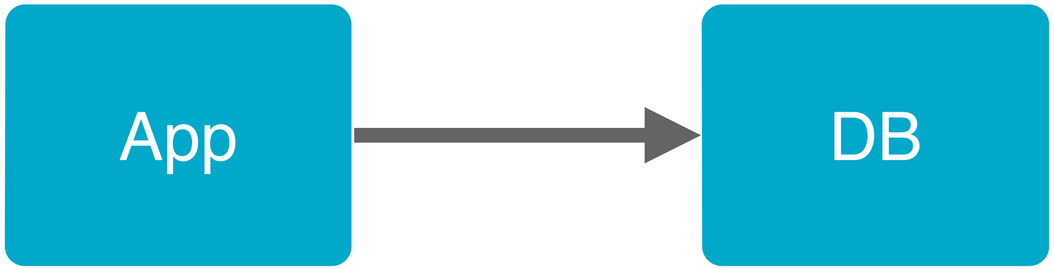
Reactive Streams

Алексей Романчук

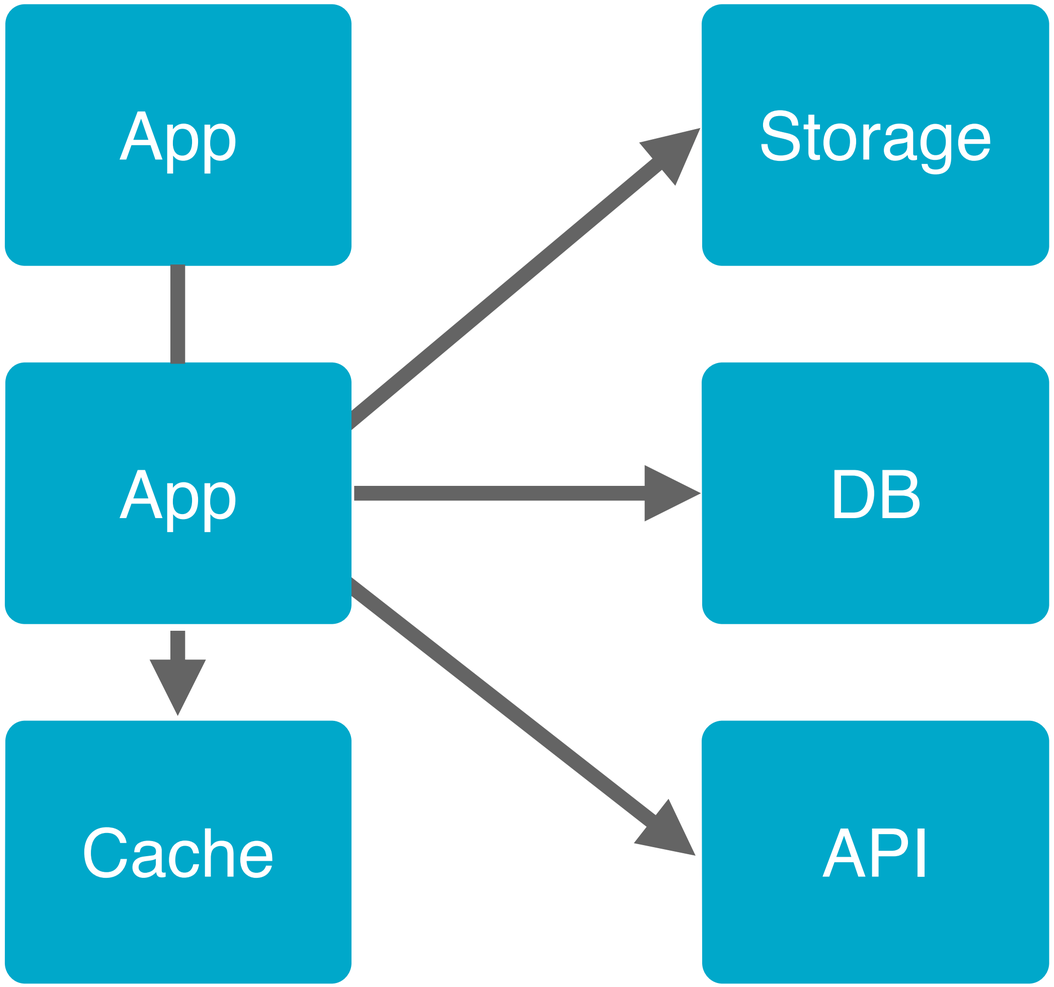
Обо мне

10k msg/s

Эволюция backendов



Эволюция backendов



60мс

60мс

60мс

- парсинг запроса - 1мс

60мс

- парсинг запроса - 1мс
- проверка в кеше - 2мс

60мс

- парсинг запроса - 1мс
- проверка в кеше - 2мс
- запрос в БД - 20мс

60мс

- парсинг запроса - 1мс
- проверка в кеше - 2мс
- запрос в БД - 20мс
- запрос в АПИ - 35мс

60мс

- парсинг запроса - 1мс
- проверка в кеше - 2мс
- запрос в БД - 20мс
- запрос в АПИ - 35мс
- формирование ответа - 2мс

Синхронная модель

Синхронная модель

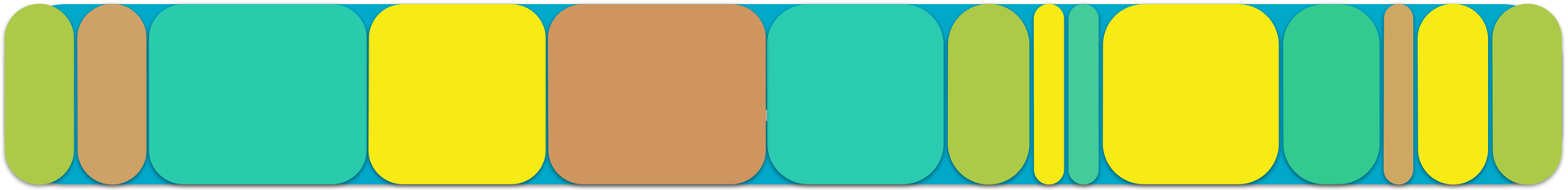
- Отдельный поток

Синхронная модель

- Отдельный поток
- 95% ожидает

Синхронная модель

- Отдельный поток
- 95% ожидает
- Занимает ресурсы



Асинхронная модель

Асинхронная модель

- Обработка в разных потоках

Асинхронная модель

- Обработка в разных потоках
- Потоки не простаивают

Асинхронная модель

- Обработка в разных потоках
- Потоки не простаивают
- Эффективное использование ресурсов

Асинхронные границы везде

Асинхронные границы везде

- БД

Асинхронные границы везде

- БД
- Внешние API

Асинхронные границы везде

- БД
- Внешние API
- Сеть

Асинхронные границы везде

- БД
- Внешние API
- Сеть
- Взаимодействие с другими потокам

Асинхронность это сложно

Асинхронность это сложно

- Композиция

Асинхронность это сложно

- Композиция
- Ветвление

Асинхронность это сложно

- Композиция
- Ветвление
- Обработка ошибок

Асинхронность это сложно

- Композиция
- Ветвление
- Обработка ошибок
- Backpressure

Асинхронность это сложно


```
fs.readdir(source, function(err, files) {
  if (err) {
    console.log('Error finding files: ' + err)
  } else {
    files.forEach(function(filename, fileIndex) {
      console.log(filename)
      gm(source + filename).size(function(err, values) {
        if (err) {
          console.log('Error identifying file size: ' + err)
        } else {
          console.log(filename + ' : ' + values)
          aspect = (values.width / values.height)
          widths.forEach(function(width, widthIndex) {
            height = Math.round(width / aspect)
            console.log('resizing ' + filename + 'to ' + height + 'x' + height)
            this.resize(width, height).write(destination + 'w' + width + '_' + filename, function(err)
              if (err) console.log('Error writing file: ' + err)
            })
          }.bind(this))
        }
      })
    })
  }
})
```

Асинхронность это сложно


```
request(httpUrl, new Callback() {
    @Override
    public void onResponse(Response response) throws IOException {
        request(response.body(), new Callback() {
            @Override
            public void onResponse(Response response) throws IOException {
                lastData1 = response.body();
                if (lastData1 != null && lastData2 != null) {
                    request(combineResult(lastData1, lastData1), new Callback() {
                        @Override
                        public void onResponse(Response response) throws IOException {
                            processResponse(response);
                        }
                    });
                }
            }
        });
    }
});
```

Модели

Модели

- Mutex, semaphore, etc

Модели

- Mutex, semaphore, etc
- Green threads

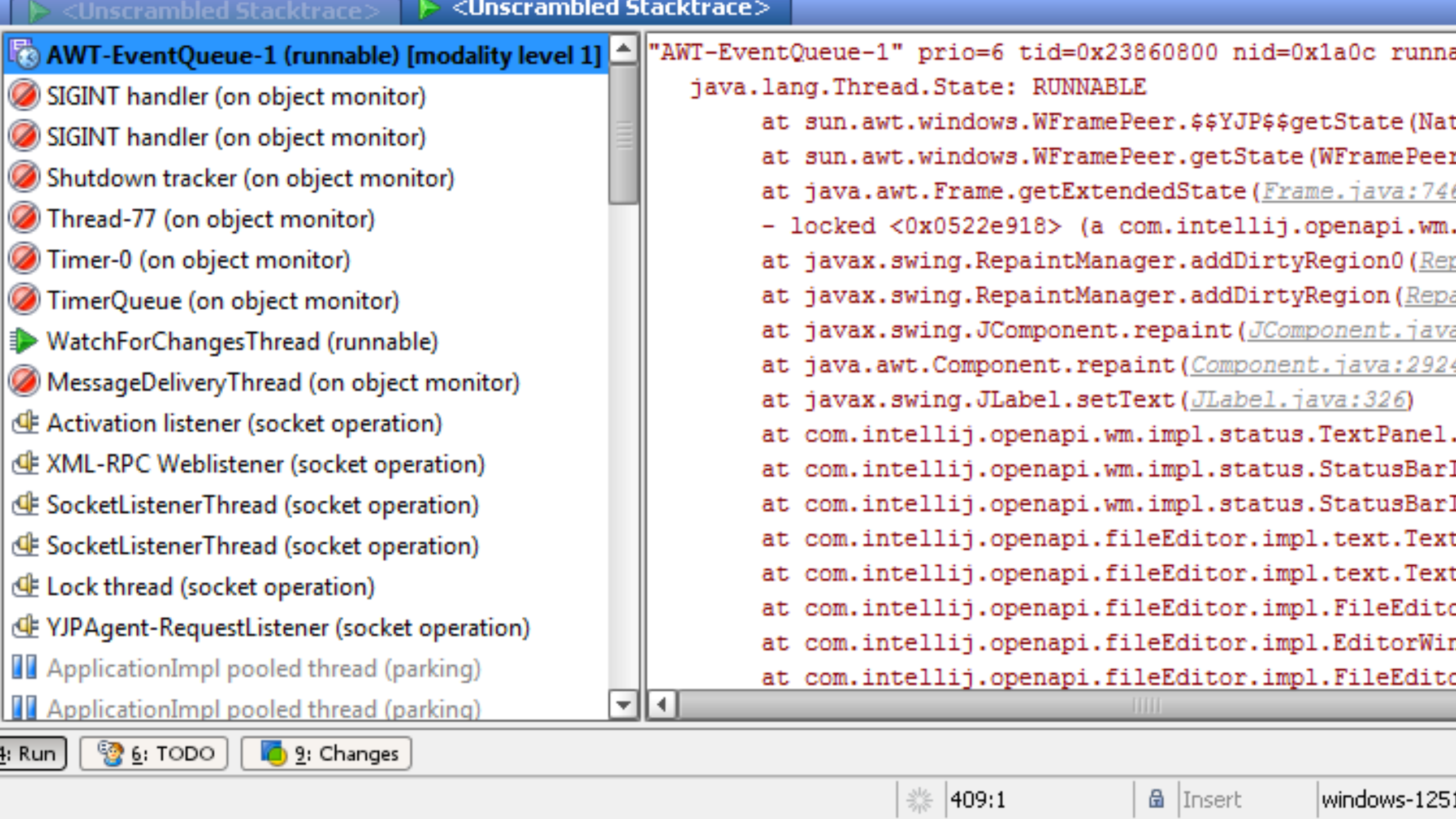
Модели

- Mutex, semaphore, etc
- Green threads
- Future-Promise

Модели

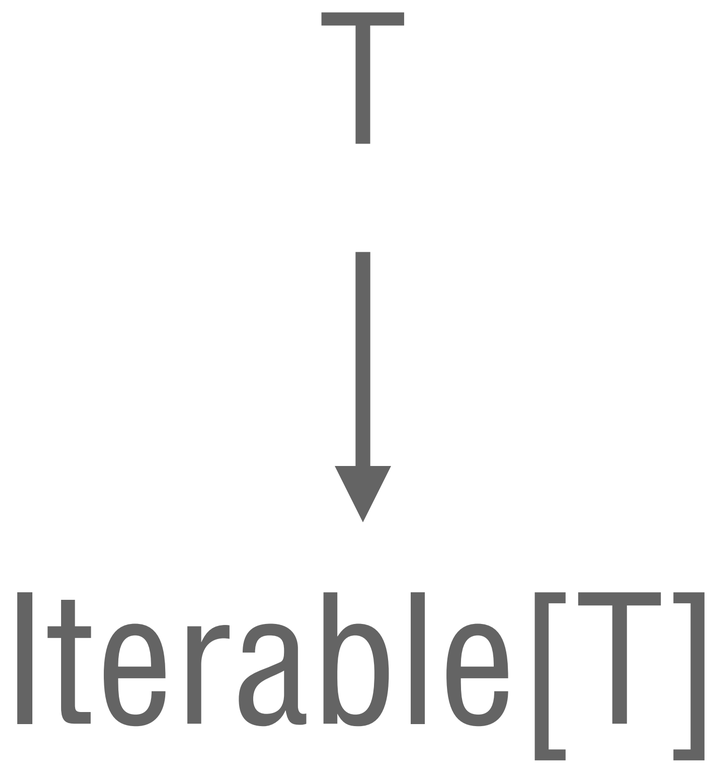
- Mutex, semaphore, etc
- Green threads
- Future-Promise
- Потоки данных

