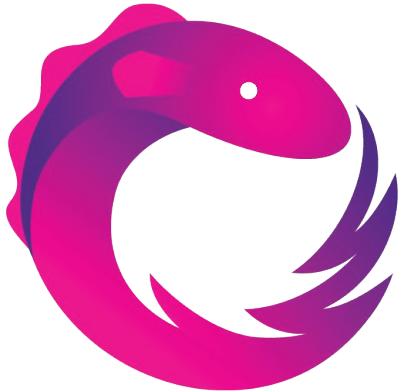


RxJava Advanced Code Lab

https://github.com/jraska/RxJava-Codelab

Josef Raska - Ocean Android team

Agenda

- Motivation
- **Let’s code and discuss!**
- Observable/Single/Maybe/Completable
- Hot/Cold observable
- Backpressure
- ConnectableObservable/Processors, Plugins
- Other Reactive Java libraries

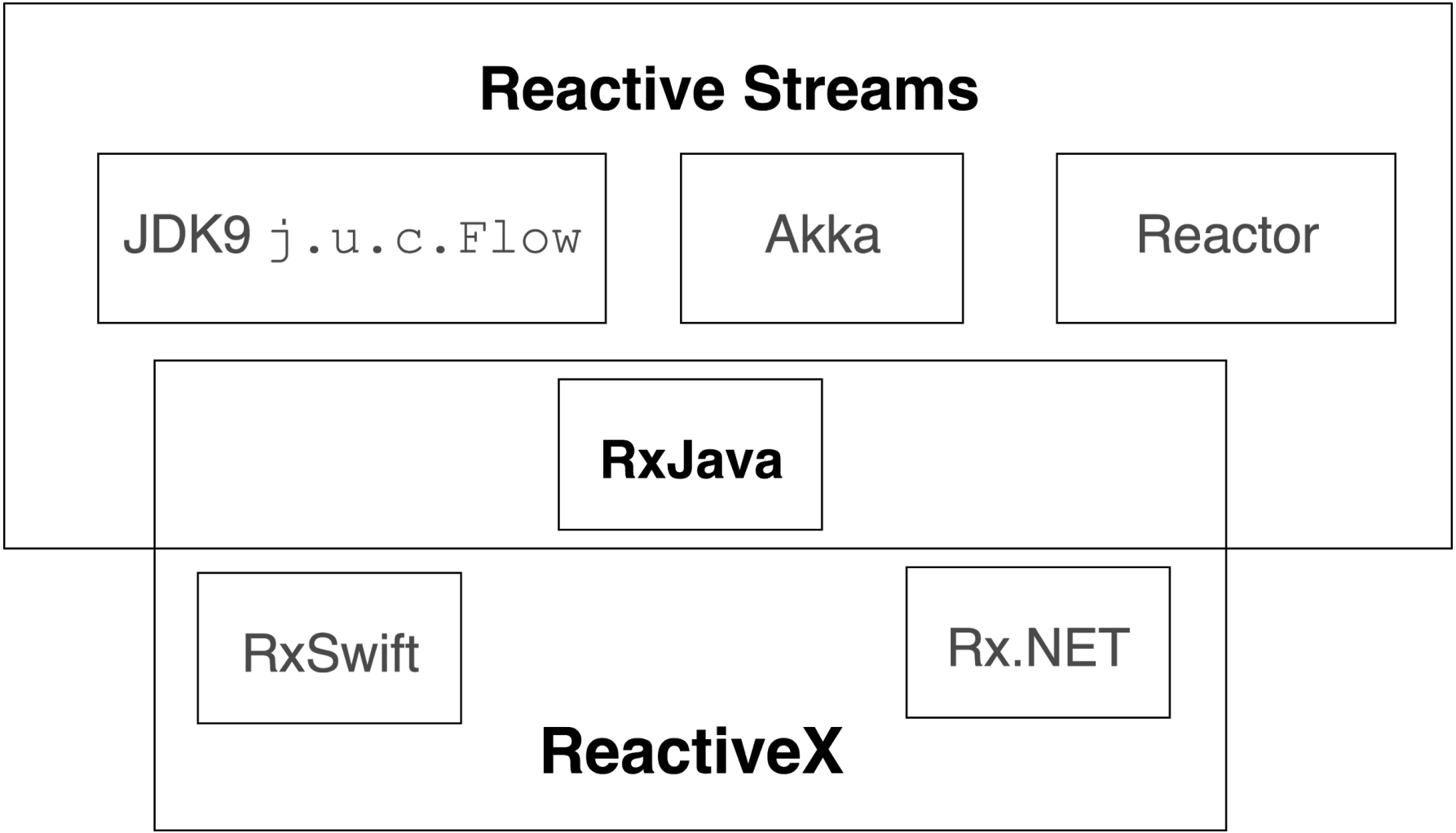
Why this Code Lab?

