COMP 333 Lecture 3

Kyle Dewey

Outline

- Dynamic typing
- Memory management
    - Reference counting
    - Garbage collection

Dynamic typing

Basic idea: types are associated with values

Dynamic typing

Basic idea: types are associated with values

JavaScript

```
var x = 7;
x = "hello";
```

Dynamic typing

Basic idea: types are associated with values

JavaScript

```
var x = 7;
x = "hello";
```

Java

```
int x = 7;
// compile-time error
x = "hello";
```

Advantages

- More programs are possible
- Potentially very different values can be used in the same context
- De-emphasizes what correct values are

Disadvantages

- More programs are possible
- Potentially very different values can be used in the same context
- De-emphasizes what correct values are

Memory Management

Reference Counting

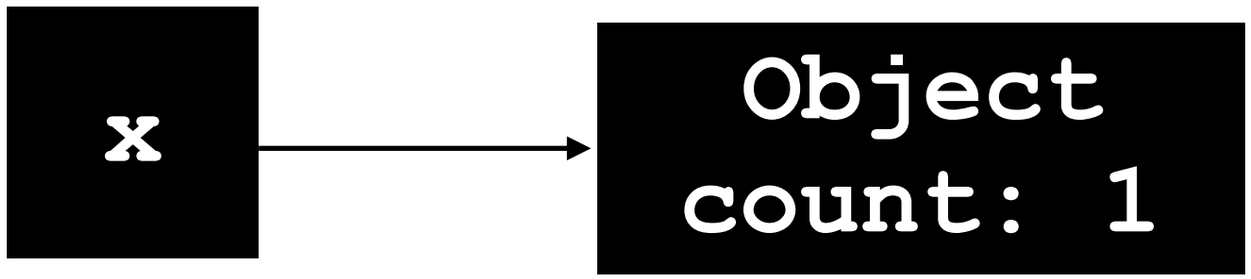
- Basic idea: objects maintain a count of how many things point to them
- Every type a new pointer to the object is added, the count is increased
- Each time a pointer is redirected elsewhere, the count is decreased
- When the count reaches 0, it deletes itself, possibly decreasing counts elsewhere

