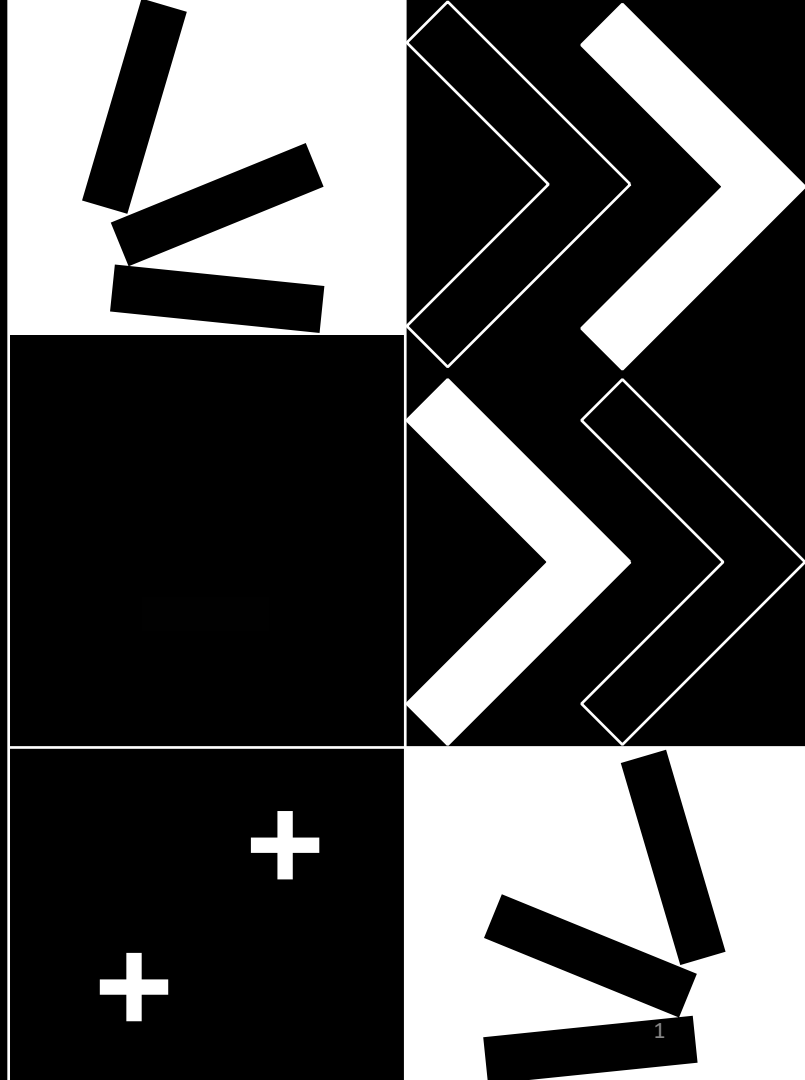
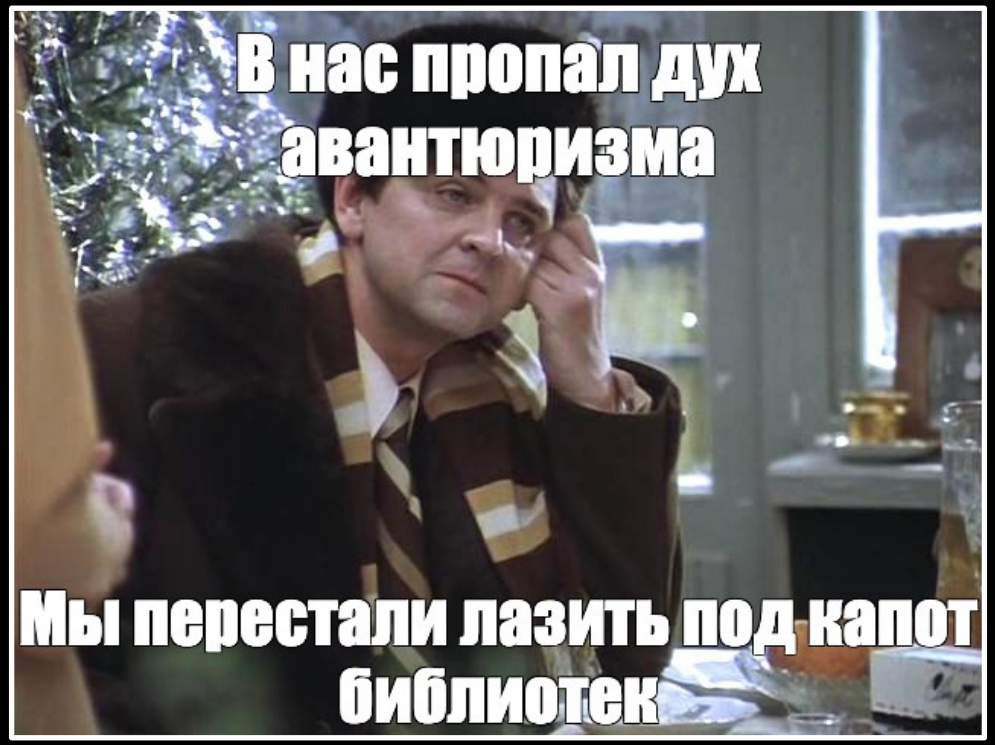
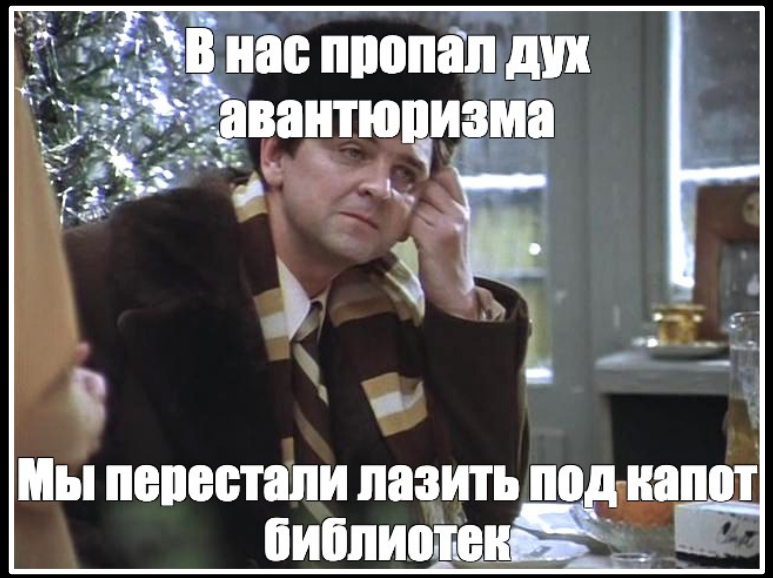
Непростые вопросы про Kotlin Coroutines

Абакар Магомедов, Альфа-Банк
Александр Гирев, Wildberries




Mobius


В нас пропал дух авантюризма
Мы перестали лазить под капот библиотек


В нас пропал дух авантюризма
Мы перестали лазить под капот библиотек

Как я представляю работу корутин внутри

Что будет в докладе

- Магия слова suspend
- Dispatchers, отмена корутин
- Синхронизация в корутинах
- Немножко о Structured concurrency

Спикеры

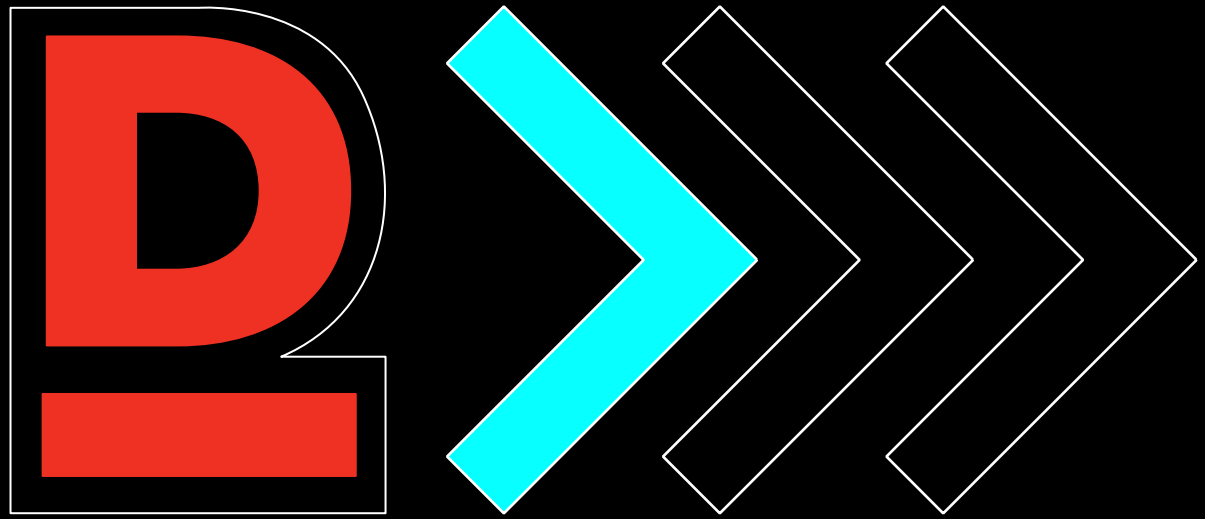
Абакар Магомедов

- Главный Android Techlead, Alfa Bank
- Пытаюсь писать статьи на Хабре
- Участвовал в ПК Android Podlodka Crew
- https://www.youtube.com/@a6ocha/videos

Александр Гирев

- Android developer, Wildberries
- Я на Хабре: @Ales_Ivanov

myQuiz



Магия слова suspend

Правила

- Синие функции возвращают значения, красные — нет, а вместо этого дергают колбэки.

- Вы не можете вызвать красную функцию из синей, потому что вы не сможете определить результат, пока красная функция не завершится позже.

https://journal.stuffwithstuff.com/2015/02/01/what-color-is-your-function/

function foo()

function bar()

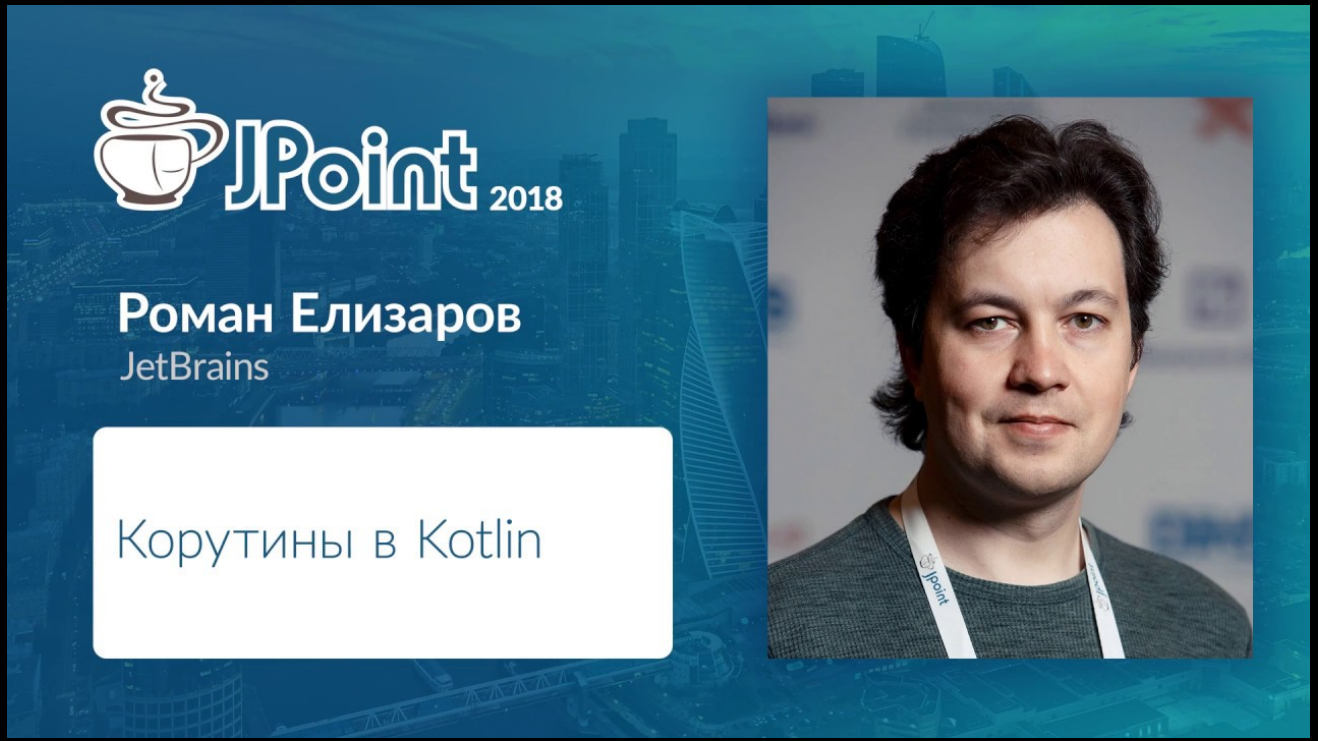
Существует ли «аналог» Kotlin Coroutines в Java? Если да, то как он называется?

