Совместная работа Kotlin/Native GC и ARC в iOS

Дмитрий Кузнецов
Мобильный разработчик

TECH

Кратко о моём опыте

- Более 13 лет в коммерческой разработке, из них 8+ в мобильной
- **C++**: Windows, macOS, Android, iOS
- **Java**, **Kotlin**: Android
- **Swift**, **Kotlin**: iOS


Познакомимся?
Mobius
ТИНЬКОФФ


Mobius
ТИНЬКОФФ


git commit -m «Отзыв о докладе»
git push
Mobius
ТИНЬКОФФ


Kotlin
Common
JVM
JS
Native
Mobius
тинькофф

Что сейчас будет?

01

Теория

Память
Алгоритмы сборки мусора
Проблематика

Что сейчас будет?

01

Теория

Память
Алгоритмы сборки мусора
Проблематика

02

Практика

Эксперименты
Выводы

Что сейчас будет?

01

Теория

Память
Алгоритмы сборки мусора
Проблематика

02

Практика

Эксперименты
Выводы

03

Kotlin source code

Как это устроено

Что сейчас будет?

01

Теория

Память
Алгоритмы сборки мусора
Проблематика

02

Практика

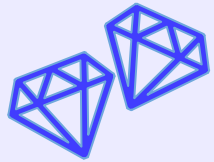
Эксперименты
Выводы

03

Kotlin source code

Как это устроено

Что сейчас будет?



01

Теория

Память
Алгоритмы сборки мусора
Проблематика

02

Практика

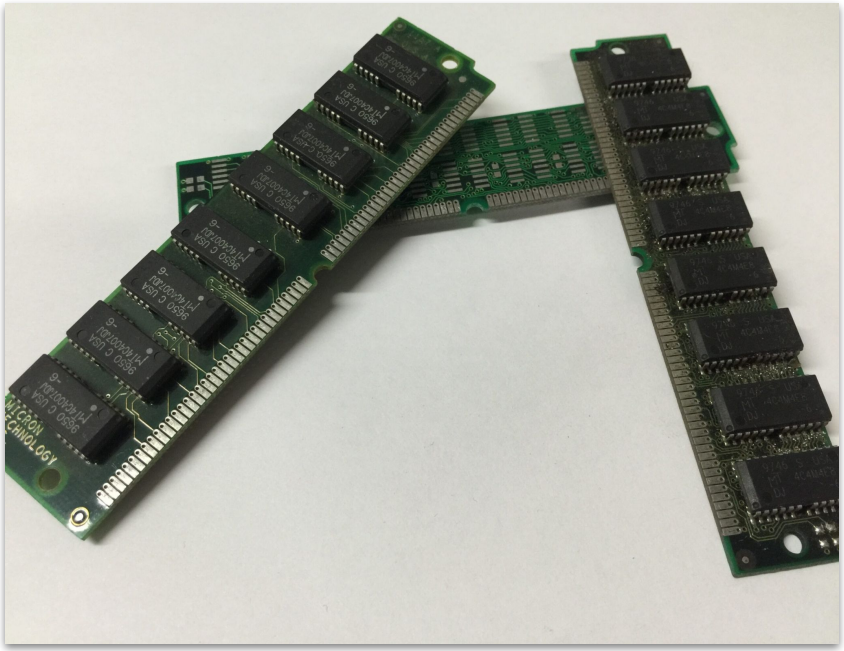
Эксперименты
Выводы

03

Kotlin source code

Как это устроено

Память

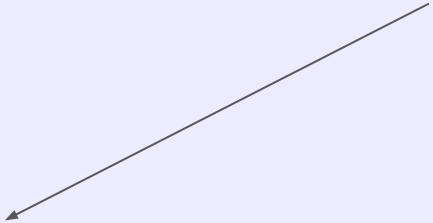


Управление памятью

Конкретный **объект** в программе ещё **используется** или уже нет**?**

Управление памятью

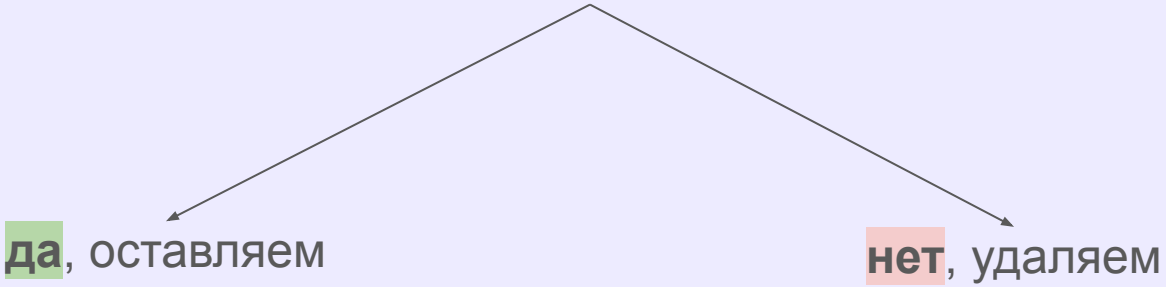
Конкретный **объект** в программе ещё **используется** или уже нет**?**



