Skillbox

Многопоточность

Skillbox

Цели модуля

Структура модуля

1. Введение в многопоточность
2. Работа с потоками
3. Проблемы многопоточности
4. Многопоточность в Android

Цели модуля

- узнаете, что такое **многопоточность**, причины ее возникновения и чем отличается **процесс** от **потока**
- научитесь работать с потоками в приложении
- узнаете, с какими **проблемами** можно столкнуться при разработке многопоточных программ
- узнаете, что предоставляет Android для работы с многопоточностью и **взаимодействия между потоками**

Skillbox

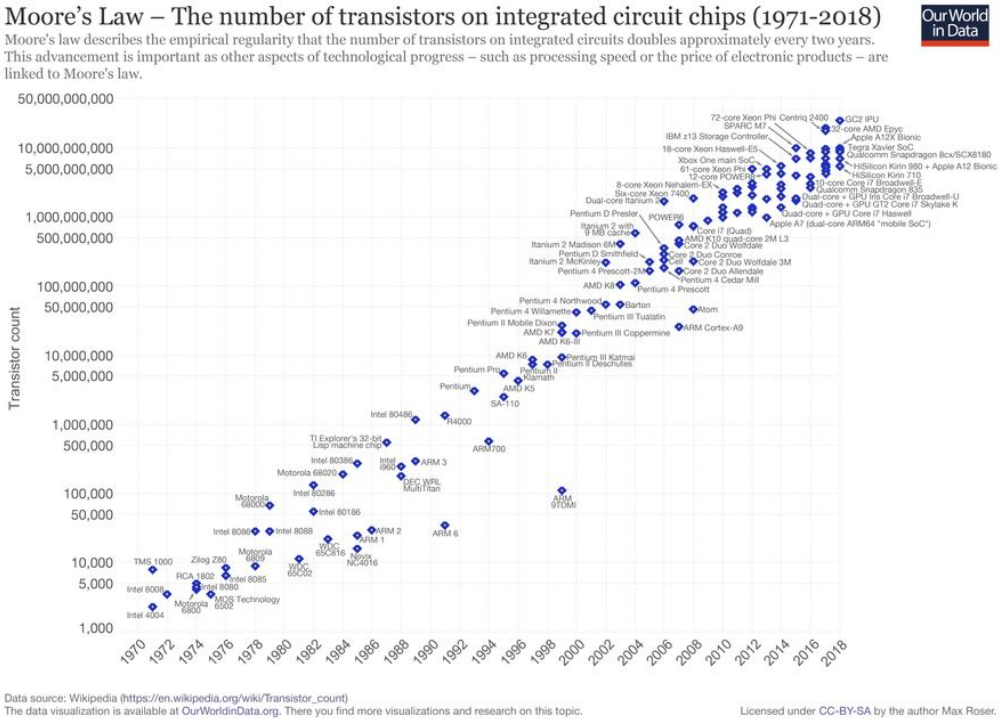
Введение в многопоточность

Цели урока

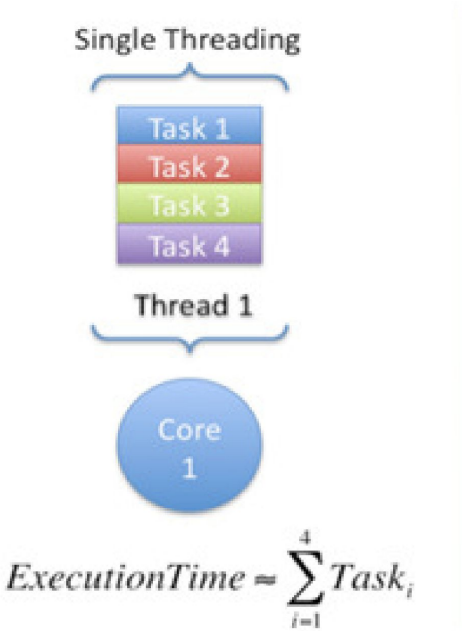
- что такое многопоточность
- причины возникновения параллелизма
- понятия процесса и потока

Закон Мура

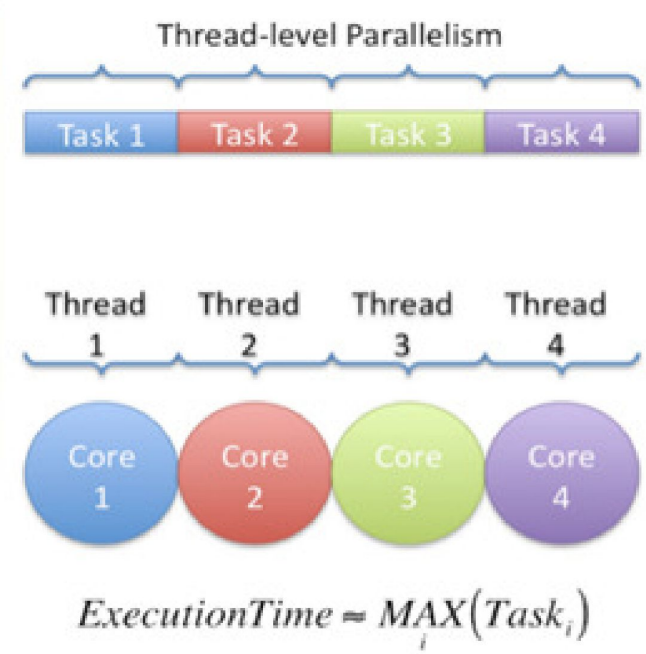
Количество транзисторов на интегральной микросхеме удваивается каждые два года, то есть растет экспоненциально



