Memory Management in iOS & Android

Santa Clara University, Operating System Fall 2017, Farokh Eskafi

Presenter: Xi (Lucas) Huang

Memory Management

Mobile vs Desktop

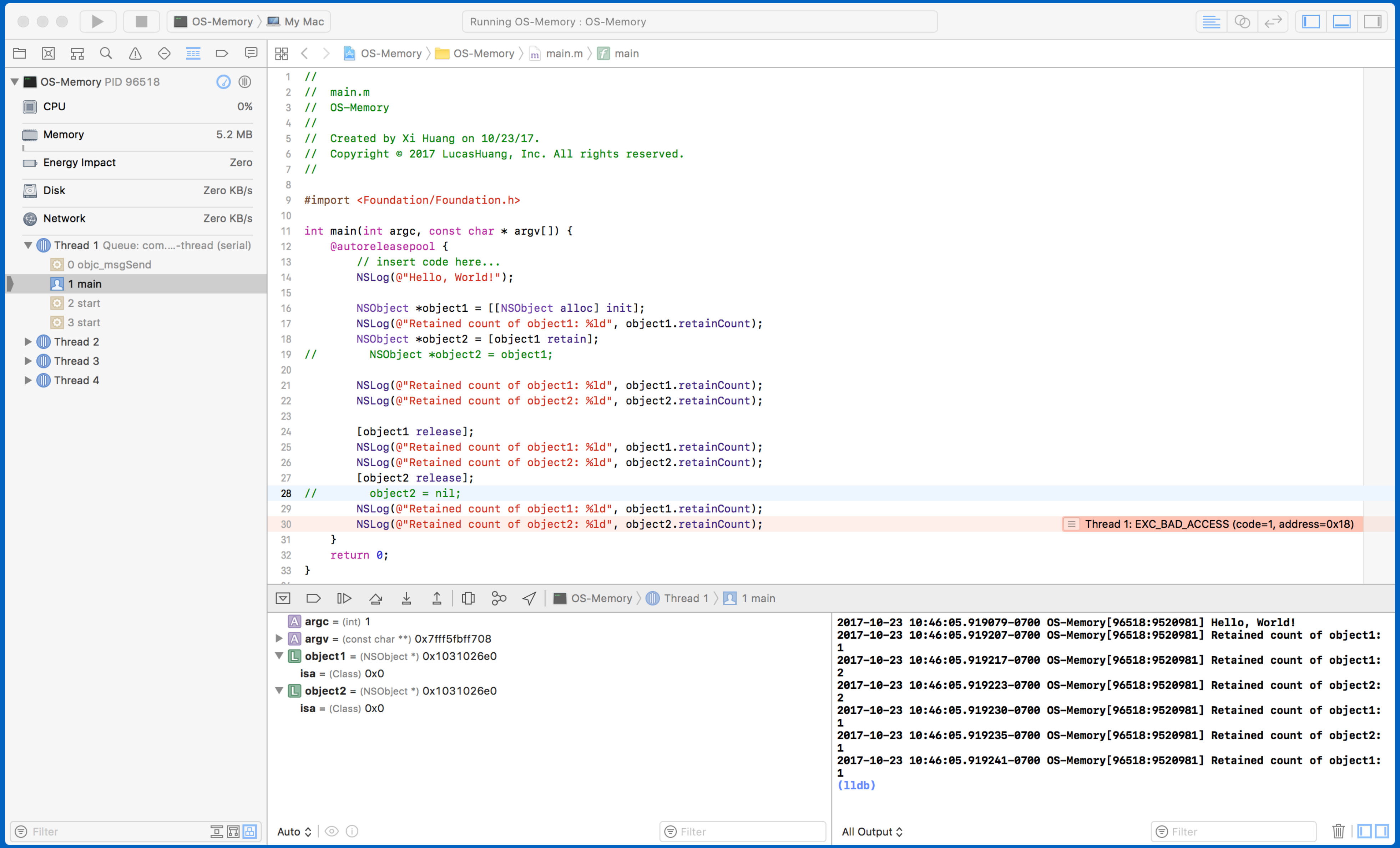
Battery life as one of the most critical concerns -> Limited RAM

iOS - Manual Retain Release (MRR)

