Object Representation in Objective-C

CS 344 Mobile App Development

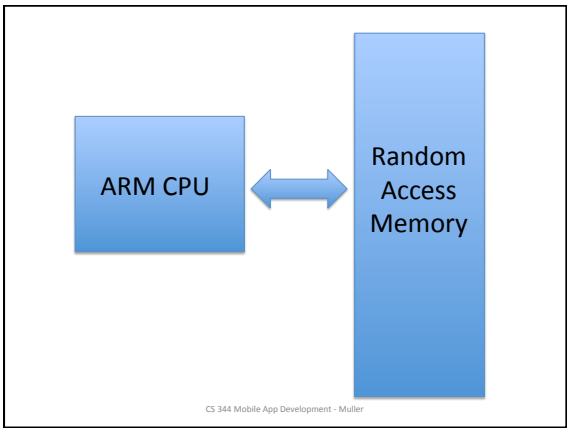
Robert Muller

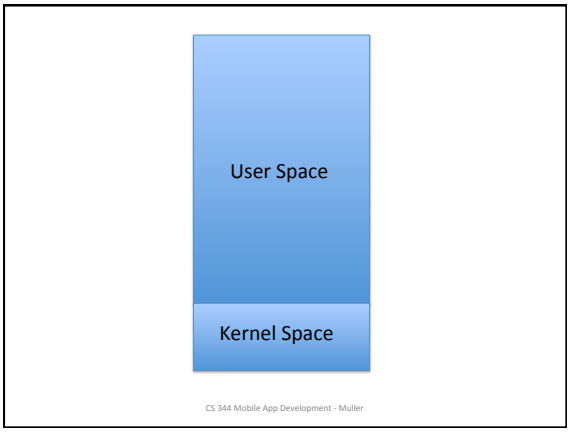
Lazy Allocation & Initialization in the RPN Calculator

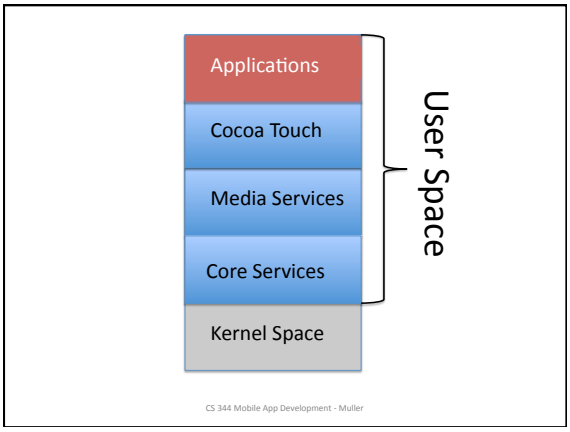
```
@implementation Model

// Allocate the stack the first time it is retrieved by a call of the getter.
//
 - (NSMutableArray *) operandStack {

    if(_operandStack == nil) _operandStack =
                                    [[NSMutableArray alloc] init];
    return _operandStack;
}
```


Eager Allocation & Initialization

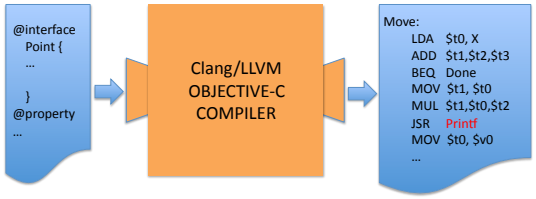
```
@implementation Point
 - (id) init
{
    self = [super init];
    if (self) {
    // initialize the Point class here
    }
    return self;
}
@end

…
Point *p = [[Point alloc] init];
```


ARM CPU
Random Access Memory
CS 344 Mobile App Development - Muller


User Space
Kernel Space
CS 344 Mobile App Development - Muller

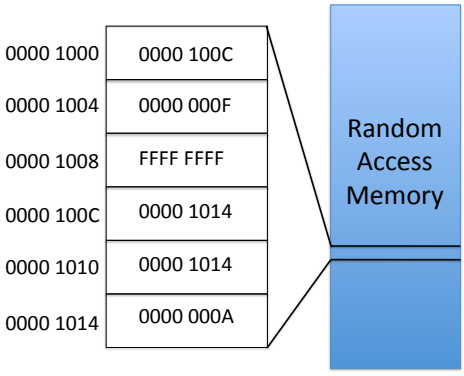

Applications
Cocoa Touch
Media Services
Core Services
Kernel Space
User Space
CS 344 Mobile App Development - Muller

