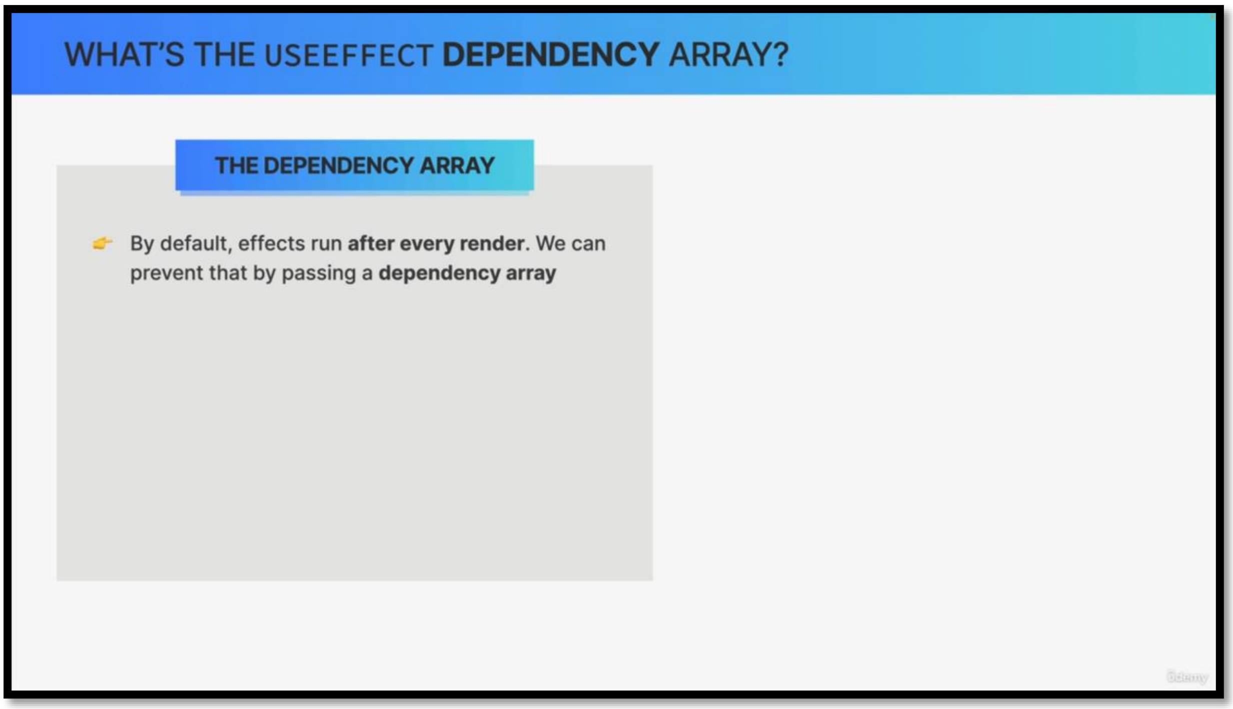
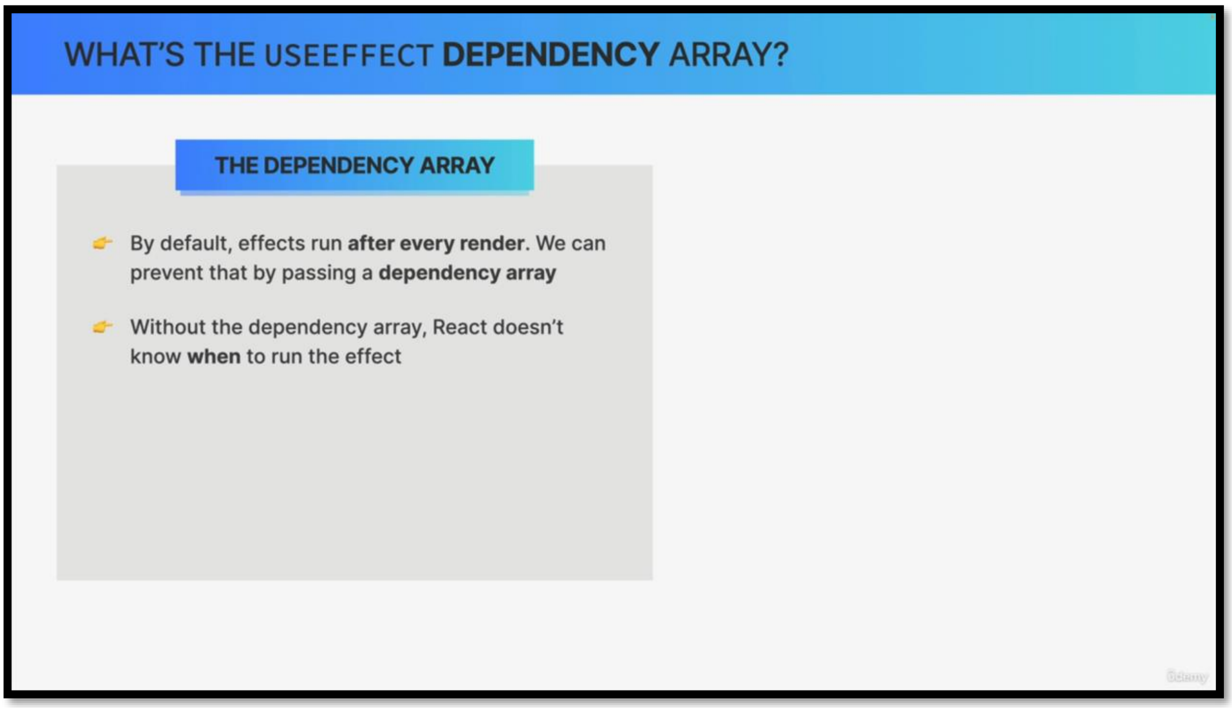


WHAT'S THE USEEFFECT **DEPENDENCY** ARRAY?

THE DEPENDENCY ARRAY

- By default, effects run **after every render**. We can prevent that by passing a **dependency array**

By default, in effect will run after each and every render. However, that's almost never what we want. The good news is that we can change this default behavior by passing a dependency array into the useEffect hook as a second argument.

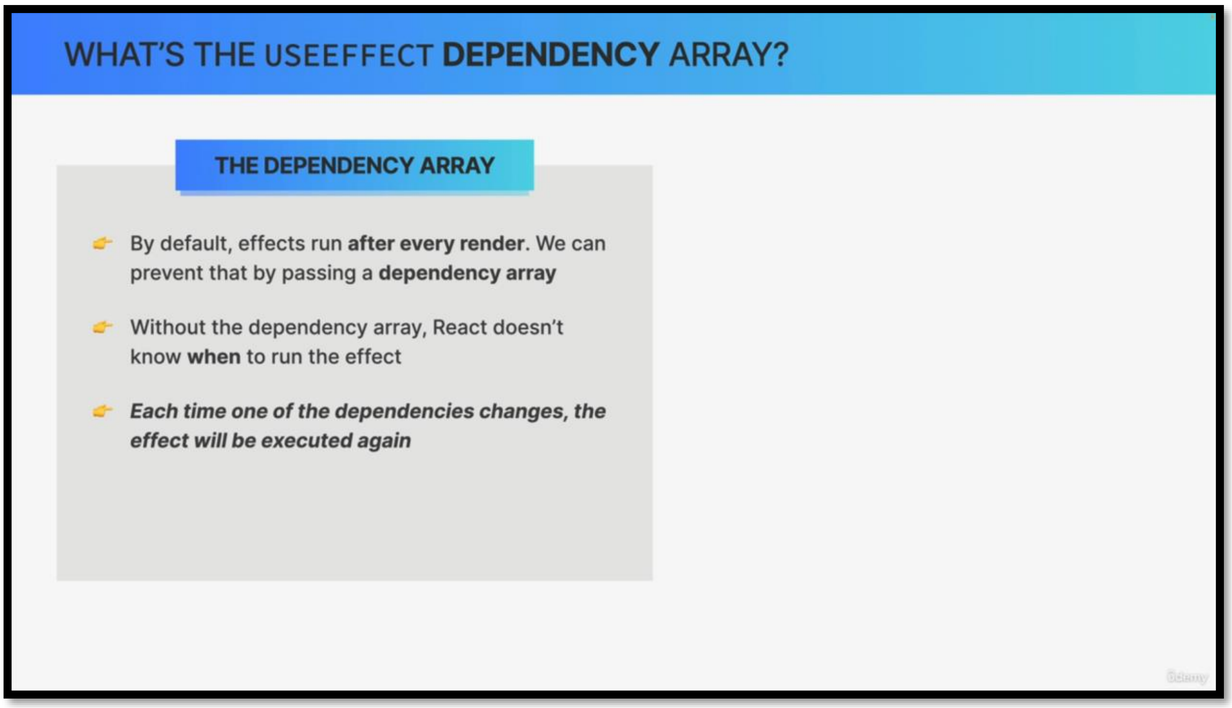


WHAT'S THE USEEFFECT **DEPENDENCY** ARRAY?

THE DEPENDENCY ARRAY

- By default, effects run **after every render**. We can prevent that by passing a **dependency array**
- Without the dependency array, React doesn't know **when** to run the effect

Why does use effect actually need an array of dependencies?
The reason is that without this array, React doesn't know when to actually run the effect.

