# Diving into coroutines for JVM

*Enrique López Mañas*

## Ego slide

```
- Android and Kotlin for backend
- GDE member
- @eenriquelopez
```

**Coroutines** are computer program components that generalize subroutines for non-preemptive multitasking, by allowing execution to be suspended and resumed. **Coroutines** are well-suited for implementing familiar program components such as cooperative tasks, exceptions, event loops, iterators, infinite lists and pipes.

## Why do we want coroutines?

```kotlin
val user = fetchUserData()
textView.text = user.name
```

NetworkOnMainThreadException

## Why do we want coroutines?

```
thread {
    val user = fetchUserData()
    textView.text = user.name
}
```

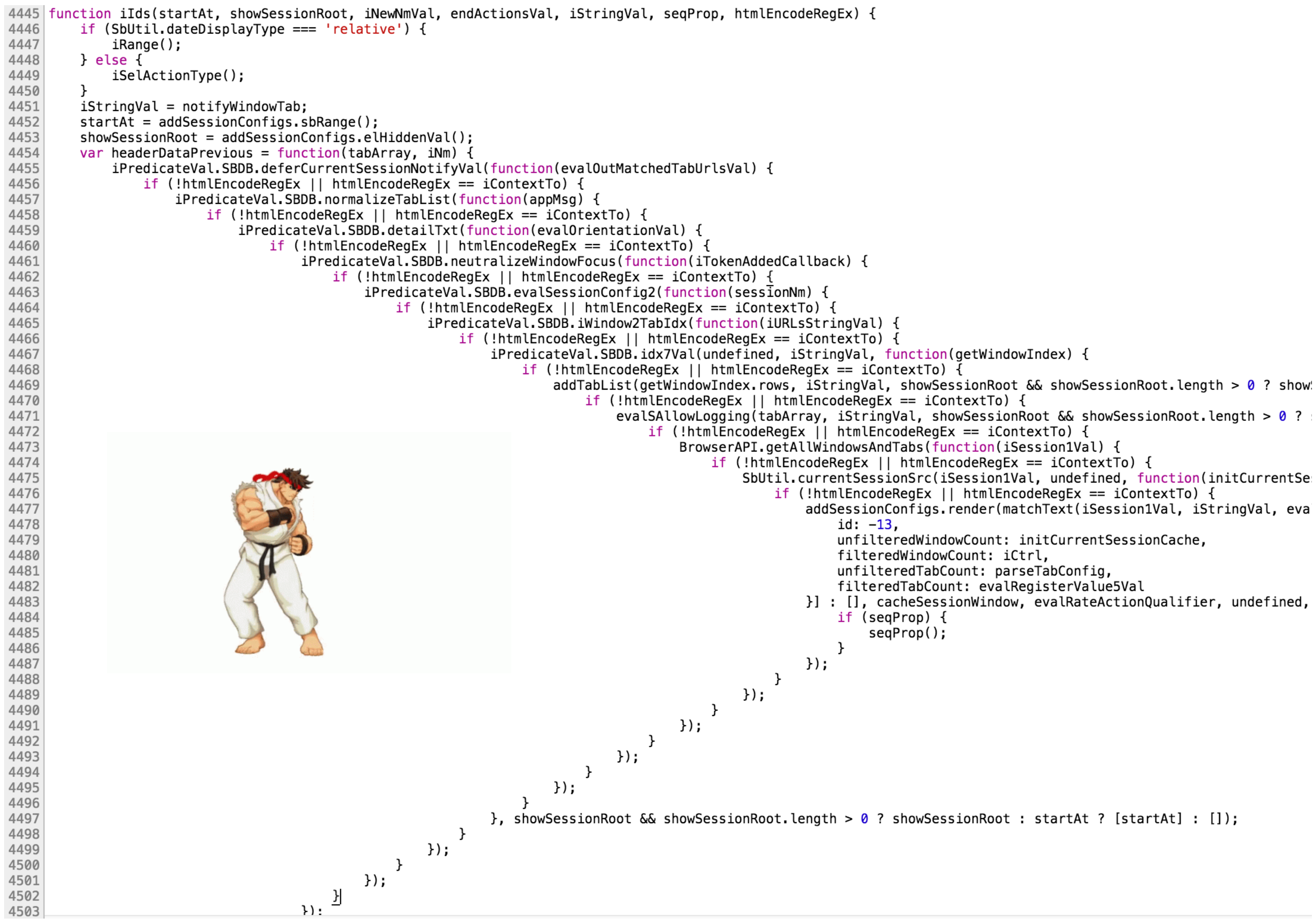
CalledFromAnotherThreadException

## Why do we want coroutines?

```
fetchUserData { user -> //callback
    textView.text = user.name
}
```

Leaking callbacks all the time

# Why do we want coroutines?



```
function iIds(startAt, showSessionRoot, iNewNmVal, endActionsVal, iStringVal, seqProp, htmlEncodeRegEx) {
    if (SbUtil.dateDisplayType === 'relative') {
        iRange();
    } else {
        iSelActionType();
    }
    iStringVal = notifyWindowTab;
    startAt = addSessionConfigs.sbRange();
    showSessionRoot = addSessionConfigs.elHiddenVal();
    var headerDataPrevious = function(tabArray, iNm) {
        iPredicateVal.SBDB.deferCurrentSessionNotifyVal(function(evalOutMatchedTabUrlsVal) {
            if (!htmlEncodeRegEx || htmlEncodeRegEx == iContextTo) {
                iPredicateVal.SBDB.normalizeTabList(function(appMsg) {
                    if (!htmlEncodeRegEx || htmlEncodeRegEx == iContextTo) {
                        iPredicateVal.SBDB.detailTxt(function(evalOrientationVal) {
                            if (!htmlEncodeRegEx || htmlEncodeRegEx == iContextTo) {
                                iPredicateVal.SBDB.neutralizeWindowFocus(function(iTokenAddedCallback) {
                                    if (!htmlEncodeRegEx || htmlEncodeRegEx == iContextTo) {
                                        iPredicateVal.SBDB.evalSessionConfig2(function(sessionNm) {
                                            if (!htmlEncodeRegEx || htmlEncodeRegEx == iContextTo) {
                                                iPredicateVal.SBDB.iWindow2TabIdx(function(iURLsStringVal) {
                                                    if (!htmlEncodeRegEx || htmlEncodeRegEx == iContextTo) {
                                                        iPredicateVal.SBDB.idx7Val(undefined, iStringVal, function(getWindowIndex) {
                                                            if (!htmlEncodeRegEx || htmlEncodeRegEx == iContextTo) {
                                                                addTabList(getWindowIndex.rows, iStringVal, showSessionRoot && showSessionRoot.length > 0 ? show
                                                                    if (!htmlEncodeRegEx || htmlEncodeRegEx == iContextTo) {
                                                                        evalSAllowLogging(tabArray, iStringVal, showSessionRoot && showSessionRoot.length > 0 ?
                                                                            if (!htmlEncodeRegEx || htmlEncodeRegEx == iContextTo) {
                                                                                BrowserAPI.getAllWindowsAndTabs(function(iSession1Val) {
                                                                                    if (!htmlEncodeRegEx || htmlEncodeRegEx == iContextTo) {
                                                                                        SbUtil.currentSessionSrc(iSession1Val, undefined, function(initCurrentSe
                                                                                            if (!htmlEncodeRegEx || htmlEncodeRegEx == iContextTo) {
                                                                                                addSessionConfigs.render(matchText(iSession1Val, iStringVal, eva
                                                                                                    id: -13,
                                                                                                    unfilteredWindowCount: initCurrentSessionCache,
                                                                                                    filteredWindowCount: iCtrl,
                                                                                                    unfilteredTabCount: parseTabConfig,
                                                                                                    filteredTabCount: evalRegisterValue5Val
                                                                                                }] : [], cacheSessionWindow, evalRateActionQualifier, undefined,
                                                                                                    if (seqProp) {
                                                                                                        seqProp();
                                                                                                    }
                                                                                                });
                                                                                            }
                                                                                        });
                                                                                    }
                                                                                });
                                                                            }
                                                                        });
                                                                    }
                                                                });
                                                            }
                                                        }, showSessionRoot && showSessionRoot.length > 0 ? showSessionRoot : startAt ? [startAt] : []);
                                                    }
                                                });
                                            }
                                        });
                                    }
                                });
```

