Лекция 13

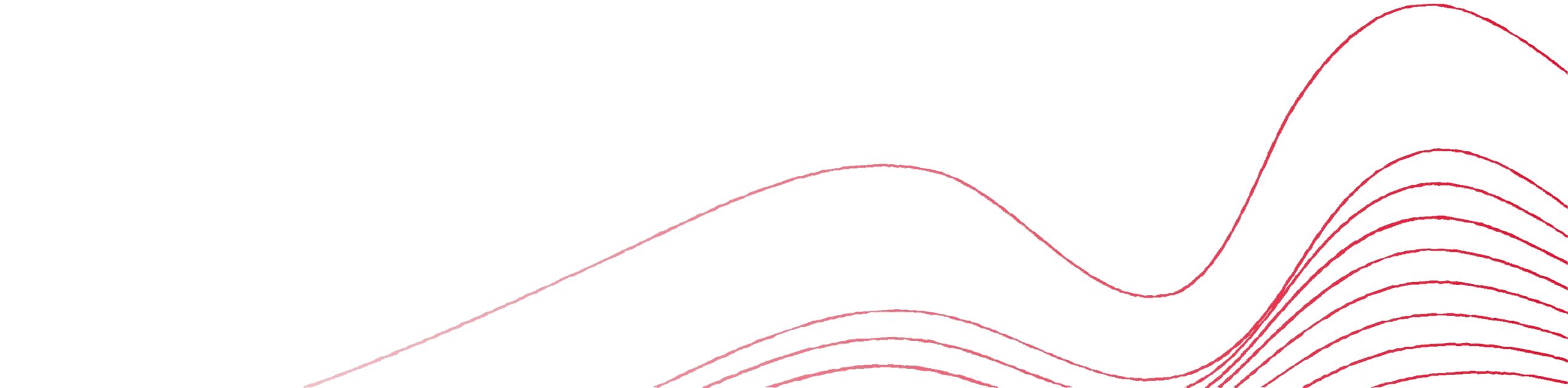
Agenda

Многопоточность: Grand Central Dispatch

Выполнение задач процессором

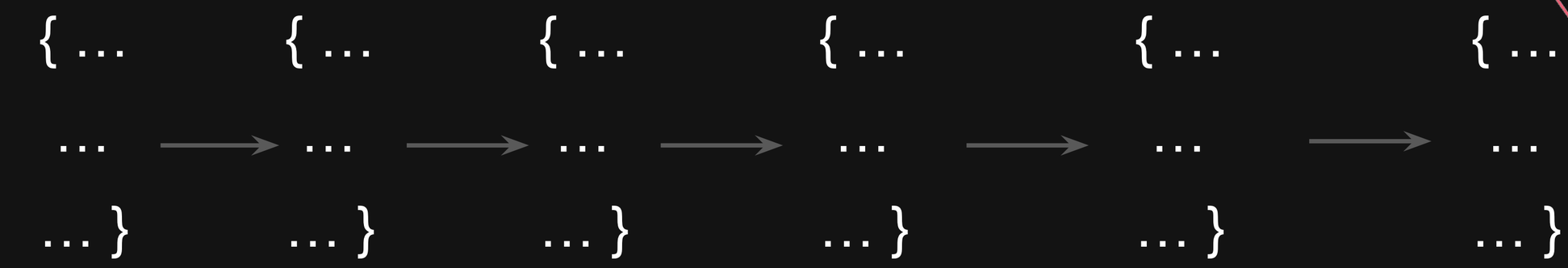
Queue

Проблемы многопоточности



Queue

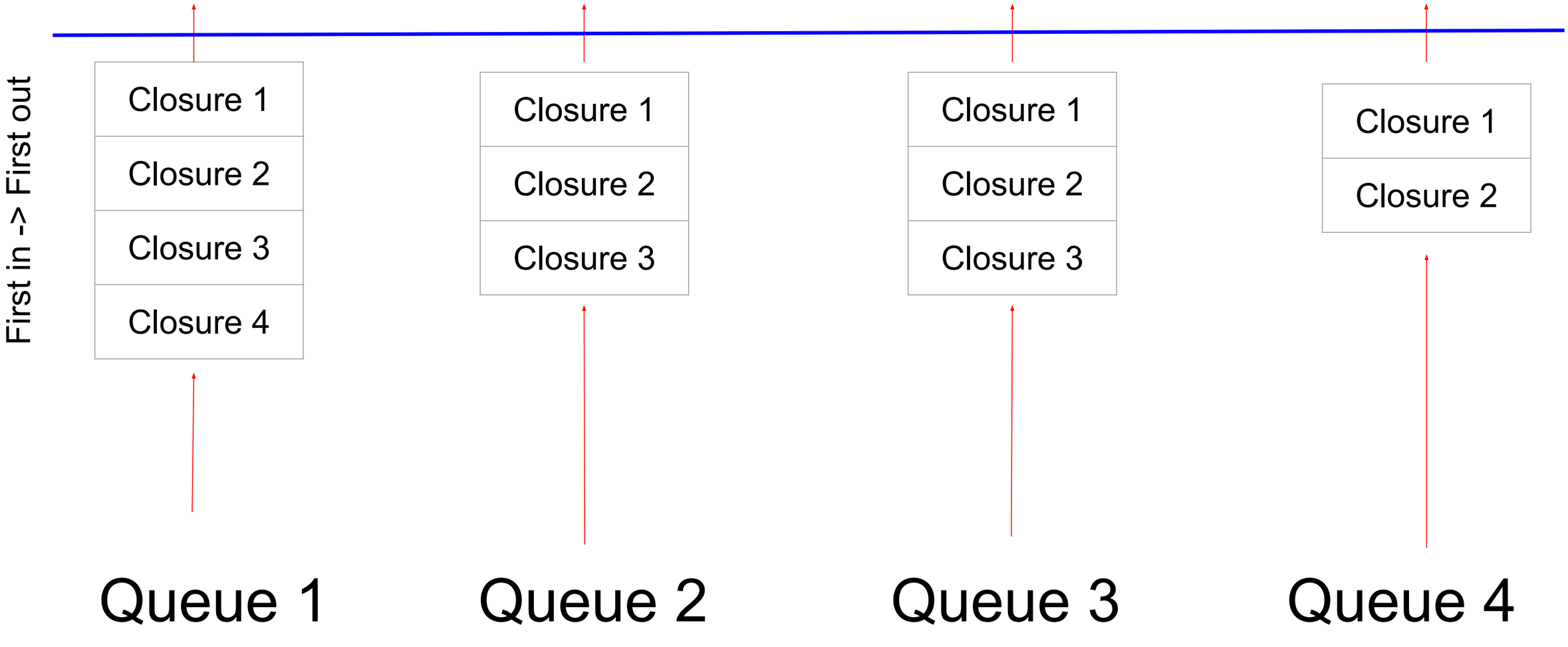
Queues - это очереди из замыканий(функций):



