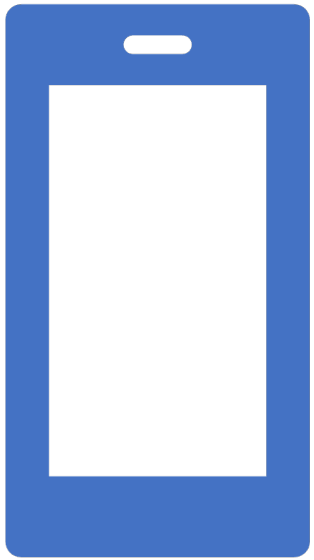
Combine

What is it?

The common situation in mobile apps

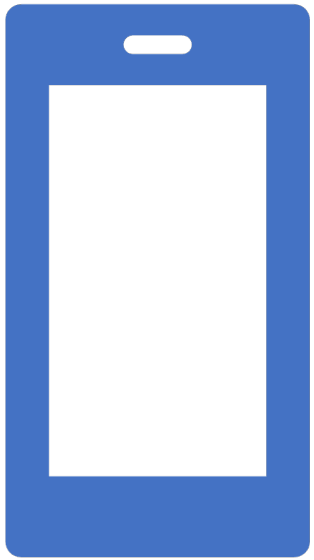
Publisher

Subscriber

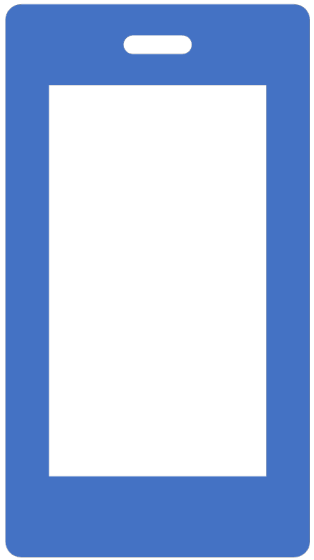


The common situation in mobile apps

The common situation in mobile apps

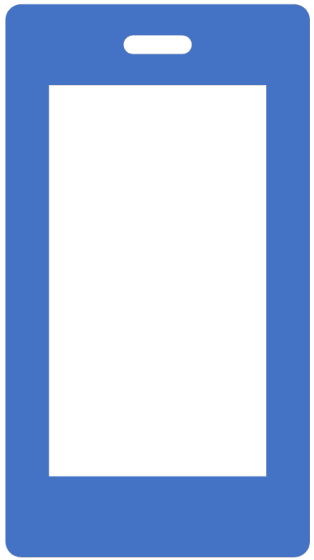


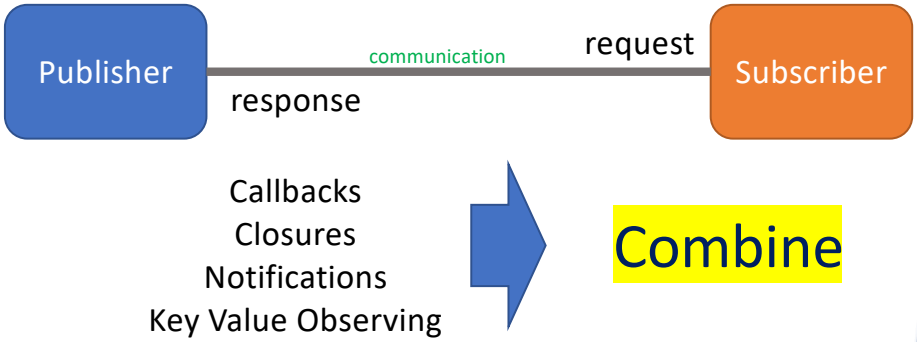
The common situation in mobile apps



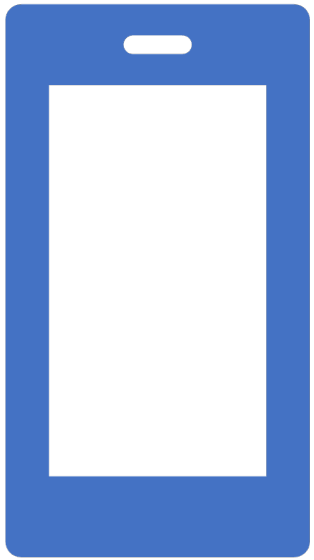
Callbacks
Closures
Notifications
Key Value Observing

The common situation in mobile apps





The common situation in mobile apps



Combine

Reactive framework

When we want to communicate objects in our program, we have a bunch of mechanisms in iOS development including delegates, closure, notifications, key value observing, and more.

Combine is a framework that unifies and adds to these existing mechanisms and techniquies.

Combine is Apple's take on a reactive programming framework that is declarative and focused on data streams and propagating changes to that data over time.

A object will have to broadcast changes somehow. This object just doesn't know or care who is listening.