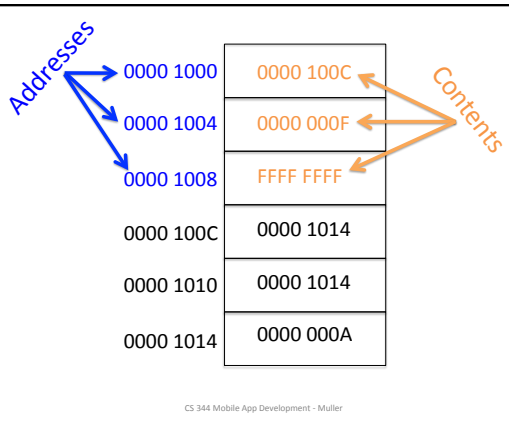
Xcode Objective-C Compiler Clang/LLVM



Xcode Compiler

Pointers


0000 1000
0000 100C
0000 1004
0000 000F
0000 1008
FFFF FFFF
0000 100C
0000 1014
0000 1010
0000 1014
0000 1014
0000 000A
Random Access Memory


Addresses
0000 1000
0000 1004
0000 1008
0000 100C
0000 1010
0000 1014
0000 100C
0000 000F
FFFF FFFF
0000 1014
0000 1014
0000 000A
Contents

Note that the value in location 0000 100C Can be interpreted as an address.

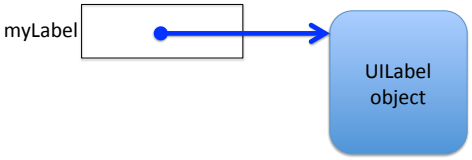
	0000 100C
	0000 000F
	FFFF FFFF
0000 100C	0000 1014
0000 1010	0000 1014
0000 1014	0000 000A

When storage cells contain addresses we'll usually draw arrows showing what they refer to.

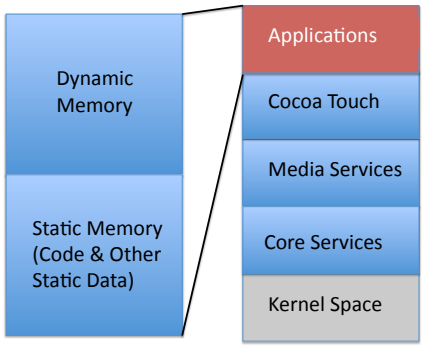
	0000 100C
	0000 000F
	FFFF FFFF
0000 100C	
0000 1010	0000 1014
0000 1014	0000 000A

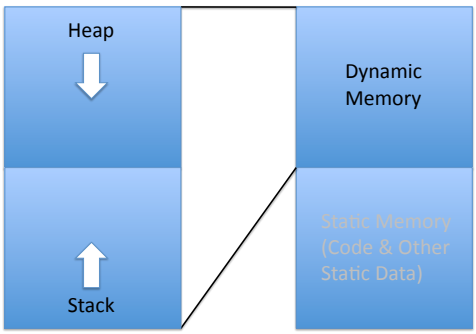
Example

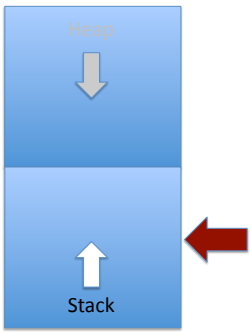
- UILabel *myLabel = [[UILabel alloc] init];



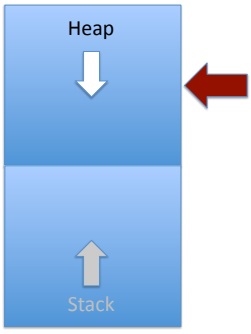
When an App is Loaded into RAM from the Flash Drive





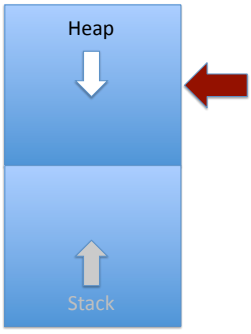


Values with dynamic extent. I.e., values whose lifetimes are bounded by the invocation of a procedure or method.