Example

```
var x = new Object();
```

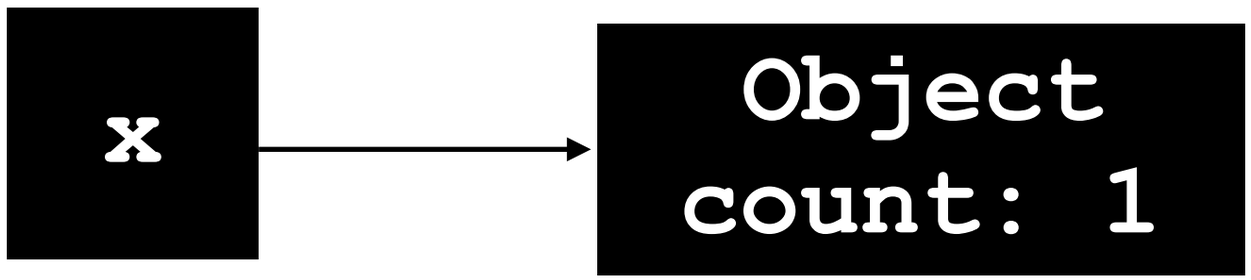
Example

```
var x = new Object();
```



Example

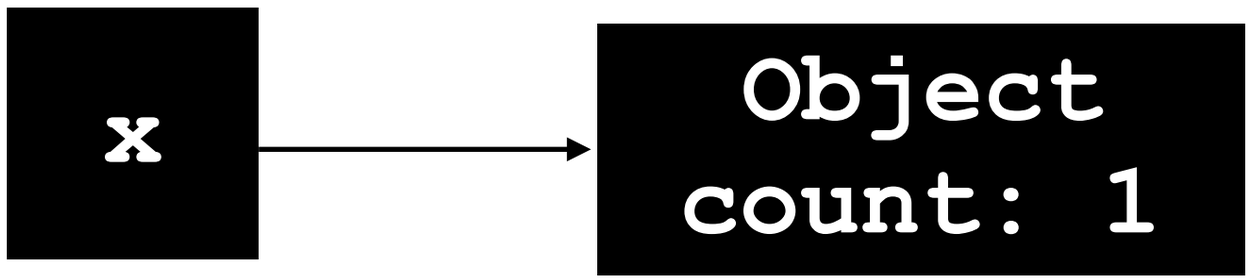
```
var x = new Object();
```



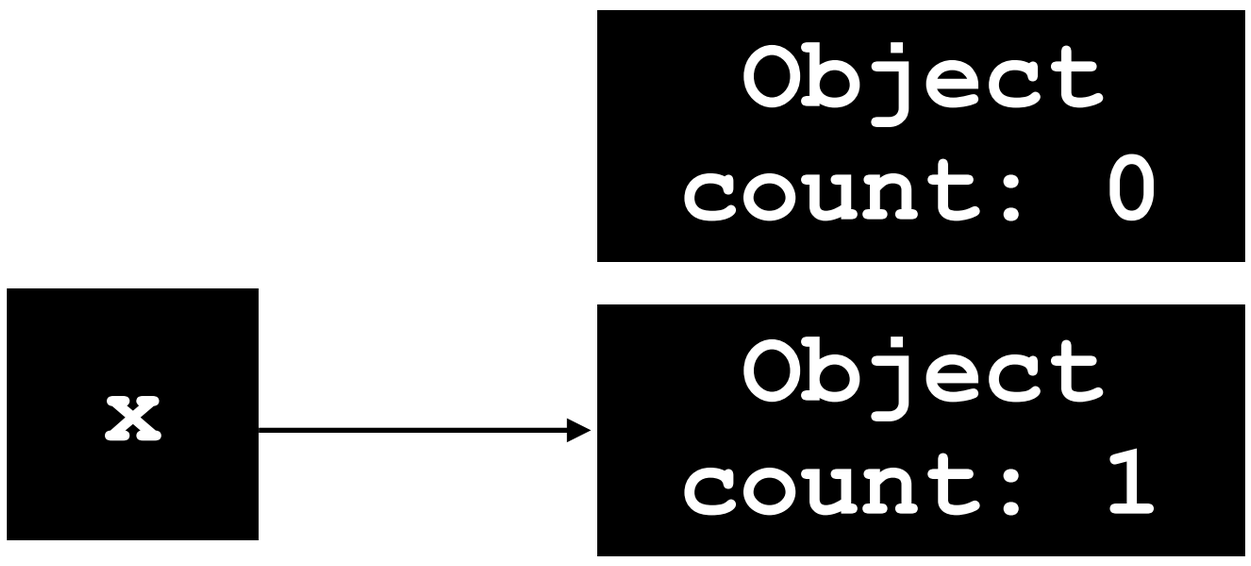

```
x = new Object();
```

Example

```
var x = new Object();
```

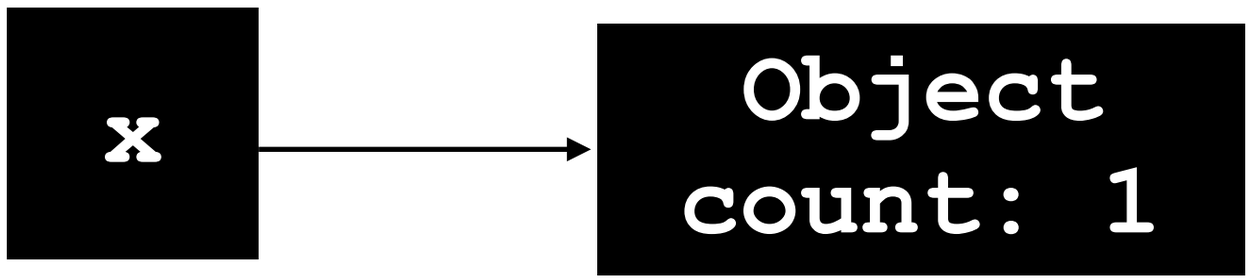



```
x = new Object();
```

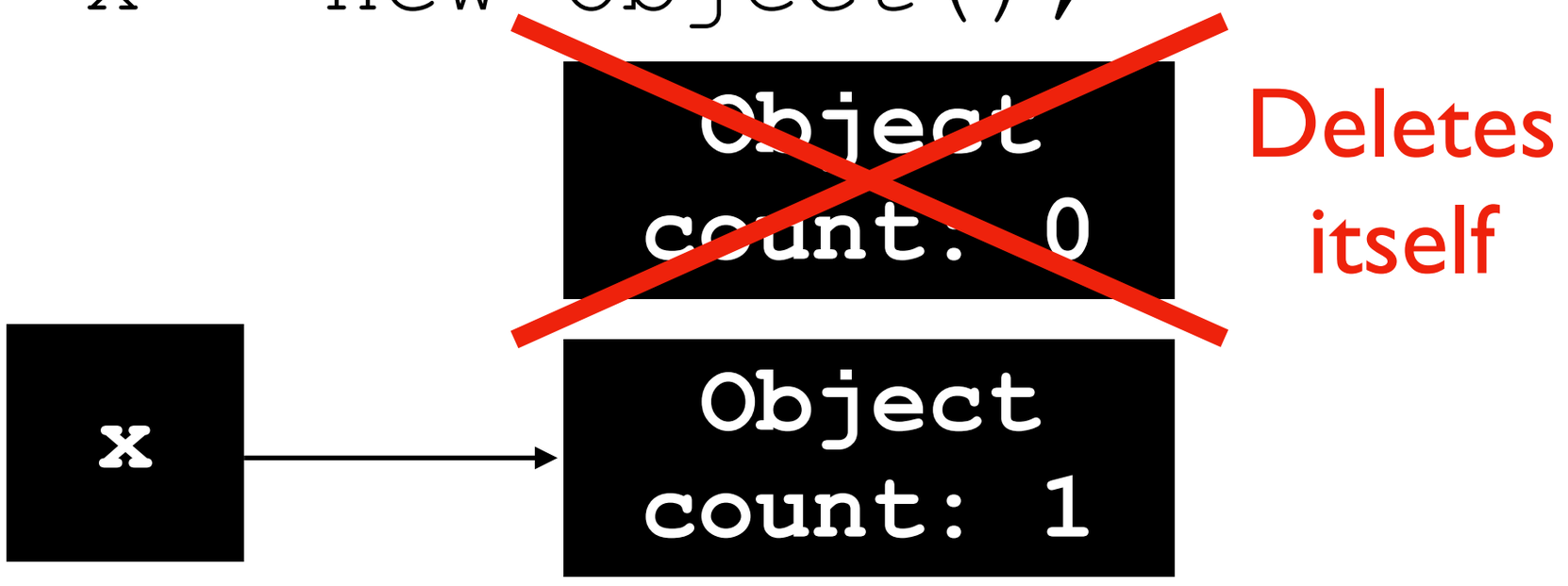


Example

```
var x = new Object();
```




```
x = new Object();
```

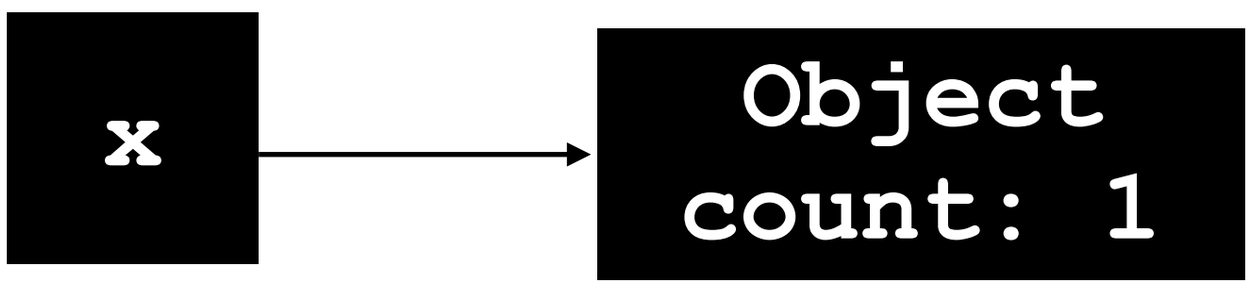


Other Example

```
function example() {
  var x = new Object();
  x.foo = new Object();
  return x.foo;
}
```

Other Example

```
