Here's The Plan

- Actions
- Store
- Immutability
- Reducers

Action Creators

```
rateCourse(rating) {
    return { type: RATE_COURSE, rating: rating }
}
```

Action

Action Creator

Creating Redux Store

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

Redux Store

store.dispatch(action)

store.subscribe(listener)

store.getState()

replaceReducer(nextReducer)

Immutability

Immutability:

To change state, return a new object.

What's Mutable in JS?

Immutable already! ☺	Mutable
Number	*Objects*
String,	*Arrays*
Boolean,	*Functions*
Undefined,	
Null	

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

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

◀ Current state

◀ Traditional App - Mutating state

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

◀ **Current state**

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

◀ **Returning new object.**
Not mutating state! ☺

Copy

Signature

Object.assign(*target*, ...*sources*)

Example

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

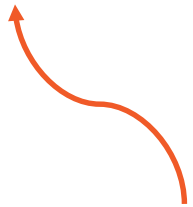
Flux	Redux
State is mutated	State is immutable

Why Immutability?

- **Clarity**
- **Performance**
- **Awesome Sauce**

Immutability = Clarity

“Huh, who changed that state?”



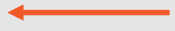
The reducer, stupid!

Why Immutability?

- Clarity
- Performance
- Awesome sauce

Immutability = Performance

```
state = {
    name: 'Cory House'
    role: 'author'
    city: 'Kansas City'
    state: 'Kansas'
    country: 'USA'
    isFunny: 'Rarely'
    smellsFunny: ‘Often'
    ...
}
```



Has this changed?

```
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**

Handling Immutability

Handling Immutable State

ES6

- Object.assign
- Spread operator

ES5

- Lodash merge
- Lodash extend
- Object-assign

Libraries

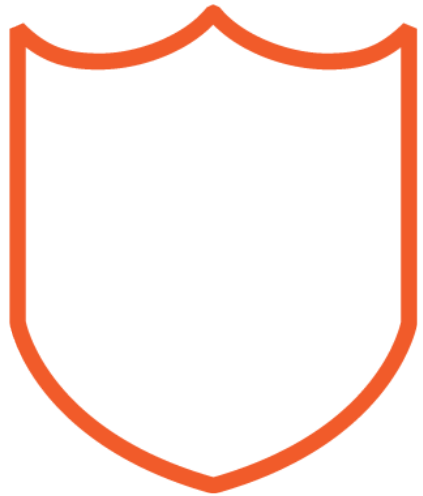
- react-addons-update
- Immutable.js

JavaScript's primitives are **immutable.**

How do I enforce immutability?

Trust

redux-immutable-state-invariant



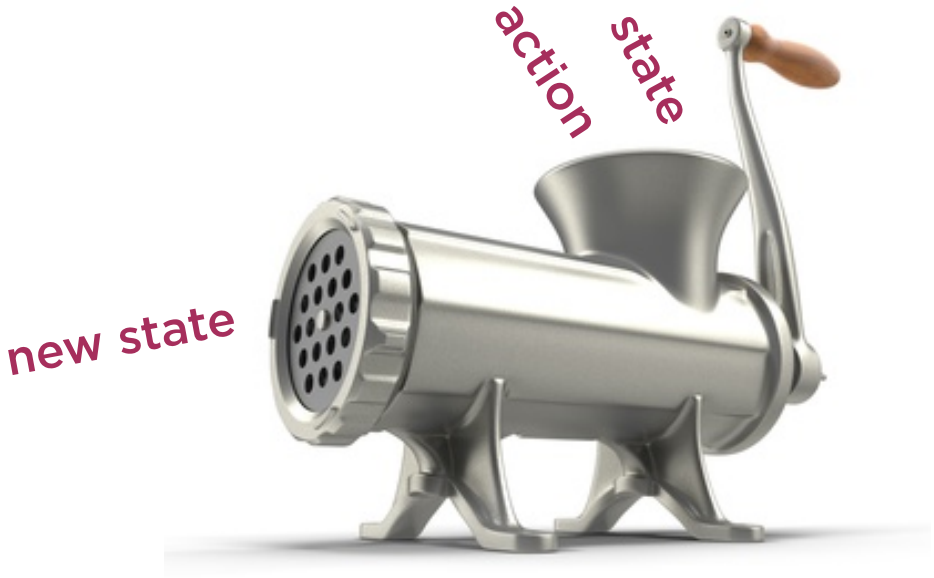
Immutable.js

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++;
            return state;
   }
}
```

Uh oh, can't do this!

