引子

从系统的角度去介绍内存管理的原理及相关技巧

- 桌面系统中很少有应用因为使用内存过多而被Kill掉，为啥iOS会呢?
- 虚拟内存为何物？为啥有时它能超过物理总内存？虚拟内存占用过高会引来内存警告吗?
- Allocations中的Dirty Size和Resident Size分别指的什么？All Heap & Anonymous VM是什么?
- iOS的内存管理机制是什么样的？它是基于什么原则来Kill掉进程的?
- 内存有分类吗？什么类型的内存可以回收?
- 我们了解自己的程序吗？什么地方占用内存多，什么地方可以优化？如何避免内存峰值过高?

目录

- 程序员对内存的关注点
- 基本概念及原理
- iOS内存管理
- 分析工具
- 最佳实践
- 业内趋势

程序员对内存的关注点

- 正确使用
  - 非法访问
  - 内存泄露
- 高效使用
  - 降低内存峰值
  - 处理内存警告
  - Cache

基本概念及原理

——内存管理的历史

- 无内存抽象

mov eax, [1000]

问题

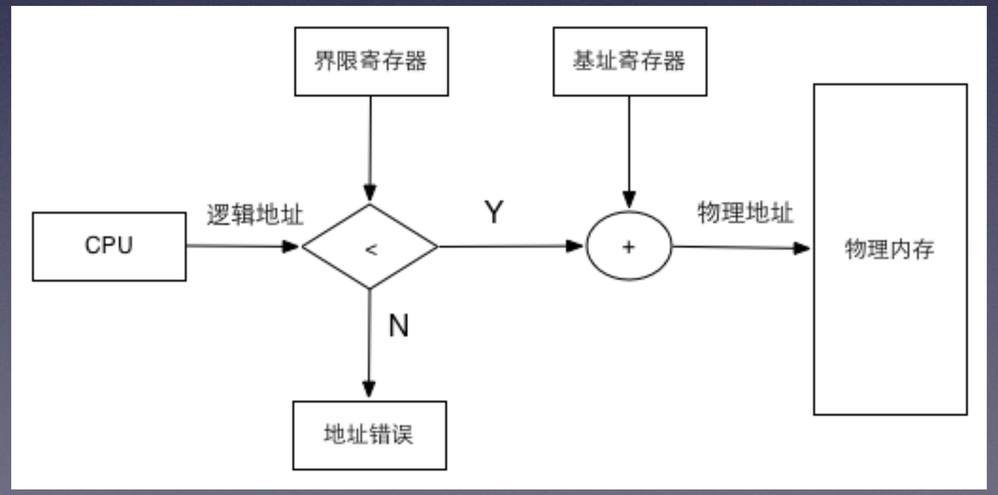
- 无存储保护
- 有限的可分配空间
- 内存利用率低（内存分配）

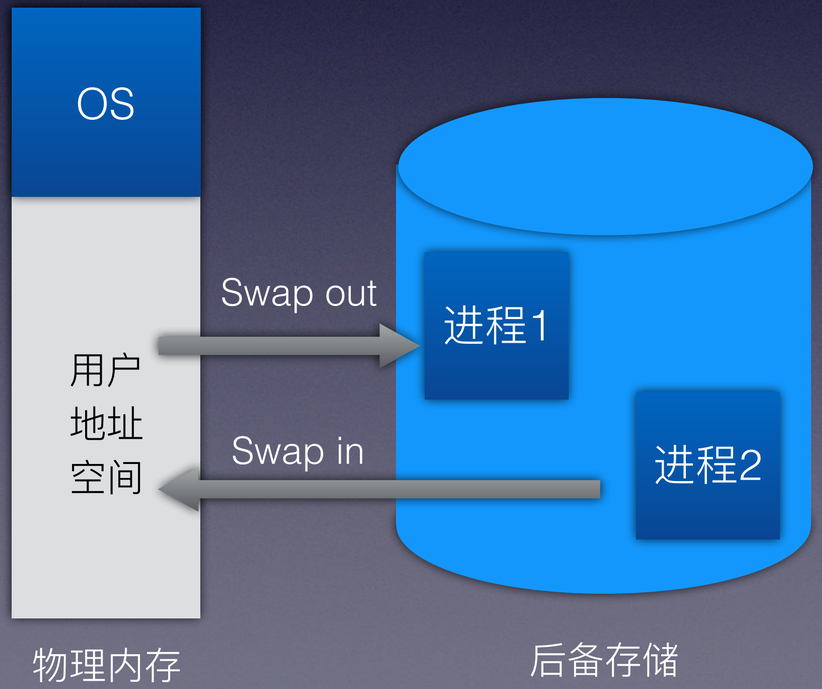
任务

- 存储保护
- 内存扩充
- 减少内存碎片

内存管理的历史(2)

