

# Coroutines Update

## Seva Tolstopyyatov



# Coroutines debugging



# Coroutines debugging

```
21     private suspend fun processUserEvents() {  
22         while (someCondition) {  
23             -> val element: Int = channel.receive()  
24                 processElement(element)  
25         }  
26         workDone()  
27     }
```

# Coroutines debugging

```
21     private suspend fun processUserEvents() {  
22         while (someCondition) {  
23             ↩→     val element: Int = channel.receive()  
24             ●     processElement(element)  
25         }  
26         workDone()  
27     }
```

# Coroutines debugging

The screenshot shows an IDE's debugger interface. At the top, the debug target is `SomeApplication`. The `Debugger` tab is active, displaying a list of frames. The top frame is `"Test worker @cor...p "main": RUNNING`. Below it, the stack of coroutine frames is visible, with `processUserEvents:24, SomeApplication (kotlinx.coroutines)` selected. The `Variables` pane on the right shows the current state of variables: `this = {SomeApplication@2363}`, `$result = {Integer@2203} 1`, and `01 element = 1`.

Debug: `SomeApplication` × `:kotlinx-coroutin...` ×

Debugger Console

Frames Threads

✓ "Test worker @cor...p "main": RUNNING

processUserEvents:24, SomeApplication (*kotlinx.coroutines*)

invokeSuspend:-1, SomeApplication\$processUserEvents\$1

run:56, DispatchedTask (*kotlinx.coroutines*)

processNextEvent:274, EventLoopImplBase (*kotlinx.coroutines*)

joinBlocking:84, BlockingCoroutine (*kotlinx.coroutines*)

runBlocking:59, BuildersKt\_\_BuildersKt (*kotlinx.coroutines*)

runBlocking\$default:38, BuildersKt\_\_BuildersKt (*kotlinx.coroutines*)

runBlocking\$default:1, BuildersKt\_\_BuildersKt (*kotlinx.coroutines*)

Variables

this = {SomeApplication@2363}

\$result = {Integer@2203} 1

01 element = 1

# Coroutines debugging – IDEA 2020.1

Debug: SomeApplication x :kotlinx-coroutin... x

Debugger Console

Frames Threads Variables

✓ "Test worker @cor...p "main": RUNNING

processUserEvents:24, SomeApplication (*kotlinx.coroutines*)

startProcessing:35, SomeApplication (*kotlinx.coroutines*)

initiateUserSession:30, SomeApplication (*kotlinx.coroutines*)

invokeSuspend:51, SomeApplication\$testDebuggingExperie

run:56, DispatchedTask (*kotlinx.coroutines*)

processNextEvent:274, EventLoopImplBase (*kotlinx.coroutines*)

joinBlocking:84, BlockingCoroutine (*kotlinx.coroutines*)

