

Яндекс




Оптимизация времени запуска iOS-приложений

Николай Лихогруд

Руководитель группы разработки Яндекс.Карт для iOS

Актуальность





It's really clear that the
most precious resource
we all have is time

Steve Jobs

Зачем сокращать время запуска?

- | Важное конкурентное преимущество
- | Влияет на retention, настроение пользователей, оценку в сторре
- | Максимальное время запуска 20s, при превышении система останавливает загрузку
 - › Актуально для слабых устройств

Актуальность

| Все больше приложений сталкивается с проблемой длительной загрузки из-за невозможности компиляции swift в статические библиотеки

Актуальность

| WWDC 2016: Optimizing App Startup Time

- › Apple впервые явно обозначила проблему, раскрыла технические детали, дала возможность профилирования работы системного загрузчика

План

1. Замеры времени запуска
2. Оптимизация pre-main
3. Оптимизация after-main
4. Контроль результата

1. Замеры времени запуска



Что есть время запуска?

Время от нажатия пользователем на иконку приложения до момента, когда пользователь может пользоваться приложением

- › загрузка UI
- › загрузка данных, обновление UI с полученными данными
- › ... (зависит от приложения)

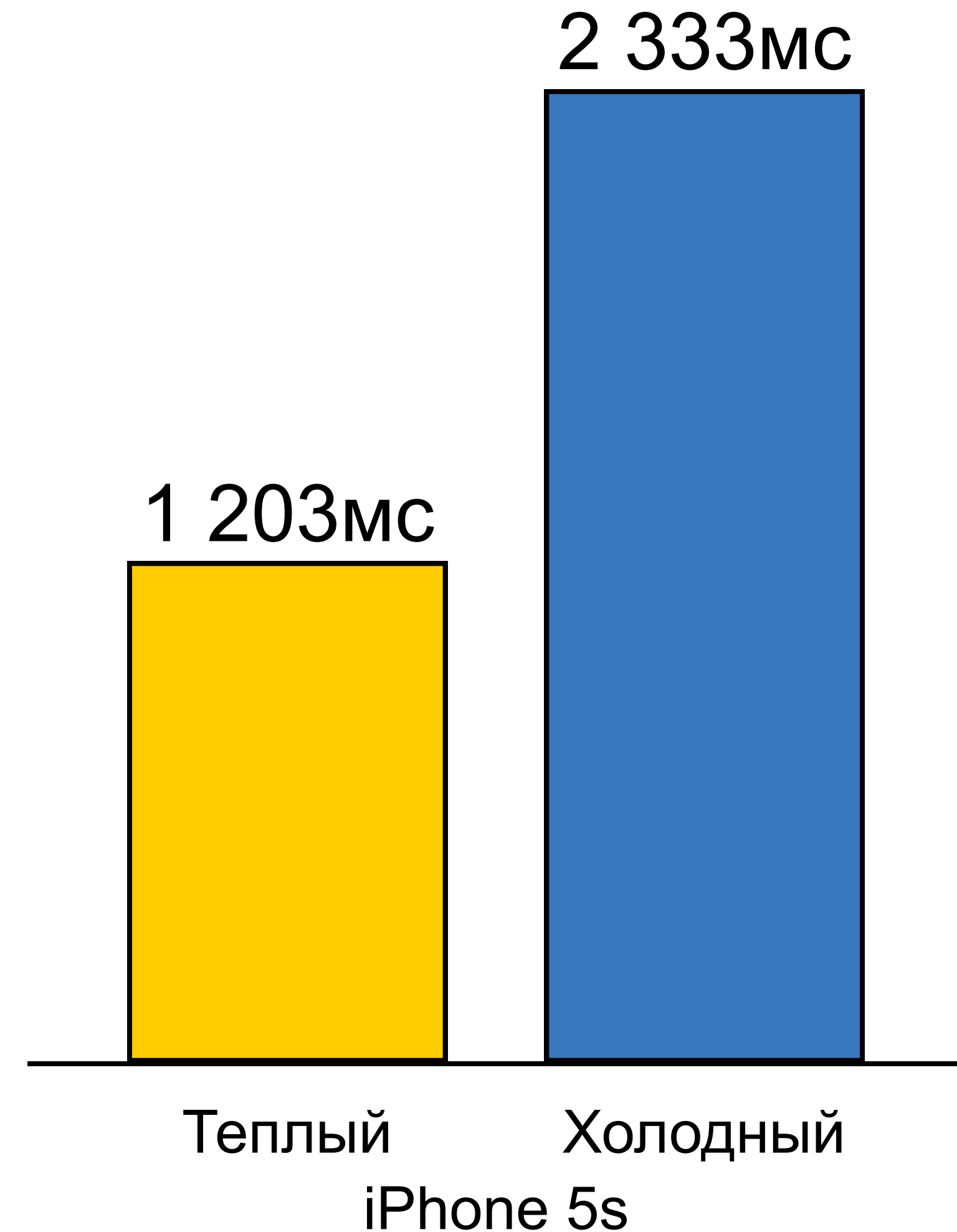
Холодный и теплый запуски

Холодный - нет образа приложения в кеше ядра iOS

› Давно не запускалось или не запускалось после перезагрузки

Теплый - есть образ приложения в кеше ядра iOS

› Недавно запускалось

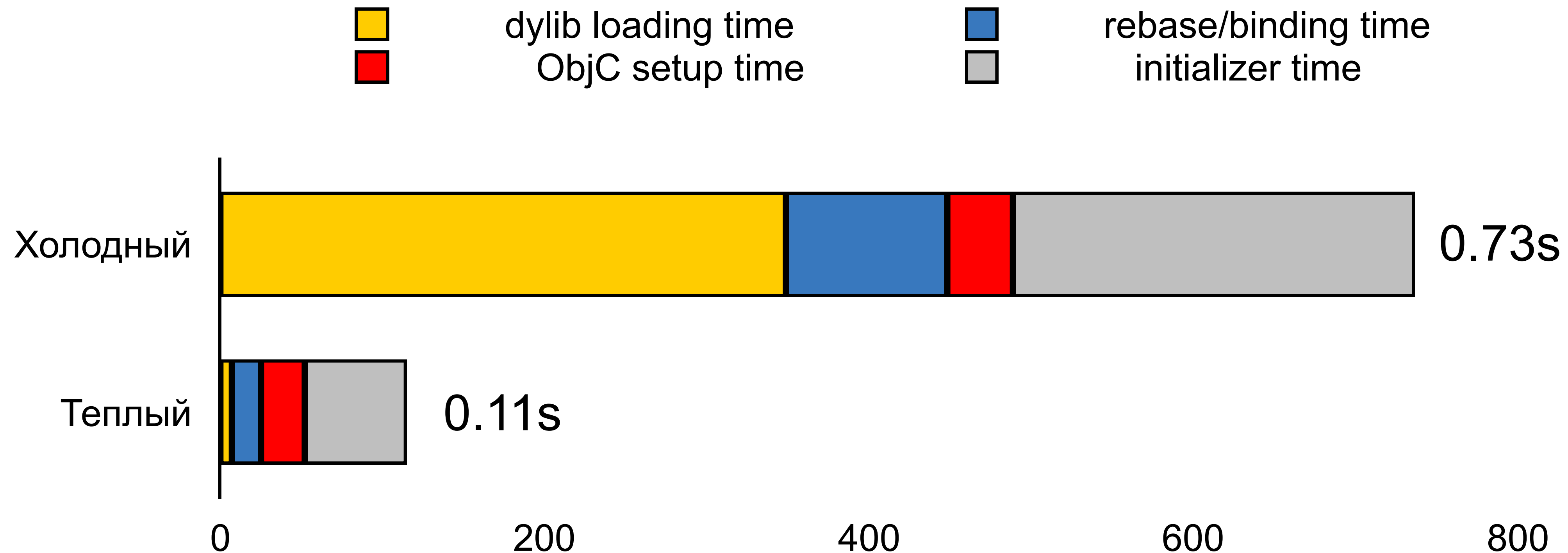


pre-main

До начала выполнения функции `main` система выполняет подготовку образа приложения в памяти:

- › загрузка динамических библиотек
- › исправление указателей, связывание
- › создание Objective-C контекста
- › вызов `+load` и конструкторов глобальных переменных `c++`

Холодный запуск и pre-main



Что замерять?

| Время от нажатия пользователем на иконку приложения до момента, когда пользователь может пользоваться приложением

- › Учитывать pre-main
- › Учитывать холодный запуск

3amep pre-main

DYLD_PRINT_STATISTICS

Empty > Generic iOS Device

Build
1 target

Run
Release

Test
Debug

Profile
Release

Analyze
Debug

Archive
Release

InfoArgumentsOptionsDiagnostics

Arguments Passed On Launch

No Arguments

Environment Variables

	Name	Value
<input checked="" type="checkbox"/>	DYLD_PRINT_STATISTICS	1

+ -

Expand Variables Based On Empty

Duplicate Scheme

Manage Schemes...

