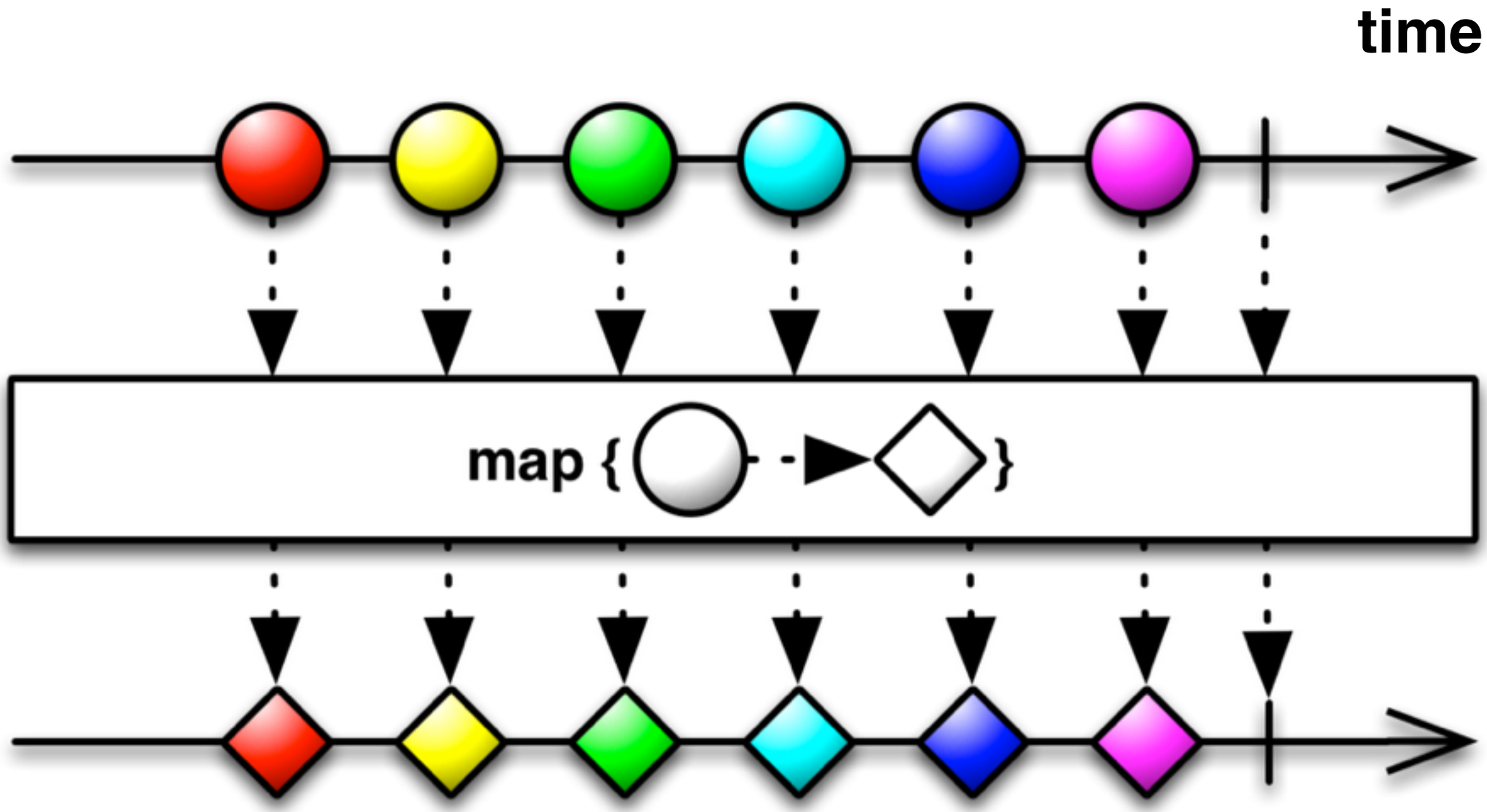
# Reactive demystified

# Что это?

- Способ программирования, оперирующий со специальными типами данных, изменяющимися в зависимости от времени (сигналами)
- Сигналы - последовательности значений, к которым можно применить функциональные преобразования

# Что это?

- Способ программирования, оперирующий со специальными типами данных, изменяющимися в зависимости от времени (сигналами)
- Сигналы - последовательности значений, к которым можно применить функциональные преобразования


time
map {
}

«1234»

Сигнал - контейнер

time

map { - }

1234

# Расчет суммы займа

```
loanAmountTextfield.throttle(0.5)
                   .distinctUntilChanged()
                   .map(calculatedLoanAmount)
                   .bindTo(amountReturnedLabel)
```

# Расчет суммы займа

```
loanAmountTextfield.throttle(0.5)
                   .distinctUntilChanged()
                   .map(calculatedLoanAmount)
                   .bindTo(amountReturnedLabel)
```

# Расчет суммы займа

```
loanAmountTextfield.throttle(0.5)
                   .distinctUntilChanged()
                   .map(calculatedLoanAmount)
                   .bindTo(amountReturnedLabel)
```

# Мотивация

- Абстрагируемся от реализации механизма отображения (UI)
- Используем функциональный аппарат для работы с последовательностями сигналов
- Не храним состояние
- Избавляемся от вложенных колбеков
- Описываем код декларативно (что сделать, а не как)

# Мотивация

- Абстрагируемся от реализации механизма отображения (UI)
- Используем функциональный аппарат для работы с последовательностями сигналов
- Не храним состояние
- Избавляемся от вложенных колбеков
- Описываем код декларативно (что сделать, а не как)

# Мотивация

- Абстрагируемся от реализации механизма отображения (UI)
- Используем функциональный аппарат для работы с последовательностями сигналов
- Не храним состояние
- Избавляемся от вложенных колбеков
- Описываем код декларативно (что сделать, а не как)

# Мотивация

- Абстрагируемся от реализации механизма отображения (UI)
- Используем функциональный аппарат для работы с последовательностями сигналов
- Не храним состояние
- Избавляемся от вложенных колбеков
- Описываем код декларативно (что сделать, а не как)

# Мотивация

- Абстрагируемся от реализации механизма отображения (UI)
- Используем функциональный аппарат для работы с последовательностями сигналов
- Не храним состояние
- Избавляемся от вложенных колбеков
- Описываем код декларативно (что сделать, а не как)

# История

# Краткий обзор

- Functional Reactive Animations (FRAN - Haskell), Yampa
- Cells (Common Lisp), FrTime (Scheme)
- Rx (C# etc)
- Elm, React (Javascript), Reactive Banana (Haskell), Trellis (Python), ReactiveCocoa (Swift, Objc)

# Краткий обзор

- Functional Reactive Animations (FRAN - Haskell), Yampa
- Cells (Common Lisp), FrTime (Scheme)
- Rx (C# etc)
- Elm, React (Javascript), Reactive Banana (Haskell), Trellis (Python), ReactiveCocoa (Swift, Objc)

# Краткий обзор

- Functional Reactive Animations (FRAN - Haskell), Yampa
- Cells (Common Lisp), FrTime (Scheme)
- Rx (C# etc)
- Elm, React (Javascript), Reactive Banana (Haskell), Trellis (Python), ReactiveCocoa (Swift, Objc)

# Краткий обзор

- Functional Reactive Animations (FRAN - Haskell), Yampa
- Cells (Common Lisp), FrTime (Scheme)
- Rx (C# etc)
- Elm, React (Javascript), Reactive Banana (Haskell), Trellis (Python), ReactiveCocoa (Swift, Objc)

# Functional Reactive Animations

# FRAN

**Behaviour**

- Непрерывные (время, температура, ток)
- Интегрирование/ дифференцирование (скорость -> ускорение)
- Используются для анимаций, моделирования физических процессов и т.д.

**Event**

- Дискретные (сообщения от устройств ввода/ вывода)
- Можно применить различные функциональные операторы (map, filter, reduce)

# FRAN

**Behaviour**

- Непрерывные (время, температура,  ток)
- Интегрирование/ дифференцирование (скорость -> ускорение)
- Используются для анимаций, моделирования физических процессов и т.д.

**Event**

- Дискретные (сообщения от устройств ввода/ вывода)
- Можно применить различные функциональные операторы (map, filter, reduce)

# FRAN

**Behaviour**

- Непрерывные (время, температура,  ток)
- Интегрирование/ дифференцирование (скорость -> ускорение)
- **Используются для анимаций, моделирования физических процессов и т.д.**

**Event**

- Дискретные (сообщения от устройств ввода/ вывода)
- Можно применить различные функциональные операторы (map, filter, reduce)

# FRAN

**Behaviour**

- Непрерывные (время, температура, ток)
- Интегрирование/ дифференцирование (скорость -> ускорение)
- Используются для анимаций, моделирования физических процессов и т.д.

**Event**

- Дискретные (сообщения от устройств ввода/ вывода)
- Можно применить различные функциональные операторы (map, filter, reduce)

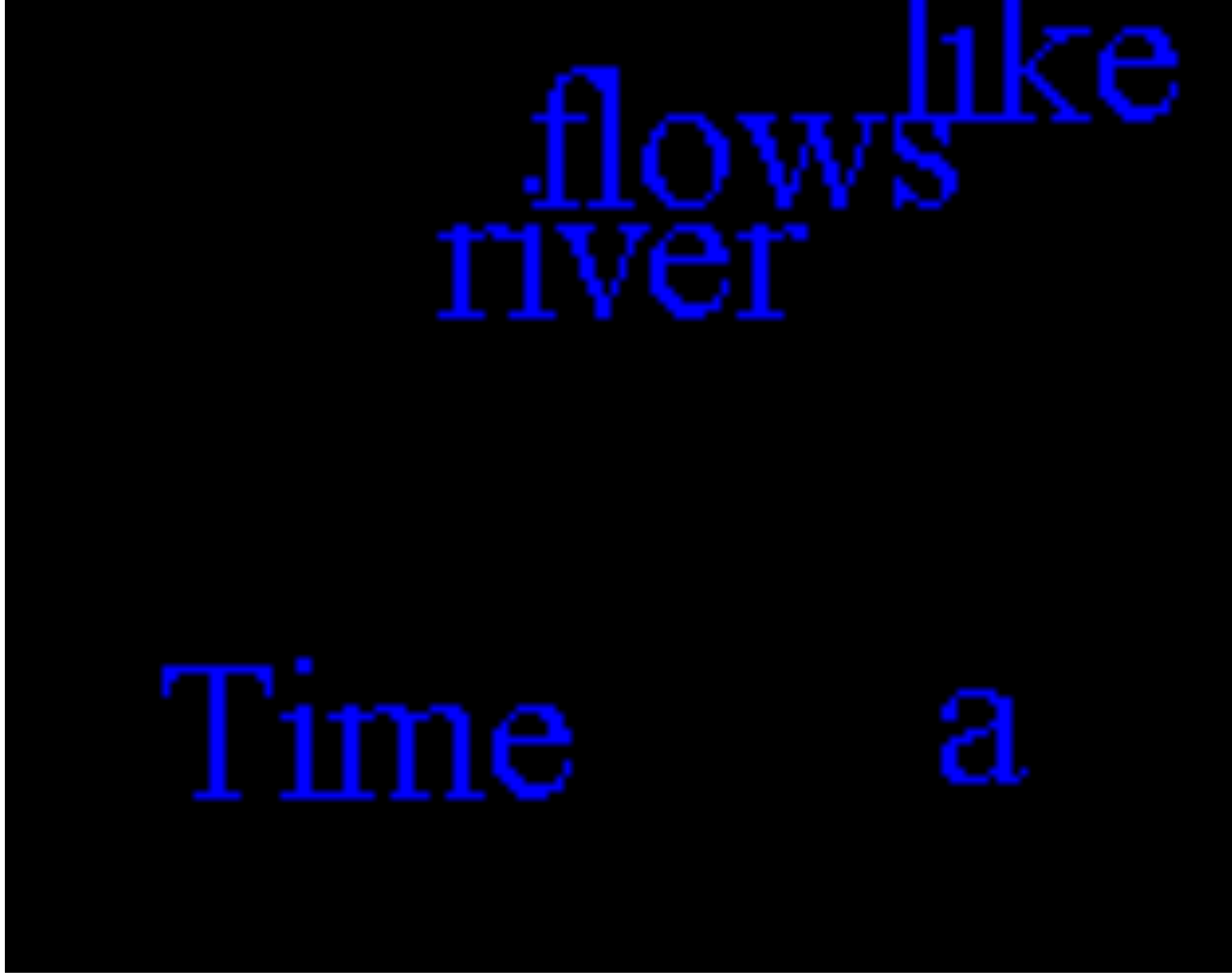
# FRAN

**Behaviour**

- Непрерывные (время, температура,  ток)
- Интегрирование/ дифференцирование (скорость -> ускорение)
- Используются для анимаций, моделирования физических процессов и т.д.

**Event**

- Дискретные (сообщения от устройств ввода/ вывода)
- Можно применить различные функциональные операторы (map, filter, reduce)




```
flows u = trailWords motion
          "Time flows like a river"
where
    motion = 0.7 *^ vector2XY(cos time)
                              (sin (2 * time))
```

# Проблемы

- **Space leaks**
- Time leaks

# Проблемы

- Space leaks
- **Time leaks**

# Метрики

# Метрики

- **Используемые абстракции**
- Модель передачи значений (push vs pull)
- Порядок вычислений (glitch avoidance)
- Существующие операторы (lifting)

# Метрики

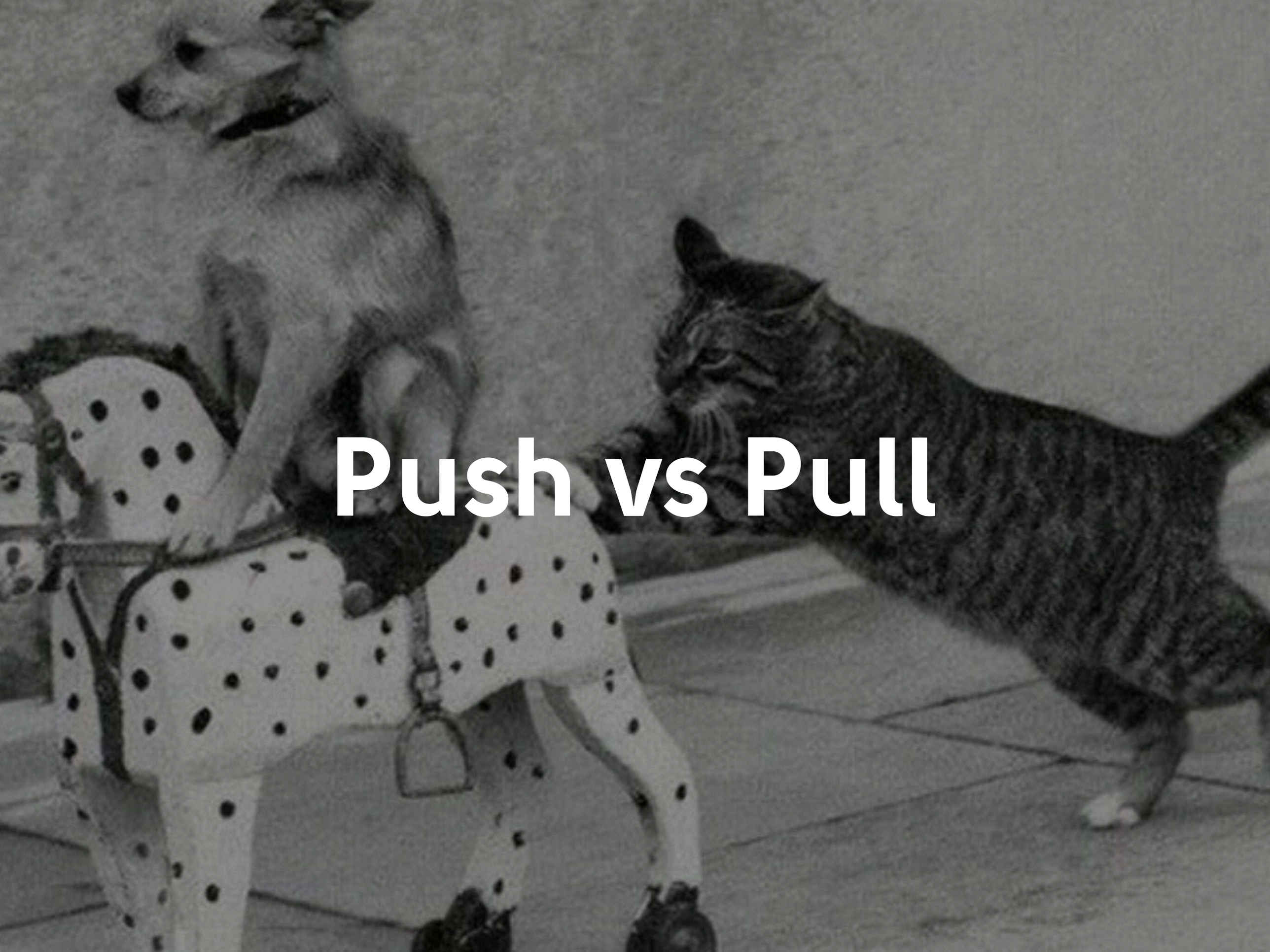
- Используемые абстракции
- **Модель передачи значений (push vs pull)**
- Порядок вычислений (glitch avoidance)
- Существующие операторы (lifting)

