PERFICIENT

REACT STUDY GROUP– DANIEL TORRES ORJUELA

React behind the scenes

07/2023

PERFICIENT

Definición, ventajas y desventajas

Componentes, JSX y transpiladores

Estado y manejadores de eventos

React behind the scenes

Mejores prácticas

PERFICIENT

Chill and have fun

CodeSandbox

QUIZIZZ

Ask all the questions you want, the idea is to debate… and learn

Agenda

1. The classic web development approach
2. The React approach
3. Three step rendering process
4. State as a snapshot
5. The React state changes queue

The web development classic approach

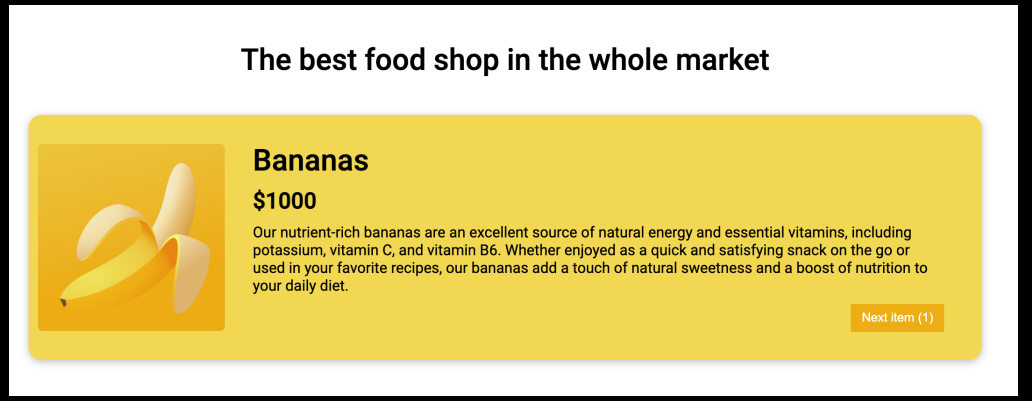
The web development classic approach

```html
<div class="root">
  <h1>The best food shop in the whole market</h1>
  <div class="card">
    <img
      src="https://img.icons8.com/?size=512&id=p7YHa8almDEQ&format=png"
      alt="banana"
    />
    <div class="card-content">
      <h1>Bananas</h1>
      <h2>$1000</h2>
      <p>
        Our nutrient-rich bananas are an excellent source of natural energy
        and essential vitamins, including potassium, vitamin C, and vitamin
        B6. Whether enjoyed as a quick and satisfying snack on the go or
        used in your favorite recipes, our bananas add a touch of natural
        sweetness and a boost of nutrition to your daily diet.
      </p>
      <button id="button" class="card-button">Next item (1)</button>
    </div>
  </div>
</div>
```

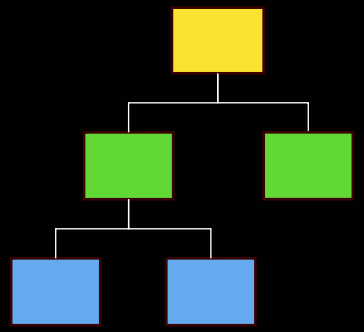
The web development classic approach

JS

```css
@import url("https://fonts.googleapis.com/css2?family=Roboto:ital,wght@0,300;0,400;0,5

html,
body {
  height: 100%;
  font-family: "Roboto", sans-serif;
  font-weight: 400;
}

h1 {
  font-weight: 500;
}

* {
  margin: 0;
}

.root {
  display: flex;
  min-height: 100%;
  flex-direction: column;
  align-items: center;
  justify-content: center;
}

.color-container {
  height: 100vh;
  width: 100%;
  background-color: antiquewhite;
  padding: 30px;
}
```



