

Actions, Store, and Reducers



Cory House

Consultant

@housecor | www.reactjsconsulting.com

Agenda



Actions

Store

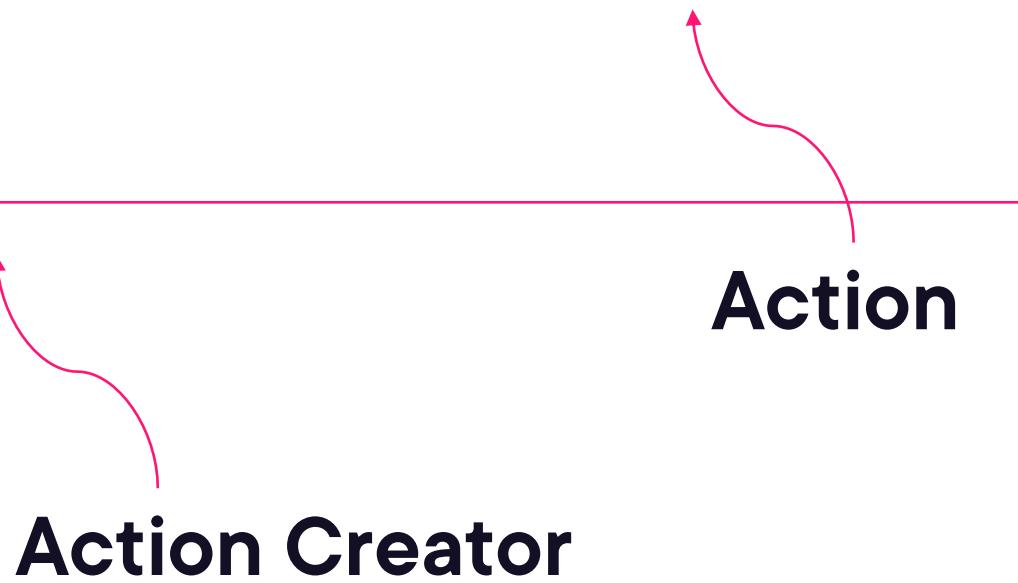
Immutability

Reducers

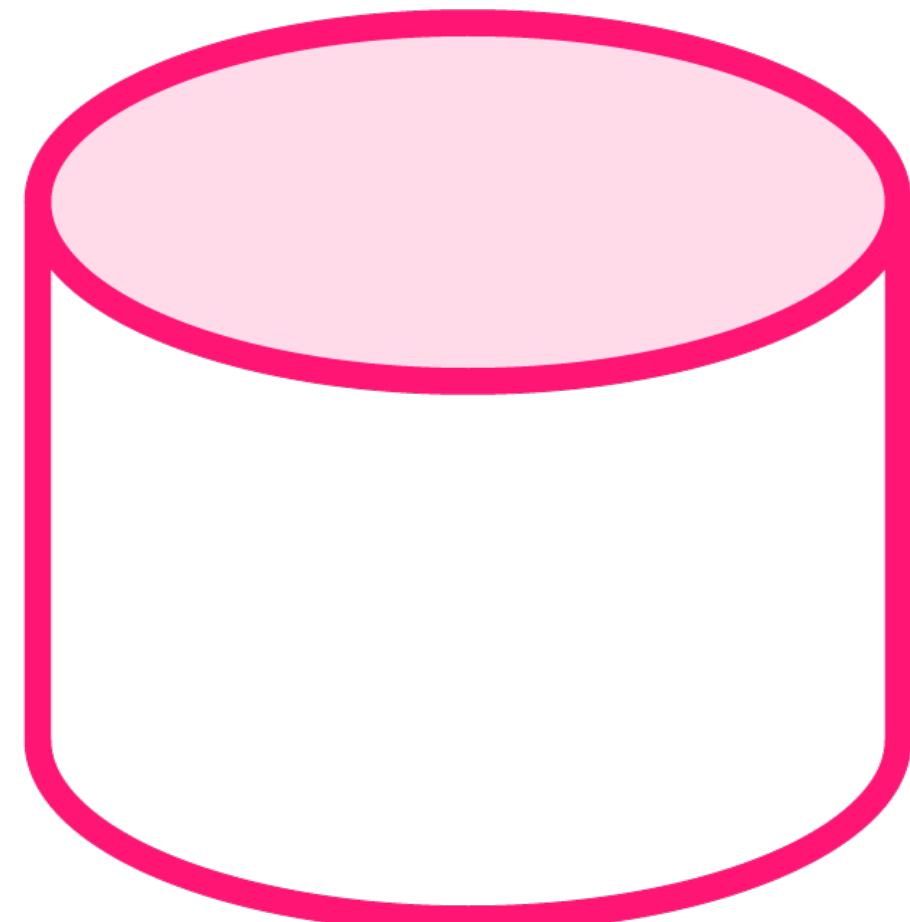


Action Creators

```
rateCourse(rating) {  
  return { type: RATE.Course, rating: rating }  
