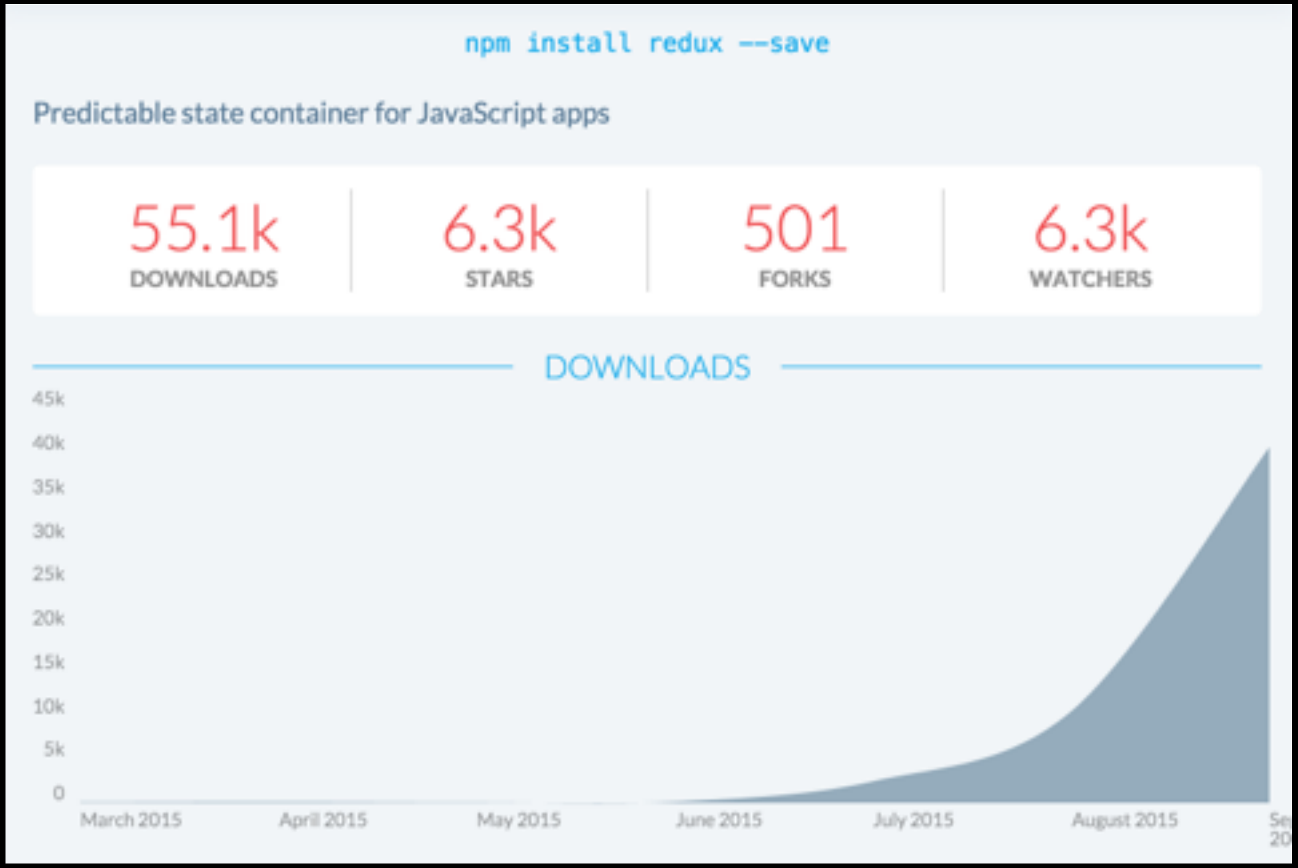
