twitch.tv/dancojocar

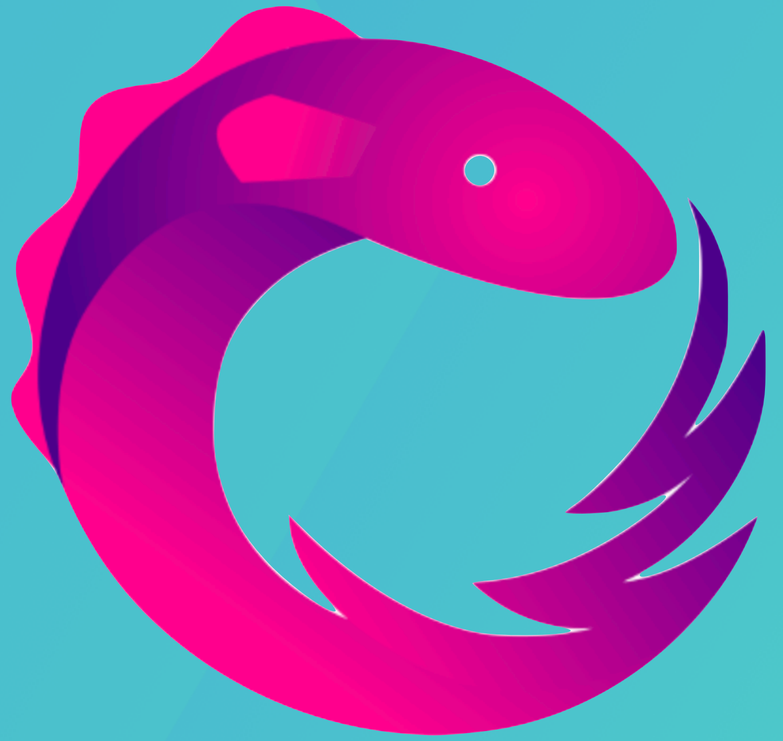
youtube.com/dancojocar

Lecture #6
Coroutines & Reactive Programming

Mobile Applications 2020-2021

Why Reactive?

- Unless you can model your entire system synchronously, a single asynchronous source breaks imperative programming.



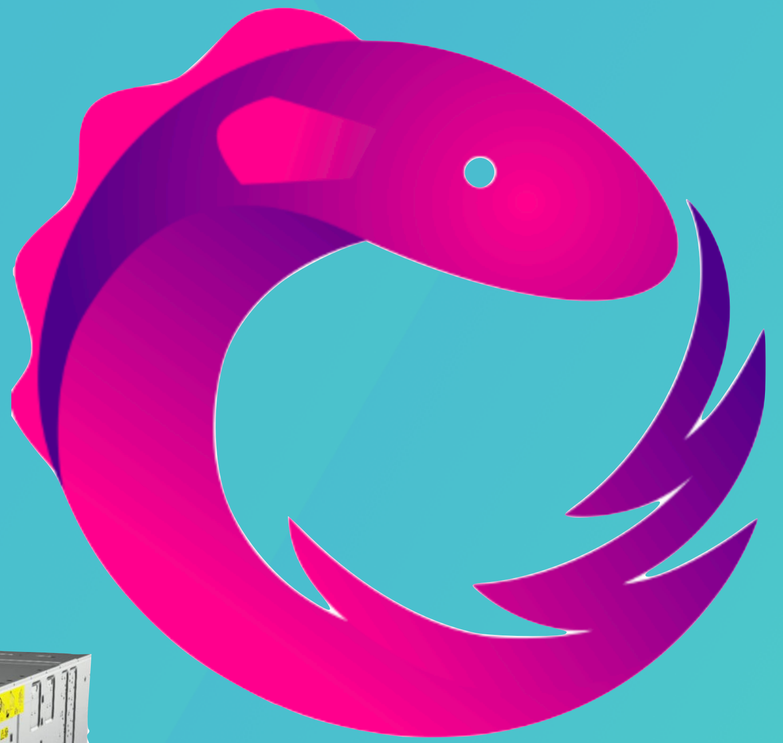
twitch.tv/dancojocar

youtube.com/dancojocar

Why Reactive?

```
val um: UserManager = UserManagerImpl()
logd(um.getUser())

interface UserManager {
    fun getUser(): User
    fun setName(name: String)
    fun setAge(age: Int)
}
```

```
interface UserManager {
    fun getUser(): User
}
um.setName("John Doe")
logd(um.getUser())
```



twitch.tv/dancojocar

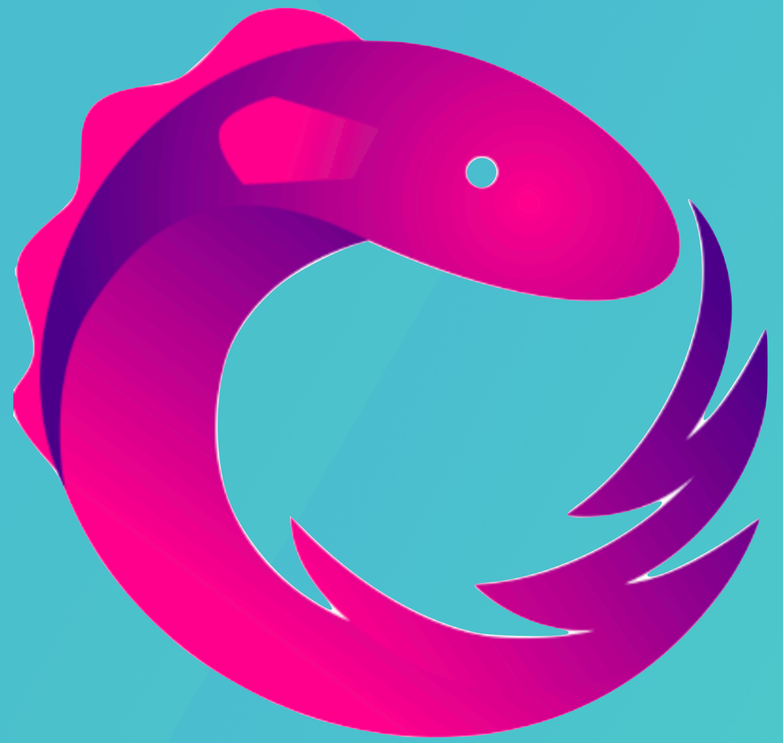
youtube.com/dancojocar

Why Reactive?

```
interface UserManager {
    fun getUser(): User
    fun setName(name: String)
    fun setAge(age: Int)
}

um.setName("John Doe")
logd(um.getUser())
```

async

async

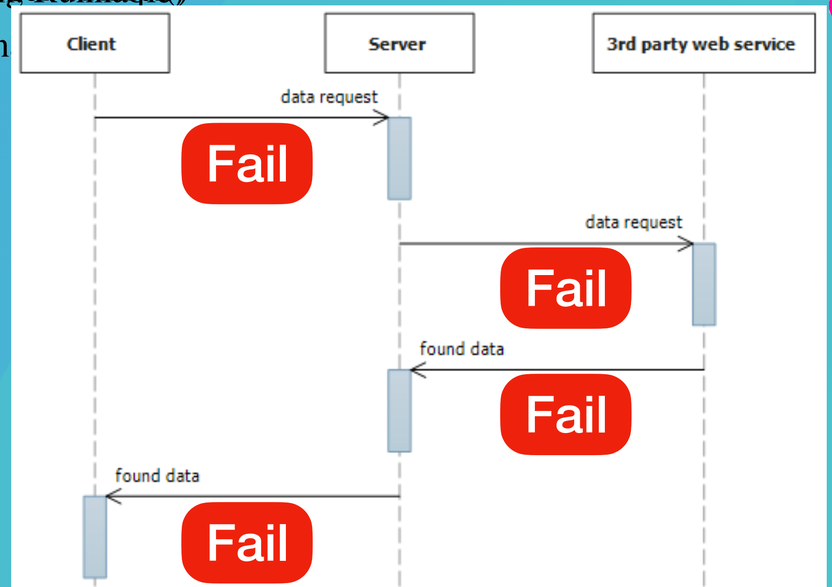


twitch.tv/dancojocar

youtube.com/dancojocar

Why Reactive?

```
val um: UserManagerV2 = UserManagerV2Impl()
interface UserManagerV2 {
    fun getUser(): User
    fun setName(name: String, callback: Runnable)
    fun setAge(age: Int, callback: Runn
}
```



