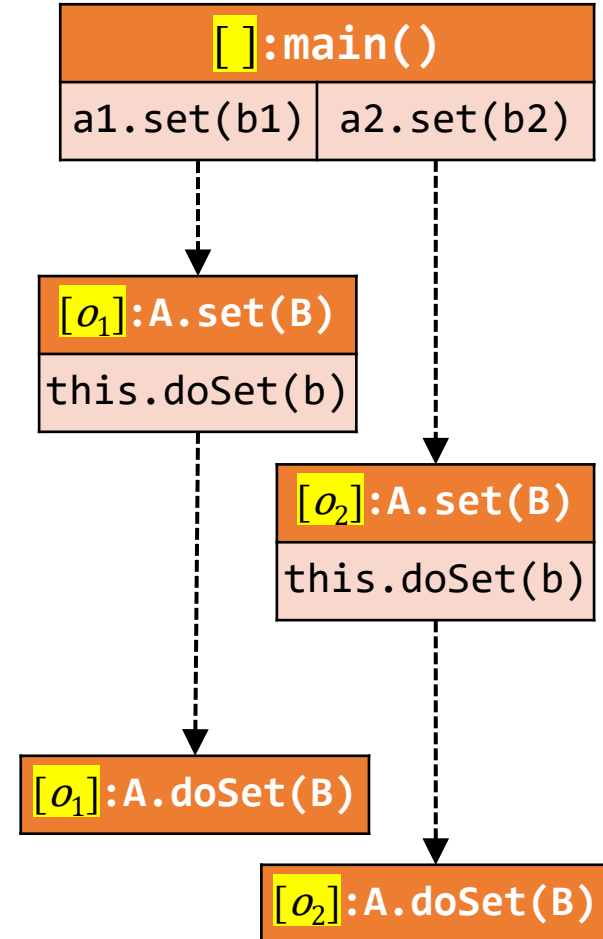
# Call-Site vs. Object Sensitivity

```
1  a1 = new A();
2  a2 = new A();
3  b1 = new B();
4  b2 = new B();
5  a1.set(b1);
6  a2.set(b2);
7  x = a1.get();
8
9  class A {
10     B f;
11     void set(B b) {
12         this.doSet(b);
13     }
14     void doSet(B p) {
15         this.f = p;
16     }
17     B get() {
18         return this.f;
19     }
20 }
```

1-call-site



1-object



# C.S. (1-Object) vs. C.S. (1-Call-Site)

## Final results

*PFG:*  $\{o_3\}$   $\{o_{12}\}$   $\{o_{12}\}$   $\{o_{12}\}$   
 $[ ]:c$   $[4]:n1 \rightarrow [14]:n \rightarrow [4]:x$

$\{o_3\}$   $\{o_{13}\}$   $\{o_{13}\}$   $\{o_{13}\}$   
 $[4]:C.m_{this}$   $[4]:n2 \rightarrow [15]:n \rightarrow [4]:y$

