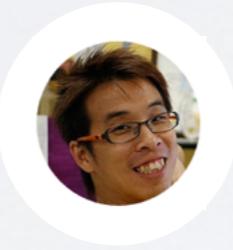


Let's Redux

for people who use React but haven't tried Redux



@josephj6802 

React.js is Awesome

Building a complicate UI was never so easy

Dashboard > Plugins

Supercharge your marketing toolkit!

Plugins extend and expand the functionality of Stackla. Supercharge your marketing cloud by integrating your existing marketing tools.

WordPress
By Stackla
Integrate social media content directly into your WordPress websites.
[Read more >](#)

Plugins

Stackla Advertising Analytics Automation CMS CRM eCommerce Email Social [Show all](#)

Advertising AdRoll
AdRoll Retargeting with UGC reaches and engages customers intelligently across devices on Facebook, Twitter and millions of websites.
[Read more >](#)

CRM, Automation Agile CRM
Create Agile CRM contacts from claimed Stackla tiles.
[Read more >](#)

Advertising Brand Networks
All your social advertising and marketing in one place combined with the authenticity of user-generated content.
[Read more >](#)

Email Automation Campaign Monitor
Create Campaign Monitor Subscriber from Stackla Claimed Tile.
[Read more >](#)

CMS ChyronHego
ChyronHego's easy-to-use social media editor Shout powered by content direct from Stackla.
[Read more >](#)

Competitions
Engage, reward and delight your customers with true multi-network social competitions.
[Create a Term](#)

Content Feed
Integrate social content into any platform that accepts RSS or Atom feeds.
[Read more >](#)

Drupal CMS Drupal
Integrate social media content directly into your Drupal websites.
[Read more >](#)

Stackla GoConnect
GoConnect is a widget that allows your customers to submit content directly to your stack.
[✓ Installed](#)

Dashboard > Access Control

api Admin Moderator Developer Support Account Admin

datatemplates Admin Moderator Developer Support Account Admin

GET Admin Moderator Developer Support Account Admin
POST Admin Moderator Developer Support Account Admin

{templateid} Admin Moderator Developer Support Account Admin

PUT Admin Moderator Developer Support Account Admin
DELETE Admin Moderator Developer Support Account Admin

extusers Admin Moderator Developer Support Account Admin

filters Admin Moderator Developer Support Account Admin

moderationviews Admin Moderator Developer Support Account Admin

tags Admin Moderator Developer Support Account Admin

terms Admin Moderator Developer Support Account Admin

tiles Admin Moderator Developer Support Account Admin

[Save](#) [Manage Groups](#)

Supercharge your marketing toolkit!

Plugins extend and expand the functionality of Stackla. Supercharge your marketing cloud by integrating your existing marketing tools.

Advertising AdRoll
Social Hootsuite
CMS WordPress
bn. Advertising Brand Networks
CMS Sitecore
[+ 24 more](#)
[View all plugins](#)

Aggregate

Active Terms 42 / 60

Content Ingested
March 2.4K
April 430
Last 30 Days 2.1K

Dashboard

Curate

Filters 58 / 20

My Content
Published 4.2K
Queued 146K
Disabled 6

Manage
Tags
[Manage Content](#)

Display

Social Hub
Our Hosted Hubs are fully hosted, socially-powered stand alone websites.

Widgets 46
Events 13

Language

Specify which languages to include for each term. This is useful if you are moderating content in different languages and you want to filter content based on language.

Do you want to enable language filtering for your term?

Language Filtering Enabled

Language Selected Chinese (Simplified), Chinese (Traditional) and English

Select Languages

Term Languages

All Languages

Arabic
Bulgarian
 Chinese (Simplified) Set as the fallback language
 Chinese (Traditional) Set as the fallback language
Croatian
Czech

Dashboard > Display: Widgets > #sydneylocal Waterfall Gen 2.0

SETTINGS **APPEARANCE** **INLINE TILE** **EXPANDED TILE** **CUSTOM CODE EDITOR** **EMBED CODE**

Inline Tile **Expanded Tile**

Widget Composer

CSS **JavaScript**

```
.tile {  
    /* Content */  
    &-content {  
        /* ... */  
    }  
    /* Image */  
    &-image-wrapper {  
        /* ... */  
    }  
    /* Video */  
    &-video-wrapper {  
        /* ... */  
    }  
    /* Avatar */  
    &-avatar-wrapper {  
        /* ... */  
    }  
}  
/*  
 * Invoked before tiles are rendered.  
 * Use it when you want to add, remove, or  
 * update tiles before rendering.  
 */  
onBeforeRenderTiles: function (addedIds, addedData) {  
    /* #3 Invoked before each tile is appended to  
     * container.  
     * Use it when you need to modify the tile H  
     * TM structure according to its data.  
     */  
    onCompleteJsonToHtml: function ($tile, tileData) {  
        console.log('onCompleteJsonToHtml', $tile, tileData);  
        return $tile;  
    }  
}
```

Preview


Celebrate with us! @christmasdownunder 3 Dec 15


James Dvorak @jaym3s2 3 Dec 15


Stephanie! 🎅🎄 27 Nov 15

Full frame shot of the Martin place Christmas decorations 🎅. And just as impressive is this shot by @jaym3s2 !! #Repost :

[Save](#) [Cancel](#)

