



# RxJava & Coroutines

A Practical Analysis



@askashdavies





@askashdavies



@askashdavies



@askashdavies

# Adoption

7.7% (2015)

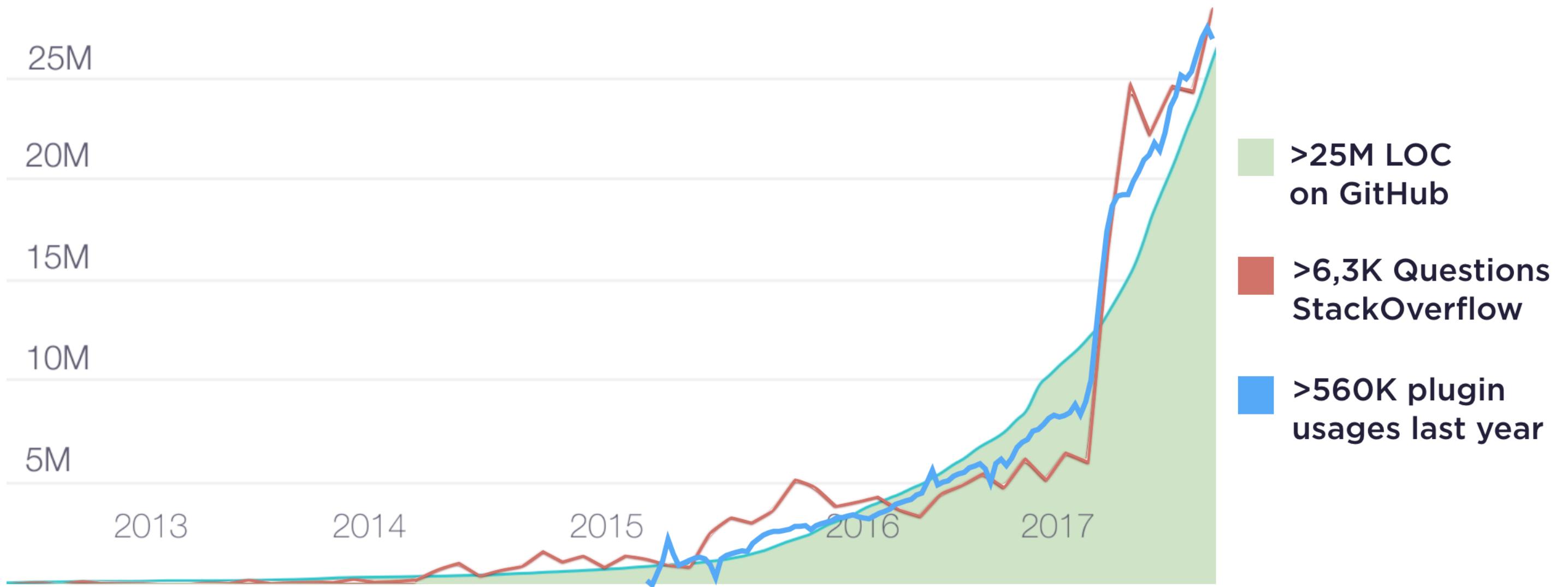
19.5% (2016)

7.7% (2015)

46.8% (2017)

19.5% (2016)

7.7% (2015)



# Multi Platform

@askashdavies





# Coroutines

```
fun main() {
    GlobalScope.launch {
        delay(1000L)
        println("World!")
    }
    println("Hello, ")
    Thread.sleep(2000L)
}
```

```
fun main() {  
    GlobalScope.launch {  
        delay(1000L)  
        println("World!")  
    }  
    println("Hello,")  
    Thread.sleep(2000L)  
}
```

```
fun main() {
    GlobalScope.launch {
        delay(1000L)
        println("World!")
    }
    println("Hello, ")
    Thread.sleep(2000L)
}
```



# Stability



@askashdavies

# async / launch

```
val deferred: Deferred<String> = async { "Hello" }
```

```
val result: String = deferred.await()
```

```
val job: Job = launch { "Hello" }
```

`job.join()`

# @Annotations

[bit.ly/2BrxgKv](https://bit.ly/2BrxgKv)



Here be dragons



@ExperimentalCoroutinesApi



@ObsoleteCoroutinesApi



@InternalCoroutinesApi



elizarov commented on 28 Sep

Collaborator

+ 😊 ...

