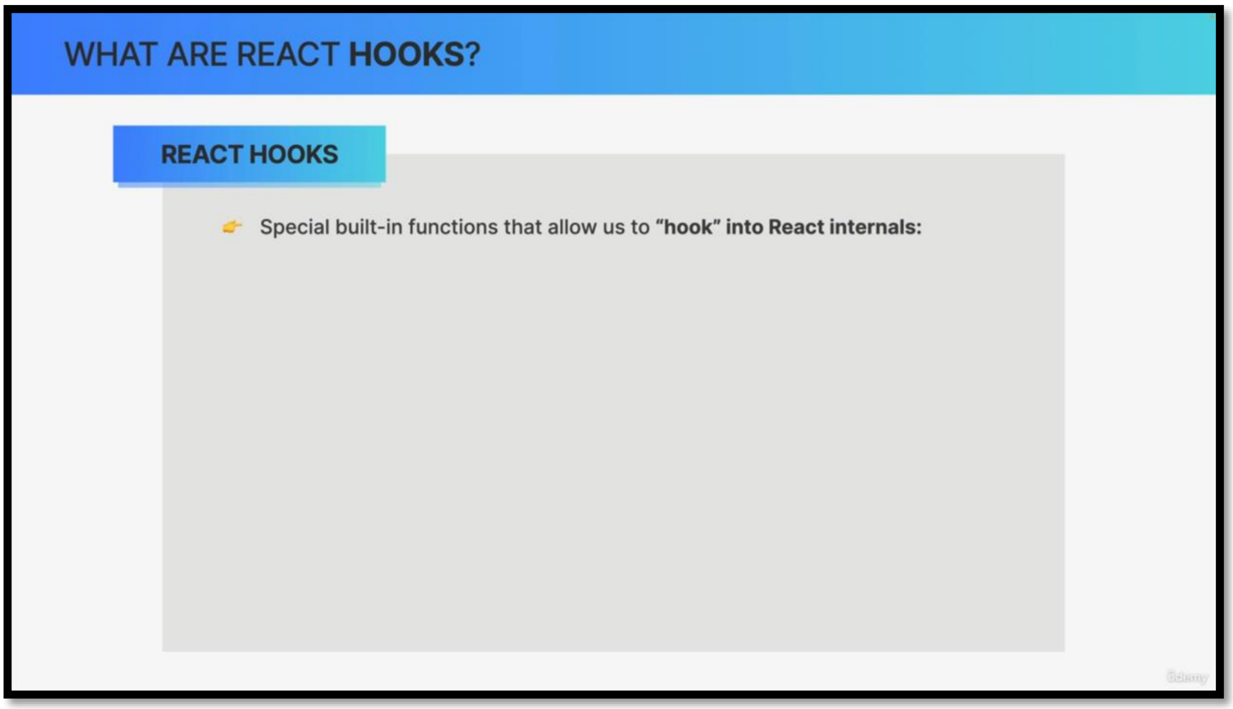
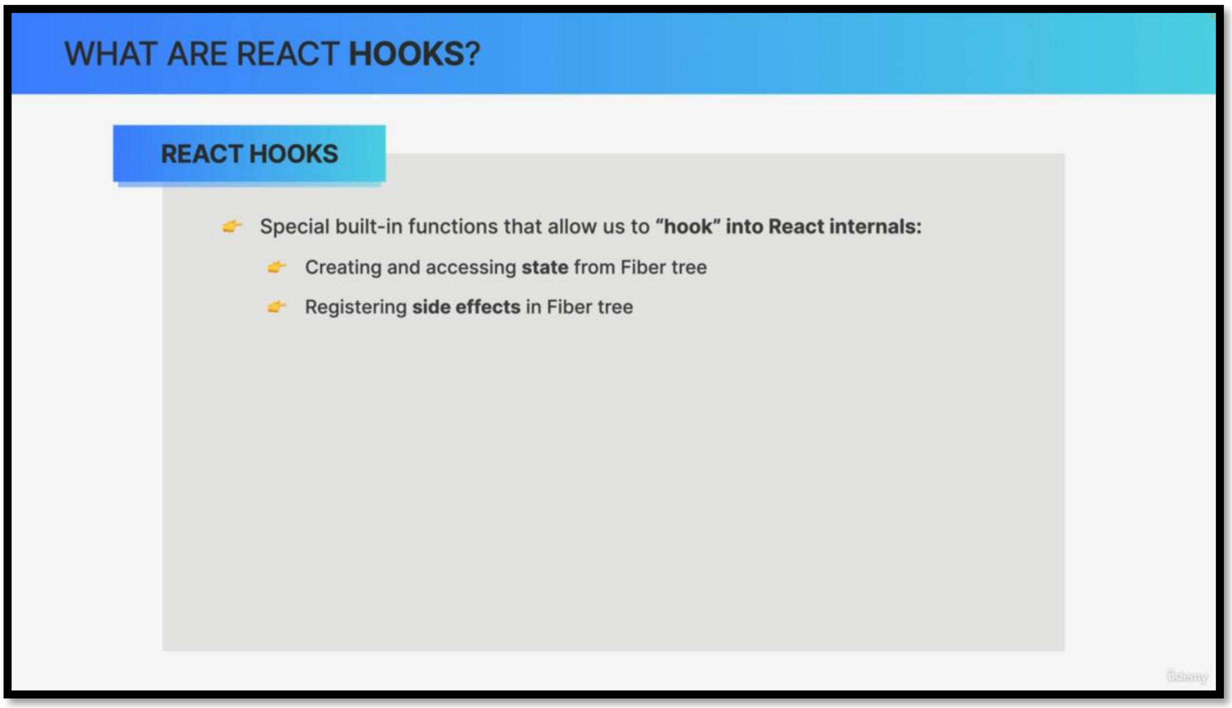


