

Kotlin Coroutines

Записки Android-разработчика

Осипов Михаил

Tinkoff.ru

О себе

- Разработка под Android больше 6 лет
- Клиент для умного дома
- Чаты
- Новостные приложения
- Преподавание (лекции, практические занятия)
- В Tinkoff.ru около 2 лет, в команде "Tinkoff mobile"

О чем будем говорить

О чем будем говорить

- Концепция корутин

О чем будем говорить

- Концепция корутин
- Зачем корутины, если есть Rx?

О чем будем говорить

- Концепция корутин
- Зачем корутины, если есть Rx?
- Основные моменты

О чем будем говорить

- Концепция корутин
- Зачем корутины, если есть Rx?
- Основные моменты
- Обработка ошибок

О чем будем говорить

- Концепция корутин
- Зачем корутины, если есть Rx?
- Основные моменты
- Обработка ошибок
- Выводы

Мотивация

Мотивация

Цели доклада:

Мотивация

Цели доклада:

- Что представляют из себя корутины в Kotlin

Мотивация

Цели доклада:

- Что представляют из себя корутины в Kotlin
- Как ими пользоваться

Мотивация

Цели доклада:

- Что представляют из себя корутины в Kotlin
- Как ими пользоваться
- На что стоит обратить внимание

Мотивация

Цели доклада:

- Что представляют из себя корутины в Kotlin
- Как ими пользоваться
- На что стоит обратить внимание

Вам будет интересен этот доклад если вы ...

Мотивация

Цели доклада:

- Что представляют из себя корутины в Kotlin
- Как ими пользоваться
- На что стоит обратить внимание

Вам будет интересен этот доклад если вы ...

- Интересуетесь корутинами в Kotlin

Мотивация

Цели доклада:

- Что представляют из себя корутины в Kotlin
- Как ими пользоваться
- На что стоит обратить внимание

Вам будет интересен этот доклад если вы ...

- Интересуетесь корутинами в Kotlin
- Android-разработчик (не обязательно)

Мотивация

Цели доклада:

- Что представляют из себя корутины в Kotlin
- Как ими пользоваться
- На что стоит обратить внимание

Вам будет интересен этот доклад если вы ...

- Интересуетесь корутинами в Kotlin
- Android-разработчик (не обязательно)
- Активно используете Rx, и думаете, что что-то идет не так

Введение

Введение

Введение

- Были реализованы еще в Simula'67

Введение

- Были реализованы еще в Simula'67
- Из современных поддерживают: Scala, Python, C#, JS, GO

Введение

- Были реализованы еще в Simula'67
- Из современных поддерживают: Scala, Python, C#, JS, GO
- В Kotlin с версии 1.3

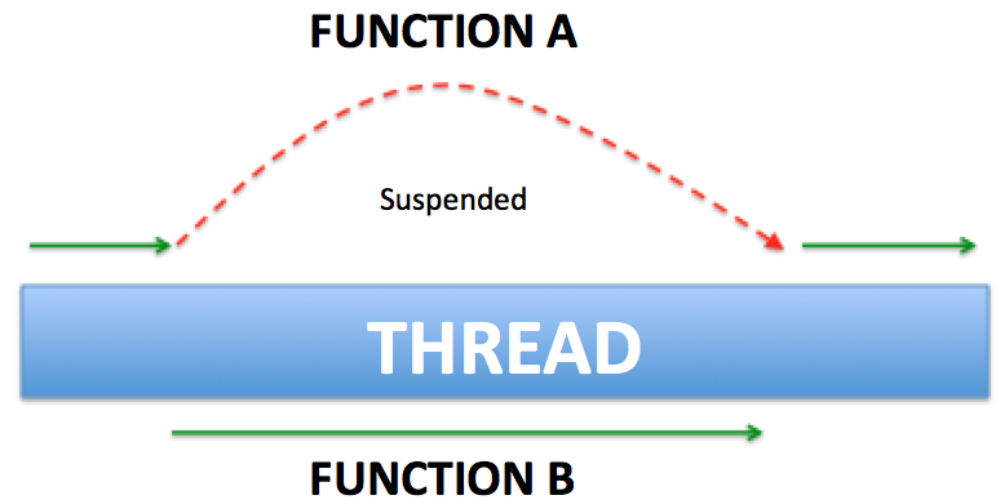
Введение

- Были реализованы еще в Simula'67
- Из современных поддерживают: Scala, Python, C#, JS, GO
- В Kotlin с версии 1.3
- *Корутина* - исполняемый код, который может быть **прерван** без блокировки потока, после завершения прерывания *корутина* продолжает выполнение

Введение

- Были реализованы еще в Simula'67
- Из современных поддерживают: Scala, Python, C#, JS, GO
- В Kotlin с версии 1.3
- *Корутина* - исполняемый код, который может быть **прерван** без блокировки потока, после завершения прерывания *корутина* продолжает выполнение

```
// coroutine
{
    regularFun1()
    regularFun2()
    functionA() // suspending without blocking
    regularFun3()
    regularFun4()
}
```



Корутины в Kotlin

```
1 ...
2 launch {
3     val deferred = async(Dispatchers.IO) {
4         repo.getUser()
5     }
6     val user = deferred.await()
7     showUser(user)
8 }
9 ...
```

Корутины в Kotlin

```
1 ...
2 launch {
3     val deferred = async(Dispatchers.IO) {
4         repo.getUser()
5     }
6     val user = deferred.await()
7     showUser(user)
8 }
9 ...
```

coroutine

Корутины в Kotlin

```
1 ...
2 launch { coroutine
3     val deferred = async(Dispatchers.IO) {
4         repo.getUser()
5     }
6     val user = deferred.await() suspension
7     showUser(user)
8 }
9 ...
```

Корутины в Kotlin

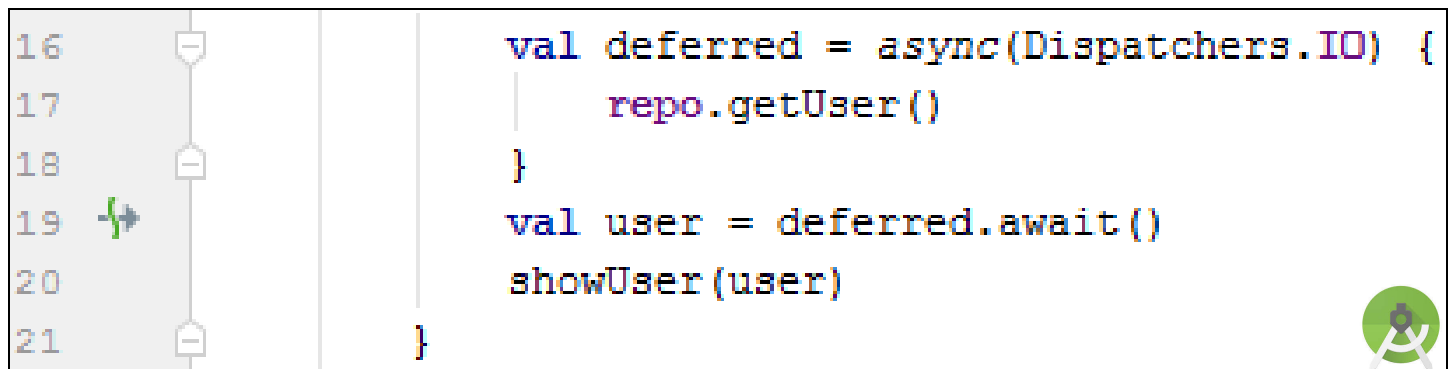
```
1 ...
2 launch {
3     val deferred = async(Dispatchers.IO) {
4         repo.getUser()
5     }
6     val user = deferred.await()
7     showUser(user)
8 }
9 ...
```

coroutine

suspension

```
16
17
18
19
20
21
```

```
val deferred = async(Dispatchers.IO) {
    repo.getUser()
}
val user = deferred.await()
showUser(user)
}
```



Coroutines vs Rx

СРАВНИМ

Rx

```
val disposable = repo.getUser()
    .subscribeOn(Schedulers.io())
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe { user ->
        showUser(user)
    }
```

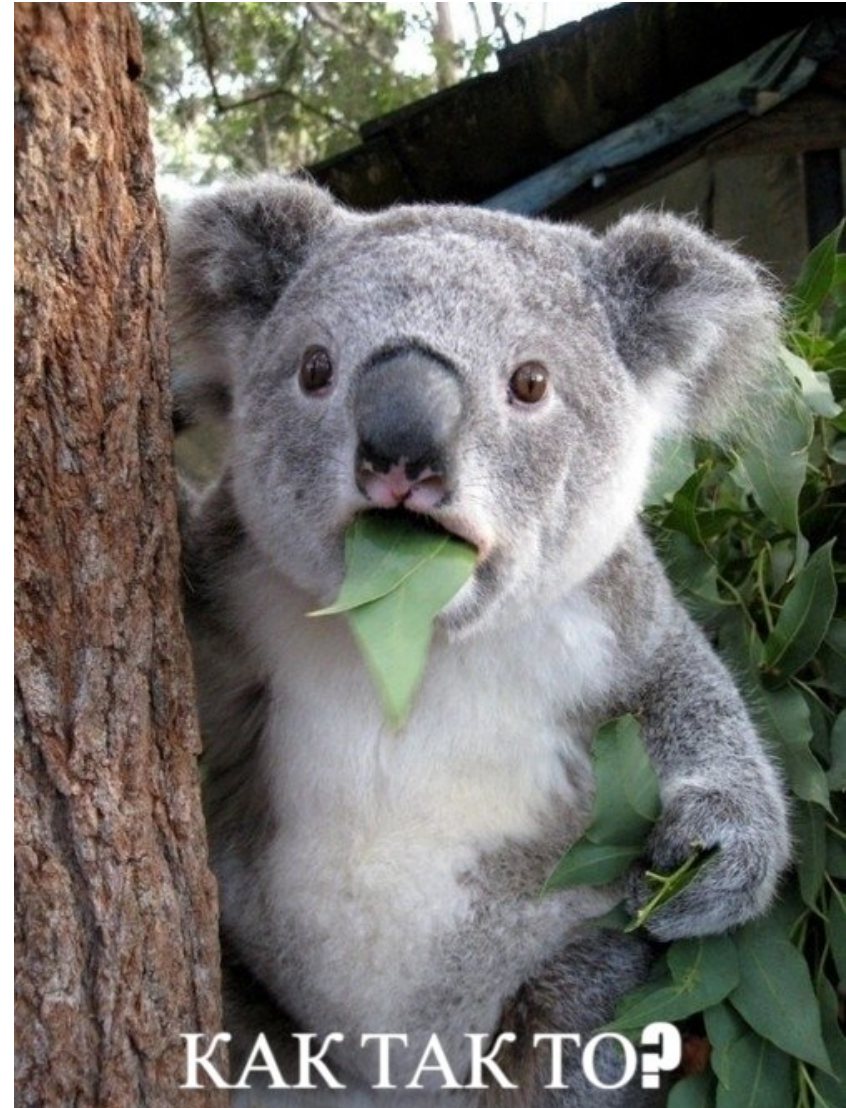
Coroutines

```
launch {
    val deferred = async(Dispatchers.IO) {
        repo.getUser()
    }
    val user = deferred.await()
    showUser(user)
}
```

А если по сложнее?

Rx

```
var savedUser: User? = null
val sessionSingle = repo.getSession().cache()
Singles.zip(
    sessionSingle,
    sessionSingle.flatMap { repo.getToken(it) }
).flatMap { (session, token) ->
    repo.getUser(session, token)
}
    .subscribeOn(Schedulers.io())
    .observeOn(AndroidSchedulers.mainThread())
    .doOnSuccess { user ->
        savedUser = user
        showUser(user)
    }
    .observeOn(Schedulers.io())
    .flatMap { repo.getUserDetails(it.id) }
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe { details ->
        showAllTogether(savedUser, details)
    }
}
```



А если по сложнее?

Rx

```
var savedUser: User? = null
val sessionSingle = repo.getSession().cache()
Singles.zip(
    sessionSingle,
    sessionSingle.flatMap { repo.getToken(it) }
).flatMap { (session, token) ->
    repo.getUser(session, token)
}
    .subscribeOn(Schedulers.io())
    .observeOn(AndroidSchedulers.mainThread())
    .doOnSuccess { user ->
        savedUser = user
        showUser(user)
    }
    .observeOn(Schedulers.io())
    .flatMap { repo.getUserDetails(it.id) }
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe { details ->
        showAllTogether(savedUser, details)
    }
}
```

Coroutines

```
launch {
    val user = withContext(Dispatchers.IO) {
        val session = repo.getSession()
        val token = repo.getToken(session)
        repo.getUser(token, session)
    }
    showUser(user)
    val details = withContext(Dispatchers.IO) {
        repo.getUserDetails(user.id)
    }
    showAllTogether(user, details)
}
```



Зачем переходить на корутины?

Зачем переходить на корутины?

Rx:

- Бизнес-логика не всегда удачно ложится на реактивную модель

Зачем переходить на корутины?

Rx:

- Бизнес-логика не всегда удачно ложится на реактивную модель
- Операторы + сопутствующий код

Зачем переходить на корутины?

Rx:

- Бизнес-логика не всегда удачно ложится на реактивную модель
- Операторы + сопутствующий код
- Тяжело поддерживать сложные chain-ы

Зачем переходить на корутины?

Rx:

- Бизнес-логика не всегда удачно ложится на реактивную модель
- Операторы + сопутствующий код
- Тяжело поддерживать сложные chain-ы

Coroutines:

- Последовательный стиль

Зачем переходить на корутины?

Rx:

- Бизнес-логика не всегда удачно ложится на реактивную модель
- Операторы + сопутствующий код
- Тяжело поддерживать сложные chain-ы

Coroutines:

- Последовательный стиль
- Лаконичность

Зачем переходить на корутины?

Rx:

- Бизнес-логика не всегда удачно ложится на реактивную модель
- Операторы + сопутствующий код
- Тяжело поддерживать сложные chain-ы

Coroutines:

- Последовательный стиль
- Лаконичность
- Легко поддерживать

Зачем переходить на корутины?

Rx:

- Бизнес-логика не всегда удачно ложится на реактивную модель
- Операторы + сопутствующий код
- Тяжело поддерживать сложные chain-ы

Coroutines:

- Последовательный стиль
- Лаконичность
- Легко поддерживать
- Инструмент языка

Suspending functions

Suspend keyword

```
public interface Deferred<out T> : Job {  
    ...  
    public suspend fun await(): T  
    ...  
}
```

Suspend keyword

```
public interface Deferred<out T> : Job {  
    ...  
    public suspend fun await(): T  
    ...  
}
```

- suspend - модификатор метода/функции/лямбды

Suspend keyword

```
public interface Deferred<out T> : Job {  
    ...  
    public suspend fun await(): T  
    ...  
}
```

- suspend - модификатор метода/функции/лямбды
- suspending-функция может прервать выполнение корутины

Suspend keyword

```
public interface Deferred<out T> : Job {  
    ...  
    public suspend fun await(): T  
    ...  
}
```

- suspend - модификатор метода/функции/лямбды
- suspending-функция может прервать выполнение корутины
- suspending-функции можно использовать исключительно внутри других suspending-функций (compile-time check)

```
fun onStart() {  
    val deferred = async(Dispatchers.IO) { this: CoroutineScope  
        repo.getUser()  
    }  
    val user = deferred.await()  
}
```

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

Suspend: вопросы

- Можно/нужно ли нам его использовать?
- А кто прерывания то делает?

Пишем свои *suspending*-функции

Зачем помечать функцию модификатором *suspend*?

Пишем свои `suspending`-функции

Зачем помечать функцию модификатором `suspend`?

Вероятнее всего, т.к. внутри вы используете другую `suspending`-функцию

```
suspend fun mySomeFunction() : String {  
    ...  
    val a = anotherSuspendingFunction() // suspending call  
    ...  
}
```


Пишем свои `suspending`-функции

Зачем помечать функцию модификатором `suspend`?

Вероятнее всего, т.к. внутри вы используете другую `suspending`-функцию

```
suspend fun mySomeFunction() : String {  
    ...  
    val a = anotherSuspendingFunction() // suspending call  
    ...  
}
```

и либо она помечена `suspend` по той же причине, либо...

Пишем свои `suspending`-функции

Зачем помечать функцию модификатором `suspend`?