- 内存抽象
  - 逻辑地址 VS 物理地址
  - Swap

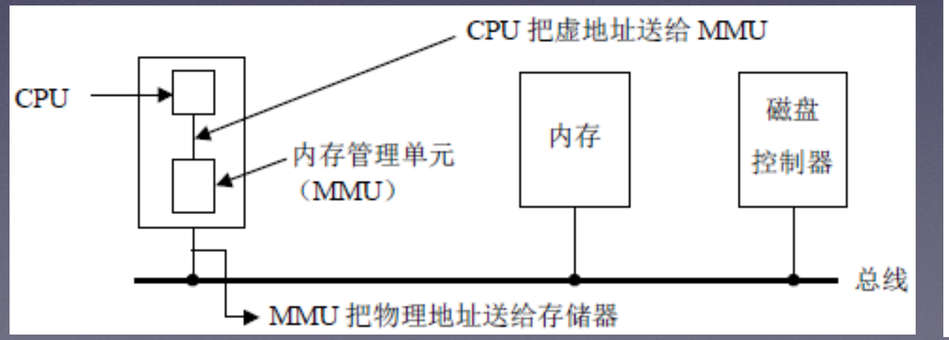


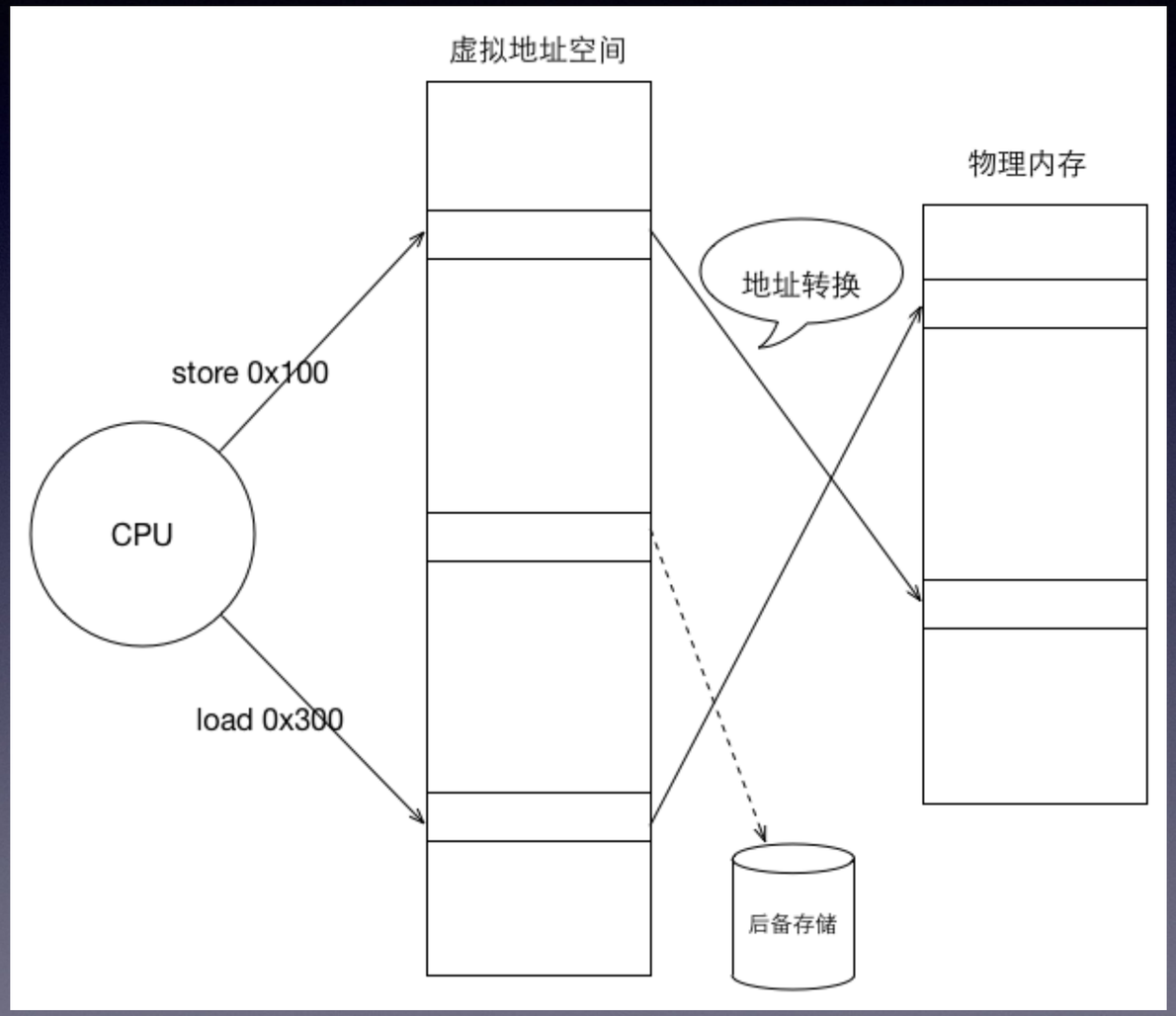


基本概念及原理

——虚拟内存

- 虚拟地址 => 物理地址，地址转换由CPU内部的内存管理单元(MMU)处理
- 巨大的虚拟地址空间
  - 32位 4GB
  - 64位 16GGB