WHAT ARE REACT **HOOKS?**

REACT HOOKS

- Special built-in functions that allow us to **"hook" into React internals:**

React hooks are essentially special functions that are built into React and which allow us to basically hook into some of React's internal mechanisms.



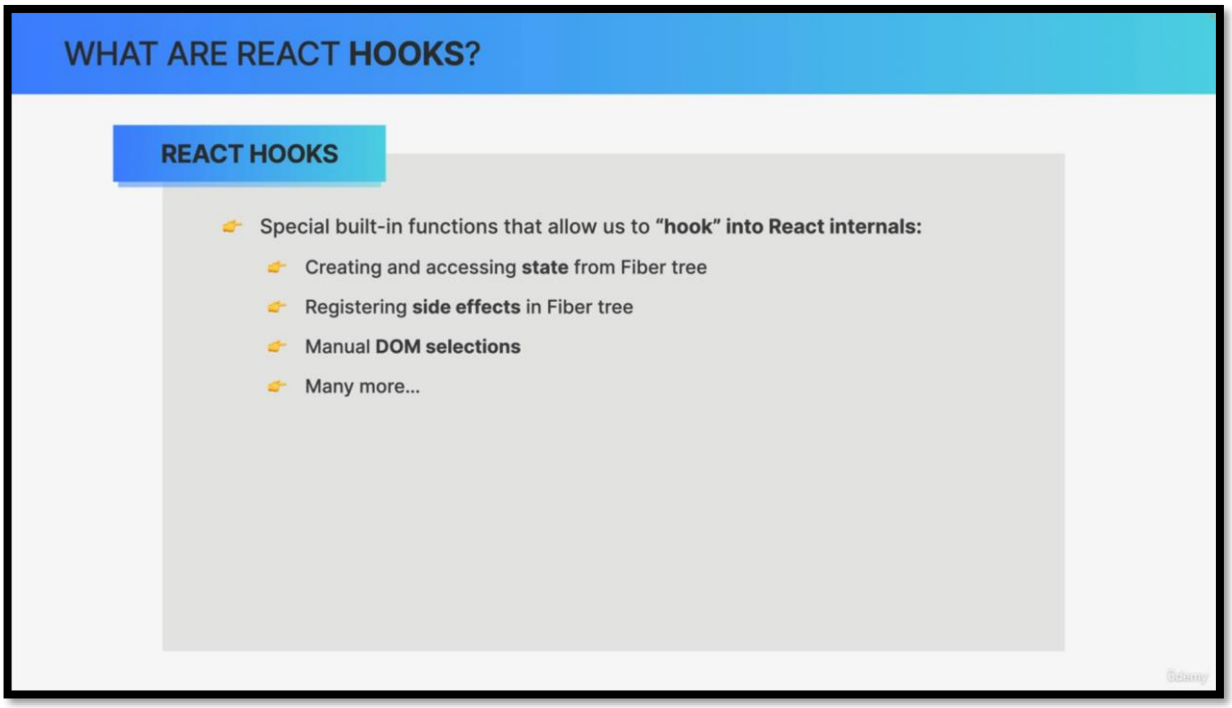
WHAT ARE REACT **HOOKS?**

REACT HOOKS

- Special built-in functions that allow us to **"hook" into React internals:**
  - Creating and accessing **state** from Fiber tree
  - Registering **side effects** in Fiber tree

Hooks are basically APIs that expose some internal React functionality, such as creating and accessing state from the fiber tree or registering side effects in the fiber tree.