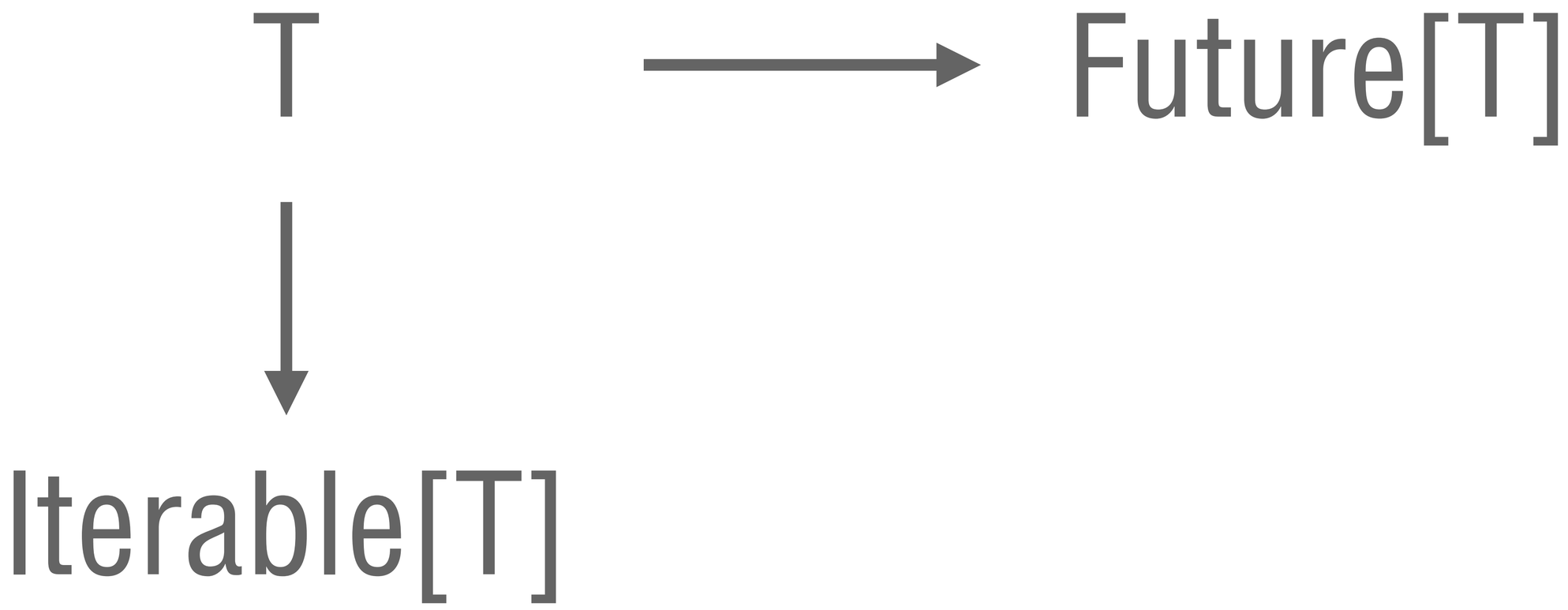
Потоки

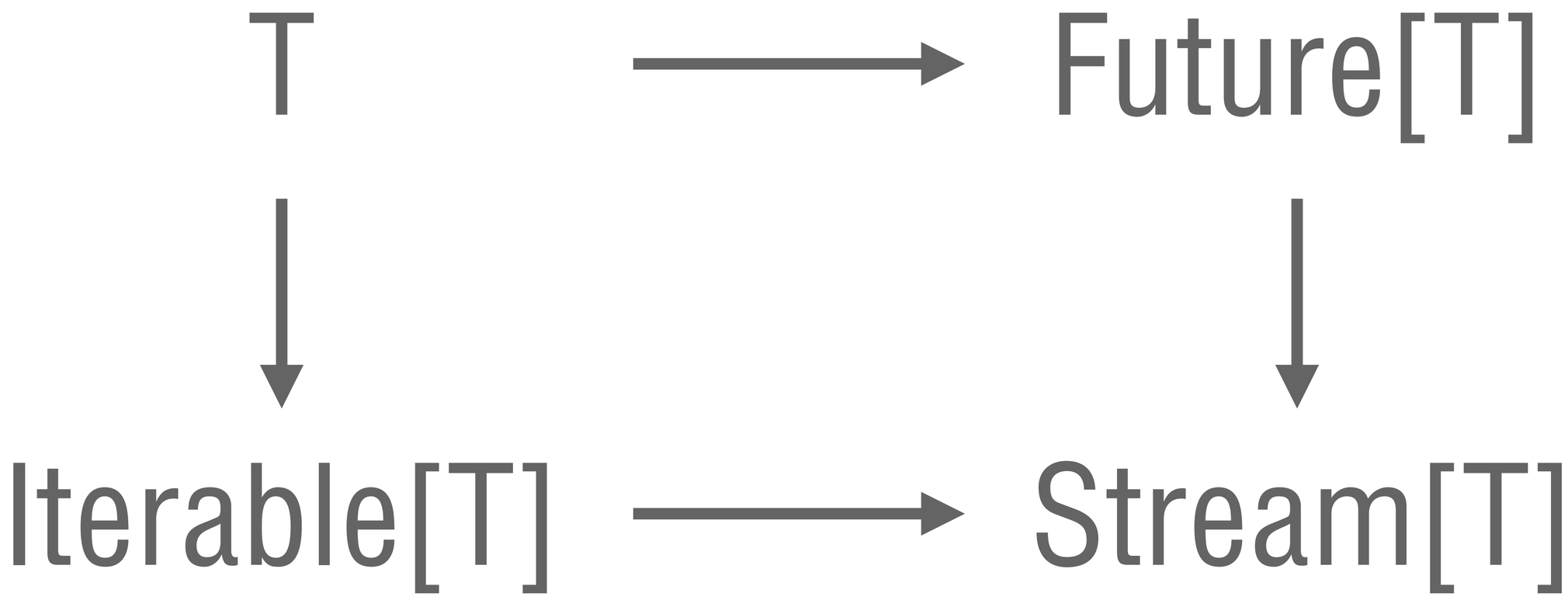

<Unscrambled Stacktrace>
<Unscrambled Stacktrace>
AWT-EventQueue-1 (runnable) [modality level 1]
SIGINT handler (on object monitor)
SIGINT handler (on object monitor)
Shutdown tracker (on object monitor)
Thread-77 (on object monitor)
Timer-0 (on object monitor)
TimerQueue (on object monitor)
WatchForChangesThread (runnable)
MessageDeliveryThread (on object monitor)
Activation listener (socket operation)
XML-RPC Weblistener (socket operation)
SocketListenerThread (socket operation)
SocketListenerThread (socket operation)
Lock thread (socket operation)
YJPAgent-RequestListener (socket operation)
ApplicationImpl pooled thread (parking)
ApplicationImpl pooled thread (parking)
"AWT-EventQueue-1" prio=6 tid=0x23860800 nid=0x1a0c runna
java.lang.Thread.State: RUNNABLE
at sun.awt.windows.WFramePeer.$$YJP$$getState(Nat
at sun.awt.windows.WFramePeer.getState(WFramePeer
at java.awt.Frame.getExtendedState(Frame.java:746
- locked <0x0522e918> (a com.intellij.openapi.wm.
at javax.swing.RepaintManager.addDirtyRegion0(Rep
at javax.swing.RepaintManager.addDirtyRegion(Repa
at javax.swing.JComponent.repaint(JComponent.java
at java.awt.Component.repaint(Component.java:2924
at javax.swing.JLabel.setText(JLabel.java:326)
at com.intellij.openapi.wm.impl.status.TextPanel.
at com.intellij.openapi.wm.impl.status.StatusBarI
at com.intellij.openapi.wm.impl.status.StatusBarI
at com.intellij.openapi.fileEditor.impl.text.Text
at com.intellij.openapi.fileEditor.impl.text.Text
at com.intellij.openapi.fileEditor.impl.FileEdito
at com.intellij.openapi.fileEditor.impl.EditorWin
at com.intellij.openapi.fileEditor.impl.FileEdito
4: Run
6: TODO
9: Changes
409:1
Insert
windows-1251

“В одну реку нельзя войти дважды”

T


T
Iterable[T]


T
Future[T]
Iterable[T]


T
Future[T]
Iterable[T]
Stream[T]

Потоки данных

Потоки данных

- Множество сообщений одного типа

Потоки данных

- Множество сообщений одного типа
- Зависит от времени начала наблюдения

Потоки данных

- Множество сообщений одного типа
- Зависит от времени начала наблюдения
- Может не иметь ни начала ни конца

Потоки вокруг нас

Потоки вокруг нас

- `curl twitter.com | grep devconf | wc -n`

Потоки вокруг нас

- `curl twitter.com | grep devconf | wc -n`
- сетевые соединения

Потоки вокруг нас

- `curl twitter.com | grep devconf | wc -n`
- сетевые соединения
- звук и видео

Потоки вокруг нас

- `curl twitter.com | grep devconf | wc -n`
- сетевые соединения
- звук и видео
- запросы и ответы сервера

Backend

Как нарисовать сову

1.

1. Рисуем кружочки

2.

