React!

So What's the Big Deal, Anyway?

Introduction

Who is this guy?
Is this for me?
Why don’t they ever have my favorite pizza?

Doug Johnson

Senior Developer (ooo!)

Balfour/Taylor Publishing

Is this for me?

What's the minimum I need to know to use React?

What is the point of React? What problem does it solve?

How does React look at the world?

When would I use React?

Am I going to have to watch this guy code?

React … reacts … rapidly

React is V

Not MVC

Not MVVM

Not MVP

Not MVCPVVPMC

React’s primary purpose is the View

React is now where web components is going to be...sort of

Fast Times at React High

React is used to make components in an application or page

The components react to data changes

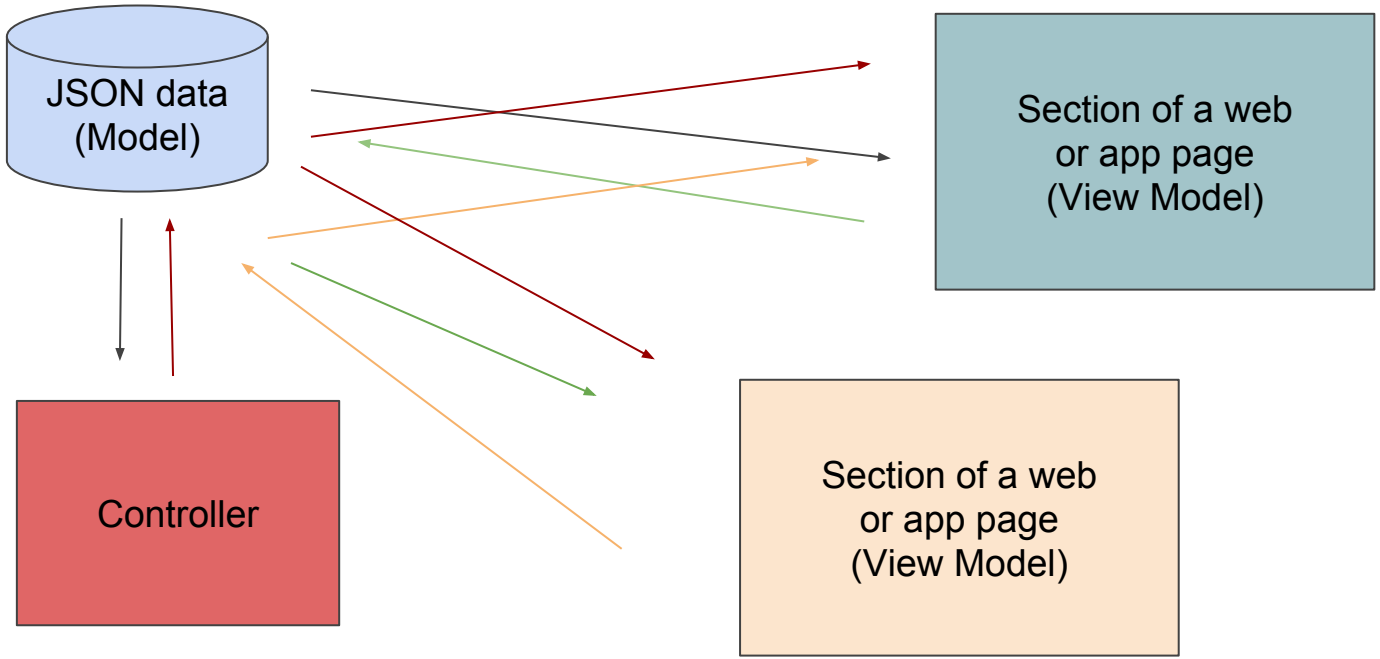
The components are not the DOM

The components create and affect the DOM

The components ONLY update the DOM that has changed

This make screen “refresh” very fast

It’s complicated...



