What mom never told you... about multithreading

@fernando_cejas

HELLO!

I am Fernando Cejas

I am here because I love to share experiences and disasters I have made in my professional life...

You can find me at **@fernando_cejas** or **http://fernandocejas.com**

HELLO!


U.S.S.R.
CHINA
JAPAN
U.S.A.
1P
U.S.A.
INDIA
JAPAN
BRAZIL
1P
GUILE
1P
CEJAS

A picture is worth a thousand words

These are MOM and SISTER...

...grabbing a BEER...

A picture is worth a thousand words

May the FORCE of
MULTI-THREADING
be with you...my SON...

Multithreading is not easy...multithreading is hard.

Redesigning your application to run multithreaded on a multicore machine is a little like learning to swim by jumping into the deep end.

The wall is there. We probably won't have any more products without multicore processors [but] we see a lot of problems in parallel programming.

We are not multi-tasking

Humans are capable of doing two things at a time especially when one of those activities is so ingrained that it can be done on autopilot.

...BUT COMPUTERS ARE!!!

Computer can run things in parallel concurrently.
But what does that mean exactly?

Concurrency vs Parallelism
What is the difference?

Concurrent

Concurrency is about dealing with A LOT of things...

Parallel

Parallelism is about doing A LOT of things at ONCE.

This talk is really about THREADING...

and maybe something else...

And not about Android...

Let's review some concepts

Process

It is an instance of a computer program that is being executed. It contains the program code and its current activity.

Thread

It is the smallest sequence of programmed instructions that can be managed independently by a scheduler, which is typically a part of the operating system.

Mutability

A mutable object can be changed after it's created, and an immutable object can't.

Deadlock

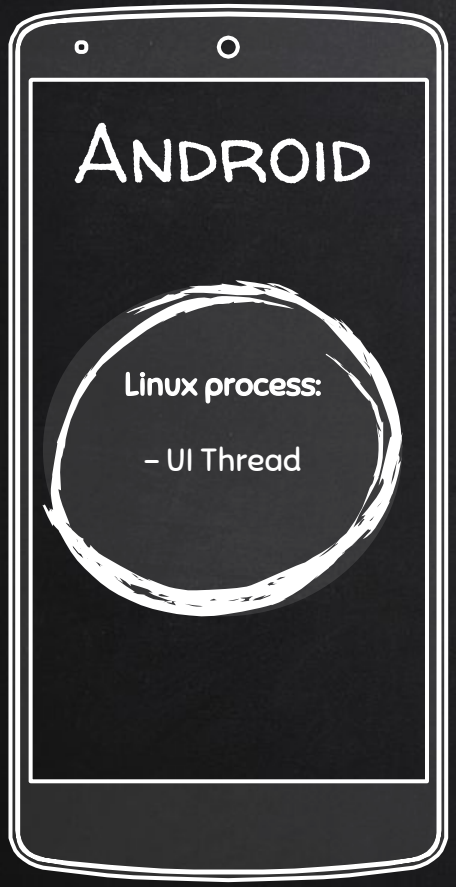
Describes a situation where two or more threads are blocked forever, waiting for each other.

Race Condition

Occurs when two or more threads can access shared data and they try to change it at the same time. Both threads are "racing" to access/change this data.

Starvation

Describes a situation where a thread is unable to gain regular access to shared resources and is unable to make progress.



- When an application component starts and the application does not have any other components running, the Android system starts a new Linux process for the application with a single thread of execution.
- By default, all components of the same application run in the same process and thread (called the "main" thread).
- Android might decide to shutdown a process at some point, when memory is low and required by other processes that are more immediately serving the user. Application components running in the process that's killed are consequently destroyed.

WHY MULTITHREADING?

It is always IMPORTANT to know the fundamentals but...

Responsiveness

Concurrency can ensure improved responsiveness of a program that interacts with the environment.