The fiber tree is somewhere deep inside React and usually not accessible to us at all but using the useState or the useEffect hook, we can essentially hook into that internal mechanism.

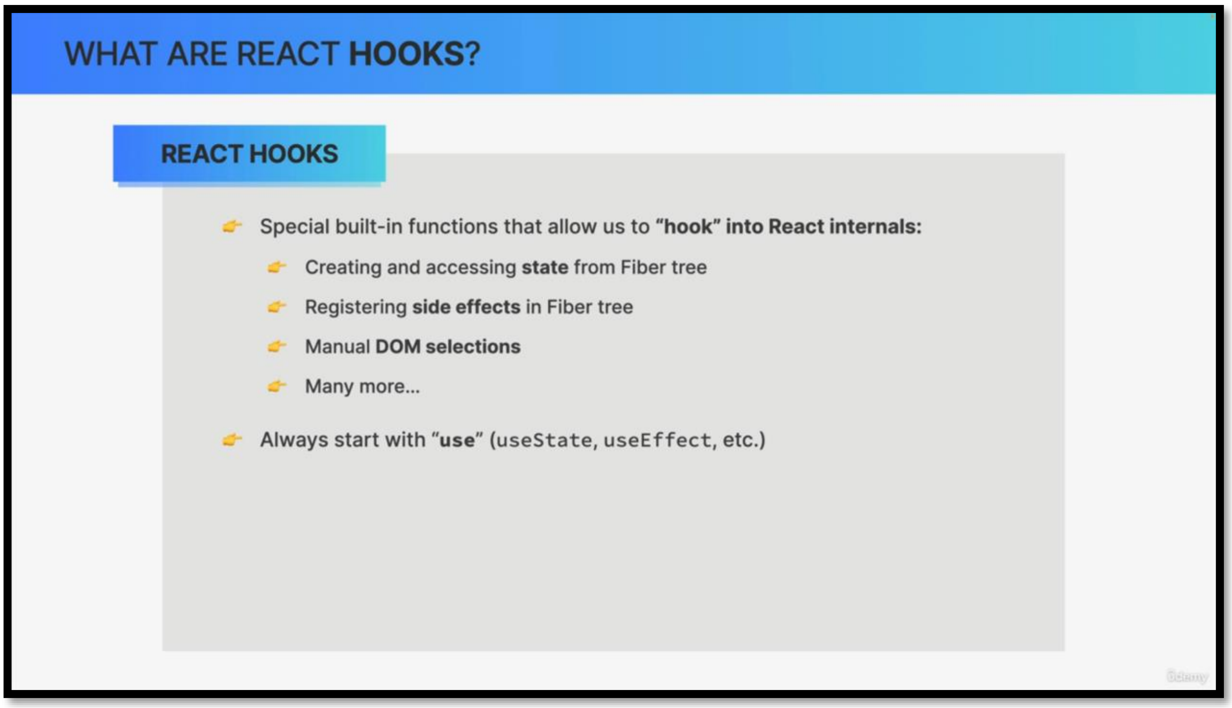


WHAT ARE REACT **HOOKS?**

REACT HOOKS

- Special built-in functions that allow us to **"hook" into React internals:**
  - Creating and accessing **state** from Fiber tree
  - Registering **side effects** in Fiber tree
  - Manual **DOM selections**
  - Many more...

Hooks also allow us to manually select in-store DOM notes access context and many other things.

