OTUS

Android Developer

Kotlin Flow

otus.ru

Проверить, идет ли запись

```
if (видно && слышно) {
   chat.print(“+”)
}
```

Ставим “+”, если все хорошо
“-”, если есть проблемы

Тема вебинара

Kotlin Flow

Николай Кочетков

Руководитель Андроид разработки FlyXO

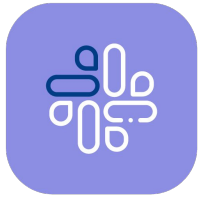
Об опыте (например):

24 года в IT, 8 лет - играющий тренер Андроид

Телефон / эл. почта / соц. сети:
LinkedIn: https://www.linkedin.com/in/motorro/
GitHub: https://github.com/motorro/
Medium: https://medium.com/@motorro

Правила вебинара

Активно
участвуем



Off-topic обсуждаем
в Slack #**канал группы**
или #general

Задаем вопрос
в чат или голосом

Вопросы вижу в чате,
могу ответить не сразу

Условные обозначения

Индивидуально

Время, необходимое
на активность

Пишем в чат

Говорим голосом

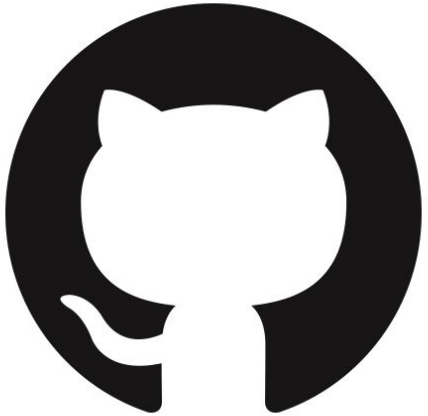
Документ

Ответьте себе или
задайте вопрос

Маршрут вебинара

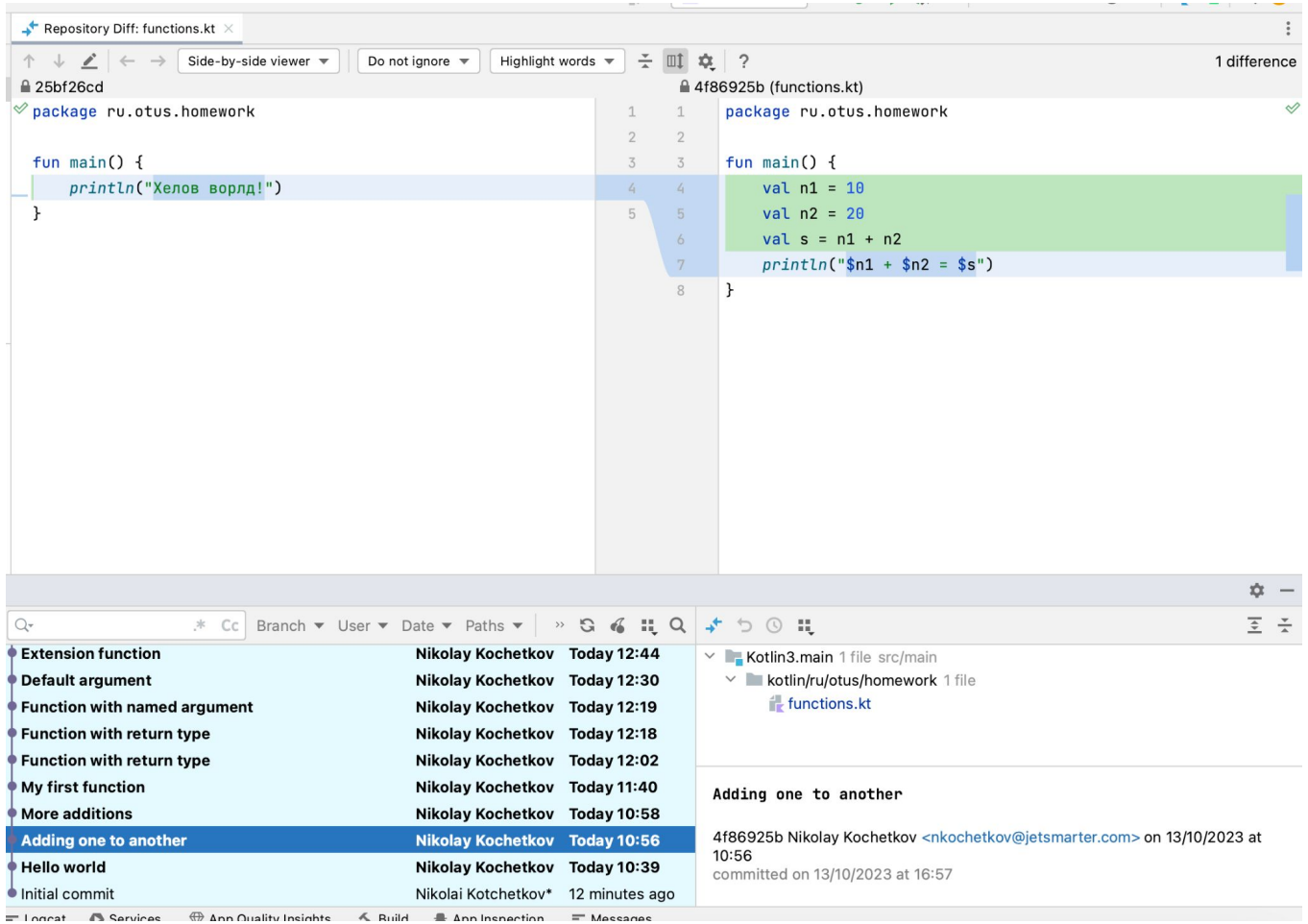
- Реактивное программирование и Flow
- Получение данных и контекст
- Преобразования и объединения
- SharedFlow и StateFlow
- Обработка ошибок

Репозиторий к занятию

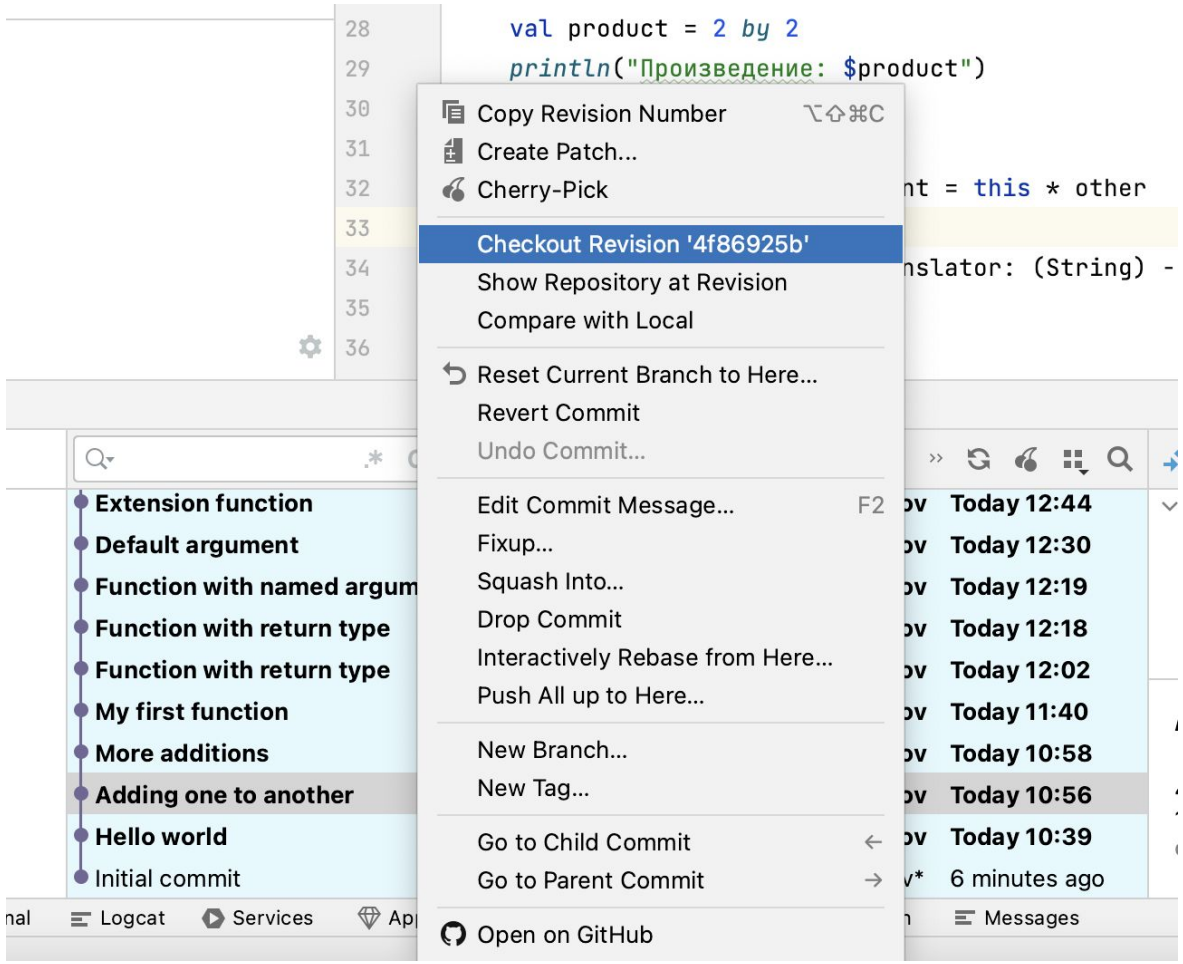


https://github.com/Android-Developer-Basic/FLow

Репозиторий к занятию - по шагам

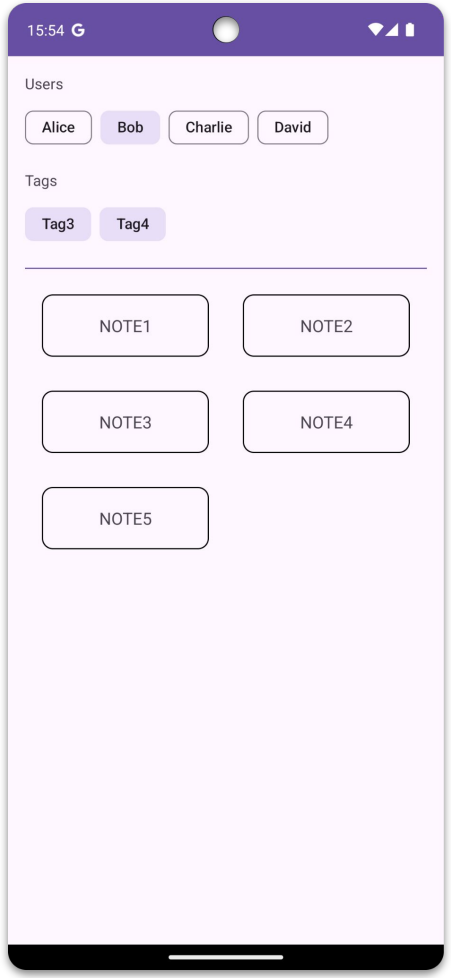


Репозиторий к занятию - по шагам



Программа показа заметок

Программа показа заметок



Получение пользователей

```kotlin
/**
 * Returns user list
 */
suspend fun getUsers(): Result<List<User>> {
    delay(NETWORK_DELAY)
    return Result.success(users)
}
```

data/users.kt

Отображение пользователей

```
private fun loadUsers() {
    Log.i(TAG, msg: "Loading users")
    lifecycleScope.launch { this: CoroutineScope
        val result = getUsers()
        if (result.isSuccess) {
            populateUsers(result.getOrThrow())
        } else {
            Log.e(TAG, msg: "Failed to load users", result.exceptionOrNull())
        }
    }
}
```

MainActivity.kt

Обработчик выбора пользователя

```
private fun selectUser(userId: Int?) {
    Log.i(TAG, msg: "Selected user: $userId")
}
```

MainActivity.kt

Вопросы?

Получение тэгов

```kotlin
/**
 * Returns tags for user
 */
suspend fun getTagsForUser(userId: Int): Result<List<Tag>> {
    delay(NETWORK_DELAY)
    return Result.success( value: tags[userId] ?: emptyList())
}
```

data/tags.kt

Отображение тэгов

```kotlin
private fun loadTags(userId: Int?) {
    Log.i(TAG, msg: "Loading tags for user: $userId")
    hideTags()
    if (null == userId) {
        populateTags(emptyList())
        return
    }

    lifecycleScope.launch { this: CoroutineScope
        val result = getTagsForUser(userId)
        if (result.isSuccess) {
            populateTags(result.getOrThrow())
            showTags()
        } else {
            Log.e(TAG, msg: "Failed to load tags", result.exceptionOrNull())
        }
    }
}
```

MainActivity.kt

Отображение тэгов

Прячем до загрузки

```
private fun loadTags(userId: Int?) {
    Log.i(TAG, msg: "Loading tags for user: $userId")
    hideTags()
    if (null == userId) {
        populateTags(emptyList())
        return
    }

    lifecycleScope.launch { this: CoroutineScope
        val result = getTagsForUser(userId)
        if (result.isSuccess) {
            populateTags(result.getOrThrow())
            showTags()
        } else {
            Log.e(TAG, msg: "Failed to load tags", result.exceptionOrNull())
        }
    }
}
```

MainActivity.kt

Отображение тэгов

Очищаем в отсутствие пользователя

```
private fun loadTags(userId: Int?) {
    Log.i(TAG, msg: "Loading tags for user: $userId")
    hideTags()
    if (null == userId) {
        populateTags(emptyList())
        return
    }

    lifecycleScope.launch { this: CoroutineScope
        val result = getTagsForUser(userId)
        if (result.isSuccess) {
            populateTags(result.getOrThrow())
            showTags()
        } else {
            Log.e(TAG, msg: "Failed to load tags", result.exceptionOrNull())
        }
    }
}
```

MainActivity.kt

Отображение тэгов

Показываем данные

```
private fun loadTags(userId: Int?) {
    Log.i(TAG, msg: "Loading tags for user: $userId")
    hideTags()
    if (null == userId) {
        populateTags(emptyList())
        return
    }

    lifecycleScope.launch { this: CoroutineScope
        val result = getTagsForUser(userId)
        if (result.isSuccess) {
            populateTags(result.getOrThrow())
            showTags()
        } else {
            Log.e(TAG, msg: "Failed to load tags", result.exceptionOrNull())
        }
    }
}
```

MainActivity.kt

Загрузка тэгов по смене пользователя

```
private fun selectUser(userId: Int?) {
    Log.i(TAG, msg: "Selected user: $userId")
    loadTags(userId)
}
```

MainActivity.kt

Выбор тэгов

```
private fun selectTags(tags: Set<Int>) {
    Log.i(TAG, msg: "Selected tags: $tags")
}
```

MainActivity.kt

Вопросы?

