Global and Shared Data

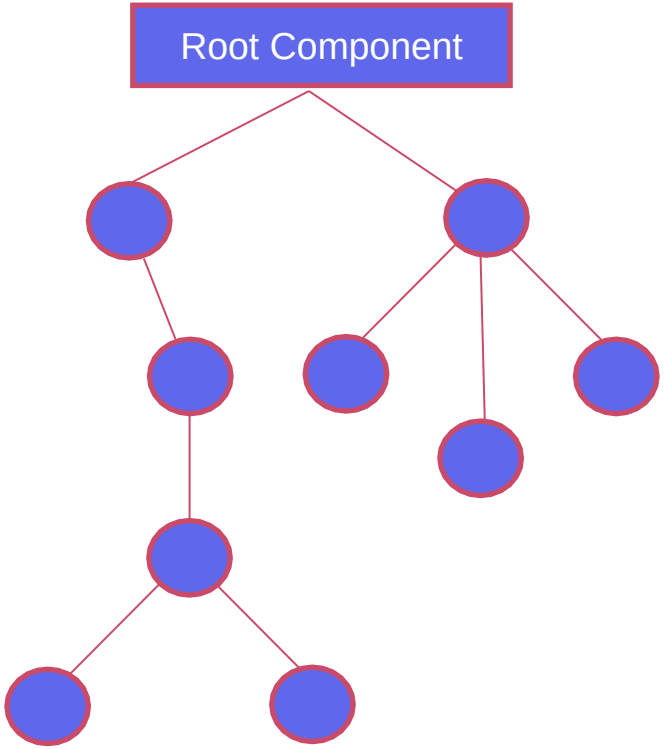
Unidirectional Data Flow

LEARNING OBJECTIVES

- Understand the nuances of data flow in a React application
- Understand the issues with sharing global data using props in a deeply nested application
- Learn and understand all about the Context API that lets you share global data without using props

UNDIRECTIONAL DATA FLOW

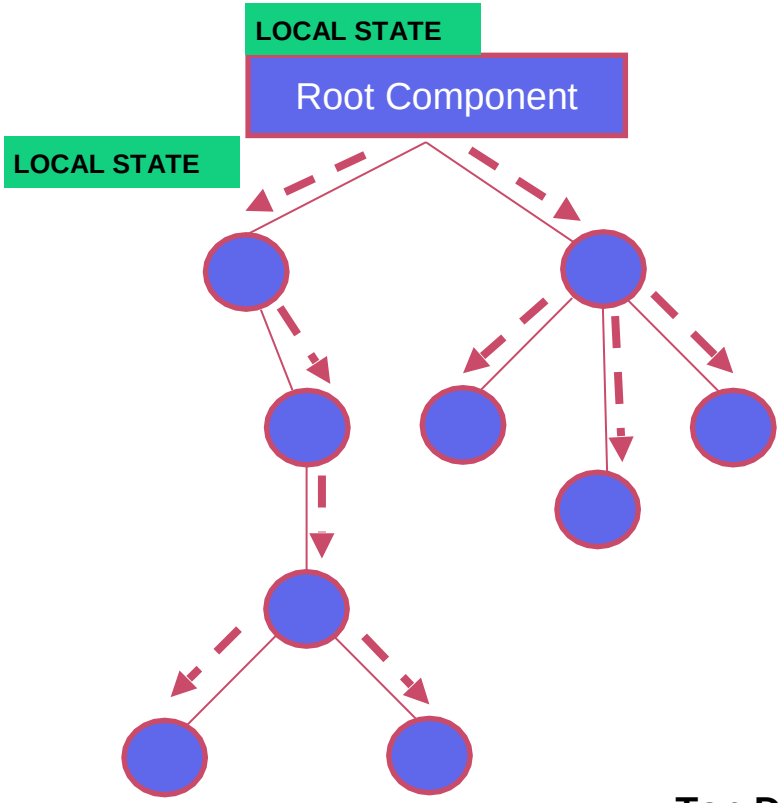
UNDIRECTIONAL DATA FLOW



A React application is made up of a number of nested components.

This results in a hierarchy that starts with a parent component and branches down at various levels

UNDIRECTIONAL DATA FLOW



Data flows down
through props
like a waterfall

Top Down Data Flow (Unidirectional)

UNIDIRECTIONAL DATA FLOW

Frameworks like Angular feature two-way data binding

Data Model

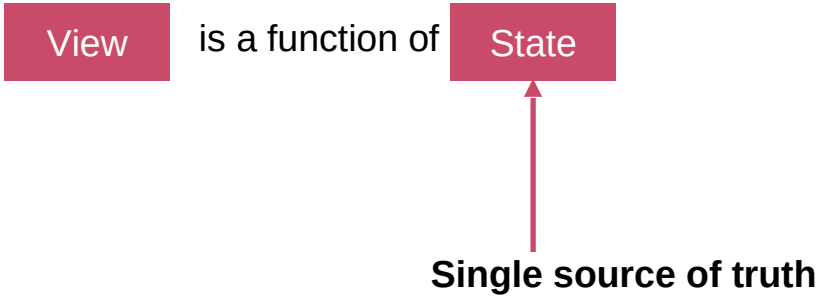
Changes to the data model updates the view

View

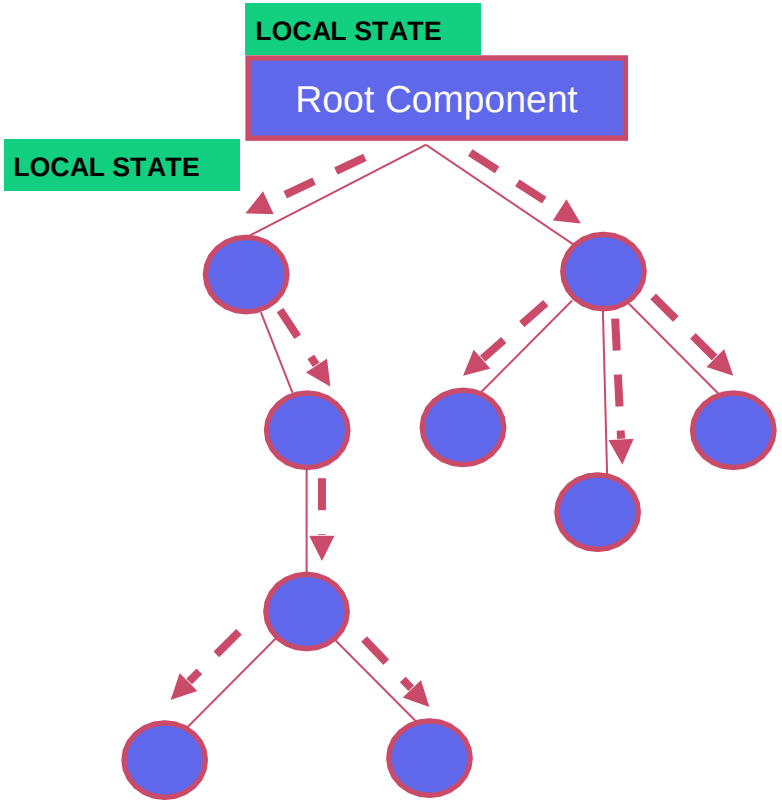
Changes to the view updates the data model

Two-way binding can pose challenges when keeping track of how updates happened, especially in large applications

UNIDIRECTIONAL DATA FLOW



UNDIRECTIONAL DATA FLOW



State is located at a parent level.

