React-REDUX

Redux is for JavaScript applications

Redux is not tied to React

Can be used with React, Angular, Vue or even vanilla JavaScript

Redux is a library for JavaScript applications

Redux is a state container

Redux stores the state of your application

Consider a React app - state of a component

State of an app is the state represented by all the individual components of that app

<u>LoginFormComponent</u>

```
state = {
  username: ‘ ’,
  password: ‘ ’,
  submitting: false
}
```

<u>UserListComponent</u>

```
state = {
  users: [ ]
}
```

<u>Application</u>

```
state = {
  isUserLoggedIn: true,
  username: ‘Vishwas’,
  profileUrl: ‘ ’,
  onlineUsers: [ ],
  isModalOpened: false
}
```

Redux is predictable

Predictable in what way?

Redux is a state container

The state of the application can change

Ex: todo list app – item (pending) → item (completed)

In redux, all state transitions are explicit and it is possible to keep track of them

The changes to your application's state become predictable

Redux

"Redux is a predictable state container for JavaScript apps“

Manage the state of your application in a predictable way, redux can help you.

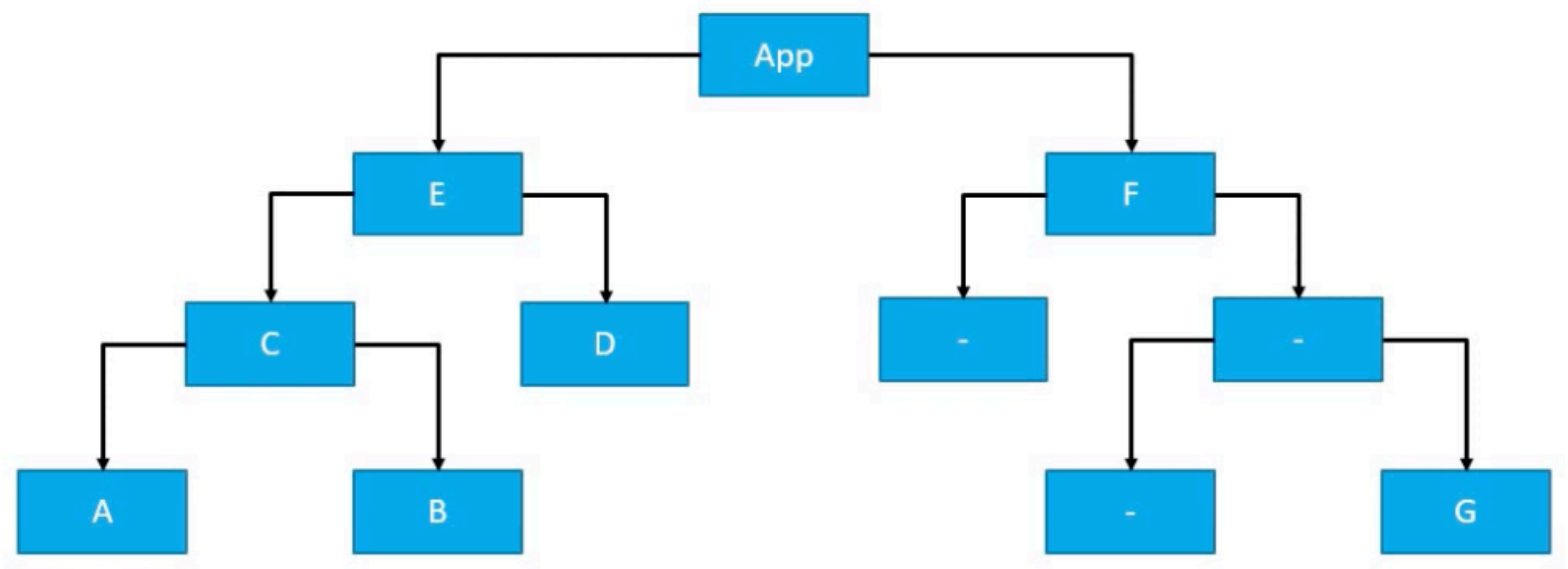
React + Redux?