From the point of usage `ObsoleteCoroutinesApi` is just like experimental. The difference is in intent. When API is experimental it might change or might graduate as is. For obsolete API we already know that there are problems with designs that will force us to deprecate this API in the future when we have a better replacement.

Right now there is no replacement for channel operations, so you have no choice but continue using them. However, we plan to replace them with mechanism based on lazy/cold streams ([#254](#)) in the future, which will get you much better performance.



1

@askashdavies



Bugger.

@askashdavies

# Coroutines



Best thing since sliced bread

# Coroutines

Best thing since ... brotchen?



# Coroutines



# Native first-party library



Easy-to-use



suspend fun

# Dispatchers.Main



# History of Android

# Background Processes

## Background Processes



# Runnable / Handler

# Background Processes



AsyncTask

# Background Processes



## IntentService

# Background Processes



Loader<T>

# Background Processes

 WorkManager

Background Processes

AlarmManager

Background Processes

AlarmManager, CountDownTimer

## Background Processes

AlarmManager, CountDownTimer ,  
FutureTask<T>

## Background Processes

AlarmManager, CountDownTimer ,  
FutureTask<T>, GcmNetworkManager

## Background Processes

AlarmManager, CountDownTimer ,  
FutureTask<T>, GcmNetworkManager,  
HandlerThread

## Background Processes

AlarmManager, CountDownTimer ,  
FutureTask<T>, GcmNetworkManager,  
HandlerThread, JobDispatcher

## Background Processes

AlarmManager, CountDownTimer ,  
FutureTask<T>, GcmNetworkManager,  
HandlerThread, JobDispatcher,  
JobScheduler

## Background Processes

AlarmManager, CountDownTimer ,  
FutureTask<T>, GcmNetworkManager,  
HandlerThread, JobDispatcher,  
JobScheduler,  
ScheduledThreadPoolExecutor

## Background Processes

AlarmManager, CountDownTimer ,  
FutureTask<T>, GcmNetworkManager,  
HandlerThread, JobDispatcher,  
JobScheduler,  
ScheduledThreadPoolExecutor, Timer

## Background Processes

AlarmManager, CountDownTimer ,  
FutureTask<T>, GcmNetworkManager,  
HandlerThread, JobDispatcher,  
JobScheduler,  
ScheduledThreadPoolExecutor, Timer,  
Task<T>



@askashdavies

# HOW STANDARDS PROLIFERATE:

(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)

SITUATION:  
THERE ARE  
14 COMPETING  
STANDARDS.

14?! RIDICULOUS!  
WE NEED TO DEVELOP  
ONE UNIVERSAL STANDARD  
THAT COVERS EVERYONE'S  
USE CASES.



SOON:

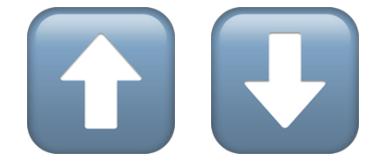
SITUATION:  
THERE ARE  
15 COMPETING  
STANDARDS.

RxJava to the rescue





# Chained operations



# Abstracted threading



# Reactive programming



@askashdavies

## RxFingerprint: Android Fingerprint Authentication and Encryption in RxJava2

## RxTuples2

RxTuples2 is a library to smooth RxJava2 usage by adding simple Tuple creation functions.

For the RxJava 1.X version, please go to [RxTuples](#).

RxFingerprint wraps the Android Fingerprint APIs (introduced in Android Marshmallow) and makes it easy to:

## RxMocks

[Apache License 2.0](#)

Mocks/Assertions for RxJava testing

## RxPermissions

[JitPack](#) 0.10.2 [nuget](#) v0.10.2 ▾ 1.2k [build](#) passing

This library allows the usage of RxJava with the new Android M permission model.