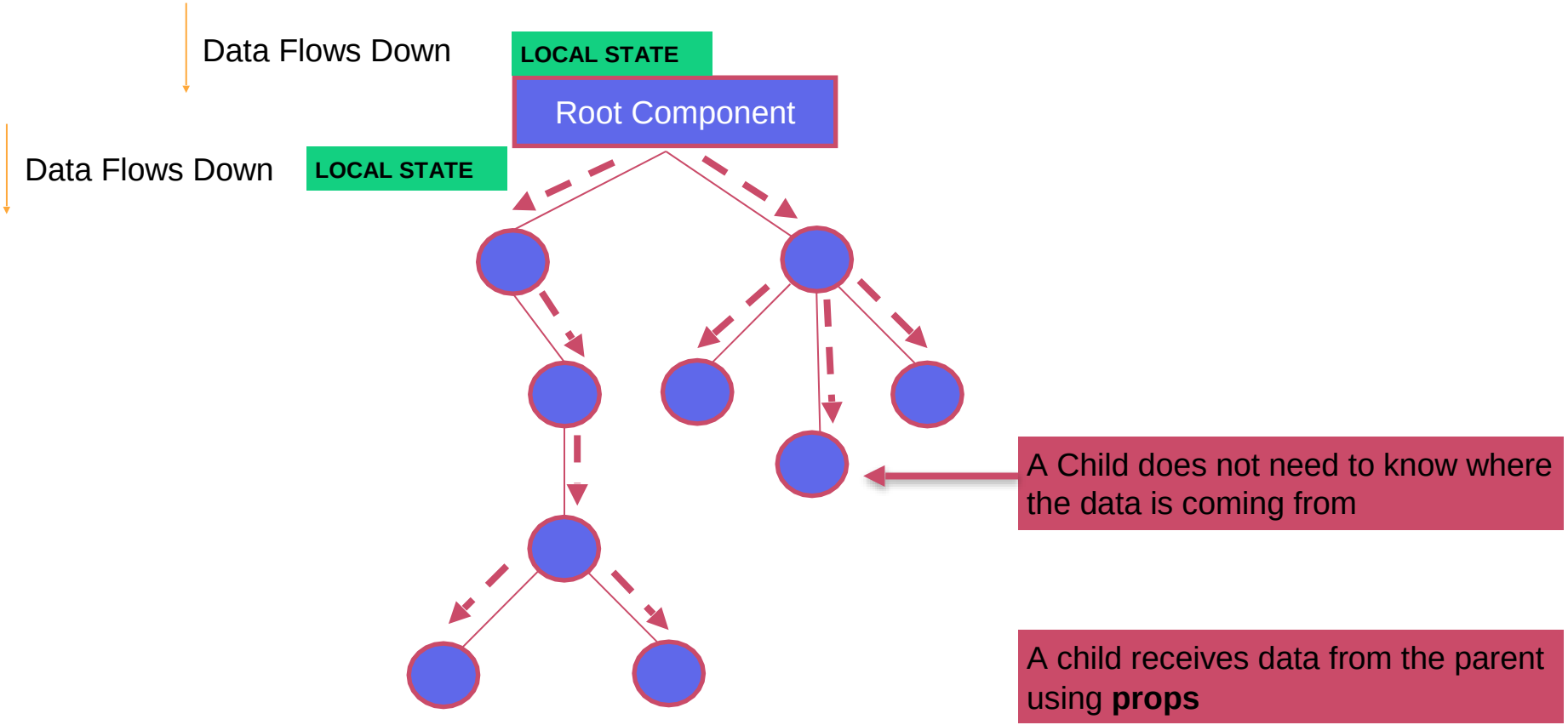
This does not necessarily have to be the top most parent or root component in the application hierarchy.

UNDIRECTIONAL DATA FLOW

```
class ToggleButton extends Component {
  state = {
    isEnabled: false
  };
  render() {
    return (
      <div
        className="tg-btn"
        onClick={() => this.setState({…})}>
        <div className="rail">
          <div className="knob" />
        </div>
      </div>
    );
  }
}
```

A toggle button component may **incorporate its own internal state to keep track of the toggle state of the UI.**

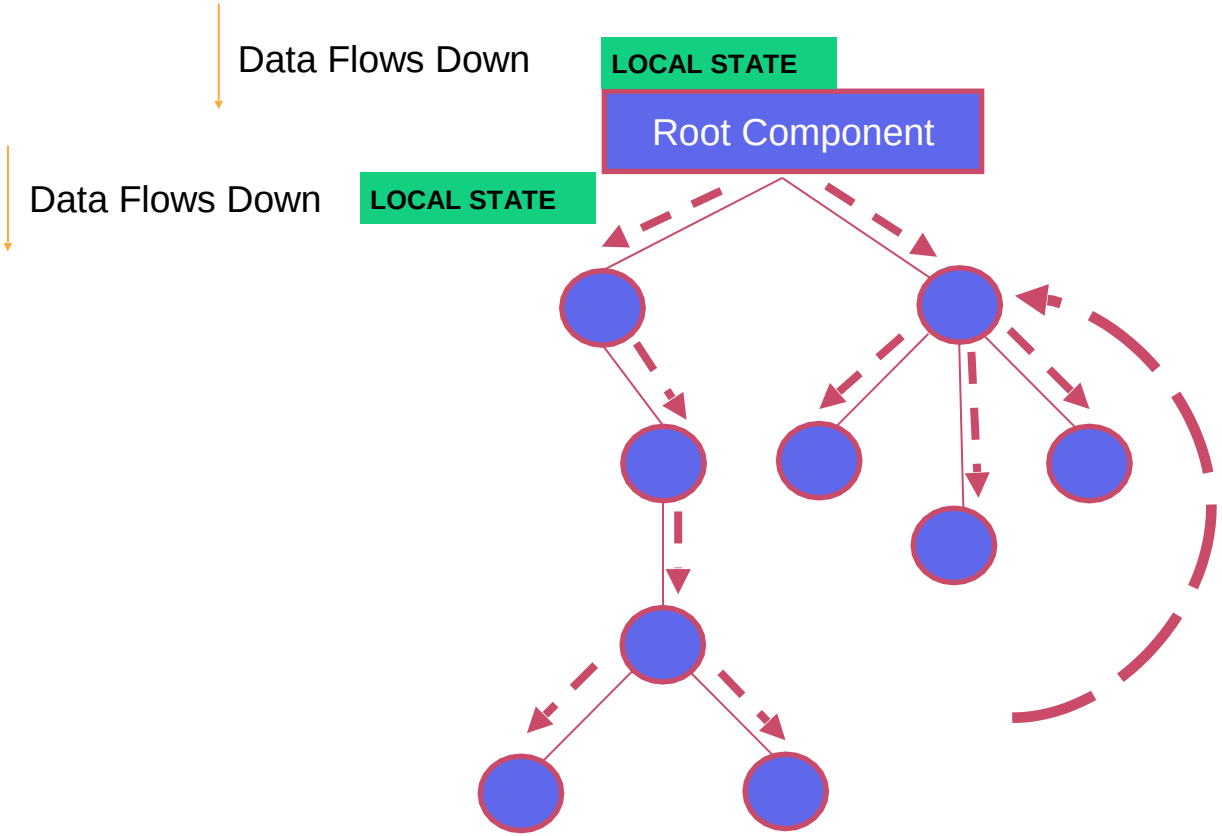
UNDIRECTIONAL DATA FLOW



The view is updated only when the data in the state updates and is passed using props to children

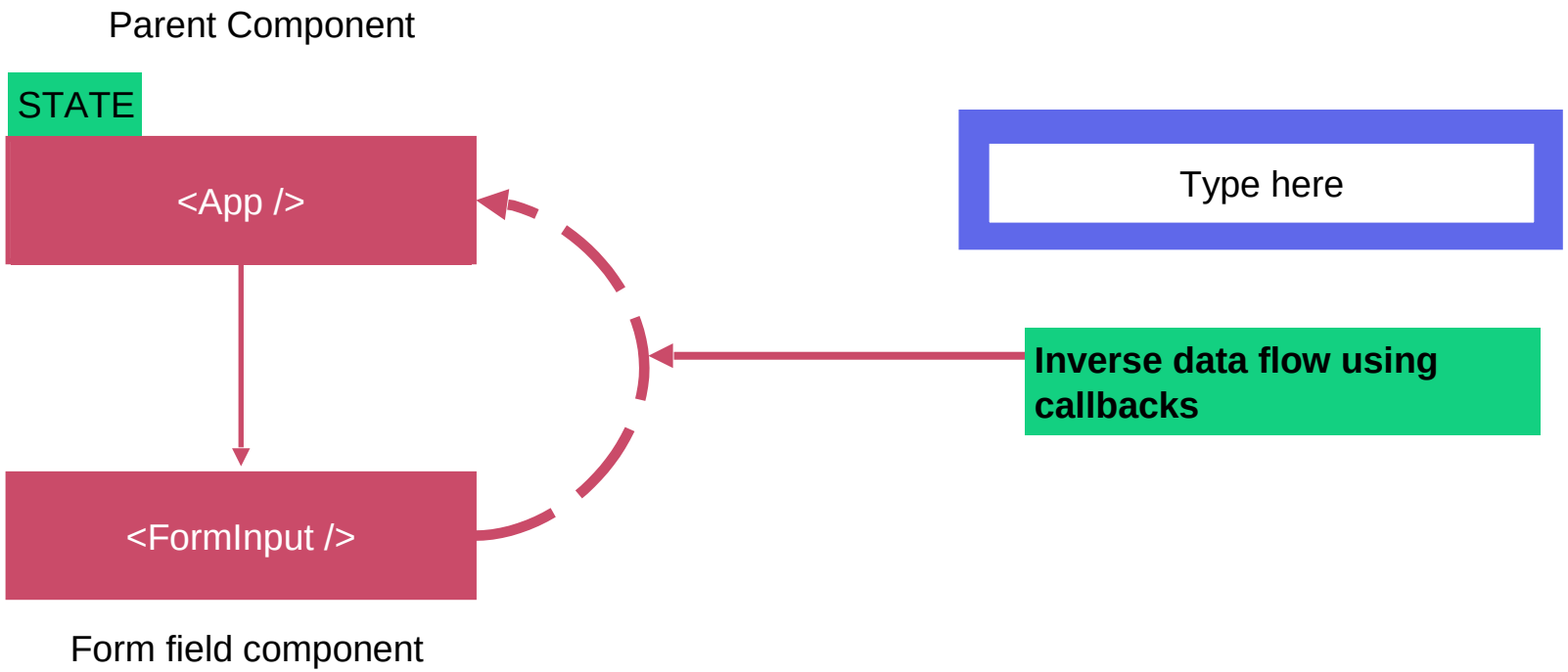
This is predictable!