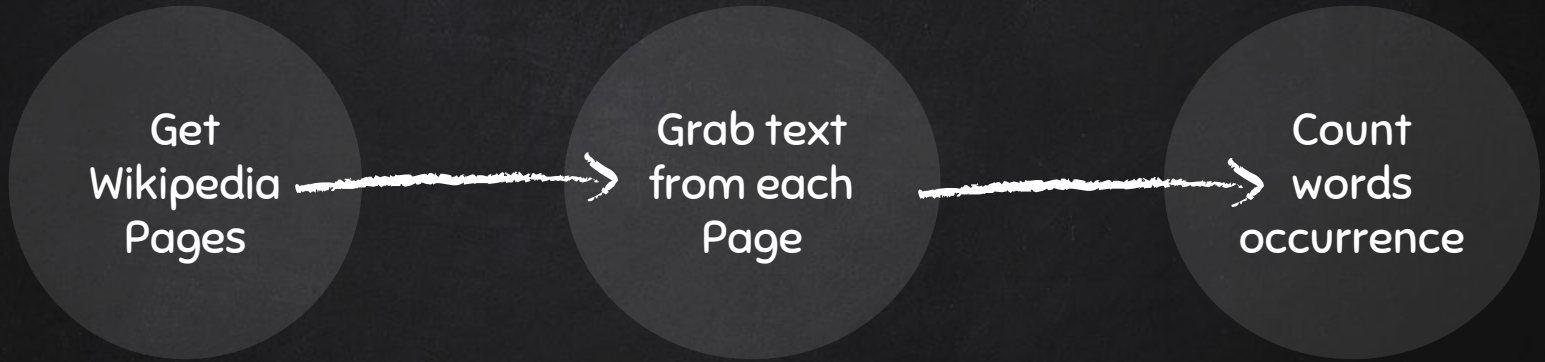
Resources

Use resources in a better and more performant way.

Simplicity

Concurrency can simplify the implementation and maintainability of computer programs. Divide and conquer.

A simple problem on Android



We will be parsing 2 XML files of 30 MB each with Wikipedia content.

✘ https://github.com/android10/Multi-Threading-Samples

FIRST, make code work, ~~THEN make it right~~, THEN make it fast...if it isn't fast enough.

In my defense...FOR LEARNING PURPOSE....

Threads and Locks

```
class SequentialWordCount {
    private val counts: HashMap<String, Int?> = HashMap()

    fun run() {
        val time = measureTimeMillis {
            Thread {
                val pagesOne = Pages(0, 700, Source().wikiPagesBatchOne())
                pagesOne.forEach { page -> Words(page.text).forEach { countWord(it) } }

                val pagesTwo = Pages(0, 700, Source().wikiPagesBatchTwo())
                pagesTwo.forEach { page -> Words(page.text).forEach { countWord(it) } }
            }.start()
        }

        Log.d(LOG_TAG, "Number of elements: ${counts.size}")
        Log.d(LOG_TAG, "Execution Time: $time ms")
    }

    private fun countWord(word: String) {
        when(counts.containsKey(word)) {
            true -> counts[word] = counts[word]?.plus(1)
            false -> counts[word] = 1
        }
    }
}
```

Threads and Locks

Run sequentially

21.678 MILLIS

Execution time

196.681 WORDS FOUND

Two 30 MB XML files processed on an android device

```
com.fernandocejas.sample.threading.threads.SequentialWordCount: Number of elements: 196681
com.fernandocejas.sample.threading.threads.SequentialWordCount: Execution Time: 21678 ms
```

```
class TwoThreadsWordCount {
    private val counts: ConcurrentHashMap<String, Int?> = HashMap()

    fun run() {
        val time = measureTimeMillis {
            val one = Thread {
                val pagesOne = Pages(0, 700, Source().wikiPagesBatchOne())
                pagesOne.forEach { page -> Words(page.text).forEach { countWord(it) } }
            }
            val two = Thread {
                val pagesTwo = Pages(0, 700, Source().wikiPagesBatchTwo())
                pagesTwo.forEach { page -> Words(page.text).forEach { countWord(it) } }
            }
            one.start(); two.start(); one.join(); two.join()
        }
        Log.d(LOG_TAG, "Number of elements: ${counts.size}")
        Log.d(LOG_TAG, "Execution Time: $time ms")
    }

    private fun countWord(word: String) {
        when(counts.containsKey(word)) {
            true -> counts[word] = counts[word]?.plus(1)
            false -> counts[word] = 1
        }
    }
}
```

Threads and Locks

Run Two Threads

18.427 MILLIS

Execution time

196.681 WORDS FOUND

Two 30 MB XML files processed on an android device

```
com.fernandocejas.sample.threading.threads.TwoThreadsWordCount: Number of elements:
196.681
com.fernandocejas.sample.threading.threads.TwoThreadsWordCount: Execution Time: 18427 ms
```

Rx Java

RxJava is more than a framework for dealing with multithreading. Concurrency is ONLY one of its features.

