

Building User Interfaces

React 1

An Introduction

Professor Bilge Mutlu

Disclaimer

As with JS, this is not a comprehensive introduction to React, so below are links to great additional resources:

- [ReactJS.org](#)
- [W3 Schools](#)
- [Build with React](#)
- [Tania Rascia's React Overview and Walkthrough](#)

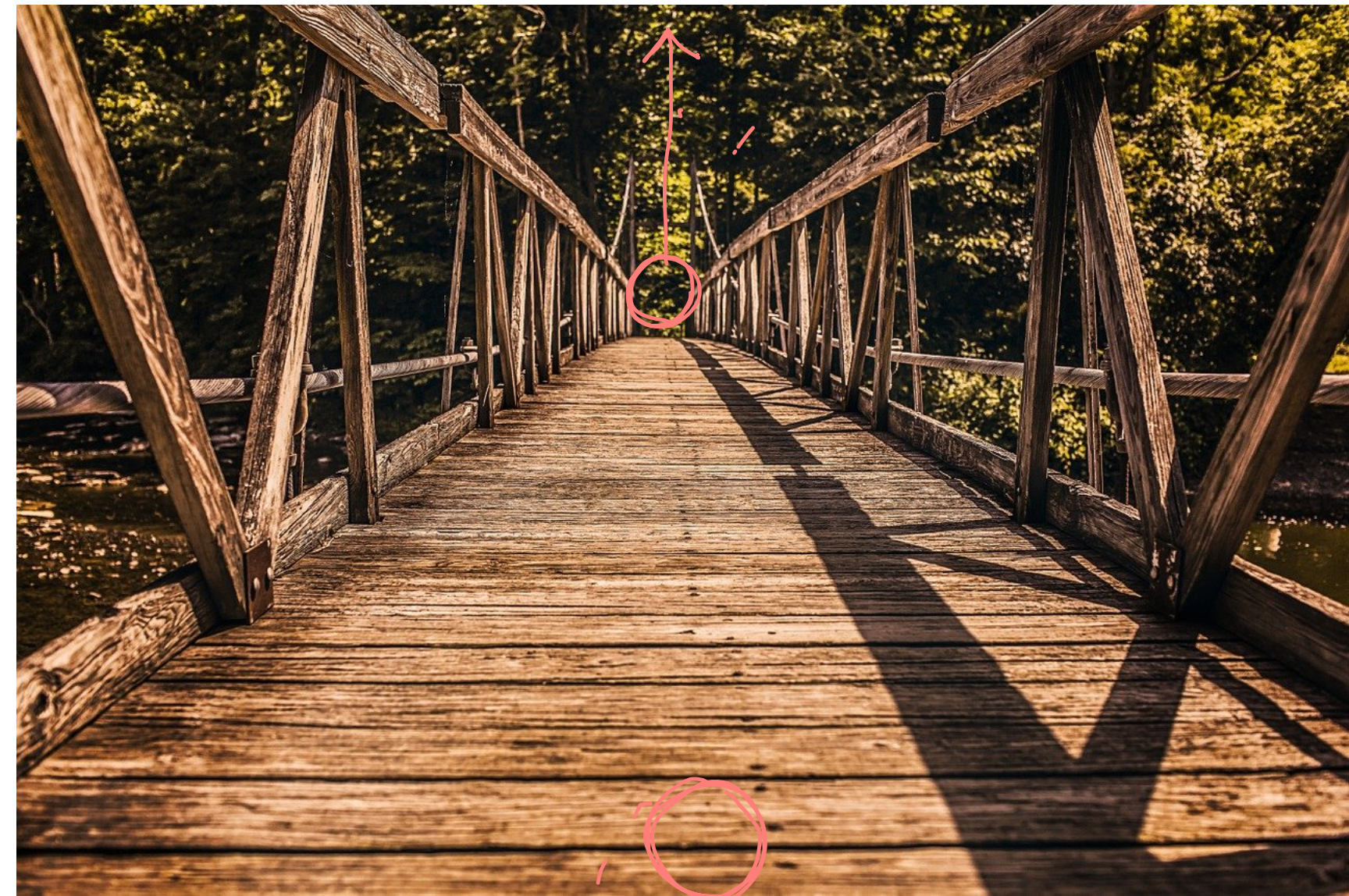
Another Disclaimer¹

If this is your first *CS* – 5 * * class, be prepared to:

- Work on more open-ended problems
- Know that there are likely more than one "correct" solution
- Find alternative ways of implementing ideas (check whether they are approved)

These are great skills to build!

¹Image source



open-ended problems with multiple
"correct" solutions in CS-5xx

constrained, narrow problems
in CS-400

What we will learn today?

- History and overview of React
- Overview of building blocks <
- Setting up a React project <

Why should we use React?

What is React?²

Definition: Also called ReactJS, React is a JS library for building user interfaces.

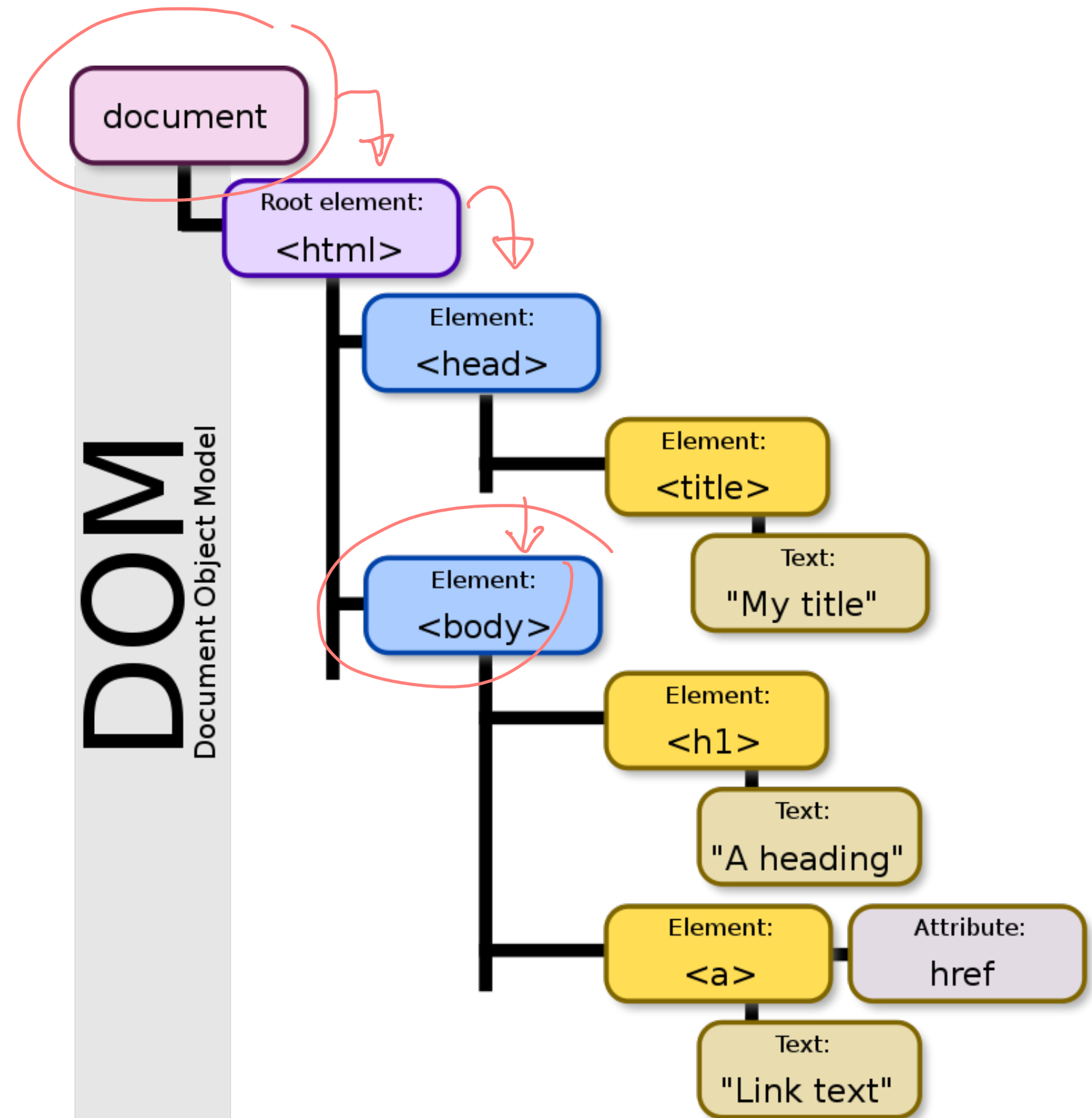
- Developed by Facebook, dating back to 2010.
- Started as an internal development tool, then open-sourced in 2013.