да, оставляем

Управление памятью

Конкретный **объект** в программе ещё **используется** или уже нет**?**



Управление памятью

- Ручное

Управление памятью

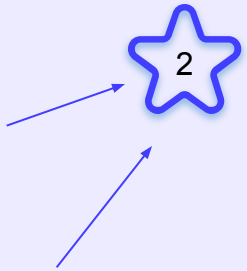
- Ручное
- Автоматическое

Автоматическое управление памятью

Reference counting

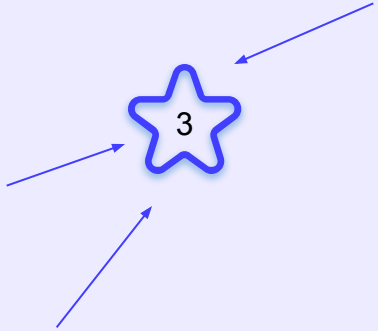
Автоматическое управление памятью

Reference counting



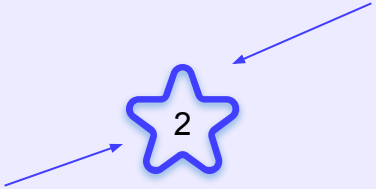
Автоматическое управление памятью

Reference counting



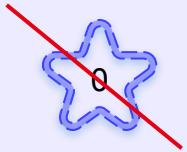
Автоматическое управление памятью

Reference counting



Автоматическое управление памятью

Reference counting



Автоматическое управление памятью

Reference counting

Автоматическое управление памятью

Reference counting

Tracing GC

Автоматическое управление памятью

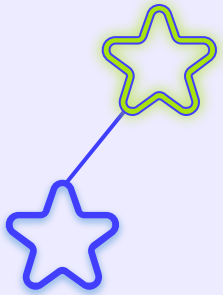
Reference counting

Tracing GC

Автоматическое управление памятью

Reference counting

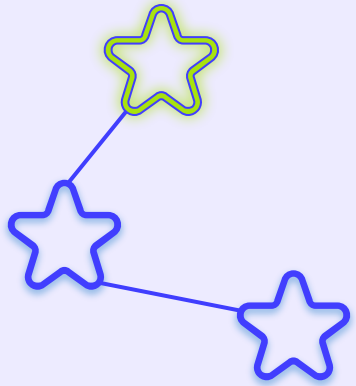
Tracing GC



Автоматическое управление памятью

Reference counting

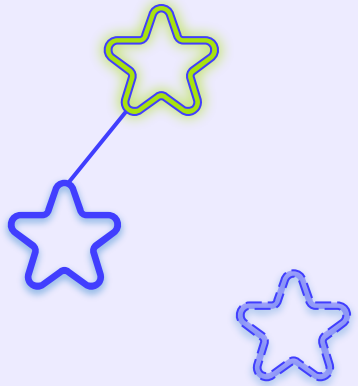
Tracing GC



Автоматическое управление памятью

Reference counting

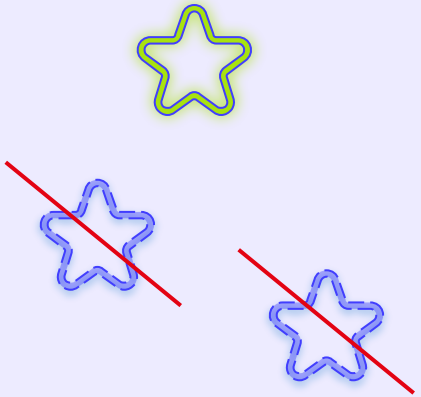
Tracing GC



Автоматическое управление памятью

Reference counting

Tracing GC

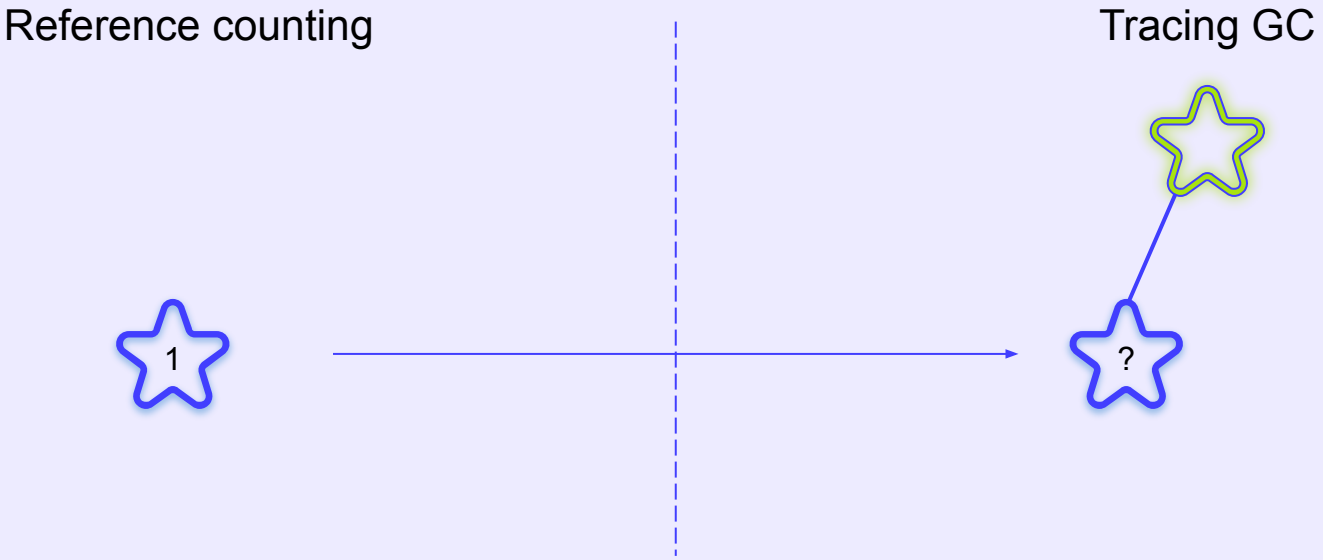


Автоматическое управление памятью

Reference counting

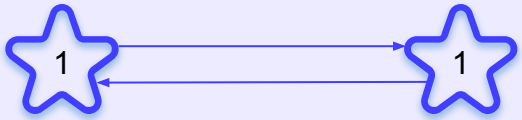
Tracing GC

Автоматическое управление памятью

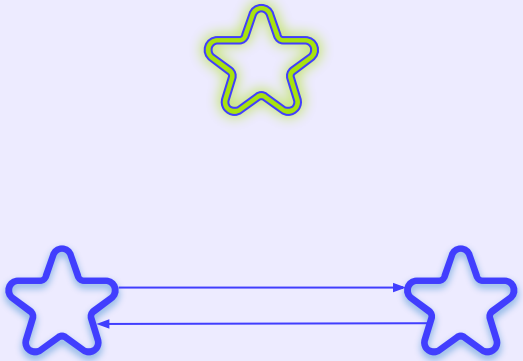


Автоматическое управление памятью

Reference counting

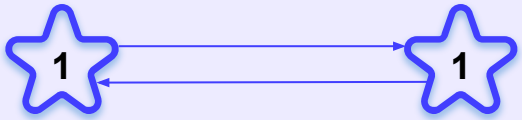


Tracing GC

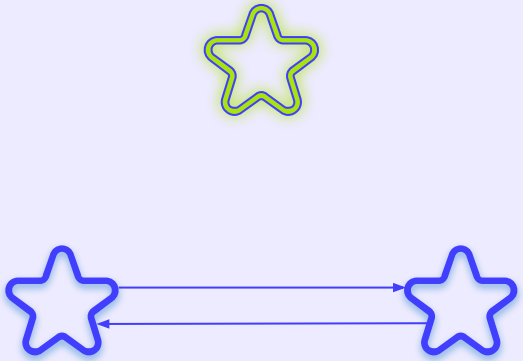


Автоматическое управление памятью

Reference counting

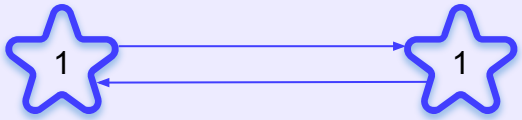


Tracing GC

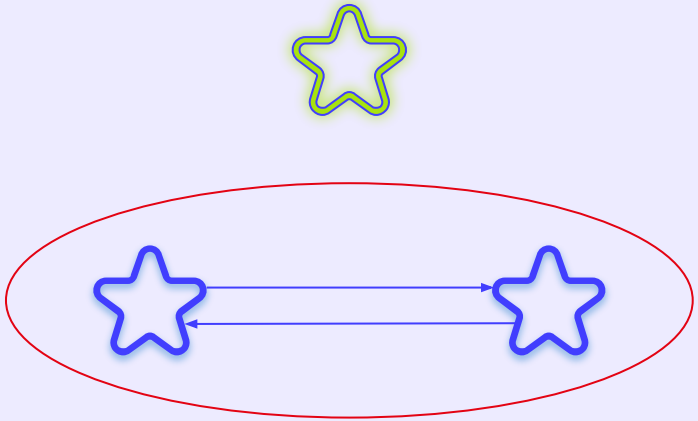


Автоматическое управление памятью

Reference counting

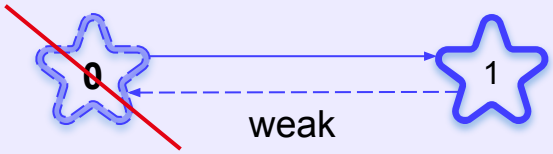


Tracing GC

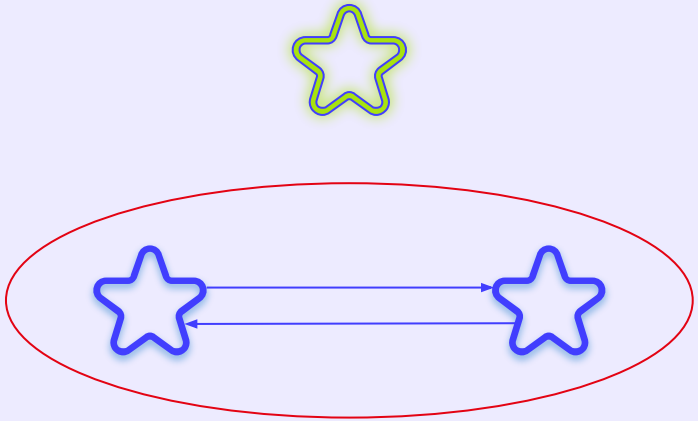


Автоматическое управление памятью

Reference counting



Tracing GC



Автоматическое управление памятью

Reference counting

Tracing GC

WeakReference

“Слабые” ссылки

Kotlin/JVM

- java.lang.ref.**WeakReference**

“Слабые” ссылки

Kotlin/Native

- kotlin.native.ref.**WeakReference**

Swift

- **weak**

Kotlin/JVM

- java.lang.ref.**WeakReference**

“Слабые” ссылки

Kotlin/Native

- kotlin.native.ref.**WeakReference**

Swift

- **weak**

Note
In systems that use garbage collection, weak pointers are sometimes used to implement a simple caching mechanism because objects with no strong references are deallocated only when memory pressure triggers garbage collection. However, with ARC, values are deallocated as soon as their last strong reference is removed, making weak references unsuitable for such a purpose.

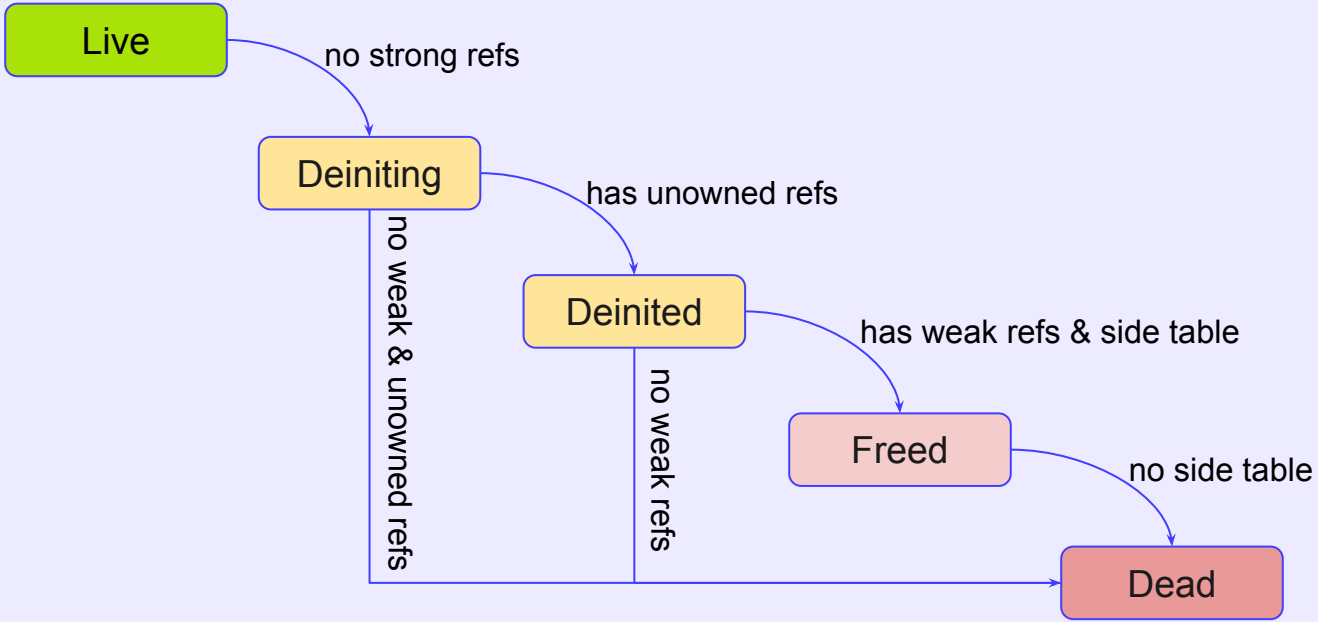
Философия Kotlin

Kotlin крутится вокруг того, чтобы … разрыв между бизнес-логикой и тем как выглядит код максимально сократить.
… управление памятью - это церемония … никакого отношения к решаемой бизнес задаче … не имеет.