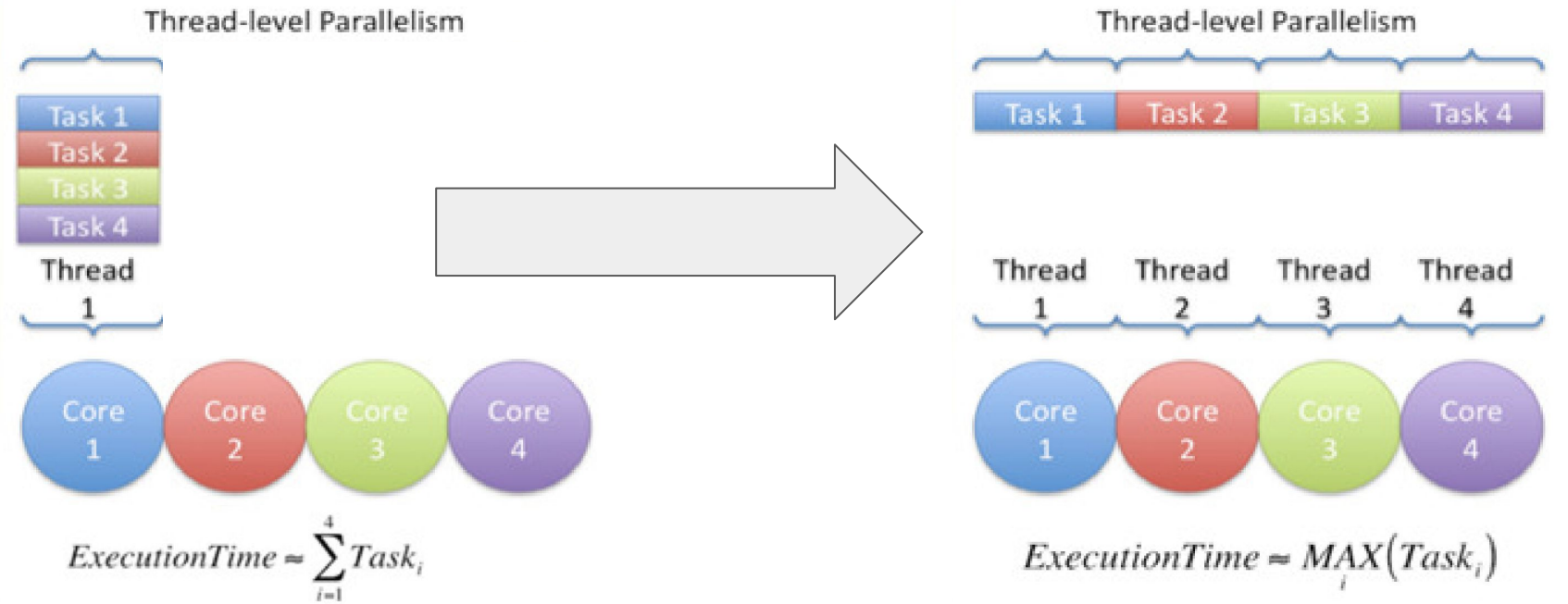
Одноядерные процессоры



Многоядерные процессоры

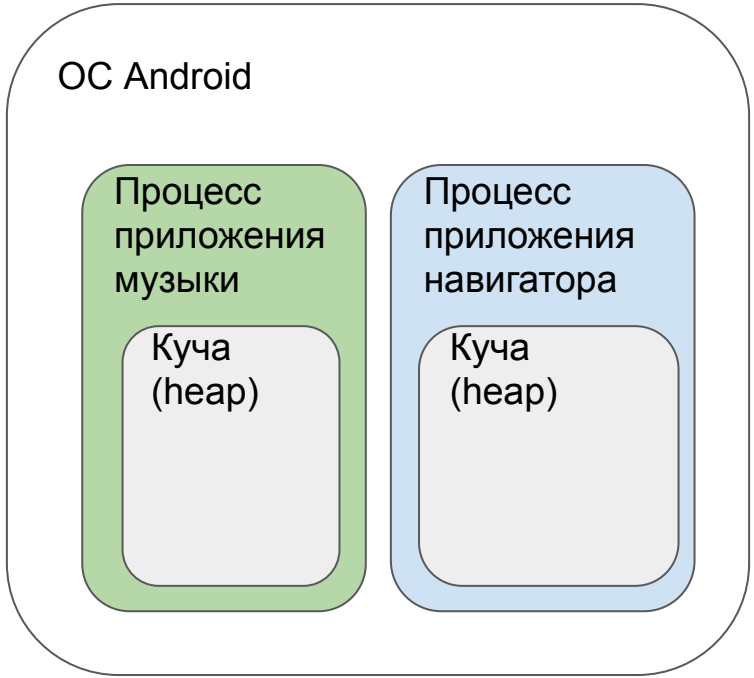


Многопоточные программы

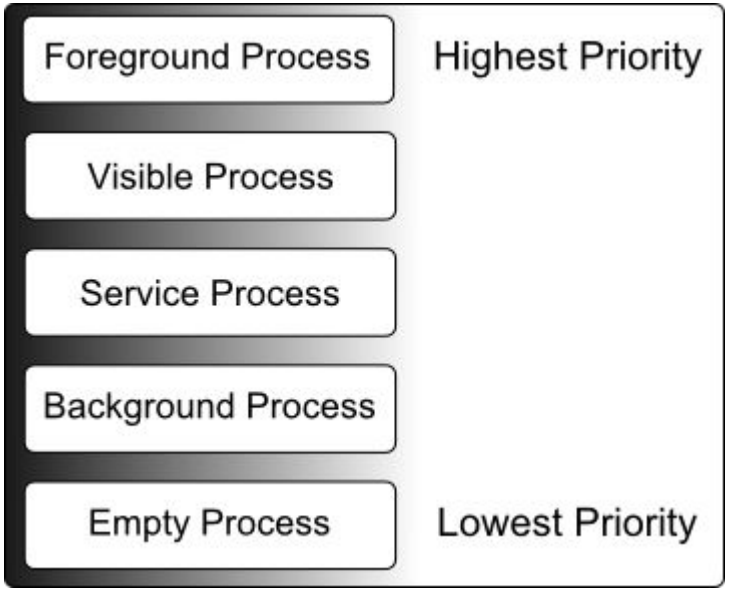


Процессы

- имеет код для исполнения
- имеет свою собственную область памяти (куча)
- приложение - минимум один процесс
- процессы не могут получать доступ к памяти друг друга напрямую