☐ Shared

Close

DYLD_PRINT_STATISTICS

```
Total pre-main time: 677.82 milliseconds (100.0%)
  dylib loading time: 520.33 milliseconds (76.7%)
  rebase/binding time: 54.31 milliseconds (8.0%)
  objc setup time: 43.16 milliseconds (6.3%)
  initializer time: 59.99 milliseconds (8.8%)
  slowest intializers :
    libSystem.B.dylib : 4.35 milliseconds (0.6%)
    TestApp : 74.83 milliseconds (11.0%)
```


DYLD_PRINT_STATISTICS

Total pre-main time: 677.82 milliseconds (100.0%)

dylib loading time: 520.33 milliseconds (76.7%)

rebase/binding time: 54.31 milliseconds (8.0%)

ObjC setup time: 43.16 milliseconds (6.3%)

initializer time: 59.99 milliseconds (8.8%)

slowest intializers :

libSystem.B.dylib : 4.35 milliseconds (0.6%)

TestApp : 74.83 milliseconds (11.0%)

DYLD_PRINT_STATISTICS

Total pre-main time: 677.82 milliseconds (100.0%)

dylib loading time: 520.33 milliseconds (76.7%)

rebase/binding time: 54.31 milliseconds (8.0%)

ObjC setup time: 43.16 milliseconds (6.3%)

initializer time: 59.99 milliseconds (8.8%)

slowest intializers :

libSystem.B.dylib : 4.35 milliseconds (0.6%)

TestApp : 74.83 milliseconds (11.0%)

DYLD_PRINT_STATISTICS

Total pre-main time: 677.82 milliseconds (100.0%)

dylib loading time: 520.33 milliseconds (76.7%)

rebase/binding time: 54.31 milliseconds (8.0%)

ObjC setup time: 43.16 milliseconds (6.3%)

initializer time: 59.99 milliseconds (8.8%)

slowest intializers :

libSystem.B.dylib : 4.35 milliseconds (0.6%)

TestApp : 74.83 milliseconds (11.0%)

DYLD_PRINT_STATISTICS

```
Total pre-main time: 677.82 milliseconds (100.0%)
  dylib loading time: 520.33 milliseconds (76.7%)
  rebase/binding time: 54.31 milliseconds (8.0%)
  ObjC setup time: 43.16 milliseconds (6.3%)
  initializer time: 59.99 milliseconds (8.8%)
  slowest intializers :
    libSystem.B.dylib : 4.35 milliseconds (0.6%)
    TestApp : 74.83 milliseconds (11.0%)
```

DYLD_PRINT_STATISTICS

Total pre-main time: 677.82 milliseconds (100.0%)

dylib loading time: 520.33 milliseconds (76.7%)

rebase/binding time: 54.31 milliseconds (8.0%)

ObjC setup time: 43.16 milliseconds (6.3%)

initializer time: 59.99 milliseconds (8.8%)

slowest intializers :

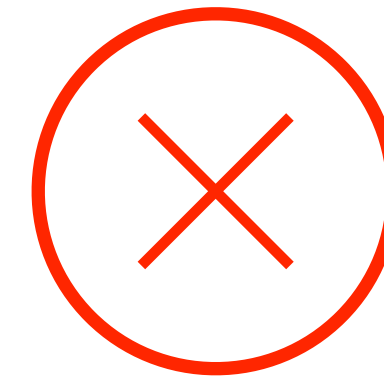
libSystem.B.dylib : 4.35 milliseconds (0.6%)

TestApp : 74.83 milliseconds (11.0%)

Замер after-main

| с начала didFinishLaunching

› не учитывается время создания UIApplication,
UIApplicationDelegate



| с начала main



Замер after-main

- | Добавляем в проект main.swift
- | В первой строке замеряем время

```
let start = Date().timeIntervalSince1970
```

```
let argc = Int(CommandLine.argc)
fileprivate let argv =
    UnsafeMutableRawPointer(CommandLine.unsafeArgv)
    .bindMemory(to: UnsafeMutablePointer<Int8>.self, capacity: argc)

UIApplicationMain(CommandLine.argc, argv, nil, NSStringFromClass(AppDelegate.self))
```

Замер after-main

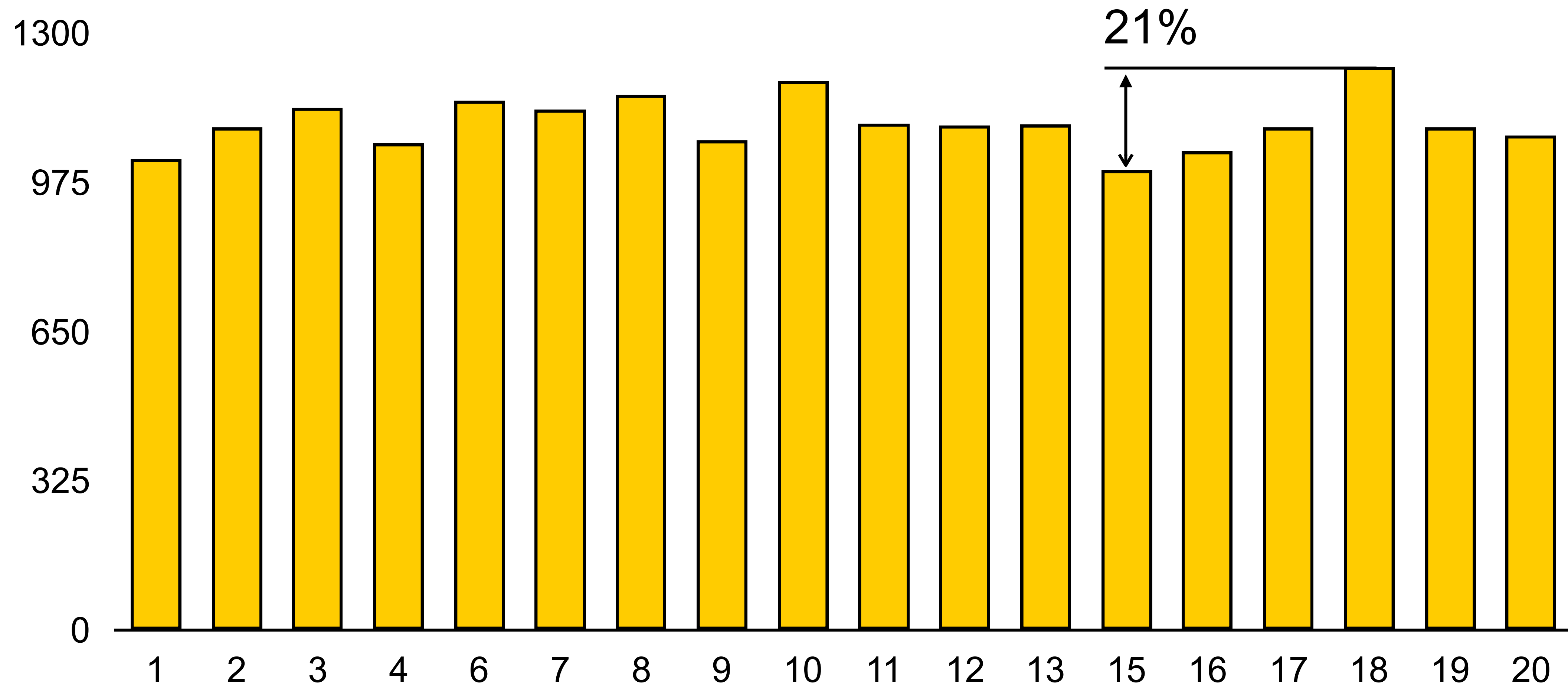
- | Добавляем в проект main.swift
- | В первой строке замеряем время

```
let start = Date().timeIntervalSince1970

let argc = Int(CommandLine.argc)
fileprivate let argv =
    UnsafeMutableRawPointer(CommandLine.unsafeArgv)
    .bindMemory(to: UnsafeMutablePointer<Int8>.self, capacity: argc)

UIApplicationMain(CommandLine.argc, argv, nil, NSStringFromClass(AppDelegate.self))
```


Важность множественных запусков



Автоматические запуски

| libimobiledevice

- › прямое взаимодействие с устройством
- › не требует jailbreak

Автоматические запуски

libimobiledevice

```
$idevice_id -l
```

```
$ideviceinstaller -u e1bbc7f0353933938929c3279fc6b7f51fac1c02 -i SomeApp.app
```

```
$idevicedebug -u e1bbc7f0353933938929c3279fc6b7f51fac1c02 run com.some.app
```

```
$idevicediagnostics restart
```

Автоматические запуски

libimobiledevice

```
$idevice_id -l
```

```
$ideviceinstaller -u e1bbc7f0353933938929c3279fc6b7f51fac1c02 -i SomeApp.app
```

```
$idevicedebug -u e1bbc7f0353933938929c3279fc6b7f51fac1c02 run com.some.app
```

```
$idevicediagnostics restart
```

Автоматические запуски

libimobiledevice

```
$idevice_id -l
```

```
$ideviceinstaller -u e1bbc7f0353933938929c3279fc6b7f51fac1c02 -i SomeApp.app
```

```
$idevicedebug -u e1bbc7f0353933938929c3279fc6b7f51fac1c02 run com.some.app
```

```
$idevicediagnostics restart
```

Автоматические запуски

libimobiledevice

```
$idevice_id -l
```

```
$ideviceinstaller -u e1bbc7f0353933938929c3279fc6b7f51fac1c02 -i SomeApp.app
```

```
$idevicedebug -u e1bbc7f0353933938929c3279fc6b7f51fac1c02 run com.some.app
```

```
$idevicediagnostics restart
```

Автоматические запуски

libimobiledevice + DYLD_PRINT_STATISTICS

```
$idevicedebug -e DYLD_PRINT_STATISTICS=1 \  
              -u e1bbc7f0353933938929c3279fc6b7f51fac1c02 run test.app
```

```
Total pre-main time: 52.85 milliseconds (100.0%)  
    dylib loading time: 7.21 milliseconds (13.6%)  
    rebase/binding time: 3.22 milliseconds (6.1%)
```

...

