Functional & Reactive UIs with JavaScript

Lambda Days 2016

Survey

OOP: Alan Key

Objects communicate by asynchronous message passing

Zaiste

@zaiste

POLY CONF 16

JUN 30 - JUL 2, 2016
POZNAN, PL

My story

2005

Python

Research Internship at LIMSI in Paris

Functional programming

First-class functions

Functions as datatype: passed around, returned, etc.

OCaml

Introduction to functional programming by Marc Pouzet

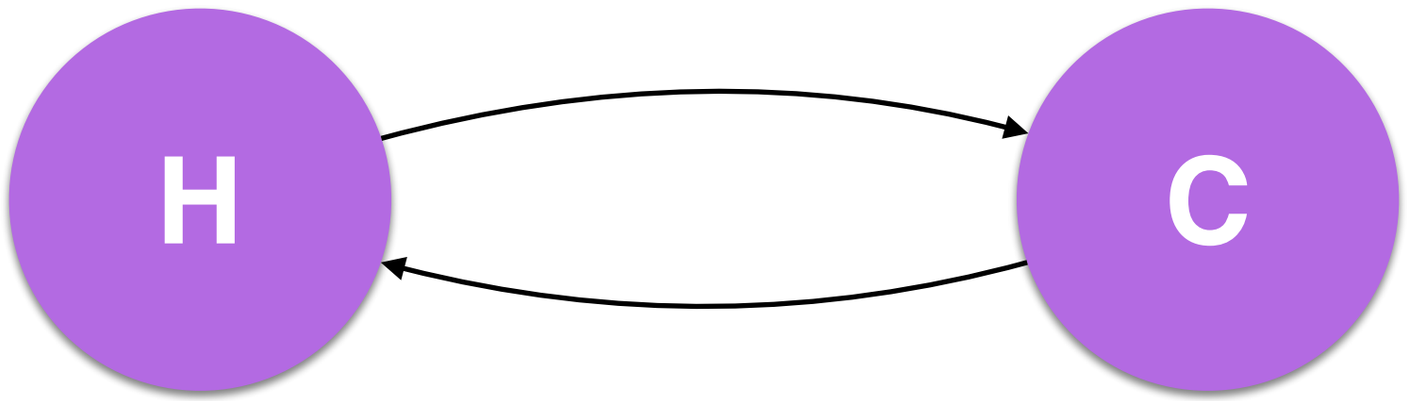
ML

Developed in 1970 by Robin Milner

ML Influence

Haskell, Elm, …

UIs



HCI