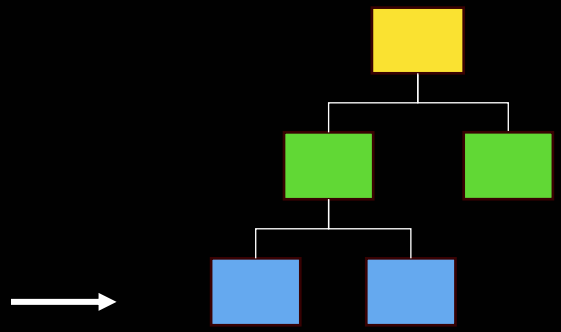
The classic approach| DOM API



Document
Object
Model

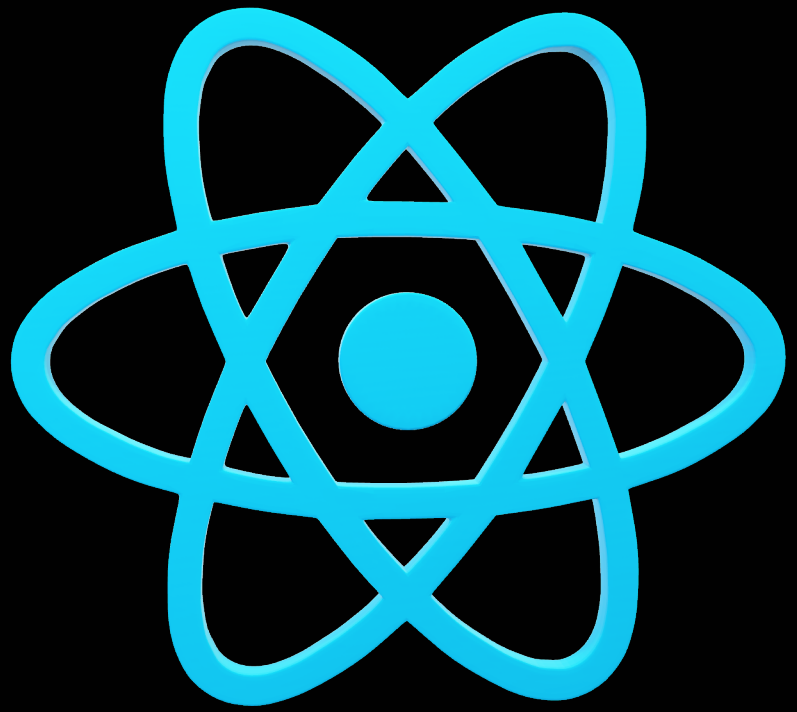
The classic approach | DOM API

```
let item = 0;
const button = document.getElementById("button");
button.addEventListener("click", () => {
  item += 1;
  button.innerHTML = `Next item (${item + 1})`;
  console.log(`Now you're in the item ${item}`);
});
```



Document
Object
Model

The React Approach

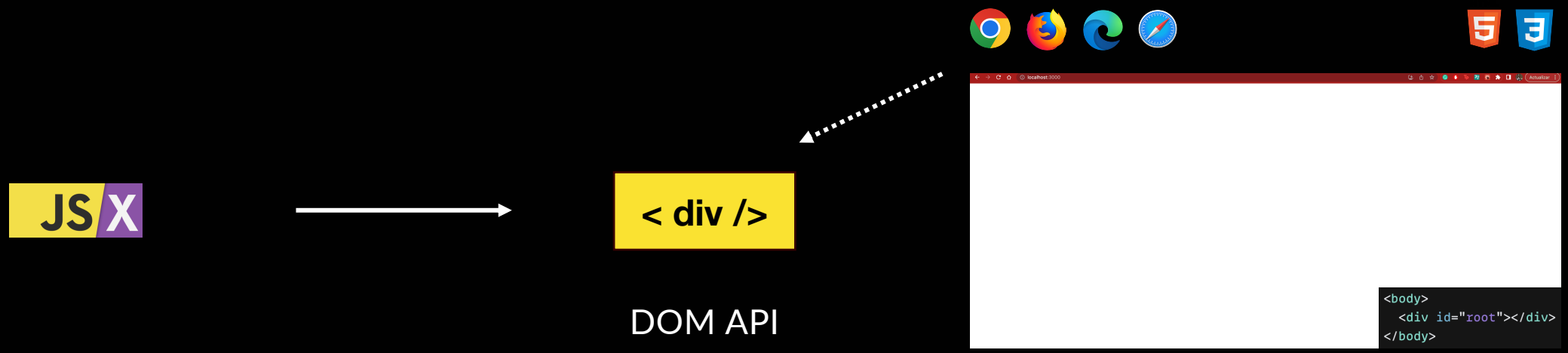


The React Approach

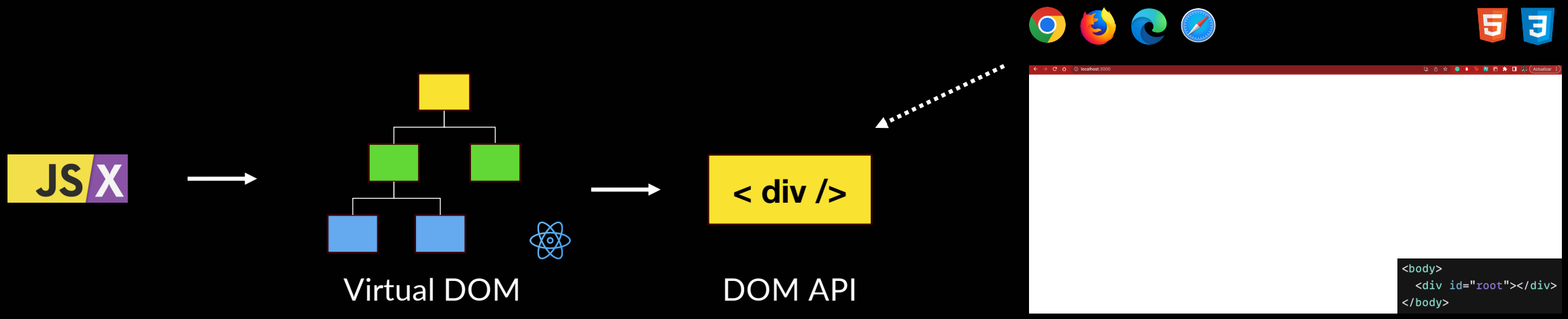
JSX

```
<body>
  <div id="root"></div>
</body>
```

