HEY THERE!

```
// whoami.component.js
import { useState } from 'react'

const [name] = useState('Lorenzo Pieri')
const [age, setAge] = useState(32)
const [job, setJob] = useState('Currently @ AVR Tech')

console.log('Hi everyone! 👋')
```

```
// whoami.component.js
import { useState } from 'react'

const [name] = useState('Lorenzo Pieri')
const [age, setAge] = useState(32)
const [job, setJob] = useState('Currently @ AVR Tech')

console.log('Hi everyone! 👋')
```

```
1 // talk.component.js
2 import { useEffect } from 'react'
3
4 const title = 'useEffect was not very effective!'
5
6 console.log(`Let's talk about ${title}`)
```

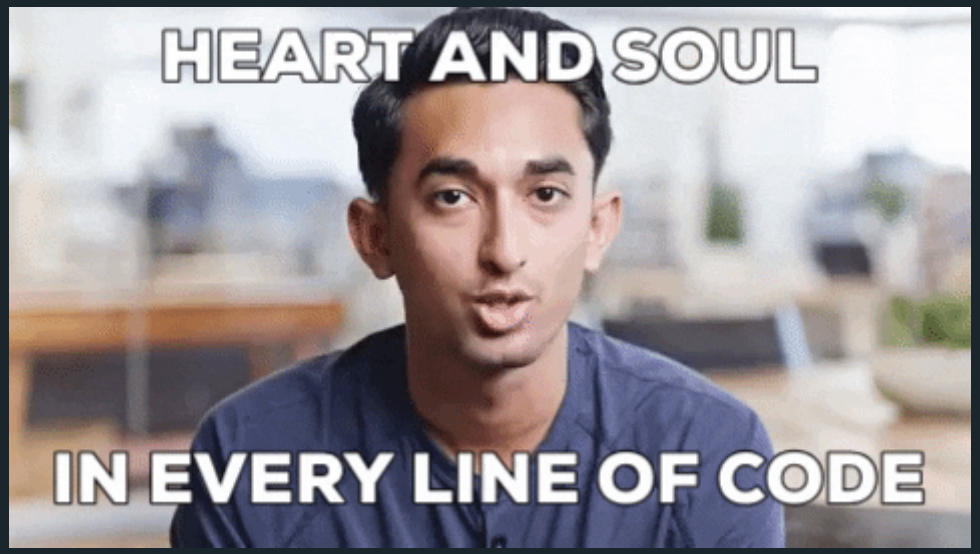

FETCH IN USEEFFECT
Wild
FUNCTIONAL COMPONENT
appeared!
imgflip.com


PROGRAMMER
used USEREF !
It's not very effective...
imgflip.com
SpankASJW

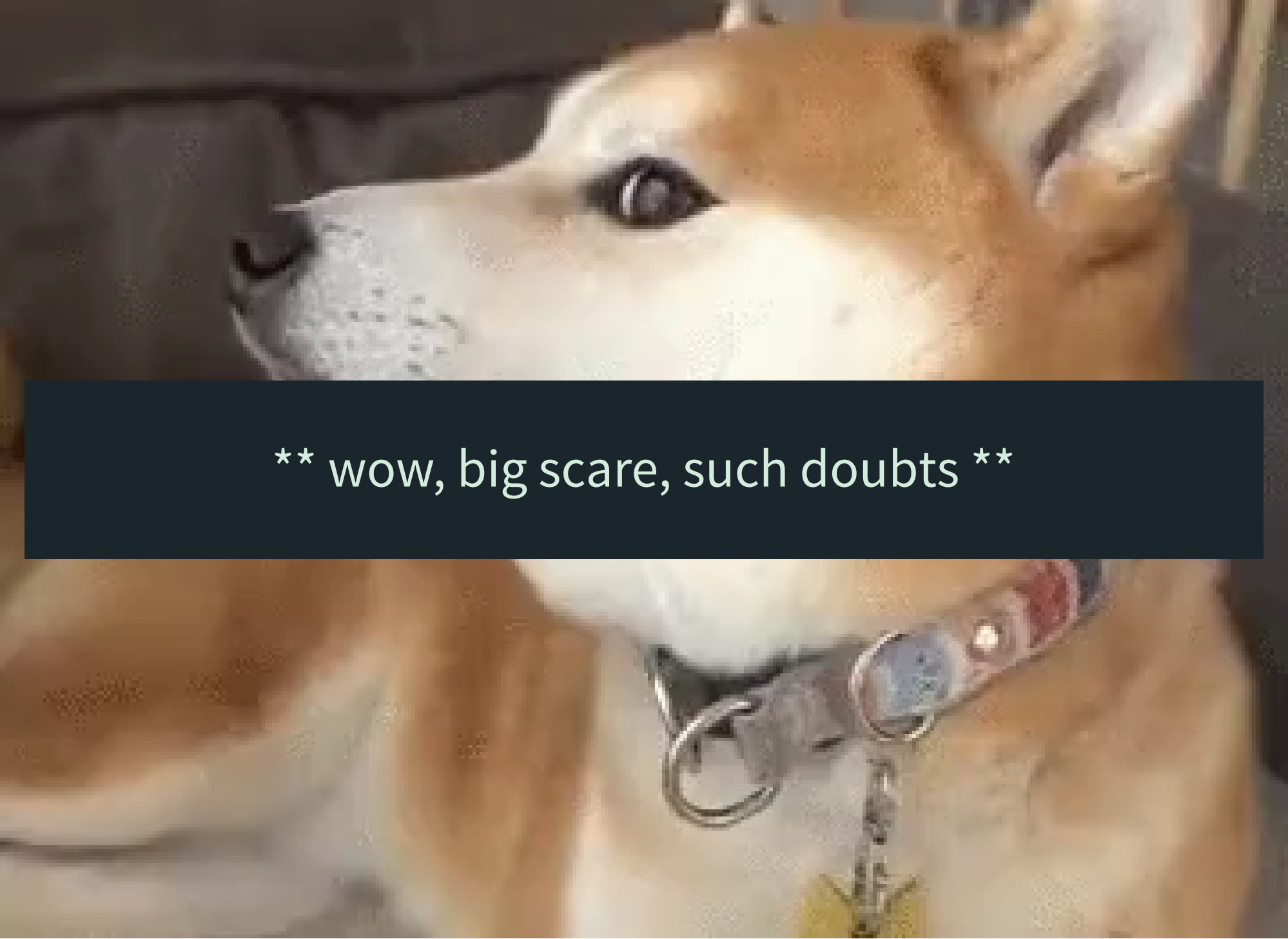
NOW,

Are you a React dev? 🙌

Are you doing it properly?



Do you know React? 🤔


** wow, big scare, such doubts **

Don't worry! React is still React and you probably used it well already

But, if you NEVER used React before

REACT IS

REACT IS

A JavaScript library for **building user interfaces**

WITH REACT YOU CAN

WITH REACT YOU CAN

 build dynamic web apps

WITH REACT YOU CAN

 build dynamic web apps

 develop reusable components

WITH REACT YOU CAN

 build dynamic web apps

 develop reusable components

 brag about the fact that you don't use Angular

