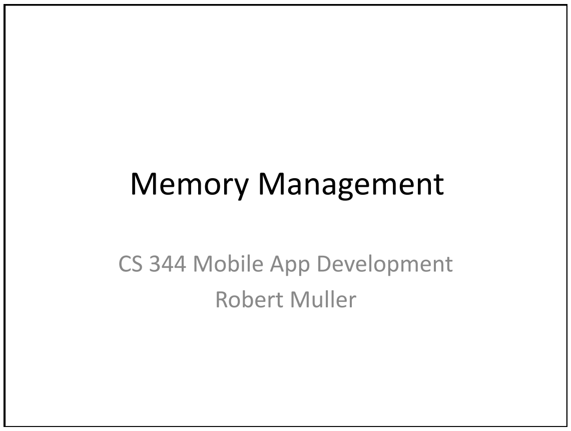
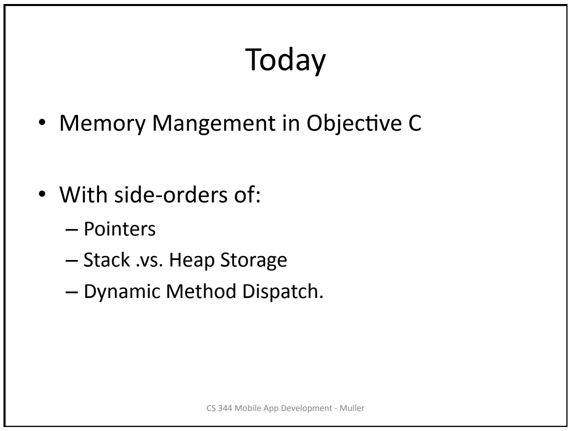


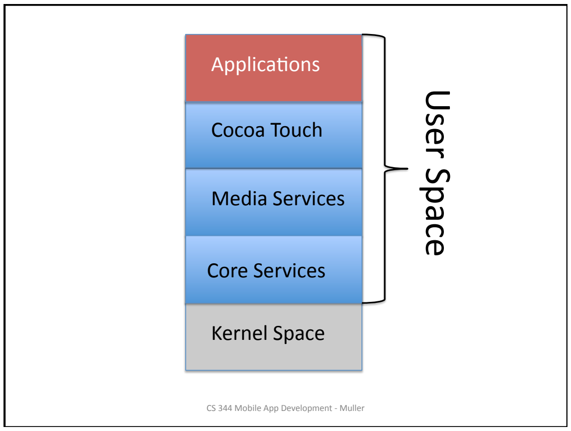
Memory Management

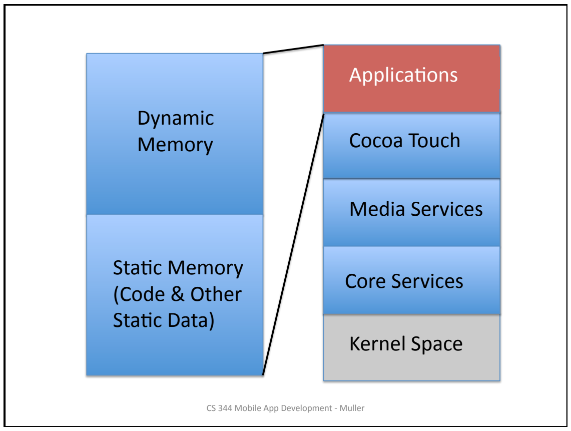
CS 344 Mobile App Development
Robert Muller



Today

- Memory Mangement in Objective C
- With side-orders of:
  - Pointers
  - Stack .vs. Heap Storage
  - Dynamic Method Dispatch.





Dynamic Memory

Static Memory (Code & Other Static Data)

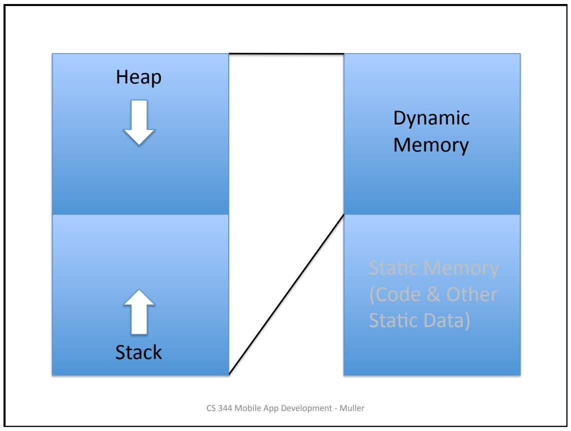
Applications

Cocoa Touch

Media Services

Core Services

Kernel Space

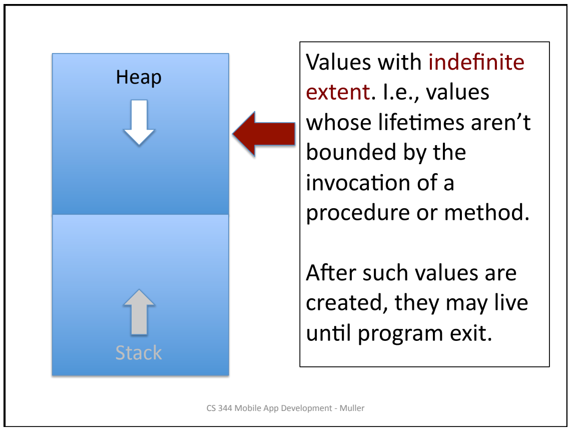


Heap

Stack

Dynamic Memory

Static Memory (Code & Other Static Data)