# Метрики

- Используемые абстракции
- Модель передачи значений (push vs pull)
- **Порядок вычислений (glitch avoidance)**
- Существующие операторы (lifting)

# Метрики

- Используемые абстракции
- Модель передачи значений (push vs pull)
- Порядок вычислений (glitch avoidance)
- **Существующие операторы (lifting)**



# Push vs Pull

# Push vs Pull

**Push**

- При наличии данных (data driven) передаем их потребителям
- Поставщик данных медленнее потребителя
- Способ передачи - например, callbacks

**Pull**

- При наличии потребности (demand driven) запрашиваем данные у поставщика
- Потребитель данных медленнее поставщика
- Способ пердачи - например, Queue

# Push vs Pull

**Push**

- При наличии данных (data driven) передаем их потребителям
- **Поставщик данных медленнее потребителя**
- Способ передачи - например, callbacks

**Pull**

- При наличии потребности (demand driven) запрашиваем данные у поставщика
- Потребитель данных медленнее поставщика
- Способ пердачи - например, Queue

# Push vs Pull

**Push**

- При наличии данных (data driven) передаем их потребителям
- Поставщик данных медленнее потребителя
- Способ передачи - например, callbacks

**Pull**

- При наличии потребности (demand driven) запрашиваем данные у поставщика
- Потребитель данных медленнее поставщика
- Способ пердачи - например, Queue

# Push vs Pull

**Push**

- При наличии данных (data driven) передаем их потребителям
- Поставщик данных медленнее потребителя
- Способ передачи - например, callbacks

**Pull**

- При наличии потребности (demand driven) запрашиваем данные у поставщика
- Потребитель данных медленнее поставщика
- Способ пердачи - например, Queue

# Push vs Pull

**Push**

- При наличии данных (data driven) передаем их потребителям
- Поставщик данных медленнее потребителя
- Способ передачи - например, callbacks

**Pull**

- При наличии потребности (demand driven) запрашиваем данные у поставщика
- **Потребитель данных медленнее поставщика**
- Способ пердачи - например, Queue

# Push vs Pull

**Push**

- При наличии данных (data driven) передаем их потребителям
- Поставщик данных медленнее потребителя
- Способ передачи - например, callbacks

**Pull**

- При наличии потребности (demand driven) запрашиваем данные у поставщика
- Потребитель данных медленнее поставщика
- Способ пердачи - например, Queue

# Как это применимо к Rx?

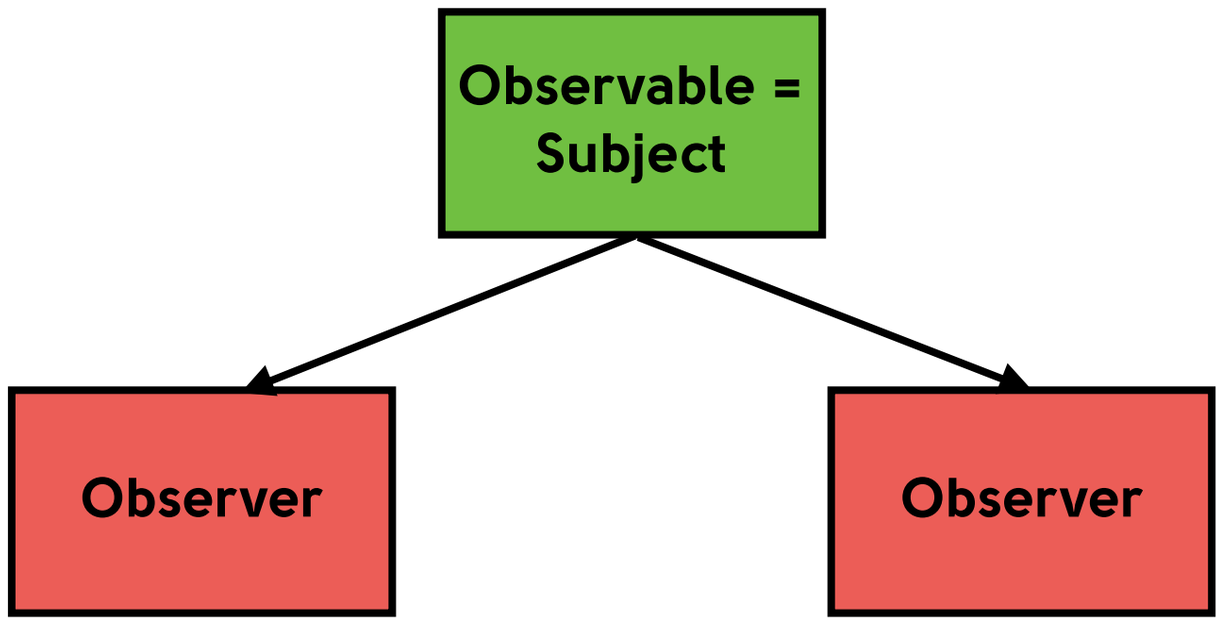
- Observable = push based sequence
- => к Observable применимы функциональные преобразования и операторы

# Как это применимо к Rx?

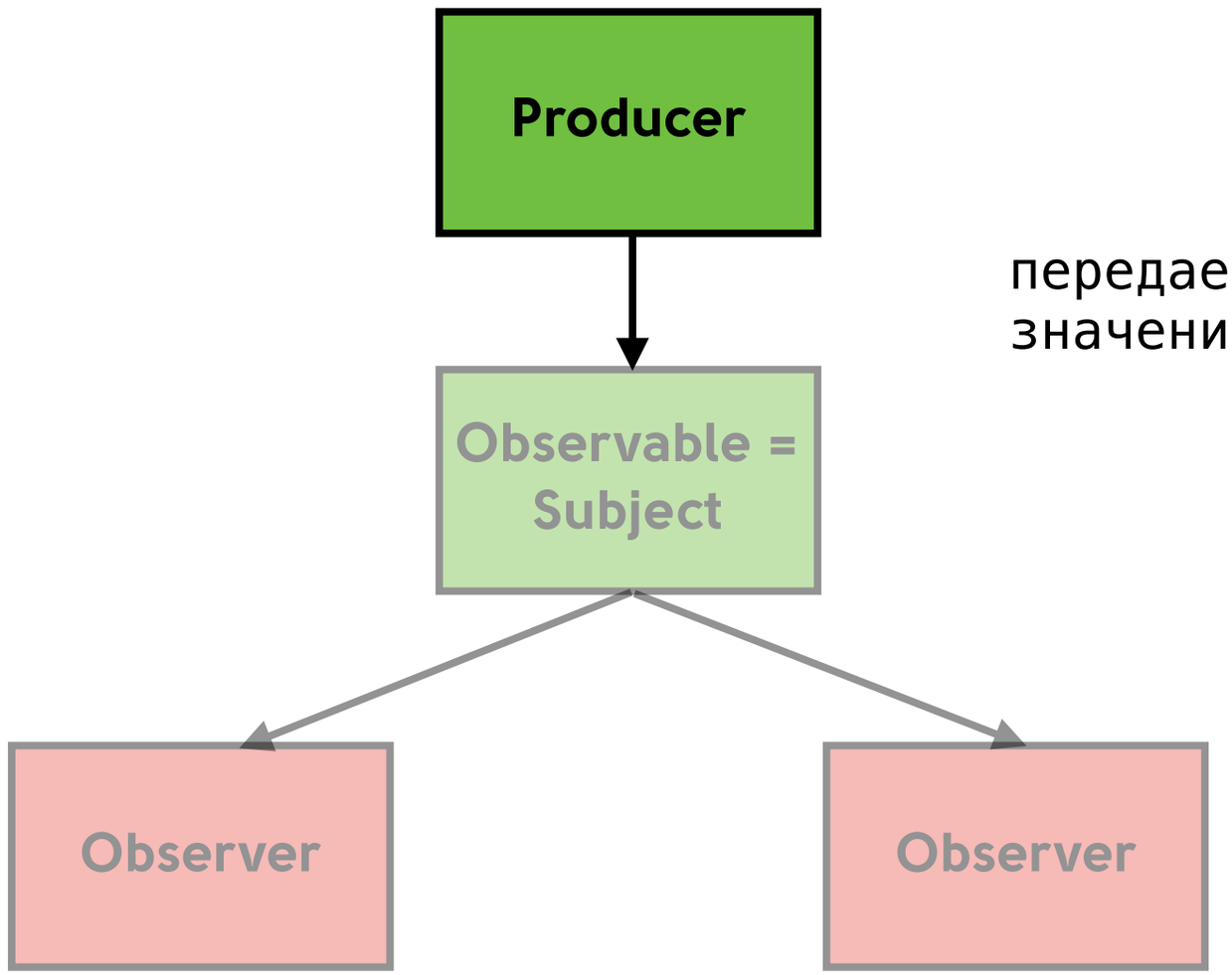
- Observable = push based sequence
- **=> к Observable применимы функциональные преобразования и операторы**