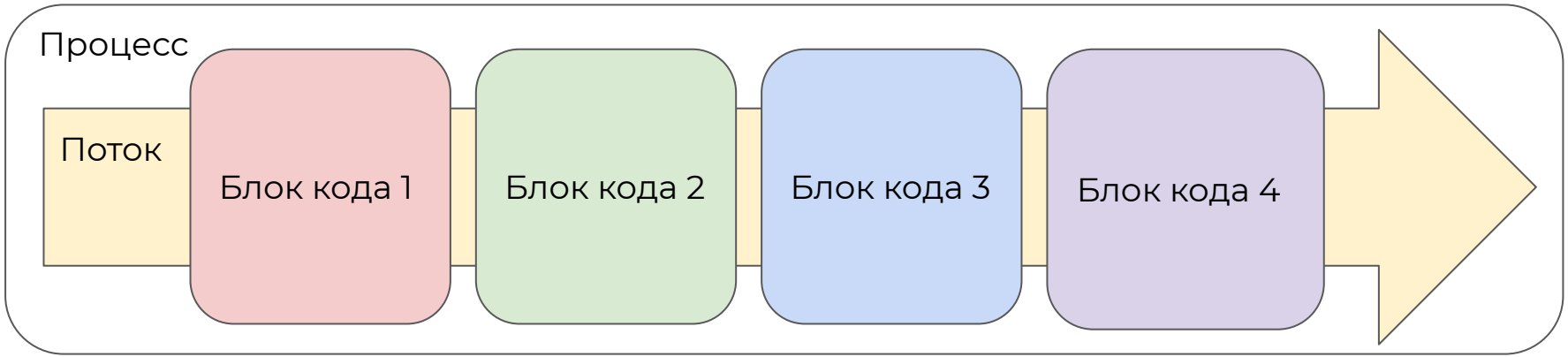


Приоритеты процессов

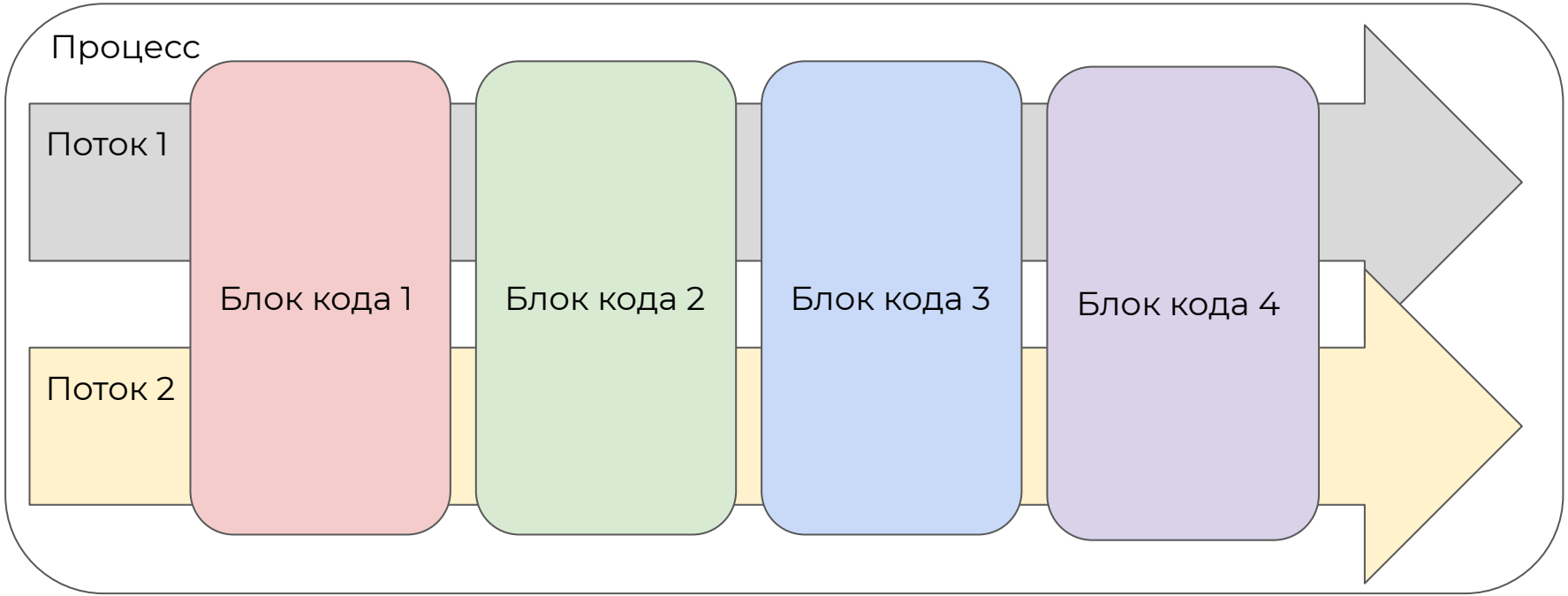


Потоки (threads)

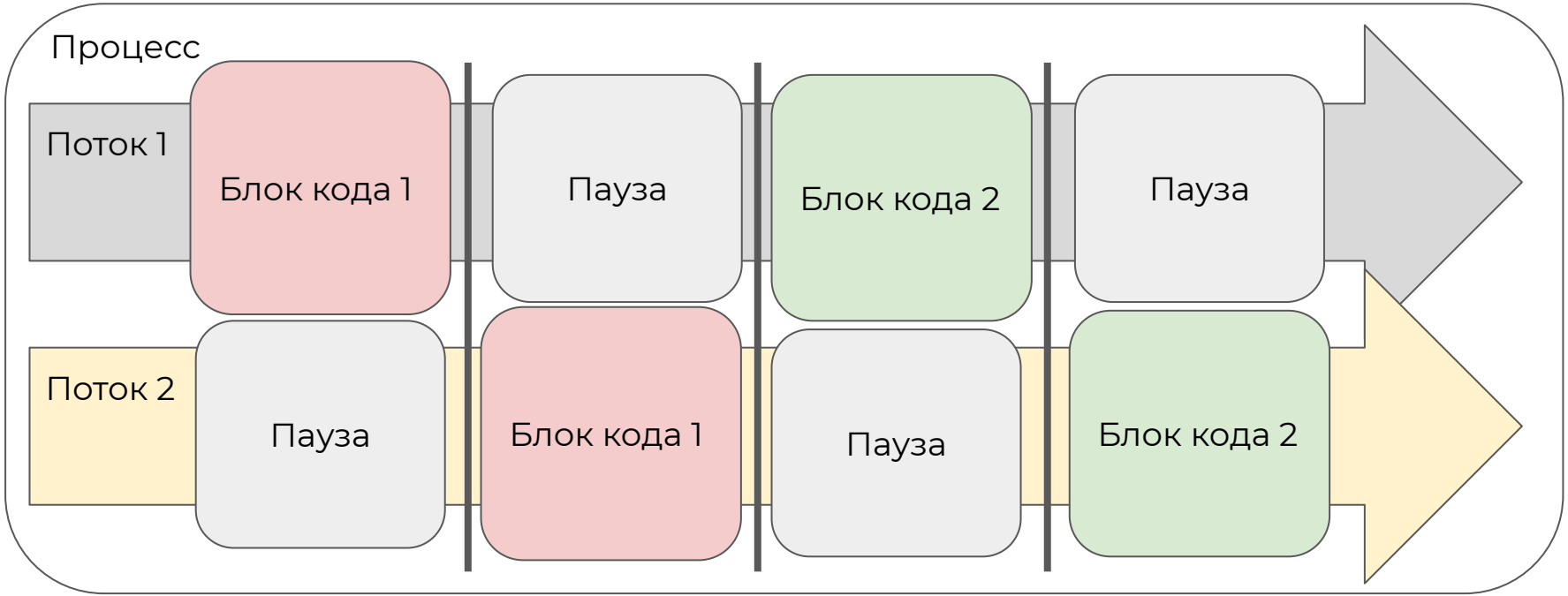
Сущность для выполнения команд и операций



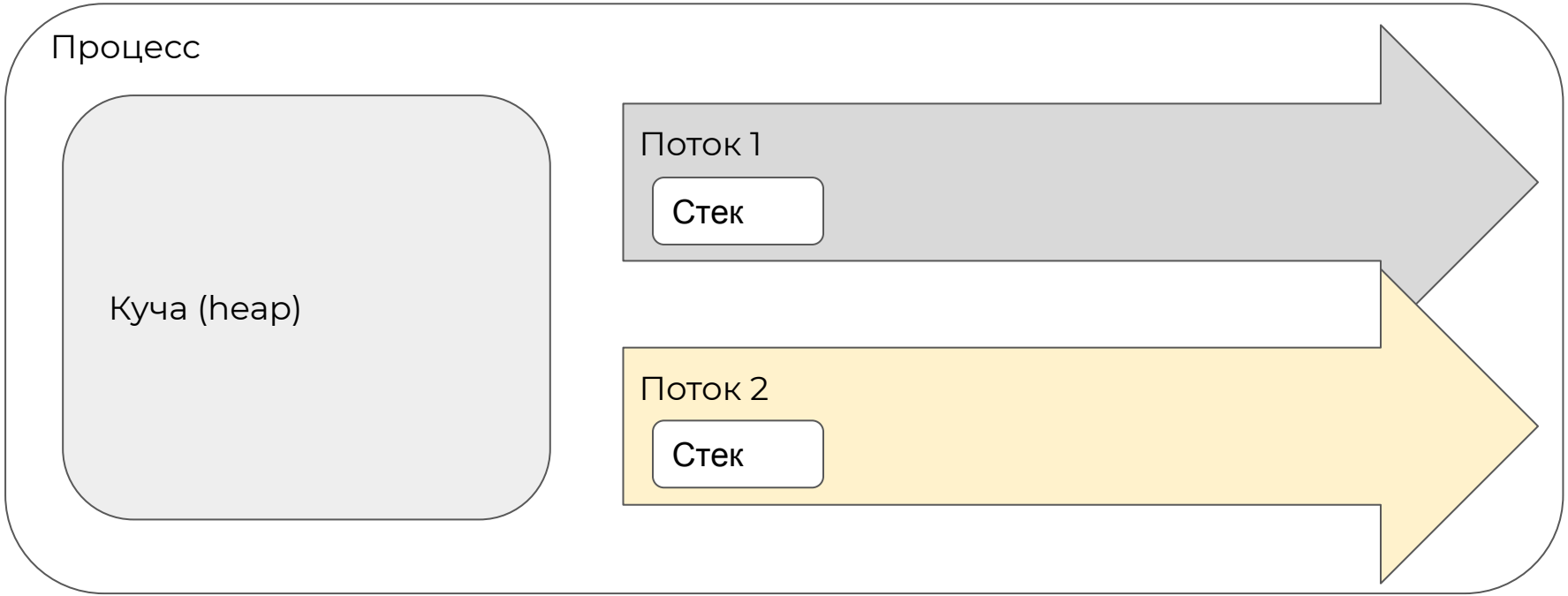
Потоки (threads)



