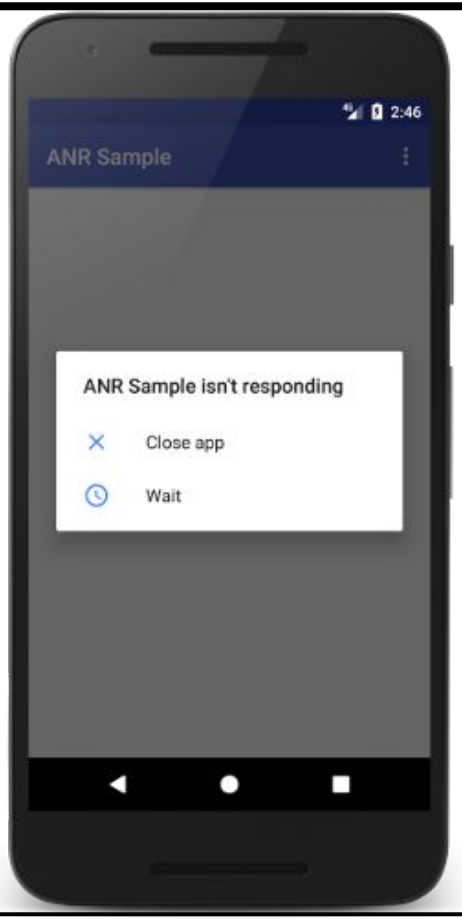
Coroutines

Andres Rubiano Del Chiaro
Rappi Inc.
@andresrubianoch


2:46
ANR Sample
ANR Sample isn't responding
Close app
Wait

Before Coroutines | Ways to do async on Android

AsyncTasks

Loaders

Executors

ThreadPools

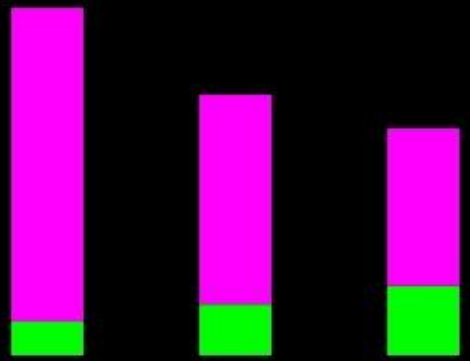
Rx

Callback hells!

```
function register()
{
    if (!empty($_POST)) {
        $msg = '';
        if ($_POST['user_name']) {
            if ($_POST['user_password_new']) {
                if ($_POST['user_password_new'] === $_POST['user_password_repeat']) {
                    if (strlen($_POST['user_password_new']) > 5) {
                        if (strlen($_POST['user_name']) < 65 && strlen($_POST['user_name']) > 1) {
                            if (preg_match('/^[a-z\d]{2,64}$/i', $_POST['user_name'])) {
                                $user = read_user($_POST['user_name']);
                                if (!isset($user['user_name'])) {
                                    if ($_POST['user_email']) {
                                        if (strlen($_POST['user_email']) < 65) {
                                            if (filter_var($_POST['user_email'], FILTER_VALIDATE_EMAIL)) {
                                                create_user();
                                                $_SESSION['msg'] = 'You are now registered so please login';
                                                header('Location: ' . $_SERVER['PHP_SELF']);
                                                exit();
                                            } else $msg = 'You must provide a valid email address';
                                        } else $msg = 'Email must be less than 64 characters';
                                    } else $msg = 'Email cannot be empty';
                                } else $msg = 'Username already exists';
                            } else $msg = 'Username must be only a-z, A-Z, 0-9';
                        } else $msg = 'Username must be between 2 and 64 characters';
                    } else $msg = 'Password must be at least 6 characters';
                } else $msg = 'Passwords do not match';
            } else $msg = 'Empty Password';
        } else $msg = 'Empty Username';
        $_SESSION['msg'] = $msg;
    }
    return register_form();
}
```

icompile.eladkarako.com

~ 90% of your time is reading



What if you'd
ddouble write-time
and thereby
cut read-time
by half?

