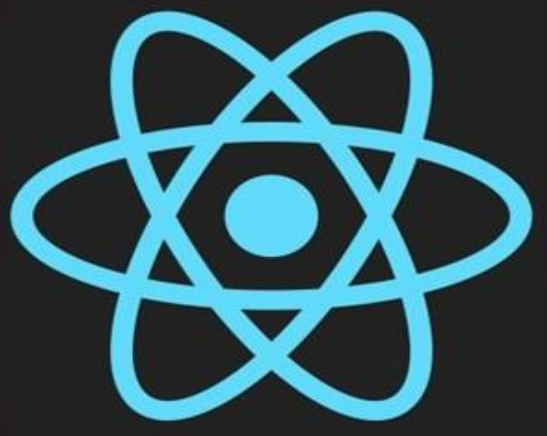
State Management using Redux

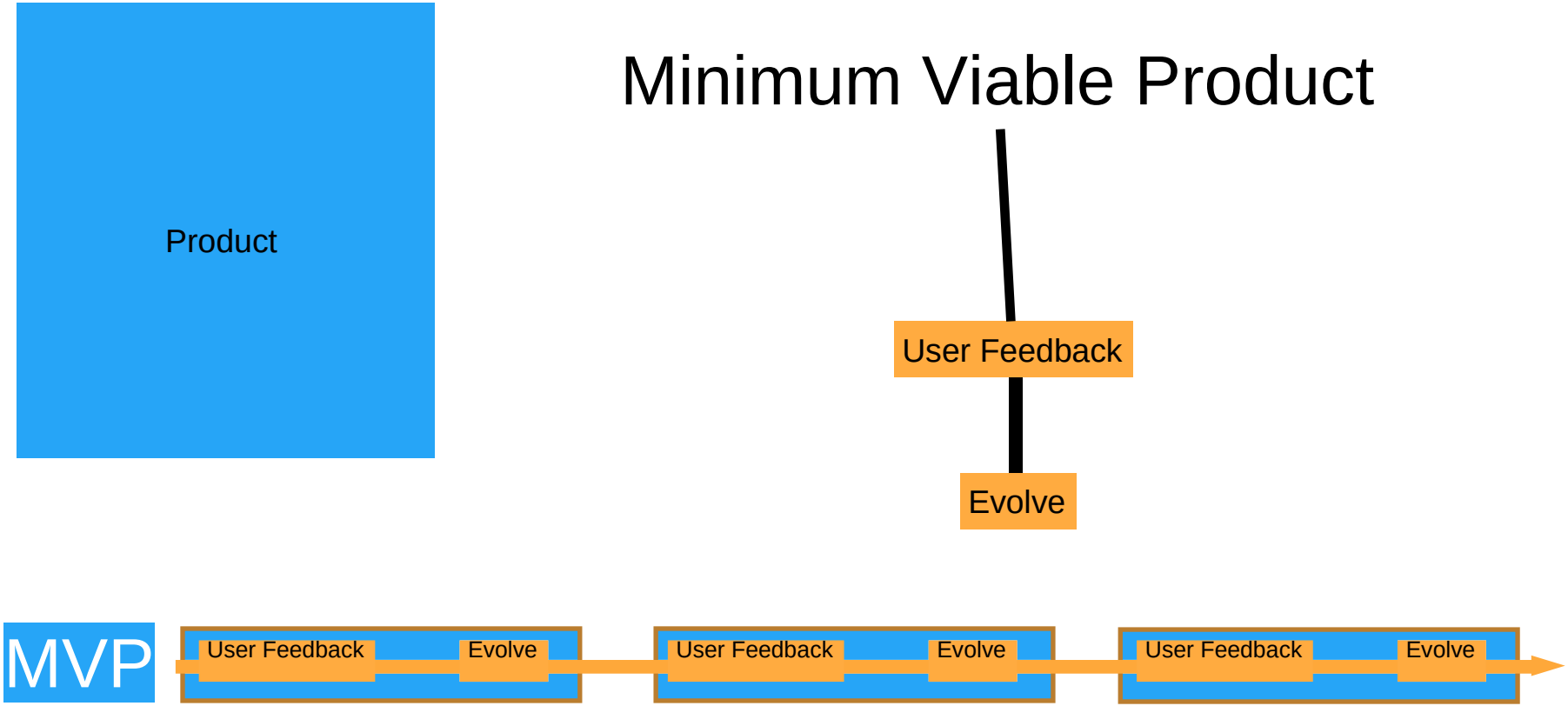


What We Will Learn

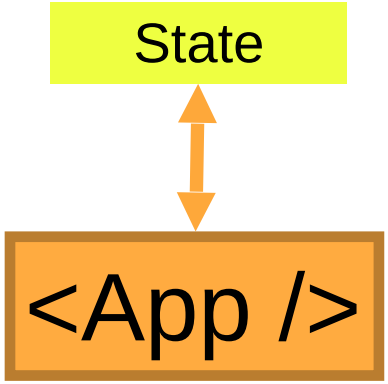
- State Management & Redux
- Setting up Redux
- Actions & Reducer for the Catalog
- Using the connect() higher order function
- Actions & Reducer for the Cart
- Using Redux Hooks