Вероятнее всего, т.к. внутри вы используете другую `suspending`-функцию

```
suspend fun mySomeFunction() : String {  
    ...  
    val a = anotherSuspendingFunction() // suspending call  
    ...  
}
```

и либо она помечена `suspend` по той же причине, либо...

```
package kotlinx.coroutines...  
  
public interface Deferred<out T> : Job {  
    ...  
    public suspend fun await(): T  
    ...  
}
```

Прерывания: api в kotlin-stdlib

```
// kotlin-stdlib-common

package kotlin.coroutines.intrinsics
...
public expect inline fun <T> (suspend () -> T).startCoroutineUninterceptedOrReturn(
    completion: Continuation<T>
): Any?

public expect inline fun <R, T> (suspend R.() -> T).startCoroutineUninterceptedOrReturn(
    receiver: R,
    completion: Continuation<T>
): Any?

public expect fun <T> (suspend () -> T).createCoroutineUnintercepted(
    completion: Continuation<T>
): Continuation<Unit>

public expect fun <R, T> (suspend R.() -> T).createCoroutineUnintercepted(
    receiver: R,
    completion: Continuation<T>
): Continuation<Unit>
```

Coroutine-builder

Coroutine-builder

Coroutine-builder - функция (не suspending), принимающая, помимо прочего, на вход suspending-лямбду и CoroutineContext

Coroutine-builder

Coroutine-builder - функция (не `suspending`), принимающая, помимо прочего, на вход `suspending`-лямбду и `CoroutineContext`

```
public fun CoroutineScope.[someBuilder](  
    //...  
    context: CoroutineContext,  
    block: suspend CoroutineScope.() -> ...  
)    //...
```

Coroutine-builder

Coroutine-builder - функция (не `suspending`), принимающая, помимо прочего, на вход `suspending`-лямбду и `CoroutineContext`

```
public fun CoroutineScope.[someBuilder](  
    //...  
    context: CoroutineContext, ←————— // Dispatchers  
    block: suspend CoroutineScope.() -> ... Dispatchers.Main  
    //... Dispatchers.IO  
)
```

Coroutine-builder

Coroutine-builder - функция (не `suspending`), принимающая, помимо прочего, на вход `suspending`-лямбду и `CoroutineContext`

```
public fun CoroutineScope.[someBuilder](
    //...
    context: CoroutineContext, ←————— // Dispatchers
    block: suspend CoroutineScope.() -> ... Dispatchers.Main
) //... Dispatchers.IO

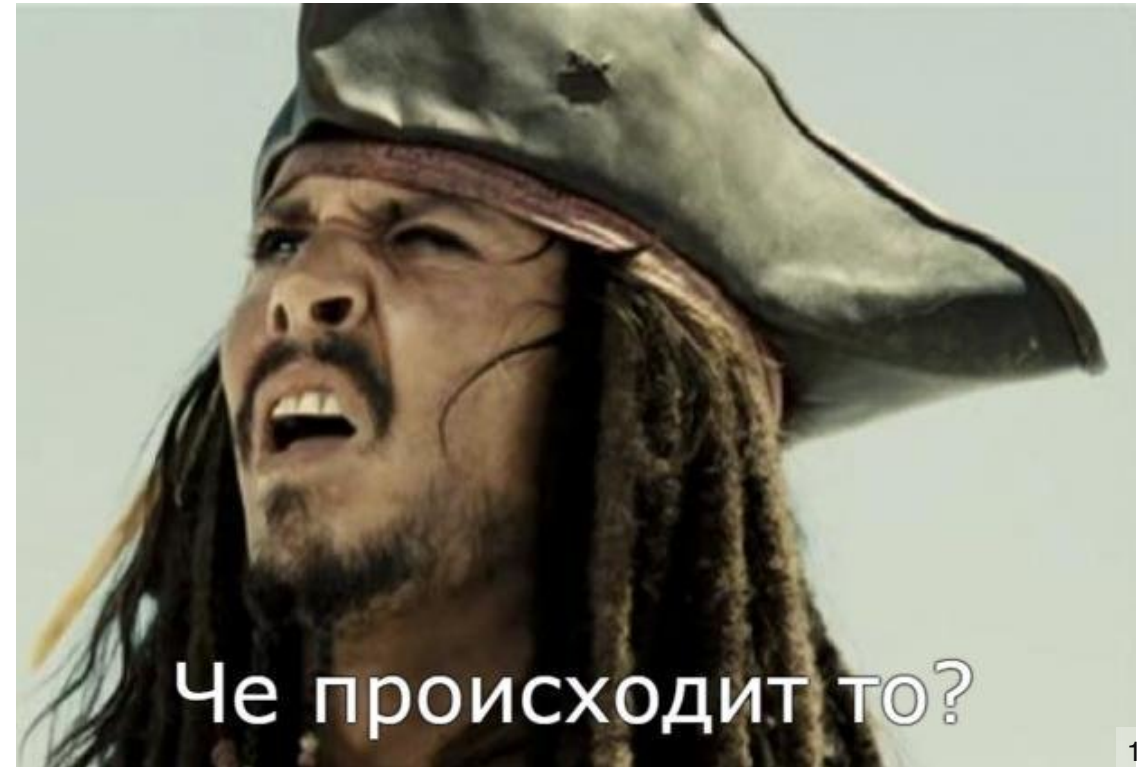
// Top builders
public fun launch(...): Job { ... }
public fun async(...): Diferred<T> { ... }
```


ПОНЯТНО, НО...

```
fun onStart() {  
    launch(Dispatchers.Main) {  
        val deferred = async(Dispatchers.IO) {  
            repo.getUser()  
        }  
        val user = deferred.await()  
        showUser(user)  
    }  
}
```

ПОНЯТНО, НО...

```
fun onStart() {  
    launch(Dispatchers.Main) {  
        val deferred = async(Dispatchers.IO) {  
            repo.getUser()  
        }  
        val user = deferred.await()  
        showUser(user)  
    }  
}
```



Важно понимать

```
fun onStart() {  
    val job = launch(Dispatchers.Main) {  
        /*  
            some code  
        */  
        val deferred = async(Dispatchers.IO) {  
            /*  
                call api, get from DB, etc.  
            */  
        }  
        /*  
            some code  
        */  
        val data = deferred.await()  
        /*  
            some code  
        */  
    }  
}
```

Coroutine-builder:

Важно понимать

```
fun onStart() {  
    val job = launch(Dispatchers.Main) {  
        /*  
            some code  
        */  
        val deferred = async(Dispatchers.IO) {  
            /*  
                call api, get from DB, etc.  
            */  
        }  
        /*  
            some code  
        */  
        val data = deferred.await()  
        /*  
            some code  
        */  
    }  
}
```

Coroutine-builder:

- всегда имеет Dispatcher

Важно понимать

```
fun onStart() {  
    val job = launch(Dispatchers.Main) {  
        /*  
            some code  
        */  
        val deferred = async(Dispatchers.IO) {  
            /*  
                call api, get from DB, etc.  
            */  
        }  
        /*  
            some code  
        */  
        val data = deferred.await()  
        /*  
            some code  
        */  
    }  
}
```

Coroutine-builder:

- всегда имеет Dispatcher
- не блокирует поток

Важно понимать

```
fun onStart() {  
    val job = launch(Dispatchers.Main) {  
        /*  
            some code  
        */  
        val deferred = async(Dispatchers.IO) {  
            /*  
                call api, get from DB, etc.  
            */  
        }  
        /*  
            some code  
        */  
        val data = deferred.await()  
        /*  
            some code  
        */  
    }  
}
```

Coroutine-builder:

- всегда имеет Dispatcher
- не блокирует поток
- по умолчанию, сразу стартует корутину

Важно понимать

```
fun onStart() {  
    val job = launch(Dispatchers.Main) {  
        /*  
            some code  
        */  
        val deferred = async(Dispatchers.IO) {  
            /*  
                call api, get from DB, etc.  
            */  
        }  
    }  
    /*  
        some code  
    */  
    val data = deferred.await()  
    /*  
        some code  
    */  
}  
}
```

Coroutine-builder:

- всегда имеет Dispatcher
- не блокирует поток
- по умолчанию, сразу стартует корутину
- возвращает ссылка на корутину

Важно понимать

```
fun onStart() {  
    val job = launch(Dispatchers.Main) {  
        /*  
            some code  
        */  
        val deferred = async(Dispatchers.IO) {  
            /*  
                call api, get from DB, etc.  
            */  
        }  
        /*  
            some code  
        */  
        val data = deferred.await()  
        /*  
            some code  
        */  
    }  
}
```

Coroutine-builder:

- всегда имеет Dispatcher
- не блокирует поток
- по умолчанию, сразу стартует корутину
- возвращает ссылка на корутину
- ... которую можно использовать

Важно понимать

```
fun onStart() {  
    val job = launch(Dispatchers.Main) {  
        /*  
            some code  
        */  
        val deferred = async(Dispatchers.IO) {  
            /*  
                call api, get from DB, etc.  
            */  
        }  
        /*  
            some code  
        */  
        val data = deferred.await()  
        /*  
            some code  
        */  
    }  
}
```

Coroutine-builder:

- всегда имеет Dispatcher
- не блокирует поток
- по умолчанию, сразу стартует корутину
- возвращает ссылка на корутину
- ... которую можно использовать

Важно понимать

```
fun onStart() {  
    val job = launch(Dispatchers.Main) {  
        /*  
         some code  
        */  
        val deferred = async(Dispatchers.IO) {  
            /*  
             call api, get from DB, etc.  
            */  
        }  
        /*  
         some code  
        */  
        val data = deferred.await()  
        /*  
         some code  
        */  
    }  
}
```

Coroutine-builder:

- всегда имеет Dispatcher
- не блокирует поток
- по умолчанию, сразу стартует корутину
- возвращает ссылка на корутину
- ... которую можно использовать

Важно понимать

```
fun onStart() {  
    val job = launch(Dispatchers.Main) {  
        /*  
         some code  
        */  
        val deferred = async(Dispatchers.IO) {  
            /*  
             call api, get from DB, etc.  
            */  
        }  
        /*  
         some code  
        */  
        val data = deferred.await()  
        /*  
         some code  
        */  
    }  
}
```

Coroutine-builder:

- всегда имеет Dispatcher
- не блокирует поток
- по умолчанию, сразу стартует корутину
- возвращает ссылка на корутину
- ... которую можно использовать

Scope

Scope

```
class CorExample(private val repo: RepositoryCor) {  
  
    fun onStart() {  
        launch {  
            ...  
        }  
    }  
  
}
```

Scope

```
class CorExample(private val repo: RepositoryCor) {  
    fun onStart() {  
        ✘ launch {  
            ...  
        }  
    }  
}
```

Scope

```
class CorExample(private val repo: RepositoryCor) {  
    fun onStart() {  
        ✘ launch {  
            ...  
        }  
    }  
}
```

ресивер

↓

```
public fun CoroutineScope.launch(  
    ...  
    block: suspend CoroutineScope.() -> Unit  
): Job { ... }
```

CoroutineScope

CoroutineScope

- Плохо, когда корутины работают сами по себе

CoroutineScope

- Плохо, когда корутины работают сами по себе
- Идея - объединить запускаемые корутин в структуру, запуская их в рамках некоторого скоупа

CoroutineScope

- Плохо, когда корутины работают сами по себе
- Идея - объединить запускаемые корутины в структуру, запуская их в рамках некоторого скоупа
- Реализуется за счет предоставления запускаемым корутинам единого "контекста" (CoroutineContext)

```
public interface CoroutineScope {  
    /**  
     * Context of this scope.  
     */  
    public val coroutineContext: CoroutineContext  
}
```

Реализация CoroutineScope


```
class CorExample(private val repo: RepositoryCor) {  
  
    fun onStart() {  
        launch {  
            ...  
        }  
    }  
}
```

Реализация CoroutineScope

```
class CorExample(private val repo: RepositoryCor) : CoroutineScope {
    override val coroutineContext: CoroutineContext
        get() = Dispatchers.Main

    fun onStart() {
        launch {
            ...
        }
    }
}
```

Реализация CoroutineScope

```
class CorExample(private val repo: RepositoryCor) : CoroutineScope {  
    override val coroutineContext: CoroutineContext  
        get() = Dispatchers.Main  
  
    fun onStart() {  
         launch {  
            ...  
        }  
    }  
}
```

Реализация CoroutineScope

```
class CorExample(private val repo: RepositoryCor) : CoroutineScope {
    override val coroutineContext: CoroutineContext
        get() = Dispatchers.Main

    fun onStart() {
        ✓ launch {
            ...
        }
    }
}
```

Важно: у каждой корутины есть свой собственный скоуп

```
public fun CoroutineScope.launch(
    ...
    block: suspend CoroutineScope.() -> Unit
): Job { ... }
```

ресивер



Context

Два контекста

```
class CorExample(private val repo: RepositoryCor) : CoroutineScope {
    override val coroutineContext: CoroutineContext
        get() = Dispatchers.Main

    fun onStart() {
        launch(Dispatchers.IO) {
            /*
                some code
            */
        }
    }
}
```

Два контекста

```
class CorExample(private val repo: RepositoryCor) : CoroutineScope {  
    override val coroutineContext: CoroutineContext  
        get() = Dispatchers.Main  
  
    fun onStart() {  
        launch(Dispatchers.IO) {  
            /*  
                some code IO или Main?  
            */  
        }  
    }  
}
```

Два контекста и... "+"?!

```
class CorExample(private val repo: RepositoryCor) : CoroutineScope {
    override val coroutineContext: CoroutineContext
        get() = Dispatchers.Main

    fun onStart() {
        launch(Dispatchers.IO + CoroutineName("my coroutine")) {
            /*
                some code
            */
        }
    }
}
```



**СЛИШКОМ МНОГО
КОНТЕКСТОВ!**

Два контекста и... "+"?!

```
class CorExample(private val repo: RepositoryCor) : CoroutineScope {
    override val coroutineContext: CoroutineContext
        get() = Dispatchers.Main

    fun onStart() {
        launch(Dispatchers.IO + CoroutineName("my coroutine")) {
            /*
                some code 
            */
        }
    }
}
```

Два контекста и... "+"?!

```
class CorExample(private val repo: RepositoryCor) : CoroutineScope {
    override val coroutineContext: CoroutineContext
        get() = Dispatchers.Main

    fun onStart() {
        launch(Dispatchers.IO + CoroutineName("my coroutine")) {
            /*
                some code 
            */
        }
    }
}
```

- CoroutineContext - интерфейс, у него может быть много реализаций

Два контекста и... "+"?!

```
class CorExample(private val repo: RepositoryCor) : CoroutineScope {
    override val coroutineContext: CoroutineContext
        get() = Dispatchers.Main

    fun onStart() {
        launch(Dispatchers.IO + CoroutineName("my coroutine")) {
            /*
                some code 
            */
        }
    }
}
```

- CoroutineContext - интерфейс, у него может быть много реализаций
- можно ли их комбинировать?

