# Скрипач не нужен: отказываемся от RxJava в пользу корутин в Kotlin

# Disclaimer

Все сказанное здесь является продуктом боевого и исследовательского опыта. Возможны допущения, опечатки, неточности или ошибки. Проверяйте все сами.

# О докладчике

→ Владимир Иванов(EPAM Systems)

→ Android приложения: > 15 опубликованных, > 7 лет опыта

→ Широкий интерес в мобильных технологиях

# .NET?

# JS?

# Async/await

```
export const loginAsync = async (login, password) => {
    let auth = `Basic ${base64(login:password)}`
    try {
        let result = await fetch('https://api.github.com/user', auth)
        if (result.status === 200) {
            return { result.body }
        } else {
            return { error: `Failed to login with ${result.status}` }
        }
    } catch (error) {
        return { error: `Failed to login with ${error}` }
    }
}
```

# Легко читается?

# Kotlin может также!

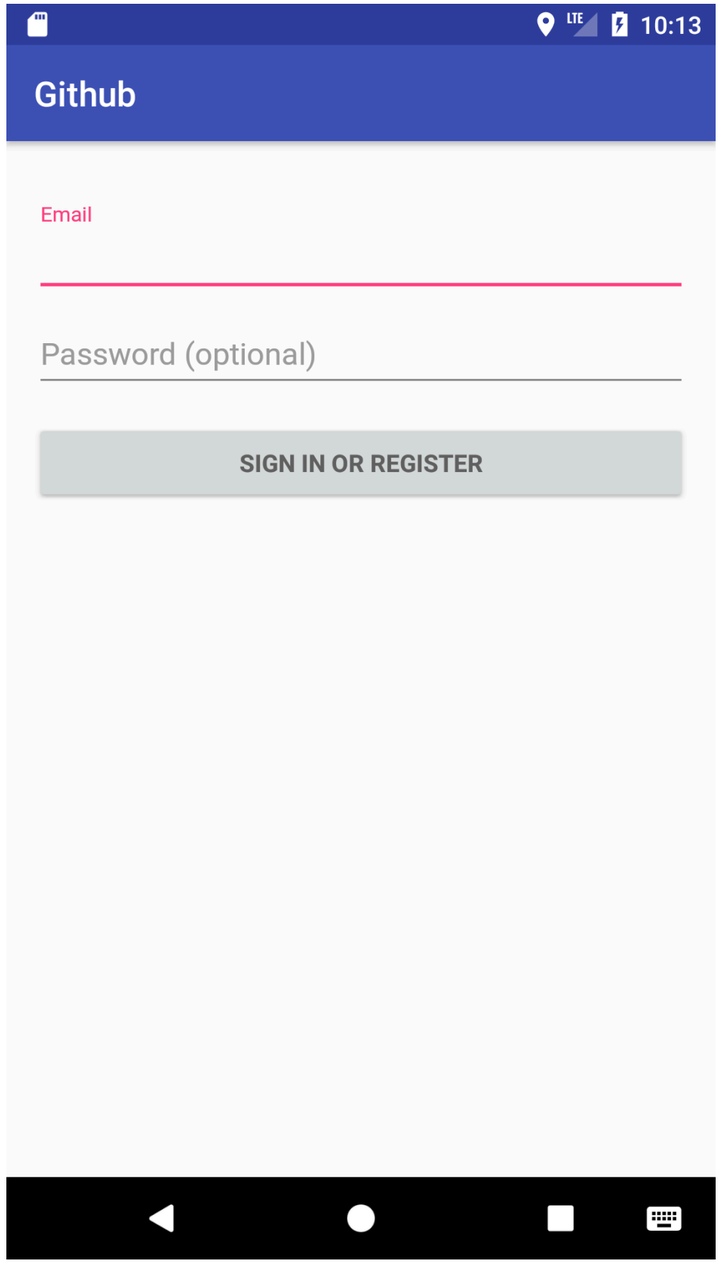
# Что мы с вами делаем сейчас?

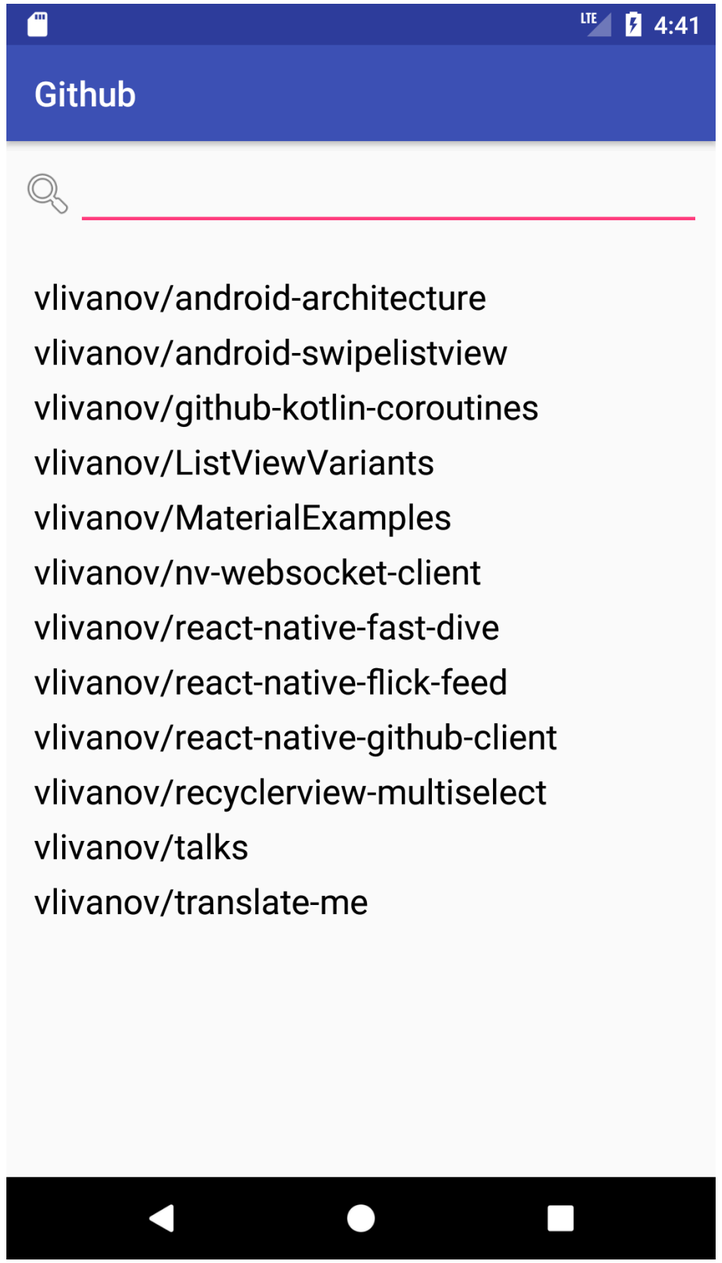
# Github login application

→ Вход с логином/паролем

→ Получение репозиториев пользователя

→ Поиск среди всех репозиториев гитхаба


10:13
Github
Email
Password (optional)
SIGN IN OR REGISTER


4:41
Github
vlivanov/android-architecture
vlivanov/android-swipelistview
vlivanov/github-kotlin-coroutines
vlivanov/ListViewVariants
vlivanov/MaterialExamples
vlivanov/nv-websocket-client
vlivanov/react-native-fast-dive
vlivanov/react-native-flick-feed
vlivanov/react-native-github-client
vlivanov/recyclerview-multiselect
vlivanov/talks
vlivanov/translate-me

