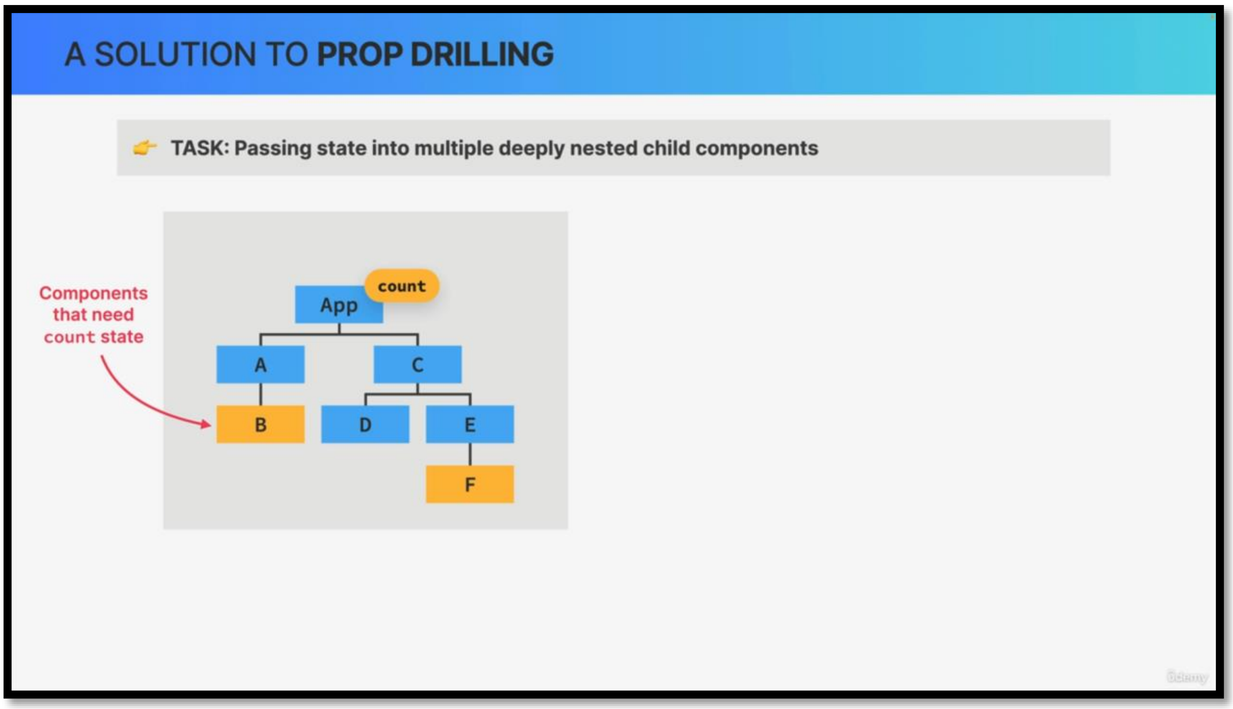
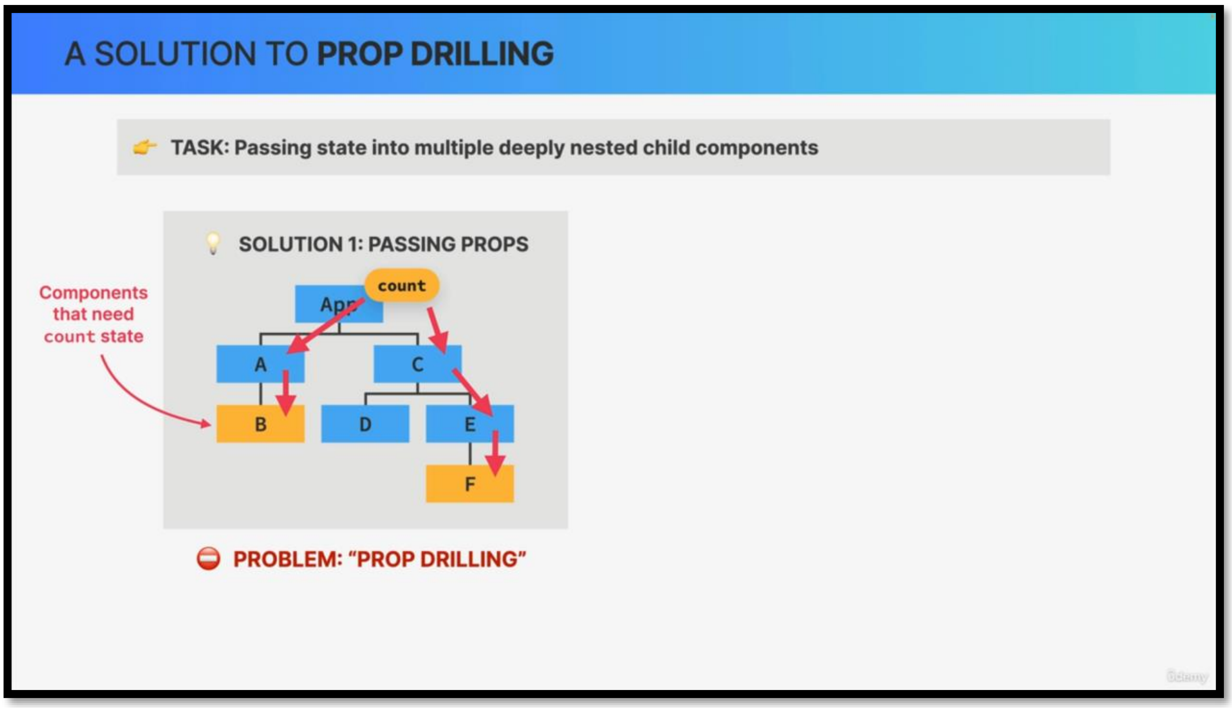


A SOLUTION TO **PROP DRILLING**

👉 **TASK: Passing state into multiple deeply nested child components**

Let's say that in our application, we need to pass some state into multiple deeply nested child components just like in this fictional example.

