

# Things I learnt from the New React Docs

Debbie O'Brien

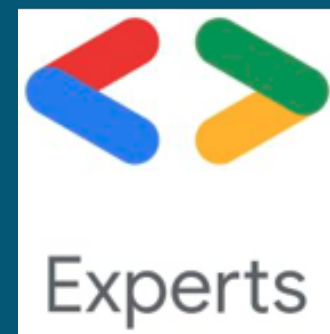
 @debs\_obrien



# Debbie O'Brien

Head Developer Advocate at Bit

Member of the React 18 Working Group



 @debs\_obrien

debbie.codes



```
};

export const ThemeContext = React.createContext(
  themes.dark // default value
);
```

### themed-button.js

```
import {ThemeContext} from './theme-context';

class ThemedButton extends React.Component {
  render() {
    let props = this.props;
    let theme = this.context;
    return (
      <button
        {...props}
        style={{backgroundColor: theme.background}}
      />
    );
  }
}

ThemedButton.contextType = ThemeContext;

export default ThemedButton;
```

INSTALLATION ▾

MAIN CONCEPTS ▾

ADVANCED GUIDES ▲

- Accessibility
- Code-Splitting
- Context**
- Error Boundaries
- Forwarding Refs
- Fragments
- Higher-Order Components



# The new React Docs

- Installation >
- Quick Start >
- Describing the UI
- Your First Component
- Importing and Exporting Components
- Writing Markup with JSX
- JavaScript in JSX with Curly Braces
- Passing Props to a Component
- Conditional Rendering**
- Rendering Lists
- Keeping Components Pure
- Adding Interactivity >
- Managing State >
- Escape Hatches >

## Try out some challenges

1. Show an icon for incomplete items with `? :` 2. Show the item importance with `&&` 3. Refactor a series of `? :` to `if` and `var`

### Challenge 1 of 3: Show an icon for incomplete items with `? :`

Use the conditional operator (`cond ? a : b`) to render a if `isPacked` isn't `true`.

App.js

Reset Fork

```
1 function Item({ name, isPacked }) {
2   return (
3     <li className="item">
4       {name} {isPacked && '✓'}
5     </li>
6   );
7 }
8
9 export default function PackingList() {
10  return (
11    <section>
12      <h1>Sally Ride's Packing List</h1>
13      <ul>
14        <Item
15          isPacked={true}
16          name="Space suit"
```

Show more

### Sally Ride's Packing List

- Space suit ✓
- Helmet with a golden leaf ✓
- Photo of Tam

Show solution

Next Challenge >

#### ON THIS PAGE

- Overview
- Conditionally returning JSX
- Conditionally returning nothing with `null`
- Conditionally including JSX
- Conditional (ternary) operator (`? :`)
- Logical AND operator (`&&`)
- Conditionally assigning JSX to a variable

#### Recap

Challenges

# React Component

A FUNCTION WITH A CAPITAL LETTER

## Step 2: Define the function

With `function Profile() { }` you define a JavaScript function with the name `Profile`.

### 🗨 Gotcha

React components are regular JavaScript functions, but **their names must start with a capital letter** or they won't work!

[bit.ly/react-first-component](https://bit.ly/react-first-component)

# JSX

WITH CURLY BRACES YOU ARE ENTERING INTO JAVASCRIPT LAND

```
<ul style={  
  {  
    backgroundColor: 'black',  
    color: 'pink'  
  }  
>
```

The next time you see `{{` and `}}` in JSX, know that it's nothing more than an object inside the JSX curlies!

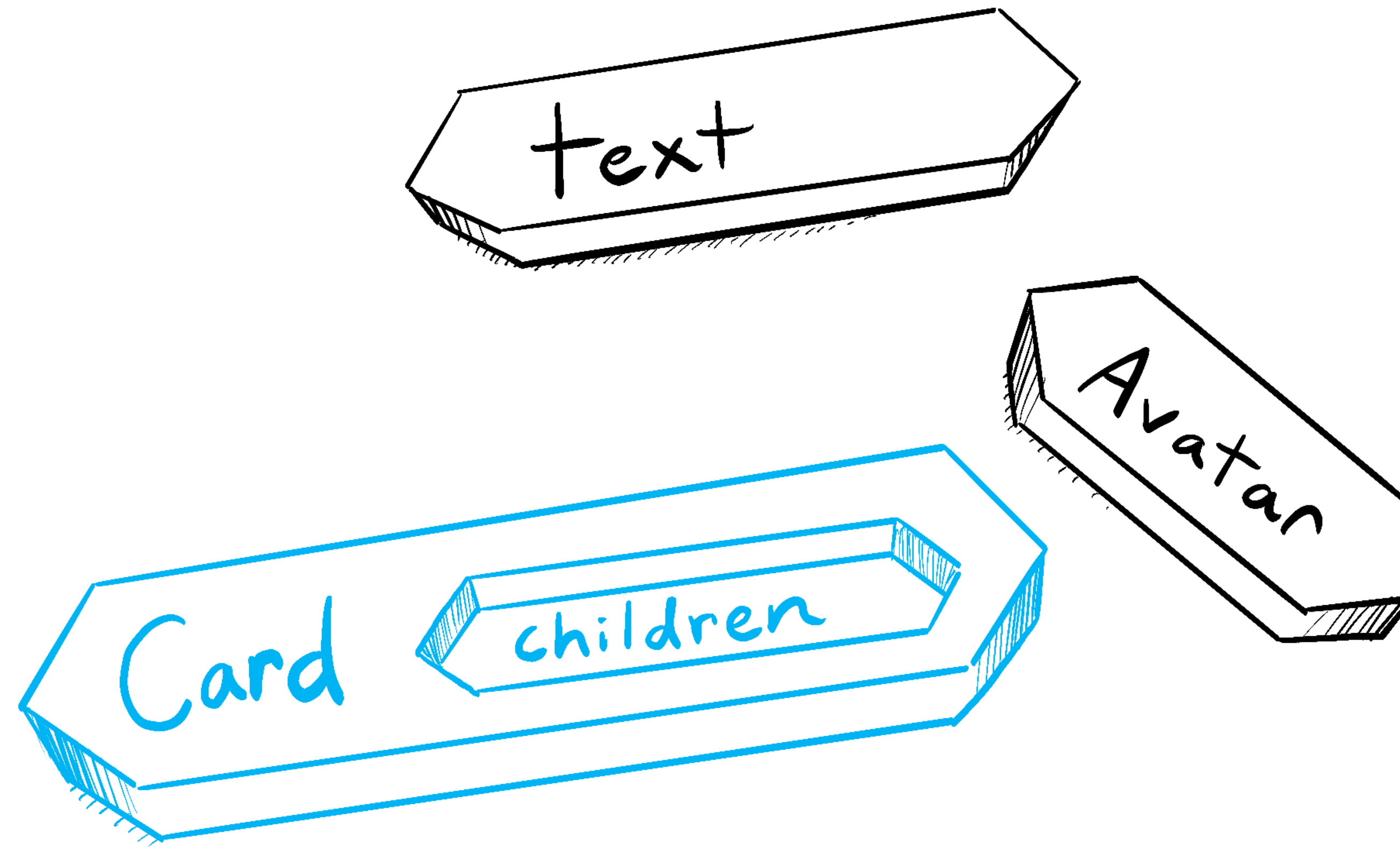
[bit.ly/jsx-curly-braces](https://bit.ly/jsx-curly-braces)