2. Рисуем остаток совы

Backend

Backend

Backend

Backend

Backend

Backend

Backend

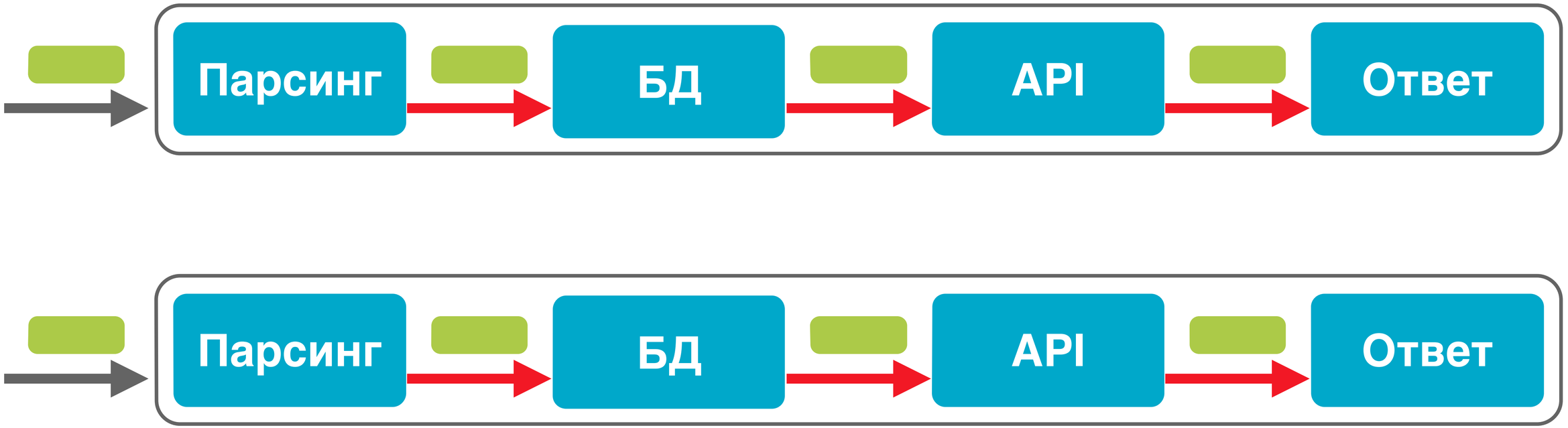
Backend

Backend

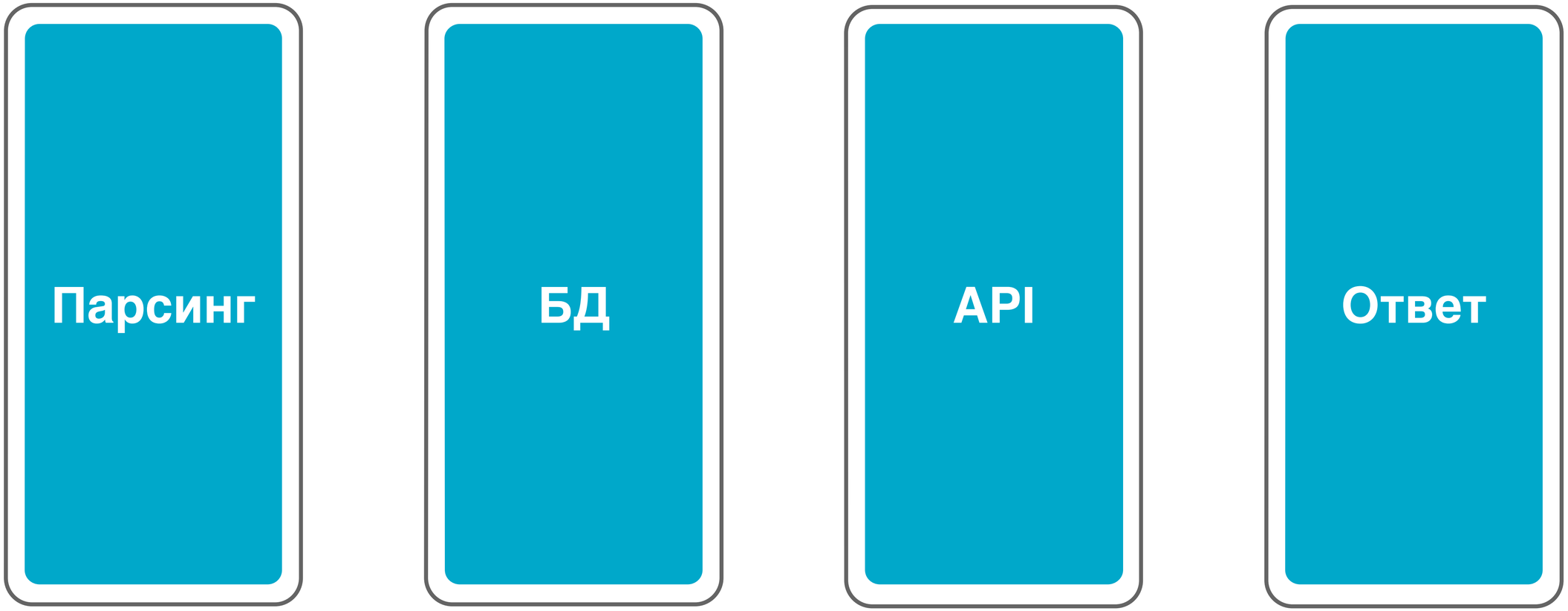
Backend

Backend

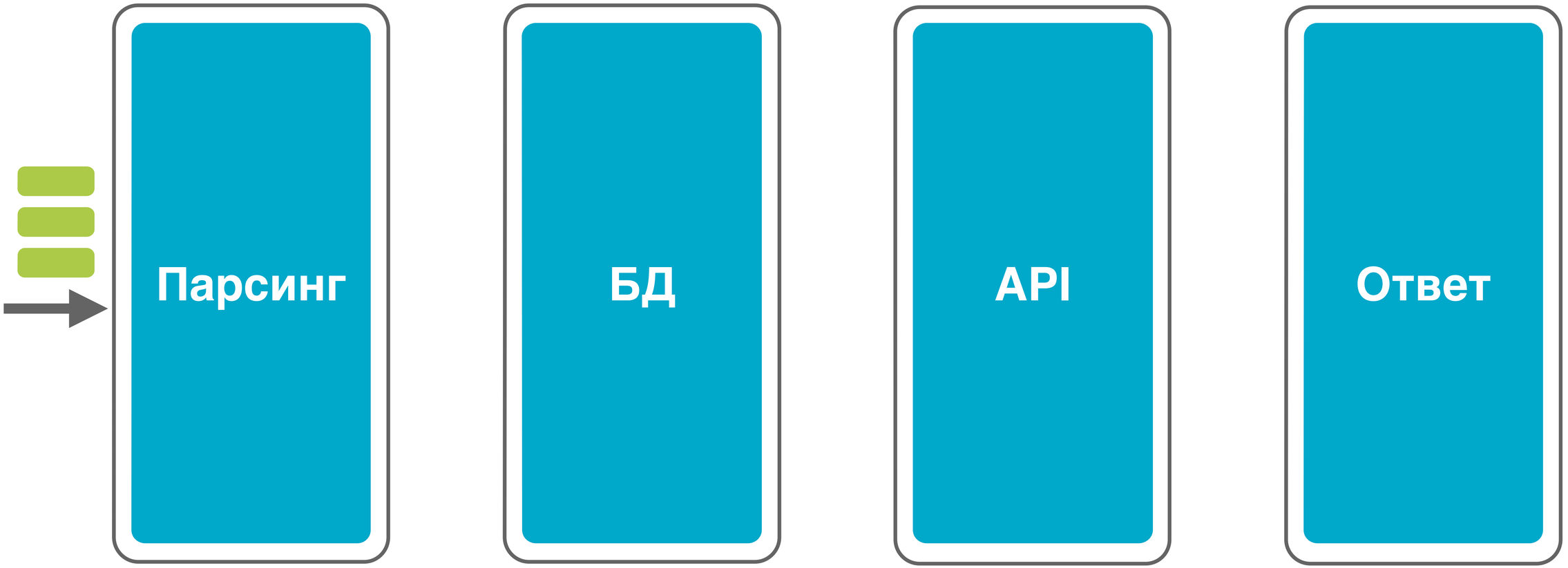
Backend



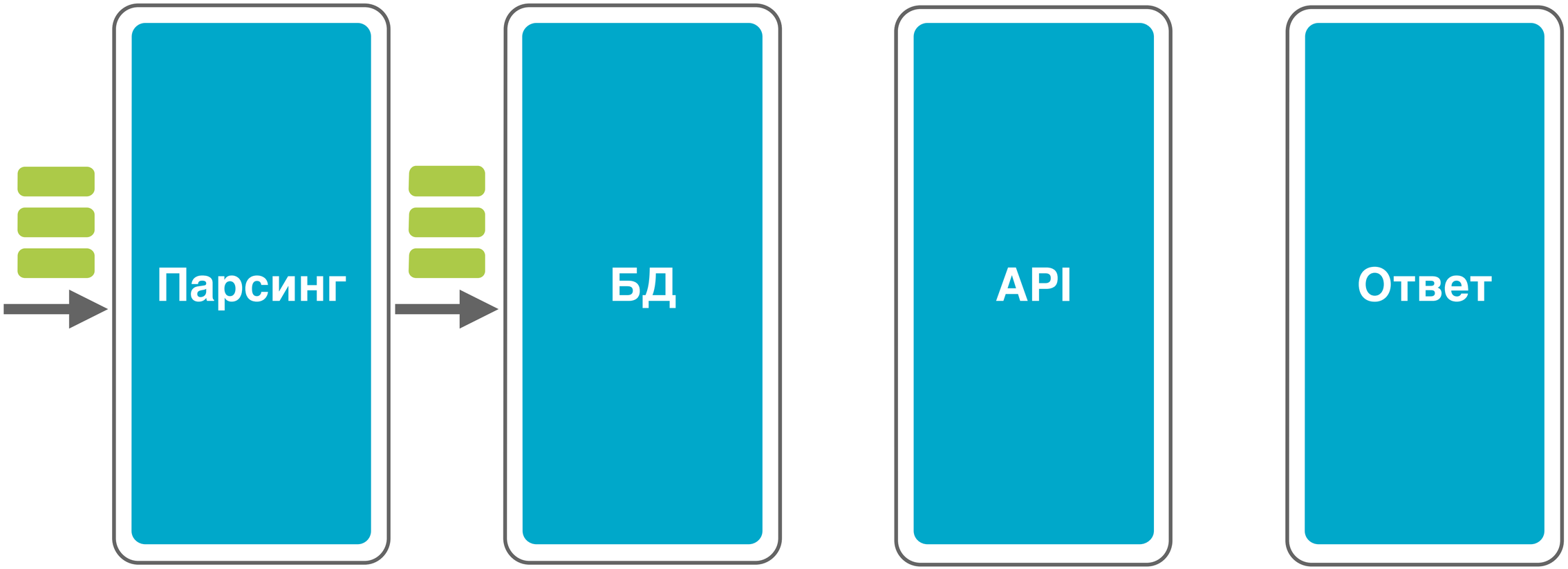
Потоки данных



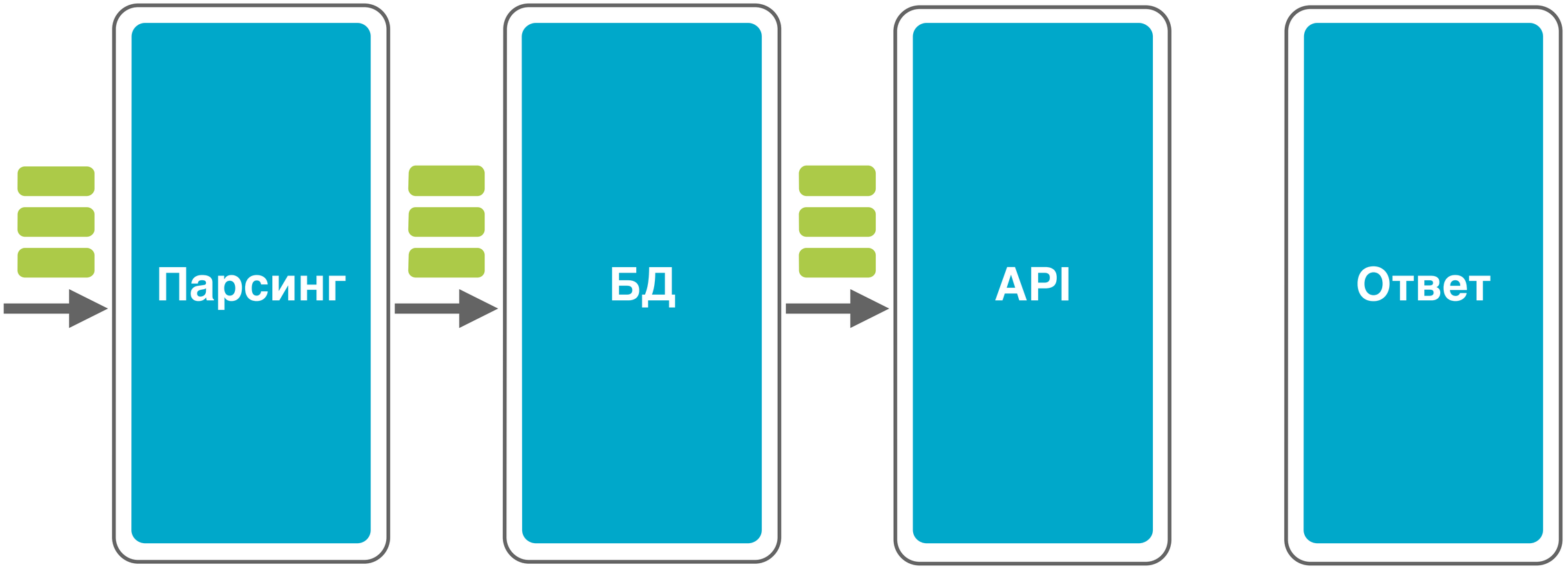
Потоки данных



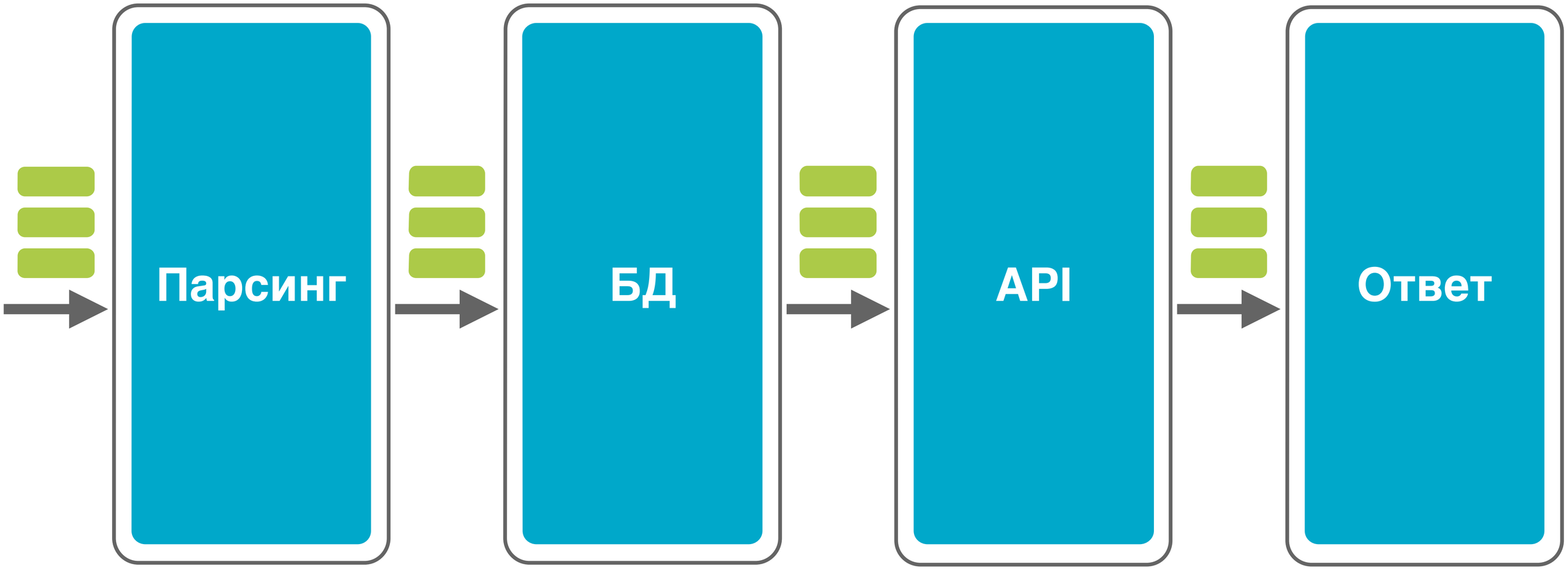
Потоки данных



Потоки данных



Потоки данных



Преимущества

Преимущества

- Простые асинхронные границы

Преимущества

- Простые асинхронные границы
- Картина в целом

Преимущества

- Простые асинхронные границы
- Картина в целом
- Параллельное программирование

Преимущества

- Простые асинхронные границы
- Картина в целом
- Параллельное программирование
- Ошибки и завершение