WITH REACT YOU CAN

 build dynamic web apps

 develop reusable components

 brag about the fact that you don't use Angular

 use the latest javascript features

WITH REACT YOU CAN

👨‍💻 build dynamic web apps

⚒️ develop reusable components

💪 brag about the fact that you don't use Angular

🔥 use the latest javascript features

and many other things...

BUT MOST IMPORTANTLY

YOU WILL EITHER LOVE OR LOVE REACT HOOKS

BUT WHAT'S A HOOK, REALLY?

HOOKS ARE FUNCTIONS THAT LET YOU HOOK INTO REACT STATE AND LIFECYCLE FEATURES

What before was done with classes

```
class Talk extends React.Component {
  constructor(props) {}

  componentDidMount() {}
  componentWillUnmount() {}

  render() {
    return <h1>Hello, world!</h1>
  }
}
```

Now we can do functional!

```
import { useState } from 'react'

function Talk() {
  const [speaker, setSpeaker] = useState('not yet defined')

  return (
    <>
      <p>The speaker today is {speaker}</p>
      <button onClick={() => setSpeaker('Lorenzo')}>
        Click here
      </button>
    </>
  )
}
```

And manage our state with **hooks**

```
1  import { useState } from 'react'
2
3  function Talk() {
4   const [speaker, setSpeaker] = useState('not yet defined')
5
6   return (
7    <>
8      <p>The speaker today is {speaker}</p>
9      <button onClick={() => setSpeaker('Lorenzo')}>
10       Click here
11     </button>
12   </>
13  )
14 }
15
```