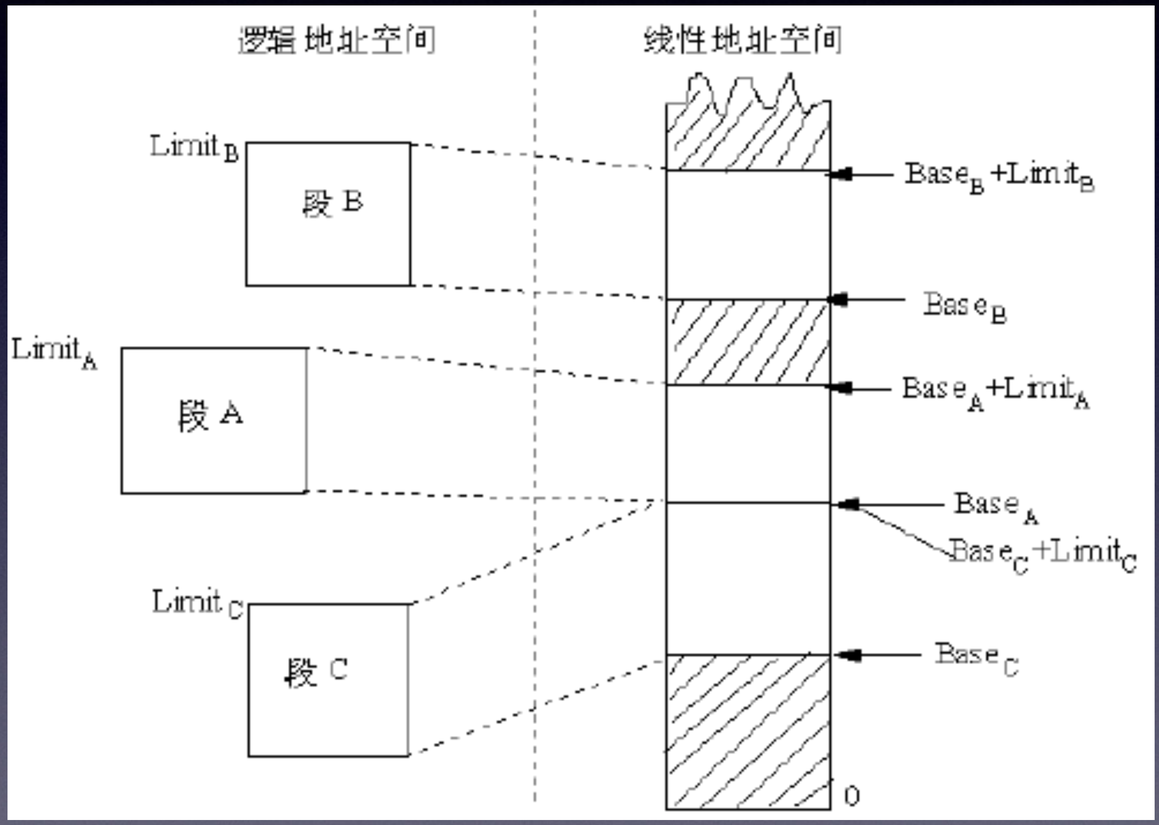




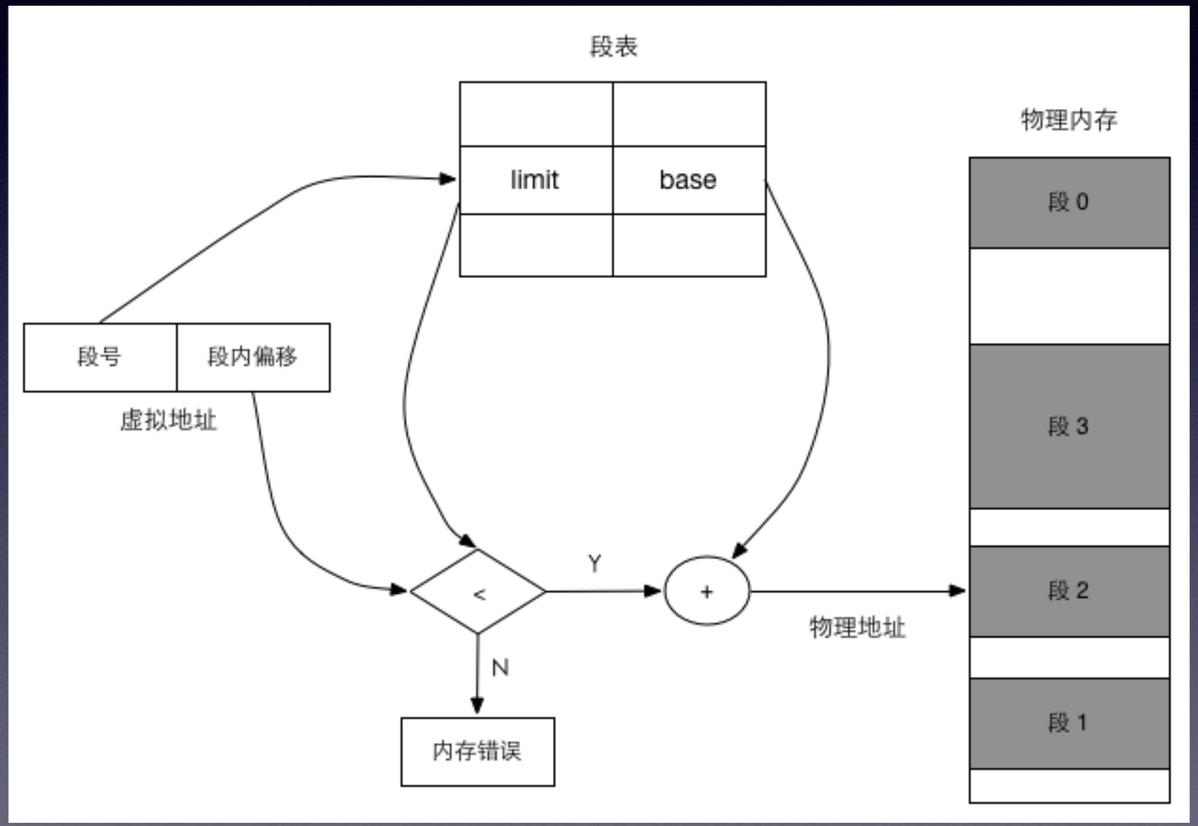
基本概念及原理

——段式虚拟内存

- 整个进程空间 VS 小单位的段
- 连续分配 VS 离散分配
- 无权限区分 VS 按逻辑分配权限



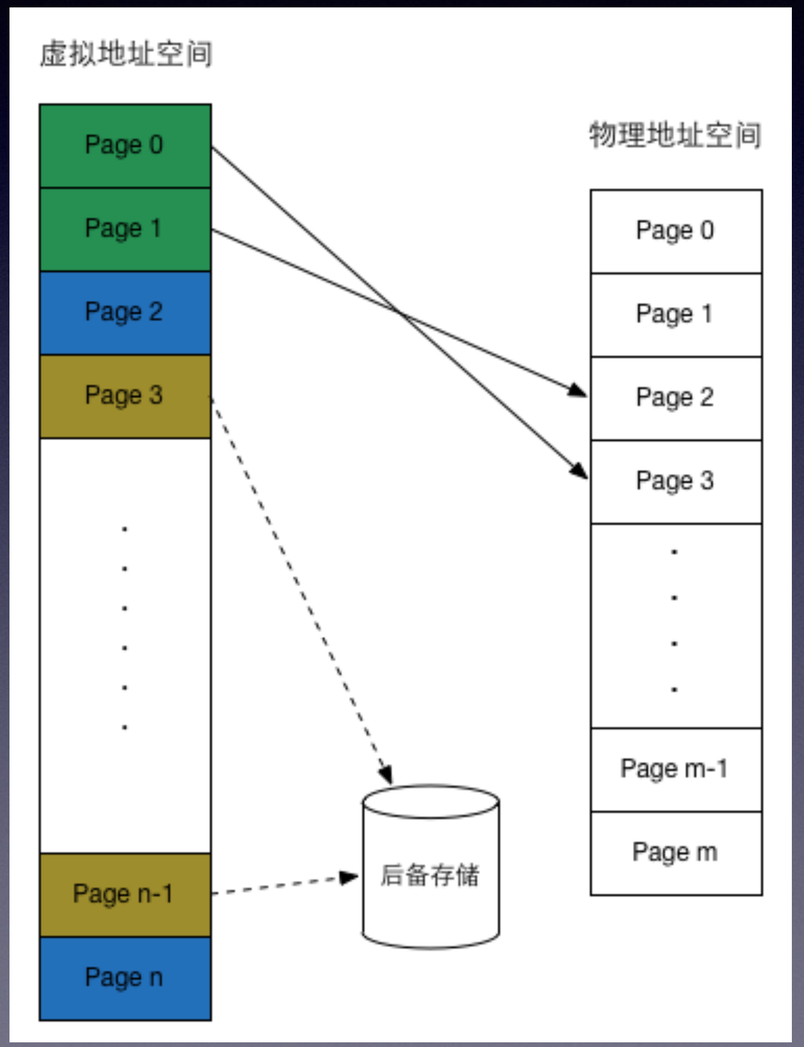
段式虚拟内存(2)