The React Approach



The React Approach



Rendering process

Executed due to the init of the app or a change in any component's state

React rendering process

	Triggering	Rendering	Commiting
First render	**Causes** The app's initialization **What happens?** A root element is created in the virtual DOM	**What happens?** The root component is called The necessary DOM nodes are created in the virtual DOM and state is associated	**What happens?** The appendChild method is called for every new node so elements are added to the DOM
Re-renders	**Causes** The change of a component's state setState, context and props cause those changes	**What happens?** The component which state changed is called (and its children) DOM changes are calculated	**What happens?** The calculated changes are applied to the DOM

React rendering process | First render

Triggering

Causes

The app's initialization

What happens?

A root element is created in the virtual DOM

Rendering

What happens?

The root component is called

The necessary DOM nodes are created in the virtual DOM and state is associated

Commiting

What happens?

The appendChild method is called for every new node so elements are added to the DOM

First render | Triggering

A root element is created in the virtual DOM


```
const root = ReactDOM.createRoot(document.getElementById("root"));
```

First render | rendering

As the root component is renderer, the whole app is executed.

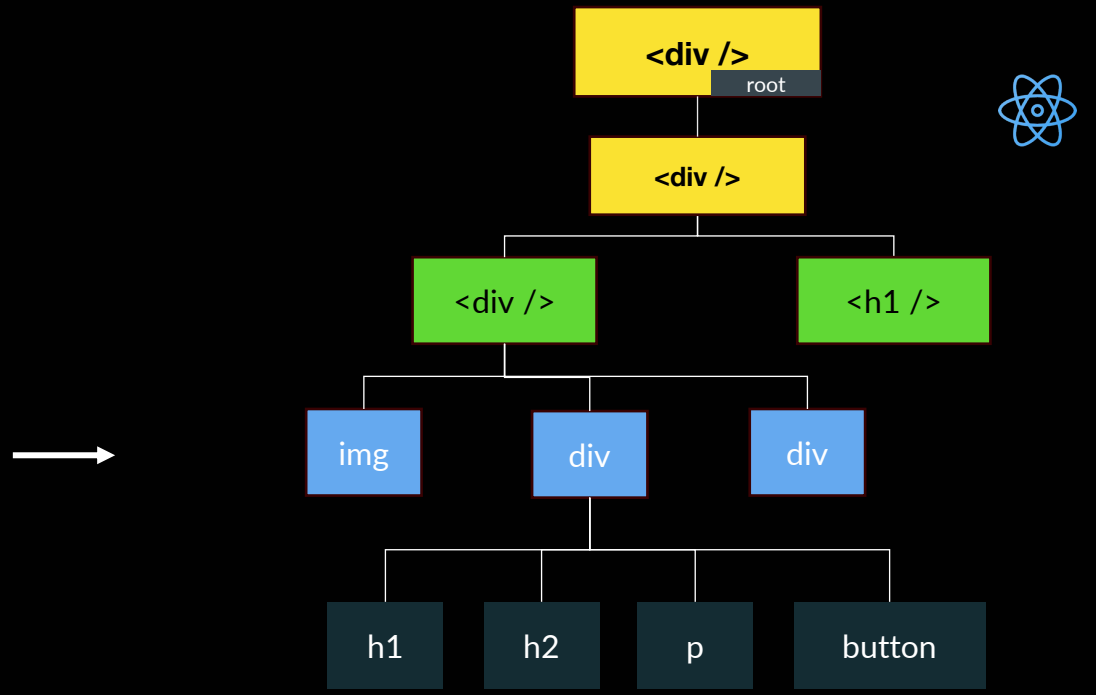
React calculates the virtual DOM based on the returned JSX and state