² More on the history of React

Refresher: Document Object Model³

Definition: Document Object Model (DOM) translates an HTML or XML document into a tree structure where each node represents an object on the page.

³[Wikipedia: DOM](#)



Refresher: DOM Programming Interface

- Objects: HTML elements, such as a paragraph of text.
- Property: Value that we can get or set, such as the id of an element.
- Method: An action we can take, such as adding or deleting an HTML element. <

For JS to interact with user-facing elements, we first need to access them...

Refresher: Accessing HTML elements

Most common way of accessing content is `getElementById()`.

```
<p id="userName"></p>  
  
<script>  
  document.getElementById("userName").innerHTML = "Cole Nelson";  
</script>
```



We can also find elements using tag name, class name, CSS selectors, and HTML object collections.

Refresher: Manipulating HTML elements

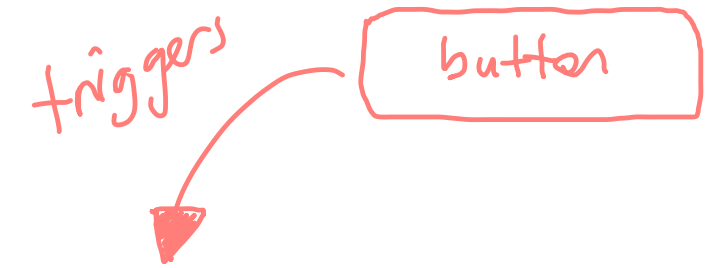
Changing content:

```
document.getElementById("userName").innerHTML = "Cole Nelson";
```

Changing attributes:

```
document.getElementById("userImage").src = "Headshot.png";  
document.getElementById("userName").style.color = "red";
```


Refresher: DOM Events



DOM provides access to HTML events, such as onclick, onload, onunload, onchange, onmouseover, onmouseout, onmousedown, onmouseup, formaction.

We can register functions to events using inline event handlers, DOM on-event handlers, and using event listeners.

What's wrong with this approach?⁴⁵

- Working with HTML DOM is slow
 - The DOM for *single-page applications (SPAs)* can be huge
 - Needle in a haystack

problem #1: finding

- Interactive applications require a large number of and frequent updates on DOM elements
 - Inefficient updating