Сборка для тестов

Релизная конфигурация с оптимизациями

- › отключены ассерты и функциональность для дебага
- › не добавляется libswiftSwiftOnoneSupport.dylib

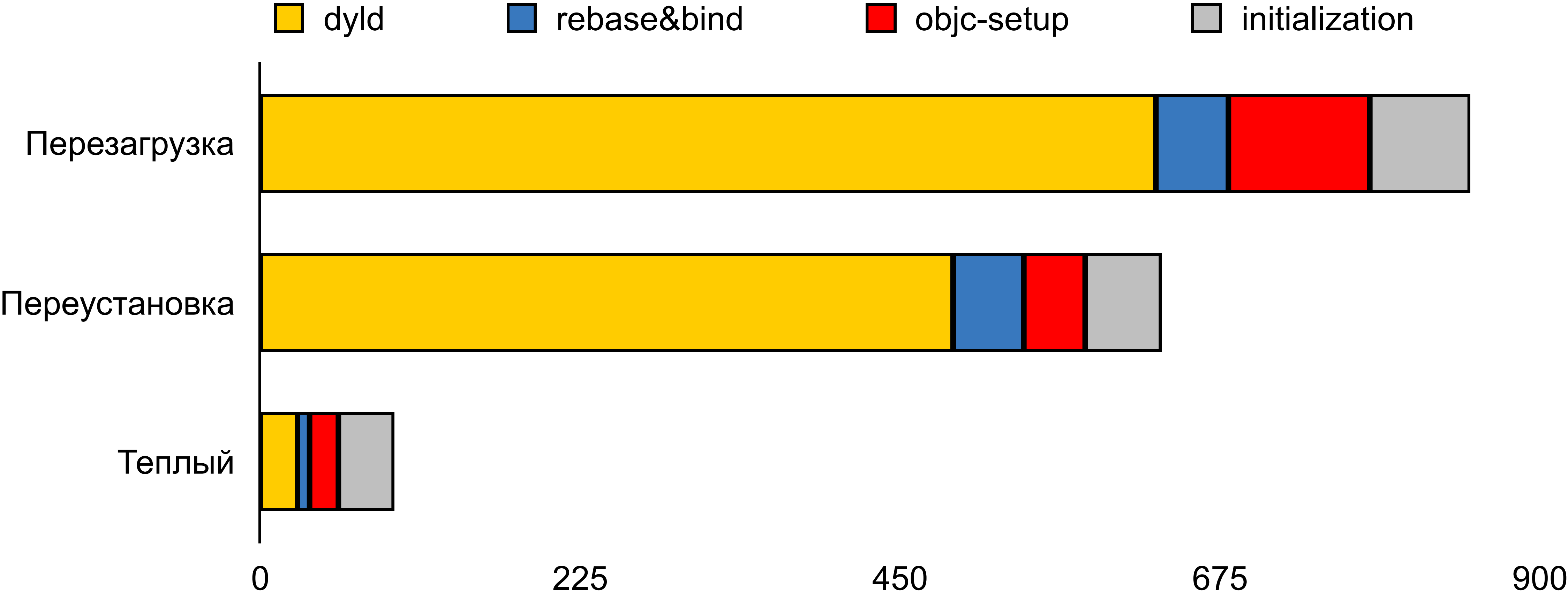
`exit(0)` после завершения загрузки, если
`processInfo.environment["DYLD_PRINT_STATISTICS"] != nil`

Организация скрипта

| На каждой итерации:

1. `idevicediagnostics restart`, дождаться загрузки
2. `idevicedebug run` - **холодный запуск**
3. `idevicedebug run` - **теплый запуск**
4. Обработать вывод, сохранить лог

pre-main: перезагрузка vs переустановка

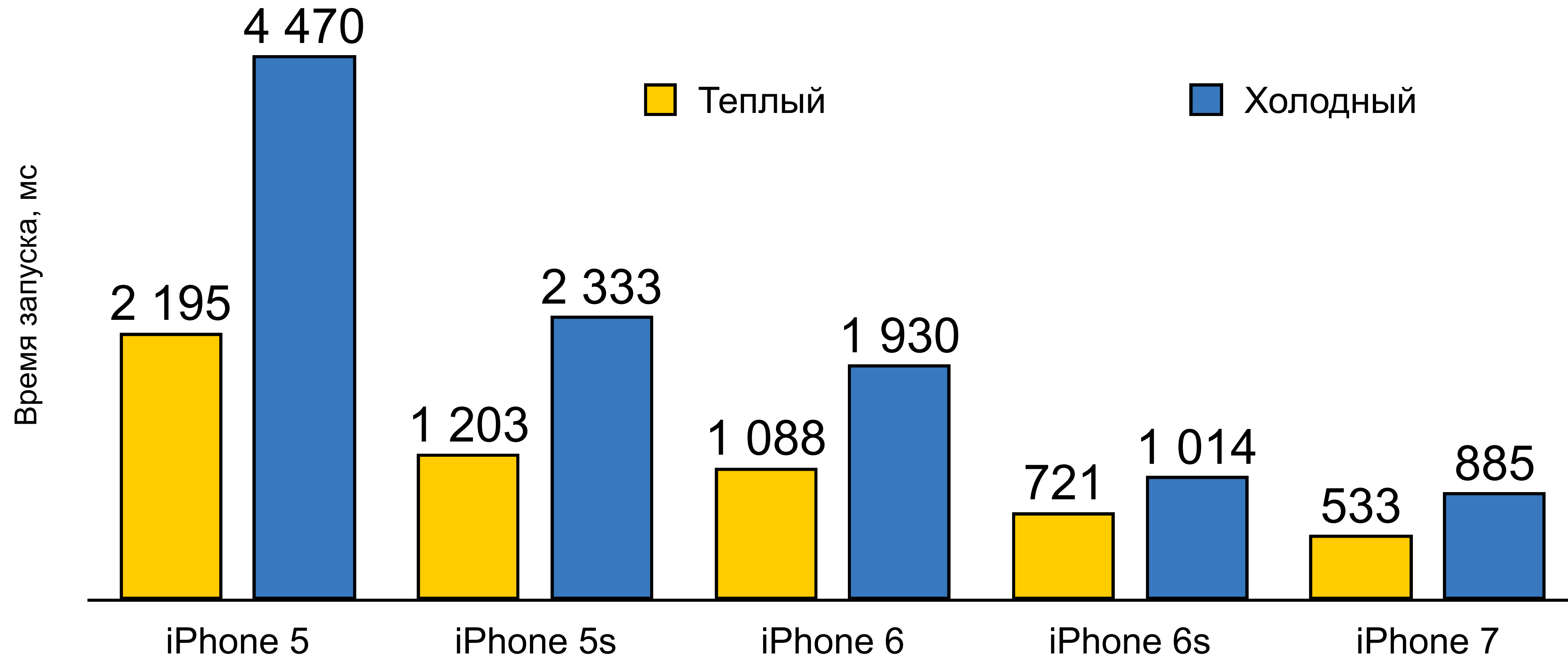


Организация скрипта

| На каждой итерации:

1. `ideviceinstaller -i`
2. `idevicedebug run` - **холодный запуск**
3. `idevicedebug run` - **теплый запуск**
4. Обработать вывод, сохранить лог

Запуски на разных устройствах



Пустой проект на swift



Пустой проект: Objective-C, iPhone 5s

Total pre-main time: 19.40 milliseconds (100.0%)

dylib loading time: 0.72 milliseconds (3.7%)

rebase/binding time: 1.14 milliseconds (5.8%)

ObjC setup time: 5.02 milliseconds (25.9%)

initializer time: 12.51 milliseconds (64.4%)

slowest intializers :

libSystem.B.dylib : 10.27 milliseconds (52.9%)

CoreFoundation : 0.66 milliseconds (3.4%)

Пустой проект: swift, iPhone 5s

Total pre-main time: 232.19 milliseconds (100.0%)

dylib loading time: 209.33 milliseconds (90.1%)

rebase/binding time: 7.98 milliseconds (3.4%)

ObjC setup time: 8.49 milliseconds (3.6%)

initializer time: 6.37 milliseconds (2.7%)

slowest intializers :

libSystem.B.dylib : 3.14 milliseconds (1.3%)

Пустой проект: swift, iPhone 5

Total pre-main time: 837.72 milliseconds (100.0%)

dylib loading time: 789.01 milliseconds (94.1%) **3.5x!**

rebase/binding time: 15.41 milliseconds (1.8%)

ObjC setup time: 13.52 milliseconds (1.6%)

initializer time: 19.57 milliseconds (2.3%)

slowest intializers :

libSystem.B.dylib : 5.56 milliseconds (0.6%)

Список загружаемых библиотек

Empty > Generic iOS Device

▶ Build
1 target

▶ ▶ Run
Release

▶ ⚙️ Test
Debug

▶ 📊 Profile
Release

▶ 📄 Analyze
Debug

▶ 📦 Archive
Release

Info

Arguments

Options

Diagnostics

Runtime Sanitization
Requires recompilation

Memory Management

☐ Address Sanitizer

☐ Thread Sanitizer

☐ Pause on issues

☐ Malloc Scribble

☐ Malloc Guard Edges

Logging

☐ Malloc Stack

All Allocation and Free History ⌵

☐ Dynamic Linker API Usage

☒ Dynamic Library Loads

Duplicate Scheme

Manage Schemes...

☐ Shared

Close

Список загружаемых библиотек

- | Для проекта на Objective C загружается само приложение + 146 системных библиотек
- | Для проекта на swift дополнительно загружаются 9 библиотек из бандла приложения, т.н. swift standard libraries

...

```
dyld: loaded: /private/var/containers/Bundle/Application/  
AE358ED8-8FE3-4E90-88C8-FC602FDEA528/Empty.app/Frameworks/libswiftCore.dylib  
dyld: loaded: /private/var/containers/Bundle/Application/  
AE358ED8-8FE3-4E90-88C8-FC602FDEA528/Empty.app/Frameworks/libswiftUIKit.dylib
```

...

| Swift продолжает активно развиваться, не обеспечивает обратную совместимость и не является частью iOS

в отличие от ObjectiveC

Выводы

- | Загрузка системных библиотек оптимизирована
- | Библиотеки из бандла приложения, в т.ч. swift standard libraries, грузятся долго
- | Любое приложения на swift будет грузиться на iPhone 5 минимум секунду при холодном запуске
- | Со временем swift standard libraries станут частью системы и проблема исчезнет

2. Оптимизация pre-main



На что теоретически можно повлиять?

