
React

artsy.github.io

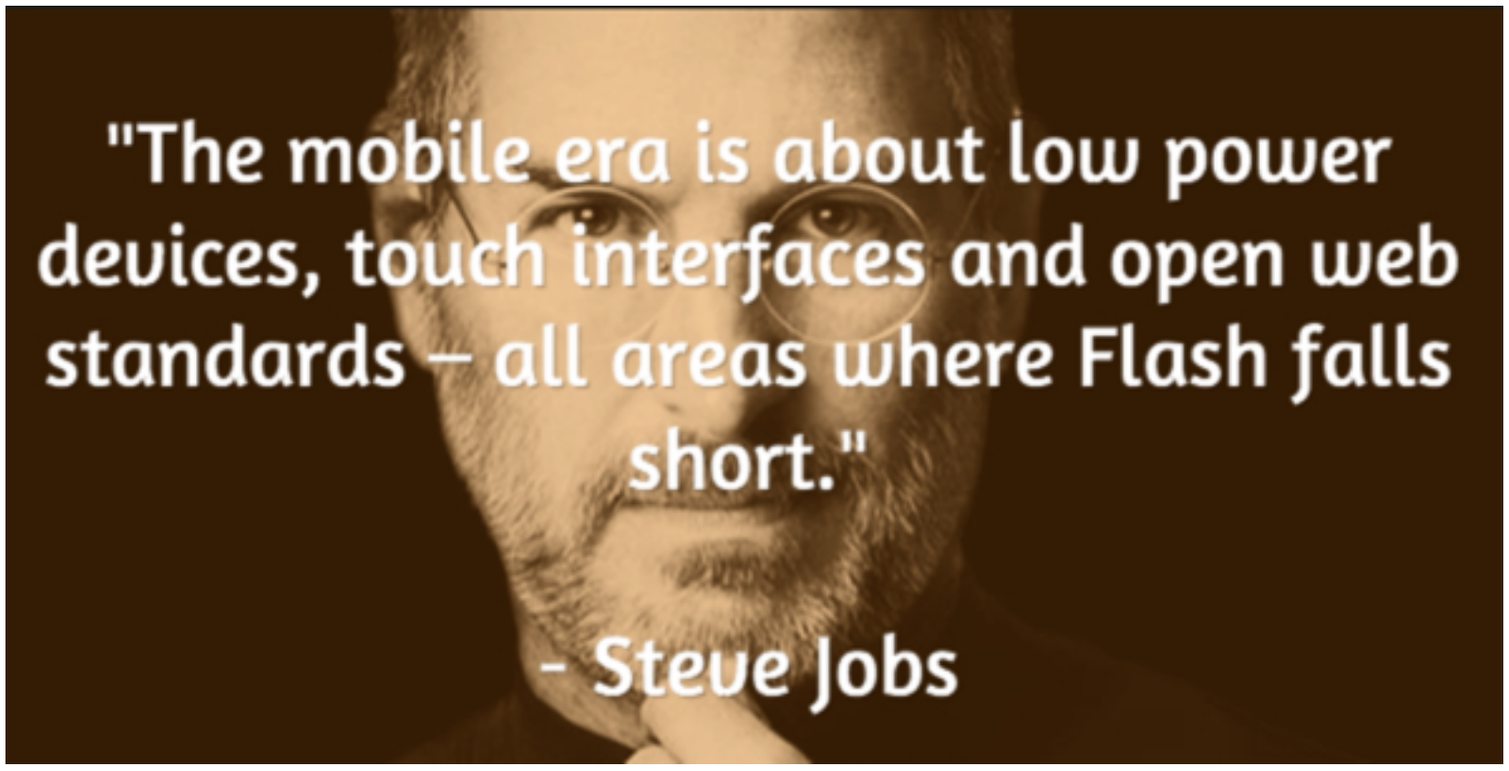
artsy.net

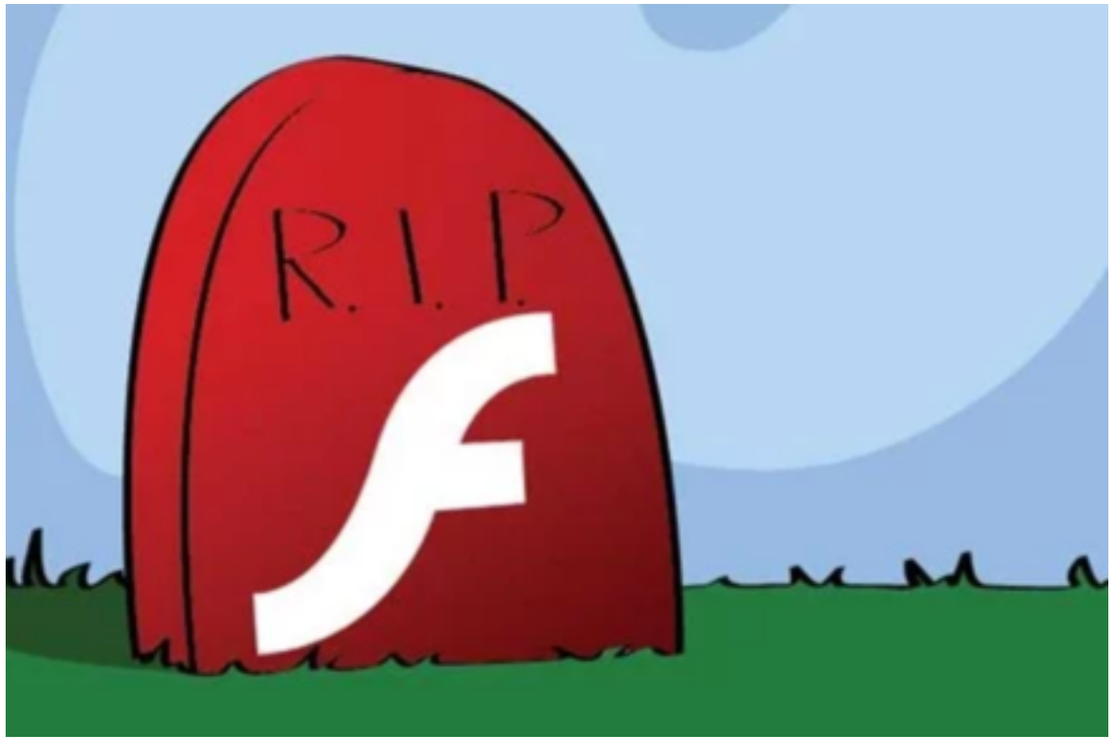
Background

- Hi! I’m Chris
- Work as a Sr. Software Engineer at Artsy
- We <3 Open Source and follow an Open Source by Default policy when possible
- Hiring for many positions! artsy.net/jobs