problem #2: updating

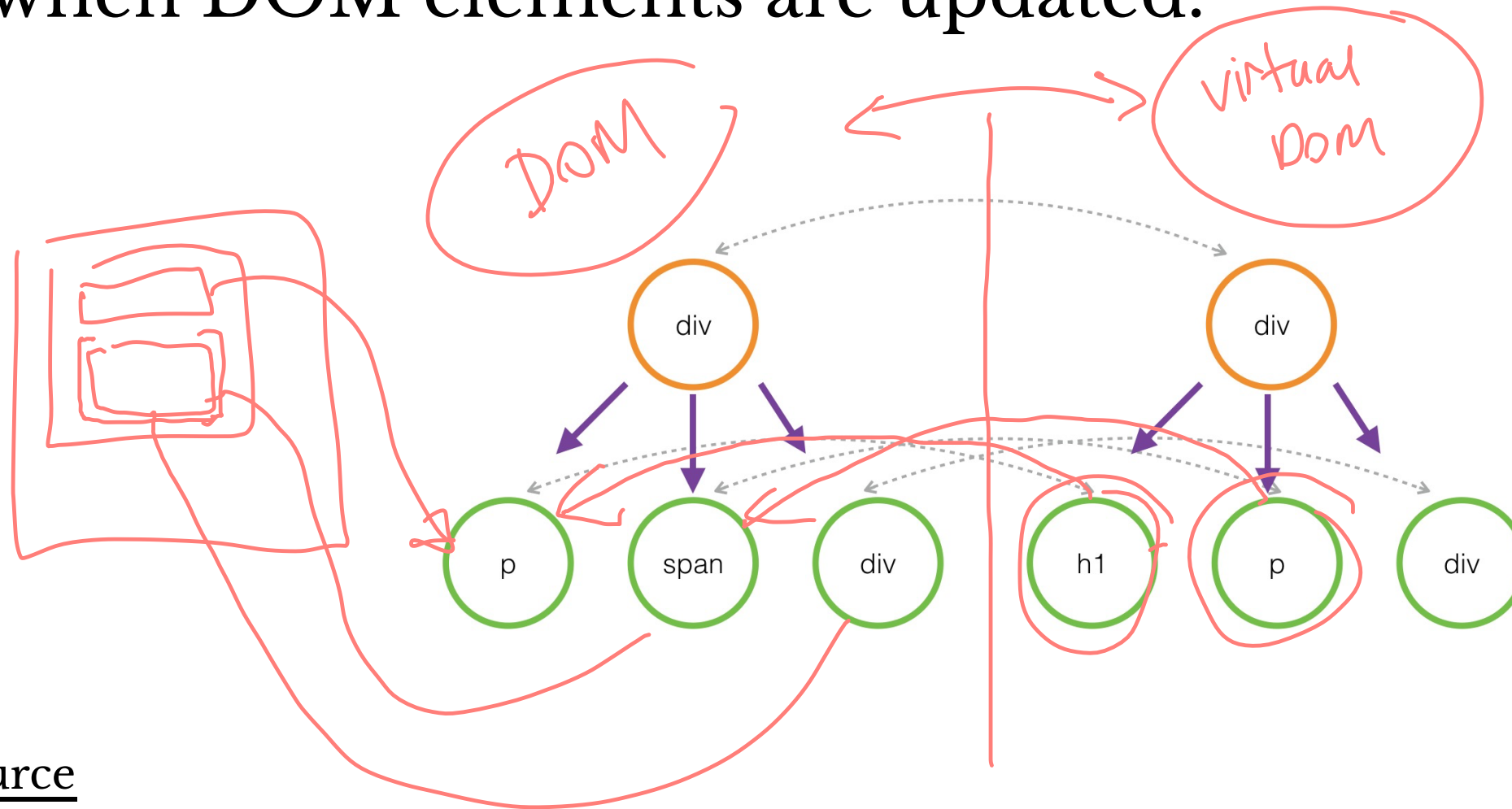


⁴ React Kung Fu

⁵ Image source

Solution: The *Virtual DOM*⁶

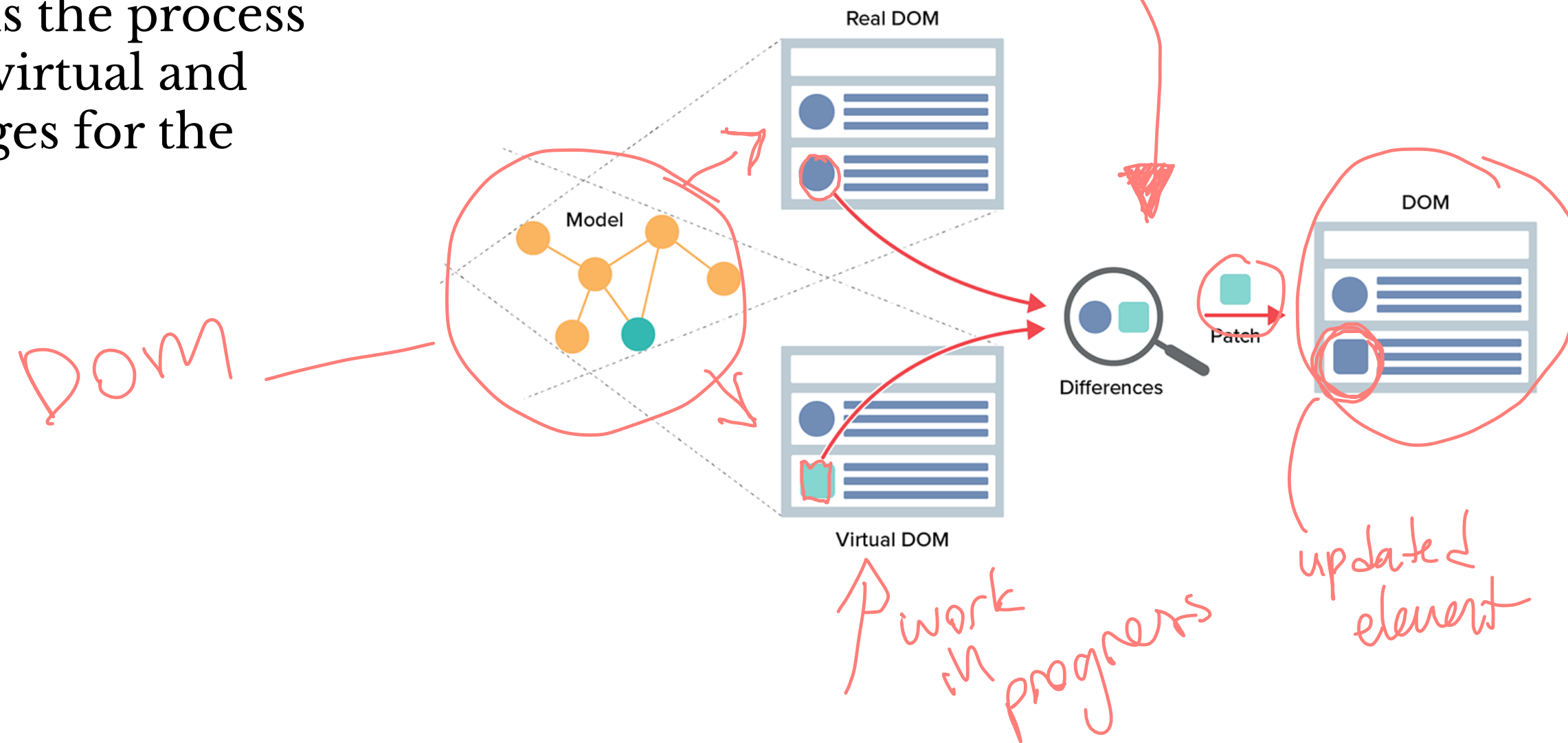
Definition: The virtual DOM is a *virtual* representation of the user-facing elements that are kept in memory and synced with the real DOM when DOM elements are updated.



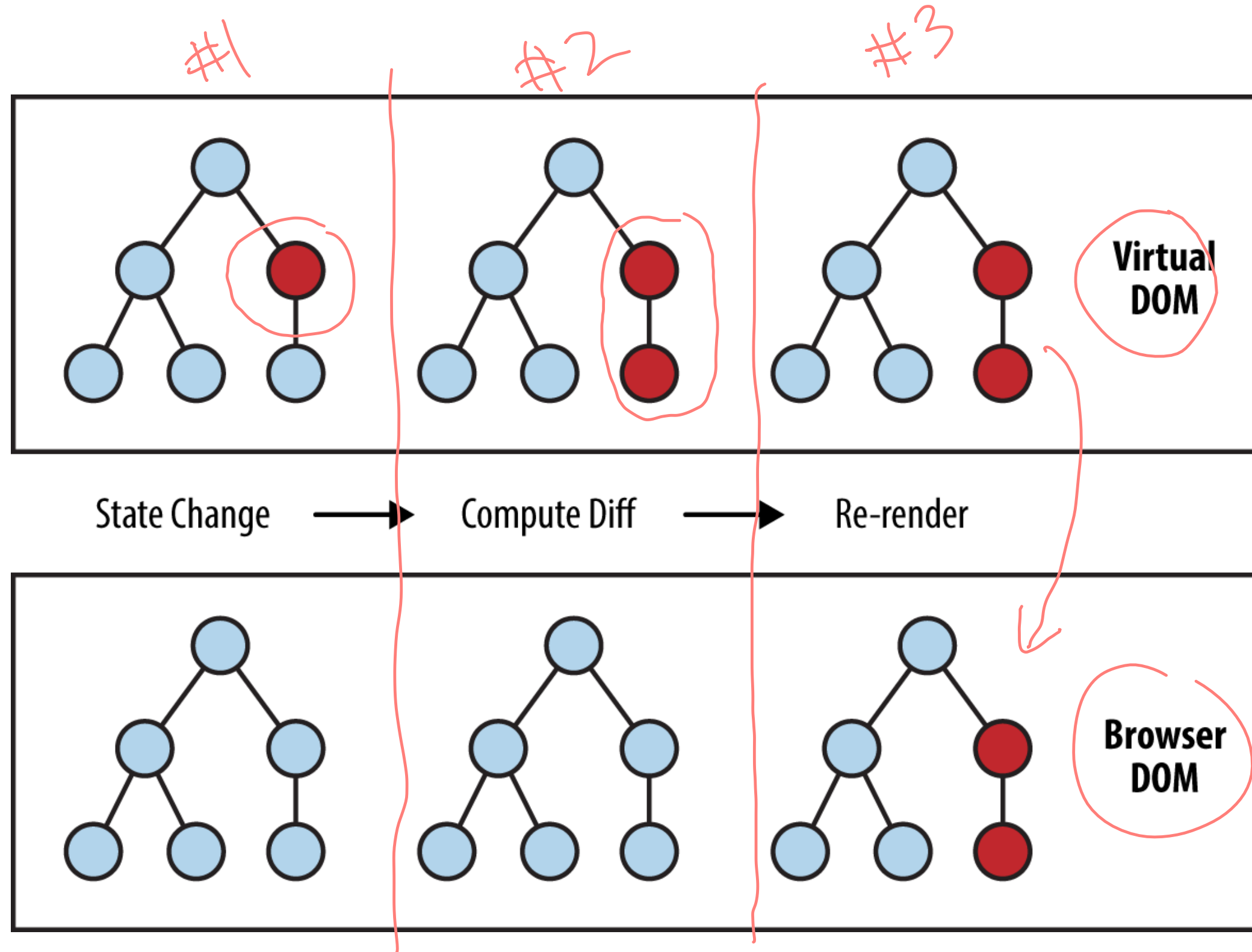
⁶Image source

More on the solution: *Reconciliation*⁷

Definition: *Reconciliation* is the process of *diffing* and syncing the virtual and real DOM to render changes for the user.