Use the best tool for the right job.

What are RxJava Schedulers?

```
Observable.just("Hello World")
.subscribeOn(Schedulers.computation())
.observeOn(Schedulers.UI)
```

- ✘ If you want to introduce multithreading into your cascade of Observable operators, you can do so by instructing those operators (or particular Observables) to operate on particular Schedulers.

```
class SequentialWordCount {
    private val counts: HashMap<String, Int?> = HashMap()

    fun run() {
        val startTime = System.currentTimeMillis()
        val observable = Observable.fromCallable {
            val pagesOne = Pages(0, 700, Source().wikiPagesBatchOne())
            pagesOne.forEach { page -> Words(page.text).forEach { countWord(it) } }

            val pagesTwo = Pages(0, 700, Source().wikiPagesBatchTwo())
            pagesTwo.forEach { page -> Words(page.text).forEach { countWord(it) } }
        }

        observable
                .doOnComplete { logData(System.currentTimeMillis() - startTime) }
                .subscribeOn(Schedulers.single())
                .subscribe()
    }

    private fun countWord(word: String) {
        when(counts.containsKey(word)) {
            true -> counts[word] = counts[word]?.plus(1)
            false -> counts[word] = 1
        }
    }
}
```

RxJava 2

Run sequentially

20.920 MILLIS

Execution time

196.681 WORDS FOUND

Two 30 MB XML files processed on an android device

```
com.fernandocejas.sample.threading.rxjava.SequentialWordCount: Number of elements: 196681
com.fernandocejas.sample.threading.rxjava.SequentialWordCount: Execution Time: 20920 ms
```

```
class TwoThreadsWordCount {
    private val counts: ConcurrentHashMap<String, Int?> = ConcurrentHashMap()

    fun run() {
        val startTime = System.currentTimeMillis()
        val observablePagesOne = Observable.fromCallable {
            val pagesOne = Pages(0, 700, Source().wikiPagesBatchOne())
            pagesOne.forEach { page -> Words(page.text).forEach { countWord(it) } }
        }.subscribeOn(Schedulers.newThread())
        val observablePagesTwo = Observable.fromCallable {
            val pagesTwo = Pages(0, 700, Source().wikiPagesBatchTwo())
            pagesTwo.forEach { page -> Words(page.text).forEach { countWord(it) } }
        }.subscribeOn(Schedulers.newThread())

        observablePagesOne.mergeWith(observablePagesTwo)
                .doOnComplete { logData(System.currentTimeMillis() - startTime) }
                .subscribe()
    }

    private fun countWord(word: String) {
        when(counts.containsKey(word)) {
            true -> counts[word] = counts[word]?.plus(1)
            false -> counts[word] = 1
        }
    }
}
```

RxJava 2

Run Two Threads

17.256 MILLIS

Execution time

196.681 WORDS FOUND

Two 30 MB XML files processed on an android device

```
com.fernandocejas.sample.threading.rxjava.TwoThreadsWordCount: Number of elements: 196681
com.fernandocejas.sample.threading.rxjava.TwoThreadsWordCount: Execution Time: 17256 ms
```

Kotlin Coroutines

What are Kotlin coroutines?

```
fun main(args: Array<String>) = runBlocking {
    val job = launch(CommonPool) {
        val result = suspendingFunction()
        println("$result")
    }
    println("The result: ")
    job.join()
}
>> prints "The result: 5"
```

- ✘ Coroutines are light-weight threads. A lightweight thread means it doesn't map on native thread, so it doesn't require context switching on processor, so they are faster.
- ✘ They are a way to write asynchronous code sequentially. Instead of running into callback hells, you write your code lines one after the other.

```
class SequentialWordCount {
    private val counts: HashMap<String, Int?> = HashMap()

    fun run() {
        launch(newSingleThreadContext("myThread")) {
            val startTime = System.currentTimeMillis()
            counter()
            logData(System.currentTimeMillis() - startTime)
        }
    }

    private suspend fun counter() {
        val pagesOne = Pages(0, 700, Source().wikiPagesBatchOne())
        pagesOne.forEach { page -> Words(page.text).forEach { countWord(it) } }

        val pagesTwo = Pages(0, 700, Source().wikiPagesBatchTwo())
        pagesTwo.forEach { page -> Words(page.text).forEach { countWord(it) } }
    }

    private fun countWord(word: String) {
        when(counts.containsKey(word)) {
            true -> counts[word] = counts[word]?.plus(1)
            false -> counts[word] = 1
        }
    }
}
```

