CLASS COMPONENT

CLASS COMPONENT

```
class Welcome extends React.Component
```

CLASS COMPONENT

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

```
const Welcome = props => {
    return <h1>Hello, {props.name}</h1>;
}
```

CLASS COMPONENT

„When should I use a function and when a class?"

~undefined

CLASS COMPONENT

DIFFERENCES

CLASS COMPONENT

SYNTAX

CLASS COMPONENT DIFFERENCES

```
var Welcome = function Welcome(props) {
  return React.createElement(
      "h1",
      null,
      "Hello, ",
      props.name);
};
```

CLASS COMPONENT DIFFERENCES

```
var Welcome = (function(_React$Component) {
    _inherits(Welcome, _React$Component);

    function Welcome() {
        _classCallCheck(this, Welcome);

        return _possibleConstructorReturn(
            this,
            (Welcome.__proto__ || Object.getPrototypeOf(Welcome)).apply(
                this,
                arguments
            )
        );
    }

    _createClass(Welcome, [
        {
            key: "render",
            value: function render() {
                return React.createElement("h1", null, "Hello, ", this.props.name);
            }
        }
    ]);

    return Welcome;
})(React.Component);
```

CLASS COMPONENT

CLASS
COMPONENT
STATE

STATE VS PROPS

CLASS
COMPONENT
STATE

PROPS

Component's configuration

Received from above

Immutable

CLASS
COMPONENT
STATE

STATE

Component's information

Created in component

Changeable

CLASS COMPONENT STATE

```
class Button extends React.Component {
    constructor() {
        super();
        this.state = {
            count: 0,
        };
    }

    render() {
      ......
    }
}
```

CLASS COMPONENT STATE

```
class Button extends React.Component {
  constructor() {
    super();
    this.state = {
      count: 0
    };
  }

  render() {
    return (
        <button>
            Clicked {this.state.count} times
        </button>
    );
  }
}
```

```
class Button extends React.Component {
  constructor() {
    super();
    this.state = {
      count: 0
    };
    this.updateCount = this.updateCount.bind(this)
  }

  updateCount() {
    this.setState((prevState, props) => {
      return { count: prevState.count + 1 };
    });
  }

  render() {
    return (
      <button onClick={this.updateCount}>
        Clicked {this.state.count} times
      </button>
    );
  }
}
```

DEMO

INITIALIZE STATE

```
this.state = {}
```

CHANGE STATE

```
this.setState()
```

setState()

```
this.setState({ count: 2 });
```

```
this.setState((state) => {
    return {count: state.count + 1};
});
```

setState()

ASYNCHRONOUS

```
constructor() {
    super();
    this.state = {
        count: 0;
    }
}

incrementCount() {
    this.setState({count: this.state.count + 1});
}

handleSomething() {
    this.incrementCount();
    this.incrementCount();
    this.incrementCount();
}
```

```
constructor() {
    super();
    this.state = {
        count: 0;
    }
}

incrementCount() {
    this.setState((state) => {
        return {count: state.count + 1}
    });
}

handleSomething() {
    this.incrementCount();
    this.incrementCount();
    this.incrementCount();
}
```

setState()

TRIGGER RENDER

REACT HOOKS

```react/1
import { useState } from 'react';

const Example = () => {
    // Declare a new state variable, which we'll call "count"
    const [count, setCount] = useState(0);

    return (
        <div>
            <p>You clicked {count} times</p>
            <button onClick={() => setCount(count + 1)}>
                Click me
            </button>
        </div>
    );
}
```

STATE
ANTI-PATTENR

```
class MyCompo        exten        mponent {
    constructo        ps
        super(p
        this.sta
            som        rops.someValue,
        };
    }
}
```