React’s purpose is to simplify interactive complexity

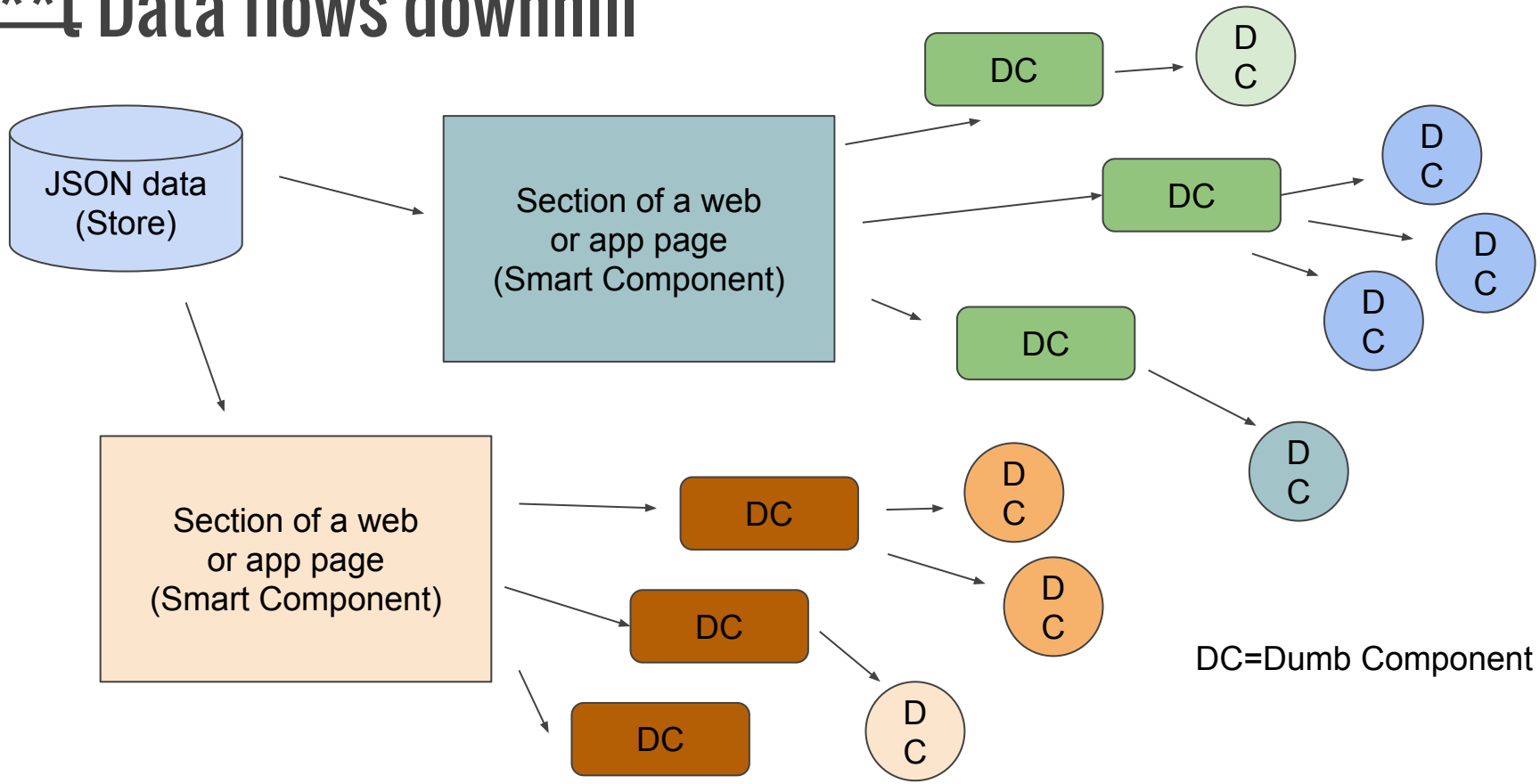
If you get nothing else … get this

Rule 1) Data, in React, flows in one direction only

Rule 2) When tempted to send data in another direction, see Rule 1

One Way or the highway!

S**t Data flows downhill



How does React enforce this?

- Data, in a component, is either
  - state
  - property (prop)
- A component can only change its own state
- A component cannot change its own props
- Neither state nor props are required
  - Not very useful without one or the other

What's the least I can do?

A React component needs to render something

```
var OrderProducts = React.createClass({
    render: function(){
        return (
            <div>
                <p>This is text from React</p>
            </div>
        )
    }
});

export default OrderProducts;
```

React Talk

You can have anything you want on the page.

This is text from React

```
<body>
<h3>React Talk</h3>
<p>You can have anything you want on the page.

<div id="react-replacer"></div>

</body>
```

Yes, I know you could do that with jQuery

Dumb and dumber

```
var OrderConfirmList = React.createClass( {
  render: function () {
     var self = this;
     var productNodes = this.props.products.map( function ( product ) {
        return (<ProductConfirm product={product} key={product.product_sku}
                          showLine={product.qty > 0}/>);
     } );
     return (
        <div className="row confirmList">
           <fieldset>
              <legend>Confirm Order</legend>
              {productNodes}
           </fieldset>
        </div>
     )
  }
} );
```

```
  },
  render: function () {
     var product = this.props.product;
     if ( this.props.showLine ) {
        return (
           <div className="row product">
              <div className="col-md-5">
                 <p className="form-control-static">{product.product_name}</p>
              </div>
              {this.Pricing()}
              {this.Quantity()}
              <div>
                 <div className="btn" onClick={this.deleteItem}>
                    <span className="glyphicon glyphicon-remove glyphicon-white"></span>
                 </div>
              </div>
           </div>

        )
     } else {
        return (<div></div>)
     }
  }
} );


export default ProductConfirm;
```