Получение заметок

```kotlin
/**
 * Returns notes for user
 */
suspend fun getNotesForUser(userId: Int, tags: Set<Int>): Result<List<Note>> {
    delay(NETWORK_DELAY)
    return Result.success( value: notes[userId]?.filter { notes -> notes.tags.any { tags.contains(it) } } ?: emptyList())
}
```

data/notes.kt

Управление экраном заметок

Очистка заметок

Загружаем заметки

```
private fun loadTags() {
    val userId = getSelectedUser()
    Log.i(TAG, msg: "Loading tags for user: $userId")
    hideTags()
    populateNotes(emptyList())
    if (null == userId) {
        populateTags(emptyList())
        return
    }

    lifecycleScope.launch { this: CoroutineScope
        val result = getTagsForUser(userId)
        if (result.isSuccess) {
            populateTags(result.getOrThrow())
            showTags()
            loadNotes()
        } else {
            Log.e(TAG, msg: "Failed to load tags", result.exceptionOrNull())
        }
    }
}
```

MainActivity.kt

Управление экраном заметок

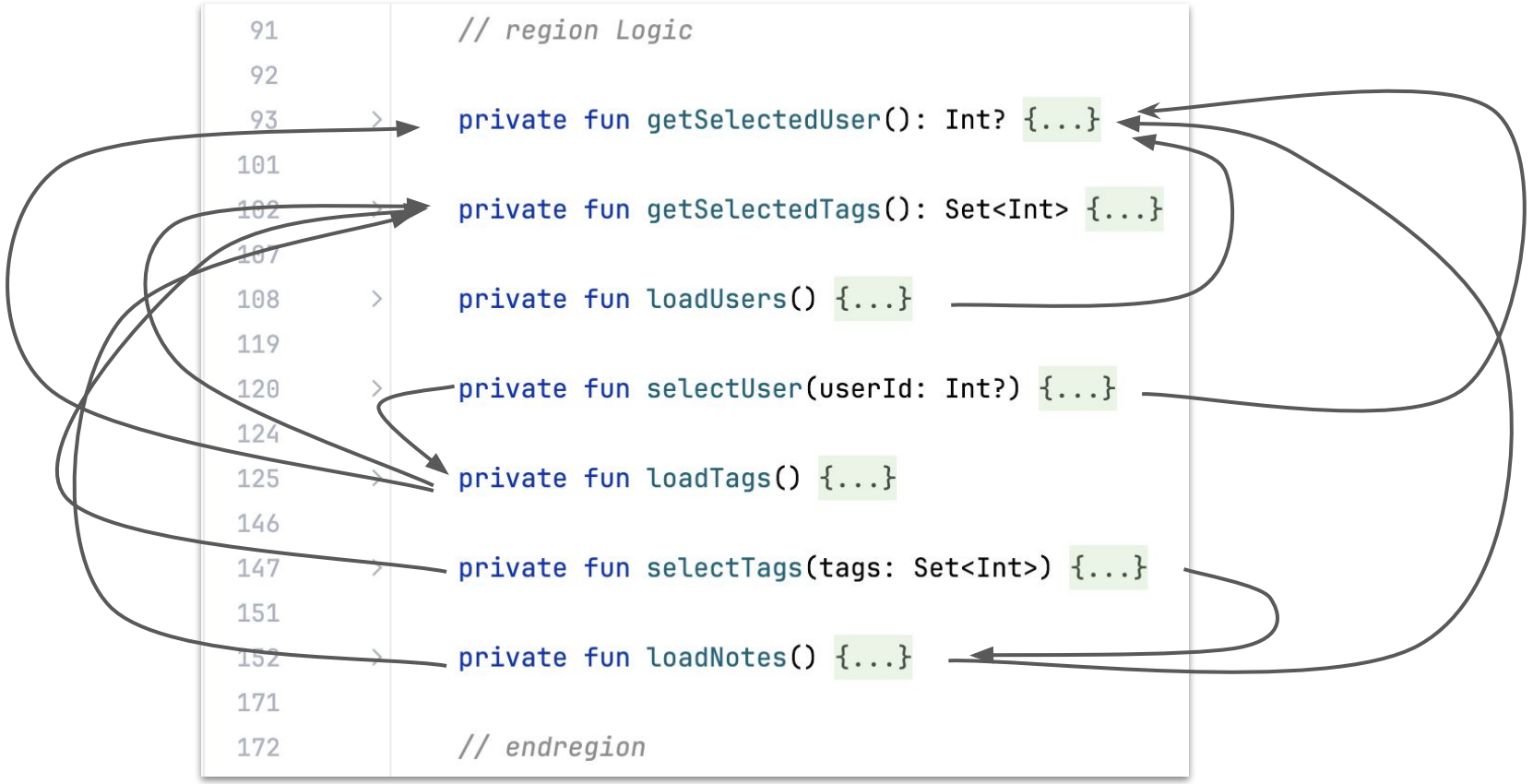
80 строк кода

```
91      // region Logic
92
93      private fun getSelectedUser(): Int? {...}
101
102     private fun getSelectedTags(): Set<Int> {...}
107
108     private fun loadUsers() {...}
119
120     private fun selectUser(userId: Int?) {...}
124
125     private fun loadTags() {...}
146
147     private fun selectTags(tags: Set<Int>) {...}
151
152     private fun loadNotes() {...}
171
172     // endregion
```

7 функций

MainActivity.kt

Управление экраном заметок



MainActivity.kt

Микроменеджмент (императивный подход)

1. Спрячь пользователей
2. Спрячь тэги
3. Загрузи пользователей
4. Загрузи тэги
5. …
6. Покажи записки

Вопросы?

Ограничения императивного подхода

Функции возвращают одно значение

На каждый вызов

```kotlin
/**
 * Returns tags for user
 */
suspend fun getTagsForUser(userId: Int): Result<List<Tag>> {
    delay(NETWORK_DELAY)
    return Result.success( value: tags[userId] ?: emptyList())
}
```

data/tags.kt

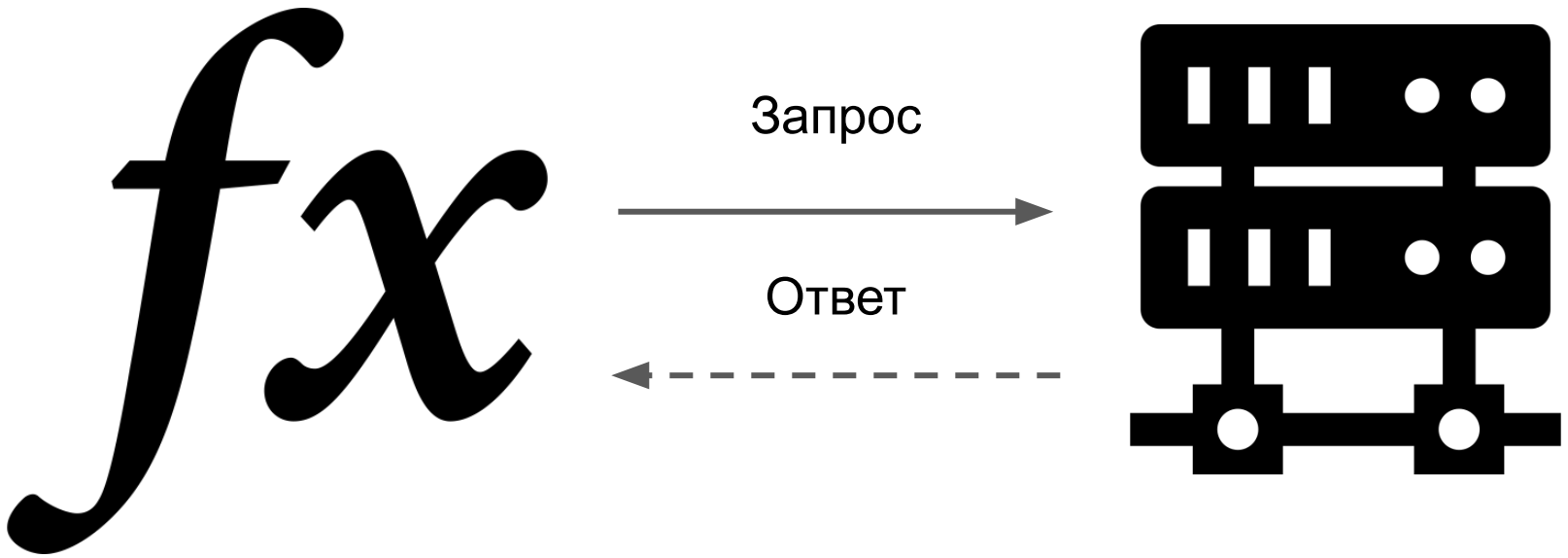
Функции возвращают одно значение

Один результат

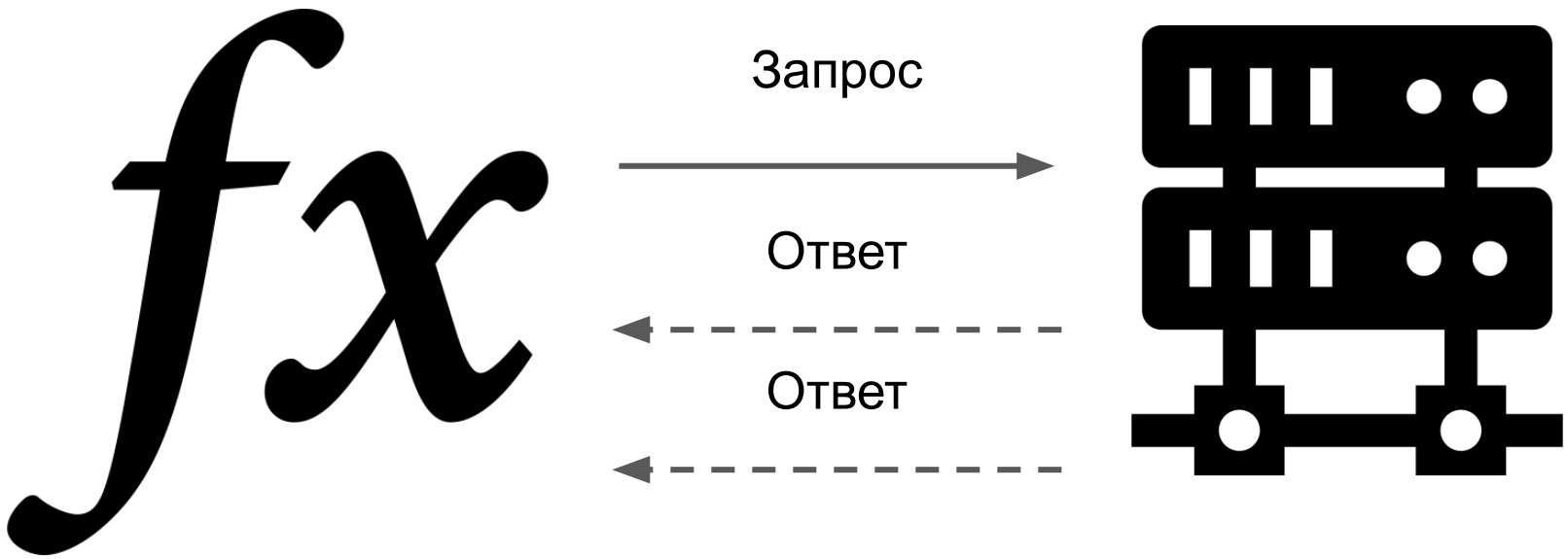
```kotlin
/**
 * Returns tags for user
 */
suspend fun getTagsForUser(userId: Int): Result<List<Tag>> {
    delay(NETWORK_DELAY)
    return Result.success( value: tags[userId] ?: emptyList())
}
```