Project Loom

Project Loom

```
1   for (int i = 0; i < 1000000; i++) {
2         // good, old Java Threads
3         new Thread( () -> getURL("mobius"))
4         .start();
5     }
6
7
8   for (int i = 0; i < 1000000; i++) {
9         // Java 19 virtual threads to the rescue?
10        Thread.startVirtualThread(() -> getURL("mobius"))
11        .start();
12    }
```

О корутинах из первых уст



Основные концепции корутин

- асинхронный код в синхронном стиле

```
fun postItem(item: Item) {
    requestTokenAsync { token ->
        createPostAsync(token, item) { post ->
            processPost(post)
        }
    }
}
```

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

Основные концепции корутин

- асинхронный код в синхронном стиле
- интерфейс Continuation

Kotlin

```
suspend fun createPost(token: Token, item: Item): Post { … }
```

Java/JVM

```
Object createPost(Token token, Item item, Continuation<Post> cont) { … }
```

callback

Основные концепции корутин

- асинхронный код в синхронном стиле
- интерфейс Continuation
- state machine

Kotlin

```
val token = requestToken()
val post = createPost(token, item)
processPost(post)
```

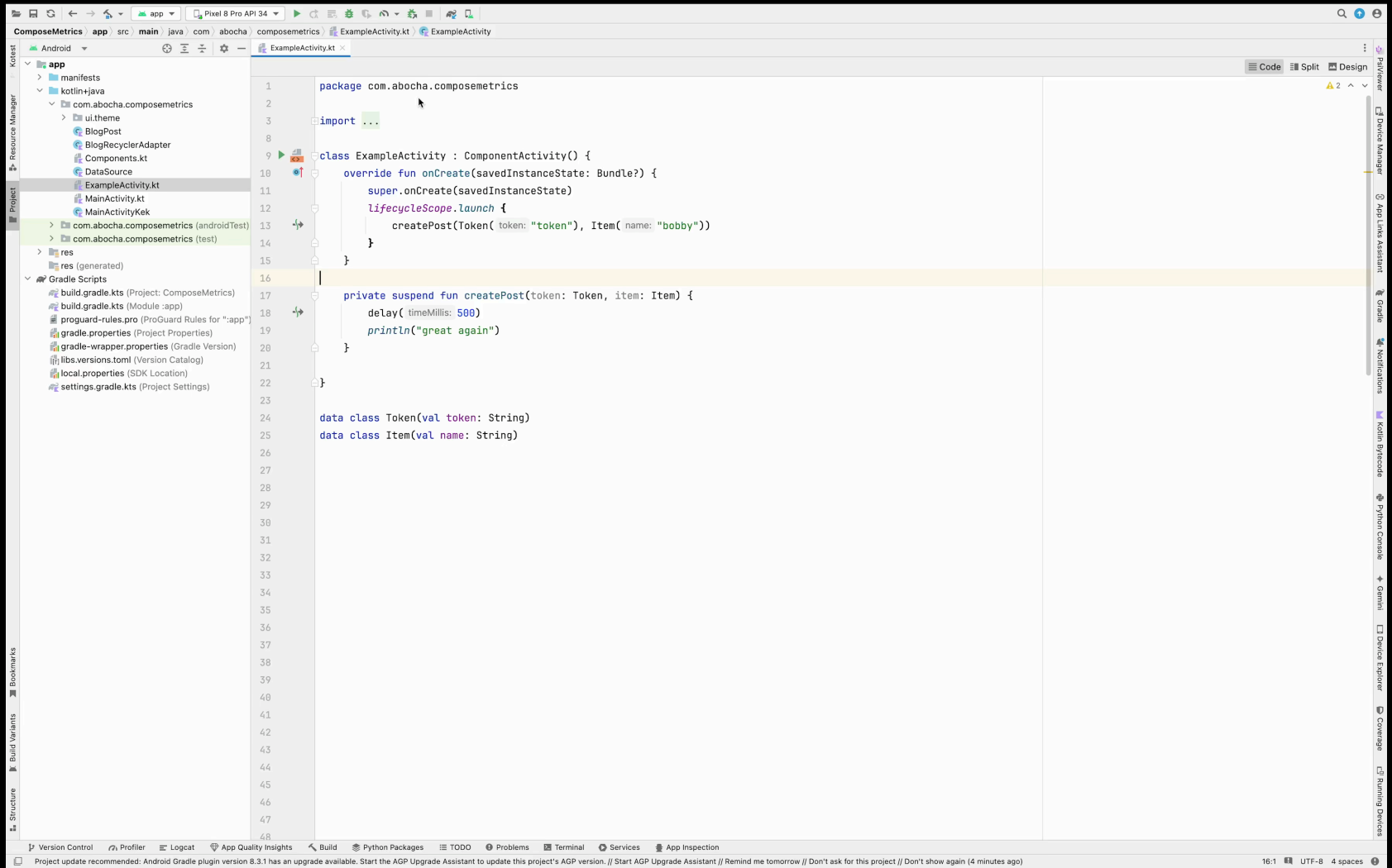
Compiles to *state machine* (simplified code shown)

Java/JVM

```
switch (cont.label) {
    case 0:
        cont.label = 1;
        requestToken(cont);
        break;
    case 1:
        Token token = (Token) prevResult;
        cont.label = 2;
        createPost(token, item, cont);
        break;
    case 2:
        Post post = (Post) prevResult;
        processPost(post);
        break;
}
```

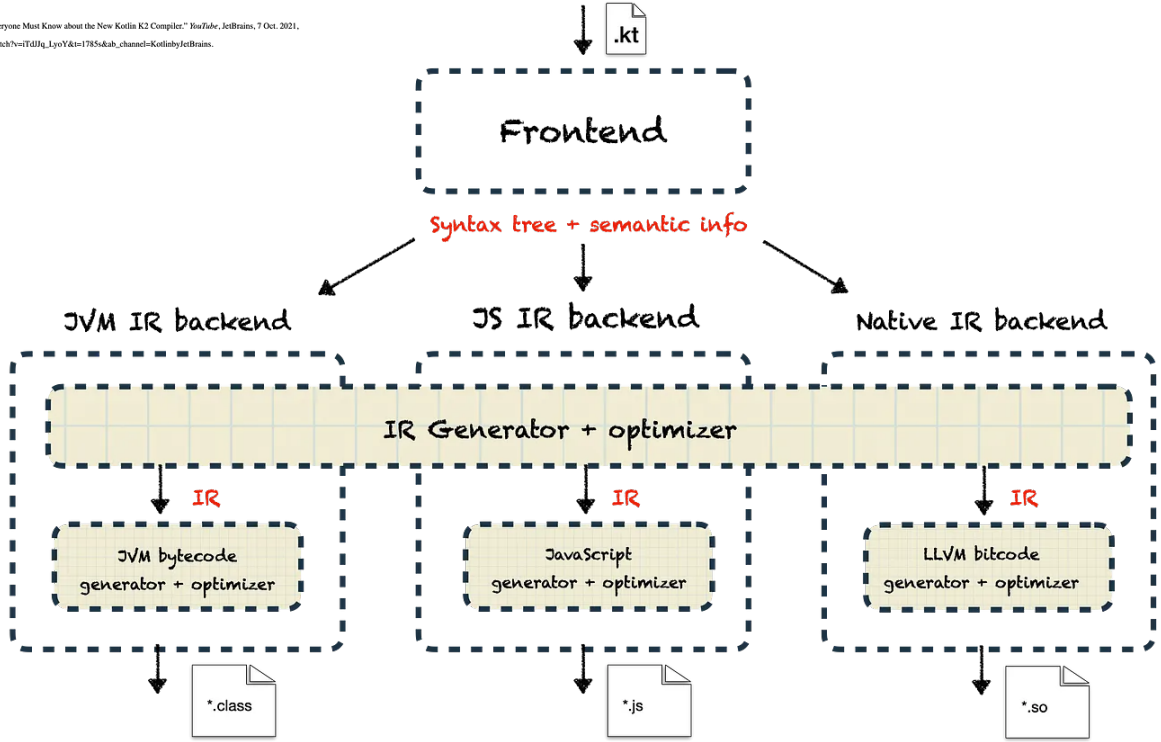
```
class ExampleActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        lifecycleScope.launch {
            createPost(Token( token: "token"), Item( name: "bobby"))
        }
    }

    private suspend fun createPost(token: Token, item: Item) {
        delay( timeMillis: 500)
        println("great again")
    }
}

data class Token(val token: String)
data class Item(val name: String)
```

```
private final Object createPost(Token var1, Item var2, Continuation var3) {
   Object $continuation;
   label20: {
      if (var3 instanceof <undefinedtype>) {
         $continuation = (<undefinedtype>)var3;
         if ((((<undefinedtype>)$continuation).label & Integer.MIN_VALUE) != 0) {
            ((<undefinedtype>)$continuation).label -= Integer.MIN_VALUE;
            break label20;
         }
      }

      $continuation = new ContinuationImpl(var3) {
         // $FF: synthetic field
         Object result;
         int label;

         @Nullable
         public final Object invokeSuspend(@NotNull Object $result) {
            this.result = $result;
            this.label |= Integer.MIN_VALUE;
            return ExampleActivity.this.createPost((Token)null, (Item)null, this);
         }
      };
   }

   Object $result = ((<undefinedtype>)$continuation).result;
   Object var7 = IntrinsicsKt.getCOROUTINE_SUSPENDED();
   switch (((<undefinedtype>)$continuation).label) {
      case 0:
         ResultKt.throwOnFailure($result);
         ((<undefinedtype>)$continuation).label = 1;
         if (DelayKt.delay(500L, (Continuation)$continuation) == var7) {
            return var7;
         }
         break;
      case 1:
         ResultKt.throwOnFailure($result);
         break;
      default:
         throw new IllegalStateException("call to 'resume' before 'invoke' with coroutine");
   }

   String var4 = "great again";
   System.out.println(var4);
   return Unit.INSTANCE;
}
}
```


ComposeMetrics app src main java com abocha composemetrics ExampleActivity.kt ExampleActivity
Android
app
manifests
kotlin+java
com.abocha.composemetrics
ui.theme
BlogPost
BlogRecyclerAdapter
Components.kt
DataSource
ExampleActivity.kt
MainActivity.kt
MainActivityKek
com.abocha.composemetrics (androidTest)
com.abocha.composemetrics (test)
res
res (generated)
Gradle Scripts
build.gradle.kts (Project: ComposeMetrics)
build.gradle.kts (Module :app)
proguard-rules.pro (ProGuard Rules for ":app")
gradle.properties (Project Properties)
gradle-wrapper.properties (Gradle Version)
libs.versions.toml (Version Catalog)
local.properties (SDK Location)
settings.gradle.kts (Project Settings)
ExampleActivity.kt
Code
Split
Design
package com.abocha.composemetrics
import ...
class ExampleActivity : ComponentActivity() {
override fun onCreate(savedInstanceState: Bundle?) {
super.onCreate(savedInstanceState)
lifecycleScope.launch {
createPost(Token(token: "token"), Item(name: "bobby"))
}
}
private suspend fun createPost(token: Token, item: Item) {
delay(timeMillis: 500)
println("great again")
}
}
data class Token(val token: String)
data class Item(val name: String)
Version Control
Profiler
Logcat
App Quality Insights
Build
Python Packages
TODO
Problems
Terminal
Services
App Inspection
Project update recommended: Android Gradle plugin version 8.3.1 has an upgrade available. Start the AGP Upgrade Assistant to update this project's AGP version. // Start AGP Upgrade Assistant // Remind me tomorrow // Don't ask for this project // Don't show again (4 minutes ago)
16:1
UTF-8
4 spaces

