Как сделать, чтобы не тормозило,

пожалуйста

Евгений Шаповалов, ВКонтакте

Содержание

Что значит “приложение тормозит”

Зачем оптимизировать

Зачем оптимизировать

хз Tue

★★★☆☆ тата 10

уберите пожалуйста задержку уже наконец-то!!!!!!!

Зачем оптимизировать

- Деградация скорости приложения отрицательно влияет на удержание и привлечение новых пользователей
- Понимание проблем с производительностью улучшит разработку приложения на ранних стадиях, и минимизирует стоимость устранения проблем с производительностью на этапах вывода в стор

Analysis of the Impact of Performance on Apps Retention, University of Helsinki, 2018

Что значит “приложение тормозит”

- Долгий старт самого приложения, > 400 ms по версии Apple
- Плохая отзывчивость. Время от нажатия до видимого действия > 200 ms (hangs)
- Лаги при анимации (hitches)
- Длительная загрузка контента

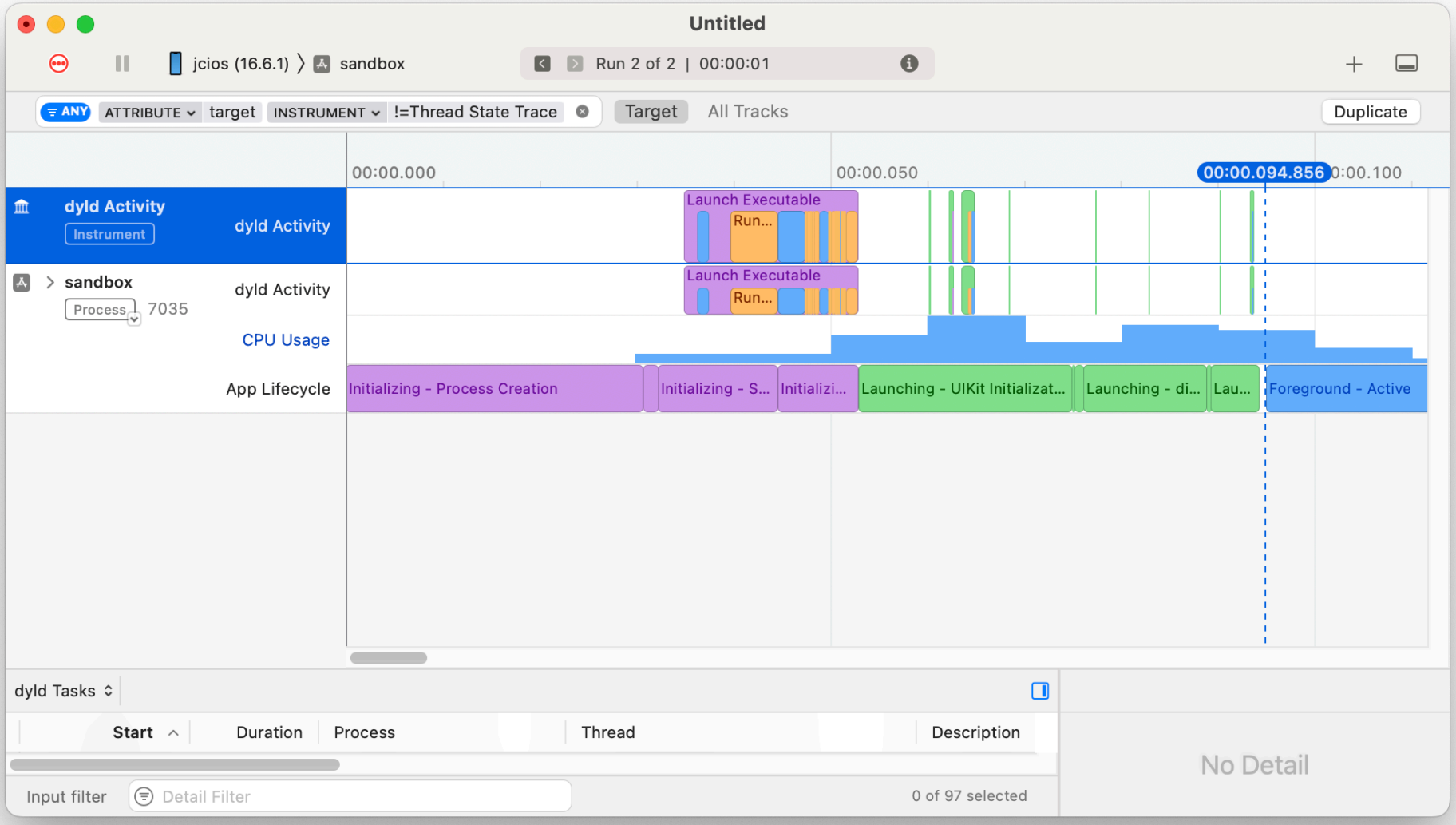
Способы измерения производительности

Что измерять для оптимизации

Время старта как основная точка измерений

- Влияет на все пользователей приложения
- Во время старта проявляются основные проблемы с производительностью
- В измерение времени старта можно включить критические пути использования при автоматизации
- Время старта относительно реально измерить

Как измерить время старта



Как измерить время старта

Профайлер (Xcode Instruments) - хороший способ. Но есть минусы:

