

<WA1/>

2020

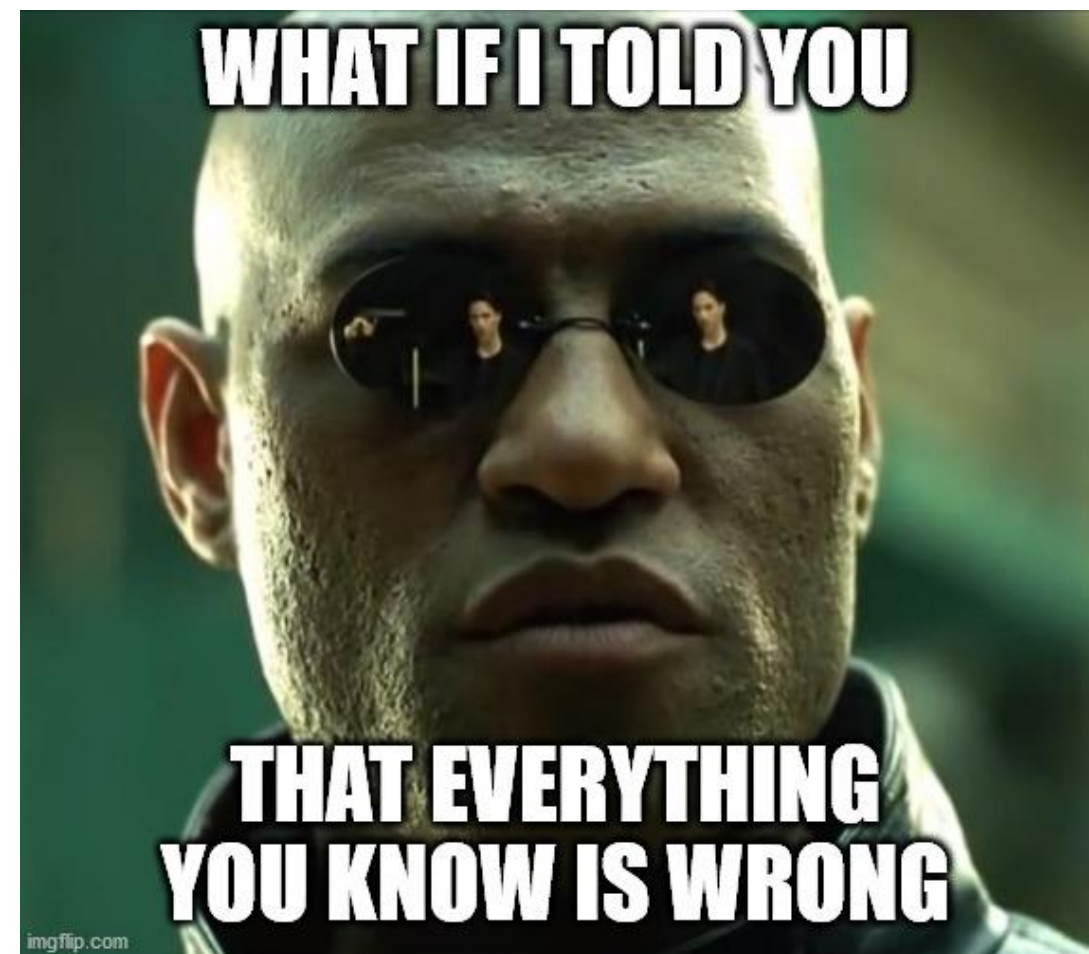
Introduction to React

JS Frameworks to the rescue

Enrico Masala

Fulvio Corno

Luigi De Russis



Goal

- Learn one of the most popular front-end frameworks
 - Basic principles
 - Application architecture
 - Programming techniques
- Leverage the knowledge of JS concepts



React

A JavaScript library for building user interfaces

<https://reactjs.org/>

Why a framework?

- Simplify the browser environment
 - Uniform DOM methods
 - More explicit hierarchy
 - **Higher-level** components than HTML elements
 - **Automatic** processing of events and updates
- Simplify the development methods
 - Predefined programming **patterns** and application architecture
 - Lots of compatible plugins and extensions
 - Explicit and rigid **state** management

Main resources

Learning the main concepts

The screenshot shows the React documentation page for the 'Hello World' tutorial. The page title is 'Hello World'. Below the title, it says 'The smallest React example looks like this:' followed by a code block containing the following code:

```
ReactDOM.render(  
  <h1>Hello, world!</h1>,  
  document.getElementById('root')  
)
```

 Below the code block, it says 'It displays a heading saying "Hello, world!" on the page.' There is a 'Try it on CodePen' link. Below that, it says 'Click the link above to open an online editor. Feel free to make some changes, and see how they affect the output. Most pages in this guide will have editable examples like this one.' There is a 'How to Read This Guide' section. At the bottom, there is a 'Tip' box that says 'This guide is designed for people who prefer learning concepts step by step. If you prefer to learn by doing, check out our step-by-step tutorial. You might find this guide and the tutorial complementary to each other.' On the right side, there is a navigation menu with sections: INSTALLATION, MAIN CONCEPTS (with '1. Hello World' selected), ADVANCED GUIDES, API REFERENCE, HOOKS, TESTING, CONCURRENT MODE (EXPERIMENTAL), CONTRIBUTING, and FAQ.