data/tags.kt

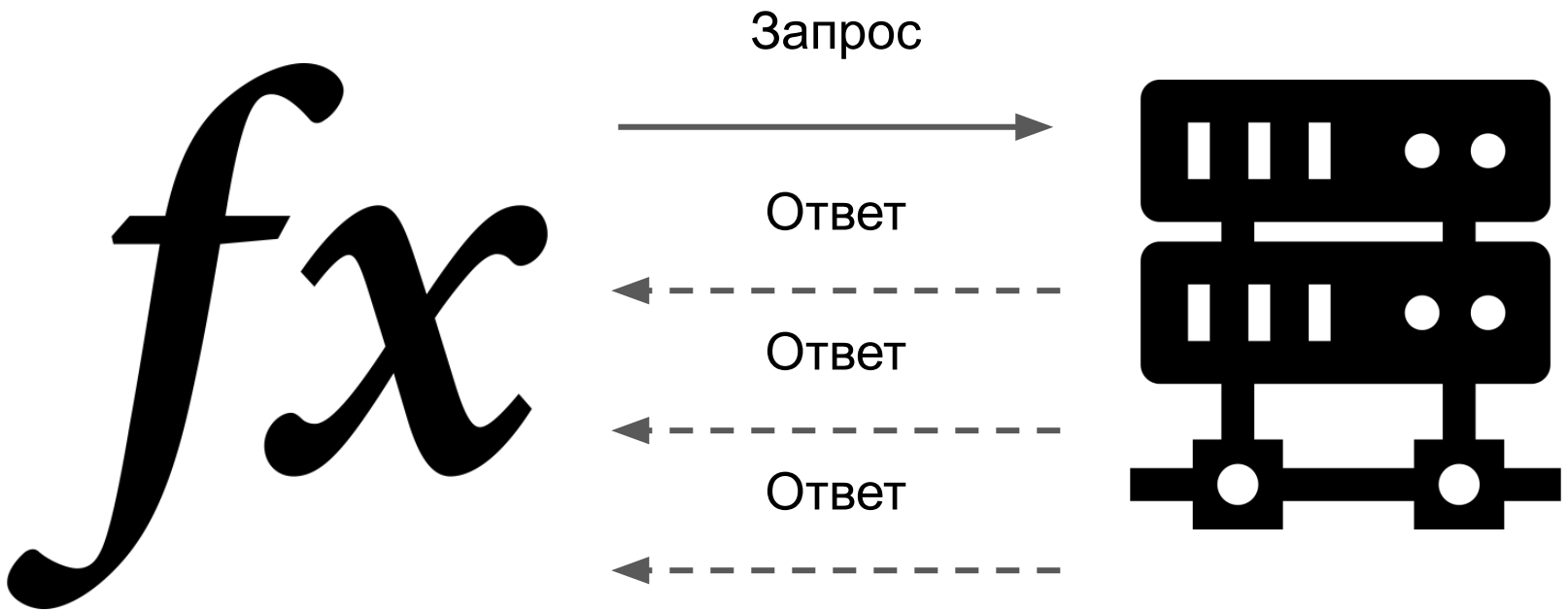
Функции возвращают одно значение



Push-обновления



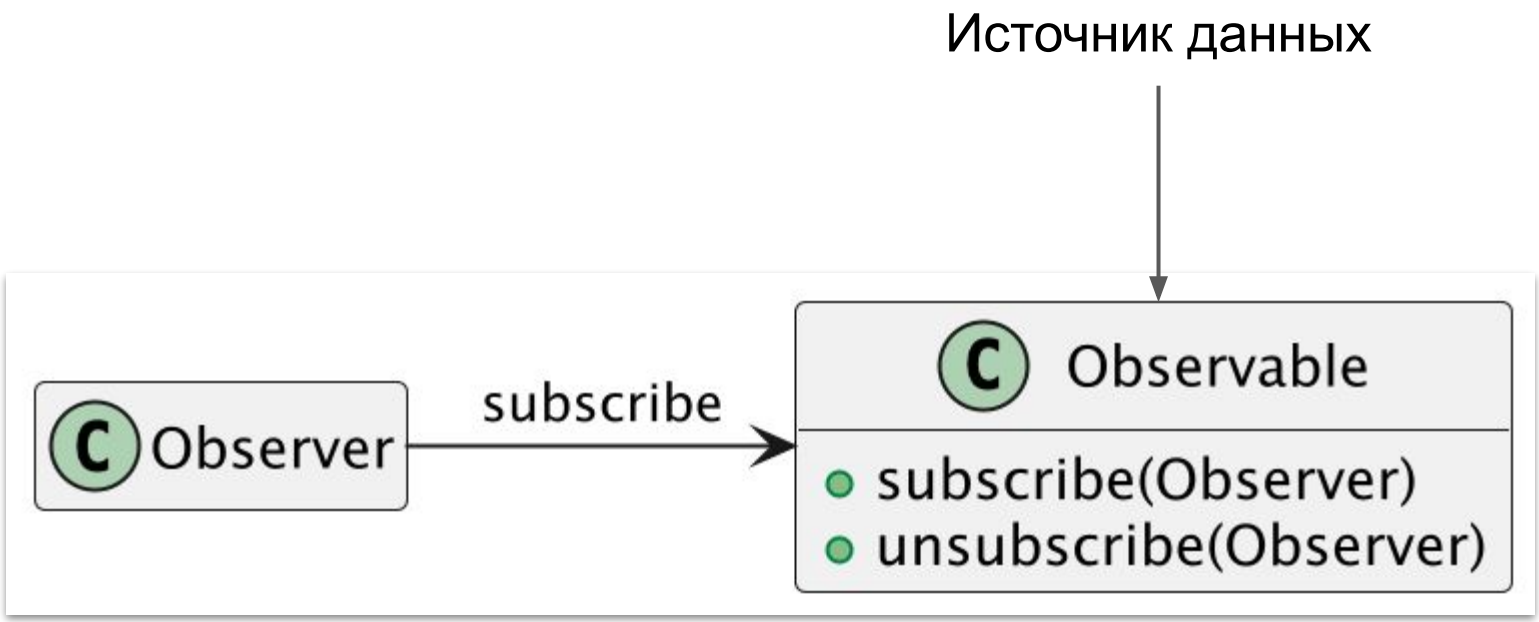
Push-обновления



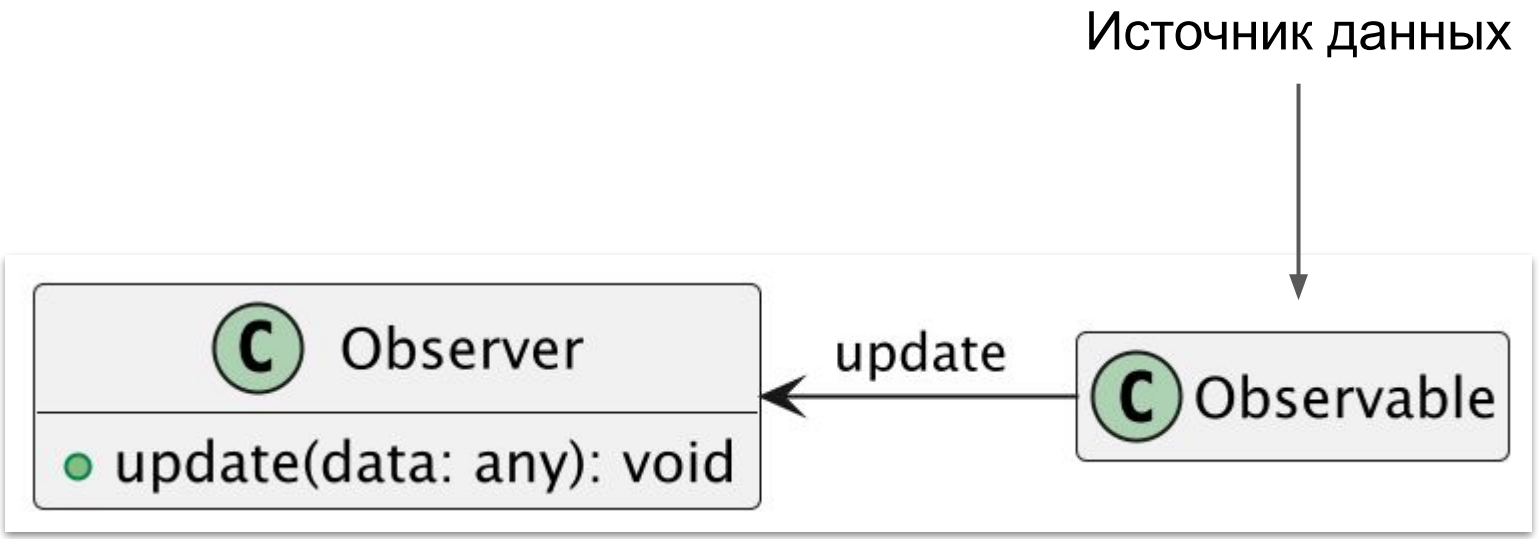
Вопросы?

Observer
Потоки данных

Observer

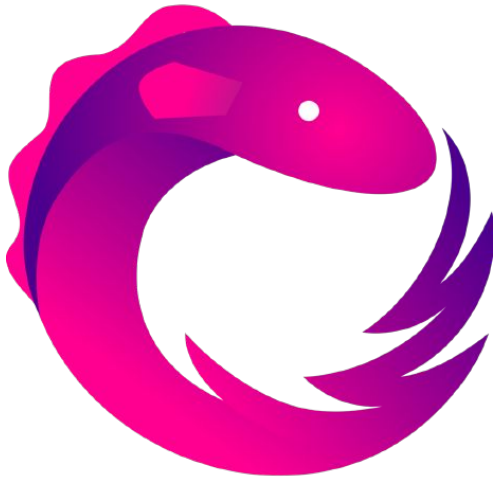

Источник данных
Observer
subscribe
Observable
subscribe(Observer)
unsubscribe(Observer)

Observer


Источник данных
Observer
update(data: any): void
update
Observable

Observer

LiveData



RxJava

Flow

Коллекция

```
fun main() {
    collection().forEach { value -> println(value) }
}

fun collection(): List<Int> = listOf(1, 2, 3)
```

collection.kt

Sequence - блокирует поток

```kotlin
fun main() {
    sequence().forEach { value -> println(value) }
}

fun sequence(): Sequence<Int> = sequence { this: SequenceScope<Int>
    for (i in 1 ≤ .. ≤ 3) {
        Thread.sleep(100) // Long operation, BLOCKS the thread!
        yield(i) // next value
    }
}
```

sequence.kt

Coroutine - один результат

```
fun main() = runBlocking { this: CoroutineScope
    suspending().forEach { value -> println(value) }
}

suspend fun suspending(): List<Int> {
    delay( timeMillis: 1000) // All values at once!
    return listOf(1, 2, 3)
}
```

Один результат

coroutine.kt

Kotlin Flow

Flow

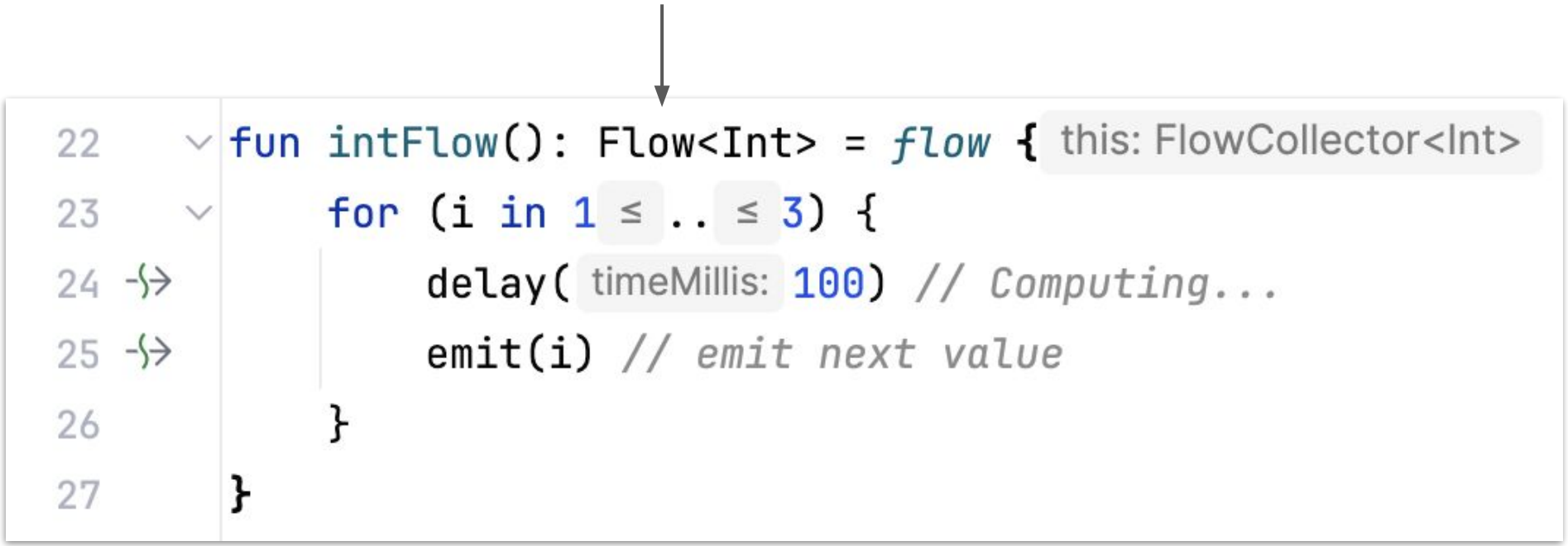
Flow

```
22  fun intFlow(): Flow<Int> = flow { this: FlowCollector<Int>
23      for (i in 1 ≤ .. ≤ 3) {
24          delay( timeMillis: 100) // Computing...
25          emit(i) // emit next value
26      }
27  }
```

flow.kt

Flow

Поток данных



flow.kt

Flow

```
22  fun intFlow(): Flow<Int> = flow { this: FlowCollector<Int>
23      for (i in 1 ≤ .. ≤ 3) {
24          delay( timeMillis: 100) // Computing...
25          emit(i) // emit next value
26      }
27  }
```

builder

flow.kt

Flow

Suspend-функция

```
22  fun intFlow(): Flow<Int> = flow { this: FlowCollector<Int>
23      for (i in 1 ≤ .. ≤ 3) {
24          delay( timeMillis: 100) // Computing...
25          emit(i) // emit next value
26      }
27  }
```

flow.kt

Flow

Следующее значение

```
22  fun intFlow(): Flow<Int> = flow { this: FlowCollector<Int>
23      for (i in 1 ≤ .. ≤ 3) {
24          delay( timeMillis: 100) // Computing...
25          emit(i) // emit next value
26      }
27  }
```

flow.kt

Создание Flow

```kotlin
// From iterable
val fromIterable: Flow<Int> = listOf(1, 2, 3).asFlow()
// From sequence
val fromSequence: Flow<Int> = sequence { for (i in 1 ≤ .. ≤ 3) yield(i) }.asFlow()
// From range
val fromRange: Flow<Int> = (1 ≤ .. ≤ 3).asFlow()
// From values
val fromValues: Flow<Int> = flowOf( ...elements: 1, 2, 3)
// Empty flow
val emptyFlow: Flow<Int> = emptyFlow()
```

flowStarters.kt

Получение Flow

```
fun main() = runBlocking { this: CoroutineScope
    // Collect the flow
    intFlow().collect { value -> println(value) }
}

fun intFlow(): Flow<Int> = flow { this: FlowCollector<Int>
    for (i in 1 ≤ .. ≤ 3) {
        delay( timeMillis: 100) // Computing...
        emit(i) // emit next value
    }
}
```

Observer

flow.kt

Получение Flow

Suspend функция

```
fun main() = runBlocking { this: CoroutineScope
    // Collect the flow
    intFlow().collect { value -> println(value) }
}

fun intFlow(): Flow<Int> = flow { this: FlowCollector<Int>
    for (i in 1 ≤ .. ≤ 3) {
        delay( timeMillis: 100) // Computing...
        emit(i) // emit next value
    }
}
```

flow.kt

Вопросы?

Cold Flow

```
fun randomFlow(): Flow<Int> = flow { this: FlowCollector<Int>
    for (i in 1 ≤ .. ≤ 3) {
        delay( timeMillis: 100)
        emit(Random.nextInt( from: 1,  until: 10)) // Random value
    }
}
```

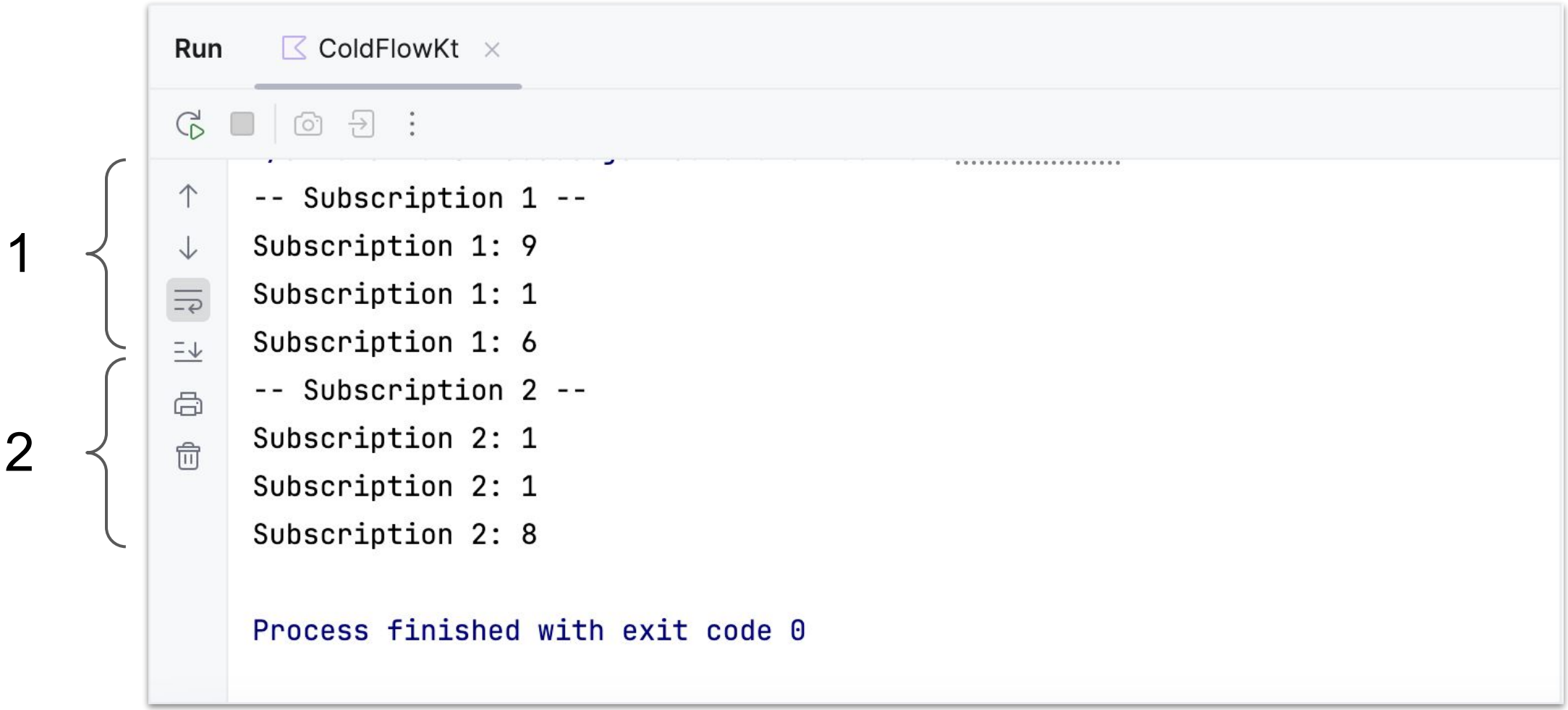
coldFlow.kt

Cold Flow

```
fun main() = runBlocking { this: CoroutineScope
    println("-- Subscription 1 --")
    randomFlow().collect { value -> println("Subscription 1: $value") }
    println("-- Subscription 2 --")
    randomFlow().collect { value -> println("Subscription 2: $value") }
}
```

coldFlow.kt

Cold Flow


Run
ColdFlowKt
-- Subscription 1 --
Subscription 1: 9
Subscription 1: 1
Subscription 1: 6
-- Subscription 2 --
Subscription 2: 1
Subscription 2: 1
Subscription 2: 8
Process finished with exit code 0
1
2

Cold Flow

```
fun main() = runBlocking { this: CoroutineScope
    println("-- Subscription 1 --")
    randomFlow().collect { value -> println("Subscription 1: $value") }
    println("-- Subscription 2 --")
    randomFlow().
}
```

```
fun randomFlow(): Flow<Int> = flow { this: FlowCollector<Int>
    for (i in 1 ≤ .. ≤ 3) {
        delay( timeMillis: 100)
        emit(Random.nextInt( from: 1,  until: 10)) // Random value
    }
}
```

В момент Collect

Cold Flow

Холодный Observable - это источник потоковых данных, который “излучает” данные только в момент подписки, а объект-producer, будет своим для каждого подписчика...

Вопросы?