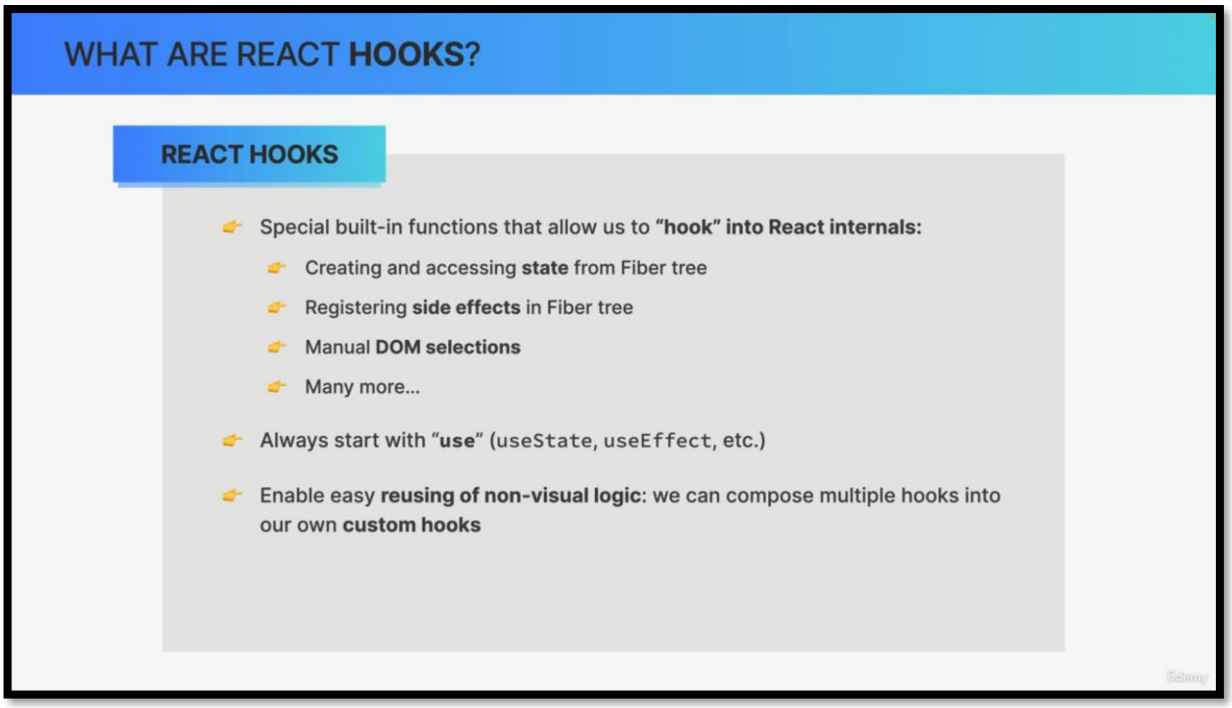


WHAT ARE REACT **HOOKS?**

REACT HOOKS

- Special built-in functions that allow us to **"hook" into React internals:**
  - Creating and accessing **state** from Fiber tree
  - Registering **side effects** in Fiber tree
  - Manual **DOM selections**
  - Many more...
- Always start with **"use"** (`useState`, `useEffect`, etc.)

All Hooks start with the word "use," in order to make it easy for us and for React to distinguish hooks from other regular functions.