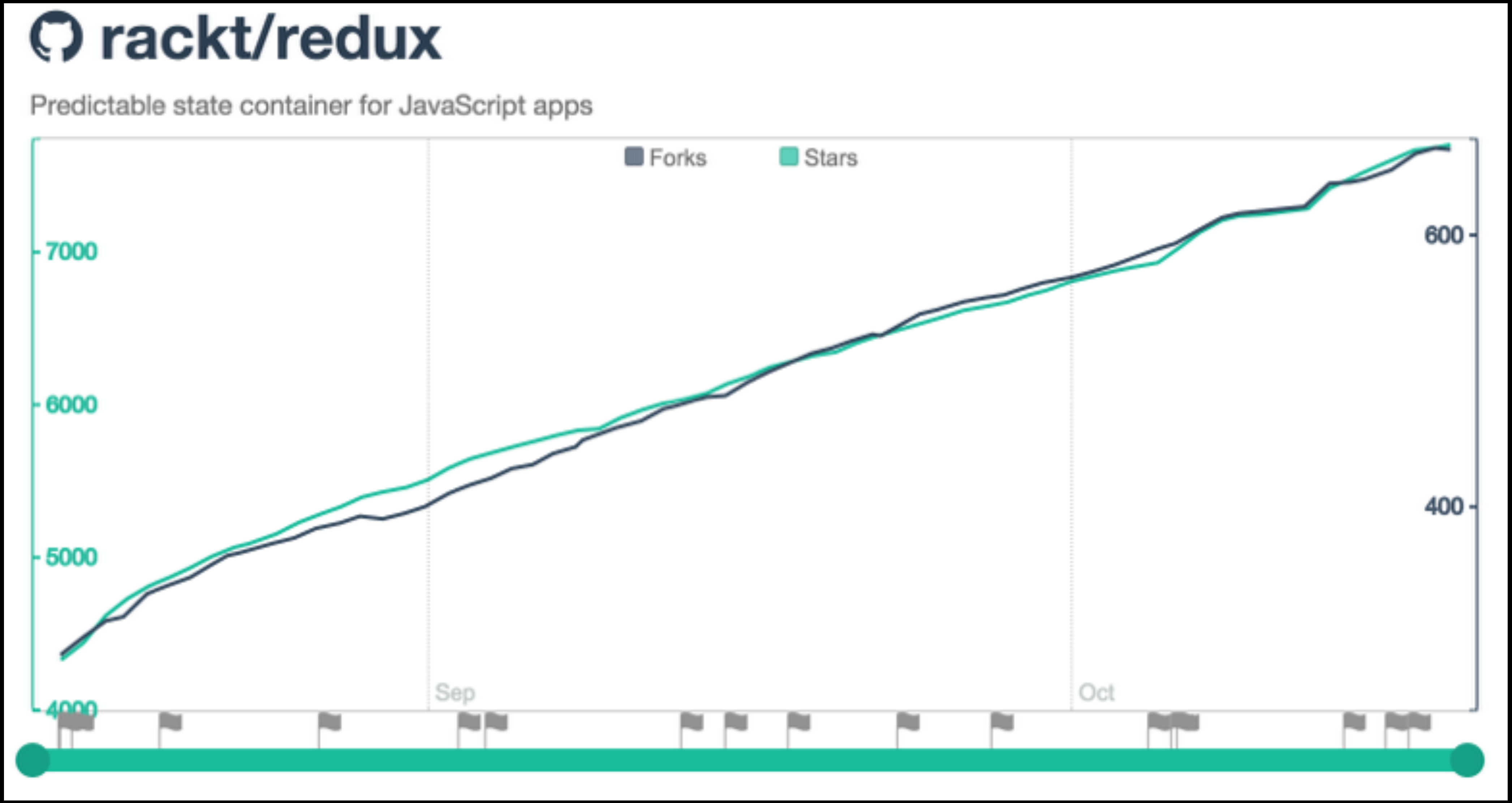
Redux

