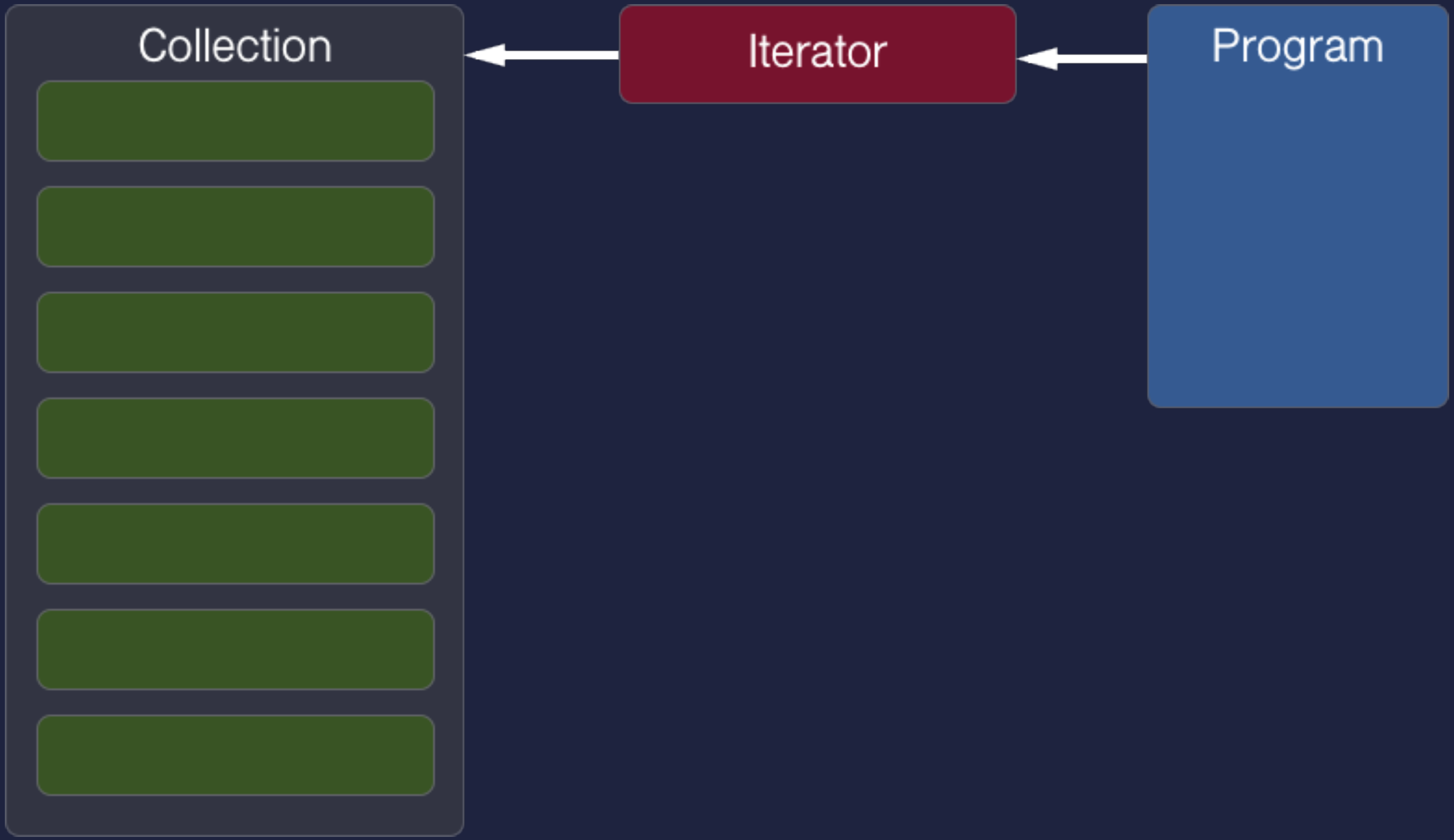
REACTIVE PROGRAMMING WITH RXSWIFT

DEFINITIONS

ITERATOR
VS
OBSERVER

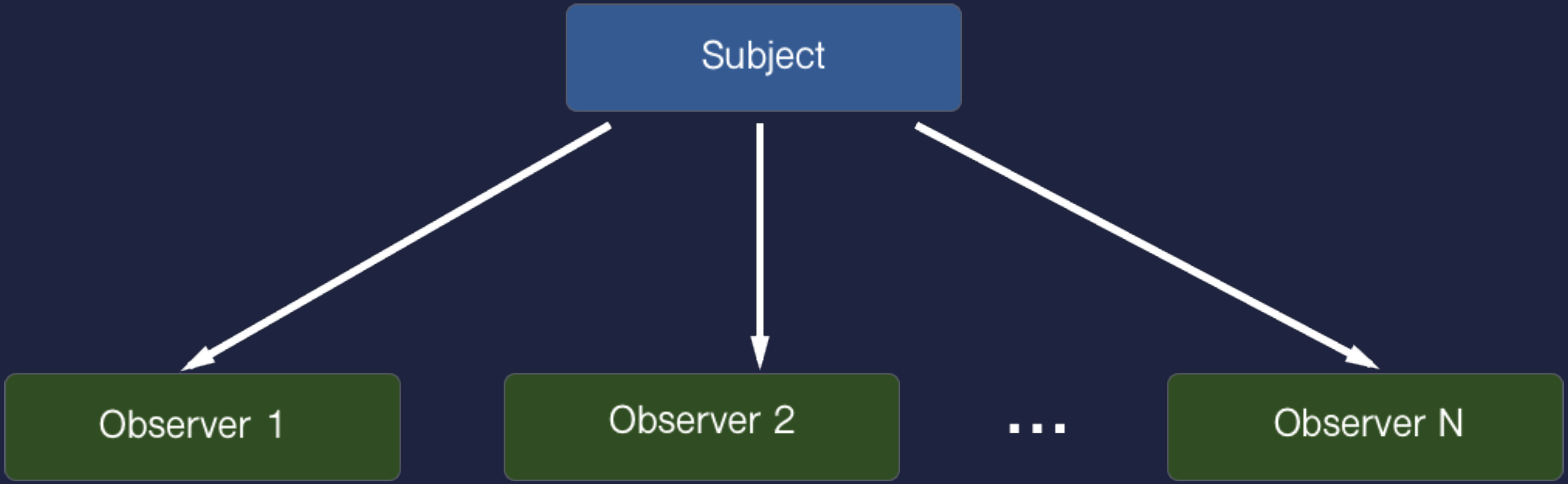
ITERATOR


Collection
Iterator
Program

```
let collection = ["1", "2", "3", "4", "5"]

for element in collection {
    print(element)
}
```

```swift
let collection = ["1", "2", "3", "4", "5"]

var iterator = collection.makeIterator()

while let element = iterator.next() {
    print(element)
}
```

