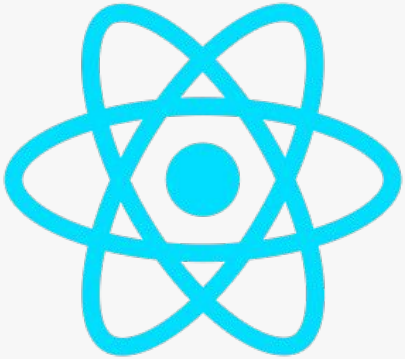
Newton School of Technology

The Ultimate React Course



@newtonschool

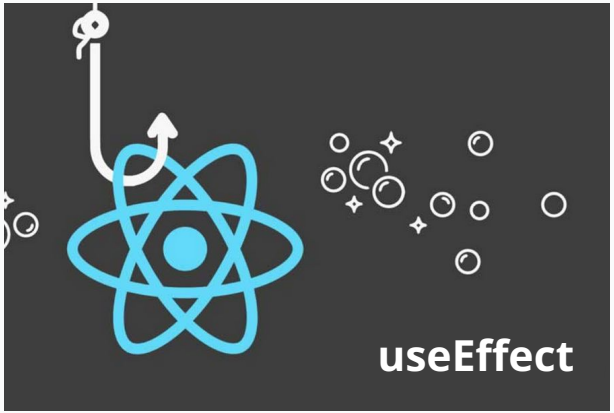
Lecture 17: API Integration

-Narendra Kumar

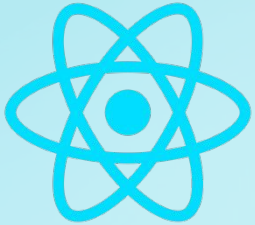
JS

Table of Contents

- Quick Recap
- Introduction to useEffect
- Side Effects in React (data fetching)
- Fetching data with fetch
- Lifting State Up
- Managing API using useState and useEffect
- Error Handling and Loading States
- Example: Displaying Public Data from a Public API



Newton School of Technology



Quick Recap

A brief Overview

Recap: Exporting Files

Newton School of Technology

To use components and in this case Header.js, we must export them from their respective files and import them where needed.

```
1  const Header = () => {
2      return <h1>Welcome to My React App</h1>;
3  };
4
5  export default Header;
```