## RxLoader

[License](#) Apache 2.0 [API](#) 16+ [JCenter](#) 2.1.0

An Android Loader that wraps an RxJava Observable.

RxAndroidBle [build](#) passing [maven-central](#) v1.7.1

## RxAnimations

Download 0.9.1 API 15+

RxAnimations is a library with the main goal to make android animations more solid and cohesive.

## RxTuples2

RxTuples2 is a library to smooth RxJava2 usage by adding simple Tuple creation functions.

For the RxJava 1.X version, please go to [RxTuples](#).

Android Arsenal RxPaparazzo

RxJava extension for Android to take photos using the camera, select files or photos from the device and optionally crop or rotate any selected images.

## RxPaparazzo

## RxPermissions

JitPack 0.10.2 nuget v0.10.2 1.2k build passing

This library allows the usage of RxJava with the new Android M permission model.

Android Arsenal RxActivityResult

chat on gitter

## RxBinding

RxJava binding APIs for Android UI widgets from the platform and support libraries.

## RxActivityResult

## RxAndroidBle

build passing

maven-central v1.7.1

## RxFingerprint: Android Fingerprint Authentication and Encryption in RxJava2

RxFingerprint wraps the Android Fingerprint APIs (introduced in Android Marshmallow) and makes it easy to:

## Rx Preferences

Reactive SharedPreferences for Android.

downloads 245/month

Android Arsenal RxCache

## RxCache

中文文档

## RxLoader

For a more reactive approach go [here](#).

License Apache 2.0 API 16+ JCenter 2.1.0

An Android Loader that wraps an RxJava Observable.

## RxLifecycle

This library allows one to automatically complete sequences based on a second lifecycle stream.

This capability is useful in Android, where incomplete subscriptions can cause memory leaks.

## RxAnimations

[Download 0.9.1](#) [API 15+](#)

RxAnimations is a library with the main goal to make ar

**RxPresso** [build passing](#) [Download 0.2.0](#) [license unknown](#)

Easy Espresso UI testing for Android applications using RxJava.

## RxTuples2

RxTuples2 is a library to smooth RxJava2 usage by adding simple Tuple creation

For the RxJava 1.X version, please go to [RxTuples](#).

[Android Arsenal](#) [RxPaparazzo](#)

RxJava extension for Android to take photos using the camera, select files or photos from the device and optionally crop or rotate any selected images.

## RxCupboard RxPaparazzo

RxCupboard brings the excellent Android [Cupboard](#) library into the world of RxJava. Using Flowable s, you can fluently store and retrieve streams of POJOs from your database.

## RxPermissions

[JitPack 0.10.2](#) [nug](#)

## RxBonjour

This library allows t

[build failing](#)

A reactive wrapper around network service discovery functionalities for Kotlin and Java.

## RxBinding

RxJava binding APIs for Android UI widgets from the platform and support libraries.

**RxAndroidBle** [build passing](#) [maven-central v1.7.1](#)

## RxFingerprint: Android Fingerprint Authentication and Encryption in RxJava2

nakes it easy to:

## Reactive Wrapper for Java8 WatchService

RxFileWatcher allows you to observe directories (recursively or not) for file system events with a RxJava observable based on the JDK WatchService, but it is much more convenient.

## Rx Preferences

Reactive SharedPreferences for Android.

**RxMocks** [Apache License 2.0](#)

ocks/Assertions fo

## RxFileObserver

[build passing](#) [codecov](#) [unknown](#) [Download 2.0.1](#) [Android Arsenal](#) [RxFit](#) [API 9+](#)

[downloads 245/month](#)

[Android Arsenal](#) [RxFileObserver](#)

Reactive wrapper around Android's [FileObserver](#)

[Android Arsenal](#) [RxCache](#)

## RxCache

[中文文档](#)

## RxLoader

[For a more reactive approach go here.](#)

[License Apache 2.0](#) [API 16+](#) [JCenter 2.1.0](#)

An Android Loader that wraps an RxJava Observable.

## RxLifecycle

This library allows one to automatically complete sequences based on a second lifecycle stream.

This capability is useful in Android, where incomplete subscriptions can cause memory leaks.