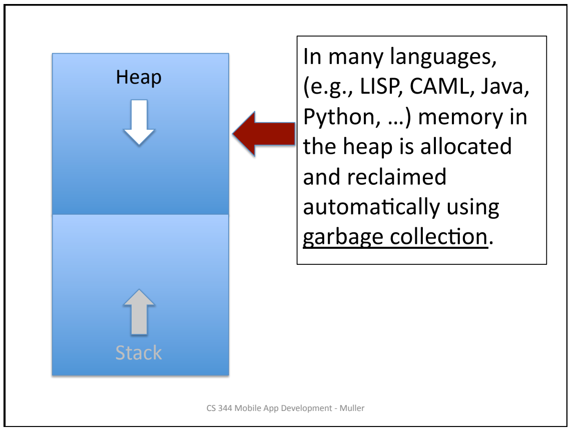


Heap

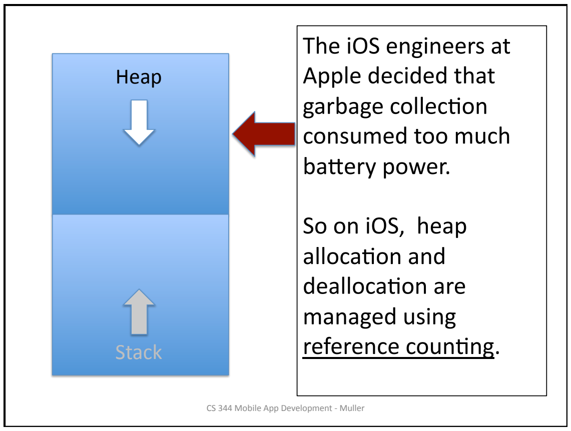
Stack

Values with indefinite extent. I.e., values whose lifetimes aren't bounded by the invocation of a procedure or method.

After such values are created, they may live until program exit.

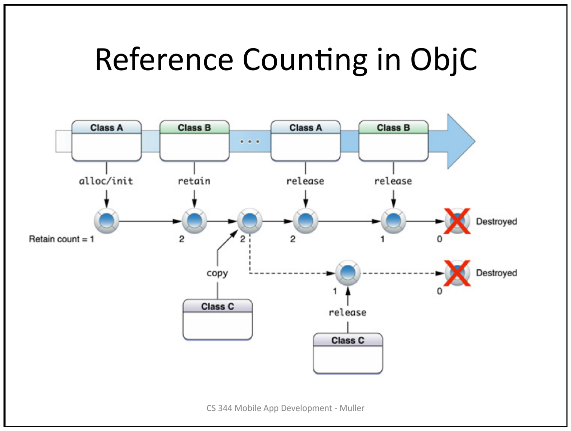


In many languages, (e.g., LISP, CAML, Java, Python, …) memory in the heap is allocated and reclaimed automatically using garbage collection.

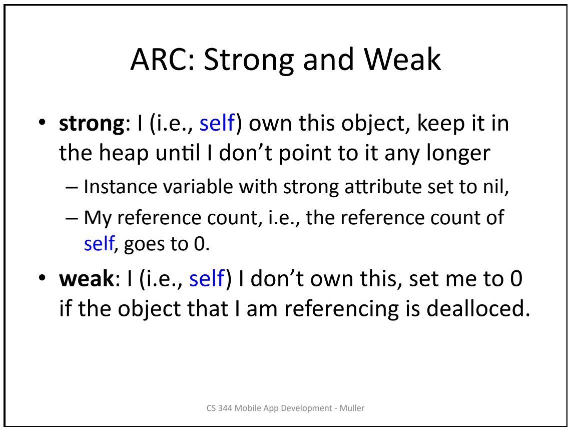


The iOS engineers at Apple decided that garbage collection consumed too much battery power.

So on iOS, heap allocation and deallocation are managed using reference counting.

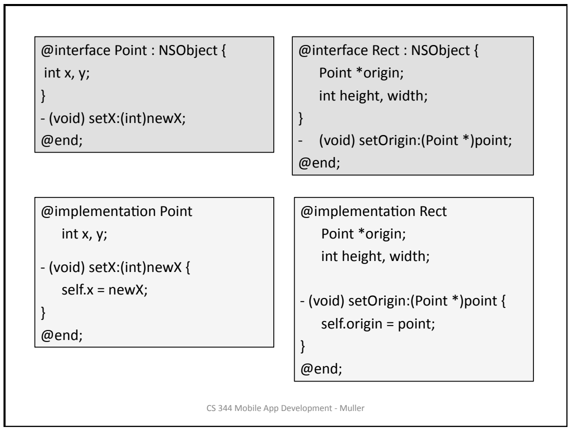


Reference Counting in ObjC



ARC: Strong and Weak

- **strong**: I (i.e., self) own this object, keep it in the heap until I don't point to it any longer
  - Instance variable with strong attribute set to nil,
  - My reference count, i.e., the reference count of self, goes to 0.
- **weak**: I (i.e., self) I don't own this, set me to 0 if the object that I am referencing is dealloced.

