Run-Time Support for Optimizations Based on Escape Analysis

CGO 2007
March 11-14, 2007

Thomas Kotzmann
kotzmann@ssw.jku.at

Hanspeter Mössenböck
moessenboeck@ssw.uni-linz.ac.at

Institute for System Software
Johannes Kepler University Linz, Austria
http://ssw.jku.at/

Sun Microsystems, Inc.
Santa Clara, USA
http://www.sun.com/

Escape Analysis for the Java HotSpot™ VM

- Detection of method-local/thread-local objects
  - Scalar replacement of fields
  - Stack allocation
  - Synchronization removal
- Interprocedural analysis
  - Inlining decisions
  - Stack allocation of parameters
  - Removal of synchronization on return value

Example

```
static void draw(Shape shape) {
  Color color = new Color(0x6699ff);
  shape.stroke = new BasicStroke();
  Figure figure = new Figure(shape, color);
  figure.draw();
}
```

```
final synchronized void draw() {
  Canvas canvas = getCanvas();
  canvas.render(this);
}
```

Example

```
static void draw(Shape shape) {
  Color color = new Color();
  color.rgb = 0x6699ff;
  shape.stroke = new BasicStroke();
  Figure figure = new Figure();
  figure.shape = shape;
  figure.rgb = color.rgb;
  synchronized (figure) {
    Canvas canvas = getCanvas();
    canvas.render(figure);
  }
}
```

Example

```
static void draw(Shape shape) {

  int rgb = 0x6699ff;
  shape.stroke = new BasicStroke();
  Figure figure = new Figure();  // on the stack
  figure.shape = shape;
  figure.rgb = rgb;

    Canvas canvas = getCanvas();
    canvas.render(figure);

}
```

Run-Time Support

- Card marking
  - Extended write barrier
- Garbage collection
  - Pointers in stack objects
- Deoptimization
  - Reallocation and relocking
  - Debugging information

Write Barriers

```
  static void draw(Shape shape) {
    int rgb = 0x6699ff;
➤   shape.stroke = new BasicStroke();
    Figure figure = new Figure();  // on the stack
    figure.shape = shape;
    figure.rgb = rgb;
    Canvas canvas = getCanvas();
    canvas.render(figure);
  }
```

```
                    shr  eax, 9
                    sub  eax, firstIndex
                    cmp  eax, arraySize
                    jae  label
                    mov  byte ptr [eax+arrayBase], 0
              label:  ...
```

Wrapper for Oop Closures

```
  static void draw(Shape shape) {
    int rgb = 0x6699ff;
    shape.stroke = new BasicStroke();
    Figure figure = new Figure();  // on the stack
➤   figure.shape = shape;
    figure.rgb = rgb;
    Canvas canvas = getCanvas();
    canvas.render(figure);
  }
```

```
void do_oop(oop obj) {
  if (is_in_heap(obj)) {
    wrapped_closure.do_oop(obj);
  } else if (!obj.has_been_scanned()) {
    obj.set_has_been_scanned();
    obj.iterate_oop_fields(this);
  }
}
```