## RxAnimations

Download 0.9.1 API 15+

RxAnimations is a library with the main goal to make android anim

RxPresso build passing Download 0.2.0 license unknown

Easy Espresso UI testing for Android applications using RxJava.

RxFingerprint: Android Fingerprint Authentication and Encryption in RxJava2

## RxTuples2

RxTuples2 is a library to smooth RxJava2 usage by adding simple Tuple creation function

For the RxJava 1.X version, please go to [RxTuples](#).

Android Arsenal RxPaparazzo

RxJava extension for Android to take photos using the camera, select files or photos from the device and optionally crop or rotate any selected images.

## RxPaparazzo

## RxPermissions

JitPack 0.10.2 nuget v0.10.2

## RxBonjour

This library allows the usage

## Reactive Wearable API Library for Android

build passing codecov unknown Download 1.3.0 Android Arsenal RxWear API 9+

build failing

## RxMocks

Apache License 2.0

ocks/Assertions fo

## RxFileObserver

build passing codecov unknown Download 2.0.1 Android Arsenal RxFit API 9+

downloads 245/month

Android Arsenal RxFileObserver

Reactive wrapper around Android's FileObserver

## RxCache

中文文档

For a more reactive approach go [here](#).

## RxBinding

RxJava binding APIs for Android UI widgets from the platform and support libraries.

## RxLoader

License Apache 2.0 API 16+ JCenter 2.1.0

an RxJava Observable.

## RxActivityResult

Android RxFirebase

build error coverage unknown bintray invalid license apache 2.0

## RxRelay

Relays are RxJava types which are both an Observable and a Consumer.

Basically: A Subject except without

## RxLifecycle

RxJava wrapper for use with the Android Firebase client

## Android RxTasks

build passing coverage unknown Download 2.0.0 license apache 2.0

library allows one to automatically complete sequences based on a second lifecycle stream.

capability is useful in Android, where incomplete subscriptions can cause memory leaks.

@askashdavies



@askashdavies

build passing

@askashdavies



# Asynchronous API's



# Synchronous API's

`Observable.fromIterable()`



Flowable / Observable

Single / Completable / Maybe

	To	Flowable	Observable	Maybe	Single	Completable
From						
Flowable		toObservable()		reduce() elementAt() firstElement() lastElement() singleElement()	scan() elementAt() first()/firstOrDefault() last()/lastOrDefault() single/singleOrDefault() all()/any()/count() (and more)	ignoreElements()
Observable	toFlowable()			reduce() elementAt() firstElement() lastElement() singleElement()	scan() elementAt() first()/firstOrDefault() last()/lastOrDefault() single/singleOrDefault() all()/any()/count() (and more)	ignoreElements()
Maybe	toFlowable()	toObservable()			toSingle() sequenceEqual()	toCompletable()
Single	toFlowable()	toObservable()	toMaybe()			toCompletable()
Completable	toFlowable()	toObservable()	toMaybe()		toSingle() toSingleDefault()	



```
Observable
    .fromIterable(resourceDraft.getResources())
    .flatMap(resourceServiceApiClient::createUploadContainer)
    .zipWith(Observable.fromIterable(resourceDraft.getResources()), Pair::create)
    .flatMap(uploadResources())
    .toList()
    .toObservable()
    .flatMapMaybe(resourceCache.getResourceCachedItem())
    .defaultIfEmpty(Resource.getDefaultItem())
    .flatMap(postResource(resourceId, resourceDraft.getText(), currentUser, resourceDraft.getIntent()))
    .observeOn(AndroidSchedulers.mainThread())
    .subscribeOn(Schedulers.io())
    .subscribe(
        resource -> repository.setResource(resourceId, resource, provisionalResourceId),
        resourceUploadError(resourceId, resourceDraft, provisionalResourceId)
    );
}
```