Why would we want to use redux in a react application?

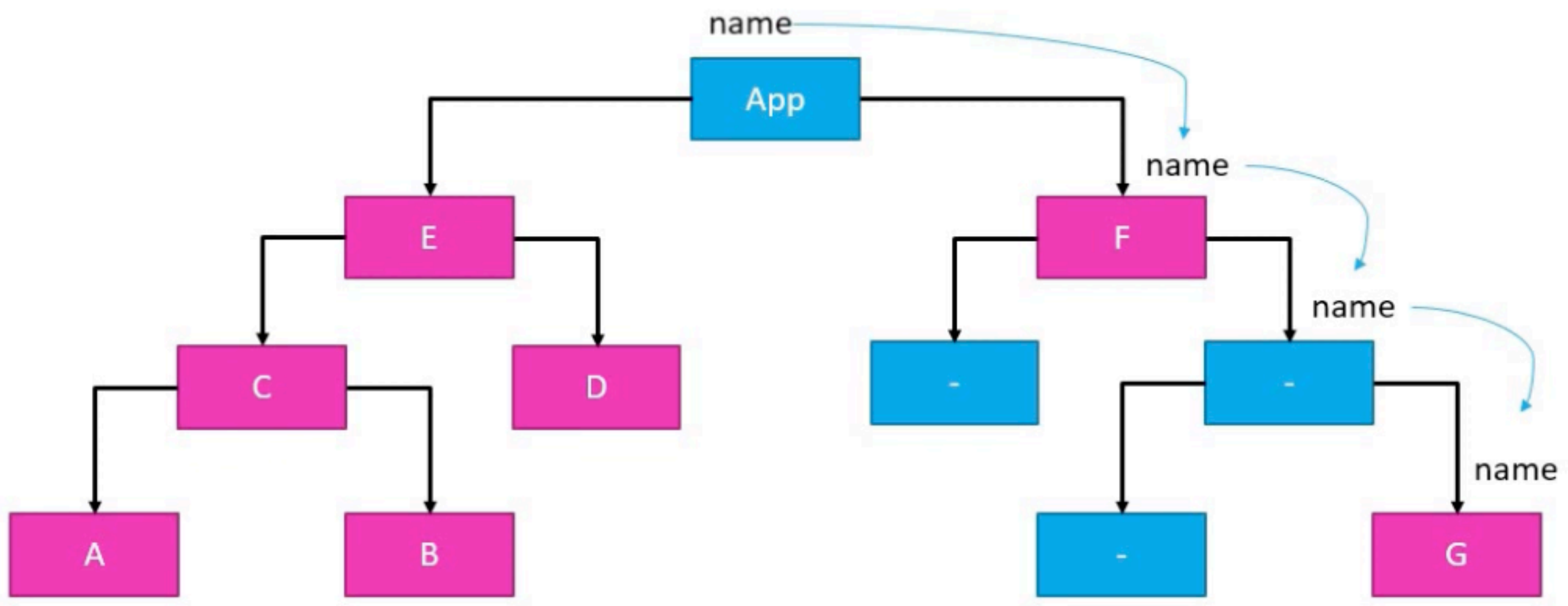
Components in React have their own state

Why do we need another tool to help manage that state?