- Нестабильный результат, производительность плавает от запуска к запуску
- Нет возможности автоматизировать измерения
- Может не отражать реальное положение дел у пользователей приложения

Как измерить время старта

Кроме Xcode Instruments:

- Xcode Organizer
- MetricKit
- Crashlytics и собственные метрики
- XCTest + XCTApplicationLaunchMetric

Собственный стенд для замеров

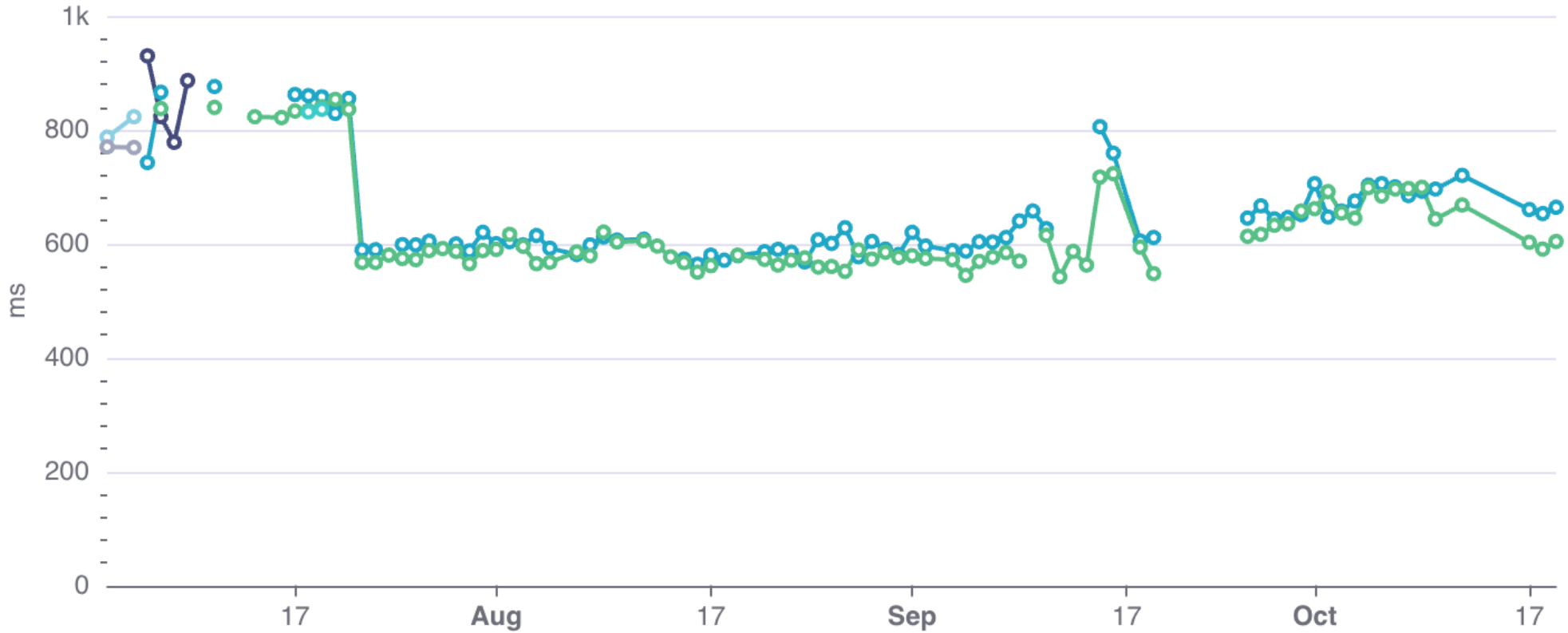
- Помогает оценить производительность и ее деградацию регулярно, до момента выкладки в стор
- Можно организовать через XCTestCase + XCTApplicationLaunchMetric
- Метрики можно собрать через stdout, или через сеть

Собственный стенд для замеров

```
func testLaunch() throws {
    let options = XCTMeasureOptions()
    options.iterationCount = 100

    let launchMetric = XCTApplicationLaunchMetric(
            waitUntilResponsive: true)

    measure(metrics: [launchMetric],
            options: options) {
        XCUIApplication.default.launch()
   }
}
```

Собственный стенд для замеров

Пример: метрика didFinishLaunching (медиана)



Если время запуска растет

- Очевидный вариант — найти коммит, который привел к удлинению запуска
- Git bisect не очень поможет: разброс большой, а данные не отсортированы
- Запускать прогон очень долго (от получаса на один коммит)