# Why do we want coroutines?

```
val subscription = fetchUserData { user ->
    textView.text = user.name
}

override fun onStop() {
    subscription.cancel()
}
```

## Why do we want coroutines?

```
override fun onStop() {
    subscription.cancel()
    subscription1.cancel()
    subscription2.cancel()
    subscription3.cancel()
    subscription4.cancel()
    subscription5.cancel()
    subscription6.cancel()
    subscription7.cancel()
    subscription8.cancel()
    subscription9.cancel()
}
```

## Why do we want coroutines?

```
object MyTask: AsyncTask() {
    override fun doInBackground { code }
    override fun onPostExecute { code }
}
```

# RxJava

```
fun fetchUser() : Observable<User> = ...

fetchUser()
        .as(autoDisposable(AndroidLifecycleScopeProvider.from(this)))
        .subscribe{ user ->
            textView.text = user.name
        }
```

# LiveData

```kotlin
fun fetchUser() : LiveData<User> = ...

fetchUser().observe(viewLifecycleOwner) {
    textView.text = user.name
}
```

**LiveData**

Observable Data Holder

**RxJava**

**Observable + Schedulers + Observers**

**Coroutines**

Suspendable computations

## LiveData

Not fully complete  (only supports MainThread)

# RxJava

Complete, but:

- Easy to misuse
- Feels like an overkill
- Learning curve

# Coroutines

- Simplified
- Comprehensive
- Robust

First class support from Google in Jetpack

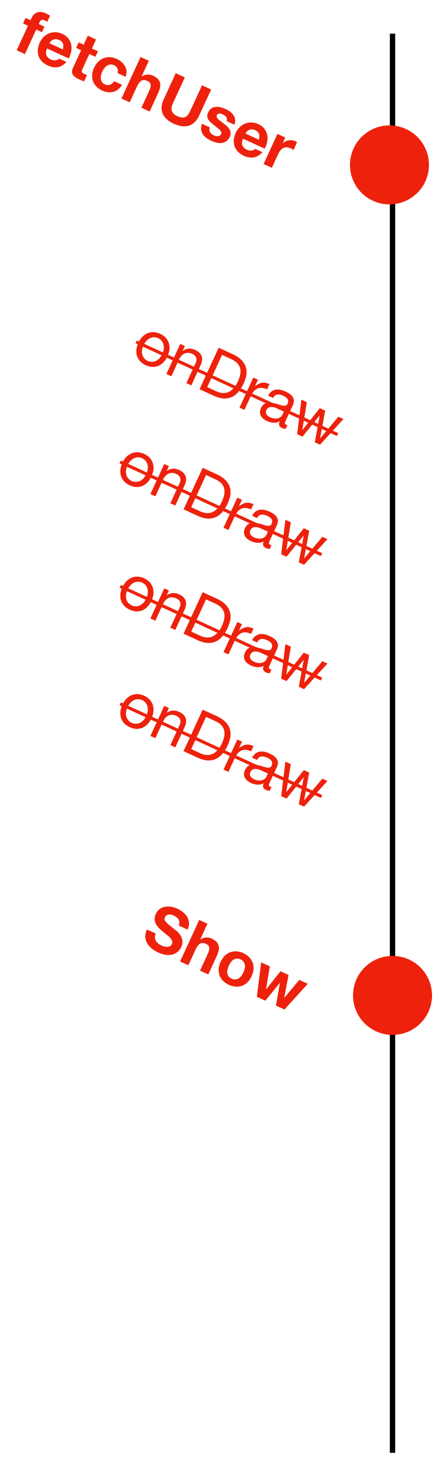
## Coroutines

In a nutshell:

Coroutines simplify async code by replacing callbacks

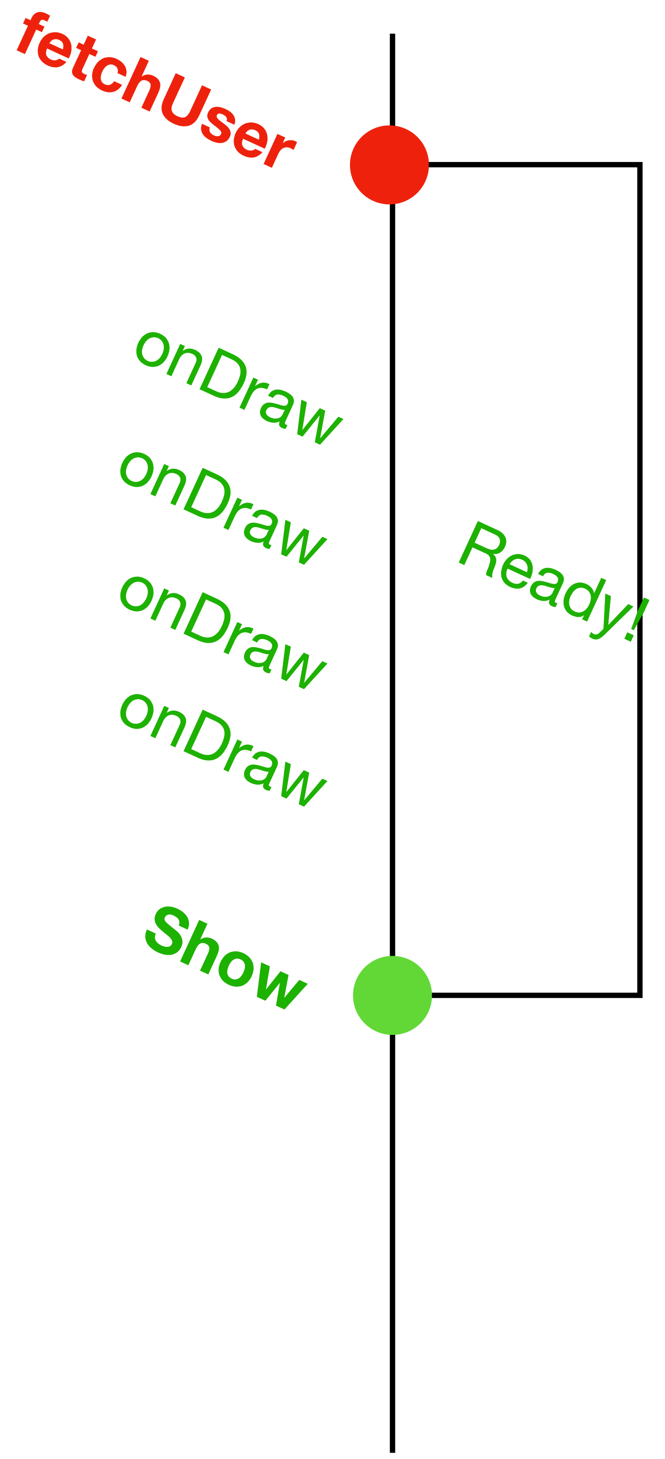
# Coroutines

```
blocking.kt

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



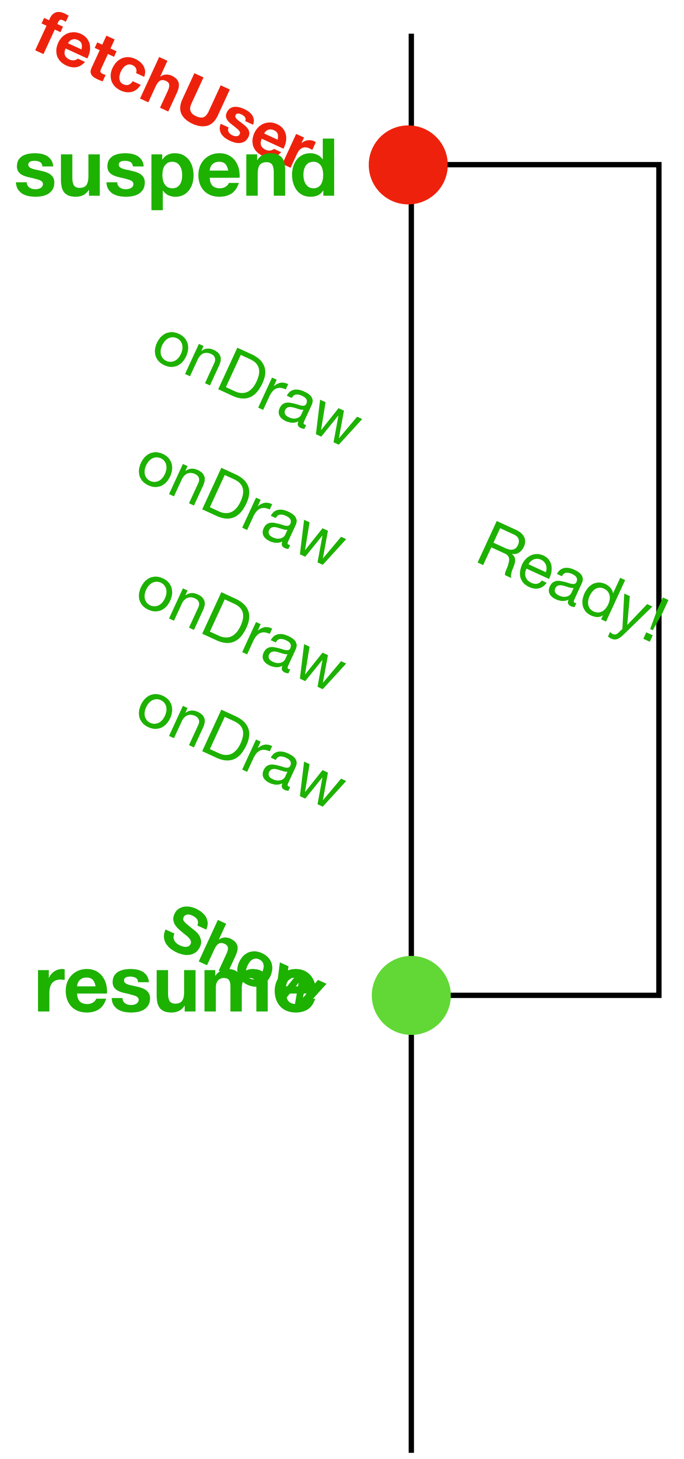
# Coroutines