Выполнение кода в программе

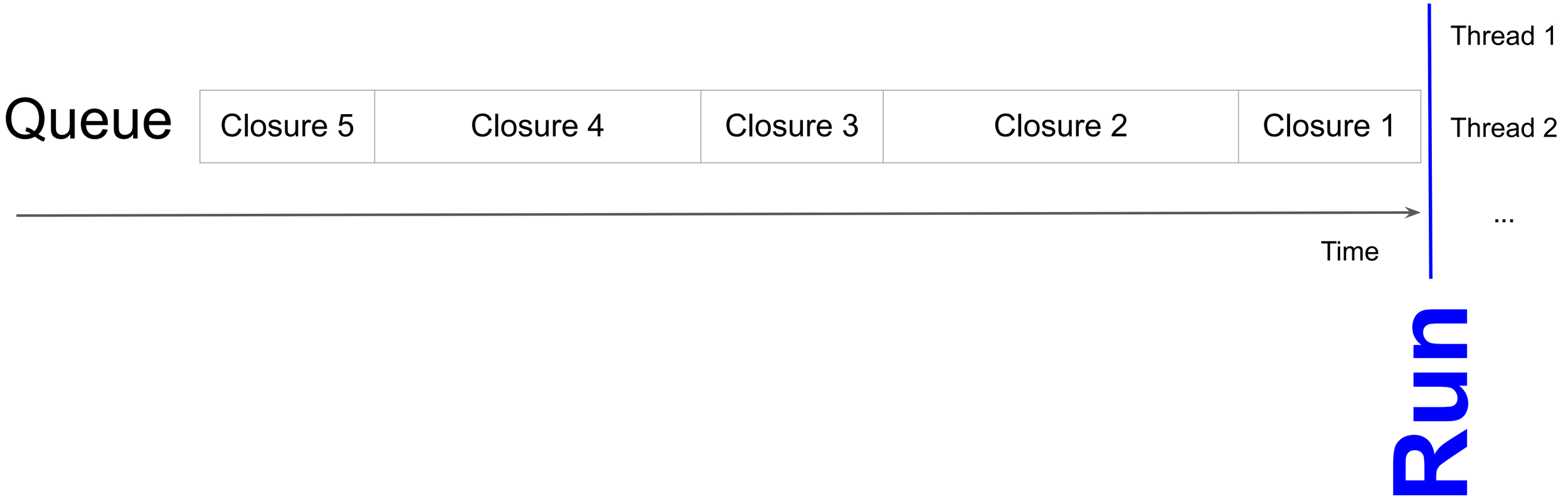
Run

Первым выполняется замыкание, которое было первым помещено в queue (FIFO)



Serial queues
(последовательная)

Задача (замыкание), которая находится на вершине очереди, “вытягивается” iOS и работает до тех пор, пока не закончится, затем вытягивается следующий элемент из очереди