Human-Computer Interaction

Model-View-Controller

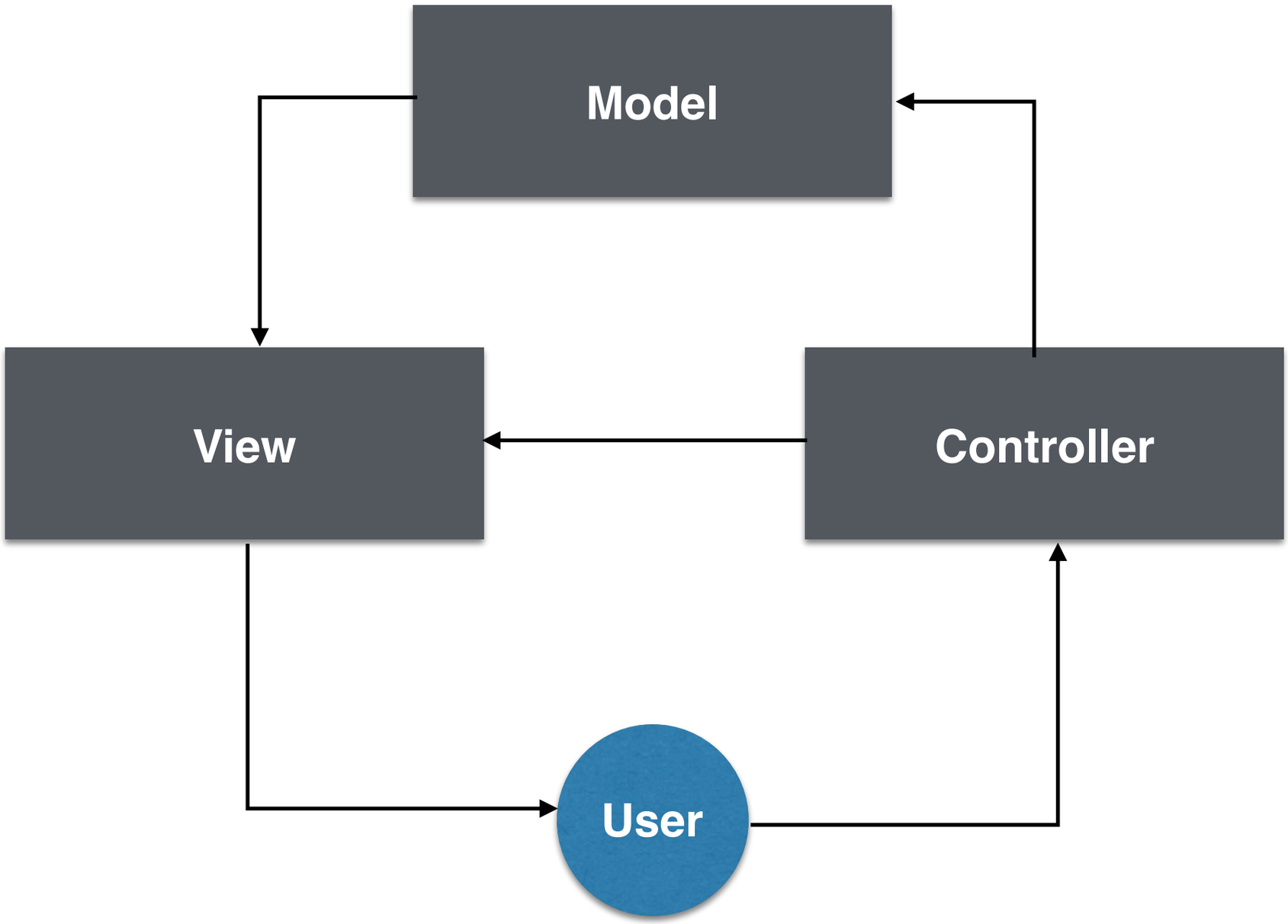
Trygve Reenskaug in the 1970s

Smalltalk-76

At XEROX PARC

MVC

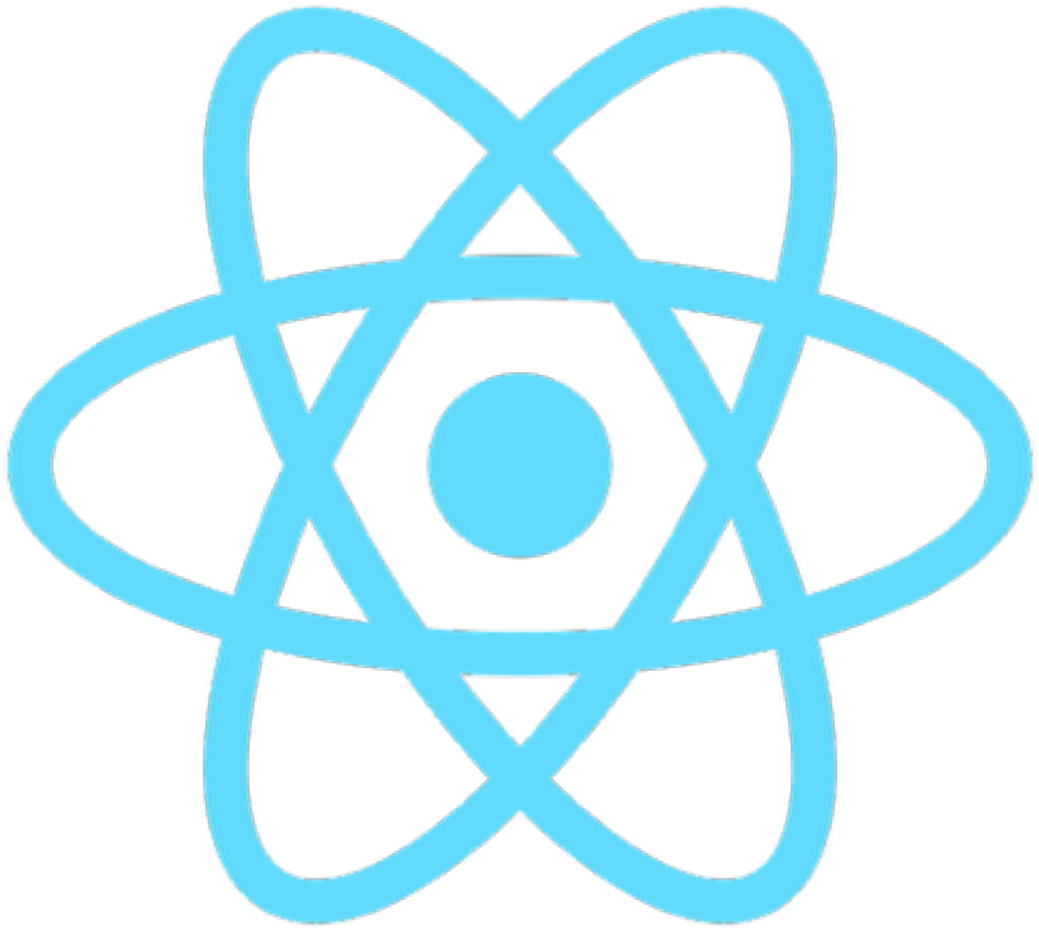
Digital model vs Mental model


Model
View
Controller
User

Callback-based

Classical approach

Functional UIs



React

Greatly improved rendering engine

Declarative

Simple and straightforward UI representation

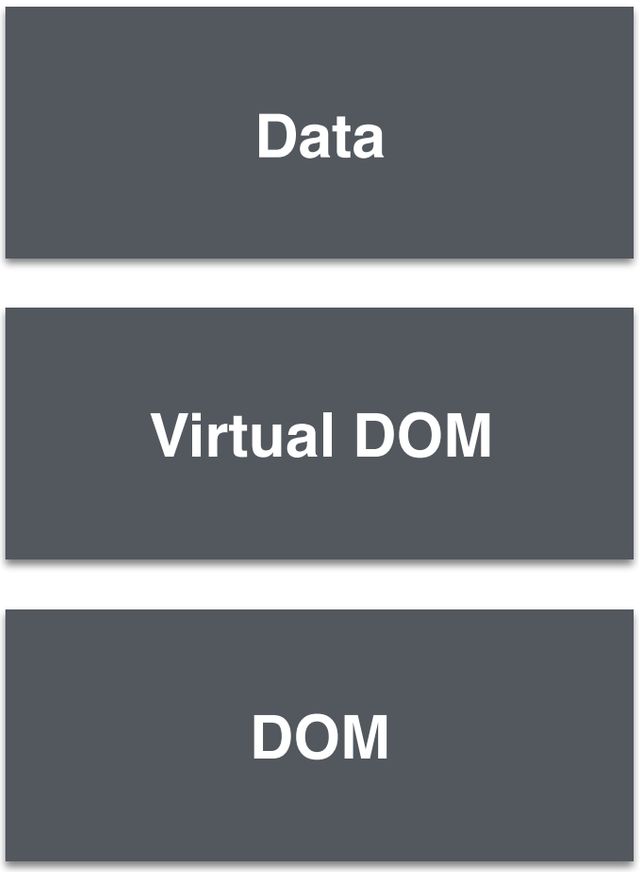
UI is fn of state

Functional foundation

$$f(D_0) \rightarrow V_0$$

$$f(D_1) \rightarrow V_1$$

