Welcome to

React/Redux Fundamentals

DevelopIntelligence
A PLURALSIGHT COMPANY

Hello

About me...

- Training for the last 5 years
- Also train on Docker/K8s, Java

We teach over 400 technology topics

You experience our impact on a daily basis!

My pledge to you

I will...

- Make this interactive
- Ask you questions
- Ensure everyone can speak
- Use an on-screen timer

Objectives

At the end of this course you will be able to:

- Work with React + Redux on a daily basis
  - We will give you real-world examples and exercises
  - Heavily exercise-oriented

Prerequisites

This course assumes you:

- Have some Javascript knowledge (we will cover ES6 to start)
- Have some background in software development

Agenda for next 3 days

- Day 1:
  - ES6 Overview/Exercises
  - React Overview + Lab 1 - Todo List
- Day 2:
  - React Hooks for state
  - Redux Overview + Lab 2
- Day 3:
  - Asynchronous programming in React, useEffect + Lab 3
  - Redux Toolkit + Lab 4

Slide Deck

- The slides will be shared with you **after** the class
- This is because I often make updates during the class :)
- I have your email addresses since they are in the meeting invite, and I will simply share it with you

Breaks

- We will take breaks every hour for about 7-10 minutes
- We will also take a longer break for lunch around 12:00 PST time
    - about 1-1.5 hours

Introductions

Student Introductions

Let’s get a sense for class knowledge and experience, I will go through the participants list

- Any experience with Javascript and React?
- Any experience with other Javascript frameworks?
- Fun fact about yourself (ie hobbies, something interesting..)

Agenda

Agenda for next 3 days

- Day 1:
  - ES6 Overview/Exercises
  - React Overview + Lab 1 - Todo List
- Day 2:
  - React Hooks for state
  - Redux Overview + Lab 2
- Day 3:
  - Asynchronous programming in React, useEffect + Lab 3
  - Redux Toolkit + Lab 4

Before React - ES6 Javascript

Ecmascript (ES) - what is it?

- ECMAScript is a standard for a scripting language
- Javascript language is ***based*** on the ECMAScript standard
- Other languages that implement the ES standard like JScript (Microsoft), SpiderMonkey also exist
    - Like dialects that share the same structure
    - But of course Javascript most popular implementation

ES6 - why is it important?

- Major revision of ECMAScript standard
- Released in 2015
- Introduces a lot of “new” syntax that you should be familiar with

ES6 Review and Practice

- Review/Study Guide here: https://github.com/varoonsahgal/react-fundamentals/blob/main/es6studyguide.md

- Practice here: https://github.com/varoonsahgal/react-fundamentals/blob/main/es6exercises.md

JSFiddle for ES6 exercises

- We will use **JSFiddle** for the exercises:
  - You can save your work without creating an account
  - Also use the console to output information
- https://jsfiddle.net/

Installations

Installations (also in the readme for lab 1)

To work with React, we will start with create-react-app, which requires Node.js

To work with Node.js, you can install Node Version Manager (nvm) following instructions here:

https://github.com/nvm-sh/nvm#installing-and-updating

Installations - code editor

Please use VS Code for the React portion of the class - it will make life a lot easier!

You do not want to use JSFiddle for the React labs - that's strictly for the ES6 exercises…

Why ReactJS?

React JS - history

- Released by Facebook in 2013
- React Native (mobile) released in 2015
- Angular released in 2016

React: Overview

- Not a framework, but a ***library***
- “V” in MVC (Model View Controller)
-

Why even use a library?

- Libraries (like React.js) are meant to save us time
- Libraries give us a foundation for starting a new project
- And they give us functions/methods that we can then invoke…
- Instead of writing maybe 100 lines of code, it could be far less - like 20

React: Virtual DOM

- Virtual DOM - updating dom directly is “expensive”, “diff”
- Other frameworks/libraries are now using a virtual DOM as well…

React vs other libraries/frameworks

- React vs JQuery
  - Jquery updates DOM directly, React doesn't
- React (FB) vs Angular (Google)
  - Angular is a MVC framework, React is a library (V in MVC)
  - React == smaller bundle size, so it's faster
  -

JSX

What is JSX?

- Write HTML with JavaScript (JSX)
- JSX == Javascript XML
- JSX is “syntactic sugar”
- Makes our React code easier to read/digest

React: Reusable Components

- Props to pass data
- Increase code reuse
  - Component explorer
- Components can have state

2.

How Components Work

Anatomy of a Component

```
import React from 'react'

const Message = (props) => {
  console.log('I am rendering with props', props)
  return (
    <p>{props.message}</p>
  )
}
```

Anatomy of a Component

View as a State Machine

```
import React from 'react'

const UserProfileCard = ({ avatar, name }) => (
  <div>
    <img src={avatar} />
    <p className="UserProfileCard__name">{name}</p>
  </div>
)
```

Given a set of props, you should know what the view will look like.

Composing Components

```
import React from 'react'

const Circle = ({ color }) => (
  <div
    style={{
      backgroundColor: color,
      width: 50,
      height: 50,
      borderRadius: 50,
    }}
  />
)
```

Can reuse “Circle” multiple times from within another component - like a “Stoplight” parent component

Composing Components

```
import React from 'react'

const Circle = ({ color }) => (
  <div
    style={{
      backgroundColor: color,
      width: 50,
      height: 50,
      borderRadius: 50,
    }}
  />
)
```

```
const Stoplight = () => (
  <div>
    <Circle color="red" />
    <Circle color="yellow" />
    <Circle color="green" />
  </div>
)
```

Function Component

```
import React from 'react'

const AlertButton = ({ message }) => {
  const alert = () => { alert(message) }

  return (
    <button onClick={alert}>Trigger Alert</button>
  )
}

export default AlertButton
```

Function Component

```
import React from 'react'

const AlertButton = ({ message }) => {
  const alert = () => { alert(message) }

  return (
    <button onClick={alert}>Trigger Alert</button>
  )
}

export default AlertButton
```

anything declared within scope belongs only to that instance of a component

declared functions have access to component props

functions often passed as props to other components to handle user events

3.

Dynamic Components with State

State: Creating Dynamic Components

- State changes over time
- Do not mutate state values (e.g. `push` to array)
- Represents user interactions over time

Stateful Functional Component

```
import React, { useState } from 'react'

const Counter = () => {
  const [count, setCount] = useState(0)
  const increment = () => { setCount(count + 1) }
  return (
    <button onClick={increment}>
      Clicked {count} times
    </button>
  )
}
```

Stateful Functional Component

```
import React, { useState } from 'react'
import TitleEditor from './title-editor'

const StatefulFunctionComponent = () => {
  const [title, setTitle] = useState('Electrode Rocks!')

  const updateTitle = (title) => { setTitle(title) }

  return (
    <main>
      <h1>{this.state.title}</h1>
      <TitleEditor handleUpdate={updateTitle} />
    </main>
  )
}

export default StatefulFunctionComponent
```

Stateful Functional Component

```
import React, { useState } from 'react'
import TitleEditor from './title-editor'

const StatefulFunctionComponent = () => {
  const [title, setTitle] = useState('Electrode Rocks!')

  const updateTitle = (title) => { setTitle(title) }

  return (
    <main>
      <h1>{this.state.title}</h1>
      <TitleEditor handleUpdate={updateTitle} />
    </main>
  )
}

export default StatefulFunctionComponent
```

initial state given to "title"

function to update state

When function is called, component re-renders with new state for "title"

Class-y Components

```
import React, { Component } from 'react'
import TitleEditor from './title-editor'

class StatefulComponent extends Component {
  state = {
    title: 'Electrode Rocks!',
  }

  updateTitle = (newTitle) => {
    this.setState({ title: newTitle })
  }

  render() {
    return (
      <main>
        <h1>{this.state.title}</h1>
        <TitleEditor handleUpdate={this.updateTitle} />
      </main>
    )
  }
}

export default StatefulComponent
```

Class-y Components

```
import React, { Component } from 'react'
import TitleEditor from './title-editor'

class StatefulComponent extends Component {
  state = {
    title: 'Electrode Rocks!',
  }

  updateTitle = (newTitle) => {
    this.setState({ title: newTitle })
  }

  render() {
    return (
      <main>
        <h1>{this.state.title}</h1>
        <TitleEditor handleUpdate={this.updateTitle} />
      </main>
    )
  }
}

export default StatefulComponent
```

initial state

class public instance field

return what gets rendered from "render" method

Function vs. Class-based

- React Hooks allow function components to have parity with class components
- React will continue to push toward function components
- Function components are easier, more performant (eventually), and better for advanced React patterns
- Classes have a lot of unnecessary complexity - constructors, “this” bindings, etc.
- Functional components + hooks are also more succinct than equivalent class based components w/ state

React and Redux Fundamentals

1.

Day 1 Recap

What did we learn in Day 1?

- React components
  - Passing props
    - Functions as props
    - Other data as props
  - Function vs. Class components

What did we learn in Day 1? (Part 2)

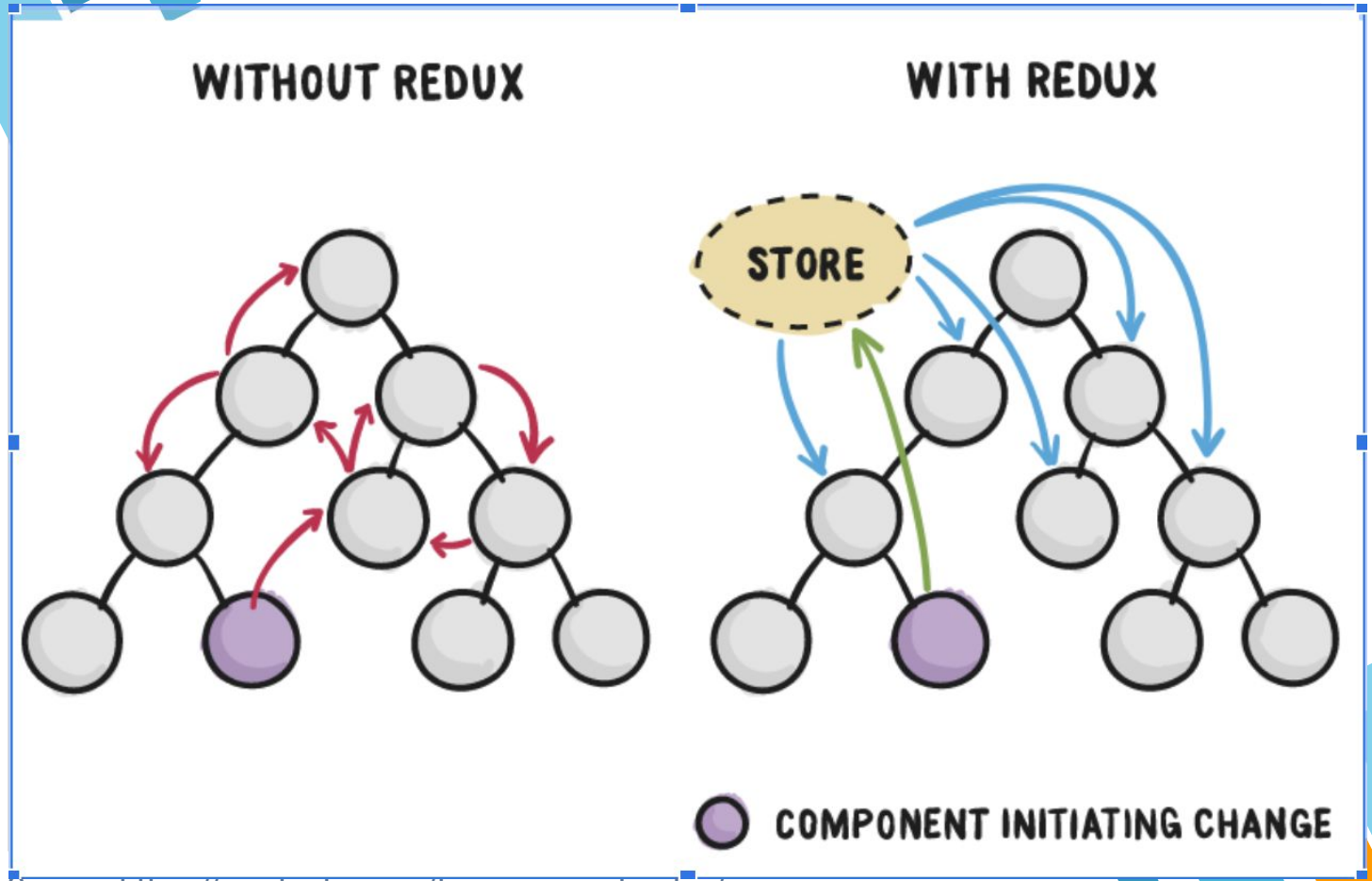
- React components
  - State in function components
    - Hooks - specifically, useState()
  - State in class based components
    - Use this.setState();
      - triggers a re-render of view
      - Dont use this.state =
    - Do not mutate state

