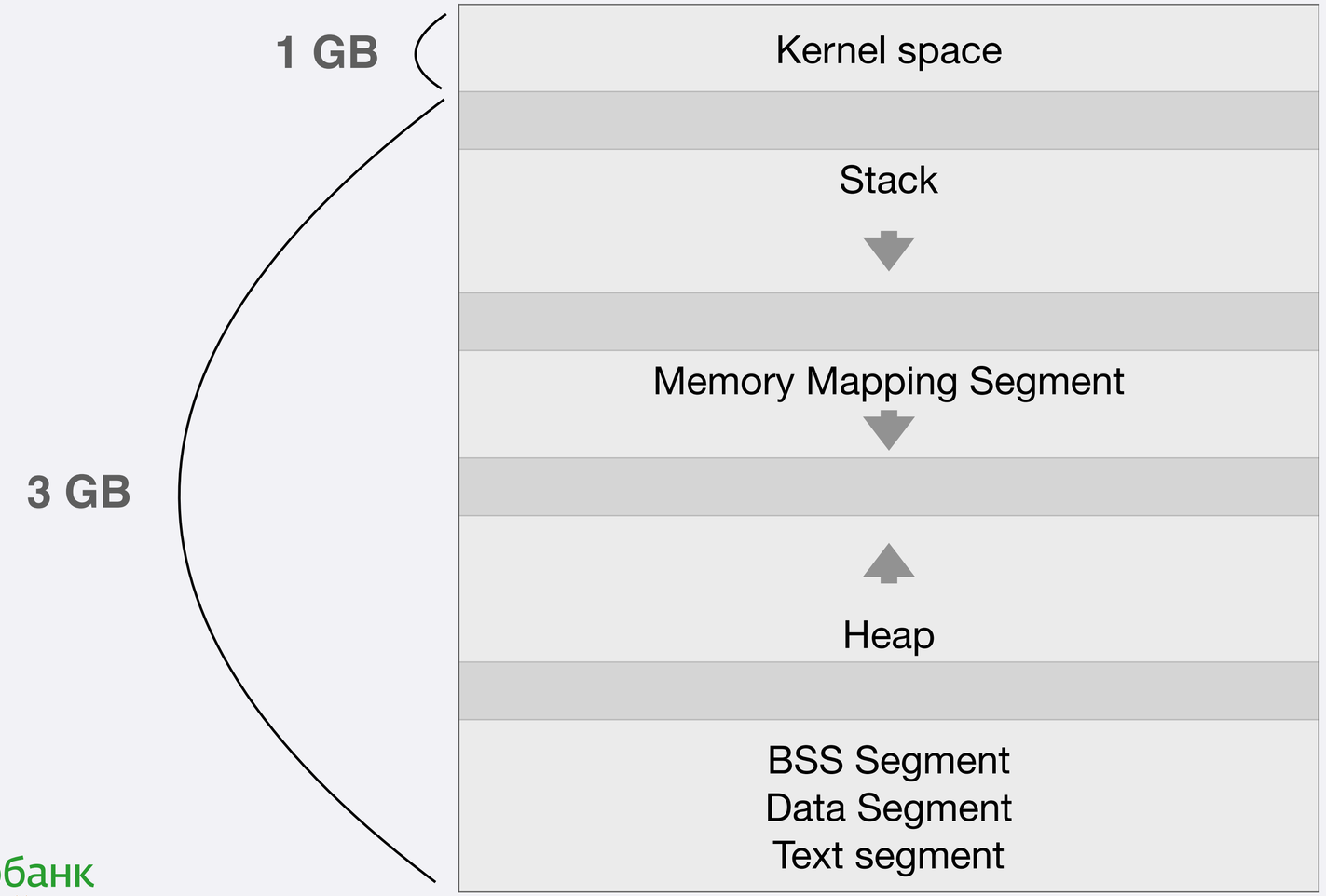
Memory management

Каждый процесс в своей «песочнице»

- Песочница - это виртуальное адресное пространство
- Представляет собой 32x - 4 GB, 64x - 256 TB блок адресов
- Программа - это тоже процесс
- Mapping виртуального адреса в физическую память
- Существует page table, которая описывает соответствие физической памяти
- Kernel space - область виртуального адресного пространства, резерверуемая под ядро
- Доступ к kernel space есть только у привилегированного кода