Backpressure

Событий слишком много

Событий слишком много

Отправитель

Событий слишком много

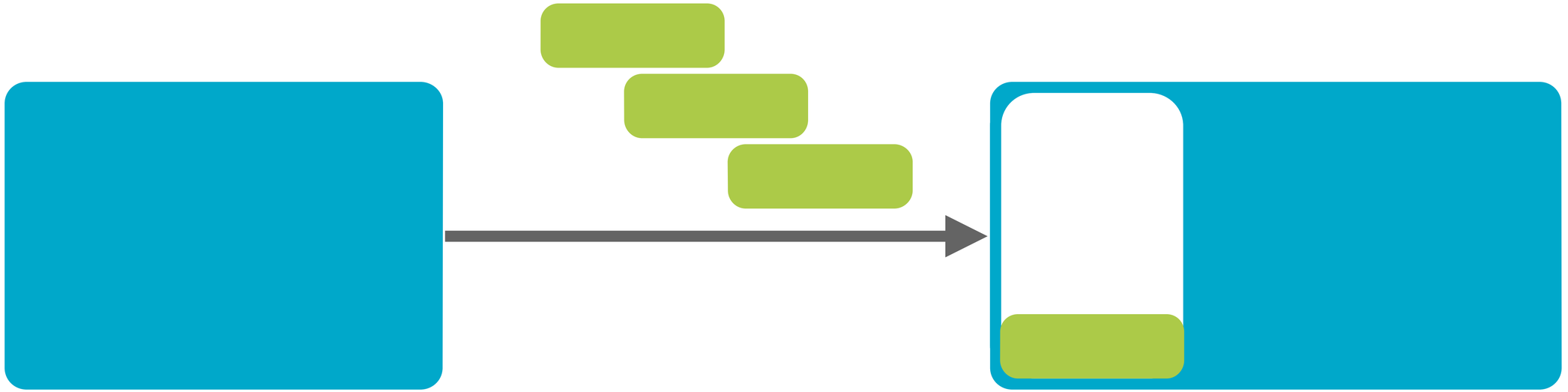
Отправитель

Получатель

Событий слишком много

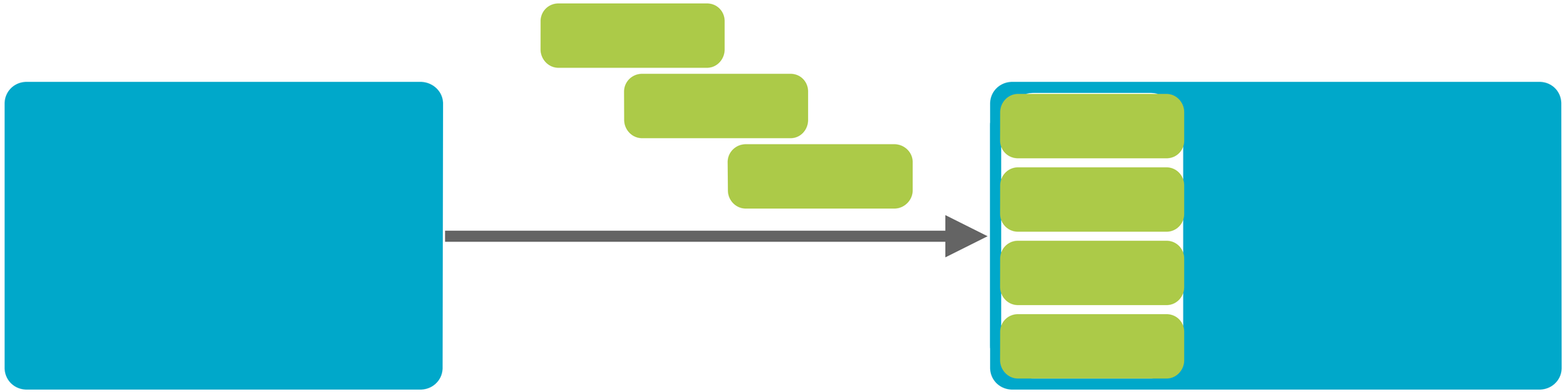
Событий слишком много

Событий слишком много

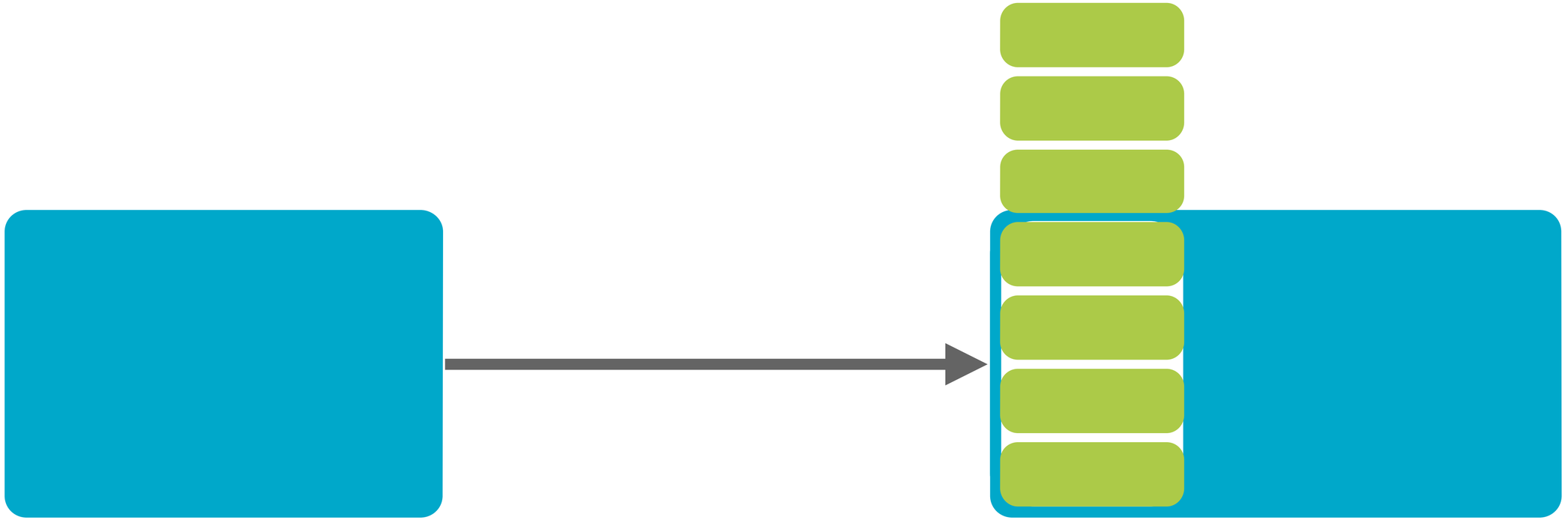


Событий слишком много

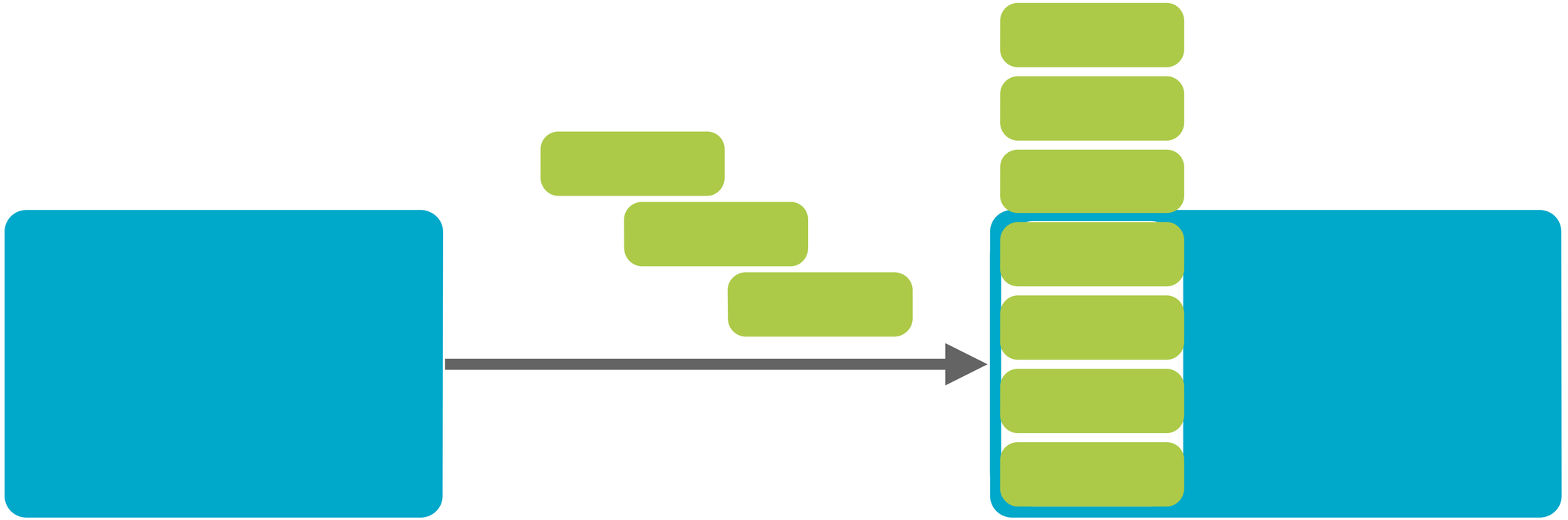
Событий слишком много



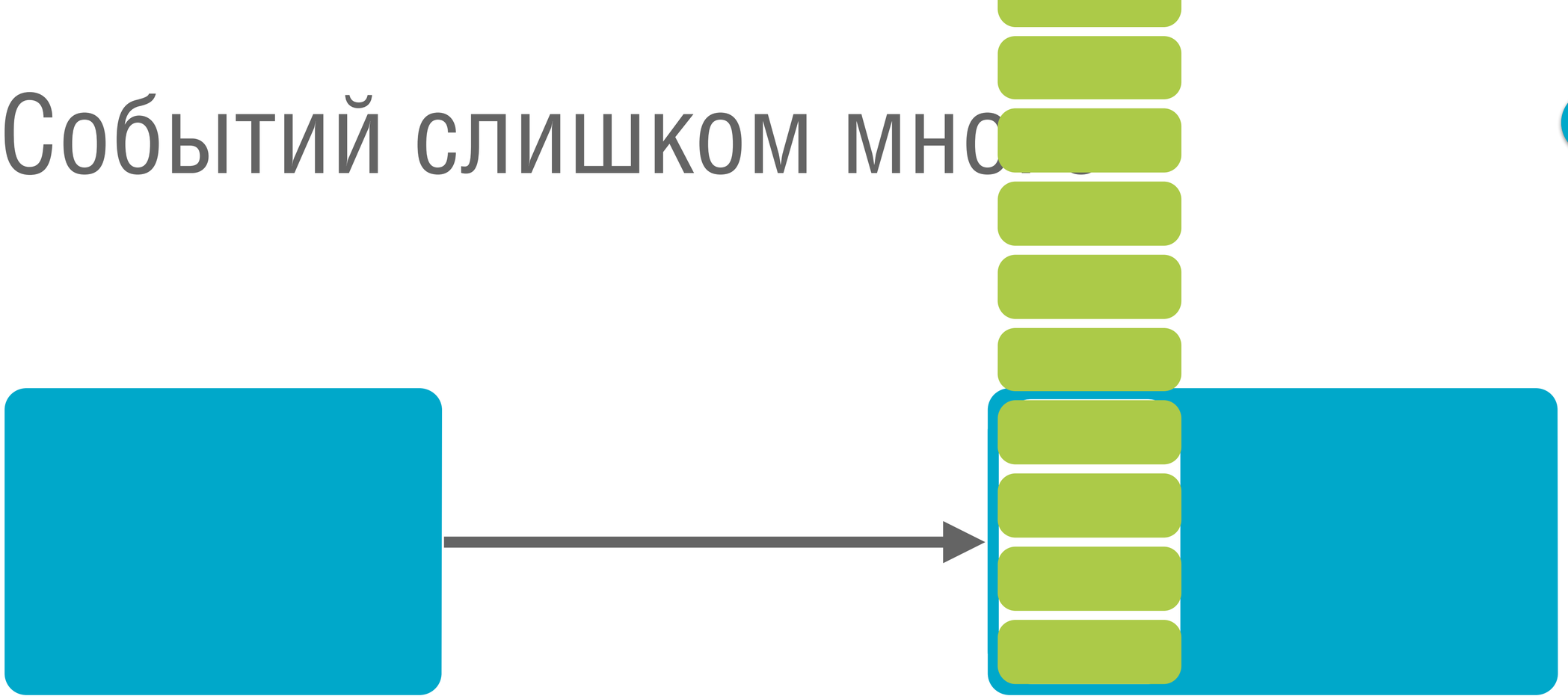
Событий слишком много



Событий слишком много



Событий слишком мно

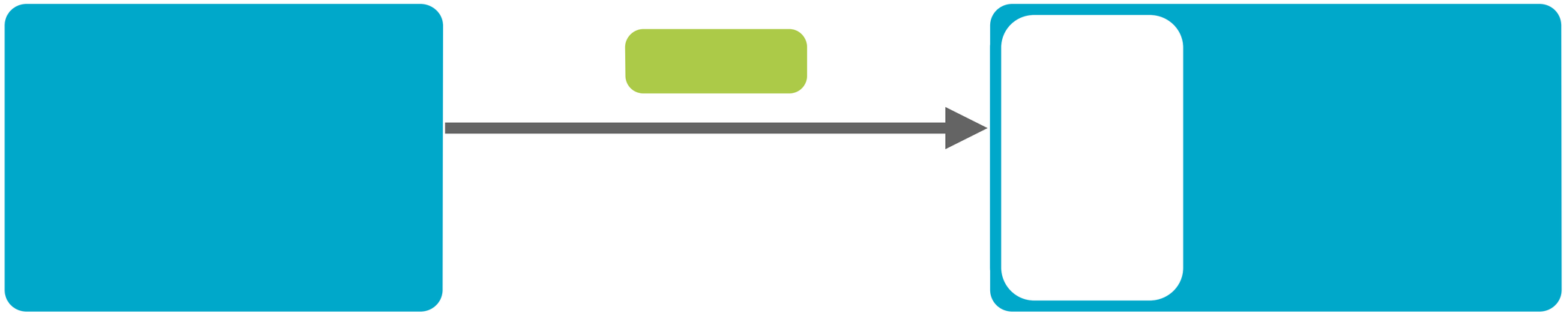


Блокирующий вызов

Отправитель

Получатель

Блокирующий вызов



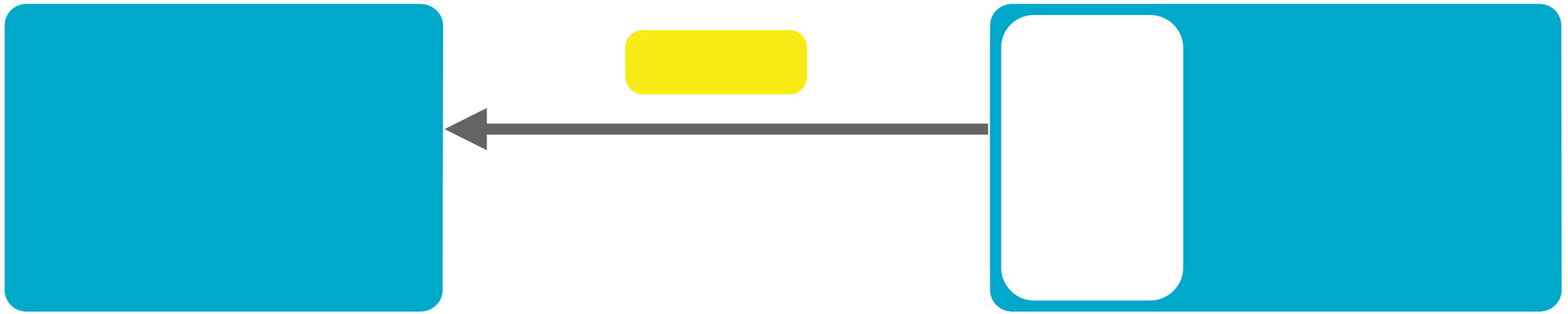
Блокирующий вызов

Pull

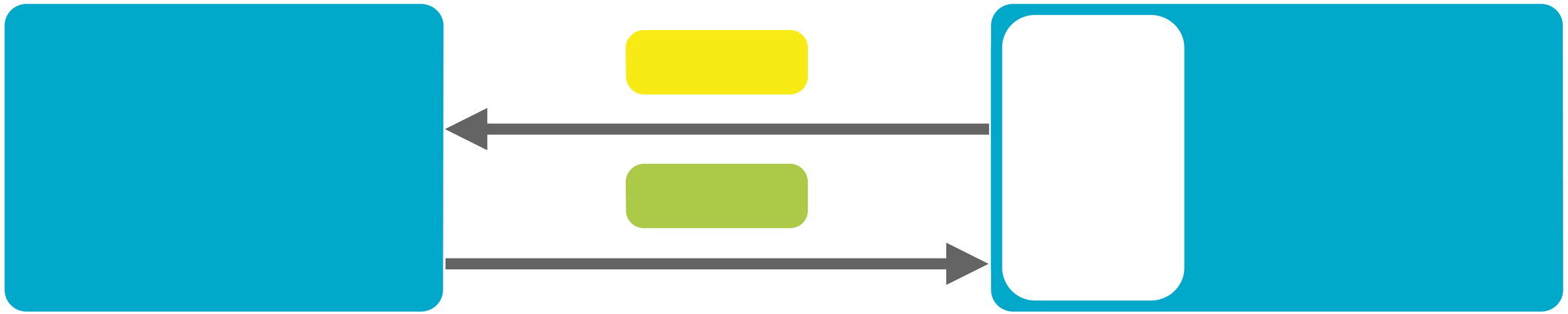
Отправитель

Получатель

Pull



Pull

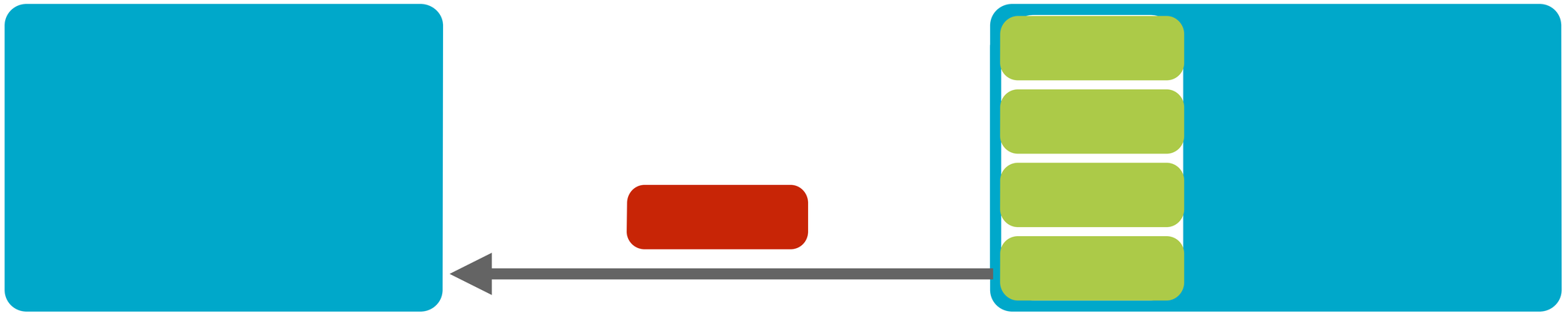


Negative Acknowledge

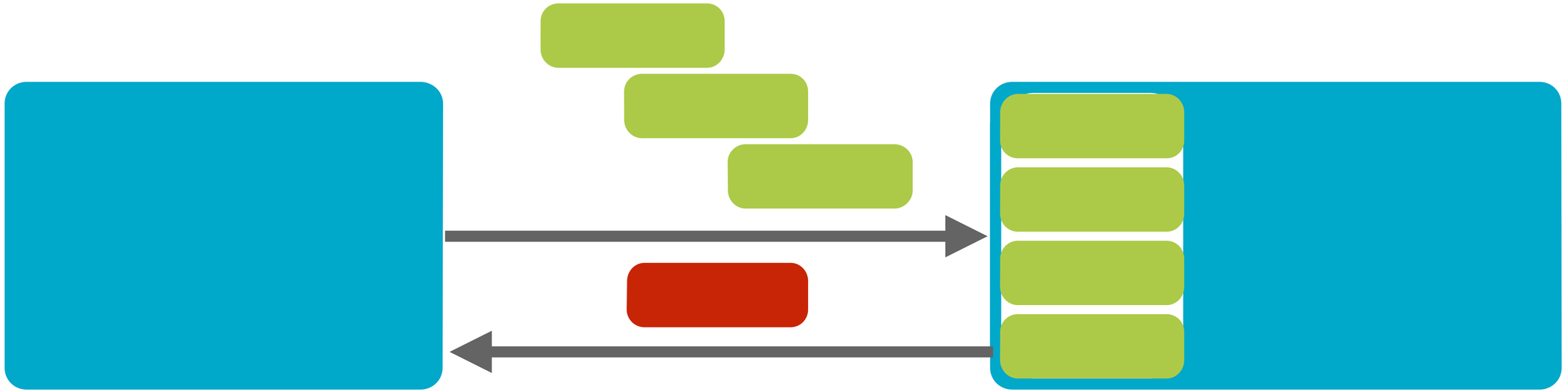
Отправитель

Получатель

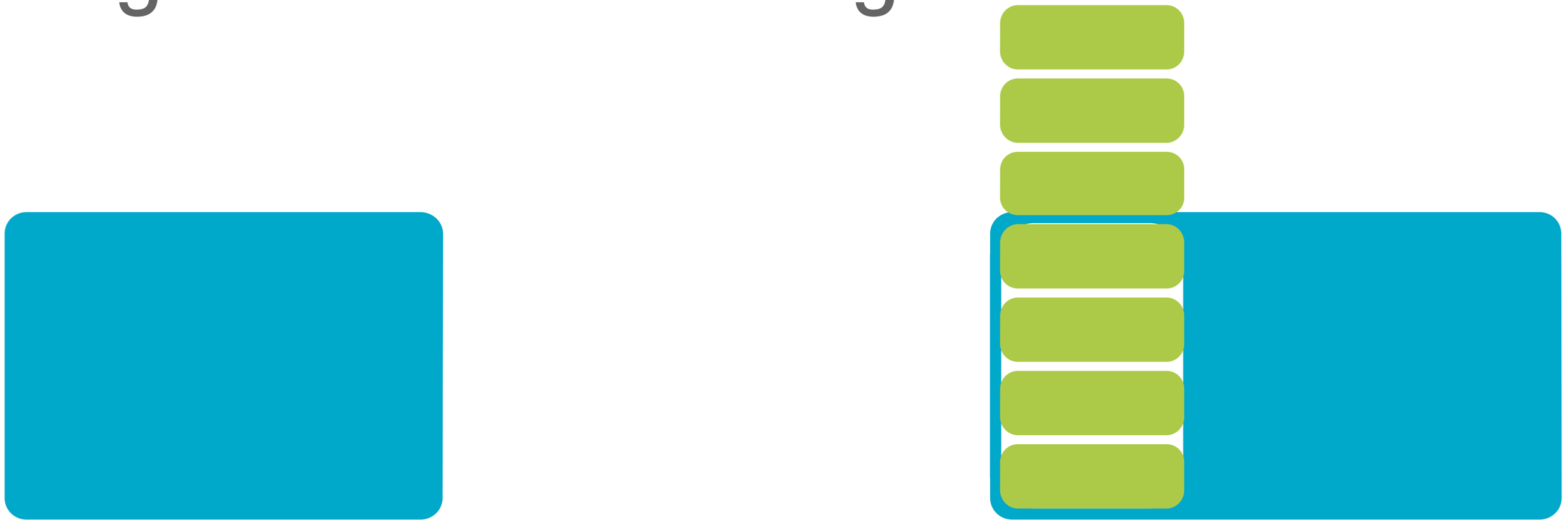
Negative Acknowledge



Negative Acknowledge



Negative Acknowledge



Отправитель

