The Art of Rx

Матвей Мальков

Обо мне

Обо мне

Обо мне

Обо мне

Искусство реактивного программирования

Искусство — высокая степень мастерства в какой–либо сфере деятельности.

Наш главный навык — умение думать

Умение думать. Практика

Умение думать. Практика

- о чем думать?

Умение думать. Практика

- о чем думать?
- как думать?

О чем думать мобильному разработчику

О чем думать мобильному разработчику

- энергопотребление

О чем думать мобильному разработчику

- энергопотребление
- плавность интерфейса (60 fps)

О чем думать мобильному разработчику

- энергопотребление
- плавность интерфейса (60 fps)
- фрагментация устройств

О чем думать мобильному разработчику

- энергопотребление
- плавность интерфейса (60 fps)
- фрагментация устройств
- потребление памяти

Как думать мобильному разработчику?

Как думать мобильному разработчику?

- как пользователь

Как думать мобильному разработчику?

- как пользователь
- как дизайнер

Как думать мобильному разработчику?

- как пользователь
- как дизайнер
- как предлагает фундаментальный фреймворк

Фундаментальный фреймворк

Фундаментальный фреймворк

- делает разработку проще/лучше/быстрее

Фундаментальный фреймворк

- делает разработку проще/лучше/быстрее
- привносит что–то концептуально новое

Фундаментальный фреймворк

- делает разработку проще/лучше/быстрее
- привносит что–то концептуально новое
- прорастает во все подсистемы

Фундаментальный фреймворк

- делает разработку проще/лучше/быстрее
- привносит что–то концептуально новое
- прорастает во все подсистемы
- сложно убрать

Фундаментальный фреймворк

- делает разработку проще/лучше/быстрее
- привносит что–то концептуально новое
- прорастает во все подсистемы
- сложно убрать
- требует навыков обращения

Наш любимый фундаментальный фреймворк — RxJava

О чем думать, используя RxJava?

О контрактах

О контрактах

О контрактах

- проблемы:

О контрактах

- проблемы:
  - понимание

О контрактах

- проблемы:
  - понимание
  - соблюдение

О контрактах

- проблемы:
  - понимание
  - соблюдение
- важно:

О контрактах

- проблемы:
  - понимание
  - соблюдение
- важно:
  - основы

Нотификации

Нотификации

- onNext — 0, n или inf

Нотификации

- onNext — 0, n или inf
- onCompleted — 0 или 1

Нотификации

- onNext — 0, n или inf
- onCompleted — 0 или 1
- onError — 0 или 1

Нотификации

- onNext — 0, n или inf
- onCompleted — 0 или 1
- onError — 0 или 1
- onCompleted + onError <= 1

Подписка и отписка

Подписка и отписка

- не нужно отписываться после завершения

Подписка и отписка

- не нужно отписываться после завершения
- при отписке завершение не гарантировано

```
Observable<Response> obs = Observable
        .interval(1, TimeUnit.SECONDS)
        .map(this::sendPing)
        .cache();
```

```
Observable<Response> obs = Observable
        .interval(1, TimeUnit.SECONDS)
        .map(this::sendPing)
        .cache();
```

```
Observable<Response> obs = Observable
        .interval(1, TimeUnit.SECONDS)
        .map(this::sendPing)
        .cache();
```

```
Observable<Response> obs = Observable
        .interval(1, TimeUnit.SECONDS)
        .map(this::sendPing)
        .cache();
```

```
Observable<Response> obs = Observable
        .interval(1, TimeUnit.SECONDS)
        .map(this::sendPing)
        .cache();

Subscription sub = obs.subscribe();
```

```
Observable<Response> obs = Observable
        .interval(1, TimeUnit.SECONDS)
        .map(this::sendPing)
        .cache();

Subscription sub = obs.subscribe();

//позже
sub.unsubscribe();
```

```
Observable<Response> obs = Observable
        .interval(1, TimeUnit.SECONDS)
        .map(this::sendPing)
        .cache();

Subscription sub = obs.subscribe();

//позже
sub.unsubscribe();
```

Note: You sacrifice the ability to unsubscribe from the origin

```
Observable<Response> obs = Observable
        .interval(1, TimeUnit.SECONDS)
        .map(this::sendPing)
        .cache();

Subscription sub = obs.subscribe();

//позже
sub.unsubscribe();
```

Note: You sacrifice the ability to unsubscribe from the origin

О контрактах

О контрактах

- отправка нотификаций

О контрактах

- отправка нотификаций
- завершение с ошибкой

О контрактах

- отправка нотификаций
- завершение с ошибкой
- полный список : http://reactivex.io/documentation/contract.html

Состояние системы

Состояние системы

- изолированно

Состояние системы

- изолированно
- один источник

Состояние системы

- изолированно
- один источник
- неизменяемо

Состояние системы

- изолированно
- один источник
- неизменяемо
- подчиняется контракту Observable

```
nameEditTextChanges.subscribe(this::updateName);
```

```
nameEditTextChanges.subscribe(this::updateName);
```

```
PublishSubject<Integer> nameEditTextChanges =
        PublishSubject.create();
```

```
nameEditTextChanges.subscribe(this::updateName);
```

```
PublishSubject<Integer> nameEditTextChanges =
        PublishSubject.create();

nameEditTextChanges.onNext("хаха, обманул!");
```

Subject — возможное нарушение контракта

Subject — возможное нарушение контракта

Состояние системы

- ~~изолированно~~
- ~~один источник~~
- ~~неизменяемо~~
- ~~подчиняется контракту Observable~~

Subjects

Subjects

- соединение императивного и реактивного мира

Subjects

- соединение императивного и реактивного мира
- когда невозможно сделать по другому

Subjects

- соединение императивного и реактивного мира
- когда невозможно сделать по другому
- это ваш случай? НЕТ!

Subjects

- соединение императивного и реактивного мира
- когда невозможно сделать по другому
- это ваш случай? НЕТ!
- все таки ваш? Нет, это не так!

Subjects

- соединение императивного и реактивного мира
- когда невозможно сделать по другому
- это ваш случай? НЕТ!
- все таки ваш? Нет, это не так!
- меньше — лучше

О тредах

О тредах

О тредах

- проблемы

О тредах

- проблемы
  - нет понимая работы

О тредах

- проблемы
  - нет понимая работы
  - суют subscribeOn и observeOn везде подряд

О тредах

- проблемы
  - нет понимая работы
  - суют subscribeOn и observeOn везде подряд
- важно :

О тредах

- проблемы
  - нет понимая работы
  - суют subscribeOn и observeOn везде подряд
- важно :
  - разбираться

Scheduler

Scheduler

- планировка задач

Scheduler

- планировка задач
- работа с тредами

Scheduler

- планировка задач
- работа с тредами
- subscribeOn

Scheduler

- планировка задач
- работа с тредами
- subscribeOn
- observeOn

Догмы
observeOn
subscribeOn

1. observeOn меняет Scheduler для всего, что ниже по коду

2. subscribeOn меняет Scheduler с самого верха цепочки

3. Каждый следующий observeOn заменяет предыдущий

4. subscribeOn актуален до первого observeOn

Всегда думайте о том,
в каком треде выполняется функция

```
Observable<Response> requestObs = userDbChanges
        .filter(u → u.age < 18)
        .map(this::toRequestModel)
        .map(this::doRequest)
        .onErrorReturn(this::doRecoverReq)
        .map(this::cacheAndModify);
//ui thread
requestObs.subscribe(this::showOnUi);
```

```
Observable<Response> requestObs = userDbChanges
        .filter(u → u.age < 18)
        .map(this::toRequestModel)
        .map(this::doRequest)
        .onErrorReturn(this::doRecoverReq)
        .map(this::cacheAndModify);
//ui thread
requestObs.subscribe(this::showOnUi);
```

```
Observable<Response> requestObs = userDbChanges
        .filter(u → u.age < 18)
        .map(this::toRequestModel)
        .observeOn(networkScheduler)
        .map(this::doRequest)
        .onErrorReturn(this::doRecoverReq)
        .subscribeOn(Schedulers.computation())
        .observeOn(mainThread)
        .map(this::toUiModel);
//ui thread
requestObs.subscribe(this::showOnUi);
```

```
Observable<Response> requestObs = userDbChanges
        .filter(u → u.age < 18)
        .map(this::toRequestModel)
        .observeOn(networkScheduler)
        .map(this::doRequest)
        .onErrorReturn(this::doRecoverReq)
        .subscribeOn(Schedulers.computation())
        .observeOn(mainThread)
        .map(this::toUiModel);
//ui thread
requestObs.subscribe(this::showOnUi);
```

```
Observable<Response> requestObs = userDbChanges
        .filter(u → u.age < 18)
        .map(this::toRequestModel)
        .observeOn(networkScheduler)
        .map(this::doRequest)
        .onErrorReturn(this::doRecoverReq)
        .subscribeOn(Schedulers.computation())
        .observeOn(mainThread)
        .map(this::toUiModel);
//ui thread
requestObs.subscribe(this::showOnUi);
```

```
Observable<Response> requestObs = userDbChanges
        .filter(u → u.age < 18)
        .map(this::toRequestModel)
        .observeOn(networkScheduler)
        .map(this::doRequest)
        .onErrorReturn(this::doRecoverReq)
        .subscribeOn(Schedulers.computation())
        .observeOn(mainThread)
        .map(this::toUiModel);
//ui thread
requestObs.subscribe(this::showOnUi);
```

```
Observable<Response> requestObs = userDbChanges
        .filter(u → u.age < 18)
        .map(this::toRequestModel)
        .observeOn(networkScheduler)
        .map(this::doRequest)
        .onErrorReturn(this::doRecoverReq)
        .subscribeOn(Schedulers.computation())
        .observeOn(mainThread)
        .map(this::toUiModel);
//ui thread
requestObs.subscribe(this::showOnUi);
```

```
Observable<Response> requestObs = userDbChanges
            .filter(u → u.age < 18)
            .map(this::toRequestModel)
            .observeOn(networkScheduler)
            .map(this::doRequest)
            .onErrorReturn(this::doRecoverReq)
            .subscribeOn(Schedulers.computation())
            .observeOn(mainThread)
            .map(this::toUiModel);
//ui thread
requestObs.subscribe(this::showOnUi);
```

```
Observable<Response> requestObs = userDbChanges
        .filter(u → u.age < 18)
        .map(this::toRequestModel)
        .observeOn(networkScheduler)
        .map(this::doRequest)
        .onErrorReturn(this::doRecoverReq)
        .subscribeOn(Schedulers.computation())
        .observeOn(mainThread)
        .map(this::toUiModel);
//ui thread
requestObs.subscribe(this::showOnUi);
```

Великолепный subscribeOn

subscribeOn

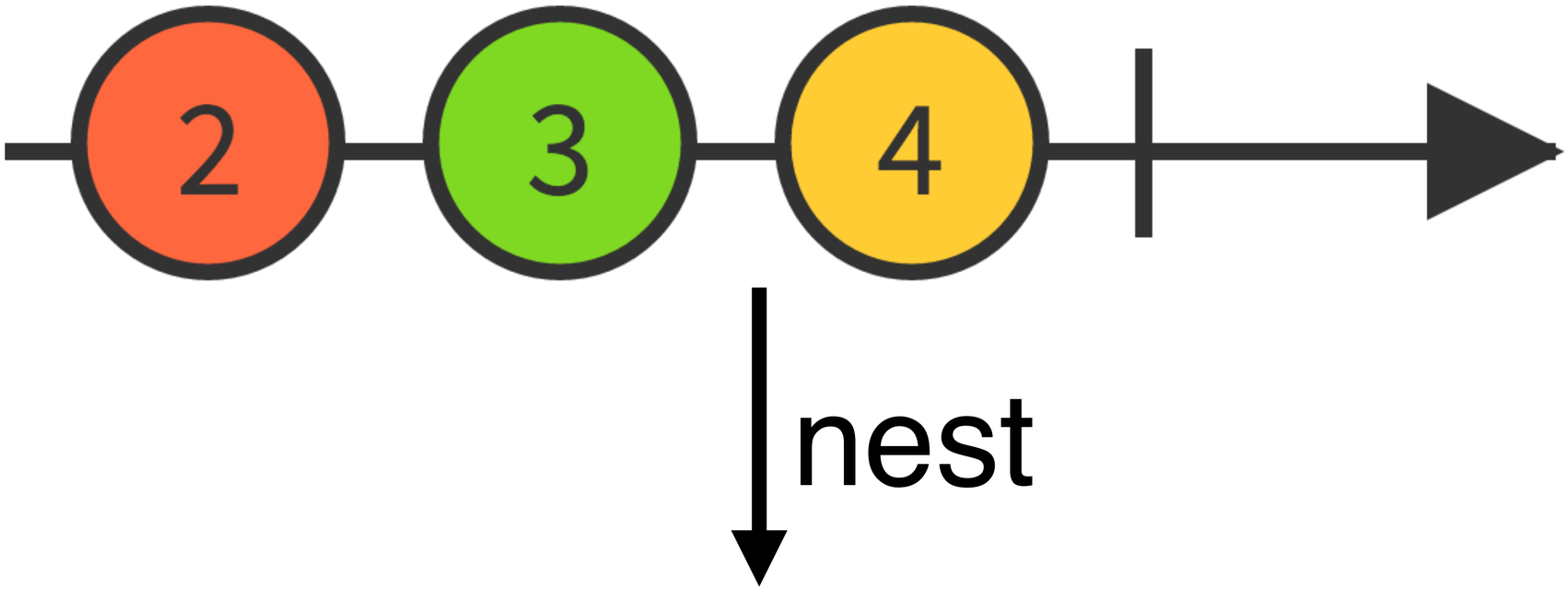

```
public final Observable<T> subscribeOn(Scheduler scheduler) {

    //какой-то неважный код

    return nest().lift(new OperatorSubscribeOn<T>(scheduler));
}
```

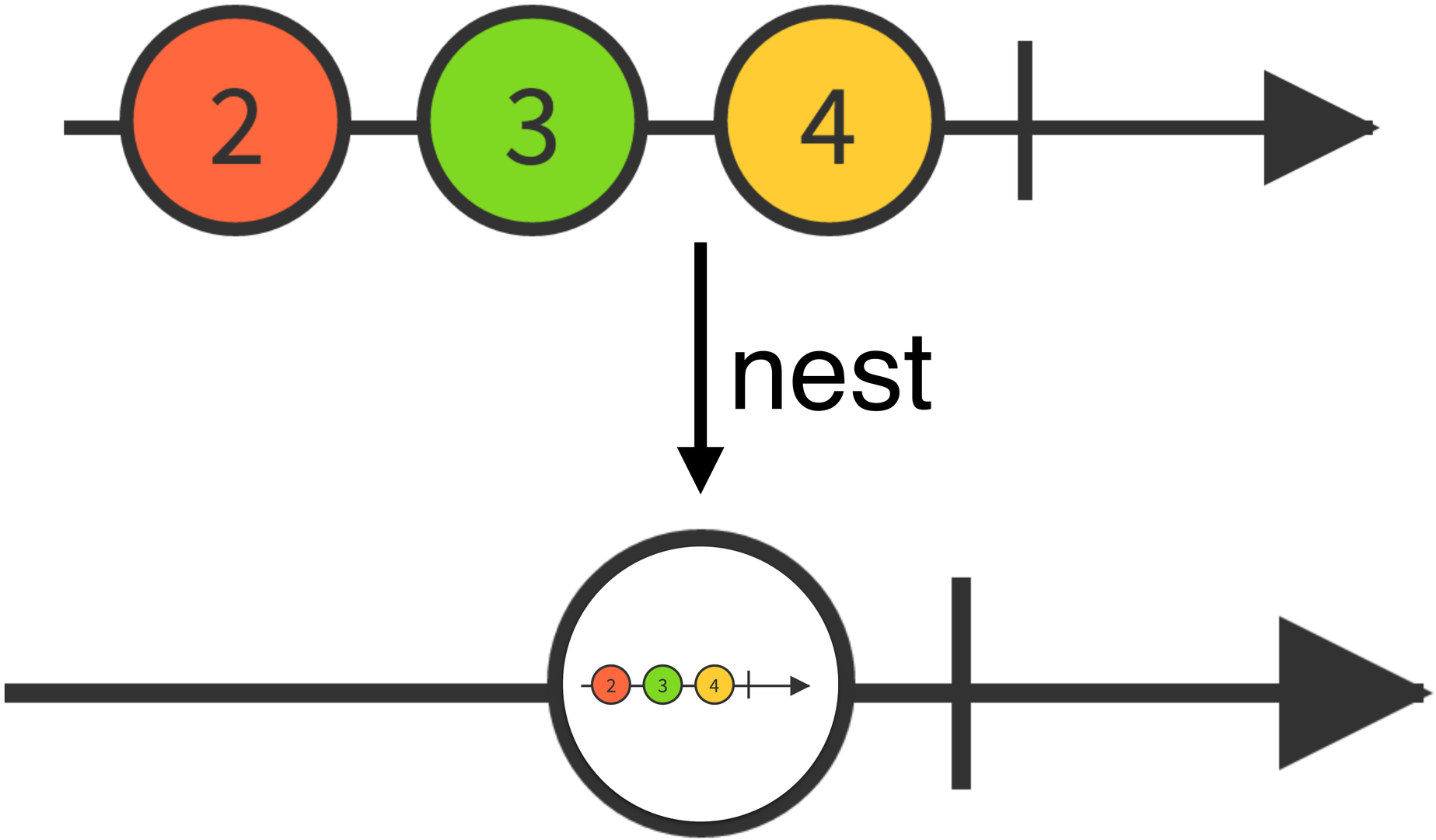
subscribeOn


```
public final Observable<T> subscribeOn(Scheduler scheduler) {
    //какой-то неважный код
    return nest().lift(new OperatorSubscribeOn<T>(scheduler));
}
```

```
public final Observable<Observable<T>> nest() {
    return just(this);
}
```