UNDIRECTIONAL DATA FLOW



What if a component wants to send data back up to the parent??

UNIDIRECTIONAL DATA FLOW



SLIDE NAME

```
const FormComponent = ({ changeHandler }) => {
  return (
    <input type="text" onChange={event =>
changeHandler(event.target.value)} />
  );
};



class App extends Component {
  state = {
    username: ""
  };
  updateUsername = val => this.setState({ username: val });
  render() {
    return <FormComponent
    changeHandler={v                                    al =>
this.updateUsername(val)} />;
```

USING PROPS TO SHARE DATA

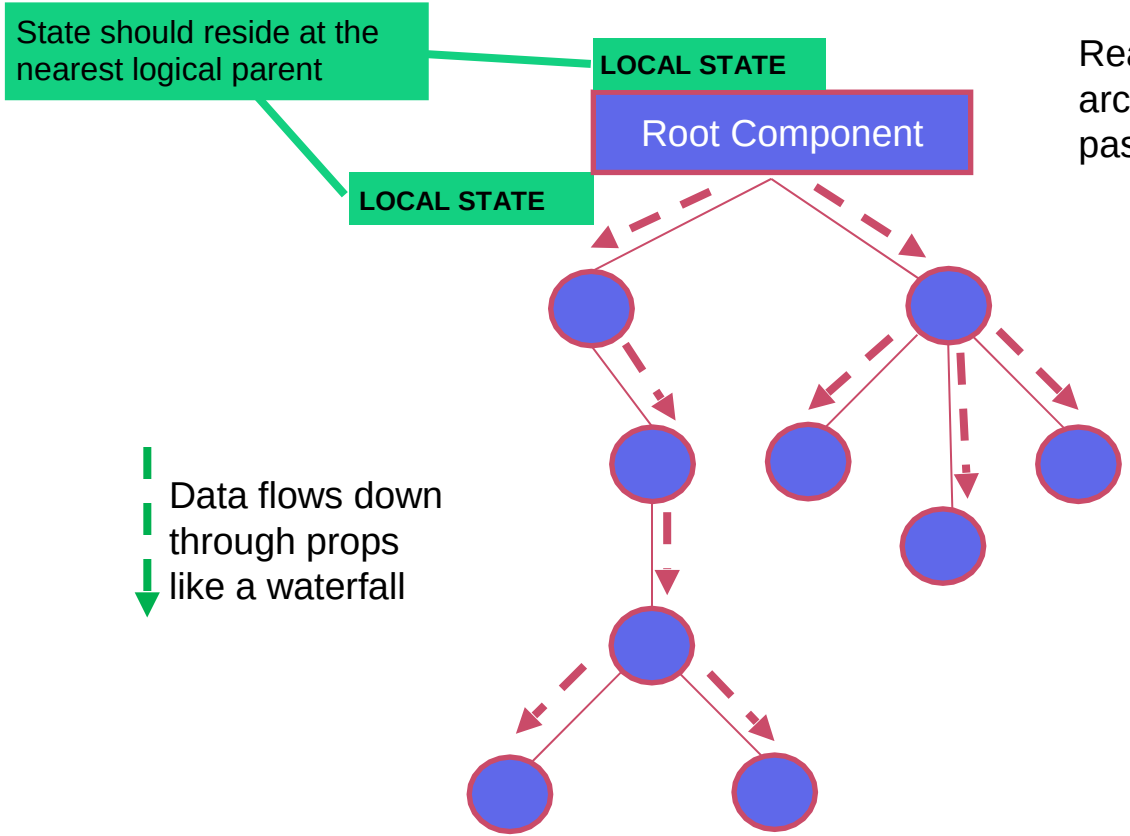
SLIDE NAME

Data from a parent component

```
<Score value={score} />
```

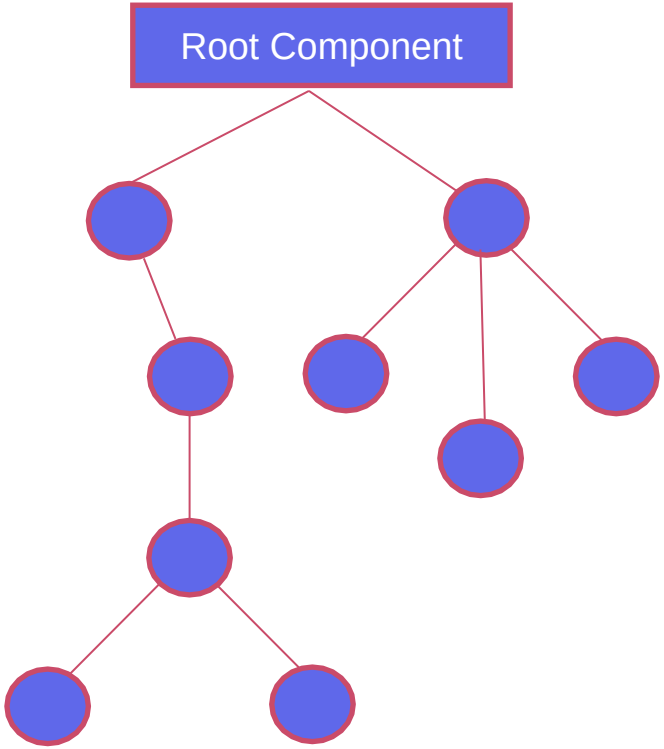
Props allow parent components to pass data to child components. They're easy to work with and simple to understand.

?