Code for humans not machines

Think that the next person who reads your code is a chainsaw maniac

If you don't write clean code, you know your fate.

Coroutines

Don’t block
Keep moving

A Tasks — Threads

The most interesting thing is that a thread can stop executing a coroutine at some specific “suspension points”, and go do some other work. It can resume executing the coroutine later on, or another thread could even take over.

Coroutines | Brief history

Founded on Continuation principle

Concept first coined in 1958 by Melvin Conway

Like threads, but far away more lightweight

Tasks that can be suspended and resumed

Stable as of Kotlin 1.3

https://github.com/Kotlin/kotlinx.coroutines

Coroutines | Brief history

Founded on Continuation principle

Concept first coined in 1958 by Melvin Conway

Like threads, but far away more lightweight

Tasks that can be suspended and resumed

Stable as of Kotlin 1.3

https://github.com/Kotlin/kotlinx.coroutines

ARCTOJCH

Coroutines | Brief history

Founded on Continuation principle

Concept first coined in 1958 by Melvin Conway

Like threads, but far away more lightweight

Tasks that can be suspended and resumed

Stable as of Kotlin 1.3

Coroutines | Brief history

Founded on Continuation principle

Concept first coined in 1958 by Melvin Conway

Like threads, but far away more lightweight

Tasks that can be suspended and resumed

Stable as of Kotlin 1.3

https://github.com/Kotlin/kotlinx.coroutines

Coroutines | Brief history

Founded on Continuation principle

Concept first coined in 1958 by Melvin Conway

Like threads, but far away more lightweight

Tasks that can be suspended and resumed

Stable as of Kotlin 1.3

https://github.com/Kotlin/kotlinx.coroutines

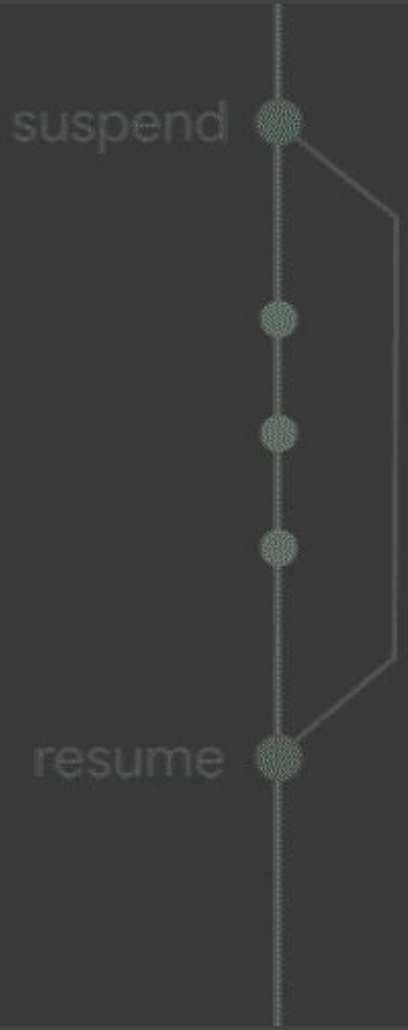
But where's the magic?

```
fun launch(
    context: CoroutineContext,
    block: suspend () -> Unit
): Job
```

Suspending lambda

Suspending functions may suspend the execution of the current coroutine without blocking the current thread.

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



Main Thread
[stack]

Coroutines Builder

runBlocking

Launch

Async { await } : Deferred

withContext(Dispatcher)


SHUT UP
AND SHOW ME THE CODE

```
1   fun main() {
2       GlobalScope.launch {
3           delay(1000L)
4           println("World!")
5       }
6       println("Hello,")
7       runBlocking {
8           delay(2000L)
9       }
10  }
```

What is going to be the outcome?

a) World!
Hello,

b) Hello,
World!

c) Runtime Exception

```
1   fun main() {
2       GlobalScope.launch { // launch new coroutine in background and continue
3           delay(1000L)
4           println("World!")
5       }
6       println("Hello,") // main thread continues here immediately
7       runBlocking {     // but this expression blocks the main thread
8           delay(2000L)  // ... while we delay for 2 seconds to keep JVM alive
9       }
10  }
```