# Observer vs Iterator

# Observer pattern

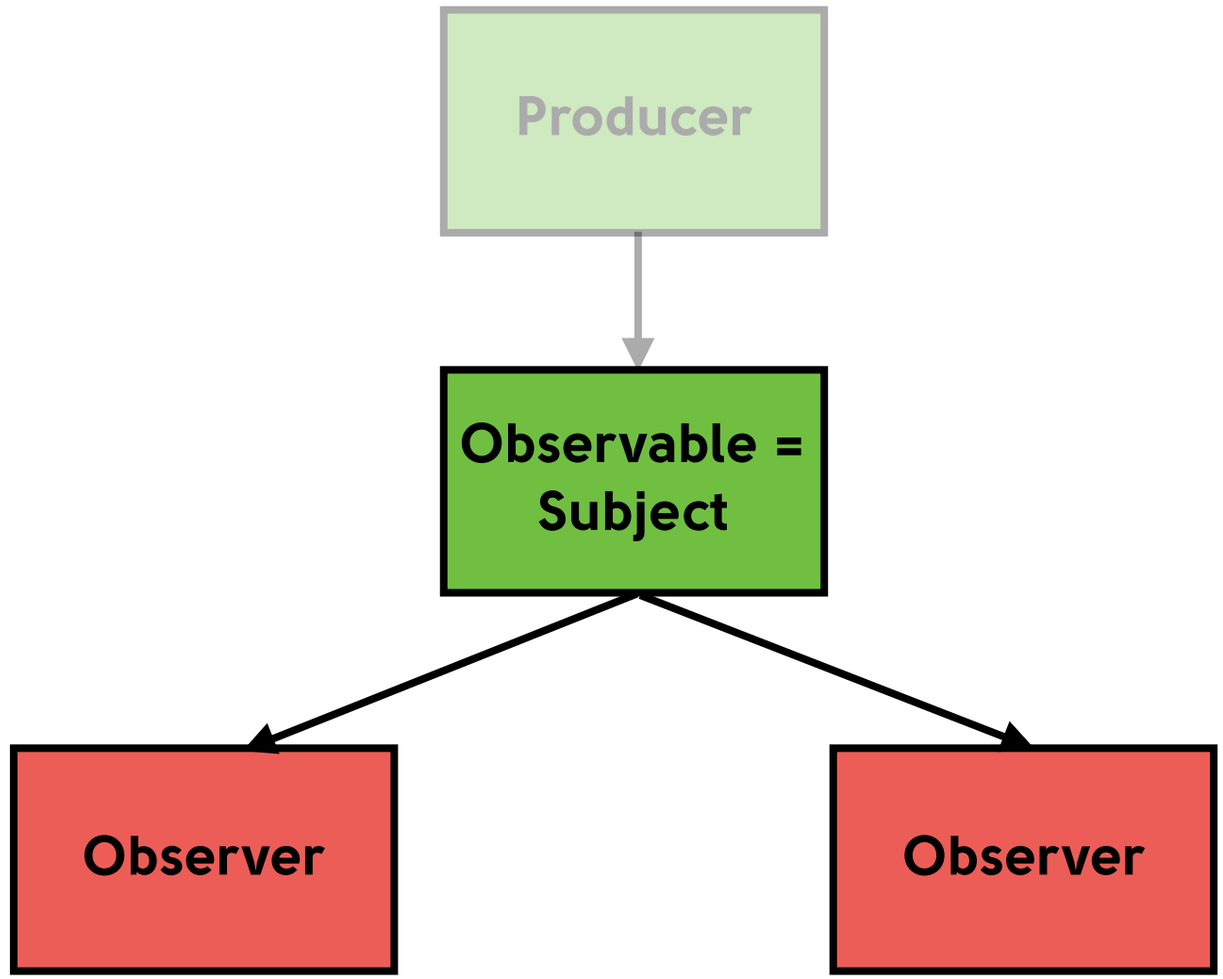


# Observer pattern




передает
значение

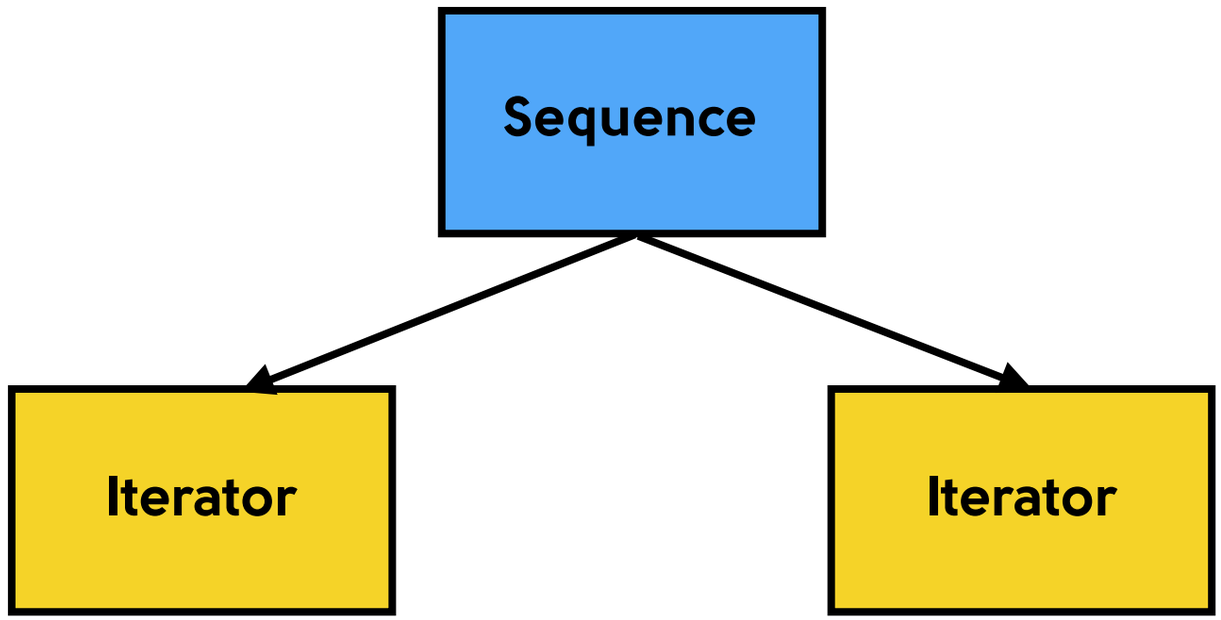
# Observer pattern



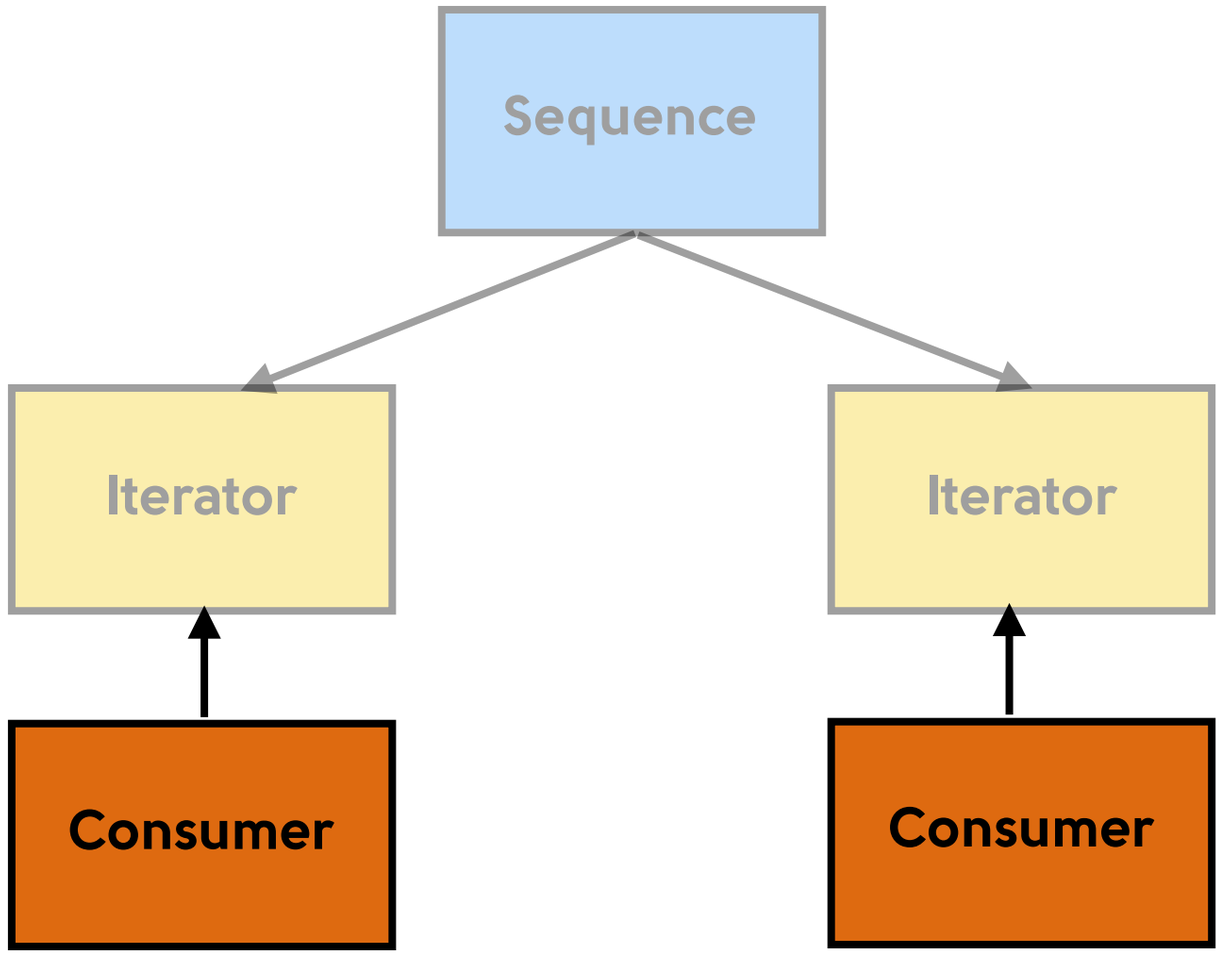

```
class Observable<Element> {
    func subscribe(_ observer: Observer<Element>)
}

class Observer<Element> {
    func onNext(_ element: Element)
}
```

# Iterator pattern

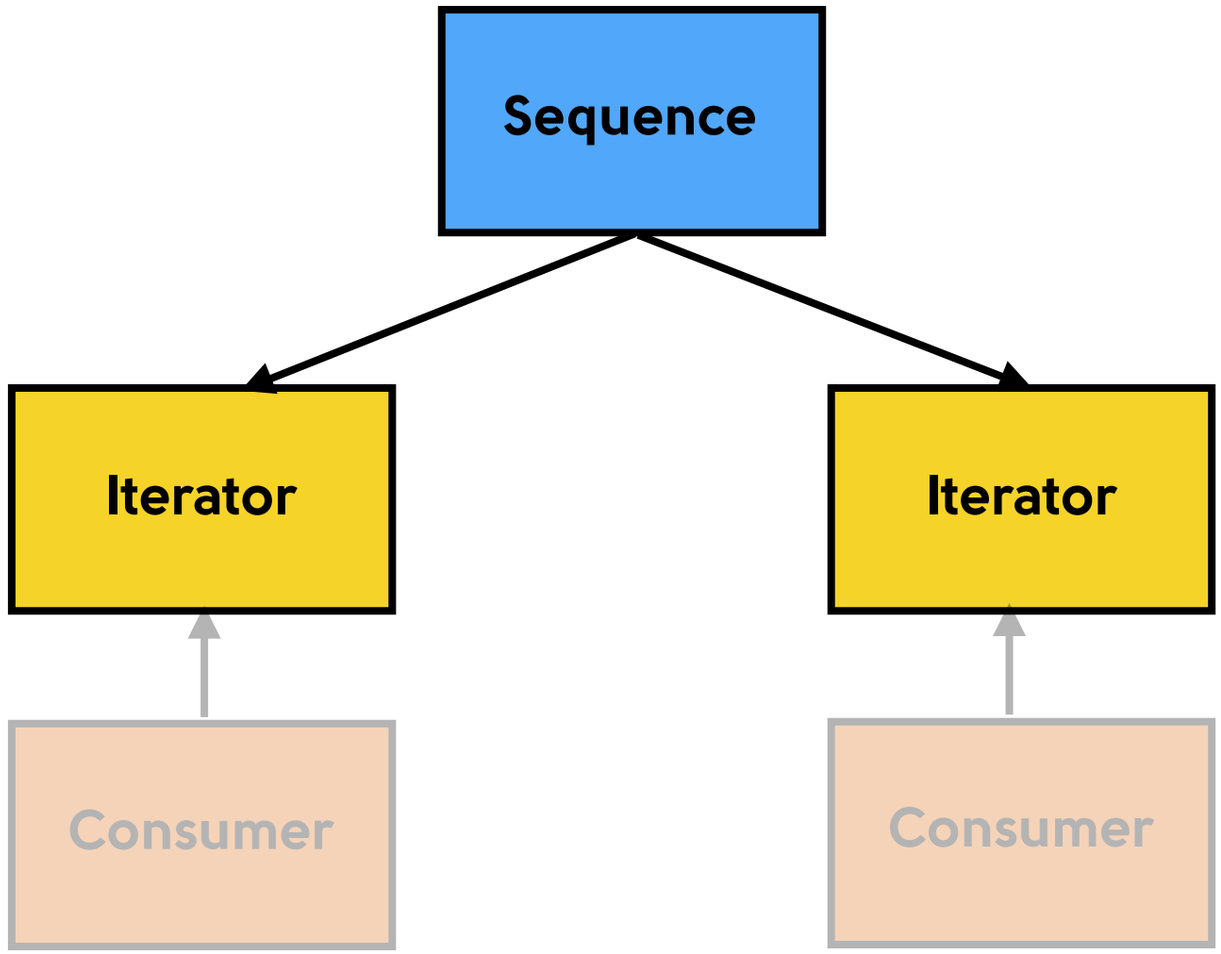


# Iterator pattern




Получает
значение

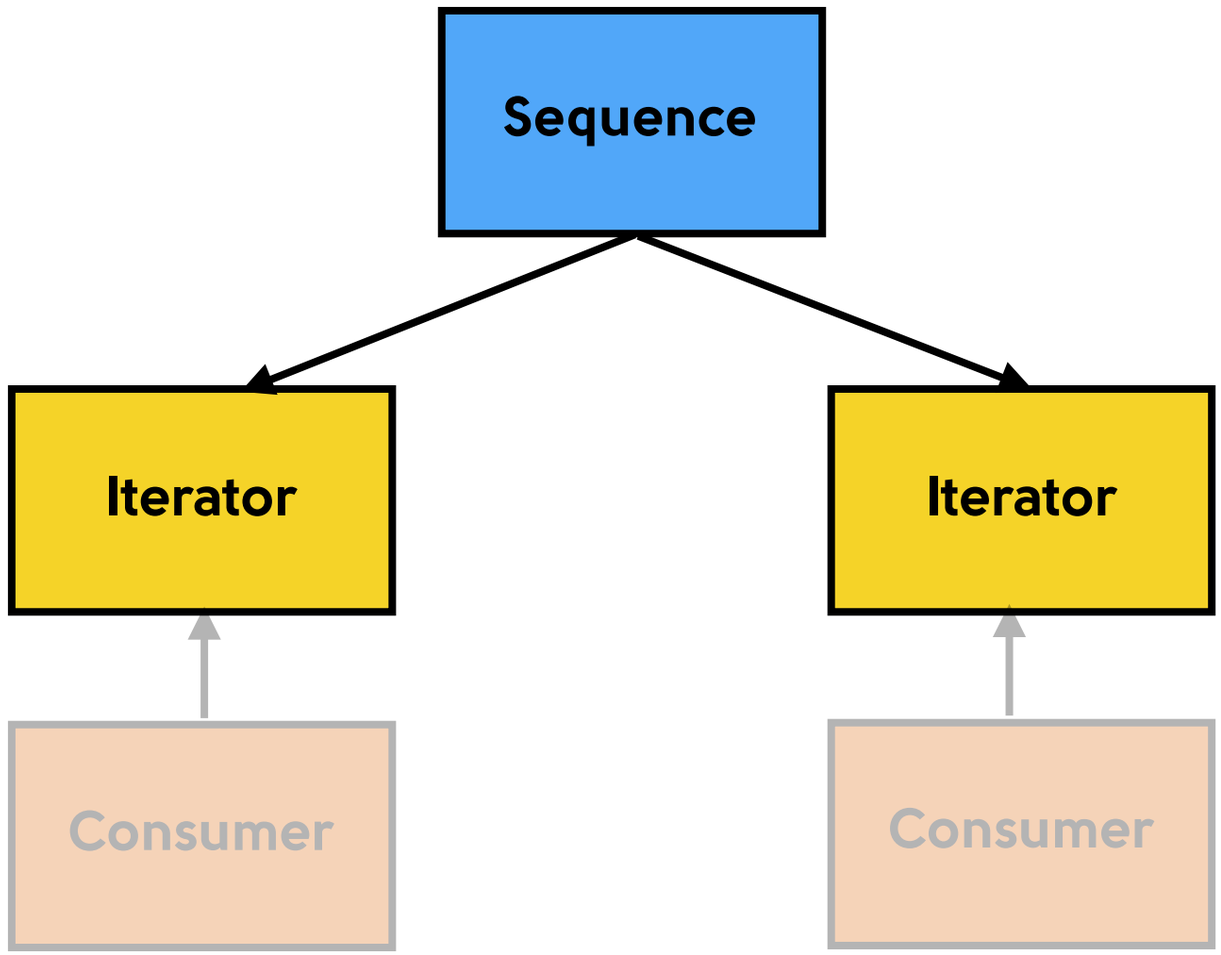
# Iterator pattern




```
class Sequence<Element> {
    func makeIterator() -> Iterator
}

class Iterator<Element> {
    func next() -> Element?
}
```

# Iterator pattern



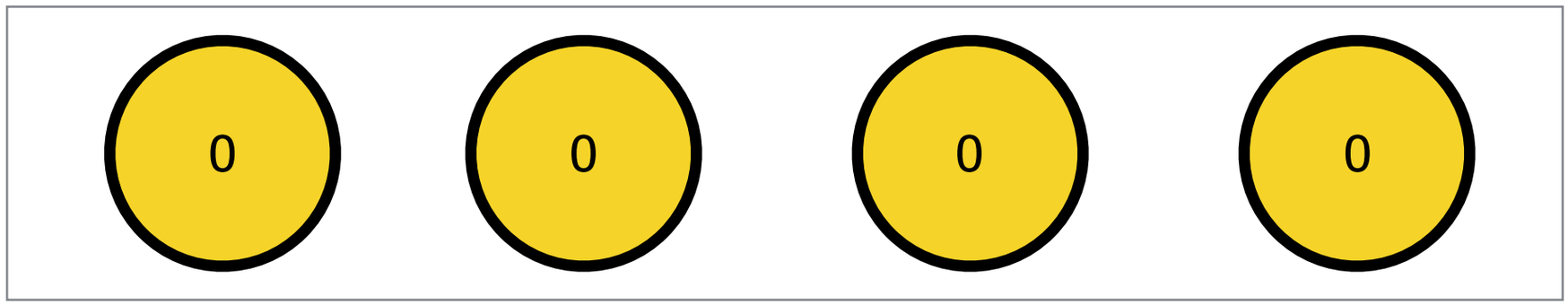

```
map
flatmap
filter
reduce
```

```
class Sequence<Element> {
    func makeIterator() -> Iterator
}

class Iterator<Element> {
    func next() -> Element?
}
```

