Taming asynchronous workflows with Functional Reactive Programming

LambdaJam - Brisbane, 2013

Leonardo Borges
@leonardo_borges
www.leonardoborges.com
www.thoughtworks.com

about:me

- Thoughtworker
- Functional Programming enthusiast
- Clojure Evangelist
- Founder & Organiser of the Sydney Clojure User Group (clj-syd)
- World traveller
- Fan of Murray's Beers :)

Leonardo Borges
@leonardo_borges
www.leonardoborges.com
www.thoughtworks.com

Functional programmers like programming with values:
`a, b, c...`

and pure functions:
`f, g, h...`

We get new values by applying functions to it

`(f a)  ;;=> b`

But that's hardly useful when
we have multiple values

```
(def vals [a b c])
```

So we use Higher Order Functions

`(map f vals)`

And compose them as we see fit

```
(-> vals
    (filter f)
    (map g)
    (reduce h))
```

But what if the value isn't known...yet?

?

We make promises

```
;; thread#1
(def a (promise))

;; ...later in the program
(f @a) ;;<= blocks thread

;; thread#2
(deliver a 10) ;; now thread#1 continues
```
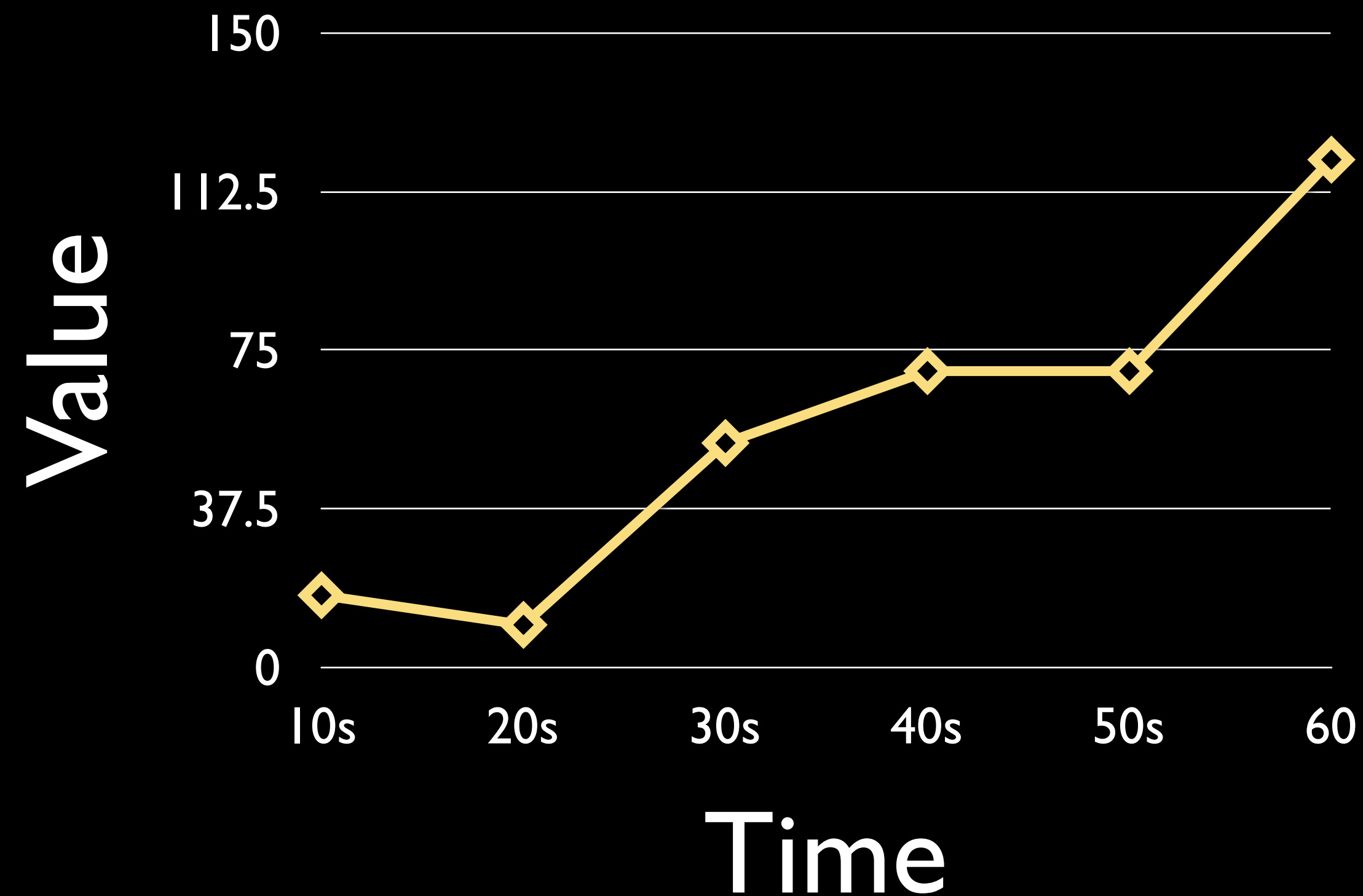
Not great if we want to 'react' to a new value
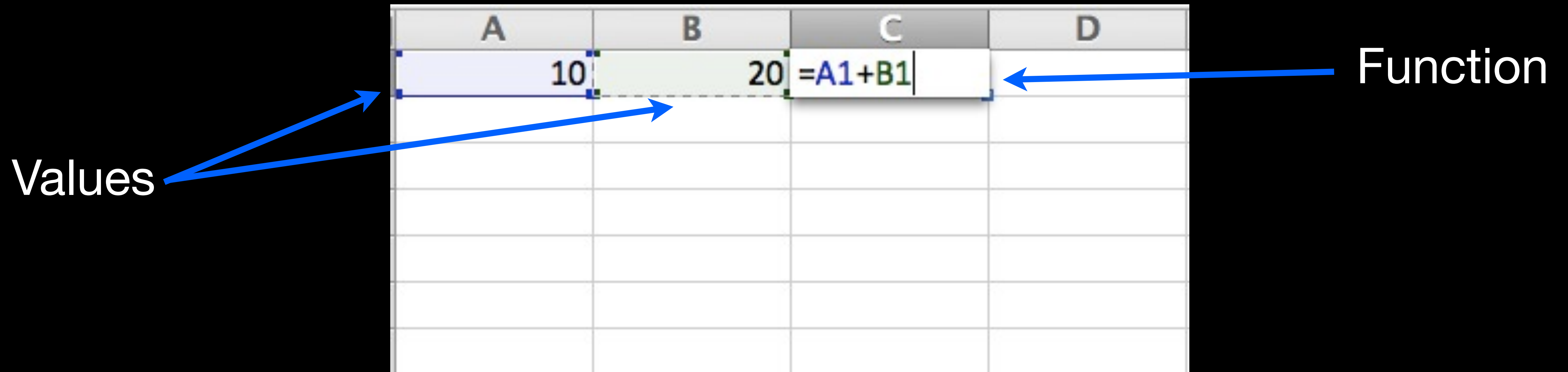
What about a list of - as of yet unknown - values?