What is going to be the outcome?

a) World!
Hello,

b) Hello,
World!

c) Runtime Exception

```
suspend fun printlnDelayed(message: String) {
    // Complex calculation
    delay( timeMillis: 1000)
    println(message)
}

fun exampleBlocking() {
    println("one")
    runBlocking { this: CoroutineScope
        printlnDelayed("two")
    }
    println("three")
```

```
suspend fun printlnDelayed(message: String) {
    // Complex calculation
    delay( timeMillis: 1000)
    println(message)
}

fun exampleBlocking() {
    println("one")
    runBlocking { this: CoroutineScope
        printlnDelayed("two")
    }
    println("three")
```

```
suspend fun printlnDelayed(message: String) {
    // Complex calculation
    delay( timeMillis: 1000)
    println(message)
}

fun exampleBlocking() {
    println("one")
    runBlocking { this: CoroutineScope
        printlnDelayed("two")
    }
    println("three")
```

```kotlin
suspend fun printlnDelayed(message: String) {
    // Complex calculation
    delay( timeMillis: 1000)
    println(message)
}

fun exampleBlocking() {
    println("one")
    runBlocking { this: CoroutineScope
        printlnDelayed("two")
    }
    println("three")
```

Outcome?

a) one
three
two

b) two
three
one

c) one
two
three

```
suspend fun printlnDelayed(message: String) {
    // Complex calculation
    delay( timeMillis: 1000)
    println(message)
}

fun exampleBlocking() {
    println("one")
    runBlocking { this: CoroutineScope
        printlnDelayed("two")
    }
    println("three")
```

Outcome?

