Efficient Async Coding in Kotlin Coroutines

Bangalore JAVA User Group (BoJUG) - Year's First Meetup - Full Day - Informatica - Jan, 19, 2019

19/01/19

Rivu Chakraborty

Efficient Async Coding in Kotlin Coroutines

Rivu Chakraborty

BYJU'S

@rivuchakraborty

About Me

Rivu Chakraborty

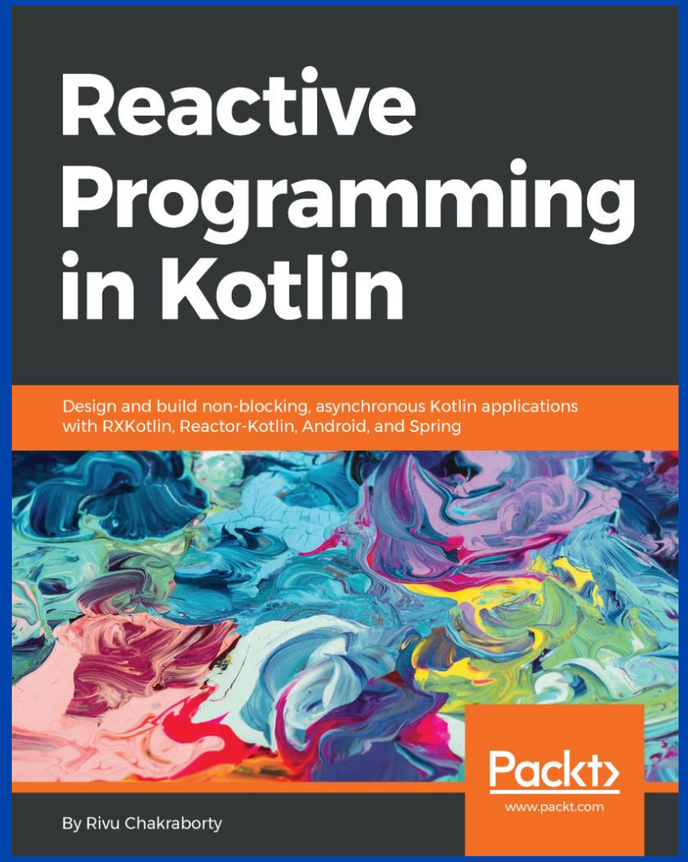
BYJU'S

@rivuchakraborty

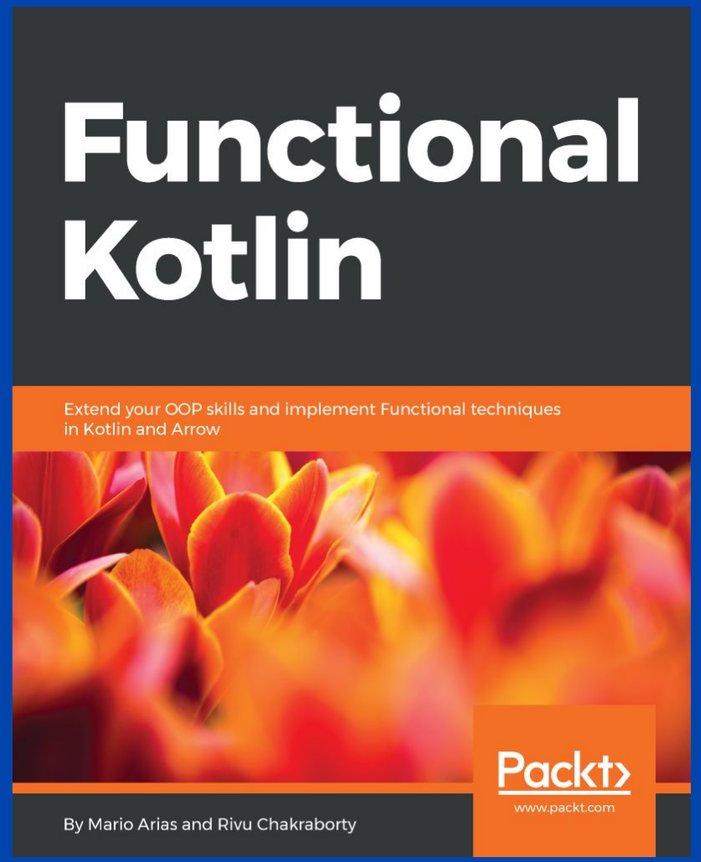
- Sr Software Engineer (Android) - BYJU'S
- Instructor - Caster.io
- Google Certified Associate Android Developer
- DroidJam Speaker
- Author - Reactive Programming in Kotlin
- Author - Functional Kotlin
- Author - Coroutines for Android Developers (WIP)