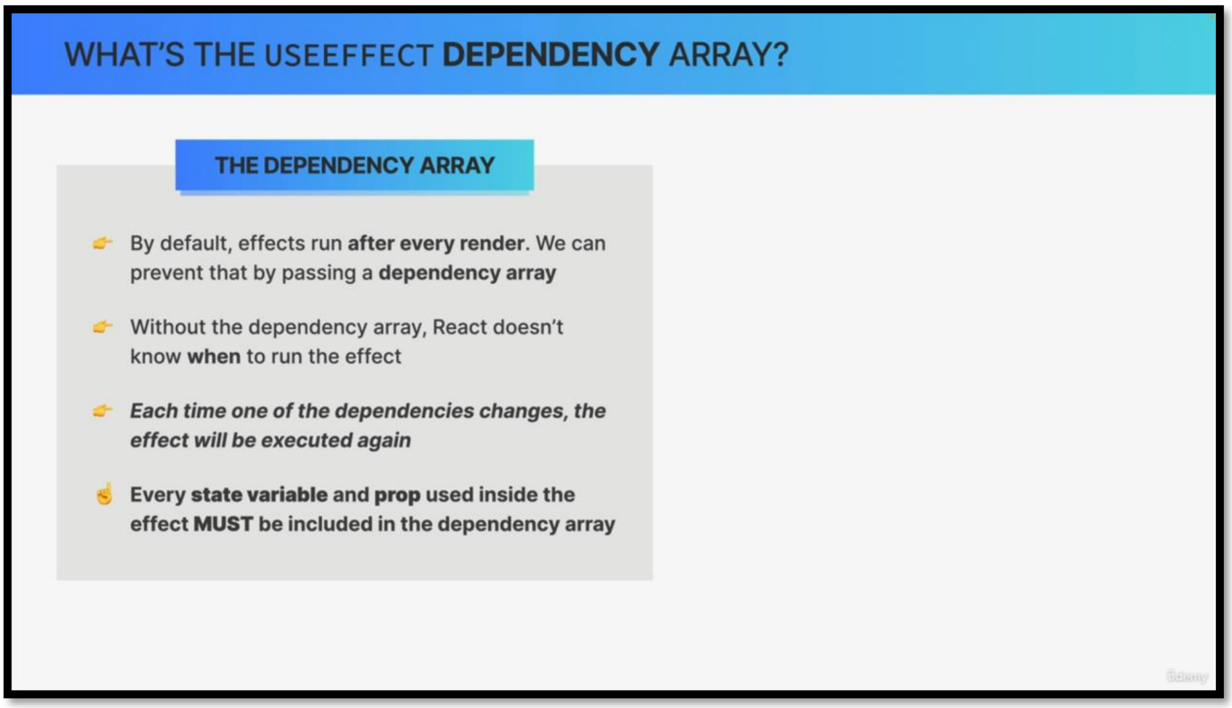


WHAT'S THE USEEFFECT **DEPENDENCY** ARRAY?

THE DEPENDENCY ARRAY

- By default, effects run **after every render**. We can prevent that by passing a **dependency array**
- Without the dependency array, React doesn't know **when** to run the effect
- ***Each time one of the dependencies changes, the effect will be executed again***

If we do specify the effect dependencies by passing in the dependency array, the effect will be executed each time that one of the dependencies changes.

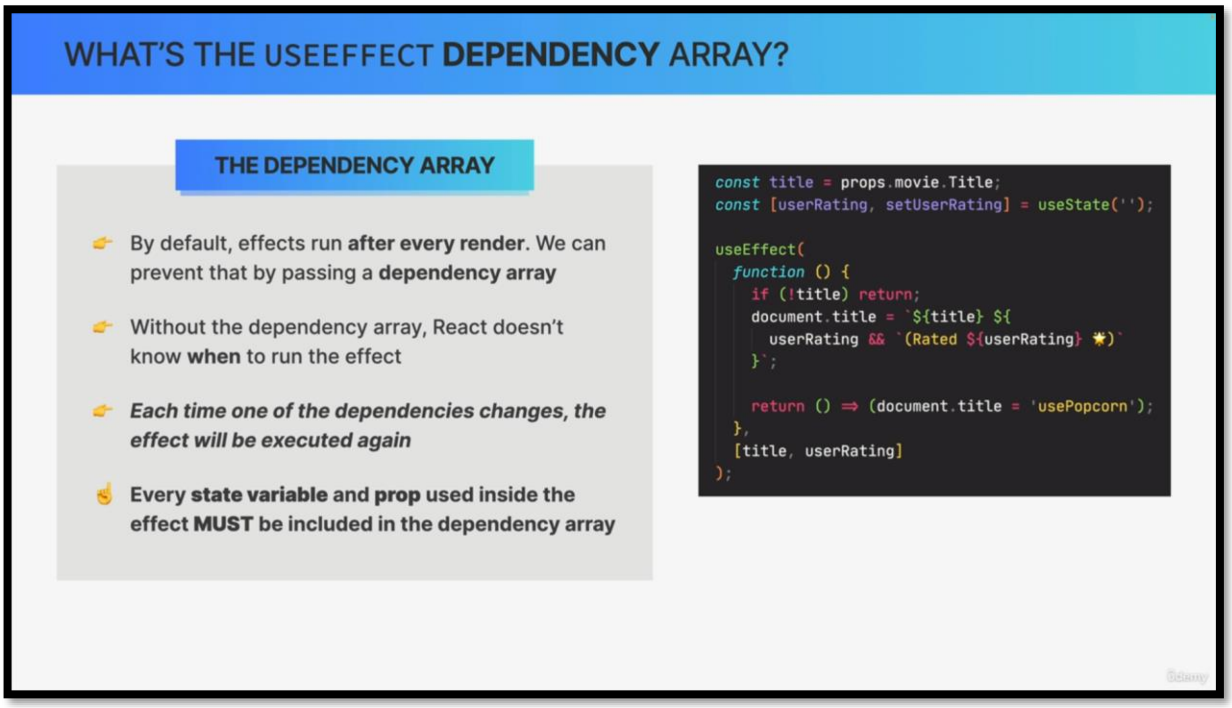


WHAT'S THE USEEFFECT **DEPENDENCY** ARRAY?

THE DEPENDENCY ARRAY

- By default, effects run **after every render**. We can prevent that by passing a **dependency array**
- Without the dependency array, React doesn't know **when** to run the effect
- ***Each time one of the dependencies changes, the effect will be executed again***
- **Every state variable and prop used inside the effect MUST be included in the dependency array**

What exactly are those dependencies?
Effect dependencies are state variables and props that are used inside the effect. The rule is that each and every one of those state variables and props must be included in the dependency array.



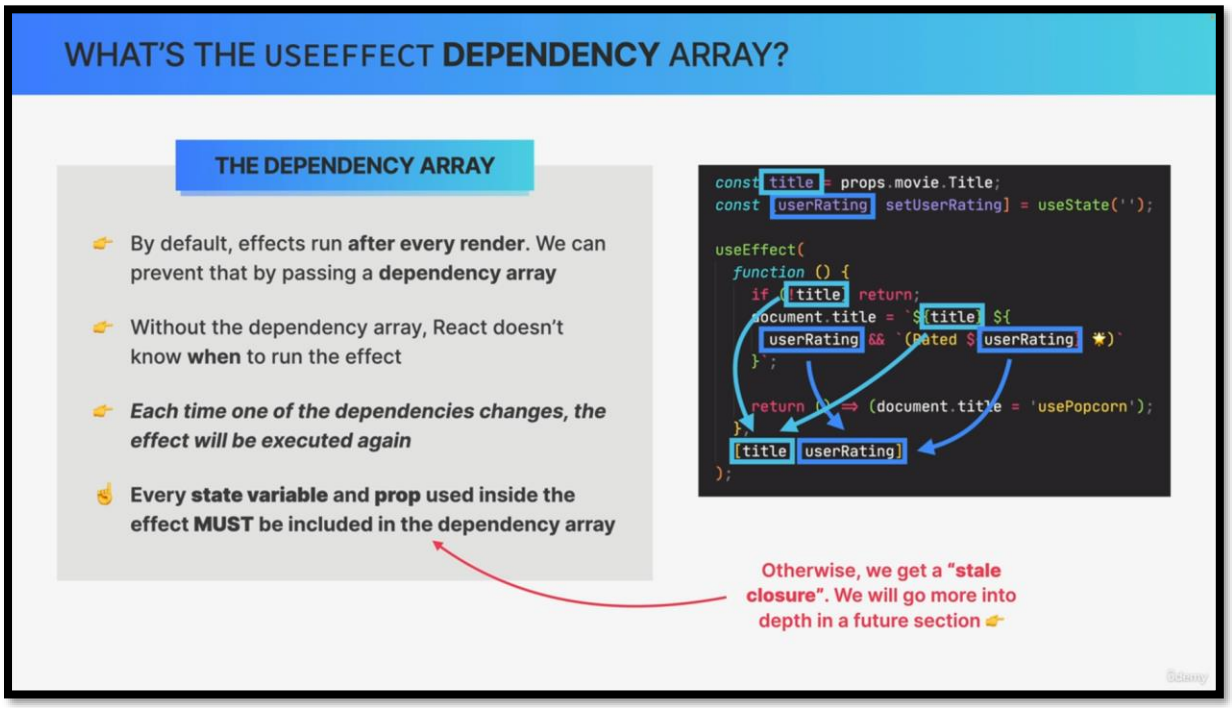
WHAT'S THE USEEFFECT **DEPENDENCY** ARRAY?