File Edit View Insert Modify Text Commands Control Debug Window Help
ANIMATOR
CS Live
COLOR SWATCHES
Solid color
TIMELINE
Layer 1
PROPERTIES
LIBRARY
PROJECT
MOTION EDITOR TOOLS
Untitled-1* Untitled-2*
Scene 1
100%
ALIGN TRANSFORM INFO
Align:
Distribute:
Match size:
Space:
Align to stage
SCENE
Scene 1
MOTION PRESETS
Preview not available
32 items
Default Presets
Custom Presets
Apply
Animation
on
FLASH

Started my career as a Flash / ActionScript developer and was quite happy with the technology but was forced to switch paths when iOS arrived


"The mobile era is about low power devices, touch interfaces and open web standards – all areas where Flash falls short."
- Steve Jobs


R.I.P

It was a sad moment, but ultimately it was timely

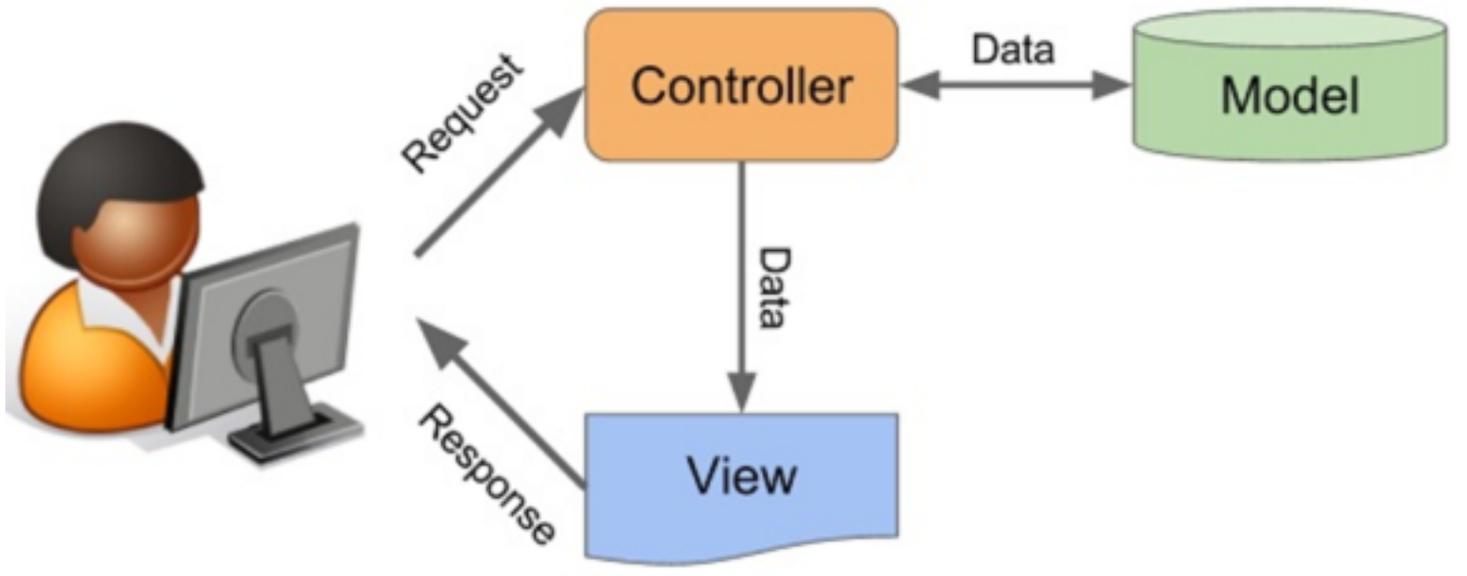

GitHub


JS

Went from ActionScript -> JavaScript and have since experienced most of the trends in front end development, including JavaScript's explosive growth