- | Уменьшить число загружаемых динамических библиотек
- | Уменьшить использование Objective-C:
 - › Использовать swift
- | Перенести код `+load` в `+initialize`
- | Избавиться от статических c++ переменных со сложными конструкторами

На что практически можно повлиять?*

- Уменьшить число загружаемых динамических библиотек
- Уменьшить размер бинарного файла приложения путем выноса символов в динамические библиотеки
 - › грузить лениво через dlopen
 - › Уменьшится rebase&bind, objc-setup

*в готовом проекте на swift

Тестовый: Podfile

use_frameworks! из-за подов на swift

много подов, поставляемых исходными файлами

...

use_frameworks!

```
target :OriginalApp do
    pod 'AlamofireObjectMapper'
    pod 'AlamofireImage'
    pod 'FacebookLogin'
    pod 'ObjectMapper'
    pod 'PhoneNumberKit'
    pod 'UIImageEffects'
    pod 'pop'
    pod 'KissXML'
    pod 'MTDates'
    pod 'Punycode-Cocoa'
    pod 'YandexSpeechKit'
    pod 'YandexMapKit'
end
```


Тестовый: Функциональность

```
func handleDirectionsRequest(_ request: MKDirectionsRequest) {  
    //... использует MapKit из iOS SDK  
}  
  
func presentMap() {  
    //... использует YandexMapKit  
}  
  
func startSpeechRecognition() {  
    //... использует YandexSpeechKit и AVFoundation  
}
```

Тестовый: Время запуска

Total pre-main time: 741.74 milliseconds (100.0%)

dylib loading time: 627.80 milliseconds (84.6%) 3x!!

rebase/binding time: 28.87 milliseconds (3.8%)

ObjC setup time: 36.00 milliseconds (4.8%)

initializer time: 49.04 milliseconds (6.6%)

slowest intializers :

libSystem.B.dylib : 4.57 milliseconds (0.6%)

OriginalApp : 28.50 milliseconds (3.8%)

Тестовый
































Пустой



Динамические библиотеки

Добавились для каждого пода, собираемого из исходных файлов

Добавились новые библиотеки swift standard libraries

- ▶  Alamofire.framework
- ▶  AlamofireImage.framework
- ▶  AlamofireObjectMapper.framework
- ▶  Bolts.framework
- ▶  FacebookCore.framework
- ▶  FacebookLogin.framework
- ▶  FBSDKCoreKit.framework
- ▶  FBSDKLoginKit.framework
- ▶  KissXML.framework
- ▶  MTDates.framework
- ▶  ObjectMapper.framework
- ▶  PhoneNumberKit.framework
- ▶  pop.framework
- ▶  Punycode_Cocoa.framework
- ▶  UIImageEffects.framework
-  libswiftAVFoundation.dylib
-  libswiftCore.dylib
-  libswiftCoreAudio.dylib
-  libswiftCoreGraphics.dylib
-  libswiftCoreImage.dylib
-  libswiftCoreLocation.dylib
-  libswiftCoreMedia.dylib
-  libswiftDarwin.dylib
-  libswiftDispatch.dylib
-  libswiftFoundation.dylib
-  libswiftMapKit.dylib
-  libswiftObjectiveC.dylib
-  libswiftQuartzCore.dylib
-  libswiftUIKit.dylib

cocoapods-amimono

```
$gem install cocoapods-amimono
```

| Плагин для cocoapods

| Патчит скрипты и xcconfig-и cocoapods:

- › Линкует объектные файлы, оставшиеся после сборки подов, непосредственно в бинарный файл приложения
- › Убирает линковку с библиотеками подов, убирает их встраивание

cocoapods-amimono

| Добавить plugin в Podfile

| Добавить post_install в Podfile

...

```
plugin 'cocoapods-amimono'  
use_frameworks!
```

```
target :StaticPodsApp do  
  pod 'AlamofireObjectMapper'  
  ...  
  pod 'YandexSpeechKit'  
  pod 'YandexMapKit'  
end
```

```
post_install do |installer|  
  require 'cocoapods-amimono/patcher'  
  Amimono::Patcher.patch!(installer)  
end
```

Тестовый: Время запуска

Total pre-main time: 741.74 milliseconds (100.0%)

dylib loading time: 627.80 milliseconds (84.6%) 3x!!

rebase/binding time: 28.87 milliseconds (3.8%)

ObjC setup time: 36.00 milliseconds (4.8%)

initializer time: 49.04 milliseconds (6.6%)

slowest intializers :

libSystem.B.dylib : 4.57 milliseconds (0.6%)

OriginalApp : 28.50 milliseconds (3.8%)

Тестовый



Пустой



После cocoapods-amimono

Total pre-main time: 413.13 milliseconds (100.0%)

dylib loading time: 320.20 milliseconds (77.5%)

rebase/binding time: 28.71 milliseconds (6.9%)

ObjC setup time: 21.40 milliseconds (5.1%)

initializer time: 42.80 milliseconds (10.3%)

slowest intializers :

libSystem.B.dylib : 4.68 milliseconds (1.1%)

StaticPodsApp : 39.18 milliseconds (9.4%)

amimono



Пустой



cocoarods-amimono: недостатки

- | Пропускает интеграцию подов, поставляемых framework-ами
 - › Их приходится добавлять вручную
- | Нет контроля какие поды вмержить в бинарный файл приложения, а какие нет
- | Пропускает таргеты, названия которых содержат "Test"

После cocoapods-amimono

Total pre-main time: 413.13 milliseconds (100.0%)

dylib loading time: 320.20 milliseconds (77.5%)

rebase/binding time: 28.71 milliseconds (6.9%)

ObjC setup time: 21.40 milliseconds (5.1%)

initializer time: 42.80 milliseconds (10.3%)

slowest intializers :

libSystem.B.dylib : 4.68 milliseconds (1.1%)

StaticPodsApp : 39.18 milliseconds (9.4%)

amimono



Пустой

















swift standard libraries

В пустом проекте

 libswiftCore.dylib
 libswiftCoreGraphics.dylib
 libswiftCoreImage.dylib
 libswiftDarwin.dylib
 libswiftDispatch.dylib
 libswiftFoundation.dylib
 libswiftObjectiveC.dylib
 libswiftQuartzCore.dylib
 libswiftUIKit.dylib

В тестовом проекте















 libswiftAVFoundation.dylib
 libswiftCore.dylib
 libswiftCoreAudio.dylib
 libswiftCoreGraphics.dylib
 libswiftCoreImage.dylib
 libswiftCoreLocation.dylib
 libswiftCoreMedia.dylib
 libswiftDarwin.dylib
 libswiftDispatch.dylib
 libswiftFoundation.dylib
 libswiftMapKit.dylib
 libswiftObjectiveC.dylib
 libswiftQuartzCore.dylib
 libswiftUIKit.dylib

swift standard libraries

В пустом проекте

 libswiftCore.dylib
 libswiftCoreGraphics.dylib
 libswiftCoreImage.dylib
 libswiftDarwin.dylib
 libswiftDispatch.dylib
 libswiftFoundation.dylib
 libswiftObjectiveC.dylib
 libswiftQuartzCore.dylib
 libswiftUIKit.dylib

В тестовом проекте


 libswiftAVFoundation.dylib
 libswiftCore.dylib
 libswiftCoreAudio.dylib
 libswiftCoreGraphics.dylib
 libswiftCoreImage.dylib
 libswiftCoreLocation.dylib
 libswiftCoreMedia.dylib
 libswiftDarwin.dylib
 libswiftDispatch.dylib
 libswiftFoundation.dylib
 libswiftMapKit.dylib
 libswiftObjectiveC.dylib
 libswiftQuartzCore.dylib
 libswiftUIKit.dylib

swift standard libraries

`import` CoreLocation в *.swift

`#import` <CoreLocation/CoreLocation.h> в bridging header

| Приводят к добавлению *libswiftCoreLocation.dylib* в бандл приложения



Некоторые библиотеки
SDK стоит
использовать только в
Objective C

*Пока библиотеки swift не станут частью системы

Objective C обертки

- | Обернуть CoreLocation в Objective-C с таким же интерфейсом, но другим префиксом

- › CLWLocationManger, CLWLocation, CLWHeading и т.д.

- | Использовать в swift только эти обертки

- › Тогда libswiftCoreLocation.dylib не добавляется

Objective C обертки

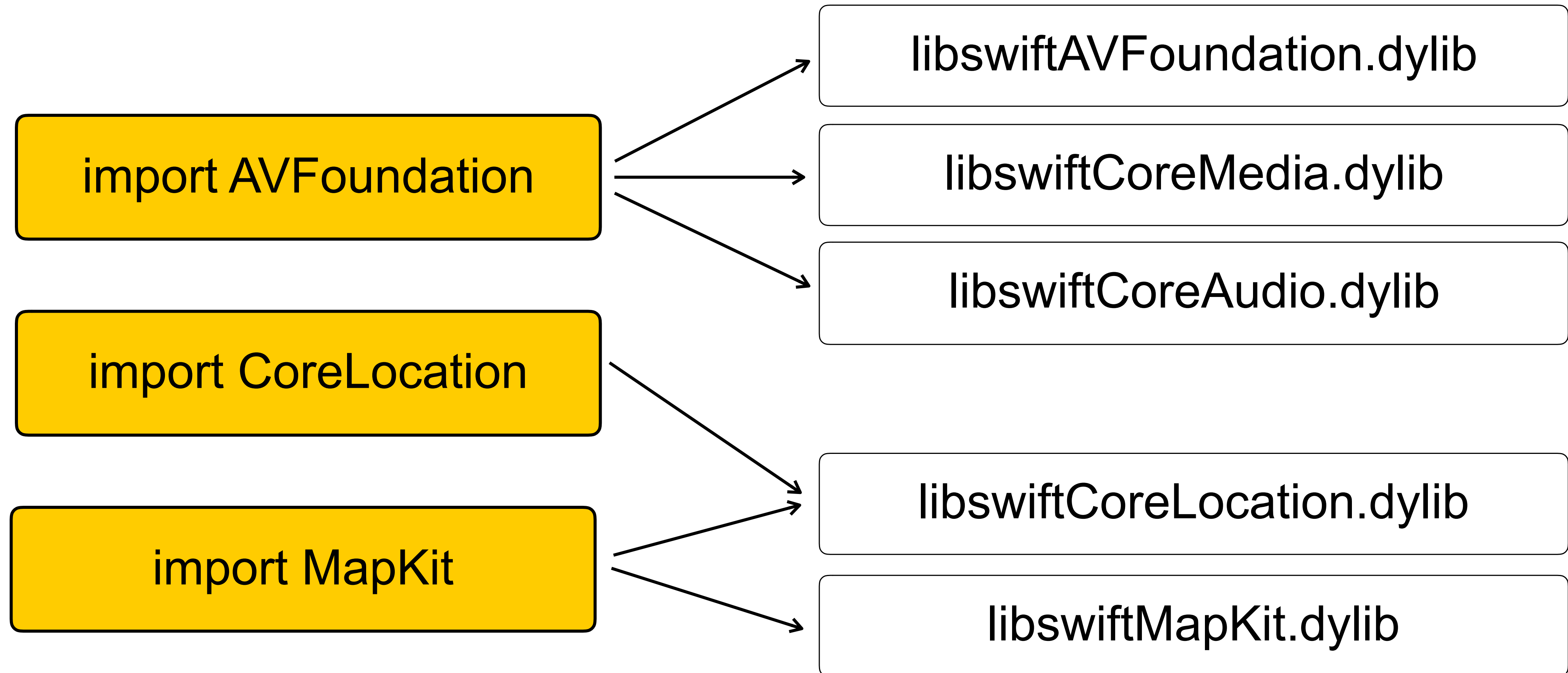
- CoreLocation может импортироваться в заголовочных файлах зависимостей

