COMP 250

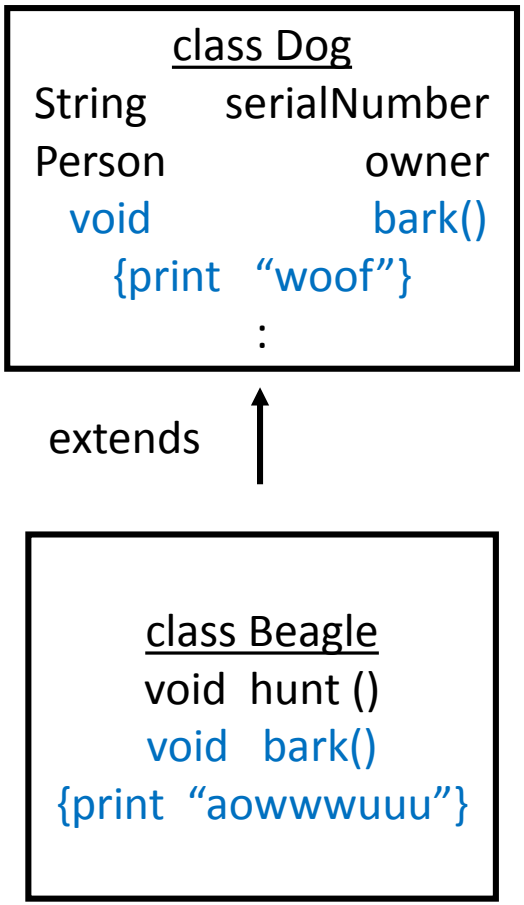
Lecture 34

Polymorphism (continued.)

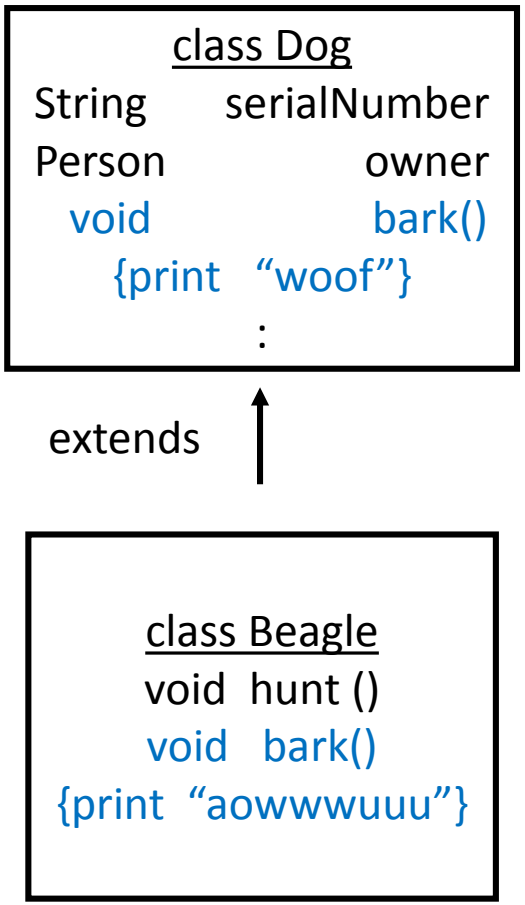
Garbage Collection
(mark and sweep)

Nov. 27, 2017

Recall last lecture

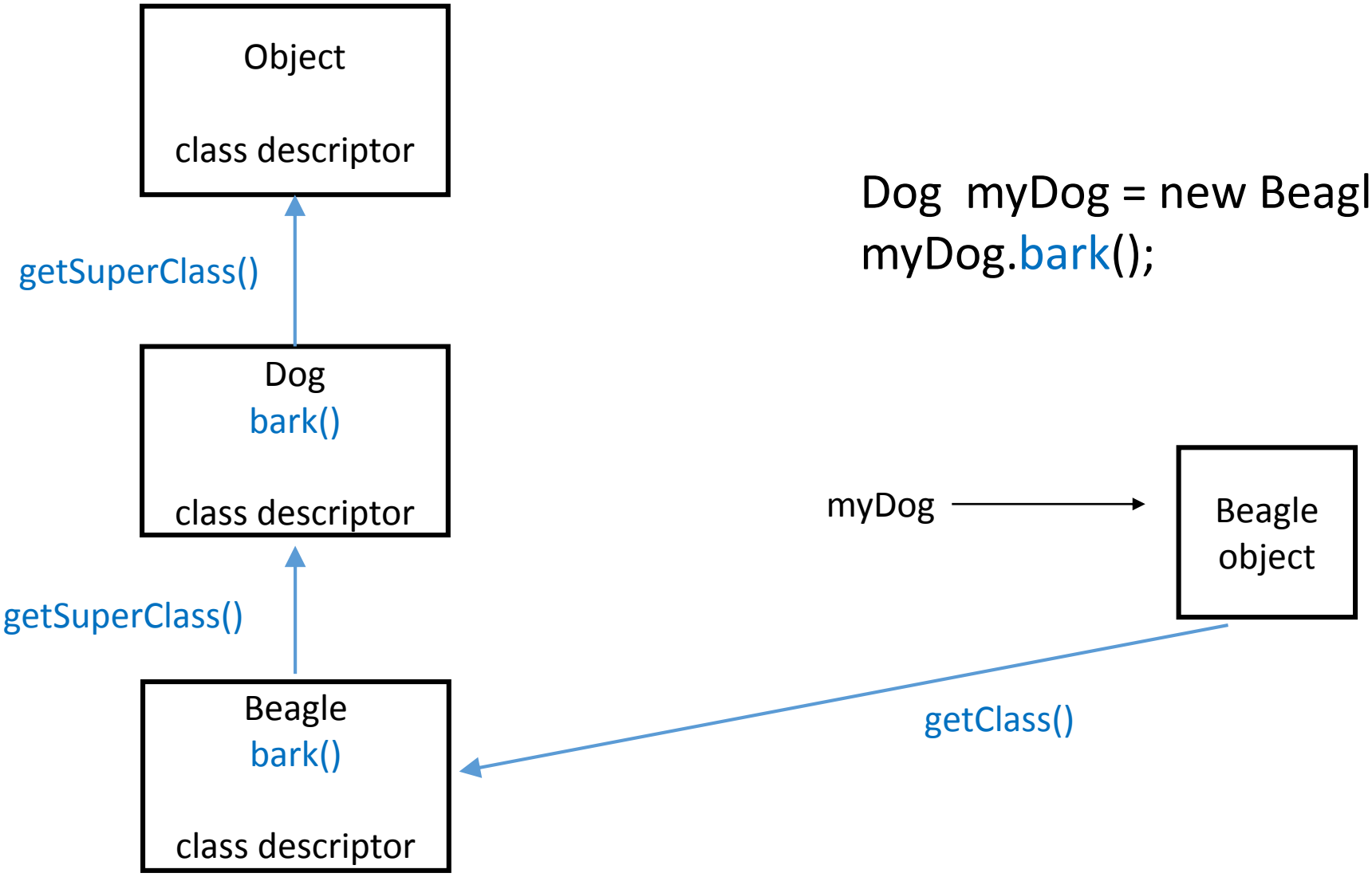


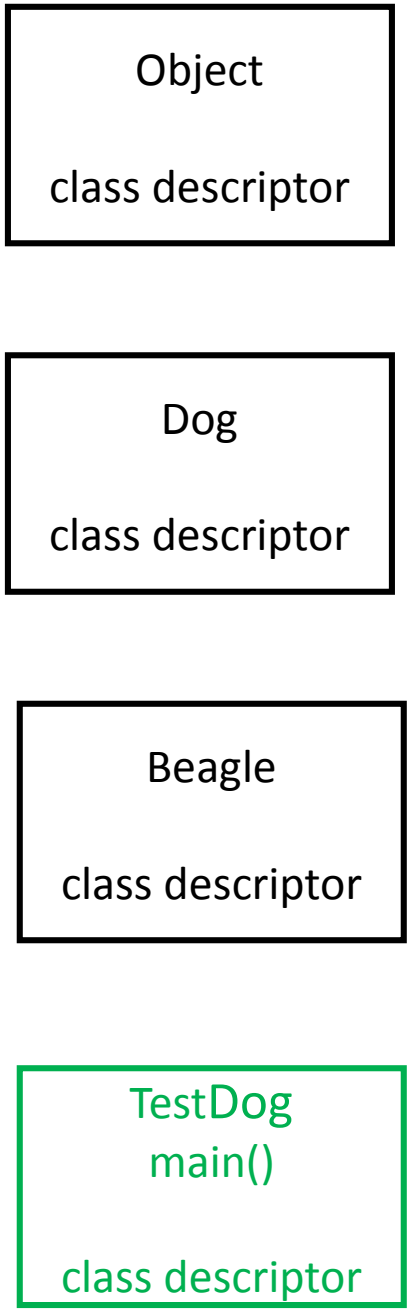
Recall last lecture




```
Dog  myDog = new Beagle();
myDog.bark();
```

This figure shows objects
in a running Java program.





Suppose we are running a class TestDog, which has a main() method.

Object

class descriptor

Dog

class descriptor

Beagle

class descriptor

TestDog
main()

class descriptor

Suppose we are running a class TestDog, which has a main() method.

TestDog.main()

Call Stack

There are no objects at the start of execution.