Have written many large single page apps

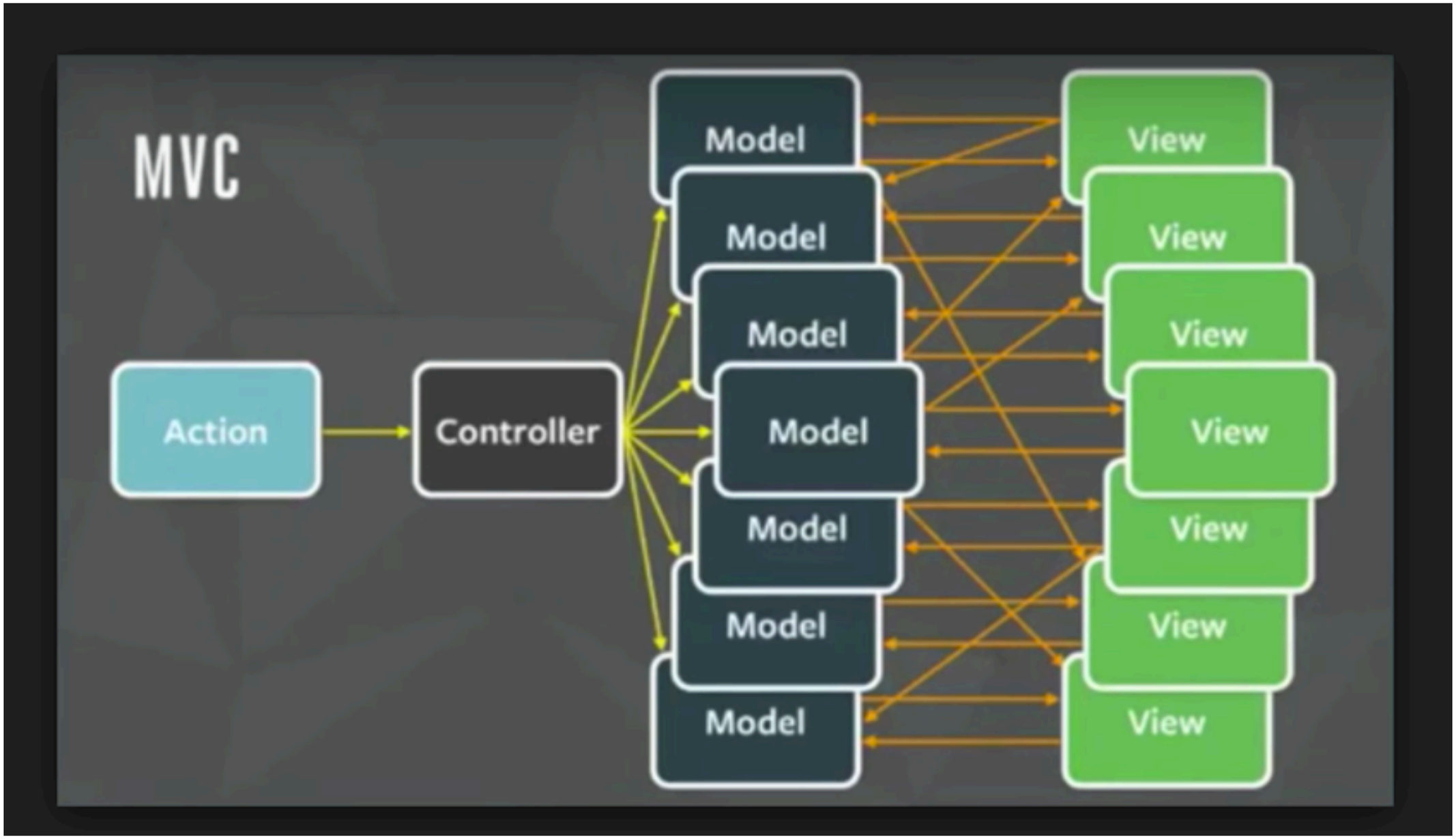
Before React, mostly adhered to (admittedly loose) Model View Controller patterns


Controller
Data
Model
Request
Data
Response
View

View listens for changes to the model, and then updates

Simple right?

No.


MVC
Action
Controller
Model
Model
Model
Model
Model
Model
Model
View
View
View
View
View
View

As things scale out, code becomes difficult to reason about

Views might listen to multiple models, which are passed throughout distant parts of the application

What is updating where??

Since JavaScript is so flexible, it's easy to cut corners (and where there are corners to cut, people will cut them)

Views would listen to models, but also maintain their own internal state

```
class About extends Backbone.View {
  constructor(appModel) {
    this.appModel = appModel

    this._isLoaded = false
  }

  render() {
    this._isLoaded = true

    // stuff happens...
  }

  // ...

  showBio() {
    if (this._isLoaded) {
      this.appModel.set('isShowingBio', true)
    } else {
      // ...
    }
  }
}
```

Now we have the view storing local variable state (isLoaded) which conditionally updates model

Things start diverging, multiplying

Q: What is the canonical source of truth in a given View -- the model, or local state?

A: Who knows?

Should be the model, but in practice it's often local state

¯_(ツ)_/¯

As applications grow they quickly run into issues with scale

Time passes... and suddenly its 2013.