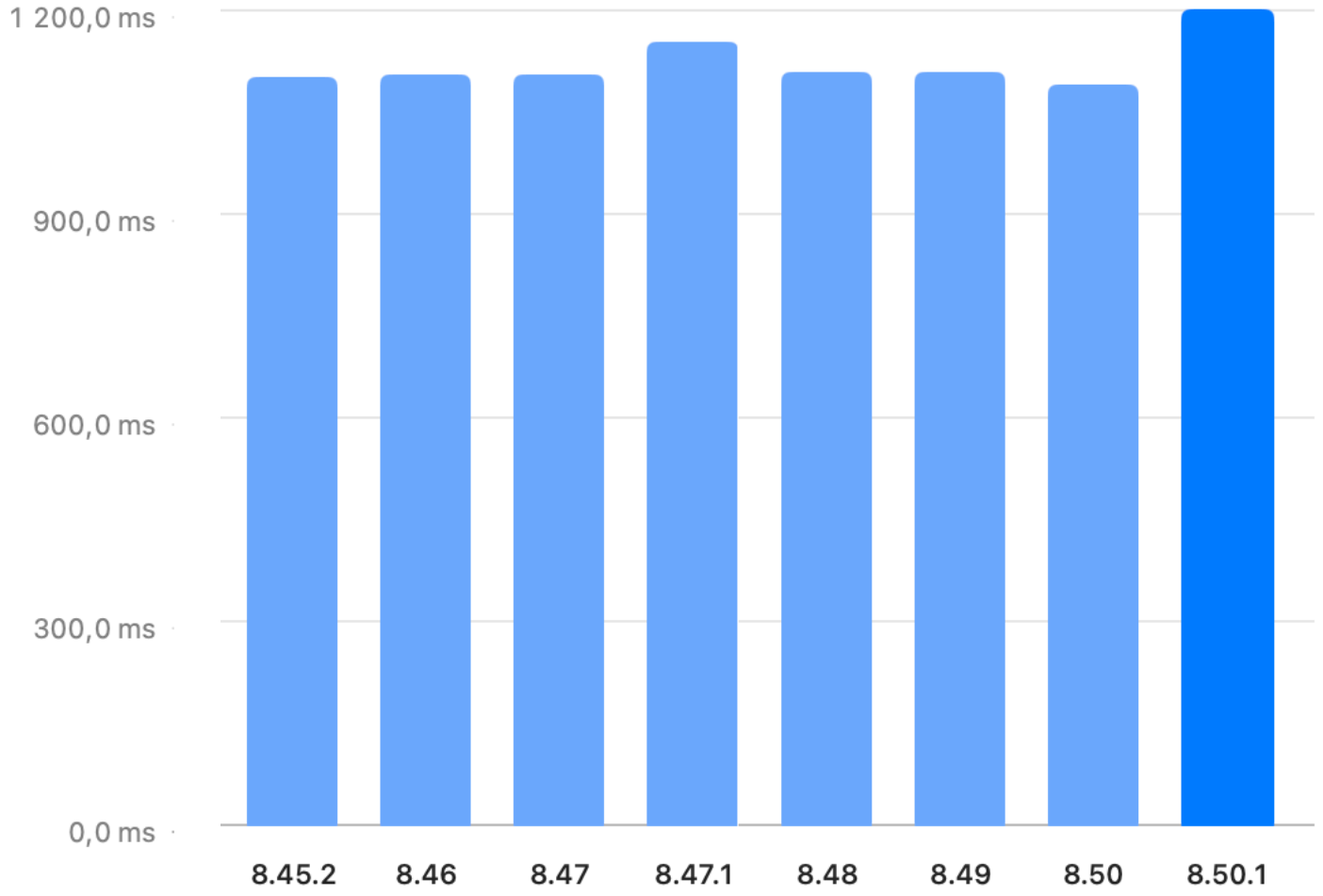
Если время запуска растет

Как упростить поиск коммита-виновника

- Первый вариант — ускорить прогоны, используя несколько устройств одновременно
- Второй — стабилизировать время запуска, чтобы git bisect работал

Стенд не всегда помогает

Пример: Xcode organizer, старт (медиана)



Стенд не всегда помогает

- У реальных пользователей разброс устройств достаточно большой
- Пути использования могут отличаться от тестового
- Стенд плохо учитывает закешированные данные, которые могут тормозить старт
- Логируем метрики старта в продакшне
- Учитываем то, как запускается приложение

Логирование времени старта у пользователей

Какие бывают виды старта

- Первый старт совсем холодный
- Последующие теплые старты: кеш dyld сохранен в /Library/Caches, часть исполняемого кода может уже быть в памяти
- Прогретый старт начиная с iOS 15 (Prewarming)