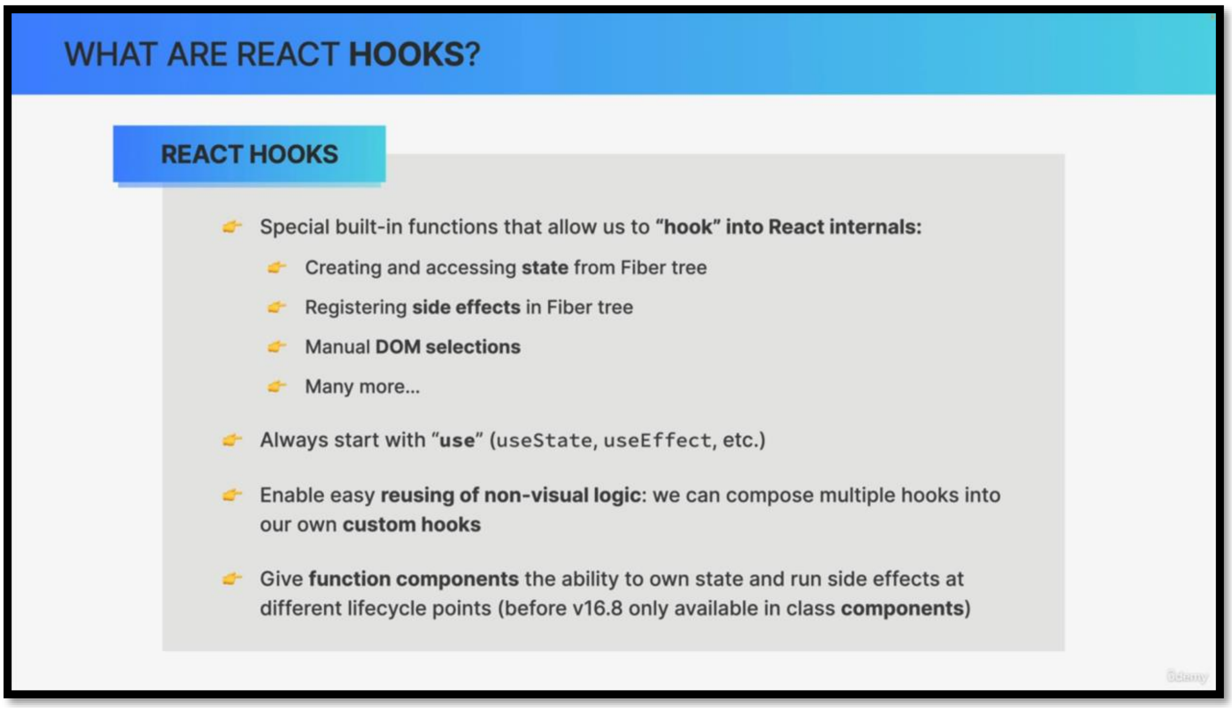


WHAT ARE REACT HOOKS?

REACT HOOKS

- Special built-in functions that allow us to **"hook" into React internals:**
  - Creating and accessing **state** from Fiber tree
  - Registering **side effects** in Fiber tree
  - Manual **DOM selections**
  - Many more...
- Always start with "**use**" (`useState`, `useEffect`, etc.)
- Enable easy **reusing of non-visual logic**: we can compose multiple hooks into our own **custom hooks**

We can even create our own so-called custom hooks, which will also start with the word "use." This is actually one of the greatest things about hooks in general, because custom hooks give us developers an easy way of reusing non-visual logic. So, logic that is not about the UI.



WHAT ARE REACT HOOKS?

