
REACT

Head of Engineering @ eMoteev

former MTI && ACU @ EPITA

Intro

QUENTIN PRÉ

This is a just a text holder in which you assume that I wrote a lot of interesting things, which obviously I have not.
If you can’t read it easily, you are too far away, get closer to the scene.
If you still have difficulties reading this, raise your hand and ask your question.

RULES

You are encouraged to make mistakes,
You are *forbidden* to make faults.

EFFORT

Do not expect knowledge to fall into your hands.

FOCUS

If you do not intend to follow the lecture, assume consequences for your actions and stay home.

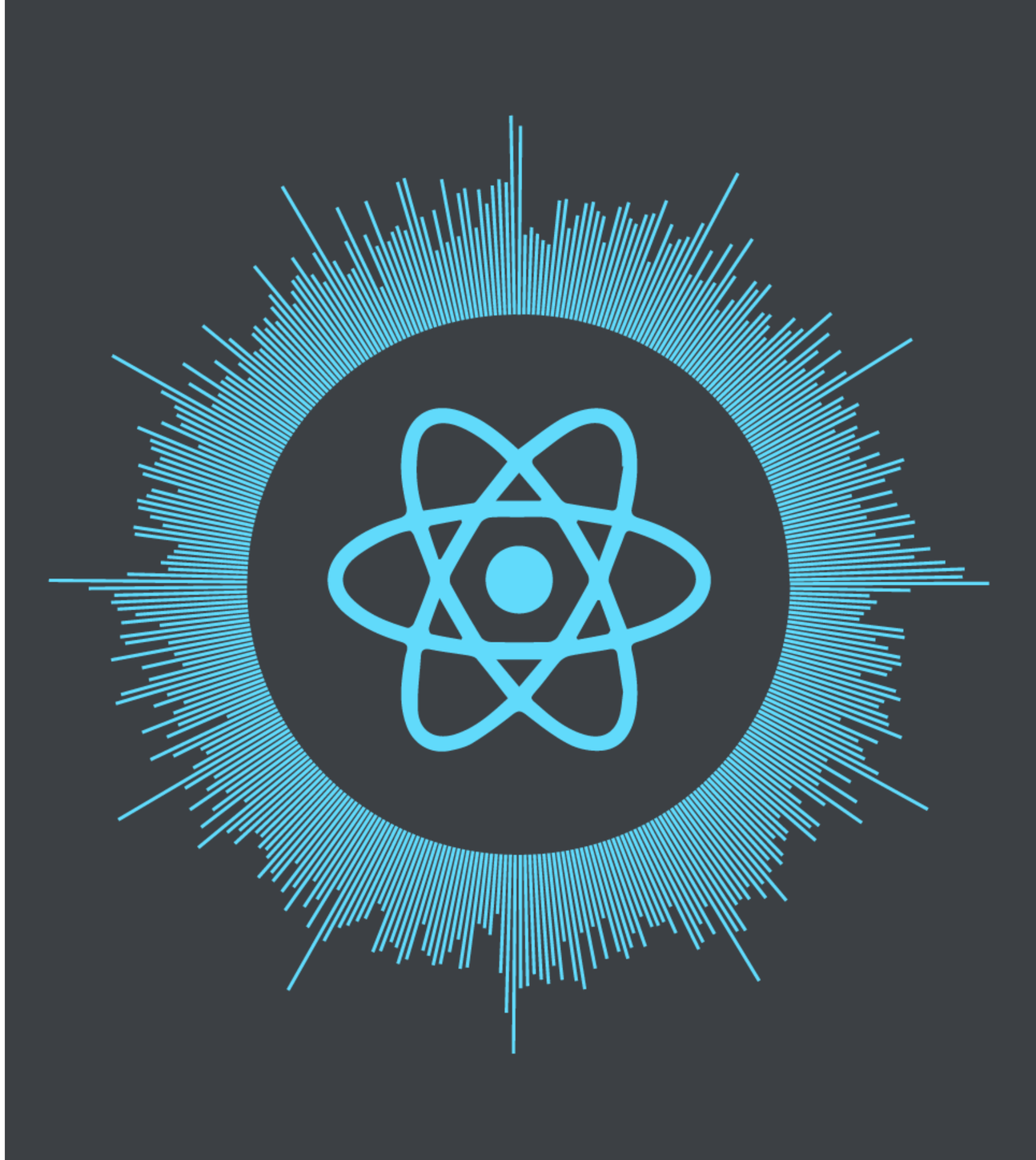
You don’t need your laptop outside of tutorials.