THE DEPENDENCY ARRAY

- 👉 By default, effects run **after every render**. We can prevent that by passing a **dependency array**
- 👉 Without the dependency array, React doesn't know **when** to run the effect
- 👉 ***Each time one of the dependencies changes, the effect will be executed again***
- ☝️ **Every state variable and prop used inside the effect MUST be included in the dependency array**

```
const title = props.movie.Title;
const [userRating, setUserRating] = useState('');

useEffect(
  function () {
    if (!title) return;
    document.title = `${title} ${
      userRating && `(Rated ${userRating} ⭐)`
    }`;

    return () => (document.title = 'usePopcorn');
  },
  [title, userRating]
);
```

The effect uses the title prop and the user rating state. We can clearly see at the top of the code that title is indeed a prop and that user rating is indeed a piece of state.



WHAT'S THE USEEFFECT **DEPENDENCY** ARRAY?

THE DEPENDENCY ARRAY

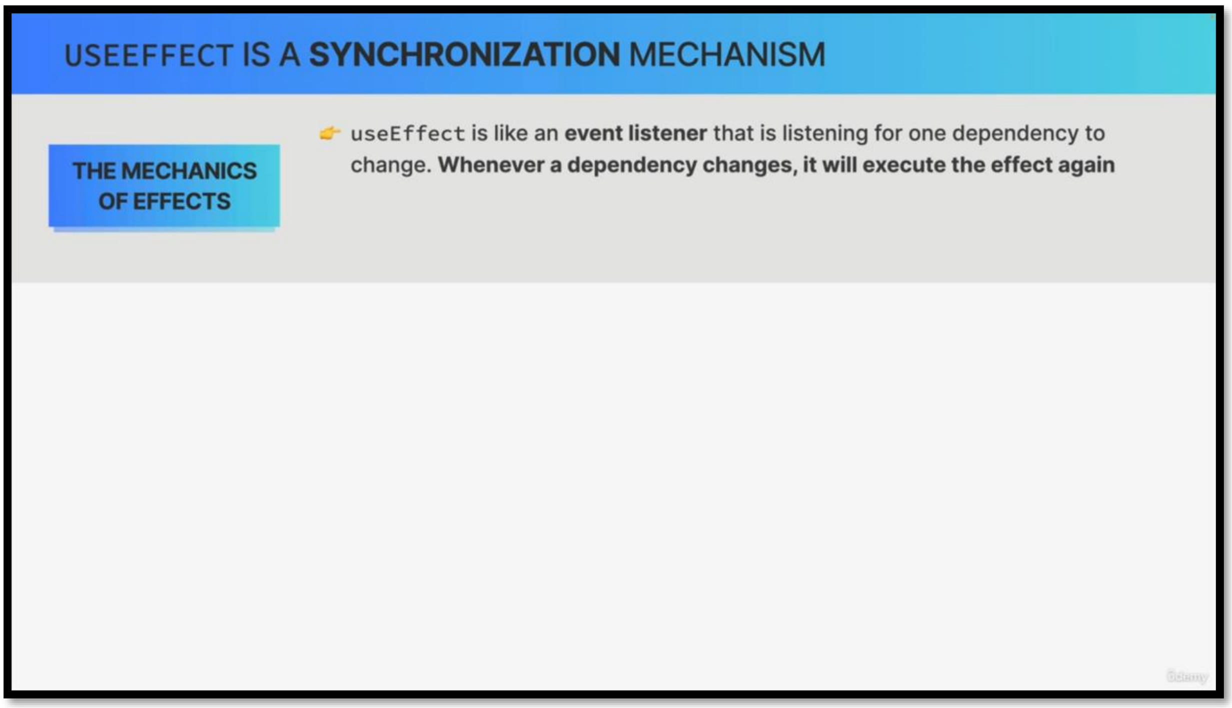
- 👉 By default, effects run **after every render**. We can prevent that by passing a **dependency array**
- 👉 Without the dependency array, React doesn't know **when** to run the effect
- 👉 ***Each time one of the dependencies changes, the effect will be executed again***
- ☝️ **Every state variable and prop used inside the effect MUST be included in the dependency array**

```
const title = props.movie.Title;
const [userRating, setUserRating] = useState('');

useEffect(
  function () {
    if (!title) return;
    document.title = `${title} ${
      userRating && `(Rated ${userRating} ⭐)`
    }`;

    return () => (document.title = 'usePopcorn');
  },
  [title, userRating]
);
```

Otherwise, we get a **"stale closure"**. We will go more into depth in a future section 👉

Therefore, both of them must be included in the dependency array. So, the effect function depends on these variables to do its work and therefore we need to tell React about them.

Otherwise, if the title or the user rating changes, React will not know about this change and therefore, it won't be able to re-execute the effect code. This will then lead to a bug called stale closure.

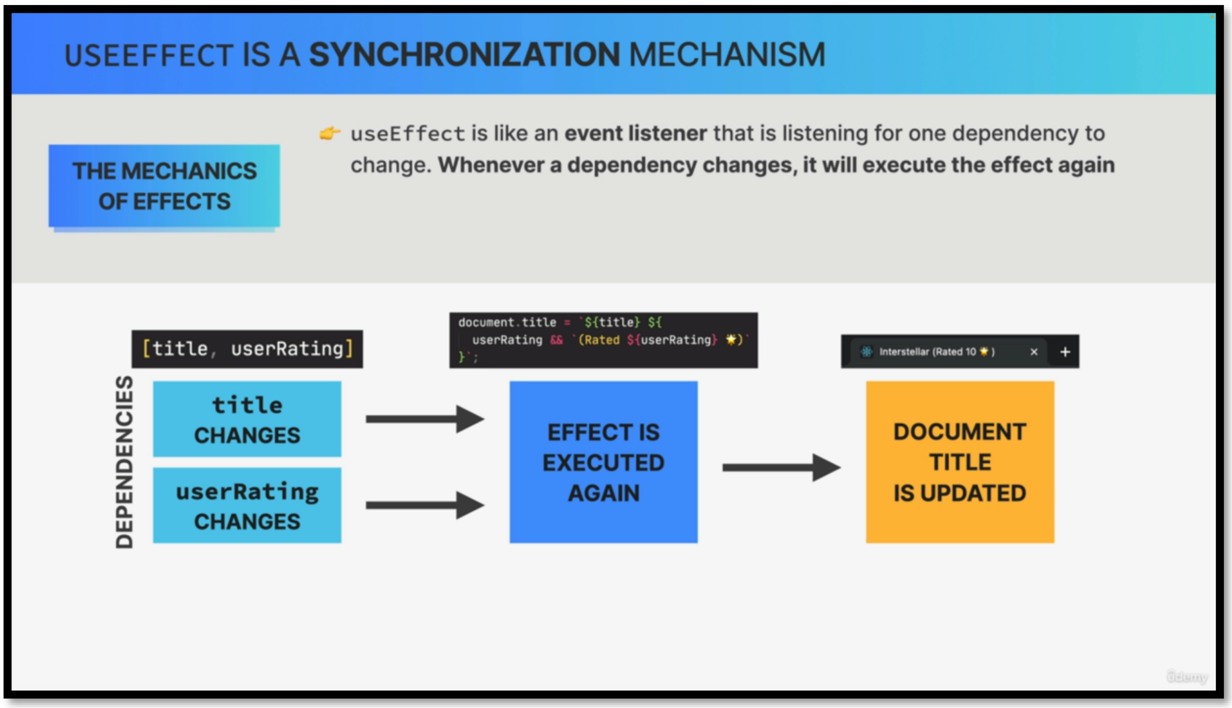


USEEFFECT IS A **SYNCHRONIZATION** MECHANISM

THE MECHANICS OF EFFECTS

👉 `useEffect` is like an **event listener** that is listening for one dependency to change. **Whenever a dependency changes, it will execute the effect again**

We can think of the useEffect hook as an event listener that is listening for one or more dependencies to change. When one of the dependencies does change, use effect will simply execute the effect again. So, a bit like a regular event listener, but for effects.