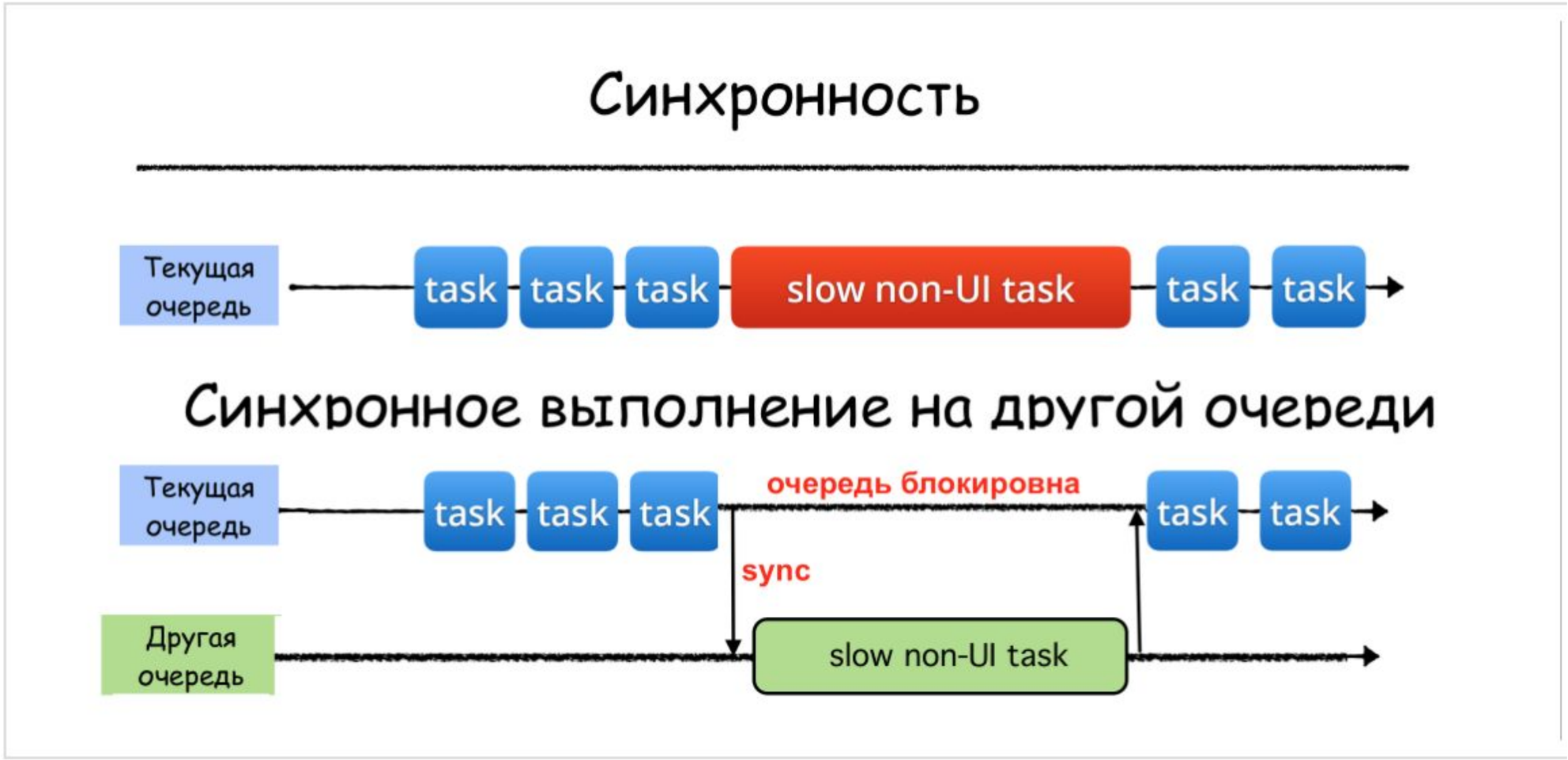


Concurrent queue

(параллельная)

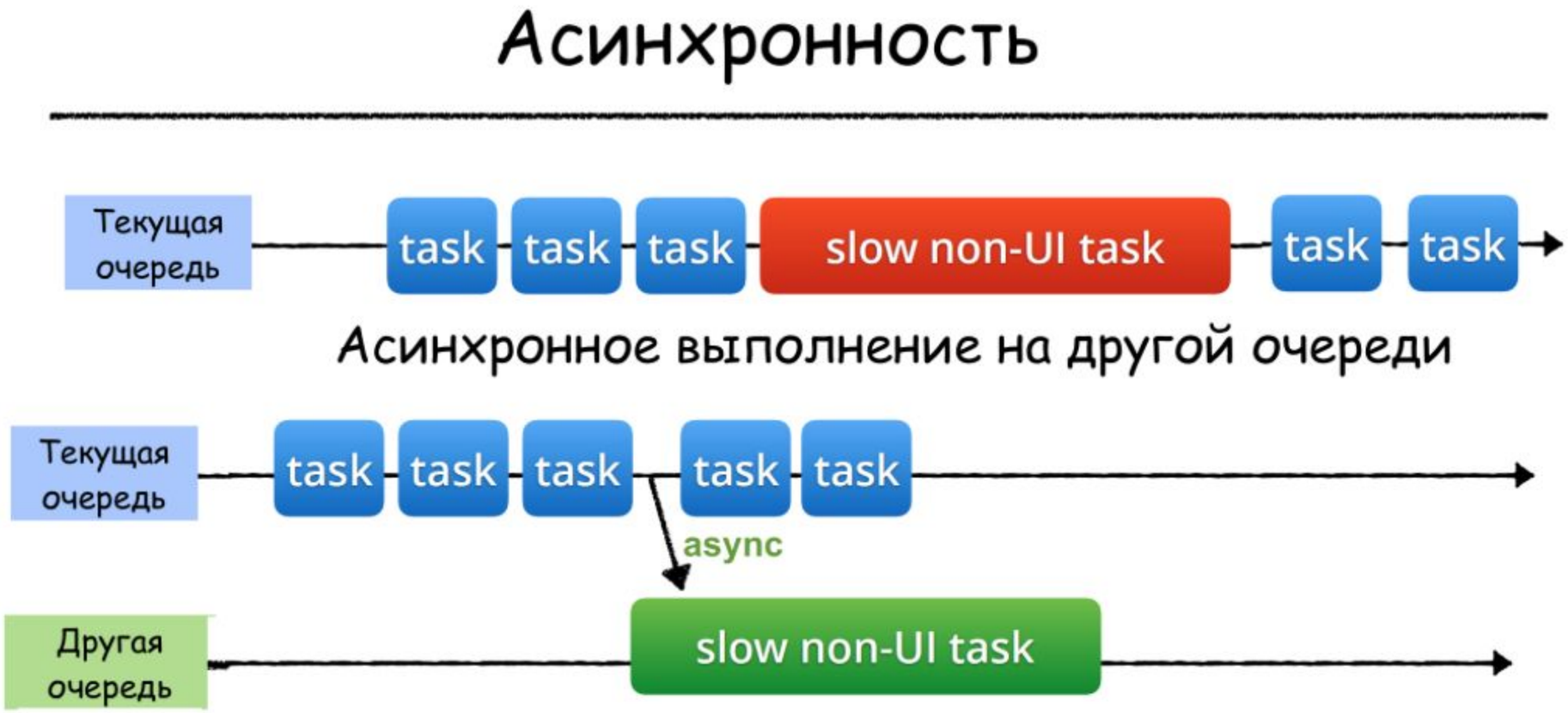
Queue

Задача (замыкание), которая находится на вершине очереди, “вытягивается” iOS и работает до тех пор, пока не закончится, затем вытягивается следующий элемент из очереди