ASK

No question makes you stupid,
Asking no questions though…

note: your question might get answered by another question.

React ?



Is it

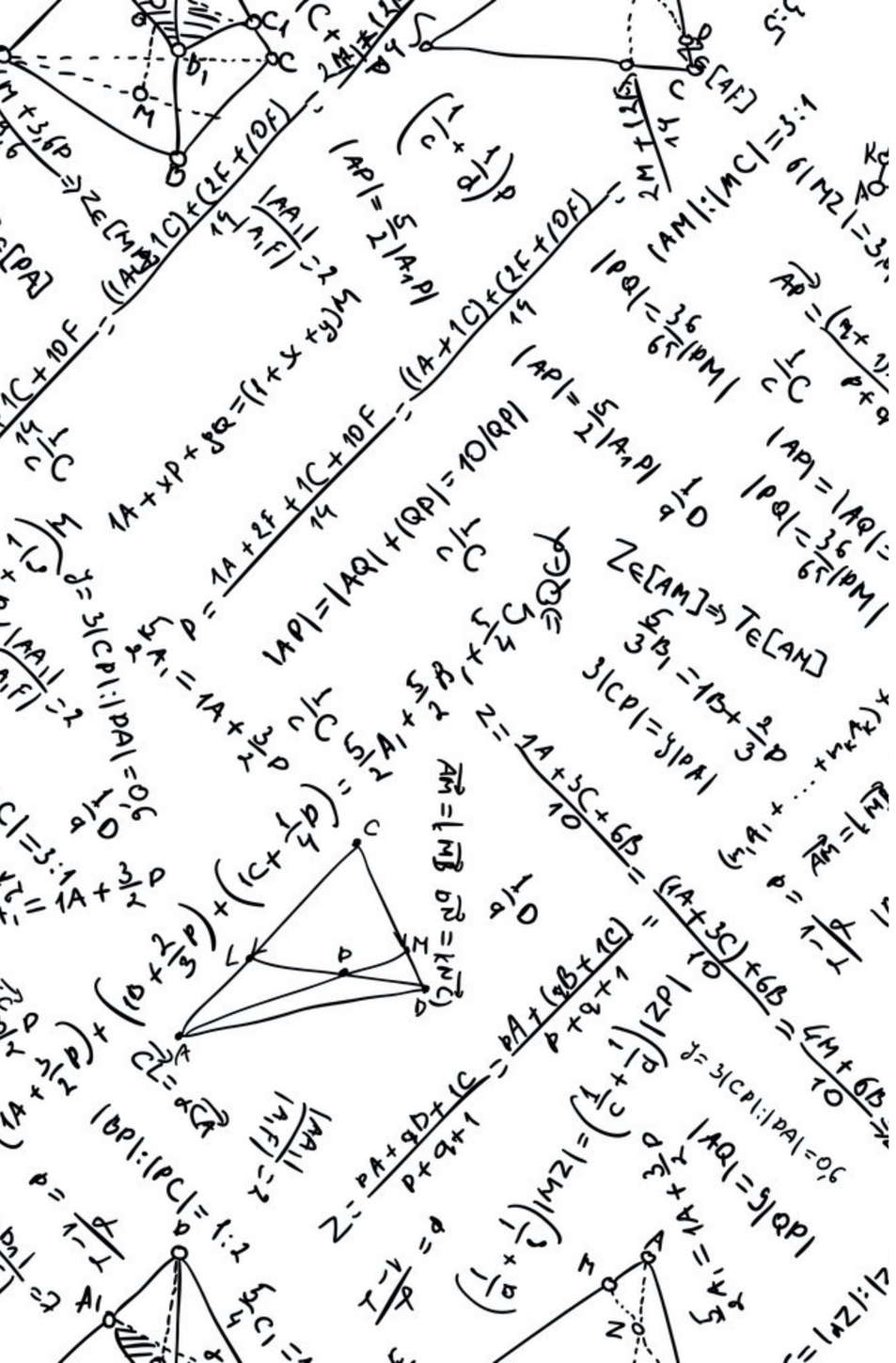
A Framework ?

Is it

A Library ?

