
Looking Ahead To
RxJava 2
Jake Wharton

# Reactive Streams

...is an initiative to provide a standard
for asynchronous stream processing
with non-blocking back pressure.

# Reactive Streams

```
interface Publisher<T> {
  void subscribe(Subscriber<? super T> s);
}
```

# Reactive Streams

```
interface Publisher<T> {
  void subscribe(Subscriber<? super T> s);
}

interface Subscriber<T> {
  void onNext(T t);
  void onComplete();
  void onError(Throwable t);
}
```

# Reactive Streams

```
interface Publisher<T> {
  void subscribe(Subscriber<? super T> s);
}

interface Subscriber<T> {
  void onNext(T t);
  void onComplete();
  void onError(Throwable t);
  void onSubscribe(Subscription s);
}
```

# Reactive Streams

```
interface Publisher<T> {
  void subscribe(Subscriber<? super T> s);
}

interface Subscriber<T> {
  void onNext(T t);
  void onComplete();
  void onError(Throwable t);
  void onSubscribe(Subscription s);
}

interface Subscription {
  void request(long n);
  void cancel();
}
```

# Reactive Streams

```
interface Publisher<T> {
  void subscribe(Subscriber<? super T> s);
}

interface Subscriber<T> {
  void onNext(T t);
  void onComplete();
  void onError(Throwable t);
  void onSubscribe(Subscription s);
}

interface Subscription {
  void request(long n);
  void cancel();
}

interface Processor<T, R> extends Subscriber<T>, Publisher<R> {
}
```

# Reactive Streams

```
class RxReactiveStreams {
  static <T> Publisher<T> toPublisher(Observable<T> o) { … }
  static <T> Observable<T> toObservable(Publisher<T> p) { … }
  static <T> Publisher<T> toPublisher(Single<T> o) { … }
  static <T> Single<T> toSingle(Publisher<T> p) { … }
  static <T> Publisher<T> toPublisher(Completable o) { … }
  static Completable toCompletable(Publisher<?> p) { … }
}
```

# Reactive Streams

```
class RxReactiveStreams {
  static <T> Publisher<T> toPublisher(Observable<T> o) { … }
  static <T> Observable<T> toObservable(Publisher<T> p) { … }
  static <T> Publisher<T> toPublisher(Single<T> o) { … }
  static <T> Single<T> toSingle(Publisher<T> p) { … }
  static <T> Publisher<T> toPublisher(Completable o) { … }
  static Completable toCompletable(Publisher<?> p) { … }
}
```

github.com/ReactiveX/RxJavaReactiveStreams

# Reactive Streams

```
interface Subscriber<T> {
  void onNext(T t);
  void onComplete();
  void onError(Throwable t);
  void onSubscribe(Subscription s);
}
```

# Reactive Streams

```
interface Subscriber<T> {
  void onNext(T t);
  void onComplete();
  void onError(Throwable t);
  void onSubscribe(Subscription s);
}
```

Calling **`onSubscribe`**, **`onNext`**, **`onError`** or **`onComplete`** MUST return normally

# Reactive Streams

```
interface Subscriber<T> {
  void onNext(T t);
  void onComplete();
  void onError(Throwable t);
  void onSubscribe(Subscription s);
}
```

Calling **`onSubscribe`**, **`onNext`**, **`onError`** or **`onComplete`** MUST return normally except when any provided parameter is **`null`** in which case it MUST throw a **`java.lang.NullPointerException`** to the caller

# Reactive Streams

```
interface Subscriber<T> {
  void onNext(T t);
  void onComplete();
  void onError(Throwable t);
  void onSubscribe(Subscription s);
}
```

Calling `onSubscribe`, `onNext`, `onError` or `onComplete` MUST return normally except when any provided parameter is `null` in which case it MUST throw a `java.lang.NullPointerException` to the caller

# Flowable

# Flowable

# Flowable

- Implementation of Reactive Streams `Publisher`.

# Flowable

- Implementation of Reactive Streams `Publisher`.
- Observed with Reactive Streams `Subscriber`.

# Flowable

- Implementation of Reactive Streams `Publisher`.
- Observed with Reactive Streams `Subscriber`.
- 0 to many items, completes or errors.

# Flowable

- Implementation of Reactive Streams `Publisher`.
- Observed with Reactive Streams `Subscriber`.
- 0 to many items, completes or errors.
- Handles backpressure to slow down events with Reactive Streams `Subscription`.

# Flowable

- Implementation of Reactive Streams `Publisher`.
- Observed with Reactive Streams `Subscriber`.
- 0 to many items, completes or errors.
- Handles backpressure to slow down events with Reactive Streams `Subscription`.
- Hundreds of convenience methods for transforming and composing data.

# Flowable

- Implementation of Reactive Streams `Publisher`.
- Observed with Reactive Streams `Subscriber`.
- 0 to many items, completes or errors.
- Handles backpressure to slow down events with Reactive Streams `Subscription`.
- Hundreds of convenience methods for transforming and composing data.

# Flowable (aka RxJava 1.x's Observable)

- Implementation of Reactive Streams `Publisher`.
- Observed with Reactive Streams `Subscriber`.
- 0 to many items, completes or errors.
- Handles backpressure to slow down events with Reactive Streams `Subscription`.
- Hundreds of convenience methods for transforming and composing data.

# FlowableProcessor

# FlowableProcessor

# FlowableProcessor

- Implementation of Reactive Streams `Processor` (aka `Publisher` via `Flowable` + `Subscriber`).

# FlowableProcessor

- Implementation of Reactive Streams `Processor` (aka `Publisher` via `Flowable` + `Subscriber`).

# FlowableProcessor (aka 1.x's Subject)

- Implementation of Reactive Streams `Processor` (aka `Publisher` via `Flowable` + `Subscriber`).

# Observable? Subject?

# Observable

- Observed with an `Observer` (not Reactive Streams type).
- 0 to many items, completes or errors.
- No backpressure support.
- Hundreds of convenience methods for transforming and composing data.

# Observable

- Observed with an `Observer` (not Reactive Streams type).
- 0 to many items, completes or errors.
- **No backpressure support.**
- Hundreds of convenience methods for transforming and composing data.

# Subject

- Implementation of `Observable` and `Observer`.

# Flowable vs. Observable

# Flowable vs. Observable

- RxJava 1.x added backpressure late in the design process.

# Flowable vs. Observable

- RxJava 1.x added backpressure late in the design process.
- All types exposed backpressure but not all sources respected it.

# Flowable vs. Observable

- RxJava 1.x added backpressure late in the design process.
- All types exposed backpressure but not all sources respected it.
- Backpressure, like inheritance, must be designed for.

# Flowable vs. Observable

- Backpressure, like inheritance, must be designed for.

```
RxView.touches(paintView)
    .subscribe(e -> draw(e));
```

# Flowable vs. Observable

- Backpressure, like inheritance, must be designed for.

```
RxView.touches(paintView)
    .subscribe(e -> draw(e));
```

# Flowable vs. Observable

- Backpressure, like inheritance, must be designed for.

```
RxView.touches(paintView)
    .subscribe(e -> draw(e));
```

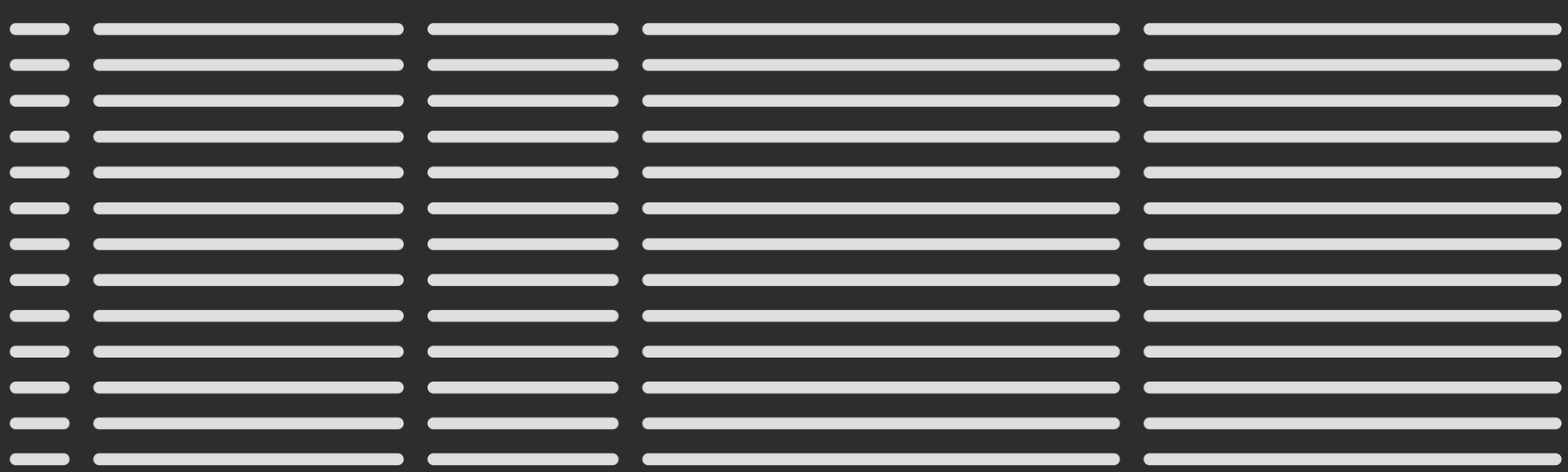
```
db.createQuery("SELECT * …")
    .subscribe(e -> update(e));
```

# Flowable vs. Observable

- Backpressure, like inheritance, must be designed for.

```
RxView.touches(paintView)
    .subscribe(e -> draw(e));
```

```
db.createQuery("SELECT * …")
    .subscribe(e -> update(e));
```



# Flowable vs. Observable

- Backpressure, like inheritance, must be designed for.

```
RxView.touches(paintView)
    .subscribe(e -> draw(e));
```

```
Observable<MotionEvent>
```

```
db.createQuery("SELECT * …")
    .subscribe(e -> update(e));
```

```
Observable<Row>
```

# Flowable vs. Observable

- Backpressure, like inheritance, must be designed for.

```
RxView.touches(paintView)
    .subscribe(e -> draw(e));
```

```
Observable<MotionEvent>
```

```
MissingBackpressureException
```

```
db.createQuery("SELECT * …")
    .subscribe(e -> update(e));
```

```
Observable<Row>
```

# Flowable vs. Observable

- Backpressure, like inheritance, must be designed for.

```
RxView.touches(paintView)
    .subscribe(e -> draw(e));
```

```
Observable<MotionEvent>
```

```
db.createQuery("SELECT * …")
    .subscribe(e -> update(e));
```

```
Flowable<Row>
```

# Flowable vs. Observable

```
Observable<MotionEvent>


interface Observer<T> {
  void onNext(T t);
  void onComplete();
  void onError(Throwable t);
  void onSubscribe(Disposable d);
}
```

```
Flowable<Row>


interface Subscriber<T> {
  void onNext(T t);
  void onComplete();
  void onError(Throwable t);
  void onSubscribe(Subscription s);
}
```