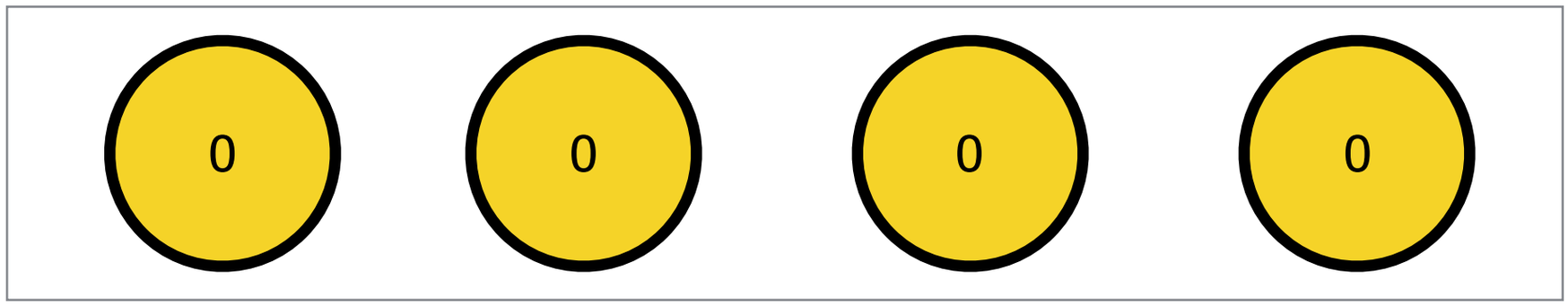
# Преобразование последовательности

**Iterator <0>**

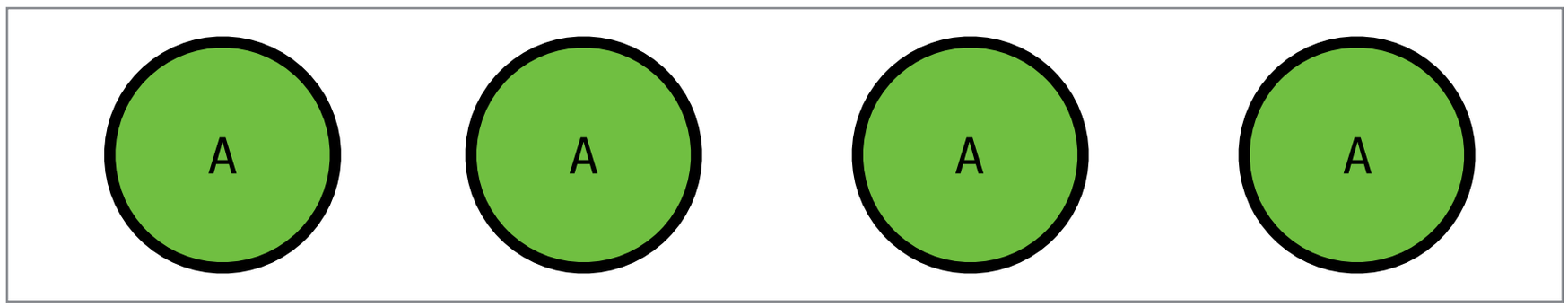


# Преобразование последовательности

**Iterator <0>**

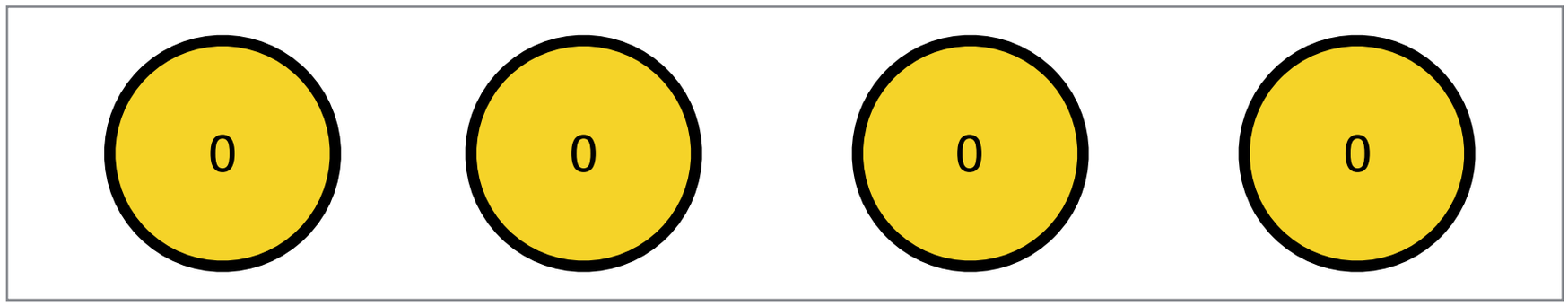



**Iterator <A>**

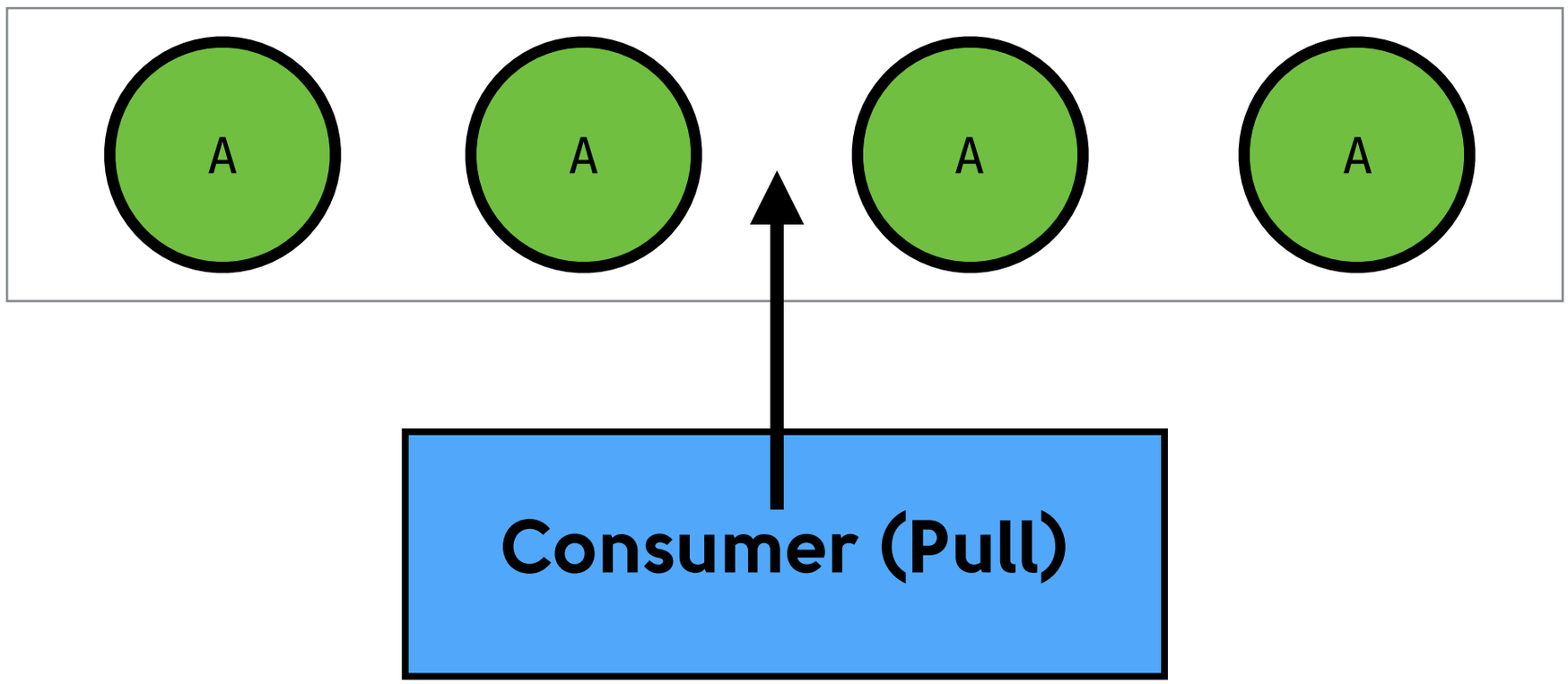


# Преобразование последовательности

**Iterator <0>**

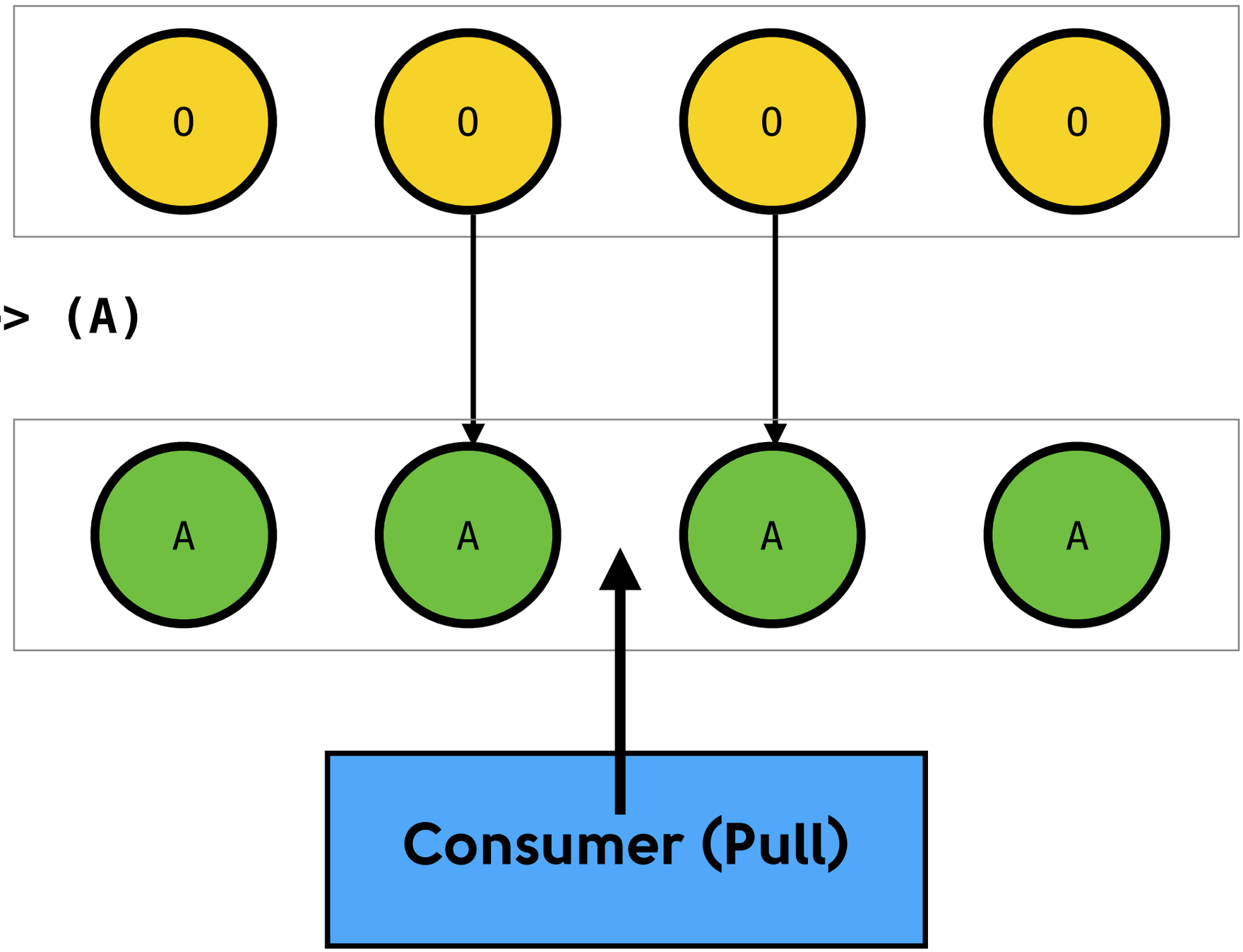



**Iterator <A>**



# Преобразование последовательности

**Iterator <O>**

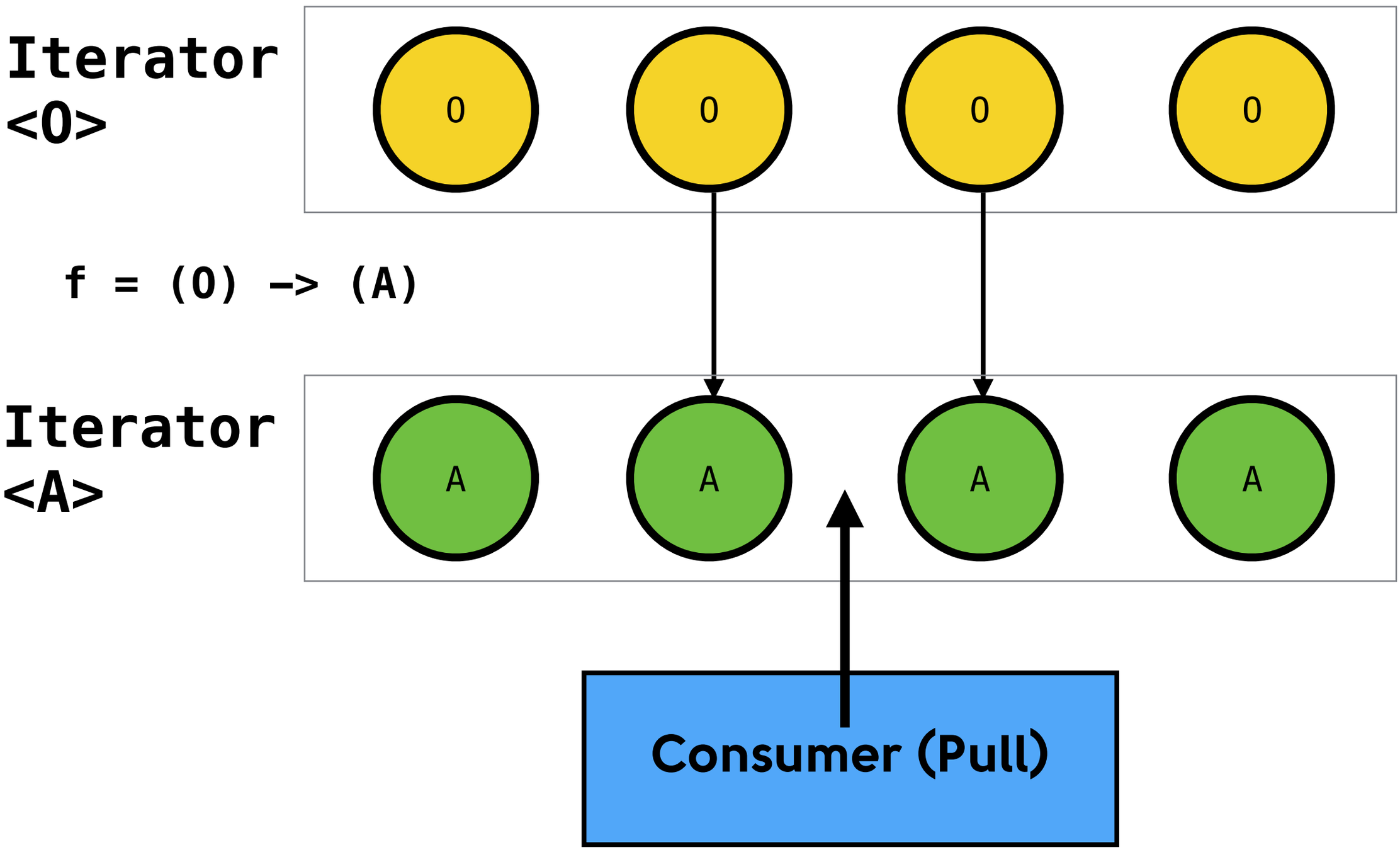



**f = (O) -> (A)**

**Iterator <A>**

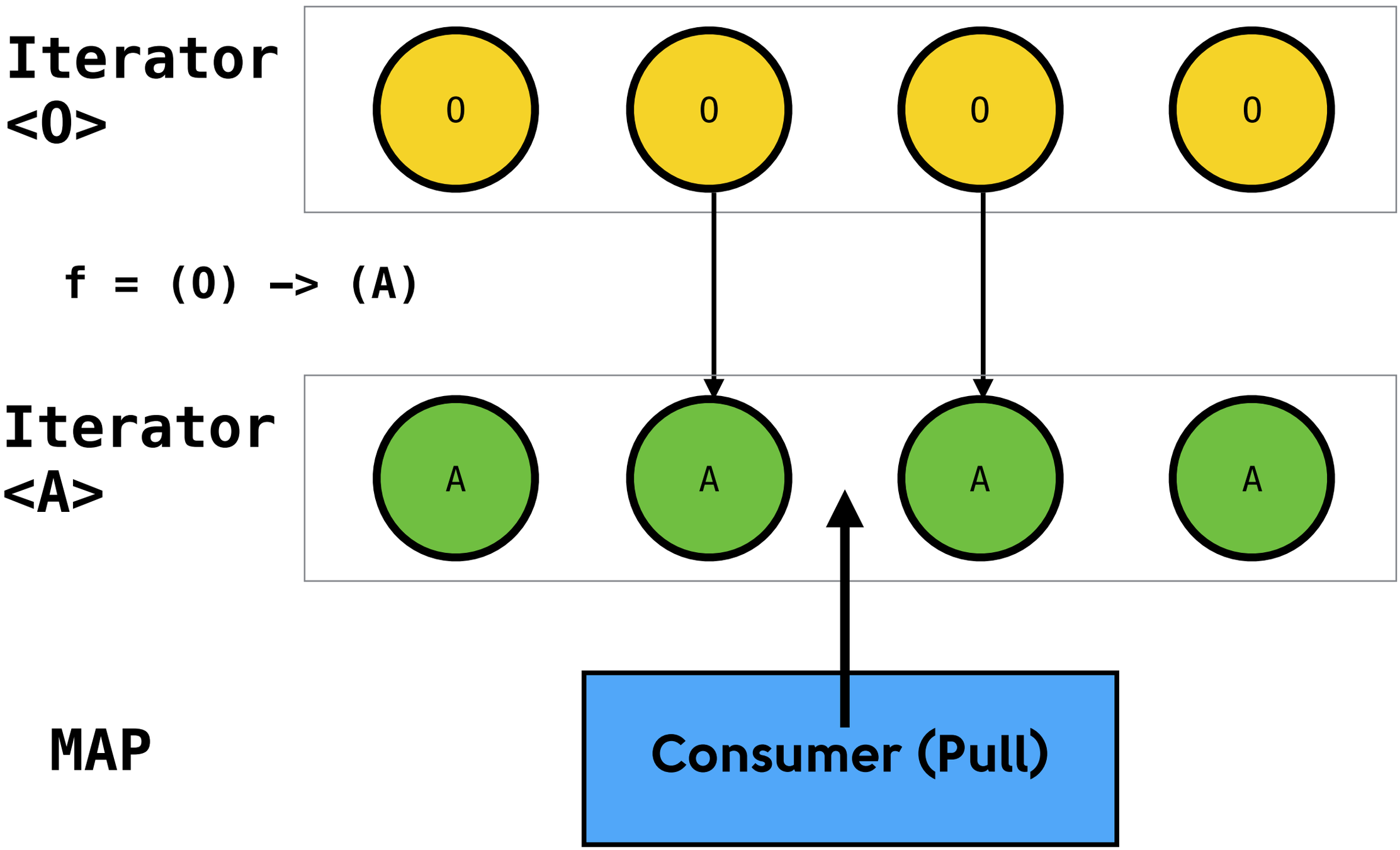
# Преобразование последовательности

`element = f(Iterator<O>.next)`

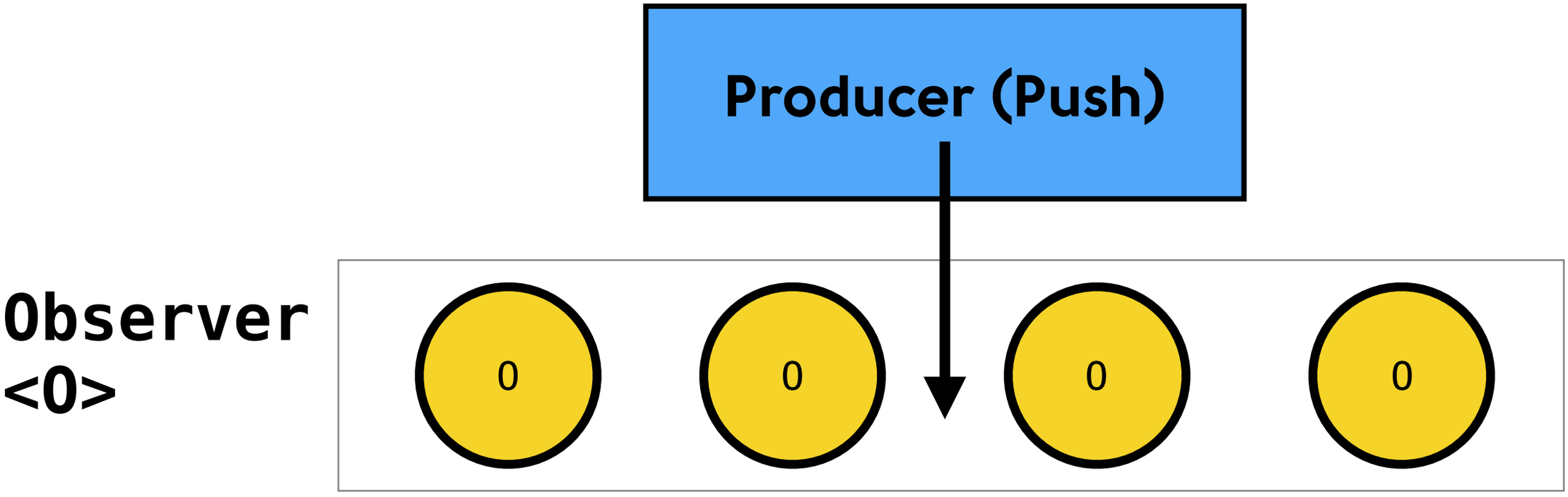


# Преобразование последовательности

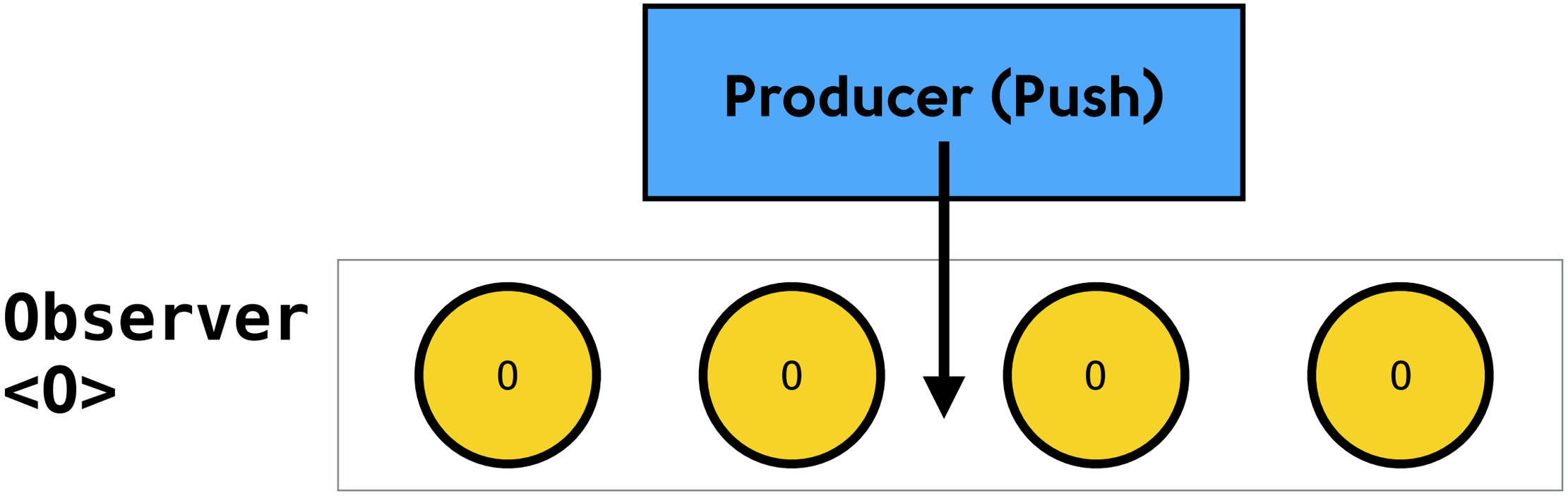
```
element = f(Iterator<O>.next)
```



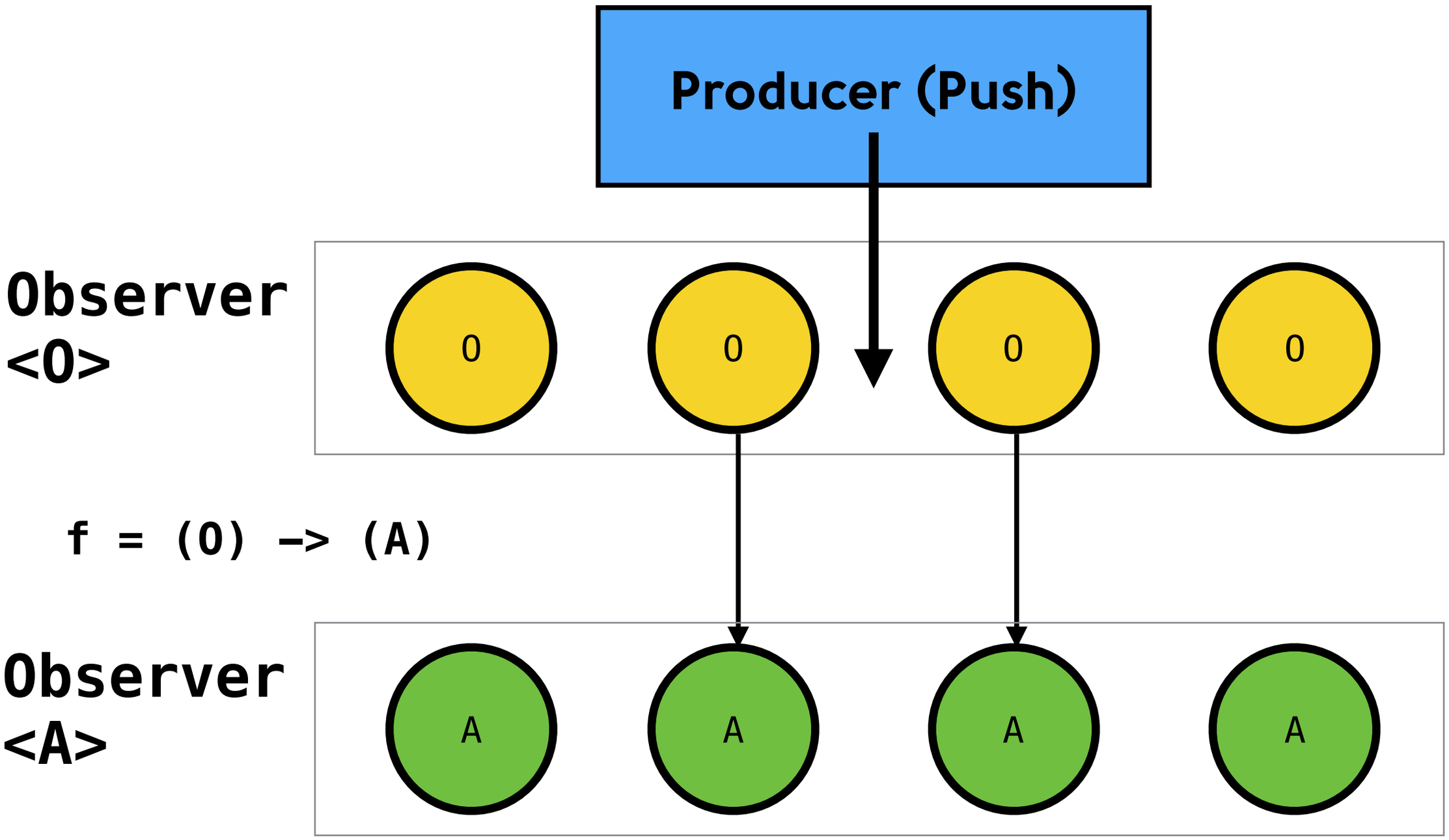
# Преобразование последовательности



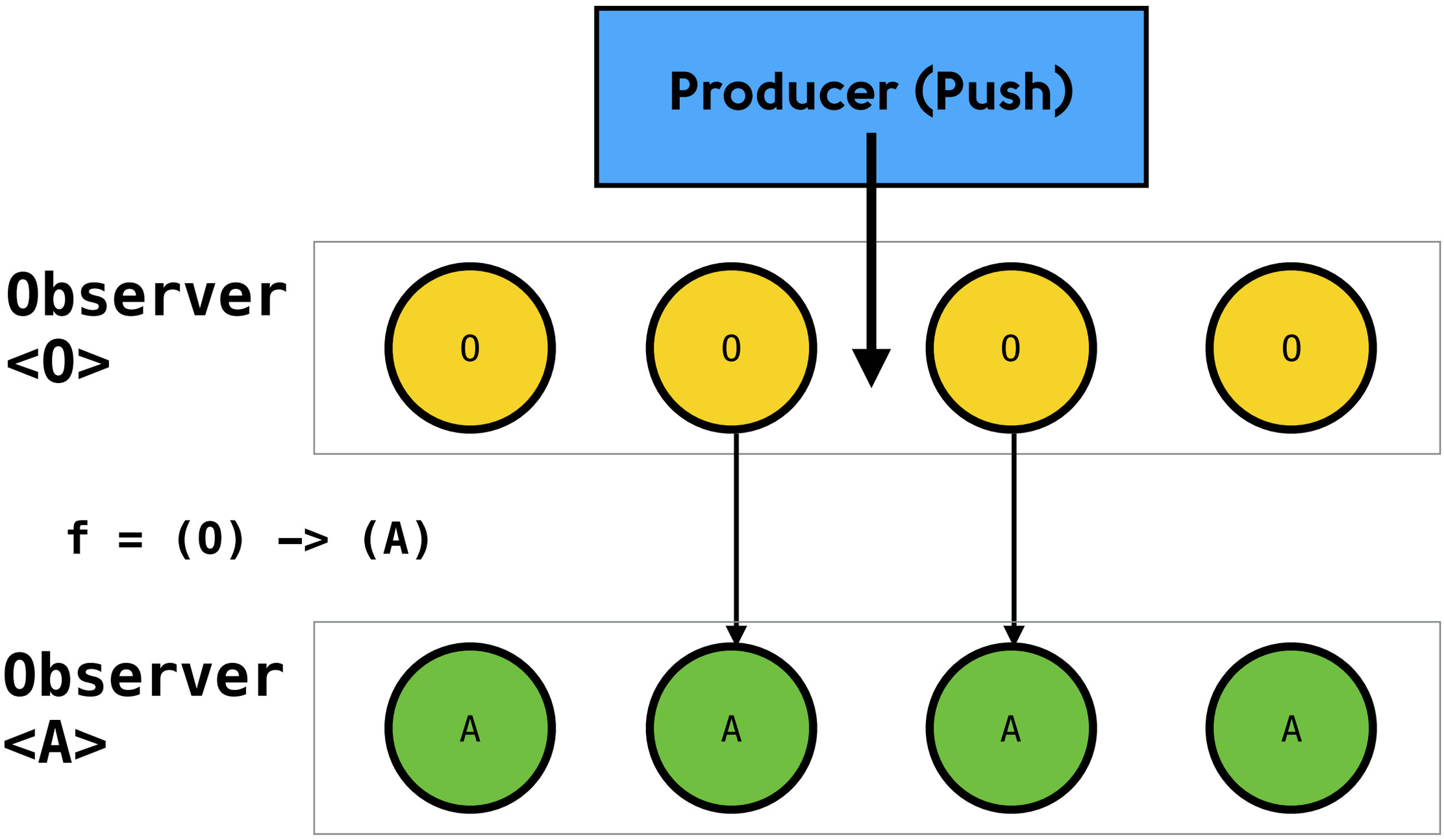
# Преобразование последовательности



# Преобразование последовательности

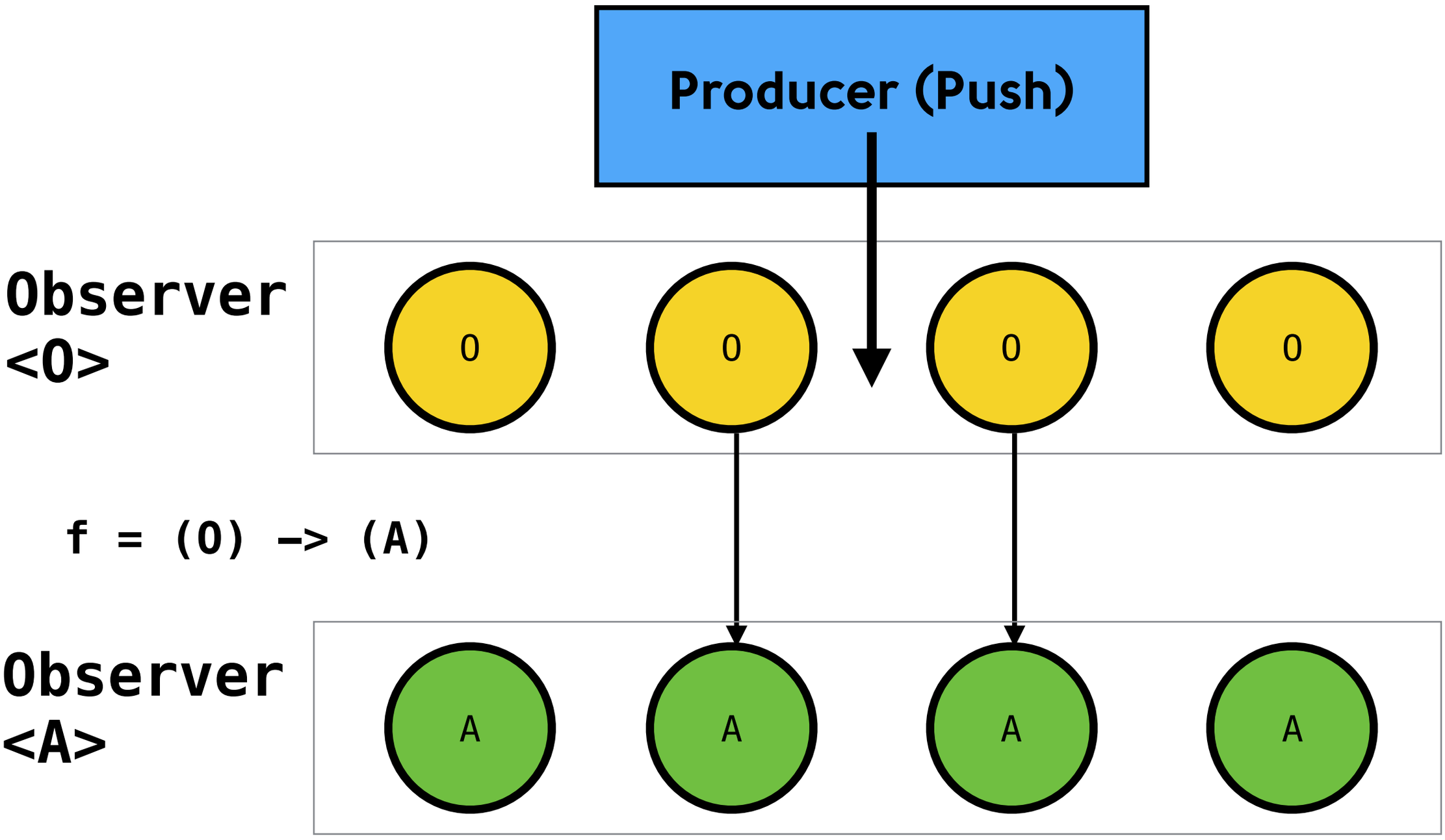


# Преобразование последовательности




```
Observer<A>.onNext(f(element))
```

# Преобразование последовательности

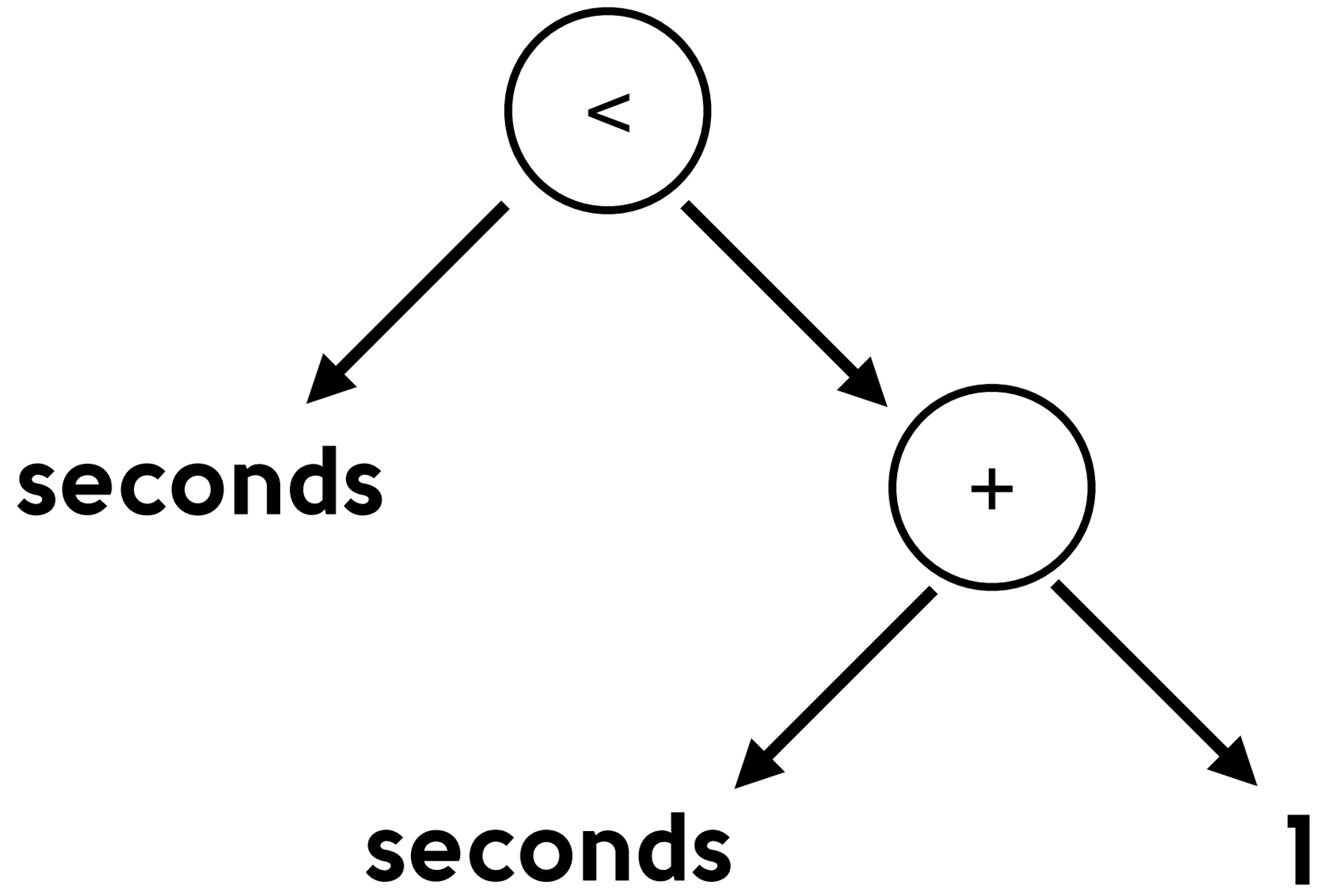



MAP

```
Observer<A>.onNext(f(element))
```

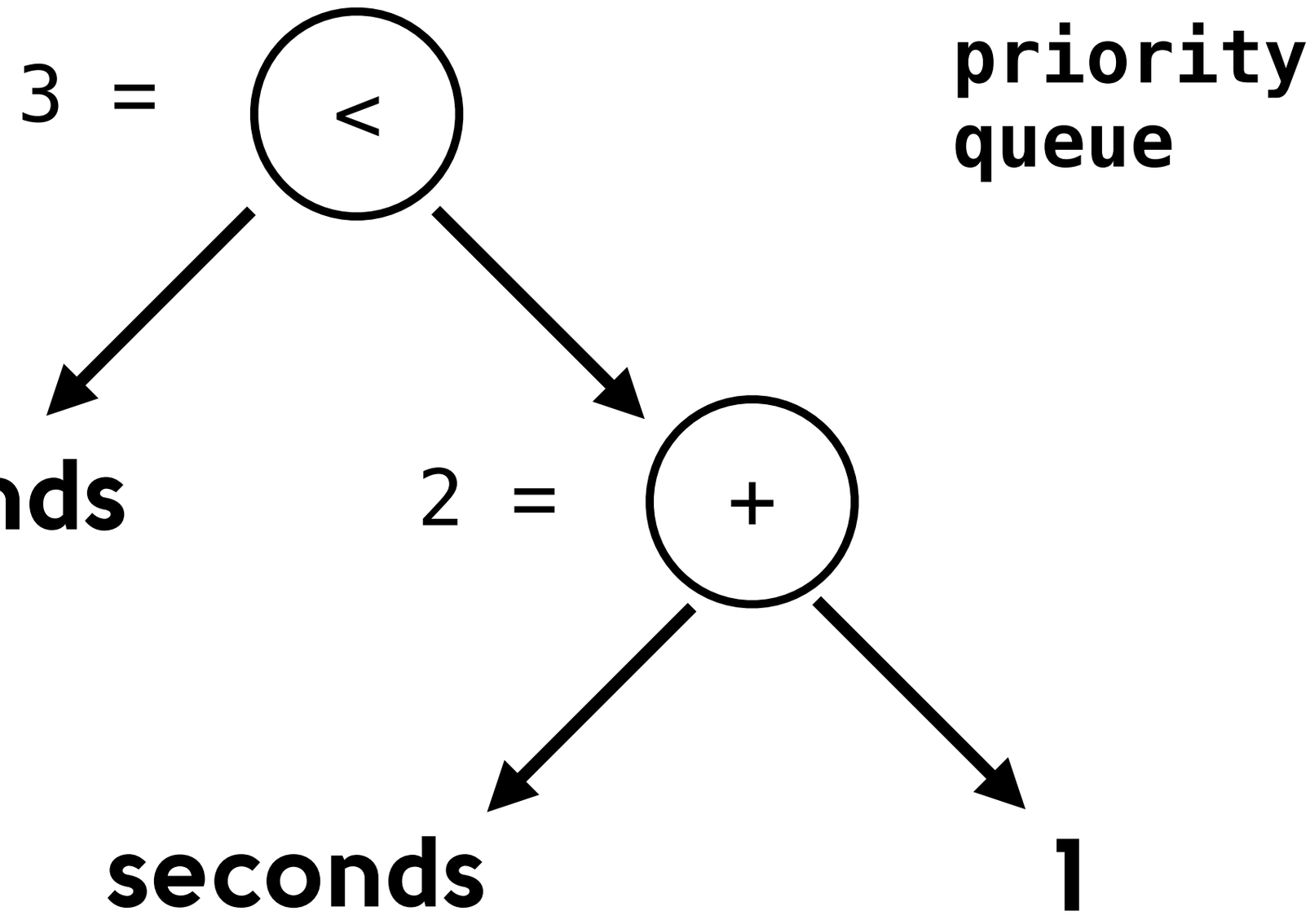
# Glitch avoidance

# seconds < 1 + seconds




FrTime