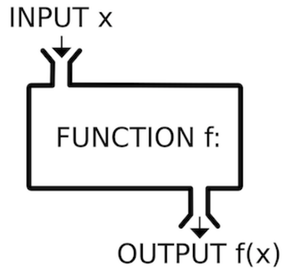
It is

An Ecosystem



A word on Maths

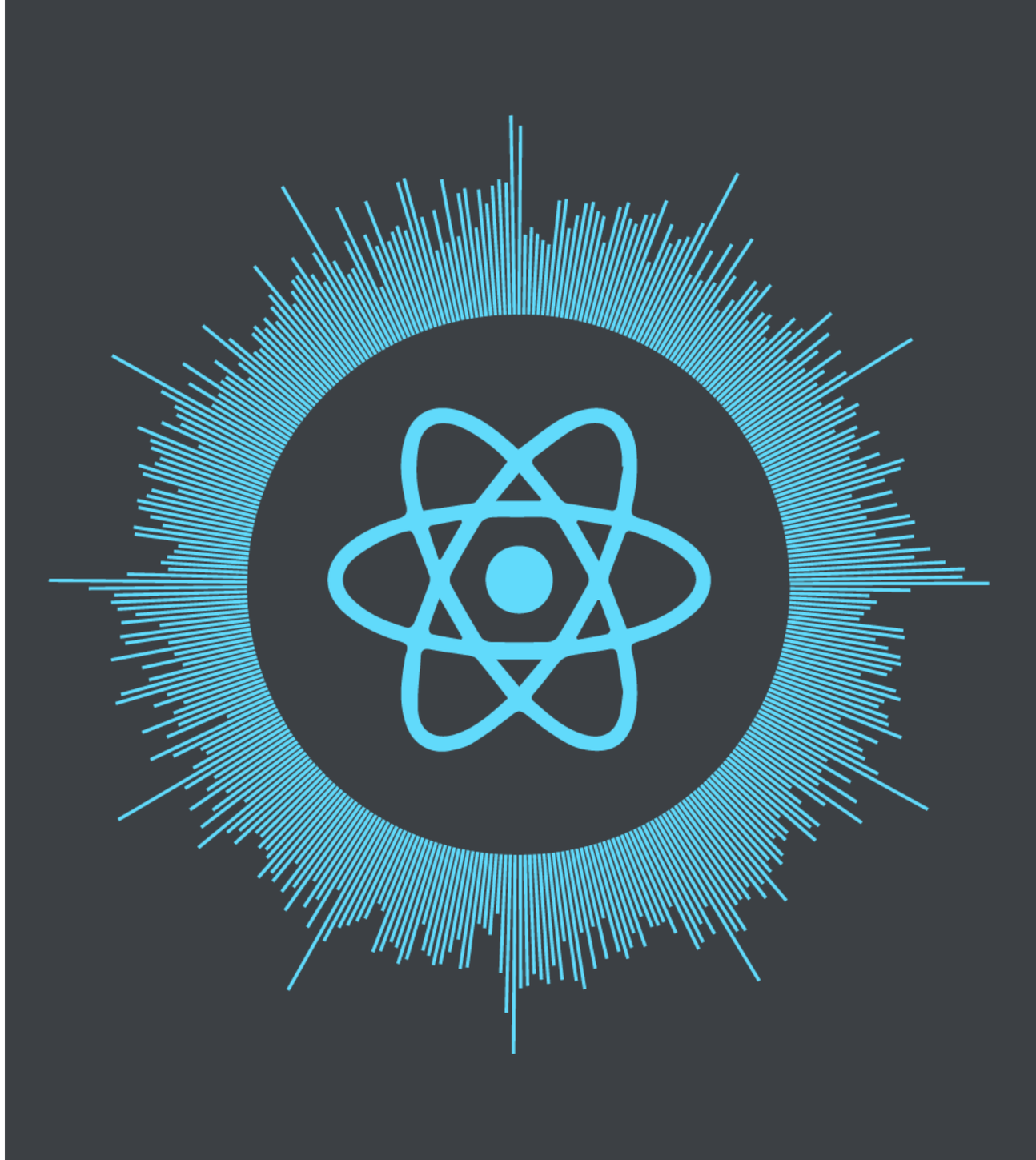
- Building a **complex** yet **consistent** GUI is **hard** due to the number of **Inputs** (keyboard, mouse, network…)
- Maths to the rescue !



