Object-Sensitive Points-to Analysis

Ondřej Lhoták

September 20, 2002

References

- Milanova, Rountev, Ryder. Parameterized Object Sensitivity for Points-to and Side-Effect Analyses for Java. ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA'02).
- Milanova, Rountev, Ryder. Points-to Analysis for Java Based on Annotated Constraints. OOPSLA 01.
- Andersen. Program Analysis and Specialization for the C Programming Language. PhD thesis, DIKU, University of Copenhagen, 1994.
- Emami, Ghiya, Hendren. Context-sensitive interprocedural points-to analysis in the presence of function pointers. PLDI 94.

Outline

- Quick Introduction to Points-to Analysis
- Sources of Imprecision in Context-Insensitive Analysis
- Object-Sensitive Points-to Analysis
- Experimental Results
- Conclusions

Points-to Analysis

- Goal: approximate the set of run-time objects to which a pointer may point

Points-to Analysis

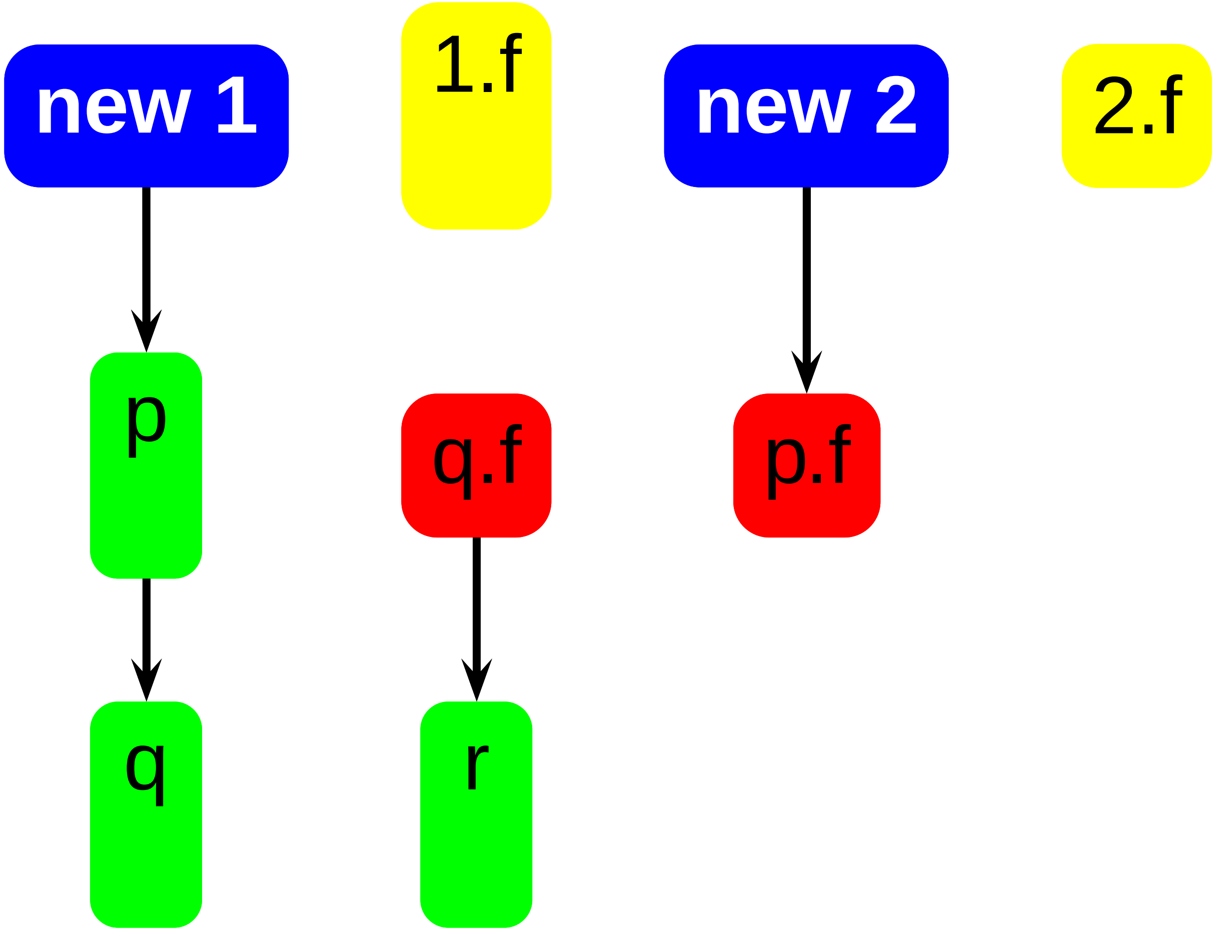
- Goal: approximate the set of run-time objects to which a pointer may point
- Common uses:
  - Side-effect information

