Redux

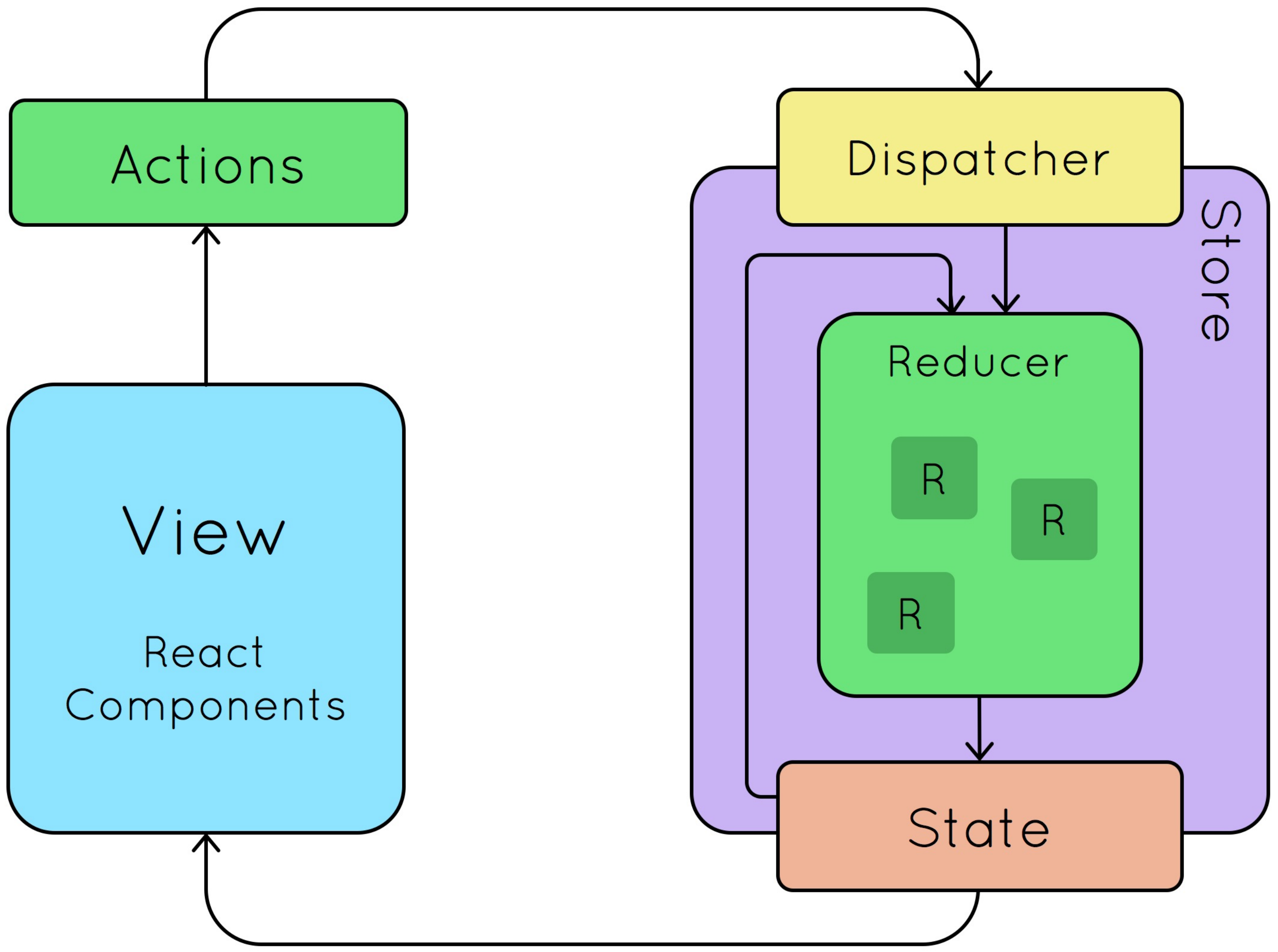
React Redux

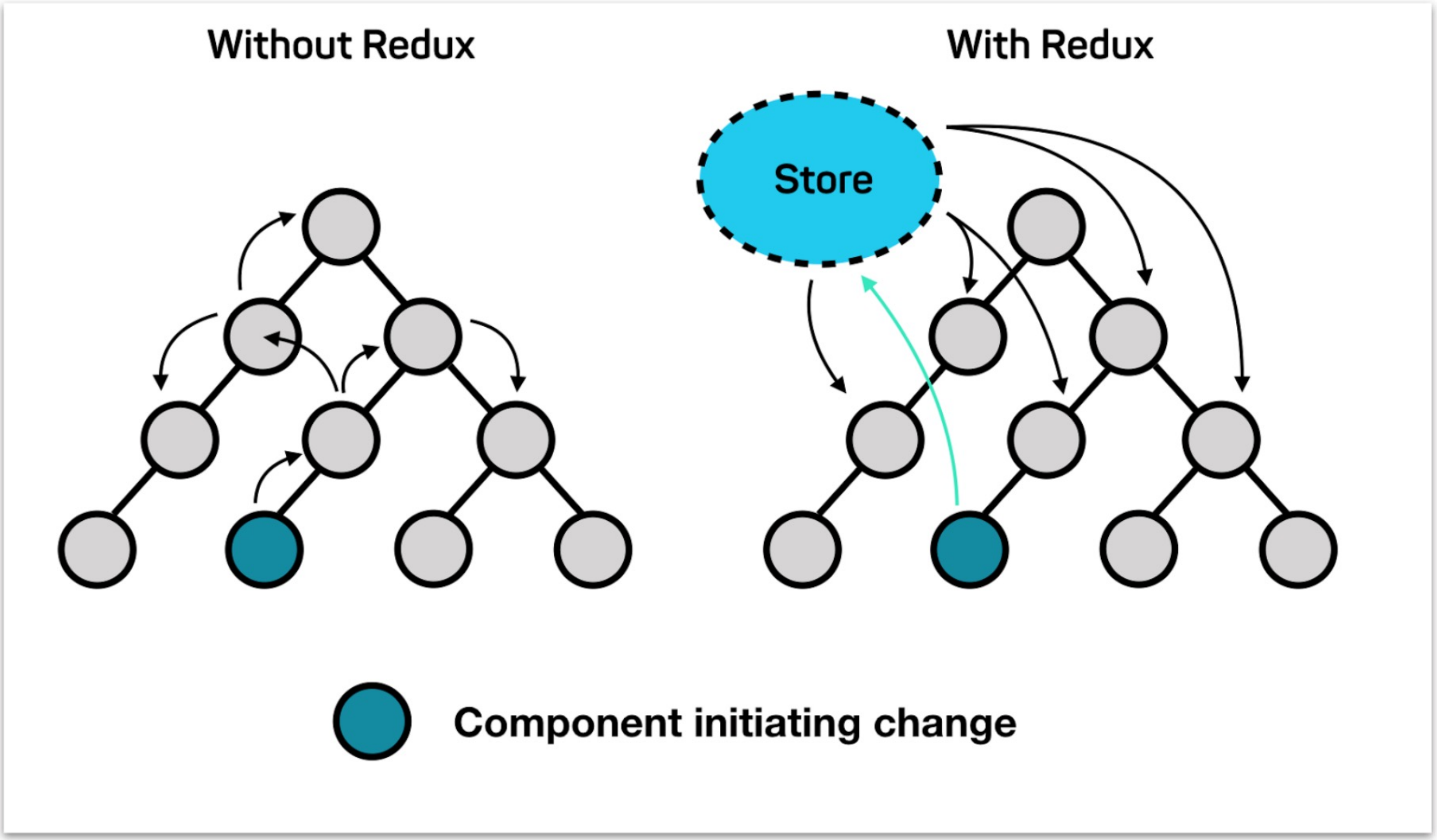
- **Redux** is a predictable state container designed to help you write JavaScript apps that behave consistently across client, server, and native environments and are easy to test. While it's mostly **used** as a state management tool with React, you can **use it** with any other JavaScript framework or library.

- `npm install redux`
- `npm install @reduxjs/toolkit`

Centralized

- Having a single store and single state tree enables:
  - Logging of all updates.
  - API handling.
  - Undo/Redo
  - State persistance.