Prewarming при логировании

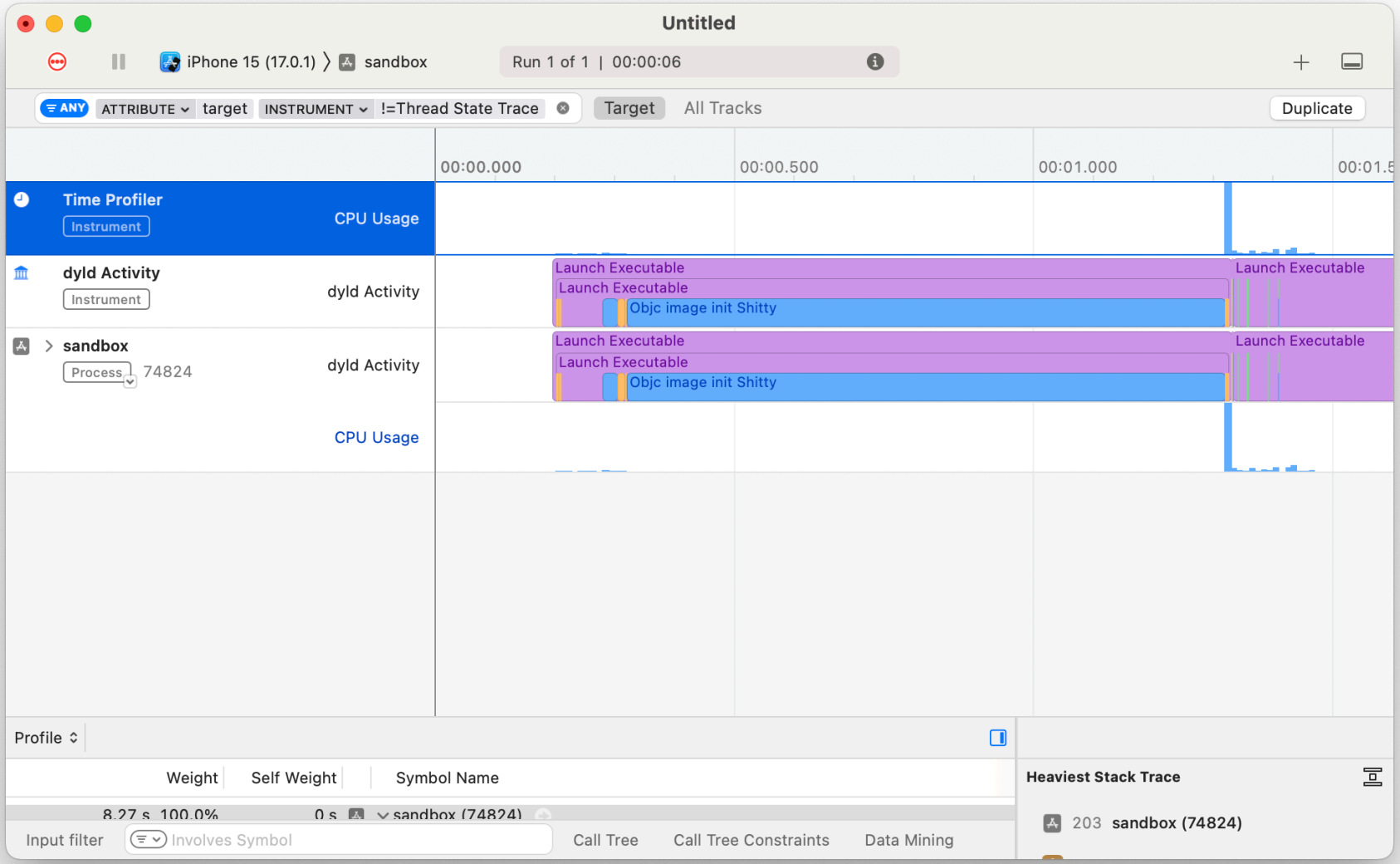
- Если логировать точки старта в зарелиженном приложении, стоит учитывать prewarming
- Определяется через переменную окружения ActivePrewarm = 1
- Доступен с iOS 15, есть разница в зависимости от версии:
- < 15.4: Загружается образ, до main. Main вызывается по нажатию на иконку
- >= 15.4: Загружается образ, + вызывается main. didFinishLaunching вызывается по нажатию на иконку

Оптимизация: есть ли жизнь до main

На что можно повлиять в pre-main

- Сделать таргет iOS 15+. Там доступны дополнительные оптимизации в ABI для DYLD
- Уменьшить количество загружаемых при старте фреймворков, так как часть связывания происходит в DYLD при старте
- Уменьшить количество вызовов +load и static initializers в линкуемых библиотеках

Как отследить статическую инициализацию?



Как отследить статическую инициализацию?

- Xcode 15 Instruments, Static initializer calls
- Xcode < 15? Есть переменная окружения OBJC_HELP=YES
- Если инициализация долгая, а фреймворк сторонний — можно попробовать загрузить его лениво через dlopen

OBJC_HELP=YES

Переменные окружения, через которые можно получить дополнительную информацию об инициализации приложения до main

```
OBJC_PRINT_OPTIONS      list which options are set
OBJC_PRINT_IMAGES       log image and library names as they are loaded
OBJC_PRINT_LOAD_METHODS log calls to class and category +load methods
OBJC_PRINT_INITIALIZE_METHODS   log calls to class +initialize methods
OBJC_PRINT_RESOLVED_METHODS     log methods created by +resolveClassMethod: and +resolveInstanceMethod:
OBJC_PRINT_CLASS_SETUP  log progress of class and category setup
OBJC_PRINT_PROTOCOL_SETUP       log progress of protocol setup
OBJC_PRINT_IVAR_SETUP   log processing of non-fragile ivars
OBJC_PRINT_VTABLE_SETUP log processing of class vtables
OBJC_PRINT_VTABLE_IMAGES        print vtable images showing overridden methods
OBJC_PRINT_CACHE_SETUP  log processing of method caches
```