Kotlin Coroutines

Run sequentially

20.958 MILLIS

Execution time

196.681 WORDS FOUND

Two 30 MB XML files processed on an android device

```
com.fernandocejas.sample.threading.coroutines.SequentialWordCount: Number of elements: 196681
com.fernandocejas.sample.threading.coroutines.SequentialWordCount: Execution Time: 20958 ms
```

```
class TwoThreadsWordCount {
    private val counts: ConcurrentHashMap<String, Int?> = ConcurrentHashMap()

    fun run() {
        val poolContext = newFixedThreadPoolContext(2, "ThreadPool")
        launch(poolContext) {
            val time = measureTimeMillis {
                val one = async(poolContext) { counterPages1() }
                val two = async(poolContext) { counterPages2() }
                one.await()
                two.await()
            }
            logData(time)
        }
    }

    private suspend fun counterPages1() {
        val pagesOne = Pages(0, 700, Source().wikiPagesBatchOne())
        pagesOne.forEach { page -> Words(page.text).forEach { countWord(it) } }
    }

    private suspend fun counterPages2() {
        val pagesTwo = Pages(0, 700, Source().wikiPagesBatchTwo())
        pagesTwo.forEach { page -> Words(page.text).forEach { countWord(it) } }
    }
}
```

Kotlin Coroutines

Run Two Threads

18.980 MILLIS

Execution time

196.681 WORDS FOUND

Two 30 MB XML files processed on an android device

```
com.fernandocejas.sample.threading.coroutines.TwoThreadsWordCount: Number of elements: 196681
com.fernandocejas.sample.threading.coroutines.TwoThreadsWordCount: Execution Time: 18980 ms
```

Round 1 Results

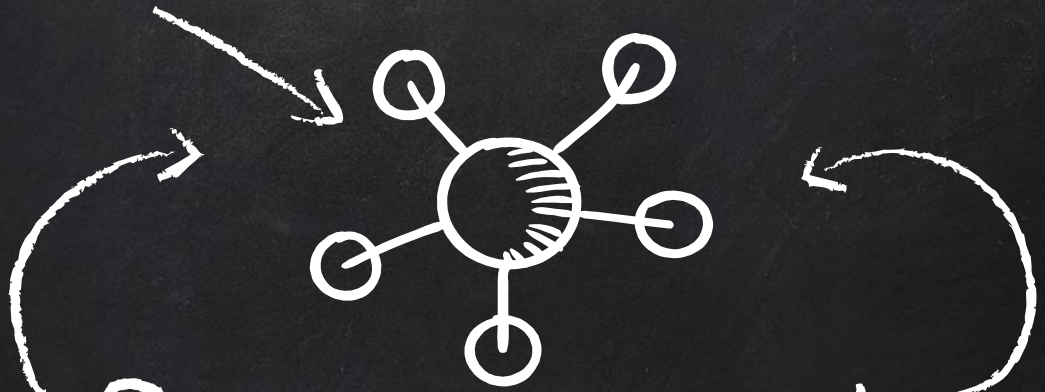
	Single Thread	Two Threads	Better?
Threads and Locks	21.678 ms	18.427 ms	???
RxJava 2	20.920 ms	17.256 ms	???
Kotlin Coroutines	20.958 ms	18.980 ms	???

Adding one thread does not have a big impact...
What is really going on?

Hypothesis: ???

We can do better!

- ✘ Producer – Consumer pattern?
- ✘ Divide and Conquer?
- ✘ Reusing Threads?
- ✘ Other synchronized collections?



Some advice!

- ✘ Analyze the problem.
- ✘ Verify your assumptions.
- ✘ Measure, measure, measure.
- ✘ Measure, measure, measure.

Hypothesis 1: Am I using the right concurrent collection for storing data?