Hooks like **useState** trigger a rerender of the component every time the **state** changes

```
1  import { useState } from 'react'
2
3  function Talk() {
4   const [speaker, setSpeaker] = useState('not yet defined')
5
6   return (
7    <>
8      <p>The speaker today is {speaker}</p>
9      <button onClick={() => setSpeaker('Lorenzo')}>
10       Click here
11     </button>
12   </>
13  )
14 }
```

To do this, **React** uses something called **virtual DOM** along with a diffing algorithm that serves for what is the main mechanism of sync between UI and states

RECONCILIATION

There are quite a few hooks that we can use to play around with React and build our apps

- useState, to manage our state and renders
- useEffect, to take care of **side-effects**
- useRef, to play with the real DOM

and many more...

and yes, hooks are functions and a function is a hook if it starts with the word **use**

we can also make **our own hooks**, and the same rules apply:

- must begin with use

- must use at least one react hook

Now that we are React professionals

Let's dive right into **useEffect**

useEffect has:

```
1  import { useEffect } from 'react'
2
3  function Talk() {
4
5    useEffect(() => {
6
7      /**
8      your side effects managed by useEffect
9      */
10
11     return () => ... // cleanup function
12
13   }, []) /** dependency array */
14
15   return <p>Hello Fullstack Florence!</p>
```

A badly configured **useEffect** can lead to weird things

```
1 import { useEffect } from 'react'
2
3 function Talk() {
4   useEffect(() => {
5     // omitted for brevity
6   }) /** no dependency array */
7
8   return <p>Hello Fullstack Florence!
9 }
```

Our minds are still a little bit intertwined with the previous **class way** of dealing with state and **life cycle** of the components

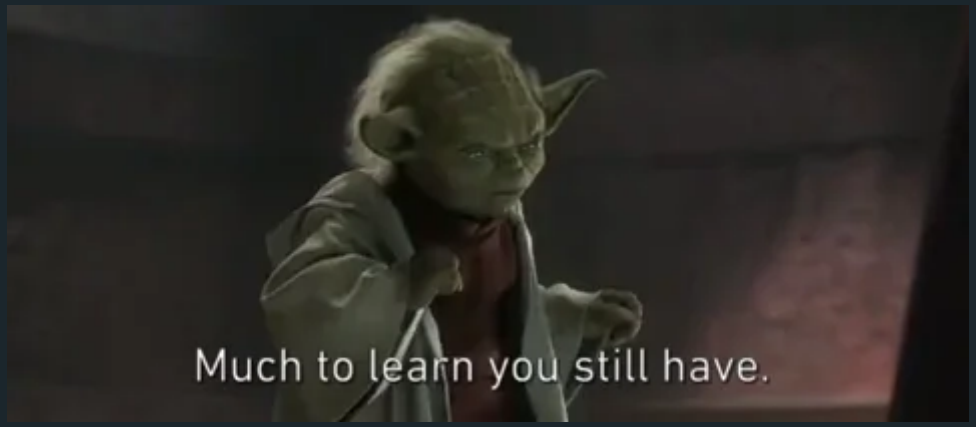
```
import { useEffect } from 'react'

function Talk() {
  useEffect(() => {
    // ...omitted for brevity
    return () => ... // componentWillUnmount
  }, []) /** empty array -> componentDidMount */

  return <p>Hello Fullstack Florence!</p>
}
```

That we forget the most important lesson about Hooks.

Hooks are not **classes**

So what should we do? We should rewire our thinking in React.

Unlearn, you must.



