Яндекс 360

Reactive Swift Concurrency

Как мы запускали реактивный двигатель и что из этого вышло

Спикер:
Башир Арсланалиев,
iOS-разработчик

Эксперт:
Владимир Бородько,
Team Lead

Яндекс 360

Дисклеймер

Яндекс 360

Про Башира

- Работает iOS-разработчиком 3+ года, 2 из которых — в Яндекс Мессенджере
- Пробует новые инструменты в новых фичах
- Много рофлит (так себе)

Яндекс 360

Про Владимира

- Yandex Messenger Team Lead (iOS)
- Прародитель реактивного двигателя на Swift Concurrency
- ~~Заставляет~~ Вдохновляет работать Башира

Яндекс 360

Про Яндекс Мессенджер

- Делаем корпоративный мессенджер
- Поставляем SDK для других команд

Яндекс 360

Зачем нужен новый реактивный двигатель

Яндекс 360

Текущий инструментарий

Делегаты

1

Мониторы

2

Grand Central Dispatch

3

Яндекс 360

Текущий инструментарий

Делегаты

1. Weak Typing
2. Ломают Single Source of Truth
3. Нет backpressure-механизма, бесконтрольный вызов методов

Яндекс 360

Текущий инструментарий

Мониторы

1. Нетредсейфные
2. Сложноразвиваемые
3. Плохо стакаются со Swift Concurrency

Виктор Брыксин,
Nuclear Reactor Monitoring

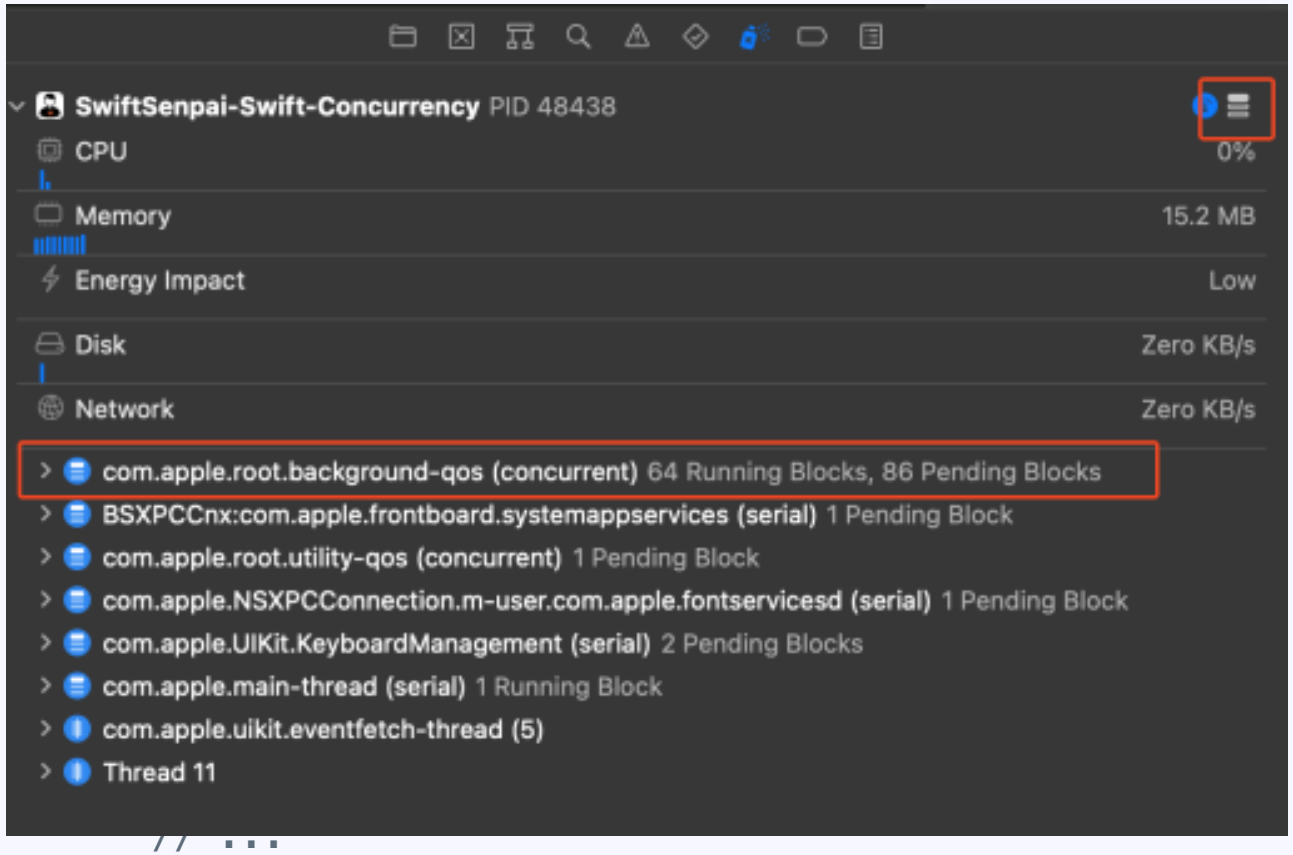
Яндекс 360

Текущий инструментарий

Grand Central Dispatch

1. Callback Hell
2. Сложно поддерживать тредсейфность сущностей
3. Thread Explosion

```
final class SomeService {
```




```
        // ...
    }
}
```

Яндекс 360

Чем фиксить

Rx/Combine	Развить мониторы	Swift Concurrency
1	2	3

Яндекс 360

Чем фиксить

RxSwift и Combine: почему не используем

- RxSwift
  - Тяжёлая third-party библиотека
  - Пространство для ошибок при работе с холодными подписками
- Combine:
  - Течёт память
  - Subscribe(on:) и cancel() на Concurrent Queue дают гонки, и подписка может отработать лишний раз