- Rx is powerful tool
- Rx is trendy
- We use Rx and we want to use it more
- Different way of thinking
- Difficult to understand
- Lot of confusion
- **Advanced? - We want even more power!**

Why Rx?

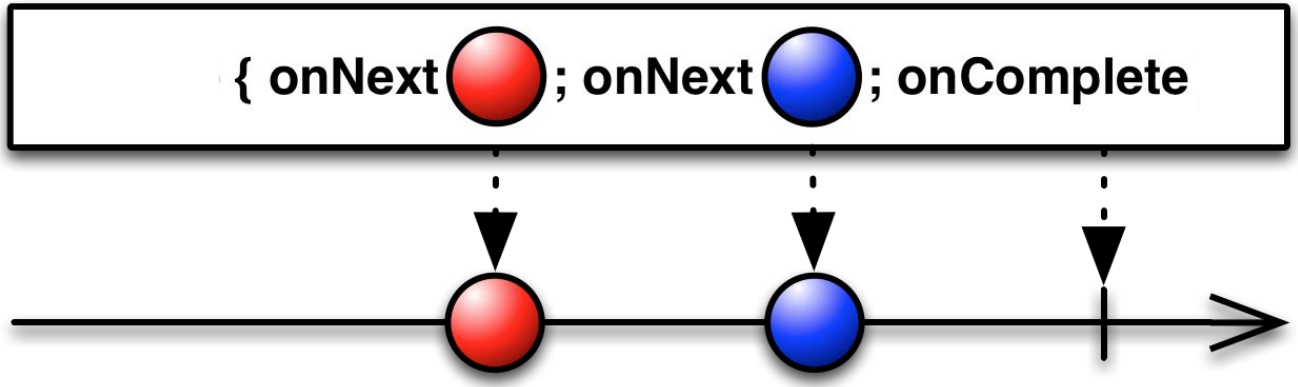
Can you model your whole system synchronously?

- Removes the web of callbacks
- Declarative
- Simple asynchronous composing
- Threading abstraction
- Developer experience matters


Reactive Streams
JDK9 j.u.c.Flow
Akka
Reactor
RxJava
RxSwift
Rx.NET
ReactiveX

Observable<T>

```
onNext(T)
onError(Throwable)
onCompleted()
```



http://rxmarbles.com/

Why Rx?

```
interface UserManager {
  User getUser();
  void setName(String name, Listener callback);
  void setAge(int age, Listener callback);

  interface Listener {
    void success();
    void failure(IOException e);
  }
}
```

↓ ↓ ↓

```
interface UserManager {
  Observable<User> getUser();
  Observable<User> setName(String name);
  Observable<User> setAge(int age);
}
```

Why Rx?

```
UserManager um = new UserManager();
System.out.println(um.getUser());

um.setName( name: "John Doe", new UserManager.Listener() {
  @Override public void success() {
    System.out.println(um.getUser());
  }

  @Override public void failure(IOException e) {
    // TODO: handle error
  }
});

um.setAge( age: 33, new UserManager.Listener() {
  @Override public void success() {
    System.out.println(um.getUser());
  }

  @Override public void failure(IOException e) {
    // TODO: handle error
  }
});
```