React.js is Awesome

Building a complex UI was never so easy

Advantages

- Easy to learn

• Manage State instead of DOM
One-way Data Flow Model

UI = fn(State)

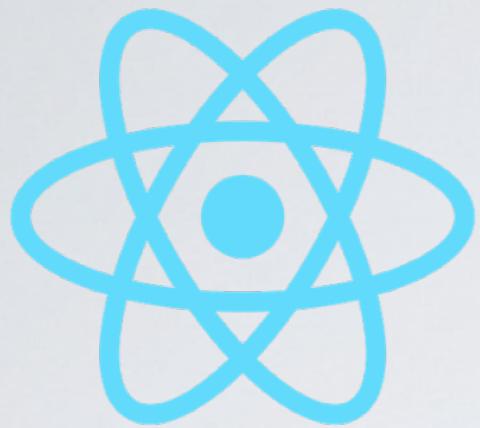
Absolutely a correct technical decision

The screenshot shows the Stackla Plugins page. It features a header "Supercharge your marketing toolkit!" and a sub-header "Plugins extend and expand the functionality of Stackla. Supercharge your marketing cloud by integrating your existing marketing tools." Below this, there are several cards for different plugins:

- WordPress**: By Stackla. Integrate social media content directly into your WordPress websites.
- AdRoll**: Advertising. AdRoll Retargeting with UGC reaches and engages customers intelligently across millions of websites.
- Agile CRM**: CRM, Automation. Create Agile CRM contacts from claimed Stackla tiles.
- Brand Networks**: Advertising. All your social advertising and marketing in one place combined with the authenticity of user-generated content.
- Campaign Monitor**: Email, Automation. Create Campaign Monitor Subscriber from Stackla Claimed Tile.
- ChyronHego**: CMS. ChyronHego's easy-to-use social media editor. Shoot powered by content direct from Stackla.
- Compete**: CMS. Engage, reward and delight your customers with true multi-network social competitions.
- Content Feed**: CMS. Integrate social content into any platform that accepts RSS or Atom feeds.
- Drupal**: CMS. GoConnect is a widget that allows your customers to submit content directly to your stack.

The screenshot shows the Stackla Language Filtering settings page. It has a header "Do you want to enable language filtering for your term?". Below it is a "Language Filtering" section with a checkbox "Enabled" and a "Select Languages" dropdown. A note says: "Specify which languages to include for each term. This is useful if you are moderating content in different languages and you want to filter content based on language." A "All Languages" section lists various languages with checkboxes. Two checkboxes are checked: "Chinese (Simplified)" and "Chinese (Traditional)". There are also radio buttons for "Set as the fallback language". At the bottom are "Save" and "Cancel" buttons.

The screenshot shows the Stackla Display Widgets page. It has a header "Dashboard > Display Widgets > #sydneylocal Waterfall". Below it is a grid of tiles: SETTINGS, APPEARANCE, INLINE TILE, EXPANDED TILE, CUSTOM CODE EDITOR, and EMBED CODE. Under "CUSTOM CODE EDITOR", there is a code editor with CSS and JavaScript tabs. The CSS tab contains code for a "tile" class. The "EXPANDED TILE" tile is expanded, showing a preview of three social media posts. One post from "Stephanie" at "stephtee" shows a photo of a Christmas tree in Sydney CBD. Another post from "James Dvorak" at "@jaym3s2" shows a photo of a city street with Christmas decorations. A third post from "Stephanie" at "stephtee" shows a photo of the Sydney CBD skyline at night.