CLASS COMPONENT DIFFERENCES

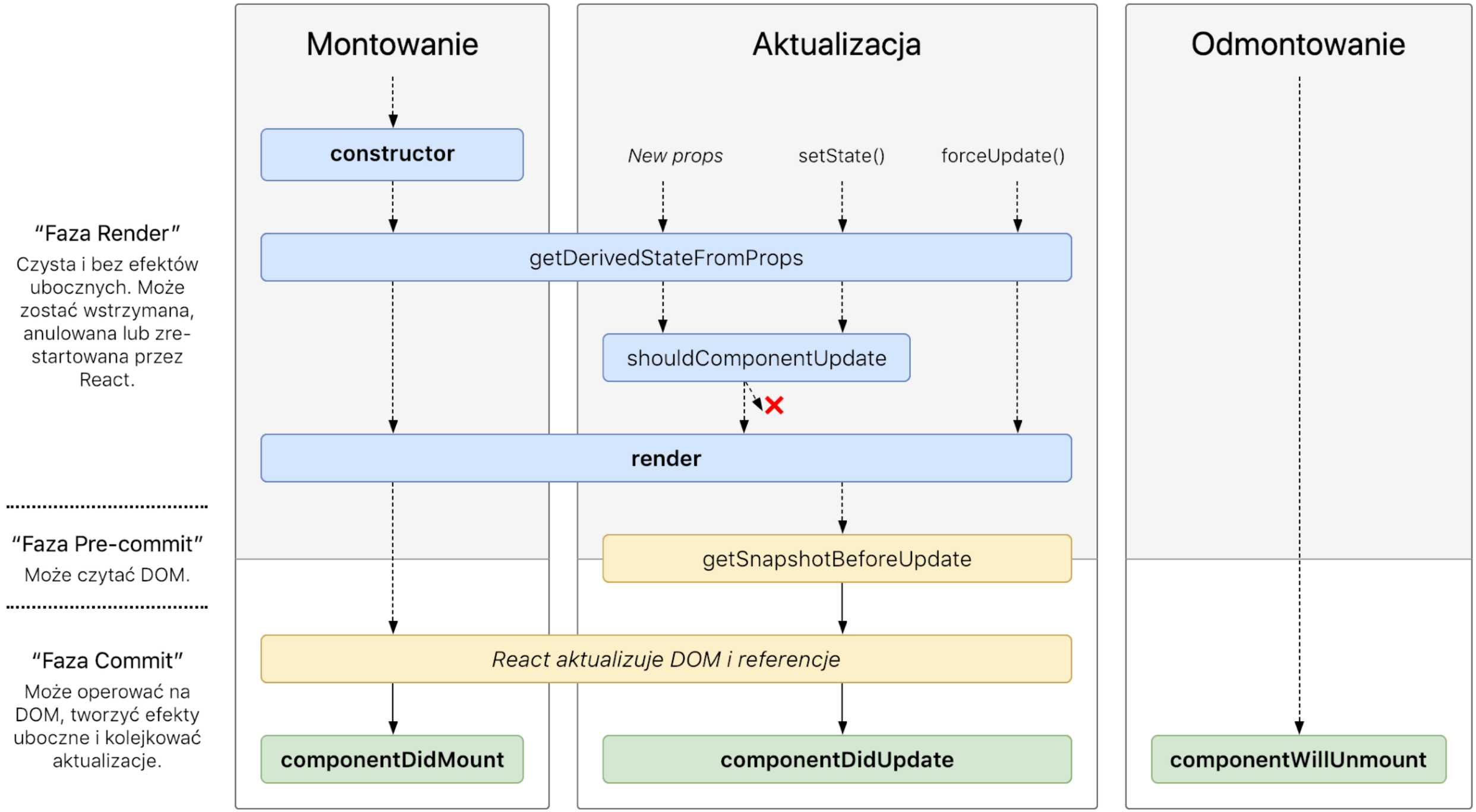
LIFECYCLE HOOKS

LIFECYCLE HOOKS

MOUNTING

UPDATING

UNMOUNTING


Montowanie
Aktualizacja
Odmontowanie
constructor
New props
setState()
forceUpdate()
getDerivedStateFromProps
shouldComponentUpdate
render
getSnapshotBeforeUpdate
React aktualizuje DOM i referencje
componentDidMount
componentDidUpdate
componentWillUnmount
"Faza Render"
Czysta i bez efektów ubocznych. Może zostać wstrzymana, anulowana lub zre-startowana przez React.
"Faza Pre-commit"
Może czytać DOM.
"Faza Commit"
Może operować na DOM, tworzyć efekty uboczne i kolejkować aktualizacje.

constructor()

initialize state (this.state)

bind methods

static getDerivedStateFromProps()

return an object to update the state

fired on every render

render()

required

componentDidMount()

load data from a remote endpoint

side-effect

DOM

shouldComponentUpdate()

performance optimization

getSnapshotBeforeUpdate()

componentDidUpdate()

invoked immediately after updating

good to do network requests

componentWillUnmount()

invoked immediately before a component is unmounted and destroyed

cleaning up

„When should I use a function and when a class?"