Exporting Header Component

```
export default Footer;
```

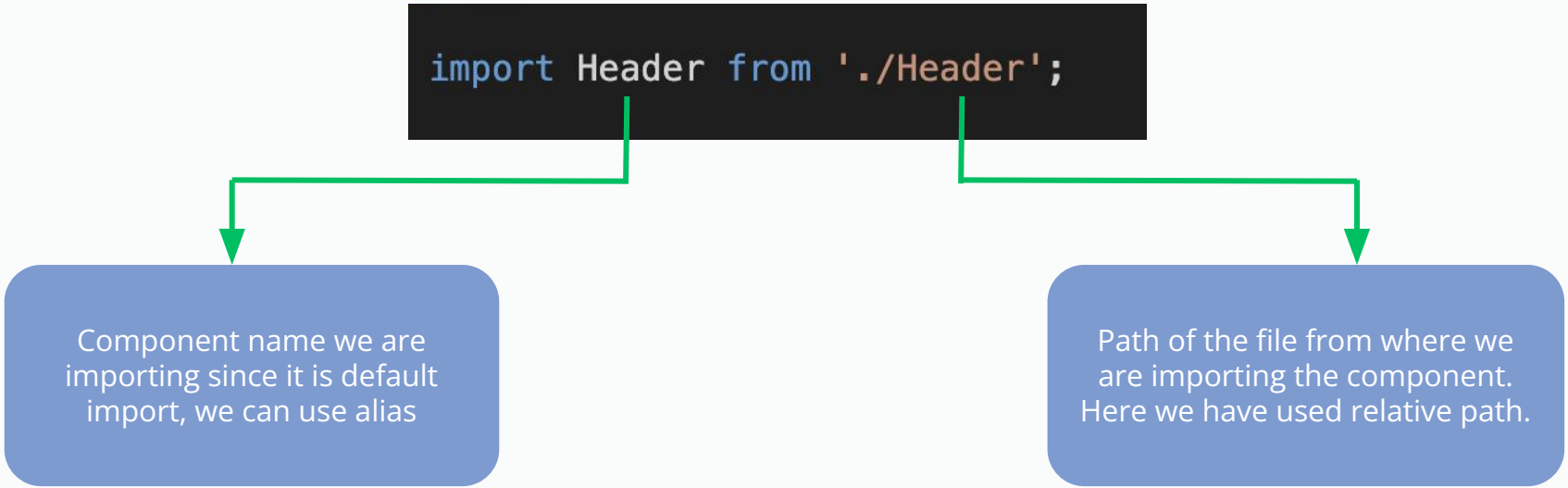
Tells we are exporting the component

And the export type is default export

We are exporting Footer component

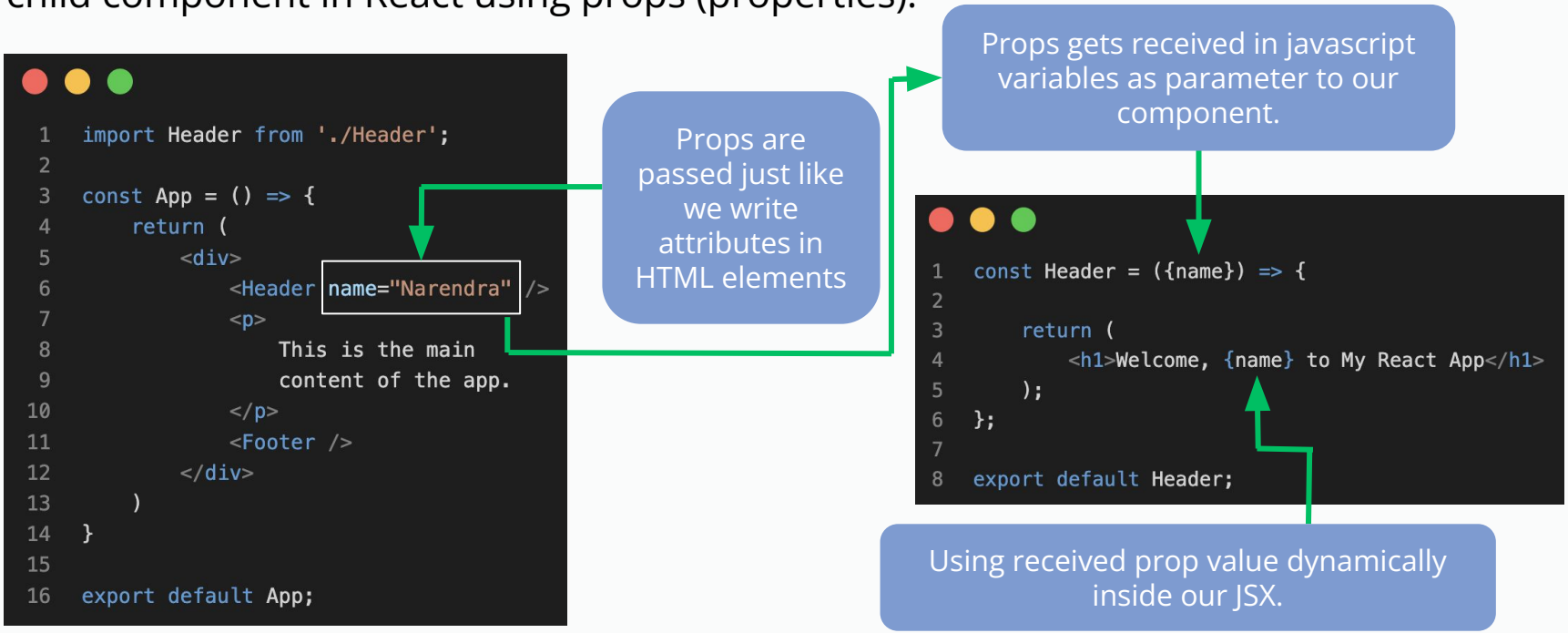
Recap: Import Files

Let’s dissect each part of our import statement:-



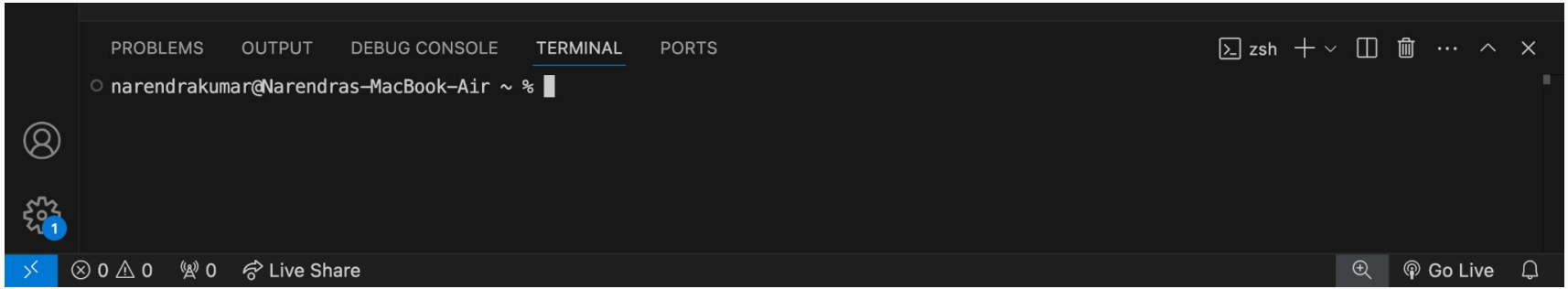
Recap: Passing props from parent to child

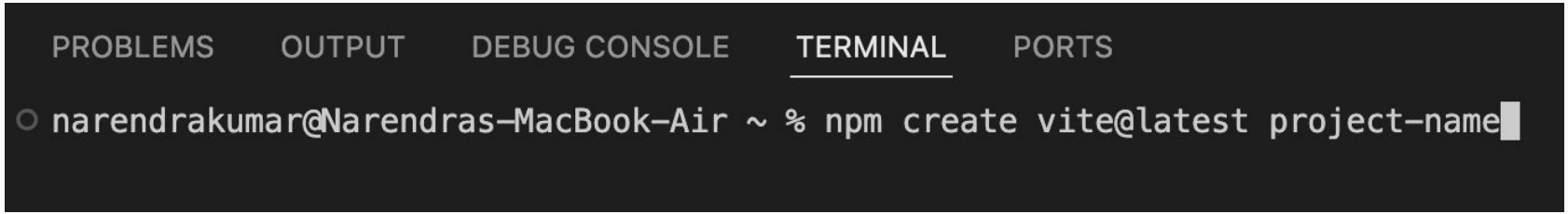
Prop passing from parent to child means sending data from a parent component to a child component in React using props (properties).



Recap: Creating a React App

Terminal would look like this:-



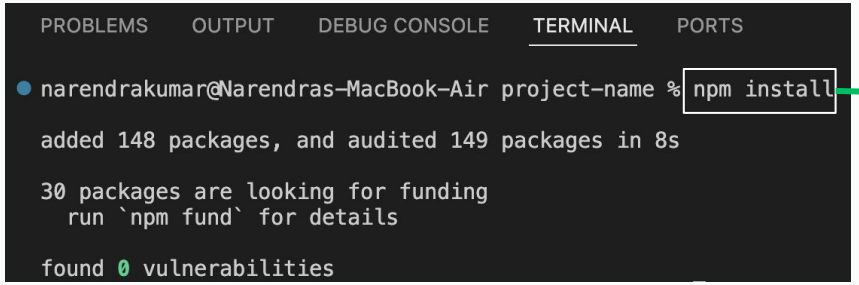


Run create vite@latest command with project directory name next to it

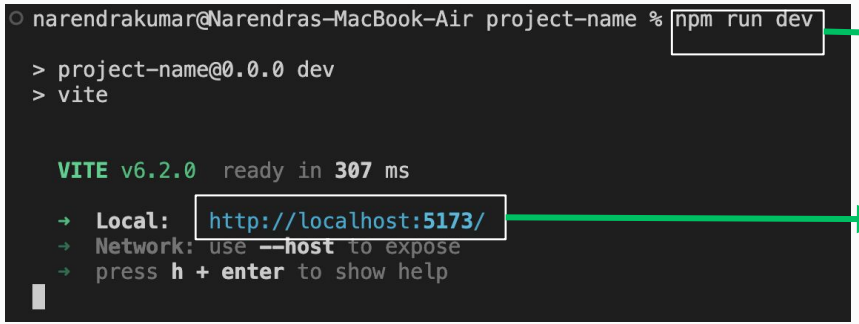
Recap: Installing and Launching App

Newton School of Technology

And then you need to install all the packages by using “npm install” and run the app using “npm run dev”.



Run “npm install” to install all the project dependencies packages



Run “npm run dev” to run your react application

Type this url in the address bar of your browser to preview your Project.

Newton School of Technology

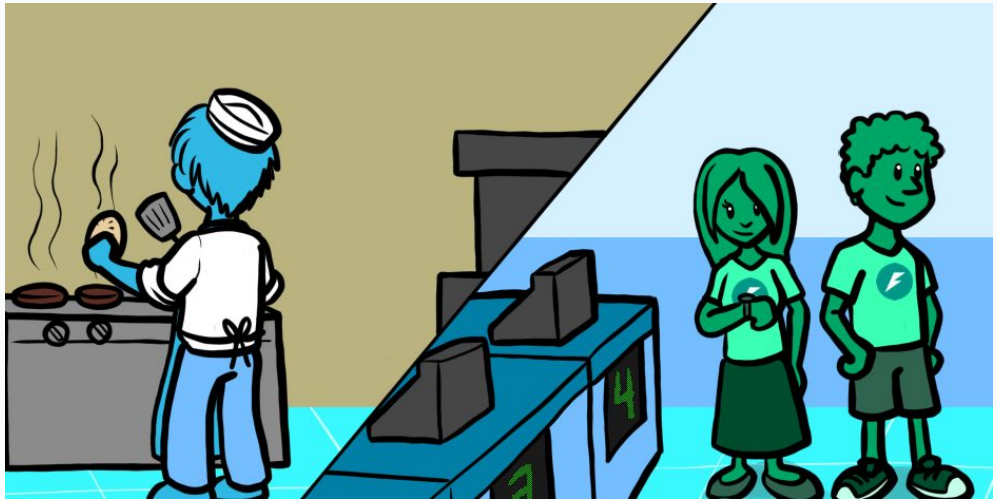
Introduction to useEffect()

Handling Side-Effects

Remember async operations in JS?