twitch.tv/dancojocar

youtube.com/dancojocar

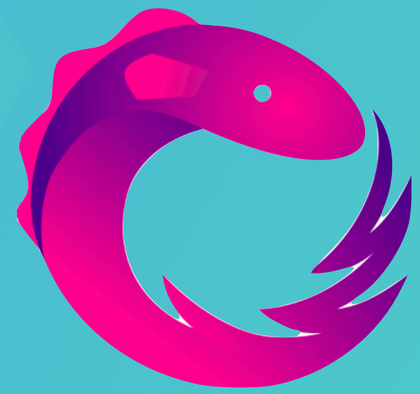
Why Reactive?

```
um.setName("John Doe", object : UserManagerV3.Listener {
    override fun success(user: User) {
        um.setAge(42, object : UserManagerV3.Listener {
            override fun success(user: User) {
                logd(user)
            }
            override fun failed(error: UserException) {
                loge("Unable to update the user details", error)
            }
        })
    }
    override fun failed(error: UserException) {
        loge("Unable to update the user details", error)
    }
})
```

```
interface UserManagerV3 {
    fun getUser(): User
    fun setName(name: String, listener: Listener): void
    fun setAge(age: Int, listener: Listener): void

    interface Listener {
        fun success(user: User)
        fun failed(error: UserException)
    }
}
```

```
val um: UserManagerV3 = UserManagerV3Impl()
logd(um.getUser())
```

```
um.setName("John Doe", object : UserManagerV3.Listener {
    override fun success(user: User) {
        logd(user)
    }
    override fun failed(error: UserException) {
        loge("Unable to update the user details", error)
    }
})
um.setAge(42, object : UserManagerV3.Listener {
    override fun success(user: User) {
        logd(user)
    }
    override fun failed(error: UserException) {
        loge("Unable to update the user details", error)
    }
})
```

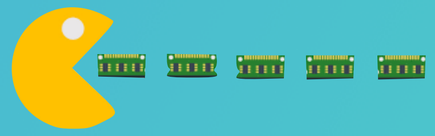


Why Reactive?

twitch.tv/dancojocar

youtube.com/dancojocar

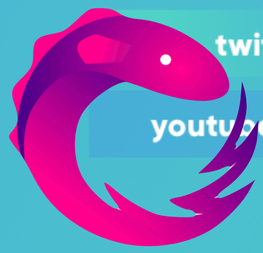
```
class UserActivity : AppCompatActivity() {
    val um: UserManager = UserManagerImpl()
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_user)
        um.setName("John Doe", object : UserManager.Listener {
            override fun success(user: User) {
                um.setAge(42, object : UserManager.Listener {
                    override fun success(user: User) {
                        loge("Unable to update the user details", error)
                    }
                })
            }

            override fun failed(error: UserException) {
                loge("Unable to update the user details", error)
            }
        })
    }
}
```



“A small leak will sink a great ship.”
Benjamin Franklin

https://github.com/square/leakcanary

Why Reactive?



twitch.tv/dancojocar

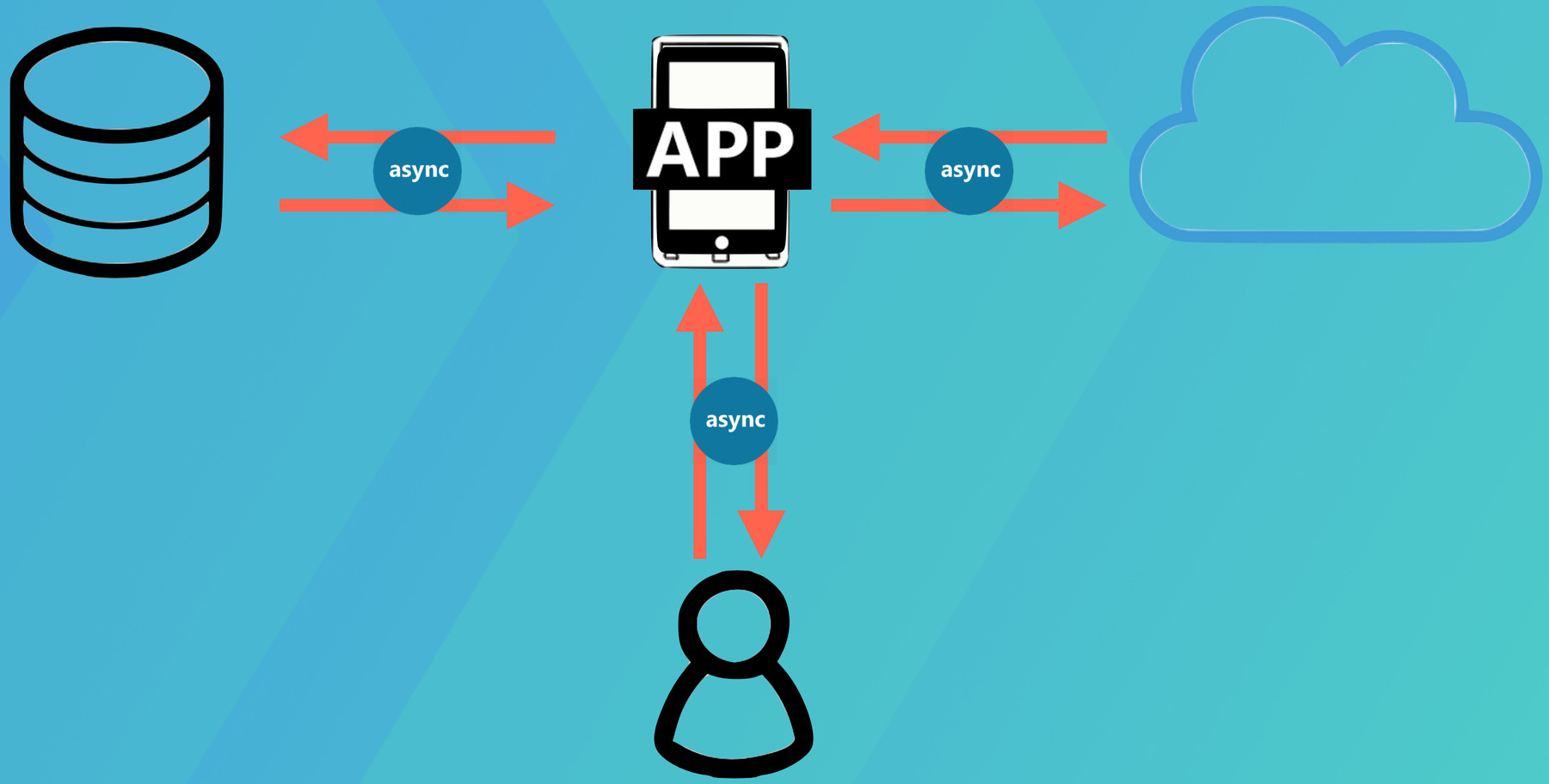
youtube.com/dancojocar

```
class UserActivity : AppCompatActivity() {
    val um: UserManager = UserManagerImpl()
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_user)
        um.setName("John Doe", object : UserManager.Listener {
            override fun success(user: User) {
                um.setAge(42, object : UserManager.Listener {
                    override fun success(user: User) {
                        runOnUiThread {
                            if (!isDestroyed) {
                                if (!isFinishing) {
                                    textView.text = user.name
                                }
                                logd(user)
                            }
                        }
                    }
                    override fun failed(error: UserException) {
                        loge("Unable to update the user details", error)
                    }
                })
            }
        })
        override fun failed(error: UserException) {
        //..loge("Unable to update the user details", error)
        })
    }
})
}
```