@rivuchakraborty

Books



https://www.packtpub.com/application-development/reactive-programming-kotlin



https://www.packtpub.com/application-development/functional-kotlin

Work in Progress

https://leanpub.com/coroutines-for-android-developers

Efficient **Async** Coding in Kotlin Coroutines

Async => Concurrency

Concurrency

- Ability to execute multiple code blocks at the same time

Concurrency

- Ability to execute multiple code blocks at the same time
- **Not only for Android Developers**

Concurrency

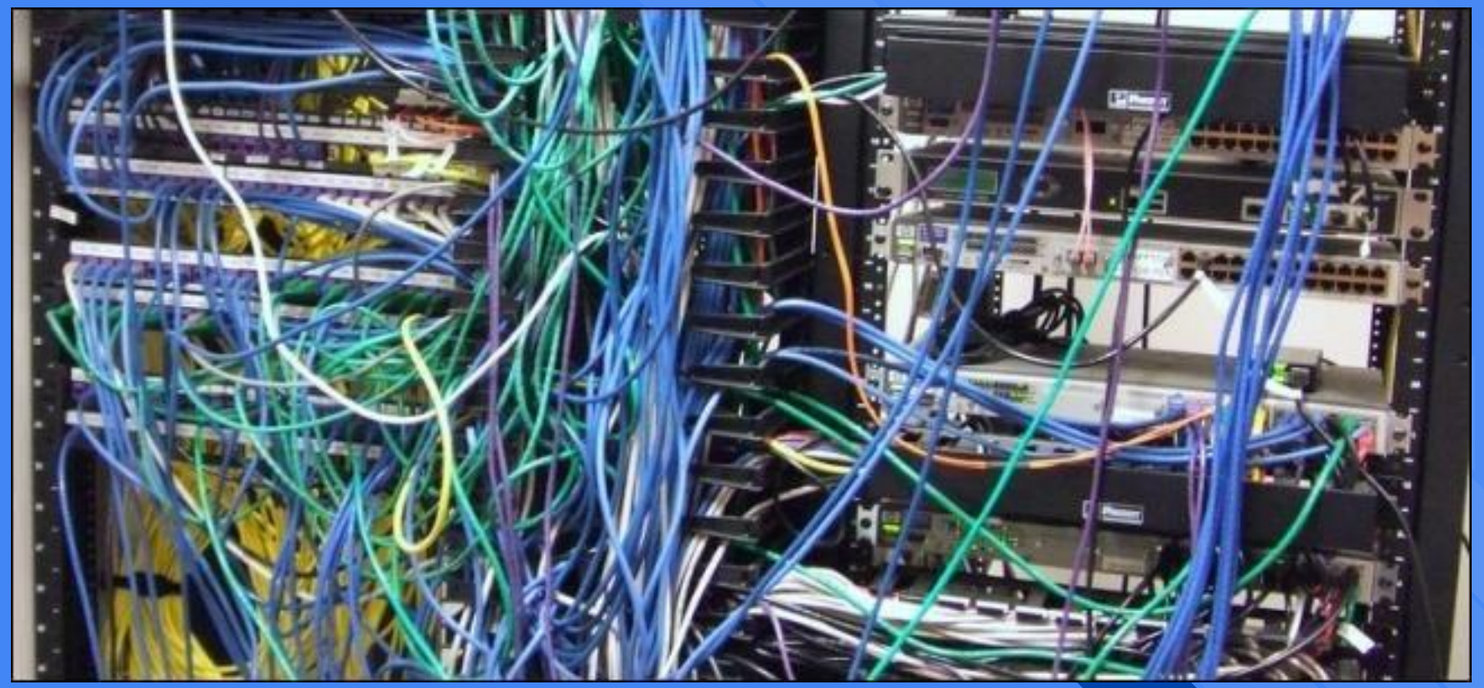
- Ability to execute multiple code blocks at the same time
- Not only for Android Developers
- **JavaScript - Promise**
- **JVM, Android - Threads**

Concurrency in Android and/or JVM

- AsyncTask (Android only)
- CompletableFuture (Android - API 24+, JVM 1.8)
- Runnable / Thread (Custom Threadpool)
- Rx
- Counting...