⁷Image source





Quiz 1

Complete the Canvas quiz.



What are the benefits?

- Incredibly fast, as only what is updated in the Virtual DOM is updated in the real DOM 
- Abstracts away interactions with DOM: declarative programming 

Detour: Declarative vs. imperative programming

Definition: *Imperative programming* expresses how the computation must flow.

- Programming how to get the outcome we want.

Definition: *Declarative programming* expresses the logic of a computation without describing its flow.

- Programming what we want the outcome to be.

Imperative example:⁹

```
var ages = [32, 45, 16, 67];  
function checkEligibility() {  
  for (i = 0; i < ages.length; i++) {  
    if (ages[i] < 18 || ages[i] > 65) {  
      console.log('Every one is not eligible');  
      return false; } }  
  console.log('Every one is eligible');  
  return true; }
```

Declarative example:

```
var ages = [32, 45, 16, 67];  
function checkEligibility() {  
  console.log(ages.every(age => age >= 18 && age <= 65)  
    ? 'Every one is eligible' : 'Every one is not eligible'); }  
}
```

⁹ See working example in CodePen

Detour: What are advantages of imperative/declarative programming?

Imperative

- + Easier to debug
- + Open to optimization / customization



Declarative

- + More abstraction
- + More concise + readable
- + Quicker to code
- Dense code
- + Modular
- Production complexity
- + Help them focus on outcome



Let's get back to React

- React assumes declarative programming
 - We only care about the outcome we want to see
- The ReactDOM library takes care of reconciliation and updating user-facing content under the hood

Let's look at this again visually

Credit: Maggie Appleton¹⁰

¹⁰ Image sources: next two slides

React's VIRTUAL DOM?