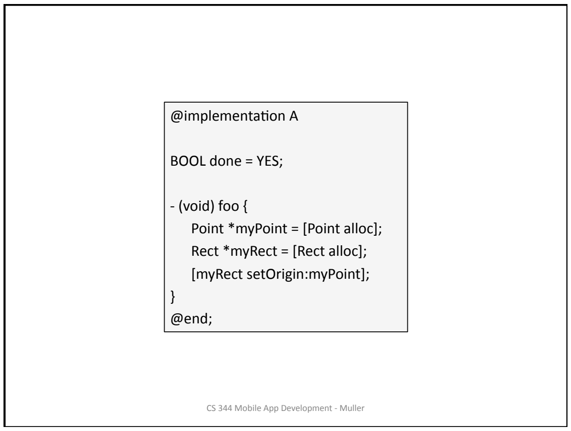


```
@interface Point : NSObject {
 int x, y;
}
- (void) setX:(int)newX;
@end;
```

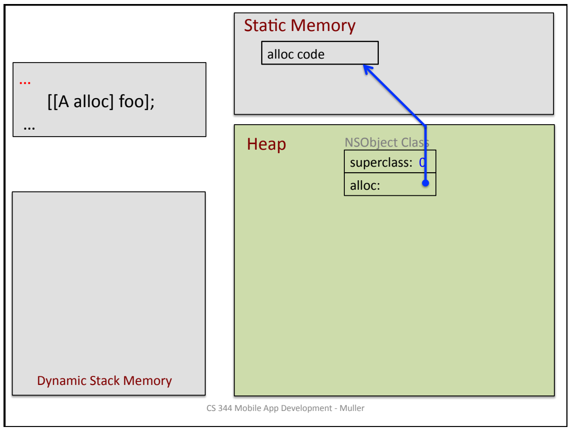
```
@interface Rect : NSObject {
    Point *origin;
    int height, width;
}
-    (void) setOrigin:(Point *)point;
@end;
```

```
@implementation Point
    int x, y;

- (void) setX:(int)newX {
    self.x = newX;
}
@end;
```

```
@implementation Rect
    Point *origin;
    int height, width;

- (void) setOrigin:(Point *)point {
    self.origin = point;
}
@end;
```



```
@implementation A

BOOL done = YES;

- (void) foo {
    Point *myPoint = [Point alloc];
    Rect *myRect = [Rect alloc];
    [myRect setOrigin:myPoint];
}
@end;
```



```
...
    [[A alloc] foo];
...
```

Static Memory

alloc code

Heap

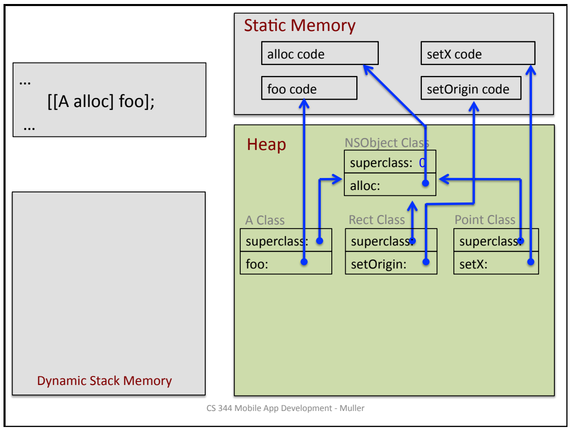
NSObject Class

superclass: 0

alloc:

Dynamic Stack Memory

CS 344 Mobile App Development - Muller



```
...
    [[A alloc] foo];
...
```

Static Memory

alloc code

foo code

setX code

setOrigin code

Heap

NSObject Class

superclass: 0

alloc:

A Class

superclass:

foo:

Rect Class

superclass:

setOrigin:

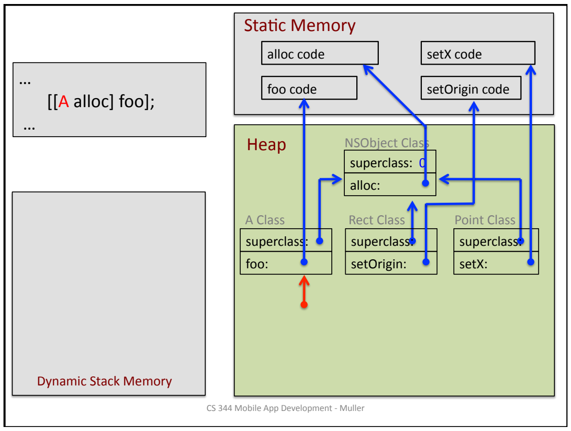
Point Class

superclass:

setX:

Dynamic Stack Memory

CS 344 Mobile App Development - Muller



```
...
    [[A alloc] foo];
...
```

Static Memory

alloc code

foo code

setX code

setOrigin code

Heap

NSObject Class

superclass: 0

alloc:

A Class

superclass:

foo:

Rect Class

superclass:

setOrigin:

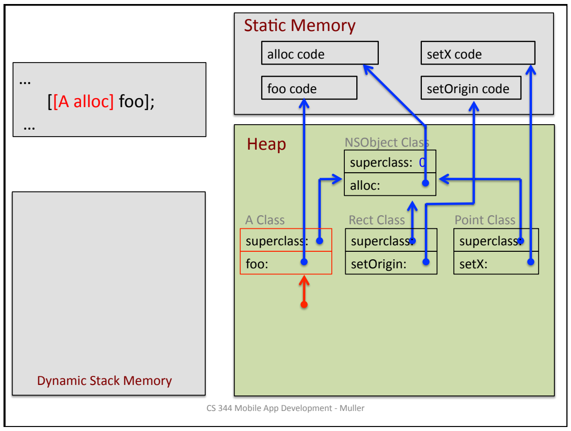
Point Class

superclass:

setX:

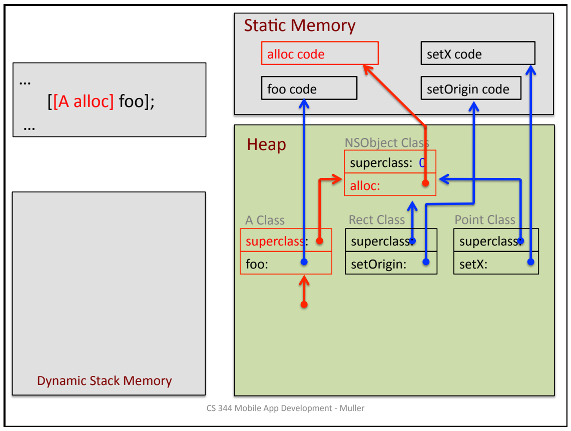
Dynamic Stack Memory

CS 344 Mobile App Development - Muller



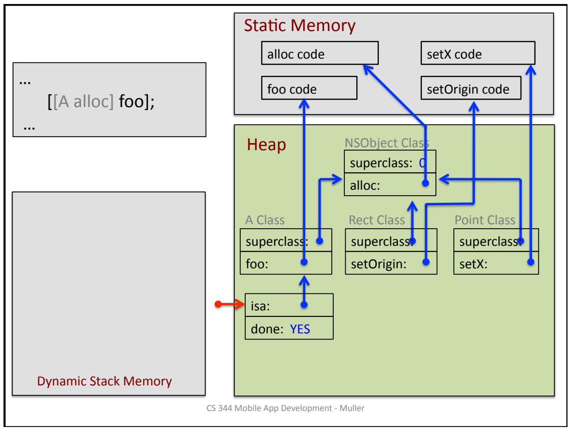
```
…
    [[A alloc] foo];
…
```

CS 344 Mobile App Development - Muller



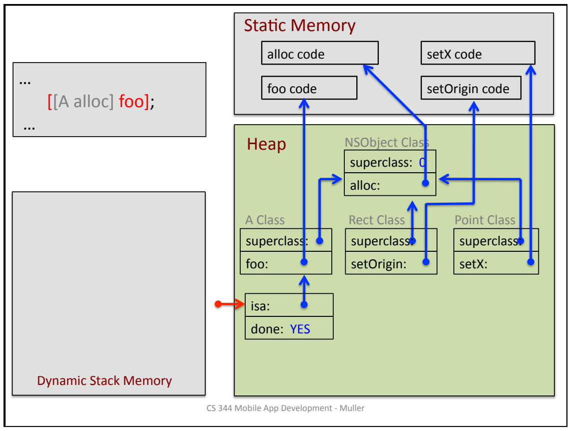
```
…
    [[A alloc] foo];
…
```

CS 344 Mobile App Development - Muller



```
…
    [[A alloc] foo];
…
```

CS 344 Mobile App Development - Muller



...

[[A alloc] foo];

...

Static Memory

alloc code

setX code

foo code

setOrigin code

Heap

NSObject Class

superclass: 0

alloc:

A Class

superclass:

foo:

Rect Class

superclass:

setOrigin:

Point Class

superclass:

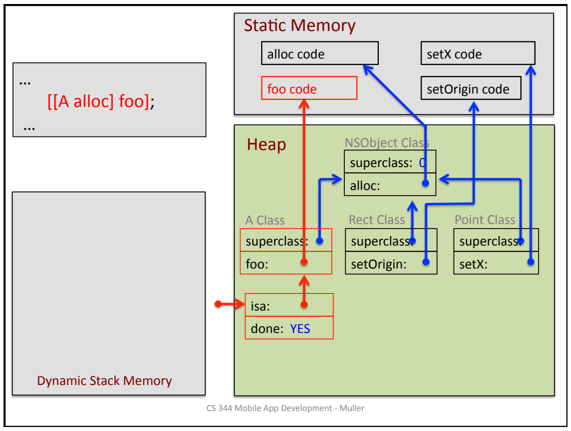
setX:

isa:

done: YES

Dynamic Stack Memory

CS 344 Mobile App Development - Muller



...

[[A alloc] foo];

...

