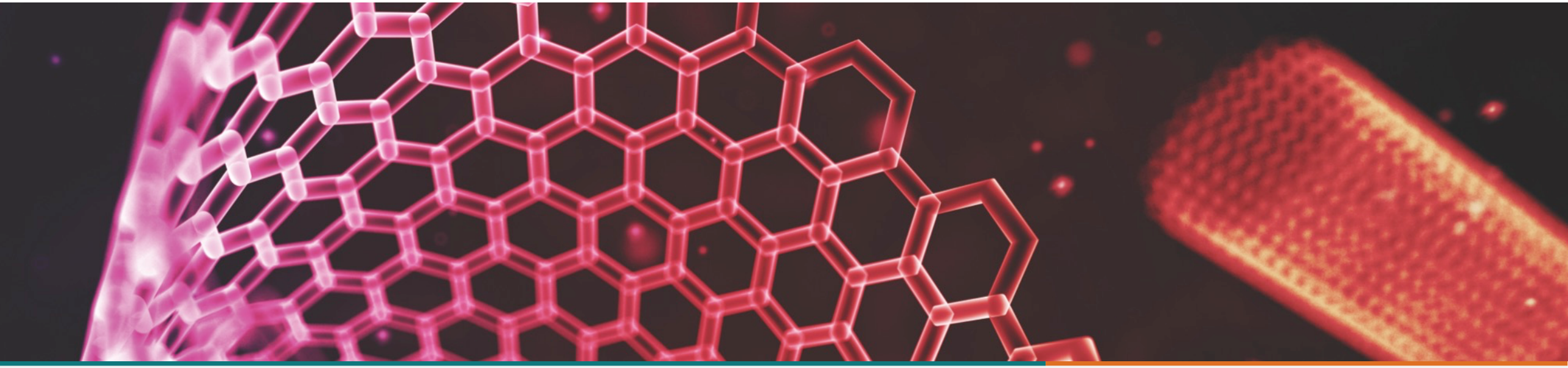
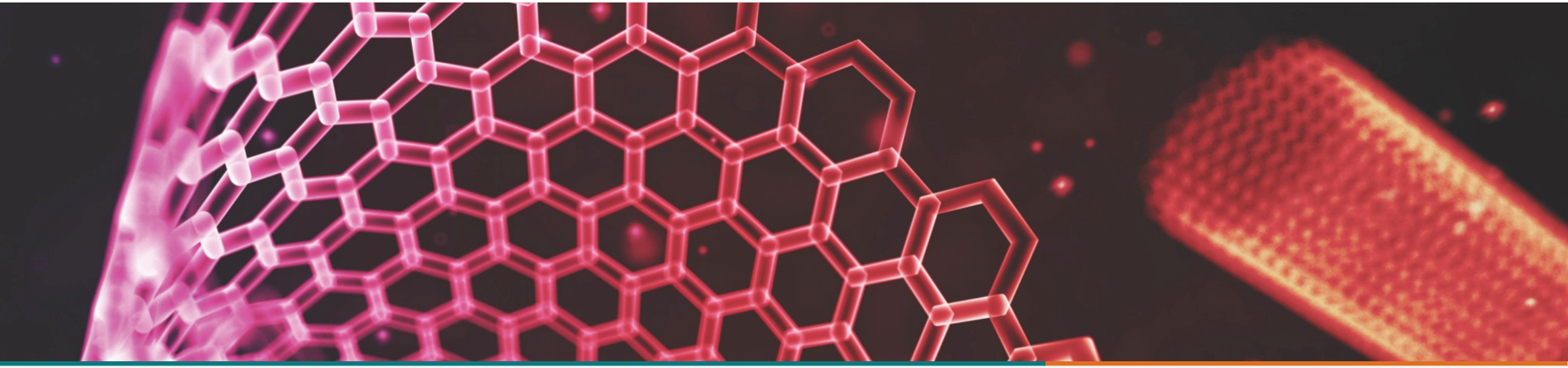
STEVENS
INSTITUTE of TECHNOLOGY
Schaefer School of Engineering & Science

