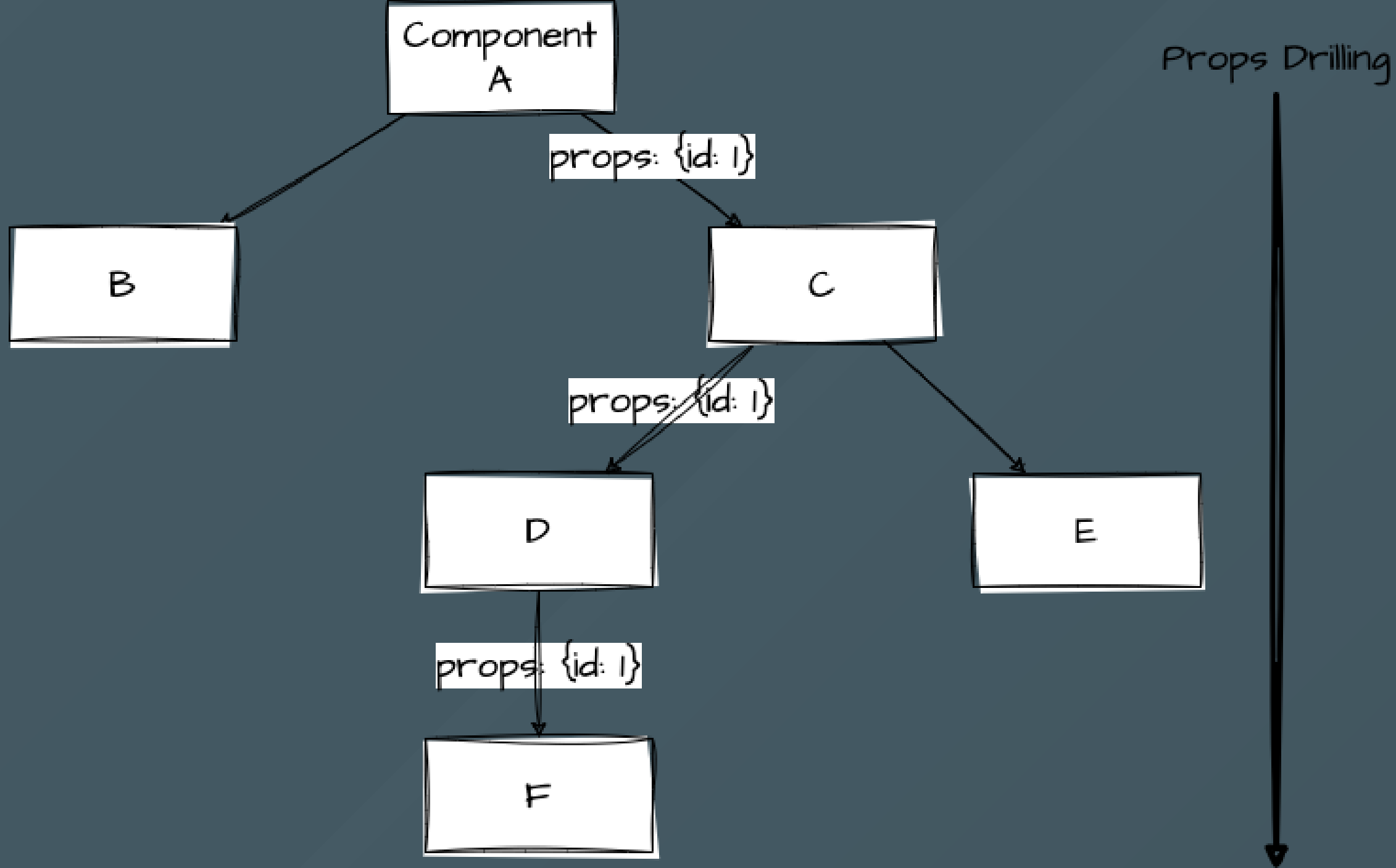


State Management

Introduction to State Management

- State management is a way to manage the state of your application
- Problem with React: state is local to a component, but sometimes you want to share state between components, so props drilling. To solve this problem, we can use state management libraries like **Redux**, MobX, etc., or React's Context API



Introduction to Redux

- What is Redux?
- Why Redux?
- How to use Redux in React?