@askashdavies



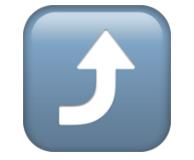
# Hidden complexity



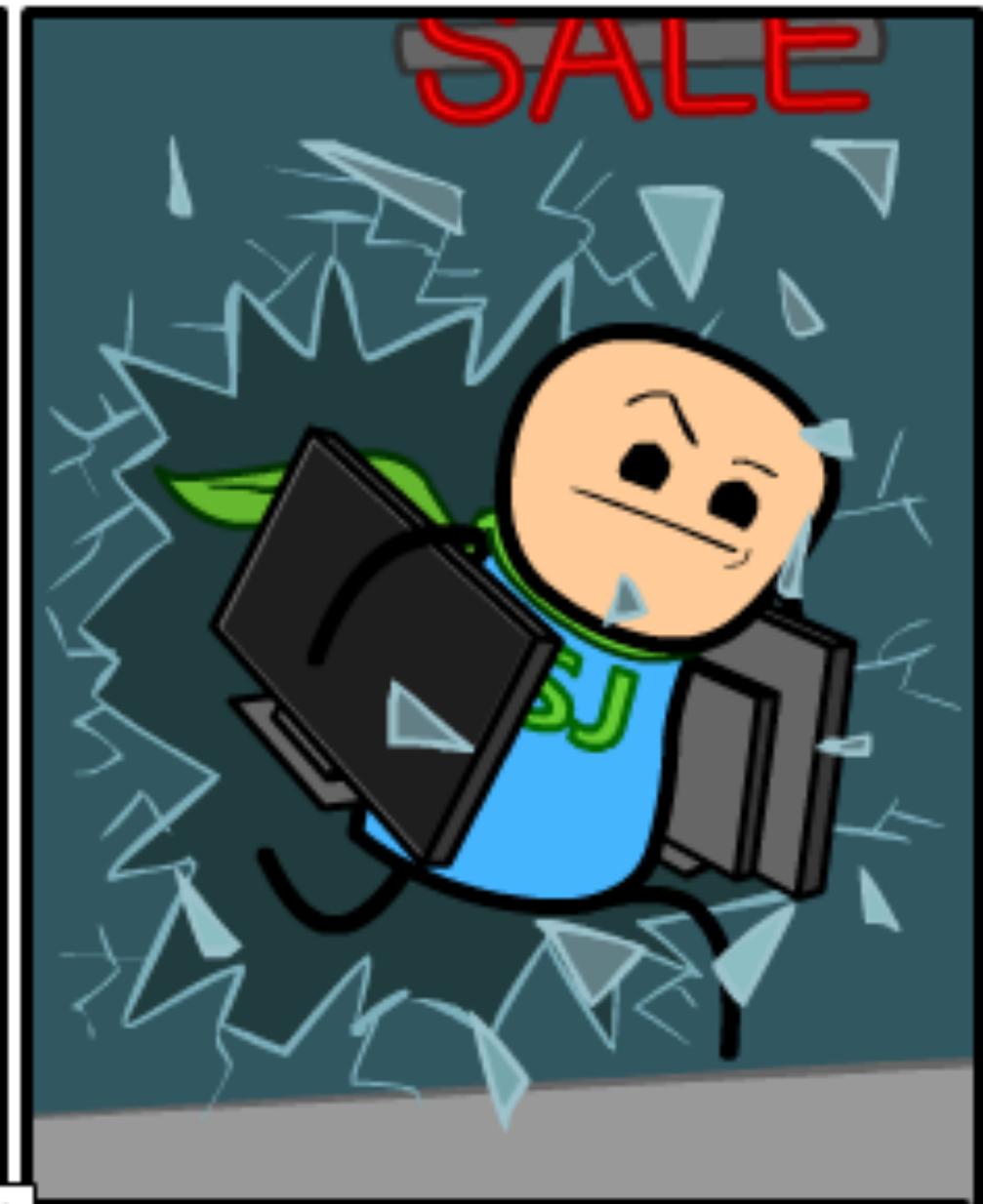
# Hidden gotchas



# Memory footprint



Steep learning curve



Cyanide and Happiness © Explosm.net



@askashdavies

# #RxMustDie

[pca.st/7IJG](http://pca.st/7IJG)

@askashdavies



"If all you have is a hammer,  
everything looks like a nail"

