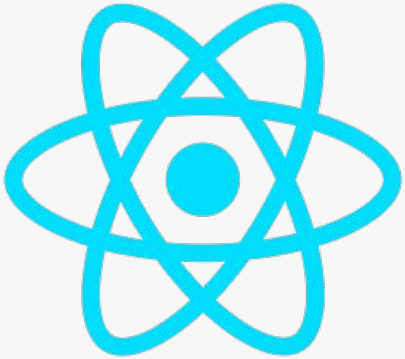
Newton School of Technology

The Ultimate React Course



@newtonschool

Lecture 18: Hooks and React Lifecycle

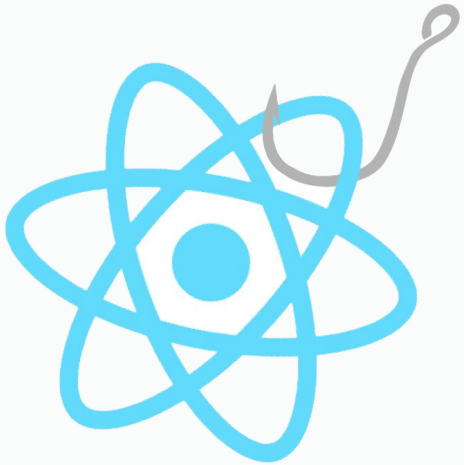
-Narendra Kumar

JS

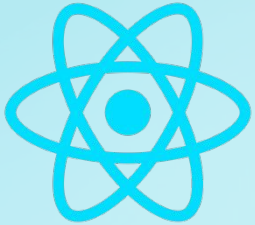
Table of Contents

Newton School of Technology

- Quick Recap
- Details about useEffect
- Component lifecycle
    - Mounting
    - updating
    - Destruction
- Cleanup functions in useEffect.
- Example: Timer or clock application.



Newton School of Technology

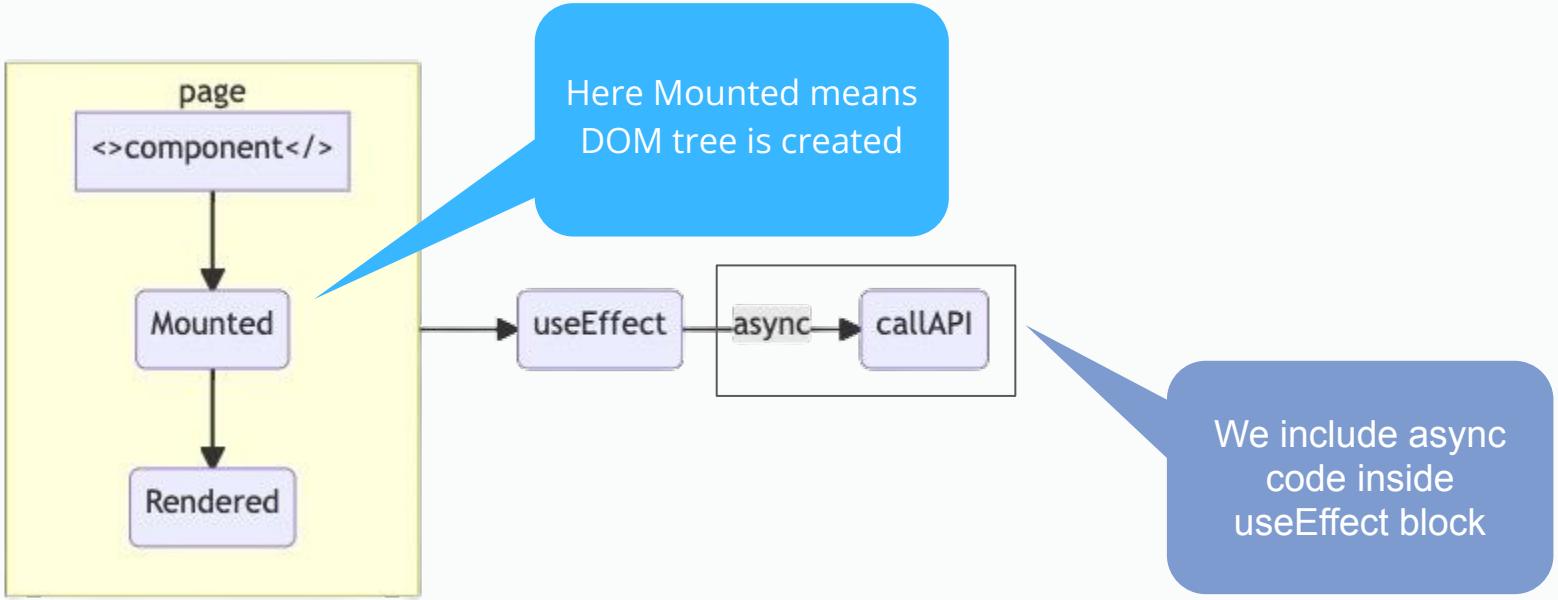


Quick Recap

A brief Overview

Recap: Purpose of useEffect()

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



Recap: Order of execution...