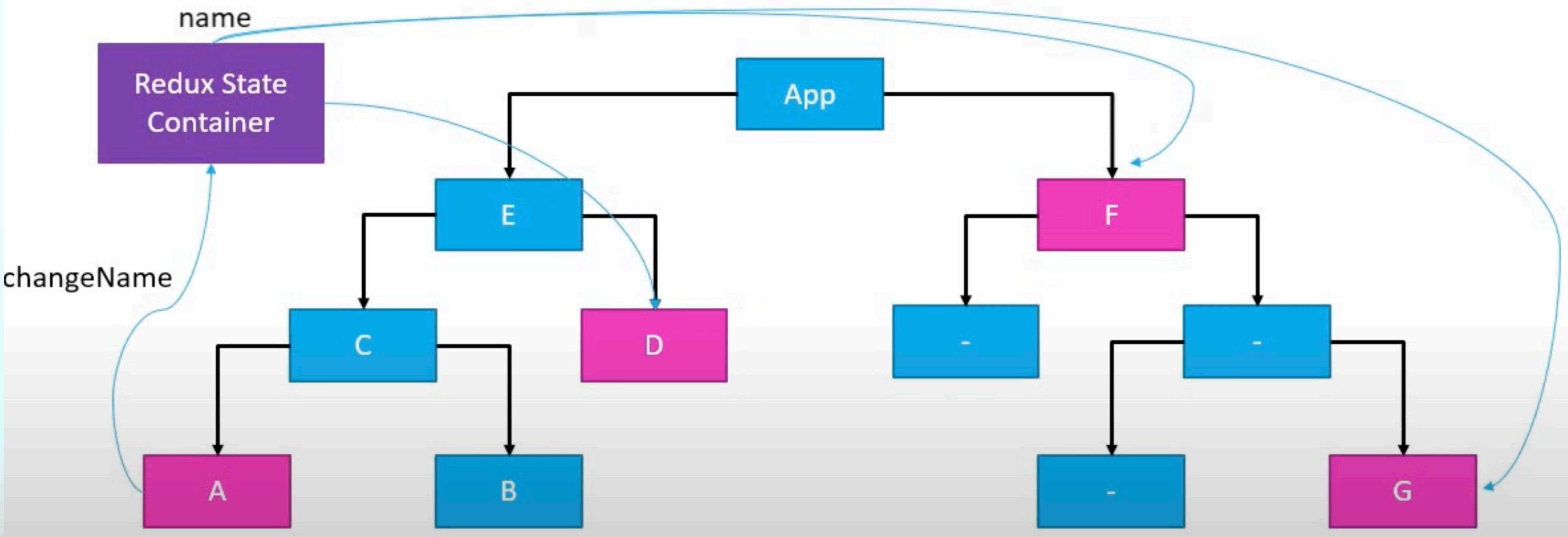
State in a React App



State in a React App



React + Redux



Do we really have a problem?

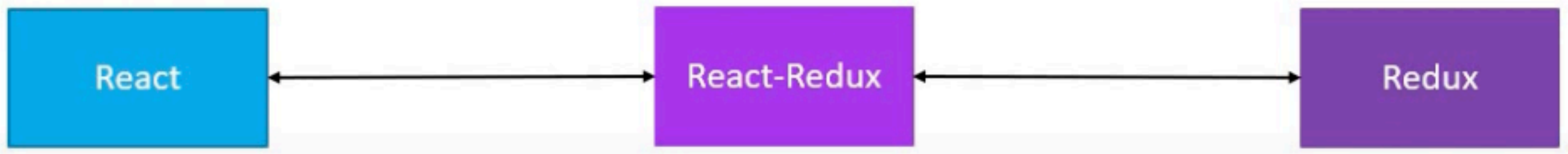
React context – Prevents prop drilling

useContext + useReducer ?