2.

Redux Fundamentals

Redux: State Management

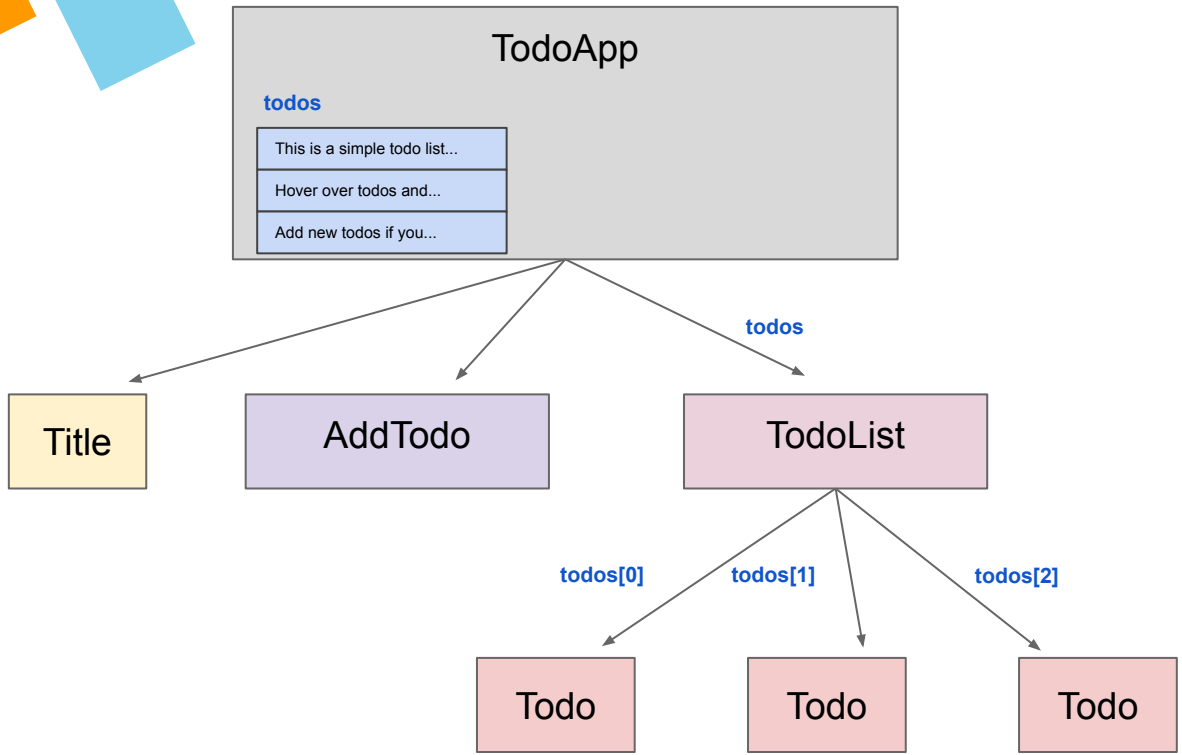
- Predictable state container
  - Single source of truth
  - Functional Pattern - pairs with React

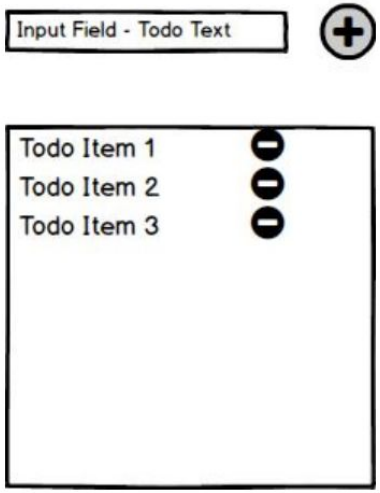


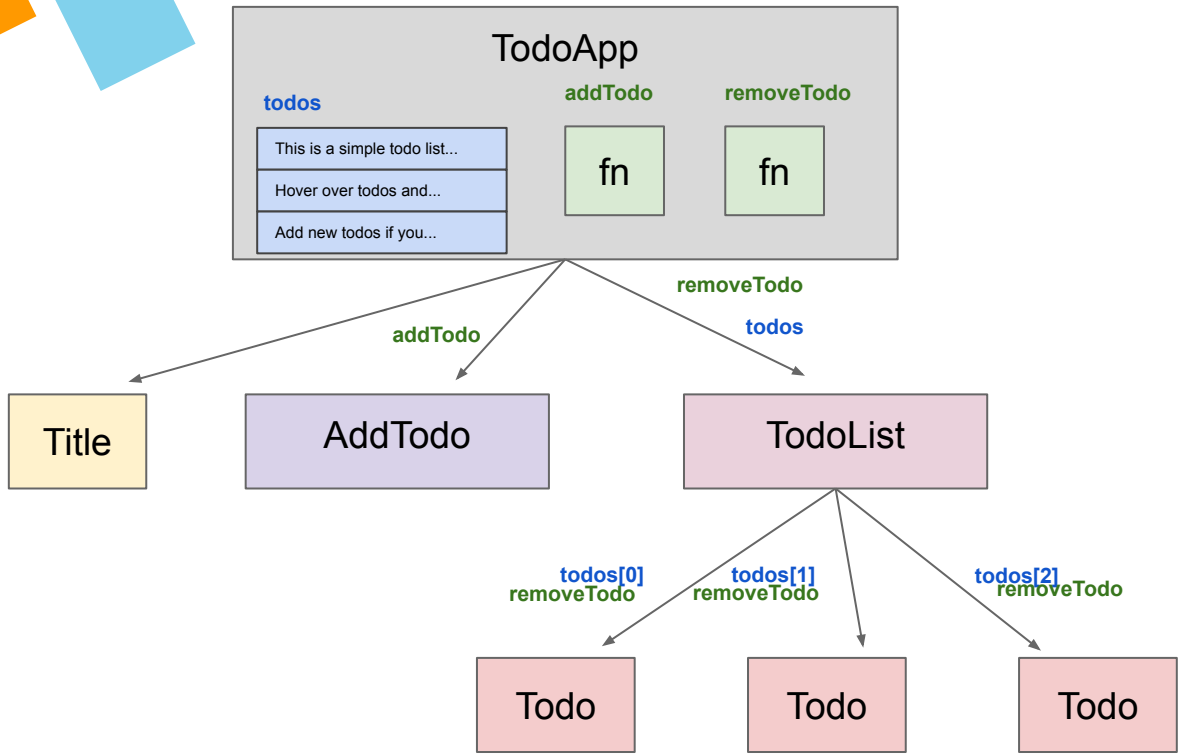
Source: https://css-tricks.com/learning-react-redux/

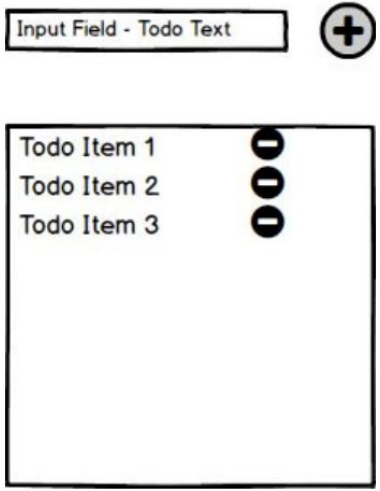
Redux: Predictable State Container

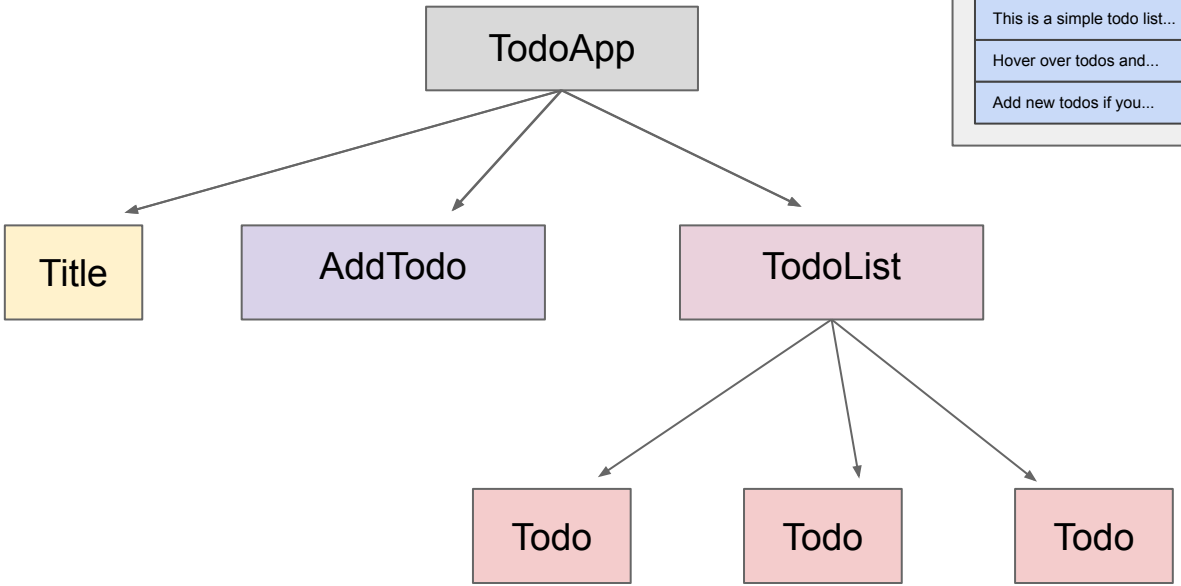
- Deterministic updates - all state transitions are defined
- Centralized state
- State logic decoupled from components
- Easy to initialize, save and debug