CoroutineContext - это... map

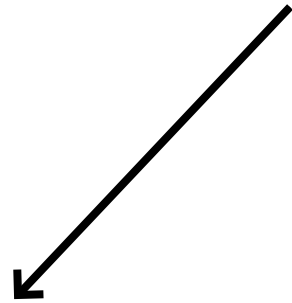
```
public interface CoroutineContext {  
    ...  
    public operator fun <E : Element> get(key: Key<E>): E?  
    ...  
}
```


CoroutineContext - это... map

```
public interface CoroutineContext {  
    ...  
    public operator fun <E : Element> get(key: Key<E>): E? ~ Map<Key<Element>, Element>  
    ...  
}
```

CoroutineContext - это... map

```
public interface CoroutineContext {  
    ...  
    public operator fun <E : Element> get(key: Key<E>): E? ~ Map<Key<Element>, Element>  
    ...  
}
```



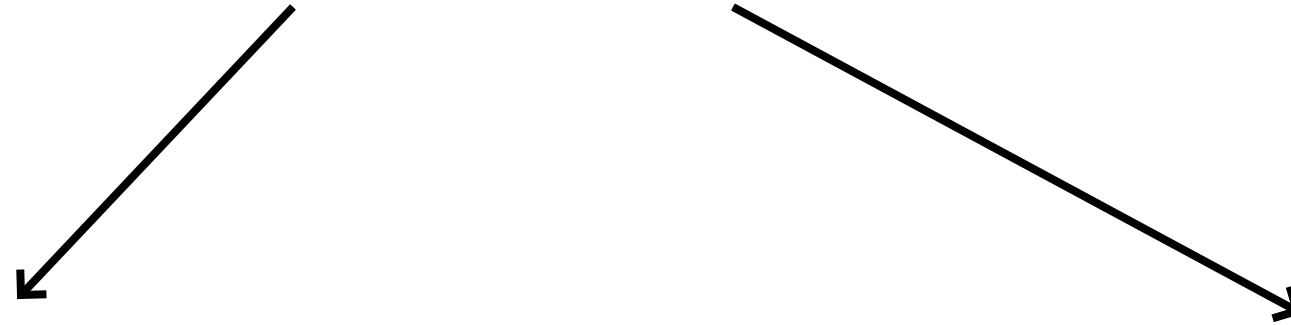
```
public interface Element : CoroutineContext {  
    ...  
    public val key: Key<*>  
  
    public override operator fun <E : Element> get(key: Key<E>): E? =  
        if (this.key == key) this as E else null  
    ...  
}
```

CoroutineContext - это... map

```
public interface CoroutineContext {  
    ...  
    public operator fun <E : Element> get(key: Key<E>): E? ~ Map<Key<Element>, Element>  
    ...  
}
```

```
public interface Element : CoroutineContext {  
    ...  
    public val key: Key<*>  
  
    public override operator fun <E : Element> get(key: Key<E>): E? =  
        if (this.key == key) this as E else null  
    ...  
}
```

```
public interface Key<E : Element>
```



Суммирование КОНТЕКСТОВ

```
public interface CoroutineContext {  
    ...  
    public operator fun plus(context: CoroutineContext): CoroutineContext { ... }  
    ...  
}
```

Суммирование КОНТЕКСТОВ

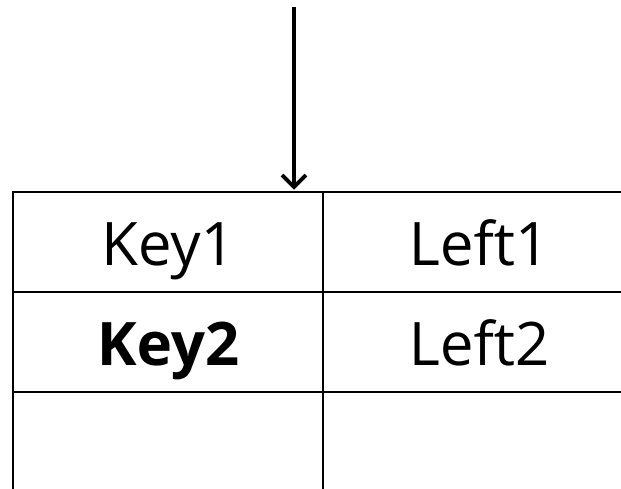
```
public interface CoroutineContext {  
    ...  
    public operator fun plus(context: CoroutineContext): CoroutineContext { ... }  
    ...  
}  
  
val contextLeft : CoroutineContext = ...  
val contextRight: CoroutineContext = ...  
  
val contextResult : CoroutineContext = contextLeft + contextRight
```

Суммирование КОНТЕКСТОВ

```
public interface CoroutineContext {  
    ...  
    public operator fun plus(context: CoroutineContext): CoroutineContext { ... }  
    ...  
}
```

```
val contextLeft : CoroutineContext = ...  
val contextRight: CoroutineContext = ...
```

```
val contextResult : CoroutineContext = contextLeft + contextRight
```



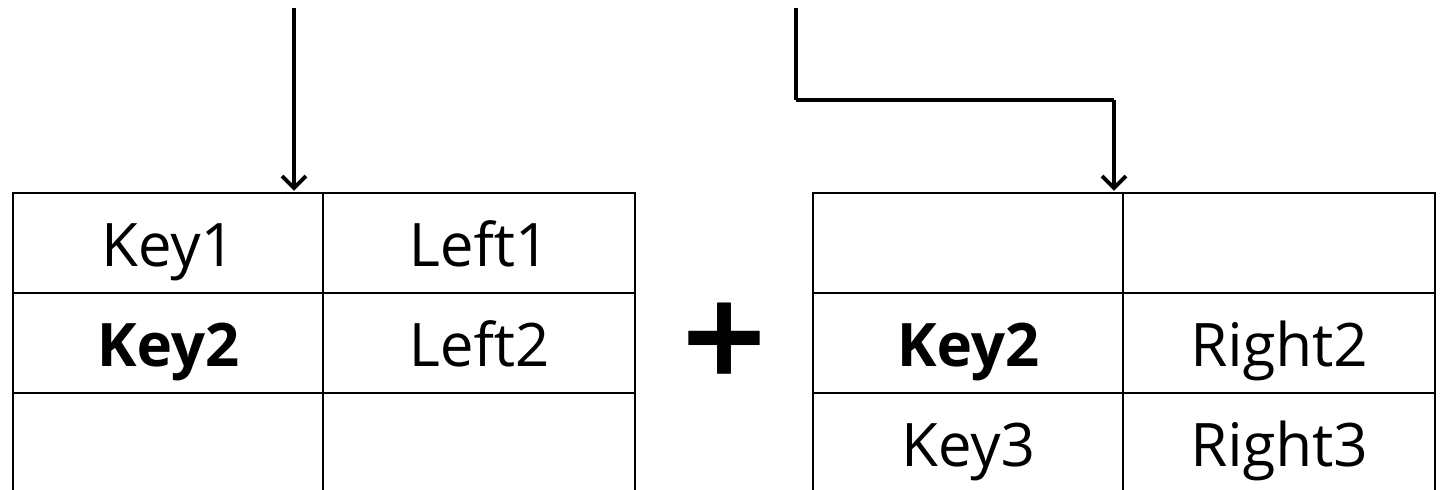
Key1	Left1
Key2	Left2

Суммирование КОНТЕКСТОВ

```
public interface CoroutineContext {  
    ...  
    public operator fun plus(context: CoroutineContext): CoroutineContext { ... }  
    ...  
}
```

```
val contextLeft : CoroutineContext = ...  
val contextRight: CoroutineContext = ...
```

```
val contextResult : CoroutineContext = contextLeft + contextRight
```

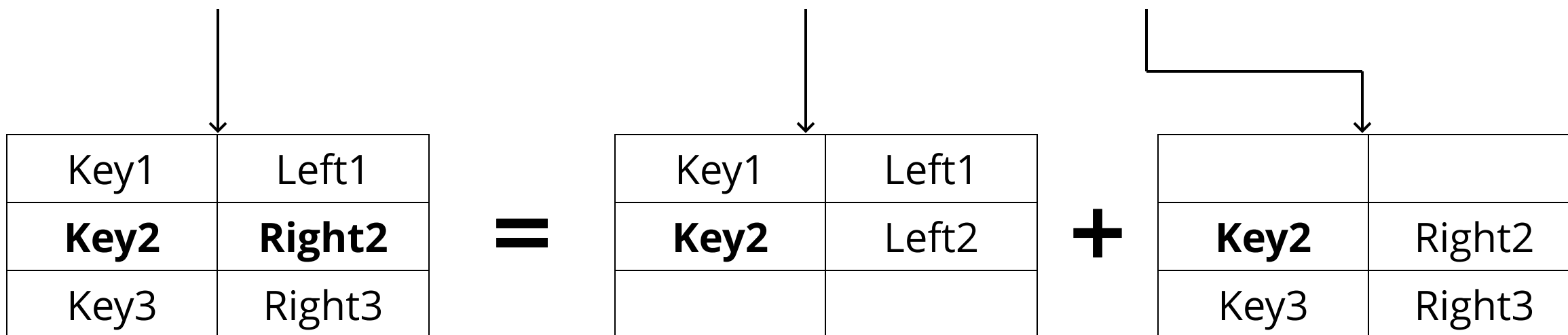


Суммирование КОНТЕКСТОВ

```
public interface CoroutineContext {  
    ...  
    public operator fun plus(context: CoroutineContext): CoroutineContext { ... }  
    ...  
}
```

```
val contextLeft : CoroutineContext = ...  
val contextRight: CoroutineContext = ...
```

```
val contextResult : CoroutineContext = contextLeft + contextRight
```



Что в итоге

```
class CorExample(private val repo: RepositoryCor) : CoroutineScope {
    override val coroutineContext: CoroutineContext
        get() = Dispatchers.Main

    fun onStart() {
        launch(Dispatchers.IO + CoroutineName("my coroutine")) {
            /*
                some code 
            */
        }
    }
}
```


Что в итоге

```
class CorExample(private val repo: RepositoryCor) : CoroutineScope {
    override val coroutineContext: CoroutineContext
    1 get() = Dispatchers.Main

    fun onStart() {
        launch(Dispatchers.IO + CoroutineName("my coroutine")) {
            /*
                some code ???
            */
        }
    }
}
```

1 Dispatchers.Main = [Dispatchers.Main] = ScopeContext

Что в итоге

```
class CorExample(private val repo: RepositoryCor) : CoroutineScope {  
    override val coroutineContext: CoroutineContext  
    1 get() = Dispatchers.Main  
  
    fun onStart() {  
    2 launch(Dispatchers.IO + CoroutineName("my coroutine")) {  
        /*  
        some code  ???  
        */  
    }  
    }  
}
```

1 Dispatchers.Main = [Dispatchers.Main] = ScopeContext

2 Dispatchers.IO + CoroutineName("my coroutine") = [Dispatchers.IO, CoroutineName] = BuilderContext

Что в итоге

```
class CorExample(private val repo: RepositoryCor) : CoroutineScope {  
    override val coroutineContext: CoroutineContext  
    1    get() = Dispatchers.Main  
  
    fun onStart() {  
    2    launch(Dispatchers.IO + CoroutineName("my coroutine")) {  
        /*  
    3        some code ???  
        */  
    }  
    }  
}
```

1 Dispatchers.Main = [Dispatchers.Main] = ScopeContext

2 Dispatchers.IO + CoroutineName("my coroutine") = [Dispatchers.IO, CoroutineName] = BuilderContext

3 ??? = ScopeContext + BuilderContext = [Dispatchers.IO, CoroutineName]

Cancellation

Отмена: простой случай

Как отменить корутину?

Отмена: простой случай

Как отменить корутину?

Очень просто:

```
val job = launch {  
    // some code  
}  
val deferred = async {  
    // some code  
}  
job.cancel()  
deferred.cancel()
```

Отмена: усложняем

Отмена: усложняем

Как отменить несколько корутин?

```
val job1 = launch {  
    // some code  
}  
...  
val jobN = launch {  
    // some code  
}  
job1.cancel()  
...  
jobN.cancel()
```

Отмена: усложняем

Как отменить несколько корутин?

```
val job1 = launch {  
    // some code  
}  
...  
val jobN = launch {  
    // some code  
}  
job1.cancel()  
...  
jobN.cancel()
```

А что с вложенными корутинами?

```
val job = launch {  
    val deferred1 = async {  
        // some code  
    }  
    val deferred2 = async {  
        // some code  
    }  
}  
job.cancel()  
deferred1.cancel() // ???  
deferred2.cancel() // ???
```

Что нужно знать о Job

Что нужно знать о Job

- каждая корутина - это Job

```
public abstract class AbstractCoroutine<in T>(
    @JvmField
    protected val parentContext: CoroutineContext,
    active: Boolean = true
) : JobSupport(active), Job, Continuation<T>, CoroutineScope {
    ...
}
```

Что нужно знать о Job

- каждая корутина - это Job
- поддерживает parent-child отношение
 - завершается только когда дочерние элементы завершатся
 - при отмене, отменяет все дочерние элементы

```
public interface Job : CoroutineContext.Element {  
    ...  
    public val children: Sequence<Job>  
  
    public fun attachChild(child: ChildJob): ChildHandle  
    ...  
}
```

Что нужно знать о Job

- каждая корутина - это Job
- поддерживает parent-child отношение
 - завершается только когда дочерние элементы завершатся
 - при отмене, отменяет все дочерние элементы
- созданная корутина становится дочерней для job в родительском контексте

```
public interface Job : CoroutineContext.Element {  
    ...  
    public val children: Sequence<Job>  
  
    public fun attachChild(child: ChildJob): ChildHandle  
    ...  
}
```

Вложенные корутины

```
val job = launch { // this : CoroutineScope
    val deferred1 = this.async {
        // some code
    }
    val deferred2 = this.async {
        // some code
    }
}
```

Вложенные корутины

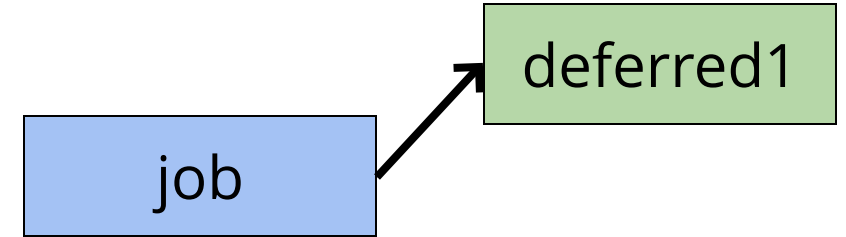
```
val job = launch { // this : CoroutineScope
    val deferred1 = this.async {
        // some code
    }
    val deferred2 = this.async {
        // some code
    }
}
```



job

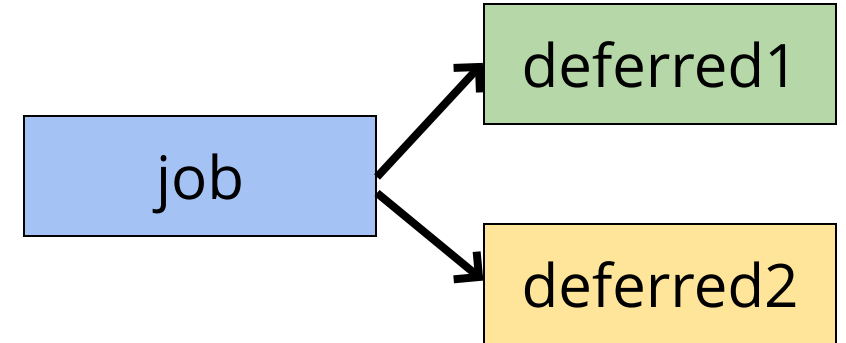
Вложенные корутины

```
val job = launch { // this : CoroutineScope
    val deferred1 = this.async {
        // some code
    }
    val deferred2 = this.async {
        // some code
    }
}
```



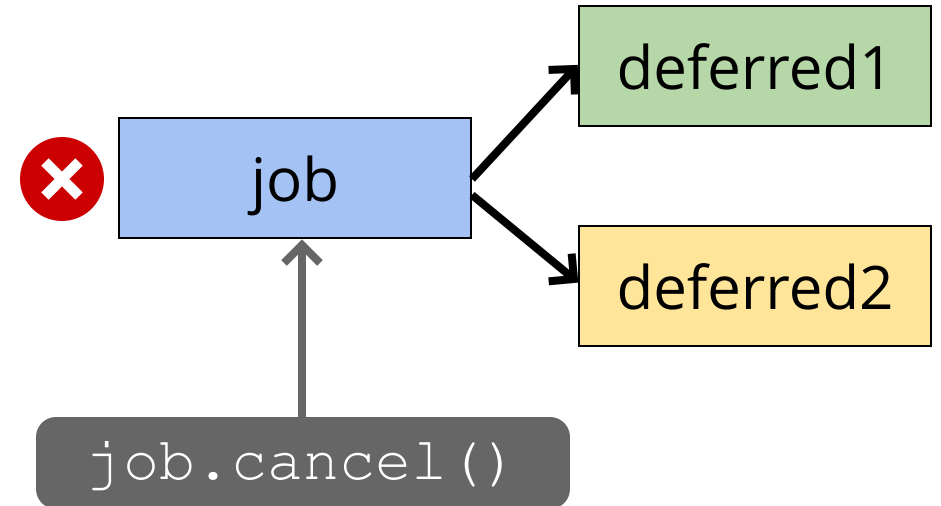
Вложенные корутины

```
val job = launch { // this : CoroutineScope
    val deferred1 = this.async {
        // some code
    }
    val deferred2 = this.async {
        // some code
    }
}
```



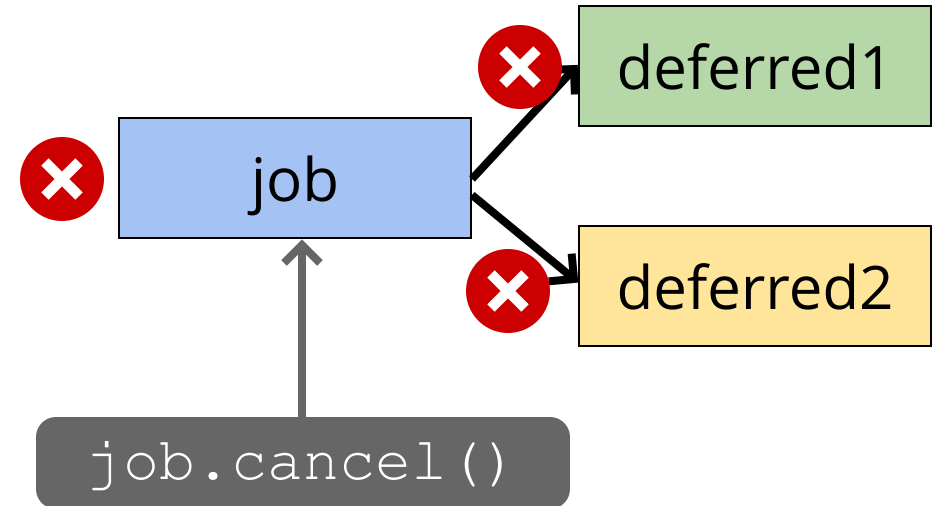
Вложенные корутины

```
val job = launch { // this : CoroutineScope
    val deferred1 = this.async {
        // some code
    }
    val deferred2 = this.async {
        // some code
    }
}
```



