RxJava не нужна: отказываемся от Rx в пользу корутин в Котлине

Владимир Иванов Ведущий разработчик EPAM Systems

Forget RxJava: Introducing Kotlin couroutines

Disclaimer

Everything said here is just a product of production and research experience; there could be mistakes, inaccurate statements, or just fallacies. Check everything by yourself.

About me

About me

— Vladimir Ivanov - Lead Software Engineer in EPAM

About me

— Vladimir Ivanov - Lead Software Engineer in EPAM

— Android apps: > 15 published, > 7 years

About me

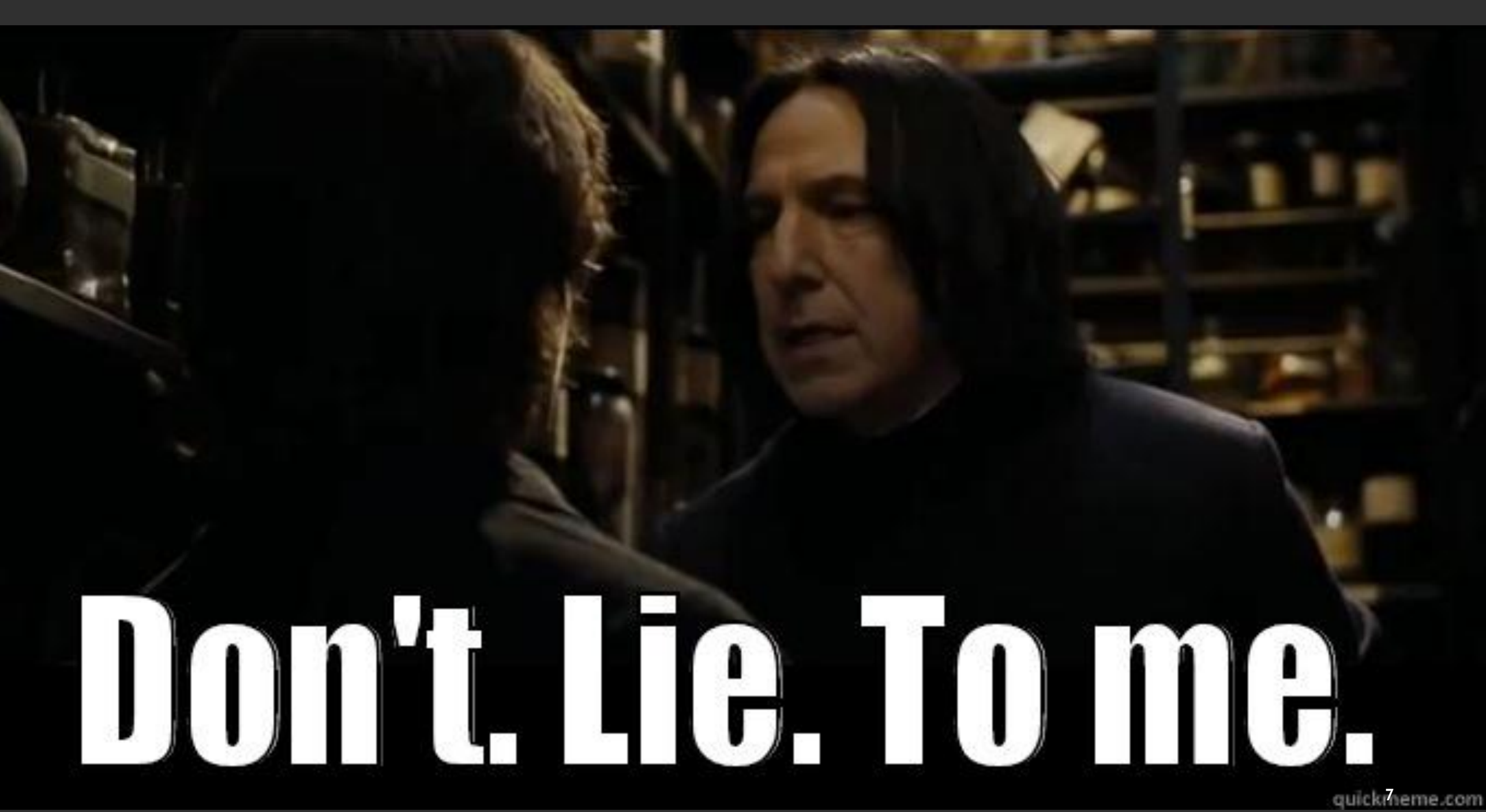
— Vladimir Ivanov - Lead Software Engineer in EPAM

