CSE115 / CSE503
Introduction to Computer Science I

Dr. Carl Alphonce

343 Davis Hall

alphonce@buffalo.edu

Office hours:

Tuesday 10:00 AM – 12:00 PM*

Wednesday 4:00 PM – 5:00 PM

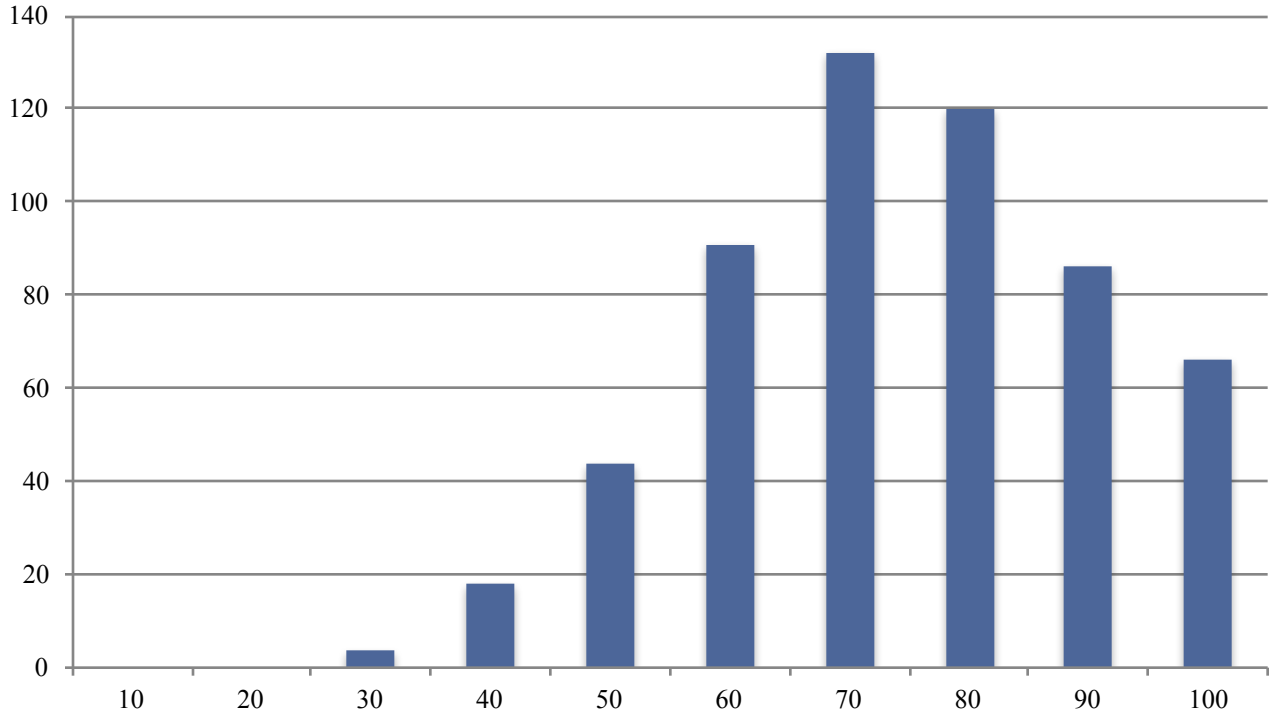
Friday 11:00 AM – 12:00 PM

OR request appointment via e-mail

*__Tuesday adjustments: 11:00 AM – 1:00 PM on 10/11__, 11/1 and 12/6

ANNOUNCEMENTS

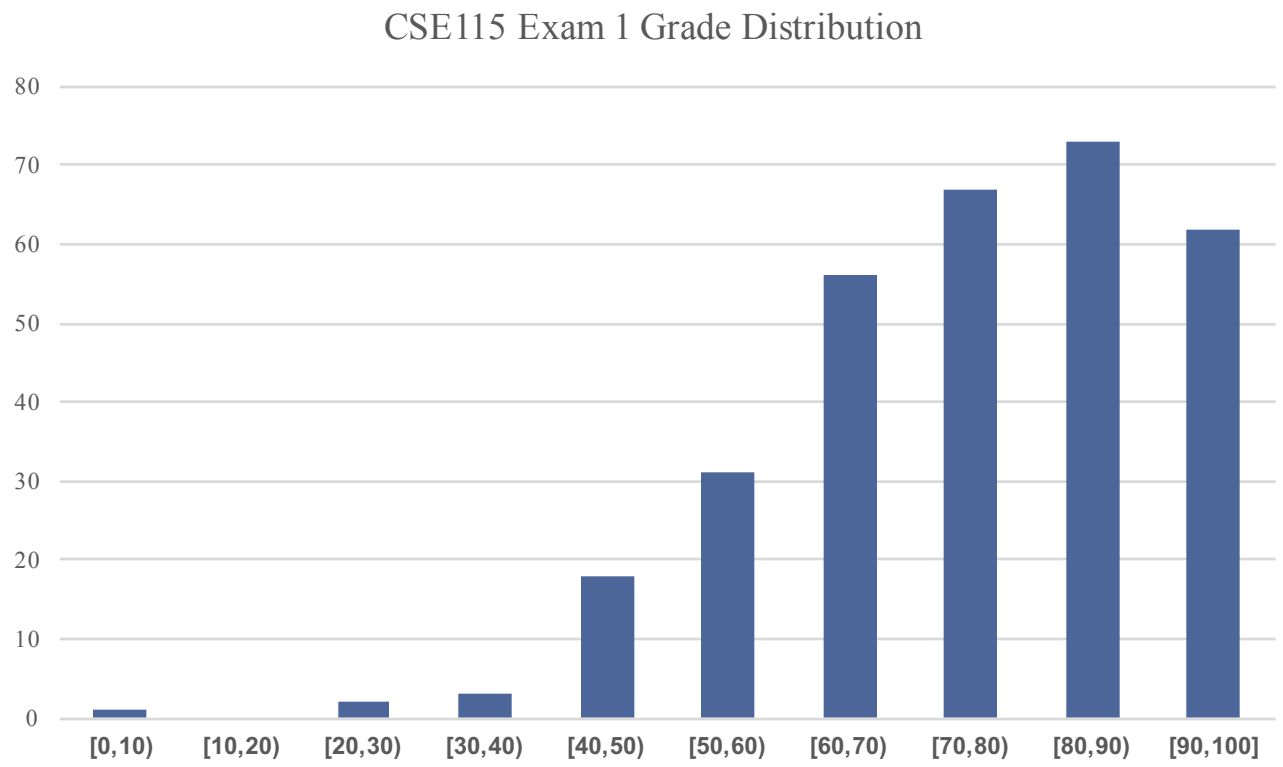
EXAM RESULTS (FA15)



MAX 100 (x19 = 3.4%)
MEDIAN 68
MEAN 68.4
MIN 20

N = 577

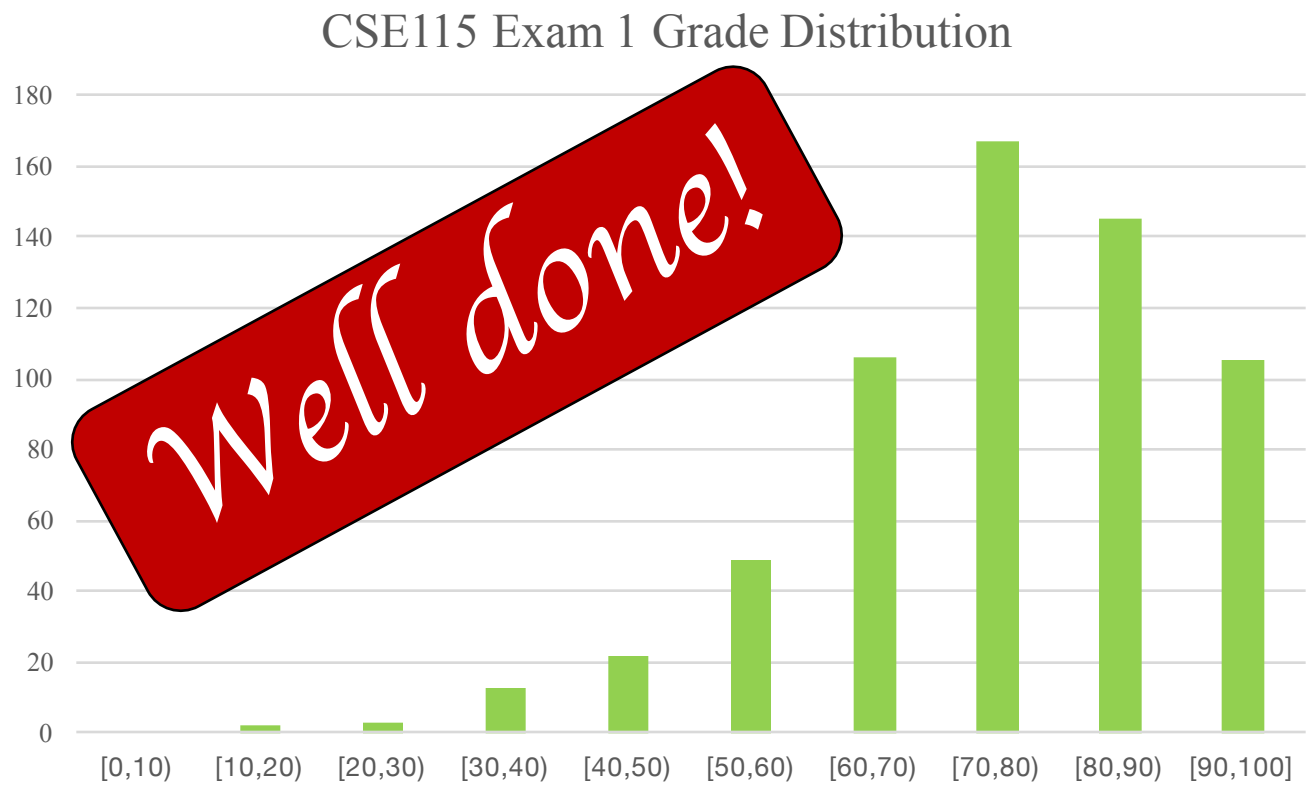
EXAM RESULTS (SP16)