ONE of the reasons
REACT IS so darn

fast*

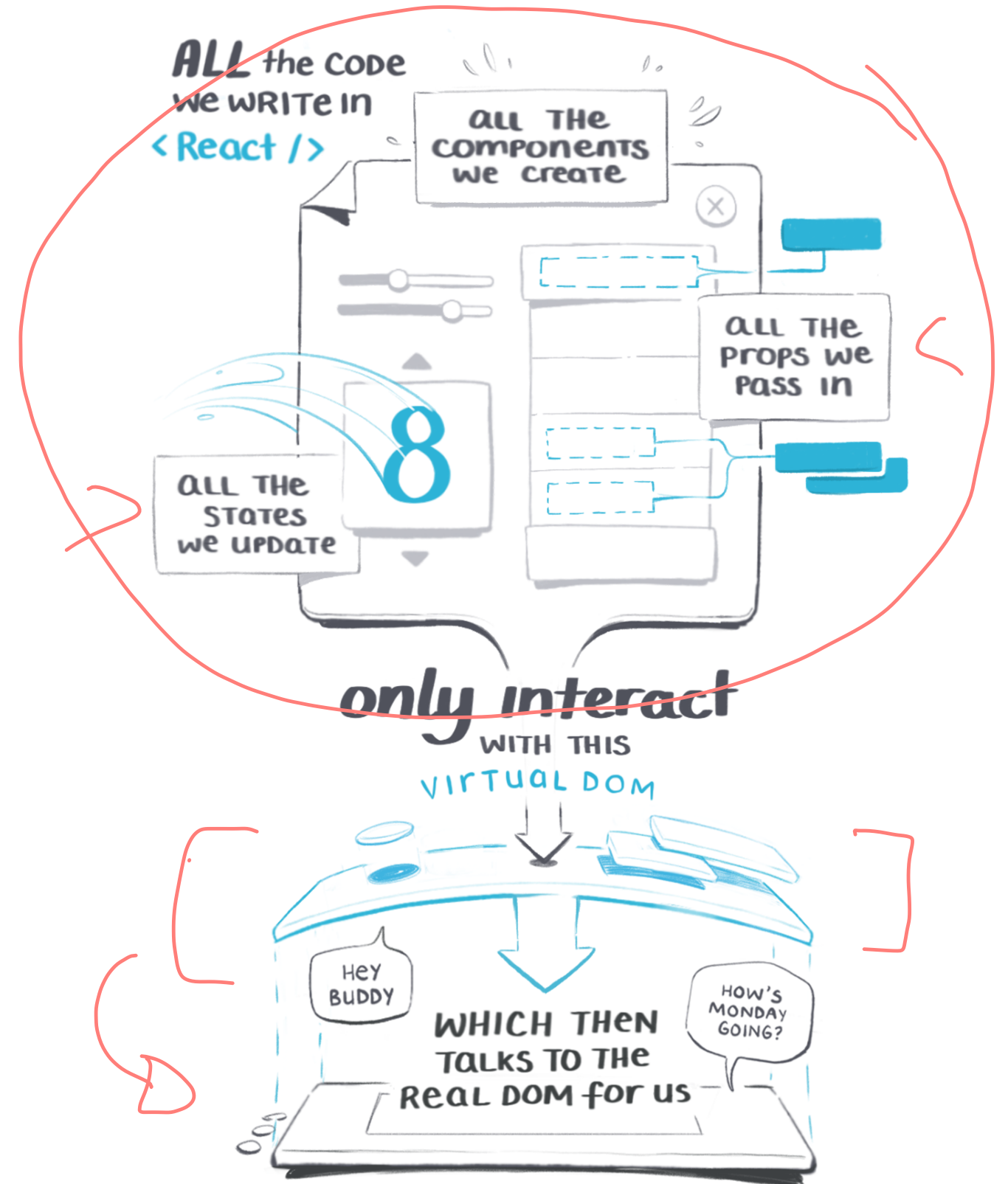
at updating
changes TO
OUR APP IS

that it
uses a
VIRTUAL
DOM

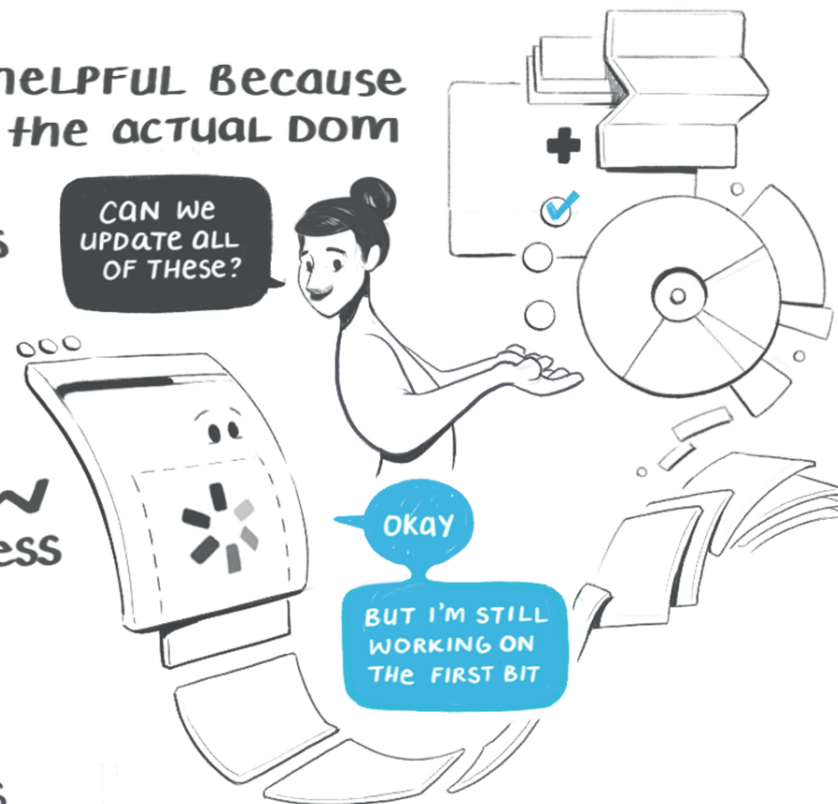
which is like
a *floating*
SECOND
LAYER
on top
of the
real DOM.

The
React
DOM

THE
"IRL"
DOM



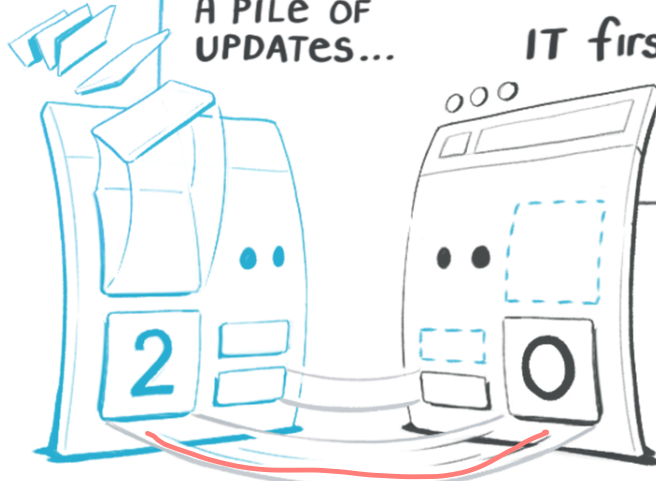
This is helpful because passing the actual DOM lots of changes to our app's state is a **SLOW** process



whereas, when we pass

THE **VIRTUAL DOM**

A PILE OF UPDATES...



IT first COMPARES IT'S OWN VERSION OF THE DOM TO THE REAL DOM and NOTES ALL THE CHANGES

AND THEN IT RUNS a **"diffing algorithm"**



WITH the Least NUMBER OF REQUESTS

OUR 1,000 UPDATES



JUST a LITTLE change..



COOL, THAT'S EASY!



(this process of syncing up your app's state with the real DOM is called **reconciliation**)

Handy dev BUZZWORD

* I know React's speed is a hotly debated point. To clarify: The *virtual DOM* doesn't speed up our initial render. It's only quicker at rendering changes to state, compared to the bog-standard, unoptimised DOM. It's also not the *absolute fastest* solution to the issue of slow DOMs.

Ok, that's why we should use React!

The name comes from *reacting* to events.

Quiz 2

Complete the Canvas quiz.

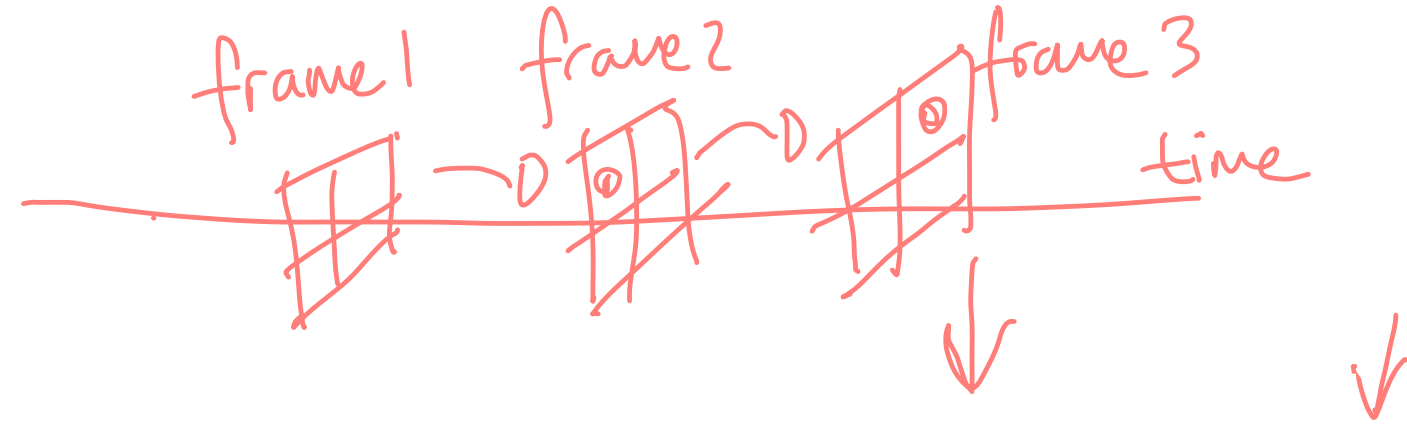


Building Blocks

Core Components

React elements and components are the two fundamental building blocks React uses to represent and change all user-facing context and events.

React Elements



Definition: A React element is a light, stateless, immutable, virtual representation of a DOM Element.

React elements are JS objects, thus browser-independent, until they are rendered. Once they are rendered, they become DOM elements.¹¹

```
JS  
var root = React.createElement('div');  
ReactDOM.render(root, document.getElementById('example'));
```

¹¹ List of ReactElements

In the browser

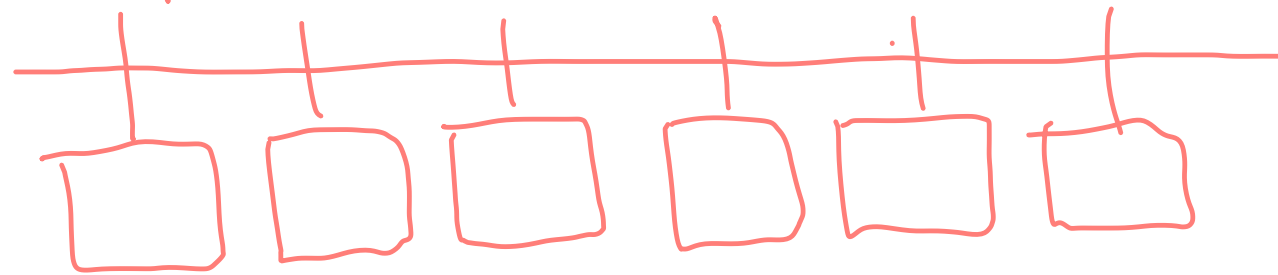
React Elements, Continued

If React elements are immutable, how do we update the page?

Think of an element as a single frame of a movie, which represents the page at a certain point in time. To update the page, we create and render a new element.