}
```



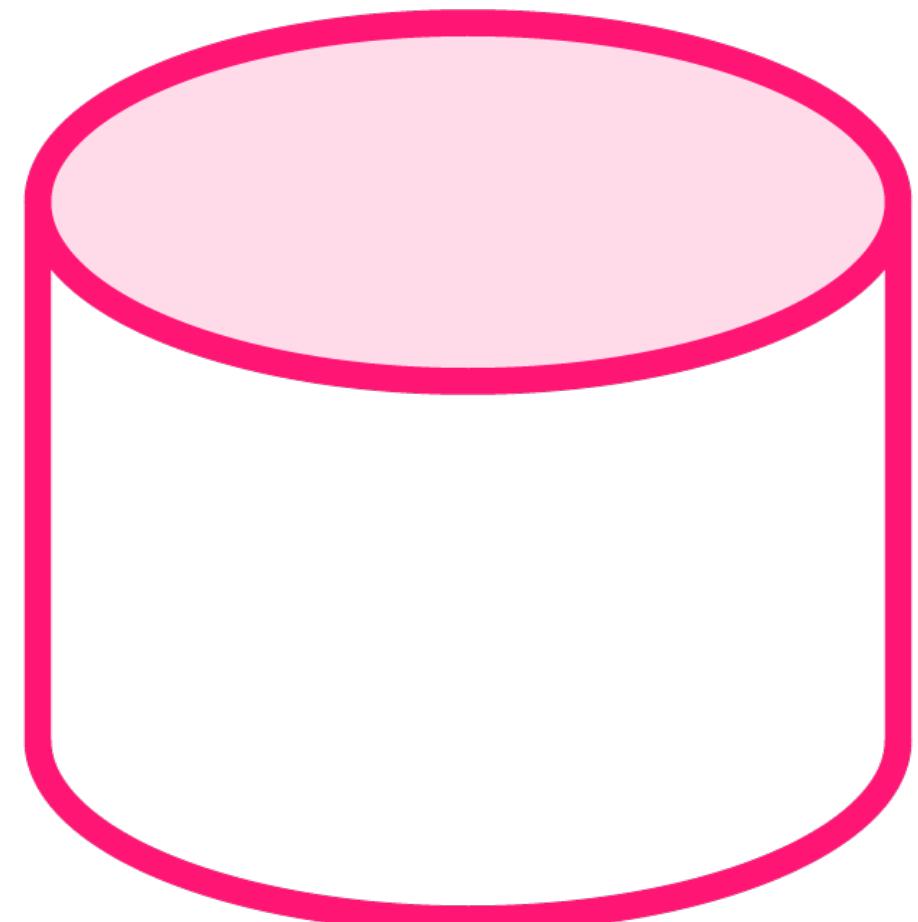
Creating Redux Store



```
let store = createStore(reducer);
```



Redux Store



`store.dispatch(action)`
`store.subscribe(listener)`
`store.getState()`
`replaceReducer(nextReducer)`





I Immutability

Immutability

To change state, return a new object.