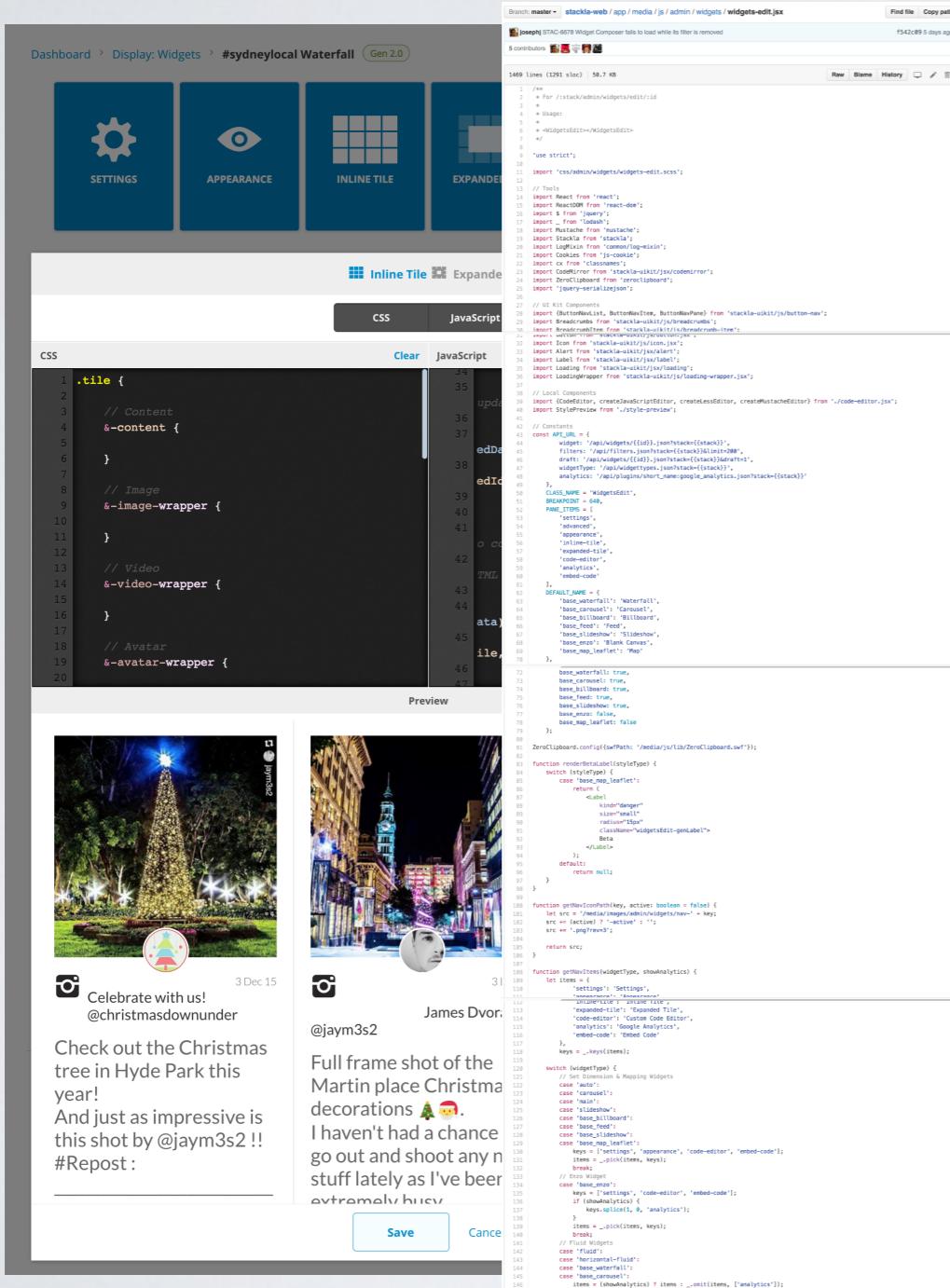
"It worked well with simple components, but **as our interfaces become more complicate** we soon found..."

PROBLEMS!!

~~React doesn't~~ We don't have a
front-end application architecture.

Code Smells

<WidgetsEdit/> Top-level Component



- **Delegation is difficult**
 - Top-down props
- **Mixed concerns**
 - View logic
 - Data fetching
 - Data decoration
- **Poor state management**
 - Component-specific
 - Mutable
 - Not maintainable

1469 lines in total!!

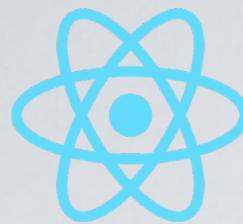
React.js = “Good” Parent

who **takes care of everything** for childs

```
class WidgetsEdit extends Component {  
  constructor(props) {  
    super(props);  
  }  
  componentWillMount() {  
    $.ajax(...).then((...) => {  
      // ... decorate data ...  
      this.setState(...)  
    });  
  }  
  render() {  
    let manyStates = that.state;  
    // ... view logic ...  
    return (  
      <ButtonNavList {...manyStates}>  
        <ButtonNavItem {...manyStates}>/>  
      </ButtonNavList>  
      <StylePreview {...manyStates}>/>  
      <CodeEditor {...manyStates}>/>  
    )  
  }  
}
```



Preparing data, passing props, handling events for childs



Handling Children Events

```
<CodeEditor
  data={that.state.data}
  stack={that.state.stack}
  count={that.state.count}
  scaffolds={that.state.scaffolds}
  draftId={that.state.draftId}
  editorGroups={that.state.editorGroups}
  onTemplateCreate={that.handleTemplateCreate}
  onSave={that.handleSave.bind(that, true)}
  onCancel={that.handleNavDismiss}
  onClone={that.handleEditorClone}
  onDelete={that.handleEditorDelete}
  onChange={that.handleEditorChange}
  onInfoChange={that.handleEditorInfoChange}>
  {loadingNode}
</CodeEditor>
```

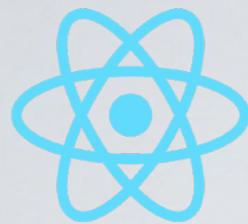
```
1105  ... // Event Handlers
1106  ...
1107  > ... handleActionsDismiss () ...
1110  ... },
1111  > ... handleBeforeUnload () ...
1112  ... },
1113  > ... handleEnableChange (e) ...
1116  ... },
1117  > ... handleEditorSave (e) ...
1118  ... },
1119  > ... handleDocumentKeydown (e) ...
1120  ... },
1121  > ... handleDiscard (e) ...
1122  ... },
1123  > ... handleNameChange (e) ...
1124  ... },
1125  > ... handleSave (real: boolean = true, e) ...
1126  ... },
1127  > ... handleFilterChange (e) ...
1128  ... },
1129  > ... handleNavClick (name: string, e) ...
1130  ... },
1131  > ... handleTemplateCreate (newKey) ...
1132  ... },
1133  > ... handleNavDismiss () ...
1134  ... },
1135  > ... handlePreviewChange (mode) ...
1136  ... },
1137  > ... handleEditorChange (changes) ...
1138  ... },
1139  > ... handleEditorInfoChange (storeKey, groupKey, data) ...
1140  ... },
1141  > ... handleEditorClone (storeKey, groupKey) ...
1142  ... },
1143  > ... handleEditorDelete (storeKey, groupKey) ...
1144  ... },
1145  > ... handleFieldChange (e) ...
1146  ... },
1147  > ... handleAjaxError (xhr, status, msg) ...
1148  ... },
1149  > ... handleSaveSuccess (real: boolean = true, data: {} = {}, status, e) ...
1150  ... },
1151  > ... handleSavedDismiss () ...
1152  ... },
1153  > ... handleToggleAdvanced () ...
```