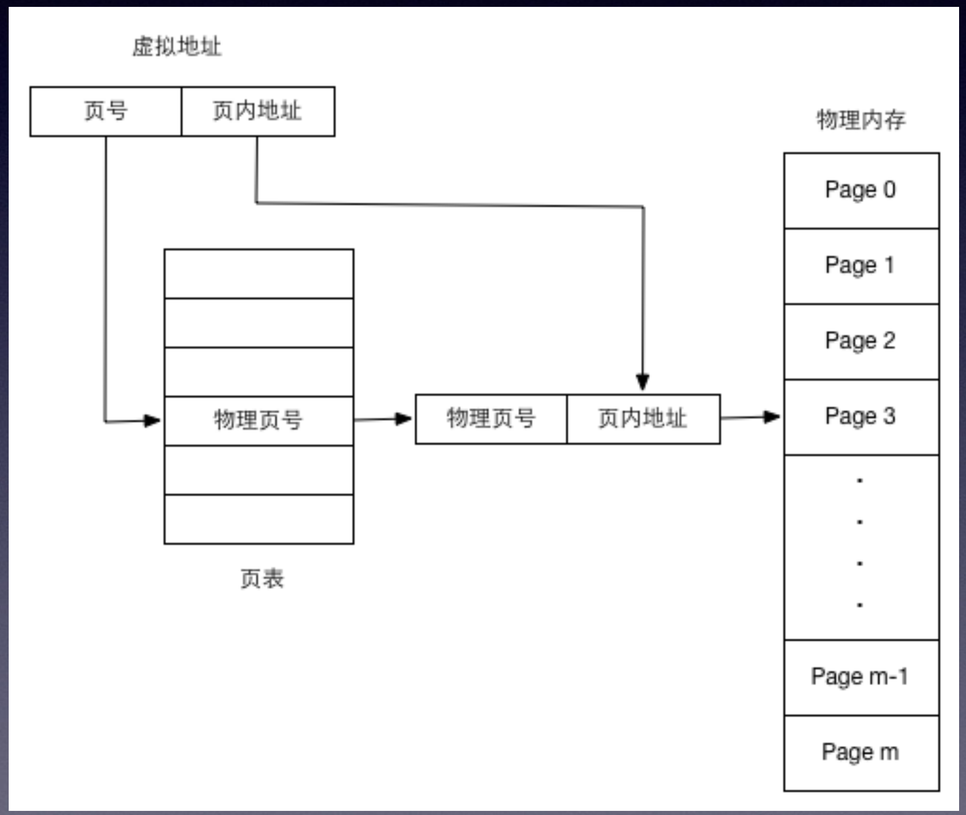
基本概念及原理

——页式虚拟内存

- 解决外部碎片
- 更小的分配及置换单位，离散分布
- page fault
- page in & page out



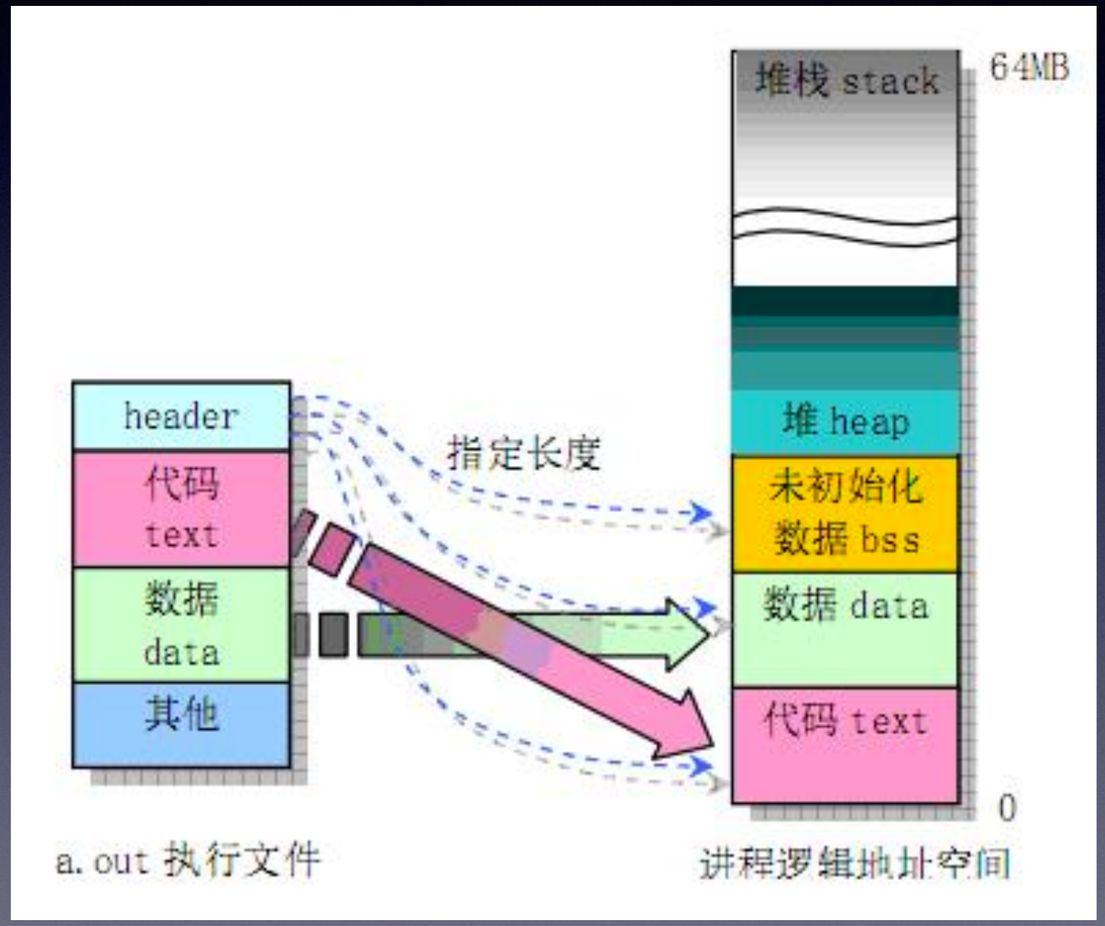
页式虚拟内存(2)