twitch.tv/dancojocar

youtube.com/dancojocar

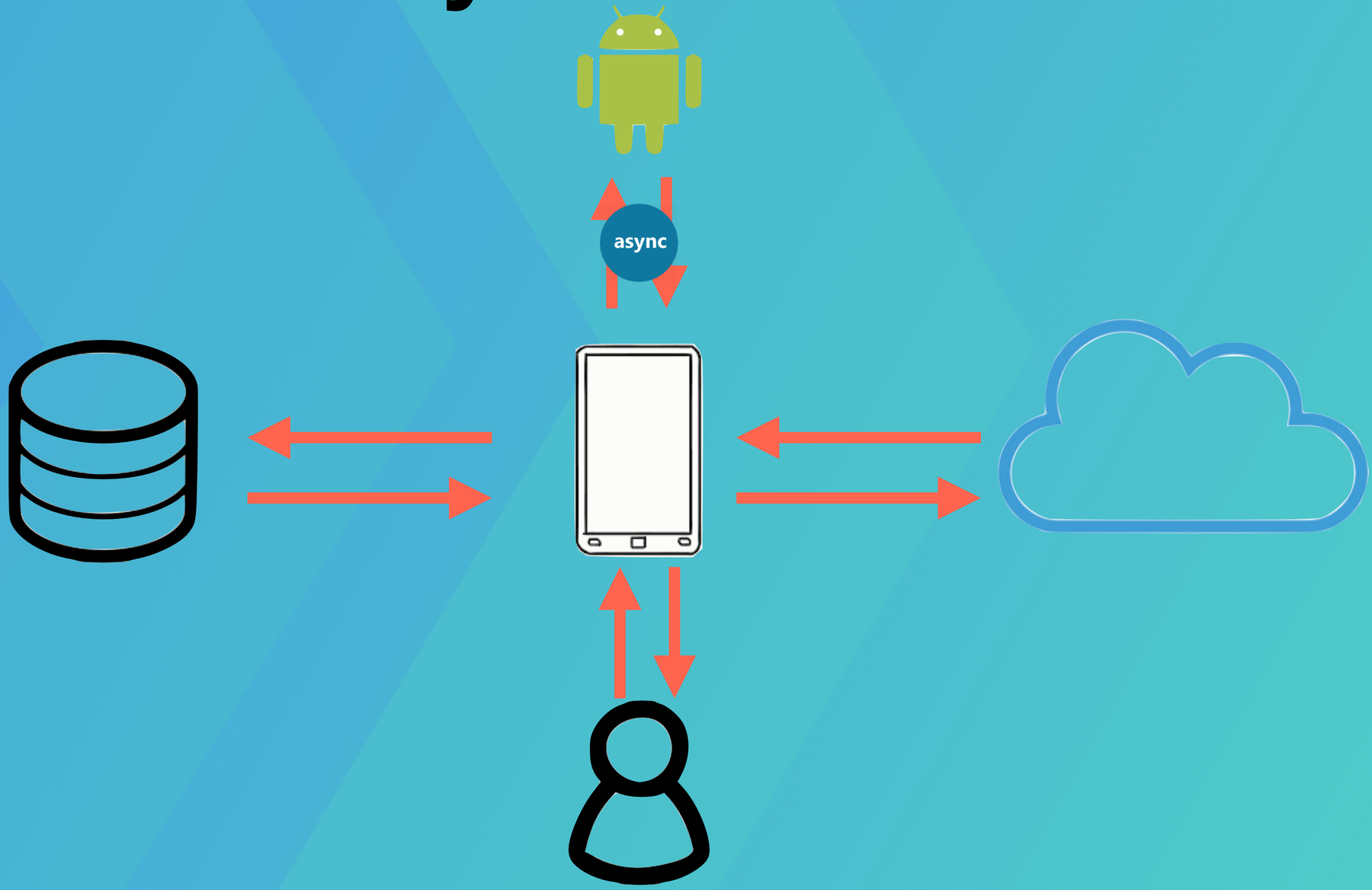
Why Reactive?



twitch.tv/dancojocar

youtube.com/dancojocar

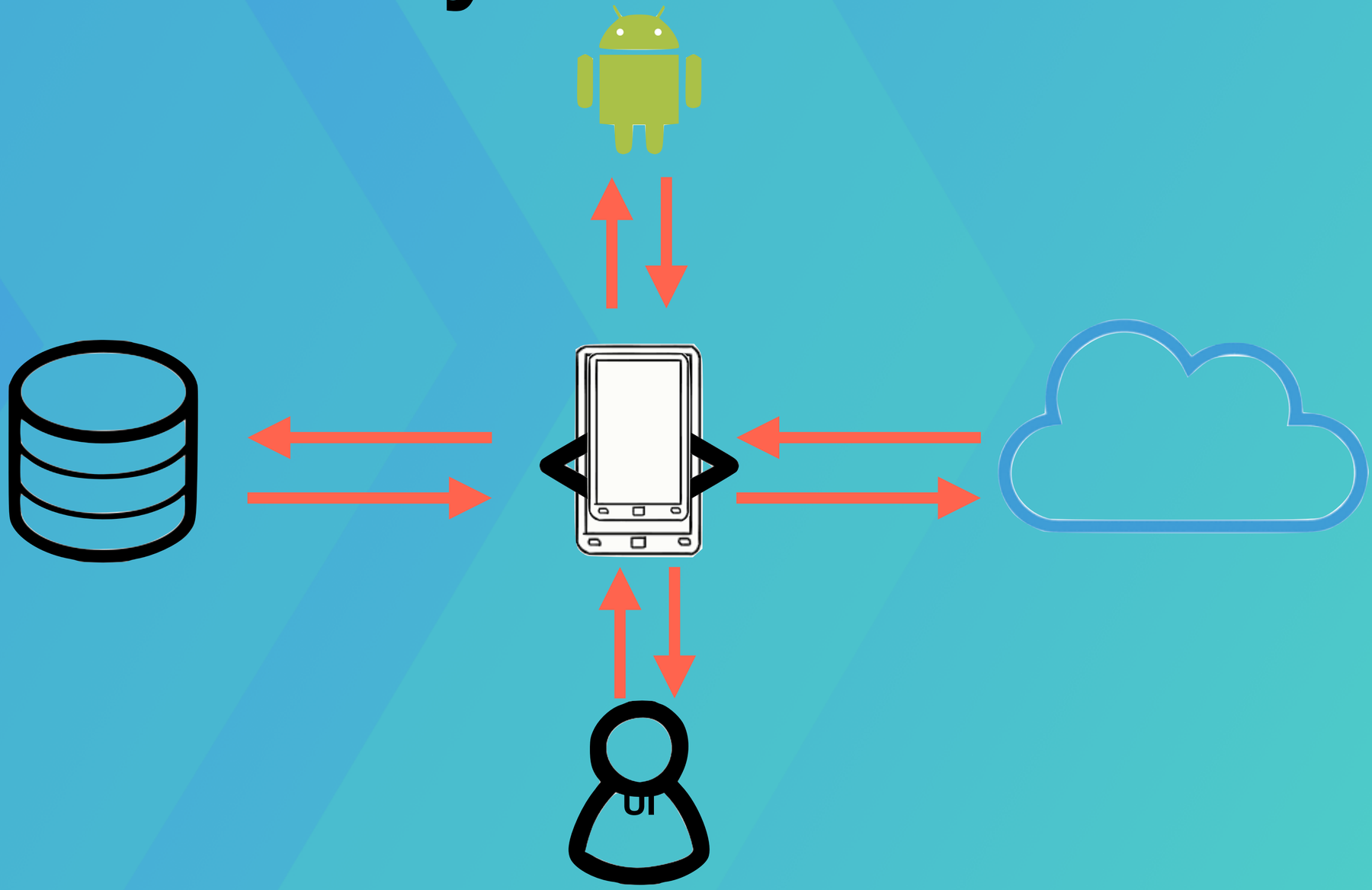
Why Reactive?



