Dimensions of Precision in Reference Analysis of Object-Oriented Programming Languages

Dr. Barbara G. Ryder

Presented by: Ming-Ho Yee

September 23, 2016

Introduction

- Object-oriented (OO) languages have become mature and popular
- Call graphs are useful for OO program analysis
  - But call graph construction is related to reference analysis
- Dimensions of a reference analysis affect precision and cost

Reference Analysis

"Determine information about the set of objects to which a reference variable or field may point during program execution."

- Applications
  - Tools: compiler optimizations, test harnesses, refactoring
  - Analyses: side-effect, escape, def-use

- Choosing the right cost/precision trade-off is very important

Dimensions of Precision Analysis

- Flow sensitivity
- Context sensitivity
- Program representation
- Object representation
- Field sensitivity
- Reference representation
- Directionality

Flow Sensitivity

An analysis is *flow-sensitive* if it accounts for the order of execution of statements in a program. Otherwise, an analysis is *flow-insensitive*.

- Flow-sensitive analyses are more precise, but expensive
- Methods in OO programs are generally small
  - Flow-sensitivity probably not that useful
  - Context-sensitivity probably more useful

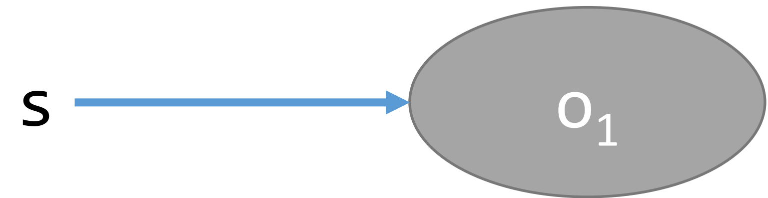
Example: Flow Insensitive Analysis

```
1 A s, t;
2 s = new A(); // o1
3 t = s;
4 s = new A(); // o2
```

Example from Dr. Ryder's presentation

Example: Flow Insensitive Analysis

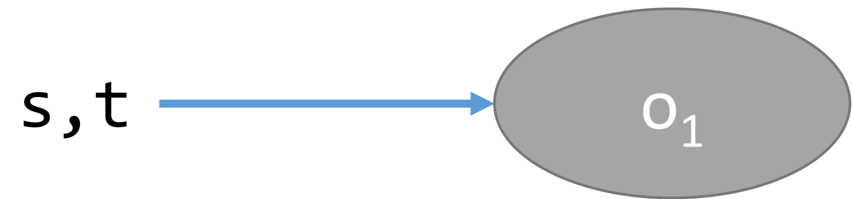
```
1 A s, t;
2 s = new A(); // o1
3 t = s;
4 s = new A(); // o2
```



Example from Dr. Ryder's presentation

Example: Flow Insensitive Analysis

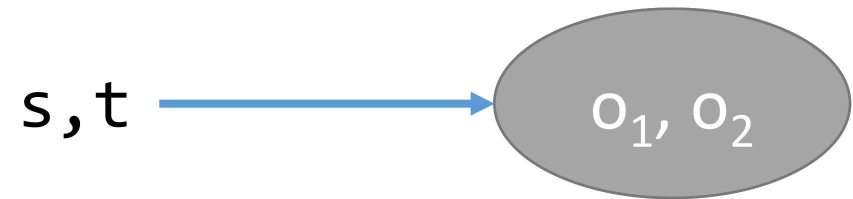
```
1 A s, t;
2 s = new A(); // o1
3 t = s;
4 s = new A(); // o2
```



Example from Dr. Ryder's presentation

Example: Flow Insensitive Analysis

```
1 A s, t;
2 s = new A(); // o1
3 t = s;
4 s = new A(); // o2
```



Example from Dr. Ryder's presentation

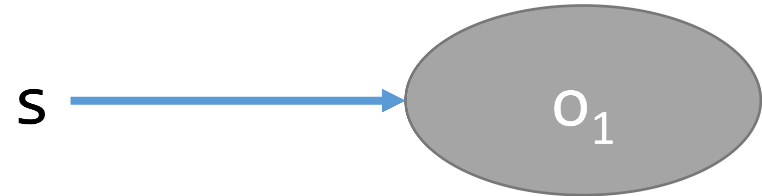
Example: Flow Sensitive Analysis

```
1 A s, t;
2 s = new A(); // o1
3 t = s;
4 s = new A(); // o2
```

Example from Dr. Ryder's presentation

Example: Flow Sensitive Analysis

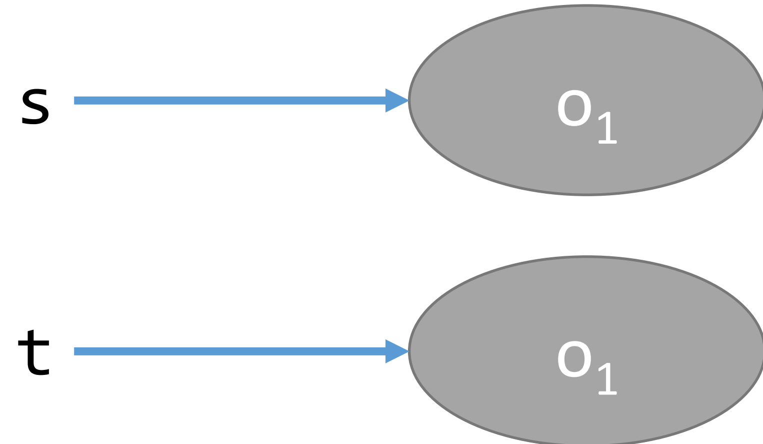
```
1 A s, t;
2 s = new A(); // o1
3 t = s;
4 s = new A(); // o2
```



Example from Dr. Ryder's presentation

Example: Flow Sensitive Analysis

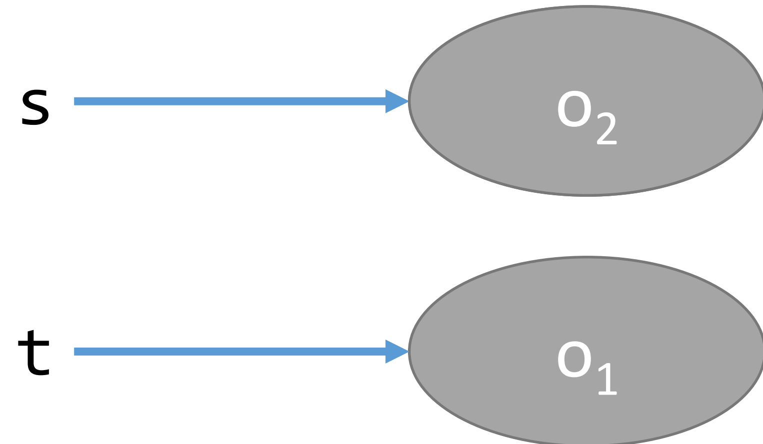
```
1 A s, t;
2 s = new A(); // o1
3 t = s;
4 s = new A(); // o2
```



Example from Dr. Ryder's presentation

Example: Flow Sensitive Analysis

```
1 A s, t;
2 s = new A(); // o1
3 t = s;
4 s = new A(); // o2
```



Example from Dr. Ryder's presentation

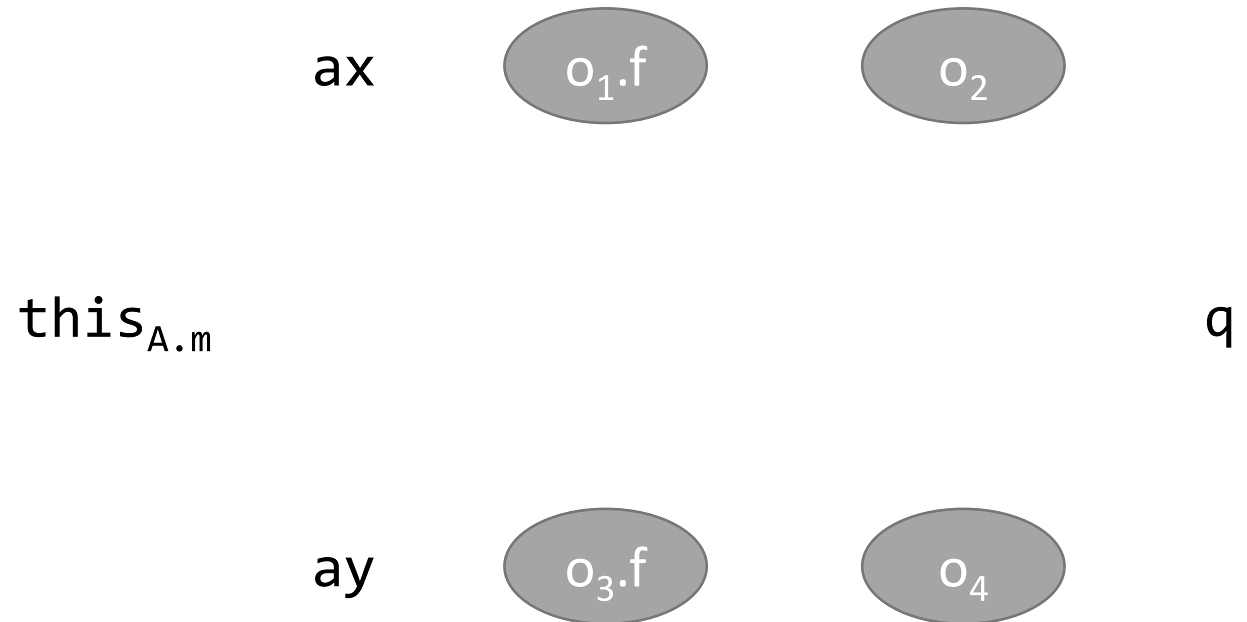
Context Sensitivity

An analysis is *context-sensitive* if it distinguishes between different calling contexts. Otherwise, an analysis is *context-insensitive*.