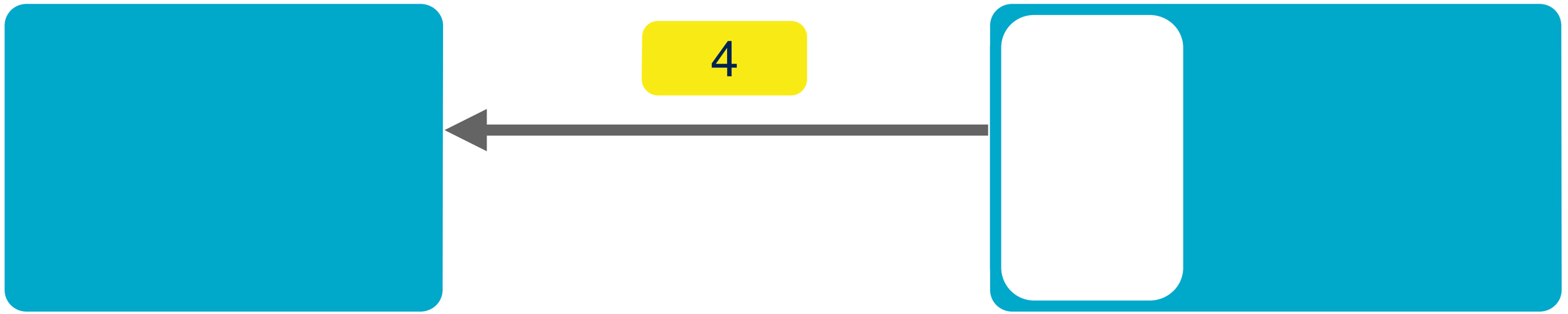
Получатель

Dynamic pull-push

Отправитель

Получатель

Dynamic pull-push



Dynamic pull-push

Отправитель

Получатель

Dynamic pull-push

Dynamic pull-push

Dynamic pull-push

Dynamic pull-push

Отправитель

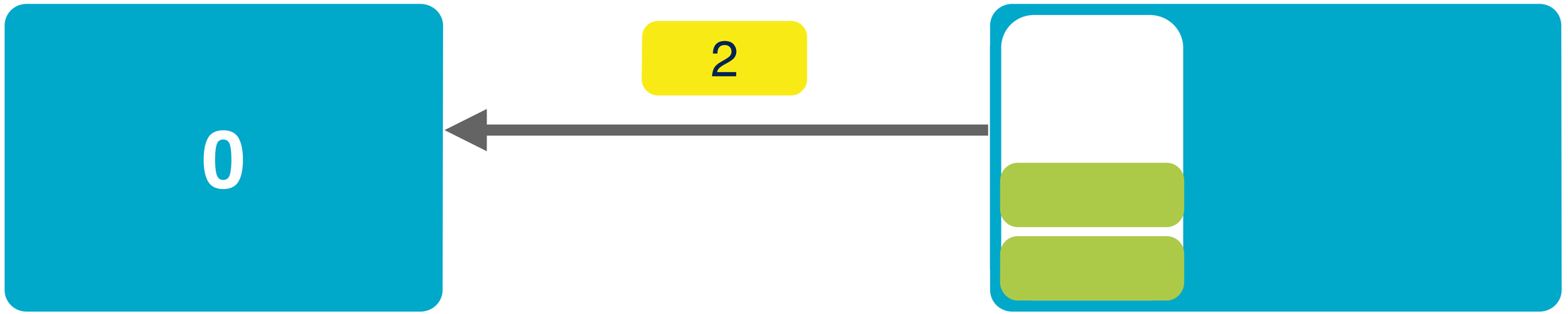
Получатель

Dynamic pull-push

Отправитель

Получатель

Dynamic pull-push



Dynamic pull-push

Отправитель

Получатель

Dynamic pull-push

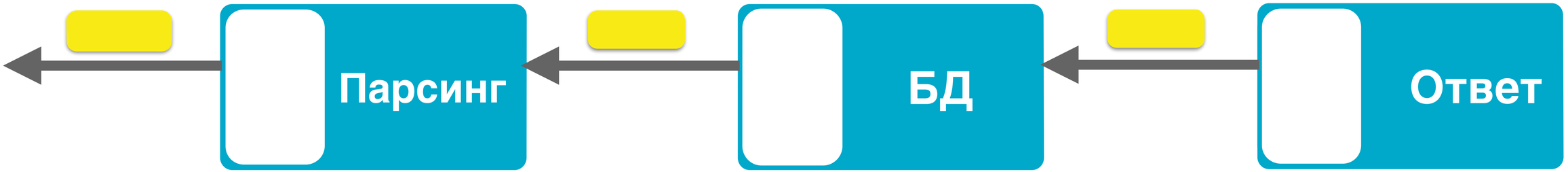
Все в сборе

Все в сборе

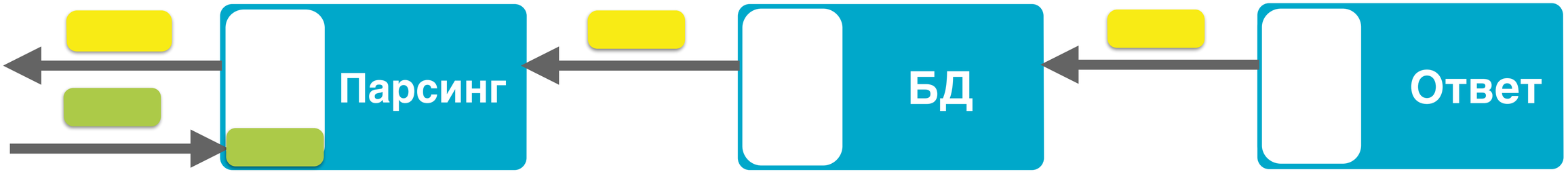
Все в сборе

Все в сборе

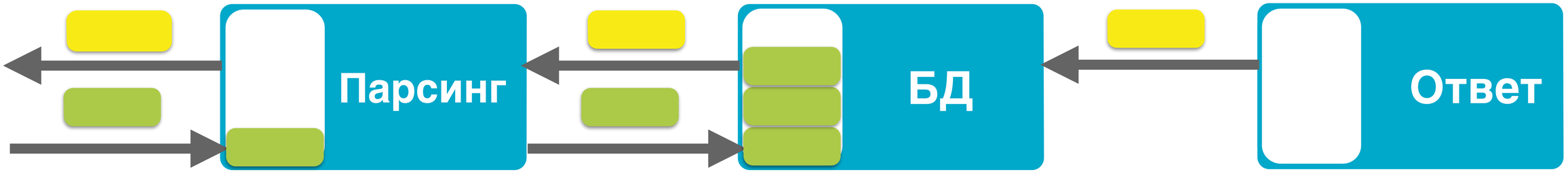
Все в сборе



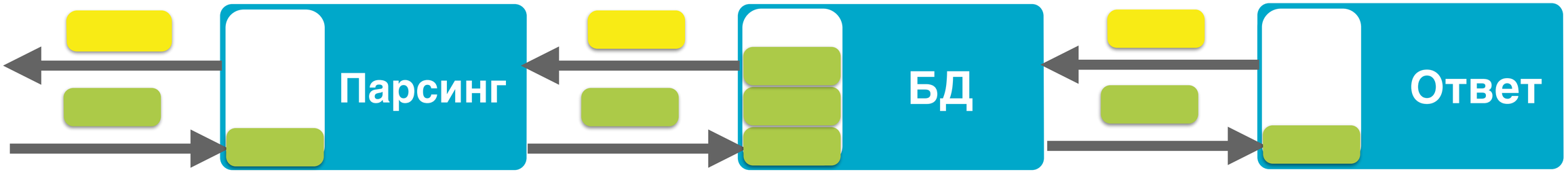
Все в сборе



Все в сборе



Все в сборе



Reactive Streams

Reactive Streams

Reactive Streams

- Спецификация

Reactive Streams

- Спецификация
- Асинхронного взаимодействия

Reactive Streams

- Спецификация
- Асинхронного взаимодействия
- Упорядоченные сообщения

Reactive Streams

- Спецификация
- Асинхронного взаимодействия
- Упорядоченные сообщения
- Обратная связь