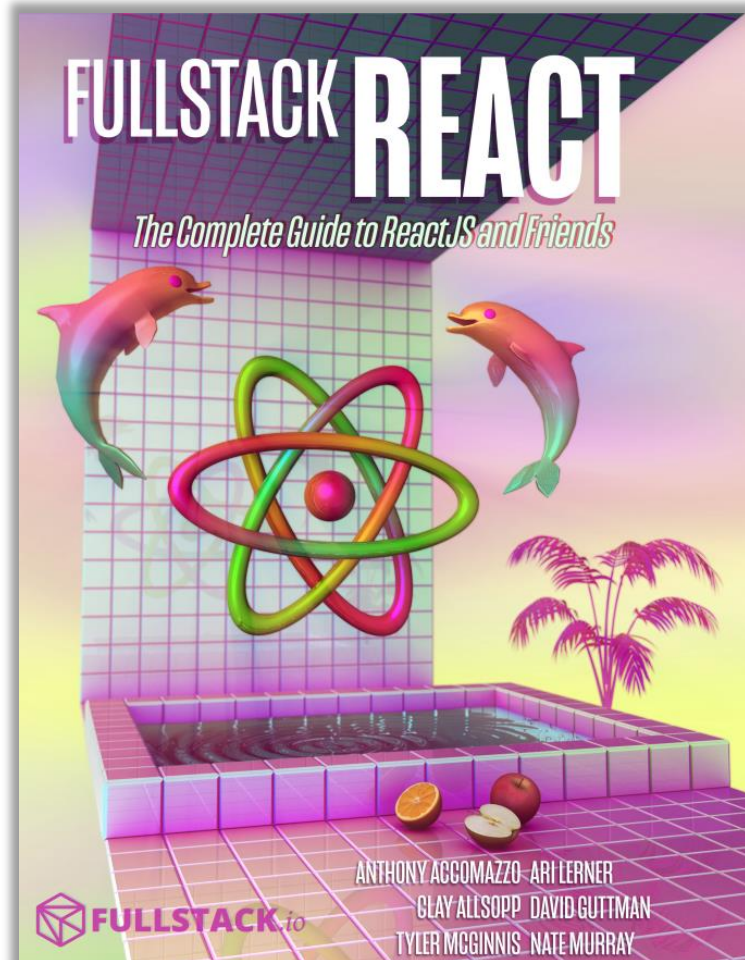
<https://reactjs.org/docs/hello-world.html>

Learn by doing tutorial

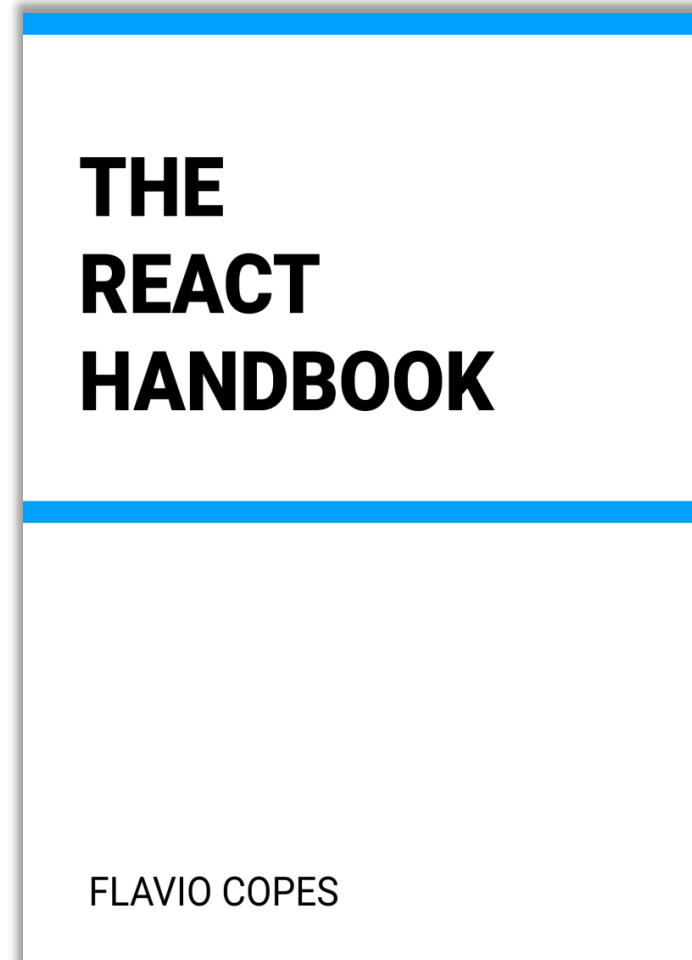
The screenshot shows the React documentation page for the 'Tutorial: Intro to React'. The page title is 'Tutorial: Intro to React'. Below the title, it says 'This tutorial doesn't assume any existing React knowledge.' There is a 'Before We Start the Tutorial' section. Below that, it says 'We will build a small game during this tutorial. You might be tempted to skip it because you're not building games — but give it a chance. The techniques you'll learn in the tutorial are fundamental to building any React app, and mastering it will give you a deep understanding of React.' There is a 'Tip' box that says 'This tutorial is designed for people who prefer to learn by doing. If you prefer learning concepts from the ground up, check out our step-by-step guide. You might find this tutorial and the guide complementary to each other.' Below that, it says 'The tutorial is divided into several sections:' followed by a list of sections: 'Setup for the Tutorial', 'Overview', 'Completing the Game', and 'Adding Time Travel'. At the bottom, it says 'You don't have to complete all of the sections at once to get the value out of this tutorial.' On the right side, there is a navigation menu with sections: TUTORIAL (with 'Before We Start the Tutorial' selected), 'Before We Start the Tutorial' (with sub-items: 'What Are We Building?', 'Prerequisites', 'Setup for the Tutorial', 'Option 1: Write Code in the Browser', 'Option 2: Local Development Environment Help, I'm Stuck!'), 'Overview' (with sub-items: 'What Is React?', 'Inspecting the Starter Code', 'Passing Data Through Props', 'Making an Interactive Component Developer Tools'), 'Completing the Game' (with sub-items: 'Lifting State Up', 'Why Immutability Is Important', 'Function Components', 'Taking Turns', 'Declaring a Winner'), and 'Adding Time Travel' (with sub-items: 'Storing a History of Moves', 'Lifting State Up, Again', 'Showing the Past Moves', 'Picking a Key', 'Implementing Time Travel', 'Wrapping Up').