Яндекс 360

Чем фиксить

Подробнее о том, почему
в проекте не было Rx

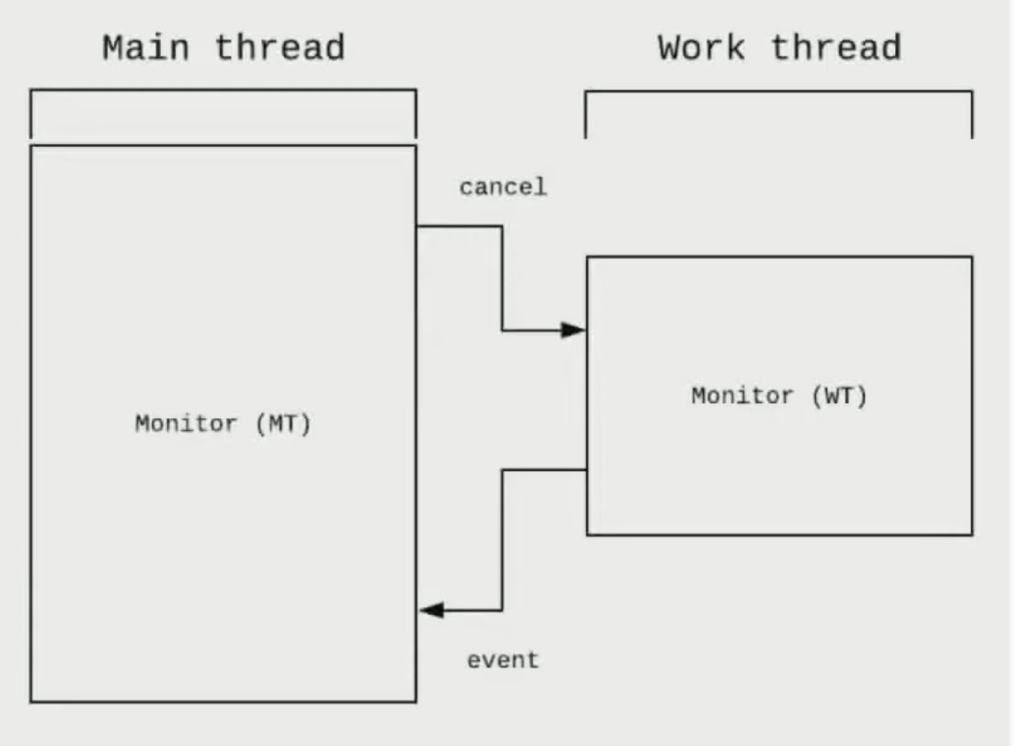
Виктор Брыксин,
Nuclear Reactor Monitoring

Яндекс 360

Чем фиксить

Мониторы: почему решили не развивать

1. Тредсейфность обеспечивается моделью A -> B -> A
2. Swift Concurrency не позволяет контролировать поток
3. Есть риск разломать всю логику на мониторах по приложению



Яндекс 360

Чем фиксить

Голый Swift Concurrency: классно, чего не хватает

1. Порядок выполнения не гарантирован
2. Проблемный AsyncSequence, нет работающей связи one-to-many
3. Нет реактивных инструментов

Яндекс 360

YaMulticast

Последовательная обработка ивентов

1

Мультикастинг от AsyncSequence на *infinite*-подписчиков

2

Всё на Swift Concurrency, а следовательно, полностью совместимо

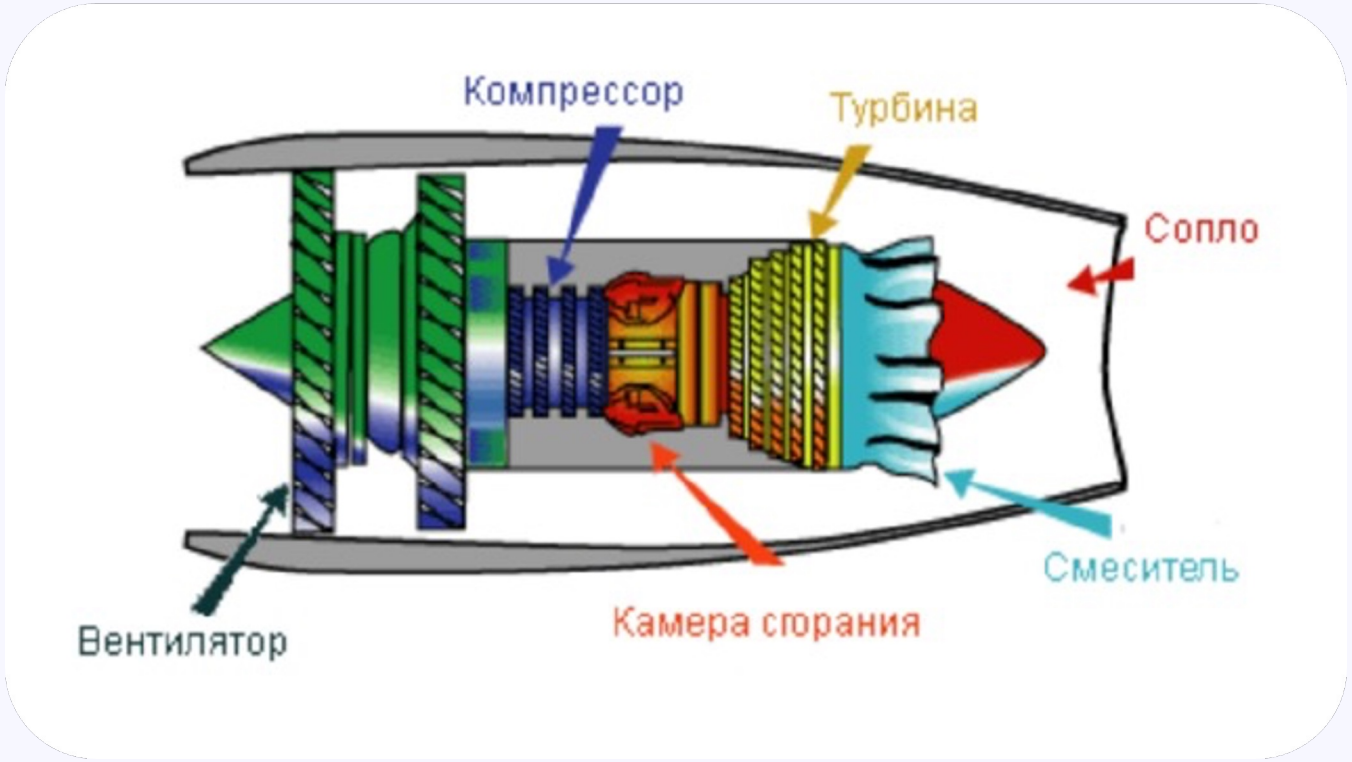
3

Яндекс 360

Как завести реактивный двигатель

Яндекс 360

Физика, восьмой класс



Яндекс 360

Основные сущности

1. Multicast<T> (aka Observable)
2. Sink<T> (events input)
3. Node<T> (aka PublishRelay)
4. Duty.Keeper (aka DisposeBag)
5. Старт транзакции, подписка и конец транзакции

```
final class ViewController: UIViewController {
  var buttonTapped: Multicast<Void> {
    buttonTappedNode.multicast
  }                                                          1

  private let buttonTappedNode = Node<Void>.warm()           3
  private let viewModel = ViewModel()
  private let dutyKeeper = Duty.Keeper()                     4

  private lazy var button: UIButton = {
    let button = UIButton()
    button.addAction(
      UIAction(handler: { [weak buttonTappedNode] in buttonTappedNode?.fire() }),
      for: .touchUpInside
    )
    return button
  }()

  override func viewDidLoad() {
    super.viewDidLoad()
    var duty = dutyKeeper.start()                            5
    duty.bind(buttonTapped, to: viewModel.printTheText)
    dutyKeeper.finish(duty: &duty)
  }
}
```

```
final class ViewModel {
  var printTheText: Sink<Void> {
    Sink { print("HelloWorld button tapped") }
  }                                                          2
}
```