Yet another Javascript library ?

JAYWAY

Why Redux ?


npm install redux --save
Predictable state container for JavaScript apps
55.1k
DOWNLOADS
6.3k
STARS
501
FORKS
6.3k
WATCHERS
DOWNLOADS
45k
40k
35k
30k
25k
20k
15k
10k
5k
0
March 2015
April 2015
May 2015
June 2015
July 2015
August 2015

rackt/redux
Predictable state container for JavaScript apps
Forks
Stars
7000
6000
5000
4000
600
400
Sep
Oct
JAYWAY

What ?

It helps you write applications that **behave consistently**, run in different environments (client, server, and native), and are **easy to test**. On top of that, it provides a **great developer experience**, such as live code editing combined with a time traveling debugger.

The Six Stages of Debugging

1. That can’t happen
2. That doesn’t happen on my machine
3. That shouldn’t happen
4. Why is that happening ?
5. Oh, I see
6. How did that ever work ?

IF Consistent Behaviour

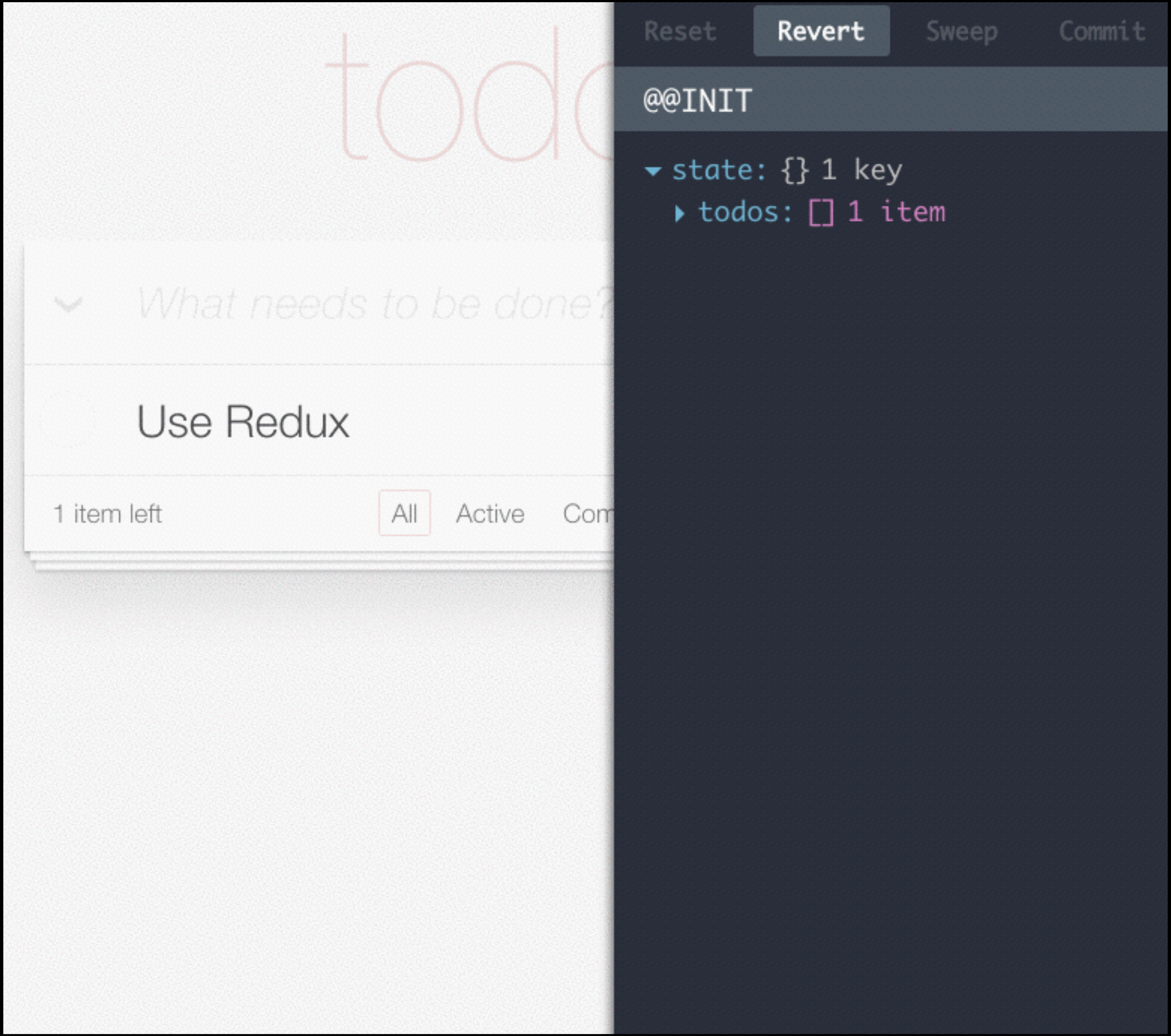
Same input always gives the same output

Always easy to recreate a state

THEN easier to understand

1. Why is that happening ?
2. Oh, I see

How
Recreate State ?