Flow Context

```
fun namedFlow(): Flow<String> = flow { this: FlowCollector<String>
    for (i in 1 ≤ .. ≤ 3) {
        // Emit coroutine name
        emit( value: "At ${coroutineContext[CoroutineName]}: $i")
    }
}
```

Имя корутины

coldFlow.kt

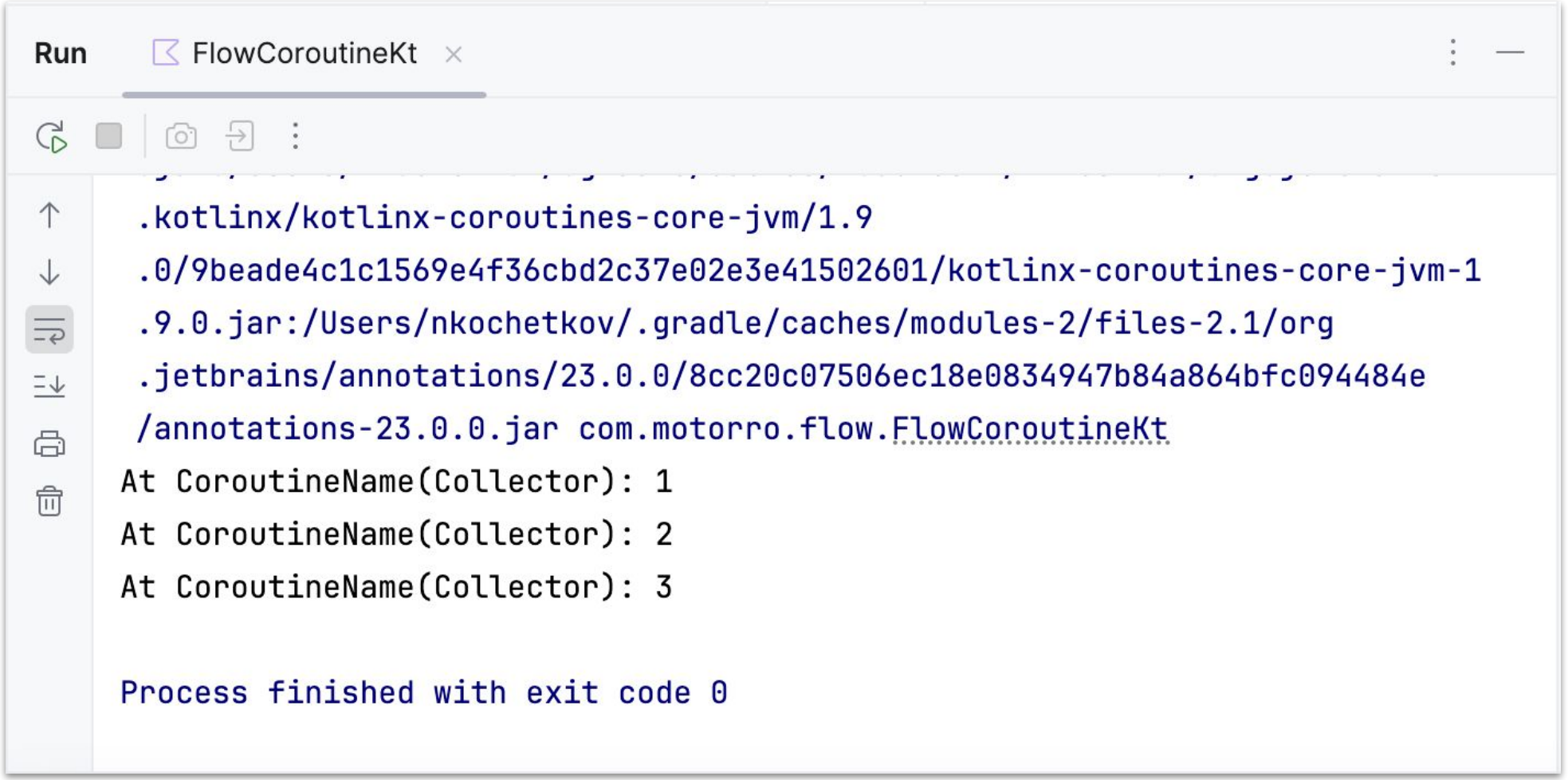
Flow Context

```
fun main(): Unit = runBlocking { this: CoroutineScope
    launch(CoroutineName( name: "Collector")) { this: CoroutineScope
        namedFlow().collect { value -> println(value) }
    }
}
```

coldFlow.kt

Flow Context

Коллектор!



Flow Context

Тело билдера Flow (по умолчанию) выполняется в контексте сборщика. Это называется **Context Preservation**...

Вопросы?

Нарушение Context Preservation

Emit из другого контекста

```
fun violatingContext(): Flow<String> = flow { this: FlowCollector<String>
    withContext(Dispatchers.IO) { this: CoroutineScope
        delay( timeMillis: 1000)
        emit( value: "Value") // Emit value from another context
    }
}
```

violatingContext.kt

Нарушение Context Preservation

Context Preservation

Получение в другом контексте

Emit из основного контекста

```
fun preservingContext(): Flow<String> = flow { this: FlowCollector<String>
    val value = withContext(Dispatchers.IO) { this: CoroutineScope
        delay( timeMillis: 1000)
        "Value" ^withContext // Return some value
    }
    emit(value) // Emit from original context
}
```

preservingContext.kt

flowOn

```
fun flowingOnAnotherContext(): Flow<String> = flow { this: FlowCollector<String>
    delay( timeMillis: 1000)
    emit( value: "Value from: ${coroutineContext[CoroutineName]}")
}.flowOn( context: Dispatchers.IO + CoroutineName( name: "Flow"))
```

Смена
контекста

flowOn

Получение в одной корутине

Отправка из другой корутины

```
fun main(): Unit = runBlocking(CoroutineName( name: "Collect")) { this: CoroutineScope
    flowingOnAnotherContext().collect { value ->
        println("Print on ${coroutineContext[CoroutineName]}: $value")
    }
}

fun flowingOnAnotherContext(): Flow<String> = flow { this: FlowCollector<String>
    delay( timeMillis: 1000)
    emit( value: "Value from: ${coroutineContext[CoroutineName]}")
}.flowOn( context: Dispatchers.IO + CoroutineName( name: "Flow"))
```

flowOn.kt

flowOn

```
Print on CoroutineName(Collect): Value from: CoroutineName(Flow)
```

Коллектор

Генератор

Автообновление заметок

Обновление заметок

```kotlin
fun getNotesFlow(userId: Int?, tags: Set<Int>): Flow<List<Note>> = flow { this: FlowCollector<List<Note>>
    val notes = getNotesForUser(userId, tags).getOrThrow()
    if (notes.isEmpty()) {
        emit(emptyList())
        return@flow
    }

    while(currentCoroutineContext().isActive) {
        val notesFeed = mutableListOf<Note>()
        for (note in notes) {
            notesFeed.add(note)
            emit(notesFeed.toList())
            delay(FEED_DELAY)
        }
        delay(REFRESH_DELAY)
    }
}
```

notes.kt

Обновление заметок

Получаем массив заметок

```
28  fun getNotesFlow(userId: Int?, tags: Set<Int>): Flow<List<Note>> = flow { this: FlowCollector<List<Note>>
29      val notes = getNotesForUser(userId, tags).getOrThrow()
30      if (notes.isEmpty()) {
31          emit(emptyList())
32          return@flow
33      }
34
35      while(currentCoroutineContext().isActive) {
36          val notesFeed = mutableListOf<Note>()
37          for (note in notes) {
38              notesFeed.add(note)
39              emit(notesFeed.toList())
40              delay(FEED_DELAY)
41          }
42          delay(REFRESH_DELAY)
43      }
44  }
```

notes.kt

Обновление заметок

Пока корутина активна

```kotlin
fun getNotesFlow(userId: Int?, tags: Set<Int>): Flow<List<Note>> = flow { this: FlowCollector<List<Note>>
    val notes = getNotesForUser(userId, tags).getOrThrow()
    if (notes.isEmpty()) {
        emit(emptyList())
        return@flow
    }

    while(currentCoroutineContext().isActive) {
        val notesFeed = mutableListOf<Note>()
        for (note in notes) {
            notesFeed.add(note)
            emit(notesFeed.toList())
            delay(FEED_DELAY)
        }
        delay(REFRESH_DELAY)
    }
}
```

notes.kt

Обновление заметок

Добавляем по одному

```kotlin
fun getNotesFlow(userId: Int?, tags: Set<Int>): Flow<List<Note>> = flow { this: FlowCollector<List<Note>>
    val notes = getNotesForUser(userId, tags).getOrThrow()
    if (notes.isEmpty()) {
        emit(emptyList())
        return@flow
    }

    while(currentCoroutineContext().isActive) {
        val notesFeed = mutableListOf<Note>()
        for (note in notes) {
            notesFeed.add(note)
            emit(notesFeed.toList())
            delay(FEED_DELAY)
        }
        delay(REFRESH_DELAY)
    }
}
```

notes.kt

Обновление заметок

```
28  fun getNotesFlow(userId: Int?, tags: Set<Int>): Flow<List<Note>> = flow {  this: FlowCollector<List<Note>>
29      val notes = getNotesForUser(userId, tags).getOrThrow()
30      if (notes.isEmpty()) {
31          emit(emptyList())
32          return@flow
33      }
34
35      while(currentCoroutineContext().isActive) {
36          val notesFeed = mutableListOf<Note>()
37          for (note in notes) {
38              notesFeed.add(note)
39              emit(notesFeed.toList())
40              delay(FEED_DELAY)
41          }
42          delay(REFRESH_DELAY)
43      }
44  }
```

Обновление

notes.kt

Обновление заметок

Подписка на изменения

```
private fun loadNotes() {
    val userId = getSelectedUser()
    val tags = getSelectedTags()
    feedSubscription?.cancel()

    Log.i(TAG, msg: "Loading notes for user: $userId and tags: $tags")

    if (null == userId) {
        populateNotes(emptyList())
        return
    }

    feedSubscription = lifecycleScope.launch { this: CoroutineScope
        getNotesFlow(userId, tags).collect { it: List<Note>
            populateNotes(it)
        }
    }
}
```

MainActivity.kt

Обновление заметок

```
private fun populateNotes(notes: List<Note>) {
    Log.i(TAG,  msg: "Populating notes: $notes")
    adapter.submitList(notes)
}
```

MainActivity.kt

Вопросы?

Преобразования и объединения

Преобразование

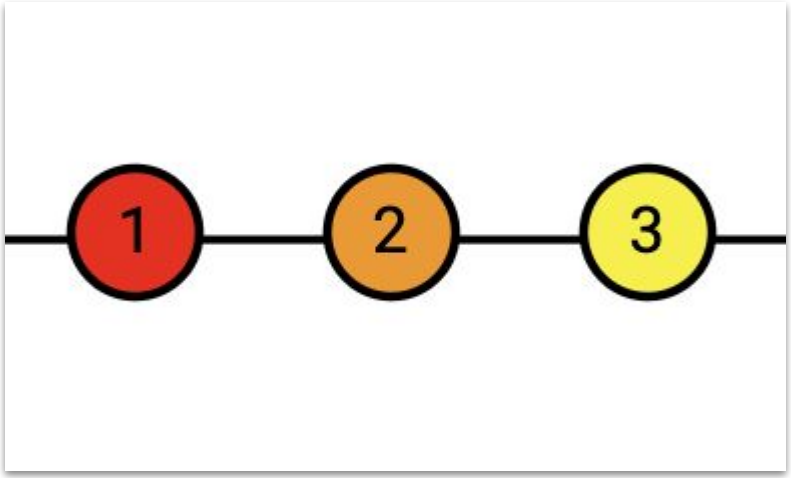
```
// 2, 4, 6
val mapped = flowOf( ...elements: 1, 2, 3).map { it * 2 }
// 2, 4
val filtered = flowOf( ...elements: 1, 2, 3, 4).filter { it % 2 == 0 }
// 2, 4, 6
val mappedNotNull = flowOf( ...elements: 1, 2, 3, null).mapNotNull { it?.times( other: 2) }
```

mapping.kt

Kotlin Flow

kotlinx.coroutines.flow

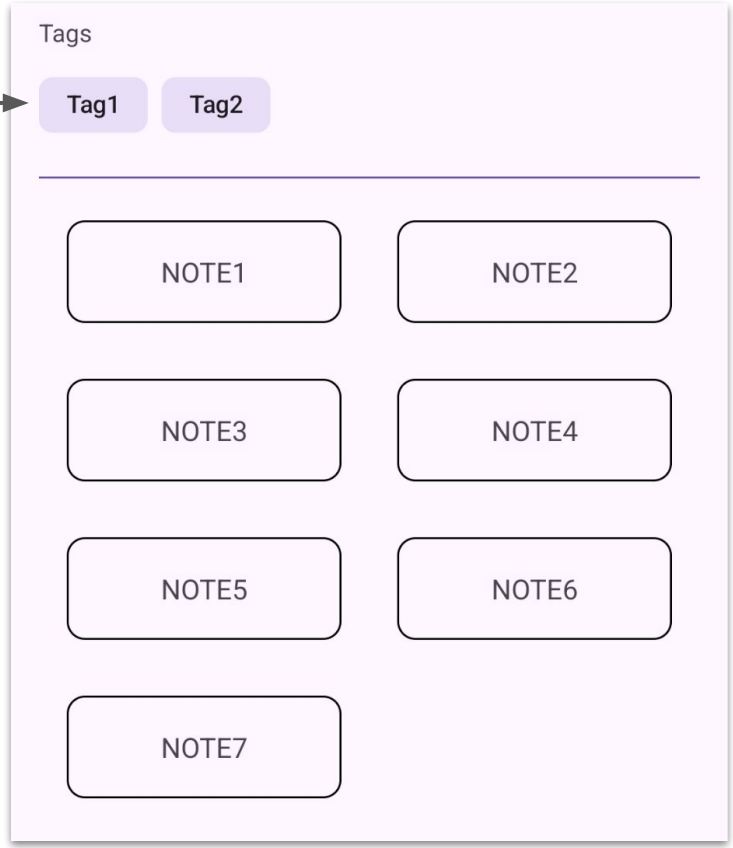
Flow Marbles



flowmarbles.com

Поток тэгов

Поток выбора тэгов



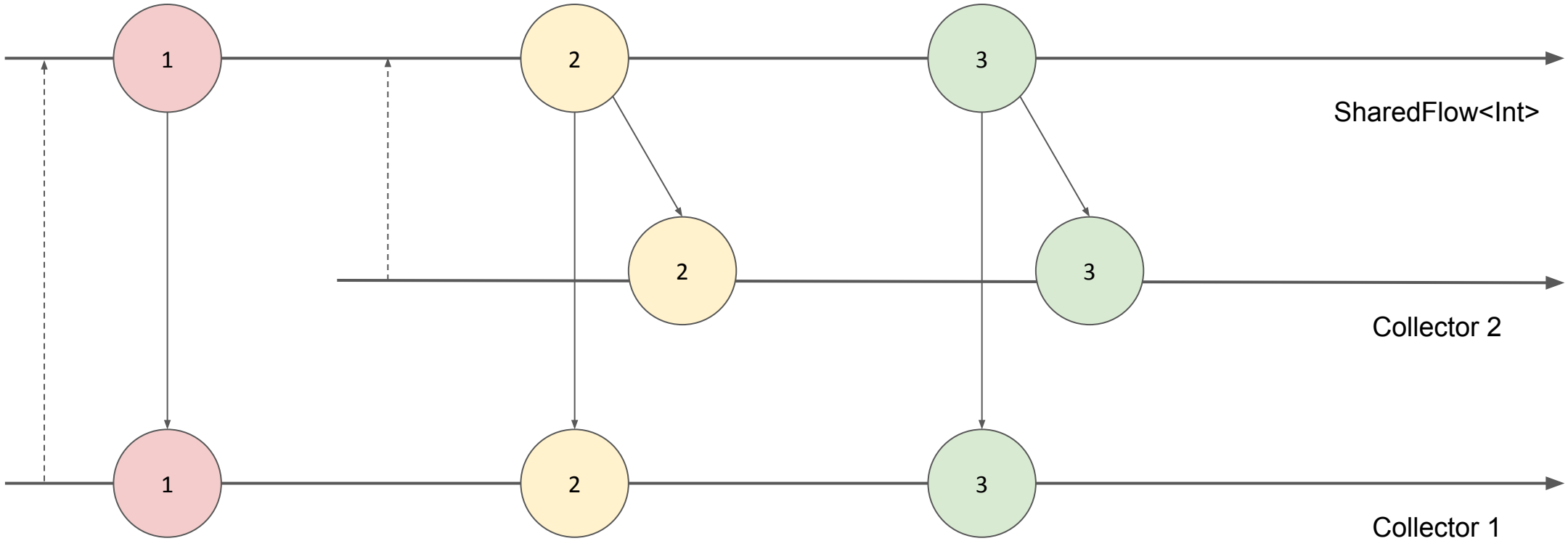
Поток тэгов

Горячие потоки

Hot Flow

Observable называется **горячим**, когда его активный экземпляр существует независимо от наличия observer’ов

Shared Flow



Shared Flow

```
val sharedFlow = MutableSharedFlow<Int>()
```

sharedFlow.kt

Shared Flow

```
val collector = launch { this: CoroutineScope
    sharedFlow.collect { value -> println("Collected: $value") }
}

val lateCollector = launch { this: CoroutineScope
    delay( timeMillis: 100)
    sharedFlow.collect { value -> println("Late collected: $value") }
}
```