TodoApp
todos
This is a simple todo list...
Hover over todos and...
Add new todos if you...
todos
Title
AddTodo
TodoList
todos[0]
todos[1]
todos[2]
Todo
Todo
Todo

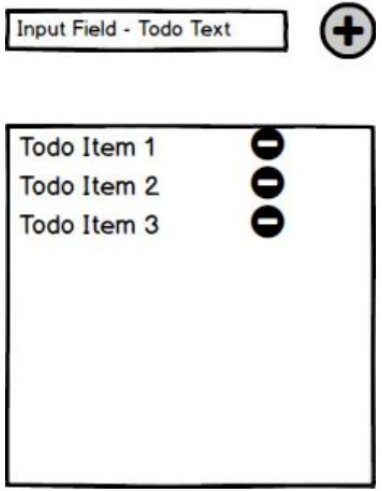

Input Field - Todo Text
Todo Item 1
Todo Item 2
Todo Item 3

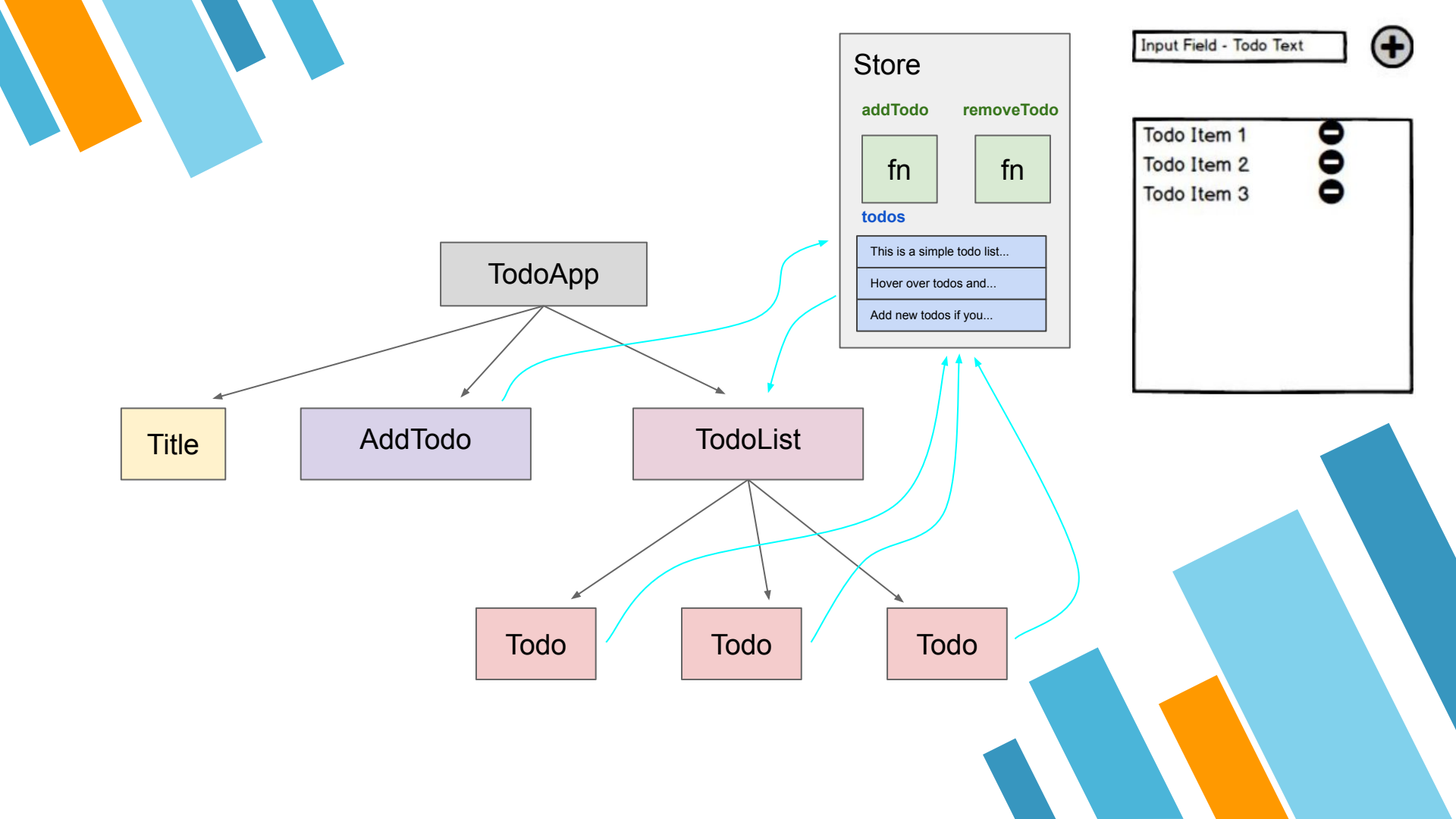

TodoApp
todos
addTodo
removeTodo
This is a simple todo list...
Hover over todos and...
Add new todos if you...
fn
fn
removeTodo
addTodo
todos
Title
AddTodo
TodoList
todos[0]
removeTodo
todos[1]
removeTodo
todos[2]
removeTodo
Todo
Todo
Todo


Input Field - Todo Text
Todo Item 1
Todo Item 2
Todo Item 3


TodoApp
Title
AddTodo
TodoList
Todo
Todo
Todo


Store
addTodo
removeTodo
fn
fn
todos
This is a simple todo list...
Hover over todos and...
Add new todos if you...


Input Field - Todo Text
Todo Item 1
Todo Item 2
Todo Item 3


Store
addTodo
removeTodo
fn
fn
todos
This is a simple todo list...
Hover over todos and...
Add new todos if you...
Input Field - Todo Text
Todo Item 1
Todo Item 2
Todo Item 3
TodoApp
Title
AddTodo
TodoList
Todo
Todo
Todo

Redux State !== React State

- Very common to use both Redux state and React state
- State shared amongst components should generally be stored in Redux
- React state good when only that component (or maybe immediate children) cares about it

Redux State

- Todos list
- User
- What else?

React State

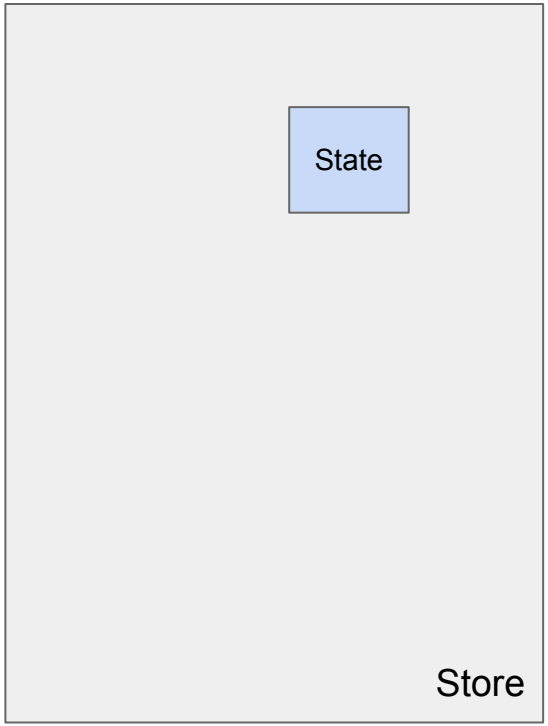
- New todo input
- Isediting user profile name/some input field
- What else?

State

- The data representing app information
- Can be any JS value
- Typically an object

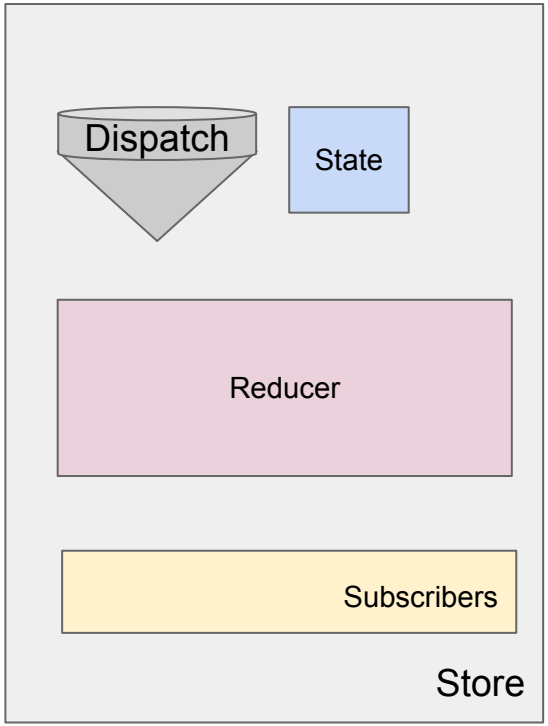
Redux Store

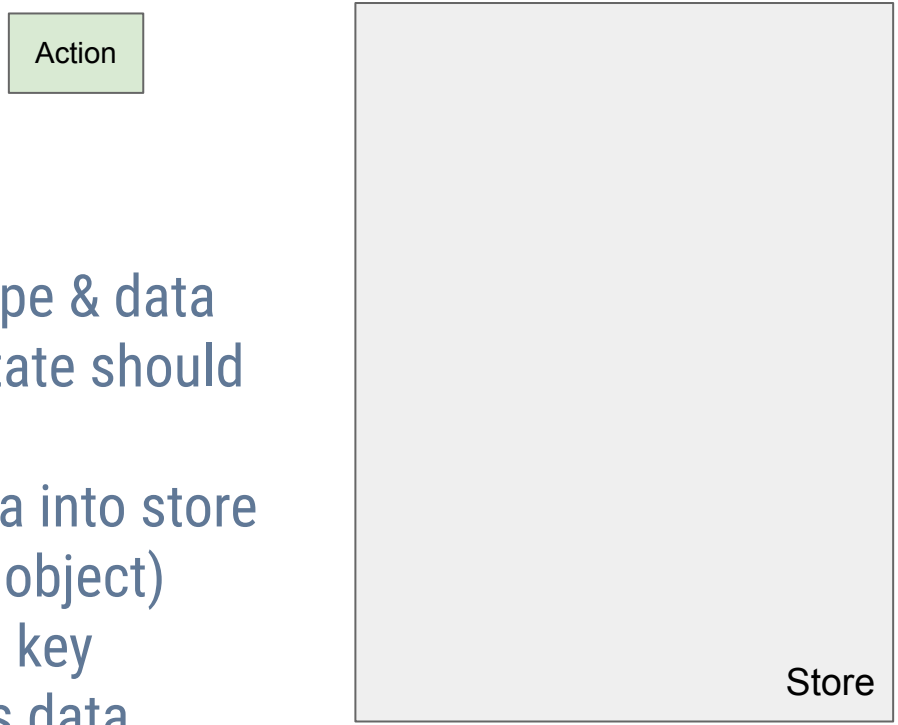
- The container for state
- Object with a few methods
  - `getState`
  - `dispatch`
    - Tells state to update
    - Sends “commands” or actions, state knows how to update itself