function example() {
    var x = new Object();
    x.foo = new Object();
    return x.foo;
}
```



Other Example

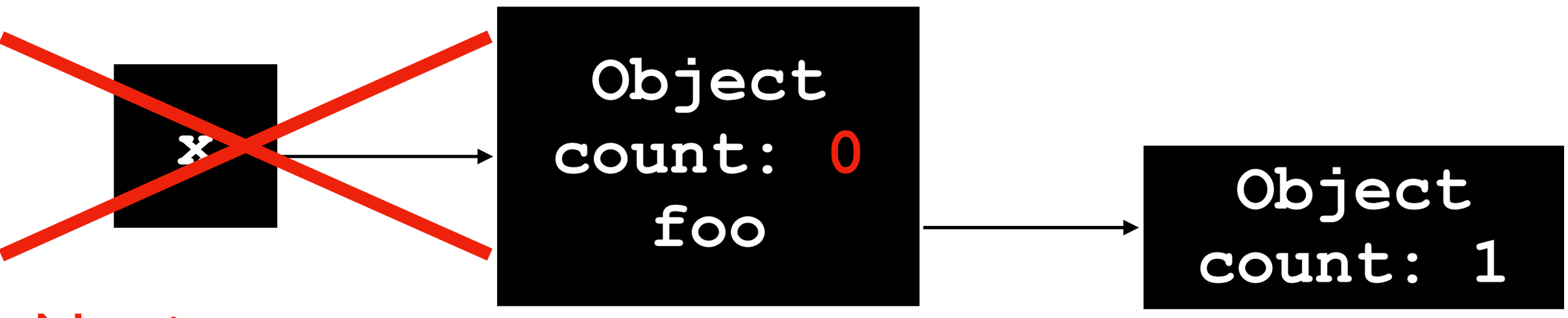
```
function example() {
   var x = new Object();
   x.foo = new Object();
   return x.foo;
}
```

Other Example

```
function example() {
   var x = new Object();
   x.foo = new Object();
   return x.foo;
}
```

Other Example

```
function example() {
   var x = new Object();
   x.foo = new Object();
   return x.foo;
}
```



Not in scope after return

Other Example

```
function example() {
   var x = new Object();
   x.foo = new Object();
   return x.foo;
}
```

Not in scope after return

Gets deleted...