[? , ? , ?]

Or better yet, a value that changes over time?

Does this sound familiar?

Spreadsheets: a poor man's reactive programming model

Function

Values

Spreadsheets: a poor man's reactive programming model

Our function cell reacts to the change

As we change a value

'Changing a value' is an event

Several events over time form an event stream

"Functional Reactive Programming is about effectively processing event streams without explicitly managing state"

- me

"FRP is about handling time-varying values like they were regular values."

- Haskell wiki

We'll use Reactive Extensions (Rx) - but there are many implementations

In Rx, event streams are called Observable sequences

Rx 101

```clojure
(-> (.returnValue js/Rx.Observable 42)
    (.map #(* % 2))
    (.subscribe #(.log js/console %)))

;; 84
```

Rx 101

```
(-> (.fromArray js/Rx.Observable
                (clj->js [10 20 30]))
    (.map #(* % 2))
    (.reduce +)
    (.subscribe #(.log js/console %)))

;; 120
```

Rx 101

```clojure
(defn project-range [n]
  (.returnValue js/Rx.Observable (range n)))

(-> (.fromArray js/Rx.Observable
                (clj->js [1 2 3]))
    (.selectMany project-range)
    (.subscribe #(.log js/console (clj->js %))))

;; [0]
;; [0 1]
;; [0 1 2]
```

Observables are Monads

The Monad Type Class

```
class Monad m where
    return :: a -> m a
    (>>=)  :: m a -> (a -> m b) -> m b
```

Monad functions: return

```
return :: a -> m a

returnValue :: a -> Observable a
```

Monad functions: >>= (bind)

```
(>>=) :: m a -> (a -> m b) -> m b

selectMany :: Observable a -> (a -> Observable b) -> Observable b
```