**Vladimir Ivanov** @vvsevolodovich · 17 февр.

If you’re starting a brand new #Android project in @kotlin you use for background

Язык твита: английский

**28%** Coroutines

**65%** RxJava 2

**4%** Threadpool Executors

**3%** Loaders

164 голоса • Окончательные итоги

9 7

# RxJava 2 implementation

```
interface ApiClientRx {

    fun login(auth: Authorization)
        : Single<GithubUser>
    fun getRepositories
        (reposUrl: String, auth: Authorization)
        : Single<List<GithubRepository>>

}
```

# RxJava 2 implementation

```
interface ApiClientRx {

    fun login(auth: Authorization)
        : Single<GithubUser>
    fun getRepositories
        (reposUrl: String, auth: Authorization)
        : Single<List<GithubRepository>>

}
```

# RxJava 2 implementation

```
override fun login(auth: Authorization)
    : Single<GithubUser?> = Single.fromCallable {
    val response = get("https://api.github.com/user", auth = auth)
    if (response.statusCode != 200) {
        throw RuntimeException("Incorrect login or password")
    }

    val jsonObject = response.jsonObject
    with(jsonObject) {
        return@with GithubUser(getString("login"), getInt("id"),
                getString("repos_url"), getString("name"))
    }
}
```

# RxJava 2 implementation

```
override fun login(auth: Authorization)
    : Single<GithubUser?> = Single.fromCallable {
    val response = get("https://api.github.com/user", auth = auth)
    if (response.statusCode != 200) {
        throw RuntimeException("Incorrect login or password")
    }

    val jsonObject = response.jsonObject
    with(jsonObject) {
        return@with GithubUser(getString("login"), getInt("id"),
                getString("repos_url"), getString("name"))
    }
}
```

# RxJava 2 implementation

```
override fun getRepositories
    (repoUrl: String, authorization: Authorization)
    : Single<List<GithubRepository>> {

    return Single.fromCallable({
            toRepos(get(repoUrl, auth = authorization).jsonArray)
    })
}
```

```
private fun attemptLoginRx() {
    showProgress(true)
    apiClient.login(auth)
            .flatMap {
                user -> apiClient.getRepositories(user.repos_url, auth)
            }
            .map {
                list -> list.map { it.full_name }
            }
            .subscribeOn(Schedulers.io())
            .observeOn(AndroidSchedulers.mainThread())
            .doFinally { showProgress(false) }
            .subscribe(
                    { list -> showRepositories(this, list)    },
                    { error -> Log.e("TAG", "Failed to show repos", error) }
            )
}
```

```
private fun attemptLoginRx() {
    showProgress(true)
    apiClient.login(auth)
            .flatMap {
                user -> apiClient.getRepositories(user.repos_url, auth)
            }
            .map {
                list -> list.map { it.full_name }
            }
            .subscribeOn(Schedulers.io())
            .observeOn(AndroidSchedulers.mainThread())
            .doFinally { showProgress(false) }
            .subscribe(
                    { list -> showRepositories(this, list)    },
                    { error -> Log.e("TAG", "Failed to show repos", error) }
            )

}
```

```
private fun attemptLoginRx() {
    showProgress(true)
    apiClient.login(auth)
            .flatMap {
                user -> apiClient.getRepositories(user.repos_url, auth)
            }
            .map {
                list -> list.map { it.full_name }
            }
            .subscribeOn(Schedulers.io())
            .observeOn(AndroidSchedulers.mainThread())
            .doFinally { showProgress(false) }
            .subscribe(
                    { list -> showRepositories(this, list)    },
                    { error -> Log.e("TAG", "Failed to show repos", error) }
            )

}
```

```
private fun attemptLoginRx() {
    showProgress(true)
    apiClient.login(auth)
            .flatMap {
                user -> apiClient.getRepositories(user.repos_url, auth)
            }
            .map {
                list -> list.map { it.full_name }
            }
            .subscribeOn(Schedulers.io())
            .observeOn(AndroidSchedulers.mainThread())
            .doFinally { showProgress(false) }
            .subscribe(
                    { list -> showRepositories(this, list)    },
                    { error -> Log.e("TAG", "Failed to show repos", error) }
            )
}
```

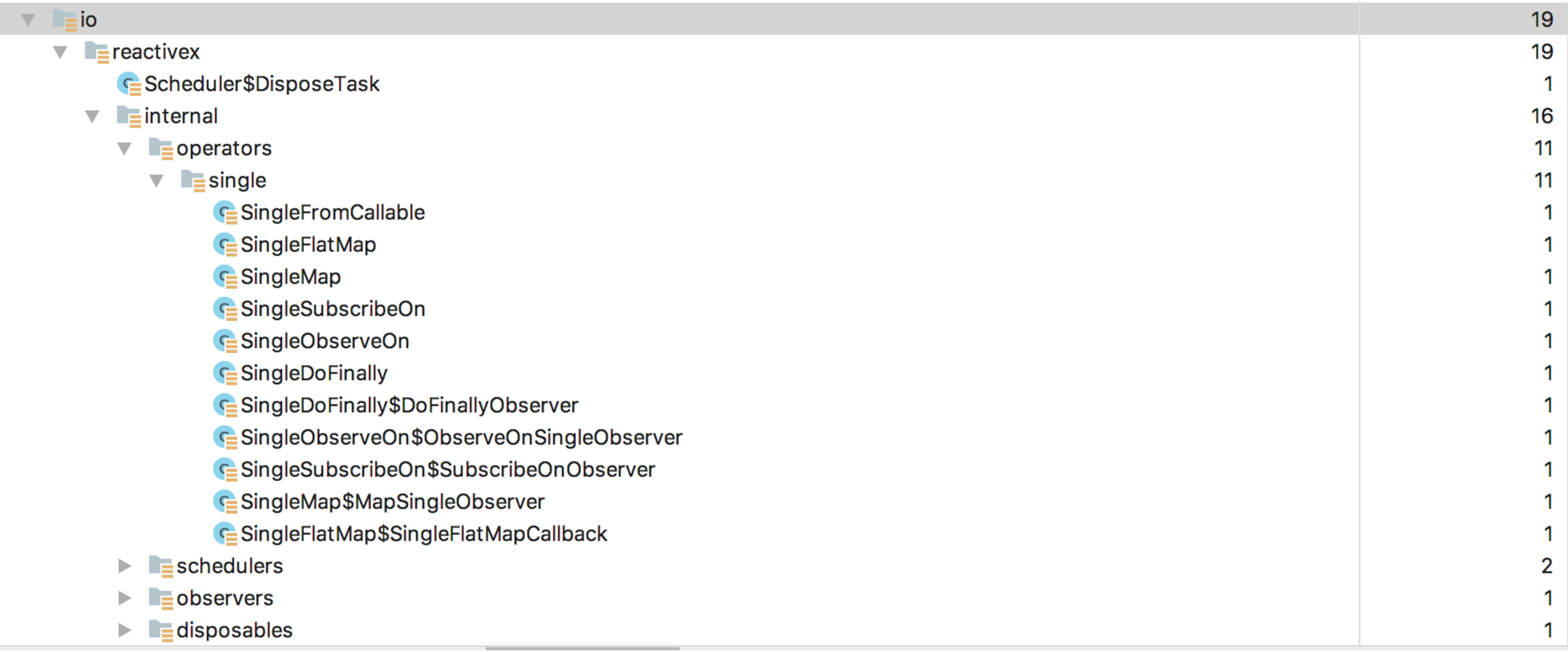
# Сложности?