MAX 100 (x13 = 4.2%)
MEDIAN 76
MEAN 74.3
MIN 8

N = 320

EXAM RESULTS (FA16)



MAX 100 (x15 = 2.5%)
MEDIAN 76
MEAN 74.4
MIN 16

N = 626

ELECTRONICS:

off & away

ROADMAP

Last time

- Modeling: sharing association

Today

- Modeling: exclusive association
- null

Coming up

- Interfaces
- Realization relationship

REVIEW

The class definition

```
public class Shape {
    private java.awt.Color _color;
    public Shape(java.awt.Color c) {
        _color = c;
    }
    public java.awt.Color getColor() {
        return _color;
    }
    public void setColor(java.awt.Color c){
        _color = c;
    }
}
```

```
Shape s1 = new Shape(java.awt.Color.BLUE);
Shape s2 = new Shape(java.awt.Color.RED);
s2.setColor(s1.getColor());
```

What are the highlighted expressions?

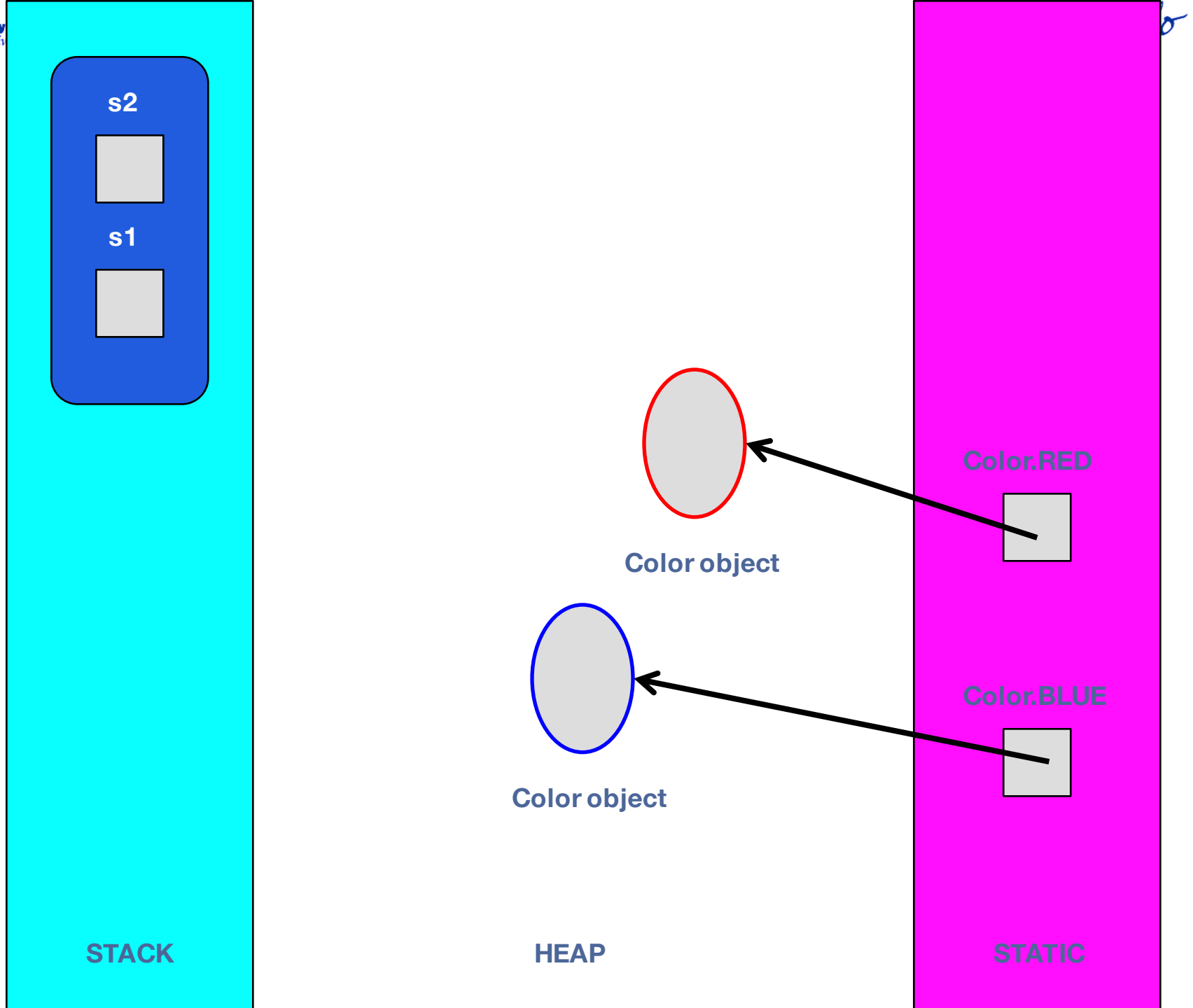
Exercise

```
Shape s1 = new Shape(java.awt.Color.BLUE);
Shape s2 = new Shape(java.awt.Color.RED);
s2.setColor(s1.getColor());
```

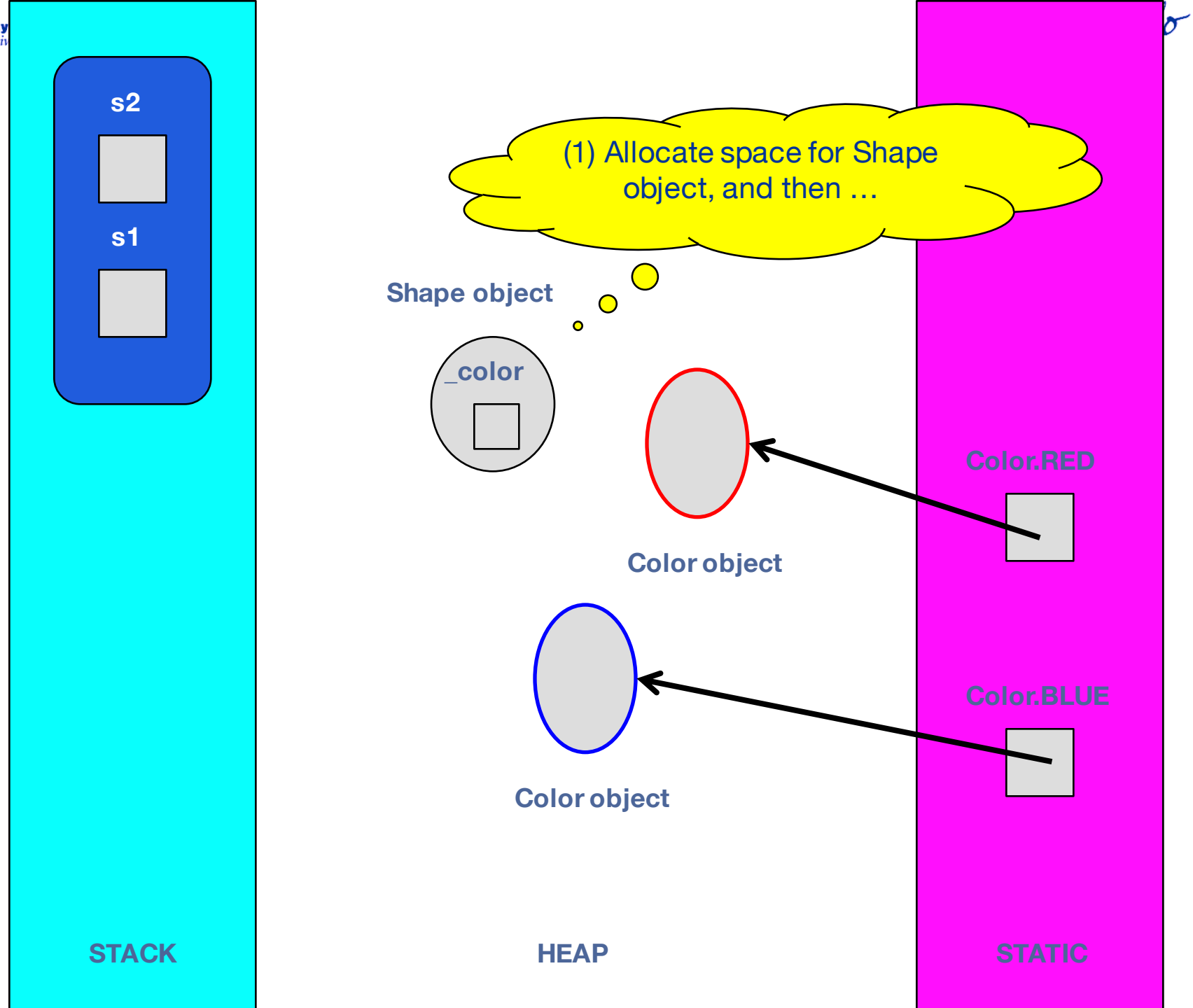
Discuss with your neighbors what happens.

What will the object diagram look like after the method calls happen?