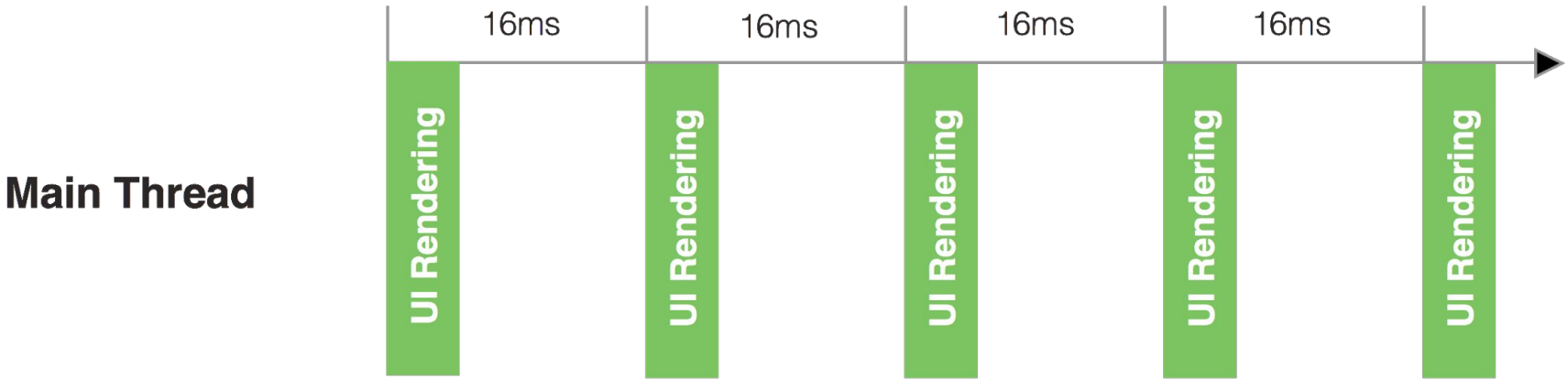
Потоки. 1 core



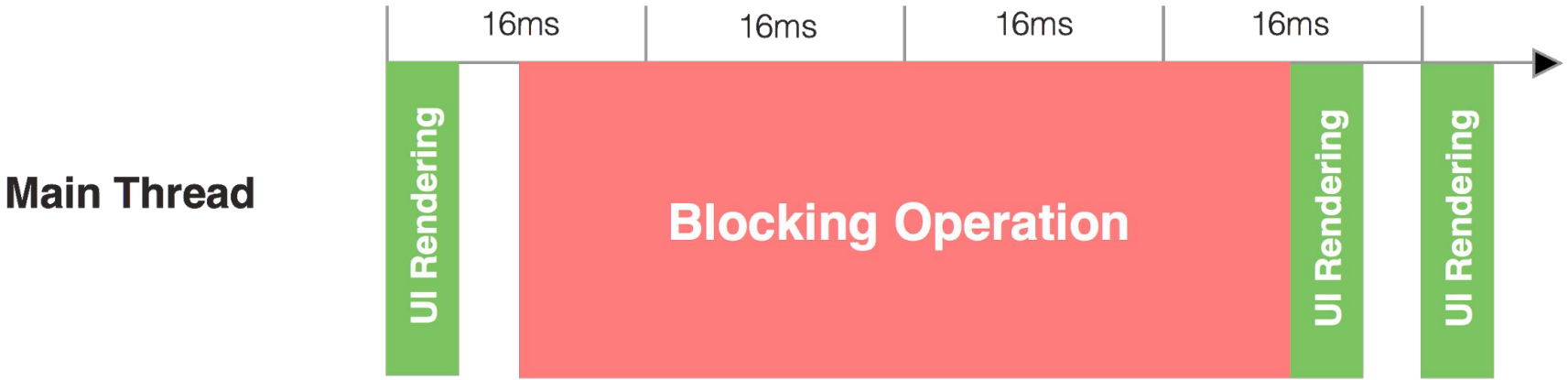
Работа с памятью



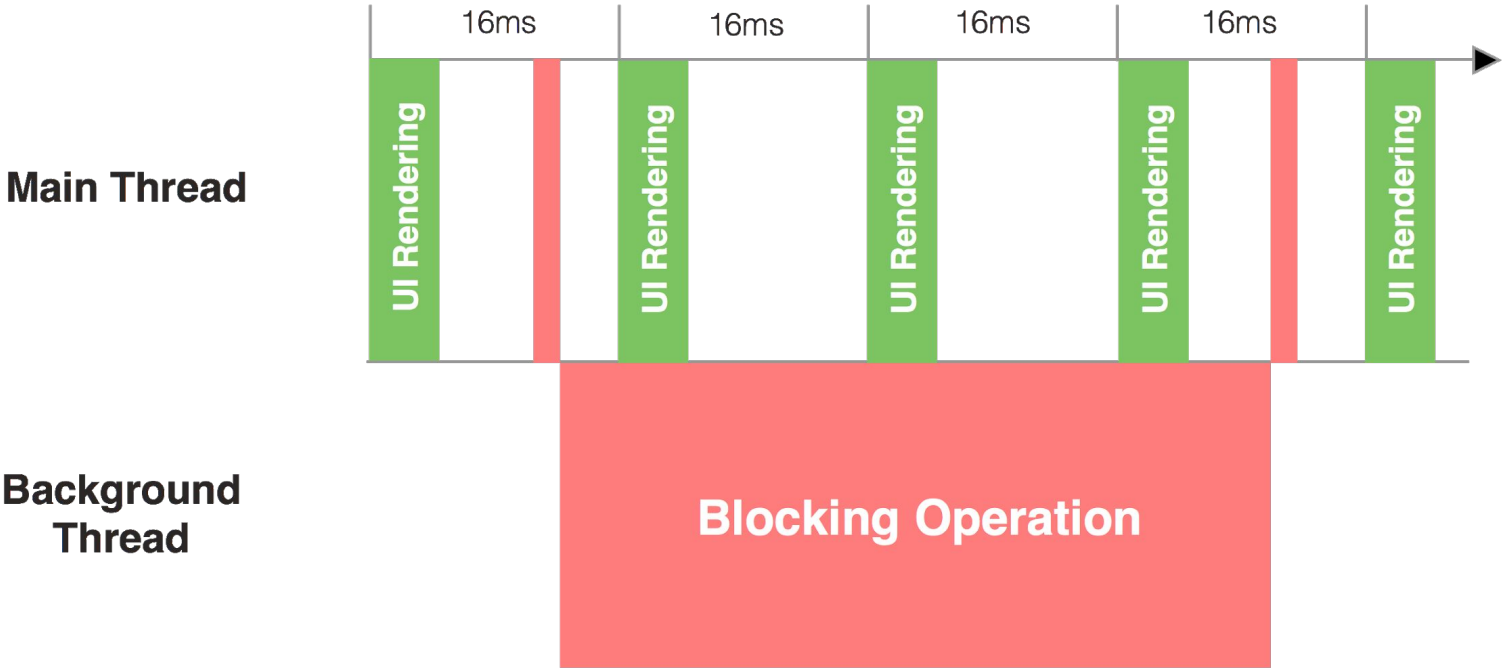
Main thread


Main Thread
16ms
16ms
16ms
16ms
UI Rendering
UI Rendering
UI Rendering
UI Rendering
UI Rendering

Main thread


Main Thread
16ms
16ms
16ms
16ms
UI Rendering
Blocking Operation
UI Rendering
UI Rendering

Фоновый поток



Выводы

- рассмотрели историю развития процессоров
- познакомились с основными понятиями - **процесс и поток**
- рассмотрели пример использования **фоновых потоков** в андроиде

Skillbox

Работа с потоками

Цели урока

- рассмотрим, как работать с **потоками** в приложении
- научимся выносить длительную работу в **фоновые потоки**
- научимся **распараллеливать** работу на несколько потоков

Thread

API для работы с низкоуровневыми потоками

```
Thread().start() //запуск потока
```


```
Thread.currentThread().name // имя текущего потока
Thread.sleep(500) // приостановить выполнение потока
```

Наследование класса Thread

```
class WorkThread: Thread() {