Objects

Object

class descriptor

Dog

class descriptor

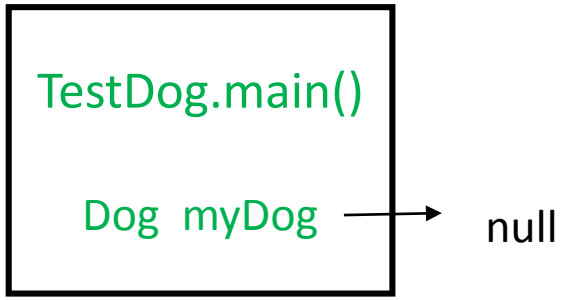
Beagle

class descriptor

TestDog

class descriptor

```
public static void main(){
        Dog    myDog = new Beagle();
               myDog.bark()
                :
}
```



Call Stack

Objects

Object

class descriptor

Dog

class descriptor

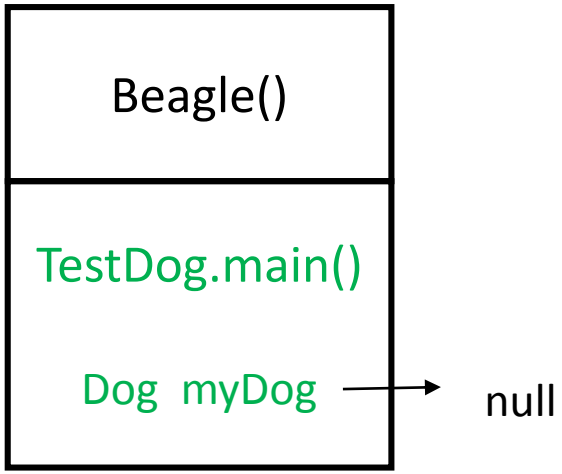
Beagle

class descriptor

TestDog

class descriptor

```
public static void main(){
        Dog    myDog = new Beagle();
                myDog.bark()
                :
}
```



Call Stack

Objects

(Beagle constructor called)

Object

class descriptor

Dog

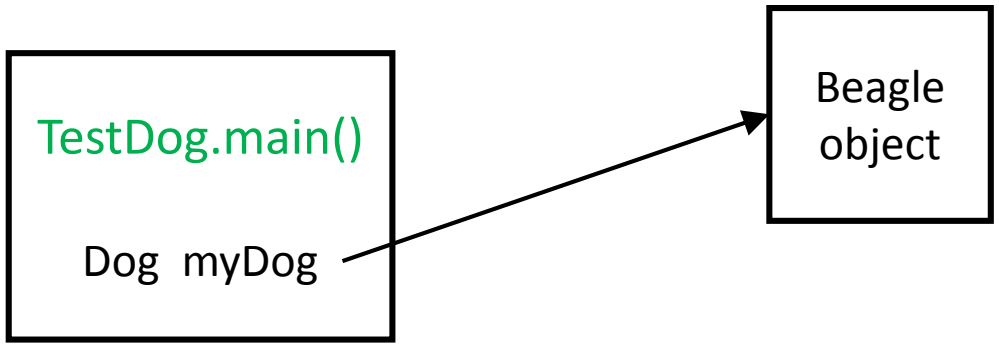
class descriptor

Beagle
class descriptor

TestDog

class descriptor

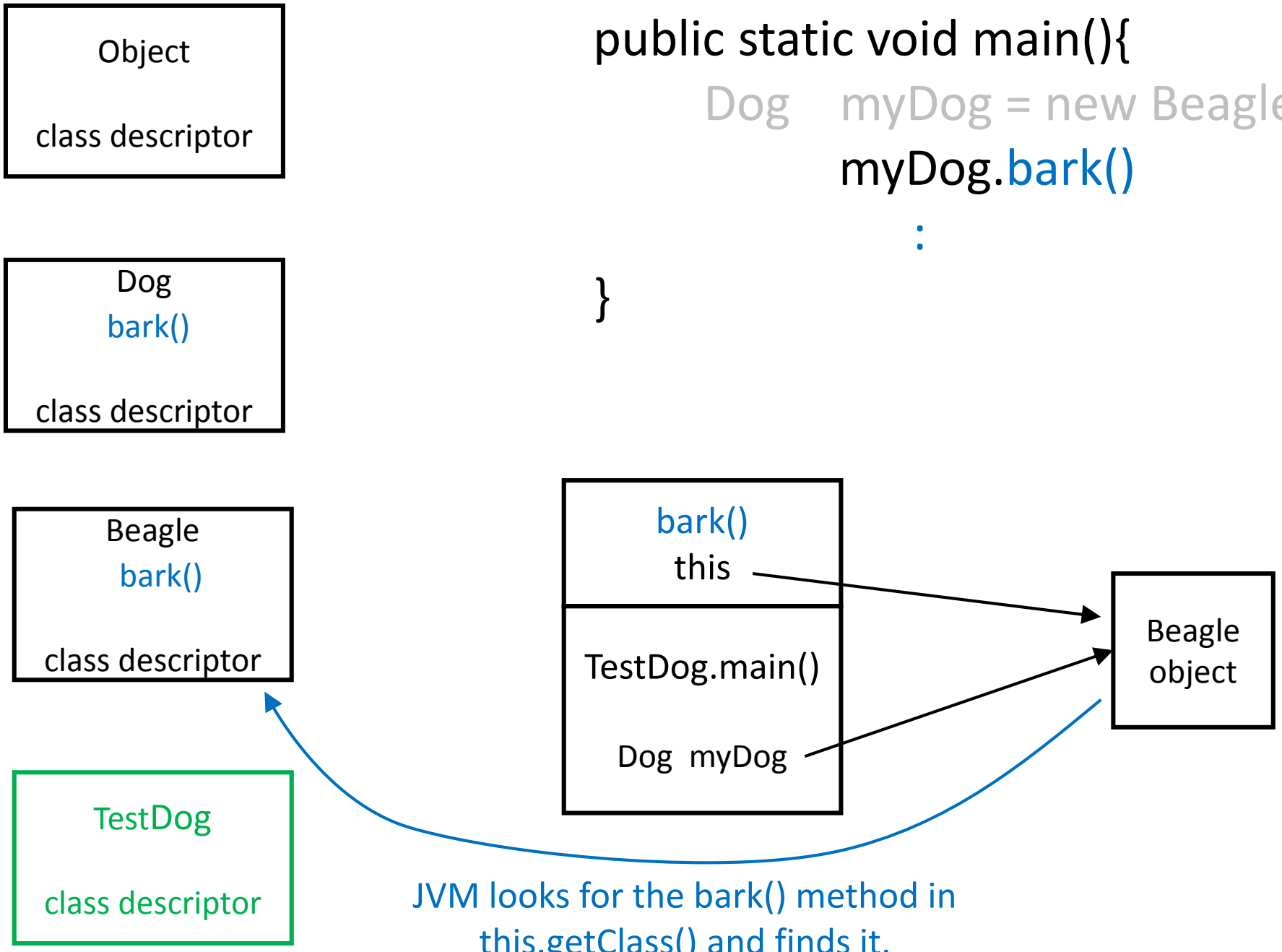
```
public static void main(){
        Dog    myDog = new Beagle();
               myDog.bark()
               :
}
```