Concurrency in Android

- Runnable / Thread



Concurrency in Android

- Rx

Concurrency in Android

- Rx

If you're using Rx only for concurrency, sorry pal, you're doing it wrong.

- Coroutines

Let’s you write non-blocking asynchronous code in your choice of Style - Sequentially, in Functional Style or whatever you prefer.

• Coroutines

Let’s you write non-blocking asynchronous code in your choice of Style - Sequentially, in Functional Style or whatever you prefer.

A Language level feature, managed by the Kotlin team itself.

Coroutines

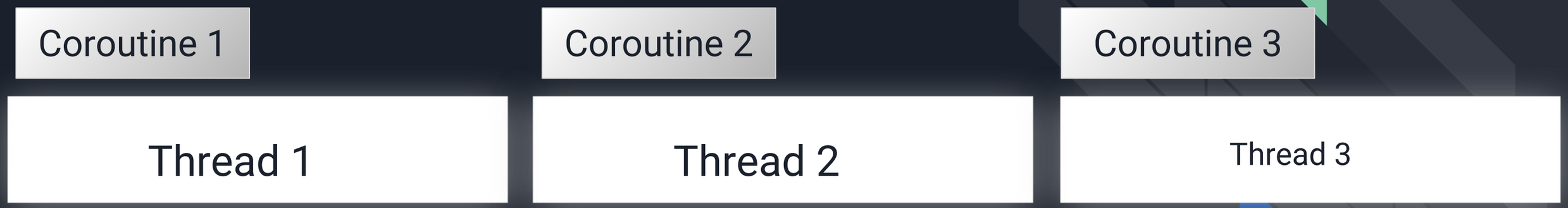
- Let’s you write non-blocking asynchronous code in your choice of Style - Sequentially, in Functional Style or whatever you prefer.
- A Language level feature, managed by the Kotlin team itself.

You can add Coroutines dependency to your Kotlin project by following the instructions in GitHub ReadMe - https://github.com/kotlin/kotlinx.coroutines/blob/master/README.md#using-in-your-projects

Coroutines

Light-Weight Threads

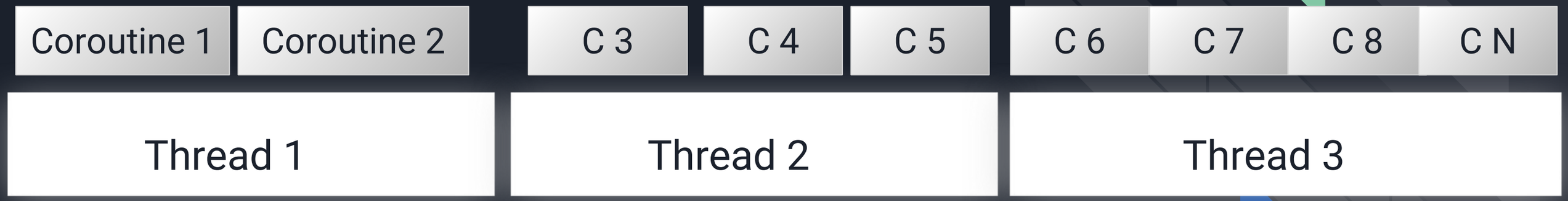
With Coroutines, Threads are still used (for JVM).



Coroutines

Light-Weight Threads

But a single Thread can run multiple coroutines.