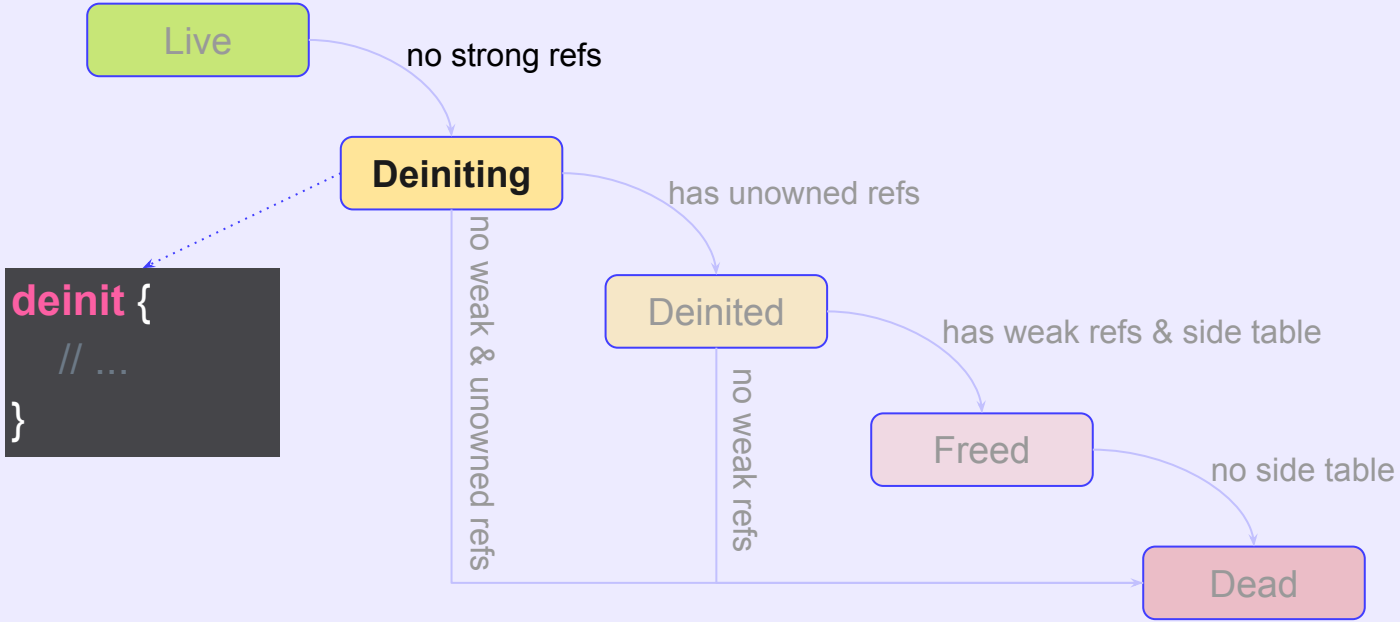
Цель - уменьшить церемонию, … **программист должен думать о задаче** и только о ней, всё **остальное лишнее**, и наша задача, как максимум, - это лишнее устранить.

Роман Елизаров
про управление памятью,
выбор алгоритма GC и
философию языка Kotlin.

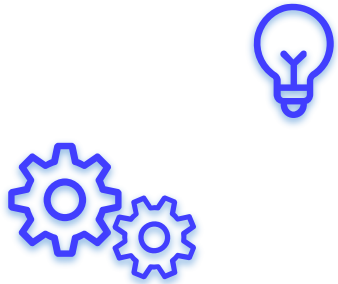
Жизненный процесс Swift объекта



Жизненный процесс Swift объекта



Эксперименты



Тестовый стенд

```
interface KotlinInterface
```

```
abstract class KotlinAbstractClass
```

Тестовый стенд

```
interface KotlinInterface
```

```
abstract class KotlinAbstractClass
```

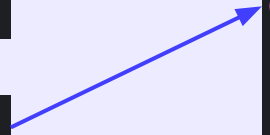


```
class Swift_KInterface: KotlinInterface {
    init() {
        print("Swift_KInterface init >")
    }
    deinit {
        print("Swift_KInterface deinit <")
    }
}
```

Тестовый стенд

```
interface KotlinInterface
```

```
abstract class KotlinAbstractClass
```



```
class Swift_KAbstractClass: KotlinAbstractClass {
    override init() {
        super.init()
        print("Swift_KAbstractClass init >")
    }
    deinit {
        print("Swift_KAbstractClass deinit <")
    }
}
```

Тестовый стенд

```
interface KotlinInterface
```

```
abstract class KotlinAbstractClass
```

```
class Swift_Class {
    init() {
        print("Swift_Class init >")
    }
    deinit {
        print("Swift_Class deinit <")
    }
}
```

Тестовый стенд

```
interface KotlinInterface
```

```
abstract class KotlinAbstractClass
```

build.gradle.kts

```
-Xruntime-logs=gc=info
```

Эксперимент #1

```
func test() {
        print(">>> BEGIN")

        Swift_Class()
        Swift_KInterface()
        Swift_KAbstractClass()

        print("<<< END")
}
```

Эксперимент #1

```
func test() {
        print(">>> BEGIN")

        Swift_Class()
        Swift_KInterface()
        Swift_KAbstractClass()

        print("<<< END")
}
```

```
>>> BEGIN
Swift_Class init >
Swift_Class deinit <
Swift_KInterface init >
Swift_KInterface deinit <
[INFO][gc][tid#6188583][0.000s] Adaptive GC scheduler initialized
…
[INFO][gc][tid#6188583][0.000s] Parallel Mark & Concurrent Sweep GC initialized
Swift_KAbstractClass init >
<<< END
```

Эксперимент #1

```
func test() {
        print(">>> BEGIN")

        Swift_Class()
        Swift_KInterface()
        Swift_KAbstractClass()

        print("<<< END")
}
```

```
>>> BEGIN
Swift_Class init >
Swift_Class deinit <
Swift_KInterface init >
Swift_KInterface deinit <
[INFO][gc][tid#6188583][0.000s] Adaptive GC scheduler initialized
…
[INFO][gc][tid#6188583][0.000s] Parallel Mark & Concurrent Sweep GC initialized
Swift_KAbstractClass init >
<<< END
```

Эксперимент #1

```
func test() {
        print(">>> BEGIN")

        Swift_Class()
        Swift_KInterface()
        Swift_KAbstractClass()

        print("<<< END")
}
```

```
>>> BEGIN
Swift_Class init >
Swift_Class deinit <
Swift_KInterface init >
Swift_KInterface deinit <
[INFO][gc][tid#6188583][0.000s] Adaptive GC scheduler initialized
…
[INFO][gc][tid#6188583][0.000s] Parallel Mark & Concurrent Sweep GC initialized
Swift_KAbstractClass init >
<<< END
```

Эксперимент #1

```
func test() {
        print(">>> BEGIN")

        Swift_Class()
        Swift_KInterface()
        Swift_KAbstractClass()

        print("<<< END")
}
```

```
>>> BEGIN
Swift_Class init >
Swift_Class deinit <
Swift_KInterface init >
Swift_KInterface deinit <
[INFO][gc][tid#6188583][0.000s] Adaptive GC scheduler initialized
…
[INFO][gc][tid#6188583][0.000s] Parallel Mark & Concurrent Sweep GC initialized
Swift_KAbstractClass init >
<<< END
```

Эксперимент #1

```
func test() {
        print(">>> BEGIN")

        Swift_Class()
        Swift_KInterface()
        Swift_KAbstractClass()

        print("<<< END")
}
```

```
>>> BEGIN
Swift_Class init >
Swift_Class deinit <
Swift_KInterface init >
Swift_KInterface deinit <
[INFO][gc][tid#6188583][0.000s] Adaptive GC scheduler initialized
…
[INFO][gc][tid#6188583][0.000s] Parallel Mark & Concurrent Sweep GC initialized
Swift_KAbstractClass init >
<<< END
```

Эксперимент #1

```
func test() {
        print(">>> BEGIN")

        Swift_Class()
        Swift_KInterface()
        Swift_KAbstractClass()

        print("<<< END")
}
```

```
>>> BEGIN
Swift_Class init >
Swift_Class deinit <
Swift_KInterface init >
Swift_KInterface deinit <
[INFO][gc][tid#6188583][0.000s] Adaptive GC scheduler initialized
…
[INFO][gc][tid#6188583][0.000s] Parallel Mark & Concurrent Sweep GC initialized
Swift_KAbstractClass init >
<<< END
…
[INFO][gc][tid#6189073][60.028s] Epoch #6: Finalization is done [...]
Swift_KAbstractClass deinit <
```

Эксперимент #2

```
func test() {
        print(">>> BEGIN")

        Swift_Class()
        Swift_KInterface()
//      Swift_KAbstractClass()

        print("<<< END")
}
```

```
>>> BEGIN
Swift_Class init >
Swift_Class deinit <
Swift_KInterface init >
Swift_KInterface deinit <
[INFO][gc][tid#6188583][0.000s] Adaptive GC scheduler initialized
…
[INFO][gc][tid#6188583][0.000s] Parallel Mark & Concurrent Sweep GC initialized
Swift_KAbstractClass init >
<<< END
…
[INFO][gc][tid#6189073][60.028s] Epoch #6: Finalization is done [...]
Swift_KAbstractClass deinit <
```

Эксперимент #3

```
fun runGC() {
    println("K/N GC RUN")
    kotlin.native.runtime.GC.collect()
}
```

Эксперимент #3

```
func test() {
        print(">>> BEGIN")

        Swift_Class()
        Swift_KInterface()
        Swift_KAbstractClass()

        FunsKt.runGC()

        print("<<< END")
}
```

```
>>> BEGIN
Swift_Class init >
Swift_Class deinit <
Swift_KInterface init >
Swift_KInterface deinit <
[INFO][gc][tid#6236286][0.000s] Adaptive GC scheduler initialized
…
[INFO][gc][tid#6236286][0.000s] Parallel Mark & Concurrent Sweep GC initialized
Swift_KAbstractClass init >
K/N GC RUN
…
[INFO][gc][tid#6236633][0.002s] Epoch #1: Finished. [...]
Swift_KAbstractClass deinit <
[INFO][gc][tid#6236634][0.002s] Epoch #1: Finalization is done [...]
<<< END
```

Эксперимент #4

```
func test() {
    print(">>> BEGIN")

    let a = Swift_Class()
    let b = Swift_KInterface()
    let c = Swift_KAbstractClass()

    FunsKt.runGC()

    print("<<< END")
}
```

```
>>> BEGIN
Swift_Class init >
Swift_KInterface init >
[INFO][gc][tid#6249034][0.000s] Adaptive GC scheduler initialized
…
…
…
…
Swift_KAbstractClass init >
K/N GC RUN
…
[INFO][gc][tid#6249479][0.002s] Epoch #1: Finalization is done [...]
<<< END
...
[INFO][gc][tid#6249479][50.028s] Epoch #6: Finalization is done [...]
Swift_KAbstractClass deinit <
```