# Flowable vs. Observable

```
Observable<MotionEvent>


interface Observer<T> {
  void onNext(T t);
  void onComplete();
  void onError(Throwable t);
  void onSubscribe(Disposable d);
}


interface Disposable {
  void dispose();
}
```

```
Flowable<Row>


interface Subscriber<T> {
  void onNext(T t);
  void onComplete();
  void onError(Throwable t);
  void onSubscribe(Subscription s);
}


interface Subscription {
  void cancel();
  void request(long r);
}
```

# Converting

# Converting

Observable 

# Converting

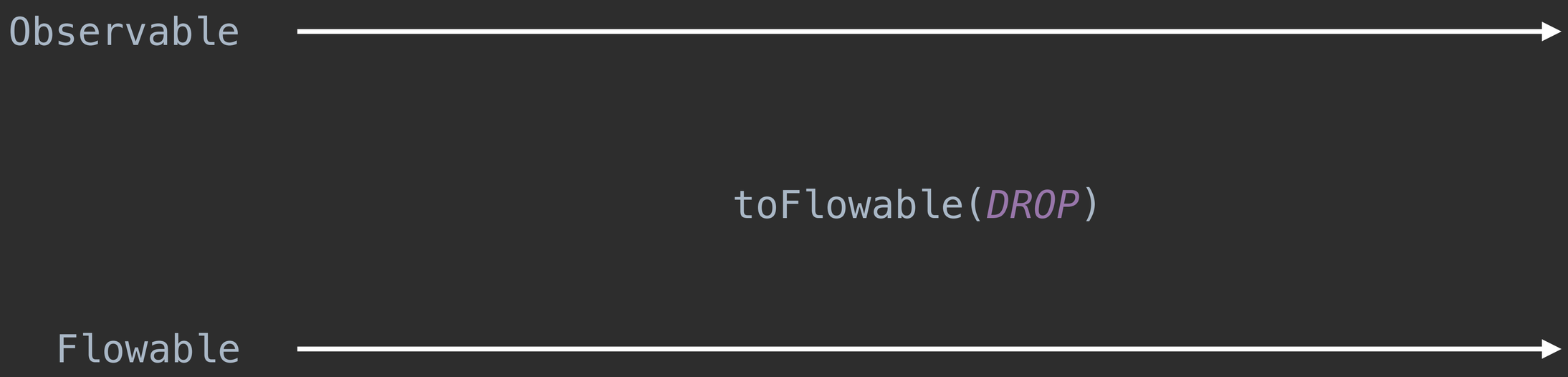
# Converting


`toFlowable()`

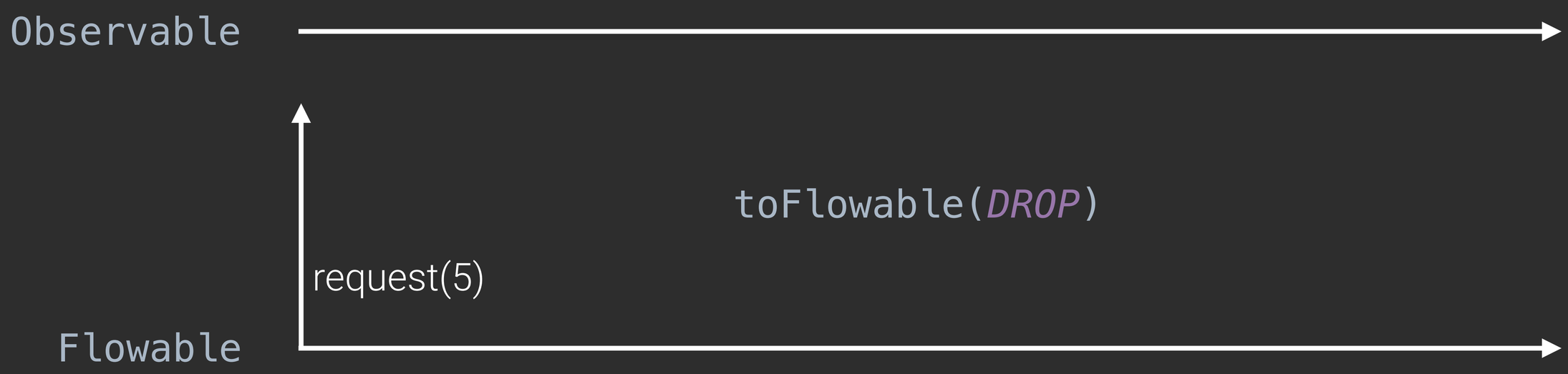
# Converting


```
toFlowable(DROP)
```

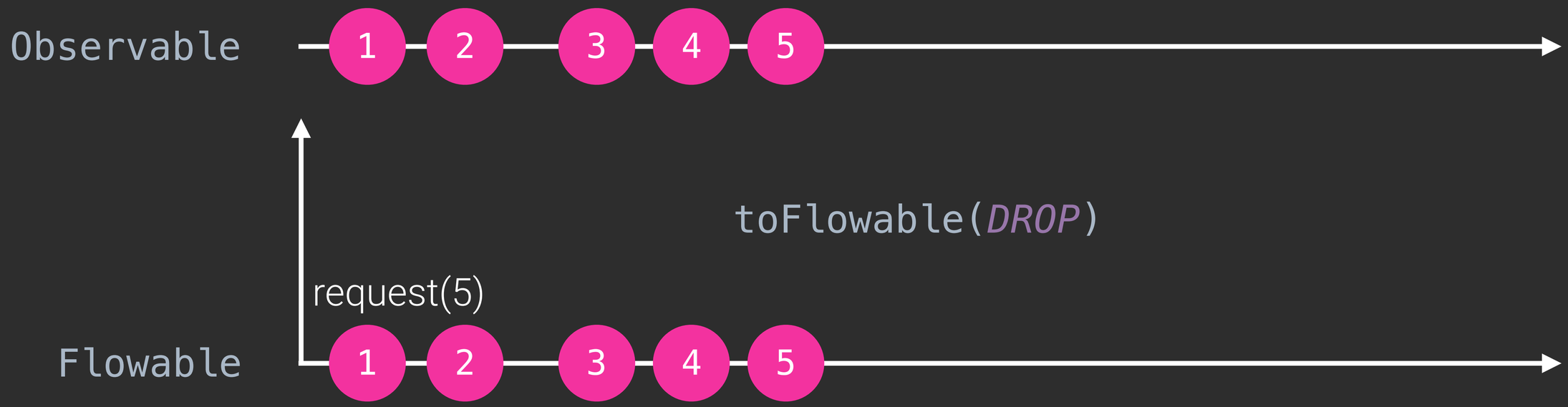
# Converting



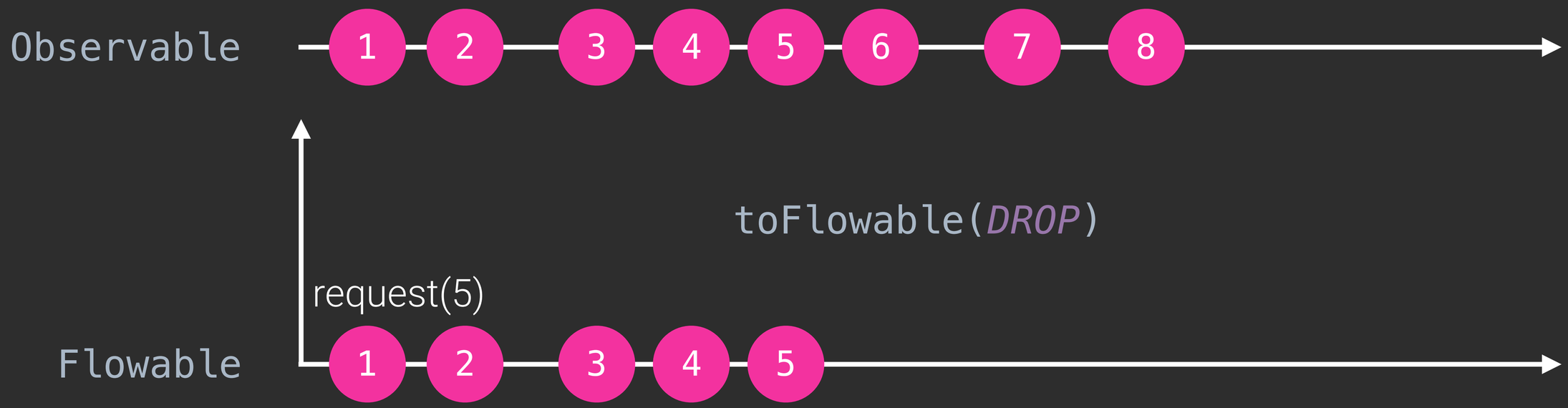
# Converting



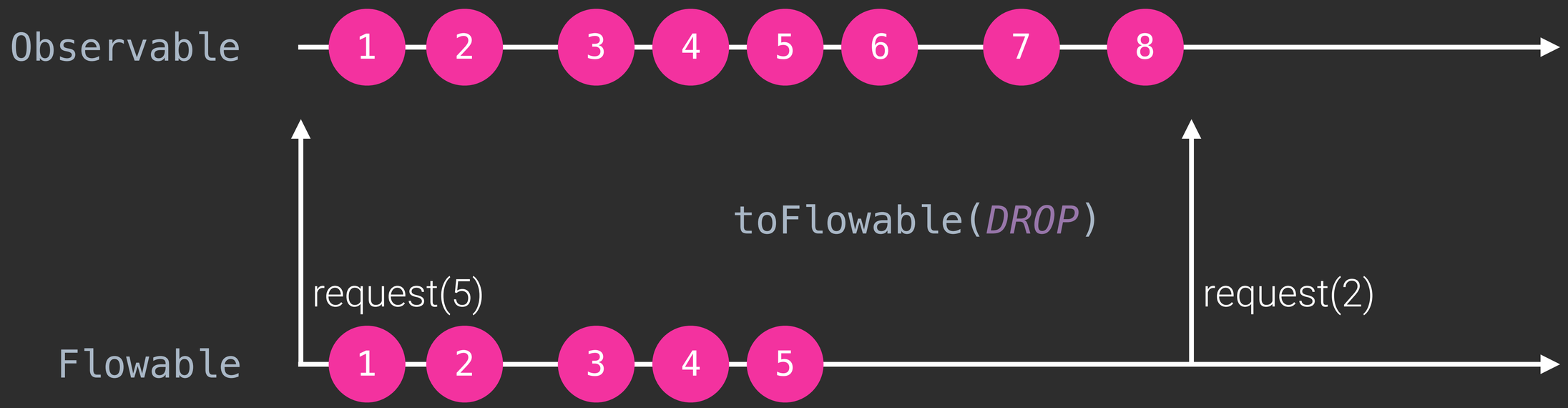
# Converting


Observable
1
2
3
4
5
toFlowable(DROP)
request(5)
Flowable
1
2
3
4
5

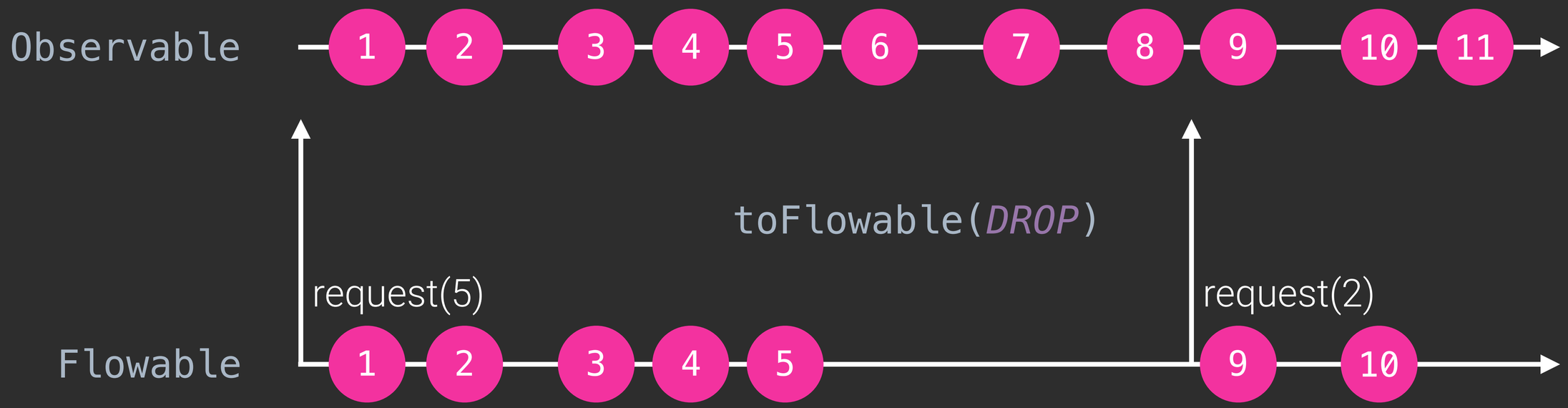
# Converting


Observable
1
2
3
4
5
6
7
8
toFlowable(DROP)
request(5)
Flowable
1
2
3
4
5

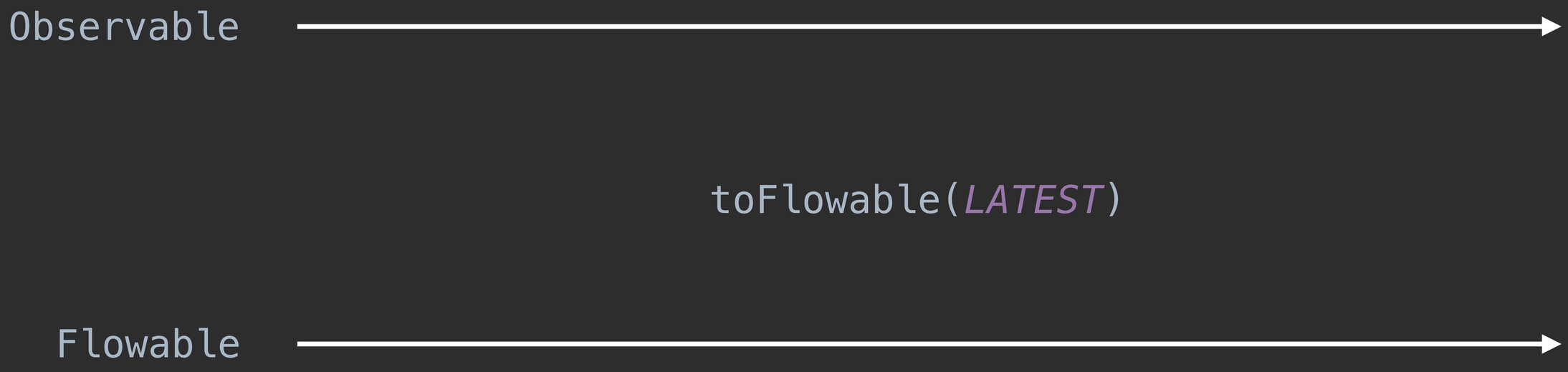
# Converting


Observable
1
2
3
4
5
6
7
8
toFlowable(DROP)
request(5)
request(2)
Flowable
1
2
3
4
5

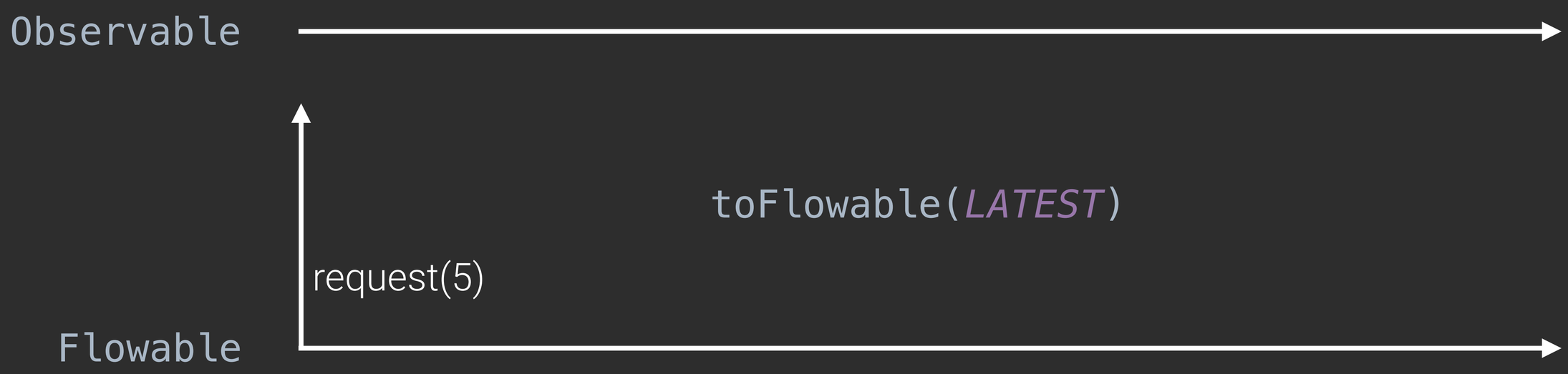
# Converting


Observable
1
2
3
4
5
6
7
8
9
10
11
toFlowable(DROP)
request(5)
request(2)
Flowable
1
2
3
4
5
9
10