State Management and Redux

State Management using Redux



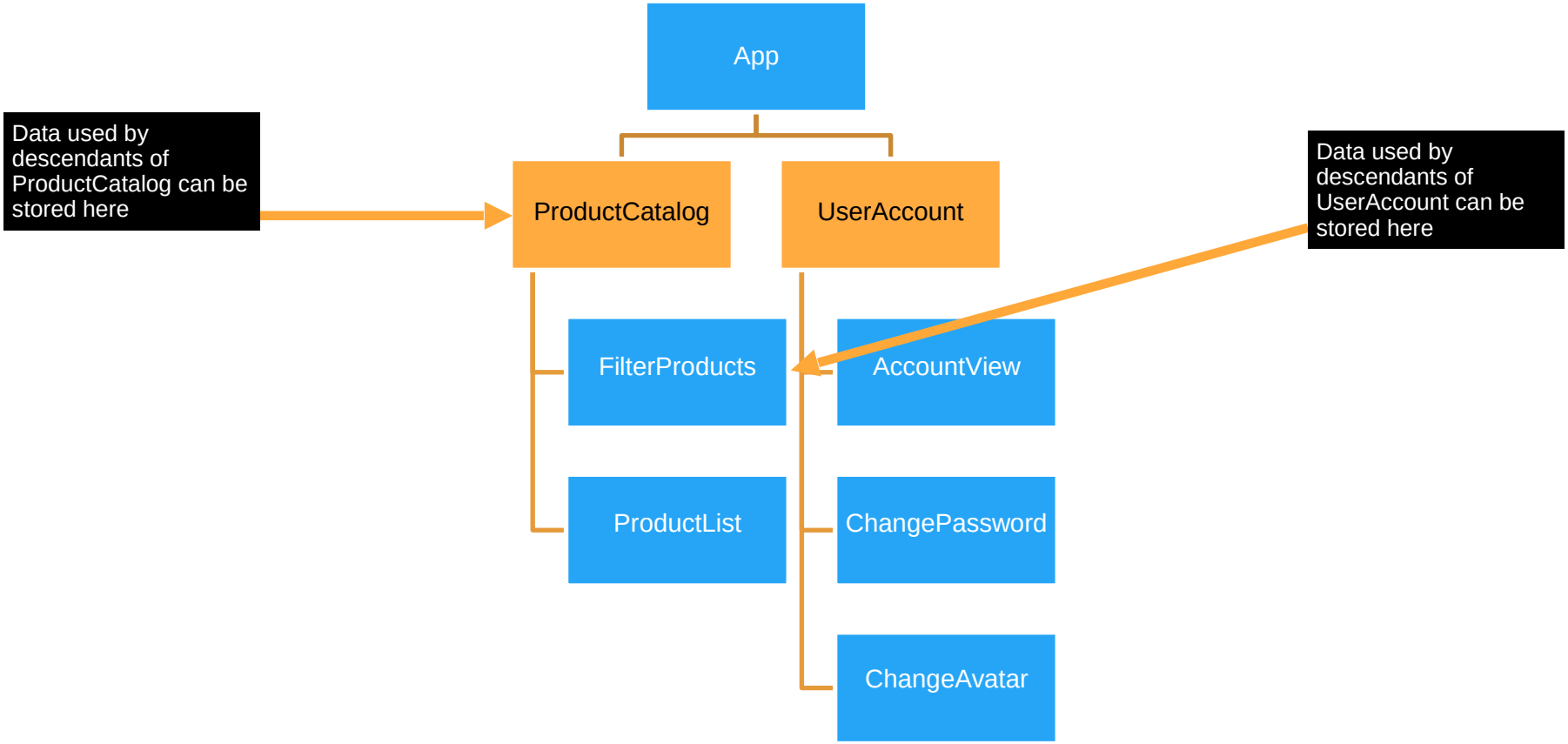
State Management using Redux



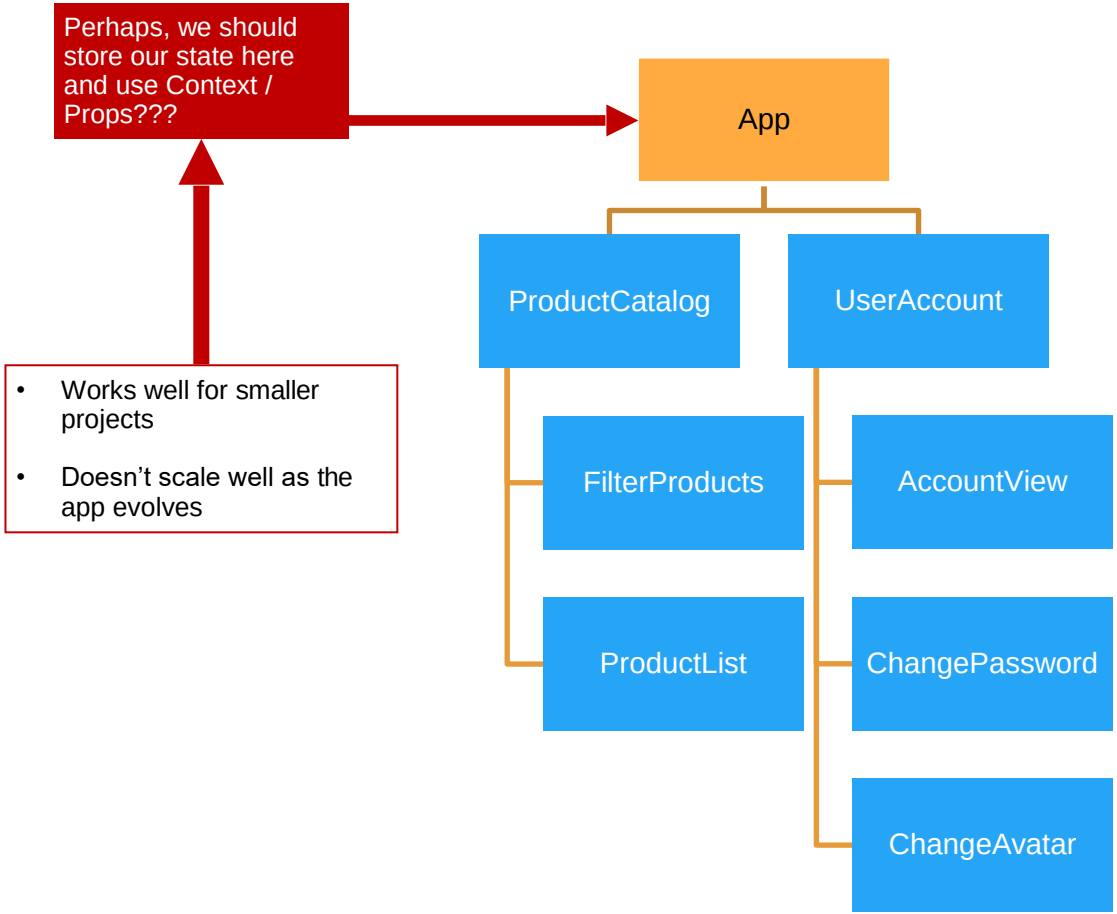
React apps rely on State – the Data layer

