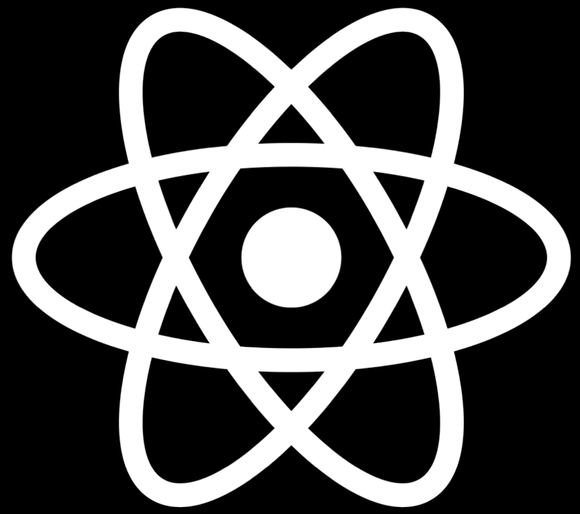
# REACT.JS

## Rethinking UI Development

2.0



React day - Verona - Oct 30th, 2015
by Stefano Ceschi Berrini @stecb

```
{
  "name" : "Stefano Ceschi Berrini",
  "location": "Padova",
  "degree": "CS @ unipd",
  "work" : [ "@Timerepublik" ],
  "tech" : [
    "JavaScript", "React", "Ruby"
  ],
  "interests" : [
    "Family", "Friends",
   "Music (Progressive Metal FTW)",
    "Mountains", "Cooking", "Sports"
  ],
  "titles" : [ "Software engineer" ],
  "website": "stecb.ninja"
}
```

TIMEREPUBLIK

# PREFACE: ES2015

babel you should use, young padawan!

# PREFACE: ES2015

```
// scopes the variable to the nearest **block** {}. No hoisting
let foo = 'bar';

// constant reference to the value, we shouldn't redefine it! And
MUST be initialised. Same scoping rules of let
const pi = 3.14;
const arr = [];

// we can change referenced value in this case
arr.push('hey');
// but we can't redefine constants i.e.
// arr = []
// or
// pi = 5

// String interpolation\templates
let name = 'John Petrucci';
let instrument = 'guitar';
console.log(`${name} can play ${instrument} faster than you`);
```

# PREFACE: ES2015

```
// arrow functions
let arr = [1,2,3];
let doubleArr = arr.map(n => n * 2);

// arrow function + lexical this
let guitars = ['music man', 'ibanez', 'taylor'];
let guitarsShop = {
  location: 'Montebelluna',
  name: 'EsseMusic',
  // shorthand assignment => guitars: guitars
  guitars,
  // method
  listGuitars() {
    this.guitars.forEach(guitar => {
      console.log(`${this.name} in ${this.location}
                  has ${guitar} guitars`);
    });
  }
}
guitarsShop.listGuitars();
```

# PREFACE: ES2015

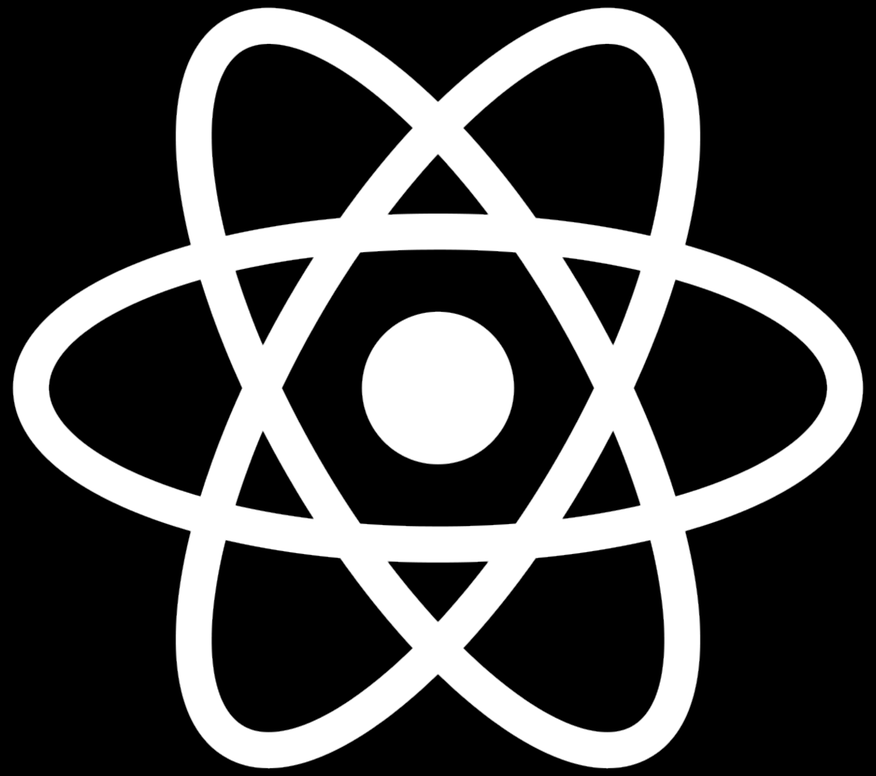
```
// class
class Person {
  // default params
  constructor(name = 'unknown', age = 0, sex = 'whatever') {
    this.age = age;
    this.name = name;
    this.sex = sex;
  }
  displayInfo(a){
    console.log(this.name, this.age, this.sex);
  }
}

// subclass
class Female extends Person {
  constructor(name, age){
    // super call
    super(name, age, 'f');
  }
  yell(what){
    super.displayInfo();
    setInterval(() => console.log(what), 1000);
  }
}
let myWife = new Female('Deborah', 28);
console.log(myWife.yell('wash your own cup and dishes please!'));
```

