

Chapter 20

iOS Memory Management Guide

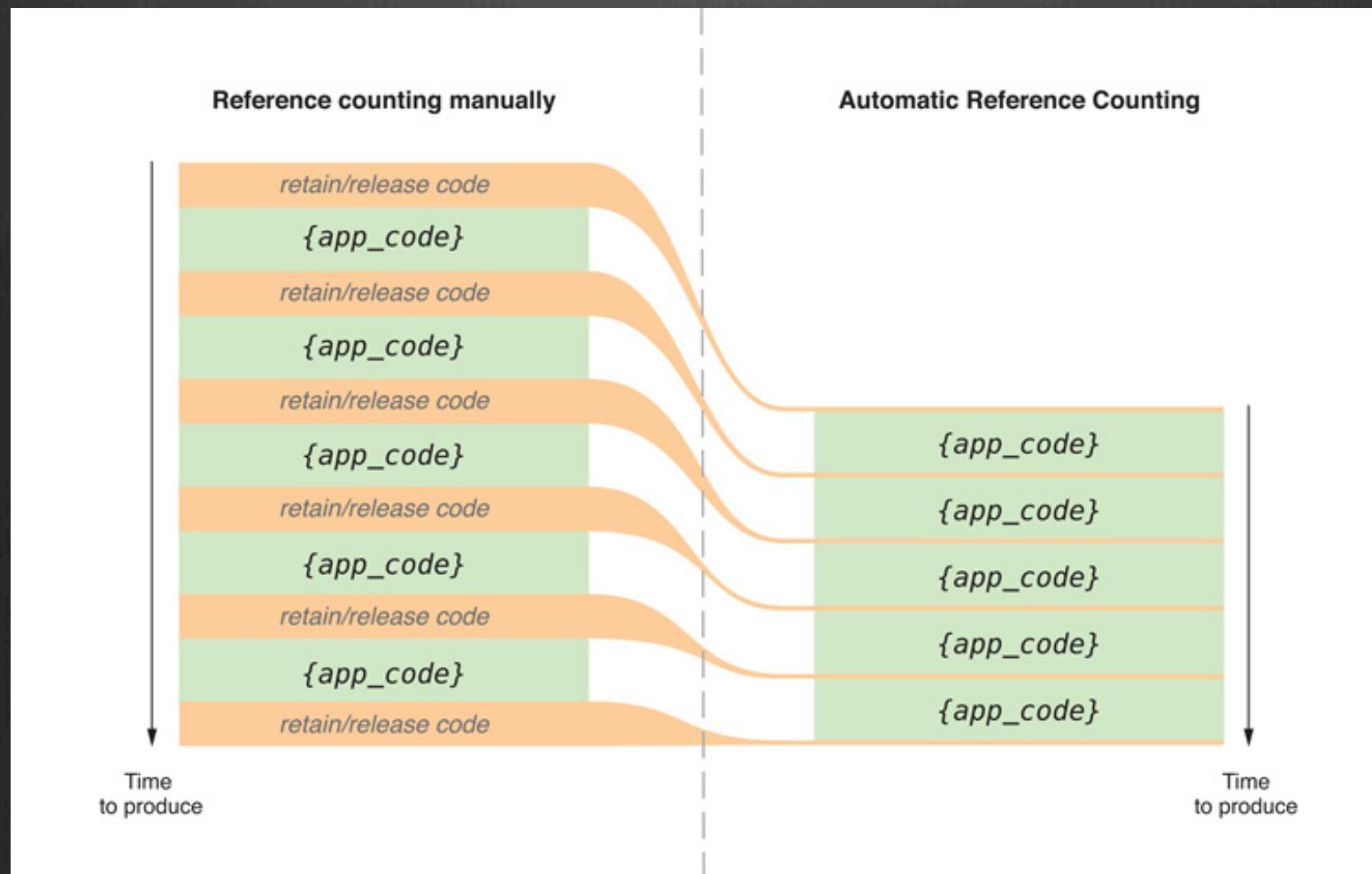
Topics

- ⦿ Behind the Scene - iOS ARC
- ⦿ alloc, release and retain in-deep
- ⦿ View-based memory management lifecycle
- ⦿ Tips for Improving Memory-Related Performance