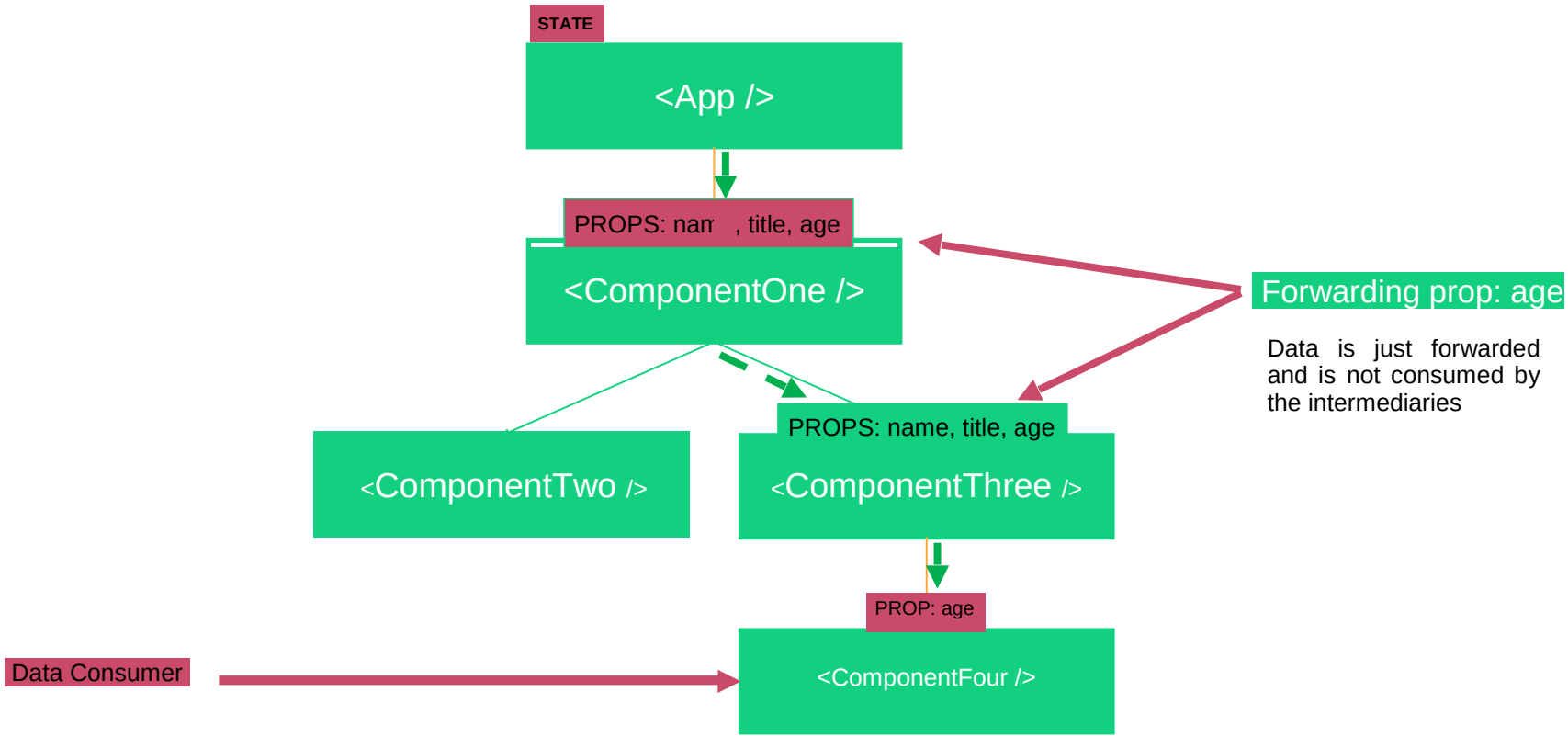
React strongly affirms a one-way data architecture that relies on the use of props for passing data down to children.

