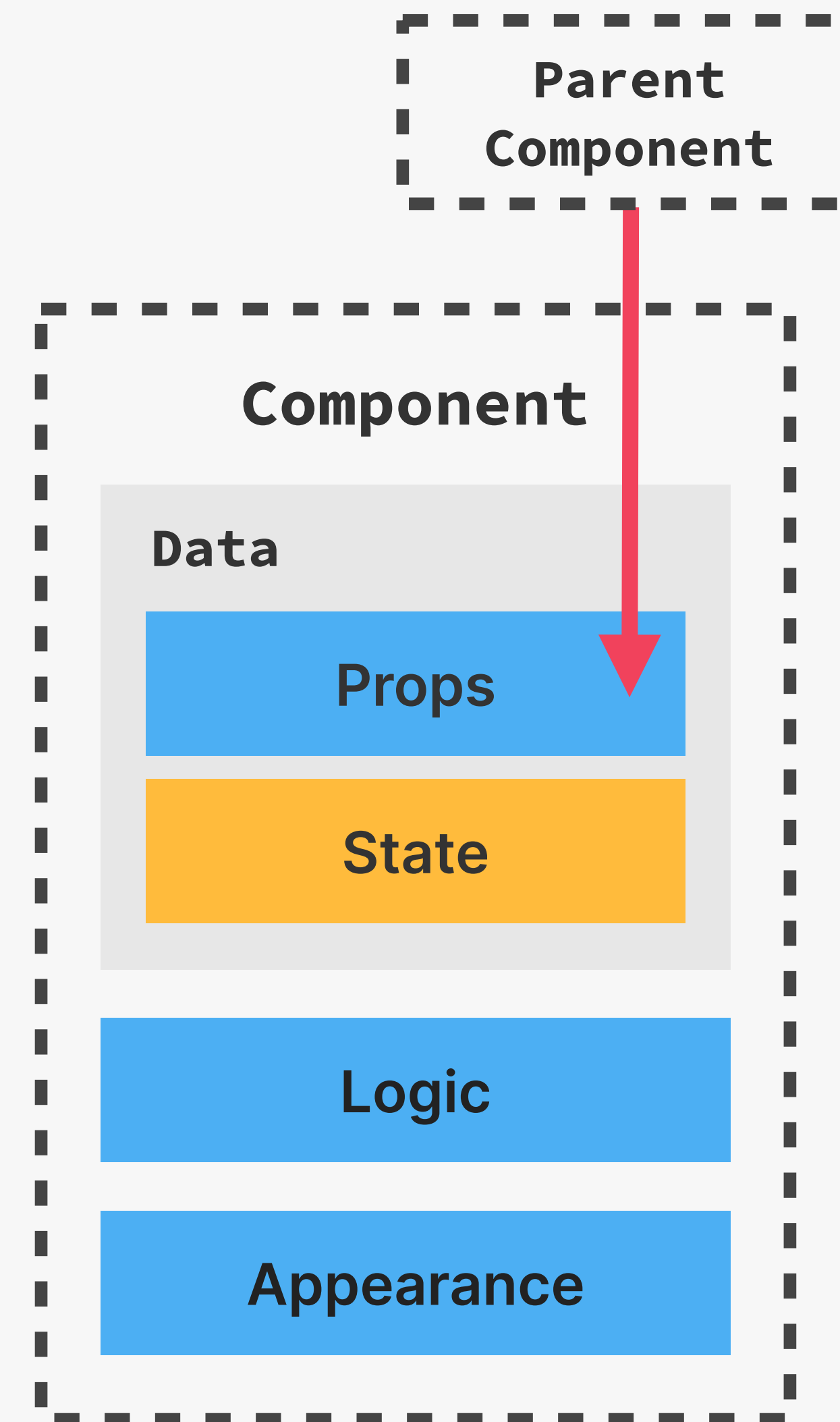


WHAT IS STATE?

STATE

👉 Data that a component **can hold over time**, necessary for information that it needs to **remember** throughout the app's lifecycle

👉 “Component’s memory”



WHAT IS STATE?