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

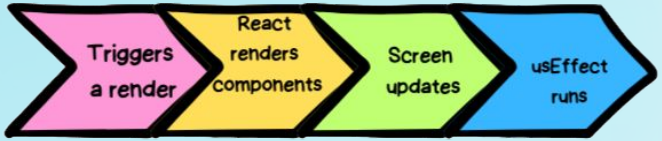
```
1  function App() {
2      const [data, setData] = useState(null);
3
4      useEffect(() => {
5          fetch("https://jsonplaceholder.typicode.com/todos/1")
6              .then((response) => response.json())
7              .then((json) => setData(json));
8      }, []); // Runs only after the initial 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

Newton School of Technology



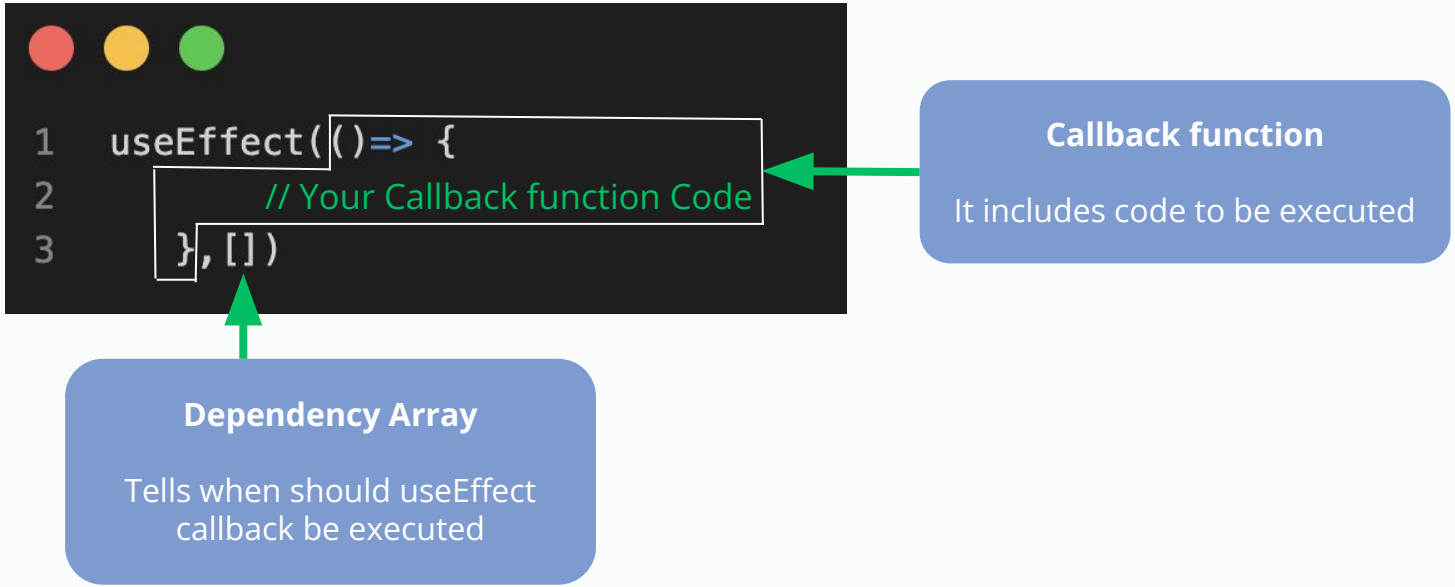
Understanding useEffect in Detail

Application updates simplified

useEffect() dependency Array

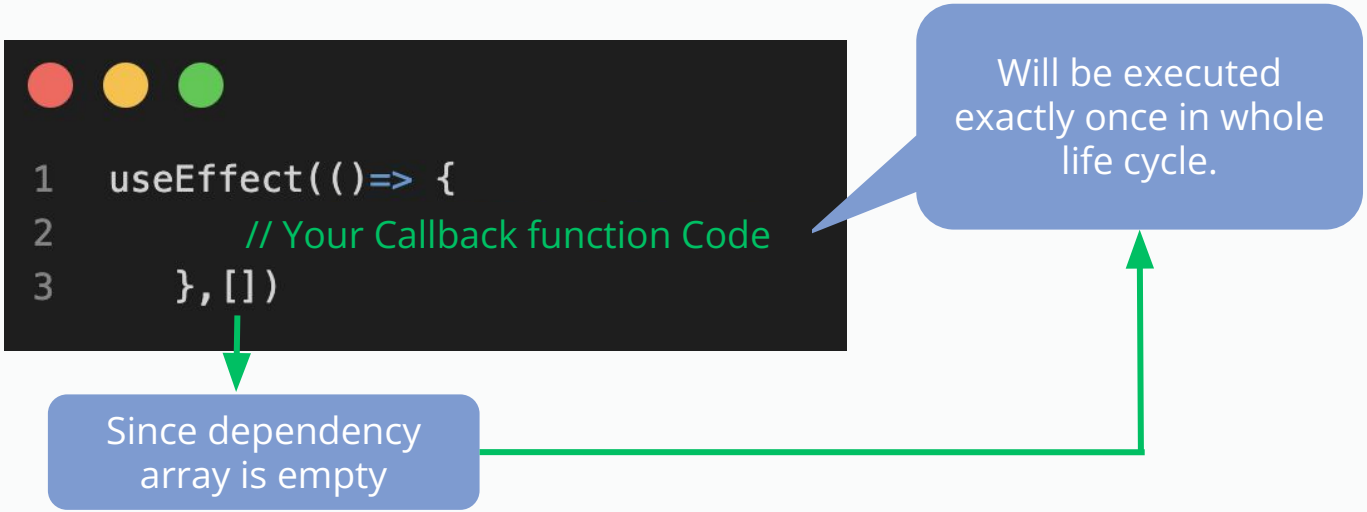
Newton School of Technology

useEffect() is a react hook which receives two arguments, the first is a callback function and second is the dependency array.



Dependency Array: Empty

Callback function will be executed just after initial render of the web page.



Dependency Array: Empty with Example

Newton School of Technology

Here we want to perform a fetch operation a fetch operation just after the page renders.

To achieve this we have kept dependency array empty.

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

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;
```