Automatic Reference Counting (ARC)

- ◉ Objective-C เป็นภาษาที่ต่อ�อดมาจากการภาษา C การใช้ memory จึงเป็นแบบเดียวกัน คือต้องมีการ allocate และ release memory
- ◉ Garbage Collector ถูกเพิ่มเข้ามาภายหลังในการพัฒนา application บน OS X แต่ใน iOS ไม่มีระบบ Garbage Collector เพราะ overhead สูง และนักพัฒนาไม่นิยมใช้บน OS X
- ◉ ในการพัฒนา iOS App ก่อน version 4 โปรแกรมเมอร์จะต้องเป็นคนเขียน code เพื่อ release object เอง
- ◉ สาเหตุ #1 ที่ทำให้ App ไม่ผ่านการพิจารณาเมื่อขึ้น AppStore คือ memory leak และ app crash
- ◉ ตั้งแต่ iOS version 4 เป็นต้นมา Apple ได้พัฒนาเทคนิคใหม่ในการจัดการกับ Memory ที่มีประสิทธิภาพมากขึ้นและลดปัญหา error ที่เกิดจากการจัดการกับ memory ที่ไม่ดีลง เทคนิคนี้เรียกว่า Automatic Reference Counting หรือ ARC

ARC



Objective-C Without ARC

• Person.h

```
@interface Person : NSObject
{
    NSString * name;
}
- (void)setName:(NSString *)personName;
- (NSString *)getName;
@end
```

• Person.m

```
@implementation Person
- (void)setName:(NSString *)personName
{
    if (!personName) {
        name = [@"NONAME" retain];
    } else
        name = personName;
}
- (NSString *)getName
{
    if (name) {
        return name;
    } else
        return [@"NONAME" autorelease];
}
- (void)dealloc
{
    [name release];
}
@end
```

Objective-C Without ARC

⦿ Using Person Class

```
Person * p = [[Person alloc] init];
[p setName:@"Steve Jobs"];
NSString * name = [p getName];
[name retain]; // keep object to prevent de-allocate
[p release];
// use person's name ...
// if you forget to retain "name", app crash here...
[name release]; // release it manually
```

ARC with Property

- ⦿ Before ARC
 - ⦿ @property (nonatomic, **retain**) NSString * name;
- ⦿ With ARC
 - ⦿ @property (nonatomic, **strong**) NSString * name;
 - ⦿ @property (nonatomic, **weak**) NSString * employer;

Understand @property

- ⌚ What you see...

```
// MyClass.h

@interface MyClass : NSObject

@property (nonatomic, strong) NSString *myString;

@end
```

Understand @property

- What happened underneath...

```
// MyClass.h

@interface MyClass : NSObject
{
    __strong NSString * _myString;
}
@property (nonatomic, strong) NSString * myString;
@end

// MyClass.m

@implementation MyClass
@synthesize myString;

- (void)dealloc
{
    [myString release];
    myString = nil;
}
@end
```

Why do I care?

- ◉ แม้ว่า iDevice รุ่น ใหม่ๆ จะมี memory มากขึ้น แต่ก็ยังไม่มากพอที่จะเปิดให้นักพัฒนา ใช้ได้อย่างไม่จำกัดเหมือนกับการพัฒนาบน PC
- ◉ แม้ว่าจะมี ARC มาช่วยแล้ว ก็ไม่ได้หมายความว่า App จะใช้ memory น้อยลง
- ◉ ถ้า application ของเราจัดการ memory ได้ไม่ดี application ตัวนั้นจะไม่ผ่านการพิจารณาเมื่อขึ้น App Store และมีโอกาสที่ application จะ crash ได้ง่ายด้วย
- ◉ การเข้าใจกระบวนการจัดการ memory จึงเป็นเรื่องสำคัญที่นักพัฒนาจะต้องเรียนรู้ก่อนที่จะลงมือพัฒนา iOS Application จริง