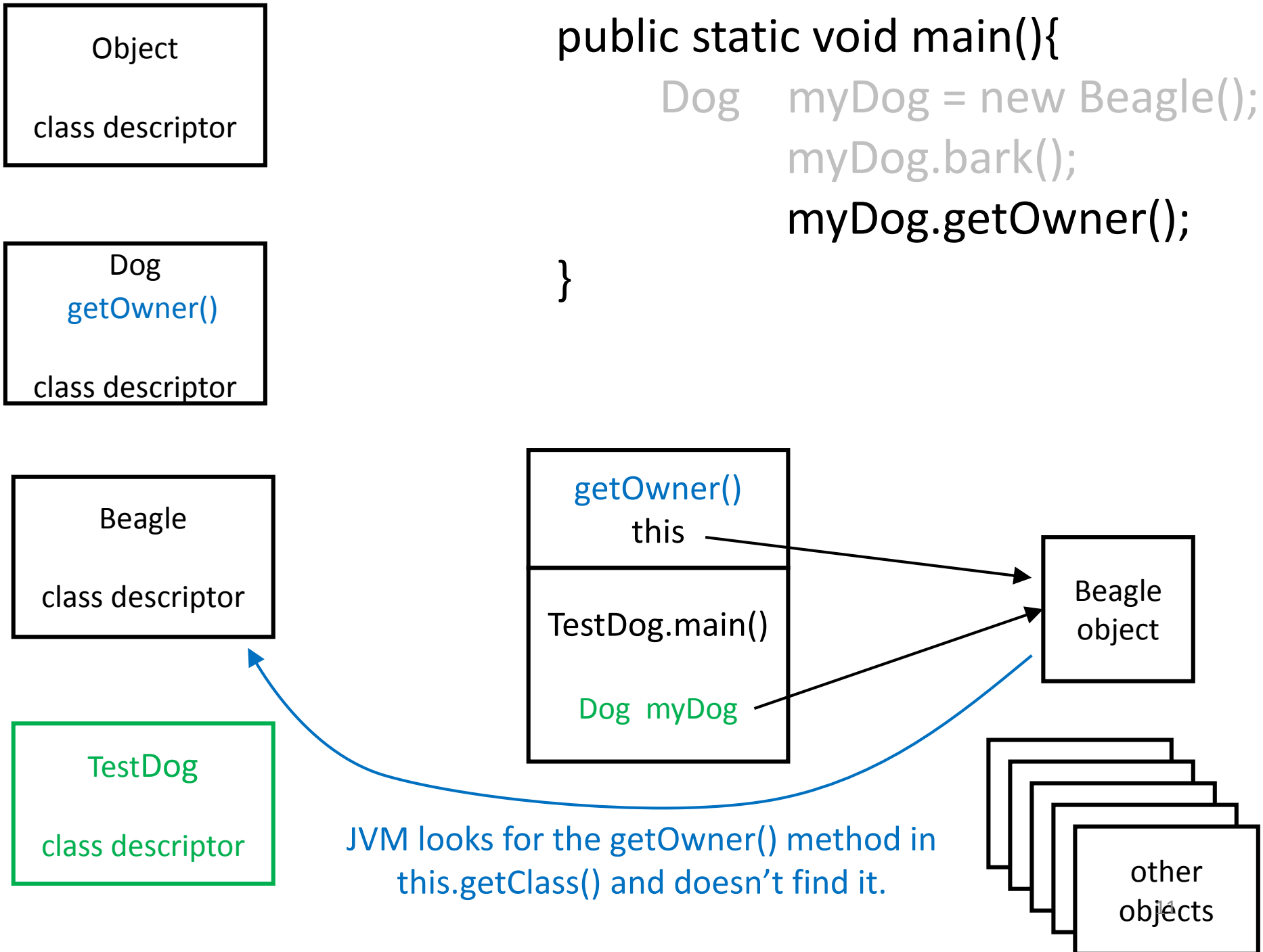


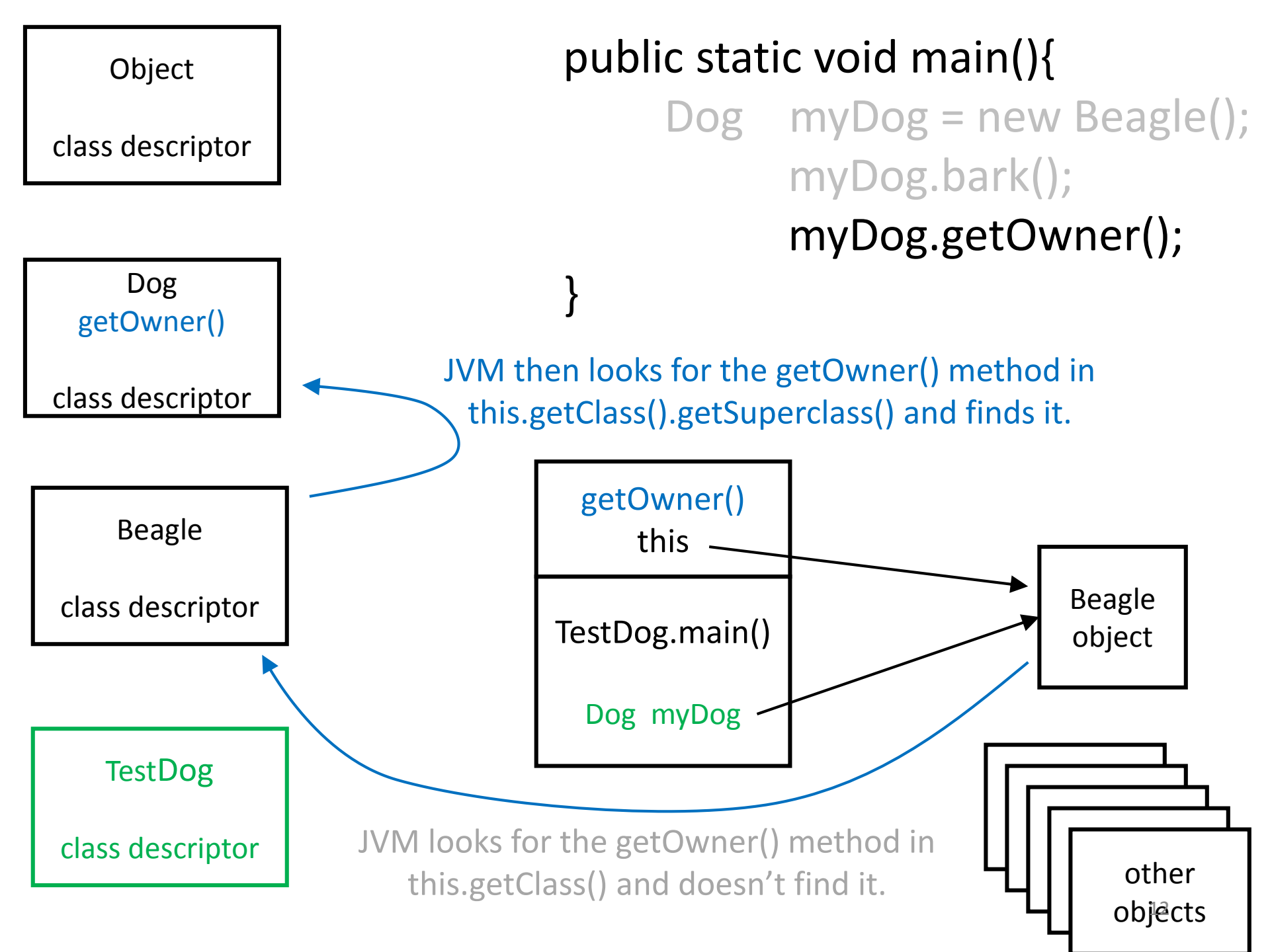
Call Stack

Objects

(after constructor is done)


Object
class descriptor
Dog
bark()
class descriptor
Beagle
bark()
class descriptor
TestDog
class descriptor
public static void main(){
Dog myDog = new Beagle();
myDog.bark()
:
}
bark()
this
TestDog.main()
Dog myDog
Beagle
object
JVM looks for the bark() method in
this.getClass() and finds it.


Object
class descriptor
Dog
getOwner()
class descriptor
Beagle
class descriptor
TestDog
class descriptor
public static void main(){
Dog myDog = new Beagle();
myDog.bark();
myDog.getOwner();
}
getOwner()
this
TestDog.main()
Dog myDog
Beagle
object
other
objects
JVM looks for the getOwner() method in
this.getClass() and doesn't find it.


Object
class descriptor
Dog
getOwner()
class descriptor
Beagle
class descriptor
TestDog
class descriptor
public static void main(){
Dog myDog = new Beagle();
myDog.bark();
myDog.getOwner();
}
JVM then looks for the getOwner() method in this.getClass().getSuperclass() and finds it.
getOwner()
this
TestDog.main()
Dog myDog
Beagle object
JVM looks for the getOwner() method in this.getClass() and doesn't find it.
other objects

Class Descriptors

Methods are here

Object

class descriptor

Dog

class descriptor

Beagle

class descriptor

TestDog

class descriptor

Call Stack

Local variables and method parameters are here

mB()

mA()

TestDog.main()

Objects

Object instance fields are here

Beagle object

other objects

COMP 250

Lecture 34

Polymorphism (continued.)

Garbage Collection
(mark and sweep)

Nov. 27, 2017

Garbage Collection

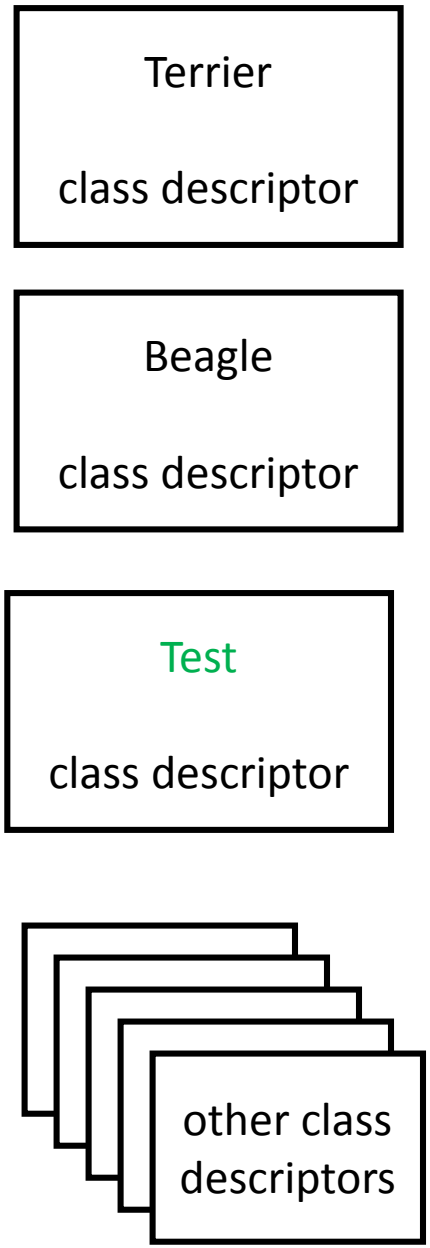
```
Dog   myDog  =  new  Beagle("Bob");

      myDog  =  new  Terrier("Tim");
```

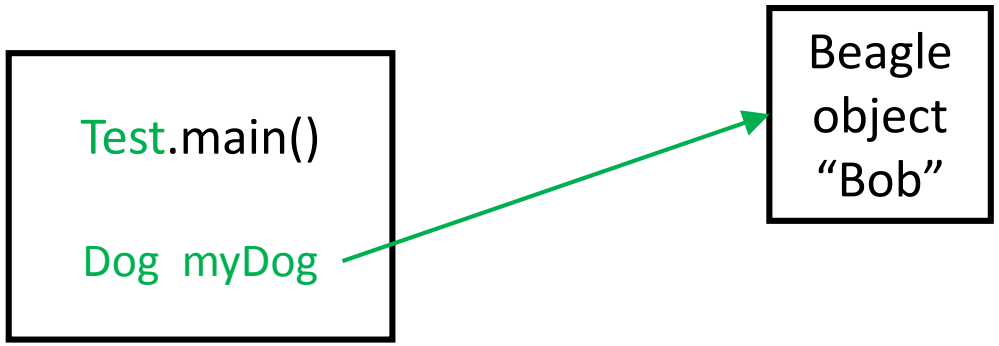
Nothing references the Bob the Beagle.

Bob is wasting memory. Bob has become garbage.

Dog myDog = new Beagle("Bob");

myDog = new Terrier("Tim");

