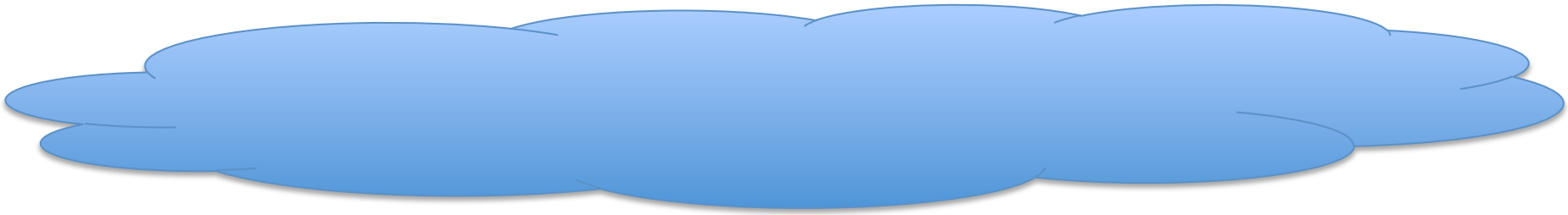
End to End Reactive Programming at Netflix

Who am I?

- Technical Lead for all the Netflix UI's
- 12 years in the industry, formerly worked at GE and Microsoft
- 4 years of experience building systems with Functional Reactive Programming