OBSERVER


Subject
Observer 1
Observer 2
...
Observer N

```swift
let someObject = ...
let keyPath = #keyPath("some_property_name")
```

...

```swift
someObject.addObserver(self, forKeyPath: keyPath)
```

...

```swift
override func observeValue(forKeyPath keyPath: String?, of object: Any?,
                           change: [NSKeyValueChangeKey : Any]?,
                           context: UnsafeMutableRawPointer?) {
    // 🎩 MAGIC HAPPENS HERE
}
```

...

```swift
someObject.removeObserver(self, forKeyPath: keyPath)
```

OBSERVABLE SEQUENCE

```swift
var cursorPosition = CGPoint.zero {
    didSet {
        print(newValue)
    }
}

...

{0, 0}, {1, 1}, {1, 2}, {1, 3}, {0, 4}, {1, 5} ...
```

```
{0, 0}, {1, 1}, {1, 2}, {1, 3}, {0, 4}, {1, 5} ...
```

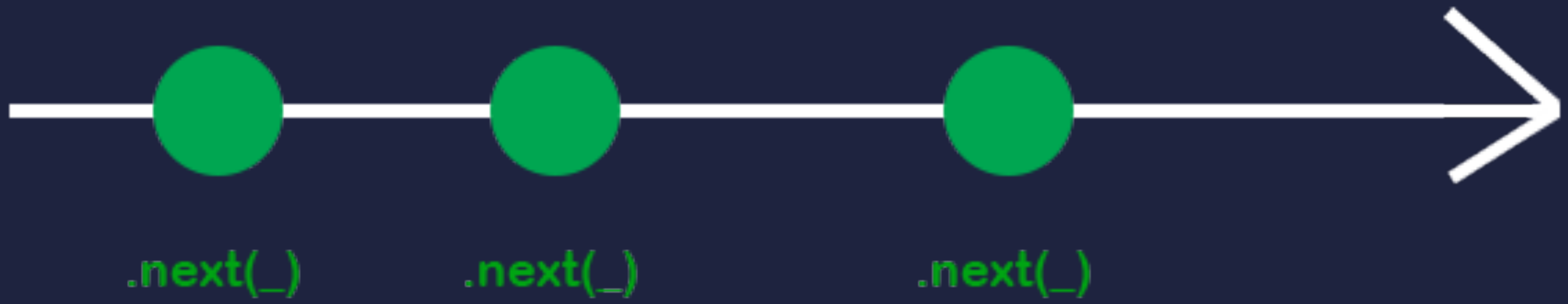
↑

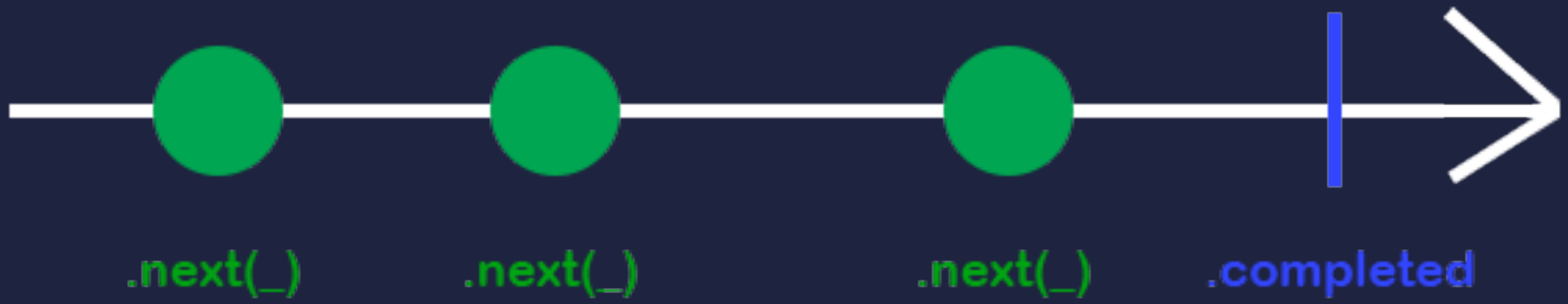
Observable sequence

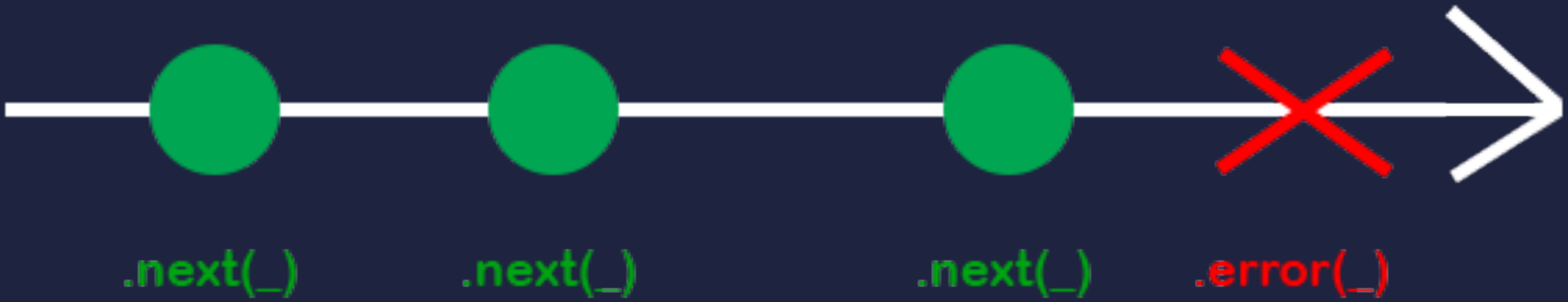
REACTIVE PROGRAMMING

MANAGING OBSERVABLE SEQUENCES

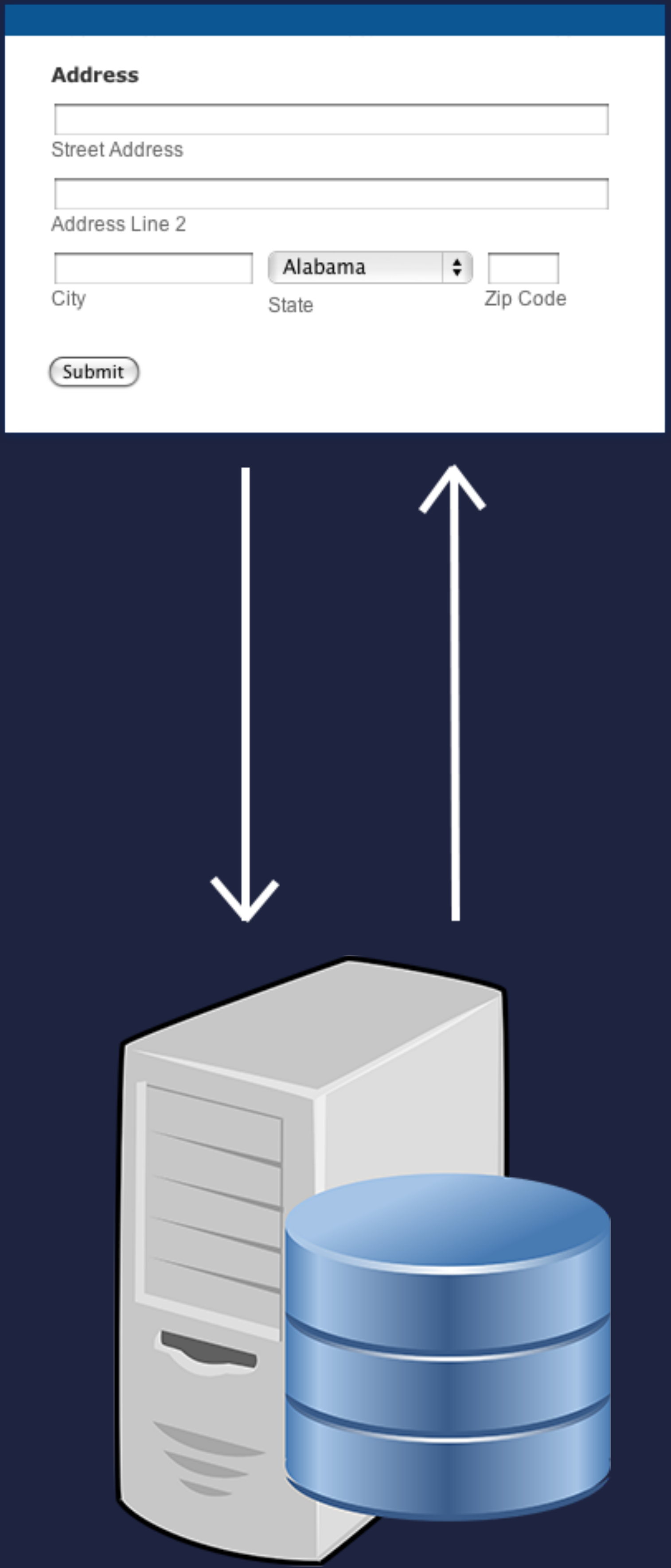
EVENT TYPES


.next(_)
.next(_)
.next(_)


.next(_)
.next(_)
.next(_)
.completed


.next(_)
.next(_)
.next(_)
.error(_)

WHY?



Address

Street Address

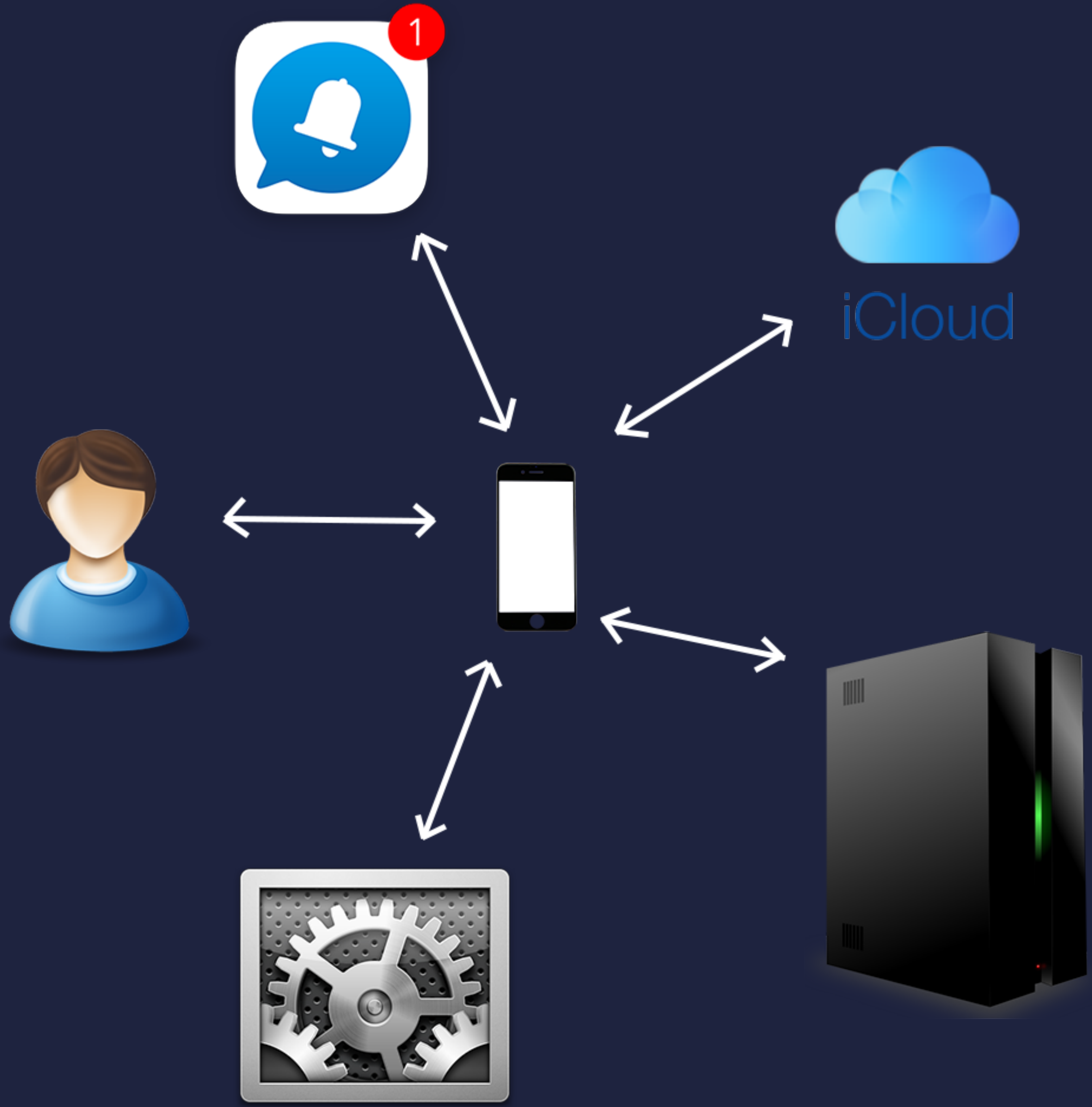
Address Line 2

City

State: Alabama

Zip Code

Submit


1
iCloud

DELEGATES

I HAVE TO CREATE YET ANOTHER
DELEGATE BOILERPLATE CODE
imgflip.com

Instead of doing the tedious and non-expressive

```
public func scrollViewDidScroll(scrollView: UIScrollView) {
    // do something here
}
```