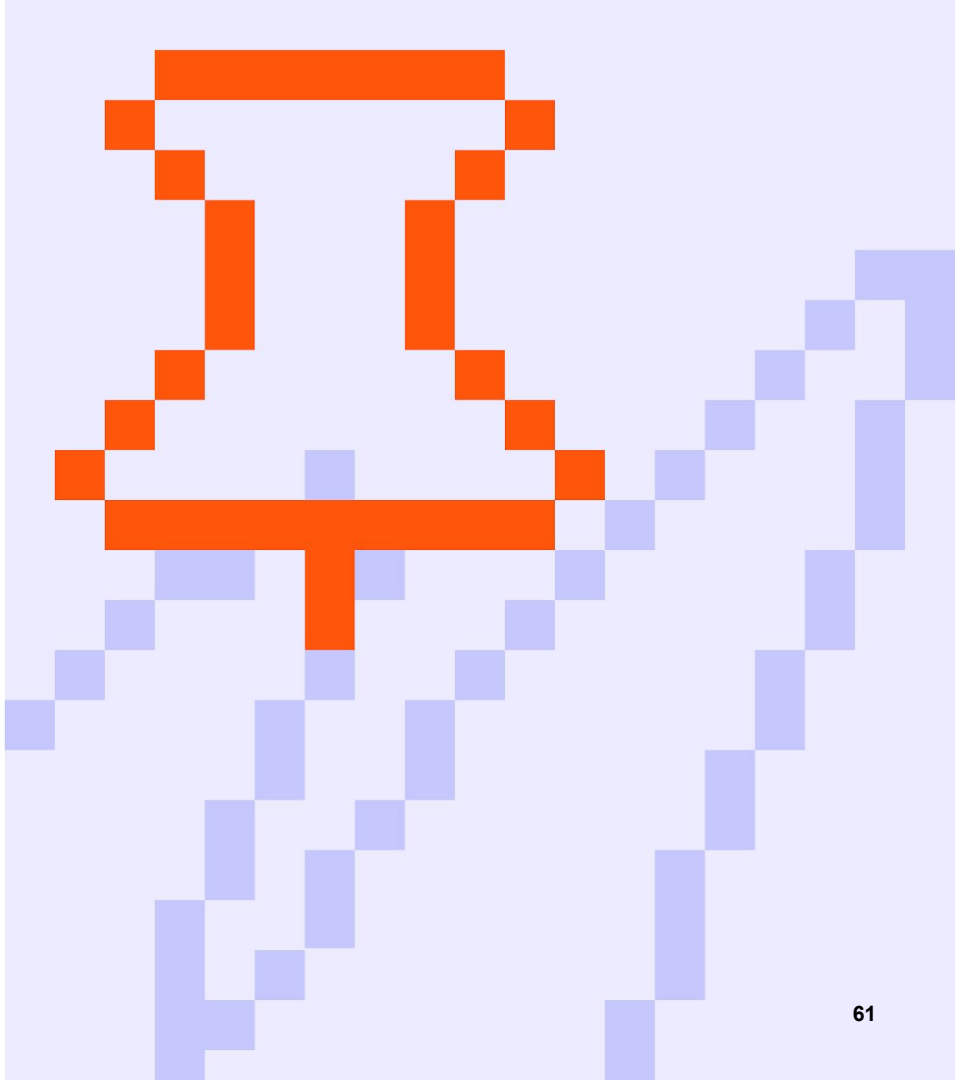
Эксперимент #5

```
func test() {
    print(">>> BEGIN")

    let a = Swift_Class()
    let b = Swift_KInterface()
    let c = Swift_KAbstractClass()
    DispatchQueue.main.async {
        FunsKt.runGC()
    }
    print("<<< END")
}
```

```
>>> BEGIN
Swift_Class init >
Swift_KInterface init >
[INFO][gc][tid#6264514][0.000s] Adaptive GC scheduler initialized
…
…
…
…
Swift_KAbstractClass init >
<<< END
…
…
K/N GC RUN
…
[INFO][gc][tid#6264974][0.009s] Epoch #1: Finished. [...]
Swift_KAbstractClass deinit <
```

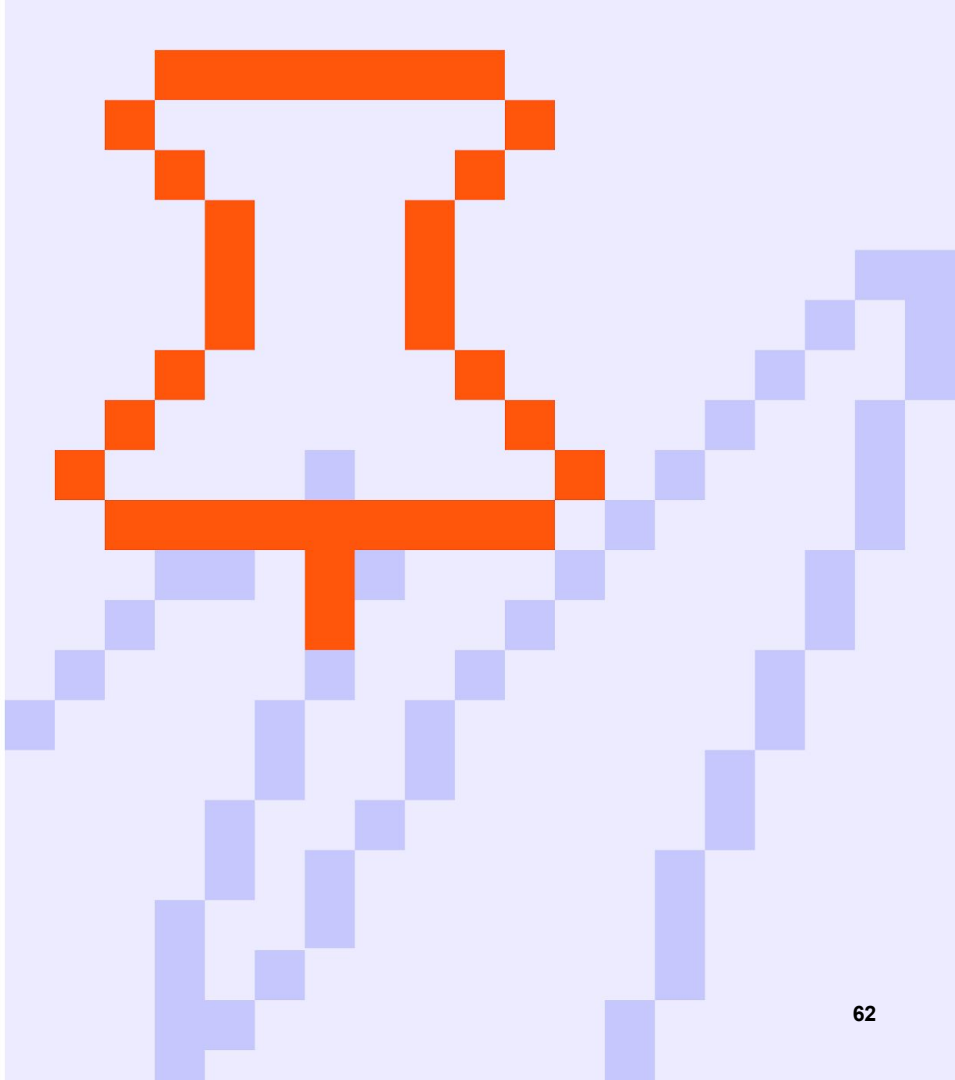
Вывод #1

- Работа K/N GC менее детерминирована, чем работа ARC



Вывод #1

- Работа K/N GC менее детерминирована, чем работа ARC
- Избегайте наследования, предпочитайте композицию



Эксперимент #6

```
func test() {
    print(">>> BEGIN")

    let a = Swift_Class()

    FunsKt.accept(value: a)

    print("<<< END")
}
```

```
fun accept(value: Any) {
}
```

Эксперимент #6

```
func test() {
    print(">>> BEGIN")

    let a = Swift_Class()

    FunsKt.accept(value: a)

    print("<<< END")
}
```

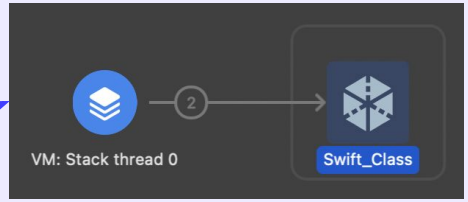
```
>>> BEGIN
Swift_Class init >
[INFO][gc][tid#6303140][0.000s] Adaptive GC scheduler initialized
…
[INFO][gc][tid#6303140][0.000s] Parallel Mark & Concurrent Sweep GC initialized
<<< END

…

[INFO][gc][tid#6313288][50.025s] Epoch #5: Finalization is done [...]
Swift_Class deinit <
```

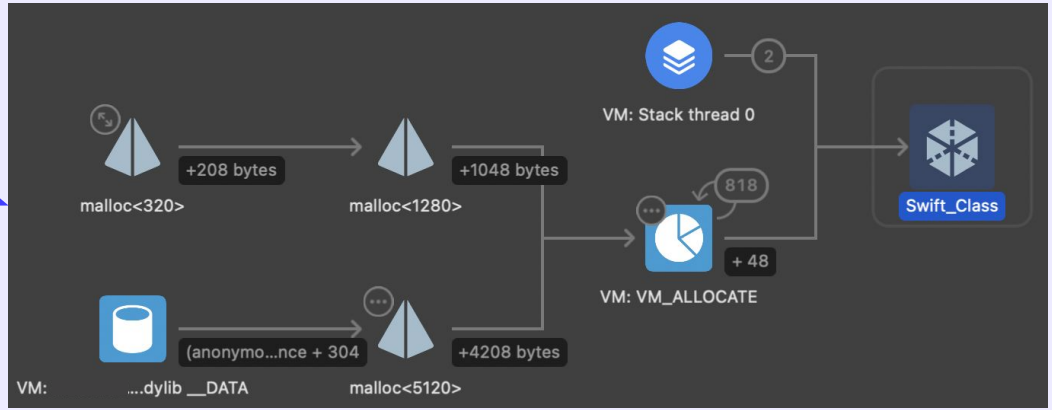
Эксперимент #7

```
func test() {
    print(">>> BEGIN")

    let a = Swift_Class()
    let b = Swift_Class()

    FunsKt.accept(value: a)

    print("<<< END")
}
```