# CS 554 – Web Programming II

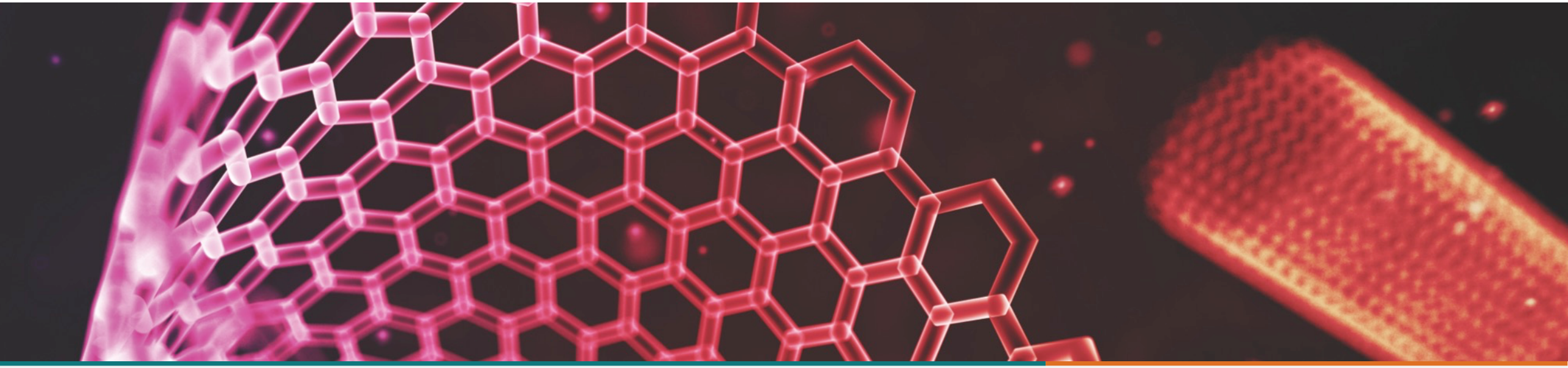
# React - Continued



# Event Handling and Form Processing



# Event Handling



# Event Handling

- React events are named using camelCase, rather than lowercase.
- With JSX you pass a function as the event handler, rather than a string.

For example, the HTML:

```
<button onclick="activateLasers()">
  Activate Lasers
</button>
```

is slightly different in React:

```
<button onClick={activateLasers}>
  Activate Lasers
</button>
```

# Event Handling Example & Binding in Class Components

```
class Toggle extends React.Component {
  constructor(props) {
    super(props);
    this.state = {isToggleOn: true};

    // This binding is necessary to make `this` work in the callback
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    this.setState(state => ({
      isToggleOn: !state.isToggleOn
    }));
  }

  render() {
    return (
      <button onClick={this.handleClick}>
        {this.state.isToggleOn ? 'ON' : 'OFF'}
      </button>
    );
  }
}

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

Notice: **this.handleClick =this.handleClick.bind(this)** in the constructor. This is one way to bind the function.  We will see two other ways to bind the function in the next slide

# Event Handling Example and Binding

```
<input
    type="text"
    name="searchTerm"
    onChange={this.handleChange.bind(this)}
/>
```

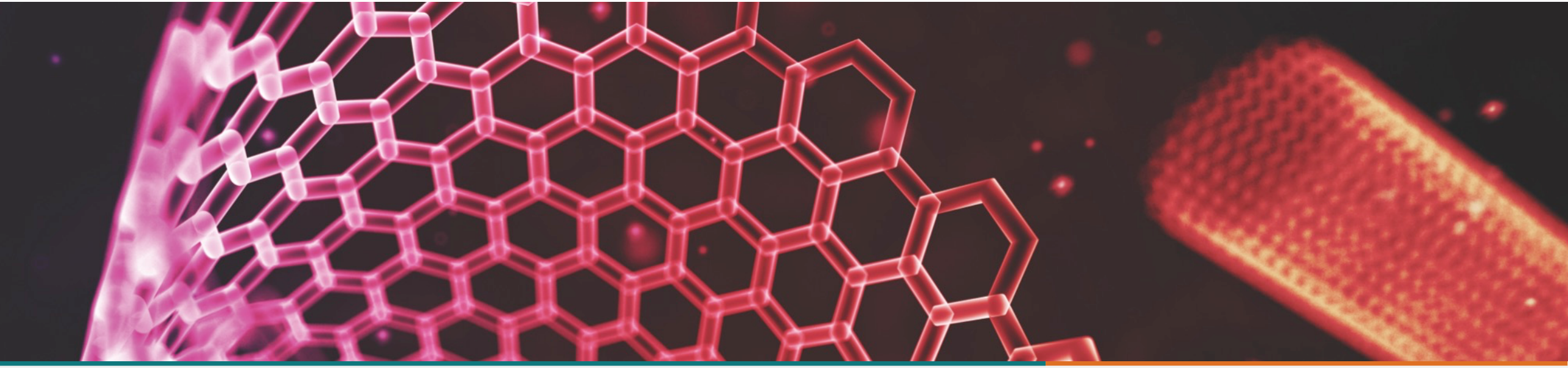
We can bind it right in the change event