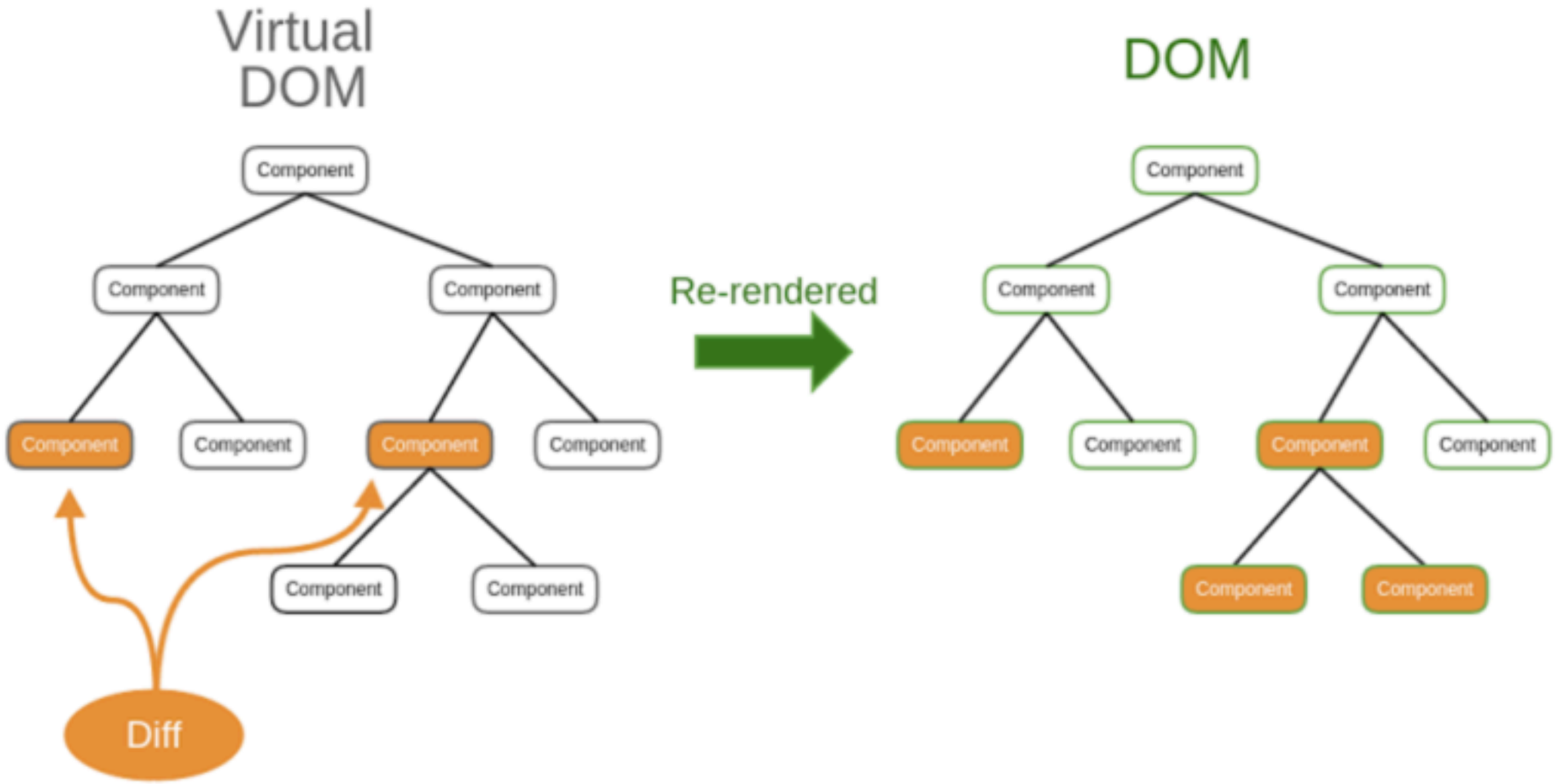
React

A JavaScript library for building user interfaces

```
function App() {
  return (
    <Container>
      <Title>
        Hey React!
      </Title>
    </Container>
  )
}
```

Back then it was understood as the “View" in "MVC"

An early selling point was the idea of a "Virtual DOM"


Virtual DOM
Component
Component
Component
Component
Component
Component
Component
Component
Diff
Re-rendered
DOM
Component
Component
Component
Component
Component
Component
Component
Component

React operations mutate the virtual DOM and only diffs with the real DOM are applied

This makes it really fast.

(The VDom is cool and people often talk about it, but its not the main point. We'll get to that.)

How you handle data is up to you

When released there was consensus around separation of concerns

There was a JS file, and a corresponding template

Backbone.js was king, and Angular.js was growing in popularity


BACKBONE.JS


ANGULAR

Both advocated this separation

```
// template.handlebars
<div>
  Hello my name is {{name}} and I live in {{location}}
</div>

// About.js
class About extends Backbone.View {
  template = viewTemplate

  render() {
    this.$el.html(this.template({
      name: 'Chris',
      location: 'Port Townsend'
    }))
  }
}
```

React advocated getting rid of this separation

```jsx
function App() {
  return (
    <div>
      <Header title='React Presentation' />

      <Navigation>
        {navItems.map(item => (
          <NavItem label={item.label} />
        ))}
      </Navigation>
    </div>
  )
}
```

"XML....omg"

JSX, a superset of JavaScript

Also JSX:

```
function Hello() {
  return (
    <html>
      <body>
        <h1>
          Hello there!
        </h1>
        <p>
          This is JSX, but its also pretty much indistinguishable
          from regular HTML.
        </p>
      </body>
    </html>
  )
}
```

Ben Alman
@cowboy

Follow

Really? Facebook React demo advocating storing HTML in your JavaScript. Which is a HUGE step back in terms of maintainability. #jsconf #wtf

2:21 PM - 29 May 2013 from Georgia, USA

What React tried to argue was that an additional template layer actually increases application complexity

Why? Different language, yet shared common abstractions (variables, loops, etc)

These abstractions were limiting, because the templating languages themselves were limited

Because of the limitations, libraries were built up around these languages to add power

This further increased complexity because each lib reinvented the wheel

One of the main points about React (and by association, JSX) is that it's "Just JavaScript"


JS

All of the power of JavaScript is available to you

But wont devs continue to cut corners / abuse it / fall into JavaScript's weirdness?

Yes and no.

Yes, because its always possible

But also no, because React introduced something interesting to the front-end world that eliminates the need to cut corners.

Functional Programming

FOR

DUMMIES

```haskell
reciprocal :: Int -> (String, Int)
reciprocal n | n > 1 = ('0' : '.' : digits, recur)
  where
  (digits, recur) = divide n 1 []

divide :: Int -> Int -> [Int] -> (String, Int)
divide n c cs | c `elem` cs = ([], position c cs)
              | r == 0      = (show q, 0)
              | r /= 0      = (show q ++ digits, recur)
  where
  (q, r) = (c*10) `quotRem` n
  (digits, recur) = divide n r (c:cs)

position :: Int -> [Int] -> Int
position n (x:xs) | n==x      = 1
                  | otherwise = 1 + position n xs

showRecip :: Int -> String
showRecip n =
  "1/" ++ show n ++ " = " ++
  if r==0 then d else take p d ++ "(" ++ drop p d ++ ")"
  where
  p = length d - r
  (d, r) = reciprocal n

main = do
  number <- readLn
  putStrLn (showRecip number)
  main
```

huh?

FP takes many forms, but at the root we're just talking about one thing

Functions

```
function sayHello() {

}

function sayGoodbye() {

}
```

The debate is an old one:

Functional Programming vs Object Oriented Programming

Object Oriented Programming **vs**. Functional Programming. ... **OOP** says that bringing together data and its associated behavior in a single location (called an "object") makes it easier to understand how a program works. **FP** says that data and behavior are distinctively different things and should be kept separate for clarity. May 12, 2015

In 2013, the front end world (and programming world in general) was mostly united around OOP

Classes, Inheritance, Interfaces etc.

```
class Page extends Backbone.View {
  // ...
}

class About extends Page {
  // ...
}
```

If a dog is a mammal and a mammal is an animal then that makes a dog an animal, yah? yah.