- Reasoning in terms of functions and purity helps you in pushing the complexity away
- *Functional reactive programming (FRP) is a programming paradigm for reactive programming (asynchronous dataflow programming) using the building blocks of functional programming (e.g. map, reduce, filter)*

KEEP IT SIMPLE

React !



DECLARATIVE

Declarative views make your code more predictable and easier to debug.

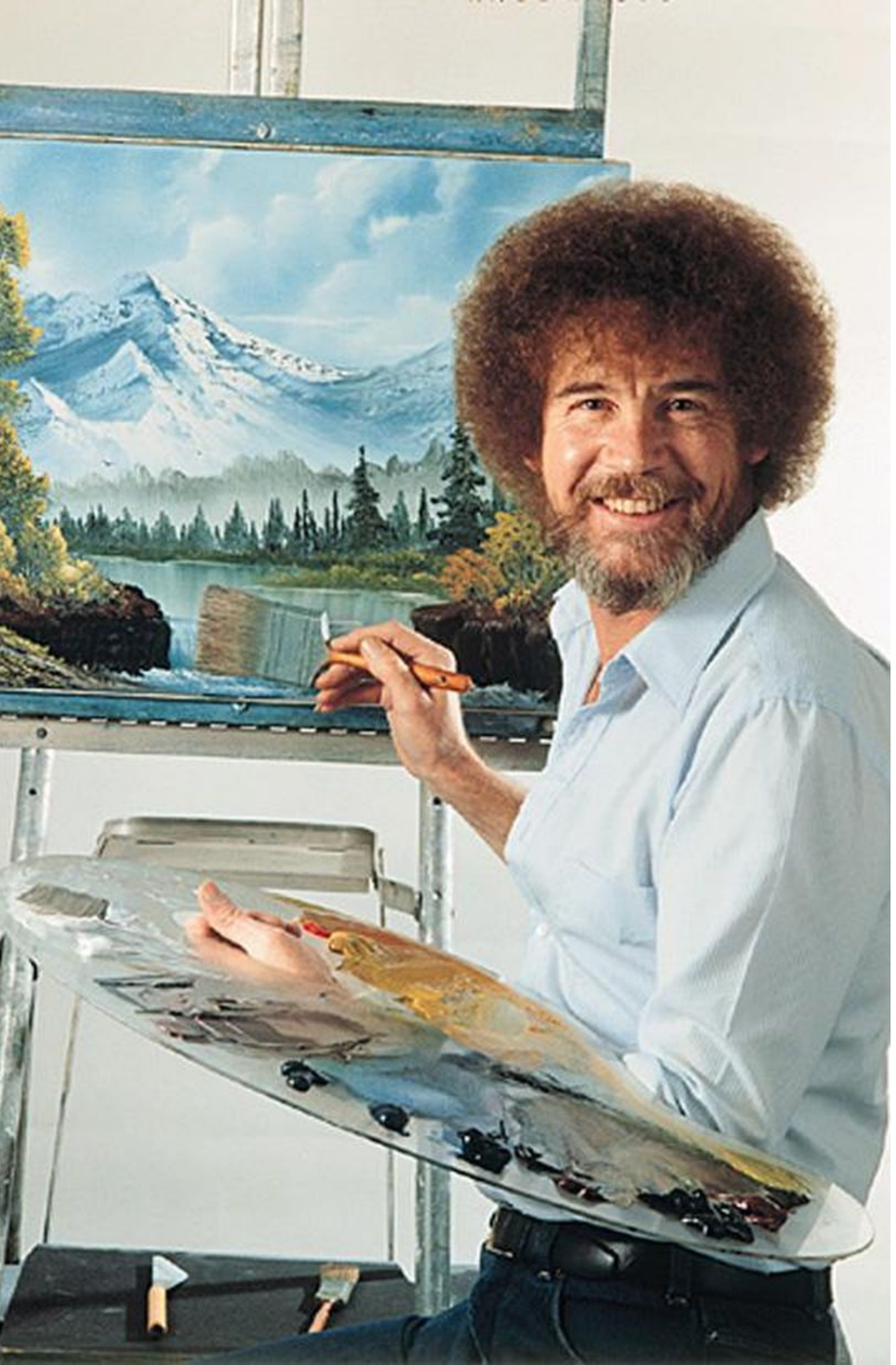
Focus on **what** to show instead of **how** to show

COMPONENT BASED

Create your blocks, compose them, enhance them.