Static Memory

alloc code

setX code

foo code

setOrigin code

Heap

NSObject Class

superclass: 0

alloc:

A Class

superclass:

foo:

Rect Class

superclass:

setOrigin:

Point Class

superclass:

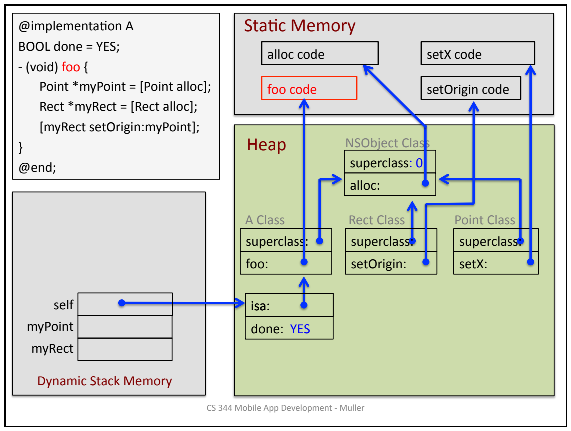
setX:

isa:

done: YES

Dynamic Stack Memory

CS 344 Mobile App Development - Muller



```
@implementation A
BOOL done = YES;
- (void) foo {
    Point *myPoint = [Point alloc];
    Rect *myRect = [Rect alloc];
    [myRect setOrigin:myPoint];
}
@end;
```

Static Memory

alloc code

setX code

foo code

setOrigin code

Heap

NSObject Class

superclass: 0

alloc:

A Class

superclass:

foo:

Rect Class

superclass:

setOrigin:

Point Class

superclass:

setX:

isa:

done: YES

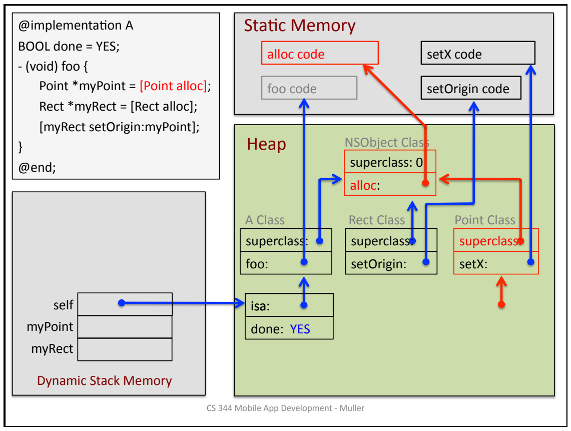
self

myPoint

myRect

Dynamic Stack Memory

CS 344 Mobile App Development - Muller



```
@implementation A
BOOL done = YES;
- (void) foo {
    Point *myPoint = [Point alloc];
    Rect *myRect = [Rect alloc];
    [myRect setOrigin:myPoint];
}
@end;
```

Static Memory

alloc code

setX code

foo code

setOrigin code

Heap

NSObject Class

superclass: 0

alloc:

A Class

superclass:

foo:

Rect Class

superclass:

setOrigin:

Point Class

superclass:

setX:

isa:

done: YES

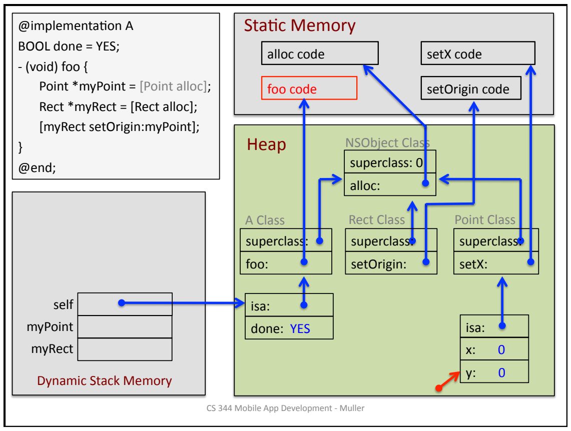
self

myPoint

myRect

Dynamic Stack Memory

CS 344 Mobile App Development - Muller



```
@implementation A
BOOL done = YES;
- (void) foo {
    Point *myPoint = [Point alloc];
    Rect *myRect = [Rect alloc];
    [myRect setOrigin:myPoint];
}
@end;
```

Static Memory

alloc code

setX code

foo code

setOrigin code

Heap

NSObject Class

superclass: 0

alloc:

A Class

superclass:

foo:

Rect Class

superclass:

setOrigin:

Point Class

superclass:

setX:

isa:

done: YES

isa:

x: 0

y: 0

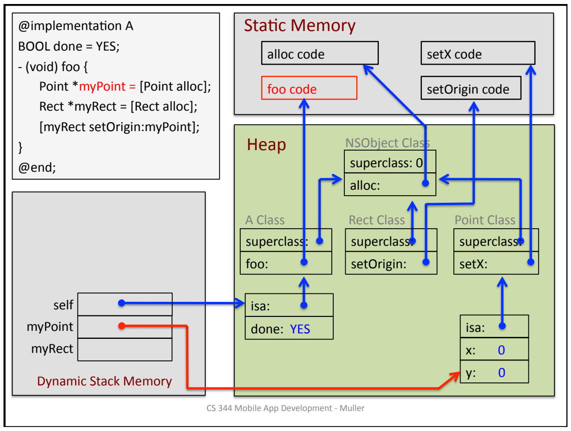
self

myPoint

myRect

Dynamic Stack Memory

CS 344 Mobile App Development - Muller



```
@implementation A
BOOL done = YES;
- (void) foo {
    Point *myPoint = [Point alloc];
    Rect *myRect = [Rect alloc];
    [myRect setOrigin:myPoint];
}
@end;
```