# Children

A prop that can be filled in by its parent

# Children

{children} A HOLE THAT CAN BE FILLED IN BY IT'S PARENT



[bit.ly/jsx-as-children](https://bit.ly/jsx-as-children)



```
1 import Avatar from './Avatar.js';
2
3 function Card({ children }) {
4   return (
5     <div className="card">
6       {children}
7     </div>
8   );
9 }
10
11 export default function Profile() {
12   return (
13     <Card>
14       <Avatar
15         size={100}
16         person={{
17           name: 'Katsuko Saruhashi',
18           imageId: 'Yfe0qp2'
19         }}
20       />
21     </Card>
22   );
23 }
24
```



# Context

An alternative to passing props

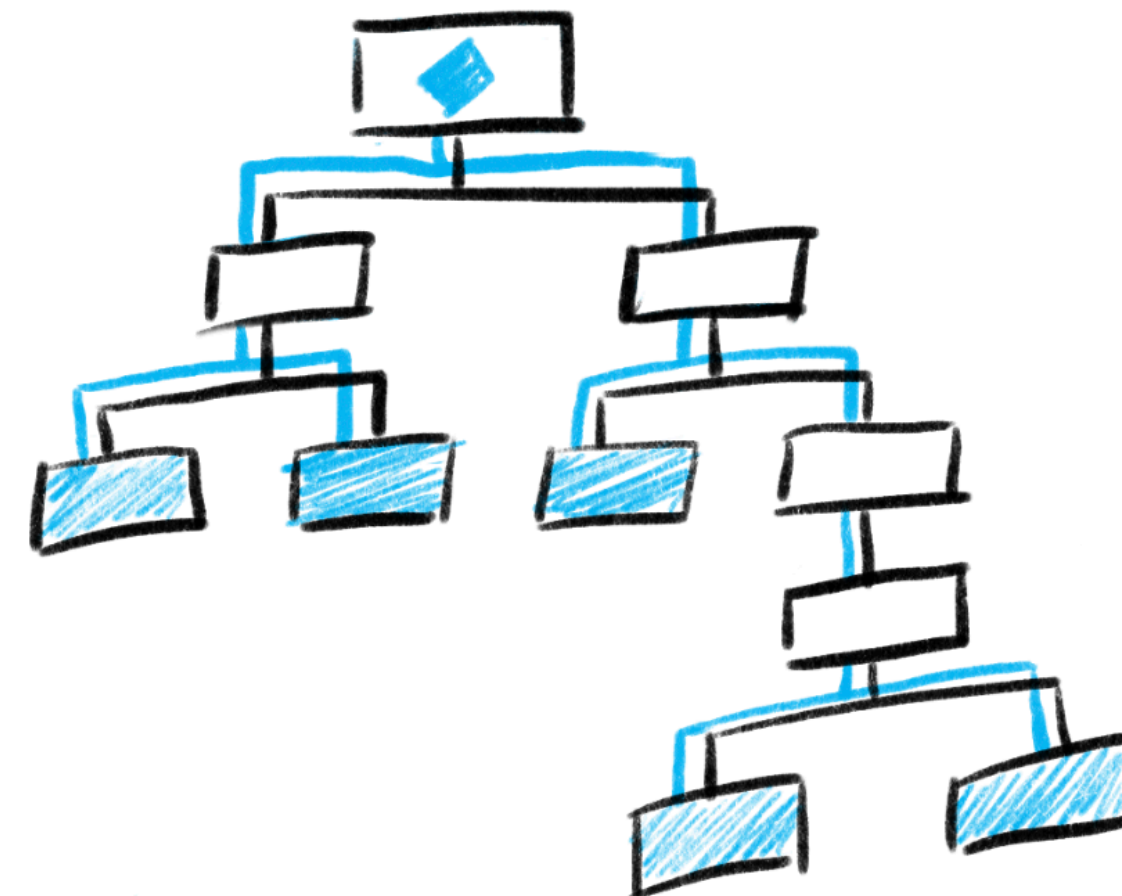
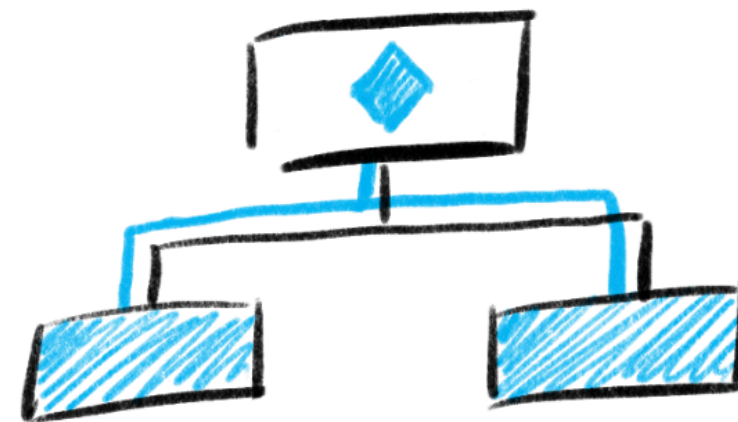
# Context