State Management using Redux

Storing state at the nearest parent



State Management using Redux

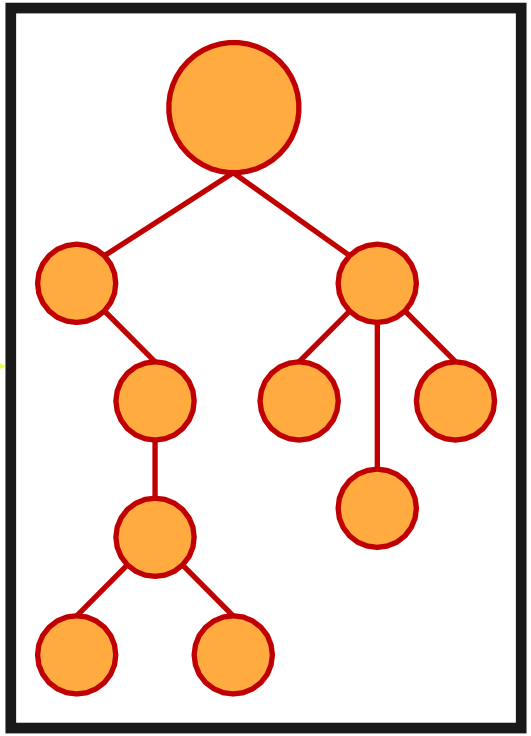


State Management using Redux

The Goal

To have, a dedicated, central state store with deterministic state

Dedicated State Store



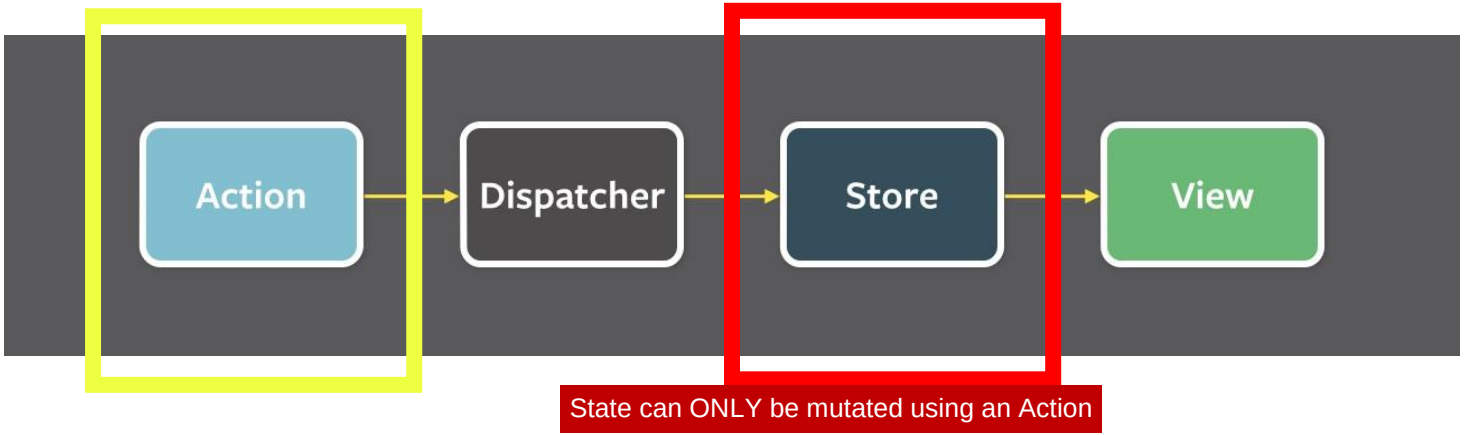
- Predictable State Mutations
- Easy to Identify Bugs

State Management using Redux

Flux Architecture

Flux

https://facebook.github.io/flux/



Source: https://facebook.github.io/flux/docs/in-depth-overview

State Management using Redux

The most popular state management solution for React

Redux

https://redux.js.org/

- Built originally by Dan Abramov in 2015
- Andrew Clark joined later as collaborator
- Redux is a predictable state container
- Inspired by Flux & the Elm language

State Management using Redux

Redux and its architecture

DISPATCH

Add to Cart

View

ACTION

```
{
  type: "ADD_ITEM",
  item: {
    id: "C23",
    product: "MACBOOK Pro 2019",
    cost: 145000,
    QUANTITY: 1
  }
}
```

READ-ONLY

Single source of truth for the entire application

Store

Stores your State as a JavaScript object

```
{
  CATALOG: {
    DEPARTMENTS:
    [], products: []
  },
  CART: {
    items: [{
    id: "C23",
    product: "MACBOOK Pro
2019",
    cost: 145000,
    QUANTITY: 1
  }]
  }
}
```

REDUCER MUTATES THE STORE

REDUCER

```
function CARTREDUCER(STATE, ACTION) {
  if (ACTION.TYPE === "ADD_ITEM") {
    return {
      items: [...STATE, ACTION.ITEM]
    }
  }
}
```

Reducers are Pure Functions

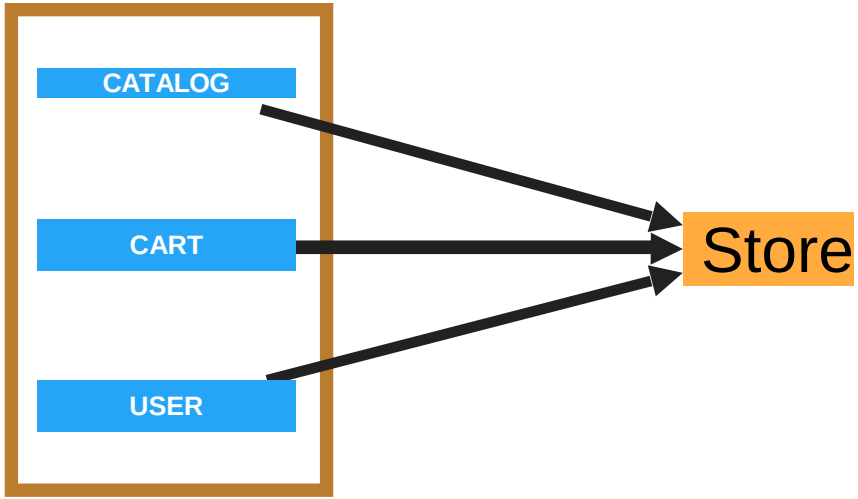
VIEW UPDATES

View (Your React Component) is subscribed to changes to the store

- Unidirectional Data Flow
- Predictable State Mutations

State Management using Redux

One store, Multiple Reducers



Data separation by domain (reducer)

```
{
  CATALOG: {},
  CART: {},
  user: {}
}
```