So, in this application, the components B and F both need access to the count state variable and that's the task that we need to solve.



A SOLUTION TO **PROP DRILLING**

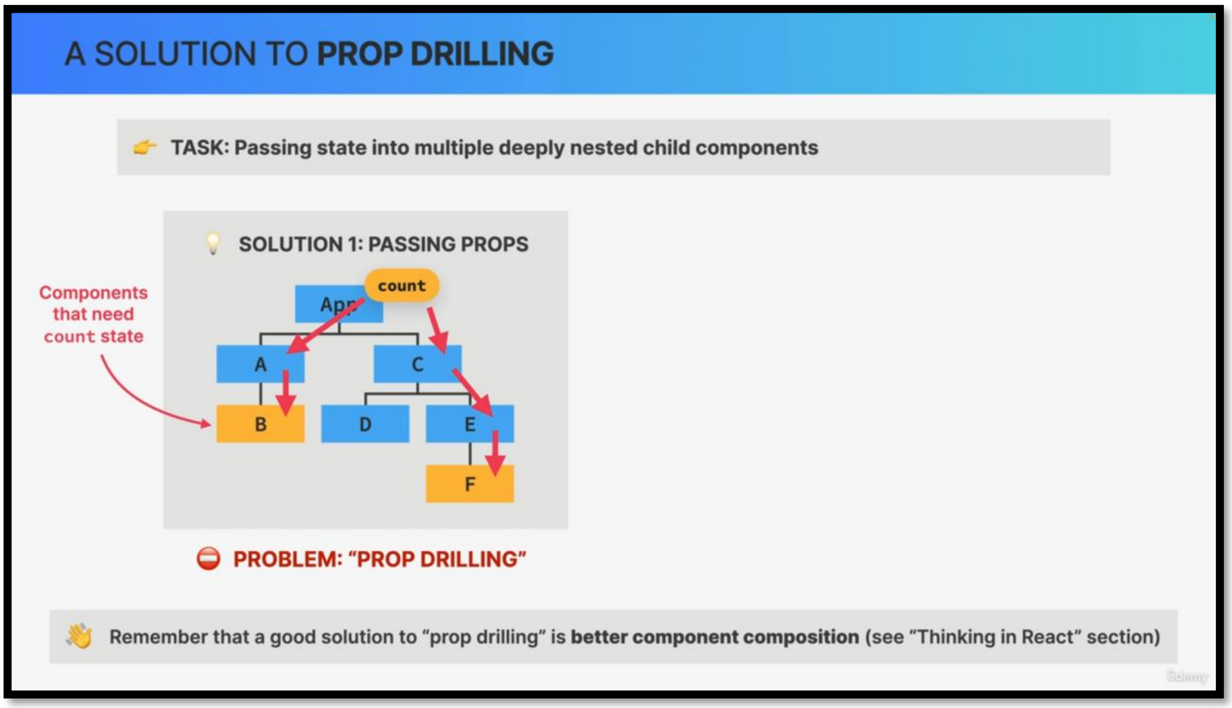
👉 **TASK: Passing state into multiple deeply nested child components**

💡 **SOLUTION 1: PASSING PROPS**

⛔ **PROBLEM: "PROP DRILLING"**

The first solution is to simply pass the state variable as props all the way down until it reaches the components that need the count state.

However, this then creates a new problem, because passing props down through multiple levels of the tree can quickly become cumbersome and inconvenient. We call this problem prop drilling.