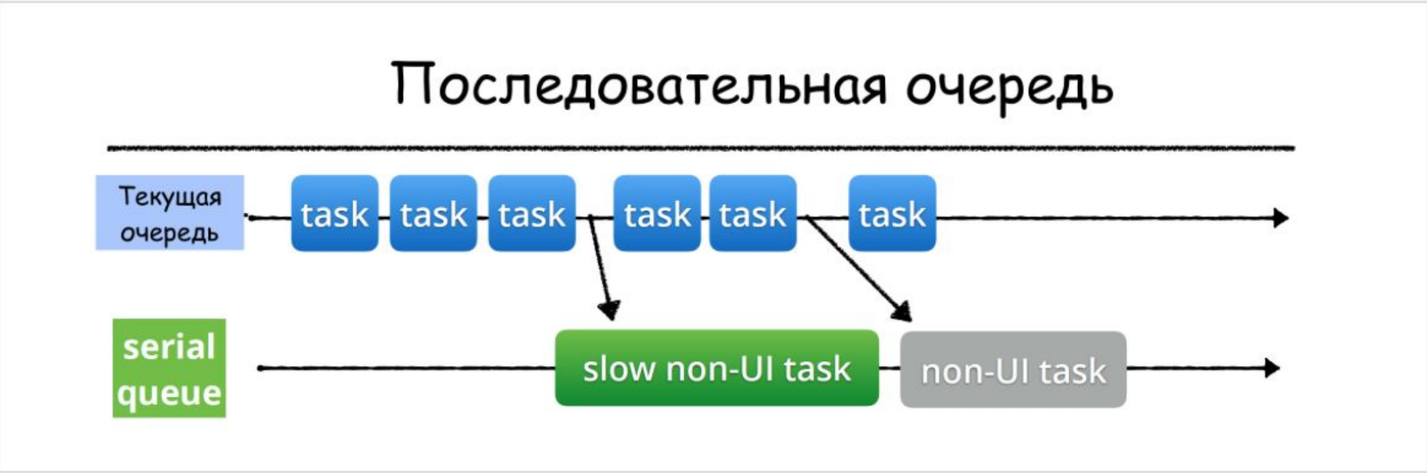
.sync - синхронное выполнение по отношению к текущей очереди.

Синхронная функция `sync` возвращает управление на текущую очередь только после полного завершения задания, тем самым блокируя текущую очередь

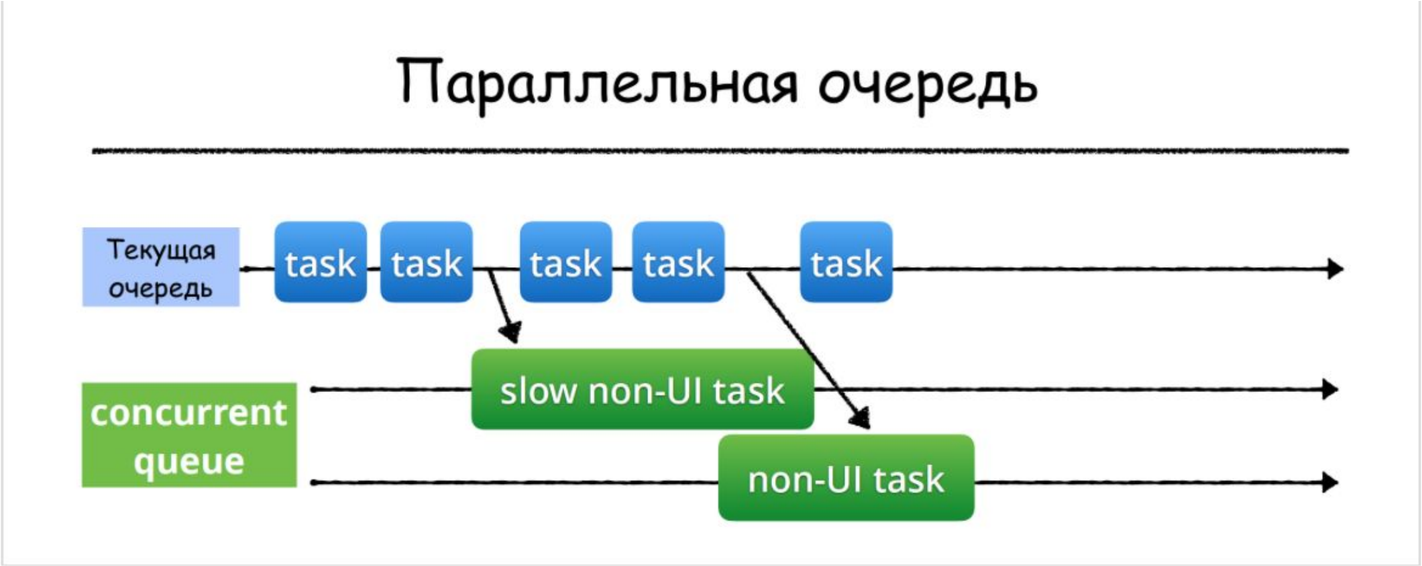


Асинхронная функция **`async`**, возвращает управление на текущую очередь немедленно после запуска задания на выполнение в другой очереди, не ожидая его завершения. Таким образом, асинхронная функция **`async`** не блокирует выполнение заданий на текущей очереди.

«Другой очередью» может оказаться в случае асинхронного выполнения как последовательная (`serial`) очередь:



так и параллельная (`concurrent`) очередь:



Main queue: (any UI tasks)

```
let main = DispatchQueue.main
```

Quos
(Quality of Service)

Global queues: (any non UI tasks)

```
// наивысший приоритет
let userInteractiveQueue = DispatchQueue.global(qos: .userInteractive)

let userInitiatedQueue = DispatchQueue.global(qos: .userInitiated)

let utilityQueue = DispatchQueue.global(qos: .utility)

// самый низкий приоритет
let backgroundQueue = DispatchQueue.global(.background)

// по умолчанию
let defaultQueue = DispatchQueue.global()
```

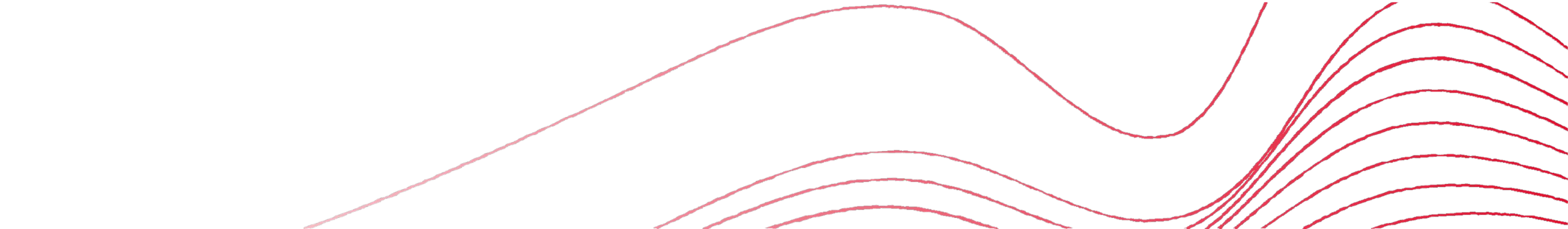
Почти как main, можно запускать ui таски

Для задач не требующих обратной связи с пользователем (очистка бд)

Для задач вроде синхронизации с сервером для бекапа (от минут до часов).