USEEFFECT IS A **SYNCHRONIZATION** MECHANISM

THE MECHANISMS OF EFFECTS

- `useEffect` is like an **event listener** that is listening for one dependency to change. **Whenever a dependency changes, it will execute the effect again**

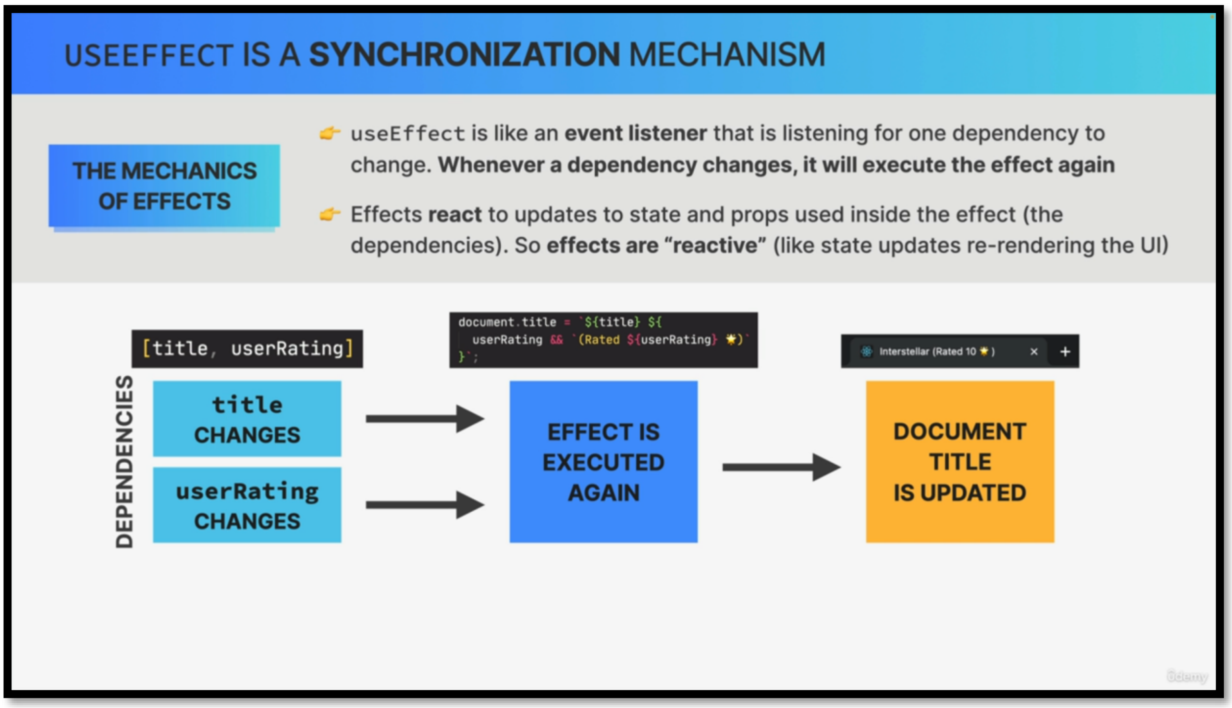
```
[title, userRating]
```

```
document.title = `${title} ${
  userRating && `(Rated ${userRating} 🌟)`
}`;
```

Let's go back to our previous example where we had the title and user rating dependencies in the array. So, whenever the title or the user rating changes, React will execute the effect again.

So, it will run the code one more time which will in turn update the document title. So, the website title that we see in a browser tab.

So, essentially, effects react to updates to state and props that are used inside the effect because, again, those are the effects' dependencies.



USEEFFECT IS A **SYNCHRONIZATION** MECHANISM

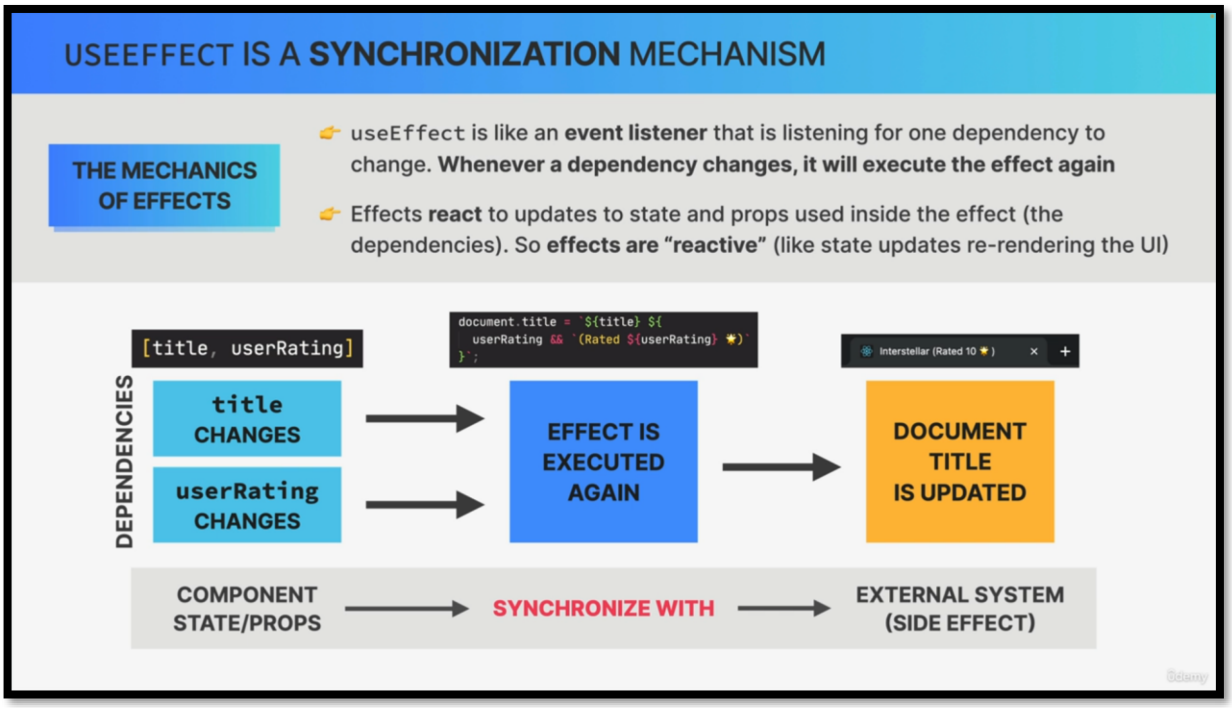
THE MECHANISMS OF EFFECTS

- `useEffect` is like an **event listener** that is listening for one dependency to change. **Whenever a dependency changes, it will execute the effect again**
- Effects **react** to updates to state and props used inside the effect (the dependencies). So **effects are "reactive"** (like state updates re-rendering the UI)

```
[title, userRating]
```

```
document.title = `${title} ${
  userRating && `(Rated ${userRating} 🌟)`
}`;
```

In a way, effects are reactive, just like React reacts to state updates by re-rendering the UI. This is extremely useful and powerful but, all this only works if we correctly specify the dependency array.



USEEFFECT IS A **SYNCHRONIZATION** MECHANISM

THE MECHANICS OF EFFECTS

- `useEffect` is like an **event listener** that is listening for one dependency to change. **Whenever a dependency changes, it will execute the effect again**
- Effects **react** to updates to state and props used inside the effect (the dependencies). So **effects are "reactive"** (like state updates re-rendering the UI)

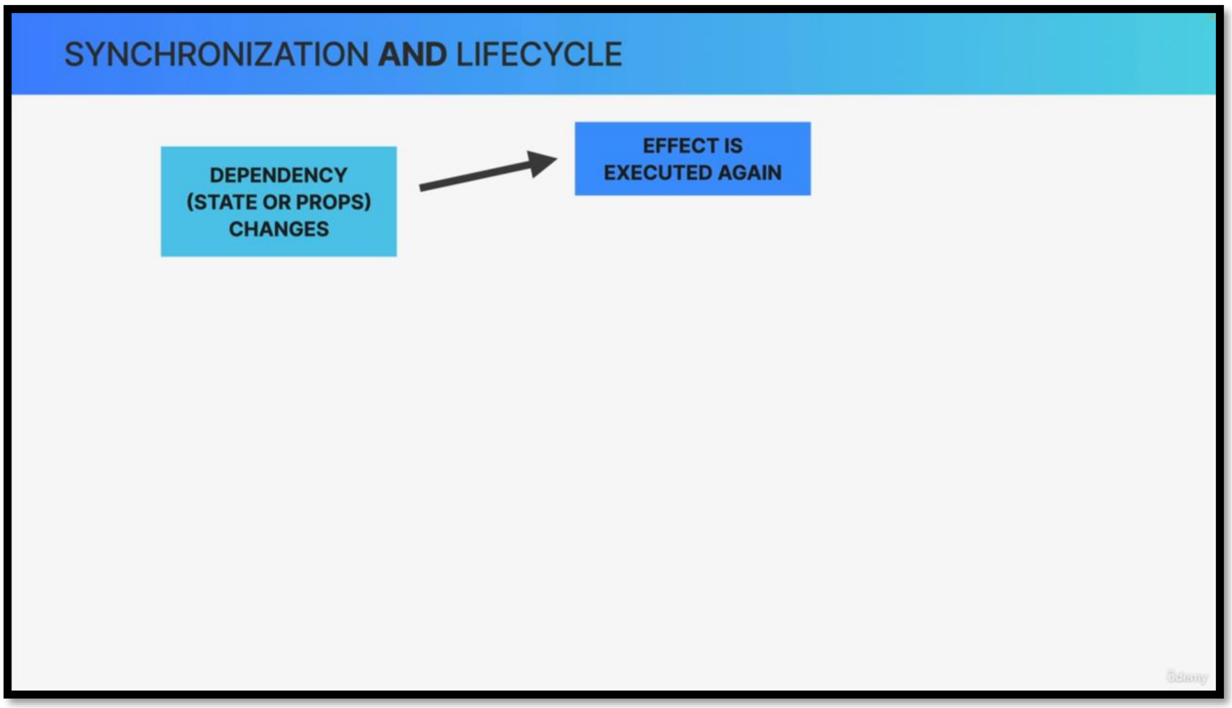
```
[title, userRating]
```

```
document.title = `${title} ${
  userRating && `(Rated ${userRating} 🌟)`
}`;
```

Effects are used to keep a component synchronized with some external system i.e. some system that lives outside of our React based code.