# Сложности?

→ Большое количество обьектов под капотом

→ Неясный стектрейс

→ Крутая кривая обучения


io 19
reactivex 19
Scheduler$DisposeTask 1
internal 16
operators 11
single 11
SingleFromCallable 1
SingleFlatMap 1
SingleMap 1
SingleSubscribeOn 1
SingleObserveOn 1
SingleDoFinally 1
SingleDoFinally$DoFinallyObserver 1
SingleObserveOn$ObserveOnSingleObserver 1
SingleSubscribeOn$SubscribeOnObserver 1
SingleMap$MapSingleObserver 1
SingleFlatMap$SingleFlatMapCallback 1
schedulers 2
observers 1
disposables 1

```
// new SingleFlatMap()
val flatMap = apiClient.login(auth)
        .flatMap { apiClient.getRepositories(it.repos_url, auth) }
// new SingleMap
val map = flatMap
        .map { list -> list.map { it.full_name } }
// new SingleSubscribeOn
val subscribeOn = map
        .subscribeOn(Schedulers.io())
// new SingleObserveOn
val observeOn = subscribeOn
        .observeOn(AndroidSchedulers.mainThread())
// new SingleDoFinally
val doFinally = observeOn
        .doFinally { showProgress(false) }
// new ConsumerSingleObserver
val subscribe = doFinally
        .subscribe(
                { list -> showRepositories(this, list) },
                { error -> Log.e("TAG", "Failed to show repos", error) }
        )
    }
```

```
at com.epam.talks.github.model.ApiClientRx$ApiClientRxImpl$login$1.call(ApiClientRx.kt:16)
    at io.reactivex.internal.operators.single.SingleFromCallable.subscribeActual(SingleFromCallable.java:44)
    at io.reactivex.Single.subscribe(Single.java:3096)
    at io.reactivex.internal.operators.single.SingleFlatMap.subscribeActual(SingleFlatMap.java:36)
    at io.reactivex.Single.subscribe(Single.java:3096)
    at io.reactivex.internal.operators.single.SingleMap.subscribeActual(SingleMap.java:34)
    at io.reactivex.Single.subscribe(Single.java:3096)
    at io.reactivex.internal.operators.single.SingleSubscribeOn$SubscribeOnObserver.run(SingleSubscribeOn.java:89)
    at io.reactivex.Scheduler$DisposeTask.run(Scheduler.java:463)
    at io.reactivex.internal.schedulers.ScheduledRunnable.run(ScheduledRunnable.java:66)
    at io.reactivex.internal.schedulers.ScheduledRunnable.call(ScheduledRunnable.java:57)
    at java.util.concurrent.FutureTask.run(FutureTask.java:266)
    at java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.run(ScheduledThreadPoolExecutor.java:301)
    at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1162)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:636)
    at java.lang.Thread.run(Thread.java:764)
```

# Корутины в Котлине

→ Доступны с kotlin 1.1

→ Статус experimental, но апи стабилизировано

# Реализация на корутинах

```
interface ApiClient {

    fun login(auth: Authorization)
        : Deferred<GithubUser>
    fun getRepositories
        (reposUrl: String, auth: Authorization)
        : Deferred<List<GithubRepository>>

}
```

# Реализация на корутинах

```
interface ApiClient {

    fun login(auth: Authorization)
        : Deferred<GithubUser>
    fun getRepositories
        (reposUrl: String, auth: Authorization)
        : Deferred<List<GithubRepository>>

}
```

```
private fun attemptLogin() {
    launch(UI) {
        val auth = BasicAuthorization(login, pass)
        try {
            showProgress(true)
            val userInfo = apiClient.login(auth).await()
            val repoUrl = userInfo.repos_url
            val list = apiClient.getRepositories(repoUrl, auth).await()
            showRepositories(
                this,
                list.map { it -> it.full_name }
            )
        } catch (e: RuntimeException) {
            showToast("Oops!")
        } finally {
            showProgress(false)
        }
    }
}
```

```
private fun attemptLogin() {
    launch(UI) {
        val auth = BasicAuthorization(login, pass)
        try {
            showProgress(true)
            val userInfo = apiClient.login(auth).await()
            val repoUrl = userInfo.repos_url
            val list = apiClient.getRepositories(repoUrl, auth).await()
            showRepositories(
                this,
                list.map { it -> it.full_name }
            )
        } catch (e: RuntimeException) {
            showToast("Oops!")
        } finally {
            showProgress(false)
        }
    }
}
```

# Плюсы

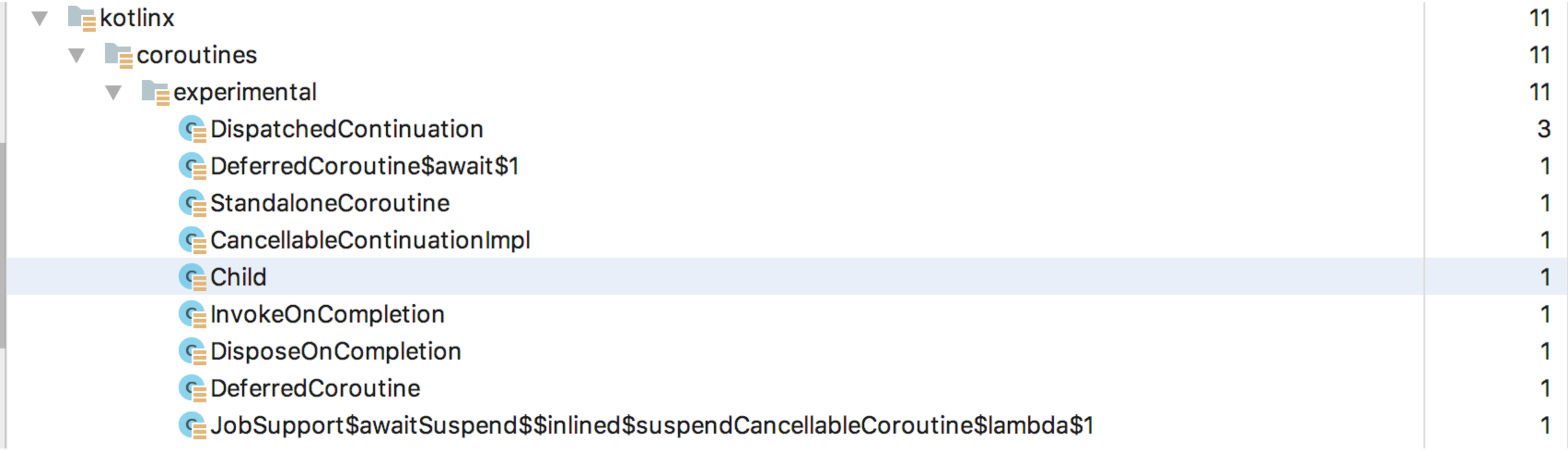
→ Легко читается, ибо

→ Асинхронный код написан в direct-стиле

→ Обработка ошибок, как для синхронного кода(try-catch-finally)