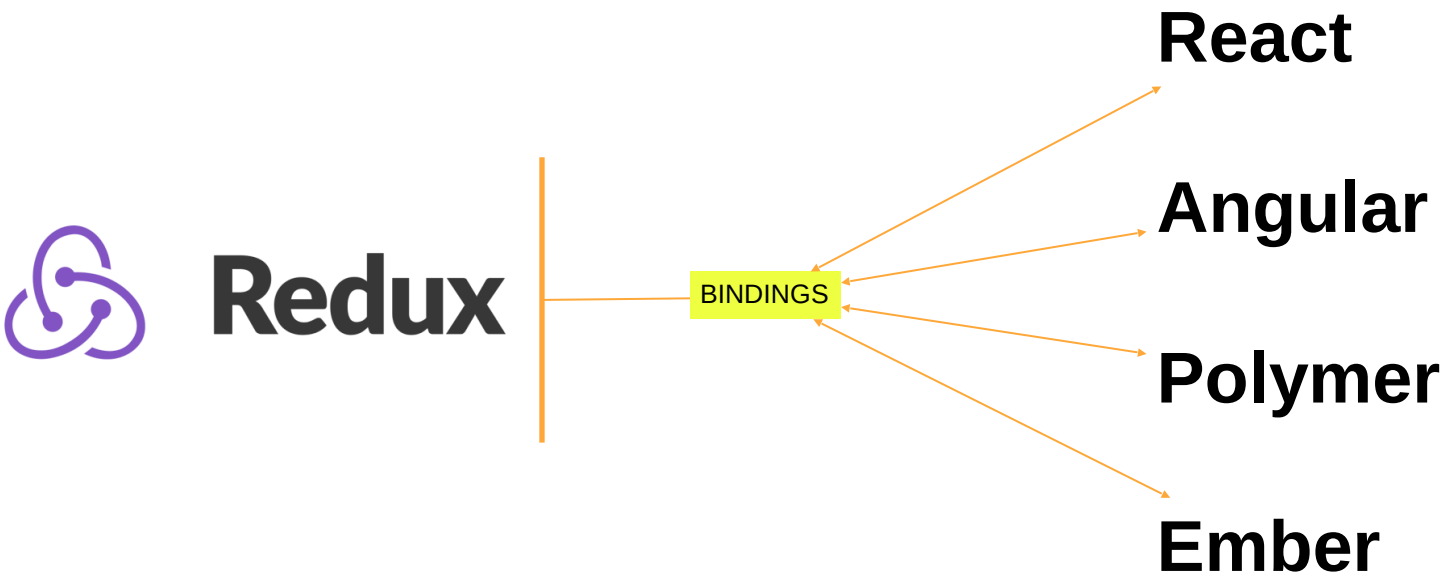
State Management using Redux

Three principles of Redux

1. A Redux store is the single source of truth
2. Redux state is read-only. There are no setters
3. State mutations must only be made by pure functions known as reducers

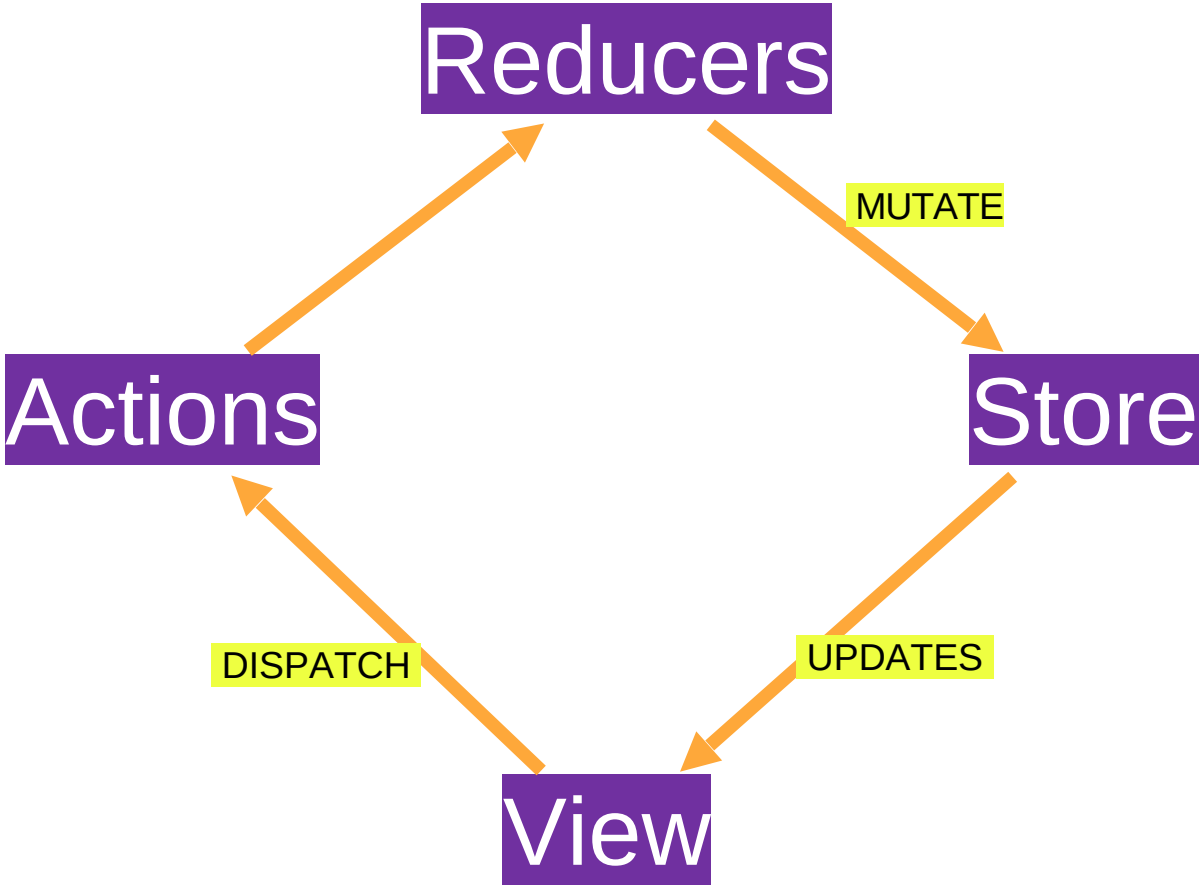
State Management using Redux

Framework agnostic library



Setting up Redux

State Management using Redux



State Management using Redux

Code Demo

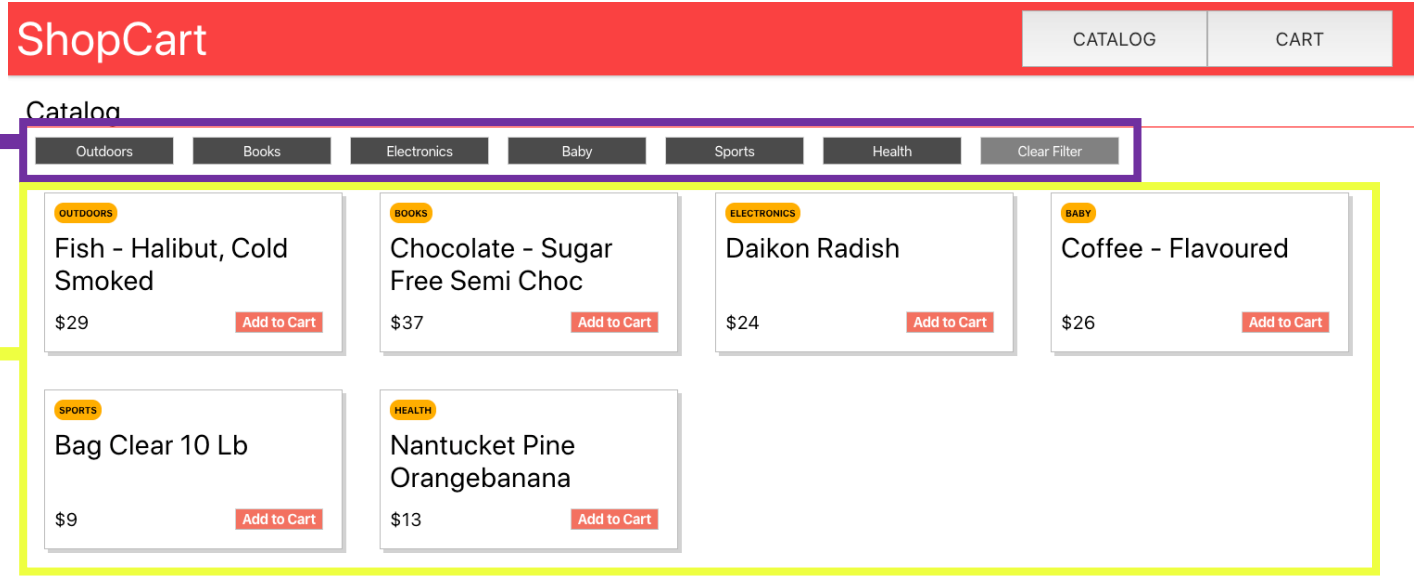
Action & Reducers for the Catalog

State Management using Redux

The Catalog Section

Filter by department buttons

Products



State Management using Redux

Setting the ‘filter’ flag


```
{
  DEPARTMENTS: [],
  products: [],
  filter: “Electronics”
}
```

```
{
  DEPARTMENTS: [],
  products: [],
  filter: null
}
```