# seconds < 1 + seconds




FrTime

# Как это применимо к Rx?

- У Rx все зависит от семантики конкретного оператора

# CombineLatest

```
let base = PublishSubject<Int>()
let first = base.map { $0 + 1 }
let second = base.map { $0 + 2 }
let combined = Observable.combineLatest(first,
second) { one, two in
    print(one < two)
}

combined.subscribe()
```

# CombineLatest

```
let base = PublishSubject<Int>()
let first = base.map { $0 + 1 }
let second = base.map { $0 + 2 }
let combined = Observable.combineLatest(first,
second) { one, two in
    print(one < two)
}

combined.subscribe()
```

# CombineLatest

```
let base = PublishSubject<Int>()
let first = base.map { $0 + 1 }
let second = base.map { $0 + 2 }
let combined = Observable.combineLatest(first,
second) { one, two in
    print(one < two)
}

combined.subscribe()
```

# CombineLatest

```
let base = PublishSubject<Int>()
let first = base.map { $0 + 1 }
let second = base.map { $0 + 2 }
let combined = Observable.combineLatest(first,
second) { one, two in
    print(one < two)
}

combined.subscribe()
```

# CombineLatest

```
let base = PublishSubject<Int>()
let first = base.map { $0 + 1 }
let second = base.map { $0 + 2 }
let combined = Observable.combineLatest(first,
second) { one, two in
    print(one < two)
}

combined.subscribe()
```

```
base.onNext(1)        true
base.onNext(2)       false
                      true
```