Other Example

```
function example() {
   var x = new Object();
   x.foo = new Object();
   return x.foo;
}
```

Exercise: Code Snippets with Reference Counting

Reference Counting Issue

- Cycles don't properly get reclaimed
- In practice, we need either a user-exposed way to forcibly decrement a count, or garbage collection augmentation

Garbage Collection

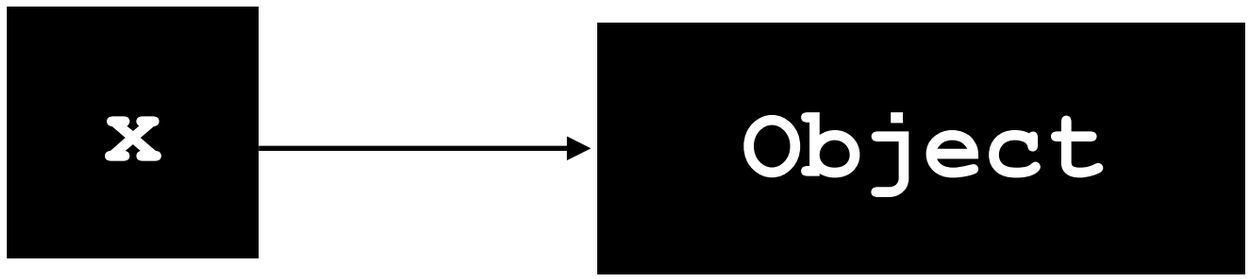
- Main idea: build a *root set* of values, usually based on variables in scope
- Treat memory like a directed graph, and record which objects are reachable from values in the root set
- Delete everything that isn't reachable

