курс | development | 12 недель

iOS: разработка приложений с 0

r_d

iOS программа курса 20 занятий

1
Swift: начало

2
ООП: основы

3
Создание iOS-приложения в Xcode

4
Создание интерфейса iOS-приложения

5
Динамические интерфейсы, часть 1

6
Динамические интерфейсы, часть 2

iOS программа курса 20 занятий

7

Динамические интерфейсы, часть 3

8

Навигация в приложении, часть 1

9

Навигация в приложении, часть 2

10

Анимации в iOS

11

Работа с памятью в iOS

12

Многозадачность в iOS, часть 1

iOS программа курса 20 занятий

13
Многозадачность в iOS, часть 2

14
Дебаг iOS-приложения

15
Тестирование

16
Хранение данных в приложении

17
Работа с сетью в приложении

18
Сборка приложения

iOS программа курса 20 занятий

19

Современные архитектуры для iOS приложений

20

Защита курсовых проектов

Многозадачность в iOS. Часть 2

- Некоторые особенности DispatchQueue
- Sync vs Async operations

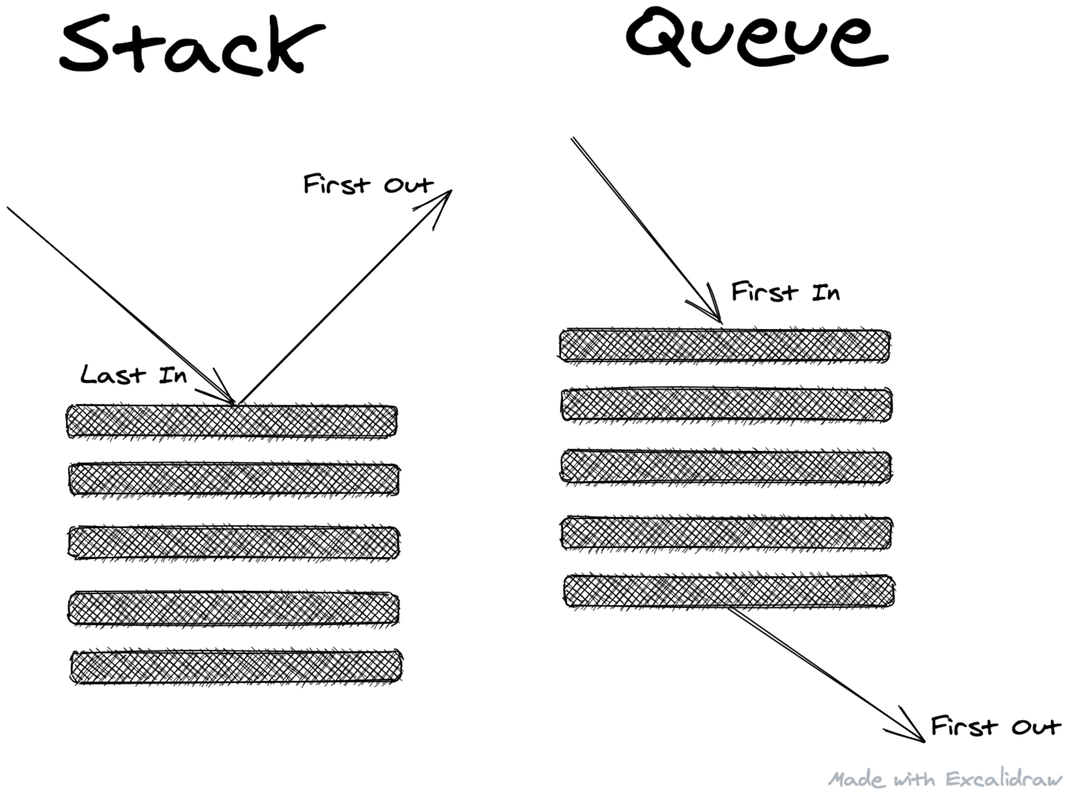
r_d

Особенности DispatchQueue

Объект класса DispatchQueue хранит задачи, как одноименная структура данных «очередь».