Combine a unified, declarative API for processing values over time

Value Publisher conform Publisher protocol

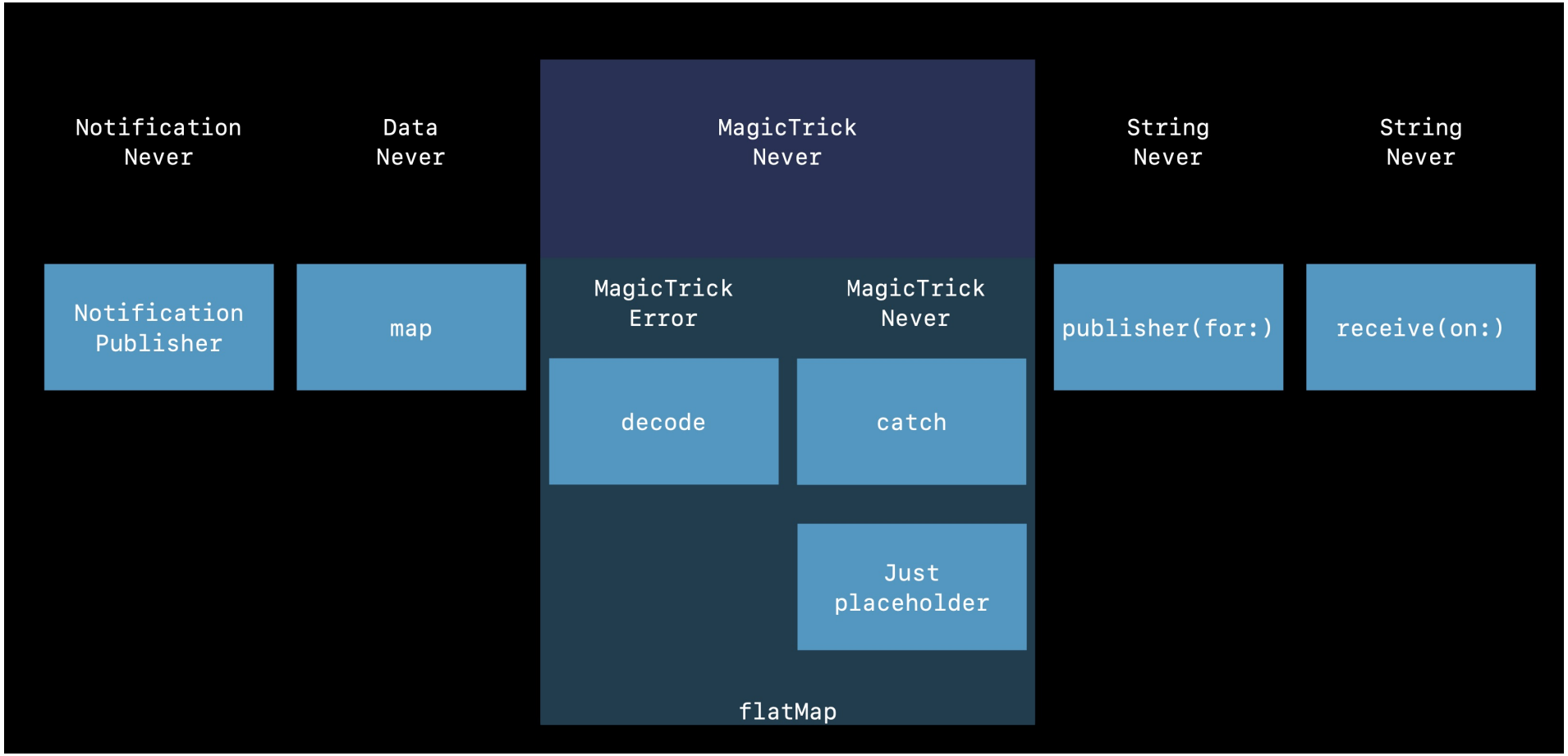
```
protocol Publisher {
    associatedtype Output
    associatedtype Failure: Error

    func subscribe<S: Subscriber>(_ subscriber: S)
        where S.Input == Output, S.Failure == Failure
}
```

```
// Using Publishers with Combine

let trickNamePublisher = NotificationCenter.default.publisher(for: .newTrickDownloaded)
    .map { notification in
        return notification.userInfo?["data"] as! Data
    }
    .flatMap { data in
        return Just(data)
            .decode(MagicTrick.self, JSONDecoder())
            .catch {
                return Just(MagicTrick.placeholder)
            }
    }
    .publisher(for: \.name)
```