Store

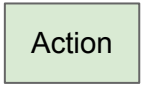
- Several different pieces
- Let's talk about each…
- New Redux terminology:
  - Actions, reducers, dispatch





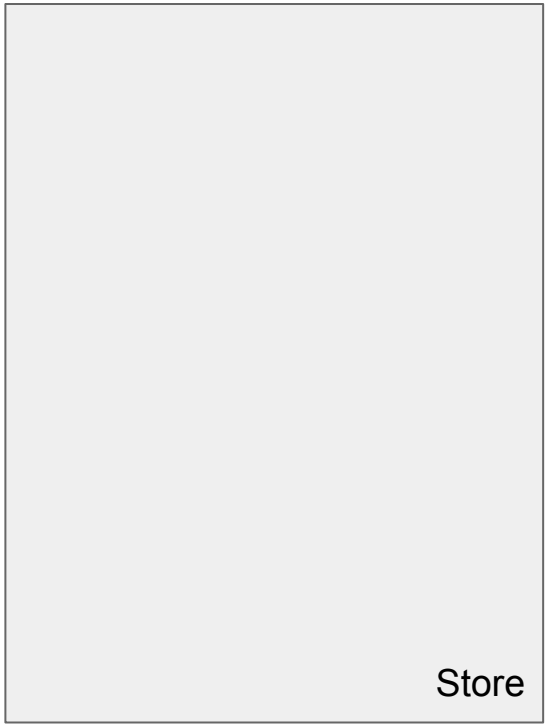
Action

- Describes event (type & data payload) for how state should update
- Only way to get data into store
- POJO (plain old JS object)
  - Requires `type` key
  - Maybe includes data
- {ADD-ITEM-TO-CART, actualItemthatneedsto beadded}

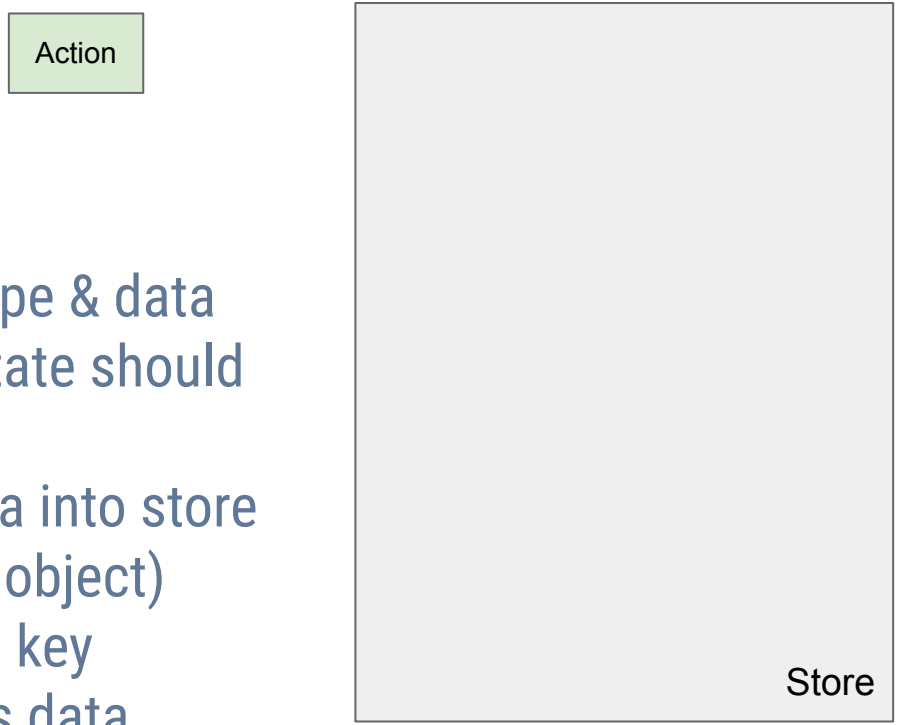


Action

- Describes event (type & data payload) for how state should update
- Only way to get data into store
- POJO (plain old JS object)
  - Requires `type` key
  - Maybe includes data




```
{
  type: ‘INCREMENT’
}
```

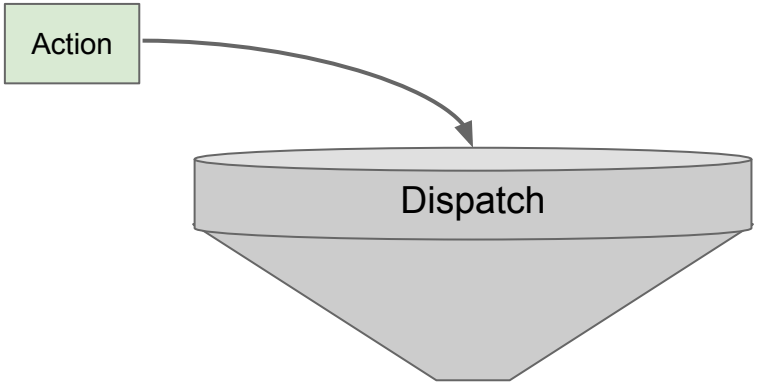


Action

- Describes event (type & data payload) for how state should update
- Only way to get data into store
- POJO (plain old JS object)
  - Requires `type` key
  - Maybe includes data

```
{
  type: ‘INCREMENT’
}
```

```
{
  type: ‘ADD_TODO’,
  todo: ‘Learn Redux’
}
```

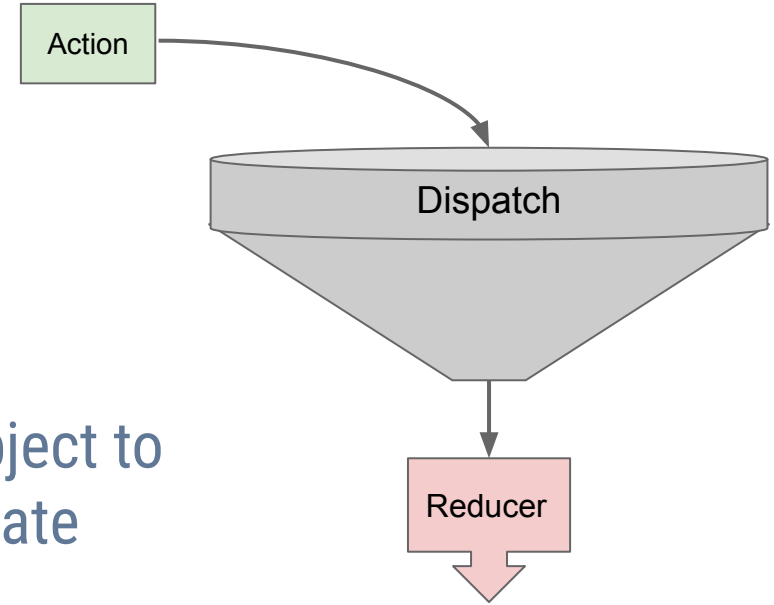


Dispatch

- The mechanism that delivers the action object to the right place to update state
- dispatch(action) -> updates the Redux store!

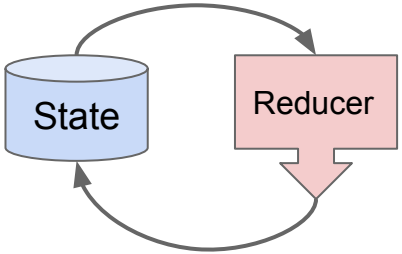
Dispatch

- The mechanism that delivers the action object to the right place to update state
- Sends actions to reducers



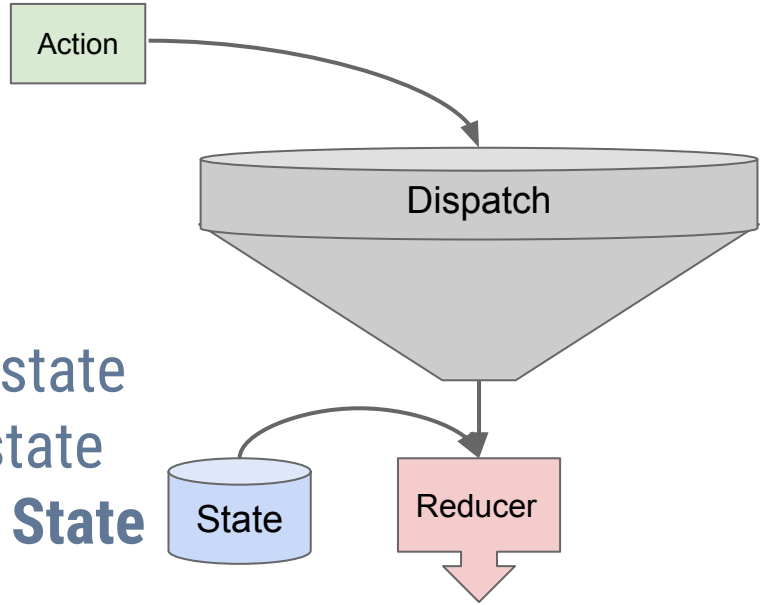
Reducer

- Function that takes a state and returns an updated state
- Actions are like events
- Reducer like event handler



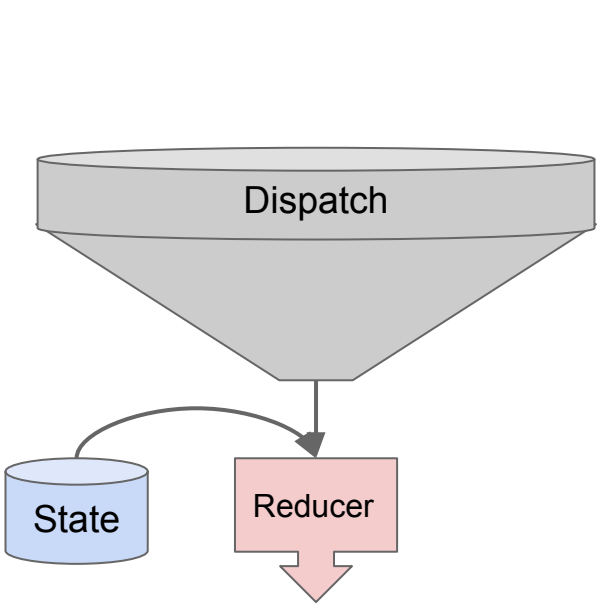
Reducer

- Function that takes a state and returns updated state
- State + Action -> **New State**



