

A large yellow square occupies the upper half of the slide. In the bottom right corner of this square, the letters 'JS' are written in a large, bold, dark grey sans-serif font.

JS

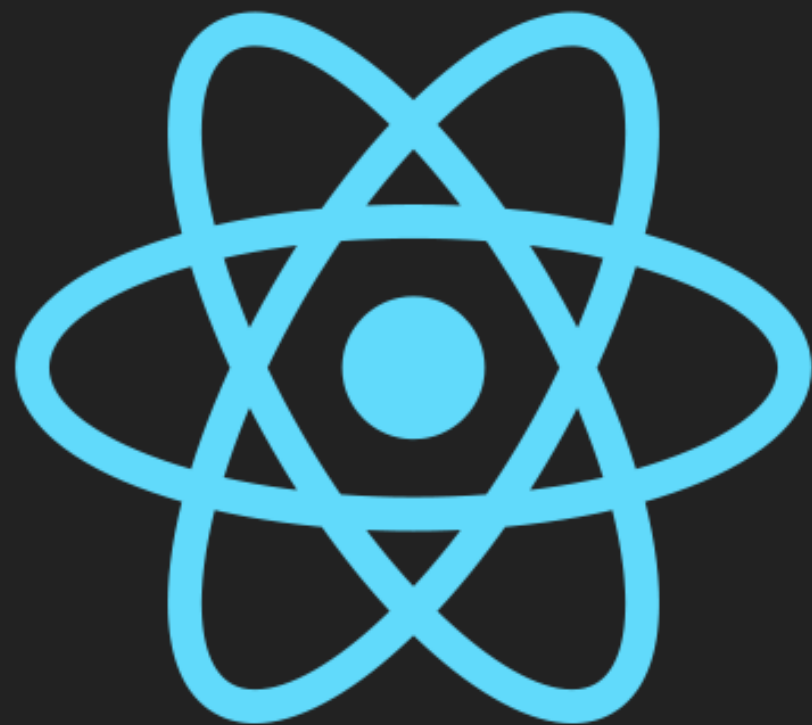
JavaScript - Lesson 2

React Basics

Intervenant

- Quentin Pré
<pre.quentin@gmail.com>
- MTI 2014
- Lead Front-End Engineer
@ Adikteev / MotionLead
- I make digital ads for a living,
- I'm pretty sure there is a
special place in hell for this.





React

First, a little grammar

Bonjour je n'arrive pas a vous invitez

Je me permet de vous envoyez

See it ?

A photograph of a middle-aged man with a grey beard and mustache, smiling broadly. He is holding a red book titled 'Bescherelle' in his right hand. The book cover has three horizontal bands: blue with 'La Conjugaison', yellow with 'L'Orthographe', and green with 'La Grammaire'. The publisher's logo 'HATIER' is at the bottom right of the cover. The background is a soft-focus outdoor scene with green leaves and a hint of a building.