Вложенные корутины

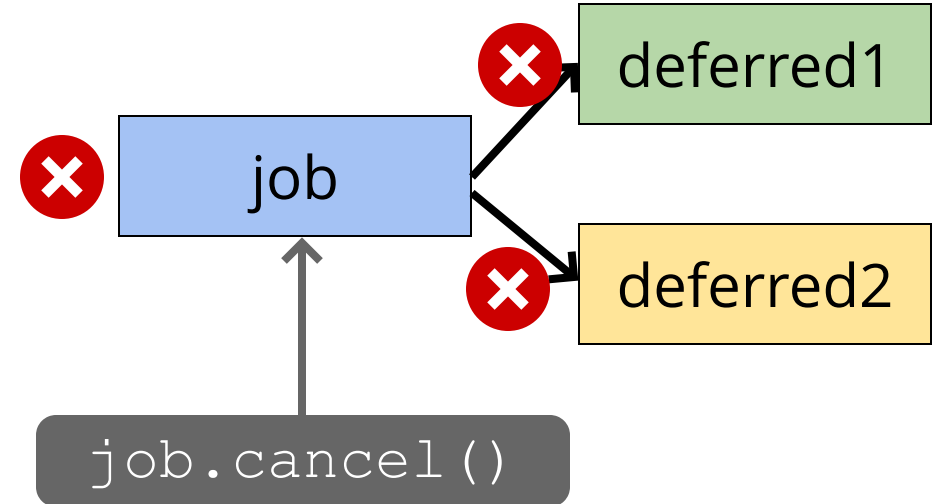
```
val job = launch { // this : CoroutineScope
    val deferred1 = this.async {
        // some code
    }
    val deferred2 = this.async {
        // some code
    }
}
```



Вложенные корутины

```
val job = launch { // this : CoroutineScope
    val deferred1 = this.async {
        // some code
    }
    val deferred2 = this.async {
        // some code
    }
    ...
}

val jobN = launch {
    val deferredN_1 = async {
        // some code
    }
    val deferredN_2 = async {
        // some code
    }
}
```



Каждое дерево нужно отменять?!

Скоуп нам в ПОМОЩЬ

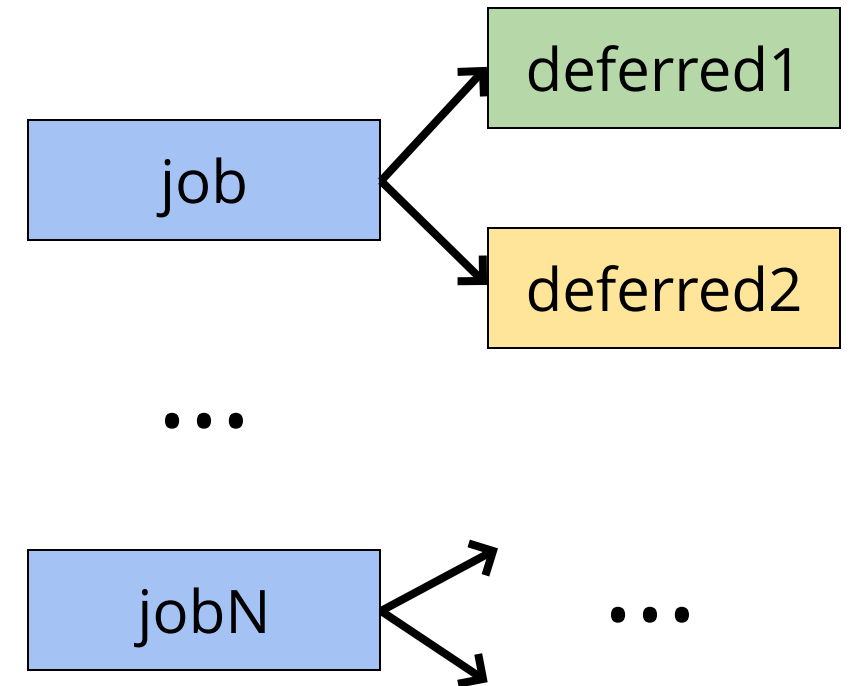
```
class CorExample : CoroutineScope {
    private var rootJob: Job = Job()
    override val coroutineContext: CoroutineContext
        get() = Dispatchers.Main + rootJob

    fun onStart() {
        val job = launch {
            val deferred1 = async {
                // some code
            }
            val deferred2 = async {
                // some code
            }
        }
        ...
        val jobN = launch {
            ...
        }
    }
}
```

Скоуп нам в ПОМОЩЬ

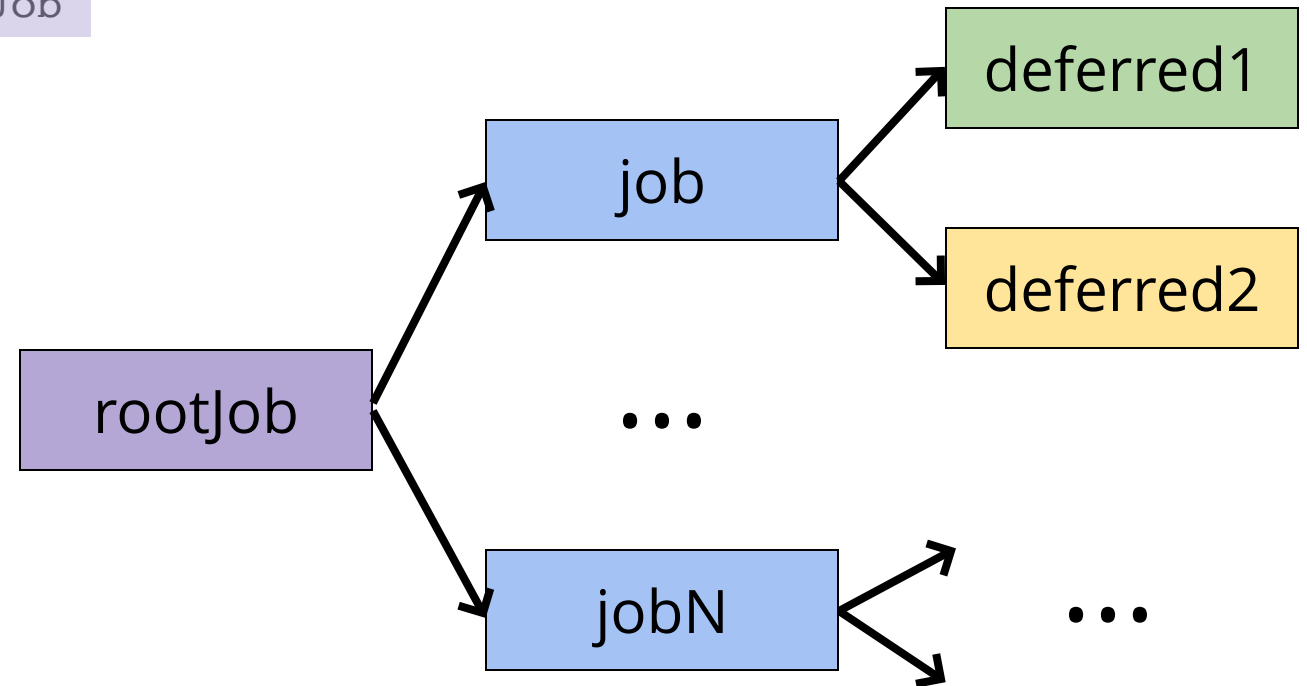
```
class CorExample : CoroutineScope {
    private var rootJob: Job = Job()
    override val coroutineContext: CoroutineContext
        get() = Dispatchers.Main + rootJob

    fun onStart() {
        val job = launch {
            val deferred1 = async {
                // some code
            }
            val deferred2 = async {
                // some code
            }
        }
        ...
        val jobN = launch {
            ...
        }
    }
}
```



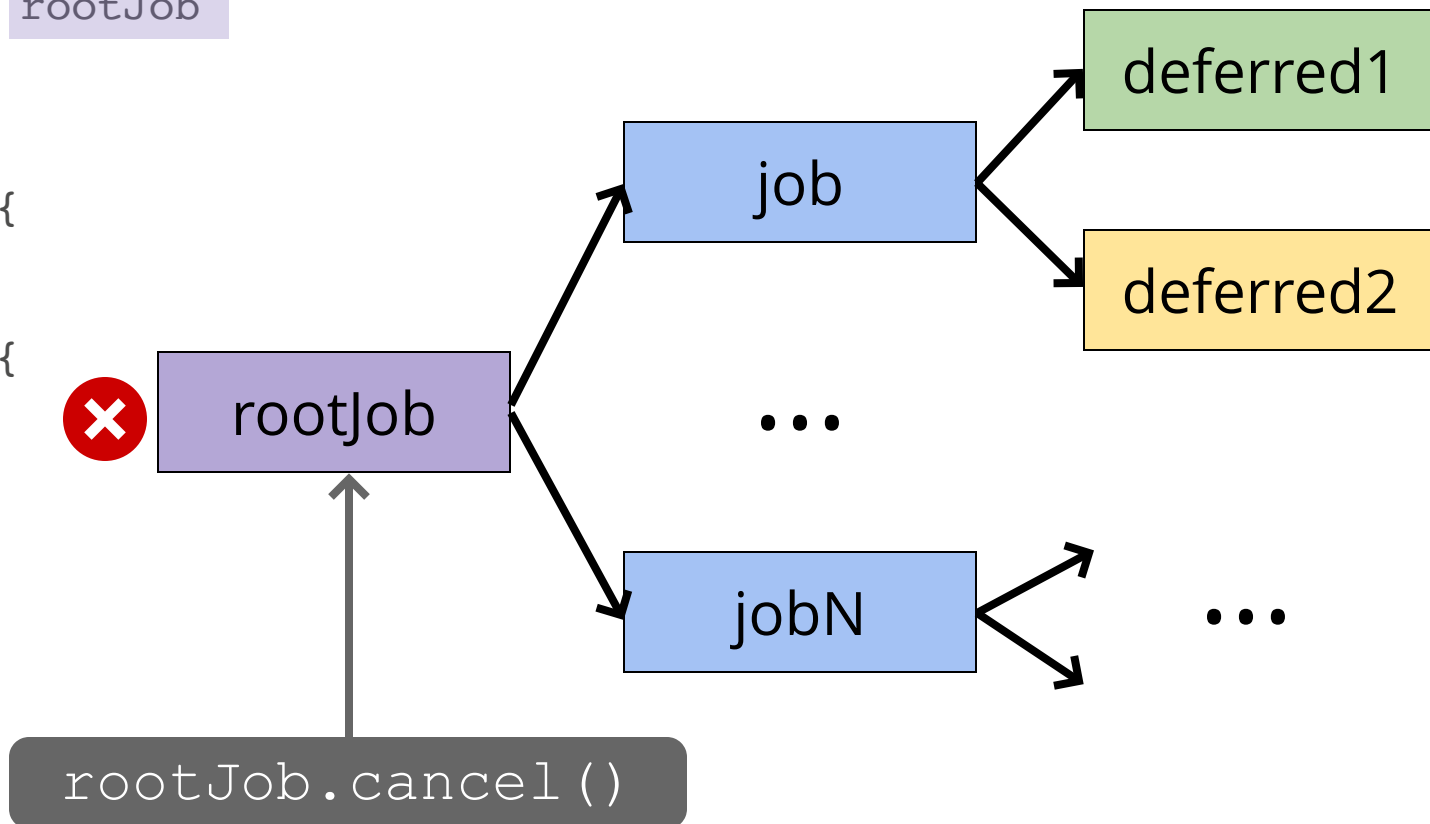
Скоуп нам в ПОМОЩЬ

```
class CorExample : CoroutineScope {  
    private var rootJob: Job = Job()  
    override val coroutineContext: CoroutineContext  
        get() = Dispatchers.Main + rootJob  
  
    fun onStart() {  
        val job = launch {  
            val deferred1 = async {  
                // some code  
            }  
            val deferred2 = async {  
                // some code  
            }  
        }  
        ...  
        val jobN = launch {  
            ...  
        }  
    }  
}
```



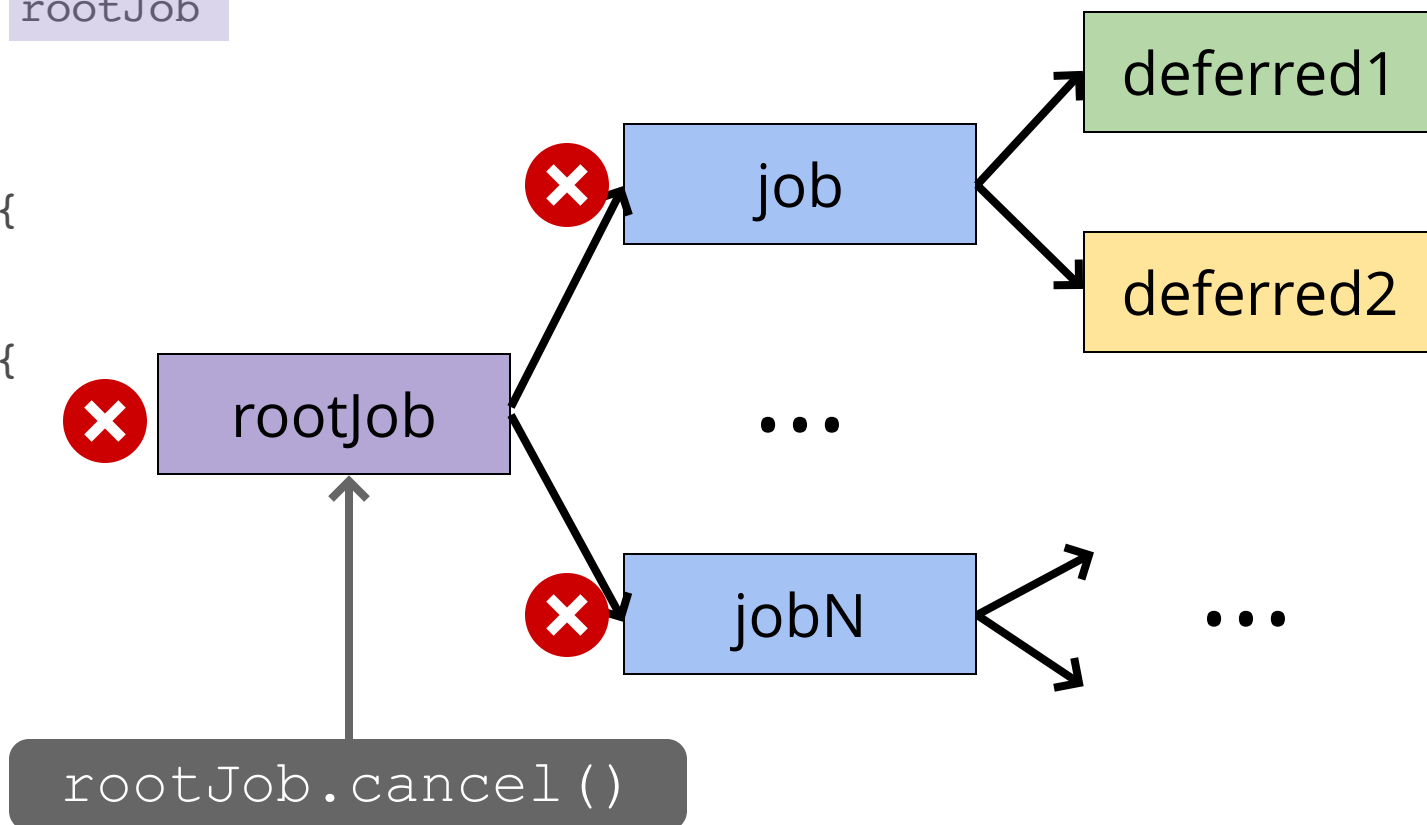
Скоуп нам в ПОМОЩЬ

```
class CorExample : CoroutineScope {  
    private var rootJob: Job = Job()  
    override val coroutineContext: CoroutineContext  
        get() = Dispatchers.Main + rootJob  
  
    fun onStart() {  
        val job = launch {  
            val deferred1 = async {  
                // some code  
            }  
            val deferred2 = async {  
                // some code  
            }  
        }  
        ...  
        val jobN = launch {  
            ...  
        }  
    }  
}
```