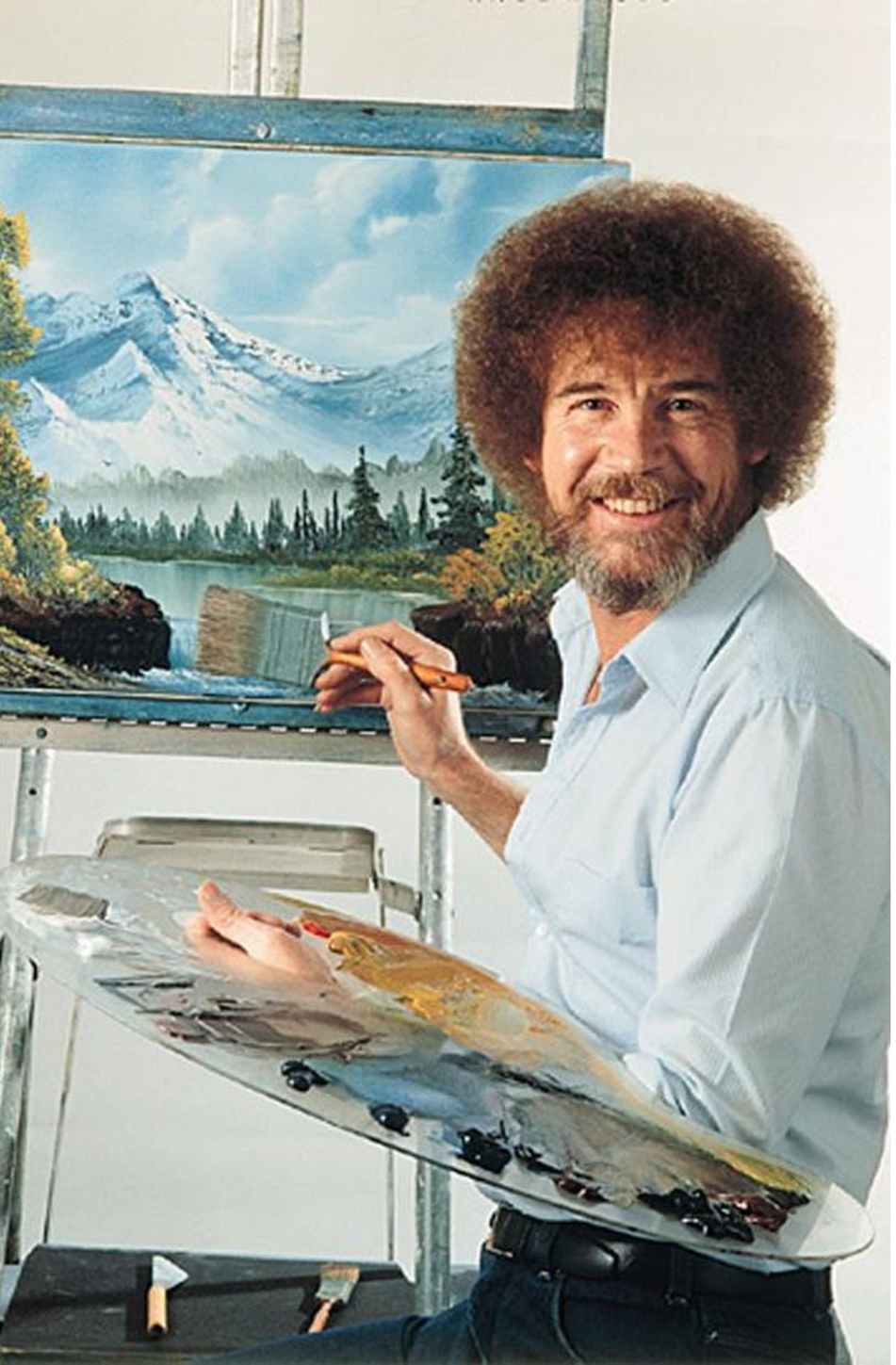
LEARN ONCE, WRITE ANYWHERE

React does not only target the DOM (see **ReactNative**, **ReactVR**...)



