
COROUTINES
vs
RX

QUEM SOU

Filipe Nunes

Mobile Specialist.

Organizador do GDG Porto Alegre e Leader Jam da Google.

Evangelista do open source, envolvido em projetos como FISL e HacktoberFest, Google IO Extended, Congressos de TI, TDCs e DevFests.

Participante de projetos e empresas como IBM, SAP, Warren, Grupo RBS dentre outras.

SETUP

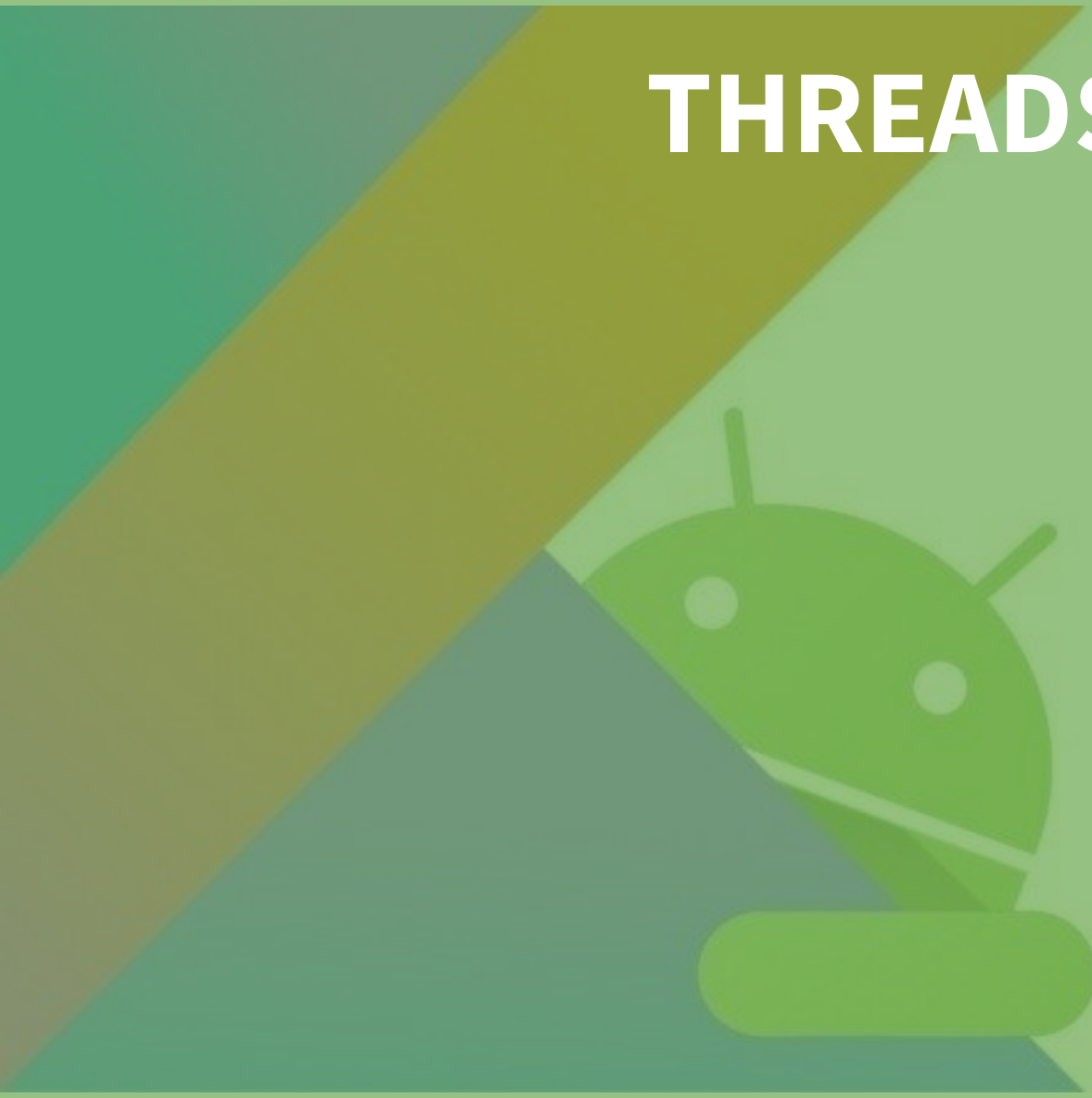
```
implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:1.3.31"

implementation "org.jetbrains.kotlinx:kotlinx-coroutines-android:1.1.1"

implementation "org.jetbrains.kotlinx:kotlinx-coroutines-core:1.2.1"
```

```
implementation 'com.squareup.retrofit2:retrofit:2.6.1'

implementation "com.squareup.retrofit2:converter-moshi:2.6.1"

implementation "com.jakewharton.retrofit:retrofit2-kotlin-coroutines-adapter:0.9.2"
```