A SOLUTION TO **PROP DRILLING**

👉 **TASK: Passing state into multiple deeply nested child components**

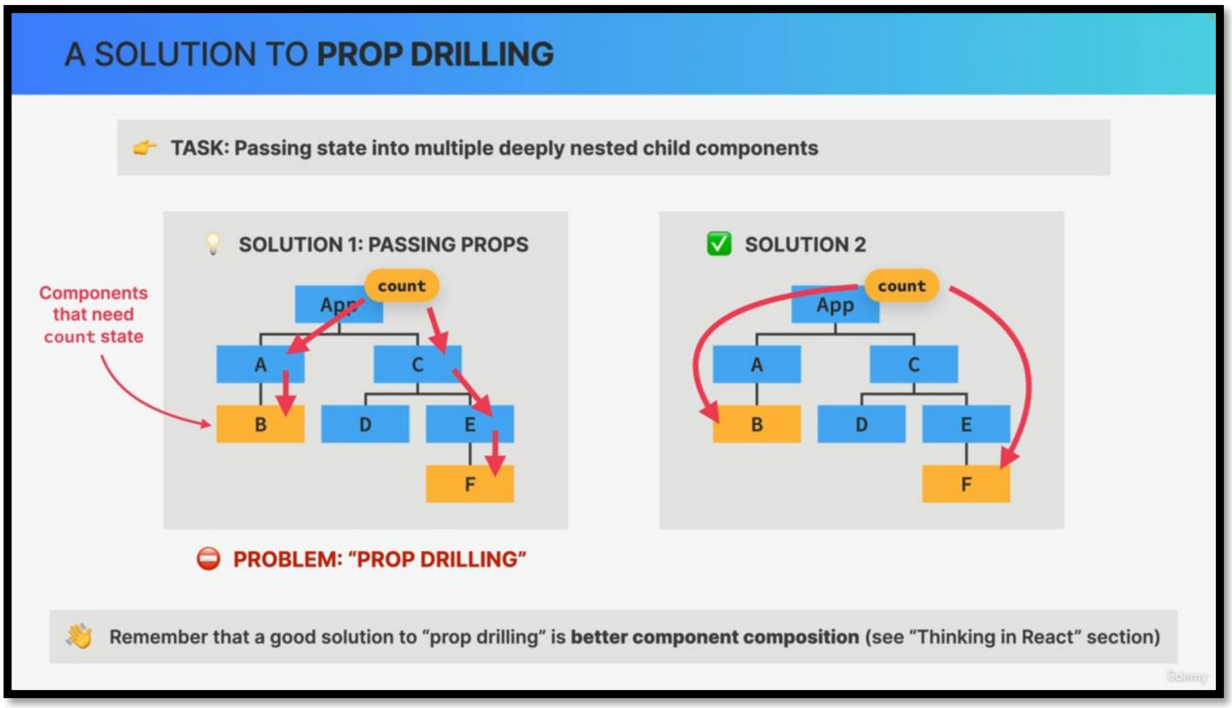
💡 **SOLUTION 1: PASSING PROPS**

⛔ **PROBLEM: "PROP DRILLING"**

👋 Remember that a good solution to "prop drilling" is **better component composition** (see "Thinking in React" section)

One good solution is to prop drilling is to compose components in a better way.

However, doing so is not always possible and so component composition doesn't always solve this problem.



A SOLUTION TO **PROP DRILLING**

👉 **TASK: Passing state into multiple deeply nested child components**

💡 **SOLUTION 1: PASSING PROPS**

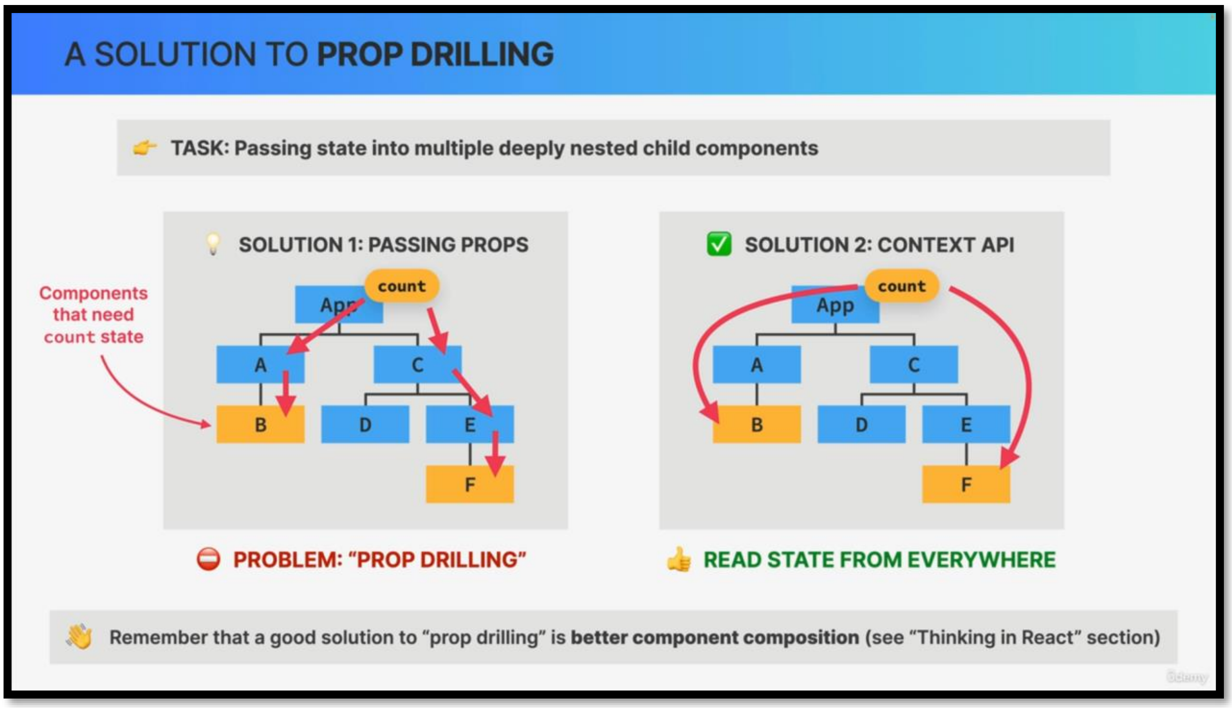
✅ **SOLUTION 2**

⛔ **PROBLEM: "PROP DRILLING"**

👋 Remember that a good solution to "prop drilling" is **better component composition** (see "Thinking in React" section)

So instead, what we need is actually a way of directly passing the state from a parent component into a deeply nested child component. So, that would immediately solve the problem.

It turns out that React has actually thought of that and has given us the Context API to do just that.