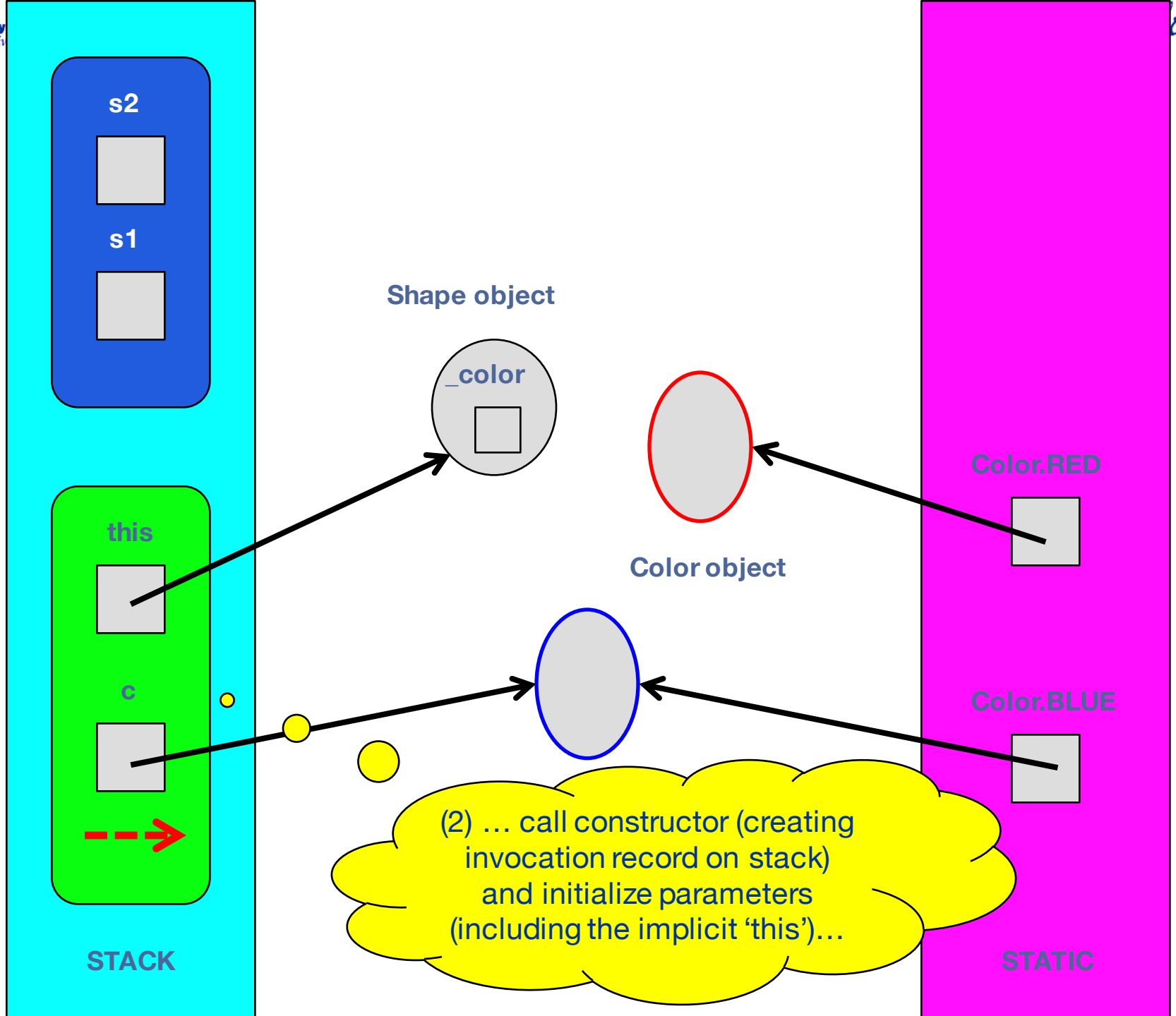
```
s1 = new Shape(java.awt.Color.BLUE)
```



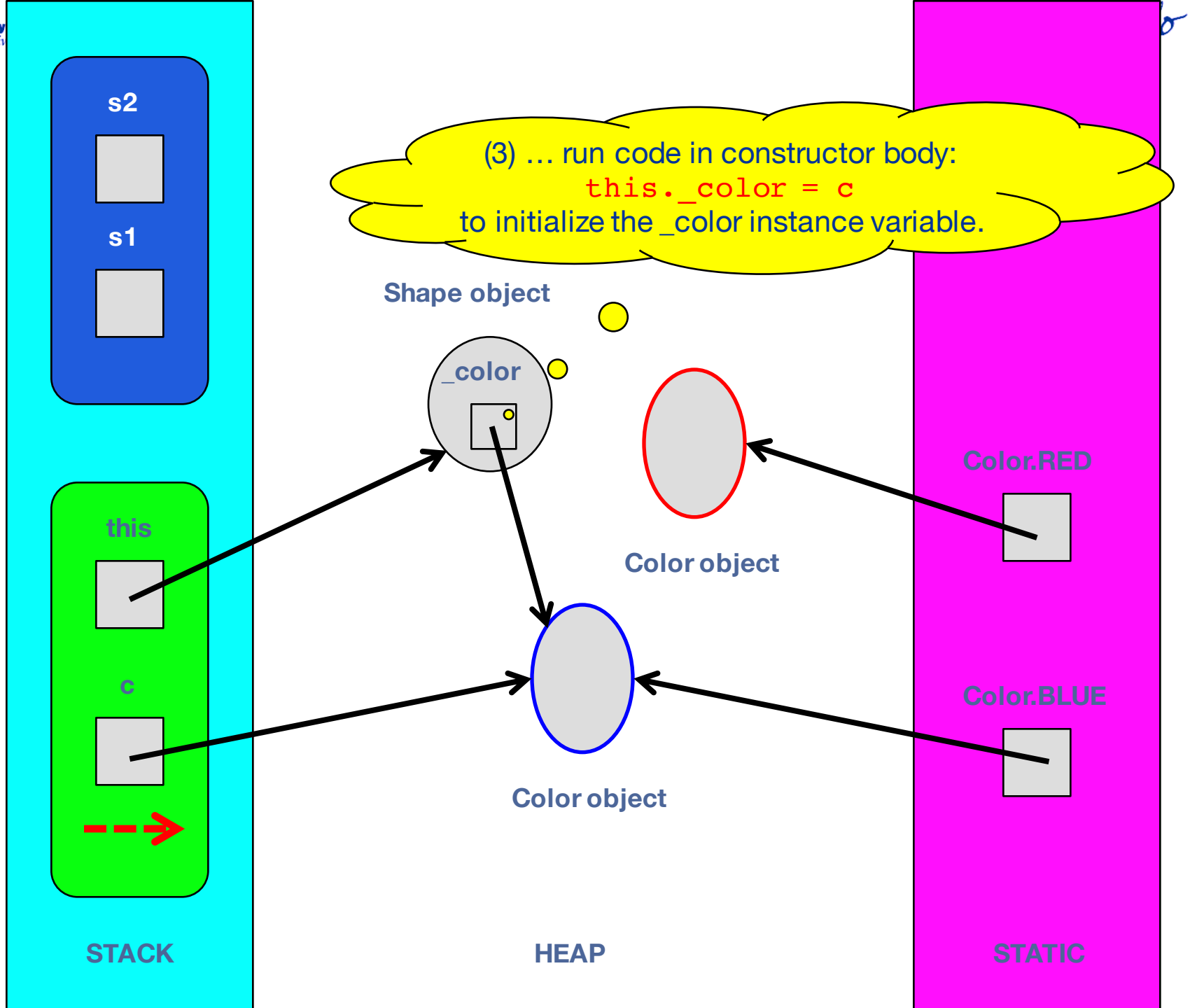
s1 = new Shape(java.awt.Color.BLUE)