Child passes all events to its parent

because only the top-level component should access state

22 Event Handlers





Delegation or Not?

```
...that.log('renderNav() is executed');

...nodes = _.map(items, (item, key) => {
...  let isActive = (that.state.activeNavKey === key), ...
...
...  if (state.loading) { ... }
...
...  if (key === 'code-editor') {
...    isError = (_.size(that.state.error) > 0);
...  } else { ... }
...
...  if (key === 'code-editor' && !that.canAccessEditor) {
...    isEnabled = true;
...  }
...
...  if (key === 'analytics' && (!that.state.showAnalytics || ['base_waterfall', ...
...    yle.style] === -1)) {
...    return;
...  }
...
...  return (
...    <ButtonNavItem
...      key={key}
...      name={key}
...      disabled={isEnabled}
...      error={isError}
...      active={isActive}
...      onSelect={that.handleClick}
...      <img src={getNavIconPath(key, isActive)} width="78" height="59"/>
...      <span className="st-small">{item}</span>
...    </ButtonNavItem>
...  );
...};

...return (
...  <ButtonNavList
...    active={(that.state.activeNavKey !== null)}
...    onDismiss={that.handleNavDismiss}
...    {nodes}
...  </ButtonNavList>
...);
```

Should `<ButtonNavItem/>` appear in top-level as illustrated?

If you think **NO**, plz put inside...

- Pass all `<ButtonNavItem/>` required props to `<ButtonNavList/>`
- Handle its events in `<ButtonNavList/>`
- Handle its events in `<WidgetsEdit/>`

If you think **YES**, plz keep there...

Handle child logic in top-level component.

Both ways are not good...

Fetching Data for Childs

```
236     componentWillMount () {
237       var that = this,
238         url = {},
239         attrs = {},
240         globalVariables = Stackla.globalVariables;
241
242       that.log('componentWillMount() is executed');
243
244       that.canAccessEditor = (globalVariables.developerFeatureEnabled && globalVariables.canAccessEditor);
245
246       attrs = {
247         stack: that.props.params.stack,
248         id: that.props.params.id
249       };
250
251       url.filters = Mustache.render(API_URL.filters, attrs);
252       url.widget = Mustache.render(API_URL.widget, attrs);
253       url.draft = Mustache.render(API_URL.draft, attrs);
254       url.widgetType = Mustache.render(API_URL.widgetType, attrs);
255       url.analytics = Mustache.render(API_URL.analytics, attrs);
256
257       that.loadDeferreds = [
258         $.ajax(url.filters),
259         $.ajax(url.widget),
260         $.ajax(url.widgetType),
261         $.ajax(url.analytics),
262         $.ajax(url.draft, {type: 'PUT', data: '{}'}) // FIXME - data is really odd
263       ];
264
265       that.loadInitData();
266     },

```

No Separation of Concerns

Why Redux?

Not have children?



<https://github.com/reactjs/redux/issues/151#issuecomment-173749456>

Why Redux?

Solve all the mentioned issues

- Delegation is ~~difficult~~ easier
 - Top-down props ~~including events~~ is not required
- Mixed Separation of concerns
 - View logic
 - Data fetching goes to “Action”
 - Data decoration goes to “Reducer”
- Poor Better state management
 - Component ~~App~~-specific
 - ~~Mutable~~ Immutable
 - ~~Not~~ Maintainable