Get Smart

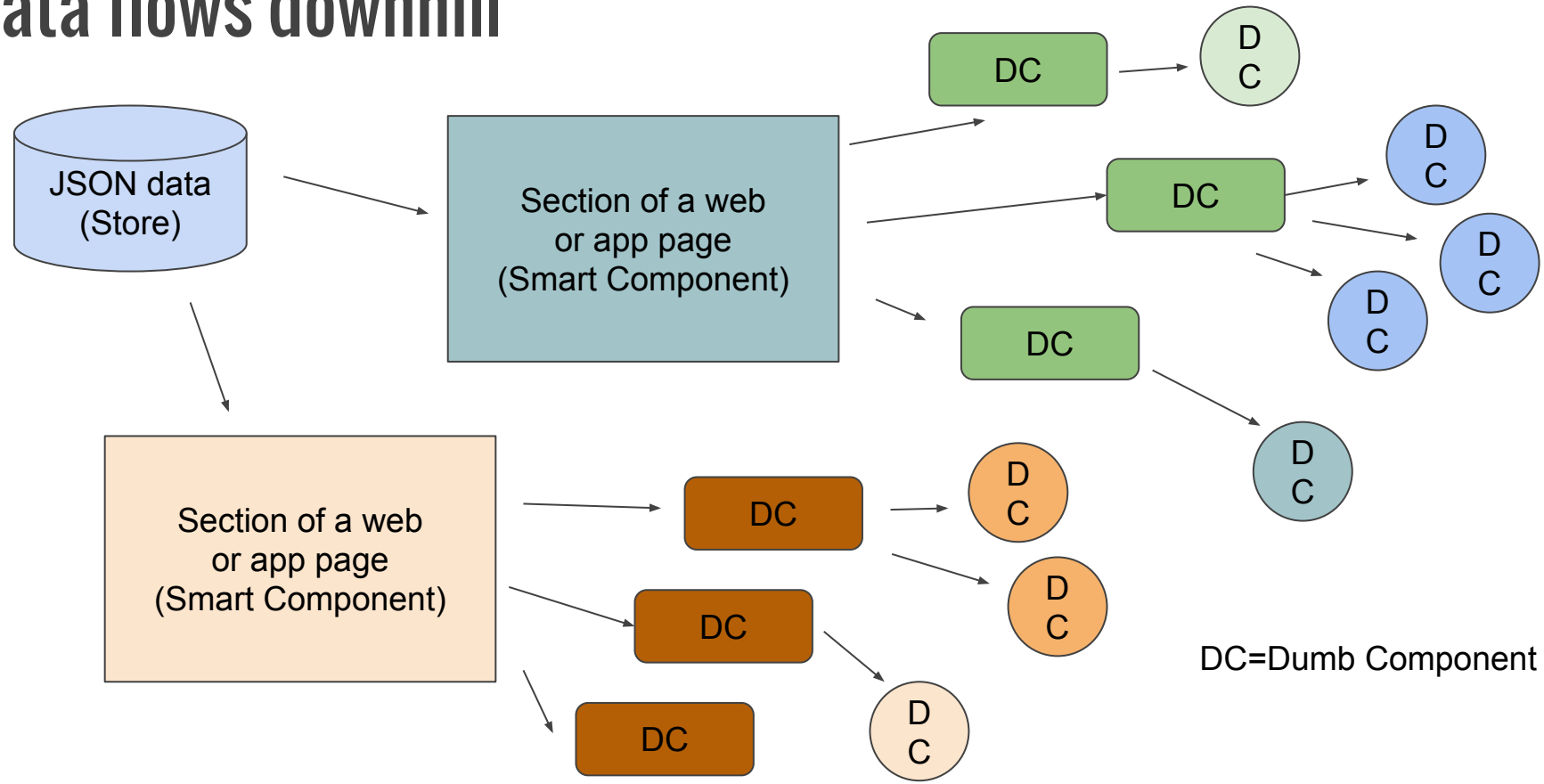
```
var OrderProducts = React.createClass( {
    getInitialState: function () {
        return {
            orderProducts: [],
            lastOrder: OrderStore.getLastOrder(),
            payment: OrderStore.getPayment(),
            sendEmailToAdviser: OrderStore.getSendEmailToAdviser()
        };
    },
```

```
    }
},
render: function () {
    return (
        <div>
            <ProductList updateOrder={this.updateOrder()} products={this.state.orderProducts}/>
            <OrderConfirmList products={this.state.orderProducts}/>
            <TotalPrice totalOrder={this.calcTotal()} balanceDue={this.balanceDue()}/>
            <Payment payment={this.state.payment} previousPayments={this.previousPayments()}/>

            <div className="col-md-5 pull-right">{this.sendEmailToAdviser()}</div>
            <div className="col-md-12">
                {this.submitContinueButton()} {this.submitButton()}
                <img id="loaderIcon" className="loader"
                     src="/wp-content/plugins/b4oncampus/images/ajax-loader.gif"/>
                {this.cancelButton()}
            </div>
            <NewOrderStatus lastOrder={this.state.lastOrder} nextOrder={this.nextOrder}/>
        </div>
    )
}
} );

export default OrderProducts;
```