A SOLUTION TO **PROP DRILLING**

👉 **TASK: Passing state into multiple deeply nested child components**

💡 SOLUTION 1: PASSING PROPS

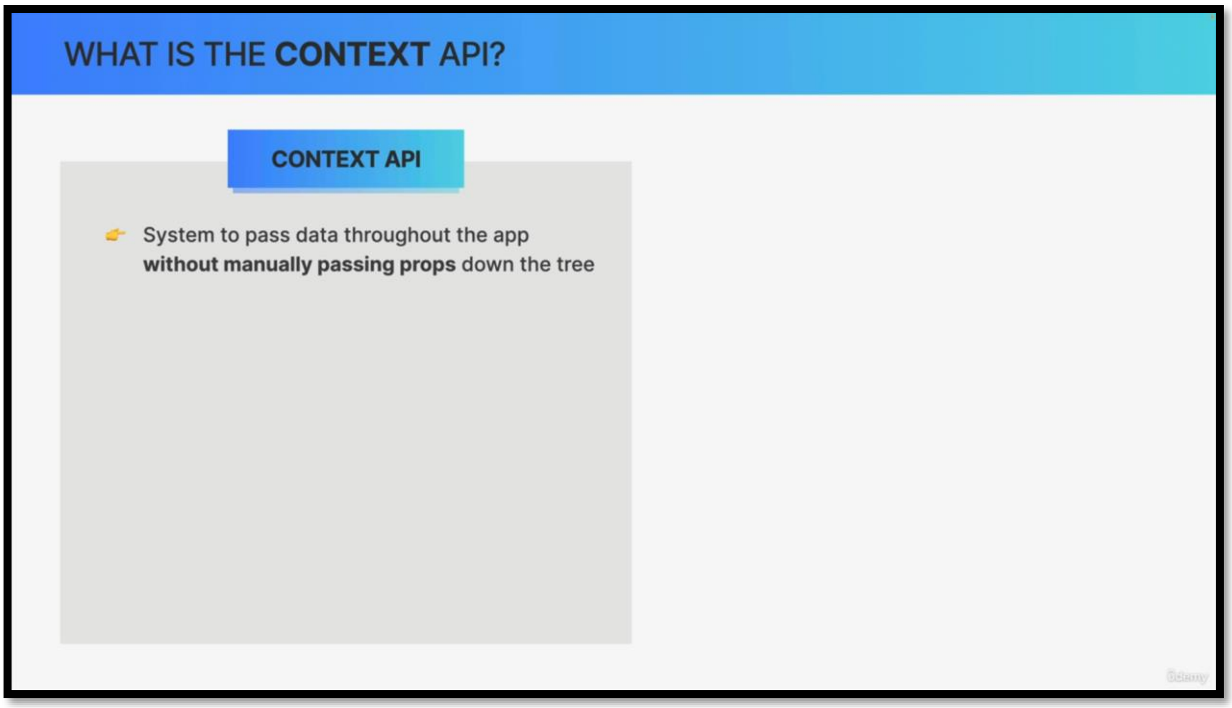
✅ SOLUTION 2: CONTEXT API

⛔ PROBLEM: "PROP DRILLING"

👍 READ STATE FROM EVERYWHERE

👋 Remember that a good solution to "prop drilling" is **better component composition** (see "Thinking in React" section)

The Context API basically allows components everywhere in the tree to read state that a context shares.

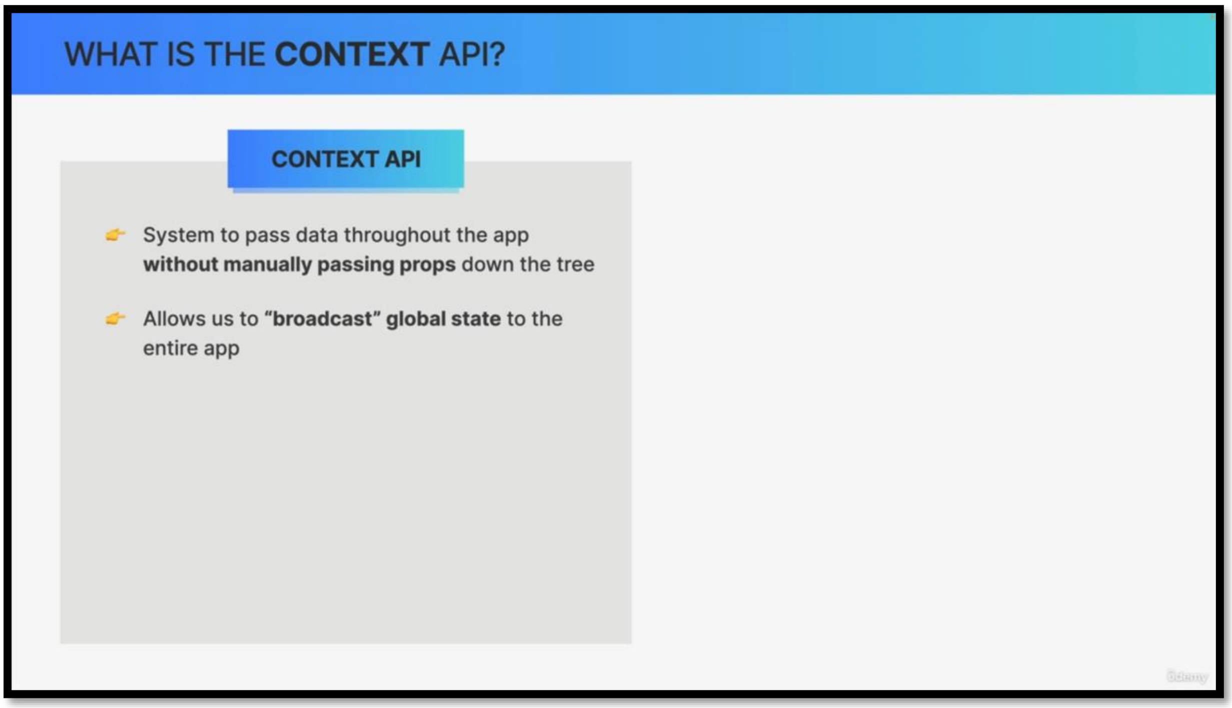


WHAT IS THE **CONTEXT** API?