Яндекс 360

Основные сущности

1. Multicast<T> (aka Observable)
2. Sink<T> (events input)
3. Node<T> (aka PublishRelay)
4. Duty.Keeper (aka DisposeBag)
5. Старт транзакции, подписка и конец транзакции

```
final class ViewController: UIViewController {
  var buttonTapped: Multicast<Void> {
    buttonTappedNode.multicast
  }                                                          1

  private let buttonTappedNode = Node<Void>.warm()           3
  private let viewModel = ViewModel()
  private let dutyKeeper = Duty.Keeper()                     4

  private lazy var button: UIButton = {
    let button = UIButton()
    button.addAction(
      UIAction(handler: { [weak buttonTappedNode] in buttonTappedNode?.fire() }),
      for: .touchUpInside
    )
    return button
  }()

  override func viewDidLoad() {
    super.viewDidLoad()
    var duty = dutyKeeper.start()                            5
    duty.bind(buttonTapped, to: viewModel.printTheText)
    dutyKeeper.finish(duty: &duty)
  }
}
```

```
final class ViewModel {
  var printTheText: Sink<Void> {
    Sink { print("HelloWorld button tapped") }
  }                                                          2
}
```

Яндекс 360

Основные сущности

Multicast & Sink

```
var buttonTapped: Multicast<Void> {
  buttonTappedNode.multicast
}

var printTheText: Sink<Void> {
  Sink { print("HelloWorld button tapped") }
}
```

Яндекс 360

Основные сущности

Multicast & Sink

1. Почти нет публичного интерфейса
2. По отдельности совсем не используются

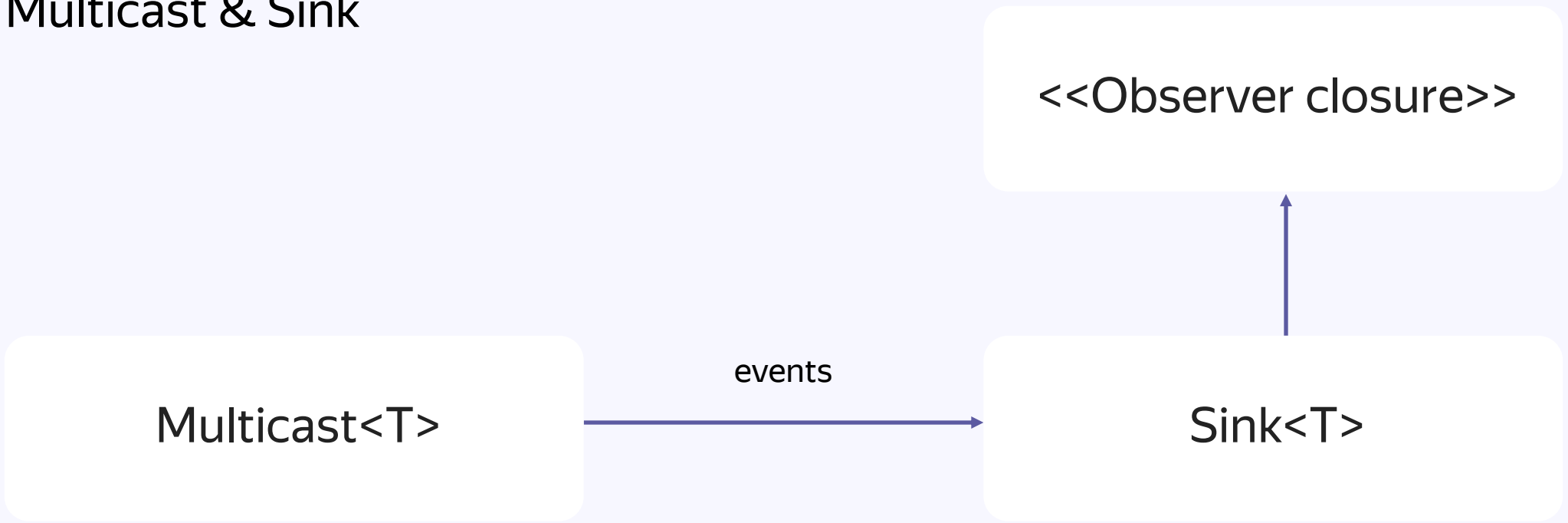
```
public final class Multicast<Value: Sendable>: Sendable { }
```

```
public final class Sink<Value: Sendable>: Sendable {
  public init(priority: TaskPriority = .userInitiated, send: @escaping Send)

  public typealias Send = @Sendable (Value) async -> Void
}
```

Яндекс 360

Основные сущности

Multicast & Sink



Яндекс 360

Основные сущности

1. Multicast<T> (aka Observable)
2. Sink<T> (events input)
3. Node<T> (aka PublishRelay)
4. Duty.Keeper (aka DisposeBag)
5. Старт транзакции, подписка и конец транзакции

```
final class ViewController: UIViewController {
  var buttonTapped: Multicast<Void> {
    buttonTappedNode.multicast
  }                                                          1

  private let buttonTappedNode = Node<Void>.warm()           3
  private let viewModel = ViewModel()
  private let dutyKeeper = Duty.Keeper()                     4

  private lazy var button: UIButton = {
    let button = UIButton()
    button.addAction(
      UIAction(handler: { [weak buttonTappedNode] in buttonTappedNode?.fire() }),
      for: .touchUpInside
    )
    return button
  }()

  override func viewDidLoad() {
    super.viewDidLoad()
    var duty = dutyKeeper.start()                            5
    duty.bind(buttonTapped, to: viewModel.printTheText)
    dutyKeeper.finish(duty: &duty)
  }
}
```

```
final class ViewModel {
  var printTheText: Sink<Void> {
    Sink { print("HelloWorld button tapped") }
  }                                                          2
}
```

Яндекс 360

Основные сущности

1. Multicast<T> (aka Observable)
2. Sink<T> (events input)
3. Node<T> (aka PublishRelay)
4. Duty.Keeper (aka DisposeBag)
5. Старт транзакции, подписка и конец транзакции

```
final class ViewController: UIViewController {
  var buttonTapped: Multicast<Void> {
    buttonTappedNode.multicast
  }

  private let buttonTappedNode = Node<Void>.warm()
  private let viewModel = ViewModel()
  private let dutyKeeper = Duty.Keeper()

  private lazy var button: UIButton = {
    let button = UIButton()
    button.addAction(
      UIAction(handler: { [weak buttonTappedNode] in buttonTappedNode?.fire() }),
      for: .touchUpInside
    )
    return button
  }()

  override func viewDidLoad() {
    super.viewDidLoad()
    var duty = dutyKeeper.start()
    duty.bind(buttonTapped, to: viewModel.printTheText)
    dutyKeeper.finish(duty: &duty)
  }
}
```