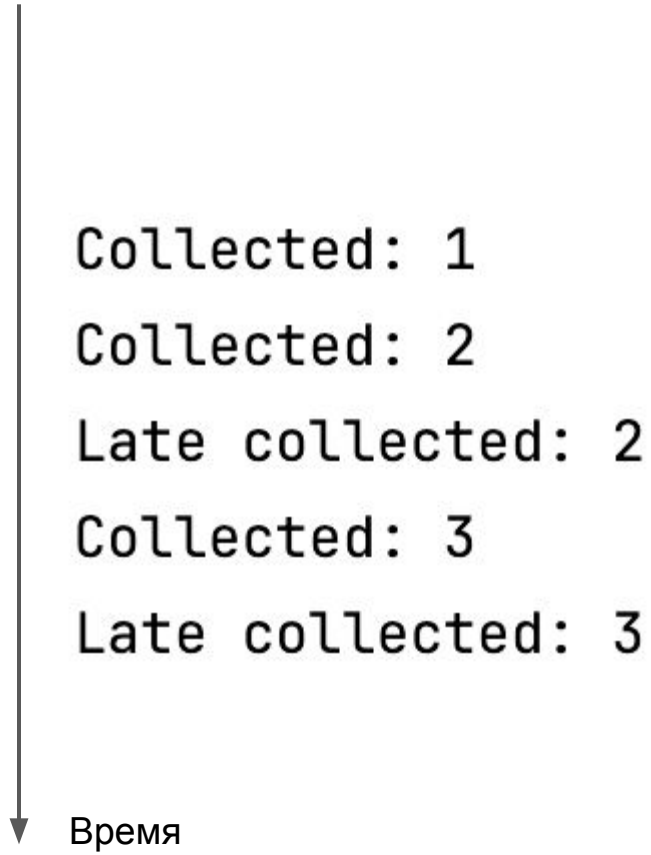
sharedFlow.kt

Shared Flow

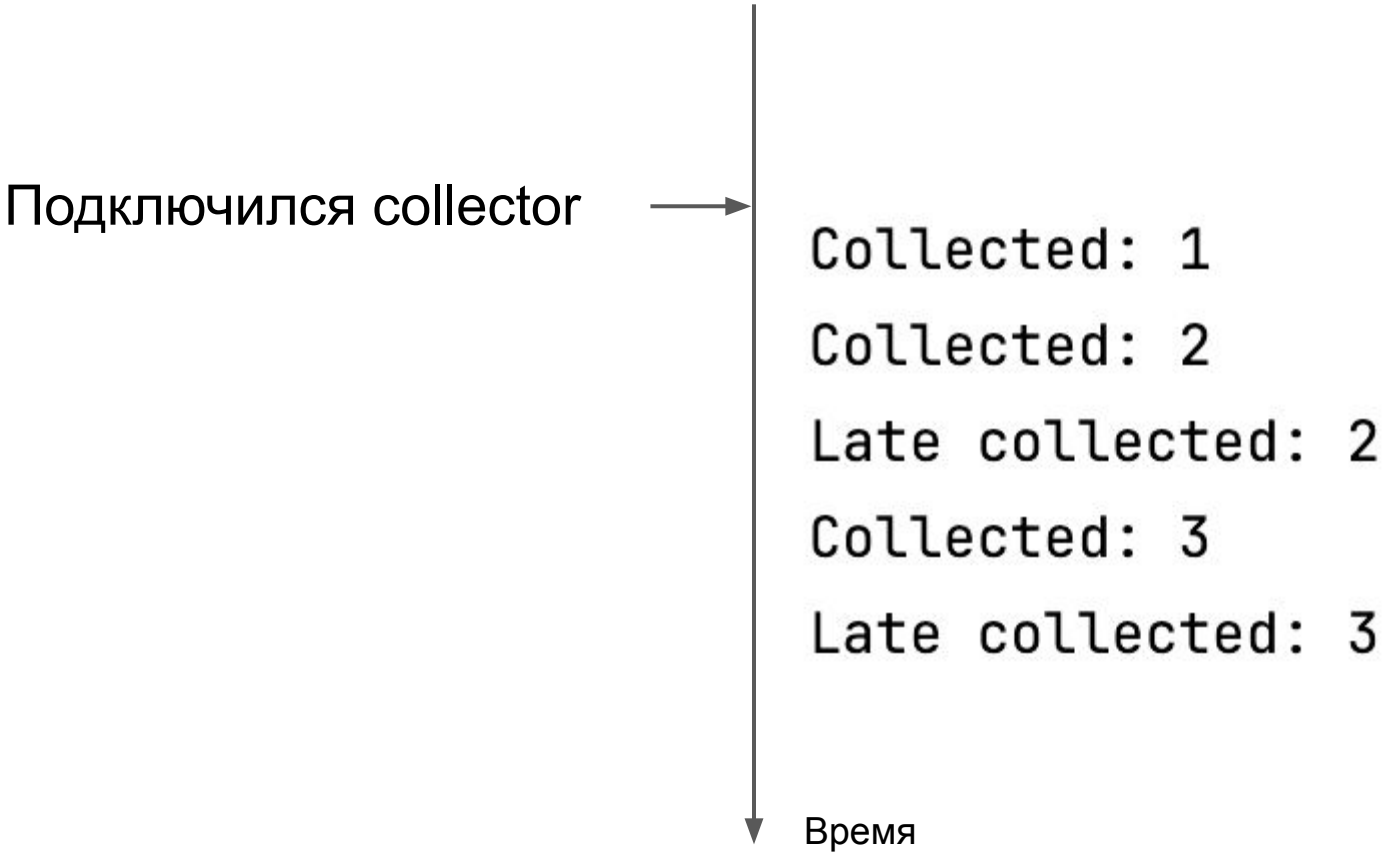
```
delay( timeMillis: 50)
sharedFlow.emit( value: 1)
delay( timeMillis: 50)
sharedFlow.emit( value: 2)
delay( timeMillis: 50)
sharedFlow.emit( value: 3)
```

sharedFlow.kt

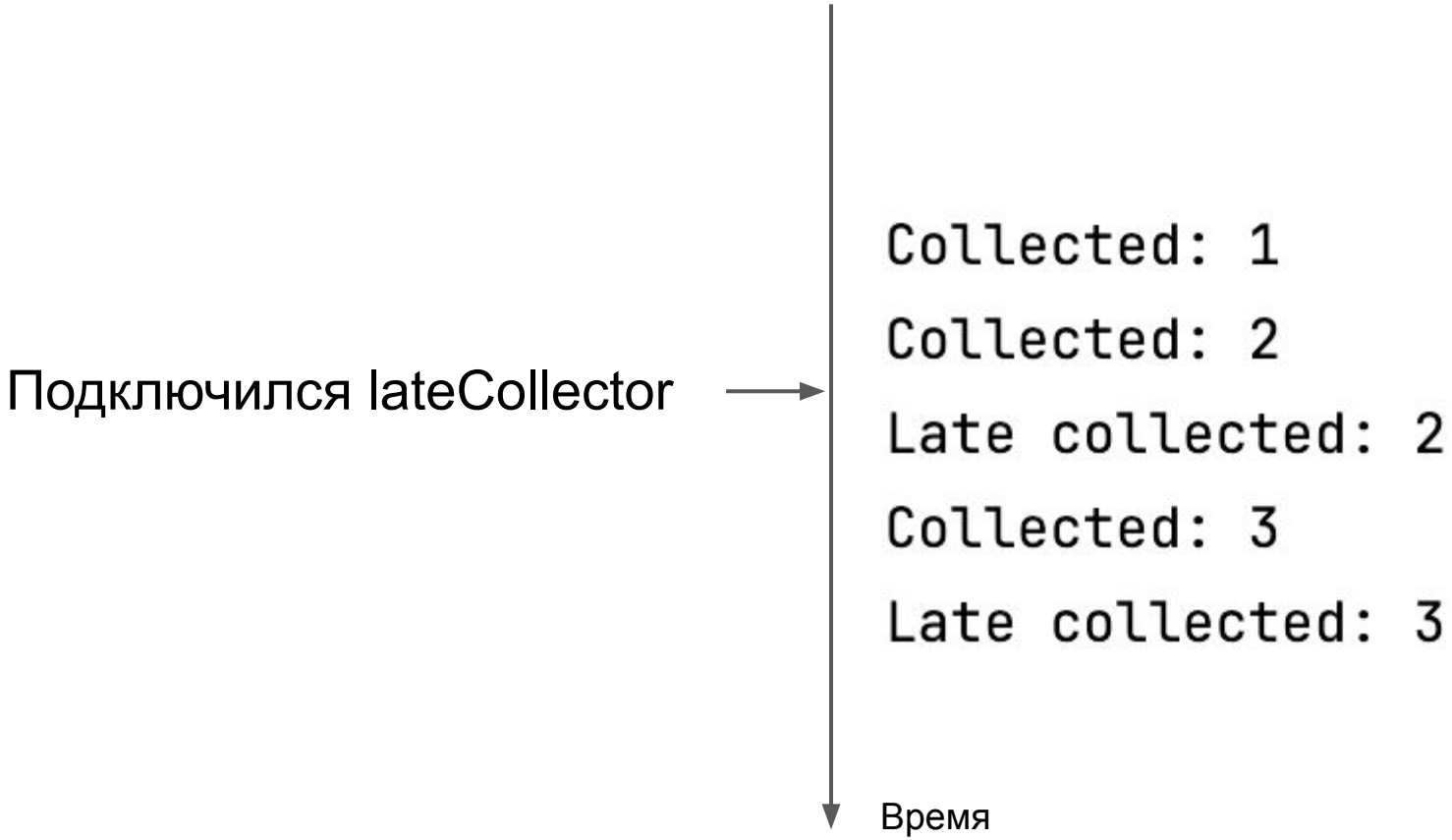
Shared Flow


Collected: 1
Collected: 2
Late collected: 2
Collected: 3
Late collected: 3
Время

Shared Flow


Подключился collector
Collected: 1
Collected: 2
Late collected: 2
Collected: 3
Late collected: 3
Время

Shared Flow


Подключился lateCollector
Collected: 1
Collected: 2
Late collected: 2
Collected: 3
Late collected: 3
Время

Shared Flow

```kotlin
fun main(): Unit = runBlocking { this: CoroutineScope
    val sharedFlow = MutableSharedFlow<Int>()

    val collector = launch { this: CoroutineScope
        sharedFlow.collect { value -> println("Collected: $value") }
    }

    val lateCollector = launch { this: CoroutineScope
        delay( timeMillis: 100)
        sharedFlow.collect { value -> println("Late collected: $value") }
    }

    // region Emit values
    delay( timeMillis: 50)
    sharedFlow.emit( value: 1)
    delay( timeMillis: 50)
    sharedFlow.emit( value: 2)
    delay( timeMillis: 50)
    sharedFlow.emit( value: 3)
    // endregion

    collector.cancelAndJoin()
    lateCollector.cancelAndJoin()
    println("Done")
}
```

sharedFlow.kt

Shared Flow

```kotlin
fun main(): Unit = runBlocking { this: CoroutineScope
    val sharedFlow = MutableSharedFlow<Int>()

    val collector = launch { this: CoroutineScope
        sharedFlow.collect { value -> println("Collected: $value") }
    }

    val lateCollector = launch { this: CoroutineScope
        delay( timeMillis: 100)
        sharedFlow.collect { value -> println("Late collected: $value") }
    }

    Emit values

    collector.cancelAndJoin()
    lateCollector.cancelAndJoin()
    println("Done")
}
```

Горячий поток никогда не заканчивается!

sharedFlow.kt

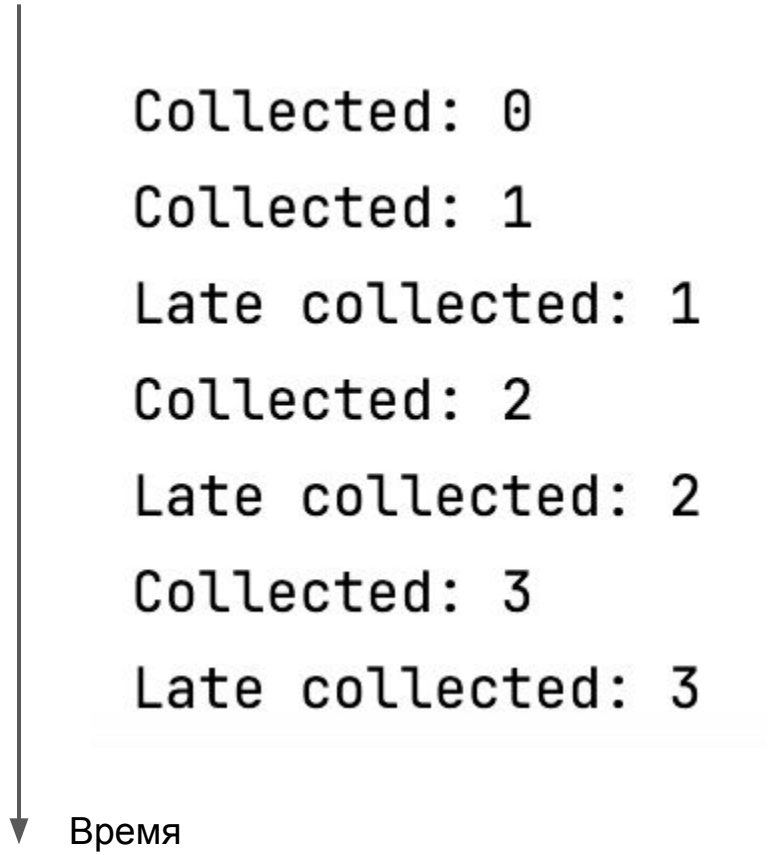
Вопросы?

State Flow

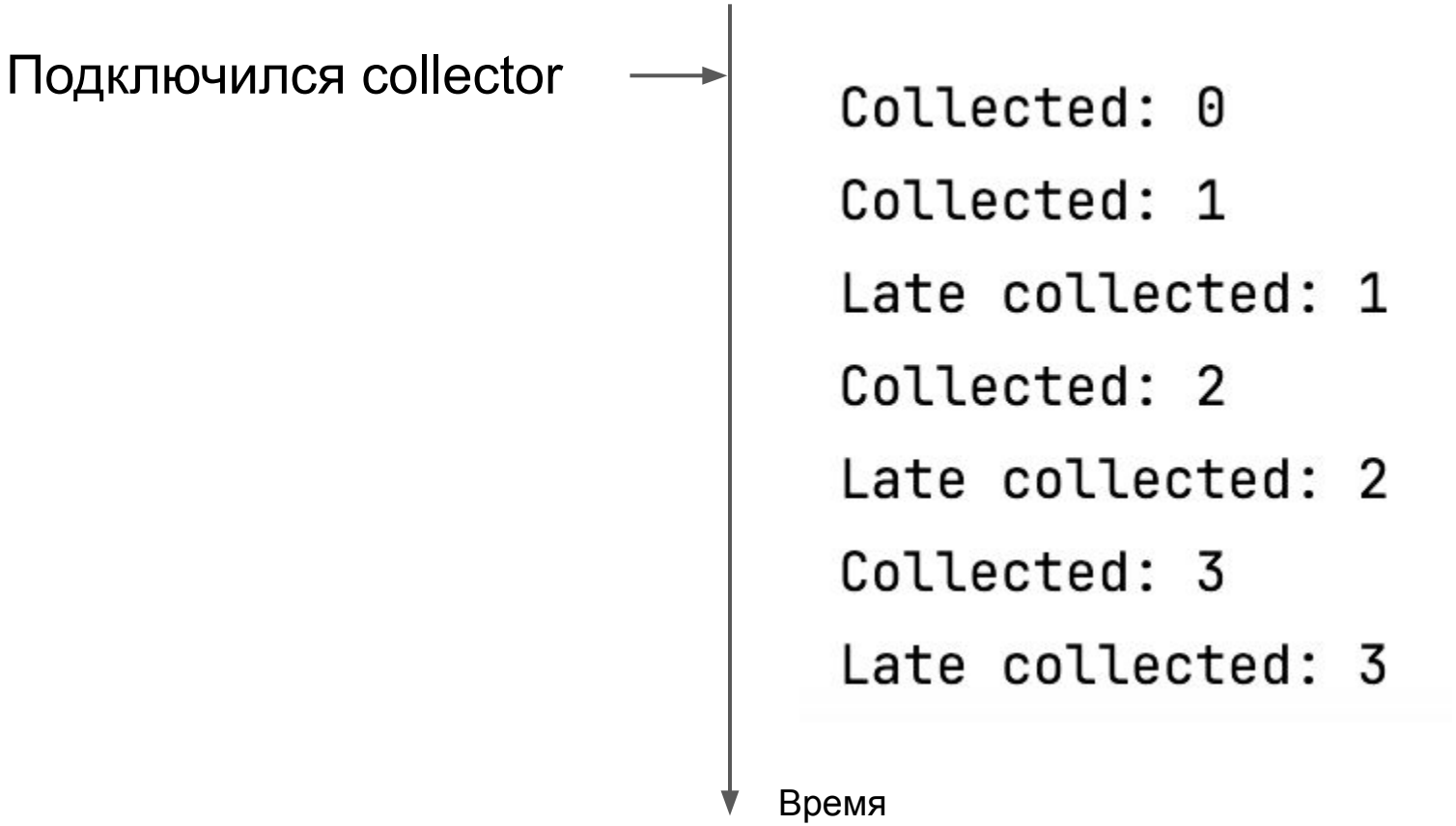
```
val stateFlow = MutableStateFlow( value: 0)
```

