

A photograph of a rocket launch at night. The rocket is ascending vertically, leaving a bright, conical trail of fire and smoke. In the foreground, two people are silhouetted against the dark landscape; one is holding a large umbrella. The scene is set against a dark sky with some distant lights and structures visible on the horizon.

Reactive demystified

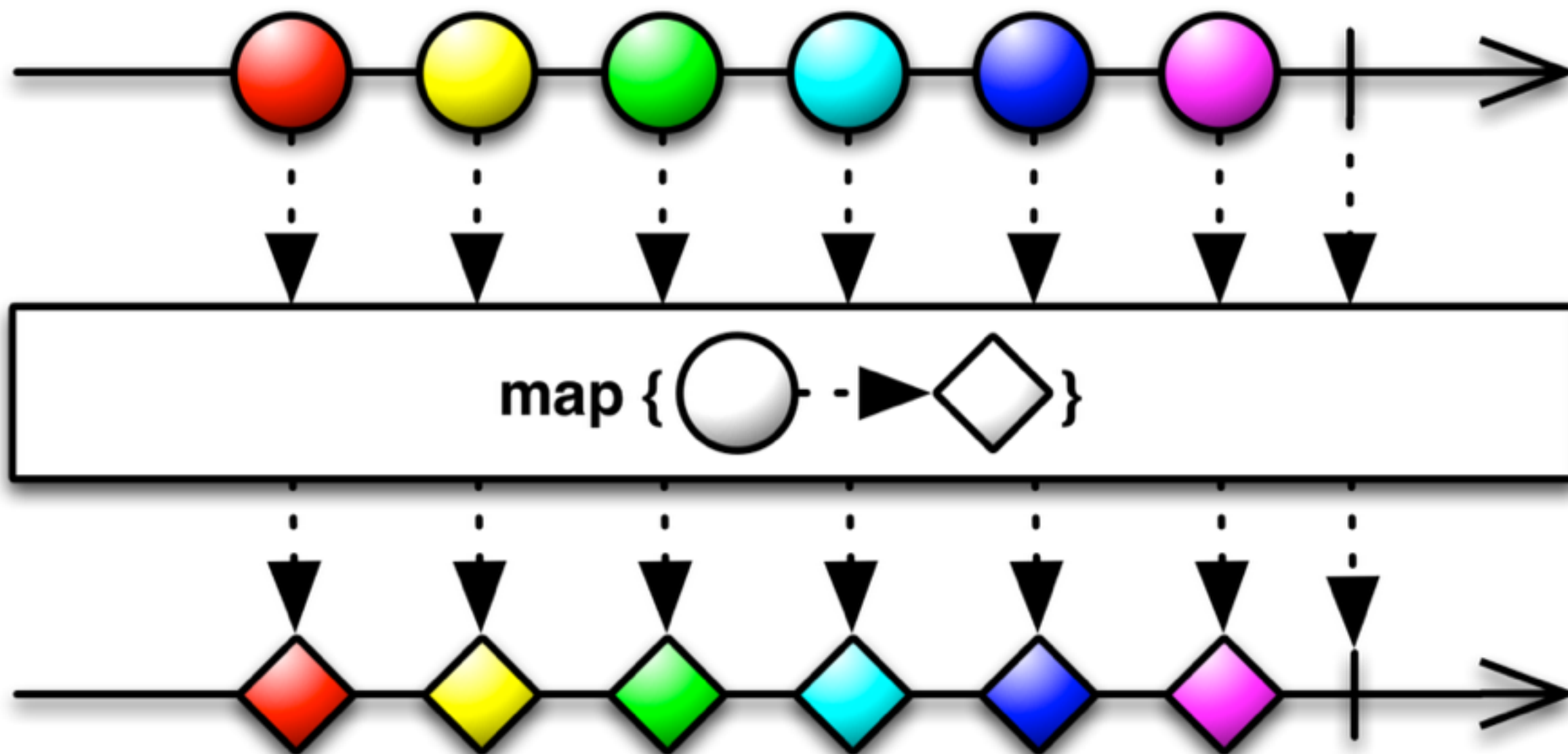
Что это?

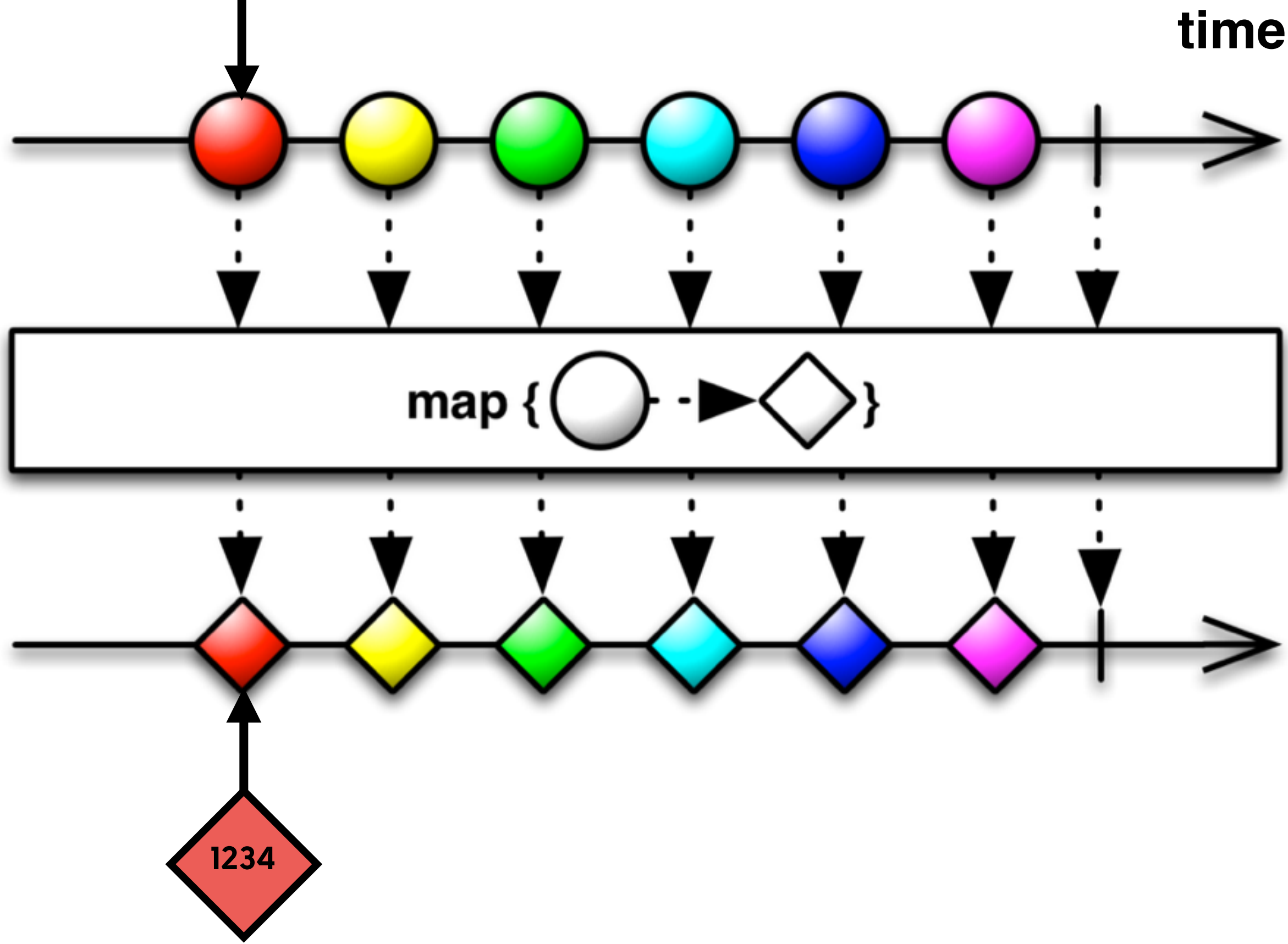
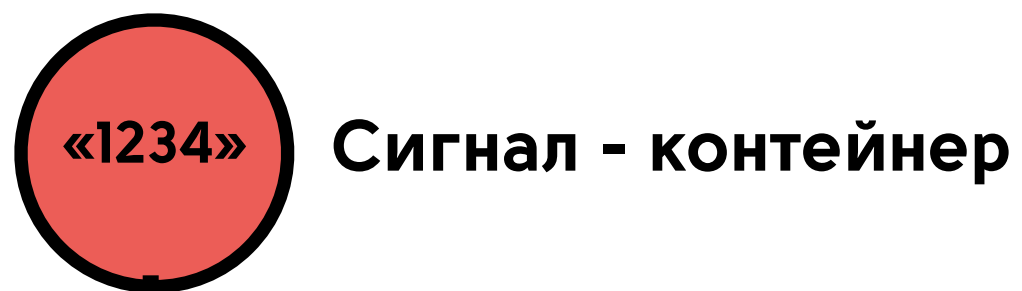
- **Способ программирования, оперирующий со специальными типами данных, изменяющимися в зависимости от времени (сигналами)**
- Сигналы - последовательности значений, к которым можно применить функциональные преобразования

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





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)**

The background is a vibrant, abstract composition. On the left side, there is a prominent spiral pattern that draws the eye inward. This spiral is composed of concentric rings of various colors, including deep blues, bright greens, and earthy browns. To the right of the spiral, the background transitions into a series of radiating, curved lines that sweep across the frame. These lines are colored in a spectrum of reds, oranges, yellows, and blues, creating a sense of dynamic movement and energy. The overall effect is one of a complex, swirling vortex of color and form.