    ```
    x.f = 1; y.f = 0; z = 1/x.f;
    ```

  - Virtual call resolution

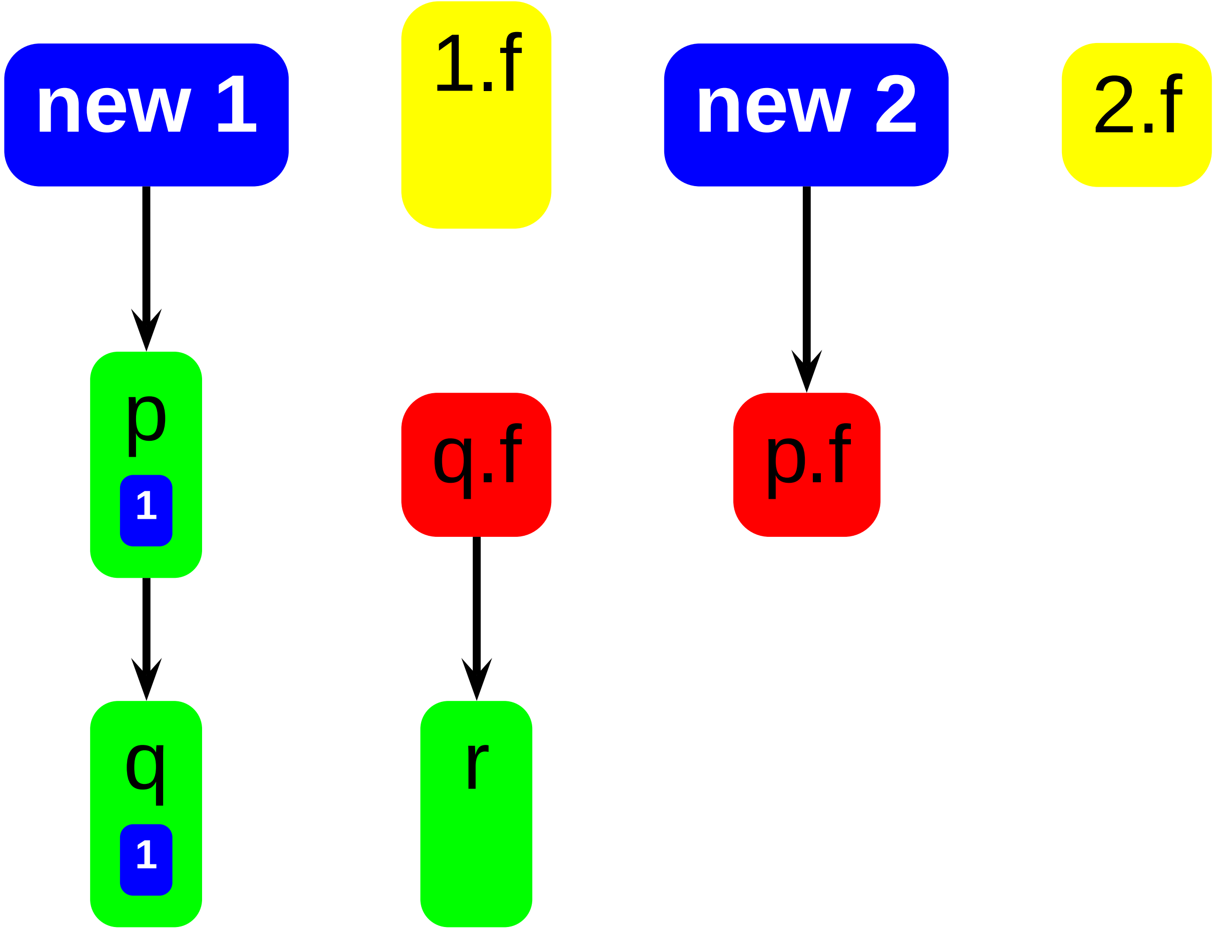
    ```
    String s = o.toString();
    ```

Pointer Assignment Graph



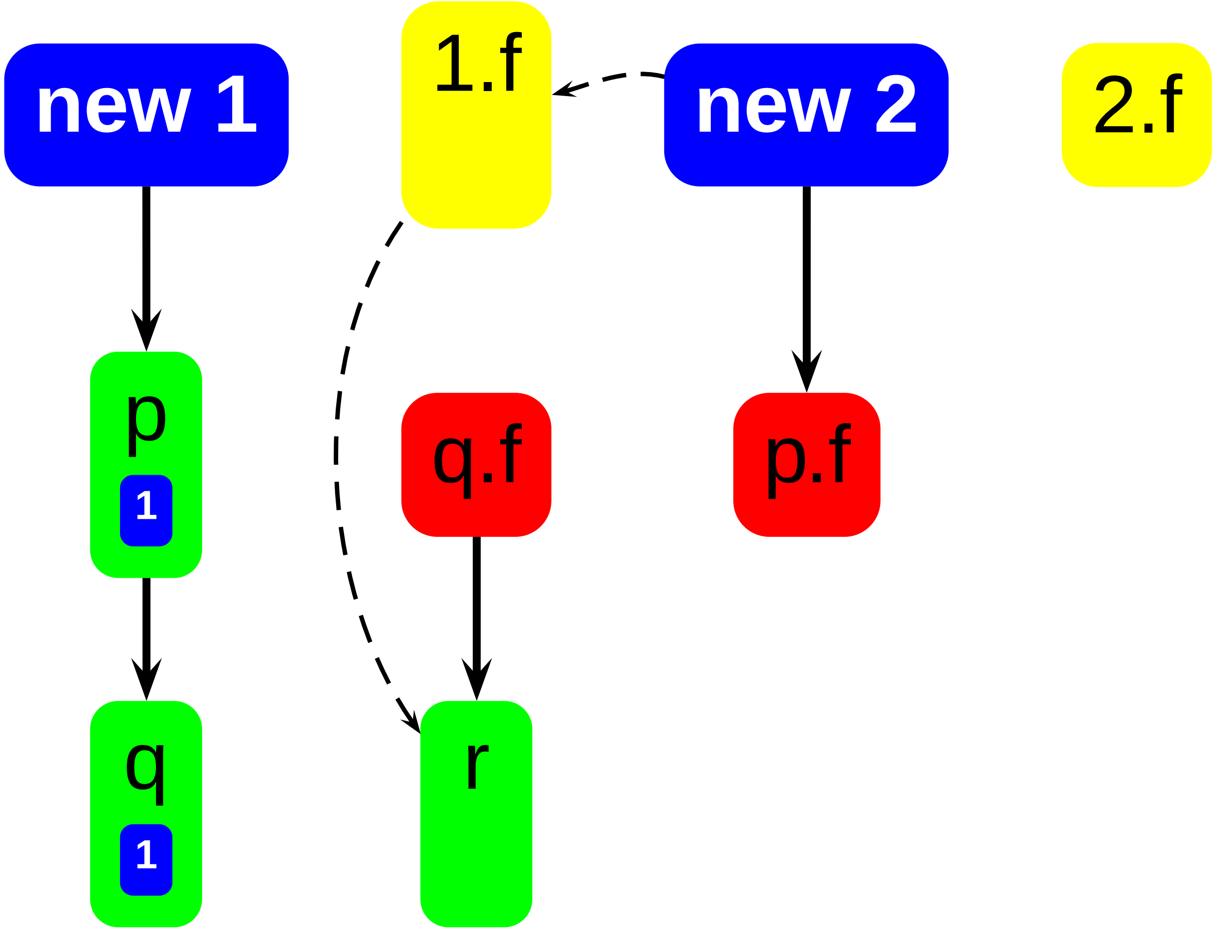
```
p = new 1;
q = p;
p.f = new 2;
r = q.f;
```