Classes come with **life cycles** and **state**

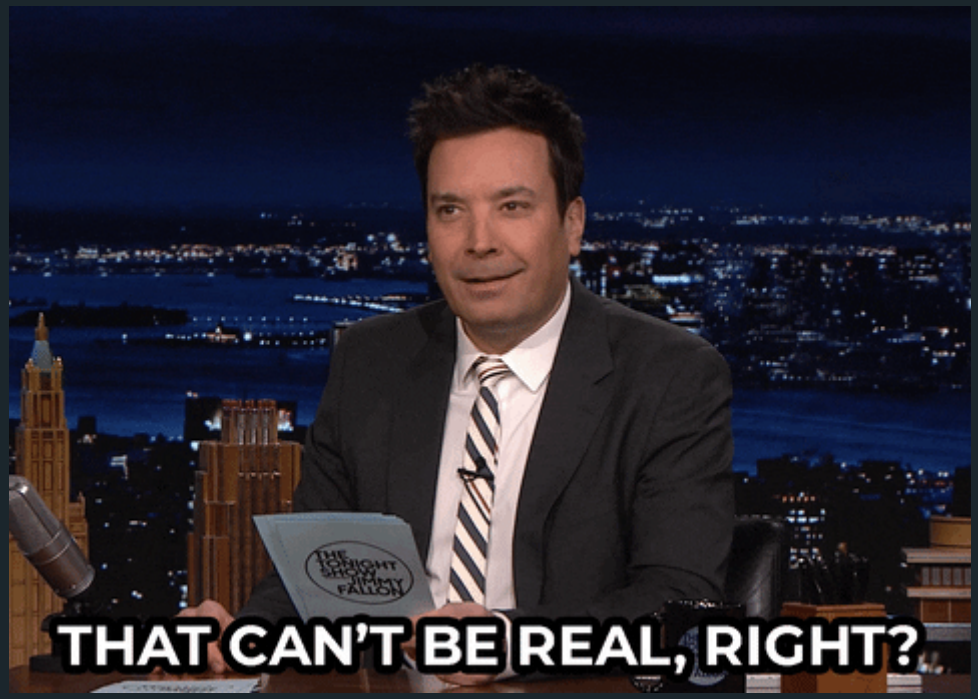
- componentDidMount
- componentWillUnmount
- componentDidUpdate

Functional Components are kind of **stateless**

but they can become stateful thanks to **hooks**

and hooks are probably easier to read and write than classes are

Now, hold on, where's the catch?!



No catch, just **some rules**

1. only call hooks **outside** of
    - loops
    - conditions
    - nested functions
2. only call hooks **inside**
    - functional components
    - custom hooks

```
function Stack({ isReact }) {

  // 🚫 Don't do this
  if(isReact){
    useEffect(() => {
      // do something
    }, [])
  }

  // ✅ Do this
  useEffect(() => {
    if(isReact){
      // do something
    }
  }, [])
```

Rules for hooks are of the uttermost importance

Breaking out of React will only result in unexpected behavior from your app

You can install **ESLint** to make sure you don't go against them

THAT'S IT

So, let's finally start

React 18 is out a few months now and everyone is having problems with it

THE NEW RELEASE BROUGHT

1. **Concurrent rendering**, which is a technicality
2. **Automatic Batching**, which is a technicality
3. **Suspense**, which is not yet production ready
4. Changes to the way you create the root node for React
5. New hooks
6. An updated strict mode

LET'S FOCUS ON THE UPDATED STRICT MODE

Strict Mode in React comes out of the box if you start your app with a CLI utility like **CRA** or **Vite**

What people tend to forget is the fact that **Strict Mode** can be opted-out at any time and is **development only**

You don't have it in your production app **after build**

So why is everyone crying about it?

Because useEffect now double renders the component. Better yet, useEffect **mounts**, **unmounts** and **remounts** the component

It does so on every component mount

It's not a bug
it's **intentional**

The application is actually **helping you** find weird component bugs before they get into production, which is actually a good thing

If your component behaves in unexpected ways

only because of this second rerender

❌ ATTENTION ❌

We should stop thinking about Functional components as if they were class components

We shouldn't be asking

"how can I run this only once?"

instead

"why is it not working properly on remount?"

```
function Stack({ isReact }) {

  // 🚫 This is not componentDidMount
  if(isReact){
    useEffect(() => {
      // do something
    }, [])
  }

  return <p>Hello Fullstack Florence!</p>
}
```