Basic Concept

3. State



1. Action

2. Reducer

1. Action

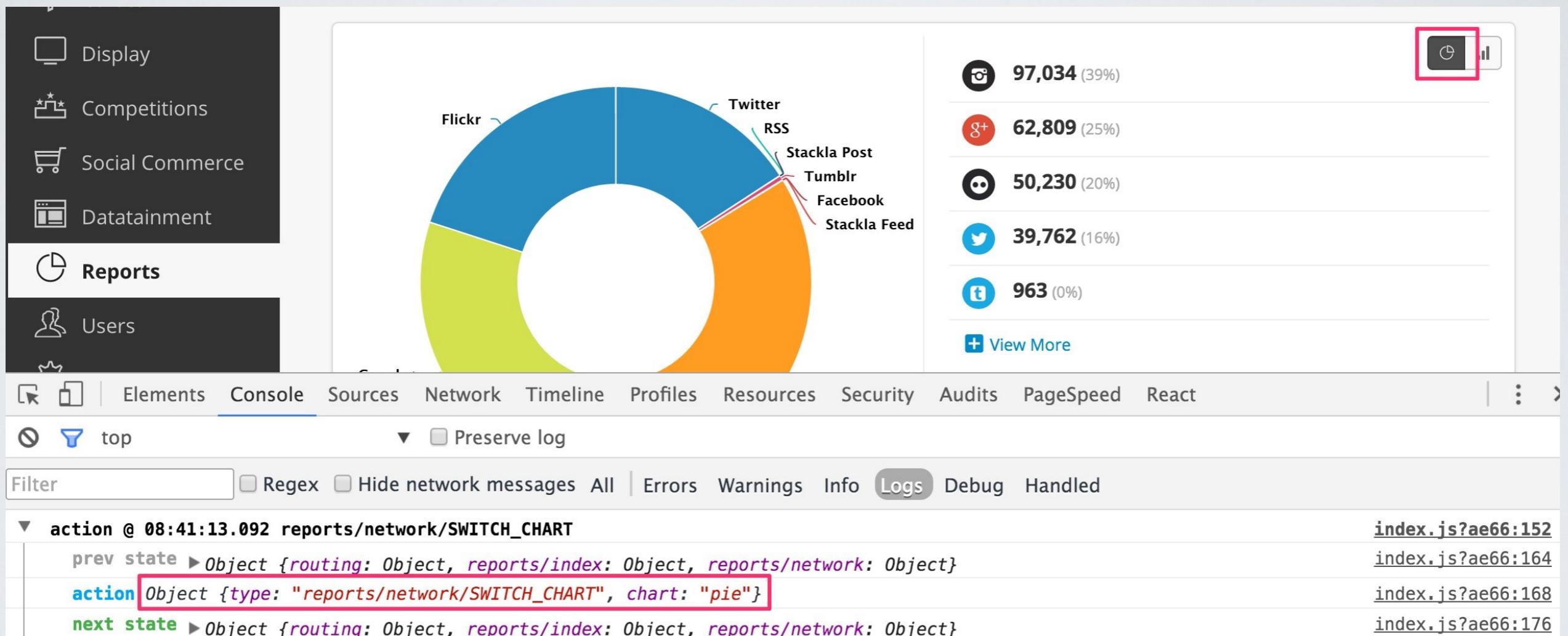
Pure Object

Minimum data which describes the change

```
{  
  type: 'SORT_PLUGINS',  
  by: 'id',  
  direction: 'desc'  
}
```

Only the “**type**” property is **required**

1. Action



The screenshot shows the Stackla developer tools interface. On the left, a sidebar menu includes: Display, Competitions, Social Commerce, Datainment, Reports (selected), and Users. Below the menu are tabs for Elements, Console (selected), Sources, Network, Timeline, Profiles, Resources, Security, Audits, PageSpeed, and React. The Network tab is active, showing a pie chart with segments for Flickr (blue), Twitter (light blue), RSS (green), Stackla Post (pink), Tumblr (red), Facebook (orange), and Stackla Feed (yellow). To the right of the chart is a list of network requests with their counts and percentages: 97,034 (39%), 62,809 (25%), 50,230 (20%), 39,762 (16%), and 963 (0%). A "View More" button is at the bottom. At the bottom of the screen, the developer console displays a log entry for an action:

```
action @ 08:41:13.092 reports/network/SWITCH_CHART
  prev state ► Object {routing: Object, reports/index: Object, reports/network: Object}
  action Object {type: "reports/network/SWITCH_CHART", chart: "pie"}
  next state ► Object {routing: Object, reports/index: Object, reports/network: Object}
```

The "Logs" tab is selected in the console. The log entry is timestamped at 08:41:13.092 and has a file path of index.js?ae66:152. It shows the previous state, the action object with type "reports/network/SWITCH_CHART" and chart "pie", and the next state. The "Logs" tab is highlighted in grey.

1-1. Action Creator



Pure **function** which **creates** an action

Make it **easier** to create an action

```
function sortPlugins(by, direction = 'desc') {  
  return {  
    type: 'SORT_PLUGINS',  
    by,  
    direction  
  };  
}
```

Reusable, Portable, and Easy to Test

(Return promise for asynchronous action)

2. Reducer



Pure **function** which returns the **next state**

reducer(previousState, action) => state

```
function reducer(state = {}, action) {
  switch (action.type) {
    case 'SORT_PLUGINS':           Initial State
      return {
        ...state,
        plugins: {
          orderBy: action.by,
          orderByDirection: action.direction,
          data: _.sortByOrder(state.data, action.by,
            action.direction)
        }
      };
    default:
      return state;                Always return current state
  };
}
```

2. Reducer



reducer(prevState, action)
=> nextState

```
▼ action @ 08:41:13.092 reports/network/SWITCH_CHART
  ▶ prev state Object {routing: Object, reports/index: Object, reports/network: Object} ⓘ
    ► reports/index: Object
    ▼ reports/network: Object
      chart: "column"
      ► data: Array[11]
      didInvalidate: false
      filters: ""
      isFetching: false
      lastUpdated: 1460673632245
      listAll: false
      total: 251028
      ► __proto__: Object
    ► routing: Object
    ► __proto__: Object
  ▶ action Object {type: "reports/network/SWITCH_CHART", chart: "pie"}
  ↓
  ▶ next state Object {routing: Object, reports/index: Object, reports/network: Object} ⓘ
    ► reports/index: Object
    ▼ reports/network: Object
      chart: "pie"
      ► data: Array[11]
      didInvalidate: false
      filters: ""
      isFetching: false
      lastUpdated: 1460673632245
      listAll: false
      total: 251028
      ► __proto__: Object
    ► routing: Object
    ► __proto__: Object
```