    override fun run() {
        // do something work
    }
}

WorkerThread().start()
```

Интерфейс Runnable

```
public interface Runnable {
    public void run();
}
```

Интерфейс Runnable

```
class WorkRunnable : Runnable {
    override fun run() {
        // do something work
    }
}
Thread(WorkRunnable()).start()

//использование лямбда функции
Thread {
    // do something work
}.start()
```

Выводы

- рассмотрели, как работать с потоками в приложении (**Runnable** и **Thread**)
- научились выносить длительную работу в **фоновые потоки**
- научились получать информацию с фоновых потоков с помощью **колбеков**
- **распараллелили** работу на несколько потоков

Skillbox

Проблемы многопоточности

Цели урока

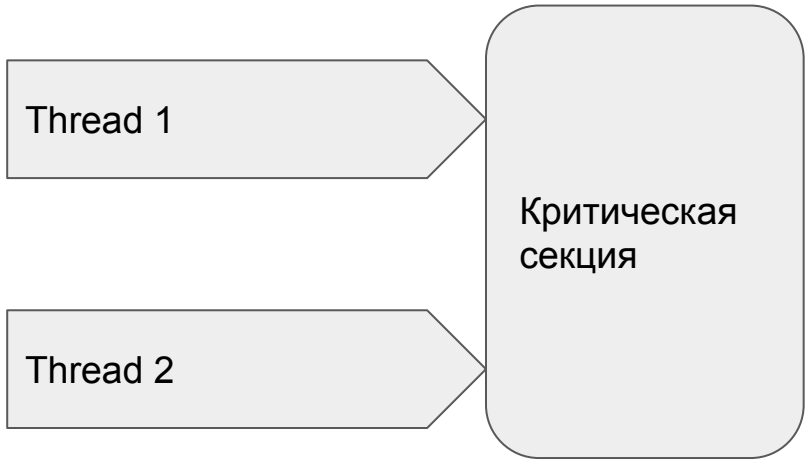
Рассмотрим проблемы многопоточности:

- race condition
- deadlock
- livelock

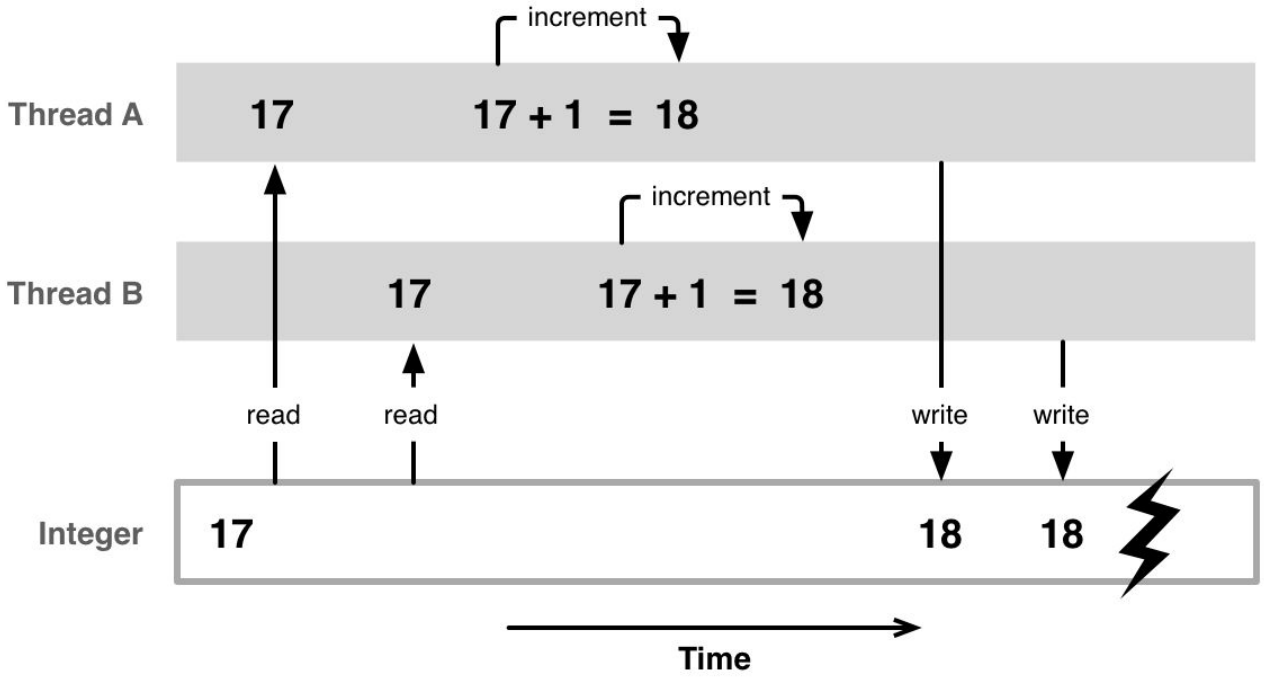
Race condition

Ситуация в многопоточном приложении, при которой правильное поведение программы будет зависеть от порядка выполнения частей кода разными потоками.