You explicitly manage memory by keeping track of objects you own

- You own any object you create
- You can take ownership of an object using retain
- When you no longer need it, you must relinquish ownership of an object you own.
- You must not relinquish ownership of an object you do not own

iOS - An example of MRR


OS-Memory
My Mac
Running OS-Memory : OS-Memory
OS-Memory PID 96518
CPU 0%
Memory 5.2 MB
Energy Impact Zero
Disk Zero KB/s
Network Zero KB/s
Thread 1 Queue: com....-thread (serial)
0 objc_msgSend
1 main
2 start
3 start
Thread 2
Thread 3
Thread 4
OS-Memory 〉 OS-Memory 〉 main.m 〉 main
Thread 1: EXC_BAD_ACCESS (code=1, address=0x18)
OS-Memory 〉 Thread 1 〉 1 main
Auto
All Output
Filter

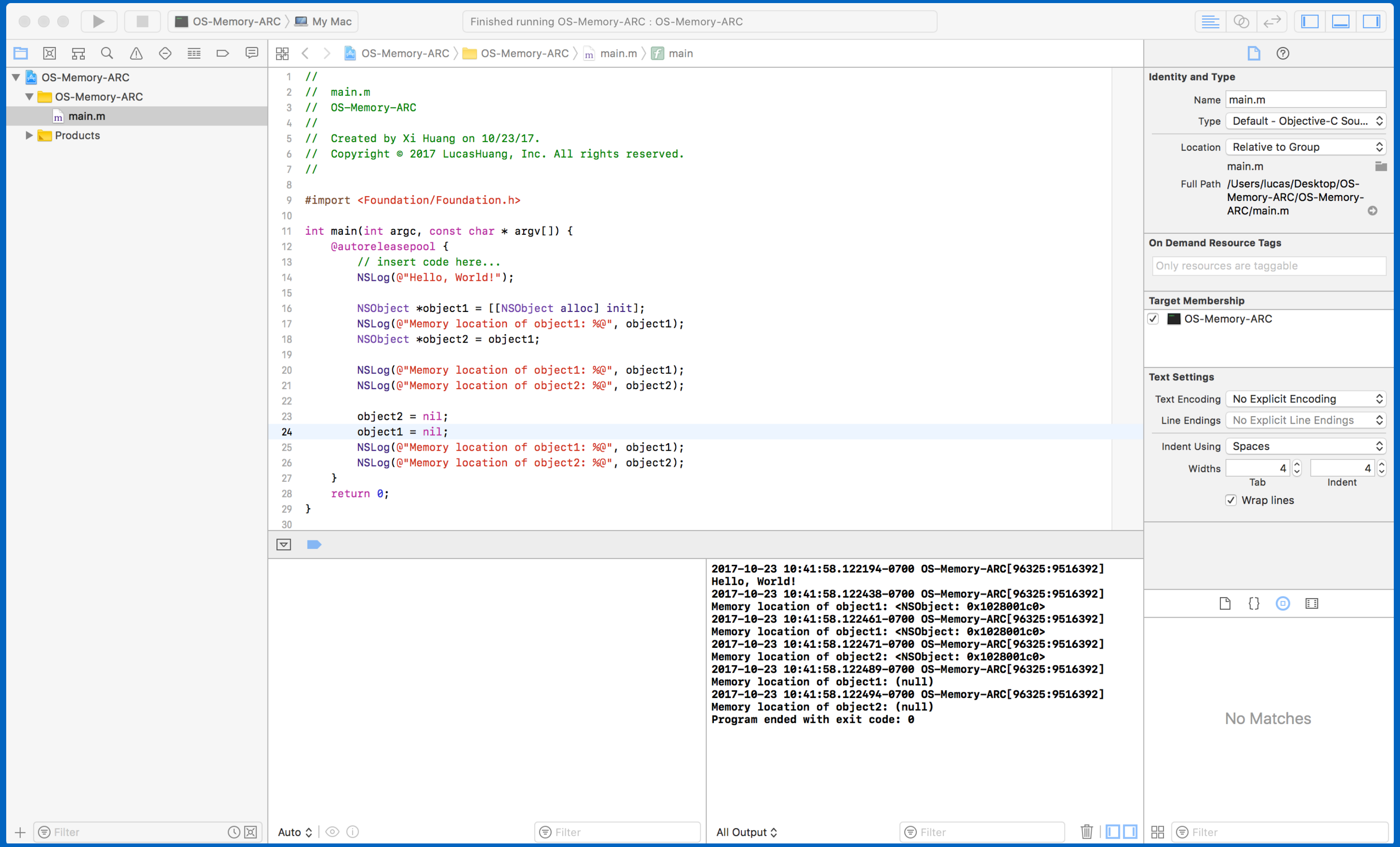
```
//
//  main.m
//  OS-Memory
//
//  Created by Xi Huang on 10/23/17.
//  Copyright © 2017 LucasHuang, Inc. All rights reserved.
//

#import <Foundation/Foundation.h>

int main(int argc, const char * argv[]) {
    @autoreleasepool {
        // insert code here...
        NSLog(@"Hello, World!");

        NSObject *object1 = [[NSObject alloc] init];
        NSLog(@"Retained count of object1: %ld", object1.retainCount);
        NSObject *object2 = [object1 retain];
//        NSObject *object2 = object1;