1

3

4

5

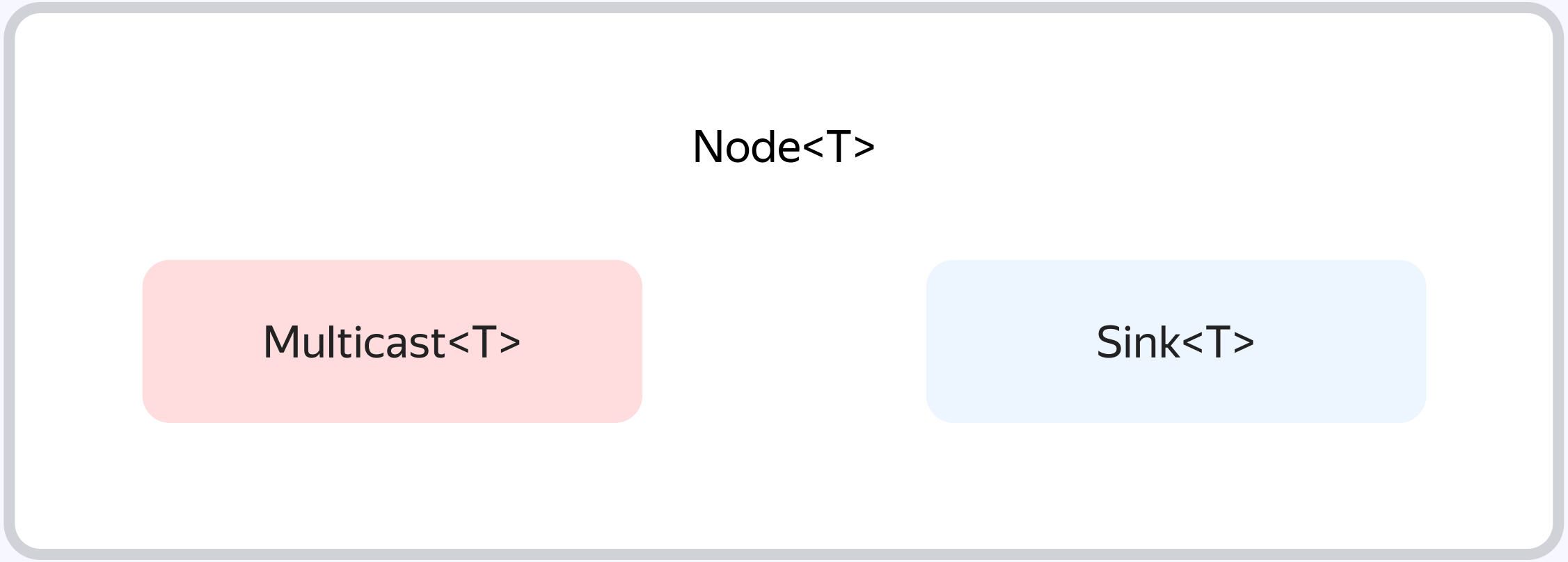
```
final class ViewModel {
  var printTheText: Sink<Void> {
    Sink { print("HelloWorld button tapped") }
  }
}
```

2

Яндекс 360

Основные сущности

Node



Яндекс 360

Основные сущности

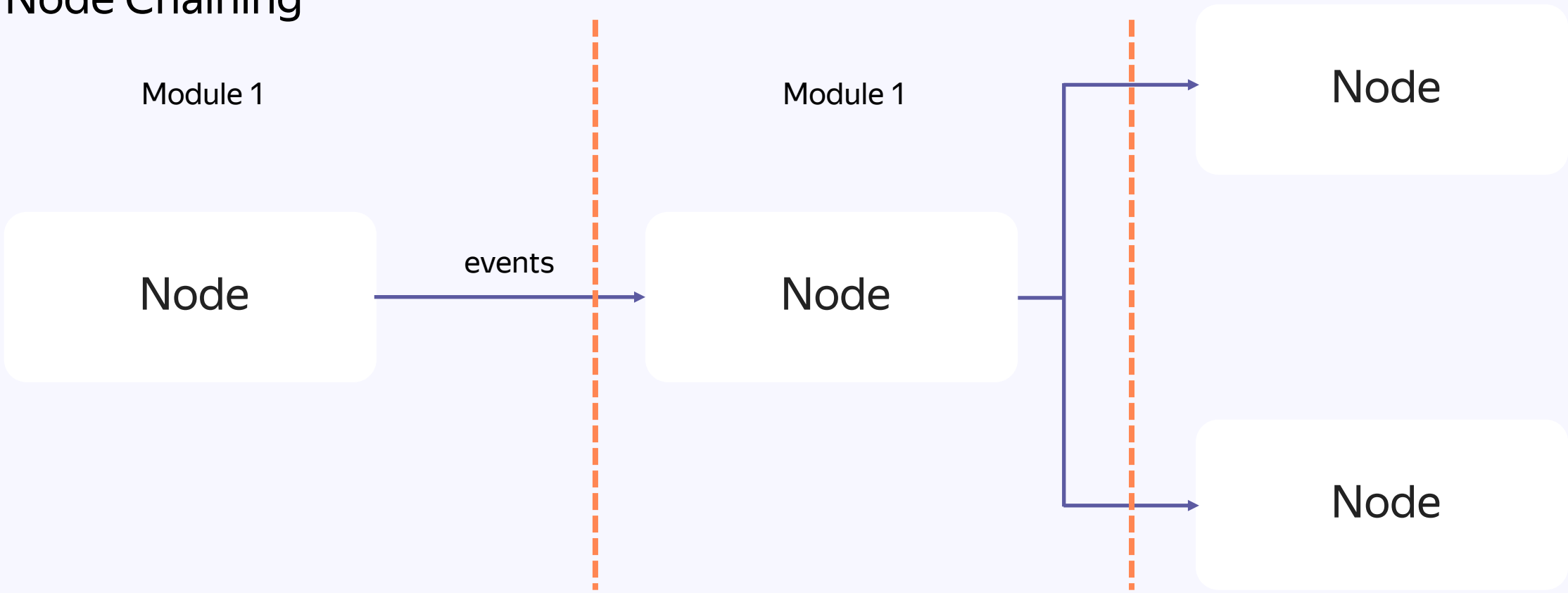
Node

```
public final class Node<Value: Sendable>: Sendable {
  public var sink: Sink<Value>
  public var multicast: Multicast<Value>

  public static func warm(
    reply: UInt32 = 0,
    priority: TaskPriority = .userInitiated,
    gage: Int? = 100
  ) -> Node<Value>

  public static func cold(
    duty: inout Duty,
    reply: UInt32 = 0,
    priority: TaskPriority = .userInitiated,
    gage: Int? = 100
  ) -> Node<Value>

  @Sendable public func send(_  value:  Value)
}
```