Эксперимент #7

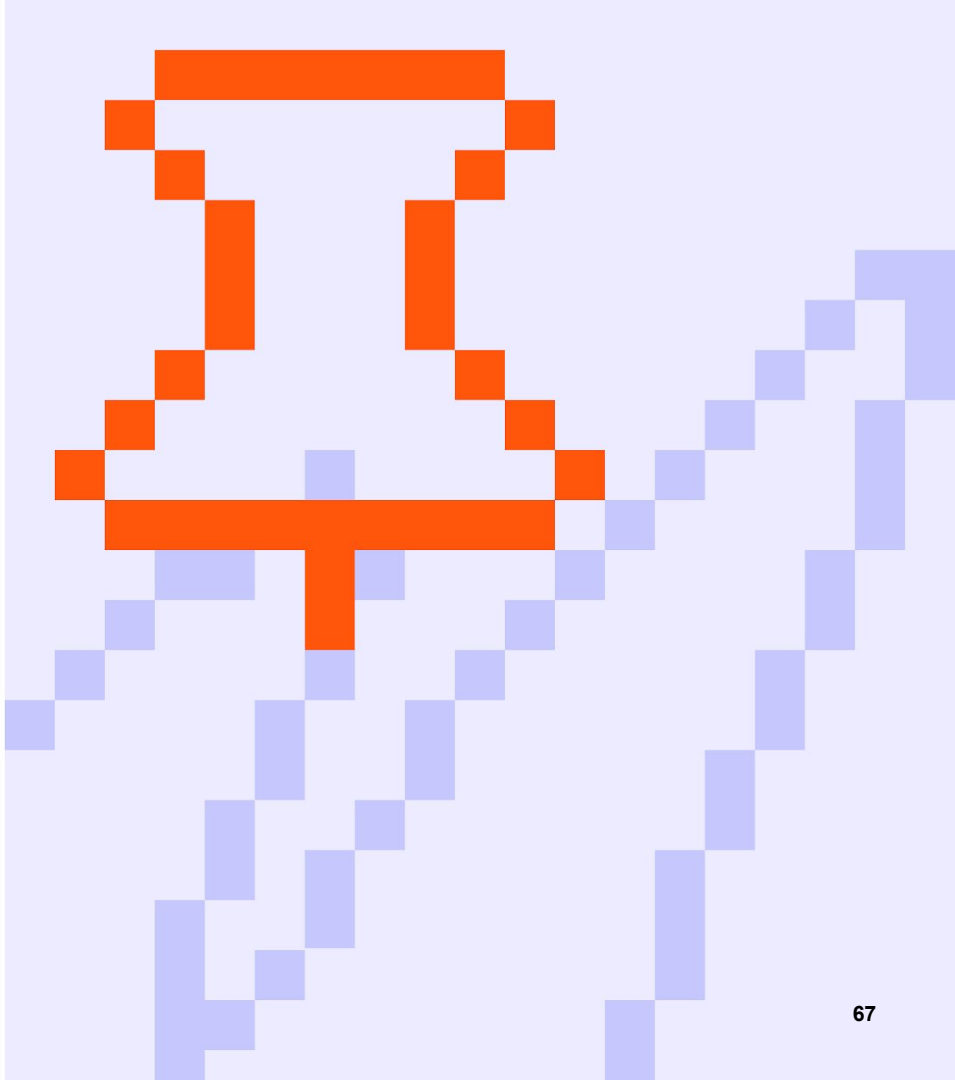
```
func test() {
    print(">>> BEGIN")

    let a = Swift_Class()
    let b = Swift_Class()

    FunsKt.accept(value: a)

    print("<<< END")
}
```





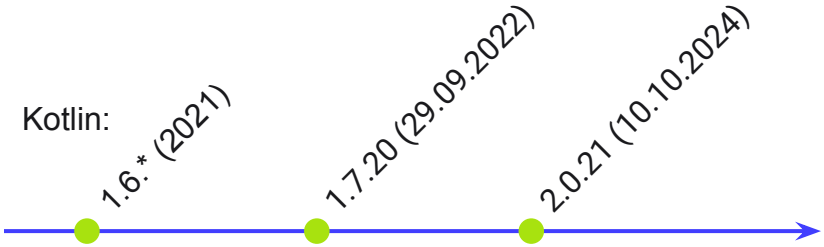
Вывод #2

1. Нет сильных ссылок со стороны ARC (Swift/Objective-C)
2. Недостижимость объекта из Kotlin/Native GC roots



Вывод #2

1. Нет сильных ссылок со стороны ARC (Swift/Objective-C)
2. Недостижимость объекта из Kotlin/Native GC roots

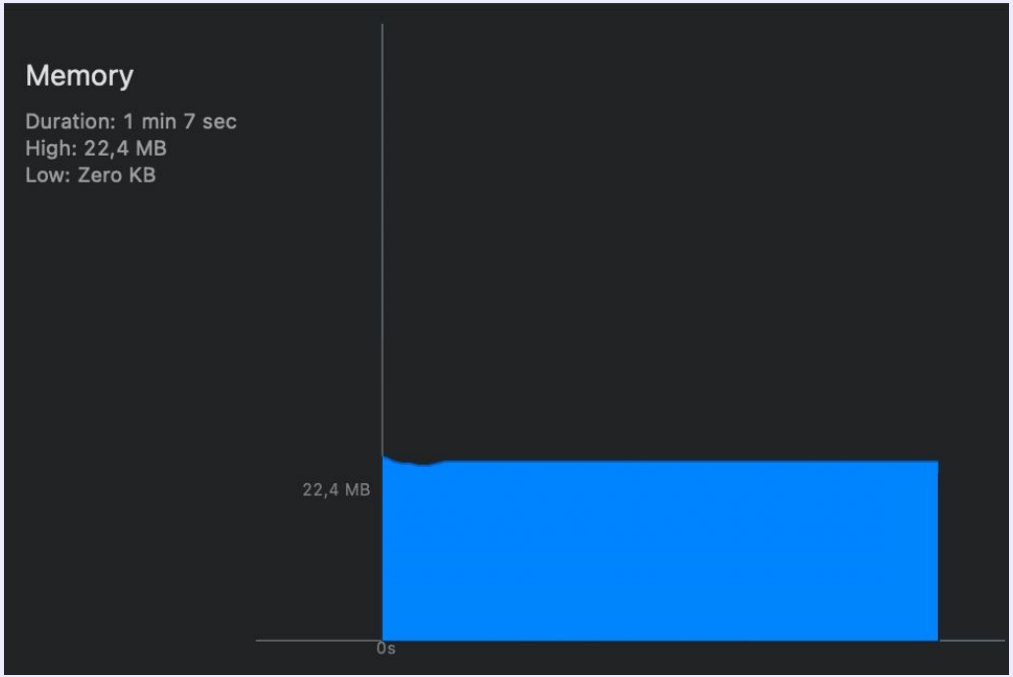


*Kotlin/Native Memory Management Roadmap

*Kotlin/Native Memory Management Update

autoreleasepool

```
fun memoryLoadTest() {
    repeat(1_000_000) {
        autoreleasepool {
            NSLog("$it\n")
        }
    }
}
```



*NSAutoreleasePool

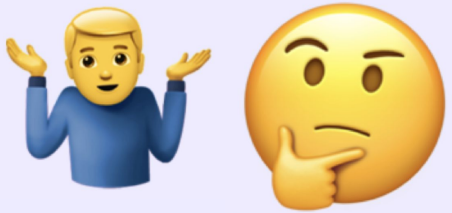
Deinitializers vs. Threads

- Main Thread

Deinitializers vs. Threads

```
func test() {
    let a = Swift_KAbstractClass()
    FunsKt.accept(value: a)
    DispatchQueue.global(
        qos: .background
    ).async {
        FunsKt.runGC()
    }
}
```

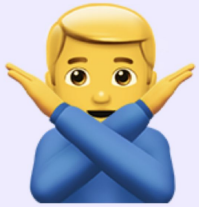
```
…
Swift_KAbstractClass init > <_NSMainThread[...]
…
Swift_KAbstractClass deinit < <_NSThread[...]
```

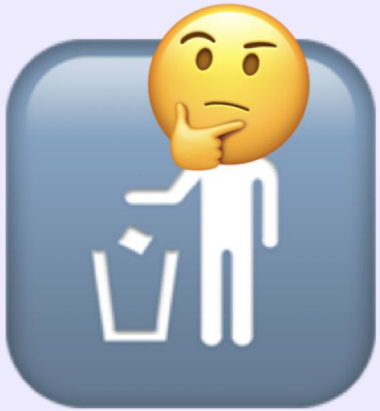


Deinitializers vs. Threads

- Main Thread
- Special GC thread
- *kotlin.native.binary.objcDisposeOnMain=false*