# Как насчет минусов RxJava?

# Объекты?


kotlinx 11
coroutines 11
experimental 11
DispatchedContinuation 3
DeferredCoroutine$await$1 1
StandaloneCoroutine 1
CancellableContinuationImpl 1
Child 1
InvokeOnCompletion 1
DisposeOnCompletion 1
DeferredCoroutine 1
JobSupport$awaitSuspend$$inlined$suspendCancellableCoroutine$lambda$1 1

# 19 -> 11

# Стектрейс?

```
at com.epam.talks.github.model.ApiClient$ApiClientImpl$login$1.doResume(ApiClient.kt:27)
   at kotlin.coroutines.experimental.jvm.internal.CoroutineImpl.resume(CoroutineImpl.kt:54)
   at kotlinx.coroutines.experimental.DispatchedTask$DefaultImpls.run(Dispatched.kt:161)
   at kotlinx.coroutines.experimental.DispatchedContinuation.run(Dispatched.kt:25)
   at java.util.concurrent.ForkJoinTask$RunnableExecuteAction.exec(ForkJoinTask.java:1412)
   at java.util.concurrent.ForkJoinTask.doExec(ForkJoinTask.java:285)
   at java.util.concurrent.ForkJoinPool$WorkQueue.runTask(ForkJoinPool.java:1152)
   at java.util.concurrent.ForkJoinPool.scan(ForkJoinPool.java:1990)
   at java.util.concurrent.ForkJoinPool.runWorker(ForkJoinPool.java:1938)
   at java.util.concurrent.ForkJoinWorkerThread.run(ForkJoinWorkerThread.java:157)
```

| RxJava 2 | Coroutines |
| --- | --- |
| at com.epam.talks.github.model.ApiClientRx$ApiClientRxImpl$login$1.call(ApiClientRx.kt:16)io.reactivex.internal.operators.single.SingleFromCallable.subscribeActual(SingleFromCallable.java:44)io.reactivex.Single.subscribe(Single.java:3096)io.reactivex.internal.operators.single.SingleFlatMap.subscribeActual(SingleFlatMap.java:36)io.reactivex.Single.subscribe(Single.java:3096)io.reactivex.internal.operators.single.SingleMap.subscribeActual(SingleMap.java:34)io.reactivex.Single.subscribe(Single.java:3096)io.reactivex.internal.operators.single.SingleSubscribeOn$SubscribeOnObserver.run(SingleSubscribeOn.java:89)io.reactivex.Scheduler$DisposeTask.run(Scheduler.java:463)io.reactivex.internal.schedulers.ScheduledRunnable.run(ScheduledRunnable.java:66)io.reactivex.internal.schedulers.ScheduledRunnable.call(ScheduledRunnable.java:57)java.util.concurrent.FutureTask.run(FutureTask.java:266)java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.run(ScheduledThreadPoolExecutor.java:301)java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1162)java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:636)java.lang.Thread.run(Thread.java:764) | at com.epam.talks.github.model.ApiClient$ApiClientImpl$login$1.doResume(ApiClient.kt:27) kotlin.coroutines.experimental.jvm.internal.CoroutineImpl.resume(CoroutineImpl.kt:54) kotlinx.coroutines.experimental.DispatchedTask$DefaultImpls.run(Dispatched.kt:161) kotlinx.coroutines.experimental.DispatchedContinuation.run(Dispatched.kt:25) java.util.concurrent.ForkJoinTask$RunnableExecuteAction.exec(ForkJoinTask.java:1412) java.util.concurrent.ForkJoinTask.doExec(ForkJoinTask.java:285) java.util.concurrent.ForkJoinPool$WorkQueue.runTask(ForkJoinPool.java:1152) java.util.concurrent.ForkJoinPool.scan(ForkJoinPool.java:1990) java.util.concurrent.ForkJoinPool.runWorker(ForkJoinPool.java:1938) java.util.concurrent.ForkJoinWorkerThread.run(ForkJoinWorkerThread.java:157) |

# Прямой стиль кода

```
private fun attemptLogin() {
    launch(UI) {
        val auth = BasicAuthorization(login, pass)
        try {
            showProgress(true)
            val userInfo = login(auth).await()
            val repoUrl = userInfo!!.repos_url
            val list = getRepositories(repoUrl, auth).await()
            showRepositories(this, list.map { it -> it.full_name })
        } catch (e: RuntimeException) {
            showToast("Oops!")
        } finally {
            showProgress(false)
        }
    }
}
```

# Обработка ошибок средствами языка(не методами библиотеки)

```
private fun attemptLogin() {
    launch(UI) {
        val auth = BasicAuthorization(login, pass)
        try {
            showProgress(true)
            val userInfo = login(auth).await()
            val repoUrl = userInfo!!.repos_url
            val list = getRepositories(repoUrl, auth).await()
            showRepositories(this, list.map { it -> it.full_name })
        } catch (e: RuntimeException) {
            showToast("Oops!")
        } finally {
            showProgress(false)
        }
    }
}
```

# Использование stdlib

```
launch(UI) {
    showProgress(true)
    val auth = BasicAuthorization(login, pass)
    try {
        val userInfo = apiClient.login(auth).await()
        val repoUrl = userInfo!!.repos_url
        val repos = apiClient.getRepositories(repoUrl, auth).await()
        repeat(list.size, {
            apiClient.getRepoFollowers().await()
        })
        showRepositories(this, repos!!.map { it -> it.full_name })
    } catch (e: RuntimeException) {
        showToast("Oops!")
    } finally {
        showProgress(false)
    }
}
```

# Реализация асинхронной функции

```
override fun login(auth: Authorization) : Deferred<GithubUser?> = async {
    val response = get("https://api.github.com/user", auth = auth)
    if (response.statusCode != 200) {
        throw RuntimeException("Incorrect login or password")
    }

    val jsonObject = response.jsonObject
    with (jsonObject) {
        return@async GithubUser(getString("login"), getInt("id"),
                getString("repos_url"), getString("name"))
    }
}
```

# Реализация асинхронной функции

```
override fun login(auth: Authorization) : Deferred<GithubUser?> = async {
    val response = get("https://api.github.com/user", auth = auth)
    if (response.statusCode != 200) {
        throw RuntimeException("Incorrect login or password")
    }

    val jsonObject = response.jsonObject
    with (jsonObject) {
        return@async GithubUser(getString("login"), getInt("id"),
                getString("repos_url"), getString("name"))
    }
}
```

# Использование ФВП Async

```
fun login(...) : Deferred<GithubUser?> = async {

        return@async GithubUser(…)
}
```

# Что такое Async?

# Билдер для корутины

→ launch

→ async

→ runBlocking

→ withContext

# Launch возвращает Job

```
interface Job : CoroutineContext.Element {

    public val isActive: Boolean
    public val isCompleted: Boolean
    public val isCancelled: Boolean
    public fun getCancellationException(): CancellationException
    public fun start(): Boolean
}
```

# Async возвращает Deferred<T>

```
public actual interface Deferred<out T> : Job {

    public suspend fun await(): T
}
```