SLIDE NAME

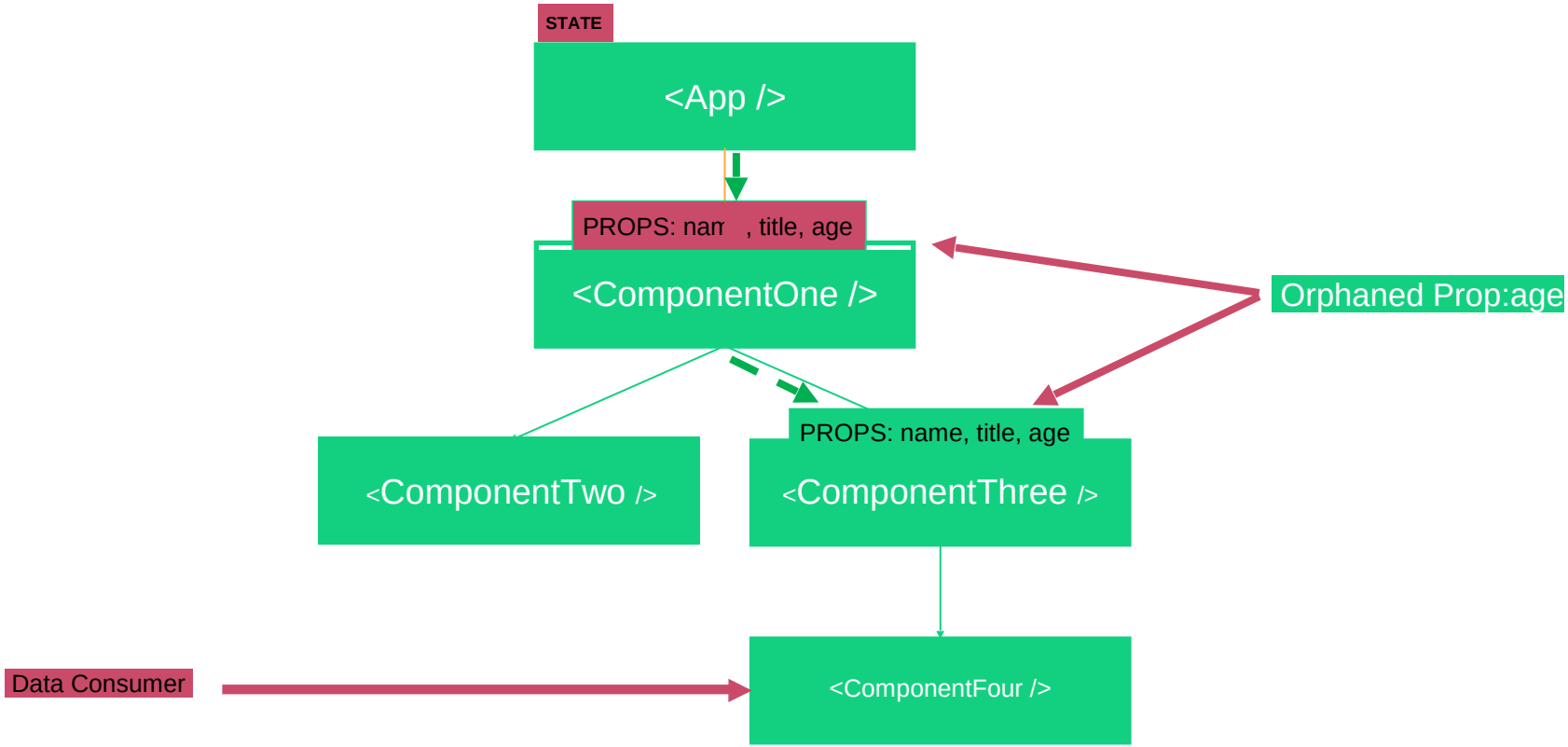


Architecting data flow in a deeply nested application can be quite challenging

PROP DRILLING

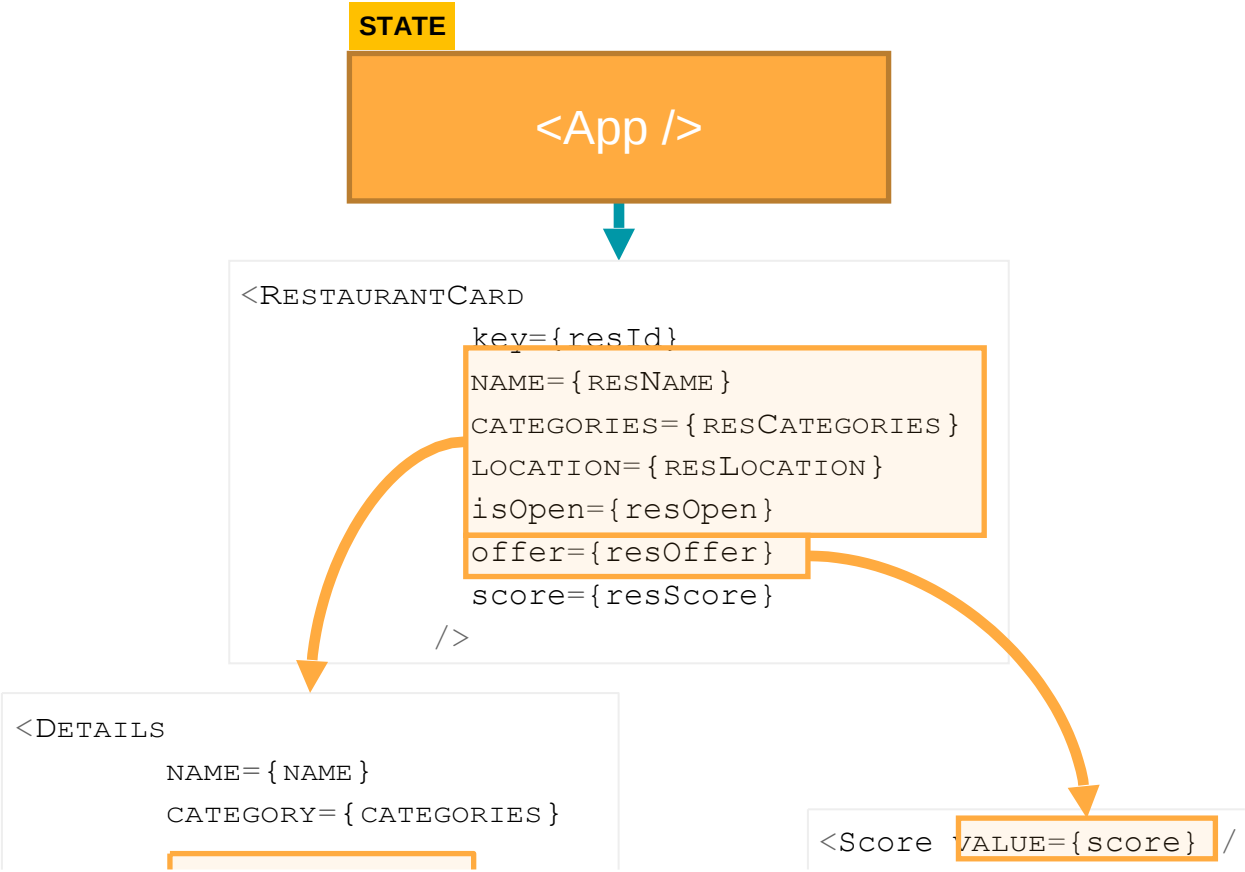


PROP DRILLING



CODE DEMO

SLIDE NAME


STATE
<App />
<RestaurantCard
key={resId}
name={resName}
categories={resCategories}
location={resLocation}
isOpen={resOpen}
offer={resOffer}
score={resScore}
/>
<Details
name={name}
category={categories}
<Score value={score} />