If I'm only concerned about the dog barking, why do I need to know anything about it being a warm blooded vertebrate?

TMI (too much information)

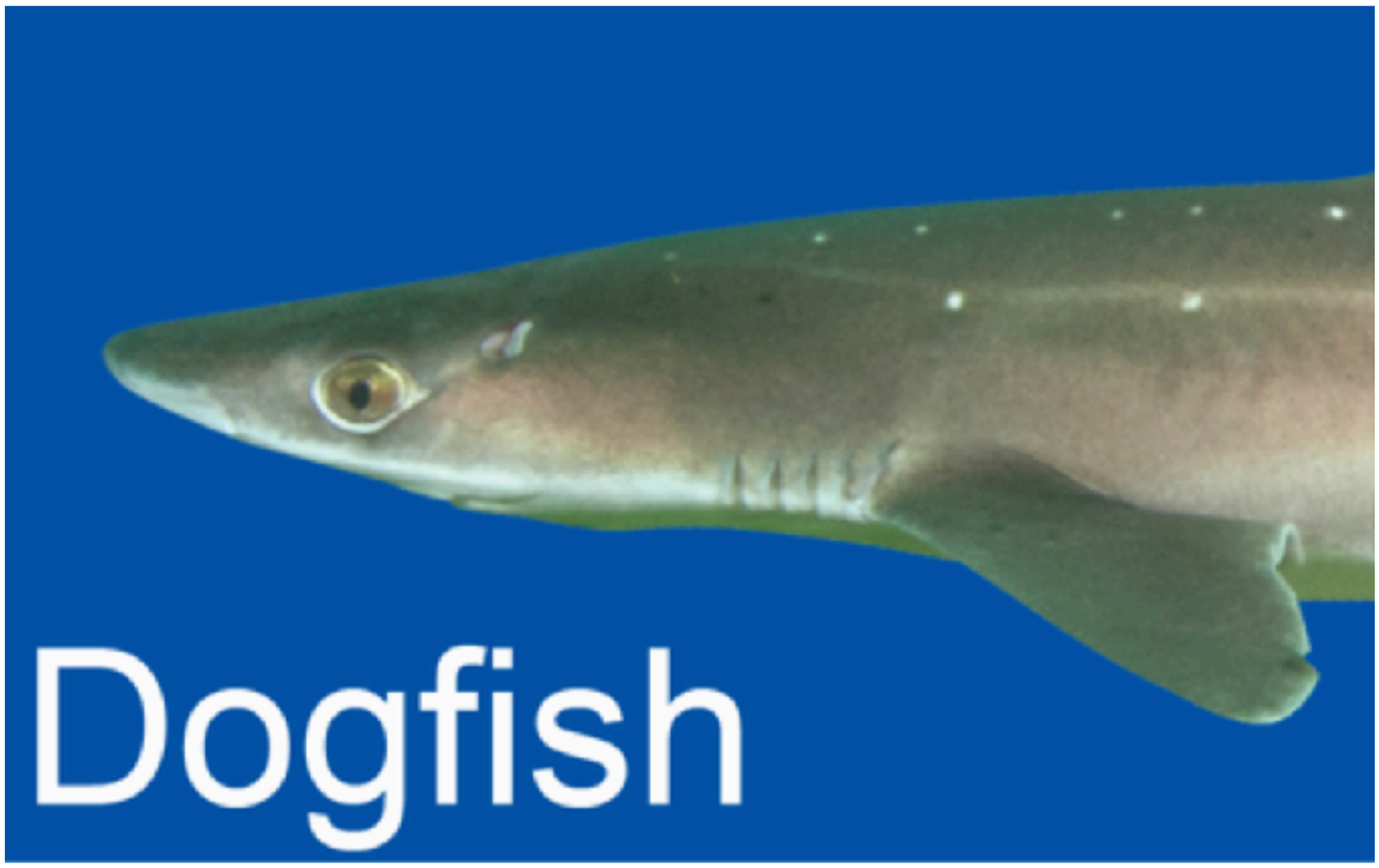
To repeat: where there are corners to cut, people will cut them

At some point in an application lifecycle, a dev will make decisions about things

Sometimes these decisions are good and sometimes they’re bad

Sometimes, oftentimes, they'll decide that the best way to make the dog stop barking is to make it cold blooded and aquatic

the



No no no....

All of this is confusing.

Just give the lil guy a treat!

```
function angryDog(food) {
  // Barking....
}

function withTreat() {
  // Delicious morsels...
}

const happyDog = angryDog(withTreat()) // => Meow!
```

React bills itself as a library for building user interfaces

But what it really is is a different way of programming

FP in a nutshell:

```
const addTwoNumbers = (x, y) => x + y

addTwoNumbers(2, 3) // => 5
addTwoNumbers(2, 3) // => 5
addTwoNumbers(2, 3) // => 5
addTwoNumbers(2, 3) // => 5 ... will always be true.
```

Idempotency: Calling a function multiple times with the same input will always return the same output

Discourages

“side effects"

A side effect means that one thing can potentially change any number of things, independent of the result returned

```
let total = 0;

function setTotalTo10() {
  total = 10
}

function addTwoNumbers (x, y) {
  setTotalTo10() // side-effect

  return x + y
}

addTwoNumbers(2, 3) // => 5, BUT, total is now 10 too
```

Rather than thinking of one thing you now have to potentially think about `*n*` number of things

OO pattern/principle	FP pattern/principle
• Single Responsibility Principle	• Functions
• Open/Closed principle	• Functions
• Dependency Inversion Principle	• Functions, also
• Interface Segregation Principle	• Functions
• Factory pattern	• Yes, functions
• Strategy pattern	• Oh my, functions again!
• Decorator pattern	• Functions
• Visitor pattern	• Functions □

work·ing mem·o·ry

noun PSYCHOLOGY

the part of short-term memory that is concerned with immediate conscious perceptual and linguistic processing.