- Context-sensitive analyses are more precise, but expensive
- Call string vs functional approach
- An area of active research
  - OO method calls would probably benefit from context-sensitive analyses

Example: Context Sensitivity

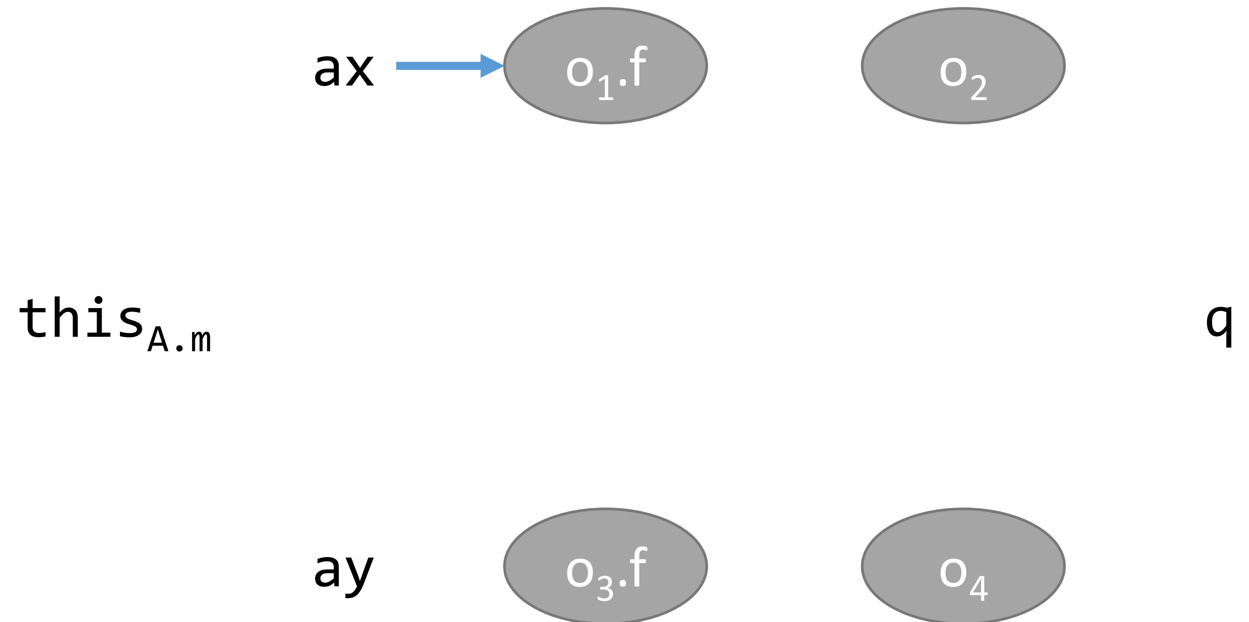
```
class Y extends X {…}
class A {
    X f;
    void m(X q)
      { this.f = q; }
}

A ax = new A(); // o1
ax.m(new X());  // o2
A ay = new A(); // o3
ay.m(new Y());  // o4
```



Example from Dr. Ryder's presentation

Example: Context Sensitivity

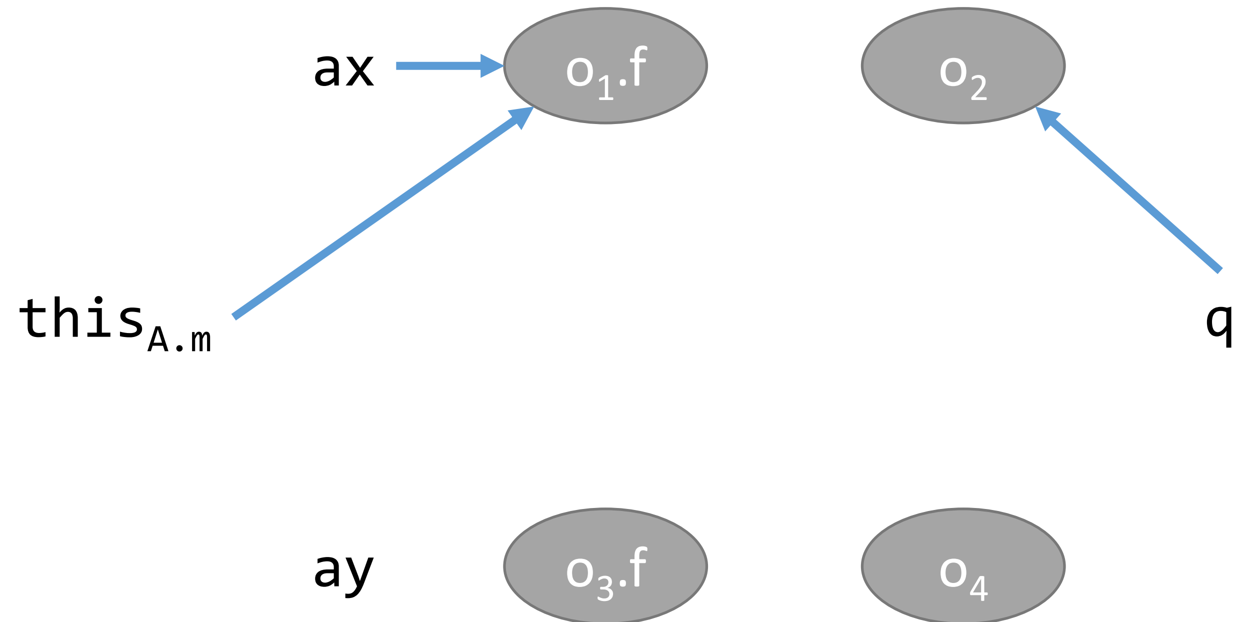
```
class Y extends X {…}
class A {
    X f;
    void m(X q)
      { this.f = q; }
}

A ax = new A(); // o1
ax.m(new X());  // o2
A ay = new A(); // o3
ay.m(new Y());  // o4
```



Example from Dr. Ryder's presentation

Example: Context Sensitivity

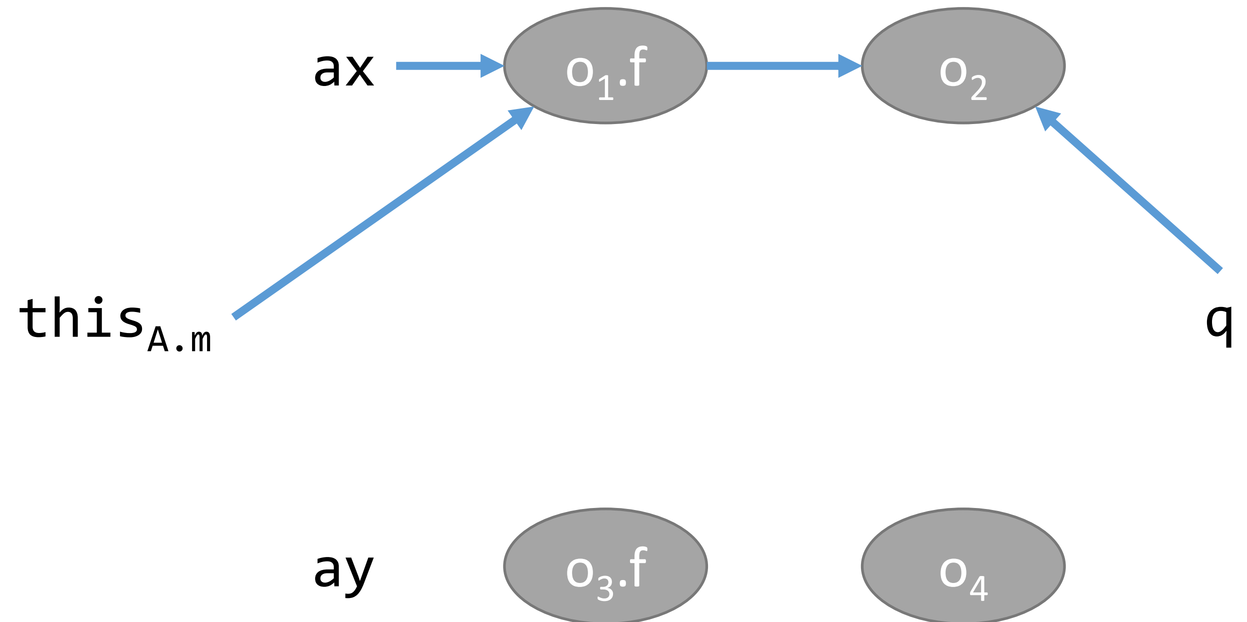
```
class Y extends X {…}
class A {
    X f;
    void m(X q)
      { this.f = q; }
}

A ax = new A(); // o1
ax.m(new X());  // o2
A ay = new A(); // o3
ay.m(new Y());  // o4
```