Newton School of Technology

We used callbacks, promises, etc. But why? What was the issue with async operations?



Async operations do not execute immediately, so waiting for them blocks the execution of other tasks.

Newton School of Technology

Use used promises to handle async ops

We used promises to handle async operations in JS. Here is an example:

```
function orderFood(order) {
    return new Promise((resolve) => {
        console.log(`Order placed: ${order}`);
        setTimeout(() => {
            resolve(`Order ready: ${order}`);
        }, 3000); // Simulating food preparation time
    });
}

console.log("Customer enters restaurant");

orderFood("Pizza").then((message) => {
    console.log(message);
});

console.log("Other customers continue ordering...");
```

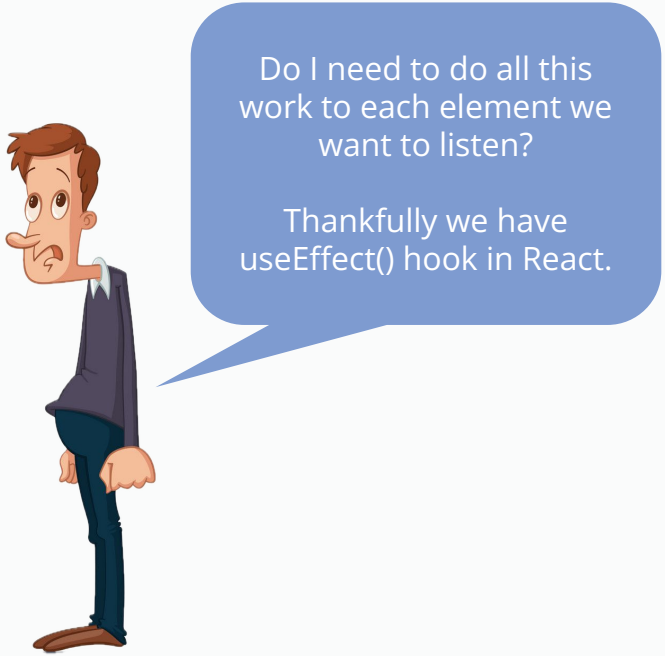
And then need to pass this orderFood() function to the event listener attached to an element.

Use used promises to handle async ops

Newton School of Technology

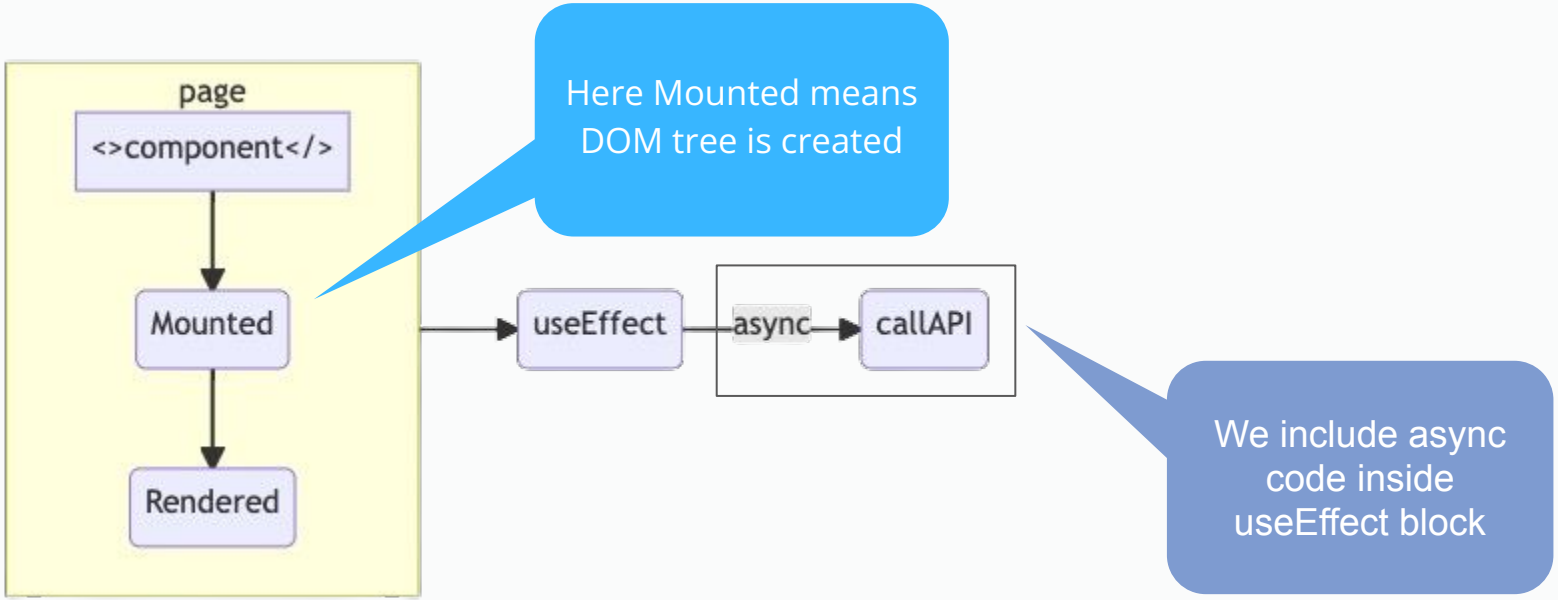
We used promises to handle async operations in JS. Here is an example:-

```
// Mimicking useEffect with an event listener
document.getElementById("orderBtn").addEventListener("click", () => {
    orderFood("Pizza").then((message) => {
        console.log(message);
    });
});

console.log("Other customers continue ordering...");


<html>
    ....
    <body>
        <button id="orderBtn">
            Order Pizza
        </button>
    </body>
</html>
```