От main до первого фрейма

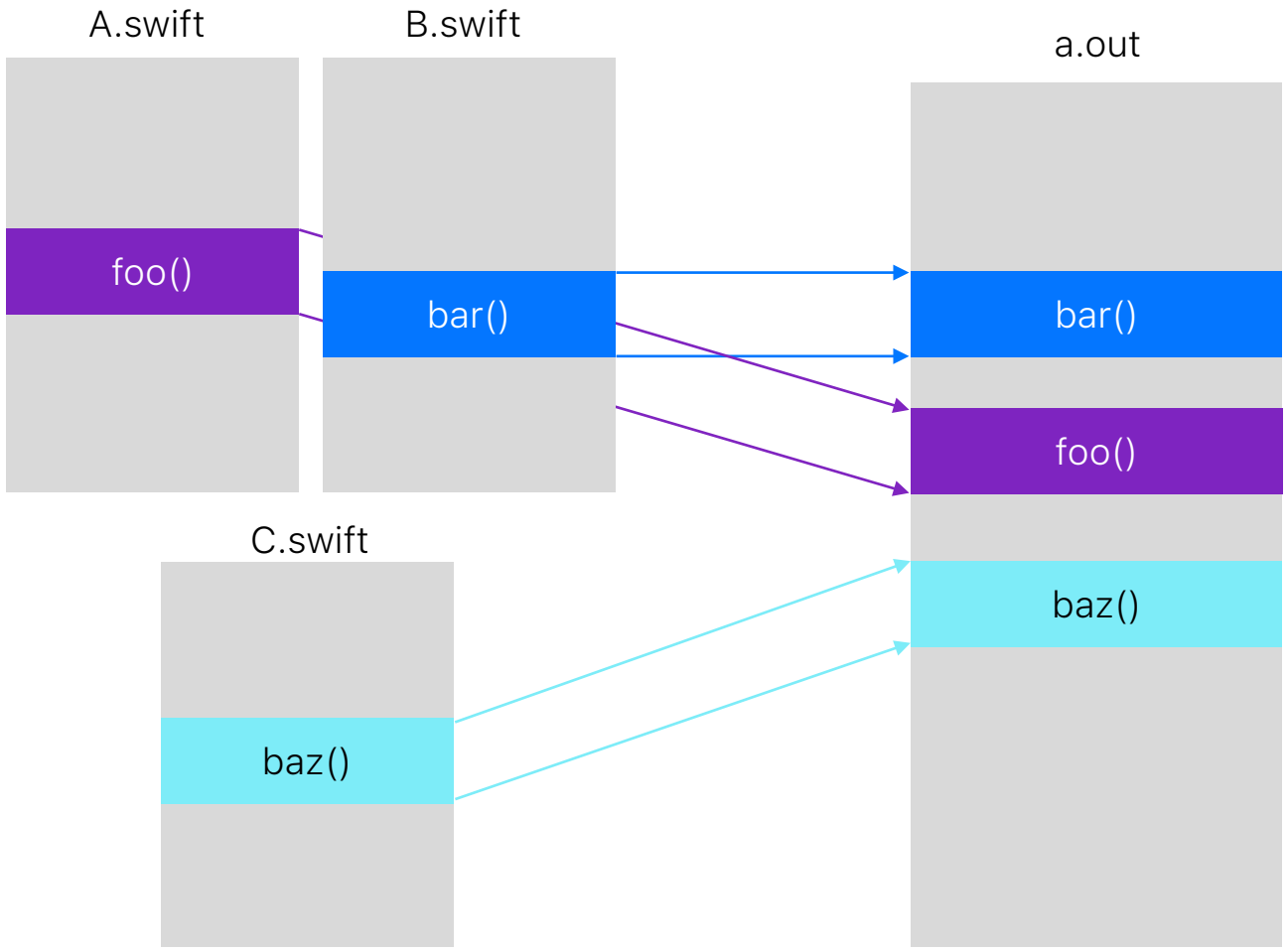
Большое количество кода ведет к page faults

- Чтобы код исполнился — его надо загрузить в память
- Но загружаются не только нужные страницы, но и еще несколько после них (aka prefetch)
- Множество вызовов методов из разных частей исполняемого файла приводит к частым page fault-ам

Как уменьшить количество page faults

- Оптимизация по размеру вместо скорости (-Os вместо -O)
- Статическая линковка фреймворков и mergeable libraries (Xcode 15, iOS 12+)
- ORDER_FILE при сборке позволяет расположить код в порядке вызова

Order File



How Order Files Reduce App Startup Time, Emerge Tools Blog

Как получить порядок вызовов при старте

```
// CFLAGS = -fsanitize-coverage=func,trace-pc-guard
// SWIFTFLAGS = -sanitize=undefined
// SWIFTFLAGS = -sanitize-coverage=func


void __sanitizer_cov_trace_pc_guard(uint32_t *guard)
{
    *guard = 0;
    void *pc = __builtin_return_address(0);
    // Сохраняем pc в массив
}


// ...
// После запуска приложения сохраняем символы

Dl_info info = {0};
dladdr(pc, &info);
NSString *name = @(info.dli_sname);
```