Data flows downhill



But What If I Want to Change Data?

What Then?

Forms change data

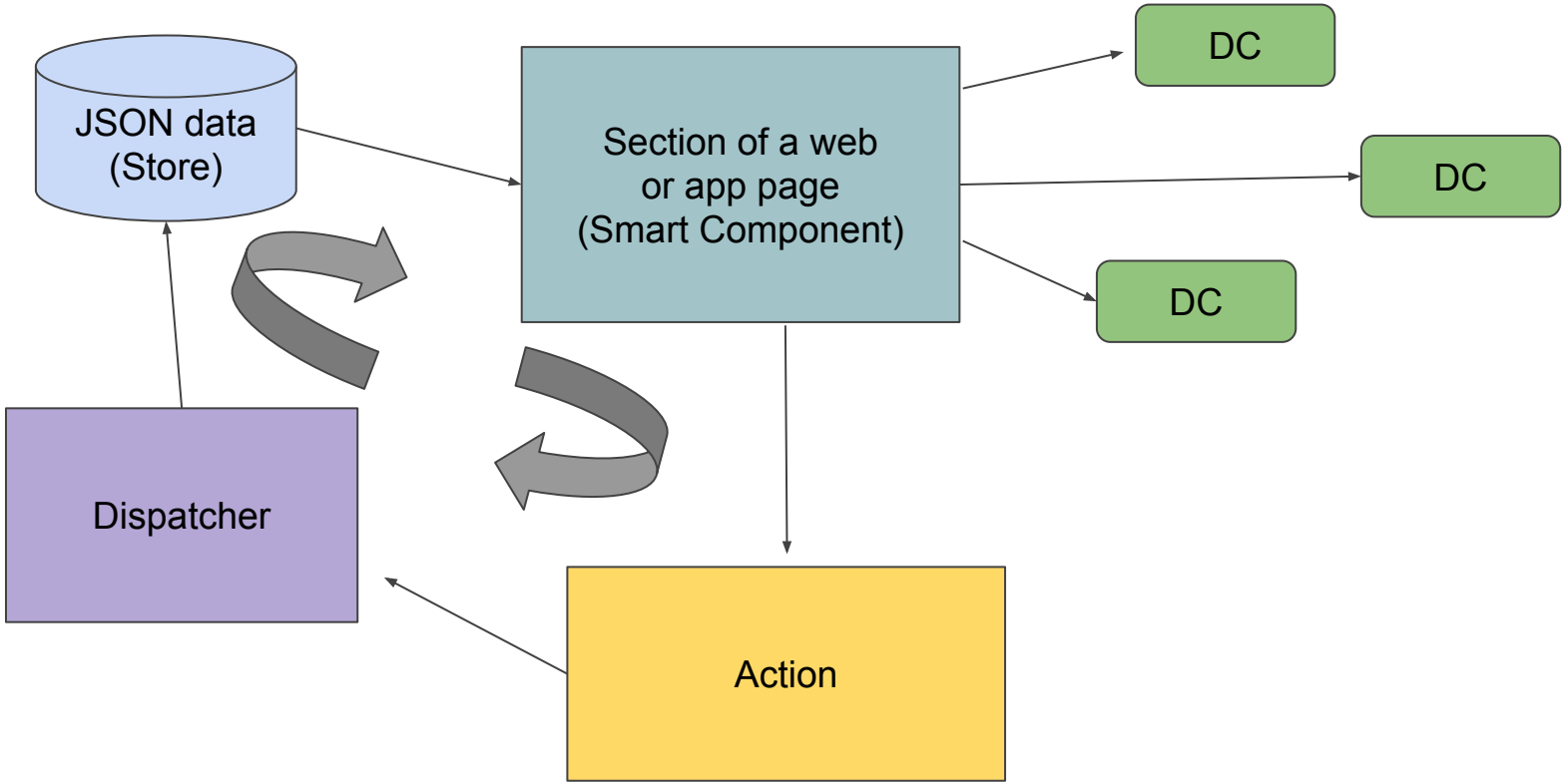