Проблемы многопоточности

- состояние гонки (**`race condition`**) — ошибка проектирования многопоточной системы или приложения, при которой работа системы или приложения зависит от того, в каком порядке выполняются части кода
- инверсия приоритетов (**`priority inversion`**)
- взаимная блокировка (**`deadlock`**) — ситуация в многопоточной системе, при которой несколько потоков находятся в состоянии бесконечного ожидания ресурсов, занятых самими этими потоками



Race condition

Создали стринг переменную со значением 😇

Асинхронно вызвали метод, добавляющий символ 🐔 к переменной (с Sleep(1))

В main-е изменили значение на 🦊

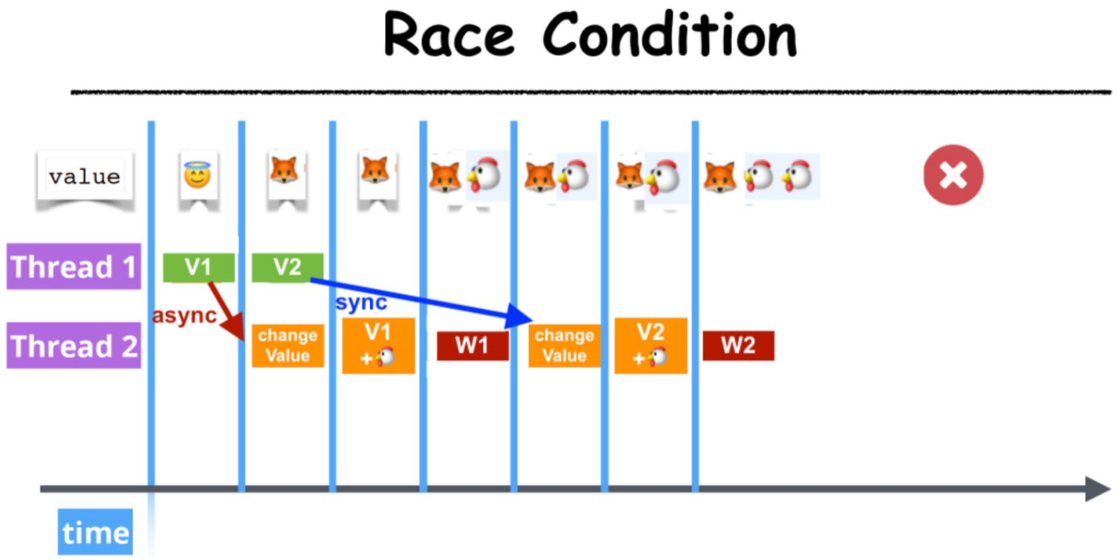
Синхронно вызвали метод, добавляющий символ 🐔 к переменной (с Sleep(1))

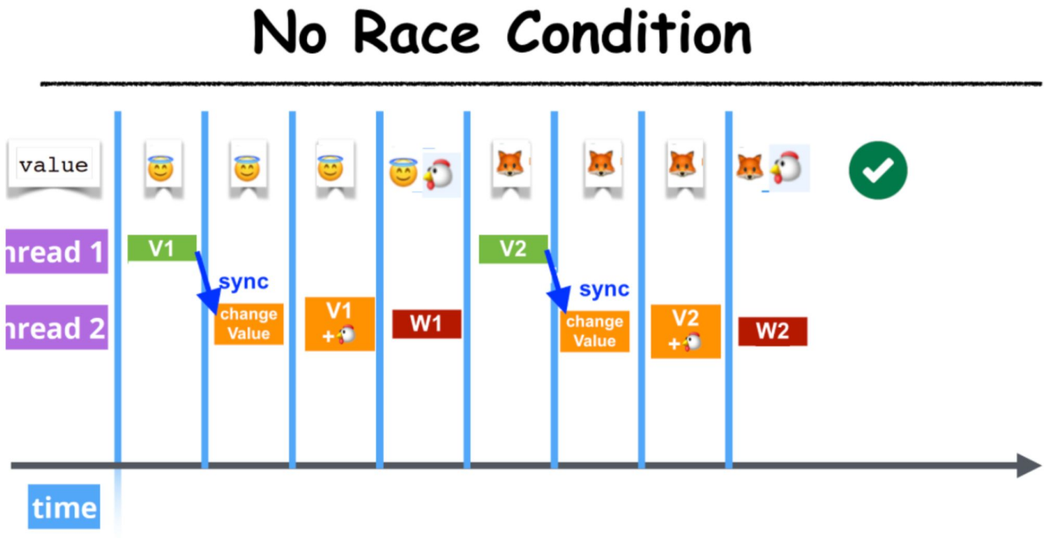
Итог: к моменту асинхронного выполнения функции, она подтянет значение, которое было изменено в синхронном выполнении функции. И мы получим значение переменной 🐔 🦊 🦊