STATE

👉 Data that a component **can hold over time**, necessary for information that it needs to **remember** throughout the app's lifecycle

👉 “Component’s memory”



👉 “State variable” / “piece of state”: A single variable in a component (component state)

We use these terms interchangeably

Notifications

9+

Messages

9+

🔍 javascr

Overview

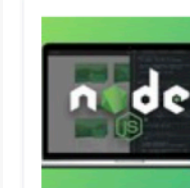
Q&A

Notes

Announcements

Shopping Cart

2 Courses in Cart



Node.js, Express, MongoDB & More: The Complete Bootcamp 2022
By Jonas Schmedtmann, Web Developer, Designer, and Teacher

€12.99
€84.99

Updated Recently

4.7 ★★★★★ (11465 ratings)

42 ore totali • 229 lectures • All Levels

[Remove](#) [Save for Later](#)



The Complete JavaScript Course 2022: From Zero to Expert!
By Jonas Schmedtmann, Web Developer, Designer, and Teacher

€12.99
€84.99

Bestseller

Updated Recently


4.7 ★★★★★ (137333 ratings)

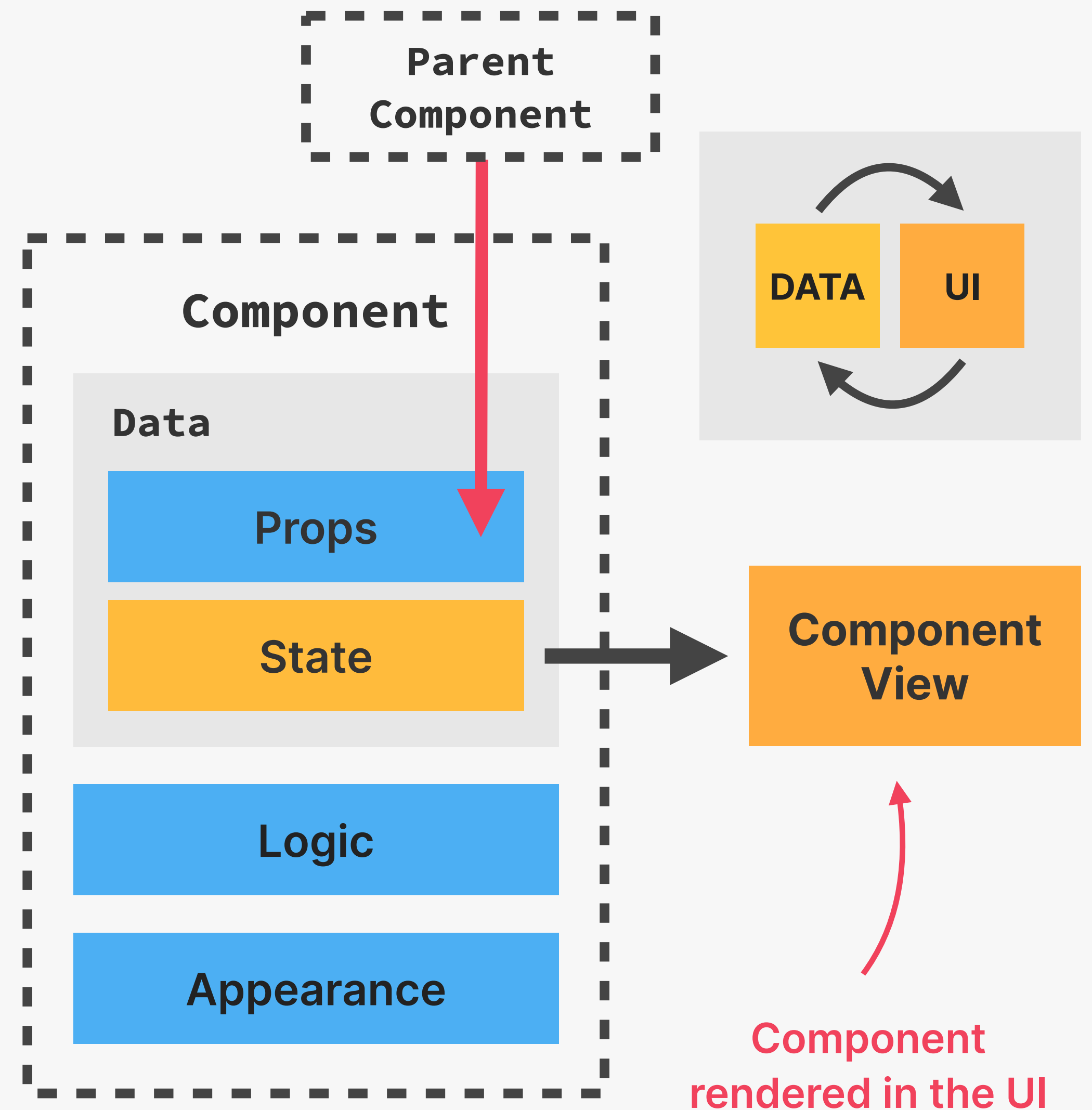
69 total hours • 320 lectures • All Levels