CONTEXT API

👉 System to pass data throughout the app **without manually passing props** down the tree

The Context API is a system to pass data throughout the application without having to manually pass props down the component tree.

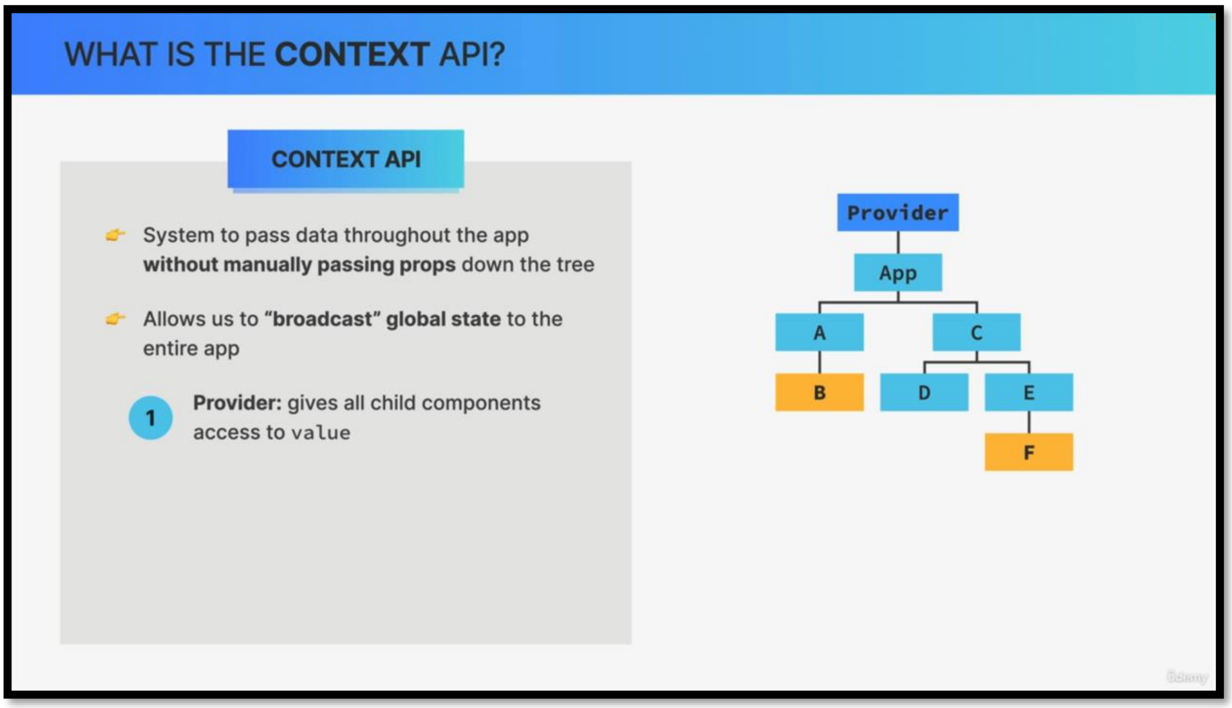


WHAT IS THE **CONTEXT** API?

CONTEXT API

- System to pass data throughout the app **without manually passing props** down the tree
- Allows us to **"broadcast" global state** to the entire app

It essentially allows us to broadcast global state. So that state should be available to all the child components of a certain context.



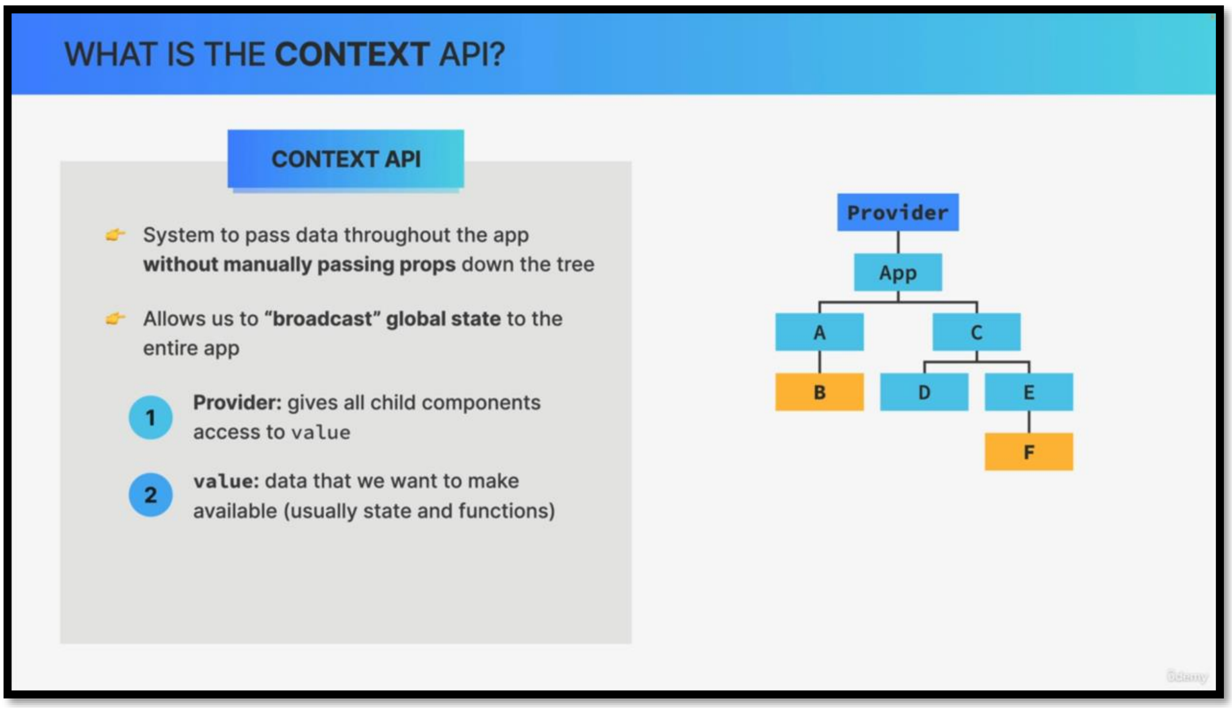
WHAT IS THE **CONTEXT** API?