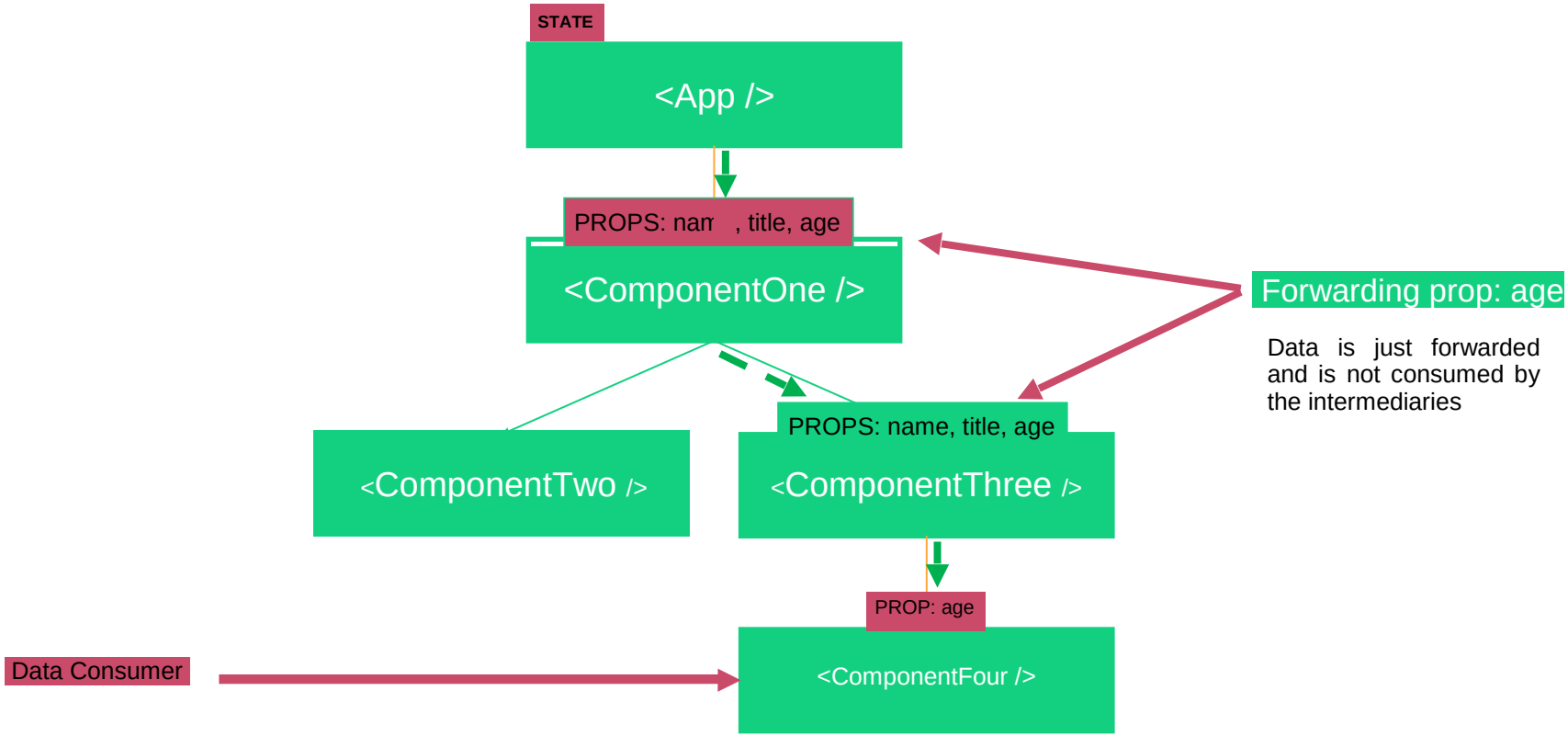
SLIDE NAME

LOCATION=

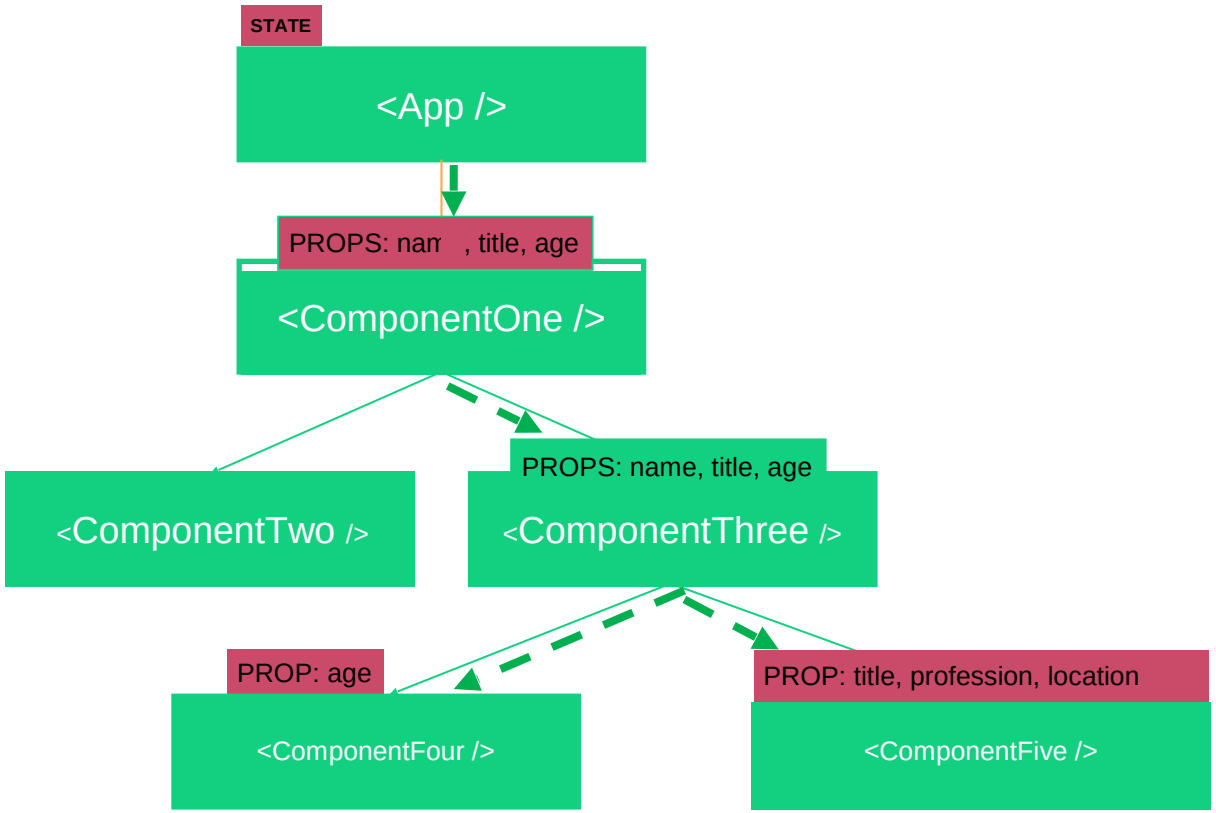
{LOCATION

}

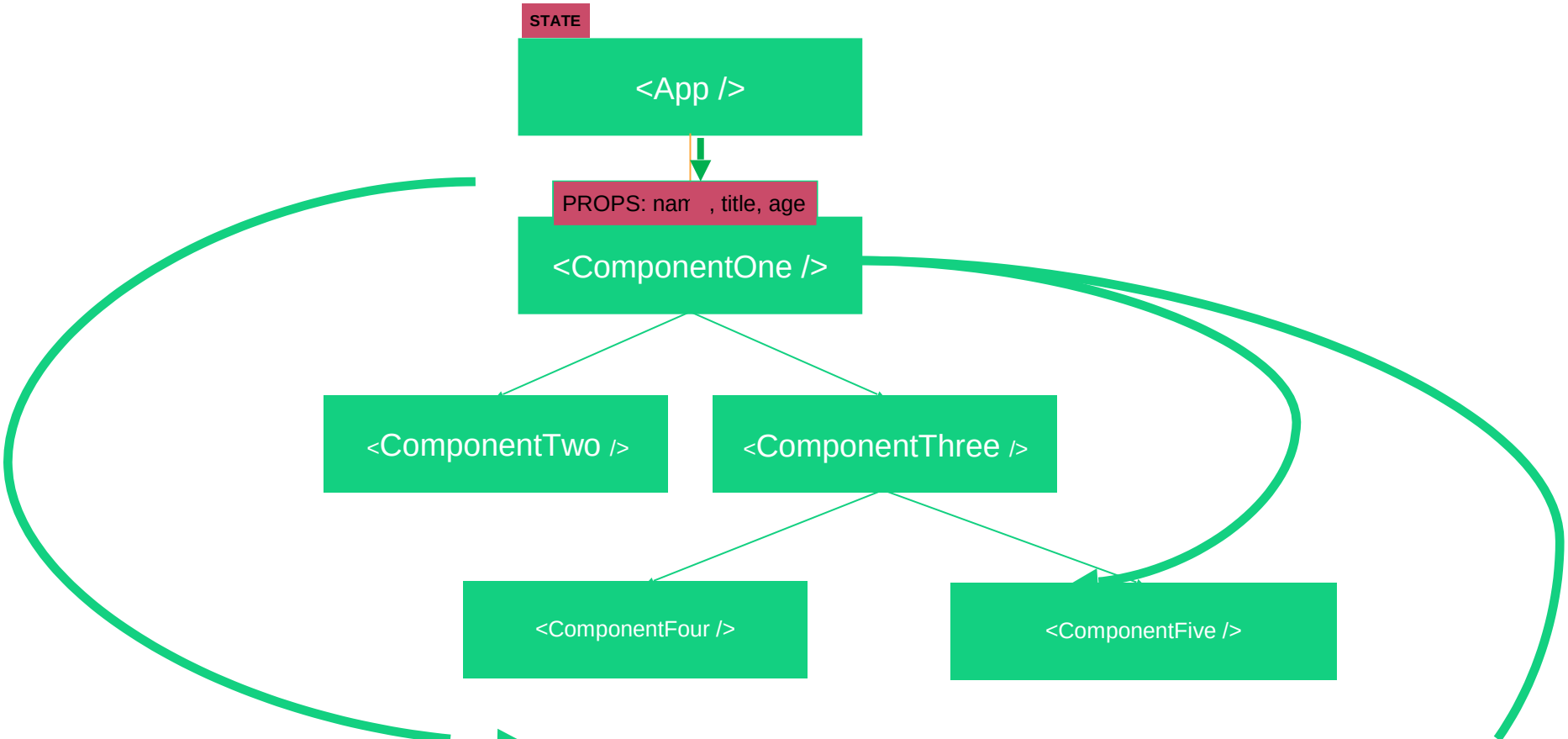
PROP DRILLING



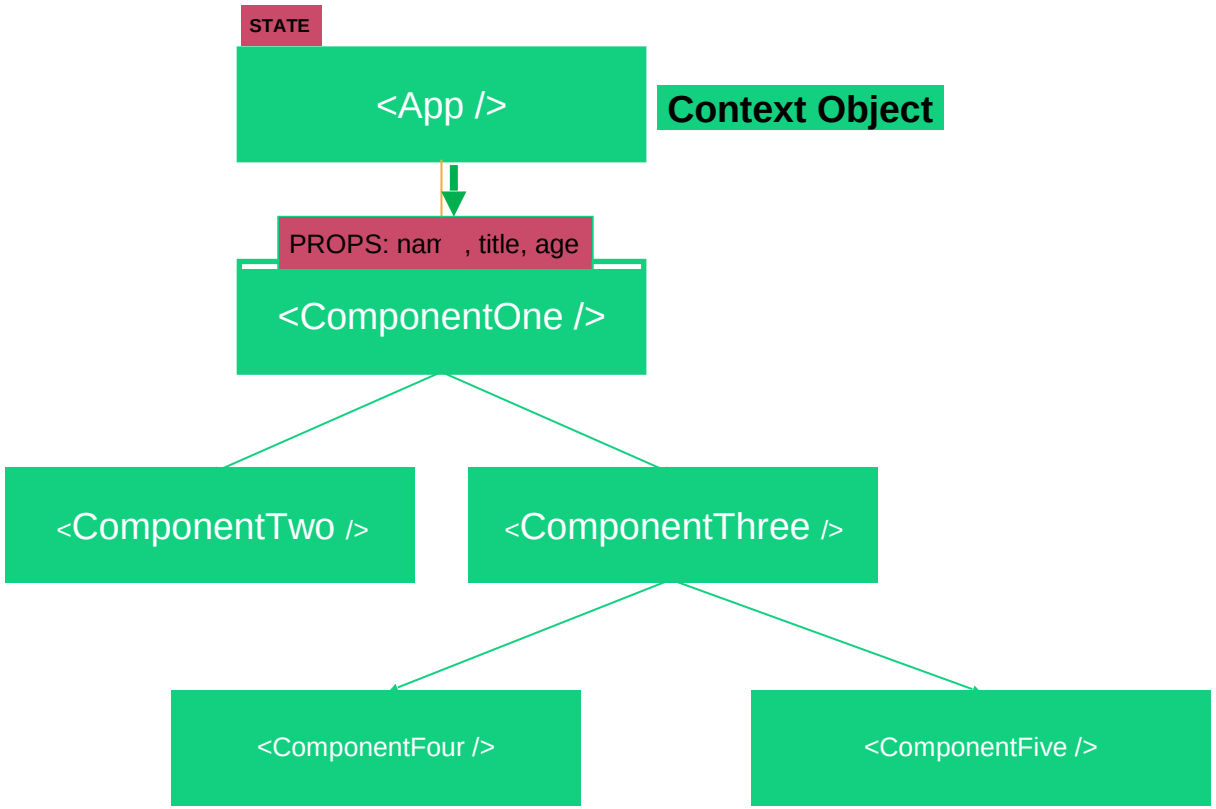
PROP DRILLING