Example from Dr. Ryder's presentation

Example: Context Sensitivity

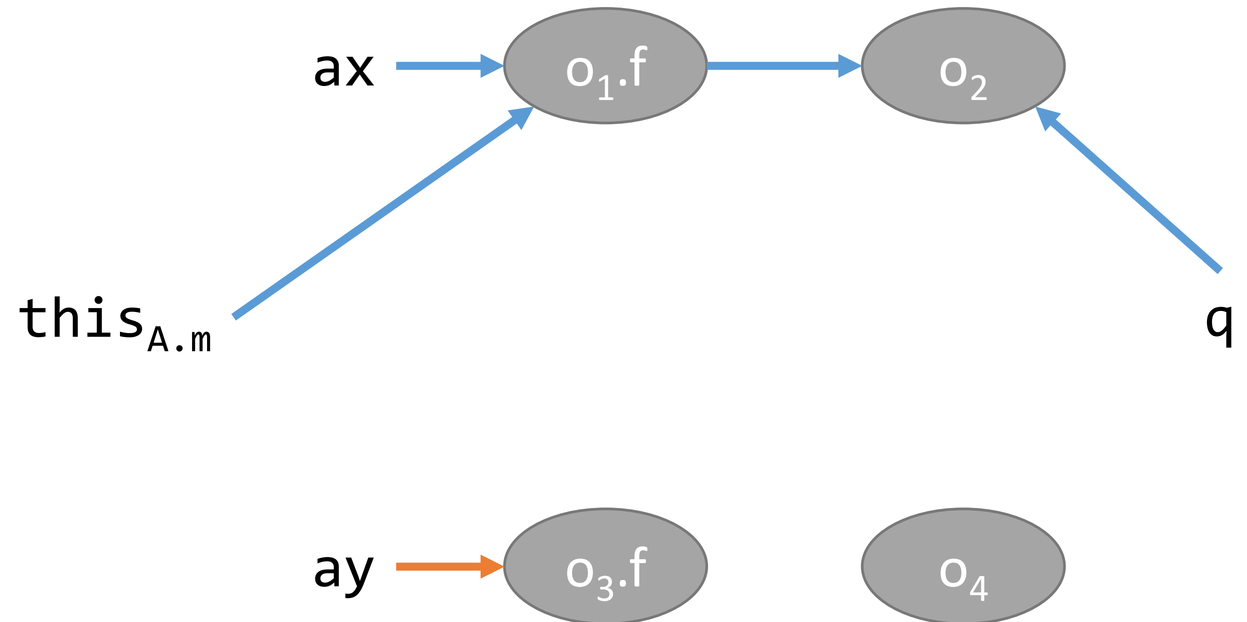
```
class Y extends X {…}
class A {
    X f;
    void m(X q)
      { this.f = q; }
}

A ax = new A(); // o1
ax.m(new X());  // o2
A ay = new A(); // o3
ay.m(new Y());  // o4
```



Example from Dr. Ryder's presentation

Example: Context Sensitivity

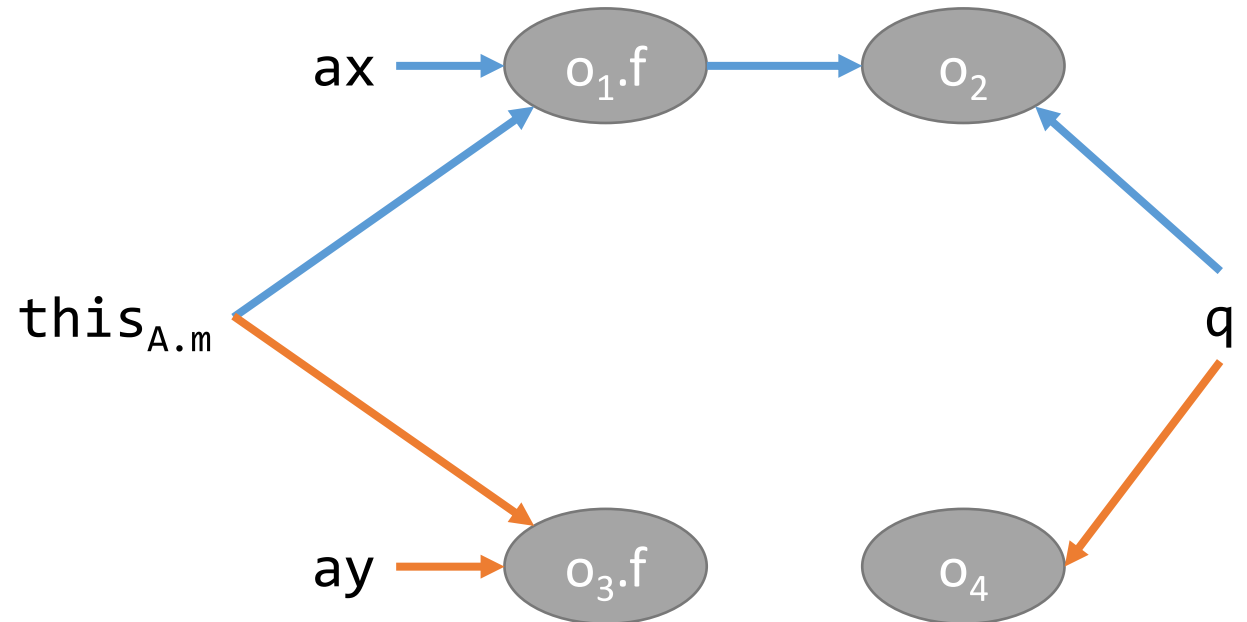
```
class Y extends X {…}
class A {
    X f;
    void m(X q)
      { this.f = q; }
}

A ax = new A(); // o1
ax.m(new X());  // o2
A ay = new A(); // o3
ay.m(new Y());  // o4
```



Example from Dr. Ryder's presentation

Example: Context Sensitivity

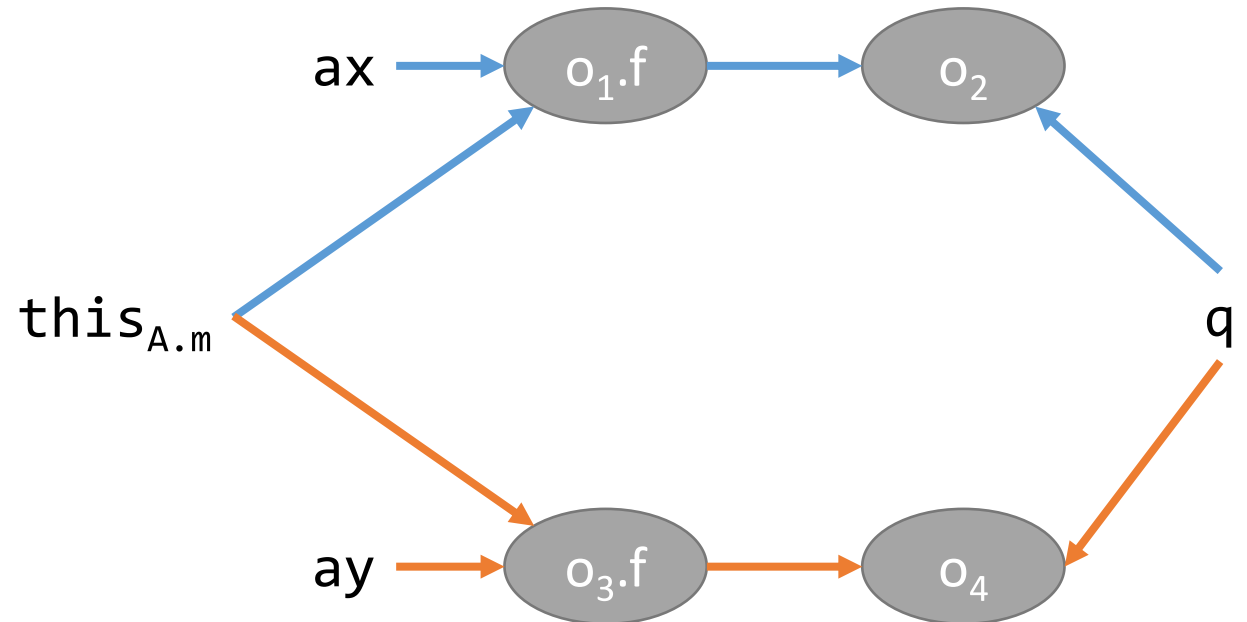
```
class Y extends X {…}
class A {
    X f;
    void m(X q)
      { this.f = q; }
}

A ax = new A(); // o1
ax.m(new X());  // o2
A ay = new A(); // o3
ay.m(new Y());  // o4
```