Behind the scene: Retain Count

- ⦿ Object ใน Objective-C จะมี property ชื่อ retainCount เพื่อระบุว่า object นั้นมี ownership กี่ตัว
 - ⦿ เมื่อเราสร้าง object, retainCount จะมีค่าเท่ากับ 1
 - ⦿ เมื่อเราสั่ง retain ค่าของ retainCount จะเพิ่ม 1
 - ⦿ เมื่อเราสั่ง release ค่าของ retainCount จะลดลง 1
 - ⦿ เมื่อเราสั่ง autorelease ค่าของ retainCount จะลดลง 1 เมื่อ Auto release pool ทำงาน
 - ⦿ เมื่อ retainCount เท่ากับ 0 method dealloc จะถูกเรียก

Retain Count Example (1)

```
// MyClass.h

#import <Foundation/Foundation.h>

@interface MyClass : NSObject {

}

@end
```

```
// MyClass.m

#import "MyClass.h"

@implementation MyClass

- (void)dealloc
{
    NSLog(@"%@", @"MyClass's dealloc method called");
    [super dealloc];
}

@end
```

Retain Count Example (2)

```
#import <Foundation/Foundation.h>
#import "MyClass.h"

int main (int argc, const char * argv[]) {

    MyClass *m = [[MyClass alloc] init];
    NSLog(@"MyClass retainCount after alloc = %ld", [m retainCount]);

    [m retain];
    NSLog(@"MyClass retainCount after 1st retain = %ld", [m retainCount]);

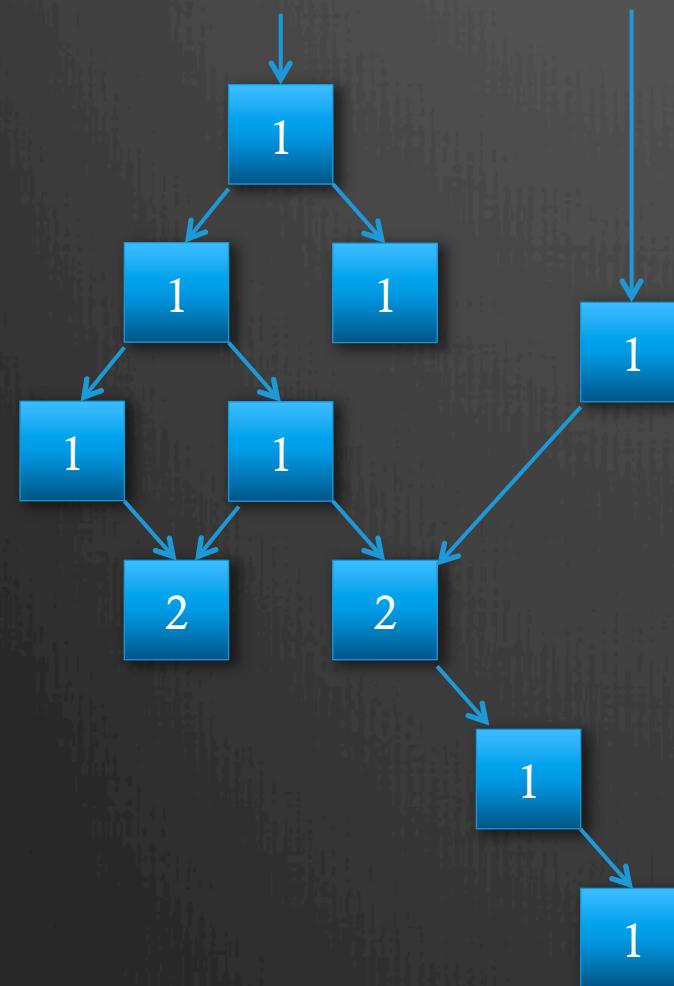
    [m release];
    NSLog(@"MyClass retainCount after 1nd release = %ld", [m retainCount]);

    [m release];
    return 0;
}
```