# Deferred это Future

# Deferred is Future

# Deferred is Future

→ неблокирующее

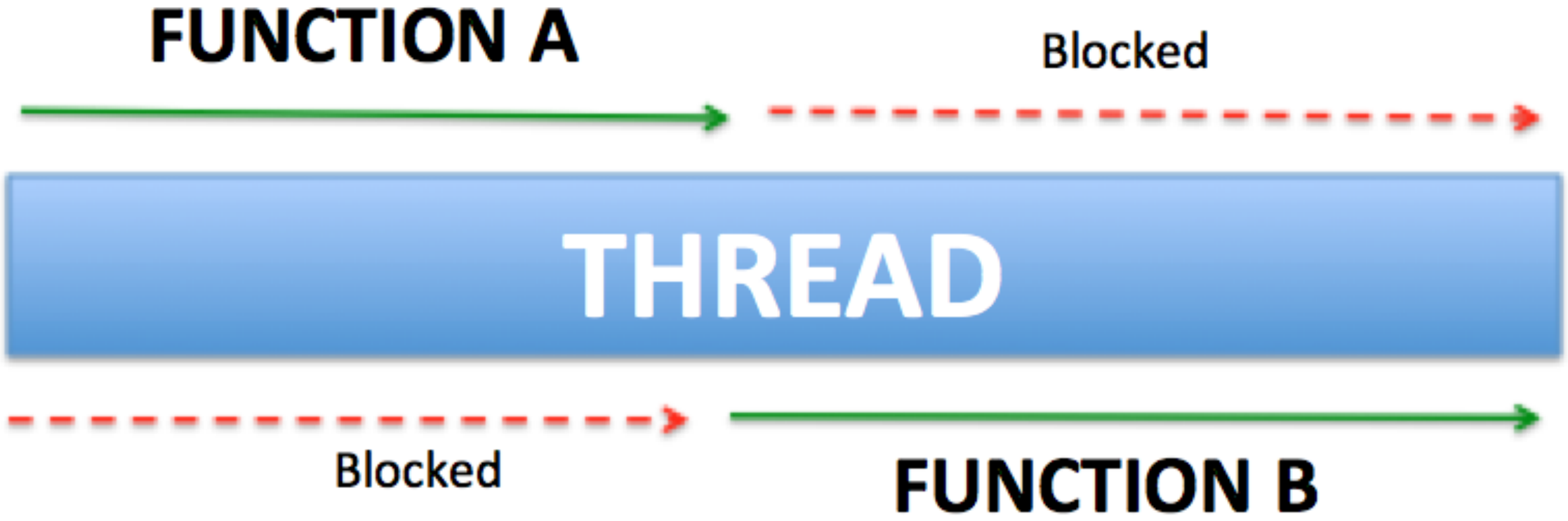
→ отменяемое

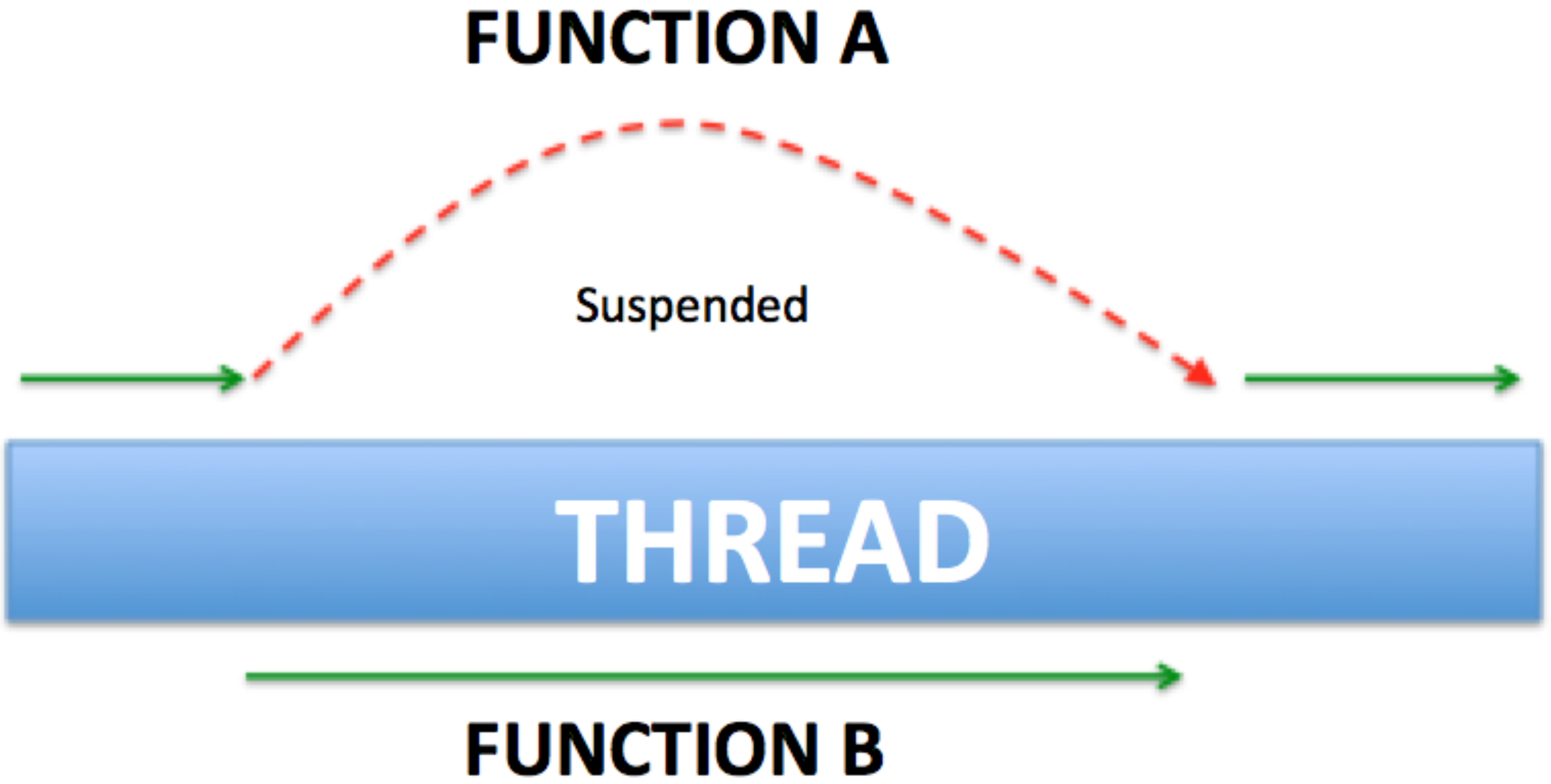
# Await - extension-функция

# Await - extension-функция

→ Как Future.get(), только неблокирующая

# Suspension


FUNCTION A
Blocked
THREAD
Blocked
FUNCTION B


FUNCTION A
Suspended
THREAD
FUNCTION B

# Suspending

→ приостановка выполнения

→ то есть, выполнение может быть восстановлено

→ Приостановка случается только в определенных местах

→ При вызове функций с модификатором `suspend`!