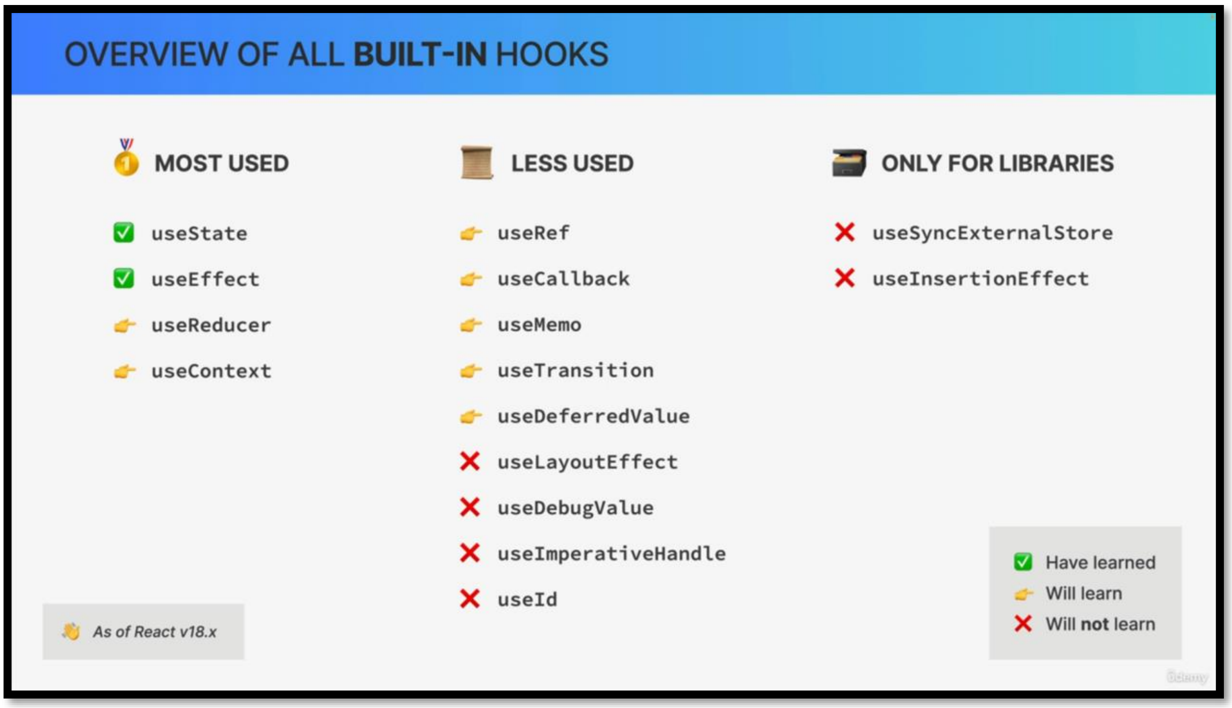
REACT HOOKS

- Special built-in functions that allow us to **"hook" into React internals:**
  - Creating and accessing **state** from Fiber tree
  - Registering **side effects** in Fiber tree
  - Manual **DOM selections**
  - Many more...
- Always start with "**use**" (`useState`, `useEffect`, etc.)
- Enable easy **reusing of non-visual logic**: we can compose multiple hooks into our own **custom hooks**
- Give **function components** the ability to own state and run side effects at different lifecycle points (before v16.8 only available in class **components**)

There was a time when we had to use components based on JavaScript classes if we wanted to give components state and access to the component life cycle.

However, this came with a few problems,which led the React team to introduce hooks.

With hooks, our components based on functions can have their own state and also run side effects. This was a huge step forward for React and made it even more popular than it already was.