Forms are made up of components

Changing the data should update the store

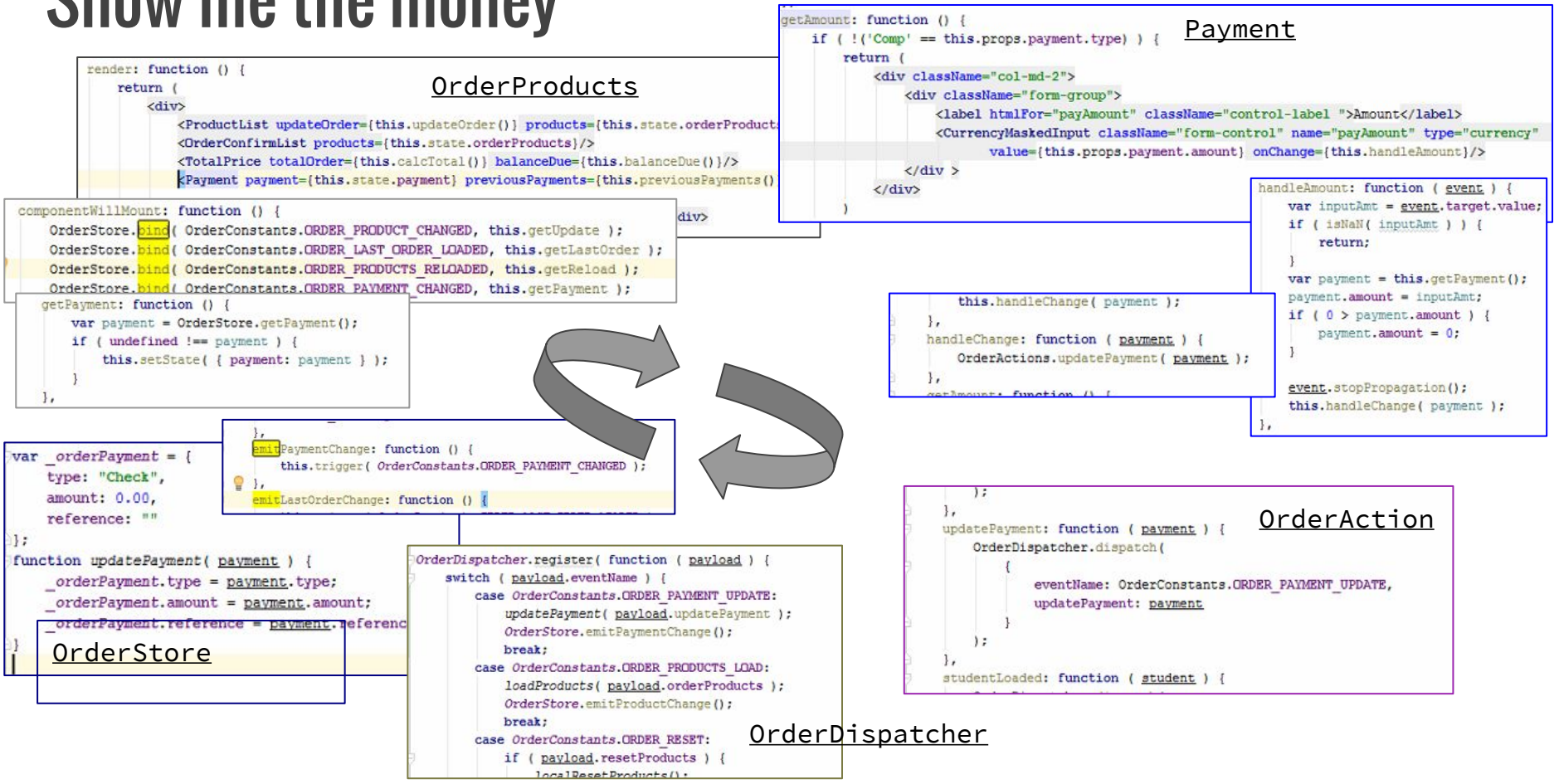
So Data is flowing uphill

Right?