stateFlow.kt

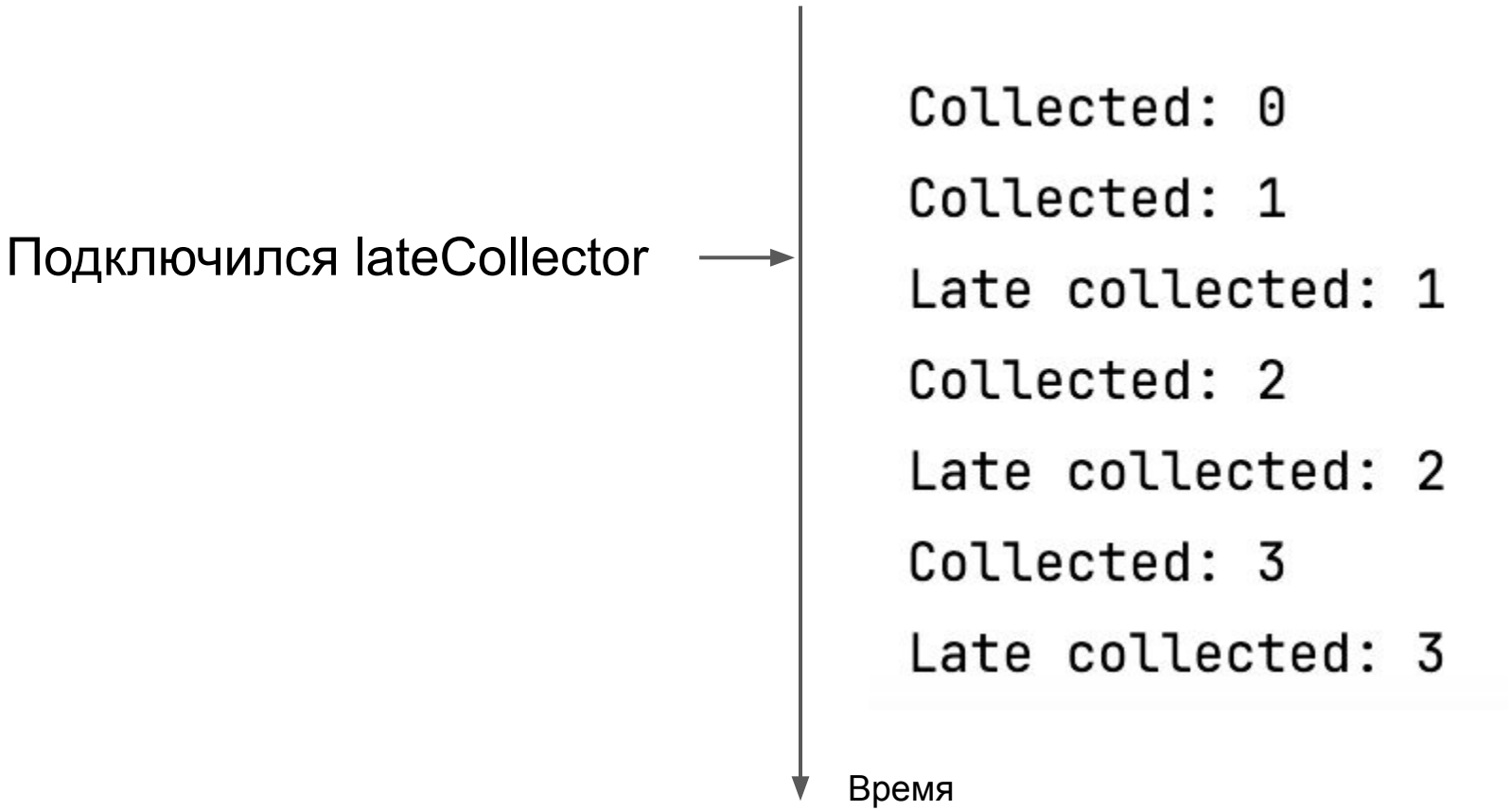
State Flow


Collected: 0
Collected: 1
Late collected: 1
Collected: 2
Late collected: 2
Collected: 3
Late collected: 3
Время

State Flow

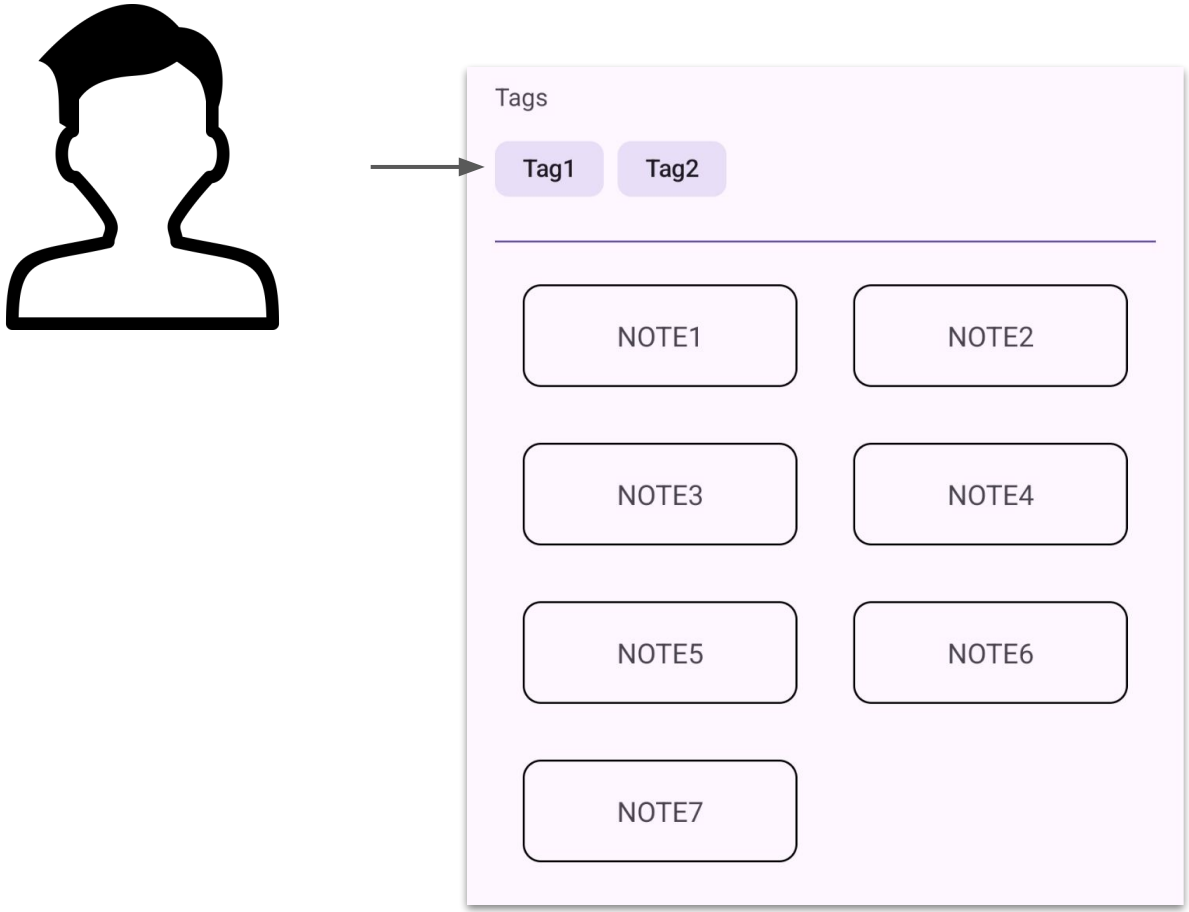

Подключился collector
Collected: 0
Collected: 1
Late collected: 1
Collected: 2
Late collected: 2
Collected: 3
Late collected: 3
Время

State Flow


Подключился lateCollector
Collected: 0
Collected: 1
Late collected: 1
Collected: 2
Late collected: 2
Collected: 3
Late collected: 3
Время

Вопросы?

Поток тэгов зависит только от пользователя



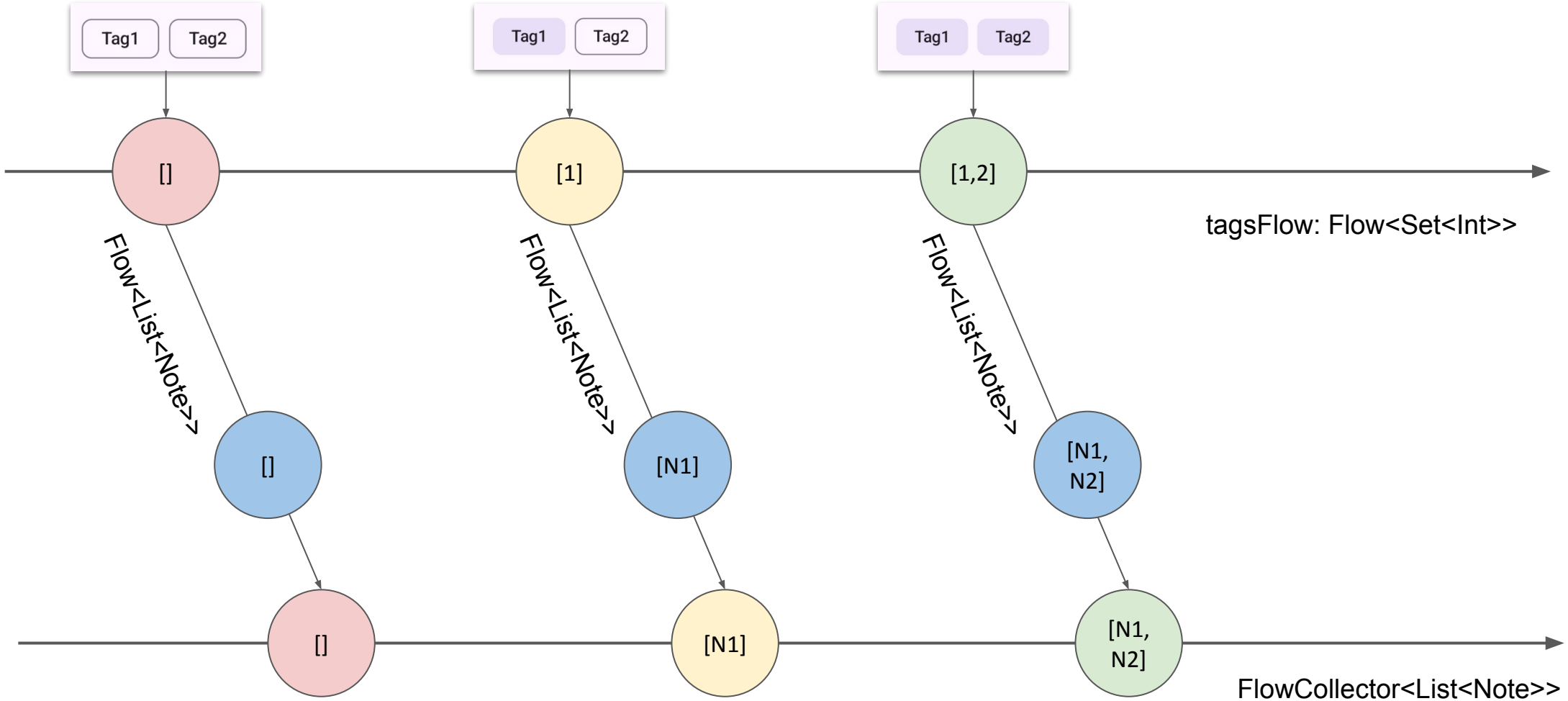
Поток тэгов

```
Keeps tags to filter notes

private val tagsFlow = MutableStateFlow(emptySet<Int>())
```

GetUserNotes.kt

Поток заметок по выбранным тэгам



Вопросы?

flatMapLatest

Возвращает поток, который переключается на **новый поток**, созданный функцией преобразования, каждый раз, когда исходный поток испускает значение. Когда исходный поток испускает новое значение, предыдущий поток, созданный блоком преобразования, **отменяется**.

flatMapLatest

```
val flow: Flow<Int> = flow { this: FlowCollector<Int>
    emit( value: 1)
    delay( timeMillis: 45)
    emit( value: 2)
    delay( timeMillis: 25)
    emit( value: 3)
}
```

flatMapLatest.kt

flatMapLatest

Для каждого T

```
fun <T, R> Flow<T>.flatMapLatest(transform: suspend (value: T) -> Flow<R>): Flow<R>
```

flatMapLatest

Выполнить transform


```
fun <T, R> Flow<T>.flatMapLatest(transform: suspend (value: T) -> Flow<R>): Flow<R>
```

flatMapLatest

Вернуть поток R

```
fun <T, R> Flow<T>.flatMapLatest(transform: suspend (value: T) -> Flow<R>): Flow<R>
```

flatMap для обычной коллекции

```
val source = listOf(1, 2, 3)
val result = source.flatMap { fromUpstream ->
    listOf(
        "$fromUpstream - 1",
        "$fromUpstream - 2",
        "$fromUpstream - 3"
    )
}
println(result)
```

```
[1 - 1, 1 - 2, 1 - 3, 2 - 1, 2 - 2, 2 - 3, 3 - 1, 3 - 2, 3 - 3]
```

flatMapLatest

Когда исходный поток испускает новое значение, предыдущий поток, созданный блоком преобразования, **отменяется**.

flatMapLatest

```
val mapped: Flow<String> = flow.flatMapLatest { fromUpstream ->
    flow { this: FlowCollector<String>
        repeat( times: 3) { count ->
            delay( timeMillis: 10)
            emit( value: "Transformed $fromUpstream - ${count + 1}")
        }
    }
}
```

flatMapLatest.kt

flatMapLatest

Запускаем новый поток

```
val mapped: Flow<String> = flow.flatMapLatest { fromUpstream ->
    flow { this: FlowCollector<String>
        repeat( times: 3) { count ->
            delay( timeMillis: 10)
            emit( value: "Transformed $fromUpstream - ${count + 1}")
        }
    }
}
```

flatMapLatest.kt

flatMapLatest

Три элемента с задержкой

```
val mapped: Flow<String> = flow.flatMapLatest { fromUpstream ->
    flow { this: FlowCollector<String>
        repeat( times: 3) { count ->
            delay( timeMillis: 10)
            emit( value: "Transformed $fromUpstream - ${count + 1}")
        }
    }
}
```