Важно: из-за «асинхронной» природы механизма очереди в GCG, все задачи начинают выполняться в порядке First in, First Out.

Окончание выполнения (задач в очереди) в таком порядке не гарантируется!



Особенности DispatchQueue

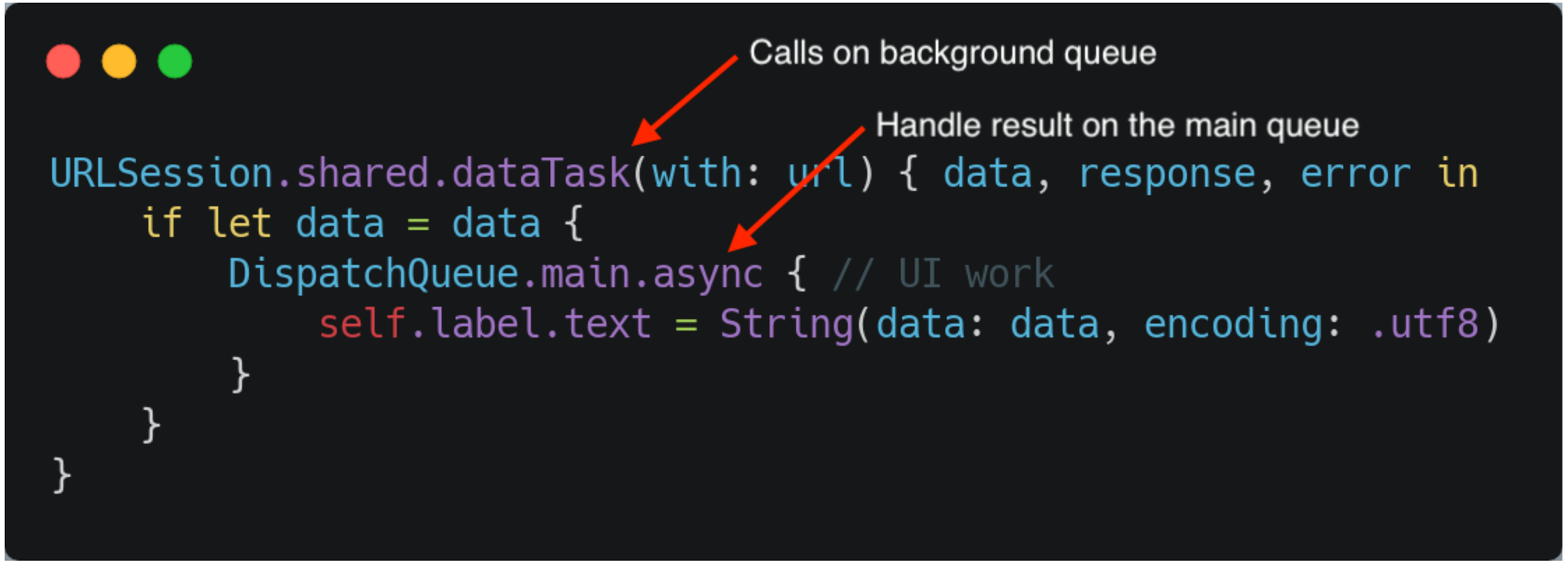
Существует три основных типа очередей:

- main dispatch queue (serial, pre-defined)
- global queues (concurrent, pre-defined)
- private queues (могут быть как serial, так и concurrent)

Особенности DispatchQueue

- Все задачи по отрисовке user interface и обработке user interactions (touch, scroll, pan) выполняются системой на main queue.
- Если на main queue выполняется какая-то длительная операция, не относящаяся к UI / user interactions, UI приложения начнёт тормозить.
- Если нужно выполнить ресурсоёмкие задачи (вычисления, клиент-серверные операции, работа с БД) и мы хотим избежать тормозов пользовательского интерфейса, сложную задачу мы должны выполнить не на main queue, а результат выполнения данной операции передать на main queue.