Яндекс 360

Основные сущности

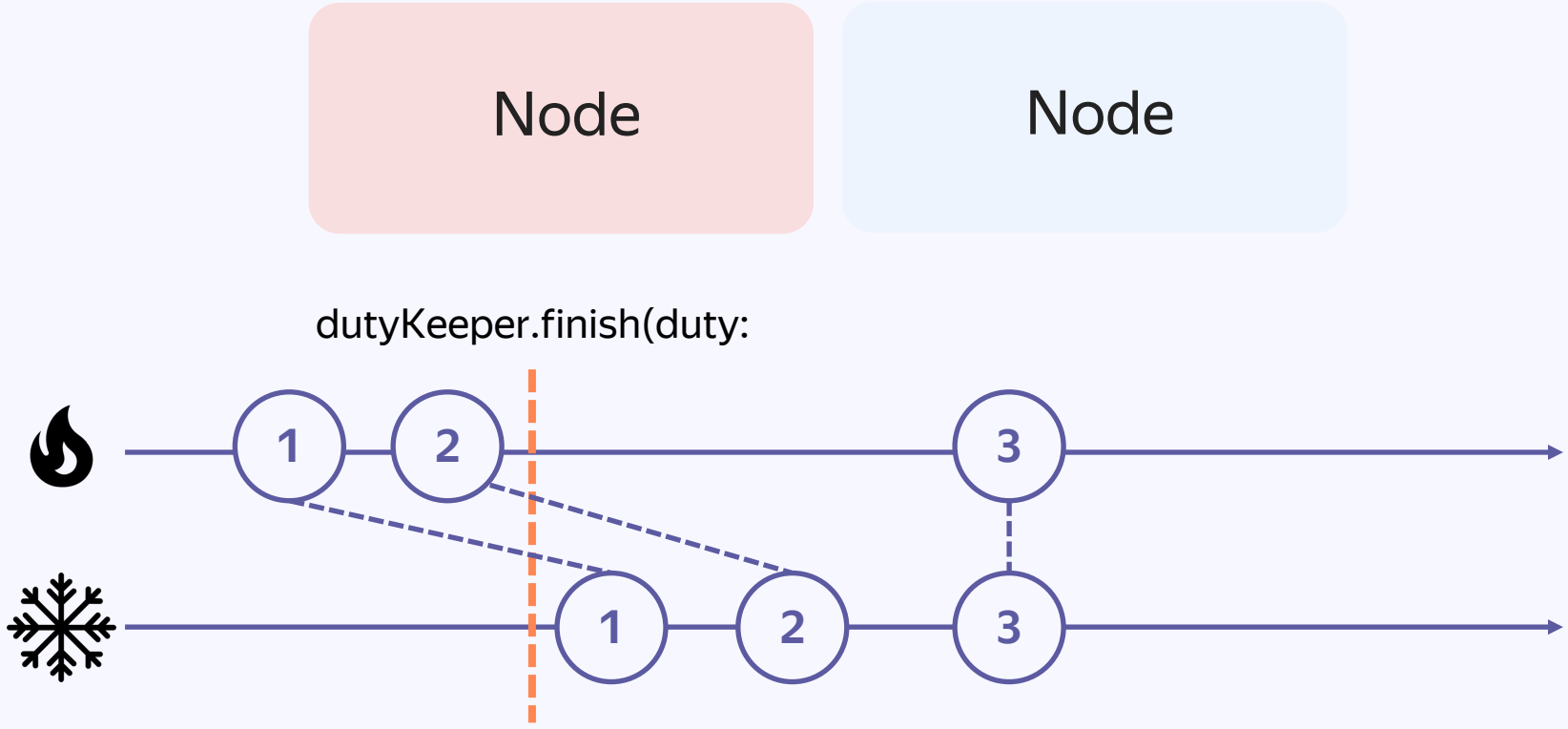
Node Chaining



Яндекс 360

Основные сущности

Hot & Cold Nodes



Яндекс 360

Основные сущности

1. Multicast<T> (aka Observable)
2. Sink<T> (events input)
3. Node<T> (aka PublishRelay)
4. Duty.Keeper (aka DisposeBag)
5. Старт транзакции, подписка и конец транзакции

```
final class ViewController: UIViewController {
  var buttonTapped: Multicast<Void> {
    buttonTappedNode.multicast
  }

  private let buttonTappedNode = Node<Void>.warm()
  private let viewModel = ViewModel()
  private let dutyKeeper = Duty.Keeper()

  private lazy var button: UIButton = {
    let button = UIButton()
    button.addAction(
      UIAction(handler: { [weak buttonTappedNode] in buttonTappedNode?.fire() }),
      for: .touchUpInside
    )
    return button
  }()

  override func viewDidLoad() {
    super.viewDidLoad()
    var duty = dutyKeeper.start()
    duty.bind(buttonTapped, to: viewModel.printTheText)
    dutyKeeper.finish(duty: &duty)
  }
}
```

1

3

4

5

```
final class ViewModel {
  var printTheText: Sink<Void> {
    Sink { print("HelloWorld button tapped") }
  }
}
```

2

Яндекс 360

Основные сущности

1. Multicast<T> (aka Observable)
2. Sink<T> (events input)
3. Node<T> (aka PublishRelay)
4. Duty.Keeper (aka DisposeBag)
5. Старт транзакции, подписка и конец транзакции

```
final class ViewController: UIViewController {
  var buttonTapped: Multicast<Void> {
    buttonTappedNode.multicast
  }

  private let buttonTappedNode = Node<Void>.warm()
  private let viewModel = ViewModel()
  private let dutyKeeper = Duty.Keeper()

  private lazy var button: UIButton = {
    let button = UIButton()
    button.addAction(
      UIAction(handler: { [weak buttonTappedNode] in buttonTappedNode?.fire() }),
      for: .touchUpInside
    )
    return button
  }()

  override func viewDidLoad() {
    super.viewDidLoad()
    var duty = dutyKeeper.start()
    duty.bind(buttonTapped, to: viewModel.printTheText)
    dutyKeeper.finish(duty: &duty)
  }
}
```

1

```
final class ViewModel {
  var printTheText: Sink<Void> {
    Sink { print("HelloWorld button tapped") }
  }
}
```