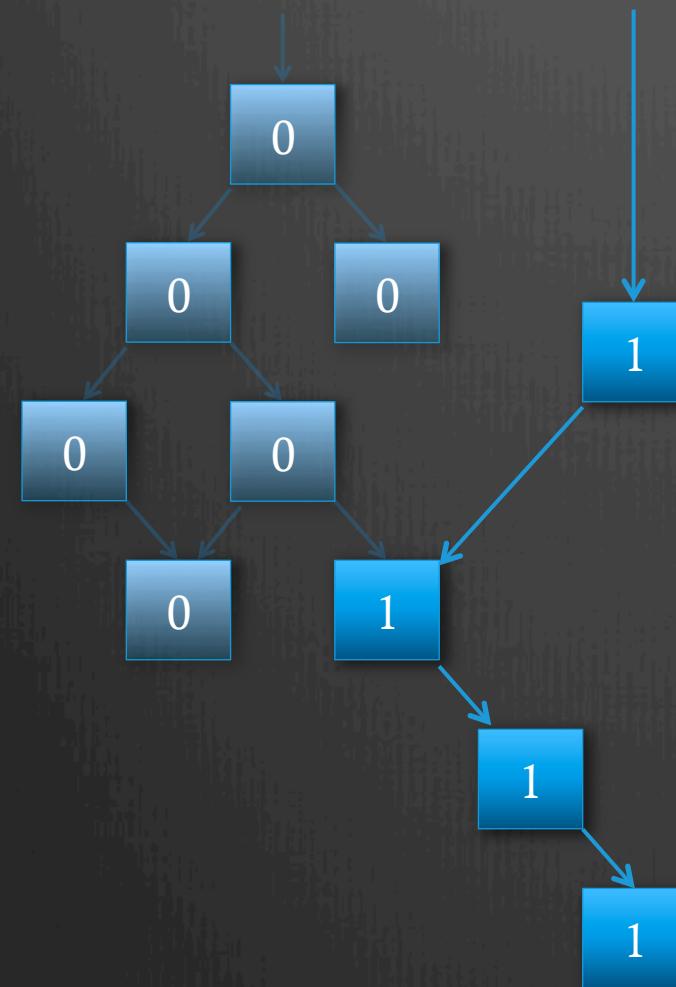
Result :

```
2011-05-07 23:05:16.648 MemorySample[5385:903] MyClass retainCount after alloc = 1
2011-05-07 23:05:16.652 MemorySample[5385:903] MyClass retainCount after 1st retain = 2
2011-05-07 23:05:16.653 MemorySample[5385:903] MyClass retainCount after 1nd release = 1
2011-05-07 23:05:16.654 MemorySample[5385:903] MyClass's dealloc method called
```