<https://reactjs.org/tutorial/tutorial.html>

Main resources

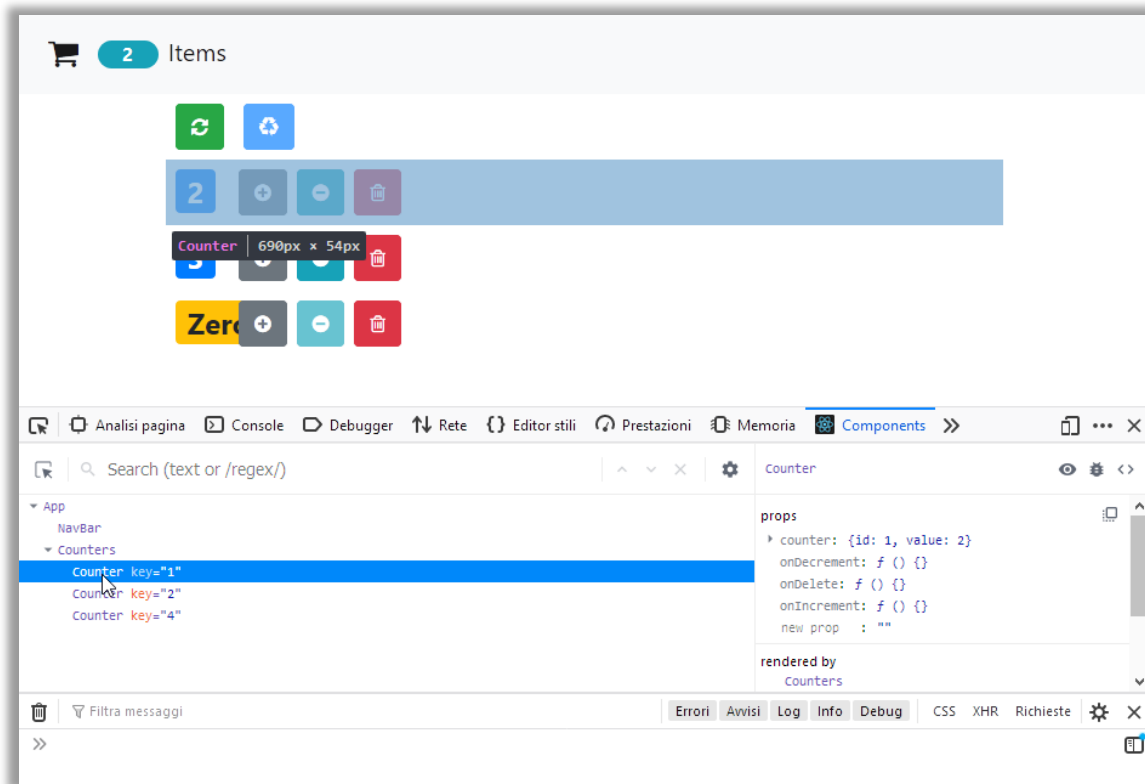


<https://www.newline.co/fullstack-react/>



<https://flaviocopes.com/page/react-handbook/>

Browser Development Tools



2/



chrome web store



React Developer Tools

Offered by: Facebook

★★★★★ 1,255 | Developer Tools | 2,000,000+ users

<https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi?hl=en>



React Developer Tools

by React

<https://addons.mozilla.org/en-US/firefox/addon/react-devtools/>



The React Handbook, Flavio Copes

<https://flaviocopes.com/page/react-handbook/>

A first high-level run about the main design concepts in React

DESIGN PRINCIPLES

React is Declarative

- Never explicitly manipulate the DOM
- Never explicitly define the order of operations
- Just define how each component is going to render itself