```
async.kt

fun loadUser() {
    api.fetchUser { user ->
        show(user)
    }
}
```



# Coroutines

```
coroutines.kt

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



# Coroutines

Suspend and resume replace callbacks

# Coroutines

```
suspend fun loadData(): Data
```

$$\big\downarrow$$

```
fun loadData(listener: Continuation<Data>)
```

# Coroutines

KotlinConf 2017 - Deep Dive into Coroutines on JVM by Roman Elizarov

## Coroutines

```
coroutines.kt

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

```
async.kt

fun loadUser() {
    api.fetchUser { user ->
        show(user)
    }
}
```

# Coroutines

```
coroutines.kt

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

```
suspend fun fetchUser() =
          withContext(Dispatchers.IO) {
          }
```

## Dispatchers

| | |
|---|---|
| CPU | .Default |
| Network, Disk | .IO |
| Main Thread on Android | .Main |

## Coroutines

```kotlin
suspend fun fetchUser() =
        withContext(Dispatchers.IO) {
           /*put your blocking calls here*/
        }
```

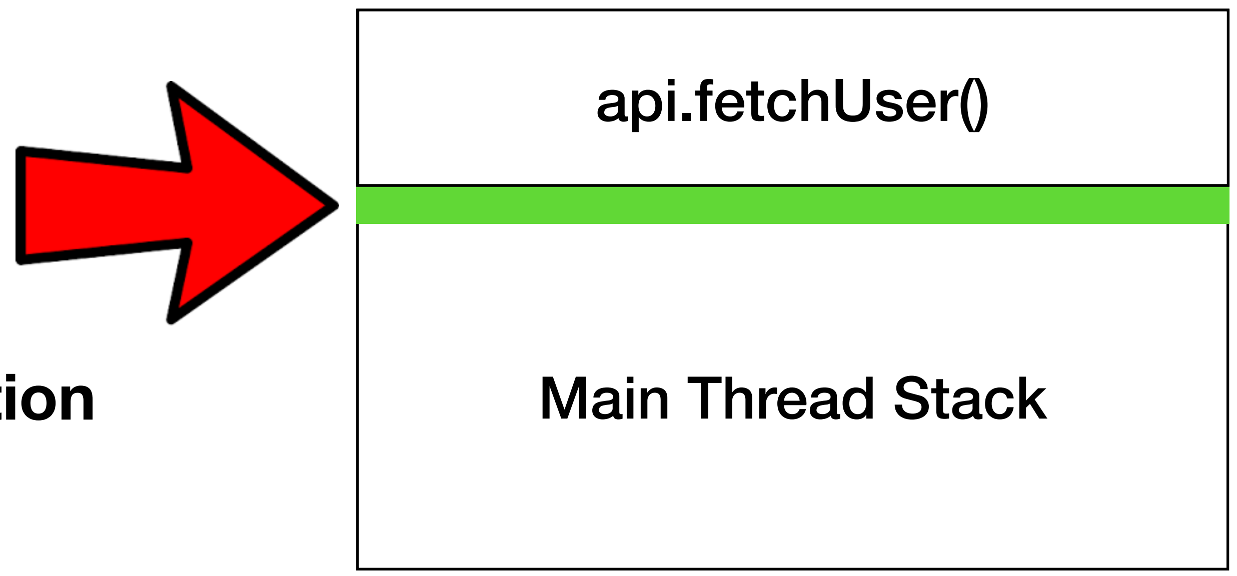
# Under the hood

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

**Suspend marker**

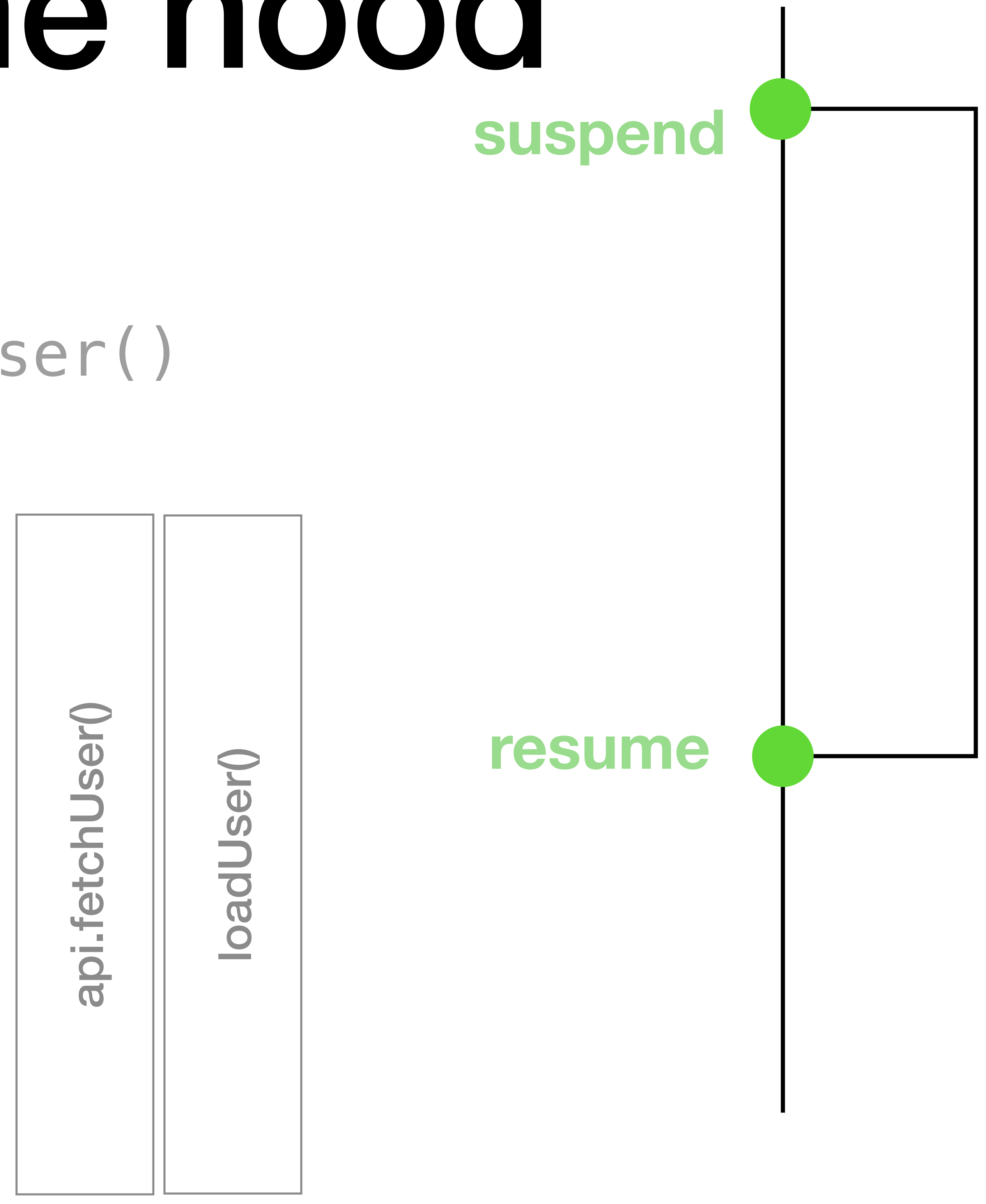
**Everything above a coroutine**

**Everything underneath a regular function**



# Under the hood

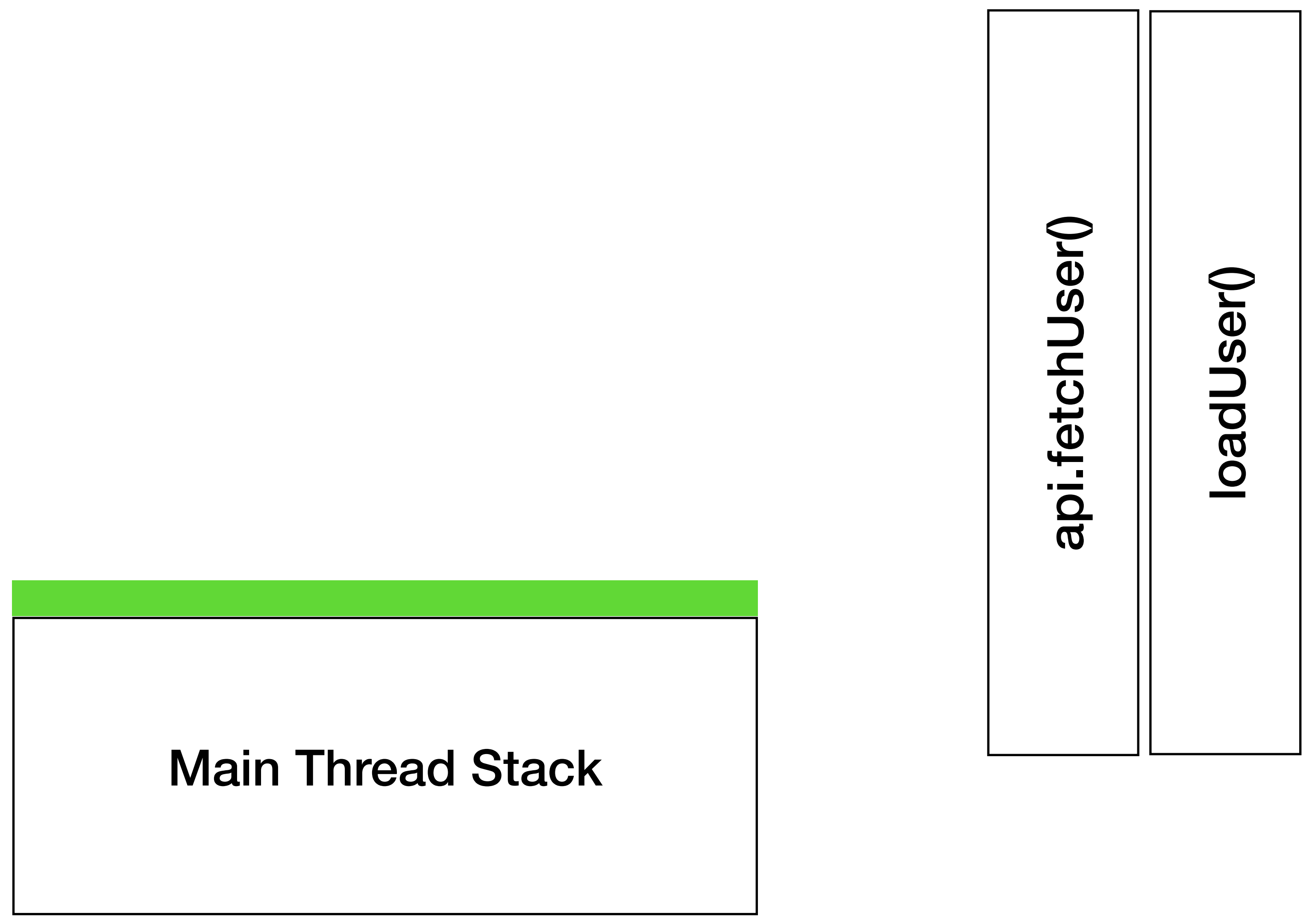
```
suspend fun loadUser() {
    val user = api.fetchUser()
    show(user)
}
```