Pointer Assignment Graph



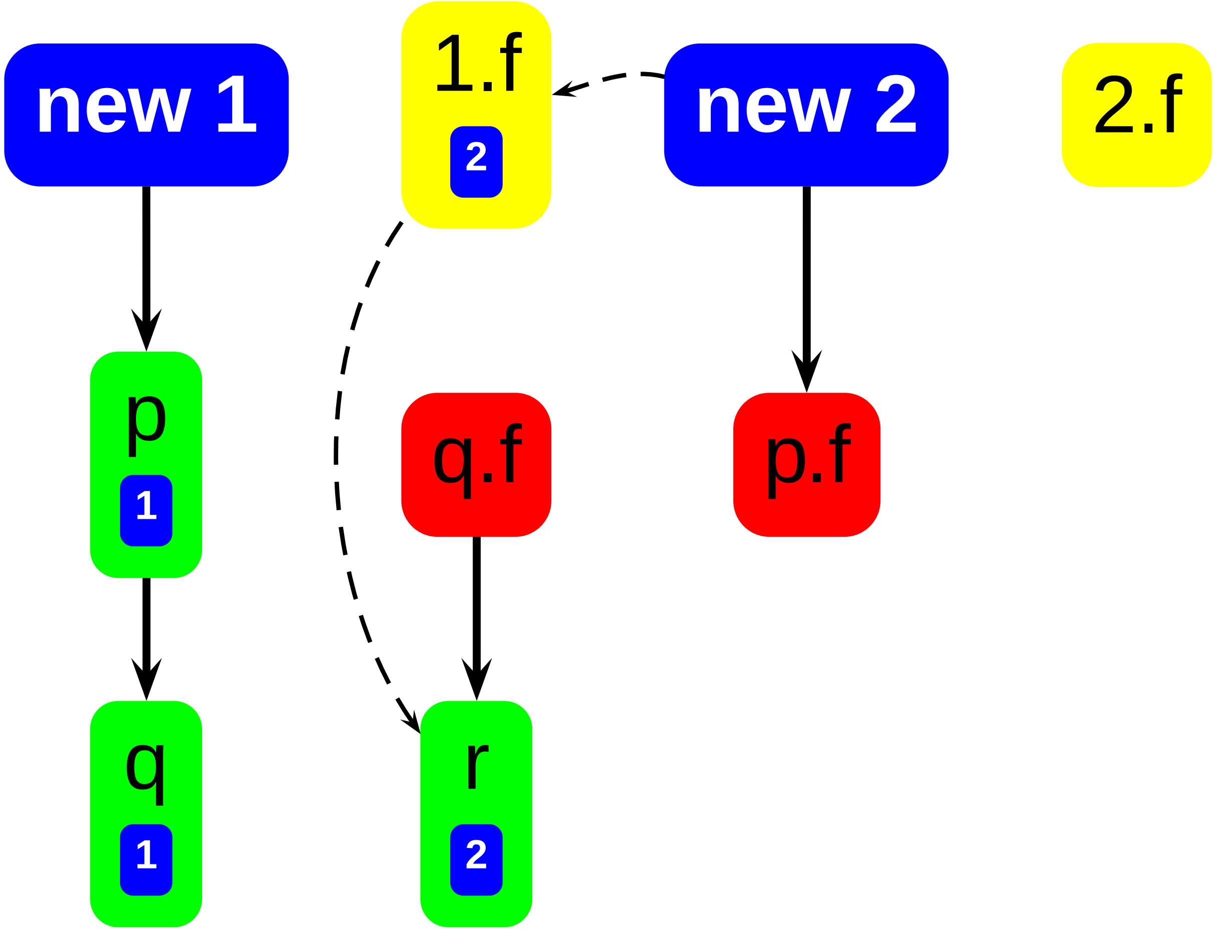
```
p = new 1;
q = p;
p.f = new 2;
r = q.f;
```

Pointer Assignment Graph



```
p = new 1;
q = p;
p.f = new 2;
r = q.f;
```

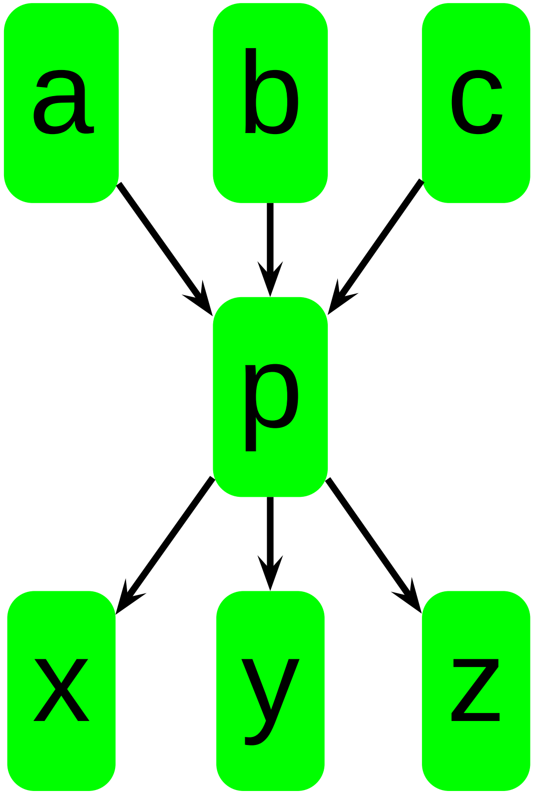
Pointer Assignment Graph



```
p = new 1;
q = p;
p.f = new 2;
r = q.f;
```

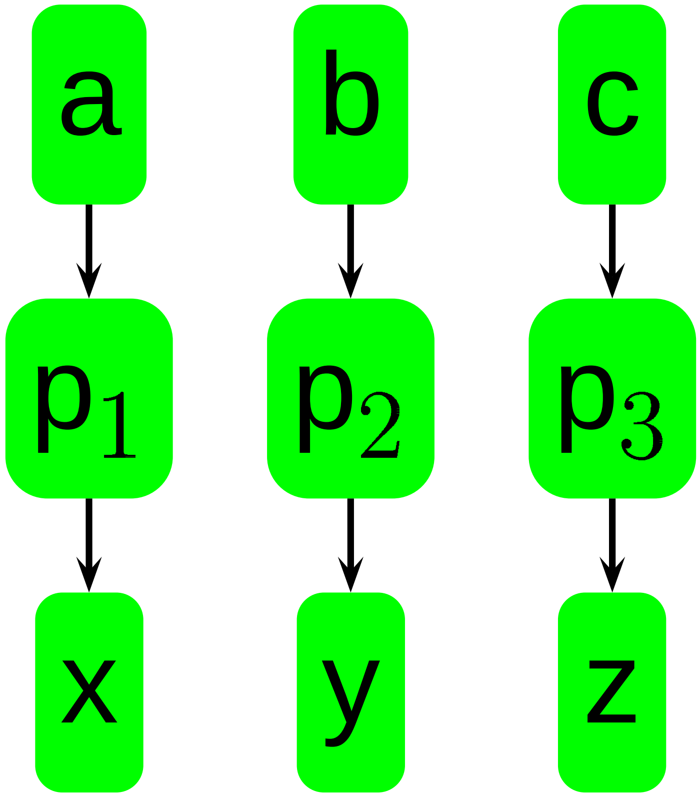
Method Calls

```
static
Obj f( Obj p ) {
    return p;
}
x = f( a );
y = f( b );
z = f( c );
```