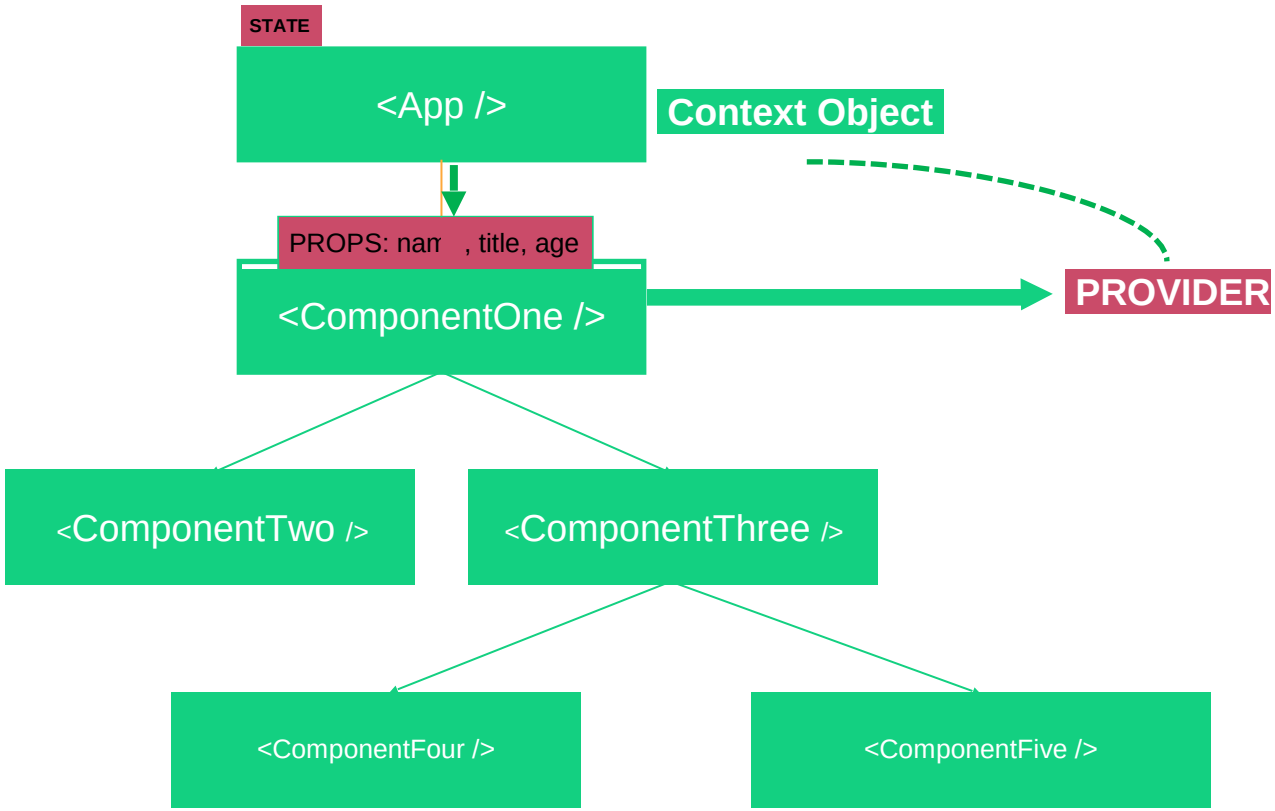
PROP DRILLING



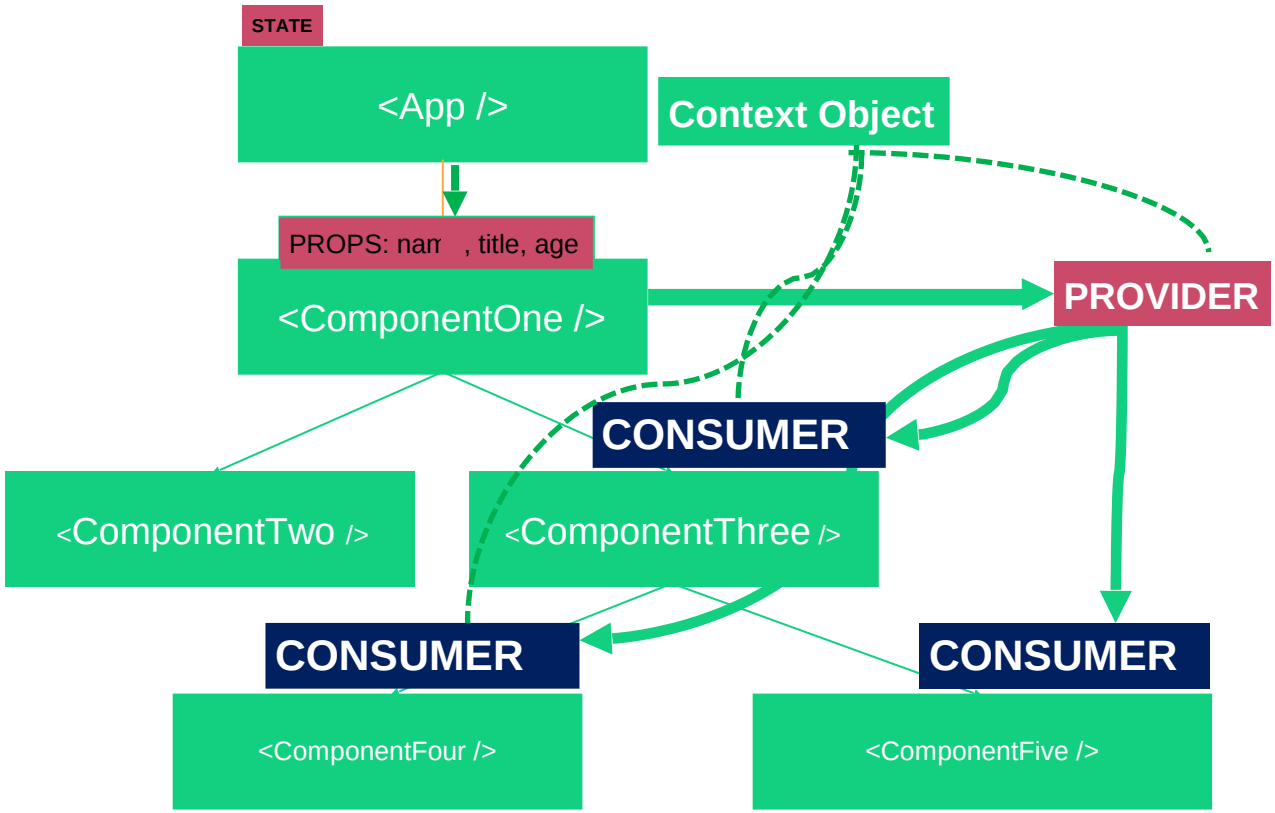
CONTEXT API


STATE
<App />
Context Object
PROPS: nam , title, age
<ComponentOne />
<ComponentTwo />
<ComponentThree />
<ComponentFour />
<ComponentFive />