Don’t worry we have useEffect() in React

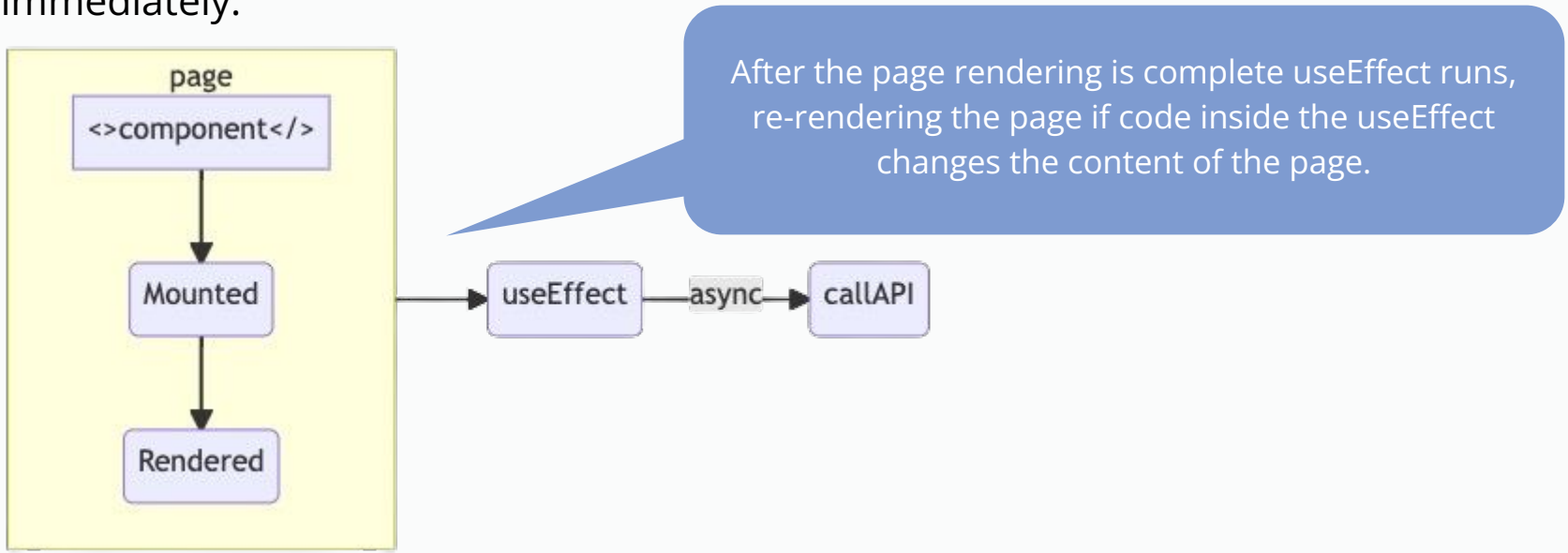
useEffect in React handles side effects like data fetching, subscriptions, or DOM updates after render.



useEffect() simplifies the page updates

Newton School of Technology

Just like promises facilitates the async operations to happen in right order, useEffect makes sure that async code runs only at the end allowing other parts of the page to load immediately.



Let's Understand with a Simple Example

Suppose you are provided a url from where you need to fetch the data and display its title in your web page.

```javascript
document.addEventListener("DOMContentLoaded", () => {
    const heading = document.createElement("h1");
    heading.textContent = "Loading...";
    document.body.appendChild(heading);

    fetch("https://jsonplaceholder.typicode.com/todos/1")
        .then(response => response.json())
        .then(data => {
            heading.textContent = data.title;
            // Update content after fetching
        });
});
```

If we use javascript we need to add event listener and create, insert dom elements by ourself

```javascript
function App() {
    const [data, setData] = useState(null);

    useEffect(() => {
        fetch("https://jsonplaceholder.typicode.com/todos/1")
            .then((response) => response.json())
            .then((json) => setData(json));
    }, []); // Runs only after the initial render

    return <h1>{data ? data.title : "Loading..."}</h1>;
}

export default App;
```

Whereas React runs useEffect at the end and if useEffect code changes content of the page, it reloads the latest copy of the page.

Order of execution...

Let's understand the order of execution of useEffect() with respect to rest of the code.

```
1  function App() {
2      const [data, setData] = useState(null);
3
4      useEffect(() => {
5          fetch("https://jsonplaceholder.typicode.com/todos/1")
6              .then((response) => response.json())
7              .then((json) => setData(json));
8      }, []); // Runs only after the initial render
9
10     return <h1>{data ? data.title : "Loading..."}</h1>;
11 }
12
13 export default App;
```

1. State Variable is declared and assigned *null*

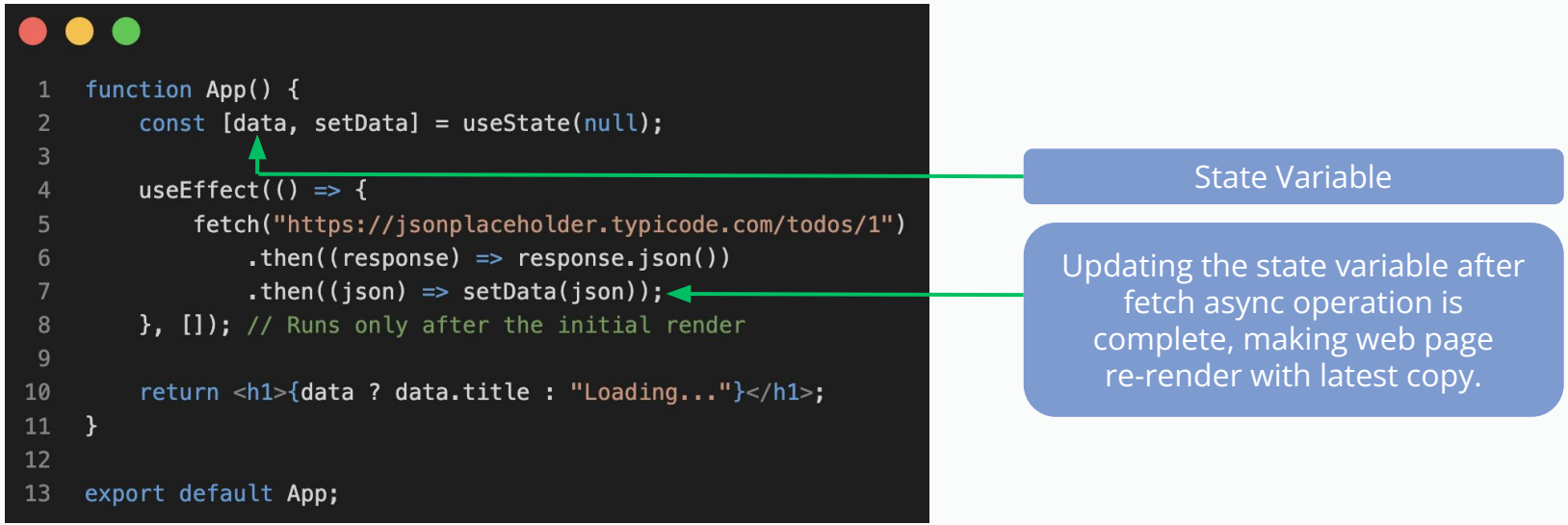
3. Code inside useEffect() block executes

2. Expression is evaluated and since data is *null*. Loading... is returned

How React knows content has changes?

Newton School of Technology

React tracks if its state variables are changed and when they are changed, a re-render of web page is triggered.