Context Sensitivity

```
static
Obj f( Obj p ) {
    return p;
}
1: x = f( a );
2: y = f( b );
3: z = f( c );
```



Sources of Imprecision

- Encapsulation

```
x1 = new 1;
x2 = new 2;
y3 = new 3;
y4 = new 4;
x1.f = y3;
x2.f = y4;
```

```
x1 = new 1;
x2 = new 2;
y3 = new 3;
y4 = new 4;
x1.set( y3 );
x2.set( y4 );
```

Sources of Imprecision

- Inheritance

```
class A {
    O f;
    A(O a) { this.f = a; }
};
class B extends A {
    B(O b) { super(b); }
};
class C extends A {
    C(O c) { super(c); }
};
y = new B(x).f;
z = new C(w).f;
```

Sources of Imprecision

- Collections and Maps

```
class Vector {
    Object[] data;
    Vector( int n ) {
        data = new Object[n];
    }
};
v1 = new Vector(5);
v2 = new Vector(5);
v1.add(x);
v2.add(y);
```

Object-Sensitive Analysis

- Local variables in instance (non-static) methods become *fields of this* (p becomes this.p)
- Objects parameterized by allocation site of *this* at allocation, as well as allocation site (**new 1** becomes **new 2** **new 1**)

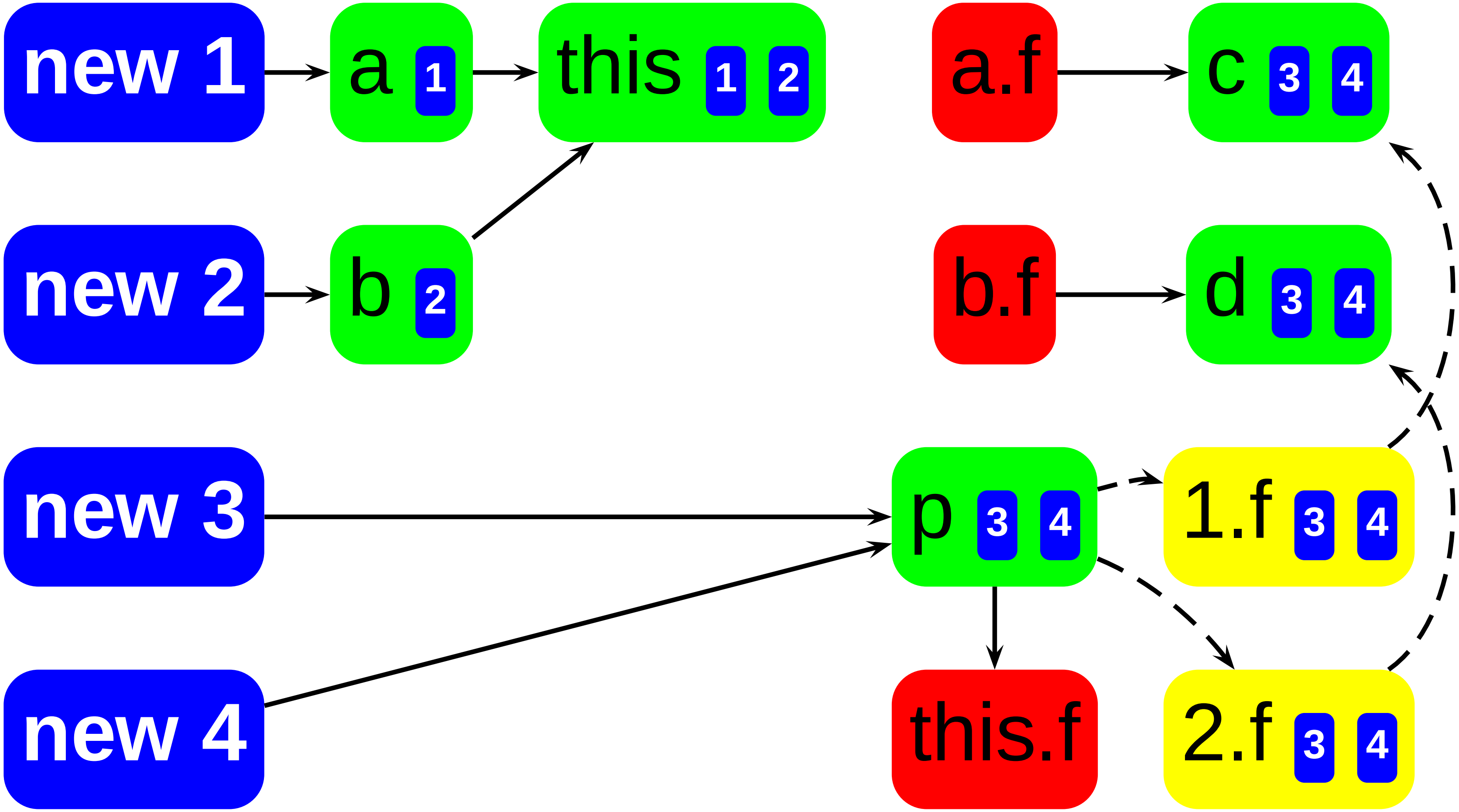
Object-Sensitive Analysis

- **Local variables in instance (non-static) methods become** *fields of this* **(p becomes this.p)**
- Objects parameterized by allocation site of *this* at allocation, as well as allocation site (**new 1** becomes **new 2** **new 1**)