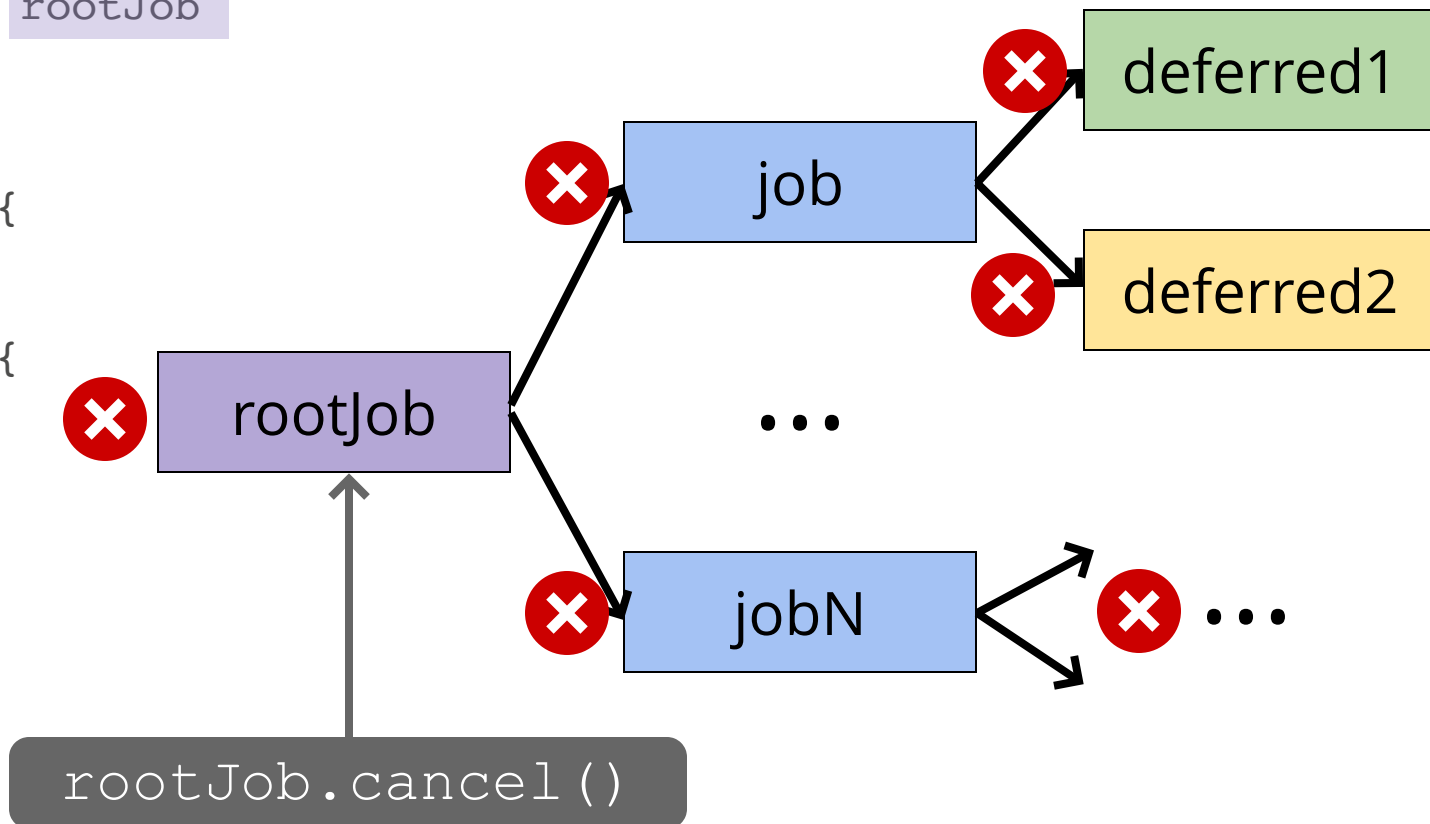
Скоуп нам в ПОМОЩЬ

```
class CorExample : CoroutineScope {  
    private var rootJob: Job = Job()  
    override val coroutineContext: CoroutineContext  
        get() = Dispatchers.Main + rootJob  
  
    fun onStart() {  
        val job = launch {  
            val deferred1 = async {  
                // some code  
            }  
            val deferred2 = async {  
                // some code  
            }  
        }  
        ...  
        val jobN = launch {  
            ...  
        }  
    }  
}
```



Скоуп нам в помощь

```
class CorExample : CoroutineScope {  
    private var rootJob: Job = Job()  
    override val coroutineContext: CoroutineContext  
        get() = Dispatchers.Main + rootJob  
  
    fun onStart() {  
        val job = launch {  
            val deferred1 = async {  
                // some code  
            }  
            val deferred2 = async {  
                // some code  
            }  
        }  
        ...  
        val jobN = launch {  
            ...  
        }  
    }  
}
```





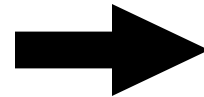
ГДЕ АРХИТЕКТУРА?

Как вынести в `suspending`-функцию?

```
fun onStart() {  
    launch {  
        val deferred = async(Dispatchers.IO){  
            repo.getUser()  
        }  
        val user = deferred.await()  
        showUser(user)  
    }  
}
```

Как вынести в `suspending`-функцию?

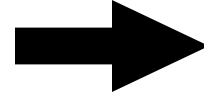
```
fun onStart() {  
    launch {  
        val deferred = async(Dispatchers.IO){  
            repo.getUser()  
        }  
        val user = deferred.await()  
        showUser(user)  
    }  
}
```



```
fun onStart() {  
    launch {  
        showUser(smith.getWithSuspend())  
    }  
}
```

Как вынести в `suspending`-функцию?

```
fun onStart() {  
    launch {  
        val deferred = async(Dispatchers.IO){  
            repo.getUser()  
        }  
        val user = deferred.await()  
        showUser(user)  
    }  
}
```

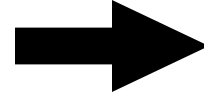


```
fun onStart() {  
    launch {  
        showUser(something.getWithSuspend())  
    }  
}
```

✓ suspend fun getWithSuspend() : User {
 val deferred = ????.async(Dispatchers.IO){
 repo.getUser()
 }
 return deferred.await()
}

Как вынести в `suspending`-функцию?

```
fun onStart() {  
    launch {  
        val deferred = async(Dispatchers.IO){  
            repo.getUser()  
        }  
        val user = deferred.await()  
        showUser(user)  
    }  
}
```



```
fun onStart() {  
    launch {  
        showUser(something.getWithSuspend())  
    }  
}
```



```
suspend fun getWithSuspend() : User {  
    val deferred = ????.async(Dispatchers.IO){  
        repo.getUser()  
    }  
    return deferred.await()  
}
```



Получаем scope в suspending-функциях

```
suspend fun getWithSuspend(scope: CoroutineScope): User {  
    val differed = scope.async(Dispatchers.IO) {  
        repo.getUser()  
    }  
    return differed.await()  
}
```

Получаем scope в suspending-функциях

```
suspend fun getWithSuspend(scope: CoroutineScope): User {  
    val differed = scope.async(Dispatchers.IO) {  
        repo.getUser()  
    }  
    return differed.await()  
}
```

```
suspend fun CoroutineScope.getWithSuspend(): User {  
    val differed = async(Dispatchers.IO) {  
        repo.getUser()  
    }  
    return differed.await()  
}
```

Получаем scope в suspending-функциях

```
suspend fun getWithSuspend(scope: CoroutineScope): User {  
    val differed = scope.async(Dispatchers.IO) {  
        repo.getUser()  
    }  
    return differed.await()  
}
```


```
suspend fun CoroutineScope.getWithSuspend(): User {  
    val differed = async(Dispatchers.IO) {  
        repo.getUser()  
    }  
    return differed.await()  
}
```

```
suspend fun CoroutineScope.getWithSuspend(): String {
```


Ambiguous coroutineContext due to CoroutineScope receiver of suspend function [more...](#) (Ctrl+F1)

```
    }  
    return differed.await()  
}
```

Получаем scope в suspending-функциях



```
suspend fun getWithSuspend(scope: CoroutineScope): User {  
    val differed = scope.async(Dispatchers.IO) {  
        repo.getUser()  
    }  
    return differed.await()  
}
```



```
suspend fun CoroutineScope.getWithSuspend(): User {  
    val differed = async(Dispatchers.IO) {  
        repo.getUser()  
    }  
    return differed.await()  
}
```

```
suspend fun CoroutineScope.getWithSuspend(): String {
```

Ambiguous coroutineContext due to CoroutineScope receiver of suspend function [more...](#) (Ctrl+F1)

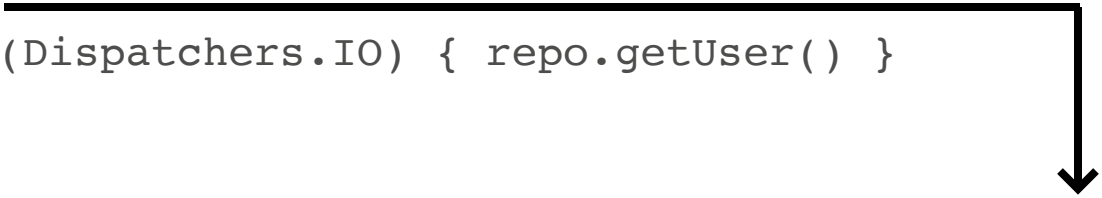
```
    }  
    return differed.await()  
}
```

Получаем `scope` в `suspending`-функциях

```
suspend fun getWithSuspend(): User {  
    return coroutineScope {  
        val differed = async(Dispatchers.IO) { repo.getUser() }  
        differed.await()  
    }  
}
```

Получаем scope в suspending-функциях

```
suspend fun getWithSuspend(): User {  
    return coroutineScope {  
        val differed = async(Dispatchers.IO) { repo.getUser() }  
        differed.await()  
    }  
}
```




```
public suspend fun <R> coroutineScope(  
    block: suspend CoroutineScope.() -> R  
) : R
```

Получаем scope в suspending-функциях




```
suspend fun getWithSuspend(): User {  
    return coroutineScope {  
        val differed = async(Dispatchers.IO) { repo.getUser() }  
        differed.await()  
    }  
}
```



```
public suspend fun <R> coroutineScope(  
    block: suspend CoroutineScope.() -> R  
) : R
```

Получаем scope в suspending-функциях



```
suspend fun getWithSuspend(): User {  
    return coroutineScope {  
        val differed = async(Dispatchers.IO) { repo.getUser() }  
        differed.await()  
    }  
}
```

```
public suspend fun <R> coroutineScope(  
    block: suspend CoroutineScope.() -> R  
) : R
```

```
suspend fun getWithSuspend(): User {  
    return withContext(Dispatchers.IO) {  
        repo.getUser()  
    }  
}
```


```
public suspend fun <T> withContext(  
    context: CoroutineContext,  
    block: suspend CoroutineScope.() -> T  
) : T
```


Получаем scope в suspending-функциях



```
suspend fun getWithSuspend(): User {  
    return coroutineScope {  
        val differed = async(Dispatchers.IO) { repo.getUser() }  
        differed.await()  
    }  
}
```

```
public suspend fun <R> coroutineScope(  
    block: suspend CoroutineScope.() -> R  
) : R
```



```
suspend fun getWithSuspend(): User {  
    return withContext(Dispatchers.IO) {  
        repo.getUser()  
    }  
}
```


```
public suspend fun <T> withContext(  
    context: CoroutineContext,  
    block: suspend CoroutineScope.() -> T  
) : T
```

Получаем scope в suspending-функциях

 `suspend fun getWithSuspend(): User {
 return coroutineScope {
 val differed = async(Dispatchers.IO) { repo.getUser() }
 differed.await()
 }
}`

СКОУП, ИЗ КОТОРОГО ВЫЗЫВАЕТСЯ
suspending-функция

`public suspend fun <R> coroutineScope(
 block: suspend CoroutineScope.() -> R
) : R`

 `suspend fun getWithSuspend(): User {
 return withContext(Dispatchers.IO) {
 repo.getUser()
 }
}`

`public suspend fun <T> withContext(
 context: CoroutineContext,
 block: suspend CoroutineScope.() -> T
) : T`

Обработка ошибок

CoroutineExceptionHandler

CoroutineExceptionHandler

- интерфейс, элемент контекста

```
public interface CoroutineExceptionHandler : CoroutineContext.Element {  
    public companion object Key : CoroutineContext.Key<CoroutineExceptionHandler>  
    public fun handleException(context: CoroutineContext, exception: Throwable)  
}
```

CoroutineExceptionHandler

- интерфейс, элемент контекста
- обрабатывает исключения, брошенные внутри корутины или дочерними элементами

```
public interface CoroutineExceptionHandler : CoroutineContext.Element {  
    public companion object Key : CoroutineContext.Key<CoroutineExceptionHandler>  
    public fun handleException(context: CoroutineContext, exception: Throwable)  
}
```

CoroutineExceptionHandler

- интерфейс, элемент контекста
- обрабатывает исключения, брошенные внутри корутины или дочерними элементами
- есть одноименная функция для получения экземпляра

```
public interface CoroutineExceptionHandler : CoroutineContext.Element {  
    public companion object Key : CoroutineContext.Key<CoroutineExceptionHandler>  
  
    public fun handleException(context: CoroutineContext, exception: Throwable)  
}
```

```
public inline fun CoroutineExceptionHandler(  
    crossinline handler: (CoroutineContext, Throwable) -> Unit  
): CoroutineExceptionHandler = {...}
```

Нельзя так просто взять ...



и отловить ошибку в корутине

Попробуем обработать ошибку

```
1 fun onStart() {
2     launch(handler) {
3         println("Start C1")
4         repeat(5) {
5             delay(1000)
6             println("tick $it")
7         }
8     }
9     launch(handler) {
10        println("Start C2")
11        delay(3000)
12        println("C2: throw exception")
13        errorCall()
14    }
15 }
```

Попробуем обработать ошибку

```
1 fun onStart() {
2     launch(handler) {
3         println("Start C1")
4         repeat(5) {
5             delay(1000)
6             println("tick $it")
7         }
8     }
9     launch(handler) {
10        println("Start C2")
11        delay(3000)
12        println("C2: throw exception")
13        errorCall()
14    }
15 }
```

Попробуем обработать ошибку

```
1 fun onStart() {
2     launch(handler) {
3         println("Start C1")
4         repeat(5) {
5             delay(1000)
6             println("tick $it")
7         }
8     }
9     launch(handler) {
10        println("Start C2")
11        delay(3000)
12        println("C2: throw exception")
13        errorCall()
14    }
15 }
```

```
val handler = CoroutineExceptionHandler { _, throwable ->
    println("handle error (${throwable.message})")
}
```

Попробуем обработать ошибку

```
1 fun onStart() {
2     launch(handler) {
3         println("Start C1")
4         repeat(5) {
5             delay(1000)
6             println("tick $it")
7         }
8     }
9     launch(handler) {
10        println("Start C2")
11        delay(3000)
12        println("C2: throw exception")
13        errorCall()
14    }
15 }
```

```
val handler = CoroutineExceptionHandler { _, throwable ->
    println("handle error (${throwable.message})")
}
```

