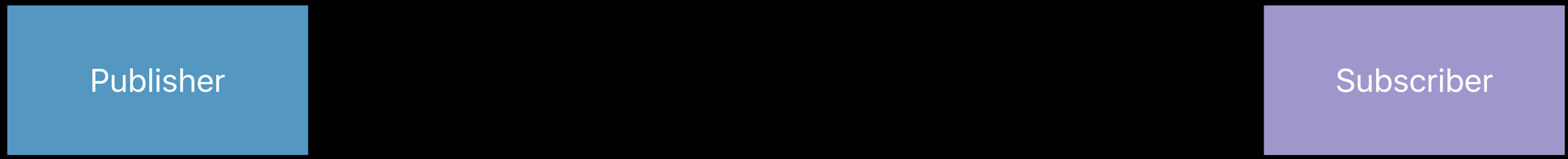
#WWDC19

Combine in Practice

Michael LeHew, Foundation
Ben D. Jones, Foundation


Publisher

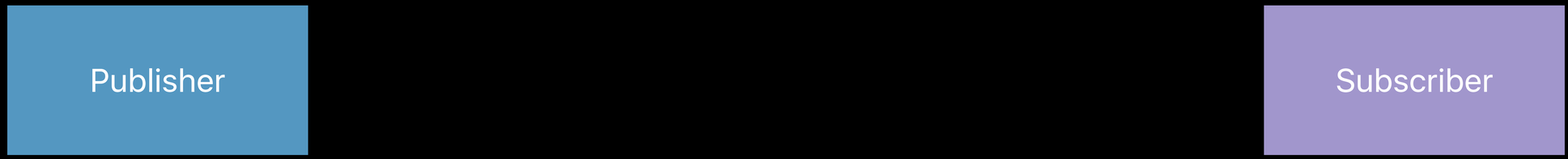

Publisher
Subscriber

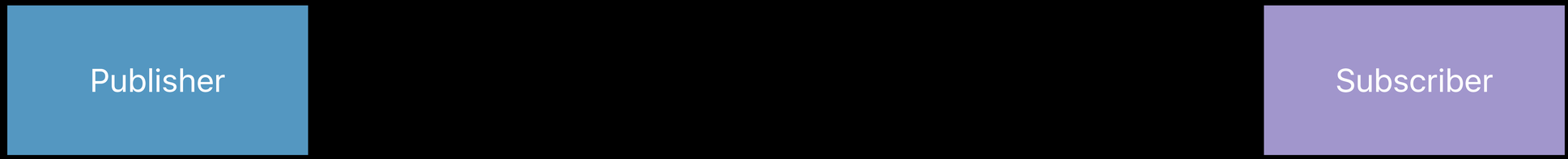

Publisher
Subscriber

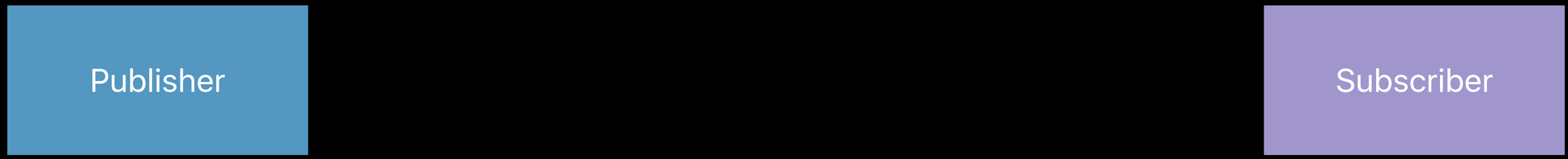

Publisher
request
Subscriber


Publisher
Subscriber


Publisher
Subscriber


Publisher
Subscriber


Publisher
Subscriber



Callbacks

Closures

Notifications

Key Value Observing

A unified, declarative API for processing values over time

Publisher

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

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

Publisher

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

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

Publisher

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

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

Publisher

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

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

Publisher

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

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

Publisher

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

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


Trick Name
Description


Trick Name
Description

```
// Using Publishers with Combine

let trickNamePublisher = NotificationCenter.default.publisher(for: .newTrickDownloaded)
```

```
// Using Publishers with Combine

let trickNamePublisher = NotificationCenter.default.publisher(for: .newTrickDownloaded)
```

```
some Publisher
```

```
// Using Publishers with Combine

let trickNamePublisher = NotificationCenter.default.publisher(for: .newTrickDownloaded)

    Notification      // Output

    Never             // Failure
```

```
// Using Publishers with Combine

let trickNamePublisher = NotificationCenter.default.publisher(for: .newTrickDownloaded)
```

```
// Using Publishers with Combine

let trickNamePublisher = NotificationCenter.default.publisher(for: .newTrickDownloaded)
    .map { notification in
        return notification.userInfo?["data"] as! Data
    }
```

```
// Using Publishers with Combine

let trickNamePublisher = NotificationCenter.default.publisher(for: .newTrickDownloaded)
    .map { notification in
        return notification.userInfo?["data"] as! Data
    }
```

Data

Never

```
// Using Publishers with Combine

let trickNamePublisher = NotificationCenter.default.publisher(for: .newTrickDownloaded)
    .map { notification in
        return notification.userInfo?["data"] as! Data
    }
```

```
// Using Publishers with Combine

let trickNamePublisher = NotificationCenter.default.publisher(for: .newTrickDownloaded)
    .map { notification in
        return notification.userInfo?["data"] as! Data
    }
    .tryMap { data in
        let decoder = JSONDecoder()
        try decoder.decode(MagicTrick.self, from: data)
    }
```

```
// Using Publishers with Combine

let trickNamePublisher = NotificationCenter.default.publisher(for: .newTrickDownloaded)
    .map { notification in
        return notification.userInfo?["data"] as! Data
    }
    .tryMap { data in
        let decoder = JSONDecoder()
        try decoder.decode(MagicTrick.self, from: data)
    }
```

MagicTrick

Error

```
// Using Publishers with Combine

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

MagicTrick

Error

Error Handling

Every `Publisher` describes how they can fail

Use operators to react/recover from errors

```
// Using Publishers with Combine

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

```
// Using Publishers with Combine

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

MagicTrick

Never

Error Handling Operators

`assertNoFailure`

Error Handling Operators

`assertNoFailure`

Error Handling Operators

`assertNoFailure`

Error Handling Operators

assertNoFailure


```
Thread 1: Fatal error: Publisher.assertNoFailure: Received error: ...
```

Failure Handling Operators

```
assertNoFailure

retry

catch

mapError

setFailureType
```

Failure Handling Operators

`catch`

Failure Handling Operators

`catch`

Failure Handling Operators

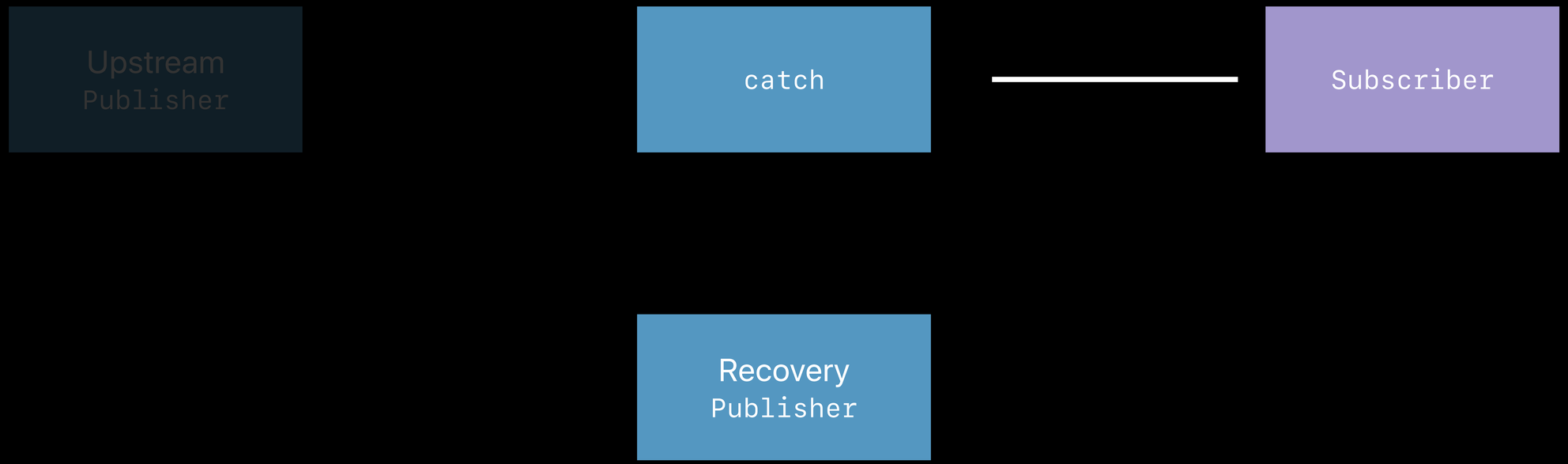
`catch`

Failure Handling Operators

`catch`

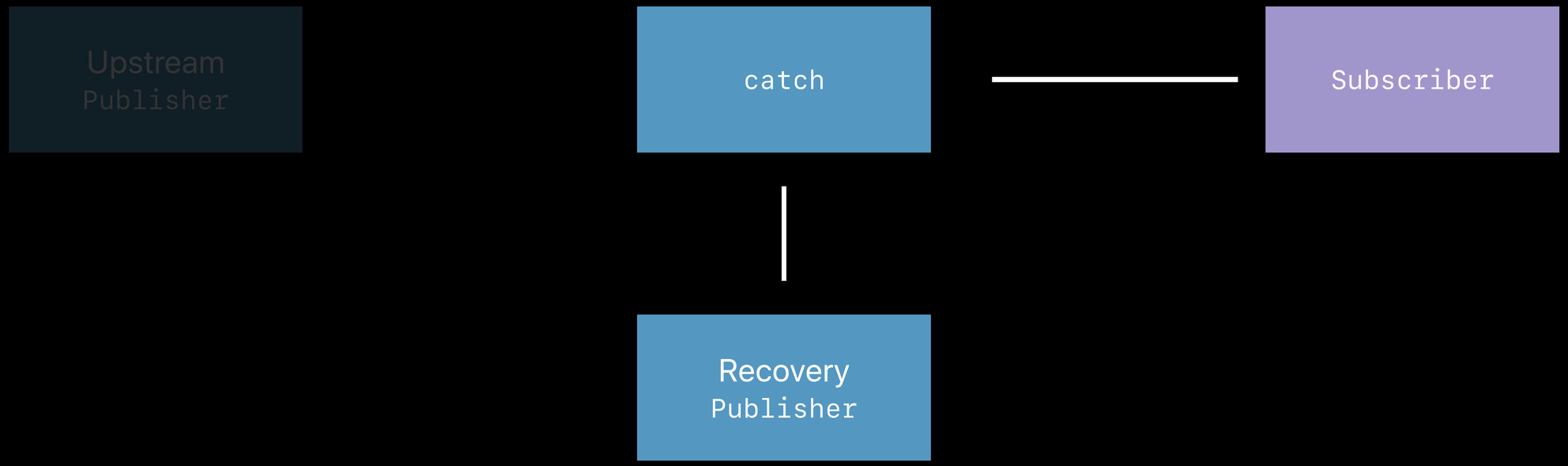
Failure Handling Operators

`catch`



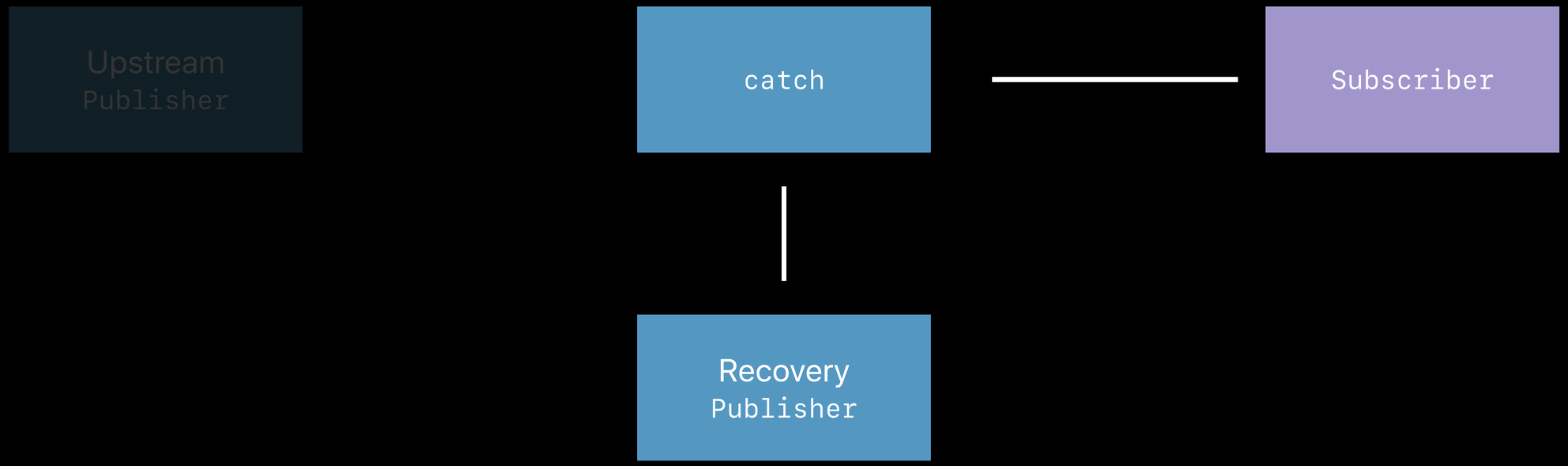
Failure Handling Operators

`catch`



Failure Handling Operators

`catch`



```
// Using Publishers with Combine

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

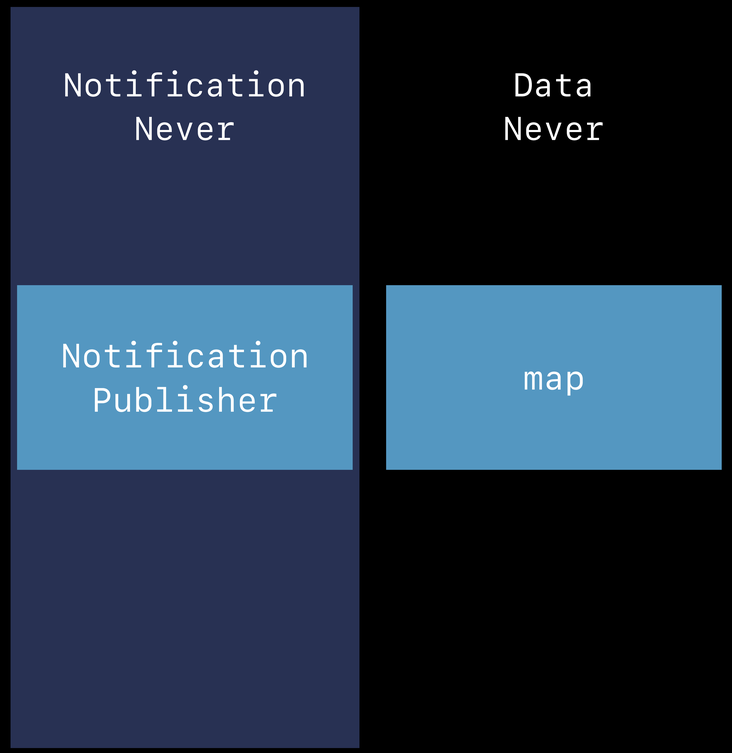
```
// Using Publishers with Combine

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

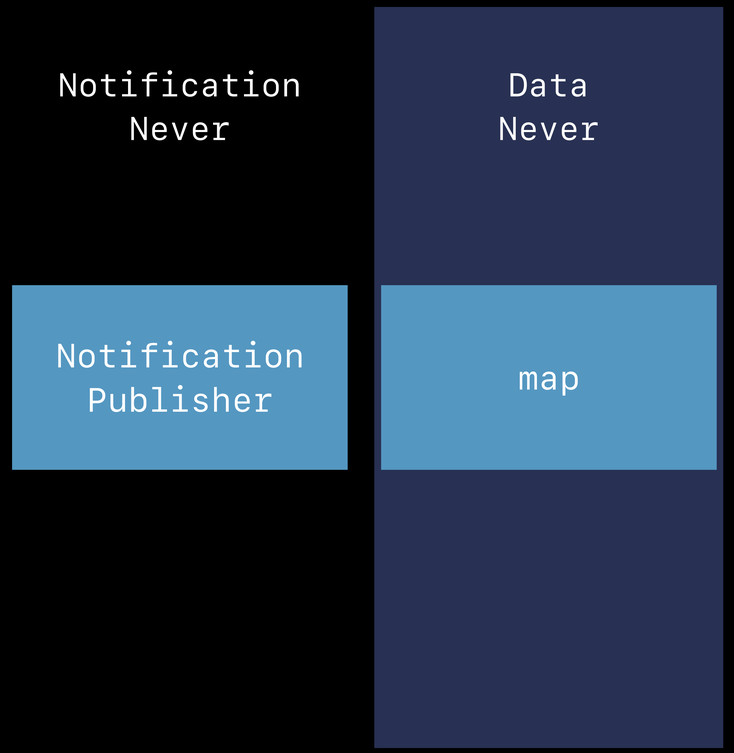
```
// Using Publishers with Combine