Why Rx?

```
UserManager um = new UserManager();

um.setName("John Doe")
  .concatWith(um.setAge(33))
  .subscribe(System.out::println);
```

Why Rx?

```
UserManager um = new UserManager();
System.out.println(um.getUser());

um.setName( name: "John Doe", new UserManager.Listener() {
  @Override public void success() {
    System.out.println(um.getUser());

    um.setAge( age: 33, new UserManager.Listener() {
      @Override public void success() {
        System.out.println(um.getUser());
      }

      @Override public void failure(IOException e) {
        // TODO: handle error
      }
    });
  }

  @Override public void failure(IOException e) {
    // TODO: handle error
  }
});
```

Why Rx?

```
UserManager um = new UserManager();

um.setName("John Doe")
  .flatMap((user) -> um.setAge(33))
  .subscribe(System.out::println);
```

Why Rx?

```
private TextView textView;
private UserManager um = new UserManager();

void updateUserName() {
  um.setName( name: "John Doe", new UserManager.Listener() {
    @Override
    public void success() {
      runOnUiThread(new Runnable() {
        @Override
        public void run() {
          textView.setText(um.getUser().toString());
        }
      });
    }

    @Override
    public void failure(IOException e) {
      // TODO: handle error
    }
  });
```

Why Rx?

```
TextView textView;
UserManager um = new UserManager();

void updateUserName() {
  um.setName("John Doe")
    .subscribeOn(schedulers.io())
    .observeOn(schedulers.mainThread())
    .subscribe(user -> textView.setText(user.toString()));
```

Codelab project

- RxJava 2
- Code playground is unit test
- Separate tasks for particular areas
- Each task has a solution in `solutions` package

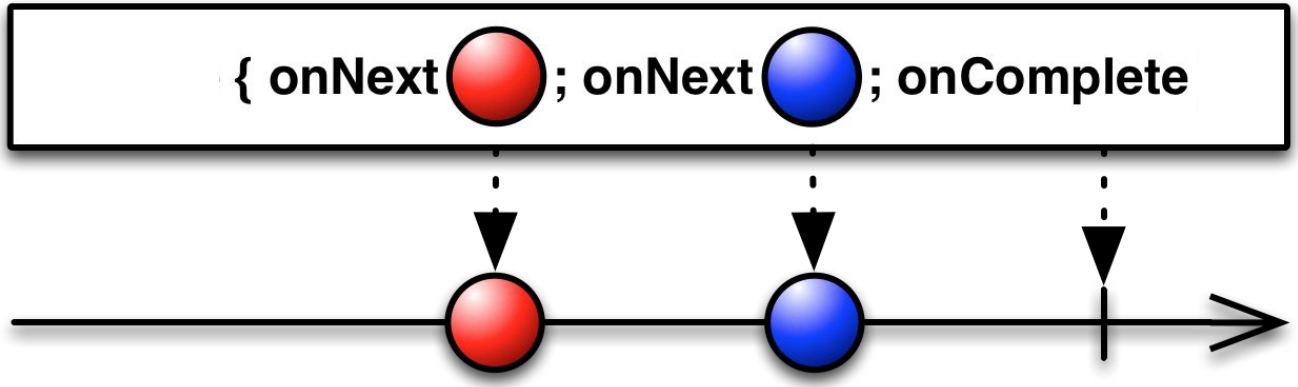
Project setup

- Running unit tests with icon → 

- Is everyone ready?
- Entertainment meanwhile:
  http://reactivex.io/intro.html
  http://rxmarbles.com/