Static Memory

alloc code

setX code

foo code

setOrigin code

Heap

NSObject Class

superclass: 0

alloc:

A Class

superclass:

foo:

Rect Class

superclass:

setOrigin:

Point Class

superclass:

setX:

isa:

done: YES

isa:

x: 0

y: 0

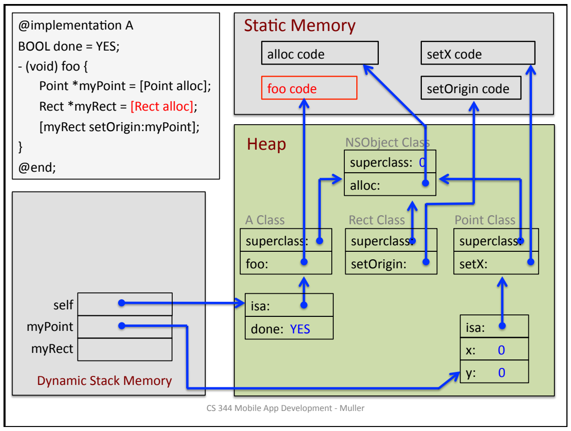
self

myPoint

myRect

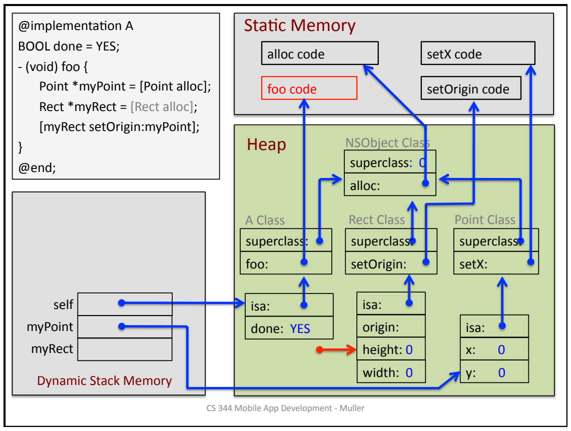
Dynamic Stack Memory

CS 344 Mobile App Development - Muller



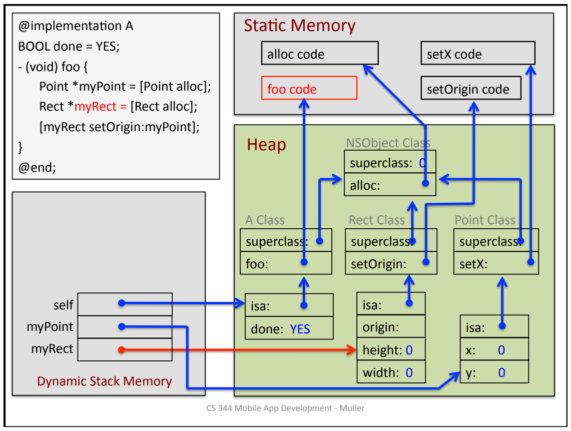
```
@implementation A
BOOL done = YES;
- (void) foo {
    Point *myPoint = [Point alloc];
    Rect *myRect = [Rect alloc];
    [myRect setOrigin:myPoint];
}
@end;
```

CS 344 Mobile App Development - Muller



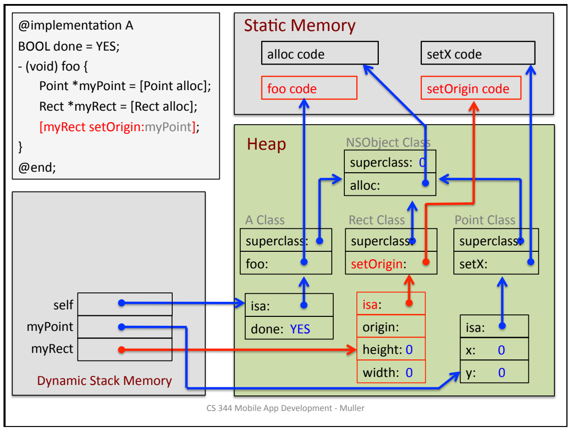
```
@implementation A
BOOL done = YES;
- (void) foo {
    Point *myPoint = [Point alloc];
    Rect *myRect = [Rect alloc];
    [myRect setOrigin:myPoint];
}
@end;
```

CS 344 Mobile App Development - Muller



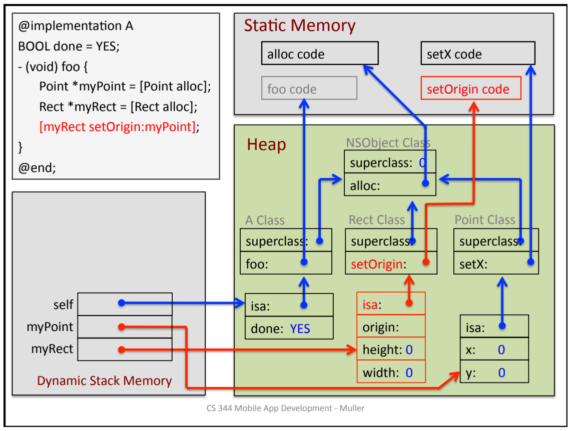
```
@implementation A
BOOL done = YES;
- (void) foo {
    Point *myPoint = [Point alloc];
    Rect *myRect = [Rect alloc];
    [myRect setOrigin:myPoint];
}
@end;
```