Concurrent collections?

```
class TwoThreadsWordCount {
    private val counts: ConcurrentHashMap<String, Int?> = ConcurrentHashMap()

    fun run() {
        val startTime = System.currentTimeMillis()
        val observablePagesOne = Observable.fromCallable {
            val pagesOne = Pages(0, 5000, Source().wikiPagesBatchOne())
            pagesOne.forEach { page -> Words(page.text).forEach { countWord(it) } }
        }.subscribeOn(Schedulers.newThread())
        val observablePagesTwo = Observable.fromCallable {
            val pagesTwo = Pages(0, 5000, Source().wikiPagesBatchTwo())
            pagesTwo.forEach { page -> Words(page.text).forEach { countWord(it) } }
        }.subscribeOn(Schedulers.newThread())

        observablePagesOne.mergeWith(observablePagesTwo)
                .doOnComplete { logData(System.currentTimeMillis() - startTime) }
                .subscribe()
    }

    private fun countWord(word: String) {
        when(counts.containsKey(word)) {
            true -> counts[word] = counts[word]?.plus(1)
            false -> counts[word] = 1
        }
    }
}
```

RxJava 2

Run Two Threads

Verifying assumptions: parallel collections

```
Test started for: class java.util.Hashtable
500K entried added/retrieved in 1432 ms
500K entried added/retrieved in 1425 ms
500K entried added/retrieved in 1373 ms
500K entried added/retrieved in 1369 ms
500K entried added/retrieved in 1438 ms
For class java.util.Hashtable the average time 1407 ms

Test started for: class java.util.Collections$SynchronizedMap
500K entried added/retrieved in 1431 ms
500K entried added/retrieved in 1460 ms
500K entried added/retrieved in 1387 ms
500K entried added/retrieved in 1456 ms
500K entried added/retrieved in 1406 ms
For class java.util.Collections$SynchronizedMap the average time 1428 ms

Test started for: class java.util.concurrent.ConcurrentHashMap
500K entried added/retrieved in 413 ms
500K entried added/retrieved in 351 ms
500K entried added/retrieved in 427 ms
500K entried added/retrieved in 337 ms
500K entried added/retrieved in 339 ms
For class java.util.concurrent.ConcurrentHashMap the average time 373 ms  <== Much faster
```

Hypothesis 2: XML parsing?

I/O: Reading from disk?

```
class SequentialWordCount {
    private val counts: HashMap<String, Int?> = HashMap()

    fun run() {
        val startTime = System.currentTimeMillis()
        val observable = Observable.fromCallable {
            val pagesOne = Pages(0, 5000, Source().wikiPagesBatchOne())
            pagesOne.forEach { page -> Words(page.text).forEach { countWord(it) } }

            val pagesTwo = Pages(0, 5000, Source().wikiPagesBatchTwo())
            pagesTwo.forEach { page -> Words(page.text).forEach { countWord(it) } }
        }

        observable
                .doOnComplete { logData(System.currentTimeMillis() - startTime) }
                .subscribeOn(Schedulers.single())
                .subscribe()
    }

    private fun countWord(word: String) {
        when(counts.containsKey(word)) {
            true -> counts[word] = counts[word]?.plus(1)
            false -> counts[word] = 1
        }
    }
}
```

2.

RxJava 2

Run sequentially

Measuring:
Measure and Measure

```
com.fernandocejas.sample.threading.rxjava.SequentialWordCount: PageOne creation time: 15 ms
com.fernandocejas.sample.threading.rxjava.SequentialWordCount: PageTwo creation time: 13 ms
com.fernandocejas.sample.threading.rxjava.SequentialWordCount: Total Execution Pages Creation: 28 ms

com.fernandocejas.sample.threading.data.Pages: Time Parsing XML File: 4062 ms
com.fernandocejas.sample.threading.data.Pages: Time Processing XML Node Elements: 611 ms
com.fernandocejas.sample.threading.data.Pages: Total Time: 4673 ms

com.fernandocejas.sample.threading.data.Pages: Time Parsing XML File: 4360 ms
com.fernandocejas.sample.threading.data.Pages: Time Processing XML Node Elements: 631 ms
com.fernandocejas.sample.threading.data.Pages: Total Time: 4991 ms
```

What are the BOTTLENECKS and FIRST CONCLUSIONS?