3. Store



A plain **object** which holds **application state**

Store is **the only one state** for the whole app

```
{  
  plugins: {  
    data: [  
      {id: 1, name: 'AdRoll'},  
      {id: 2, name: 'Agile CRM'},  
      {id: 3, name: 'Brand Networks'}  
    ],  
    orderBy: 'id',  
    orderByDirection: 'desc'  
  }  
}
```

Dispatching actions is the **only way** to **update** store

3. Store



```
▼ action @ 08:41:13.092 reports/network/SWITCH_CHART
  prev state ▼ Object {routing: Object, reports/index: Object, reports/network: Object} ⓘ
    ► reports/index: Object
    ▼ reports/network: Object
      chart: "column"
      ► data: Array[11]
      didInvalidate: false
      filters: ""
      isFetching: false
      lastUpdated: 1460673632245
      listAll: false
      total: 251028
      ► __proto__: Object
    ► routing: Object
    ► __proto__: Object
  action Object {type: "reports/network/SWITCH_CHART", chart: "pie"}
  next state ▼ Object {routing: Object, reports/index: Object, reports/network: Object} ⓘ
    ► reports/index: Object
    ▼ reports/network: Object
      chart: "pie"
      ► data: Array[11]
      didInvalidate: false
      filters: ""
      isFetching: false
      lastUpdated: 1460673632245
      listAll: false
      total: 251028
      ► __proto__: Object
    ► routing: Object
    ► __proto__: Object
```

3-1. Store Creation

with Reducer (the spec of your store)

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

```
import {createStore} from 'redux';  
  
let store = createStore(reducer);
```

The store provides several useful **API methods**

3-2. Store APIs

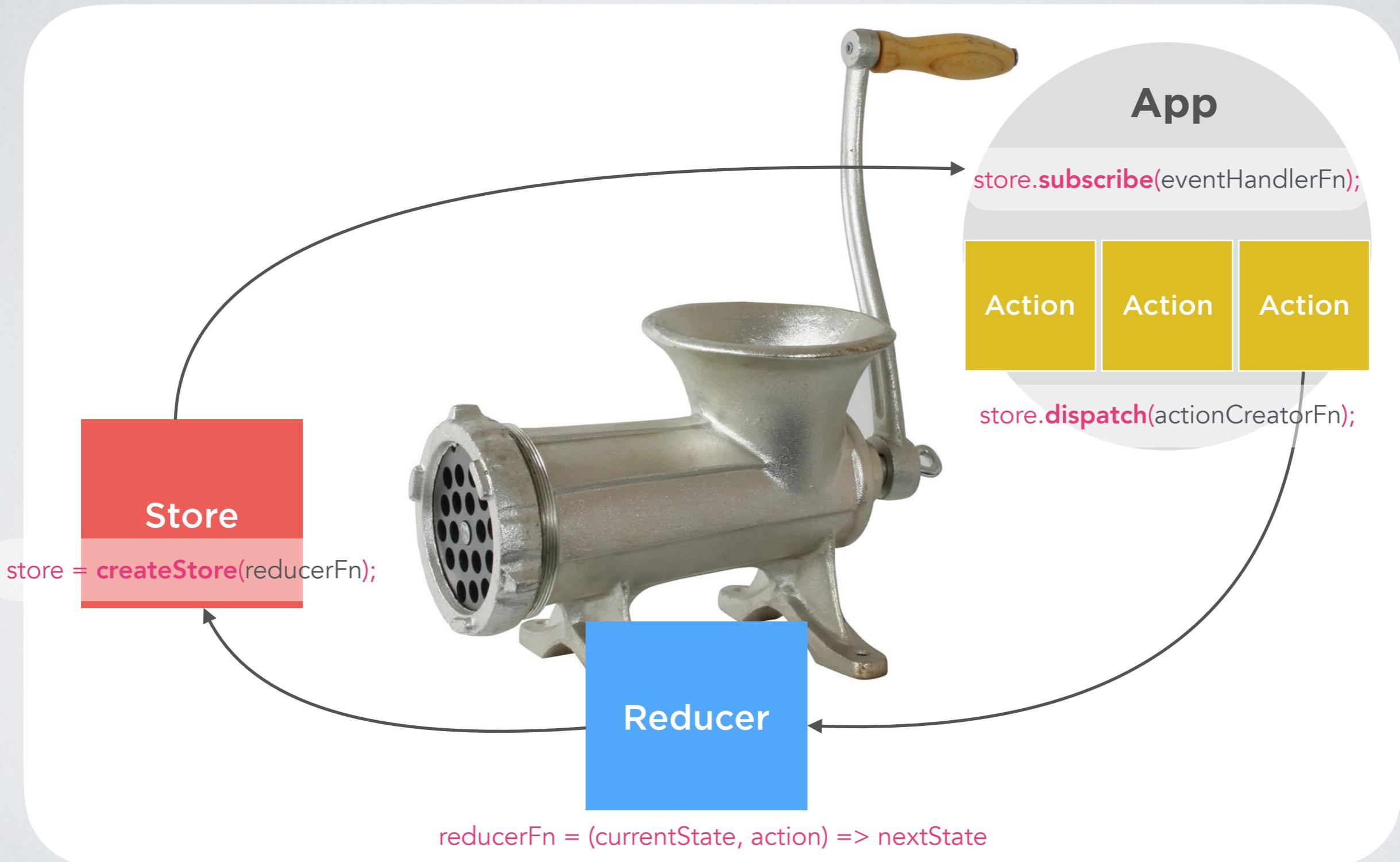
store#dispatch

```
store.dispatch(sortPlugins('name', 'desc'));
```