CONTEXT API

- System to pass data throughout the app **without manually passing props** down the tree
- Allows us to **"broadcast" global state** to the entire app

1. **Provider:** gives all child components access to `value`

The first part of the Context API is the provider, which is a special React component that gives all child components access to a so-called value. This provider can sit everywhere in the component tree, but it's common to place it at the very top.

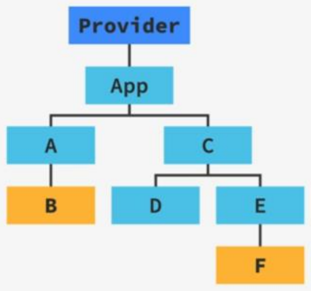


WHAT IS THE **CONTEXT** API?

CONTEXT API

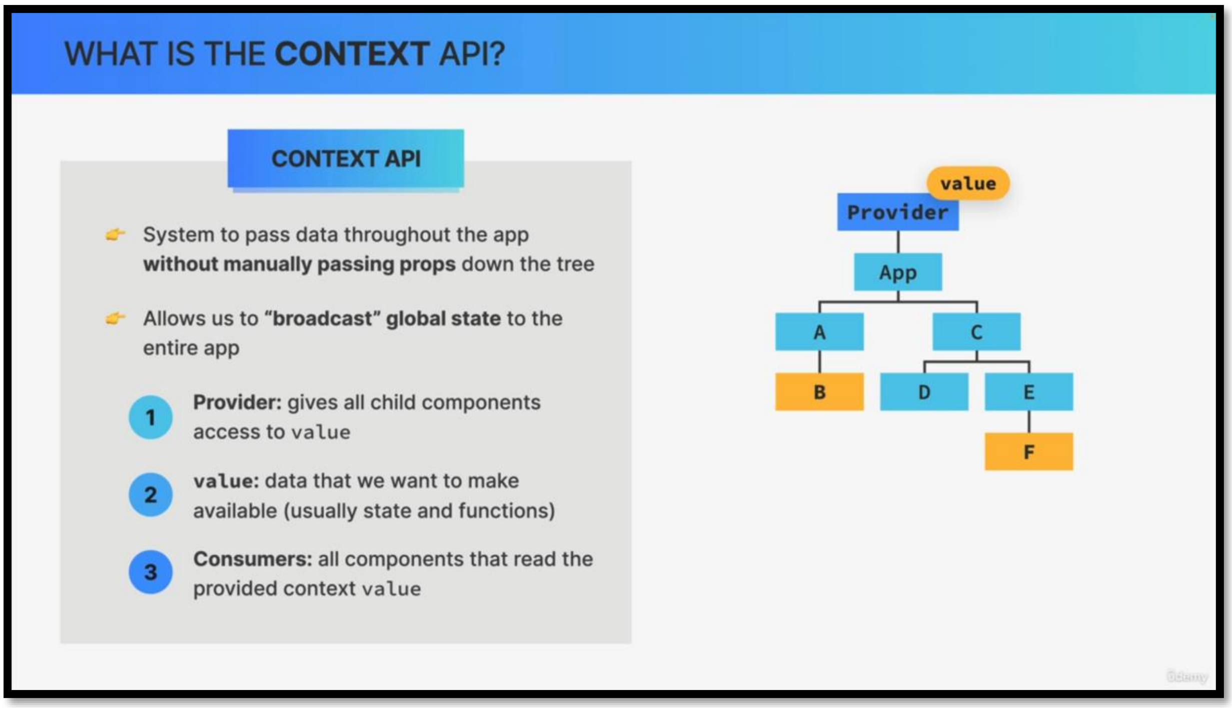
- System to pass data throughout the app **without manually passing props** down the tree
- Allows us to **"broadcast" global state** to the entire app

1. **Provider:** gives all child components access to `value`
2. **`value`:** data that we want to make available (usually state and functions)



This value that we were talking about is the data that we want to make available. The data that we want to broadcast through the provider.

So, we pass this value into the provider. This value contains one or more state variables and even some setter functions.