Demo: Simple polling app

# Server exposes poll questions and results

```
e.g.:
{:id 7
 :question "Which is the best music style?"
 :results {:a 10
           :b 47
           :c 17}}
```
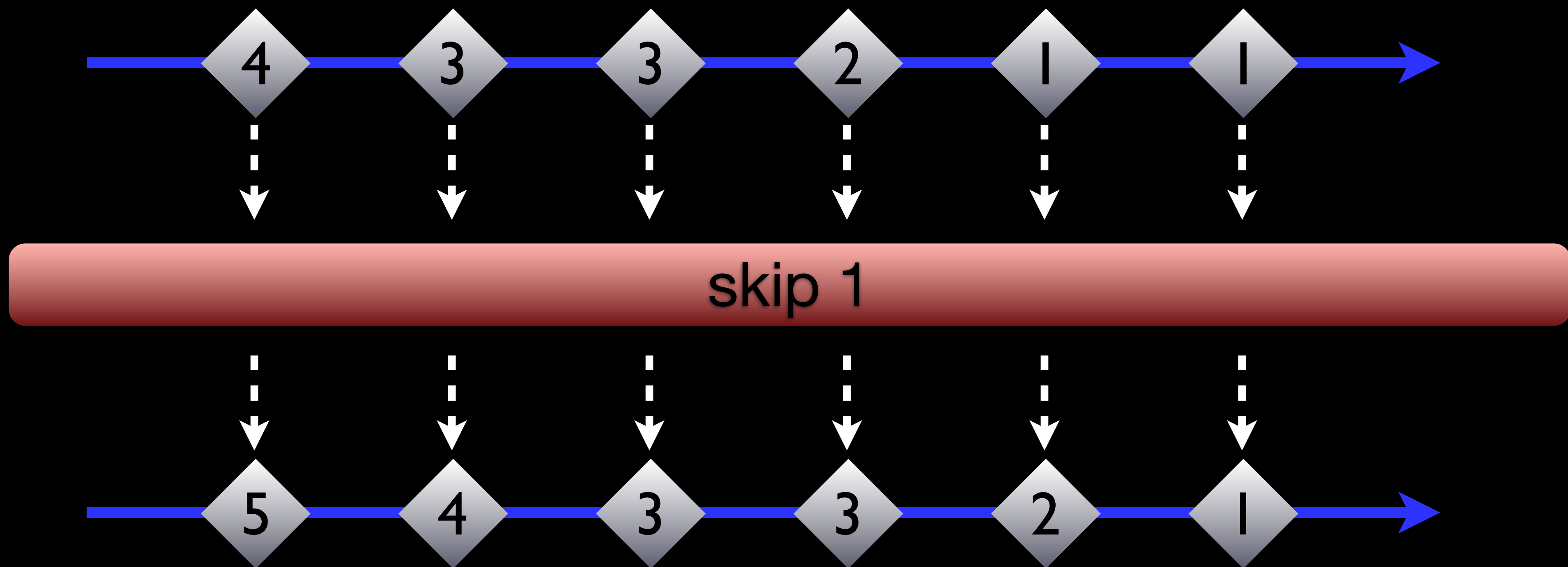
What we want