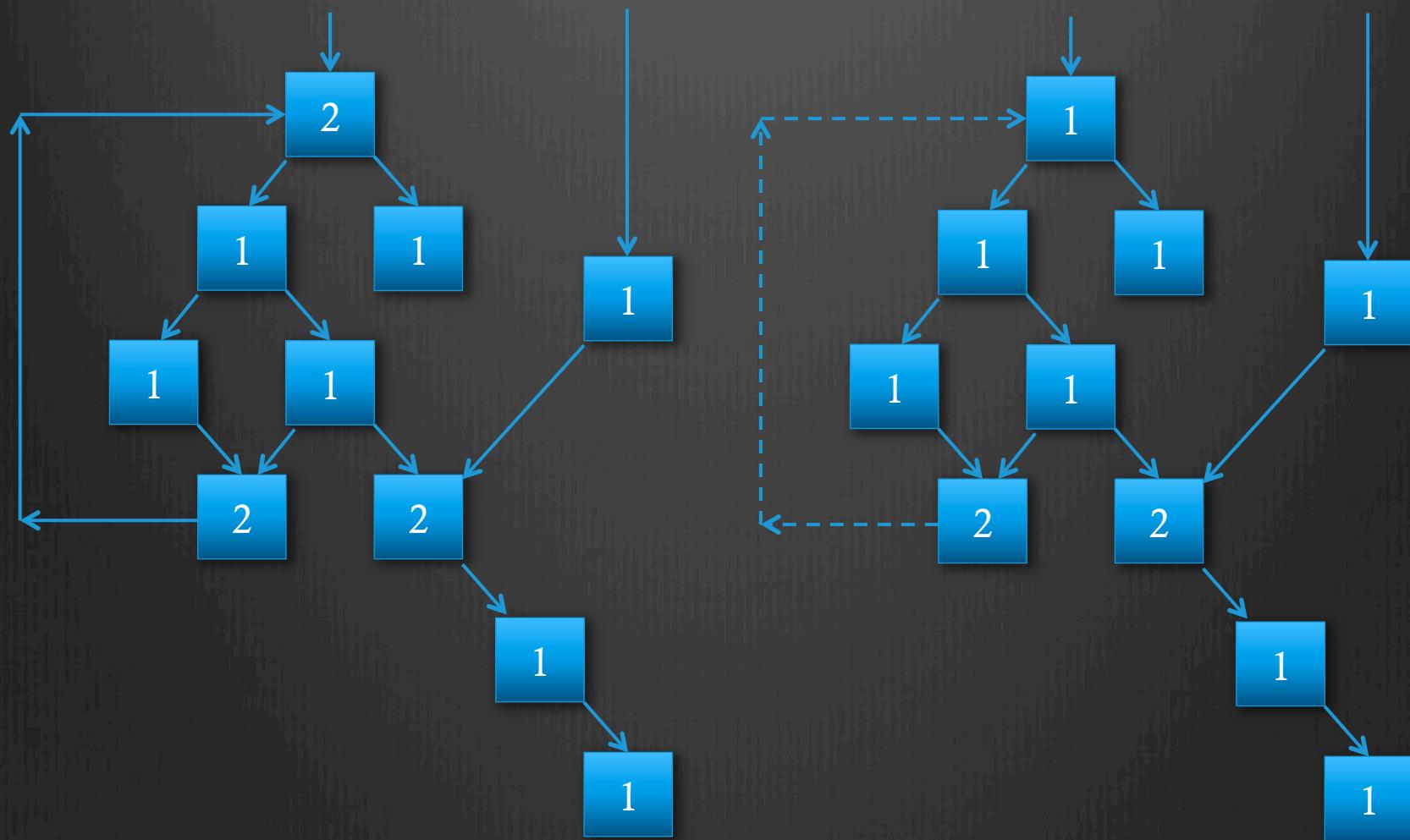
Directed Acyclic Graph



Directed Acyclic Graph



Cyclic Graph



Strong References to Objects

- การกำหนด retain ให้กับ object เรียกว่า เป็นการสร้าง “strong reference” ให้กับ object นั่นคือ class ใดที่มี property และกำหนด attribute ให้กับ property นั้นเป็น retain เมื่อ assign object ให้กับ property ของ class นั้น ค่า retainCount ของ object จะเพิ่มขึ้น 1

```
#import <Foundation/Foundation.h>
#import "MyClass.h"

@interface MyClass2 : NSObject {
@private
    MyClass *m;
}

@property(nonatomic, retain)MyClass *m;

@end
```

```
int main (int argc, const char * argv[]) {

    NSAutoreleasePool * pool =
        [[NSAutoreleasePool alloc] init];

    MyClass2 *m2 =
        [[[MyClass2 alloc] init] autorelease];

    MyClass *m =
        [[[MyClass alloc] init] autorelease];
    NSLog(@"Before associate. MyClass
        retainCount = %ld", [m retainCount]);

    m2.m = m;
    NSLog(@"After associate. MyClass
        retainCount = %ld", [m retainCount]);

    [pool drain];
    return 0;
}
```

Weak References to Objects

- ส่วนการสร้าง instant ไปชี้ที่ object จะไม่ใช้การ retain object แต่เป็นการ reference instant ไปยัง object เท่านั้น ค่า retainCount จะไม่เพิ่มขึ้น

```
int main (int argc, const char * argv[]) {  
  
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];  
  
    MyClass *m = [[[MyClass alloc] init] autorelease];  
    NSLog(@"%@", [m retainCount]);  
  
    MyClass *m2 = m;  
    NSLog(@"%@", [m2 retainCount]);  
  
    [pool drain];  
    return 0;  
}
```