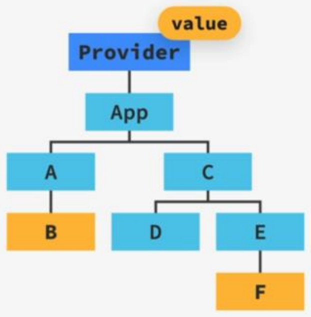


WHAT IS THE **CONTEXT** API?

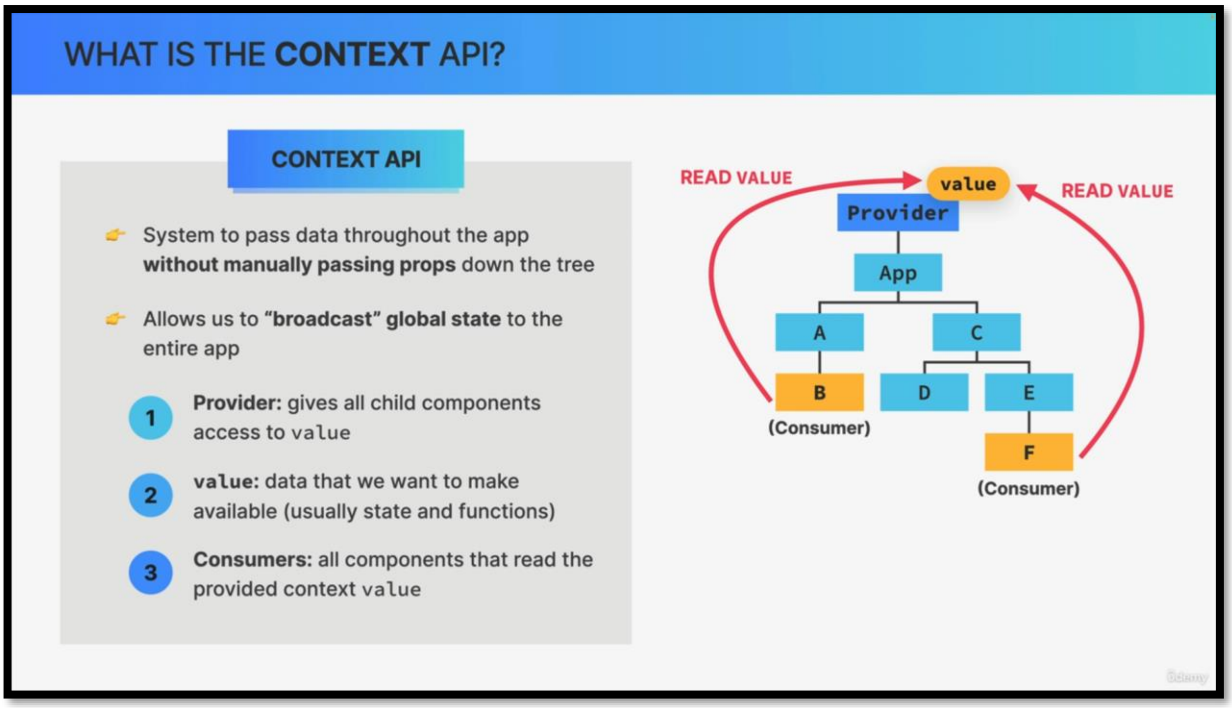
CONTEXT API

- System to pass data throughout the app **without manually passing props** down the tree
- Allows us to **"broadcast" global state** to the entire app

1. **Provider:** gives all child components access to `value`
2. **`value`:** data that we want to make available (usually state and functions)
3. **Consumers:** all components that read the provided context `value`



Finally, we have consumers which are all the components that read the value that we passed into the provider.



WHAT IS THE **CONTEXT** API?