App.jsx

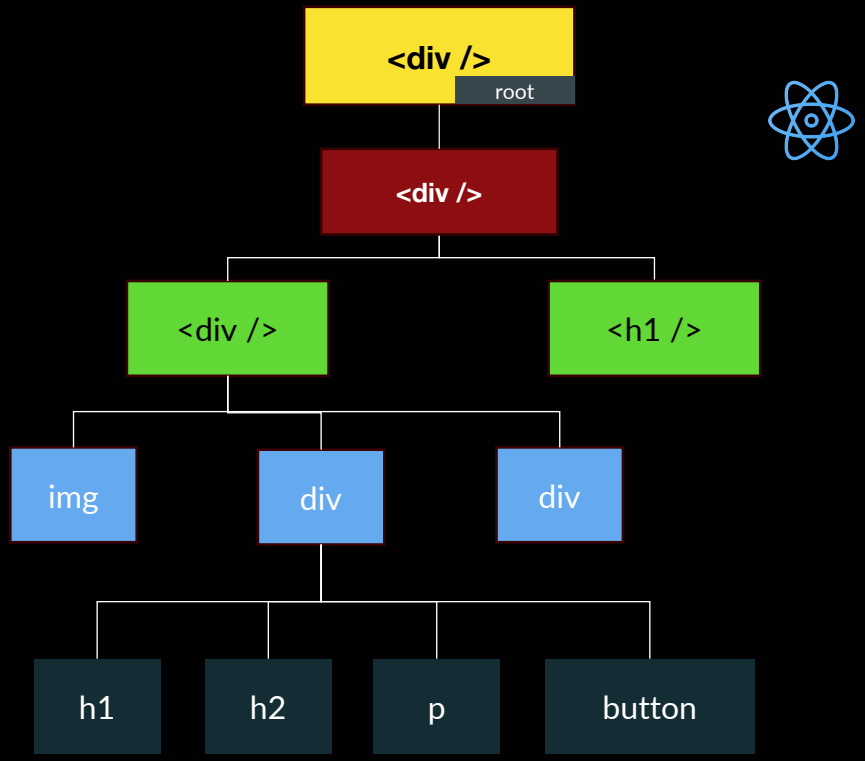
```
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

App.jsx

```
return (
  <div className={classes.root}>
    <h1>{componentsTitle}</h1>
    <Card {...product} onClick={handleClick} index={index} />
  </div>
);
```



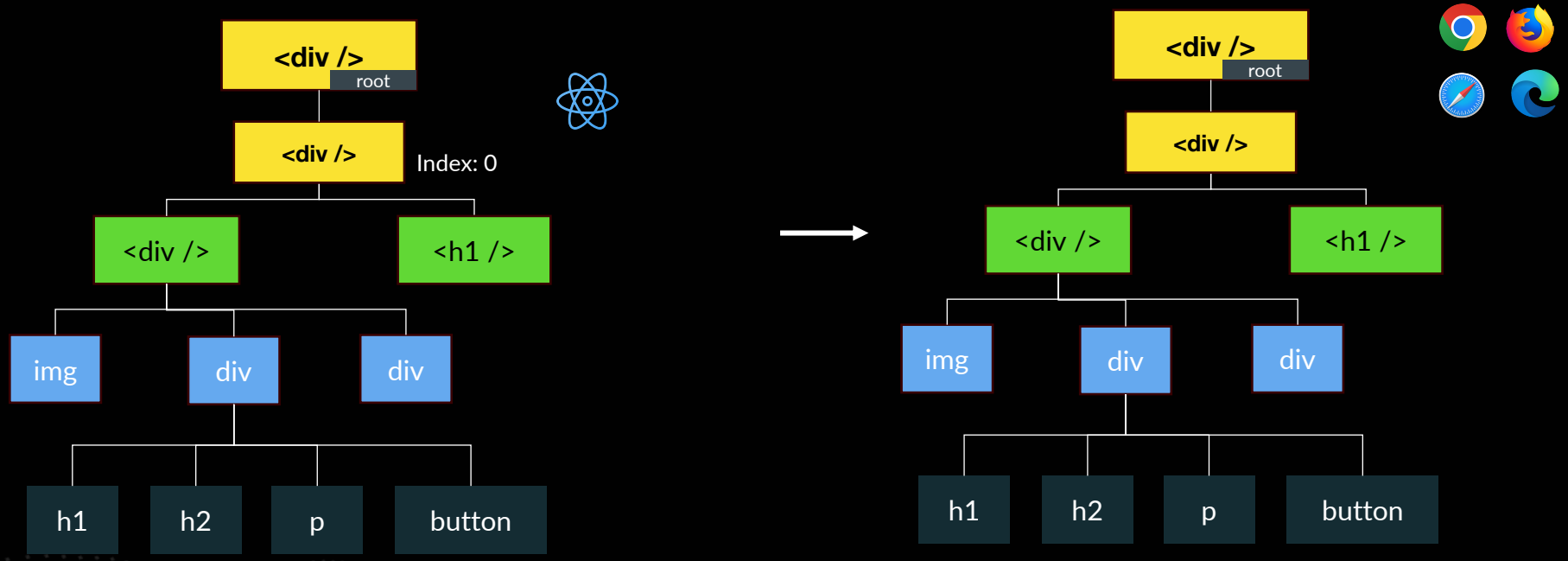
First render | rendering



Which are the state variables of the red div and what value do they have when the app is initialized?

First render | Commiting

The DOM API is changed based on the state of the Virtual DOM for the initial render



React rendering process | Re-renders

Triggering

Causes

The change of a component's state

setState, context and props cause those changes

Rendering

What happens?

The component which state changed is called (and its children)

DOM changes are calculated

Commiting

What happens?

The calculated changes are applied to the DOM

Re-renders | Triggering

A component's state changed, this process triggers a new render with the new state

Handle click is executed, so the index state variable changes

App.jsx