虚拟地址
页号
页内地址
物理内存
Page 0
Page 1
Page 2
Page 3
物理页号
物理页号
页内地址
Page m-1
Page m
页表

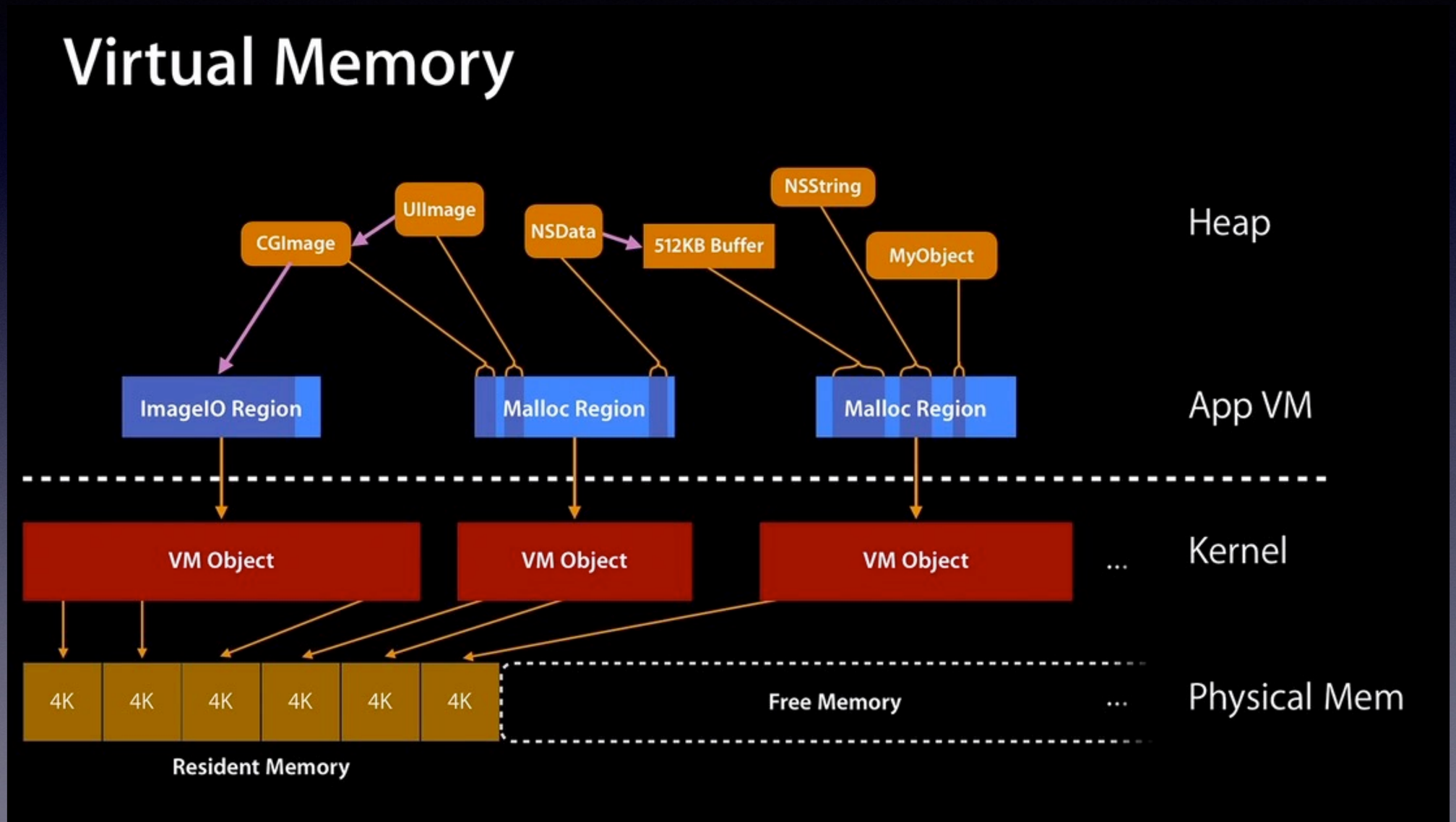
基本概念及原理

——程序内存分布

- iOS的内存段
  - __PAGEZERO
  - __TEXT
  - __DATA
  - __MALLOC_TINY
  - __MALLOC_SMALL
  - __MALLOC_LARGE



iOS内存管理


Virtual Memory
UIImage
CGImage
NSData
512KB Buffer
NSString
MyObject
Heap
ImageIO Region
Malloc Region
Malloc Region
App VM
VM Object
VM Object
VM Object
...
Kernel
4K
4K
4K
4K
4K
4K
Free Memory
...
Physical Mem
Resident Memory

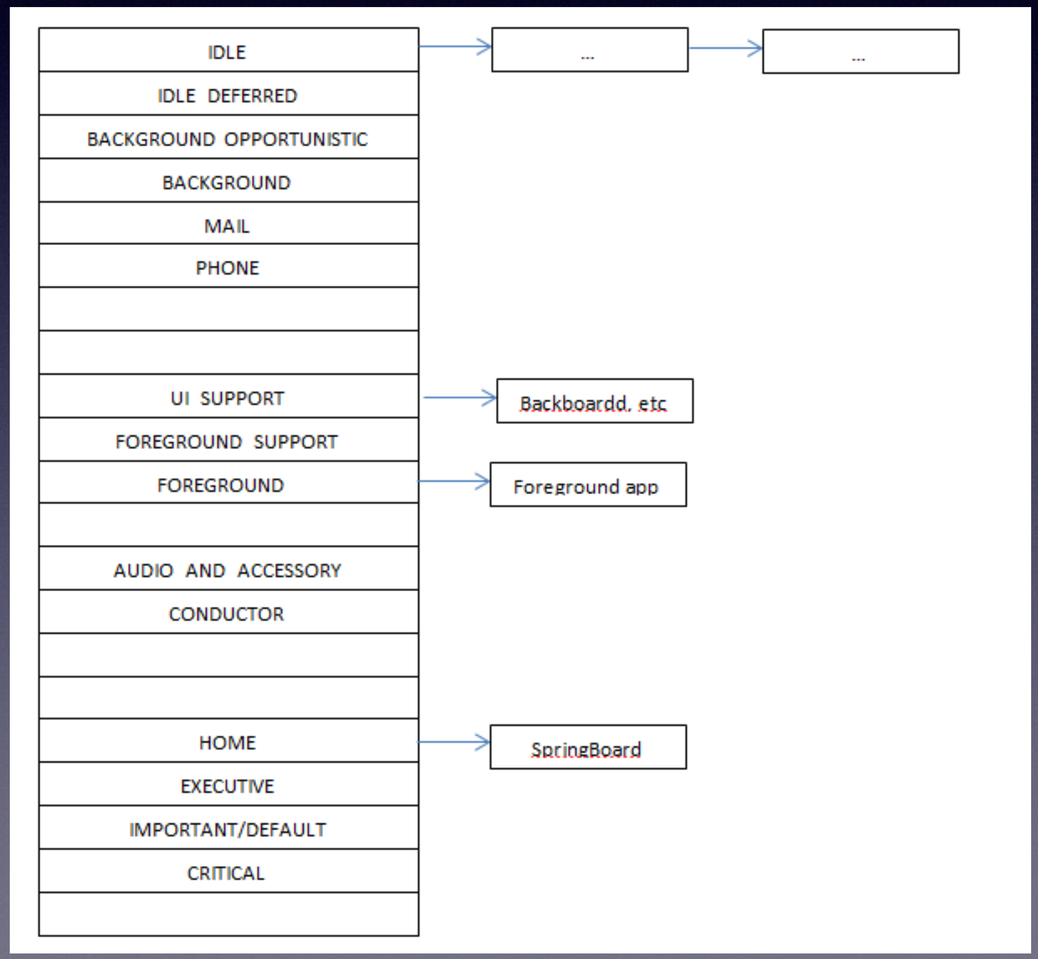
iOS内存管理(2)