— Android apps: > 15 published, > 7 years

— Wide expertise in Mobile

.NET?

JS?


Don't. Lie. To me.
quickmeme.com

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}` }
    }
}
```

Easy to read?

Kotlin can do this too!

What do we do now?

Github login application

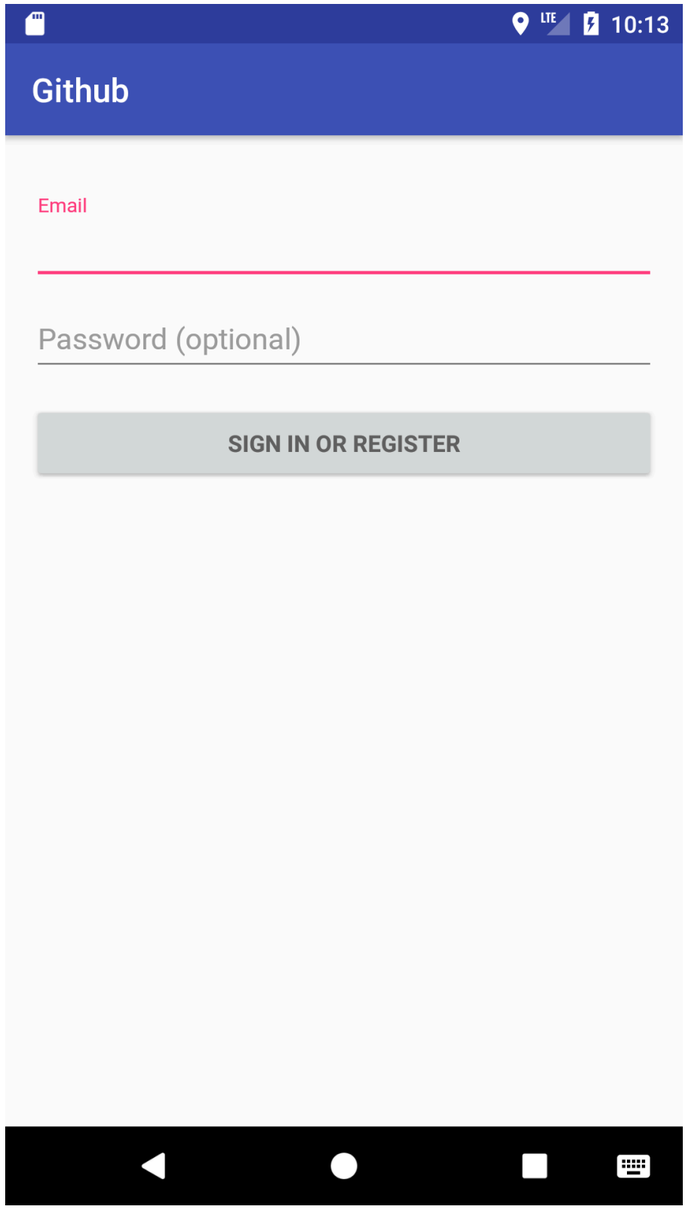
Github login application

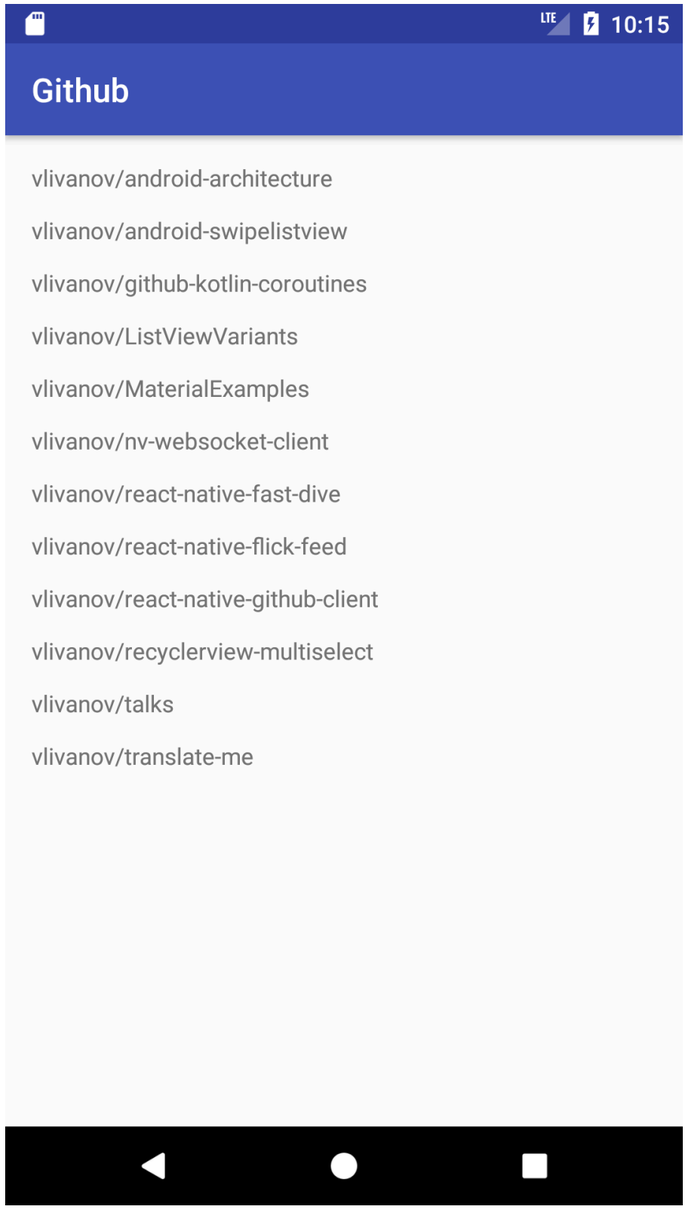
— Login to github

Github login application

— Login to github

— Get user’s repositories


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


10:15
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 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@LoginActivity, 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@LoginActivity, 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@LoginActivity, 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@LoginActivity, list)    },
                    { error -> Log.e("TAG", "Failed to show repos", error) }
            )
}
```