Since we are doing state update inside the useEffect on successful completion of fetch operation, page gets re-rendered once again.

useEffect() dependency Array

Newton School of Technology

useEffect() is a react hook which receives two arguments, the first is a callback function and second is the dependency array.

```
useEffect(
    () => {
    fetch("https://jsonplaceholder.typicode.com/todos/1")
        .then((response) => response.json())
        .then((json) => setData(json))
    }
    ,[]
);
```

If dependency array empty then useEffect is executed just once after the page loads.

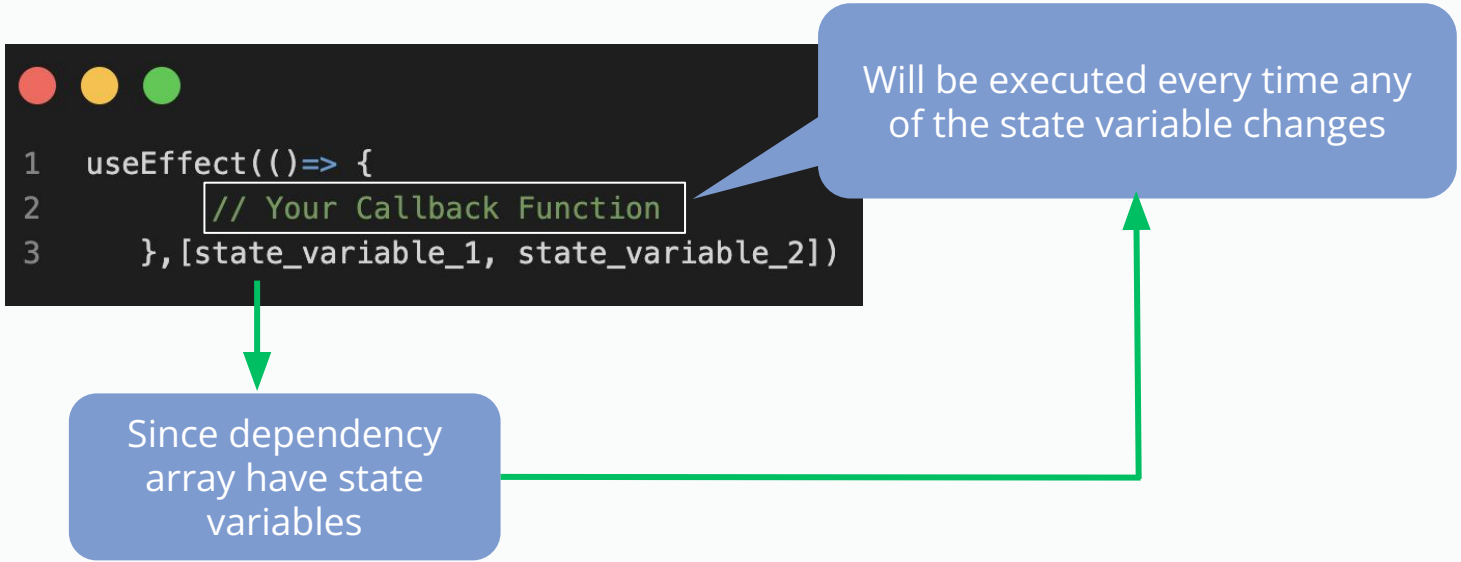
```
useEffect(() => {
    console.log("useEffect runs after render!");
}, [count]); // Runs after every `count` update
```

But if it is non-empty, then useEffect runs whenever the state variables in that dependency arrays gets updated.

Dependency Array: State Variables

Newton School of Technology

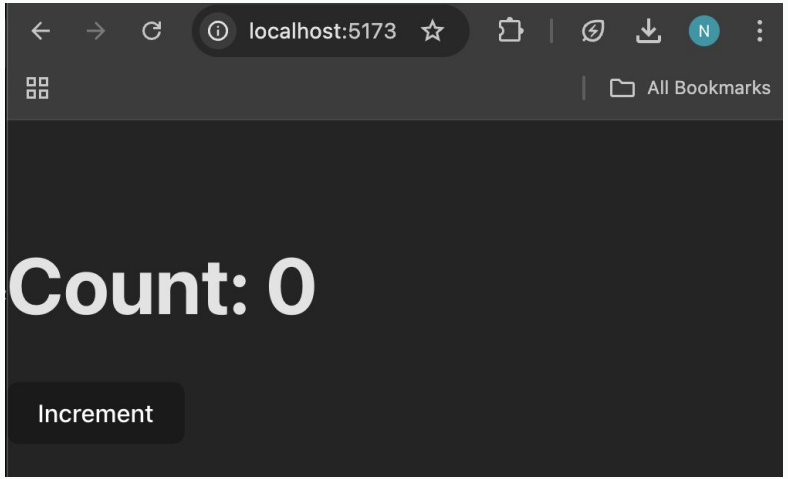
If dependency array contains state variables then callback function executes for each change in any of the state variables it contains.