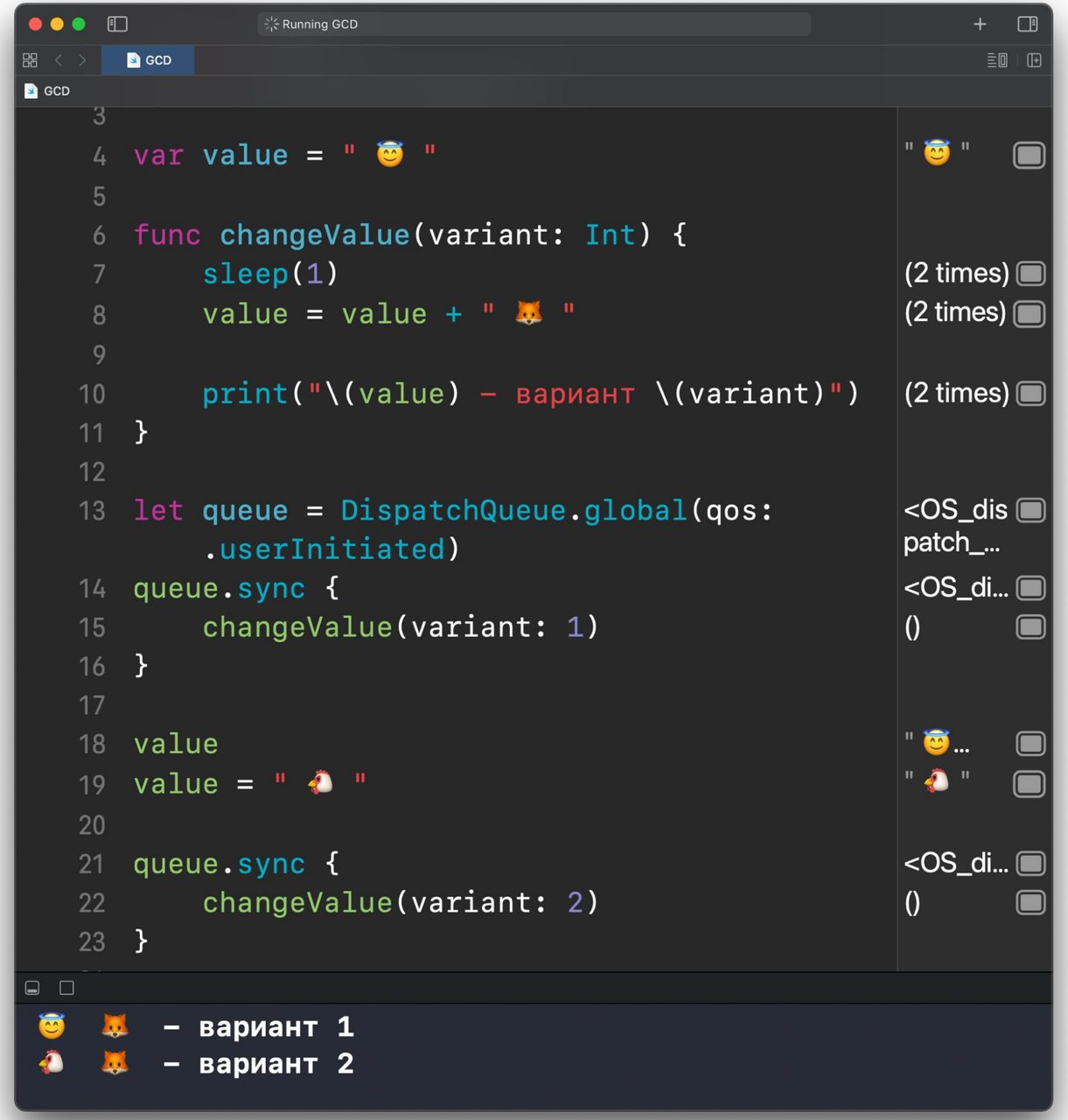
🐔 🦊 – вариант 2
🐔 🦊 🦊 – вариант 1

Если мы будем выполнять код синхронно работая с переменной, race condition не случится.