2
3
4


2
3
4
nest


2
3
4
nest
2
3
4

subscribeOn


```
public final Observable<T> subscribeOn(Scheduler scheduler) {

    //какой-то неважный код

    return nest().lift(new OperatorSubscribeOn<T>(scheduler));
}
```

subscribeOn


```
public final Observable<T> subscribeOn(Scheduler scheduler) {

    //какой-то неважный код

    return nest().lift(new OperatorSubscribeOn<T>(scheduler));
}
```

subscribeOn


```
public final Observable<T> subscribeOn(Scheduler scheduler) {
    //какой-то неважный код
    return nest().lift(new OperatorSubscribeOn<T>(scheduler));
}
```

```
public class OperatorSubscribeOn<T> implements Operator<T, Observable<T>> {

    private final Scheduler scheduler;

    public OperatorSubscribeOn(Scheduler scheduler) {
        this.scheduler = scheduler;
    }

    @Override
    public Subscriber<? super Observable<T>> call(final Subscriber<? super T> subscriber) {
            …
       }
```

возвращает принимает

```
public class OperatorSubscribeOn<T> implements Operator<T, Observable<T>> {

    private final Scheduler scheduler;

    public OperatorSubscribeOn(Scheduler scheduler) {
        this.scheduler = scheduler;
    }

    @Override
    public Subscriber<? super Observable<T>> call(final Subscriber<? super T> subscriber) {
            …
       }
```

```
public class OperatorSubscribeOn<T> implements Operator<T, Observable<T>> {

    private final Scheduler scheduler;

    public OperatorSubscribeOn(Scheduler scheduler) {
        this.scheduler = scheduler;
    }

    @Override
    public Subscriber<? super Observable<T>> call(final Subscriber<? super T> subscriber) {
            …
       }
```

```
@Override
public Subscriber<? super Observable<T>> call(final Subscriber<? super T> subscriber) {
    final Worker inner = scheduler.createWorker();
    subscriber.add(inner);
    return new Subscriber<Observable<T>>(subscriber) {

        @Override
        public void onNext(final Observable<T> o) {
            inner.schedule(new Action0() {

                @Override
                public void call() {
                    final Thread t = Thread.currentThread();
                    o.unsafeSubscribe(new Subscriber<T>(subscriber) {
                        //обычный inner subscribe
                    }


//тут еще ооооченьь много закрывающихся скобочек
}
```

```
@Override
public Subscriber<? super Observable<T>> call(final Subscriber<? super T> subscriber) {
    final Worker inner = scheduler.createWorker();
    subscriber.add(inner);
    return new Subscriber<Observable<T>>(subscriber) {

        @Override
        public void onNext(final Observable<T> o) {
            inner.schedule(new Action0() {

                @Override
                public void call() {
                    final Thread t = Thread.currentThread();
                    o.unsafeSubscribe(new Subscriber<T>(subscriber) {
                        //обычный inner subscribe
                    }


//тут еще ооооченьь много закрывающихся скобочек
}
```

```
@Override
public Subscriber<? super Observable<T>> call(final Subscriber<? super T> subscriber) {
    final Worker inner = scheduler.createWorker();
    subscriber.add(inner);
    return new Subscriber<Observable<T>>(subscriber) {

        @Override
        public void onNext(final Observable<T> o) {
            inner.schedule(new Action0() {

                @Override
                public void call() {
                    final Thread t = Thread.currentThread();
                    o.unsafeSubscribe(new Subscriber<T>(subscriber) {
                        //обычный inner subscribe
                    }


//тут еще ооооченьь много закрывающихся скобочек
}
```

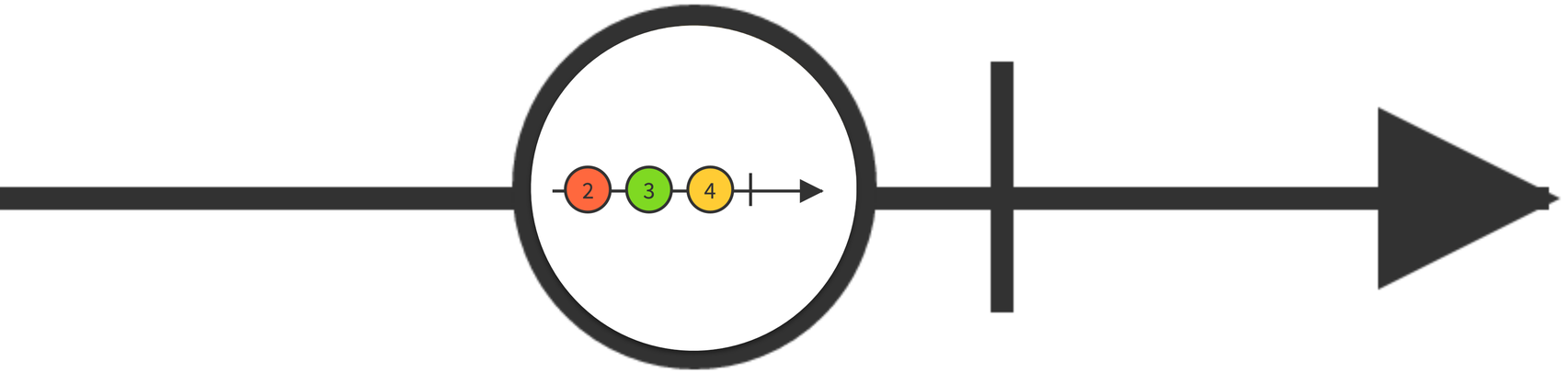
```
@Override
public Subscriber<? super Observable<T>> call(final Subscriber<? super T> subscriber) {
    final Worker inner = scheduler.createWorker();
    subscriber.add(inner);
    return new Subscriber<Observable<T>>(subscriber) {

        @Override
        public void onNext(final Observable<T> o) {
            inner.schedule(new Action0() {

                @Override
                public void call() {
                    final Thread t = Thread.currentThread();
                    o.unsafeSubscribe(new Subscriber<T>(subscriber) {
                        //обычный inner subscribe
                    }


//тут еще ооооченьь много закрывающихся скобочек
}
```

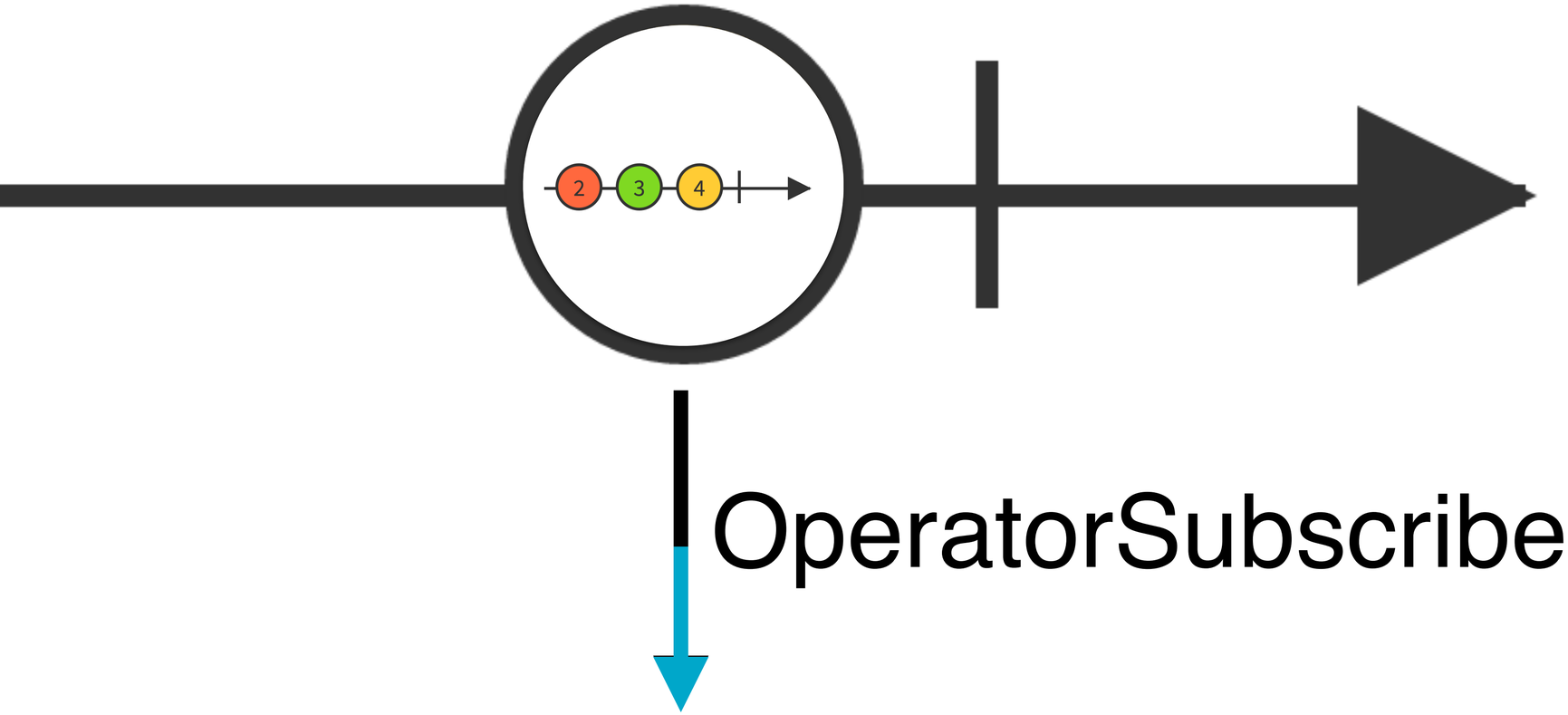
```
@Override
public Subscriber<? super Observable<T>> call(final Subscriber<? super T> subscriber) {
    final Worker inner = scheduler.createWorker();
    subscriber.add(inner);
    return new Subscriber<Observable<T>>(subscriber) {

        @Override
        public void onNext(final Observable<T> o) {
            inner.schedule(new Action0() {

                @Override
                public void call() {
                    final Thread t = Thread.currentThread();
                    o.unsafeSubscribe(new Subscriber<T>(subscriber) {
                        //обычный inner subscribe
                    }


//тут еще ооооченьь много закрывающихся скобочек
}
```

Наш Observable

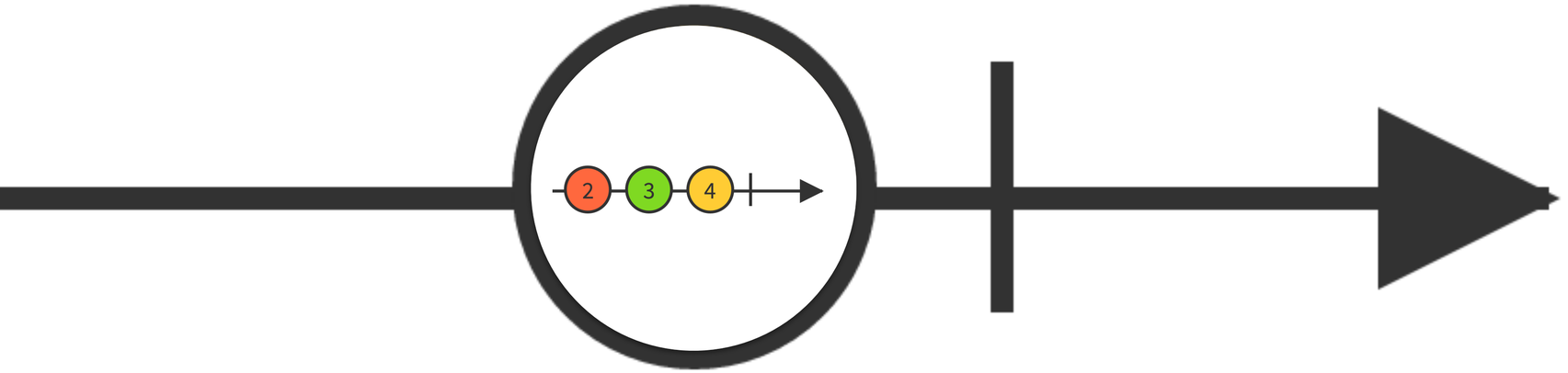
```
@Override
public Subscriber<? super Observable<T>> call(final Subscriber<? super T> subscriber) {
    final Worker inner = scheduler.createWorker();
    subscriber.add(inner);
    return new Subscriber<Observable<T>>(subscriber) {

        @Override
        public void onNext(final Observable<T> o) {
            inner.schedule(new Action0() {

                @Override
                public void call() {
                    final Thread t = Thread.currentThread();
                    o.unsafeSubscribe(new Subscriber<T>(subscriber) {
                        //обычный inner subscribe
                    }


//тут еще ооооченьь много закрывающихся скобочек
}
```

```
@Override
public Subscriber<? super Observable<T>> call(final Subscriber<? super T> subscriber) {
    final Worker inner = scheduler.createWorker();
    subscriber.add(inner);
    return new Subscriber<Observable<T>>(subscriber) {

        @Override
        public void onNext(final Observable<T> o) {
            inner.schedule(new Action0() {

                @Override
                public void call() {
                    final Thread t = Thread.currentThread();
                    o.unsafeSubscribe(new Subscriber<T>(subscriber) {
                        //обычный inner subscribe
                    }


//тут еще ооооченьь много закрывающихся скобочек
}
```


2
3
4



OperatorSubscribeOn(Scheduler)



OperatorSubscribeOn(Scheduler)

Гениально!

```
Observable.just(1)
          .subscribeOn(scheduler1)
          .subscribeOn(scheduler2)
          .subscribe();
```

```
Observable.just(1)
          .subscribeOn(scheduler1)
          .subscribeOn(scheduler2)
          .subscribe(); //scheduler1
```

2. subscribeOn меняет Scheduler с самого верха цепочки

2. Первый subscribeOn меняет Scheduler с самого верха цепочки

```
//computation
Observable<Long> o1 = Observable.interval(1, TimeUnit.SECONDS);
//network
Observable<Integer> o2 = Observable.just(1)
        .observeOn(networkScheduler);

Observable.zip(o1, o2, o3, (r1, r2) → r1 * r2);
```

```
//computation
Observable<Long> o1 = Observable.interval(1, TimeUnit.SECONDS);
//network
Observable<Integer> o2 = Observable.just(1)
        .observeOn(networkScheduler);

Observable.zip(o1, o2, o3, (r1, r2) → r1 * r2);
```

```
//computation
Observable<Long> o1 = Observable.interval(1, TimeUnit.SECONDS);
//network
Observable<Integer> o2 = Observable.just(1)
        .observeOn(networkScheduler);

Observable.zip(o1, o2, o3, (r1, r2) → r1 * r2);
                                //computation or network
```

```
final class InnerSubscriber extends Subscriber {

    @Override
    public void onNext(Object t) {
        try {
            items.onNext(t);
        } catch (MissingBackpressureException e) {
            onError(e);
        }
        tick();
    }
};
```

```
final class InnerSubscriber extends Subscriber {

    @Override
    public void onNext(Object t) {
        try {
            items.onNext(t);
        } catch (MissingBackpressureException e) {
            onError(e);
        }
        tick();
    }
};
```

```
void tick() {
    //тут много кода, не имеющего сейчас значения

            if (requested.get() > 0 && allHaveValues) {
                try {
                  // всем потокам есть что zip'ать
                  child.onNext(zipFunction.call(vs));
                }
   // еще много кода и скобочки
```

```
void tick() {
    //тут много кода, не имеющего сейчас значения

            if (requested.get() > 0 && allHaveValues) {
                try {
                  // всем потокам есть что zip'ать
                  child.onNext(zipFunction.call(vs));
                }
   // еще много кода и скобочки
```

```
void tick() {
    //тут много кода, не имеющего сейчас значения

            if (requested.get() > 0 && allHaveValues) {
                try {
                    // всем потокам есть что zip'ать
                    child.onNext(zipFunction.call(vs));
                }
   // еще много кода и скобочки
```

Всегда думайте о том,
в каком треде выполняется функция

О количестве элементов

Количество элементов

Количество элементов

- проблемы:

Количество элементов

- проблемы:
  - больше элементов — больше мусора

Количество элементов

- проблемы:
  - больше элементов — больше мусора
  - RxJava постоянно все копирует