Let’s understand it with an example

Here we have our old good counter app, but for debugging purpose we want to display the value of the counter. How to do it??

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

function App() {
    const [count, setCount] = useState(0);

    return (
        <div>
            <h1>Count: {count}</h1>
            <button onClick={() => setCount(count + 1)}>
                Increment
            </button>
        </div>
    );
}

export default App;
```



useEffect: state variable in dependency array

Let's write a useEffect() hook, with a callback function displaying the value of "count" along with dependency array having count as its value.

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

function App() {
    const [count, setCount] = useState(0);

    useEffect(() => {
        console.log("Count: ", count);
    }, [count]); // Runs after every `count` update

    return (
        <div>
            <h1>Count: {count}</h1>
            <button onClick={() => setCount(count + 1)}>
                Increment
            </button>
        </div>
    );
}

export default App;
```

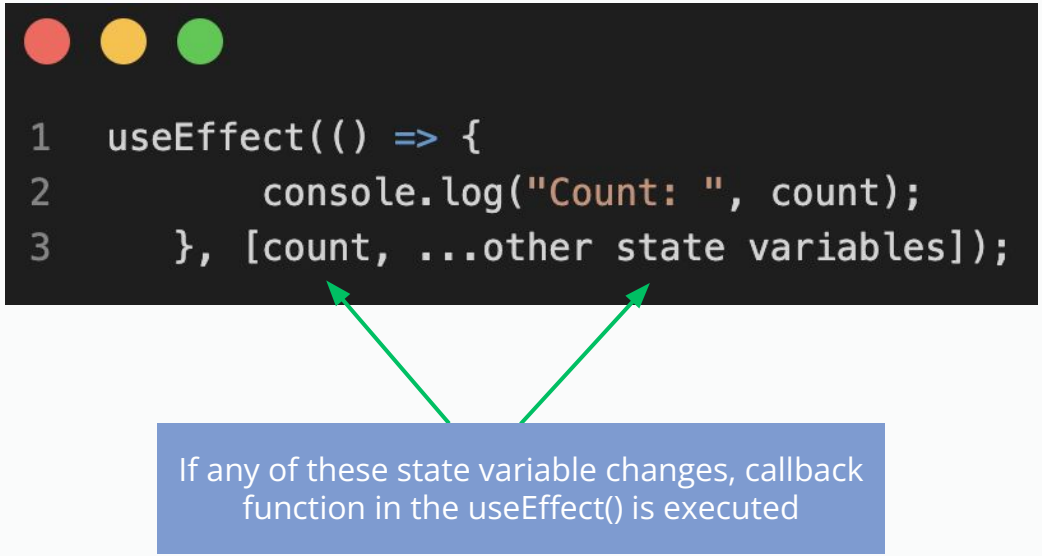
Whenever user clicks on Increment, count state increments by one

And whenever count state updates by one, code inside useEffect runs as count is provided as a value in the dependency array.

useEffect: state variable in dependency array

Newton School of Technology

Similarly we can add any number of state variables in the dependency array, and if any of these state value changes, useEffect() callback function is executed.



Newton School of Technology

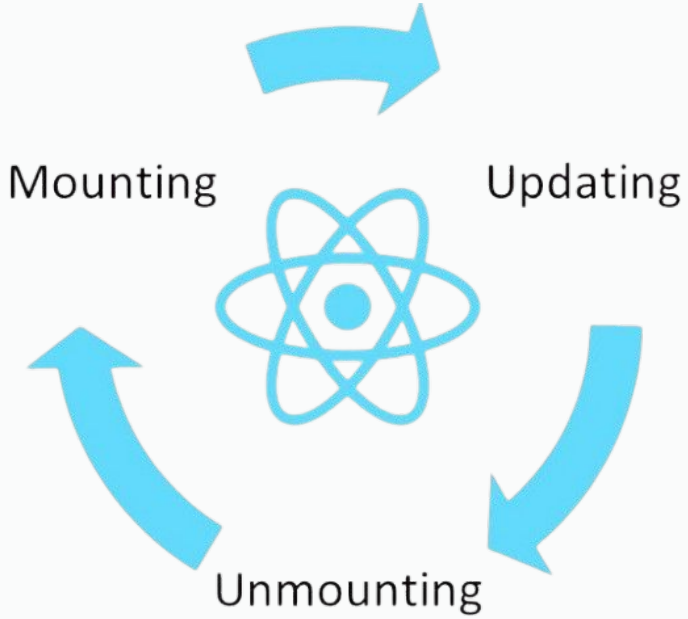
Functional Component Lifecycle

Mounting, updating and destruction

Lifecycle of Functional Components

Newton School of Technology

In class components, lifecycle methods like componentDidMount, componentDidUpdate etc control side effects. In functional components, useEffect replaces them.