JSX

```
const helloDiv = <div>Hello Student !</div>
```



JSX

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

```
const helloDiv = <div className="hello">Hello Student !</div>

const helloDiv = React.createElement(
  'div',
  { className: 'hello' },
  'Hello Student !'
)
```

React DOM

- Allows you to render your React tree to the DOM
- NPM package: **react-dom**
- *import ReactDOM from ‘react-dom’;*

```
const element = <h1>Hello, world</h1>;

ReactDOM.render(
  element,
  document.getElementyId(‘root’)
);
```

Compute, diff, patch

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

Hello, world!

It is 12:26:46 PM.

Console Sources Network Timeline

```
▼<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>
```

Basic Concepts

components

Functional

- A function that returns a DOM tree
- *const MyComponent = () => <h1>hey</h1>*

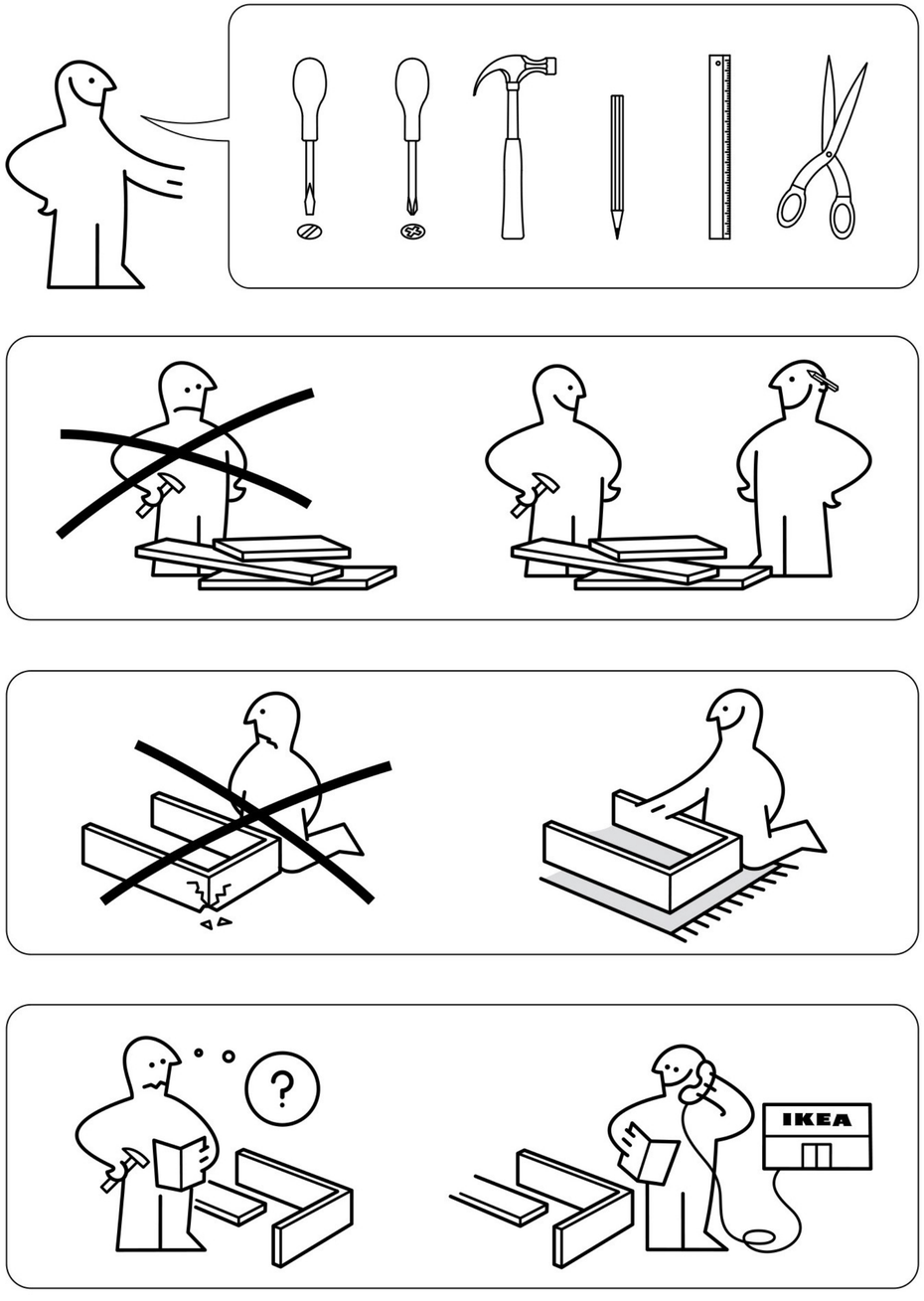
components

Class

- A class that renders a DOM tree
-

```
class MyComponent extends React.Component {
  render() {
    return (<h1>hey</h1>);
  }
}
```