ASYNCHRONOUS PROGRAMMING

For decades, as developers we are confronted with a problem to solve - how to prevent our applications from blocking.

THREADS



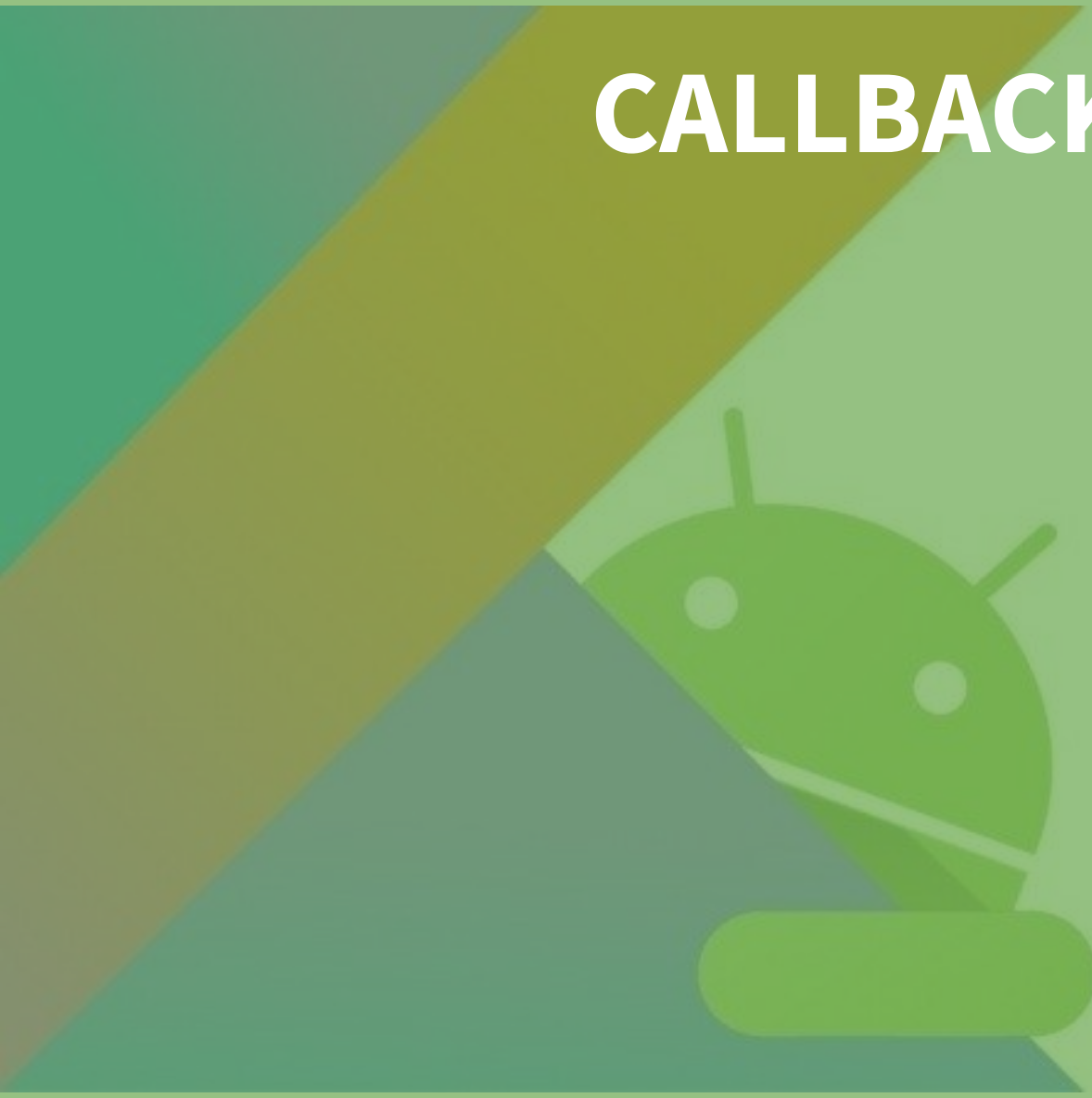
android

Kotlin

```
fun run() {
    println("${Thread.currentThread()} has run.")
}

fun th() {
    val thread = Thread {
        println("${Thread.currentThread()} has run.")
    }
    thread.start()
}

fun thread(
    start: Boolean = true,
    isDaemon: Boolean = false,
    contextClassLoader: ClassLoader? = null,
    name: String? = null,
    priority: Int = -1,
    block: () -> Unit
): Thread
```

It’s tempting to think that spawning more threads can help us execute more tasks concurrently. Unfortunately, that’s not always true.

CALLBACKS



android

Kotlin

```
fun saveUser(user: CreateUserPhysical) {
    preparePostAsync { token ->
        submitPostAsync(token, user) { post ->
            processPost(post)
        }
    }
}

fun processPost(post: Any): Any {
    return post
}

fun submitPostAsync(token: Any, item: Any, post: Any) {

}

fun preparePostAsync(callback: (Token) -> Unit) {
    // make request and return immediately
    // arrange callback to be invoked later
}
```

Rx
PROMISSES
FUTURES

android

Kotlin

```
interface StackOverflowService {
  @GET("/users")
  fun getTopUsers(): Single<List<User>>

  @GET("/users/{userId}/badges")
  fun getBadges(
      @Path("userId") userId: Int
  ): Single<List<Badge>>

  @GET("/users/{userId}/top-tags")
  fun getTags(
      @Path("userId") userId: Int
  ): Single<List<Tag>>
}
```

```
class MyViewModel(
  private val service: StackOverflowService
) : ViewModel() {

  private val disposable = CompositeDisposable()

  fun load() {
      disposable +=
          service.getTopUsers()
              .subscribeOn(io())
              .observeOn(mainThread())
              .subscribe(
                  { users -> updateUi(users) },
                  { e -> updateUi(e) }
              )
  }

  private fun updateUi(s: Any) {
      //...
  }

  override fun onCleared() {
      disposable.clear()
  }
}
```