If we think about it, that's exactly what is happening here. So, the state and props of our component are now in fact synchronized with an external system, which is, in this case, the title of the document.

useEffect truly is a synchronization mechanism. So, a mechanism to synchronize effects with the state of the application.

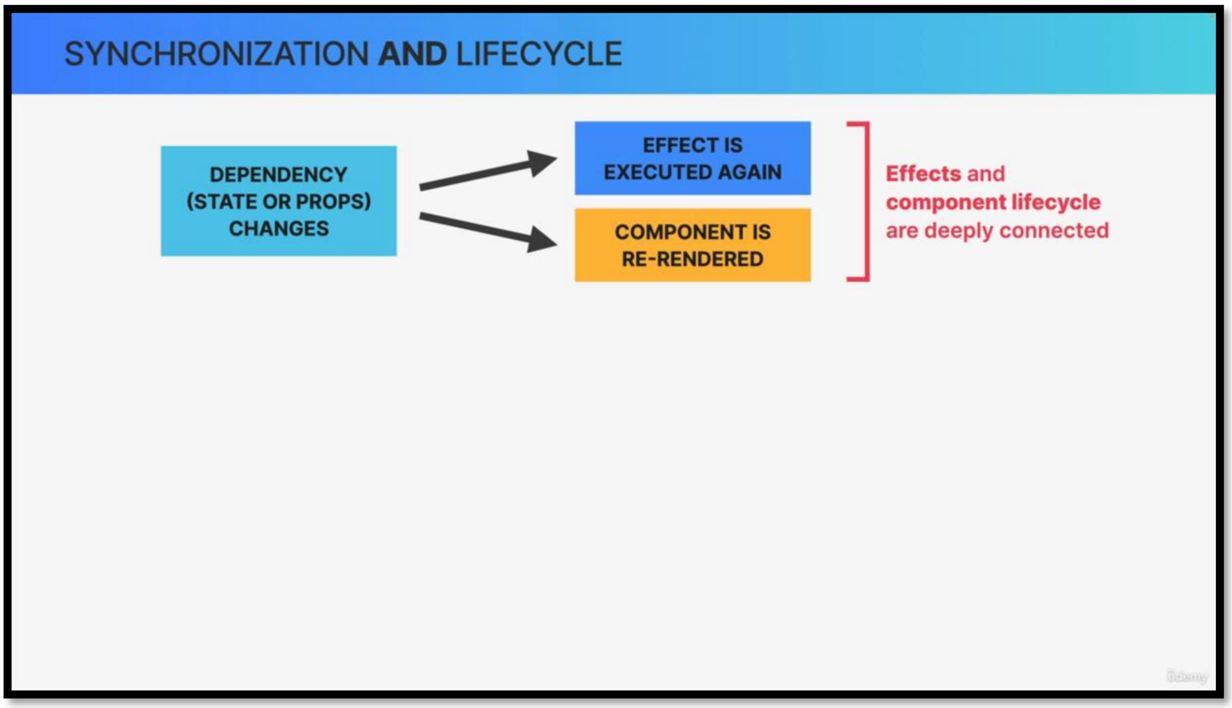


SYNCHRONIZATION AND LIFECYCLE

$$\begin{tikzcd}\text{DEPENDENCY (STATE OR PROPS) CHANGES} \arrow[r] & \text{EFFECT IS EXECUTED AGAIN}\end{tikzcd}$$

Whenever a dependency changes, the effect is executed again.

Dependencies are always state or props.



SYNCHRONIZATION AND LIFECYCLE

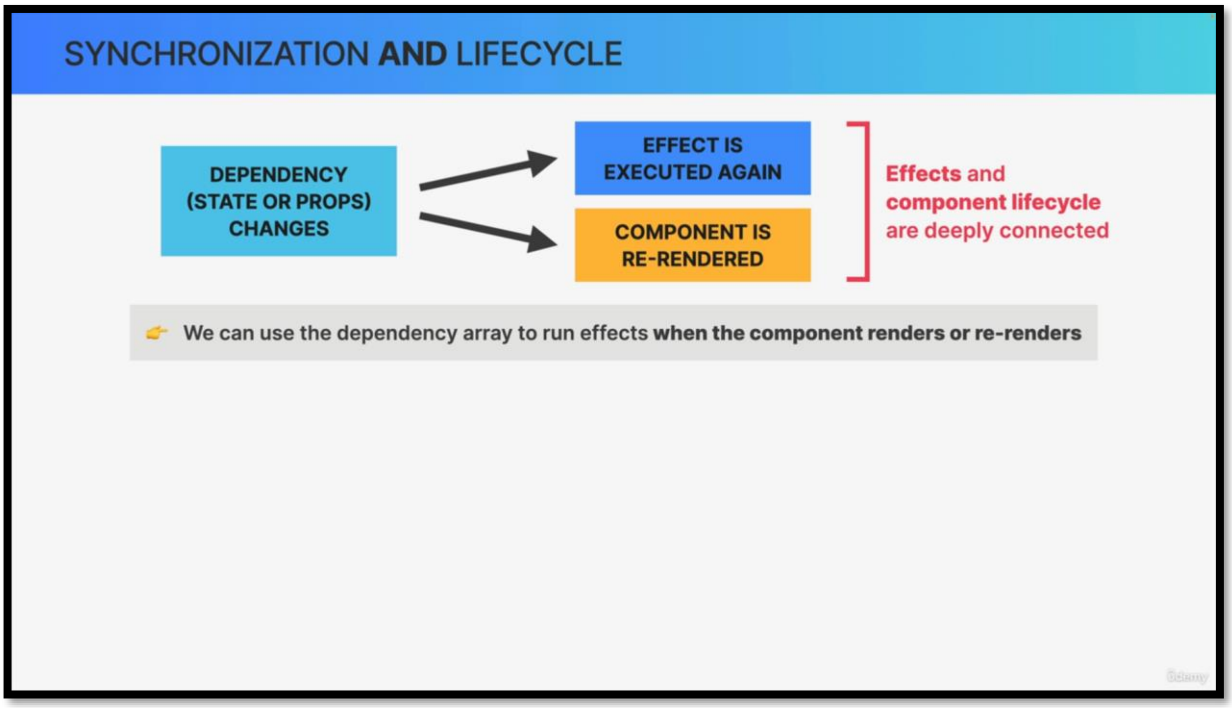
$$\begin{tikzcd}\text{DEPENDENCY (STATE OR PROPS) CHANGES} \arrow[r] \arrow[dr] & \text{EFFECT IS EXECUTED AGAIN} \\ & \text{COMPONENT IS RE-RENDERED}\end{tikzcd}$$

Effects and **component lifecycle** are deeply connected

What happens to a component each time that its state or props are updated?
The component will re-render.

This means that effects and the life cycle of a component instance are deeply interconnected.

That's why when the useEffect hook was first introduced, many people thought that it was a life cycle hook rather than a hook for synchronizing the component with a side effect.



SYNCHRONIZATION **AND** LIFECYCLE

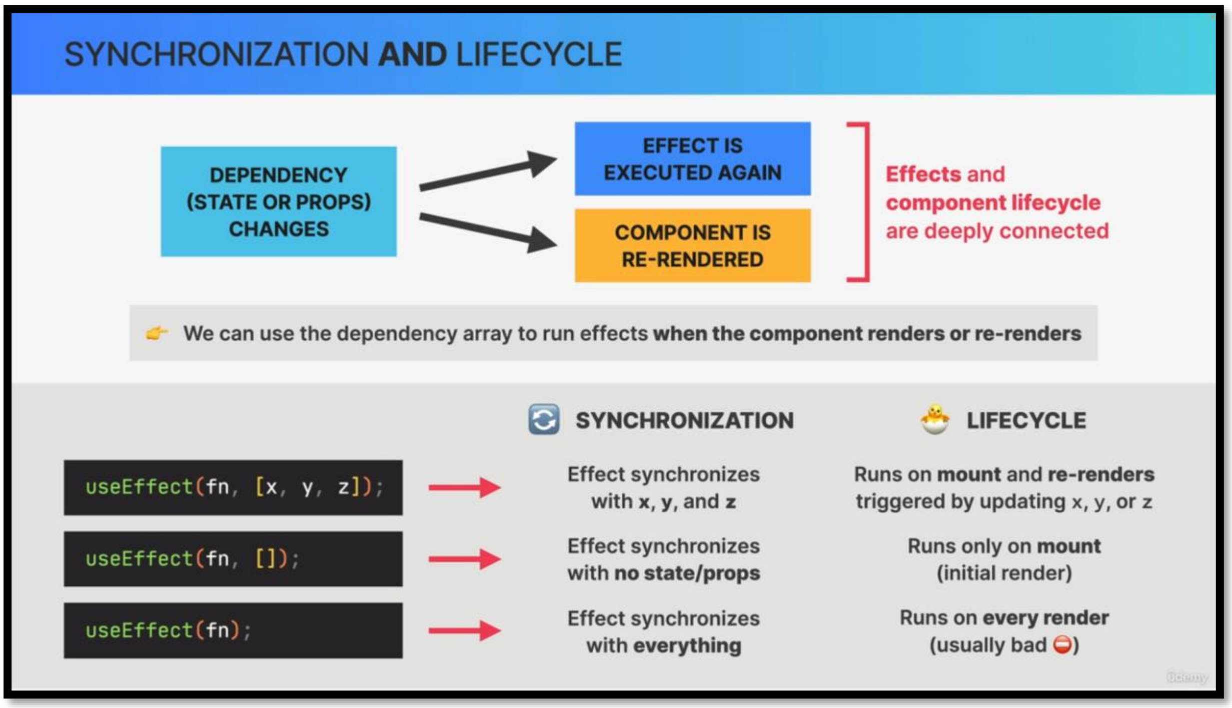
DEPENDENCY (STATE OR PROPS) CHANGES
EFFECT IS EXECUTED AGAIN
COMPONENT IS RE-RENDERED
Effects and component lifecycle are deeply connected

👉 We can use the dependency array to run effects **when the component renders or re-renders**

The conclusion and the big takeaway from this is that we can use the dependency array in order to run effects whenever the component renders or re-renders.