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 like
river

Time a

```
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)

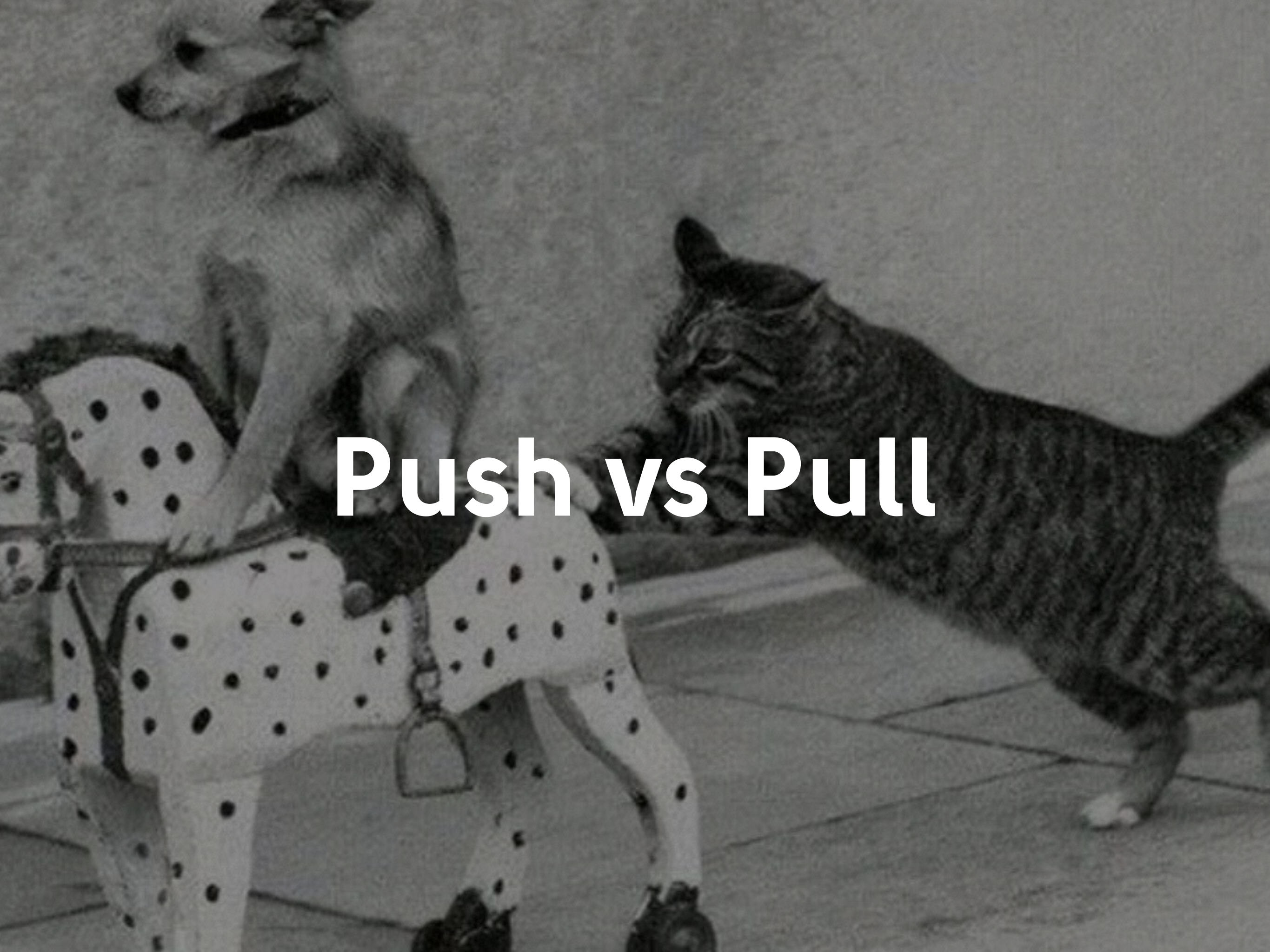
Метрики

- 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 передачи - 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 передачи - 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 передачи - 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 передачи - 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 передачи - например, Queue**

Как это применимо к Rx?

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

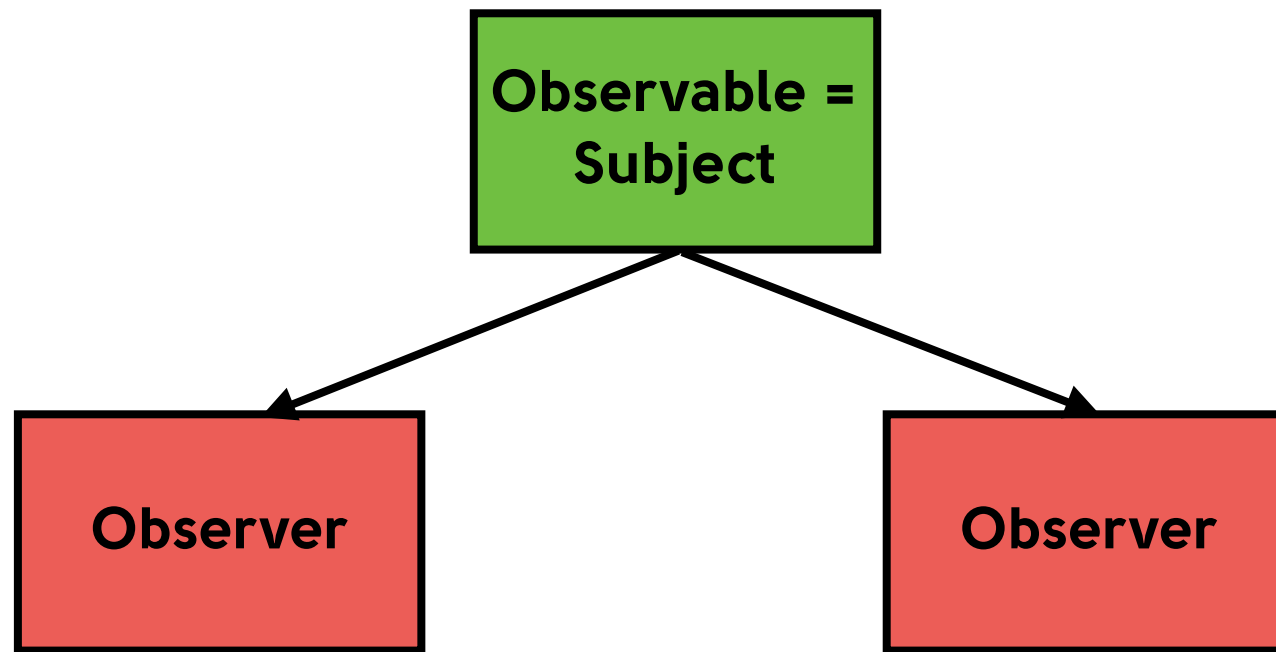
Как это применимо к Rx?

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

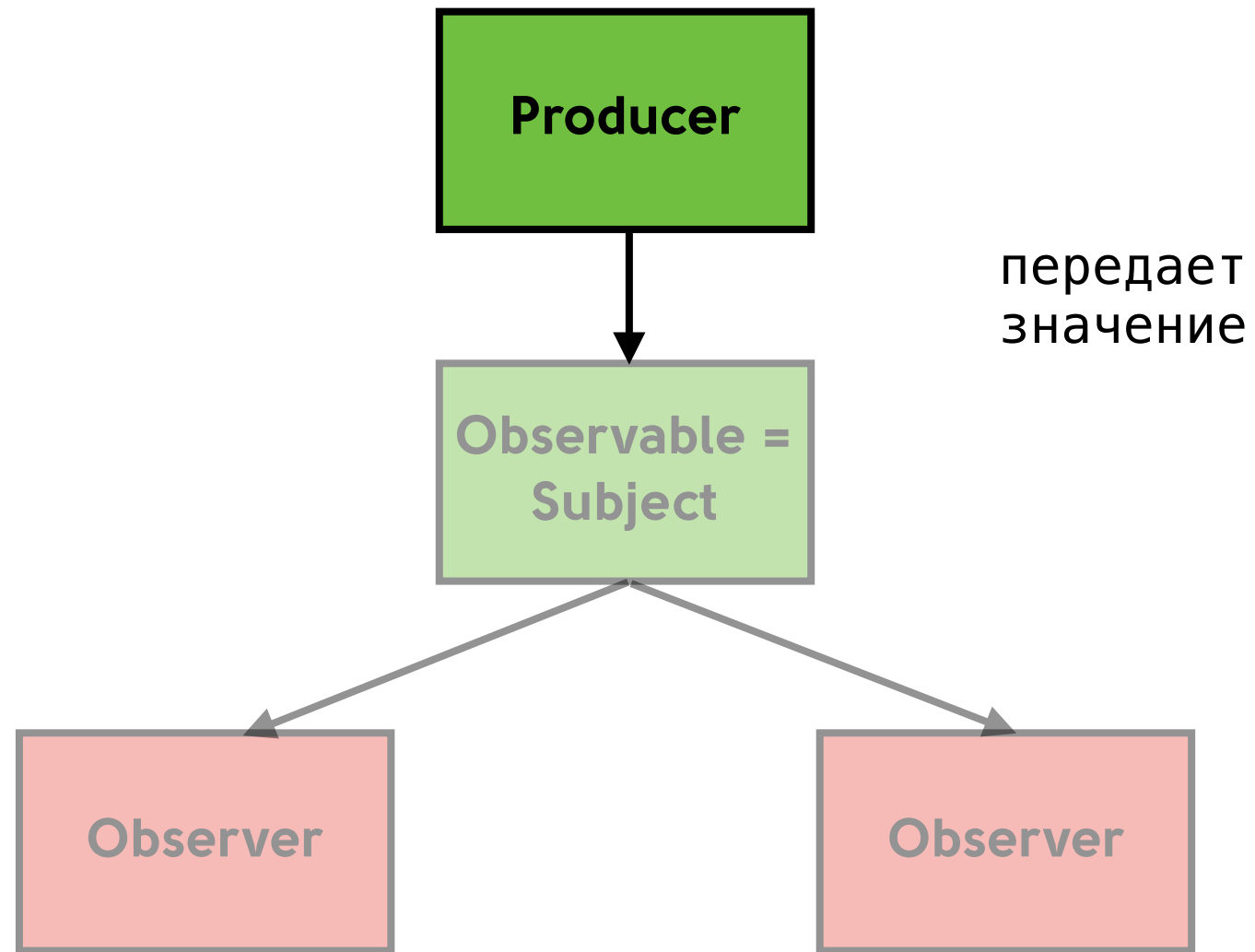


Observer vs Iterator

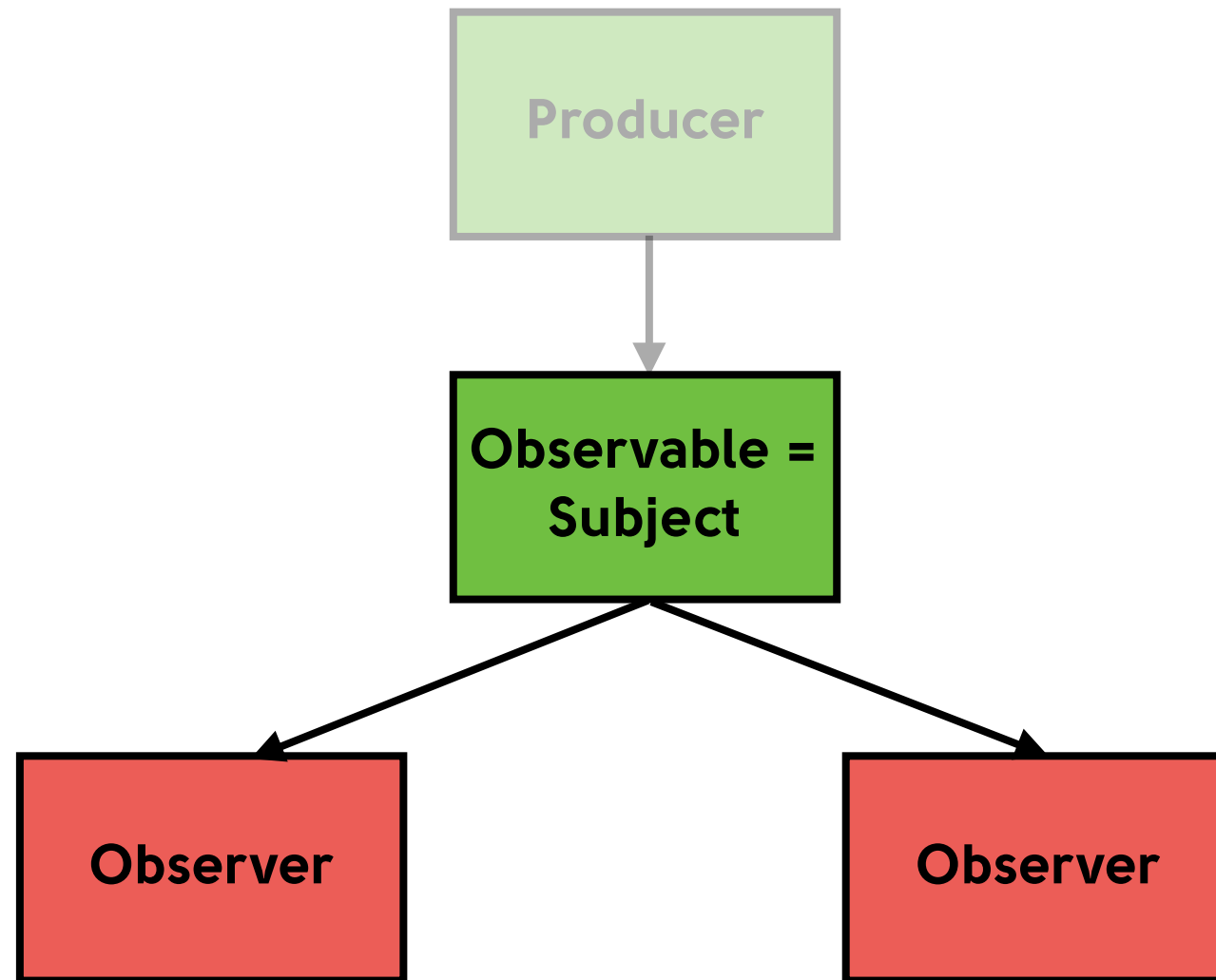
Observer pattern



Observer pattern