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

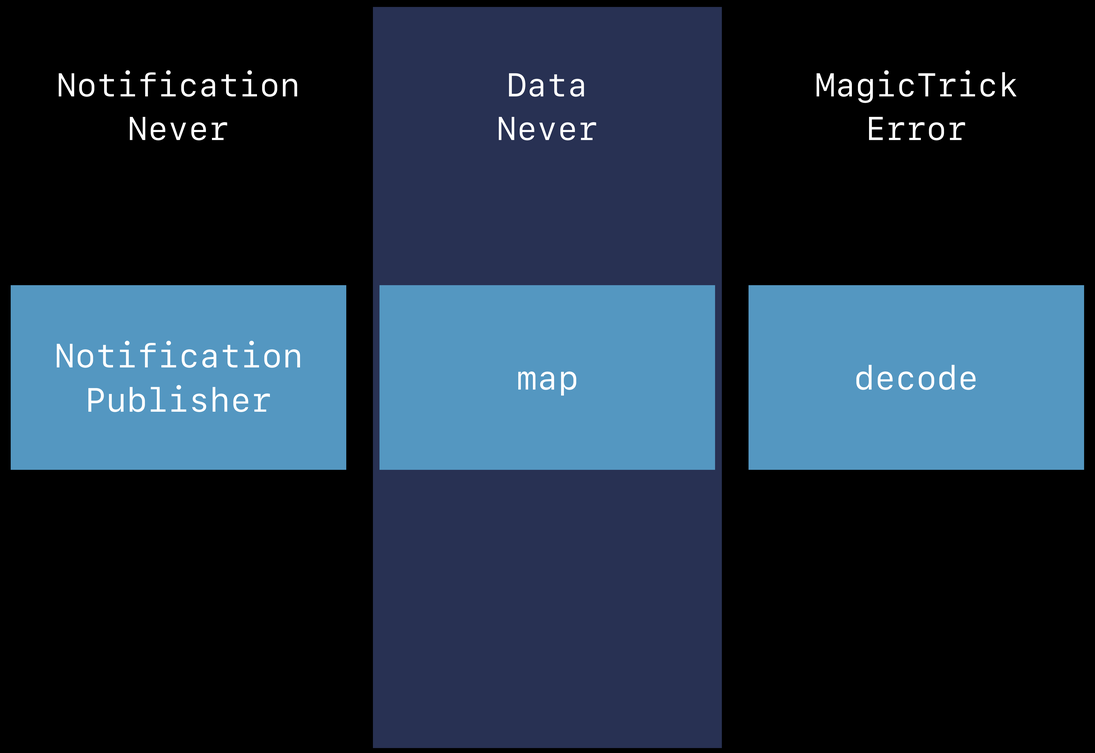
MagicTrick

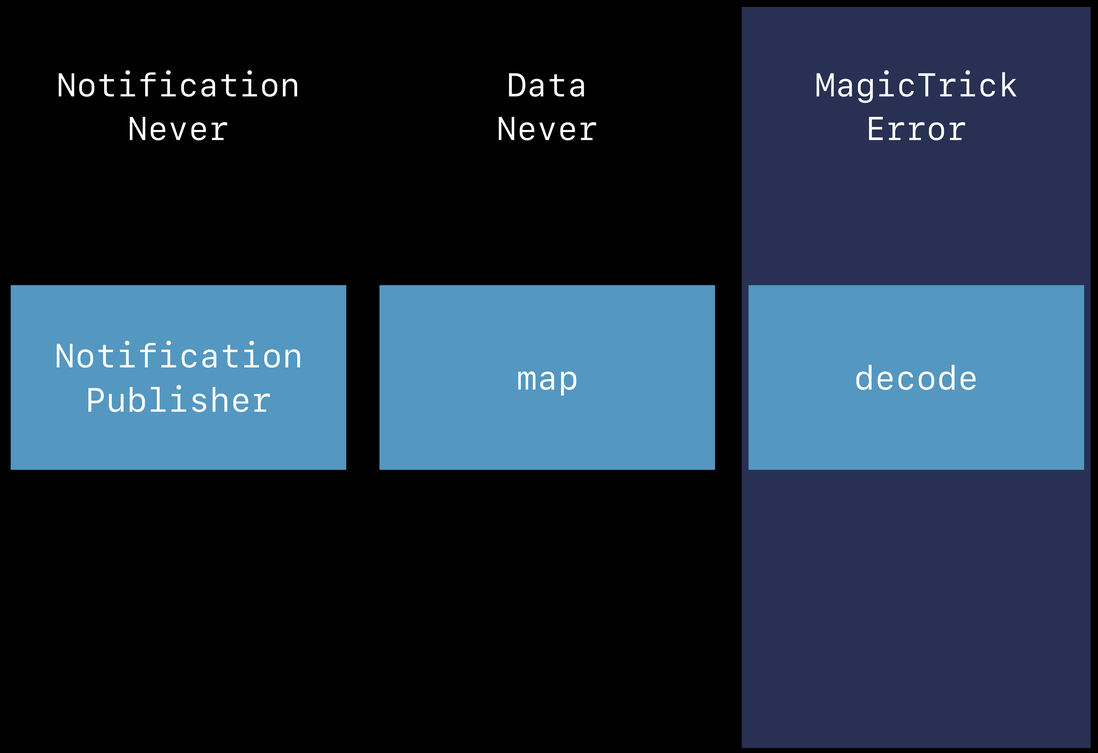
Never

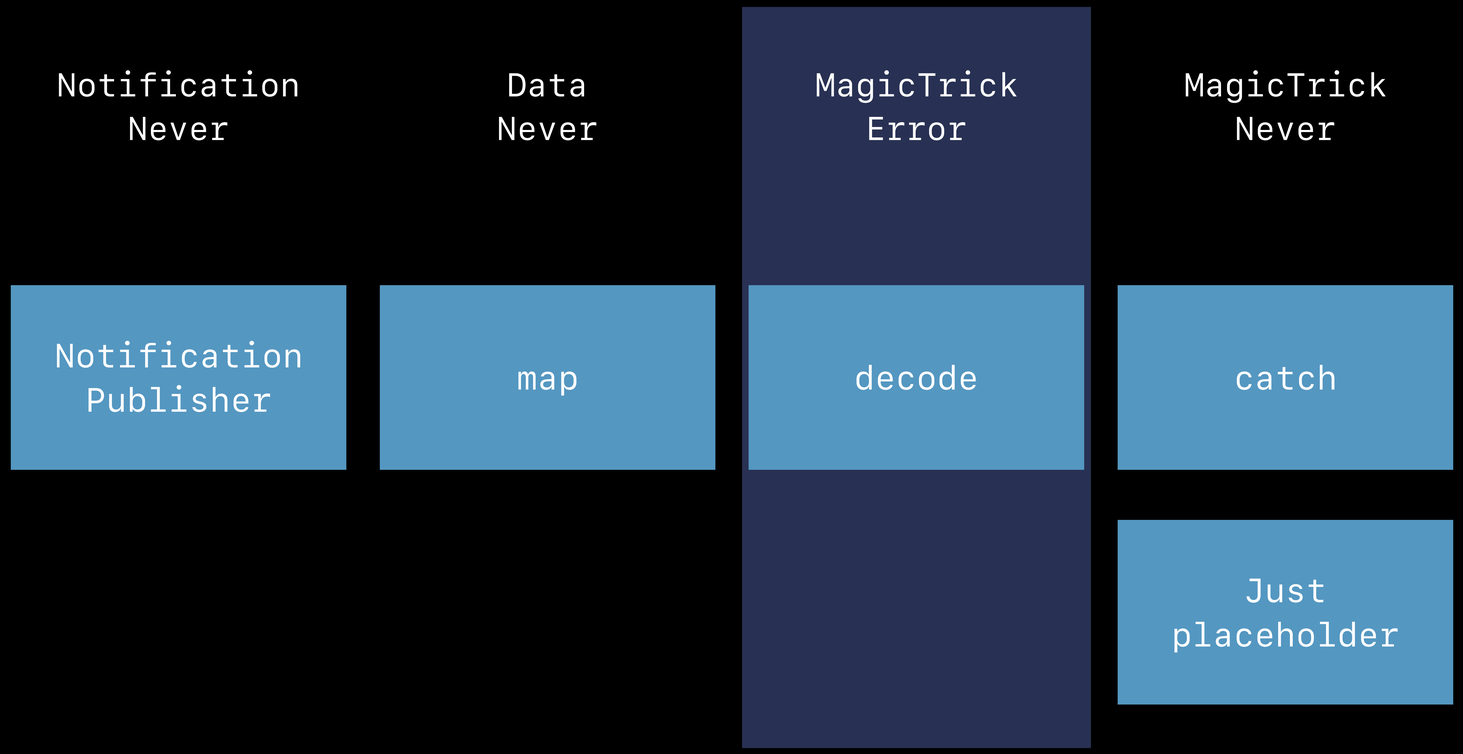

Notification
Never
Notification
Publisher


Notification
Never
Data
Never
Notification
Publisher
map


Notification
Never
Data
Never
Notification
Publisher
map

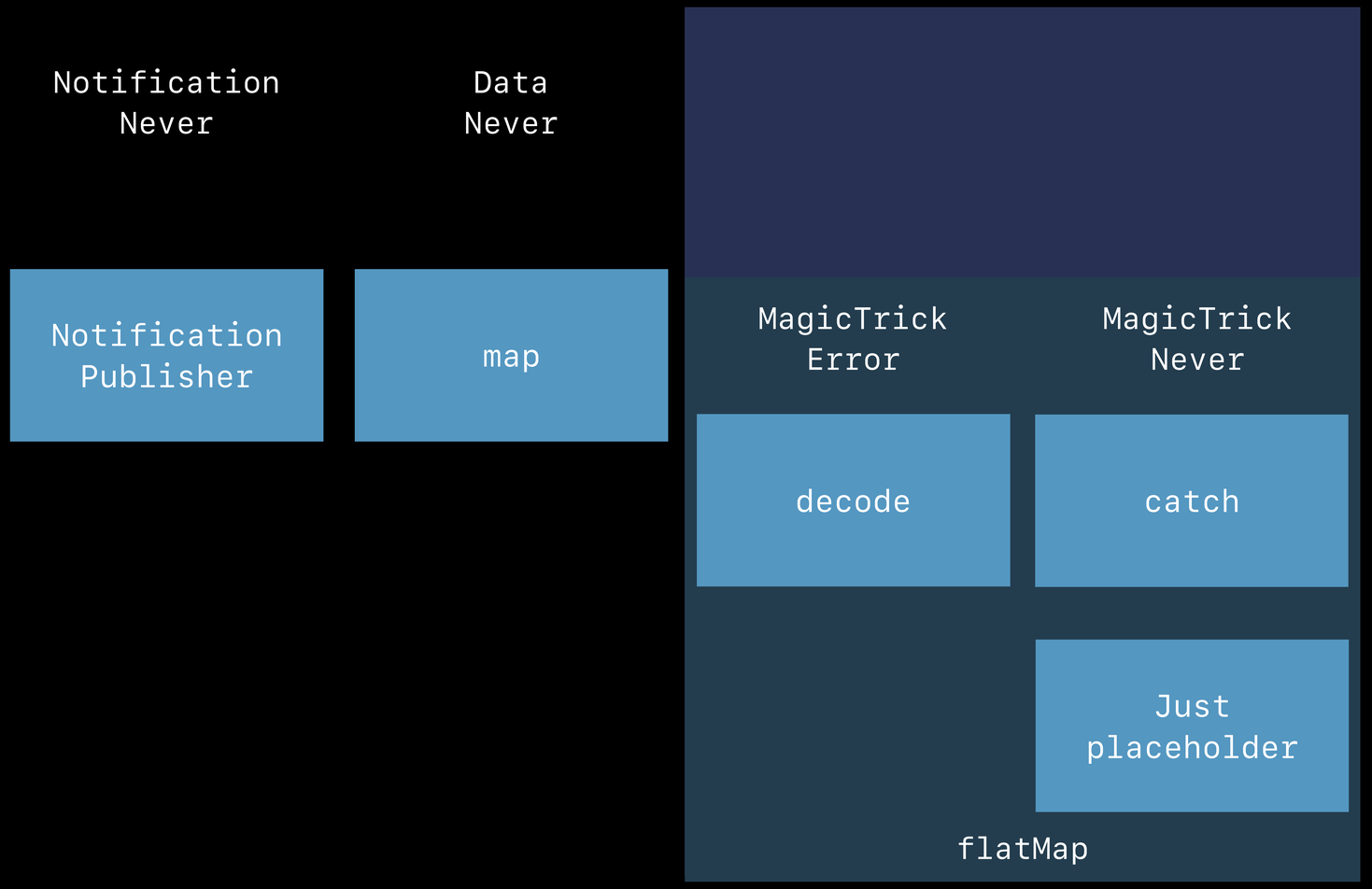

Notification
Never
Data
Never
MagicTrick
Error
Notification
Publisher
map
decode

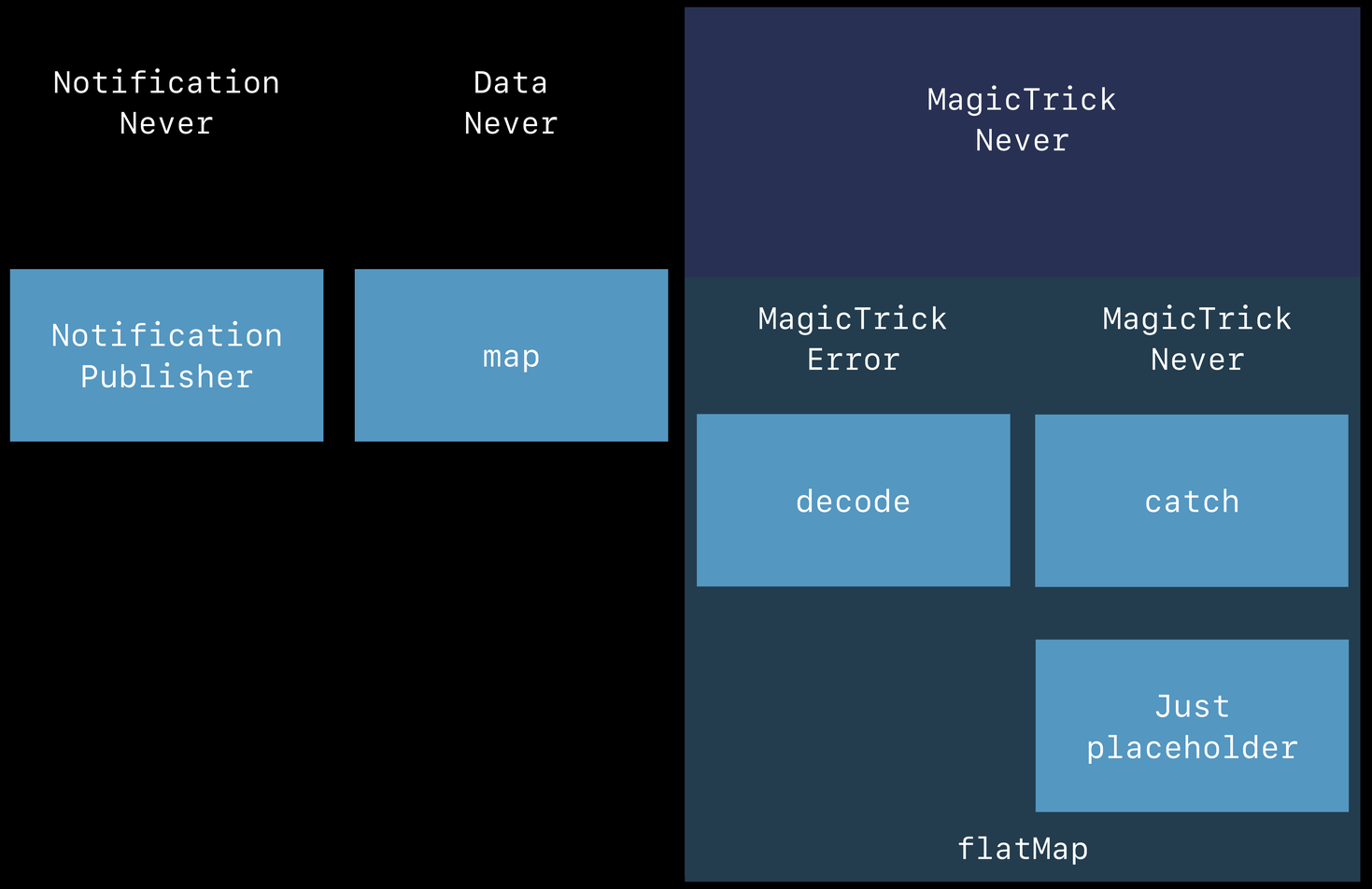

Notification
Never
Data
Never
MagicTrick
Error
Notification
Publisher
map
decode


Notification
Never
Data
Never
MagicTrick
Error
MagicTrick
Never
Notification
Publisher
map
decode
catch
Just
placeholder


Notification
Never
Data
Never
MagicTrick
Error
MagicTrick
Never
Notification
Publisher
map
decode
catch
Just
placeholder


Notification
Never
Data
Never
MagicTrick
Error
MagicTrick
Never
Notification
Publisher
map
decode
catch
Just
placeholder

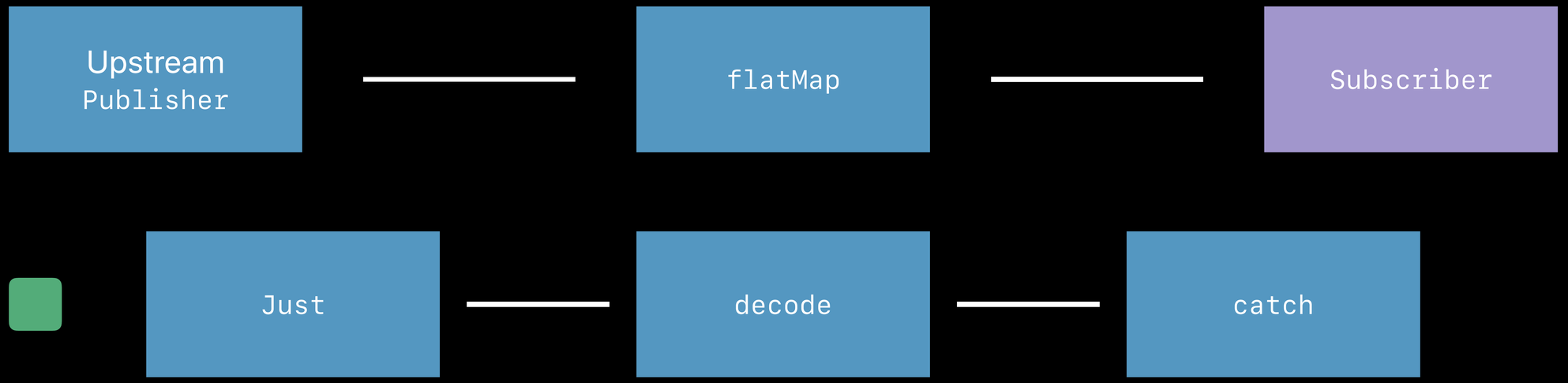

Notification
Never
Data
Never
Notification
Publisher
map
MagicTrick
Error
MagicTrick
Never
decode
catch
Just
placeholder
flatMap


Notification
Never
Data
Never
MagicTrick
Never
Notification
Publisher
map
MagicTrick
Error
MagicTrick
Never
decode
catch
Just
placeholder
flatMap