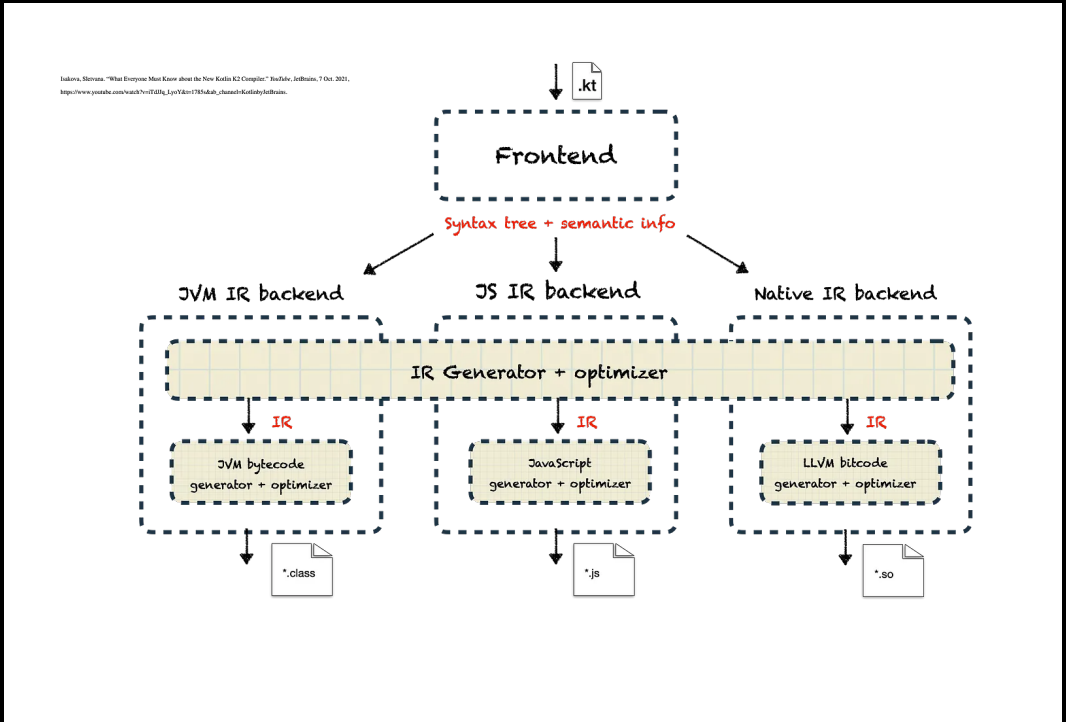
А как continuation появляется в suspend функции ?

Isakova, Svetvana. "What Everyone Must Know about the New Kotlin K2 Compiler." *YouTube*, JetBrains, 7 Oct. 2021, https://www.youtube.com/watch?v=iTdJJq_LyoY&t=1785s&ab_channel=KotlinbyJetBrains.