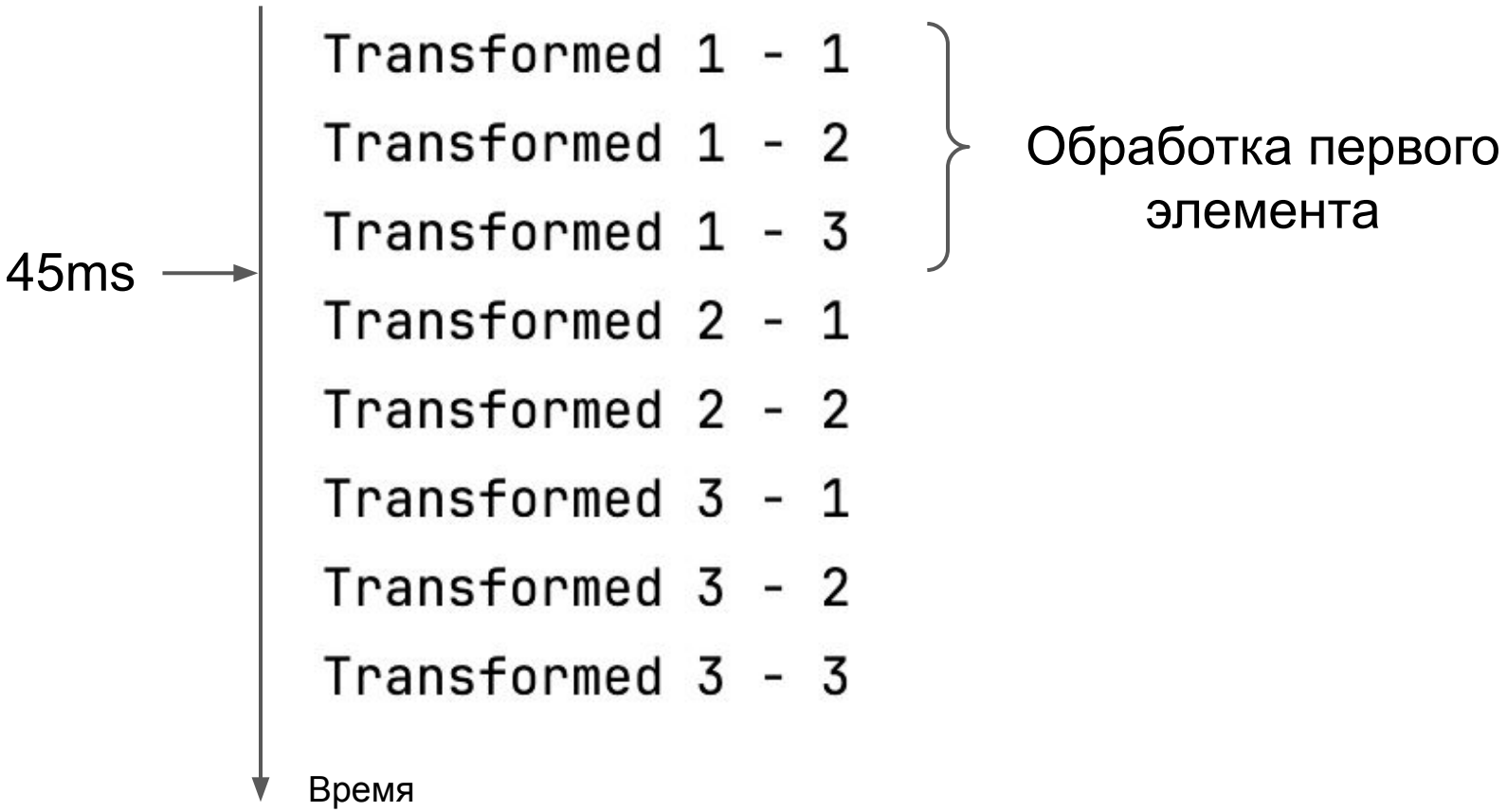
flatMapLatest.kt

flatMapLatest

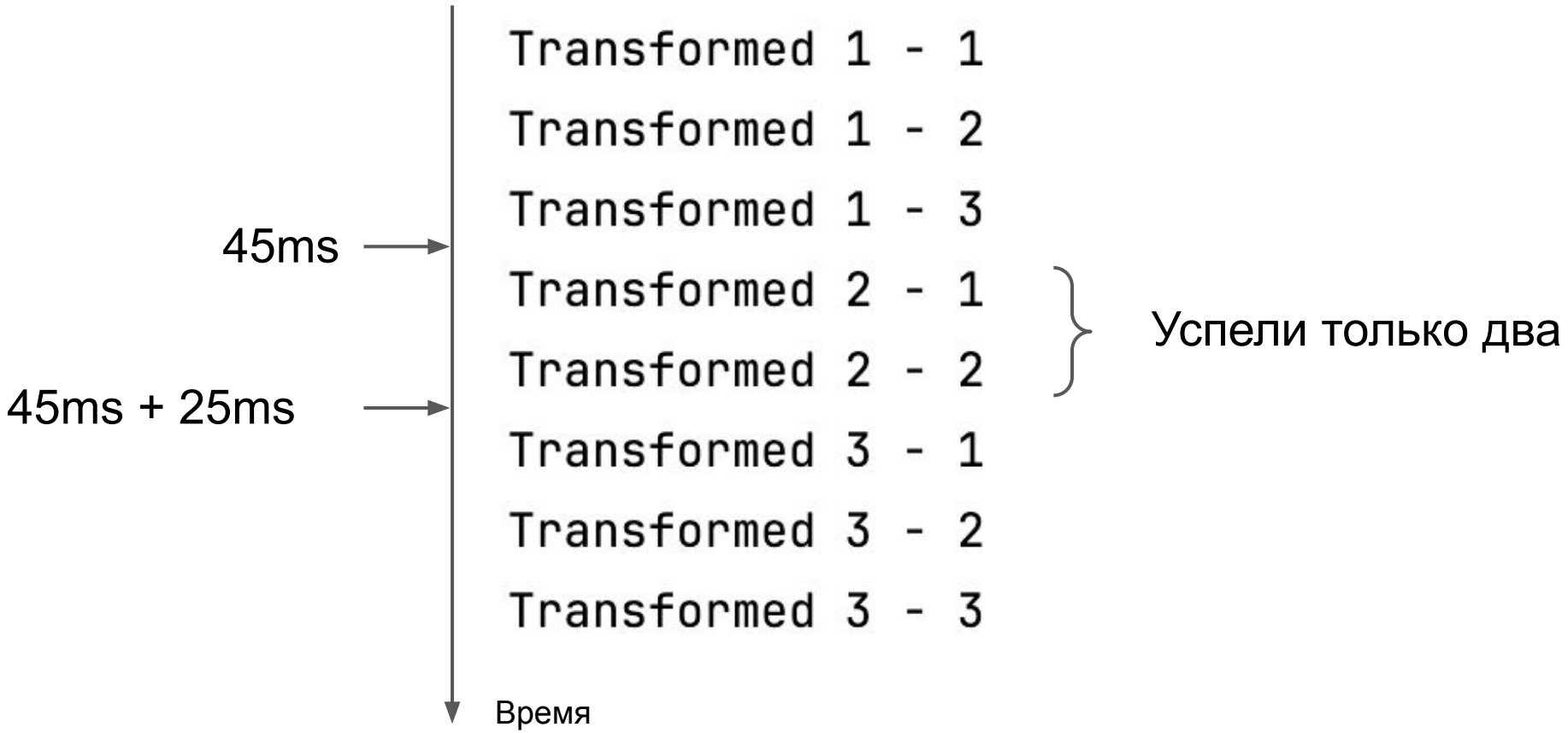
```
val flow: Flow<Int> = flow { this: FlowCollector<Int>
    emit( value: 1)
    delay( timeMillis: 45)
    emit( value: 2)
    delay( timeMillis: 25)
    emit( value: 3)
}
```

flatMapLatest.kt

State Flow


Transformed 1 - 1
Transformed 1 - 2
Transformed 1 - 3
Transformed 2 - 1
Transformed 2 - 2
Transformed 3 - 1
Transformed 3 - 2
Transformed 3 - 3
45ms
Время
Обработка первого элемента

State Flow


Transformed 1 - 1
Transformed 1 - 2
Transformed 1 - 3
45ms
Transformed 2 - 1
Transformed 2 - 2
45ms + 25ms
Успели только два
Transformed 3 - 1
Transformed 3 - 2
Transformed 3 - 3
Время

State Flow


Transformed 1 - 1
Transformed 1 - 2
Transformed 1 - 3
45ms
Transformed 2 - 1
Transformed 2 - 2
45ms + 25ms
Transformed 3 - 1
Transformed 3 - 2
Transformed 3 - 3
Переключились на третий!
Время

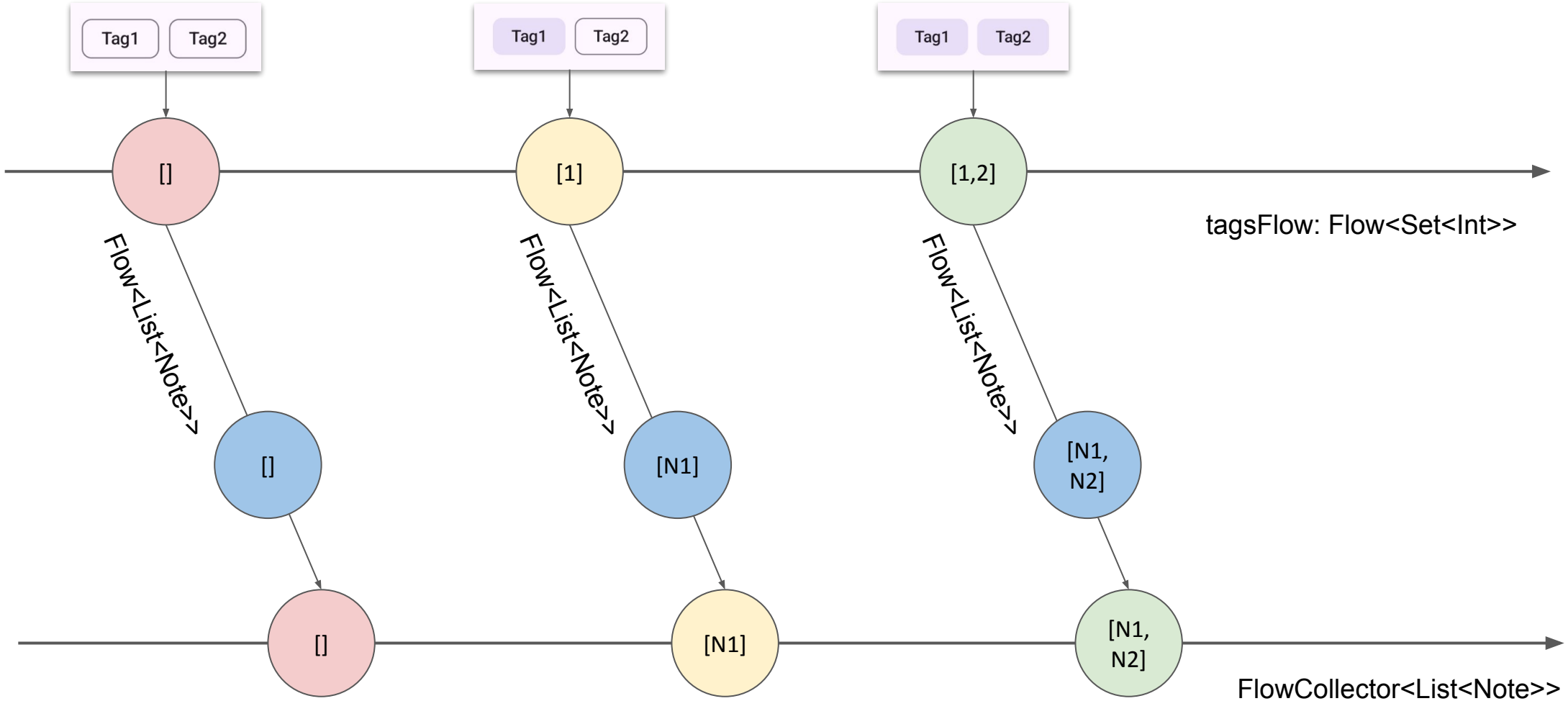
flatMapLatest

Когда исходный поток испускает новое значение, предыдущий поток, созданный блоком преобразования, **отменяется**.

Операторы сочетания потоков

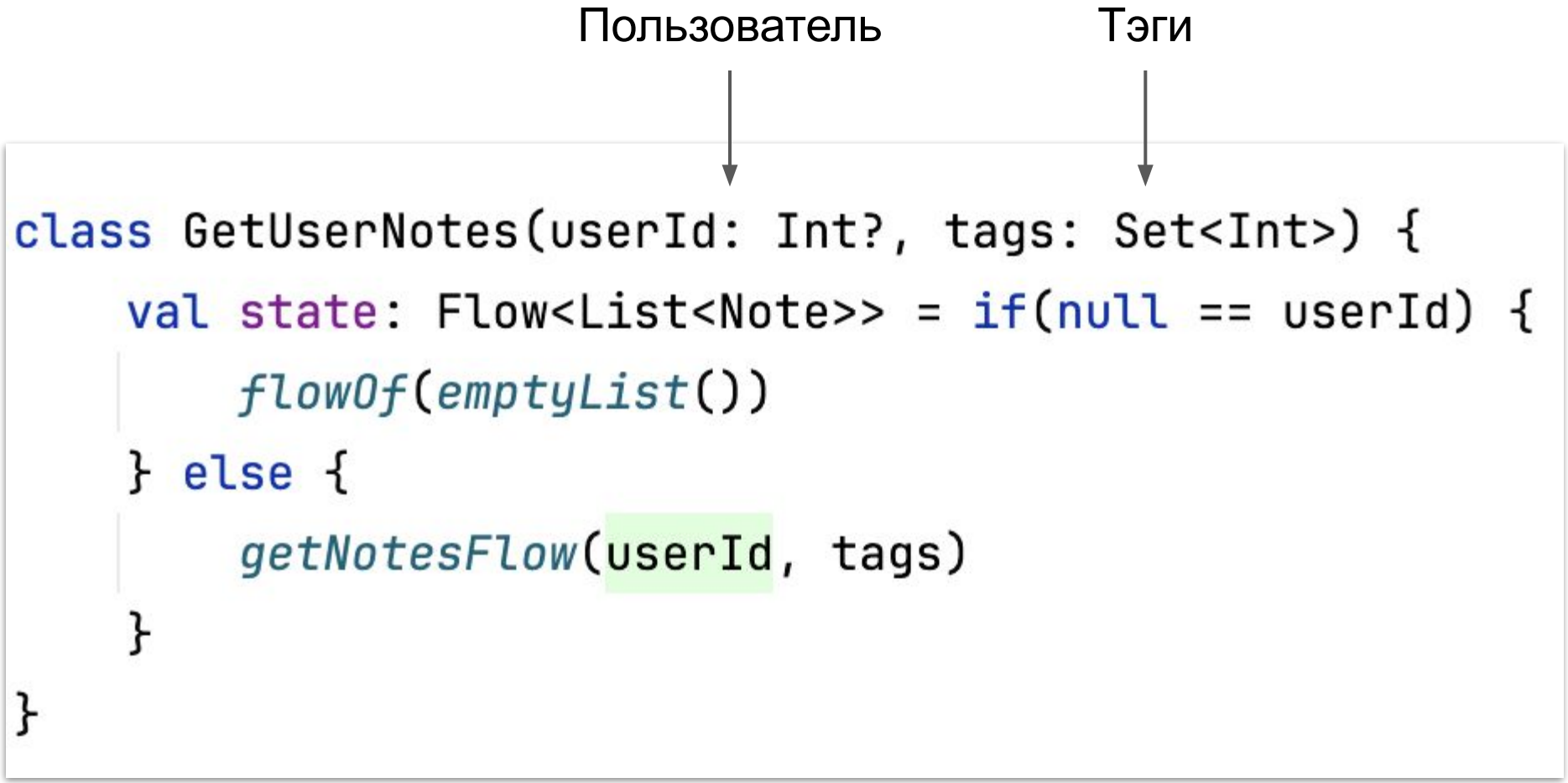
- flatMapConcat - последовательные элементы вторичного потока целиком
- flatMapMerge - смешанные элементы вторичных потоков

Поток заметок по выбранным тэгам



Вопросы?

Поток заметок



GetUserNotes.kt

Поток заметок

```
feedSubscription = lifecycleScope.launch { this: CoroutineScope
    GetUserNotes(userId, tags).state.collect { it: List<Note>
        populateNotes(it)
    }
}
```

MainActivity.kt

Вопросы?