Let's understand each of these phases in the upcoming slides

Mounting: useEffect with empty array

Newton School of Technology

Let’s say you want an operation like data fetch from the remote server for the first time website loads.

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

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;
```

Best for operations which happens once after the component loads/mounts like data fetch etc.

If we keep dependency array empty then useEffect callback function only executes once after initial render

Updating: useEffect with state variables

Newton School of Technology

Many a times we need useEffect to run again and again whenever a state variable changes, to make it happen *we can add state variables in the dependency array of the useEffect hook*.

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

function App() {
    const [count, setCount] = useState(0);

    useEffect(() => {
        console.log("Count: ", count);
    }, [count]); // Runs after every `count` update

    return (
        <div>
            <h1>Count: {count}</h1>
            <button onClick={() => setCount(count + 1)}>
                Increment
            </button>
        </div>
    );
}

export default App;
```

Here we were running useEffect code for each updation in the state variable.

And whenever count state updates by one, code inside useEffect runs as count is provided as a value in the dependency array.

Whenever user clicks on Increment, count state increments by one

Unmount: useEffect with return

When a component is no longer needed (e.g., navigating to another page), it should release resources through cleanup—this is called unmounting. We do it inside return.

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

function Timer() {
    useEffect(() => {
        const interval = setInterval(() => console.log("Timer running..."), 1000);
        return () => clearInterval(interval); // Cleanup on unmount
    }, []);

    return <h1>Timer Running...</h1>;
}

function App() {
    const [show, setShow] = useState(true);
    return (
        <div>
            <button onClick={() => setShow(!show)}>
                {show ? "Stop Timer" : "Start Timer"}
            </button>
            {show && <Timer />}
        </div>
    );
}

export default App;
```

setInterval() acquired when component mounted

setInterval() got released when component unmounted i.e. component got destroyed.

Unmount: useEffect with return

Newton School of Technology

We can acquire more than one resources after mounting the component and *release all of them at once inside the return block when component unmounts*.

```
1  useEffect(() => {
2
3      const interval = setInterval(() => {
4          console.log("Timer running...")
5      }, 1000);
6      const timer = setTimeout(()=> {
7          console.log("Timed out!");
8      }, 5000);
9
10     return () => {
11         clearInterval(interval);
12         clearTimeout(timer);
13     }
14 }, []);
```

Key Takeaway

We always need to run clean up functions if resources are acquired during mounting stage. If not then performance of our application is compromised.

There isn't much space; we need to clean up the area by leaving.

Newton School of Technology

Handling Error States

Catching Up the Errors

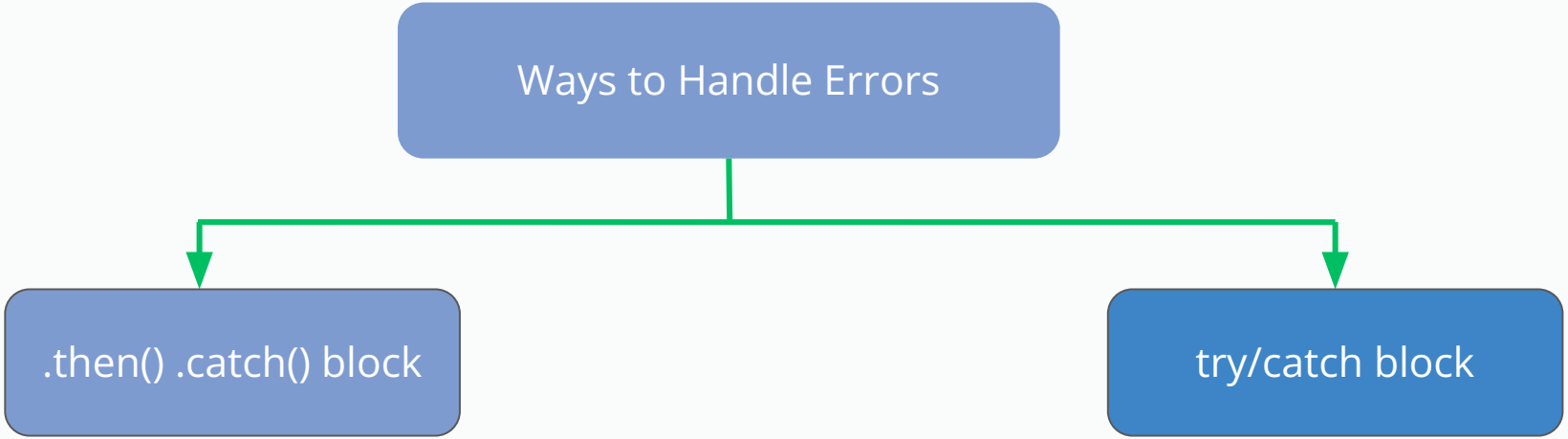
Error can happen due to various reasons

Many a time async operations can fail leading to erroneous state, we need to handle such scenario by adding proper error handling.

Newton School of Technology

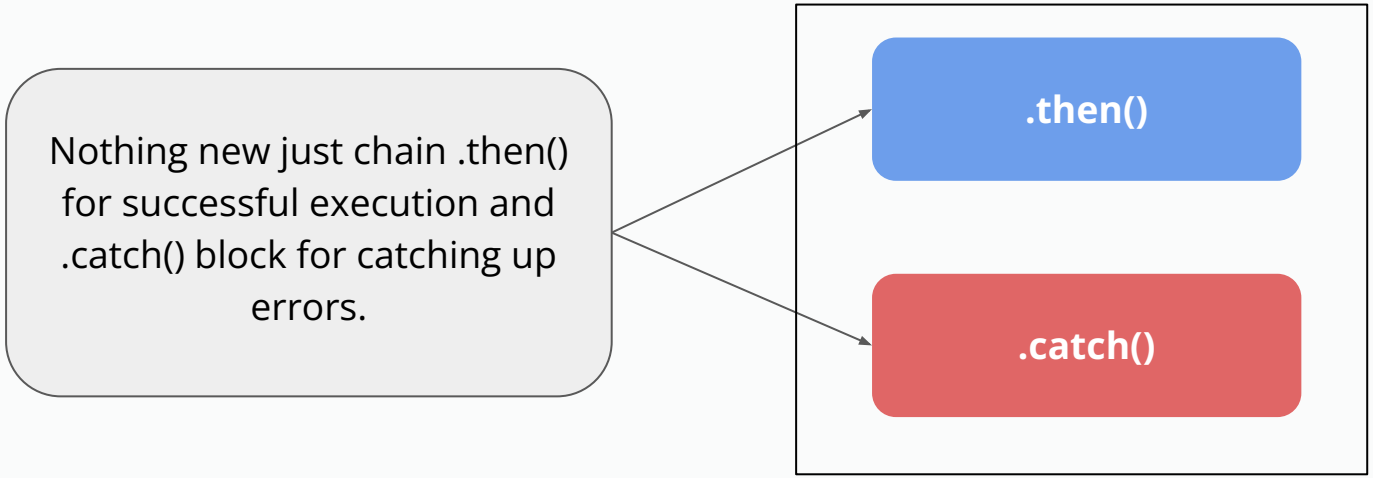
We need to handle those errors

There are different ways to handle those errors:-



then/catch to Handle the Errors

We can handle the errors using then/catch block in case of promises.



Newton School of Technology

Using Error State to Manage Errors: then/catch

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.