Use Redux

1 item left

```
@@INIT
▾ state: {} 1 key
  ▸ todos: [] 1 item
```

JAYWAY

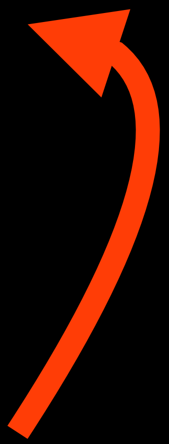
Unidirectional Data Flow

Data enters through actions

Creates a new state

View responds to state change

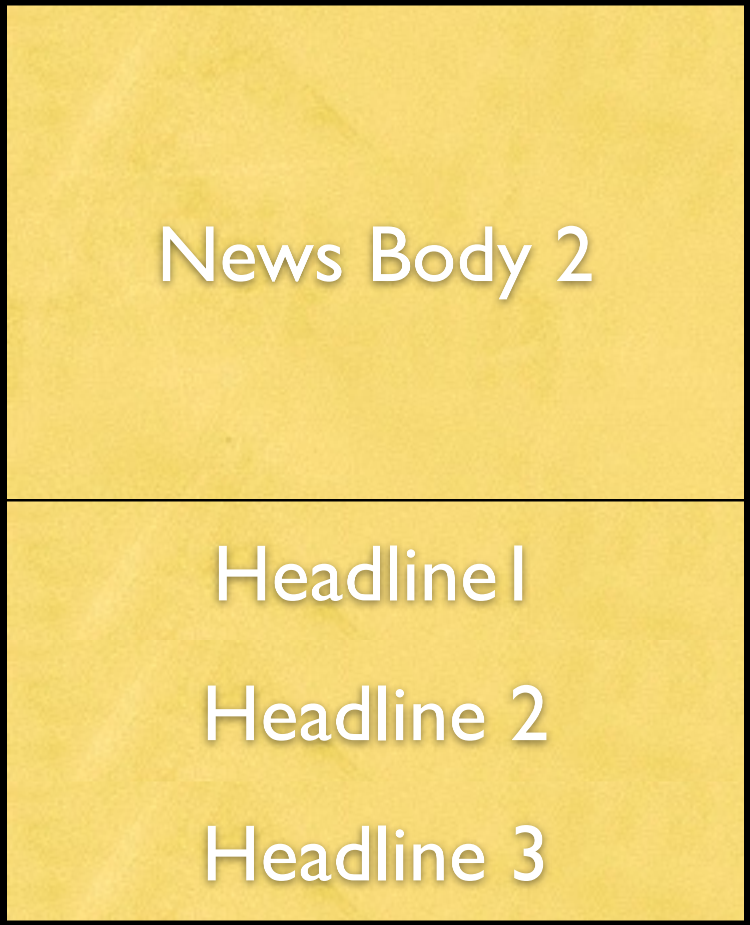
View triggers new action



That's just events updating a state. We can implement that !

Use Case

- Show a list of news headlines
- Toggle show headline or body
- Spinner while loading



JAYWAY

Example of State

```
var state = {
  headlines: {
    10: {title: 'React 0.14 released'},
    11: {title: 'Awesome Redux'}
  },
  bodies: {
    10: {text: 'Bla bla'}
  },
  isLoading: false
}
```

JAYWAY

Actions

- fetchNewsHeadline
- fetchNewsBody
- toggleExpand

Action Creators

```
export function requestHeadlines() {
  return {
    type: ‘REQUEST_NEWS_HEADERS',
  };
}

export function toggleExpand(id) {
  return {
    type: ‘TOGGLE_EXPAND',
    id
  };
}
```

JAYWAY

State Transitions

requestHeadlines

```
{
  headlines: {},
  isLoading: false,
  bodies: {}
}
```

requestHeadlinesSuccess

```
{
  headlines: {},
  isLoading: true,
  bodies: {}
}
```

some spinner

```
{
  headlines: {
    1: {title: 'React 0.14 released'},
    2: {title: 'Jayway using Redux'}
  },
  isLoading: false,
  bodies: {}
}
```