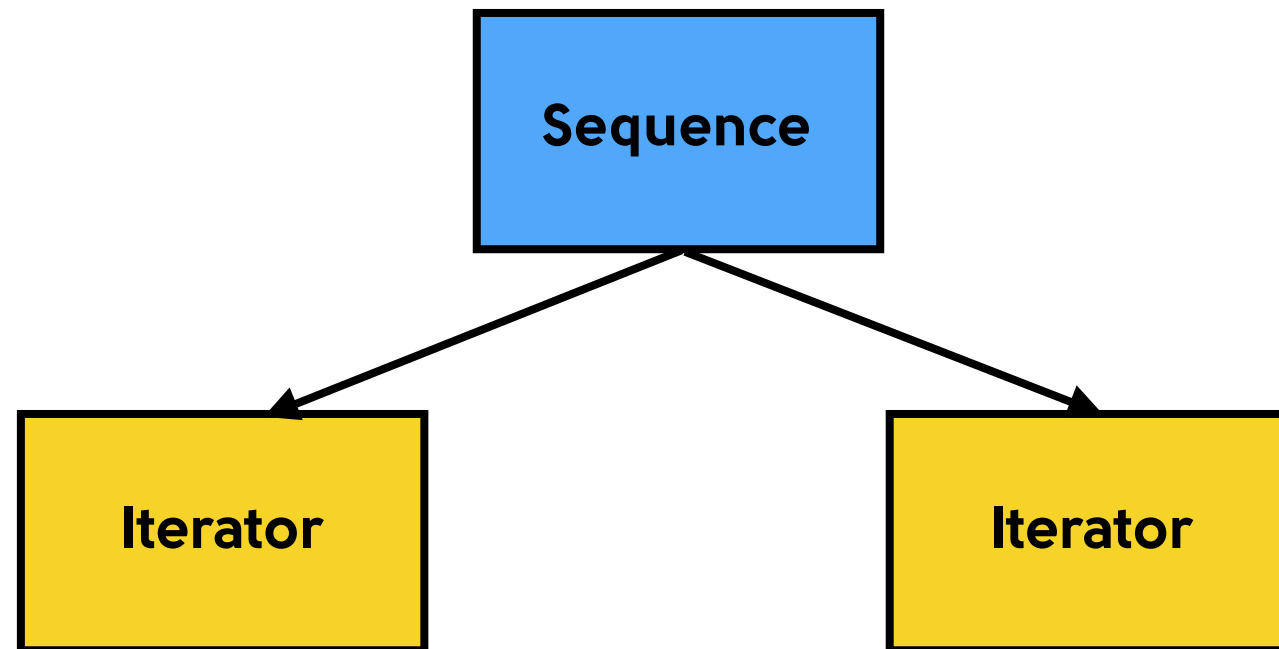
Observer pattern



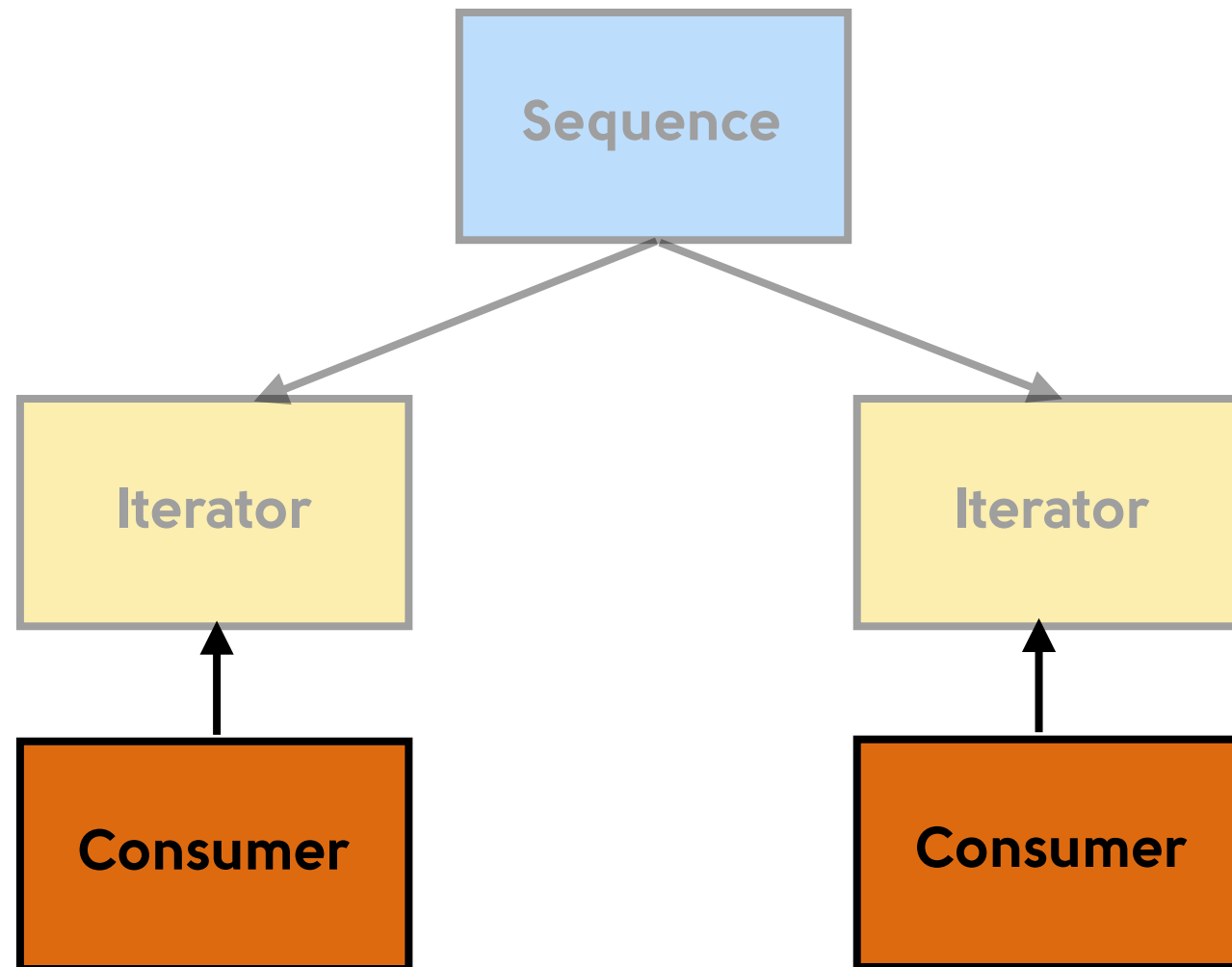
```
class Observable<Element> {  
    func subscribe(_ observer: Observer<Element>)  
}
```

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

Iterator pattern

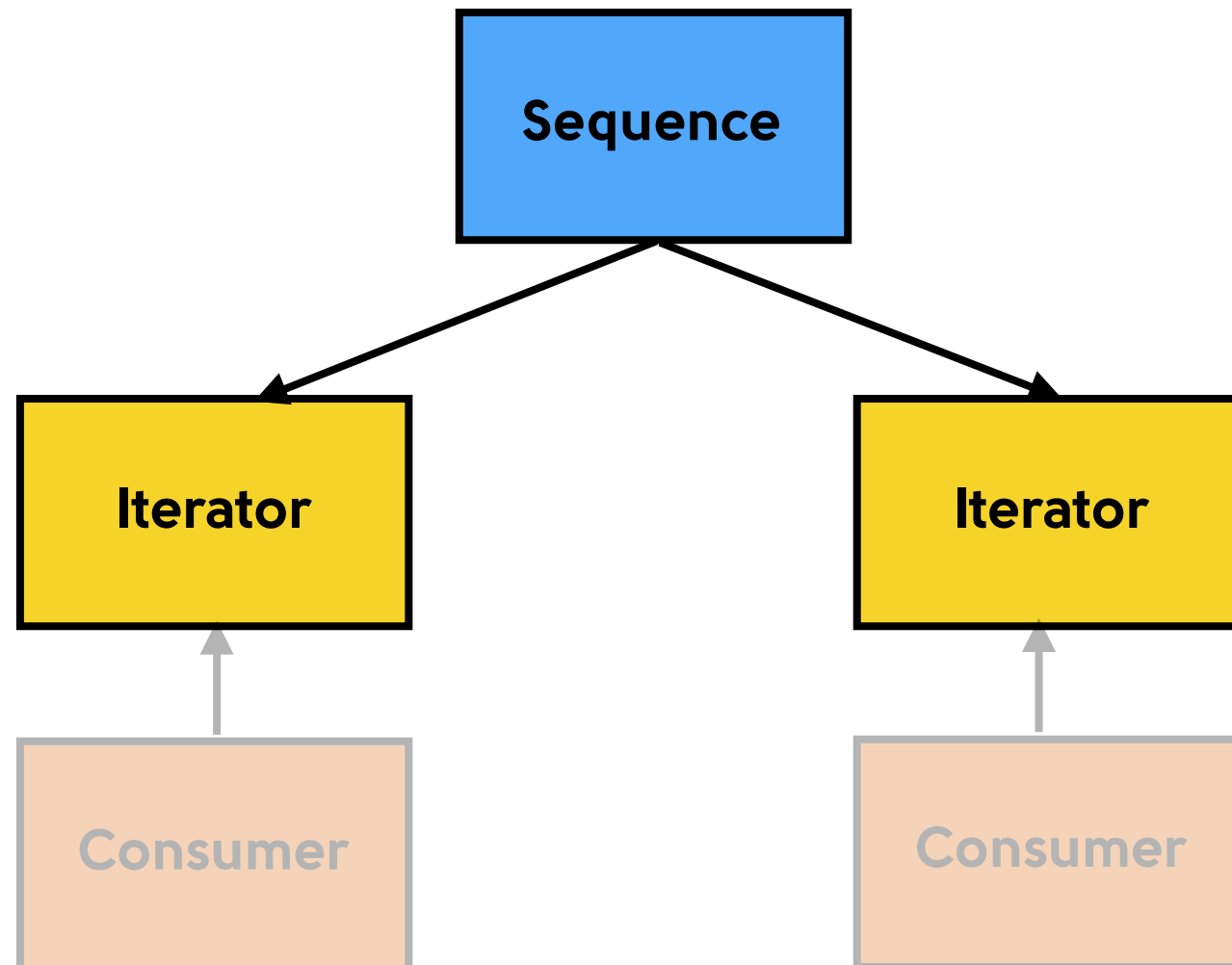


Iterator pattern



Получает
значение

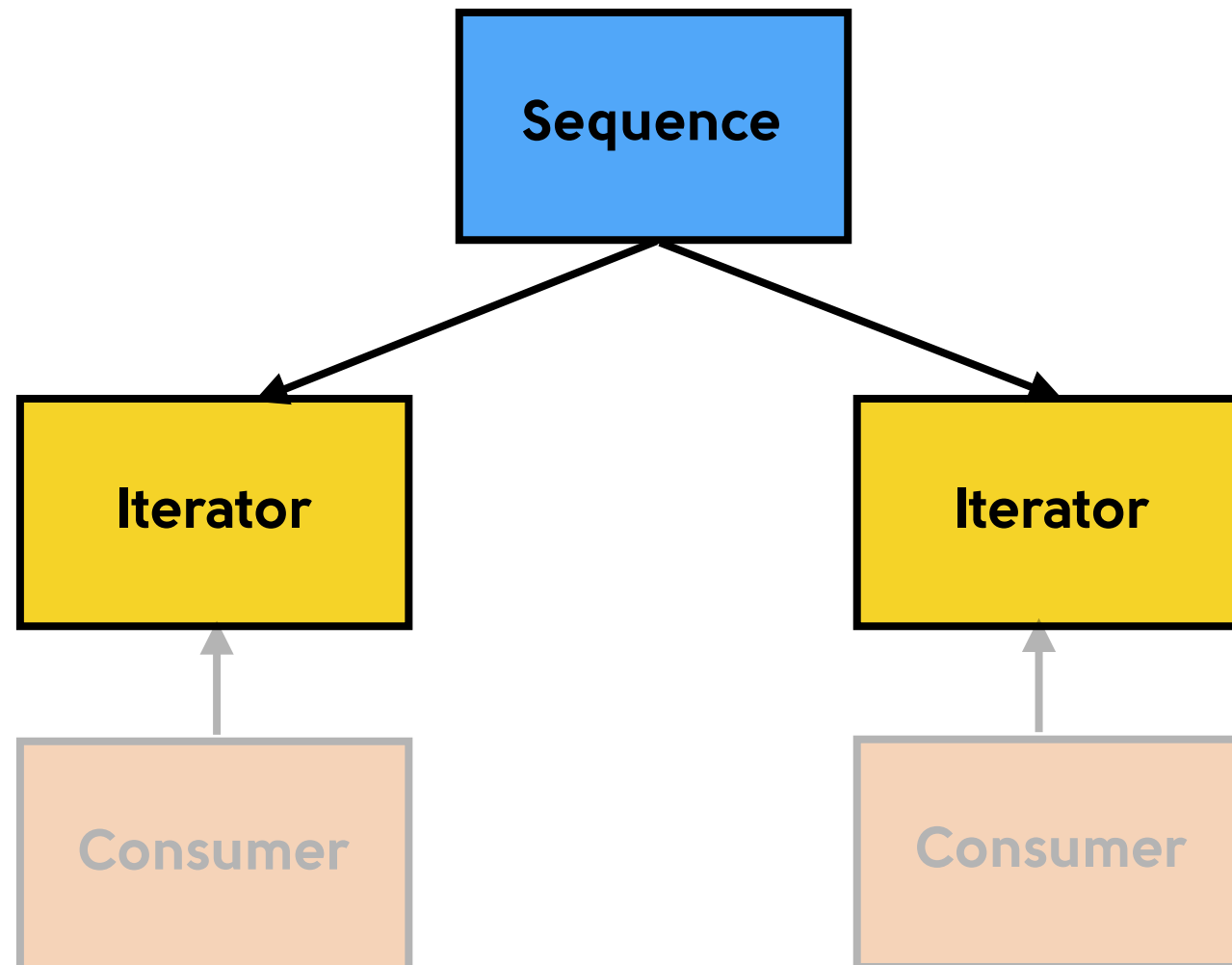
Iterator pattern



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

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

Iterator pattern



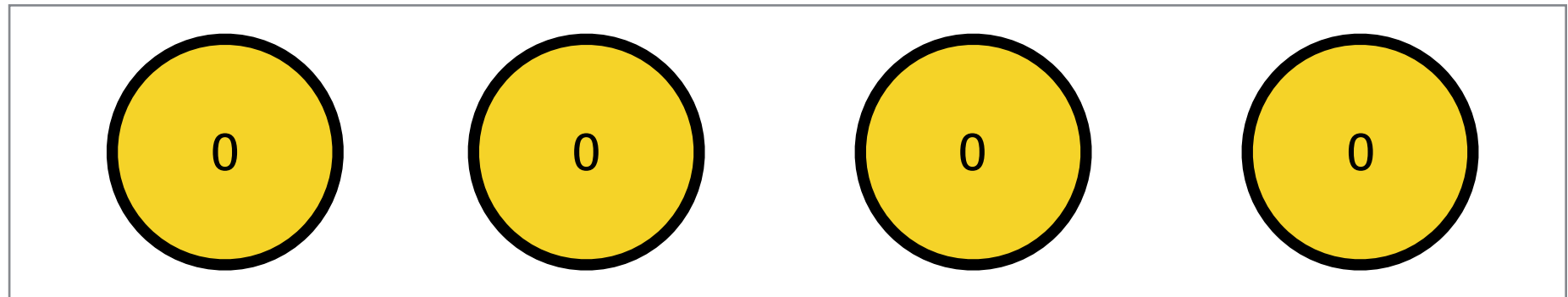
map
flatMap
filter
reduce

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

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

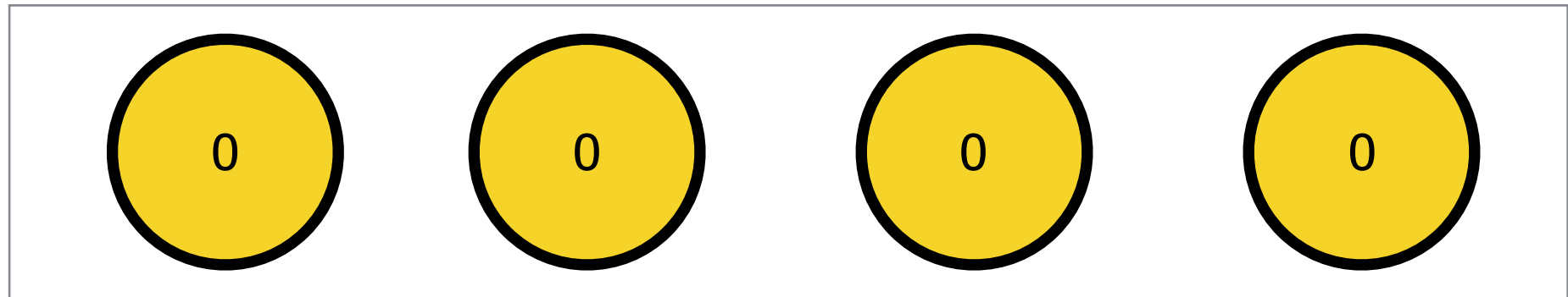
Преобразование последовательности

Iterator
<0>

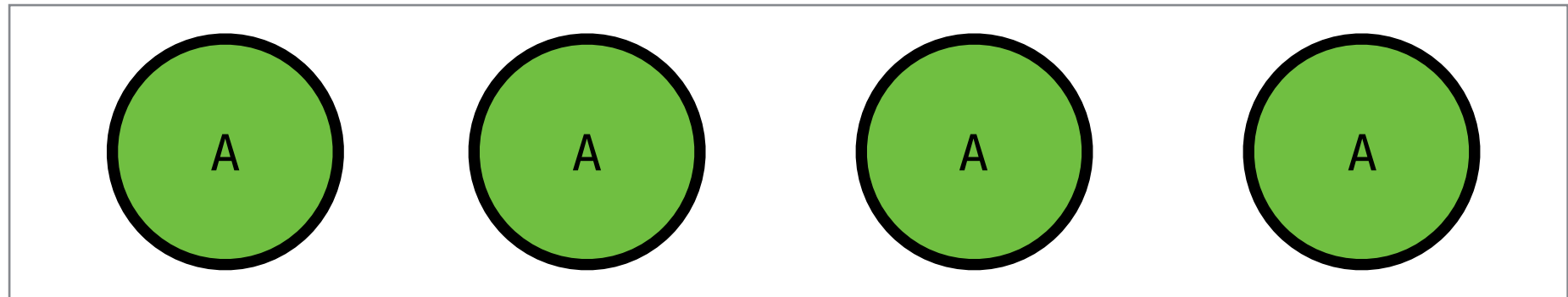