twitch.tv/dancojocar

youtube.com/dancojocar

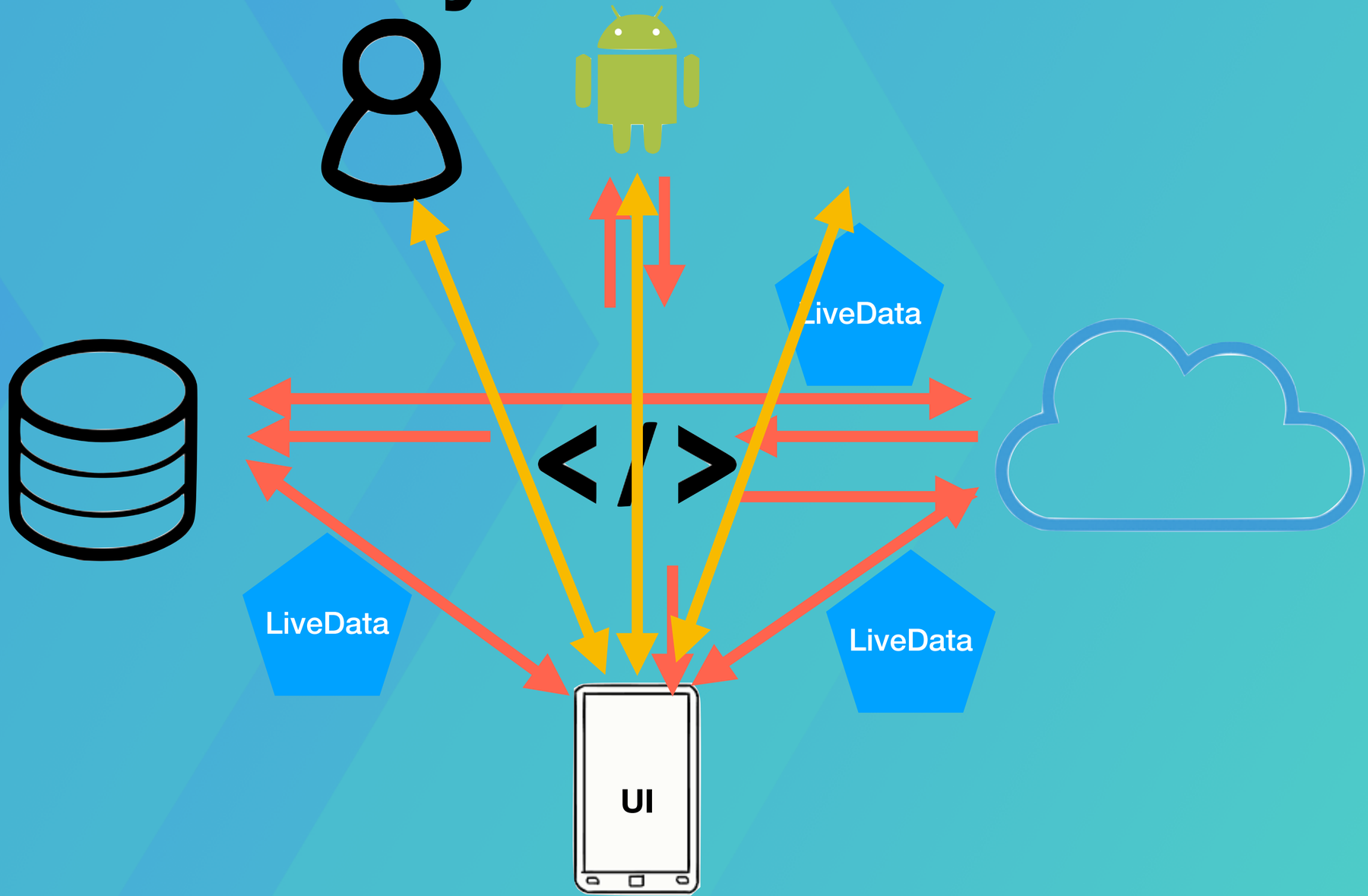
Why Reactive?



twitch.tv/dancojocar

youtube.com/dancojocar

Why Reactive?



Rx{Java|Kotlin|Swift|Dart}

- A set of classes for representing sources of data.
- A set of classes for listening to data sources.
- A set of methods for modifying and composing the data.