# Где же suspend?

```
public expect fun <T> async(
    context: CoroutineContext = DefaultDispatcher,
    start: CoroutineStart = CoroutineStart.DEFAULT,
    parent: Job? = null,
    block: suspend CoroutineScope.() -> T
): Deferred<T>
```

```
public expect fun <T> async(
    context: CoroutineContext = DefaultDispatcher,
    start: CoroutineStart = CoroutineStart.DEFAULT,
    parent: Job? = null,
    block: suspend CoroutineScope.() -> T
): Deferred<T>
```

# Отмена

# Отмена всегда кооперативна

# RxJava 2

# Отмена в RxJava в целом ок, но может требовать доступ к текущей подписке

# Отмена в корутинах тоже ок

# Как проверить отмену?

```
launch(UI) {
    showProgress(true)
    …
    try {
        val userInfo = apiClient.login(auth).await()
        if (!isActive) {
            return@launch
        }
    …
}
```

```
launch(UI) {
    showProgress(true)
    …
    try {
        val userInfo = apiClient.login(auth).await()
        if (!isActive) {
            return@launch
        }
    …
}
```

# Пишем тесты!

# Тест на RxJava

```
@Test
fun login() {
    val apiClientImpl = ApiClientRx.ApiClientRxImpl()
    val genericResponse = mockLoginResponse()

    staticMockk("khttp.KHttp").use {
        every { get("https://api.github.com/user", auth = any()) }
            returns genericResponse

        val githubUser =
                apiClientImpl
                        .login(BasicAuthorization("login", "pass"))

        githubUser.subscribe({ githubUser ->
            assertNotNull(githubUser)
            assertEquals("name", githubUser.name)
            assertEquals("url", githubUser.repos_url)
        })

    }
}
```

# Тест на RxJava

```
@Test
fun login() {
    …

        val githubUser =
                apiClientImpl
                        .login(BasicAuthorization("login", "pass"))

        githubUser.subscribe({ githubUser ->
            assertNotNull(githubUser)
            assertEquals("name", githubUser.name)
            assertEquals("url", githubUser.repos_url)
        })

    …
}
```

# Тест на корутины

```
@Test
fun login() {
    val apiClientImpl = ApiClient.ApiClientImpl()
    val genericResponse = mockLoginResponse()

    staticMockk("khttp.KHttp").use {
        every { get("https://api.github.com/user", auth = any()) }
            returns genericResponse

        runBlocking {
            val githubUser =
                    apiClientImpl
                        .login(BasicAuthorization("login", "pass"))
                        .await()

            assertNotNull(githubUser)
            assertEquals("name", githubUser.name)
            assertEquals("url", githubUser.repos_url)
        }
    }
}
```

# Тест на корутины

```
@Test
fun login() {
…
    runBlocking {
        val githubUser =
            apiClientImpl
                .login(BasicAuthorization("login", "pass"))
                .await()


        assertEquals("name", githubUser.name)
    }
}
```

# Тест на корутины

```
@Test
fun login() {
…
    runBlocking {
        val githubUser =
            apiClientImpl
                .login(BasicAuthorization("login", "pass"))
                .await()


        assertEquals("name", githubUser.name)
    }
}
```

# Тесты пока выглядят одинаково

# А можно лучше?

```
interface SuspendingApiClient {


    suspend fun login(auth: Authorization)
        : GithubUser

    suspend fun getRepositories(reposUrl: String, auth: Authorization)
        : List<GithubRepository>

    suspend fun searchRepositories(searchQuery: String)
        : List<GithubRepository>


}
```

```
class SuspendingApiClientImpl : SuspendingApiClient {

    override suspend fun searchRepositories(query: String)
            : List<GithubRepository> =

            get("https://api.github.com/search/repositories?q=${query}")
            .jsonObject
            .getJSONArray("items")
            .toRepos()
}
```

```
private fun attemptLoginSuspending() {
    val apiClient = SuspendingApiClient.SuspendingApiClientImpl()
    launch(UI) {
        showProgress(true)
        val auth = BasicAuthorization(login, pass)
        try {
            val userInfo = async { apiClient.login(auth) }.await()
            val repoUrl = userInfo!!.repos_url
            val list = async { apiClient.getRepositories(repoUrl, auth) }.await()
            showRepositories(this, list!!.map { it -> it.full_name })
        } catch (e: RuntimeException) {
            showToast("Oops!")
        } finally {
            showProgress(false)
        }
    }
}
```

```
@Test
fun login() = runBlocking {
    val apiClientImpl = SuspendingApiClient.SuspendingApiClientImpl()
    val genericResponse = mockLoginResponse()

    staticMockk("khttp.KHttp").use {
        every { get("https://api.github.com/user", auth = any()) }
            returns genericResponse

        val githubUser =
                apiClientImpl
                    .login(BasicAuthorization("login", "pass"))

        assertNotNull(githubUser)
        assertEquals("name", githubUser.name)
        assertEquals("url", githubUser.repos_url)
    }
}
```

```
@Test
fun login() = runBlocking {
    val apiClientImpl = SuspendingApiClient.SuspendingApiClientImpl()
    val genericResponse = mockLoginResponse()

    staticMockk("khttp.KHttp").use {
        every { get("https://api.github.com/user", auth = any()) }
            returns genericResponse

        val githubUser =
                apiClientImpl
                    .login(BasicAuthorization("login", "pass"))

        assertNotNull(githubUser)
        assertEquals("name", githubUser.name)
        assertEquals("url", githubUser.repos_url)
    }
}
```

```
@Test
fun testLogin() = runBlocking {
    val apiClient = mockk<SuspendingApiClient.SuspendingApiClientImpl>()
    val githubUser = GithubUser("login", 1, "url", "name")
    val repositories = GithubRepository(1, "repos_name", "full_repos_name")

    coEvery { apiClient.login(any()) }
        returns githubUser
    coEvery { apiClient.getRepositories(any(), any()) }
        returns Arrays.asList(repositories)

    val loginPresenterImpl = SuspendingLoginPresenterImpl(apiClient, CommonPool)
    runBlocking {
        val repos = loginPresenterImpl.doLogin("login", "password")
        assertNotNull(repos)
    }
}
```

```
@Test
fun testLogin() = runBlocking {
    val apiClient = mockk<SuspendingApiClient.SuspendingApiClientImpl>()
    val githubUser = GithubUser("login", 1, "url", "name")
    val repositories = GithubRepository(1, "repos_name", "full_repos_name")

    coEvery { apiClient.login(any()) }
        returns githubUser
    coEvery { apiClient.getRepositories(any(), any()) }
        returns Arrays.asList(repositories)

    val loginPresenterImpl = SuspendingLoginPresenterImpl(apiClient, CommonPool)
    runBlocking {
        val repos = loginPresenterImpl.doLogin("login", "password")
        assertNotNull(repos)
    }
}
```

```
@Test
fun testLogin() = runBlocking {
    val apiClient = mockk<SuspendingApiClient.SuspendingApiClientImpl>()
    val githubUser = GithubUser("login", 1, "url", "name")
    val repositories = GithubRepository(1, "repos_name", "full_repos_name")

    coEvery { apiClient.login(any()) }
        returns githubUser
    coEvery { apiClient.getRepositories(any(), any()) }
        returns Arrays.asList(repositories)

    val loginPresenterImpl = SuspendingLoginPresenterImpl(apiClient, CommonPool)
    runBlocking {
        val repos = loginPresenterImpl.doLogin("login", "password")
        assertNotNull(repos)
    }
}
```