React 0.14 released

Jayway using Redux

JAYWAY

Action

```
{
    type: ‘REQUEST_NEWS_HEADERS',
}
```

(oldState, action) => newState

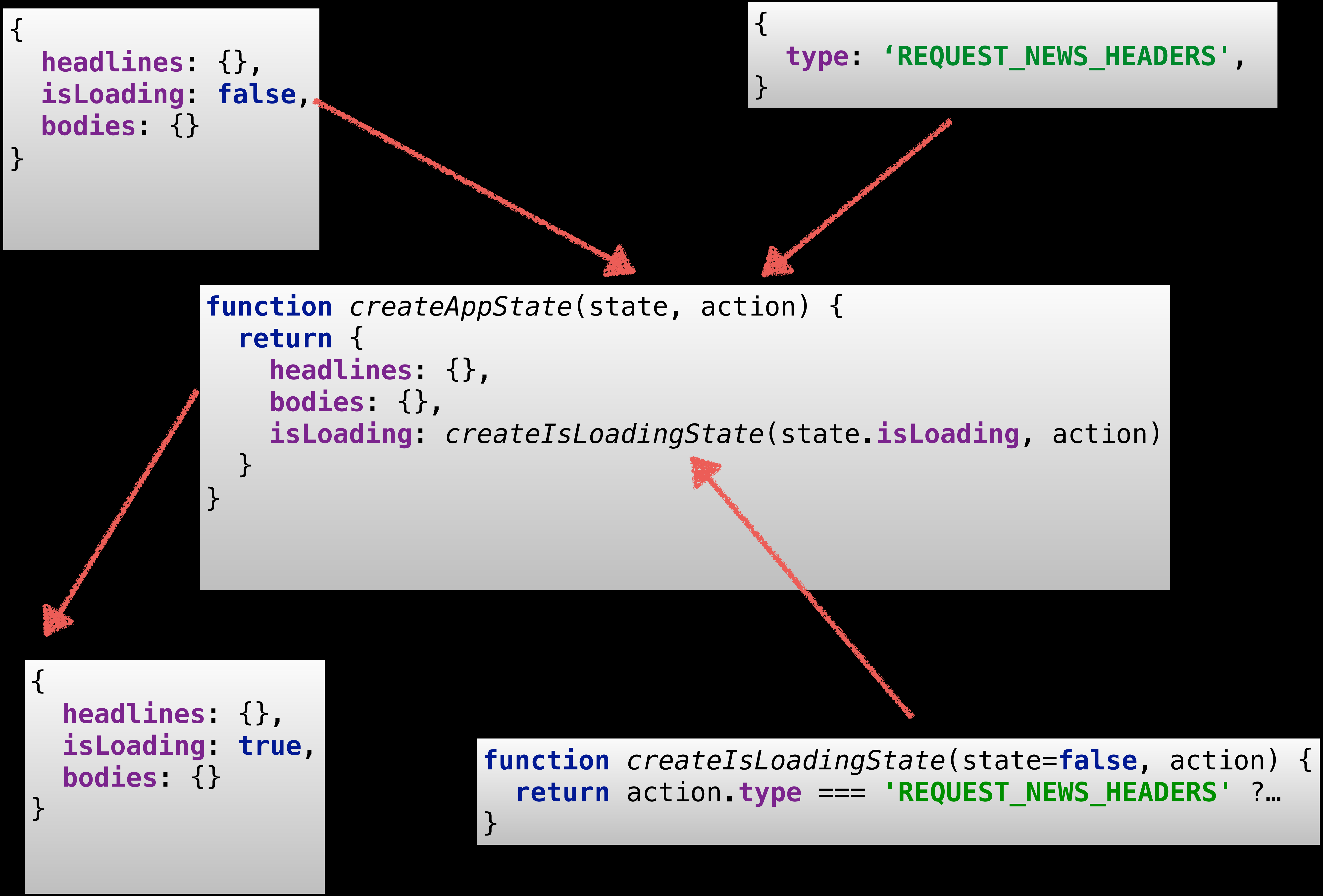
```
function createIsLoadingState(state=false, action) {
  return action.type === 'REQUEST_NEWS_HEADERS' ? true : state;
}
```

oldState

```
{
  headlines: {},
  isLoading: false,
  bodies: {}
}
```

newState

```
{
  headlines: {},
  isLoading: true,
  bodies: {}
}
```



```
{
  headlines: {},
  isLoading: false,
  bodies: {}
}
```

```
{
  type: ‘REQUEST_NEWS_HEADERS',
}
```

```
function createAppState(state, action) {
  return {
    headlines: {},
    bodies: {},
    isLoading: createIsLoadingState(state.isLoading, action)
  }
}
```

```
{
  headlines: {},
  isLoading: true,
  bodies: {}
}
```

```
function createIsLoadingState(state=false, action) {
  return action.type === 'REQUEST_NEWS_HEADERS' ?…
}
```

JAYWAY

Reducers ?