React Key Concepts

- Functional design approach
- Components
- Re-render everything on every change
- Virtual DOM
- Synthetic Events
- Controls the *state* of the application

React is Functional

- UI Fragment = $f(\text{state}, \text{props})$
- Many components don't need to manage state
- UI Fragment = $f(\text{props})$
 - Idempotent
 - Immutable
- Jargon note: props = properties

Immutability

- Reacts exploits **Immutability** of objects, for ease of programming and efficiency of processing
- Component **'props'** are immutable (read-only by the component)
- Component **'state'** is not directly mutable (can be changed only through special calls)
- Functions are **'pure'** (have no side-effects besides computing the return value)
 - Idempotency (re-rendering the same component always yields the same result)
 - Predictability

Re-Rendering

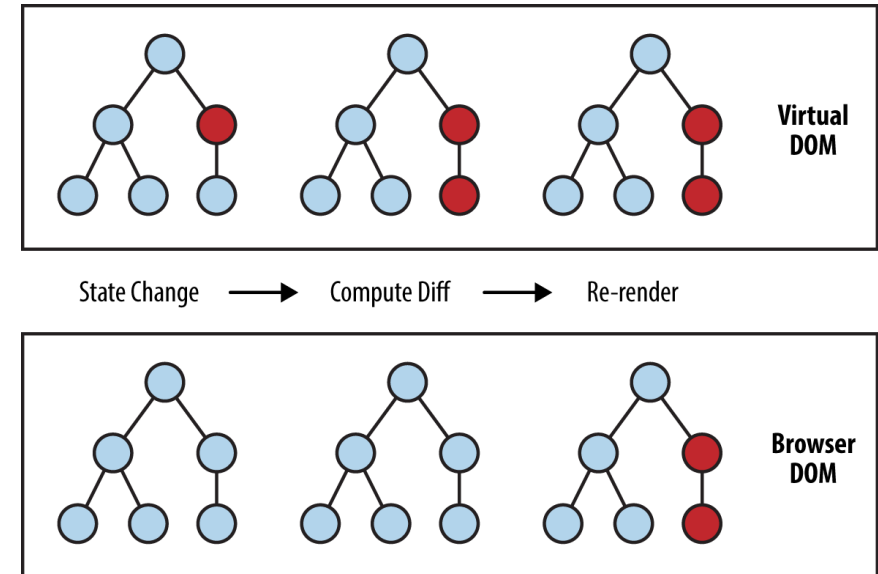
- The application is made of Components
- The entire application is re-rendered
 - Every time a state is changed
 - Every time a property is changed
- Each Component will re-build itself from scratch
 - With minor variations, or
 - Radically different
- Performance?

Re-Rendering performance

- Modifications to the DOM are expensive (re-computing layout and updating GUI)
- React implements a **Virtual DOM** layer
 - Internal in-memory data structure, optimized and very fast to update
 - Corrects some DOM anomalies and asymmetries
 - Manages its own set of “synthetic” events
 - After components re-render, React computes the difference between the “old” DOM and the new modified Virtual DOM
 - Only modifications and differences are selectively applied to the browser’s DOM, in batch

Update cycle

- Build new Virtual DOM tree
- Diff with old one
- Compute minimal set of changes
- Put them in a queue
- Batch render all changes to browser