— Abraham Maslow, *The Psychology of Science*, 1966

# Reactive / Imperative

# Coroutine Use Cases

# Network Call Handling

```
// RxJava2  
fun getUser(): Single<User> = Single.fromCallable { /* ... */ }  
  
// Coroutine  
suspend fun getUser(): User = /* ... */
```

# Cache Retrieval

```
// RxJava2  
fun getUser(): Maybe<User> = Maybe.fromCallable { /* ... */ }  
  
// Coroutine  
suspend fun getUser(): User? = /* ... */
```

# Background Task Handling

```
// RxJava2  
fun storeUser(user: User): Completable.fromCallable { /* ... */ }  
  
// Coroutine  
suspend fun storeUser(user: User) { /* ... */ }
```

# RxJava2 / Coroutines

Single<T> / T

Maybe<T> / T?

Completable / Unit

# Thread Handling

```
// RxJava2  
getUser()  
.subscribeOn(Schedulers.computation())  
.observeOn(AndroidSchedulers.mainThread())  
.subscribe { /* Do something */ }
```

```
// Coroutine  
launch(Dispatchers.Main) {  
    val user = getUser()  
    /* Do something */  
}
```

# 💪 FlatMap

```
// RxJava2
getUser()
    .flatMap { doSomethingWithUser(it) }
    .subscribeOn(Schedulers.computation())
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe { /* Do something else */ }
```

```
// Coroutine
launch(Dispatchers.Main) {
    val user = getUser()
    val smth = doSomethingWithUser(user)

    /* Do something else */
}
```

# Callback Consumption

```
// RxJava2
fun getSingle(): Single = Single.create<T> { emitter ->
    doSomethingAsync(object: Callback<T> {
        override fun onComplete(result: T) = emitter.onSuccess(result)
        override fun onException(exception: Exception) = emitter.onError(exception)
    })
}

// Coroutine
suspend fun getCoroutine() : T = suspendCoroutine { continuation ->
    doSomethingAsync(object : Callback<T> {
        override fun onComplete(result: T) = continuation.resume(result)
        override fun onException(exception: Exception) = continuation.resumeWithException(exception)
    })
}
```



# Continuation

# Task Cancellation

```
// RxJava2  
val disposable: Disposable = Single  
    .create { /* Do something */ }  
    .subscribe { /* Do something else */ }  
disposable.dispose()
```

```
// Coroutine  
val parent: Job = Job()  
launch(Dispatchers.Main + parent) { /* Do something */ }  
parent.cancelChildren()
```

# ViewModel.viewModelScope

androidx.lifecycle:2.1.0-alpha01

[bit.ly/2FZ0FPN](https://bit.ly/2FZ0FPN)

# Lifecycle Task Cancellation

```
class MainViewModel : ViewModel {  
  
    fun init() {  
        viewModelScope.launch {  
            someSuspendFunction()  
        }  
    }  
}  
}
```

# Value Streams

```
// RxJava2  
val publisher = PublishSubject()  
publisher.subscribe { /* Do something */ }  
publisher.onNext("Hello")
```

```
// Coroutine  
val channel = Channel<String>()  
launch { channel.send("Hello") }  
launch { channel.consumeEach { /* Do something */ } }
```

 Backpressure

# 💪 Backpressure

```
launch { channel.send("Hello") }
```

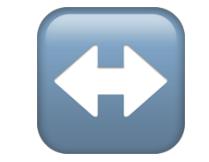


RxJava

~~Channels~~

[bit.ly/2DQU7lb](https://bit.ly/2DQU7lb)

@askashdavies



# Transformations



# Cold Streams



# Complex Business Logic

# Asynchronicity Comparison

Manuel Vivo (@manuelvicnt)

[bit.ly/2R27stP](https://bit.ly/2R27stP)

# Is Coroutines a replacement for RxJava?

Maybe...

# Should I migrate to Coroutines?

Probably not...

"Don't fix what ain't broke"

— *Some Guy*



© 2019 Viacom International Inc.

# Did you migrate to Coroutines?



Yes!