        NSLog(@"Retained count of object1: %ld", object1.retainCount);
        NSLog(@"Retained count of object2: %ld", object2.retainCount);

        [object1 release];
        NSLog(@"Retained count of object1: %ld", object1.retainCount);
        NSLog(@"Retained count of object2: %ld", object2.retainCount);
        [object2 release];
//        object2 = nil;
        NSLog(@"Retained count of object1: %ld", object1.retainCount);
        NSLog(@"Retained count of object2: %ld", object2.retainCount);
    }
    return 0;
}
```

```
argc = (int) 1
argv = (const char **) 0x7fff5fbff708
object1 = (NSObject *) 0x1031026e0
  isa = (Class) 0x0
object2 = (NSObject *) 0x1031026e0
  isa = (Class) 0x0
```

```
2017-10-23 10:46:05.919079-0700 OS-Memory[96518:9520981] Hello, World!
2017-10-23 10:46:05.919207-0700 OS-Memory[96518:9520981] Retained count of object1:
1
2017-10-23 10:46:05.919217-0700 OS-Memory[96518:9520981] Retained count of object1:
2
2017-10-23 10:46:05.919223-0700 OS-Memory[96518:9520981] Retained count of object2:
2
2017-10-23 10:46:05.919230-0700 OS-Memory[96518:9520981] Retained count of object1:
1
2017-10-23 10:46:05.919235-0700 OS-Memory[96518:9520981] Retained count of object2:
1
2017-10-23 10:46:05.919241-0700 OS-Memory[96518:9520981] Retained count of object1:
1
(lldb)
```