Observable<T>

```
onNext(T)
onError(Throwable)
onCompleted()
```



http://rxmarbles.com/

Observable feat. Single, Maybe, Completable

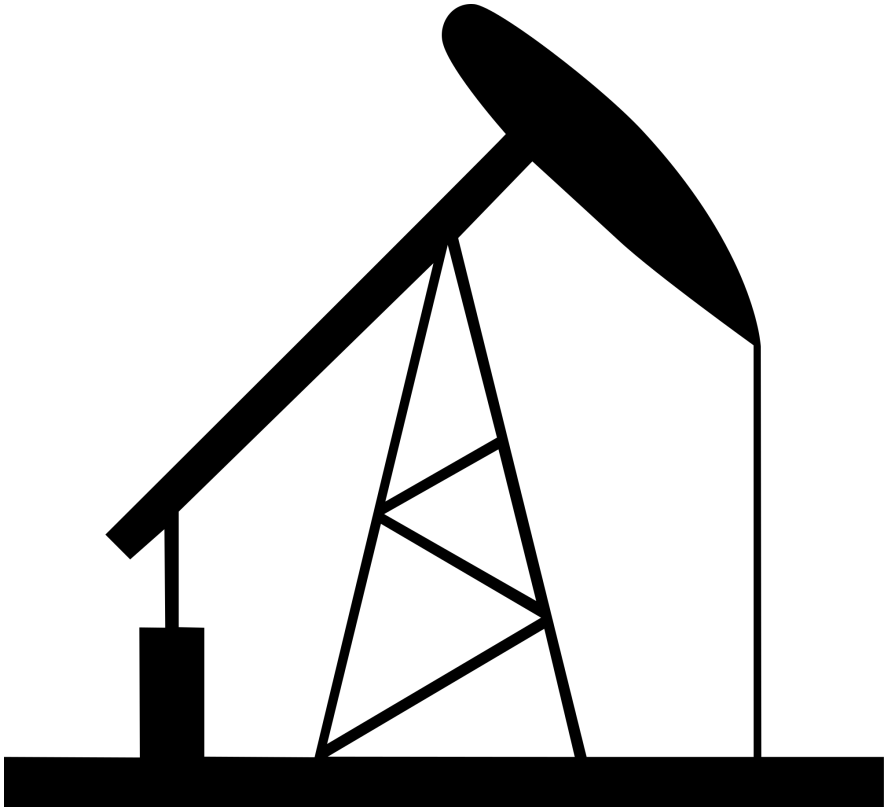
emits:	**0**	**0 or 1**	**1**	**0 .. N**
Sync	`void`	`Optional<T>`	`T`	`Iterable<T>`
Async	`Completable`	`Maybe<T>`	`Single<T>`	`Observable<T>`

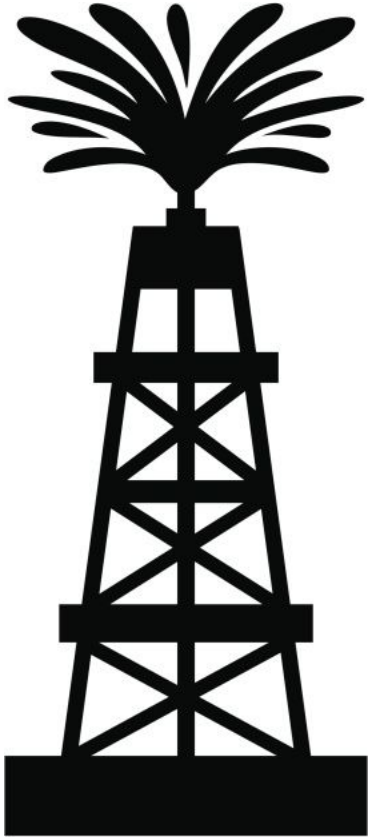
- **Producer**
- Gets values and pass them to `observer.next(value)`
- Hot producer is active regardless the subscriptions
- Cold producer gets activated by subscription

Emits whether the observer is ready or not	Emits at controlled rate whenever requested by observer
Mouse & keyboard events System events Stock prices Time	Iterable Web request Database query Reading file