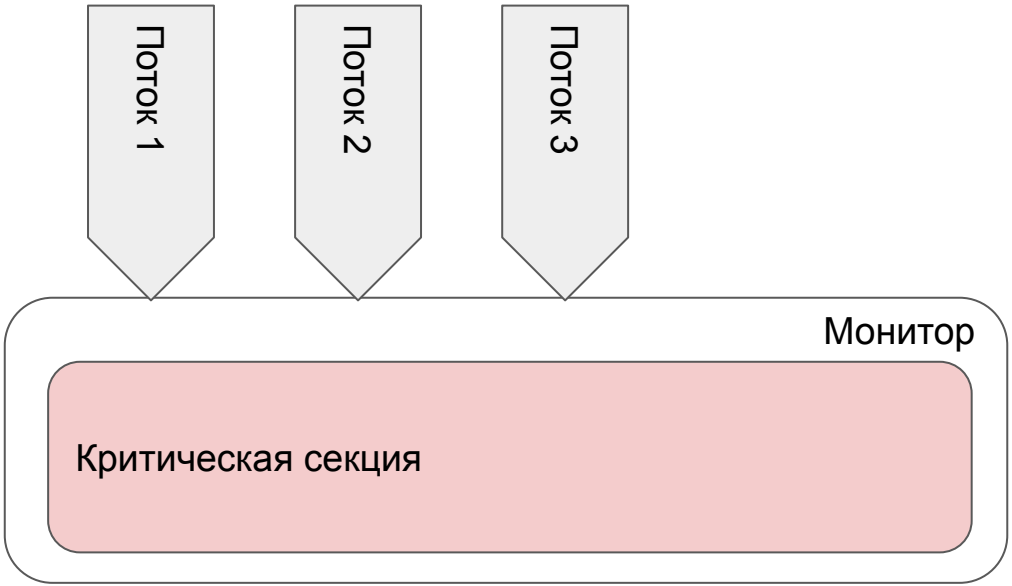
Возникает в критических секциях



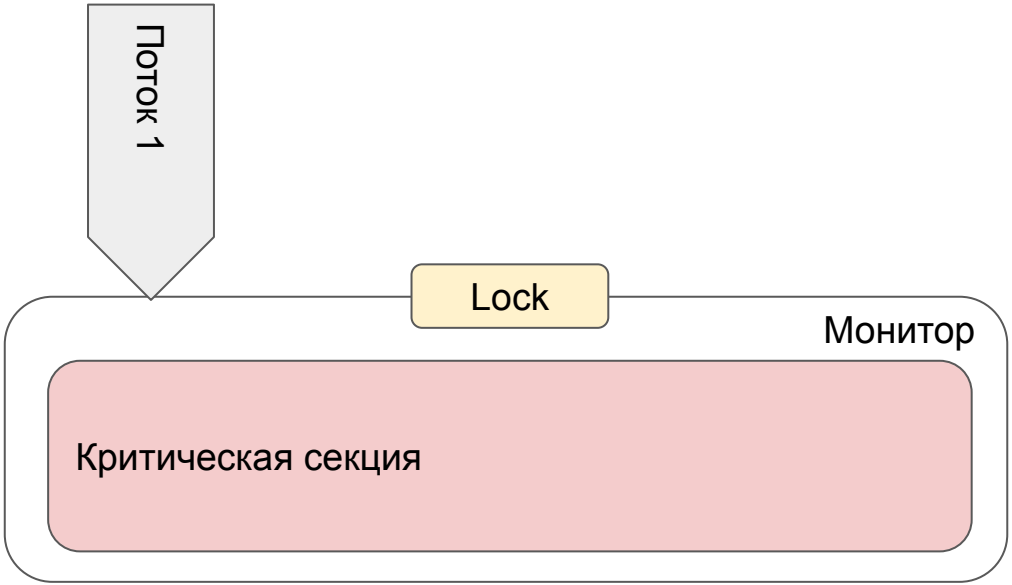
Race condition



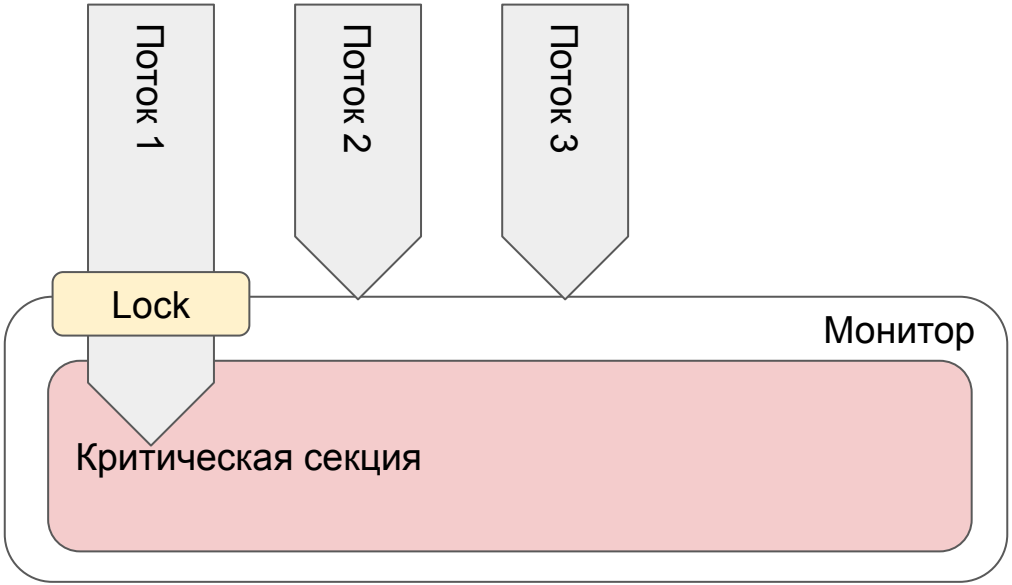
Синхронизация



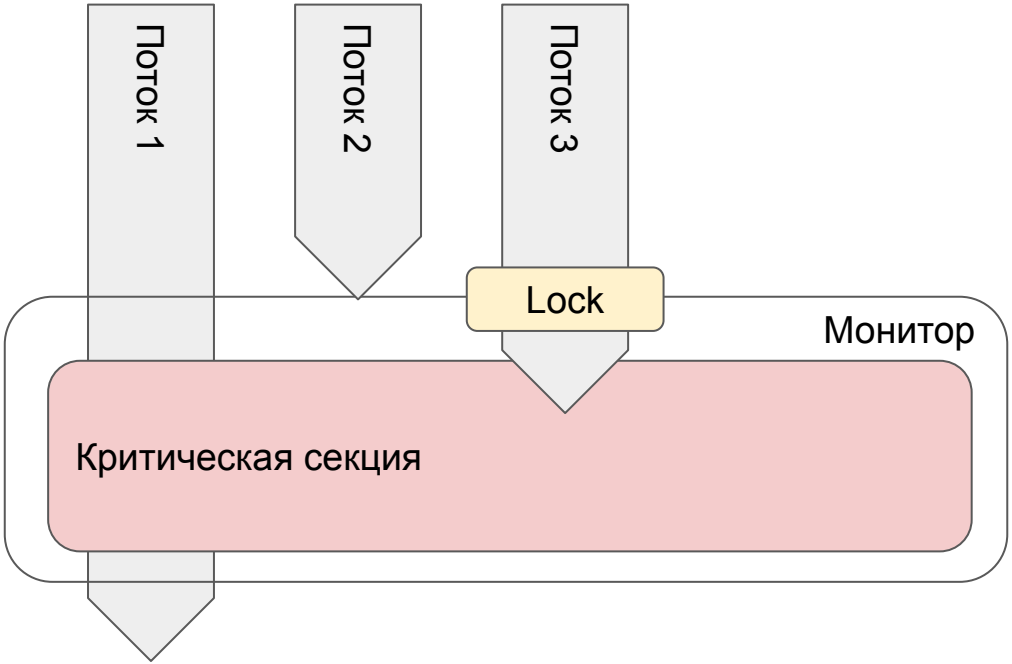
Синхронизация



Синхронизация



Синхронизация



Synchronized

```
@Synchronized
private fun synchronizedMethod() {
    // only one thread at the time
}

@Synchronized
private fun syncronizedMethod2() {
    // only one thread at the time
}
```