Bescherelle

La Conjugaison

L'Orthographe

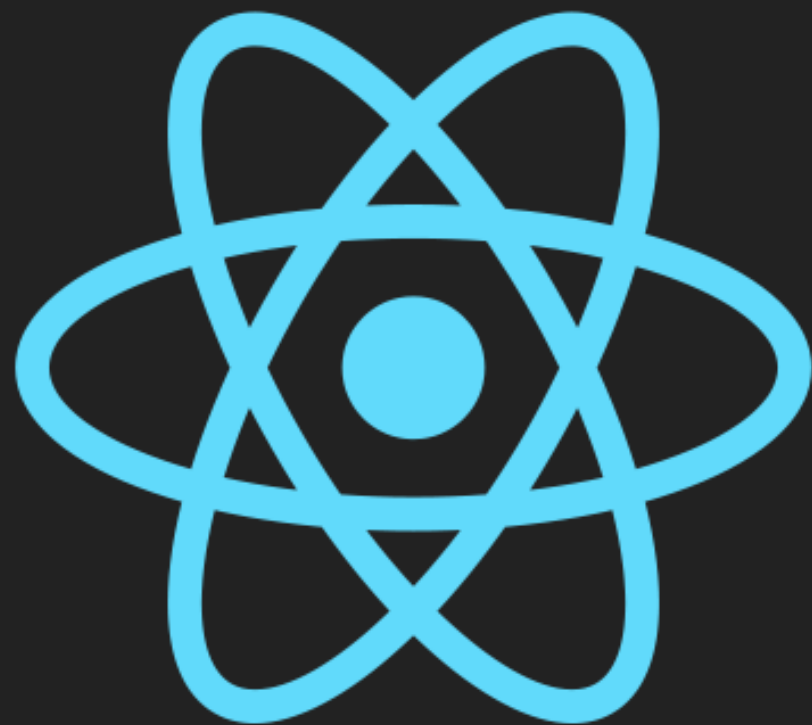
La Grammaire

HATIER

**"Réveillez-vous les enfants,
un Bescherelle, ça sert
et c'est pas cher !"**



Available here: <https://www.amazon.fr/Bescherelle-conjugaison-pour-tous-r%C3%A9f%C3%A9rence/dp/2218951983>



React

What is it ?

A framework ?

nope

A library ?

yep, but not only...

(Even though the website says:

“React is a JavaScript library for building user interfaces”)

An ecosystem ?

Yes

React key-points

- **Declarative:** describe your interface.
- **Component-based:** create your blocks, compose them, enhance them.
- **Learn once, write anywhere:** React does not only target the DOM (see ReactNative, ReactVR...)

JSX

```
const helloDiv = <div>Hello Epita !</div>;
```

Do not run away, it's **only** syntax sugar

JSX

```
const helloDiv = (  
  <div className="hello">  
    Hello Epita !  
  </div>  
)  
  
// <=>  
  
const helloDiv = React.createElement(  
  'div',  
  {className: 'hello'},  
  'Hello Epita !'  
)
```

- JSX is React's way to bring you its declarative part
- Translates straight to Javascript objects

React DOM

- Allows you to render your React tree to the DOM
- NPM package name: **react-dom**

```
const element = <h1>Hello, world</h1>;  
  
ReactDOM.render(  
  element,  
  document.getElementById('root')  
);
```

React DOM: loop([compute, diff, patch])

- Anytime an update is made, a new tree is built.
- ReactDOM compares the previous elements with the new ones
- ReactDOM only updates what has changed.

Hello, world!

It is 12:26:46 PM.



The screenshot shows a web browser's developer console with the 'Console' tab selected. It displays the React DOM tree structure for the rendered page. The root element is a `<div id='root'>` containing a `<div data-reactroot='>` which contains an `<h1>Hello, world!</h1>` and an `<h2>`. The `<h2>` contains several text nodes wrapped in `<!-- react-text: X -->` comments. The text nodes contain 'It is ', '12:26:46 PM' (highlighted in purple), and '.'. The console also shows the closing tags for `</h2>`, `</div>`, and `</div>`.

```
▼<div id="root">
  ▼<div data-reactroot=
    <h1>Hello, world!</h1>
    ▼<h2>
      <!-- react-text: 4 -->
      "It is "
      <!-- /react-text -->
      <!-- react-text: 5 -->
      "12:26:46 PM"
      <!-- /react-text -->
      <!-- react-text: 6 -->
      "."
      <!-- /react-text -->
    </h2>
  </div>
</div>
```


Components

```
const MyComponent = () => <h1>Hello, world</h1>;
```

The “atom” element in React is the component, the most basic of which is a simple function returning an element.

Components: functional

```
const MyComponent = (props) => (  
  <h1>Hello, {props.name}</h1>  
)
```

A component made of a function taking a single **props** argument and returning a React element is called a **functional component**

Components: class

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

The same component can be written using the ES6 **class** keyword, it is therefore called a **class component**

Components: render

```
ReactDOM.render(  
  <MyComponent name="quentin" />,  
  window.document.body.getElementById('root')  
)
```

This component can be rendered by using its name as a JSX tag.

Convention: components **always** start with a capital letter.