# CombineLatest

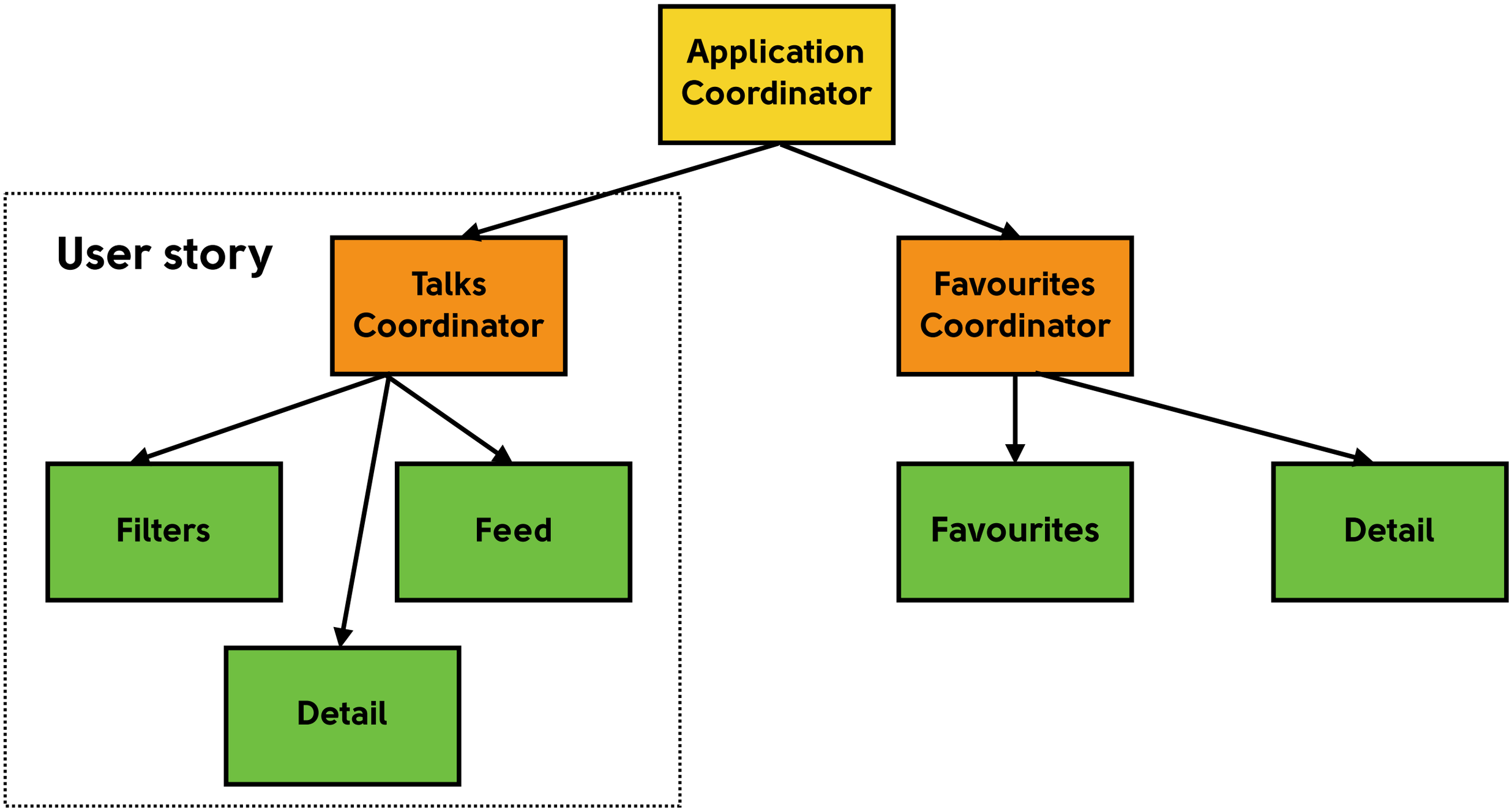
```
let base = PublishSubject<Int>()
let first = base.map { $0 + 1 }
let second = base.map { $0 + 2 }
let combined = Observable.combineLatest(first,
second) { one, two in
    print(one < two)
}

combined.subscribe()
```

```
base.onNext(1)          true
base.onNext(2)         false
                        true
```

# Архитектура

# Архитектура



# Роль координатора

- Создание дочерних координаторов
- Создание модулей (с помощью фабрик)
- Переход между модулями (через роутер/без него)
- Общение между модулями

# Роль координатора

- Создание дочерних координаторов
- Создание модулей (с помощью фабрик)
- Переход между модулями (через роутер/без него)
- Общение между модулями

# Роль координатора

- Создание дочерних координаторов
- Создание модулей (с помощью фабрик)
- Переход между модулями (через роутер/без него)
- Общение между модулями

# Роль координатора

- Создание дочерних координаторов
- Создание модулей (с помощью фабрик)
- Переход между модулями (через роутер/без него)
- Общение между модулями

# Пример (Application Coordinator)

```
let favouritesNC = UINavigationController()
tabBarController.embed(viewController:
favouritesNavigationController)

let talksCoordinator =
factory.talksCoordinator
(rootNavigationController: favouritesNC)

add(coordinator: talksCoordinator)
talksCoordinator().start()
```