So, in a way, the useEffect hook is actually about synchronization and about the component life cycle

Three types of dependencies are



SYNCHRONIZATION AND LIFECYCLE

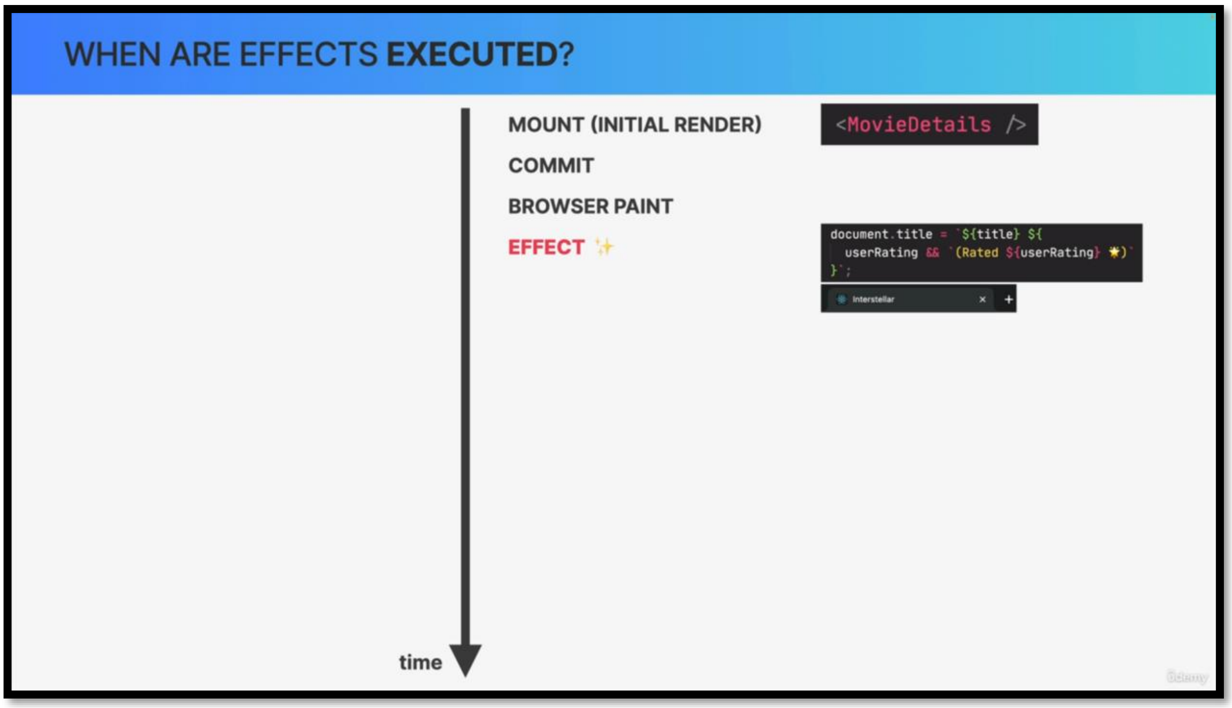
DEPENDENCY (STATE OR PROPS) CHANGES → EFFECT IS EXECUTED AGAIN

DEPENDENCY (STATE OR PROPS) CHANGES → COMPONENT IS RE-RENDERED

Effects and component lifecycle are deeply connected

👉 We can use the dependency array to run effects **when the component renders or re-renders**

		SYNCHRONIZATION	LIFECYCLE
`useEffect(fn, [x, y, z]);`	→	Effect synchronizes with **x, y, and z**	Runs on **mount** and **re-renders** triggered by updating x, y, or z
`useEffect(fn, []);`	→	Effect synchronizes with **no state/props**	Runs only on **mount** (initial render)
`useEffect(fn);`	→	Effect synchronizes with **everything**	Runs on **every render** (usually bad ⛔)



WHEN ARE EFFECTS EXECUTED?

MOUNT (INITIAL RENDER)

COMMIT

BROWSER PAINT

EFFECT ✨

time

```
<MovieDetails />
```

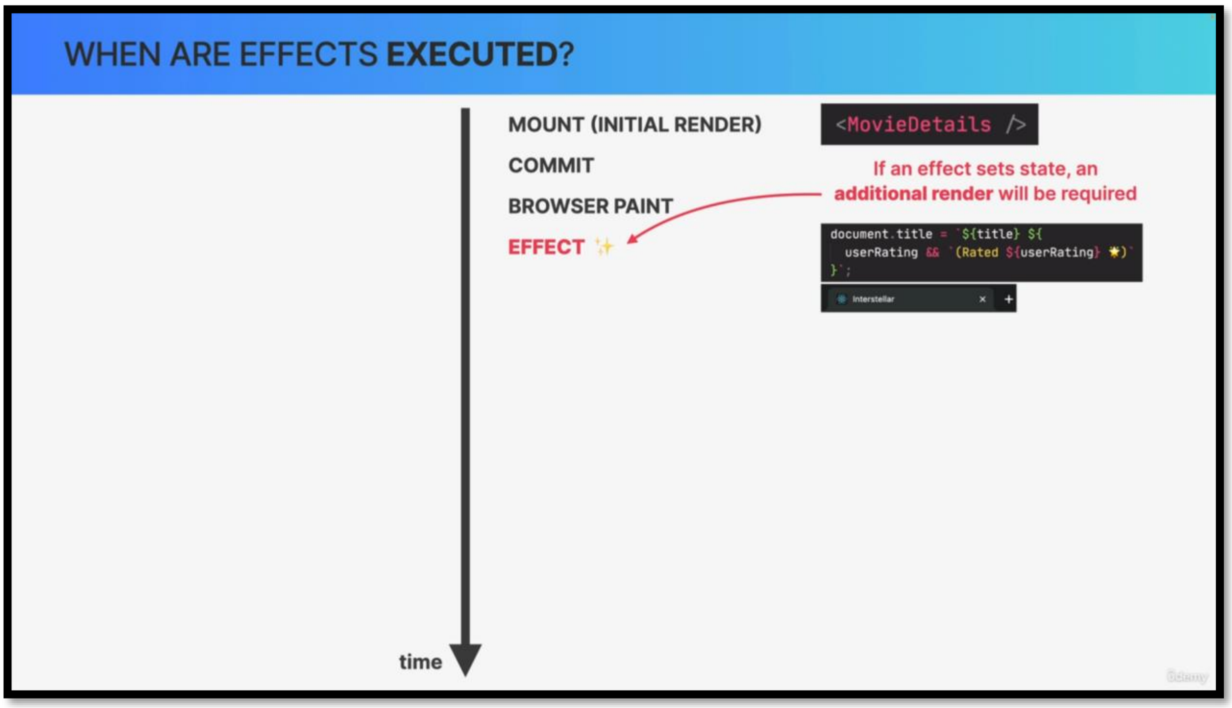
```
document.title = `${title} ${
  userRating && `(Rated ${userRating} ⭐)`
}`;
```

Effects are executed after the browser has painted the component instance on the screen.

Not immediately after render, as you might have thought initially. That's why we say that effects run asynchronously after the render has already been painted to the screen.

The reason why effect work this way is that effects may contain long-running processes such as fetching data.