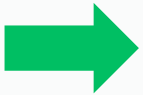


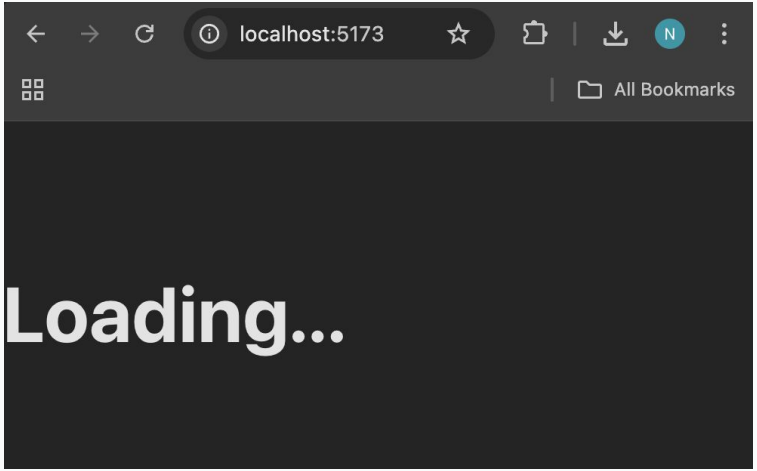
Initial Render: Data is not fetched yet

Newton School of Technology

If your internet connection is fast enough then initial render might not visible to you, in that case you need to slow down your internet connect from network tab of inspect tool.

```jsx
function App() {
    const [data, setData] = useState(null);

    useEffect(() => {
        fetch("https://jsonplaceholder.typicode.com/todos/1")
            .then((response) => response.json())
            .then((json) => setData(json));
    }, []); // Runs only after the initial render

    return <h1>{data ? data.title : "Loading..."}</h1>;
}

export default App;
```

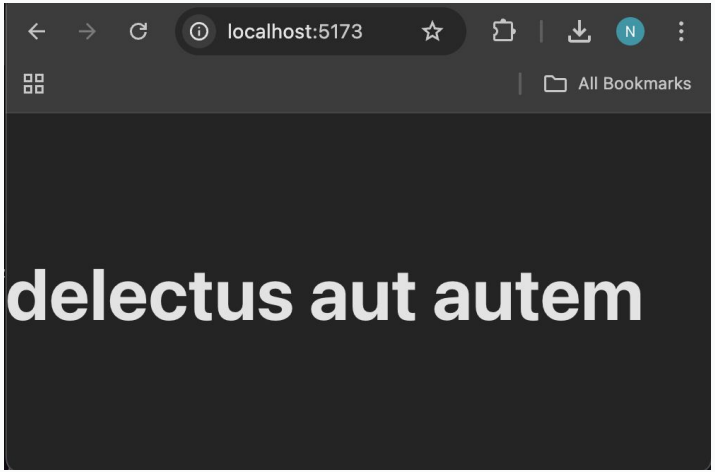




After fetch operation is completed

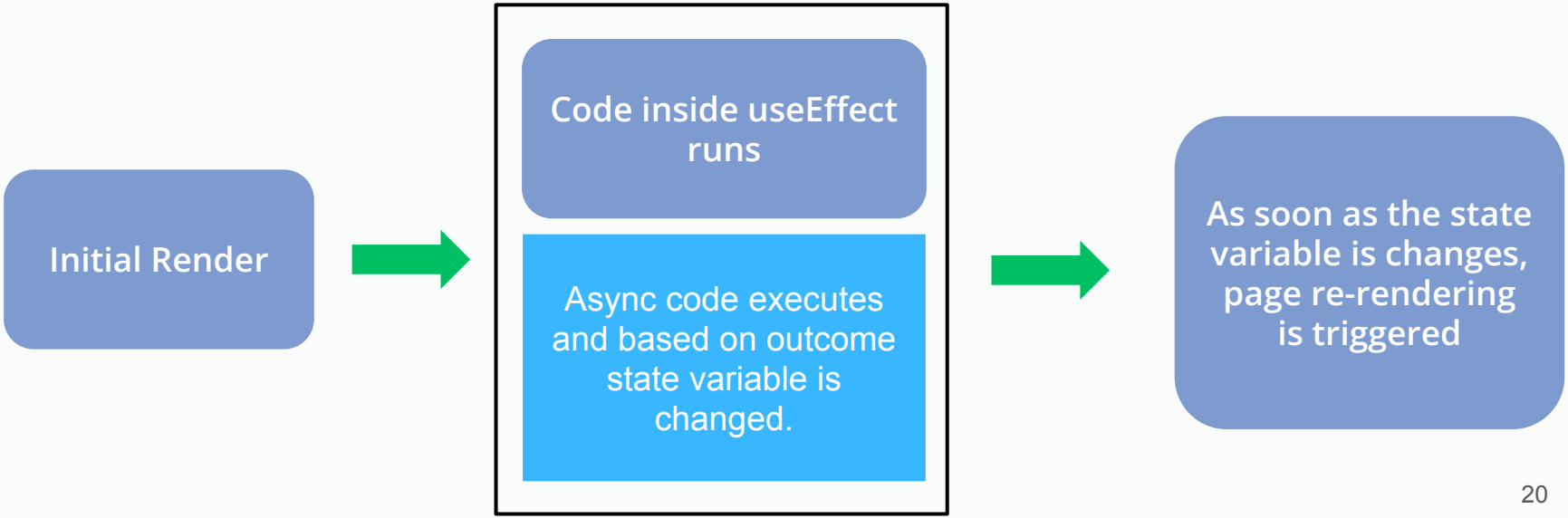
Have you observed that entire return block was executed once and as soon as state variable inside useEffect executed, JSX inside return statement got re-evaluated and re-rendered.

```jsx
function App() {
    const [data, setData] = useState(null);

    useEffect(() => {
        fetch("https://jsonplaceholder.typicode.com/todos/1")
            .then((response) => response.json())
            .then((json) => setData(json));
    }, []); // Runs only after the initial render

    return <h1>{data ? data.title : "Loading..."}</h1>;
}

export default App;
```



Key Takeaway

The only major work of useEffect is to execute its code after the initial render. We execute all async operations inside it, and make state changes if needed after async operations are complete, triggering the page re-render again.



Newton School of Technology

Error Handling and Loading States

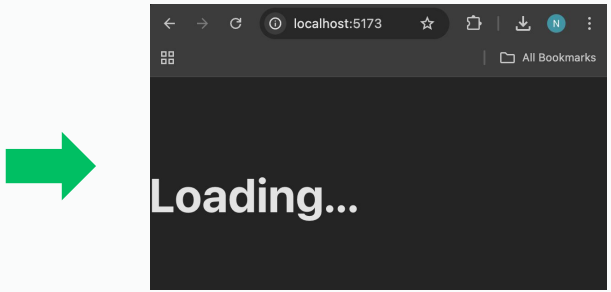
Managing failure states

What if fetch api fails?

Newton School of Technology

In case data fetching fails, then block is never executed and data state variable is never set. Would that mean, our app is always in Loading state? Isn't it inappropriate...

```jsx
function App() {
    const [data, setData] = useState(null);

    useEffect(() => {
        fetch("https://jsonplaceholder.typicode.com/todos/1")
            .then((response) => response.json())
            .then((json) => setData(json));
    }, []); // Runs only after the initial render

    return <h1>{data ? data.title : "Loading..."}</h1>;
}

export default App;
```



Always shows Loading state, but we should be showing that data fetch failed.

Using Error State to Manage Errors

Newton School of Technology

We need to maintain an error state to handle failures. Check response.ok to detect API failure. If false, throw an error, which is caught in catch to update the error state.

```jsx
import { useEffect, useState } from "react";

function App() {
    const [data, setData] = useState(null);
    const [error, setError] = useState(null);

    useEffect(() => {
        fetch("https://jsonplaceholder.typicode.com/todos/1")
            .then((response) => {
                if (!response.ok) {
                    throw new Error(`HTTP error! Status: ${response.status}`);
                }
                return response.json();
            })
            .then((json) => setData(json))
            .catch((err) => setError(err.message));
            // Catch and store error
    }, []); // Runs only after the initial render

    if (error) return <h1>Error: {error}</h1>;
    return <h1>{data ? data.title : "Loading..."}</h1>;
}

export default App;
```

If an error occurs, it is thrown and caught in catch, where the error state is updated with the message.