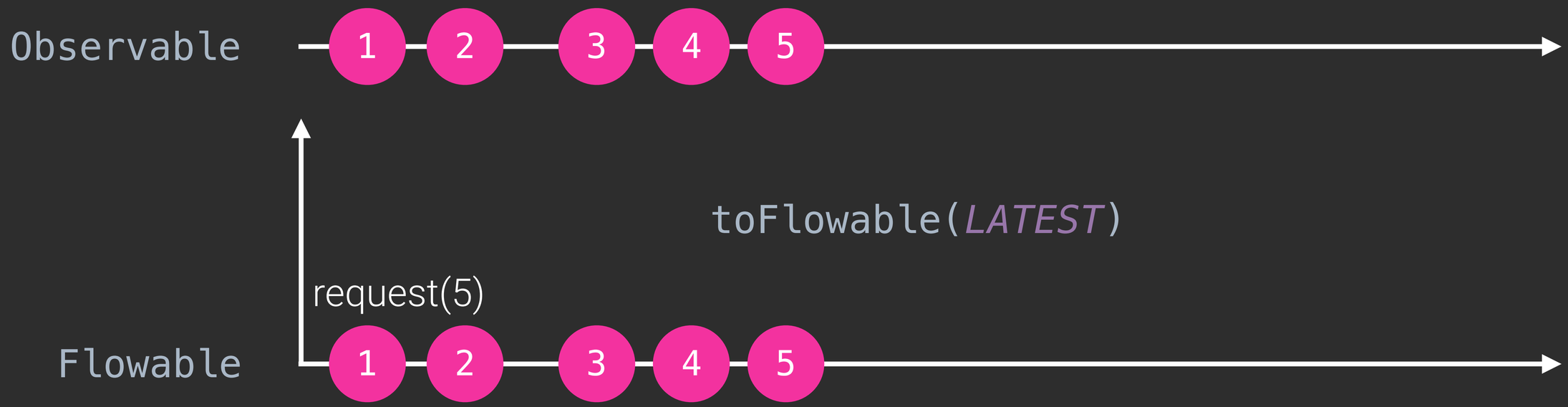
# Converting



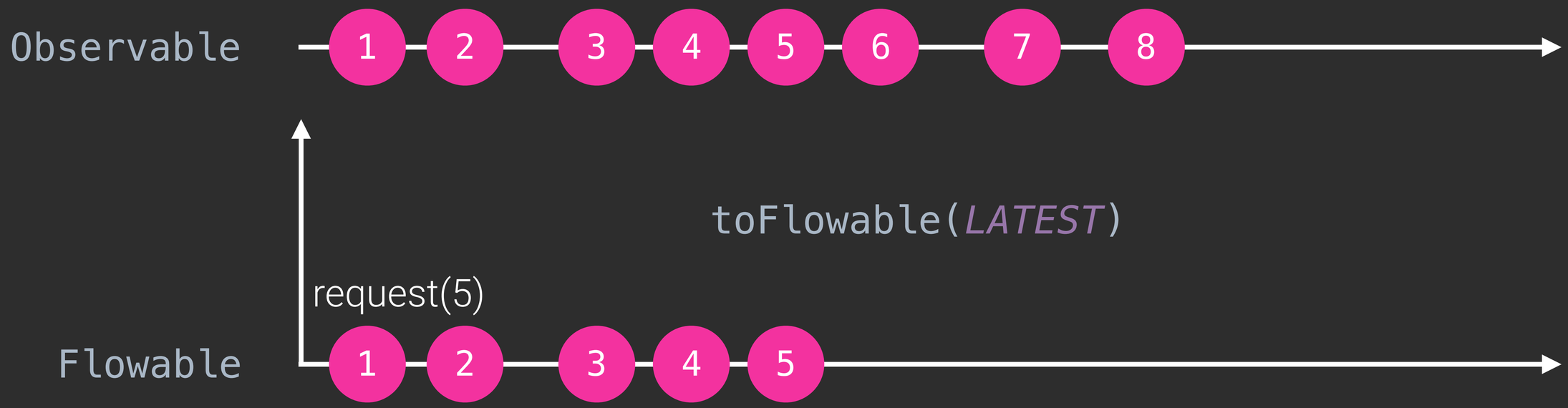
# Converting



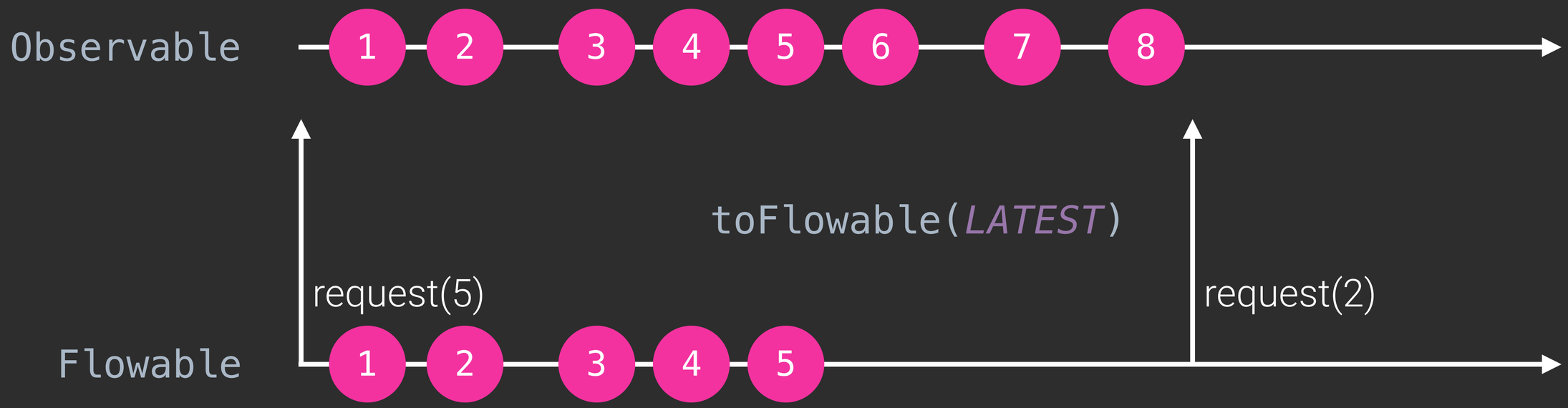
# Converting


Observable
1
2
3
4
5
toFlowable(LATEST)
request(5)
Flowable
1
2
3
4
5

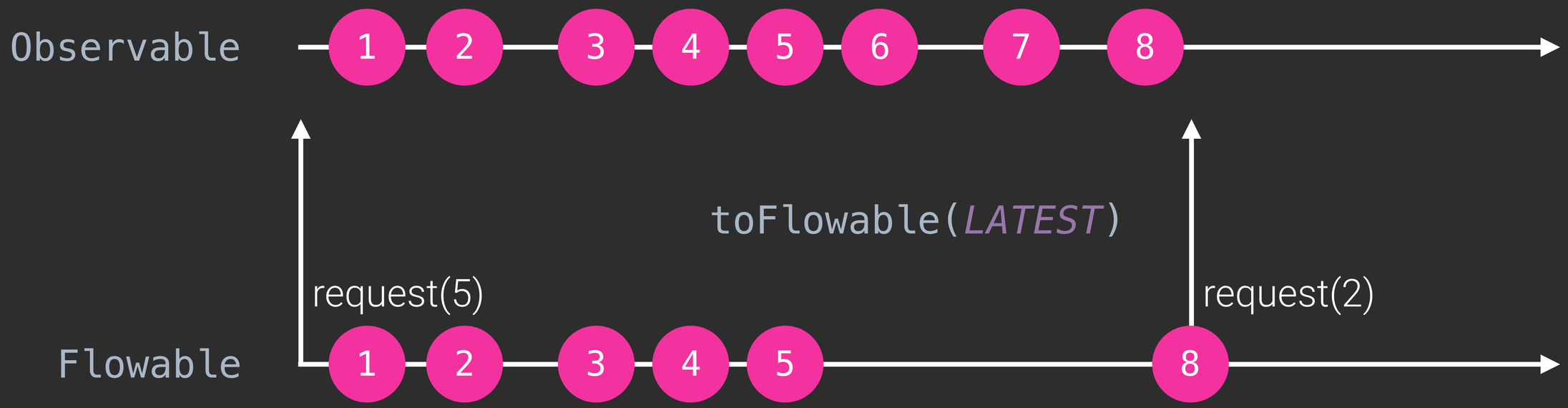
# Converting


Observable
1
2
3
4
5
6
7
8
toFlowable(LATEST)
request(5)
Flowable
1
2
3
4
5

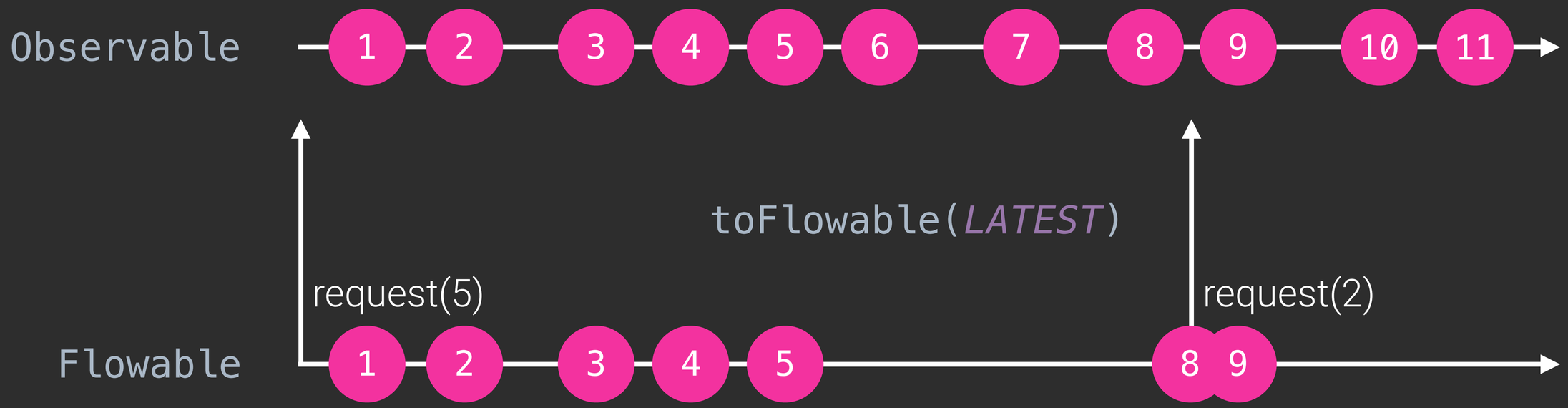
# Converting


Observable
1
2
3
4
5
6
7
8
toFlowable(LATEST)
request(5)
request(2)
Flowable
1
2
3
4
5

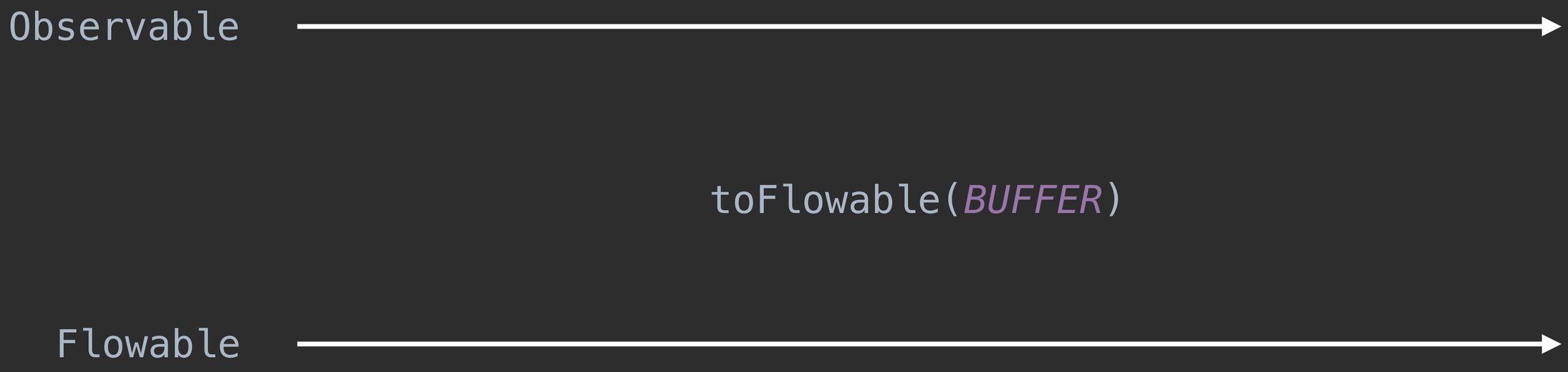
# Converting


Observable
1
2
3
4
5
6
7
8
toFlowable(LATEST)
request(5)
request(2)
Flowable
1
2
3
4
5
8

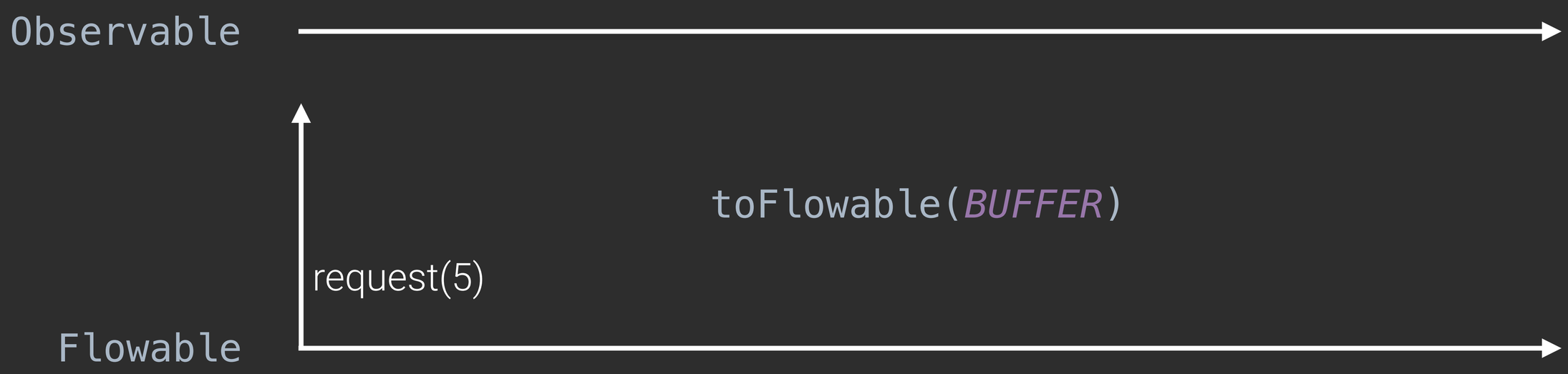
# Converting


Observable
1
2
3
4
5
6
7
8
9
10
11
toFlowable(LATEST)
request(5)
request(2)
Flowable
1
2
3
4
5
8
9