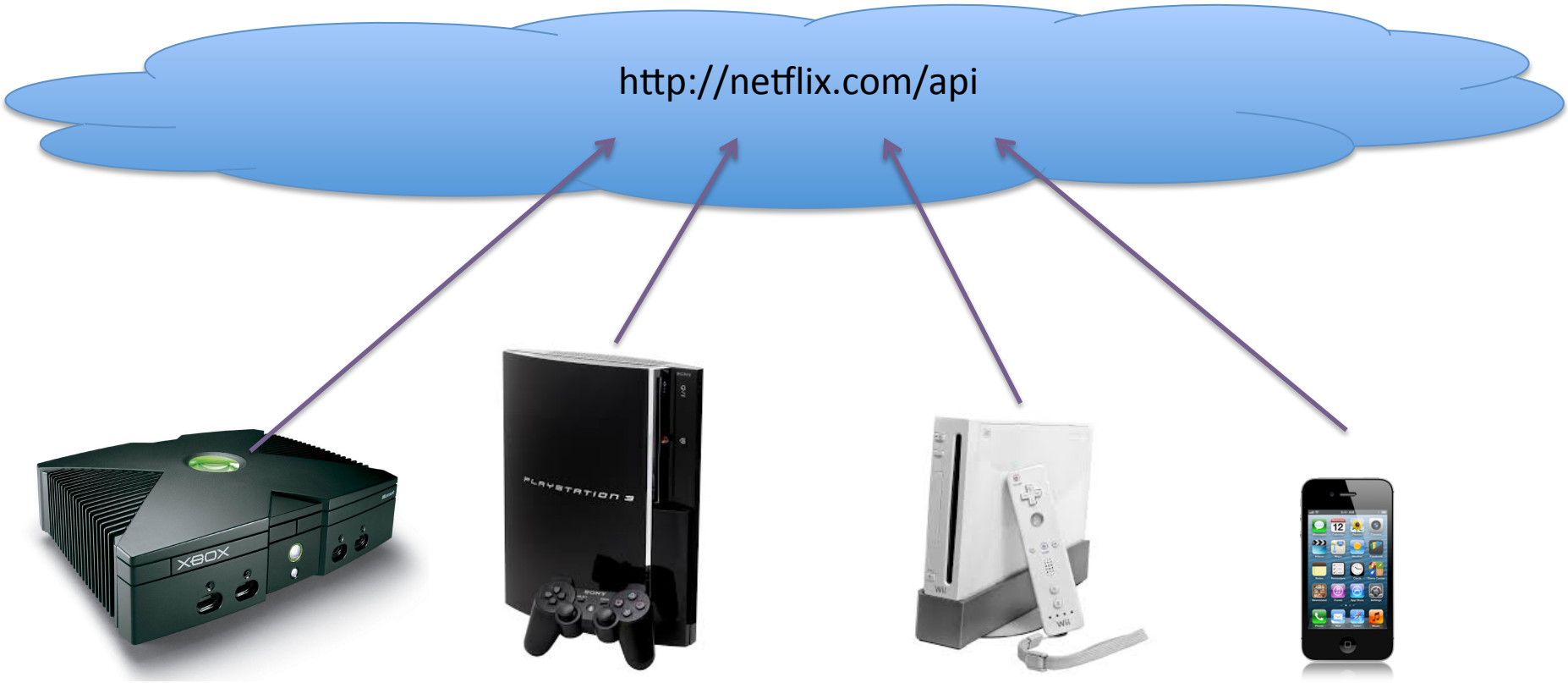
Rewind Two Years

Netflix had decided to change our client-server interaction model.



Before

All UI's used the same endpoints.