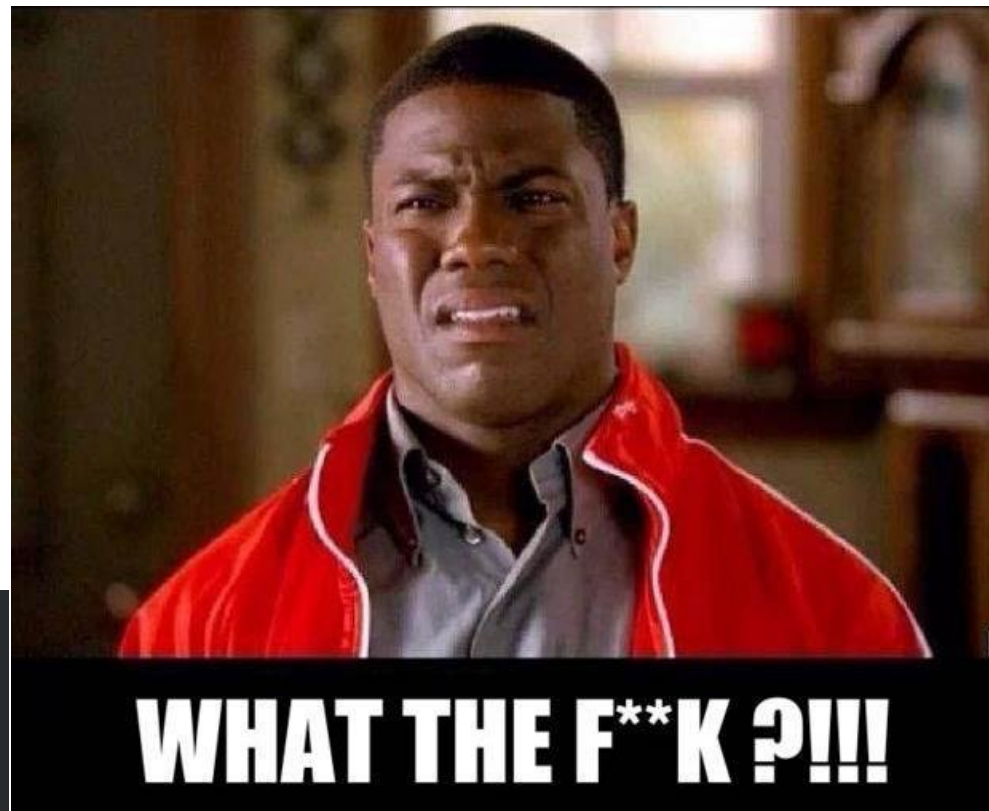
```
18:27:56.861 I/System.out: Start C1
18:27:56.866 I/System.out: Start C2
18:27:57.867 I/System.out: tick 0
18:27:58.869 I/System.out: tick 1
18:27:59.869 I/System.out: C2: throw exception
18:27:59.870 I/System.out: handle error (some error happened)
18:27:59.872 I/System.out: handle error (some error happened)
// no more ticks :(
```

Попробуем обработать ошибку

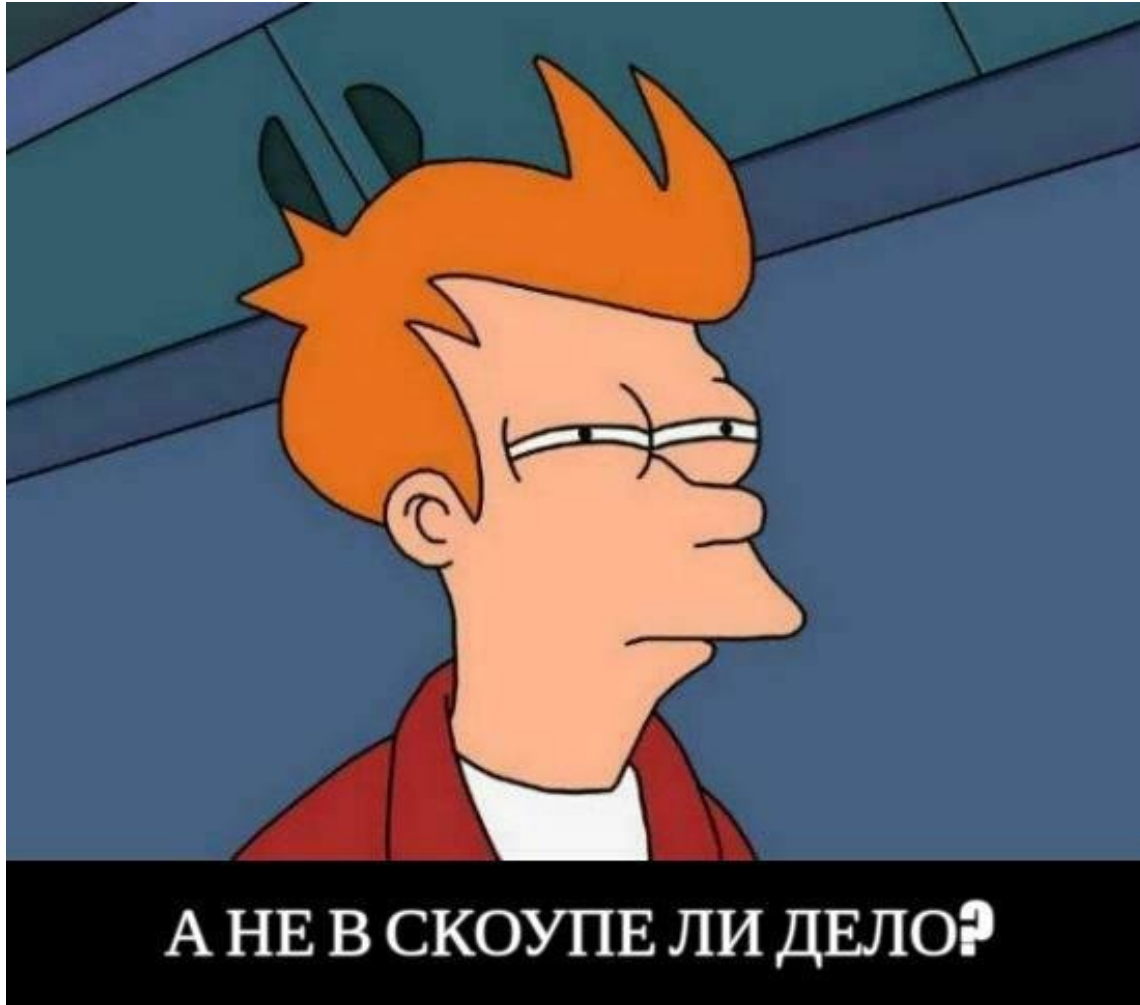
```
1 fun onStart() {
2     launch(handler) {
3         println("Start C1")
4         repeat(5) {
5             delay(1000)
6             println("tick $it")
7         }
8     }
9     launch(handler) {
10        println("Start C2")
11        delay(3000)
12        println("C2: throw exception")
13        errorCall()
14    }
15 }
```

```
val handler = CoroutineExceptionHandler { _, throwable ->
    println("handle error (${throwable.message})")
}
```

```
18:27:56.861 I/System.out: Start C1
18:27:56.866 I/System.out: Start C2
18:27:57.867 I/System.out: tick 0
18:27:58.869 I/System.out: tick 1
18:27:59.869 I/System.out: C2: throw exception
18:27:59.870 I/System.out: handle error (some error happened)
18:27:59.872 I/System.out: handle error (some error happened)
// no more ticks :(
```

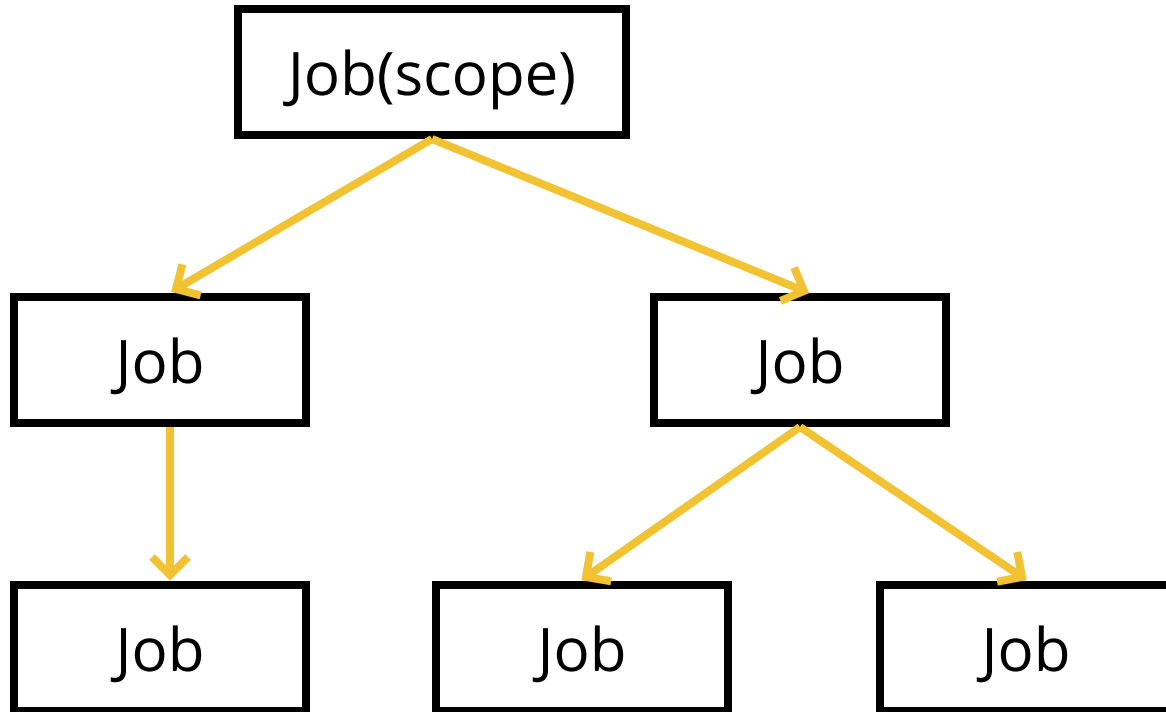


Попробуем обработать ошибку

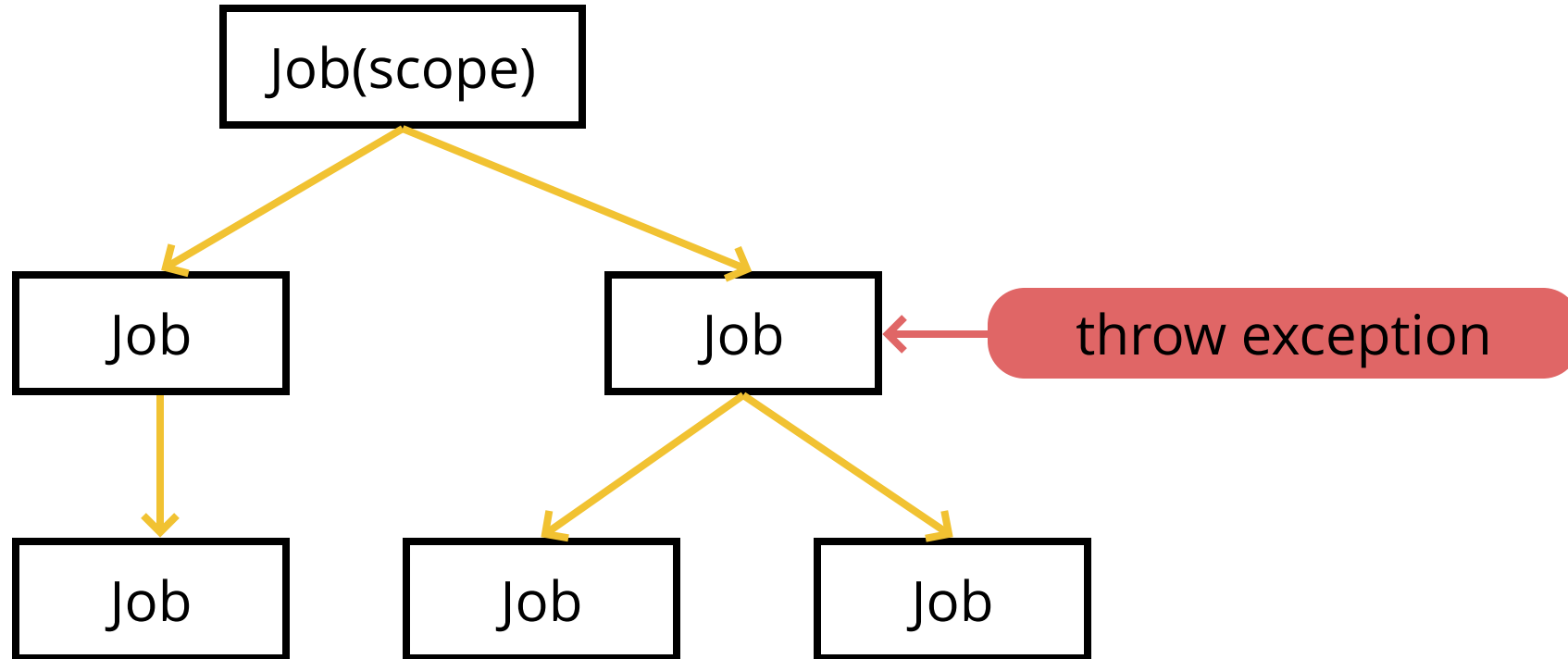


```
class CorExample() : CoroutineScope {  
    private var rootJob: Job = Job()  
    override val coroutineContext: CoroutineContext  
        get() = Dispatchers.Main + rootJob  
  
    fun onStart() {  
        ...  
    }  
}
```