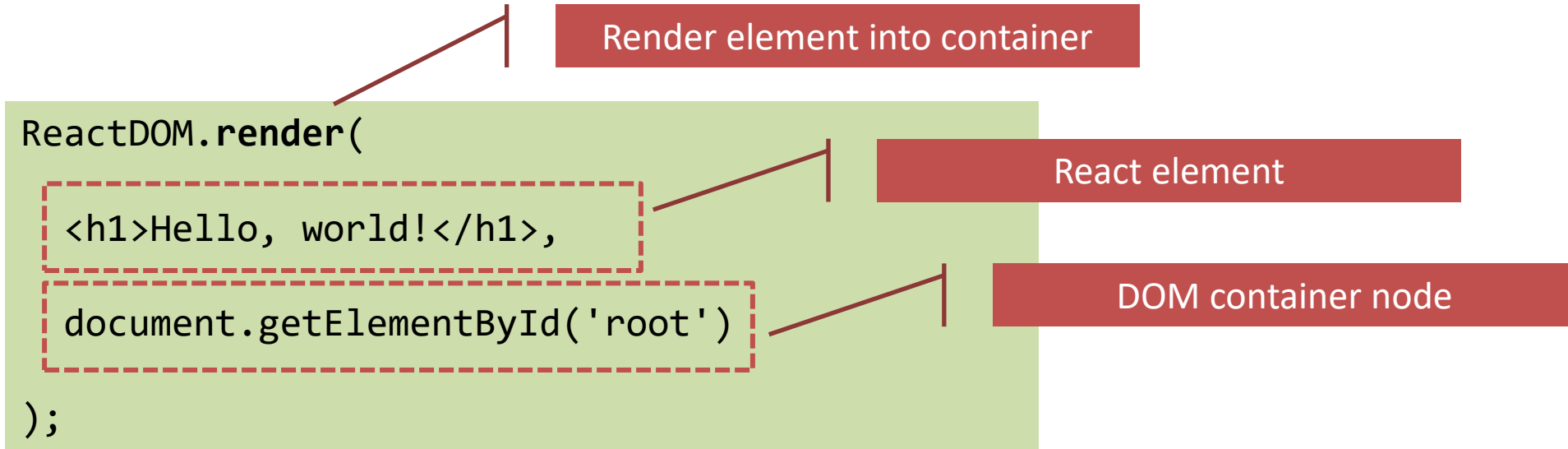


<https://www.oreilly.com/library/view/learning-react-native/9781491929049/ch02.html>

Synthetic events

- React implements its own event system
- A single native event handler at root of each component
- Normalizes events across browsers
- Decouples events from DOM

How React code looks like



JSX Syntax

```
ReactDOM.render(  
  <div id="test">  
  <h1>A title</h1>  
  <p>A paragraph</p>  
</div>,  
  document.getElementById('myapp')  
);
```

JSX Syntax

Equivalent

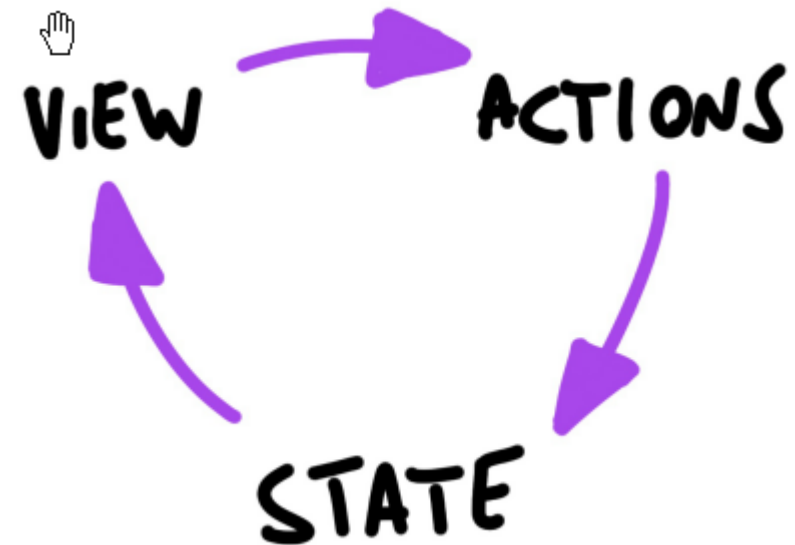
```
ReactDOM.render(  
  React.DOM.div(  
    { id: 'test' },  
    React.DOM.h1(null, 'A title'),  
    React.DOM.p(null, 'A paragraph')  
  ),  
  document.getElementById('myapp')  
);
```

JS calls to `React.createElement`

Transpiling
(Babel)

Unidirectional Data Flow

- State is passed to the view and to child components
- Actions are triggered by the view
- Actions can update the state
- The state change is passed to the view and to child component