Количество элементов

- проблемы:
  - больше элементов — больше мусора
  - RxJava постоянно все копирует
- важно:

Количество элементов

- проблемы:
  - больше элементов — больше мусора
  - RxJava постоянно все копирует
- важно:
  - не паниковать

Количество элементов

- проблемы:
  - больше элементов — больше мусора
  - RxJava постоянно все копирует
- важно:
  - не паниковать
  - заботиться о GC

Не паниковать

Не паниковать

- «Что с GC?» : самый жаркий вопрос

Не паниковать

- «Что с GC?» : самый жаркий вопрос
- копирование только при нотификациях

Не паниковать

- «Что с GC?» : самый жаркий вопрос
- копирование только при нотификациях
- все оптимизированно

```
locationUpdates()
        .distinctUntilChanged()
        .subscribe();
```

```
locationUpdates()
        .distinctUntilChanged()
        .subscribe();
```

```
locationUpdates()
        .distinctUntilChanged()
        .subscribe();
```

```
touchEvents(view)
        .filter(e → e.x > minWidth && e.y > minHeight)
        .doOnNext(this::processEvents)
        .subscribe();
```

```
locationUpdates()
        .distinctUntilChanged()
        .subscribe();
```

```
touchEvents(view)
        .filter(e → e.x > minWidth && e.y > minHeight)
        .doOnNext(this::processEvents)
        .subscribe();
```

```
nameEditTextChanges
        .map(this::prepareForSending)
        .map(this::sendName)
        .map(this::cacheAndModify);
```

```
nameEditTextChanges
        .map(this::prepareForSending) //раз
        .map(this::sendName)
        .map(this::cacheAndModify);
```

```
nameEditTextChanges
        .map(this::prepareForSending) //раз
        .map(this::sendName) // два
        .map(this::cacheAndModify);
```

```
nameEditTextChanges
        .map(this::prepareForSending) //раз
        .map(this::sendName) // два
        .map(this::cacheAndModify); // три
```

```
nameEditTextChanges
        .debounce(500, TimeUnit.MILLISECONDS)
        .map(this::prepareForSending)
        .map(this::sendName)
        .map(this::cacheAndModify);
```

debounce

debounce

- бережет GC

debounce

- бережет GC
- бережет трафик пользователя

debounce

- бережет GC
- бережет трафик пользователя
- бережет нас от **BackPressure**

Backpressure

Backpressure

- отдающий тред дает много

Backpressure

- отдающий тред дает много
- принимающий тред не успевает

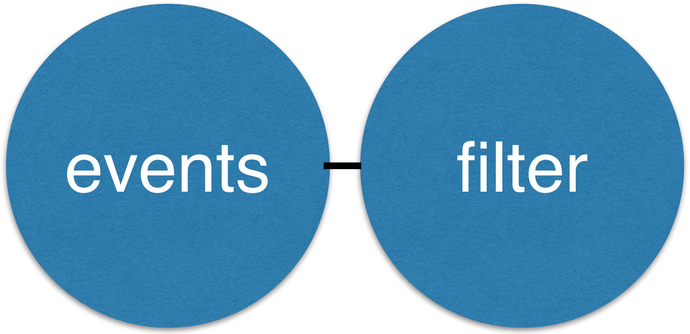
Backpressure

- отдающий тред дает много
- принимающий тред не успевает
- в многопоточной среде многовероятно

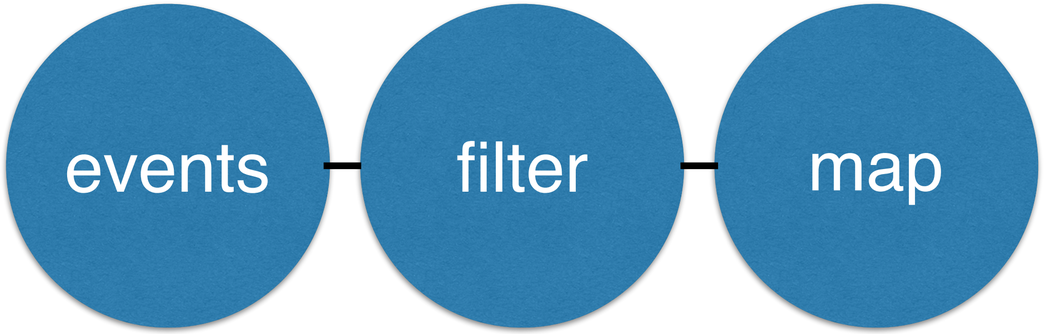
Backpressure

- отдающий тред дает много
- принимающий тред не успевает
- в многопоточной среде многовероятно
- первый признак — observeOn

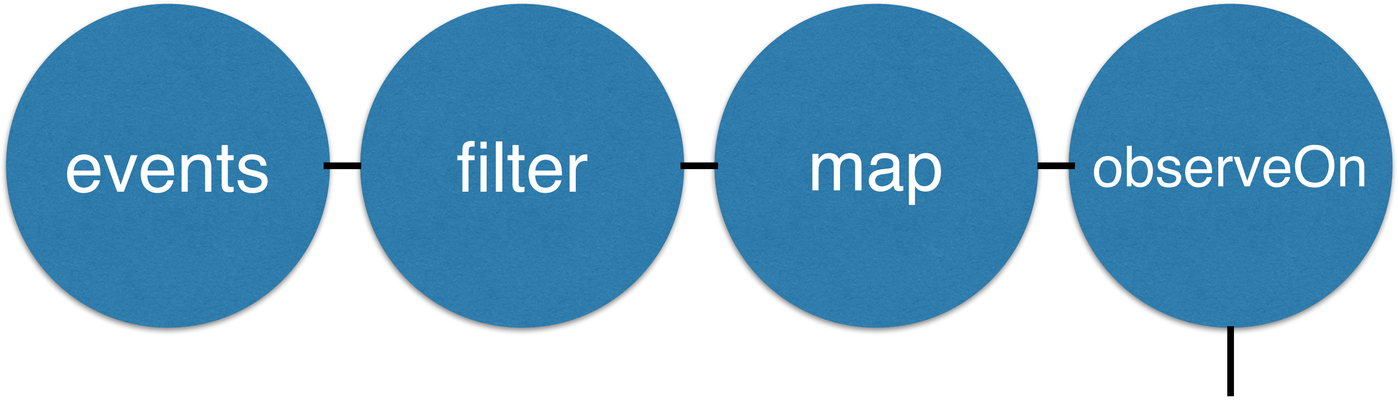

```
touchEvents(view)
```



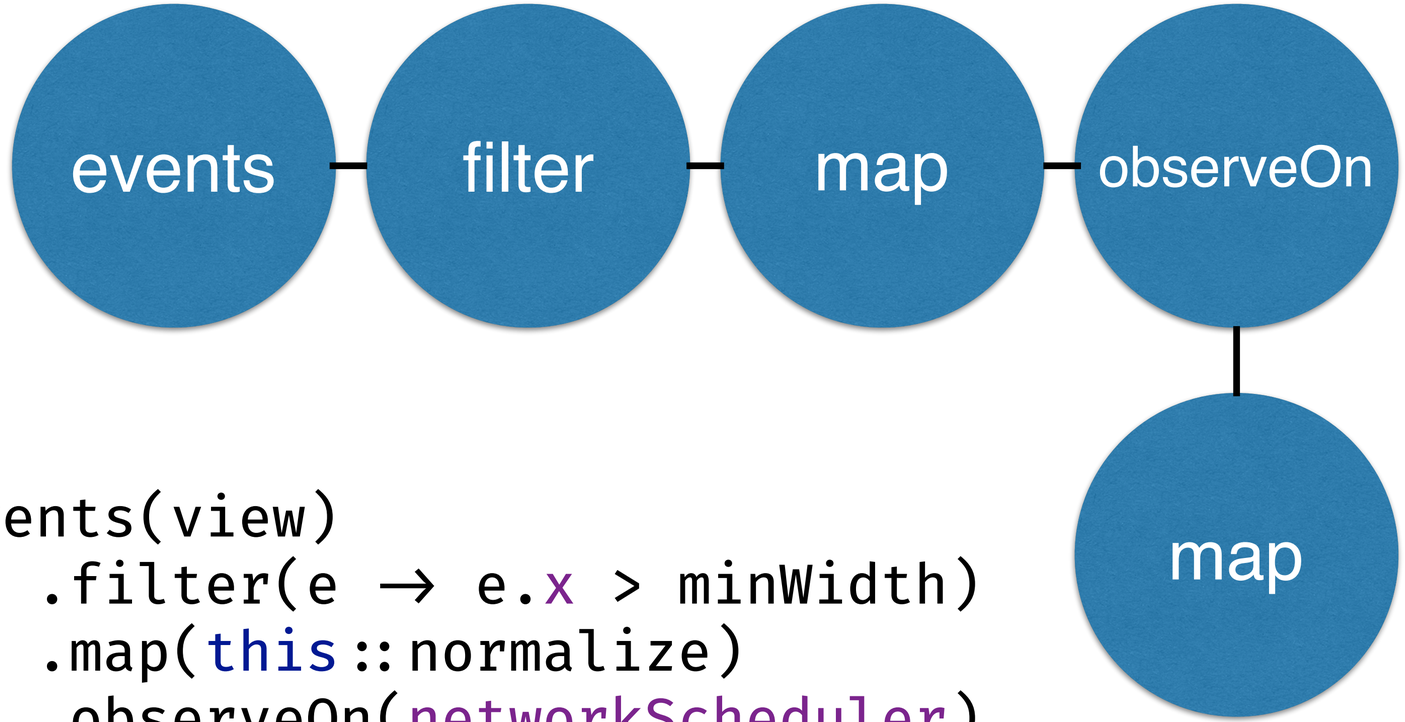

```
touchEvents(view)
        .filter(e → e.x > minWidth)
```



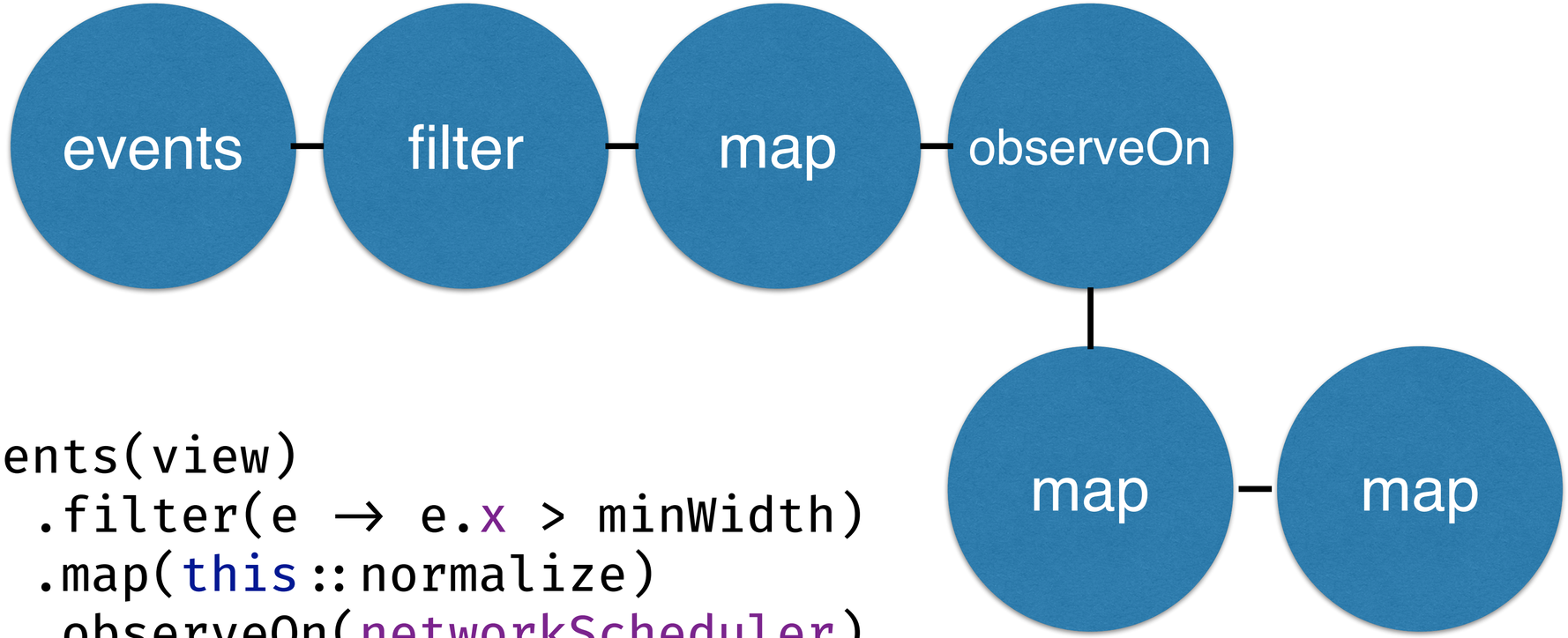

```
touchEvents(view)
        .filter(e → e.x > minWidth)
        .map(this::normalize)
```



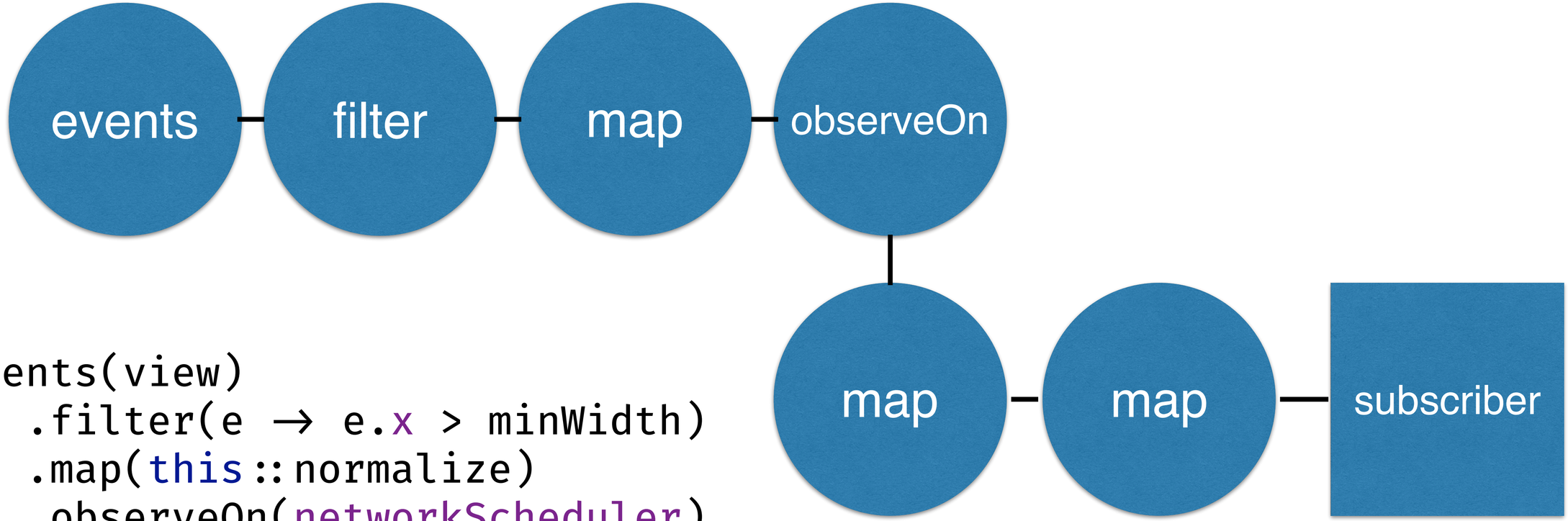

```
touchEvents(view)
        .filter(e → e.x > minWidth)
        .map(this::normalize)
        .observeOn(networkScheduler)
```