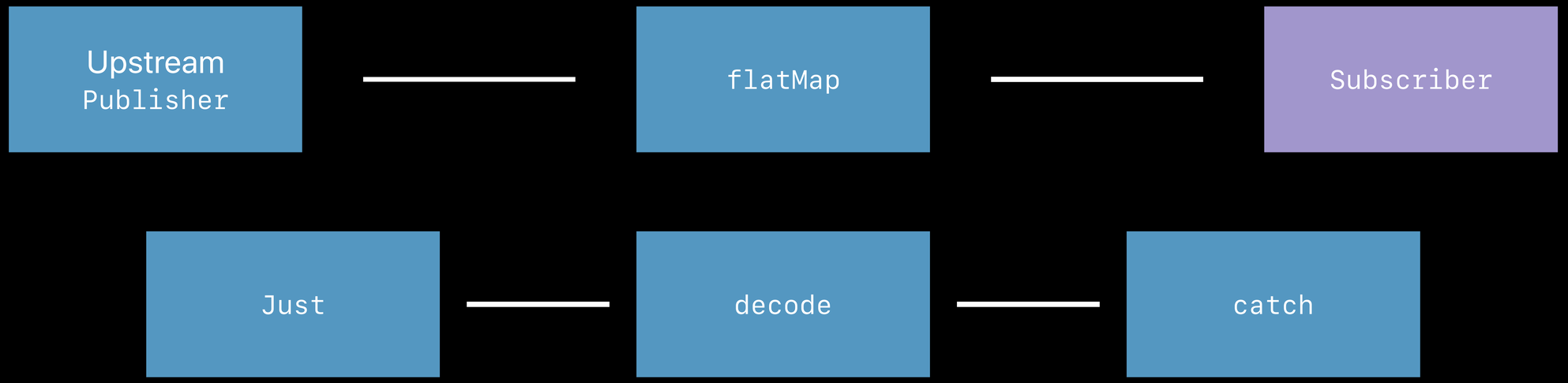
Flat Map

Flat Map

Flat Map



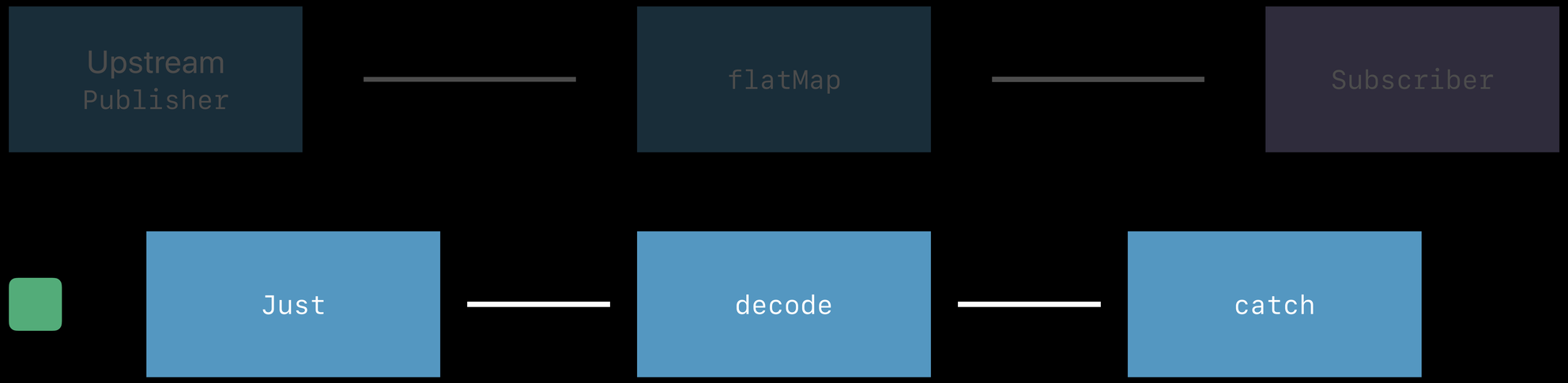
Flat Map



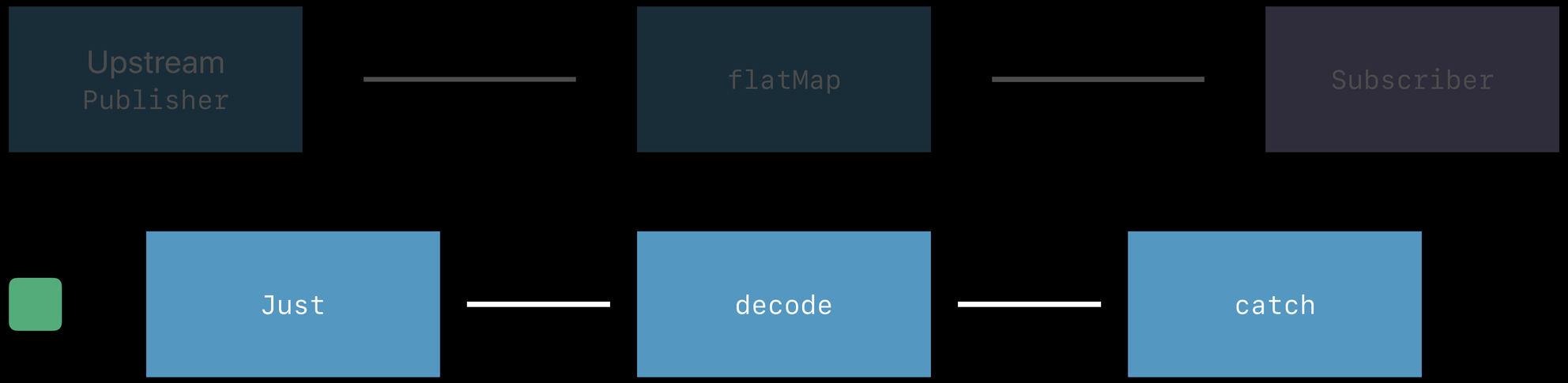
Flat Map

Flat Map

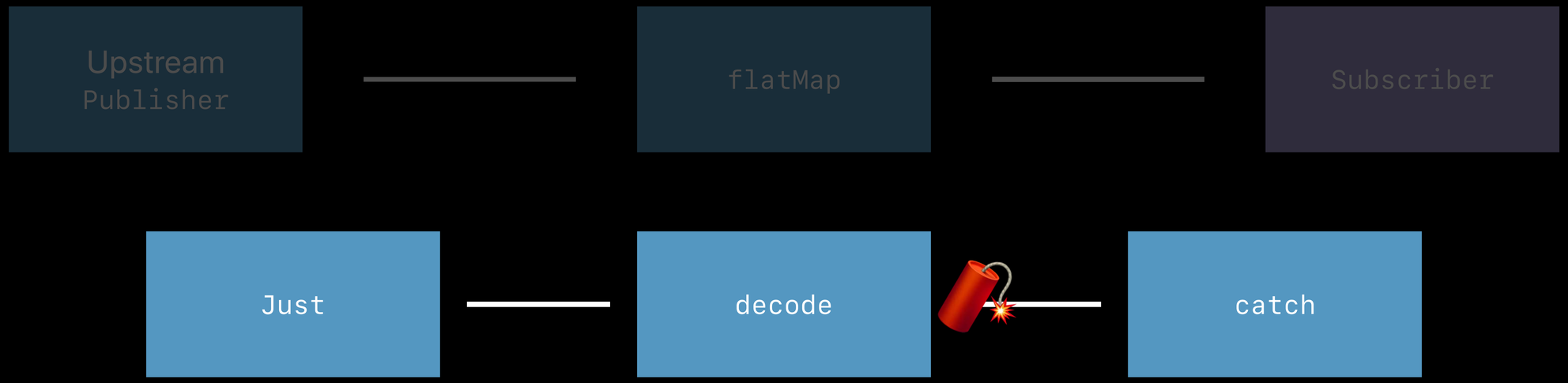
Flat Map



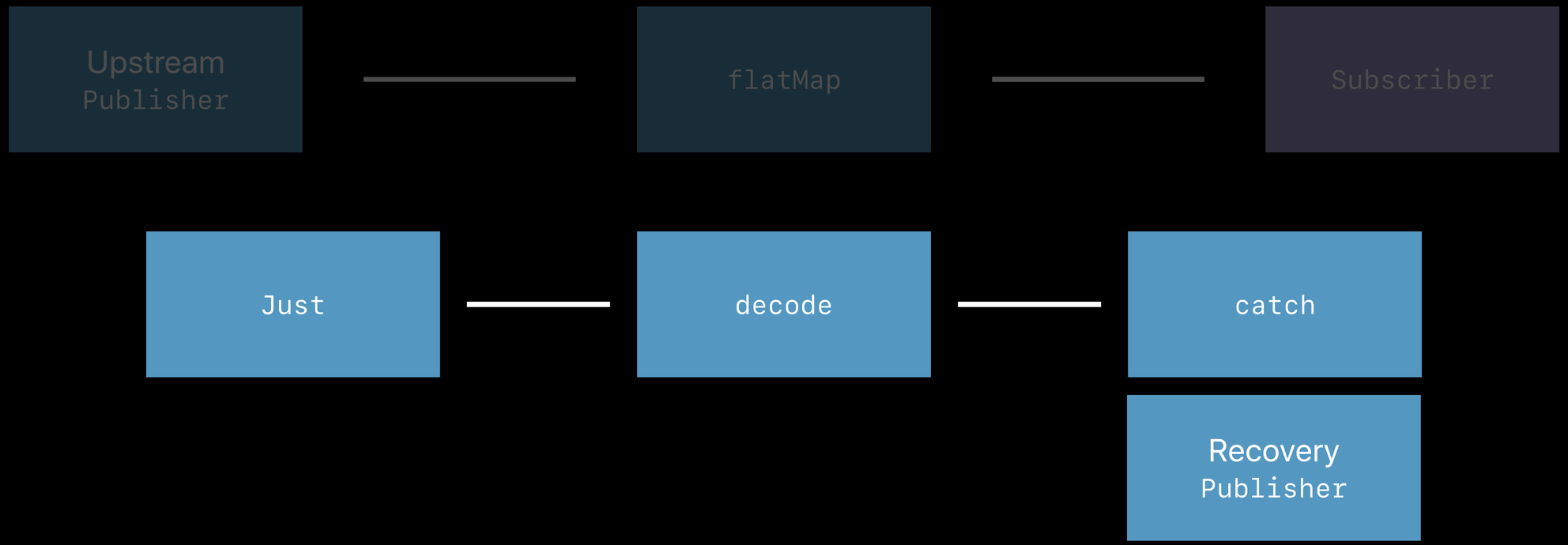
Flat Map



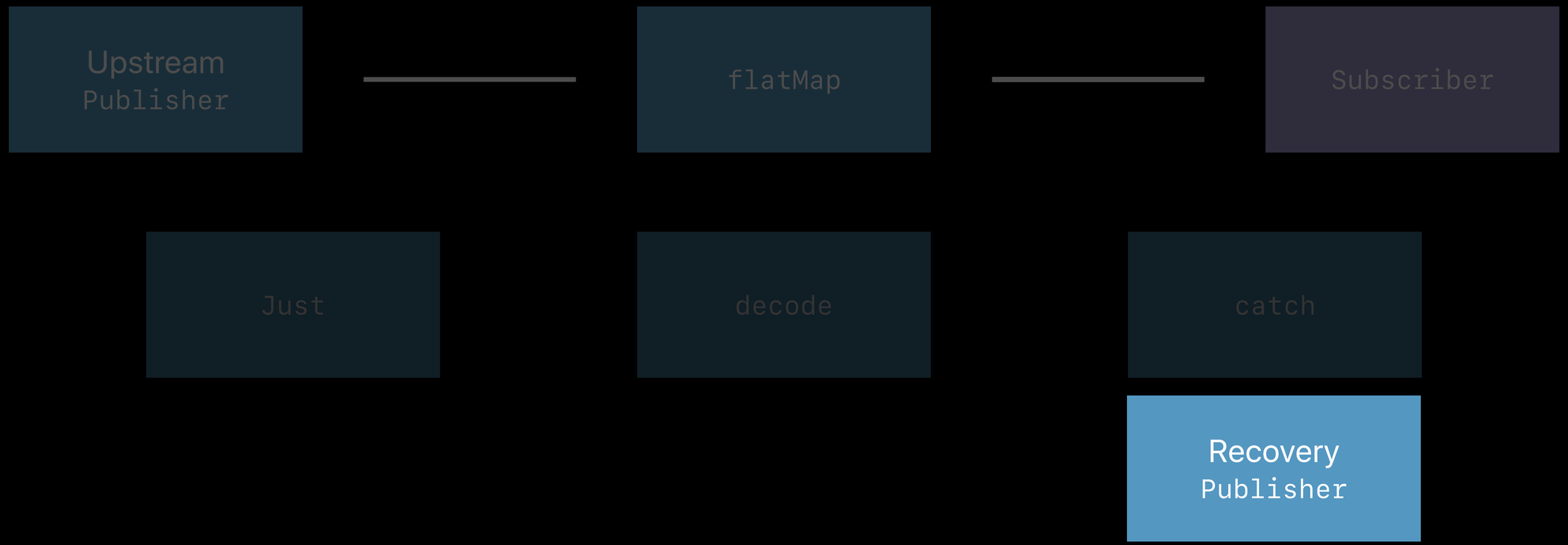
Flat Map



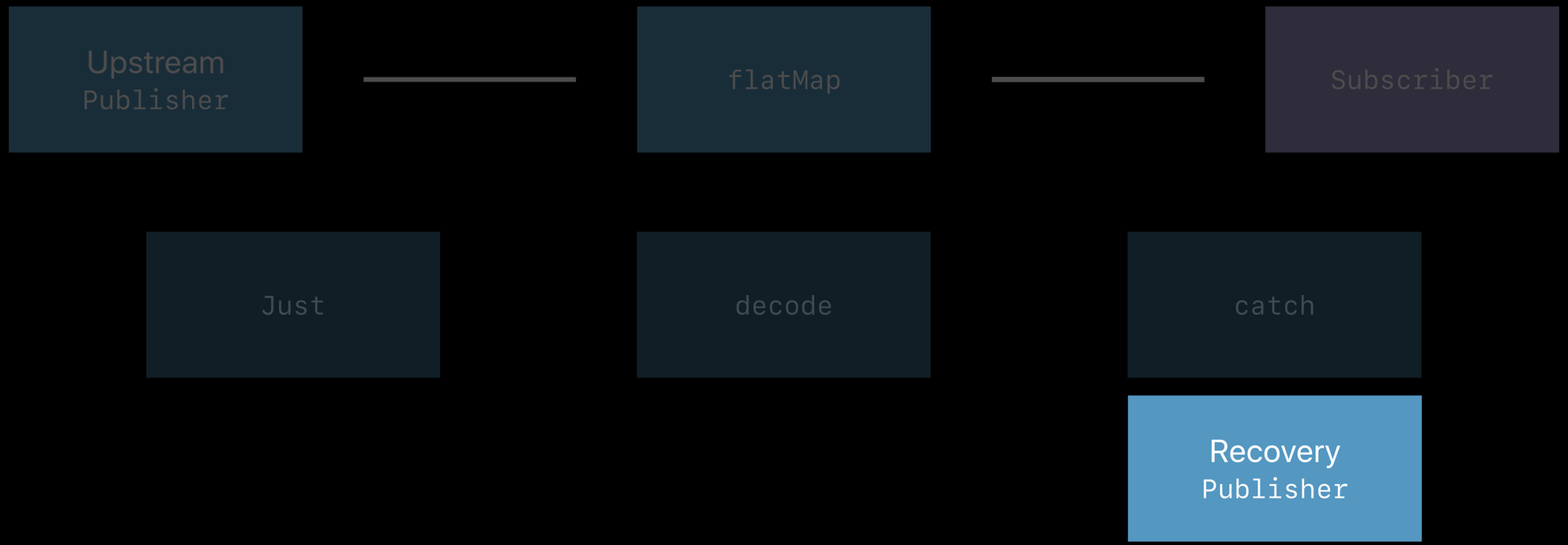
Flat Map


Upstream
Publisher
flatMap
Subscriber
Just
decode
catch
Recovery
Publisher

Flat Map


Upstream
Publisher
flatMap
Subscriber
Just
decode
catch
Recovery
Publisher

Flat Map


Upstream
Publisher
flatMap
Subscriber
Just
decode
catch
Recovery
Publisher

Flat Map

Flat Map

```
// Using Publishers with Combine

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

```
// 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)
            }
    }
```

```
// 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)
            }
    }
```

```
// 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)
            }
    }
```

MagicTrick

Never

```
// 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)
```

```
// 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

Scheduled Operators

Scheduler describes

- When
- Where

Supported by `RunLoop` and `DispatchQueue`

Scheduled Operators

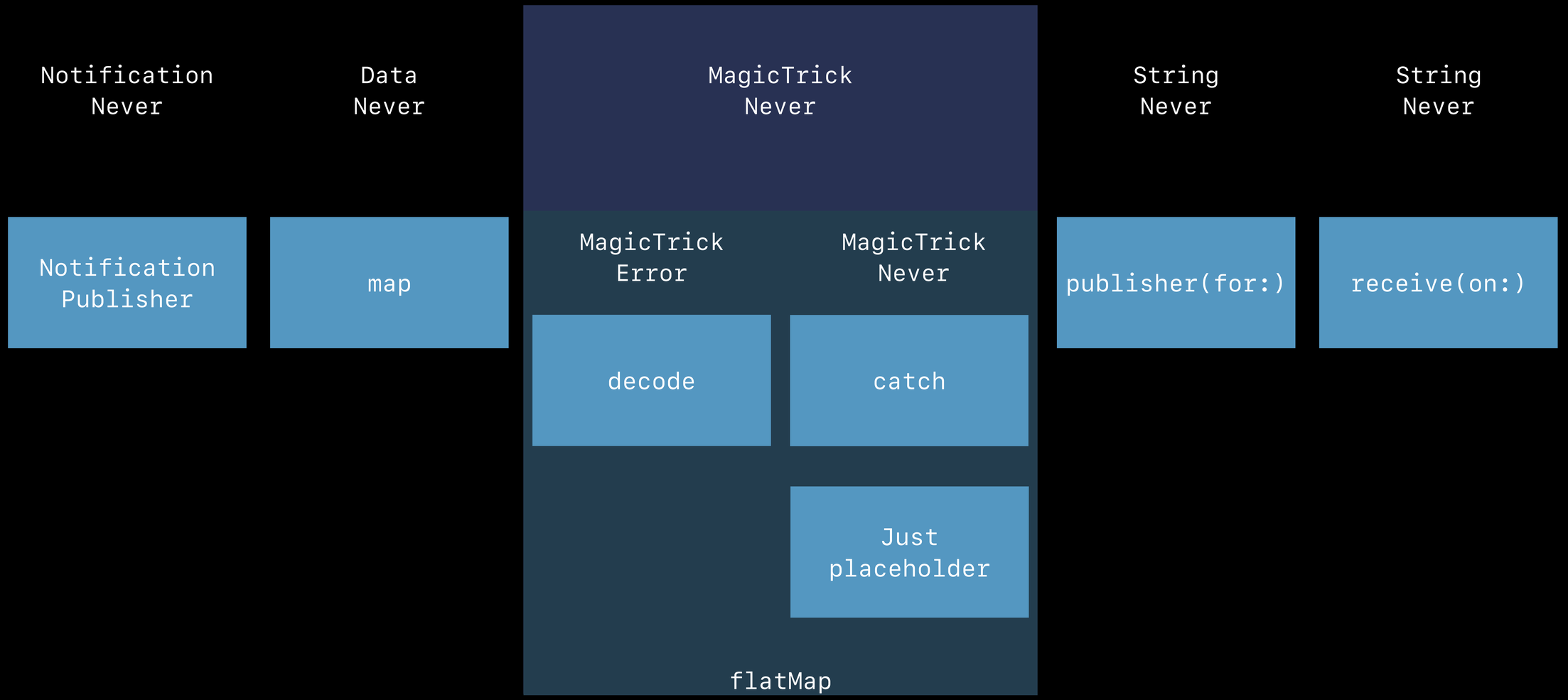
```
delay

debounce

throttle

receive(on:)

subscribe(on:)
```

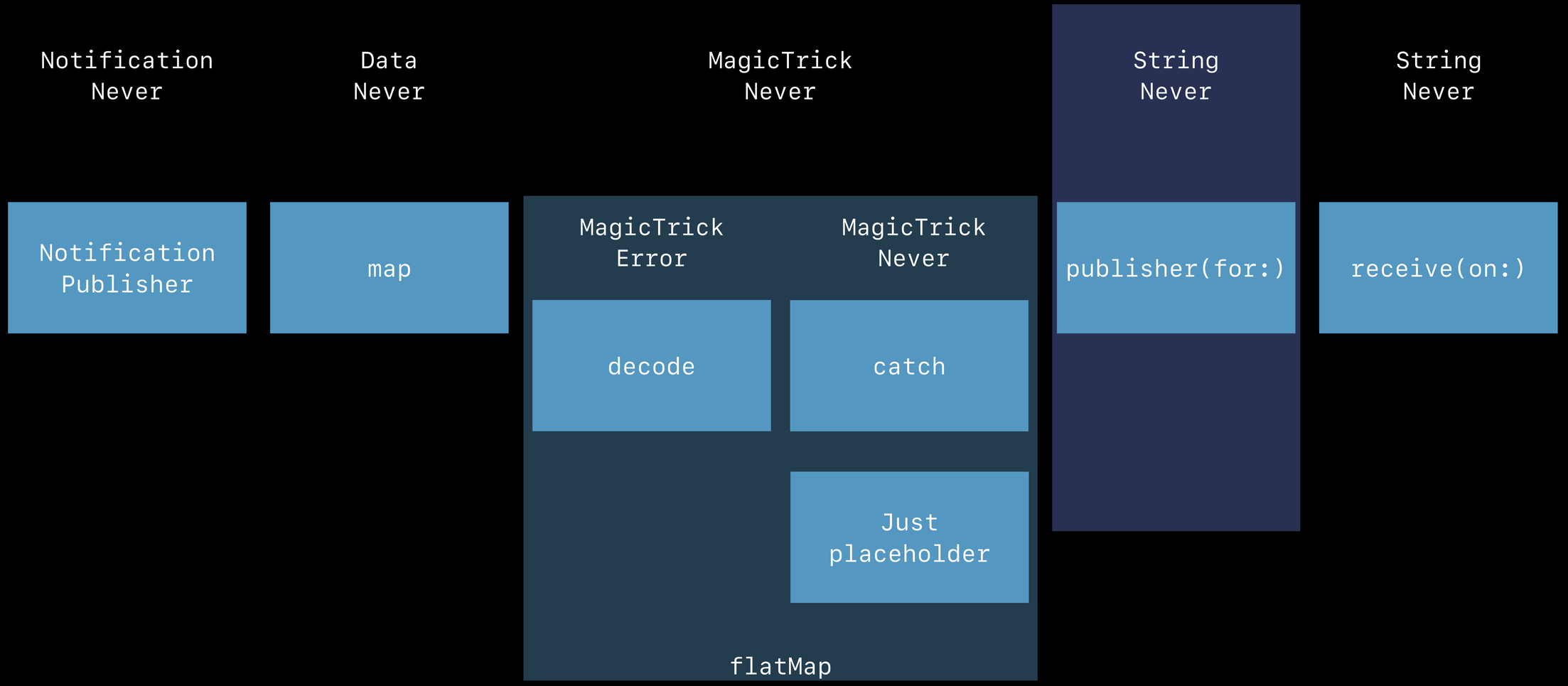
```
// Using Publishers with Combine

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)
```

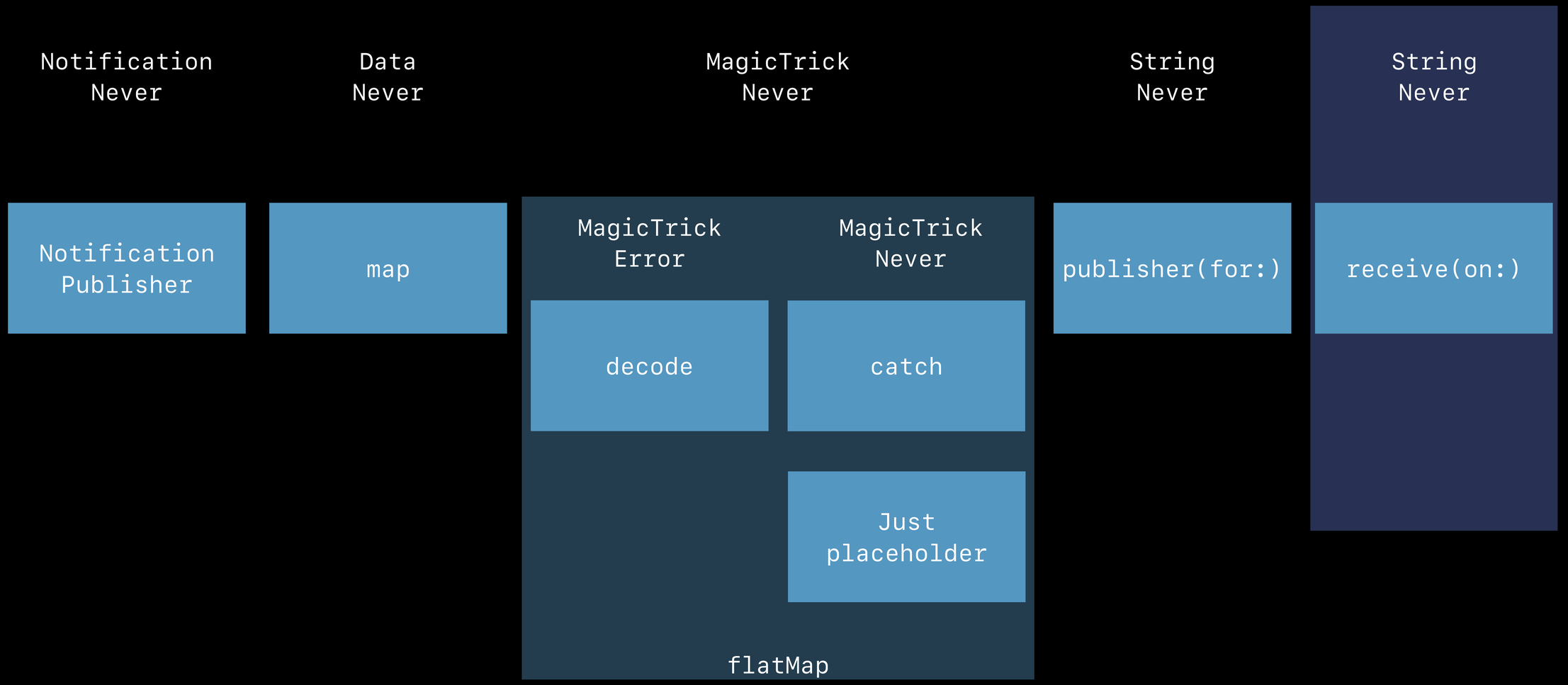
```
// Using Publishers with Combine

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)
```

String

Never


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


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


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

Publishers

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>)
}
```

Subscriber

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

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

Subscriber

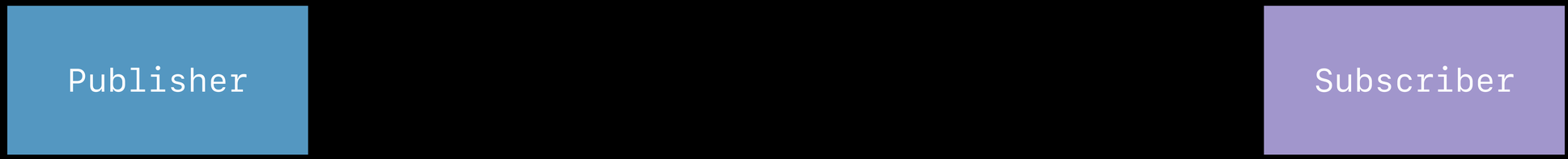
```
protocol Subscriber {
    associatedtype Input
    associatedtype Failure: Error

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

Subscriber

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

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

Valid Streams

Subscription



Valid Streams

Subscription

Valid Streams

Zero or more values

Valid Streams

Zero or more values

Valid Streams

Completion

`receive(completion:)`

Valid Streams

Completion

`receive(completion:)`

Valid Streams

Completion

`receive(completion:)`

Valid Streams

Completion

`receive(completion:)`

Valid Streams

Completion

```
receive(completion:)
```

Publisher

Subscriber

Exactly One
Subscription

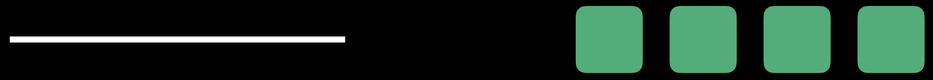
Exactly One
Subscription

Exactly One
Subscription

Zero or More
Values

Exactly One
Subscription

Zero or More
Values



Exactly One Subscription

Zero or More Values

At Most One Completion

Exactly One Subscription

Zero or More Values

At Most One Completion

Kinds of Subscribers

Key Path Assignment

Sinks

Subjects

SwiftUI

```
// Using Subscribers 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)
```

```
// Using Subscribers with Combine

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

```
// Using Subscribers with Combine

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

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

// ...

canceller.cancel()
```

```
// Using Subscribers with Combine

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

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

// ...

canceller.cancel()
```

Cancellation

Built into Combine

Terminate subscriptions early

Cancellation

Built into Combine

Terminate subscriptions early

```
protocol Cancellable {
    func cancel()
}

final class AnyCancellable: Cancellable {} // Calls `cancel` on deinit
```

Cancellation

Built into Combine

Terminate subscriptions early

```
protocol Cancellable {
    func cancel()
}

final class AnyCancellable: Cancellable {} // Calls `cancel` on deinit
```

Cancellation

Built into Combine

Terminate subscriptions early

```
protocol Cancellable {
    func cancel()
}

final class AnyCancellable: Cancellable {} // Calls `cancel` on deinit
```

```
// Using Subscribers with Combine

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

let canceller = trickNamePublisher.sink { trickName in
   // Do Something with trickName
}
```

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>)
}
```

Subjects

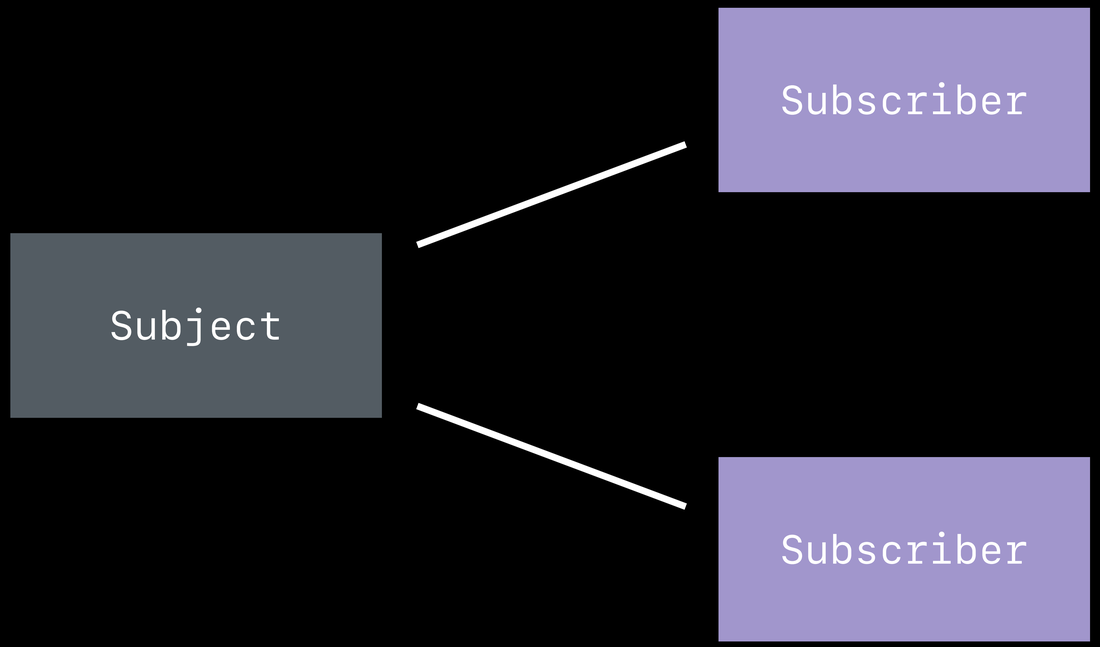
Behave like both Publisher and Subscriber

