Agenda
November 26

18.00

Intro

18.05

An introduction to Combine by Peter Haldbæk

18.15

Data sources in Combine by Michael Skiba

SHORTCUT

Copenhagen Cocoa — November 26

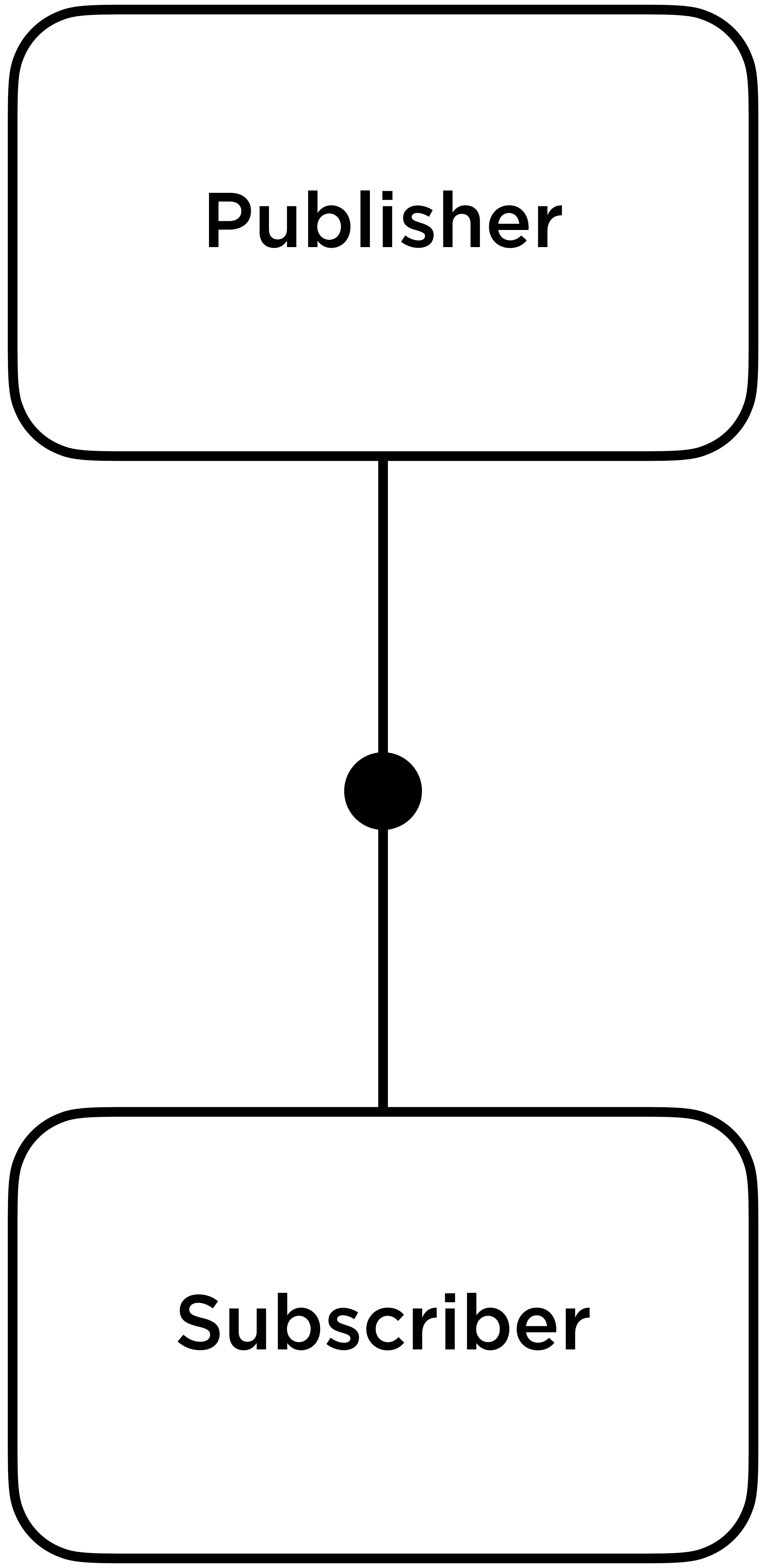
An introduction to Combine

Peter Haldbæk

What is Combine?

Framework which models all sorts of asynchronous events and operations as “values over time”

— John Sundell



A publisher is an observable object that emits values over time, and that can also optionally complete either when no more values are available, or when it encountered an error.