# Пример (Application Coordinator)

```
let favouritesNC = UINavigationController()
tabBarController.embed(viewController:
favouritesNavigationController)

let talksCoordinator =
factory.talksCoordinator
(rootNavigationController: favouritesNC)

add(coordinator: talksCoordinator)
talksCoordinator().start()
```

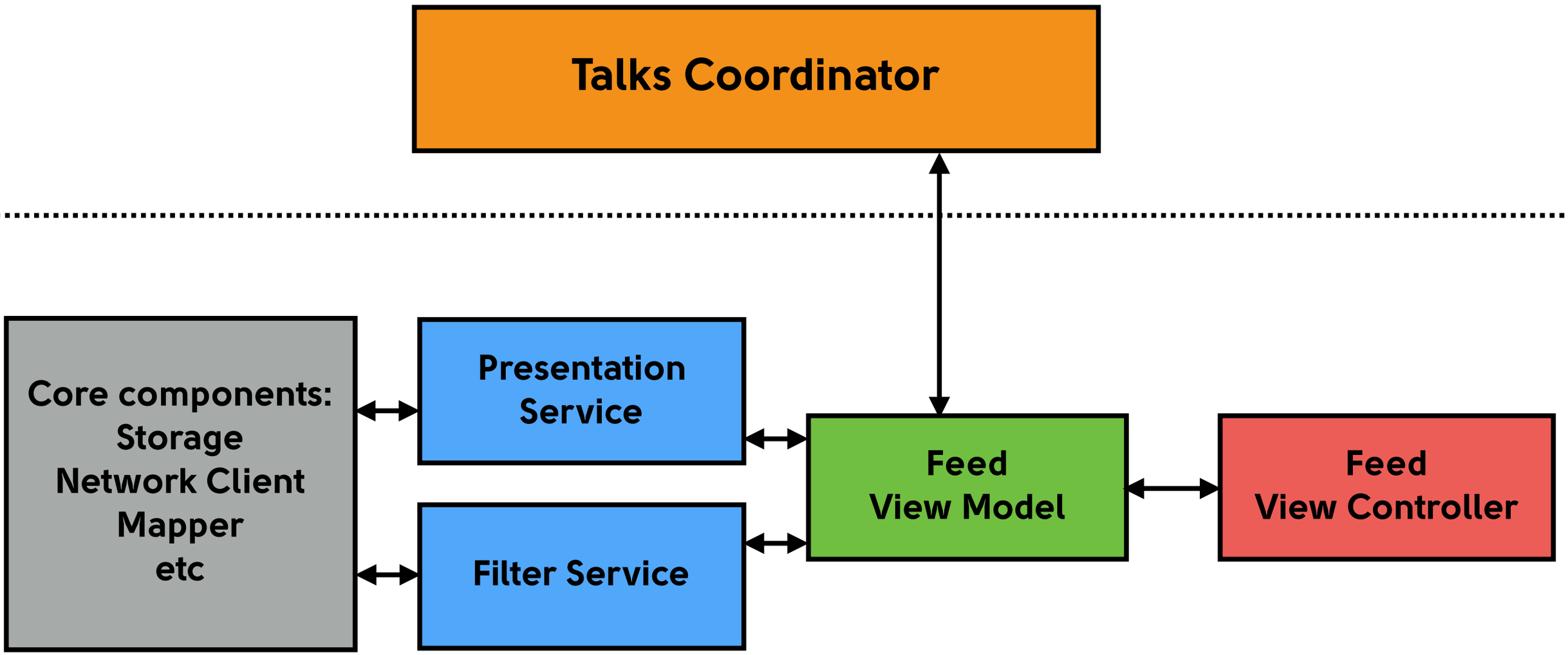
# Пример (Application Coordinator)

```
let favouritesNC = UINavigationController()
tabBarController.embed(viewController:
favouritesNavigationController)

let talksCoordinator =
factory.talksCoordinator
(rootNavigationController: favouritesNC)

add(coordinator: talksCoordinator)
talksCoordinator().start()
```

# Архитектура отдельного модуля




SOA

Presentation

# Требование к архитектуре

- Абстрактные интерфейсы (все компоненты закрыты протоколами)
- На сигнал, идущий из компонента (почти) нельзя повлиять

# Требование к архитектуре

- Абстрактные интерфейсы (все компоненты закрыты протоколами)
- На сигнал, идущий из компонента (почти) нельзя повлиять

# Требование к архитектуре

- Абстрактные интерфейсы (все компоненты закрыты протоколами)
- На сигнал, идущий из компонента (почти) нельзя повлиять

Observable VS PublishSubject

# Требование к архитектуре

- Абстрактные интерфейсы (все компоненты закрыты протоколами)
- На сигнал, идущий из компонента (почти) нельзя повлиять

Observable

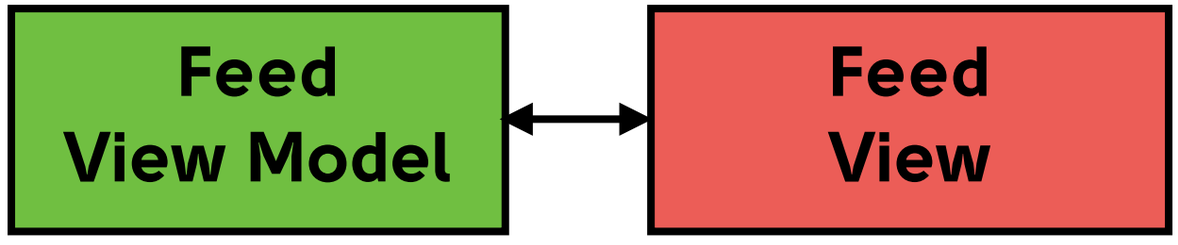
VS

PublishSubject

||

Observer + Observable

# Интерфейс через Observable




```
protocol FeedViewModel {
    var presentations: Observable<[…]>
    { get }
}
```

```
protocol FeedView {
    var indexSelected:
        Observable<IndexPath> { get }

    var filtersButtonTapped:
        Observable<Void> { get }
}
```

# View Controller

```
var filtersButtonTapped: Observable<Void> {
    return navigationItem.rightBarButtonItem!
                         .rx.tap.asObservable()
}
```

```
var indexSelected: Observable<IndexPath> {
    return tableView.rx.itemSelected.asObservable()
}
```

```
viewModel?.presentations
          .bindTo(tableView.rx.items(dataSource: source))
          .disposed(by: disposeBag)
```

# View Controller

```
var filtersButtonTapped: Observable<Void> {
    return navigationItem.rightBarButtonItem!
                         .rx.tap.asObservable()
}
```

```
var indexSelected: Observable<IndexPath> {
    return tableView.rx.itemSelected.asObservable()
}
```

```
viewModel?.presentations
          .bindTo(tableView.rx.items(dataSource: source))
          .disposed(by: disposeBag)
```

# View Controller

```
var filtersButtonTapped: Observable<Void> {
    return navigationItem.rightBarButtonItem!
                         .rx.tap.asObservable()
}
```

```
var indexSelected: Observable<IndexPath> {
    return tableView.rx.itemSelected.asObservable()
}
```

```
viewModel?.presentations
          .bindTo(tableView.rx.items(dataSource: source))
          .disposed(by: disposeBag)
```

# View Model

```
fileprivate var _presentations:
Variable<[PresentationSectionModel]> = Variable([])


var presentations: Observable<[PresentationSectionModel]> {
    return _presentations.asObservable()
}


view.indexSelected
    .flatMap { indexPath in
        let presentationKey = …
        return presentationService.presentation(withKey:
                                                presentationKey)
    }
    .bindTo(presentationPublisher)
    .disposed(by: disposeBag)
```

# View Model

```
fileprivate var _presentations:
Variable<[PresentationSectionModel]> = Variable([])


var presentations: Observable<[PresentationSectionModel]> {
    return _presentations.asObservable()
}


view.indexSelected
    .flatMap { indexPath in
        let presentationKey = …
        return presentationService.presentation(withKey:
                                    presentationKey)
    }
    .bindTo(presentationPublisher)
    .disposed(by: disposeBag)
```

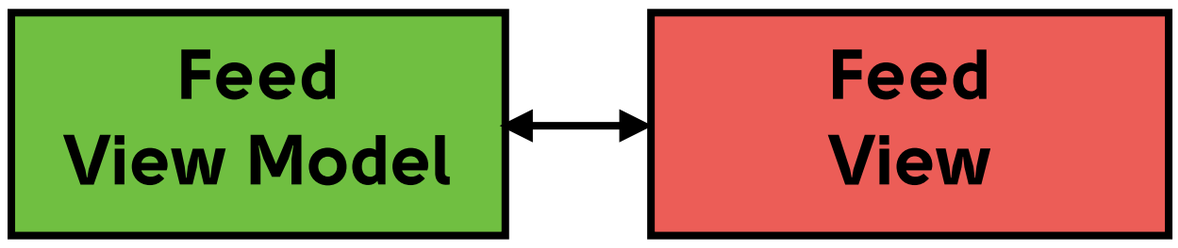
# View Model

```
fileprivate var _presentations:
Variable<[PresentationSectionModel]> = Variable([])


var presentations: Observable<[PresentationSectionModel]> {
    return _presentations.asObservable()
}


view.indexSelected
    .flatMap { indexPath in
        let presentationKey = …
        return presentationService.presentation(withKey:
                                                presentationKey)
    }
    .bindTo(presentationPublisher)
    .disposed(by: disposeBag)
```

# Интерфейс через PublishSubject

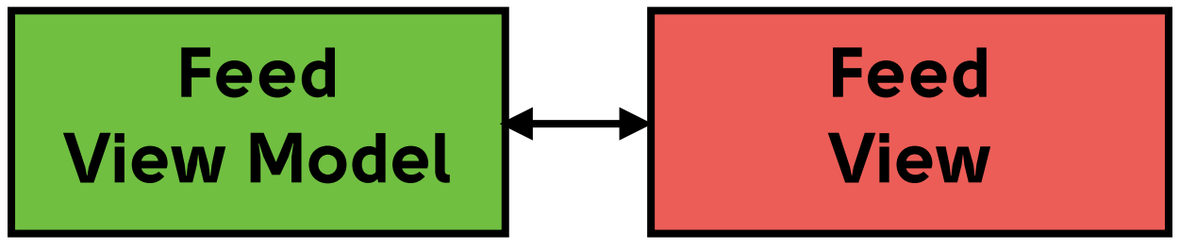



```
protocol FeedViewModel {
    var presentations: Observable<[…]> { get }

    var didSelectIndex:
        PublishSubject<IndexPath> { get }

    var didTapFiltersButton:
        PublishSubject<Void> { get }
}
```

# Интерфейс через PublishSubject




```
protocol FeedViewModel {
    var presentations: Observable<[…]> { get }

    var didSelectIndex:
        PublishSubject<IndexPath> { get }

    var didTapFiltersButton:
        PublishSubject<Void> { get }
}
```

позволяет и передать значение, и подписаться

# ViewController

```
tableView.rx
         .itemSelected
         .bindTo(viewModel.didSelectIndex)
         .disposed(by: disposeBag)

navigationItem.rightBarButtonItem!
              .rx.tap
              .bindTo(viewModel.didTapFiltersButton)
              .disposed(by: disposeBag)
```

# ViewController

```
tableView.rx
         .itemSelected
         .bindTo(viewModel.didSelectIndex)
         .disposed(by: disposeBag)
```

```
navigationItem.rightBarButtonItem!
              .rx.tap
              .bindTo(viewModel.didTapFiltersButton)
              .disposed(by: disposeBag)
```

# View Model

```
self.didSelectIndex
    .flatMap { indexPath in
        let presentationKey = …
        return presentationService.presentation(withKey:
                                                presentationKey)
}
.bindTo(presentationPublisher)
.disposed(by: disposeBag)
```

# Общение между модулями

# Общение через координатор

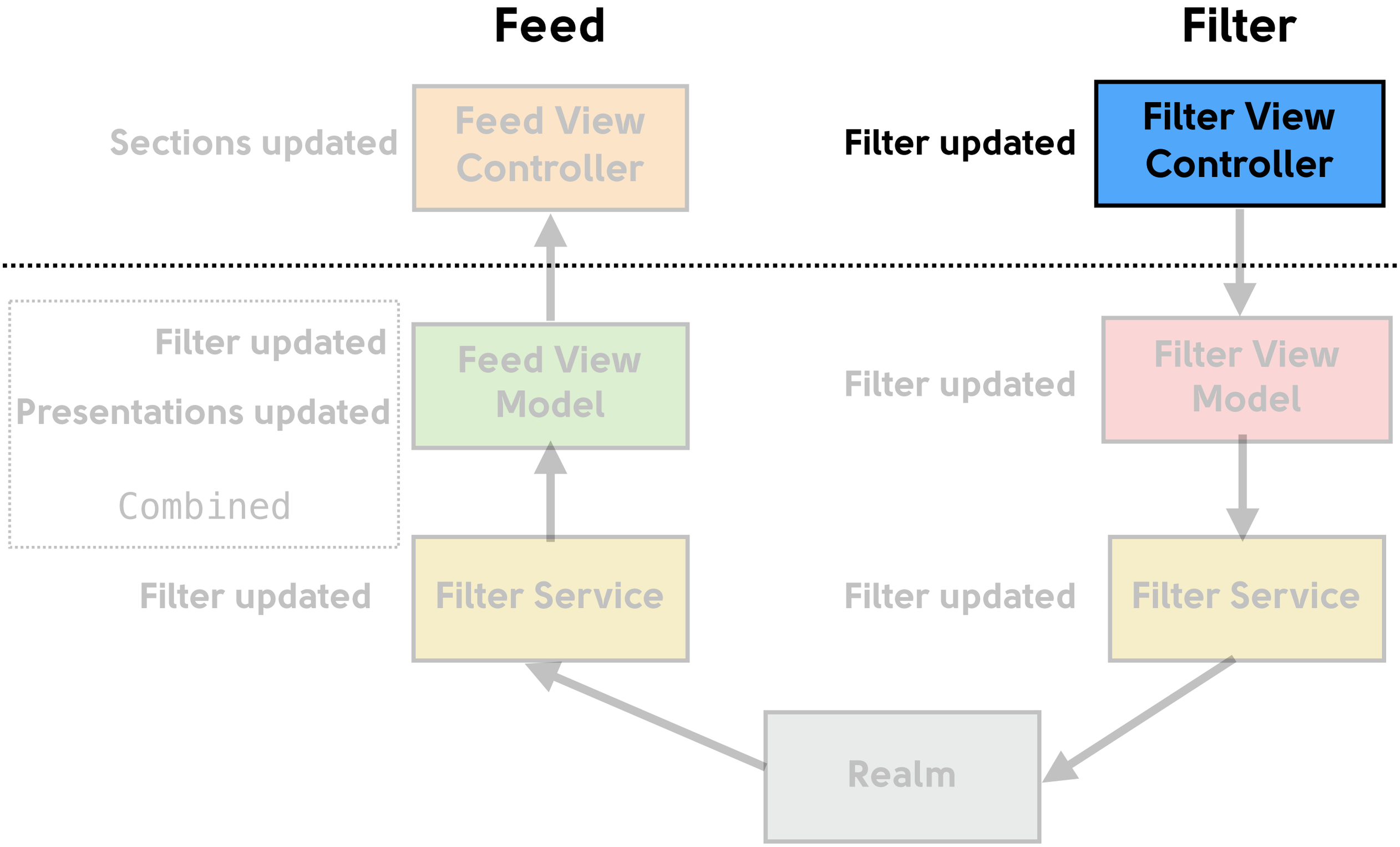

```
protocol FeedModuleInput {
    var didChangeFilters:
        PublishSubject<[Filter]>
        { get }
}
```

```
protocol FiltersModuleOutput {
    var filtersChanged:
        Observable<[Filter]> { get }
}
```

# На уровне координатора

```
filtersModuleOutput
            .filtersChanged
            .bindTo(feedViewModel.didChangeFilters)
            .disposed(by: disposeBag)
```