~undefined

BIND

```
class Foo extends Component {
  handleClick() {
    console.log("Click happened", this.state);
  }
  render() {
    return <button onClick={this.handleClick}>Click Me</button>;
  }
}
```

BIND

```
class Foo extends Component {
    constructor(props) {
        super(props);
        this.handleClick = this.handleClick.bind(this);
    }
    handleClick() {
        console.log('Click happened', this.state);
    }
    render() {
        return <button onClick={this.handleClick}>Click Me</button>;
    }
}
```

BIND

```
class Foo extends Component {
  handleClick = () => {
    console.log("Click happened", this.state);
  }
  render() {
    return <button onClick={this.handleClick}>Click Me</button>;
  }
}
```

BIND

```
class Foo extends Component {
  handleClick() {
    console.log("Click happened", this.state);
  };
  render() {
    return (
      <button onClick={this.handleClick.bind(this)}>
        Click Me
      </button>
    );
  }
}
```

BIND

```
class Foo extends Component {
  handleClick() {
    console.log("Click happened", this.state);
  }
  render() {
    return (
      <button onClick={() => this.handleClick()}>
        Click Me
      </button>
    );
  }
}
```

AJAX & APIS

```
class App extends React.Component {
  state = {
    user: null
  };

  async componentDidMount() {
    const response = await fetch("https://my.awesome.api/v1/");
    const data = await response.json();
    const [user] = data.results;
    this.setState({ user });
  }

  render() {
    const { user } = this.state;

    return (
      <div>
        <img src={user.avatar} alt="avatar" />
        Name: {user.name}
        E-mail: {user.email}
      </div>
    );
  }
}
```

← → 1 of 2 errors on the page

TypeError: Cannot read property 'avatar' of null

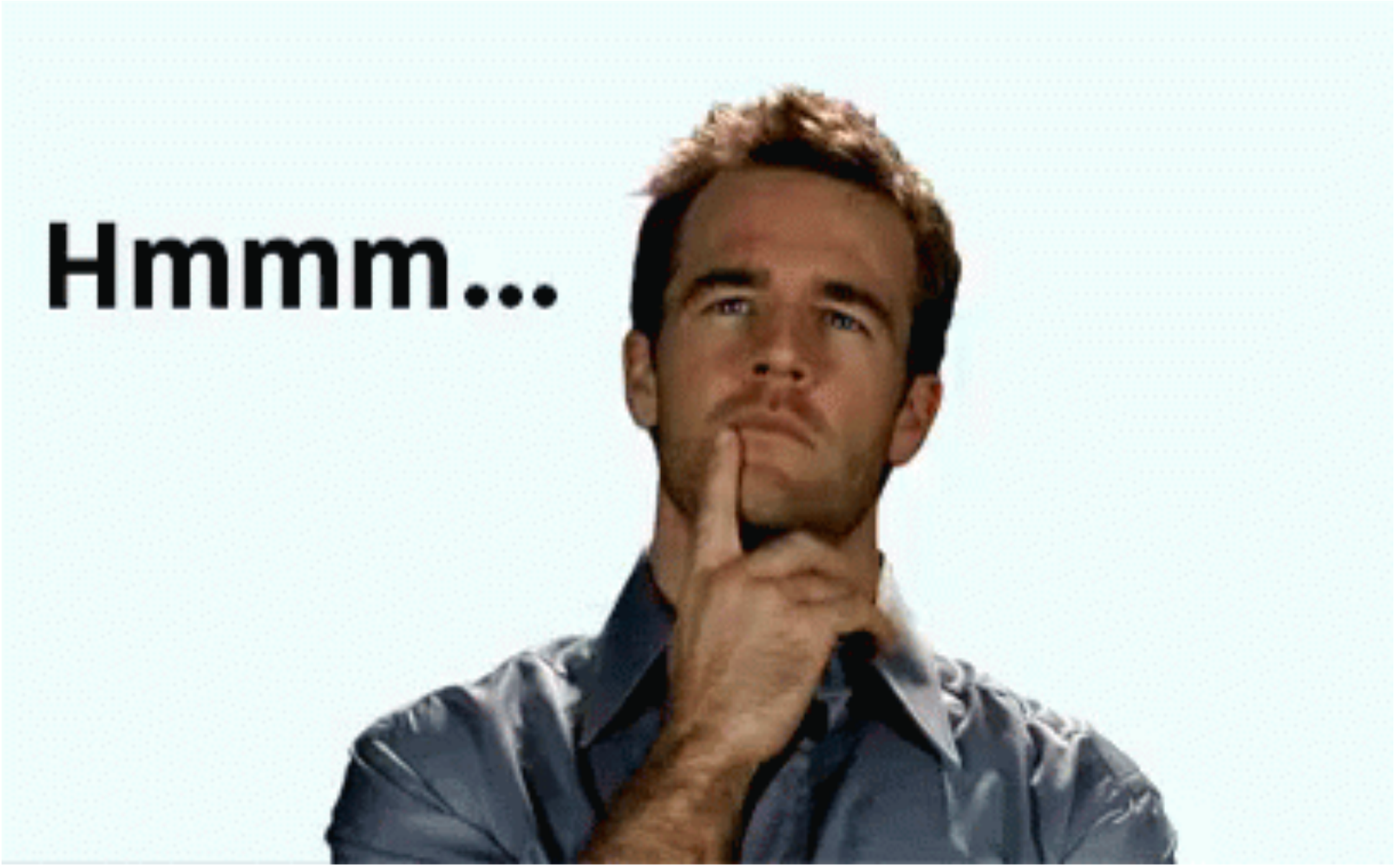
App.render
src/App.js:24

```
  21 |
  22 | return (
  23 |   <div>
> 24 |     <img src={user.avatar} alt="avatar" />
  25 | ^   Name: {user.name}
  26 |     E-mail: {user.email}
  27 |   </div>
```