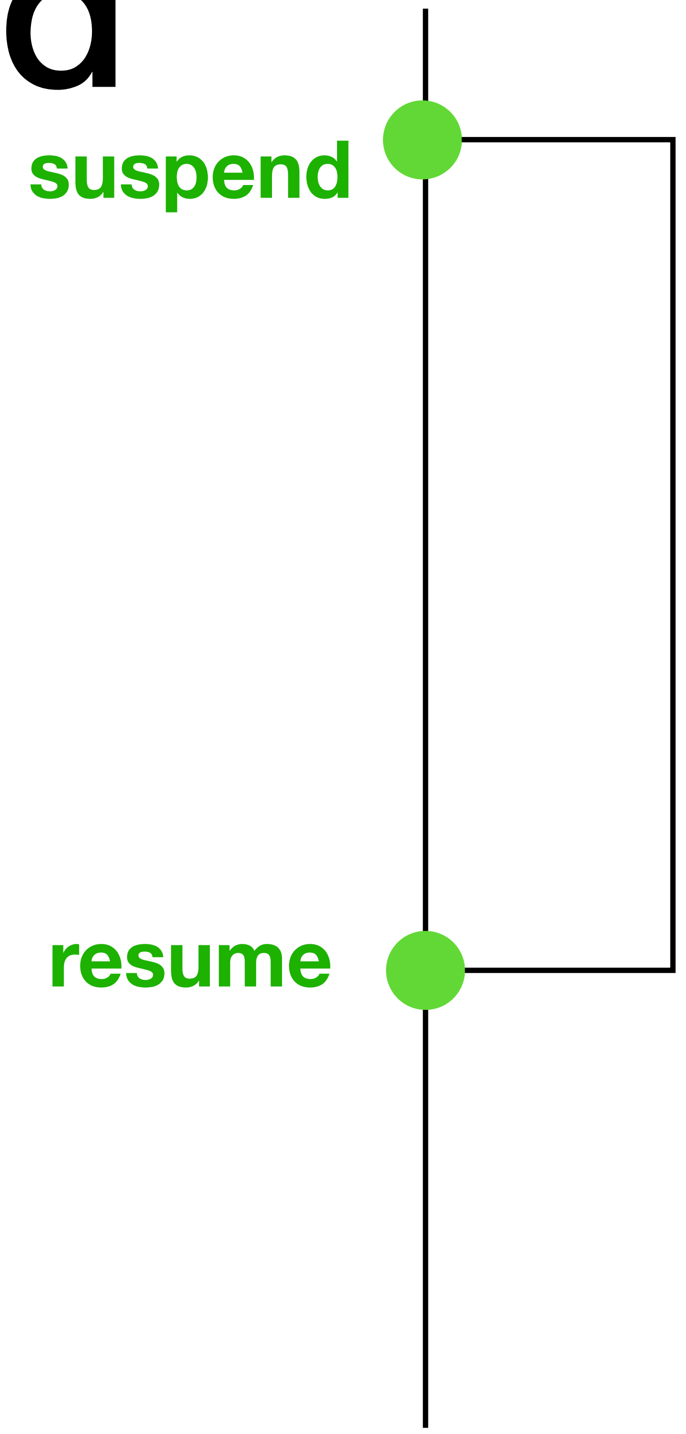


Main Thread Stack

# Under the hood

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

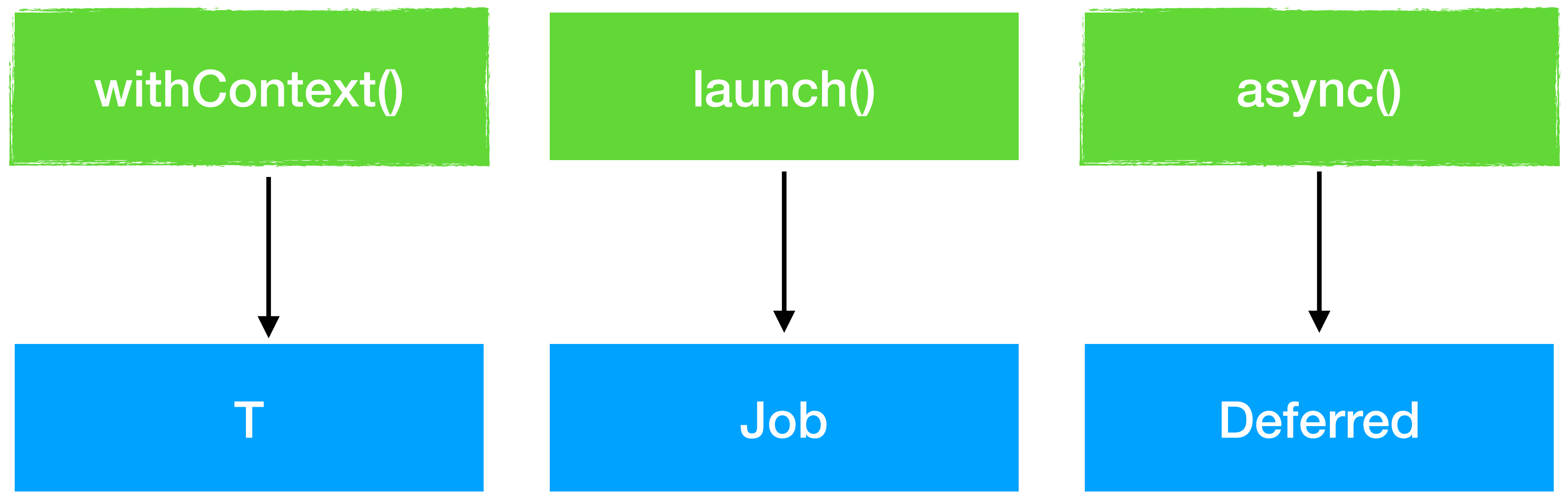




# Summarizing…

-They replace callbacks
- They provide Main Thread safety

# Building blocks



Returns result

Uncaught exception
Crashes app

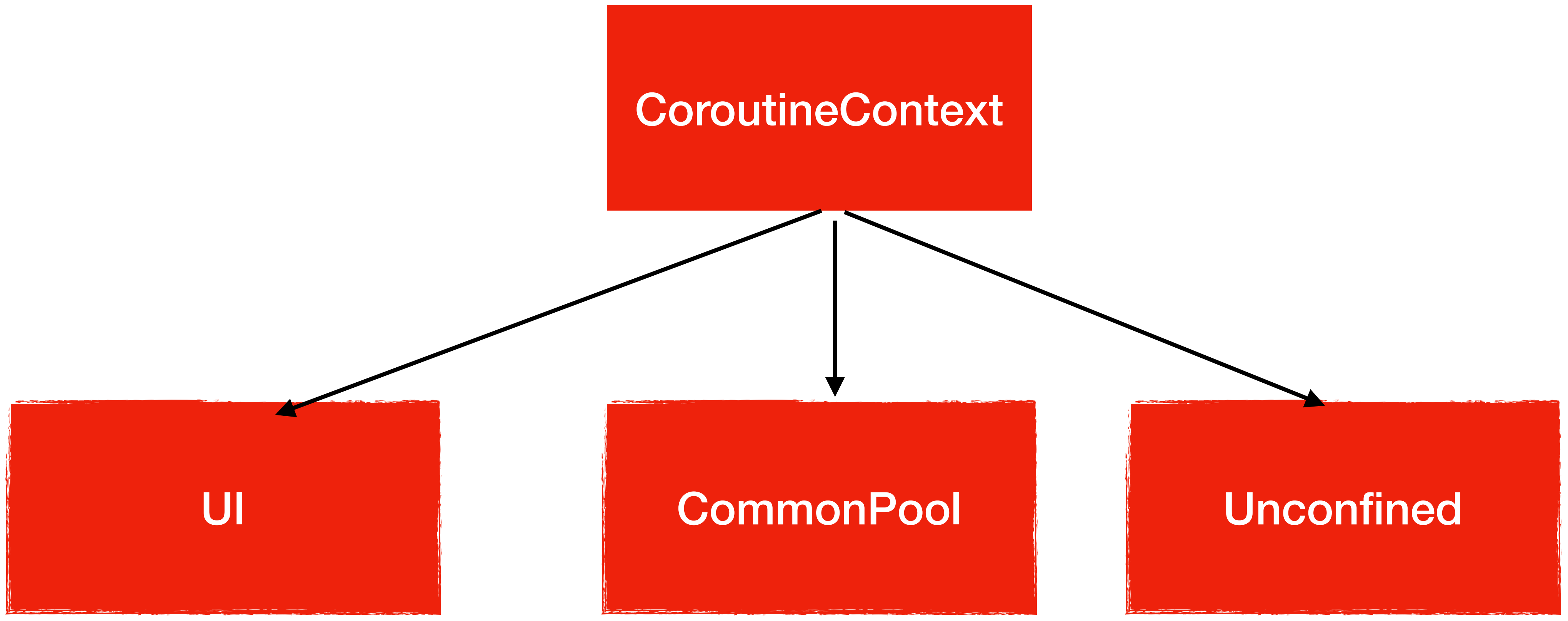
Fire and forget

Uncaught exception
Crashes app

Returns non-blocking
future

Uncaught exception is
returned in deferred

## Building blocks



Dispatch execution into Android MainThread

Dispatch execution into Android Background

Dispatch execution into Current thread

## Launch

## Launch

```
val uiContext: CoroutineContext = UI

val bgContext: CoroutineContext = CommonPool
```

# Launch

```
private fun loadData() {
    view.showLoading()

    val result = dataProvider.provideData()

    view.showData(result)
    view.hideLoading()
}
```

# Launch

```kotlin
private fun loadData() = launch(uiContext) {
    view.showLoading()

    val result = dataProvider.provideData()

    view.showData(result)
    view.hideLoading()
}
```

# Launch

```
private fun loadData() = launch(uiContext) {
    view.showLoading()

    val result = withContext(bgContext) { dataProvider.provideData() }

    view.showData(result)
    view.hideLoading()
}
```

# Launch two tasks sequentially

```
private fun loadData() = launch(uiContext) {
    view.showLoading() // ui thread

    // non ui thread, suspend until task is finished
    val result1 = withContext(bgContext) { dataProvider.provideData() }

    // non ui thread, suspend until task is finished
    val result2 = withContext(bgContext) { dataProvider.provideData() }

    val result = "$result1 $result2" // ui thread
    view.showData(result)
    view.hideLoading()
}
```