- Render results
- Continuously poll server every 2 secs
- If current question is the same as the previous one update results;
  - Otherwise:
    - Stop polling;
    - Display countdown message;
    - Render new question and results;
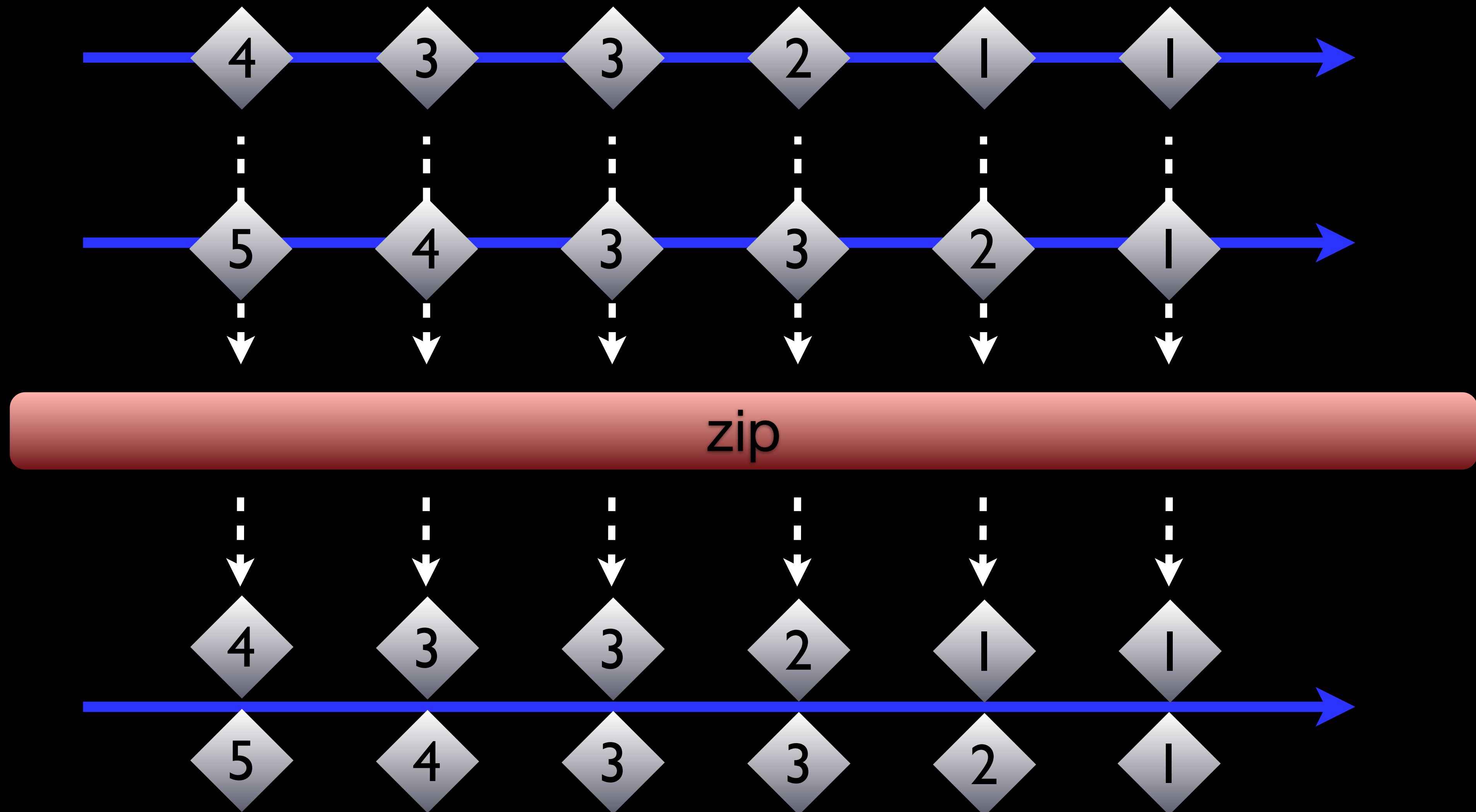   - Restart polling;

The core idea

Turn server results into an event stream
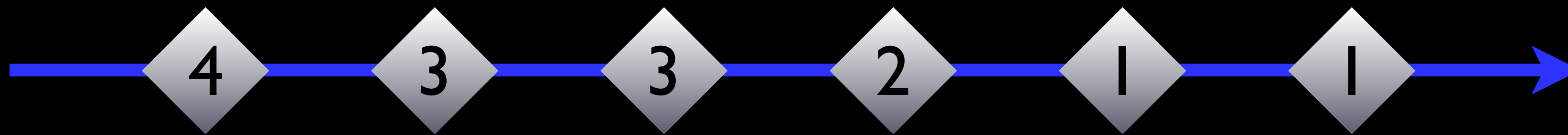
Duplicate stream, skipping one

Zip them together

Now we have access to both the previous and current results, with no local variables

Show me the code!

https://github.com/leonardoborges/frp-code

The core idea

```clojure
(def results-connectable
  (let [obs (-> js/Rx.Observable
                (.interval 2000)
                (.selectMany results-observable)
                (.publish)
                (.refCount))
        obs-1 (.skip obs 1)]
    (.zip obs obs-1 (fn [prev curr]
                      {:prev prev
                       :curr curr})))))
```
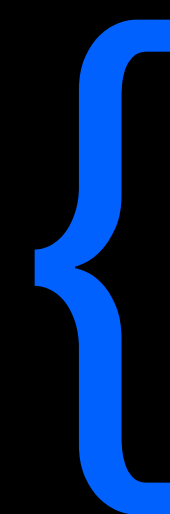
Turn server results into an event stream {

Clone stream, skip one →

Zip them together {

"FRP is about handling time-varying values like they were regular values."

- Haskell wiki

Questions?

Leonardo Borges
@leonardo_borges
www.leonardoborges.com
www.thoughtworks.com

References

Code - https://github.com/leonardoborges/frp-code

RxJS - https://github.com/Reactive-Extensions/RxJS
RxJava - https://github.com/Netflix/RxJava

Other FRP implementations:
Reactive-banana - http://www.haskell.org/haskellwiki/Reactive-banana
Javelin (Clojurescript) - https://github.com/tailrecursion/javelin
Bacon.js - https://github.com/raimohanska/bacon.js