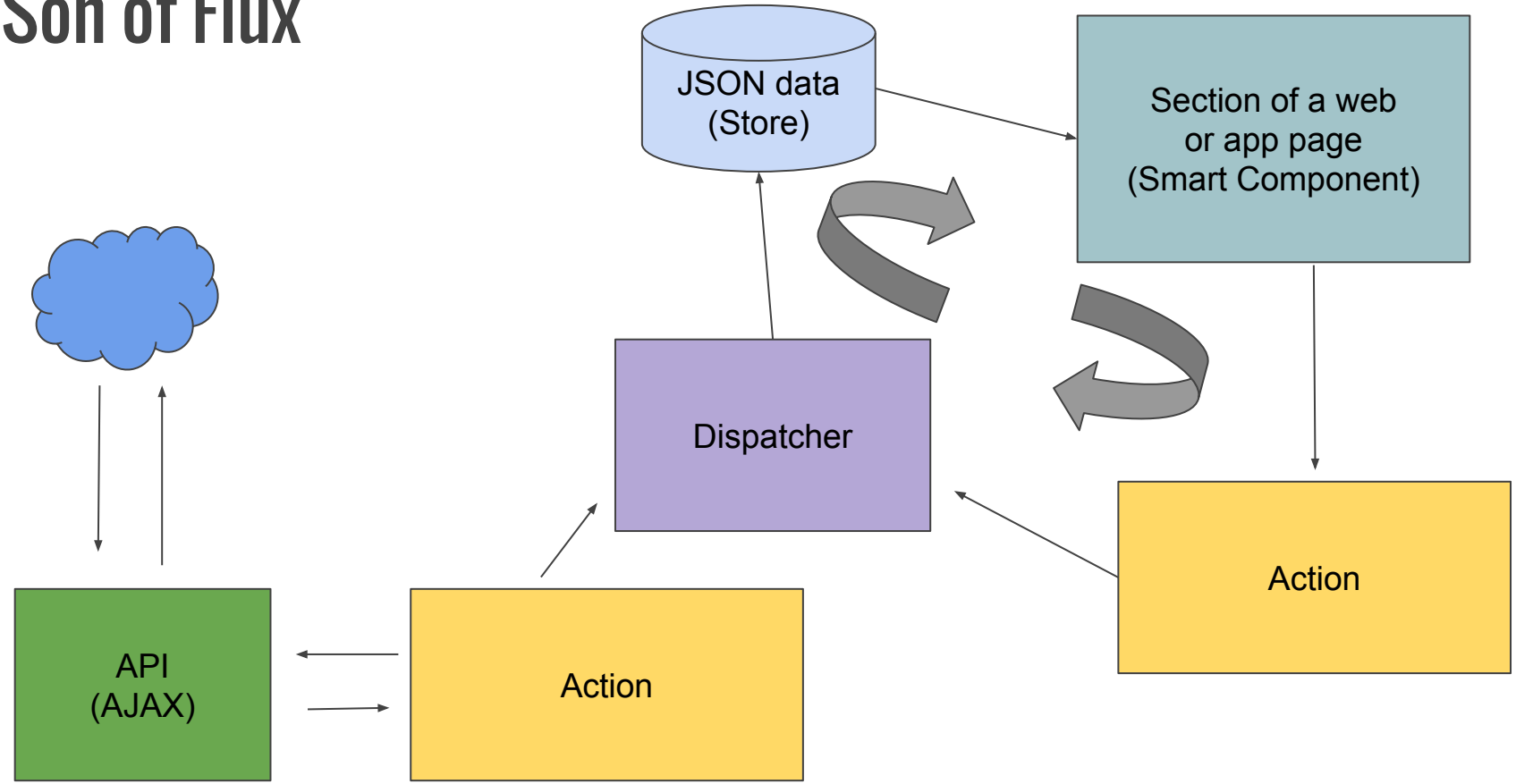
What the Flux?



Show me the money



Son of Flux


JSON data
(Store)
Section of a web
or app page
(Smart Component)
Dispatcher
Action
API
(AJAX)
Action

But I don’t LIKE mixing markup with code! It’s wroooooong!

JSX is not a railroad

You don’t HAVE to use JSX

You don’t HAVE to use ES2015

You don’t have to mix code and markup

You don’t have to drive on the right side of the road

It’s all just easier

```
statusInfo: function () {
  if ( this.props.lastOrder.order_number ) {
     var viewAddress = {};
     viewAddress.href = './?orderid=' + this.props.lastOrder.
order_number;

     var viewButton = <a className="btn btn-warning" id="viewOrder"
target="_blank"
        {...viewAddress}>View this order</a>;
     var printButton = <a className="btn btn-warning" id="
printReceipt" onClick={this.printReceipt}>Print
        this
        order</a>;
     var nextOrder = <a className="btn btn-warning" id="nextOrder"
onClick={this.props.nextOrder}>Start next
        order</a>;
     return (
        <div>
           <fieldset>
              <legend>Order Status</legend>
              <div><p>Order Submitted - this.props.lastOrder.
order_number} created {viewButton} {printButton} {nextOrder}</p></div>
           </fieldset>
        </div>
     )
  }
},
render: function () {
  return (
     <div className="row confirmList">
        {this.statusInfo()}
     </div>
  )
}
```