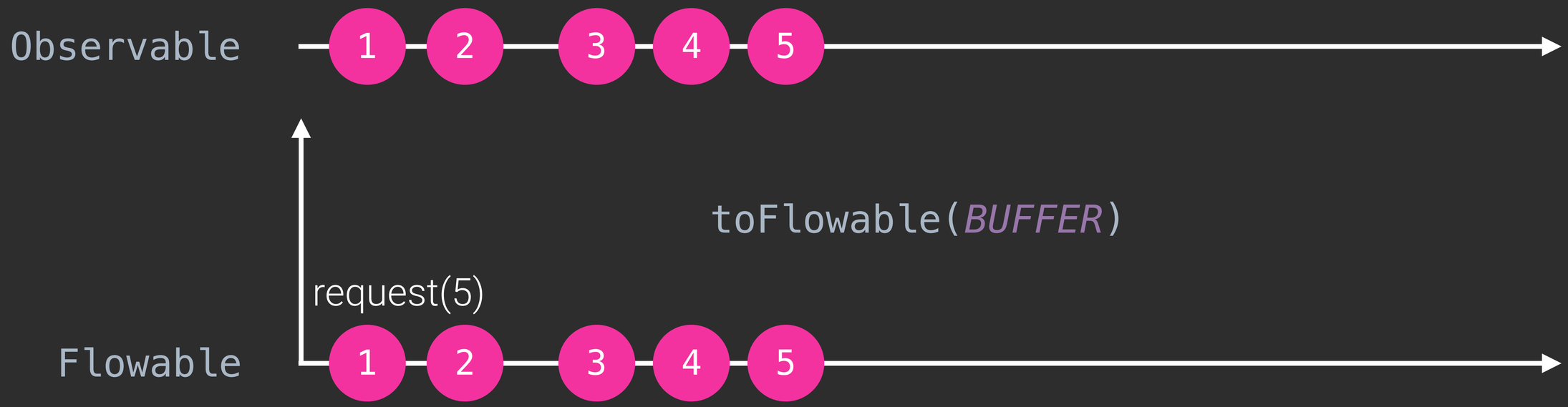
# Converting



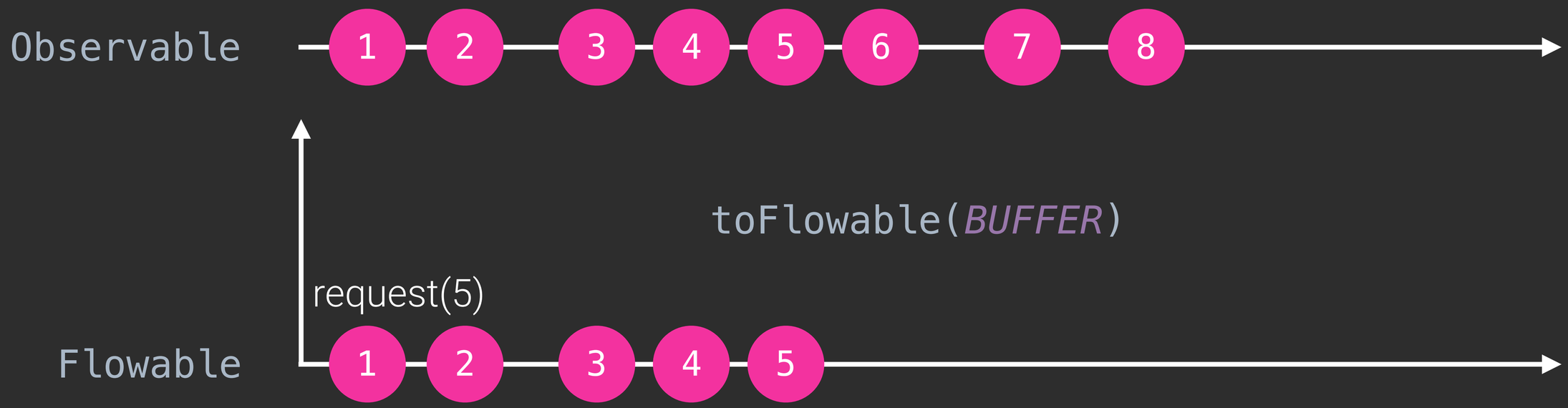
# Converting



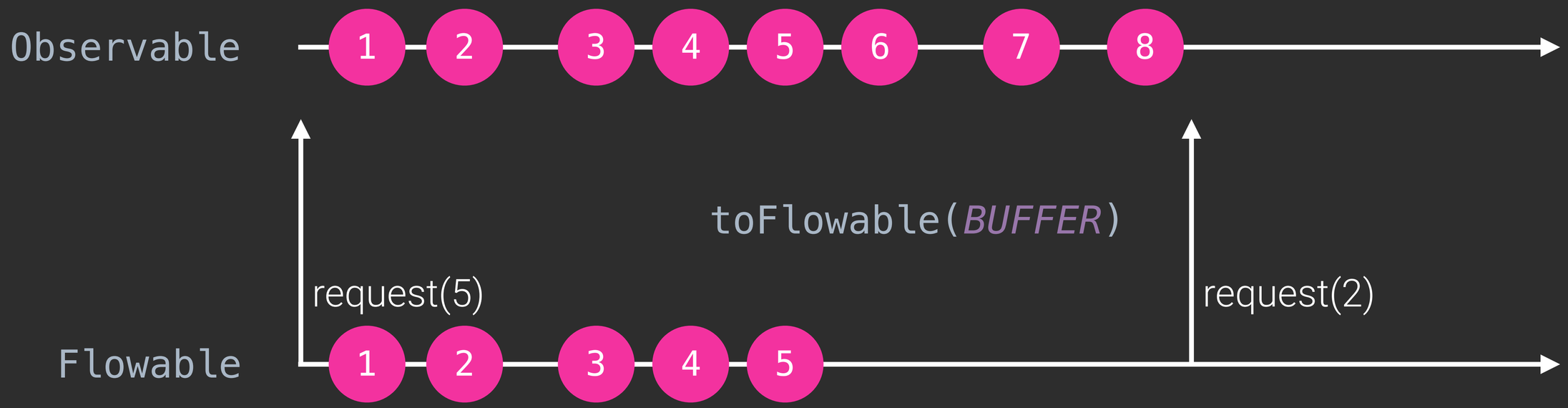
# Converting


Observable
1
2
3
4
5
toFlowable(BUFFER)
request(5)
Flowable
1
2
3
4
5

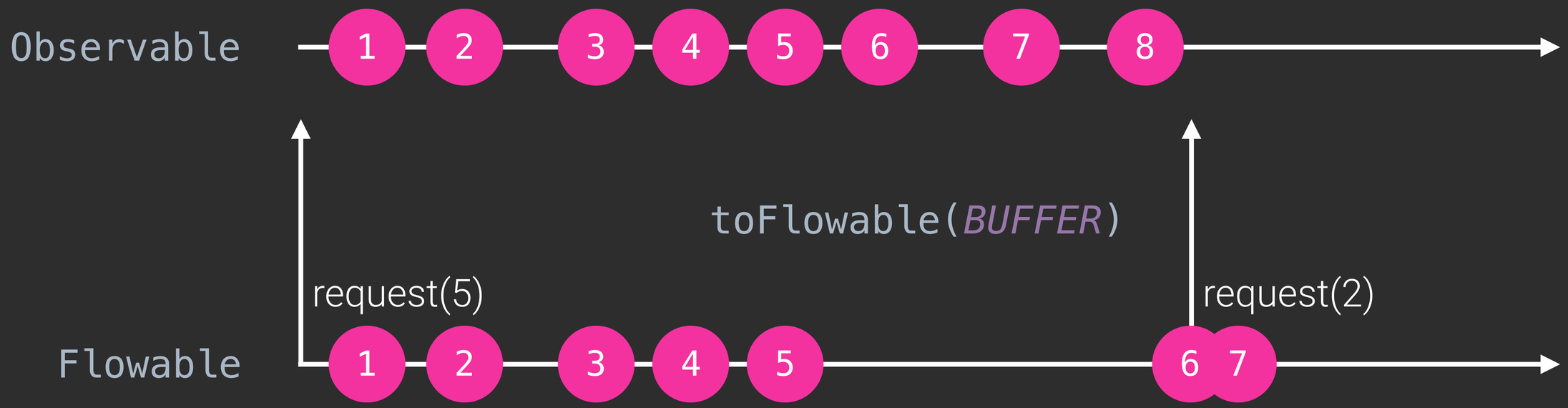
# Converting


Observable
1
2
3
4
5
6
7
8
toFlowable(BUFFER)
request(5)
Flowable
1
2
3
4
5

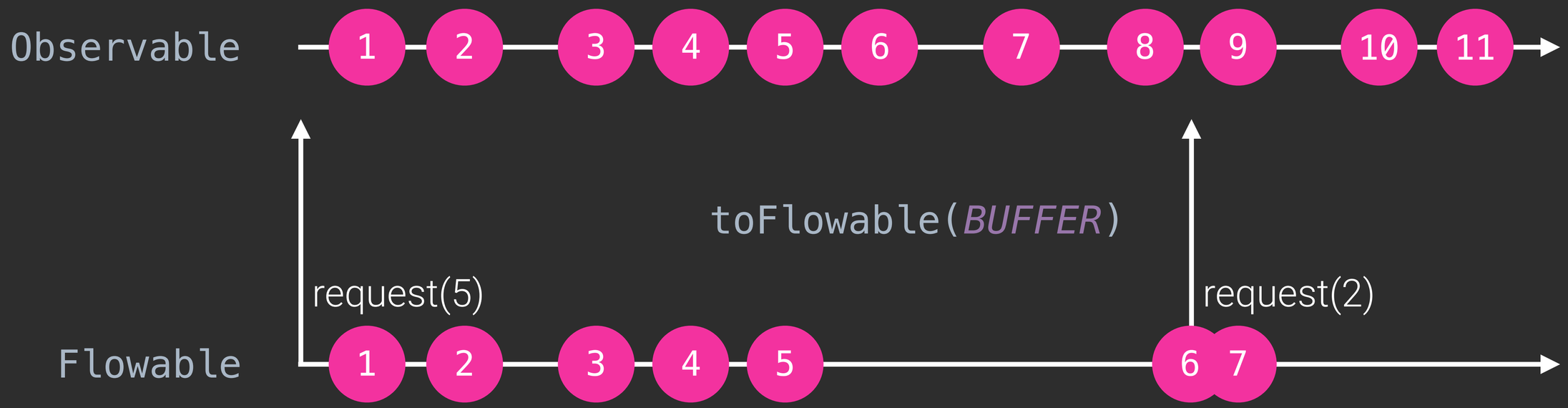
# Converting


Observable
1
2
3
4
5
6
7
8
toFlowable(BUFFER)
request(5)
request(2)
Flowable
1
2
3
4
5

# Converting


Observable
1
2
3
4
5
6
7
8
toFlowable(BUFFER)
request(5)
request(2)
Flowable
1
2
3
4
5
6
7

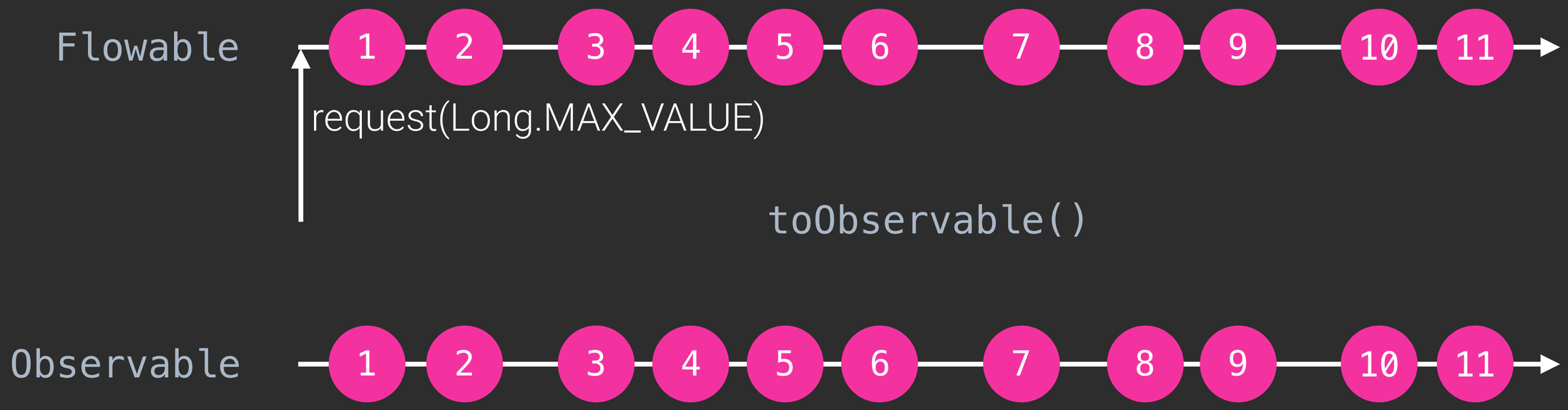
# Converting


Observable
1
2
3
4
5
6
7
8
9
10
11
toFlowable(BUFFER)
request(5)
request(2)
Flowable
1
2
3
4
5
6
7

# Converting

# Converting

# Converting


Flowable
1
2
3
4
5
6
7
8
9
10
11
request(Long.MAX_VALUE)
toObservable()
Observable
1
2
3
4
5
6
7
8
9
10
11

# Specializations

# Specializations

- Encoding subsets of `Observable` into the type system.