Организация процесса



Организация процесса

- Stack (Стэк) - автоматические переменные, передача аргументов функций и адресов возврата
- Heap (Куча) - динамические переменные
- BSS - неинициализированные статические и глобальные переменные
- Data - константы, глобальные неинициализированные переменные

Управление памятью

- Ручное (Allocators, GNU malloc)
- Автоматическое (Garbage collector)
- Полу-автоматическое (Reference counting, Memory pools)

Objective-C Reference counting

- Manual Retain Release (MRR)
- Automatic Reference Counting (ARC) - Доступна с iOS 4 и Mac OS X 10.6

Manual Retain Release

- Если вы посылаете **alloc, new, copy**, или **retain** объекту, вы должны компенсировать это, используя **release** или **autorelease**
- Если вы получили объект другим путем, и вам необходимо, чтоб объект был “живым” достаточно долгое время, вы должны использовать **retain, copy или autorelease**. Естественно, позже это должно быть компенсировано вами
- Не все объекты подчиняются этим правилам, а именно: объекты созданные литеральным способом NSString, NSNumber

Manual Retain Release

```
NSString *str = @"str";
NSLog(@"String retain count: %@", @([str retainCount]));

NSMutableString *mutableStr = [NSMutableString new];
NSLog(@"MutableString retain count: %@", @([mutableStr retainCount]));

NSNumber *number = @(2);
NSLog(@"Number retain count: %@", @([number retainCount]));
```

Manual Retain Release

- Существует конвенция, которая гласит, что все порождающие методы, которые не посылают объекту alloc, copy, new должны добавлять этот объект в autorelease pool.

```
+ (CustomClass *)createCustomClassWrong
{
    return [[CustomClass alloc] init];
}

+ (CustomClass *)createCustomClassCorrect
{
    return [[[CustomClass alloc] init] autorelease];
}
```

Manual Retain Release

```
NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
CustomClass *class = [[CustomClass new] autorelease];
[pool drain];
NSLog(@"%@", @([class retainCount]));
```

Что произойдет при выполнение программы с таким кодом?

- Не с компилируется
- Ничего не произойдет
- Программа упадет в Runtime

Manual Retain Release

```
NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
CustomClass *class = [CustomClass new];
[pool drain];
NSLog(@"%@", @([class retainCount]));
```

```
NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
CustomClass *class = [CustomClass new];
[pool addObject:class];
[pool drain];
NSLog(@"%@", @([class retainCount]));
```

Manual Retain Release

Пишем код

Automatic Reference Counting

- Тоже самое, что и MRR, но за нас это делает в момент компиляции приложения.
- NSAutoreleasePool стал @autoreleasepool { }
- retain стал strong
- Появился unsafe_unretained

Memory Management Attributes

strong	«Удерживает объект», увеличивая счетчик ссылок на 1
retain	Тоже самое, что и strong. В проекте с ARC ведет себя также как и strong
copy	Копирует объект и «удерживает» копию, увеличивая счетчик ссылок на 1
assign	«Не удерживает объект». Счетчик ссылок остается неизменным. Указатель не является безопасным *.
unsafe_unretained	«Не удерживает объект». Счетчик ссылок остается неизменным. Указатель не является безопасным *.
weak	«Не удерживает объект». Счетчик ссылок остается неизменным. Указатель будет указывать на nil (0x0), если объект деаллоцирован

**Небезопасный указатель * - если объект деаллоцирован,
то указатель на данный участок памяти остается,
что при обращении может вызвать неотложное завершение программы.**