LETS A PARENT COMPONENT PROVIDE DATA TO THE ENTIRE TREE BELOW IT

## The problem with passing props

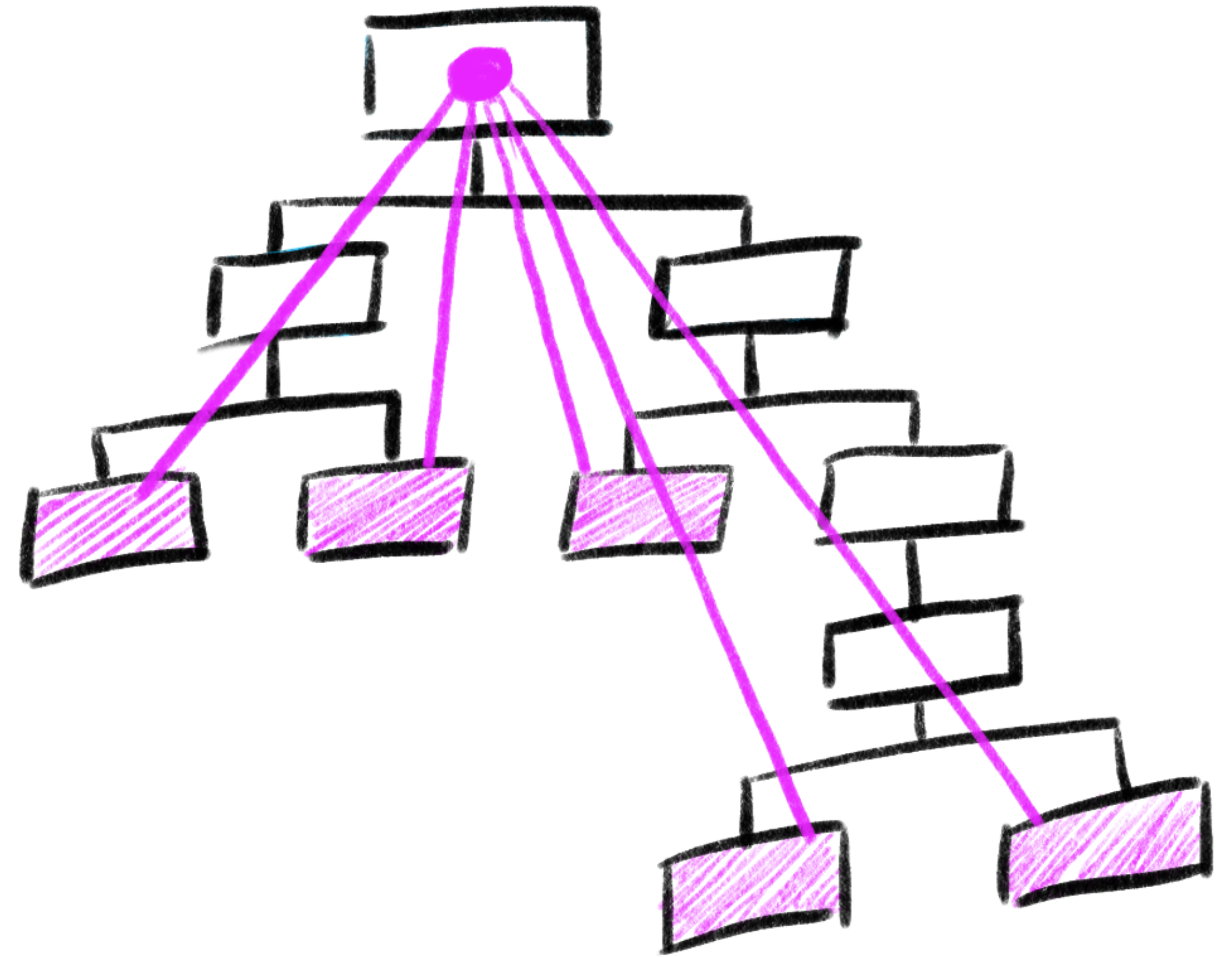
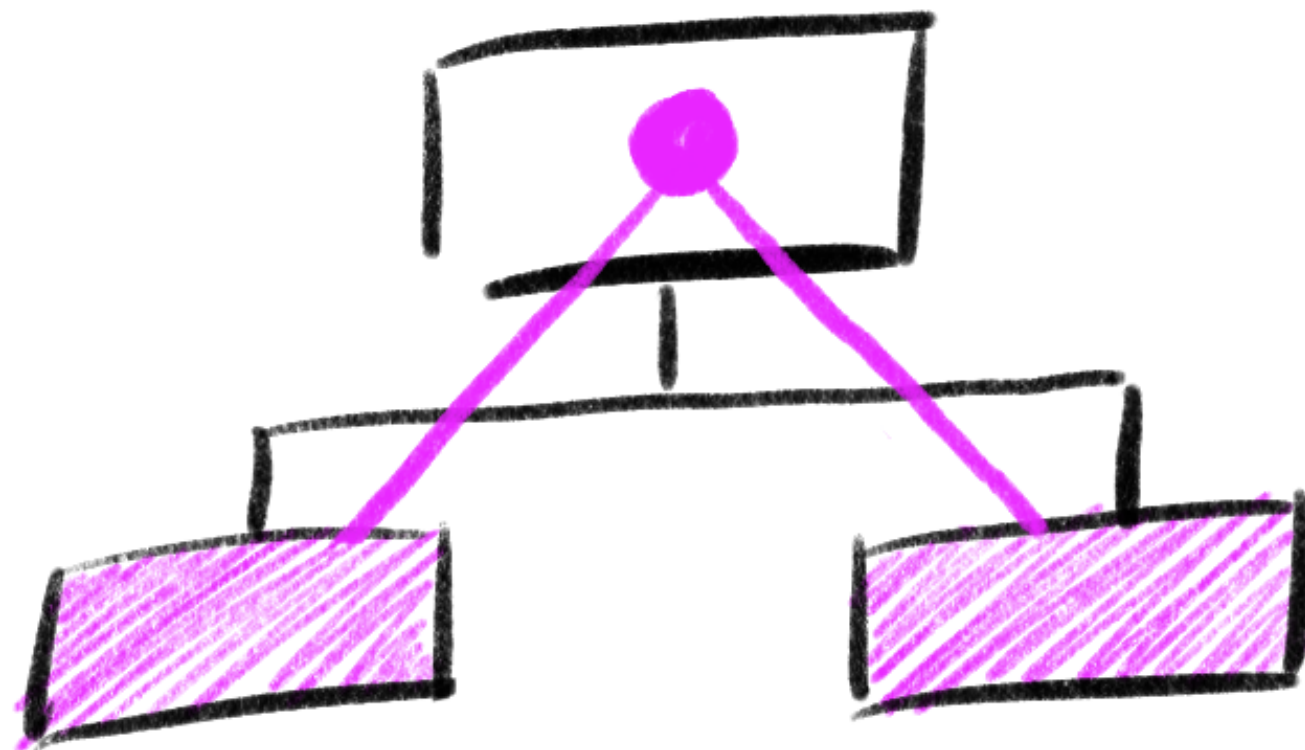
**Passing props** is a great way to explicitly pipe data through your UI tree to the components that use it. But it can become verbose and inconvenient when you need to pass some prop deeply through the tree, or if many components need the same prop. The nearest common ancestor could be far removed from the components that need data, and **lifting state up** that high can lead to a situation sometimes called “prop drilling.”

“Lifting state up” → “prop drilling”



# Context

LETS A PARENT COMPONENT PROVIDE DATA TO THE ENTIRE TREE BELOW IT



[bit.ly/use-context](https://bit.ly/use-context)

# Hooks

Special functions that start with “use” and are only available while React is rendering

# useState() Hook

STATE: A COMPONENT'S MEMORY

## 1 Declare a state variable

Call `useState` and pass the initial state to it. React will store the state that you passed, and give it back to you.