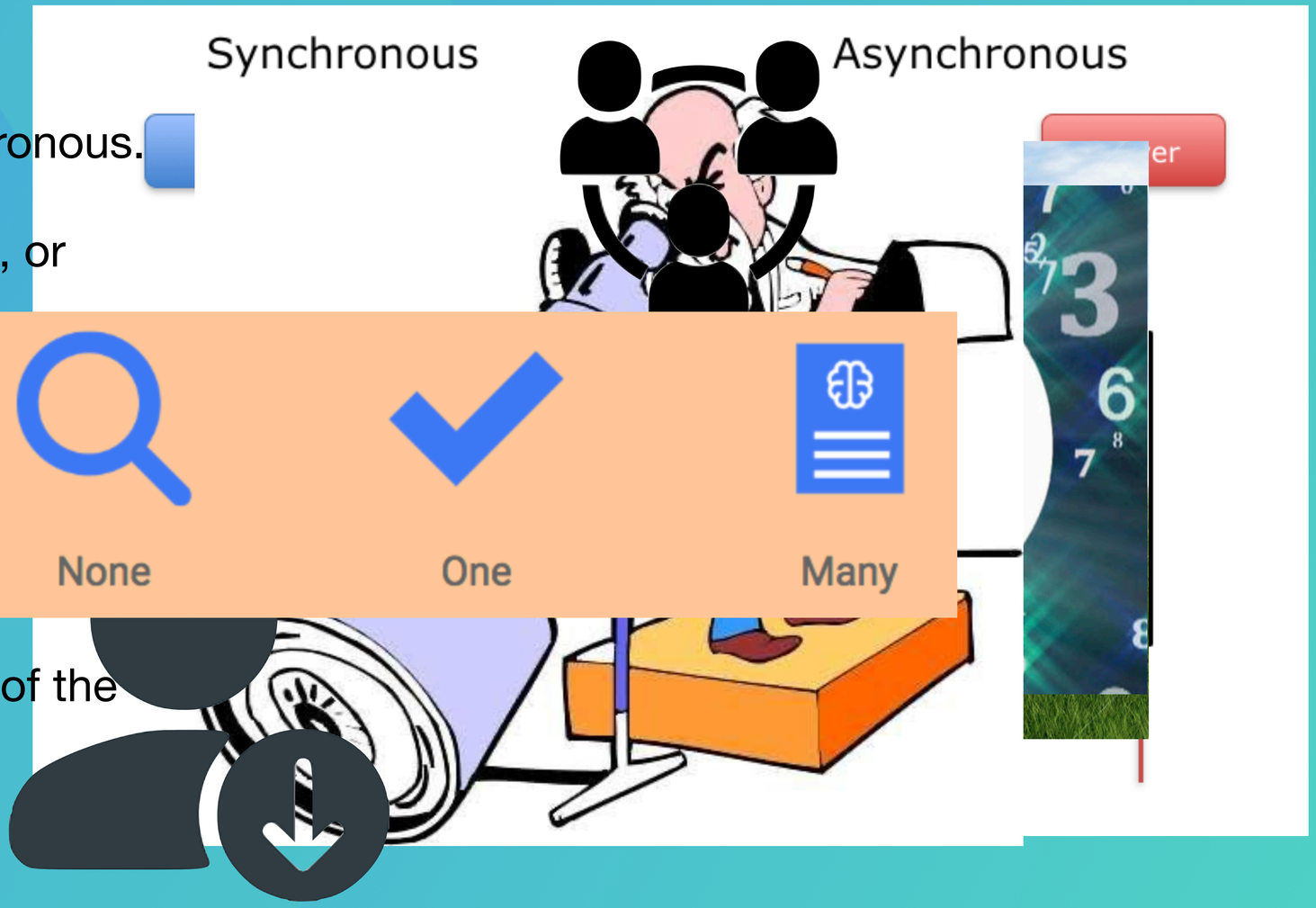
http://reactivex.io/

twitch.tv/dancojocar

youtube.com/dancojocar

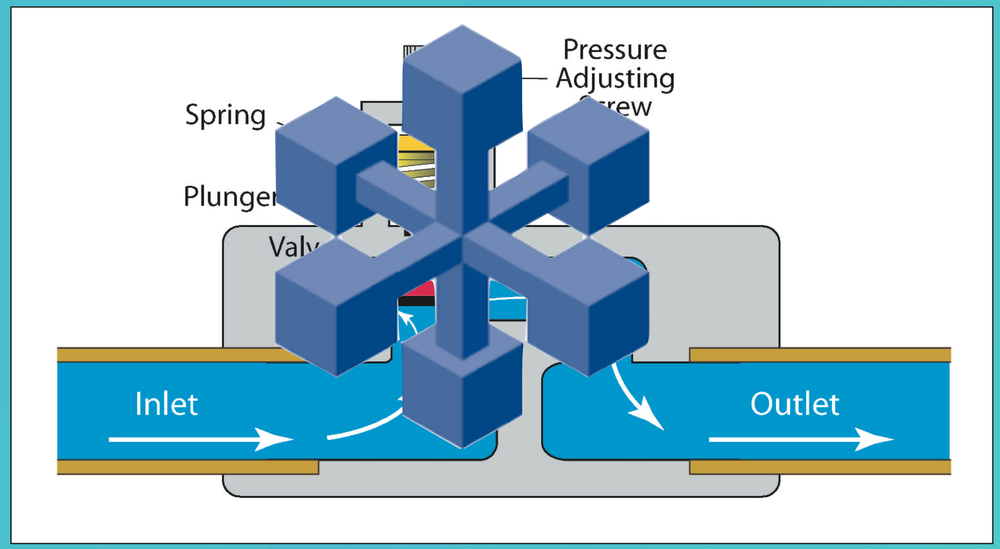
Sources

- Synchronous or asynchronous.
- Single item, many items, or empty.
- Terminates with an erro succeeds to completio
- May never terminate!
- Just an implementation of the Observer pattern.



Sources

- Observable<T>
- Observable<T>
  - Emits 0 to N items.
  - Terminates with complete or error.
  - Does not have backpressure.
- Flowable<T>
- Flowable<T>
  - Emits 0 to N items.
  - Terminates with complete or error.
  - Has backpressure.

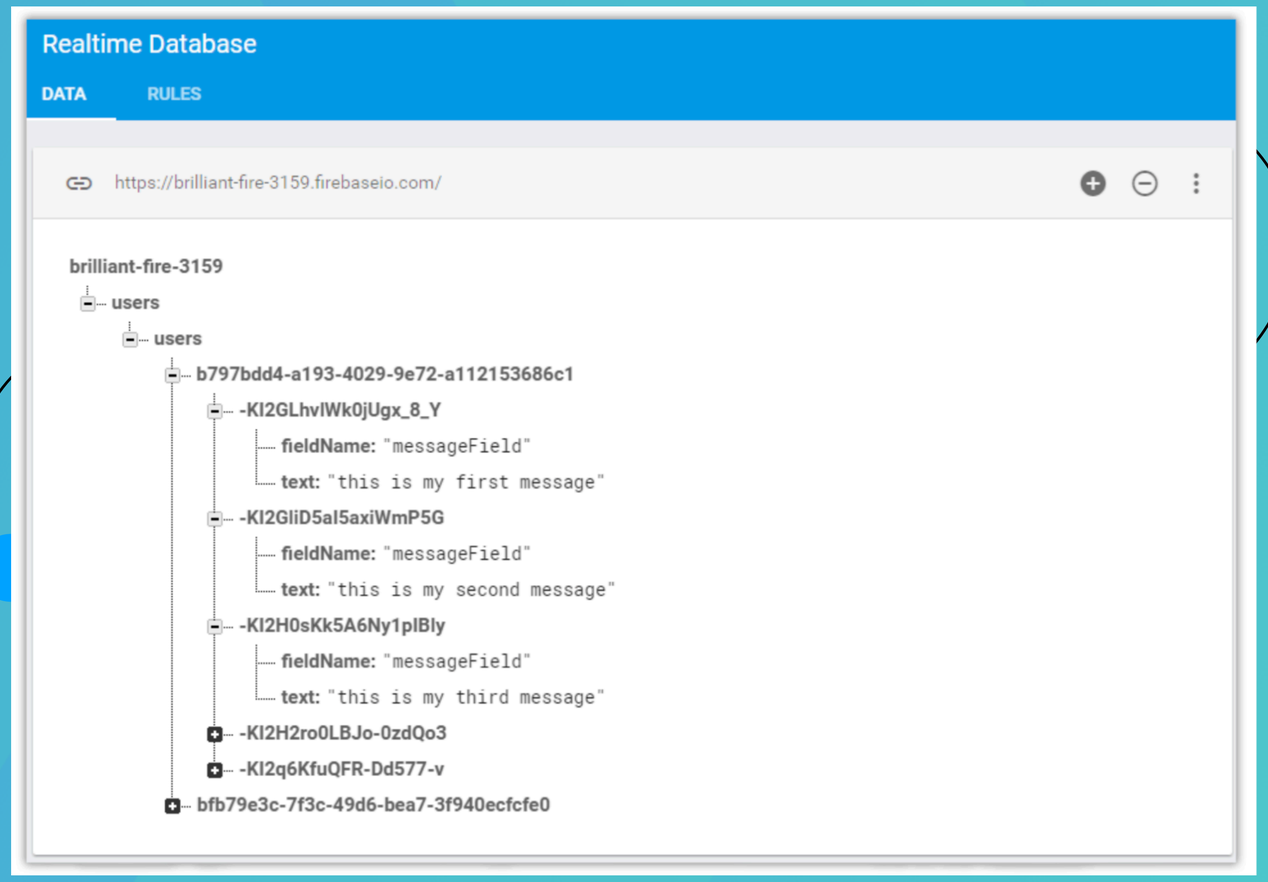


twitch.tv/dancojocar

youtube.com/dancojocar

Flowable vs. Observable

```
val events: Observable<MotionEvent> = RxView.touches(paintView);

val users: Observable<User> = db.query("select * from...");
```



Flowable vs. Observable

Observable<MotionEvent>

```
interface Observer<T>{
    fun onNext(t: T)
    fun onComplete()
    fun onError(t: Throwable)
    fun onSubscribe(d: Disposable)
}

interface Disposable{
    fun dispose()
}
```

Flowable<User>

```
interface Subscriber<T>{
    fun onNext(t: T)
    fun onComplete()
    fun onError(t: Throwable)
    fun onSubscribe(s: Subscription)
}

interface Subscription{
    fun cancel()
    fun request(r: Long)
}
```

twitch.tv/dancojocar

youtube.com/dancojocar

Sources

```
interface UserManager {
    fun getUser(): Observable<User>
    fun setName(name: String)
    fun setAge(age: Int)
}
```

Specialized Sources

- Encoding subsets of Observable into the type system:
  - Single scalar
    - Either succeeds with an item or an error.
    - No backpressure support.
  - Completable Runnable
    - Either completes or errors. Has no items!
    - No backpressure support.
  - Maybe Optional
    - Either succeeds with an item, completes with no items, or error.
    - No backpressure support.

twitch.tv/dancojocar

youtube.com/dancojocar

Sources

```
interface UserManager {
    fun getUser(): Observable<User>
    fun setName(name: String): Completable
    fun setAge(age: Int): Completable
}
```

twitch.tv/dancojocar

youtube.com/dancojocar

Creating Sources

```
Flowable.just("Hello")
Flowable.just("Hello", "World")

Observable.just("Hello")
Observable.just("Hello", "World")

Maybe.just("Hello")
val array = arrayListOf("Hello", "World")
val list = array.toList()
Single.just("Hello")
Flowable.fromArray(array)
Flowable.fromIterable(list)
Observable.fromCallable {
    setName()
}
Observable.fromArray(array)
Observable.fromIterable(list)
```