Threads being idle waiting for I/O.

Locking the map when adding elements.

We can do better!

- ✘ Producer – Consumer pattern?
- ✘ Divide and Conquer?
- ✘ Reusing Threads?
- ✘ Other synchronized collections?

```
class BetterWordCount(source: Source) {

    fun run() {
        launch(CommonPool) {
            val time = measureTimeMillis {
                val one = async(CommonPool) { counter(0.rangeTo(749), filePagesOne) }
                val two = async(CommonPool) { counter(750.rangeTo(1500), filePagesOne) }
                val three = async(CommonPool) { counter(0.rangeTo(749), filePagesTwo) }
                val four = async(CommonPool) { counter(750.rangeTo(1500), filePagesTwo) }
                one.await(); two.await(); three.await(); four.await()
            }
            logData(time)
        }
    }

    private suspend fun counter(range: IntRange, file: File): HashMap<String, Int?> {
        val counts: HashMap<String, Int?> = HashMap()
        val pagesOne = Pages(range.start, range.endInclusive, file)
        pagesOne.forEach { page -> Words(page.text).forEach { countWord(counts, it) } }
        return counts
    }
}
```

Better solution

Kotlin Coroutines

14.621 MILLIS

Execution time

196.681 WORDS FOUND

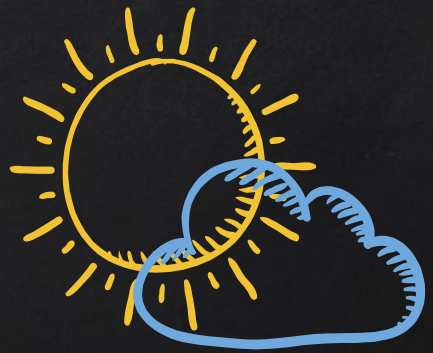
Two 30 MB XML files processed on an android device

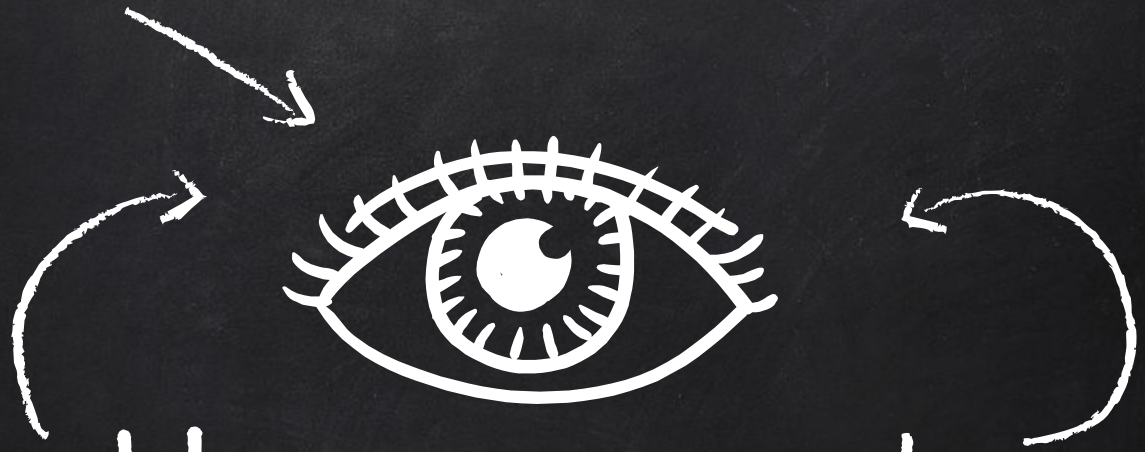
```
com.fernandocejas.sample.threading.coroutines.BetterWordsCount: Number of elements: 196781
com.fernandocejas.sample.threading.coroutines.BetterWordsCount: Execution Time: 14621 ms
```

Round 2 Results

	Single Thread	Two Threads	Better?
Threads and Locks	21.678 ms	18.427 ms	???
RxJava 2	20.920 ms	17.256 ms	???
Kotlin Coroutines	20.958 ms	18.980 ms	14.621 ms

WE CAN DO BETTER!





Homework!