Преобразование последовательности

Iterator
<0>

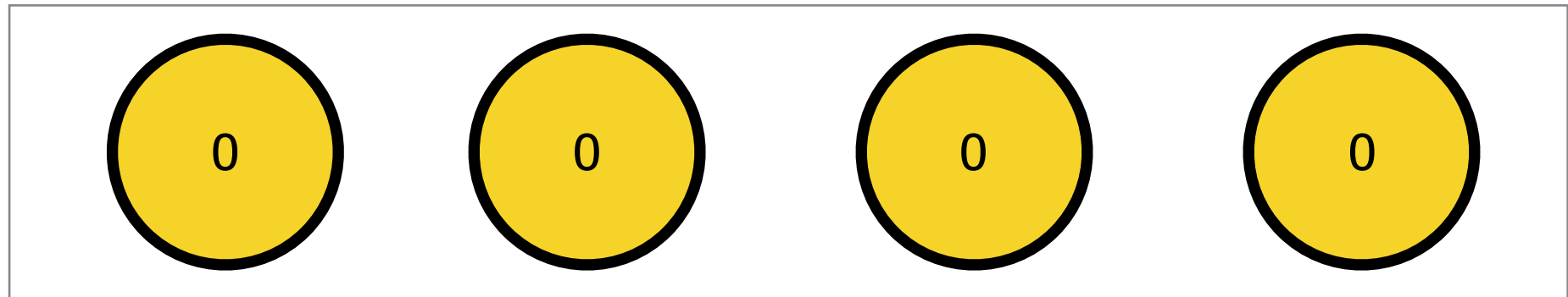


Iterator
<A>

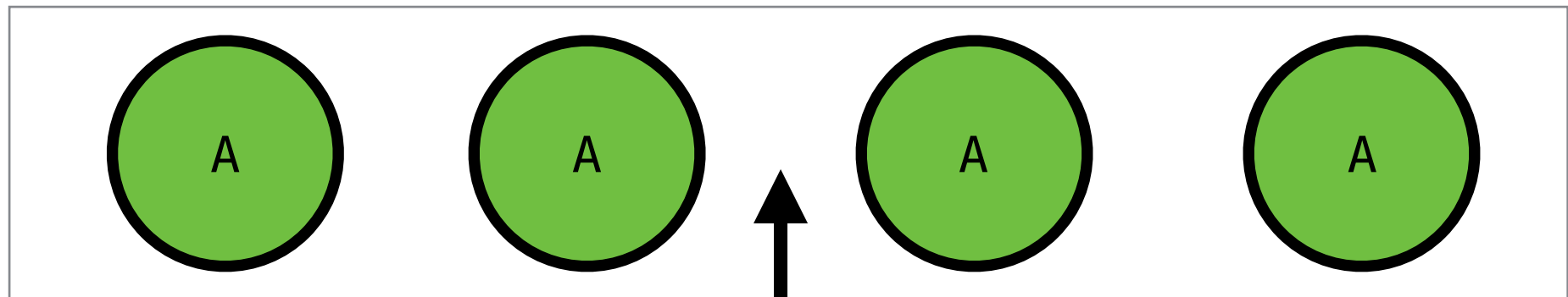


Преобразование последовательности

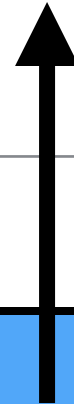
Iterator
<0>



Iterator
<A>

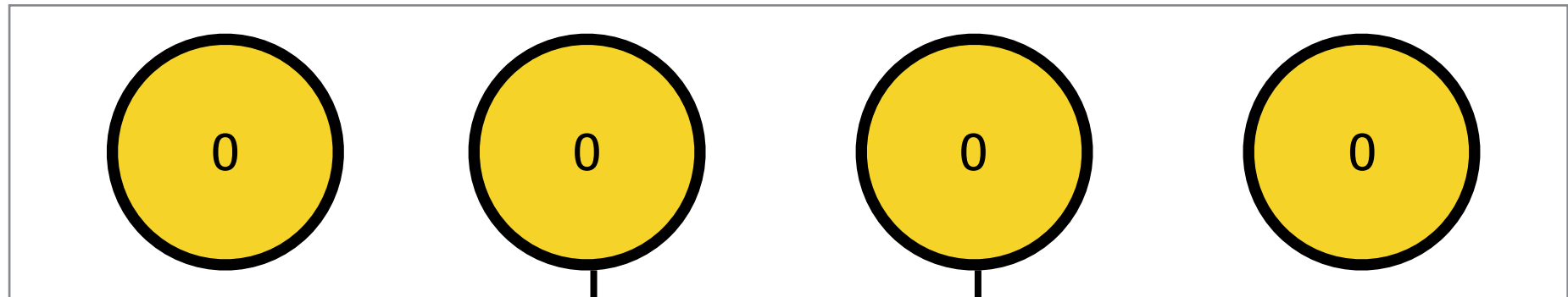


Consumer (Pull)



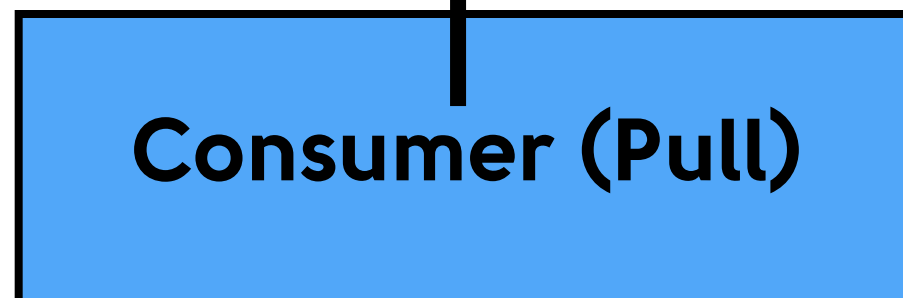
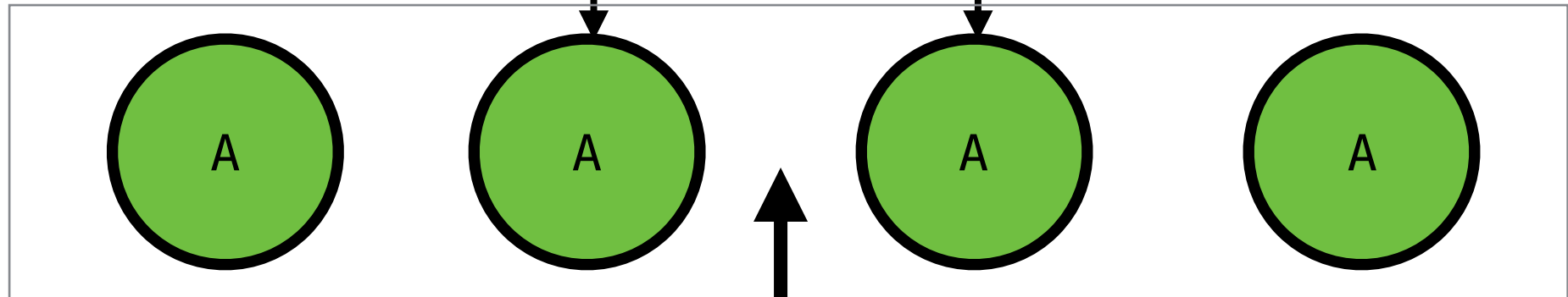
Преобразование последовательности

Iterator
<0>



f = (0) -> (A)

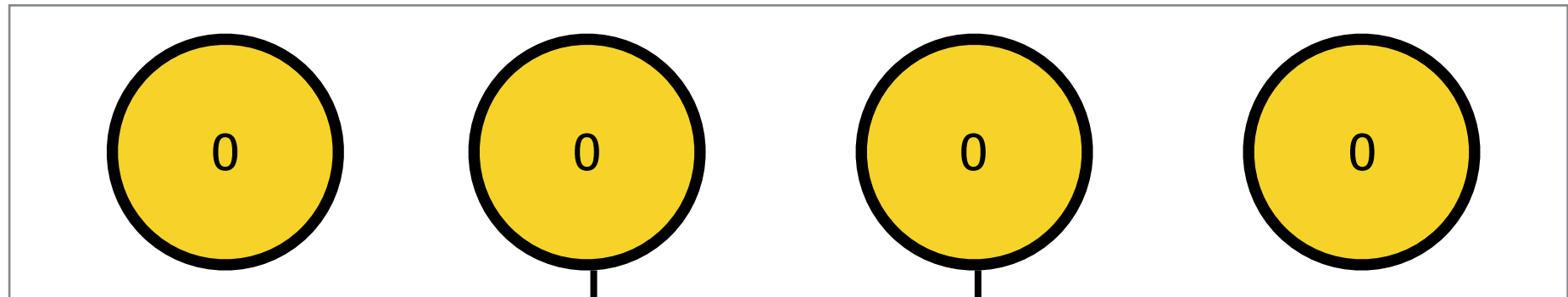
Iterator
<A>



Преобразование последовательности

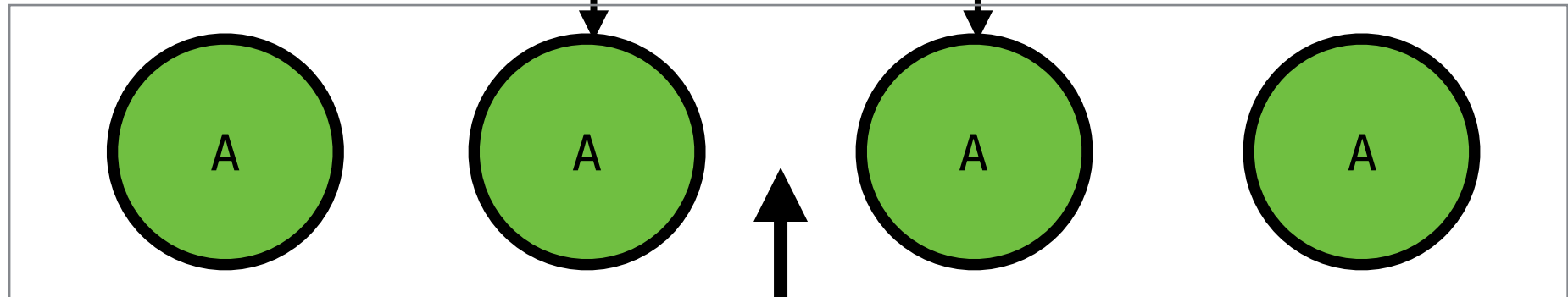
`element = f(Iterator<0>.next)`

Iterator
<0>



$f = (0) \rightarrow (A)$

Iterator
<A>



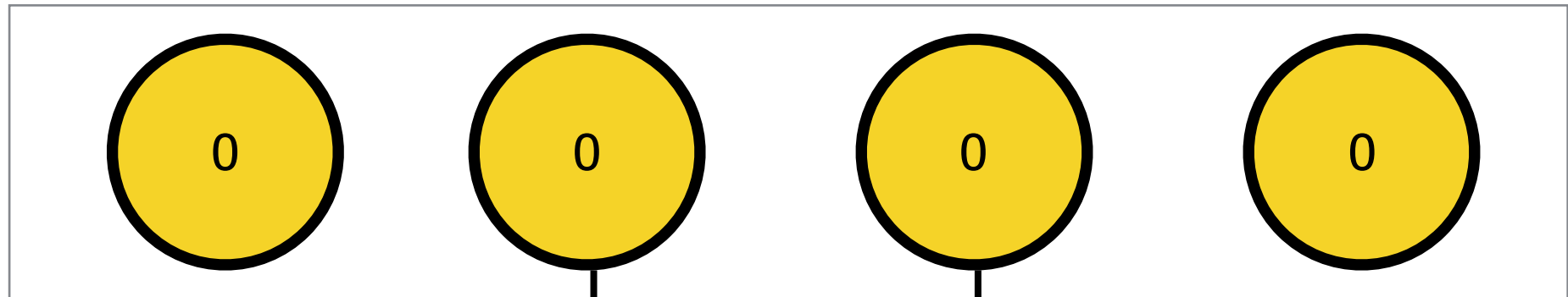
Consumer (Pull)



Преобразование последовательности

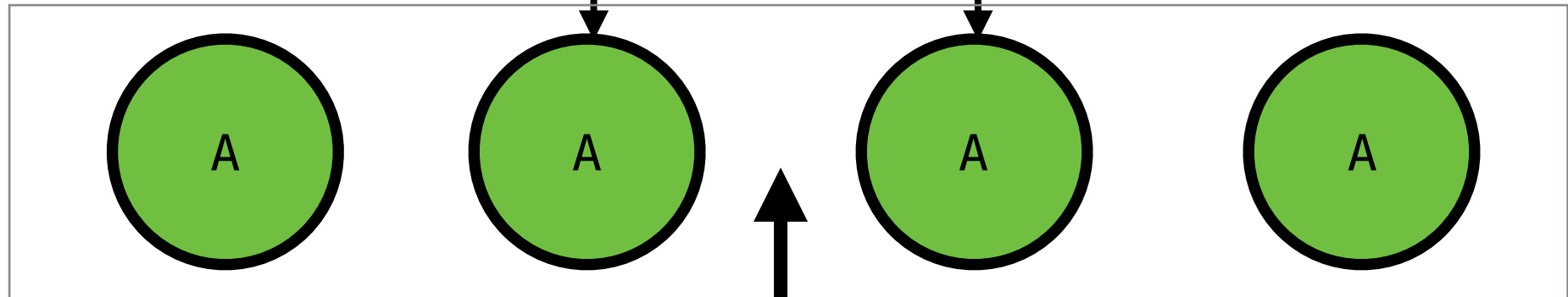
`element = f(Iterator<0>.next)`

Iterator
<0>