iOS - Automatic References Counting (ARC)

In Automatic Reference Counting, or ARC, the system uses the same reference counting system as MRR, but it inserts the appropriate memory management method calls for you at compile-time.

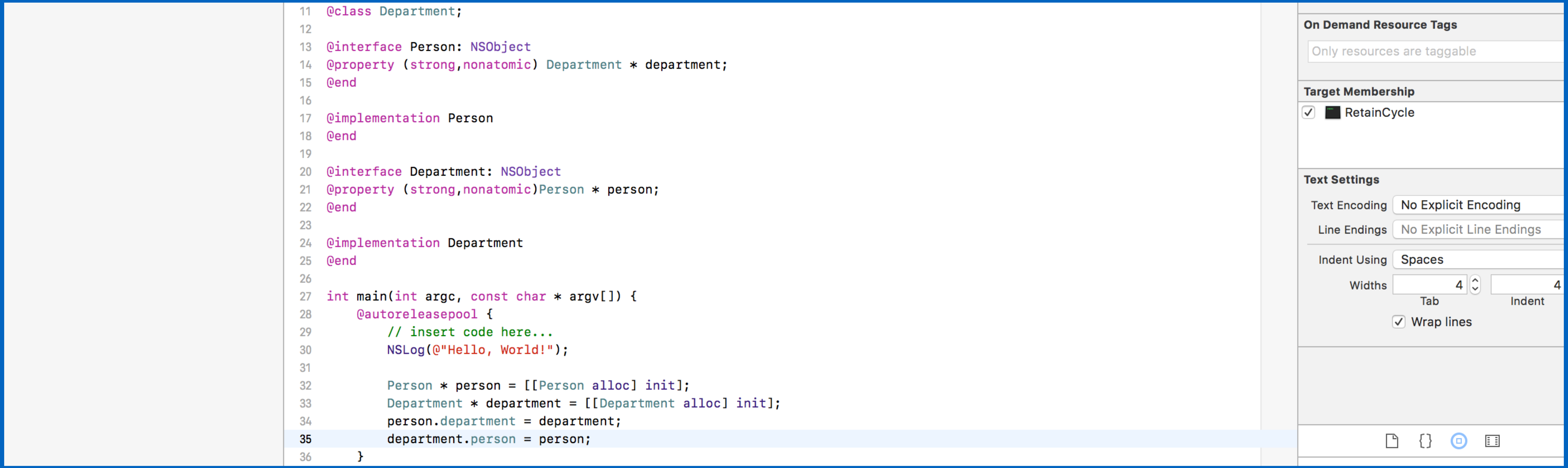


```objc
//
//  main.m
//  OS-Memory-ARC
//
//  Created by Xi Huang on 10/23/17.
//  Copyright © 2017 LucasHuang, Inc. All rights reserved.
//

#import <Foundation/Foundation.h>

int main(int argc, const char * argv[]) {
    @autoreleasepool {
        // insert code here...
        NSLog(@"Hello, World!");

        NSObject *object1 = [[NSObject alloc] init];
        NSLog(@"Memory location of object1: %@", object1);
        NSObject *object2 = object1;

        NSLog(@"Memory location of object1: %@", object1);
        NSLog(@"Memory location of object2: %@", object2);

        object2 = nil;
        object1 = nil;
        NSLog(@"Memory location of object1: %@", object1);
        NSLog(@"Memory location of object2: %@", object2);
    }
    return 0;
}
```

```
2017-10-23 10:41:58.122194-0700 OS-Memory-ARC[96325:9516392]
Hello, World!
2017-10-23 10:41:58.122438-0700 OS-Memory-ARC[96325:9516392]
Memory location of object1: <NSObject: 0x1028001c0>
2017-10-23 10:41:58.122461-0700 OS-Memory-ARC[96325:9516392]
Memory location of object1: <NSObject: 0x1028001c0>
2017-10-23 10:41:58.122471-0700 OS-Memory-ARC[96325:9516392]
Memory location of object2: <NSObject: 0x1028001c0>
2017-10-23 10:41:58.122489-0700 OS-Memory-ARC[96325:9516392]
Memory location of object1: (null)
2017-10-23 10:41:58.122494-0700 OS-Memory-ARC[96325:9516392]
Memory location of object2: (null)
Program ended with exit code: 0
```

Retained Cycle

Retaining an object creates a strong reference to that object. An object cannot be deallocated until all of its strong references are released. A problem, known as a retain cycle, can therefore arise if two objects may have cyclical references—that is, they have a strong reference to each other (either directly, or through a chain of other objects each with a strong reference to the next leading back to the first).



```
@class Department;

@interface Person: NSObject
@property (strong,nonatomic) Department * department;
@end

@implementation Person
@end

@interface Department: NSObject
@property (strong,nonatomic)Person * person;
@end

@implementation Department
@end

int main(int argc, const char * argv[]) {
    @autoreleasepool {
        // insert code here...
        NSLog(@"Hello, World!");

        Person * person = [[Person alloc] init];
        Department * department = [[Department alloc] init];
        person.department = department;
        department.person = person;
    }
```

On Demand Resource Tags

Only resources are taggable

Target Membership

✓ RetainCycle

Text Settings

Text Encoding: No Explicit Encoding

Line Endings: No Explicit Line Endings

Indent Using: Spaces

Widths: 4 (Tab) 4 (Indent)