```
const handleClick = () => {
  setIndex(index + 1);
  console.log(`Now you're in the item ${index}`);
};
```

The component is executed again with it's new state

App.jsx

```
return (
  <div className={classes.root}>
    <h1>{componentsTitle}</h1>
    <Card {...product} onClick={handleClick} index={index} />
  </div>
);
```

Re-renders | Triggering

A component's state changed, this process triggers a new render with the new state

The component is executed again with it's new state

App.jsx

```
return (
  <div className={classes.root}>
    <h1>{componentsTitle}</h1>
    <Card {...product} onClick={handleClick} index={index} />
  </div>
);
```

As the component is executed, its children are executed too (their props have changed)

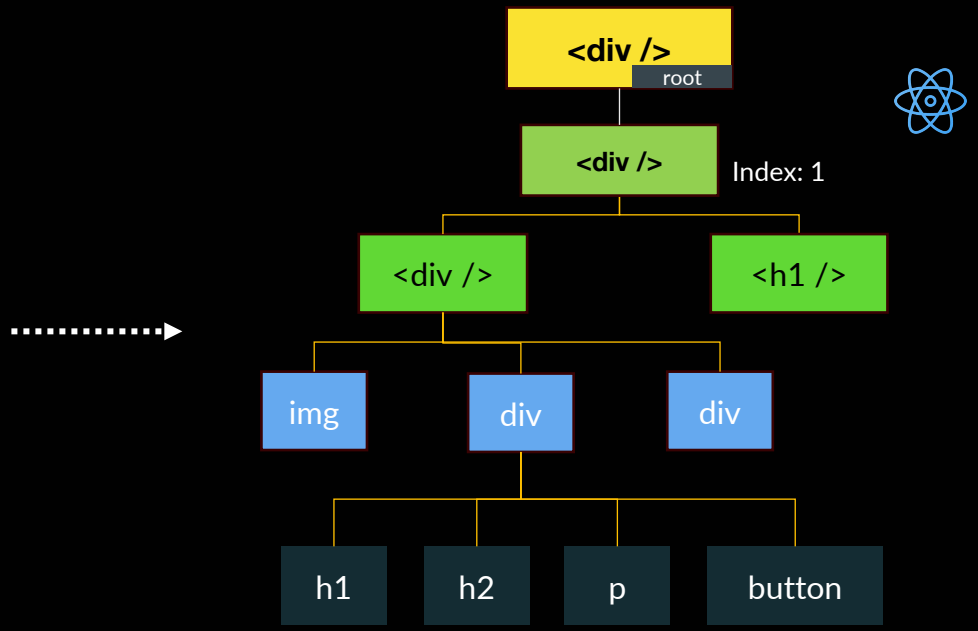
Card.jsx

```
const Card = ({
  index,
  image,
  title,
  price,
  description,
  onClick,
  bgImage,
  bgContent
}) => {
```

Re-renders | rendering

Each render is a snapshot.

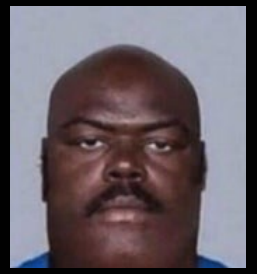
```
return (
  <div className={classes.card} style={{ backgroundColor: bgContent }}>
    {price > 3000 && <Discount />}
    <img src={image} alt={title} style={{ backgroundColor: bgImage }} />
    <div className={classes["card-content"]}>
      <h1>{title}</h1>
      <h2>{price}</h2>
      <p>{description}</p>

      <button
        id="button"
        className={classes["card-button"]}
        onClick={onClick}
        style={{ backgroundColor: bgImage }}
      >
        Next item ({index + 1})
      </button>
    </div>
  </div>
);
```

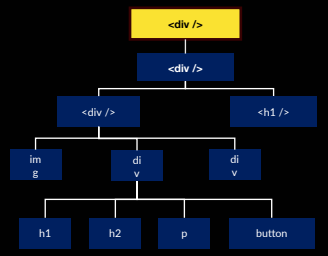


Re-renders | rendering

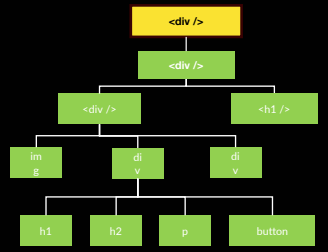
Different states should return different results

Index: 1





Index: 2



Re-renders | rendering

React is based on functional components. Those components must be pure.

It’s like ordering food in a restaurant!

Index: 1

Index: 3

Index: 245

Re-renders | rendering