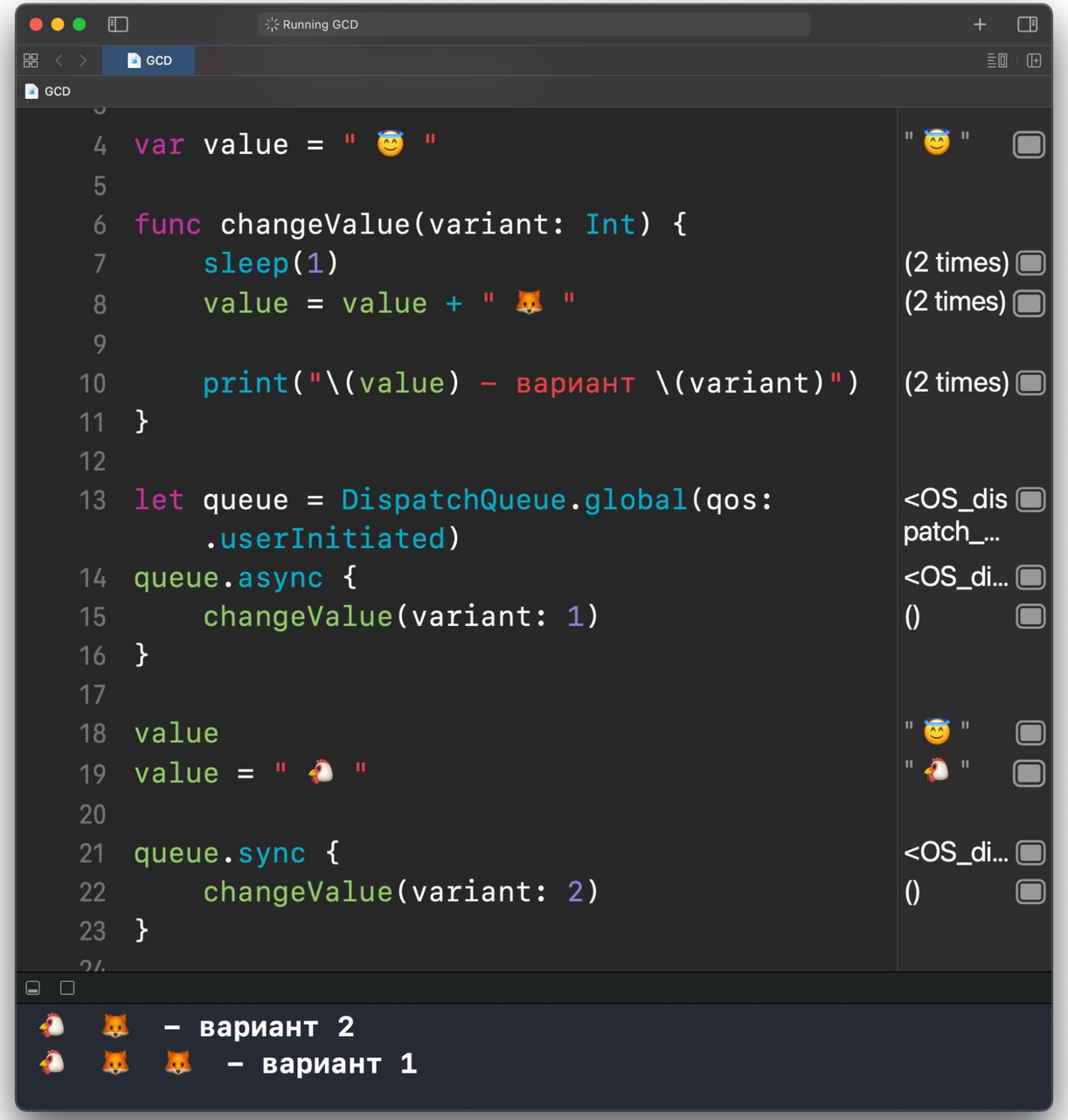
😇 🦊 – вариант 1
🐔 🦊 – вариант 2







No race condition

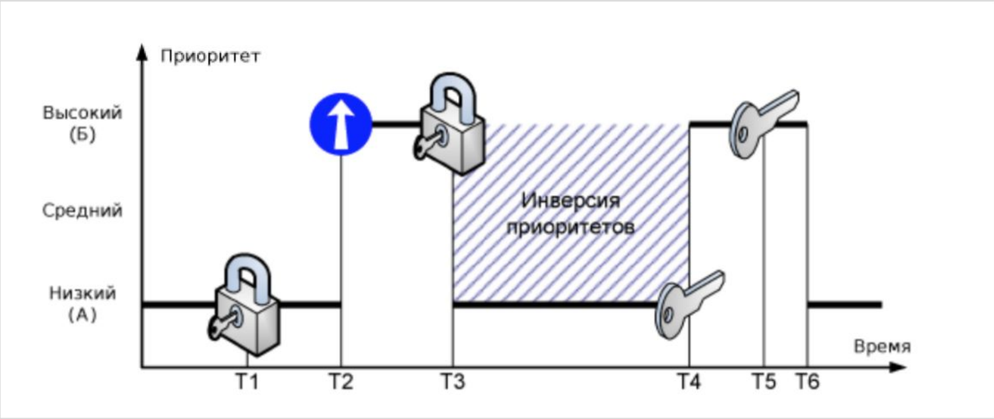


Race condition

Инверсия приоритетов (priority inversion)

Ограниченная инверсия приоритетов

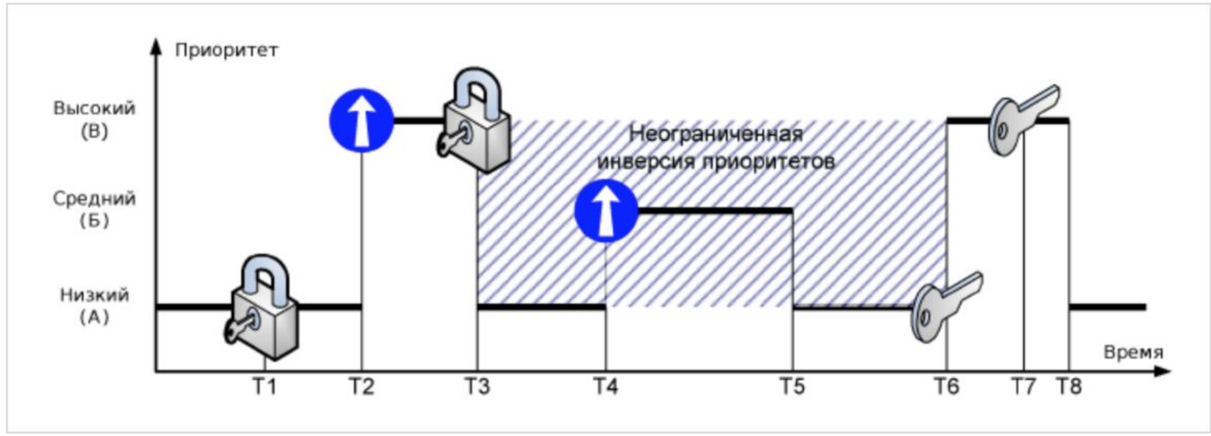
Задача с более высоким приоритетом находится в ожидании в то время как низко приоритетная задача выполняется.



Неограниченная инверсия приоритетов

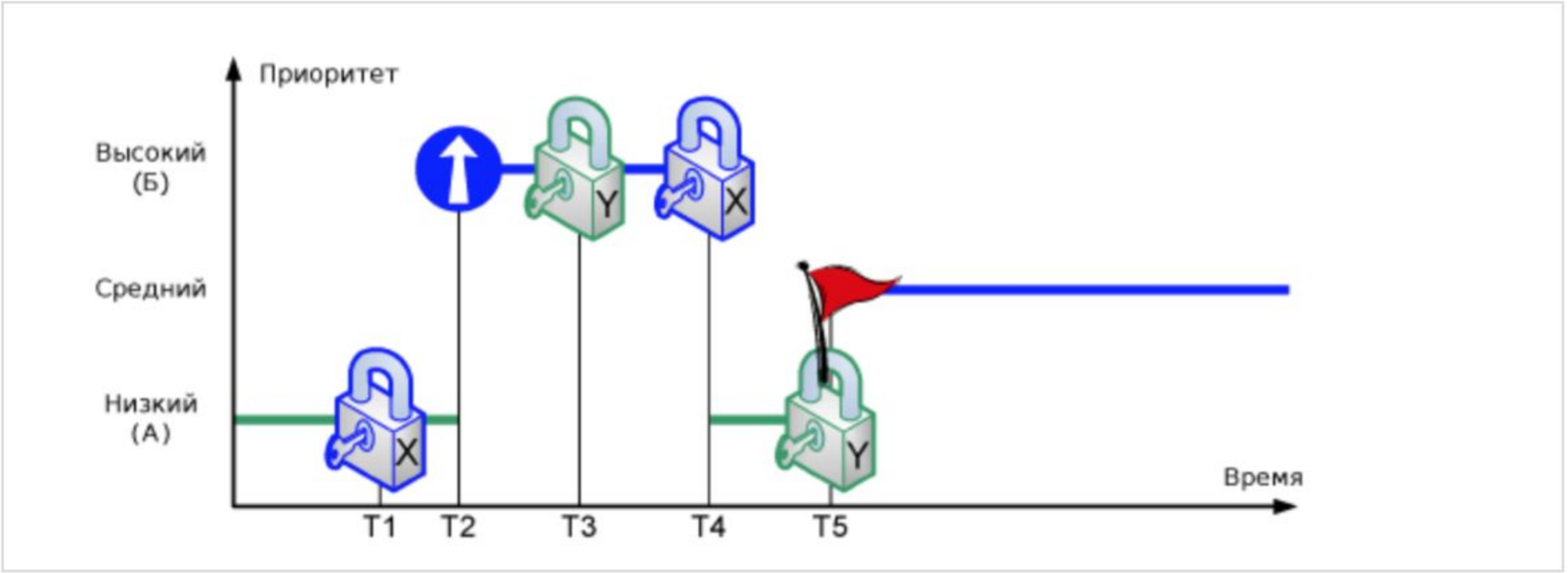
Если ресурс заблокирован задачей (А), а он требуется задаче (В), то наблюдается та же ситуация