State Management using Redux

Actions for the Catalog

```
{
  type: "SET_FILTER",
  filter: <DEPARTMENTNAME>
}
```

```
{
  type: "CLEAR_FILTER"

}
```

catalogReducer

state

State Management using Redux

Code Demo

The connect higher order function

State Management using Redux

Interfacing React Components with Redux

connect()

<Catalog />

Redux Hooks

- useSelector()
- useDispatch()

<Cart />

State Management using Redux

The connect() higher order function

Redux

connect(mapState, mapDispatch)(Component)

<Component ***props*** />

Optimizations | Caching

State Management using Redux

Code Demo

Actions and Reducers for the Cart

State Management using Redux

The Cart

ShopCart

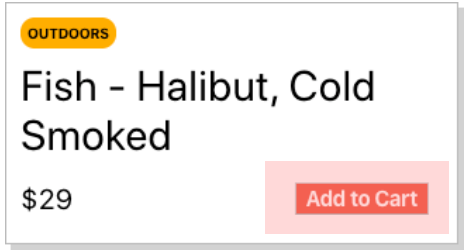
CATALOG | CART 10

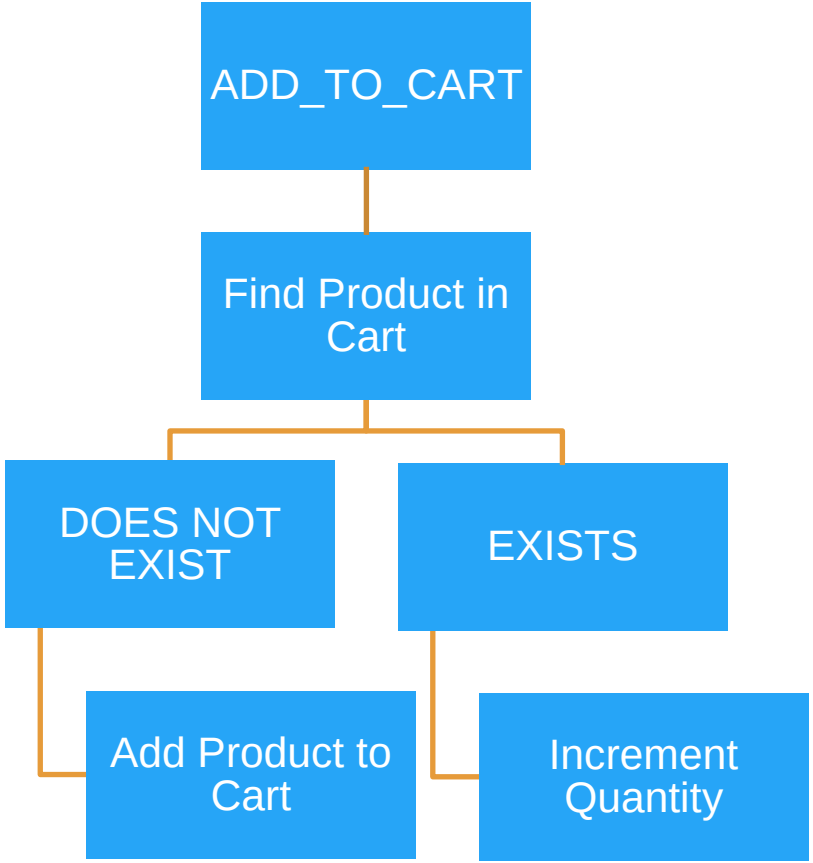
Cart

Coffee - Flavoured	5	Remove	$26	$130
Bag Clear 10 Lb	2	Remove	$9	$18
Chocolate - Sugar Free Semi Choc	3	Remove	$37	$111
	10			**$259**