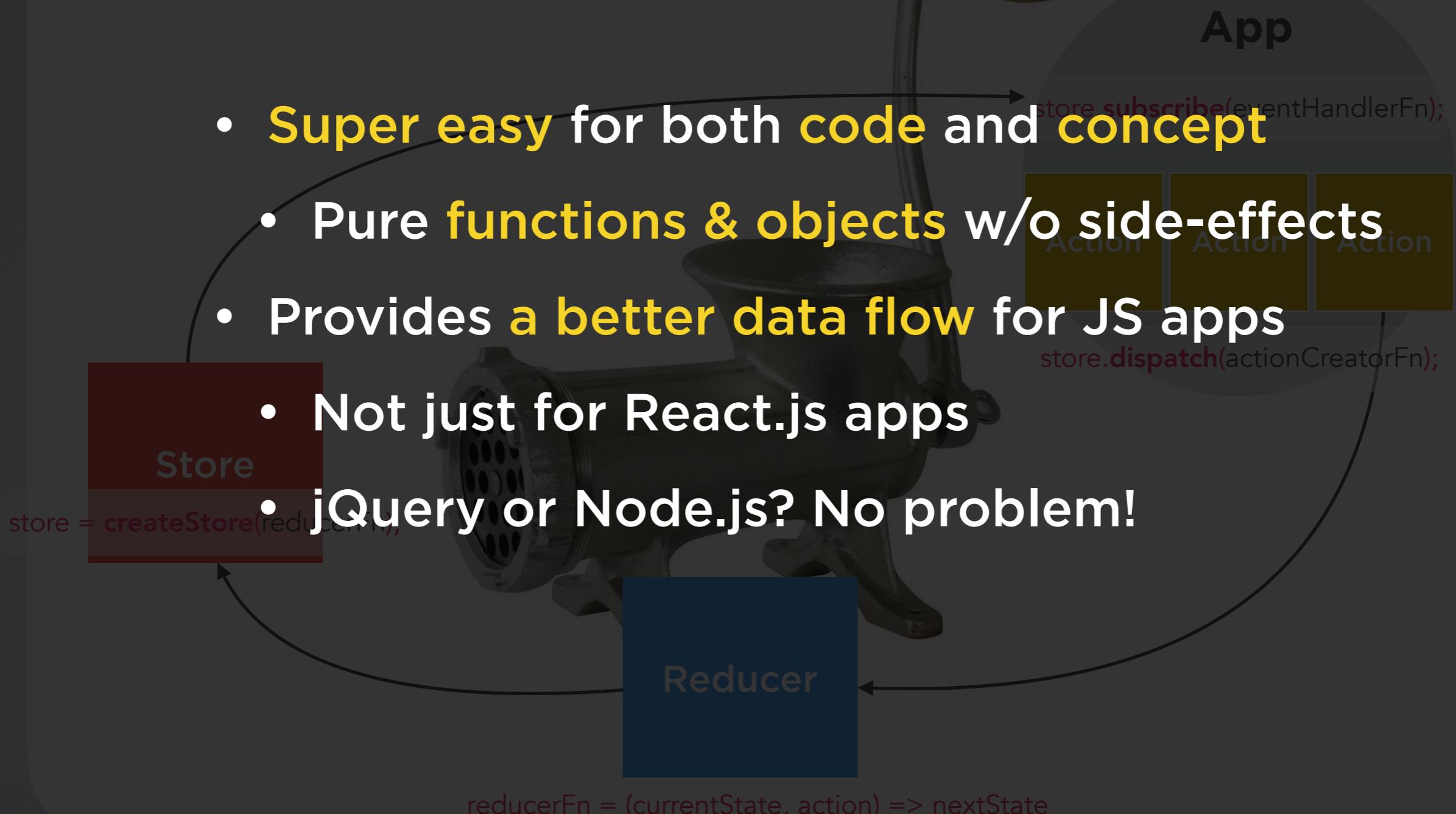
store#subscribe **store#getState**

```
let currentValue;
store.subscribe(( ) => {
  let previousValue = currentValue;
  currentValue = store.getState();
  if (previousValue === currentValue) {
    return;
  }
  // DO SOMETHING...
});
```

Redux Data Flow



Redux Data Flow That's it!



Redux and React?