- 无Swap机制
  - 移动设备的闪存容量有限
  - 闪存的写次数有限，频繁写会降低寿命
- 思考
  - 代码是要加载到内存执行的，如果没有Swap机制那代码很大的程序岂不是很占内存?

iOS内存管理(3)

——低内存处理机制Jetsam

- 基于优先级队列



低内存处理机制Jetsam(2)

- UIKit提供3种通知方式
  - [UIApplicationDelegate applicationDidReceiveMemoryWarning:]
  - [UIViewController didReceiveMemoryWarning:]
  - UIApplicationDidReceiveMemoryWarningNotification
- 内存警告消息来自主线程，应避免主线程这时候卡顿或者分配过大内存或者快速分配
- 如果App因为内存警告被Kill掉，会生成LowMemory***.log

iOS内存管理(4)

——内存分类

- Clean Memory: 在闪存中有备份，能再次读取重建
  - Code, framework, memory-mapped files
- Dirty Memory: 所有非Clean Memory，**系统无法回收**
  - Heap allocations, decompressed images, caches

iOS内存管理(4)

——Clean & Dirty Memory

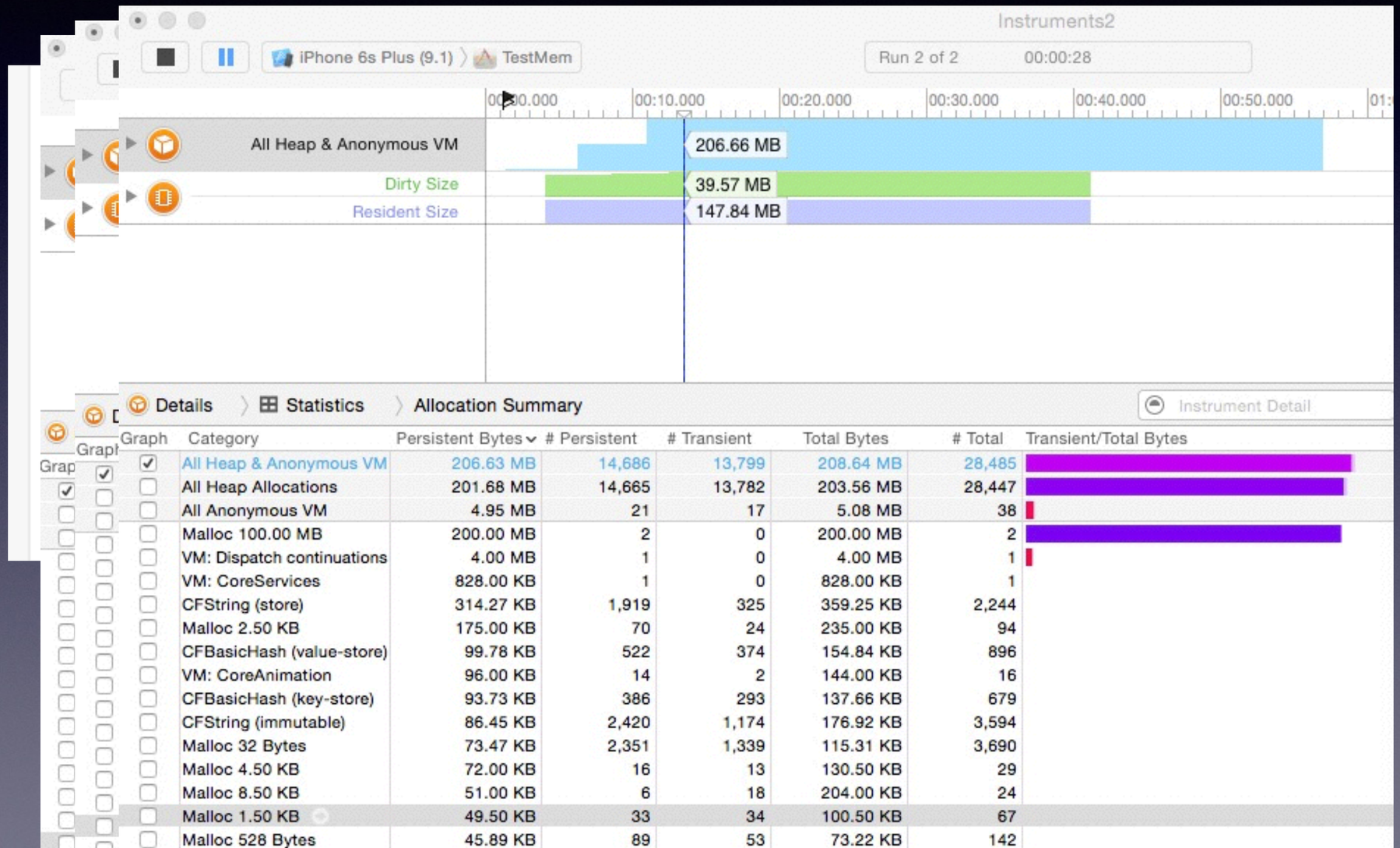
```
- (void)dirtyOrCleanMemory
{
    NSString *str1 = [NSString stringWithString:@"Welcome!"];

    NSString *str2 = @"Welcome!";

    char *buf = malloc(100 * 1024 *1024);

    for (int i = 0; i < 3 * 1024 * 1024; ++i) {
        buf[i] = rand();
    }


}
```

iOS内存管理(5)