# Mockk

```
coEvery {
    apiClient.login(any())
} returns githubUser
```

# Mockito-kotlin

```
given {
    runBlocking {
        apiClient.login(any())
    }
}.willReturn (githubUser)
```

# Промежуточные итоги

→ Стектрейс стал меньше, но все равно неясный

→ Объектов под капотом создается меньше

→ Код проще писать

→ Интерфейс чище

→ Тесты чище

# Лирическое отступление

# Обработка ошибок

# Обработка ошибок

→ По умолчанию исключение в корутине роняет приложение

→ Если это нежелательно, то нужно менять контекст

→ Если ошибки разные, их много и они на разных слоях, то нужно менять контекст

# CoroutineExceptionHandler

```
val errorHandler = CoroutineExceptionHandler { _, e ->
    // Handle the exception gracefully
}
```

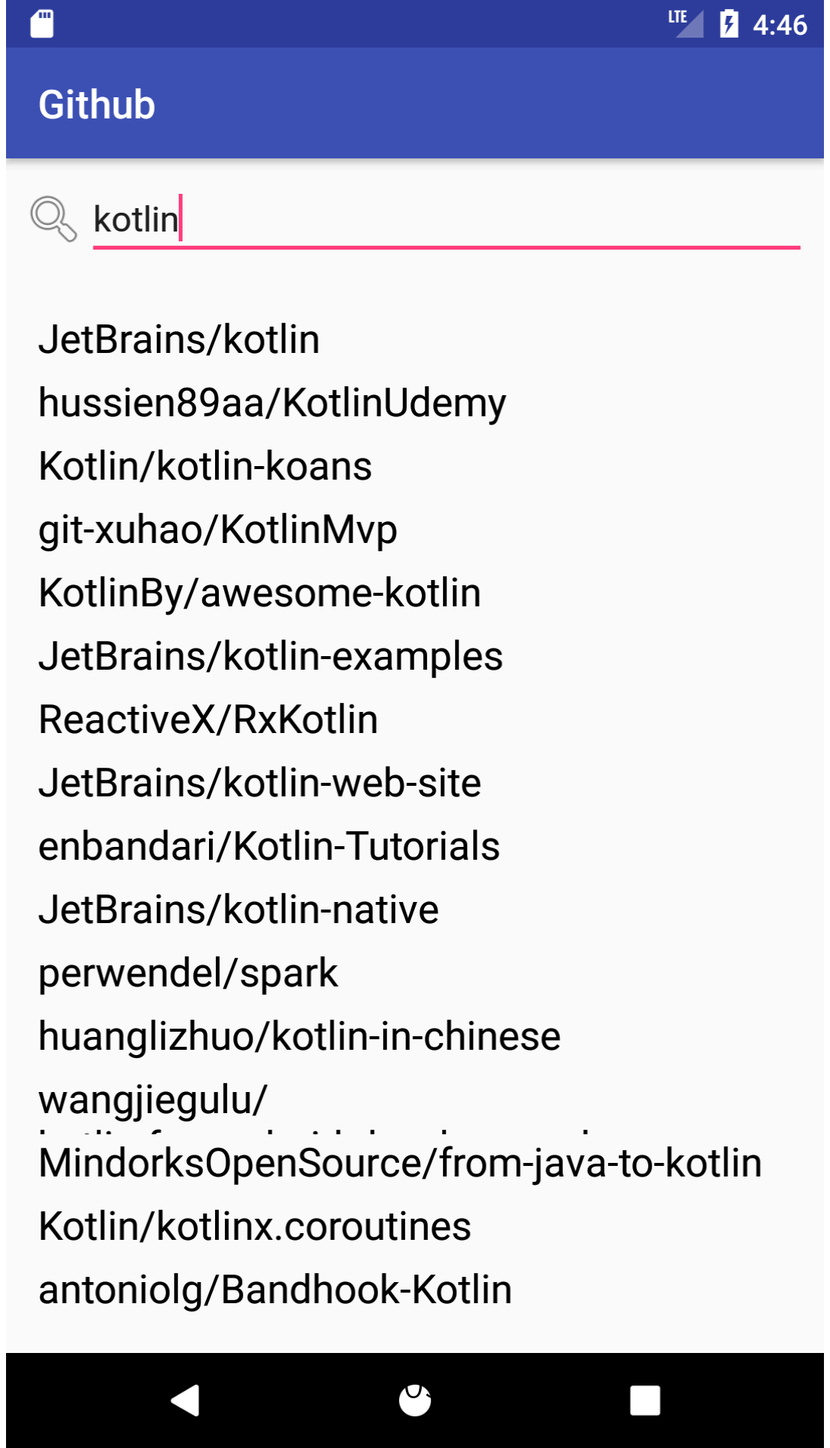
# CoroutineExceptionHandler

```
val errorHandler = CoroutineExceptionHandler { _, e ->
    // Handle the exception gracefully
}

val context = UI + errorHandler

launch(context) {
    …
}
```

# Зачем нужна RxJava?


4:46
Github
kotlin
JetBrains/kotlin
hussien89aa/KotlinUdemy
Kotlin/kotlin-koans
git-xuhao/KotlinMvp
KotlinBy/awesome-kotlin
JetBrains/kotlin-examples
ReactiveX/RxKotlin
JetBrains/kotlin-web-site
enbandari/Kotlin-Tutorials
JetBrains/kotlin-native
perwendel/spark
huanglizhuo/kotlin-in-chinese
wangjiegulu/
MindorksOpenSource/from-java-to-kotlin
Kotlin/kotlinx.coroutines
antoniolg/Bandhook-Kotlin

# Реализация поиска с RxJava 2

```
publishSubject
    .debounce(300, TimeUnit.MILLISECONDS)
    .distinctUntilChanged()
    .switchMap {
        searchQuery -> apiClientRxImpl.searchRepositories(searchQuery)
    }
    .subscribeOn(Schedulers.io())
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe({
        repos.adapter = ReposAdapter(it.map { it.full_name }, this)
    })
```

# Реализация поиска с RxJava 2

```
publishSubject
    .debounce(300, TimeUnit.MILLISECONDS)
    .distinctUntilChanged()
    .switchMap {
        searchQuery -> apiClientRxImpl.searchRepositories(searchQuery)
    }
    .subscribeOn(Schedulers.io())
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe({
        repos.adapter = ReposAdapter(it.map { it.full_name }, this)
    })
```

# Реализация поиска с RxJava 2

```
publishSubject
    .debounce(300, TimeUnit.MILLISECONDS)
    .distinctUntilChanged()
    .switchMap {
        searchQuery -> apiClientRxImpl.searchRepositories(searchQuery)
    }
    .subscribeOn(Schedulers.io())
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe({
        repos.adapter = ReposAdapter(it.map { it.full_name }, this)
    })
```

# Что классно?

# Реализация поиска с RxJava 2

```
publishSubject
    .debounce(300, TimeUnit.MILLISECONDS)
    .distinctUntilChanged()
    .switchMap {
        searchQuery -> apiClientRxImpl.searchRepositories(searchQuery)
    }
    .subscribeOn(Schedulers.io())
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe({
        repos.adapter = ReposAdapter(
            it.map { it.full_name },
            this@RepositoriesActivity)
    })
```

# Корутины тоже могут...

# ...с каналами

# Реализация поиска с каналами

```
launch(UI) {
    broadcast.consumeEach {
        delay(300)
        val foundRepositories =
                apiClient.searchRepositories(it).await()
        repos.adapter = ReposAdapter(
                            foundRepositories.map { it.full_name },
                            this@RepositoriesActivity
        )
    }
}
```