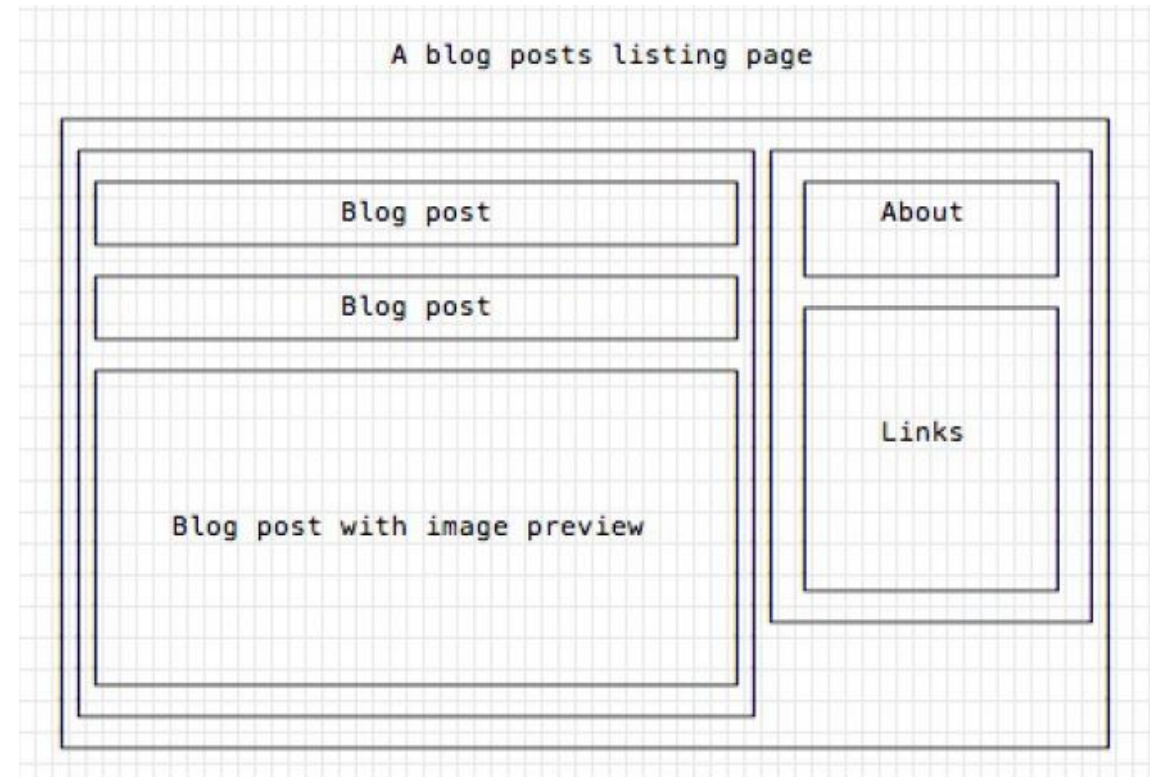


Corollary

- A **state** is always **owned by one Component**
 - Any data that's affected by this state can only affect Components below it: its children.
- Changing state on a Component will never affect its parent, or its siblings, or any other Component in the application
 - Just its children
- For this reason, state is often **moved up** in the Component tree, so that it can be **shared** between components that need to access it.

Components

- Everything on a page is a Component
 - Even simple HTML tags (React.DOM.element)
- Components may be **nested**
- ReactDOM.render builds a component and attaches it to a DOM container



Defining custom components

As a function, returning DOM elements

```
const BlogPostExcerpt = () => {  
  return (  
    <div>  
      <h1>Title</h1>  
      <p>Description</p>  
    </div>  
  )  
}
```

As a class, with a render() method

```
import React, { Component } from 'react'  
  
class BlogPostExcerpt extends Component {  
  render() {  
    return (  
      <div>  
        <h1>Title</h1>  
        <p>Description</p>  
      </div>  
    )  
  }  
}
```

Types of components

Presentational Components

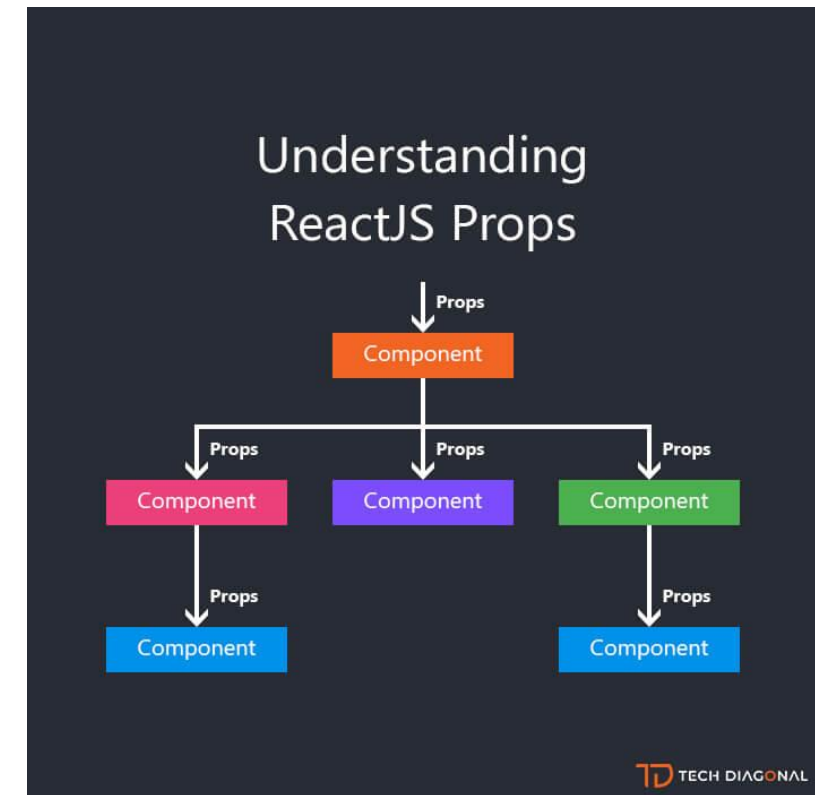
- Generate DOM nodes to be displayed
- Do not manage application state
- Might have some internal state, uniquely for presentation purposes