What's Mutable in JS?

Immutable already! 😊

VS

Mutable

number

string

boolean

undefined

null

objects

arrays

functions



Current state

```
state = {  
  name: 'Cory House'  
  role: 'author'  
}
```

Mutating state

```
state.role = 'admin';  
return state;
```



Current state

Return new object

```
state = {  
  name: 'Cory House'  
  role: 'author'  
}
```

```
return state = {  
  name: 'Cory House'  
  role: 'admin'  
}
```



Handling Immutable Data in JS

`Object.assign`

`{ ...myObj }`

`.map`

`Object.assign`

`Spread syntax`

`Immutable-friendly
array methods`



Copy via Object.assign

```
Object.assign({}, state, { role: "admin" }));
```



Copy via Spread

```
const newState = { ...state, role: 'admin' };
```

```
const newUsers = [ ...state.users ]
```



Warning: Shallow Copies

```
const user = {  
  name: 'Cory',  
  address: {  
    state: 'California'  
  }  
}
```

// Watch out, it didn't clone the nested address object!
const userCopy = { ...user };

// This clones the nested address object too
const userCopy = { ...user, address: { ...user.address } };



Warning: Only Clone What Changes

You might be tempted to use deep merging tools like [clone-deep](#), or [lodash.merge](#), but avoid blindly deep cloning.

Here's why:

1. Deep cloning is expensive
2. Deep cloning is typically wasteful
3. Deep cloning causes unnecessary renders

Instead, clone only the sub-object(s) that have changed.



Changing Data via Immer

```
import produce from "immer"

const user = {
  name: "Cory",
  address: {
    state: "California"
  }
}

const userCopy = produce(user, draftState => {
  draftState.address.state = "New York"
})

console.log(user.address.state); // California
console.log(userCopy.address.state) // New York
```



Array

Languages

Edit

Jump to: [Syntax](#) [Description](#) [Properties](#) [Methods](#) [Array instances](#) [Array generic methods](#) [Examples](#) [Specifications](#) [Browser compatibility](#) [See also](#)

Web technology for developers > [JavaScript](#) >
[JavaScript reference](#) >
[Standard built-in objects](#) > [Array](#)

Related Topics

[Standard built-in objects](#)[Array](#)

Properties

[Array.length](#)[Array.prototype](#)[Array.prototype\[@@unscopables\]](#)

Methods

[Array.from\(\)](#)[Array.isArray\(\)](#) [Array.observe\(\)](#)[Array.of\(\)](#)[Array.prototype.concat\(\)](#)[Array.prototype.copyWithin\(\)](#)[Array.prototype.entries\(\)](#)[Array.prototype.every\(\)](#)[Array.prototype.fill\(\)](#)[Array.prototype.filter\(\)](#)

The JavaScript **Array** object is a global object that is used in the construction of arrays; which are high-level, list-like objects.

Create an Array

```
1 | var fruits = ['Apple', 'Banana'];
2 |
3 | console.log(fruits.length);
4 | // 2
```

Access (index into) an Array item

```
1 | var first = fruits[0];
2 | // Apple
3 |
4 | var last = fruits[fruits.length - 1];
5 | // Banana
```

Loop over an Array

```
1 | fruits.forEach(function(item, index, array) {
2 |   console.log(item, index);
3 | });
4 | // Apple 0
5 | // Banana 1
```

