# Arena Allocation via Bytecode Transformation

Adam Megacz

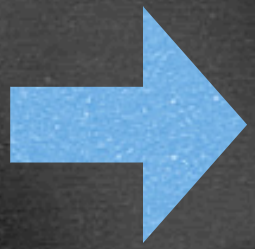
CS264 / Spring 2005

# Outline

- The Transformation
  - Various cases that need handling
- Analyses
  - Flow-insensitive
  - Flow-sensitive escape analysis
- Results
- Availability

# Simple Case

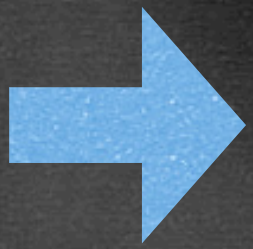
```
class Arena {
  class Foo implements Gladiator {
      char c;
      Foo f;
      String s;
  }
}
```



```
class Arena {
  int      Foo$$num;
  char[]   Foo$c;
  int[]    Foo$f;
  String[] Foo$s;
}
```

# Allocation

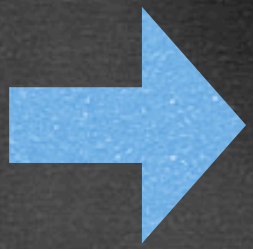
```
class Arena {
  class Foo {
      char c;
      Foo f;
      String s;
  }

  void foo() {
    Foo x = new Foo();


  }

}
```



```
class Arena {
  int      Foo$$num;
  char[]   Foo$c;
  int[]    Foo$f;
  String[] Foo$s;


  void foo() {
    int x = Foo$$num++;
    if (Foo$$num > Foo$c.length){
       // grow the arrays
    }
  }

}
```

# Boxing / Unboxing

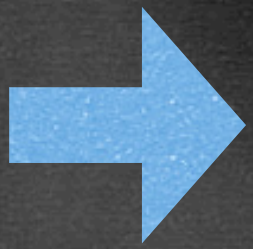
```
class Arena {
  class Foo {
      char c;
      Foo f;
      String s;
  }
  Vector v;
  void foo() {
    Foo x = new Foo();


    v.add(x);
    x = v.get(0);

  }

}
```



```
class Arena {
  int      Foo$$num;
  char[]   Foo$c;
  int[]    Foo$f;
  String[] Foo$s;

  Vector v;
  void foo() {
    int x = Foo$$num++;
    if (Foo$$num > Foo$c.length){
       // grow the arrays
    }
    v.add(new Integer(x));
    x = ((Integer)v.get(0))
            .intValue();
  }

}
```

# Dealing with null

```
class Arena {
  class Foo {
      char c;
      Foo f;
      String s;
  }
  Vector v;
  void foo() {
    Foo x = new Foo();



    v.add(x);
    x = v.get(0);

    if (x==null)
        throw RuntimeException();
  }
}
```



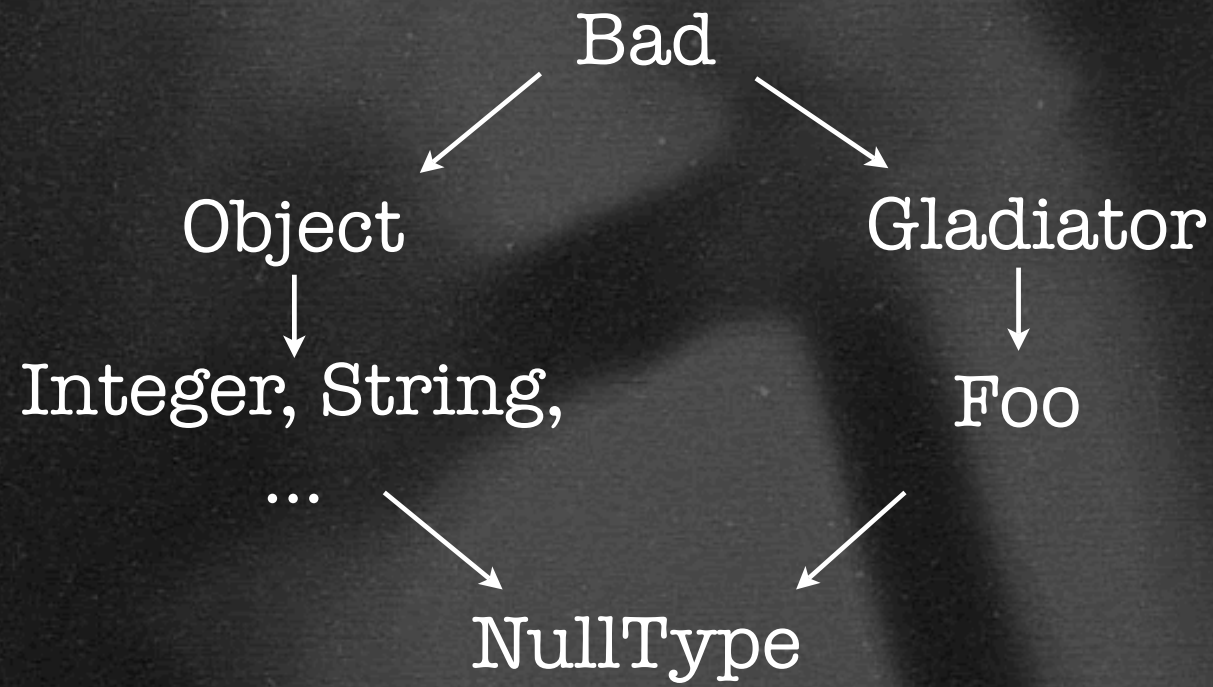
```
class Arena {
  int      Foo$$num;
  char[]   Foo$c;
  int[]    Foo$f;
  String[] Foo$s;

  Vector v;
  void foo() {
    int x = Foo$$num++;
    if (Foo$$num > Foo$c.length){
       // grow the arrays
    }
    v.add(new Integer(x));
    x = ((Integer)v.get(0))
            .intValue();
    if (x==0)
        throw RuntimeException();
  }
}
```