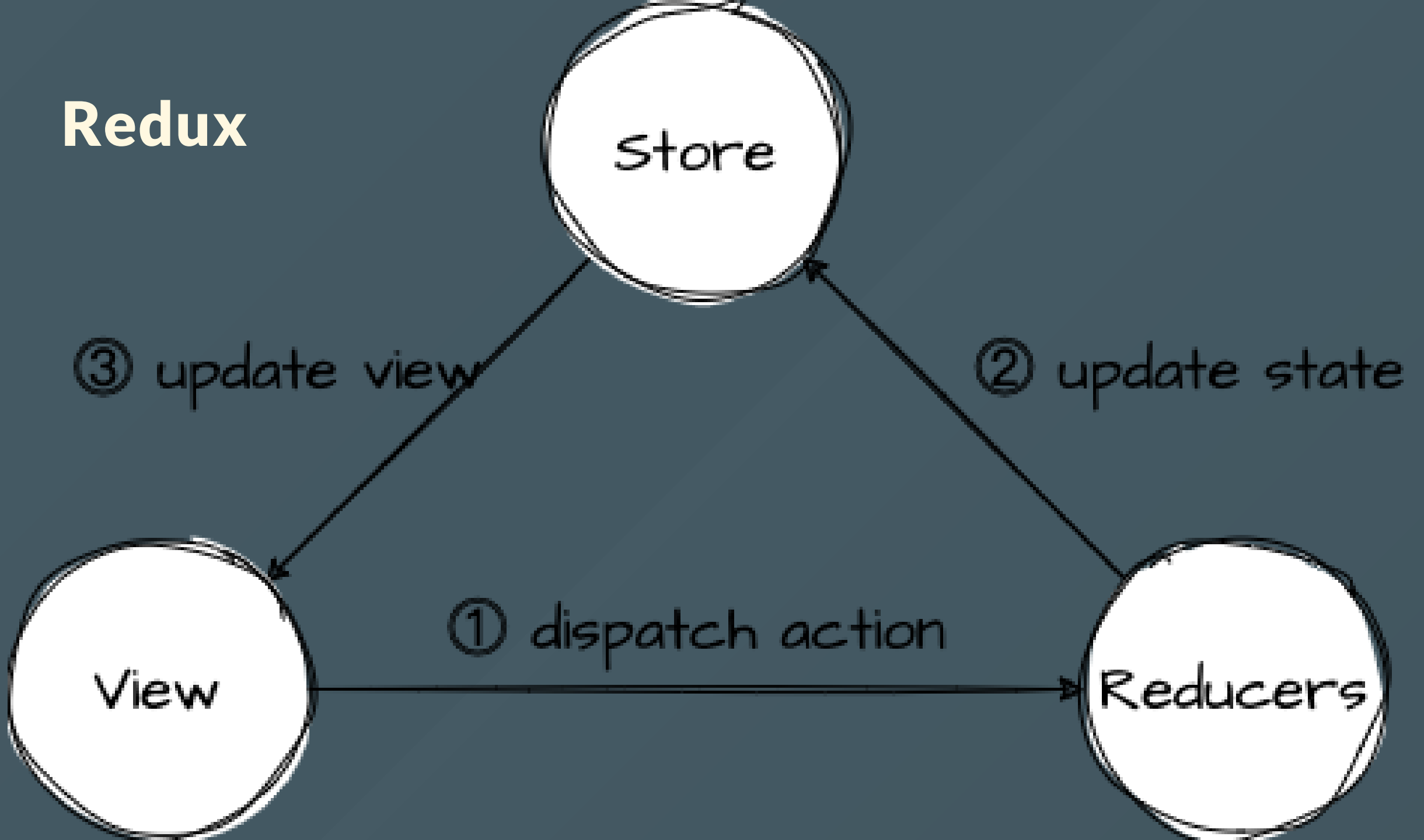
What is Redux?

- Redux is a predictable state container for JavaScript applications
- State transition machine
- For **any** JS application, not just React

Why Redux?

- Predictable
- Centralized
 - Store: single source of truth
- Debuggable
- Flexible
 - Middleware

Redux



Terms

- Store: single source of truth, only place to hold the state of your application
- Reducer: pure function that takes the previous state and an action, and returns the next state
- View: an app that connect store, React in our case
- Action: plain JS object that describes what happened
- Dispatch: only way to trigger a state change
- Slice: a collection of reducer login and actions for a single feature of your app (aka slice reducer)

Redux Principles

- Single source of truth
 - The state of your whole application is stored in an object tree within a single store
- State is read-only and immutable
 - The only way to change the state is to emit an action, an object describing what happened
- Changes are made with pure functions
 - To specify how the state tree is transformed by actions, you write pure reducers

Reducer

```
const initialState = {  
  count: 0  
};  
  
function reducer(state = initialState, action) {  
  switch (action.type) {  
    case 'INCREMENT':  
      return {  
        ...state,  
        count: state.count + 1  
      };  
    case 'DECREMENT': // ...  
    default: // ...  
  }  
}
```

Store

<p class="mark">redux-native.js</p> <div class="columns">

```
import { createStore } from 'redux';
import reducer from './reducer';

const store = createStore(reducer);

store.subscribe(() => {
  console.log(store.getState());
});

store.dispatch({ type: 'INCREMENT' });
store.dispatch({ type: 'DECREMENT' });
```

- `dispatch` is the only way to trigger a state change

Action

```
const increment = () => ({  
  type: 'INCREMENT'  
});
```

- Action is an object that describes what happened
- Action creators are functions that create actions
- `dispatch` can take an action object or an action creator

Redux Toolkit (RTK)

<p class="mark">redux-rtk.js</p>

- RTK is the official, opinionated, batteries-included toolset for efficient Redux development
- RTK is a wrapper around Redux
- RTK provides utilities that simplify common Redux use cases, including store setup, defining reducers, immutable update logic, and even creating entire "slices" of state at once
- `createSlice`, `configureStore`