- COMPUTING
  an area of high-speed memory used to store programs or data currently in use.

Working memory is your baby and needs care

It grows over time in ways that are not necessarily immediate

Without cognizance around side-effects, as your program grows the amount of information in the system increases exponentially

React specifically limits the amount of information in the system by enforcing a simple pattern

???

Data that flows into a React component from the outside as props cannot change

It is "Immutable"

im·mu·ta·ble

/i(m)ˈmyo͞odəb(ə)l/

adjective

unchanging over time or unable to be changed.
"an immutable fact"
synonyms: **fixed**, **set**, rigid, inflexible, **permanent**, **established**, carved in stone; More

When props are passed to a component, one should get the same result back every time, unless explicitly designed otherwise

```
<Person name='Chris' /> => <div>{Chris}</div>
```

What happens when you try to change it?

```jsx
function App(props) {
  props.title = 'Angular Presentation' // Oops! Throws error

  return (
    <div>
      {props.title}
    </div>
  )
}

ReactDOM.render(<App title='React Presentation'/>, document.getElementById('root'))
```

```
▶Uncaught TypeError: Cannot assign to read only property 'title' of
object '#<Object>'
    at LargeSlider (Slider.tsx:63)
    at mountIndeterminateComponent (react-dom.development.js:13272)
    at beginWork (react-dom.development.js:13711)
    at performUnitOfWork (react-dom.development.js:15741)
    at workLoop (react-dom.development.js:15780)
    at HTMLUnknownElement.callCallback (react-dom.development.js:100)
    at Object.invokeGuardedCallbackDev (react-dom.development.js:138)
    at invokeGuardedCallback (react-dom.development.js:187)
    at replayUnitOfWork (react-dom.development.js:15194)
    at renderRoot (react-dom.development.js:15840)
▶Uncaught TypeError: Cannot add property title, object is not
extensible
    at LargeSlider (Slider.tsx:63)
    at mountIndeterminateComponent (react-dom.development.js:13272)
    at beginWork (react-dom.development.js:13711)
    at performUnitOfWork (react-dom.development.js:15741)
    at workLoop (react-dom.development.js:15780)
    at HTMLUnknownElement.callCallback (react-dom.development.js:100)
    at Object.invokeGuardedCallbackDev (react-dom.development.js:138)
    at invokeGuardedCallback (react-dom.development.js:187)
    at replayUnitOfWork (react-dom.development.js:15194)
    at renderRoot (react-dom.development.js:15840)
▶The above error occurred in the <LargeSlider>      react-dom.develop
component:
    in LargeSlider
```

When this scales out to a real application the benefits become immediately apparent

It provides a sense of security

You can be fairly certain that a component will always behave a certain way

This makes things very lego-like, and sturdy

Whole classes of common bugs disappear due to how data flows through the tree

Now... let’s return to 2013.

React… with its weird JSX "XML" syntax… is utterly ignored and laughed at.

Ben Alman
@cowboy

Follow

Facebook: Rethink established best practices™

Reply Retweet Favorite More

10 RETWEETS	1 FAVORITE

5:40 PM - 29 May 13

These concerns are mostly surface level

But had anyone outside of Facebook really taken a good hard look at it?

Enter David Nolen,

Dec 2013

Idée Fixe • David Nolen - https://www.youtube.com/watch?v=lzXHMy4ewtM

David Nolen is the lead maintainer of ClojureScript, a compile-to-JavaScript language on top of Clojure

Clojure, a lisp, is a dialect of Java and built on top of functional programming principles

When Nolen saw React he noticed the similarities between this model and FP

He took it for a spin, and something jumped right out

<- React

Backbone ->

He realized that React was more amenable to global optimization due to its design

And paired React with ClojureScript's data structures and discovered something profound

David Nolen
@swannodette

Follow

ClojureScript om based TodoMVC looks 30-40X faster than Backbone.js TodoMVC, which means other JS frameworks left completely in the dust

5:36 PM - 14 Dec 2013

This is what’s known as a "Zen Moment"

30-40x… How is this possible?

Merrick Christensen @iammerrick · 14 Dec 2013

@swannodette How does it get that kind of performance increase? Backbone.js is pretty minimal when it comes to what its doing, no?

3

David Nolen @swannodette · 14 Dec 2013

.@iammerrick most JS MVCs have a completely busted design. Worry not details forthcoming #reactjs

2 6 11

React can do exactly the amount of work it needs to, and nothing more

shouldComponentUpdate

```
shouldComponentUpdate: (prevName, nextName) => {
  if (prevName !== nextName) {
    return true
  } else {
    return false
  }
}
```