Broadcast values to multiple subscribers

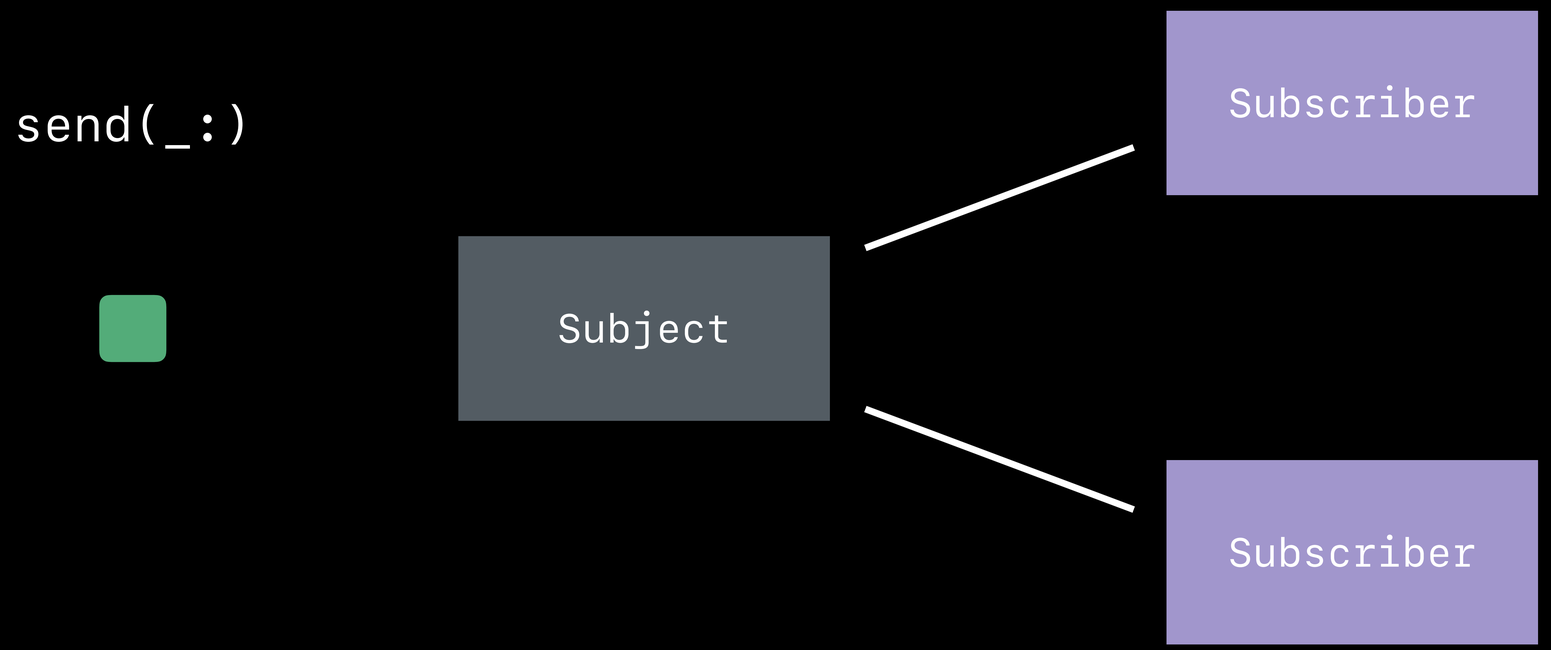
```
protocol Subject: Publisher, AnyObject {
    func send(_ value: Output)
    func send(completion: Subscribers.Completion<Failure>)
}
```

Subject

Subject

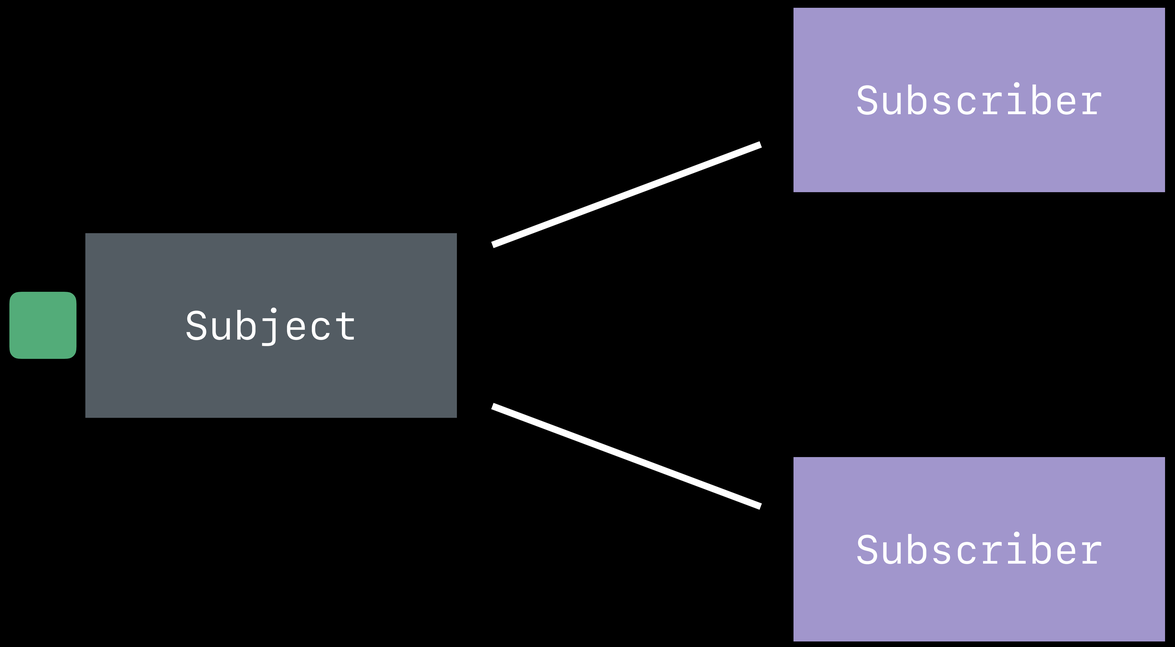


Subject



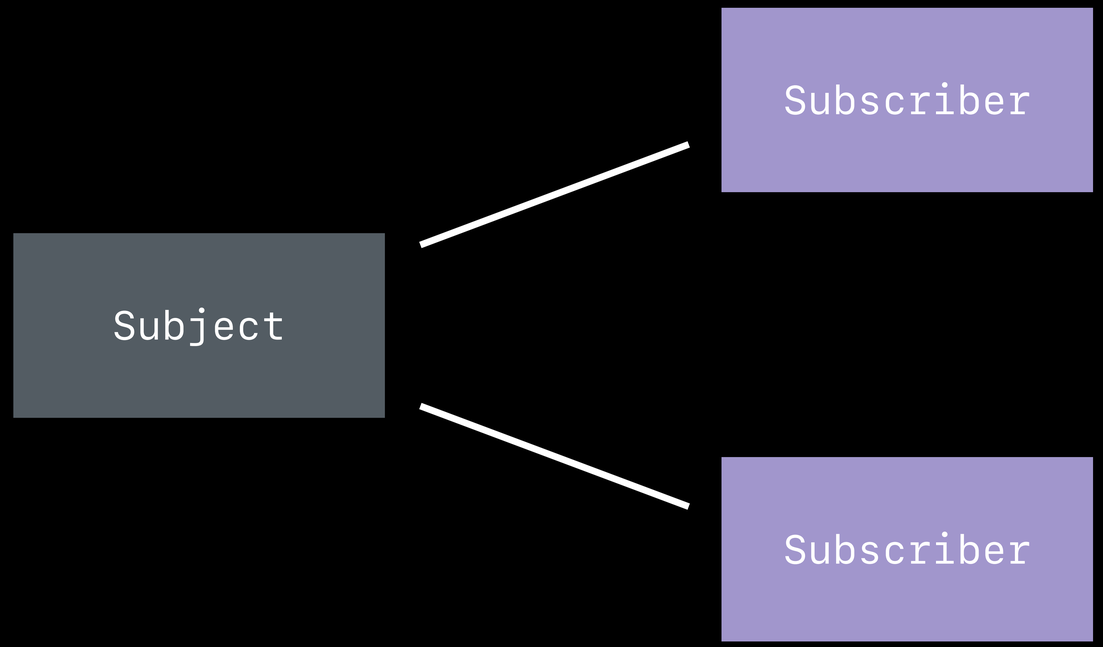
Subject

`send(_:)`

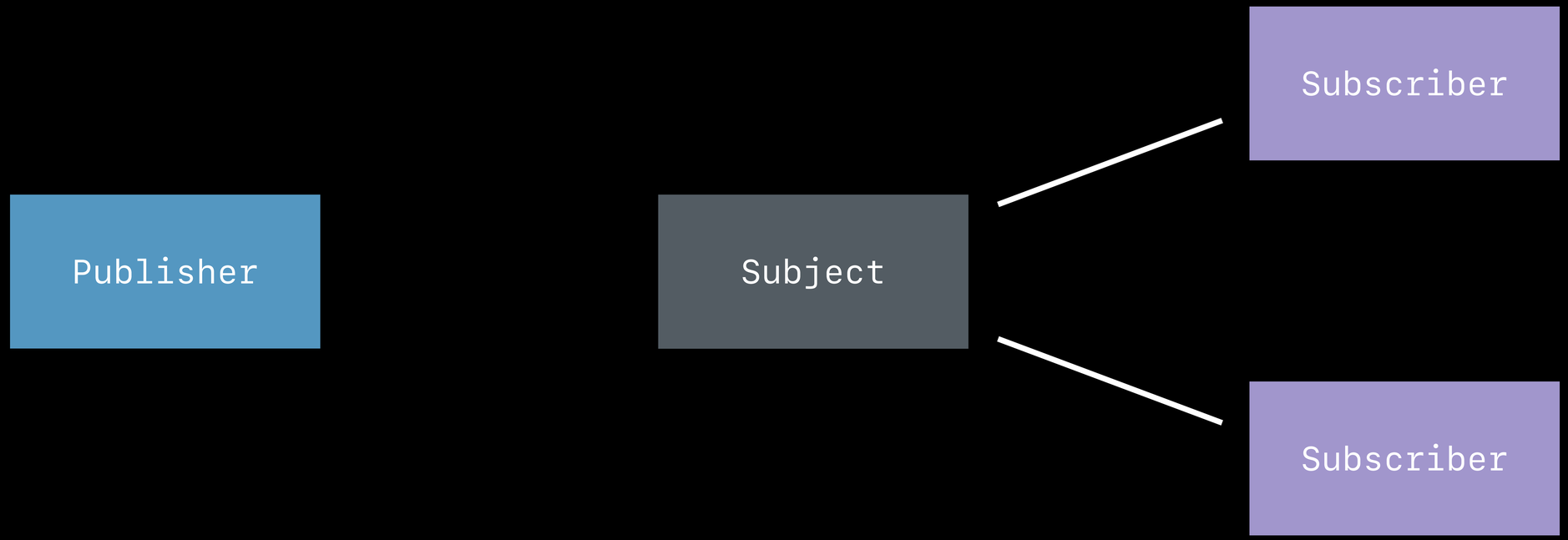


Subject

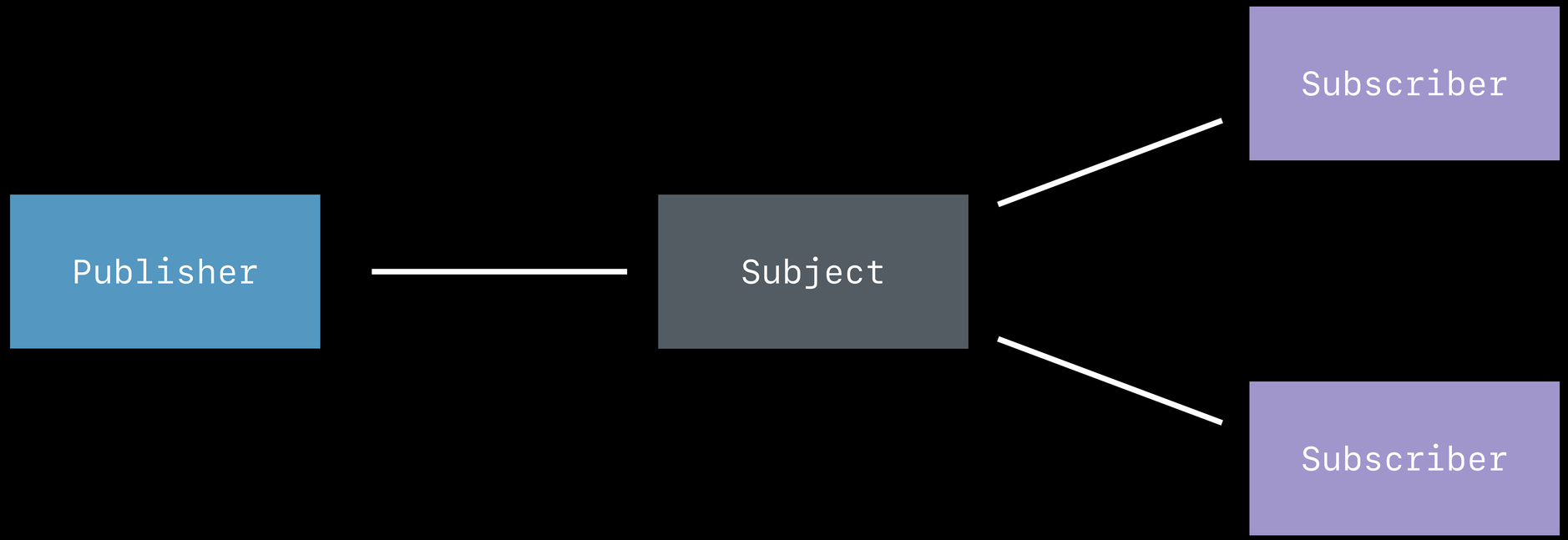
`send(_:)`



Subject



Subject



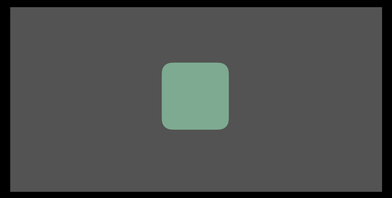
Kinds of Subjects

`Passthrough`

`CurrentValue`

Kinds of Subjects

`Passthrough`



`CurrentValue`

Kinds of Subjects

`Passthrough`

`CurrentValue`

Kinds of Subjects

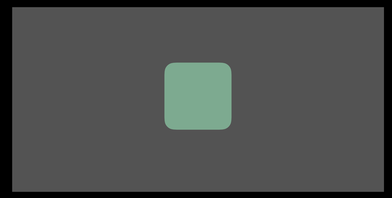
`Passthrough`

`CurrentValue`

Kinds of Subjects

`Passthrough`

`CurrentValue`



```
// Using Subjects with Combine

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

```
// Using Subjects with Combine

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

let magicWordsSubject = PassthroughSubject<String, Never>()
```

```
// Using Subjects with Combine

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

let magicWordsSubject = PassthroughSubject<String, Never>()

trickNamePublisher.subscribe(magicWordsSubject)
```

```
// 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
}
```

```
// 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")
```

```
// 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")

let sharedTrickNamePublisher = trickNamePublisher.share()
```

Working with SwiftUI

SwiftUI owns the `Subscriber`

You just need to bring a `Publisher`

SwiftUI BindableObject

```
protocol BindableObject {
    associatedtype PublisherType : Publisher where PublisherType.Failure == Never

    var didChange: PublisherType { get }
}
```

SwiftUI BindableObject

```
protocol BindableObject {
    associatedtype PublisherType : Publisher where PublisherType.Failure == Never

    var didChange: PublisherType { get }
}
```

SwiftUI BindableObject

```
protocol BindableObject {
    associatedtype PublisherType : Publisher where PublisherType.Failure == Never

    var didChange: PublisherType { get }
}
```

SwiftUI BindableObject

```
protocol BindableObject {
    associatedtype PublisherType : Publisher where PublisherType.Failure == Never

    var didChange: PublisherType { get }
}
```

SwiftUI BindableObject

```
protocol BindableObject {
    associatedtype PublisherType : Publisher where PublisherType.Failure == Never

    var didChange: PublisherType { get }
}
```

SwiftUI BindableObject

```
protocol BindableObject {
    associatedtype PublisherType : Publisher where PublisherType.Failure == Never

    var didChange: PublisherType { get }
}
```

```
// Combine with SwiftUI

class WizardModel {
    var trick: WizardTrick
    var wand: Wand?
}
```

```
// Combine with SwiftUI

class WizardModel : BindableObject {
    var trick: WizardTrick
    var wand: Wand?

    let didChange = PassthroughSubject<Void, Never>()
}
```

```
// Combine with SwiftUI

class WizardModel : BindableObject {
    var trick: WizardTrick
    var wand: Wand?

    let didChange = PassthroughSubject<Void, Never>()
}
```

```
// Combine with SwiftUI

class WizardModel : BindableObject {
    var trick: WizardTrick { didSet { didChange.send() } }
    var wand: Wand? { didSet { didChange.send() } }

    let didChange = PassthroughSubject<Void, Never>()
}
```

```
// Combine with SwiftUI

class WizardModel : BindableObject {
    var trick: WizardTrick { didSet { didChange.send() } }
    var wand: Wand? { didSet { didChange.send() } }

    let didChange = PassthroughSubject<Void, Never>()
}

struct TrickView: View {
    @ObjectBinding var model: WizardModel

    var body: some View {
        Text(model.trick.name)
    }
}
```

```
// Combine with SwiftUI

class WizardModel : BindableObject {
    var trick: WizardTrick { didSet { didChange.send() } }
    var wand: Wand? { didSet { didChange.send() } }