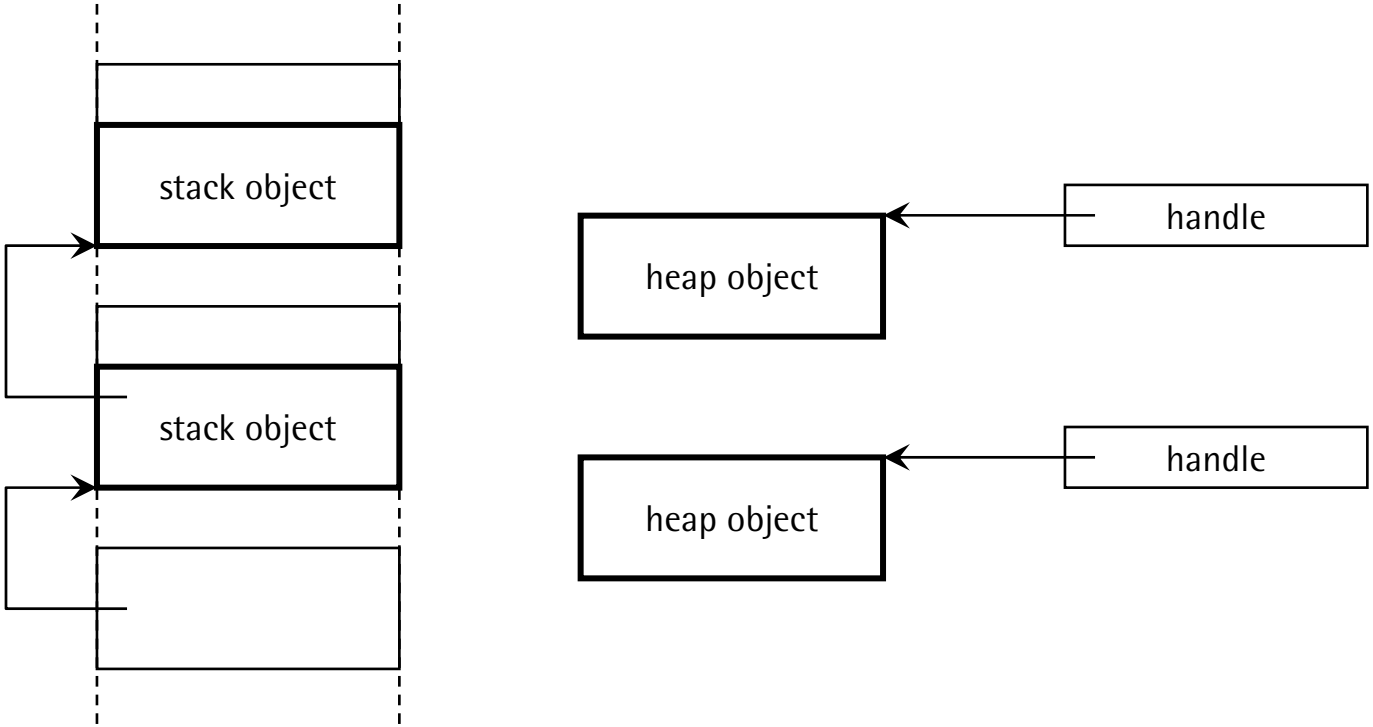
Dynamic Class Loading

```
  static void draw(Shape shape) {
    int rgb = 0x6699ff;
    shape.stroke = new BasicStroke();
    Figure figure = new Figure();  // on the stack
    figure.shape = shape;
    figure.rgb = rgb;
➤   Canvas canvas = getCanvas();
    canvas.render(figure);
  }
```

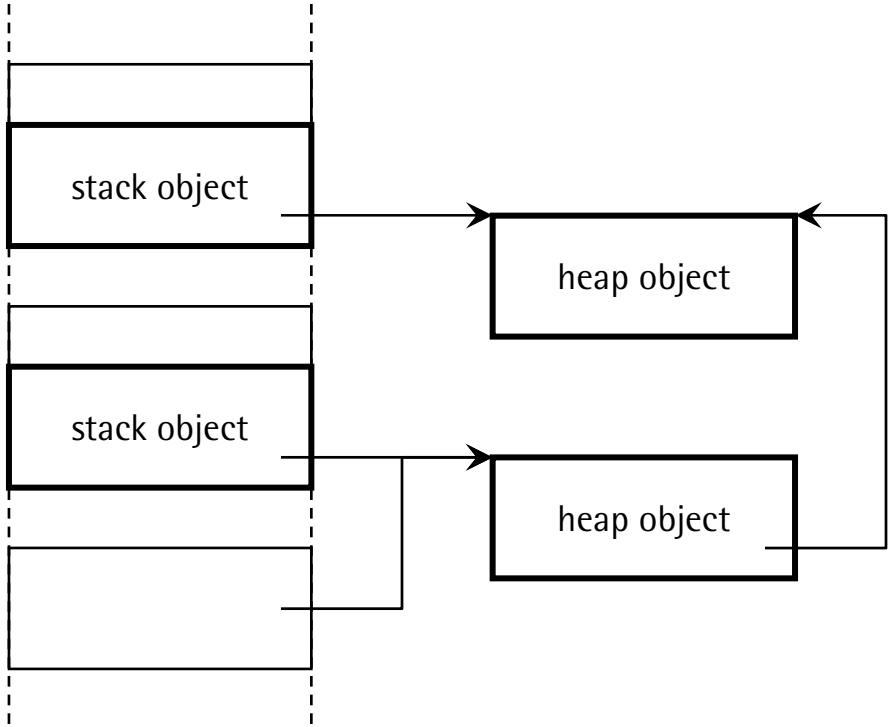
Deoptimization

- Upon class loading
  - Identify dependent methods
  - Patch machine code for lazy deoptimization
  - Reallocate and relock stack objects
- Lazy deoptimization
  - Reallocate and relock scalar-replaced objects
  - Set up interpreter frame
- Continue execution in interpreter

Reallocation of Stack Objects



Reallocation of Stack Objects



Information for GC and Deoptimization

- Oop maps
  - Location of root pointers
  - Registration of stack objects
- Method dependencies
  - Use of interprocedural escape information
- Debugging information
  - Local variables and operand stack
  - Type and field values of scalar-replaced objects
  - Position of stack objects
  - Objects for which synchronization was removed

Conclusions

- Abstraction from stack objects
  - Extended write barrier
  - Wrapper for oop closures
- Debugging information
  - Representation of optimized objects
- Deoptimization
  - Reallocation and relocking
  - Lazy reallocation of scalar-replaced objects
- Implemented in production system