Problems?

Problems?

Problems?

— A lot of intermediate objects created under the hood

Problems?

— A lot of intermediate objects created under the hood

— Unrelated stacktrace

Problems?

— A lot of intermediate objects created under the hood
— Unrelated stacktrace
— The learning curve for new developers is steppy


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@LoginActivity, 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)
```

Let’s see if Kotlin coroutines can help here

Coroutines implementation

```
interface ApiClient {

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

}
```

Coroutines implementation

```
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@LoginActivity,
                list.map { it -> it.full_name }
            )
        } catch (e: RuntimeException) {
            Toast.makeText(this@LoginActivity, e.message, LENGTH_LONG).show()
        } finally {
            showProgress(false)
        }
    }
}
```

Pluses

Pluses

— Easy to read(as js), because

Pluses

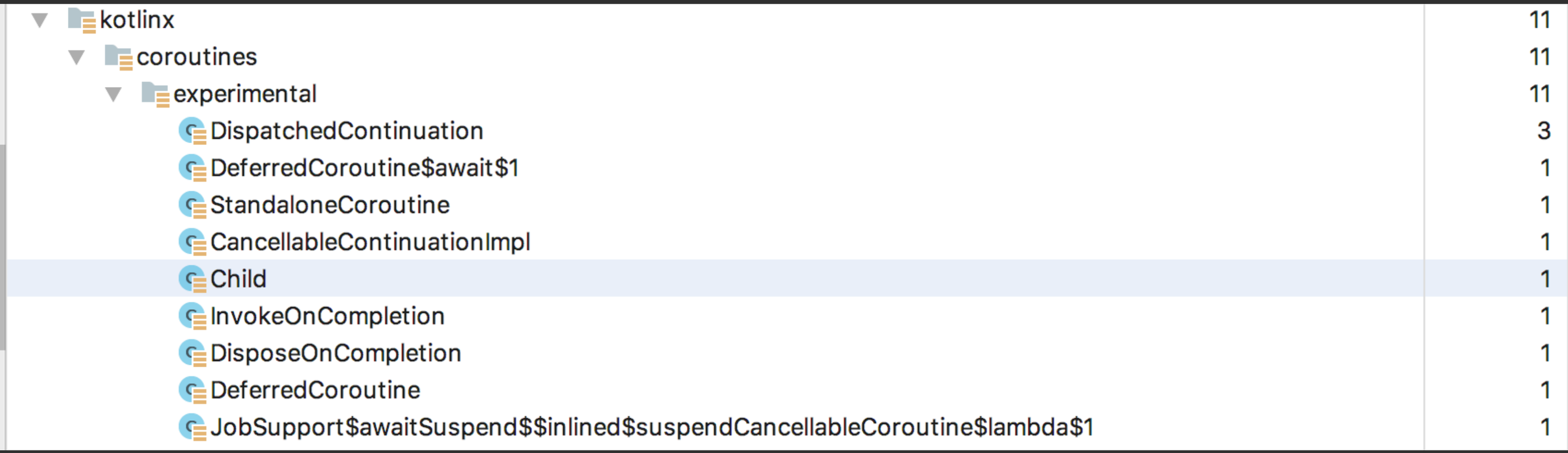
— Easy to read(as js), because
— Code is async, but written as sync

Pluses

— Easy to read(as js), because
— Code is async, but written as sync
— Error handling like for sync code(try-catch-finally)

What about minuses of RxJava?

Allocations?


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

Stacktraces?

```
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)

Sync-styled

```
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@LoginActivity, list.map { it -> it.full_name })
        } catch (e: RuntimeException) {
            Toast.makeText(this@LoginActivity, e.message, LENGTH_LONG).show()
        } finally {
            showProgress(false)
        }
    }
}
```

Error handling using language(not library methods)

```
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@LoginActivity, list.map { it -> it.full_name })
        } catch (e: RuntimeException) {
            Toast.makeText(this@LoginActivity, e.message, LENGTH_LONG).show()
        } finally {
            showProgress(false)
        }
    }
}
```

Several async calls look sequential

```
launch(UI) {
    showProgress(true)
    val auth = BasicAuthorization(login, pass)
    try {
        val userInfo = login(auth).await()
        val repoUrl = userInfo!!.repos_url
        val repos = getRepositories(repoUrl, auth).await()
        val pullRequests = getPullRequests(repos![0], auth).await()
        showRepositories(this@LoginActivity, repos!!.map { it -> it.full_name })
    } catch (e: RuntimeException) {
        Toast.makeText(this@LoginActivity, e.message, LENGTH_LONG).show()
    } finally {
        showProgress(false)
    }
}
```

Async function implementation

```
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"))
    }
}
```

Async function implementation

```
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"))
    }
}
```