```
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.

Recap: try/catch handling errors

Newton School of Technology

Imagine a nice day, driving your car and suddenly met with an accident. But don't worry airbags are there to catch and save you.

Car/application running smoothly

A crash occurred causing an injury/error

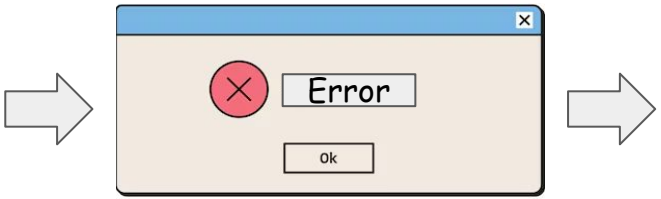
Thankfully error got caught not causing much damage

Recap: try/catch Saves your day

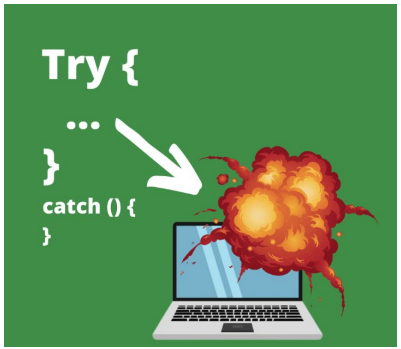
Newton School of Technology

Definitely we were not talking about car!!

Web application running smoothly



Error occurred



try...catch caught the error

Newton School of Technology

Using Error State to Manage Errors: try/catch

We can achieve same using try/catch instead.

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

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

    useEffect(() => {
        const fetchData = async () => {
            try {
                const response = await fetch("https://jsonplaceholder.typicode.com/todos/1");

                if (!response.ok) {
                    throw new Error(`HTTP error! Status: ${response.status}`);
                }
                const json = await response.json(); // ✅ Correctly parse JSON
                setData(json);
            } catch (err) {
                setError(err.message);
            }
        };

        fetchData();
    }, []);