a) one
three
two

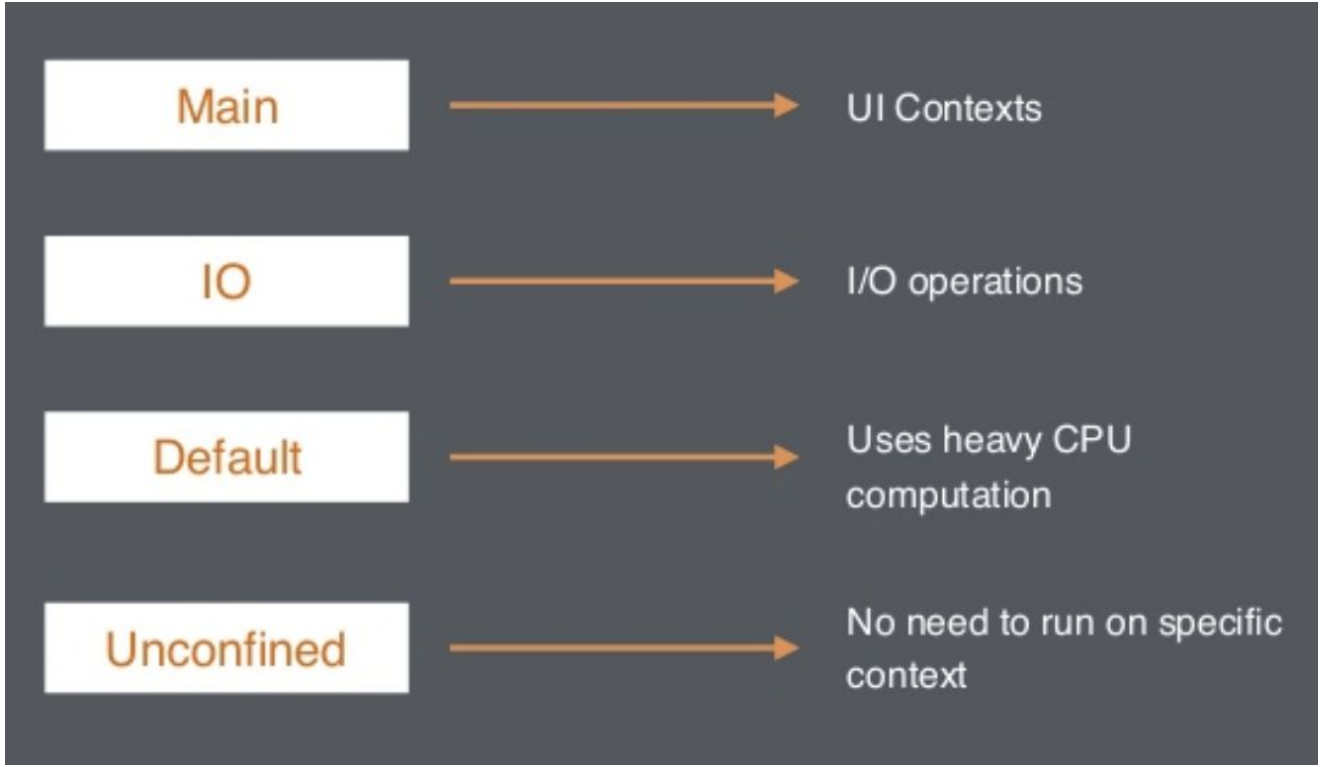
b) two
three
one

c) one
two
three

runBlocking as a method root

```
fun exampleBlocking() = runBlocking { this: CoroutineScope
    println("one")
    printlnDelayed("two")
    println("three")
}
```

Dispatchers



runBlocking + Dispatchers

```
// Running on another thread but still blocking the main thread
fun exampleBlockingDispatcher(){
    runBlocking(Dispatchers.Default) { this: CoroutineScope
        println("one - from thread ${Thread.currentThread().name}")
        printlnDelayed("two - from thread ${Thread.currentThread().name}")
    }
    // Outside of runBlocking to show that it's running in the blocked main thread
    println("three - from thread ${Thread.currentThread().name}")
    // It still runs only after the runBlocking is fully executed.
}
```

runBlocking + Dispatchers

```
// Running on another thread but still blocking the main thread
fun exampleBlockingDispatcher(){
    runBlocking(Dispatchers.Default) { this: CoroutineScope
        println("one - from thread ${Thread.currentThread().name}")
        printlnDelayed("two - from thread ${Thread.currentThread().name}")
    }
    // Outside of runBlocking to show that it's running in the blocked main thread
    println("three - from thread ${Thread.currentThread().name}")
    // It still runs only after the runBlocking is fully executed.
}
```

```
one - from thread DefaultDispatcher-worker-1
two - from thread DefaultDispatcher-worker-1
three - from thread main
```

Using Launch in Global Scope

```
fun exampleLaunchGlobal() = runBlocking { this: CoroutineScope
    println("one - from thread ${Thread.currentThread().name}")

    GlobalScope.launch { this: CoroutineScope
        printlnDelayed("two - from thread ${Thread.currentThread().name}")
    }

    println("three - from thread ${Thread.currentThread().name}")
}
```

Using Launch in Global Scope

```
fun exampleLaunchGlobal() = runBlocking { this: CoroutineScope
    println("one - from thread ${Thread.currentThread().name}")

    GlobalScope.launch { this: CoroutineScope
        printlnDelayed("two - from thread ${Thread.currentThread().name}")
    }

    println("three - from thread ${Thread.currentThread().name}")
}
```

outcome ->

```
one - from thread main
three - from thread main
```

Using Launch in Global Scope + delay

```
fun exampleLaunchGlobal() = runBlocking { this: CoroutineScope
    println("one - from thread ${Thread.currentThread().name}")

    GlobalScope.launch { this: CoroutineScope
        printlnDelayed("two - from thread ${Thread.currentThread().name}")
    }

    println("three - from thread ${Thread.currentThread().name}")
    delay( timeMillis: 3000)
}
```

outcome -> ???