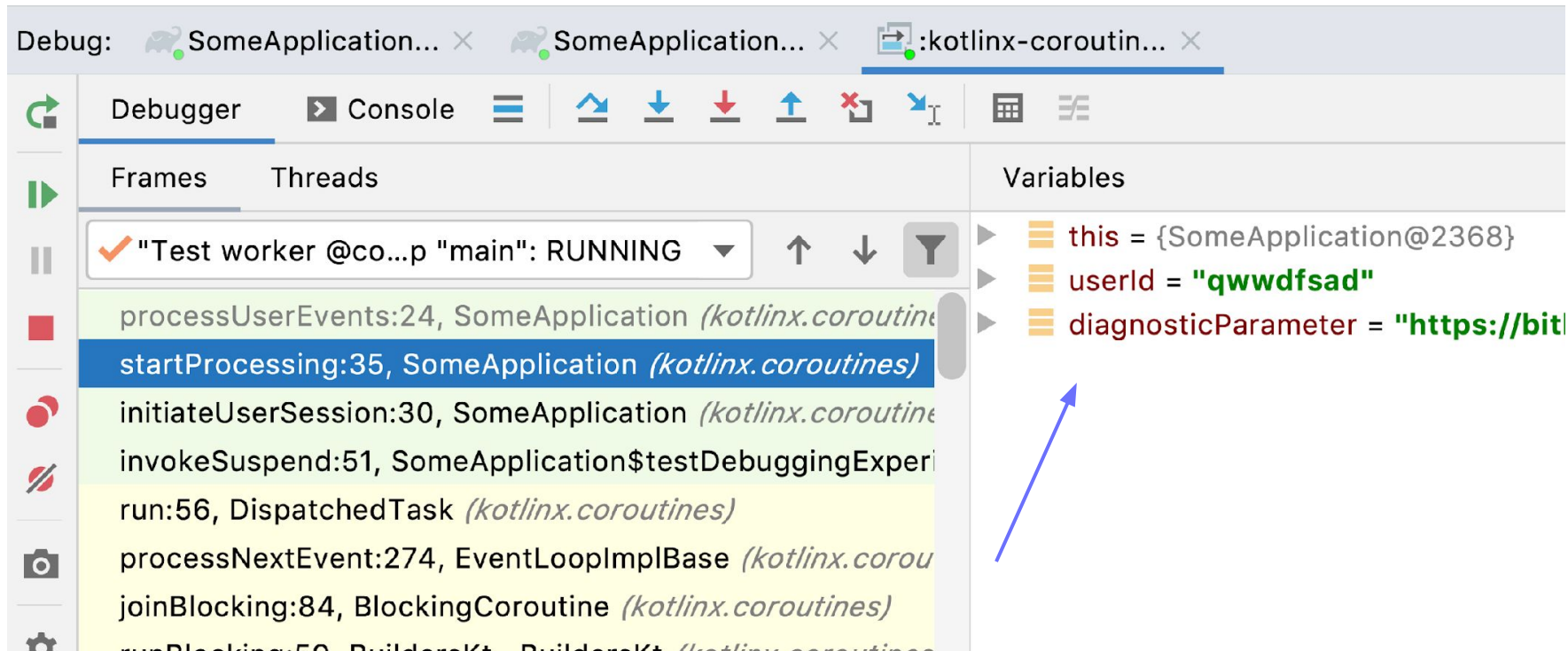
runBlocking:50, BuildersKt BuildersKt (*kotlinx.coroutines*)

this = {SomeApplication@2363}

\$result = {Integer@2203} 1

01 element = 1

# Coroutines debugging



The screenshot shows the debug console of an IDE with three tabs: "SomeApplication...", "SomeApplication...", and ":kotlinx-coroutin...". The "Debugger" tab is active, displaying a list of frames on the left and variables on the right.