# Launch two tasks sequentially

```
private fun loadData() = launch(uiContext) {
    view.showLoading() // ui thread

    // non ui thread, suspend until task is finished
    val result1 = withContext(bgContext) { dataProvider.provideData() }

    // non ui thread, suspend until task is finished
    val result2 = withContext(bgContext) { dataProvider.provideData() }

    val result = "$result1 $result2" // ui thread
    view.showData(result)
    view.hideLoading()
}
```

# Launch two tasks in parallel

```kotlin
private fun loadData() = launch(uiContext) {
    view.showLoading() // ui thread

    val task1 = async(bgContext) { dataProvider.provideData() }
    val task2 = async(bgContext) { dataProvider.provideData() }

    // non ui thread, suspend until finished
    val result = "${task1.await()} ${task2.await()}"

    view.showData(result) // ui thread
    view.hideLoading()
}
```

# Launch two tasks in parallel

```kotlin
private fun loadData() = launch(uiContext) {
    view.showLoading() // ui thread

    val task1 = async(bgContext) { dataProvider.provideData() }
    val task2 = async(bgContext) { dataProvider.provideData() }

    // non ui thread, suspend until finished
    val result = "${task1.await()} ${task2.await()}"

    view.showData(result) // ui thread
    view.hideLoading()
}
```

# Launch two tasks in parallel

```kotlin
private fun loadData() = launch(uiContext) {
    view.showLoading() // ui thread

    val deferred = listOf(
            async(bgContext) { dataProvider.provideData() }
            async(bgContext) { dataProvider.provideData() }
    )


    deferred.awaitAll()
}
```

# Launch coroutine with timeout

```
private fun loadData() = launch(uiContext) {
    view.showLoading() // ui thread

    // non ui thread, suspend until the task is finished
    // or return null in 2 sec
    val result = withTimeoutOrNull(2, TimeUnit.SECONDS) {
        withContext(bgContext) { dataProvider.provideData() }
    }

    view.showData(result) // ui thread
    view.hideLoading()
}
```

# Launch coroutine with timeout

```
private fun loadData() = launch(uiContext) {
    view.showLoading() // ui thread

    // non ui thread, suspend until the task is finished
    // or return null in 2 sec
    val result = withTimeoutOrNull(2, TimeUnit.SECONDS) {
        withContext(bgContext) { dataProvider.provideData() }
    }

    view.showData(result) // ui thread
    view.hideLoading()
}
```

## Cancelation

# Cancel a coroutine

```
private var job: Job? = null

fun onNextClicked() {
    job = loadData()
}

fun onBackClicked() {
    job?.cancel()
}

private fun loadData() = launch(uiContext) {
    // code
}
```

# Cancel a coroutine

```kotlin
private var job: Job? = null

fun onNextClicked() {
    job = loadData()
}

fun onBackClicked() {
    job?.cancel()
}

private fun loadData() = launch(uiContext) {
    // code
}
```

# Cancel a coroutine

```
private var job: Job? = null

fun onNextClicked() {
    job = loadData()
}

fun onBackClicked() {
    job?.cancel()
}

private fun loadData() = launch(uiContext) {
    // code
}
```

# More with jobs

```
job?.isActive
job?.isCancelled
job?.isComplete
job?.getCancellationException()
job?.children
job?.cancelChildren
job?.invokeOnCompletion {}
```

## Error Handling

## Try-catch

```
private fun loadData() = launch(uiContext) {
    view.showLoading() // ui thread

    try {
        val result = withContext(bgContext) { dataProvider.provideData() }
        view.showData(result) // ui thread
    } catch (e: IllegalArgumentException) {
        e.printStackTrace()
    }
    view.hideLoading()
}
```

## Store inside deferred

```
private fun loadData() = async(uiContext) {
    view.showLoading() // ui thread

    val task = async(bgContext) { dataProvider.provideData() }
    val result = task.await() // non ui thread

    view.showData(result) // ui thread
    view.hideLoading()
}
```

## Store inside deferred

```
var job: Deferred = loadData()
job.invokeOnCompletion { it: Throwable? ->
    it?.printStackTrace()
}
```

# Exception Handler

```kotlin
val exceptionHandler= CoroutineExceptionHandler { _, throwable ->
    throwable.printStackTrace()
}


private fun loadData() = launch(uiContext + exceptionHandler) {
    // code
}
```

## Return null if exception

```kotlin
suspend fun <T> Deferred<T>.awaitSafe(): T? = try {
    await()
} catch (e: Exception) {
    e.printStackTrace()
    null
}
```

# Return null if exception

```
private fun loadData() = launch(uiContext) {

    val task = async(bgContext) { dataProvider.provideData() }
    val result = task.awaitSafe()

    if(result != null) {
        // success
    } else {
        // failure
    }

}
```

# Return <T>

```kotlin
suspend fun <T> Deferred<T>.awaitResult(): Result<T> = try {
    Result(success = await(), failure = null)
} catch (e: Exception) {
    Result(success = null, failure = e)
}

data class Result<out T>(val success: T?, val failure: Exception?)
```

# Return <T>

```
private fun loadData() = launch(uiContext) {

    val task = async(bgContext) { dataProvider.provideData() }
    val (success, failure) = task.awaitResult()

    if(success != null) {
        // success T
    } else {
        // failure Exception
    }

}
```

## Testing

# Testing

```
class MainPresenter(val uiContext: CoroutineContext = UI,
                    val bgContext: CoroutineContext = CommonPool) {

    fun loadData() = launch(uiContext) { ... }
}
```

## Testing

```kotlin
@Test
fun test() {
    val presenter = MainPresenter(Unconfined, Unconfined)

    // test
    presenter.loadData()

    // verify
    verify(mockView).showLoading()
    verify(mockDataProvider).provideData()
}
```

## Does this replace RxJava?

# RxJava analogies

**SubscribeOn = Initial Context**

**ObserveOn = withContext**

**Disposable <> Jobs**

# More

```
// retrofit 2
interface MyService {

    @GET("/user")
    fun getUser(): Deferred<User>

    // or

    @GET("/user")
    fun getUser(): Deferred<Response<User>>
}
```

## More

```
@GET("users/{id}")
suspend fun user(@Path("id") id: Long): User
```

# Room

```
@Dao
interface UsersDao {

    @Query("SELECT * FROM users")
    suspend fun getUsers(): List<User>

    @Query("UPDATE users SET age = age + 1 WHERE userId = :userId")
    suspend fun incrementUserAge(userId: String)

    @Insert
    suspend fun insertUser(user: User)

    @Update
    suspend fun updateUser(user: User)

    @Delete
    suspend fun deleteUser(user: User)

}
```

# Room

```
@Dao
abstract class UsersDao {

    @Transaction
    open suspend fun setLoggedInUser(loggedInUser: User) {
        deleteUser(loggedInUser)
        insertUser(loggedInUser)
    }

    @Query("DELETE FROM users")
    abstract fun deleteUser(user: User)

    @Insert
    abstract suspend fun insertUser(user: User)
}
```