Nolen took advantage of this feature…

…And published a blog post

The Future of JavaScript MVC Frameworks

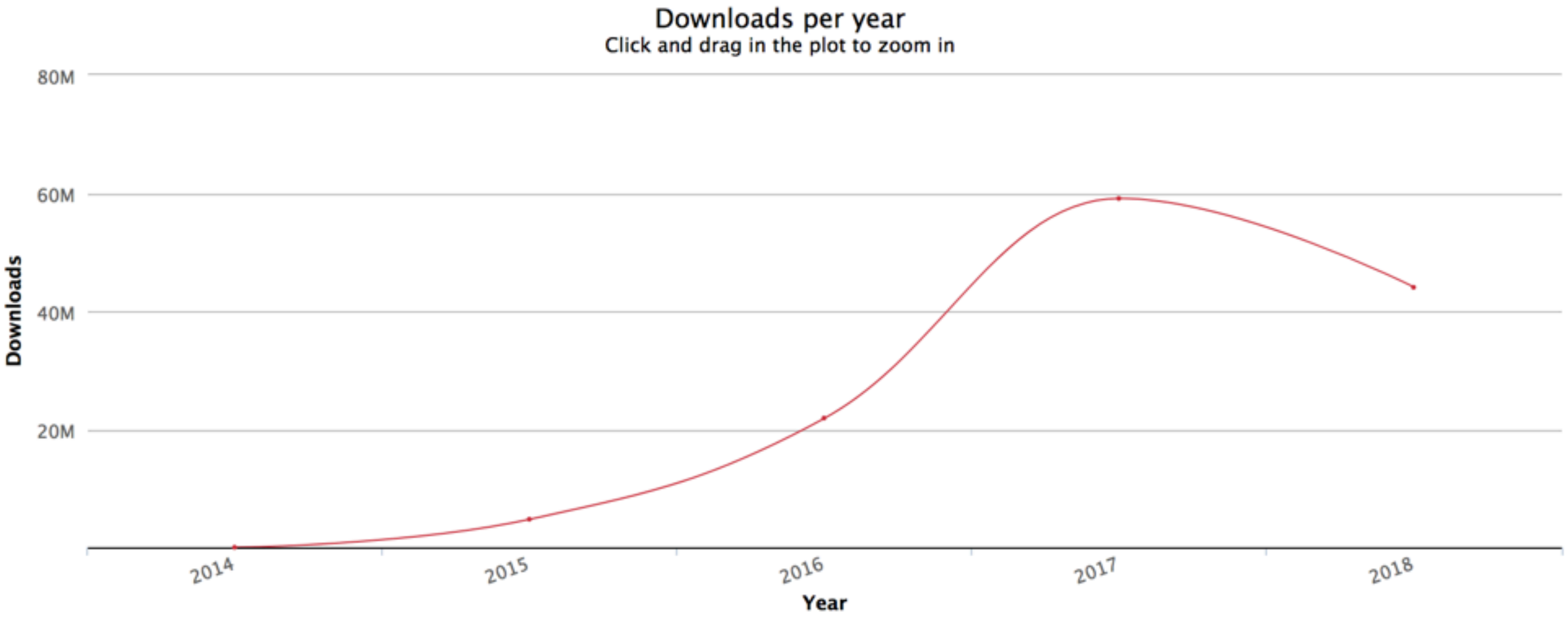
17 DECEMBER 2013

The world took notice


National Salary Trend from Indeed.com
Front End Developer
Salary Index
1.30
1.20
1.10
1.00
Jul '12
Oct '12
Jan '13
Apr '13
Jul '13
Oct '13
Jan '14
Apr '14

While it can't be proven, I'm pretty certain that his blog post gave us all a huge pay raise

From that point on React was seen as viable tech


Downloads per year
Click and drag in the plot to zoom in
80M
60M
40M
20M
Downloads
2014
2015
2016
2017
2018
Year

https://www.youtube.com/watch?v=MTawgp3SKy8

Fast forward to 2018.

React is eating the world

When Facebook released React they billed it as a tool for building user interfaces