**Debugger Frames:**

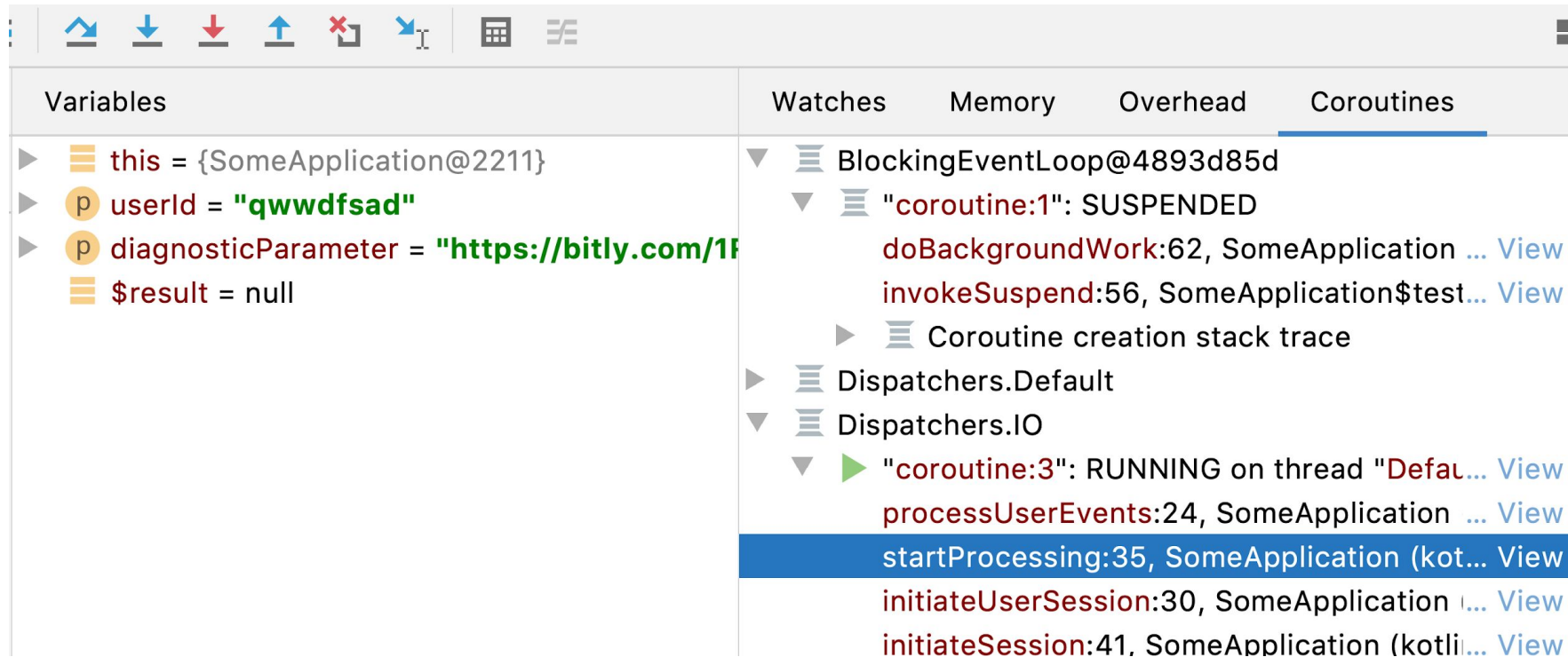
- Frames: Threads
- Selected frame: "Test worker @co...p "main": RUNNING" (indicated by a checkmark icon)
- Frame list (from top to bottom):
  - processUserEvents:24, SomeApplication (kotlinx.coroutines)
  - startProcessing:35, SomeApplication (kotlinx.coroutines)** (highlighted in blue)
  - initiateUserSession:30, SomeApplication (kotlinx.coroutines)
  - invokeSuspend:51, SomeApplication\$testDebuggingExperi
  - run:56, DispatchedTask (kotlinx.coroutines)
  - processNextEvent:274, EventLoopImplBase (kotlinx.corou
  - joinBlocking:84, BlockingCoroutine (kotlinx.coroutines)
  - runBlocking:50, BuildersKt BuildersKt (kotlinx.coroutines)

**Variables:**

- this = {SomeApplication@2368}
- userId = "qwdfsad"
- diagnosticParameter = "https://bit"

A blue arrow points from the "diagnosticParameter" variable to the "run:56, DispatchedTask (kotlinx.coroutines)" frame in the list.

# Debugging meets coroutines 1.3.8+



The screenshot shows the debug console of an IDE. The top toolbar contains icons for navigating between variables, watches, memory, overhead, and coroutines. The 'Coroutines' tab is selected. The 'Variables' pane on the left shows the current state of the application: `this` is a `SomeApplication@2211`, `userId` is `"qwdfsad"`, `diagnosticParameter` is `"https://bitly.com/1f"`, and `$result` is `null`. The 'Coroutines' pane on the right shows a list of coroutines. The coroutine `"coroutine:3"` is currently `RUNNING` on thread `"DefaultDispatcher"`. The stack trace for this coroutine is visible, showing the following steps: `processUserEvents:24`, `startProcessing:35`, `initiateUserSession:30`, and `initiateSession:41`. The `startProcessing:35` step is currently selected and highlighted in blue.