... write

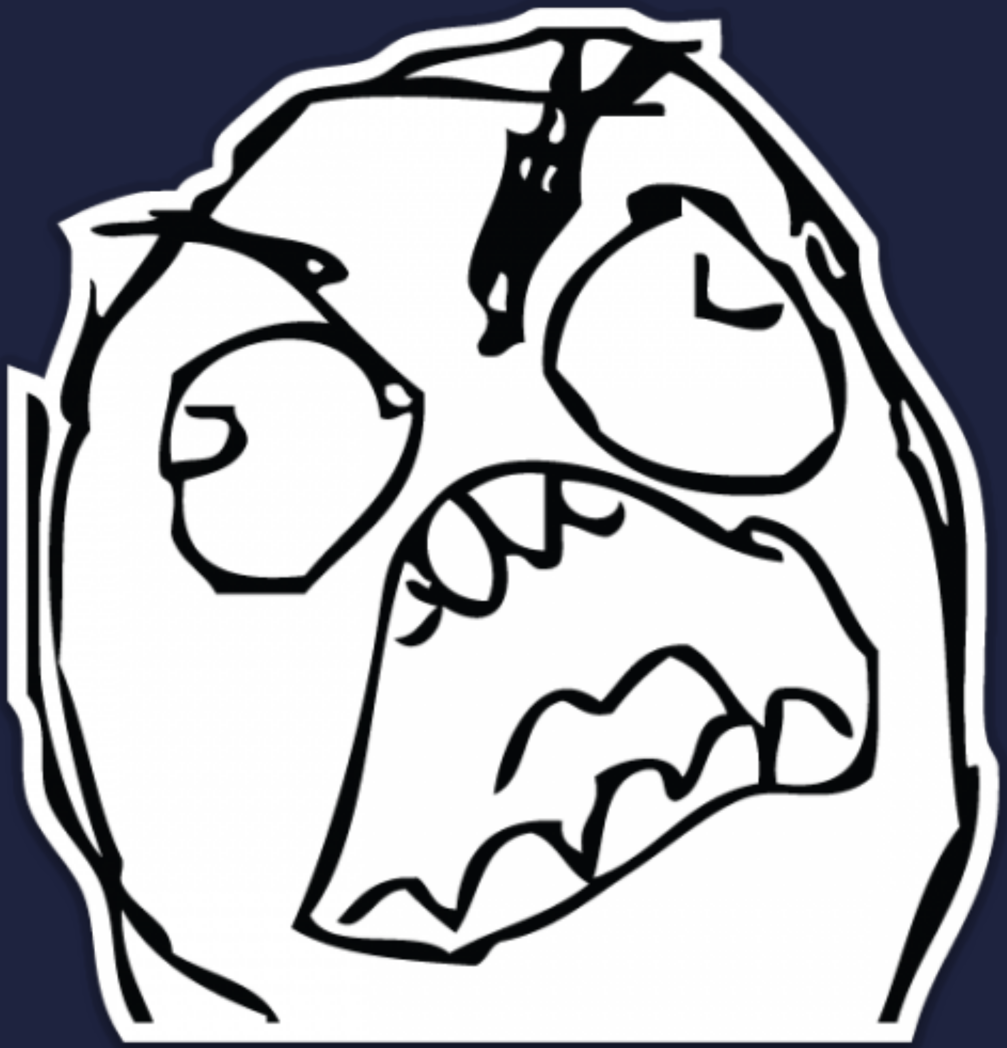
```
tableView.rx.contentOffset.subscribeNext { x in
    // do the same thing here
}
```

KVO

OR

KEY VALUE OBSERVING

```
<Something> was deallocated while key value observers were still registered with it.
```



Instead of

```
func observeValue(forKeyPath keyPath: String?, of object: Any?,
                  change: [NSKeyValueChangeKey : Any]?, context: UnsafeMutableRawPointer?)
```

... write

```
someView.rx.observe(CGRect.self, "frame").subscribeNext { frame in
    // do something with the new frame
}
```

NOTIFICATIONS

Instead of using:

```
let notificationName = "some_notification_name"

...

@objc func handleNotification() { ... }

...

NotificationCenter.default.addObserver(self, selector: #selector(handleNotification),
                                       name: Notification.Name(rawValue: notificationName),
                                       object: nil)

...

NotificationCenter.default.removeObserver(self)
```

... just write

```
NotificationCenter.default.rx.notification(Notification.Name(rawValue: notificationName), object: nil).map {
    /* do something */
}
```

OBSERVABLE OPERATORS

CREATING OBSERVABLE SEQUENCES

JUST

```
_ = Observable<String>.just("Hello CodeWăy ☀️").subscribe(onNext: { element in
    print(element)
}, onCompleted: {
    print("I'm done")
}

// Hello CodeWăy ☀️
// I'm done
```

REPEAT

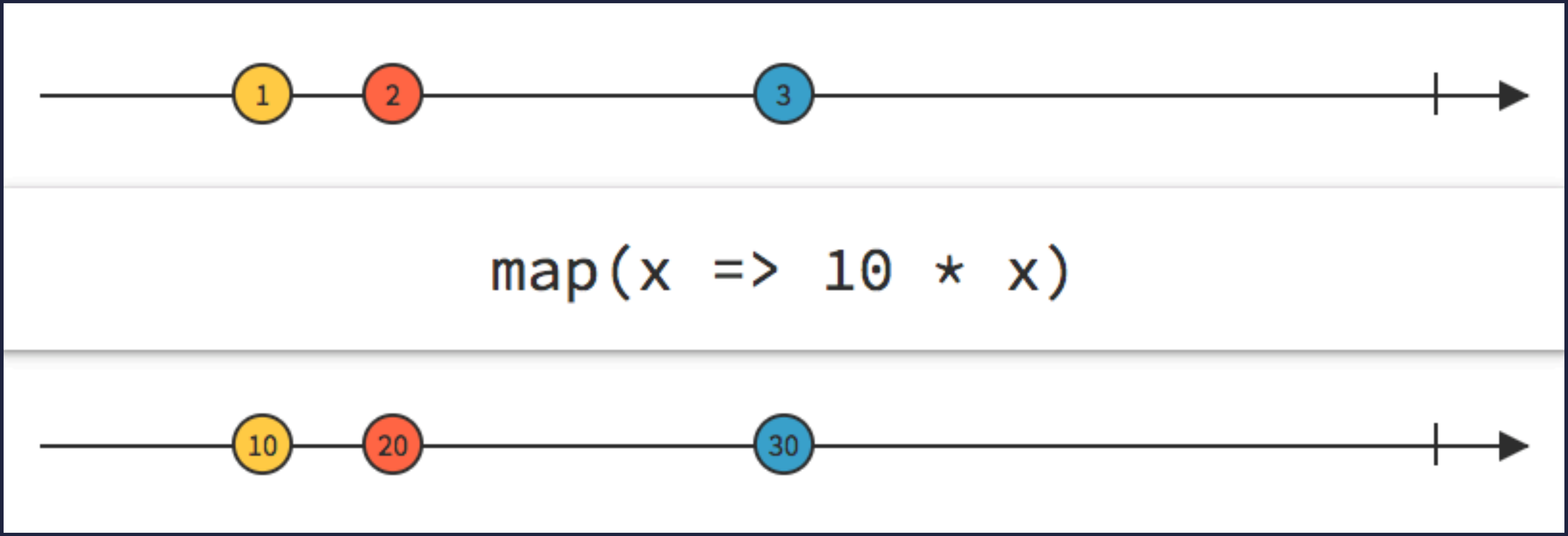
```swift
_ = Observable<String>.repeatElement("Reactive is fun 👍").subscribeNext { element in
    print(element)
}

// Reactive is fun 👍
// Reactive is fun 👍
// Reactive is fun 👍
// Reactive is fun 👍
// ...
```