- *createIsLoadingState: (state, action) => state*
- *createAppState: (state, action) => state*

IS this not similar to reduce ?

(e.g. Lodash/Underscore reduce)

Reduce

```
import _ from 'lodash';

_.reduce([1, 2], function(total, n) {
  return total + n;
});

// We can use the reduce function on actions !
let actions = [requestHeadlines(), toggleExpand(11)];
let appState = _.reduce(actions, createAppState);
```

The *createAppState* function is a reducer !

createAppState: (state, action) => state

JAYWAY

So what ?

- **IF** reducers does not have side effects we can recreate the state !

```
function createAppState(state, action) { state.x = state.x+1
```

- App logics is a stream of events updating the app state

Redux combineReducer

```
import { combineReducers } from 'redux';

combineReducers({
    headlines: () => {},
    bodies: () => {},
    isLoading: (state, action) => action.type === 'REQUEST_NEWS_HEADERS' ? true : state
  }
);
```

JAYWAY

A new framework is born :=)

```
function createStore(initialState, reducer) {
  let state = initialState;
  return({
    getState: () => state,
    dispatch: (action) => state = reducer(state, action)
  });
}
```

```
const store = createStore(originalState, combineReducers);
store.dispatch({type: 'REQUEST_NEWS_HEADERS'});
expect(store.getState()).toEqual(expectedState);
```

How can I extend it ?

Example: I want to log actions

First Attempt

```
function createStore(reducer) {
  let state = undefined;
  return({
    getState: () => state,
    dispatch: (action) => {
      console.log('--> before action', action);
      state = reducer(state, action);
      console.log('--> after action', action);
    }
  });
}
```

Can we dispatch a promise ?

```
{
  type: 'REQUEST_NEWS_HEADERS',
  promise: api.get('news')
};
```

```
// Stubbed in unit test
const api = {
  get: () => ({
    then(cb) {cb('some data')}
  })
};
```

Action:

```
{
  type: 'REQUEST_NEWS_HEADERS',
  promise: api.get('news')
};
```

createStore:

```
function createStore(reducer) {
  let state = undefined;
  return({
    getState: () => state,
dispatch: function dispatch(action) {
  console.log('--> before action', action);
  state = reducer(state, action);

  if (action.promise && action.promise.then) {
    const success = (response) => {
      dispatch({type: action.type + '_SUCCESS', response})
    };
    const failure = () => {}; // TODO
    action.promise.then(success, failure)
  }

  console.log('--> after action', action);
}
```

The Unit Test

```
it('can resolve a promise', () => {
  const store = createStore({}, combineReducers);
  store.dispatch({type: 'REQUEST_NEWS_HEADERS', promise: api.get('news')});
  const expectedState = { bodies: {}, headlines: 'some data', isLoading: true };
  expect(store.getState()).toEqual(expectedState);
});
```

```
// Stubbed in unit test
const api = {
  get: () => ({
    then(cb) {cb('some data')}
  })
};
```

Middleware

Replace the dispatch function like this:

- Do something before
- Call original function
- Do something after

Replace it many times, e.g. logger, promise handling.

This is what a middleware does !

Apply Middleware

```
const store = createStore(originalState, combineReducers);
const storeWithLogger = applyMiddleware(store, [logger]);
```

```
const logger = store => next => action => {
  console.log('dispatching', action);
  let result = next(action);
  console.log('next state', store.getState());
  return result;
};
```

What ?

```
const logger = store => next => action => {…
```

Same as:

```
var logger = function logger(store) {
  return function (next) {
    return function (action) {
      console.log('dispatching', action);
      var result = next(action);
      console.log('next state', store.getState());
      return result;
    };
  };
};
```

JAYWAY

```
function applyMiddleware(store, middlewares) {
  middlewares = middlewares.slice();
  middlewares.reverse();
  let dispatch = store.dispatch;
  middlewares.forEach(middleware => dispatch = middleware(store)(dispatch));
  return Object.assign({}, store, {dispatch});
}
```

```
var logger = function logger(store) {
  return function (next) {
    return function (action) {
      console.log('dispatching', action);
      var result = next(action);
      console.log('next state', store.getState());
      return result;
    };
  };
};
```