Variables	Watches	Memory	Overhead	Coroutines
<ul style="list-style-type: none"><li><code>this</code> = {SomeApplication@2211}</li><li><code>userId</code> = "qwdfsad"</li><li><code>diagnosticParameter</code> = "https://bitly.com/1f"</li><li><code>\$result</code> = null</li></ul>	<ul style="list-style-type: none"><li>BlockingEventLoop@4893d85d<ul style="list-style-type: none"><li>"coroutine:1": SUSPENDED<ul style="list-style-type: none"><li>doBackgroundWork:62, SomeApplication ... <a href="#">View</a></li><li>invokeSuspend:56, SomeApplication\$test... <a href="#">View</a></li></ul></li><li>Coroutine creation stack trace</li></ul></li><li>Dispatchers.Default</li><li>Dispatchers.IO<ul style="list-style-type: none"><li>"coroutine:3": RUNNING on thread "DefaultDispatcher"<ul style="list-style-type: none"><li>processUserEvents:24, SomeApplication ... <a href="#">View</a></li><li>startProcessing:35, SomeApplication (kot... <a href="#">View</a></li><li>initiateUserSession:30, SomeApplication ... <a href="#">View</a></li><li>initiateSession:41, SomeApplication (kotli... <a href="#">View</a></li></ul></li></ul></li></ul>			



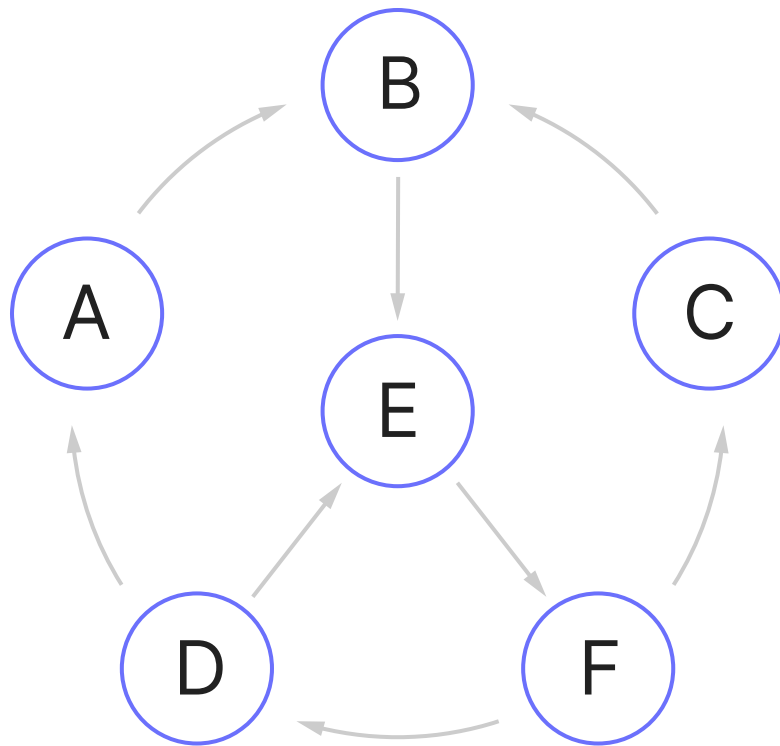
Flow since 1.3



# Flow

```
 val flow: Flow<Int> = flow {  
    delay(100)  
    for (i in 1..10) {  
        emit(i)  
    }  
}.map {  
    delay(100)  
    it * it  
}
```

