



Kotlin Coroutines para Android Apps

Por Danilo Domínguez

Acerca de mí

- Danilo Domínguez Pérez
- Egresado de la UTP - Ing. en Sistemas y Computación
- Miembro de  Flosspa
- Ingeniero Móvil en Automattic
- Miembro de IEEE Computer Society Chapter Panamá
- Maestría en Ciencias Computacionales del Rochester Institute of Technology
- Doctorado en Ciencias Computacionales de Iowa State University
 - Investigación en Análisis y Testing de Aplicaciones Móviles



<https://www.linkedin.com/in/danilo-dominguez-perez>



@danilo04



@danilo04

Agenda

1.

¿Qué es Kotlin?

Historia de Kotlin

2.

Variables

Definición de variables

3.

Funciones

Funciones y extensiones

4.

Null safety

Capacidades para Null safety

5.

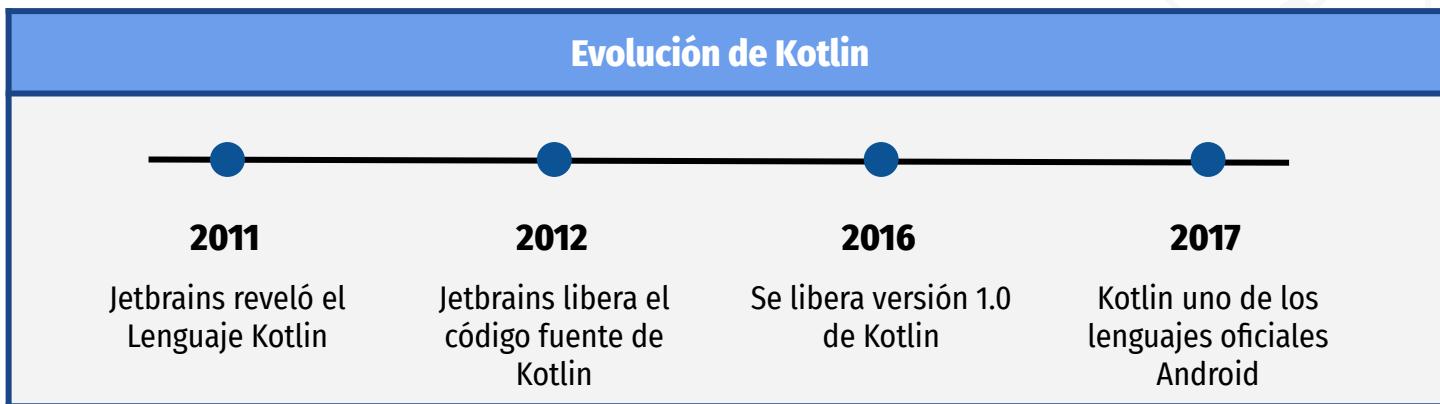
Coroutines

Cómo utilizar Coroutines en Android

¿Qué es Kotlin?

¿Qué es Kotlin?

- Lenguaje de Programación
- Desarrollado por Jetbrains (IntelliJ IDEA, Webstorm, PHPStorm, PyCharm, ...)
- 100% interoperabilidad con Java
- Orientado a Objetos con sintaxis concisa
- Compila a JVM, javascript y nativo (LLVM)



Comparación con Java

Similitudes

- Organización en paquetes y archivos
- Clases y herencia (reglas similares)
- Sintaxis es similar
- Compatible con la JVM

Diferencias

- Null safety
- Funciones de extensión
- Coroutines
- Smart casts
- Delegación de propiedades
- Overloading de operadores
- ...

Conciso

```
// Java
public class Address {
    private String street;
    private int streetNumber;

    public Address(String street, int streetNumber) {
        this.street = street;
        this.streetNumber = streetNumber;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        Address address = (Address) o;

        if (streetNumber != address.streetNumber) return false;
        return street.equals(address.street)
    }

    @Override
    public int hashCode() {
        int result = street.hashCode();
        result = 31 * result + streetNumber;
        return result;
    }

    @Override
    public String toString() {
        return "Address{" +
            "street='" + street + '\'' +
            ", streetNumber=" + streetNumber +
            '}';
    }
}
```

```
// Kotlin
data class Address(var street: String,
                    var streetNumber: Int)
```