State Management using Redux

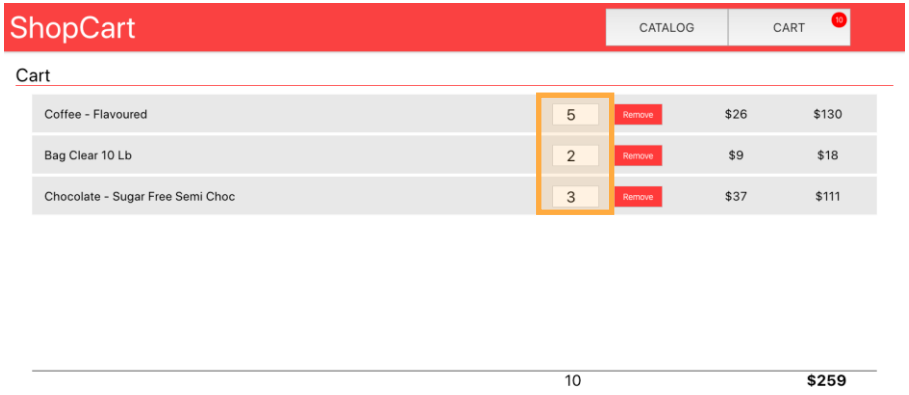
Adding a Product to the Cart

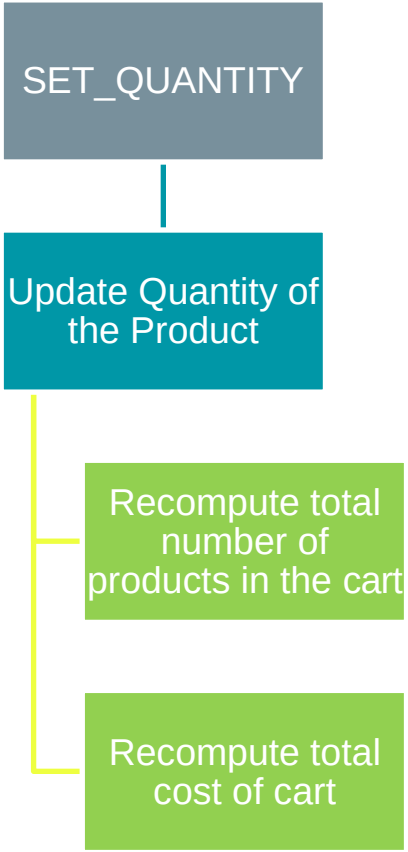




State Management using Redux

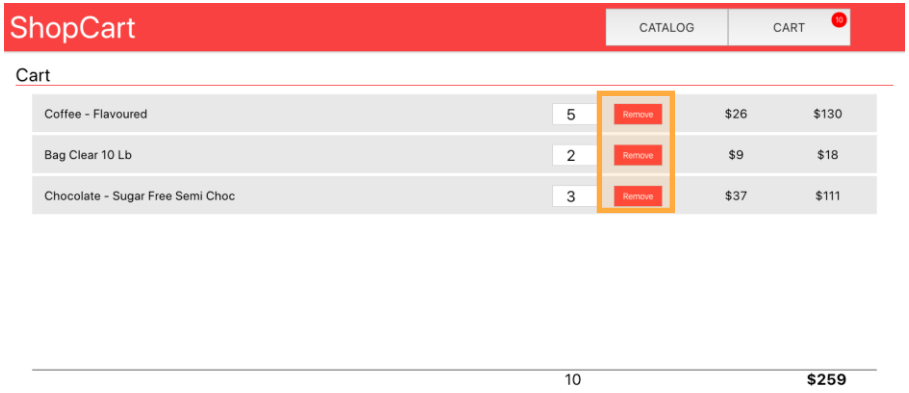
Updating Quantity of a Product

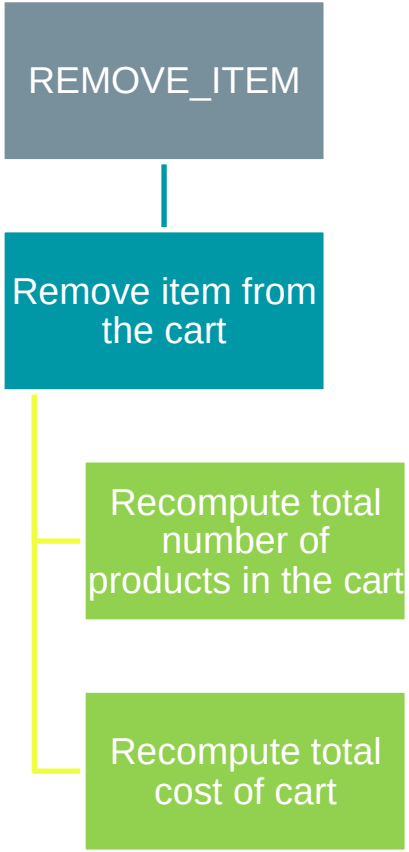




State Management using Redux

Removing a product from the cart





State Management using Redux

Actions for the Cart

```
{
    type: "ADD_TO_CART",
    product
}
```

```
{
    type: "SET_QUANTITY",
    code,
    QUANTITY
  }
```

```
{
    type: "REMOVE_ITEM",
    code
  }
```

State Management using Redux

Code Demo

Using Redux Hooks

State Management using Redux

Using the connect() higher order component

```
connect(mapStateToProps, mapDispatchToProps)(Catalog)
```

```
const mapStateToProps = state => {
  return {
    catalog: filterProducts(state.catalog.products, state.catalog.filter),
    departments: state.catalog.departments
  };
};
```

```
const mapDispatchToProps = dispatch => {
  return {
    onSetFilter: department => dispatch(setFilter(department)),
    onClearFilter: () => dispatch(clearFilter())

  };
};
```

```
const itemCount = useSelector(state =>
        state.cart.itemCount);
```

- `const dispatch = useDispatch();`

State Management using Redux

Code Demo

State Management using Redux

Component Rendering

Function Component Renders → useSelector() is called

- Initial render
- Change in props
- Parent component re-renders

- If the value is same, returns cached
- Uses strict equality checks (===)

State Management using Redux

The problem with objects & ===

An Object

```
const { totalProducts, totalCost, items } = useSelector(state => state.cart);
```

Strict equality checks are unable to compare shallow objects & force a re-render even if values are the same

State Management using Redux

Objects aren't a problem when using connect()

```
connect(mapStateToProps, mapDispatchToProps)(Catalog)
```

```
const mapStateToProps = state => {
  return {
    catalog: filterProducts(state.catalog.products, state.catalog.filter),
    departments: state.catalog.departments
  };
};
```

Shallow Equality Check ensures that the Component re-renders ONLY if one or more properties change

State Management using Redux

Solution : Multiple instances of useSelector()

An Object

```
const { totalProducts, totalCost, items } = useSelector(state => state.cart);
```

```
const totalProducts = useSelector(state => state.cart.totalProducts);
const totalCost = useSelector(state => state.cart.totalCost);
const items = useSelector(state => state.cart.items);
```

State Management using Redux

Solution : Using shallowEqual for Shallow Equality Checks

An Object

```
const { totalProducts, totalCost, items } = useSelector(state => state.cart);
```

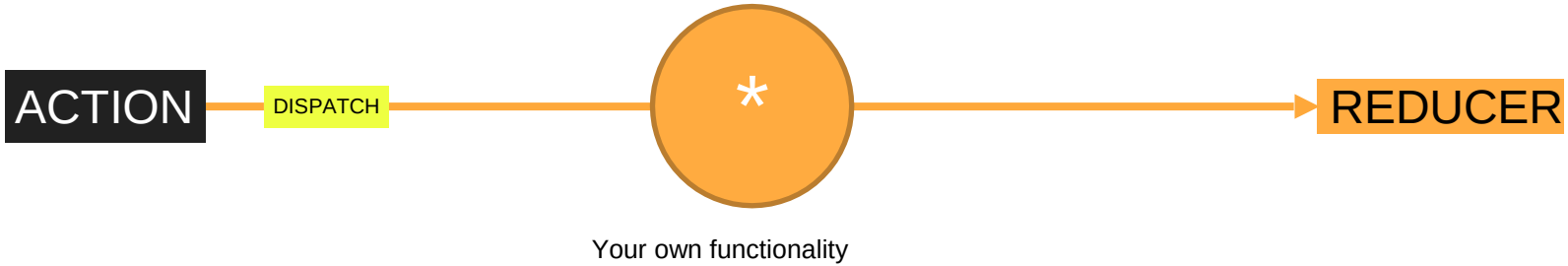
```
import { shallowEqual } from "react-redux";

const { totalProducts, totalCost, items } = useSelector(
  state => state.cart,
  shallowEqual
);
```