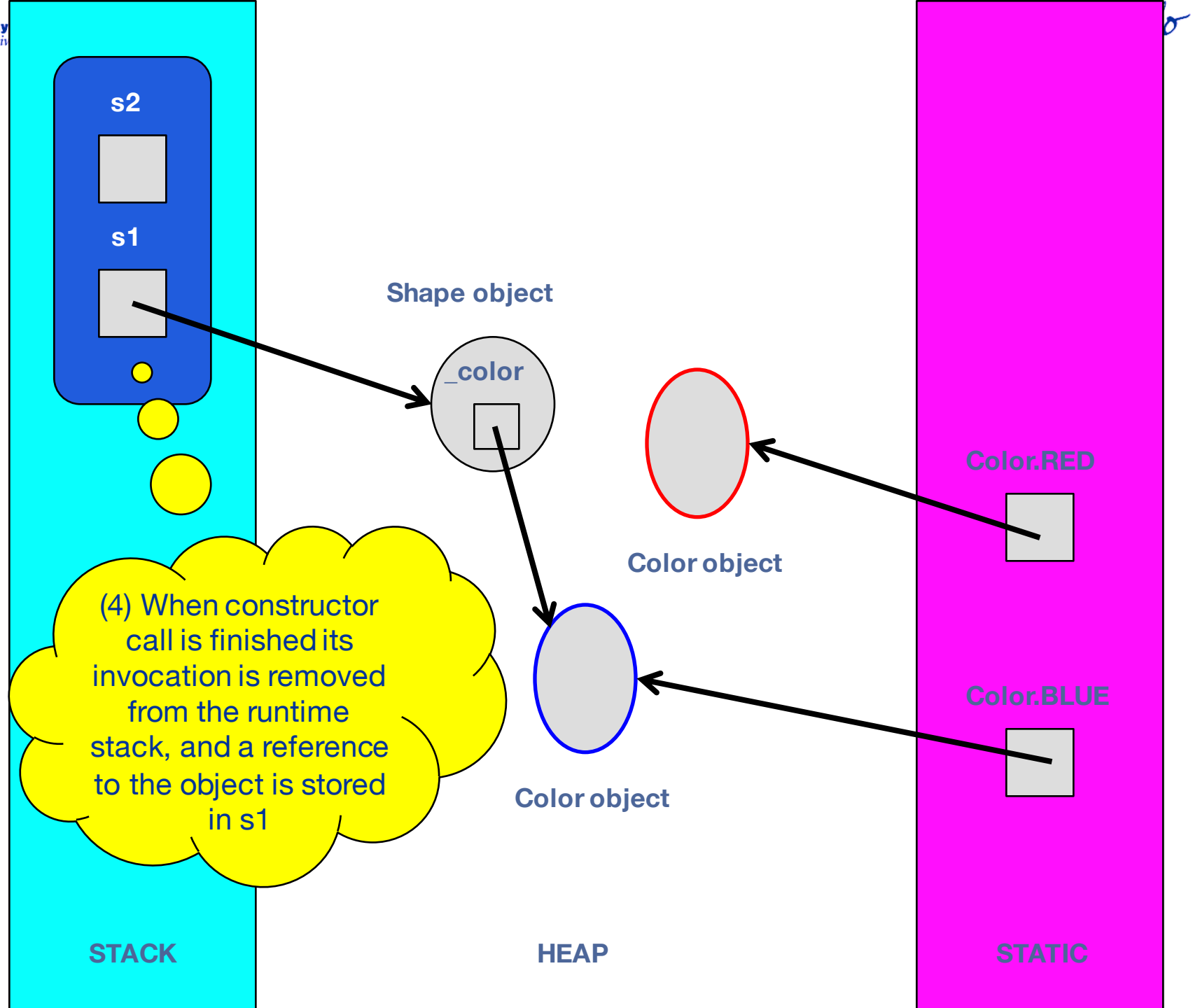
s1 = new Shape(java.awt.Color.BLUE)



s1 = new Shape(java.awt.Color.BLUE)



s1 = new Shape(java.awt.Color.BLUE)

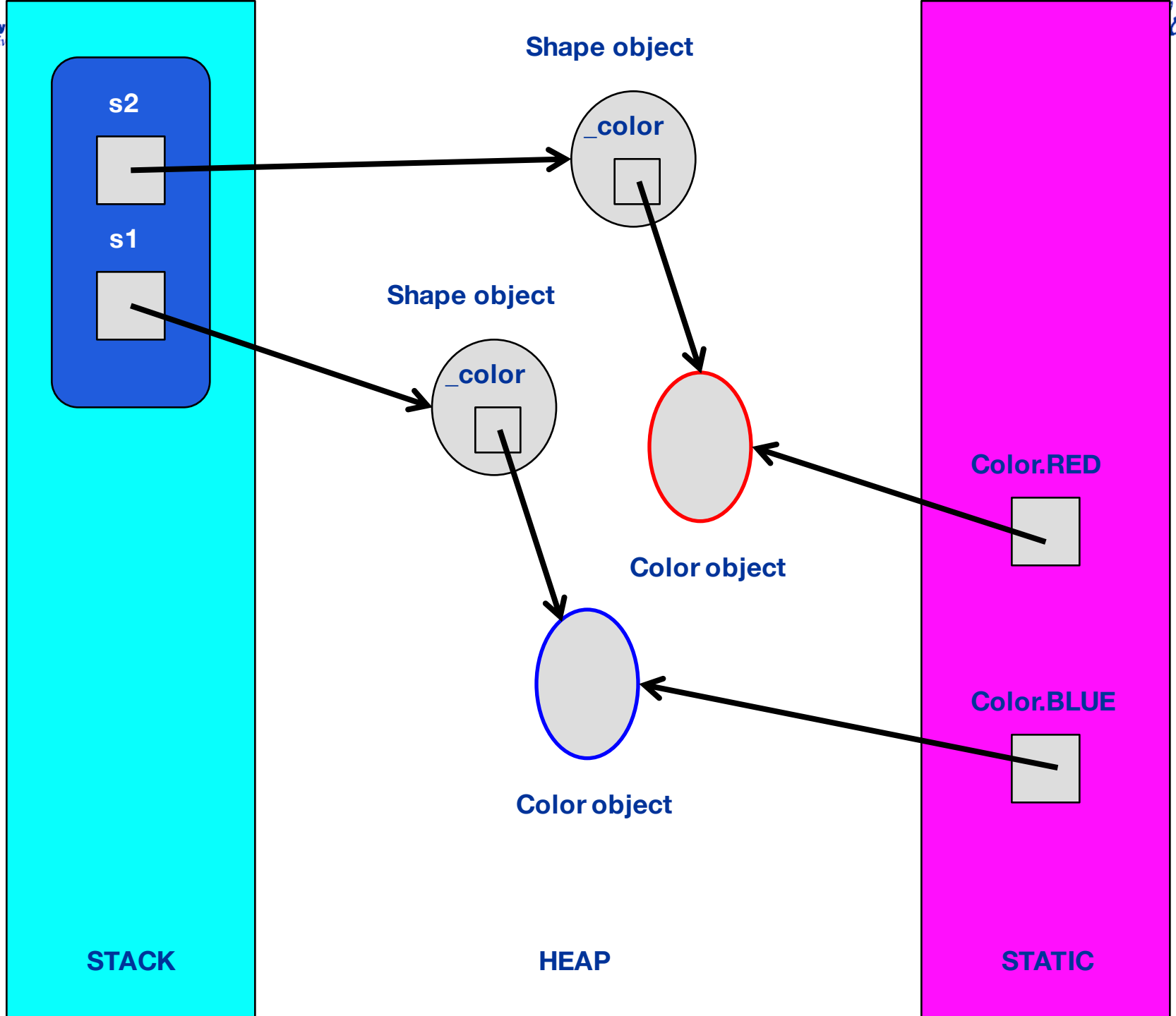


BEFORE

The next slide shows the "before" object diagram that you and your neighbors should have come up with.

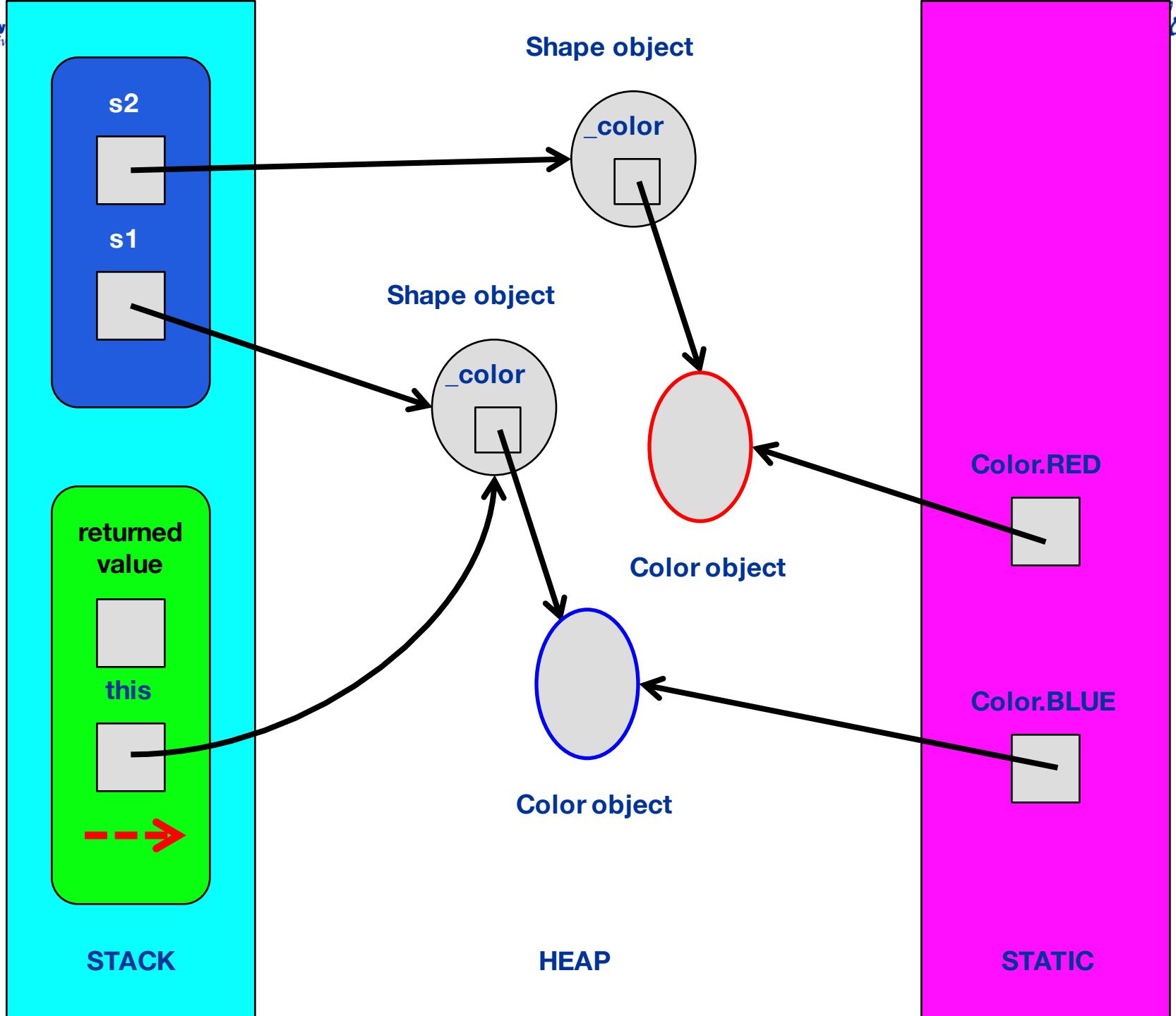
Before call:

`s2.setColor(s1.getColor())`



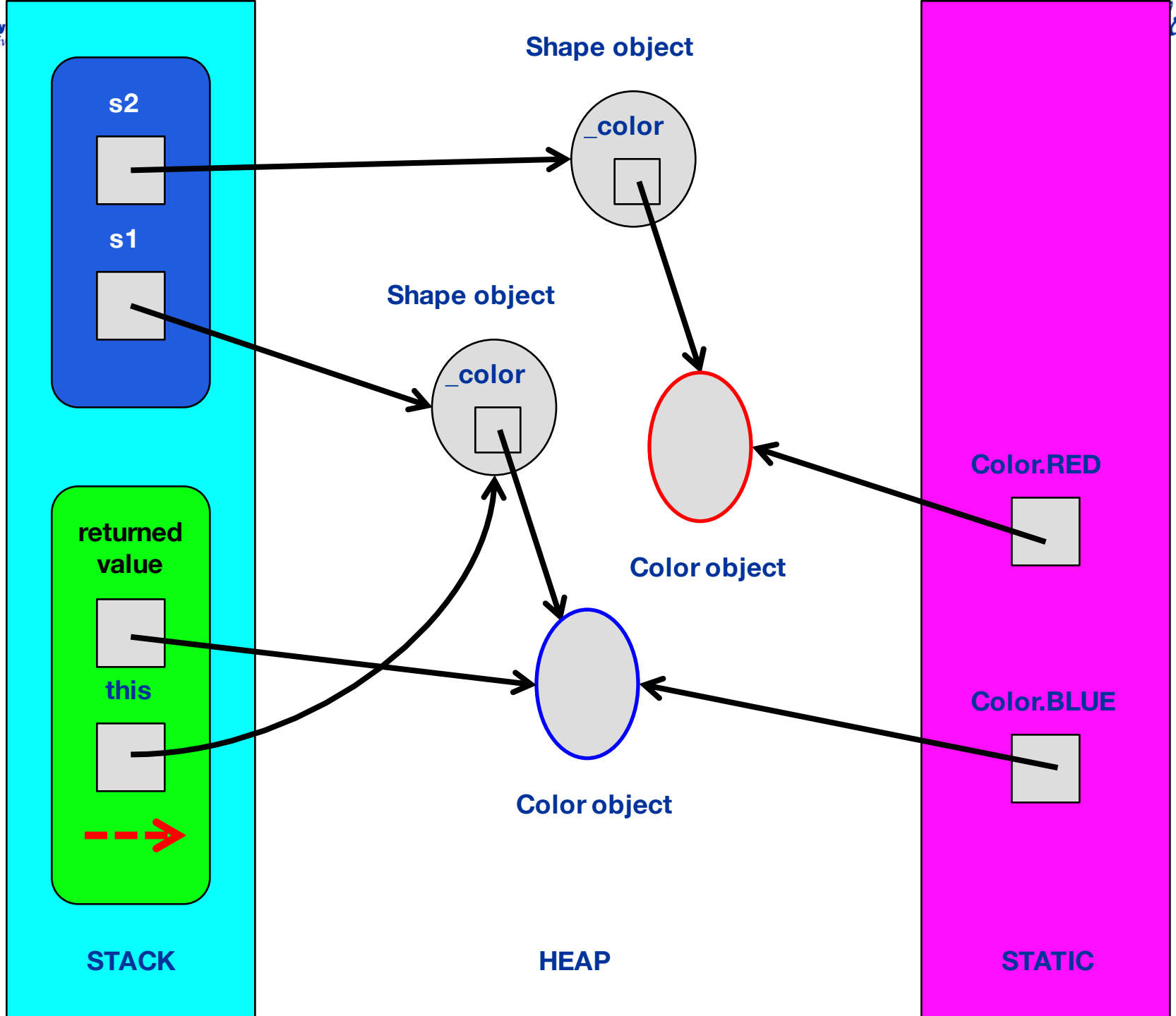
At call entry:

`s2.setColor(s1.getColor())`

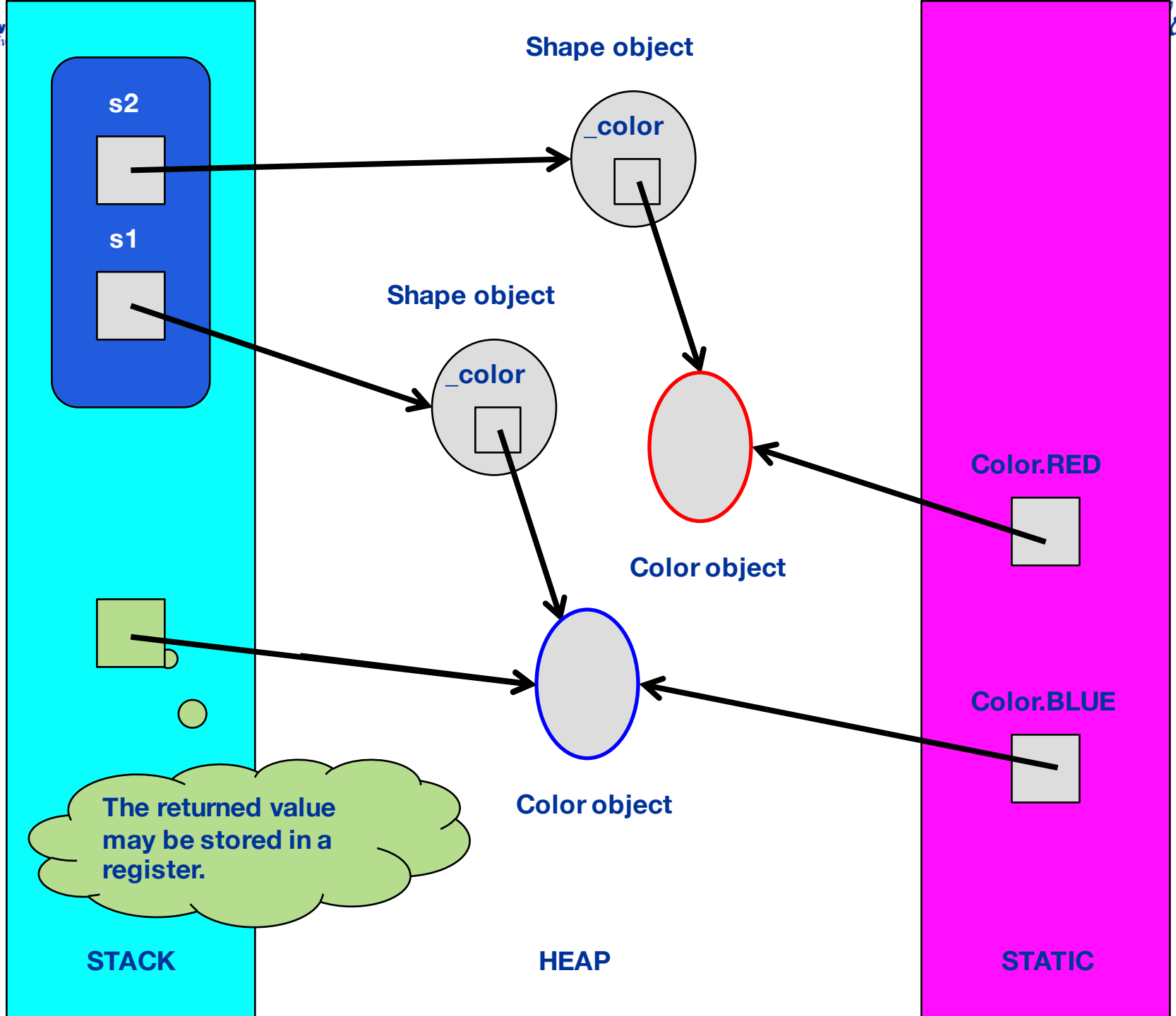


At call exit:

`s2.setColor(s1.getColor())`

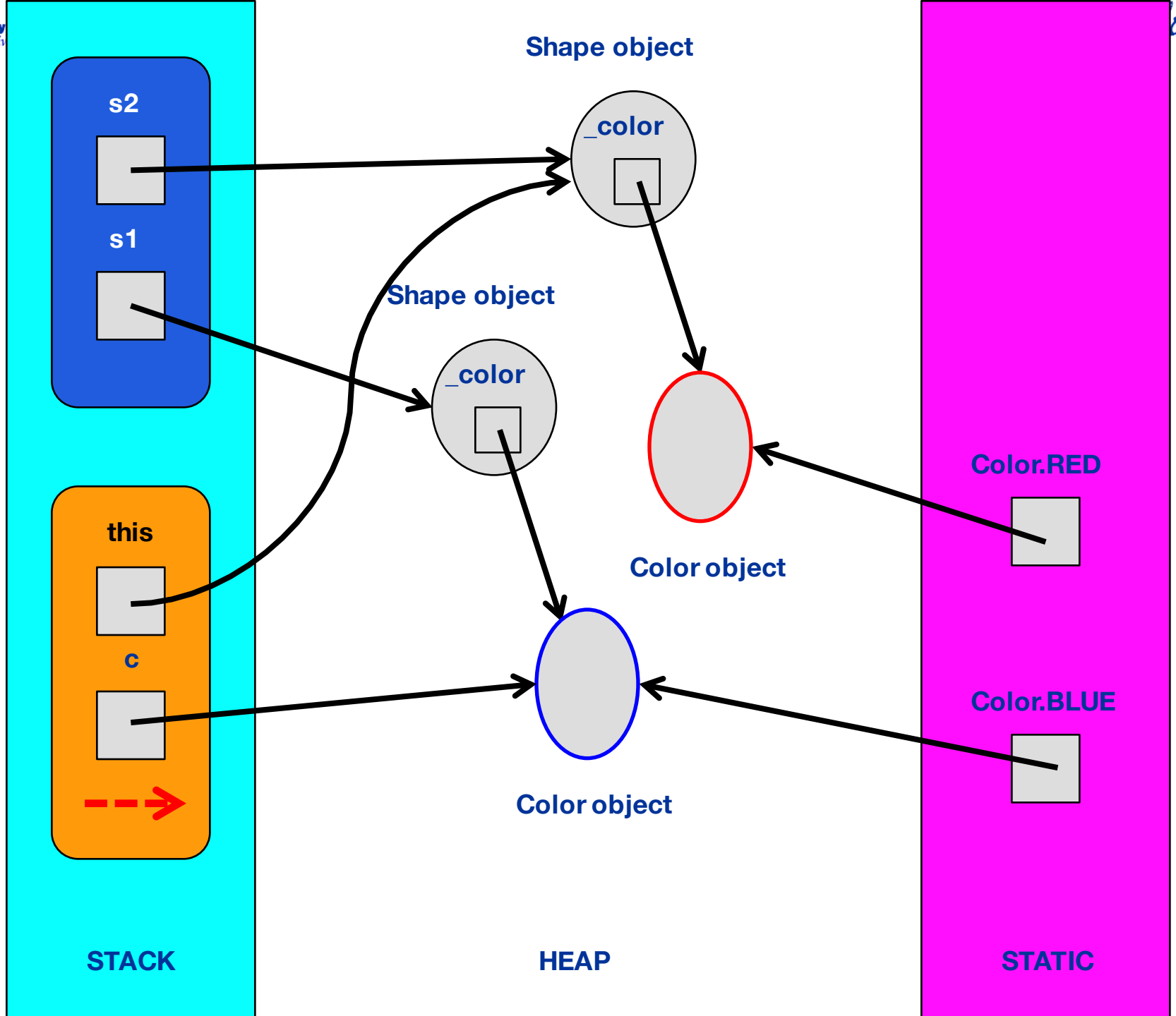


The returned value is held in a temporary: `s2.setColor(s1.getColor())`



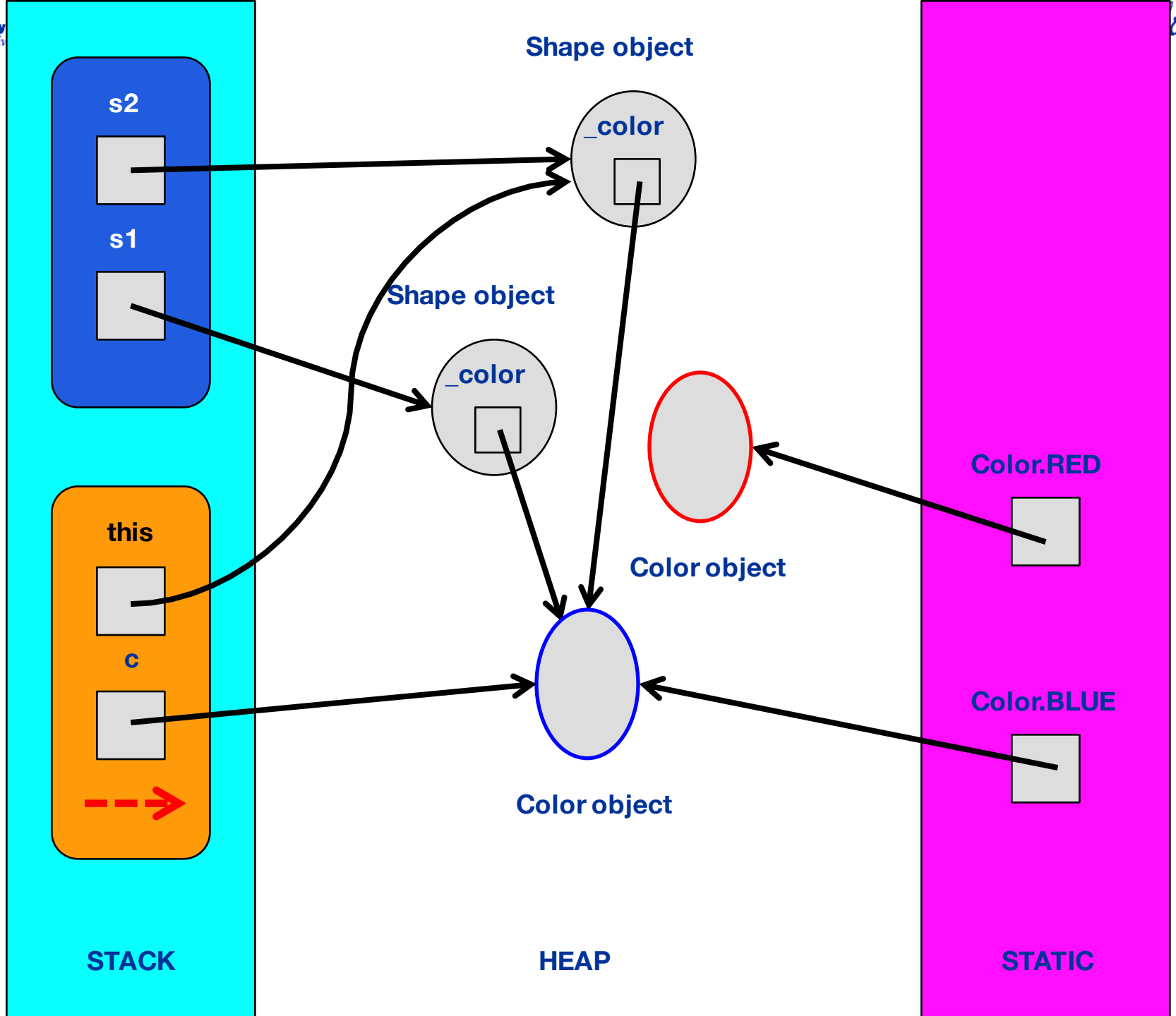
At call entry:

`s2.setColor(s1.getColor())`



At call exit:

`s2.setColor(s1.getColor())`

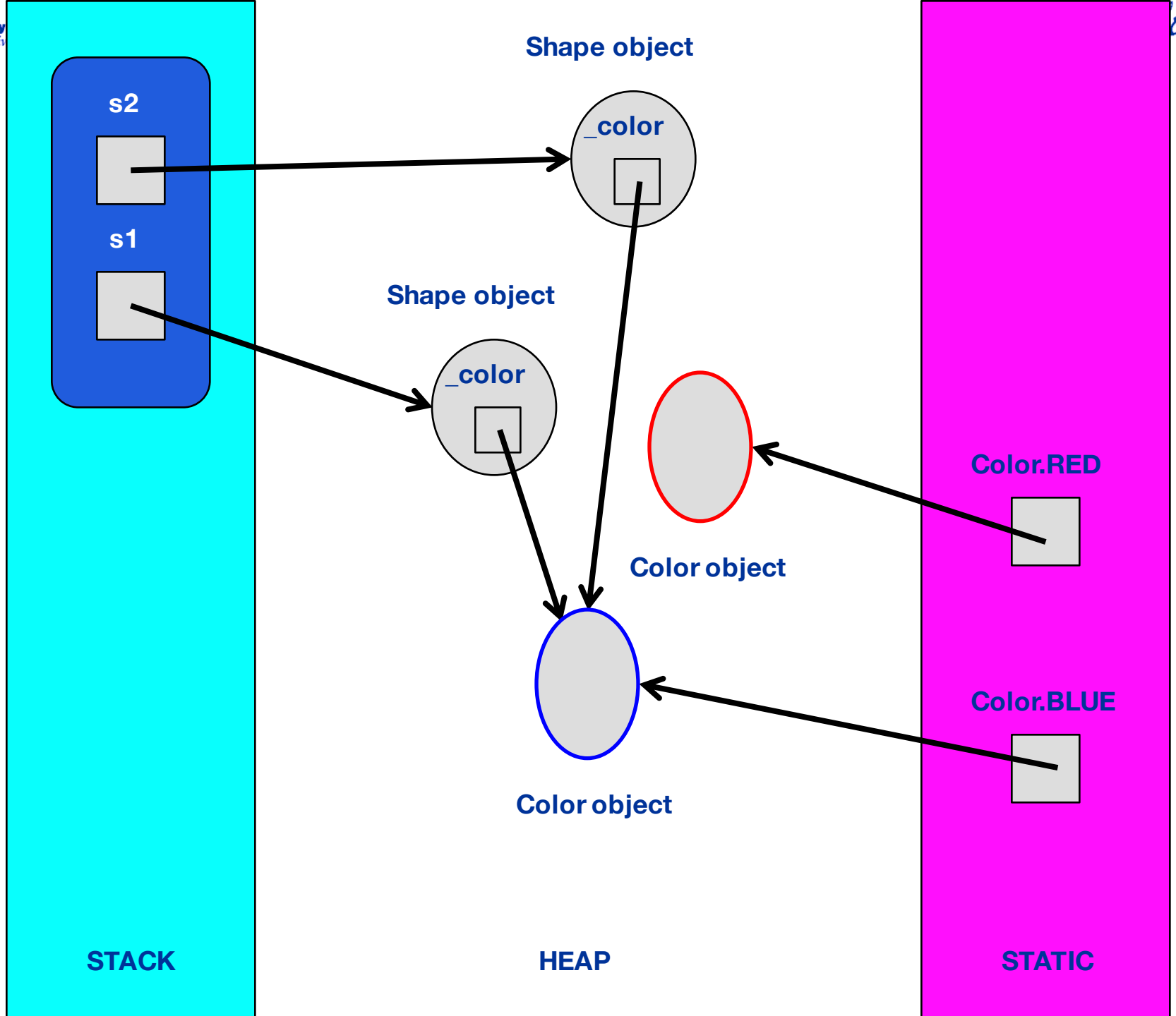


AFTER

The next slide shows the "after" object diagram that you and your neighbors should have come up with.