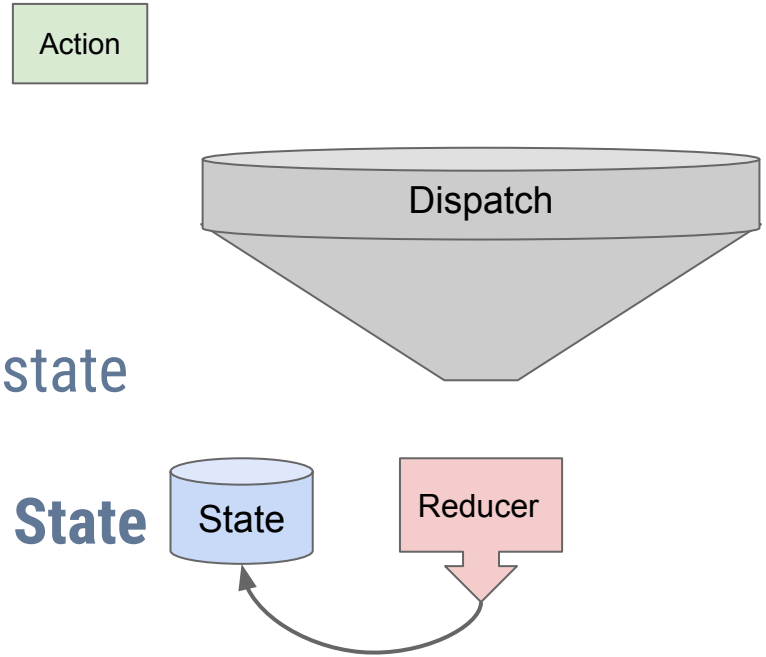
Reducer

- Function that takes a state and returns a state
- State + Action -> **New State**



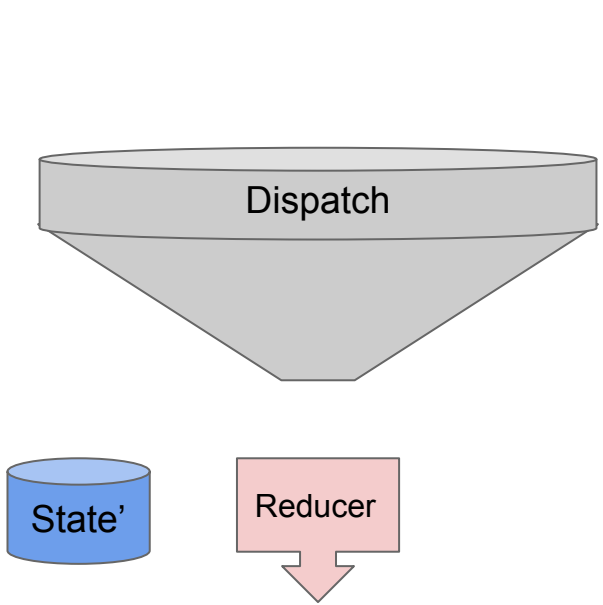
Reducer

- Function that takes a state and returns a state
- State + Action -> **New State**



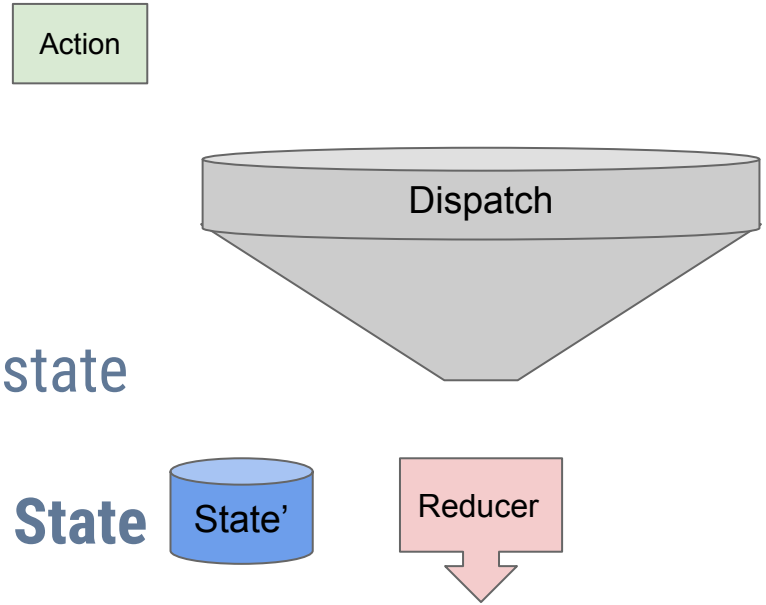
Reducer

- Function that takes a state and returns a state
- State + Action -> **New State**
- **dispatch({DELETE_TODO, todo.id})**



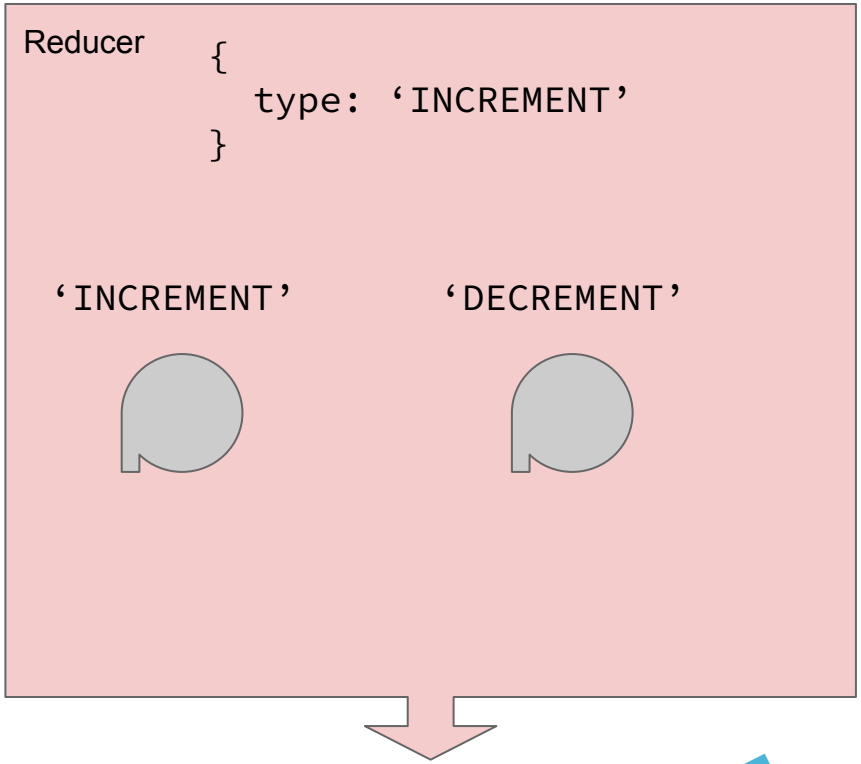
Reducer

- Function that takes a state and returns a state
- State + Action -> **New State**
- Pure function
  - No side effects
  - No mutations



Reducer

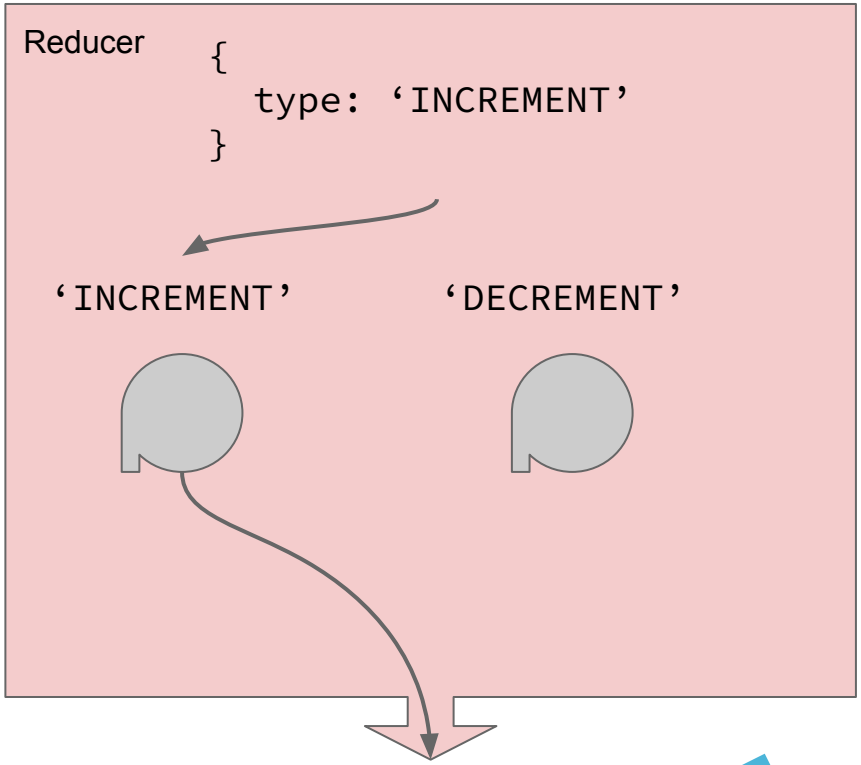
- Basic style:
  - switch/case statement



Reducer

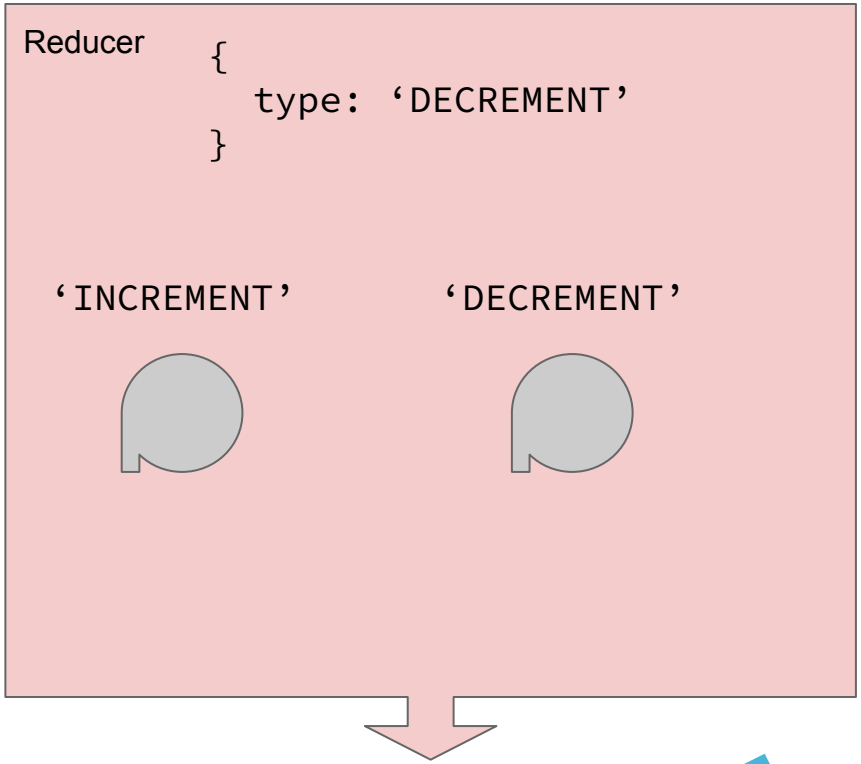
- Basic style:
  - switch/case statement

```
switch(action.type)
{ case “delete_todo:
...update the state
Case “add_todo::
...update state
```