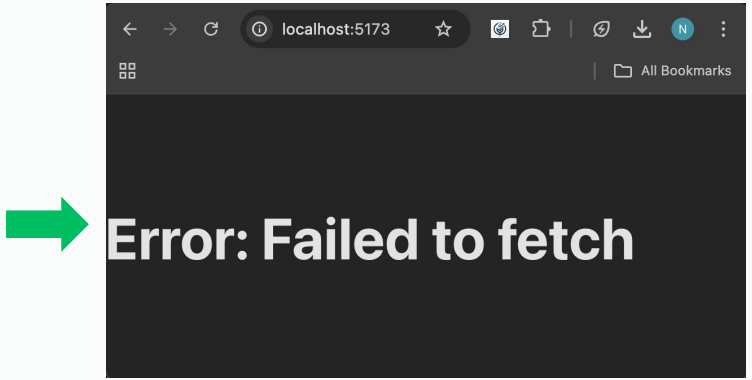
We check for an error first and display the error message if present; otherwise, we render the data.

Let’s introduce a typo to check error message

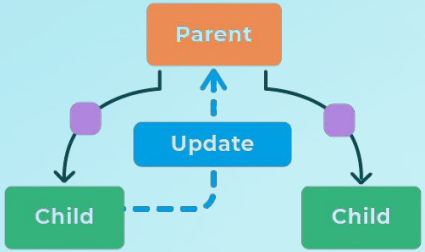
Here we have intentionally introduced a typo to check if the error message is rendered in the screen.

```
import { useEffect, useState } from "react";

function App() {
    const [data, setData] = useState(null);
    const [error, setError] = useState(null);

    useEffect(() => {
        fetch("https://jsonplaceholder.typicode.com/todo/1")
            .then((response) => {
                if (!response.ok) {
                    throw new Error(`HTTP error! Status: ${response.status}`);
                }
                return response.json();
            })
            .then((json) => setData(json))
            .catch((err) => setError(err.message));
    }, []); // Runs only after the initial render

    if (error) return <h1>Error: {error}</h1>;
    return <h1>{data ? data.title : "Loading..."}</h1>;
}

export default App;
```

Typo “todo” instead of “todos”



Newton School of Technology



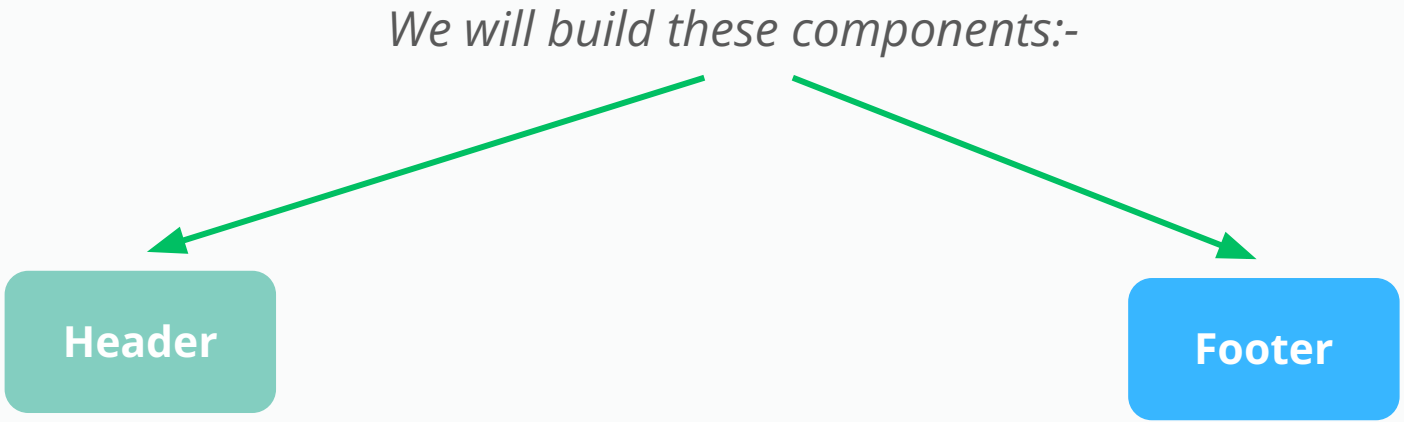
Lifting State Up

Child Talking to Parent

Recap: Create new components

Newton School of Technology

Just like App.js which is the root component, we can build other components. For now we will build functional components.



Creating Header.jsx

Newton School of Technology

Here we have created Header component in the separate named Header.jsx:-

```jsx
const Header = () => {
  return <h1>Welcome to My React App</h1>;
};
```

Header Component

Here we have created a simple header component with a simple message.

Pre-discussed:Including Header in App.js

We can simply import it in App.js file and use it here:-

```
// App.js
import React from "react";
import Header from "./Header";

const App = () => {
  return (
    <div>
      <Header />
      <p>This is the main content of the app.</p>
    </div>
  );
};

export default App;
```

While importing we have used relative path with respect to location of App.js

We can include our components using custom JSX tags, similar to HTML elements.

Recap: Creating Footer Component

Similarly we can create Footer component and use it in inside App like this:-

```
const Footer = () => {
    return (
        <p>
            @ 2025 My React App. All right reserved.
        </p>
    )
}
```

```
const App = () => {
    return (
        <div>
            <Header />
            <p>This is the main content of the app.</p>
            <Footer />
        </div>
    )
}

export default App;
```