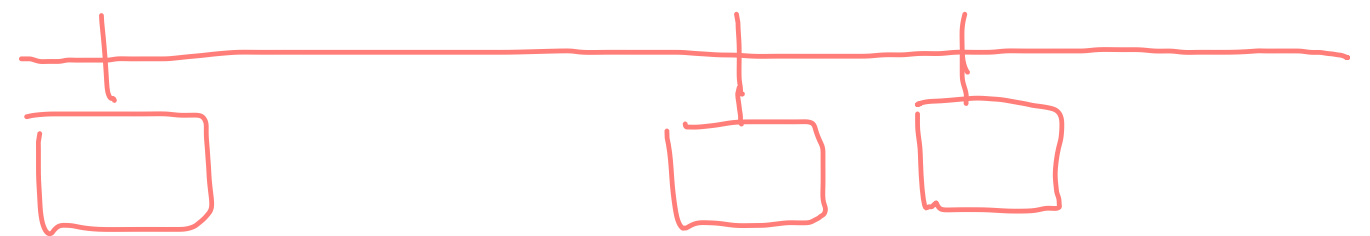
regular movie

updates at regular intervals



how React works

updates at user/data events



React.Component

Definition: A React component is a function or class that accepts an input and returns a React element.

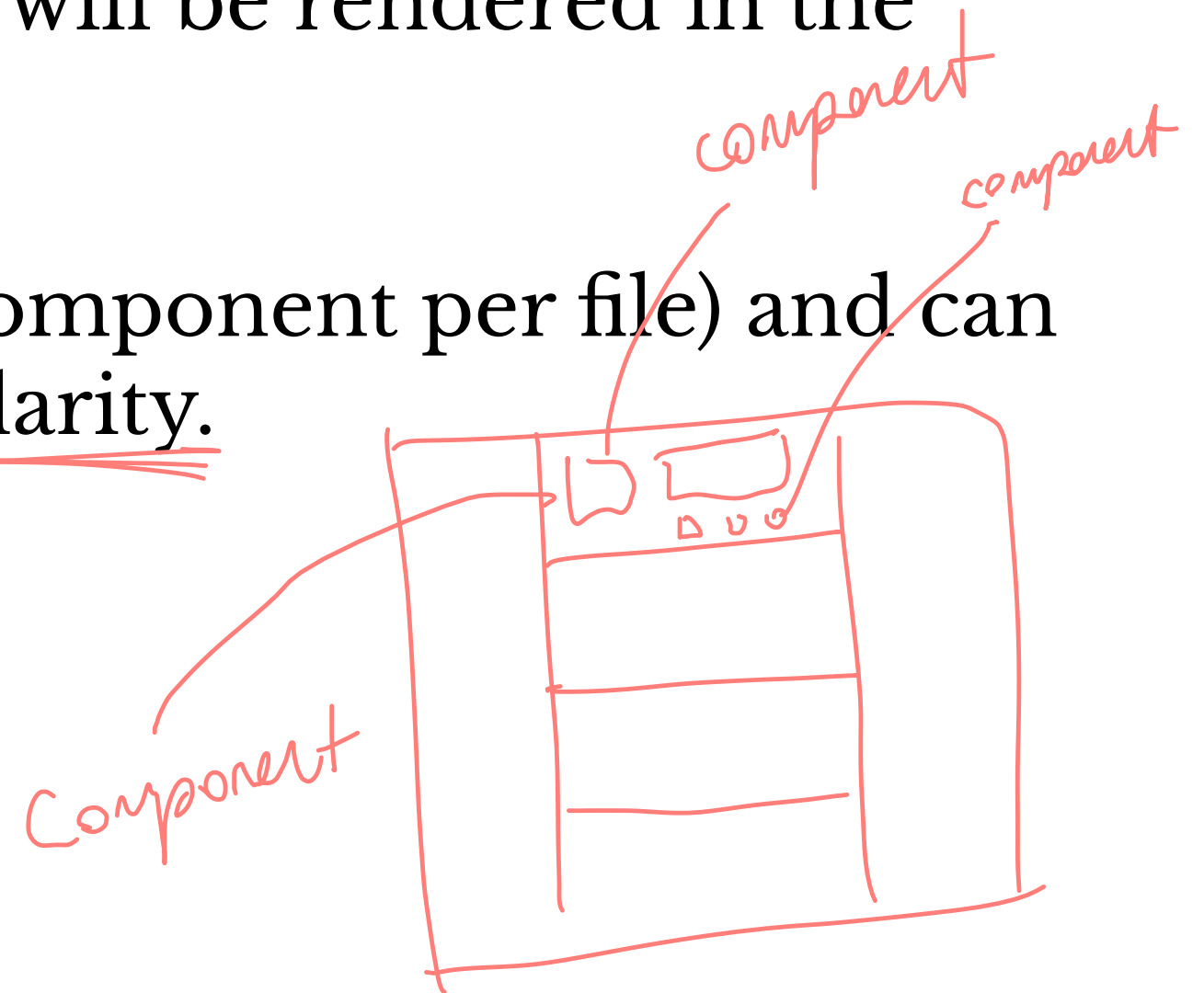
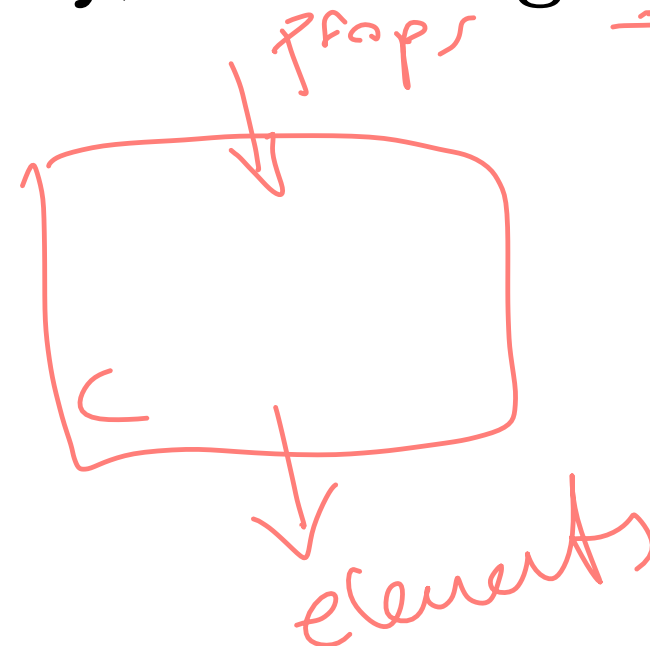
```
class Welcome extends React.Component {  
  render() {  
    return <h1>Hello, {this.props.name}</h1>;  
  }  
}
```

React element

React.Component, Continued

Components work like JS functions; they accept props and return React elements that correspond to what will be rendered in the DOM.

Each component is encapsulated (one component per file) and can operate independently, affording modularity.



render()

ReactDOM

Definition: render() returns a React element to be displayed on the page. Think of what you would like to see on the screen to determine what should go in render().