SUSPEND FUNCTIONS

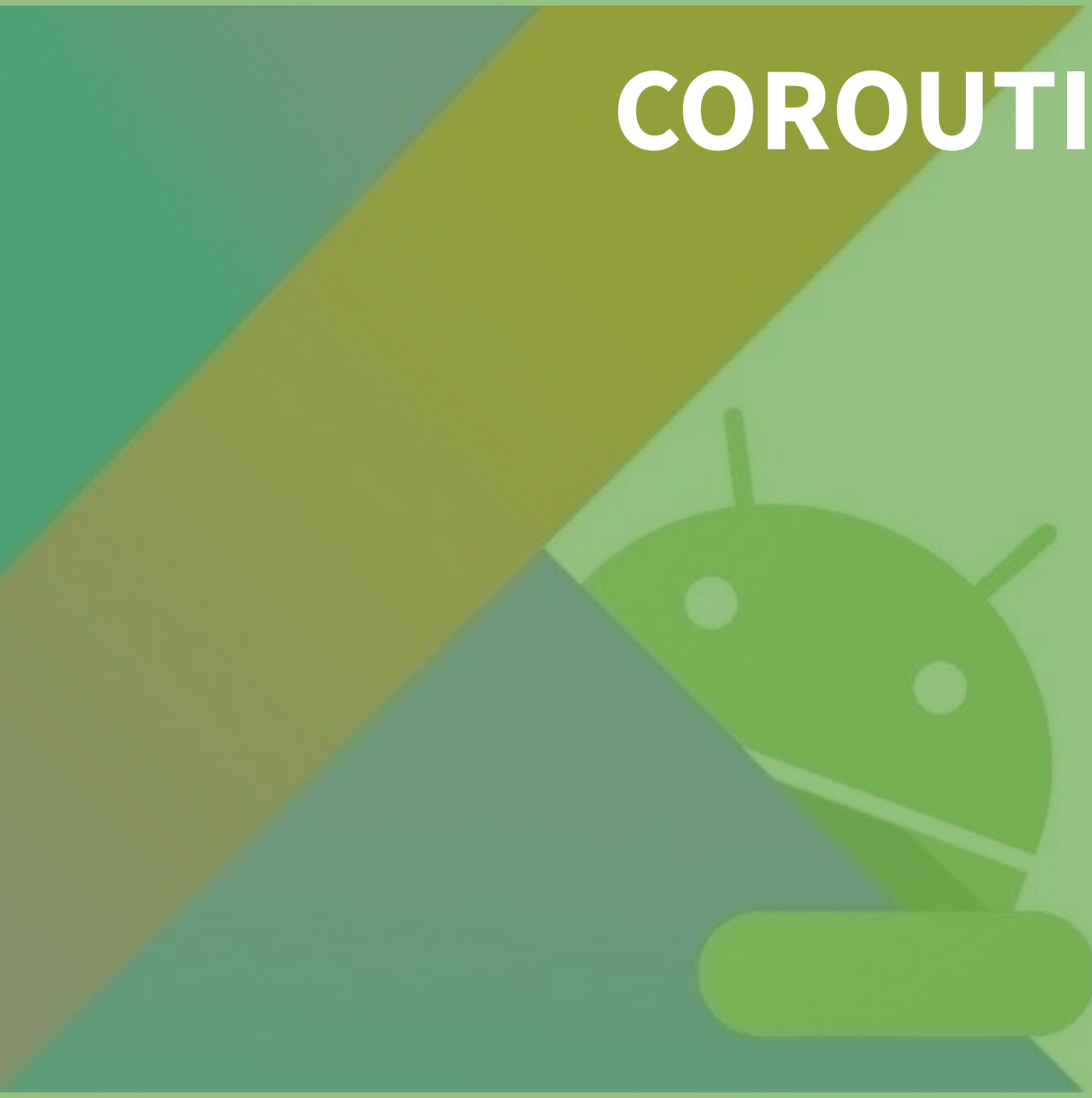
android

Kotlin

```
interface StackOverflowService {
    @GET("/users")
    fun getTopUsers(): Deferred<List<User>>

    @GET("/users/{userId}/badges")
    fun getBadges(
        @Path("userId") userId: Int
    ): Deferred<List<Badge>>

    @GET("/users/{userId}/top-tags")
    fun getTags(
        @Path("userId") userId: Int
    ): Deferred<List<Tag>>
}
```

```
interface StackOverflowService {
    @GET("/users")
    suspend fun getTopUsers(): List<User>

    @GET("/users/{userId}/badges")
    suspend fun getBadges(
        @Path("userId") userId: Int
    ): List<Badge>

    @GET("/users/{userId}/top-tags")
    suspend fun getTags(
        @Path("userId") userId: Int
    ): List<Tag>
}
```

COROUTINES



android

Kotlin

```
class MyViewModel(
    private val service: StackOverflowService
) : ViewModel() {

    fun load() {
        launch {
            try {
                val users = service.getTopUsers()
                updateUi(users)
            } catch (e: Exception) {
                updateUi(e)
            }
        }
    }

    private fun updateUi(s: Any) {
        //...
    }
}
```