Consideración: Java 14 tiene **Records (JEP 359)**

Conciso

Verboso	Conciso
<pre>// With else var max: Int if (a > b) { max = a } else { max = b }</pre>	<pre>// As expression val max = if (a > b) a else b</pre>
<pre>if (x >= 1 && x <10) { print("x is in the range") } else if (x in validNumbers) { print("x is valid") } else if (!in 10..20) { print("x is outside the range") } else { print("none of the above") }</pre>	<pre>when (x) { in 1..10 -> print("x is in the range") in validNumbers -> print("x is valid") !in 10..20 -> print("x is outside the range") else -> print("none of the above") }</pre>

Variables

Variables

Mutables

```
var lastName: String = "Dominguez"  
lastName = "Perez"
```

Solo lectura

```
val lastName: String = "Dominguez"  
lastName = "Perez" // error de compilación
```

Instanciar Objetos

```
// no se necesita new keyword  
val lastName: StringBuiler = StringBuiler("Hola")  
val lastName = StringBuiler("Hola")
```

Funciones

Funciones

```
fun compareDates(first: LocalDate, second: LocalDate): Boolean {
    return first.isAfter(second);
}

fun compareDates2(first: LocalDate, second: LocalDate = LocalDate.now()): Boolean {
    return first.isAfter(second);
}

val halloween = LocalDate.of(2016, 10, 31)
val christmas = LocalDate.of(2016, 12, 25)

fun main(args: Array<String>) {
    println("after: " + compareDates(halloween, christmas));
    println("before: " + compareDates(christmas, halloween));
    println("now?: " + compareDates2(christmas));
    println("now?: " + compareDates(second = christmas, first = LocalDate.now()));
}
```

Funciones de Extensión

- Kotlin permite extender funcionalidad de una clase sin herencia o modificar la clase directamente
- Puedes escribir funciones para una clase de una librería de terceros

```
fun Int.maximum(other: Int) =  
    if (this > other) this else other  
  
max = 3.maximum(6)  
  
infix fun Int.maximum(other: Int) =  
    if (this > other) this else other  
  
max = 3 maximum 6
```

```
fun MutableList<Int>.swap(index1: Int, index2: Int) {  
    val tmp = this[index1]  
    this[index1] = this[index2]  
    this[index2] = tmp  
}  
  
val list = mutableListOf(1, 2, 3)  
list.swap(0, 2)
```

Android KTX

- Provee funciones de extensión y API más en la sintaxis de Kotlin para el framework de Android
- Incluye:
 - Funciones de extensión
 - Parámetros con nombre
 - Valores por defecto en parámetros
 - Coroutines

```
// Commit a new value asynchronously
sharedPreferences.edit { putBoolean("key", value) }

// Commit a new value synchronously
sharedPreferences.edit(commit = true) {
    putBoolean("key", value)
}

db.transaction {
    // insert data
}
```

Null Safety

Null safety

- El sistema de tipos de Kotlin trata de eliminar NPE
- Variables por defecto no pueden tener valores nulos
- Se pueden detectar errores en tiempo de compilación



```
var a: String = "abc"
a = null // error de compilación
var b: String? = "abc"
b = null // ok
print(b)
```

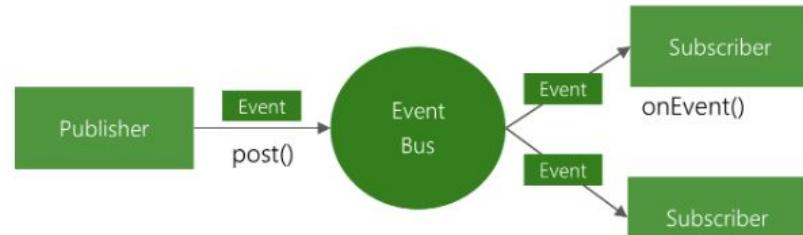
Null safety

Verboso	Conciso
<pre>val l: Int = if (b != null && b.a != null) b.a.length else -1</pre>	<pre>val l = b?.a?.length ?: -1 // Operador Elvis</pre>
<pre>var name: String? = null if (bob != null && bob.department != null) { name = bob.department.name }</pre>	<pre>val name = bob?.department?.name</pre>