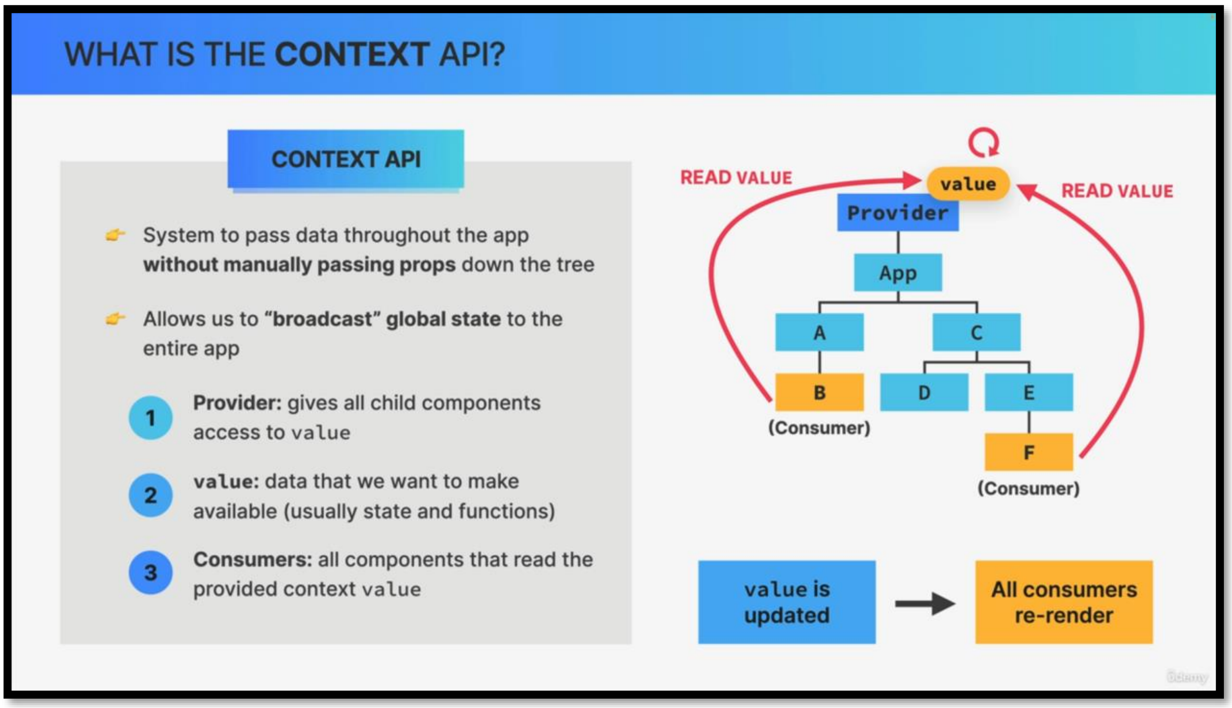
CONTEXT API

- System to pass data throughout the app **without manually passing props** down the tree
- Allows us to **"broadcast" global state** to the entire app

1. **Provider:** gives all child components access to `value`
2. **`value`:** data that we want to make available (usually state and functions)
3. **Consumers:** all components that read the provided context `value`

So, in other words, consumers are the components that subscribe to the context, and able to read the value from the context. We can create as many consumers as we want for any context provider.

So that's how the Context API works.



WHAT IS THE **CONTEXT** API?

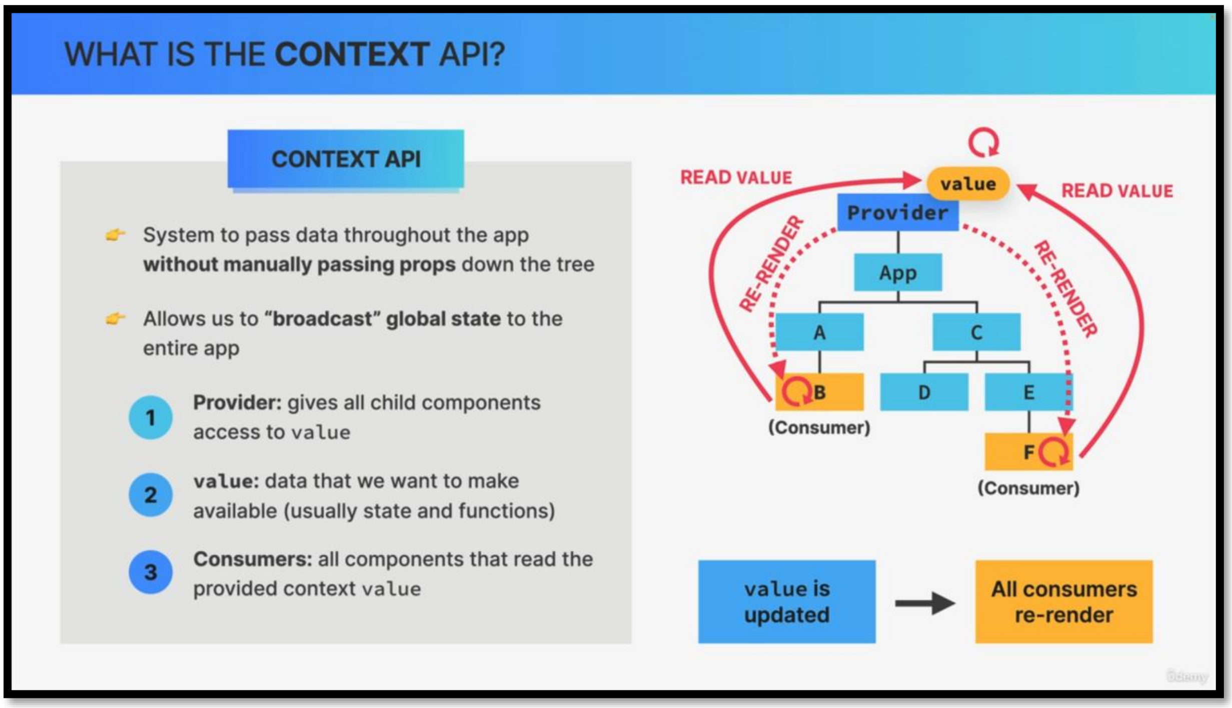
CONTEXT API

- System to pass data throughout the app **without manually passing props** down the tree
- Allows us to **"broadcast" global state** to the entire app

1. **Provider:** gives all child components access to `value`
2. **`value`:** data that we want to make available (usually state and functions)
3. **Consumers:** all components that read the provided context `value`

What happens when the context value actually changes, when it gets updated?

Whenever the context value is updated, all consumers will automatically be re-rendered i.e. all the components that are reading the context value.



WHAT IS THE CONTEXT API?

CONTEXT API

- System to pass data throughout the app **without manually passing props** down the tree
- Allows us to **"broadcast" global state** to the entire app

1. **Provider:** gives all child components access to `value`
2. **`value`:** data that we want to make available (usually state and functions)
3. **Consumers:** all components that read the provided context `value`

Whenever the value that is shared is updated in some way, the provider will immediately notify the consumers about the value change and it'll then re-render those components.

This means that now we have a new way in which component instances can be re-rendered.

We already knew that state updates re-render a component instance, but now we know that an update to a context value also re-renders a component as long as that component is subscribed to that exact context.

This is the fundamentals of the Context API and how it solves the prop drilling problem.

We can create as many contexts as we want in our application and place them wherever we want in the component tree.