$$\mathrm{diff}(V_0, V_1) = \mathrm{changes}$$


Data
Virtual DOM
DOM

User Inputs

Action Creators

Actions

Dispatcher

Callbacks

Store

View

New Concepts

Difficult for newcomers

Reactive programming

A spreadsheet

Most popular reactive programming tool


A
B

Passive

B allows others to change its state

Reactive

B manages its state by reacting to external events

Separation of concerns

A and **B** are responsible for themselves

Encapsulation

B hides its internals

Reactive

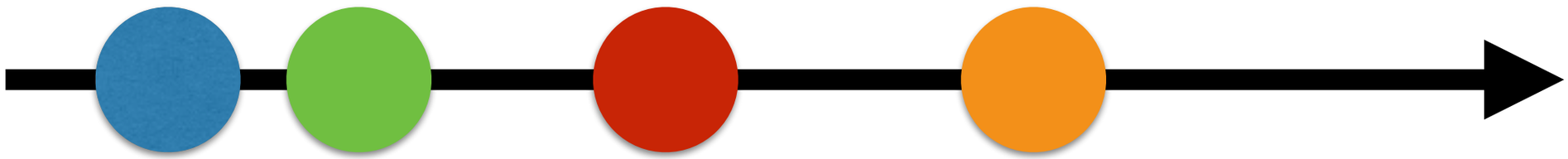
Independent, self-contained modules

Values over time

First class concept

Observable

Simple abstraction: Observer + Iterator



Marble Diagram

Lazy event stream

0+ events, finite or infinite

Asynchronicity

Doing more simpler

What, not *how*

Declarative approach to program logic

Simple composition

Outputs can be given as inputs

Unification

Promises, Callbacks, Web Workers, Web Sockets

RxJS

JavaScript reactive programming library

```
const doubleClickObservable = clickObservable
  .buffer(() => clickObservable.debounce(250))
.map(arr => arr.length)
.filter(x => x === 2);
```