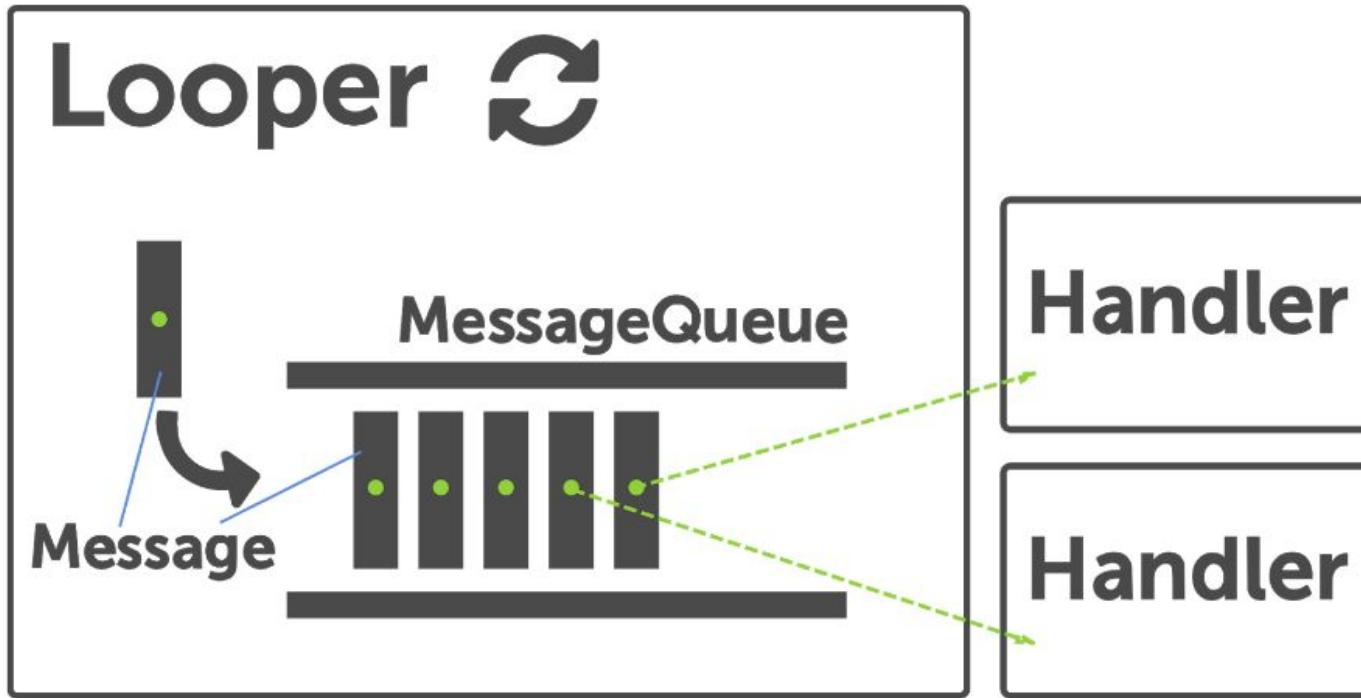
Coroutines

Programación Basada en Eventos

- La ejecución de una aplicación está regida por eventos
- Eventos como
 - Acciones de UI (clicks, tabs, etc)
 - Sensores
 - Mensajes
- Utilizado por
 - Aplicaciones gráficas de escritorio
 - Aplicaciones Web
 - Aplicaciones Móviles

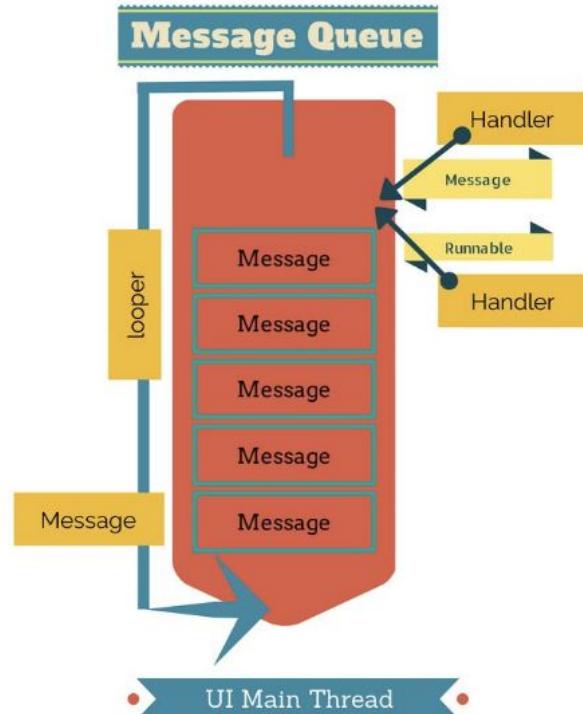


Como Android implementa la programación basada en eventos



UI Thread

- Todas las apps tienen un thread principal que maneja la mayoría de eventos (principalmente de UI)
- UI thread tiene un message queue asociado
- Cuando un evento (e.j. de UI) pasa, el OS publica un mensaje en el message queue del UI thread
- Cuando el Looper dequeue el mensaje, ejecuta el callback que escucha el evento (event listener)