[Remove](#) [Save for Later](#)

WHAT IS STATE?

STATE

- 👉 Data that a component **can hold over time**, necessary for information that it needs to **remember** throughout the app's lifecycle
- 👉 “Component's memory” 
- 👉 **Component state**: Single local component variable (“Piece of state”, “state variable”)
- 👉 Updating **component state** triggers React to **re-render** the component



WHAT IS STATE?

STATE

👉 Data that a component **can hold over time**, necessary for information that it needs to **remember** throughout the app's lifecycle

👉 “Component’s memory”



👉 **Component state**: Single local component variable (“Piece of state”, “state variable”)

👉 Updating **component state** triggers React to **re-render the component**

STATE ALLOWS DEVELOPERS TO:

1

Update the component’s view (by re-rendering it)

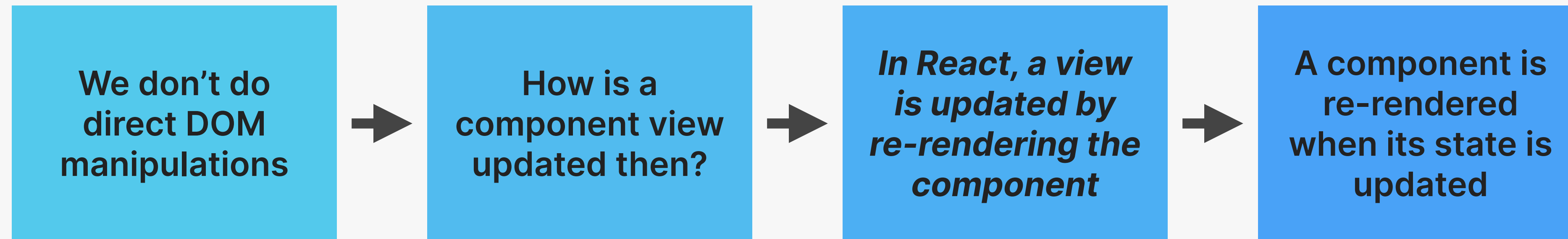
2

Persist local variables between renders



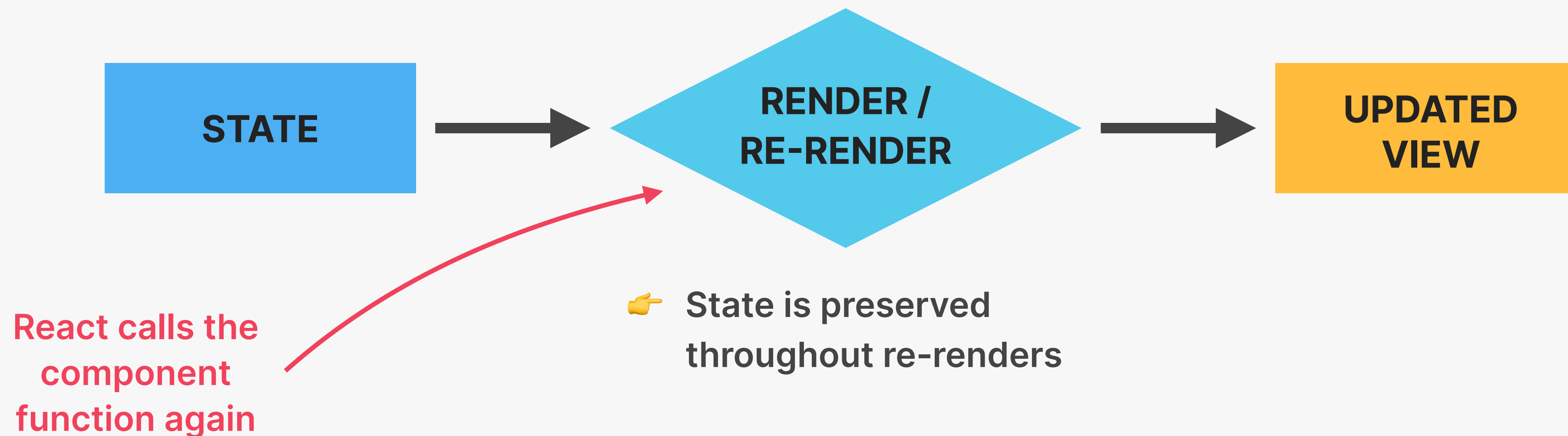
*State is a **tool**. Mastering state will unlock the power of React development*

THE MECHANICS OF STATE IN REACT

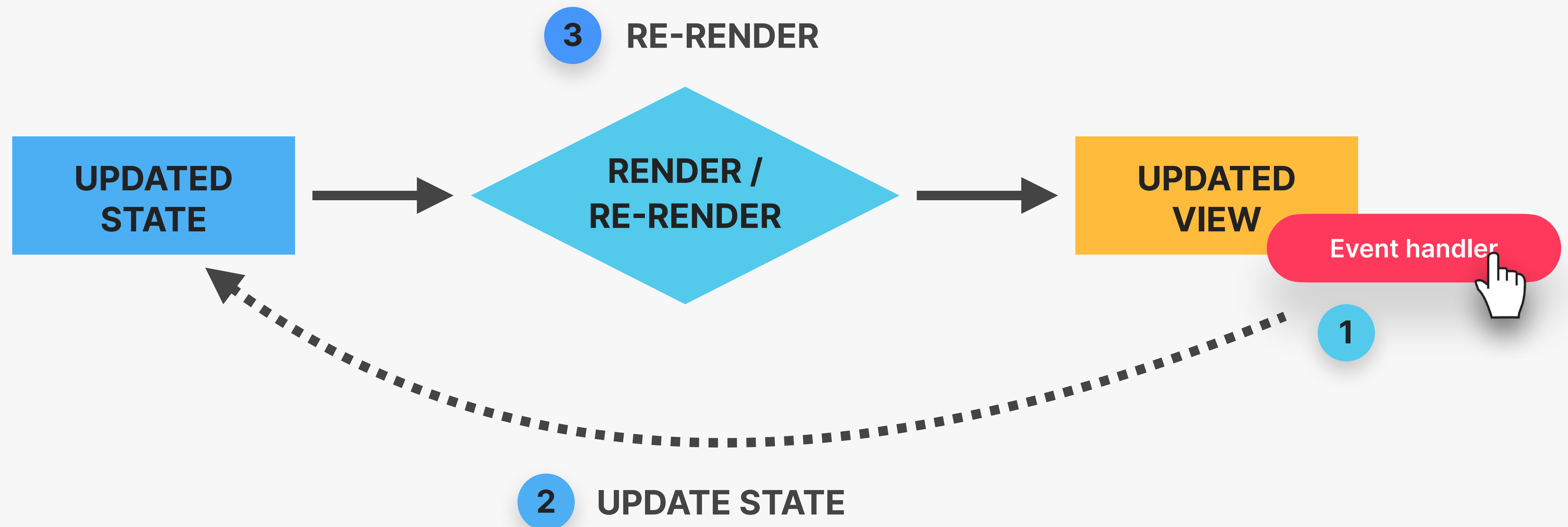
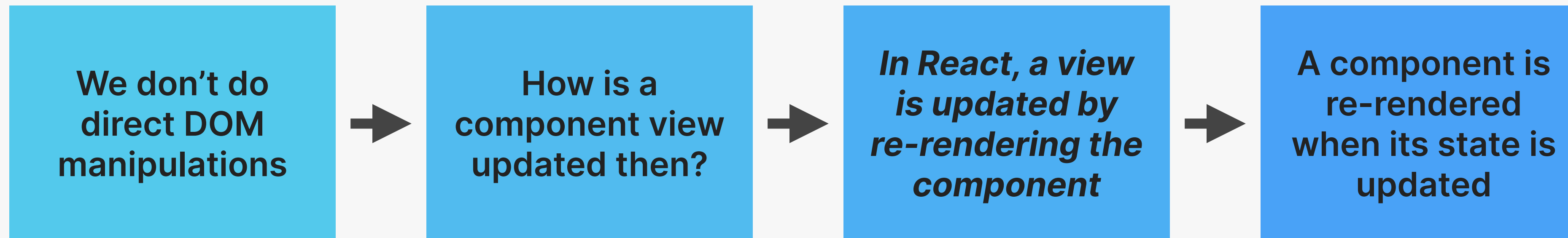