Usage



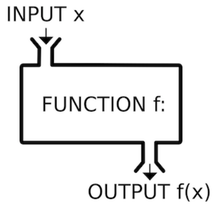
Convention: components **always** start with a capital letter.

```
const MyComponent = () => (<div className="hello">Hello Student !</div>);

ReactDOM.render(
  <MyComponent />,
  document.getElementById('root')
);
```

Components Lifecycle

Props

- Props are your component **Inputs**



- Props are the **single** argument to your components functions
- Props are passed as an **object**, allowing for **naming** parameters
- The ***children*** property allows you to nest components
- Anytime a prop changes a new render is triggered
- Props are **read-only**

Props

```
const MyComponent = ({ name }) => (
   <span>Hello, {name} !</name>
) ;

const Container = ({ children }) => (
   <div>{children}</div>
)

ReactDOM.render(
   <Container>
      <MyComponent name="quentin" />
   </Container>,
    …
);
```

A component's lifecycle

"Render Phase"

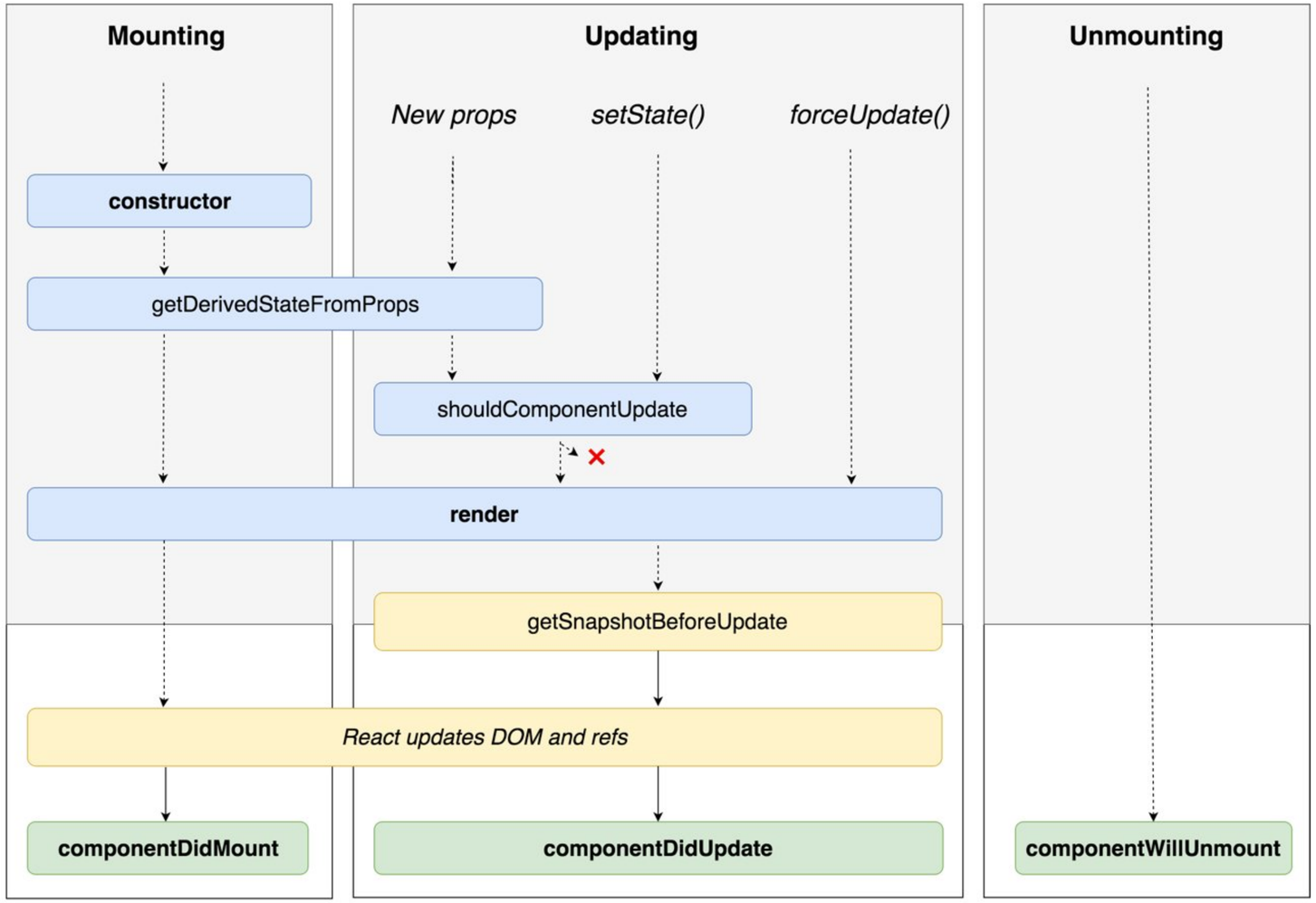
Pure and has no side effects. May be paused, aborted or restarted by React.