Long Running Tasks

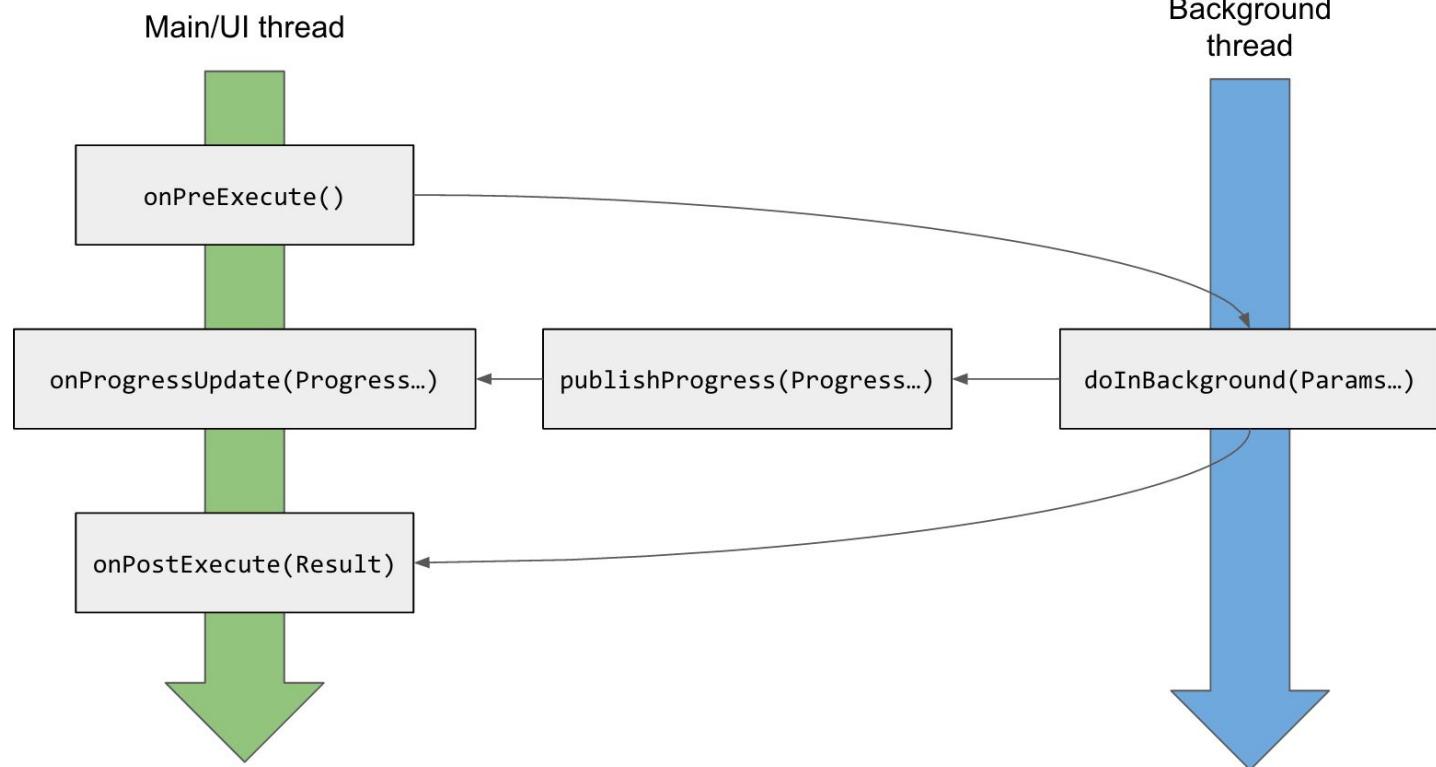
Long Running Tasks

- Deben ser ejecutadas en un thread diferente a **UI Thread**
- Cualquier cambio en UI debe ejecutarse en el **UI Thread**
- Android Framework ofrece diferentes mecanismos:
 - AsyncTasks
 - Services
 - Threads
 - **Coroutines**