2

3

4

5

Яндекс 360

Основные сущности

Duty. Keeper & Duty

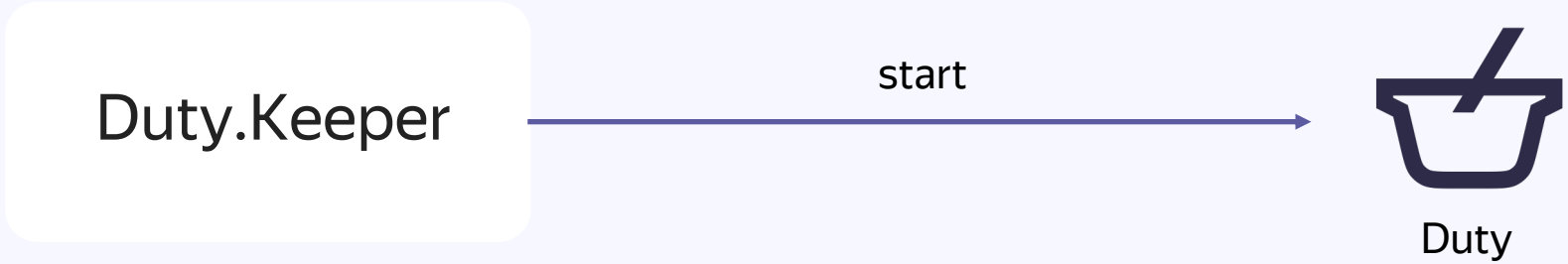
- Duty. Keeper
  - Хранит все запущенные подписки
- Duty
  - Хранит ещё не запущенные подписки в стеке

Яндекс 360

Основные сущности

Как работает подписка

```
var duty = dutyKeeper.start()
```

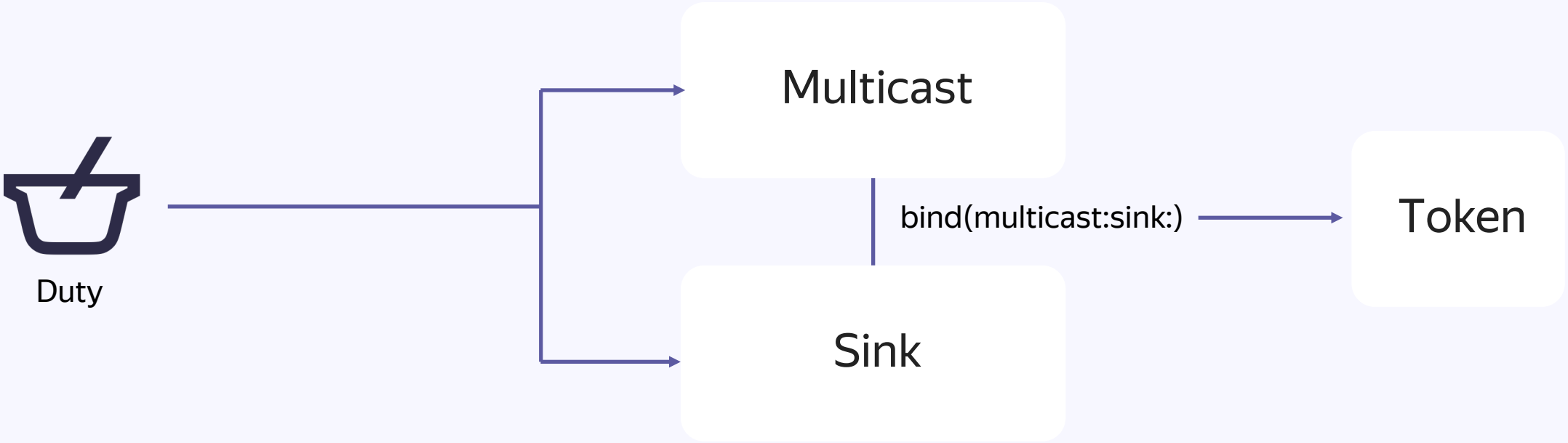


Яндекс 360

Основные сущности

Как работает подписка

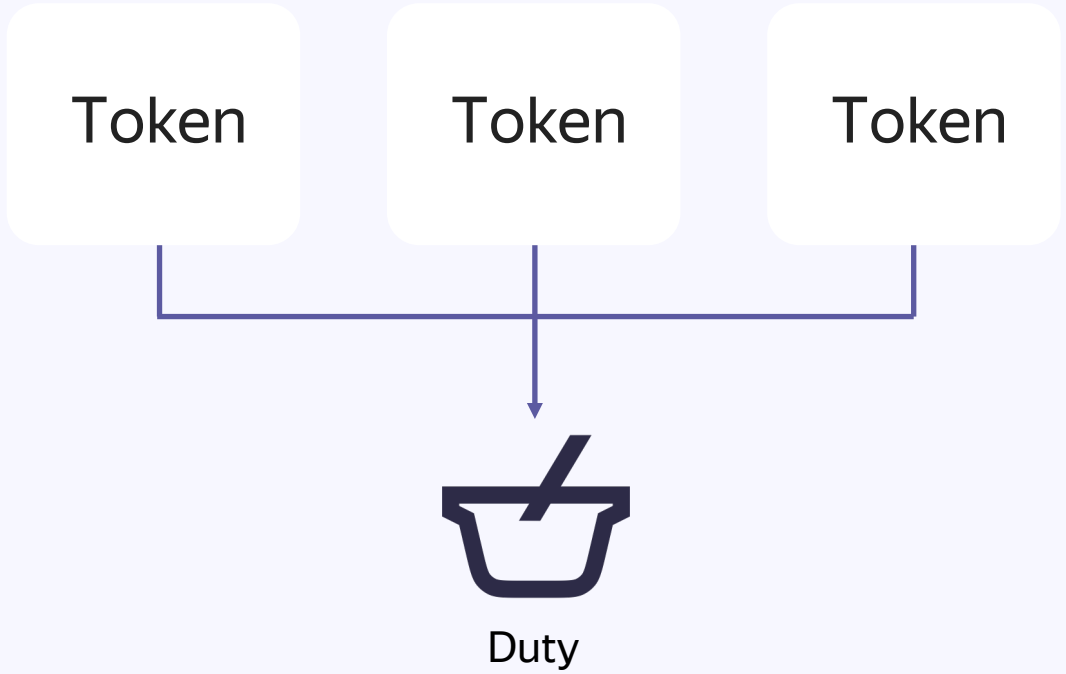
```
duty.bind(buttonTapped, to: viewModel.printTheText)
```