# StateFlow

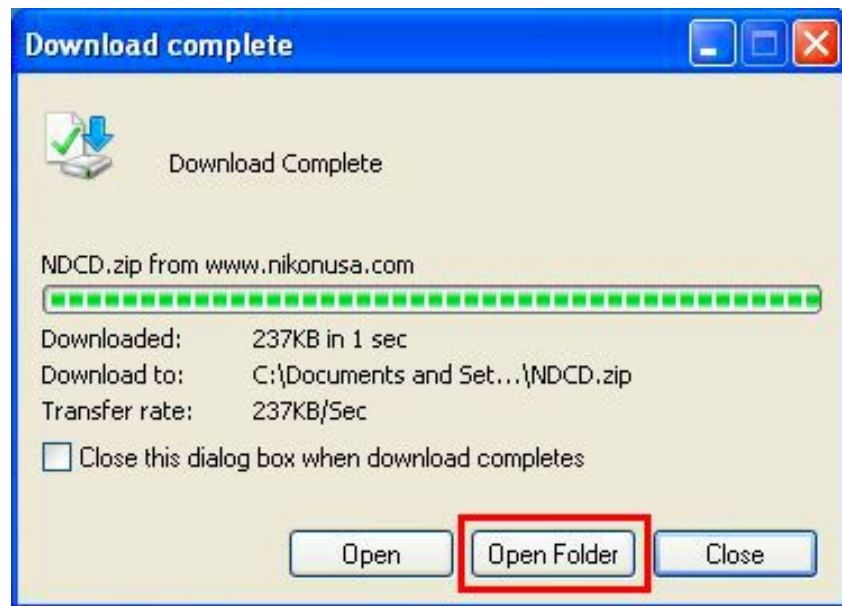


# State

A condition or way of being  
that exists at a particular time

```
var variable: Int = 42
```

# StateFlow



# StateFlow

[kotlinx-coroutines-core](#) / [kotlinx.coroutines.channels](#) / [ConflatedBroadcastChannel](#)

## ConflatedBroadcastChannel

```
@ExperimentalCoroutinesApi class ConflatedBroadcastChannel<E> :  
    BroadcastChannel<E> (source)
```

Broadcasts the most recently sent element (aka [value](#)) to all [openSubscription](#) subscribers.

Back-to-send sent elements are *conflated* – only the the most recently sent value is received, while previously sent elements **are lost**. Every subscriber immediately receives the most recently sent element. Sender to this broadcast channel never suspends and [offer](#) always returns `true`.

A secondary constructor can be used to create an instance of this class that already holds a value. This channel is also created by `BroadcastChannel(Channel.CONFLATED)` factory function invocation.

This implementation is fully lock-free. In this implementation [opening](#) and [closing](#) subscription takes O(N) time, where N is the number of subscribers.

**Note: This API is experimental.** It may be changed in the future updates.

# StateFlow

[kotlinx-coroutines-core](#) / [kotlinx.coroutines.channels](#) / [ConflatedBroadcastChannel](#)

## ConflatedBroadcastChannel

```
@ExperimentalCoroutinesApi class ConflatedBroadcastChannel<E> :  
    BroadcastChannel<E> (source)
```

Broadcasts the most recently sent element (aka [value](#)) to all [openSubscription](#) subscribers.

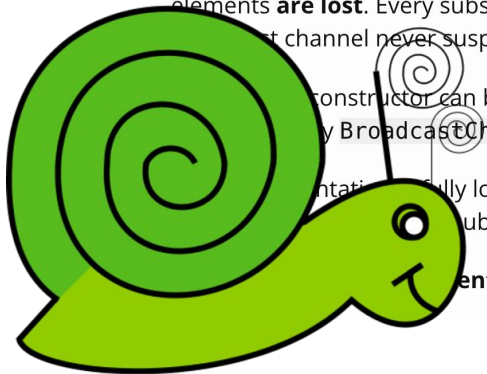
Back-to-send sent elements are *conflated* – only the the most recently sent value is received, while previously sent elements **are lost**. Every subscriber immediately receives the most recently sent element. Sender to this

channel never suspends and [offer](#) always returns `true`.

Constructor can be used to create an instance of this class that already holds a value. This channel is created by `BroadcastChannel(Channel.CONFLATED)` factory function invocation.

Implementation is fully lock-free. In this implementation [opening](#) and [closing](#) subscription takes  $O(N)$  time, where  $N$  is the number of subscribers.