Как получить порядок вызовов при старте

```
_main
-[AppDelegate
application:didFinishLaunchingWithOptions:]
-[RootViewController viewDidLoad]
-[DiscoverViewController viewDidLoad]
___copy_helper_block_e8_32s
-[DiscoverViewController appendRow:]
_CGRectMake
-[DiscoverViewController appendButton:action:]
-[DiscoverViewController testCheck]
_$s7sandbox12OrderingDemoC6sharedACvgZTo
_$s7sandbox12OrderingDemoC6sharedACvgZ
_$s7sandbox12OrderingDemoC6sharedACvau
_$s7sandbox12OrderingDemoC6shared_WZ
_$s7sandbox12OrderingDemoCMa
_$s7sandbox12OrderingDemoCACycfC
_$s7sandbox12OrderingDemoCACycfcTo
_$s7sandbox12OrderingDemoCACycfc
_$s7sandbox12OrderingDemoC3runyyFTo
_$s7sandbox12OrderingDemoC3runyyF
_CxxOrderingDemo
__Z18DoSomethingFromCxxv
_$s7sandbox12OrderingDemoC12secondInCodeyyF
_$ss27_finalizeUninitializedArrayySayxGABnlF
_$sSa12_endMutationyyF
_$ss5print_9separator10terminatoryypd_S2StFfA0_
…
```

Пример order file

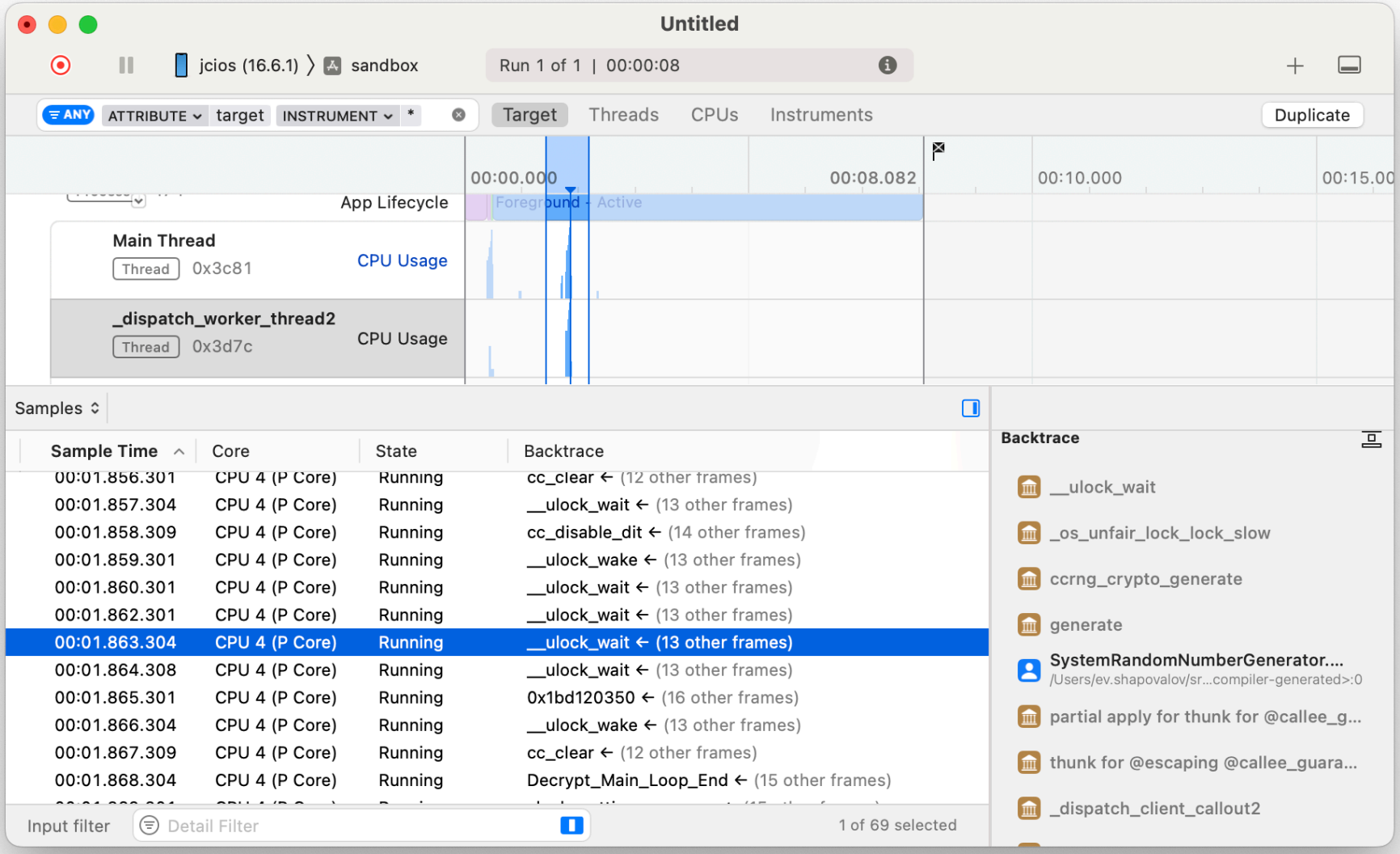
Параллелизация = скорость?