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

export default App;
```

Any error raised inside try block gets catched up in the catch block

Newton School of Technology

Cleanup useEffect()

Releasing resources after the use

Resources you need to run React App

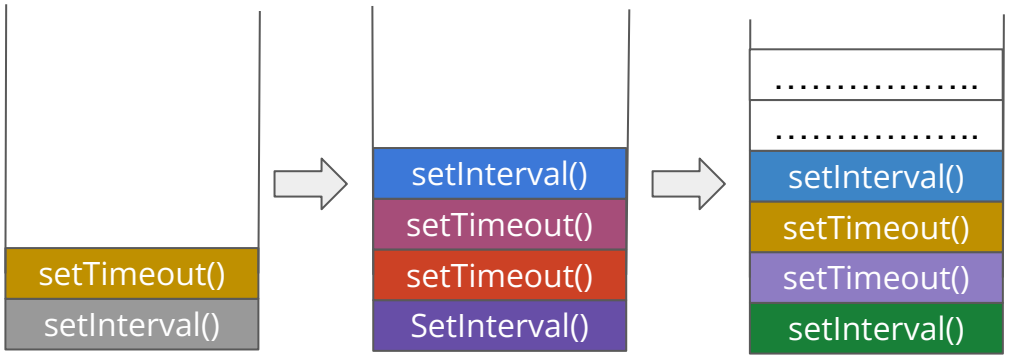
Many a time you need to acquire resources for instance to perform a repetitive task you need setInterval() and to finish a tasks after a certain time you need setTimeout().

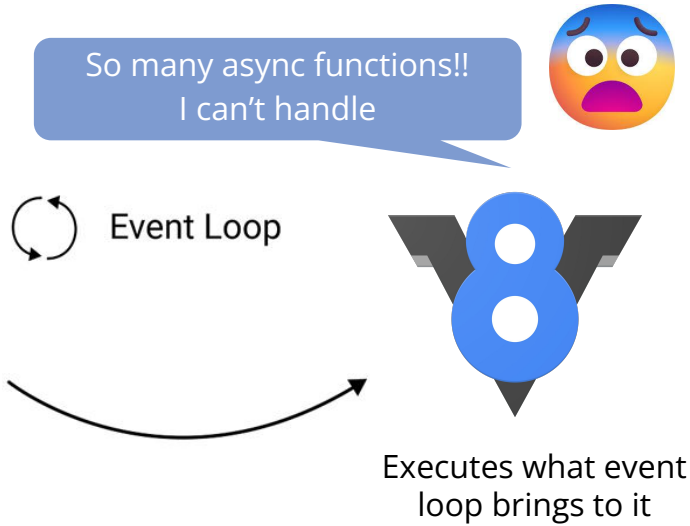
setTimeout

setInterval

But these resources stack if not released

These resources work asynchronously. If there are too many such resources executing simultaneously, we face performance issues. We need to release them if not in use.





When these resources are acquired

We acquire these resources in the mounting state inside useEffect(). Let's build a clock using setInterval() to elaborate this:-

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

function Clock() {
  const [time, setTime] = useState(new Date());

  useEffect(() => {
    const interval = setInterval(() => {
      setTime(new Date());
    }, 1000);
  }, []);

  return (
    <div style={clockStyle}>
      <h1>Live Clock</h1>
      <h2>{time.toLocaleTimeString()}</h2>
    </div>
  );
}

export default Clock;
```

Adding style to our clock

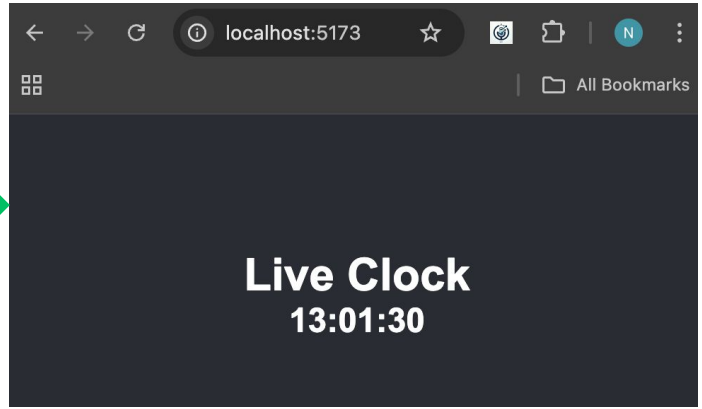
```jsx
const clockStyle = {
    display: "flex",
    flexDirection: "column",
    justifyContent: "center",
    alignItems: "center",
    backgroundColor: "#282c34",
    color: "white",
    fontFamily: "Arial",
};
```

Let’s Render it in browser

Let’s include it in the App.jsx and render in the browser.

What if we conditionally hide this clock component, would setInterval stop executing?

```jsx
import Clock from "./components/Clock";
import { useState } from "react";

function App() {
  const [show, setShow] = useState(true);

  return (
    <div style={AppStyle}>
      <Clock />
    </div>
  )
}

export default App;
```



Let’s Hide it conditionally

Here we have added added a hide button which when clicked hides the clock i.e. removes it from the DOM.