Реализации

Реализации

- RxJava

Реализации

- RxJava
- Akka Stream

Реализации

- RxJava
- Akka Stream
- Reactor

Реализации

- RxJava
- Akka Stream
- Reactor
- Ratpack

“Talk is cheap,
show me the code”

Akka Stream

Akka Stream

- Reactive Stream

Akka Stream

- Reactive Stream
- Dynamic pull-push model

Akka Stream

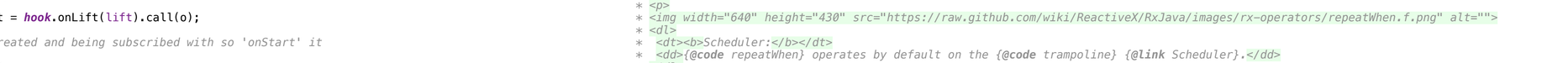
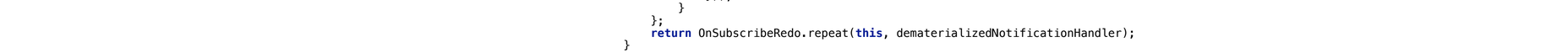
- Reactive Stream
- Dynamic pull-push model
- Scala, Java

Akka Stream

- Reactive Stream
- Dynamic pull-push model
- Scala, Java
- Статическая типизация

Akka Stream

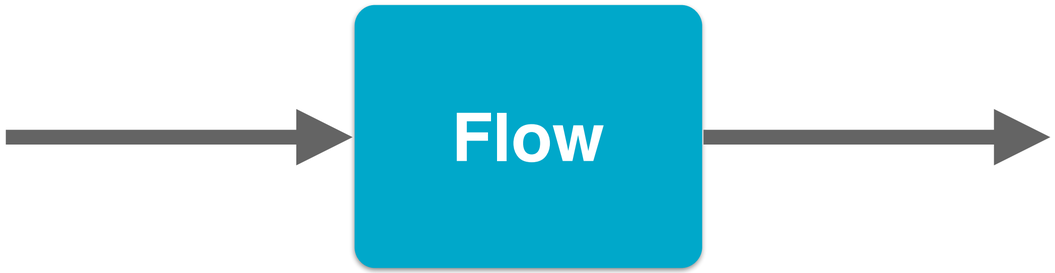
- Reactive Stream
- Dynamic pull-push model
- Scala, Java
- Статическая типизация
- Модель акторов

```java
/**
 * Operator function for lifting into an Observable.
 */
public interface Operator<R, T> extends Func1<Subscriber<? super R>, Subscriber<? super T>> {
    // cover for generics insanity
}

/**
 * Lifts a function to the current Observable and returns a new Observable that when subscribed to will pass
 * the values of the current Observable through the Operator function.
 * <p>
 * In other words, this allows chaining Observers together on an Observable for acting on the values within
 * the Observable.
 * <p> {@code
 * observable.map(...).filter(...).take(5).lift(new OperatorA()).lift(new OperatorB(...)).subscribe()
 * }
 * <p>
 * If the operator you are creating is designed to act on the individual items emitted by a source
 * Observable, use {@code lift}. If your operator is designed to transform the source Observable as a whole
 * (for instance, by applying a particular set of existing RxJava operators to it) use {@link #compose}.
 * <dl>
 *  <dt><b>Scheduler:</b></dt>
 *  <dd>{@code lift} does not operate by default on a particular {@link Scheduler}.</dd>
 * </dl>
 *
 * @param lift the Operator that implements the Observable-operating function to be applied to the source
 *             Observable
 * @return an Observable that is the result of applying the lifted Operator to the source Observable
 * @see <a href="https://github.com/ReactiveX/RxJava/wiki/Implementing-Your-Own-Operators">RxJava wiki: Implementing Your Own Operators</a>
 */
public final <R> Observable<R> lift(final Operator<? extends R, ? super T> lift) {
    return new Observable<R>(new OnSubscribe<R>() {
        @Override
        public void call(Subscriber<? super R> o) {
            try {
                Subscriber<? super T> st = hook.onLift(lift).call(o);
                try {
                    // new Subscriber created and being subscribed with so 'onStart' it
                    st.onStart();
                    onSubscribe.call(st);
                } catch (Throwable e) {
                    // localized capture of errors rather than it skipping all operators
                    // and ending up in the try/catch of the subscribe method which then
                    // prevents onErrorResumeNext and other similar approaches to error handling
                    if (e instanceof OnErrorNotImplementedException) {
                        throw (OnErrorNotImplementedException) e;
                    }
                    st.onError(e);
                }
            } catch (Throwable e) {
                if (e instanceof OnErrorNotImplementedException) {
                    throw (OnErrorNotImplementedException) e;
                }
                // if the lift function failed all we can do is pass the error to the final Subscriber
                // as we don't have the operator available to us
                o.onError(e);
            }
        }
    });
}


/**
 * Transform an Observable by applying a particular Transformer function to it.
 * <p>
 * This method operates on the Observable itself whereas {@link #lift} operates on the Observable's
 * Subscribers or Observers.
 * <p>
 * If the operator you are creating is designed to act on the individual items emitted by a source
 * Observable, use {@link #lift}. If your operator is designed to transform the source Observable as a whole
 * (for instance, by applying a particular set of existing RxJava operators to it) use {@code compose}.
 * <dl>
 *  <dt><b>Scheduler:</b></dt>
 *  <dd>{@code compose} does not operate by default on a particular {@link Scheduler}.</dd>
 * </dl>
 *
 * @param transformer implements the function that transforms the source Observable
 * @return the source Observable, transformed by the transformer function
 * @see <a href="https://github.com/ReactiveX/RxJava/wiki/Implementing-Your-Own-Operators">RxJava wiki: Implementing Your Own Operators</a>
 */
@SuppressWarnings("unchecked")
public <R> Observable<R> compose(Transformer<? super T, ? extends R> transformer) {
    return ((Transformer<T, R>) transformer).call(this);
}

/**
 * Transformer function used by {@link #compose}.
 * @warn more complete description needed
 */
public static interface Transformer<T, R> extends Func1<Observable<T>, Observable<R>> {
    // cover for generics insanity
}



/* ********************************************************************************************************
 * Operators Below Here
 * ********************************************************************************************************
 */

/**
 * Returns an Observable that emits the same values as the source Observable with the exception of an
 * {@code onCompleted}. An {@code onCompleted} notification from the source will result in the emission of
 * a {@code void} item to the Observable provided as an argument to the {@code notificationHandler}
 * function. If that Observable calls {@code onComplete} or {@code onError} then {@code repeatWhen} will
 * call {@code onCompleted} or {@code onError} on the child subscription. Otherwise, this Observable will
 * resubscribe to the source observable.
 * <p>
 * 
 * <dl>
 *  <dt><b>Scheduler:</b></dt>
 *  <dd>{@code repeatWhen} operates by default on the {@code trampoline} {@link Scheduler}.</dd>
 * </dl>
 *
 * @param notificationHandler
 *            receives an Observable of notifications with which a user can complete or error, aborting the repeat.
 * @param scheduler
 *            the {@link Scheduler} to emit the items on
 * @return the source Observable modified with repeat logic
 * @see <a href="http://reactivex.io/documentation/operators/repeat.html">ReactiveX operators documentation: Repeat</a>
 */
public final Observable<T> repeatWhen(final Func1<? super Observable<? extends Void>, ? extends Observable<?>> notificationHandler, Scheduler scheduler) {
    Func1<? super Observable<? extends Notification<?>>, ? extends Observable<?>> dematerializedNotificationHandler = new Func1<Observable<? extends Notification<?>>, Observable<?>>() {
        @Override
        public Observable<?> call(Observable<? extends Notification<?>> notifications) {
            return notificationHandler.call(notifications.map(new Func1<Notification<?>, Void>() {
                @Override
                public Void call(Notification<?> notification) {
                    return null;
                }
            }));
        }
    };
    return OnSubscribeRedo.repeat(this, dematerializedNotificationHandler, scheduler);
}

/**
 * Returns an Observable that emits the same values as the source Observable with the exception of an
 * {@code onCompleted}. An {@code onCompleted} notification from the source will result in the emission of
 * a {@code void} item to the Observable provided as an argument to the {@code notificationHandler}
 * function. If that Observable calls {@code onComplete} or {@code onError} then {@code repeatWhen} will
 * call {@code onCompleted} or {@code onError} on the child subscription. Otherwise, this Observable will
 * resubscribe to the source observable.
 * <p>
 * 
 * <dl>
 *  <dt><b>Scheduler:</b></dt>
 *  <dd>{@code repeatWhen} operates by default on the {@code trampoline} {@link Scheduler}.</dd>
 * </dl>
 *
 * @param notificationHandler
 *            receives an Observable of notifications with which a user can complete or error, aborting the repeat.
 * @return the source Observable modified with repeat logic
 * @see <a href="http://reactivex.io/documentation/operators/repeat.html">ReactiveX operators documentation: Repeat</a>
 */
public final Observable<T> repeatWhen(final Func1<? super Observable<? extends Void>, ? extends Observable<?>> notificationHandler) {
    Func1<? super Observable<? extends Notification<?>>, ? extends Observable<?>> dematerializedNotificationHandler = new Func1<Observable<? extends Notification<?>>, Observable<?>>() {
        @Override
        public Observable<?> call(Observable<? extends Notification<?>> notifications) {
            return notificationHandler.call(notifications.map(new Func1<Notification<?>, Void>() {
                @Override
                public Void call(Notification<?> notification) {
                    return null;
                }
            }));
        }
    };
    return OnSubscribeRedo.repeat(this, dematerializedNotificationHandler);
}
/**
 * An Observable that never sends any information to an {@link Observer}.
 * This Observable is useful primarily for testing purposes.
 *
 * @param <T>
 *            the type of item (not) emitted by the Observable
 */
private static class NeverObservable<T> extends Observable<T> {
    public NeverObservable() {
        super(new OnSubscribe<T>() {

            @Override
            public void call(Subscriber<? super T> observer) {
                // do nothing
            }

        });
    }
}

/**
 * An Observable that invokes {@link Observer#onError onError} when the {@link Observer} subscribes to it.
 *
 * @param <T>
 *            the type of item (ostensibly) emitted by the Observable
 */
private static class ThrowObservable<T> extends Observable<T> {

    public ThrowObservable(final Throwable exception) {
        super(new OnSubscribe<T>() {

            /**
             * Accepts an {@link Observer} and calls its {@link Observer#onError onError} method.
             *
             * @param observer
             *            an {@link Observer} of this Observable
             */
            @Override
```

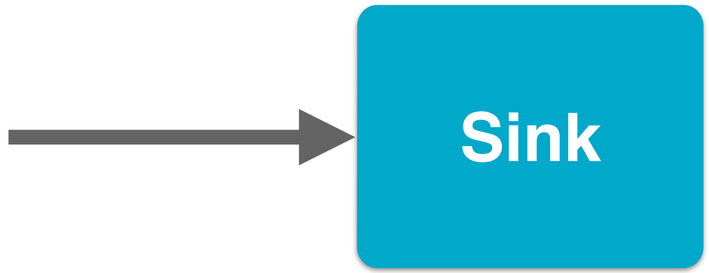
Ключевые абстракции

Ключевые абстракции

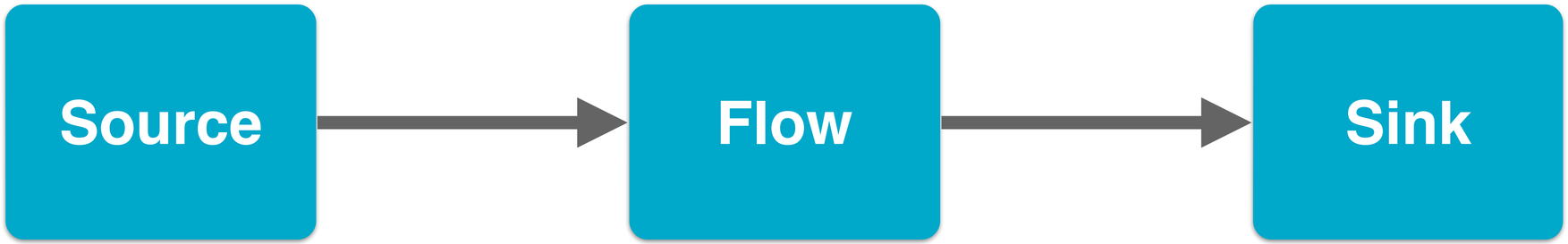
Ключевые абстракции



Ключевые абстракции



Ключевые абстракции



Source


```
val iterableSource = Source(1 to 50)
val tickSource = Source(1 second, 1 second, "Tick")
val singleSource = Source.single("devconf")
val emptySource = Source.empty()
val zmqSource = ???
```

Sink


```
val blackhole = Sink.ignore
val onComplete = Sink.onComplete { result =>
  System.exit(0)
}
val foreach = Sink.foreach(println)
val firstElement = Sink.head[Int]
```

Flow


```
implicit val as = ActorSystem("devconf")
implicit val materializer = ActorFlowMaterializer()

val source = Source(1 to 50)
val sink = Sink.foreach[Int](println)
val flow = source.to(sink)
flow.run()
```

Flow


```
implicit val as = ActorSystem("devconf")
implicit val materializer = ActorFlowMaterializer()

val source = Source(1 to 50)
val sink = Sink.foreach[Int](println)
val flow = source.to(sink)
flow.run()
```

Flow


```
implicit val as = ActorSystem("devconf")
implicit val materializer = ActorFlowMaterializer()

val source = Source(1 to 50)
val sink = Sink.foreach[Int](println)
val flow = source.to(sink)
flow.run()
```

Flow


```
implicit val as = ActorSystem("devconf")
implicit val materializer = ActorFlowMaterializer()

val source = Source(1 to 50)
val sink = Sink.foreach[Int](println)
val flow = source.to(sink)
flow.run()
```