Яндекс 360

Основные сущности

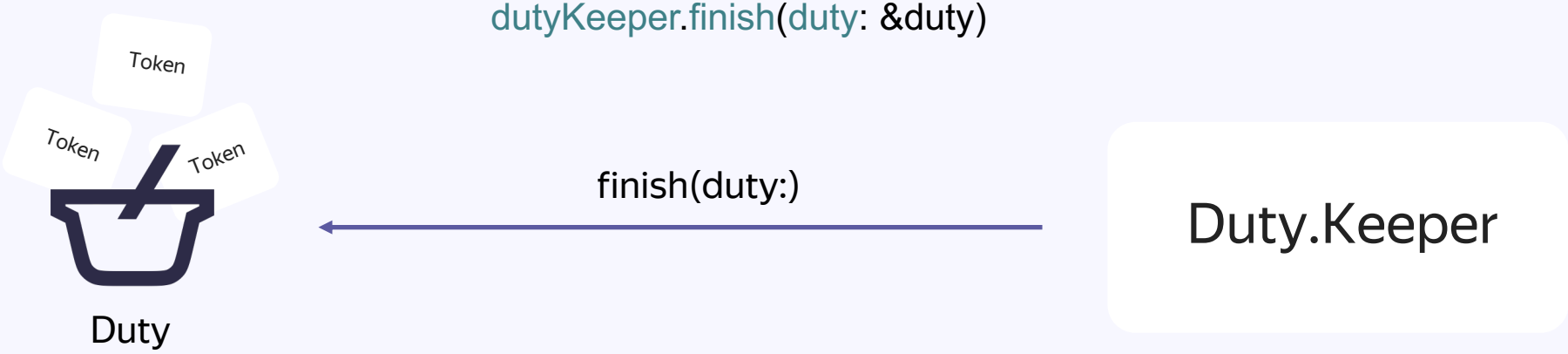
Как работает подписка



Яндекс 360

Основные сущности

Как работает подписка



Яндекс 360

Основные сущности

Стек в Duty

1. Ноды создаются через .cold интерфейс внутри init
2. Добавляются подписки
3. Где-то снаружи вызывается keeper.finish(duty:)
4. Активируется подписка для numbers
5. Активируется подписка для isActive
6. Прогреваются ноды: numbersNode и isActiveNode
7. Мы не продолбали ивенты 🎉

```
final class ViewModel {
  let isActiveNode: Node<Bool>
  let numbersNode: Node<Int>

  init(
    isActiveSink: Sink<Bool>,
    numbersSink: Sink<Int>,
    duty: inout Duty
  ) {
    isActiveNode = Node.cold(duty: &duty)
    numbersNode = Node.cold(duty: &duty)


    duty.bind(isActiveNode.multicast, to: isActiveSink)
    duty.bind(numbersNode.multicast, to: numbersSink)
  }
}
```

Яндекс 360

Основные сущности

1. Multicast<T> (aka Observable)
2. Sink<T> (events input)
3. Node<T> (aka PublishRelay)
4. Duty.Keeper (aka DisposeBag)
5. Старт транзакции, подписка и конец транзакции

```
final class ViewController: UIViewController {
  var buttonTapped: Multicast<Void> {
    buttonTappedNode.multicast
  }

  private let buttonTappedNode = Node<Void>.warm()
  private let viewModel = ViewModel()
  private let dutyKeeper = Duty.Keeper()

  private lazy var button: UIButton = {
    let button = UIButton()
    button.addAction(
      UIAction(handler: { [weak buttonTappedNode] in buttonTappedNode?.fire() }),
      for: .touchUpInside
    )
    return button
  }()

  override func viewDidLoad() {
    super.viewDidLoad()
    var duty = dutyKeeper.start()
    duty.bind(buttonTapped, to: viewModel.printTheText)
    dutyKeeper.finish(duty: &duty)
  }
}
```

1

```
final class ViewModel {
  var printTheText: Sink<Void> {
    Sink { print("HelloWorld button tapped") }
  }
}
```