CREATE

```swift
_ = Observable<String>.create { observer -> Disposable in
    let task = URLSession.shared.task(with: URL(string: <SomeURL>) { data, response, error in
        DispatchQueue.main.async {
            if let error = error {
                observer.onError(error)
            } else {
                observer.onNext(String(data: data!, encoding: .ascii)!)
                onserver.onCompleted()
            }
        }
    }
    task.resume()
    return AnonymousDisposable { task.cancel() }
}.subscribe { event in
    print(event)
}
```

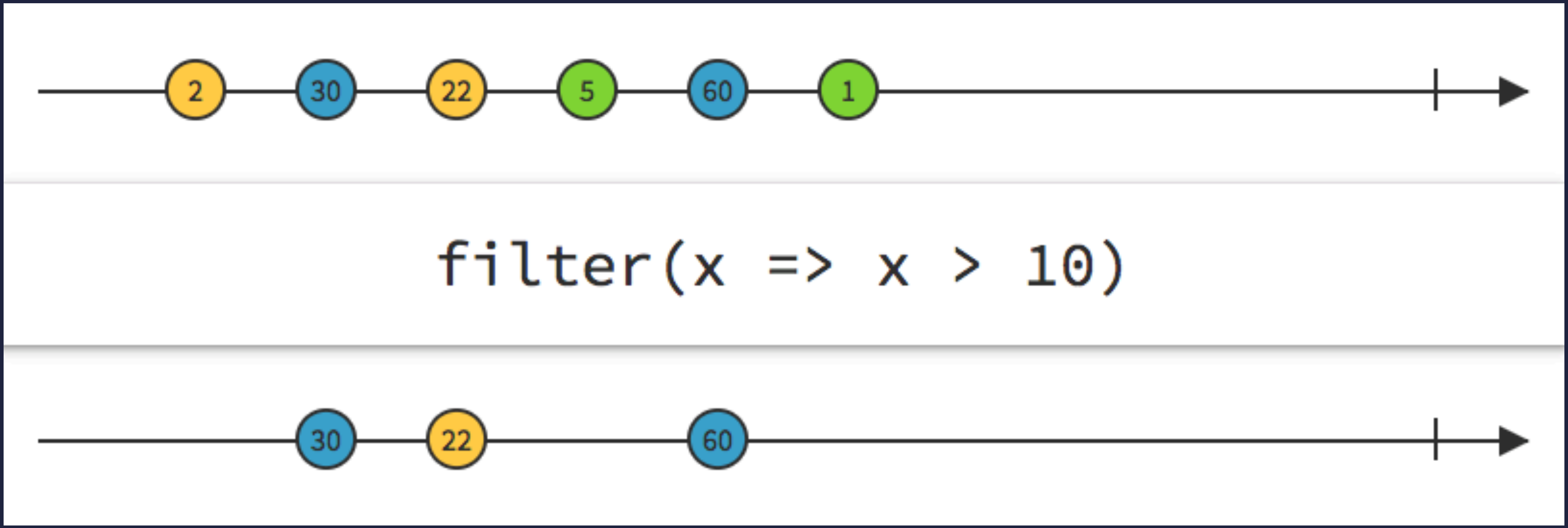
TRANSFORMING

MAP


1
2
3
map(x => 10 * x)
10
20
30

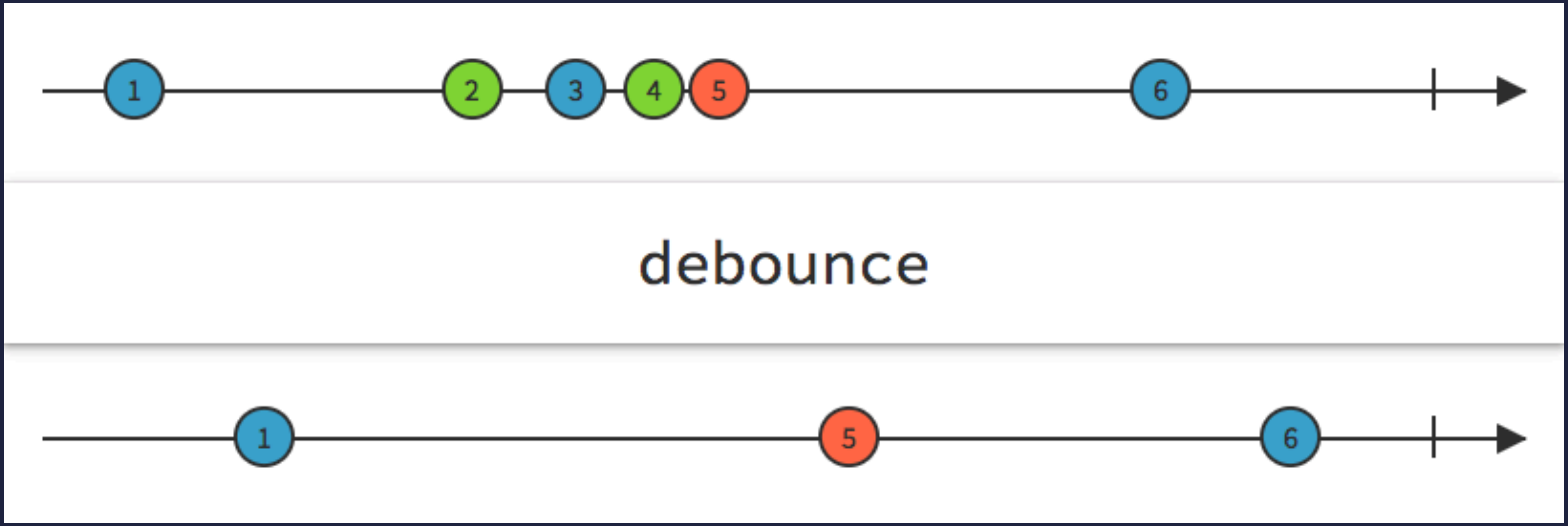
```
_ = Observable<Int>.create { observer -> Disposable in
    observer.onNext(1)
    observer.onNext(2)
    return NopDisposable.instance
}.map { $0 * 10 }.subscribeNext { print($0) }

// 10
// 20
```