——Dirty & Resident & Virtual Memory

- 虚拟内存层面
  - Virtual Mem = Clean Mem + Dirty Mem
- 物理内存层面
  - Resident Mem = Clean Mem(Loaded in Physical Mem) + Dirty Mem

分析工具
——Allocations



Graph	Category	Persistent Bytes	# Persistent	# Transient	Total Bytes	# Total	Transient/Total Bytes
☑	All Heap & Anonymous VM	206.63 MB	14,686	13,799	208.64 MB	28,485	
☐	All Heap Allocations	201.68 MB	14,665	13,782	203.56 MB	28,447	
☐	All Anonymous VM	4.95 MB	21	17	5.08 MB	38	
☐	Malloc 100.00 MB	200.00 MB	2	0	200.00 MB	2	
☐	VM: Dispatch continuations	4.00 MB	1	0	4.00 MB	1	
☐	VM: CoreServices	828.00 KB	1	0	828.00 KB	1	
☐	CFString (store)	314.27 KB	1,919	325	359.25 KB	2,244	
☐	Malloc 2.50 KB	175.00 KB	70	24	235.00 KB	94	
☐	CFBasicHash (value-store)	99.78 KB	522	374	154.84 KB	896	
☐	VM: CoreAnimation	96.00 KB	14	2	144.00 KB	16	
☐	CFBasicHash (key-store)	93.73 KB	386	293	137.66 KB	679	
☐	CFString (immutable)	86.45 KB	2,420	1,174	176.92 KB	3,594	
☐	Malloc 32 Bytes	73.47 KB	2,351	1,339	115.31 KB	3,690	
☐	Malloc 4.50 KB	72.00 KB	16	13	130.50 KB	29	
☐	Malloc 8.50 KB	51.00 KB	6	18	204.00 KB	24	
☐	Malloc 1.50 KB	49.50 KB	33	34	100.50 KB	67	
☐	Malloc 528 Bytes	45.89 KB	89	53	73.22 KB	142	

最佳实践
——Weak Strong Dance

```
Person *person = [Person new];
person.doSomething = ^{
    [person sayHi];
};
```

```
Person *person = [Person new];
__weak __typeof(person) weakPerson = person;
person.doSomething = ^{
    __typeof(person) strongPerson = weakPerson;
    [strongPerson sayHi];
};
```

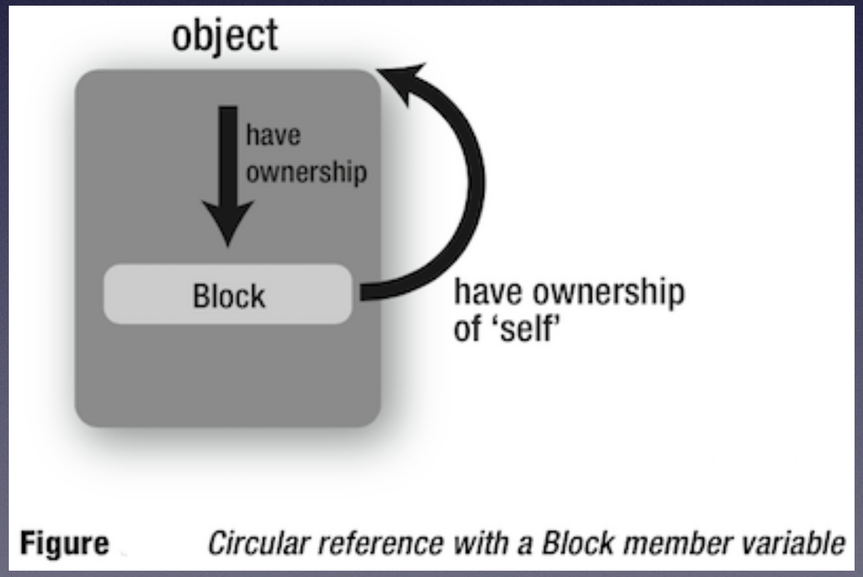


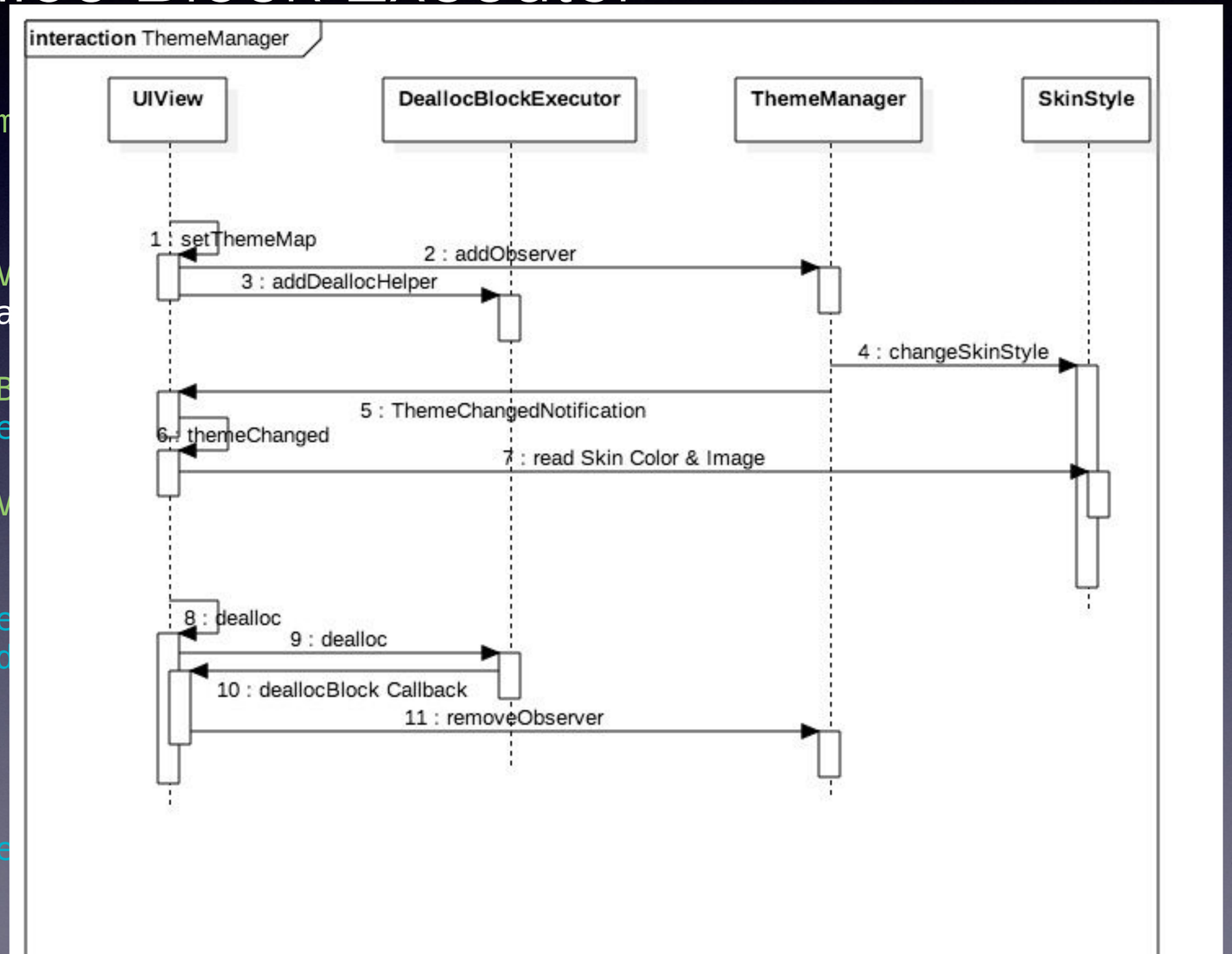
Figure *Circular reference with a Block member variable*

```
Person *person = [Person new];
DefineWeakVarBeforeBlock(person);
person.doSomething = ^{
    DefineStrongVarInBlock(person);
    [person sayHi];
};
```