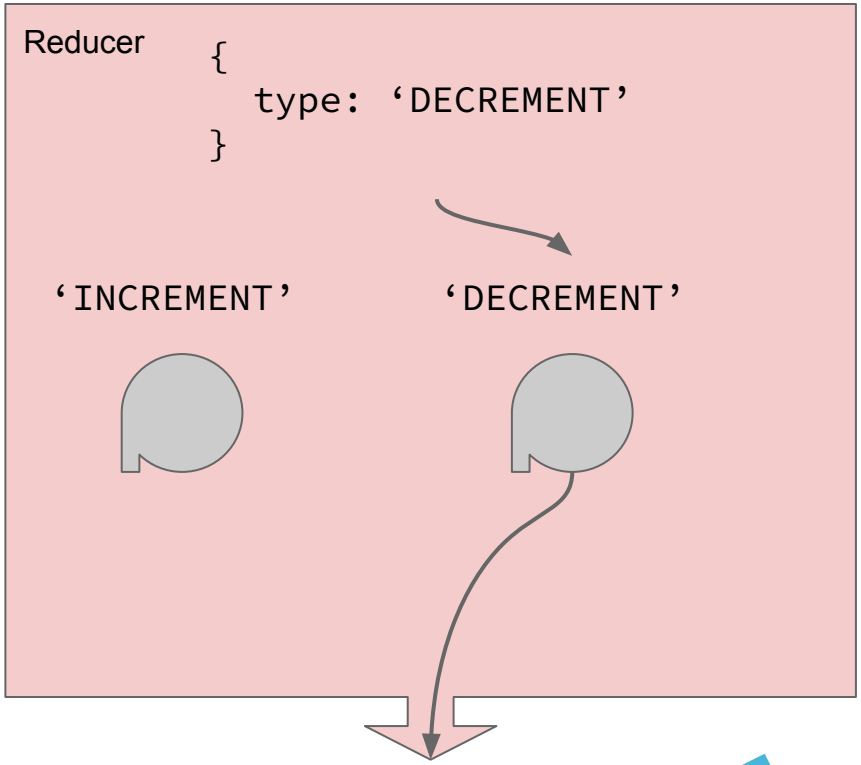
Reducer

- Basic style:
  - switch/case statement



Reducer

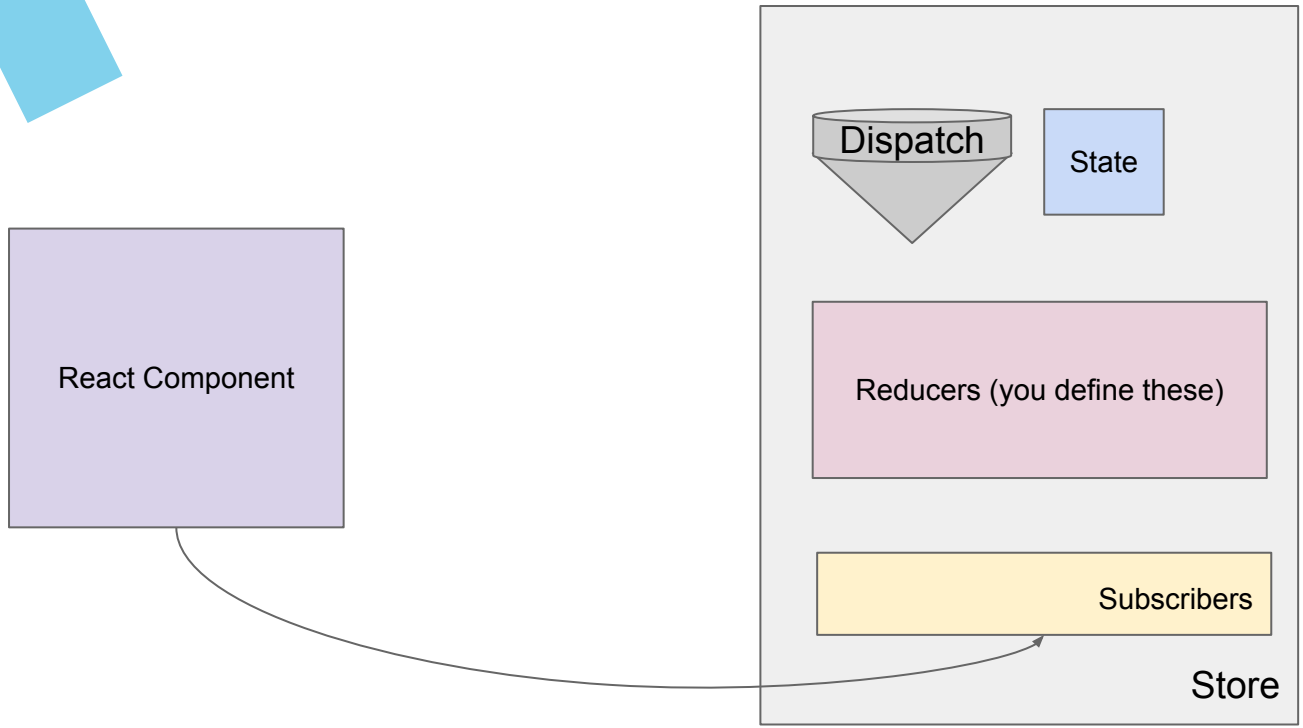
- Basic style:
  - switch/case statement



Reducer

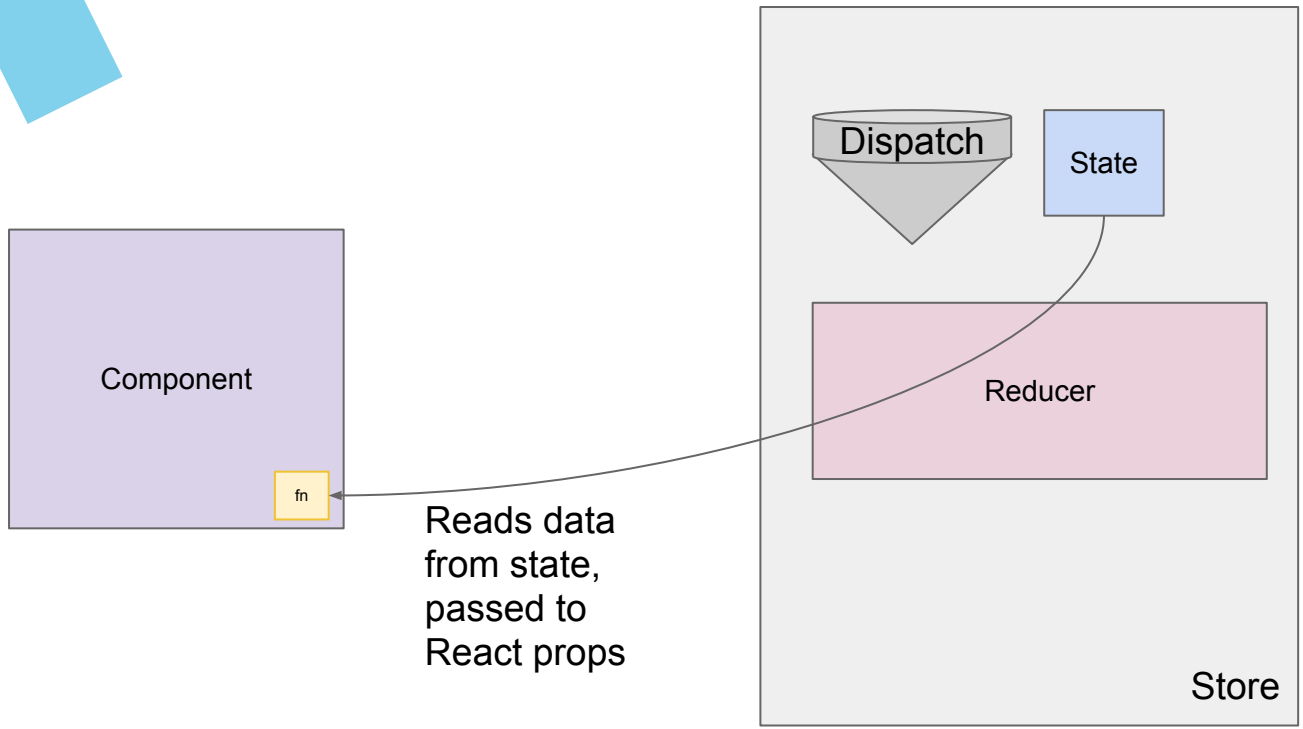
- Basic style:
  - switch/case statement

```
const counterReducer = (state = 0, action) => {
  switch (action ) {
    case 'INCREMENT': return state + 1
    case 'DECREMENT': return state - 1
    default: return state
  }
}
```

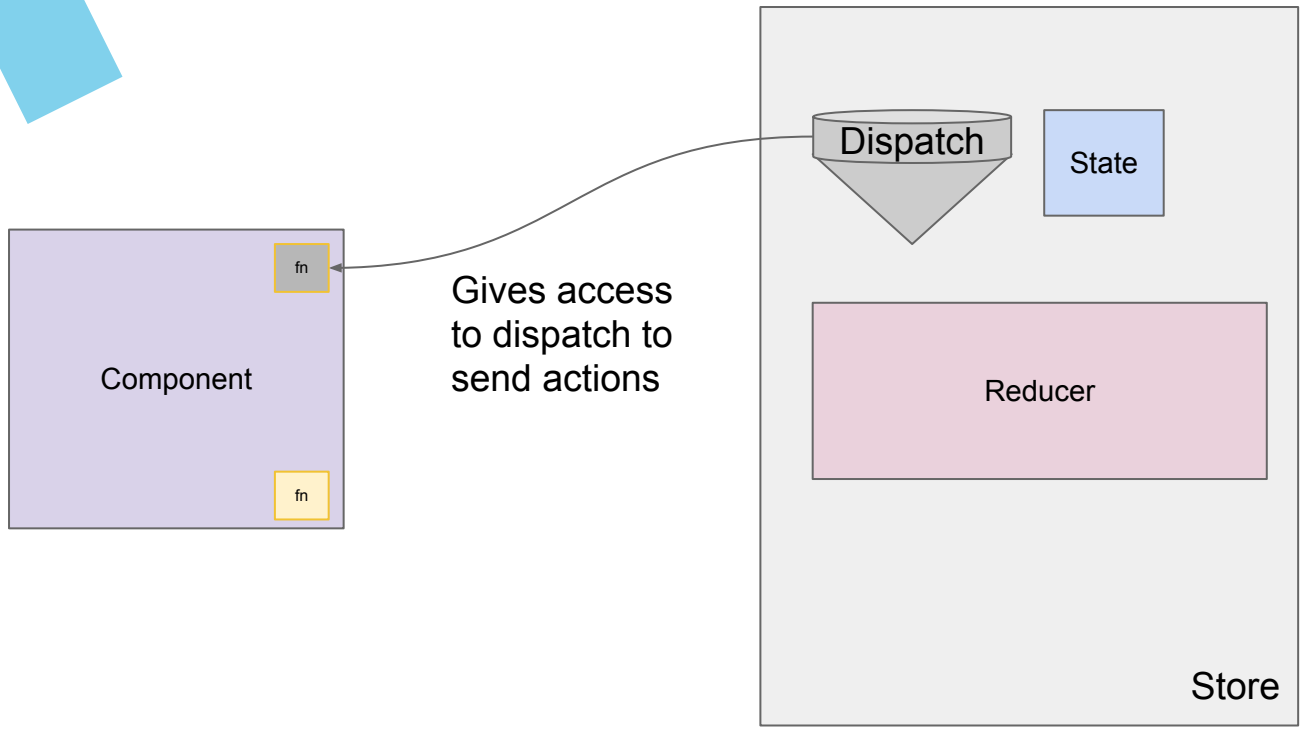


React + Redux

connect to store:
- access state
- access dispatch
- subscribe to updates

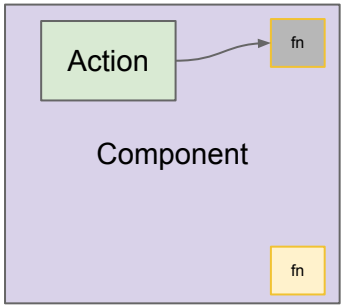

Dispatch
State
Component
Reducer
fn
Reads data
from state,
passed to
React props
Store