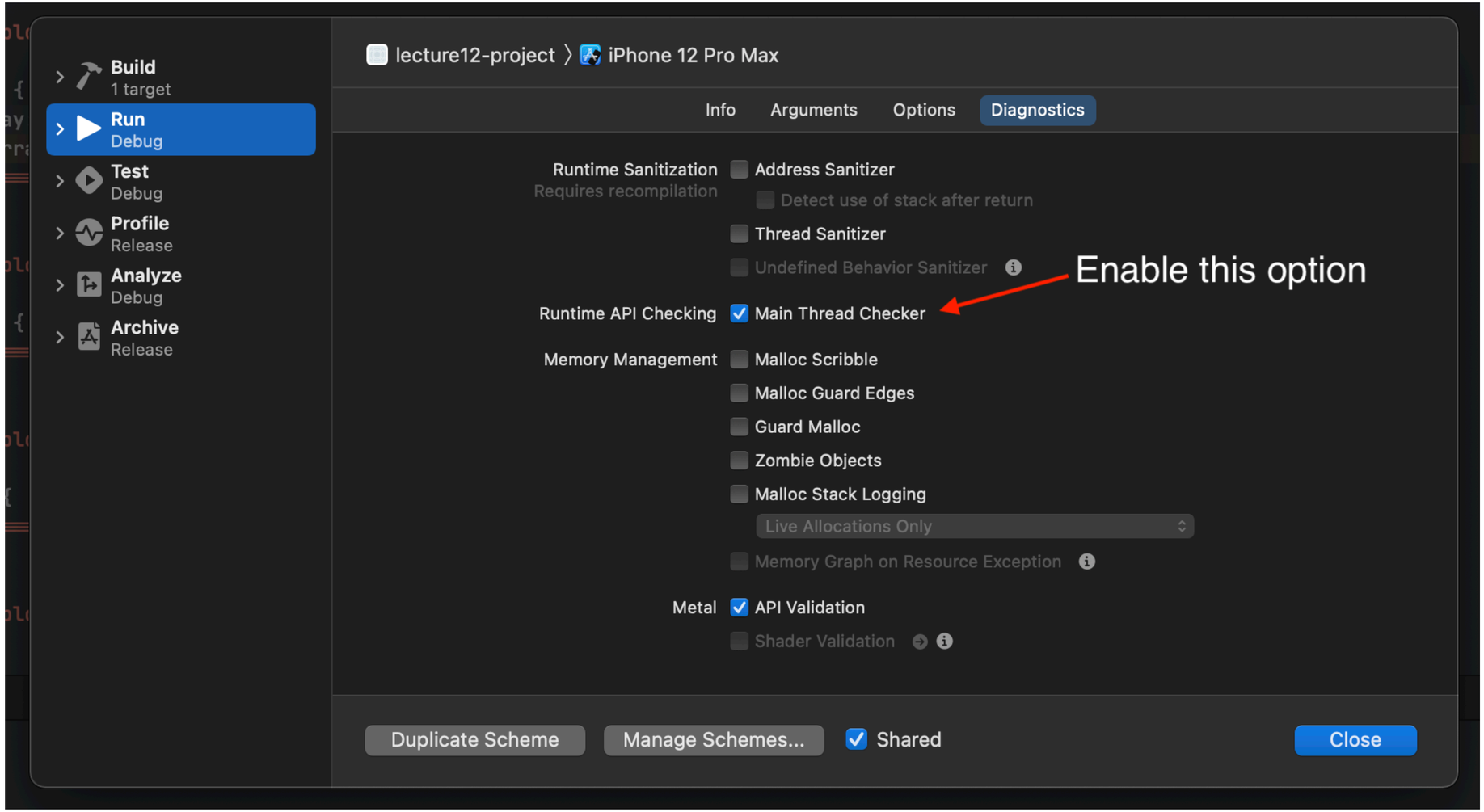
Особенности DispatchQueue



Особенности DispatchQueue

- Важно: любые операции по работе с user interface должны быть выполнены на основном потоке!
- Для дополнительной подстраховки можно в настройках схемы запуска проекта включить опцию Main Thread Checker.
- Данная опция выдаёт warnings в момент выполнения программы (если Xcode обнаружил, что какой-то UI-related код не выполняется на main thread).