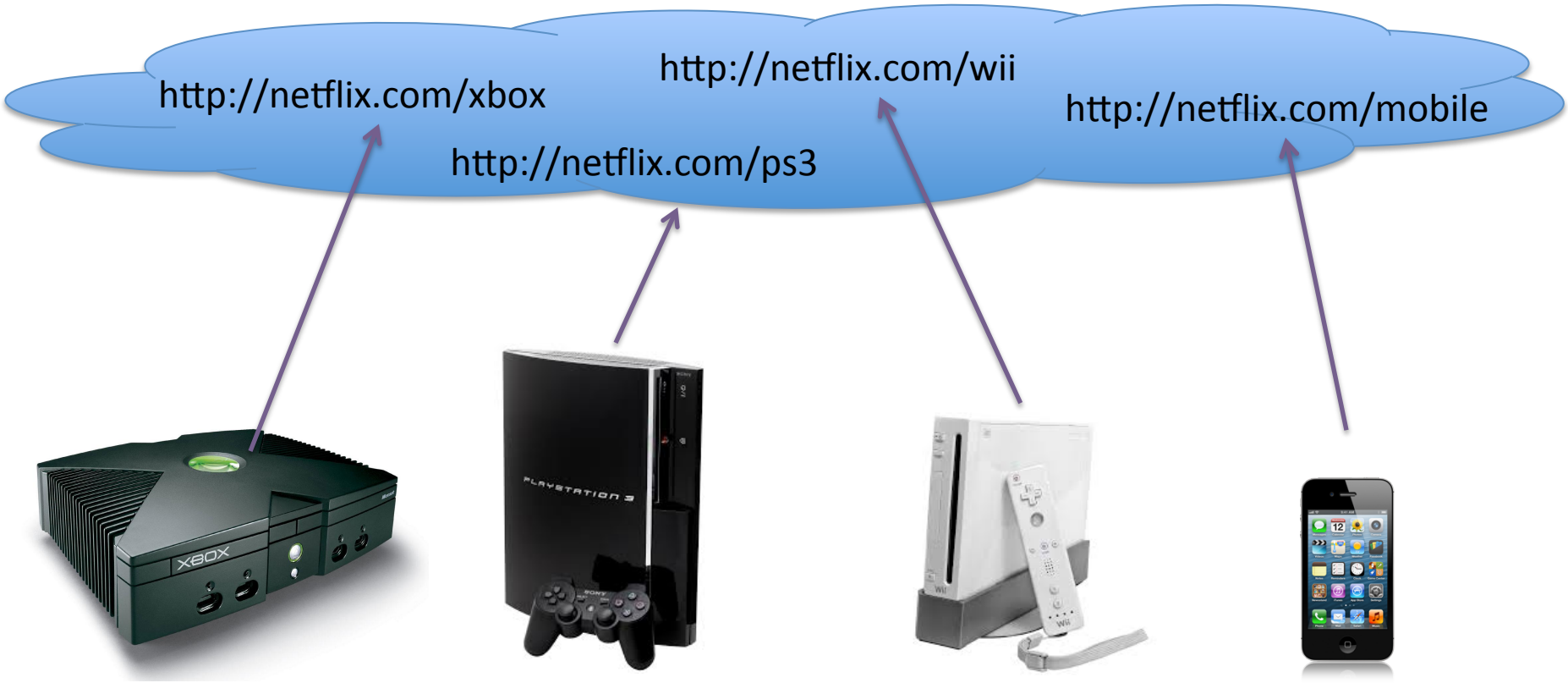


Problems

- Tight coupling between Middle Tier and UI teams
- One-sized fits all messages
- Inefficient call patterns