Redux 1.0 – August 2015

React-Redux

React-Redux is the official Redux UI binding library for React



Summary

React is a library used to build user interfaces

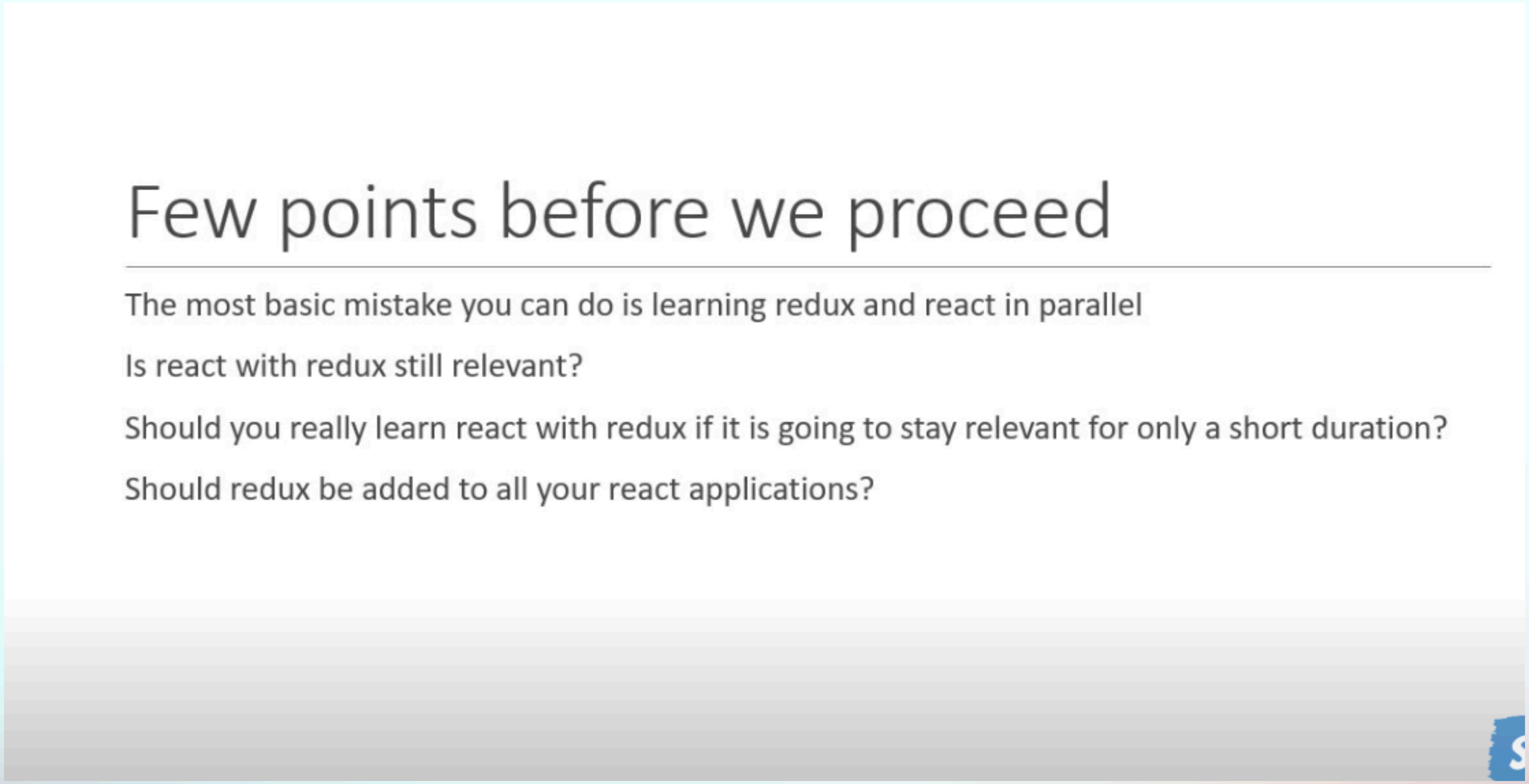
Redux is a library for managing state in a predictable way in JavaScript applications

React-redux is a library that provides bindings to use React and Redux to
in an application

Few points before we proceed

The most basic mistake you can do is learning redux and react in parallel

Is react with redux still relevant?