$f = (0) \rightarrow (A)$

Iterator
<A>

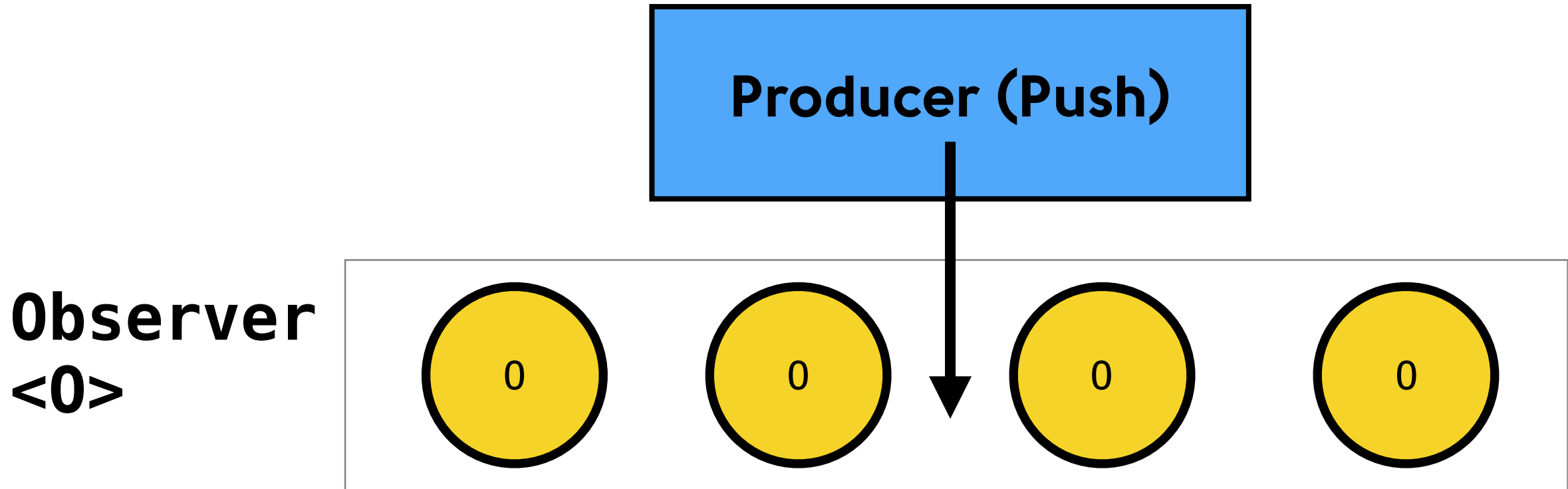


MAP

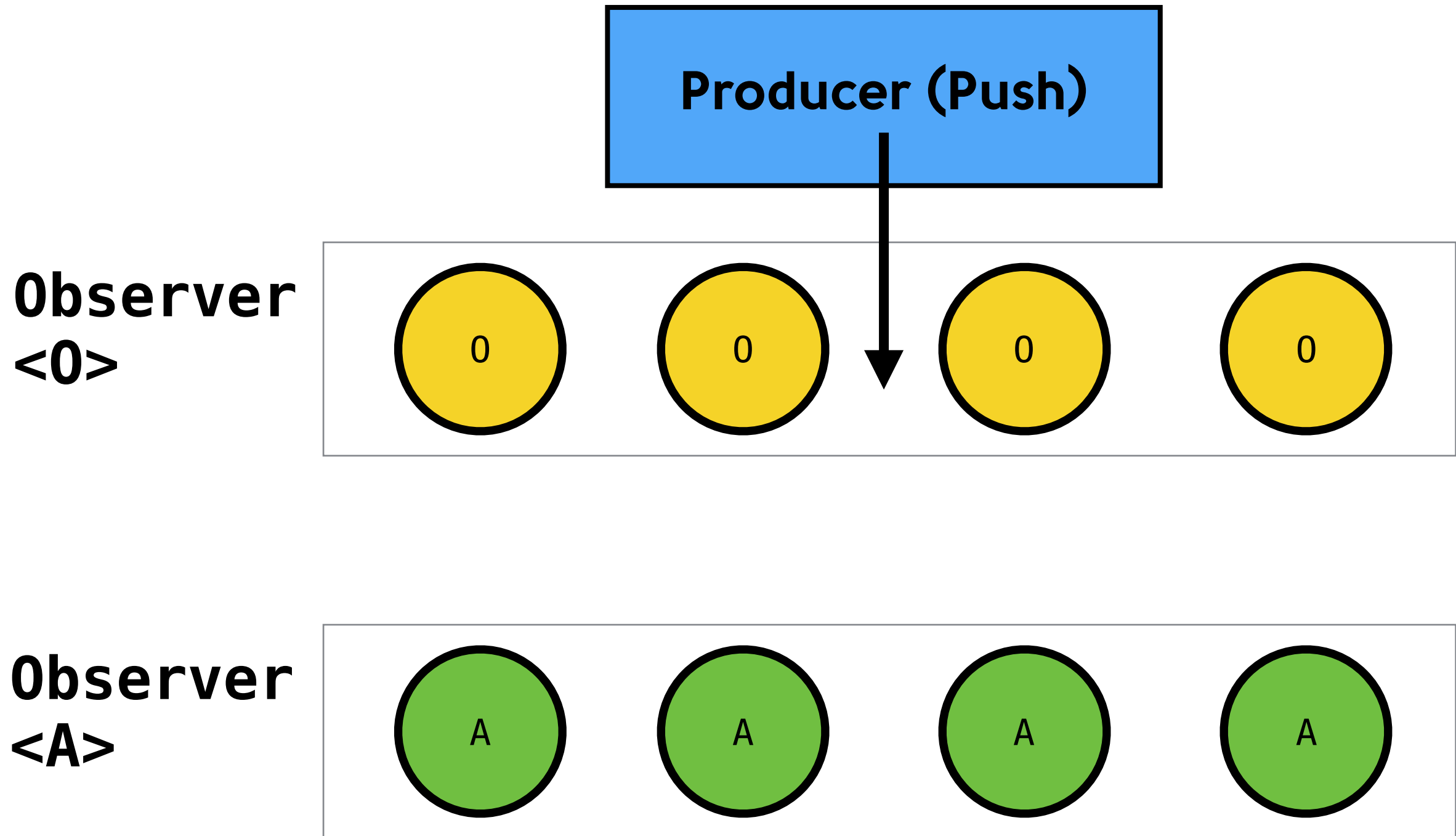
Consumer (Pull)

A blue rectangular box with a black outline. An arrow points from the top center of the box to the second green circle in the row above.

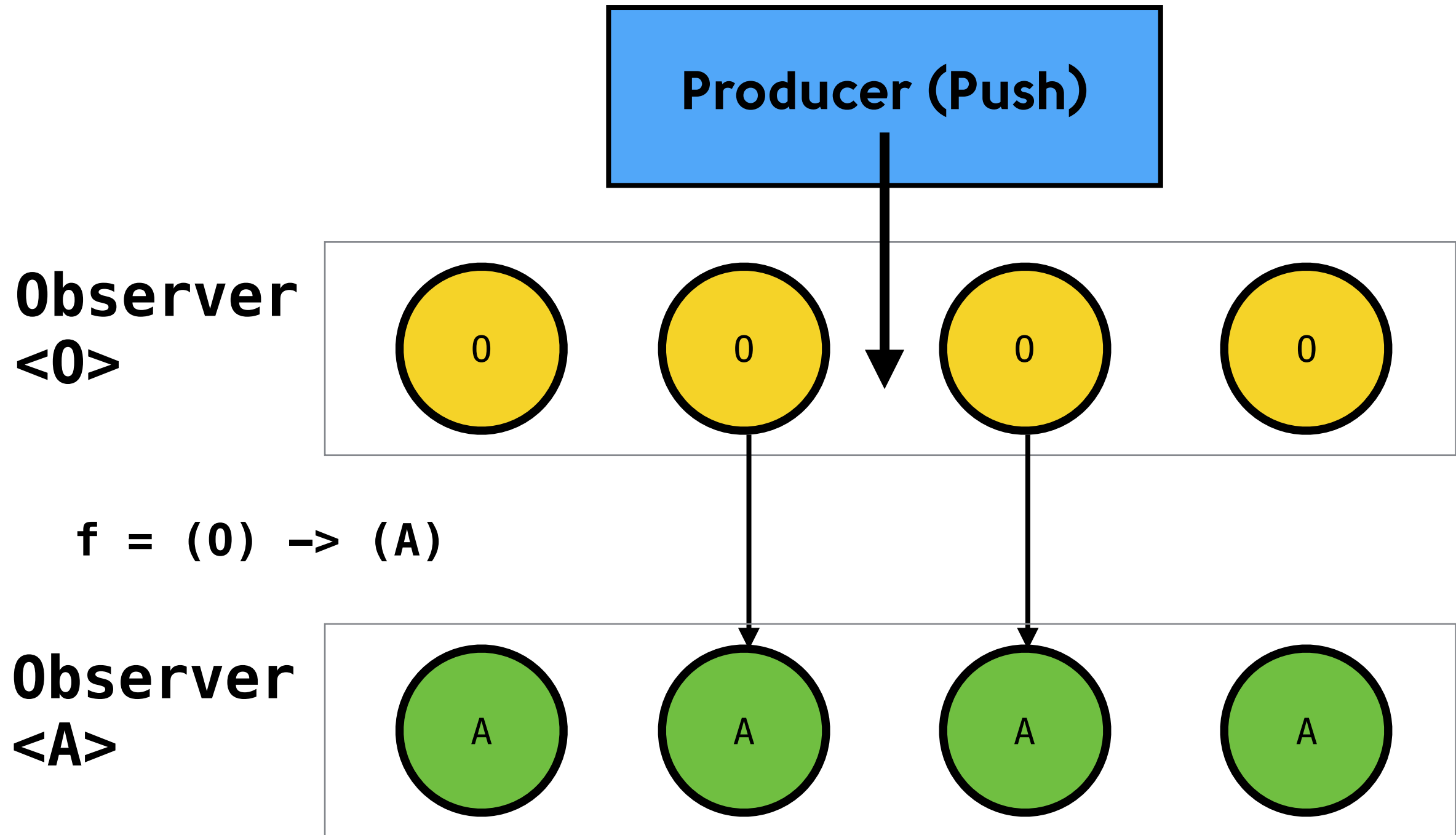
Преобразование последовательности



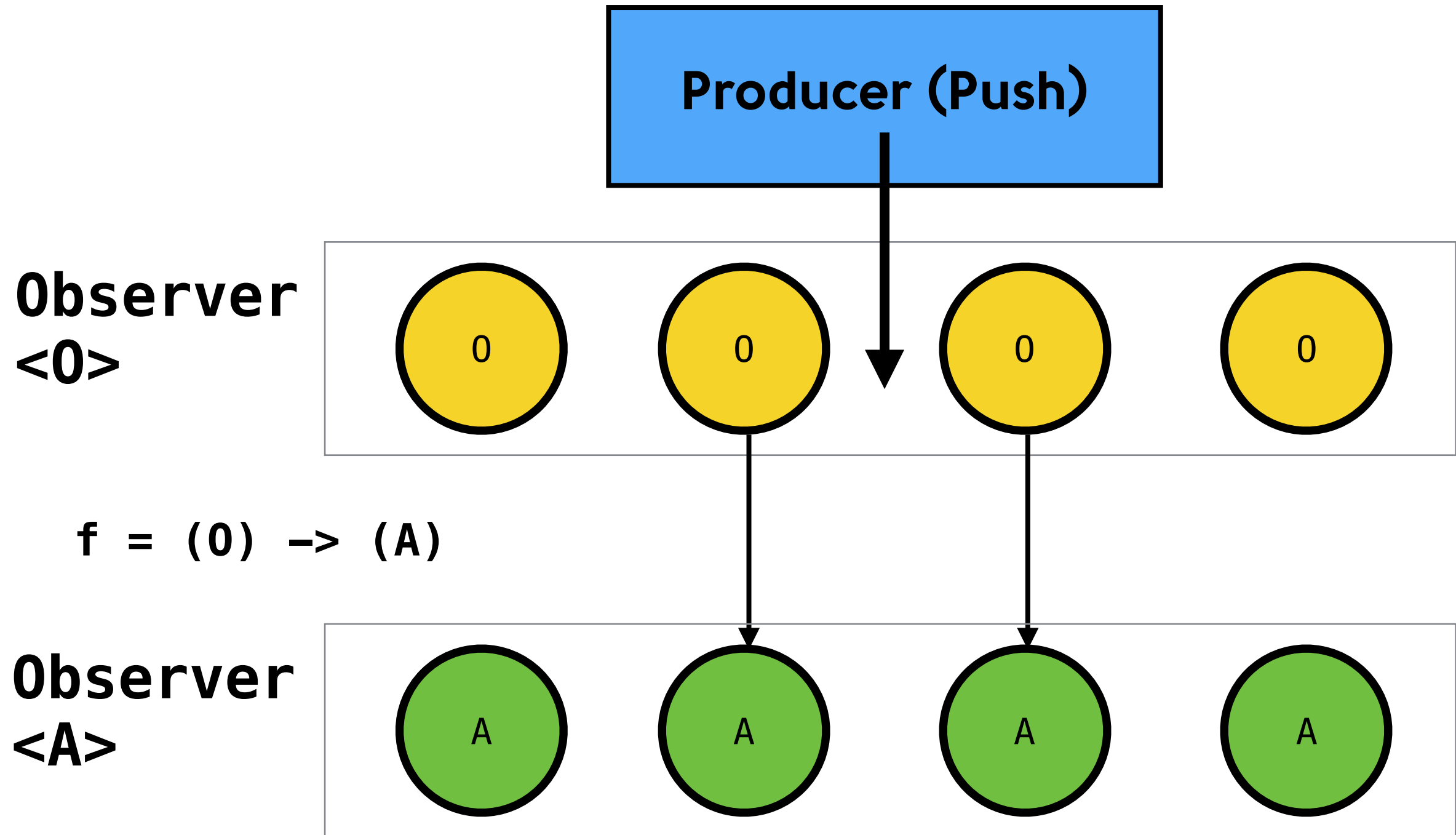
Преобразование последовательности



Преобразование последовательности

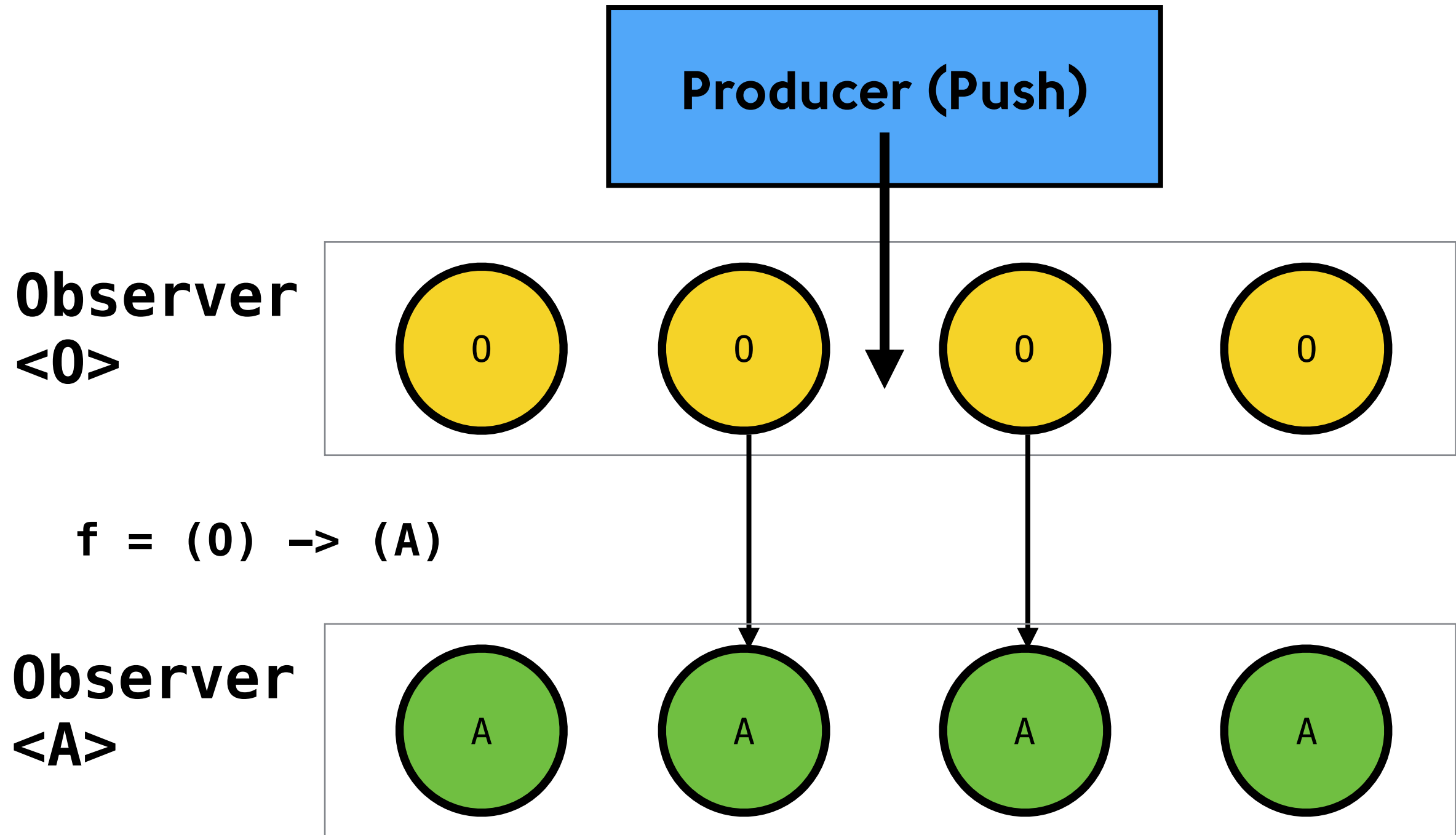


Преобразование последовательности



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

Преобразование последовательности

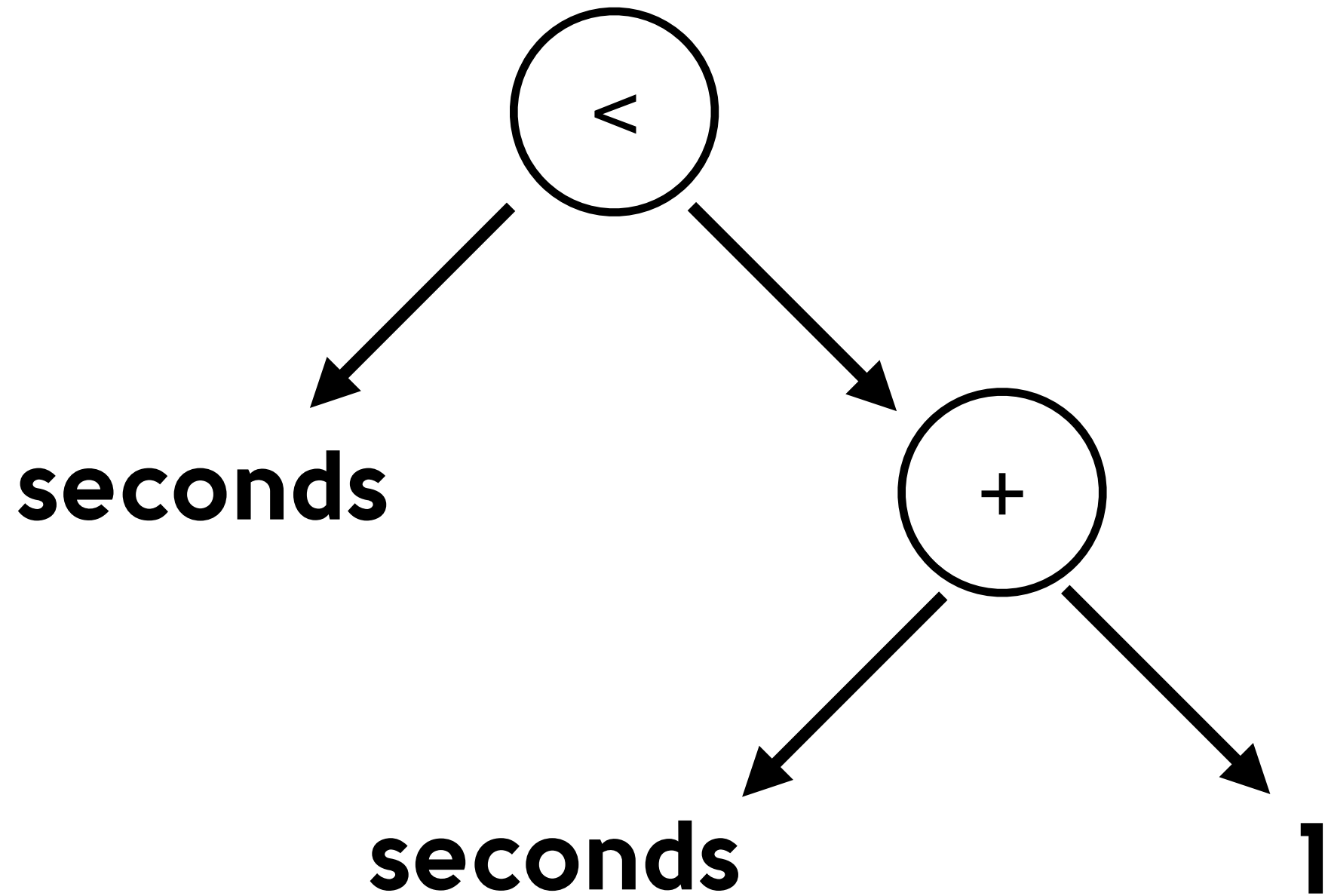


MAP

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

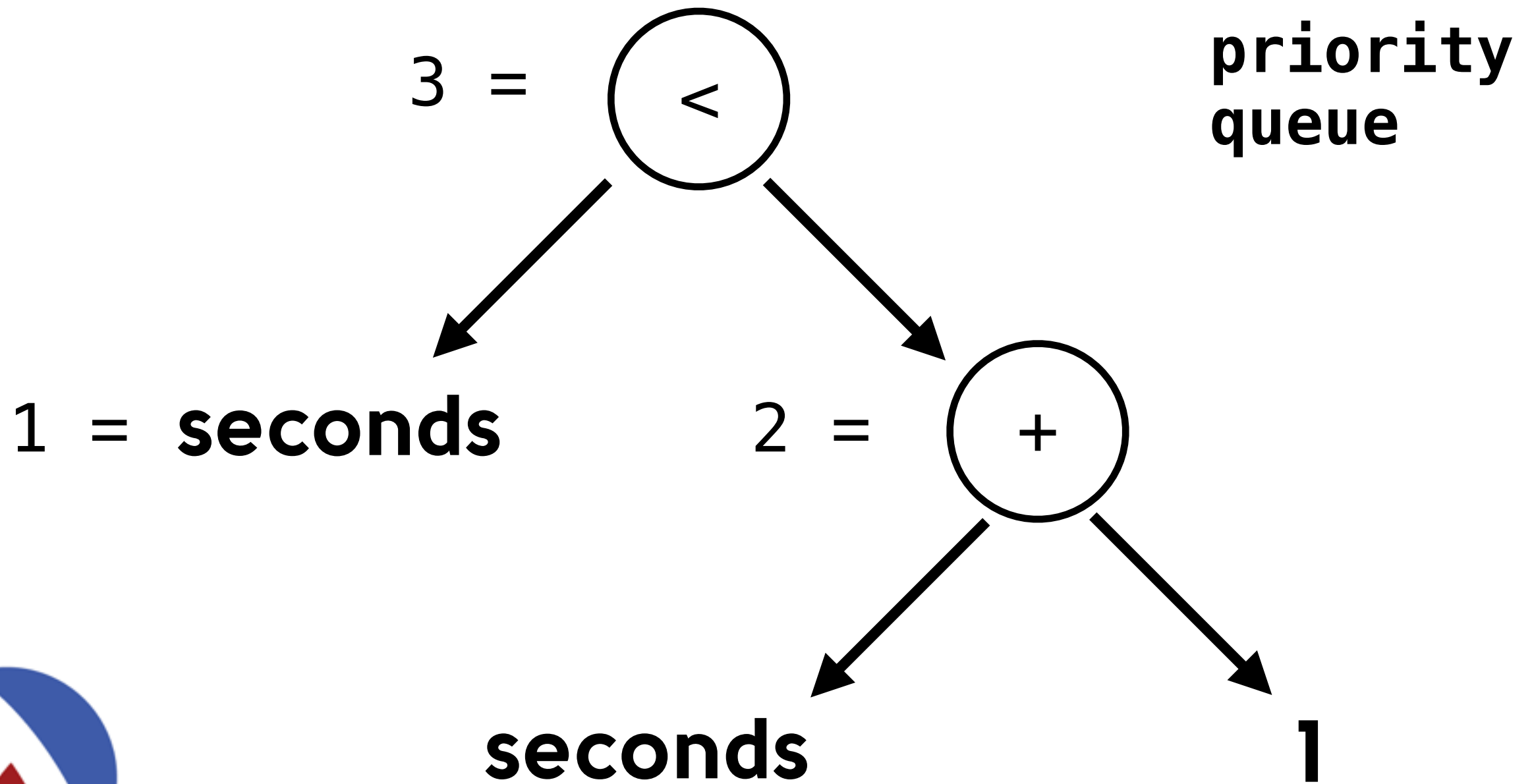
Glitch avoidance

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)
```