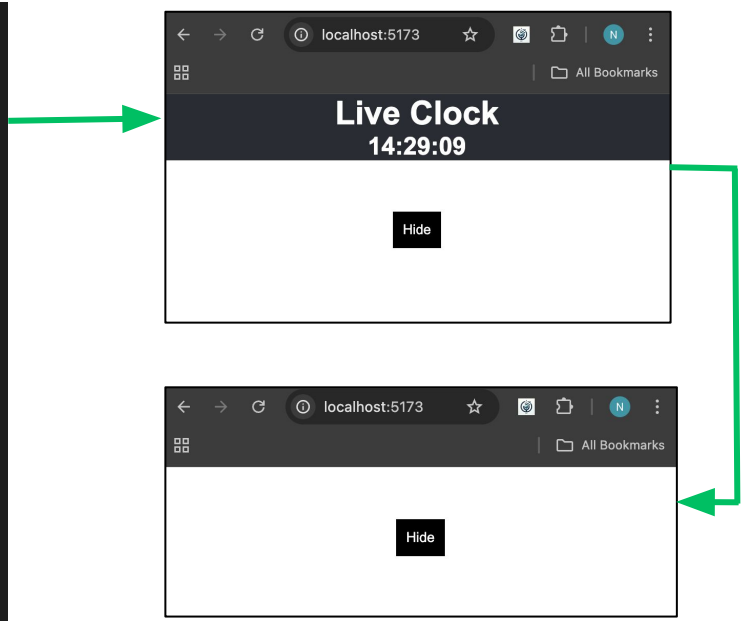
```jsx
function App() {
  const [show, setShow] = useState(true);

  return (
    <div style={AppStyle}>
      {show ? <Clock /> : ""}
     <div style={buttonContainer}>
        <button
            onClick={() => setShow(false)}
            style={ButtonStyle}
          >
            Hide
          </button>
      </div>
    </div>
  )
}
```

```jsx
const buttonContainer = {
  marginTop: "50px",
  width: "100%",
  display: "flex",
  justifyContent: "center"
}

const ButtonStyle = {
  color: "white",
  backgroundColor: "black",
  padding: "10px",
  border: "none",
  cursor: "pointer"
}
```

Let’s see how it looks now

Here when hide button is clicked then the <Clock /> component is removed from DOM, but does that mean setInterval also gets removed? I don’t think so.

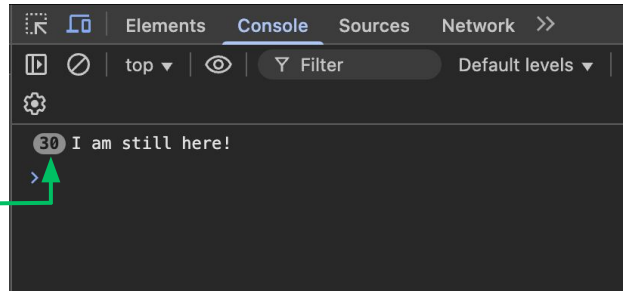
```
function App() {
  const [show, setShow] = useState(true);

  return (
    <div style={AppStyle}>
      {show ? <Clock /> : ""}
     <div style={buttonContainer}>
        <button
            onClick={() => setShow(false)}
            style={ButtonStyle}
          >
            Hide
          </button>
      </div>
    </div>
  )
}
```



What happens when hide is clicked?

Here when hide button is clicked then the <Clock /> component is removed from DOM, but does that mean setInterval also gets removed? I don't think so.

```
1  useEffect(() => {
2     const interval = setInterval(() => {
3       setTime(new Date());
4       console.log("I am still here!");
5     }, 1000);
6   }, []);
```



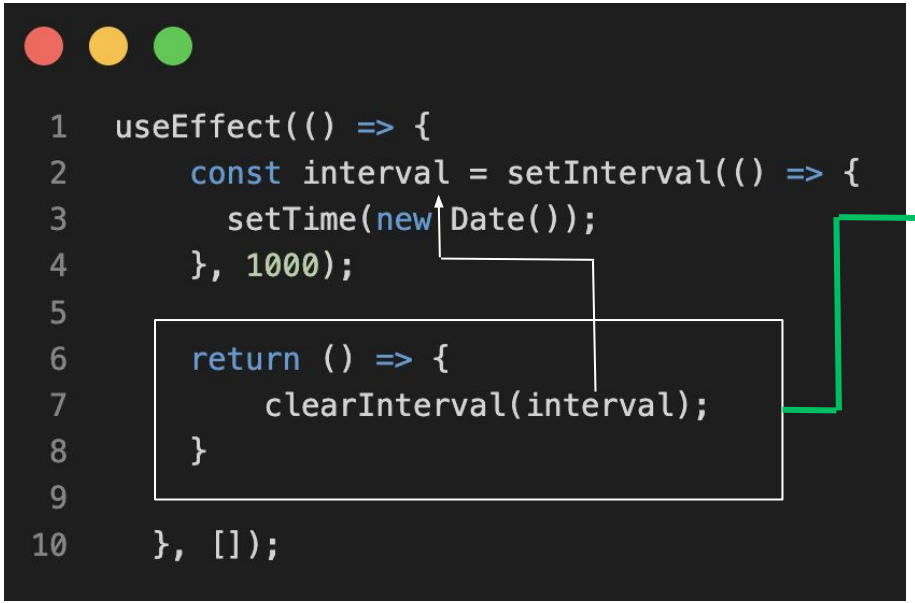
Message keep on printing even after Clock is unmounted

Add this debugging statement before clicking hide, and you will observe that this message will still be printed after every 1000ms

Observe this count if it keeps on increasing even after component is unmounted

How to avoid this: cleanup

If we attach a return block inside useEffect, it would be executed only after component is unmounted and there we can include code to remove resources from the DOM.



Block executes only after component unmounts.

And clearInterval() removed the interval from the DOM

This entire process is called CleanUp.

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!