Recap: Customized Message in Header

Newton School of Technology

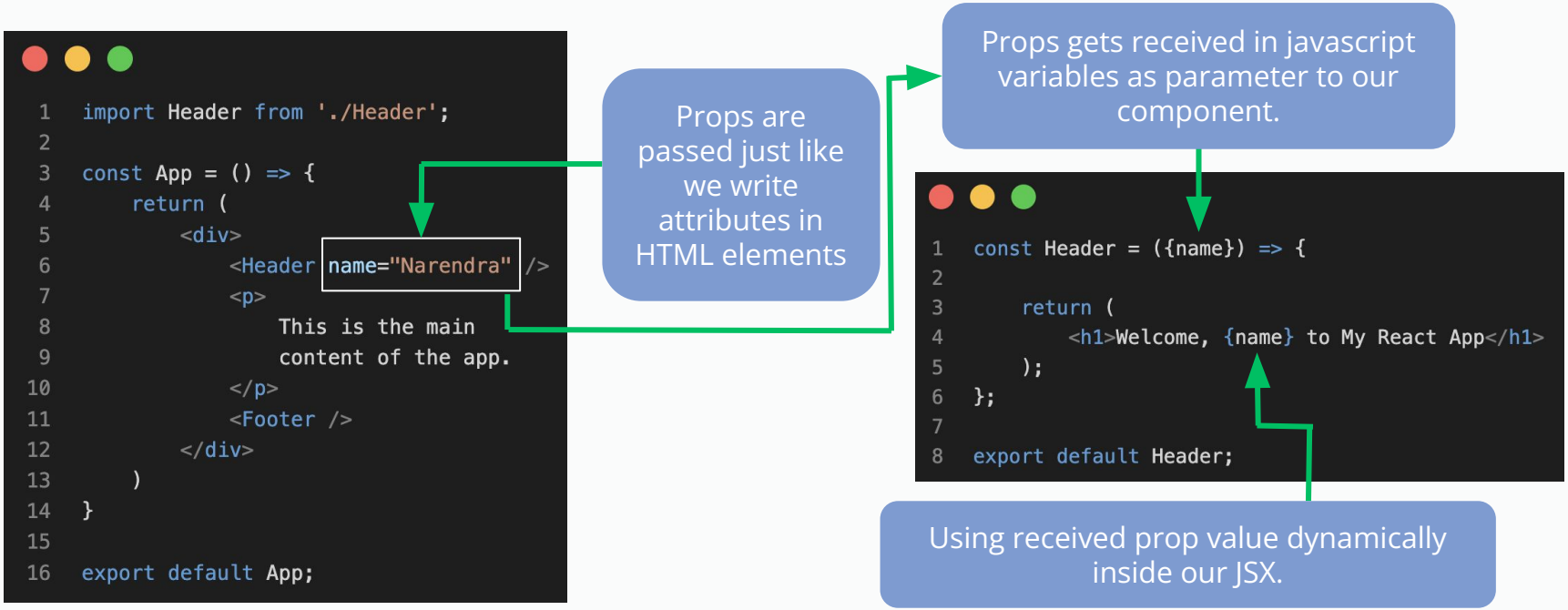
Often, we need to display names or similar data in the webpage header. We can store the value and pass it to the Header component.

```
1  const Header = () => {
2      const name = "Narendra";
3
4      return (
5          <h1>Welcome, {name} to My React App</h1>
6      );
7  };
8
9  export default Header;
```

This doesn't seems right, we should be taking data from Home Page i.e. App.jsx

Recap: Passing props from parent to child

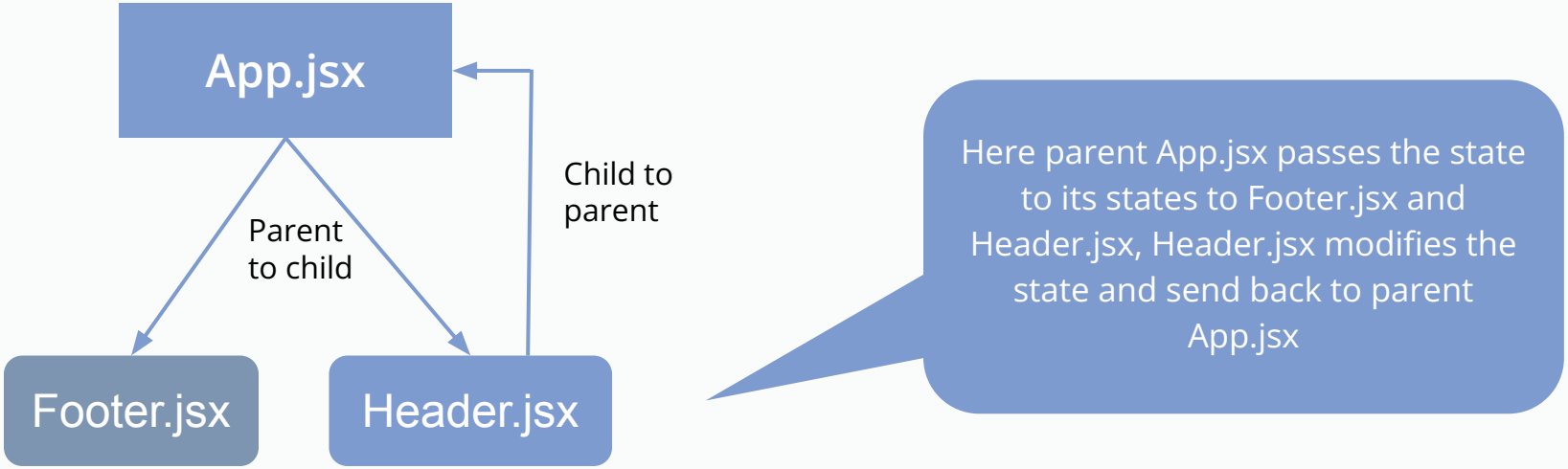
Instead we can use prop. Prop passing from parent to child means sending data from a parent component to a child component in React using props (properties).



What if we need to pass prop child to parent

Lifting State Up moves state from a child to its parent, allowing shared management.

The child updates the parent using callback functions passed as props.



Lifting State Up: How it happens

The child updates the parent using callback functions passed as props.

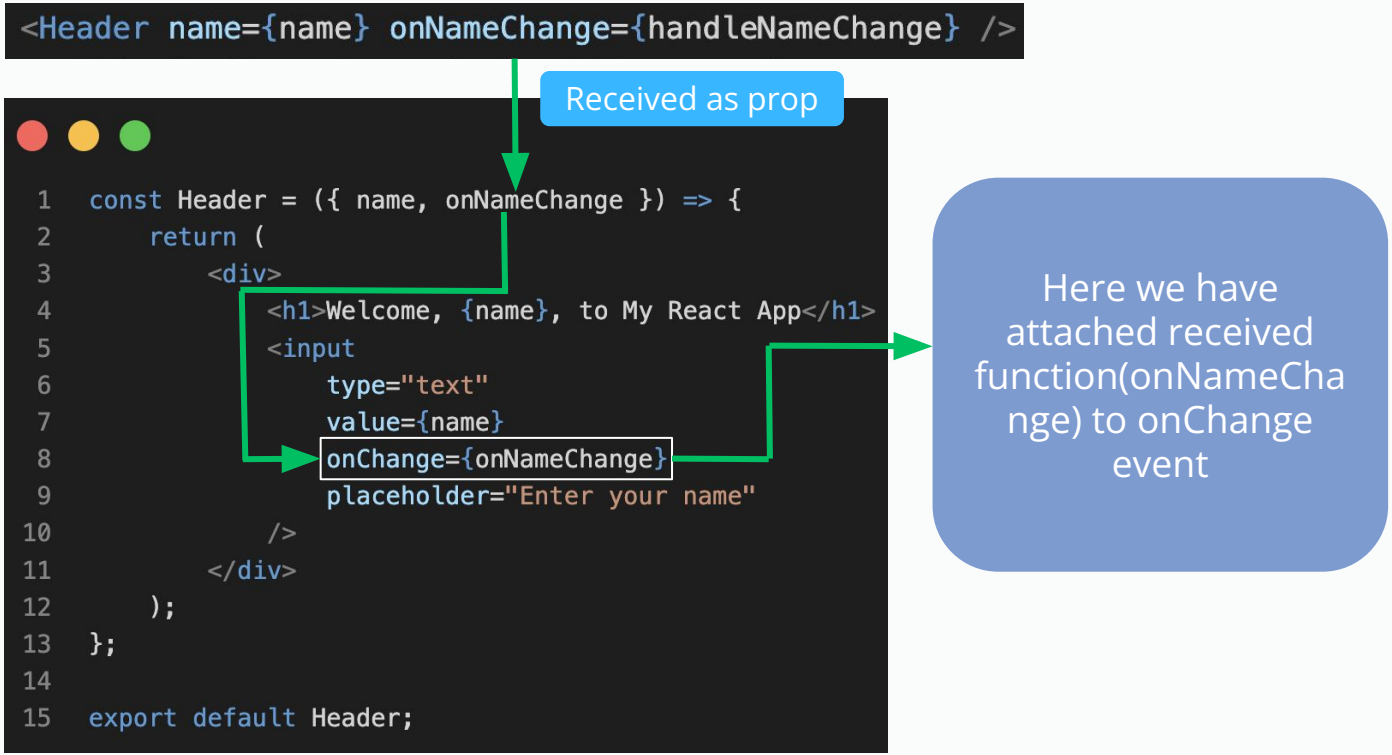
```jsx
import { useState } from 'react';
import Header from './Header';
import Footer from './Footer';

const App = () => {
    const [name, setName] = useState("Narendra");

    // Function to update the name (passed to child)
    const handleNameChange = (event) => {
        setName(event.target.value);
    };

    return (
        <div>
            <Header name={name} onNameChange={handleNameChange} />
            <p>This is the main content of the app.</p>
            <strong>Logged user: {name} </strong>
            <Footer />
        </div>
    );
};

export default App;
```