NotificationCenter.publisher

URLSession.dataTaskPublisher

Objects or closures that are used to observe a publisher are referred to as subscribers.

.sink(receiveCompletion:receiveValue:)

.assign(to:on:)

```swift
let url = URL(string: “https://api.github.com/users/peterhaldbaek")!
let publisher = URLSession.shared.dataTaskPublisher(for: url)

let cancellable = publisher.sink(
    receiveCompletion: { completion in
        // Called once, when the publisher was completed.
        print(completion)
    },
    receiveValue: { value in
        // Can be called multiple times, each time that a
        // new value was emitted by the publisher.
        print(value)
    }
)
```

```
let cancellable = publisher.sink(
    receiveCompletion: { completion in
        switch completion {
        case .failure(let error): print(error)
        case .finished: print("Success")
        }
    },
    receiveValue: { value in
        let decoder = JSONDecoder()
        do {
            let user = try decoder.decode(User.self, from: value.data)
            print(user)
        } catch {
            print(error)
        }
    }
)
```

```
let cancellable = publisher
    .map(\.data)
    .decode(
        type: User.self,
        decoder: JSONDecoder()
    )
    .sink(
        receiveCompletion: { completion in
            switch completion {
            case .failure(let error): print(error)
            case .finished: print("Success")
            }
        },
        receiveValue: { user in
            print(user)
        }
    )
```

Writing your own publisher

Subject

.send(_:)

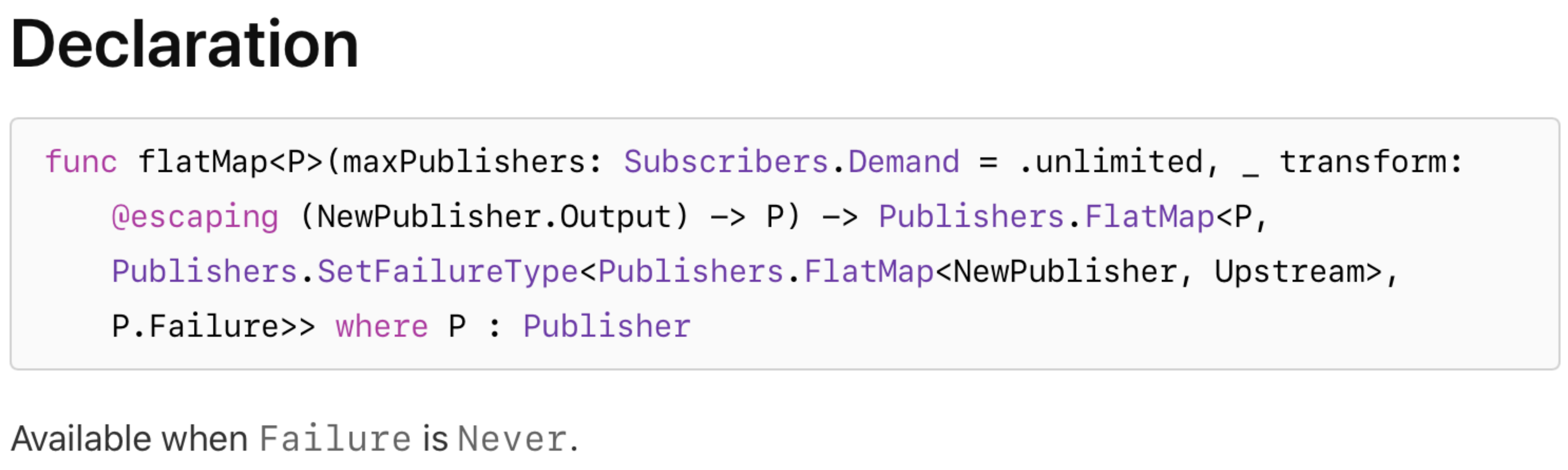
PassthroughSubject

CurrentValueSubject

Conclusions

Steep learning curve.

Debugging is hard.



Declaration

```
func flatMap<P>(maxPublishers: Subscribers.Demand = .unlimited, _ transform:
    @escaping (NewPublisher.Output) -> P) -> Publishers.FlatMap<P,
    Publishers.SetFailureType<Publishers.FlatMap<NewPublisher, Upstream>,
    P.Failure>> where P : Publisher
```

Available when `Failure` is `Never`.

Signatures are **hard** to read.

Error handling is still a mystery to me.

Questions?