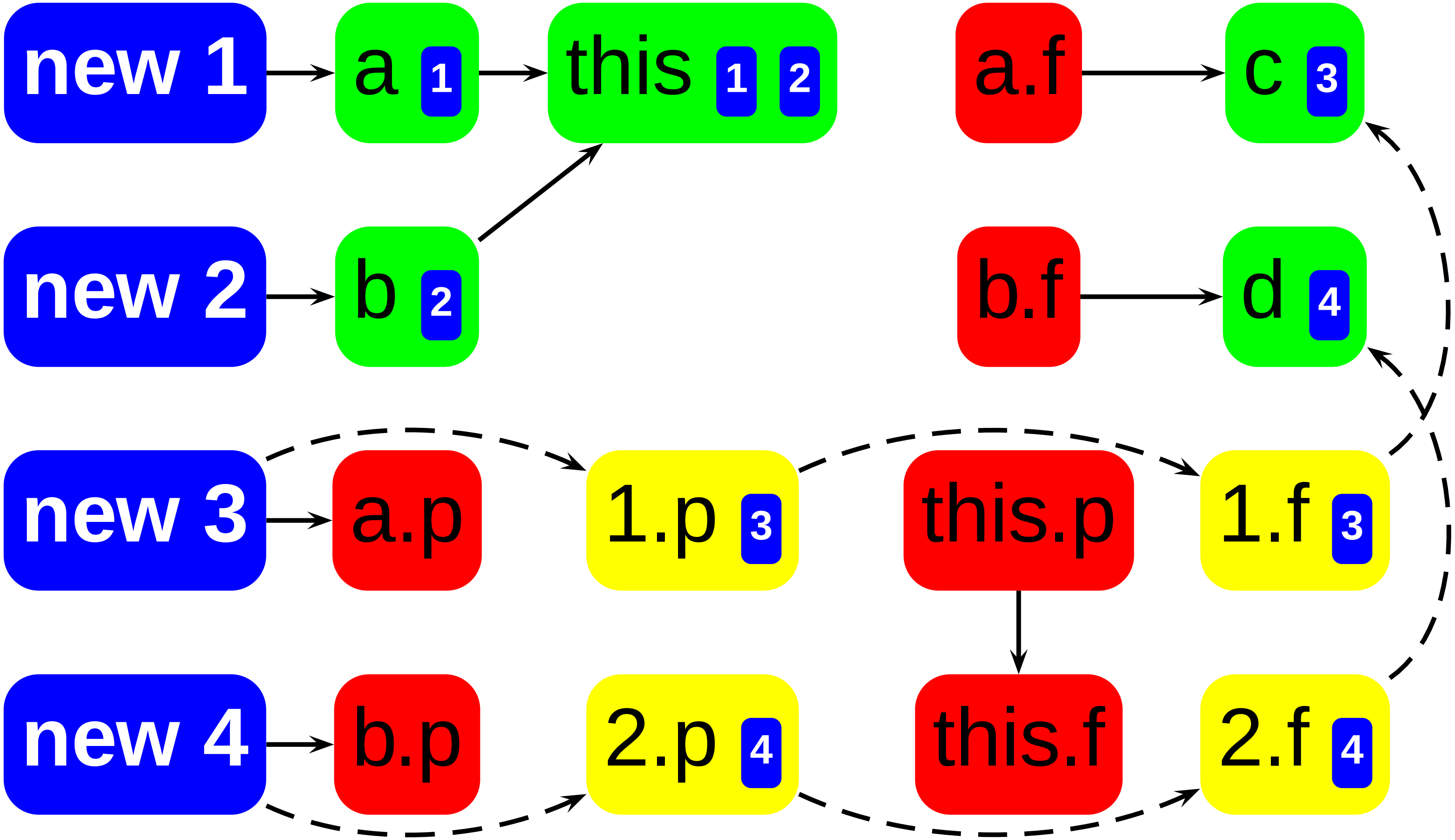
Object-Sensitive Analysis

```
set( O p ) {
    this.f = p;
}
a = new 1;
b = new 2;
a.set( new 3 );
b.set( new 4 );
c = a.f;
d = b.f;
```