Container Components

- Manage the state for a group of children
- Interact with the back-end
- Create (presentational) children to display the information

State and Props

- Props are passed to a component by its parent
 - May be values to configure how the component displays
 - Top-to-bottom data flow
 - May be functions (callbacks) to access the parent's methods
 - Bottom-to-top action requests
- State is a set of variables local to the component
 - May be initialized by props
 - May be mutated only by calling `.setState()`
 - Asynchronous
 - Will initiate re-rendering of the Virtual DOM
 - May be passed to children (as props)



https://www.techdiagonal.com/reactjs_courses/beginner/understanding-reactjs-props/

Installing, configuring and running the Hello World

FIRST REACT APPLICATION

Starting small

1. Identify a “container” element (e.g., a DIV) in the HTML
 - The component will be “mounted” inside this element
 - The rest of the HTML will not be touched

```
<!DOCTYPE html>
<html>
<head>
  <title>React By Hand</title>
</head>
<body>
  <h1>Welcome to React</h1>
  <div id="react-container"></div>
</body>
</html>
```

Starting small

2. Add the `<script>` tags

- `react` : the framework
 - 25 kB (5 kB minified)
- `react-dom` : the mapping onto the browser's DOM
 - 183 kB (36 kB minified)

```
<!DOCTYPE html>
<html>
<head>
  <title>React By Hand</title>
  <script src="https://unpkg.com/react@16/umd/react.development.js" crossorigin></script>
  <script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js" crossorigin></script>
</head>
<body>
  <h1>Welcome to React</h1>
  <div id="react-container"></div>

  <script src='main.js'></script>
</body>
</html>
```

Starting small

3. Create a React component

- A class extending `React.Component`
- `constructor()`
 - Receive props
 - Initialize state
- `render()`
 - Create a new Element
 - Set `onClick` property
 - Will change the state

```
'use strict';

class WelcomeButton extends React.Component {
  constructor(props) {
    super(props);
    this.state = { english: true };
  }

  render() {
    return React.createElement('button',
      { onClick: ()=>{this.setState(
        {english: !this.state.english})} },
      this.state.english ? 'Hello' : 'Ciao' );
  }
}
```

Starting small

4. Render the component into the container

- ReactDOM.render
 - React Element
 - HTML container

```
'use strict';

class WelcomeButton extends React.Component {
  constructor(props) {
    super(props);
    this.state = { english: true };
  }



  render() {
    return React.createElement('button',
      { onClick: ()=>{this.setState(
        {english: !this.state.english})} },
      this.state.english ? 'Hello':'Ciao' );
  }
}

const container = document.querySelector('#react-
container');
ReactDOM.render(React.createElement(WelcomeButton), con
tainer);
```

What's missing?

- Can't use **JSX** directly
 - Babel required
 - Would transpile `main.js` on the fly, in the browser 😞
- Doesn't run with a web server
 - Can't use `import`
 - Problems with CORS
- Missing polyfills for browser compatibility
- Cumbersome development (edit-save-reload cycle)
- ...

Starting with all the needed infrastructure

1. `npx create-react-app react-first`
2.  ... 270 Megabytes later ... 
3. `cd react-first`
4. `npm start`
5. Visit `http://localhost:3000`

<https://create-react-app.dev/>

Folder structure

```
my-app
├── README.md
├── node_modules
├── package.json
├── .gitignore
├── public
│   ├── favicon.ico
│   ├── index.html
│   ├── logo192.png
│   ├── logo512.png
│   ├── manifest.json
│   └── robots.txt
└── src
    ├── App.css
    ├── App.js
    ├── App.test.js
    ├── index.css
    ├── index.js
    ├── logo.svg
    └── serviceWorker.js
```

- `public/index.html` is the page template
 - Published at <http://localhost:3000>
 - Automatically reloads when the application is modified
 - No need to modify, normally
- `src/index.js` is the JavaScript entry point
 - Contains the `ReactDOM.render` call to mount the App in the `#root` element
 - Do not touch, normally
- `src/App.js` is the file containing your application
 - **Develop here!**
 - Feel free to import other components
- `src/app.test.js` contains test executed by 'npm test'

Example

App.js

```
import React from 'react';
import WelcomeButton from './';

function App() {
  return (
    <div className="App">
      <h1>Welcome</h1>
      <WelcomeButton/>
    </div>
  );
}

export default App;
```