FILTERING


2
30
22
5
60
1
filter(x => x > 10)
30
22
60

```
_ = Observable<String>.create { observer -> Disposable in
    observer.onNext("📱")
    observer.onNext("☎️")
    observer.onNext("📱")
    observer.onNext("☎️")
    return NopDisposable.instance
}.filter { $0 == "📱" }
 .subscribeNext { print($0) }

// 📱
// 📱
```

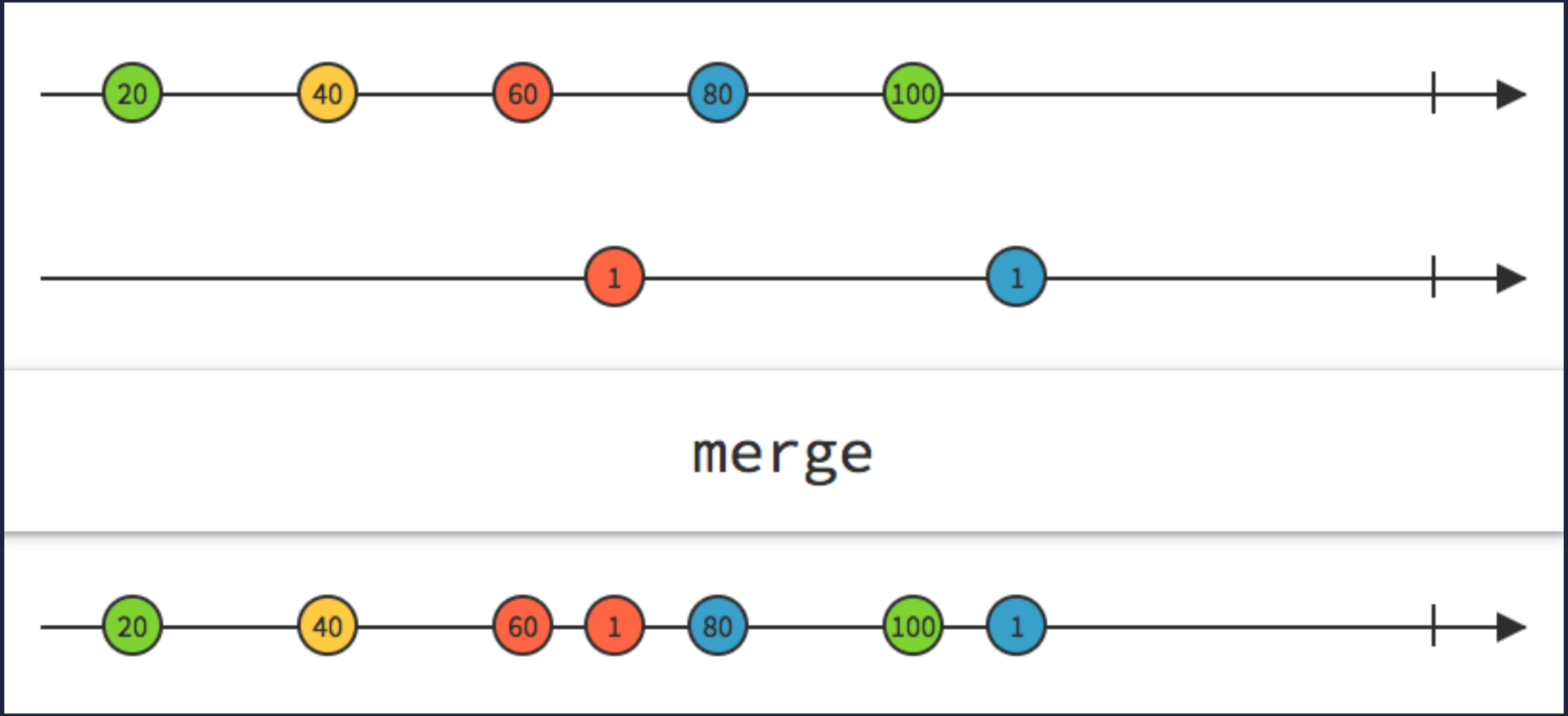
DEBOUNCE


1
2
3
4
5
6
debounce
1
5
6

```swift
observable.debounce(2, scheduler: MainScheduler.instance)
           .subscribeNext { print($0) }
```