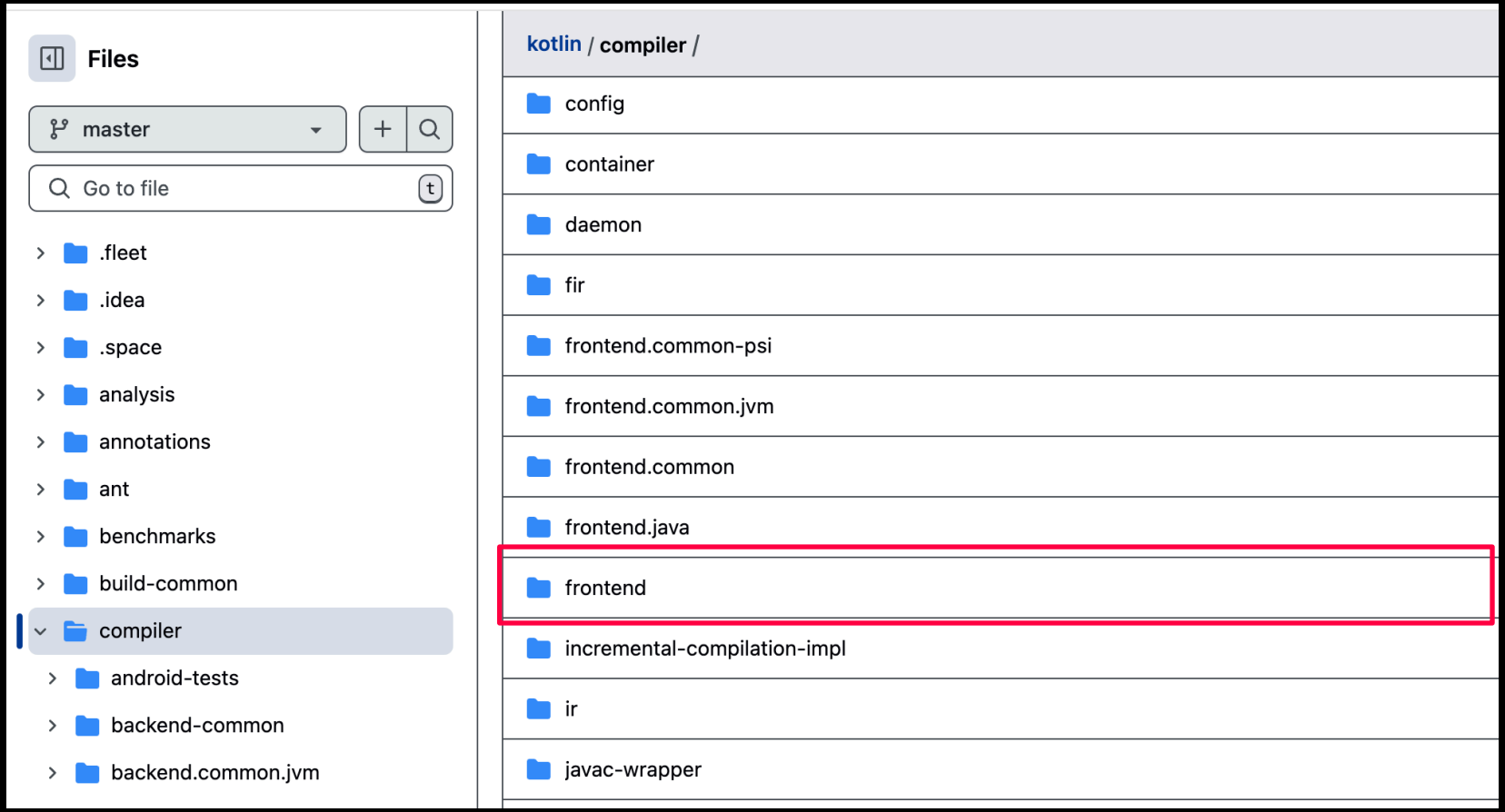


Scheme by Svetlana Isakova

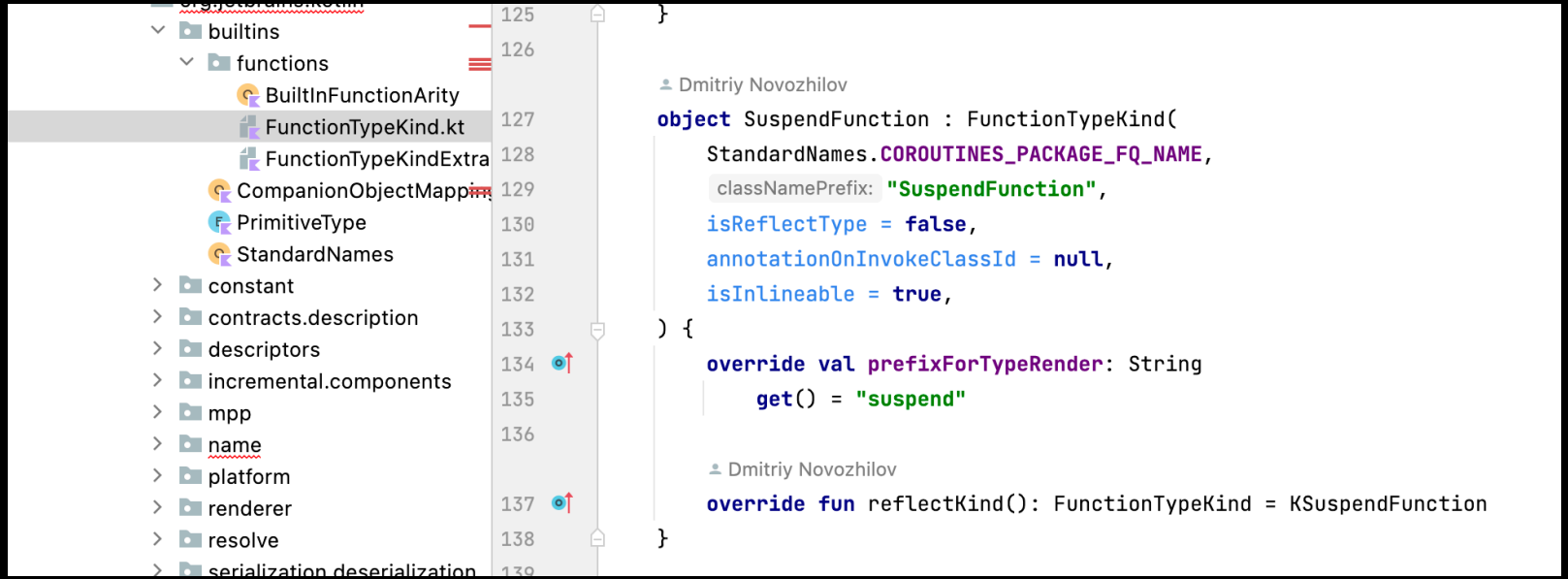
Crash Course on the Kotlin Compiler

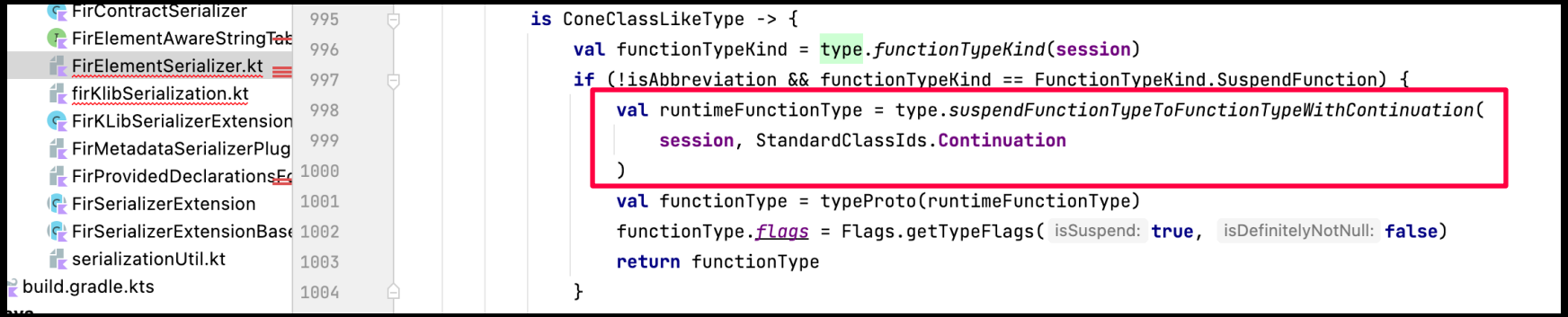


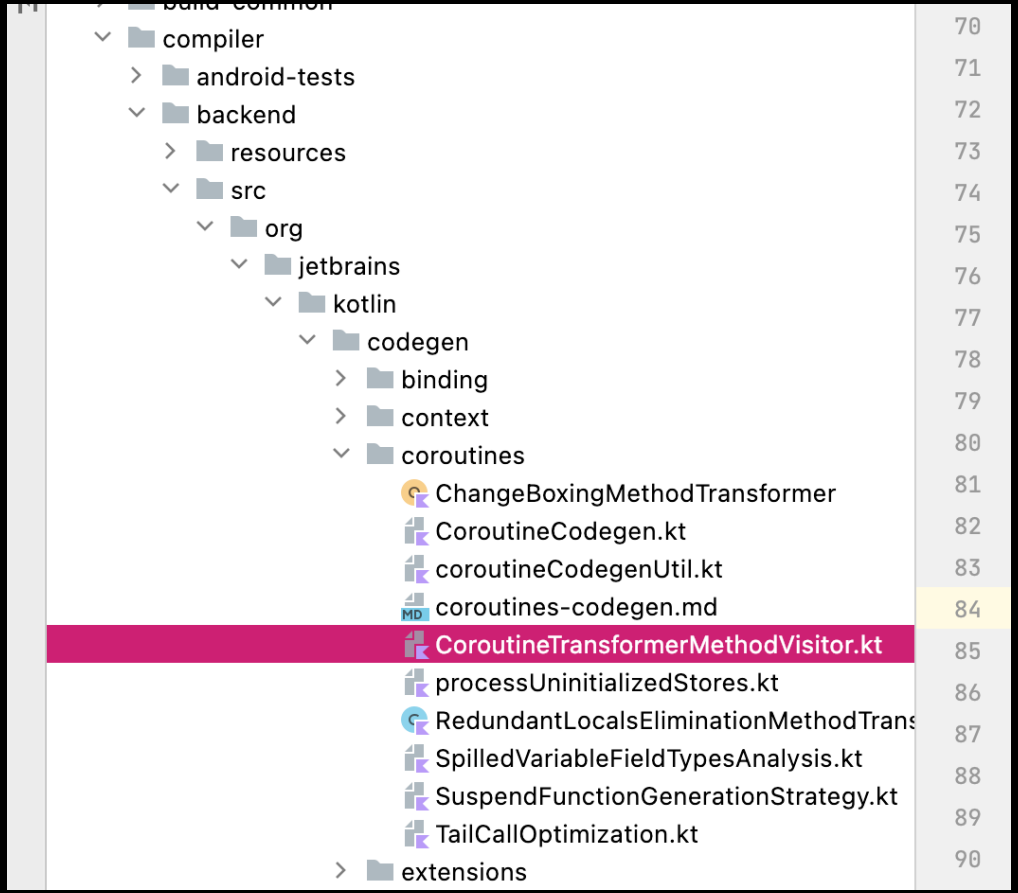
https://github.com/JetBrains/kotlin


Files
master
Go to file
t
.fleet
.idea
.space
analysis
annotations
ant
benchmarks
build-common
compiler
android-tests
backend-common
backend.common.jvm
kotlin / compiler /
config
container
daemon
fir
frontend.common-psi
frontend.common.jvm
frontend.common
frontend.java
frontend
incremental-compilation-impl
ir
javac-wrapper

```
        result = type(
            createFunctionType(
                moduleDescriptor.builtIns, annotations, receiverType, contextReceiversTypes,
                parameterDescriptors.map { it.type },
                parameterDescriptors.map { it.name },
                returnType,
                suspendFunction = hasSuspendModifier
            )
        )
    }
```


builtins
functions
BuiltInFunctionArity
FunctionTypeKind.kt
FunctionTypeKindExtra
CompanionObjectMappi
PrimitiveType
StandardNames
constant
contracts.description
descriptors
incremental.components
mpp
name
platform
renderer
resolve
serialization.deserialization
125 }
126
Dmitriy Novozhilov
127 object SuspendFunction : FunctionTypeKind(
128 StandardNames.COROUTINES_PACKAGE_FQ_NAME,
129 classNamePrefix: "SuspendFunction",
130 isReflectType = false,
131 annotationOnInvokeClassId = null,
132 isInlineable = true,
133) {
134 override val prefixForTypeRender: String
135 get() = "suspend"
136
Dmitriy Novozhilov
137 override fun reflectKind(): FunctionTypeKind = KSuspendFunction
138 }


FirContractSerializer
FirElementAwareStringTab
FirElementSerializer.kt
firKlibSerialization.kt
FirKLibSerializerExtension
FirMetadataSerializerPlug
FirProvidedDeclarationsEc
FirSerializerExtension
FirSerializerExtensionBase
serializationUtil.kt
build.gradle.kts
995 is ConeClassLikeType -> {
996 val functionTypeKind = type.functionTypeKind(session)
997 if (!isAbbreviation && functionTypeKind == FunctionTypeKind.SuspendFunction) {
998 val runtimeFunctionType = type.suspendFunctionTypeToFunctionTypeWithContinuation(
999 session, StandardClassIds.Continuation
1000)
1001 val functionType = typeProto(runtimeFunctionType)
1002 functionType.flags = Flags.getTypeFlags(isSuspend: true, isDefinitelyNotNull: false)
1003 return functionType
1004 }


compiler
android-tests
backend
resources
src
org
jetbrains
kotlin
codegen
binding
context
coroutines
ChangeBoxingMethodTransformer
CoroutineCodegen.kt
coroutineCodegenUtil.kt
coroutines-codegen.md
CoroutineTransformerMethodVisitor.kt
processUninitializedStores.kt
RedundantLocalsEliminationMethodTrans
SpilledVariableFieldTypesAnalysis.kt
SuspendFunctionGenerationStrategy.kt
TailCallOptimization.kt
extensions
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90

```
Ilmir Usmanov +10
class CoroutineTransformerMethodVisitor(
    delegate: MethodVisitor,
    access: Int,
    name: String,
    desc: String,
    signature: String?,
    exceptions: Array<out String>?,
    private val containingClassInternalName: String,
    obtainClassBuilderForCoroutineState: () -> ClassBuilder,
    private val isForNamedFunction: Boolean,
    // Since tail-call optimization of functions with Unit return type relies on ability of call-site to recognize them,
    // in order to ignore return value and push Unit, when we cannot ensure this ability, for example, when the function overrides function,
    // returning Any, we need to disable tail-call optimization for these functions.
    private val disableTailCallOptimizationForFunctionReturningUnit: Boolean,
    private val reportSuspensionPointInsideMonitor: (String) -> Unit,
    private val lineNumber: Int,
    private val sourceFile: String,
    // It's only matters for named functions, may differ from '!isStatic(access)' in case of DefaultImpls
    private val needDispatchReceiver: Boolean = false,
    // May differ from containingClassInternalName in case of DefaultImpls
    private val internalNameForDispatchReceiver: String? = null,
    // JVM_IR backend generates $completion, while old backend does not
    private val putContinuationParameterToLvt: Boolean = true,
    // Parameters of suspend lambda are put to the same fields as spilled variables
    private val initialVarsCountByType: Map<Type, Int> = emptyMap(),
    private val shouldOptimiseUnusedVariables: Boolean = true
) : TransformationMethodVisitor(delegate, access, name, desc, signature, exceptions) {
```

```
override fun performTransformations(methodNode: MethodNode) {
    removeFakeContinuationConstructorCall(methodNode)

    replaceReturnsUnitMarkersWithPushingUnitOnStack(methodNode)

    replaceFakeContinuationsWithRealOnes(
        methodNode,
        if (isForNamedFunction) getLastParameterIndex(methodNode.desc, methodNode.access) else 0
    )

    // If there are in-place argument and call markers around suspend call, they end up in separate
    // states of state-machine, leading to AnalyzerError.
    InplaceArgumentsMethodTransformer().transform(containingClassInternalName, methodNode)
    FixStackMethodTransformer().transform(containingClassInternalName, methodNode)
    val suspensionPoints = collectSuspensionPoints(methodNode)
    RedundantLocalsEliminationMethodTransformer(suspensionPoints)
        .transform(containingClassInternalName, methodNode)
    ChangeBoxingMethodTransformer.transform(containingClassInternalName, methodNode)
    methodNode.updateMaxStack()

    checkForSuspensionPointInsideMonitor(methodNode, suspensionPoints)

    // First instruction in the method node may change in case of named function
    val actualCoroutineStart = methodNode.instructions.first

    if (isForNamedFunction) {
        if (putContinuationParameterToLvt) {
            addCompletionParameterToLVT(methodNode)
        }
```

```
fun coolFun() {
  val k = "hey bro"
  println(k)
}
```

```
// access flags 0x19
public final static coolFun()V
 L0
  LINENUMBER 19 L0
  LDC "hey bro"
  ASTORE 0
 L1
  LINENUMBER 20 L1
  GETSTATIC java/lang/System.out : Ljava/io/PrintStream;
  ALOAD 0
  INVOKEVIRTUAL java/io/PrintStream.println (Ljava/lang/Object;)V
 L2
  LINENUMBER 21 L2
  RETURN
 L3
  LOCALVARIABLE k Ljava/lang/String; L1 L3 0
  MAXSTACK = 2
  MAXLOCALS = 1
}
```

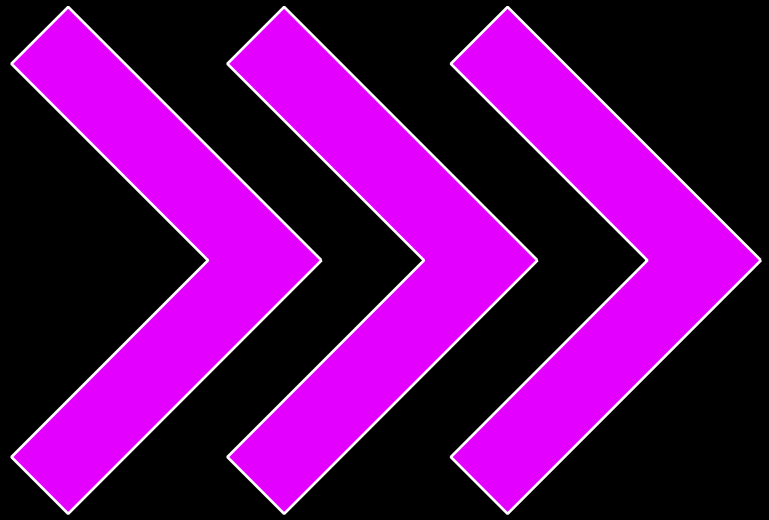
```
suspend fun coolFun() {
  val k = "hey bro"
  println(k)
}
```

```
// access flags 0x19
// signature (Lkotlin/coroutines/Continuation<-Lkotlin/Unit;>;)Ljava/lang/Object;
// declaration:  coolFun(kotlin.coroutines.Continuation<? super kotlin.Unit>)
public final static coolFun(Lkotlin/coroutines/Continuation;)Ljava/lang/Object;
@Lorg/jetbrains/annotations/Nullable;() // invisible
  // annotable parameter count: 1 (invisible)
  @Lorg/jetbrains/annotations/NotNull;() // invisible, parameter 0
 L0
  LINENUMBER 19 L0
  LDC "hey bro"
  ASTORE 1
 L1
  LINENUMBER 20 L1
  GETSTATIC java/lang/System.out : Ljava/io/PrintStream;
  ALOAD 1
  INVOKEVIRTUAL java/io/PrintStream.println (Ljava/lang/Object;)V
 L2
  LINENUMBER 21 L2
  GETSTATIC kotlin/Unit.INSTANCE : Lkotlin/Unit;
  ARETURN
 L3
  LOCALVARIABLE k Ljava/lang/String; L1 L3 1
  LOCALVARIABLE $completion Lkotlin/coroutines/Continuation; L0 L3 0
  MAXSTACK = 2
  MAXLOCALS = 2
}
```

О том как устроены компиляторы

Книги

- Компиляторы: принципы, технологии и инструменты - Ахо, Ульман
- Теория вычислений для программистов - Том Стюарт