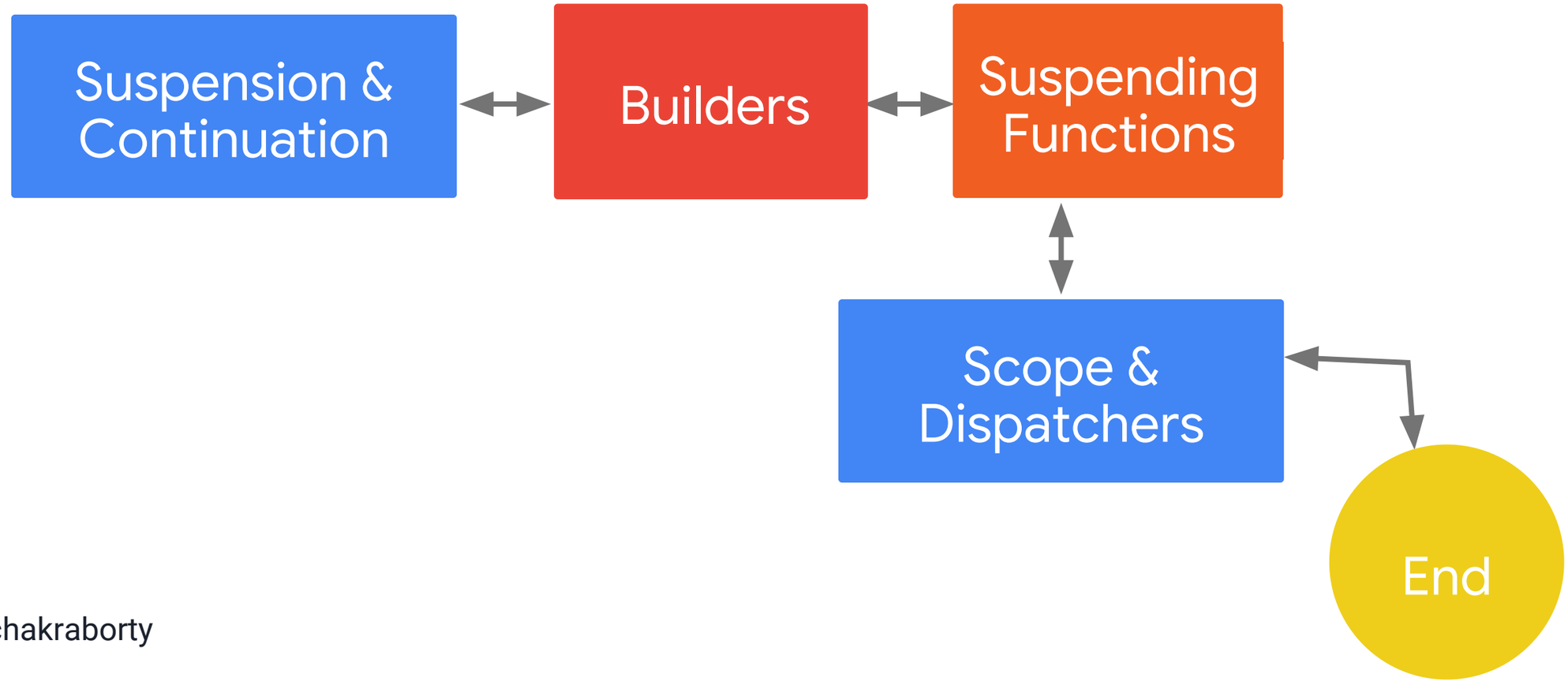
Coroutines

Easy to Use

```
val serverData = async {
    getDatafromServer()
}
val data = serverData.await()
```