```
function App() {
  const [index, setIndex] = useState(0);
  registerSession();

  const handleClick = () => {
    setIndex(index + 1);
    console.log(`Now you're in the item ${index}`);
  };

  let product = objects[index];

  return (
    <div className={classes.root}>
      <h1>The best food shop in the whole market</h1>
      <Card {...product} onClick={handleClick} index={index} />
    </div>
  );
}
```

We want to execute the registerSession() method once per user's session. There's an approach for that... do you think it'll work?

Re-renders | rendering

```
function onNextClick() {
  setIndex(index + 1);
  setIndex(index + 2);
  setIndex(index - 1);
}
```

```
function onNextClick() {
  setIndex("two");
}
```

```
function onNextClick() {
  setIndex(index + 1);
  setIndex(index + 4);
  setIndex(index + 2);
}
```

Now the market's owner wants us to add a "Next item" button outside the card that shows only the even index items. Which function accomplish that?

React makes a queue of the state changes it'll do

Reacts first takes all the changes and add them to the queue, once there is no change left, it executes them all according to the "given instructions"

```
function onNextClick() {
  setIndex(index + 1);
  setIndex(index + 4);
  setIndex(index + 2);
}
```

Assigning a direct value to the setIndex is like saying "hey, it doesn't matter what value it has, reset it to this one"

React makes a queue of the state changes it'll do

If we want to make operations in state variables on a single render, we must use a function that takes the actual value as parameter and returns the expected result

```
function onNextClick() {
  setIndex(index => index + 1);
  setIndex(index => index + 1);
}
```

Here you're saying "Hey, take the actual value and do an operation with that. I want to accumulate those operations

Re-renders | rendering

```
function onNextClick() {
  setIndex((index) => index + 2);
  setIndex(0);
  setIndex((index) => index + 3);
  setIndex(index - 1);
}
```

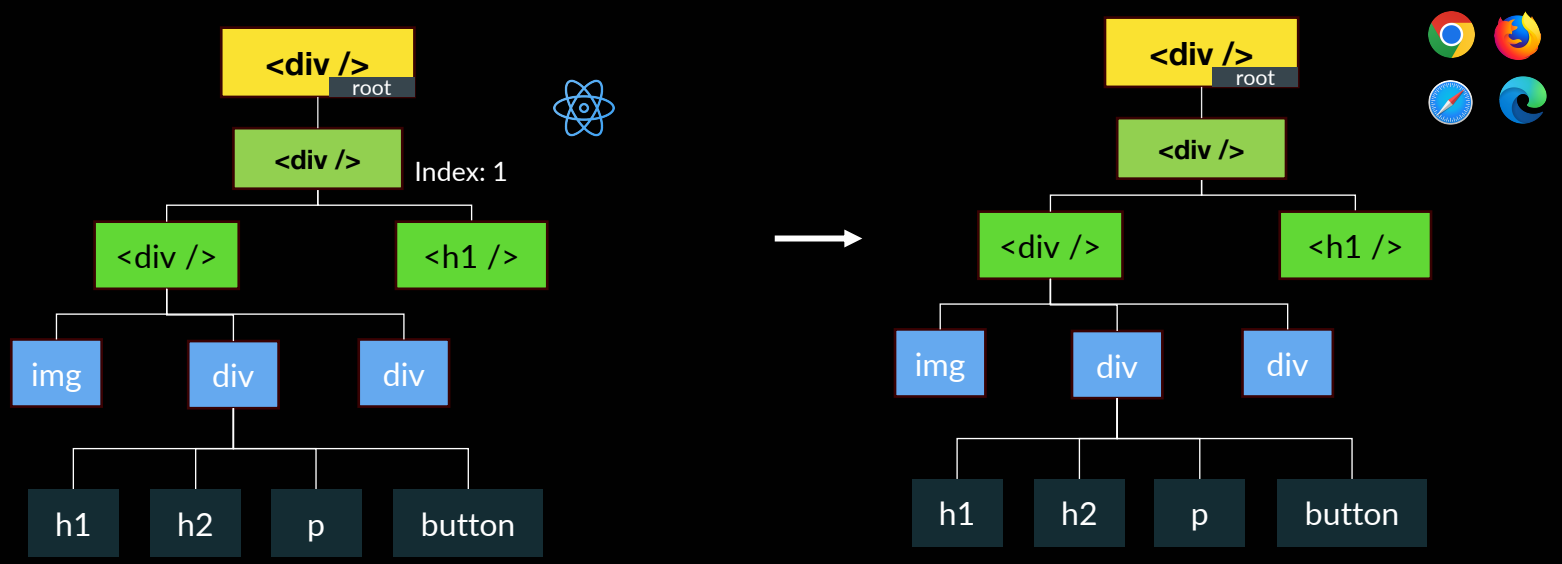
```
function onNextClick() {
  setIndex(5000000);
  setIndex(index + 3);
  setIndex((index) => index - 1);
  setIndex((index) => index + 0);
}
```