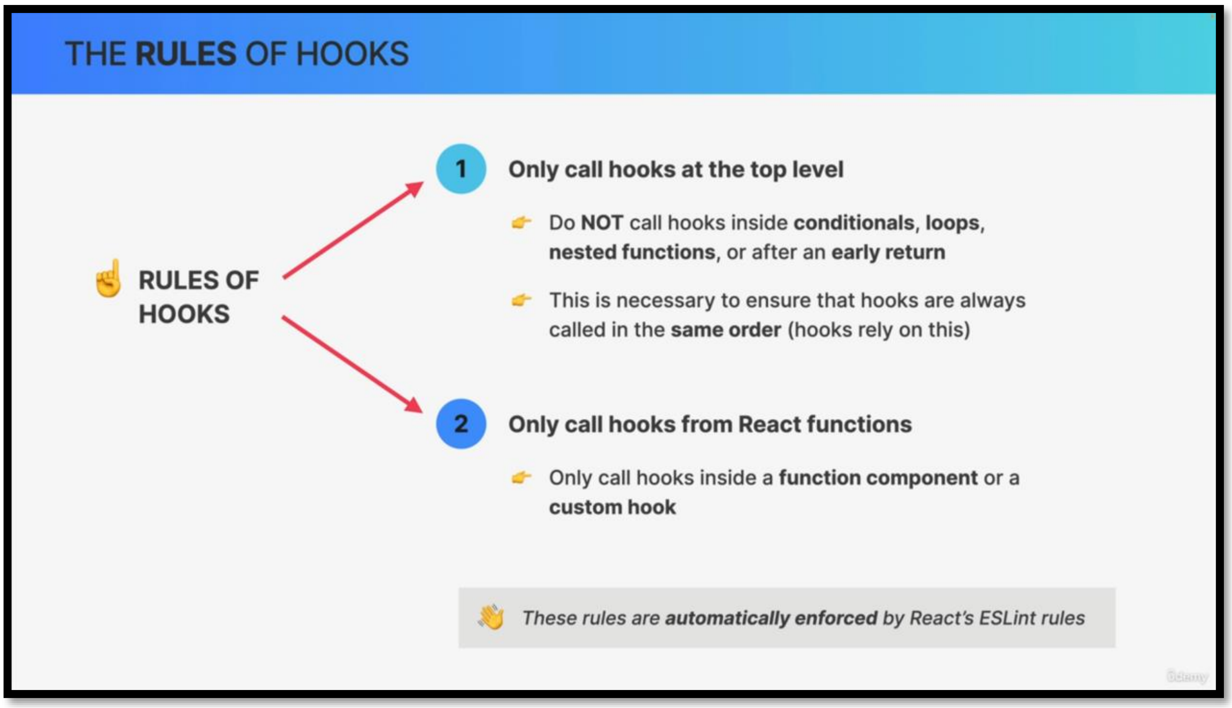


OVERVIEW OF ALL **BUILT-IN** HOOKS

MOST USED	LESS USED	ONLY FOR LIBRARIES
✓ `useState`	👉 `useRef`	✗ `useSyncExternalStore`
✓ `useEffect`	👉 `useCallback`	✗ `useInsertionEffect`
👉 `useReducer`	👉 `useMemo`	
👉 `useContext`	👉 `useTransition`	
	👉 `useDeferredValue`	
	✗ `useLayoutEffect`	
	✗ `useDebugValue`	
	✗ `useImperativeHandle`	
	✗ `useId`	

As of React v18.x

✓ Have learned
👉 Will learn
✗ Will **not** learn



THE **RULES** OF HOOKS

RULES OF HOOKS

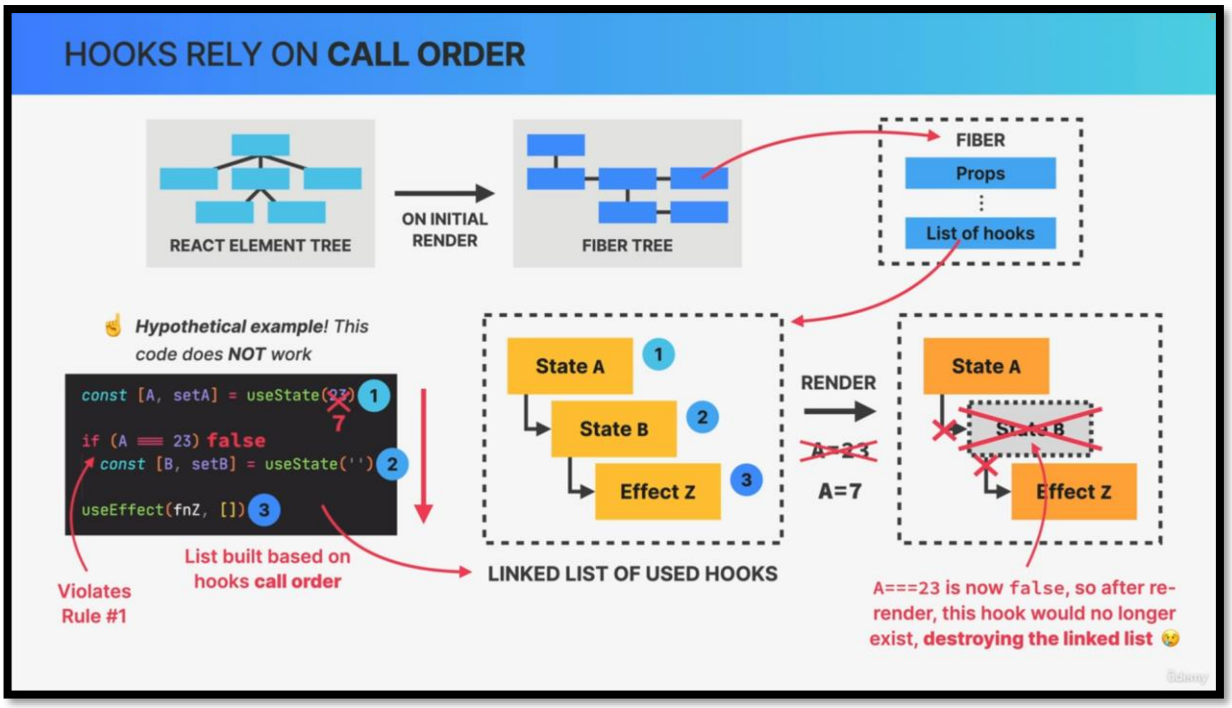
1 **Only call hooks at the top level**

- Do **NOT** call hooks inside **conditionals**, **loops**, **nested functions**, or after an **early return**
- This is necessary to ensure that hooks are always called in the **same order** (hooks rely on this)