## 2 Update state on interaction

To change it, call the state setter function with the next state value. React will put that value into state instead.

## 3 Render state in the UI

Use the state for rendering by putting it into the JSX.

```
import { useState } from 'react';

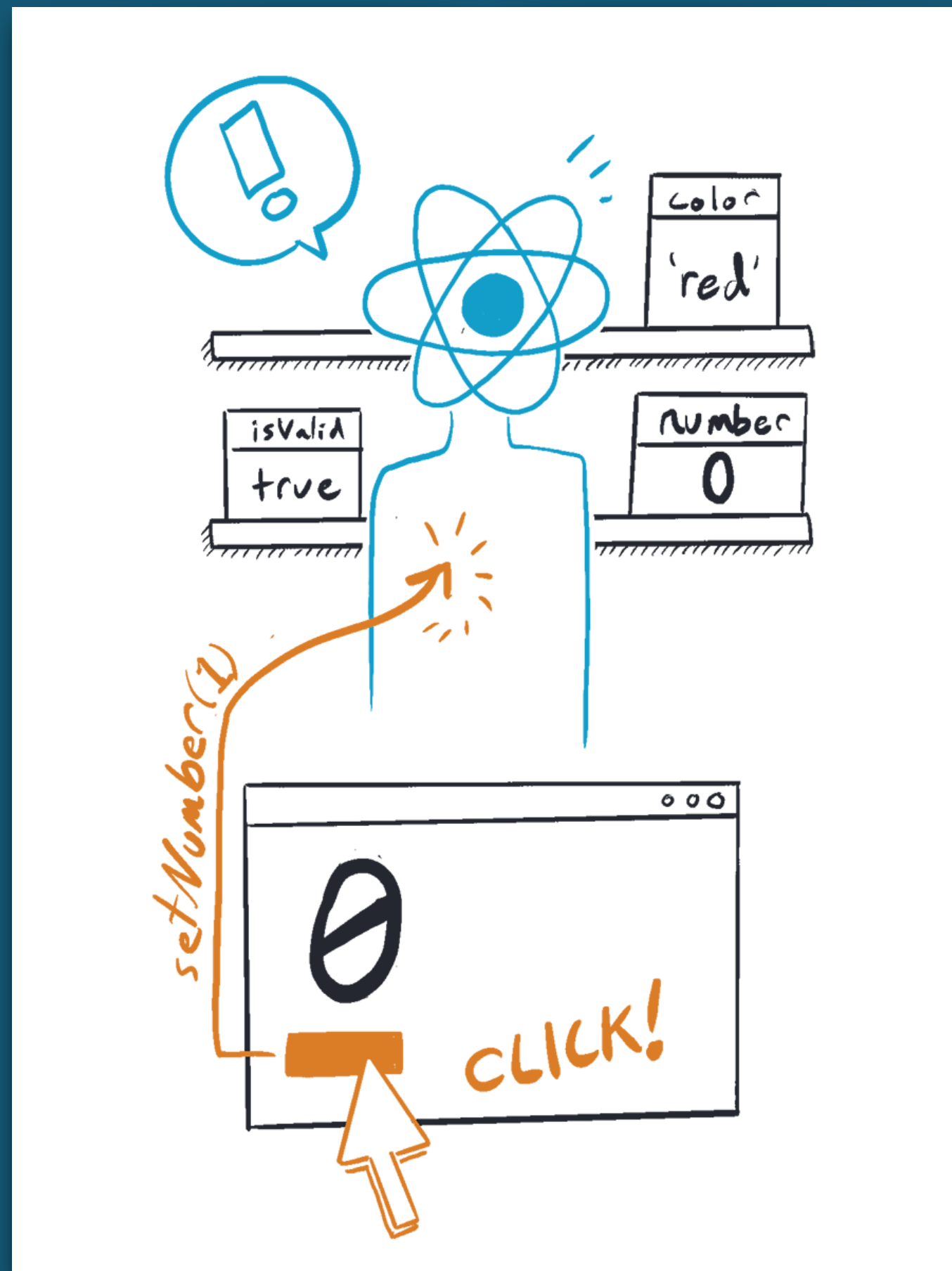
function Counter() {
  1 const [count, setCount] = useState(0);

  function handleClick() {
    2 setCount(count + 1);
  }

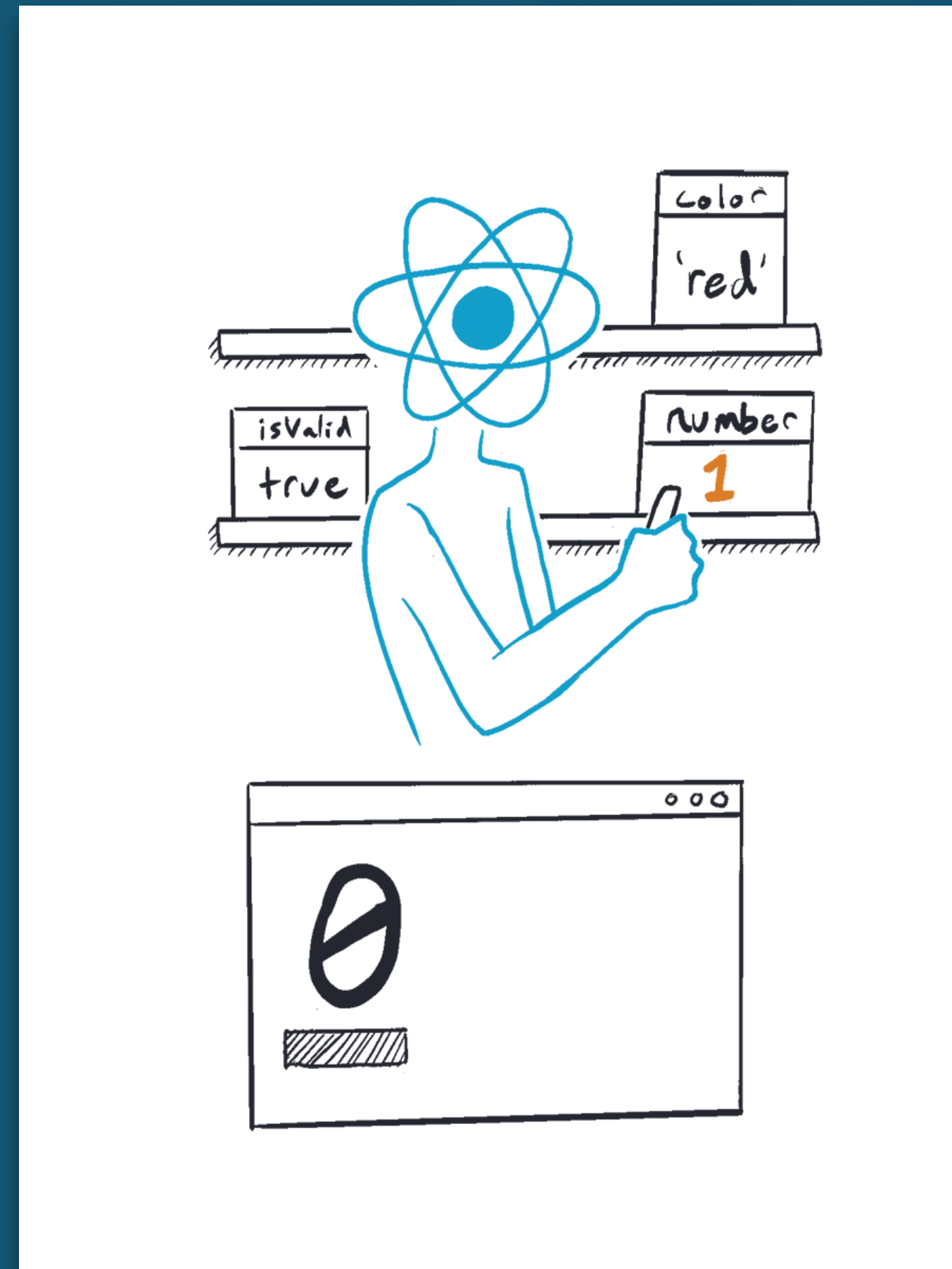
  return (
    <button onClick={handleClick}>
      You pressed me {3 count} times
    </button>
  );
}
```