You’re already mixing code and markup

If you use mustaches or twigs or any other templating framework...those things aren’t markup

If you use angular, anything REAL with directives that you do is not markup

You have one or more scripts on a page...code plus markup

This is just a bit tighter

50 Shades of OK - avoiding the gotchas

When using JSX if you have any complexity to the markup, it is a good idea to contain it in a div

```
return (<option value={icon.value} key={icon.value}>{icon.text}</option>);  OK

return (<div>Item</div><div>Price</div><div>Qty</div>); might not

So wrap it in a div
return(<div>
<div>Item</div>
<div >Price</div>
<div>Qty</div>
</div>);
```

50 Shades of OK - avoiding the gotchas

When putting a class in your markup, remember that class is a Javascript word, so you have to use className

```
<div class="col-md-5">Item</div> No


<div className="col-md-5">Item</div> Yes
```

50 Shades of OK - avoiding the gotchas

Everything in the React world MUST have a unique identifier. React assigns its own identifiers BUT if you have code like this

```
var productNodes = this.props.products.map(function (product) {
return (<Product product={product} personalizationActive={product.qty > 0}
displayOnly={self.props.displayOnly}/>); });  no identifier for each product

var productNodes = this.props.products.map(function (product) {
return (<Product product={product} key={product.product_sku}
personalizationActive={product.qty > 0} displayOnly={self.props.displayOnly}
/>); });
```

Key tells React to use that as the id

50 Shades of OK - avoiding the gotchas

{ } can contain any JavaScript expression within the JSX markup. That said, you're better off doing function calls and variables, in general

```
var productNodes = this.props.products.map(function (product) {
return (<Product product={product} key={product.product_sku}
personalizationActive={product.qty > 0} displayOnly={self.props.displayOnly}
/>); });
```

50 Shades of OK - avoiding the gotchas

---

props and state are objects in each component

Anything passed into a component is attached to the prop object, and the receiving component cannot change it.

```
<Product key={index} product={product} showPrice={false} isPartOfPackage={true}
                                packageSKU={self.props.packageSKU}
                                personalizationActive={self.props.personalizationActive}
                                displayOnly={self.props.displayOnly}/>);
```

Is treated, in each product as

this.props.product, this.props.showPrice, this.isPartOfPackage

50 Shades of OK - avoiding the gotchas

---

props can be anything that Javascript can hold in a variable which is just about anything.

You can pass objects into a component for display

You can pass in a function that can be used in the component

Smart components and dumb components can be separated by passing in any functions that need to be called based on events

50 Shades of OK - avoiding the gotchas

Refactoring to spread out complexity is a good strategy

Begin with your containing component

Break it into pieces

Break the pieces into pieces

Pass functionality into the pieces

50 Shades of OK - avoiding the gotchas

It is a good idea to initialize your props or state

```
    getDefaultProps: function () {
     return {
          showPrice: true,
          isPartOfPackage: false,
          displayOnly: false,
          personalizationActive: false,
          packageSKU: ''
     }
   },
```

This is very important if an AJAX call is going to fill the values. If your component references the props or state before it has been created, you'll get an error

50 Shades of OK - avoiding the gotchas

If you bind to an event triggered from the store, be sure you unbind from that event

```
componentWillMount: function () {
OrderStore.bind(OrderConstants.ORDER_PRODUCT_CHANGED, this.getUpdate);
OrderActions.loadOrderProducts(this.props.orderId);
},
componentWillUnMount: function () {
OrderStore.unbind(OrderConstants.ORDER_PRODUCT_CHANGED, this.getUpdate);
},
```

50 Shades of OK - avoiding the gotchas

It’s a good idea to use constants for your event and trigger names. Easier reading, consistency, self documenting

```
var keyMirror = require( 'keymirror' );

module.exports = keyMirror( {
    ORDER_CREATE: null,
    STUDENT_INFORMATION_UPDATE: null,
    ORDER_HEADER_UPDATE: null,
    ORDER_HEADER_CHANGED: null,
    ORDER_LOAD: null,
} );
```