What is a Reducer?

```
function myReducer(state, action) {
   switch (action.type) {
      case ‘INCREMENT_COUNTER’:
            return (Object.assign({}, state, counter + 1);
   }
}
```

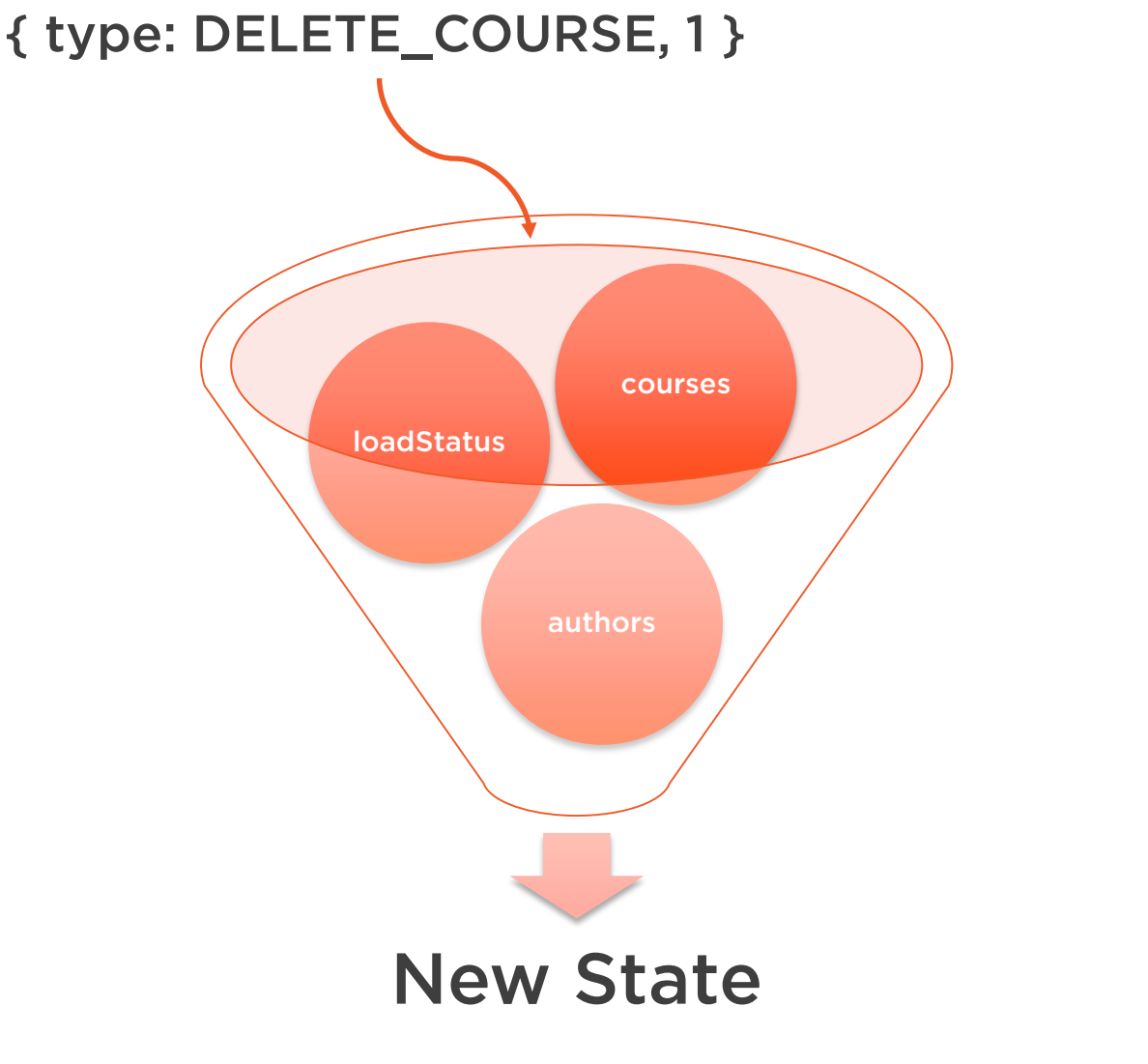
Reducers must be pure.

