Școala
informală
de IT

React Props, State, Events

Agenda

- What are props?
- Using props. Passing data using props.
- What is state?
- Using state. Update Component state
- Props vs state
- Using axios to fetch data
- React Events
- React Refs

What are props?

- short for “properties”
- used to pass data between React components
- data flow between components is uni-directional

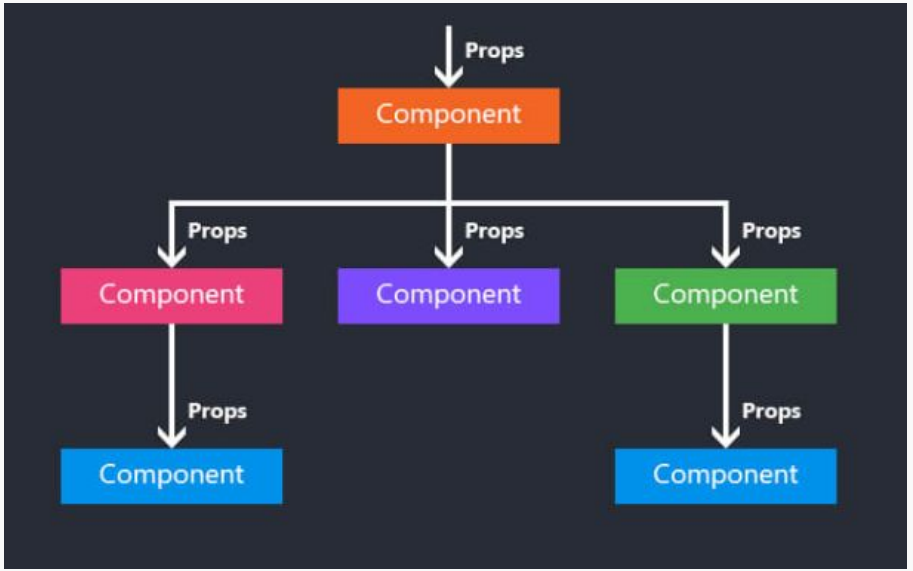
```jsx
function Greeting(props) {
  return (
    <div>
      <h1>Hello, {props.name}!</h1>
    </div>
  );
}

function App() {
  return (
    <div>
      <Greeting name="React" />
      <div>
        <Greeting name="Chris" />
      </div>
    </div>
  );
}
```

props is first argument to function components

Access props inside of curly braces to show value

Set a prop on a child component by passing an attribute

Different instances of component can have different prop values



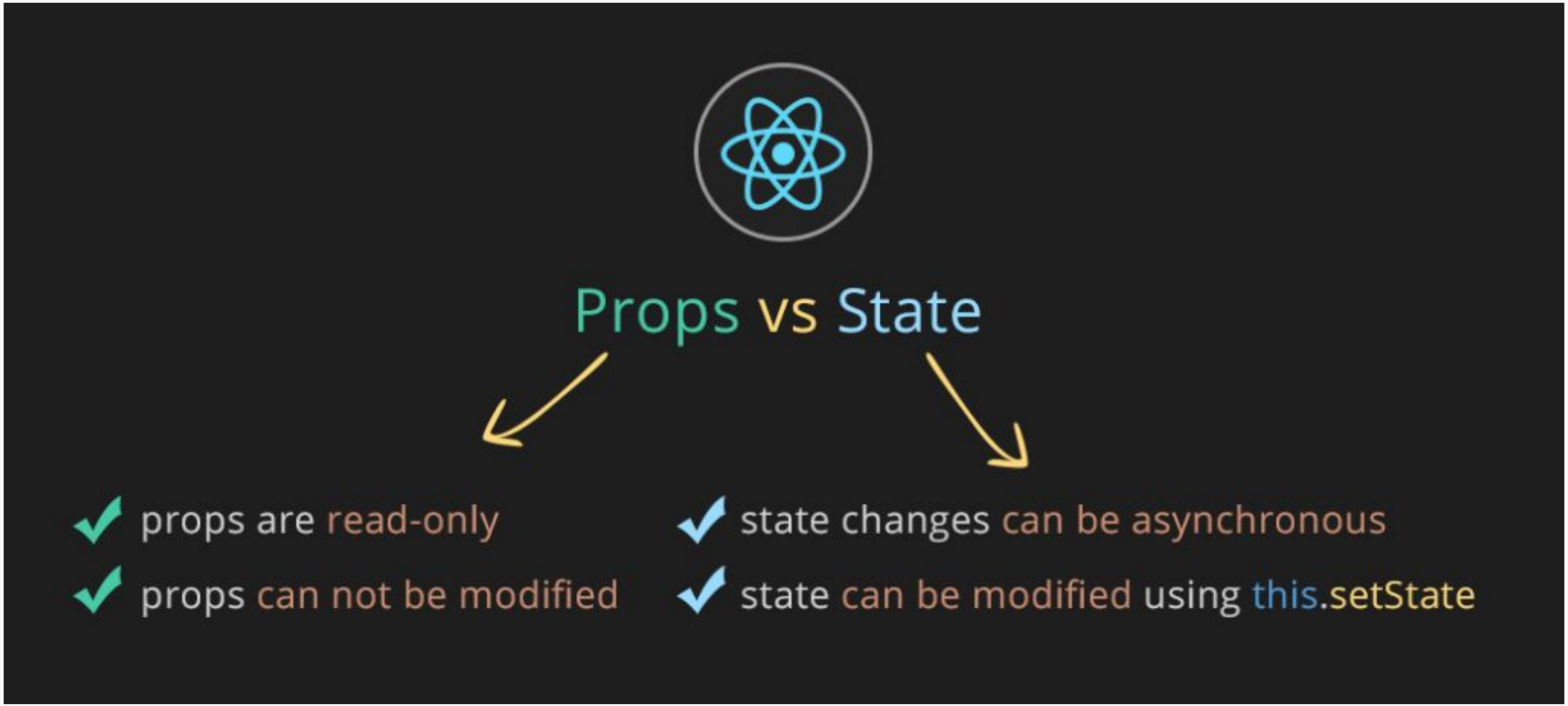
What is state?