Using Launch in Global Scope + delay

```
fun exampleLaunchGlobal() = runBlocking { this: CoroutineScope
    println("one - from thread ${Thread.currentThread().name}")

    GlobalScope.launch { this: CoroutineScope
        printlnDelayed("two - from thread ${Thread.currentThread().name}")
    }

    println("three - from thread ${Thread.currentThread().name}")
    delay( timeMillis: 3000)
}
```

outcome ->

```
one - from thread main
three - from thread main
two - from thread DefaultDispatcher-worker-1
```

Spoiler de tu vida…
BAD PRACTICE!

Using Launch in Global Scope + job

```
fun exampleLaunchGlobalWaiting() = runBlocking { this: CoroutineScope
    println("one - from thread ${Thread.currentThread().name}")

    val job = GlobalScope.launch { this: CoroutineScope
        printlnDelayed("two - from thread ${Thread.currentThread().name}")
    }

    println("three - from thread ${Thread.currentThread().name}")
    job.join()
}
```

outcome ->

```
one - from thread main
three - from thread main
two - from thread DefaultDispatcher-worker-1
```

Using Launch in Global Scope + job

```
fun exampleLaunchCoroutineScope() = runBlocking { this: CoroutineScope
    println("one - from thread ${Thread.currentThread().name}")

    launch { this: CoroutineScope
        printlnDelayed("two - from thread ${Thread.currentThread().name}")
    }

    println("three - from thread ${Thread.currentThread().name}")
}
```

outcome -> ???

Using Launch in Global Scope + job

```
fun exampleLaunchCoroutineScope() = runBlocking { this: CoroutineScope
    println("one - from thread ${Thread.currentThread().name}")

    launch { this: CoroutineScope
        printlnDelayed("two - from thread ${Thread.currentThread().name}")
    }

    println("three - from thread ${Thread.currentThread().name}")
}
```

outcome ->

```
one - from thread main
three - from thread main
two - from thread main
```

Using Launch in Global Scope + job

```
fun exampleLaunchCoroutineScope() = runBlocking { this: CoroutineScope
    println("one - from thread ${Thread.currentThread().name}")

    launch(Dispatchers.) { this: CoroutineScope
        printlnDelay(
    }
    Default        CoroutineDispatcher
    IO             CoroutineDispatcher
    Main       MainCoroutineDispatcher
    Unconfined     CoroutineDispatcher
    equals(other: Any?)        Boolean
    kotlinx.coroutines.Dispatchers
    @JvmStatic
    public final val Unconfined: CoroutineDispatche
    A coroutine dispatcher that is not confined to any
    specific thread. It executes initial continuation of the
    println("three -
}
```

Custom Dispatchers

```
fun exampleLaunchCoroutineScope() = runBlocking { this: CoroutineScope
    println("one - from thread ${Thread.currentThread().name}")

    val customDispatcher = Executors.newFixedThreadPool( nThreads: 2).asCoroutineDispatcher()

    launch(customDispatcher) { this: CoroutineScope
        printlnDelayed("two - from thread ${Thread.currentThread().name}")
    }

    println("three - from thread ${Thread.currentThread().name}")
}
```

```
"C:\Program Files\Java\jdk1.8.0_152\bin\java.exe" ...
one - from thread main
three - from thread main
two - from thread pool-1-thread-1
```

Custom Dispatchers

```
fun exampleLaunchCoroutineScope() = runBlocking { this: CoroutineScope
    println("one - from thread ${Thread.currentThread().name}")

    val customDispatcher = Executors.newFixedThreadPool( nThreads: 2).asCoroutineDispatcher()

    launch(customDispatcher) { this: CoroutineScope
        printlnDelayed("two - from thread ${Thread.currentThread().name}")
    }

    println("three - from thread ${Thread.currentThread().name}")

    (customDispatcher.executor as ExecutorService).shutdown()
}
```

```
"C:\Program Files\Java\jdk1.8.0_152\bin\java.exe" ...
one - from thread main
three - from thread main
two - from thread pool-1-thread-1

Process finished with exit code 0
```