Creating Sources

```
val url = "https://example.com"

val request = Request.Builder().url(url).build()
val client = OkHttpClient()

Observable.fromCallable {
    client.newCall(request).execute()
}
```

twitch.tv/dancojocar

youtube.com/dancojocar

Create Sources

```
Observable.create<String> {
    it.onNext("Hello")
    it.onNext("World")
    it.onComplete()
}
```

```
Observable.create(ObservableOnSubscribe<String> {
    it.onNext("Hello")
    it.onComplete()
})
```

```
Observable.create(ObservableOnSubscribe<String>(
    function = fun(it: ObservableEmitter<String>) {
        it.onNext("Hello")
        it.onComplete()
    }))
```

twitch.tv/dancojocar

youtube.com/dancojocar

Create Sources

```
val request = Request.Builder().url(url).build()
val client = OkHttpClient()
Observable.create<String> {
    val call = client.newCall(request)
    it.setCancellable { call.cancel() }
    call.enqueue(object : Callback {
        override fun onFailure(call: Call, e: IOException) {
            it.onError(e)
        }

        override fun onResponse(call: Call, response: Response) {
            it.onNext(response.body().toString())
            it.onComplete()
        }
    })
}
```

```
Observable.create<View> {
    it.setCancellable { textView.setOnClickListener(null) }
    textView.setOnClickListener { v -> it.onNext(v) }
}
```

twitch.tv/dancojocar

youtube.com/dancojocar

Observing Sources

`Observable<MotionEvent>`

```
interface Observer<T>{
    fun onNext(t: T)
    fun onComplete()
    fun onError(t: Throwable)
    fun onSubscribe(d: Disposable)
}
```

```
interface Disposable{
    fun dispose()
}
```

`Flowable<User>`

```
interface Subscriber<T>{
    fun onNext(t: T)
    fun onComplete()
    fun onError(t: Throwable)
    fun onSubscribe(s: Subscription)
}
```

```
interface Subscription{
    fun cancel()
    fun request(r: Long)
}
```

twitch.tv/dancojocar

youtube.com/dancojocar

Observing Sources

```
val observable: Observable<String> = Observable.just("Hello")

val observer = object : DisposableObserver<String>() {
    override fun onComplete() {
        //...
    }

    override fun onNext(t: String) {
        //...
    }

    override fun onError(e: Throwable) {
        //...
    }
}
observable.subscribe(observer)
disposable.dispose()
observer.dispose()
```

How to we dispose?

twitch.tv/dancojocar

youtube.com/dancojocar

Rx{Java|Kotlin|Swift|Dart}

- A set of classes for representing sources of data.
- A set of classes for listening to data sources.
- A set of methods for modifying and composing the data.

RxSwift

Operators

- Manipulate or combine data in some way.
- Manipulate threading in some way.
- Manipulate emissions in some way.

twitch.tv/dancojocar

youtube.com/dancojocar

Operators

```
val greeting = Observable.just("Hello")
val yelling = greeting.map { it.toUpperCase() }
```

twitch.tv/dancojocar

youtube.com/dancojocar

Operators

```
class UserActivity : AppCompatActivity() {
    val um: UserManager = UserManagerImpl()
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_user)
        um.setName("John Doe", object : UserManager.Listener {
            override fun success(user: User) {
                um.setAge(42, object : UserManager.Listener {
                    override fun success(user: User) {
                        runOnUiThread {
                            if (!isDestroyed) {
                                textView.text = user.name
                            }
                        }
                    }
                    override fun failed(error: UserException) {
                        loge("Unable to update the user details", error)
                    }
                })
            }
            //...
        })
    }
}
```

twitch.tv/dancojocar

youtube.com/dancojocar

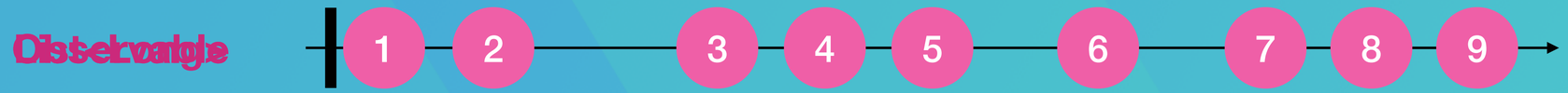
Operators

```
val url = "https://example.com"
  val user: Observable<User> = um.getUser()
  val mainThreadUser = user.observeOn(AndroidSchedulers.mainThread())
val request = Request.Builder().url(url).build()
val client = OkHttpClient()

val reponse = Observable.fromCallable { client.newCall(request).execute() }
    .subscribeOn(Schedulers.io()).map { it.body()?.string() }
    .flatMap { Observable.fromArray(it.split(" ")) }
    .observeOn(AndroidSchedulers.mainThread())
```