50 Shades of OK - avoiding the gotchas

Set up your application to keep everything findable

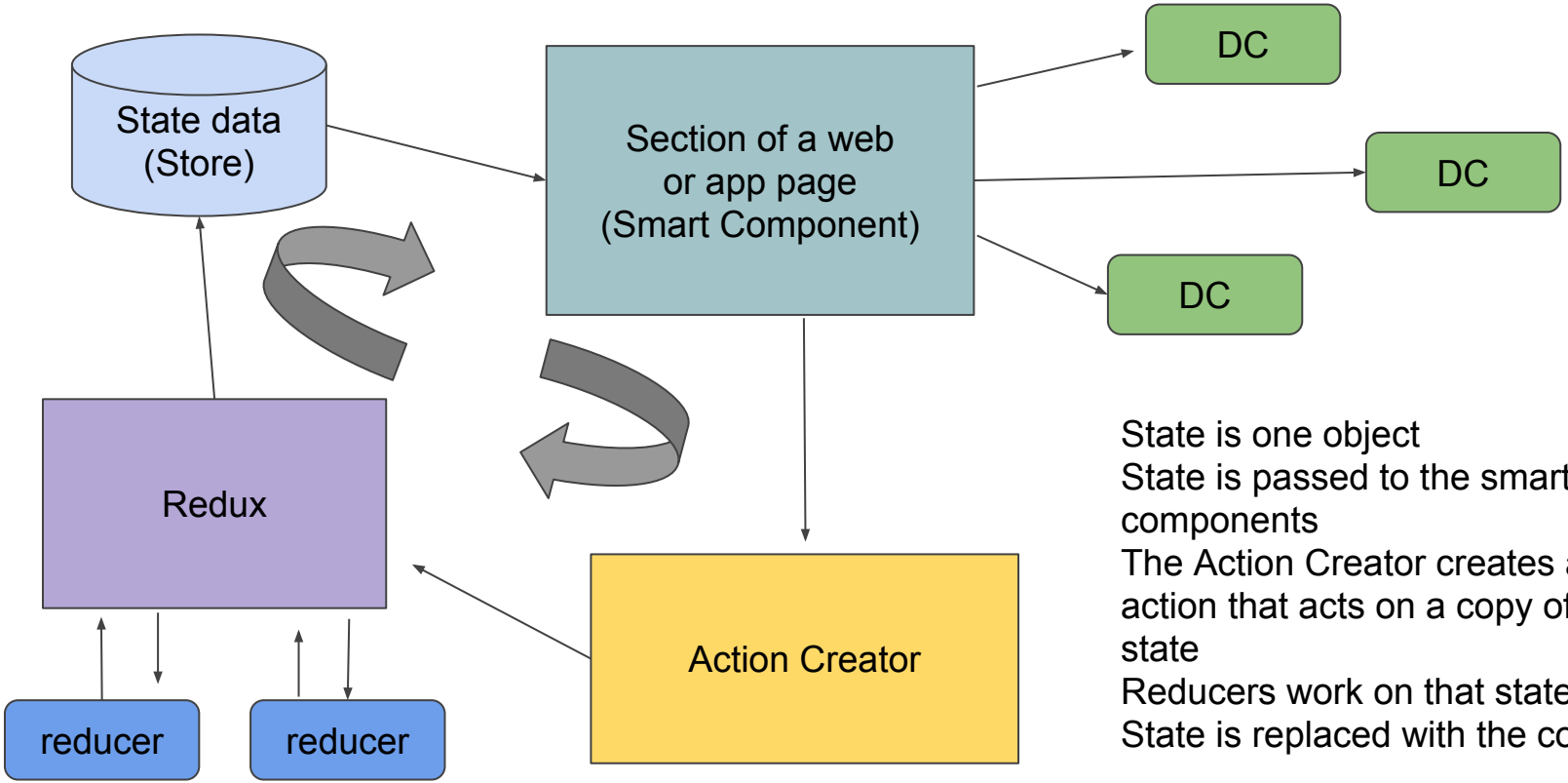
Use gulp, babelify, browserify (and others) to take your scattered modules and pull them into one file

```
gulp.task('browserify',function(){
     return browserify('./oncampusapp.js')
     .transform(babelify, {stage: 2})
     .bundle()
     .pipe(source('oncampusbundle.js'))
     .pipe(gulp.dest('.'));

});
```

Bonus Round - Redux

Redux, sort of



State is one object
State is passed to the smart components
The Action Creator creates an action that acts on a copy of the state
Reducers work on that state
State is replaced with the copy

One More Time!

Components create and update DOM

Components can only change their own state

Components cannot change their props

Components can be smart or dumb depending on what they do with state and communication

Data flows in one direction only

With Flux, data still flows in one direction...which is circular

Data coming into the system comes in through Actions and the Dispatcher

With Redux, state is handled as a totality. state is never changed. It is copied and the copy changed

En Fin

Question time

email:
    doug.johnson@balfour.com
    doug@asknice.com

github:
    dugjohnson
    dugjohnson/react-talk

google plus:
    DougJohnsonSolutionVisionary

twitter:
    @dugjohnson