# Подписка на storage



# Подписка на storage

## Feed

Sections updated

Feed View Controller

Filter updated

Presentations updated

Combined

Feed View Model

Filter updated

Filter Service

## Filter

Filter updated

Filter View Controller

Filter updated

**Filter View Model**

Filter updated

Filter Service

Realm

# Подписка на storage

## Feed

Sections updated

Feed View Controller

Feed View Model

Filter updated

Presentations updated

Combined

Filter updated

Filter Service

## Filter

Filter updated

Filter View Controller

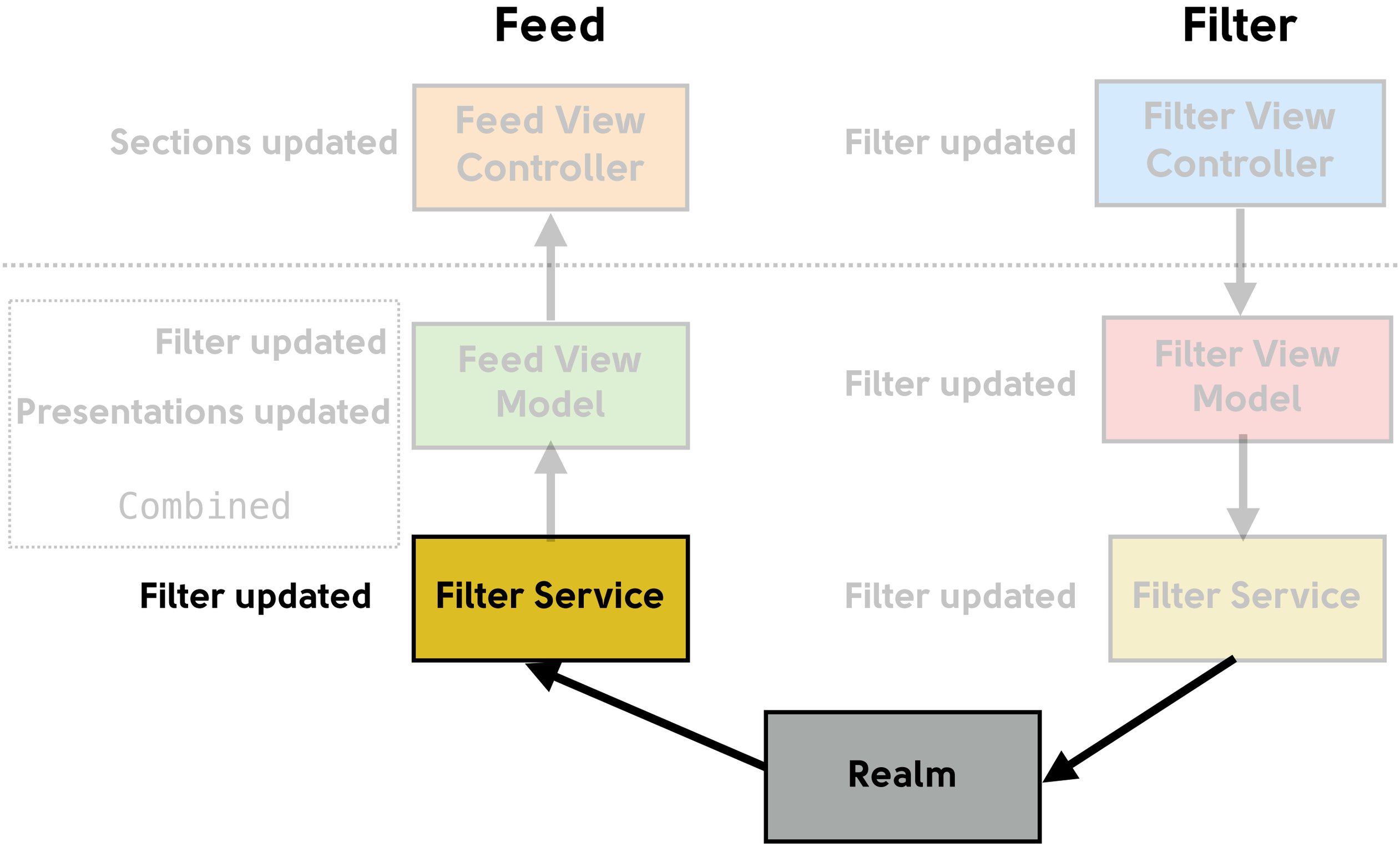
Filter View Model

Filter updated

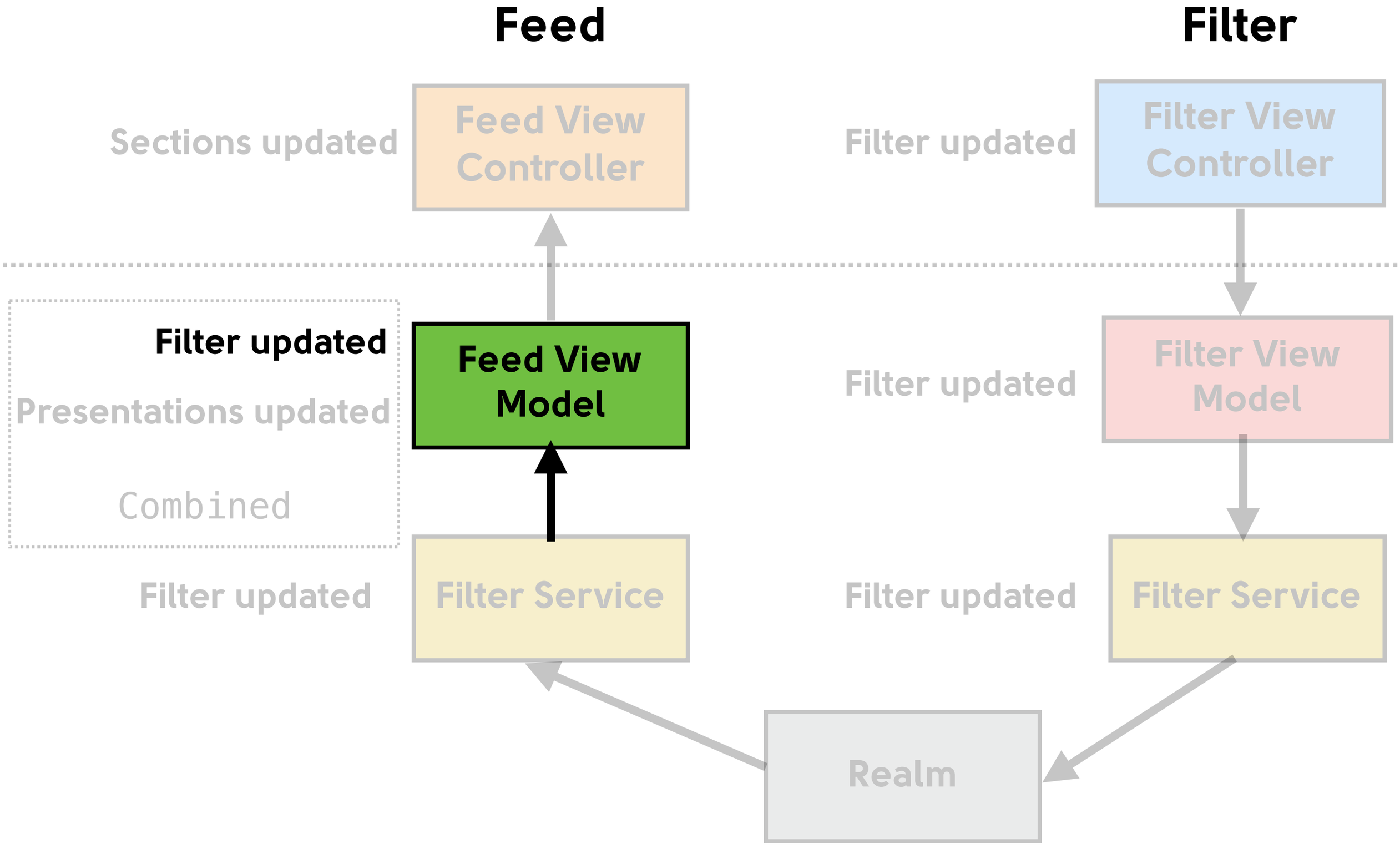
**Filter Service**

Realm

# Подписка на storage



# Подписка на storage



# Подписка на storage

## Feed

Sections updated

Feed View Controller

Filter updated

**Feed View Model**

Filter updated

Filter Service

**Filter updated**

**Presentations updated**

Combined

## Filter

Filter updated

Filter View Controller

Filter updated

Filter View Model

Filter updated

Filter Service

Realm

# View Model

```
Observable.combineLatest(presentationService
                         .presentationsUpdated(),
                          currentFilters
                         .asObservable())
              { _, filters in
                  return filters
              }
              .flatMap { [weak self] filters in
                  return self.presentationService
                             .filteredPresentations(with: filters)
              }
              .map { [weak self] presentations in
                  return self.cellViewModelFactory
                             .sections(from: presentations)
              }
              .bindTo(_presentations)
              .disposed(by: disposeBag)
```

# View Model

```
Observable.combineLatest(presentationService
                         .presentationsUpdated(),
                          currentFilters
                         .asObservable())
            { _, filters in
                return filters
            }
            .flatMap { [weak self] filters in
                return self.presentationService
                           .filteredPresentations(with: filters)
            }
            .map { [weak self] presentations in
                return self.cellViewModelFactory
                           .sections(from: presentations)
            }
            .bindTo(_presentations)
            .disposed(by: disposeBag)
```

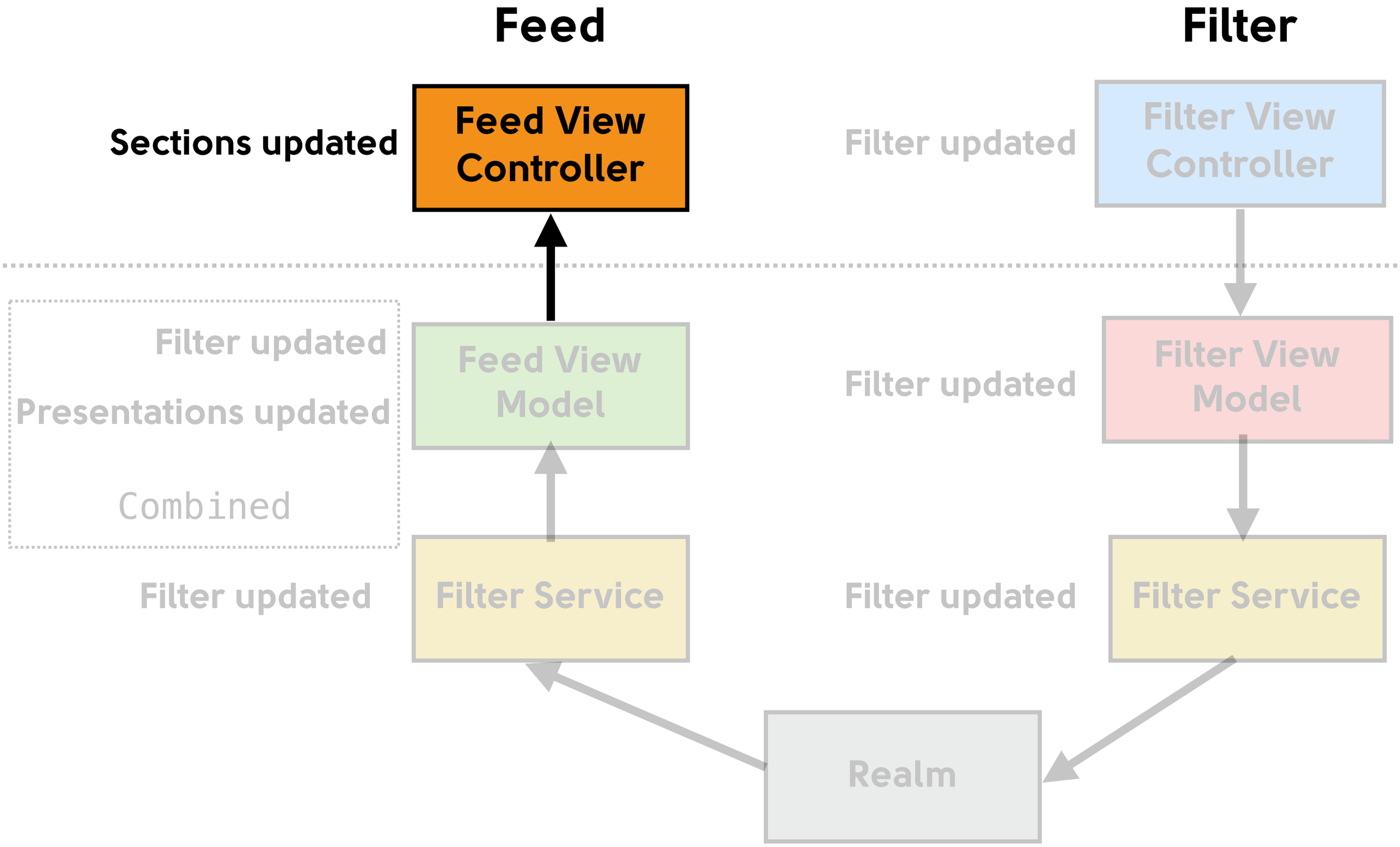
# View Model

```
Observable.combineLatest(presentationService
                         .presentationsUpdated(),
                          currentFilters
                         .asObservable())
            { _, filters in
                return filters
            }
            .flatMap { [weak self] filters in
                return self.presentationService
                           .filteredPresentations(with: filters)
            }
            .map { [weak self] presentations in
                return self.cellViewModelFactory
                           .sections(from: presentations)
            }
            .bindTo(_presentations)
            .disposed(by: disposeBag)
```

# View Model

```
Observable.combineLatest(presentationService
                         .presentationsUpdated(),
                          currentFilters
                         .asObservable())
            { _, filters in
                return filters
            }
            .flatMap { [weak self] filters in
                return self.presentationService
                           .filteredPresentations(with: filters)
            }
            .map { [weak self] presentations in
                return self.cellViewModelFactory
                           .sections(from: presentations)
            }
            .bindTo(_presentations)
            .disposed(by: disposeBag)
```

# Подписка на storage



# Преимущества архитектуры

- Работа с событиями в функциональном стиле
- Удобная работа с асинхронными операциями
- Декларативное описание взаимодействия между компонентами

**при этом**

- Возможность легко заменить делегатное взаимодействие через Observable/PublishSubject

# Преимущества архитектуры

- Работа с событиями в функциональном стиле
- Удобная работа с асинхронными операциями
- Декларативное описание взаимодействия между компонентами

**при этом**

- Возможность легко заменить делегатное взаимодействие через Observable/PublishSubject

# Преимущества архитектуры

- Работа с событиями в функциональном стиле
- Удобная работа с асинхронными операциями
- Декларативное описание взаимодействия между компонентами

**при этом**

- Возможность легко заменить делегатное взаимодействие через Observable/PublishSubject

# Преимущества архитектуры

- Работа с событиями в функциональном стиле
- Удобная работа с асинхронными операциями
- Декларативное описание взаимодействия между компонентами

при этом

- Возможность легко заменить делегатное взаимодействие через Observable/PublishSubject

# Преимущества архитектуры

- Работа с событиями в функциональном стиле
- Удобная работа с асинхронными операциями
- Декларативное описание взаимодействия между компонентами

**при этом**

- Возможность легко заменить делегатное взаимодействие через Observable/PublishSubject

# Преимущества архитектуры

- Работа с событиями в функциональном стиле
- Удобная работа с асинхронными операциями
- Декларативное описание взаимодействия между компонентами

при этом

- Возможность легко заменить делегатное взаимодействие через Observable/PublishSubject

# Недостатки

- Производительность при частых сигналах
- Большой порог вхождения
- Относительно сложная отладка

# Недостатки

- Производительность при частых сигналах
- Большой порог вхождения
- Относительно сложная отладка

# Недостатки

- Производительность при частых сигналах
- Большой порог вхождения
- Относительно сложная отладка

# Спасибо!

https://github.com/StachkaConf/ios-app