Components: props

```
const MyComponent = (props) => (  
  <h1>Hello, {props.name}</h1>  
)
```

```
ReactDOM.render(  
  <MyComponent name="quentin" />,  
  window.document.body.getElementById('root')  
)
```

- `name` is passed as a prop to `MyComponent`
- As components are **pure** functions, props are **read-only**.

Components: state

```
class MyComponent extends React.Component {  
  constructor(props) {  
    super(props);  
  
    this.state = {  
      greeting: "Hello",  
    };  
  }  
  
  render() {  
    return <h1>{this.state.greeting}, {this.props.name}</h1>  
  }  
}
```

Components can have a local state

Components: setState

- State updates are made using **setState**
- **!!! setState is asynchronous !!!**
- setState can take a callback as a second argument. It will be called when the state and the tree have been updated.

```
class Counter extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = { value: 0 };  
  }  
  
  handleClick(event) {  
    this.setState({ value: this.state.value + 1 });  
  }  
  
  render() {  
    return (  
      <div onClick={e => this.handleClick(e)}>  
        {this.state.value}  
      </div>  
    )  
  }  
}
```

Components: refs

- **refs** give you a reference to the component right after it is mounted.
- when referencing your node, the ref is called with the node as an argument.
- when changing reference, the ref is called with null before it's run again with the new reference.
- you should avoid using it as much as possible.

Components lifecycle

- Components provide hooks to their lifecycle
- These hooks allow you to “react” to changes

Components lifecycle: mounting & unmounting

- **componentWillMount:** invoked once right before the initial render
- **componentDidMount:** invoked right after the initial render
- **componentWillUnmount:** invoked right before the component is unmounted from the DOM, use it for cleanup.

Components lifecycle: mounting & unmounting

```
class Counter extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = { value: 0 };  
  }  
  
  componentWillMount() { console.log("we are about to count !"); }  
  
  componentDidMount() {  
    const update = () => {  
      this.setState(  
        { value: this.state.value + 1 },  
        () => { this.timeout = setTimeout(update, 1000) };  
      );  
    }  
  
    update();  
  }  
  
  componentWillUnmount() {  
    clearTimeout(this.timeout); // clean up your mess  
  }  
  
  render() { return <div> {this.state.value}</div>; }  
}
```