Promise Middelware

```
const promiseMiddleware = store => next => action => {
  const { types, meta } = action;
  const { promise, data } = action.payload;
  const [ PENDING, FULFILLED, REJECTED ] = types;
  if (!promise) return next(action)
  /**
   * Dispatch the first async handler. This tells the
   * reducer that an async action has been dispatched.
   */
  next({
    type: PENDING,
    payload: data,
    meta
  });

  /**
   * Return either the fulfilled action object or the rejected
   * action object.
   */
  return promise.then(
    payload => next({
      type: FULFILLED,
      payload,
      meta
    }), // handle REJECT
```

```
function myAsyncActionCreator(data) {
  return {
    types: [
      ‘NEWS_ACTION_PENDING',
      ‘NEWS_ACTION_FULFILLED',
      ’NEWS_ACTION_REJECTED’
    ],
    payload: {
      promise: api.get(‘news'),
      data: data
    }
  };
}
```

Redux Thunk

What

Dispatch multiple (delayed) actions from a single action

How

Dispatch a function receiving dispatch/getStore functions

```
export default function fetchHeaders() {
  return (dispatch, getStore) => {
    dispatch(requestHeaders());
    return api.get('/news/headers').then(
        ({ data }) => dispatch(requestHeadersSuccess(data)),
        ({ data }) => dispatch(requestHeadersFailure(data)));
  };
}
```

React - Redux

- Need to
  - map React Props to Redux state
  - dispatch Redux actions from React

Mapping State to React Props ?

```
function connect(Component, store) {
  class Wrapper extends React.Component {
    constructor() {
      store.subscribe(this.handleChange.bind(this));
    }
    handleChange() {
      this.setState({ storeState: store.getState() });
    }
    render() {
      return (<Component {...this.state.storeState} />);
    }
  }
  return Wrapper;
}
```

state to props

Subscribe to Redux Store

Merge state props to component props

```
function mapStateToProps(state) {
  // return {fetching: state.isFetching }
  return state; // expose the whole state object here
}

// connects App component to Redux store.
// App component is not modified.
// Instead new component that should be used is returned.
connect(mapStateToProps)(App);
```

Advanced Connect

Redux State → React Props →

```
export function mapStateToProps({news: {expanded, details, overviews}}, {newsId}) {
  const isExpanded = expanded[newsId];
  return {
    isExpanded,
    item: isExpanded ? details[newsId] : overviews[newsId],
  }
}

function mapDispatchToProps(dispatch, {newsId}) {
  return {
    toggleExpand: () => dispatch(newsItemToggleExpand(newsId)),
    fetchNewsDetails: () => dispatch(fetchNewsDetails(newsId)),
  };
}

export default connect(mapStateToProps, mapDispatchToProps)(NewsItemContainer);
```

JAYWAY

Best Practices

- Avoid a deeply nested state object
- Use id and refs in state object

```
headlines: {
    1: {title: 'React 0.14 released'},
    2: {title: 'Jayway using Redux'}
```

- Combine many reducers in a nested way
- Reducers should work on small state objects

```
function createIsLoadingState(state=false, action) {
  return action.type === 'REQUEST_NEWS_HEADERS' ? return true :…
}
```

Best Practices

- Use an immutable library (e.g. seamless-immutable)
- Use React State if it’s simpler
- Keep state in URL

Cons

- Very young
- One state obj/multiple reducers => lot’s of mapping
- What’s the future of FLUX frameworks ?
- Very flexible framework, how do I design ?
- How do I work with side effects ?
- Hard coordinating data from many sources (consider rxjs/baconjs with or without redux)

Should I use Redux ?

Do you have the problem: “Why is that happening ?“

Do you need a FLUX framework ?

Too early ? First commit was in May

JAYWAY

Pros

- Easier to understand
- Easier to test
- Rewind/replay of actions
- Flux: Stateless Components, Unidirectional Data Flow
- Middleware
- Great for isomorphic apps
- Fantastic docs and tutorials !

Future ?

- Redux is currently the best flux framework
- Not really follow the flux architecture pattern
- The last flux framework before moving on to rxjs, cyclejs, baconjs … ?

JAYWAY

Lab

https://github.com/andreasronge/react-webpack-babel