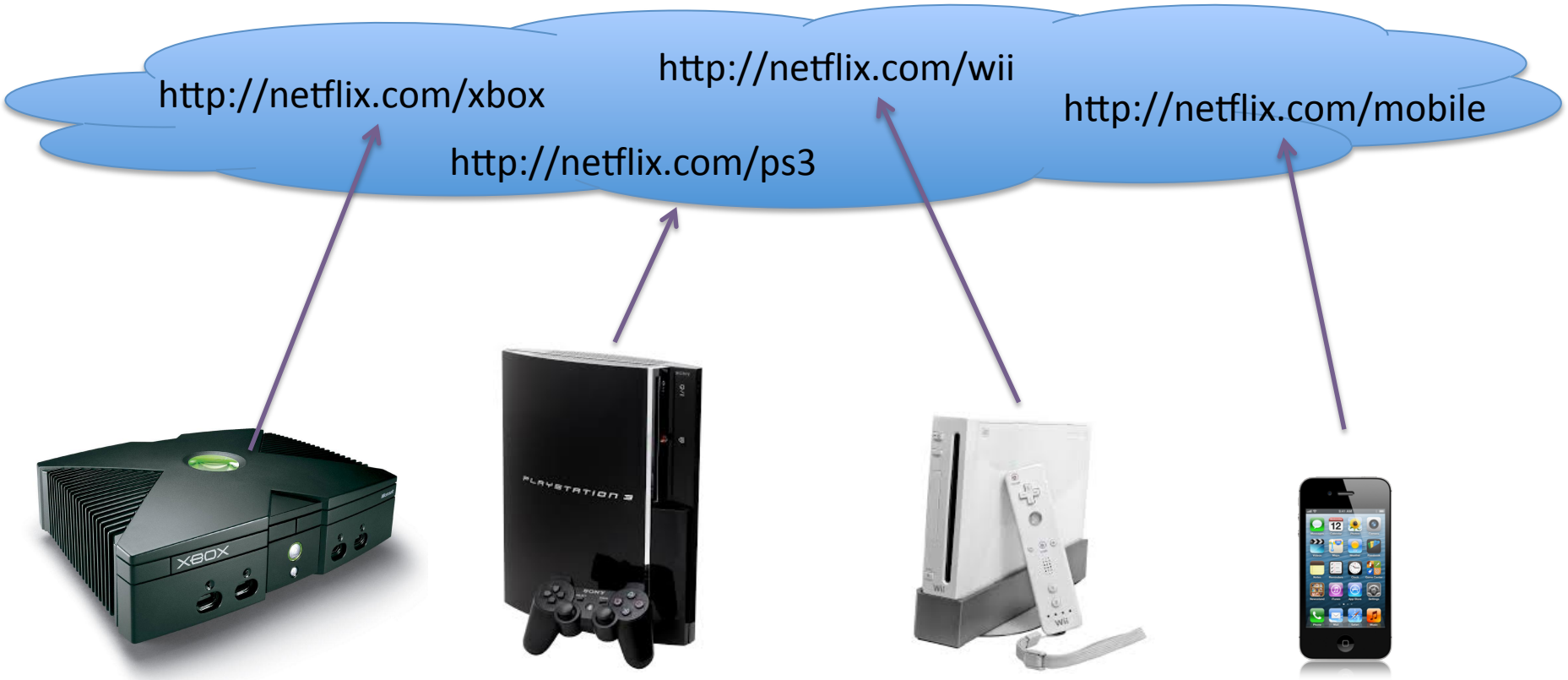
The Plan

Give UI developers the ability to create endpoints specialized for their devices.