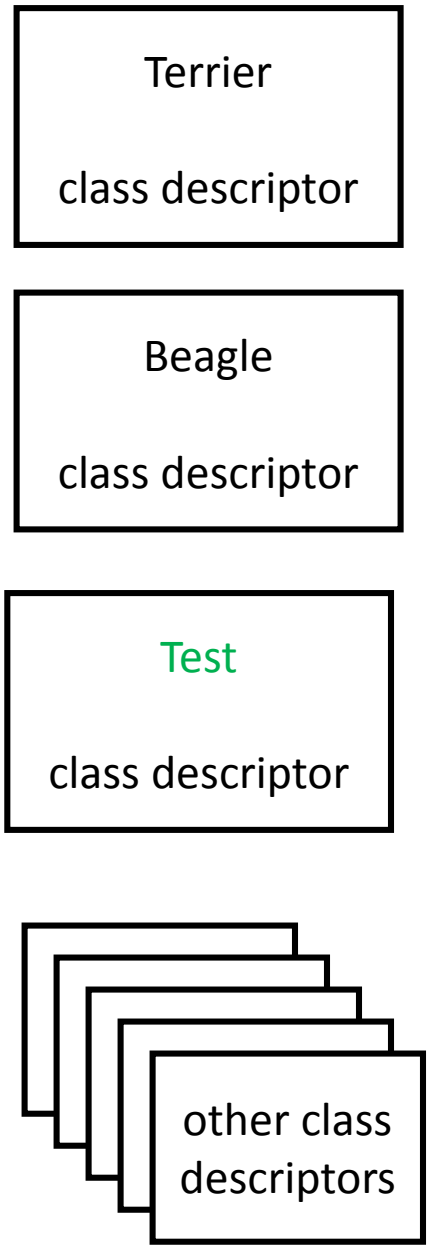


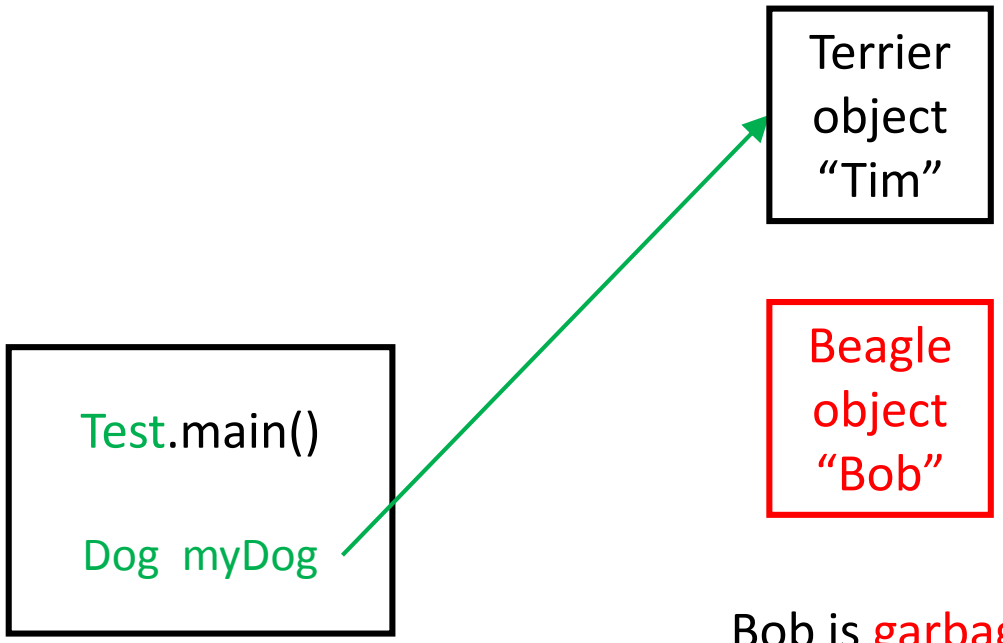


Call Stack

Dog myDog = new Beagle("Bob");

myDog = new Terrier("Tim");

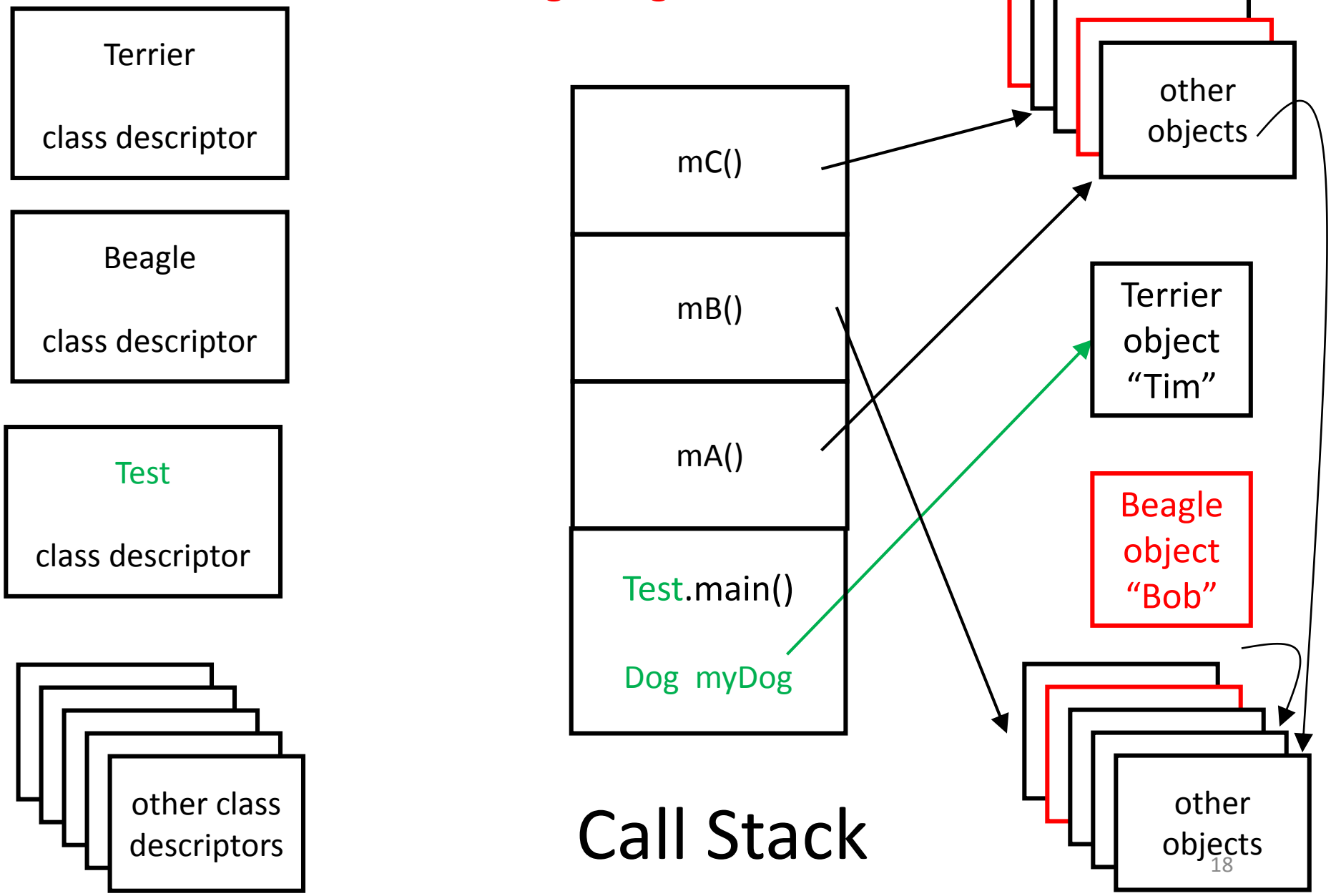


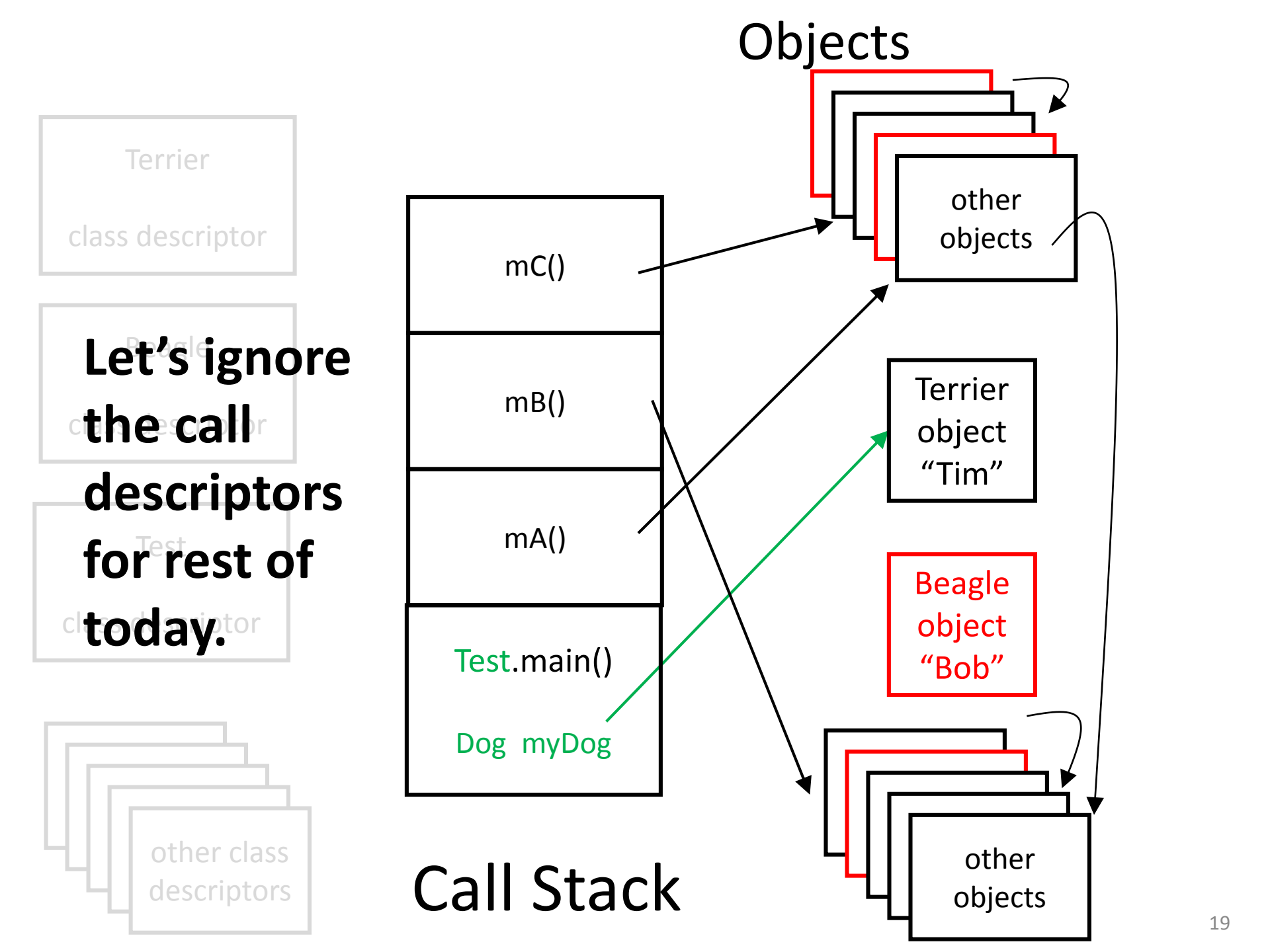


Bob is garbage.