CS 344 Mobile App Development - Muller



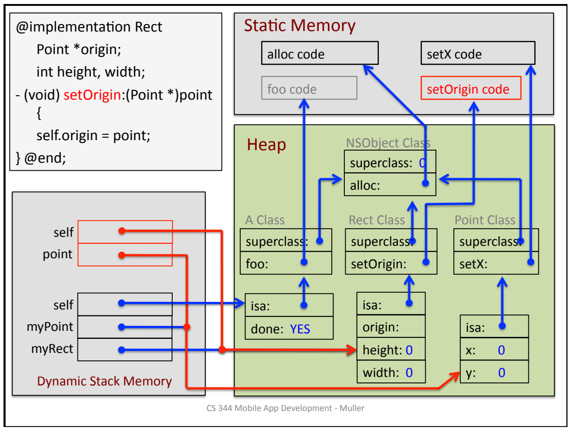
```
@implementation A
BOOL done = YES;
- (void) foo {
    Point *myPoint = [Point alloc];
    Rect *myRect = [Rect alloc];
    [myRect setOrigin:myPoint];
}
@end;
```

CS 344 Mobile App Development - Muller



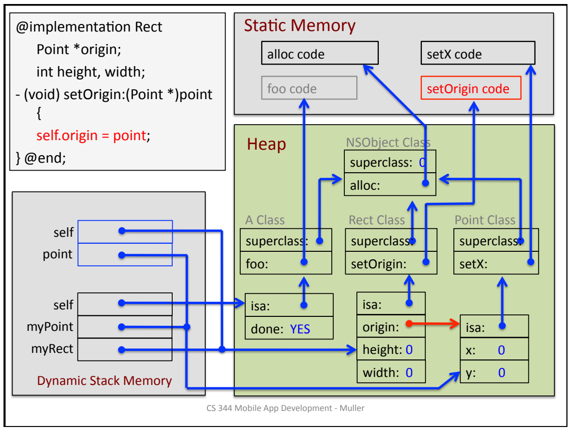
```
@implementation A
BOOL done = YES;
- (void) foo {
    Point *myPoint = [Point alloc];
    Rect *myRect = [Rect alloc];
    [myRect setOrigin:myPoint];
}
@end;
```

CS 344 Mobile App Development - Muller



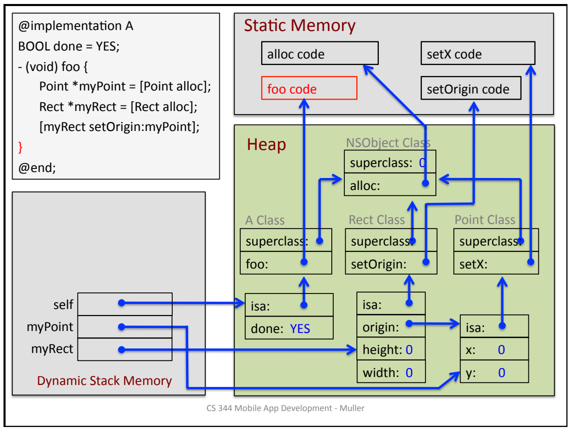
```
@implementation Rect
    Point *origin;
    int height, width;
- (void) setOrigin:(Point *)point
    {
    self.origin = point;
} @end;
```

CS 344 Mobile App Development - Muller



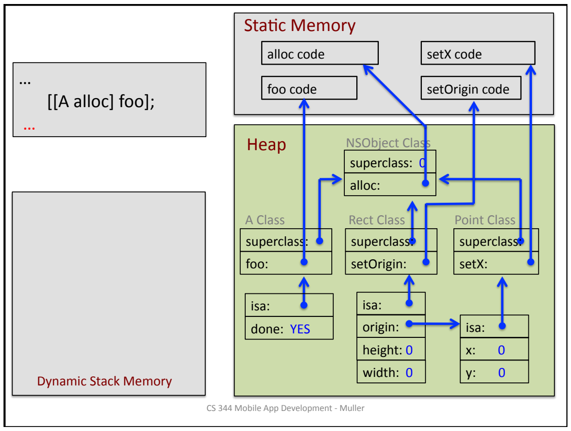
```
@implementation Rect
    Point *origin;
    int height, width;
- (void) setOrigin:(Point *)point
    {
    self.origin = point;
} @end;
```