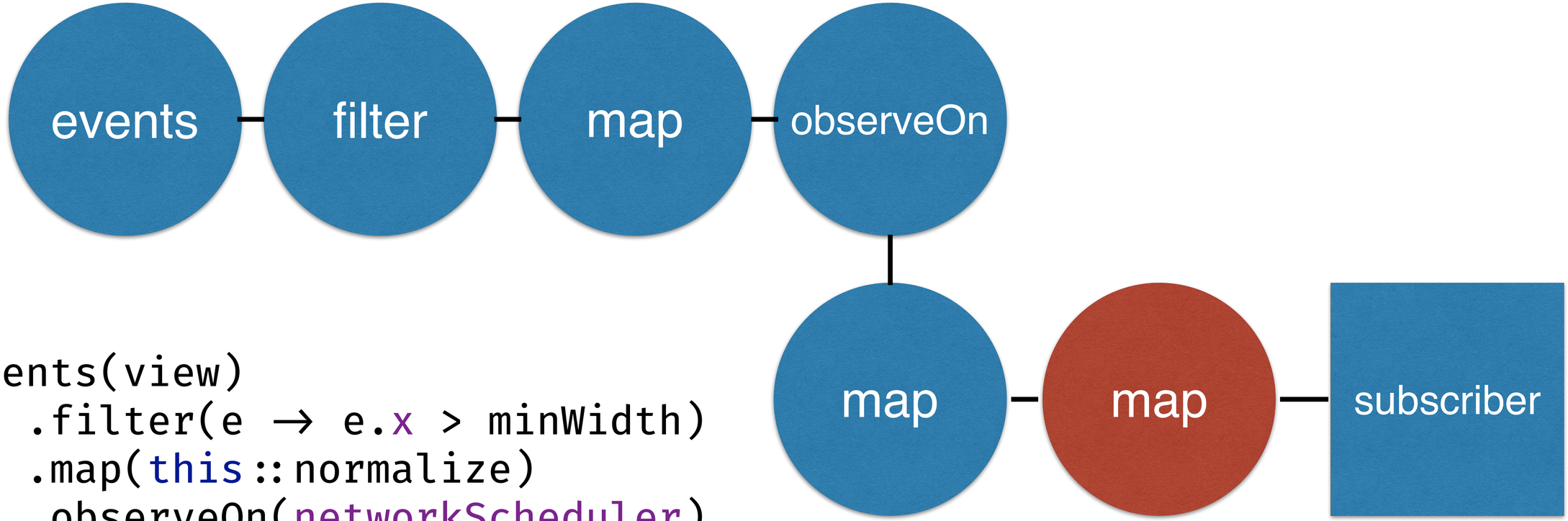

```
touchEvents(view)
        .filter(e → e.x > minWidth)
        .map(this::normalize)
        .observeOn(networkScheduler)
        .map(this::prepareForSending)
```



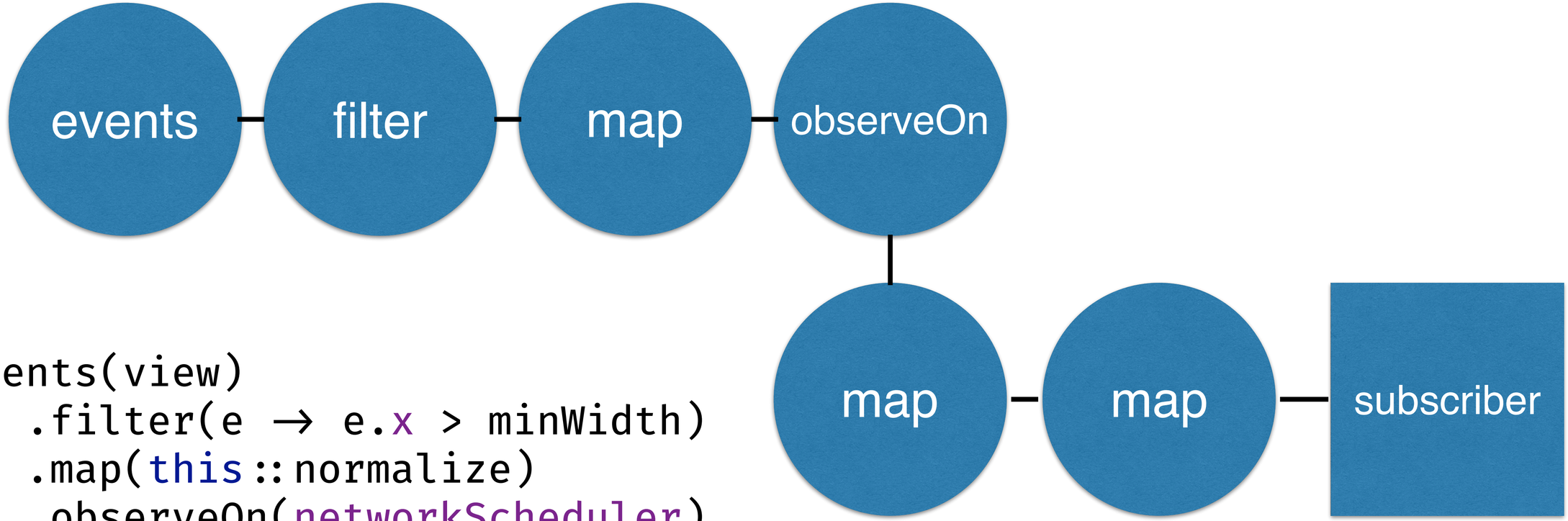

```
touchEvents(view)
        .filter(e → e.x > minWidth)
        .map(this::normalize)
        .observeOn(networkScheduler)
        .map(this::prepareForSending)
        .map(this::doMotionRequest)
```



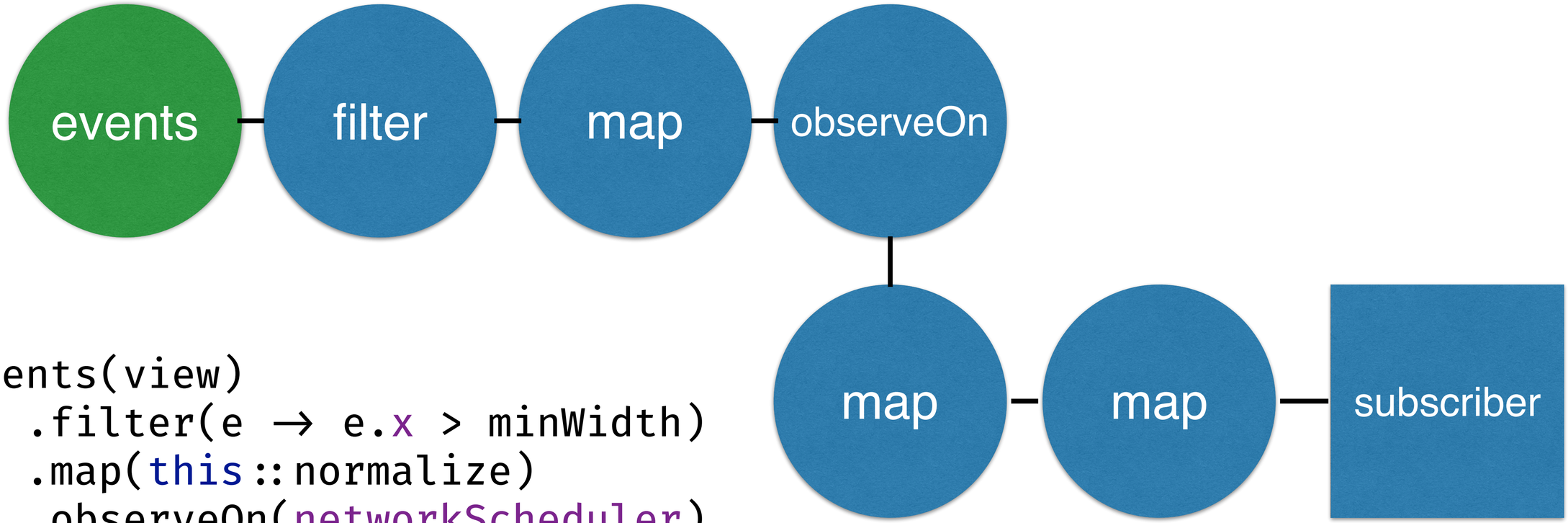

```
touchEvents(view)
        .filter(e → e.x > minWidth)
        .map(this::normalize)
        .observeOn(networkScheduler)
        .map(this::prepareForSending)
        .map(this::doMotionRequest)
        .subscribe()
```



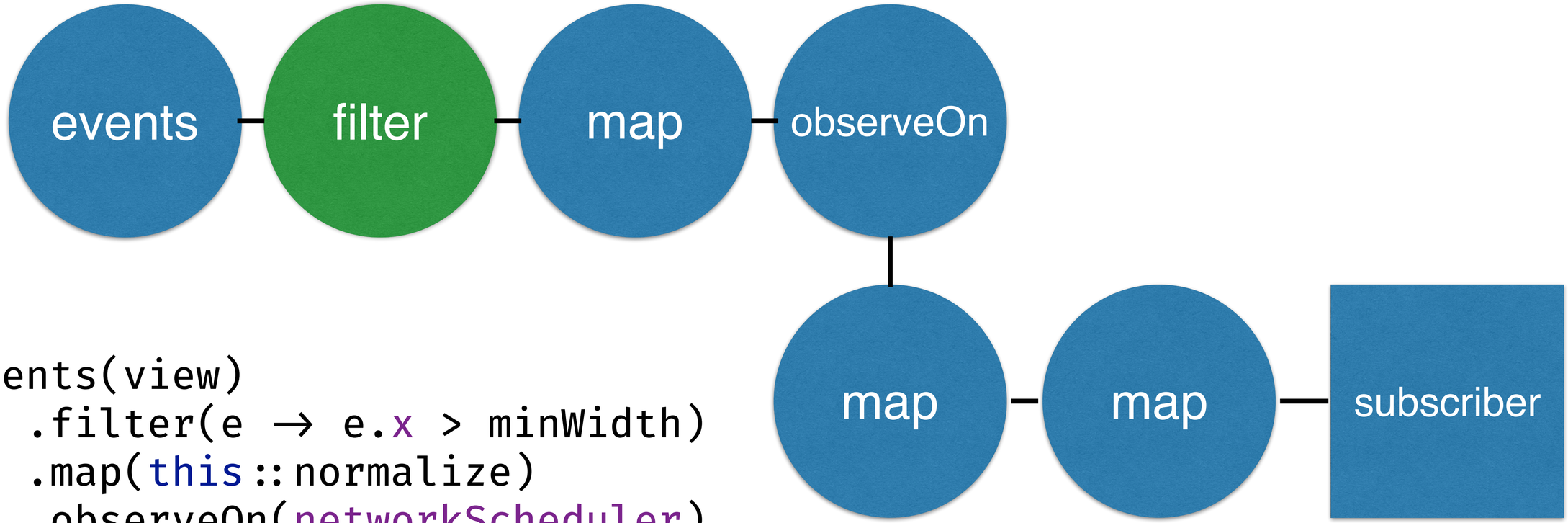

```
touchEvents(view)
        .filter(e → e.x > minWidth)
        .map(this::normalize)
        .observeOn(networkScheduler)
        .map(this::prepareForSending)
        .map(this::doMotionRequest)
        .subscribe()
```



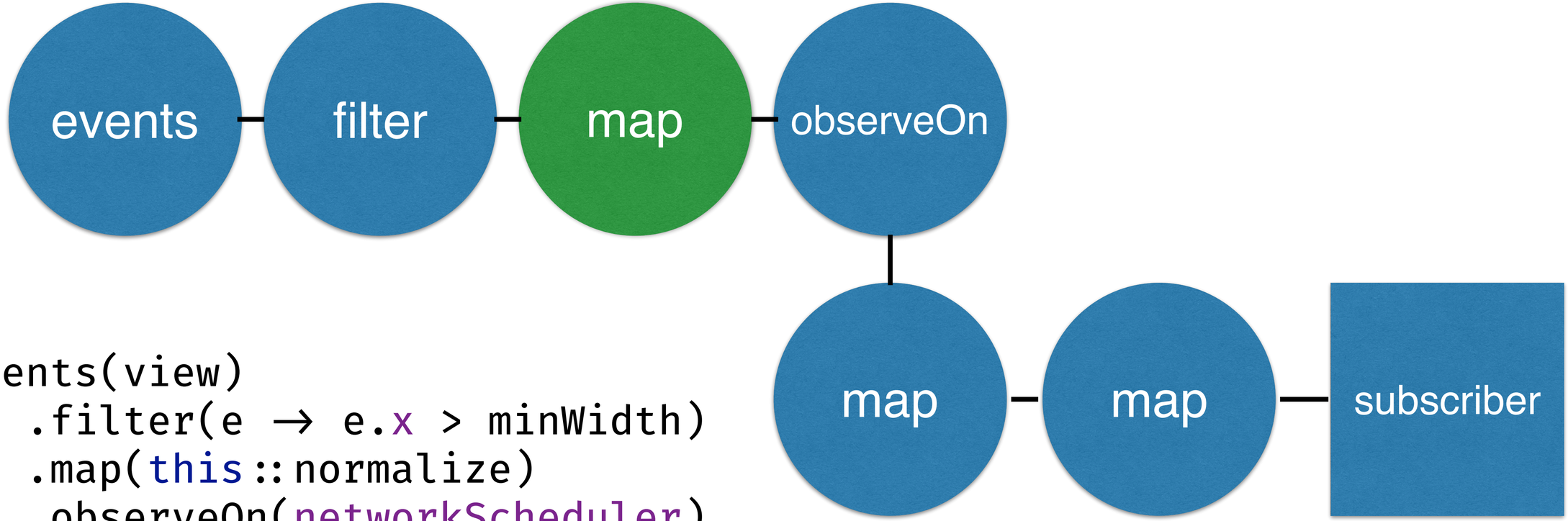

```
touchEvents(view)
        .filter(e → e.x > minWidth)
        .map(this::normalize)
        .observeOn(networkScheduler)
        .map(this::prepareForSending)
        .map(this::doMotionRequest)
        .subscribe()
```



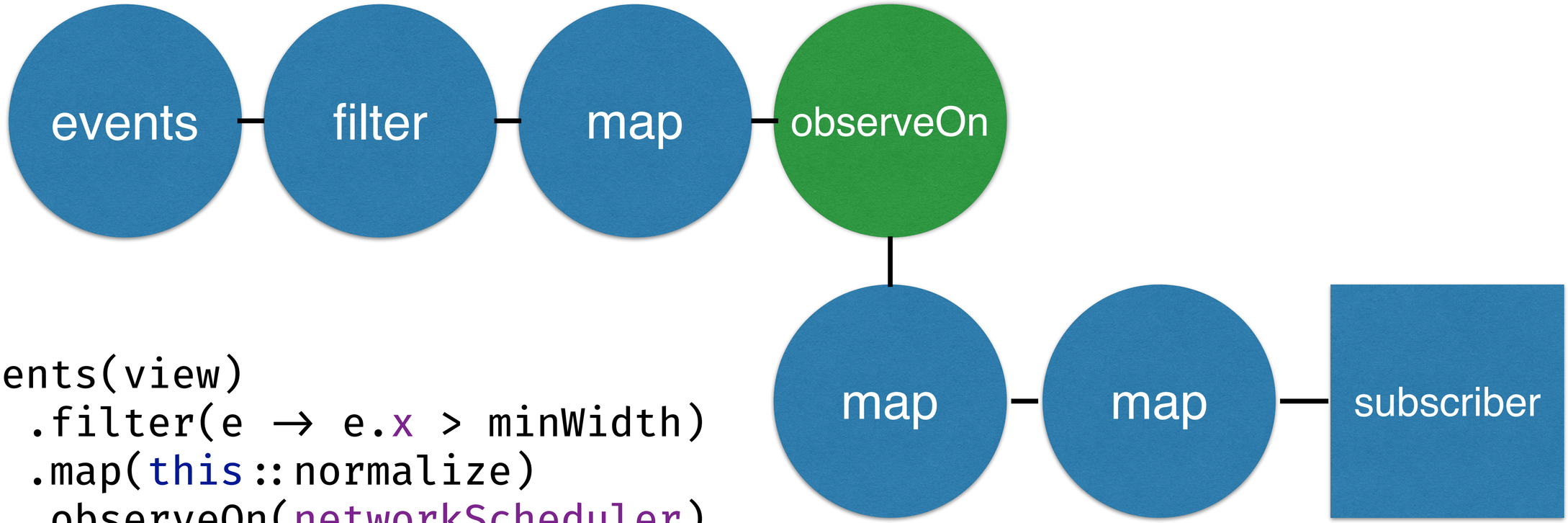

```
touchEvents(view)
        .filter(e → e.x > minWidth)
        .map(this::normalize)
        .observeOn(networkScheduler)
        .map(this::prepareForSending)
        .map(this::doMotionRequest)
        .subscribe()
```



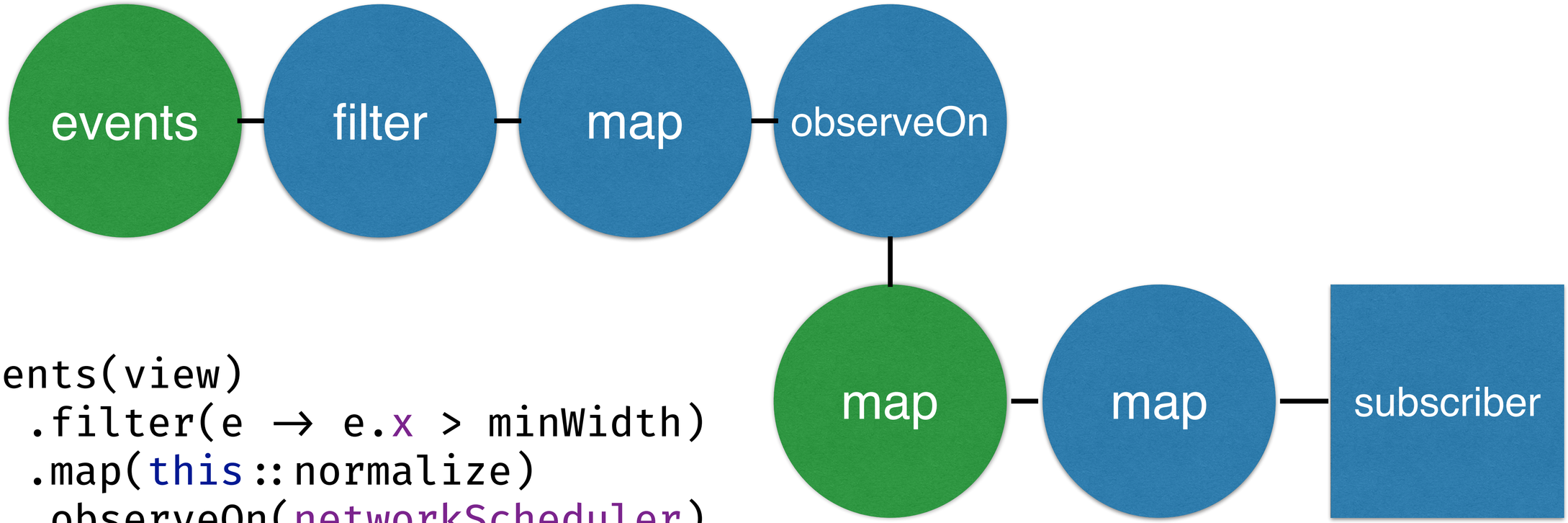

```
touchEvents(view)
        .filter(e → e.x > minWidth)
        .map(this::normalize)
        .observeOn(networkScheduler)
        .map(this::prepareForSending)
        .map(this::doMotionRequest)
        .subscribe()
```