- a plain JavaScript object used by React to represent component current information
- a “normal” variable “disappears” when their function exits
- state variables are preserved by React.
- local state is a feature only available to classes.

```
class App extends React.Component {
  constructor() {
    super()
    this.state = {
      name: 'Bob',
      isLoggedIn: false
    }
  }

  handleLogIn = () => {
    this.setState({isLoggedIn: true})
  }
```

```
function 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>
  );
}
```

Props vs state



Using axios to fetch data

- HTTP client library that allows you to make requests to a given endpoint
- Axios has function names that match any HTTP methods
  - `get(), post(), put(), delete()`
- Unlike the Fetch API, you only need one `.then()` callback to access your requested JSON data
- Axios has better error handling. Axios throws 400 and 500 range errors for you
- Axios can be used on the server as well as the client. If you are writing a Node.js application

```
npm install axios
```

GET Request using axios

```
import axios from 'axios';
import React from 'react';

const baseURL = 'https://jsonplaceholder.typicode.com/posts/1';

export default function App() {
   const [post, setPost] = React.useState(null);

   React.useEffect(() => {
      axios.get(baseURL).then((response) => {
         setPost(response.data);
      });
   }, []);

   if (!post) return null;

   return (
      <div>
         <h1>{post.title}</h1>
         <p>{post.body}</p>
      </div>
   );
}
```

POST Request using axios

```
import axios from 'axios';
import React from 'react';

const baseURL =
'https://jsonplaceholder.typicode.com/posts';

export default function App() {
   const [post, setPost] = React.useState(null);

   React.useEffect(() => {
      axios.get(`${baseURL}/1`).then((response) => {
         setPost(response.data);
      });
   }, []);
```

```
   function createPost() {
      axios
         .post(baseURL, {
            title: 'Hello World!',
            body: 'This is a new post.',
         })
         .then((response) => {
            setPost(response.data);
         });
   }

   if (!post) return 'No post!';

   return (
      <div>
         <h1>{post.title}</h1>
         <p>{post.body}</p>
         <button onClick={createPost}>Create</button>
      </div>
   );
}
```

PUT Request using axios

```
import axios from 'axios';
import React from 'react';

const baseURL =
'https://jsonplaceholder.typicode.com/posts';

export default function App() {
   const [post, setPost] = React.useState(null);

   React.useEffect(() => {
      axios.get(`${baseURL}/1`).then((response) => {
         setPost(response.data);
      });
   }, []);
```

```
function updatePost() {
      axios
         .put(`${baseURL}/1`, {
            title: 'Hello World!',
            body: 'This is an updated post.',
         })
         .then((response) => {
            setPost(response.data);
         });
   }

   if (!post) return 'No post!';

   return (
      <div>
         <h1>{post.title}</h1>
         <p>{post.body}</p>
         <button onClick={updatePost}>Update
Post</button>
      </div>
   );
}
```