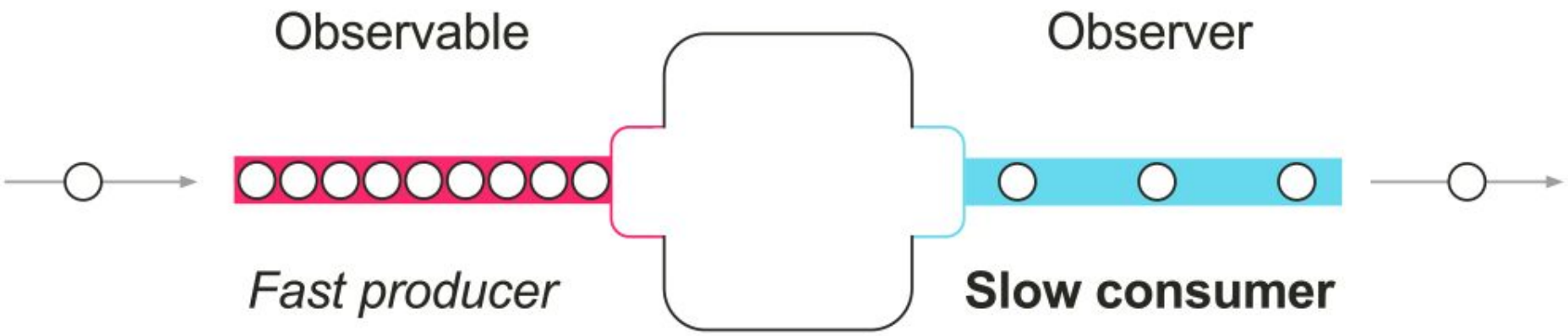
River Thames - Hot Observable

Oil Well - Cold Observable



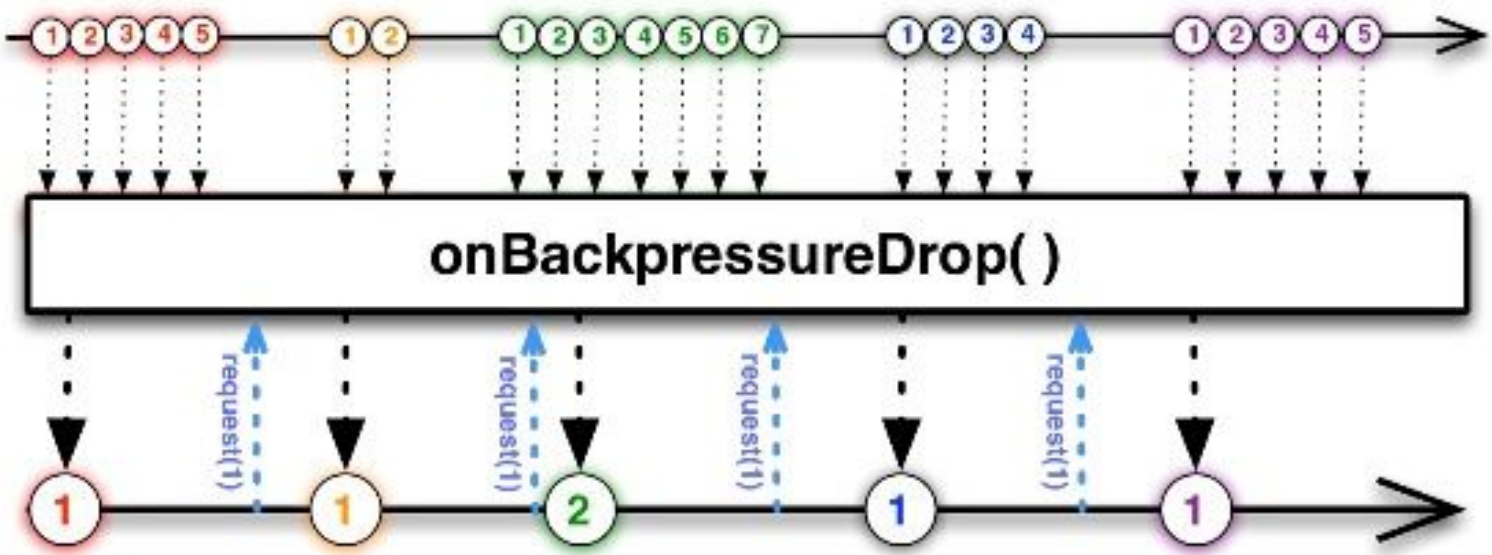


Backpressure

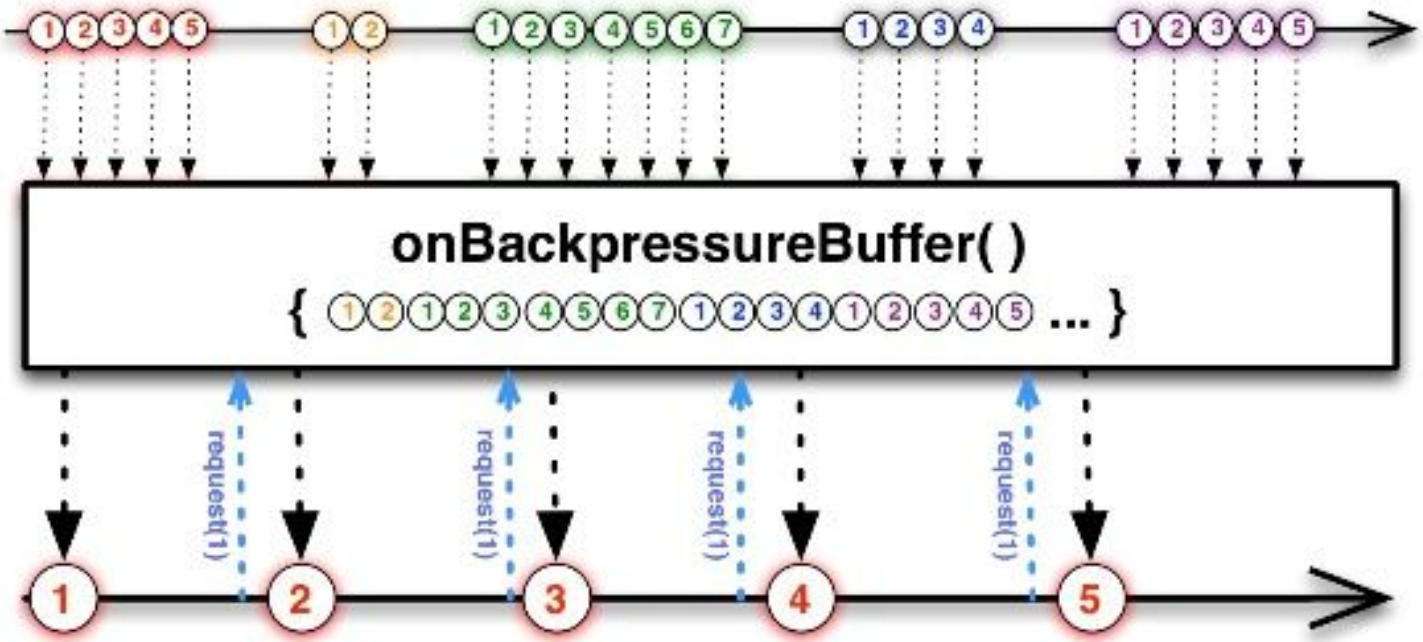


MissingBackpressureException

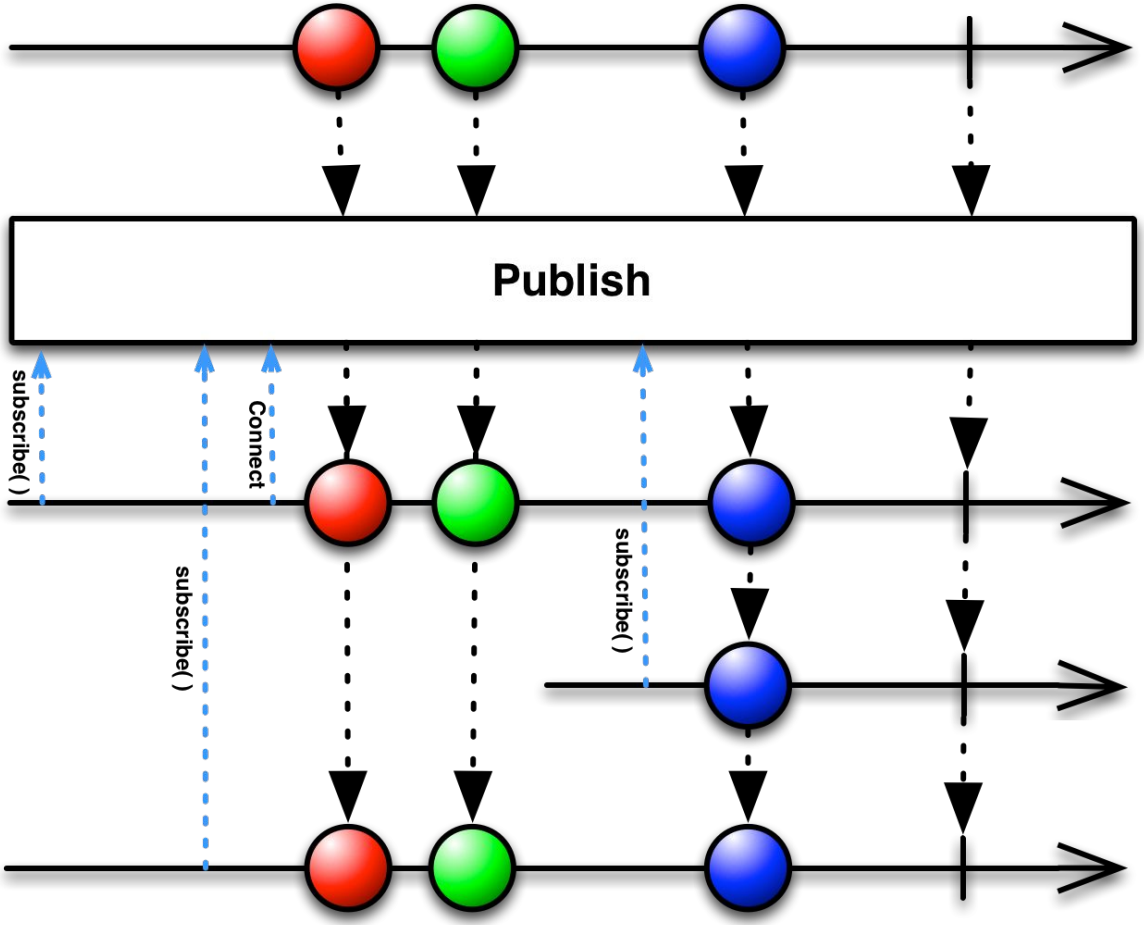