twitch.tv/dancojocar

youtube.com/dancojocar

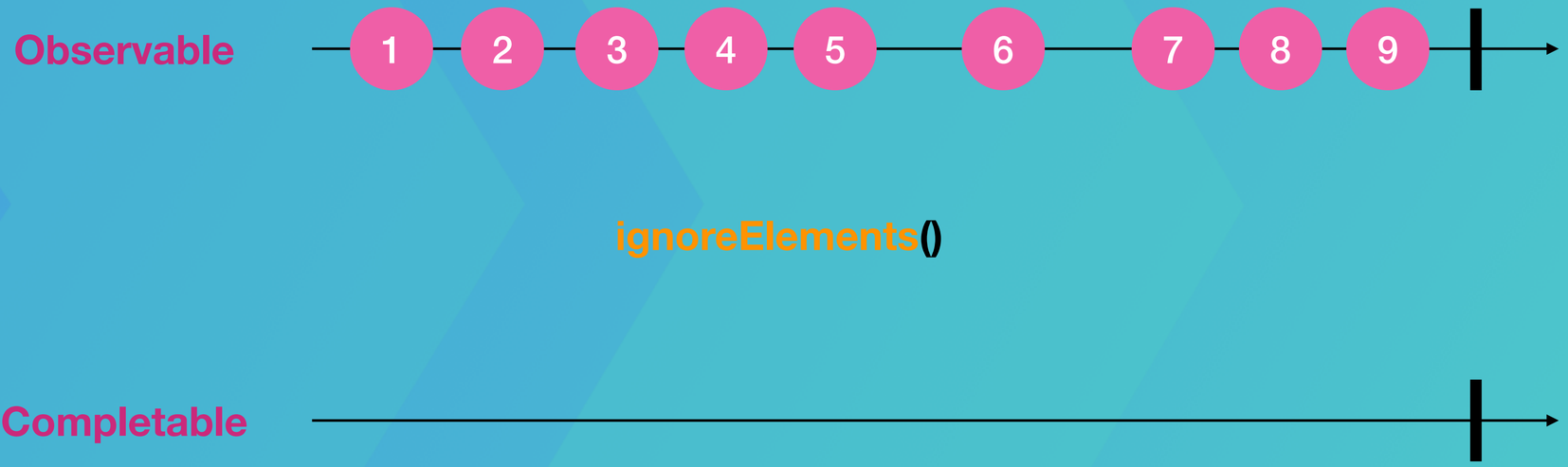
Operators



twitch.tv/dancojocar

youtube.com/dancojocar

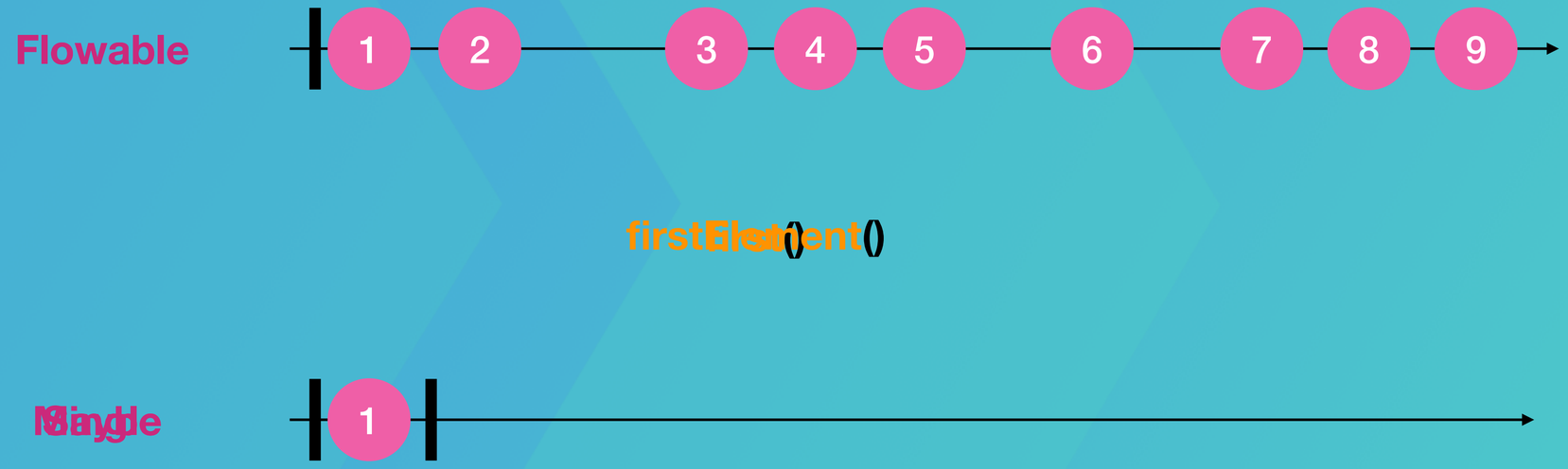
Operators



twitch.tv/dancojocar

youtube.com/dancojocar

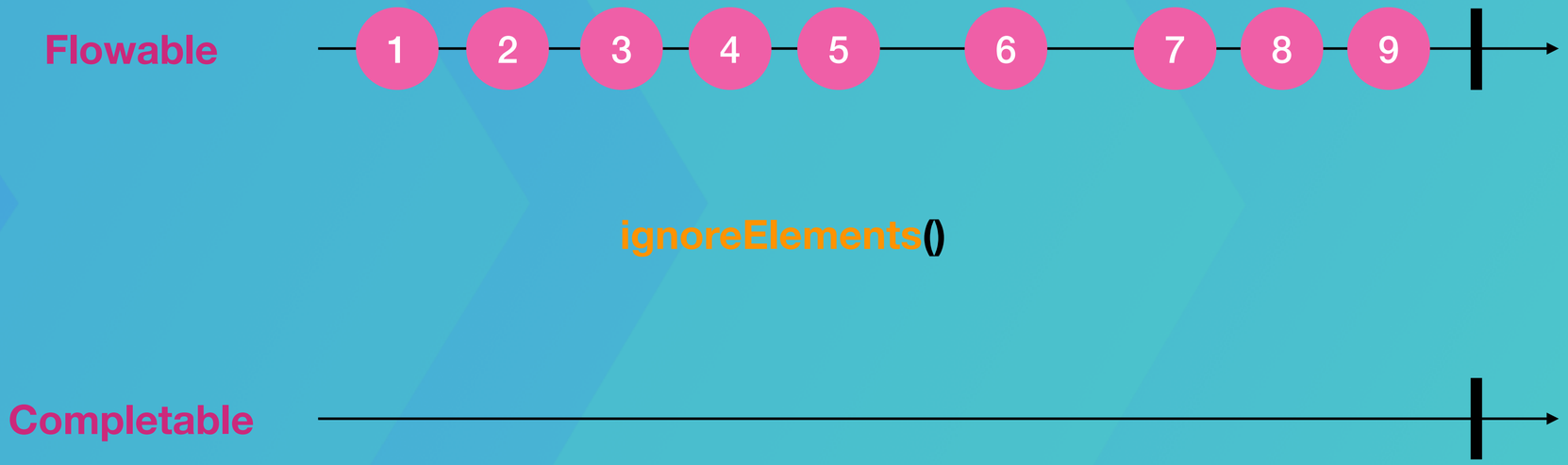
Operators



twitch.tv/dancojocar

youtube.com/dancojocar

Operators



http://reactivex.io/documentation/operators.html

twitch.tv/dancojocar

youtube.com/dancojocar

Being Reactive

```
// onCreate
val disposables = CompositeDisposable()
disposables.add(um.getUser()
    .observeOn(AndroidSchedulers.mainThread())
    .subscribeWith(object: DisposableObserver<User>(){
        override fun onNext(user: User) {
        }
        override fun onError(e: Throwable) {
        }
        override fun onComplete() {
        }
    }))

// onDestroy
disposable.dispose()
```

twitch.tv/dancojocar

youtube.com/dancojocar

Dependencies

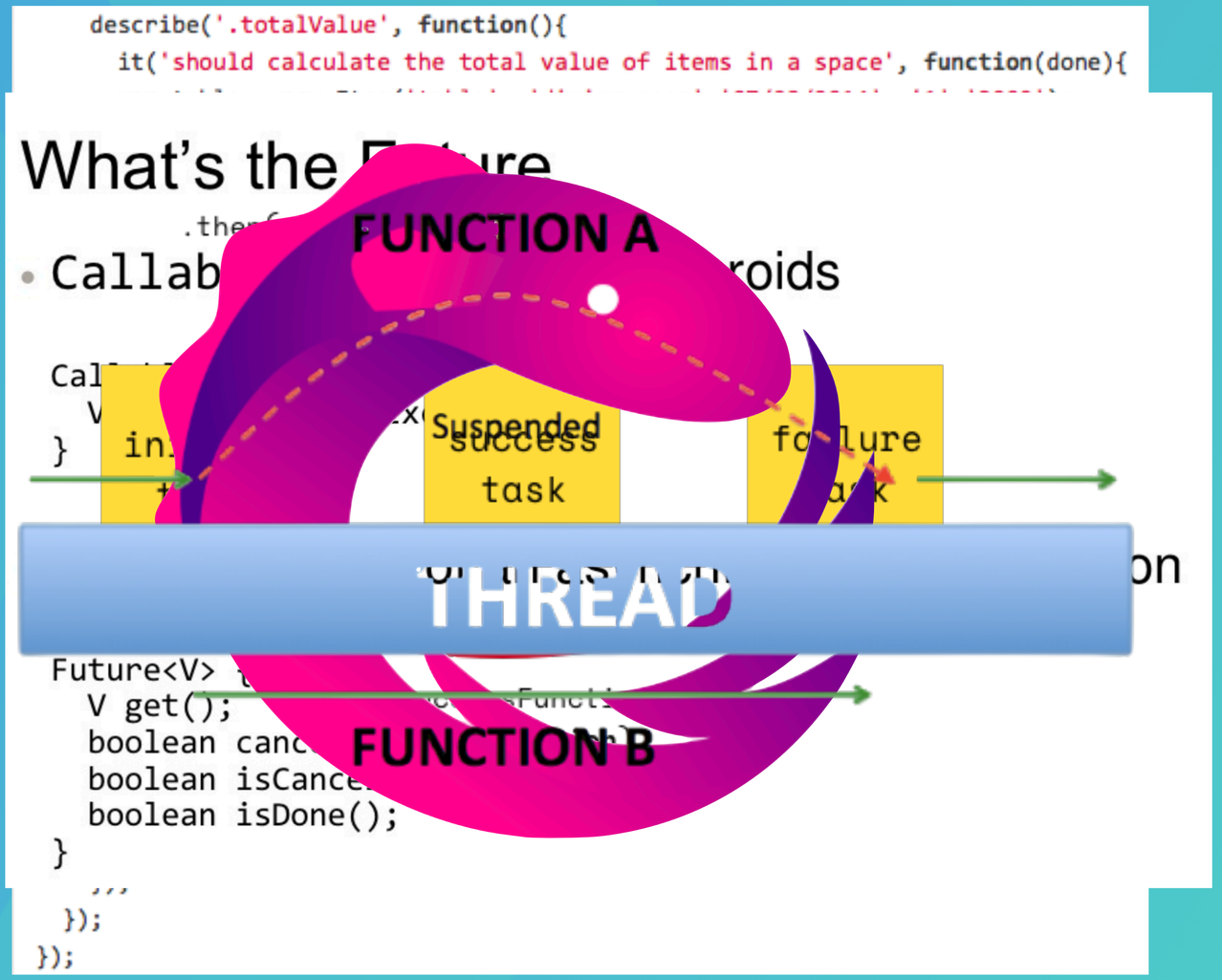

```
allprojects {
    repositories {
        maven { url "https://oss.jfrog.org/libs-snapshot" }
    }
}

dependencies {
    implementation("io.reactivex.rxjava3:rxkotlin:3.0.0")
    implementation 'io.reactivex.rxjava3:rxandroid:3.0.0'
    // Because RxAndroid releases are few and far between, it is recommended you also
    // explicitly depend on RxJava's latest version for bug fixes and new features.
    // (see https://github.com/ReactiveX/RxJava/releases for latest 3.x.x version)
    implementation 'io.reactivex.rxjava3:rxjava:3.0.0'
}
```