```
class LoggingButton extends React.Component {
  // This syntax ensures `this` is bound within handleClick.
  // Warning: this is *experimental* syntax.
  handleClick = () => {
    console.log('this is:', this);
  }

  render() {
    return (
      <button onClick={this.handleClick}>
        Click me
      </button>
    );
  }
}
```

Or we can use an arrow function.

# Form Processing



# Form Processing

Let’s look at a component that just renders a form:

```
import React from 'react';
import logo from './logo.svg';
import './App.css';

function Form() {
    return (
        <form>
            <label>
                First Name:
                <input id='firstName' name='firstName' type='text' placeholder='First Name' />
            </label>
            <br />
            <label>
                Last Name:
                <input id='lastName' name='lastName' type='text' placeholder='Last Name' />
            </label>
            <br />
            <label>
                Username:
                <input id='username' name='username' type='text' placeholder='Username' />
            </label>
            <br />
            <input type='submit' value='Submit' />
        </form>
    );
}

export default Form;
```

# Form Processing

After we write our form, we need to tell the form which function to fire on the submit event

```
function Form() {
    return (
        <form onSubmit={formSubmit}>
            <label>
                First Name:
                <input id='firstName' name='firstName' type='text' placeholder='First Name' />
            </label>
            <br />
            <label>
                Last Name:
                <input id='lastName' name='lastName' type='text' placeholder='Last Name' />
            </label>
            <br />
            <label>
                Username:
                <input id='username' name='username' type='text' placeholder='Username' />
            </label>
            <br />
            <input type='submit' value='Submit' />
        </form>
    );
}

export default Form;
```

# Form Processing

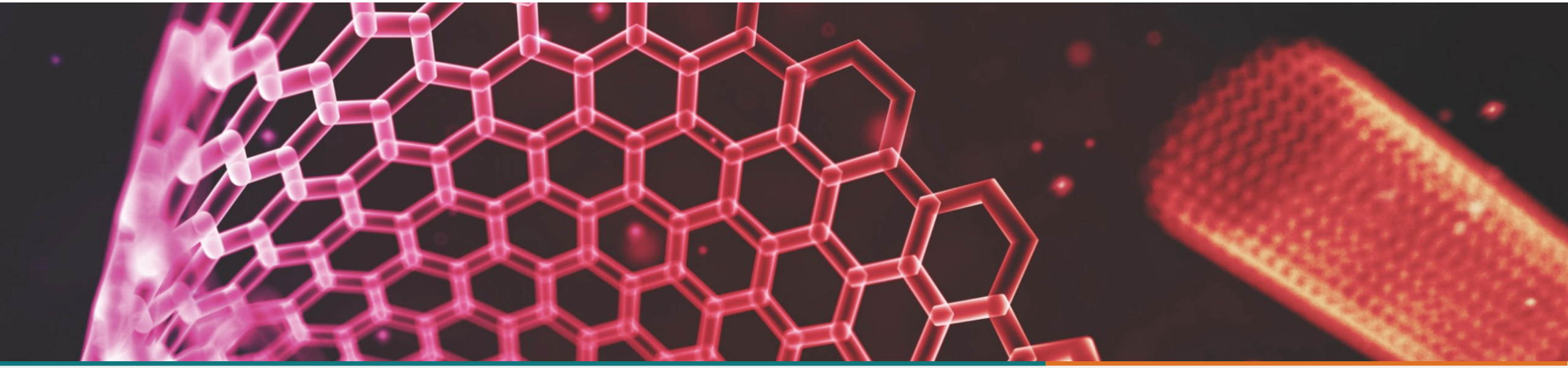
Then we write our submit function

```
function Form() {
    const formSubmit = (e) => {
        //disable form's default behavior
        e.preventDefault();
        //get references to form fields.
        let firstName = document.getElementById('firstName').value;
        let lastName = document.getElementById('lastName').value;
        let username = document.getElementById('username').value;