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.

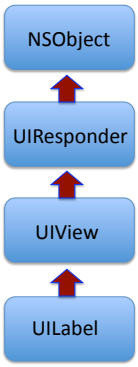


Objects are always allocated in the heap.

Object Representation

Representations of Objects in Java and Obj-C

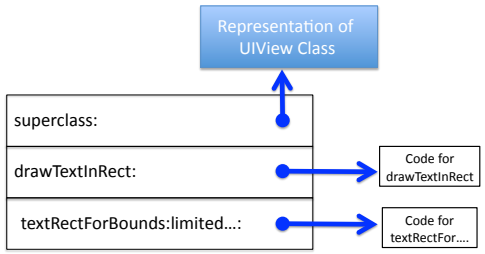
= Subclass



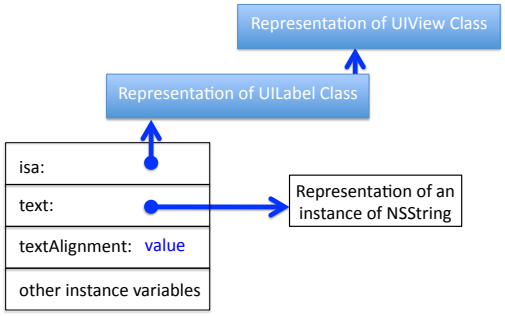
UILabel, E.g.,

```
@interface UILabel : IUView {
    NSString *text;
    UITextAlignment textAlignment;   // NB: No *!
    …
}
@property(nonatomic, copy) NSString *text;
@property(nonatomic) UITextAlignment textAlignment;
…
-- (void) drawTextInRect:(CGRect)rect;
-- (CGRect) textRectForBounds:(CGRect)bounds
                limitedToNumberOfLines(NSInteger)numberOfLines;
…
@end;
```

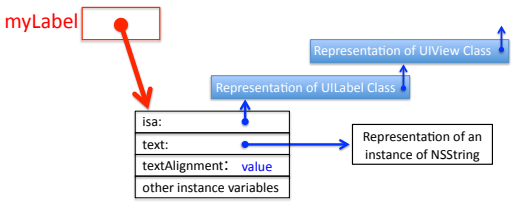
Run-Time Representation of the UILabel Class



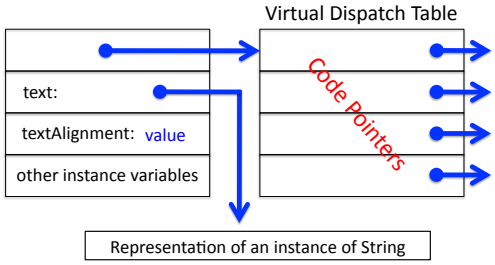
Representation of an Instance of UILabel

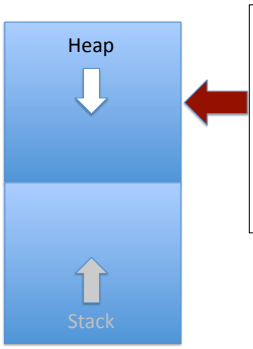


UILabel *myLabel = [[UILabel alloc] init];