Flow


```
implicit val as = ActorSystem("devconf")
implicit val materializer = ActorFlowMaterializer()

val source = Source(1 to 50)
val sink = Sink.foreach[Int](println)
val flow = source.to(sink)
flow.run()
```

Flow


```
implicit val as = ActorSystem("devconf")
implicit val materializer = ActorFlowMaterializer()

val source = Source(1 to 50)
val sink = Sink.foreach[Int](println)
val flow = source.to(sink)
flow.run()
```

Flow


```
implicit val as = ActorSystem("devconf")
implicit val materializer = ActorFlowMaterializer()

val source = Source(1 to 50)
val sink = Sink.foreach[Int](println)
val flow = source.to(sink)
flow.run()
```

Flow


```
val flow2 = source
   .map { x => x * 2 }
   .filter { x => x % 3 == 0 }
   .to(sink)
flow2.run()
```

Flow


```
val flow2 = source
  .map { x => x * 2 }
  .filter { x => x % 3 == 0 }
  .to(sink)
flow2.run()
```

Flow


```
val flow2 = source
   .map { x => x * 2 }
   .filter { x => x % 3 == 0 }
   .to(sink)
flow2.run()
```

Flow


```
val flow2 = source
   .map { x => x * 2 }
   .filter { x => x % 3 == 0 }
   .to(sink)
flow2.run()
```

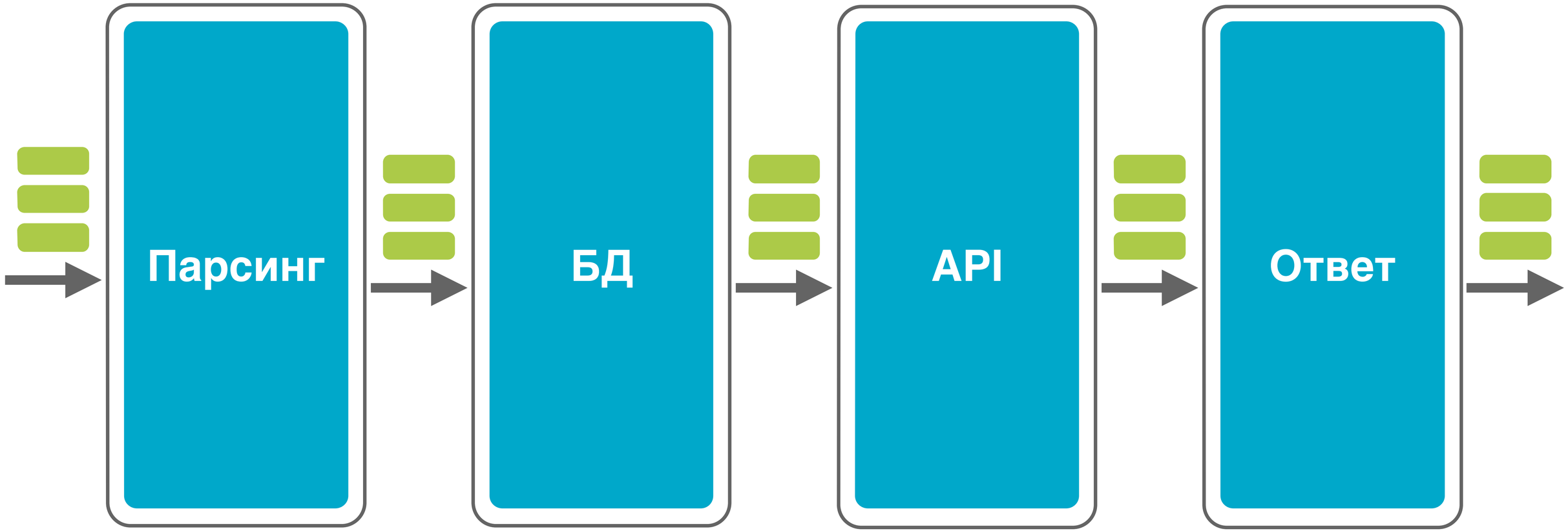
Flow


```
val source = Source(1 to 50)
val sink = Sink.foreach[String](println)

val flow2 = source
  .map { x => x.toString }
  .map { x => x / 13 }
  .to(sink)
flow2.run()
```

Flow


```
val source = Source(1 to 50)
val sink = Sink.foreach[String](println)

val flow2 = source
  .map { x => x.toString }
  .map { x => x / 13 }
  .to(sink)
flow2.run()
```

Flow



Flow


```
val request: Source[Request] = ???
def parser: Request => Query = ???
def dbCall: Query => Future[List[Int]] = ???
def apiCall: List[Int] => Future[List[String]] = ???
def buildResponse: List[String] => Response = ???
```

Flow


```
val request: Source[Request] = ???
def parser: Request => Query = ???
def dbCall: Query => Future[List[Int]] = ???
def apiCall: List[Int] => Future[List[String]] = ???
def buildResponse: List[String] => Response = ???
```

Flow


```
val request: Source[Request] = ???
def parser: Request => Query = ???
def dbCall: Query => Future[List[Int]] = ???
def apiCall: List[Int] => Future[List[String]] = ???
def buildResponse: List[String] => Response = ???
```

Flow


```
val request: Source[Request] = ???
def parser: Request => Query = ???
def dbCall: Query => Future[List[Int]] = ???
def apiCall: List[Int] => Future[List[String]] = ???
def buildResponse: List[String] => Response = ???
```

Flow


```
val request: Source[Request] = ???
def parser: Request => Query = ???
def dbCall: Query => Future[List[Int]] = ???
def apiCall: List[Int] => Future[List[String]] = ???
def buildResponse: List[String] => Response = ???
```

Flow


```
val request: Source[Request] = ???
def parser: Request => Query = ???
def dbCall: Query => Future[List[Int]] = ???
def apiCall: List[Int] => Future[List[String]] = ???
def buildResponse: List[String] => Response = ???
```

Flow


```
val request: Source[Request] = ???
def parser: Request => Query = ???
def dbCall: Query => Future[List[Int]] = ???
def apiCall: List[Int] => Future[List[String]] = ???
def buildResponse: List[String] => Response = ???
```

Flow


```
val request: Source[Request] = ???
def parser: Request => Query = ???
def dbCall: Query => Future[List[Int]] = ???
def apiCall: List[Int] => Future[List[String]] = ???
def buildResponse: List[String] => Response = ???


val flow3 = request
  .map(parser)
  .mapAsync(dbCall)
  .mapAsync(apiCall)
  .map(buildResponse)
  .to(response)
```

Flow


```
val request: Source[Request] = ???
def parser: Request => Query = ???
def dbCall: Query => Future[List[Int]] = ???
def apiCall: List[Int] => Future[List[String]] = ???
def buildResponse: List[String] => Response = ???


val flow3 = request
  .map(parser)
  .mapAsync(dbCall)
  .mapAsync(apiCall)
  .map(buildResponse)
  .to(response)
```

Flow


```
val request: Source[Request] = ???
def parser: Request => Query = ???
def dbCall: Query => Future[List[Int]] = ???
def apiCall: List[Int] => Future[List[String]] = ???
def buildResponse: List[String] => Response = ???


val flow3 = request
  .map(parser)
  .mapAsync(dbCall)
  .mapAsync(apiCall)
  .map(buildResponse)
  .to(response)
```

Flow


```
val request: Source[Request] = ???
def parser: Request => Query = ???
def dbCall: Query => Future[List[Int]] = ???
def apiCall: List[Int] => Future[List[String]] = ???
def buildResponse: List[String] => Response = ???

val flow3 = request
  .map(parser)
  .mapAsync(dbCall)
  .mapAsync(apiCall)
  .map(buildResponse)
  .to(response)
```

Flow


```
val request: Source[Request] = ???
def parser: Request => Query = ???
def dbCall: Query => Future[List[Int]] = ???
def apiCall: List[Int] => Future[List[String]] = ???
def buildResponse: List[String] => Response = ???


val flow3 = request
  .map(parser)
  .mapAsync(dbCall)
  .mapAsync(apiCall)
  .map(buildResponse)
  .to(response)
```

Flow


```
val request: Source[Request] = ???
def parser: Request => Query = ???
def dbCall: Query => Future[List[Int]] = ???
def apiCall: List[Int] => Future[List[String]] = ???
def buildResponse: List[String] => Response = ???


val flow3 = request
  .map(parser)
  .mapAsync(dbCall)
  .mapAsync(apiCall)
  .map(buildResponse)
  .to(response)
```

Flow


```
val request: Source[Request] = ???
def parser: Request => Query = ???
def dbCall: Query => Future[List[Int]] = ???
def apiCall: List[Int] => Future[List[String]] = ???
def buildResponse: List[String] => Response = ???


val flow3 = request
  .map(parser)
  .mapAsync(dbCall)
  .mapAsync(apiCall)
  .map(buildResponse)
  .to(response)
```

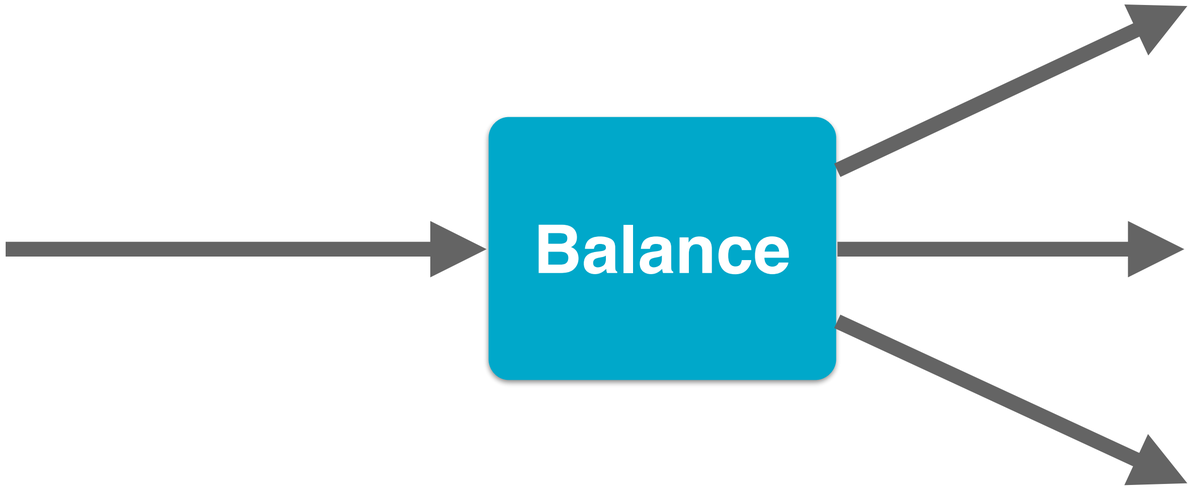
Flow


```
val request: Source[Request] = ???
def parser: Request => Query = ???
def dbCall: Query => Future[List[Int]] = ???
def apiCall: List[Int] => Future[List[String]] = ???
def buildResponse: List[String] => Response = ???


val flow3 = request
  .map(parser)
  .mapAsync(dbCall)
  .mapAsync(apiCall)
  .map(buildResponse)
  .to(response)
```