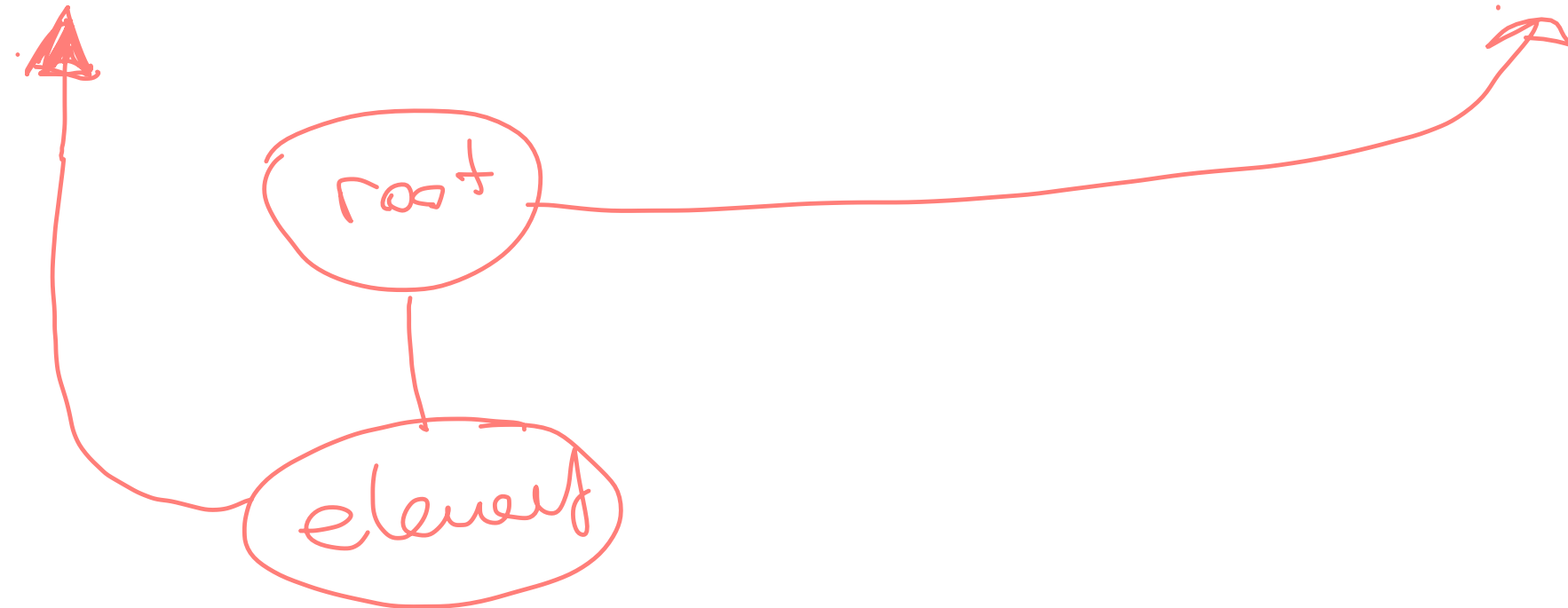
There are two render() methods in React: ReactDOM.render() and Component.render().

ReactDOM.render()¹²

ReactDOM.render(element, container) mounts the declared element as a child to the specified container in the DOM.

```
const element = <h1>Welcome to React</h1>;  
ReactDOM.render(element, document.getElementById('root'));
```

#1 create
#2 push to DOM



¹² See in CodePen

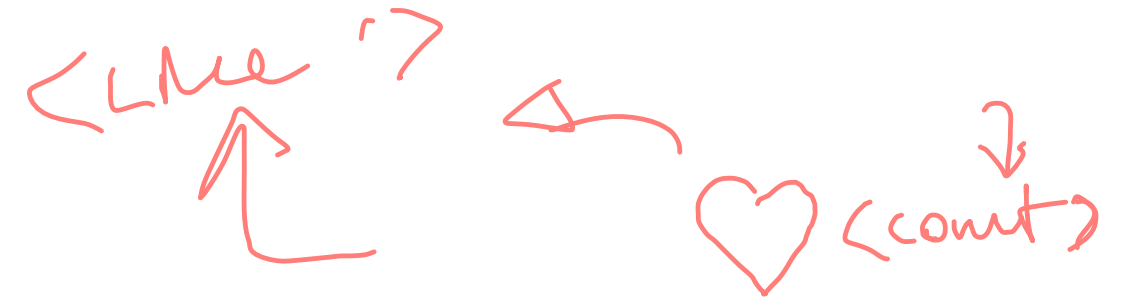
Component.render()¹³

Component.render() creates the virtual DOM representation of the contents of the React component. And then, we call ReactDOM.render() to mount the elements on the DOM.

```
class App extends React.Component {  
  render() {  
    return (  
      <div>  
        <h1 className="App-title">Welcome to React</h1>  
      </div>  
    );  
  }  
}  
ReactDOM.render(<App />, document.getElementById("root"));
```

¹³ See in CodePen

props and states



Definition: props, or properties, are the arbitrary input provided into React components that utilize them to render content.

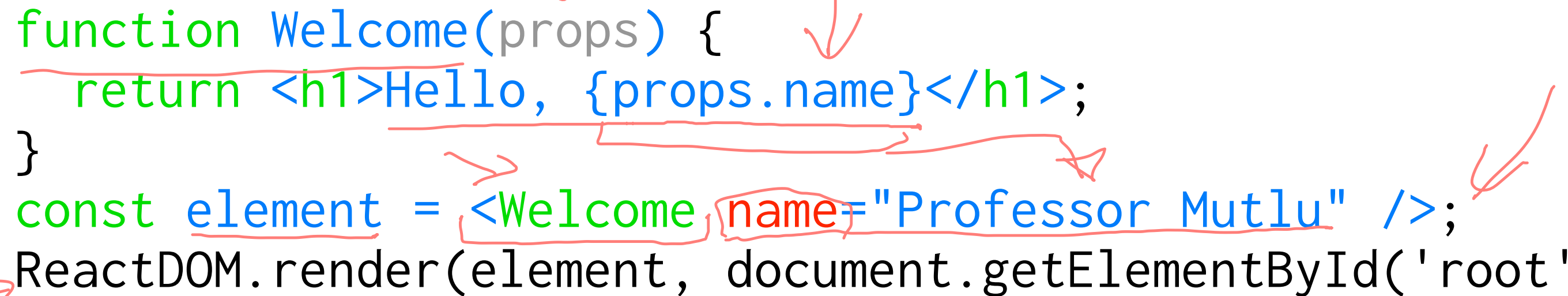
Components should never modify props; they are read-only and immutable.

Definition: state is similar to props, but they are fully private and controlled by the component.

state is what helps us keep track of changes in data.

Usage: props¹⁴

Props are passed into functions as arguments, e.g.:



```
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}  
const element = <Welcome name="Professor Mutlu" />;  
ReactDOM.render(element, document.getElementById('root'));
```

The diagram shows red arrows indicating the flow of props. An arrow points from the text 'Props are passed into functions as arguments' to the 'props' parameter in the function definition. Another arrow points from the 'props' parameter to the 'props.name' property access in the return statement. A third arrow points from the 'name' prop in the JSX element to the 'name' property access in the return statement. The 'name' prop in the JSX element is also underlined.