**Experimental.** It may be changed in the future updates.





# StateFlow

```
public interface StateFlow<out T> : Flow<T> {  
    public val value: T  
}
```



# StateFlow

```
public interface MutableStateFlow<T> : Flow<T> {  
    public override var value: T  
}
```

# StateFlow

```
class DownloadingModel {  
  
    private val _status = MutableStateFlow<DownloadStatus>(DownloadStatus.NOT_REQUESTED)  
    val status: StateFlow<DownloadStatus> get() = _status  
  
    suspend fun download() {  
  
    }  
}
```

# StateFlow

```
class DownloadingModel {  
  
    private val _status = MutableStateFlow<DownloadStatus>(DownloadStatus.NOT_REQUESTED)  
    val status: StateFlow<DownloadStatus> get() = _status  
  
    suspend fun download() {  
        _status.value = DownloadStatus.INITIALIZED  
        initializeConnection()  
    }  
}
```

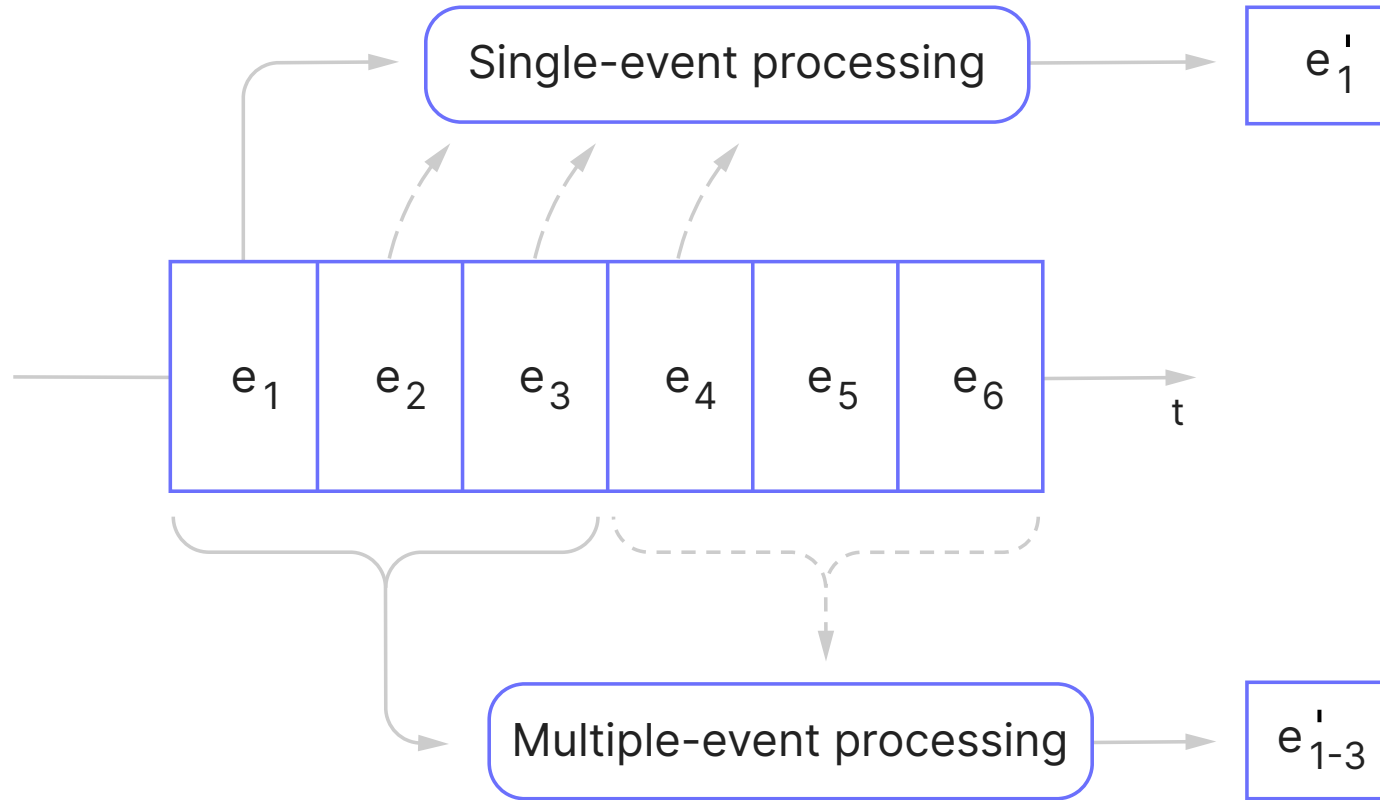
# StateFlow

```
class DownloadingModel {  
  
    private val _status = MutableStateFlow<DownloadStatus>(DownloadStatus.NOT_REQUESTED)  
    val status: StateFlow<DownloadStatus> get() = _status  
  
    suspend fun download() {  
        _status.value = DownloadStatus.INITIALIZED  
        initializeConnection()  
        processAvailableContent { partialData: ByteArray,  
                                downloadedBytes: Long,  
                                totalBytes: Long ->  
            storePartialData(partialData)  
            _status.value = DownloadProgress(downloadedBytes.toDouble() / totalBytes)  
        }  
    }  
}
```