✓ Wrap lines

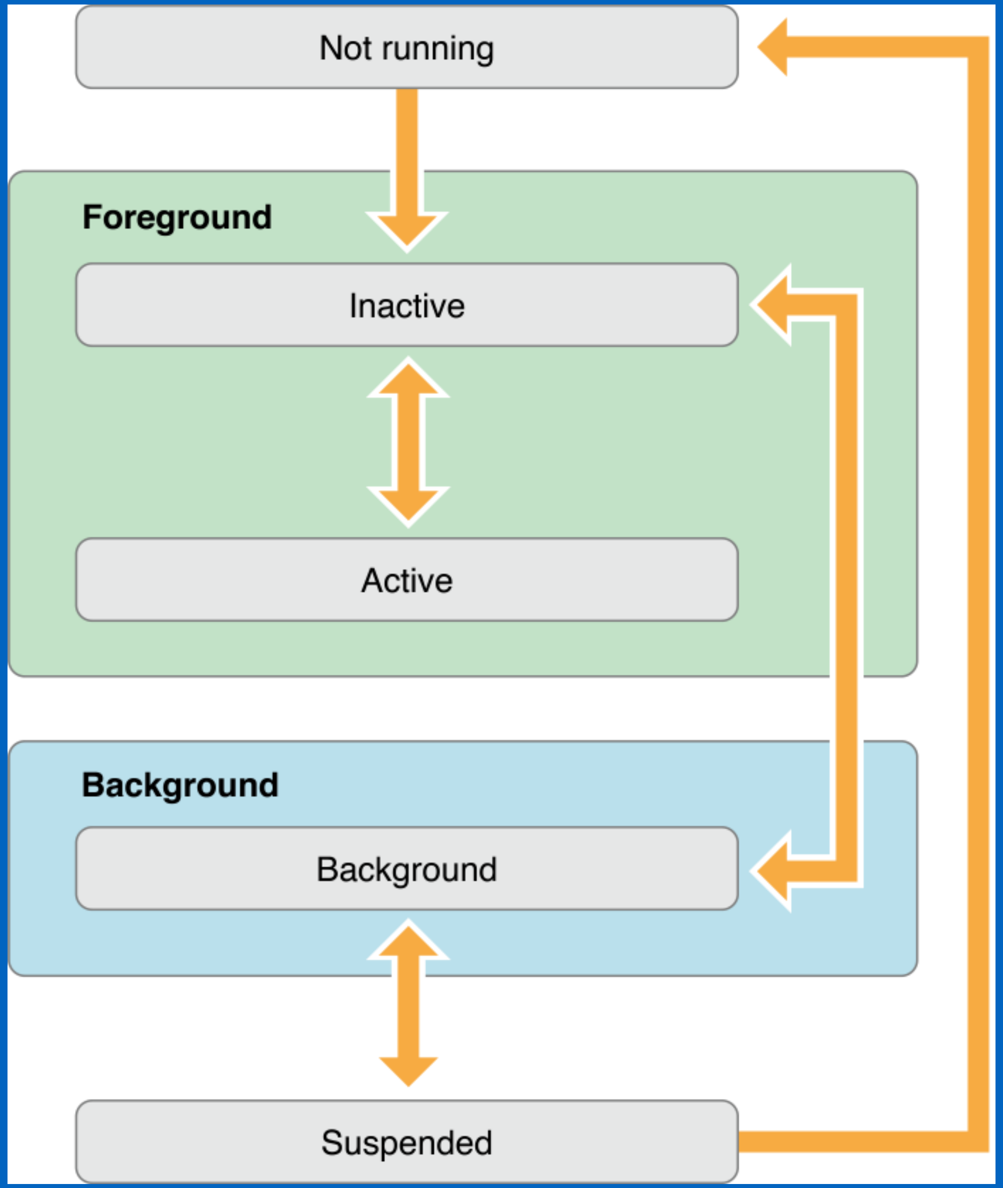
Overview of Android Memory Management

The Android Runtime (ART) and Dalvik virtual machine use **paging** and **memory-mapping (mmapping)** to manage memory. This means that any memory an app modifies—whether by allocating new objects or touching mmapped pages—remains resident in RAM and cannot be paged out. The only way to release memory from an app is to release object references that the app holds, making the memory available to the **garbage collector**.