Object-Sensitive Analysis



Object-Sensitive Analysis



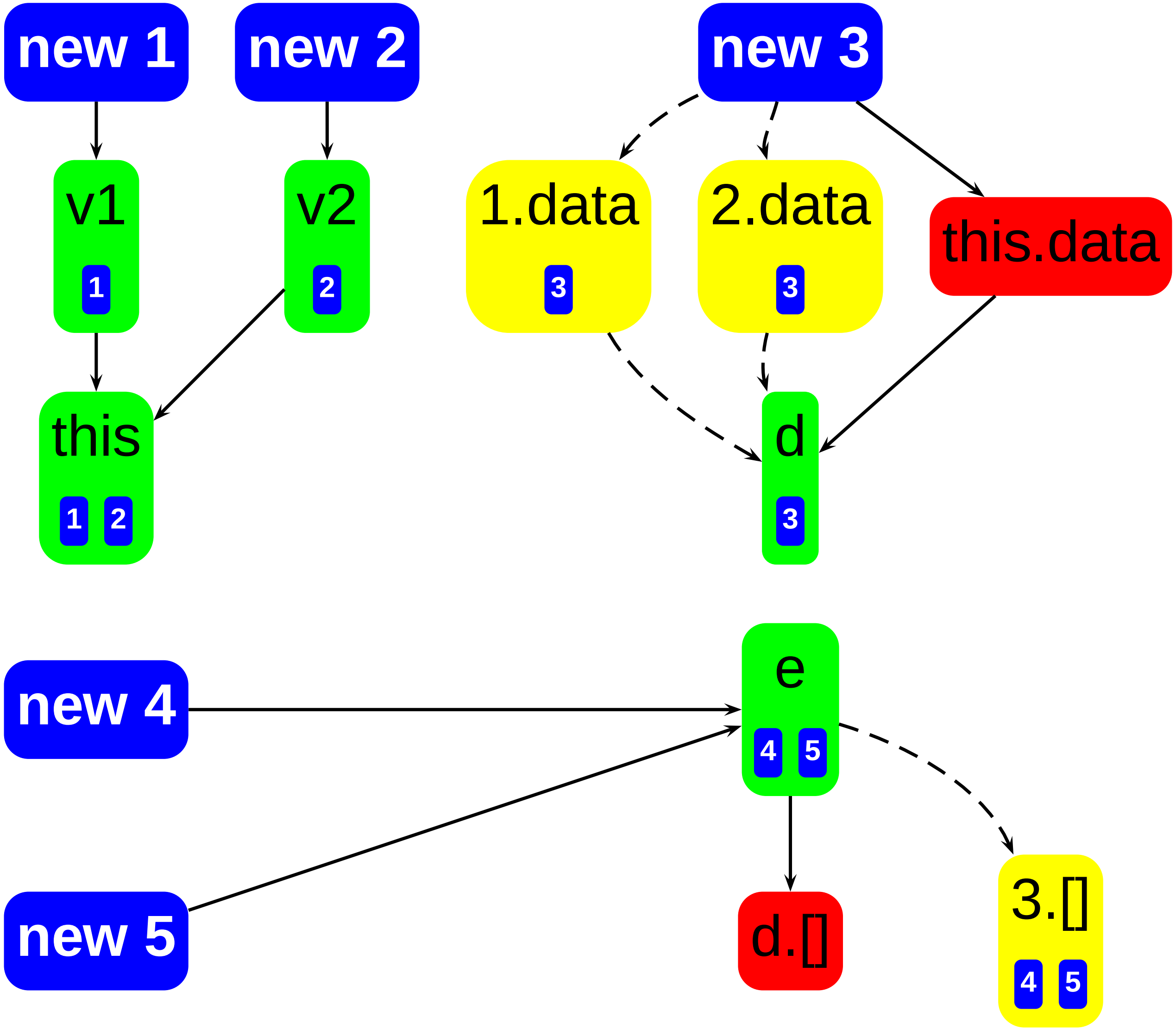
Object-Sensitive Analysis

- Local variables in instance (non-static) methods become *fields of this* (p becomes this.p)
- **Objects parameterized by allocation site of** *this* **at allocation, as well as allocation site (new 1 becomes new 2 new 1)**

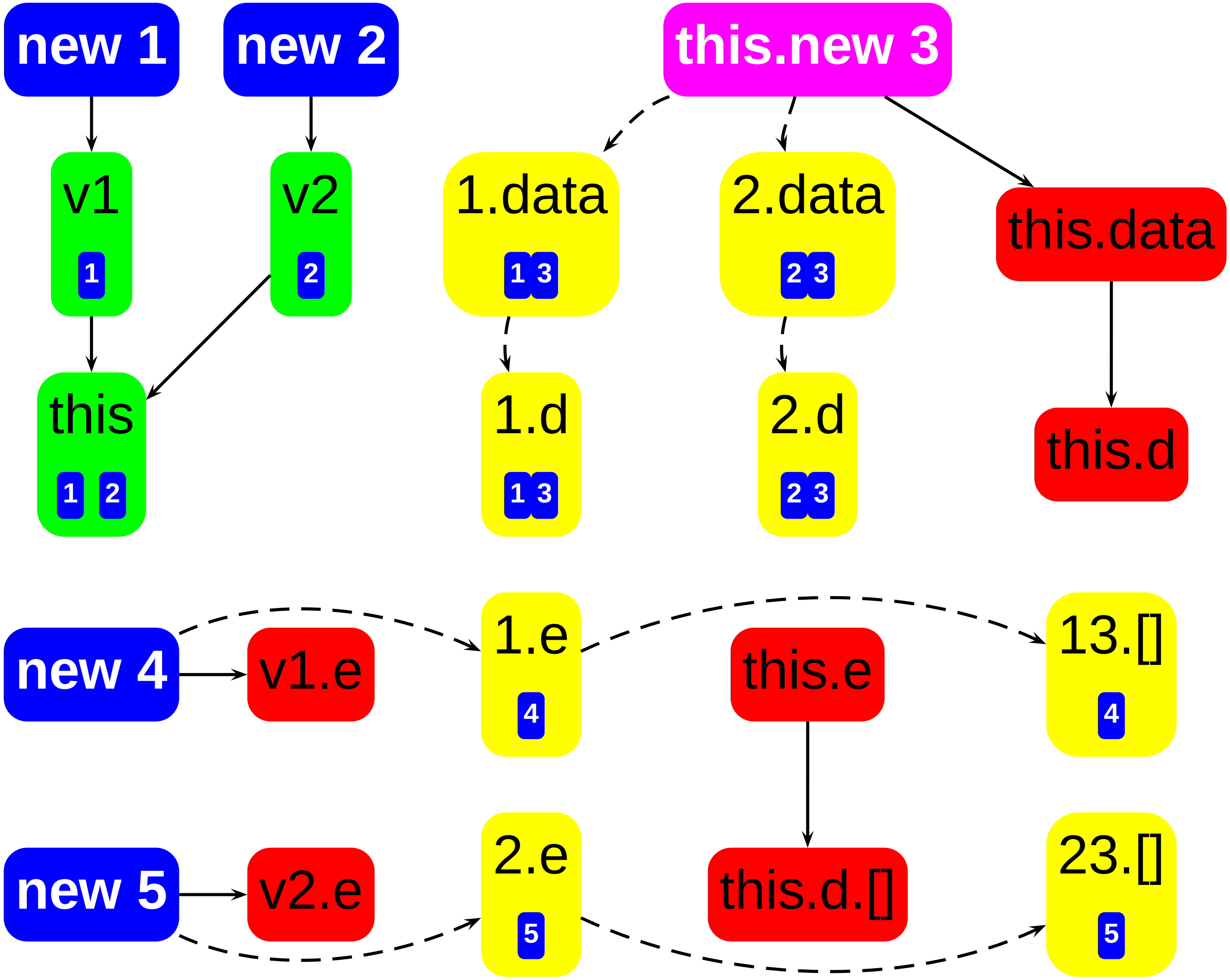
Object-Sensitive Analysis

```
Vector() {
    this.data = new 3;
}
add(Obj e) {
    d = this.data;
    d[0] = e;
}
v1 = new 1();
v2 = new 2();
v1.add( new 4 );
v2.add( new 5 );
```