Поток заметок

```
class GetUserNotes(userId: Int?) {

    Keeps tags to filter notes

    private val tagsFlow = MutableStateFlow(emptySet<Int>())

    fun setTags(tags: Set<Int
        tagsFlow.value = tags
    }

    val state: Flow<List<Note>> = if(null == userId) {...}
}
```

GetUserNotes.kt

Поток заметок

```
private fun selectTags(tags: Set<Int>) {
    Log.i(TAG, msg: "Selected tags: $tags")
    getUserNotes.setTags(tags)
}
```

MainActivity.kt

Поток заметок

Для каждого тэга

```
val state: Flow<List<Note>> = if(null == userId) {
    flowOf(emptyList())
} else {
    // For each tag change get notes
    tagsFlow.flatMapLatest { tags ->
        getNotesFlow(userId, tags)
    }
}
```

GetUserNotes.kt

Поток заметок

Создать фильтрованный поток заметок


```
val state: Flow<List<Note>> = if(null == userId) {
    flowOf(emptyList())
} else {
    // For each tag change get notes
    tagsFlow.flatMapLatest { tags ->
        getNotesFlow(userId, tags)
    }
}
```

GetUserNotes.kt

Поток заметок по выбранным тэгам

```
11  @OptIn(ExperimentalCoroutinesApi::class)
12  class GetUserNotes(userId: Int?) {
13
        Keeps tags to filter notes
17      private val tagsFlow = MutableStateFlow(emptySet<Int>())
18
19      fun setTags(tags: Set<Int>) {
20          tagsFlow.value = tags
21      }
22
23      val state: Flow<List<Note>> = if(null == userId) {
24          flowOf(emptyList())
25      } else {
26          // For each tag change get notes
27          tagsFlow.flatMapLatest { tags ->
28              getNotesFlow(userId, tags)
29          }
30      }
31  }
```

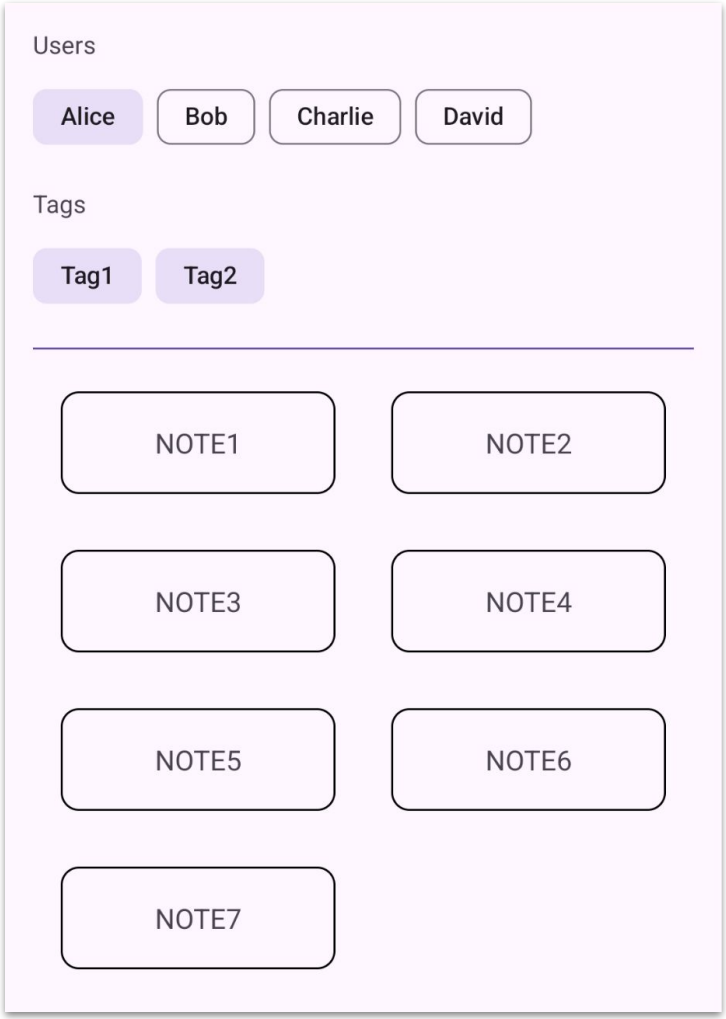
Вопросы?

Полный контент пользователя

Входные потоки

Поток пользователя

Поток тэгов



Данные пользователя

```
data class UserContent(
    val userId: Int?,
    val tags: List<Pair<Tag, Boolean>>,
    val notes: List<Note>
)
```

GetUserContent.kt

Данные пользователя

Набор и
состояние тэгов


```
data class UserContent(
    val userId: Int?,
    val tags: List<Pair<Tag, Boolean>>,
    val notes: List<Note>
)
```

GetUserContent.kt

Данные пользователя

Набор заметок

```
data class UserContent(
    val userId: Int?,
    val tags: List<Pair<Tag, Boolean>>,
    val notes: List<Note>
)
```

GetUserContent.kt

Данные пользователя

```
val state: Flow<UserContent>
```

GetUserContent.kt

Данные пользователя - обработка

```
private fun subscribeUserContent() {
    getUserContent.state.onEach { content ->
        populateTags(content.tags)
        populateNotes(content.notes)
        if (content.tags.isEmpty()) {
            hideTags()
        } else {
            showTags()
        }
    }.launchIn(lifecycleScope)
}
```

MainActivity.kt

Выбранный пользователь

```
Keeps user to filter notes

private val userFlow = MutableStateFlow<Int?>( value: null)


Sets user to get content for

fun setUser(userId: Int?) {
    userFlow.value = userId
}
```

GetUserContent.kt

Выбранные тэги

```
Keeps tags to filter notes

private val tagsFlow = MutableStateFlow(emptySet<Int>())

Sets tags to filter notes

fun setTags(tags: Set<Int>) {
    tagsFlow.value = tags
}
```

GetUserContent.kt

Поток состояния

```
val state: Flow<UserContent> = userFlow.transformLatest { this: FlowCollector<UserContent> user ->
    emit(UserContent(user, emptyList(), emptyList()))

    getTagsFlow(user).collectLatest { tags ->
        emit(UserContent(user, tags.map { it to false }, emptyList()))

        tagsFlow.onStart { emit(emptySet()) }.collectLatest { selectedTags ->
            emitAll(
                getNotesFlow(user, selectedTags).map { it: List<Note>
                    UserContent(
                        user,
                        tags.map { tag -> tag to selectedTags.contains(tag.id) },
                        it
                    )
                }
            )
        }
    }
}
```

GetUserContent.kt

Поток состояния

transformLatest

```kotlin
val state: Flow<UserContent> = userFlow.transformLatest { this: FlowCollector<UserContent> user ->
    emit(UserContent(user, emptyList(), emptyList()))

    getTagsFlow(user).collectLatest { tags ->
        emit(UserContent(user, tags.map { it to false }, emptyList()))

        tagsFlow.onStart { emit(emptySet()) }.collectLatest { selectedTags ->
            emitAll(
                getNotesFlow(user, selectedTags).map { it: List<Note>
                    UserContent(
                        user,
                        tags.map { tag -> tag to selectedTags.contains(tag.id) },
                        it
                    )
                }
            )
        }
    }
}
```

GetUserContent.kt

Поток состояния

Обнуляем →

```
val state: Flow<UserContent> = userFlow.transformLatest { this: FlowCollector<UserContent> user ->
    emit(UserContent(user, emptyList(), emptyList()))

    getTagsFlow(user).collectLatest { tags ->
        emit(UserContent(user, tags.map { it to false }, emptyList()))

        tagsFlow.onStart { emit(emptySet()) }.collectLatest { selectedTags ->
            emitAll(
                getNotesFlow(user, selectedTags).map { it: List<Note>
                    UserContent(
                        user,
                        tags.map { tag -> tag to selectedTags.contains(tag.id) },
                        it
                    )
                }
            )
        }
    }
}
```

GetUserContent.kt

Поток состояния

Следим за тэгами

```
val state: Flow<UserContent> = userFlow.transformLatest { this: FlowCollector<UserContent> user ->
    emit(UserContent(user, emptyList(), emptyList()))

    getTagsFlow(user).collectLatest { tags ->
        emit(UserContent(user, tags.map { it to false }, emptyList()))

        tagsFlow.onStart { emit(emptySet()) }.collectLatest { selectedTags ->
            emitAll(
                getNotesFlow(user, selectedTags).map { it: List<Note>
                    UserContent(
                        user,
                        tags.map { tag -> tag to selectedTags.contains(tag.id) },
                        it
                    )
                }
            )
        }
    }
}
```

GetUserContent.kt

Вопросы?

Управление экраном заметок (императивный подход)

70 строк кода

```
91      // region Logic
92
93      private fun getSelectedUser(): Int? {...}
101
102     private fun getSelectedTags(): Set<Int> {...}
107
108     private fun loadUsers() {...}
119
120     private fun selectUser(userId: Int?) {...}
124
125     private fun loadTags() {...}
146
147     private fun selectTags(tags: Set<Int>) {...}
151
152     private fun loadNotes() {...}
171
172     // endregion
```

7 функций

MainActivity.kt

Управление экраном заметок (реактивный подход)

20 строк кода

```
42      val state: Flow<UserContent> = userFlow.transformLatest { this: FlowCollector<UserContent> user ->
43          emit(UserContent(user, emptyList(), emptyList()))
44
45          getTagsFlow(user).collectLatest { tags ->
46              emit(UserContent(user, tags.map { it to false }, emptyList()))
47
48              tagsFlow.onStart { emit(emptySet()) }.collectLatest { selectedTags ->
49                  emitAll(
50                      getNotesFlow(user, selectedTags).map { it: List<Note>
51                          UserContent(
52                              user,
53                              tags.map { tag -> tag to selectedTags.contains(tag.id) },
54                              it
55                          )
56                      }
57                  )
58              }
59          }
60      }
```

GetUserContent.kt

Вопросы?

Обработка ошибок

Ошибка в билдере

Ошибка

```
fun main(): Unit = runBlocking { this: CoroutineScope
    val flowWithError = flow { this: FlowCollector<Int>
        emit( value: 1)
        emit( value: 2)
        throw NumberFormatException("Not a number")
    }

    try {
        flowWithError.collect { value -> println(value) }
    } catch (e: NumberFormatException) {
        println("Caught exception: $e")
    }
}
```

error/catchingFaultyBuilder.kt

Ошибка в билдере

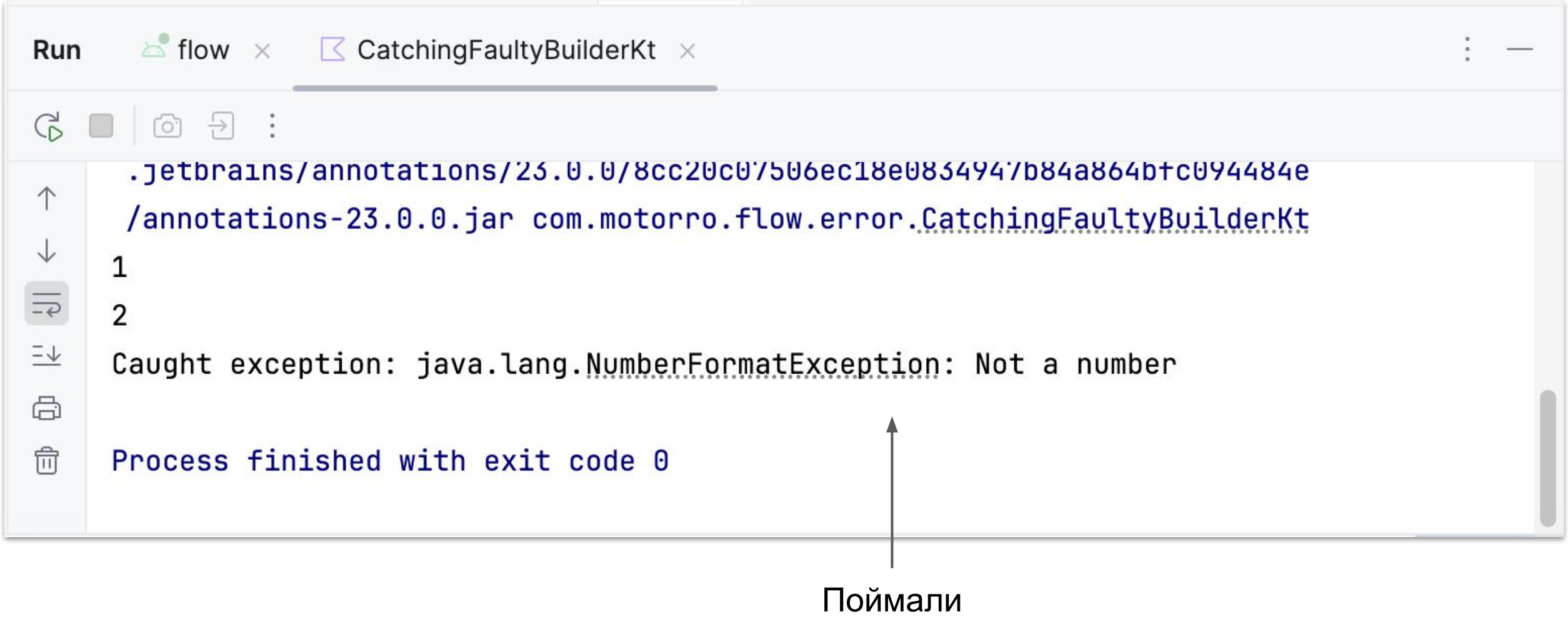
```
fun main(): Unit = runBlocking { this: CoroutineScope
    val flowWithError = flow { this: FlowCollector<Int>
        emit( value: 1)
        emit( value: 2)
        throw NumberFormatException("Not a number")
    }

    try {
        flowWithError.collect { value -> println(value) }
    } catch (e: NumberFormatException) {
        println("Caught exception: $e")
    }
}
```

Обработка коллектора

error/catchingFaultyBuilder.kt

Ошибка в билдере



Поймали

Ошибка в коллекторе

```
fun main(): Unit = runBlocking { this: CoroutineScope
    val flowWithError = flow { this: FlowCollector<Int>
        emit( value: 1)
        emit( value: 2)
        emit( value: 3)
    }

    try {
        flowWithError.collect { throw NumberFormatException("Faulty collector") }
    } catch (e: NumberFormatException) {
        println("Caught exception: $e")
    }
}
```

Ошибка в коллекторе

error/catchingFaultyCollector.kt

Ошибка в коллекторе

Получить ноль, вместо ошибки?

```
fun main(): Unit = runBlocking { this: CoroutineScope
    val flowWithError = flow { this: FlowCollector<Int>
        emit( value: 1)
        emit( value: 2)
        throw NumberFormatException("Not a number")
    }
```

error/catchOperator.kt

Оператор catch

Обработка ошибки

```
fun main(): Unit = runBlocking { this: CoroutineScope
    val flowWithError = flow { this: FlowCollector<Int>
        emit( value: 1)
        emit( value: 2)
        throw NumberFormatException("Not a number")
    }.catch { this: FlowCollector<Int>  e ->
        println("Caught exception: $e")
        emit( value: 0)
    }

    flowWithError.collect { value -> println(value) }
}
```

error/catchOperator.kt

Оператор catch - возможности

- Можно вернуть какое-то значение
- Можно перевыбросить исключение
- Можно проигнорировать и залогировать исключение

Оператор catch - только upstream

Поймает →

```
fun main(): Unit = runBlocking { this: CoroutineScope
    val flow = flowOf( ...elements: 1, 2, 3)
        .map { if (2 == it) throw NumberFormatException("Faulty number 2") else it }
        .catch { this: FlowCollector<Int>  e ->
            println("Caught exception: $e")
            emit( value: 3)
        }
        .map { if (3 == it) throw NumberFormatException("Faulty number 3") else it }

    flow.collect { value -> println(value) }
}
```

Оператор catch - только upstream

```
fun main(): Unit = runBlocking { this: CoroutineScope
    val flow = flowOf( ...elements: 1, 2, 3)
        .map { if (2 == it) throw NumberFormatException("Faulty number 2") else it }
        .catch { this: FlowCollector<Int>  e ->
            println("Caught exception: $e")
            emit( value: 3)
        }
        .map { if (3 == it) throw NumberFormatException("Faulty number 3") else it }

    flow.collect { value -> println(value) }
}
```

Не поймает

Рефлексия

Маршрут вебинара

- Реактивное программирование и Flow
- Получение данных и контекст
- Преобразования и объединения
- SharedFlow и StateFlow
- Обработка ошибок

Kotlin Flow

Flow

Вопросы для проверки

По пройденному материалу всего вебинара

1. Что такое реактивное программирование?
2. Как мы создаем реактивные потоки в Kotlin?
3. Как подписаться на поток и получить данные?
4. Когда поток начинает выполняться?
5. Чем отличается SharedFlow от StateFlow?

Рефлексия

С какими впечатлениями уходите с вебинара?

Как будете применять на практике то,
что узнали на вебинаре?

Заполните, пожалуйста, опрос о занятии по ссылке в чате

Спасибо за внимание!

Приходите на следующие вебинары

Николай Кочетков

Руководитель Андроид разработки FlyXO

Об опыте (например):

24 года в IT, 8 лет - играющий тренер Андроид

Телефон / эл. почта / соц. сети:
LinkedIn: https://www.linkedin.com/in/motorro/
GitHub: https://github.com/motorro/
Medium: https://medium.com/@motorro