On backpressure Drop

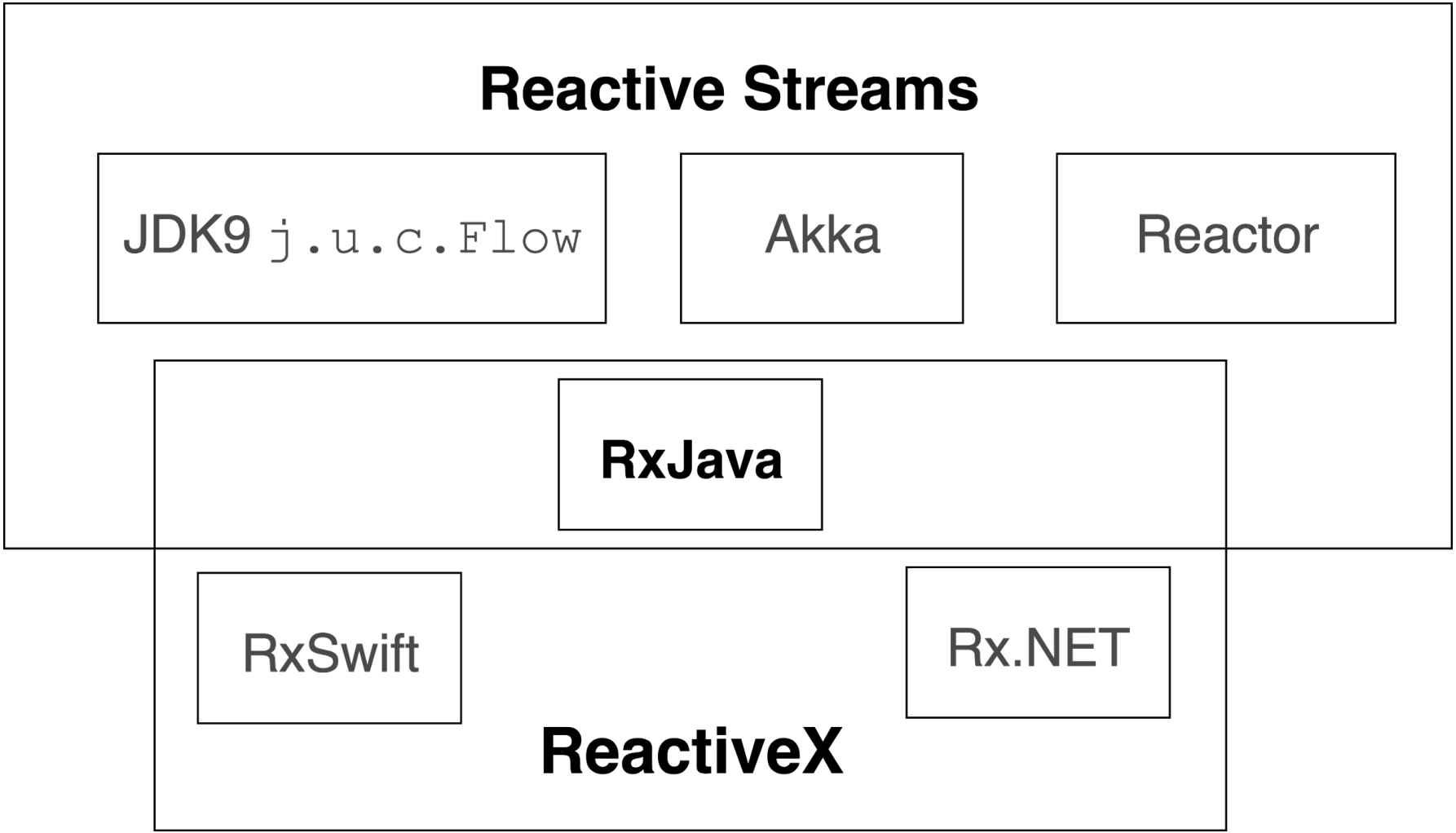


On backpressure buffer



ConnectableObservable




Reactive Streams
JDK9 j.u.c.Flow
Akka
Reactor
RxJava
RxSwift
Rx.NET
ReactiveX

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

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

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

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

References

- First place to go: http://reactivex.io/
- Marble diagrams for operators - http://rxmarbles.com
- https://github.com/Reactive-extensions/Rx.Net
- https://github.com/ReactiveX
- Dávid Karnok's blog - http://akarnokd.blogspot.com
- http://www.reactive-streams.org/
- https://github.com/Froussios/Intro-To-RxJava
- https://github.com/jraska/RxJava-Codelab