# Single

# Single

- Either succeeds with an item or errors.
- No backpressure support.
- Convenience methods for transforming and composing data.

# Single

- Either succeeds with an item or errors.
- No backpressure support.
- Convenience methods for transforming and composing data.
- Think "reactive scalar".

# Completable

# Completable

- Either completes or errors. Has no items!
- No backpressure support.
- Convenience methods for transforming and composing data.

# Completable

- Either completes or errors. Has no items!
- No backpressure support.
- Convenience methods for transforming and composing data.
- Think "reactive runnable".

# Maybe

# Maybe

# Maybe

- Either succeeds with an item, completes with no items, or errors.
- No backpressure support.
- Convenience methods for transforming and composing data.

# Maybe

- Either succeeds with an item, completes with no items, or errors.
- No backpressure support.
- Convenience methods for transforming and composing data.
- Think "reactive optional".

# Specializations

- Encoding subsets of `Observable` into the type system.
- `Single` – Item or error. Think "scalar".
- `Completable` – Complete or error. Think "runnable".
- `Maybe` – Item, complete, or error. Think "optional".

| | Backpressure<br>& Reactive Streams | No Backpressure |
|---|---|---|
| 0…n items, complete\|error | `Flowable` | `Observable` |
| item\|complete\|error | | `Maybe` |
| item\|error | | `Single` |
| complete\|error | | `Completable` |
| Multicast | `FlowableProcessor` | `Subject` |