- › Их тоже нужно обернуть или использовать только в Objective C

- CoreLocation может импортироваться как зависимость других библиотек SDK

- › например, MapKit

swift standard libraries



После cocoapods-amimono

Total pre-main time: 413.13 milliseconds (100.0%)

dylib loading time: 320.20 milliseconds (77.5%)

rebase/binding time: 28.71 milliseconds (6.9%)

ObjC setup time: 21.40 milliseconds (5.1%)

initializer time: 42.80 milliseconds (10.3%)

slowest intializers :

libSystem.B.dylib : 4.68 milliseconds (1.1%)

StaticPodsApp : 39.18 milliseconds (9.4%)

amimono

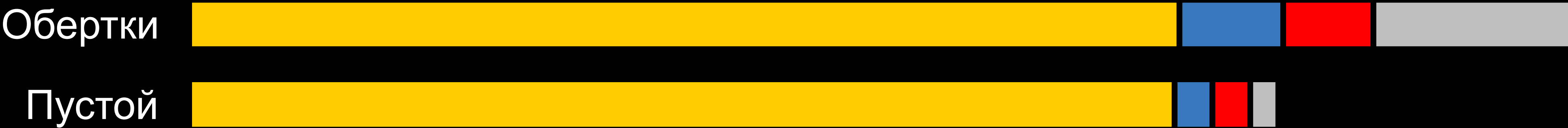


Пустой



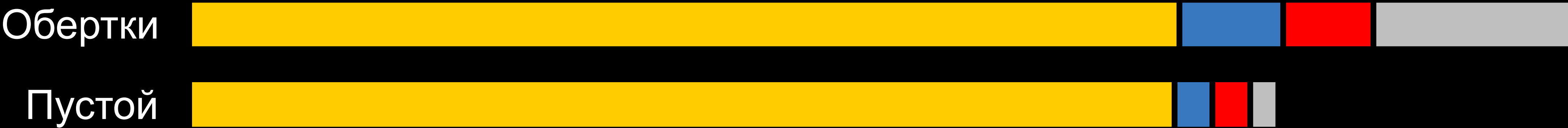
Objective C обертки


```
Total pre-main time: 294.30 milliseconds (100.0%)
  dylib loading time: 210.64 milliseconds (71.5%)
  rebase/binding time: 22.18 milliseconds (7.5%)
  ObjC setup time: 19.08 milliseconds (6.4%)
  initializer time: 42.38 milliseconds (14.4%)
  slowest intializers :
    libSystem.B.dylib : 4.32 milliseconds (1.4%)
    RemoveSwiftLibsApp : 39.59 milliseconds (13.4%)
```



Objective C обертки

```
Total pre-main time: 294.30 milliseconds (100.0%)
  dylib loading time: 210.64 milliseconds (71.5%)
  rebase/binding time: 22.18 milliseconds (7.5%)
  ObjC setup time: 19.08 milliseconds (6.4%)
  initializer time: 42.38 milliseconds (14.4%)
  slowest intializers :
    libSystem.B.dylib : 4.32 milliseconds (1.4%)
    RemoveSwiftLibsApp : 39.59 milliseconds (13.4%)
```





На старте должны
загружаться ТОЛЬКО
необходимые СИМВОЛЫ

Все остальное можно загружать лениво

Уменьшение размера бинарного файла

- | В рассматриваемом примере карта и распознавание звука не включаются на старте
 - › Можно конвертировать YandexSpeechKit и YandexMapKit в динамические библиотеки и лениво загружать через dlopen
- | Неизбежно уменьшит rebase&bind, obj startup, initialization

static lib -> dynamic lib

- | Создать отдельный таргет «Cocoa Touch Framework»
- | В созданный таргет
 - › Добавить *pod* со статической библиотекой в *Podfile*
 - › Прилинковать недостающие зависимости

static lib -> dynamic lib

Для основного таргета

- › Убрать *pod* со статической библиотекой в *Podfile*
- › Добавить новый framework в «embedded binaries»
- › Перенести ресурсы
- › Сделать Objective-C обертки для ленивой загрузки символов библиотеки

dlopen: загрузка библиотеки

```
#import <dlfcn.h>
```

```
NSString *frameworksPath = [[NSBundle mainBundle] privateFrameworksPath];
```

```
NSString *dyLib = @"SomeFramework.framework/SomeFramework";
```

```
NSString *path = [NSString stringWithFormat:@"%s/%s", frameworksPath, dyLib];
```

```
const char *pathStr = [path cStringUsingEncoding:NSUTF8StringEncoding];
```

```
void *handle = dlopen(pathStr, RTLD_LAZY);
```


dlsym: функции и глобальные переменные

```
void (*someFuncPtr)(int) = dlsym(handle, "SomeFunc");  
someFuncPtr(5);
```

```
NSString *__autoreleasing *someGlobalVarPtr =  
    (NSString *__autoreleasing *)dlsym(handle, "SomeGlobalVar");
```

```
NSLog(@"%@", *someGlobalVarPtr);
```

dlsym: функции и глобальные переменные

```
void (*someFuncPtr)(int) = dlsym(handle, "SomeFunc");  
someFuncPtr(5);
```

```
NSString *__autoreleasing *someGlobalVarPtr =  
    (NSString *__autoreleasing *)dlsym(handle, "SomeGlobalVar");
```

```
NSLog(@"%@@", *someGlobalVarPtr);
```

dlsym: функции и глобальные переменные

```
void (*someFuncPtr)(int) = dlsym(handle, "SomeFunc");  
someFuncPtr(5);
```

```
NSString *__autoreleasing *someGlobalVarPtr =  
    (NSString *__autoreleasing *)dlsym(handle, "SomeGlobalVar");
```

```
NSLog(@"%@", *someGlobalVarPtr);
```

dlsym: функции и глобальные переменные

```
void (*someFuncPtr)(int) = dlsym(handle, "SomeFunc");  
someFuncPtr(5);
```

```
NSString *__autoreleasing *someGlobalVarPtr =  
    (NSString *__autoreleasing *)dlsym(handle, "SomeGlobalVar");
```

```
NSLog(@"%@", *someGlobalVarPtr);
```

dlsym: классы

```
#import <SomeFramework/SomeFramework.h>
```

```
Class someClass = (__bridge Class)dlsym(handle, "OBJC_CLASS_$_SomeClass");  
SomeClass *someObj = [[someClass alloc] initWithParameter: param];  
[someObj someMethod];
```

dlsym: классы

```
#import <SomeFramework/SomeFramework.h>
```

```
Class someClass = (__bridge Class)dlsym(handle, "OBJC_CLASS_$_SomeClass");  
SomeClass *someObj = [[someClass alloc] initWithParameter: param];  
[someObj someMethod];
```

dlsym: классы

```
#import <SomeFramework/SomeFramework.h>
```

```
Class someClass = (__bridge Class)dlsym(handle, "OBJC_CLASS_$_SomeClass");  
SomeClass *someObj = [[someClass alloc] initWithParameter: param];  
[someObj someMethod];
```

dlsym: классы

```
#import <SomeFramework/SomeFramework.h>
```

```
Class someClass = (__bridge Class)dlsym(handle, "OBJC_CLASS_$_SomeClass");  
SomeClass *someObj = [[someClass alloc] initWithParameter: param];  
[someObj someMethod];
```

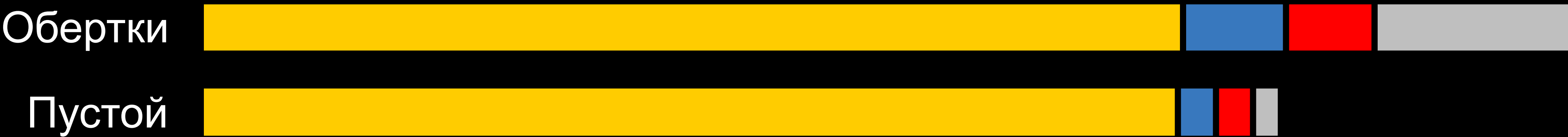

dlsym: классы

```
#import <SomeFramework/SomeFramework.h>
```

```
Class someClass = (__bridge Class)dlsym(handle, "OBJC_CLASS_$_SomeClass");  
SomeClass *someObj = [[someClass alloc] initWithParameter: param];  
[someObj someMethod];
```

Objective C обертки

```
Total pre-main time: 294.30 milliseconds (100.0%)
  dylib loading time: 210.64 milliseconds (71.5%)
  rebase/binding time: 22.18 milliseconds (7.5%)
    ObjC setup time: 19.08 milliseconds (6.4%)
  initializer time: 42.38 milliseconds (14.4%)
  slowest intializers :
    libSystem.B.dylib : 4.32 milliseconds (1.4%)
    RemoveSwiftLibsApp : 39.59 milliseconds (13.4%)
```



Уменьшение размера бинарного файла

Total pre-main time: 263.41 milliseconds (100.0%)

dylib loading time: 210.72 milliseconds (79.9%)

rebase/binding time: 16.89 milliseconds (6.4%)