twitch.tv/dancojocar

youtube.com/dancojocar

Options

- Callbacks
- Futures
- Promises
- Rx
- Coroutines



https://kotlinlang.org/docs/reference/coroutines-overview.html

twitch.tv/dancojocar

youtube.com/dancojocar

Coroutines


```
fun requestToken(): Token {
    // make a token request and waits
    // blocks the thread while waiting for result
    return token
}

fun createPost(token: Token, item: Item): Post {
    logd("Posting an $item using $token")
    // sends the item to the server and waits
    // sends post item to the server and waits
    return post
}

fun processPost(post: Post) {
    // processing the post
    logd("Processing a $post")
}

fun postItem(item: Item) {
    val token = requestToken()
    val post = createPost(token, item)
    processPost(post)
}
```

twitch.tv/dancojocar

youtube.com/dancojocar

Coroutines

```
suspend fun requestToken(): Token {
    // make a token request and suspends
    return token
}

suspend fun createPost(token: Token, item: Item): Post {
    logd("Posting an $item using $token")
    //sends the item to the server and waits
    return post
}

fun processPost(post: Post) {
    // processing the post
    logd("Processing a $post")
}

fun postItem(item: Item) {
    val token = requestToken()
    val post = createPost(token, item)
    processPost(post)
}
```

twitch.tv/dancojocar

youtube.com/dancojocar

Bonuses

Regular exception handling:

```
try {
    for ((token, item) in list) {
        createPost(token, item)
    } catch (e: BadTokenException) {
        // ...
    }
```

Regular higher-order function:

```
file.readLines().forEach { line ->
    createPost(token, line.toItem())
}
```

Any of: forEach, let, apply, repeat, filter, map, use, etc.

Builders

twitch.tv/dancojocar

youtube.com/dancojocar

```
suspend fun requestToken(): Token {
    // make a token request and suspends
    return token
}

suspend fun createPost(token: Token, item: Item): Post {
    logd("Posting an $item using $token")
    //sends the item to the server and waits
    return post
}

fun processPost(post: Post) {
    // processing the post
    logd("Processing a $post")
}

fun postItem(item: Item) {
    GlobalScope.launch(Dispatchers.Main) {
        val token = requestToken()
        val post = createPost(token, item)
        processPost(post)
    }
}
```

Suspend function 'requestToken' should be called only from a coroutine or another suspend function

Builders

```
public fun CoroutineScope.launch(
    context: CoroutineContext = EmptyCoroutineContext,
    start: CoroutineStart = CoroutineStart.DEFAULT,
    block: suspend CoroutineScope.() -> Unit
): Job {
    val newContext = newCoroutineContext(context)
    val coroutine = if (start.isLazy)
        LazyStandaloneCoroutine(newContext, block) else
        StandaloneCoroutine(newContext, active = true)
    coroutine.start(start, coroutine, block)
    return coroutine
}
```

Dependencies

twitch.tv/dancojocar

youtube.com/dancojocar


```
dependencies {
  implementation “org.jetbrains.kotlinx:kotlinx-coroutines-core:1.4.1"
  implementation "org.jetbrains.kotlinx:kotlinx-coroutines-android:1.4.1"
}
```

twitch.tv/dancojocar

youtube.com/dancojocar

Usage

```
class MyViewModel : ViewModel() {
  private val _result = MutableLiveData<String>()
  val result: LiveData<String> = _result

  init {
    viewModelScope.launch {
      val computationalResult = doComputation()
      _result.value = computationalResult
    }
  }
}
```

```
implementation "androidx.lifecycle:lifecycle-viewmodel-ktx:2.2.0"
```

twitch.tv/dancojocar

youtube.com/dancojocar

Usage

```
class MyNewViewModel : ViewModel() {
  val result = liveData {
    emit(doComputation())
  }
}
```

implementation “androidx.lifecycle:lifecycle-livedata-ktx:2.2.0"

Flow

```
suspend fun foo(): List<Int> {
    delay(1000) // pretend we are doing something asynchronous here
    return listOf(1, 2, 3)
}

fun main() = runBlocking<Unit> {
    foo().forEach { value -> println(value) }
}
```

```
Output:

1
2
3
```

Flow

```
fun foo(): Flow<Int> = flow { // flow builder
    for (i in 1..3) {
        delay(100) // pretend we are doing something useful here
        emit(i) // emit next value
    }
}
fun main() = runBlocking<Unit> {
    // Launch a concurrent coroutine to check if the main thread is blocked
    launch {
        for (k in 1..3) {
            println("I'm not blocked $k")
            delay(100)
        }
    }
    // Collect the flow
    foo().collect { value -> println(value) }
}
```

```
Output:

I'm not blocked 1
1
I'm not blocked 2
2
I'm not blocked 3
3
```

twitch.tv/dancojocar

youtube.com/dancojocar

Flows are cold

```
fun foo(): Flow<Int> = flow {
    println("Flow started")
    for (i in 1..3) {
        delay(100)
        emit(i)
    }
}
fun main() = runBlocking<Unit> {
    println("Calling foo...")
    val flow = foo()
    println("Calling collect...")
    flow.collect { value -> println(value) }
    println("Calling collect again...")
    flow.collect { value -> println(value) }
}
```

```
Output:

Calling foo...
Calling collect...
Flow started
1
2
3
Calling collect again...
Flow started
1
2
3
```

Flow Cancellation

```
fun fooCancellation(): Flow<Int> = flow {
  for (i in 1..3) {
    delay(100)
    println("Emitting $i")
    emit(i)
  }
}

fun main() = runBlocking<Unit> {
  withTimeoutOrNull(250) { // Timeout after 250ms
    fooCancellation().collect { value -> println(value) }
  }
  println("Done")
}
```

```
Output:

Emitting 1
1
Emitting 2
2
Done
```

twitch.tv/dancojocar

youtube.com/dancojocar

Flow Operators

```
suspend fun performRequest(request: Int): String {
    delay(1000) // imitate long-running asynchronous work
    return "response $request"
}

fun main() = runBlocking<Unit> {
    (1..3).asFlow() // a flow of requests
        .map { request -> performRequest(request) }
        .collect { response -> println(response) }
}
```

```
Output:

response 1
response 2
response 3
```

twitch.tv/dancojocar

youtube.com/dancojocar

Transform Operator

```
suspend fun performRequest(request: Int): String {
    delay(1000) // imitate long-running asynchronous work
    return "response $request"
}
(1..3).asFlow() // a flow of requests
    .transform { request ->
        emit("Making request $request")
        emit(performRequest(request))
    }
    .collect { response -> println(response) }
```

```
Output:

Making request 1
response 1
Making request 2
response 2
Making request 3
response 3
```

Size-limiting Operators

```
fun numbers(): Flow<Int> = flow {
  try {
    emit(1)
    emit(2)
    println("This line will not execute")
    emit(3)
  } finally {
    println("Finally in numbers")
  }
}
@ExperimentalCoroutinesApi
fun main() = runBlocking<Unit> {
  numbers()
    .take(2) // take only the first two
    .collect { value -> println(value) }
}
```

```
Output:

1
2
Finally in numbers
```

twitch.tv/dancojocar

youtube.com/dancojocar

Terminal Flow Operators

```
@ExperimentalCoroutinesApi
fun main() = runBlocking<Unit> {
  val sum = (1..5).asFlow()
    .map { it * it } // squares of numbers from 1 to 5
    .reduce { a, b -> a + b } // sum them (terminal operator)
  println(sum)
}
```

```
Output: 55
```

kotlinlang.org/docs/reference/coroutines/flow.html

twitch.tv/dancojocar

youtube.com/dancojocar

Lecture outcomes

- Understand the reactive programming (Rx) concepts.
- Use Rx to re-write the application logic.
- Design a real time application logic against a real time backend.
- Understand coroutines and flow.