# PREFACE: ES2015

```
// modules
// inside **foo.js**, export something
export function isOdd(n) {
  return n%2 !== 0;
}
export var threshold = 30;

// On another file **bar.js** import everything from foo.js
import * from 'foo';

let height = window.innerHeight - threshold;
let arr = [1, 2, 3, 4];
console.log(arr.map(n => isOdd(n)));

export default height;
```

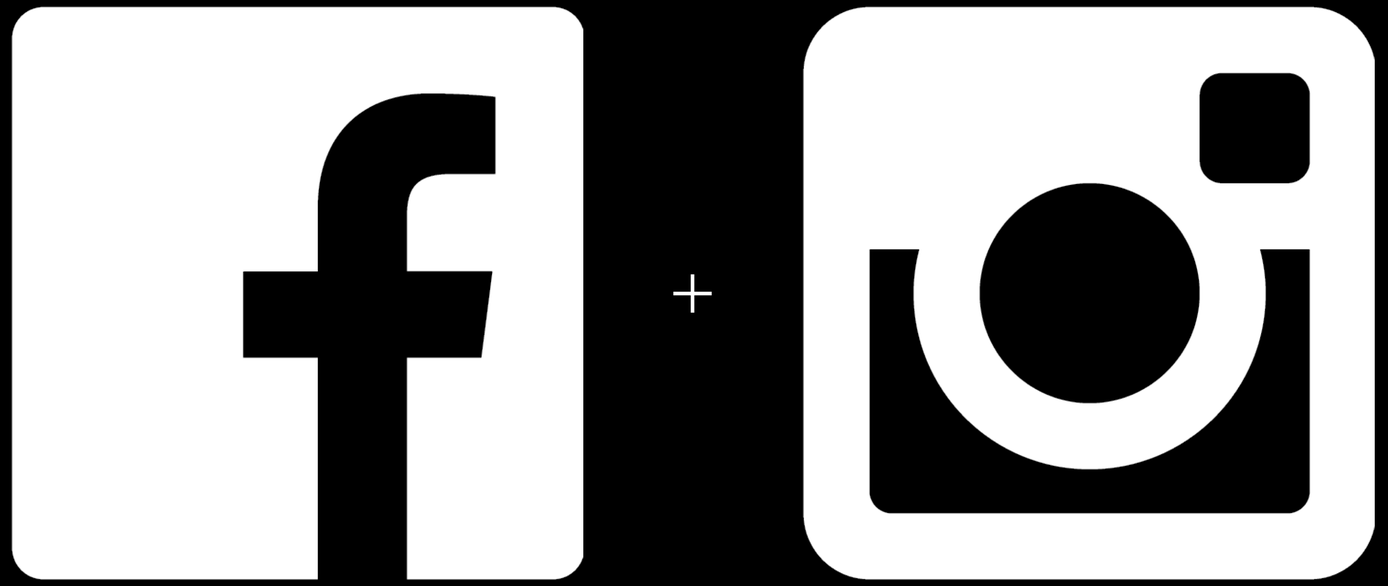
# REACT

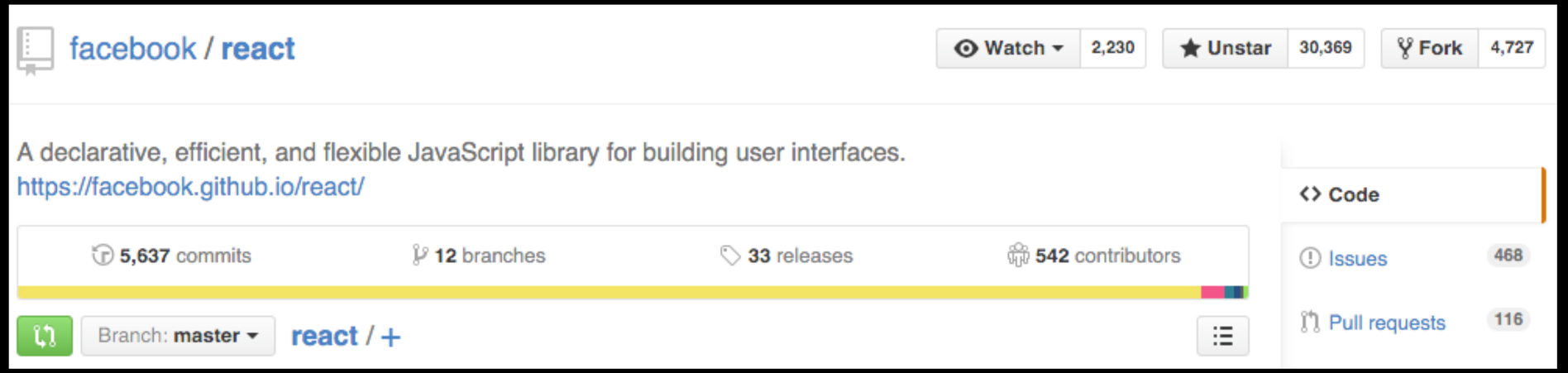
Let's finally talk about this library!



# WHO

The minds behind React





# WHAT

It’s a JavaScript library for the the UI

M V C

it automa(g|t)ically keeps the interface up-to-date when data changes

# WHY

Because of:

# IS SOMEONE USING IT?


+ AirBnB, Atlassian, Apple, Imgur, Microsoft, Paypal, Pivotal Labs, Reddit, Whatsapp, Wired…

# YOU LEARN JAVASCRIPT!

# Forget about jQuery

React is **Declarative**: code that **describes** what you want

VS

**Imperative**: **how** you want something, step by step

# 3 BUILDING BLOCKS OF REACT

Components (+ JSX)
Virtual DOM
One-way data binding

# COMPONENTS

A component is a React class. It can take input data, it can have a state and you can:

- define methods
- render a tree of functions

When you add a component to your UI, you're just invoking a function.

# COMPONENTS

React HTML elements i.e. <button>
Custom react components i.e. <Profile>
Collections/composition of the above

# COMPONENTS

Think about components as