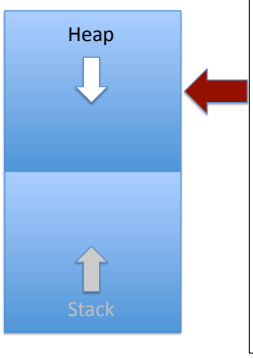


Object Representation in Java



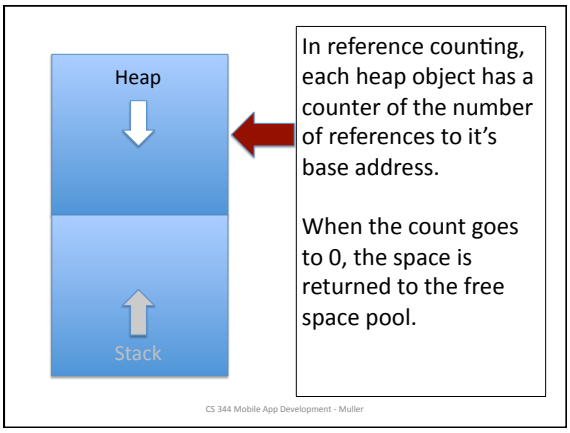


In many languages, (e.g., LISP, CAML, Java, Python, …) memory in the heap is allocated and reclaimed automatically using garbage collection.



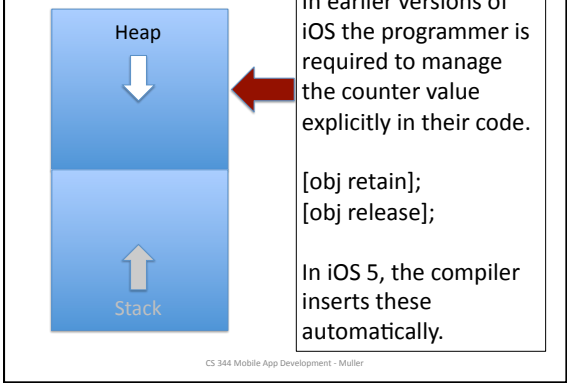
The engineers at Apple decided that garbage collection consumed too much battery power. And they didn't like the unpredictability.

So iOS manages memory using *reference counting.*



In reference counting, each heap object has a counter of the number of references to it's base address.

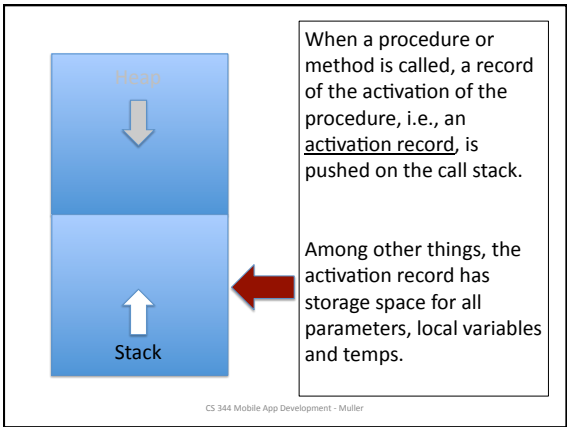
When the count goes to 0, the space is returned to the free space pool.



In earlier versions of iOS the programmer is required to manage the counter value explicitly in their code.

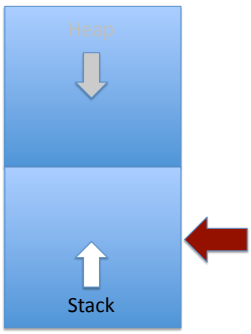
[obj retain];
[obj release];

In iOS 5, the compiler inserts these automatically.



When a procedure or method is called, a record of the activation of the procedure, i.e., an activation record, is pushed on the call stack.

Among other things, the activation record has storage space for all parameters, local variables and temps.



When the procedure or method exits, the activation record is popped from the stack and the storage is thereby automatically reclaimed and can be reused.

Activation Records for Functions

```
void doSomething(int x, int y) {
   double z;
...
}
```

AR pushed on the stack when doSomething is called.

y:
x:
other bookkeeping
z:

Activation Records for Methods

```
- (void) doSomething: (int) x  and: (int) y {
   double z;
...
}
```

sel:
y:
x:
self: → Repr. of host object (in heap).
other bookkeeping
z:

objc_msgSend

- Objective C compiler converts:

x = [receiver method];

To

x = objc_msgSend(receiver, selector, arg_1, arg_2,…);