Example from Dr. Ryder's presentation

Example: Context Sensitivity

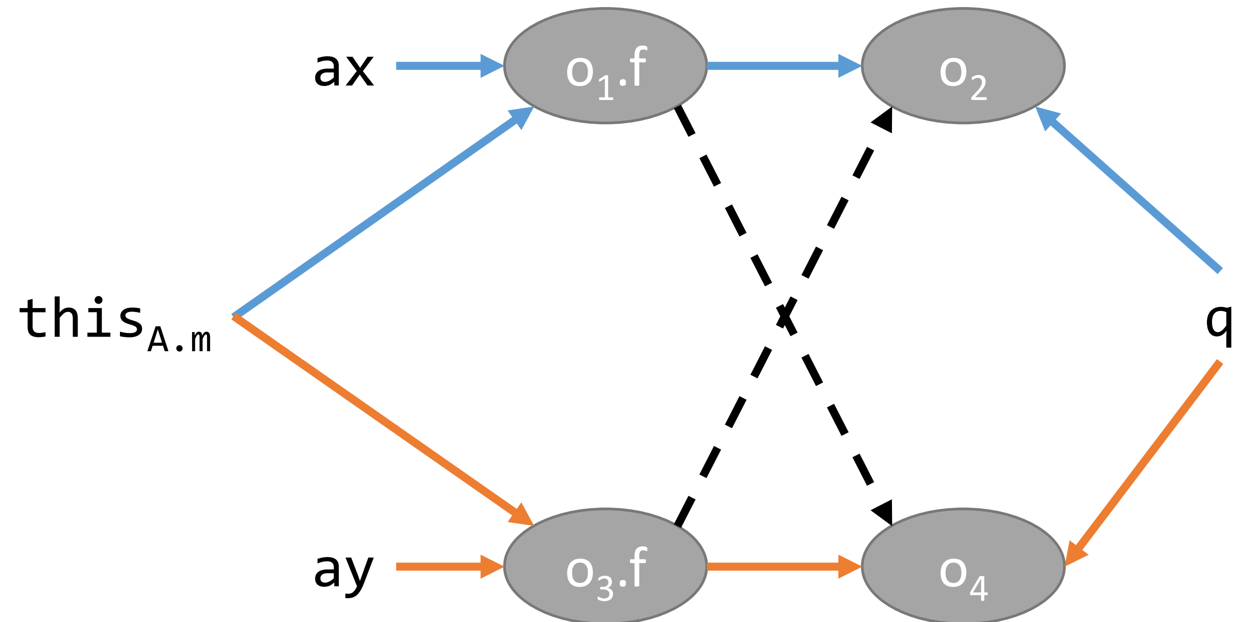
```
class Y extends X {…}
class A {
    X f;
    void m(X q)
      { this.f = q; }
}

A ax = new A(); // o1
ax.m(new X());  // o2
A ay = new A(); // o3
ay.m(new Y());  // o4
```



Example from Dr. Ryder's presentation

Example: Context Sensitivity

```
class Y extends X {…}
class A {
    X f;
    void m(X q)
      { this.f = q; }
}

A ax = new A(); // o1
ax.m(new X());  // o2
A ay = new A(); // o3
ay.m(new Y());  // o4
```



Example from Dr. Ryder's presentation

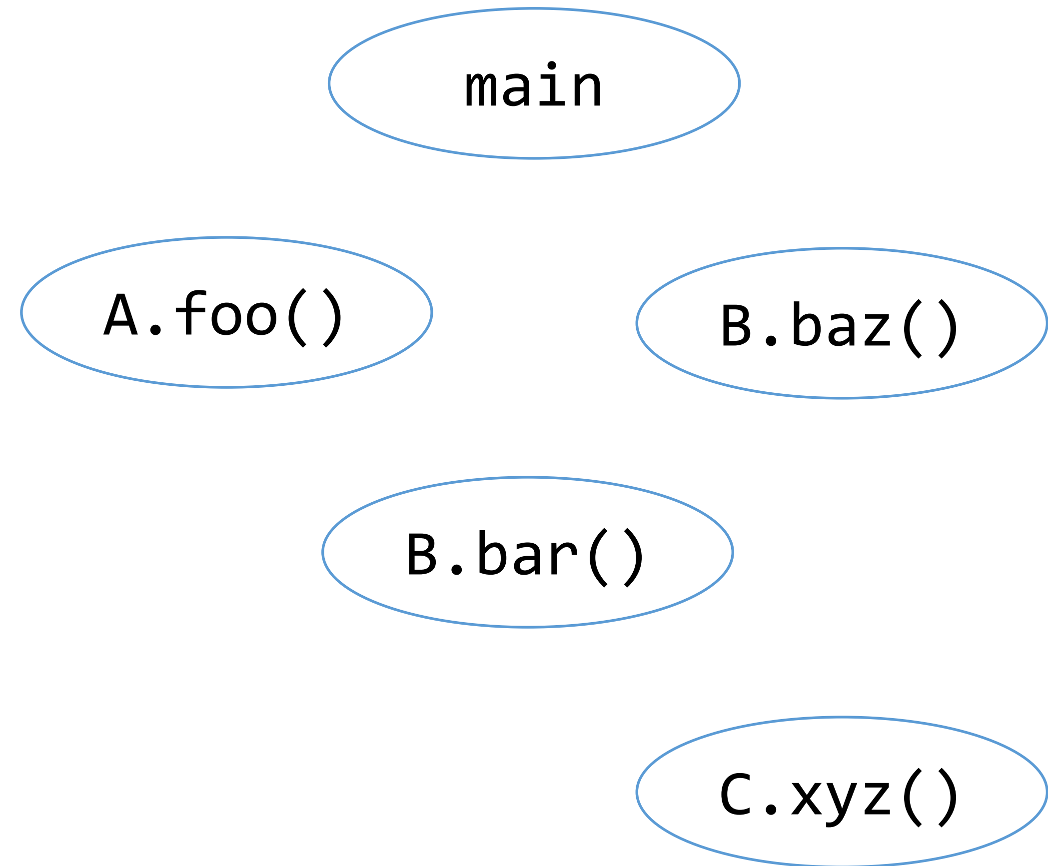
Program Representation (Calling Structure)

- Program calling structure is related to reference analysis solution
- Two approaches:
  - Approximate call graph and then do the reference analysis
  - Interleave reference analysis with call graph construction

- Lazy approach is preferred
  - Only reachable methods are in the call graph
  - Excluding unused methods improves cost and precision

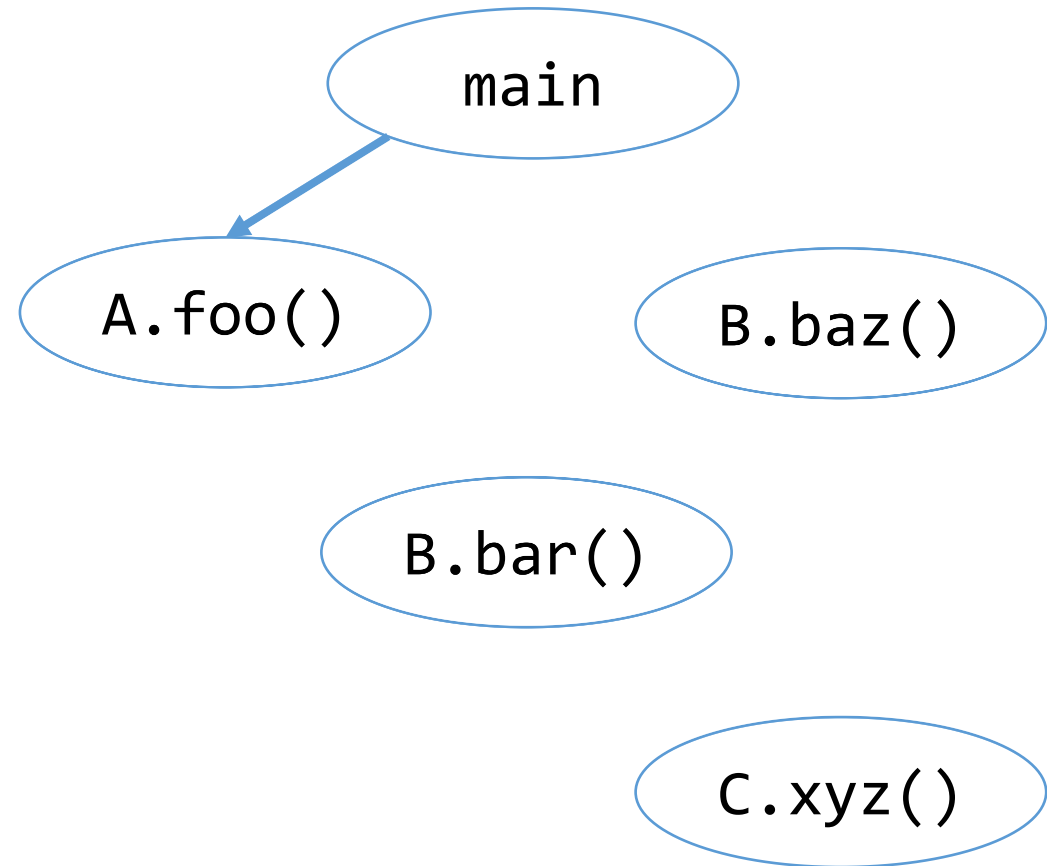
Example: Program Representation

```
class A {
    public void foo(B b)
      { b.bar(); }
}

class B {
    public void bar() {}
    public void baz()
      { bar(); C.xyz(); }

public static void main(…) {
    A a = new A();
    a.foo(new B());
}
```