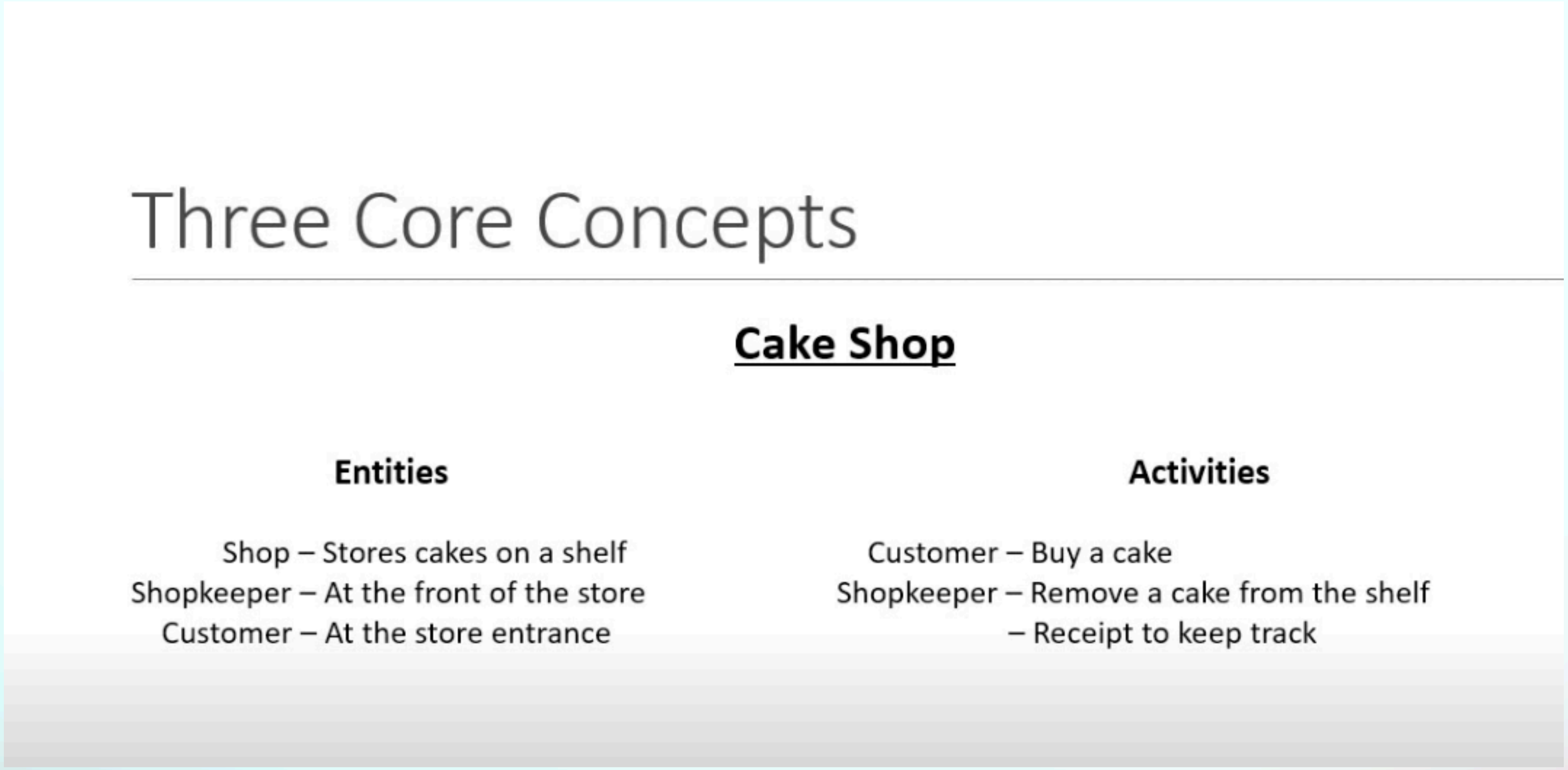
Few points before we proceed

The most basic mistake you can do is learning redux and react in parallel

Is react with redux still relevant?

Should you really learn react with redux if it is going to stay relevant for only a short duration?

Should redux be added to all your react applications?