Middleware & Persistence

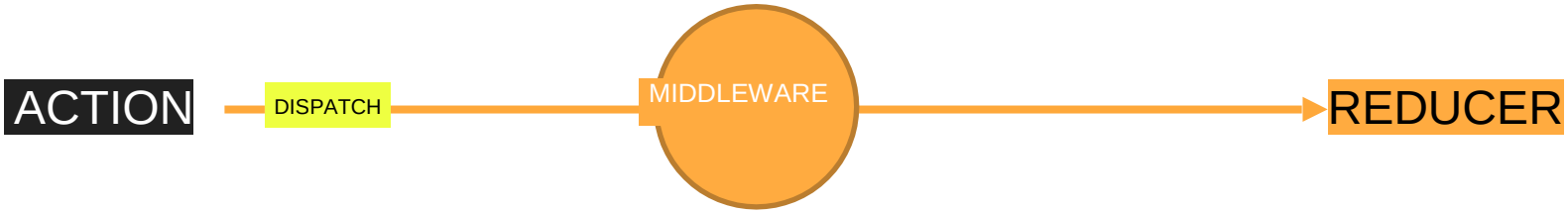
State Management using Redux

Plug your own feature between an Action & the Reducer



State Management using Redux

Redux Middleware



State Management using Redux

Populating the State for the Catalog

HARD-CODED DATA

```
const INITSTATE = {
  DEPARTMENTS: […],
  products: […],
  filter: null
};
```

REDUCER

State Management using Redux

API

Fetching and Populating Data from an API

```
const INITSTATE = {
  DEPARTMENTS: […],
  products: […],
  filter: null
};
```

REDUCER

State Management using Redux

Fetching and Populating Data from an API

```
const MyComponent = () => {
  const DISPATCH = USEDISPATCH();
  useEffect(() => {
    DATASERVICE().THEN(DATA => DISPATCH({ type: "INIT_DATA", DATA }));
  }, []);
  return <div>Hey there!</div>;
};
```

- Decouples and scatters functionality across your components
- Not a reusable workflow if other parts of the app need to invoke the same action

State Management using Redux

Action Creators

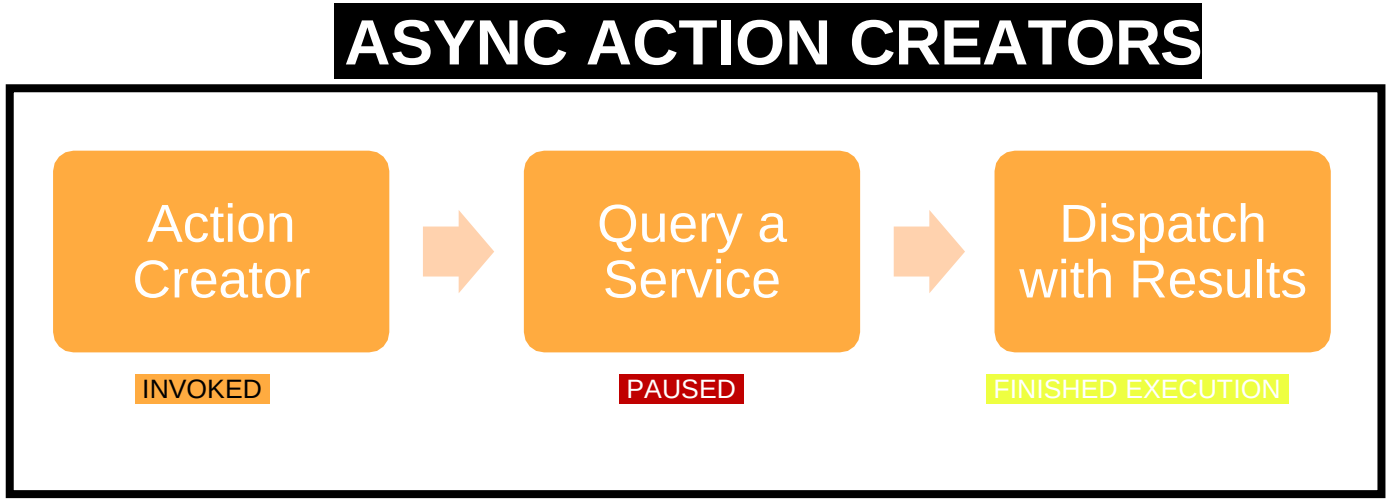
```
export const SETQUANTITY = (code, QUANTITY) => {
  return {
    type: "SET_QUANTITY",
    code,
    QUANTITY
  };
};
```

An action creator is responsible for composing actions and keeping intricacies away from Components

They're **synchronous** by default!

State Management using Redux

Action creators which can be paused



Middleware to build Async Creators

Async Creators using Middleware

redux-thunk

https://github.com/reduxjs/redux-thunk

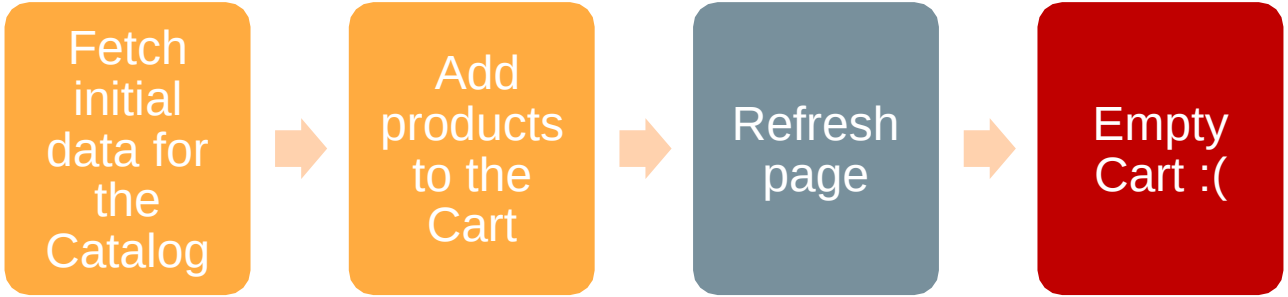
Allows you create async action creators that can perform a side-effect before dispatching an action directly