# Реализация поиска с каналами

```
launch(UI) {
    broadcast.consumeEach {
        delay(300)
        val foundRepositories =
                apiClient.searchRepositories(it).await()
        repos.adapter = ReposAdapter(
                            foundRepositories.map { it.full_name },
                            this@RepositoriesActivity
        )
    }
}
```

# Реализация поиска с каналами

```
launch(UI) {
    broadcast.consumeEach {
        delay(300)
        val foundRepositories =
                apiClient.searchRepositories(it).await()
        repos.adapter = ReposAdapter(
                            foundRepositories.map { it.full_name },
                            this@RepositoriesActivity
        )
    }
}
```

# Что такое броадкаст?

```
val broadcast = ConflatedBroadcastChannel<String>()
```

```
val broadcast = ConflatedBroadcastChannel<String>()

searchQuery.addTextChangedListener(object: TextWatcher {

    override fun afterTextChanged(text: Editable?) {
            broadcast.offer(text.toString())
        }
    })
}
```

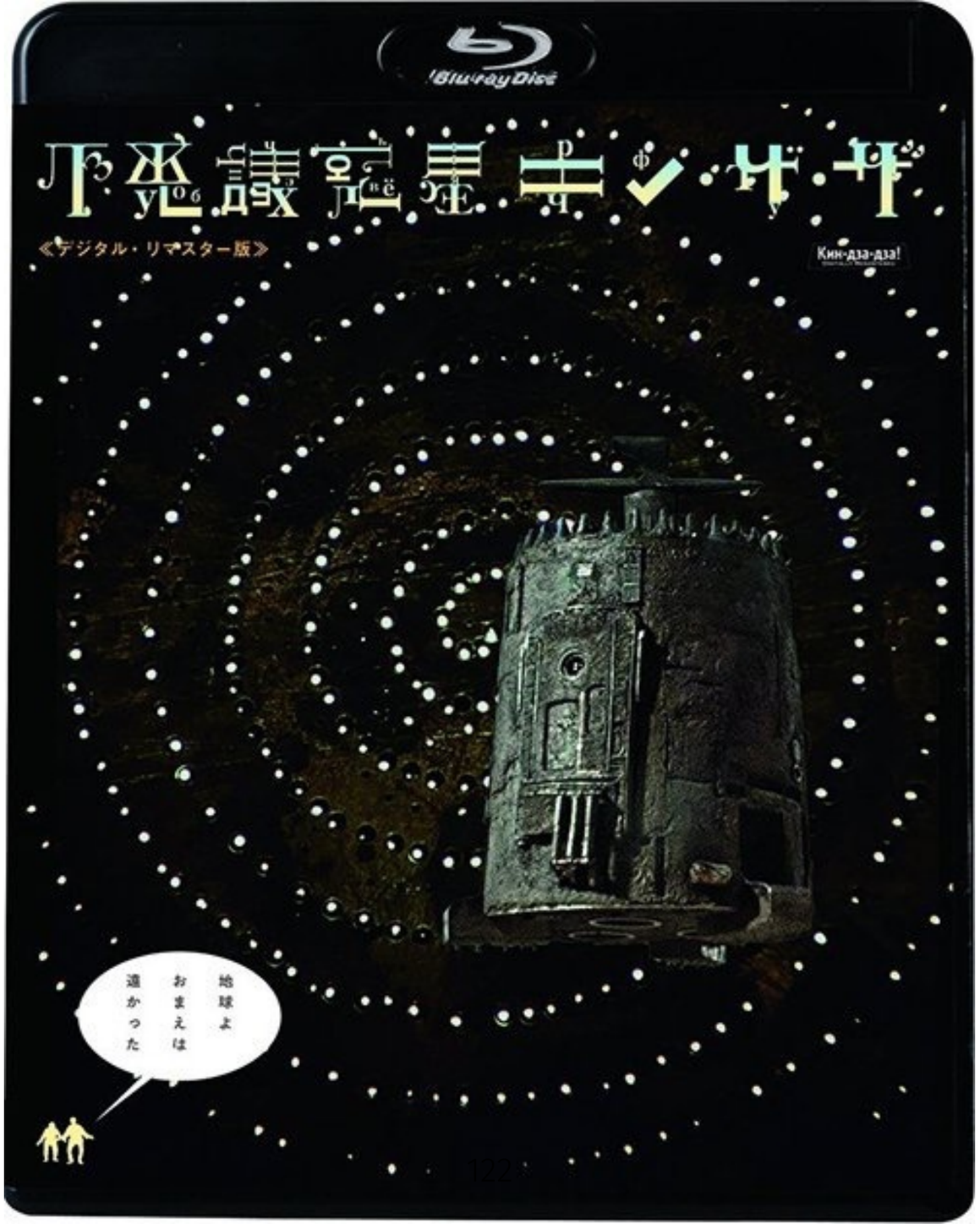
# Вопросы к этому коду

→ Что за каналы?

→ Что за BroadcastChannel?

→ Что за conflated ?

# Что за каналы?


Blu-ray Disc
《デジタル・リマスター版》
Кин-дза-дза!
地球よ
おまえは
遠かった

# Канал – блокирующая очередь

# но не совсем

| BlockingQueue | Channel |
| --- | --- |
| put | send |
| take | receive |

```
public suspend fun send(element: E)

public suspend fun receive(): E
```

# BroadcastChannel?

# BroadcastChannel - Subject

# но не совсем

| Subject | BroadcastChannel |
| --- | --- |
| `Subject<T> extends Observable<T> implements Observer<T>` | `BroadcastChannel<E> : SendChannel<E>` |
| - | `public fun openSubscription(): SubscriptionReceiveChannel<E>` |

# Что такое conflated channel?

# BroadcastChannel, но с потерей элементов

# Неубедительно! Скрипач нужен!

# Неубедительно! Скрипач нужен!?[1]

[1] конечно, нужен, вы же не можете заменить огромную либу одной фичей языка

# Kotlin корутины на самом деле поддерживают Rx...

# c kotlinx-coroutines-rx2

| Name | Result | Scope | Description |
| --- | --- | --- | --- |
| rxCompletable | Completable | CoroutineScope | Cold completable that starts coroutine on subscribe |
| rxMaybe | Maybe | CoroutineScope | Cold maybe that starts coroutine on subscribe |
| rxSingle | Single | CoroutineScope | Cold single that starts coroutine on subscribe |
| rxObservable | Observable | ProducerScope | Cold observable that starts coroutine on subscribe |
| rxFlowable | Flowable | ProducerScope | Cold observable that starts coroutine on subscribe with backpressure support |

| Name | Description |
| --- | --- |
| Job.asCompletable | Converts job to hot completable |
| Deferred.asSingle | Converts deferred value to hot single |
| ReceiveChannel.asObservable | Converts streaming channel to hot observable |
| Scheduler.asCoroutineDispatcher | Converts scheduler to CoroutineDispatcher |

# Где применимы конвертеры?

→ Использование библиотек с Rx-адаптером

→ Написание библиотек на корутиных для использования в приложениях, построенных на Rx

# Links

→ https://github.com/vlivanov/github-kotlin-coroutines

→ https://github.com/oleksiyp/mockk

→ http://khttp.readthedocs.io 💻

→ https://twitter.com/vvsevolodovich 🐦

→ https://medium.com/@dzigorium