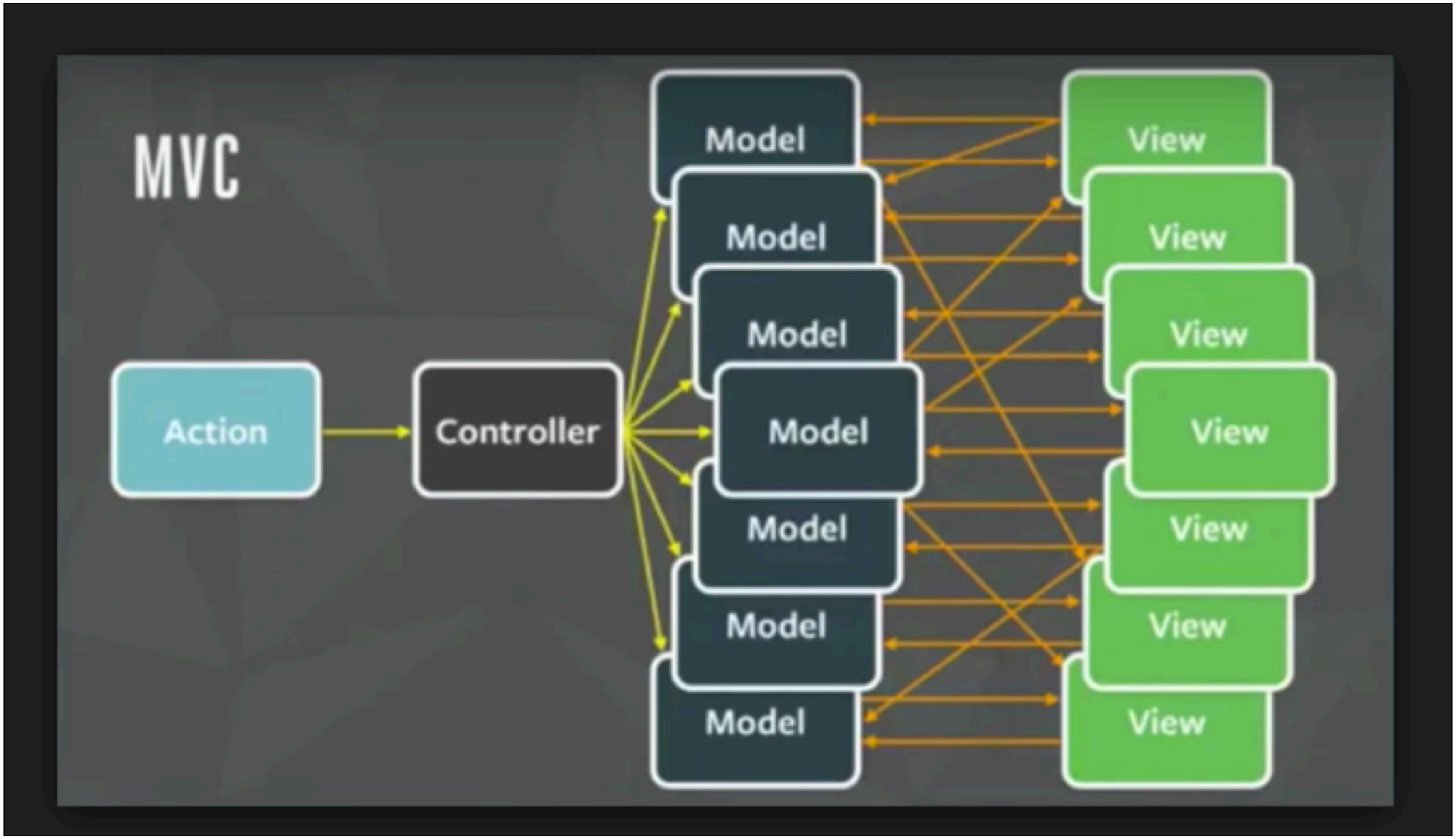
React

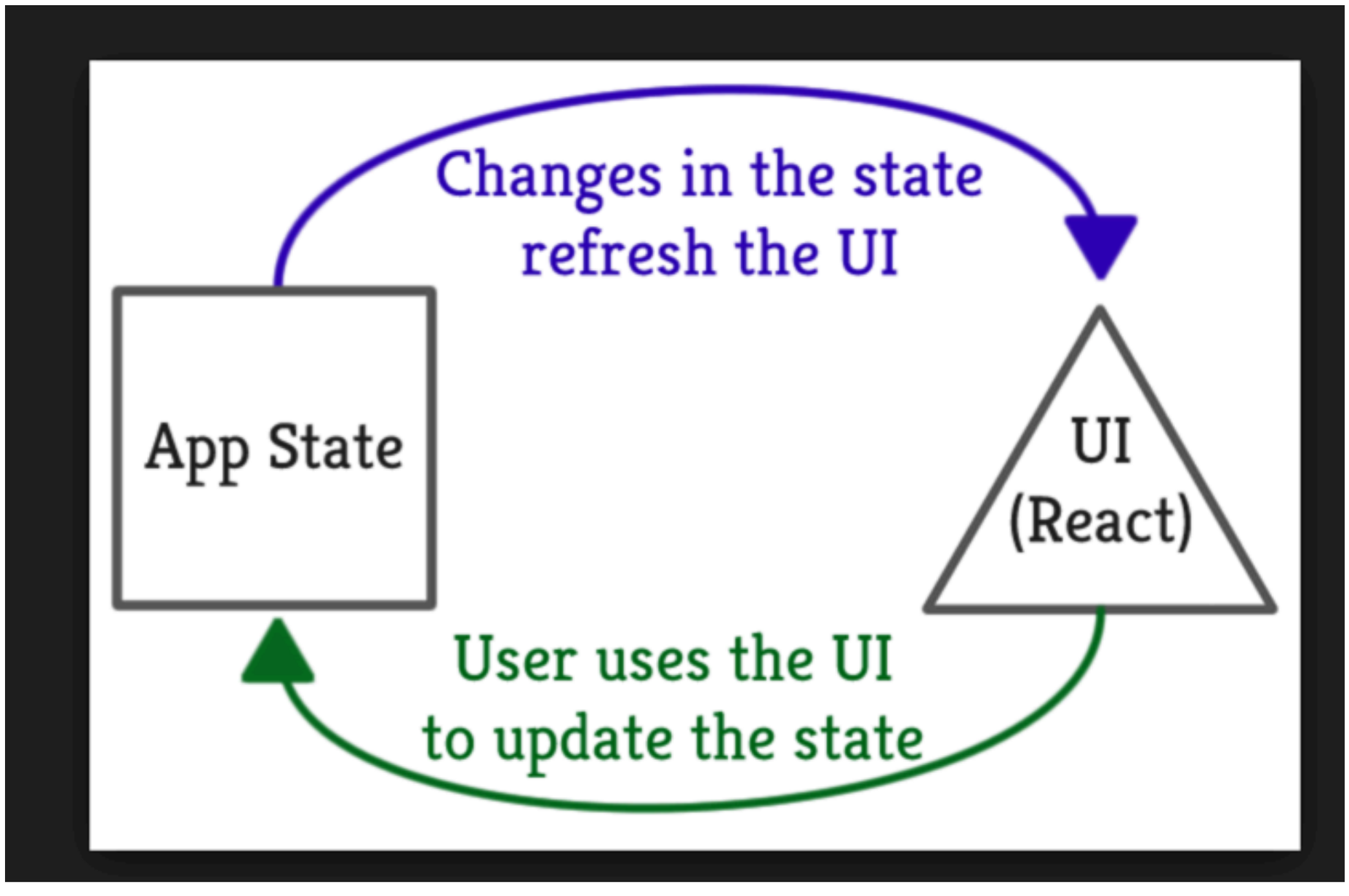
A JavaScript library for building user interfaces

Get Started

Take the Tutorial

What they really released was an old idea that advocated for simplicity over complexity


MVC
Action
Controller
Model
Model
Model
Model
Model
Model
Model
View
View
View
View
View
View


Changes in the state
refresh the UI
App State
UI
(React)
User uses the UI
to update the state

That triangle is not a single View, but typically an entire application tree.

It unlocked the frontend for whole new groups of people

Before, backend devs wouldn't touch the frontend because it was so unpredictable

Now much more common to see "full stack" developers

New ideas from many domains are streaming into user interface architectural design

But this goes beyond the web

The unidirectional data flow model is now being applied all over

It’s simplifying our understanding of programming *in general* by bringing awareness to issues around mutability

It's a programming philosophy that frees one's mind to think about larger pictures

I highly encourage you all to give it a try!


React

Thank you!

Questions?

Resources:

https://github.com/facebook/create-react-app
(the quickest way to get started with React)

https://github.com/damassi/react-styled-components-for-designers

**(A project scaffold I set up that brings together
some best-in-class frontend / editor tools and provides
some advice for designers looking to learn
React and Styled Components)**