

React Native

ReactJS for mobile applications

Satvik Jagannath

Software Development Engineer,
CISCO

ReactJS

- by Facebook
- in 2013
- Facebook, Instagram, Airbnb, Wordpress, Automattic, BBC, Flipkart, Myntra, Facebook Ads app, Box, Coursera, Dropbox, Dailymotion, Flipboard, IMDB, Hipchat, Khan Academy, Netflix, OkCupid, Paypal, Quizup, Reddit, Salesforce, Whatsapp, Wired, Yahoo, Atom Editor, Brackets Editor, Firefox Hello etc.,

100's of others - [Facebook Github](#)



Hello ReactJS

```
var LikeCount = React.createClass({
  getInitialState: function() {
    return {
      likes: 7,
    };
  },
  onClick: function() {
    this.setState({likes: this.state.likes + 1});
  },
  render: function() {
    var thumbsUp = '\uD83D\uDC4D';
    return (
      <View style={styles.likeContainer}>
        <TouchableOpacity onPress={this.onClick} style={styles.likeButton}>
          <Text style={styles.likesText}>
            {thumbsUp + ' Like'}
          </Text>
        </TouchableOpacity>
        <Text style={styles.likesText}>
          {this.state.likes + ' likes'}
        </Text>
      </View>
    );
  },
});
```

- UGLY?
- Templating language!!
- Separation of Concerns

--

Hello ReactJS

```
var LikeCount = React.createClass({
  getInitialState: function() {
    return {
      likes: 7,
    };
  },
  onClick: function() {
    this.setState({likes: this.state.likes + 1});
  },
  render: function() {
    var thumbsUp = '\uD83D\uDC4D';
    return (
      <View style={styles.likeContainer}>
        <TouchableOpacity onPress={this.onClick} style={styles.likeButton}>
          <Text style={styles.likesText}>
            {thumbsUp + ' Like'}
          </Text>
        </TouchableOpacity>
        <Text style={styles.likesText}>
          {this.state.likes + ' likes'}
        </Text>
      </View>
    );
  },
});
```

- It's just Javascript
- JSX Syntax
- You better like it ;)

Hello ReactJS

```
var LikeCount = React.createClass({
  getInitialState: function() {
    return {
      likes: 7,
    };
  },
  onClick: function() {
    this.setState({likes: this.state.likes + 1});
  },
  render: function() {
    var thumbsUp = '\uD83D\uDC4D';
    return (
      <View style={styles.likeContainer}>
        <TouchableOpacity onPress={this.onClick} style={styles.likeButton}>
          <Text style={styles.likesText}>
            {thumbsUp + ' Like'}
          </Text>
        </TouchableOpacity>
        <Text style={styles.likesText}>
          {this.state.likes + ' likes'}
        </Text>
      </View>
    );
  },
});
```

- 2 way binding like Angular?
- Models? Controllers?
- Where is directives?
- Global Event listener?
- Templates?...
- Works with Backbone?
- Can I also use jQuery? ;)
- I know jQuery, not JS!
is this JS?



**WHY FIRST
IMPRESSIONS
DON'T
MATTER.**

React has no

- Controllers

React has no

- Controllers
- Directives

React has no

- Controllers
- Directives
- Templates

React has no

- Controllers
- Directives
- Templates
- Global Event Listener

React has no

- Controllers
- Directives
- Templates
- Global Event Listener
- Models

React has no

- Controllers
- Directives
- Templates
- Global Event Listener
- Models
- View-Models

React has no

- Controllers
- Directives
- Templates
- Global Event Listener
- Models
- View-Models
- Blah
- Blah Blah
- Blah Blah Blah




React has

COMPONENTS

How do you separate concerns?

Shopping Cart

Items in your cart

Product		Qty	Total	
	Product title \$50.00	<input type="text" value="1"/>	\$50.00	<button>Remove</button>
	Long Product title \$150.00	<input type="text" value="1"/>	\$150.00	<button>Remove</button>
	Product title \$100.00	<input type="text" value="2"/>	\$200.00	<button>Remove</button>

Update cart Continue shopping

Subtotal




\$250.00

Checkout

Angular - Separation.

Shopping Cart

Items in your cart

Product		Qty	Total
	Product title \$50.00	<input type="text" value="1"/>	\$50.0
	Long Product title \$150.00	<input type="text" value="1"/>	\$150.0
	Product title \$100.00	<input type="text" value="2"/>	\$200.0

Update cartContinue sh

```
var CartController = new Controller({  
  addItem: function(item) {  
    ...  
  },  
  removeItem: function(item) {  
    ...  
  },  
  changeQty: function(item, qty) {  
    ...  
  },  
  ...  
});
```

Ember Separation ..