state machines

# COMPONENTS

```
class Ciao extends React.Component {
  render() {
    return (
      <div>Ciao, {this.props.name}</div>
    );
  }
}

ReactDom.render(<Ciao name='Stefano' />, document.getElementById('wrapper'))
```

Show example


WAT??
Ciao, {this.props.name}
TAGS Inside JS????

# COMPONENTS, JSX

JS syntactic sugar: a concise and familiar syntax for defining tree structures with attributes.

XML like elements => function + args

# COMPONENTS, PROPERTIES

Props are the options/configuration of a component, they are data passed from parents to children and they are immmutable.

If you think a property of a component could changed over time, then that should be part of the state.

# COMPONENTS, PROPERTIES

```
export class Box extends React.Component {
  render() {
    const list = this.props.list.map(item => <li key={item}>{item}</li>);
    return (
      <div>
        <h1>{this.props.title}</h1>
        <ul>
          {list}
        </ul>
      </div>
    );
  }
}

ReactDOM.render(<Box title="Cool Box" list={['item 1', 'item 2', 'item 3',
‘item N’]} />, document.getElementById('wrapper'))
```

# COMPONENTS, STATE

State is mutable and should contain data that a component's event handlers may change to trigger a UI update (i.e. User interactions, Server responses etc)

setState(data, callback)

State is optional and you should avoid it as much as possible, to reduce complexity!