[bit.ly/use-state](https://bit.ly/use-state)

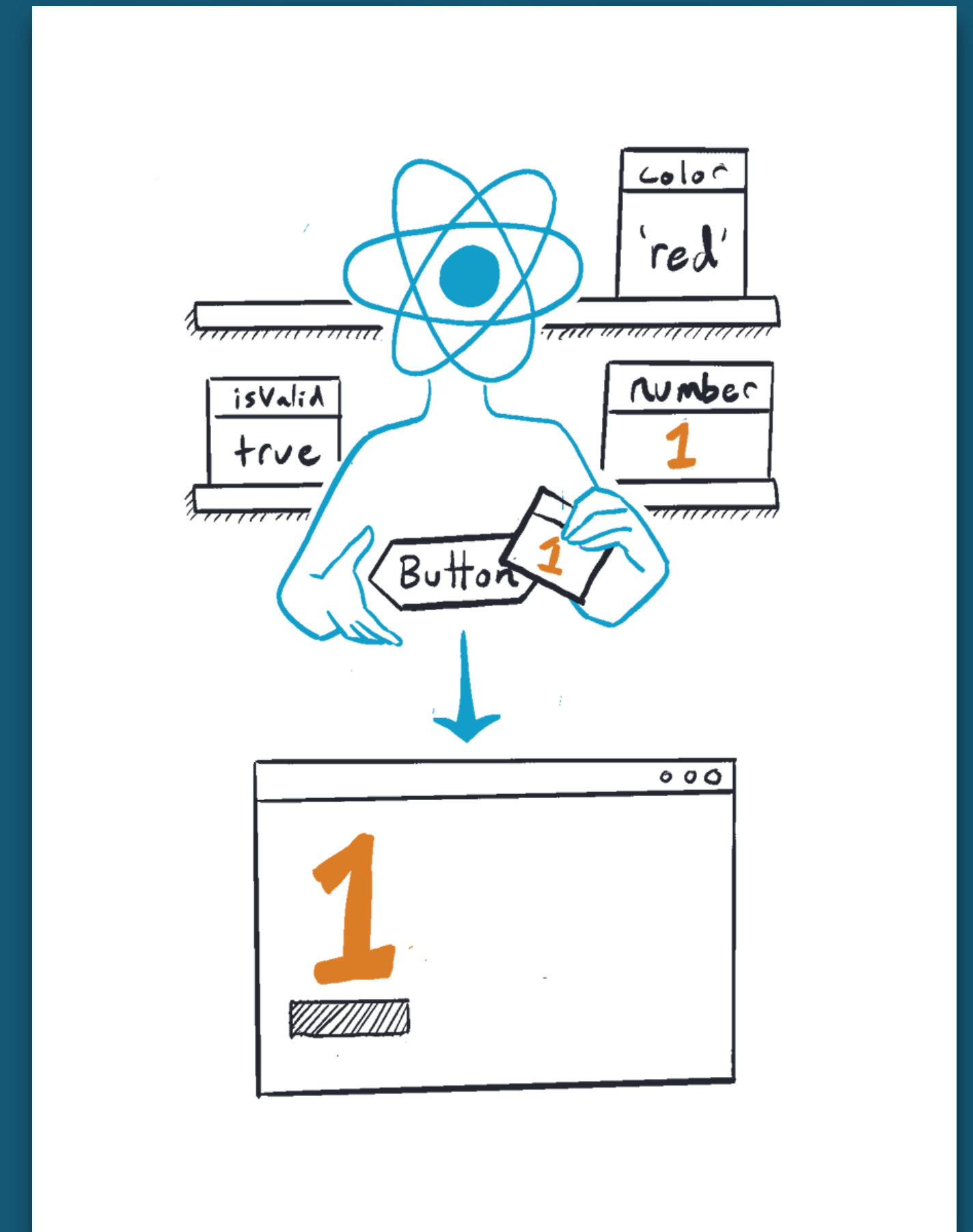
# Updating State Triggers a Re-Render



React gets a state update



React updates the state value



React passes a snapshot of the state value into the component

# State over time

WHAT NUMBER WILL THE ALERT BE?

App.js

Download Reset Fork

```
1 import { useState } from 'react';
2
3 export default function Counter() {
4   const [number, setNumber] = useState(0);
5
6   return (
7     <>
8       <h1>{number}</h1>
9       <button onClick={() => {
10         setNumber(number + 5);
11         alert(number);
12       }}>+5</button>
13     </>
14   )
15 }
16
```