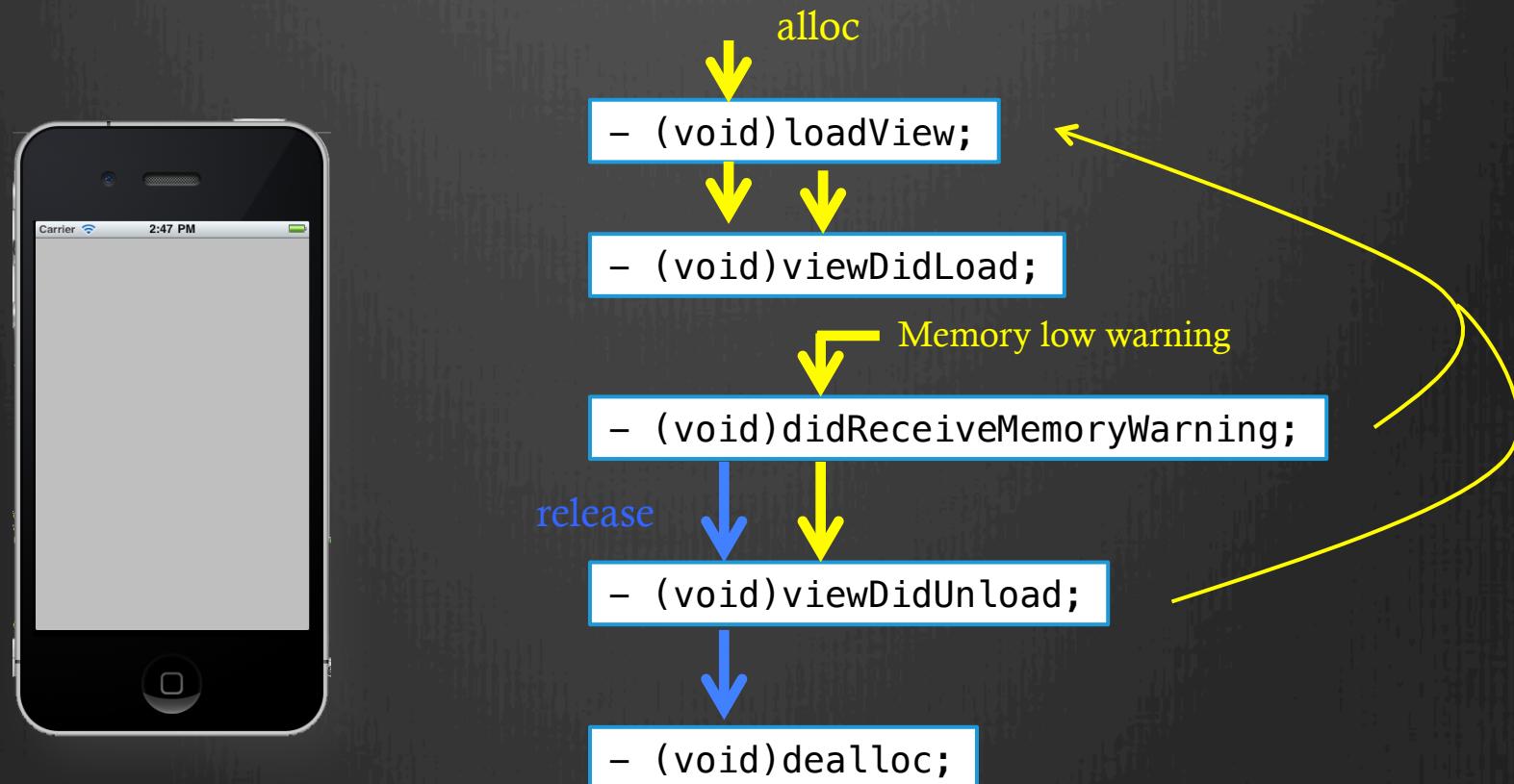
- ในกรณีที่ object ของ class A และ B มี strong reference ไปยังซึ่งกันและกัน จะเกิดปัญหาที่เรียกว่า Retain Circular คือทั้ง 2 class จะไม่มีวันที่ค่า retainCount เป็น 0
- ให้แก้ปัญหาโดยกำหนดให้ class หนึ่งเป็น parent และอีก class หนึ่งเป็น child, parent class จะมี retain ของ child ไว้ ส่วน child เก็บ reference ของ parent ไว้เท่านั้น

Auto Memory Management Lifecycle in UI View Controller (1)