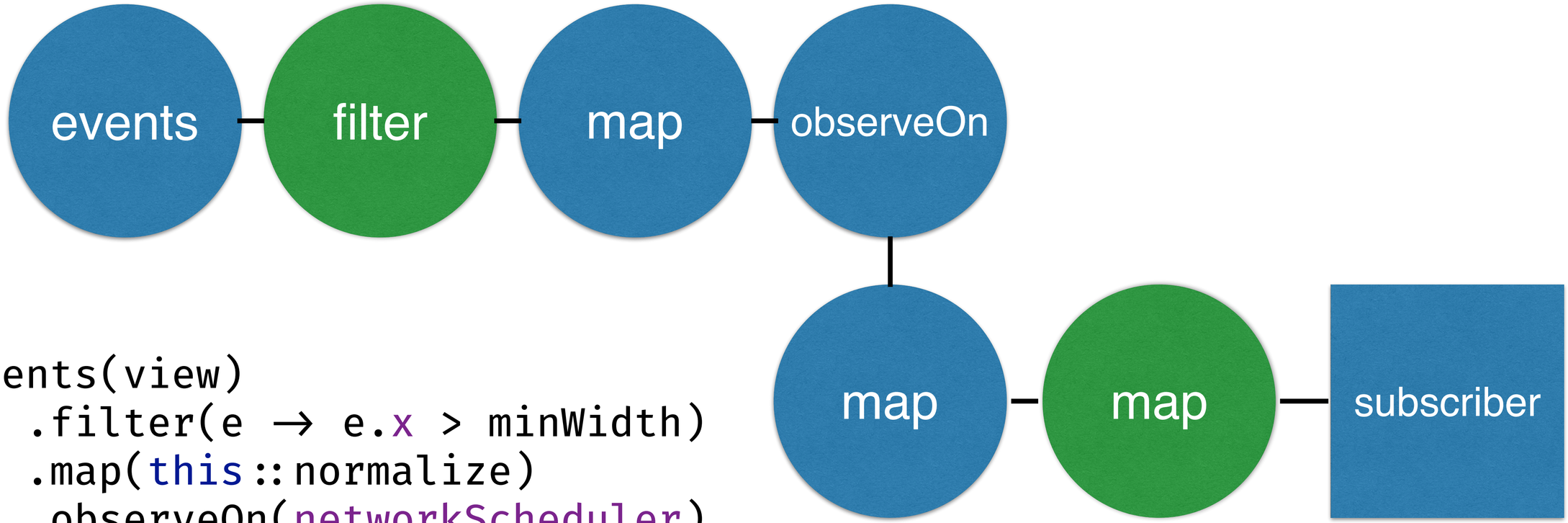

```
touchEvents(view)
        .filter(e → e.x > minWidth)
        .map(this::normalize)
        .observeOn(networkScheduler)
        .map(this::prepareForSending)
        .map(this::doMotionRequest)
        .subscribe()
```



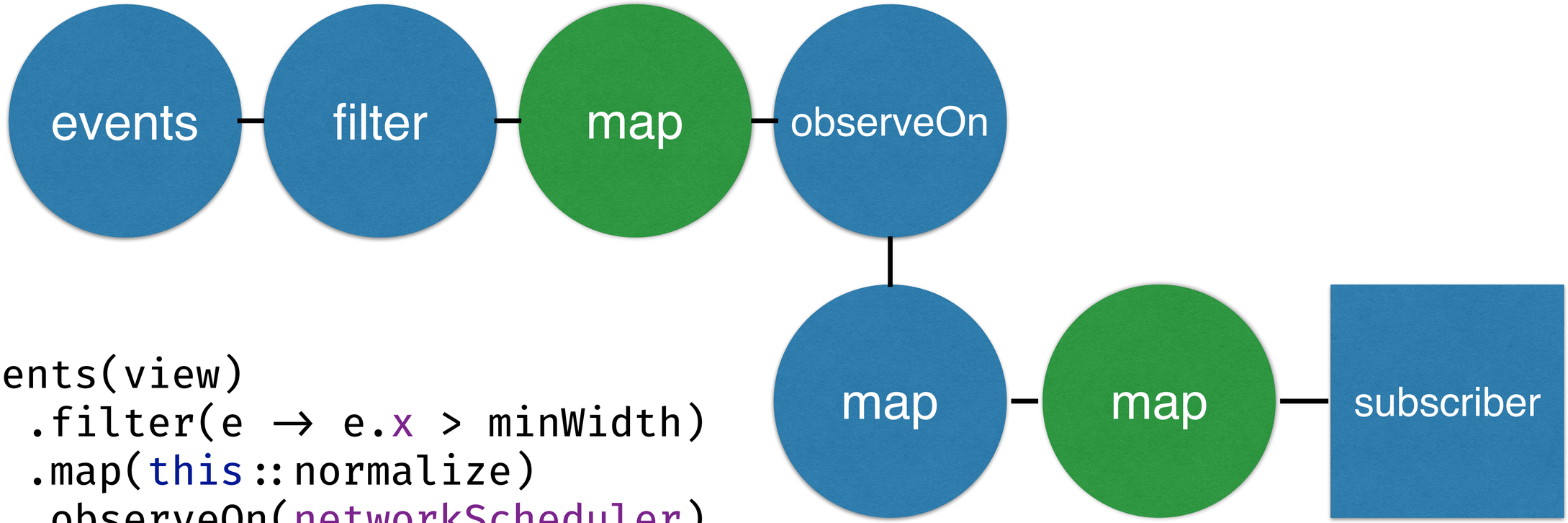

```
touchEvents(view)
        .filter(e → e.x > minWidth)
        .map(this::normalize)
        .observeOn(networkScheduler)
        .map(this::prepareForSending)
        .map(this::doMotionRequest)
        .subscribe()
```



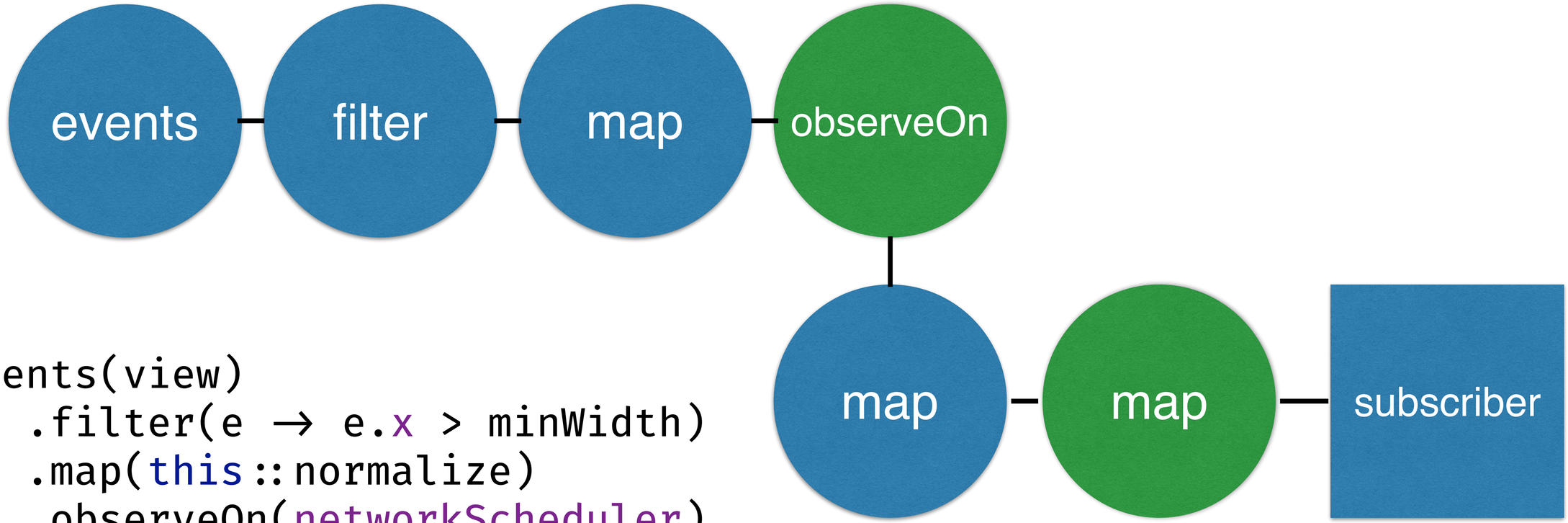

```
touchEvents(view)
        .filter(e → e.x > minWidth)
        .map(this::normalize)
        .observeOn(networkScheduler)
        .map(this::prepareForSending)
        .map(this::doMotionRequest)
        .subscribe()
```



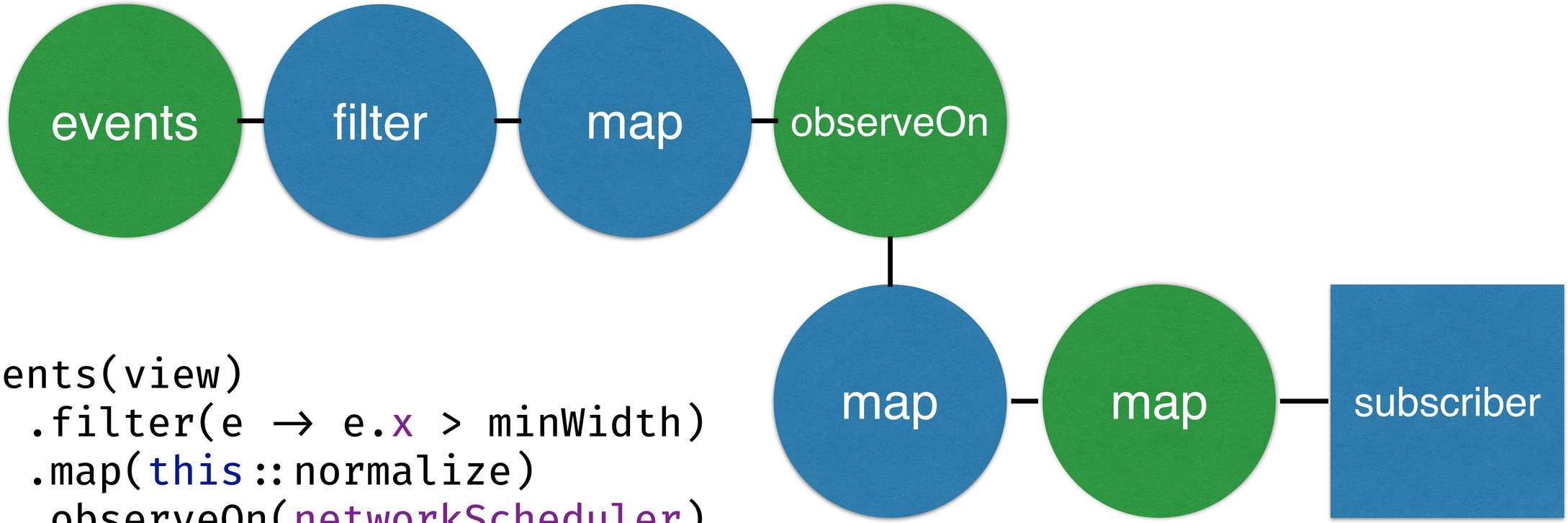

```
touchEvents(view)
        .filter(e → e.x > minWidth)
        .map(this::normalize)
        .observeOn(networkScheduler)
        .map(this::prepareForSending)
        .map(this::doMotionRequest)
        .subscribe()
```



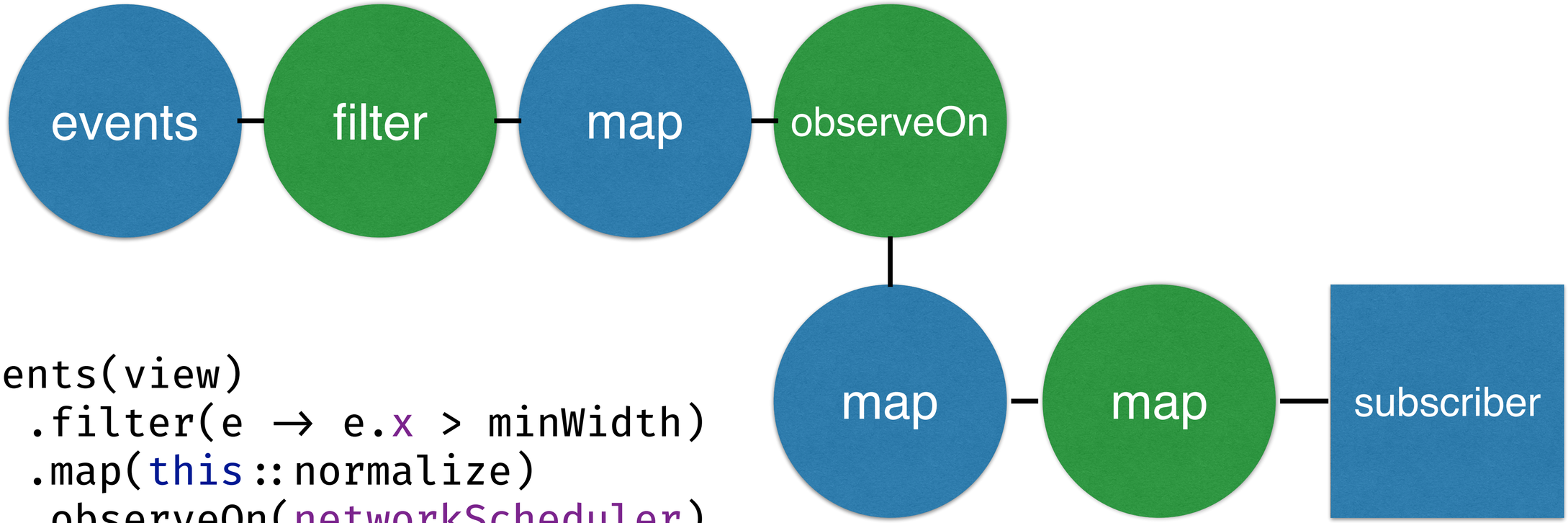

```
touchEvents(view)
        .filter(e → e.x > minWidth)
        .map(this::normalize)
        .observeOn(networkScheduler)
        .map(this::prepareForSending)
        .map(this::doMotionRequest)
        .subscribe()
```