Lock = this

Несколько методов - общий lock

Synchronized

```
private fun synchronizedMethod() {
    synchronized(this) {
        // only one thread at the time
    }
}
private fun syncronizedMethod2() {
    synchronized(this) {
        // only one thread at the time
    }
}
```

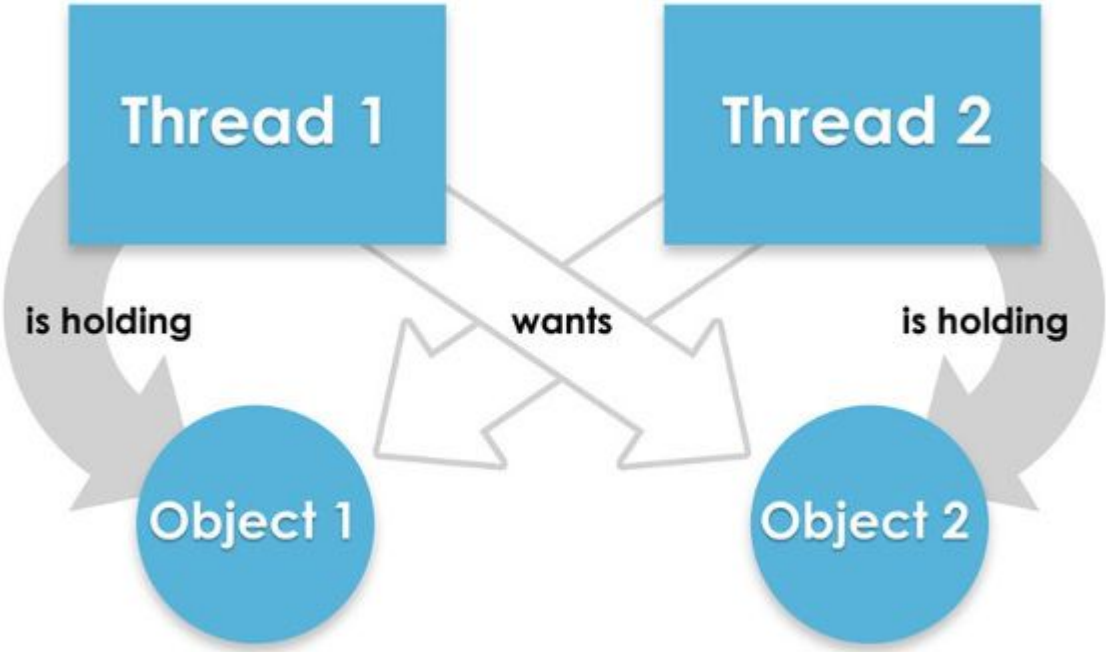
Synchronized

```
private val readLock = Any()
private val writeLock = Any()

private fun read() {
    // multiple threads at the time
    synchronized(readLock) {
        // only one thread at the time
    }
    // multiple threads at the time
}

private fun write() {
    synchronized(writeLock) {
        // only one thread at the time
    }
}
```

Deadlock

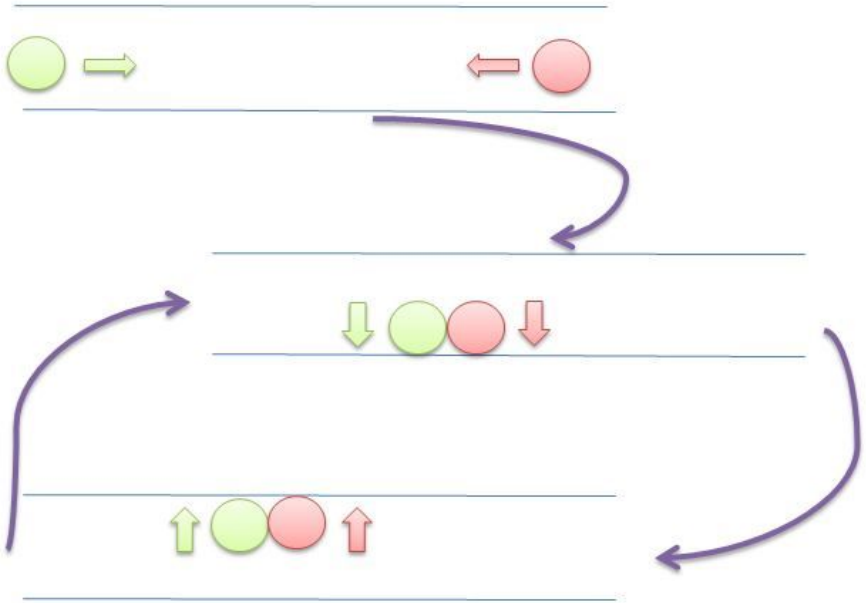

Thread 1
Thread 2
is holding
wants
is holding
Object 1
Object 2

Deadlock

Решение:

- при запросе ресурсов запрашивать сразу все требуемые ресурсы
- при ожидании ресурса потоком отпускать занятые ресурсы
- правилом обратного порядка освобождения ресурсов
- поставить таймаут на ожидание ресурса

Livelock



Livelock

Время	Поток 1	Поток 2
1	Пытается захватить ресурс 1	Пытается захватить ресурс 2
2	Захвачен ресурс 1	Захвачен ресурс 2
3	Пытается захватить ресурс 2	Пытается захватить ресурс 1
4	Отпускает ресурс 1	Отпускает ресурс 2
5	Пытается захватить ресурс 1	Пытается захватить ресурс 2

Выводы

- рассмотрели основные проблемы многопоточности: **race condition, deadlock и livelock**
- узнали способы их решения
- поговорили о **синхронизации**
- научились синхронизировать потоки в **критической секции**

Skillbox

Многопоточность в Android

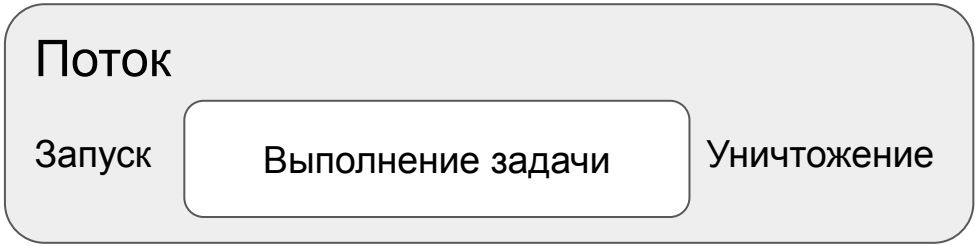
Цели урока

- познакомимся с классами:
  - Looper
  - MessageQueue
  - Handler
  - HandlerThread
- научимся взаимодействовать между потоками