Automatic Reference Counting

Пишем код

- Создаем RC. Delegate. Свойства с различными атрибутами владения, блоки

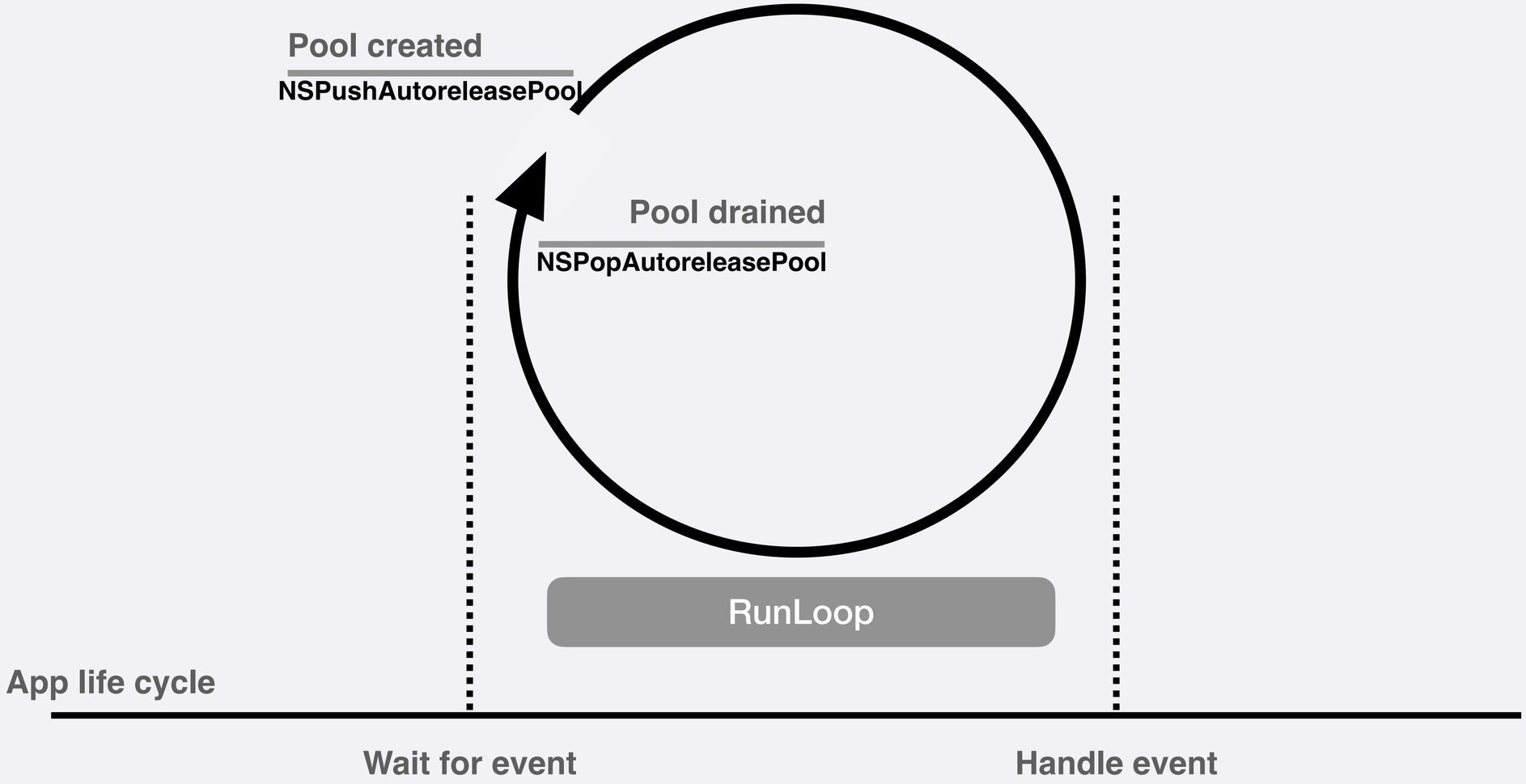
Automatic Reference Counting and Core Foundation

- Его там нет. Нужно управлять самому.
- *__bridge* Оставит количество ссылок на старые объекты неизменным. CF-объект надо будет освободить вручную.
- *__bridge_transfer* нужен для смены типа объекта с CF на Objective-C. ARC декрементирует счетчик ссылок CF, так что убедитесь, что он больше нуля.
- *__bridge_retained* нужен для смены типа объекта с Objective-C на CF. Он вернет CF-объект с счетчиком ссылок +1. Не забудьте освободить объект вызовом *CFRelease()*.

ARC in Swift

- Работает для reference types
- swift_retain() swift_release()
- weak, unowned
- Есть подсчет weak и unowned references
- Side table

AutoreleasePool in RunLoop



References

- https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/MemoryMgmt/Articles/mmRules.html#//apple_ref/doc/uid/20000994-BAJHFBGH
- https://mikeash.com/pyblog/friday-qa-2017-09-22-swift-4-weak-references.html
- https://clang.llvm.org/docs/AutomaticReferenceCounting.html