        //provide inout checking/validation
        //then perhaps post form data to an API or your express server end point

        let user = {
            firstName,
            lastName,
            username
        };
        alert(JSON.stringify(user));
    };
    return (
        <form onSubmit={formSubmit}>
```

# Fetching Data Using Class-Based Components



# Fetching Data Using Classes and Lifecycle Methods

Today, we are going to be looking at an application that pulls data from the **TV Maze API.** We will see the same application, first using class components and then using function components with hooks/.

Let’s take the following function that gets data from the TV Maze API and sets the data state to its results

```
async getShows() {
    try {
        const { data } = await axios.get('http://api.tvmaze.com/shows');
        this.setState({ data: data });
    } catch (e) {
        console.log(e);
    }
}
```

# Fetching Data Using Classes and Lifecycle Methods

We can call this function in the ***componentDidMount()*** lifecycle method

```
}
componentDidMount() {
    this.getShows();
}
```

This will call our ***getShows()*** function that fetches the data from the API

# Fetching Data Using Classes and Lifecycle Methods

Once we have the data and it's in the component state, we can then render the data using the map function to create list items in our UL:

```
render() {
    return (
        <div className='App-body'>
            <ul className='list-unstyled'>
                {this.state.data &&
                    this.state.data.map((show) => (
                        <li key={show.id}>
                            <Link to={`/shows/${show.id}`}>
                                <img alt='Show' src={show.image.medium} />; <br />
                                {show.name}
                            </Link>
                        </li>
                    ))}
            </ul>
        </div>
    );
}
```

# Fetching Data Using Classes and Lifecycle Methods

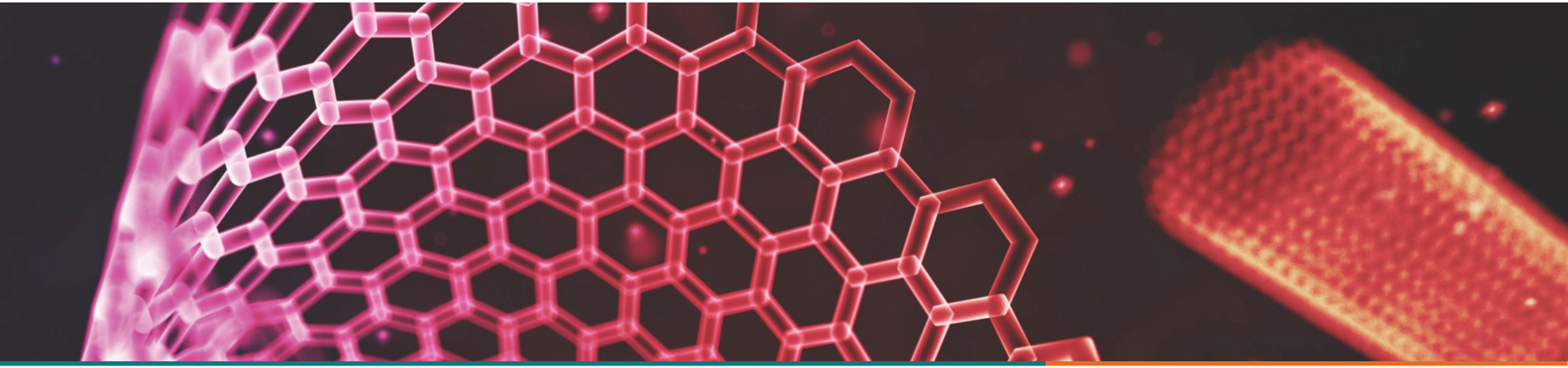
In our TV Maze example, we will have the following components:

- App (main container)
- ShowList (Component that displays the list of shows)
- Show (Component that shows an individual show's details)
- SearchShows (component that is nothing more than an input field to search the shows and passes that value to the ShowList component)

We will also see using React Router and URL parameters.

The application, when loaded and the /shows URL is navigated to, will display all the shows from the TV Maze API. When the user starts typing a search term in the search input, the onChange event will fire, setting the searchTerm state value to the value the user entered into the input, then query the API for that term. When The user clicks on a show, it will load the Show component and fetch the data for that show based on the URL parameter which is the show's ID.

# React Hooks



# What Are React Hooks?


React hooks are nothing more than functions that “hook” into React’s state and lifecycle features from a function component.

- Hooks allow us to easily manipulate the state of our functional component without needing to convert them into class components.

So no longer do we need to use class-based components if we want our component to deal with state. Although, it is still valuable to know class-based components as a lot of existing applications currently in use still utilize them.

**React Hooks do not work in class-based components, they can only be used in function components.**

# Rules of Hooks

- Don’t call Hooks inside loops, conditions, or nested functions—*Only call Hooks **at the top level**.*
- Don’t call Hooks from regular JavaScript functions—Only call Hooks **from React function components**.

# Some Basic React Hooks

Today we will go over two of the React Hooks which are the most useful and used (We will go over more in the next lecture):

- **useState**
- **useEffect**

# useState

Let’s recap how we set initial state in a class-based component:

```
    this.state = {
        data: undefined,
        searchTerm: undefined,
        searchData: undefined
    };
}
```

And how we set state in a class-based component:

```
this.setState({ data: data });
```

# useState

Using the React Hook useState, we can more easily handle state.

First, we need to import the hook:

```
import React, { useState} from 'react';
```

Once we import the hook, then we can set the initial state of the component:

```
const ShowList = () => {
    const [ searchData, setSearchData ] = useState(undefined);
    const [ showsData, setShowsData ] = useState(undefined);
    const [ searchTerm, setSearchTerm ] = useState('');
```

For each piece of state that our component uses, we have a separate statement, no more having to have a state object.

# useState

```
const ShowList = () => {
    const [ searchData, setSearchData ] = useState(undefined);
    const [ showsData, setShowsData ] = useState(undefined);
    const [ searchTerm, setSearchTerm ] = useState('');
```

**useState** is actually an array, the first element is the value of the state, the second element is a function that sets the state, so we destructure those two elements out into useful names like searchData for the value, and setSearchData as the update function.

Let’s see an example without using destructuring and then using it.

```jsx
import React, { useState } from 'react';

const App = (props) => {
    const count = useState(0);
    const text = useState('');

    return (
        <div>
            <p>
                The current {text[0] || 'count'} is: {count[0]}
            </p>
            <button onClick={() => count[1](count[0] + 1)}>Increment</button>
            <button onClick={() => (count[0] <= 0 ? count[1](0) : count[1](count[0] - 1))}>Decrement</button>
            <button onClick={() => count[1](0)}>Reset</button>
            <br />
            <br />
            <label>
                Enter Text:
                <input value={text[0]} onChange={(e) => text[1](e.target.value)} />
            </label>
        </div>
    );
};

export default App;
```

```jsx
import React, { useState } from 'react';

const App = (props) => {
    const [ count, setCount ] = useState(0);
    const [ text, setText ] = useState('');

    return (
        <div>
            <p>
                The current {text || 'count'} is: {count}
            </p>
            <button onClick={() => setCount(count + 1)}>Increment</button>
            <button onClick={() => (count <= 0 ? setCount(0) : setCount(count - 1))}>Decrement</button>
            <button onClick={() => setCount(0)}>Reset</button>
            <br />
            <br />
            <label>
                Enter Text:
                <input value={text} onChange={(e) => setText(e.target.value)} />
            </label>
        </div>
    );
};

export default App;
```

# useEffect

If you’re familiar with React class lifecycle methods, you can think of useEffect Hook as **componentDidMount**, **componentDidUpdate**, and **componentWillUnmount** combined.

# useEffect

Using the React Hook useEffect, we can more have lifecycle method like behavior in function components.

First, we need to import the hook:

```
import React, { useState, useEffect } from 'react';
```

Once we import the hook, then we can call the useEffect method:

useEffect() takes a function as an input and returns nothing. The function it takes will be executed for you **after every** render cycle.