COMBINING

MERGE

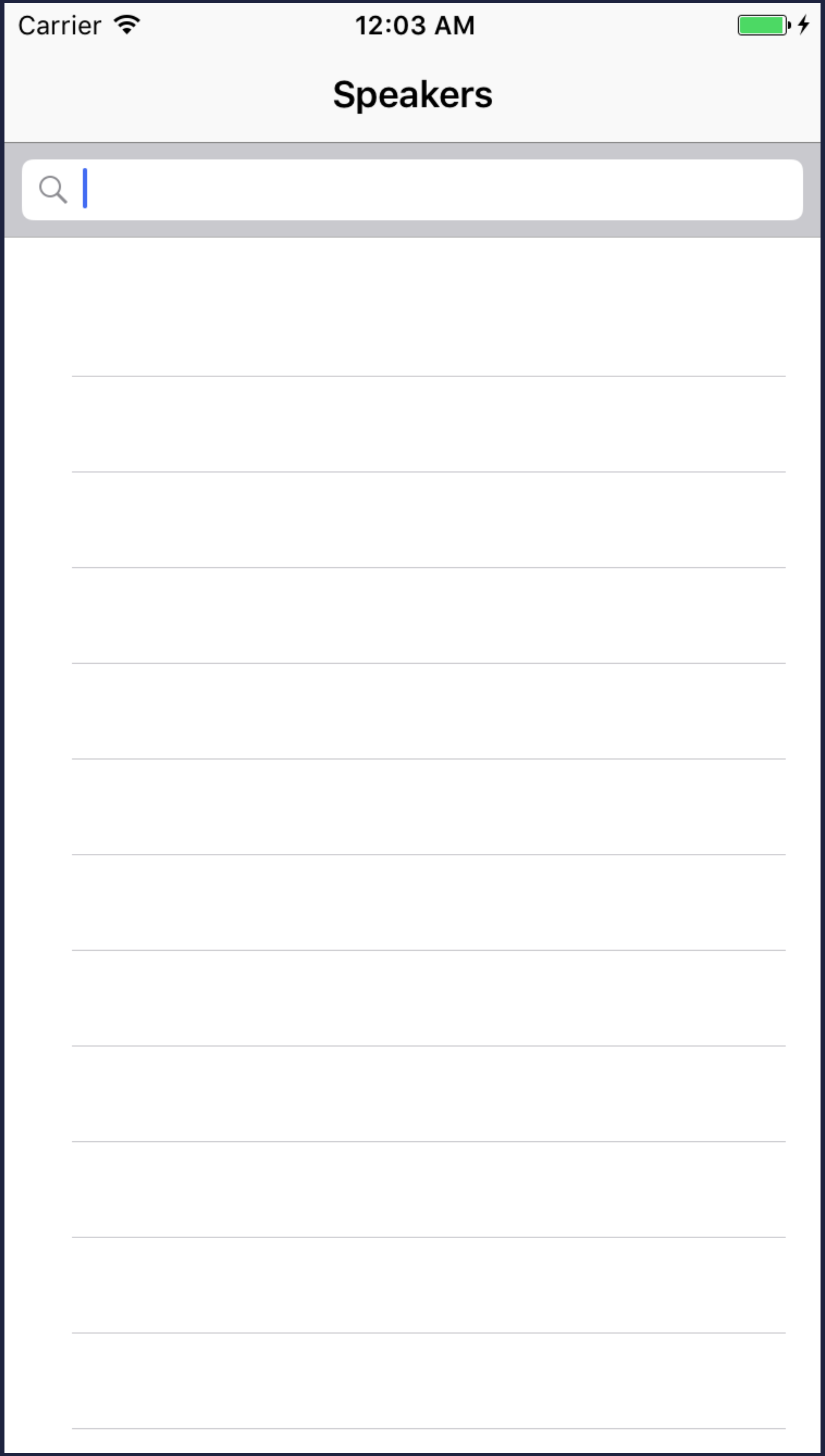

20
40
60
80
100
1
1
merge
20
40
60
1
80
100
1

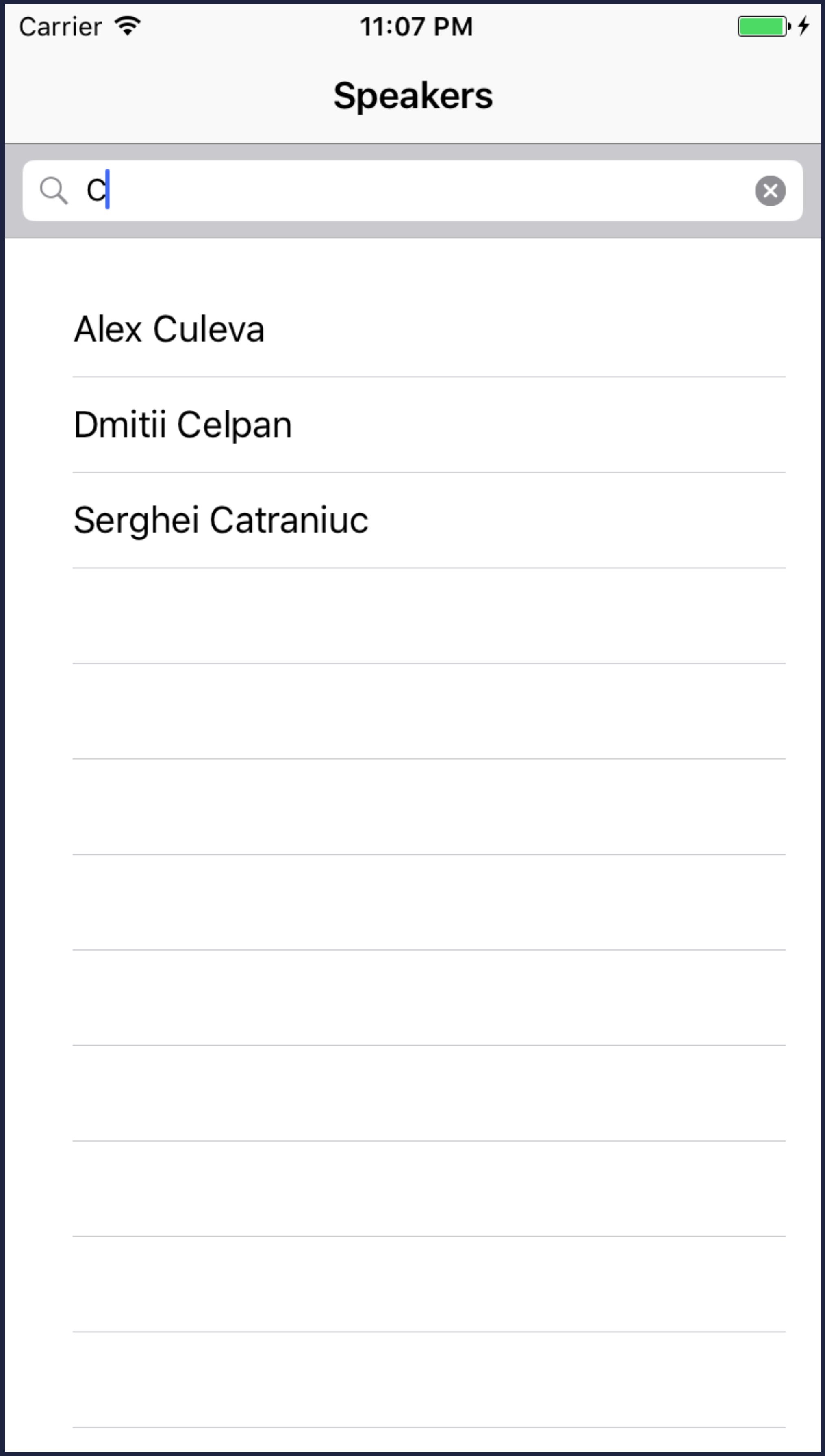
```swift
let observable1 = Observable<String>.create { observer -> Disposable in
    observer.onNext("🍺")
    observer.onNext("🍺")
    return NopDisposable.instance
}

let observable2 = Observable<String>.create { observer -> Disposable in
    observer.onNext("🍕")
    observer.onNext("🍕")
    return NopDisposable.instance
}

Observable.of(observable1, observable2)
          .merge()
          .subscribeNext { print($0) }

// 🍺
// 🍺
// 🍕
// 🍕
```

REAL LIFE EXAMPLE

```swift
struct Speaker {
    let firstName: String
    let lastName: String

    var fullName: String {
        return firstName + " " + lastName
    }
}
```

```swift
struct SpeakerViewModel {
    let speakers: [Speaker] = [
        Speaker(firstName: "Alex",    lastName: "Culeva"),
        Speaker(firstName: "Dmitii",  lastName: "Celpan"),
        Speaker(firstName: "Andrei",  lastName: "Raifura"),
        Speaker(firstName: "Serghei", lastName: "Catraniuc"),
        Speaker(firstName: "Andrei",  lastName: "Vidrasco")
    ]
}
```