If codesandbox.io doesn't work

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

export default function App() {
return (
  <div className="App">
  <Talk />
  <DoubleRenderThis />
  </div>
);
}

function Talk() {
const [speaker, setSpeaker] = useState("not yet defined");

return (
```

Let's remember

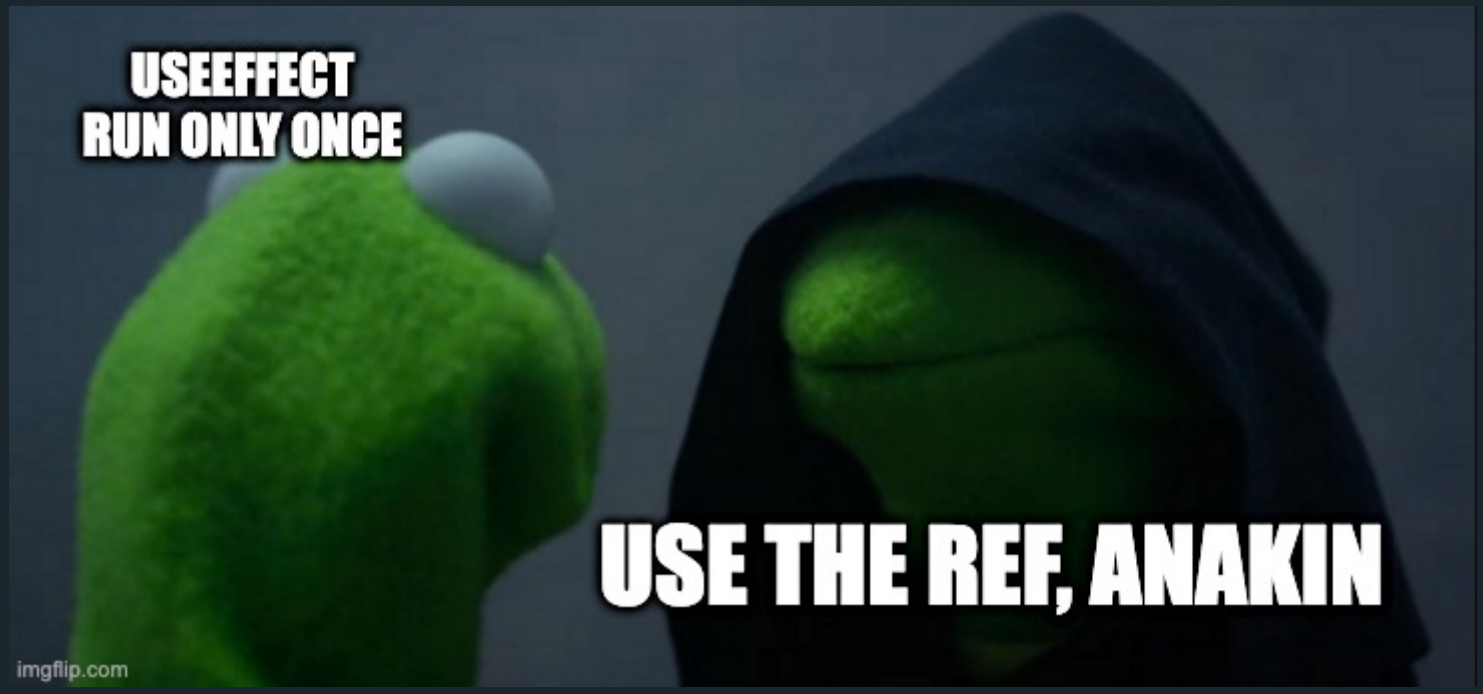
- double rendering happens only in development
- double rendering happens only in strict mode
- double rendering **is my friend**


we need to stop this double rendering inside useEffect
useRef weird trick?
write about it twitter?
why don't we just work WITH react instead of working against react?
imgflip.com

You get the point

And yet, you want to stop React from doing that in Development, for no particular reason

There are solutions



Solution 1, **useRef**

```
function RenderThisOnce() {
  const isUnmountedRef = useRef(false);

  useEffect(() => {

    if (isUnmountedRef.current) {
    console.log("mounted only once");
    }

    isUnmountedRef.current = true;

    return () => {
    console.log("unmounted only once");
    };
  }, [isUnmountedRef]);
```

Solution 2, **delete StrictMode**

```
1  import { StrictMode } from "react";
2  import { createRoot } from "react-dom/client";
3
4  import App from "./App";
5
6  const rootElement = document.getElementById("root");
7  const root = createRoot(rootElement);
8
9  root.render(
10   <StrictMode>
11     <App />
12   </StrictMode>
13 );
```