ObjC setup time: 17.47 milliseconds (6.6%)

initializer time: 18.32 milliseconds (6.9%)

slowest intializers :

libSystem.B.dylib : 4.18 milliseconds (1.5%)

LazyLibs : 12.19 milliseconds (4.6%)

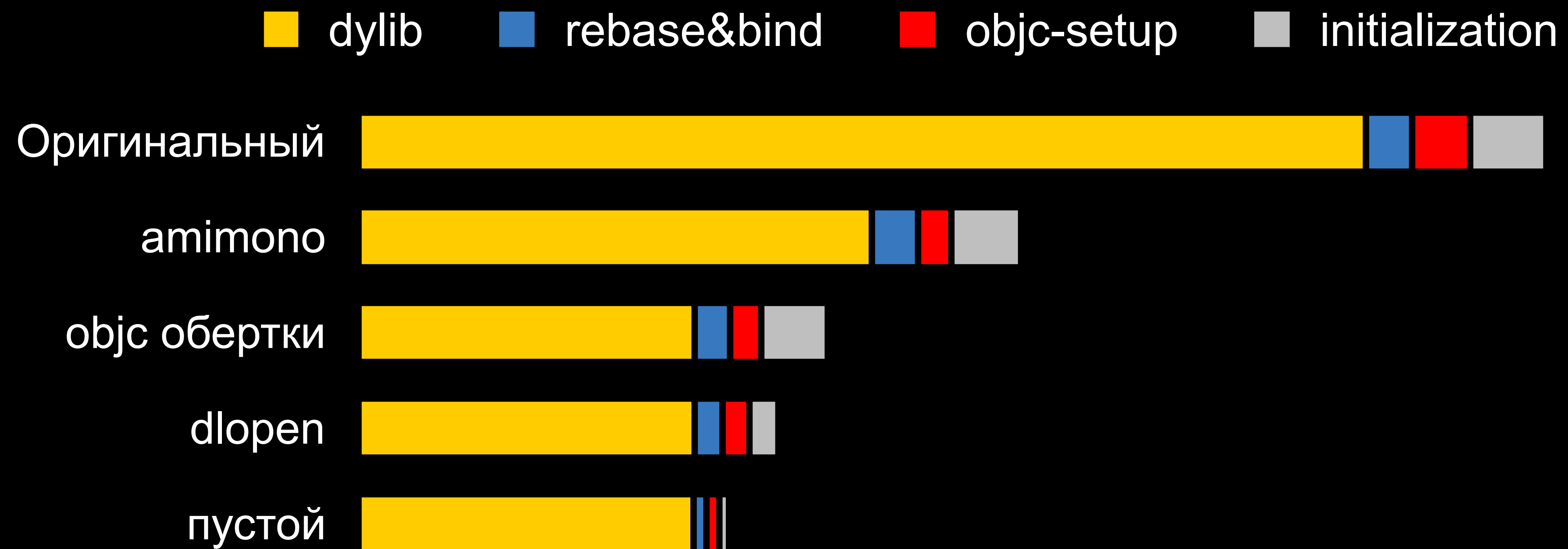
dlopen 

Пустой 

dlopen


| Таким же образом можно загружать динамические библиотеки из подов и собственные модули приложения

Итог



3. Оптимизация *after-main*





Нужно на старте делать
как можно меньше
работы

На что обратить внимание в первую очередь

- | Избыточное создание сущностей на старте
- | Избыточный UI
- | Инициализация, необязательная для показа стартового UI

Инъекция зависимостей

```
class RootViewController {
    init(searchFacade: SearchFacade, routingFacade: RoutingFacade) {
        ...
    }
}
```

[illegible]

Инъекция зависимостей

- На старте приложения подсистемы поиска и маршрутизации не активны, но фасады этих подсистем все равно создаются
- Зависимости фасадов поиска и маршрутизации тоже создаются
- В итоге на старте приложения создаются вообще все возможные сущности

Инъекция зависимостей

- | В итоге на старте приложения создаются вообще все возможные сущности
 - › Дополнительное процессорное время
 - › Сложность профилирования

Инъекция ленивых зависимостей

- | Оформить зависимости сущностей в протоколы
- | В конструкторе принимать реализацию протокола
- | В реализации использовать `lazy var`

Инъекция ленивых зависимостей

```
protocol RootViewControllerDeps {  
    var searchFacade: SearchFacade { get }  
    var routingFacade: RoutingFacade { get }  
}  
  
class RootViewController {  
    init(deps: RootViewControllerDeps) {  
        ...  
    }  
}
```

Инъекция ленивых зависимостей

```
protocol RootViewControllerDeps {  
    var searchFacade: SearchFacade { get }  
    var routingFacade: RoutingFacade { get }  
}
```

```
class RootViewControllerDepsImpl: RootViewControllerDeps {  
    lazy var searchFacade: SearchFacade = { return SearchFacadeImpl(...) }()  
    lazy var routingFacade: RoutingFacade = { return RoutingFacadeImpl(...) }()  
}
```

Composition Root

```
class ApplicationDeps: SearchFacadeImplDeps, RoutingFacadeImplDeps,  
    RootViewControllerDeps {
```

```
    lazy var searchFacade: SearchFacade =  
        { return SearchFacadeImpl(deps: self) }()
```

```
    lazy var routingFacade: RoutingFacade =  
        { return RoutingFacadeImpl(deps: self) }()
```

```
    lazy var rootViewController: RootViewController =  
        { return RootViewController(deps: self) }()
```

```
}
```

Composition Root

```
class ApplicationDeps: SearchFacadeImplDeps, RoutingFacadeImplDeps,
    RootViewControllerDeps {

    lazy var searchFacade: SearchFacade =
        { return SearchFacadeImpl(deps: self) }()

    lazy var routingFacade: RoutingFacade =
        { return RoutingFacadeImpl(deps: self) }()

    lazy var rootViewController: RootViewController =
        { return RootViewController(deps: self) }()
}
```


Инъекция ленивых зависимостей

- | На старте создаются только нужные сущности
 - › Уменьшает время запуска
 - › Упрощает профилирование
- | Не используется рефлексия, resolve проверяется компилятором
 - › Упрощает рефакторинг

Оптимизация UI

| Сократить view-tree, создаваемое на старте

- › ленивое создание контейнерных view-контроллеров
- › ленивое создание пустых контейнерных view
- › ленивое создание view не содержащих контента

Обычное создание view-контроллеров

```
let containerController = ContainerViewController()
let navController = UINavigationController()

override func viewDidLoad() {
    super.viewDidLoad()

    navController.isNavigationBarHidden = true
    navController.pushViewController(containerController, animated: false)

    self.addOverlayController(navController)
}
```

Ленивое создание view-контроллеров

```
let contentController = ContentViewController()
```

```
lazy var navController: UINavigationController = {  
    let res = UINavigationController()
```

```
    res.isNavigationBarHidden = true
```

```
    res.pushViewController(self.containerController, animated: false)
```

```
    self.addOverlayController(res)
```

```
    return res
```

```
}()
```

Ленивое создание view-контроллеров

```
let contentController = ContentViewController()
lazy var navController: UINavigationController = { ... }

override func viewDidLoad() {
    super.viewDidLoad()

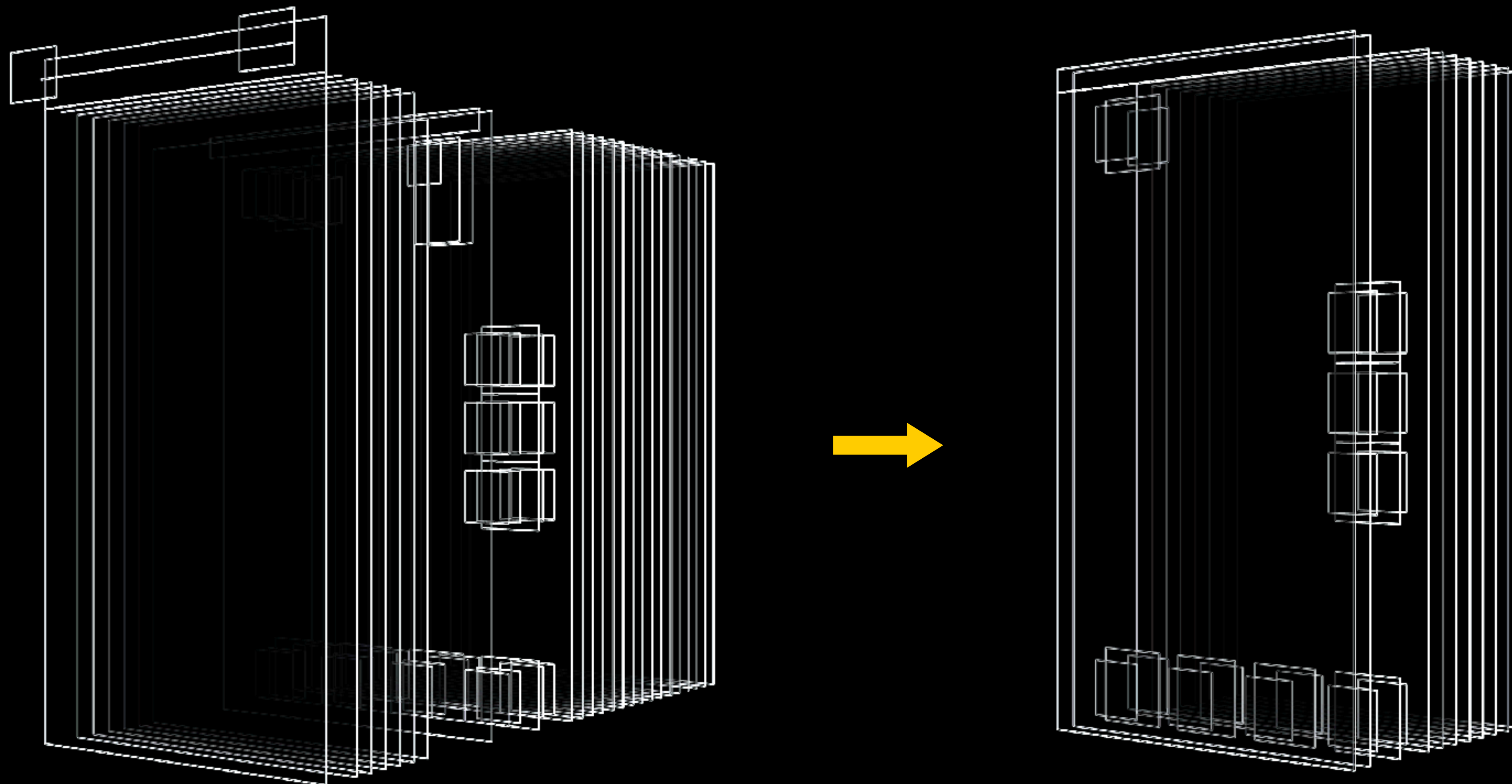
    self.addOverlayController(contentController)
}
```

В Яндекс.Картах

Лениво создаются

- › UINavigationController
- › SplitViewController (на iPad)

Оптимизация UI в Яндекс.Картах



Прочие оптимизации UI

- › Ленивая загрузка шрифтов
- › Графику, генерируемую программно, отрисовать в ассеты
- › Текст также отрисовать в ассеты
- › Сложный autolayout перевести на фреймы
- › ...