Kotlin/Native vs. retain cycles

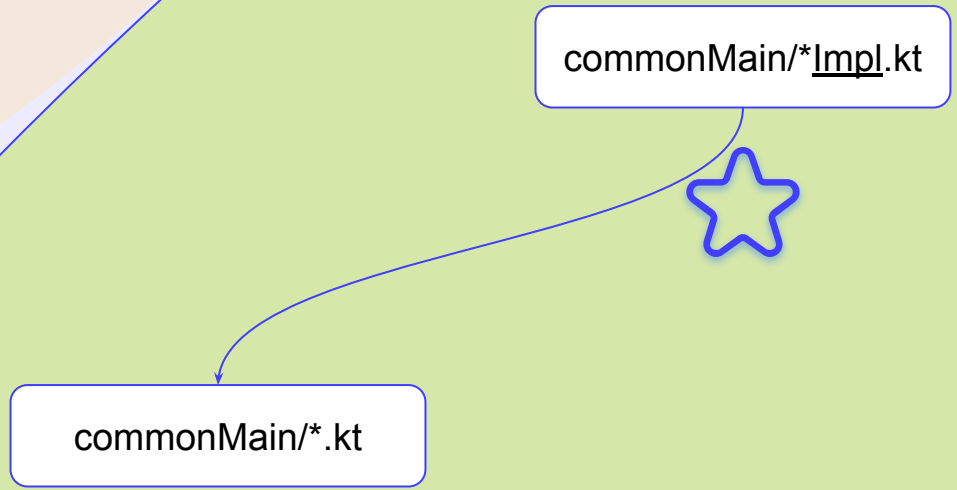




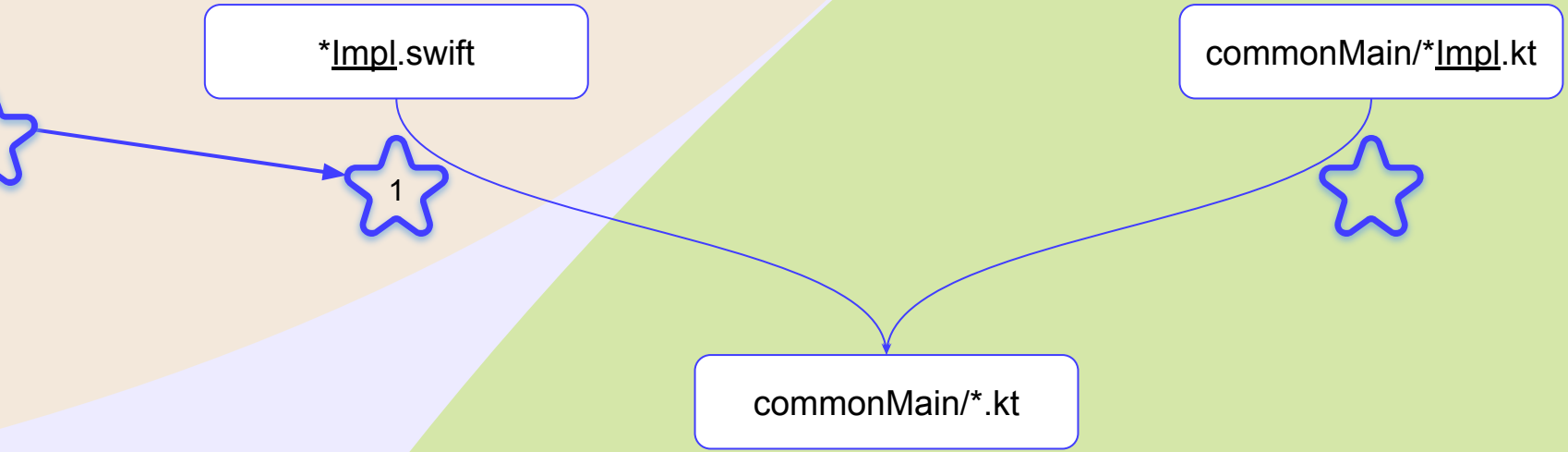
Kotlin/Native deep dive

commonMain/*.kt

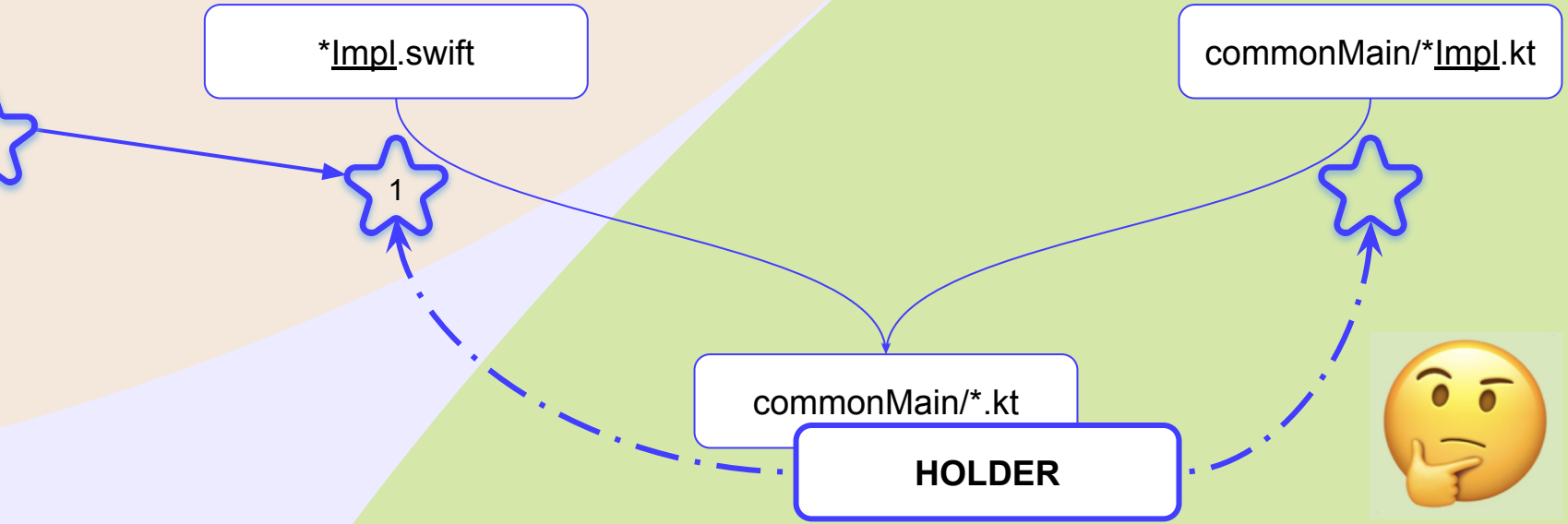
Kotlin/Native deep dive



Kotlin/Native deep dive



Kotlin/Native deep dive



managed by ARC ?

```
fun Any.isManagedByARC(): Boolean {
    val pointer = interpretCPointer<CPointed>(objcPtr())
    val originalRetainCount = CFGetRetainCount(pointer)

    CFRetain(pointer) // увеличиваем счётчик ссылок
    val modifiedRetainCount = CFGetRetainCount(pointer)
    CFRelease(pointer) // уменьшаем (возвращаем обратно) счётчик ссылок

    return originalRetainCount != modifiedRetainCount
}
```

Kotlin/Native deep dive

```
interface KotlinInterface
```

```
class Swift_KInterface: KotlinInterface {}
```

```
abstract class KotlinAbstractClass
```

```
class Swift_KAbstractClass: KotlinAbstractClass {...}
```

```
class Swift_Class {}
```

1. isManagedByARC()
2. inheritance hierarchy

Swift

```
inspect(obj: Swift_KInterface())
inspect(obj: Swift_KAbstractClass())
inspect(obj: Swift_Class())
inspect(obj: 123)
```

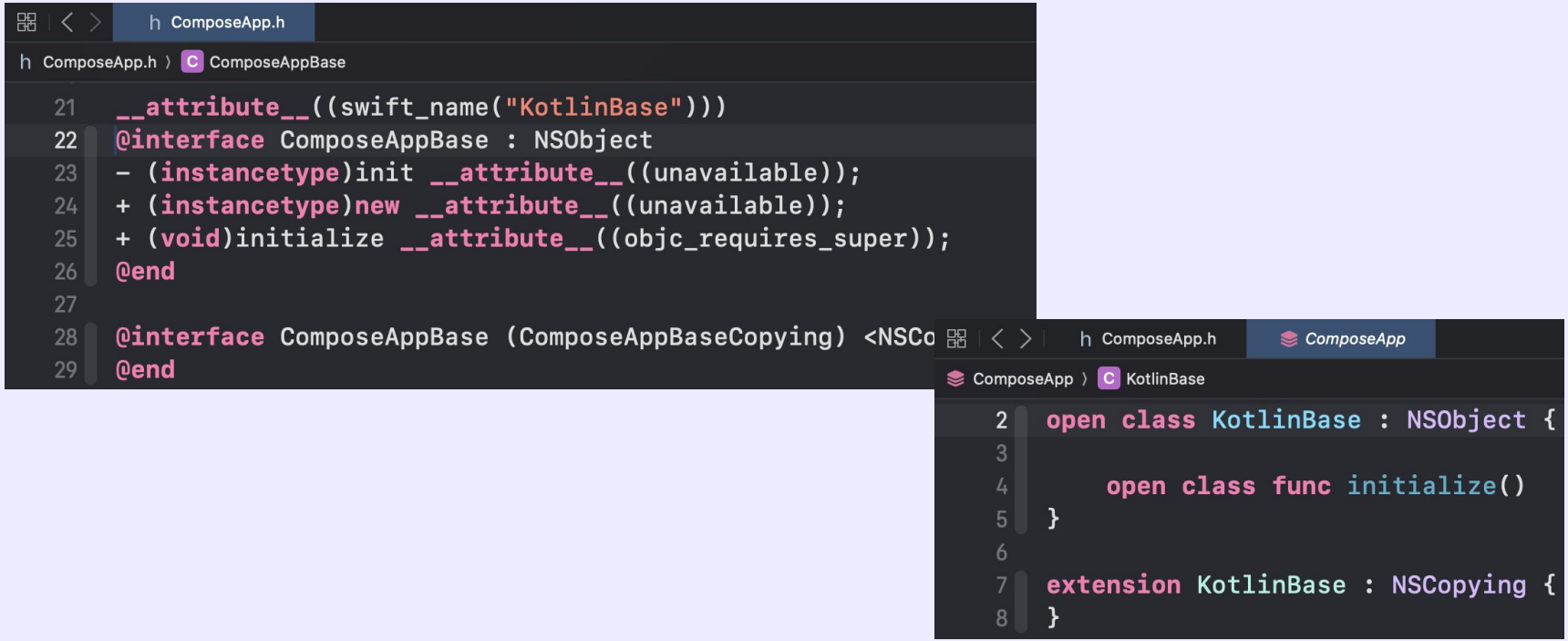
```
[TestApp.Swift_KInterface] is managed by ARC [true]
TestApp.Swift_KInterface
_TtCs12_SwiftObject
--------------------------
[TestApp.Swift_KAbstractClass@3f70058] is managed by ARC [false]
TestApp.Swift_KAbstractClass
ComposeAppKotlinAbstractClass
ComposeAppBase
NSObject
--------------------------
[TestApp.Swift_Class] is managed by ARC [true]
TestApp.Swift_Class
_TtCs12_SwiftObject
--------------------------
[123] is managed by ARC [true]
ComposeAppLong
ComposeAppNumber
NSNumber
NSValue
NSObject
--------------------------
```