Because React is **declarative**

Important React principle

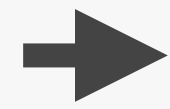


THE MECHANICS OF STATE IN REACT

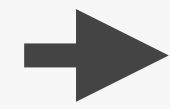


THE MECHANICS OF STATE IN REACT

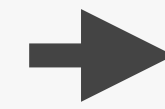
We don't do direct DOM manipulations



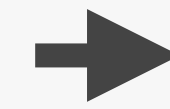
How is a component view updated then?



In React, a view is updated by re-rendering the component



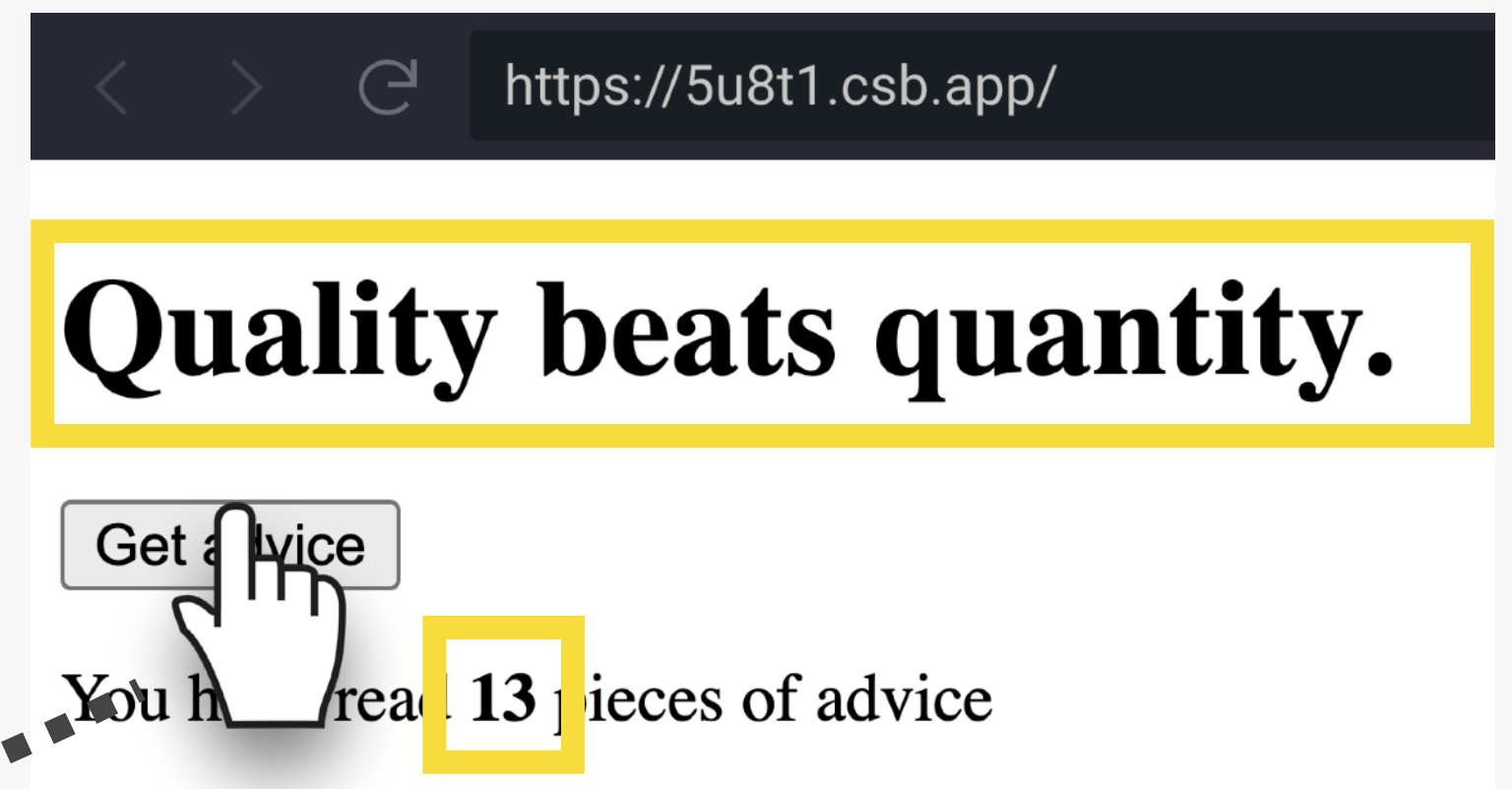
A component is re-rendered when its state is updated



So to update a view, we update state

```
const [advice, setAdvice] =  
  useState("Quality beats quantity.");  
const [countAdvice, setCountAdvice] =  
  useState(13);
```

RE-RENDER

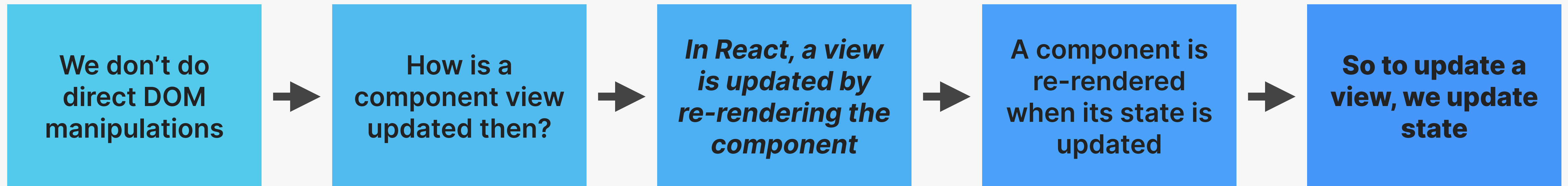


UPDATE STATE

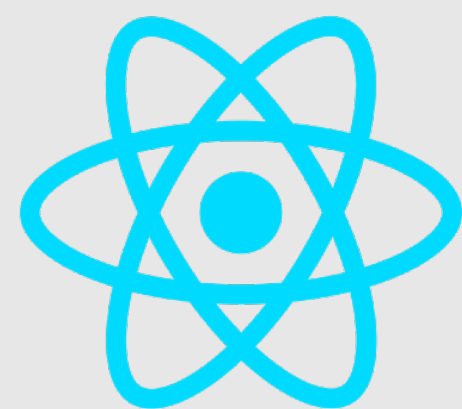


```
setAdvice(data.slip.advice);  
setCountAdvice((count) => count + 1);
```