# COMPONENTS, STATE

```
class BoxWithState extends React.Component {
  constructor(props) {
    super(props);
    this.state = {hasDetailsVisible: false};
  }
  handleToggle() {
    this.setState({
      hasDetailsVisible: !this.state.hasDetailsVisible
    });
  }
  render() {
    const list = this.props.list.map(item => <li key={item}>{item}</li>);
    const detailsStyle = {display: this.state.hasDetailsVisible ? 'block' :
                          'none'};
    return (
      <div>
        <h1>{this.props.title}</h1>
        <button onClick={() => this.handleToggle()}>toggle details</button>
        <ul style={detailsStyle}>
          {list}
        </ul>
      </div>
    );
  }
}
```

# COMPONENTS, EVENTS

Synthetic events. They work identically on every browser

Clipboard, Keyboard, Focus, Form,
Mouse, Touch, UI, Wheel

http://facebook.github.io/react/docs/events.html

# COMPONENTS, EVENTS

```
class ClickableCounter extends React.Component {
  constructor() {
    super();
    this.state = {count: 0};
  }
  increaseCounter() {
    this.setState({
      count: ++this.state.count
    });
  }
  render() {
    return (
      <div>
        <button onClick={() => this.increaseCounter()}>Click here Luke!</button>
        <span>{this.state.count}</span>
      </div>
    );
  }
}
```

# COMPONENTS, DOM

```
class Input extends React.Component {
  componentDidMount() {
    this.refs.myInput.focus();
  }
  render() {
    return (
      <input type='text' ref='myInput' placeholder='look!' />
    );
  }
}
```

# COMPONENTS, COMBINATION

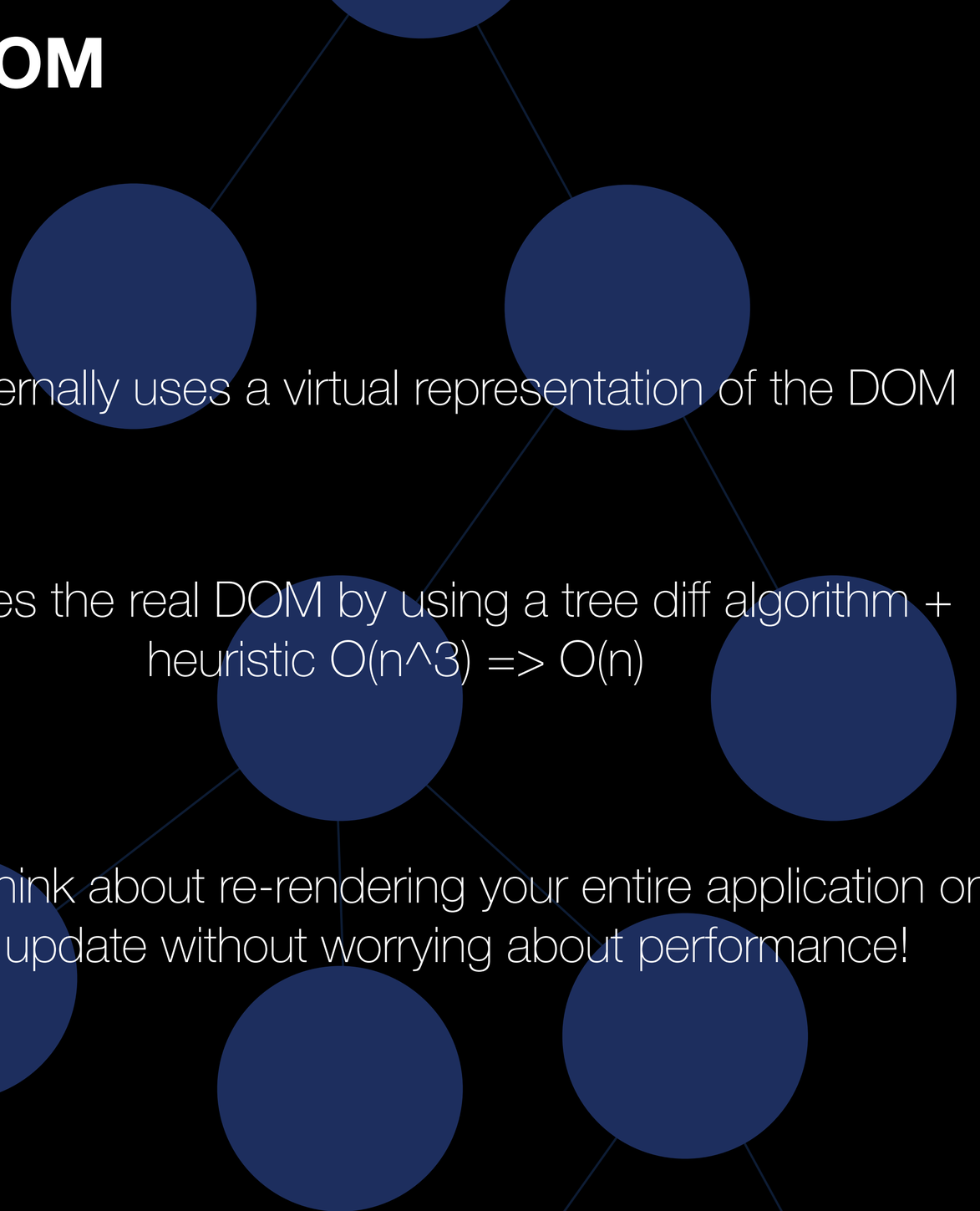
```
class MyCustomComponent extends React.Component {
  render() {
    return (
      <div>
        <h1>{this.props.name}</h1>
        {this.props.children || 'No children :('}
      </div>
    );
  }
}

ReactDOM.render(
  <div>
    <MyCustomComponent>
      <Input />
    </MyCustomComponent>
    <MyCustomComponent />
  </div>
, document.getElementById('wrapper'));
```