Dispatchers, отмена корутин

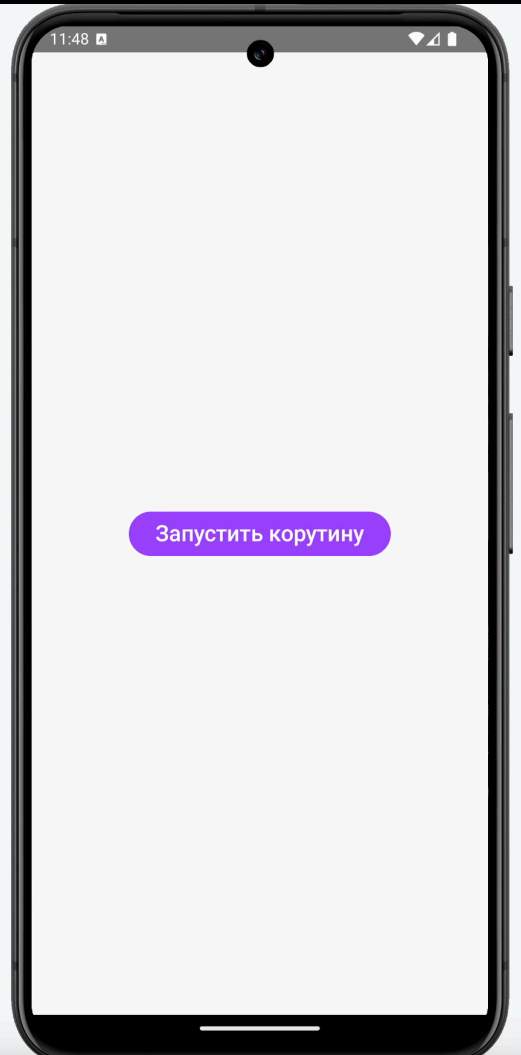
```
class MainActivity : ComponentActivity() {

    val myScope = CoroutineScope(Dispatchers.Default)

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            MyCoroutineButton {
                onClick = myScope.launch {
                    for (i in 1..1000) {
                        Log.d(MOBIUS, i.toString())
                        delay(1000)
                    }
                }
            }
        }
    }
}
```

```
class MainActivity : ComponentActivity() {

    val myScope = CoroutineScope(Dispatchers.Default)

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            MyCoroutineButton {
                onClick = myScope.launch {
                    for (i in 1..1000) {
                        Log.d(MOBIUS, i.toString())
                        delay(1000)
                    }
                }
            }
        }
    }
}
```

Что будет, если нажать на кнопку и перевернуть экран?

1) вывод в логи прекратится сразу

2) вывод в логи продолжится, пока не умрёт процесс приложения

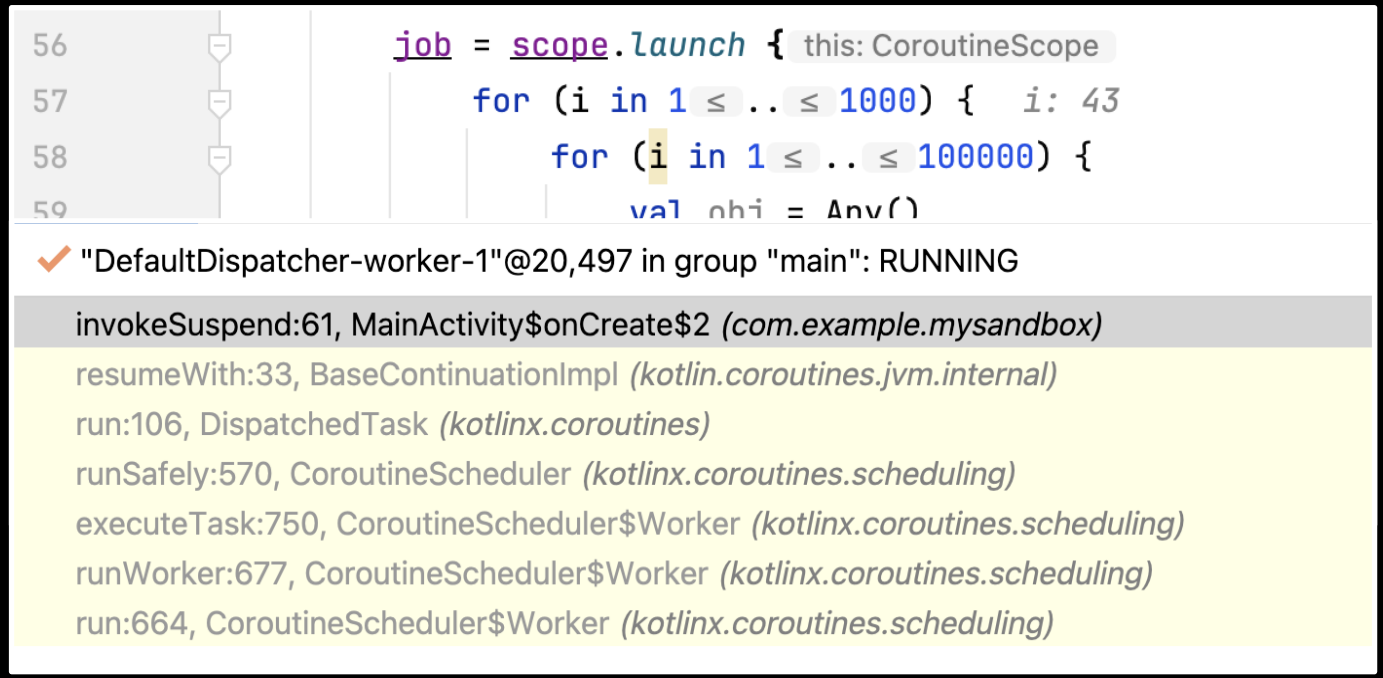

11:48
Запустить корутину

Проверяем, выживет ли Scope

```
class MainActivity : ComponentActivity() {

 companion object {
     var weakReferenceScope: WeakReference<CoroutineScope>? = null
     var weakReferenceJob: WeakReference<Job>? = null

     fun setReference(scope: CoroutineScope, job: Job) {
         weakReferenceScope = WeakReference(scope)
         weakReferenceJob = WeakReference(job)
     }
 }
   val myScope = CoroutineScope(Dispatchers.Default)

    override fun onCreate(savedInstanceState: Bundle?) {
      …
       }
   }
```

Проверяем, выживет ли Scope

```
class MainActivity : ComponentActivity() {

 companion object {
  …
 }

   val myScope = CoroutineScope(Dispatchers.Default)

    override fun onCreate(savedInstanceState: Bundle?) {
     val job = myScope.launch {
         for (i in 1..1000) {
             Log.d(MOBIUS, i.toStr
             delay(1000)
             checkReferences()
         }
     }
      setReference(myScope, job)

 }
}
```


56 job = scope.launch { this: CoroutineScope
57 for (i in 1 ≤ .. ≤ 1000) { i: 43
58 for (i in 1 ≤ .. ≤ 100000) {
"DefaultDispatcher-worker-1"@20,497 in group "main": RUNNING
invokeSuspend:61, MainActivity$onCreate$2 (com.example.mysandbox)
resumeWith:33, BaseContinuationImpl (kotlin.coroutines.jvm.internal)
run:106, DispatchedTask (kotlinx.coroutines)
runSafely:570, CoroutineScheduler (kotlinx.coroutines.scheduling)
executeTask:750, CoroutineScheduler$Worker (kotlinx.coroutines.scheduling)
runWorker:677, CoroutineScheduler$Worker (kotlinx.coroutines.scheduling)
run:664, CoroutineScheduler$Worker (kotlinx.coroutines.scheduling)

Идём в исходники

Иерархия корутин-сущностей

Иерархия корутин-сущностей

```
resumeWith:33, BaseContinuationImpl (kotlin.coroutines.jvm.internal)
run:106, DispatchedTask (kotlinx.coroutines)
```

Иерархия корутин-сущностей

`DispatchedContinuation : DispatchedTask`

`DispatchedTask: SchedulerTask`

`typealias SchedulerTask = Task`

`Task : Runnable`

CoroutineScheduler

```
runSafely:570, CoroutineScheduler (kotlinx.coroutines.scheduling)
executeTask:750, CoroutineScheduler$Worker (kotlinx.coroutines.scheduling)
runWorker:677, CoroutineScheduler$Worker (kotlinx.coroutines.scheduling)
run:664, CoroutineScheduler$Worker (kotlinx.coroutines.scheduling)
```

CoroutineScheduler

- расширяет Executor

```
override fun execute(command: Runnable) = dispatch(command)
```

CoroutineScheduler

- расширяет Executor из Java.util.concurrent
- имеет расширяющийся массив с Workers

```
593     internal inner class Worker private constructor() : Thread() {
594         init {
595             isDaemon = true
596         }
597
```

kotlinx.coroutines.scheduling.CoroutineScheduler

CoroutineScheduler

- расширяет Executor из Java.util.concurrent
- имеет расширяющийся массив с Workers
- передаётся в SchedulerCoroutineDispatcher

```
102
103     override val executor: Executor
104         get() = coroutineScheduler
105
```

kotlinx.coroutines.scheduling.Dispatcher

`class SchedulerCoroutineDispatcher`

class SchedulerCoroutineDispatcher

object DefaultScheduler

```
12      // Instance of Dispatchers.Default
13      internal object DefaultScheduler : SchedulerCoroutineDispatcher(
14          CORE_POOL_SIZE, MAX_POOL_SIZE,
15          IDLE_WORKER_KEEP_ALIVE_NS, DEFAULT_SCHEDULER_NAME
16      ) {
```

kotlinx.coroutines.scheduling.Dispatcher

class SchedulerCoroutineDispatcher

object DefaultScheduler

object Dispatchers

```
public actual object Dispatchers {

    public actual val Default:
CoroutineDispatcher = DefaultScheduler


    public actual val Main: MainCoroutineDispatcher
get() = MainDispatcherLoader.dispatcher


    public actual val Unconfined: CoroutineDispatcher
= kotlinx.coroutines.Unconfined

public val IO: CoroutineDispatcher =
DefaultIoScheduler

}
```

Краткие выводы

- В основе корутин лежат Thread-ы, Executor и Runnable
- Scope - это пульт, а корутина - фильм на экране

Синхронизация в корутинах

Критическая секция:

```
fun main() {
    repeat( times: 2) {
        thread { criticalSection() }
    }
}

@Synchronized
fun criticalSection() {
    println("Starting!")
    Thread.sleep( millis: 10)
    println("Ending!")
}
```

Критическая секция:

```
fun main() {
    val scope = CoroutineScope(Job())

    repeat(2) {
        scope.launch
{ criticalSectionSuspending() }
    }
}

@Synchronized
suspend fun criticalSectionSuspending()
{
    println("Starting!")
    delay(10)
    println("Ending!")
}
```

Критическая секция:

Как поведёт себя программа?

1) код не скомпилируется

2) код скомпилируется и отработает

3) код скомпилируется и упадёт с ошибкой

```
fun main() {
    val scope = CoroutineScope(Job())

    repeat(2) {
        scope.launch
{ criticalSectionSuspending() }
    }
}

@Synchronized
suspend fun criticalSectionSuspending()
{
    println("Starting!")
    delay(10)
    println("Ending!")
}
```

Критическая секция:

```
fun main() {
    val scope = CoroutineScope(Job())

    repeat( times: 2) {
        scope.launch { criticalSectionSuspending() }
    }
}

@Synchronized
suspend fun criticalSectionSuspending() {
    println("Starting!")
    delay( timeMillis: 10)
    println("Ending!")
}
```

Mutex:

```
public interface Mutex {
```

Returns `true` if this mutex is locked.

```
    public val isLocked: Boolean
```

Tries to lock this mutex, returning `false` if this mutex is already locked.

It is recommended to use `withLock` for safety reasons, so that the acquired lock is always released at the end of your critical section, and `unlock` is never invoked before a successful lock acquisition.

Params: `owner` - Optional owner token for debugging. When `owner` is specified (non-null value) and this mutex is already locked with the same token (same identity), this function throws IllegalStateException.

```
    public fun tryLock(owner: Any? = null): Boolean
```

Locks this mutex, suspending caller while the mutex is locked.