State Management using Redux

Code Demo

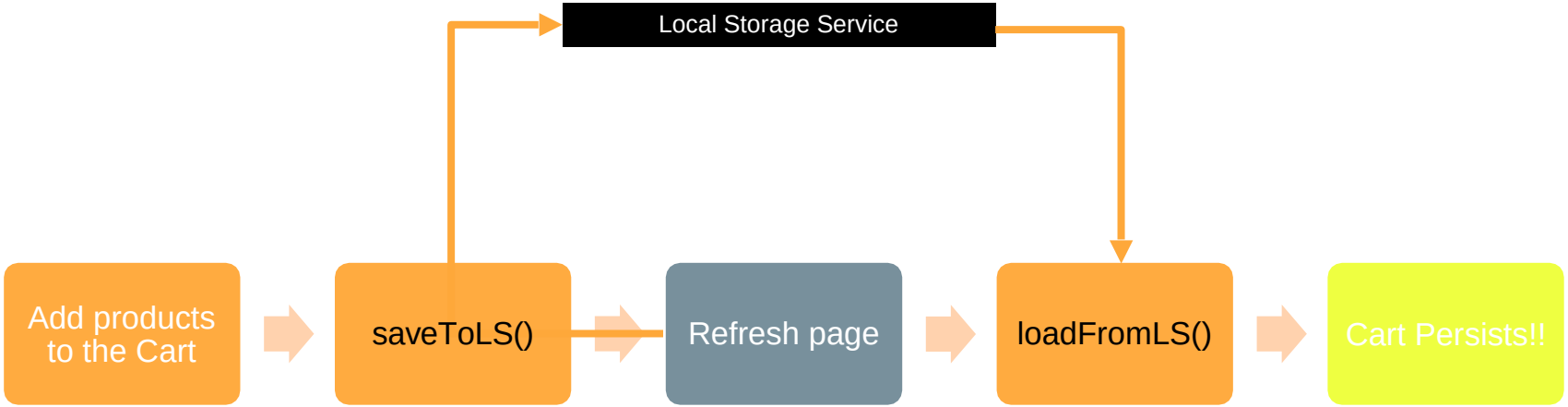
State Management using Redux

Data doesn’t persist



State Management using Redux

Data Persistence using localStorage



State Management using Redux

Hydrating the Redux Store

```
export const initApp = () => DISPATCH => {
  PROMISE.ALL([GETDEPARTMENTS(), getProducts(), LOADFROMLS()]).THEN(DATA => {
    DISPATCH({
      type: "INITIALIZE_CATALOG",
      DEPARTMENTS: DATA[0],
      products: DATA[1]
    });

    DISPATCH({
      type: "REHYDRATE_CART",
      items: DATA[2]
    })
  });
};
```

Hydrating the Cart from localStorage

State Management using Redux

Redux

Getting Started API FAQ GitHub Need help? Search

- Introduction
  - Getting Started with Redux
  - Installation
  - Motivation
  - Core Concepts
  - Three Principles
  - Prior Art
  - Learning Resources
  - Ecosystem
  - Examples
- Basic Tutorial
- Advanced Tutorial
- Recipes
- FAQ
- Style Guide
- Other
- API Reference
- Redux Toolkit

Picture

Ecosystem

Redux is a tiny library, but its contracts and APIs are carefully chosen to spawn an ecosystem of tools and extensions, and the community has created a wide variety of helpful addons, libraries, and tools. You don't need to use any of these addons to use Redux, but they can help make it easier to implement features and solve problems in your application.

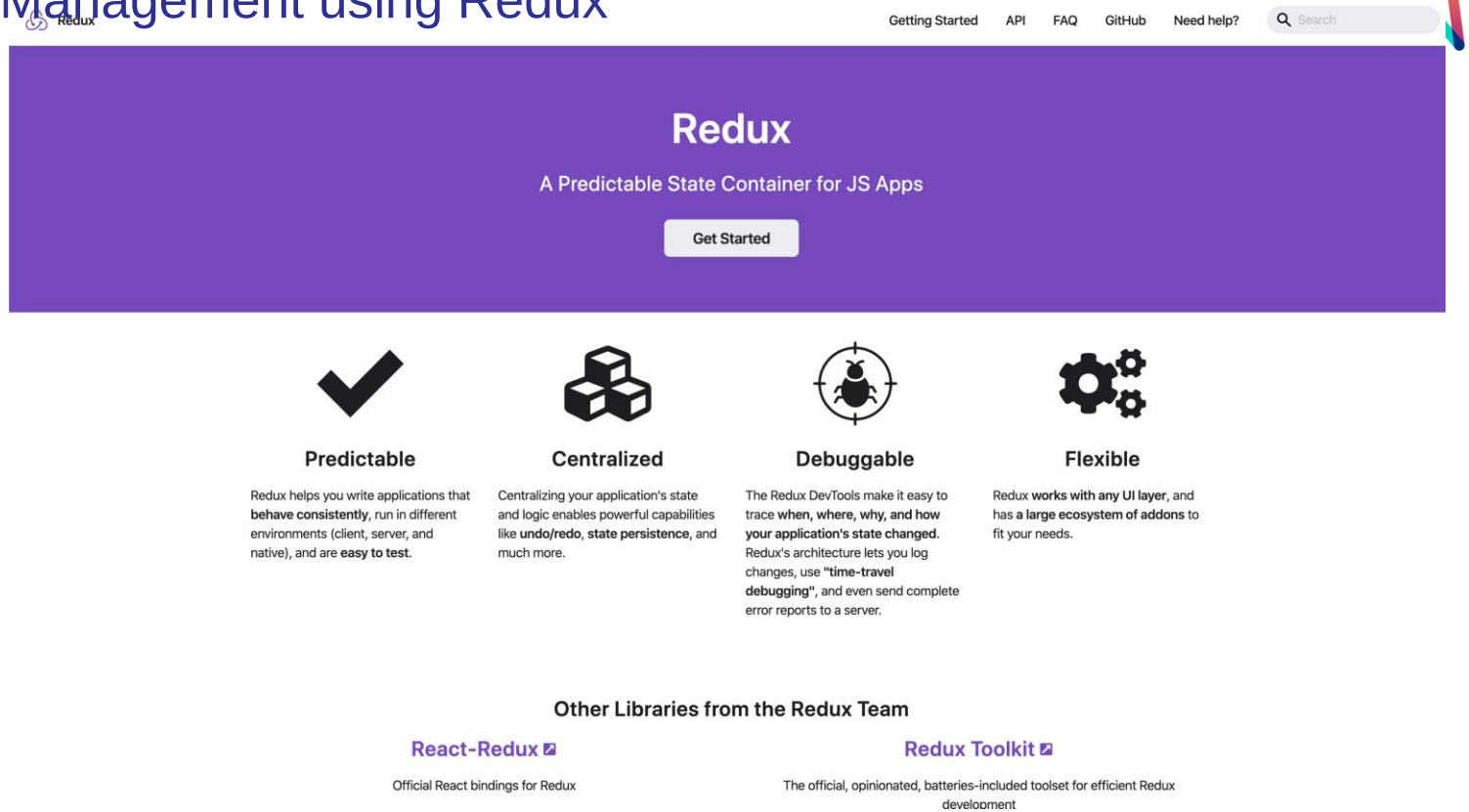
For an extensive catalog of libraries, addons, and tools related to Redux, check out the Redux Ecosystem Links list. Also, the React/Redux Links list contains tutorials and other useful resources for anyone learning React or Redux.

This page lists some of the Redux-related addons that the Redux maintainers have vetted personally, or that have shown widespread adoption in the community. Don't let this discourage you from trying the rest of them! The ecosystem is growing too fast, and we have a limited time to look at everything. Consider these the "staff picks", and don't hesitate to submit a PR if you've built something wonderful with Redux.

Table of Contents

https://redux.js.org/introduction/ecosystem/#persistence

State Management using Redux



https://redux.js.org/

thank you!