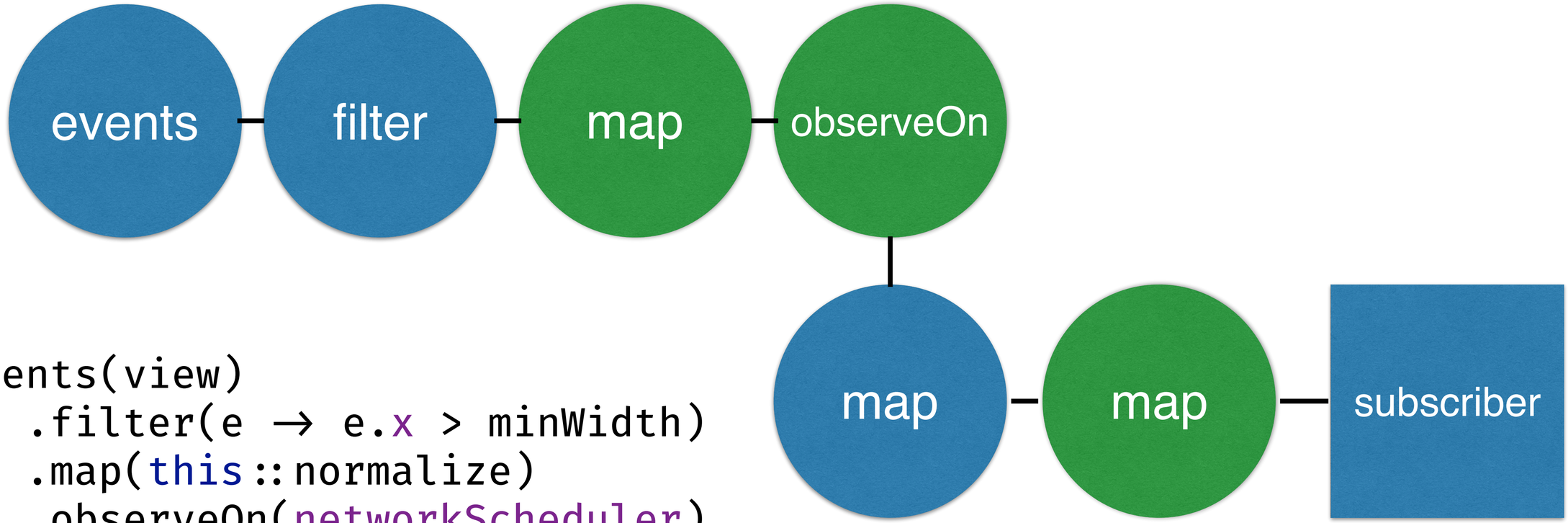

```
touchEvents(view)
        .filter(e → e.x > minWidth)
        .map(this::normalize)
        .observeOn(networkScheduler)
        .map(this::prepareForSending)
        .map(this::doMotionRequest)
        .subscribe()
```




```
touchEvents(view)
        .filter(e → e.x > minWidth)
        .map(this::normalize)
        .observeOn(networkScheduler)
        .map(this::prepareForSending)
        .map(this::doMotionRequest)
        .subscribe()
```

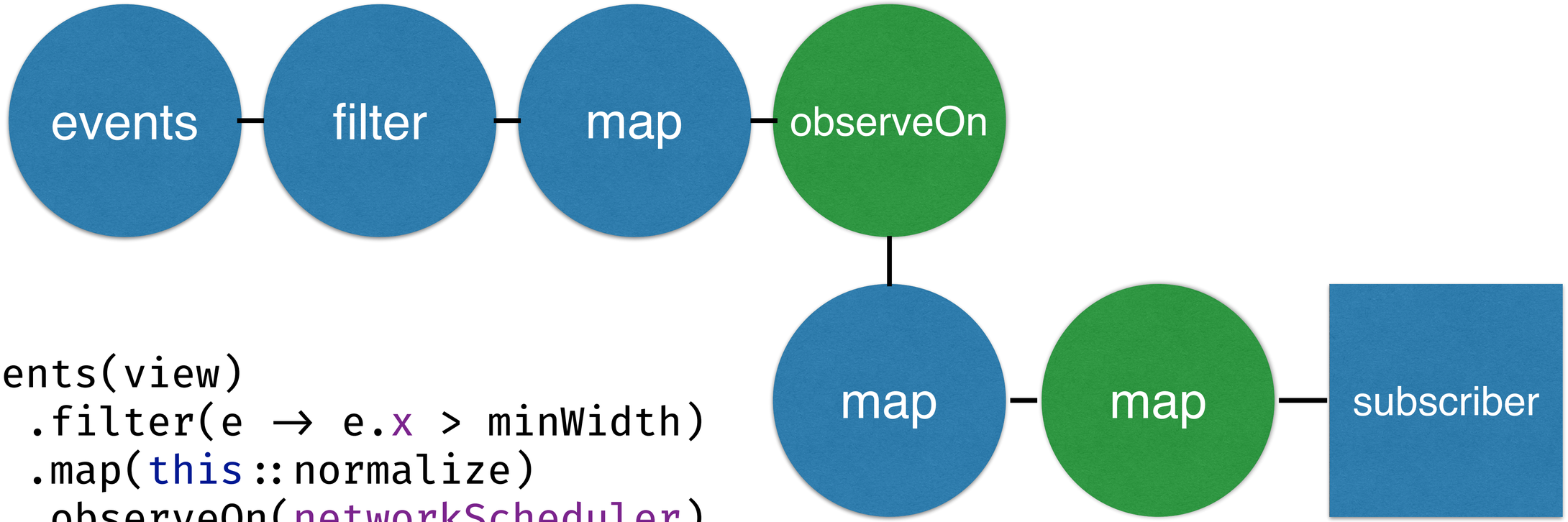



```
touchEvents(view)
        .filter(e → e.x > minWidth)
        .map(this::normalize)
        .observeOn(networkScheduler)
        .map(this::prepareForSending)
        .map(this::doMotionRequest)
        .subscribe()
```



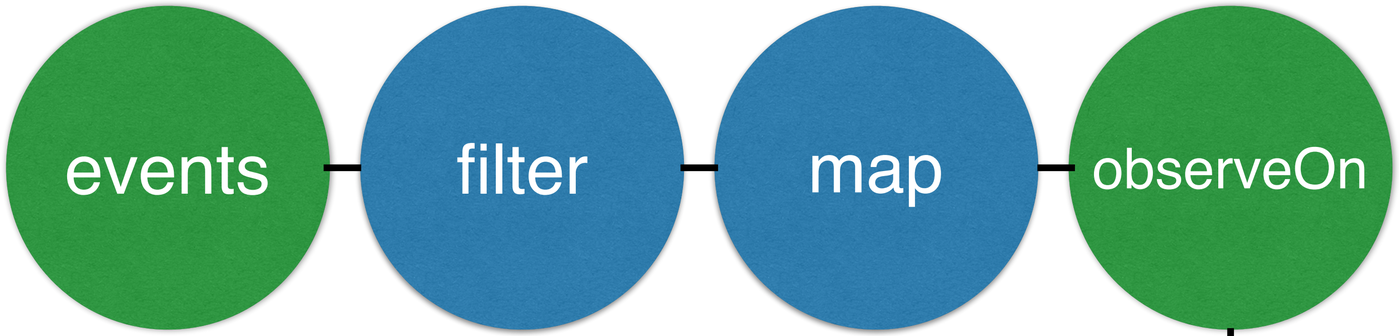

```
touchEvents(view)
        .filter(e → e.x > minWidth)
        .map(this::normalize)
        .observeOn(networkScheduler)
        .map(this::prepareForSending)
        .map(this::doMotionRequest)
        .subscribe()
```

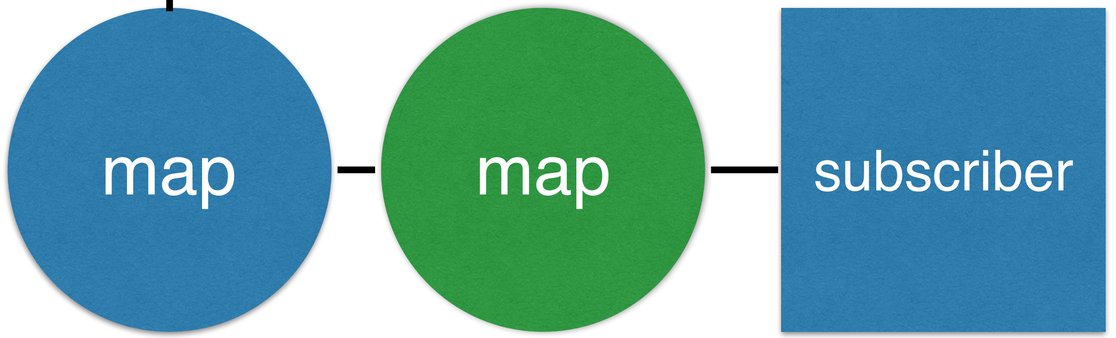
2




```
touchEvents(view)
        .filter(e → e.x > minWidth)
        .map(this::normalize)
        .observeOn(networkScheduler)
        .map(this::prepareForSending)
        .map(this::doMotionRequest)
        .subscribe()
```

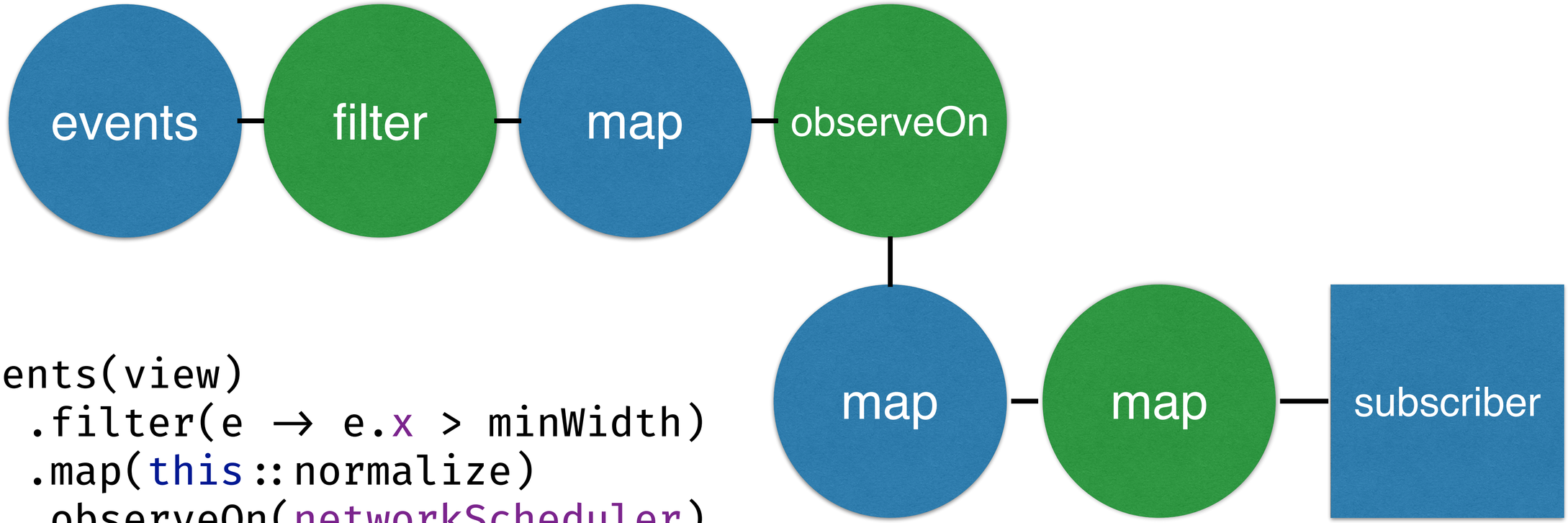
2





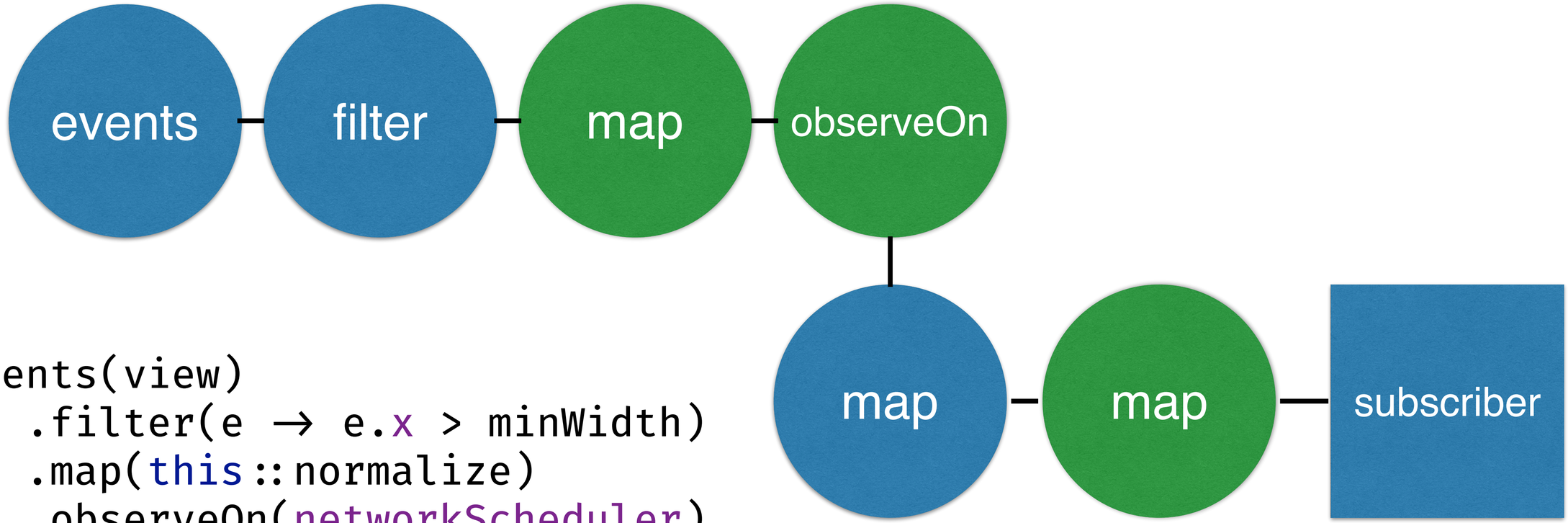

```
touchEvents(view)
        .filter(e → e.x > minWidth)
        .map(this::normalize)
        .observeOn(networkScheduler)
        .map(this::prepareForSending)
        .map(this::doMotionRequest)
        .subscribe()
```

2



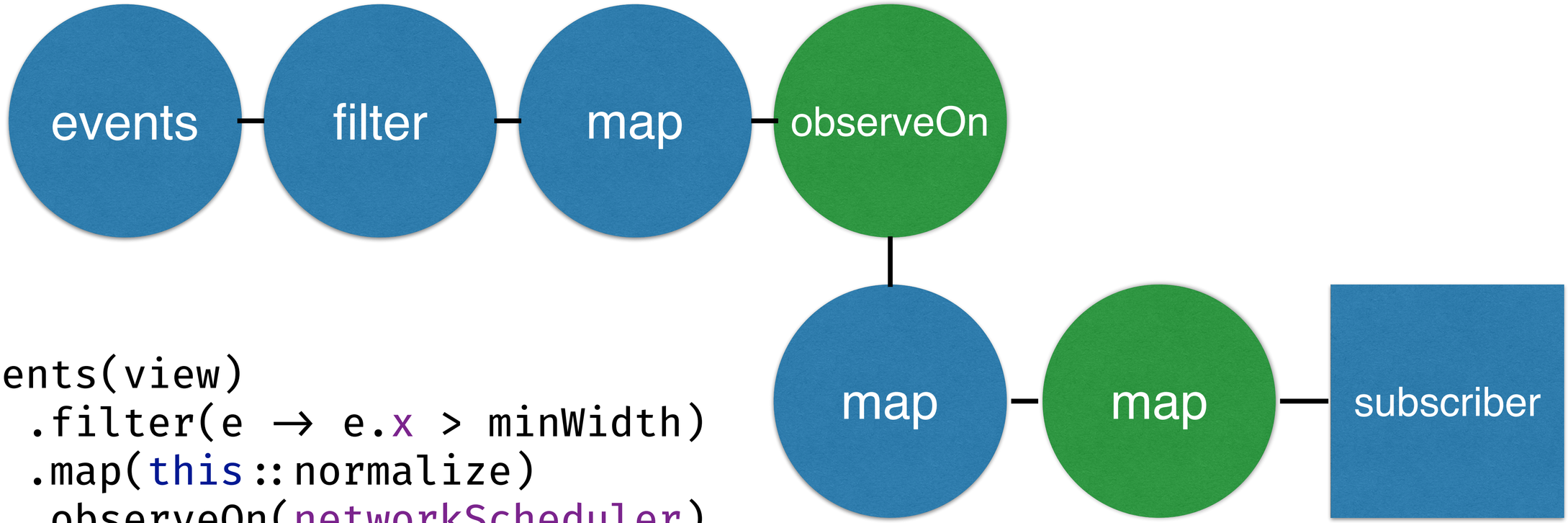

```
touchEvents(view)
        .filter(e → e.x > minWidth)
        .map(this::normalize)
        .observeOn(networkScheduler)
        .map(this::prepareForSending)
        .map(this::doMotionRequest)
        .subscribe()
```

