See slides 205 - ..

* We define useEffect() hook based on Components lifecycle (initial render, re-render, unmount)
* side-effect is basically any interaction between a React component and the world outside the component
* Side-effects should Not be in render logic
* Effects allow us to write code that will run at different moments: mount, re-render or unmount
* Event handlers is the preferred way of creating side-effects rathen than useEffect

Example:

*UseEffect will take place only at initial Rendering. If we use only fetch then an infinite loop of http requests will take place*

 useEffect(function () {

    async function **fetchMovies**() {

      const res = await fetch(

        `http://www.omdbapi.com/?apikey=${KEY}&s=${query}`

      );

      const data = await res.json();

      setMovies(data.Search);

      console.log(data.Search);

    }

    fetchMovies();

****

**Your Explanation (Refined)**

✅ **useEffect** runs **side effects** in React, such as fetching data.  
✅ **An empty dependency array []** means it runs **only once, when the component mounts**.  
✅ Inside useEffect, an **async function (fetchMovies)** is created and executed.  
✅ **Inside the try block:**

* setIsLoading(true): Marks the start of the loading state.
* fetch() sends an HTTP request to the OMDb API.
* await res.json() converts the response to a JavaScript object.
* If the response contains Response: "False", it throws an error saying **"Movie not found"**.
* Otherwise, setMovies(data.Search) updates state with the fetched movie data.

✅ **Inside the catch block:**

* If an error occurs (e.g., network failure, invalid response), it is logged with console.log(err.message).
* The error message is stored in state with setError(err.message).

✅ **Inside the finally block:**

* setIsLoading(false) ensures that the loading state is **always** turned off, whether the request succeeds or fails.

**The useEffect Dependency Array**