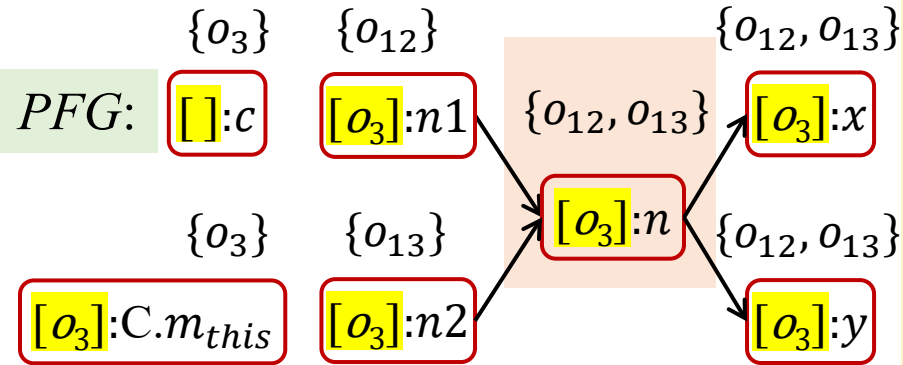
*CG:*  $\{ [ ]:4 \rightarrow [4]:C.m(),$   
 $[4]:14 \rightarrow [14]:C.id(\text{Number}),$   
 $[4]:15 \rightarrow [15]:C.id(\text{Number}),$   
 $[4]:16 \rightarrow [16]:\text{One.get()} \}$

```

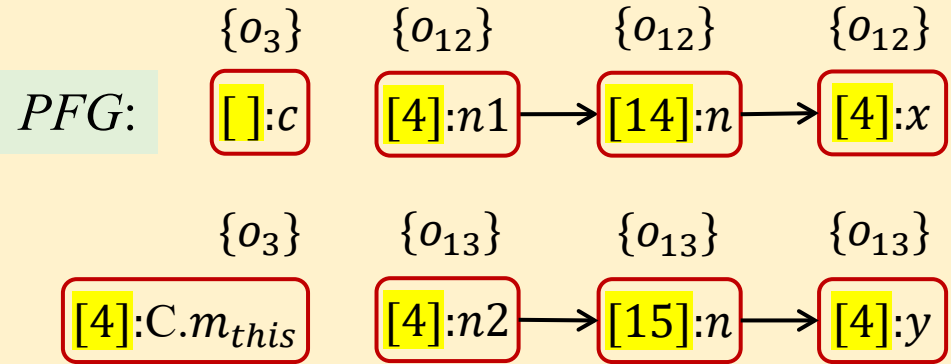
1 class C {
2     static void main() {
3         C c = new C();
4         c.m();
5     }
6
7     Number id(Number n) {
8         return n;
9     }
10    void m() {
11        Number n1,n2,x,y;
12        n1 = new One();
13        n2 = new Two();
14        x = this.id(n1);
15        y = this.id(n2);
16        x.get(); ?
17    }
18 }