Особенности DispatchQueue



Sync vs Async operations

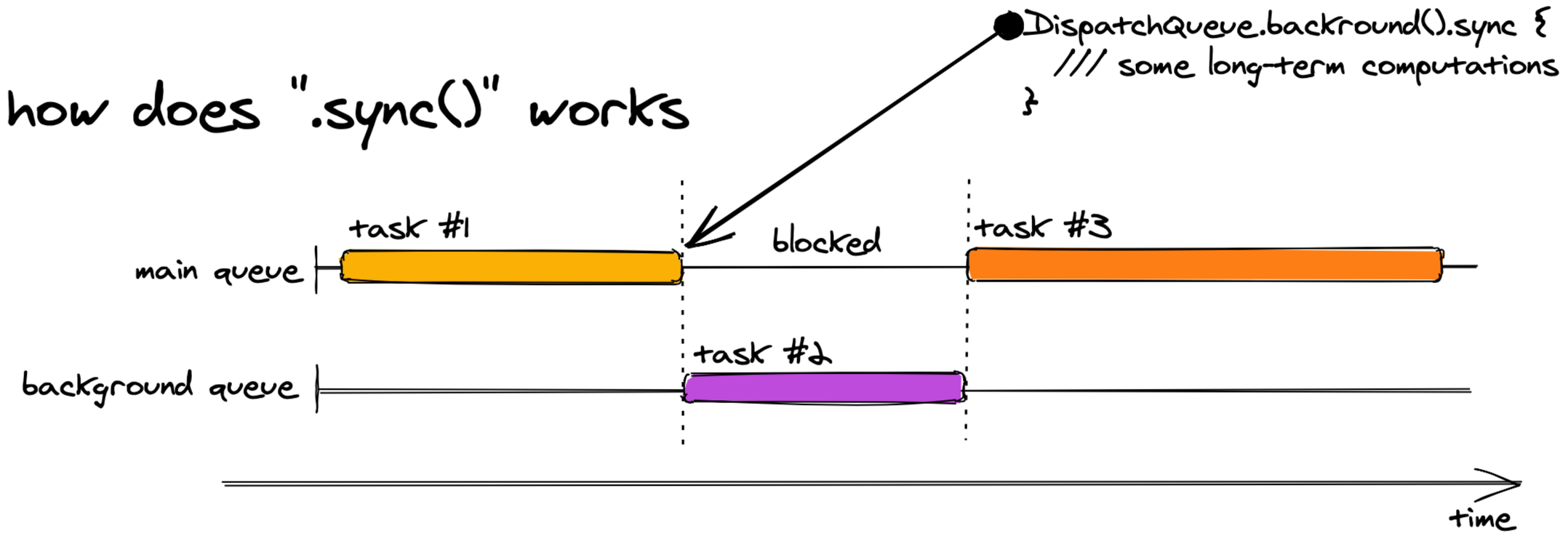
- Когда вы добавляете задачу «в очередь» на выполнение, есть две основные опции: синхронное выполнение и асинхронное.
- Любая из опций выше влияет на источник (очередь, где вызвали метод sync или async), откуда данная задача была добавлена в очередь.

Sync vs Async operations

- Как работают синхронно добавленные операции на очередь.
- Когда программа достигает выражения «.sync()» в коде, текущая очередь (на которой был вызван метод sync) заблокируется на время выполнения операции внутри замыкания метода sync.
- Когда операция закончит своё выполнение, та очередь, на которой был вызван метод sync, продолжит свое выполнение (iOS вернёт этой очереди управление).

Sync vs Async operations

- Как работают синхронно добавленные операции на очередь.