    let didChange = PassthroughSubject<Void, Never>()
}

struct TrickView: View {
    @ObjectBinding var model: WizardModel

    var body: some View {
        Text(model.trick.name)
    }
}
```

Combine

Many built in

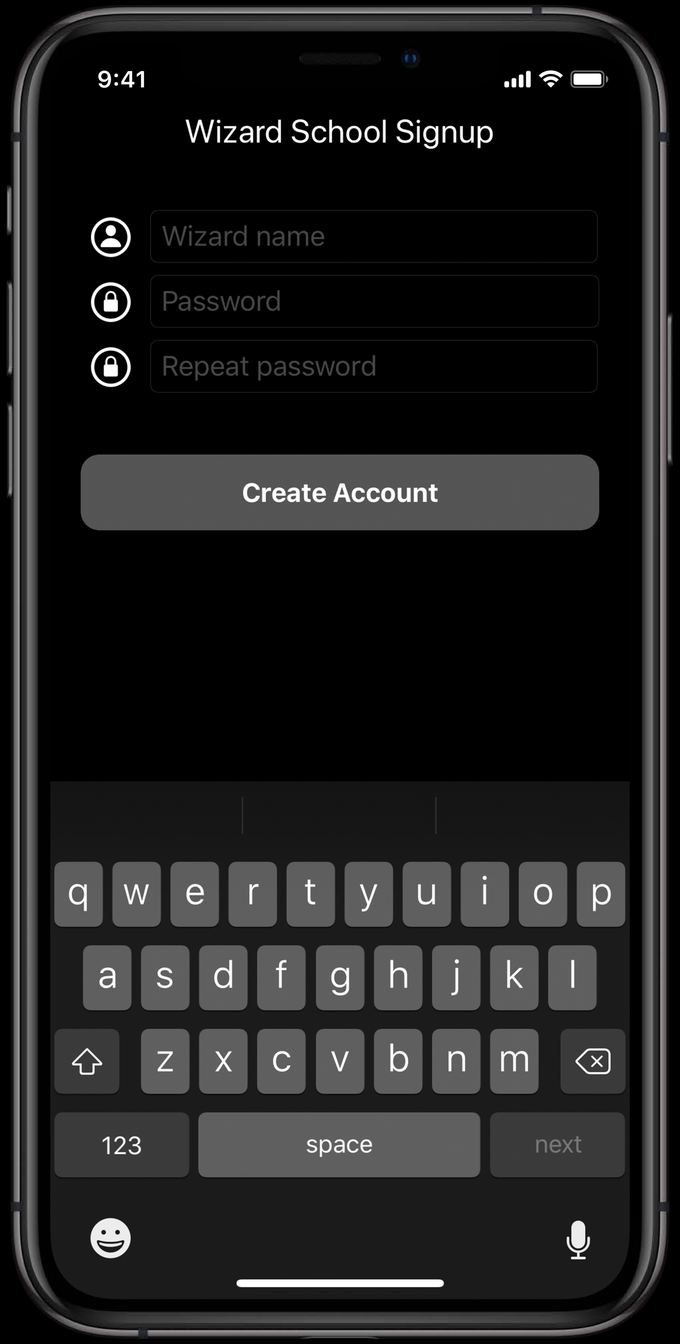
- Publishers
- Subscribers
- Subjects

Common functionality in over 90 operators

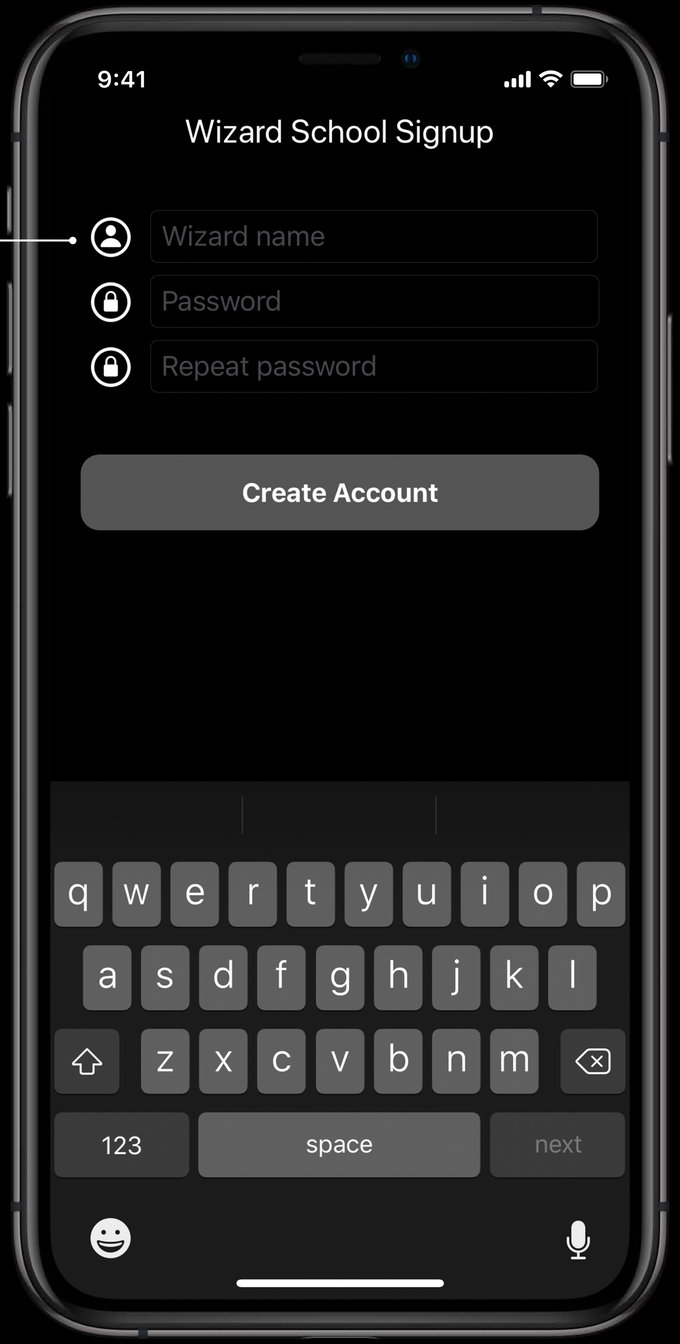
Integrating Combine

Ben D. Jones, Foundation

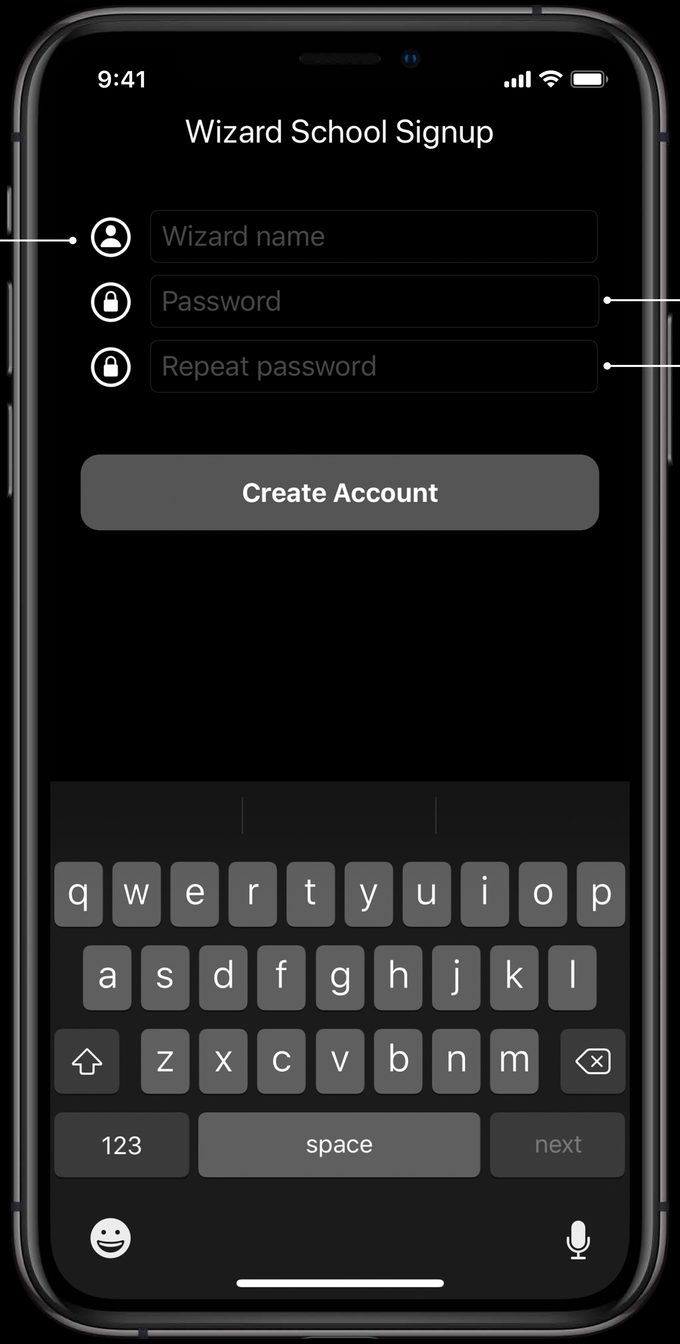
Designed for composition


9:41
Wizard School Signup
Wizard name
Password
Repeat password
Create Account
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
123
space
next

User name is valid
according to server

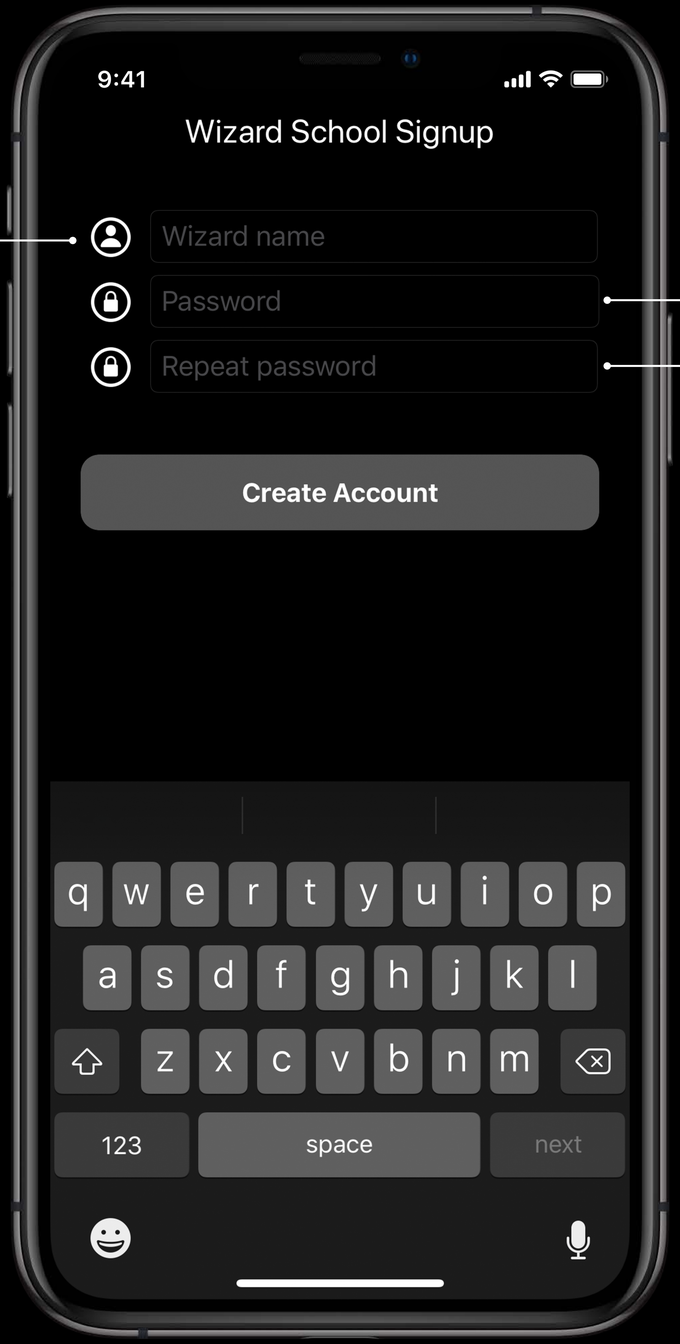


User name is valid according to server



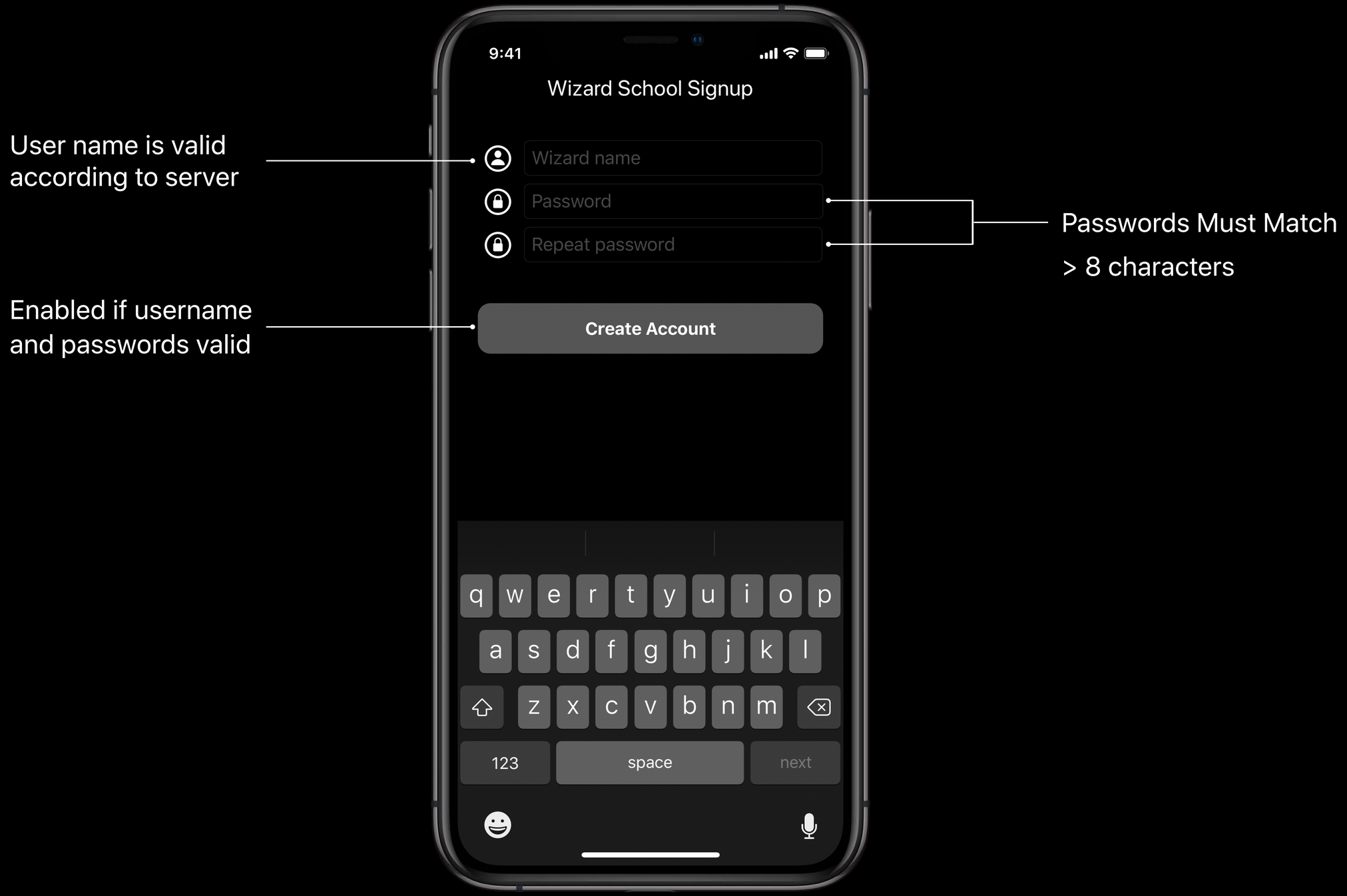
Passwords Must Match

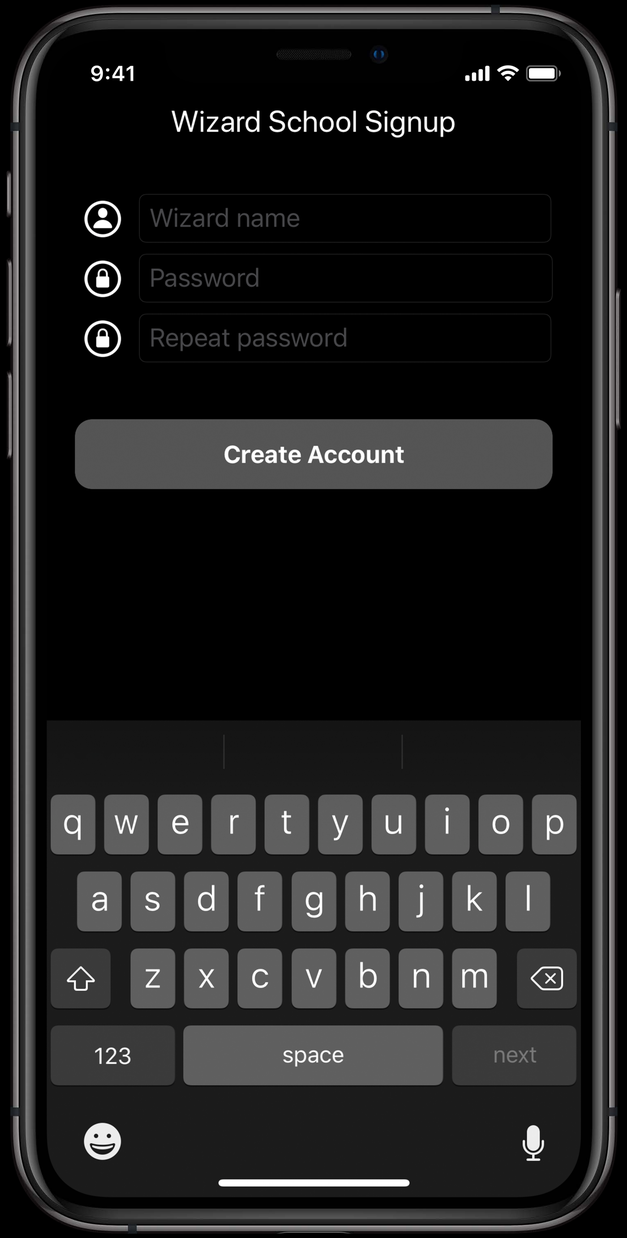
User name is valid according to server

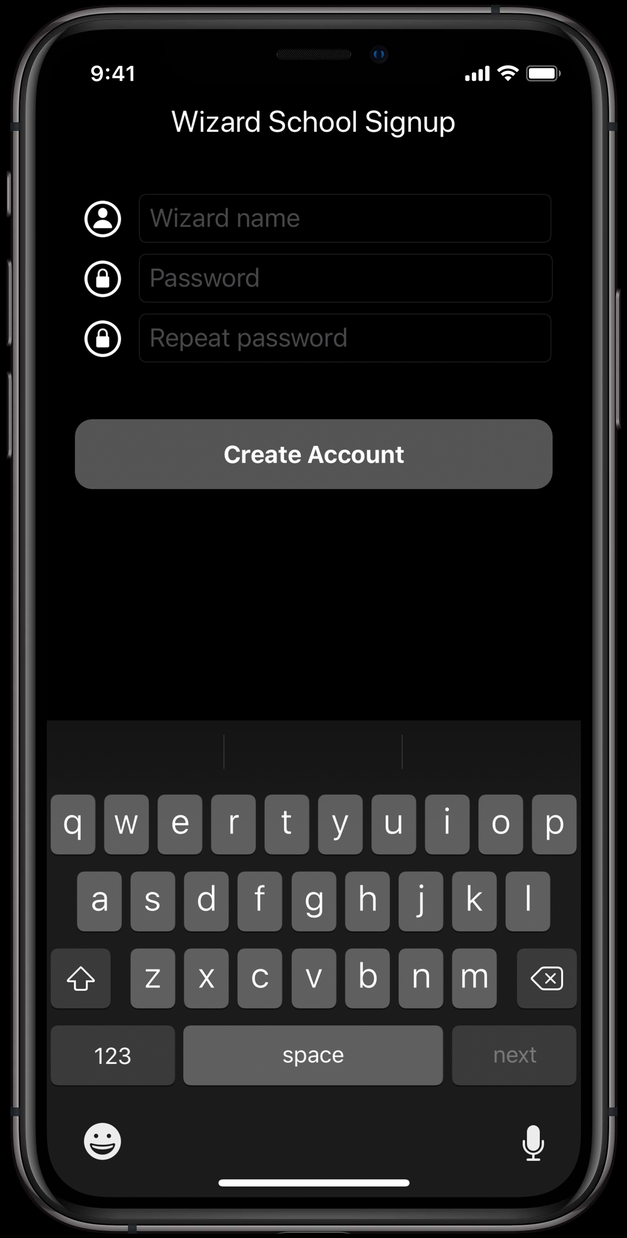


Passwords Must Match

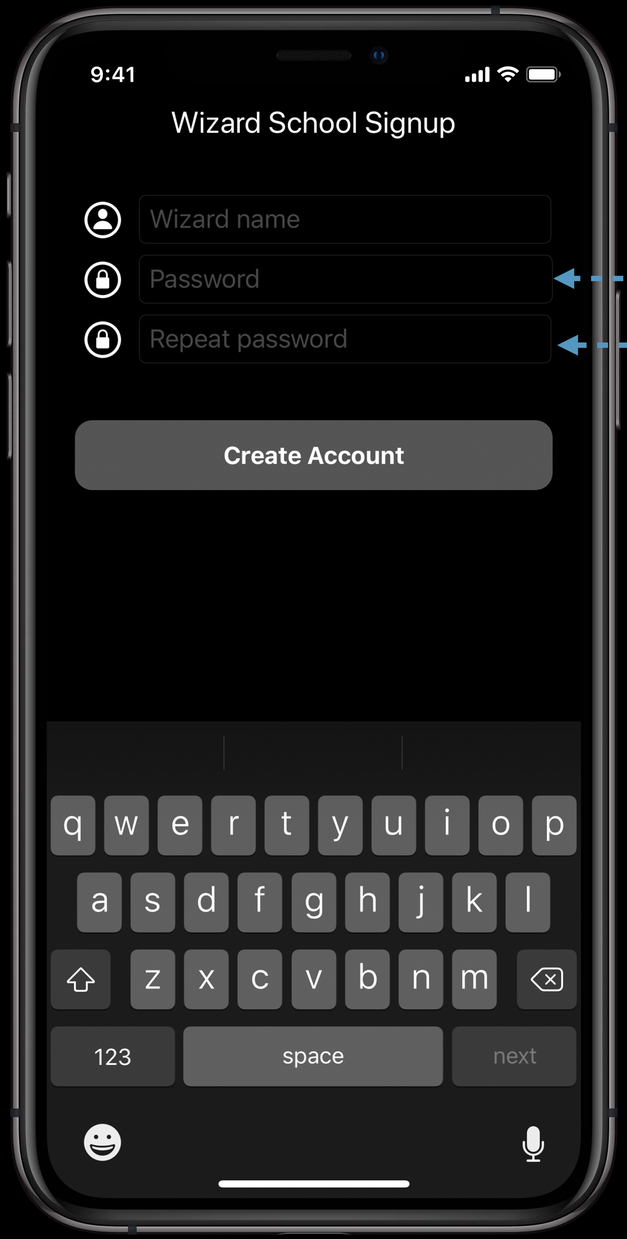
> 8 characters


User name is valid according to server
Enabled if username and passwords valid
9:41
Wizard School Signup
Wizard name
Password
Repeat password
Create Account
Passwords Must Match
> 8 characters
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
123
space
next


9:41
Wizard School Signup
Wizard name
Password
Repeat password
Create Account
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
123
space
next




```
@IBAction func passwordChanged(_ sender:
UITextField)

@IBAction func passwordAgainChanged(_ sender:
UITextField)
```




```
@IBAction func passwordChanged(_ sender:
UITextField)

@IBAction func passwordAgainChanged(_ sender:
UITextField)
```

```
var password: String = ""
@IBAction func passwordChanged(_ sender: UITextField) {
    password = sender.text ?? ""
}

var passwordAgain: String = ""
@IBAction func passwordAgainChanged(_ sender: UITextField) {
    passwordAgain = sender.text ?? ""
}
```

```
var password: String = ""
@IBAction func passwordChanged(_ sender: UITextField) {
    password = sender.text ?? ""
}

var passwordAgain: String = ""
@IBAction func passwordAgainChanged(_ sender: UITextField) {
    passwordAgain = sender.text ?? ""
}
```

```swift
var password: String = ""
@IBAction func passwordChanged(_ sender: UITextField) {
    password = sender.text ?? ""
}

var passwordAgain: String = ""
@IBAction func passwordAgainChanged(_ sender: UITextField) {
    passwordAgain = sender.text ?? ""
}
```

```
@Published var password: String = ""
@IBAction func passwordChanged(_ sender: UITextField) {
    password = sender.text ?? ""
}

@Published var passwordAgain: String = ""
@IBAction func passwordAgainChanged(_ sender: UITextField) {
    passwordAgain = sender.text ?? ""
}
```

`@Published`

Property wrapper

Adds a publisher to any property