Async + await (parallelism)

```
suspend fun calculateHardThings(startNum: Int): Int {
    delay( timeMillis: 1000)
    return startNum * 10
}
```

```
fun exampleAsyncAwait() = runBlocking { this: CoroutineScope
    val startTime = System.currentTimeMillis()

    val deferred1 = async { calculateHardThings( startNum: 10) }
    val deferred2 = async { calculateHardThings( startNum: 20) }
    val deferred3 = async { calculateHardThings( startNum: 30) }

    val sum = deferred1.await() + deferred2.await() + deferred3.await()
    println("async/await result = $sum")

    val endTime = System.currentTimeMillis()
    println("Time taken: ${endTime - startTime}")
}
```

outcome -> ???

Async + await (parallelism)

```
fun exampleAsyncAwait() = runBlocking { this: CoroutineScope
    val startTime = System.currentTimeMillis()

    val deferred1 = async { calculateHardThings( startNum: 10) }
    val deferred2 = async { calculateHardThings( startNum: 20) }
    val deferred3 = async { calculateHardThings( startNum: 30) }

    val sum = deferred1.await() + deferred2.await() + deferred3.await()
    println("async/await result = $sum")

    val endTime = System.currentTimeMillis()
    println("Time taken: ${endTime - startTime}")
}
```

outcome

```
async/await result = 600
Time taken: 1028

Process finished with exit code 0
```

Async + await (suspending)

```
fun exampleAsyncAwait() = runBlocking { this: CoroutineScope
    val startTime = System.currentTimeMillis()

    val deferred1 = async { calculateHardThings( startNum: 10) }.await()
    val deferred2 = async { calculateHardThings( startNum: 20) }.await()
    val deferred3 = async { calculateHardThings( startNum: 30) }.await()

    val sum = deferred1 + deferred2 + deferred3
    println("async/await result = $sum")

    val endTime = System.currentTimeMillis()
    println("Time taken: ${endTime - startTime}")
}
```

outcome -> ???

Async + await (suspending)

```
fun exampleAsyncAwait() = runBlocking { this: CoroutineScope
    val startTime = System.currentTimeMillis()

    val deferred1 = async { calculateHardThings( startNum: 10) }.await()
    val deferred2 = async { calculateHardThings( startNum: 20) }.await()
    val deferred3 = async { calculateHardThings( startNum: 30) }.await()

    val sum = deferred1 + deferred2 + deferred3
    println("async/await result = $sum")

    val endTime = System.currentTimeMillis()
    println("Time taken: ${endTime - startTime}")
}
```

```
async/await result = 600
Time taken: 3032

Process finished with exit code 0
```

withContext

```
fun exampleWithContext() = runBlocking { this: CoroutineScope
    val startTime = System.currentTimeMillis()

    val result1 = withContext(Dispatchers.Default) { calculateHardThings( startNum: 10) }
    val result2 = withContext(Dispatchers.Default) { calculateHardThings( startNum: 20) }
    val result3 = withContext(Dispatchers.Default) { calculateHardThings( startNum: 30) }

    val sum = result1 + result2 + result3
    println("async/await result = $sum")

    val endTime = System.currentTimeMillis()
    println("Time taken: ${endTime - startTime}")
}
```

outcome -> ???

withContext

```
async/await result = 600
Time taken: 3069

Process finished with exit code 0
```