FIGHT

android

Kotlin

- Very light-weight threads
- Transformation technique during a compilation
- Running in a shared thread pool

Coroutine vs Thread

Gerenciado pela aplicação	Gerenciado pelo sistema operacional
Não necessita context switching	Necessita context switching
Concorrente	Paralelo
Não mapeiam threads nativas	Mapeiam threads nativas

1.000 interações

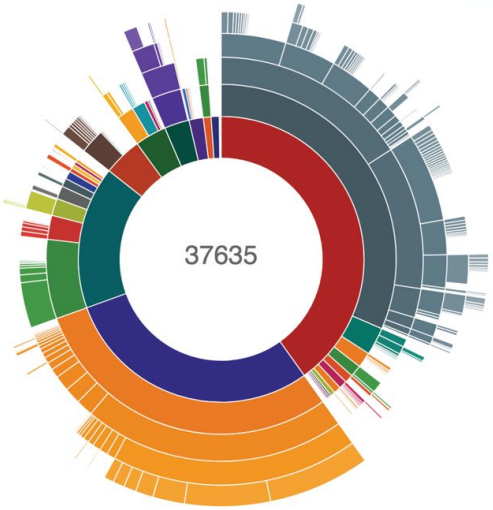
Test	base mem	max mem	delta	max CPU	time
Coroutines 1	61.2 MB	61.7 MB	0.5 MB	7 %	0.06 s
Coroutines 2	61.5 MB	61.9 MB	0.4 MB	7 %	0.01 s
RxJava	59.6 MB	61.4 MB	1.8 MB	12 %	0.04 s

10.000 interações

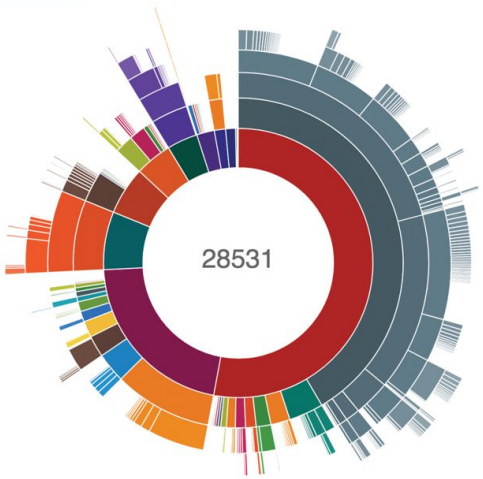
Test	base mem	max mem	delta	max CPU	time
Coroutines 1	60.9 MB	67.8 MB	6.9 MB	17 %	1.5 s
Coroutines 2	61.9 MB	64.6 MB	2.7 MB	17 %	0.1 s
RxJava	59.6 MB	76.0 MB	16.4 MB	24 %	9.5 s