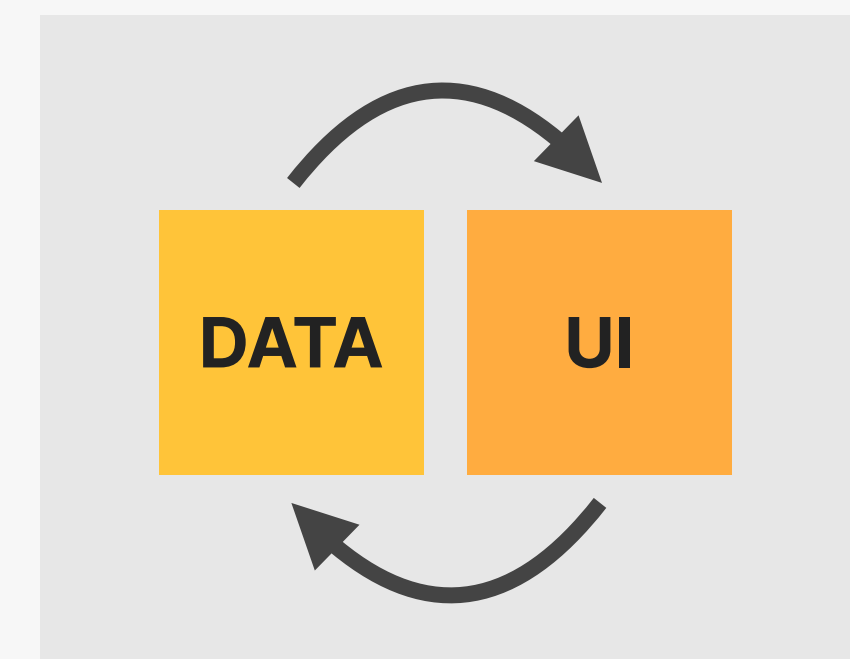
THE MECHANICS OF STATE IN REACT



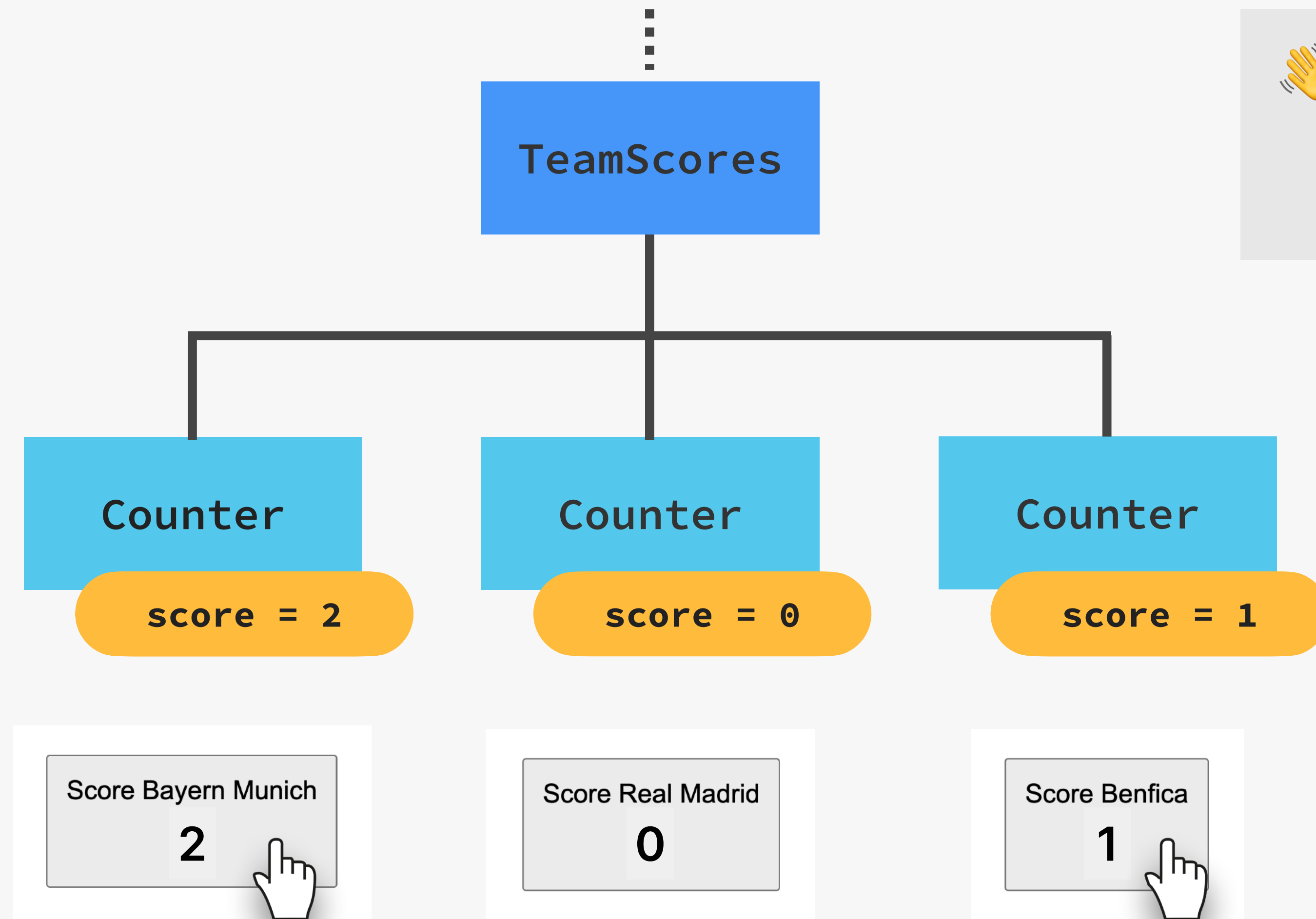
👉 *React is called "React" because...*



REACT *REACTS* TO STATE CHANGES
BY RE-RENDERING THE UI



ONE COMPONENT, ONE STATE



Each component has and manages **its own state**, no matter how many times we render the same component

Multiple *instances* of the Counter component

STATE VS. PROPS

STATE

- 👉 Internal data, owned by component
- 👉 Component “memory”
- 👉 Can be updated by the component itself
- 👉 Updating state causes component to re-render
- 👉 Used to make components interactive

```
function Question() {  
  const [upvotes, setUpvotes] = useState(0);  
  
  return (  
    <div>  
      { /* ... */ }  
      <Button upvotes={upvotes} bgColor="blue" />  
    </div>  
  );  
}
```

The diagram illustrates the flow of state from the parent component `Question()` to the child component `Button`. A blue box highlights `[upvotes, setUpvotes]` in the `Question()` function. A solid blue arrow points from this box to the `upvotes={upvotes}` prop in the `Button` component's JSX. A circular blue arrow next to the `setUpvotes` variable indicates a self-update cycle within the parent component.

```
function Button(upvotes, bgColor) {  
  const [hovered, setHovered] = useState(false);  
  
  return (  
    <div>  
      { /* ... */ }  
      <button  
        onMouseEnter={() => setHovered(true)}  
        onMouseLeave={() => setHovered(false)}  
        style={{ background: bgColor }}  
      >  
        {hovered ? "Upvote" : `👍 ${upvotes}`}  
      </button>  
    </div>  
  );  
}
```

The diagram illustrates the state flow within the `Button` component and its interaction with the parent. A blue box highlights the `upvotes, bgColor` props in the function signature. A solid blue arrow points from the `upvotes` prop to the `{upvotes}` interpolation in the button's text. A circular blue arrow next to `setHovered` indicates a self-update cycle. A dashed blue arrow points from the `setHovered` function back to the `upvotes` prop, indicating that a state change in the child can trigger a re-render of the parent.

PROPS

- 👉 External data, owned by parent component
- 👉 Similar to function parameters
- 👉 Read-only
- 👉 Receiving new props causes component to re-render. Usually when the parent's state has been updated
- 👉 Used by parent to configure child component (“settings”)