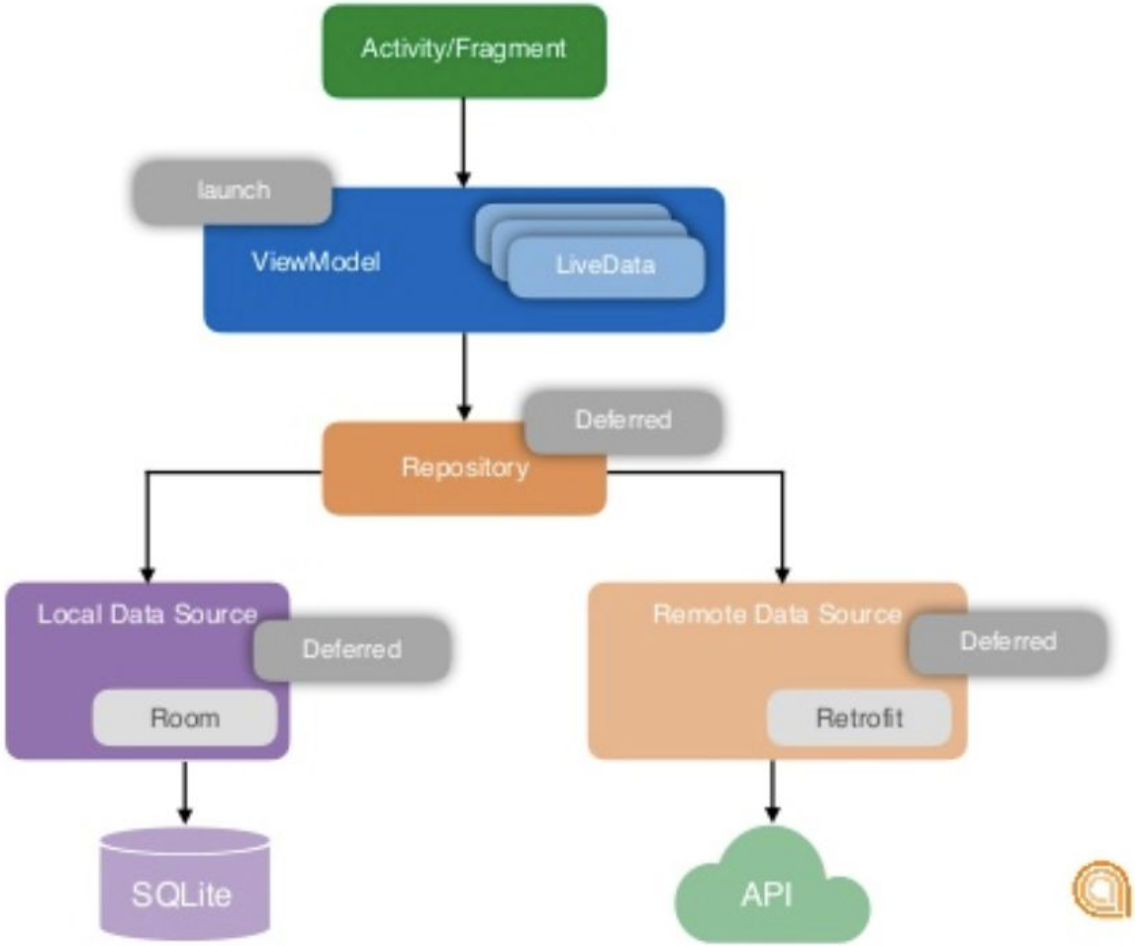
outcome

```
fun exampleWithContext() = runBlocking { this: CoroutineScope
    val startTime = System.currentTimeMillis()

    val result1 = withContext(Dispatchers.Default) { calculateHardThings( startNum: 10) }
    val result2 = withContext(Dispatchers.Default) { calculateHardThings( startNum: 20) }
    val result3 = withContext(Dispatchers.Default) { calculateHardThings( startNum: 30) }

    val sum = result1 + result2 + result3
    println("async/await result = $sum")

    val endTime = System.currentTimeMillis()
    println("Time taken: ${endTime - startTime}")
}
```


Activity/Fragment
launch
ViewModel
LiveData
Deferred
Repository
Local Data Source
Deferred
Room
Remote Data Source
Deferred
Retrofit
SQLite
API

ARCTOJCH

Coroutine builder | withContext - Switching contexts

```
val uiContext: CoroutineContext = Dispatchers.Main

val ioContext: CoroutineContext = Dispatchers.IO

fun loadTrending() = launch(uiContext) {
    _uiStateEvent.postValue(UiState.Loading)

    val result = withContext(ioContext){ repository.loadTrending() }

    _trendingData.postValue(Result.success(result))
    _uiStateEvent.postValue(UiState.Loaded)
}
```