View compiled

```
class App extends React.Component {
  state = {
    isLoaded: false,
    user: null
  };

  async componentDidMount() {
    ///…pobieranie danych
    this.setState({
      user,
      isLoaded: true
    });
  }

  render() {
    const { user, isLoaded } = this.state;

    if(!isLoaded) {
      return <div>Loading...</div>;
    }

    ///…dalsza część render
  }
}
```

ERRORS?



```
class App extends React.Component {
  state = {
    error: null,
    isLoaded: false,
    user: null
  };

  async componentDidMount() {
    try {
      ///…pobieranie danych
      this.setState({
        user,
        isLoaded: true
      });
    } catch (e) {
      this.setState({
        error: e.message,
     });
    }
  }

  render() {
    const { user, isLoaded, error } = this.state;

    if (error) { return <div>{error}</div>; }

    if (!isLoaded) { return <div>Loading…</div>; }

    ///…dalsza część render
```

UNCONTROLLED COMPONENTS

UNCONTROLLED COMPONENTS

```
class Form extends Component {
  render() {
    return (
        <div>
          <input type="text" />
        </div>
    );
  }
}
```

UNCONTROLLED COMPONENTS

Form data handled by DOM

UNCONTROLLED COMPONENTS

```
// Create ref
this.inputRef = React.createRef();
```

```
// Read value
this.inputRef.current.value;
```

```
<input type="text" ref={this.inputRef}/>
```

INPUT [TYPE=FILE]

Always uncontrolled

INPUT [TYPE=FILE]

```
this.fileInputRef = React.createRef();
```

```
this.fileInputRef.current.files;
```

```
<input type="file" ref={this.fileInputRef} />
```

CONTROLLED COMPONENTS

CONTROLLED COMPONENTS

Form data handled by

REACT Component

CONTROLLED COMPONENTS

```
<input type="text" value={value} onChange={handleChange} />
```

CONTROLLED COMPONENTS

```
class Form extends React.Component {
  state = {
    name: ""
  };

  handleChange = event => {
    this.setState({ name: event.target.value });
  };

  render() {
    return (
      <input type="text" value={this.state.name} onChange={this.handleChange} />
    );
  }
}
```

CONTROLLED VS UNCONTROLLED

feature	uncontrolled	controlled
one-time value retrieval (e.g. on submit)		
validating on submit		
instant field validation		
conditionally disabling submit button		
enforcing input format		
several inputs for one piece of data		
dynamic inputs		