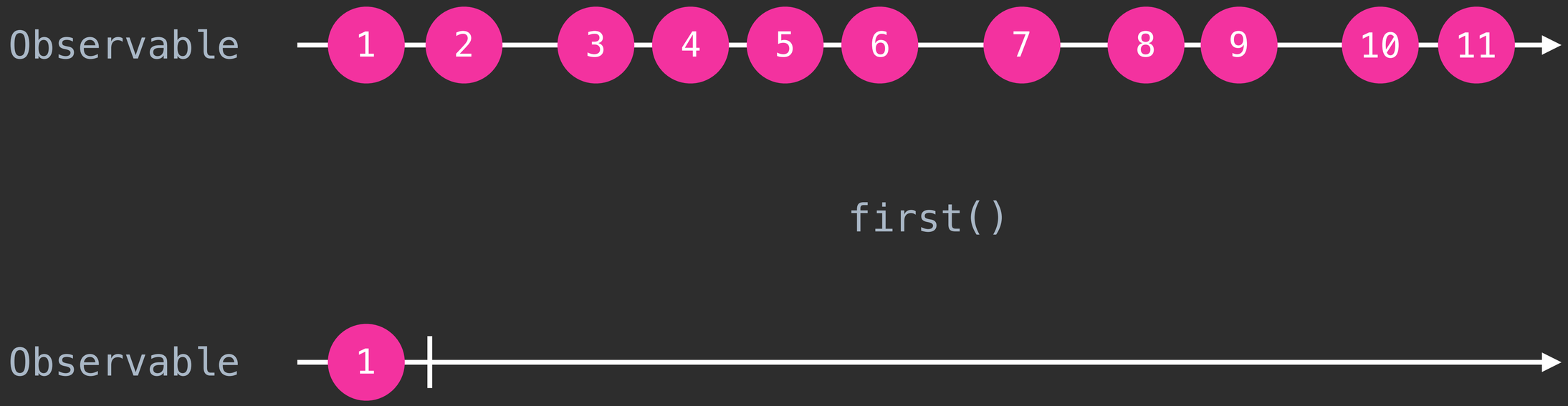
# Operator Specialization

# Operator Specialization


`first()`

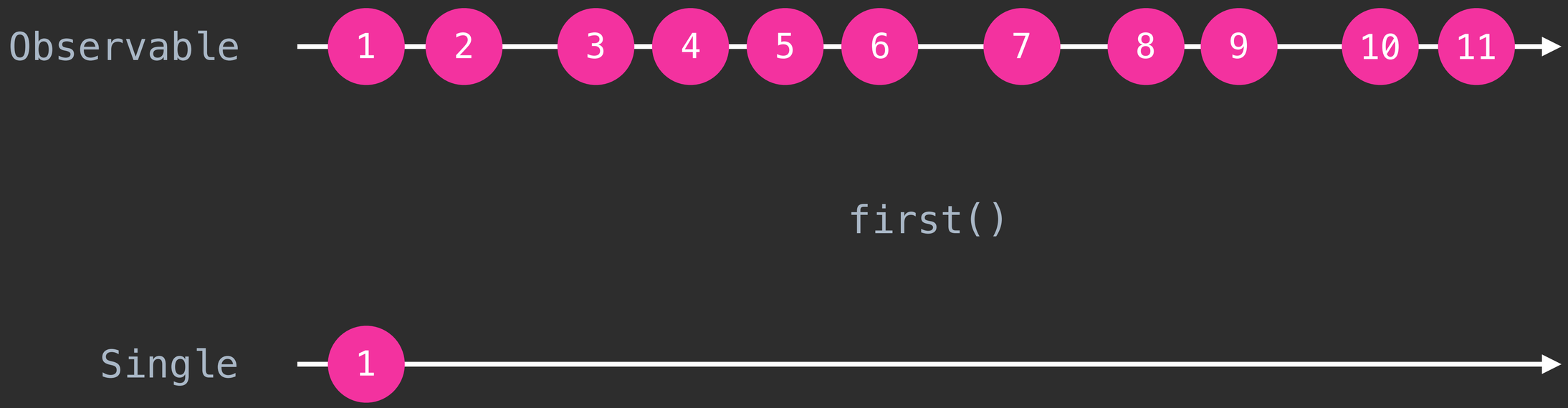
# Operator Specialization

# Operator Specialization

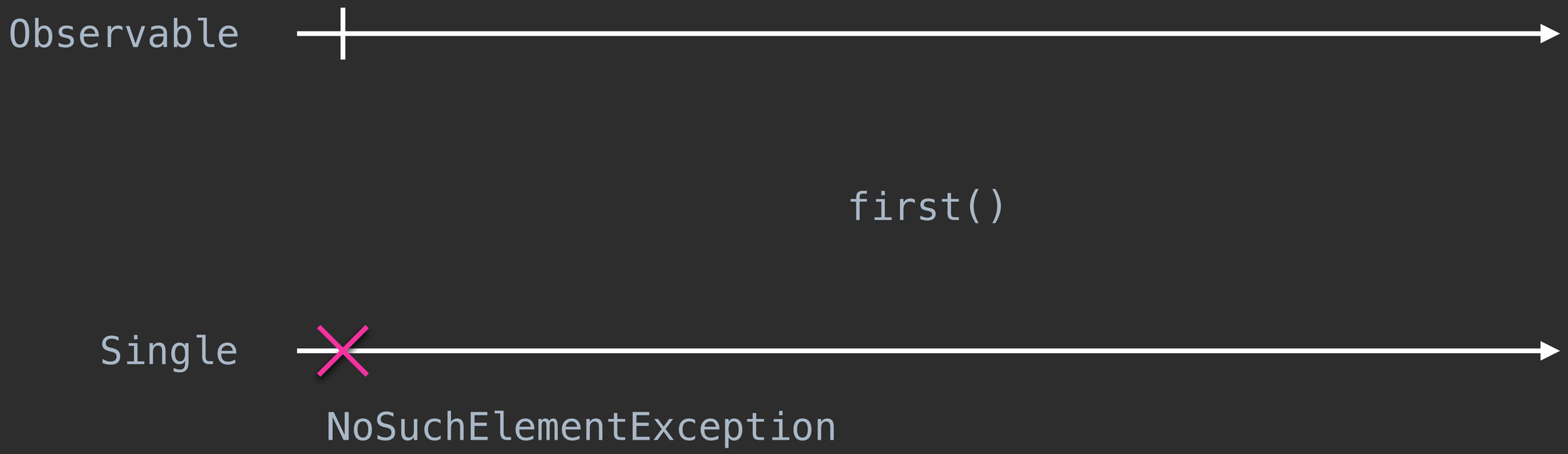


# Operator Specialization

# Operator Specialization



# Operator Specialization

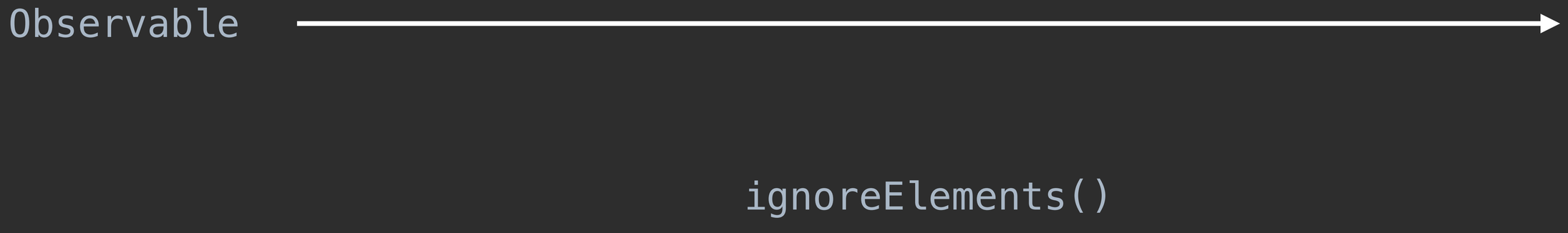


# Operator Specialization

# Operator Specialization

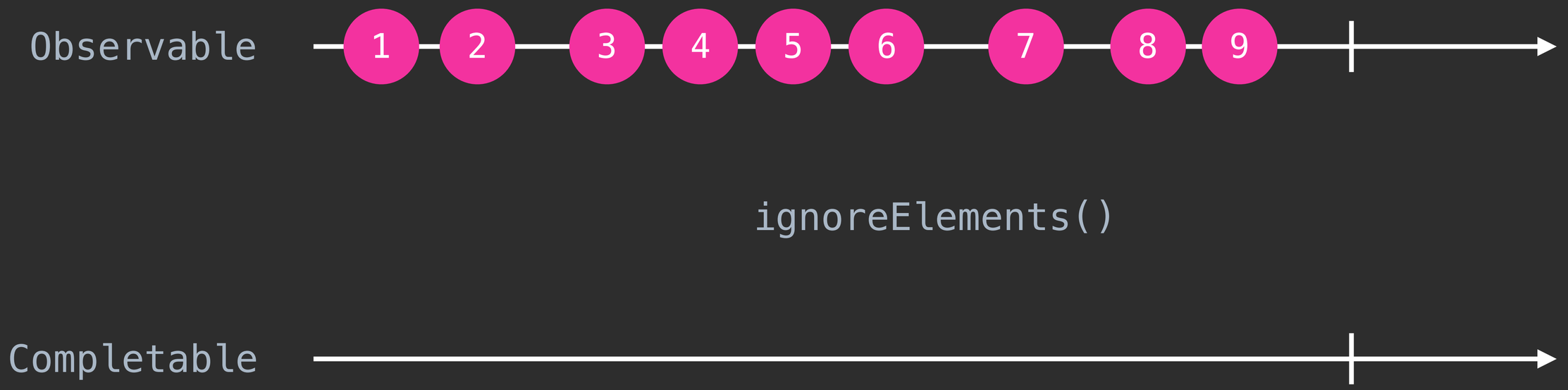
# Operator Specialization

# Operator Specialization



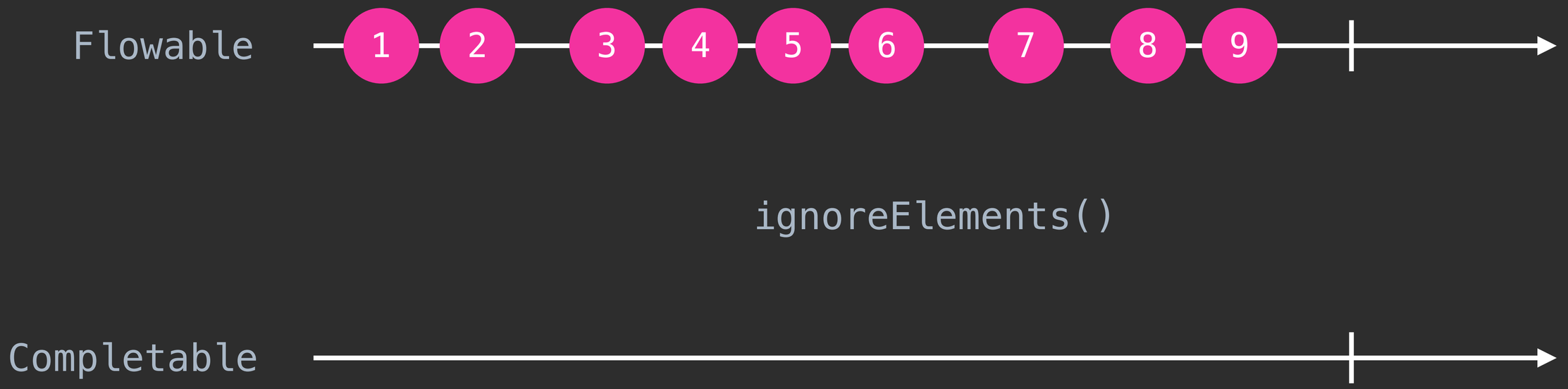
# Operator Specialization

# Operator Specialization



# Operator Specialization

# Operator Specialization



# Operator Specialization

# Operator Specialization

| From \ To | Flowable | Observable | Maybe | Single | Completable |
|---|---|---|---|---|---|
| Flowable | | toObservable() | reduce()<br>elementAt()<br>firstElement()<br>lastElement()<br>singleElement() | scan()<br>elementAt()<br>first()/firstOrError()<br>last()/lastOrError()<br>single/singleOrError()<br>all()/any()/count()<br>(and more) | ignoreElements() |
| Observable | toFlowable() | | reduce()<br>elementAt()<br>firstElement()<br>lastElement()<br>singleElement() | scan()<br>elementAt()<br>first()/firstOrError()<br>last()/lastOrError()<br>single/singleOrError()<br>all()/any()/count()<br>(and more) | ignoreElements() |
| Maybe | toFlowable() | toObservable() | | toSingle()<br>sequenceEqual() | toCompletable() |
| Single | toFlowable() | toObservable() | toMaybe() | | toCompletable() |
| Completable | toFlowable() | toObservable() | toMaybe() | toSingle()<br>toSingleDefault() | |

| From \ To | Flowable | Observable | Maybe | Single | Completable |
|---|---|---|---|---|---|
| Flowable | | toObservable() | reduce()<br>elementAt()<br>firstElement()<br>lastElement()<br>singleElement() | scan()<br>elementAt()<br>first()/firstOrError()<br>last()/lastOrError()<br>single/singleOrError()<br>all()/any()/count()<br>(and more) | ignoreElements() |
| Observable | toFlowable() | | reduce()<br>elementAt()<br>firstElement()<br>lastElement()<br>singleElement() | scan()<br>elementAt()<br>first()/firstOrError()<br>last()/lastOrError()<br>single/singleOrError()<br>all()/any()/count()<br>(and more) | ignoreElements() |
| Maybe | toFlowable() | toObservable() | | toSingle()<br>sequenceEqual() | toCompletable() |
| Single | toFlowable() | toObservable() | toMaybe() | | toCompletable() |
| Completable | toFlowable() | toObservable() | toMaybe() | toSingle()<br>toSingleDefault() | |

| From \ To | Flowable | Observable | Maybe | Single | Completable |
|---|---|---|---|---|---|
| Flowable | | toObservable() | reduce()<br>elementAt()<br>firstElement()<br>lastElement()<br>singleElement() | scan()<br>elementAt()<br>first()/firstOrError()<br>last()/lastOrError()<br>single/singleOrError()<br>all()/any()/count()<br>(and more) | ignoreElements() |
| Observable | toFlowable() | | reduce()<br>elementAt()<br>firstElement()<br>lastElement()<br>singleElement() | scan()<br>elementAt()<br>first()/firstOrError()<br>last()/lastOrError()<br>single/singleOrError()<br>all()/any()/count()<br>(and more) | ignoreElements() |
| Maybe | toFlowable() | toObservable() | | toSingle()<br>sequenceEqual() | toCompletable() |
| Single | toFlowable() | toObservable() | toMaybe() | | toCompletable() |
| Completable | toFlowable() | toObservable() | toMaybe() | toSingle()<br>toSingleDefault() | |

# Creating Sources

# Creating Sources

```
Flowable.just("Hello");
Flowable.just("Hello", "World");

Observable.just("Hello");
Observable.just("Hello", "World");

Maybe.just("Hello");

Single.just("Hello");
```

# Creating Sources

```
String[] array = { "Hello", "World" };
List<String> list = Arrays.asList(array);

Flowable.fromArray(array);
Flowable.fromIterable(list);

Observable.fromArray(array);
Observable.fromIterable(list);
```

# Creating Sources

```
Future<String> future = CompletableFuture.completedFuture("Hello");

Flowable.fromFuture(future);

Observable.fromFuture(future);
```

# Creating Sources

```
Flowable.empty();

Observable.empty();

Maybe.empty();

Completable.complete();
```

# Creating Sources

```
Flowable.never();

Observable.never();

Maybe.never();

Single.never();

Completable.never();
```

# Creating Sources

```
Flowable.error(new RuntimeException());