```
function onNextClick() {
  setIndex(12392912);
  setIndex((index) => index + 2);
}
```

Now there are other approachs to fulfill the same task... Which one is correct?

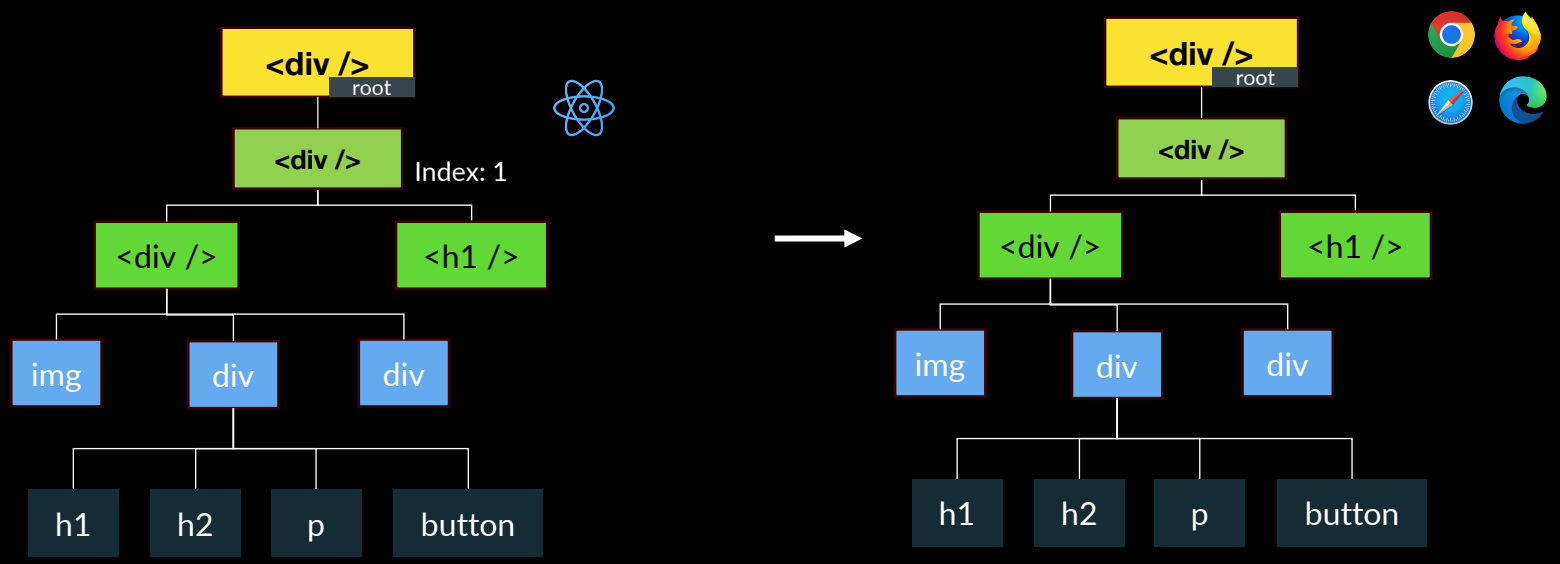
React rendering process | Re-renders

React calculates the fastest way to make a change in the DOM