- ⦿ เมื่อ memory ของ iDevice ลดลง iOS จะส่ง message ไปยังทุก application ที่ยังทำงานอยู่ เพื่อขอคืน memory
- ⦿ เราสามารถจำลองเหตุการณ์ memory low บน emulator ได้ โดย run emulator และเลือก menu Hardware -> Simulate Memory Warning
- ⦿ บน device จริง การ warning จะมี 2 level โดย level แรก จะเตือนเมื่อ memory ลดลงเหลือ **20MB** และ warning level 2 จะเตือนเมื่อ memory เหลือ **10MB**
 - ⦿ เมื่อ application ได้รับ warning **level 1** method **didReceiveMemoryWarning:** ในทุก view จะถูกเรียก
 - ⦿ เมื่อ application ได้รับ warning **level 2** method **viewDidUnload:** จะถูกเรียก
 - ⦿ viewDidUnload: จะทำงานเฉพาะ view ที่ไม่ได้แสดงอยู่บน screen เช่น view ที่ถูก removeFromSuperview หรือ dismissViewControllerAnimated แต่ยังไม่ถูก release

Auto Memory Management Lifecycle in UI View Controller (2)

- ⌚ Cycle ของ view ตั้งแต่เริ่มถูก create, เมื่อเกิด memory low และกลับมา active อีกครั้ง



Tips for Improving Memory-Related Performance (1)

- ◎ alloc ช้าที่สุดเท่าที่จะเป็นไปได้ ถ้ายังไม่ได้ใช้อาย่าเพิ่ง alloc
- ◎ เมื่อไม่ได้ใช้ object เหล่านั้น ให้ release ทันที อาย่าเก็บไว้ สำหรับ ARC ให้ set เป็น nil
- ◎ พยายาม allocate memory block ให้เล็กที่สุด
- ◎ เมื่อกำหนด property เป็น strong นั่นหมายความว่า View จะ take ownership object นั้น ถ้าไม่กำหนด object เป็น nil object นั้นจะอยู่กับ view จนกว่า view จะถูก release
- ◎ ใช้ retain กับ property เท่าที่จำเป็นเท่านั้น การใช้ weak reference จะทำให้เราสามารถควบคุม memory ของ object ได้ค่อนข้างกว่า แต่โอกาสที่ app จะ crash ก็มีมากกว่าเช่นกัน
- ◎ พยายามเขียน code ใน didReceiveMemoryWarning: หรือ viewDidUnload: เพื่อ release memory ที่สามารถ load ใหม่ได้ใน viewDidLoad:
- ◎ ระวัง cycle ระหว่าง viewDidLoad: และ viewDidUnload: เพราะการเขียน code ไม่ดีจะทำให้เกิด memory leak และ application crash ได้ง่ายมาก

Tips for Improving Memory-Related Performance (2)

- ⊕ ใช้การเช็ค `if (object == nil)` เพื่อตรวจสอบว่า object นั้นถูก free ไปหรือไม่
- ⊕ พยายามตรวจสอบการใช้ memory และ memory leak ด้วย instrument บ่อยๆ ทุกครั้งที่เพิ่ม code ใหม่ๆ เข้าไป เพราะยิ่งปล่อยไว้นาน code ก็จะซับซ้อนมากขึ้น การ trace การใช้ memory ก็จะยากขึ้นด้วย
- ⊕ ศึกษาคู่มือการใช้ memory ของ class ต่างๆ ให้ดี เช่น `[UIImage imageNamed:]` กับ `[[UIImage alloc] initWithContentsOfFile:]` นั้นต่างกัน ควรใช้ให้เหมาะสมกับงาน (ใน ARC ไม่มีผล เป็นต้น)
- ⊕ เลือกวิธีที่ประหดด memory ที่สุด มากกว่าวิธีที่ละลอกที่สุด (วิธีที่ประหดด memory อาจต้องเขียน code มากกว่า หรือมีโอกาสที่จะทำให้ application crash ได้ง่ายกว่าถ้าไม่ระวัง)
- ⊕ Test application บน device ด้วย การใช้ memory บน emulator จะไม่ค่อยมีปัญหา แต่เมื่อ test บน device จริงจะพบว่า memory low warning จะเกิดบ่อยกว่าที่คิด ทำให้เราพบปัญหาจาก memory ได้ก่อน app จะถูก release

References

- Transitioning to ACR Release Note
 - <https://developer.apple.com/Library/ios/releasenotes/ObjectiveC/RN-TransitioningToARC/Introduction/Introduction.html>
- WWDC 2013 Video -- Fixing Memory Issues – Session 410