0

+5



# React batches state updates

REACT PROCESSES STATE UPDATES AFTER EVENT HANDLERS HAVE FINISHED RUNNING

App.js

Download Reset Fork

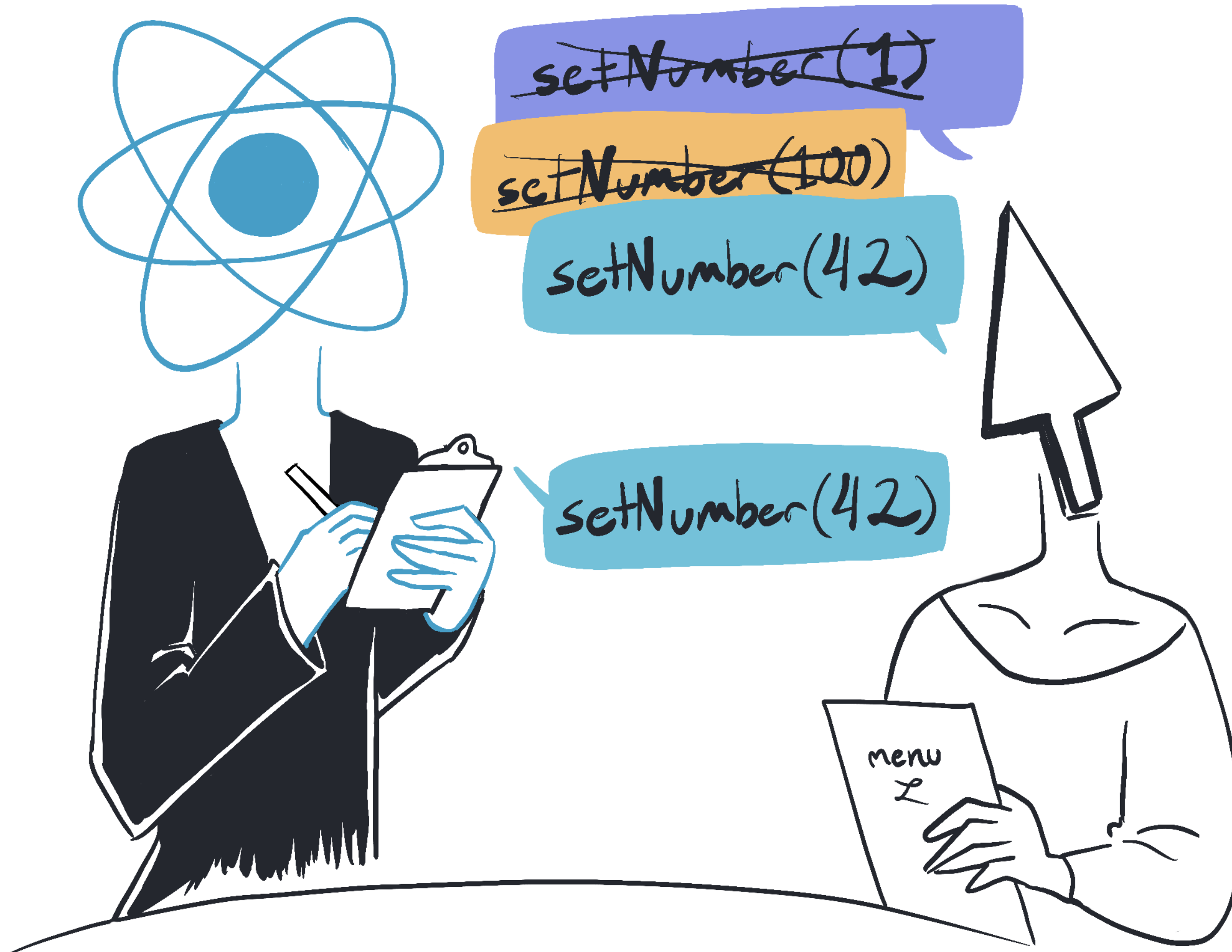
```
1 import { useState } from 'react';
2
3 export default function Counter() {
4   const [number, setNumber] = useState(0);
5
6   return (
7     <>
8       <h1>{number}</h1>
9       <button onClick={() => {
10         setNumber(number + 1);
11         setNumber(number + 1);
12         setNumber(number + 1);
13       }}>+3</button>
14     </>
15   )
16 }
```

0

+3

# React batches state updates

REACT PROCESSES STATE UPDATES AFTER EVENT HANDLERS HAVE FINISHED RUNNING



# Replacing state after updating it

ADDS "REPLACE WITH 42" TO THE QUEUE, IGNORING WHAT'S ALREADY QUEUED

App.js

Download Reset Fork

```
1 import { useState } from 'react';
2
3 export default function Counter() {
4   const [number, setNumber] = useState(0);
5
6   return (
7     <>
8       <h1>{number}</h1>
9       <button onClick={() => {
10         setNumber(number + 99);
11         setNumber(42);
12       }}>Increase the number</button>
13     </>
14   )
15 }
16
```

0

Increase the number

# Updater Function

UPDATE THE STATE MULTIPLE TIMES DURING THE SAME EVENT

## 1 Setting state with next value

Usually, when you set state, you replace it.

## 2 Setting state with an updater

But you can pass an updater function to transform it.

```
function handleClick() {  
  setCount(1 123 );  
  
  setCount(2 c => c + 1 ); // Result: 124  
}
```

[bit.ly/updater-function](https://bit.ly/updater-function)

# Updater Function

UPDATE THE STATE MULTIPLE TIMES DURING THE SAME EVENT

App.js

Download Reset Fork

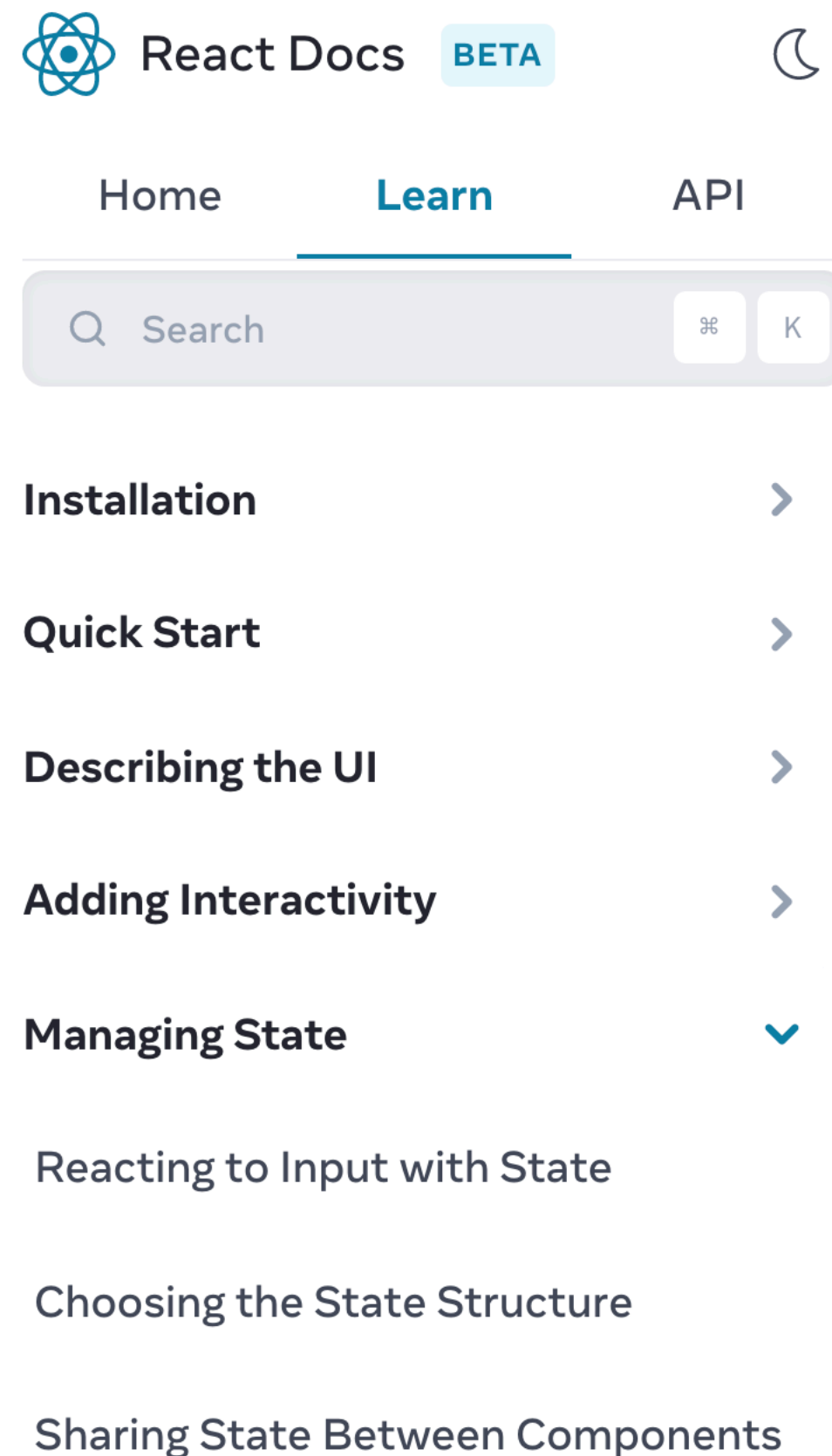
```
1 import { useState } from 'react';
2
3 export default function Counter() {
4   const [number, setNumber] = useState(0);
5
6   return (
7     <>
8       <h1>{number}</h1>
9       <button onClick={() => {
10         setNumber(number + 5);
11         setNumber(n => n + 1);
12       }}>Increase the number</button>
13     </>
14   )
15 }
16
```