Using Async HOF

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

        return@async GithubUser(...)
}
```

What is Async?

Coroutine builders

Coroutine builders

— launch

Coroutine builders

— launch

— async

Coroutine builders

— launch

— async

— runBlocking

Coroutine builders

— launch

— async

— runBlocking

— withContext

Launch returns 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 returns Deferred<T>

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

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

Deferred is Future

Deferred is Future

STEREO ● ● LS 86016

LIMELIGHT

LES McCANN LTD.

BUT NOT REALLY

Deferred is Future

Deferred is Future

— Non-blocking

Deferred is Future

— Non-blocking

— Cancellable

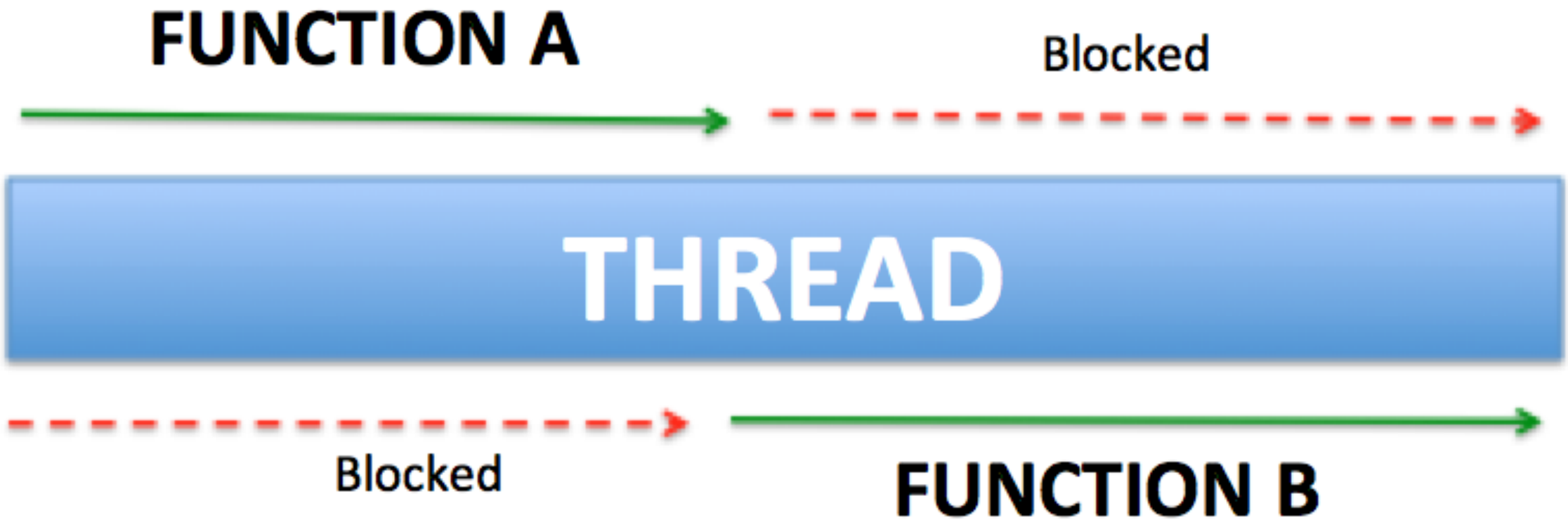