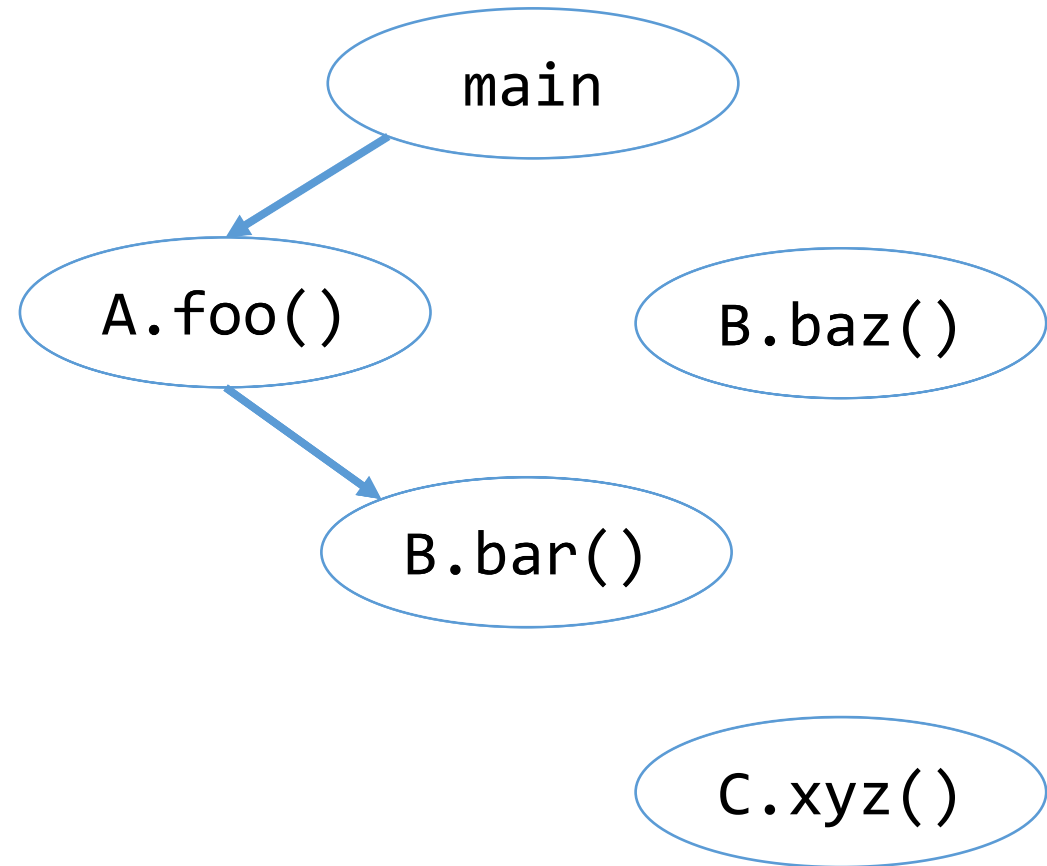
Example: Program Representation

```
class A {
    public void foo(B b)
      { b.bar(); }
}

class B {
    public void bar() {}
    public void baz()
      { bar(); C.xyz(); }

public static void main(…) {
    A a = new A();
    a.foo(new B());
}
```



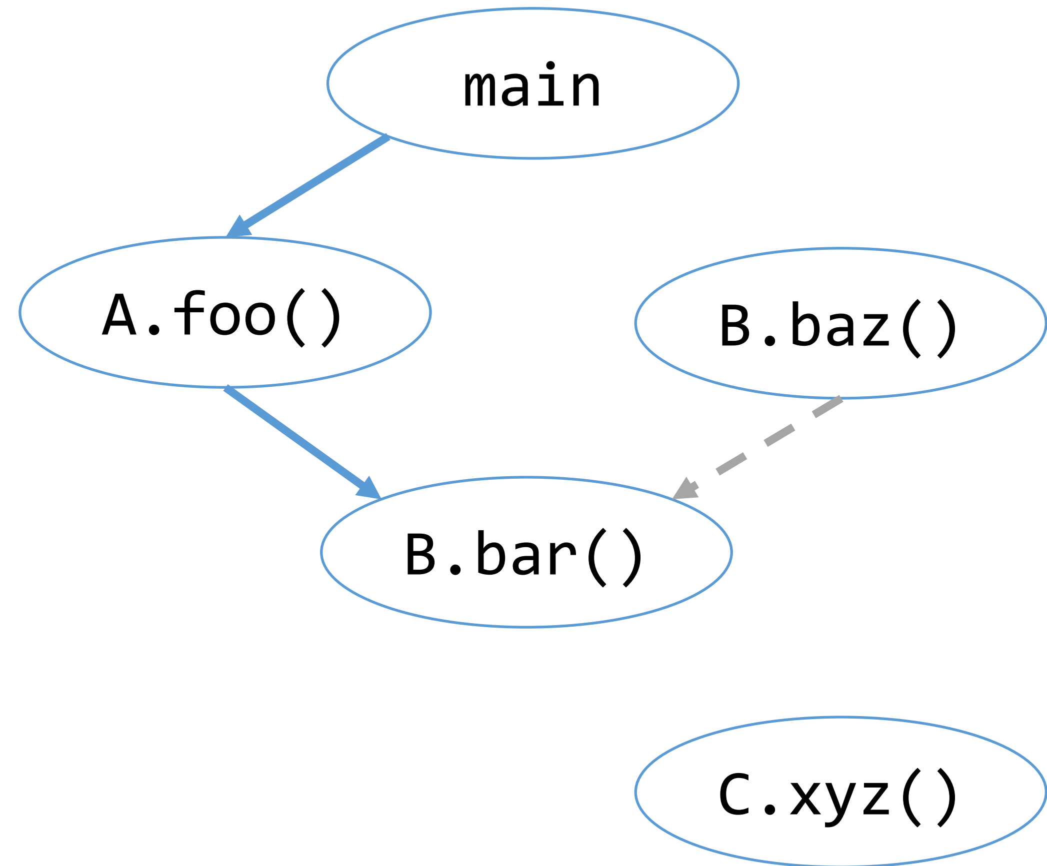
Example: Program Representation

```
class A {
    public void foo(B b)
      { b.bar(); }
}

class B {
    public void bar() {}
    public void baz()
      { bar(); C.xyz(); }

public static void main(…) {
    A a = new A();
    a.foo(new B());
}
```



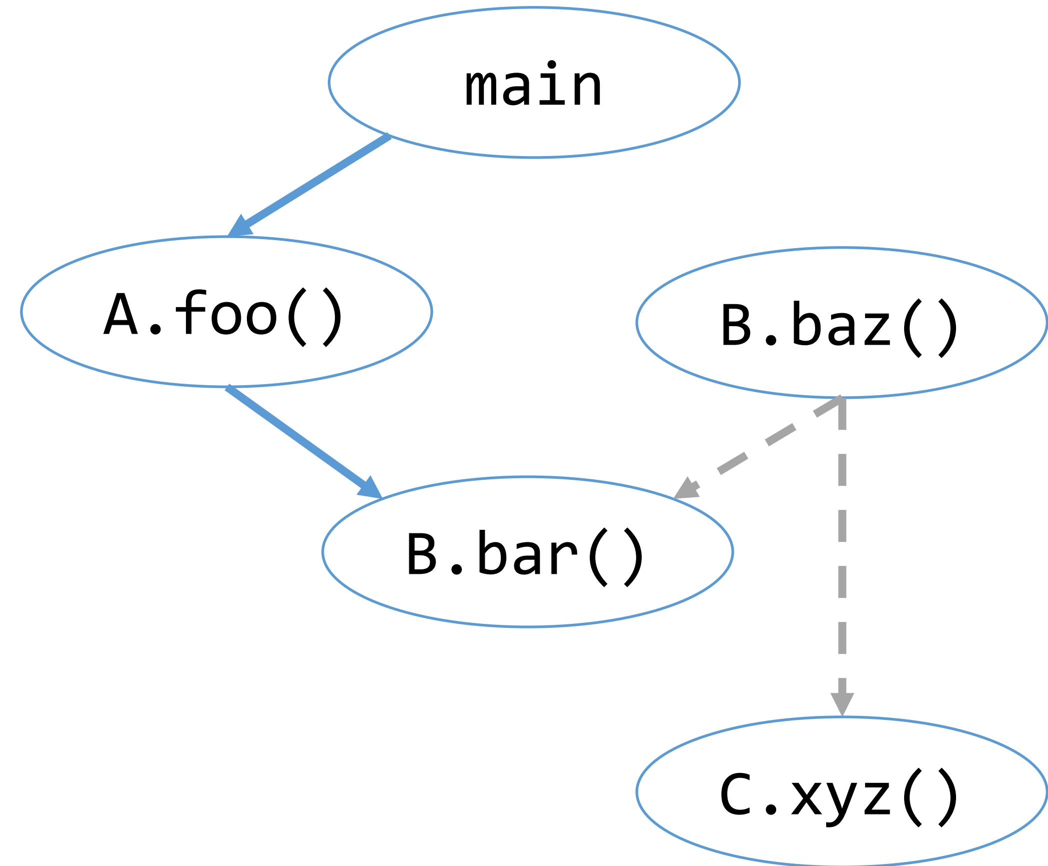
Example: Program Representation

```
class A {
    public void foo(B b)
      { b.bar(); }
}

class B {
    public void bar() {}
    public void baz()
      { bar(); C.xyz(); }

public static void main(…) {
    A a = new A();
    a.foo(new B());
}
```



Example: Program Representation

```
class A {
    public void foo(B b)
      { b.bar(); }
}

class B {
    public void bar() {}
    public void baz()
      { bar(); C.xyz(); }

public static void main(…) {
    A a = new A();
    a.foo(new B());
}
```



Object Representation

- Two common approaches for elements in the analysis solution
  - One abstract object for all instantiations of a class
  - One abstract object for each creation site of a class
- There are also other, more precise approaches