CS 344 Mobile App Development - Muller



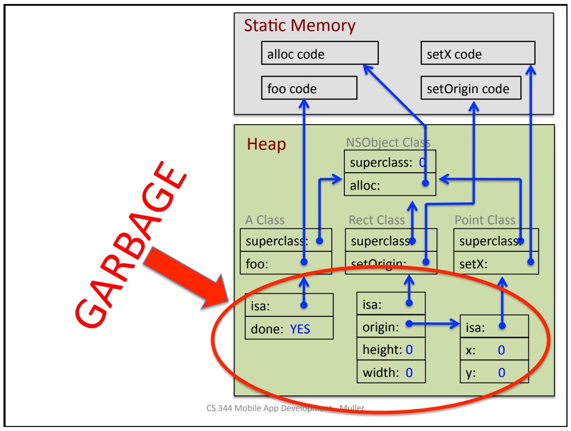
```
@implementation A
BOOL done = YES;
- (void) foo {
    Point *myPoint = [Point alloc];
    Rect *myRect = [Rect alloc];
    [myRect setOrigin:myPoint];
}
@end;
```

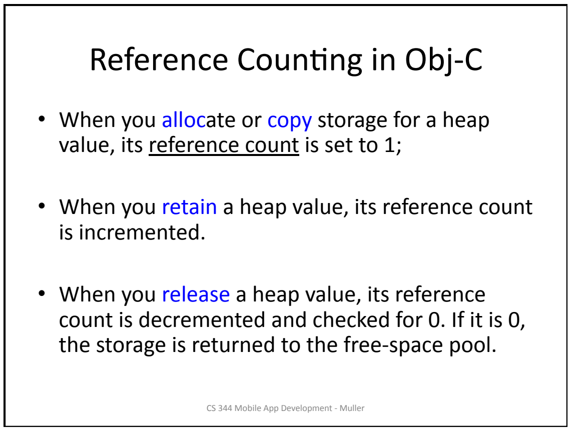
CS 344 Mobile App Development - Muller



```
...
    [[A alloc] foo];
...
```

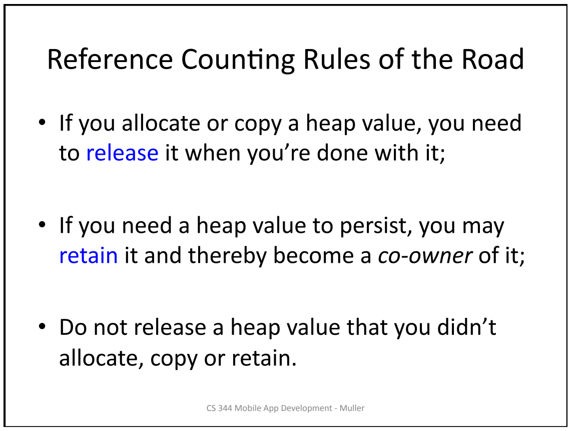
CS 344 Mobile App Development - Muller





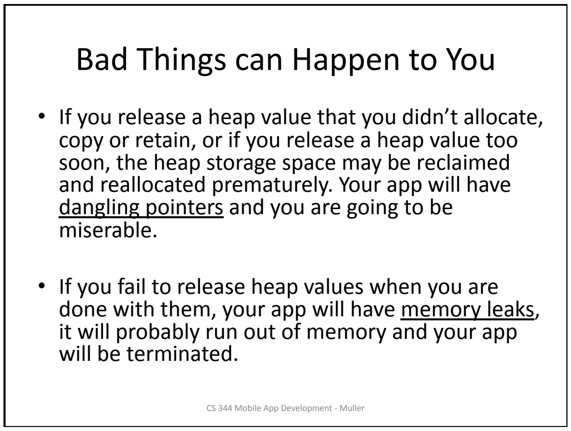
Reference Counting in Obj-C

- When you allocate or copy storage for a heap value, its reference count is set to 1;
- When you retain a heap value, its reference count is incremented.
- When you release a heap value, its reference count is decremented and checked for 0. If it is 0, the storage is returned to the free-space pool.



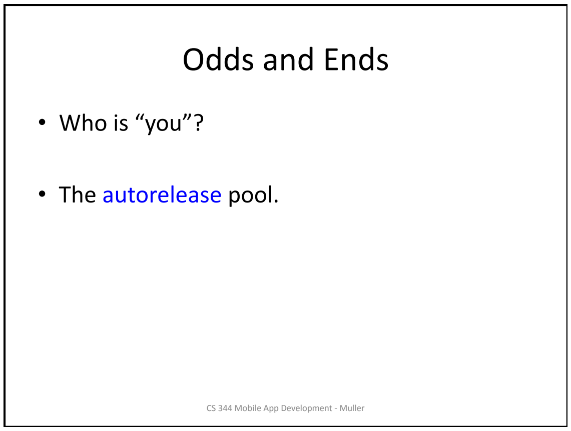
Reference Counting Rules of the Road

- If you allocate or copy a heap value, you need to release it when you're done with it;
- If you need a heap value to persist, you may retain it and thereby become a *co-owner* of it;
- Do not release a heap value that you didn't allocate, copy or retain.



Bad Things can Happen to You

- If you release a heap value that you didn't allocate, copy or retain, or if you release a heap value too soon, the heap storage space may be reclaimed and reallocated prematurely. Your app will have dangling pointers and you are going to be miserable.
- If you fail to release heap values when you are done with them, your app will have memory leaks, it will probably run out of memory and your app will be terminated.



Odds and Ends

- Who is "you"?
- The autorelease pool.