Forbidden in Reducers

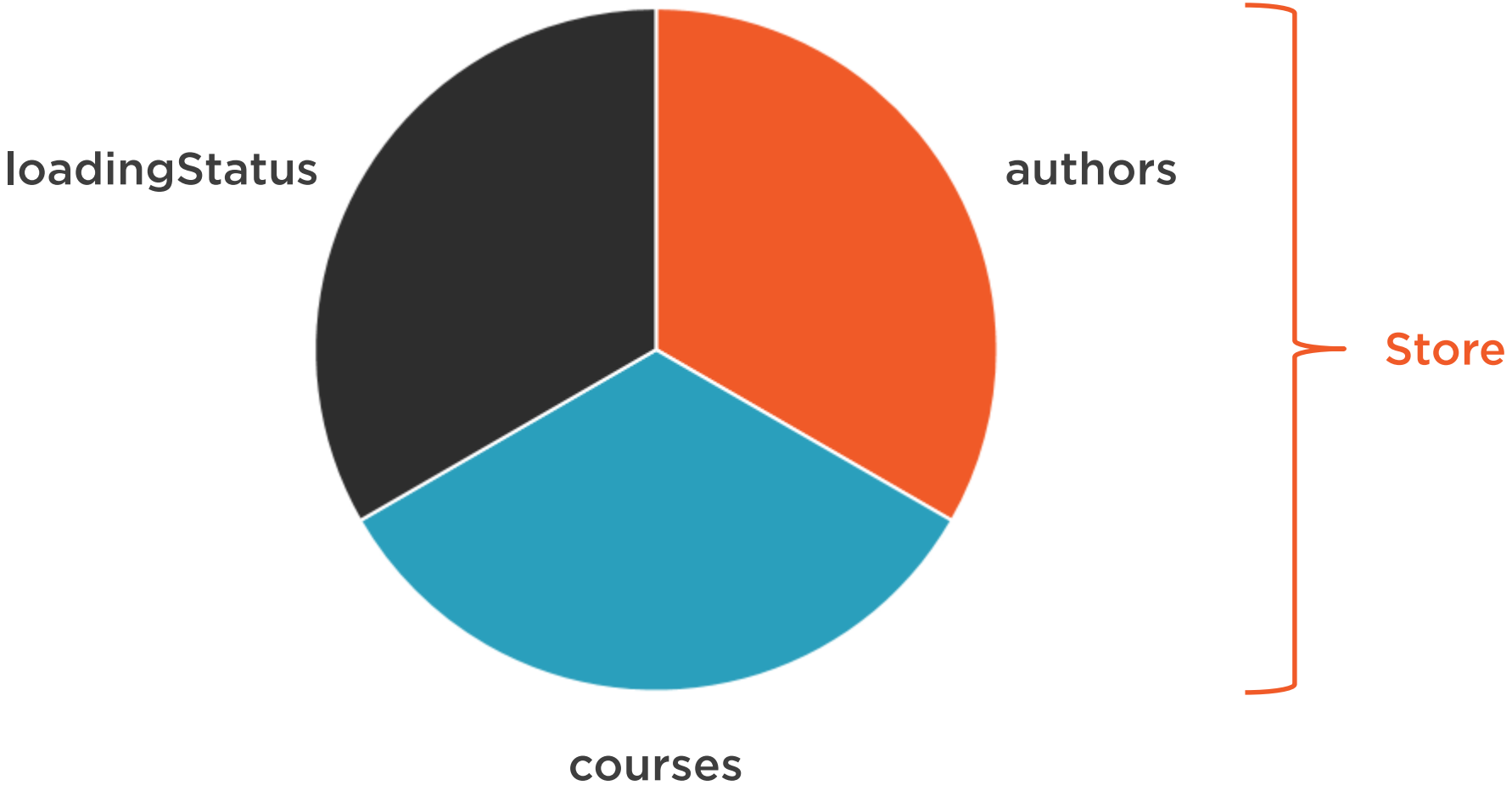
- Mutate arguments
- Perform side effects
- Call non-pure functions

1 Store. Multiple Reducers.

All Reducers Are Called on Each Dispatch



Reducer = “Slice” of State



“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.”

Redux FAQ

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

Next up: Connecting React to Redux