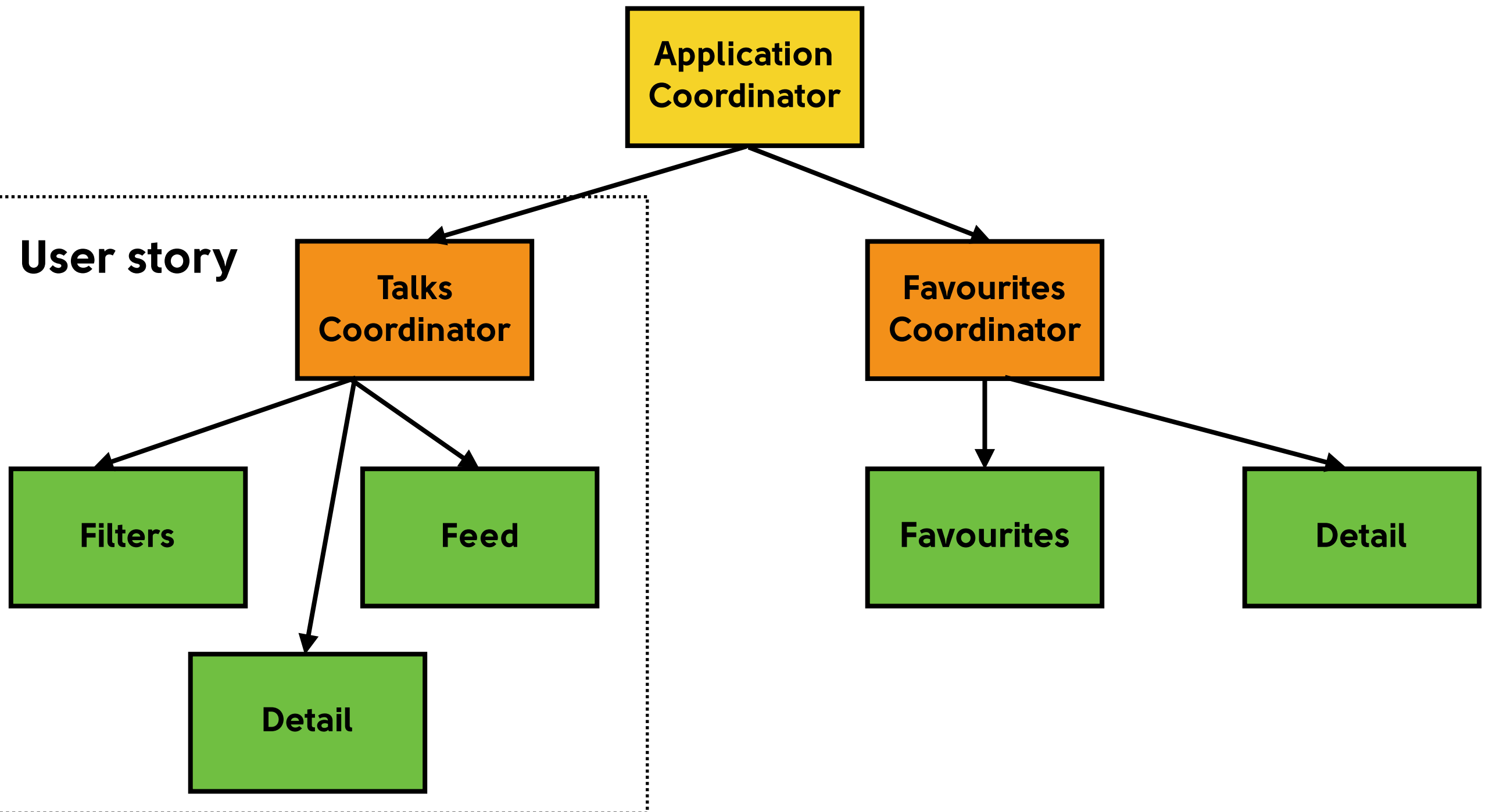
false

true

The background is a black and white abstract architectural composition. On the left, there is a grid of white lines forming a series of overlapping squares and rectangles, creating a sense of depth and perspective. To the right of this grid, a series of parallel white lines recede into the distance, also creating a strong sense of perspective. The right side of the image is a solid black field. The word 'Архитектура' is centered in the middle of the image, overlapping the grid and the perspective lines.

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()
```

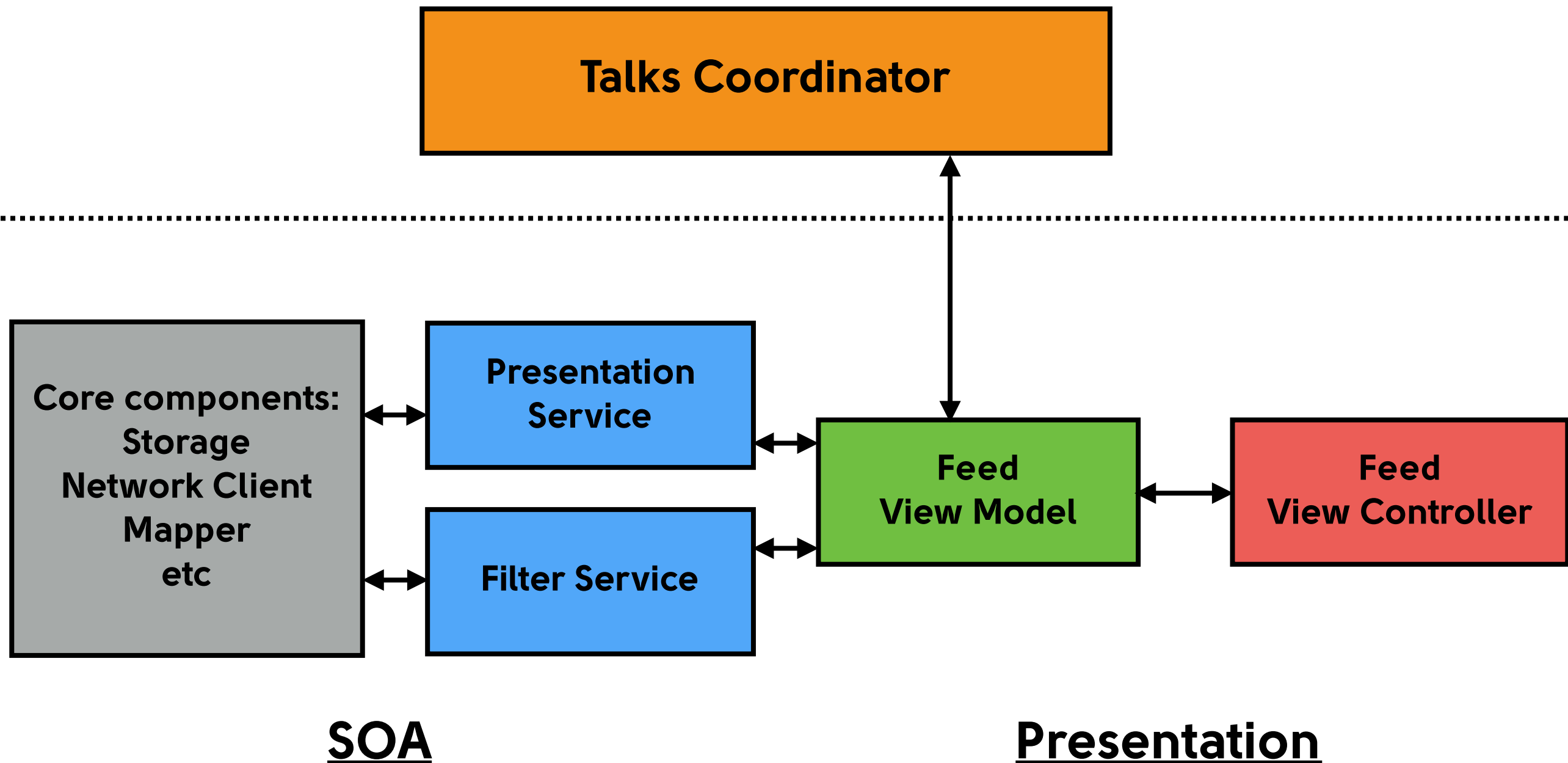
Пример (Application Coordinator)

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

```
let talksCoordinator =  
factory.talksCoordinator  
(rootNavigationController: favouritesNC)
```

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

Архитектура отдельного модуля



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



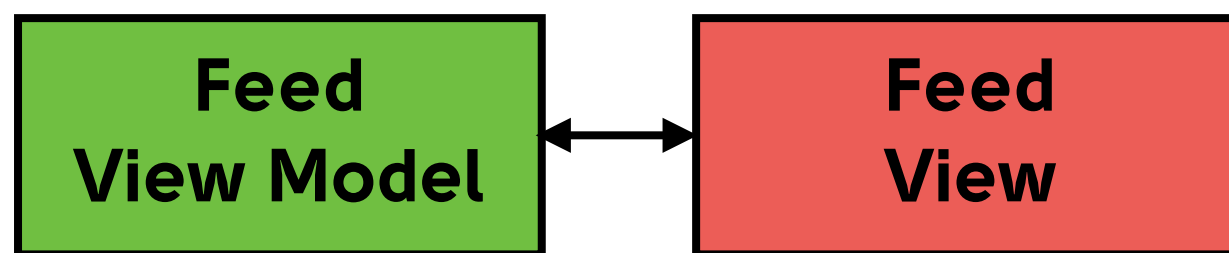
VS



||

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)
```

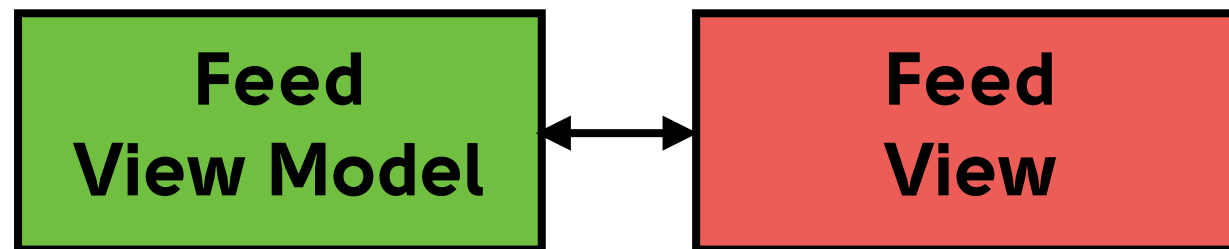
View Model

```
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

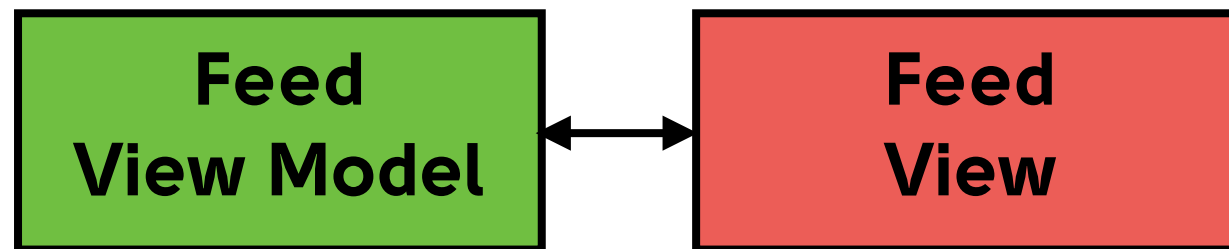
```
fileprivate var _presentations:  
Variable<[PresentationSectionModel]> = Variable([])  
  
var presentations: Observable<[PresentationSectionModel]> {  
    return _presentations.asObservable()  
}  
  