Handling Arrays

Avoid

push

pop

reverse

} **Must clone array first**

Prefer

map

filter

reduce

concat

spread



Handling Immutable State

Native JS

Object.assign

Spread operator

Map, filter, reduce

Libraries

Immer

seamless-immutable

react-addons-update

Immutable.js

Many more



Flux

VS

Redux

State is mutated

State is immutable



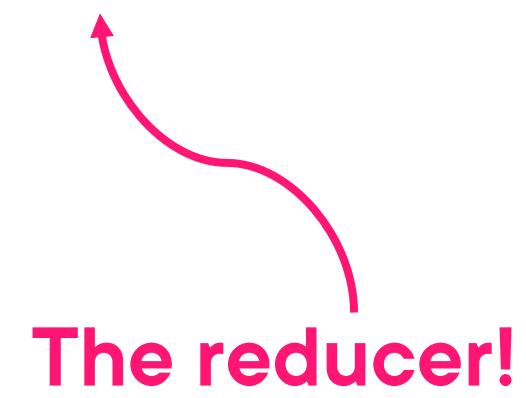
Why Immutability?

- Clarity
- Performance
- Awesome Sauce



Immutability = Clarity

“Huh, who changed that state?”



The reducer!



Why Immutability?

- Clarity
- Performance
- Awesome sauce



Immutability = Performance

```
state = {  
  name: 'Cory House',    ← Has this changed?  
  role: 'author',  
  city: 'Kansas City',  
  state: 'Kansas',  
  country: 'USA',  
  isFunny: 'Rarely',  
  smellsFunny: 'Often'  
  ...  
}
```



```
if (prevStoreState !== storeState) ...
```



Why Immutability?

- Clarity
- Performance
- Awesome Sauce *(Amazing debugging)*



Immutability = AWESOME SAUCE!

- Time-travel debugging
- Undo/Redo
- Turn off individual actions
- Play interactions back

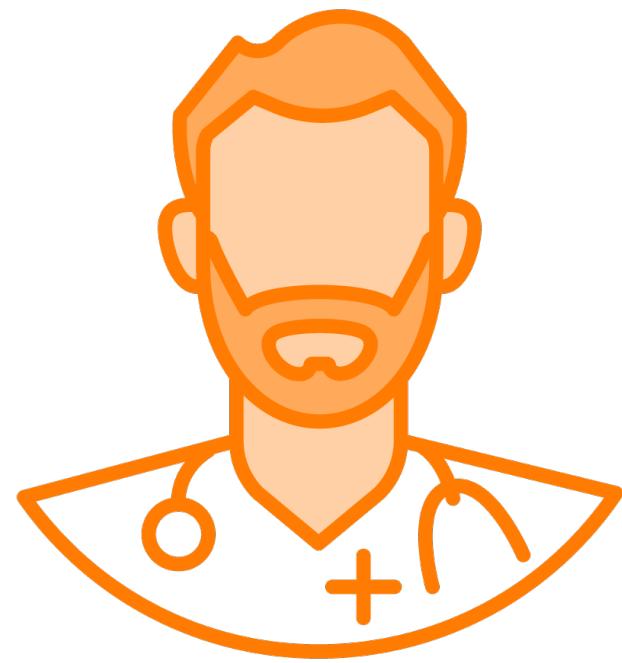




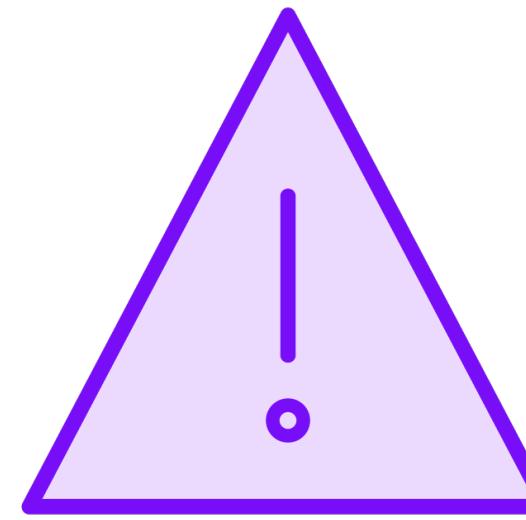
Enforcing Immutability



How Do I Enforce Immutability?