This suspending function is cancellable. If the `Job` of the current coroutine is cancelled or completed while this function is suspended, this function immediately resumes with `CancellationException`. There is a **prompt cancellation guarantee**. If the job was cancelled while this function was suspended, it will not resume successfully. See `suspendCancellableCoroutine` documentation for low-level details. This function releases the lock if it was already acquired by this function before the `CancellationException` was thrown.

Note that this function does not check for cancellation when it is not suspended. Use `yield` or `CoroutineScope.isActive` to periodically check for cancellation in tight loops if needed.

Use `tryLock` to try acquiring the lock without waiting.

This function is fair; suspended callers are resumed in first-in-first-out order.

It is recommended to use `withLock` for safety reasons, so that the acquired lock is always released at the end of the critical section, and `unlock` is never invoked before a successful lock acquisition.

Params: `owner` - Optional owner token for debugging. When `owner` is specified (non-null value) and this mutex is already locked with the same token (same identity), this function throws IllegalStateException.

```
    public suspend fun lock(owner: Any? = null)
```

Mutex:

```
val mutex = Mutex()
val scope = CoroutineScope(Job())

fun main() {

    repeat( times: 2) {
        scope.launch { criticalSectionSuspendingLocked() }
    }
}

suspend fun criticalSectionSuspendingLocked() {
    mutex.withLock {
        println("Starting!")
        delay( timeMillis: 10)
        println("Ending!")
    }
}
```

Но почему это работает?

Mutex (coroutines 1.7.3):

```
override suspend fun lock(owner: Any?) {
    if (tryLock(owner)) return
    lockSuspend(owner)
}

private suspend fun lockSuspend(owner: Any?) = suspendCancellableCoroutineReusable<Unit> { cont ->
    val contWithOwner = CancellableContinuationWithOwner(cont, owner)
    acquire(contWithOwner)
}
```

Mutex:

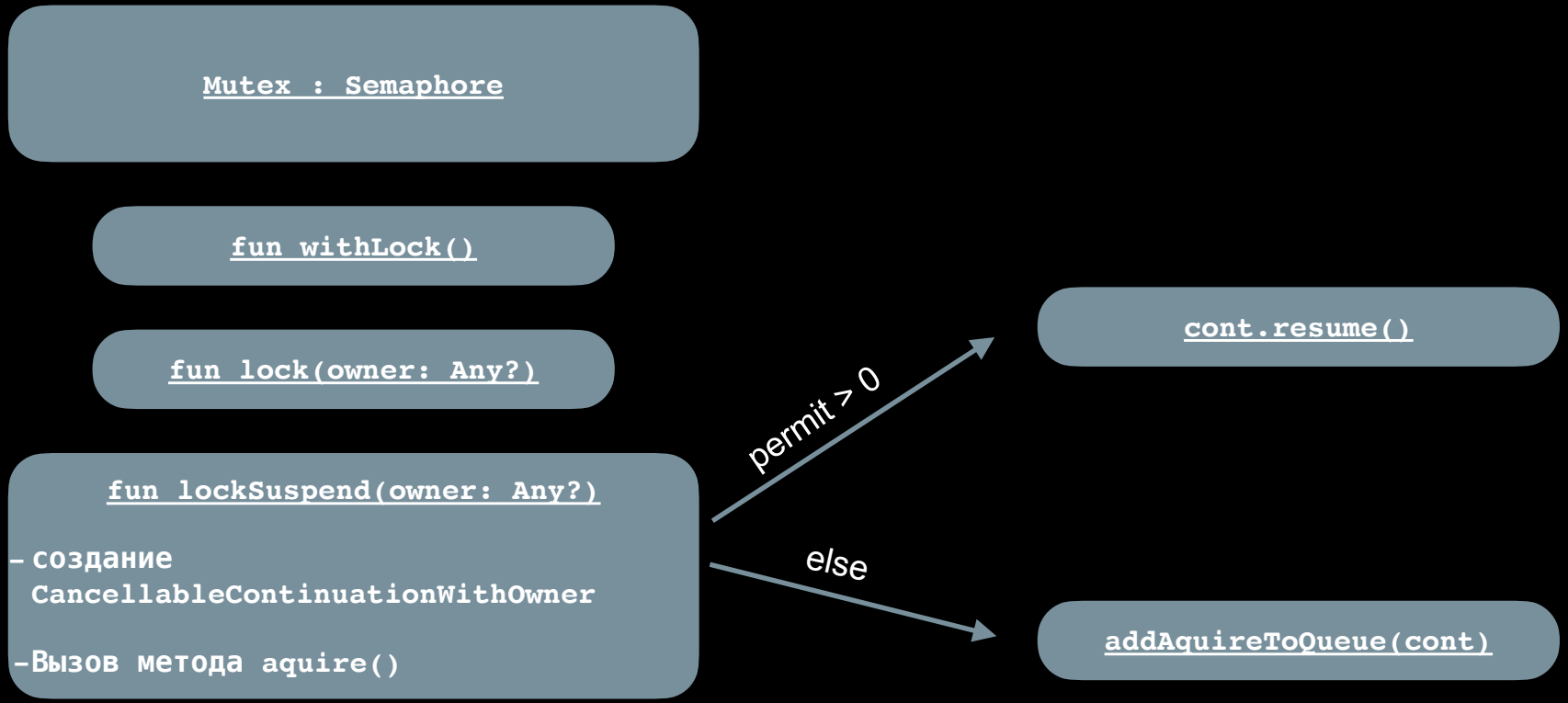
```
private inner class CancellableContinuationWithOwner(
    @JvmField
    val cont: CancellableContinuationImpl<Unit>,
    @JvmField
    val owner: Any?
) : CancellableContinuation<Unit> by cont, Waiter by cont {
    override fun tryResume(value: Unit, idempotent: Any?, onCancellation: ((cause: Throwable) -> Unit)?): Any? {
        assert { this@MutexImpl.owner.value === NO_OWNER }
        val token = cont.tryResume(value, idempotent) {
            assert { this@MutexImpl.owner.value.let { it === NO_OWNER || it === owner } }
            this@MutexImpl.owner.value = owner
            unlock(owner)
        }
        if (token != null) {
            assert { this@MutexImpl.owner.value === NO_OWNER }
            this@MutexImpl.owner.value = owner
        }
        return token
    }

    override fun resume(value: Unit, onCancellation: ((cause: Throwable) -> Unit)?) {
        assert { this@MutexImpl.owner.value === NO_OWNER }
        this@MutexImpl.owner.value = owner
        cont.resume(value) { unlock(owner) }
    }
}
```

Mutex:

```
@JsName( name: "acquireCont")
protected fun acquire(waiter: CancellableContinuation<Unit>) = acquire(
    waiter = waiter,
    suspend = { cont -> addAcquireToQueue(cont as Waiter) },
    onAcquired = { cont -> cont.resume(Unit, onCancellationRelease) }
)
```

Mutex:

```
@JsName( name: "acquireInternal")
private inline fun <W> acquire(waiter: W, suspend: (waiter: W) -> Boolean, onAcquired: (waiter: W) -> Unit) {
    while (true) {
        // Decrement the number of available permits at first.
        val p = decPermits()
        // Is the permit acquired?
        if (p > 0) {
            onAcquired(waiter)
            return
        }
        // Permit has not been acquired, try to suspend.
        if (suspend(waiter)) return
    }
}
```


Mutex : Semaphore
fun withLock()
fun lock(owner: Any?)
fun lockSuspend(owner: Any?)
- создание CancellableContinuationWithOwner
-Вызов метода aquire()
permit > 0
else
cont.resume()
addAquireToQueue(cont)

Есть ли тут проблемы ?

```kotlin
override fun intercept(chain: Interceptor.Chain): Response {
  val req = chain.request()

  val token = runBlocking(appDispatchers.io) { userLocalSource.user().first() }?.token
  val res = chain.proceedWithToken(req, token)

  if (res.code != HTTP_UNAUTHORIZED || token == null) return res

  val newToken: String? = runBlocking(appDispatchers.io) {
    val user = userLocalSource.user().first()
    val tokenWasUpdated = user?.token
```

```
override fun intercept(chain: Interceptor.Chain): Response {
  val req = chain.request()

  val token = runBlocking(appDispatchers.io) { userLocalSource.user().first() }?.token
  val res = chain.proceedWithToken(req, token)

  if (res.code != HTTP_UNAUTHORIZED || token == null) return res

  val newToken: String? = runBlocking(appDispatchers.io) {
    val user = userLocalSource.user().first()
    val tokenWasUpdated = user?.token
```

```
override fun intercept(chain: Interceptor.Chain): Response {
  val req = chain.request()

  val token = runBlocking(appDispatchers.io) { userLocalSource.user().first() }?.token
  val res = chain.proceedWithToken(req, token)

  if (res.code != HTTP_UNAUTHORIZED || token == null) return res

  val newToken: String? = runBlocking(appDispatchers.io) {
    val user = userLocalSource.user().first()
    val tokenWasUpdated = user?.token
```

```
    when {
      user == null || tokenWasUpdated == null -> null
      tokenWasUpdated != token -> tokenWasUpdated
      else -> {
        val refreshTokenRes = apiService.get().refreshToken(user.toRefreshTokenBody())

        when (refreshTokenRes.code()) {
          HTTP_OK -> {
            refreshTokenRes.body()!!.token.also { updatedToken ->
              userLocalSource.update {
                (it ?: return@update null)
                  .toBuilder()
                  .setToken(updatedToken)
                  .build()
              }
            }
          }
          else -> null
        }
      }
    }
  }

  return if (newToken !== null) chain.proceedWithToken(req, newToken) else res
}
```

```
    when {
      user == null || tokenWasUpdated == null -> null
      tokenWasUpdated != token -> tokenWasUpdated
      else -> {
        val refreshTokenRes = apiService.get().refreshToken(user.toRefreshTokenBody())

        when (refreshTokenRes.code()) {
          HTTP_OK -> {
            refreshTokenRes.body()!!.token.also { updatedToken ->
              userLocalSource.update {
                (it ?: return@update null)
                  .toBuilder()
                  .setToken(updatedToken)
                  .build()
              }
            }
          }
          else -> null
        }
      }
    }
  }

  return if (newToken !== null) chain.proceedWithToken(req, newToken) else res
}
```

```
    when {
      user == null || tokenWasUpdated == null -> null
      tokenWasUpdated != token -> tokenWasUpdated
      else -> {
        val refreshTokenRes = apiService.get().refreshToken(user.toRefreshTokenBody())

        when (refreshTokenRes.code()) {
          HTTP_OK -> {
            refreshTokenRes.body()!!.token.also { updatedToken ->
              userLocalSource.update {
                (it ?: return@update null)
                  .toBuilder()
                  .setToken(updatedToken)
                  .build()
              }
            }
          }
          else -> null
        }
      }
    }
  }

  return if (newToken !== null) chain.proceedWithToken(req, newToken) else res
}
```

```kotlin
override fun intercept(chain: Interceptor.Chain): Response {
  val req = chain.request()

  val token = runBlocking(appDispatchers.io) { userLocalSource.user().first() }?.token
  val res = chain.proceedWithToken(req, token)

  if (res.code != HTTP_UNAUTHORIZED || token == null) return res

  val newToken: String? = runBlocking(appDispatchers.io) {
    val user = userLocalSource.user().first()
    val tokenWasUpdated = user?.token

    when {
      user == null || tokenWasUpdated == null -> null
      tokenWasUpdated != token -> tokenWasUpdated
      else -> {
        val refreshTokenRes = apiService.get().refreshToken(user.toRefreshTokenBody())

        when (refreshTokenRes.code()) {
          HTTP_OK -> {
            refreshTokenRes.body()!!.token.also { updatedToken ->
              userLocalSource.update {
                (it ?: return@update null)
                  .toBuilder()
                  .setToken(updatedToken)
                  .build()
              }
            }
          }
          else -> null
        }
      }
    }
  }

  return if (newToken !== null) chain.proceedWithToken(req, newToken) else res
}
```

Какая проблема может возникнуть ?

1) Не обновим токен

2) Несколько запросов одновременно попытаются обновить токен

3) Никаких проблем - пуляем в прод !