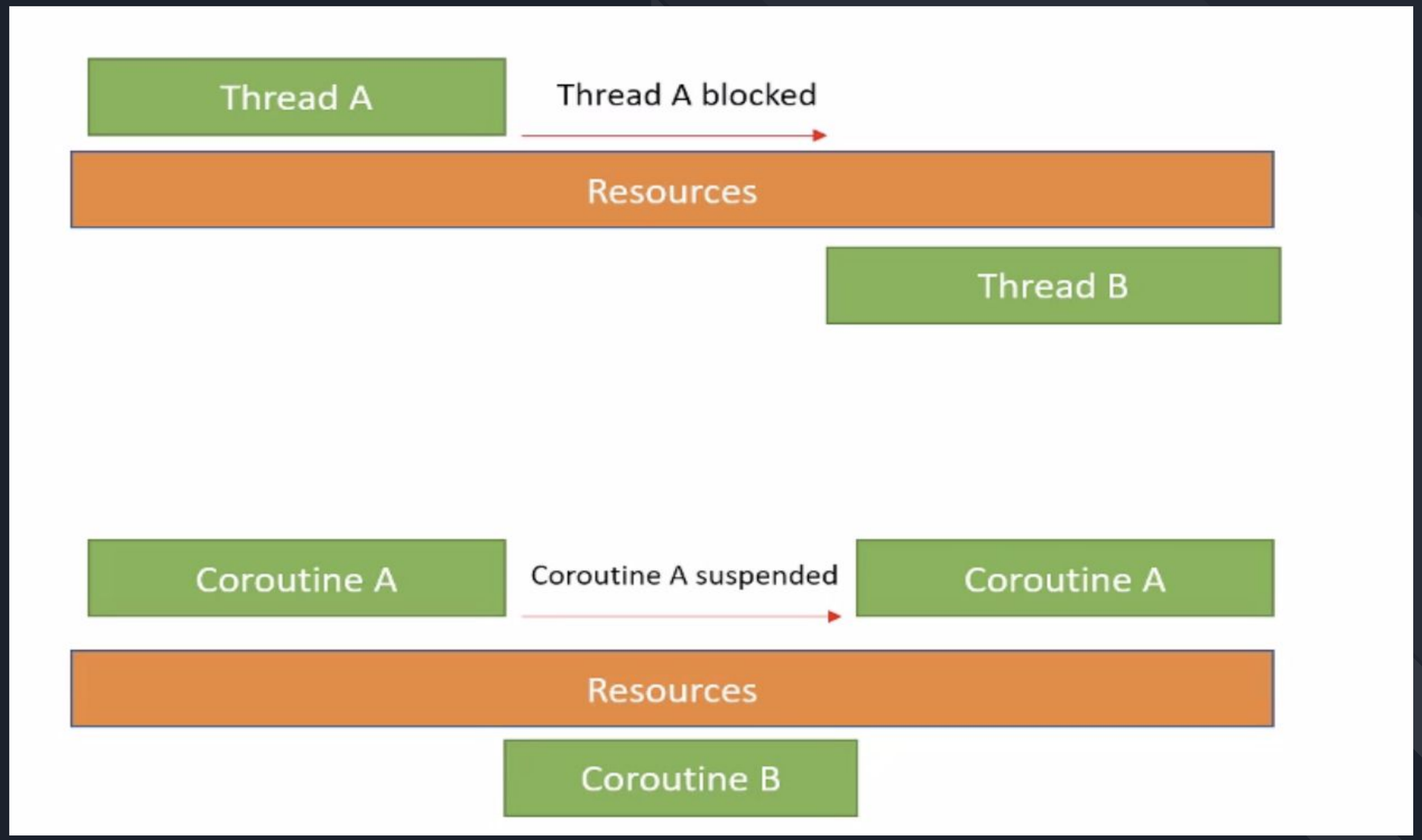
Coroutines

Components



Coroutines

Suspending vs Blocking



@rivuchakraborty

Coroutines

Suspending vs Blocking

```
public suspend fun delay(time: Long, unit: TimeUnit = TimeUnit.MILLISECONDS) {
    require( value: time >= 0) { "Delay time $time cannot be negative" }
    if (time <= 0) return // don't delay
    return suspendCancellableCoroutine sc@ { cont: CancellableContinuation<Unit> ->
        cont.context.delay.scheduleResumeAfterDelay(time, unit, cont)
    }
}
```

```
static void sleep(long ms, int ns) throws InterruptedException
{
    if (ms == 0 && ns == 0){
        if (Thread.interrupted())
            throw new InterruptedException();
        return;
    }

    long now = System.currentTimeMillis();
    long end = now + ms;
    if (end < now)
    end = Long.MAX_VALUE;
    VMThread vt = Thread.currentThread().vmThread;
    synchronized (vt)
    {
        while (true)
        {
            vt.wait(ms, ns);
            now = System.currentTimeMillis();
            if (now >= end)
                break;
            ms = end - now;
            ns = 0;
        }
    }
}
```

Coroutines

Suspending vs **Blocking**

```java
static void sleep(long ms, int ns) throws InterruptedException
{
    if (ms == 0 && ns == 0){
        if (Thread.interrupted())
            throw new InterruptedException();
        return;
    }

    long now = System.currentTimeMillis();
    long end = now + ms;
    if (end < now)
    end = Long.MAX_VALUE;
    VMThread vt = Thread.currentThread().vmThread;
    synchronized (vt)
    {
        while (true)
        {
            vt.wait(ms, ns);
            now = System.currentTimeMillis();
            if (now >= end)
                break;
            ms = end - now;
            ns = 0;
        }
    }
}
```

Coroutines

Suspending vs
Blocking

```
public suspend fun delay(time: Long, unit: TimeUnit = TimeUnit.MILLISECONDS) {
    require( value: time >= 0) { "Delay time $time cannot be negative" }
    if (time <= 0) return // don't delay
    return suspendCancellableCoroutine sc@ { cont: CancellableContinuation<Unit> ->
        cont.context.delay.scheduleResumeAfterDelay(time, unit, cont)
    }
}
```