```
1  import { createRoot } from "react-dom/client";
2
3  import App from "./App";
4
5  const rootElement = document.getElementById("root");
6  const root = createRoot(rootElement);
7
8  root.render(
9    <App />
10 );
```

The solutions proposed are just escape hatches

So let's go back a bit and ask ourselves, again

Why are we escaping this helping mechanism?

"But I don't want to **fetch twice**!"

Fair enough. Why are you doing it in useEffect than?

"That's where it's **supposed** to be done!"

Also good point. Why are you building your own fetch calls tho?

Wake up!

React is mainly a UI library

Want to fetch inside useEffect, only on mount, only once?

Build your own caching mechanism

Build your own stale-while-revalidate technique

Build your own error handling and loading

That's a lot of stuff to build, so why would you?

WE ALREADY INVENTED THE WHEEL, JUST USE IT

 React Query

〰️ React SWR

 RTK Query

This solutions come with what you need out of the box, no need to reinvent anything and they are production ready and open source

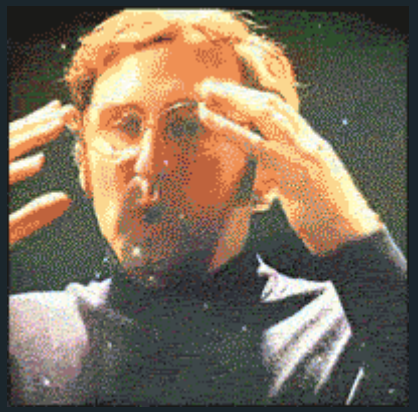
Will this stop the double rendering with useEffect?

No, it will not, but, you don't need that

HTTP requests will be cached and canceled by those solutions, hence you will have no problems with rerendering components

fun fact

page changes by users trigger the same mechanism so you are actually virtualizing real world usages with StrictMode



and yet, you want to build your own fetching mechanism

```
1  import { useState, useEffect } from "react"
2  import axios from "axios"
3
4  function useFetch(url) {
5    const [data, setData] = useState(null)
6    const [loading, setLoading] = useState(null)
7    const [error, setError] = useState(null)
8
9    useEffect(() => {
10     setLoading("loading...")
11     setData(null)
12     setError(null)
13     const source = axios.CancelToken.source()
14     axios
15       .get(url, { cancelToken: source.token })
```

There it is, a bad copy of React Query

just use react-query

Nice resources to check out online!

🔗 Fetch API and AbortController

🔗 A complete guide to useEffect

🔗 React custom useFetch implementation

Nice resources to check out online! part 2

🔗 How to fetch data with React Hooks

🔗 React useEffect double rendering fix and why you should stop crying about it

🔗 React 18 is out! What's good about it?

```
// final.component.js
import { useState } from 'react'

const [slides] = useState('https://404answernotfound.eu/use
const [blog] = useState('https://404answernotfound.eu/')
const [twitter] = useState('@404answnotfound')
const [github] = useState('404answernotfound')



console.log('Thank you Fullstack Florence! 👋')
```

```
// final.component.js
import { useState } from 'react'

const [slides] = useState('https://404answernotfound.eu/use
const [blog] = useState('https://404answernotfound.eu/')
const [twitter] = useState('@404answnotfound')
const [github] = useState('404answernotfound')



console.log('Thank you Fullstack Florence! 👋')
```

```
1  // final.component.js
2  import { useState } from 'react'
3
4  const [slides] = useState('https://404answernotfound.eu/use
5  const [blog] = useState('https://404answernotfound.eu/')
6  const [twitter] = useState('@404answnotfound')
7  const [github] = useState('404answernotfound')
8
9
10
11 console.log('Thank you Fullstack Florence! 👋')
```

```
1  // final.component.js
2  import { useState } from 'react'
3
4  const [slides] = useState('https://404answernotfound.eu/use
5  const [blog] = useState('https://404answernotfound.eu/')
6  const [twitter] = useState('@404answnotfound')
7  const [github] = useState('404answernotfound')
8
9
10
11 console.log('Thank you Fullstack Florence! 👋')
```

```
1  // final.component.js
2  import { useState } from 'react'
3
4  const [slides] = useState('https://404answernotfound.eu/use
5  const [blog] = useState('https://404answernotfound.eu/')
6  const [twitter] = useState('@404answnotfound')
7  const [github] = useState('404answernotfound')
8
9
10
11 console.log('Thank you Fullstack Florence! 👋')
```

THANK YOU!

🔗 404answernotfound Blog