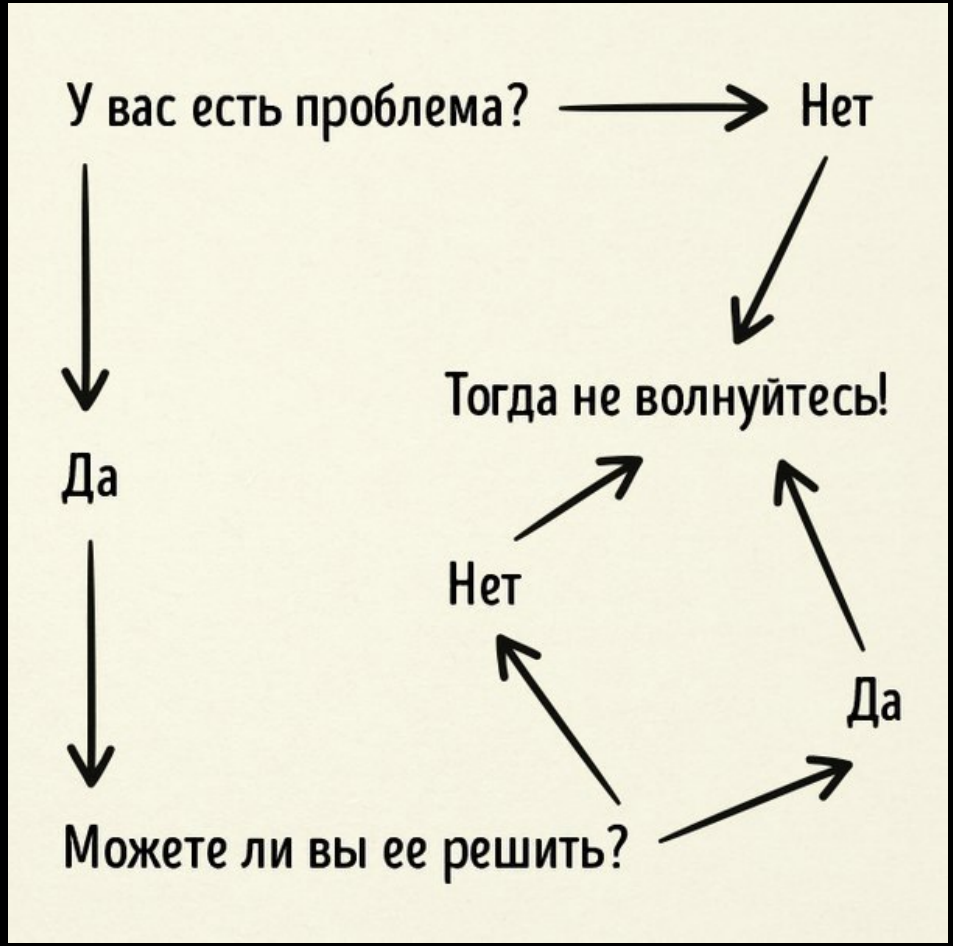
```
override fun intercept(chain: Interceptor.Chain): Response {
  val req = chain.request()

  val token = runBlocking(appDispatchers.io) { userLocalSource.user().first() }?.token
  val res = chain.proceedWithToken(req, token)

  if (res.code != HTTP_UNAUTHORIZED || token == null) return res

  val newToken: String? = runBlocking(appDispatchers.io) {
    val user = userLocalSource.user().first()
    val tokenWasUpdated = user?.token

    when {
      user == null || tokenWasUpdated == null -> null
      tokenWasUpdated != token -> tokenWasUpdated
      else -> {
        val refreshTokenRes = apiService.get().refreshToken(user.toRefreshTokenBody())

        when (refreshTokenRes.code()) {
          HTTP_OK -> {
            refreshTokenRes.body()!!.token.also { updatedToken ->
              userLocalSource.update {
                (it ?: return@update null)
                  .toBuilder()
                  .setToken(updatedToken)
                  .build()
              }
            }
          }
          else -> null
        }
      }
    }
  }

  return if (newToken !== null) chain.proceedWithToken(req, newToken) else res
}
```

Какая проблема может возникнуть ?

1) Не обновим токен

2) Несколько запросов одновременно попытаются обновить токен

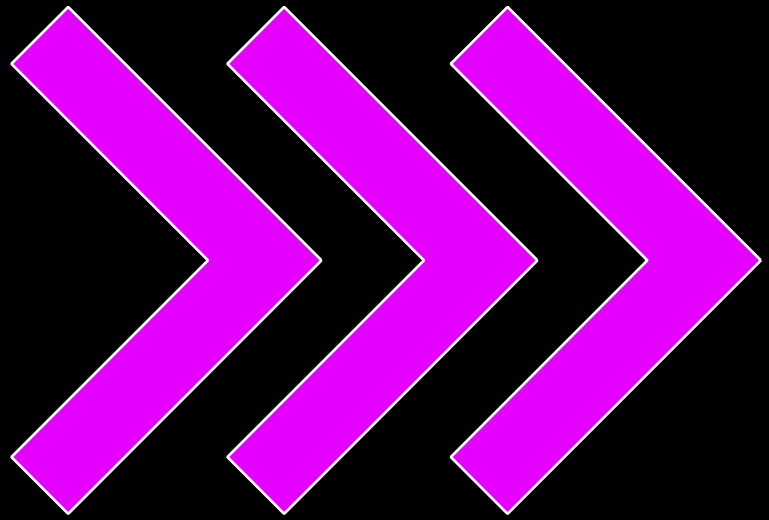
3) Никаких проблем - пуляем в прод !



```
private val mutex = Mutex()

override fun intercept(chain: Interceptor.Chain): Response {
  val req = chain.request()

  val token = runBlocking(appDispatchers.io) { userLocalSource.user().first() }?.token
  val res = chain.proceedWithToken(req, token)

  if (res.code != HTTP_UNAUTHORIZED || token == null) return res

  val newToken: String? = runBlocking(appDispatchers.io) {
    mutex.withLock {
      val user = userLocalSource.user().first()
      val tokenWasUpdated = user?.token
```

Выводы

- Синхронизация в корутинах нужна, но обычная синхронизация не работает
- Mutex в корутинах - это Semafore
- Mutex работает просто потому, что он знает, что такое корутины и как они устроены



Немного о Structured concurrency

Качаем скриншоты из битв Абакара

Качаем скриншоты из битв Абакара

```
fun loadCsScreenShots(version: String) {
    for (i in 1..10) {
        Log.d(MOBIUS, "Качаем скриншоты для CS $version")
        Delay(1000)
        if (i == 2 && version == "2.0") {
            throw Exception("MOBIUS ошибка загрузки")
        }
    }
}
```

Качаем скриншоты из битв Абакара

```
fun loadCsScreenShots(version: String) {
    for (i in 1..10) {
        Log.d(MOBIUS, "Качаем скриншоты для CS $version")
        Delay(1000)
        if (i == 2 && version == "2.0") {
            throw Exception("MOBIUS ошибка загрузки")
        }
    }
}
```

```
class MainActivity : ComponentActivity() {

 val handler = CoroutineExceptionHandler { _, exception ->
    Log.d(MOBIUS, "отловили исключение»)
 }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            MyCoroutineButton {
                onClick = {
     job1 = lifecycleScope.launch(Dispatchers.IO + handler) {
            loadCsScreenShots("1.6")
        }

     job2 = lifecycleScope.launch(Dispatchers.IO + handler) {
           loadCsScreenShots("2.0")
        }
      }
    }
  }
```

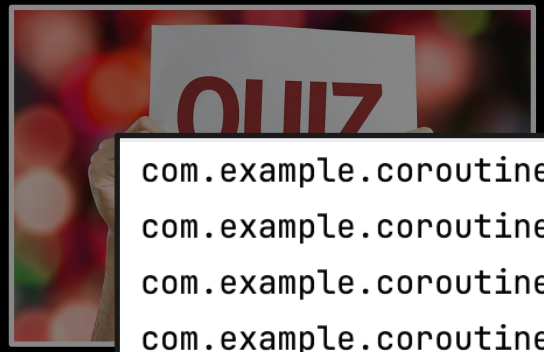
Качаем скриншоты из битв Абакара

Как отработает программа?

1) программа упадет с ошибкой
2) исключение отловится, обе корутины отработают до конца
3) исключение отловится, но обе корутины прекратят работу после ошибки
4) исключение отловится, job1 продолжит работу, job2 прекратится

```
class MainActivity : ComponentActivity() {

 val handler = CoroutineExceptionHandler { _, exception ->
    Log.d(MOBIUS, "отловили исключение»)
 }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            MyCoroutineButton {
                onClick = {
     job1 = lifecycleScope.launch(Dispatchers.IO + handler) {
            loadCsScreenShots("1.6")
        }

     job2 = lifecycleScope.launch(Dispatchers.IO + handler) {
           loadCsScreenShots("2.0")
        }
      }
    }
  }
```

Качаем скриншоты из битв Абакара




```
com.example.coroutinesandbox  D  Качаем скриншоты для CS 2.0
com.example.coroutinesandbox  D  Качаем скриншоты для CS 1.6
com.example.coroutinesandbox  D  Качаем скриншоты для CS 1.6
com.example.coroutinesandbox  D  Качаем скриншоты для CS 2.0
com.example.coroutinesandbox  D  Качаем скриншоты для CS 1.6
com.example.coroutinesandbox  D  отловили исключение
com.example.coroutinesandbox  D  Качаем скриншоты для CS 1.6
com.example.coroutinesandbox  D  Качаем скриншоты для CS 1.6
com.example.coroutinesandbox  D  Качаем скриншоты для CS 1.6
com.example.coroutinesandbox  D  Качаем скриншоты для CS 1.6
```

Как отработает

1) программа у
2) исключение
   отработают д
3) исключение
   прекратят работу после ошибки
4) исключение отловится, job1 продолжит работу, job2 прекратится

```
class MainActivity : ComponentActivity() {

                                                    eption ->

                                                    dle?) {

                                                    + handler) {

                                                    + handler) {

        }
      }
    }
  }
```

Лезем под капот

lifecycleScope

```
public val Lifecycle.coroutineScope: LifecycleCoroutineScope
    get() {
        while (true) {
            val existing = internalScopeRef.get() as LifecycleCoroutineScopeImpl?
            if (existing != null) {
                return existing
            }
            val newScope = LifecycleCoroutineScopeImpl(
                this,
                SupervisorJob() + Dispatchers.Main.immediate
            )
            if (internalScopeRef.compareAndSet(null, newScope)) {
                newScope.register()
                return newScope
            }
        }
    }
```

Job

```
public val children: Sequence<Job>
public fun attachChild(child: ChildJob): ChildHandle


public val parent: Job?
```

DispatchedTask :: run

continuation.*resumeWithException*(exception)

BaseContinuationImpl :: resumeWith

AbstractCoroutine :: resumeWith

JobSupport :: makeCompletingOnce

DispatchedTask :: run

continuation.*resumeWithException*(exception)

BaseContinuationImpl :: resumeWith

AbstractCoroutine :: resumeWith

JobSupport :: makeCompletingOnce

DispatchedTask :: run

continuation.*resumeWithException*(exception)