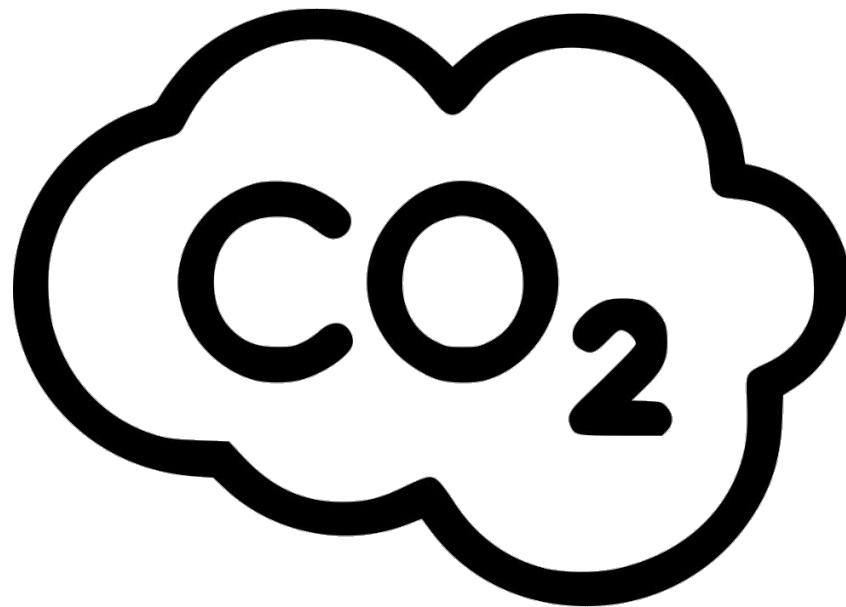
# StateFlow

```
class DownloadingModel {  
  
    private val _status = MutableStateFlow<DownloadStatus>(DownloadStatus.NOT_REQUESTED)  
    val status: StateFlow<DownloadStatus> get() = _status  
  
    suspend fun download() {  
        _status.value = DownloadStatus.INITIALIZED  
        initializeConnection()  
        processAvailableContent { partialData: ByteArray,  
                                downloadedBytes: Long,  
                                totalBytes: Long ->  
            storePartialData(partialData)  
            _status.value = DownloadProgress(downloadedBytes.toDouble() / totalBytes)  
        }  
        _status.value = DownloadStatus.SUCCESS  
    }  
}
```

# SharedFlow



# SharedFlow



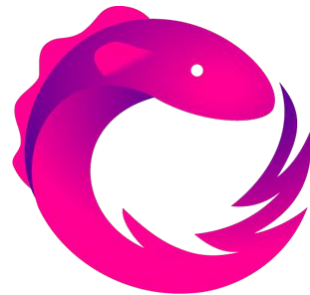
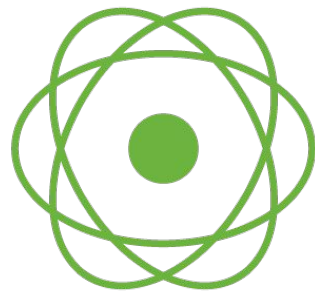
# SharedFlow