After call exit:

`s2.setColor(s1.getColor())`



Result?

Both shapes have the same color (java.awt.Color.BLUE).

This is OK.

MOVING ON

MODELING

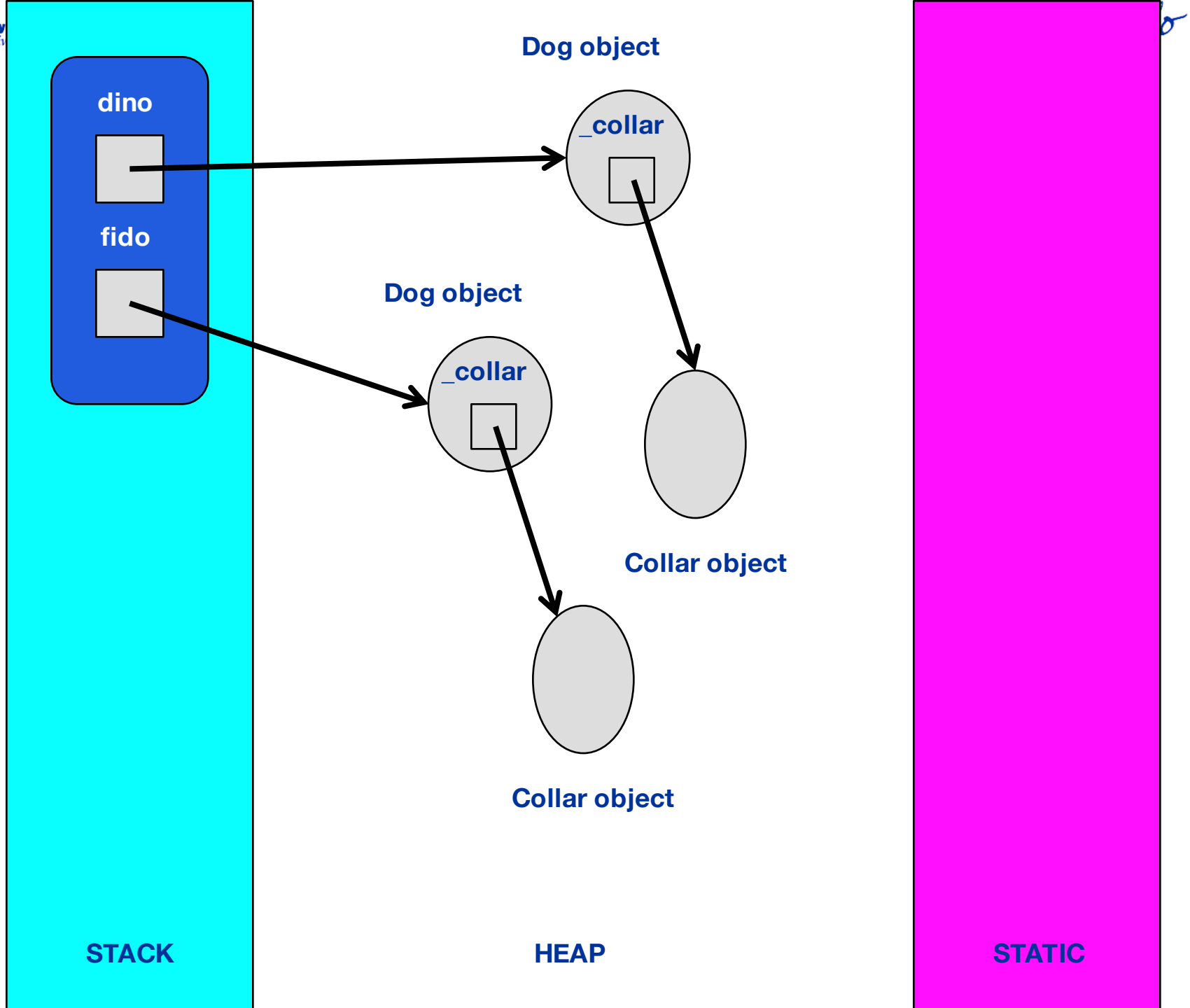
(also: execution model)

EXERCISE

Now consider this scenario:

```
Dog fido = new Dog(new Collar());
Dog dino = new Dog(new Collar());
```

```
Dog dino = new Dog(new Collar());
Dog fido = new Dog(new Collar());
```



Exercise

```
Dog fido = new Dog(new Collar());
Dog dino = new Dog(new Collar());
dino.setCollar(fido.getCollar());
```

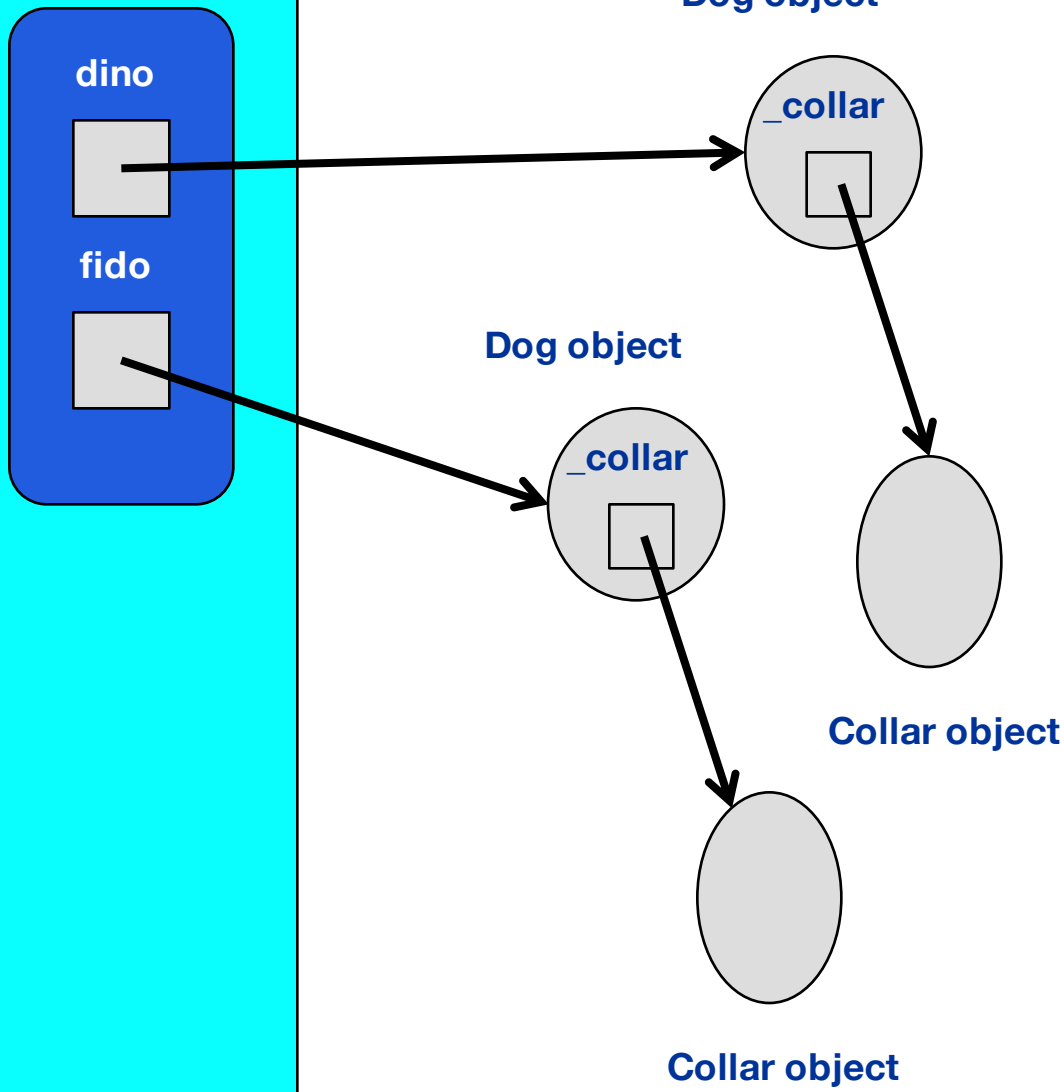
*This example has *exactly* the same structure as our Shape-Color example.*

Consider what happens in the object diagram. Run through it with your neighbors. Make sure you understand it!

Does it make sense?