Observable.error(new RuntimeException());

Maybe.error(new RuntimeException());

Single.error(new RuntimeException());

Completable.error(new RuntimeException());
```

# Creating Sources

```
Observable.fromCallable(new Callable<String>() {
  @Override public String call() {
    return getName();
  }
});
```

# Creating Sources

```
Observable.fromCallable(new Callable<String>() {
  @Override public String call() throws Exception {
    return getName();
  }
});
```

# Creating Sources

```
OkHttpClient client = // …
Request request = // …

Observable.fromCallable(new Callable<String>() {
  @Override public String call() throws Exception {
    return client.newCall(request).execute();
  }
});
```

# Creating Sources

```
Flowable.fromCallable(() -> "Hello");

Observable.fromCallable(() -> "Hello");

Maybe.fromCallable(() -> "Hello");
Maybe.fromAction(() -> System.out.println("Hello"));
Maybe.fromRunnable(() -> System.out.println("Hello"))

Single.fromCallable(() -> "Hello");

Completable.fromCallable(() -> "Ignored!");
Completable.fromAction(() -> System.out.println("Hello"));
Completable.fromRunnable(() -> System.out.println("Hello"));
```

# Creating Sources

```
Observable.create();
```

# Creating Sources

```
Observable.create();
```

# Creating Sources

```
Observable.create(new ObservableOnSubscribe<String>() {
  @Override
  public void subscribe(ObservableEmitter<String> e) throws Exception {
    e.onNext("Hello");
    e.onComplete();
  }
});
```

# Creating Sources

```
Observable.create(new ObservableOnSubscribe<String>() {
  @Override
  public void subscribe(ObservableEmitter<String> e) throws Exception {
    e.onNext("Hello");
    e.onComplete();
  }
});
```

# Creating Sources

```
Observable.create(new ObservableOnSubscribe<String>() {
  @Override
  public void subscribe(ObservableEmitter<String> e) throws Exception {
    e.onNext("Hello");
    e.onComplete();
  }
});
```

# Creating Sources

```
Observable.create(e -> {
  e.onNext("Hello");
  e.onComplete();
});
```

# Creating Sources

```
Observable.create(e -> {
  e.onNext("Hello");
  e.onNext("World");
  e.onComplete();
});
```

# Creating Sources

```
OkHttpClient client = // …
Request request = // …

Observable.create(e -> {
  client.newCall(request).enqueue(new Callback() {
    @Override public void onResponse(Response r) throws IOException {
      e.onNext(r.body().string());
      e.onComplete();
    }
    @Override public void onFailure(IOException e) {
      e.onError(e);
    }
  });
});
```

# Creating Sources

```
OkHttpClient client = // …
Request request = // …

Observable.create(e -> {
  Call call = client.newCall(request);
  call.enqueue(new Callback() {
    @Override public void onResponse(Response r) throws IOException {
      e.onNext(r.body().string());
      e.onComplete();
    }
    @Override public void onFailure(IOException e) {
      e.onError(e);
    }
  });
});
```

# Creating Sources

```
OkHttpClient client = // …
Request request = // …

Observable.create(e -> {
  Call call = client.newCall(request);
  e.setCancelation(() -> call.cancel());
  call.enqueue(new Callback() {
    @Override public void onResponse(Response r) throws IOException {
      e.onNext(r.body().string());
      e.onComplete();
    }
    @Override public void onFailure(IOException e) {
      e.onError(e);
    }
  });
});
```

# Creating Sources

```
OkHttpClient client = // …
Request request = // …

Observable.create(e -> {
  Call call = client.newCall(request);
  e.setCancelation(() -> call.cancel());
  call.enqueue(new Callback() {
    @Override public void onResponse(Response r) throws IOException {
      e.onNext(r.body().string());
      e.onComplete();
    }
    @Override public void onFailure(IOException e) {
      e.onError(e);
    }
  });
});
```

# Creating Sources

```
View view = // …

Observable.create(e -> {
  e.setCancelation(() -> view.setOnClickListener(null));
  view.setOnClickListener(v -> e.onNext(v));
});
```

# Creating Sources

```
Flowable.create(e -> { … });

Observable.create(e -> { … });

Maybe.create(e -> { … });

Single.create(e -> { … });

Completable.create(e -> { … });
```

# Observing Sources

# Observing Sources

Observable<String>

```
interface Observer<T> {
  void onNext(T t);
  void onComplete();
  void onError(Throwable t);
  void onSubscribe(Disposable d);
}
```