Some UI developers saw it this way…

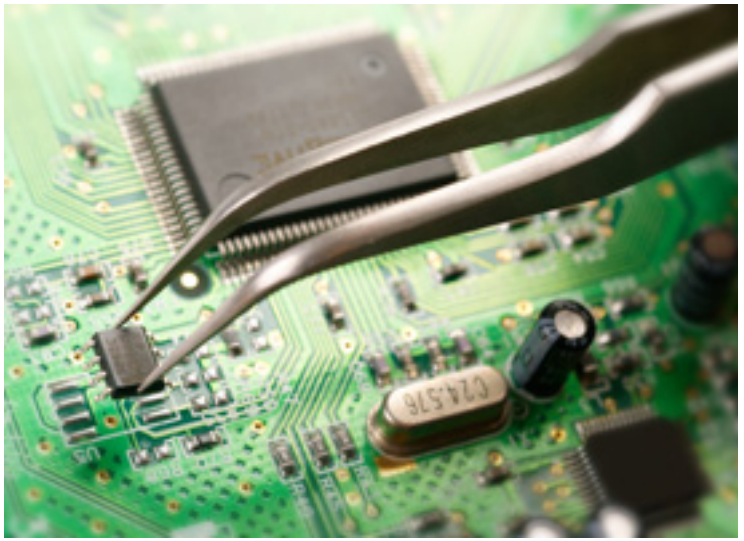
~~Give~~ Force UI developers ~~the ability~~ to create endpoints specialized for their devices.



Two Developer Personas

Cloud

UI



Challenge

How to turn UI developers into effective cloud developers?

Comforts for UI Developers

- Groovy
- OO API
- Reactive API

Reactive is Not Enough

- Parallelism + Aggregation == Contention
- Most ~~UI~~ developers can't be trusted with locks

How to make parallel programming safe for UI developers?

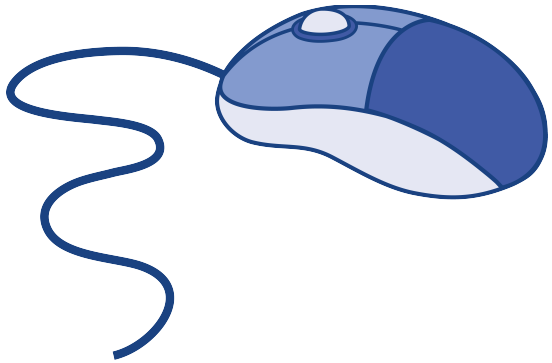
Rewind *Another* 2 Years

Microsoft


Erik Meijer

“What’s the difference between a database query…

…and a mouse drag event?”



“Nothing. They are both collections.”

New JS Closure syntax

ES5

```
var add = function(x,y) { return x + y; }
```

ES6

```
var add = (x,y) => x + y
```

Query for well-rated Movies

```
var getTopRatedFilms = user =>
   user.videoLists.
      map(videoList =>
         videoList.videos.
            filter(video => video.rating === 5.0)).
      flatten();

getTopRatedFilms(user).
   forEach(film => console.log(film));
```

Mouse Drag Event

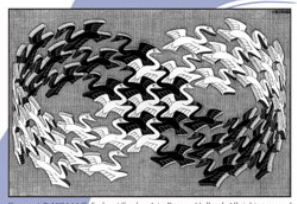
```
var getElementDrags  = elmt =>
   elmt.mouseDowns.
      map(mouseDown =>
         document.mouseMoves.
            filter takeUntil(document.mouseUps)).
      flatten();

getElementDrags(image).
   forEach(pos => image.position = pos);
```

Design Patterns

Elements of Reusable Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides



Foreword by Grady Booch

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Iterable<T>

Iterator<T> iterator()

Observable<T>

Disposable subscribe(Observer<T>)

Iterator<T> : Disposable

T next()

boolean hasNext()

throw new Throwable()

void dispose()

Observer<T>

void onNext(T)

void onCompleted()

void onError(Throwable)

Disposable

void dispose()

Observable and Iterable are dual!

Reactive Extensions

- Combinator Library for Observable type
- Open Source
- Ported to
  - C
  - C#/VB.Net
  - Javascript
  - Java (Netflix)

Observable Monad

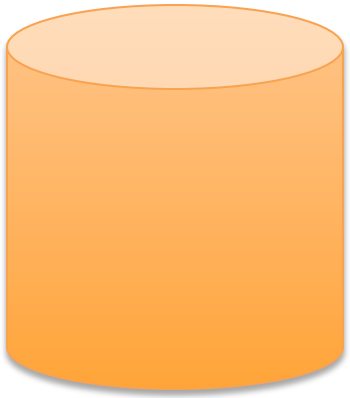
- Vector version of Continuation monad
- Null propagation semantics of Maybe monad
- Error propagation semantics of Either monad