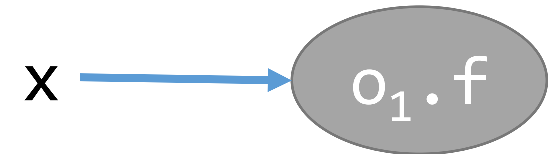
- One abstract object per class might be OK for call graph construction
  - May not be precise enough for other analyses

Example: One Abstract Object Per Class

```
class A {
    public B f;
    public void foo() {}
}
class B {}

A x = new A(); // o1
A y = new A(); // o1
A z = x;       // o1
y.f = new B()  // o2
```

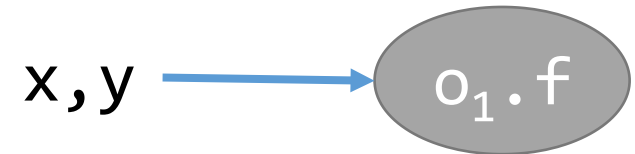
Example: One Abstract Object Per Class

```
class A {
    public B f;
    public void foo() {}
}
class B {}

A x = new A(); // o1
A y = new A(); // o1
A z = x;       // o1
y.f = new B()  // o2
```



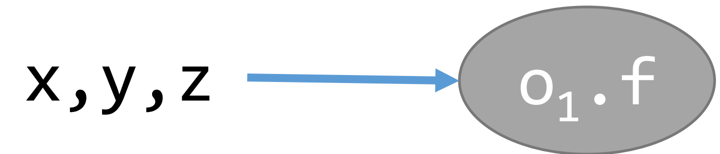
Example: One Abstract Object Per Class

```
class A {
    public B f;
    public void foo() {}
}
class B {}

A x = new A(); // o1
A y = new A(); // o1
A z = x;       // o1
y.f = new B()  // o2
```



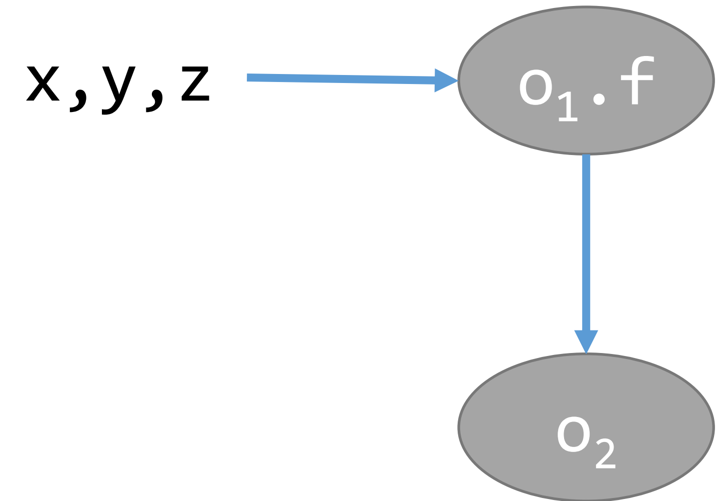
Example: One Abstract Object Per Class

```
class A {
    public B f;
    public void foo() {}
}
class B {}

A x = new A(); // o1
A y = new A(); // o1
A z = x;       // o1
y.f = new B()  // o2
```



Example: One Abstract Object Per Class

```
class A {
    public B f;
    public void foo() {}
}
class B {}

A x = new A(); // o1
A y = new A(); // o1
A z = x;       // o1
y.f = new B()  // o2
```

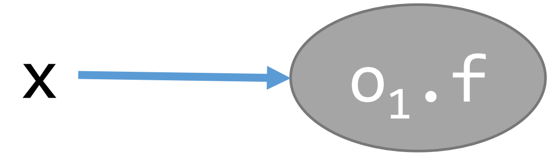


Example: One Abstract Object Per Creation Site

```
class A {
    public B f;
    public void foo() {}
}
class B {}

A x = new A(); // o1
A y = new A(); // o2
A z = x;       // o1
y.f = new B()  // o3
```

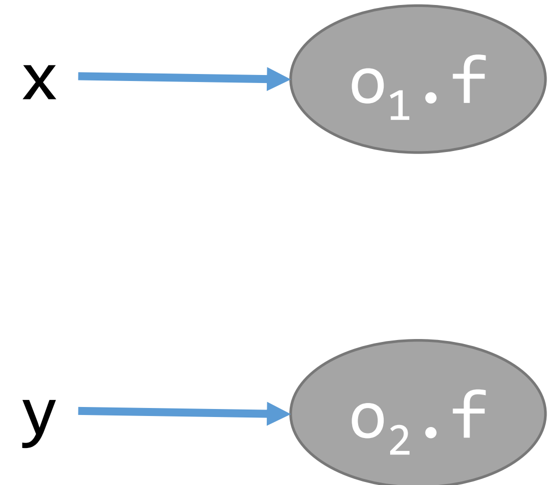
Example: One Abstract Object Per Creation Site

```
class A {
    public B f;
    public void foo() {}
}
class B {}

A x = new A(); // o1
A y = new A(); // o2
A z = x;       // o1
y.f = new B()  // o3
```



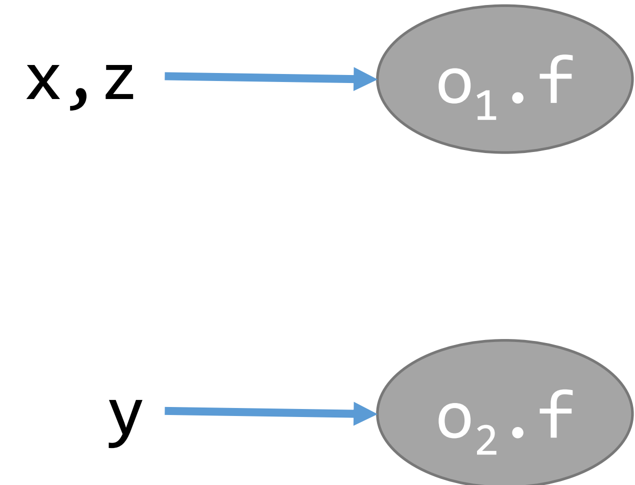
Example: One Abstract Object Per Creation Site

```
class A {
    public B f;
    public void foo() {}
}
class B {}

A x = new A(); // o1
A y = new A(); // o2
A z = x;       // o1
y.f = new B()  // o3
```



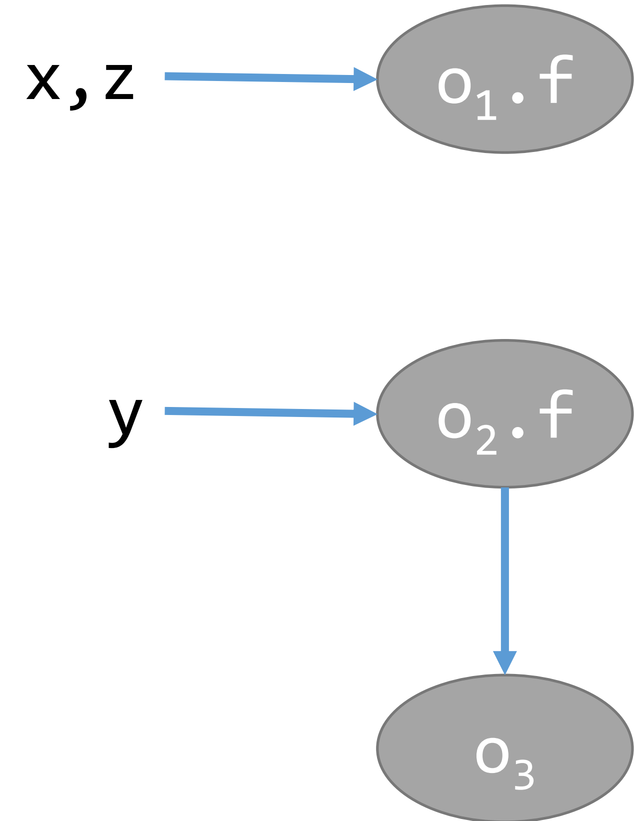
Example: One Abstract Object Per Creation Site

```
class A {
    public B f;
    public void foo() {}
}
class B {}

A x = new A(); // o1
A y = new A(); // o2
A z = x;       // o1
y.f = new B()  // o3
```



Example: One Abstract Object Per Creation Site

```
class A {
    public B f;
    public void foo() {}
}
class B {}

A x = new A(); // o1
A y = new A(); // o2
A z = x;       // o1
y.f = new B()  // o3
```



Field Sensitivity

An analysis is *field-sensitive* if its fields are distinctly represented in the solution. Otherwise, an analysis is *field insensitive*.