Необязательная на старте работа

- | Цель - как можно быстрее разблокировать UI для пользователя
- | Пока UI не загружен - выполнять только необходимые действия, прочие отложить на 0.1-0.3с
 - › Зависит от приложения
 - › Использовать профилировщик

В Яндекс.Картах

Отложили на 0.3с

- › Синхронизацию закладок
- › Отображение закладок на карте
- › Загрузку конфигурации приложения
- › Настройку аудиосесии
- › ...

4. Сохранение результата



Continuous integration

- | Встроить вызов скрипта с автозапусками в CI
- | Собирать статистику запусков
- | Обеспечить доступ к статистике

Логирование создания зависимостей

```
fileprivate func trackCreation<T>(_ creator: () -> T ) -> T {  
    let className = "\(T.self)"  
    let res = creator()  
    // ... логирование  
    return res  
}
```

Логирование создания зависимостей

```
fileprivate func trackCreation<T>(_ creator: () -> T ) -> T {  
    let className = "\(T.self)"  
    let res = creator()  
    // ... логирование  
    return res  
}
```

```
lazy var some: SomeClass = { return self.trackCreation {  
    return SomeClass(deps: self)  
}}()
```

Логирование списка загружаемых библиотек

```
var count: UInt32 = 0
let imagesNamesPointer = objc_copyImageNames(&count)
var names: [String] = []

for i in 0..
```

Информация о процессе через sysctl

```
#import <sys/sysctl.h>
```

```
int mib[] = { CTL_KERN, KERN_PROC, KERN_PROC_PID, getpid() };
```

```
struct kinfo_proc kp;
```

```
size_t len = sizeof(kp);
```

```
sysctl(mib, 4, &kp, &len, NULL, 0)
```


Время старта через sysctl

```
#import <sys/sysctl.h>

int mib[] = { CTL_KERN, KERN_PROC, KERN_PROC_PID, getpid() };
struct kinfo_proc kp;
size_t len = sizeof(kp);
sysctl(mib, 4, &kp, &len, NULL, 0)
struct timeval start_time = kp.kp_proc.p_starttime;
```

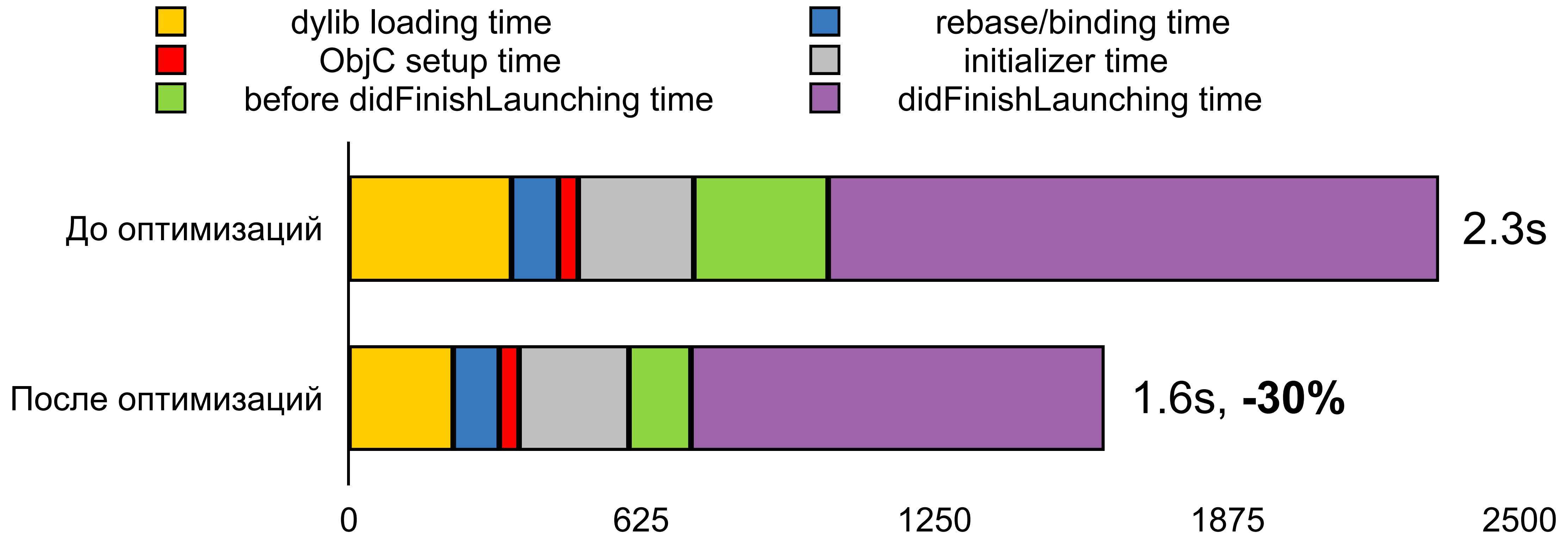
Замеры пользовательских запусков

- | Замерять полное время запуска от нажатия на иконку приложения через `sysctl` и `gettimeofday`
- | Отправлять в системы аналитики полное время запуска