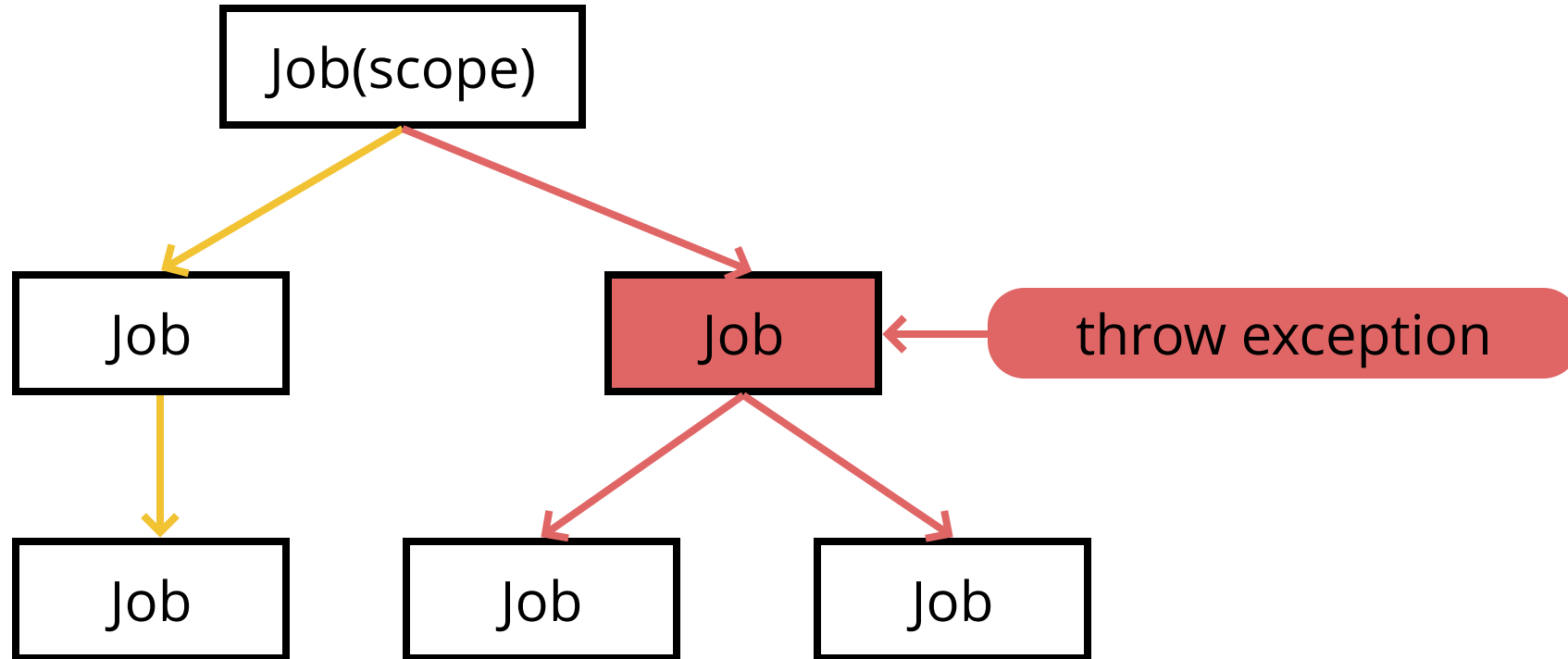
Error propagating



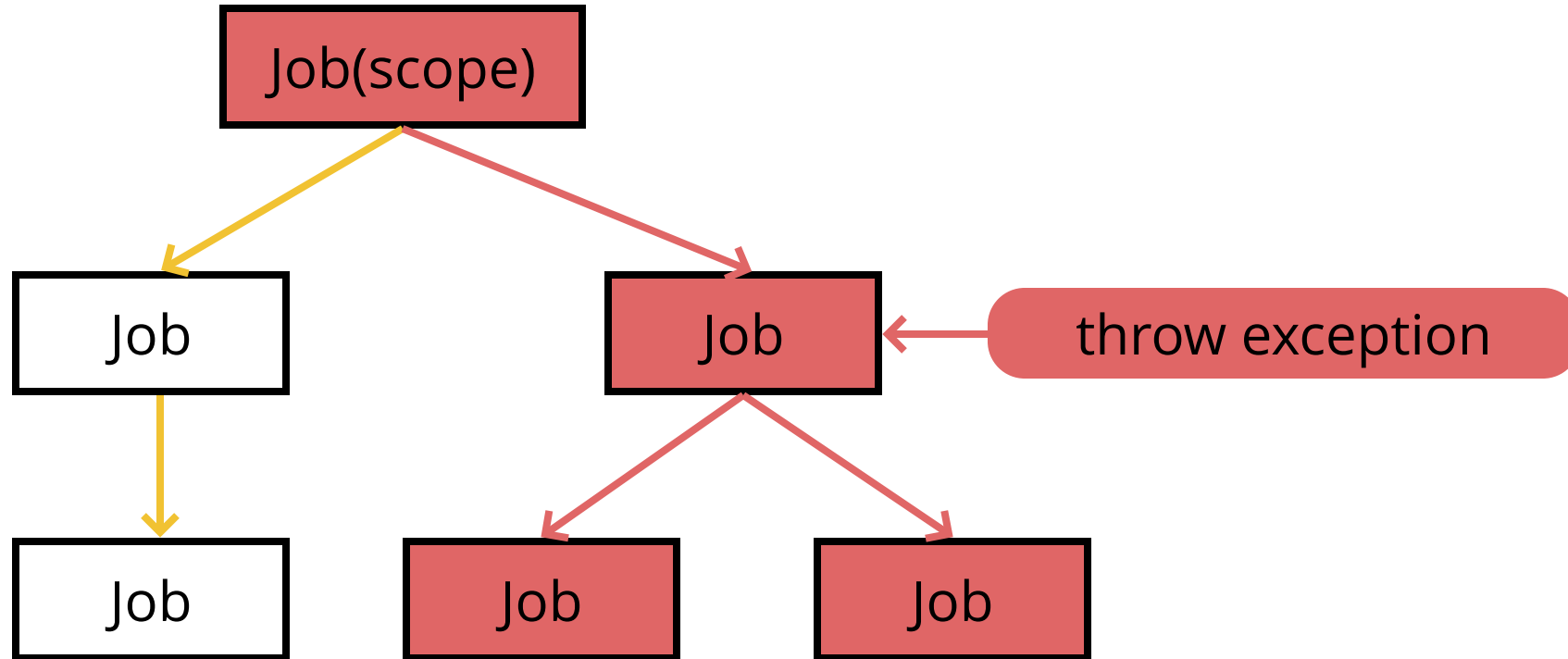
Error propagating



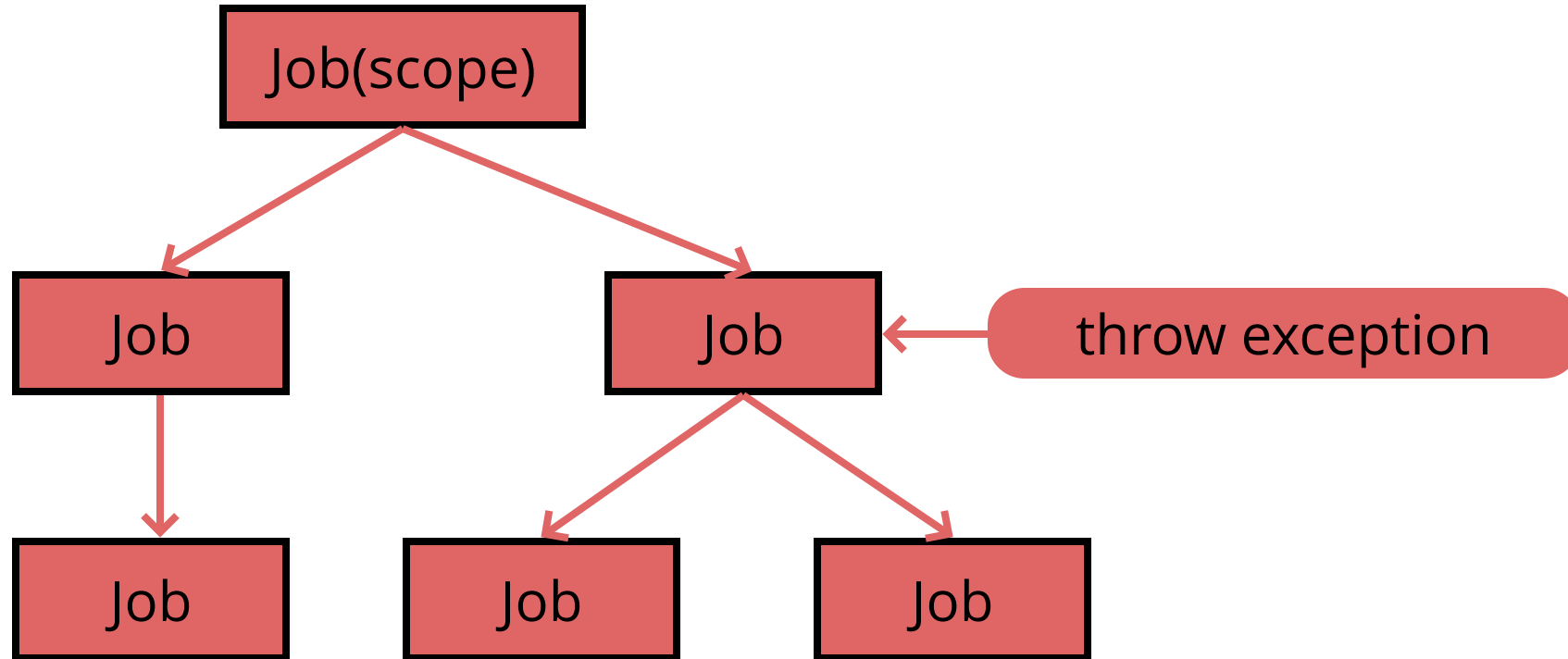
Error propagating



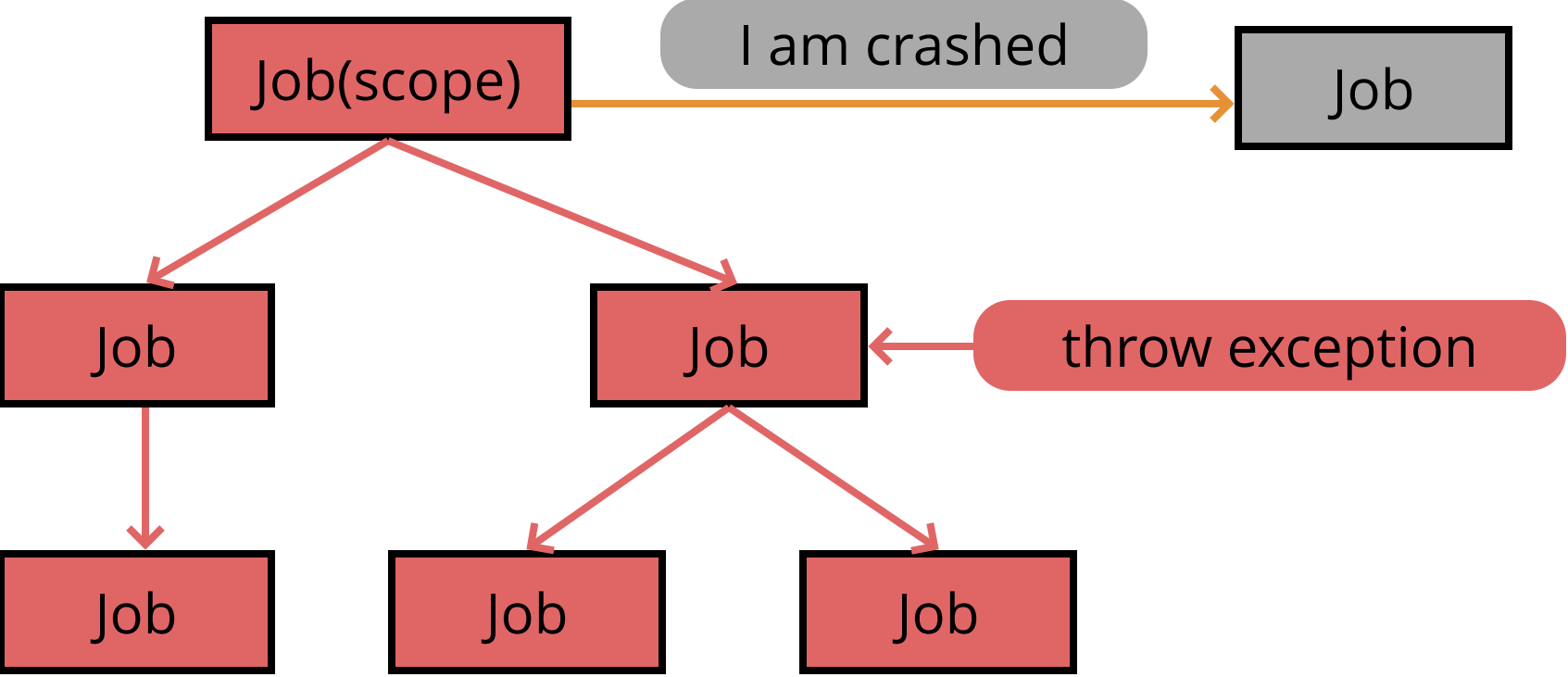
Error propagating



Error propagating



Error propagating

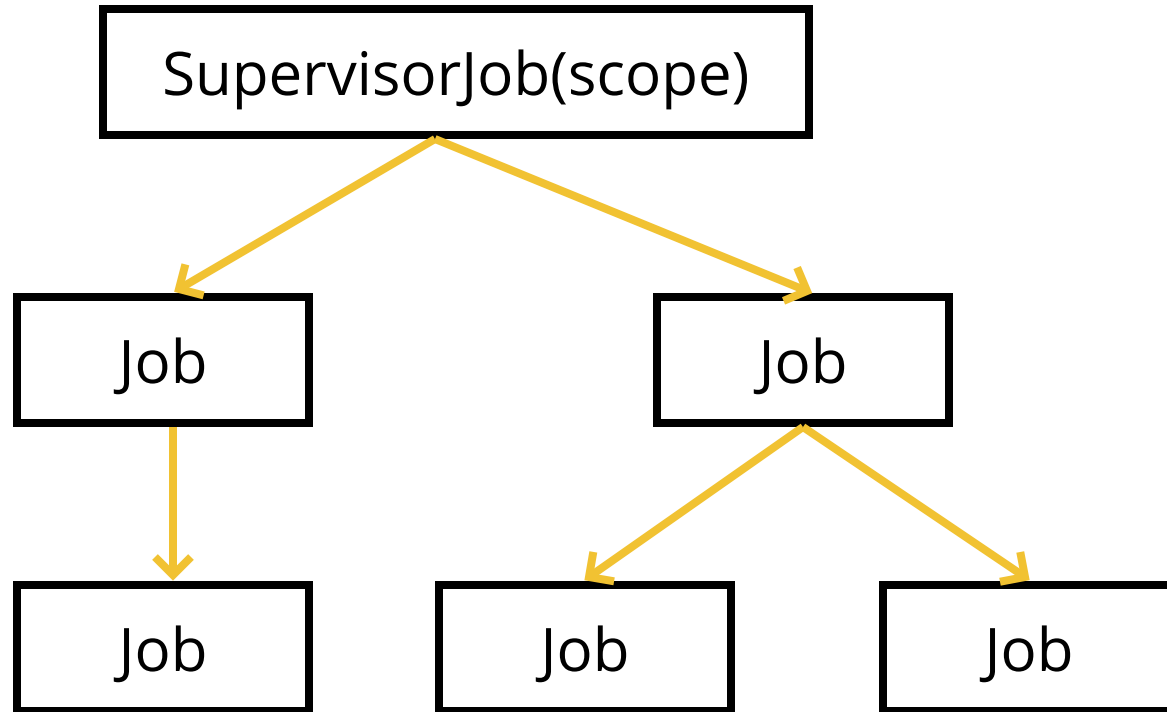




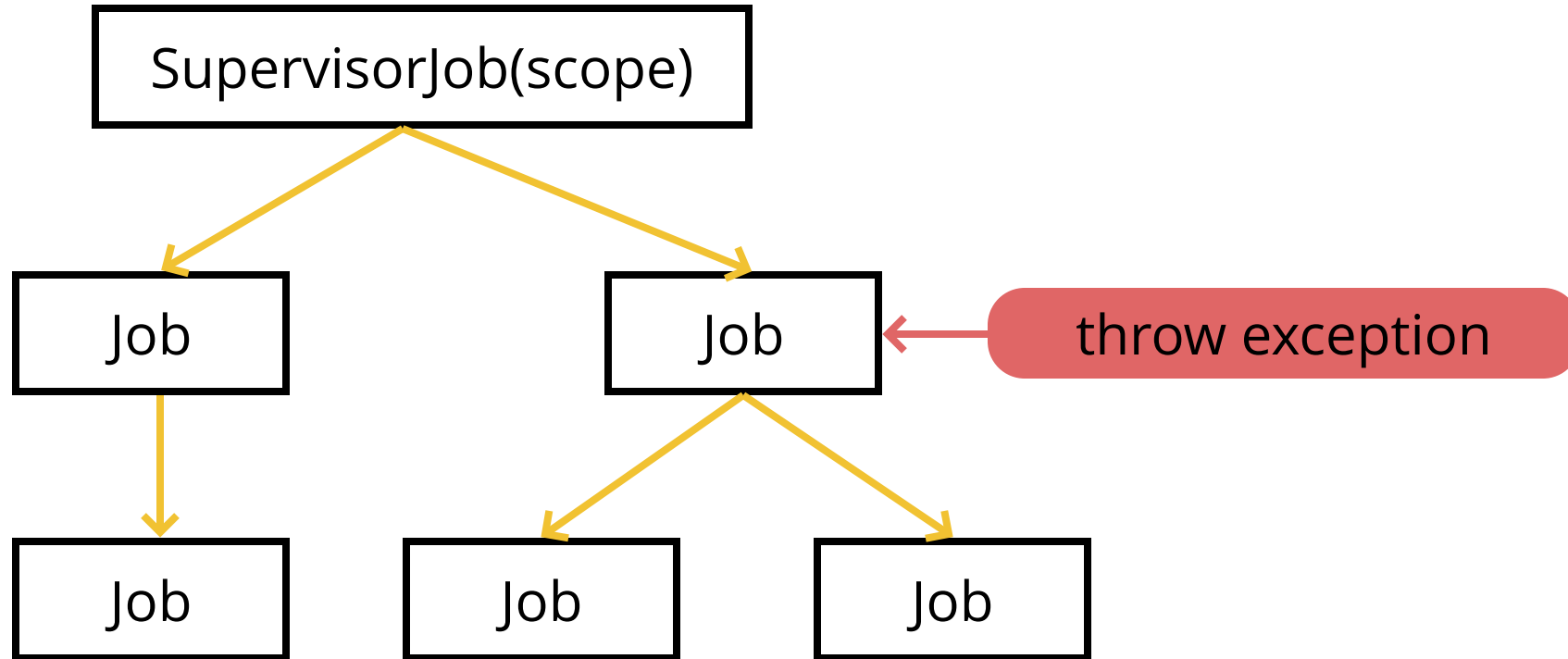
SupervisorJob

```
class CorExample(private val repo: RepositoryCor) : CoroutineScope {  
    private var rootJob: Job = SupervisorJob()  
    override val coroutineContext: CoroutineContext  
        get() = Dispatchers.Main + rootJob  
  
    ...  
  
}
```