# COMPONENTS, LIFECYCLE

```
class MyComponent extends React.Component {
  componentWillMount() {
    // fired before is mounted
  }
  componentDidMount() {
    // fired when component mounted into the DOM
  }
  shouldComponentUpdate(nextProp, nextState) {
    // fired before rendering
    // return true|false depending whether component should update
    // (i.e. if you're sure component won't need to be re-rendered)
  }
  componentWillUnmount() {
    // fired just before the component will be removed from the DOM
    // (useful i.e. if you need to remove some custom event listeners)
  }
  // ...
  render() {
    return (
      <Something />
    );
  }
}
```

http://facebook.github.io/react/docs/component-specs.html

# VIRTUAL DOM

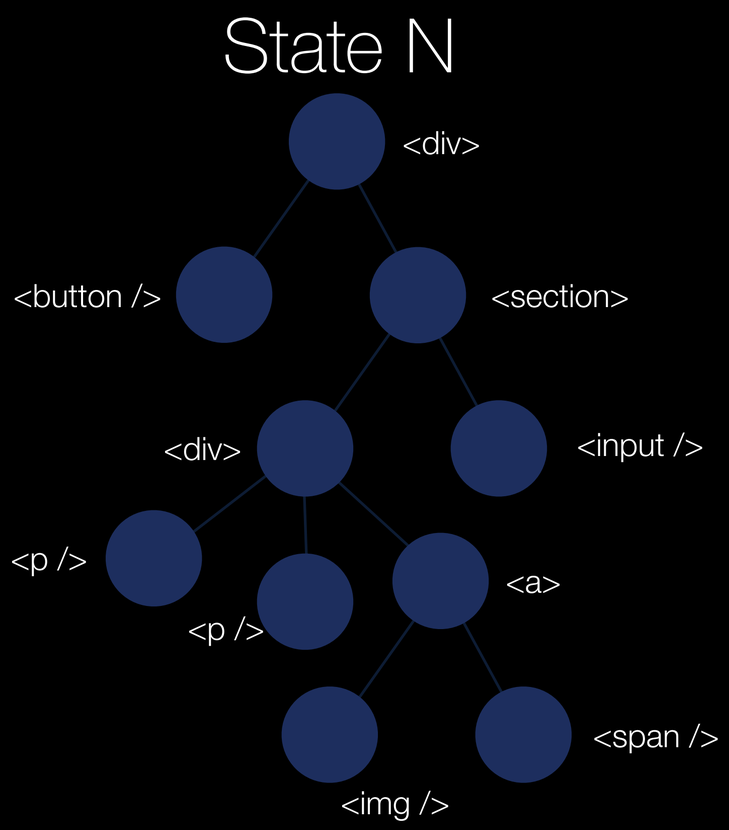


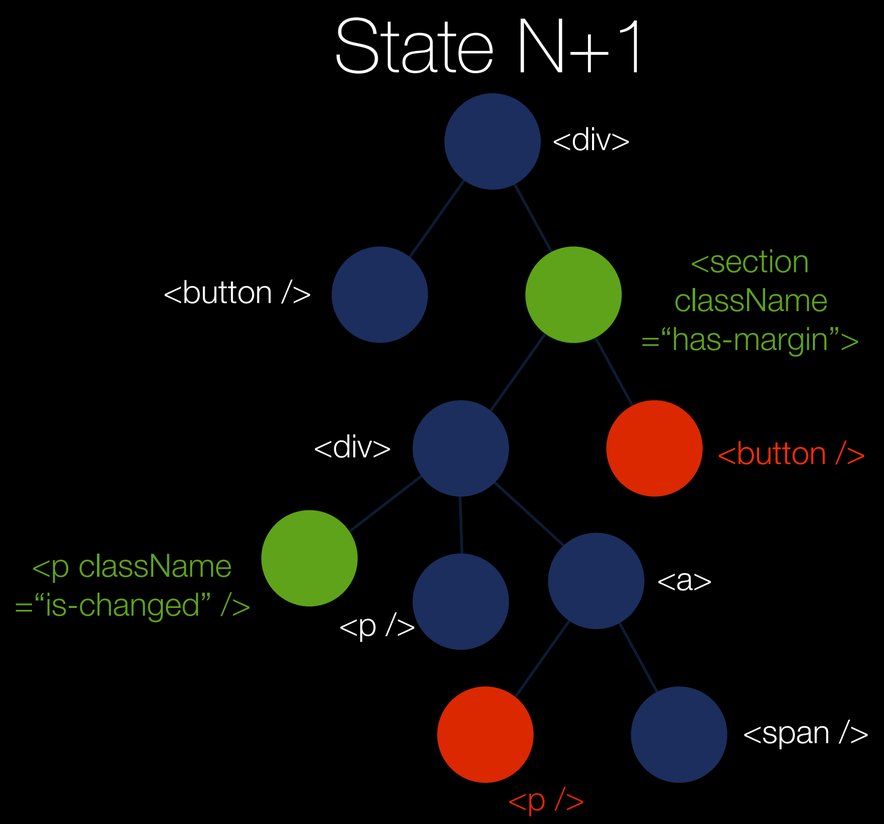
React internally uses a virtual representation of the DOM

It mutates the real DOM by using a tree diff algorithm + heuristic O(n^3) => O(n)

You can think about re-rendering your entire application on every update without worrying about performance!

# VIRTUAL DOM




Changed attributes, just update them on the real DOM

The NodeType has changed, throw away that node (+subtree) and replace it with a new node!

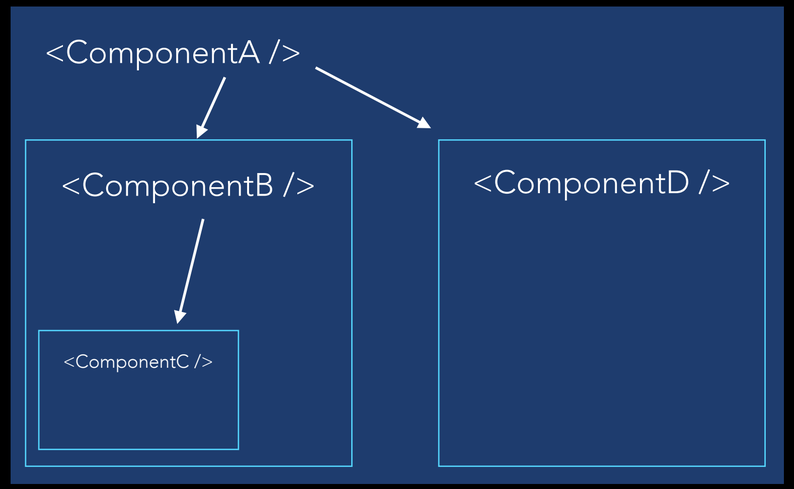
# VIRTUAL DOM

JS execution is FAST, real DOM mutations are SLOW

So, being able to re-render the entire application by mutating
JUST what changed on the UI is the most important thing