Await - extension function

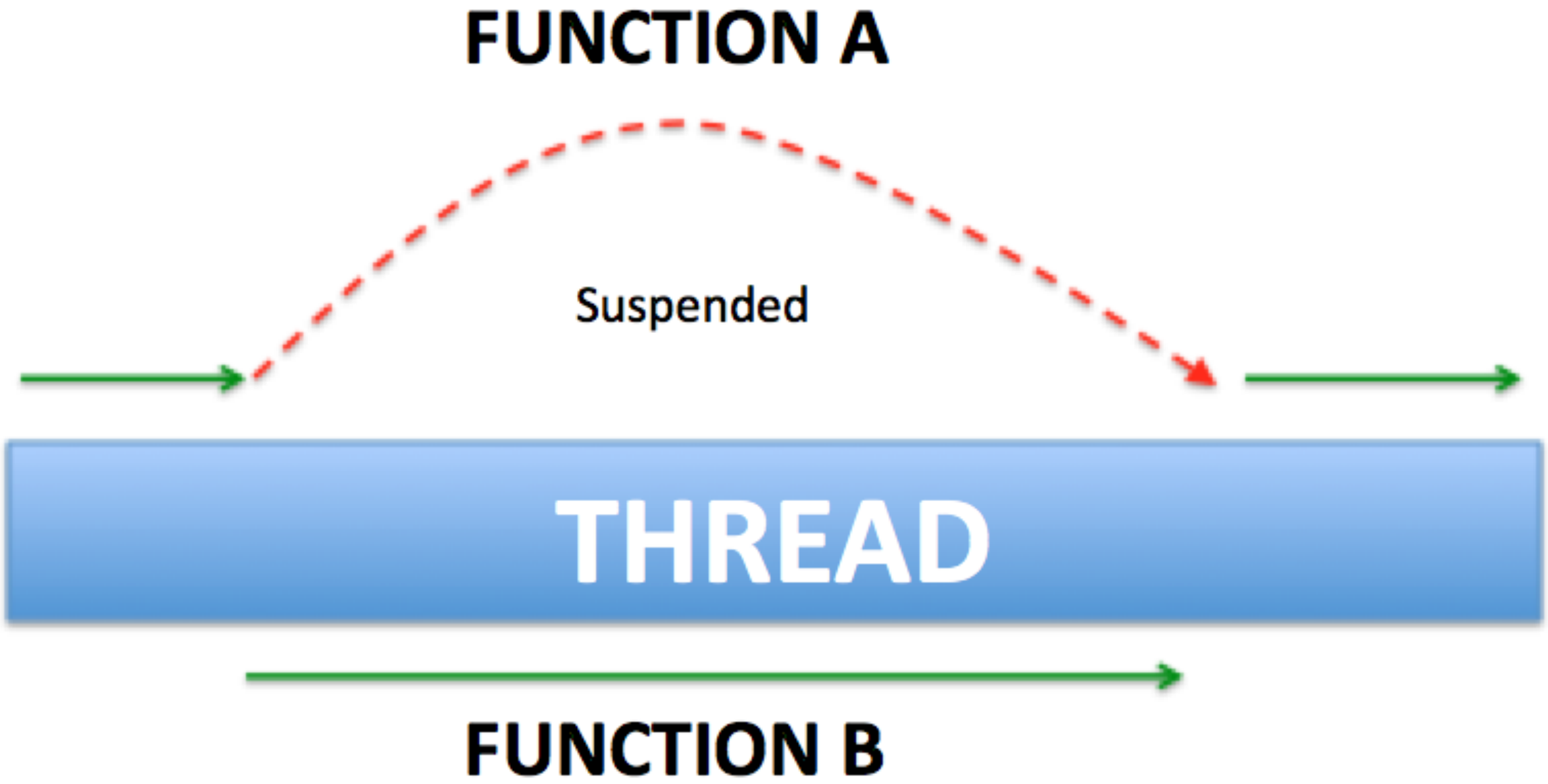
Await - extension function

Await - extension function

— Is like Future.get() but suspends instead of blocking

Suspension


FUNCTION A
Blocked
THREAD
Blocked
FUNCTION B


FUNCTION A
Suspended
THREAD
FUNCTION B

Suspending

Suspending

— means pause of executing

Suspending

— means pause of executing

— which means ability to resume

Suspending

— means pause of executing

— which means ability to resume

— But suspend may happen only in predefined places

Suspending

— means pause of executing

— which means ability to resume

— But suspend may happen only in predefined places

— When calling functions with `suspend` modifier!

Where our 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>
```