# How could I migrate to Coroutines?



# Migration Policy

# Retrofit Services

# Retrofit2 Coroutines Adapter [bit.ly/2TyGXOh](https://bit.ly/2TyGXOh)

com.jakewharton.retrofit:retrofit2-kotlin-coroutines-adapter:+

```
interface UserService {  
  
    @GET("/user")  
    fun getUser(): Deferred<User>  
}  
  
val retrofit = Retrofit.Builder()  
    .baseUrl("https://example.com/")  
    .addCallAdapterFactory(CoroutineCallAdapterFactory())  
    .build()  
  
GlobalScope.launch {  
    val user = retrofit  
        .create<UserService>() // >= 2.5.0  
        .getUser()  
        .await()  
}
```

@askashdavies

```
interface UserService {  
  
    @GET("/user")  
    fun getUser(): Deferred<User>  
}  
  
val retrofit = Retrofit.Builder()  
    .baseUrl("https://example.com/")  
    .addCallAdapterFactory(CoroutineCallAdapterFactory())  
    .build()  
  
GlobalScope.launch {  
    val user = retrofit  
        .create<UserService>() // >= 2.5.0  
        .getUser()  
        .await()  
}
```

@askashdavies

```
interface UserService {  
  
    @GET("/user")  
    fun getUser(): Deferred<User>  
}  
  
val retrofit = Retrofit.Builder()  
    .baseUrl("https://example.com/")  
    .addCallAdapterFactory(CoroutineCallAdapterFactory())  
    .build()  
  
GlobalScope.launch {  
    val user = retrofit  
        .create<UserService>() // >= 2.5.0  
        .getUser()  
        .await()  
}
```

@askashdavies

```
interface UserService {  
  
    @GET("/user")  
    fun getUser(): Deferred<User>  
}  
  
val retrofit = Retrofit.Builder()  
    .baseUrl("https://example.com/")  
    .addCallAdapterFactory(CoroutineCallAdapterFactory())  
    .build()  
  
GlobalScope.launch {  
    val user = retrofit  
        .create<UserService>() // >= 2.5.0  
        .getUser()  
        .await()  
}
```

@askashdavies

Retrofit [bit.ly/2CU0LUK](https://bit.ly/2CU0LUK)

First-party "suspend" support

Coroutines RxJava2 [bit.ly/2DQ2ZYn](https://bit.ly/2DQ2ZYn)

org.jetbrains.kotlinx:kotlinx-coroutines-rx2:+

Name	Scope	Description
rxCompletable	CoroutineScope	Cold completable that starts coroutine on subscribe
rxMaybe	CoroutineScope	Cold maybe that starts coroutine on subscribe
rxSingle	CoroutineScope	Cold single that starts coroutine on subscribe
rxObservable	ProducerScope	Cold observable that starts coroutine on subscribe
rxFlowable	ProducerScope	Cold observable that starts coroutine on subscribe with <b>backpressure</b> support

```
val service: UserService = /* ... */  
  
GlobalScope.rxSingle { service.getUser() }  
.observeOn(AndroidSchedulers.mainThread())  
.subscribeOn(Schedulers.io())  
.subscribe(  
    { /* Do something with user */ },  
    { /* Handle error ... maybe */ }  
)
```

# Completable.await()

```
GlobalScope.launch {  
    Completable  
        .complete()  
        .await()  
}
```

# Maybe.await()

```
GlobalScope.launch {  
    val result: String? = Maybe  
        .fromCallable { null as String? }  
        .await()  
}
```

# Observable.await... [bit.ly/2Tl37TK](https://bit.ly/2Tl37TK)

```
GlobalScope.launch {  
    val observable = Observable.just(1, 2, 3, 4)  
  
    // Await first item  
    val item = observable.awaitFirst()  
  
    // Print each item  
    observable.consumeEach {  
        println(it)  
    }  
  
    // Consume all items  
    observable  
        .openSubscription()  
        .consume {  
            println(it.size)  
        }  
}
```



# Conclusion



THE END



Cheers! 🍻



@askashdavies





**ITS OVER. GO HOME!**