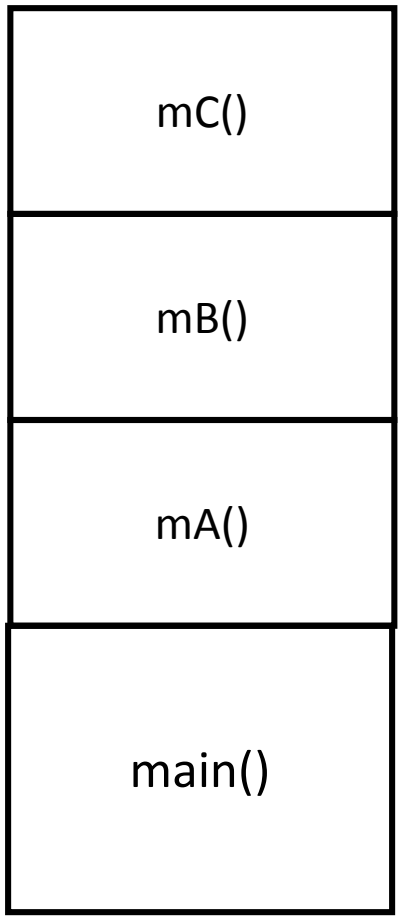
Call Stack

As the program continues,
more “garbage” accumulates.

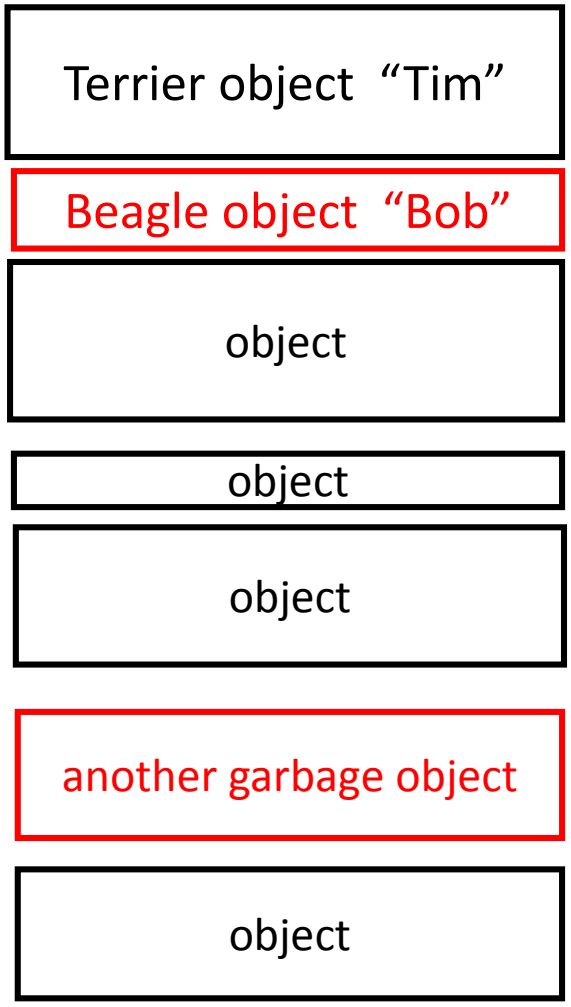



Objects
Terrier
class descriptor
Beagle
class descriptor
Test
class descriptor
Let's ignore the call descriptors for rest of today.
other class descriptors
mC()
mB()
mA()
Test.main()
Dog myDog
other objects
Terrier object "Tim"
Beagle object "Bob"
other objects
Call Stack

Every object has a location in memory: Object.hashCode().

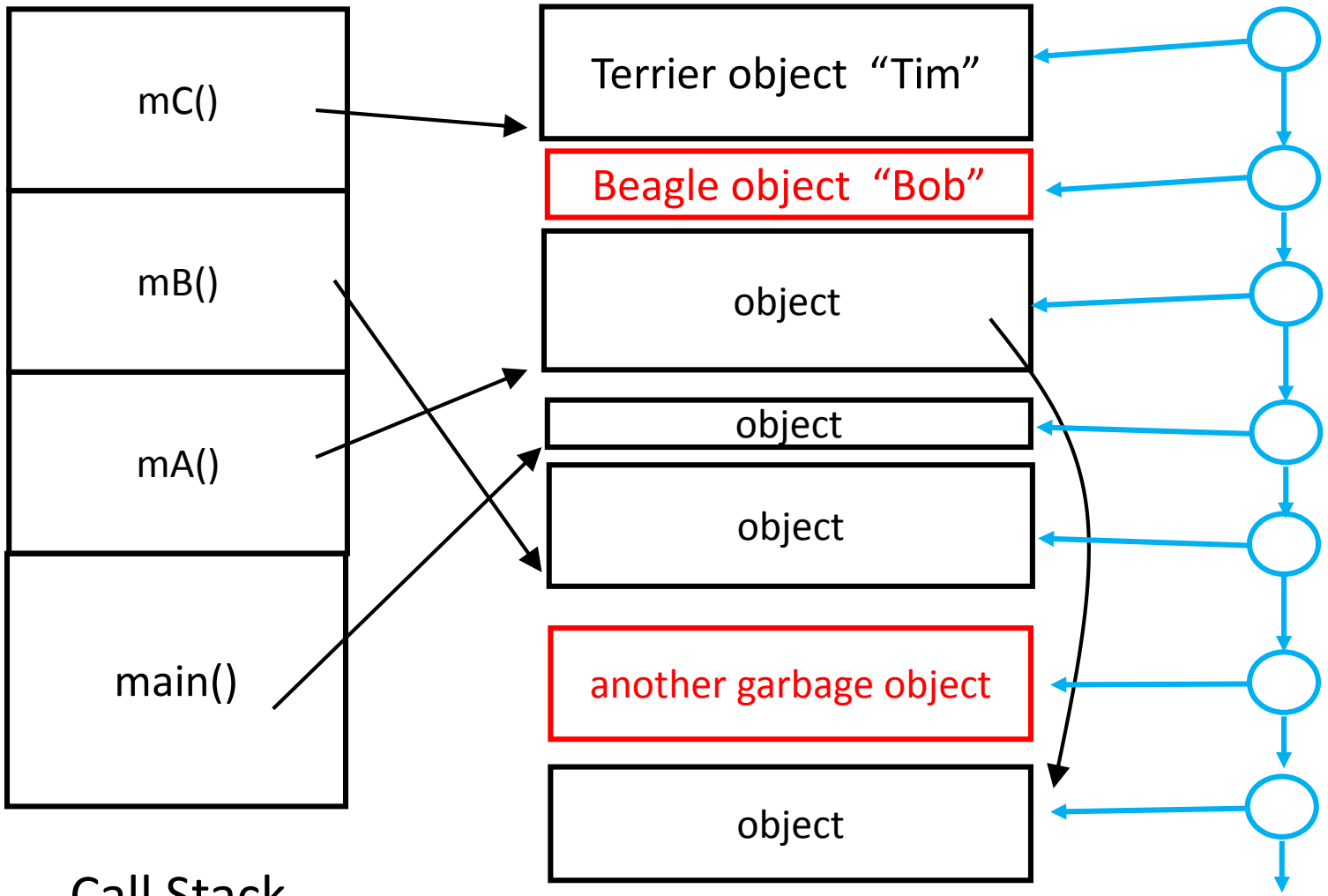


Call Stack



Objects

The Java Virtual Machine (JVM) maintains a linked list of all objects. i.e. The list stores the Object.hashCode() of each object.