Write sample code:

- ✘ Using Threads and Locks
- ✘ Using Kotlin Coroutines
- ✘ Using a pure FP Language

- https://github.com/android10/Multi-Threading-Samples

Facing multithreading problems:

- ✘ Debugging.
- ✘ Mutability.
- ✘ Performance.
- ✘ Testing.
- ✘ Sharing state.
- ✘ ???

...that is why is important to know the **fundamentals** and **building blocks**.

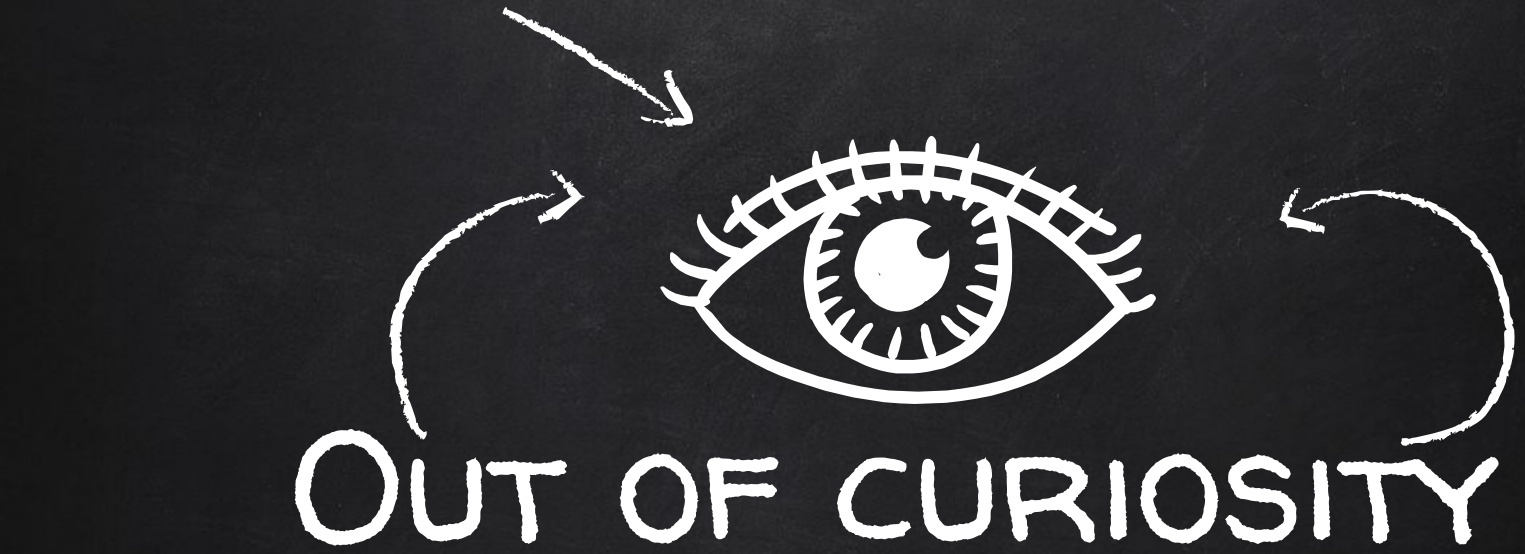
1. Use the right tool for the right job.
2. Always measure.
3. No silver bullets.

FIRST, make code work, THEN make it right, THEN make it fast...if it isn't fast enough.

TODO (to explore):

- ✘ Memory Consumption.
- ✘ Android Threading Components.
- ✘ External Libraries.
- ✘ iOS Threading Model.
- ✘ Server Side Multithreading.

- https://github.com/android10/Multi-Threading-Samples



Other threading approaches:

- ✘ Actor Model
- ✘ FP Languages:
  - Clojure
  - Scala
  - Haskel

Wrapping up...

Responsiveness

Concurrency can ensure improved responsiveness, favoring better user experiences and faster applications.

Resources

By running tasks in parallel, the use of resources is better and more performant.

Simplicity

Multithreading is not easy...but in many cases can simplify the design of your system by using a divide and conquer approach.

THANKS!

Any questions?

You can find me here:

@fernando_cejas
http://fernandocejas.com
github.com/android10

Credits

Special thanks to all the people who made and released these awesome resources for free:

- ✘ Presentation template by SlidesCarnival
- ✘ Photographs by Unsplash