KotlinBase

```
inspect(obj: Swift_KInterface())
inspect(obj: Swift_KAbstractClass())
inspect(obj: Swift_Class())
inspect(obj: 123)
```

```
[TestApp.Swift_KInterface] is managed by ARC [true]
TestApp.Swift_KInterface
_TtCs12_SwiftObject
--------------------------
[TestApp.Swift_KAbstractClass@3f70058] is managed by ARC [false]
TestApp.Swift_KAbstractClass
ComposeAppKotlinAbstractClass
ComposeAppBase
NSObject
--------------------------
[TestApp.Swift_Class] is managed by ARC [true]
TestApp.Swift_Class
_TtCs12_SwiftObject
--------------------------
[123] is managed by ARC [true]
ComposeAppLong
ComposeAppNumber
NSNumber
NSValue
NSObject
--------------------------
```

KotlinBase


ComposeApp.h
ComposeApp.h ⟩ ComposeAppBase
21 __attribute__((swift_name("KotlinBase")))
22 @interface ComposeAppBase : NSObject
23 - (instancetype)init __attribute__((unavailable));
24 + (instancetype)new __attribute__((unavailable));
25 + (void)initialize __attribute__((objc_requires_super));
26 @end
27
28 @interface ComposeAppBase (ComposeAppBaseCopying) <NSCo
29 @end
ComposeApp.h
ComposeApp
ComposeApp ⟩ KotlinBase
2 open class KotlinBase : NSObject {
3
4 open class func initialize()
5 }
6
7 extension KotlinBase : NSCopying {
8 }

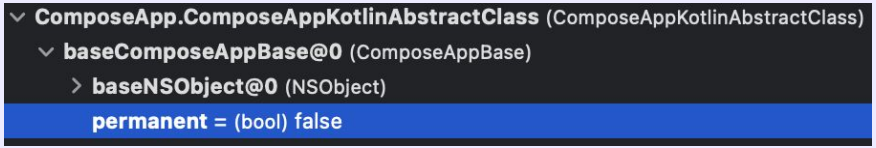
KotlinBase.h

```
@interface KotlinBase : NSObject <NSCopying>

+ (instancetype)createRetainedWrapper:(struct ObjHeader *)obj;

// Given kotlin.native.internal.ref.ExternalRCRef `ref`:
// * if it's already bound to another `KotlinBase` instance, replaces `self` with that instance
// * otherwise:
//   * find the best-fitting Obj-C class corresponding to `ref`'s Kotlin class
//   * construct its instance and replace `self`
// The code panics if the determined best-fitting class is not a subclass of `self`'s type.
// This situation happens if there's some unexported Swift class inheriting from an exported
// open class: this is not currently supported.
- (instancetype)initWithExternalRCRef:(uintptr_t)ref NS_REFINED_FOR_SWIFT;

// Return kotlin.native.internal.ref.ExternalRCRef stored in this class
- (uintptr_t)externalRCRef NS_REFINED_FOR_SWIFT;

@end
```

ObjCExportClasses.mm

```
-(instancetype)retain {
  if (permanent) {
    [super retain];
  } else {
    refHolder.addRef();
  }
  return self;
}
```

```
-(oneway void)release {
  if (permanent) {
    [super release];
  } else {
    refHolder.releaseRef();
  }
}
```



Kotlin/Native deep dive

```
interface KotlinInterface
```

```
abstract class KotlinAbstractClass
```

```
class KotlinNSObject: NSObject()
```

Kotlin

```
inspect(object : KotlinInterface {})
inspect(object : KotlinAbstractClass() {})
inspect(KotlinNSObject())
```

```
[tryKotlinPart$1@6094028] is managed by ARC [false]
ComposeApp_kobjcc0
ComposeAppBase
 > ivar[0] name:'refHolder'
 > ivar[1] name:'permanent'
NSObject
 > ivar[0] name:'isa'
-------------------------
[tryKotlinPart$2@6094068] is managed by ARC [false]
ComposeApp_kobjcc1
ComposeAppKotlinAbstractClass
ComposeAppBase
 > ivar[0] name:'refHolder'
 > ivar[1] name:'permanent'
NSObject
 > ivar[0] name:'isa'
-------------------------
[<ComposeAppKotlinNSObject0: 0x3019d0720>] is managed by ARC [false]
ComposeAppKotlinNSObject0
 > ivar[0] name:'kotlinBody'
NSObject
 > ivar[0] name:'isa'
-------------------------
```

Kotlin

```
inspect(object : KotlinInterface {})
inspect(object : KotlinAbstractClass() {})
inspect(KotlinNSObject())
```

```
[tryKotlinPart$1@6094028] is managed by ARC [false]
ComposeApp_kobjcc0
ComposeAppBase
 > ivar[0] name:'refHolder'
 > ivar[1] name:'permanent'
NSObject
 > ivar[0] name:'isa'
-------------------------
[tryKotlinPart$2@6094068] is managed by ARC [false]
ComposeApp_kobjcc1
ComposeAppKotlinAbstractClass
ComposeAppBase
 > ivar[0] name:'refHolder'
 > ivar[1] name:'permanent'
NSObject
 > ivar[0] name:'isa'
-------------------------
[<ComposeAppKotlinNSObject0: 0x3019d0720>] is managed by ARC [false]
ComposeAppKotlinNSObject0
 > ivar[0] name:'kotlinBody'
NSObject
 > ivar[0] name:'isa'
-------------------------
```

Kotlin

```
class KotlinNSObject: NSObject()
```

```
open class KotlinNSObject: NSObject()
```

Non-final Kotlin subclasses of Objective-C classes are not yet supported

```
class KotlinNSObject: NSObject(), KotlinInterface
```

Mixing Kotlin and Objective-C supertypes is not supported

ObjCInterop.mm

```
void* CreateKotlinObjCClass(const KotlinObjCClassInfo* info) {
    …
    Class newClass = allocateClass(info);
    …
    AddNSObjectOverride(false, newClass, @selector(retain), (void*)&retainImp);
    AddNSObjectOverride(false, newClass, @selector(release), (void*)&releaseImp);
    …
    return newClass;
}
```

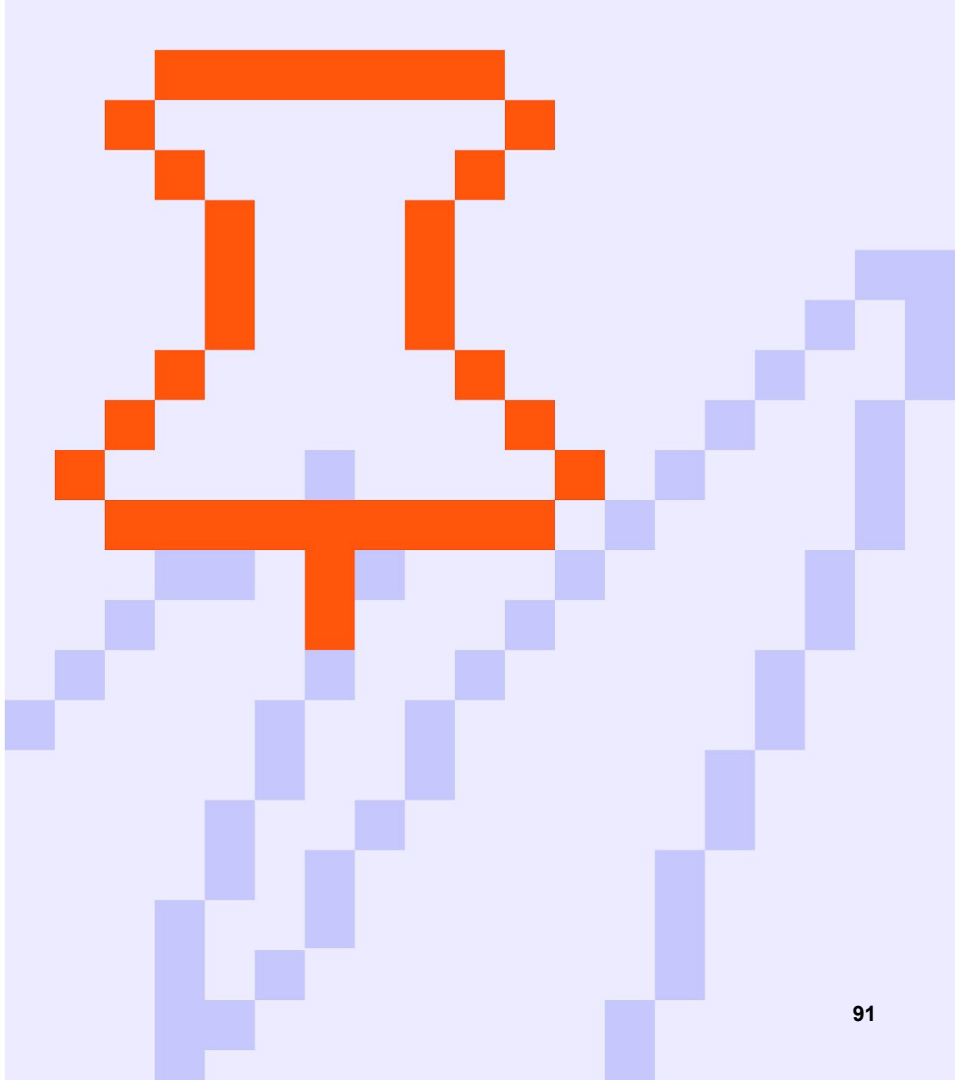
ObjCInterop.mm

```
id retainImp(id self, SEL _cmd) {
  getBackRef(self)->addRef();
  return self;
}
```

```
void releaseImp(id self, SEL _cmd) {
  getBackRef(self)->releaseRef();
}
```