Example

```
var x = new Object();
```

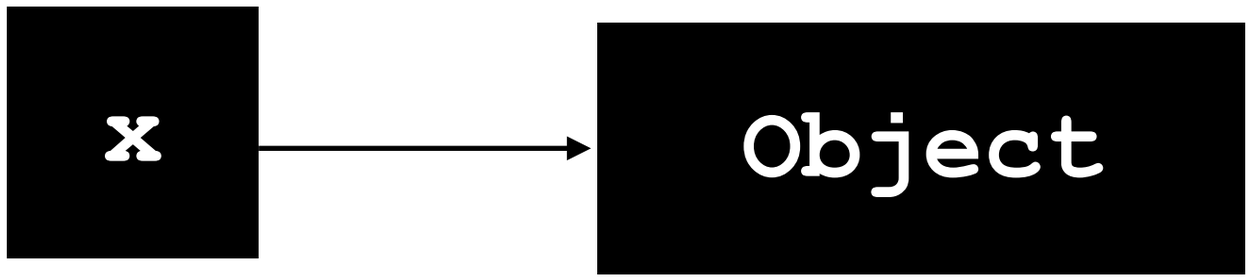
Example

```
var x = new Object();
```



Example

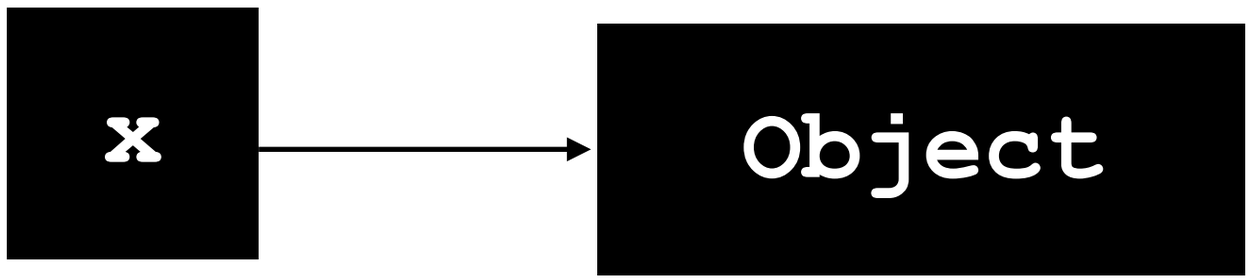
```
var x = new Object();
```



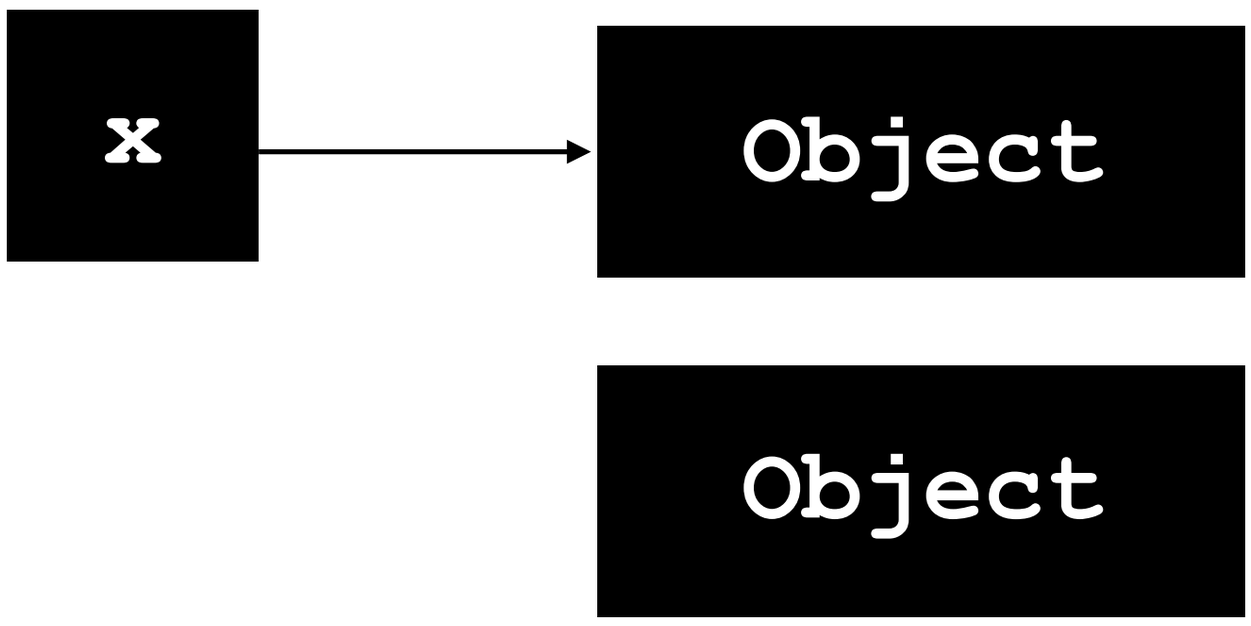

```
x = new Object();
```

Example

```
var x = new Object();
```




```
x = new Object();