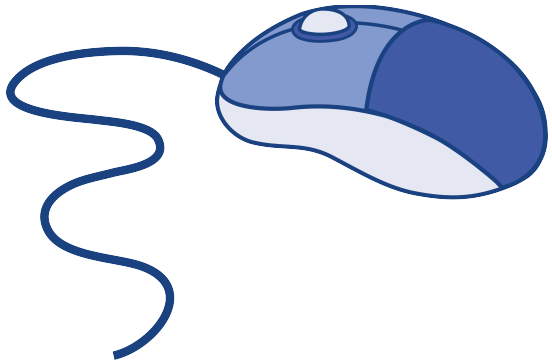
Observable Monad (cont.)

- Produced and consumed with side-effects
- Composed functionally
- Cancellation semantics
- Can be synchronous or asynchronous

Observable Monad (cont.)

Cleanly abstract over IO streams and UI events.



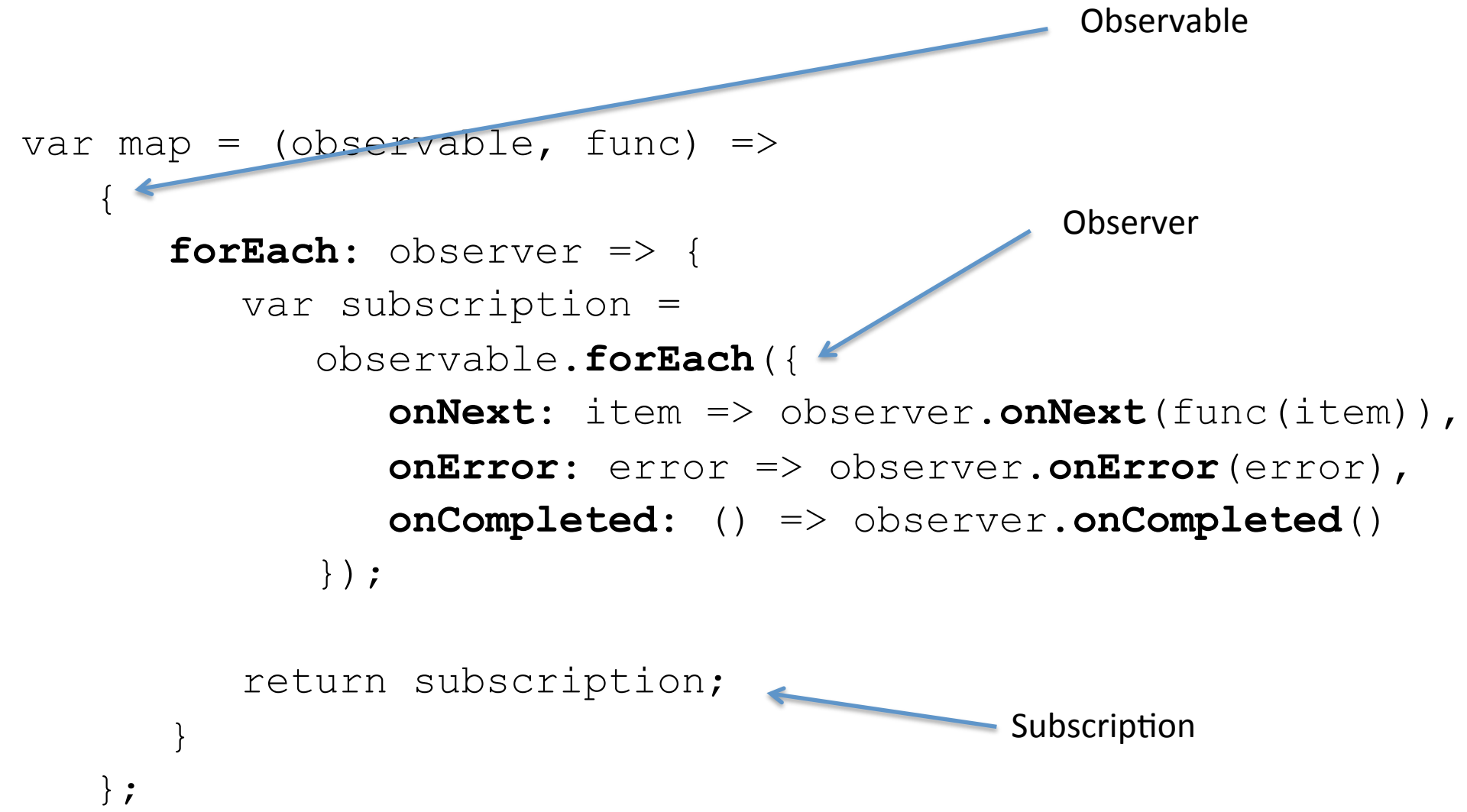


Map over Observable

```
var map = (observable, func) =>
    {
        forEach: observer => {
            var subscription =
                observable.forEach({
                    onNext: item => observer.onNext(func(item)),
                    onError: error => observer.onError(error),
                    onCompleted: () => observer.onCompleted()
                });

            return subscription;
        }
    };
```

Three Types of Composition




```
var map = (observable, func) =>
    {
        forEach: observer => {
            var subscription =
                observable.forEach({
                    onNext: item => observer.onNext(func(item)),
                    onError: error => observer.onError(error),
                    onCompleted: () => observer.onCompleted()
                });

            return subscription;
        }
    };
```

bservable<T>

One reactive type for cloud and UI developers.