- Появляется многопоточность
- Можно распараллелить код, например вынести загрузку из файлов и ресурсов в отдельный поток
- Параллельный код может работать дольше, чем последовательный

Места синхронизации потоков не всегда очевидны

```
func iteration() {
    for _ in 0..<100000 {
        Int.random(in: 0...255)
    }
}


// 2 секунды:

DispatchQueue.concurrentPerform(iterations: 2) {
    iteration()
}

// 1 секунда:

for _ in 0..<2 {
    iteration()
}
```

Места синхронизации потоков не всегда очевидны



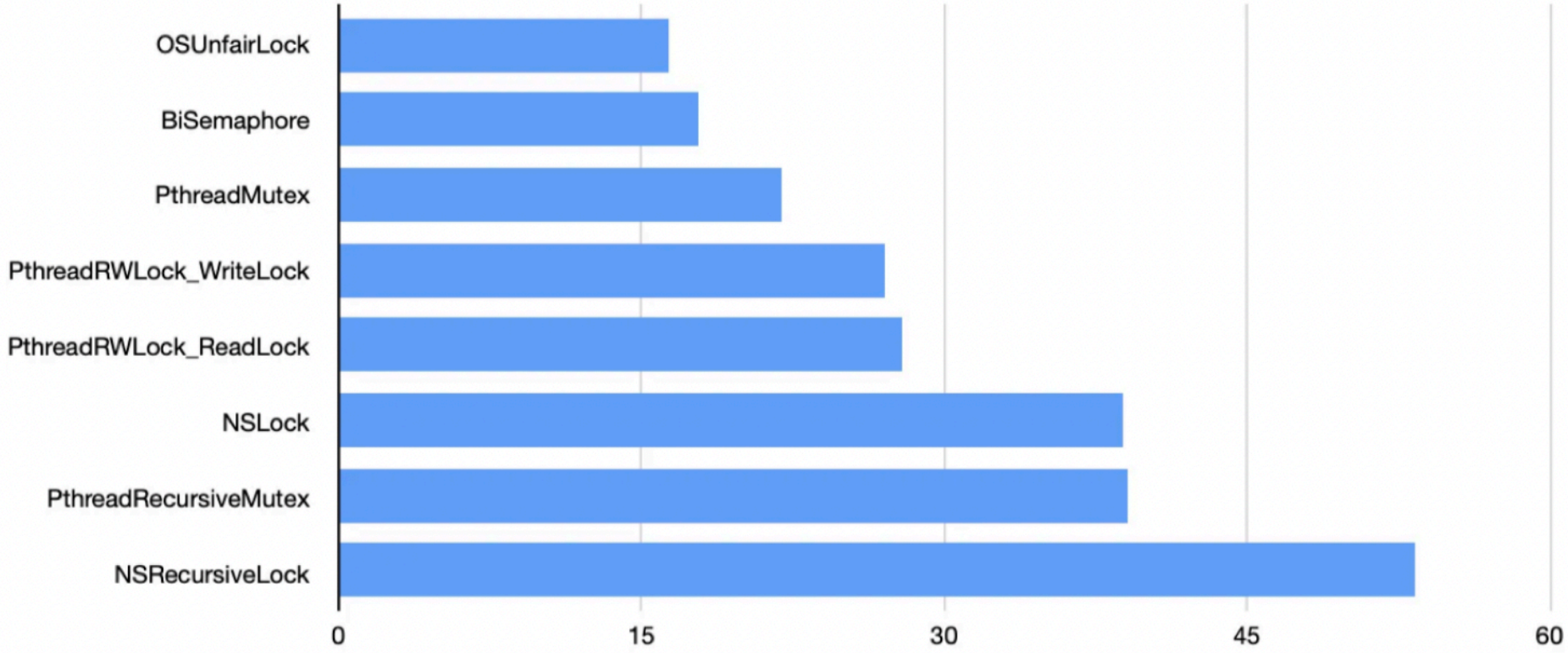
Места синхронизации потоков не всегда очевидны

- Сравните время параллельной обработки с последовательной
- Xcode Instruments: в narrative видно, сколько времени поток ждет в блокировках и где
- Стоит минимизировать количество потоков по возможности

Синхронизация потоков может занять время

- Каждый dispatch блок имеет оверхед из-за синхронизации
- Если нужна ручная синхронизация через примитивы, то OSUnfairLock — самый оптимальный вариант

Синхронизация потоков может занять время



Время lock + unlock, наносекунды

Слишком много потоков тормозит вычисления

- Блокирующие примитивы синхронизации, или доступ к IO, порождает новые потоки в GCD
- Переключение контекстов дает оверхед по CPU
- Это отнимает время выполнения других потоков, в том числе main
- А еще контексты потоков потребляют память

Слишком много потоков тормозит вычисления

- Неблокирующий доступ к IO уменьшает количество потоков
- Предпочтение — серийным очередям, вместо global / concurrent
- Пул потоков Swift concurrency ограничен числом ядер, что поможет оптимально распределить нагрузку на CPU