First, let's check the list again

- **Delegation is easier**
 - Top-down props is not required 
- **Separation of concerns**
 - View logic 
 - Data fetching goes to “Action” 
 - Data decoration goes to “Reducer” 
- **Better state management**
 - App-specific 
 - Immutable 
 - Maintainable 

React.js = “Good” Parent

who takes care of everything of its child components

```
class WidgetsEdit extends Component {  
  constructor(props) {  
    super(props);  
  }  
  componentWillMount() {  
    $ajax(...).then((...) => {  
      // ... decorate data ...  
      this.setState(...)  
    });  
  }  
  render() {  
    let manyStates = that.state;  
    // ... decorate state ...  
    return (  
      <ButtonNavList {...manyStates}>  
        <ButtonNavItem {...manyStates}>/>  
      </ButtonNavList>  
      <StylePreview {...manyStates}>/>  
      <CodeEditor {...manyStates}>/>  
    )  
  }  
}
```



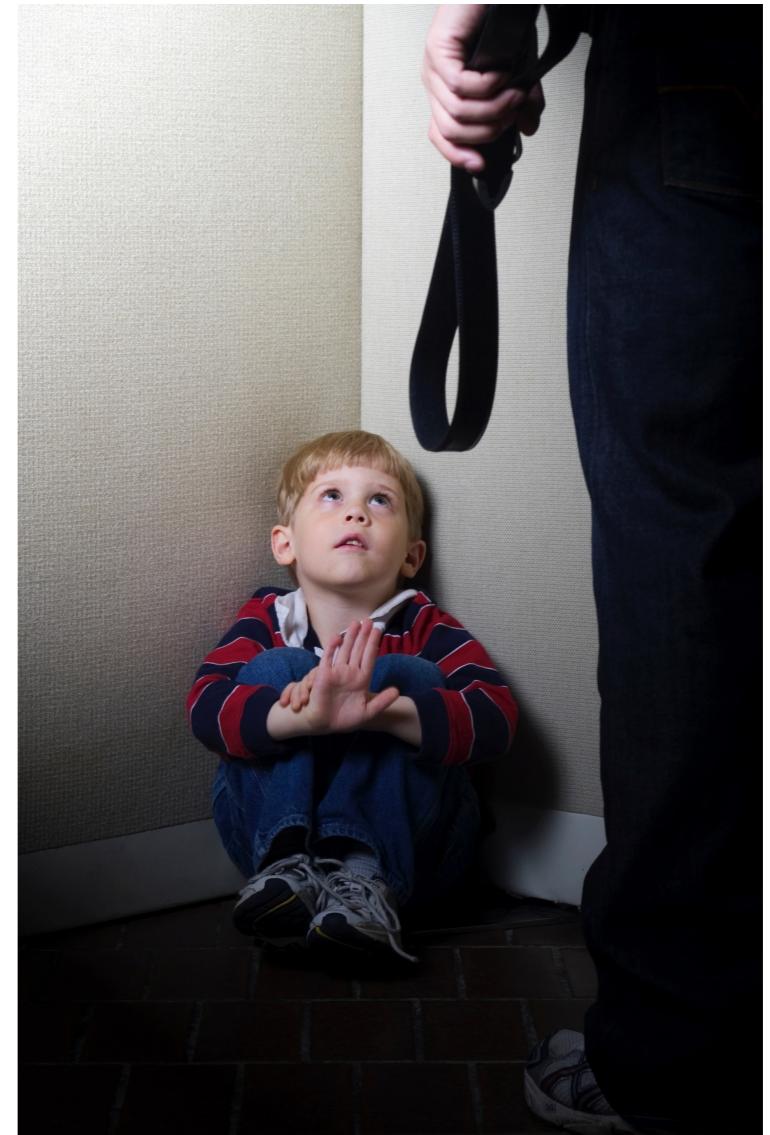
Preparing data & handling events for child components

React + Redux = Bad Parent

who only cares about himself

```
import React, {Component} from 'react';
import {connect} from 'react-redux';
import {fetchData} from './actions';

class WidgetsEdit extends Component {
  constructor(props) {
    super(props);
  }
  componentWillMount() {
    this.props.dispatch(fetchData());
  }
  render() {
    return (
      <ButtonNavList>
        <ButtonNavItem/>
      <ButtonNavList>
        <StylePreview/>
        <CodeEditor/>
      )
    }
}
WidgetsEdit = connect(mapStateToProps)(WidgetsEdit);
```



Delegation = Preparing data and dispatching **for itself**

React.js w/ Redux

Child component **connects to store by itself**

```
import React from 'react';
import {connect} from 'react-redux';
import {modeSwitch} from './actions';

let StylePreview = ({url, dispatch} = props) => {
  return (
    <a onClick={() => dispatch(modeSwitch('mobile'))}>/>
      Mobile View
    </a>
    <iframe src={url}>/>
  );
}

StylePreview = connect((state) => { // map state to props
  return {
    url: state.widget.url,
  }
})(StylePreview);
```



... And **dispatches actions by itself**

Summary

- Good start to learn **functional programming**
- **Add complexity** but also **add maintenance** = worthy
- Single Store
 - **Time Travel** (e.g. **Undo**) is possible
- No need to access React state API anymore
- Not only for React.js

Further Reading

[egghead.io](#)



<https://egghead.io/series/getting-started-with-redux>

Very good material for learning Redux!

Discussion - Store Structure

One Store - Could it possibly fit our application?

```
{  
  "app": {  
    data: {  
      filters: [],  
      terms: [],  
    },  
    isFetching: false,  
  },  
  "reports/network": {  
    data: {  
      networks: [],  
    },  
    isFetching: false,  
    didInvalidate: false,  
    lastUpdated: 15372124,  
    orderBy: 'created_at',  
    orderByDirection: 'asc',  
  },  
  "widgets/index": {  
    data: {  
      widgets: []  
    }  
    isFetching: false,  
    didInvalidate: false,  
    lastUpdated: 15372124,  
    orderBy: 'created_at',  
    orderByDirection: 'asc',  
  }  
}
```

THANK YOU!

Let's Redux!!