"Pre-Commit Phase"

Can read the DOM.

"Commit Phase"

Can work with DOM, run side effects, schedule updates.



components

Hooks

- A utility function that allow you to store state in-between renders
- Must always be in the same amount and order between renders
- - useState
  - useEffect
  - useRef
  - useCallback
  - useReducer
- More on that in a few minutes…

components

Methods

- Class methods that allow you to interact with React’s lifecycle
- - componentWillMount
  - componentDidMount
  - componentWillUnmount
  - componentWillReceiveProps
  - componentWillUpdate
  - shouldComponentUpdate
  - componentDidUpdate
- Class methods are slowly being deprecated

State

- A component can store state, it is then named a ***Stateful Component***

```
const MyStatefulComponent = () => {
  const [count, setCount] = useState(0);

  return (
    <button
      onClick={() => setCount(count + 1)}
    >
      you clicked me {count} times
    </button>
  )
}
```

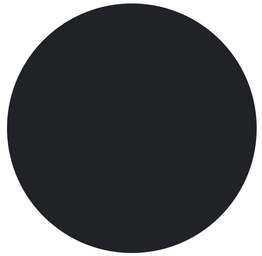

State

- Using class methods

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = { count: 0 };
  }
  render() {
    return (
    <button
      onClick={
        () => this.setState({ count: count + 1 })
      }
    >
      you clicked me {count} times
    </button>
    );
  }
}
```

```jsx
import React from 'react'
import { Card, Row, Input, Text } from './components'
import ThemeContext from './ThemeContext'

export default class Greetings extends React.Component {
  constructor(props) {
    super(props)
    this.state = {
      name: 'Mary',
      surname: 'Popins',
      width: width.innerWidth
    }

    this.handleNameChange = this.handleNameChange.bind(this)
    this.handleSurnameChange = this.handleSurnameChange.bind(this)
    this.handleResize = this.handleResize.bind(this)
  }

  componentDidMount() {
    window.addEventListener('resize', this.handleResize)
    document.title = this.state.name + ' ' + this.state.surname
  }

  componentDidUpdate() {
    document.title = this.state.name + ' ' + this.state.surname
  }

  componentWillUnmount() {
    window.removeEventListener('resize', this.handleResize)
  }

  handleNameChange(event) {
    this.setState({ name: event.target.value })
  }

  handleSurnameChange(event) {
    this.setState({ surname: event.target.value })
  }

  handleResize() {
    this.setState({ width: innerWidth })
  }

  render() {
    const { name, surname, width } = this.state

    return (
      <ThemeContext.Consumer>
        {theme => (
          <Card theme={theme}>
            <Row label="Name">
              <Input value={name} onChange={this.handleNameChange} />
            </Row>
            <Row label="Surname">
              <Input value={surname} onChange={this.handleSurnameChange} />
            </Row>
            <Row label="Width">
              <Text>{width}</Text>
            </Row>
          </Card>
        )}
      </ThemeContext.Consumer>
    )
  }
}
```

Persons of Interest



Jordan Walke

@jordwalke

Creator of React, ReasonML.

Christopher Chedeau

@vjeux

Co-creator of React Native and Prettier.
Creator of “CSS-in-JS".
Former EPITA.

Cheng Lou

@_chenglou

Member of React’s core team at Facebook

Dan Abramov

@dan_abramov

Co-creator of Redux, core-maintainer of React at Facebook

Culture

Talk: The cost of Abstraction — Cheng Lou