2 **Only call hooks from React functions**

- Only call hooks inside a **function component** or a **custom hook**

*These rules are **automatically enforced** by React's ESLint rules*


HOOKS RELY ON CALL ORDER

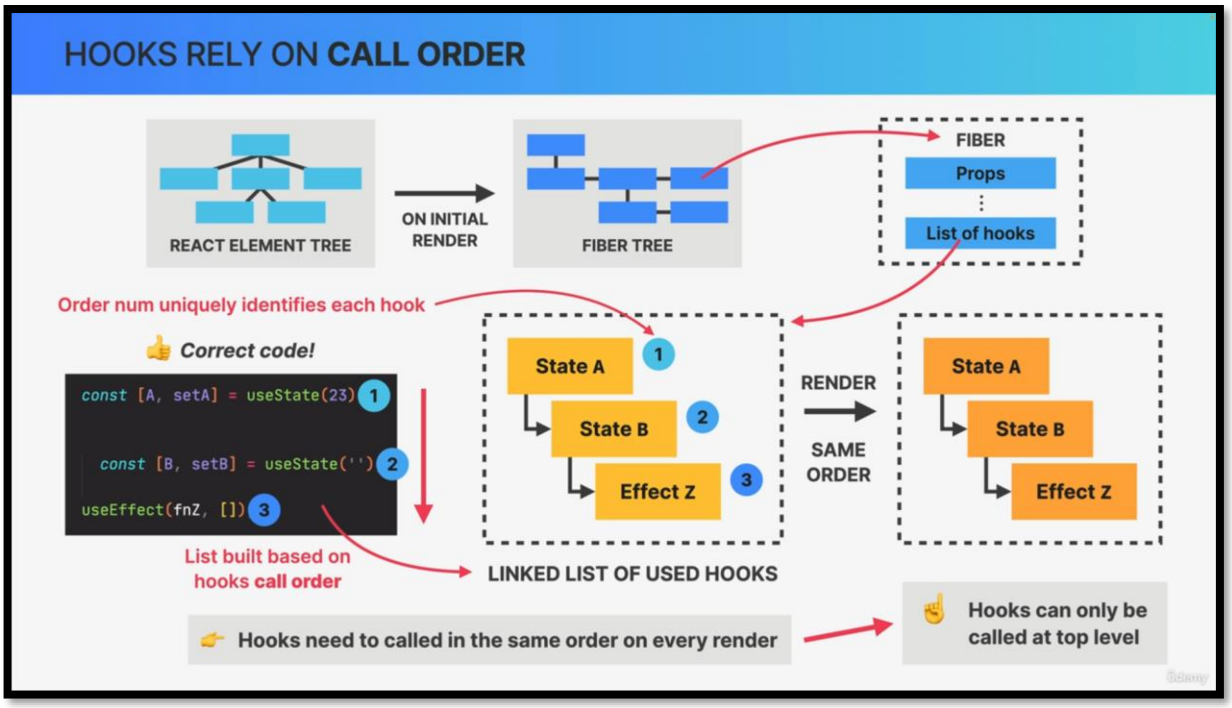
☝ *Hypothetical example! This code does NOT work*

```
const [A, setA] = useState(~~23~~ 7) 1

if (A === 23) false
  const [B, setB] = useState('') 2

useEffect(fnZ, []) 3
```

REACT ELEMENT TREE
ON INITIAL RENDER
FIBER TREE
FIBER
Props
List of hooks
State A
1
State B
2
Effect Z
3
RENDER
~~A=23~~
A=7
State A
~~State B~~
Effect Z
LINKED LIST OF USED HOOKS
List built based on hooks **call order**
Violates Rule #1
A===23 is now false, so after re-render, this hook would no longer exist, **destroying the linked list** 😢


HOOKS RELY ON CALL ORDER

👍 *Correct code!*

```
const [A, setA] = useState(23) 1

  const [B, setB] = useState('') 2

useEffect(fnZ, []) 3
```

👉 Hooks need to called in the same order on every render

☝ Hooks can only be called at top level