Cancellation

Thread.stop()

~~Thread.stop()~~

Cancellation is always cooperative

RxJava 2

Cancel in RxJava requires access to current subscription

How to check if coroutine gets cancelled?

```
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
        }
    …
}
```

Let’s write tests!

```
@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 ->
            Assert.assertNotNull(githubUser)
            Assert.assertEquals("name", githubUser.name)
            Assert.assertEquals("url", githubUser.repos_url)
        })

    }
}
```

```
@Test
fun login() {
    …

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

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

    …
}
```

```
@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)
        }
    }
}
```

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


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

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


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

So tests are pretty much the same

What if Coroutines can simplify our interface even more?

```
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@LoginActivity, list!!.map { it -> it.full_name })
        } catch (e: RuntimeException) {
            Toast.makeText(this@LoginActivity, e.message, LENGTH_LONG).show()
        } 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)
    }
}
```

Short summary

Short summary

— Shorter stacktrace, but still unclear

Short summary

— Shorter stacktrace, but still unclear

— Less memory footprint

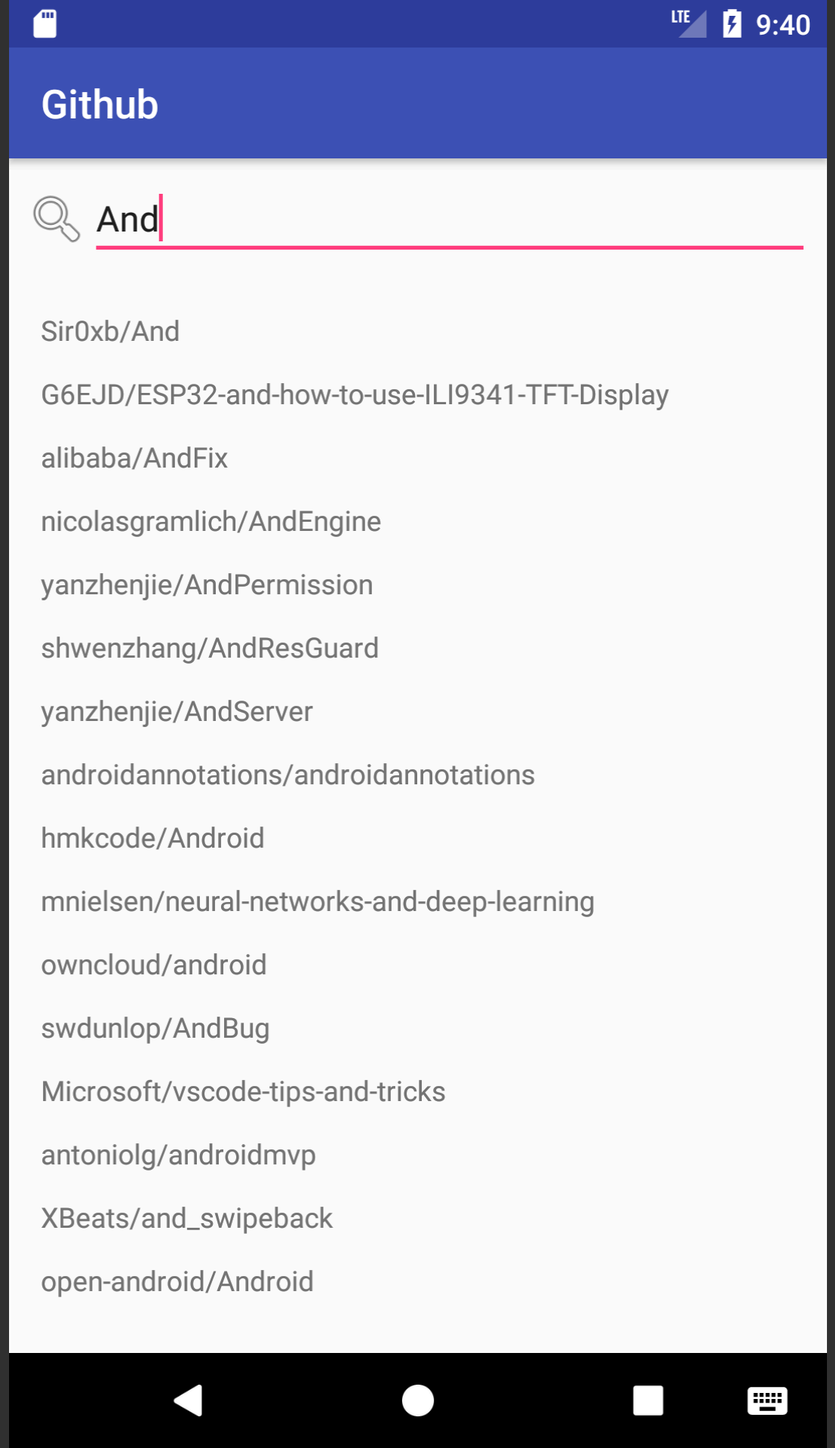
Short summary

— Shorter stacktrace, but still unclear
— Less memory footprint
— Code is more explicit, therefore easier to read and understand

Short summary

— Shorter stacktrace, but still unclear
— Less memory footprint
— Code is more explicit, therefore easier to read and understand
— Clean interfaces, clean tests(awesome!)

Why Rx in the first place?


9:40
Github
And
Sir0xb/And
G6EJD/ESP32-and-how-to-use-ILI9341-TFT-Display
alibaba/AndFix
nicolasgramlich/AndEngine
yanzhenjie/AndPermission
shwenzhang/AndResGuard
yanzhenjie/AndServer
androidannotations/androidannotations
hmkcode/Android
mnielsen/neural-networks-and-deep-learning
owncloud/android
swdunlop/AndBug
Microsoft/vscode-tips-and-tricks
antoniolg/androidmvp
XBeats/and_swipeback
open-android/Android

Search RxJava 2 implementation

```
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)
    })
```

Search RxJava 2 implementation

```
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)
    })
```