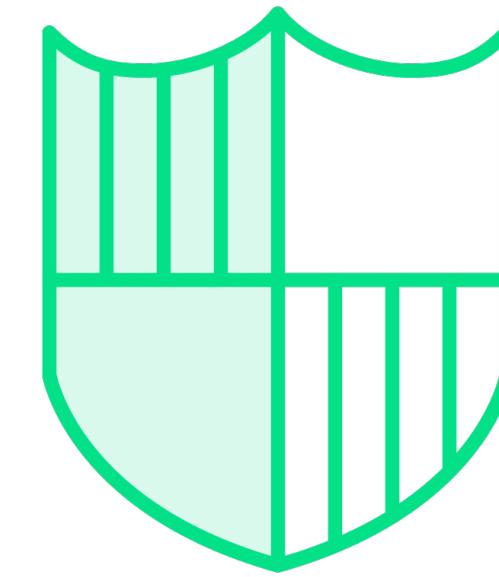


Trust



Warn

@redux/toolkit



Enforce

Immer

Immutable.js

Seamless-immutable



Reducers

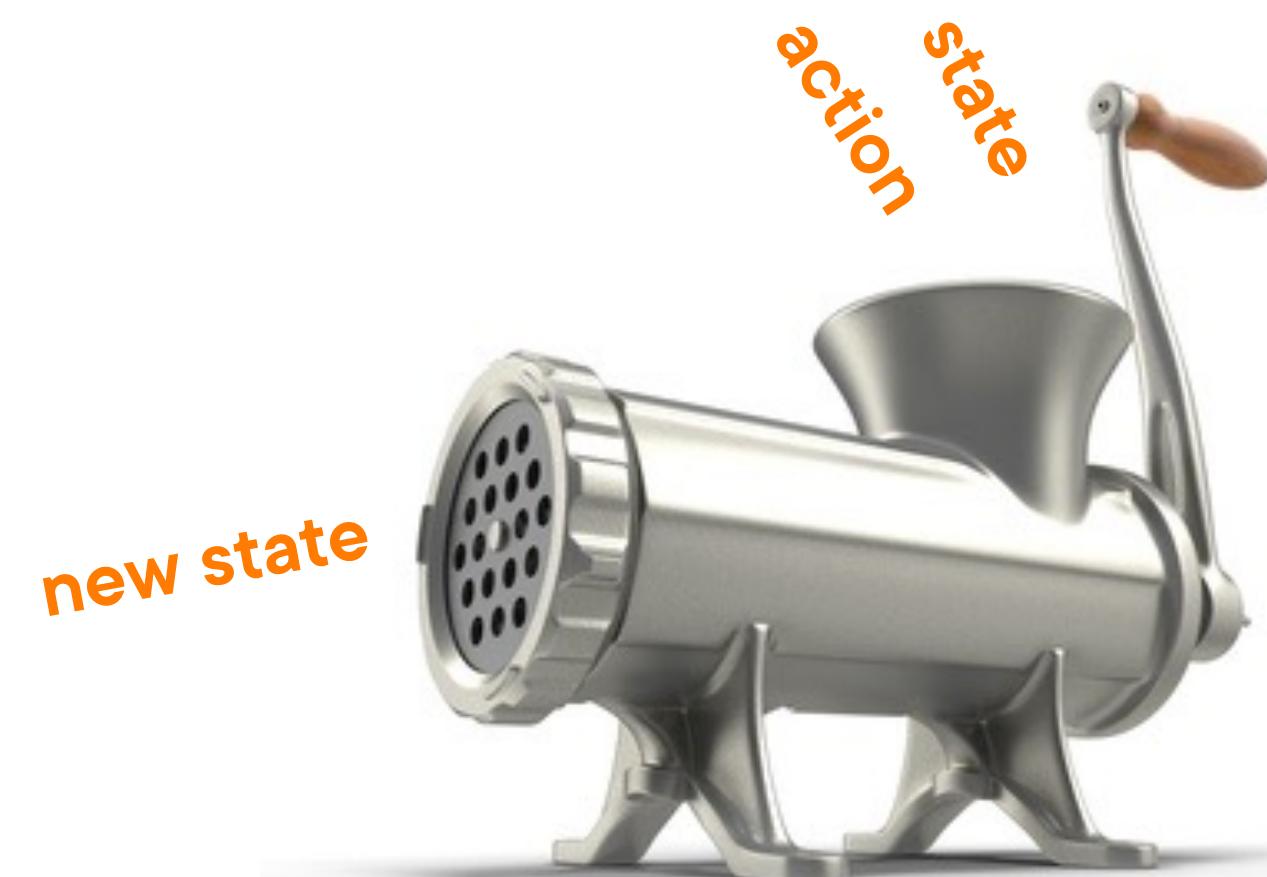


What is a Reducer?

```
function myReducer(state, action) {  
  // Return new state based on action passed  
}
```



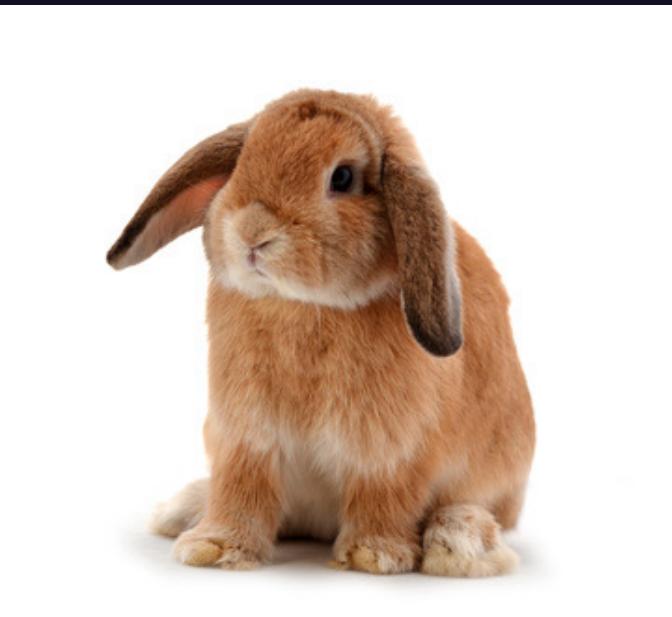
(state, action) => state



What is a Reducer?

```
function myReducer(state, action) {  
  // Return new state based on action passed  
}  
  
}
```

So approachable.
So simple.
~~So tasty.~~



What is a Reducer?

```
function myReducer(state, action) {  
  switch (action.type) {  
    case "INCREMENT_COUNTER":  
      state.counter++; // Uh oh, can't do this!  
      return state;  
    default:  
      return state;  
  }  
}
```



What is a Reducer?

```
function myReducer(state, action) {  
  switch (action.type) {  
    case "INCREMENT_COUNTER":  
      return { ...state, counter: state.counter + 1 };  
    default:  
      return state;  
  }  
}
```



Reducers must be pure.