String

Never

```swift
// Using Publishers with Combine

let trickNamePublisher = NotificationCenter.default.publisher(for: .newTrickDownloaded)
    .map { notification in
        return notification.userInfo?["data"] as! Data
    }
    .flatMap { data in
        return Just(data)
            .decode(MagicTrick.self, JSONDecoder())
            .catch {
                return Just(MagicTrick.placeholder)
            }
    }
    .publisher(for: \.name)
    .receive(on: RunLoop.main)
```


Notification
Never
Data
Never
MagicTrick
Never
String
Never
String
Never
Notification
Publisher
map
MagicTrick
Error
MagicTrick
Never
publisher(for:)
receive(on:)
decode
catch
Just
placeholder
flatMap

Publishers can be

- Recipe for an event stream
- Operators describe new publishers from existing
  Strongly typed values/errors over time
- Can be synchronous or asynchronous
- Can attach compatible Subscribers

Subscriber

```
protocol Subscriber {
    associatedtype Input
    associatedtype Failure: Error

    func receive(subscription: Subscription)
    func receive(_ value: Subscribers.Demand)
    func receive(completion: Subscribers.Completion<Failure>)
}
```

Subscribers' types

- Key Path Assignment
- Sinks
- **Subjects**
  - Can be subscriber
  - Can be subscriptor
- SwiftUI

Key Path Assignment

```
// Using Subscribers with Combine

let trickNamePublisher = ... // Publisher of <String, Never>

let canceller = trickNamePublisher.assign(to: \.someProperty, on: someObject)

// ...

canceller.cancel()
```

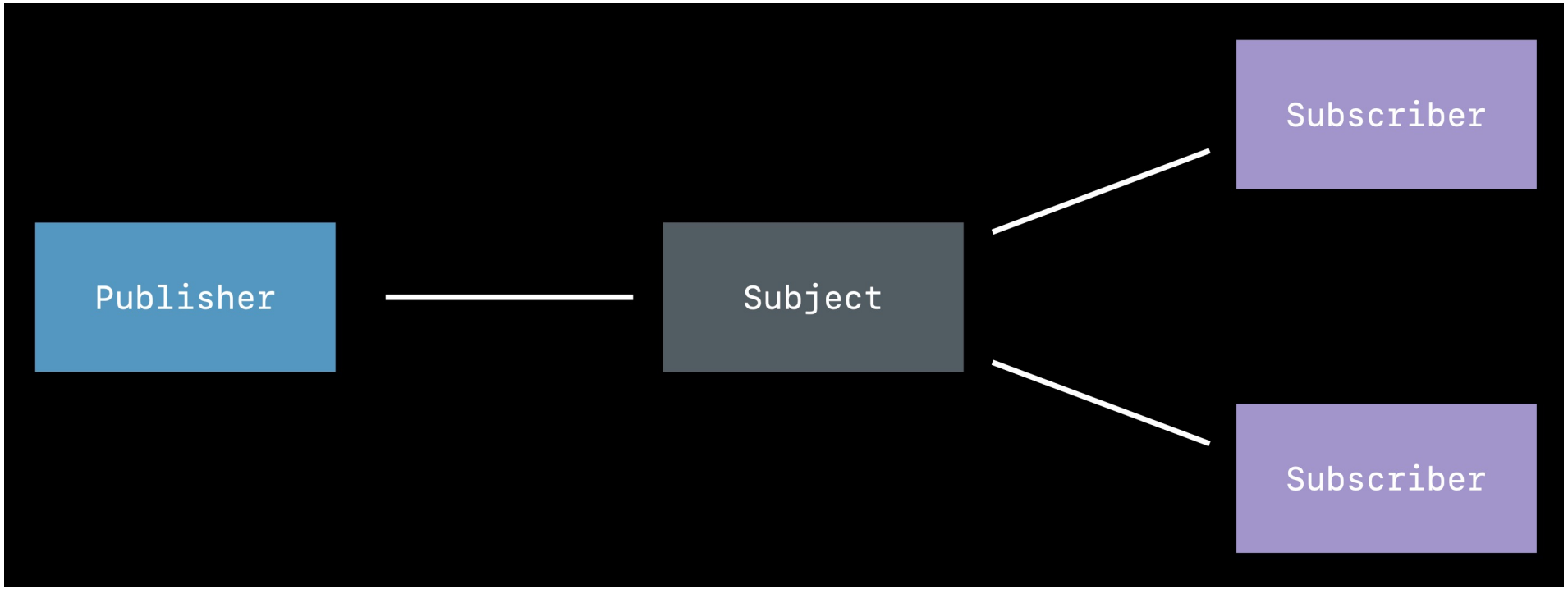
This is how we assign the value to a Subscriber.
Using the key path value and the object.
This create a cancellation object that it uses the Cancellable protocol that it has the method .cancel()

Subjects

Behave like both Publisher and Subscriber

Broadcast values to multiple subscribers

```
protocol Subject: Publisher, AnyObject {
    func send(_ value: Output)
    func send(completion: Subscribers.Completion<Failure>)
}
```