0

Increase the number

# Reducer and Context for State Management

COMBINE CONTEXT WITH USEREDUCER YOU ESSENTIALLY END UP WITH A STATE MANAGEMENT SOLUTION



LEARN REACT > MANAGING STATE >

## Scaling Up with Reducer and Context

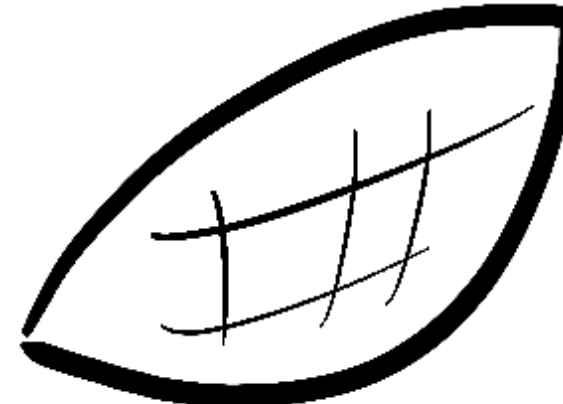
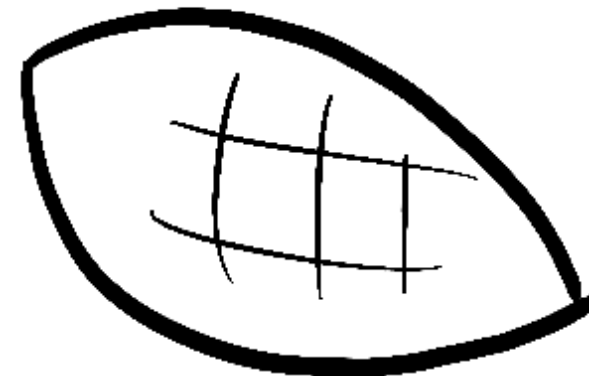
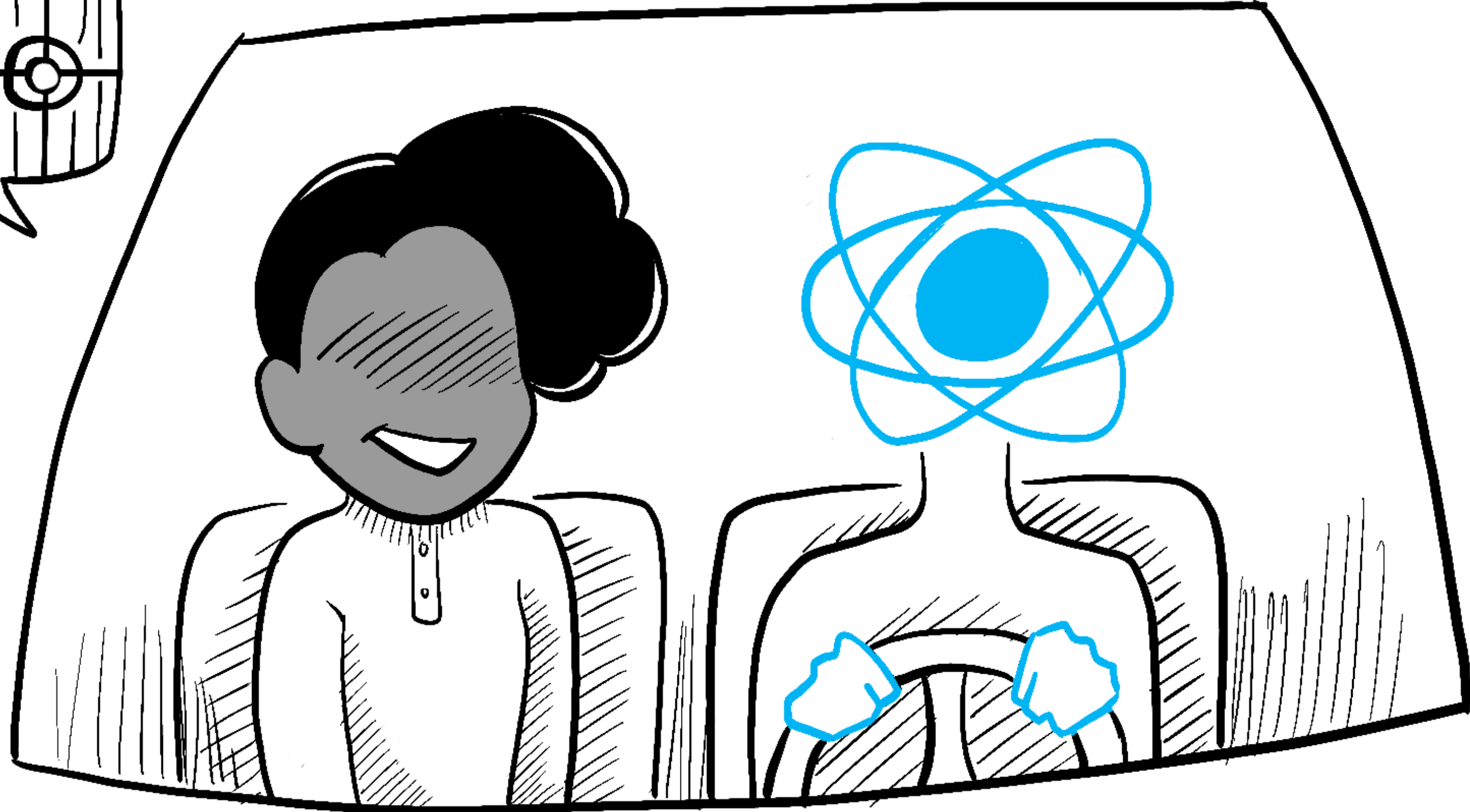
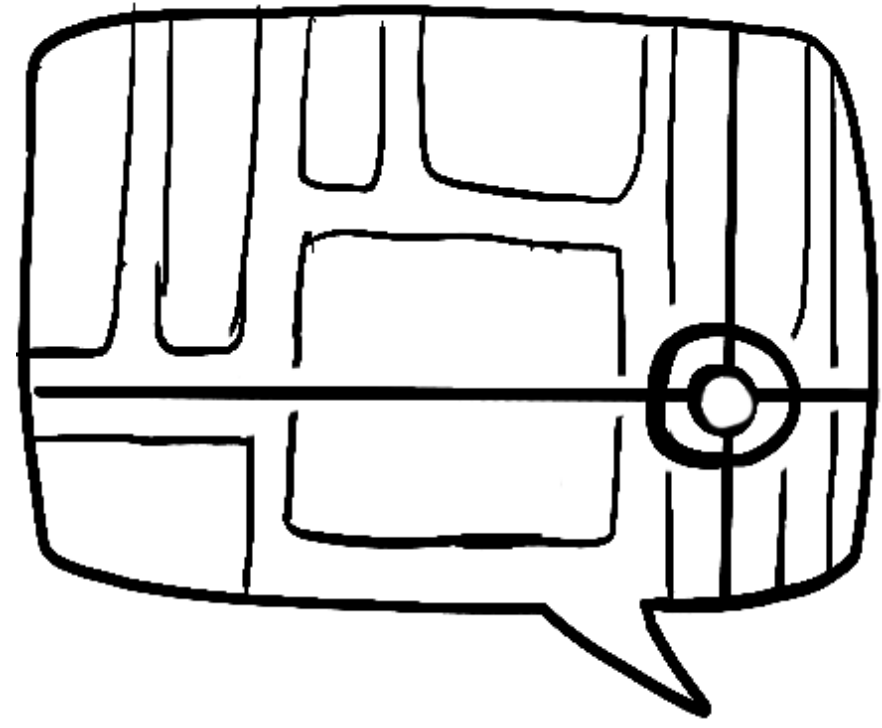
Reducers let you consolidate a component's state update logic. Context lets you pass information deep down to other components. You can combine reducers and context together to manage state of a complex screen.