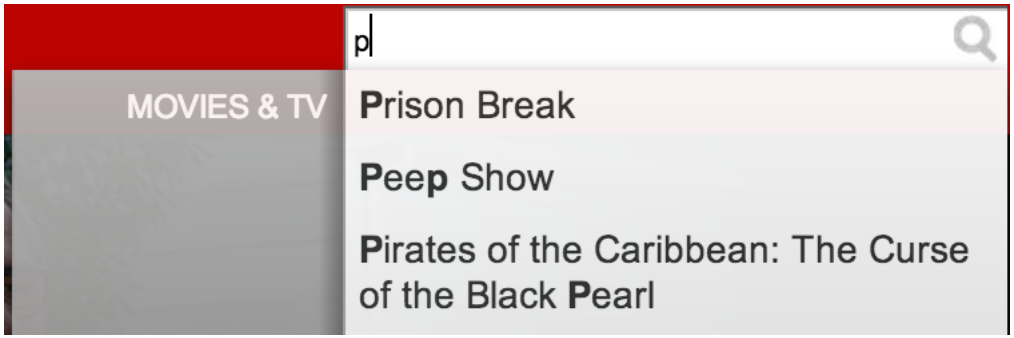
Social Notifications on Middle Tier

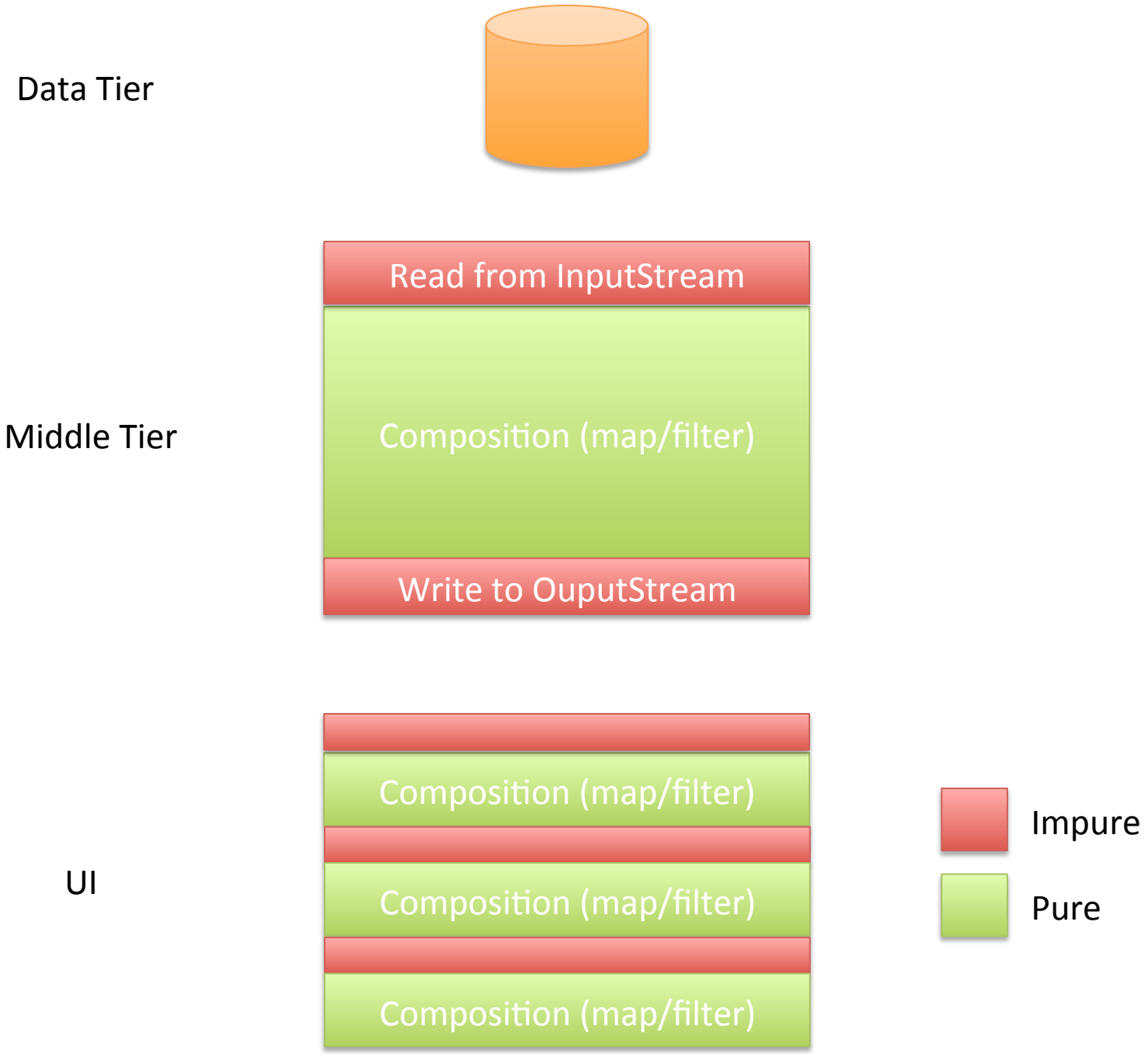
```
Observable.join(
   socialService.getFriends(user),
   messageService.
      getNotifications().
      filter(notification =>
          notification.video.isAvailable),
   friend => friend.id,   // join key selector
   notification => notification.friend.id, // join key selector
   (friend, notification) =>
      {
         id: notification.id,
         name: notification.video.name,
         message: notification.message,
         friend: { name: friend.name, id: friend.id }
      });
```

Search Auto-complete on the UI

```
var searchResultSets =
   keyPresses.
      throttle(20).
      flatMap(search =>
         getSearchResults(search).
            takeUntil(keyPresses));

searchResultSets.forEach(
   resultSet => listBox.setItems(resultSet));
```




Data Tier
Middle Tier
Read from InputStream
Composition (map/filter)
Write to OuputStream
UI
Composition (map/filter)
Composition (map/filter)
Composition (map/filter)
Impure
Pure

Wins

- Got Rx Open-sourced
- Ported Observable combinators to Java (RxJava)
- Currently using FRP on 3 different platforms
- Large sections of UI now written in FRP-style
- Growing awareness of and competency in functional programming in general

Challenges

- Evangelism
- Training
- Performance

Challenges: Evangelism

- Don’t assume best technical solution will win
- Practice public speaking
- Focus on the soft skills

Challenges: Training/Hiring

- Be available for support 24/7
- Teach at the same time
  - Functional Programming,
  - Vector Programming
  - Reactive Programming
- Look outside UI teams for FP competence
- bind/flatMap/concatMap/mapcat/mapMany
- Interactive training exercises
- Understanding where to apply FRP on the client

Challenges: Performance

- Chunking for low-end devices
- Best applied to less chatty event streams
- Decomposition to reduce per-item cost
- Type-unsafe flatMap easier to understand and faster

Resources

- https://github.com/Reactive-Extensions/RxJS
- http://jhusain.github.io/learnrx/

Questions