https://www.flickr.com/photos/usnavy/5815022252

# ONE WAY DATA BINDING

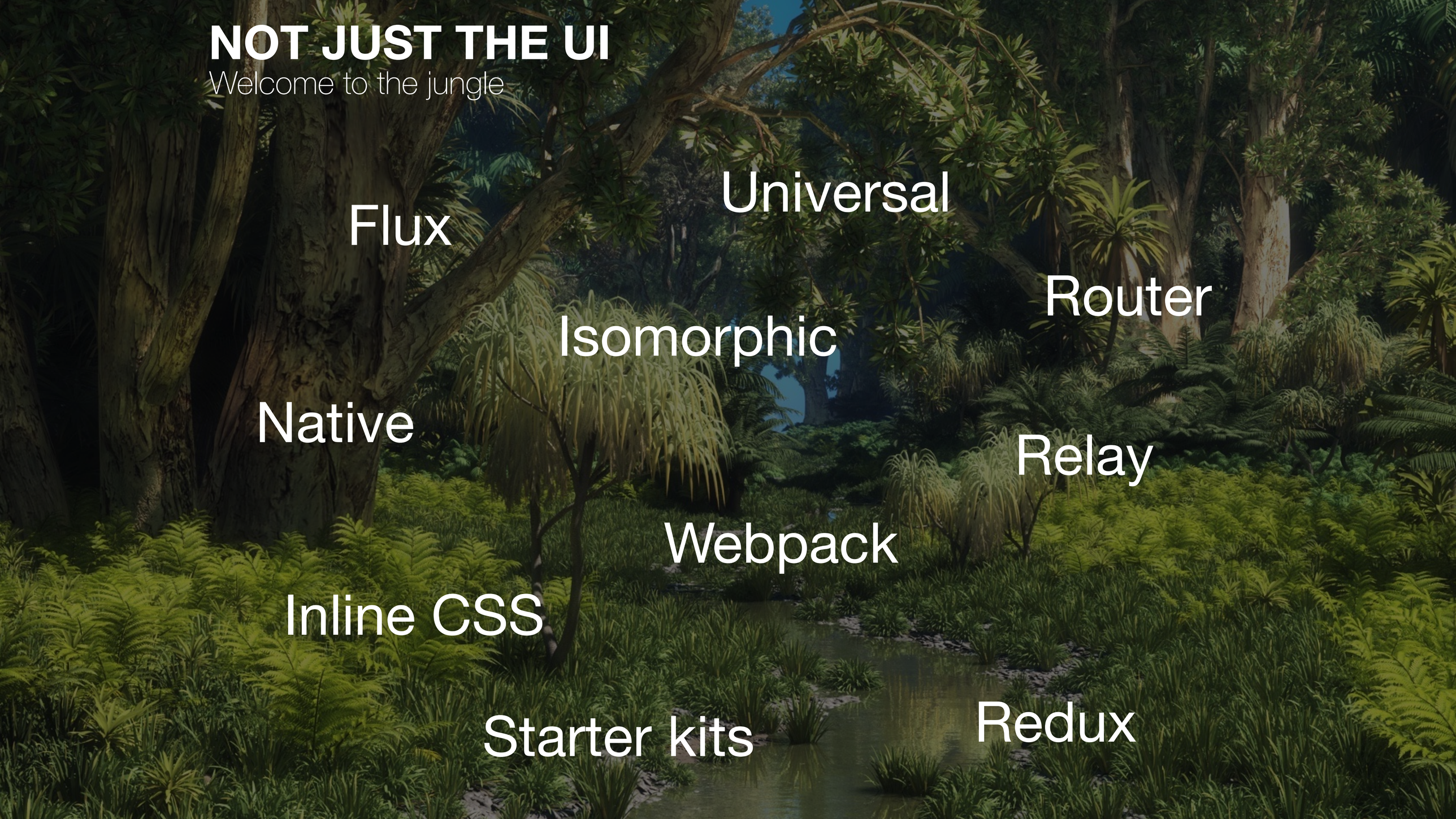



Data (props) flows only in one way. From the Owner to child.

# RECAP

- Easy to learn, small API
- Fast
- Same components, full stack
- Native for Android/iOs
- Virtual DOM
- Easy to reason about apps
- Easy to test components
- Easy to integrate
- Working w/ React is pure fun

- “It’s just the UI”
- No more M and C
- Unclear where state should live
- Ecosystem kinda messy


NOT JUST THE UI
Welcome to the jungle
Flux
Universal
Router
Isomorphic
Native
Relay
Webpack
Inline CSS
Starter kits
Redux

# THANKS

Stefano Ceschi Berrini @stecb

https://github.com/stecb/react_examples