```

# C.S. (1-Object) vs. C.S. (1-Call-Site)



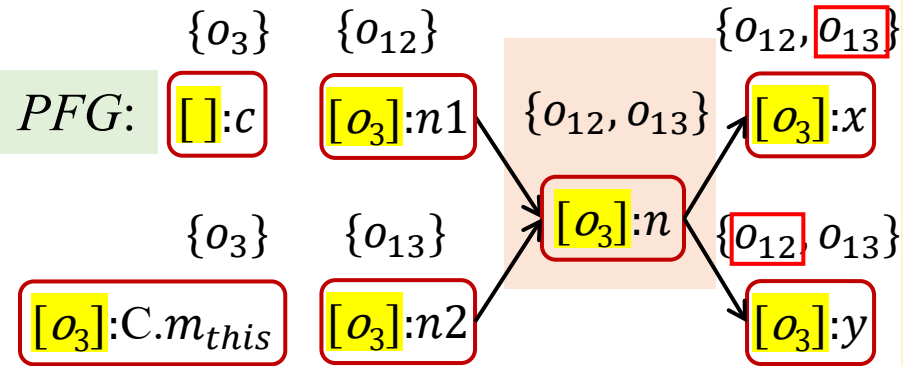
## Final results



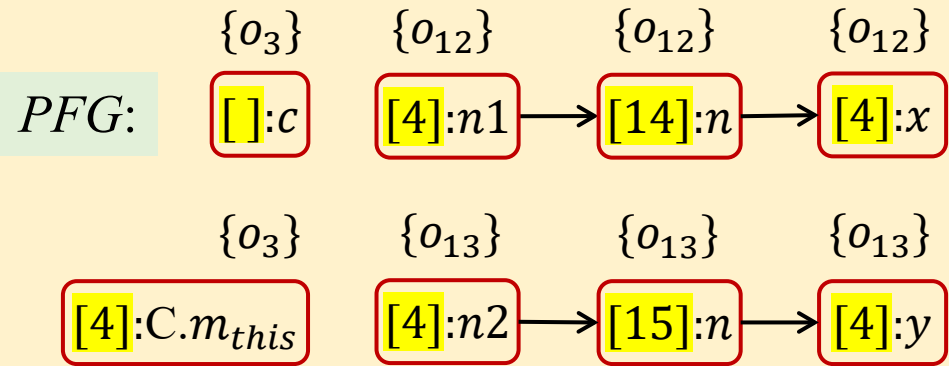
```

1 class C {
2     static void main() {
3         C c = new C();
4         c.m();
5     }
6     Number id(Number n) {
7         return n;
8     }
9
10    void m() {
11        Number n1,n2,x,y;
12        n1 = new One();
13        n2 = new Two();
14        x = this.id(n1);
15        y = this.id(n2);
16        x.get(); ?
17    }
18 }
  
```

# C.S. (1-Object) vs. C.S. (1-Call-Site)



## Final results



```

1 class C {
2   static void main() {
3     C c = new C();
4     c.m();
5   }
6
7   Number id(Number n) {
8     return n;
9   }
10  void m() {
11    Number n1,n2,x,y;
12    n1 = new One();
13    n2 = new Two();
14    x = this.id(n1);
15    y = this.id(n2);
16    x.get();
17  }
18 }

```

# Call-Site vs. Object Sensitivity

- In theory, their precision is incomparable

# Call-Site vs. Object Sensitivity

- In theory, their precision is incomparable
- In practice, object sensitivity generally outperforms call-site sensitivity for OO languages (like Java)

# Call-Site vs. Object Sensitivity

	Time (s)		#may-fail-cast		#call-graph-edge	
	2-call	2-obj	2-call	2-obj	2-call	2-obj
batik	6,886	3,300	2,452	1,606	94,211	76,807
checkstyle	2,277	2,003	863	581	54,171	48,809
sunflow	5,570	1,208	2,504	1,837	100,701	89,866
findbugs	3,812	2,661	2,056	1,409	72,118	65,836
jpc	3,343	559	1,855	1,392	89,677	81,030
eclipse	1,896	146	886	546	42,872	38,151
chart	2,705	282	1,481	883	59,691	52,374
fop	5,503	1,200	1,975	1,446	79,524	71,408
xalan	1,927	1,093	919	533	48,763	44,871
bloat	5,712	3,525	1,699	1,193	58,696	53,143

For all numbers, lower is better (in terms of efficiency or precision)

Yue Li, Tian Tan, Anders Møller, and Yannis Smaragdakis. “*A Principled Approach to Selective Context Sensitivity for Pointer Analysis*”. TOPLAS 2020.

# Call-Site vs. Object Sensitivity

	Time (s)		#may-fail-cast		#call-graph-edge	
	2-call	2-obj	2-call	2-obj	2-call	2-obj
batik	6,886	3,300	2,452	1,606	94,211	76,807
checkstyle	2,277	2,003	863	581	54,171	48,809
sunflow	5,570	1,208	2,504	1,837	100,701	89,866
findbugs	3,812	2,661	2,056	1,409	72,118	65,836
jpc	3,343	559	1,855	1,392	89,677	81,030
eclipse	1,896	146	886	546	42,872	38,151
chart	2,705	282	1,481	883	59,691	52,374
fop	5,503	1,200	1,975	1,446	79,524	71,408
xalan	1,927	1,093	919	533	48,763	44,871
bloat	5,712	3,525	1,699	1,193	58,696	53,143

$A \ a = (A) \ o;$

For all numbers, lower is better (in terms of efficiency or precision)

Yue Li, Tian Tan, Anders Møller, and Yannis Smaragdakis. “*A Principled Approach to Selective Context Sensitivity for Pointer Analysis*”. TOPLAS 2020.