Forbidden in Reducers

- Mutate arguments
- Perform side effects
- Call non-pure functions

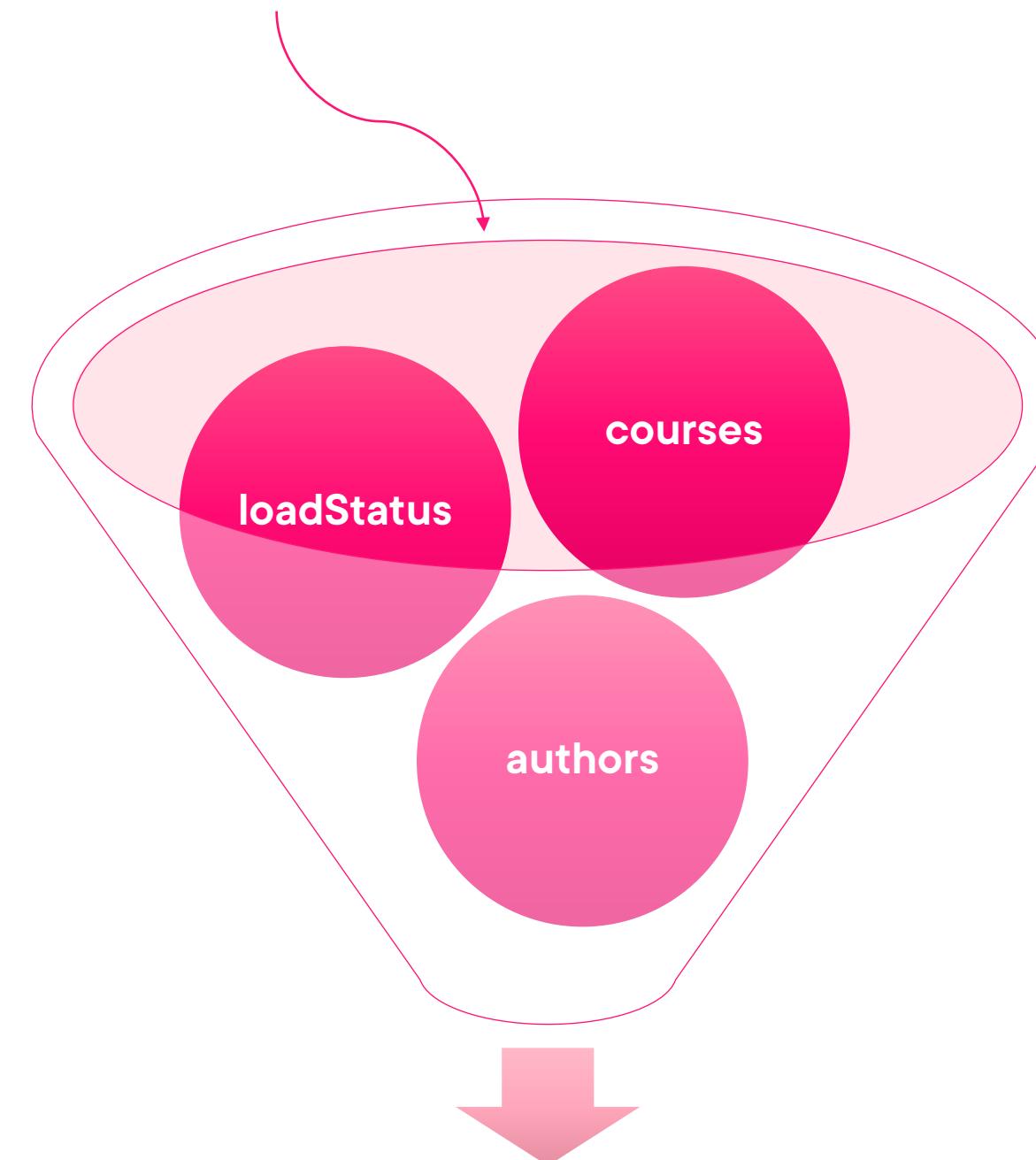


1 Store. Multiple Reducers.



All Reducers Are Called on Each Dispatch

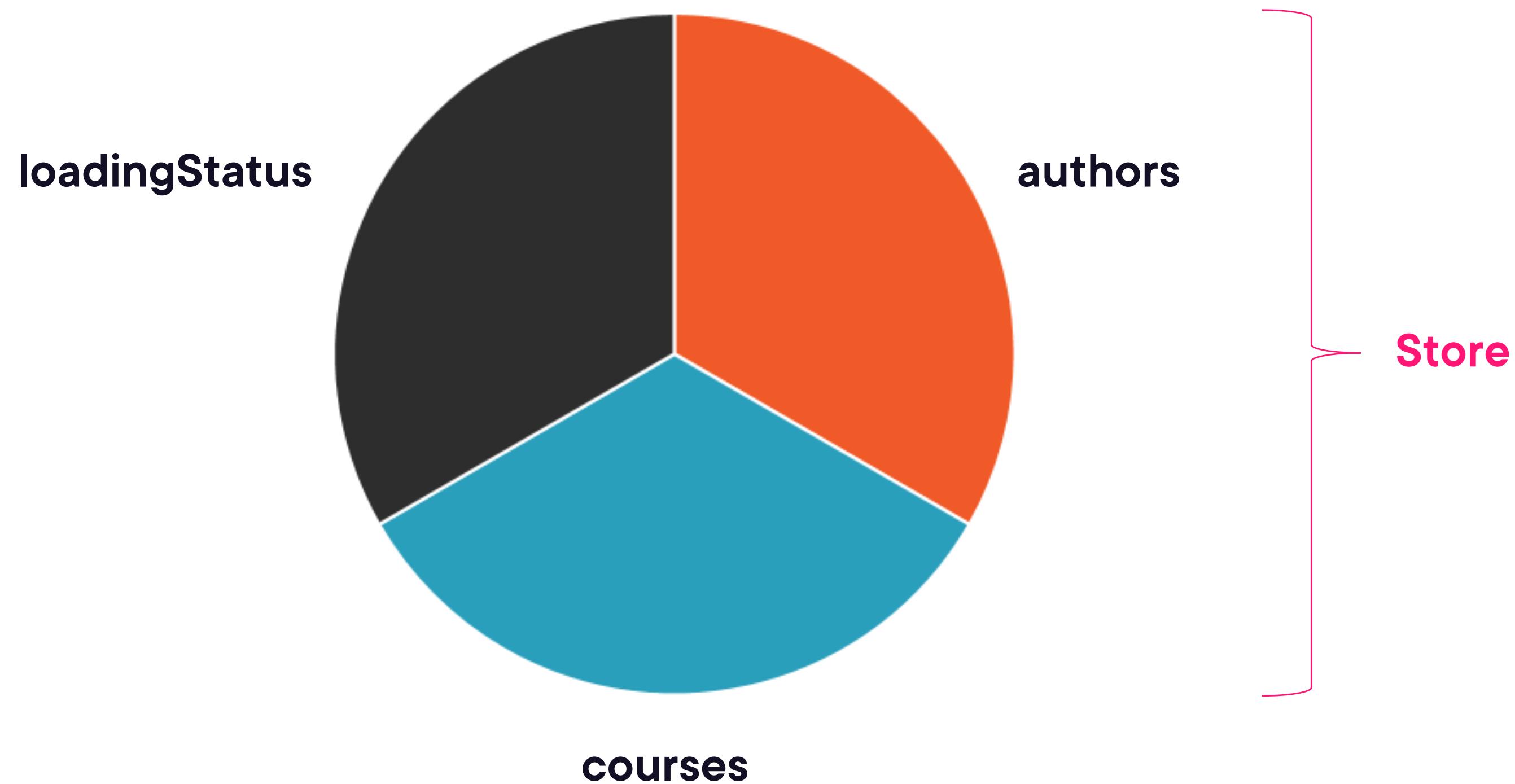
```
{ type: DELETE_COURSE, 1 }
```



New State



Reducer = “Slice” of State



Redux FAQ

“Write independent small reducer functions that are each responsible for updates to a specific slice of state. We call this pattern “reducer composition”. A given action could be handled by all, some, or none of them.”



Summary



Actions

- Represent user intent
- Must have a type

Store

- dispatch, subscribe, getState...

Immutability

- Just return a new copy

Reducers

- Must be pure
- Multiple per app
- Slice of state



Up Next:

Connecting React to Redux