2

3

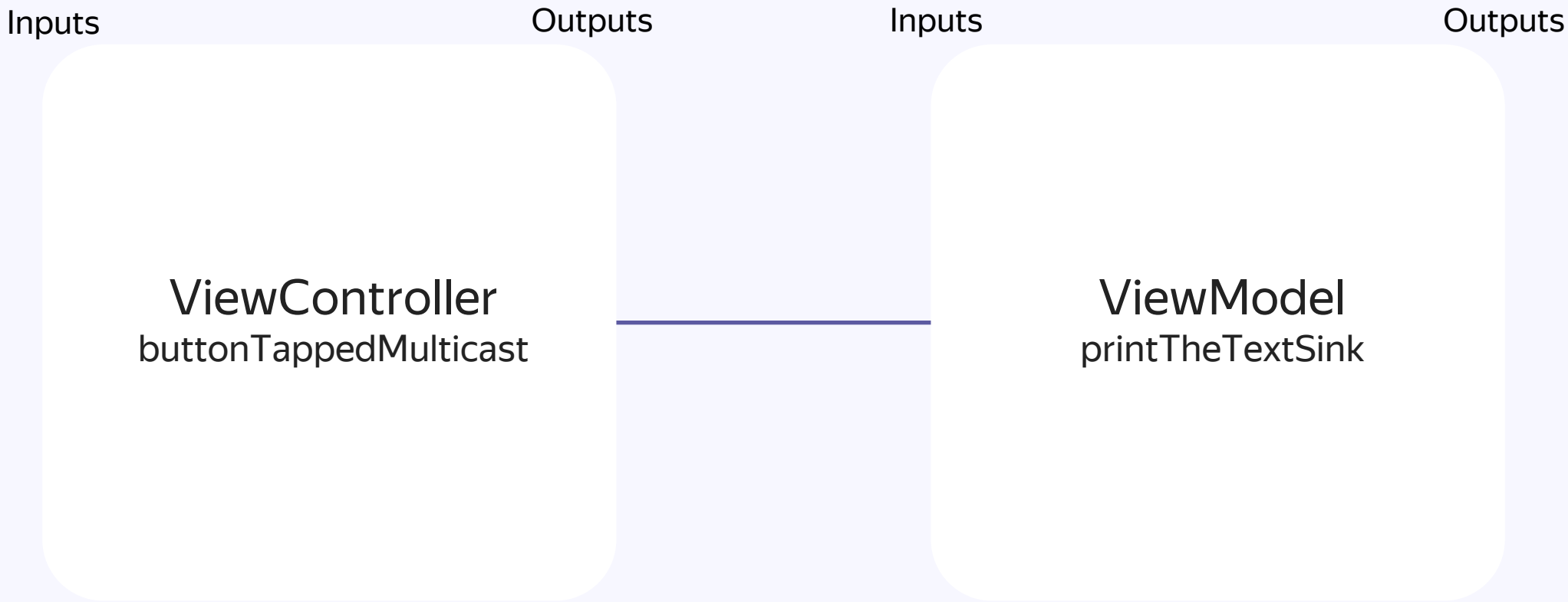
4

5

Яндекс 360

Основные сущности

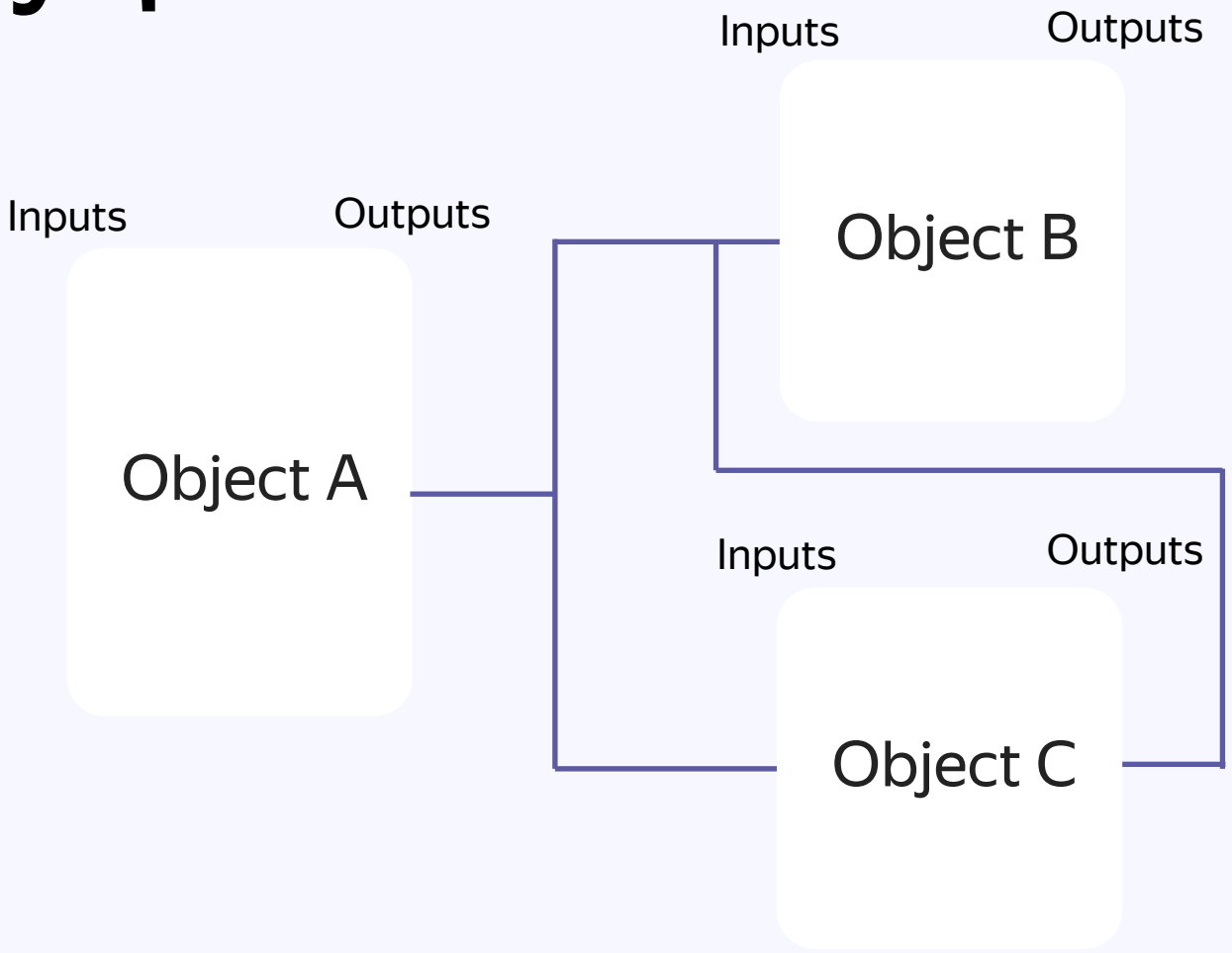
Схема



Яндекс 360

Основные сущности

Общая схема



Яндекс 360

Ограничения

Ивенты идут строго один за другим

```
Sink<String> { value in
  try? await Task.sleep(nanoseconds: 10 * nanosecsInSecond)
  print(value)
}

Sink<String> { value in
  Task {
    try? await Task.sleep(nanoseconds: 10 * nanosecsInSecond)
    print(value)
  }
}
```

Яндекс 360

Ограничения

Лучше всё делать через .cold

```
final class ViewModel {
  let isActiveNode: Node<Bool>
  let numbersNode: Node<Int>

  init(
    isActiveSink: Sink<Bool>,
    numbersSink: Sink<Int>,
    duty: inout Duty
  ) {
    isActiveNode = Node.cold(duty: &duty)
    numbersNode = Node.cold(duty: &duty)


    duty.bind(isActiveNode.multicast, to: isActiveSink)
    duty.bind(numbersNode.multicast, to: numbersSink)
  }
}
```

Яндекс 360

Ограничения

Нет временных подписок*

```
final class Foo {
  var dutyKeeper: Duty.Keeper? = Duty.Keeper()

  func makeTemporarySubscribtions() {
    guard var duty = dutyKeeper?.start() else { return }
    // make subscribtions
  }

  func unsubscribe() {
    dutyKeeper = nil
    dutyKeeper = Duty.Keeper()
  }
}
```

Яндекс 360

Запуск!

Яндекс 360

Яндекс 360

Яндекс 360

Яндекс 360

Яндекс 360

А что с операторами?

Яндекс 360

Они есть!
Но есть нюанс

Яндекс 360

Swift Concurrency (пока) не любит порядок

Custom Actor Executors

Custom Task Executors

Яндекс 360

```
let node1 = Node<Int>.warm()
let node2 = Node<Int>.warm()
let sink = Sink<(Int, Int)> { print($0, $1) }
```

Яндекс 360

```
let cast1 = node1.cast()
  .prepend(0)

let cast2 = node2.cast()
  .map { value in
    try? await Task.sleep(nanoseconds: 100 * nanosecsInSecond)
    return value
  }
  .prepend(0)

Cast.combineLatest(cast1, cast2)
  .bind(duty: &duty, to: sink)

node2.send(10)
node1.send(100)
```

Яндекс 360

Что мы ожидали

```
0 0
0 10
100 10
```

```
node2.send(10)
node1.send(100)
```

Что получили

```
0 0
100 0
100 10
```

Яндекс 360

Все latest операторы получились мертворождёнными 🤷‍♂️

Яндекс 360

Яндекс 360

Что мы поняли

Писать реактивно на Swift Concurrency можно

1

Делать это с async операторами можно, но больно

2

Нам есть, над чем работать

3

Яндекс 360

Что должны были понять вы

Пробуйте новые инструменты

1

Делитесь своим опытом

2

Приходите в дискуссионную зону пообщаться

3

Яндекс 360

А чт

1. Работ
   фрейм
2. Ждём

Яндекс 360

Контакты и ссылки

@JoelColbeck

Башир Арсланалиев
iOS разработчик

Яндекс Мессенджер

@Vladimir_Borodko

Владимир Бородько
iOS Team Lead

Яндекс Мессенджер

Исходники