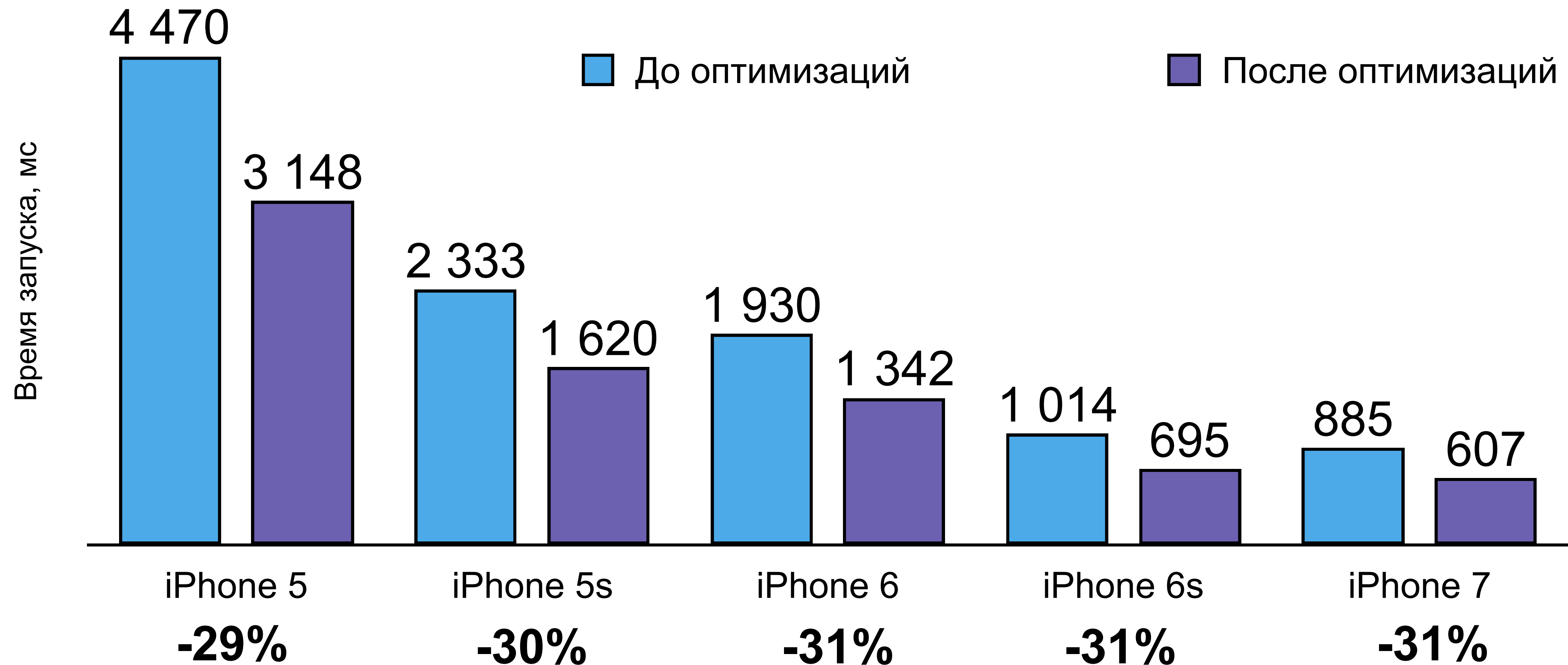
Заключение



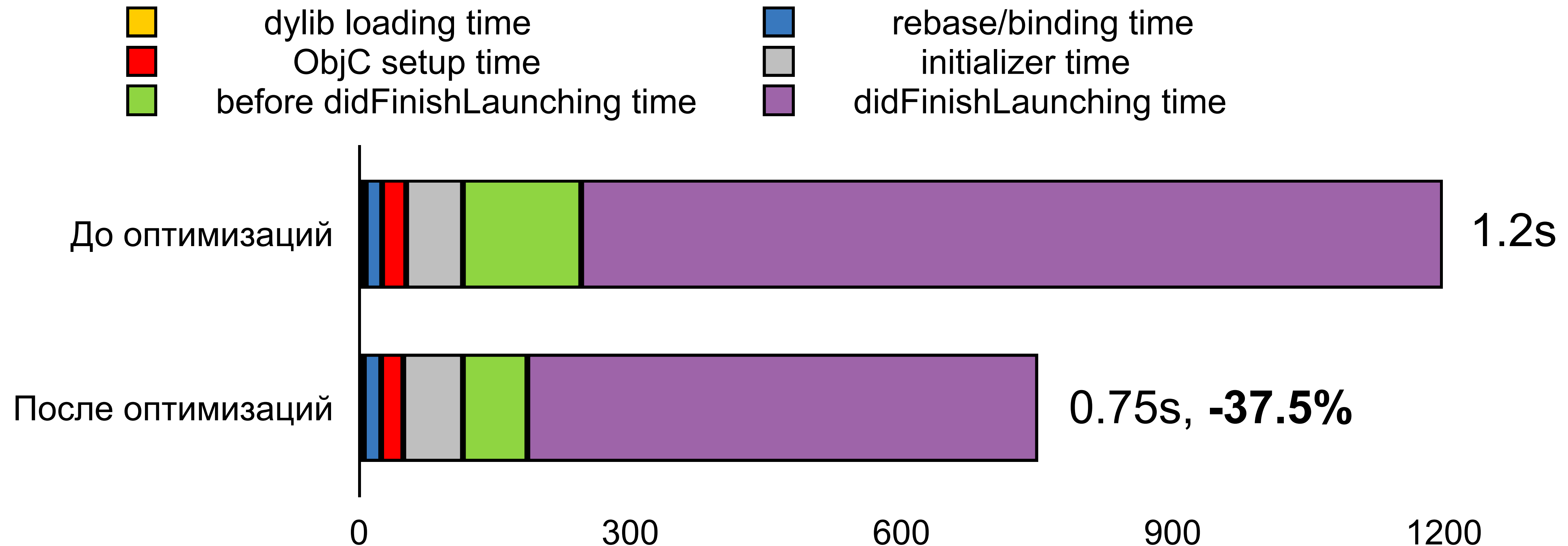
Яндекс.Карты - холодный, 5s



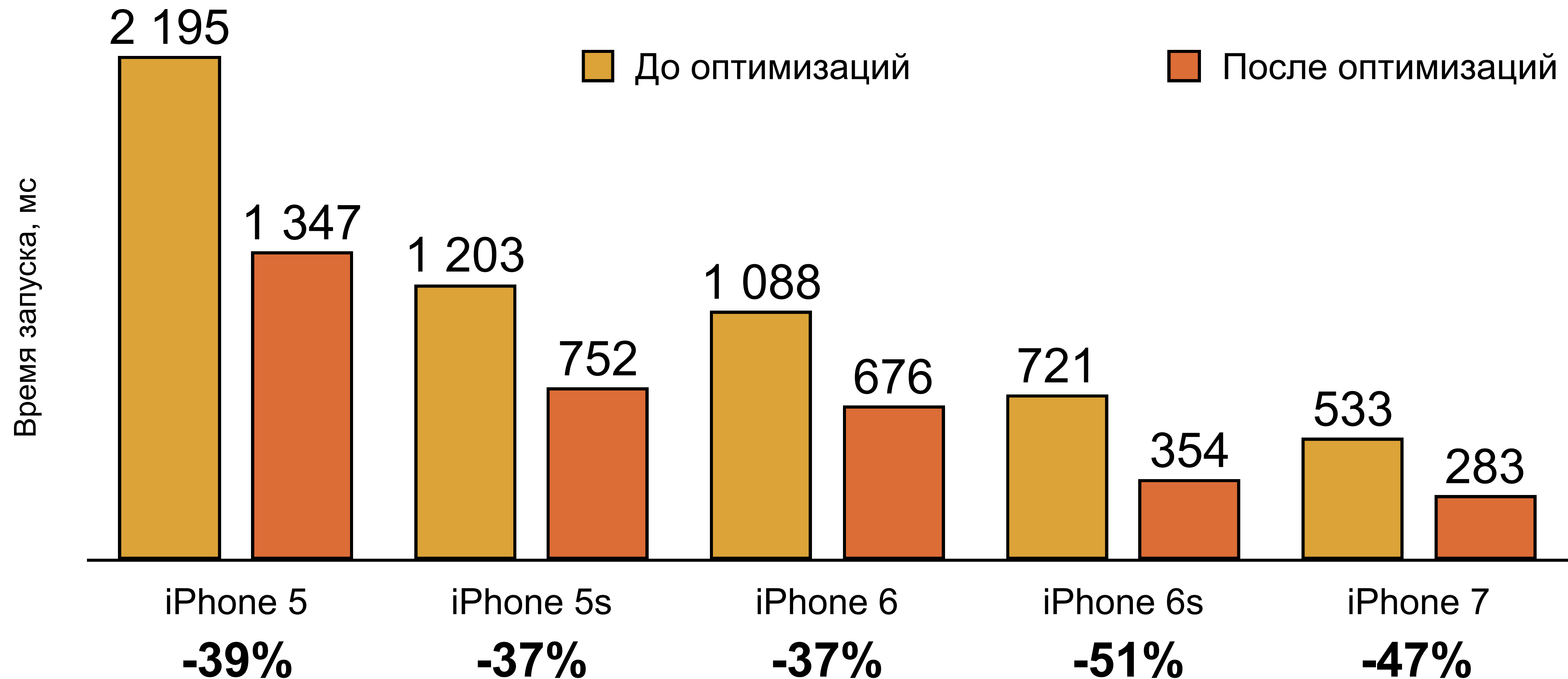
Яндекс.Карты - холодный запуск



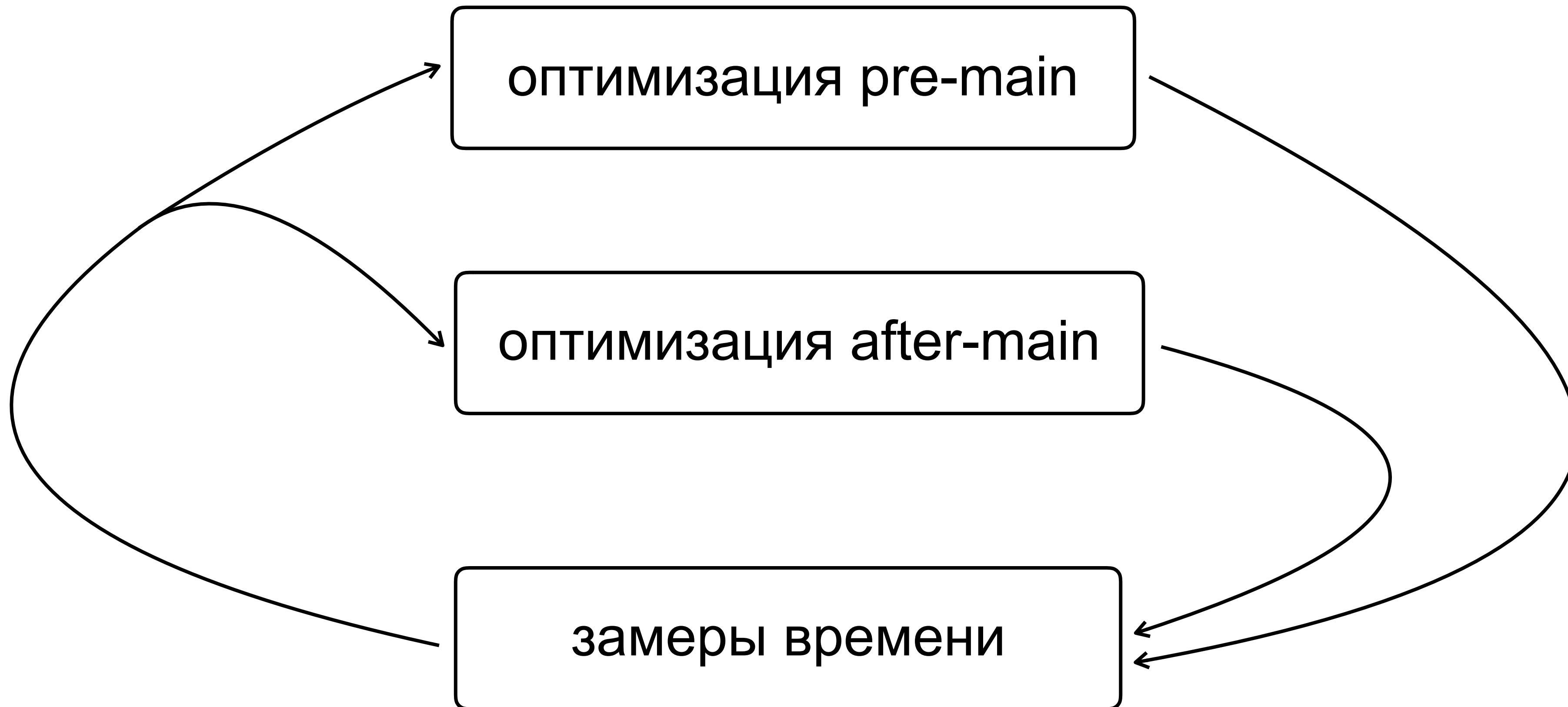
Яндекс.Карты - теплый, 5s



Яндекс.Карты - теплый запуск



Оптимизация запуска - итерационный процесс



Ссылки

Примеры из доклада

WWDC 2016 "Optimizing App Startup Time"

libimobiledevice

Доклад Почты Mail.Ru

cocoapods-amimono

sysctl

Спасибо за внимание

Николай Лихогруд

Руководитель группы разработки Яндекс.Карт для iOS



likhogrud@yandex-team.ru