# Call-Site vs. Object Sensitivity

	Time (s)		#may-fail-cast		#call-graph-edge	
	2-call	2-obj	2-call	2-obj	2-call	2-obj
batik	6,886	3,300	2,452	1,606	94,211	76,807
checkstyle	2,277	2,003	863	581	54,171	48,809
sunflow	5,570	1,208	2,504	1,837	100,701	89,866
findbugs	3,812	2,661	2,056	1,409	72,118	65,836
jpc	3,343	559	1,855	1,392	89,677	81,030
eclipse	1,896	146	886	546	42,872	38,151
chart	2,705	282	1,481	883	59,691	52,374
fop	5,503	1,200	1,975	1,446	79,524	71,408
xalan	1,927	1,093	919	533	48,763	44,871
bloat	5,712	3,525	1,699	1,193	58,696	53,143

For all numbers, lower is better (in terms of eff

In general

- Precision: object > call-site
- Efficiency: object > call-site

Yue Li, Tian Tan, Anders Møller, and Yannis Smaragdakis. “*A Principled Approach to Selective Context Sensitivity for Pointer Analysis*”. TOPLAS 2020.

# Type Sensitivity\*

- Each context consists of a list of types
  - At a method call, use the **type** containing the allocation site of the **receiver object** with its **heap context** as callee context
  - A **coarser abstraction** over object sensitivity

$$\text{Select}(\_, \_, c' : o_i) = [t', \dots, t'', \text{InType}(o_i)] \\ \text{where } c' = [t', \dots, t'']$$

\* Yannis Smaragdakis, Martin Bravenboer, and Ondrej Lhoták. “*Pick Your Contexts Well: Understanding Object-Sensitivity*”. POPL 2011.

# Type Sensitivity\*

- Each context consists of a list of types
  - At a method call, use the **type** containing the allocation site of the **receiver object** with its **heap context** as callee context
  - A **coarser abstraction** over object sensitivity

$\text{Select}(\_, \_, c' : o_i) = [t', \dots, t'', \text{InType}(o_i)]$   
where  $c' = [t', \dots, t'']$

```
1 class X {  
2   void m() {  
3     Y y = new Y();  
4   }  
5 }
```

$\text{InType}(o_3) = X$  (not Y!)

\* Yannis Smaragdakis, Martin Bravenboer, and Ondrej Lhoták. “*Pick Your Contexts Well: Understanding Object-Sensitivity*”. POPL 2011.

# Type vs. Object Sensitivity

Under the same  $k$ -limiting, the precision of type sensitivity is **no better than** object sensitivity

- Type sensitivity is a **coarser abstraction** over object sensitivity

```
1  class X {  
2      void main() {  
3          Y y1 = new Y();  
4          y1.foo();  
5          Y y2 = new Y();  
6          y2.foo();  
7          Y y3 = new Y();  
8          y3.foo();  
9      }  
10 }  
11 class Y {  
12     void foo() {}  
13 }
```

Contexts of `foo()`

➤ Object-sensitivity: ?


# Type vs. Object Sensitivity

Under the same  $k$ -limiting, the precision of type sensitivity is **no better than** object sensitivity

- Type sensitivity is a **coarser abstraction** over object sensitivity

```
1  class X {  
2      void main() {  
3          Y y1 = new Y();  
4          y1.foo();  
5          Y y2 = new Y();  
6          y2.foo();  
7          Y y3 = new Y();  
8          y3.foo();  
9      }  
10 }  
11 class Y {  
12     void foo() {}  
13 }
```

Contexts of `foo()`

- Object-sensitivity:  $[o_3], [o_5], [o_7]$
- Type-sensitivity: 

# Type vs. Object Sensitivity

Under the same  $k$ -limiting, the precision of type sensitivity is **no better than** object sensitivity

- Type sensitivity is a **coarser abstraction** over object sensitivity

```
1  class X {  
2      void main() {  
3          Y y1 = new Y();  
4          y1.foo();  
5          Y y2 = new Y();  
6          y2.foo();  
7          Y y3 = new Y();  
8          y3.foo();  
9      }  
10 }  
11 class Y {  
12     void foo() {}  
13 }
```