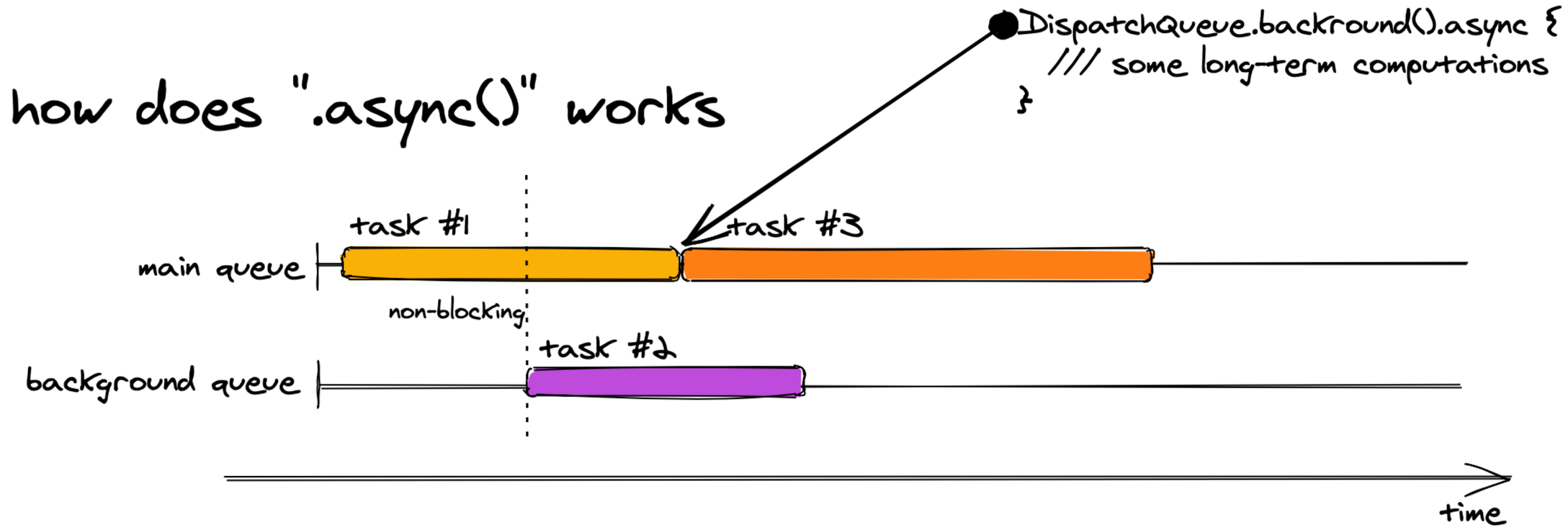


Sync vs Async operations

- Как работают асинхронно добавленные операции на очередь.
- Когда программа достигает выражения «.sync()» в коде, текущая очередь (на которой был вызван метод *async*) не заблокируется на время выполнения операции внутри замыкания метода sync.
- Операции на той очереди, на которой был вызван метод async, сразу же продолжит исполнять код после вызова метода async.

Sync vs Async operations

- Как работают асинхронно добавленные операции на очередь.



Sync vs Async operations

Когда что использовать?

- Как рекомендует Apple — в большинстве случаев стоит использовать async.
- sync стоит использовать, когда хочешь сделать свою thread-safe структуру данных (но это уже Advanced Level).

Sync vs Async operations

- Важно помнить, что async операции не дают возможности дождаться окончания выполнения операции, поэтому результат их выполнения нельзя вернуть в «том же» месте, используя return.
- Давайте посмотрим на код 🎉

Sync vs Async operations

- Обратите внимание на одну из ошибок, которую нам подсказал компилятор.
- Для того, чтобы код функции `processImageAsync(…)` компилировался без ошибок, необходимо аргумент completion передавать как *@escaping*.

Sync vs Async operations

- Указывать @escaping явно нужно в том случае, когда замыкание передано в качестве аргумента функции, но вызывается (замыкание) уже после вызова return внутри функции.
- По-умолчанию любое замыкание, которое передаётся в качестве аргумента в функцию, имеет атрибут @non-escaping.

Sync vs Async operations

- В случае с функцией `processImageAsync(…)` аргумент `completion` должен быть `@escaping`, т.к. замыкание будет вызвано асинхронно, после выполнения всего кода внутри функции `processImageAsync(…)`.

```
func processImageAsync(data: Data, completion: @escaping (Image?) -> Void) {
    guard let image = Image(data: data) else { return }
    imageProcessingQueue.async {
        let processedImage = upscaleAndFilter(image: image)
        completion(processedImage)
    }
}
```

спасибо

задавайте вопросы