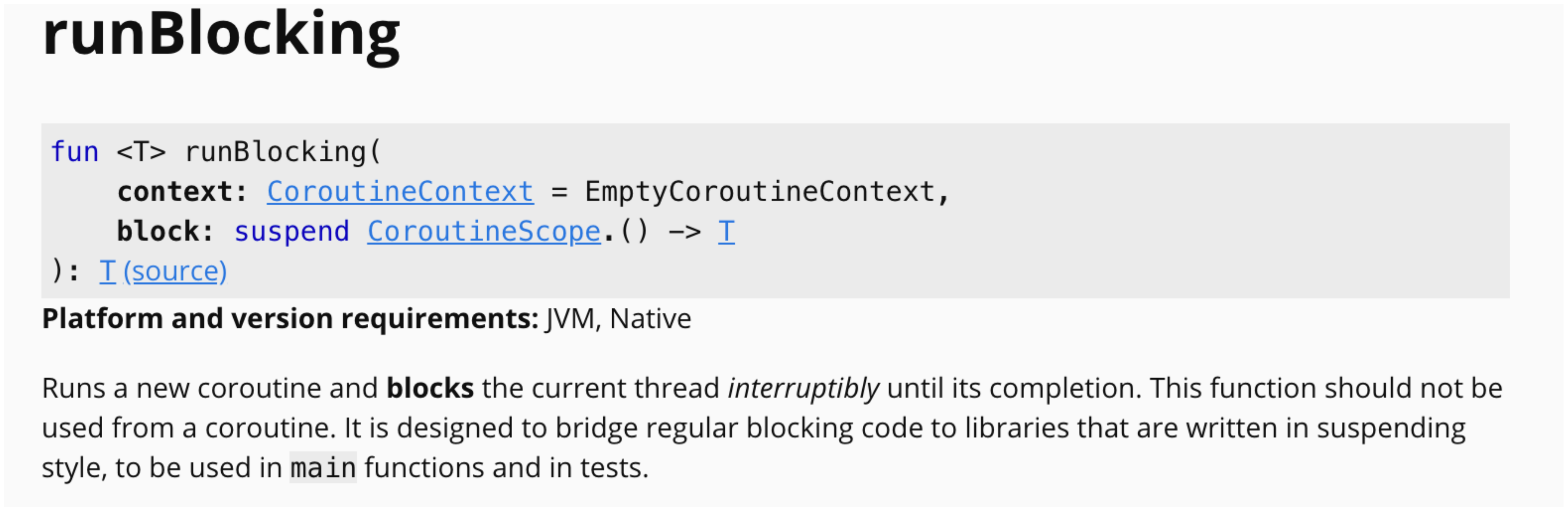
# Room

```
class Repository(val database: MyDatabase) {

    suspend fun clearData(){
        database.withTransaction {
            database.userDao().deleteLoggedInUser() // suspend function
            database.commentsDao().deleteComments() // suspend function
        }
    }
}
```

# Room - Testing

```kotlin
@Test fun insertAndGetUser() = runBlocking {
    // Given a User that has been inserted into the DB
    userDao.insertUser(user)

    // When getting the Users via the DAO
    val usersFromDb = userDao.getUsers()

    // Then the retrieved Users matches the original user object
    assertEquals(listOf(user), userFromDb)
}
```

# Room - Testing

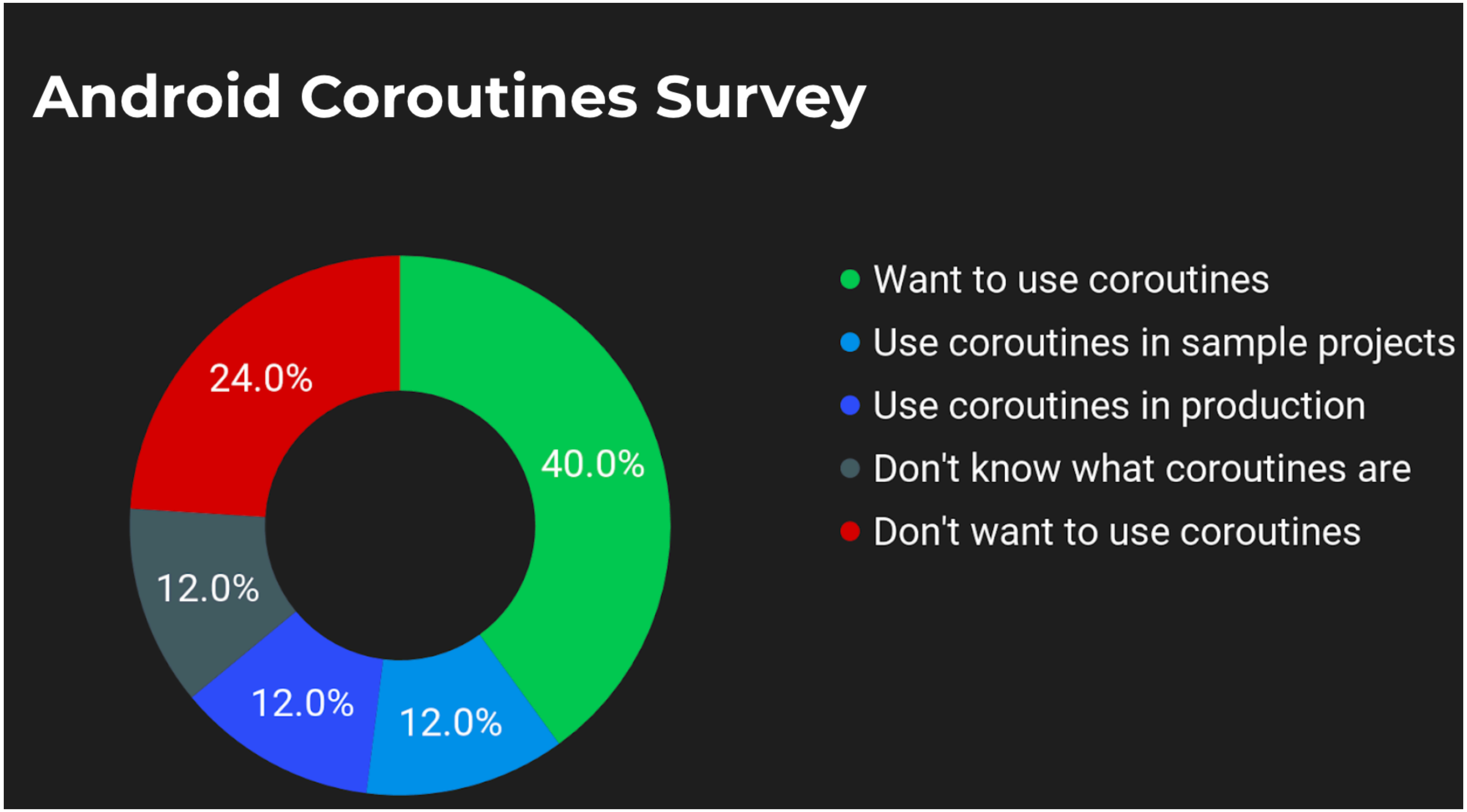

## runBlocking

```
fun <T> runBlocking(
    context: CoroutineContext = EmptyCoroutineContext,
    block: suspend CoroutineScope.() -> T
): T (source)
```