```
useEffect(() => {
  console.log('useEffect has been called');
  setUser({ firstName: 'Patrick', lastName: 'Hill', username: 'graffixnyc' });
});
```

# useEffect

The function inside of useEffect() gets executed unnecessarily (i.e. whenever the component re-renders)

We have an infinite loop because setUser() causes the function to re-render

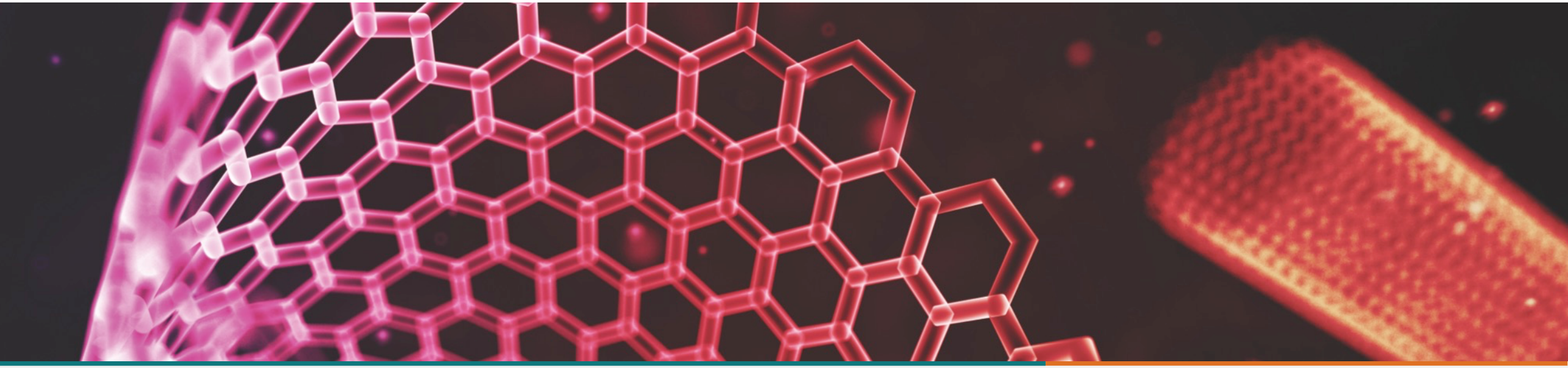
```
useEffect(() => {
  console.log('useEffect has been called');
  setUser({ firstName: 'Patrick', lastName: 'Hill', username: 'graffixnyc' });
});
```

useEffect() takes a second argument which allows us to control when the function that was passed in as the first argument will be executed.

```
useEffect(() => {
  console.log('useEffect has been called');
  setUser({ firstName: 'Patrick', lastName: 'Hill', username: 'graffixnyc' });
}, []);
```

Here, we pass an empty array ([]) as the second argument.
This leads React to only execute the function passed as the first argument when the component is rendered for the first time. Effectively, it now behaves like componentDidMount. We will see a case where the array is not empty in our code demo!

# Fetching Data Using Function Components, useState and useEffect



# Fetching Data

So we have seen how we can use useEffect to set state of a component when the component loads, let's see how we can use it to get data from our TV Maze API.

In our useEffect, we can make an axios call using promises

We then map our shows state and list each show in its own list item within a UL in our return statement

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import './App.css';

function App() {
  const [ loading, setLoading ] = useState(true);
  const [ shows, setShowData ] = useState(undefined);

  useEffect(() => {
    console.log('useEffect has been called');
    axios.get('http://api.tvmaze.com/shows').then(({ data }) => {
      setShowData(data);
      setLoading(false);
    });
  }, []);
  if (loading) {
    return <div>Loading...</div>;
  } else {
    return (
      <div className='App'>
        <ul>{shows && shows.map((show) => <li key={show.id}>{show.name}</li>)};</ul>
      </div>
    );
  }
}

export default App;
```

# Fetching Data

We can use promises in useEffect as we saw, but we can also define and call an async function within useEffect that will query our API end point!

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import './App.css';

function App() {
  const [ loading, setLoading ] = useState(true);
  const [ shows, setShowData ] = useState(undefined);

  useEffect(() => {
    console.log('useEffect has been called');

    async function fetchData() {
      try {
        const { data } = await axios.get('http://api.tvmaze.com/shows');
        setShowData(data);
        setLoading(false);
      } catch (e) {
        console.log(e);
      }
    }

    fetchData();
  }, []);
  if (loading) {
    return <div>Loading...</div>;
  } else {
    return (
      <div className='App'>
        <ul>{shows && shows.map((show) => <li key={show.id}>{show.name}</li>)};</ul>
      </div>
    );
  }
}

export default App;
```

# useEffect

Now we will take a look at our TV Maze API example, using function components, useState and useEffects!

# Next Lecture

In our next lecture, we will continue with our Firebase authentication app and learn about a few other React Hooks!

# Questions?