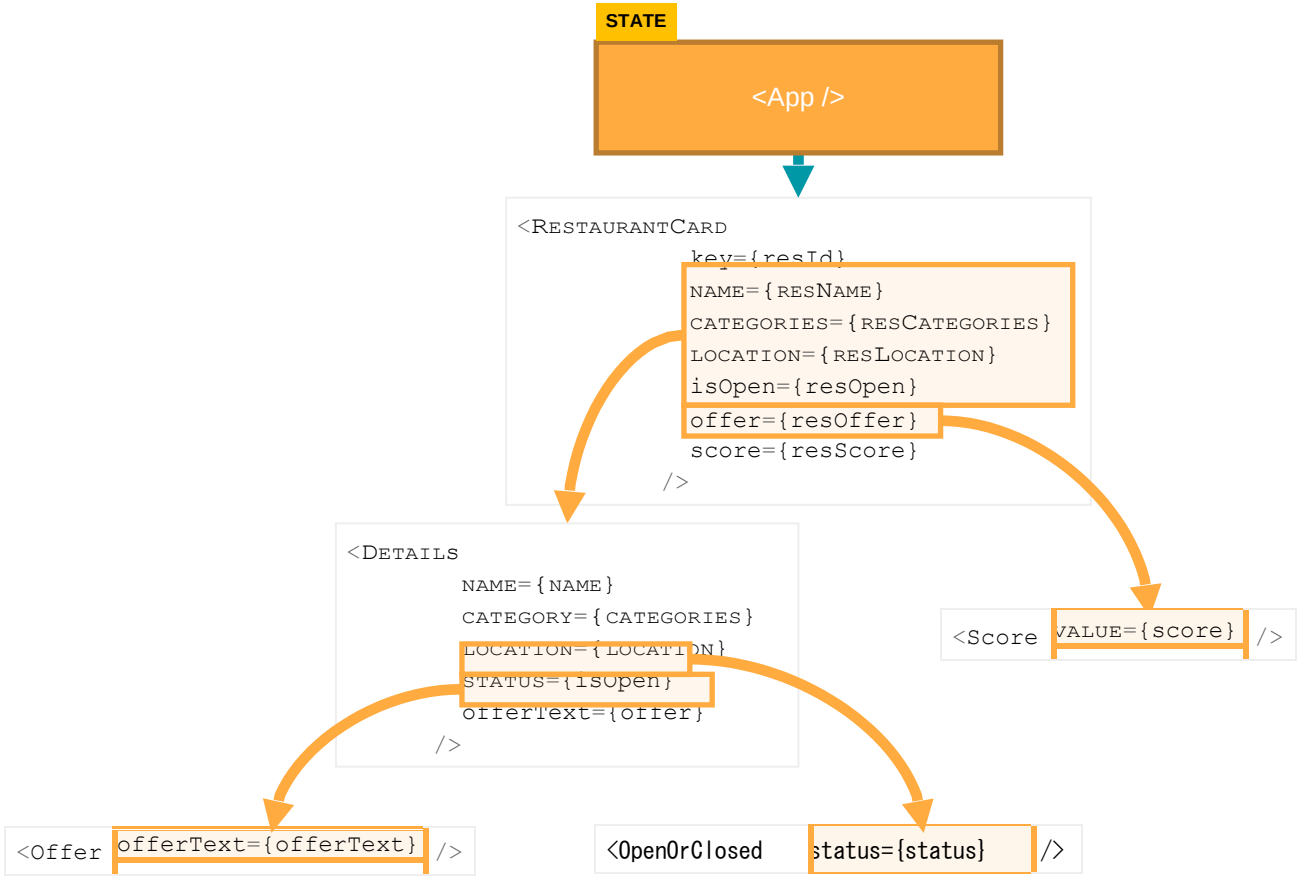
CONTEXT API


STATE
<App />
Context Object
PROPS: nam , title, age
<ComponentOne />
PROVIDER
<ComponentTwo />
<ComponentThree />
<ComponentFour />
<ComponentFive />

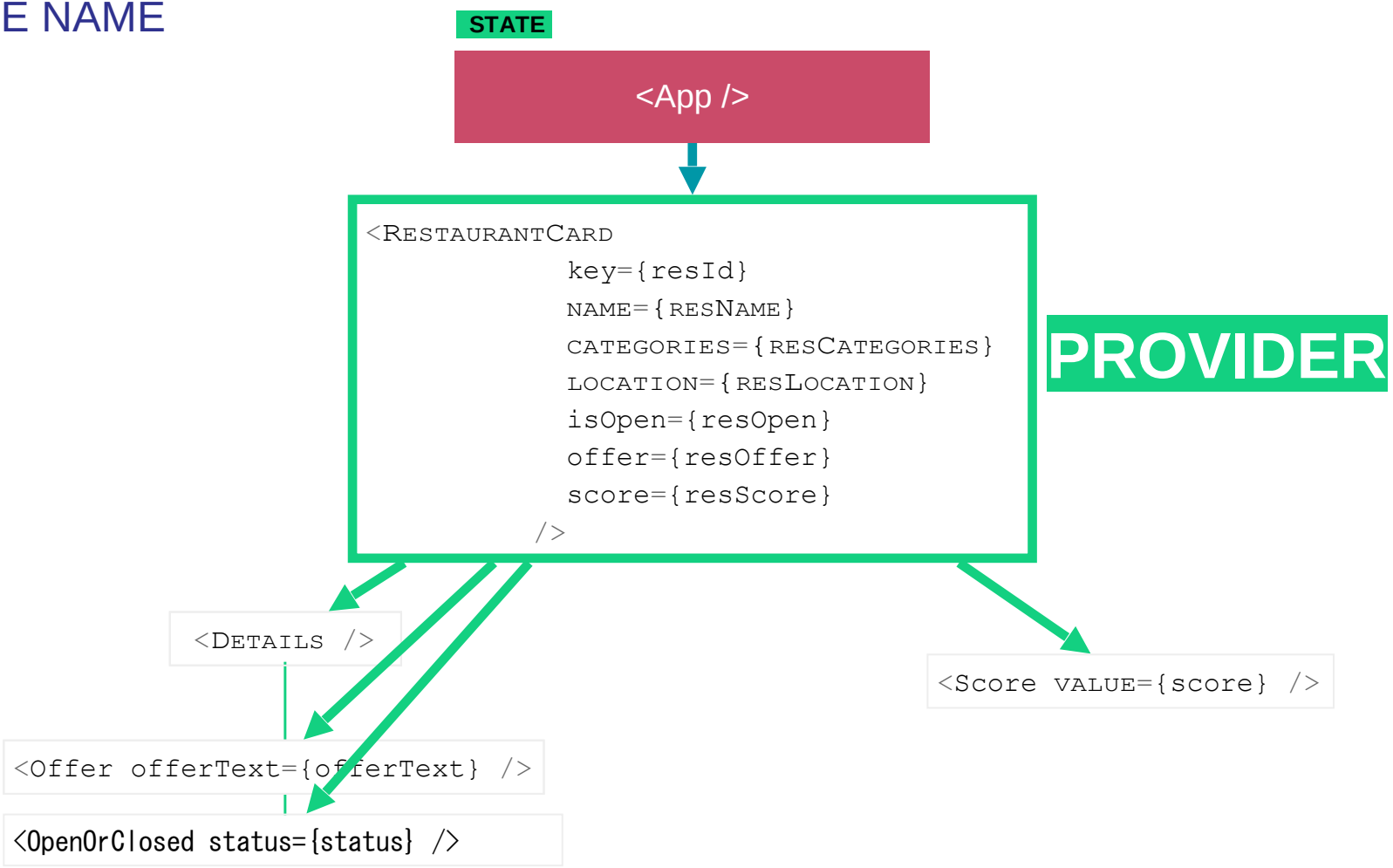
CONTEXT API


STATE
<App />
Context Object
PROPS: nam , title, age
<ComponentOne />
PROVIDER
CONSUMER
<ComponentTwo />
<ComponentThree />
CONSUMER
CONSUMER
<ComponentFour />
<ComponentFive />

SLIDE NAME


STATE
<App />
<RestaurantCard
key={resId}
name={resName}
categories={resCategories}
location={resLocation}
isOpen={resOpen}
offer={resOffer}
score={resScore}
/>
<Details
name={name}
category={categories}
location={location}
status={isOpen}
offerText={offer}
/>
<Score value={score} />
<Offer offerText={offerText} />
<OpenOrClosed status={status} />

SLIDE NAME


STATE
<App />
<RestaurantCard
key={resId}
name={resName}
categories={resCategories}
location={resLocation}
isOpen={resOpen}
offer={resOffer}
score={resScore}
/>
PROVIDER
<Details />
<Score value={score} />
<Offer offerText={offerText} />
<OpenOrClosed status={status} />

SLIDE NAME

```
class Score extends Component {
  static contextType = RestaurantContext;
  state = {
    show: false
  };
  componentDidMount() {
                if (this.context.score) {
                                this.setState({show: true});
                }
  }
  render() {
    return this.state.show ? <div>{this.context.score}</div> : null;
  }
}
```

Using Context with Class Components

SLIDE NAME

```
const myContext = createContext({
  panel: false,
  showConfig: true
});
```

SLIDE NAME

```
const myContext = createContext({
  storeDate = () => {}
});
```

```
const ParentComponent = () => {
  const [color, setColor] = useState("Black");
  const changeColor = (color) => {
  setColor(color);
  }
  return <MyContext.Provider value={{color, changeColor}}>
            <ChildComponent />
         </MyContext.Provider>
}
```

```
const ChildComponent = () => {
  return <MyContext.Consumer>
    {({color, changeColor}) => {
      // Consume color
      // Run changeColor("Red") to change color in the context
    }}
  </MyContext.Consumer>
}
```

SLIDE NAME

```
const Offer = () => {
  const [showOffer, setShowOffer] = useState(false);
  return (
    <RestaurantContext.Consumer>
      {({offer: offerText}) => (
        <div className="res-offers" onClick={() => setShowOffer(true)}>
          {showOffer
            ? offerText
              ? offerText
              : "No offers available"
            : "Get Offers"}
        </div>
      )}
    </RestaurantContext.Consumer>
  );
};
```

SLIDE NAME

Context API is great but consider it after exploring regular options like compositional strategies, render props, HOCs etc.

thank you!