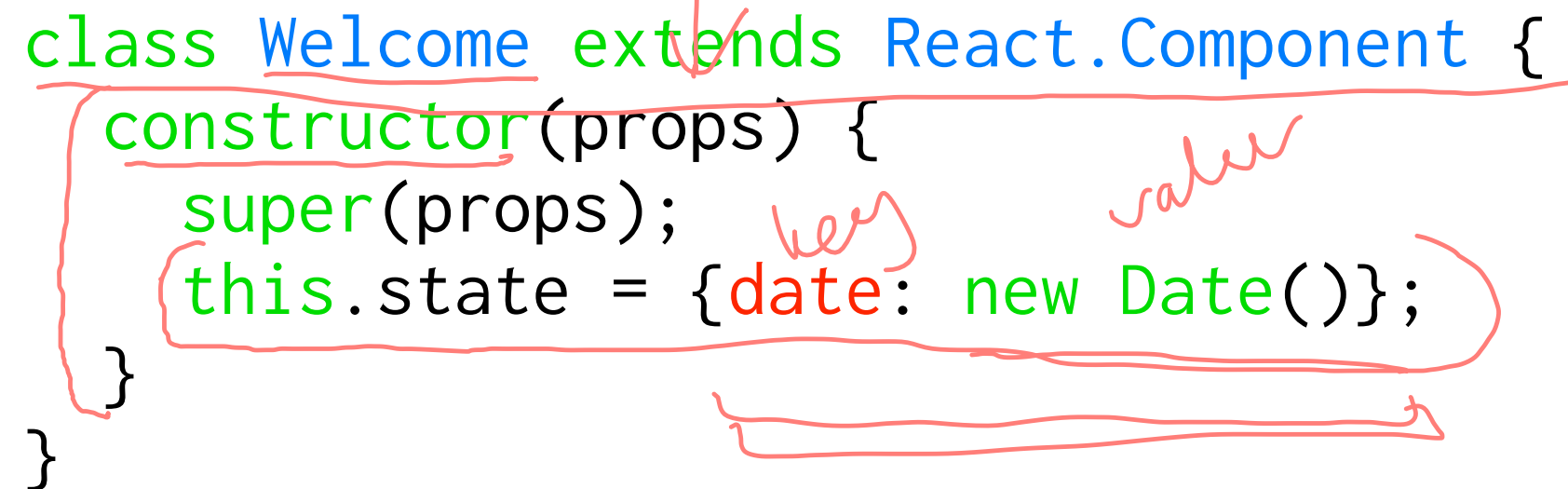
Reference using `this.props` within the component.

¹⁴ See in CodePen

Usage: state¹⁵

States are defined and manipulated within components, e.g.:

```
class Welcome extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {date: new Date()};  
  }  
}
```



The state can be referenced using `this.state` and changed using `this.setState()`. More on this later...

¹⁵ See in CodePen

JSX

Definition: A preprocessor step that adds XML syntax to JavaScript. JSX declarations produce React elements.

`<div className="red">Text</div>;`

... is compiled into ...

`React.createElement("div", { className: "red" }, "Text");`

Babel¹⁶ is the preprocessor that compiles JSX into JS.

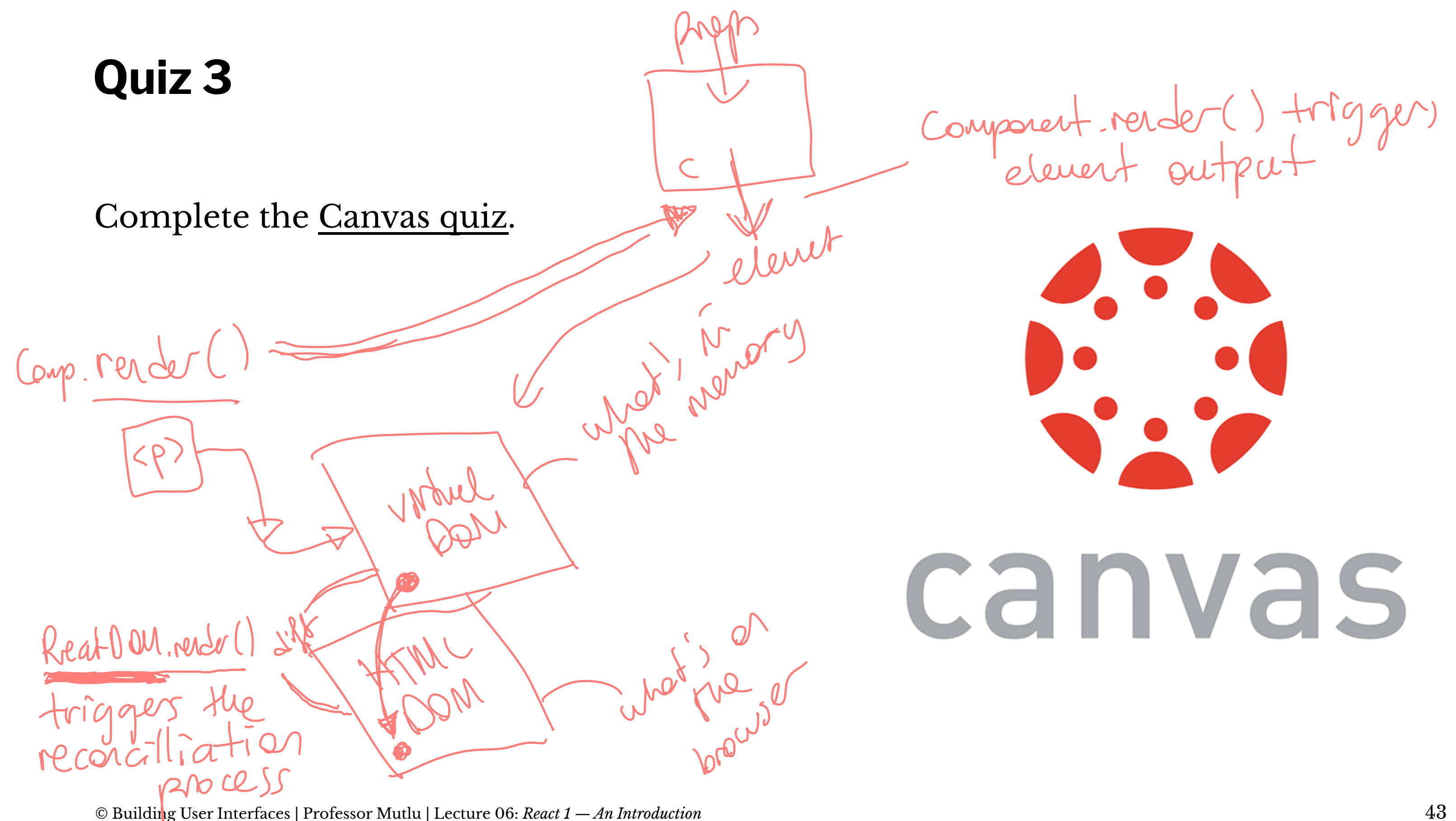
¹⁶ See example in Babel REPL

JSX, Continued

- JSX is not required, but makes React programming extremely effective
- Any JS expression can be included in JSX

Quiz 3

Complete the Canvas quiz.



Detour: Naming Conventions¹⁷

Definition: camelCase involves writing phrases such that each word begins capitalized with no spaces/punctuation.

Definition: hyphen-case (aka ~~kebab-case~~) involves writing phrases in lower case and using a hyphen as a separator.



¹⁷Wikipedia

Detour: Naming Conventions, Continued

Definition: PascalCase capitalizes all words with no spaces/punctuation.

- ReactDOM and JSX use camelCase ↙
- React components use PascalCase ↙
 <welcome />

Setting up a React project

Setting up a React project

In the development environment

What you will need: *terminal, coding environment, Node.js*

sudo

```
npm install -g create-react-app
```

```
npx create-react-app <your-app-name>
```

```
cd <your-app-name>
```

```
npm start
```

Setting up a React project

In a sandbox

Simple React application:¹⁸

1. set sandbox settings to Babel preprocessor
2. import react and react-dom CDNs

React project:

— Create a project using a template / upload your project

¹⁸ See example in CodePen

What did we learn today?

- History and overview of React
- Overview of building blocks
- Setting up a React project