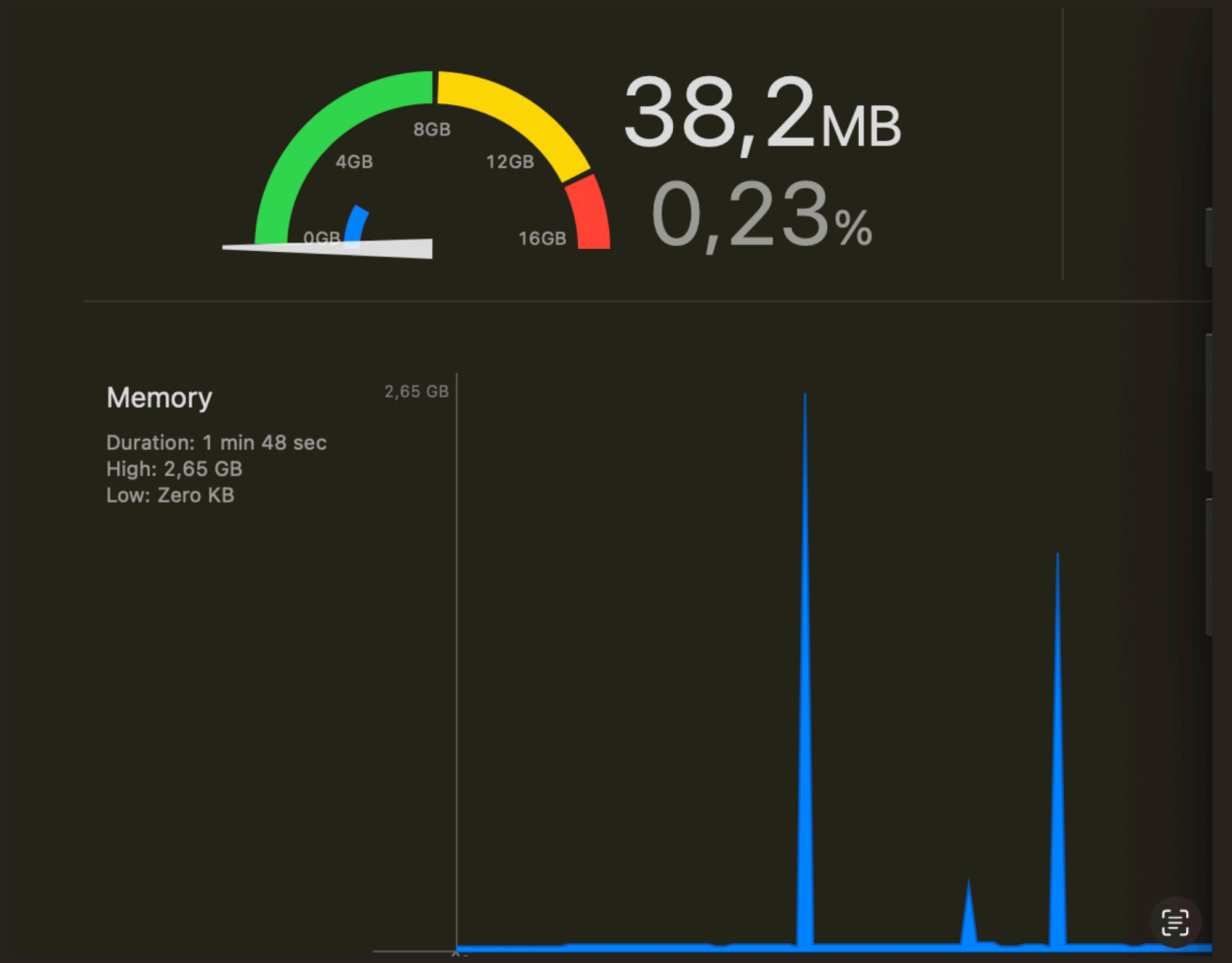
Всплески потребления памяти

- Некоторые вызовы API (например, UIImage.init) забивают autorelease pool
- Dispatch очереди имеют AutoreleaseFrequency = .never по-умолчанию

Все это может приводить к всплескам потребления памяти

Такие всплески, помимо OOM, тормозят приложение из-за выгрузки памяти memory compressor-ом

Всплески потребления памяти



Всплески потребления памяти

```
func loadImages(paths: [String]) {
    for path in paths {
        DispatchQueue.global().async {
            let image = UIImage(contentsOfFile: path)
            // store image somehow...
        }
    }
}
```

Всплески потребления памяти

- Имеет смысл оборачивать большое количество вызовов UIKit API в autoreleasepool
- Может помочь AutoreleaseFrequency = .workItem для dispatch queues
- Аллокации больших объектов на старте могут занять дополнительное время из-за memory compression, стоит их отложить по возможности

Ленивая инициализация

- Создание объектов при необходимости — будет тормозить переход к функциональности
- Отложенное создание объектов по таймеру — может тормозить приложение
- Частный случай: стараемся избежать создания view до перехода на экран
- К созданию UIView раньше времени приводит обращение к UIViewConroller.view. Например, view.isVisible

Оптимизация приложения после запуска

Лаги на UI потоке

- От действия пользователя до обработки события > 200 ms
- Случается, когда main поток забит длительными блоками
- Xcode Instruments — Hangs
- Xcode Organizer — Hang rate
- Поможет watchdog, который мониторит отзывчивость UI потока, и шлет статистику в базу

Если тормозит загрузка контента

- Интернет на телефонах не самый быстрый
- Можно использовать сжатие трафика (zstd, msgpack вместо plain json)
- Современные протоколы HTTP2/HTTP3 (QUIC) быстрее устанавливают соединение, и имеют мультиплексирование запросов

Выводы

Коротко, где искать оптимизации

- Сделайте стенд с регулярными замерами старта
- Используйте свежие инструменты по возможности (Xcode 15, Merged libraries)
- Попробуйте распараллелить код, но следите, чтобы не стало медленнее
- Может помочь сжатие контента и современные протоколы (HTTP2/HTTP3)

Спасибо за внимание!

Источники и полезные материалы доступны по ссылке через QR-код