Publisher
Subject
Subscriber
Subscriber

```swift
// Using Subjects with Combine

let trickNamePublisher = ... // Publisher of <String, Never>

let magicWordsSubject = PassthroughSubject<String, Never>()

trickNamePublisher.subscribe(magicWordsSubject)

let canceller = magicWordsSubject.sink { value in
   // do something with the value
}

magicWordsSubject.send("Please")
```

Combine

`@Published` creates a publisher for value. It doesn't turn value into a publisher

```
@Published private var value = 0
```

The publisher created is `$value`

A publisher neds a subscribers `Sink` and `Assign`.

```
$value
  .sink {int in
    self.label.text = int.description
  }
```

Cancellable

AnyCancellable is a type that is used by subscribers so they can provide a link to a token that can be used to cancel a subscription without providing a link to the subscription itself. AnyCancellable is a concrete type that conforms to the Cancellable protocol.

```
cancellable
  = $value
  .sink {[weak self] int in
    self?.label.text = int.description
  }
```

Combine - Operators

Operators are inserted between a publisher and a subscriber. They modify the upstream publisher in some way.

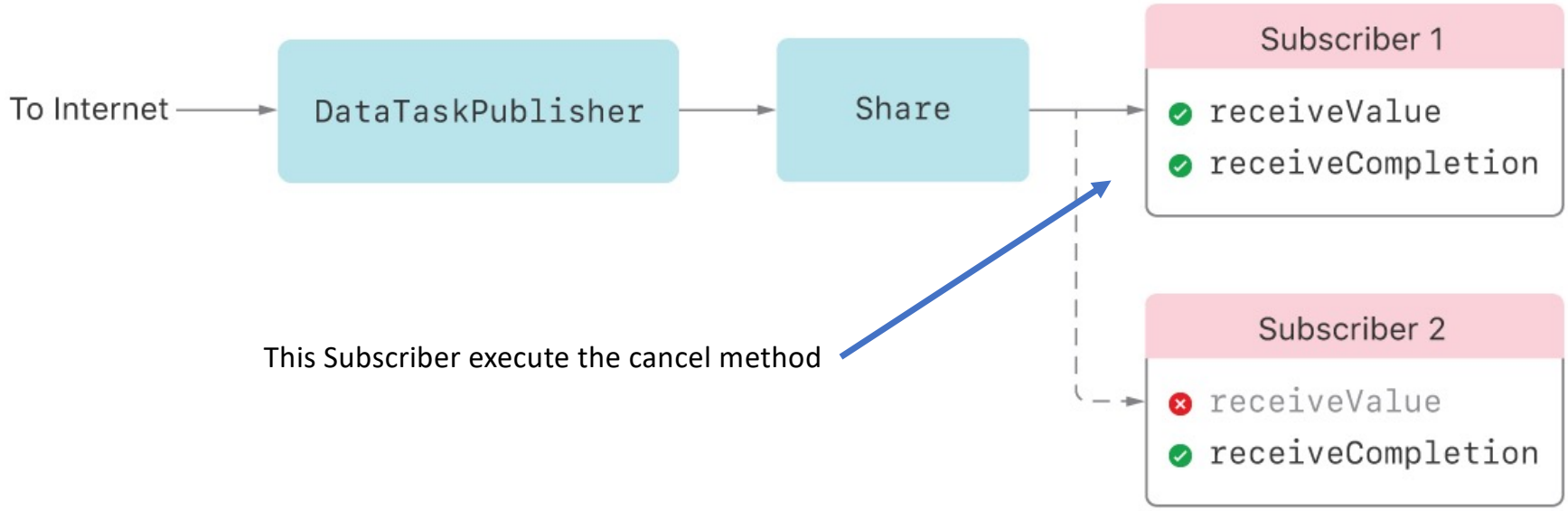
We going to see 5 operators:

- filter .- to do a filter
- compactMap .- lets us transform the elements of an array just like map() does, except once the transformation completes an extra step happens: all optionals get unwrapped, and any nil values get discarded.
- replaceNil .- allows us to replace nil with a default value.
- dropFirst .- to drop the first element from the list.
- map .- change the element to other

Prevent “Race Condition”

- Race condition.- A race condition is an undesirable situation that occurs when a program attempts to perform two or more operations at the same time on a shared resources. It results in undesirable outcomes.
- Case: A thread changes a state/property and at the same time other thread is using it.

- To Prevent this case in Combine we can use Connectable Publisher


To Internet
DataTaskPublisher
Share
Subscriber 1
receiveValue
receiveCompletion
Subscriber 2
receiveValue
receiveCompletion
This Subscriber execute the cancel method

We can create te ConnectablePublisher and send at the same time to all subscribers attached to the publisher.
To send values is needed call the method “connect()”