Let’s combine

Cycle.js

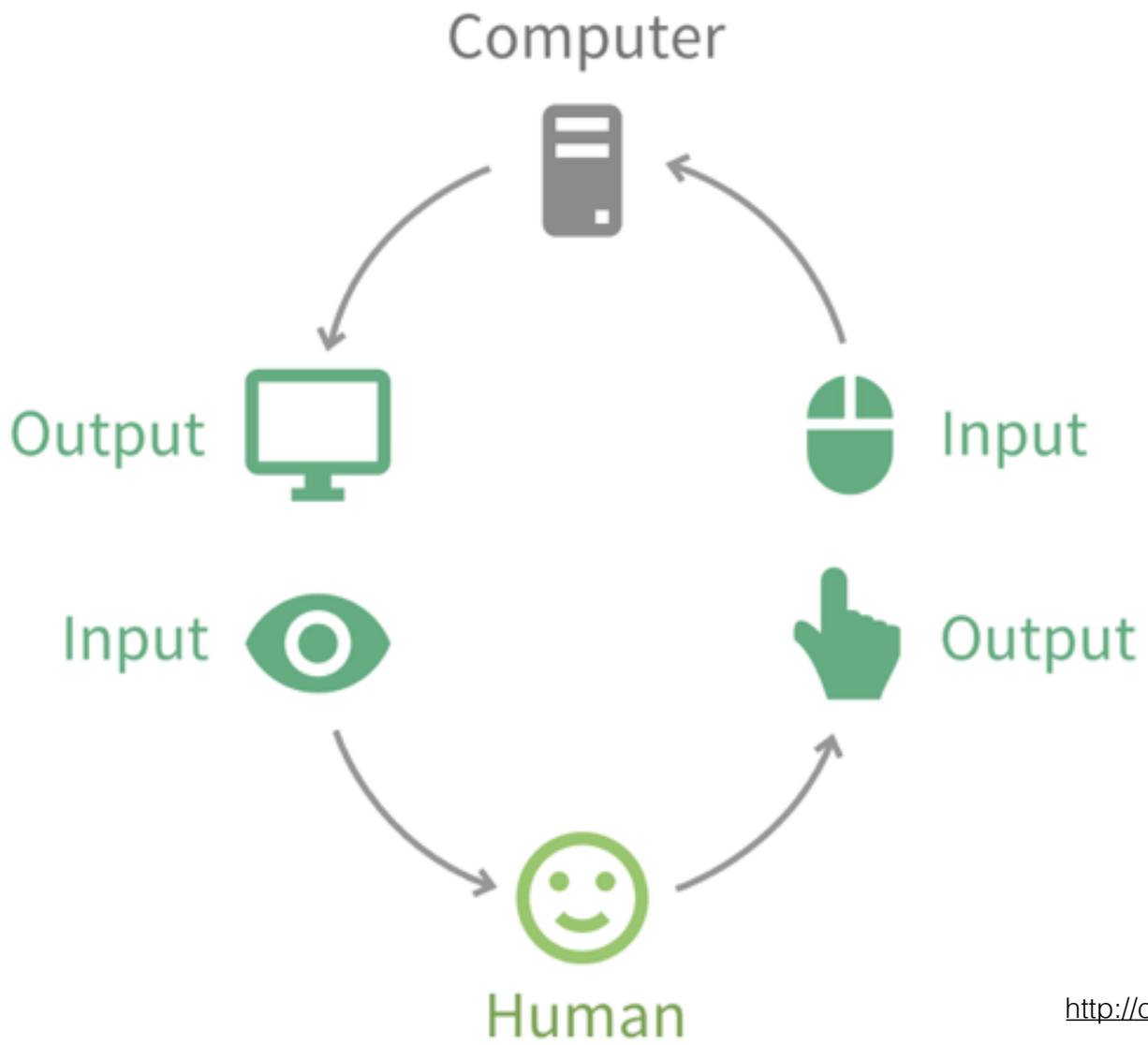
Functional and reactive UI framework in JavaScript

Concise

Both the framework and applications built with it

Simple API

Cycle.run()



http://cycle.js.org/dialogue.html

fn : input -> output

Constant dialogue

computer()

```
fn : inputDevices -> outputDevices
```

human()

```
fn : senses -> actuators
```

UIs as « cycles »

Natural way for interactions

main()

Your application as a pure function

```
function computer(userEventsObservable) {
  return userEventsObservable
    .map(event => /* ... */)
    .filter(somePredicate)
    .flatMap(transformItToScreenPixels);
}
```

Sources

inputs: read effects from the external world

Sinks

outputs: write effects to apply to the external world

Drivers

Plugins that manage the side effects

Model-View-Intent

Parts of the main() function

Intent

Processing inputs from the external world

Model

It represents the state

View

it creates the output e.g. virtual dom

Composable

Dataflow components

Demo

```
function main() {
  return {
    DOM: Rx.Observable.interval(1000)
      .map(i => CycleDOM.h1('' + i + ' seconds elapsed'))
  };
}
```

```
function main(sources) {
  return {
    DOM: sources.DOM.select('.field').events('input')
      .map(ev => ev.target.value)
      .startWith('')
      .map(name =>
        div([
          input('.field', {attributes: {type: 'text'}}),
          h1('Hello ' + name),
        ])
      )
  };
}
```

Functional

Applications made of pure functions

Reactive

Observables simplify events, async & errors handling

Limitations

http://lambda-the-ultimate.org/node/4900

Q? / Thank you