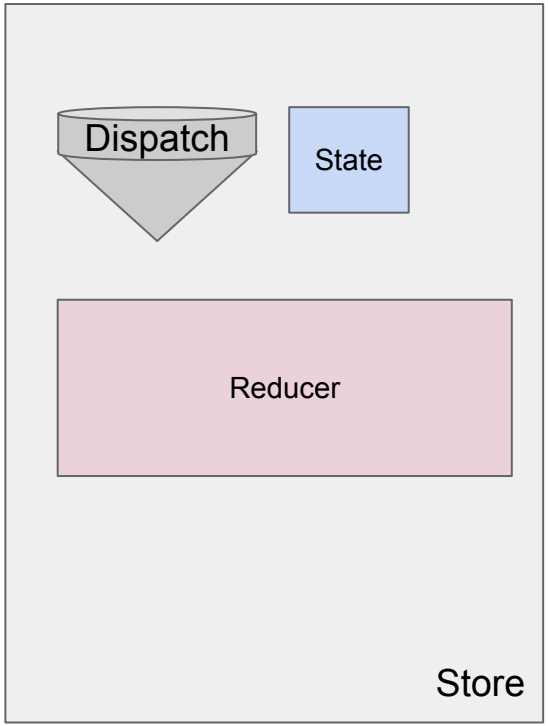
React + Redux


Dispatch
State
fn
Component
fn
Gives access
to dispatch to
send actions
Reducer
Store

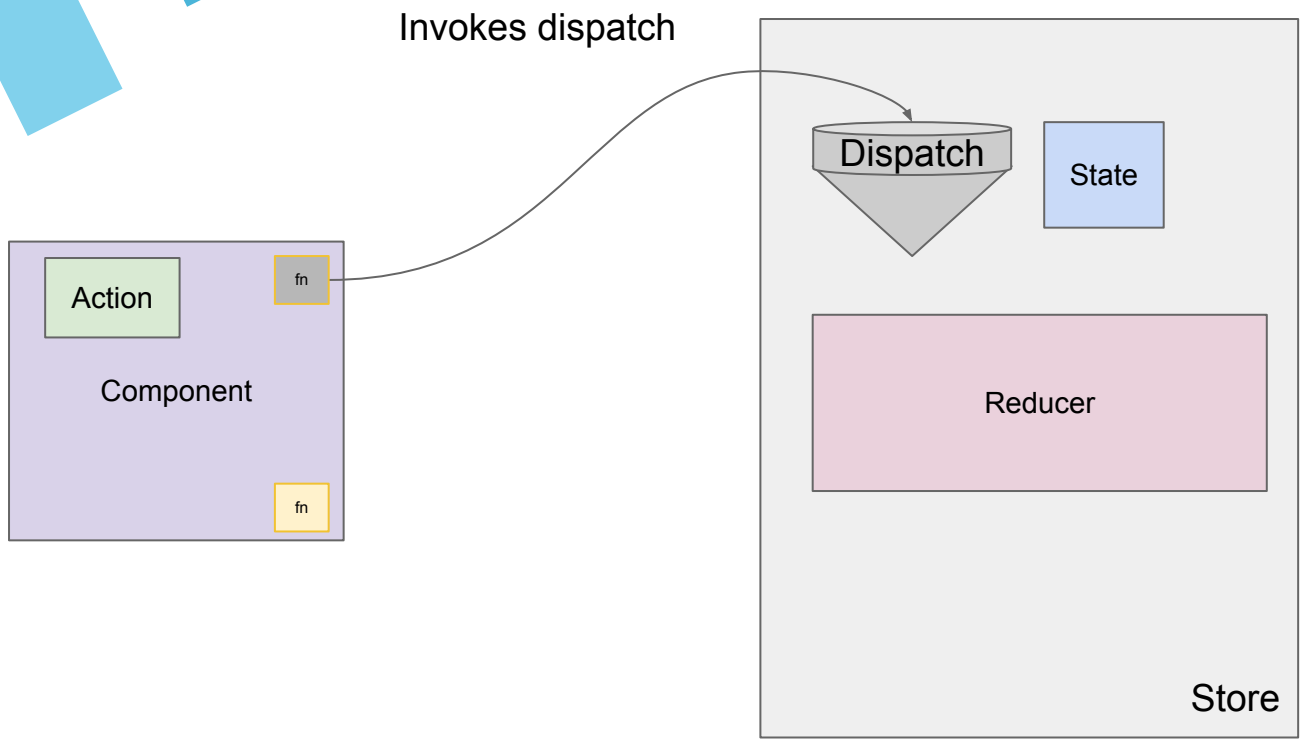
React + Redux

An event happens...