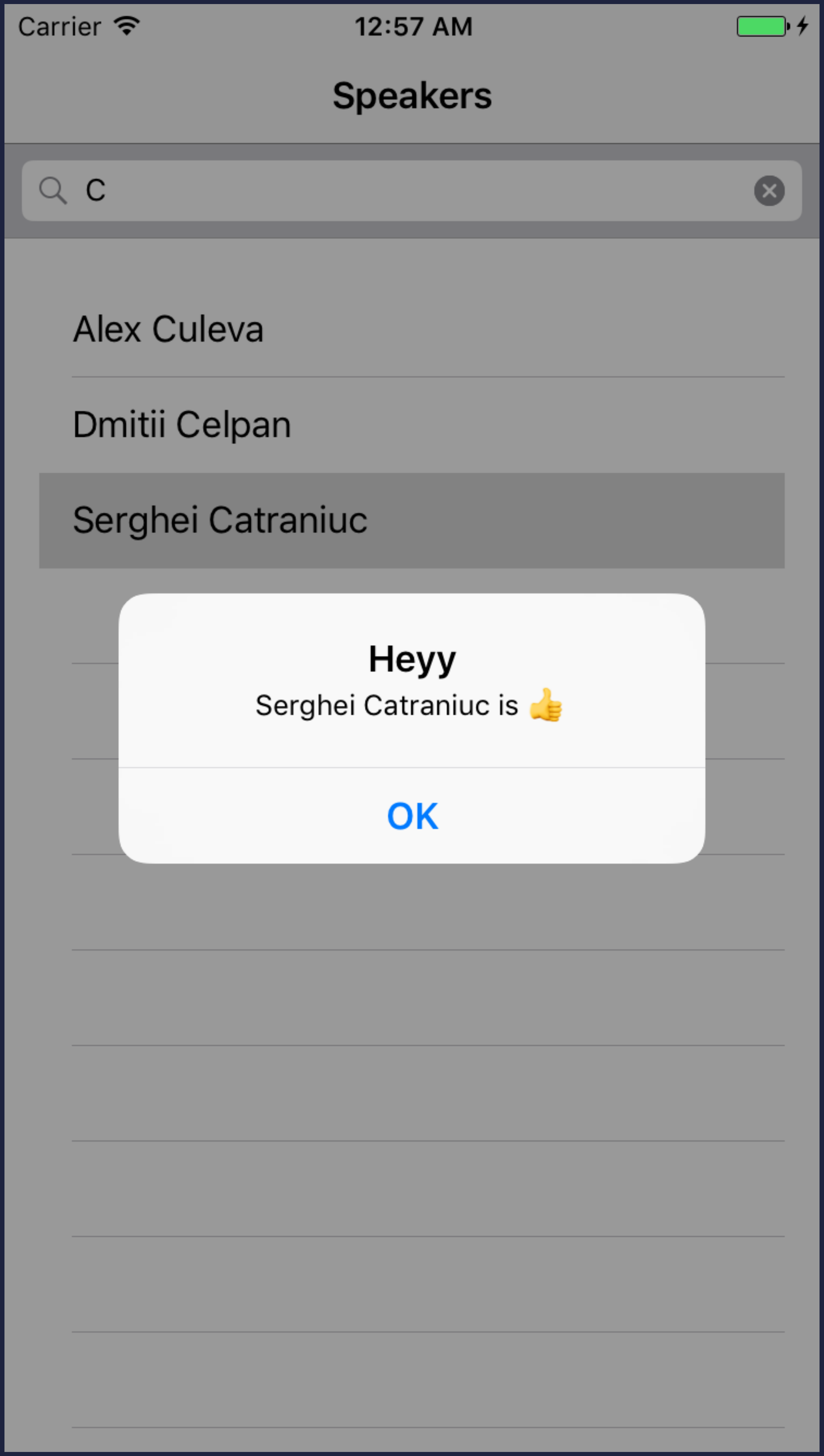

Carrier
12:03 AM
Speakers



Alex Culeva

Dmitii Celpan

Serghei Catraniuc



Alex Culeva

Dmitii Celpan

Serghei Catraniuc

Heyy

Serghei Catraniuc is 👍

NON-REACTIVE WAY

```swift
fileprivate let viewModel = SpeakerViewModel()
fileprivate var foundSpeakers: [Speaker] = []

override func viewDidLoad() {
    super.viewDidLoad()
    tableView.delegate = self
    tableView.dataSource = self
    searchBar.delegate = self
}
```

UITableViewDataSource

```swift
func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
    return foundSpeakers.count
}

func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCell(withIdentifier: "SpeakerCell")!
    cell.textLabel?.text = foundSpeakers[indexPath.row].fullName
    return cell
}
```

UITableViewDelegate

```
func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
    speakerSelected(foundSpeakers[indexPath.row])
}

func speakerSelected(_ speaker: Speaker) {
    let message = speaker.fullName + " is 👍"
    let alertController = UIAlertController(title: "Heyy",
                                            message: message, preferredStyle: .alert)
    alertController.addAction(UIAlertAction(title: "OK", style: .cancel, handler: nil))
    present(alertController, animated: true, completion: nil)
}
```

UISearchBarDelegate

```swift
func searchBar(_ searchBar: UISearchBar, shouldChangeTextIn range: NSRange, replacementText text: String) -> Bool {
    defer { tableView.reloadData() }
    guard let oldQuery = searchBar.text else {
        foundSpeakers.removeAll()
        return true
    }
    let newQuery = oldQuery.replacingCharacters(in: oldQuery.convertNSRangeToRange(range), with: text)
    if newQuery.isEmpty {
        foundSpeakers.removeAll()
        return true
    }
    foundSpeakers = viewModel.speakers.filter { $0.fullName.contains(newQuery) }
    return true
}
```

REACTIVE WAY

```swift
extension SpeakerViewModel {
    func search(by query: String?) -> Observable<[Speaker]> {
        guard let query = query else { return .just([]) }
        return .just(speakers.filter { $0.fullName.contains(query) })
    }
}
```

```swift
override func viewDidLoad() {
    super.viewDidLoad()

    // Search by query + binding to UI + reloading UI

    searchBar.rx.text
            .flatMap(viewModel.search)
            .bindTo(tableView.rx.items(cellIdentifier: "SpeakerCell")) { _, speaker, cell in
                cell.textLabel?.text = speaker.fullName
            }

    // Handling cell selection

    tableView.rx.modelSelected(Speaker.self)
            .subscribe(onNext: speakerSelected)
}
```

BENEFITS

1. COMPOSABLE

2. REUSABLE

3. DECLARATIVE

4. LESS STATEFUL

5. FLEXIBLE

GitHub repo

https://github.com/ReactiveX/RxSwift

All Operators

http://reactivex.io/documentation/operators.html

Marbles

http://rxmarbles.com/

QUESTIONS?

Mihai Seremet

mihai8804858@gmail.com

@SeremetMihai

Thank you!