view.indexSelected  
    .flatMap { indexPath in  
        let presentationKey = ...  
        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 }  
}
```

.....

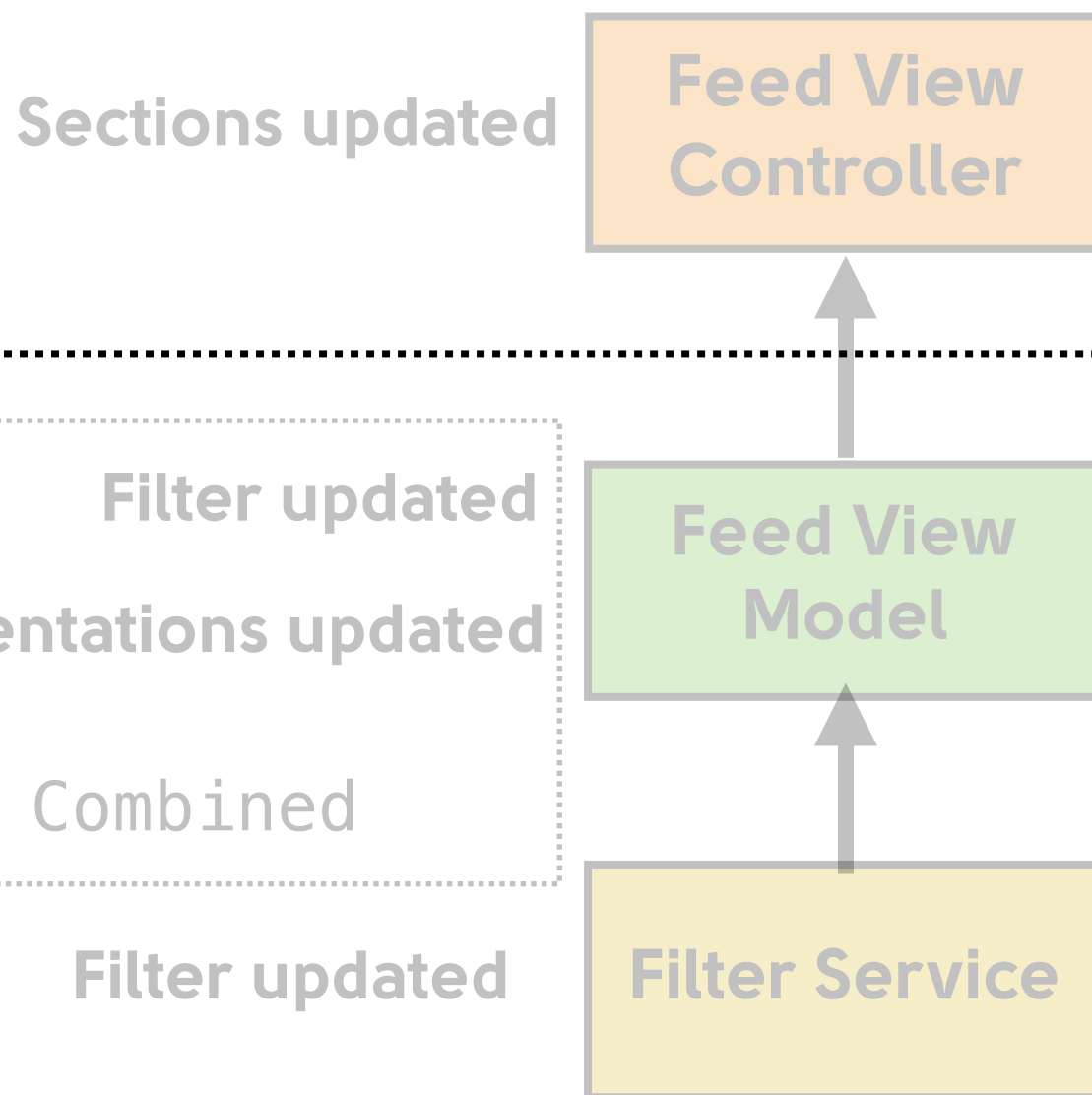
```
protocol FiltersModuleOutput {  
    var filtersChanged:  
        Observable<[Filter]> { get }  
}
```

На уровне координатора

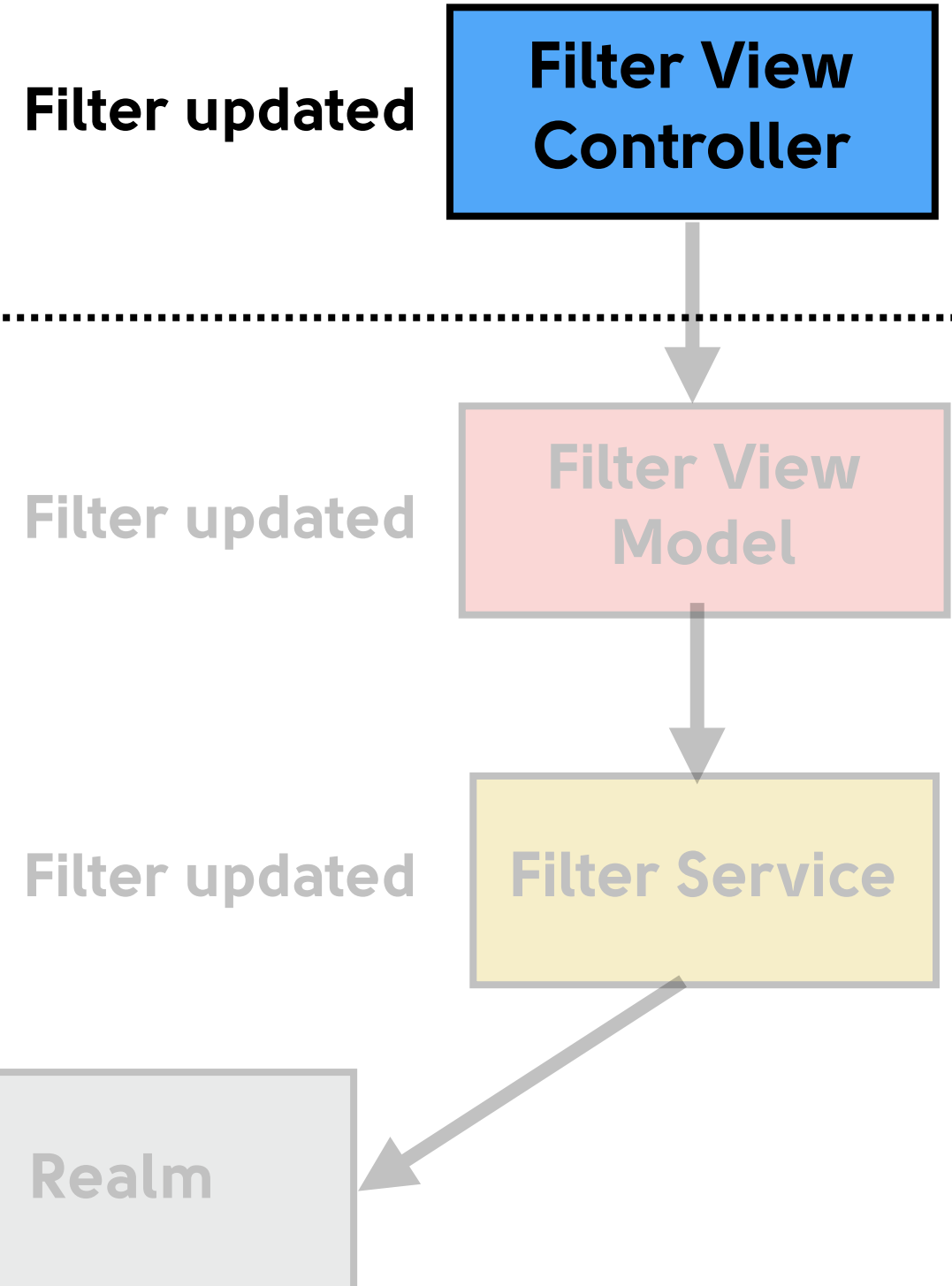
```
filtersModuleOutput  
    .filtersChanged  
    .bindTo(feedViewModel.didChangeFilters)  
    .disposed(by: disposeBag)
```

Подписка на storage

Feed

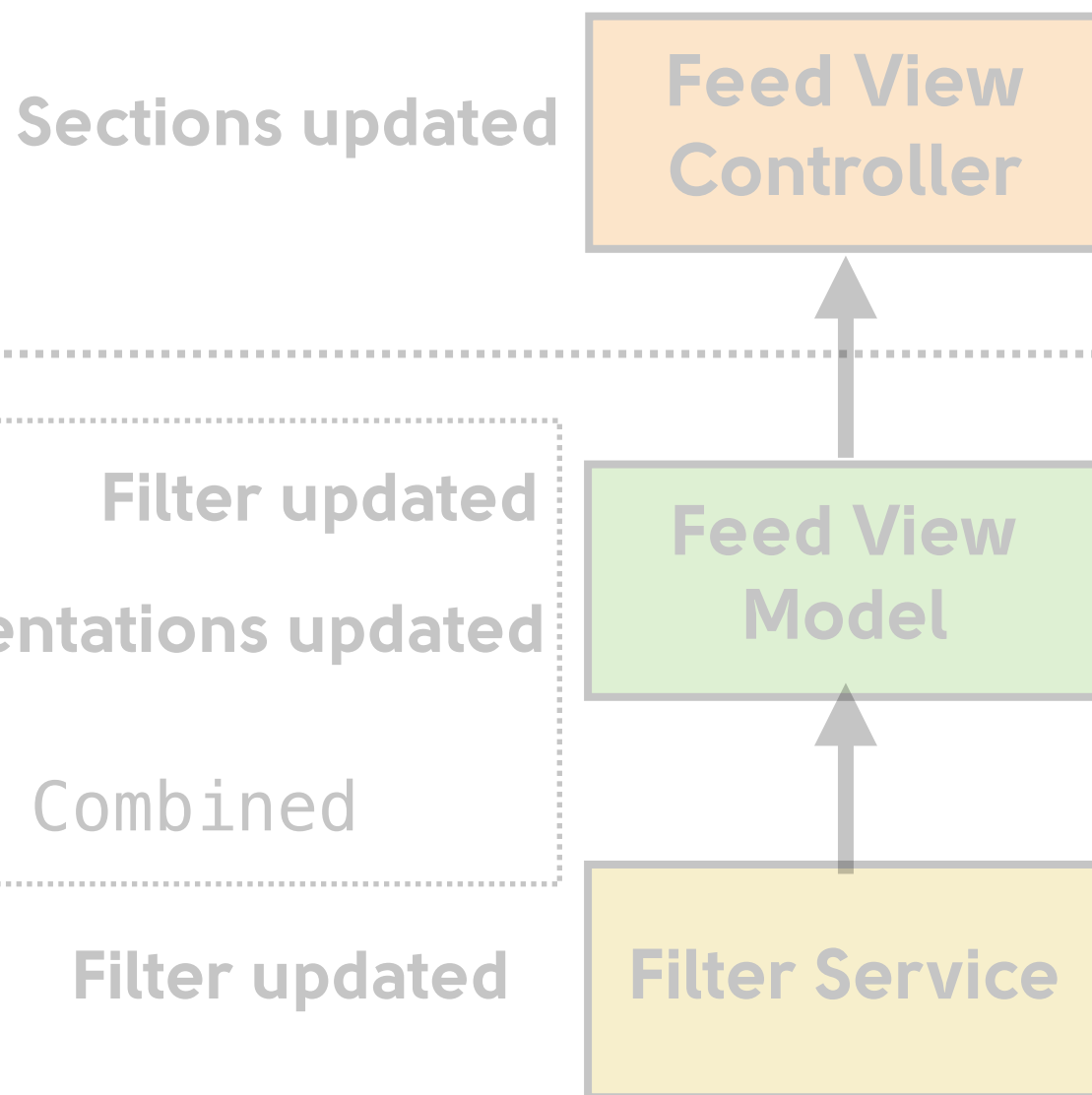


Filter

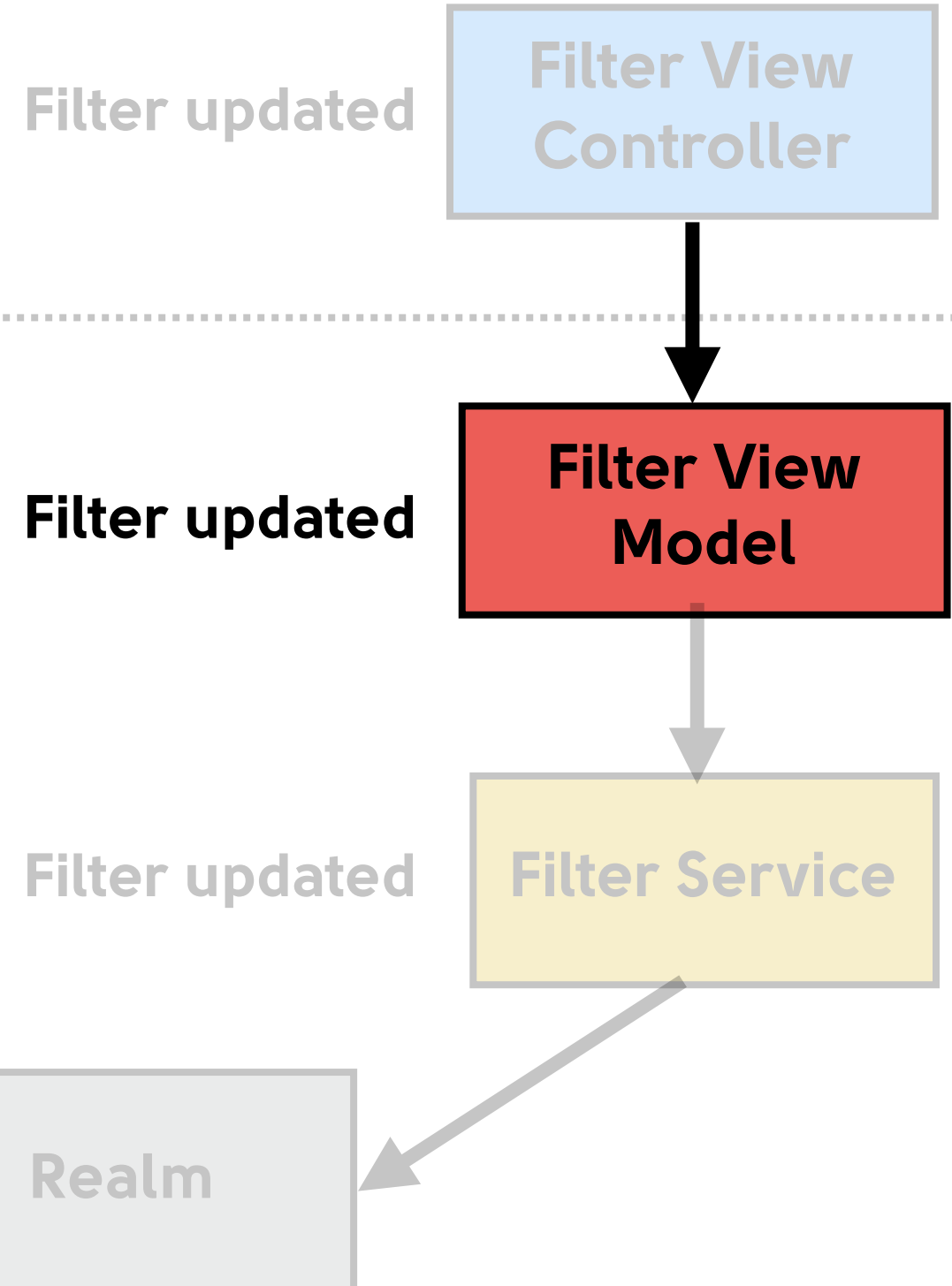


Подписка на storage

Feed



Filter



Подписка на storage

Feed

Filter

Sections updated

Feed View
Controller

Filter updated

Filter View
Controller

Filter updated

Presentations updated

Feed View
Model

Filter updated

Filter View
Model

Combined

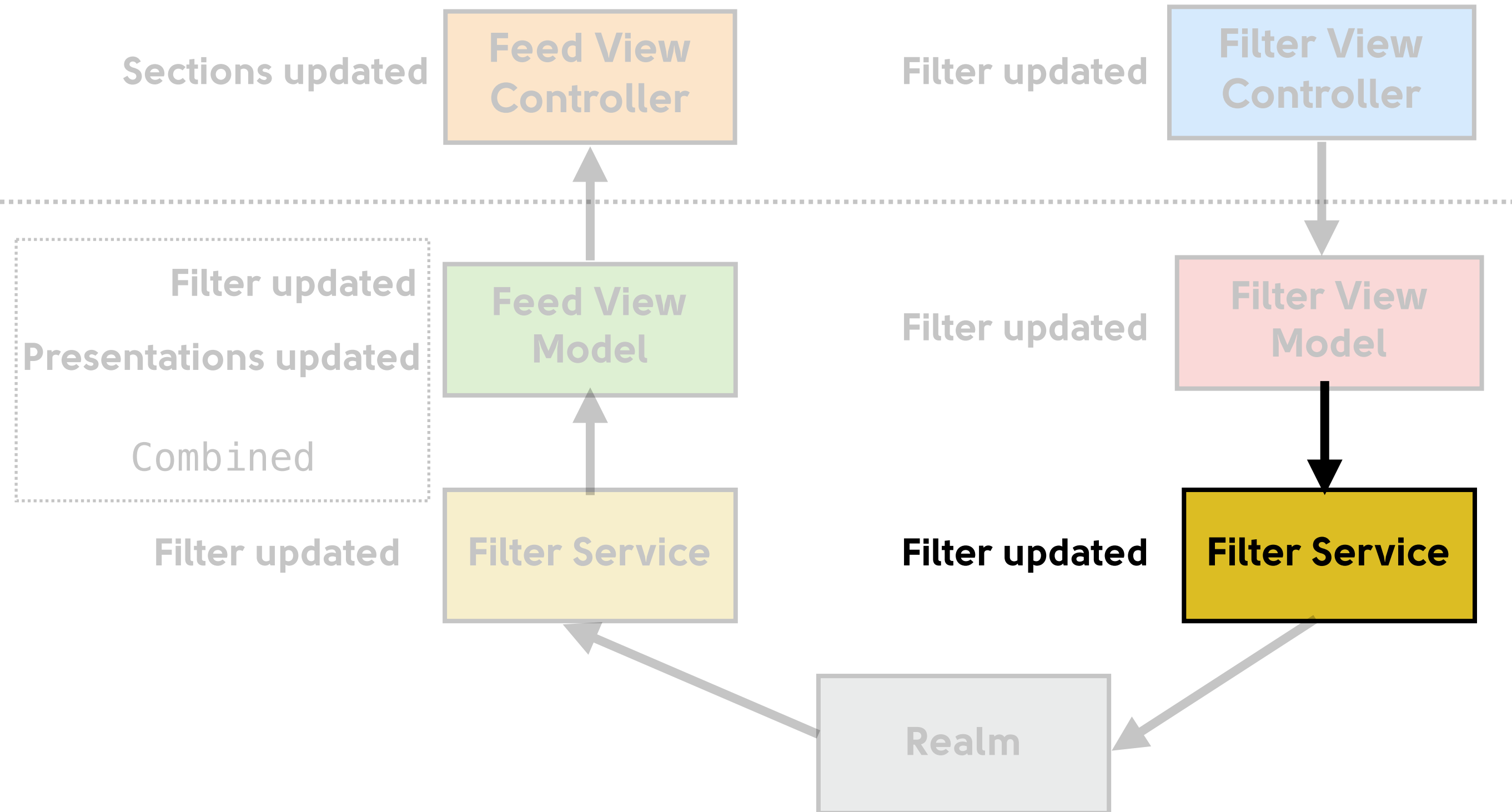
Filter updated

Filter Service

Filter updated

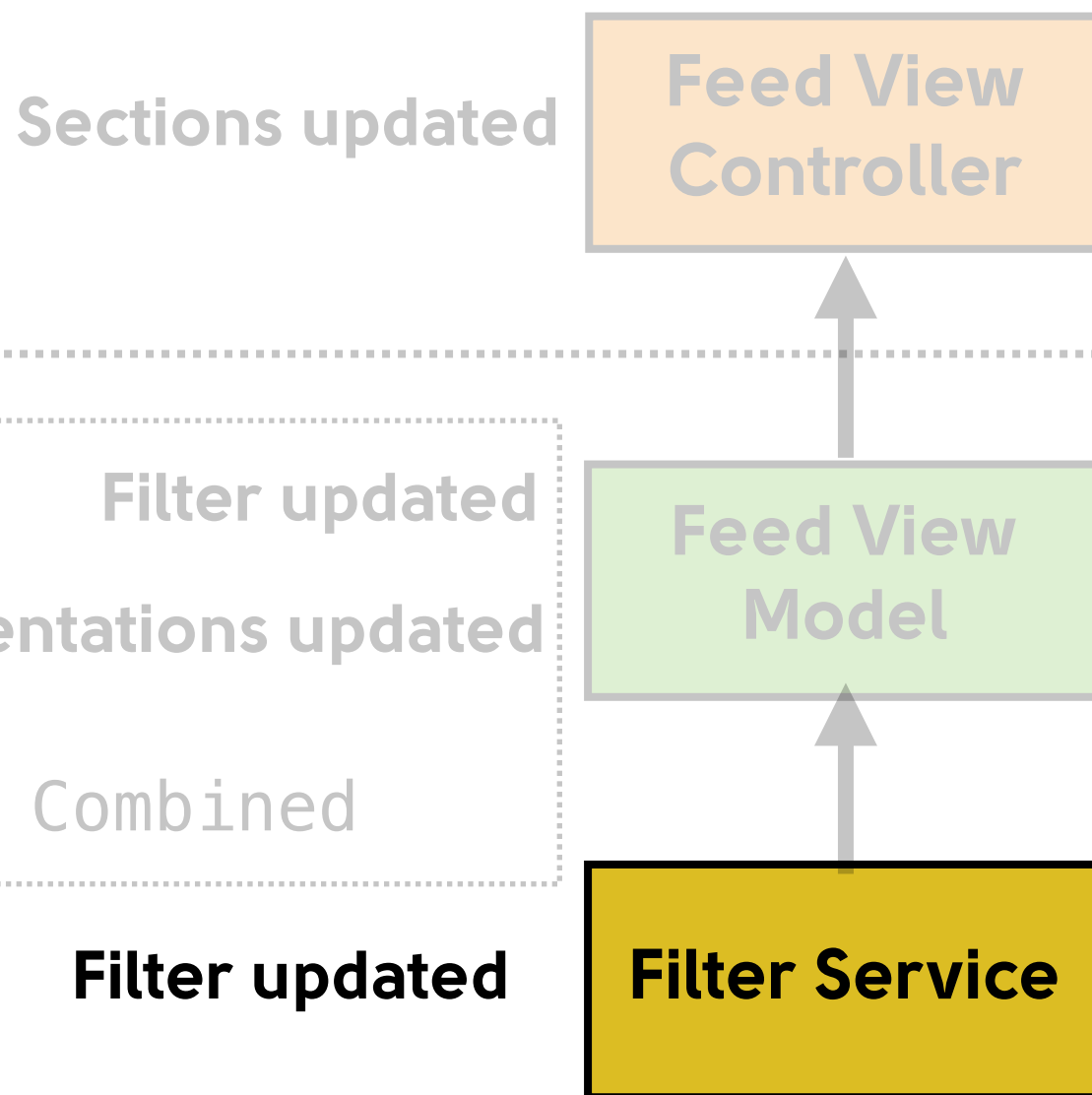
Filter Service

Realm

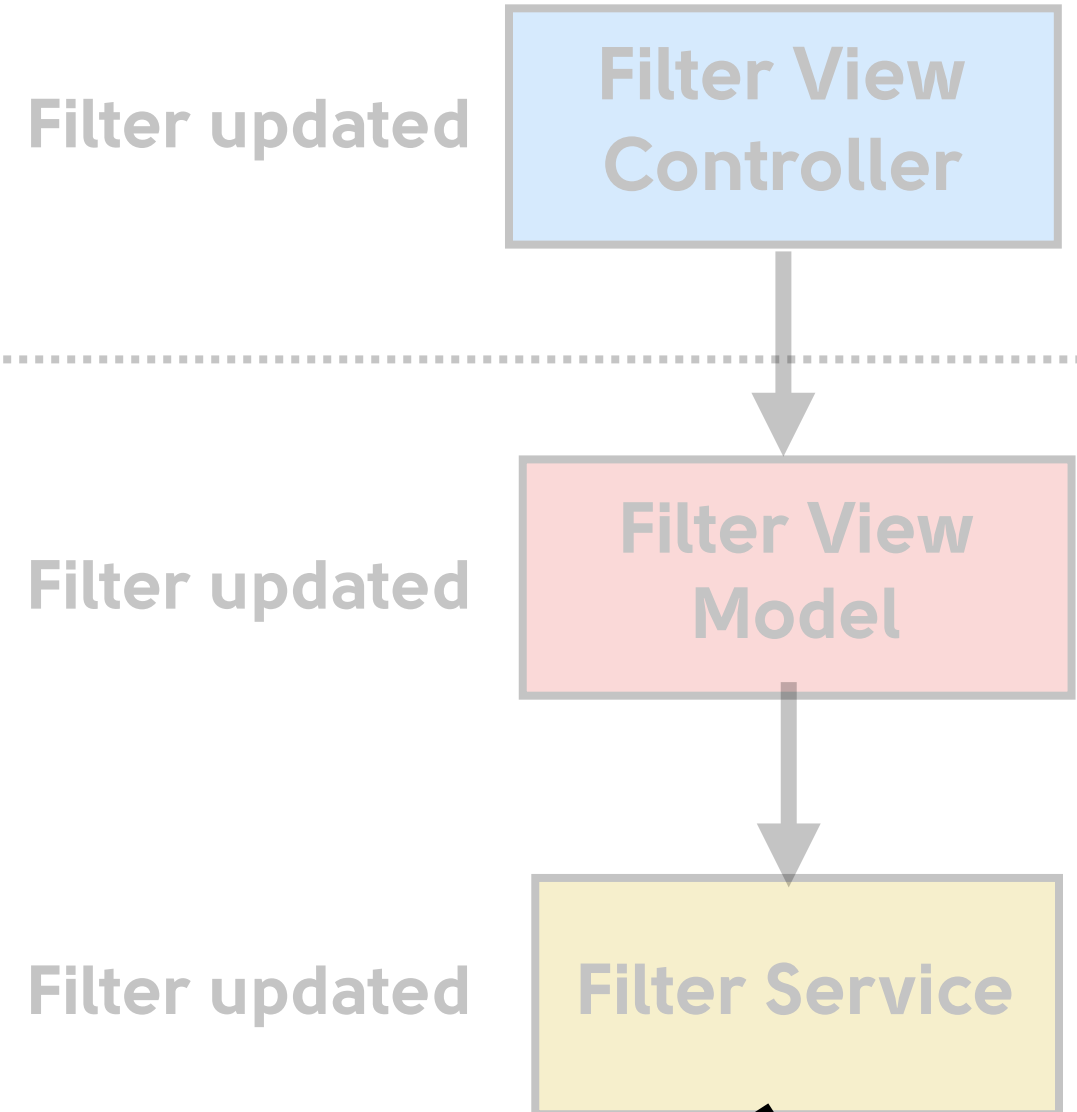


Подписка на storage