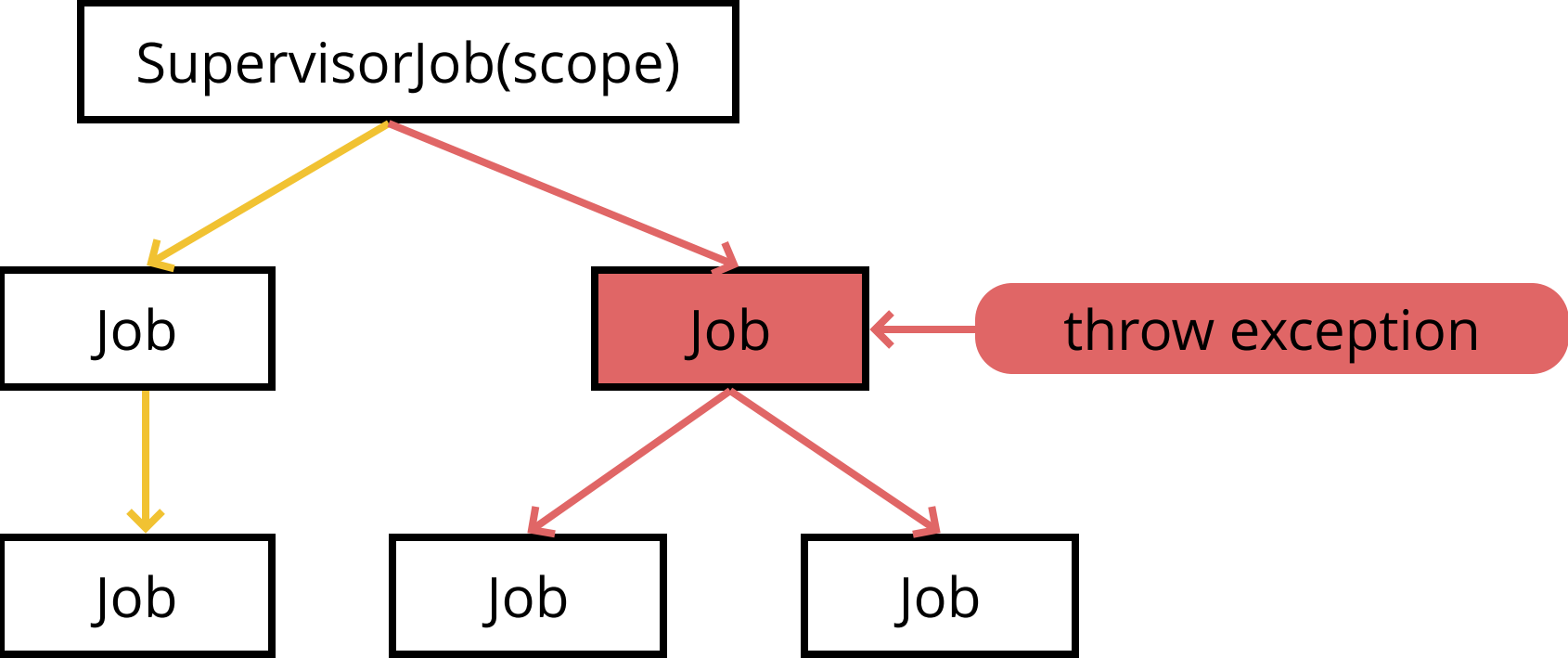
Error propagating: SupervisorJob



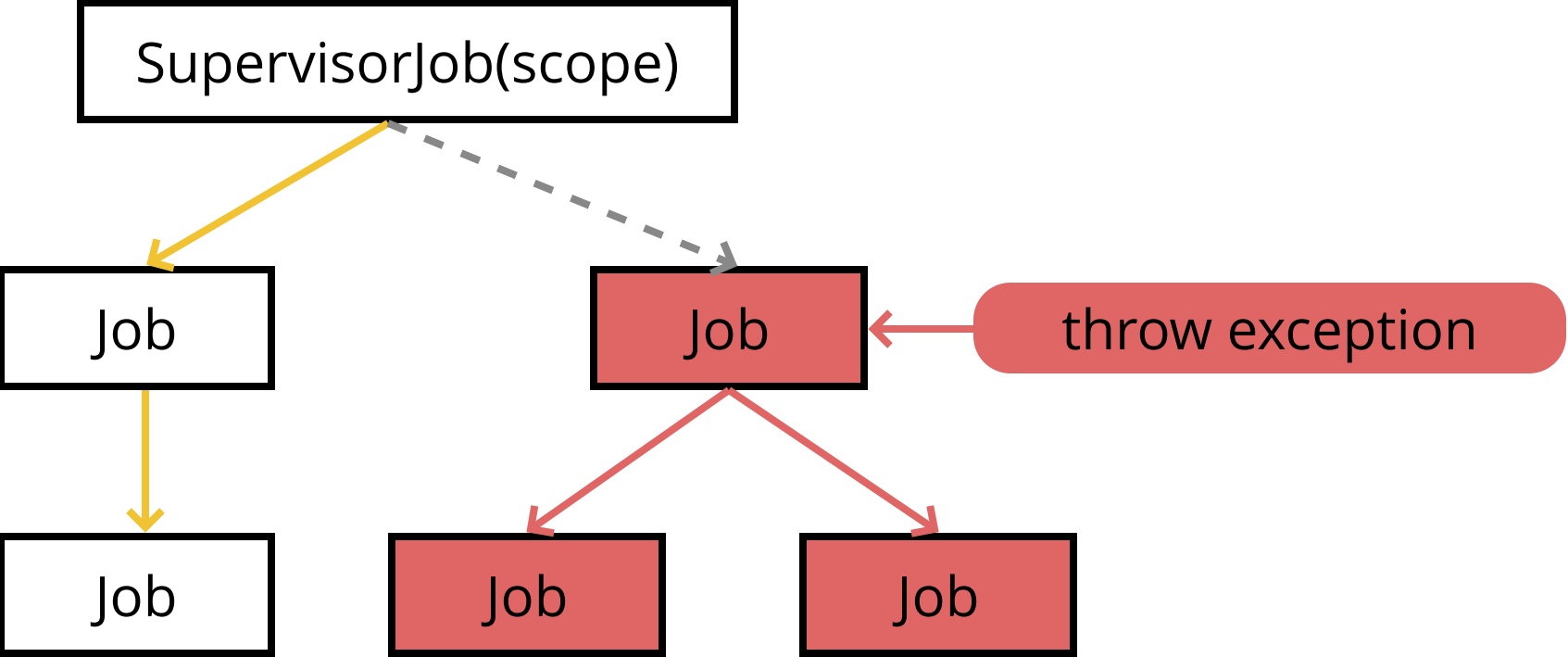
Error propagating: SupervisorJob



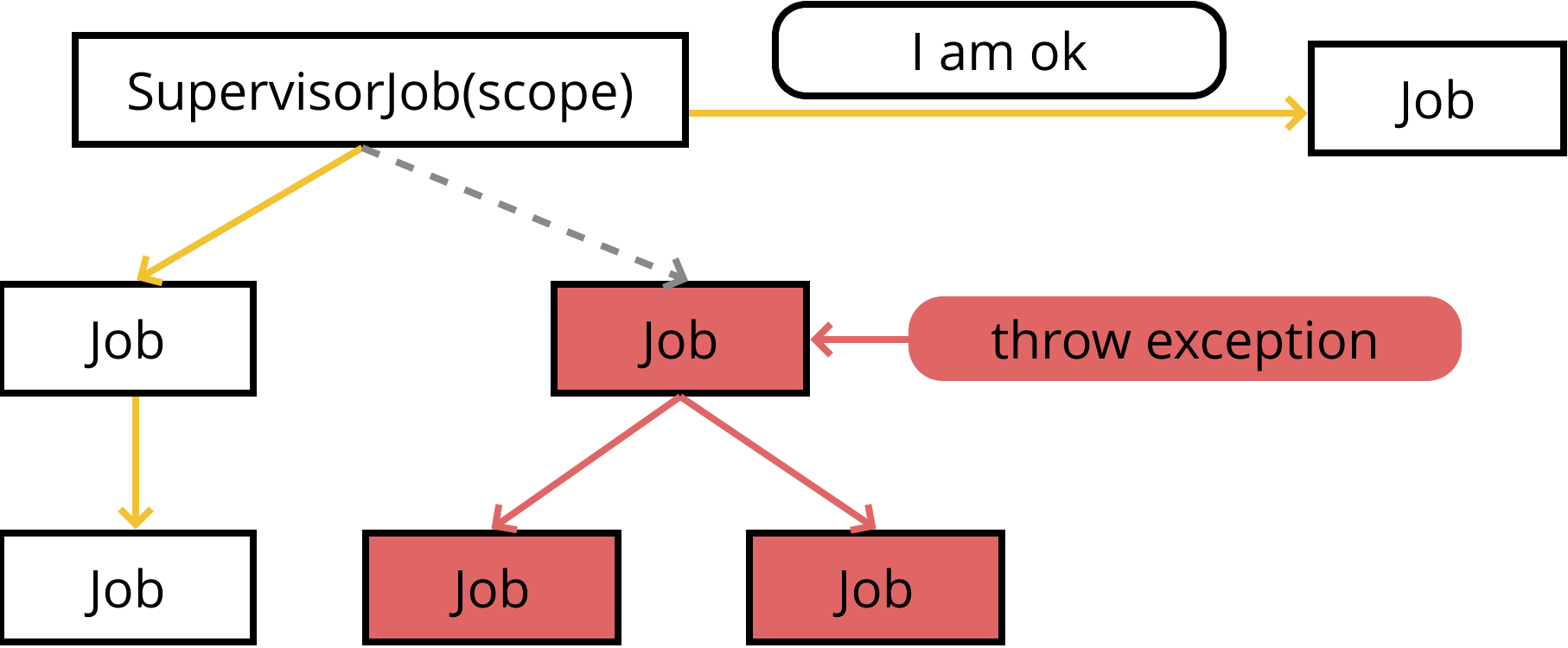
Error propagating: SupervisorJob



Error propagating: SupervisorJob



Error propagating: SupervisorJob



SupervisorJob: пример

```
fun onStart() { // with SupervisorJob
    launch(handler) {
        println("Start C1")
        repeat(5) {
            delay(1000)
            println("tick $it")
        }
    }
    launch(handler) {
        println("Start C2")
        delay(3000)
        println("C2: throw exception")
        errorCall()
    }
}

val handler = CoroutineExceptionHandler { _, throwable ->
    println("handle error (${throwable.message})")
}
```

SupervisorJob: пример

```
fun onStart() { // with SupervisorJob
    launch(handler) {
        println("Start C1")
        repeat(5) {
            delay(1000)
            println("tick $it")
        }
    }
    launch(handler) {
        println("Start C2")
        delay(3000)
        println("C2: throw exception")
        errorCall()
    }
}
```

```
val handler = CoroutineExceptionHandler { _, throwable ->
    println("handle error (${throwable.message})")
}
```

```
18:38:51.012 I/System.out: Start C1
18:38:51.015 I/System.out: Start C2
18:38:52.015 I/System.out: tick 0
18:38:53.017 I/System.out: tick 1
18:38:54.017 I/System.out: C2: throw exception
18:38:54.018 I/System.out: handle error (some error happened)
18:38:54.019 I/System.out: tick 2
18:38:55.022 I/System.out: tick 3
18:38:56.025 I/System.out: tick 4
```

try-catch: будьте внимательны

```
fun onStart() {
    launch {
        try {
            launch (Dispatchers.IO){
                delay(3000)
                println("some work is done")
            }
            errorCall() // throwing exception
        } catch (e: Exception) {
            e.printStackTrace()
            println("catch error")
        }
    }
}
```

try-catch: будьте внимательны

```
fun onStart() {
    launch {
        try {
            launch (Dispatchers.IO){
                delay(3000)
                println("some work is done")
            }
            errorCall() // throwing exception
        } catch (e: Exception) {
            e.printStackTrace()
            println("catch error")
        }
    }
}
```

```
12:55:44.444 W/System.err: java.lang.RuntimeException: some error happened
12:55:44.444 W/System.err:     at ...
12:55:44.445 I/System.out: catch error
...
12:55:47.455 I/System.out: some work is done
```


try-catch: для suspending вызовов

```
{
  launch {
    try {
      val user = getUserSuspended()
      showUser(user)
    } catch (e: Exception) {
      e.printStackTrace()
      println("catch error")
    }
  }
}
```


try-catch: для suspending ВЫЗОВОВ

```
{
  launch {
    try {
      val user = getUserSuspended()
      showUser(user)
    } catch (e: Exception) {
      e.printStackTrace()
      println("catch error")
    }
  }
}

suspend fun getUserSuspended(): User =
  withContext(Dispatchers.IO) {
    repo.getUser()
  }
```

A diagram consisting of a solid black line that starts from the right side of the call to `getUserSuspended()` in the `try` block of the `launch` function. The line extends horizontally to the right, then turns 90 degrees upwards, and then 90 degrees to the right again, ending with a small black square at the start of the `suspend fun getUserSuspended()` definition.

try-catch: для suspending ВЫЗОВОВ

```
{
  launch {
    try {
      val user = getUserSuspended()
      showUser(user)
    } catch (e: Exception) {
      e.printStackTrace()
      println("catch error")
    }
  }
}
```

```
suspend fun getUserSuspended(): User =
    withContext(Dispatchers.IO) {
        repo.getUser()
    }

suspend fun getUserSuspended(): User =
    coroutineScope {
        async(Dispatchers.IO) {
            repo.getUser()
        }.await()
    }
```

try-catch: для suspending ВЫЗОВОВ

```
{
  launch {
    try {
      val user = getUserSuspended()
      showUser(user)
    } catch (e: Exception) {
      e.printStackTrace()
      println("catch error")
    }
  }
}
```

```
suspend fun getUserSuspended(): User =
  withContext(Dispatchers.IO) {
    repo.getUser()
  }

suspend fun getUserSuspended(): User =
  coroutineScope {
    async(Dispatchers.IO) {
      repo.getUser()
    }.await()
  }
```

```
W/System.err: java.lang.RuntimeException: error while user getting
W/System.err:     at ...
I/System.out: catch error
```

Обработка ошибок: *summary*

Совместное использование CoroutineExceptionHandler и try-catch:

Обработка ошибок: `summary`

Совместное использование `CoroutineExceptionHandler` и `try-catch`:

- `CoroutineExceptionHandler` - можно использовать как глобальный обработчик ошибок в корутине

Обработка ошибок: summary

Совместное использование CoroutineExceptionHandler и try-catch:

- CoroutineExceptionHandler - можно использовать как глобальный обработчик ошибок в корутине
- try-catch как обработку для suspending вызовов

```
{  
    val handler = CoroutineExceptionHandler { context, throwable ->  
        // handle global errors  
    }  
    launch(handler) {  
        val user = try {  
            getUserSuspended()  
        } catch (e: Exception) {  
            getDefaultUser()  
        }  
        showUser(user)  
    }  
}
```

Заключение

Coroutines vs Rx

Coroutines vs Rx

- Ни чего не имею против Rx

Coroutines vs Rx

- Ни чего не имею против Rx
- Но для простых случаев он избыточен
 - Походы в сеть, БД, и т.п.
 - Логика, где нет ни каких потоков данных

Coroutines vs Rx

- Ни чего не имею против Rx
- Но для простых случаев он избыточен
 - Походы в сеть, БД, и т.п.
 - Логика, где нет ни каких потоков данных
- Переезд на корутины - шаг в верном направлении
 - Фича языка
 - Прямое назначение

Coroutines

Coroutines

- Внутриязыковой инструмент
 - Стандартная библиотека содержит базовый API
 - Билдеры, диспетчеры, и практически все остальное находится в `kotlinx-coroutines-*` библиотеках

Coroutines

- Внутриязыковой инструмент
 - Стандартная библиотека содержит базовый api
 - Билдеры, диспетчеры, и практически все остальное находится в `kotlinx-coroutines-*` библиотеках
- Лаконичность (как минимум не многословнее чем Rx)

Coroutines

- Внутриязыковой инструмент
 - Стандартная библиотека содержит базовый api
 - Билдеры, диспетчеры, и практически все остальное находится в `kotlinx-coroutines-*` библиотеках
- Лаконичность (как минимум не многословнее чем Rx)
- Простое переключение между потоками

Coroutines

- Внутриязыковой инструмент
 - Стандартная библиотека содержит базовый api
 - Билдеры, диспетчеры, и практически все остальное находится в `kotlinx-coroutines-*` библиотеках
- Лаконичность (как минимум не многословнее чем Rx)
- Простое переключение между потоками
- Можно воспринимать как "легковесные потоки" (схожесть с Rx)

Coroutines

- Внутриязыковой инструмент
 - Стандартная библиотека содержит базовый api
 - Билдеры, диспетчеры, и практически все остальное находится в `kotlinx-coroutines-*` библиотеках
- Лаконичность (как минимум не многословнее чем Rx)
- Простое переключение между потоками
- Можно воспринимать как "легковесные потоки" (схожесть с Rx)
- Код в последовательном стиле (ни каких `zip`, `flat-map`, `map` операторов и коллбеков)



There is no going back now

Coroutines: трезвый взгляд

Coroutines: трезвый взгляд

- Порог вхождения

Coroutines: трезвый взгляд

- Порог вхождения
- Есть не очевидные моменты
 - Структура Job
 - Обработка ошибок
 - и т.п.

Coroutines: трезвый взгляд

- Порог вхождения
- Есть не очевидные моменты
 - Структура Job
 - Обработка ошибок
 - и т.п.
- Тренируйтесь "на кошках"

Coroutines: трезвый взгляд

- Порог вхождения
- Есть не очевидные моменты
 - Структура Job
 - Обработка ошибок
 - и т.п.
- Тренируйтесь "на кошках"
- Ни кто не говорил, что будет легко

Всем спасибо!
Вопросы?