- Costly connections
- May be unused
- Replay log
- Flexibility



# Existing solutions

- Subjects: BehaviorSubject, AsyncSubject, ReplaySubject
- ConnectableFlowable: connect, refCount, autoConnect
- Processors: Emitter, Unicast
- Share, publish, replay



# SharedFlow

```
interface SharedFlow<out T> : Flow<T> {  
    public val replayCache: List<T>  
}
```

# SharedFlow

```
interface MutableSharedFlow<T> : SharedFlow<T>, FlowCollector<T> {  
    suspend fun emit(value: T)  
    fun tryEmit(value: T): Boolean  
    val subscriptionCount: StateFlow<Int>  
    fun resetReplayCache()  
}
```

# SharedFlow

```
public fun <T> MutableSharedFlow(  
    replay: Int,  
    extraBufferCapacity: Int = 0,  
    onBufferOverflow: BufferOverflow = BufferOverflow.SUSPEND  
) : MutableSharedFlow<T>
```

# SharedFlow

```
public fun <T> Flow<T>.shareIn(  
    scope: CoroutineScope,  
    replay: Int,  
    started: SharingStarted = SharingStarted.Eagerly  
)
```

# Flow since 1.3

- Core operators
  - `catch`, `onEmpty`, `onCompletion`, `onStart`
  - `onEach`, `transform`, `transformWhile`
- Invariants

# Flow since 1.3

```
suspend fun Flow<Int>.stopOn42() = collect {  
    println(it)  
    if (it == 42) {  
        throw AnswerFoundException()  
    }  
}
```

# Flow since 1.3

```
flow {  
    try {  
        emit(42)  
    } catch (e: AnswerFoundException) {  
        emit(21)  
    }  
}.stopOn42()
```



# Flow since 1.3

java.lang.IllegalStateException: Flow exception transparency is violated:

Previous 'emit' call has thrown exception

java.util.concurrent.CancellationException: Thanks, I had enough of your data, but then emission attempt of value '21' has been detected.

Emissions from 'catch' blocks are prohibited in order to avoid unspecified behaviour, **'Flow.catch' operator can be used instead.**

For a more detailed explanation, please refer to Flow documentation.

at

kotlinx.coroutines.flow.internal.SafeCollector.exceptionTransparencyViolated(SafeCollector.kt:114)

# Flow since 1.3

```
flowOf(42)  
  .catch { e -> println("Answer was found") }  
  .stopOn42()
```

# Android update



# Android update

- The coroutines DEX size is optimized by 30%
- Startup time was significantly optimised
- CPU consumption of default dispatchers was drastically reduced

[1] [github.com/Kotlin/kotlinx.coroutines/pull/1652](https://github.com/Kotlin/kotlinx.coroutines/pull/1652)

# JDK update



+



# JDK update – Blocking calls

```
withTimeout(500.milliseconds) {  
    runInterruptible(Dispatchers.IO) {  
        serverSocket.accept()  
    }  
}
```

# More JDK updates

- Out-of-the-box integration with [BlockHound](#)
- Integration with JDK 9 [java.util.concurrent.Flow](#)

[1] [github.com/reactor/BlockHound](https://github.com/reactor/BlockHound)

[2] [docs.oracle.com/javase/9/docs/api/java/util/concurrent/Flow.html](https://docs.oracle.com/javase/9/docs/api/java/util/concurrent/Flow.html)

# The future of coroutines

- SharedFlow and StateFlow stabilization
- [Concise and cancellation-aware resource management](#)
- Replacement for offer and poll
- More Flow time API for UI programming
- kotlinx-coroutines-test stabilization
- Sliceable dispatchers

[1] [github.com/Kotlin/kotlinx.coroutines/pull/1937](https://github.com/Kotlin/kotlinx.coroutines/pull/1937)



Thanks!  
Have a nice Kotlin!



@qwdfsad