State update function passed as a value in onNameChange prop

Lifting State Up: How it happens

Newton School of Technology

Props passed are received in the child component, and then child can use it.



Lifting State Up: How it happens

Let’s understand the entire flow, what get passed to where:-

```jsx
const App = () => {
    const [name, setName] = useState("Narendra");

    // Function to update the name (passed to chil
    const handleNameChange = (event) => {
        setName(event.target.value);
    };

    return (
        <div>
            <Header name={name} onNameChange={handleNameChange} />
            <p>This is the main content of the app.</p>
            <strong>Logged user: {name} </strong>
            <Footer />
        </div>
    );
```

State changer function passed as prop

App.jsx

Prop got received

```jsx
const Header = ({ name, onNameChange }) => {
    return (
        <div>
            <h1>Welcome, {name}, to My React App</h1>
            <input
                type="text"
                value={name}
                onChange={onNameChange}
                placeholder="Enter your name"
            />
        </div>
    );
};
```

Prop function attached to onChange event

Header.jsx

Lifting State Up: How it happens

Here we have attached state updater function to an even in the child component.

```jsx
const handleNameChange = (event) => {
    setName(event.target.value);
};
```

App.jsx

handleNameChange() got attached to onChange event

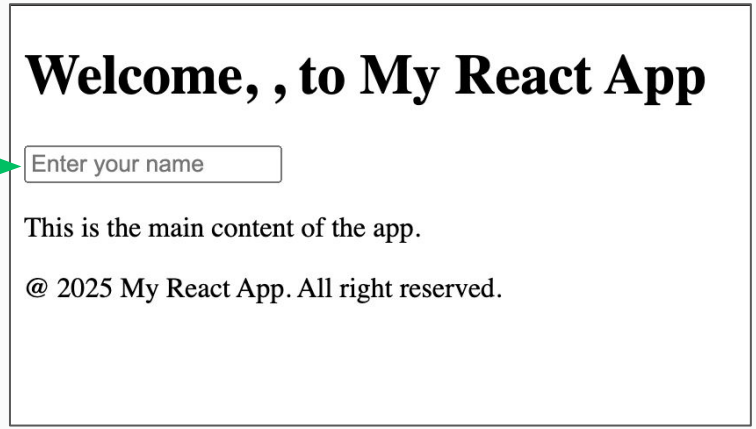
```jsx
const Header = ({ name, onNameChange }) => {
    return (
        <div>
            <h1>Welcome, {name}, to My React App</h1>
            <input
                type="text"
                value={name}
                onChange={onNameChange}
                placeholder="Enter your name"
            />
        </div>
    );
};
```

Header.jsx

Lifting State Up: How it happens

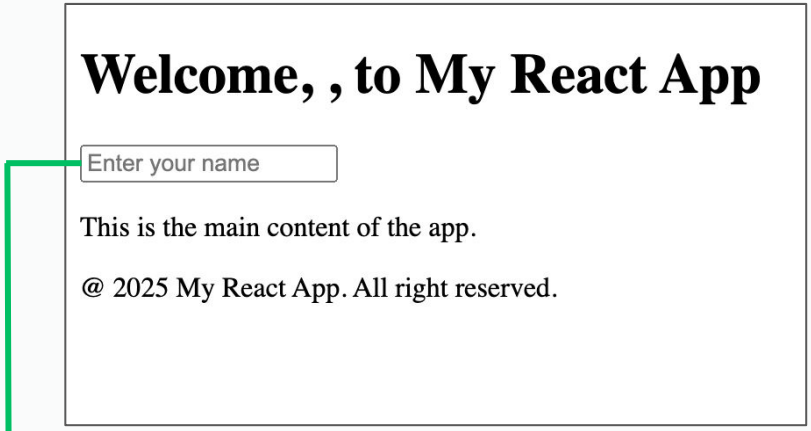
When user writes its name on input box, onChange event handler gets invoked and an event gets passed to its associated function i.e. onNameChange().

```
const Header = ({ name, onNameChange }) => {
    return (
        <div>
            <h1>Welcome, {name}, to My React App</h1>
            <input
                type="text"
                value={name}
                onChange={onNameChange}
                placeholder="Enter your name"
            />
        </div>
    );
};
```



Lifting State Up: How it happens

onNameChange() in turn corresponds to handleNameChange() function in the parent component.



When user writes in input box onNameChange() receives and event object

Event received in onNameChange()

```
<input
    type="text"
    value={name}
    onChange={onNameChange}
    placeholder="Enter your name"
/>
```

Header.jsx

Event gets passed to parent

```
const handleNameChange = (event) => {
    setName(event.target.value);
};

return (
    <div>
        <Header name={name} onNameChange={handleNameChange} />
```

App.jsx

Lifting State Up: How it happens

Hence we can update the state of parent function by passing state updater function in a prop.

```jsx
const App = () => {
    const [name, setName] = useState("Narendra");

    // Function to update the name (passed to child)
    const handleNameChange = (event) => {
        setName(event.target.value);
    };

    return (
        <div>
            <Header name={name} onNameChange={handleNameChange} />
            <p>This is the main content of the app.</p>
            <strong>Logged user: {name} </strong>
            <Footer />
        </div>
    );
};

export default App;
```

Updates the value in react state *“name”*

App.jsx

Workshop

In Class Questions

References

1. **MDN React Guide**: Comprehensive and beginner-friendly documentation for React https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/React_getting_started
2. **Websites and Tutorials**:
   a. 📌 w3schools: https://www.w3schools.com/REACT/DEFAULT.ASP
   b. 📌codecademy: https://www.codecademy.com/learn/react-101
3. **Other Useful Resources**
   a. 📌 React GitHub Repository: https://github.com/facebook/react
   b. 📌 React Patterns & Best Practices: https://reactpatterns.com/

Newton School of Technology

Thanks for watching!