Method count

debug



11051 io.reactivex

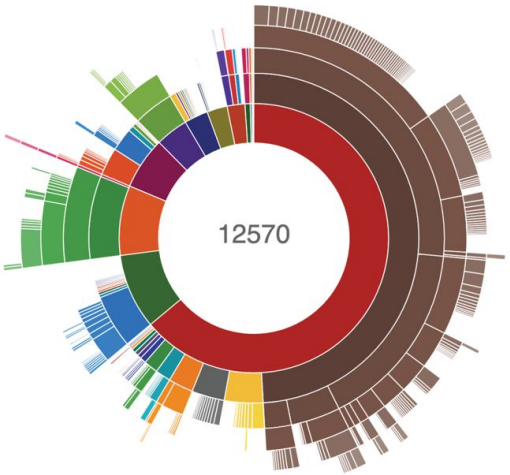


144 kotlin.coroutines

1984 kotlinx.coroutines

release

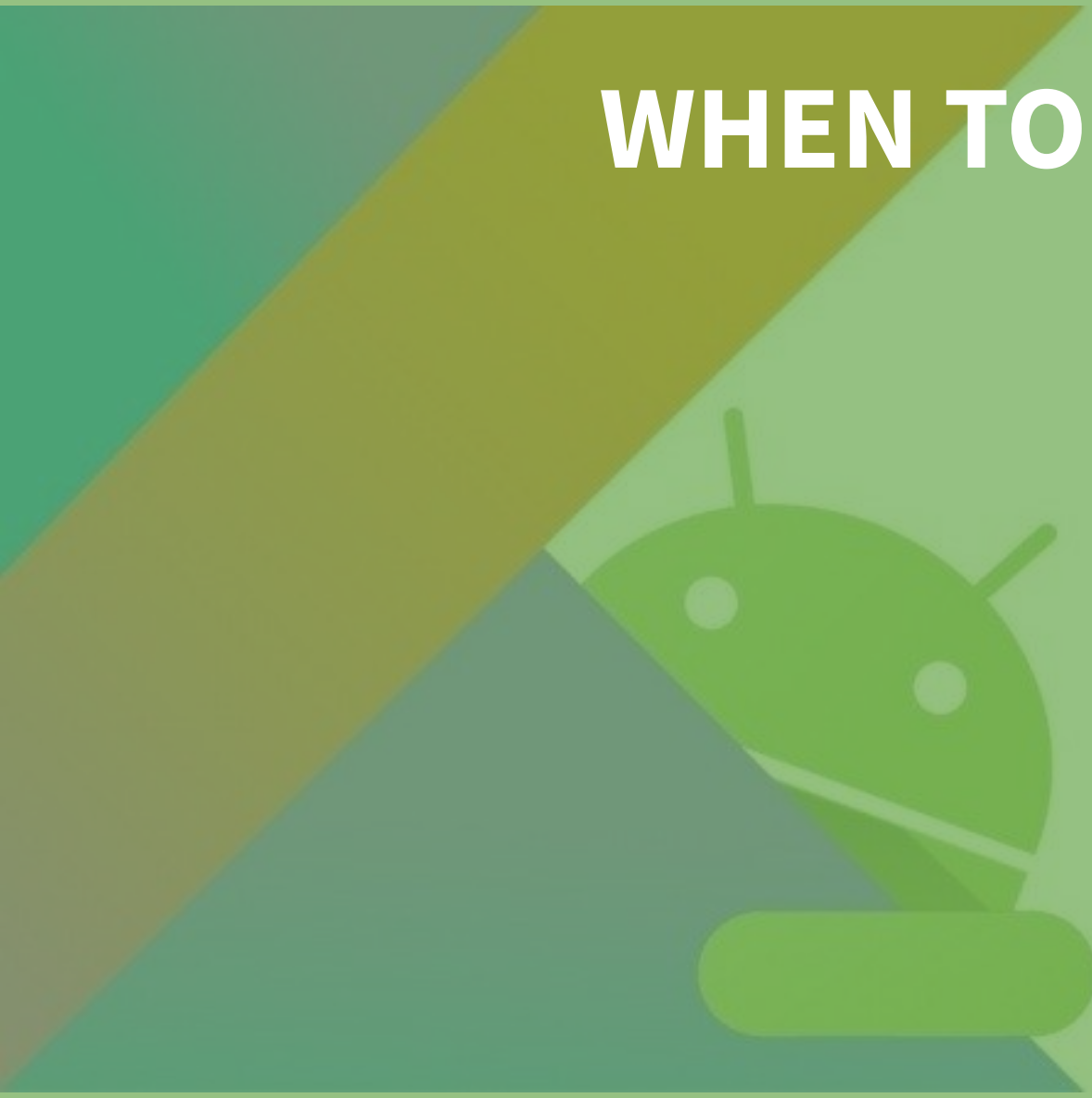
427 io.reactivex



77 kotlin.coroutines

493 kotlinx.coroutines

WHEN TO USE



android

Kotlin

if (you are already using RxJava and it works for you) { RXJAVA }

if (the architecture is based on reactive stream) { RXJAVA }

if (the project is multiplatform with Kotlin Native) { coroutines}

if (the codebase is Java/Kotlin) { RXJAVA }

else { coroutines }

DÚVIDAS?