Actions
Dispatcher
Store
Reducer
R
R
R
View
React Components
State


Without Redux
With Redux
Store
Component initiating change

Redux Toolkit

- Redux Toolkit includes the Redux core, as well as other key packages we feel are essential for building Redux applications (such as Redux Thunk and Reselect).
- **Redux Toolkit** is our official, opinionated, batteries-included toolset for efficient Redux development. It is intended to be the standard way to write Redux.
- **Redux Toolkit** was originally created to help address three common concerns about Redux:
  - "Configuring a Redux store is too complicated"
  - "I have to add a lot of packages to get Redux to do anything useful"
  - "Redux requires too much boilerplate code"

"we strongly recommend that you use it".

https://redux.js.org/redux-toolkit/overview

Actions

- To change something in the state, Dispatch an action.
- An action is a plain JS object with a type field.

Reducers

- All state updates logic lives in functions called reducers.
- Smaller functions can be composed into larger functions.
- Reducers should be pure functions, with no side effects.
- Reducers need to update data immutably.

Store

- A Redux store contains the current state value.
- Stores are created using the createStore method, which takes the root reducer function.
- Stores have 3 main methods.
  - dispatch
    - Starts a state update with the provided action object.
  - getState
    - Returns the current stored state value.
  - subscribe
    - Accepts a callback funtion that will be run every time an action is disptched.

Provider

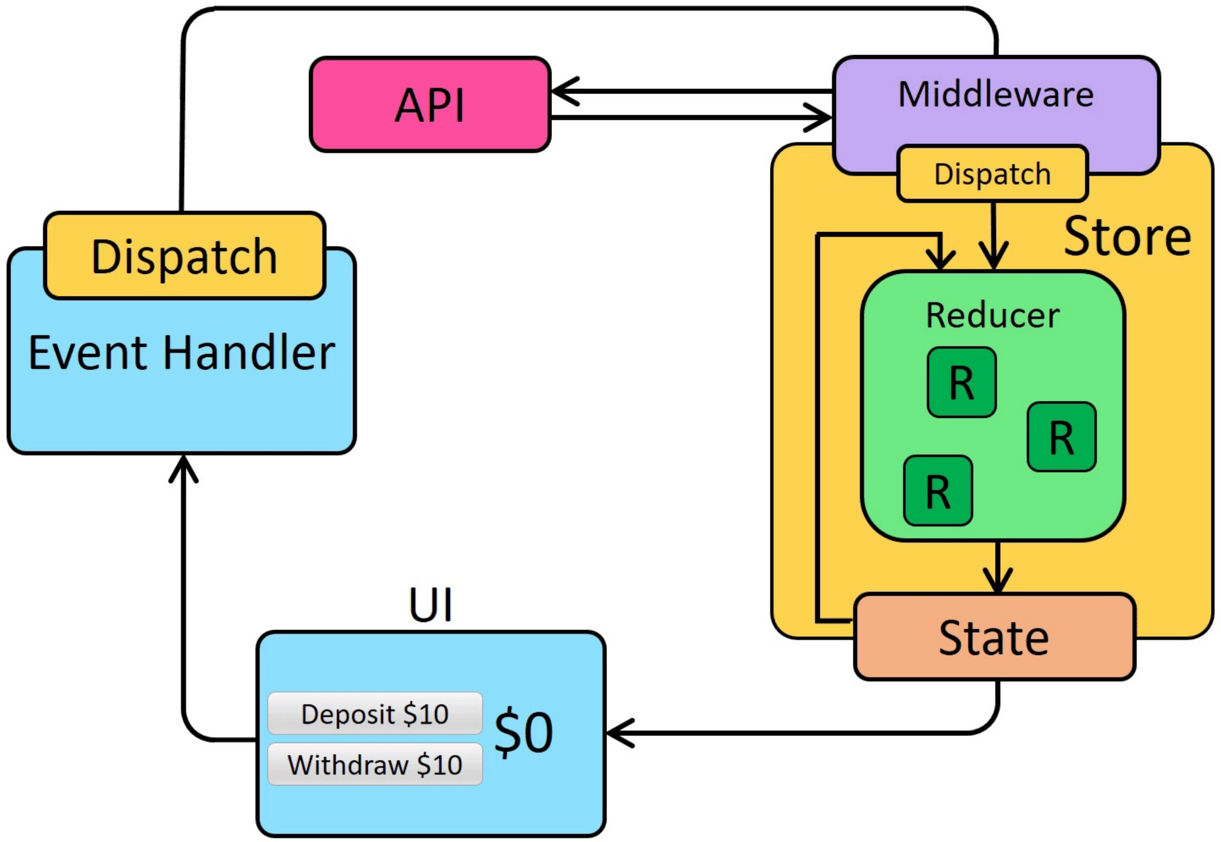
- Makes the Redux store accessible to all components in the app.
- Should be set up in app entry point file and wrap entire app component.
- Set the store property.