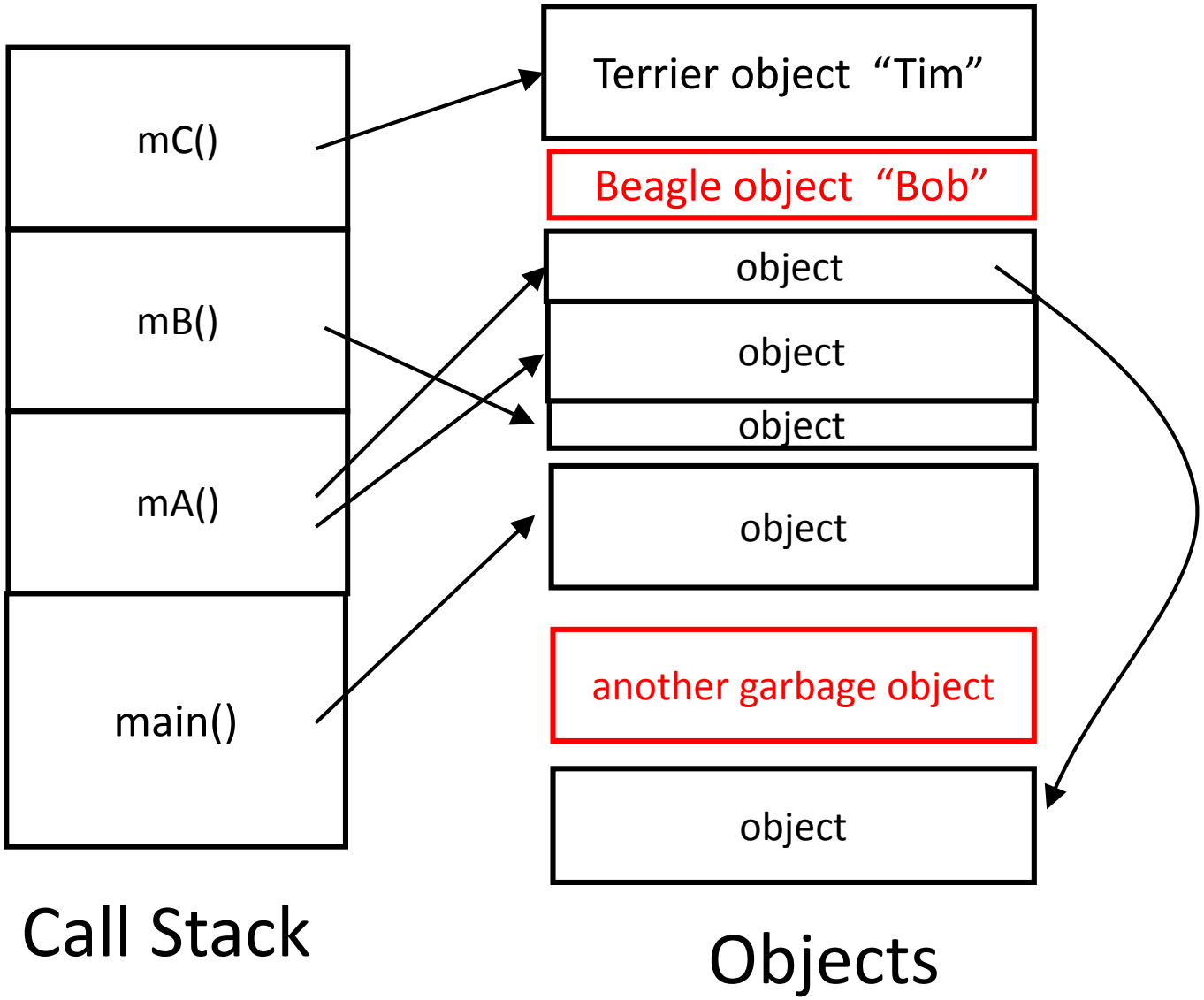


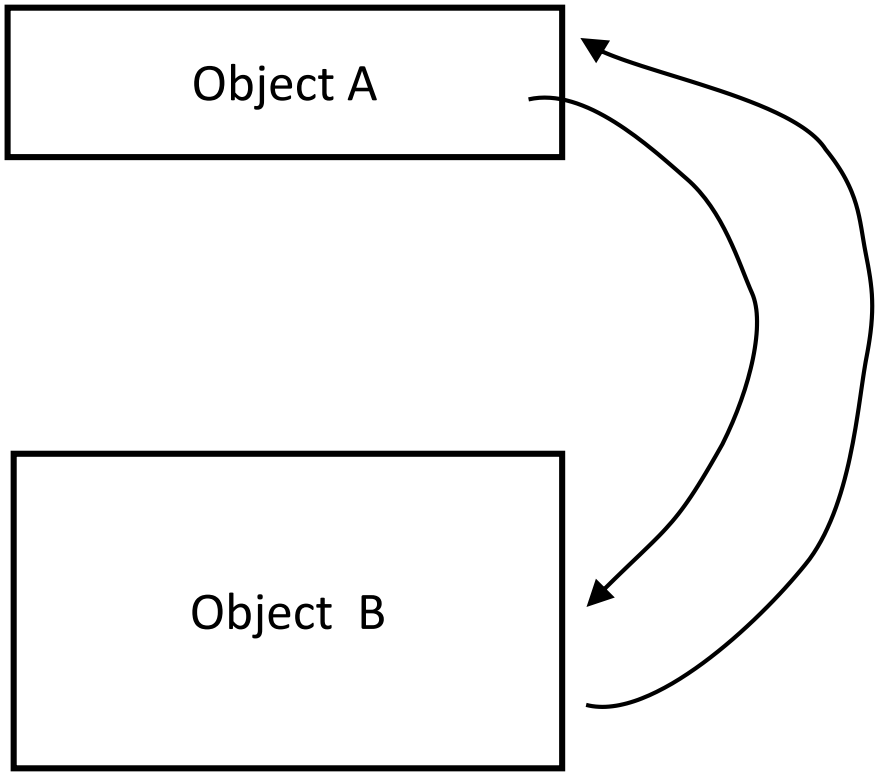
Q: What to do when object space fills up?

A: Let the program crash.

A: Reuse the space we don't need.
(Garbage collection)

"Live objects" (not garbage) are those referenced either from a call stack variable or from an instance variable in a live object.





Q: If these objects are only referenced by each other, then are they garbage ?

A: Yes, because they will never be used by the program.

Garbage collection: “Mark and Sweep”

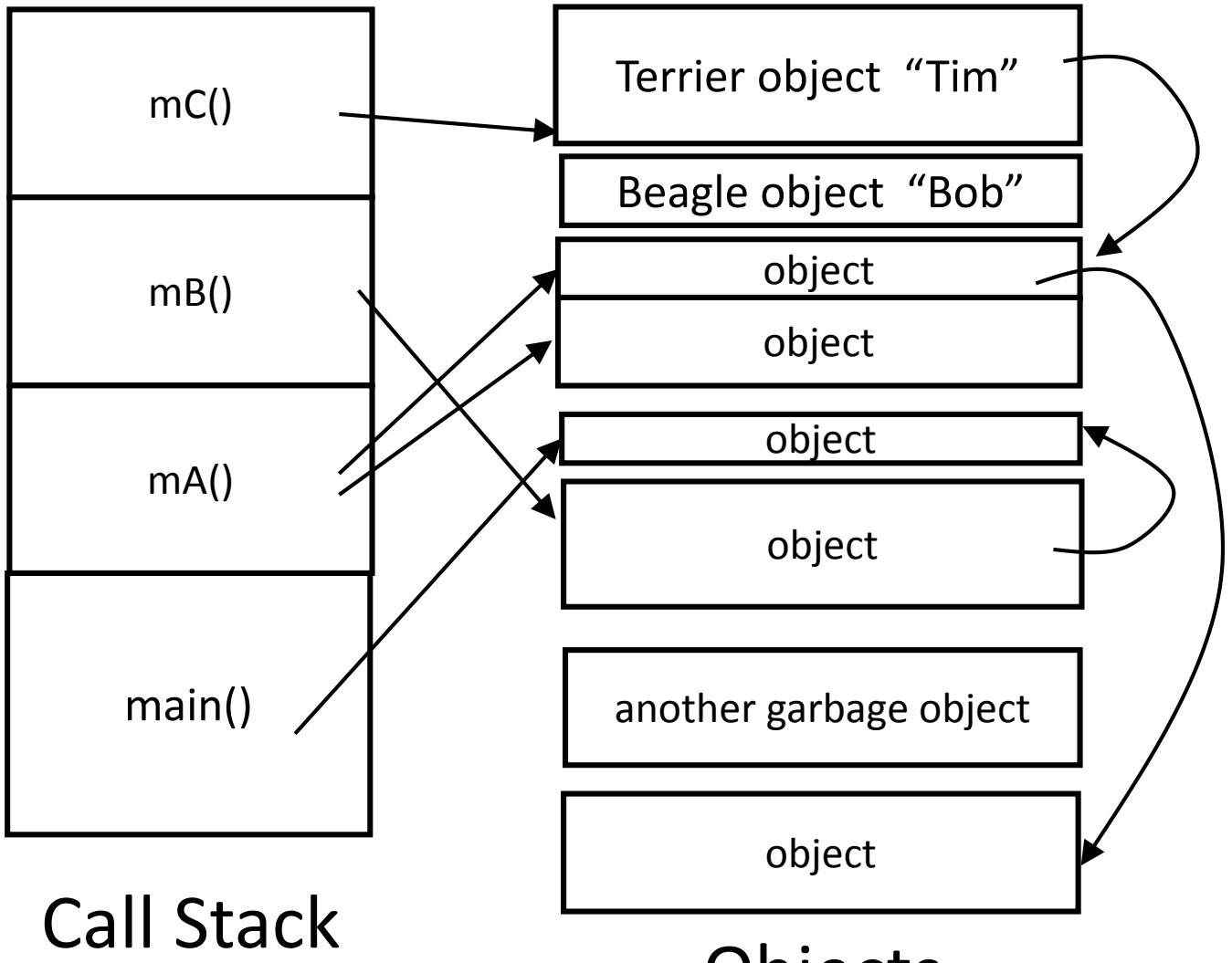
1) Build a graph, and identify live objects (“Mark”)

2) Remove garbage (“sweep”)

Garbage collector builds a graph that *corresponds* to the one here:

Vertices *correspond to* reference variables in call stack, and to objects.

Edges *correspond to* references.

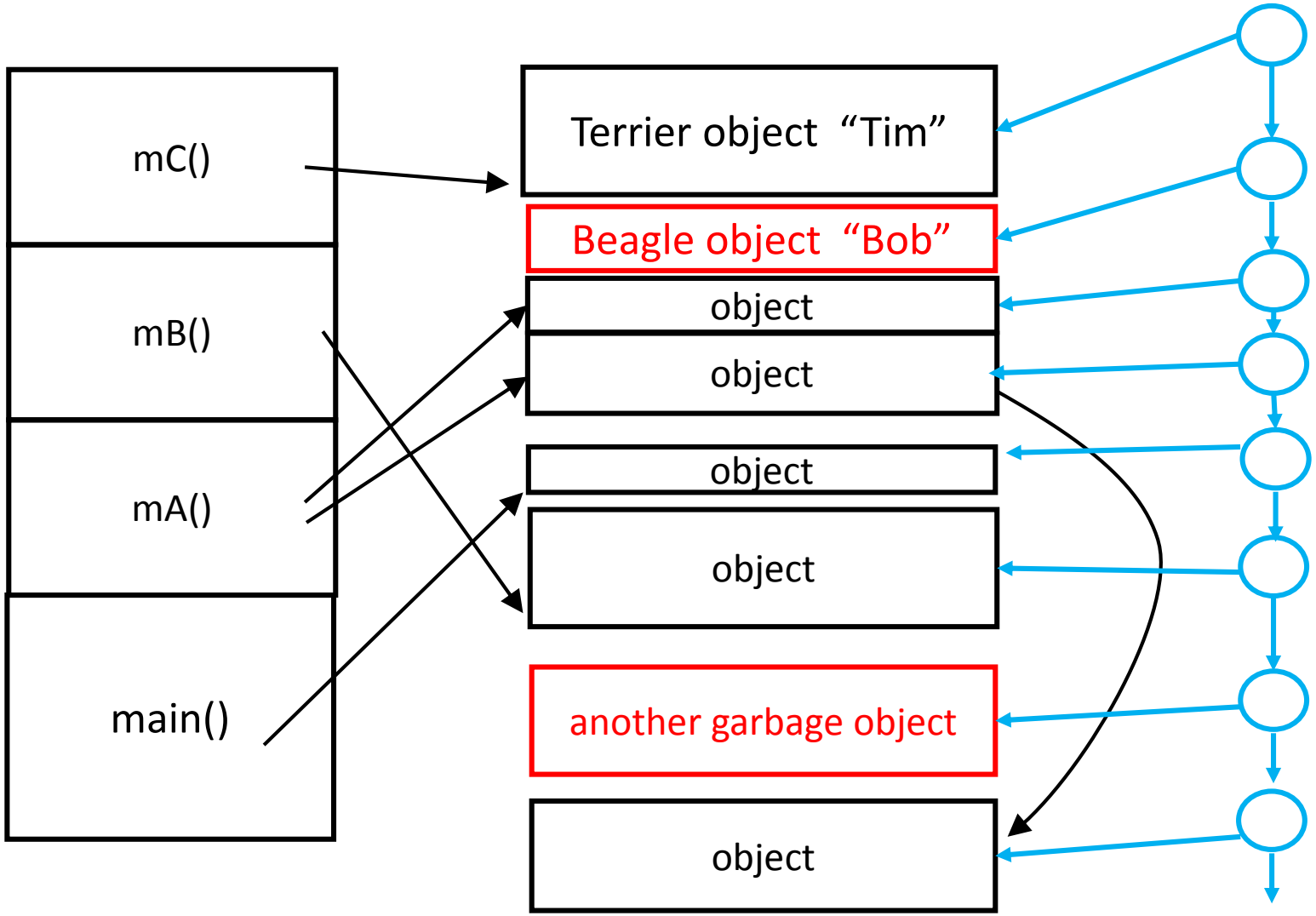


For each vertex that corresponds to a reference variable on the call stack:

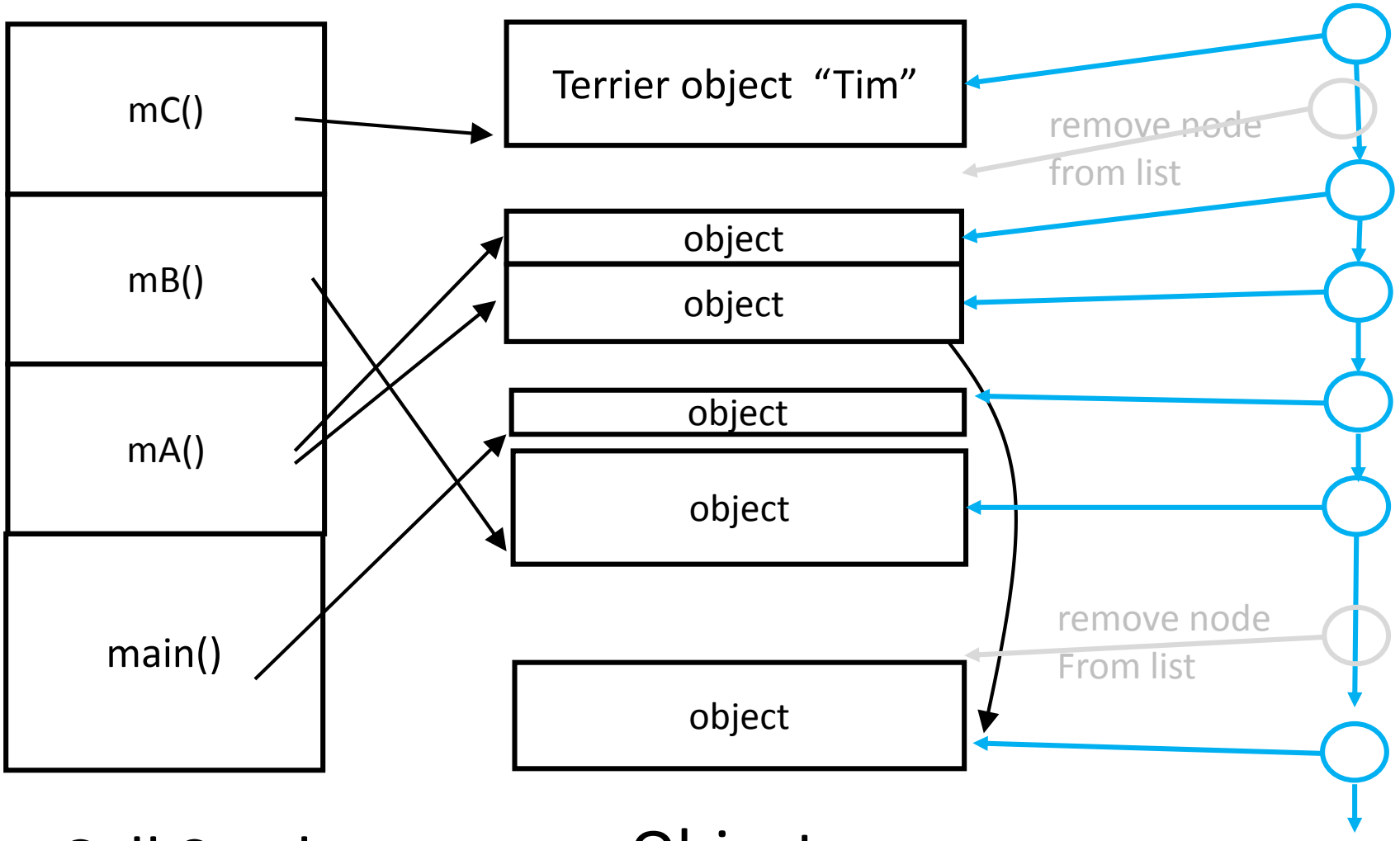
traverse the graph.

Visiting a node means ***mark*** it as live.