Callback latency debe ser **16ms** para
asegurar **60fps**

AsyncTasks



Coroutines

- Utilizadas para programación asíncrona
- Primer lenguaje que exploró su uso fue *Simula* en 1967
- En Android son útiles para ejecutar long running task
 - Rest API requests
 - Base de Datos
 - Análisis de datos complejos
- Principal language construct **suspend**
- Dependencias
 - implementation
'org.jetbrains.kotlinx:kotlinx-coroutines-core:1.0.0'
 - implementation
'org.jetbrains.kotlinx:kotlinx-coroutines-android:1.0.0'



suspend functions

- Coroutines son **computations** que pueden ser **suspendidas** sin bloquear el thread donde son llamados hasta que termine un cómputo o retorne un valor
- **suspend** functions pueden ser ejecutadas sólo dentro de un **coroutine scope** o dentro de otra **suspend** function
- Para eso utilizamos **coroutine builders**

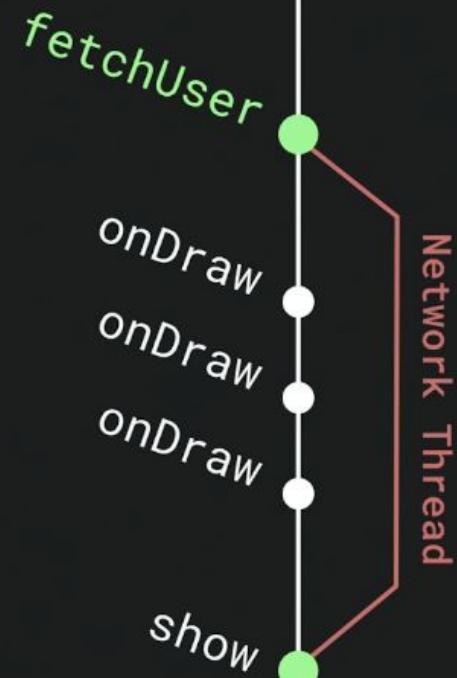


One cannot simply invoke
a suspending function

suspend functions

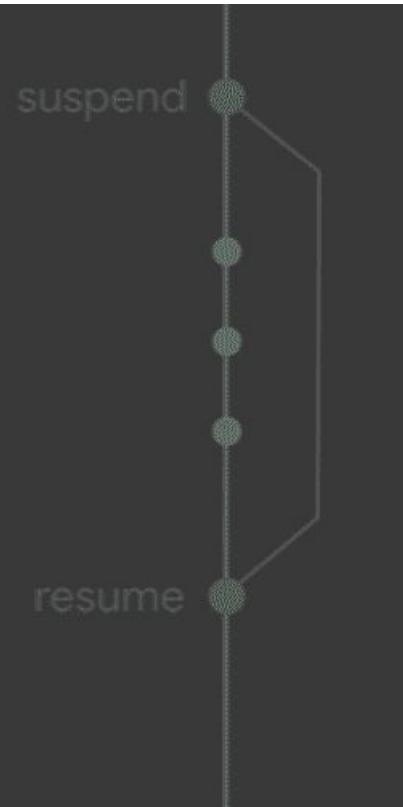
coroutines.kt

```
suspend fun loadUser() {  
    ↳ val user = api.fetchUser()  
    show(user)  
}
```



Ejecución de suspend functions

```
suspend fun fetchDocs() {  
    val docs = get("")  
    show(docs)  
}
```



Main Thread
[stack]

Coroutine Builders

- Los coroutines son ejecutados por un **coroutine builder** en el contexto de un **coroutine scope**
- Los **coroutine scopes** defines el lifetime de los coroutines que corren en su contexto
 - “[**GlobalScope**](#), meaning that the lifetime of the new coroutine is limited only by the lifetime of the whole application”
 - Podemos definir scopes: e.j. Activity scoped, Fragment scoped
- **launch** y **async** son dos coroutine builders que normalmente utilizamos

```
GlobalScope.launch {  
    delay(1000L)  
    println("World!")  
}
```

Composing suspend functions

```
suspend fun doSomethingUsefulOne(): Int {  
    delay(1000L)  
    return 13  
}
```

```
suspend fun doSomethingUsefulTwo(): Int {  
    delay(1000L)  
    return 29  
}
```

```
val time = measureTimeMillis {  
    val one = doSomethingUsefulOne()  
    val two = doSomethingUsefulTwo()  
    println("The answer is ${one + two}")  
}  
println("Completed in $time ms")
```