Flowable<String>

```
interface Subscriber<T> {
  void onNext(T t);
  void onComplete();
  void onError(Throwable t);
  void onSubscribe(Subscription s);
}
```

# Observing Sources

```
Observable<String>


interface Observer<T> {
  void onNext(T t);
  void onComplete();
  void onError(Throwable t);
  void onSubscribe(Disposable d);
}


interface Disposable {
  void dispose();
}
```

```
Flowable<String>


interface Subscriber<T> {
  void onNext(T t);
  void onComplete();
  void onError(Throwable t);
  void onSubscribe(Subscription s);
}


interface Subscription {
  void cancel();
  void request(long r);
}
```

# Observing Sources

```
Observable<String> o = Observable.just("Hello");

o.subscribe(new Observer<String>() {
  @Override public void onNext(String s) { … }
  @Override public void onComplete() { … }
  @Override public void onError(Throwable t) { … }

  @Override public void onSubscribe(Disposable d) {
    ???
  }
});
```

# Observing Sources

```
Observable<String> o = Observable.just("Hello");

o.subscribe(new DisposableObserver<String>() {
  @Override public void onNext(String s) { … }
  @Override public void onComplete() { … }
  @Override public void onError(Throwable t) { … }
});
```

# Observing Sources

```
Observable<String> o = Observable.just("Hello");

o.subscribe(new DisposableObserver<String>() {
  @Override public void onNext(String s) { … }
  @Override public void onComplete() { … }
  @Override public void onError(Throwable t) { … }
});

unsubscribe???
```

# Observing Sources

```
Observable<String> o = Observable.just("Hello");

DisposableObserver observer = new DisposableObserver<String>() {
  @Override public void onNext(String s) { … }
  @Override public void onComplete() { … }
  @Override public void onError(Throwable t) { … }
}
o.subscribe(observer);
```

# Observing Sources

```
Observable<String> o = Observable.just("Hello");

DisposableObserver observer = new DisposableObserver<String>() {
  @Override public void onNext(String s) { … }
  @Override public void onComplete() { … }
  @Override public void onError(Throwable t) { … }
}
o.subscribe(observer);

observer.dispose();
```

# Observing Sources

```
Observable<String> o = Observable.just("Hello");

o.subscribe(new DisposableObserver<String>() {
  @Override public void onNext(String s) { … }
  @Override public void onComplete() { … }
  @Override public void onError(Throwable t) { … }
});
```

# Observing Sources

```
Observable<String> o = Observable.just("Hello");

o.subscribeWith(new DisposableObserver<String>() {
  @Override public void onNext(String s) { … }
  @Override public void onComplete() { … }
  @Override public void onError(Throwable t) { … }
});
```

# Observing Sources

```
Observable<String> o = Observable.just("Hello");

Disposable d = o.subscribeWith(new DisposableObserver<String>() {
  @Override public void onNext(String s) { … }
  @Override public void onComplete() { … }
  @Override public void onError(Throwable t) { … }
});

d.dispose();
```

# Observing Sources

```
Observable<String> o = Observable.just("Hello");

CompositeDisposable disposables = new CompositeDisposable();

Disposable d = o.subscribeWith(new DisposableObserver<String>() {
  @Override public void onNext(String s) { … }
  @Override public void onComplete() { … }
  @Override public void onError(Throwable t) { … }
});

disposables.add(d);
```

# Observing Sources

```
Observable<String> o = Observable.just("Hello");

CompositeDisposable disposables = new CompositeDisposable();

Disposable d = o.subscribeWith(new DisposableObserver<String>() {
  @Override public void onNext(String s) { … }
  @Override public void onComplete() { … }
  @Override public void onError(Throwable t) { … }
});

disposables.add(d);

disposables.dispose();
```

# Observing Sources

```
Observable<String> o = Observable.just("Hello");
o.subscribeWith(new DisposableObserver<String>() { … });

Maybe<String> m = Maybe.just("Hello");
m.subscribeWith(new DisposableMaybeObserver<String>() { … });

Single<String> s = Single.just("Hello");
s.subscribeWith(new DisposableSingleObserver<String>() { … });

Completable c = Completable.completed();
c.subscribeWith(new DisposableCompletableObserver<String>() { … });
```

# Observing Sources

```
Flowable<String> f = Flowable.just("Hello");
f.subscribeWith(new DisposableSubscriber<String>() { … });

Observable<String> o = Observable.just("Hello");
o.subscribeWith(new DisposableObserver<String>() { … });

Maybe<String> m = Maybe.just("Hello");
m.subscribeWith(new DisposableMaybeObserver<String>() { … });

Single<String> s = Single.just("Hello");
s.subscribeWith(new DisposableSingleObserver<String>() { … });

Completable c = Completable.completed();
c.subscribeWith(new DisposableCompletableObserver<String>() { … });
```

# Observing Sources

```
Flowable<String> f = Flowable.just("Hello");
f.subscribeWith(new DisposableSubscriber<String>() { … });

Observable<String> o = Observable.just("Hello");
o.subscribeWith(new DisposableObserver<String>() { … });

Maybe<String> m = Maybe.just("Hello");
m.subscribeWith(new DisposableMaybeObserver<String>() { … });

Single<String> s = Single.just("Hello");
s.subscribeWith(new DisposableSingleObserver<String>() { … });

Completable c = Completable.completed();
c.subscribeWith(new DisposableCompletableObserver<String>() { … });
```

# Observing Sources

```
Flowable<String> f = Flowable.just("Hello");
Disposable d1 = f.subscribeWith(new DisposableSubscriber<String>() { … });

Observable<String> o = Observable.just("Hello");
Disposable d2 = o.subscribeWith(new DisposableObserver<String>() { … });

Maybe<String> m = Maybe.just("Hello");
Disposable d3 = m.subscribeWith(new DisposableMaybeObserver<String>() { … });

Single<String> s = Single.just("Hello");
Disposable d4 = s.subscribeWith(new DisposableSingleObserver<String>() { … });

Completable c = Completable.completed();
Disposable d5 = c.subscribeWith(new DisposableCompletableObserver<String>() { … });
```

# RxJava

# RxJava

- 2.0.0-RC3 was released 2016-09-23.

# RxJava

- 2.0.0-RC3 was released 2016-09-23.
- Releases are more like developer previews than release candidates.

# RxJava

- 2.0.0-RC3 was released 2016-09-23.
- Releases are more like developer previews than release candidates.
- Next two are scheduled for 2016-10-07 and 2016-10-21.

# RxJava

- 2.0.0-RC3 was released 2016-09-23.
- Releases are more like developer previews than release candidates.
- Next two are scheduled for 2016-10-07 and 2016-10-21.
- Final release scheduled for 2016-10-29.

# RxJava

- Package name is `io.reactivex.*`.

# RxJava

- Package name is `io.reactivex.*`.
- Maven coordinates are `io.reactivex.rxjava2:rxjava`.

# RxJava

- Package name is `io.reactivex.*`.
- Maven coordinates are `io.reactivex.rxjava2:rxjava`.
- Why the change? See http://jakes.link/major-versions

# RxAndroid

# RxAndroid

- 2.0.0-RC1 was released 2016-08-25.

# RxAndroid

- 2.0.0-RC1 was released 2016-08-25.
- Provides facilities for dealing with the main thread (or any `Looper`).

# RxAndroid

- 2.0.0-RC1 was released 2016-08-25.
- Provides facilities for dealing with the main thread (or any `Looper`).
- Package name is `io.reactivex.android.*`.

# RxAndroid

- 2.0.0-RC1 was released 2016-08-25.
- Provides facilities for dealing with the main thread (or any `Looper`).
- Package name is `io.reactivex.android.*`.
- Maven coordinates are `io.reactivex.rxjava2:rxandroid`.

# Living with 1.x and 2.x

# Living with 1.x and 2.x

```
class RxJavaInterop {
  static <T> Flowable<T> toV2Flowable(rx.Observable<T> o) { … }
  static <T> Observable<T> toV2Observable(rx.Observable<T> o) { … }
  static <T> Maybe<T> toV2Maybe(rx.Single<T> s) { … }
  static <T> Maybe<T> toV2Maybe(rx.Completable c) { … }
  static <T> Single<T> toV2Single(rx.Single<T> s) { … }
  static Completable toV2Completable(rx.Completable c) { … }

  static <T> rx.Observable<T> toV1Observable(Publisher<T> p) { … }
  static <T> rx.Observable<T> toV1Observable(Observable<T> o, …) { … }
  static <T> rx.Single<T> toV1Single(Single<T> o) { … }
  static <T> rx.Single<T> toV1Single(Maybe<T> m) { … }
  static rx.Completable toV1Completable(Completable c) { … }
  static rx.Completable toV1Completable(Maybe<T> m) { … }
}
```

# Living with 1.x and 2.x

github.com/akarnokd/RxJava2Interop

```
class RxJavaInterop {
  static <T> Flowable<T> toV2Flowable(rx.Observable<T> o) { … }
  static <T> Observable<T> toV2Observable(rx.Observable<T> o) { … }
  static <T> Maybe<T> toV2Maybe(rx.Single<T> s) { … }
  static <T> Maybe<T> toV2Maybe(rx.Completable c) { … }
  static <T> Single<T> toV2Single(rx.Single<T> s) { … }
  static Completable toV2Completable(rx.Completable c) { … }

  static <T> rx.Observable<T> toV1Observable(Publisher<T> p) { … }
  static <T> rx.Observable<T> toV1Observable(Observable<T> o, …) { … }
  static <T> rx.Single<T> toV1Single(Single<T> o) { … }
```

```
dependencies {
  compile 'io.reactivex.rxjava2:rxjava:2.0.0-RC3'
  compile 'io.reactivex.rxjava2:rxandroid:2.0.0-RC1'

  // Optionally...
  compile 'com.github.akarnokd:rxjava2-interop:0.3.0'
}
```

# Other Libraries We Control

# Other Libraries We Control

- Retrofit 2 – github.com/JakeWharton/retrofit2-rxjava2-adapter/

# Other Libraries We Control

- Retrofit 2 – github.com/JakeWharton/retrofit2-rxjava2-adapter/
- RxBinding – waiting for 2.0 final
- RxRelay – waiting for 2.0 final
- RxReplayingShare – waiting for 2.0 final
- SQL Brite – waiting for 2.0 final
- Whorlwind – waiting for 2.0 final

# RxJava 2

# RxJava 2

- Reactive Streams compliant.

# RxJava 2

- Reactive Streams compliant.
- Backpressure in the type system (`Flowable`).

# RxJava 2

- Reactive Streams compliant.
- Backpressure in the type system (`Flowable`).
- New `Maybe` type.

# RxJava 2

- Reactive Streams compliant.
- Backpressure in the type system (`Flowable`).
- New `Maybe` type.
- Actually usable `create()`.

# RxJava 2

- Reactive Streams compliant.
- Backpressure in the type system (`Flowable`).
- New `Maybe` type.
- Actually usable `create()`.
- All the same operators.

# RxJava 2

- Reactive Streams compliant.
- Backpressure in the type system (`Flowable`).
- New `Maybe` type.
- Actually usable `create()`.
- All the same operators.
- Preview releases now, final release in a month.


Looking Ahead To
RxJava 2
twitter.com/ jakewharton
google.com/+ jakewharton
jakewharton .com