### You will learn

- How to combine a reducer with context
- How to avoid passing state and dispatch through props
- How to keep context and state logic in a separate file

### Combining a reducer with context

<https://bit.ly/context-reducer>



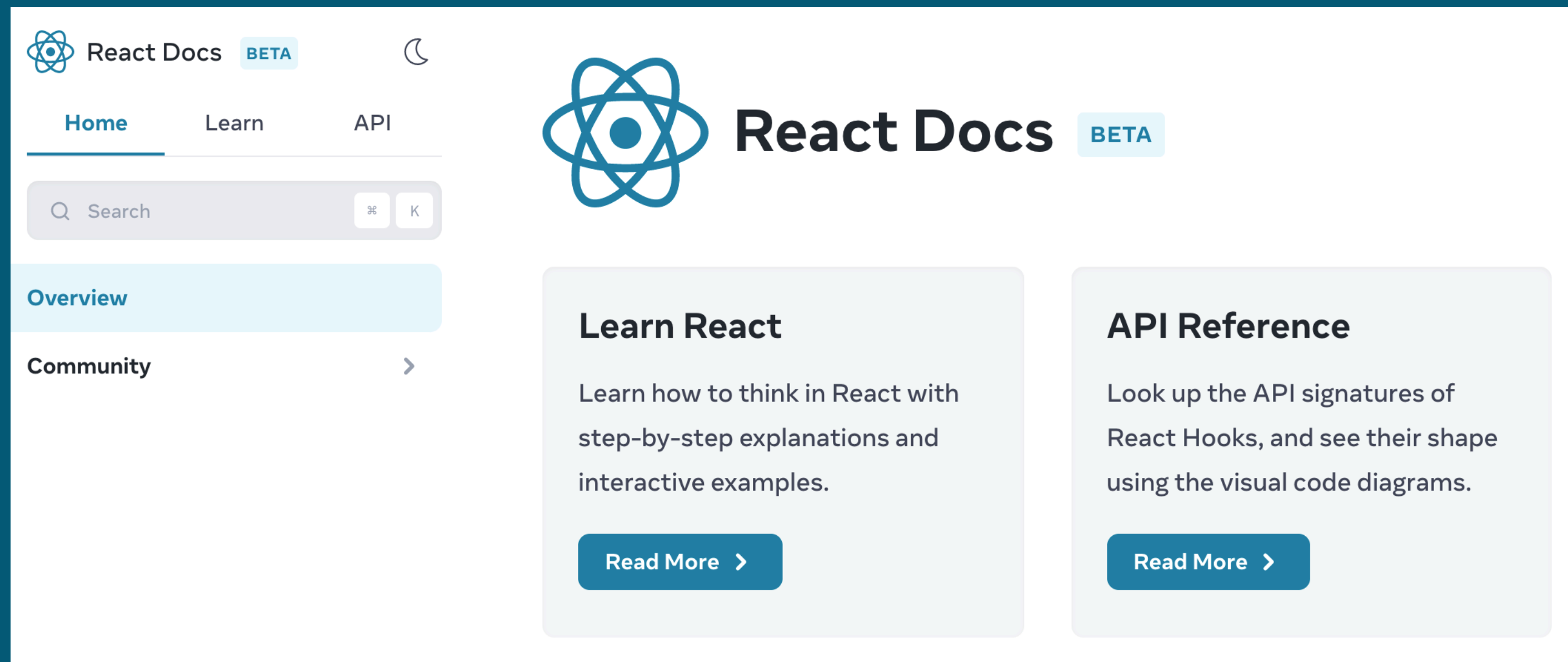
**There has never been a better time to learn**





# Thank You

## beta.reactjs.org



 @debs\_obrien  
debbie.codes