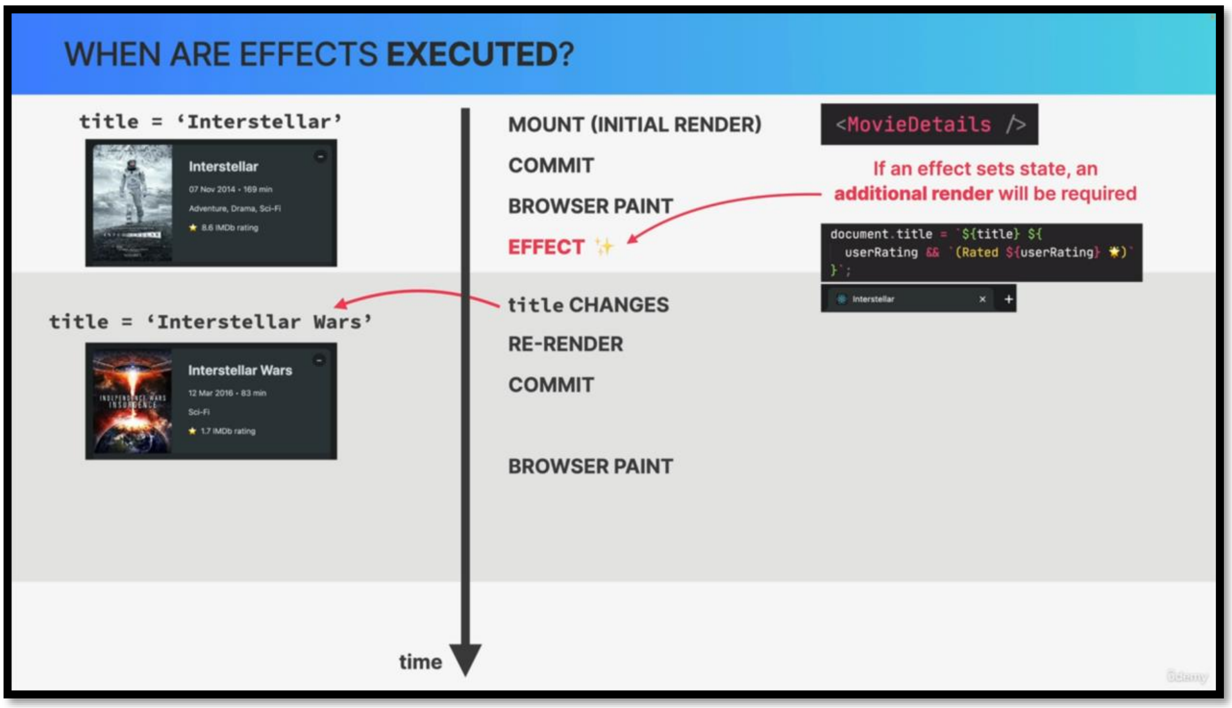
So, in a situation like that, if React would execute the effect before the browser paints a new screen, it would block this entire process and users would see an old version of the component for way too long. That would be very undesirable.


WHEN ARE EFFECTS EXECUTED?
MOUNT (INITIAL RENDER)
COMMIT
BROWSER PAINT
EFFECT ✨
time
<MovieDetails />
If an effect sets state, an additional render will be required
Interstellar

```
document.title = `${title} ${
  userRating && `(Rated ${userRating} ⭐)`
}`;
```

One important consequence of the fact that effects do not run during render is that if an effect sets state, a second additional render will be required to display the UI correctly.

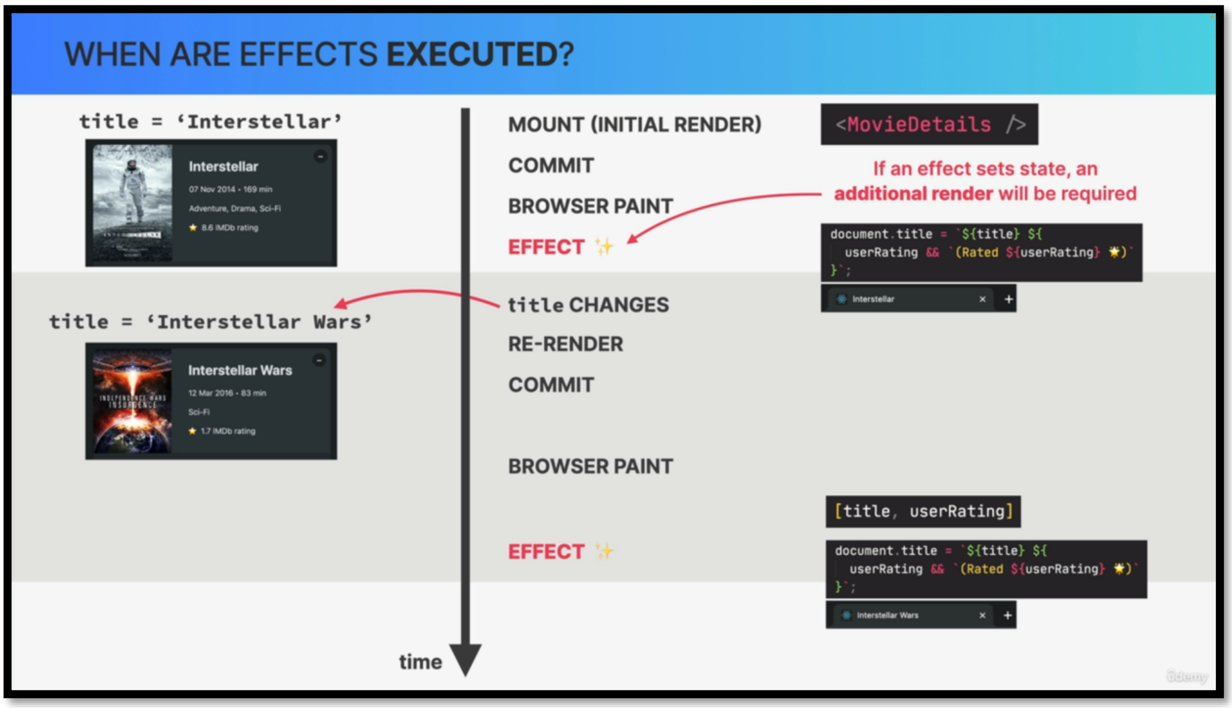
So, this is one of the reasons why you shouldn't overuse effects.



```
document.title = `${title} ${
  userRating && `(Rated ${userRating} ⭐)`
}`;
```

Let's say that the title was initially set to Interstellar but then it changes to Interstellar Wars.

Since this title is a prop, it means that the component will re-render, the dom changes will be committed and painted to the screen again.



WHEN ARE EFFECTS **EXECUTED?**

title = 'Interstellar'

Interstellar
07 Nov 2014 · 169 min
Adventure, Drama, Sci-Fi
★ 8.6 IMDb rating

title = 'Interstellar Wars'

Interstellar Wars
12 Mar 2016 · 83 min
Sci-Fi
★ 1.7 IMDb rating

MOUNT (INITIAL RENDER)
COMMIT
BROWSER PAINT
EFFECT ✨
title CHANGES
RE-RENDER
COMMIT
BROWSER PAINT
EFFECT ✨
time

<MovieDetails />

If an effect sets state, an **additional render** will be required

```
document.title = `${title} ${
  userRating && `(Rated ${userRating} ⭐)`
}`;
```

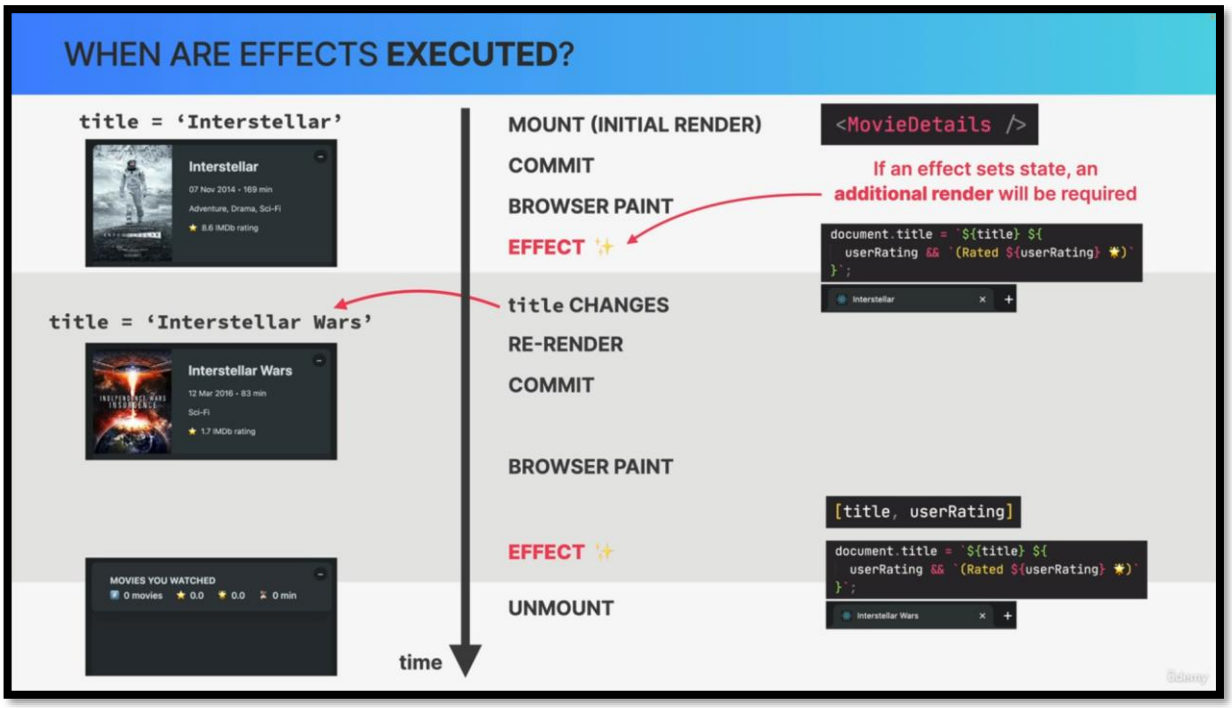
Interstellar

[title, userRating]

```
document.title = `${title} ${
  userRating && `(Rated ${userRating} ⭐)`
}`;
```

Interstellar Wars

Since title is part of the dependency array of this effect, the effect will be executed again at this point.