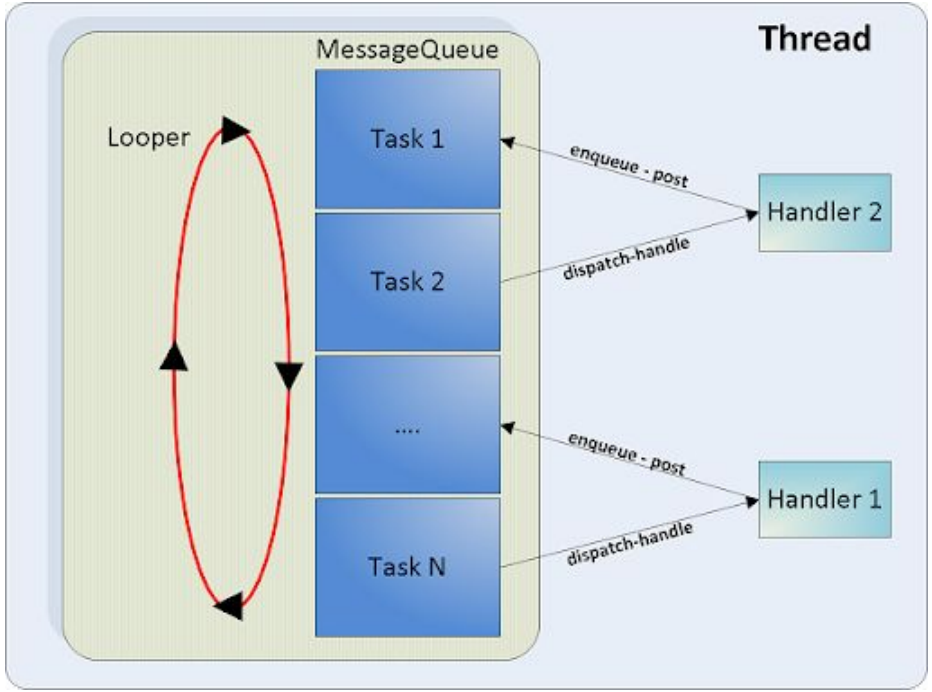
Взаимодействие между потоками

```
userRepository.fetchMovies(movieIds) { movies, fetchTime ->
    //callback runs on backgrouund thread
}
```

ЖЦ потока



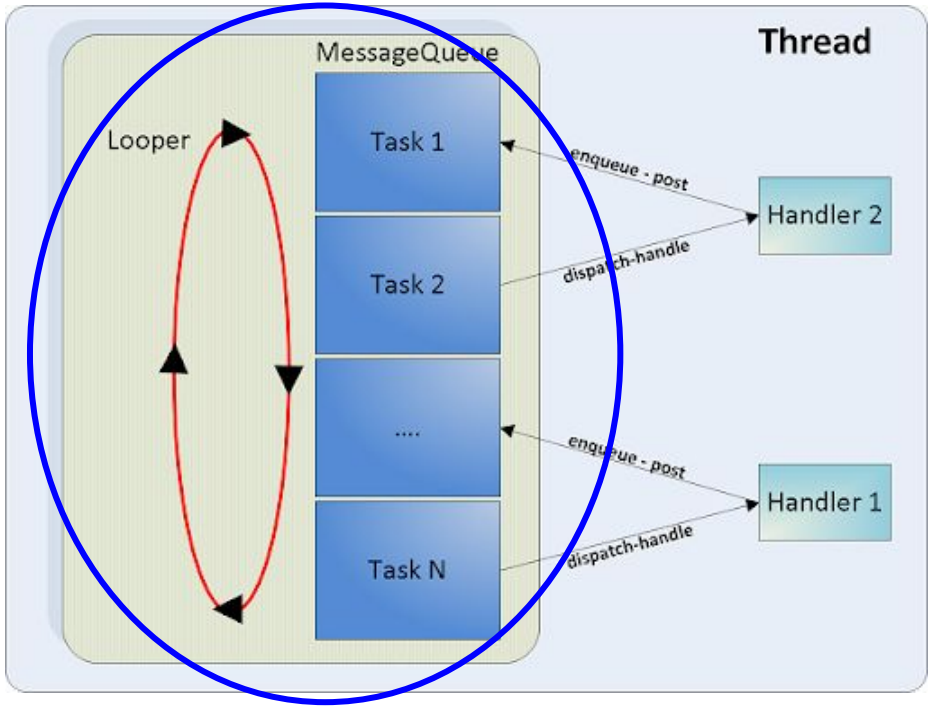
Handler, Looper, MessageQueue



Looper, MessageQueue

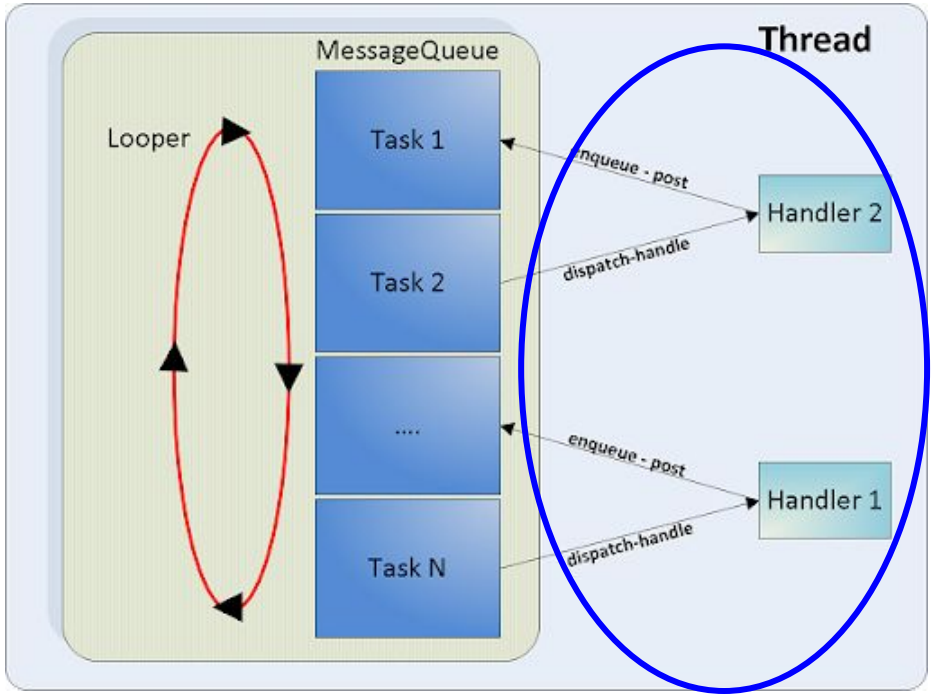
MessageQueue - очередь сообщений или задач для потока.

Looper - это класс, позволяющий вытаскивать задачи из messagequeue в бесконечном цикле.



Handler

- связывается с очередью MessageQueue
- позволяет отправить задачи на выполнение
- позволяет выполнить задачу когда придет время



Выводы

- поговорили о классах:
  - Looper
  - MessageQueue
  - Handler, HandlerThread
- узнали для чего они нужны, как взаимодействуют между собой
- научились отправлять задачи между потоками

Skillbox

Домашнее задание

Домашнее задание

- поработаете на практике с потоками в приложении
- выполните длительную работу в фоновом потоке
- распараллелите работу на несколько потоков
- попытаетесь воспроизвести проблемы многопоточности и устранить их
- научитесь работать с Looper, MessageQueue, Handler