Redux in React

<p class="mark"> redux-class.jsx redux-hooks.jsx </p>

- `react-redux` is the official Redux UI binding library for React
- `Provider` component
- hooks: `useSelector`, `useDispatch`
- class components: `connect` HOC

Redux in Class Components (legacy)

```
import { connect } from 'react-redux';

class Counter extends React.Component {}

const mapStateToProps = (state) => ({
  count: state.count
});

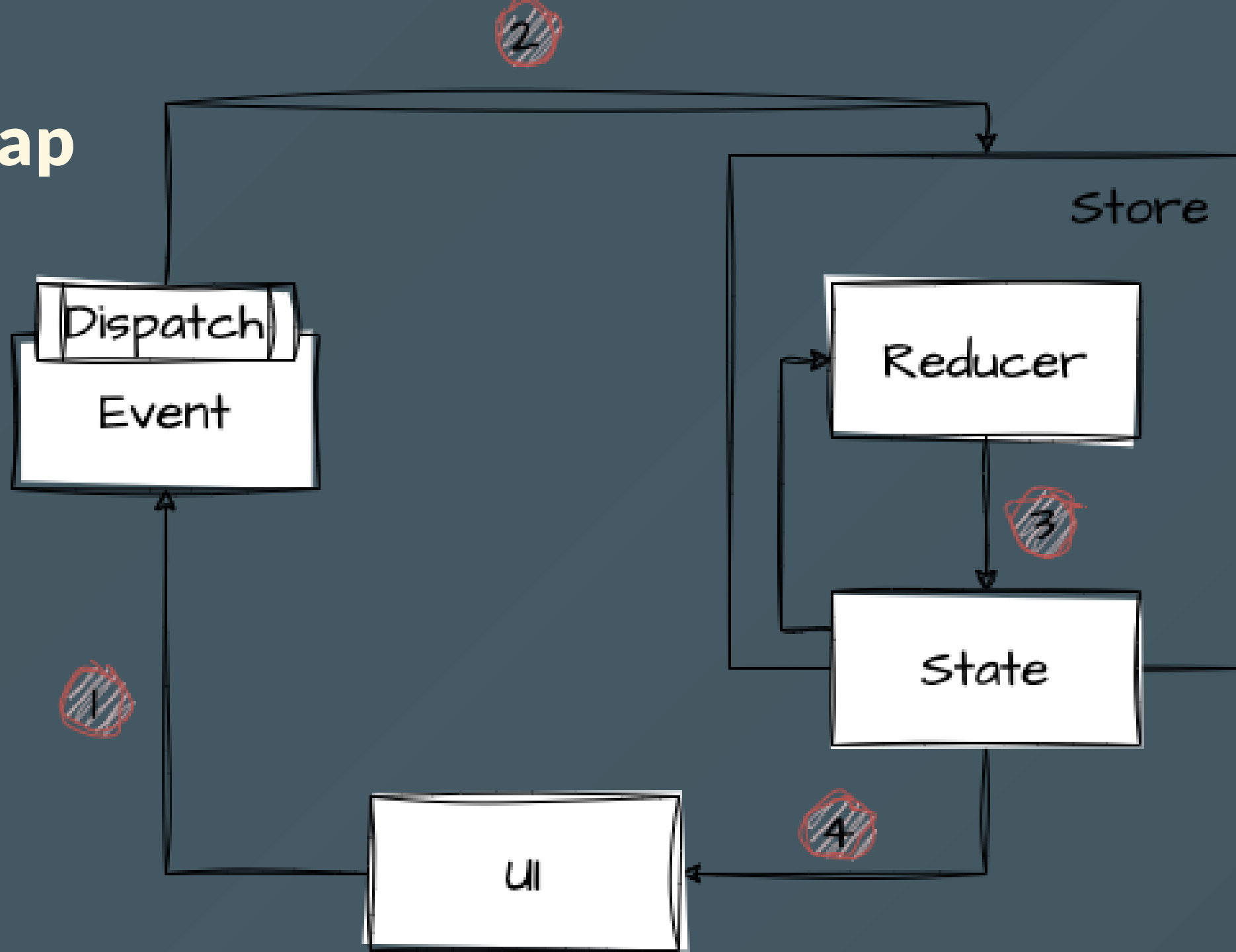
const mapDispatchToProps = (dispatch) => ({
  increment: () => dispatch({ type: 'INCREMENT' }),
  decrement: () => dispatch({ type: 'DECREMENT' })
});

export default connect(mapStateToProps, mapDispatchToProps)(Counter);
```


Redux devtools

- Browser extension
- Time travel
- Very useful for debugging
- Setup Redux devtools
 - `npm install --save-dev redux-devtools-extension`
 - in RTK `configureStore`:
`devTools: process.env.NODE_ENV !== 'production'`

Recap



React Reducer Hook

<p class="mark">Reducer.jsx</p>

- `useReducer` is a React hook that accepts a reducer function and an initial state, and returns the current state paired with a `dispatch` method

React Context API

<p class="mark">Context.jsx</p>

- Context provides a way to pass data through the component tree without having to pass props down manually at every level
- `createContext`, `Context.Provider`, `useContext`