Search RxJava 2 implementation

```
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)
    })
```

What’s really awesome here?

Search RxJava 2 implementation

```
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)
    })
```

We can do the same or better in Kotlin Coroutines…

…with channels

Search with Channels implementation

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

Search with Channels implementation

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

Search with Channels implementation

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

What is broadcast?

Search with Channels implementation

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

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

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

searchQuery.addTextChangedListener(object: TextWatcher {

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

Questions

— What are channels?

— What are channels?

— What is a BroadcastChannel?

— What are channels?

— What is a BroadcastChannel?

— What is a conflated channel?

What are channels?

Channel is a blocking queue

STEREO ● ● LS 86016

LIMELIGHT

LES McCANN LTD.

BUT NOT REALLY

BlockingQueue	Channel
put	send
take	receive

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

public suspend fun receive(): E
```

What is a BroadcastChannel?

BroadcastChannel is Subject

STEREO ● ● LS 86016

LIMELIGHT

LES McCANN LTD.

BUT NOT REALLY

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

What is a conflated channel?

ConflatedBroadcastChannel is a BroadcastChannel…

…but previuosly sent elements are lost

Not quite persuasively…

What if I still need Rx?

What if I still need Rx?[1]

[1] And you actually will, because you can’t compare a language feature with a shittone library

Kotlin coroutines actually supports Rx…

with 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

Links

Links

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

Links

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

— https://github.com/vlivanov/github-kotlin-coroutines 💻

Links

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

— https://github.com/vlivanov/github-kotlin-coroutines 💻

— https://twitter.com/vvsevolodovich 🐦

Links

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

— https://github.com/vlivanov/github-kotlin-coroutines 💻

— https://twitter.com/vvsevolodovich 🐦

— https://medium.com/@dzigorium 

Вопросы?

@vvsevolodovich

https://medium.com/@dzigorium

vladimir.vs.ivanov@gmail.com

Владимир Иванов Ведущий разработчик EPAM Systems