WHEN ARE EFFECTS **EXECUTED?**

title = 'Interstellar'

Interstellar
07 Nov 2014 · 169 min
Adventure, Drama, Sci-Fi
★ 8.6 IMDb rating

title = 'Interstellar Wars'

Interstellar Wars
12 Mar 2016 · 83 min
Sci-Fi
★ 1.7 IMDb rating

MOVIES YOU WATCHED
0 movies ⭐ 0.0 ⭐ 0.0 ⏳ 0 min

MOUNT (INITIAL RENDER)
COMMIT
BROWSER PAINT
EFFECT ✨
title CHANGES
RE-RENDER
COMMIT
BROWSER PAINT
EFFECT ✨
UNMOUNT
time

<MovieDetails />

If an effect sets state, an **additional render** will be required

```
document.title = `${title} ${
  userRating && `(Rated ${userRating} ⭐)`
}`;
```

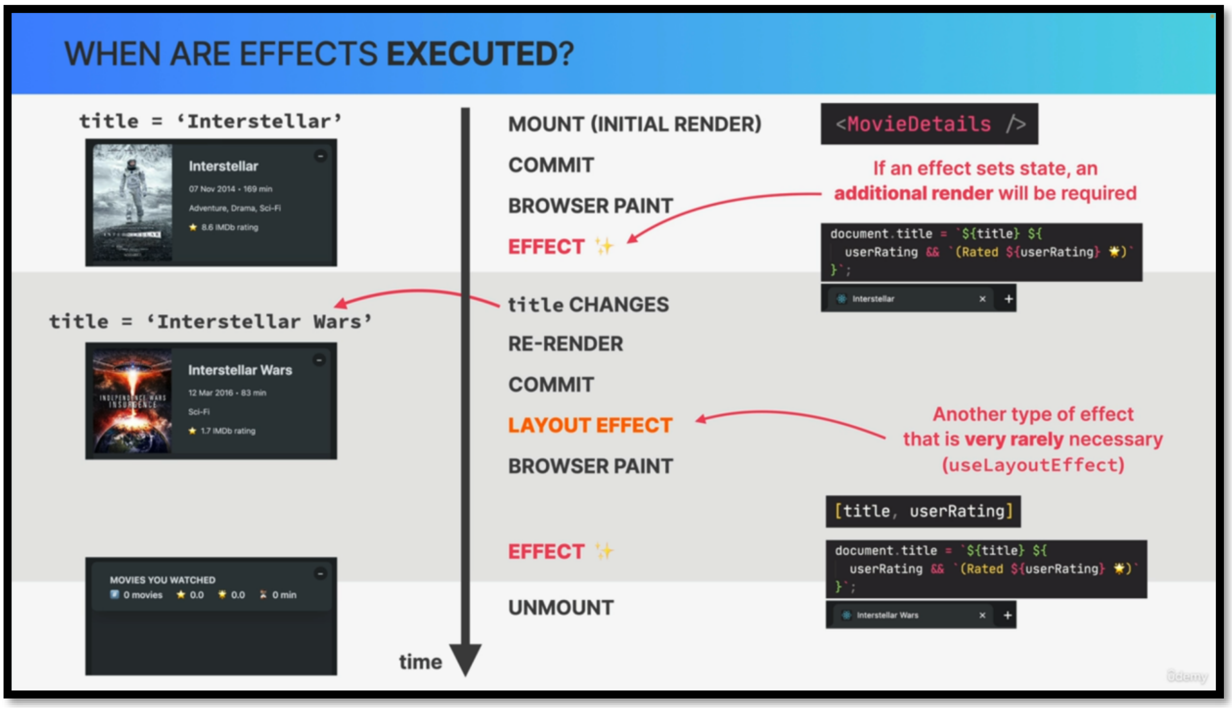
Interstellar

[title, userRating]

```
document.title = `${title} ${
  userRating && `(Rated ${userRating} ⭐)`
}`;
```

Interstellar Wars

This whole process can of course be repeated over and over again until this movie details instance finally unmounts and disappears from the screen.



WHEN ARE EFFECTS EXECUTED?

title = 'Interstellar'

MOUNT (INITIAL RENDER)
COMMIT
BROWSER PAINT
EFFECT ✨

title = 'Interstellar Wars'

title CHANGES
RE-RENDER
COMMIT
LAYOUT EFFECT
BROWSER PAINT

EFFECT ✨

UNMOUNT

time

```
<MovieDetails />
```

If an effect sets state, an **additional render** will be required

```
document.title = `${title} ${
  userRating && `(Rated ${userRating} ⭐)`
}`;
```

Another type of effect that is **very rarely** necessary (useLayoutEffect)

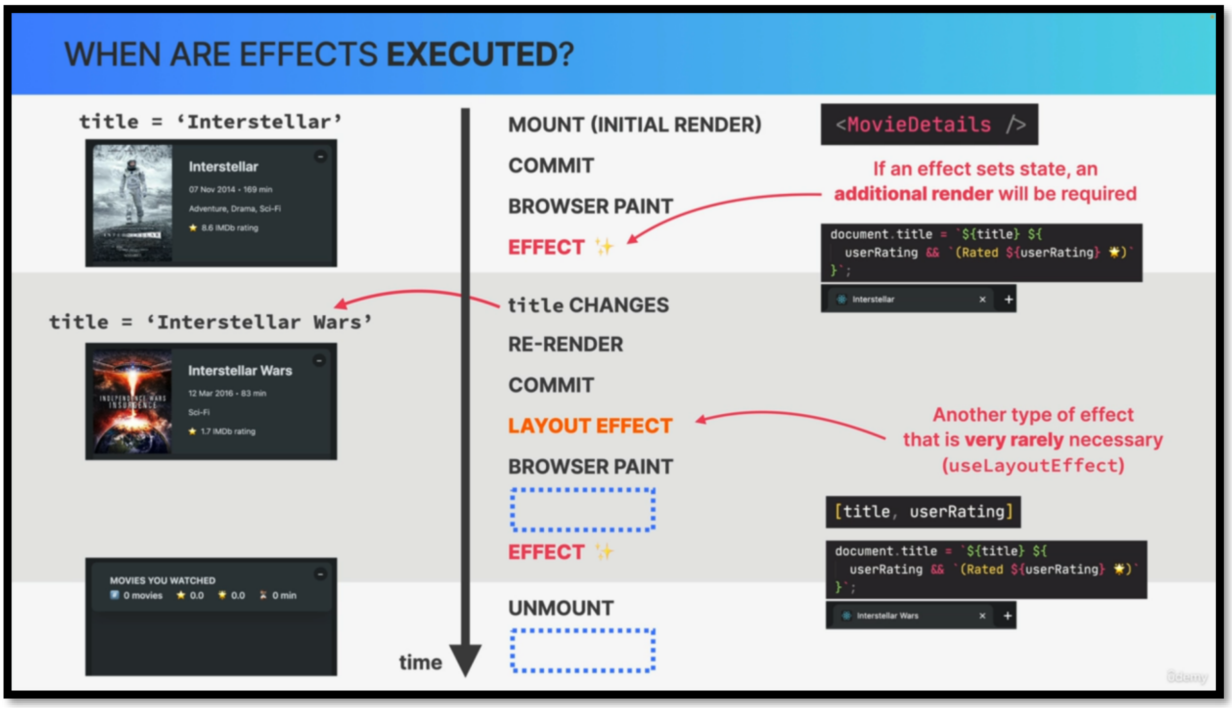
```
[title, userRating]
```

```
document.title = `${title} ${
  userRating && `(Rated ${userRating} ⭐)`
}`;
```

You might notice that there is actually a hole between the commit and browser paint.

The reason is that, in React, there's actually another type of effect called a layout effect. So, the only difference between a regular effect and a layout effect is that the layout effect runs before the browser actually paints the new screen.

But, we almost never need this. So, the React team actually discourages the use of this use layout effect hook.



WHEN ARE EFFECTS **EXECUTED?**

`title = 'Interstellar'`

MOUNT (INITIAL RENDER)
COMMIT
BROWSER PAINT
EFFECT ✨

`<MovieDetails />`

If an effect sets state, an **additional render** will be required

```
document.title = `${title} ${
  userRating && `(Rated ${userRating} 🌟)`
}`;
```

`title = 'Interstellar Wars'`

`title` CHANGES
RE-RENDER
COMMIT
LAYOUT EFFECT
BROWSER PAINT
EFFECT ✨

Another type of effect that is **very rarely** necessary (`useLayoutEffect`)

`[title, userRating]`

```
document.title = `${title} ${
  userRating && `(Rated ${userRating} 🌟)`
}`;
```

UNMOUNT

Actually, there are even two more holds in this timeline but, we will talk about these mystery steps by the end of the section.