# Flow-insensitive Analysis

- Simple flow-insensitive analysis is used for
  - boxing/unboxing
  - devirtualizability checking
  - visibility of the Gladiator class
- Very similar to to Bogda&Holze (OOPSLA'99)

# Flow-insensitive Analysis

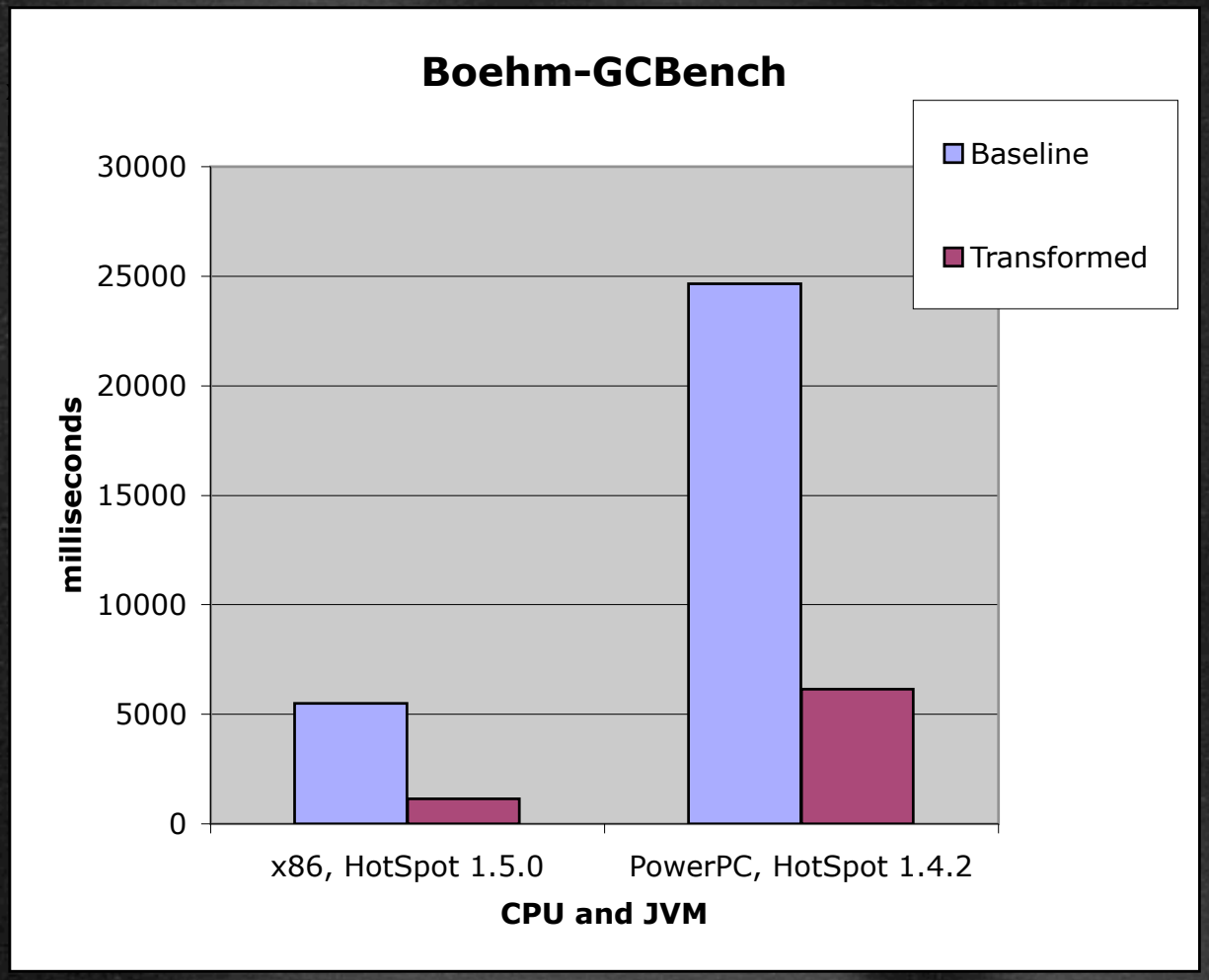
- Slightly more sophisticated flow-insensitive analysis for null/0 conversion. Simple unification over this lattice:



# Flow-Sensitive Analysis

- Ideal analysis: Escape Analysis for Java, Choi, Gupta, Serrano, Sreedhar, Midkiff (OOPSLA'09)
- I attempted a much simpler (and much, much less efficient) analysis
  - single bit for each SSA variable
    - 0 = cannot point to an [un]boxed Gladiator
    - 1 = might point to an [un]boxed Gladiator
  - Run Soot's convenient flow-sensitive analysis with simple bitwise OR as the join function.
  - It's sorta working. Somewhat. Okay, not really.

# Results: 5x on GCBench



# Availability

http://arenaj.sourceforge.net