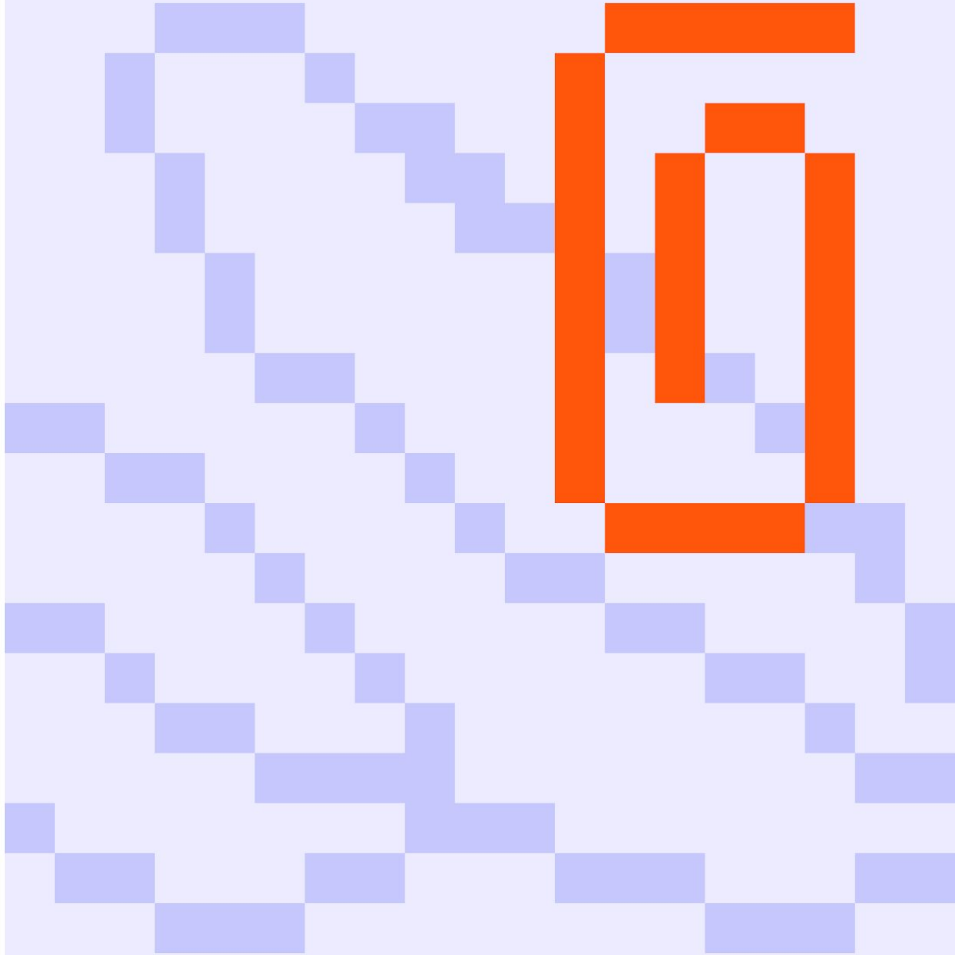
Вывод #3

1. Недокументированное поведение - это интересно, но багоопасно и, потенциально, нестабильно.
2. Пишите тесты!!!



Лимит памяти

- физические параметры
- ограничения ОС
- варианты ослабления ограничений



Android

ActivityManager				Runtime		
		MemoryInfo				
memoryClass	largeMemoryClass	availMem	totalMem	maxMemory	totalMemory	freeMemory

```
<application
    ...
    android:largeHeap="true">
```

*android:largeHeap

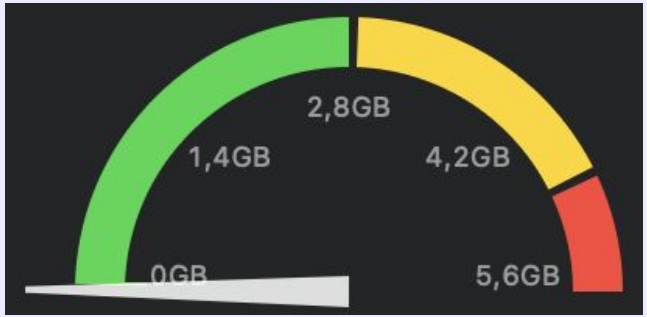
Android

ComponentCallbacks2.onTrimMemory(int)

iOS

os_proc_available_memory	task_info / task_vm_info	ProcessInfo
		physicalMemory

iOS



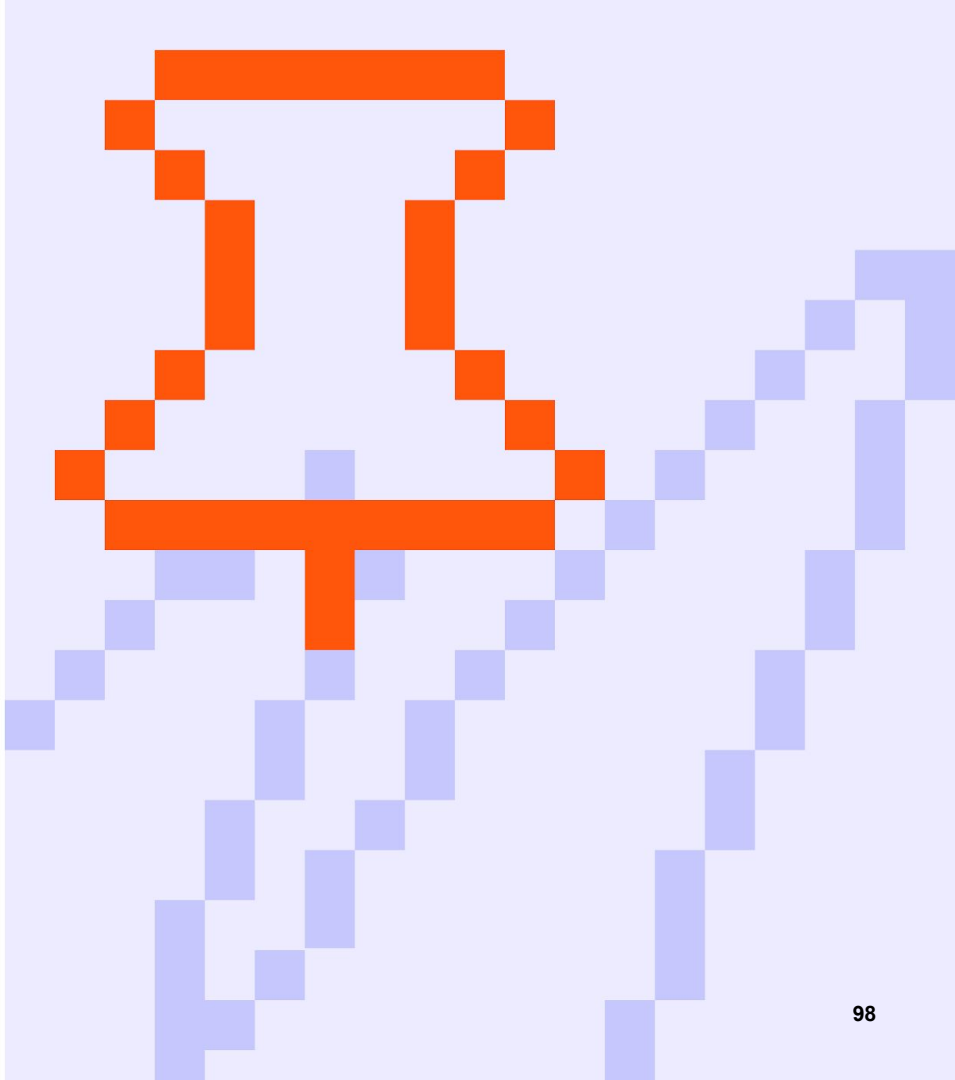
com.apple.developer.kernel.increased-memory-limit

iOS

- UIKit
  - applicationDidReceiveMemoryWarning
  - didReceiveMemoryWarning
- NotificationCenter / didReceiveMemoryWarningNotification
- DispatchQueue / DISPATCH_SOURCE_TYPE_MEMORYPRESSURE

Итоги

1. Kotlin упрощает церемонии.
2. Kotlin Multiplatform is Stable 🥳
3. Compose Multiplatform 😉



Спасибо за внимание

Спасибо за внимание

Kotlin

Бонусная подборка

1. Видео
   1.1. Kotlin Multiplatform Alchemy: Making Gold out of Your Swift Interop | Pamela Hill
   1.2. Святослав Щербина — Kotlin для написания общего кода под Android и iOS
   1.3. KotlinConf 2018 - Kotlin/Native Concurrency Model by Nikolay Igotti
   1.4. iOS Memory Management (Part 1): ARC, MRC, Autorelease, Object deallocation.
   1.5. Analyze heap memory - WWDC24
2. Почитать
   2.1. Integration with Swift/Objective-C ARC
   2.2. Weak References
3. Tasks
   3.1. Kotlin/Native DetachedObjectGraph
   3.2. Kotlin/Native Support direct interoperability with Swift
   3.3. Kotlin/Native WeakReference
   3.4. Kotlin/Native Unify SpecialRef handling
   3.5. Swift Thread safety for weak references

```
@OptIn(BetaInteropApi::class)
fun inspect(obj: Any) {
    println("[$obj] is managed by ARC [ ${obj.isManagedByARC()}]")

    val objCClass = object_getClass(obj)!!
    printHierarchy(obj, objCClass)
    println("--------------------------" )
}

@OptIn(BetaInteropApi::class, ExperimentalForeignApi::class)
private fun printHierarchy(obj: Any, objCls: ObjCClass) {
    println("${class_getName(objCls)?.toKString()}")

    memScoped {
        val countValue: CValue<UIntVarOf<UInt>> =  cValue()
        val countPtr = countValue.getPointer( this)
        class_copyIvarList(objCls, countPtr)?.let { ivars ->
            val count = countPtr.pointed.value
            for (i in 0 until count.toInt()) {
                val ivar = ivars[i]!!
                val ivarName = ivar_getName(ivar)?.toKString()
                println(" > ivar[$i] name:'$ivarName'")
            }
        }
    }

    val objSuperCls = class_getSuperclass(objCls)
    if (objSuperCls != null) {
        printHierarchy(obj, objSuperCls)
    }
}
```