```
// Using @Published

@Published var password: String = ""

self.password = "1234"

let currentPassword: String = self.password

let printerSubscription = $password.sink {
   print("The published value is '\($0)'")
}

self.password = "password"
```

```
// Using @Published

@Published var password: String = ""

self.password = "1234"

let currentPassword: String = self.password

let printerSubscription = $password.sink {
   print("The published value is '\($0)'")
}

self.password = "password"
```

```
// Using @Published

@Published var password: String = ""

self.password = "1234"

let currentPassword: String = self.password

let printerSubscription = $password.sink {
   print("The published value is '\($0)'")
}

self.password = "password"
```

```swift
// Using @Published

@Published var password: String = ""

self.password = "1234"

let currentPassword: String = self.password                    // 1234

let printerSubscription = $password.sink {
   print("The published value is '\($0)'")
}

self.password = "password"
```

```
// Using @Published

@Published var password: String = ""

self.password = "1234"

let currentPassword: String = self.password

let printerSubscription = $password.sink {
   print("The published value is '\($0)'")
}

self.password = "password"
```

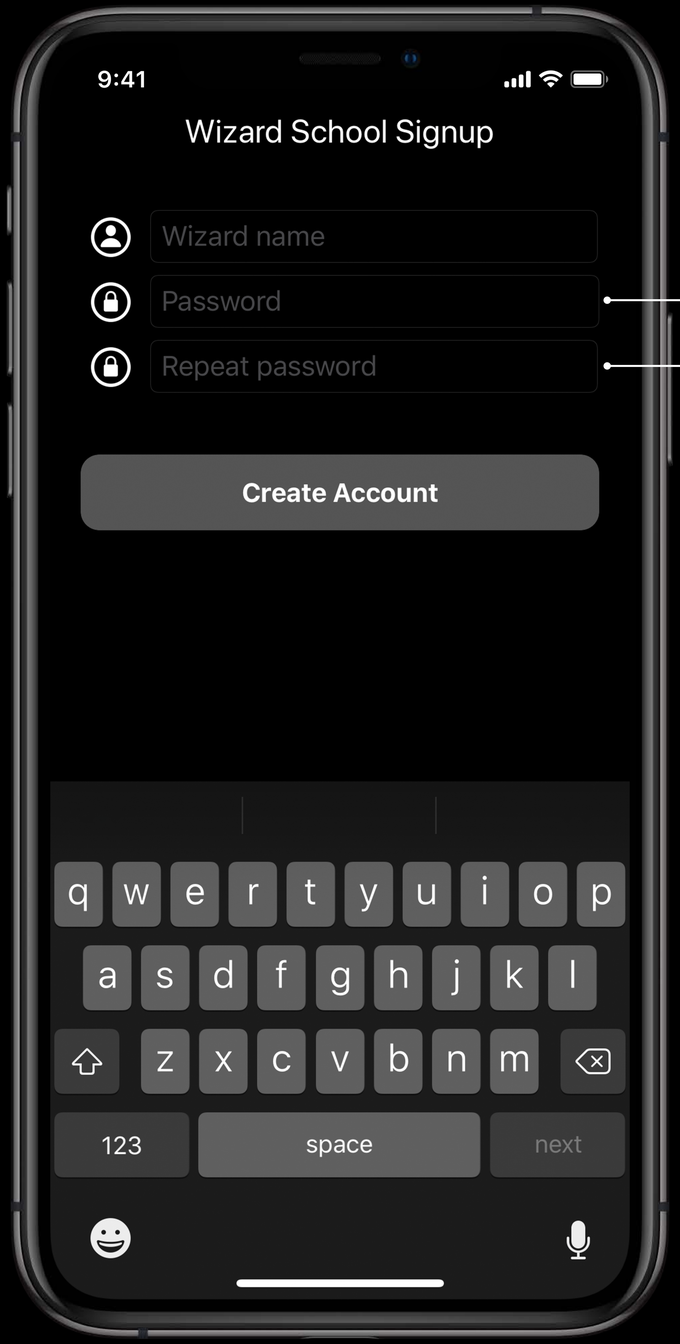
```
// Using @Published

@Published var password: String = ""

self.password = "1234"

let currentPassword: String = self.password

let printerSubscription = $password.sink {
   print("The published value is '\($0)'")
}

self.password = "password"
```

```
// Using @Published

@Published var password: String = ""

self.password = "1234"

let currentPassword: String = self.password

let printerSubscription = $password.sink {
   print("The published value is '\($0)'")
}

self.password = "password"
```

```
// Using @Published

@Published var password: String = ""

self.password = "1234"

let currentPassword: String = self.password

let printerSubscription = $password.sink {
    print("The published value is '\($0)'")
}

self.password = "password"
```

```swift
// Using @Published

@Published var password: String = ""

self.password = "1234"

let currentPassword: String = self.password

let printerSubscription = $password.sink {
    print("The published value is '\($0)'")
}

self.password = "password"
```

```
The published value is 'password'
```

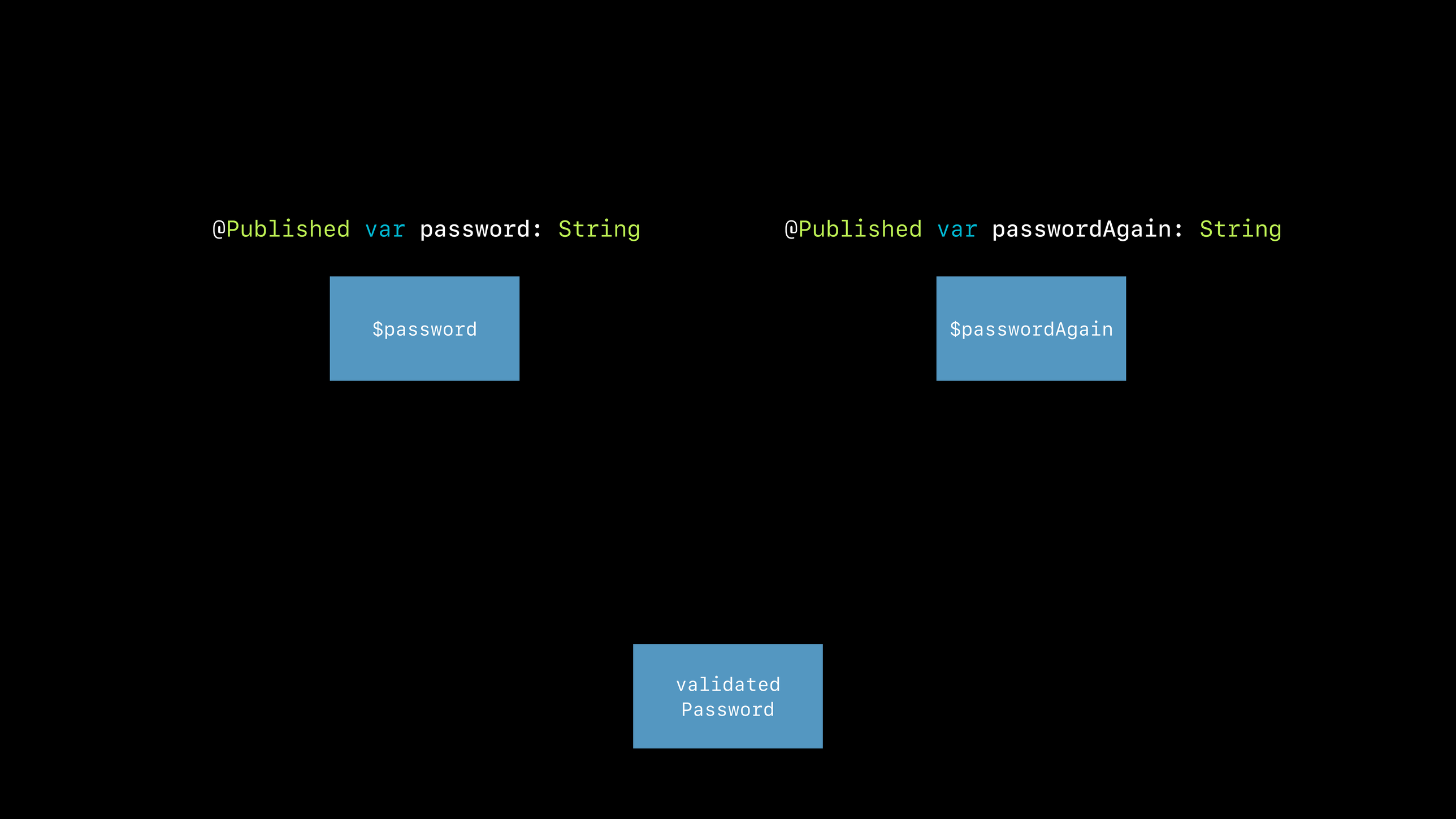


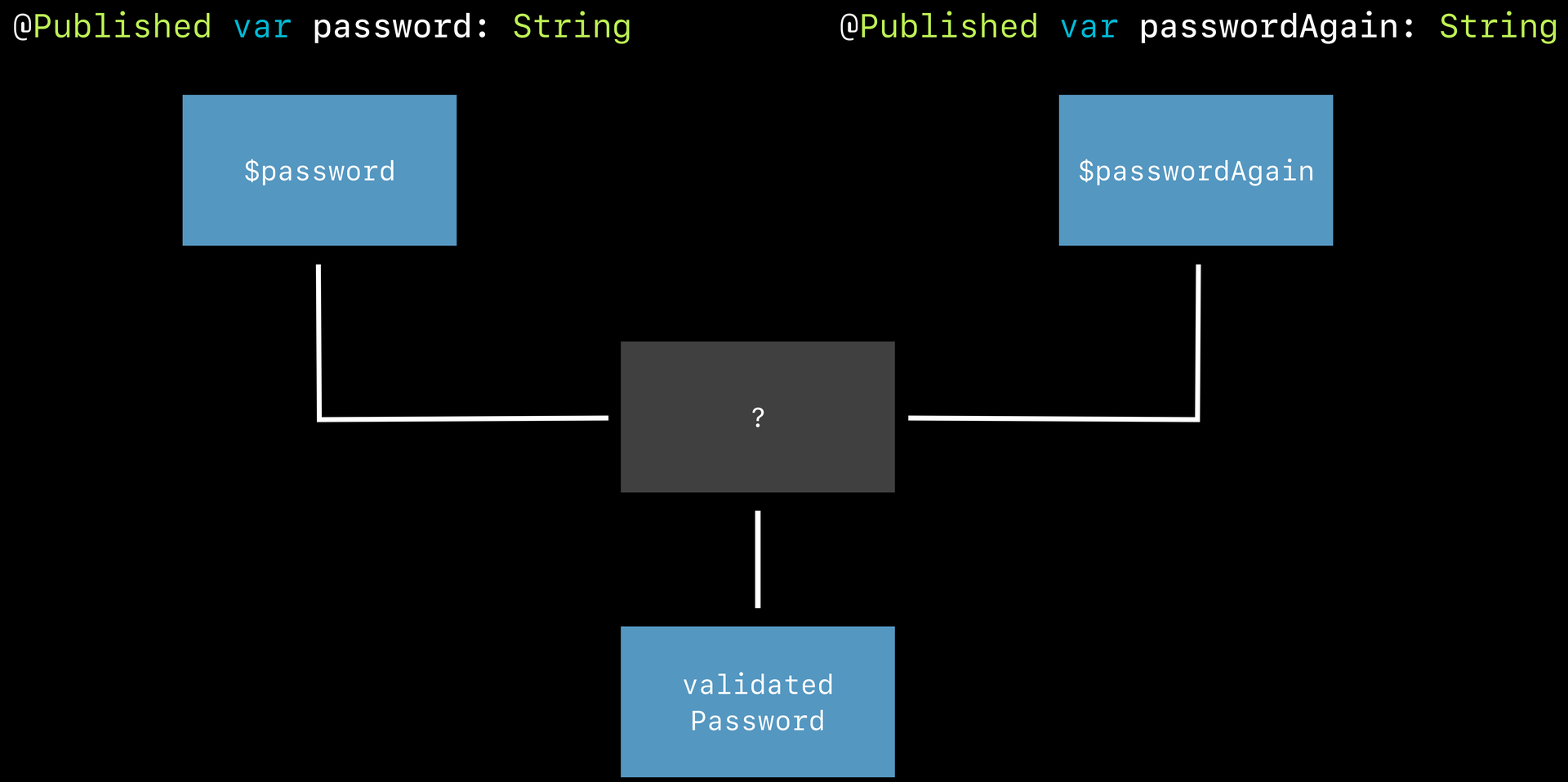
Passwords must match

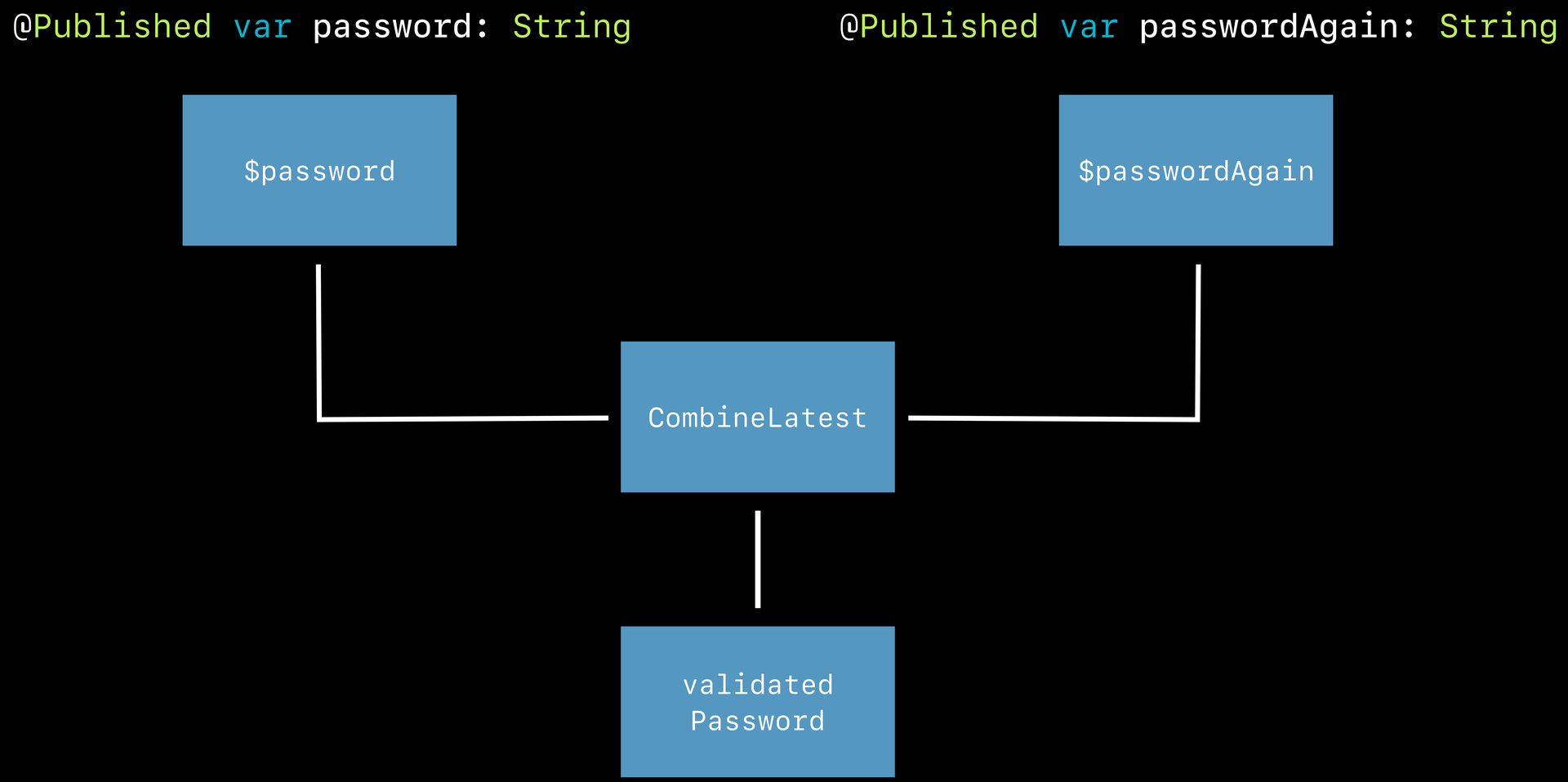
> 8 characters

```swift
@Published var password: String
```


```swift
@Published var passwordAgain: String
```


@Published var password: String
@Published var passwordAgain: String
$password
$passwordAgain
validated
Password


@Published var password: String
@Published var passwordAgain: String
$password
$passwordAgain
?
validated
Password


@Published var password: String
@Published var passwordAgain: String
$password
$passwordAgain
CombineLatest
validated
Password

```
@Published var password: String = ""
@Published var passwordAgain: String = ""

var validatedPassword: CombineLatest<Published<String>, Published<String>, String?> {
    return CombineLatest($password, $passwordAgain) { password, passwordAgain in
        guard password == passwordAgain, password.count > 8 else { return nil }
        return password
    }
}
```

```
@Published var password: String = ""
@Published var passwordAgain: String = ""

var validatedPassword: CombineLatest<Published<String>, Published<String>, String?> {
    return CombineLatest($password, $passwordAgain) { password, passwordAgain in
        guard password == passwordAgain, password.count > 8 else { return nil }
        return password
    }
}
```

```
@Published var password: String = ""
@Published var passwordAgain: String = ""

var validatedPassword: CombineLatest<Published<String>, Published<String>, String?> {
    return CombineLatest($password, $passwordAgain) { password, passwordAgain in
        guard password == passwordAgain, password.count > 8 else { return nil }
        return password
    }
}
```

```
@Published var password: String = ""
@Published var passwordAgain: String = ""

var validatedPassword: CombineLatest<Published<String>, Published<String>, String?> {
    return CombineLatest($password, $passwordAgain) { password, passwordAgain in
        guard password == passwordAgain, password.count > 8 else { return nil }
        return password
    }
}
```

```
@Published var password: String = ""
@Published var passwordAgain: String = ""

var validatedPassword: CombineLatest<Published<String>, Published<String>, String?> {
    return CombineLatest($password, $passwordAgain) { password, passwordAgain in
        guard password == passwordAgain, password.count > 8 else { return nil }
        return password
    }
}
```

```
@Published var password: String = ""
@Published var passwordAgain: String = ""

var validatedPassword: CombineLatest<Published<String>, Published<String>, String?> {
    return CombineLatest($password, $passwordAgain) { password, passwordAgain in
        guard password == passwordAgain, password.count > 8 else { return nil }
        return password
    }
}
```

String?

Never

```
@Published var password: String = ""
@Published var passwordAgain: String = ""

var validatedPassword: Map<CombineLatest<Published<String>, Published<String>, String?>> {
    return CombineLatest($password, $passwordAgain) { password, passwordAgain in
        guard password == passwordAgain, password.count > 8 else { return nil }
        return password
    }
    .map { $0 == "password1" ? nil : $0 }
}
```

```
@Published var password: String = ""
@Published var passwordAgain: String = ""

var validatedPassword: Map<CombineLatest<Published<String>, Published<String>, String?>> {
    return CombineLatest($password, $passwordAgain) { password, passwordAgain in
        guard password == passwordAgain, password.count > 8 else { return nil }
        return password
    }
    .map { $0 == "password1" ? nil : $0 }
}
```

String?

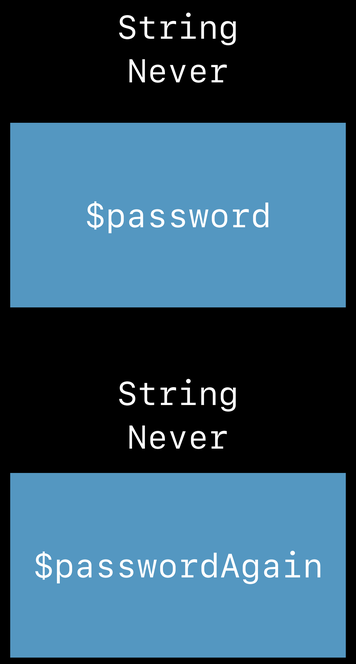
Never

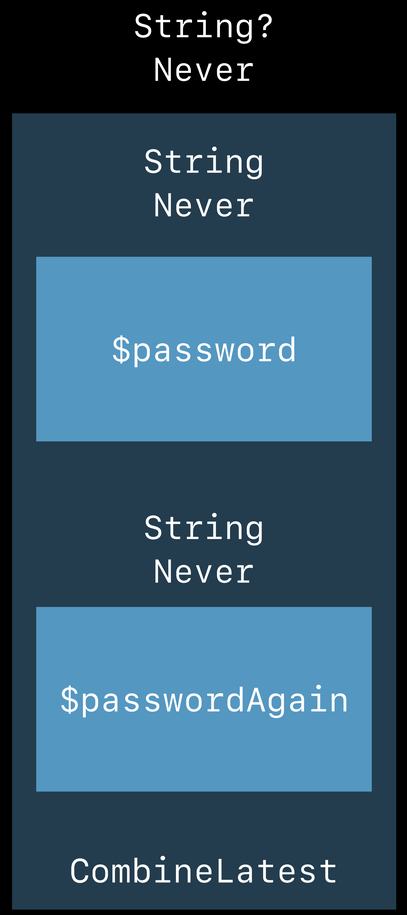
```
@Published var password: String = ""
@Published var passwordAgain: String = ""

var validatedPassword: AnyPublisher<String?, Never> {
    return CombineLatest($password, $passwordAgain) { password, passwordAgain in
        guard password == passwordAgain, password.count > 8 else { return nil }
        return password
    }
    .map { $0 == "password1" ? nil : $0 }
    .eraseToAnyPublisher()
}
```

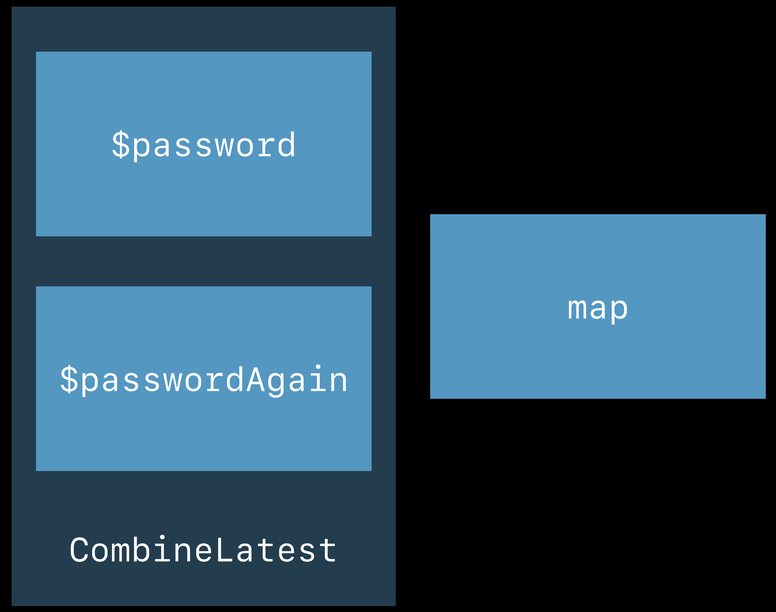
```swift
@Published var password: String = ""
@Published var passwordAgain: String = ""