**Platform and version requirements:** JVM, Native

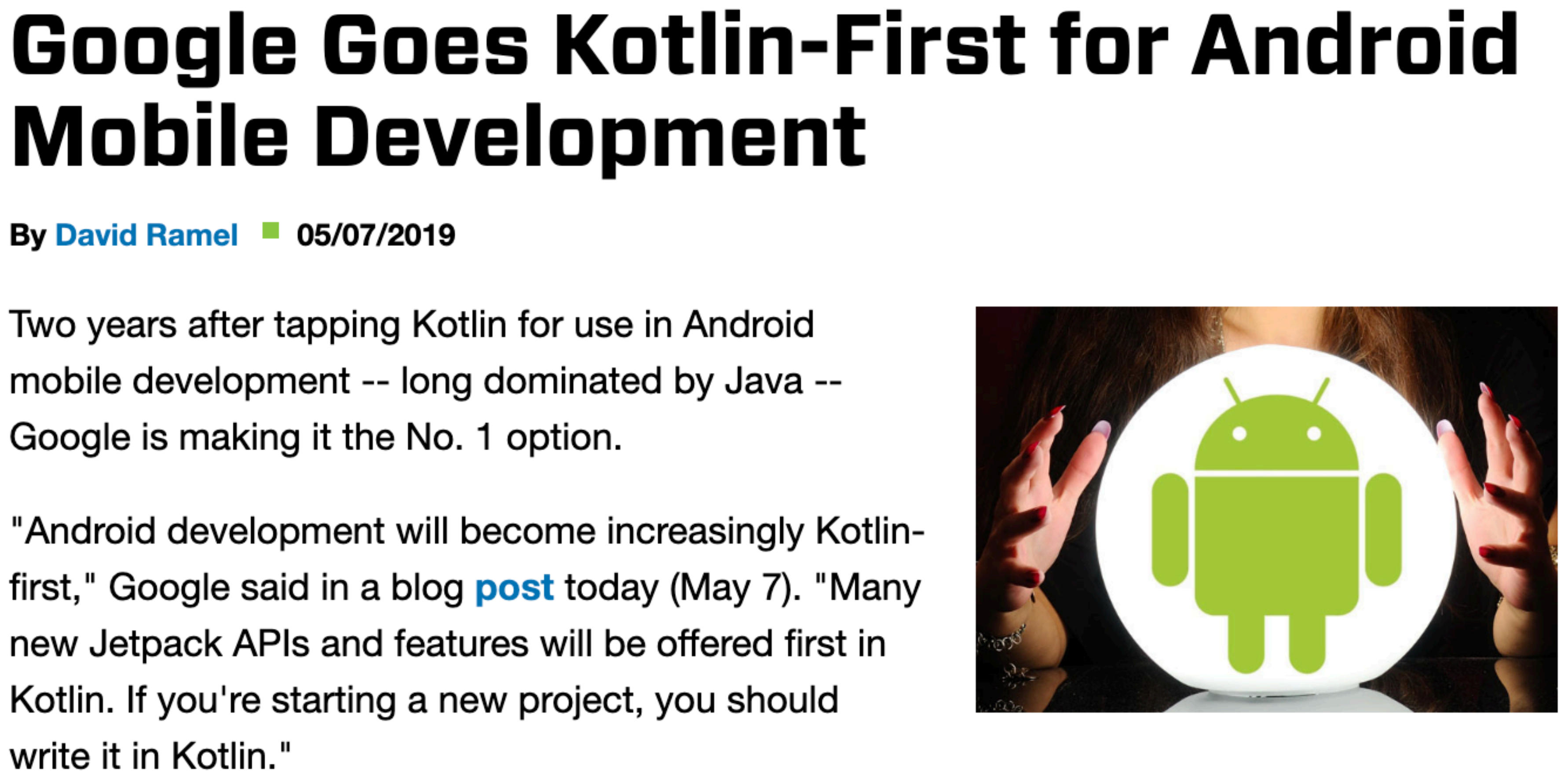
Runs a new coroutine and **blocks** the current thread *interruptibly* until its completion. This function should not be used from a coroutine. It is designed to bridge regular blocking code to libraries that are written in suspending style, to be used in `main` functions and in tests.

# Survey


Android Coroutines Survey
40.0%
12.0%
12.0%
12.0%
24.0%
● Want to use coroutines
● Use coroutines in sample projects
● Use coroutines in production
● Don't know what coroutines are
● Don't want to use coroutines

# What to do?

- Greenfield project? Probably try



**Google Goes Kotlin-First for Android Mobile Development**

**By David Ramel ▪ 05/07/2019**

Two years after tapping Kotlin for use in Android mobile development -- long dominated by Java -- Google is making it the No. 1 option.

"Android development will become increasingly Kotlin-first," Google said in a blog post today (May 7). "Many new Jetpack APIs and features will be offered first in Kotlin. If you're starting a new project, you should write it in Kotlin."

# What to do?

- Existing project?
  - Do they suit you?
  - Refactoring?
  - Time?
  - Small sections?

## Further resources

Coroutines Guide (https://kotlinlang.org/docs/reference/coroutines/coroutines-guide.html)

Coroutines (https://github.com/Kotlin/kotlinx.coroutines)

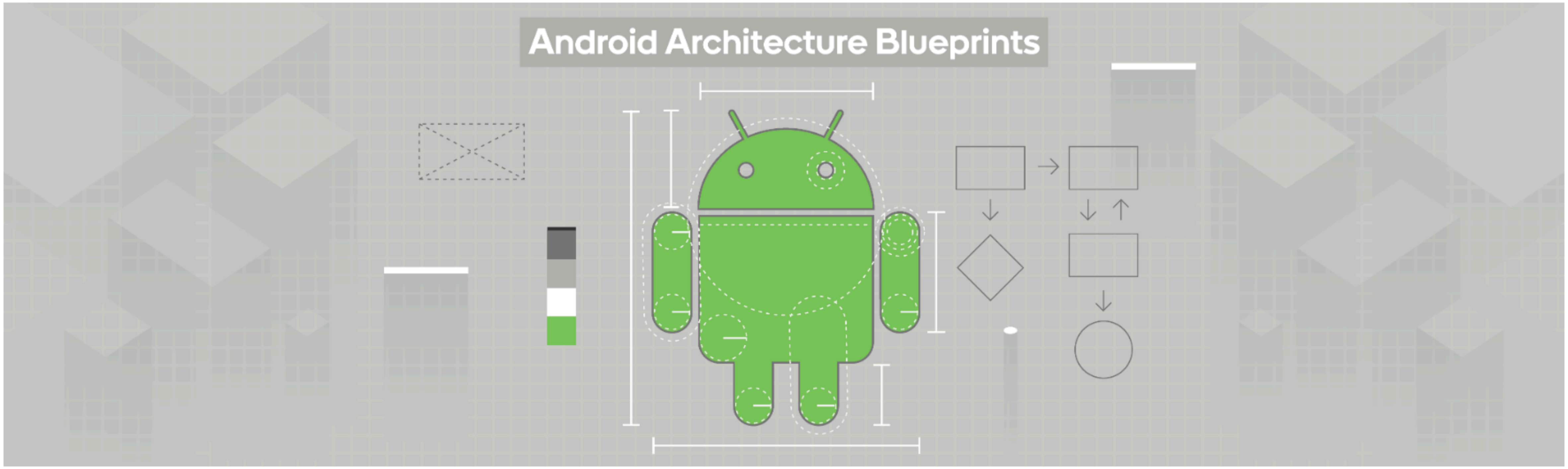
Kotlin Slack (kotlinlang.slack.com)

Kotlin Weekly (http://kotlinweekly.net)

Codelabs (https://codelabs.developers.google.com/codelabs/kotlin-coroutines/#0)

# Further resources

## Android Architecture Blueprints v2

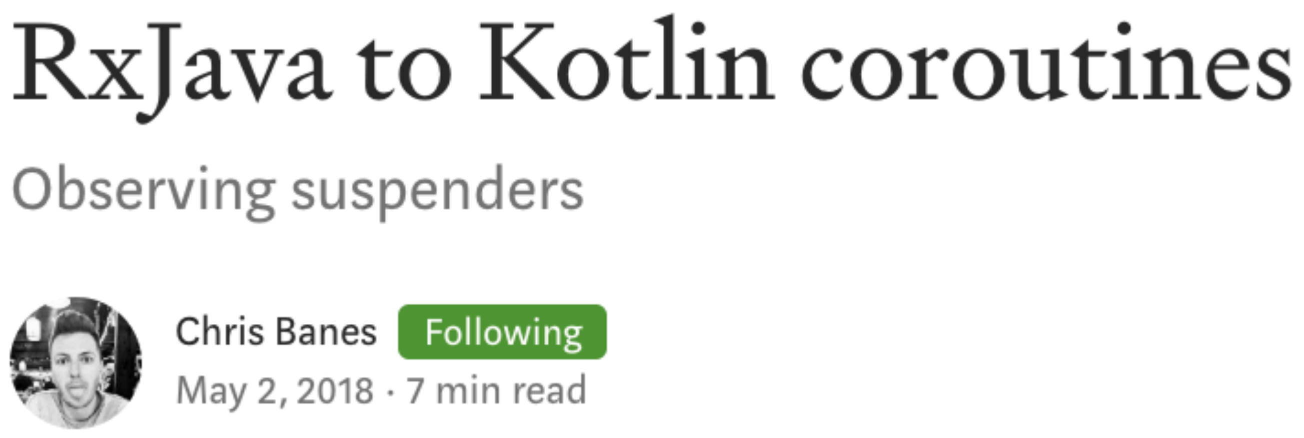


Android Architecture Blueprints is a project to showcase different architectural approaches to developing Android apps. In its different branches you'll find the same app (a TODO app) implemented with small differences.

In this branch you'll find:

- Kotlin Coroutines for background operations.
- A single-activity architecture, using the Navigation component to manage fragment operations.
- A presentation layer that contains a fragment (View) and a **ViewModel** per screen (or feature).

## Further resources



RxJava to Kotlin coroutines

Observing suspenders

Chris Banes Following

May 2, 2018 · 7 min read