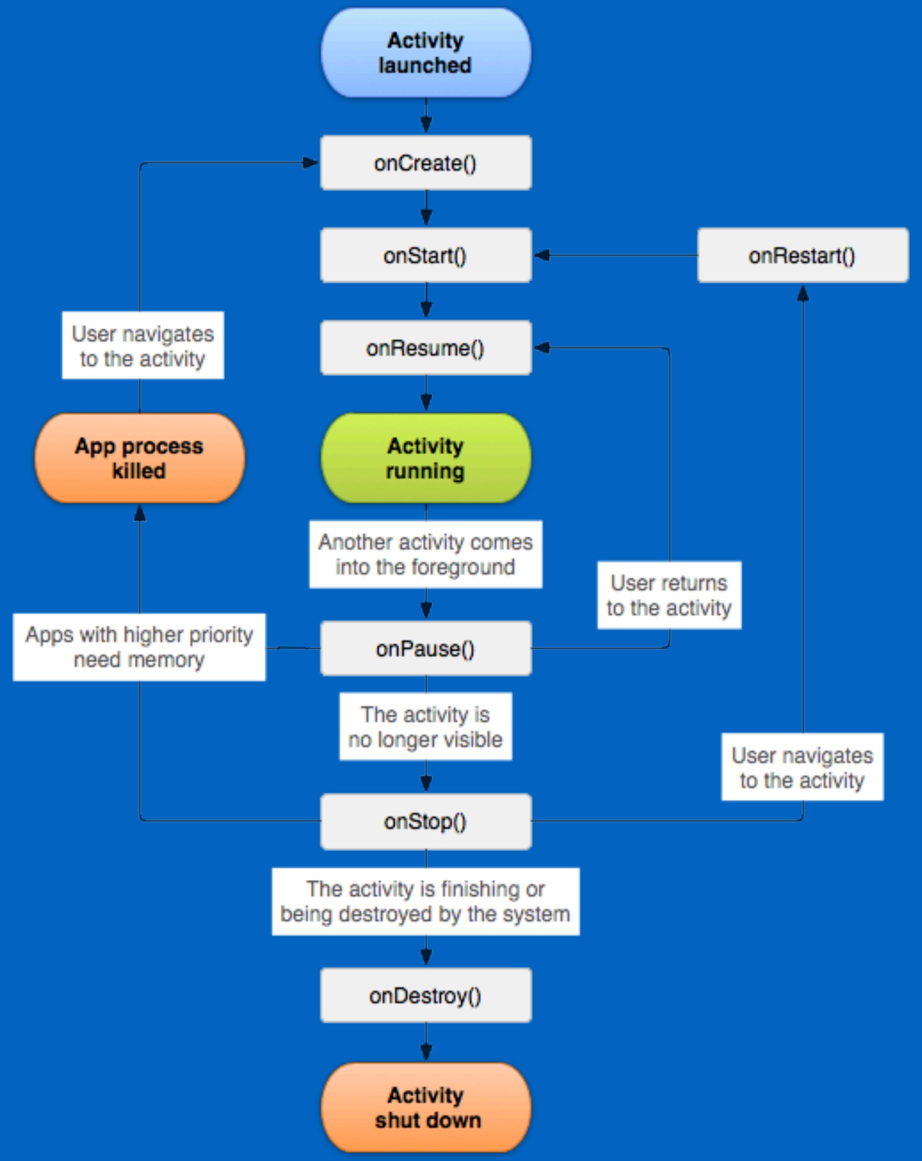
Garbage collection

A managed memory environment, like the ART or Dalvik virtual machine, keeps track of each memory allocation. Once it determines that a piece of memory is no longer being used by the program, it frees it back to the heap, without any intervention from the programmer.

iOS Application LifeCycle



Android Application LifeCycle


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

References and Sample Codes:

- Sample Codes: https://github.com/Lucashuang0802/283-2017-Fall-Presentation
- Apple Developer Guide: https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/MemoryMgmt/Articles/MemoryMgmt.html#//apple_ref/doc/uid/10000011-SW1
- Android Developer Guide: https://developer.android.com/topic/performance/memory-overview.html#gc