2



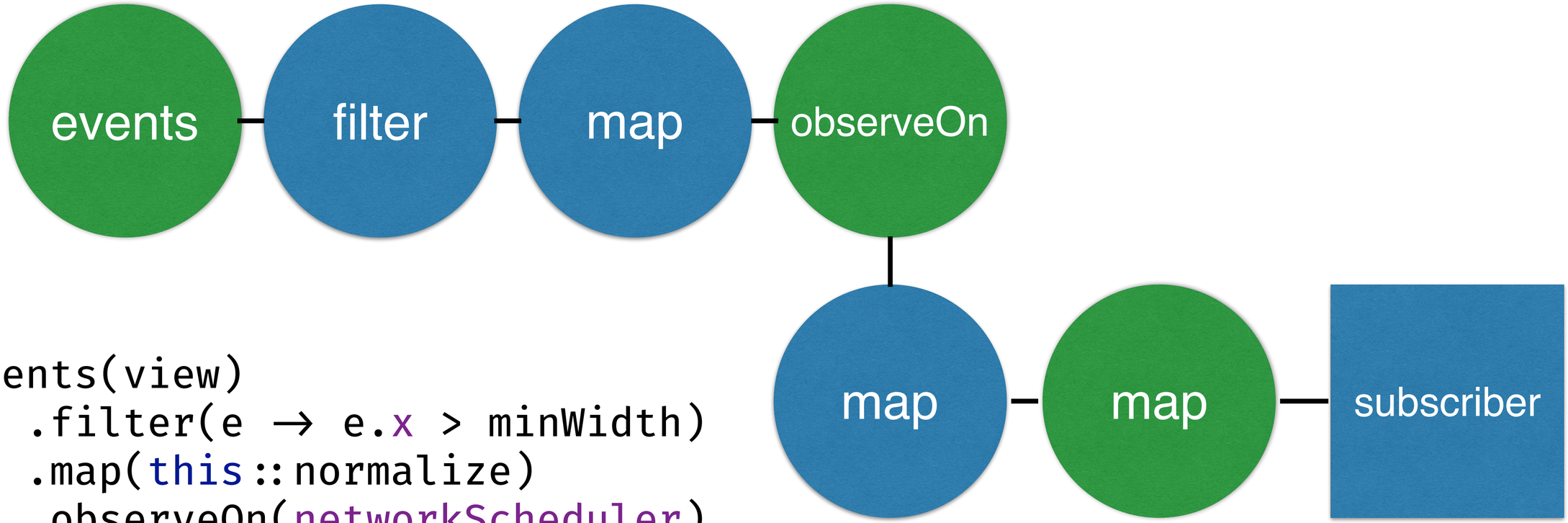

```
touchEvents(view)
        .filter(e → e.x > minWidth)
        .map(this::normalize)
        .observeOn(networkScheduler)
        .map(this::prepareForSending)
        .map(this::doMotionRequest)
        .subscribe()
```

3



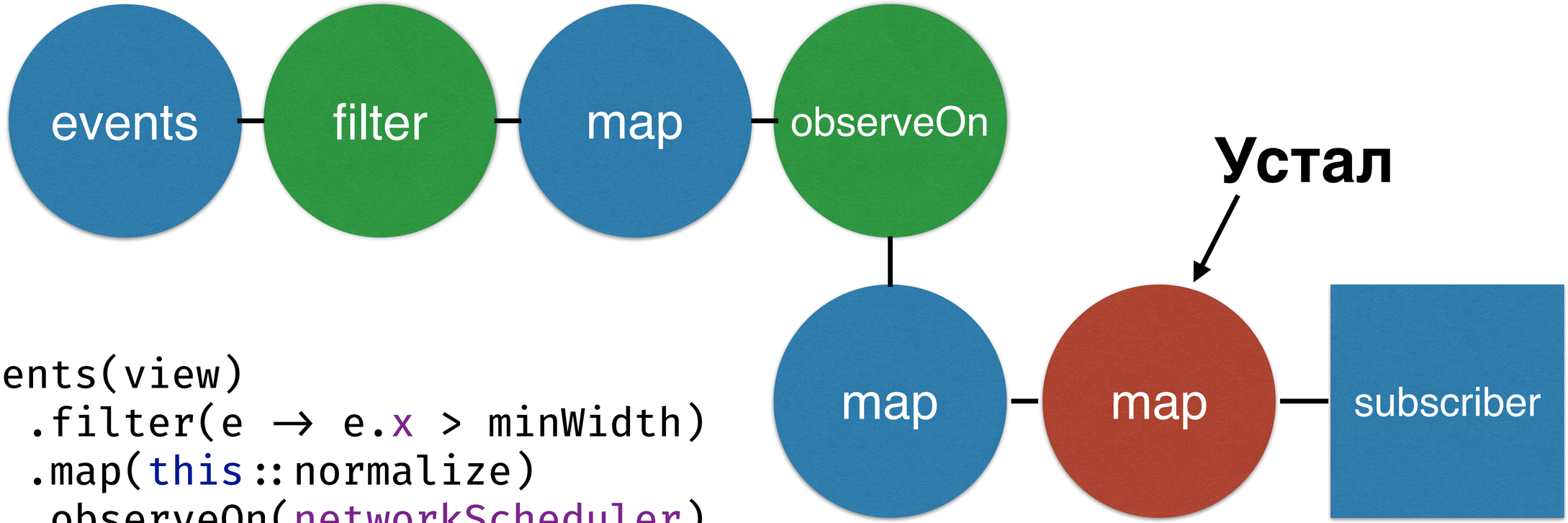

```
touchEvents(view)
        .filter(e → e.x > minWidth)
        .map(this::normalize)
        .observeOn(networkScheduler)
        .map(this::prepareForSending)
        .map(this::doMotionRequest)
        .subscribe()
```

3



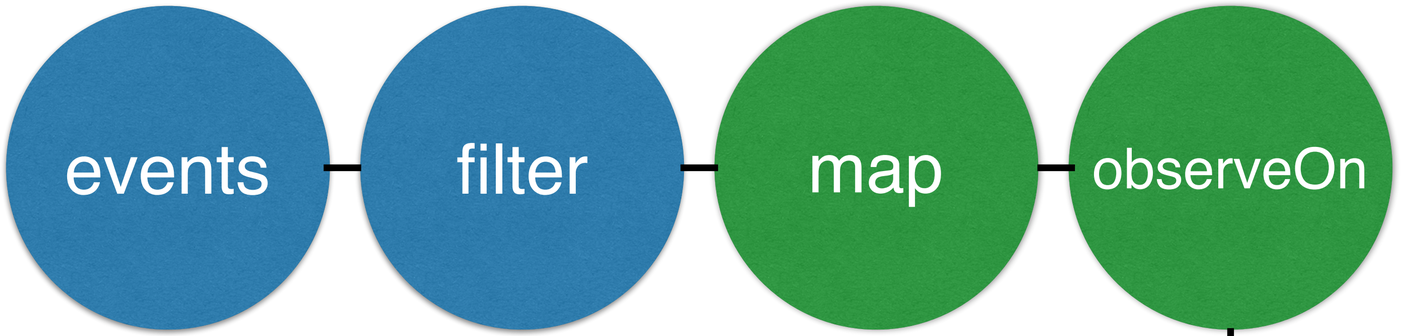

```
touchEvents(view)
        .filter(e → e.x > minWidth)
        .map(this::normalize)
        .observeOn(networkScheduler)
        .map(this::prepareForSending)
        .map(this::doMotionRequest)
        .subscribe()
```

3

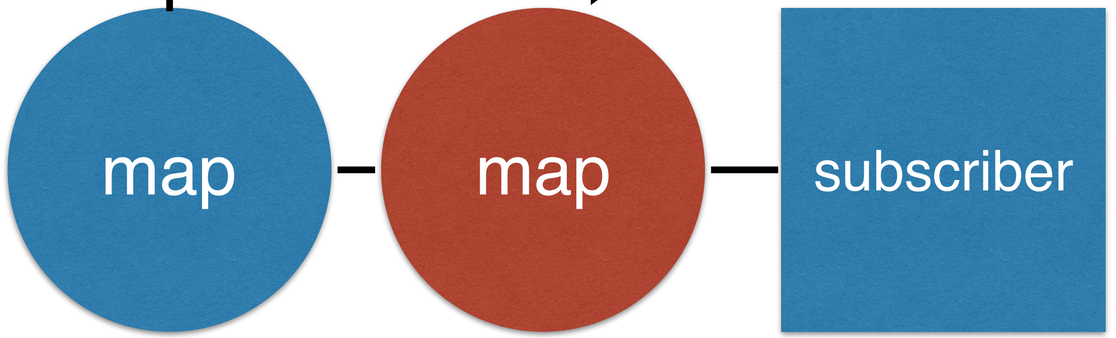



```
touchEvents(view)
        .filter(e → e.x > minWidth)
        .map(this::normalize)
        .observeOn(networkScheduler)
        .map(this::prepareForSending)
        .map(this::doMotionRequest)
        .subscribe()
```

3

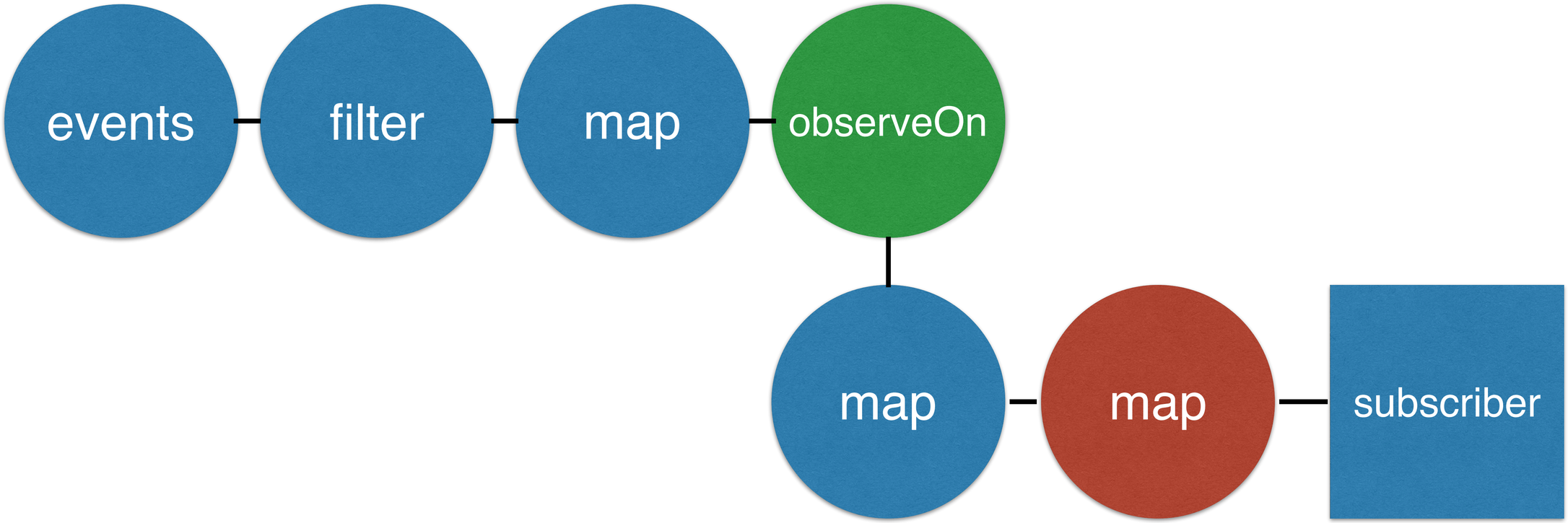


Устал




```
touchEvents(view)
        .filter(e → e.x > minWidth)
        .map(this::normalize)
        .observeOn(networkScheduler)
        .map(this::prepareForSending)
        .map(this::doMotionRequest)
        .subscribe()
```

4



rx.exceptions.MissingBackpressureException

```
Observable.zip(
        Observable.interval(10, TimeUnit.MILLISECONDS),
        Observable.interval(20, TimeUnit.MILLISECONDS),
        (r1, r2) → r1 * r2
).subscribe();
```

```
Observable.zip(
        Observable.interval(10, TimeUnit.MILLISECONDS),
        Observable.interval(20, TimeUnit.MILLISECONDS),
        (r1, r2) → r1 * r2
).subscribe();
```

```
Observable.zip(
        Observable.interval(10, TimeUnit.MILLISECONDS),
        Observable.interval(20, TimeUnit.MILLISECONDS),
        (r1, r2) → r1 * r2
).subscribe();
```

```
Observable.zip(
        Observable.interval(10, TimeUnit.MILLISECONDS),
        Observable.interval(20, TimeUnit.MILLISECONDS),
        (r1, r2) → r1 * r2
).subscribe();
```

```
Observable.zip(
        Observable.interval(10, TimeUnit.MILLISECONDS),
        Observable.interval(20, TimeUnit.MILLISECONDS),
        (r1, r2) → r1 * r2
).subscribe();
```

rx.exceptions.MissingBackpressureException

Борьба с Backpressure

Борьба с Backpressure

- debounce, throttle

Борьба с Backpressure

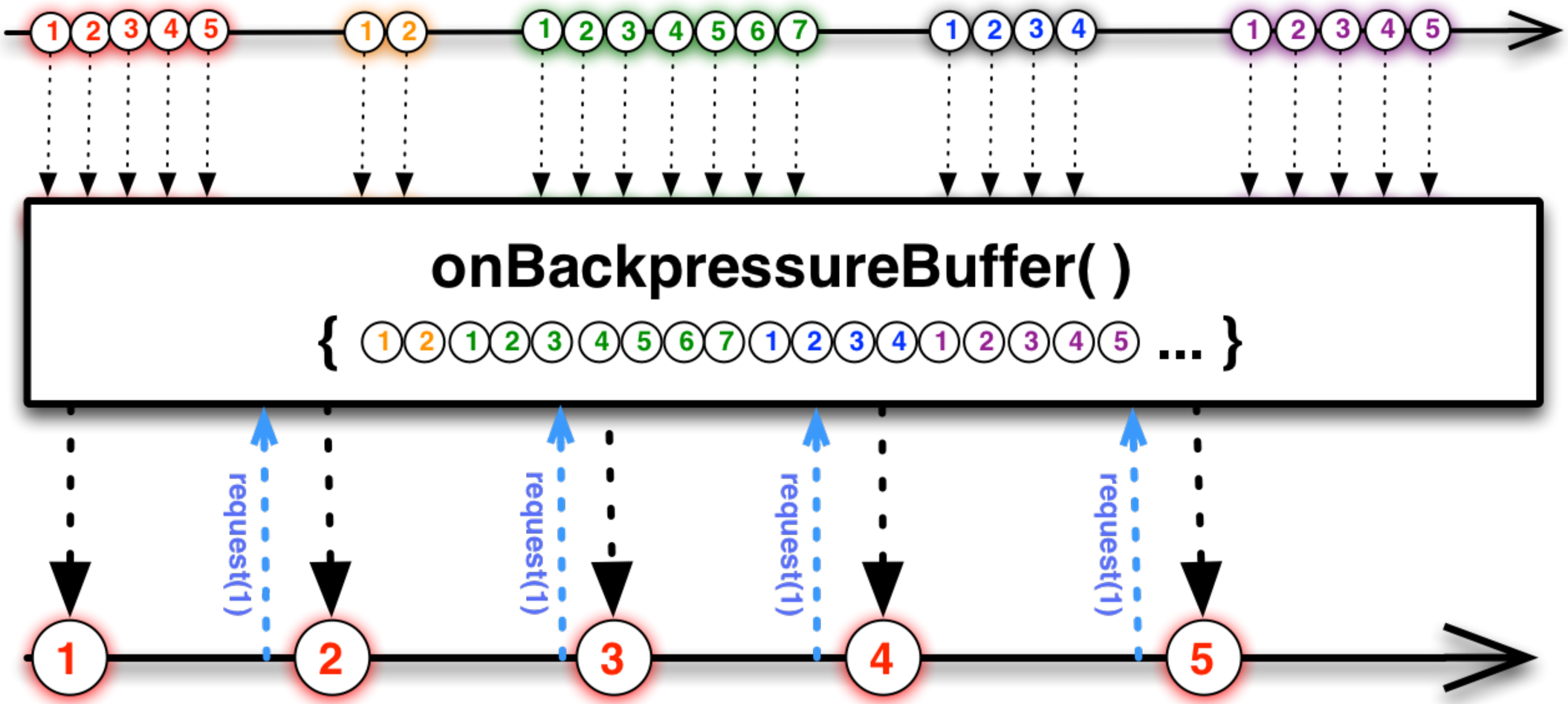
- debounce, throttle
- window, buffer

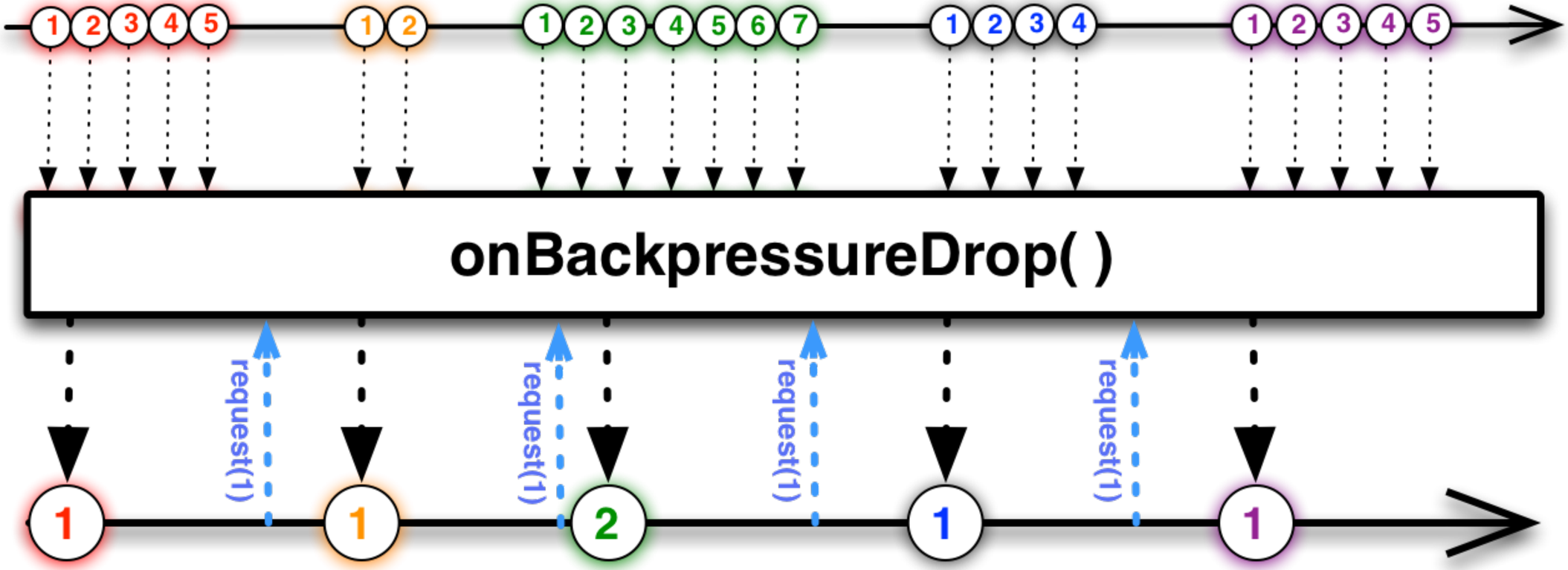
Борьба с Backpressure

- debounce, throttle
- window, buffer
- onBackpressureBuffer

Борьба с Backpressure

- debounce, throttle
- window, buffer
- onBackpressureBuffer
- onBackpressureDrop


onBackpressureBuffer( )
{ 1 2 1 2 3 4 5 6 7 1 2 3 4 1 2 3 4 5 ... }
request(1)
request(1)
request(1)
request(1)
1
2
3
4
5


1 2 3 4 5
1 2
1 2 3 4 5 6 7
1 2 3 4
1 2 3 4 5
onBackpressureDrop( )
request(1)
request(1)
request(1)
request(1)
1
1
2
1
1

Всегда думайте о том, сколько элементов вам нужно от Observable

Как думать, используя RxJava?