Launch a Coroutine

```
GlobalScope.launch {
    someLongRunningTask()
    anotherLongRunningTask()
}
```

Get the Job done or cancel it, your choice. :)

```
val job:Job = GlobalScope.launch {
    someLongRunningTask()
    anotherLongRunningTask()
}
....
....
job.cancel()
```

Compute Async(hrounously)

```
val job:Job = GlobalScope.launch {
    val data:Deferred<MyDataClass> = async {
        getDataFromServer()
    }
    data.await()
}
```

Deferred

- **Deferred Extends Job**
- **Wrapper around your data**

```
val job:Job = GlobalScope.launch {
    val data:Deferred<MyDataClass> = async {
        getDataFromServer()
    }
    data.await()
}
```

Suspending Function

- **suspend is a Keyword in Kotlin**

```kotlin
suspend fun myLongRunningFunction() {
  //do your staff
}
```

Suspending Function

- suspend is a Keyword in Kotlin
- **Compiler level restriction - can't call suspend function outside CoroutineScope**

```
suspend fun myLongRunningFunction() {
  //do your staff
}
```

Suspending Function

- suspend is a Keyword in Kotlin
- Compiler level restriction - can’t call suspend function outside CoroutineScope
- **Suspends execution of current coroutine**

```
suspend fun myLongRunningFunction() {
  //do your staff
}
```

Coroutine Scope

- **Container of CoroutineContext**

```
public interface CoroutineScope {
    public val coroutineContext: CoroutineContext
}
```

Coroutine Scope

- Container of CoroutineContext
- **Every Coroutine Builder is an extension over CoroutineScope, and thus inherits its CoroutineContext**

```
public interface CoroutineScope {
    public val coroutineContext: CoroutineContext
}
```

Coroutine Scope

- **Container of CoroutineContext**
- **Every Coroutine Builder is an extension over CoroutineScope, and thus inherits its CoroutineContext**
- **All Coroutines must be launched within a CoroutineScope**

```
public interface CoroutineScope {
    public val coroutineContext: CoroutineContext
}
```

Coroutine Scope

```
object MyCoroutineScope : CoroutineScope {
    override val coroutineContext: CoroutineContext = newSingleThreadContext(
            "Single-Thread-Context")
}
```

Dispatchers

```
object MyCoroutineScope : CoroutineScope {
    override val coroutineContext: CoroutineContext = Dispatchers.IO
}
```

- **Determines which Thread / ThreadPool, the coroutine will run on.**
- **Similar to the Rx Schedulers**


Let’s Build for
Android

Repository

```
class Repository(private val gitHubApi: GitHubApi) {
    suspend fun searchRepos(query: String): List<GitHubRepo> =
        gitHubApi.searchRepos(query).await().asGitHubRepoList
}
```

API (Retrofit)

```
@GET("/search/repositories")
fun searchRepos(@Query("q") query: String): Deferred<GitHubRepos>
```

Activity

```
class MainActivity : AppCompatActivity(), CoroutineScope {
    val job = Job()
    override val coroutineContext: CoroutineContext = Dispatchers.IO + job

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        //...
    }

    override fun onDestroy() {
        super.onDestroy()
        job.cancel()
    }
}
```

Call it

```
private fun loadData(view:View = fab) {
    Snackbar.make(view, "Loading", Snackbar.LENGTH_INDEFINITE).show()
    launch(exceptionHandler) {
        val repolist = Repository.defaultRepository.searchRepos("Kotlin")
        withContext(Dispatchers.Main) {
            adapter.items = repolist
            Snackbar.make(view, "Data Loaded", Snackbar.LENGTH_LONG).show()
        }
    }
}
```

Handle Exception (Gracefully) :)

```
val exceptionHandler = CoroutineExceptionHandler {
    _,e->
    Snackbar.make(fab, "Error Loading Data", Snackbar.LENGTH_INDEFINITE).show()
}
```

Resources

- http://bit.ly/CodeLabCoroutines (Codelab)
- http://bit.ly/CoroutinesAndroid (Article)
- http://bit.ly/DroidCoroutines (Book, yet to be published)
- http://bit.ly/CoroutinesGuide (Official Guide)
- https://www.meetup.com/BlrKotlin/ (Bangalore Kotlin User Group)

https://caster.io/courses/kotlin-coroutines-fundamentals

Rivu Chakraborty
@rivuchakraborty

Thank you!

Rivu Chakraborty
@rivuchakraborty