var validatedPassword: AnyPublisher<String?, Never> {
    return CombineLatest($password, $passwordAgain) { password, passwordAgain in
        guard password == passwordAgain, password.count > 8 else { return nil }
        return password
    }
    .map { $0 == "password1" ? nil : $0 }
    .eraseToAnyPublisher()
}
```

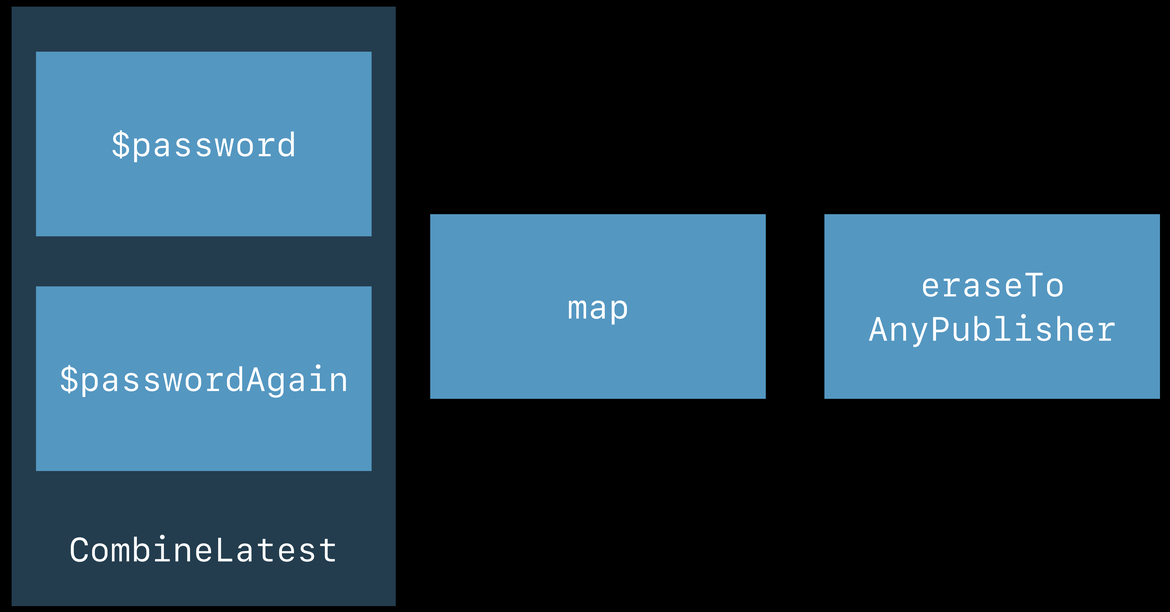
String?

Never


String
Never
$password
String
Never
$passwordAgain


String?
Never
String
Never
$password
String
Never
$passwordAgain
CombineLatest


$password
$passwordAgain
CombineLatest
map

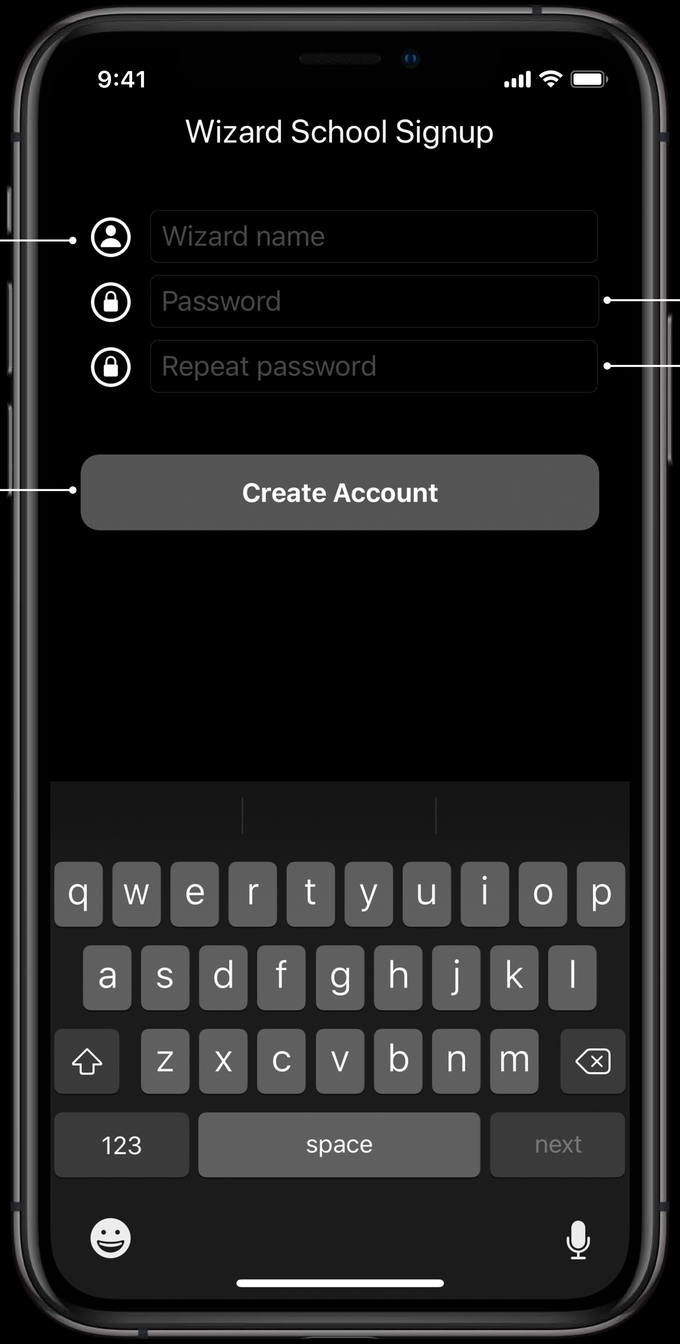

$password
$passwordAgain
CombineLatest
map
eraseTo
AnyPublisher


eraseTo
AnyPublisher


```
var validatedPassword: AnyPublisher<String?, Never>
```

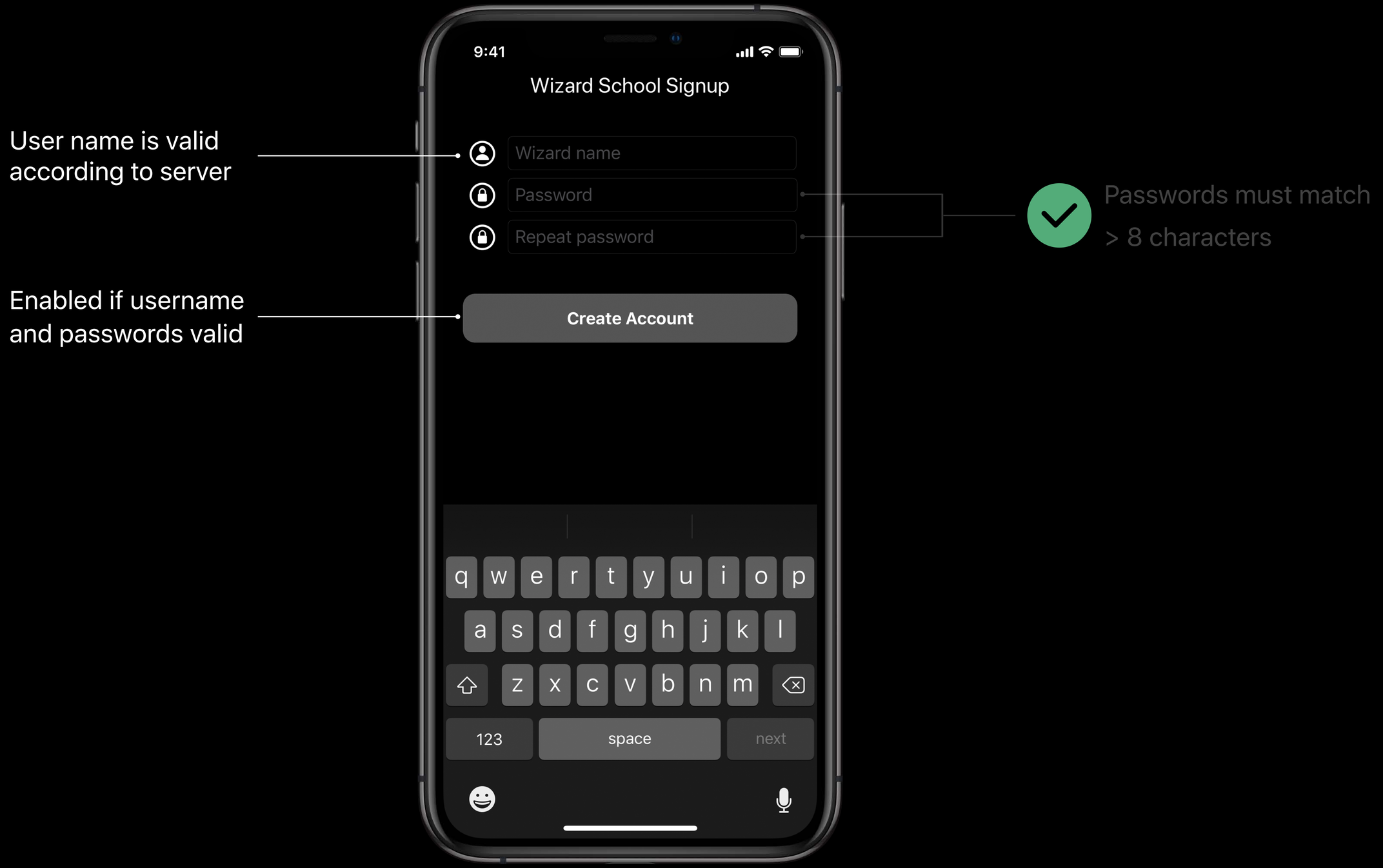
User name is valid according to server

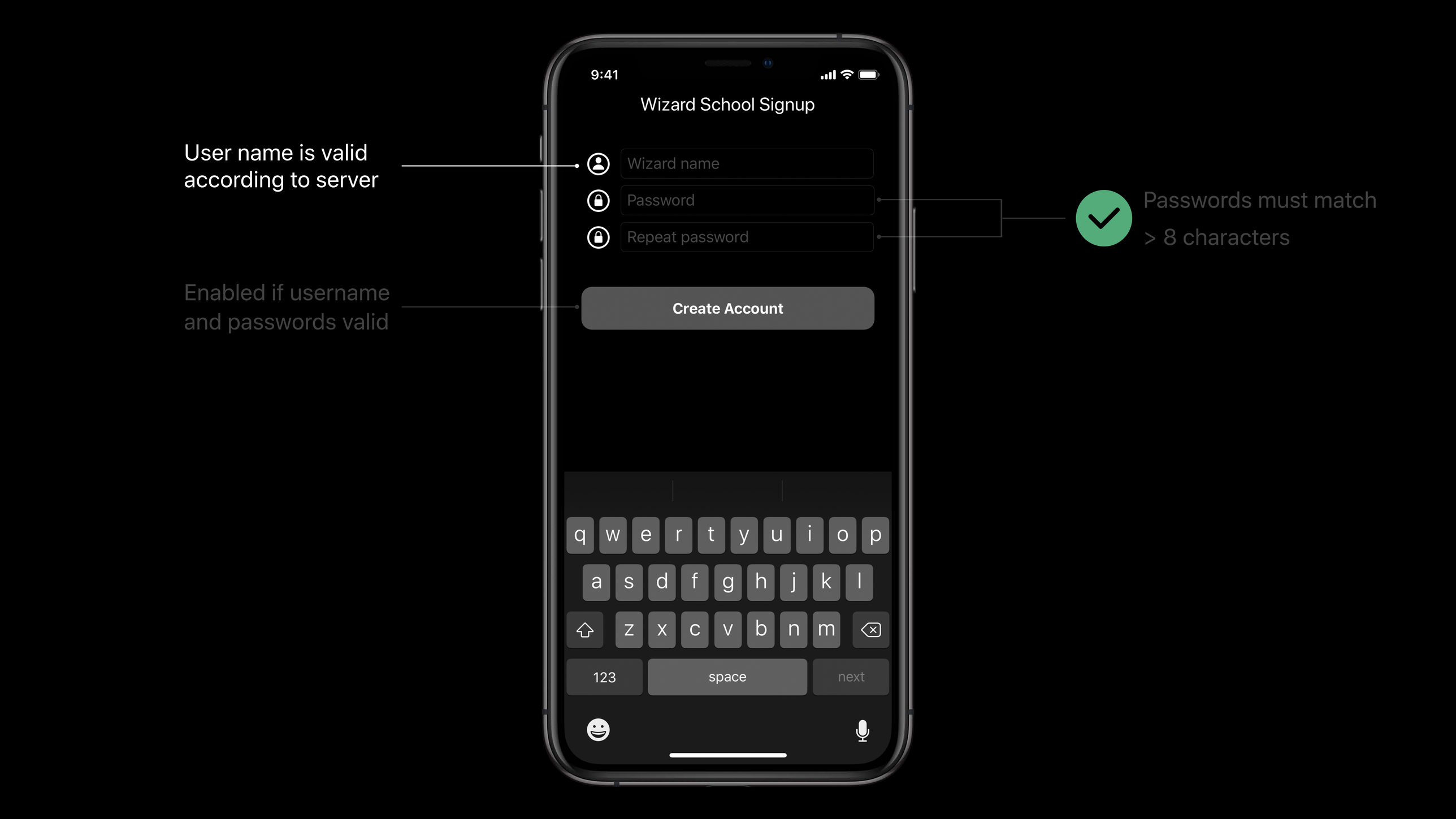
Enabled if username and passwords valid



Passwords must match

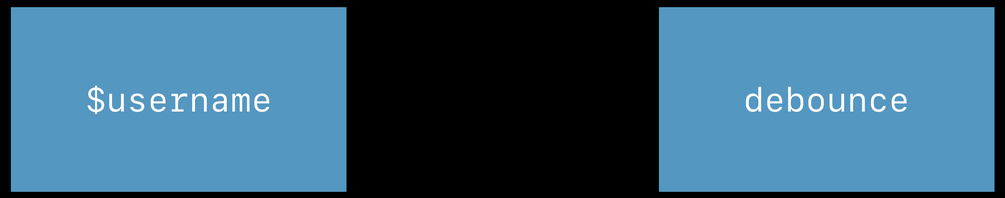
> 8 characters


User name is valid according to server
Enabled if username and passwords valid
9:41
Wizard School Signup
Wizard name
Password
Repeat password
Create Account
Passwords must match
> 8 characters
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
123
space
next


User name is valid according to server
Enabled if username and passwords valid
9:41
Wizard School Signup
Wizard name
Password
Repeat password
Create Account
Passwords must match
> 8 characters
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
123
space
next

```
@Published var username: String
```

```
@Published var username: String
```

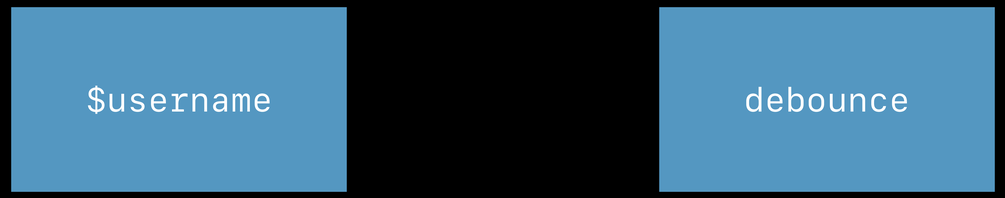


Debounce

Debounce

Debounce

```
@Published var username: String
```



```
@Published var username: String
```

```
@Published var username: String = ""

var validatedUsername: AnyPublisher<String, Never> {
    return $username
        .debounce(for: 0.5, scheduler: RunLoop.main)
        .removeDuplicates()
        .eraseToAnyPublisher()
}
```

```swift
@Published var username: String = ""

var validatedUsername: AnyPublisher<String, Never> {
    return $username
        .debounce(for: 0.5, scheduler: RunLoop.main)
        .removeDuplicates()
        .eraseToAnyPublisher()
}
```

```swift
@Published var username: String = ""

var validatedUsername: AnyPublisher<String, Never> {
    return $username
        .debounce(for: 0.5, scheduler: RunLoop.main)
        .removeDuplicates()
        .eraseToAnyPublisher()
}
```

```swift
@Published var username: String = ""

var validatedUsername: AnyPublisher<String, Never> {
    return $username
        .debounce(for: 0.5, scheduler: RunLoop.main)
        .removeDuplicates()
        .eraseToAnyPublisher()
}
```

String

Never

```swift
@Published var username: String = ""

var validatedUsername: AnyPublisher<String, Never> {
    return $username
        .debounce(for: 0.5, scheduler: RunLoop.main)
        .removeDuplicates()
        .eraseToAnyPublisher()
}

// func usernameAvailable(_ username: String, completion: (Bool) -> Void)
```

```swift
@Published var username: String = ""

var validatedUsername: AnyPublisher<String, Never> {
    return $username
        .debounce(for: 0.5, scheduler: RunLoop.main)
        .removeDuplicates()
        .flatMap { username in

            // func usernameAvailable(_ username: String, completion: (Bool) -> Void)




        .eraseToAnyPublisher()
}
```

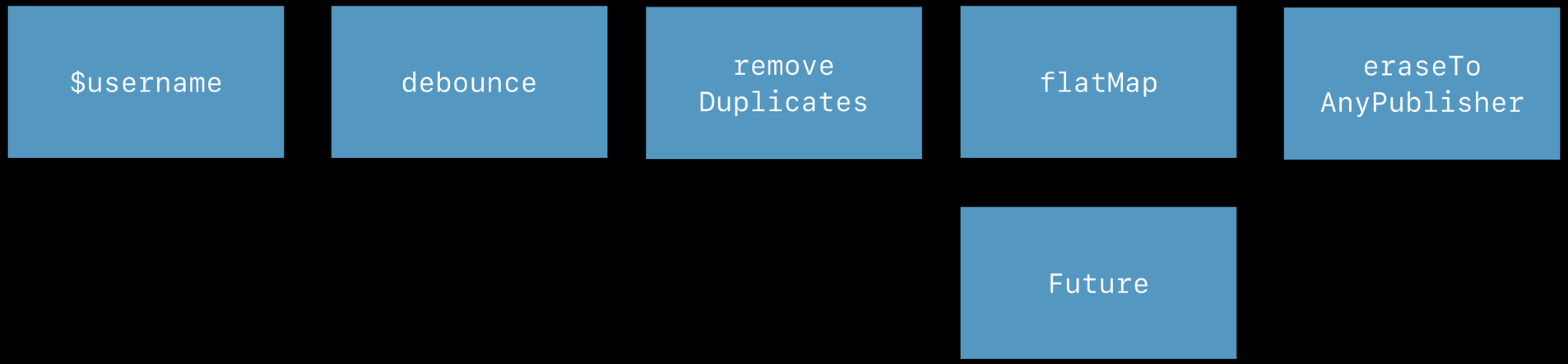
```swift
@Published var username: String = ""

var validatedUsername: AnyPublisher<String, Never> {
    return $username
        .debounce(for: 0.5, scheduler: RunLoop.main)
        .removeDuplicates()
        .flatMap { username in

            // func usernameAvailable(_ username: String, completion: (Bool) -> Void)



        }
        .eraseToAnyPublisher()
}
```

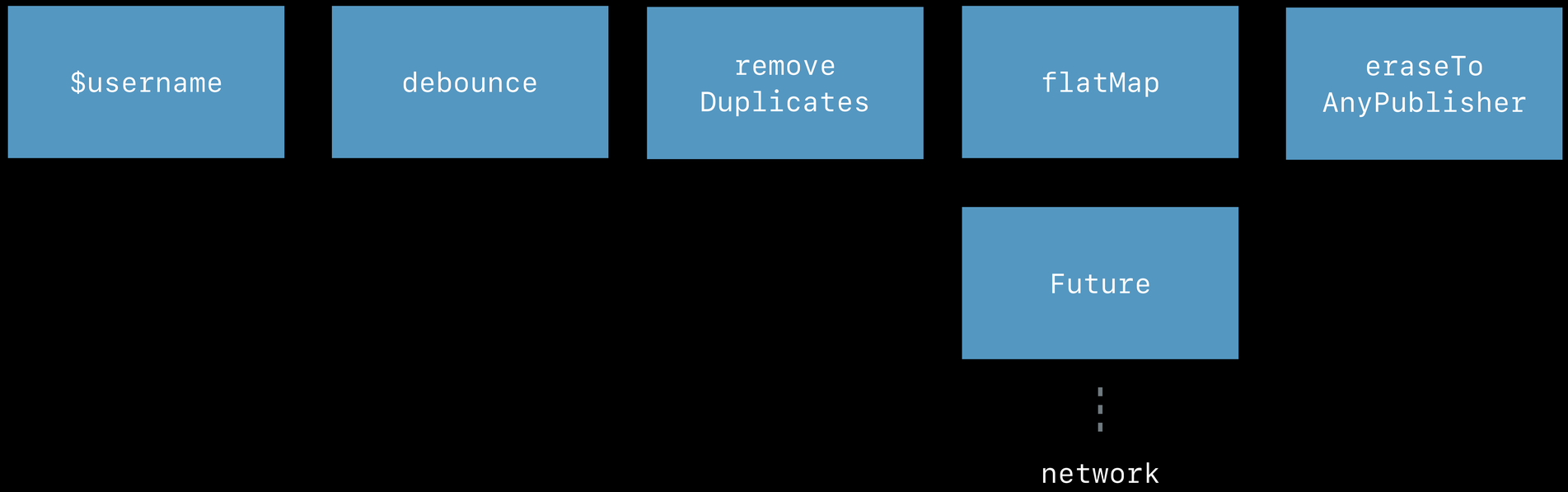
```
@Published var username: String = ""

var validatedUsername: AnyPublisher<String, Never> {
    return $username
        .debounce(for: 0.5, scheduler: RunLoop.main)
        .removeDuplicates()
        .flatMap { username in
            return Future { promise in



            }
        }
        .eraseToAnyPublisher()
}
```

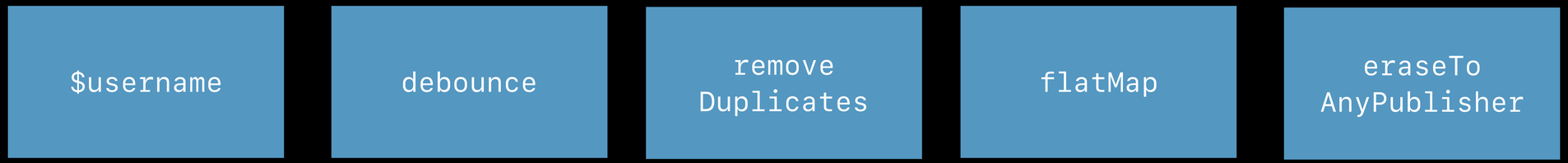
```swift
@Published var username: String = ""

var validatedUsername: AnyPublisher<String, Never> {
    return $username
        .debounce(for: 0.5, scheduler: RunLoop.main)
        .removeDuplicates()
        .flatMap { username in
            return Future { promise in