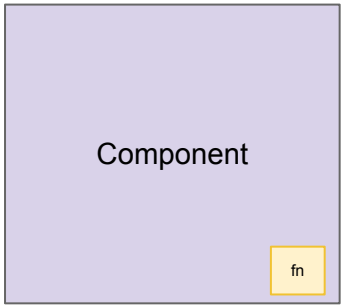


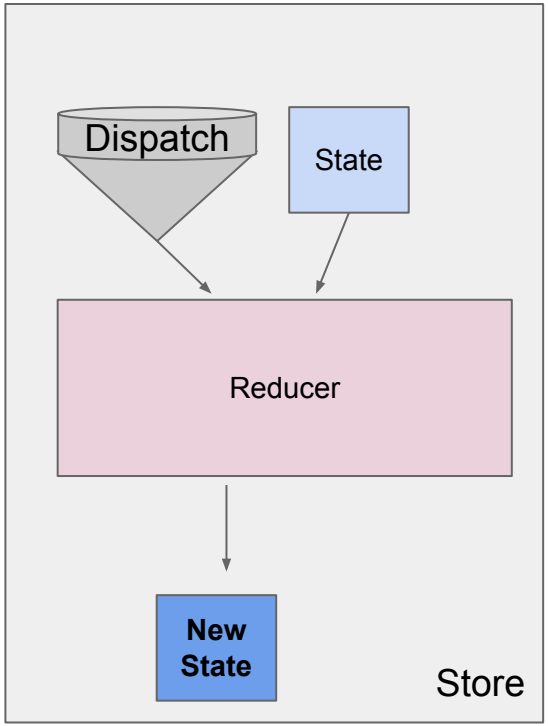


React + Redux

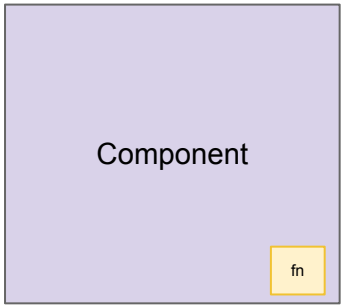

Invokes dispatch
Dispatch
State
Action
fn
Component
fn
Reducer
Store

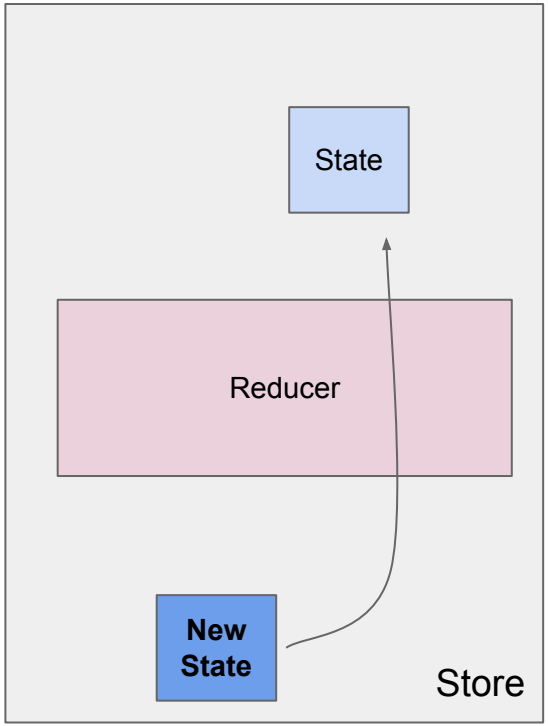
React + Redux



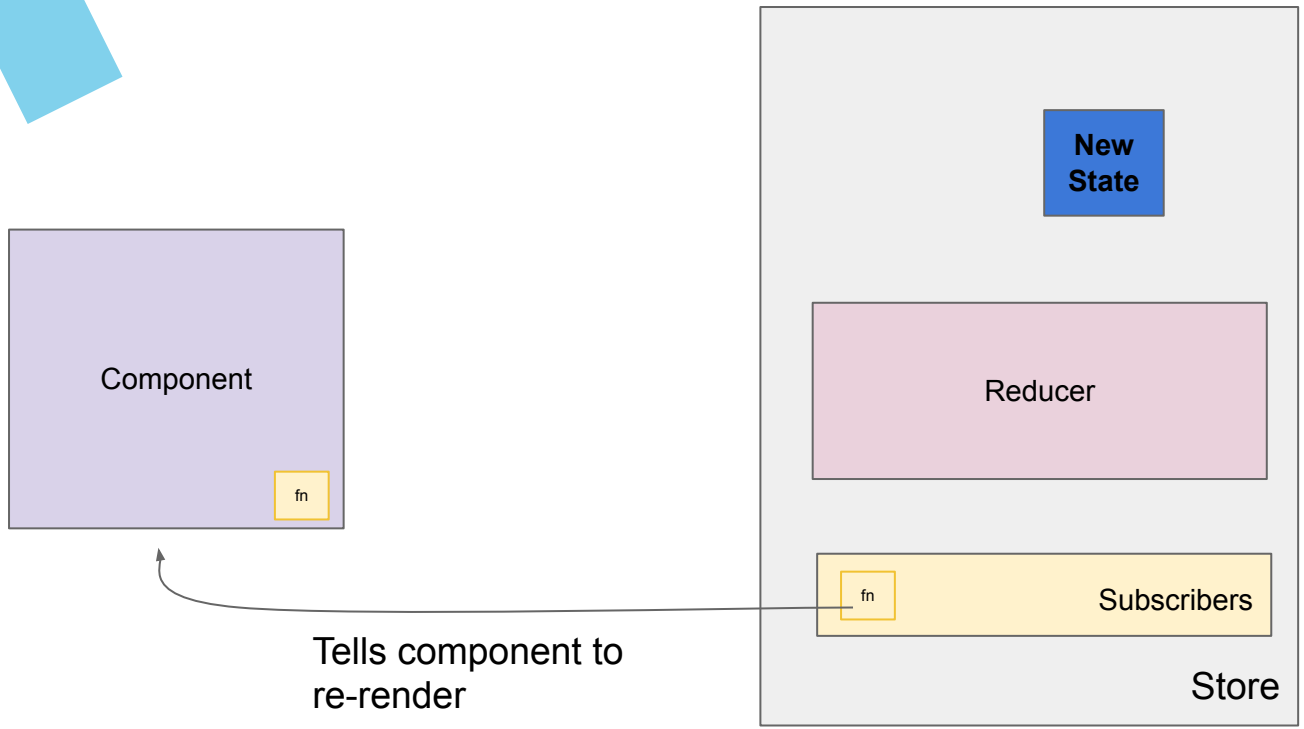


React + Redux

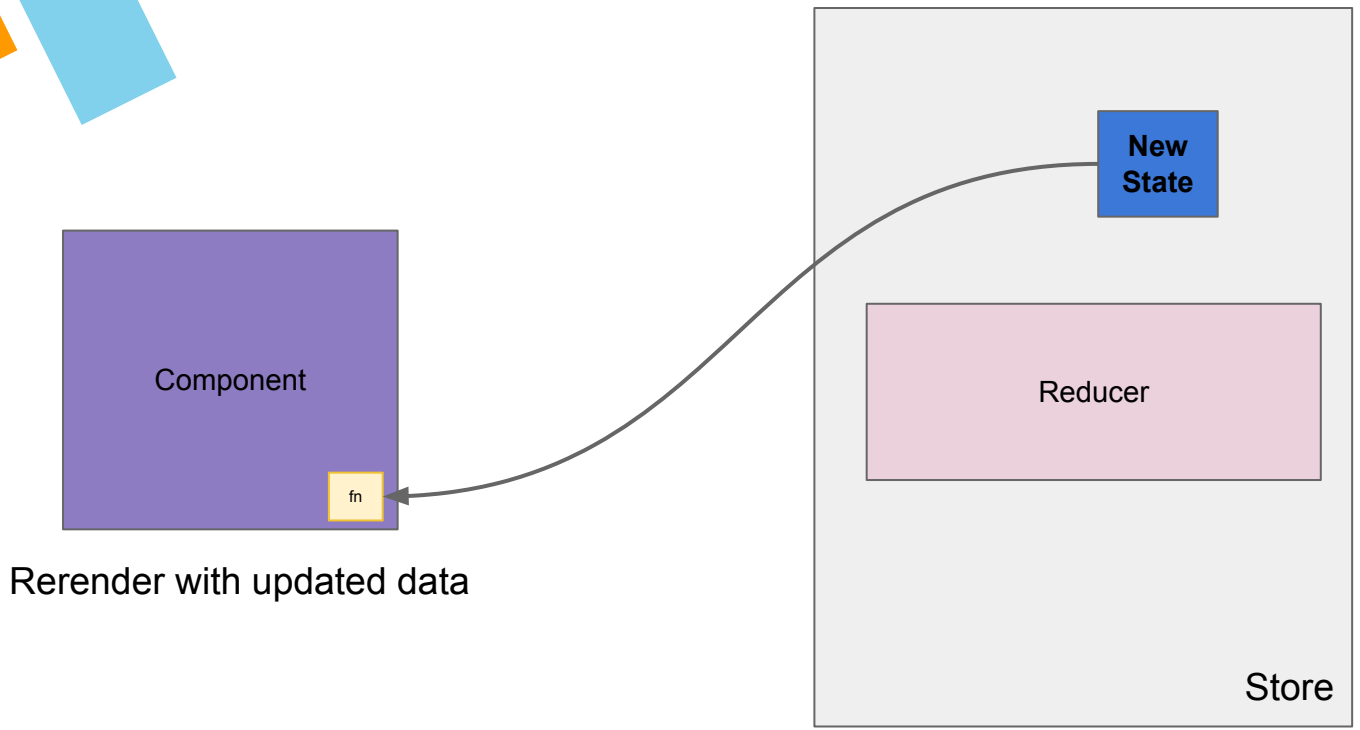




React + Redux


New State
Component
fn
Reducer
fn
Subscribers
Tells component to re-render
Store

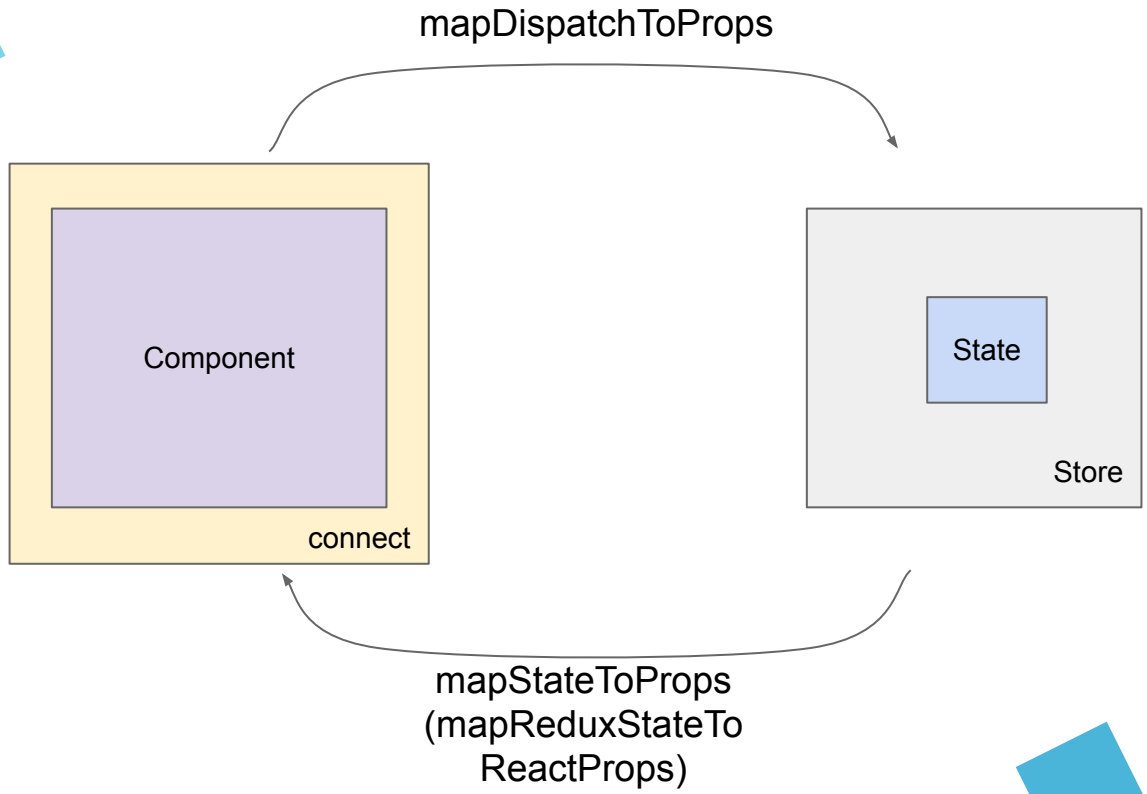
React + Redux


New State
Component
fn
Reducer
Store
Rerender with updated data

React + Redux

Redux Toolkit

- Makes Redux easier!
  - Less boilerplate
  - Packages included
  - Abstracts complexity away
- We will be using it in lab 2!


mapDispatchToProps
Component
connect
State
Store
mapStateToProps
(mapReduxStateTo
ReactProps)

React + Redux

mapStateToProps

```
const mapStateToProps = (state, ownProps) => {
  return {

  }
}
```

mapStateToProps

redux state

props passed directly on React component

```
const mapStateToProps = (state, ownProps) => {
  return {

  }
}
```

returned object is merged into component props

mapDispatchToProps

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

mapDispatchToProps

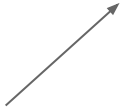
dispatch talks to reducers

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

returned object merged into component props

mapDispatchToProps

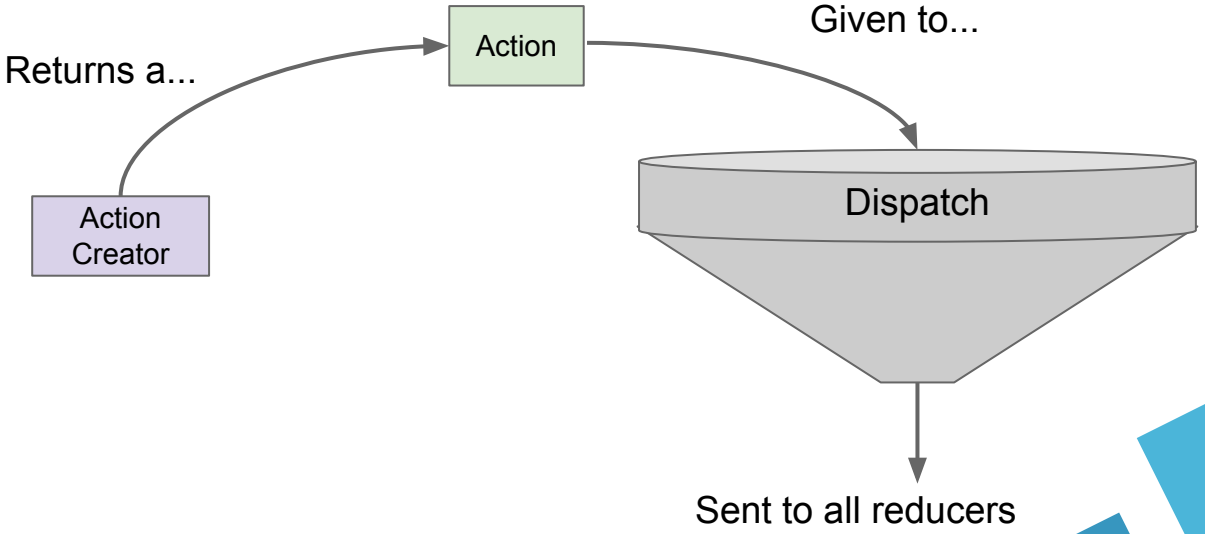
```
const mapDispatchToProps = (dispatch) => {
  return {
    increment() { dispatch({ type: 'INCREMENT' }) },
    decrement() { dispatch({ type: 'DECREMENT' }) },
  }
}
```



These “action types” might be reused!

Action Creators

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



Action Creators

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

You can then use these functions as a shorthand for the mapDispatchToProps

```
import { increment, decrement } from './actions'

const mapDispatchToProps = {
  increment,
  decrement,
}
```