Feed



Filter



Подписка на storage

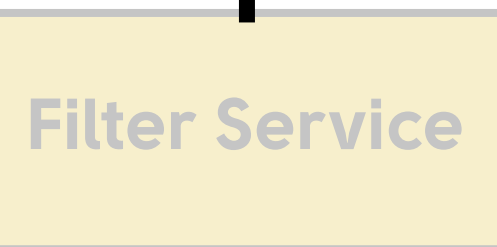
Feed



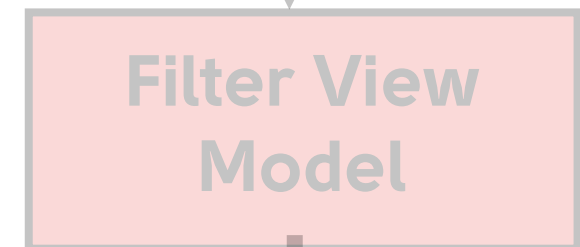
Filter



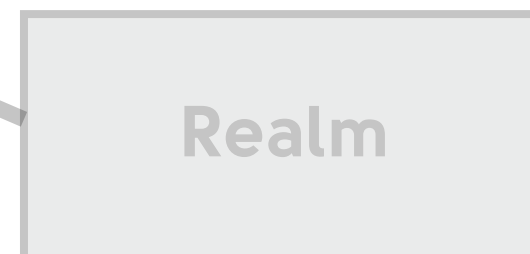
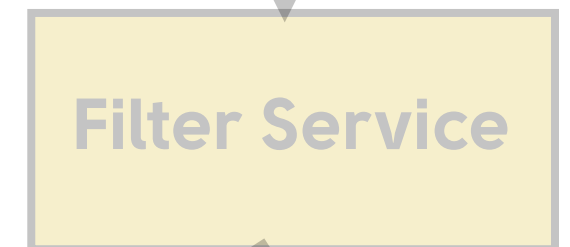
Filter updated



Filter updated



Filter updated



Подписка на storage

Feed

Filter

Sections updated

Feed View
Controller

Filter updated

Filter View
Controller

Filter updated

Presentations updated

Feed View
Model

Filter updated

Filter View
Model

Combined

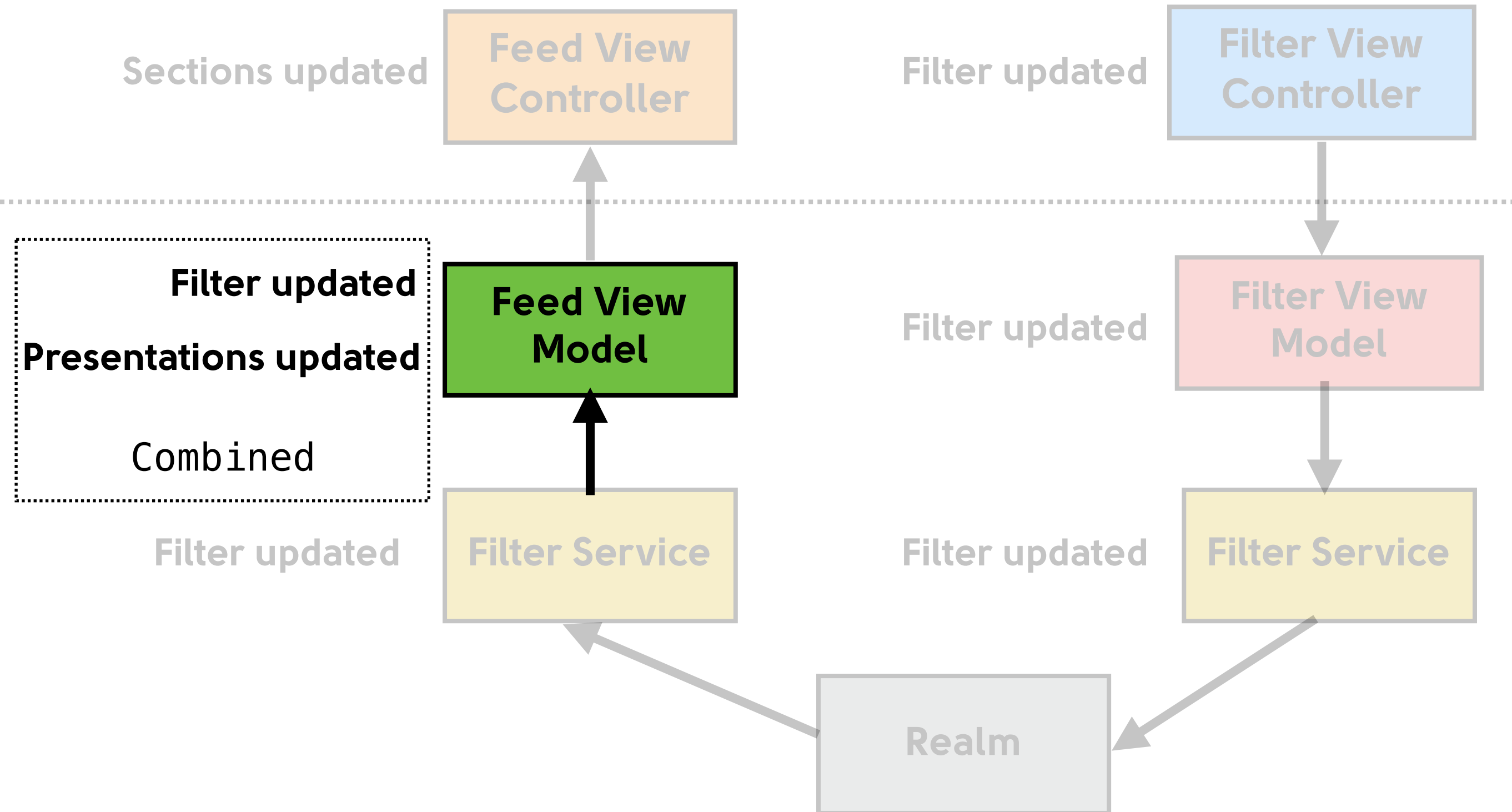
Filter updated

Filter Service

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

Feed

Sections updated

**Feed View
Controller**

Filter updated

Presentations updated

Combined

Filter updated

**Feed View
Model**

Filter Service

Filter updated

**Filter View
Controller**

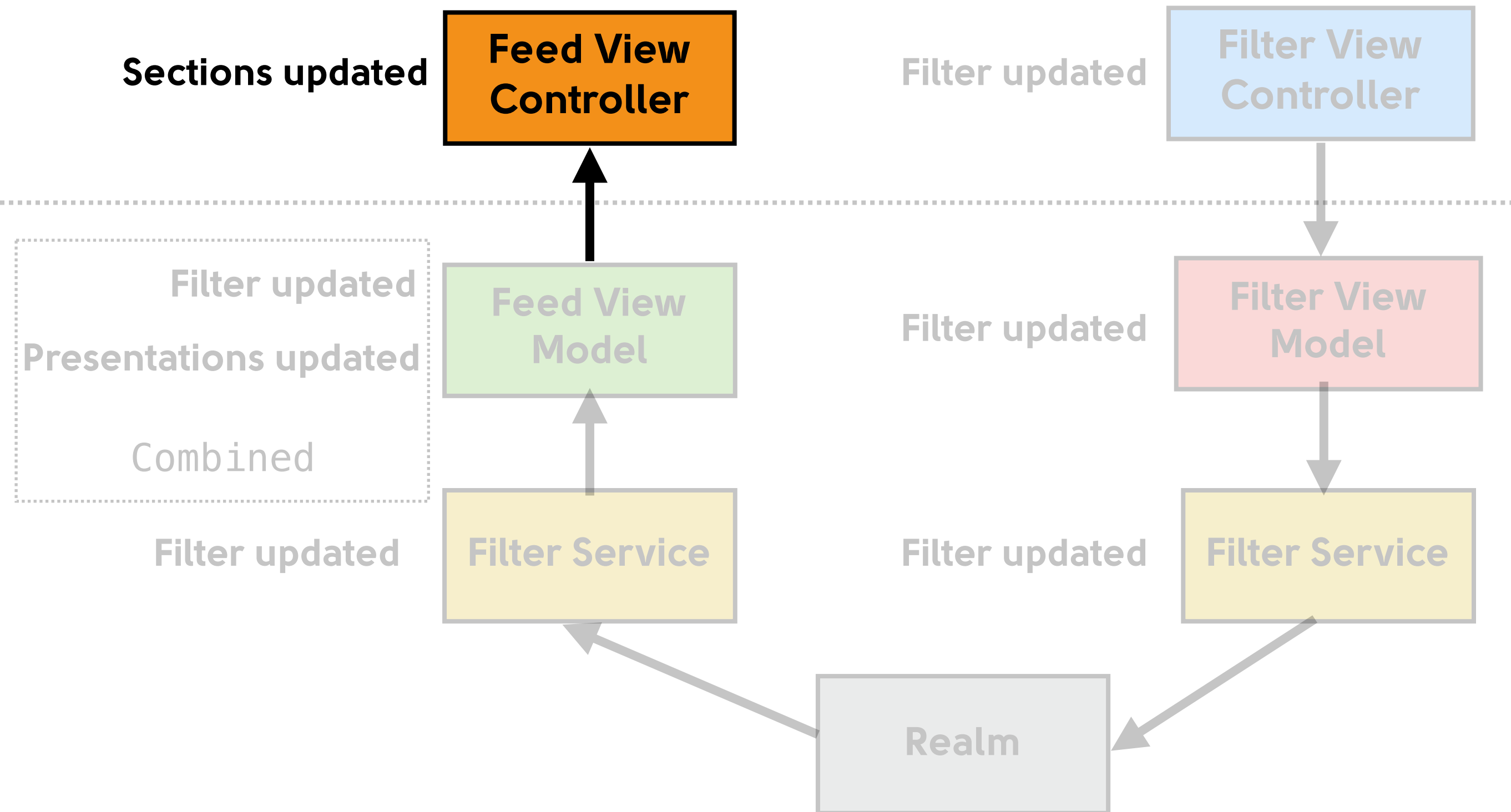
Filter updated

**Filter View
Model**

Filter updated

Filter Service

Realm



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 входа
- **Относительно сложная отладка**

Спасибо!

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