BaseContinuationImpl :: resumeWith

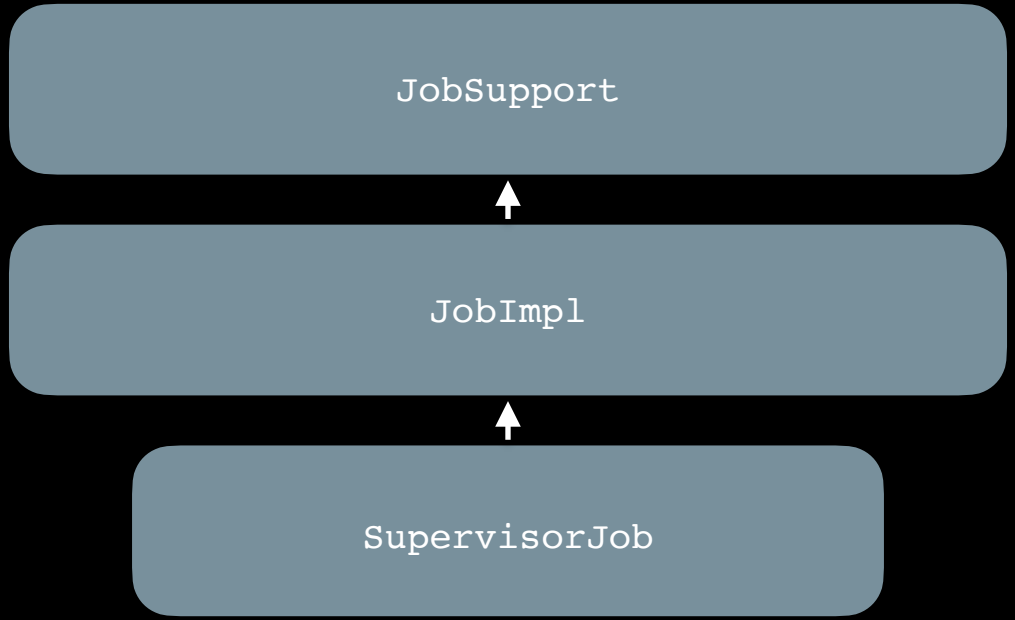
AbstractCoroutine :: resumeWith

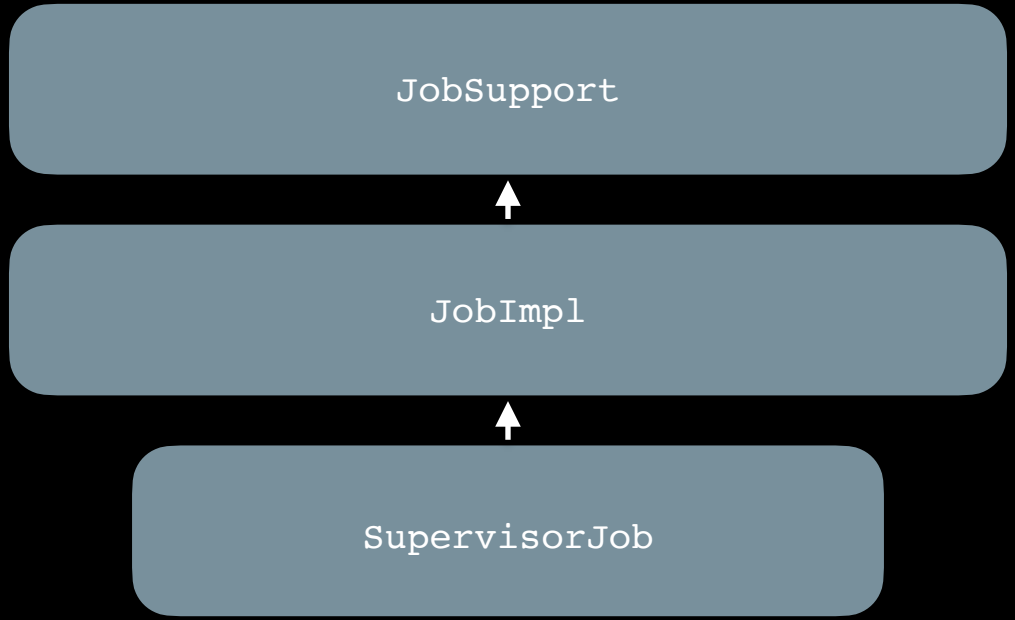
Job$ etingOnce

JobSupport :: cancelParent

```
652    public open fun childCancelled(cause: Throwable): Boolean {
653        if (cause is CancellationException) return true
654        return cancelImpl(cause) && handlesException
655    }
656
```


SupervisorJob


JobSupport
JobImpl
SupervisorJob




```
private class SupervisorJobImpl(parent: Job?) : JobImpl(parent) {
    override fun childCancelled(cause: Throwable): Boolean = false
}
```

Выводы

- Магия SupervisorJob - старый добрый Boolean

Что не так с GlobalScope?

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    scope.launch { this: CoroutineScope
        val job1 = scope.launch { this: CoroutineScope
            for (i in 1 ≤ .. ≤ 10) {
                Log.d( tag: "MOBIUS",  msg: "логи в нашем scope №$i")
                delay( timeMillis: 1000)
            }
        }

        val job2 = GlobalScope.launch { this: CoroutineScope
            for (i in 1 ≤ .. ≤ 10) {
                Log.d( tag: "MOBIUS",  msg: "логи GlobalScope №$i")
                delay( timeMillis: 1000)
            }
        }
    }
}

override fun onResume() {
    super.onResume()
    scope.cancel()
}
```

```
D  логи GlobalScope №1
D  логи в нашем scope №1
D  логи GlobalScope №2
D  логи GlobalScope №3
D  логи GlobalScope №4
D  логи GlobalScope №5
D  логи GlobalScope №6
D  логи GlobalScope №7
D  логи GlobalScope №8
D  логи GlobalScope №9
D  логи GlobalScope №10
```

```kotlin
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    scope.launch { this: CoroutineScope
        val job1 = scope.launch { this: CoroutineScope
            for (i in 1 ≤ .. ≤ 10) {
                Log.d( tag: "MOBIUS",  msg: "логи в нашем scope №$i")
                delay( timeMillis: 1000)
            }
        }

        val job2 = GlobalScope.launch { this: CoroutineScope
            for (i in 1 ≤ .. ≤ 10) {
                Log.d( tag: "MOBIUS",  msg: "логи GlobalScope №$i")
                delay( timeMillis: 1000)
            }
        }
    }
}

override fun onResume() {
    super.onResume()
    scope.cancel()
}
```

```
@DelicateCoroutinesApi
public object GlobalScope : CoroutineScope {
    Returns EmptyCoroutineContext.
    override val coroutineContext: CoroutineContext
        get() = EmptyCoroutineContext
}
```

```
public object EmptyCoroutineContext : CoroutineContext, Serializable {
    private const val serialVersionUID: Long = 0
    private fun readResolve(): Any = EmptyCoroutineContext

    public override fun <E : Element> get(key: Key<E>): E? = null
```

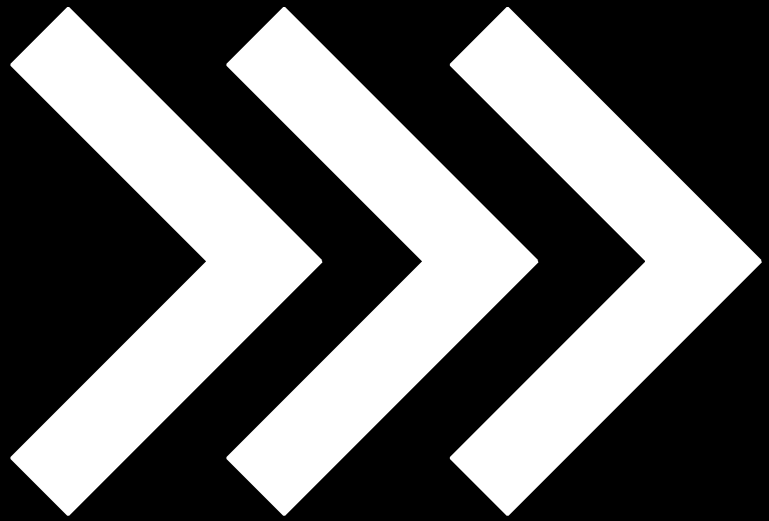
```
public fun CoroutineScope.launch(
    context: CoroutineContext = EmptyCoroutineContext,
    start: CoroutineStart = CoroutineStart.DEFAULT,
    block: suspend CoroutineScope.() -> Unit
): Job
    val newContext = newCoroutineContext(context)
    val coroutine = if (start.isLazy)
        LazyStandaloneCoroutine(newContext,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

> Initializes parent job. It shall be invoked at most once after construction after all other initialization.

```
protected fun initParentJob(parent: Job?) {
    assert { parentHandle == null }
    if (parent == null) {
        parentHandle = NonDisposableHandle
        return
    }
    parent.start() // make sure the parent is started
    @Suppress( ...names: "DEPRECATION")
    val handle = parent.attachChild( child: this)
    parentHandle = handle
    // now check our state _after_ registering (see tryFinalizeSimpleState order of actions)
    if (isCompleted) {
        handle.dispose()
        parentHandle = NonDisposableHandle // release it just in case, to aid GC
    }
}
```

```
@InternalCoroutinesApi
public object NonDisposableHandle : DisposableHandle, ChildHandle {

    override val parent: Job? get() = null

    Does not do anything.

    Suppress:

    override fun dispose() {}

    Returns false.

    Suppress:

    override fun childCancelled(cause: Throwable): Boolean = false

    Returns "NonDisposableHandle" string.

    Suppress:

    override fun toString(): String = "NonDisposableHandle"
}
```

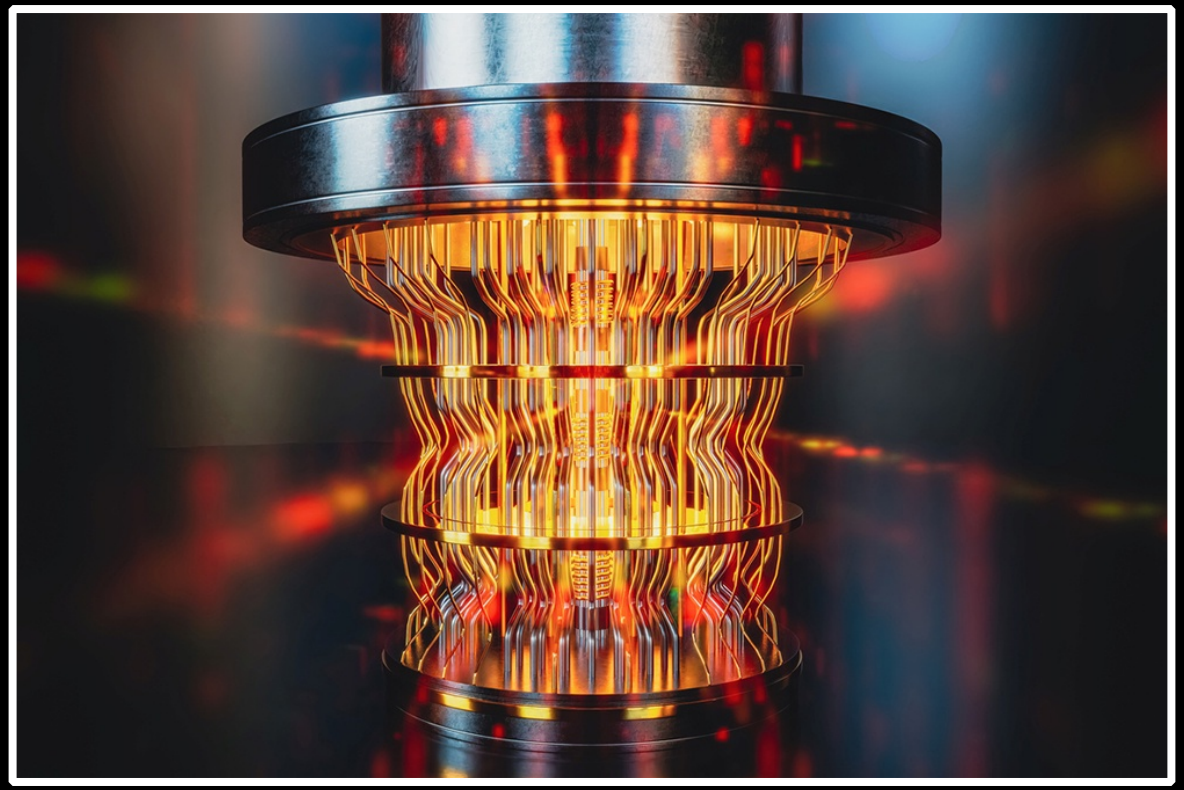
```
@InternalCoroutinesApi
public ob                                                    {
    overr
        Does
        Supp
    overr
        Retu
        Supp
    overr                                                    e
        Retu
        Supp
    overr
}
```



Подведём итоги

1. Исследование библиотек — открытие terra incognita

2. Магии нет, но всё работает волшебным образом



СПАСИБО!

Абакар Магомедов

Технический лидер разработки, Альфа-банк

Александр Гирев

Android developer, Wildberries