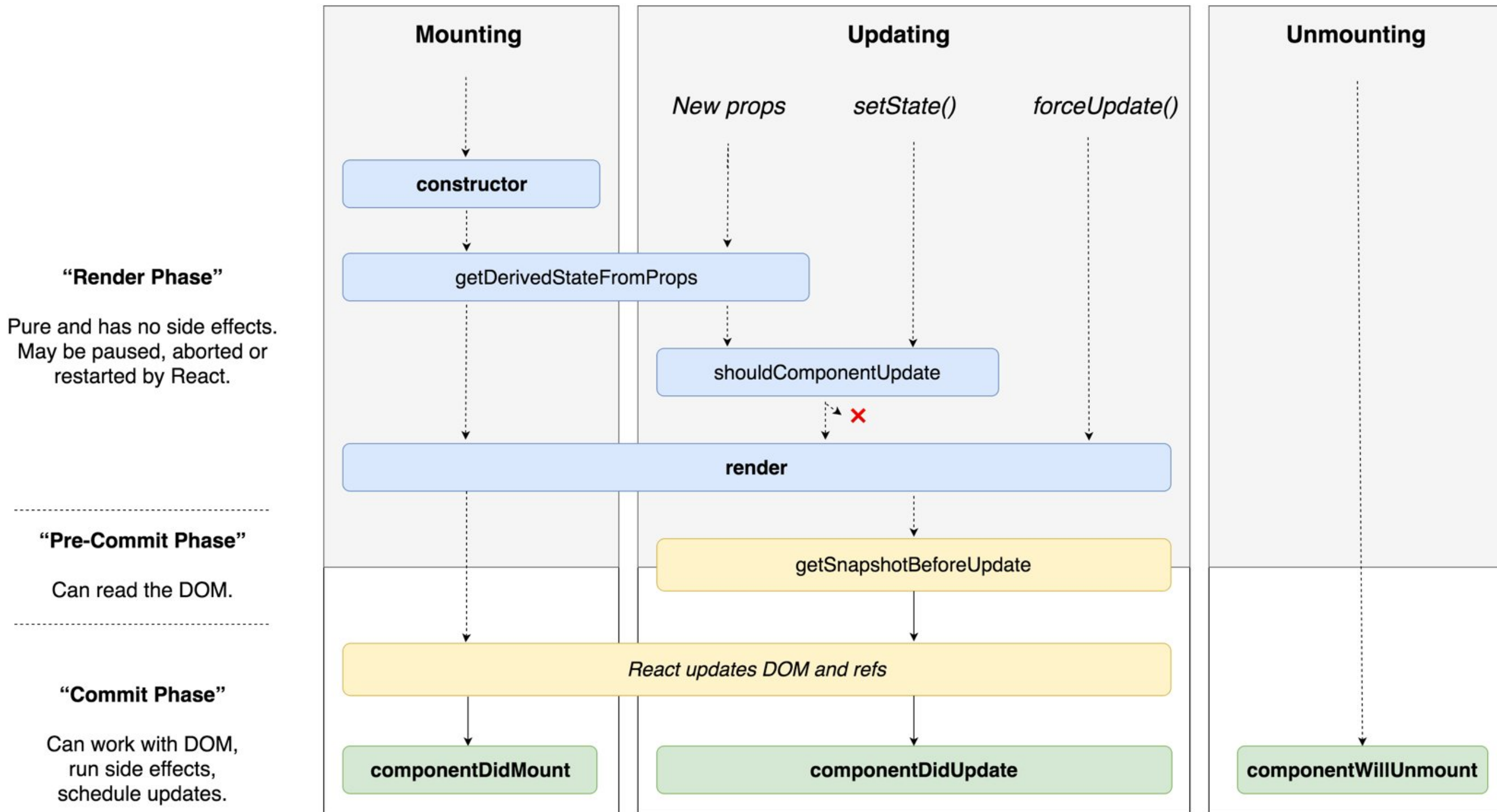
Components lifecycle: updating

- **componentWillReceiveProps**: invoked before rendering after a props update.
- **shouldComponentUpdate**: use it to tackle performance bottlenecks and discard renders that are not useful
- **componentWillUpdate**: your component is going to update any way
- **componentDidUpdate**: honestly, who uses this ?

Components lifecycle: updating

```
class Counter extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {  
      isMouseOver: false,  
      color: 'blue',  
    };  
  }  
  
  componentWillReceiveProps(nextProps, nextState) {  
    if (this.state.isMouseOver && !nextState.isMouseOver) {  
      this.setState({ color: 'blue' });  
    }  
  
    if (!this.state.isMouseOver && nextState.isMouseOver) {  
      this.setState({ color: 'green' });  
    }  
  }  
  
  render() { return <div style={{ backgroundColor: this.state.color }} />; }  
}
```

Lifecycle: sum up



Components lifecycle: after React 16.3

- *componentWillReceiveProps*, *componentWillUpdate*, *componentWillMount*, have been prefixed with **UNSAFE_**
- **static getDerivedStateFromProps(props, prevState):**
this method is now called before every render, use it to detect changes and update state consequently
this method must return a new PartialState.

Debug

- <https://github.com/facebook/react-devtools>
- Profile with `?react_perf`

More & Credits

React was initiated by Jordan Walke, who's also working on ReasonML

An EPITA alumni is part of React Native core team @vjeux (he's also been a great influence on React and also works on prettier)

A keystone of the ecosystem is Dan Abramov, co-creator of Redux and now core-member of the React team

Checkout @_chenglou's talk on the cost of abstraction
(I think he was an intern at Facebook at the time...)

Checkout Reason-React

Have a question later ?

<pre.quentin@gmail.com>

“Gouter, on va gouter !”

– Richard Bullit

You can fetch the assets for the workshop here:

<https://github.com/qpre/epita-js-tp2>