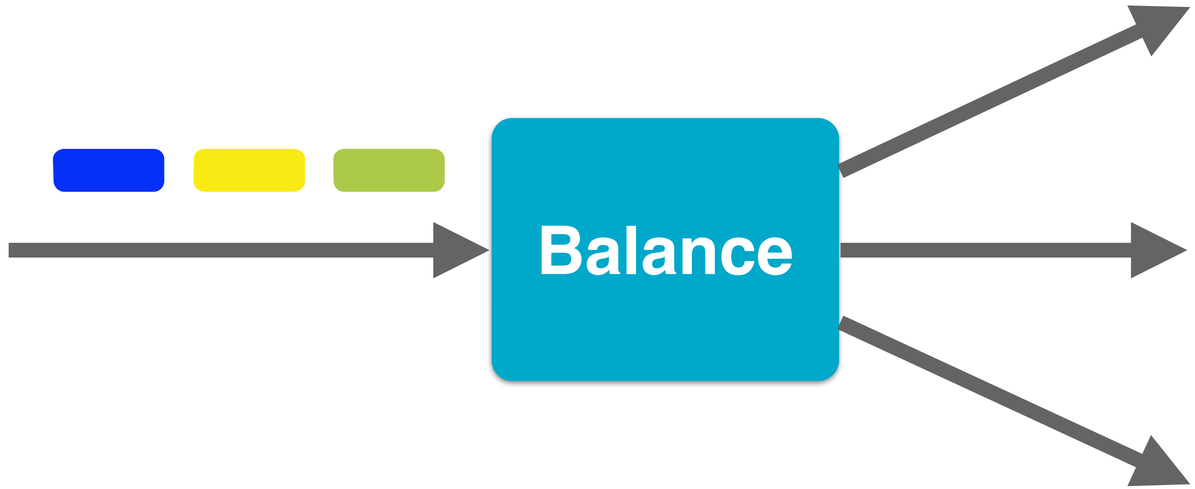
Flow

- drop, take
- group, mapConcat
- grouped, flatten
- buffer, conflate, expand

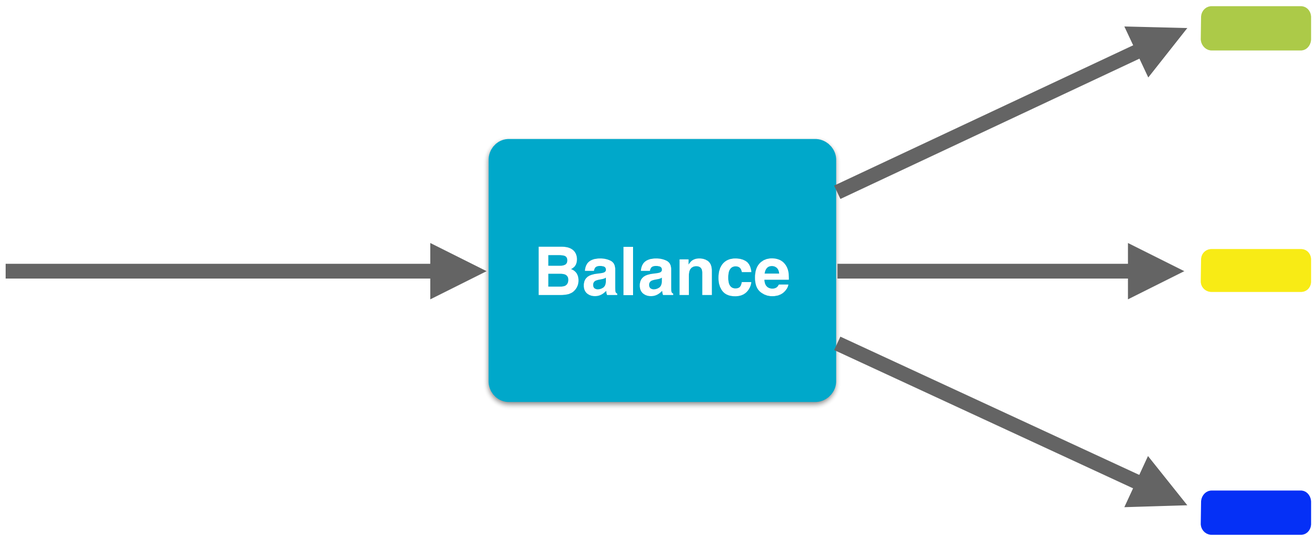
Balance



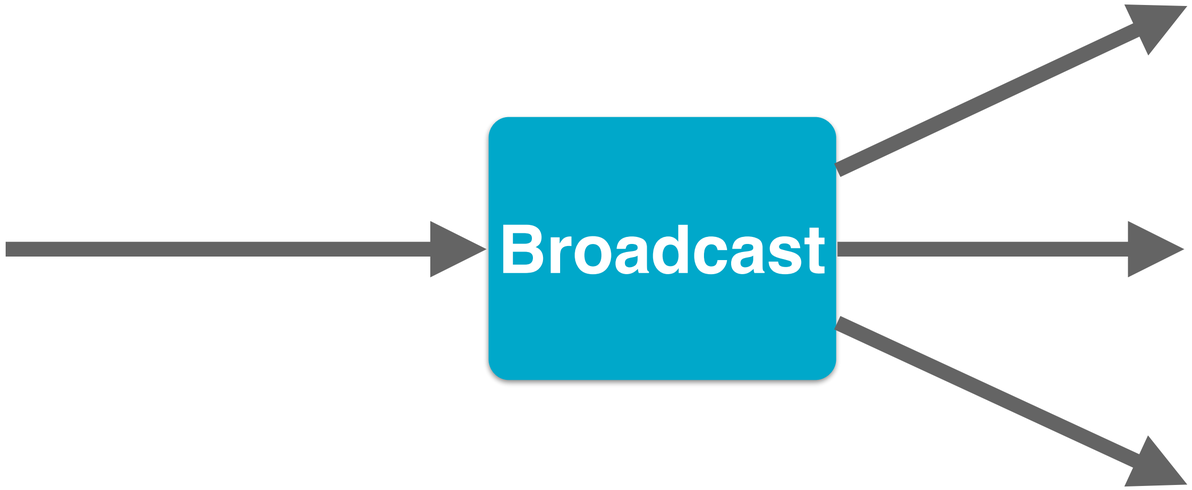
Balance



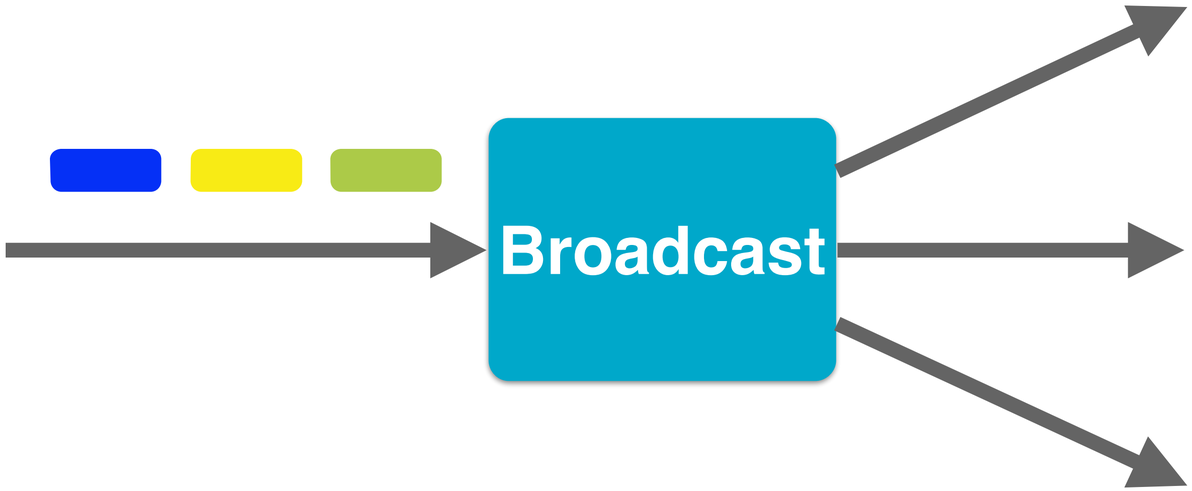
Balance



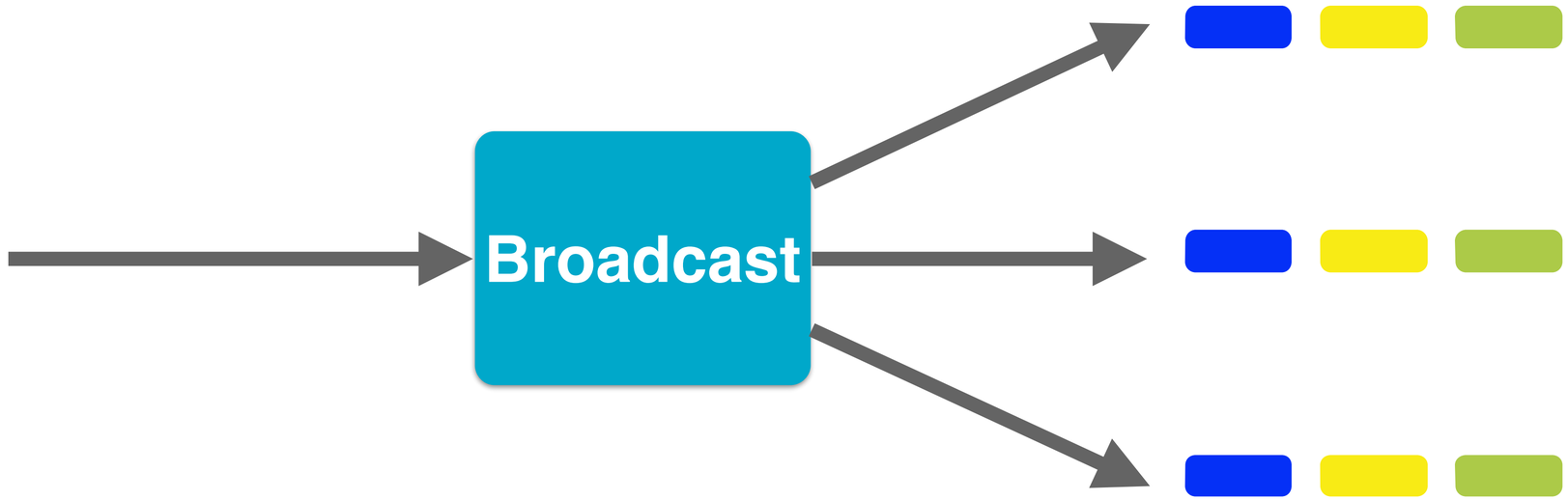
Broadcast



Broadcast

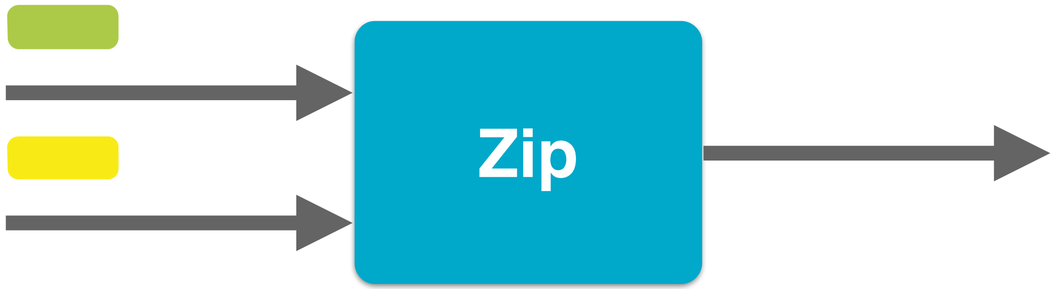


Broadcast

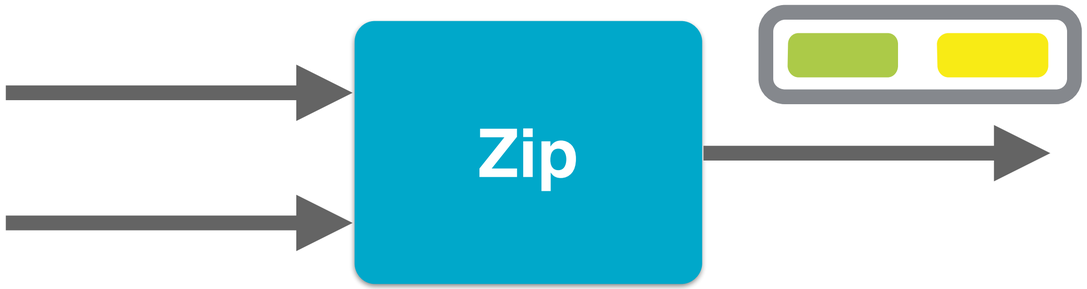


Zip

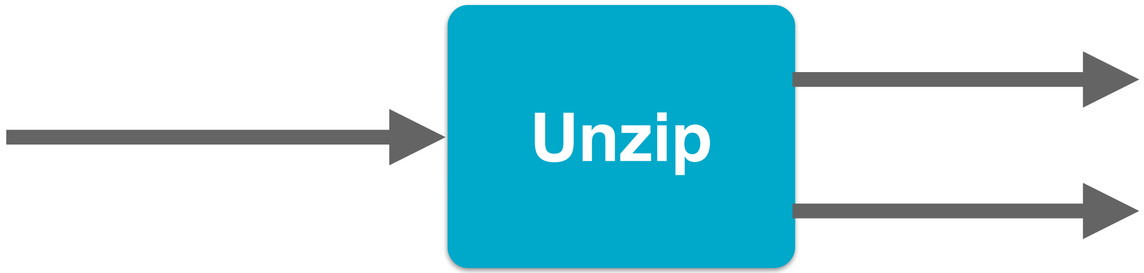
Zip



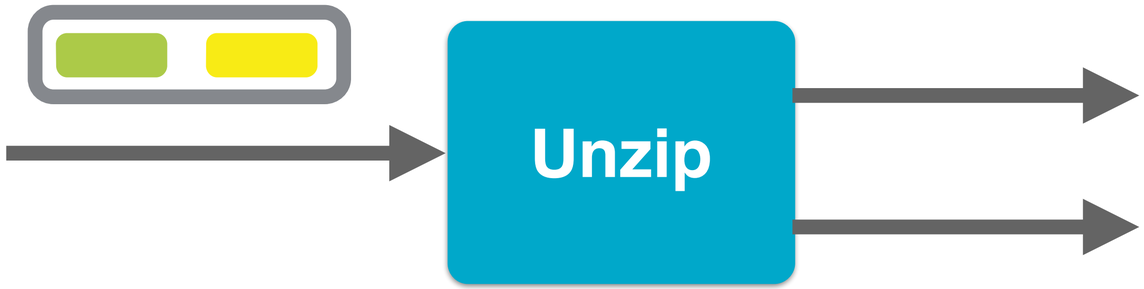
Zip



Unzip

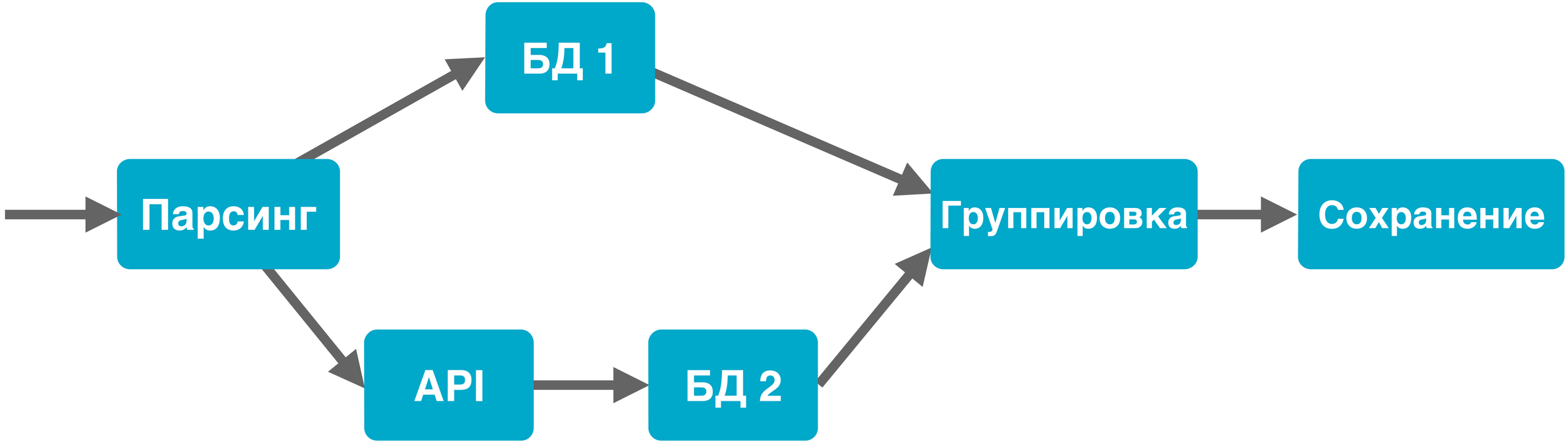


Unzip



Unzip

Пример посложнее



Применения

Применения

- MQ

Применения

- MQ
- Потоки данных (события, метрики, файлы, видео)

Применения

- MQ
- Потоки данных (события, метрики, файлы, видео)
- UI

Применения

- MQ
- Потоки данных (события, метрики, файлы, видео)
- UI
- Очереди задач

Adopters

Языки программирования

- C#
- Java, Scala
- JavaScript
- Objective-C
- Python
- Ruby
- PHP

Вопросы?

al.romanchuk@2gis.ru @1esha

Ссылки

- Reactive Streams
- Akka Stream
- Reactor
- Ratpack
- RxJava
- Reactive Manifesto

Ссылки

- Akka HTTP
- RxMongo
- https://github.com/pkinsky/akka-streams-example

Ссылки

- Reactive Extensions
- Reactive Extensions for JavaScript
- Reactive Cocoa
- Rx.py
- Rx.rb
- Rx.php