Async logic

- Middleware
  - Middleware are store plugins that wrap dispatch.
  - Redux thunk middleware is standard async middleware.
    - Allows passing functions to dispatch instead of actions.
    - Functions receive (dispatch, getState) as arguments.
    - Can do any sync or async logic inside.

- https://redux-toolkit.js.org/api/createAsyncThunk

Async logic


API
Middleware
Dispatch
Store
Reducer
R
R
R
State
Dispatch
Event Handler
UI
Deposit $10
Withdraw $10
$0

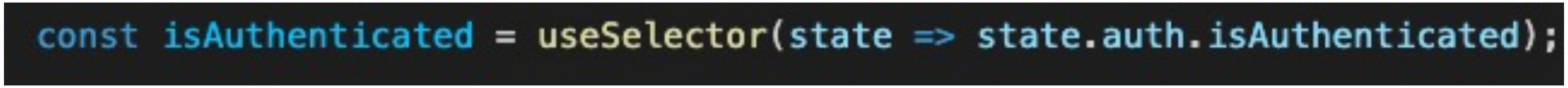
Redux Toolkit includes:

- A configureStore() function with simplified configuration options. It can automatically combine your slice reducers, adds whatever Redux middleware you supply, includes redux-thunk by default, and enables use of the Redux DevTools Extension.
- A createReducer() utility that lets you supply a lookup table of action types to case reducer functions, rather than writing switch statements. In addition, it automatically uses the immer library to let you write simpler immutable updates with normal mutative code, like state.todos[3].completed = true.
- A createAction() utility that returns an action creator function for the given action type string. The function itself has toString() defined, so that it can be used in place of the type constant.
- A createSlice() function that accepts a set of reducer functions, a slice name, and an initial state value, and automatically generates a slice reducer with corresponding action creators and action types.
- The createSelector utility from the Reselect library, re-exported for ease of use.

createSlice()

- A function that accepts an initial state, an object full of reducer functions, and a "slice name", and automatically generates action creators and action types that correspond to the reducers and state.
- This API is the standard approach for writing Redux logic.
- Internally, it uses createAction and createReducer, so you may also use Immer to write "mutating" immutable updates:

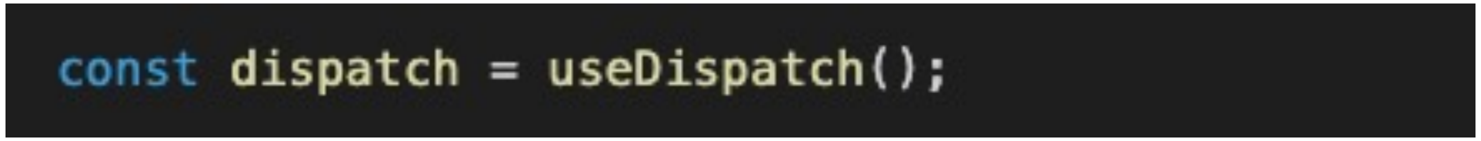
useSelector()



```
const isAuthenticated = useSelector(state => state.auth.isAuthenticated);
```

- **useSelector() will also subscribe to the Redux store, and run your selector whenever an action is dispatched.**
- The selector will be called with the entire Redux store state as its only argument.
- The selector will execute whenever the function component renders (unless its reference hasn't changed since a previous render of the component so that a cached result can be returned by the hook without re-running the selector).

useDispatch



```
const dispatch = useDispatch();
```

- This hook returns a reference to the dispatch function from the Redux store.
- You may use it to dispatch actions as needed.