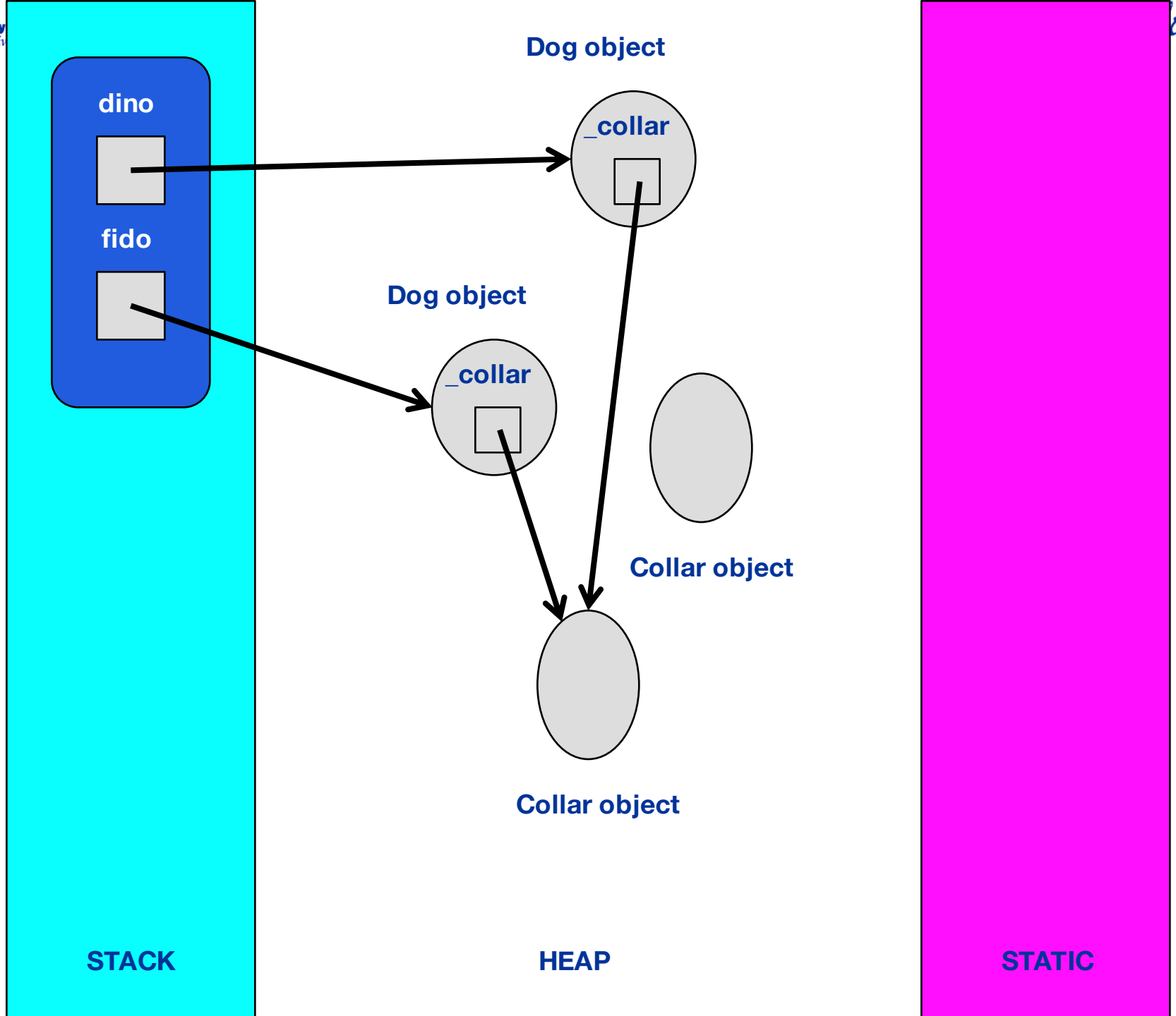
Before call:

```
dino.setCollar(fido.getCollar())
```



After call:

`dino.setCollar(fido.getCollar())`



Result?

Both dogs have the same collar.

Two objects can sensibly share the color blue,

but two dogs cannot sensibly share a single collar.

Aside: the second collar is “lost”, since we no longer have a reference to it. When we no longer maintain any reference to an object it becomes eligible for garbage collection.

null

What could we do instead?

Try to express what the basic problem is, and what we could do instead, in plain English.

'null' denotes the null reference, a reference which does not refer to any object.

There is also a special *null type*, the type of the expression null which has no name. Because the null type has no name, it is impossible to declare a variable of the null type or to cast to the null type. The null reference is the only possible value of an expression of null type. The null reference can always undergo a widening reference conversion to any reference type.

In practice, the programmer can ignore the null type and just pretend that null is merely a special literal that can be of any reference type.

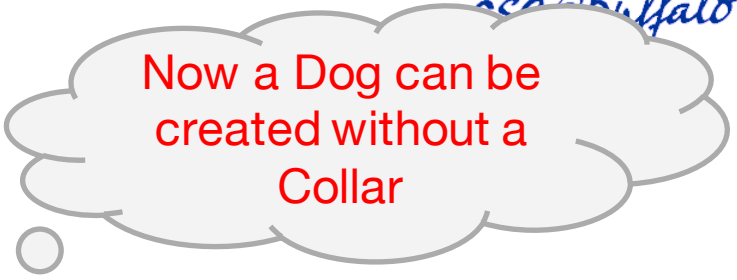
https://docs.oracle.com/javase/specs/jls/se7/html/jls-4.html

We can use 'null' to solve the two dogs, one collar problem (see code on next slide).

removeCollar rather than getCollar

```
public class Dog {
    private Collar _collar;
    public Dog(Collar collar) {
        this._collar = collar;
    }
    public void setCollar(Collar collar) {
                    this._collar = collar;
    }
    public Collar removeCollar() {
            Collar temp = this._collar;
            this._collar = null;
        return temp;
    }
}
```

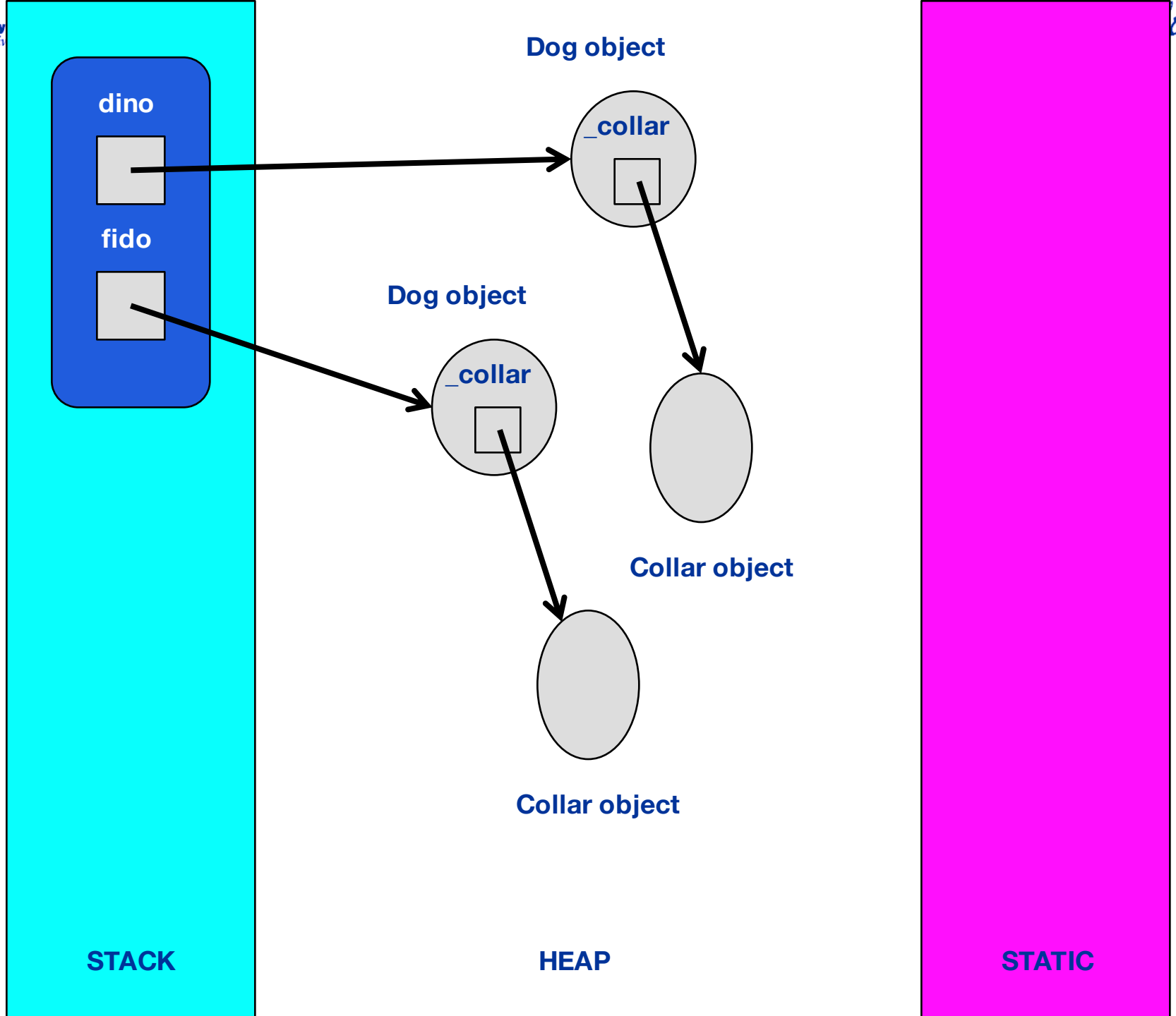
Can also use in constructor




```
public class Dog {
    private Collar _collar;
    public Dog() {
            _collar = null;
    }
    public void setCollar(Collar collar) {
                _collar = collar;
    }
    public Collar removeCollar() {
            Collar temp = _collar;
            _collar = null;
        return temp;
    }
}
```

Before call:

`dino.setCollar(fido.removeCollar())`



After call:

`dino.setCollar(fido.removeCollar())`