— высокоприоритетная задача блокируется. Но допустим, что задача (Б) вытеснила (А), после того как (В) ушла в ожидание ресурса. Задача (Б) ничего не знает о конфликте, поэтому может выполняться сколь угодно долго на промежутке времени (Т5-Т4). Кроме того, помимо (Б) в системе могут быть и другие задачи, с приоритетами больше (А), но меньше (Б). Поэтому длительность периода (Т6-Т3) в общем случае не определена.



Взаимная блокировка (deadlock)

В момент времени T1 задача (А) блокирует ресурс X. Затем в момент времени T2 задачу (А) вытесняет более приоритетная задача (Б), которая в момент времени T3 блокирует ресурс Y. Если задача (Б) попытается заблокировать ресурс X (T4) не освободив ресурс Y, то она будет переведена в состояние ожидания, а выполнение задачи (А) будет продолжено. Если в момент времени T5 задача (А) попытается заблокировать ресурс Y, не освободив X, возникнет состояние взаимной блокировки — ни одна из задач (А) и (Б) не сможет получить управление.

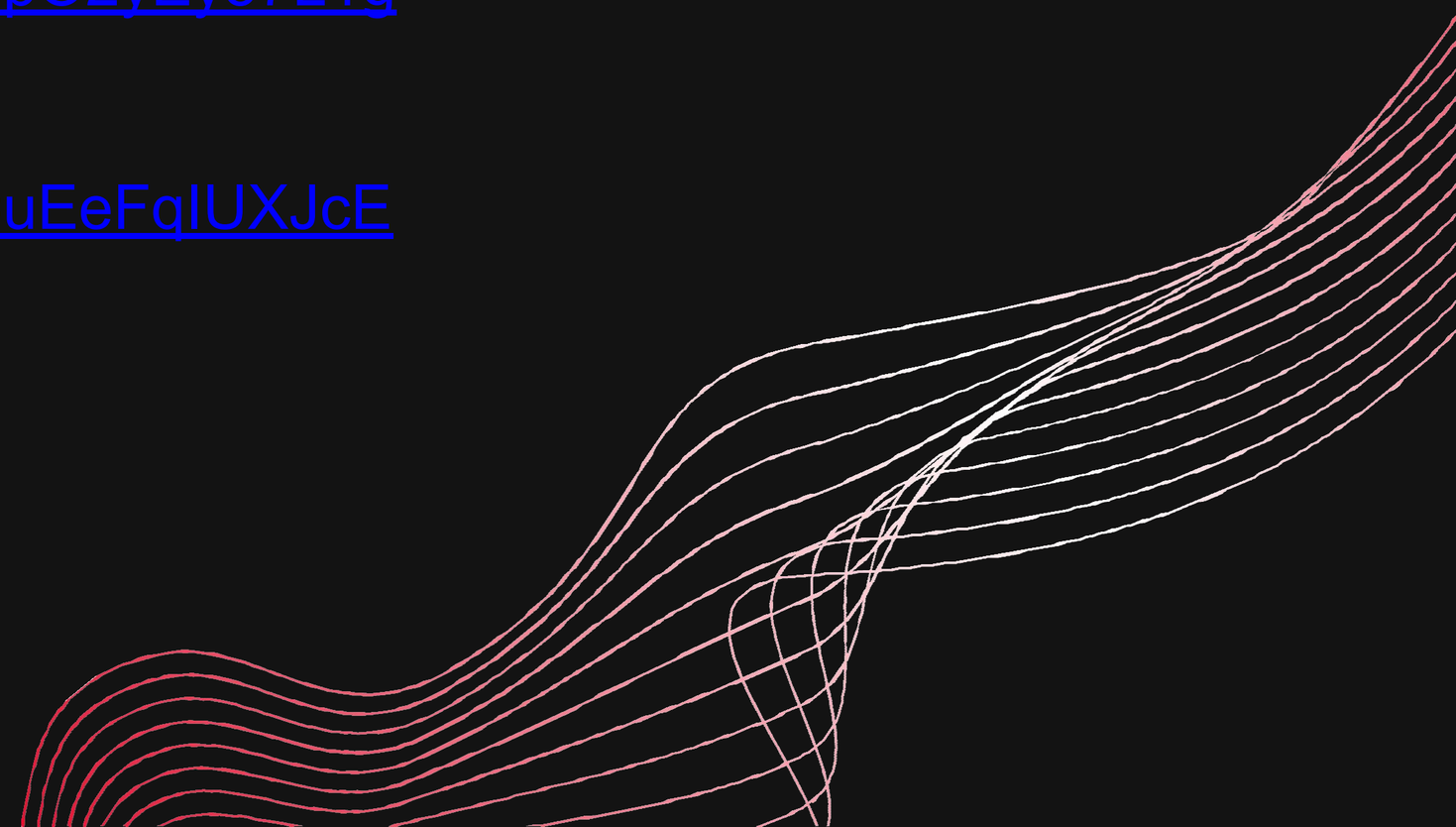


Все ссылки:

Подробная статья о многопоточности:
https://habr.com/ru/post/320152/

Разбор многопоточности на примерах в коде:
https://www.youtube.com/watch?v=pO2yEy57L1g

Каверзные вопрос по GCD:
https://www.youtube.com/watch?v=uEeFqIUXJcE



Web Academy

CREATE YOUR IT FUTURE.

WITH WEB ACADEMY