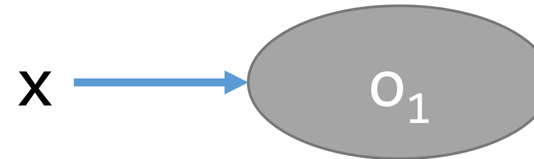
- Not distinguishing fields may decrease precision and *increase cost*
- But the interaction with other dimensions of precision is not clear
  - More evaluation is needed

Example: Field Insensitive Analysis

```
class A {
    public B f1;
    public C f2;
}
class B {}
class C { public D f; }
class D {}

A x = new A();      // o1
x.f1 = new B();     // o2
x.f2 = new C();     // o3
x.f2.f = new D(); // o4
```

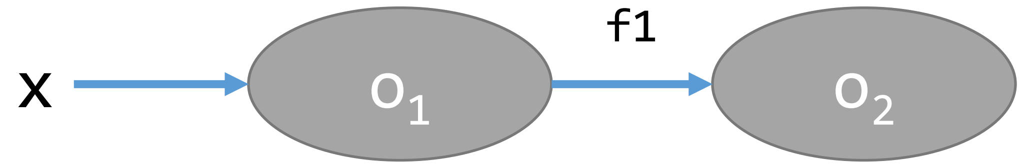
Example: Field Insensitive Analysis

```
class A {
    public B f1;
    public C f2;
}
class B {}
class C { public D f; }
class D {}

A x = new A();     // o1
x.f1 = new B();    // o2
x.f2 = new C();    // o3
x.f2.f = new D(); // o4
```



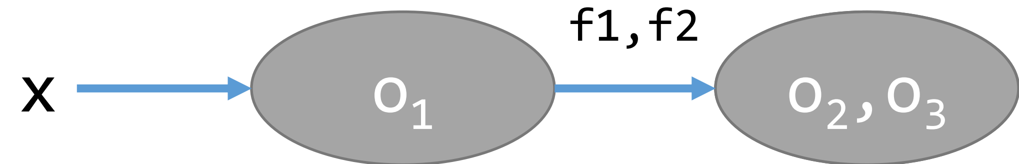
Example: Field Insensitive Analysis

```
class A {
    public B f1;
    public C f2;
}
class B {}
class C { public D f; }
class D {}

A x = new A();     // o1
x.f1 = new B();    // o2
x.f2 = new C();    // o3
x.f2.f = new D(); // o4
```



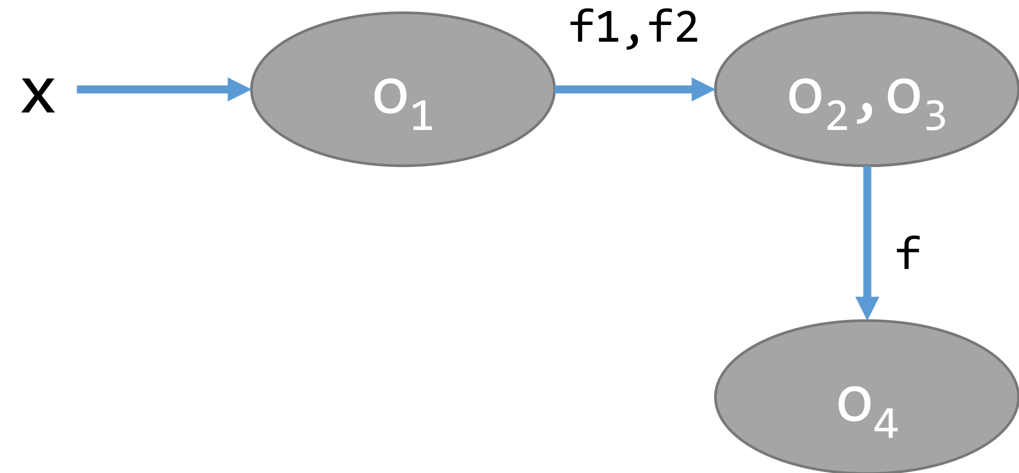
Example: Field Insensitive Analysis

```
class A {
    public B f1;
    public C f2;
}
class B {}
class C { public D f; }
class D {}

A x = new A();     // o1
x.f1 = new B();    // o2
x.f2 = new C();    // o3
x.f2.f = new D(); // o4
```



Example: Field Insensitive Analysis

```
class A {
    public B f1;
    public C f2;
}
class B {}
class C { public D f; }
class D {}

A x = new A();      // o1
x.f1 = new B();     // o2
x.f2 = new C();     // o3
x.f2.f = new D(); // o4
```

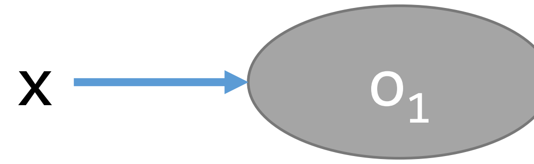


Example: Field Sensitive Analysis

```
class A {
    public B f1;
    public C f2;
}
class B {}
class C { public D f; }
class D {}

A x = new A();      // o1
x.f1 = new B();     // o2
x.f2 = new C();     // o3
x.f2.f = new D();   // o4
```

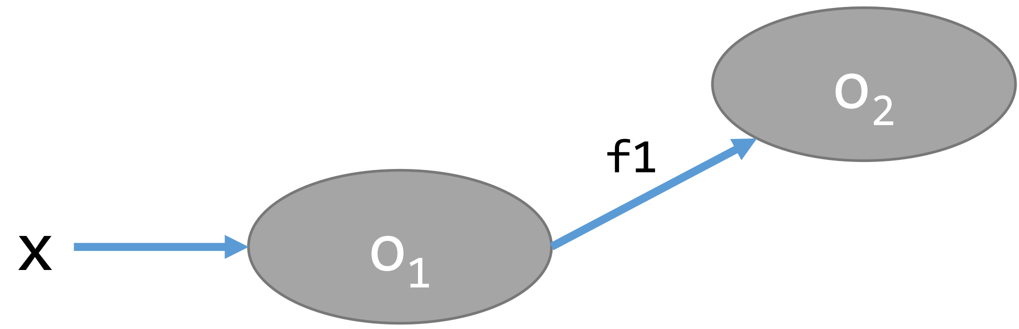
Example: Field Sensitive Analysis

```
class A {
    public B f1;
    public C f2;
}
class B {}
class C { public D f; }
class D {}

A x = new A();     // o1
x.f1 = new B();    // o2
x.f2 = new C();    // o3
x.f2.f = new D(); // o4
```



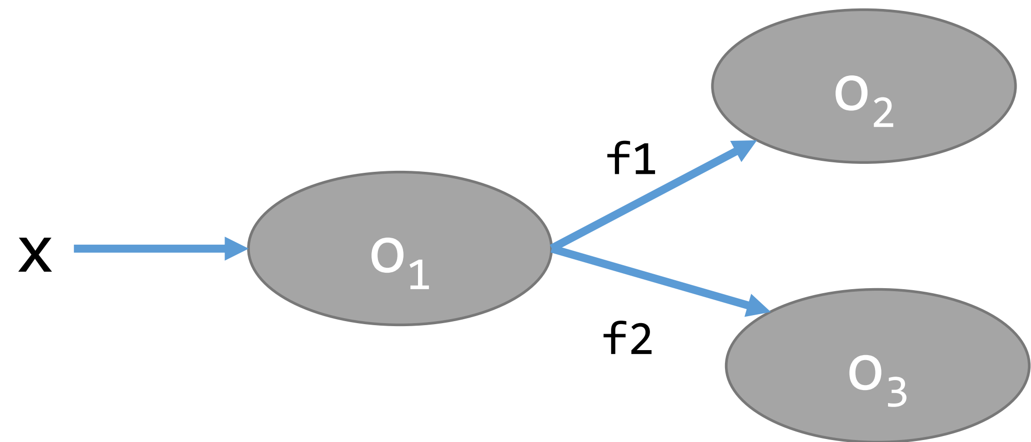
Example: Field Sensitive Analysis

```
class A {
    public B f1;
    public C f2;
}
class B {}
class C { public D f; }
class D {}

A x = new A();      // o1
x.f1 = new B();     // o2
x.f2 = new C();     // o3
x.f2.f = new D();   // o4
```



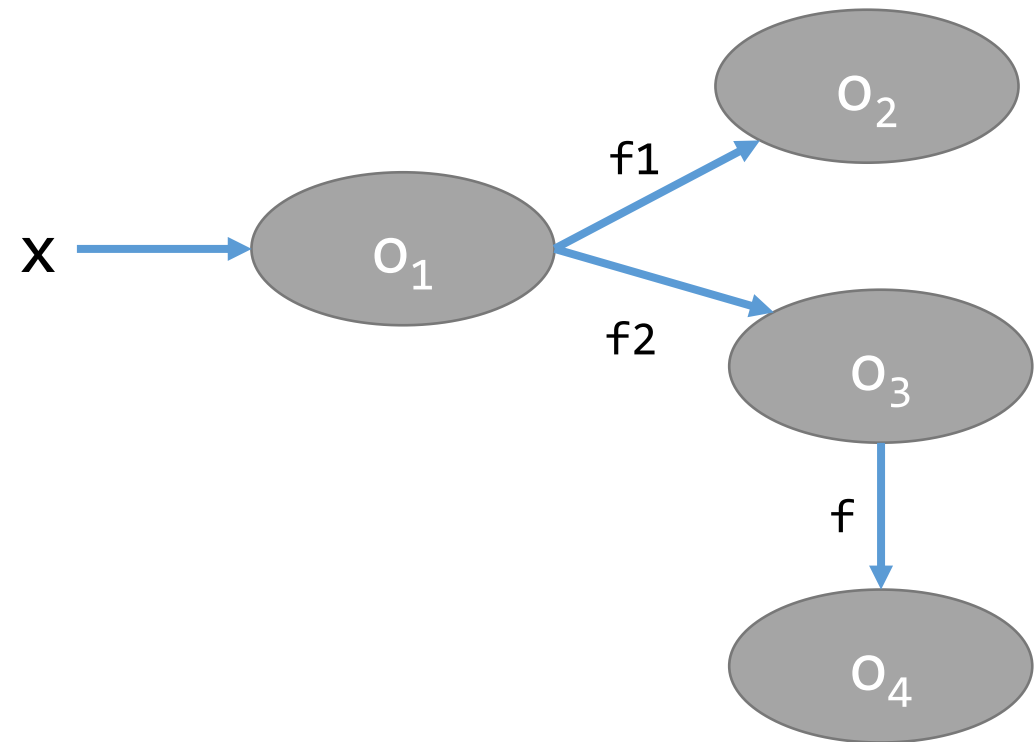
Example: Field Sensitive Analysis

```
class A {
    public B f1;
    public C f2;
}
class B {}
class C { public D f; }
class D {}

A x = new A();      // o1
x.f1 = new B();     // o2
x.f2 = new C();     // o3
x.f2.f = new D();   // o4
```