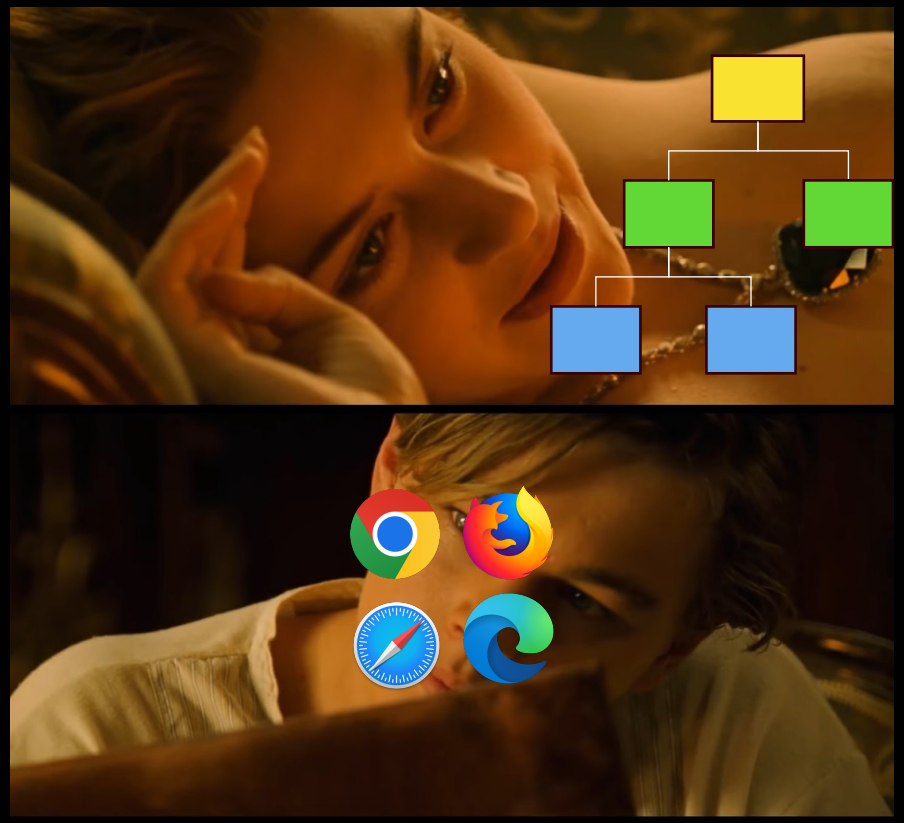


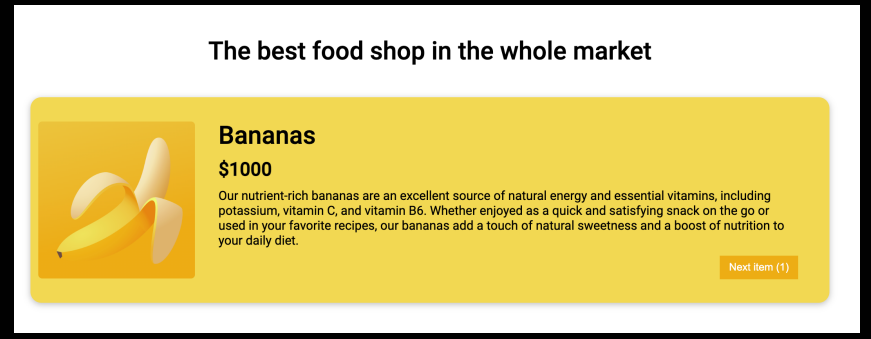
React rendering process | Re-renders

React calculates the fastest way to make a change in the DOM

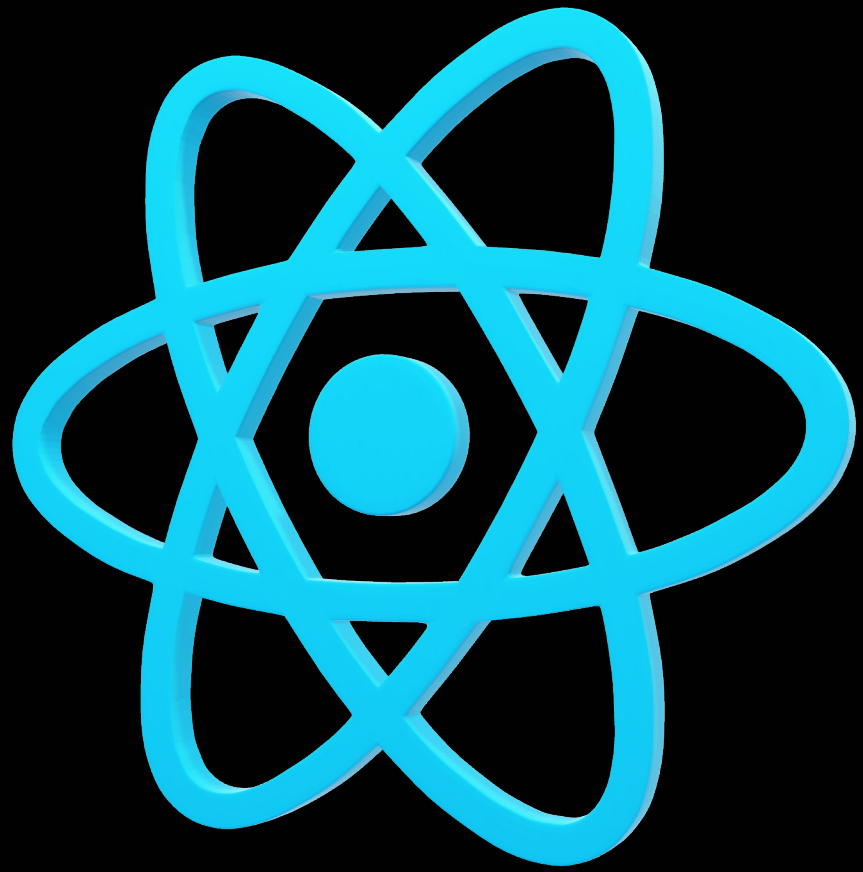


Browser Paint





Questions?



Thank you

Nicolas Imbachi –
Nicolas.ImbachiN@Perficient.com