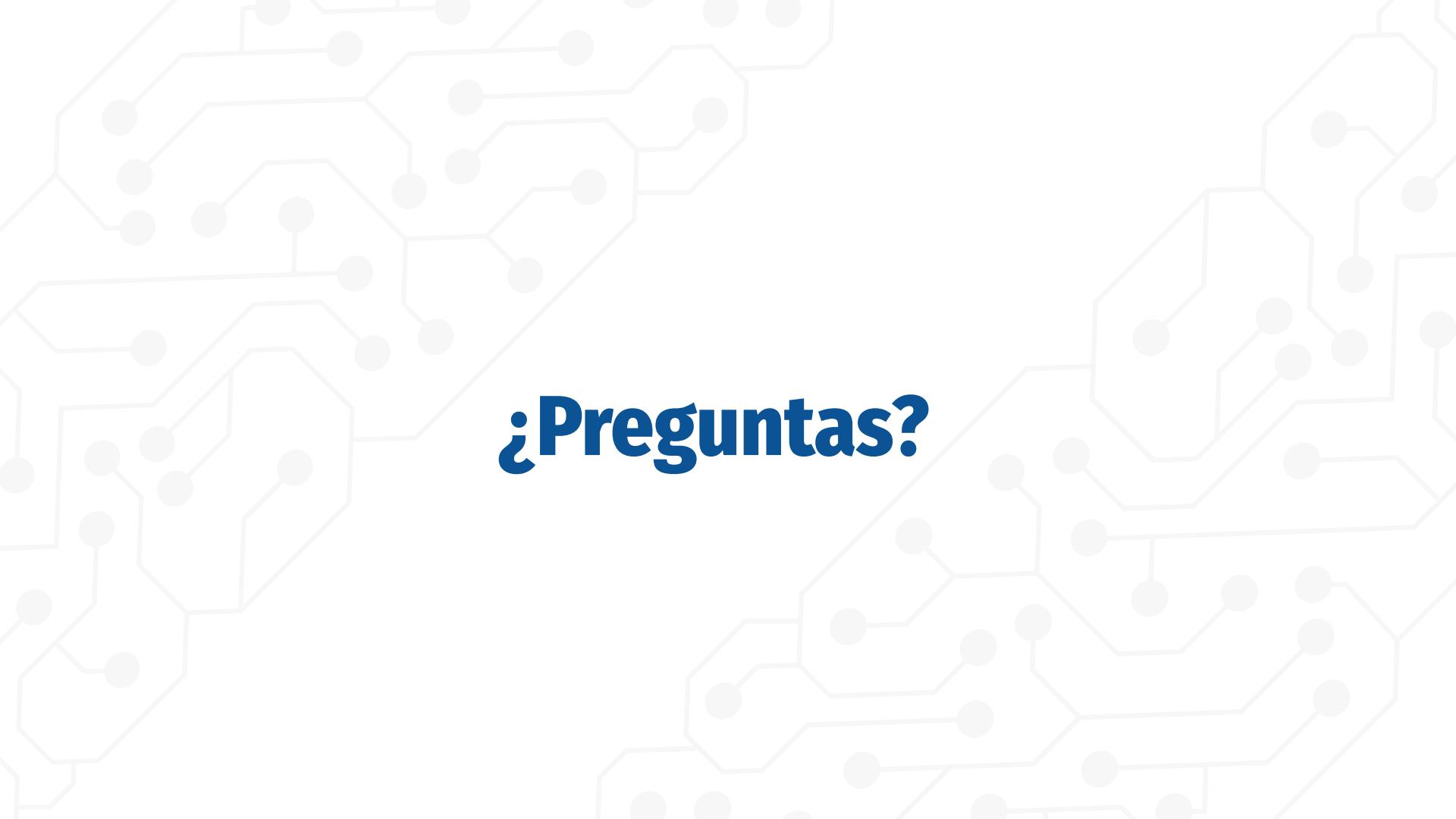
Completa en 2017 ms

```
val time = measureTimeMillis {  
    val one = async { doSomethingUsefulOne() }  
    val two = async { doSomethingUsefulTwo() }  
    println("Answer ${one.await() + two.await()}")  
}  
println("Completed in $time ms")
```

Completa en 1017 ms

Exceptions

- Exception propagation es diferente en **launch** y **actor** que en **async** y **produce**
- **Referencia:**
<https://kotlinlang.org/docs/reference/coroutines/exception-handling.html>



¿Preguntas?