Example: Field Sensitive Analysis

```
class A {
    public B f1;
    public C f2;
}
class B {}
class C { public D f; }
class D {}

A x = new A();     // o1
x.f1 = new B();    // o2
x.f2 = new C();    // o3
x.f2.f = new D(); // o4
```



Reference Representation

- Generally, each reference has a unique representative
- Alternative approaches:
  - One abstract reference per type
  - One abstract reference per method
- Fewer references are less precise, but the analysis is more efficient

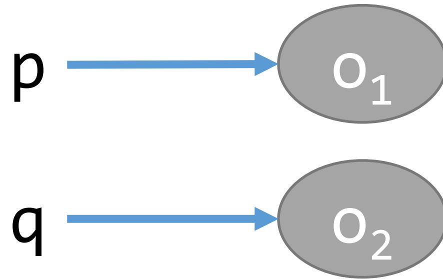
- Examples: CTA, FTA/MTA, XTA

Directionality

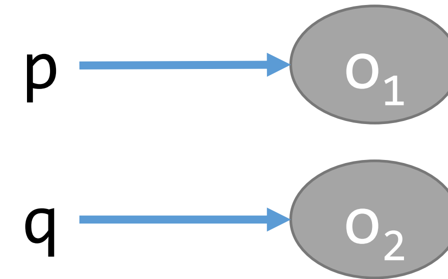
- How does the analysis interpret assignments (`p = q`)?
- Symmetric (unification constraints)
  - p and q have the same information after the assignment
  - E.g. Steensgaard's pointer analysis (worst case almost linear time)
- Directional (inclusion constraints)
  - Information flows from q to p
  - E.g. Andersen's pointer analysis (worst case cubic time)
- Inclusion more precise than unification, but slightly more cost

Example: Directionality

Unification Constraints

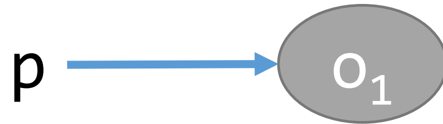


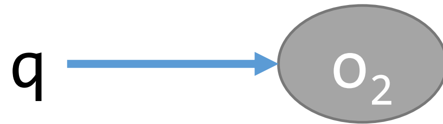
Inclusion Constraints



Example: Directionality

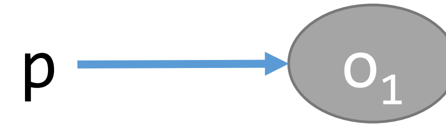
Unification Constraints

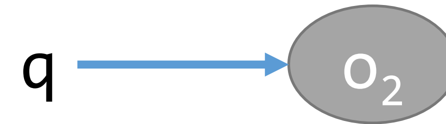




p = q

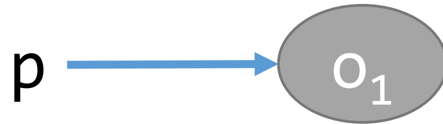
Inclusion Constraints

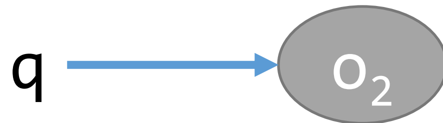




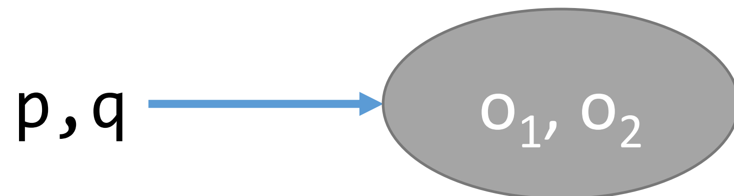
Example: Directionality

Unification Constraints

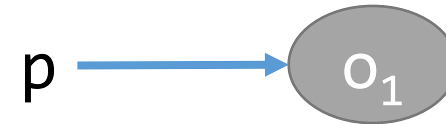


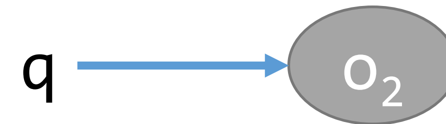


p = q



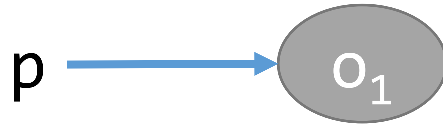
Inclusion Constraints

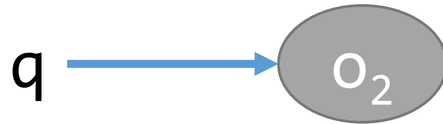




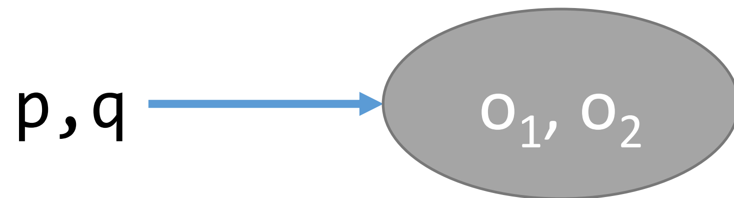
Example: Directionality

Unification Constraints

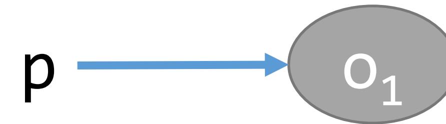


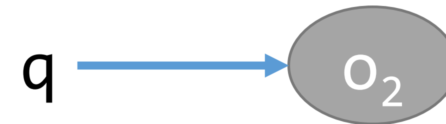


p = q



Inclusion Constraints

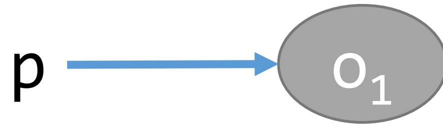


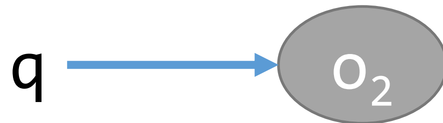


p = q

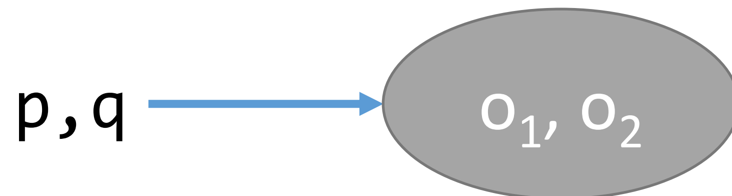
Example: Directionality

Unification Constraints

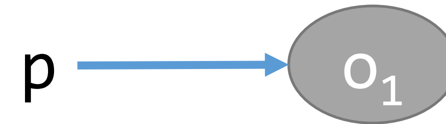


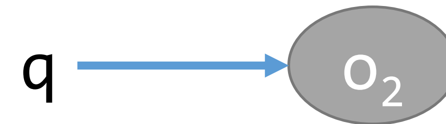


p = q

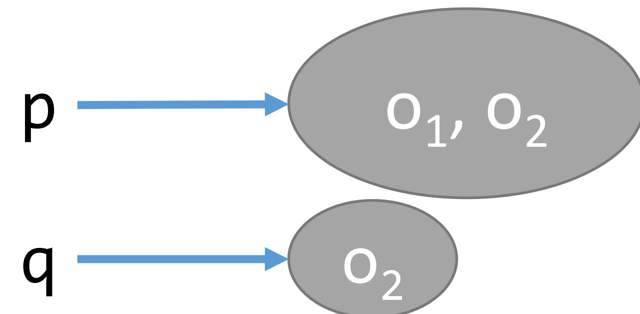


Inclusion Constraints





p = q



Open Issues

- Reflection
- Native methods
- Exceptions
- Dynamic class loading
- Incomplete programs
- Benchmarks

Conclusions

- Different dimensions affect the precision and cost of an analysis
  - Challenge: picking the right analysis for a specific application, and making the appropriate precision/cost trade-off
- Observations:
  - OO programs usually have many small methods, and method calls are primary control flow structure
  - Context sensitivity probably more useful than flow sensitivity
  - Inclusion constraints more precise than unification, and still practical
- No single analysis works for all applications

Discussion

- Some of the approaches seem to be “obviously” better than others. Are there cases where this might not be the case?

- Not all of the dimensions are binary. Could we use some hybrid approach?
  - E.g. part of an analysis is flow sensitive, the rest is flow insensitive

- What are other dimensions of precision in a reference analysis?