Object-Sensitive Analysis



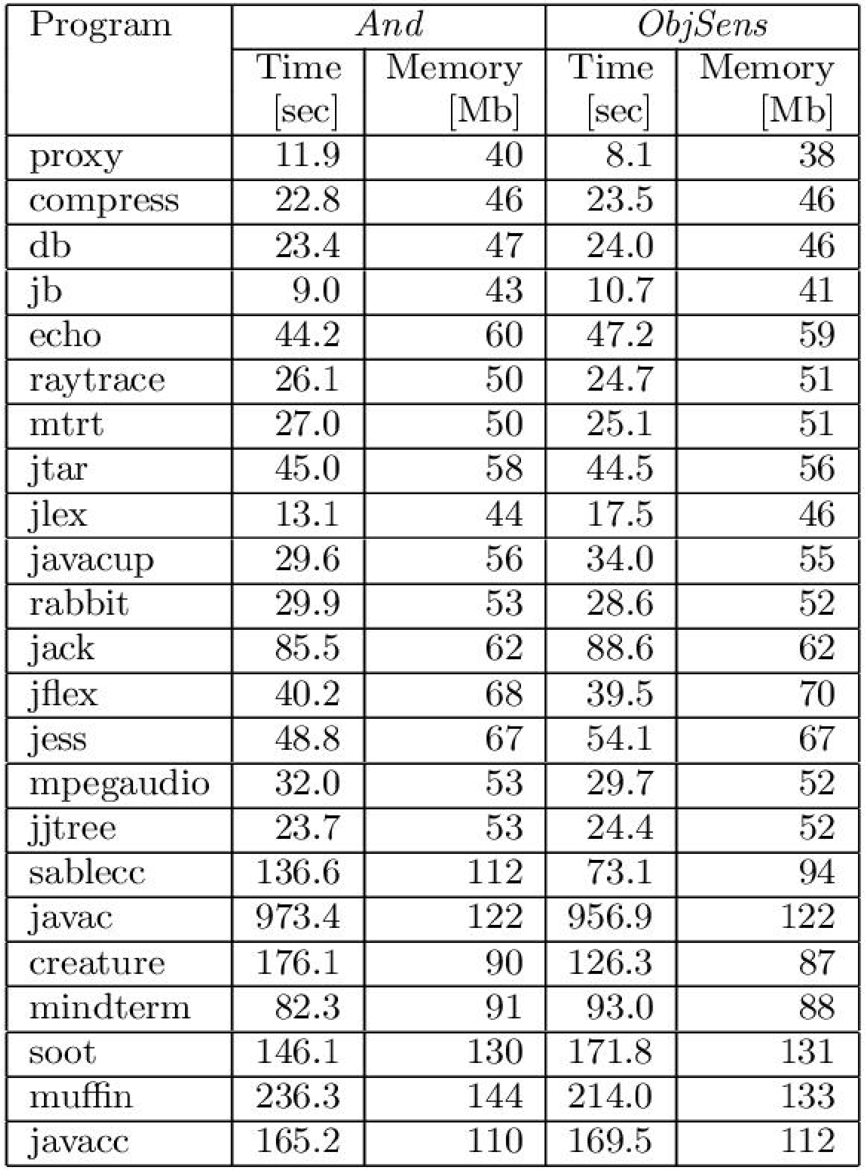
Object-Sensitive Analysis



Experimental Results

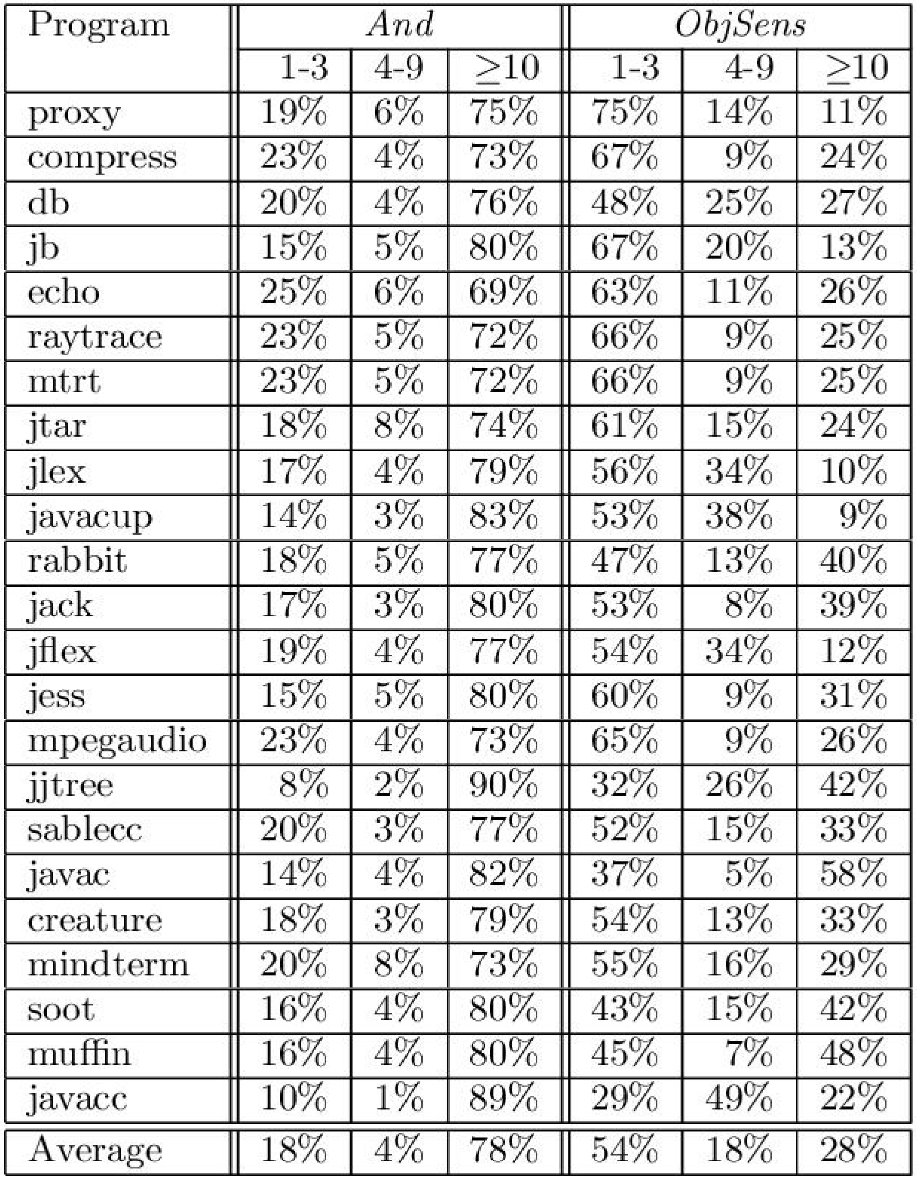
- Experiments used BANE solver.
- Experiments tested a partial instantiation of object sensitivity:
  - Formal parameters, return values, this represented by red nodes.
  - No purple nodes.