                (Result<Output, Failure>) -> Void


            }
        }
        .eraseToAnyPublisher()
}
```

```
@Published var username: String = ""

var validatedUsername: AnyPublisher<String?, Never> {
    return $username
        .debounce(for: 0.5, scheduler: RunLoop.main)
        .removeDuplicates()
        .flatMap { username in
            return Future { promise in
                self.usernameAvailable(username) { available in
                    promise(.success(available ? username : nil))
                }
            }
        }
        .eraseToAnyPublisher()
}
```


$username
debounce
remove
Duplicates


$username
debounce
remove Duplicates
flatMap
eraseTo AnyPublisher
Future


$username
debounce
remove Duplicates
flatMap
eraseTo AnyPublisher
Future
network


$username
debounce
remove Duplicates
flatMap
eraseTo AnyPublisher

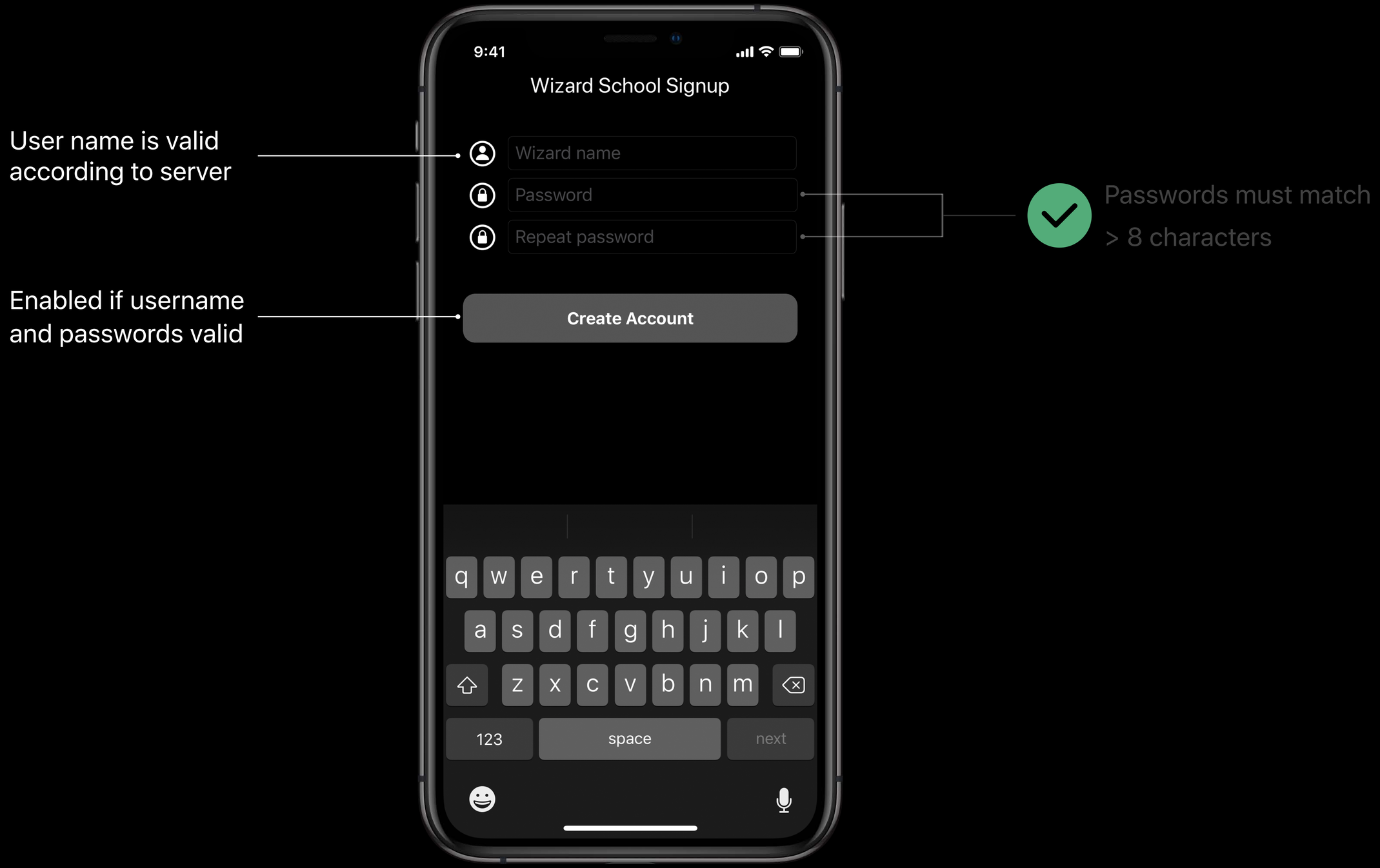

eraseTo
AnyPublisher

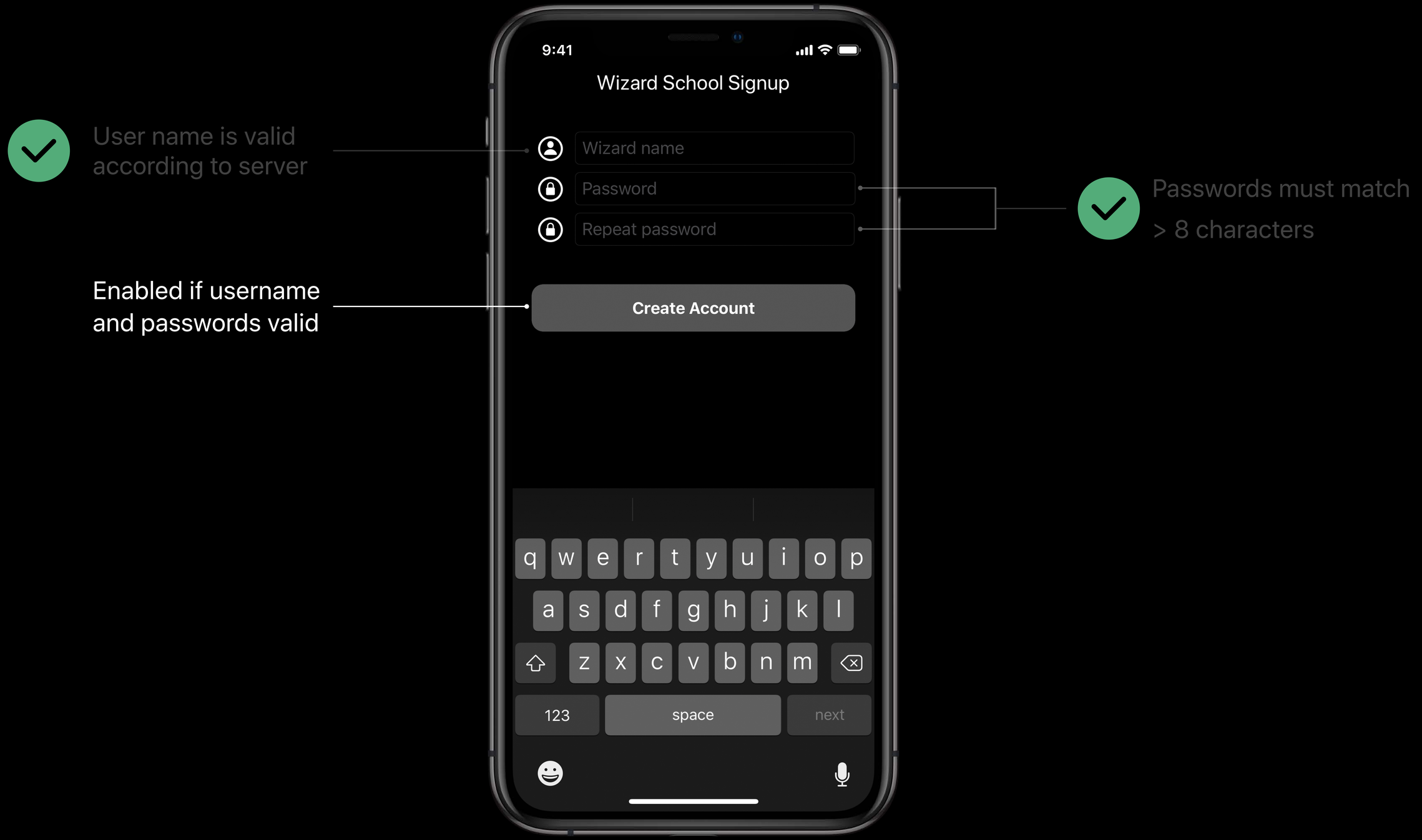

```
var validatedUsername: AnyPublisher<String?, Never>
```


```
var validatedUsername: AnyPublisher<String?, Never>
```


```
var validatedPassword: AnyPublisher<String?, Never>
```


```
var validatedUsername: AnyPublisher<String?, Never>
```


User name is valid according to server
Enabled if username and passwords valid
9:41
Wizard School Signup
Wizard name
Password
Repeat password
Create Account
Passwords must match
> 8 characters
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
123
space
next


User name is valid according to server
Enabled if username and passwords valid
9:41
Wizard School Signup
Wizard name
Password
Repeat password
Create Account
Passwords must match
> 8 characters
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
123
space
next

```
var validatedCredentials: AnyPublisher<(String, String)?, Never> {
    return CombineLatest(validatedUsername, validatedPassword) { username, password in
        guard let uname = username, let pwd = password else { return nil }
        return (uname, pwd)
    }
    .eraseToAnyPublisher()
}
```

```
var validatedCredentials: AnyPublisher<(String, String)?, Never> {
    return CombineLatest(validatedUsername, validatedPassword) { username, password in
        guard let uname = username, let pwd = password else { return nil }
        return (uname, pwd)
    }
    .eraseToAnyPublisher()
}
```

(String, String)?

Never

```
@IBOutlet var signupButton: UIButton!

var signupButtonStream: AnyCancellable?

override func viewDidLoad() {
    super.viewDidLoad()

    self.signupButtonStream = self.validatedCredentials
        .map { $0 != nil }
        .receive(on: RunLoop.main)
        .assign(to: \.isEnabled, on: signupButton)
}
```

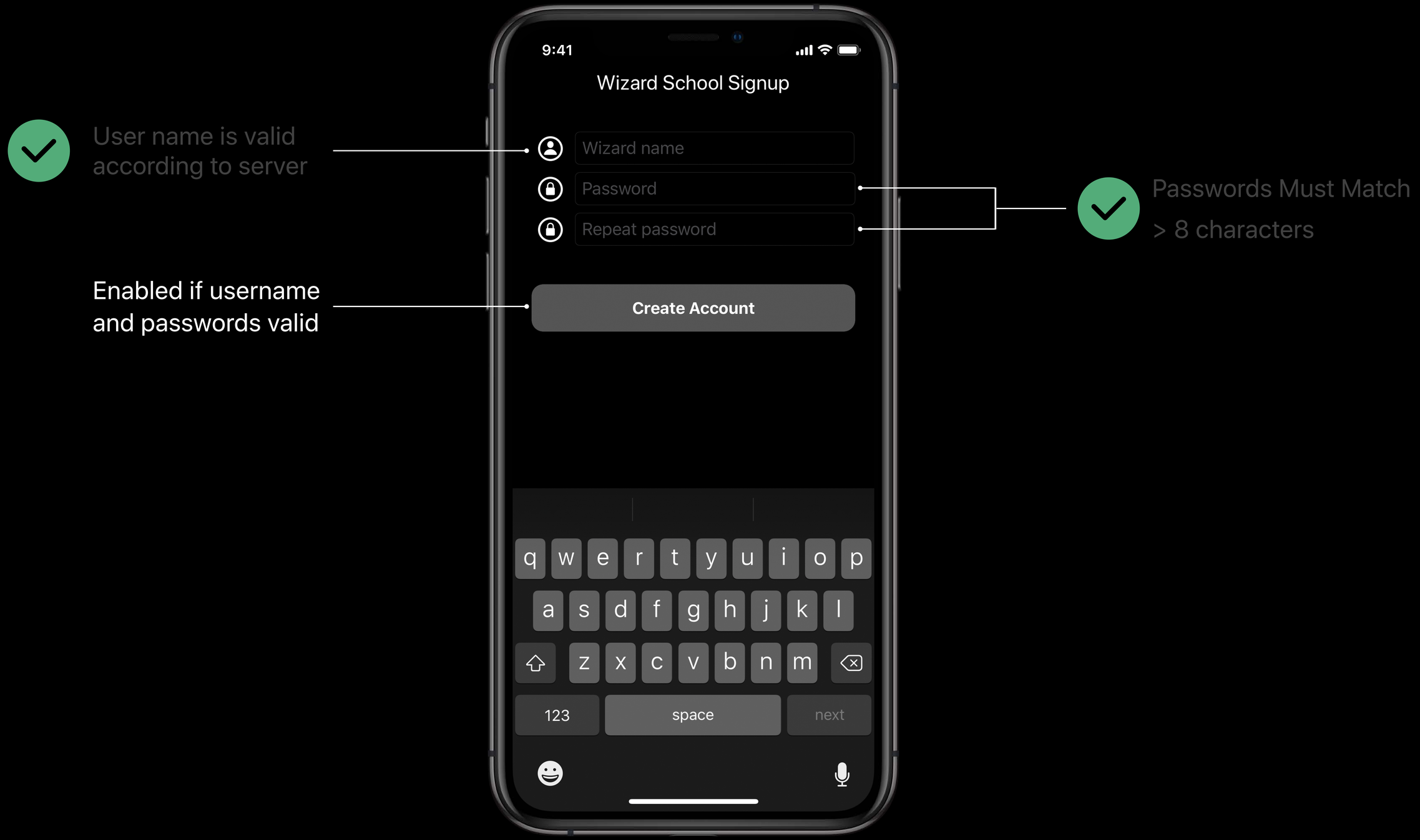
```
@IBOutlet var signupButton: UIButton!

var signupButtonStream: AnyCancellable?

override func viewDidLoad() {
    super.viewDidLoad()

    self.signupButtonStream = self.validatedCredentials
        .map { $0 != nil }
        .receive(on: RunLoop.main)
        .assign(to: \.isEnabled, on: signupButton)
}
```

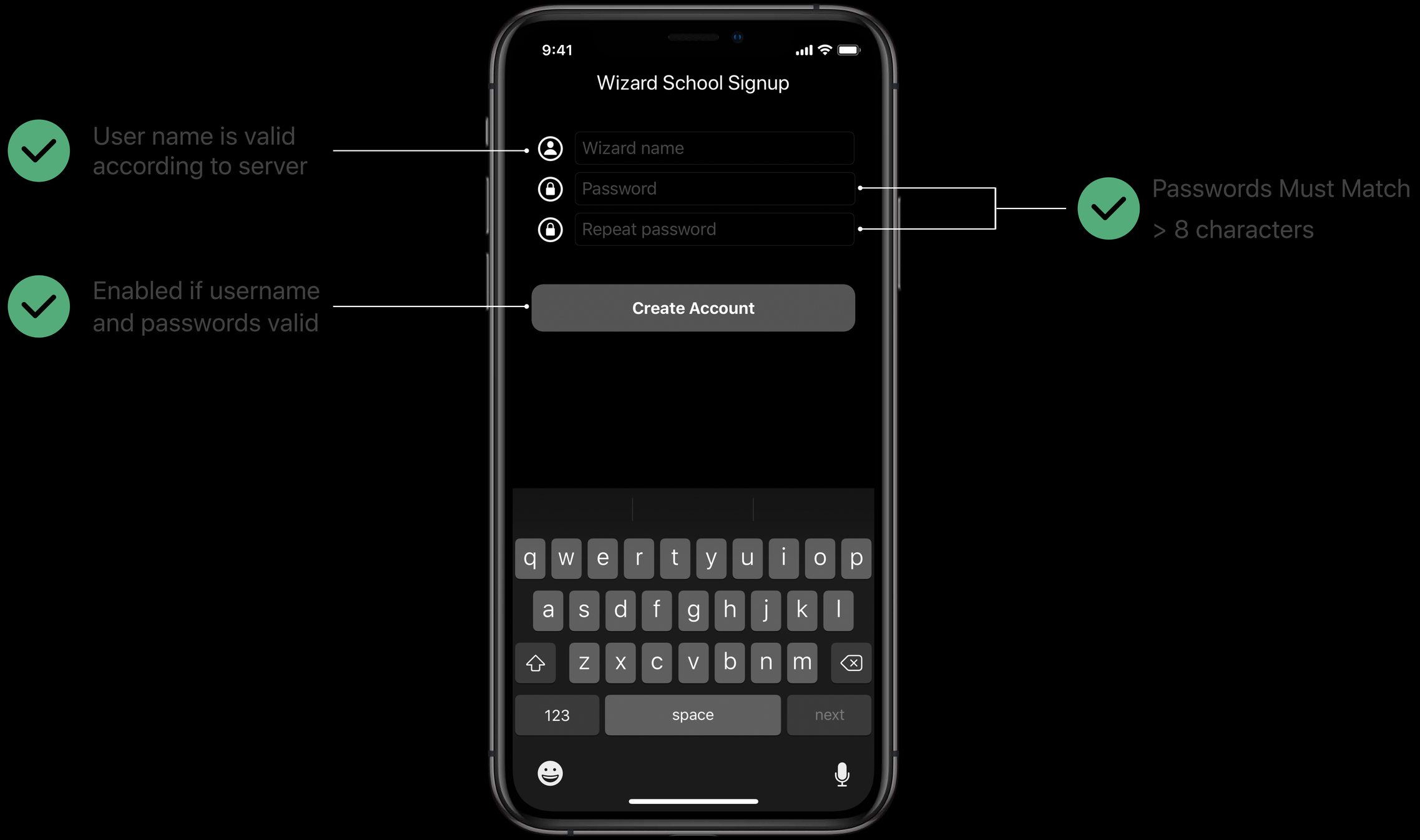
```
@IBOutlet var signupButton: UIButton!

var signupButtonStream: AnyCancellable?

override func viewDidLoad() {
    super.viewDidLoad()

    self.signupButtonStream = self.validatedCredentials
        .map { $0 != nil }
        .receive(on: RunLoop.main)
        .assign(to: \.isEnabled, on: signupButton)
}
```

```
@IBOutlet var signupButton: UIButton!

var signupButtonStream: AnyCancellable?

override func viewDidLoad() {
    super.viewDidLoad()

    self.signupButtonStream = self.validatedCredentials
        .map { $0 != nil }
        .receive(on: RunLoop.main)
        .assign(to: \.isEnabled, on: signupButton)
}
```

```
@IBOutlet var signupButton: UIButton!

var signupButtonStream: AnyCancellable?

override func viewDidLoad() {
    super.viewDidLoad()

    self.signupButtonStream = self.validatedCredentials
        .map { $0 != nil }
        .receive(on: RunLoop.main)
        .assign(to: \.isEnabled, on: signupButton)
}
```

```
@IBOutlet var signupButton: UIButton!

var signupButtonStream: AnyCancellable?

override func viewDidLoad() {
    super.viewDidLoad()

    self.signupButtonStream = self.validatedCredentials
        .map { $0 != nil }
        .receive(on: RunLoop.main)
        .assign(to: \.isEnabled, on: signupButton)
}
```


User name is valid according to server
Enabled if username and passwords valid
9:41
Wizard School Signup
Wizard name
Password
Repeat password
Create Account
Passwords Must Match
> 8 characters
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
123
space
next

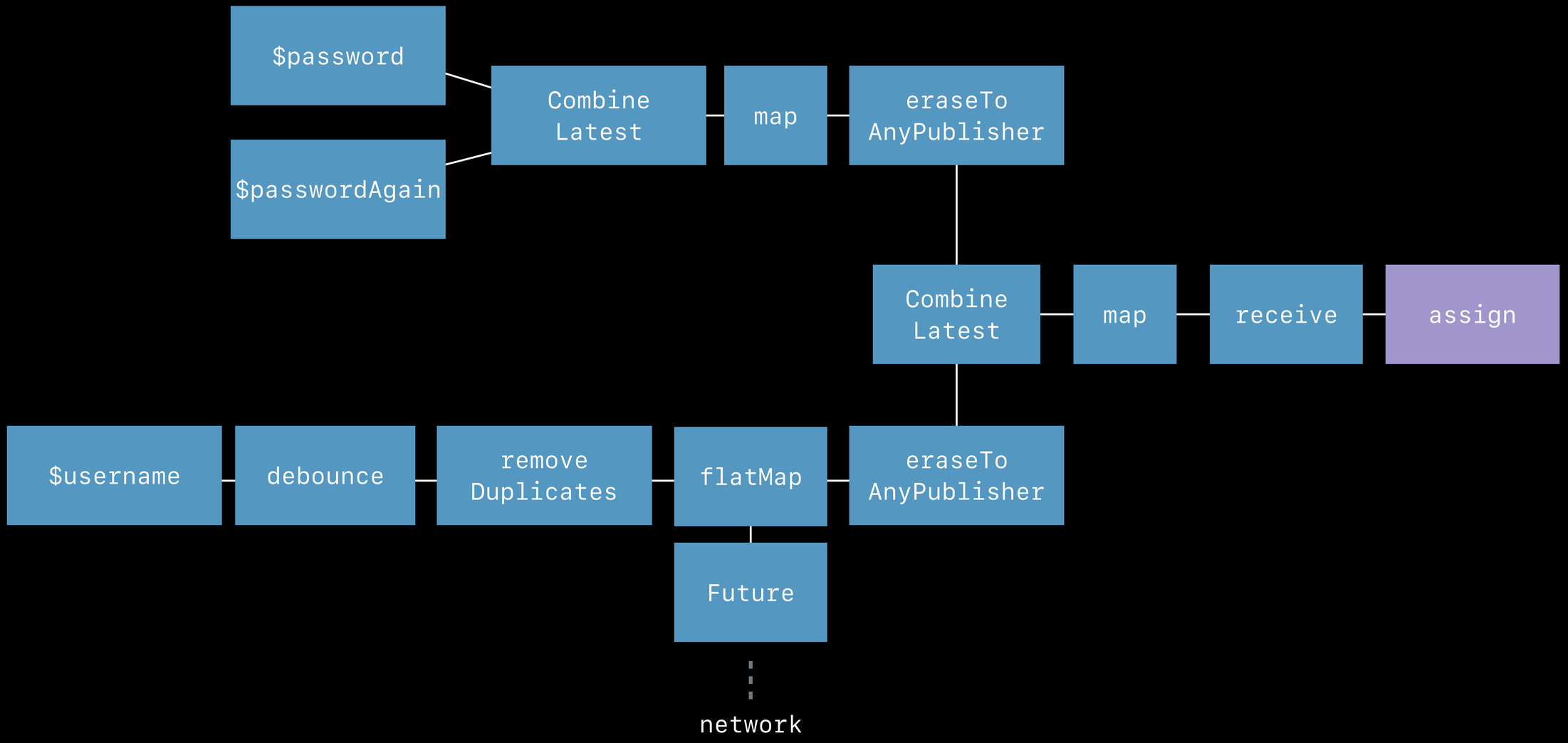

User name is valid according to server
Enabled if username and passwords valid
9:41
Wizard School Signup
Wizard name
Password
Repeat password
Create Account
Passwords Must Match
> 8 characters
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
123
space
next


$password
$passwordAgain
$username

$password

$passwordAgain

$username


$password
$passwordAgain
Combine Latest
map
eraseTo AnyPublisher
Combine Latest
map
receive
assign
$username
debounce
remove Duplicates
flatMap
eraseTo AnyPublisher
Future
network

Use Combine Today

Compose small parts into custom publishers

Adopt incrementally

Add a `Publisher` to a property with `@Published`

Compose callbacks and Publishers with `Future`

More Information

developer.apple.com/wwdc19/721

Introducing Combine	WWDC 2019
Data Flow Through SwiftUI	WWDC 2019


WWDC19