WelcomeButton.js

```
import React from 'react';

class WelcomeButton extends React.Component {
  constructor(props) {
    super(props);
    this.state = { english: true };
    this.handleClick = this.handleClick.bind(this);
  }

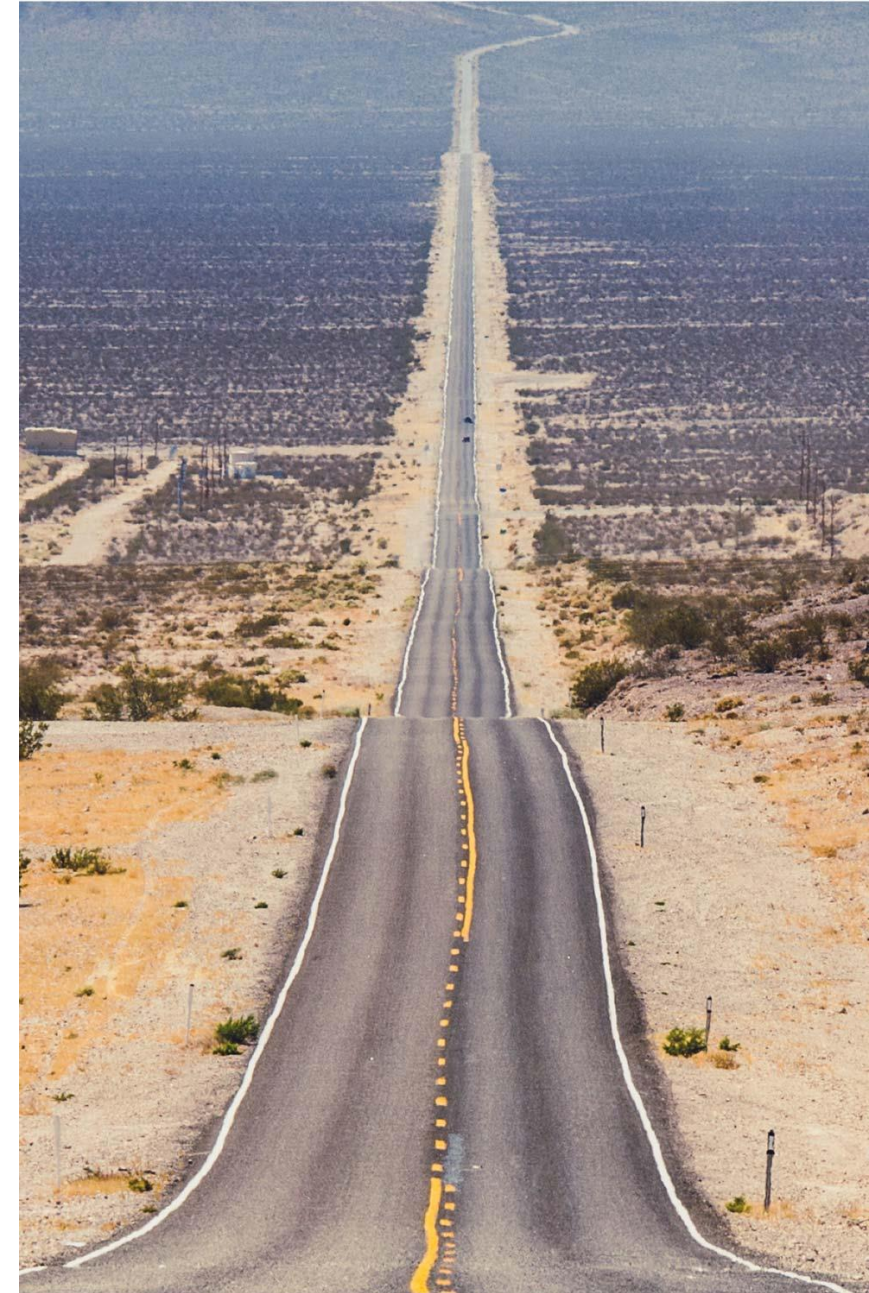
  handleClick() {
    this.setState({ english: !this.state.english });
  }

  render() {
    return <button onClick={this.handleClick}>
      {this.state.english ? 'Hello' : 'Ciao'}
    </button>
  }
}

export default WelcomeButton;
```


What's next?

- Components and props
- JSX
- State
- Events
- Forms
- Lifecycle
- Router
- Hooks
- ...



License

- These slides are distributed under a Creative Commons license “**Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0)**”
- **You are free to:**
 - **Share** — copy and redistribute the material in any medium or format
 - **Adapt** — remix, transform, and build upon the material
 - The licensor cannot revoke these freedoms as long as you follow the license terms.
- **Under the following terms:**
 - **Attribution** — You must give [appropriate credit](#), provide a link to the license, and [indicate if changes were made](#). You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
 - **NonCommercial** — You may not use the material for [commercial purposes](#).
 - **ShareAlike** — If you remix, transform, or build upon the material, you must distribute your contributions under the [same license](#) as the original.
 - **No additional restrictions** — You may not apply legal terms or [technological measures](#) that legally restrict others from doing anything the license permits.
- <https://creativecommons.org/licenses/by-nc-sa/4.0/>