Analysis Time and Memory



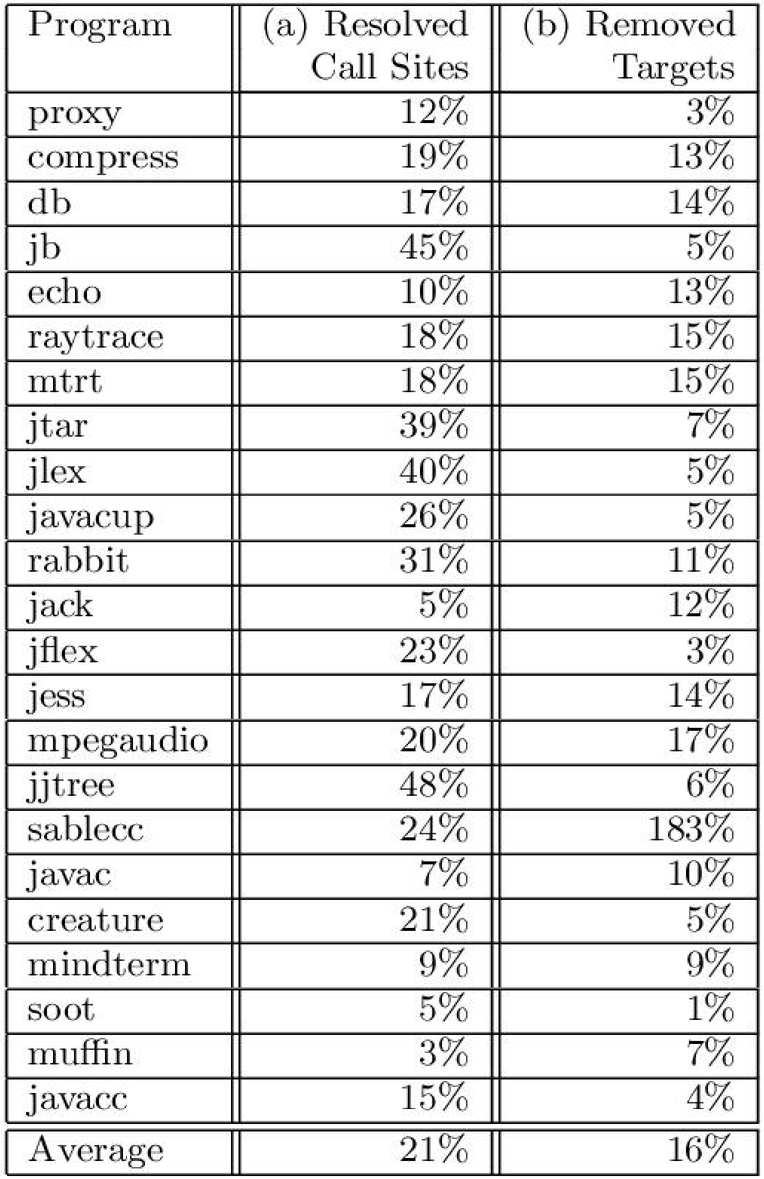
Program	*And*		*ObjSens*	
	Time [sec]	Memory [Mb]	Time [sec]	Memory [Mb]
proxy	11.9	40	8.1	38
compress	22.8	46	23.5	46
db	23.4	47	24.0	46
jb	9.0	43	10.7	41
echo	44.2	60	47.2	59
raytrace	26.1	50	24.7	51
mtrt	27.0	50	25.1	51
jtar	45.0	58	44.5	56
jlex	13.1	44	17.5	46
javacup	29.6	56	34.0	55
rabbit	29.9	53	28.6	52
jack	85.5	62	88.6	62
jflex	40.2	68	39.5	70
jess	48.8	67	54.1	67
mpegaudio	32.0	53	29.7	52
jjtree	23.7	53	24.4	52
sablecc	136.6	112	73.1	94
javac	973.4	122	956.9	122
creature	176.1	90	126.3	87
mindterm	82.3	91	93.0	88
soot	146.1	130	171.8	131
muffin	236.3	144	214.0	133
javacc	165.2	110	169.5	112

Precision for Side-Effect Analysis



Program	*And*			*ObjSens*		
	1-3	4-9	≥10	1-3	4-9	≥10
proxy	19%	6%	75%	75%	14%	11%
compress	23%	4%	73%	67%	9%	24%
db	20%	4%	76%	48%	25%	27%
jb	15%	5%	80%	67%	20%	13%
echo	25%	6%	69%	63%	11%	26%
raytrace	23%	5%	72%	66%	9%	25%
mtrt	23%	5%	72%	66%	9%	25%
jtar	18%	8%	74%	61%	15%	24%
jlex	17%	4%	79%	56%	34%	10%
javacup	14%	3%	83%	53%	38%	9%
rabbit	18%	5%	77%	47%	13%	40%
jack	17%	3%	80%	53%	8%	39%
jflex	19%	4%	77%	54%	34%	12%
jess	15%	5%	80%	60%	9%	31%
mpegaudio	23%	4%	73%	65%	9%	26%
jjtree	8%	2%	90%	32%	26%	42%
sablecc	20%	3%	77%	52%	15%	33%
javac	14%	4%	82%	37%	5%	58%
creature	18%	3%	79%	54%	13%	33%
mindterm	20%	8%	73%	55%	16%	29%
soot	16%	4%	80%	43%	15%	42%
muffin	16%	4%	80%	45%	7%	48%
javacc	10%	1%	89%	29%	49%	22%
Average	18%	4%	78%	54%	18%	28%

Precision for Call Graph



Program	(a) Resolved Call Sites	(b) Removed Targets
proxy	12%	3%
compress	19%	13%
db	17%	14%
jb	45%	5%
echo	10%	13%
raytrace	18%	15%
mtrt	18%	15%
jtar	39%	7%
jlex	40%	5%
javacup	26%	5%
rabbit	31%	11%
jack	5%	12%
jflex	23%	3%
jess	17%	14%
mpegaudio	20%	17%
jjtree	48%	6%
sablecc	24%	183%
javac	7%	10%
creature	21%	5%
mindterm	9%	9%
soot	5%	1%
muffin	3%	7%
javacc	15%	4%
Average	21%	16%

Conclusions

- Context-insensitivity is a major source of imprecision
- “Right” level of context-sensitivity need not slow down the analysis

  higher precision ⇒ smaller points-to sets ⇒ faster analysis
- More experiments are required to find this “right” level