Contexts of `foo()`

- Object-sensitivity:  $[o_3], [o_5], [o_7]$
- Type-sensitivity:  $[X]$

# Type vs. Object Sensitivity

Under the same  $k$ -limiting, the precision of type sensitivity is **no better than** object sensitivity

- Type sensitivity is a **coarser abstraction** over object sensitivity

```
1 class X {
2     void main() {
3         Y y1 = new Y();
4         y1.foo();
5         Y y2 = new Y();
6         y2.foo();
7         Y y3 = new Y();
8         y3.foo();
9     }
10 }
11 class Y {
12     void foo() {}
13 }
```

Compared to object sensitivity, type sensitivity trades precision for better efficiency, **by merging the allocation sites in the same type** in contexts

- In practice, type sensitivity is **less precise** but **more efficient** than object sensitivity

Contexts of `foo()`

- Object-sensitivity:  $[o_3], [o_5], [o_7]$
- Type-sensitivity:  $[X]$

# Call-Site vs. Object vs. Type Sensitivity

	Time (s)			#may-fail-cast			#call-graph-edge		
	2-call	2-obj	2-type	2-call	2-obj	2-type	2-call	2-obj	2-type
batik	6,886	3,300	378	2,452	1,606	1,938	94,211	76,807	77,337
checkstyle	2,277	2,003	125	863	581	695	54,171	48,809	49,274
sunflow	5,570	1,208	197	2,504	1,837	2,247	100,701	89,866	90,967
findbugs	3,812	2,661	265	2,056	1,409	1,683	72,118	65,836	66,443
jpc	3,343	559	128	1,855	1,392	1,599	89,677	81,030	81,527
eclipse	1,896	146	57	886	546	665	42,872	38,151	38,337
chart	2,705	282	84	1,481	883	1,155	59,691	52,374	52,965
fop	5,503	1,200	251	1,975	1,446	1,753	79,524	71,408	71,847
xalan	1,927	1,093	99	919	533	729	48,763	44,871	45,444
bloat	5,712	3,525	74	1,699	1,193	1,486	58,696	53,143	54,279

For all numbers, lower is better (in terms of efficiency or precision)

Yue Li, Tian Tan, Anders Møller, and Yannis Smaragdakis. “*A Principled Approach to Selective Context Sensitivity for Pointer Analysis*”. TOPLAS 2020.



# Call-Site vs. Object vs. Type Sensitivity

	Time (s)			#may-fail-cast			#call-graph-edge		
	2-call	2-obj	2-type	2-call	2-obj	2-type	2-call	2-obj	2-type
batik	6,886	3,300	378	2,452	1,606	1,938	94,211	76,807	77,337
checkstyle	2,277	2,003	125	863	581	695	54,171	48,809	49,274
sunflow	5,570	1,208	197	2,504	1,837	2,247	100,701	89,866	90,967
findbugs	3,812	2,661	265	2,056	1,409	1,683	72,118	65,836	66,443
jpc	3,343	559	128	1,855	1,392	1,599	89,677	81,030	81,527
eclipse	1,896	146	57	886	546	665	42,872	38,151	38,337
chart	2,705	282	84	1,481	883	1,155	59,691	52,374	52,965
fop	5,503	1,200	251	1,975	1,446	1,753	79,524	71,408	71,847
xalan	1,927	1,093	99	919	533	729	48,763	44,871	45,444
bloat	5,712	3,525	74	1,699	1,193	1,486	58,696	53,143	54,279

For all numbers, lower is better (in terms

In general

- Precision: object > type > call-site
- Efficiency: type > object > call-site

Yue Li, Tian Tan, Anders Møller, and Yannis Smaragdakis. “*A Principled Approach to Selective Context Sensitivity for Pointer Analysis*”. TOPLAS 2020.

# The X You Need To Understand in This Lecture

- Algorithm for context-sensitive pointer analysis
- Common context sensitivity variants
- Differences and relationship among common context sensitivity variants

注意注意！  
划重点了！