Three Core Concepts

<u>Cake Shop</u>

Entities

Shop – Stores cakes on a shelf
Shopkeeper – At the front of the store
Customer – At the store entrance

Activities

Customer – Buy a cake
Shopkeeper – Remove a cake from the shelf
– Receipt to keep track

Three Core Concepts contd.

Cake Shop Scenario	Redux	Purpose
Shop	Store	Holds the state of your application
Intention to BUY_CAKE	Action	Describes what happened
Shopkeeper	Reducer	Ties the store and actions together

A **store** that holds the state of your application.

An **action** that describes the changes in the state of the application.

A **reducer** which actually carries out the state transition depending on the action.

Entities

1. **Shop**:
    - Represents the overall store, inventory, and operations.
    - Stores data such as available cakes, prices, and promotions.
2. **Shopkeeper**:
    - Manages the inventory and handles the backend operations.
    - May perform activities such as adding new cakes, updating prices, and monitoring sales.
3. **Customer**:
    - Interacts with the shop to browse, select, and purchase cakes.
    - Has activities related to choosing products, adding items to a cart, and placing orders.
    -

Activities for Customer in Redux

To handle customer-related activities, you'll typically set up a Redux store with state slices, actions, and reducers. Here’s how this could look:

1. **State Slices**:
   - `inventory`: Stores available cakes and their details.
   - `cart`: Keeps track of items added by the customer.
   - `orders`: Stores order history after a purchase.
2. **Actions**:
   - `BROWSE_CAKES`: Action to display available cakes.
   - `ADD_TO_CART`: Adds a selected cake to the customer’s cart.
   - `REMOVE_FROM_CART`: Removes a selected item from the cart.
   - `PLACE_ORDER`: Finalizes the order, clearing the cart and adding the order to the history.
   -

Reducers:

- `inventoryReducer`: Handles actions related to browsing and updating available cakes.
- `cartReducer`: Handles adding and removing cakes from the cart.
- `orderReducer`: Manages the process of finalizing and storing completed orders.
-

```
// Actions

const ADD_TO_CART = 'ADD_TO_CART';

const REMOVE_FROM_CART = 'REMOVE_FROM_CART';

const PLACE_ORDER = 'PLACE_ORDER';
```

```
// Action Creators

export const addToCart = (cakeId) =>



({ type: ADD_TO_CART, payload: cakeId, });
```

```
export const removeFromCart = (cakeId) =>


 ({ type: REMOVE_FROM_CART, payload: cakeId,

});
```

```
export const placeOrder = () =>

({ type: PLACE_ORDER, });
```

```
// Reducers

const cartReducer = (state = [], action) => {

switch (action.type) {


case ADD_TO_CART: return [...state, action.payload];


case REMOVE_FROM_CART:

return state.filter((id) => id !== action.payload);

case PLACE_ORDER:return [];

default:
 return state; } };
```

```
/ Example state initialization

const initialState = {

 inventory: [{ id: 1, name: 'Chocolate Cake', price: 20 },
{ id: 2, name: 'Vanilla Cake', price: 15 }],

cart: [], orders: [], };
```

```
// Usage in a component

 import { useSelector, useDispatch } from 'react-redux';
const CakeShopComponent = () => {const dispatch = useDispat
const cakes = useSelector((state) => state.inventory);
const cart = useSelector((state) =>state.cart);
const handleAddToCart = (cakeId) =>
{ dispatch(addToCart(cakeId)); };
( <div>
<h1>Cake Shop</h1> <ul>
{cakes.map((cake) => (
 <li key={cake.id}> {cake.name} - ${cake.price}
<button onClick={() => handleAddToCart(cake.id)}>Add to Cart<
button> </li> ))}
 </ul>
 <h2>Cart</h2> <ul>{cart.map((itemId) => ( <li key={itemId}
>{cakes.find((cake) => cake.id === itemId)?.name}</li> ))} </ul>
div> ); };
```