Как инженер

Как инженер

Как инженер

- не бойся расширять фреймворк

Как инженер

- не бойся расширять фреймворк
- понимай, как он работает внутри

DoOnNextAsync

DoOnNextAsync

- doOnNext == сайд эффект

DoOnNextAsync

- doOnNext == сайд эффект
- не блокировать выполнение родителя

DoOnNextAsync

- doOnNext == сайд эффект
- не блокировать выполнение родителя
- был привязан к жизненному циклу родителя

DoOnNextAsync

- doOnNext == сайд эффект
- не блокировать выполнение родителя
- был привязан к жизненному циклу родителя
- не через Subjects

```
userDbChanges
        .map(this::doRequest)
        .doOnNext(resp → {
            globalSubscription = createCacheObs(resp)
                    .subscribe();
        })
        .map(this::doAnotherRequest);
```

```
userDbChanges
        .map(this::doRequest)
        .doOnNext(resp → {
            globalSubscription = createCacheObs(resp)
                    .subscribe();
        })
        .map(this::doAnotherRequest);
```

```
userDbChanges
        .map(this::doRequest)
        .lift(new DoOnNextAsync<>(this::cacheObs))
        .map(this::doAnotherRequest);
```

```
userDbChanges
        .map(this::doRequest)
        .lift(new DoOnNextAsync<>(this::cacheObs))
        .map(this::doAnotherRequest);
```

```
class DoOnNextAsync<F, T> implements Observable.Operator<F, F> {
    private final Func1<F, Observable<T>> nested;

    public DoOnNextAsync(Func1<F, Observable<T>> func) {
        this.nested = func;
    }

    //больше кода дальше
```

```
class DoOnNextAsync<F, T> implements Observable.Operator<F, F> {
    private final Func1<F, Observable<T>> nested;

    public DoOnNextAsync(Func1<F, Observable<T>> func) {
        this.nested = func;
    }

    //больше кода дальше
```

```
class DoOnNextAsync<F, T> implements Observable.Operator<F, F> {
    private final Func1<F, Observable<T>> nested;

    public DoOnNextAsync(Func1<F, Observable<T>> func) {
        this.nested = func;
    }

    //больше кода дальше
```

```
@Override
public Subscriber<? super F> call(Subscriber<? super F> subscriber) {
    return new Subscriber<F>() {

        @Override
        public void onNext(F next) {
            if (!subscriber.isUnsubscribed()) {
                Subscription s = nested.call(next).subscribe();
                subscriber.add(s);
            }
        }

        //тривиальные onError и onCompleted здесь
    };
}
```

```
@Override
public Subscriber<? super F> call(Subscriber<? super F> subscriber) {
    return new Subscriber<F>() {

        @Override
        public void onNext(F next) {
            if (!subscriber.isUnsubscribed()) {
                Subscription s = nested.call(next).subscribe();
                subscriber.add(s);
            }
        }

        //тривиальные onError и onCompleted здесь
    };
}
```

```
@Override
public Subscriber<? super F> call(Subscriber<? super F> subscriber) {
    return new Subscriber<F>() {

        @Override
        public void onNext(F next) {
            if (!subscriber.isUnsubscribed()) {
                Subscription s = nested.call(next).subscribe();
                subscriber.add(s);
            }
        }

        //тривиальные onError и onCompleted здесь
    };
}
```

```
@Override
public Subscriber<? super F> call(Subscriber<? super F> subscriber) {
    return new Subscriber<F>() {

        @Override
        public void onNext(F next) {
            if (!subscriber.isUnsubscribed()) {
                Subscription s = nested.call(next).subscribe();
                subscriber.add(s);
            }
        }

        //тривиальные onError и onCompleted здесь
    };
}
```

```
@Override
public Subscriber<? super F> call(Subscriber<? super F> subscriber) {
    return new Subscriber<F>() {

        @Override
        public void onNext(F next) {
            if (!subscriber.isUnsubscribed()) {
                Subscription s = nested.call(next).subscribe();
                subscriber.add(s);
            }
        }

        //тривиальные onError и onCompleted здесь
    };
}
```

Понимай, как работает внутри

- Operator, lift,
- subscribeOn, observeOn
- nest, defer, create

Думай реактивно

Думать реактивно

Думать реактивно

- соединять императивный и реактивный мир

Думать реактивно

- соединять императивный и реактивный мир
- не использовать сабджекты

Думать реактивно

- соединять императивный и реактивный мир
- не использовать сабджекты
- описывать, «как делать», а не «что делать»

AutoLoadingRecyclerView

AutoLoadingRecyclerView

- научить RecycerView автоподгрузке

AutoLoadingRecyclerView

- научить RecycerView автоподгрузке
- сделать с помощью RxJava

AutoLoadingRecyclerView

- научить RecycerView автоподгрузке
- сделать с помощью RxJava
- в реактивном стиле

```
public class AutoLoadingRecyclerView<T> extends RecyclerView {

    protected PublishSubject<Pair<Integer, Integer>> scrollLoadingChannel =
            PublishSubject.create();

    protected void startScrollingChannel() {
        addOnScrollListener(new RecyclerView.OnScrollListener() {
            @Override
            public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
                if (shouldLoad()) {
                    int offset = getAdapter().getItemCount();
                    scrollLoadingChannel.onNext(new Pair(offset, limit));
                }
            }
        });
    }
```

```
public class AutoLoadingRecyclerView<T> extends RecyclerView {

    protected PublishSubject<Pair<Integer, Integer>> scrollLoadingChannel =
            PublishSubject.create();

    protected void startScrollingChannel() {
        addOnScrollListener(new RecyclerView.OnScrollListener() {
            @Override
            public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
                if (shouldLoad()) {
                    int offset = getAdapter().getItemCount();
                    scrollLoadingChannel.onNext(new Pair(offset, limit));
                }
            }
        });
    }
```

```
public class AutoLoadingRecyclerView<T> extends RecyclerView {

    protected PublishSubject<Pair<Integer, Integer>> scrollLoadingChannel =
            PublishSubject.create();

    protected void startScrollingChannel() {
        addOnScrollListener(new RecyclerView.OnScrollListener() {
            @Override
            public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
                if (shouldLoad()) {
                    int offset = getAdapter().getItemCount();
                    scrollLoadingChannel.onNext(new Pair(offset, limit));
                }
            }
        });
    }
```

```
public class AutoLoadingRecyclerView<T> extends RecyclerView {

    protected PublishSubject<Pair<Integer, Integer>> scrollLoadingChannel =
            PublishSubject.create();

    protected void startScrollingChannel() {
        addOnScrollListener(new RecyclerView.OnScrollListener() {
            @Override
            public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
                if (shouldLoad()) {
                    int offset = getAdapter().getItemCount();
                    scrollLoadingChannel.onNext(new Pair(offset, limit));
                }
            }
        });
    }
```

```
public class AutoLoadingRecyclerView<T> extends RecyclerView {

    protected PublishSubject<Pair<Integer, Integer>> scrollLoadingChannel =
            PublishSubject.create();

    protected void startScrollingChannel() {
        addOnScrollListener(new RecyclerView.OnScrollListener() {
            @Override
            public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
                if (shouldLoad()) {
                    int offset = getAdapter().getItemCount();
                    scrollLoadingChannel.onNext(new Pair(offset, limit));
                }
            }
        });
    }
```

```
public class AutoLoadingRecyclerView<T> extends RecyclerView {

    protected PublishSubject<OffsetAndLimit> scrollLoadingChannel =
            PublishSubject.create();
    protected Subscription loadNewItemsSubscription;
    protected Subscription subscribeToLoadingChannelSubscription;
```

```
public class AutoLoadingRecyclerView<T> extends RecyclerView {

    protected PublishSubject<OffsetAndLimit> scrollLoadingChannel =
            PublishSubject.create();
    protected Subscription loadNewItemsSubscription;
    protected Subscription subscribeToLoadingChannelSubscription;
```

```
public class AutoLoadingRecyclerView<T> extends RecyclerView {

    protected PublishSubject<OffsetAndLimit> scrollLoadingChannel =
            PublishSubject.create();
    protected Subscription loadNewItemsSubscription;
    protected Subscription subscribeToLoadingChannelSubscription;
```

```
private static Observable<Integer>
getScrollObservable(RecyclerView recyclerView, int limit) {

    return Observable.create(subscriber → {
        final RecyclerView.OnScrollListener sl = new RecyclerView.OnScrollListener() {
            @Override
            public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
                if (!subscriber.isUnsubscribed() && shouldLoad()) {
                    subscriber.onNext(recyclerView.getAdapter().getItemCount());
                }
            }
        };
        recyclerView.addOnScrollListener(sl);
        subscriber.add(Subscriptions.create(() → recyclerView.removeOnScrollListener(sl)));
    });
}
```

```
private static Observable<Integer>
getScrollObservable(RecyclerView recyclerView, int limit) {

    return Observable.create(subscriber → {
        final RecyclerView.OnScrollListener sl = new RecyclerView.OnScrollListener() {
            @Override
            public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
                if (!subscriber.isUnsubscribed() && shouldLoad()) {
                    subscriber.onNext(recyclerView.getAdapter().getItemCount());
                }
            }
        };
        recyclerView.addOnScrollListener(sl);
        subscriber.add(Subscriptions.create(() → recyclerView.removeOnScrollListener(sl)));
    });
}
```

```
private static Observable<Integer>
getScrollObservable(RecyclerView recyclerView, int limit) {

    return Observable.create(subscriber → {
        final RecyclerView.OnScrollListener sl = new RecyclerView.OnScrollListener() {
            @Override
            public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
                if (!subscriber.isUnsubscribed() && shouldLoad()) {
                    subscriber.onNext(recyclerView.getAdapter().getItemCount());
                }
            }
        };
        recyclerView.addOnScrollListener(sl);
        subscriber.add(Subscriptions.create(() → recyclerView.removeOnScrollListener(sl)));
    });
}
```

```
private static Observable<Integer>
getScrollObservable(RecyclerView recyclerView, int limit) {

    return Observable.create(subscriber → {
        final RecyclerView.OnScrollListener sl = new RecyclerView.OnScrollListener() {
            @Override
            public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
                if (!subscriber.isUnsubscribed() && shouldLoad()) {
                    subscriber.onNext(recyclerView.getAdapter().getItemCount());
                }
            }
        };
        recyclerView.addOnScrollListener(sl);
        subscriber.add(Subscriptions.create(() → recyclerView.removeOnScrollListener(sl)));
    });
}
```

```
private static Observable<Integer>
getScrollObservable(RecyclerView recyclerView, int limit) {

    return Observable.create(subscriber → {
        final RecyclerView.OnScrollListener sl = new RecyclerView.OnScrollListener() {
            @Override
            public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
                if (!subscriber.isUnsubscribed() && shouldLoad()) {
                    subscriber.onNext(recyclerView.getAdapter().getItemCount());
                }
            }
        };
        recyclerView.addOnScrollListener(sl);
        subscriber.add(Subscriptions.create(() → recyclerView.removeOnScrollListener(sl)));
    });
}
```

```
private static Observable<Integer>
getScrollObservable(RecyclerView recyclerView, int limit) {

    return Observable.create(subscriber → {
        final RecyclerView.OnScrollListener sl = new RecyclerView.OnScrollListener() {
            @Override
            public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
                if (!subscriber.isUnsubscribed() && shouldLoad()) {
                    subscriber.onNext(recyclerView.getAdapter().getItemCount());
                }
            }
        };
        recyclerView.addOnScrollListener(sl);
        subscriber.add(Subscriptions.create(() → recyclerView.removeOnScrollListener(sl)));
    });
}
```

```
getScrollObservable()
        .distinctUntilChanged()
        .switchMap(offset → getLoadingObservable(offset))
        .subscribeOn(Schedulers
                .from(BackgroundExecutor.getSafeBackgroundExecutor())
        )
        .observeOn(AndroidSchedulers.mainThread())
        .subscribe(loadNewItemsSubscriber);
```

```
getScrollObservable()
        .distinctUntilChanged()
        .switchMap(offset → getLoadingObservable(offset))
        .subscribeOn(Schedulers
                .from(BackgroundExecutor.getSafeBackgroundExecutor())
        )
        .observeOn(AndroidSchedulers.mainThread())
        .subscribe(loadNewItemsSubscriber);
```

```
getScrollObservable()
        .distinctUntilChanged()
        .switchMap(offset → getLoadingObservable(offset))
        .subscribeOn(Schedulers
                .from(BackgroundExecutor.getSafeBackgroundExecutor())
        )
        .observeOn(AndroidSchedulers.mainThread())
        .subscribe(loadNewItemsSubscriber);
```

```
getScrollObservable()
        .distinctUntilChanged()
        .switchMap(offset → getLoadingObservable(offset))
        .subscribeOn(Schedulers
                .from(BackgroundExecutor.getSafeBackgroundExecutor())
        )
        .observeOn(AndroidSchedulers.mainThread())
        .subscribe(loadNewItemsSubscriber);
```

```
getScrollObservable()
        .distinctUntilChanged()
        .switchMap(offset → getLoadingObservable(offset))
        .subscribeOn(Schedulers
                .from(BackgroundExecutor.getSafeBackgroundExecutor())
        )
        .observeOn(AndroidSchedulers.mainThread())
        .subscribe(loadNewItemsSubscriber);
```

```
getScrollObservable()
        .distinctUntilChanged()
        .switchMap(offset → getLoadingObservable(offset))
        .subscribeOn(Schedulers
                .from(BackgroundExecutor.getSafeBackgroundExecutor())
        )
        .observeOn(AndroidSchedulers.mainThread())
        .subscribe(loadNewItemsSubscriber);
```

Думать реактивно

Думать реактивно

- использовать стандартные средства RxJava

Думать реактивно

- использовать стандартные средства RxJava
  - но не сабджекты

Думать реактивно

- использовать стандартные средства RxJava
  - но не сабджекты
  - при необходимости — писать свое

Думать реактивно

- использовать стандартные средства RxJava
  - но не сабджекты
  - при необходимости — писать свое
- оборачивать состояние в Observable

Думать реактивно

- использовать стандартные средства RxJava
  - но не сабджекты
  - при необходимости — писать свое
- оборачивать состояние в Observable
- стараться и не сдаваться

The Art of Rx

The Art of Rx

- думать о

The Art of Rx

- думать о
  - о контрактах

The Art of Rx

- думать о
  - о контрактах
  - о тредах

The Art of Rx

- думать о
  - о контрактах
  - о тредах
  - о количестве элементов

The Art of Rx

- думать о
  - о контрактах
  - о тредах
  - о количестве элементов
- думать как

The Art of Rx

- думать о
  - о контрактах
  - о тредах
  - о количестве элементов
- думать как
  - инженер

The Art of Rx

- думать о
  - о контрактах
  - о тредах
  - о количестве элементов
- думать как
  - инженер
  - реактивно

Стать профессионалом — непросто

Не ленитесь

Думайте много

Думайте много
О важных вещах

Спасибо

Мальков Матвей

matveyka_jj

Q & A