最佳实践(2)

——Dealloc Block Executor

```
- (void)setThemeMap:(NSDictionary *)themeMap
{
    objc_setAssociatedObject(self, &kUIView_Them

    if (themeMap) {
        // Need to removeObserver in dealloc
        if (objc_getAssociatedObject(self, &kUIV
            __unsafe_unretained typeof(self) wea
__weak var will be reset to nil in dealloc
            id deallocHelper = [self addDeallocB
                [[NSNotificationCenter defaultCe
            }];
            objc_setAssociatedObject(self, &kUIV
        }

        [[NSNotificationCenter defaultCenter] re                    ;
        [[NSNotificationCenter defaultCenter] ad
name:kThemeDidChangeNotification object:nil];
        [self themeChanged];
    }
    else {
        [[NSNotificationCenter defaultCenter] re                    ;
    }
}
```



最佳实践(3)

——降低内存峰值

- Lazy Allocation
- alloca VS malloc
- calloc VS malloc + memset
- AutoreleasePool
- imageNamed VS imageWithContentOfFile
- NSData with fileMapping

Lazy Allocation

```
MyBuffer *GetGlobalBuffer()
{
    static MyBuffer *sMyBuffer = NULL;
    static dispatch_once_t onceToken;
    dispatch_once(&onceToken, ^{
        sMyBuffer = [MyBuffer new];
    });
    return sMyBuffer;
}
```

- 直到使用的时候才分配
- 线程安全
- 不仅是分配对象，还有资源文件读取。。
- 方便Patch，如JSPatch的场景

栈内存分配alloca(size_t)

- 栈分配仅仅修改栈指针寄存器，比malloc遍历并修改空闲列表要快得多
- 栈内存一般都已经在物理内存中，不用担心页错误
- 函数返回的时候栈分配的空间会自动释放
- 但仅适合于小空间的分配，并且函数嵌套不宜过深

calloc VS malloc + memset

calloc(size_t num, size_t size)

- 分配内存并初始化
- 立即分配虚拟空间并设置清0标记位，但不分配物理内存
- 只有相应的虚拟地址空间被读写操作的时候才需要分配相应的物理内存页并初始化

NSAutoReleasePool

- 基于引用计数，Pool执行drain方法会release所有该Pool中的autorelease对象
- 可以同时嵌套多个AutoReleasePool
- 每个线程并没用默认的AutoReleasePool，需要手动创建，避免内存泄露
- 在一段内存分配频繁的代码中适当嵌套AutoReleasePool有利于降低整体内存峰值

图片读取

- imageNamed
  - 使用系统缓存，适用于频繁使用的小图片
- imageWithContentOfFile
  - 不带缓存机制，适用于大图片，使用完就释放

NSData & 内存映射文件

```
[NSData dataWithContentsOfFile:path];

[NSData dataWithContentsOfFile:path
options:NSDataReadingMappedIfSafe error:&error];
```

- 映射文件到虚拟内存，只有读取操作的时候才会读取相应页的内容到物理内存页中
- 大文件建议采用内存映射的方式

最佳实践(4)

——NSCache & NSPurgableData

- NSCache
  - 2种界限条件：totalCostLimit & countLimit
  - 类NSMutableDictionary，setObject:forKey:cost
  - evictsObjectWithDiscardContent & <NSDicardableContent>
  - 最好监听内存警告消息并移除所有Cache
- NSPurgableData
  - 当系统处于低内存的时候会自动移除
  - 适用于大数据

最佳实践(5)

——内存警告处理

- 尽可能释放多资源，尤其图片等占内存多的资源，等需要用的时候再重建
- 单例对象不要创建之后就一直持有数据，在内存紧张的时候释放掉
- iOS6之后系统内存紧张会自动释放CALayer的CABackingStore对象，需要使用的时候再调drawRect来构建，所以没必要将将self.view=nil，但有时候对于隐藏的ViewController直接设置self.view=nil能简化代码逻辑

内存警告处理（2）

```
@interface ViewController ()
@property (strong,readonly)NSString *testData;
@end

@implementation ViewController

@synthesize testData=_testData;

// Override the default getter for testData
-(NSString*)testData
{
    if(nil==_testData)
        _testData=[self createSomeData];
    return _testData;
}

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];

    _testData=nil;
}
```

参考

- Mac OS X and iOS Internals: To the Apple’s Core
- Apple Memory Usage Performance Guidelines
- Mobile Handset Memory Management
- Instruments User Guide