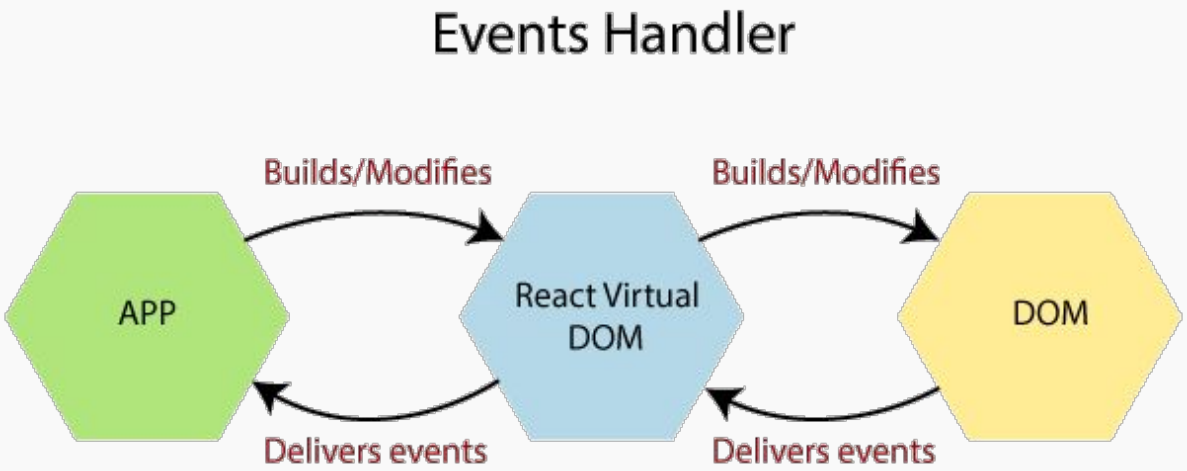
Delete Request using axios

```
import axios from 'axios';
import React from 'react';

const baseURL =
'https://jsonplaceholder.typicode.com/posts';

export default function App() {
   const [post, setPost] = React.useState(null);

   React.useEffect(() => {
      axios.get(`${baseURL}/1`).then((response) => {
         setPost(response.data);
      });
   }, []);
```

```
function deletePost() {
      axios.delete(`${baseURL}/1`).then(() => {
         alert('Post deleted!');
         setPost(null);
      });
   }

   if (!post) return 'No post!';

   return (
      <div>
         <h1>{post.title}</h1>
         <p>{post.body}</p>
         <button onClick={deletePost}>Delete
Post</button>
      </div>
   );
}
```

React Events

- rather than calling `addEventListener`, as you would in Vanilla JavaScript, you provide an event listener when the element is initially rendered.
- React events are named in `camelCase` (like `onClick`), rather than all lowercase.
- Using JSX, you pass a function as the event handler, rather than a string.



React Events - example

```
class App extends React.Component {
   constructor(props) {
      super(props);
      this.state = {
         companyName: '',
      };
   }
   changeText(event) {
      this.setState({
         companyName: event.target.value,
      });
   }
   render() {
      return (
         <div>
            <h2>Simple Event Example</h2>
            <label htmlFor="name">Enter company name: </label>
            <input
               type="text"
               id="companyName"
               onChange={this.changeText.bind(this)}
            />
            <h4>You entered: {this.state.companyName}</h4>
         </div>
      );
   }
}
```

What are refs?

- short for “references”
- attribute which makes it possible to store a reference to particular DOM nodes or React elements
- used when we want to change the value of a child component, without making the use of props.
- use when we need DOM measurements such as managing focus, text selection
- should be avoided for anything that can be done **declaratively**
- For example, instead of using **open()** and **close()** methods on a Dialog component, you need to pass an **isOpen** prop to it.

How to create refs

```
class MyComponent extends
React.Component {
    constructor(props) {
        super(props);
        this.callRef = React.createRef();
    }
    render() {
        return <div ref={this.callRef} />;
    }
}
```

```
import { useRef } from 'react';
function AccessingElement() {
    const callRef = useRef();

    return <div ref={callRef}>I'm an
element</div>;
}
```

How to access and use refs - class component

```
class App extends React.Component {
   constructor(props) {
     super(props)

     this.textInput = React.createRef();
     this.state = {
       value: ''
     }
   }

 handleSubmit = e => {
   e.preventDefault();
   this.setState({ value:
this.textInput.current.value})
 };
```

```
 render() {
   return (
     <div>
       <h1>React Ref - createRef</h1>
       <h3>Value: {this.state.value}</h3>
       <form onSubmit={this.handleSubmit}>
         <input type="text" ref={this.textInput} />
         <button>Submit</button>
       </form>
     </div>
   );
 }
}

ReactDOM.render(<App />,
document.getElementById("root"));
```