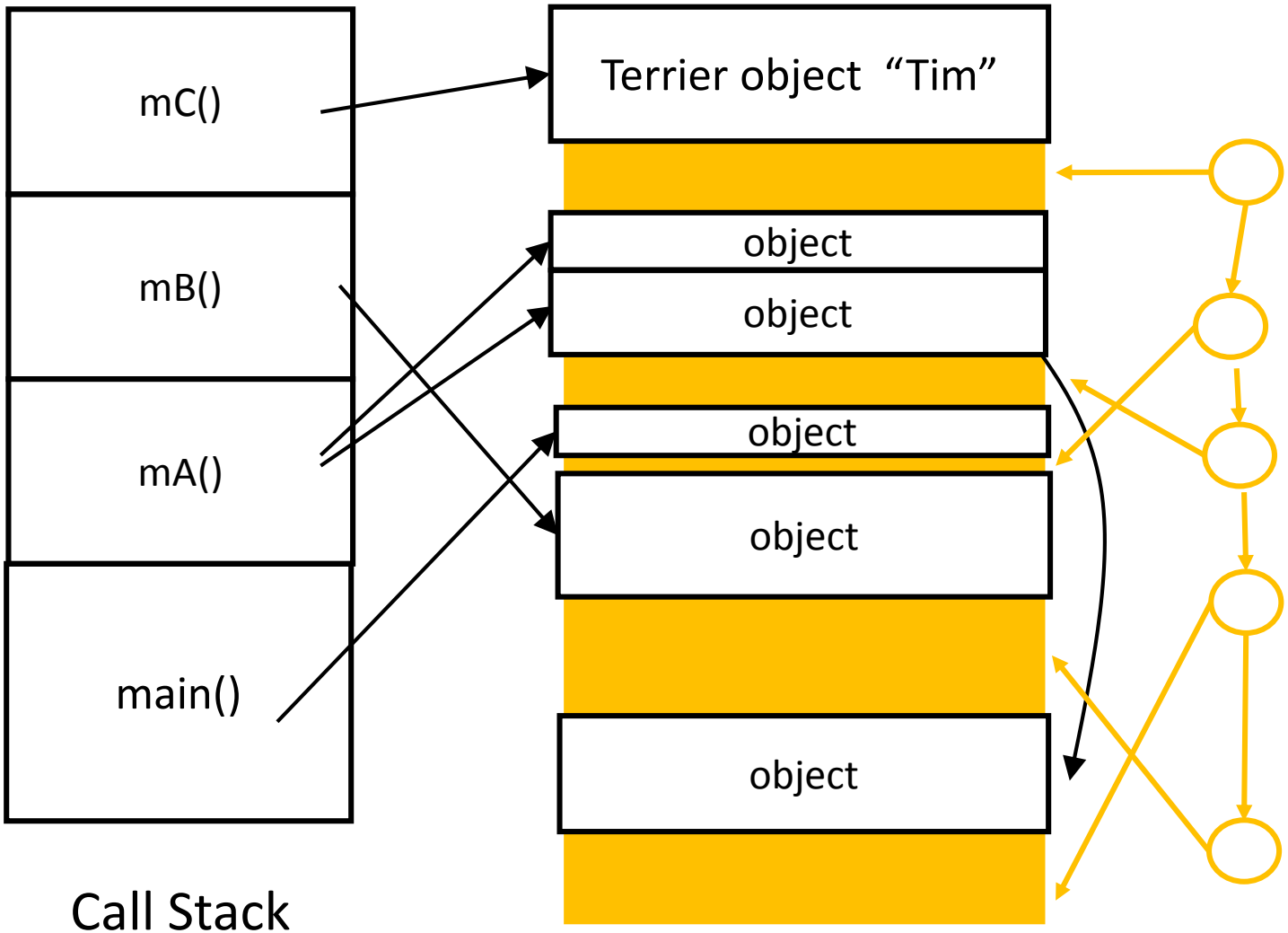
Phase 1: "Mark" the garbage

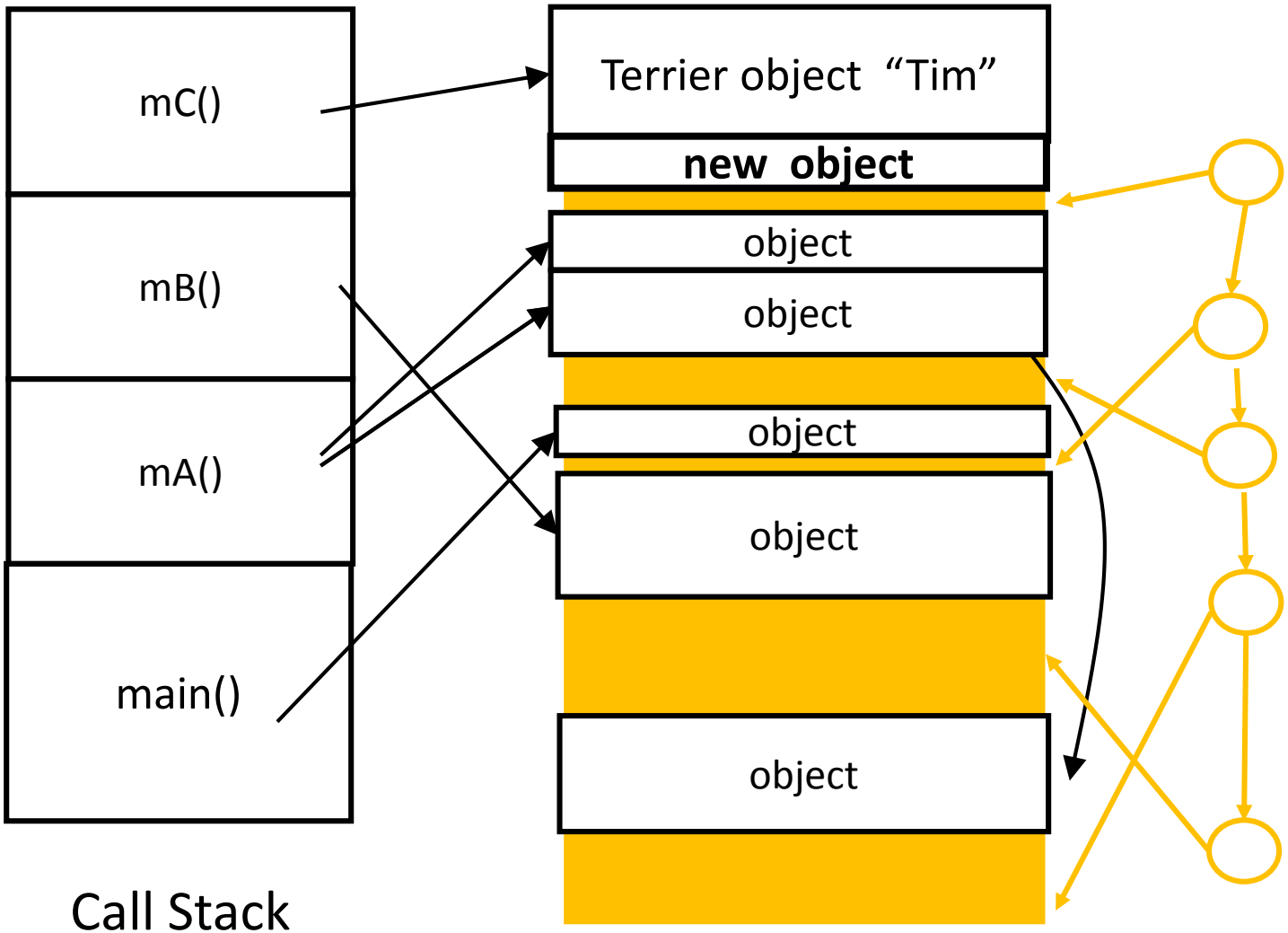


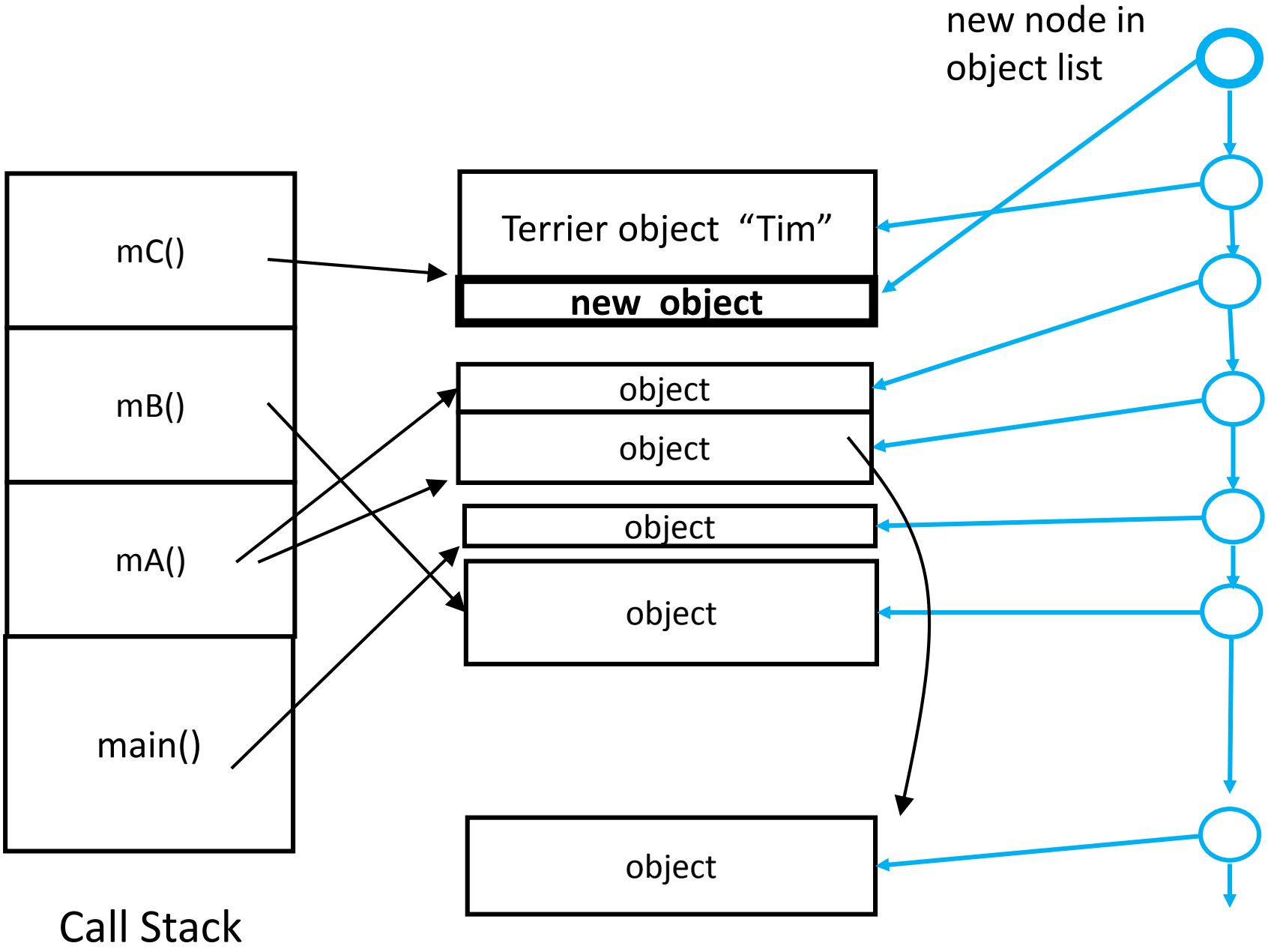
Phase 2: "Sweep" the garbage



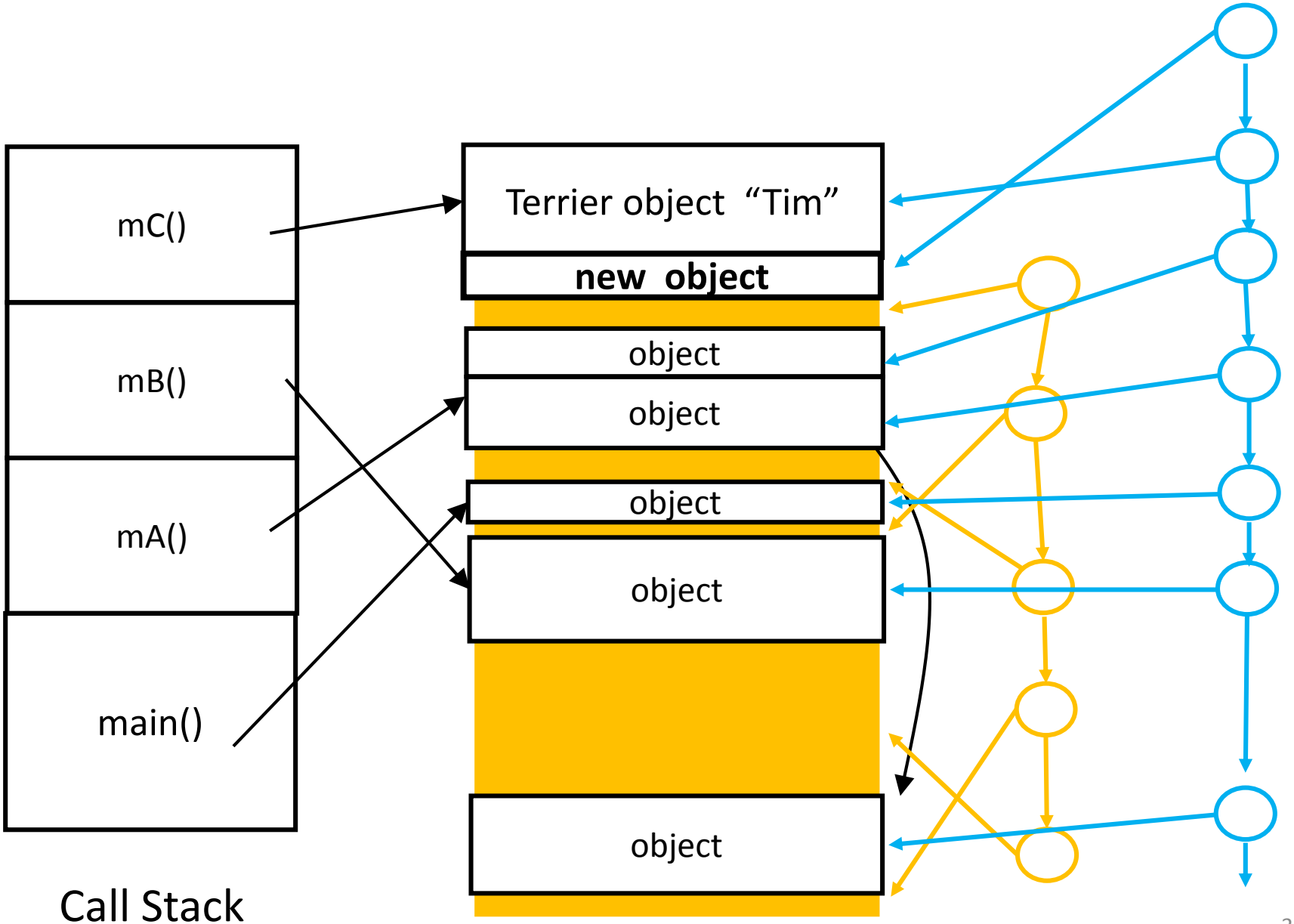
Use another list to keep track of free space between objects.




mC()
mB()
mA()
main()
Terrier object "Tim"
new object
object
object
object
object
object
Call Stack


new node in
object list
mC()
mB()
mA()
main()
Terrier object "Tim"
new object
object
object
object
object
object
Call Stack

Two lists: free space, live objects



After garbage collection, continue execution..

- New objects can be added, where there is a big enough gap in free space.
- Garbage collection is needed again when there is no gap big enough for the new object.
- Program needs to stop (temporarily) to do garbage collection. This is not good for real time time applications.