```



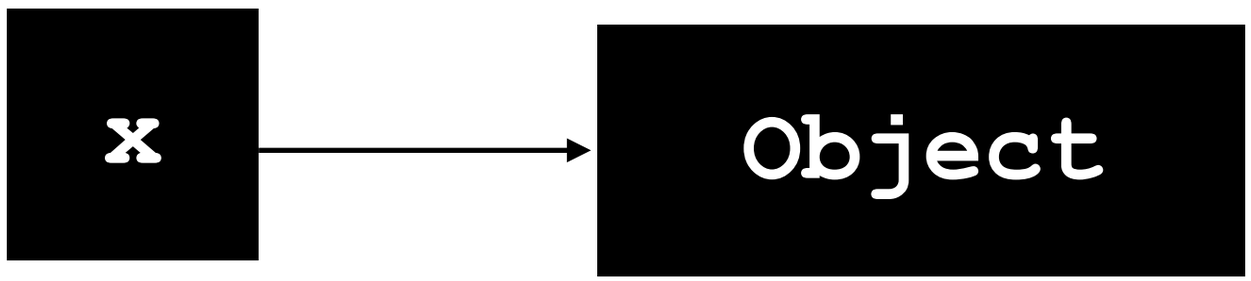
Example

```
var x = new Object();
```

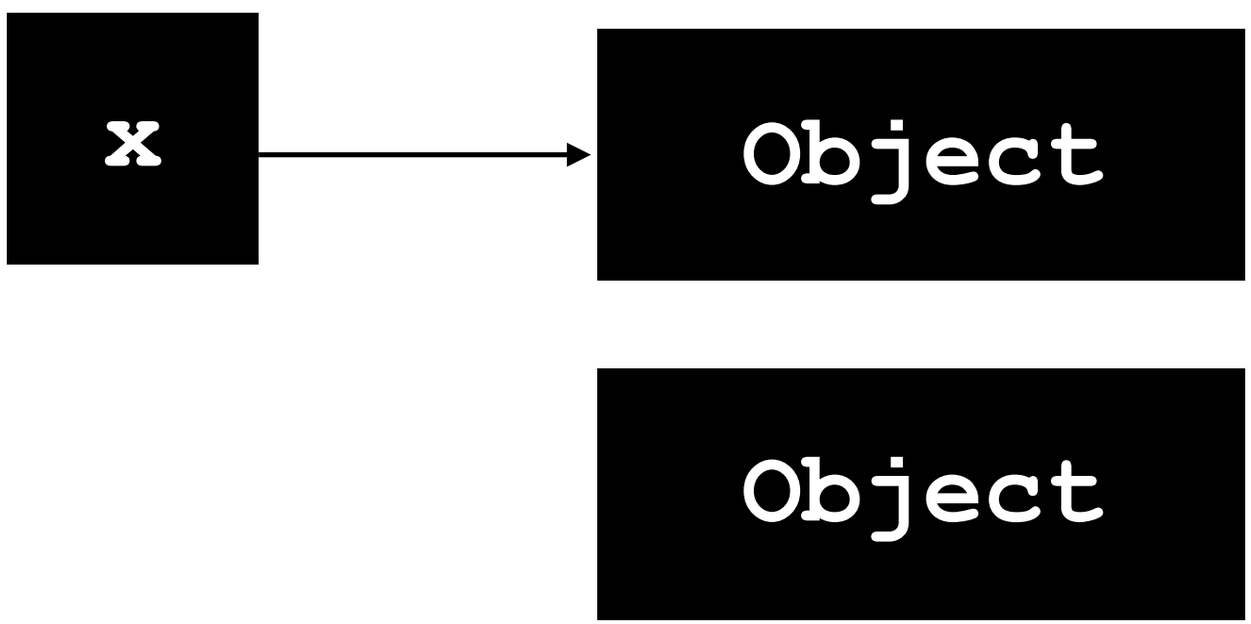
GC Starts

Root set:

x




```
x = new Object();
```

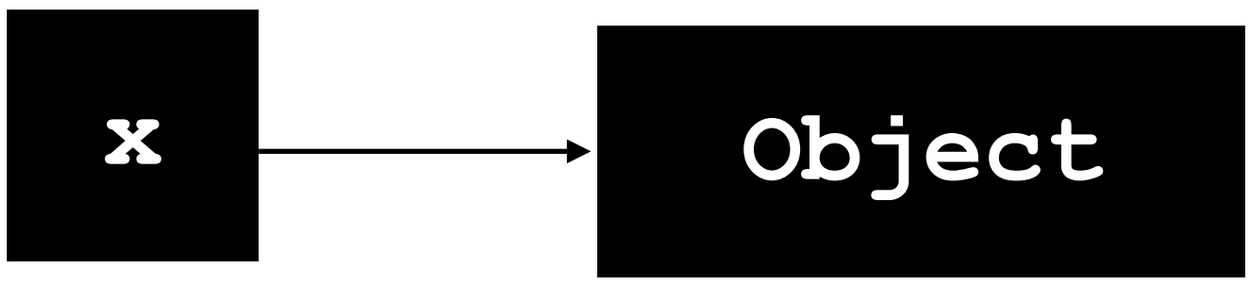


Example

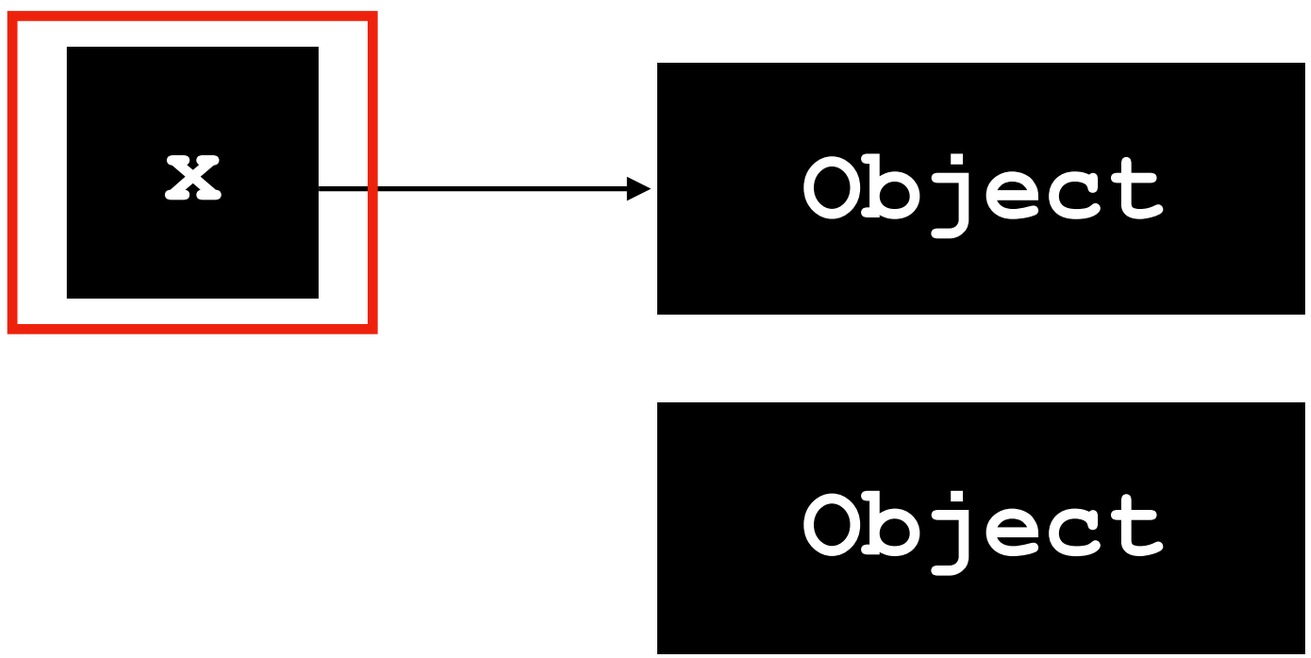
```
var x = new Object();
```

GC Starts
Root set:

x




```
x = new Object();
```

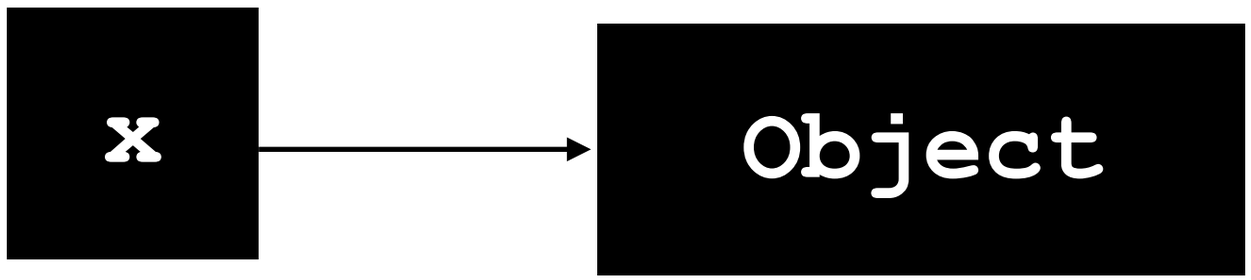


Example

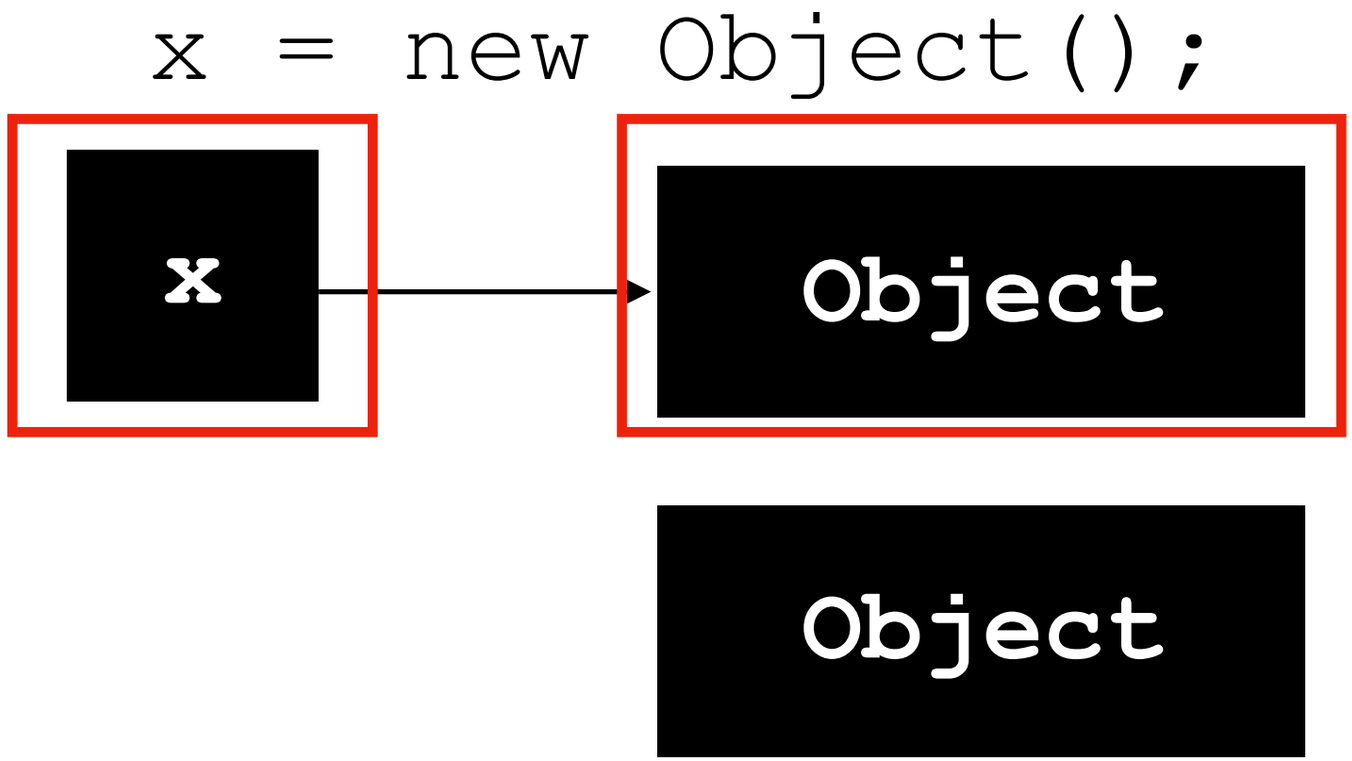
```
var x = new Object();
```

GC Starts
Root set:

x




```
x = new Object();
```

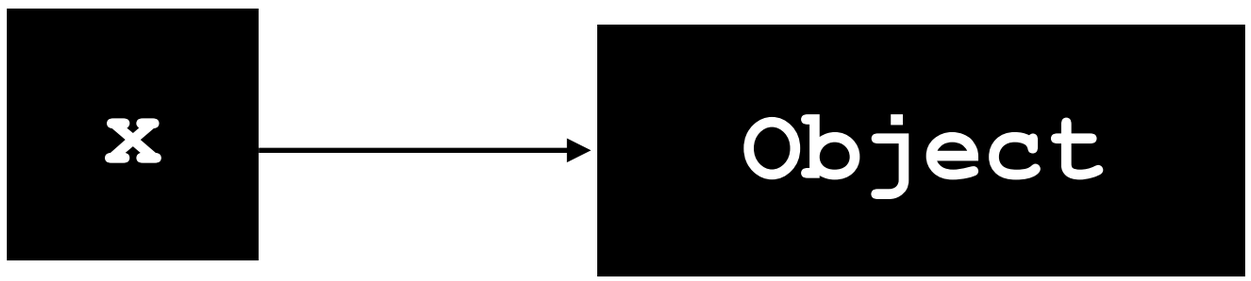


Example

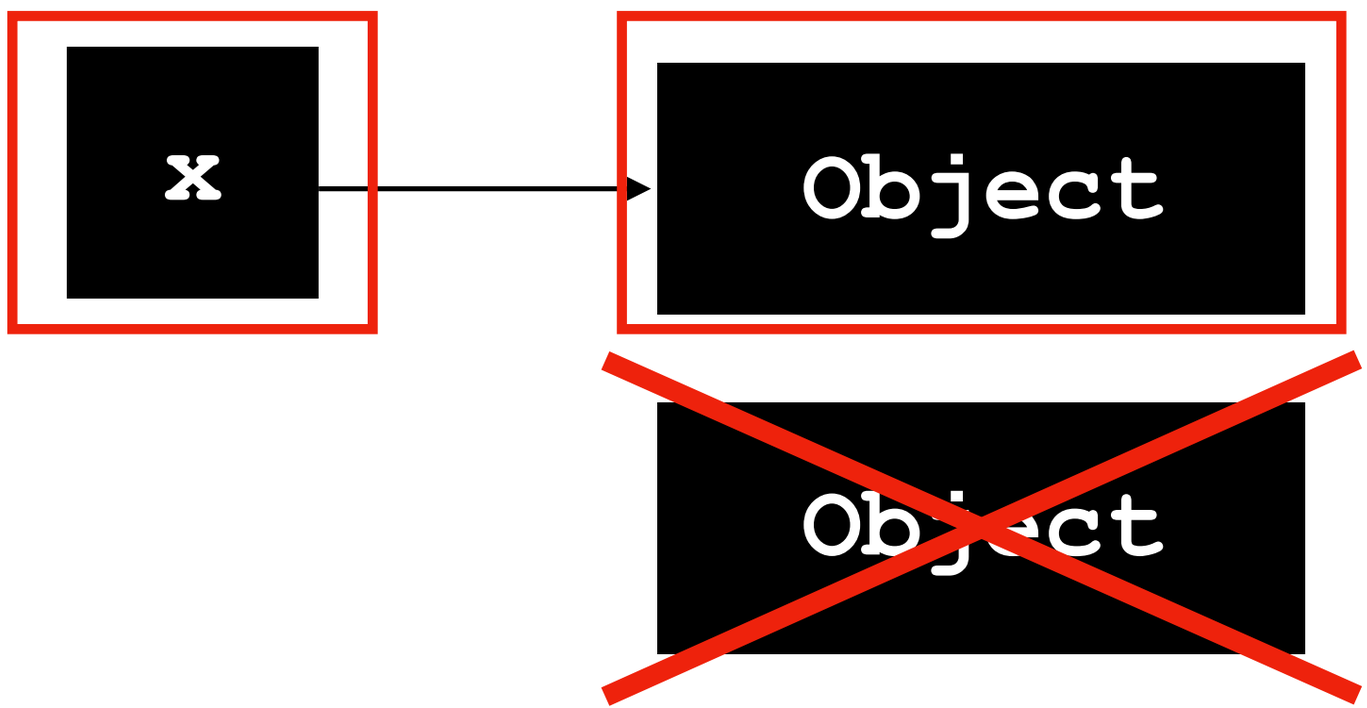
```
var x = new Object();
```

GC Starts
Root set:

x




```
x = new Object();
```



Exercise: Code Snippets with Garbage Collection

Reference Counting vs. GC

- Reference counting is good for real time systems
- GC tends to be faster overall, but incurs sporadic pauses
- GC can collect everything without user support
- GC tends to be more popular (Java/JVM, JavaScript, Go, Ruby), but reference counting is common (Swift, mostly Python, some of Rust)