Shopping Cart

Items in your cart

Product		Qty	Total
	Product title \$50.00	<input type="text" value="1"/>	\$50.0

```
<div class="shopping-cart">
  <h1>Shopping Cart</h1>
  <div class="cart-item-count">{{cart.items.count}} items in cart</div>
  <ul class="cart-list">
    {{#cart.items}}
      <li class="cart-item"> {{name}}</li>
    {{/cart.items}}
  </ul>
</div>
```

```
var CartController = new Controller({
  addItem: function(item) {
    ...
  },
  removeItem: function(item) {
    ...
  },
  changeQty: function(item, qty) {
```

Backbone Separation...

```
var CartController = new Controller({  
  addItem: function(item) {  
    ...  
  },  
});
```

Shopping Cart

Item
Price

```
this.$el.find('#delete').on('click', function() {  
  // call addItem  
});
```

```
this.$el.find('#qty').on('change', function() {  
  // call changeQty  
});
```

```
<div>  
  <div>  
    <div>  
      <ul>  
        {{#cart.items}}  
        <li class="cart-item"> {{name}}</li>  
        {{/cart.items}}  
      </ul>  
    </div>  
  </div>
```

Shonning Cart

```
var CartController = new Controller({  
  addItem: function(item) {  
    ...  
  },
```

```
Item: this.$el.find('#delete').on('click', function() {  
  // call addItem
```



"Separation of concerns!!"




```
    // call changeQty  
  });
```

```
    {{#cart.items}}  
    <li class="cart-item"> {{name}}</li>  
    {{/cart.items}}  
  
  </ul>  
</div>
```

What if we Separate in ANOTHER way?

Shopping Cart

Items in your cart

Product	Qty	Total	
 Product title \$50.00	<input type="text" value="1"/>	\$50.00	Remove
 Long Product title \$150.00	<input type="text" value="1"/>	\$150.00	Remove
 Product title \$100.00	<input type="text" value="2"/>	\$200.00	Remove

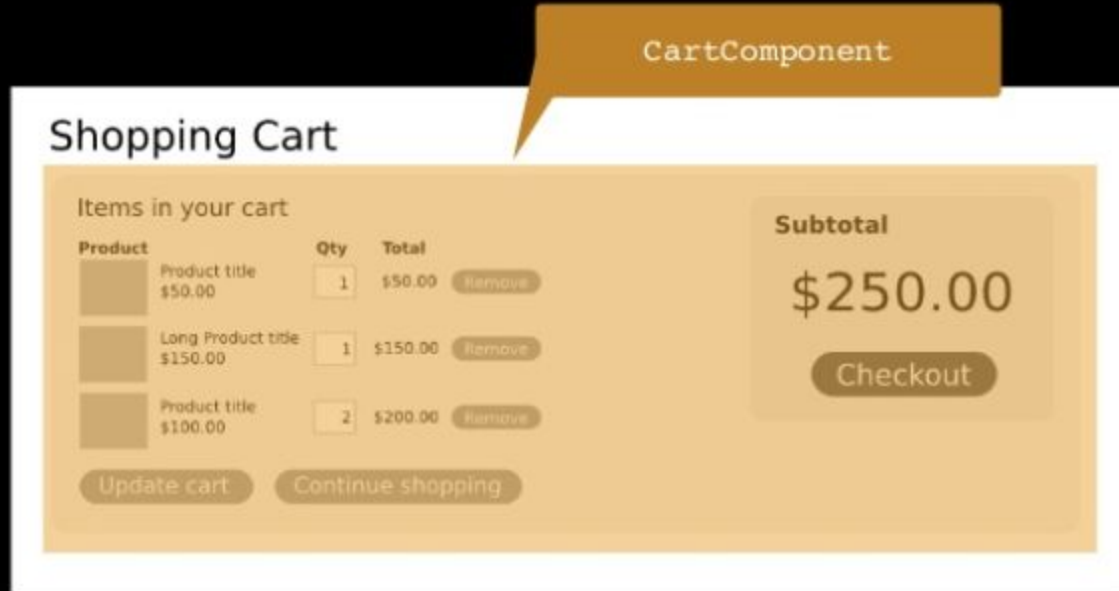
[Update cart](#) [Continue shopping](#)

Subtotal

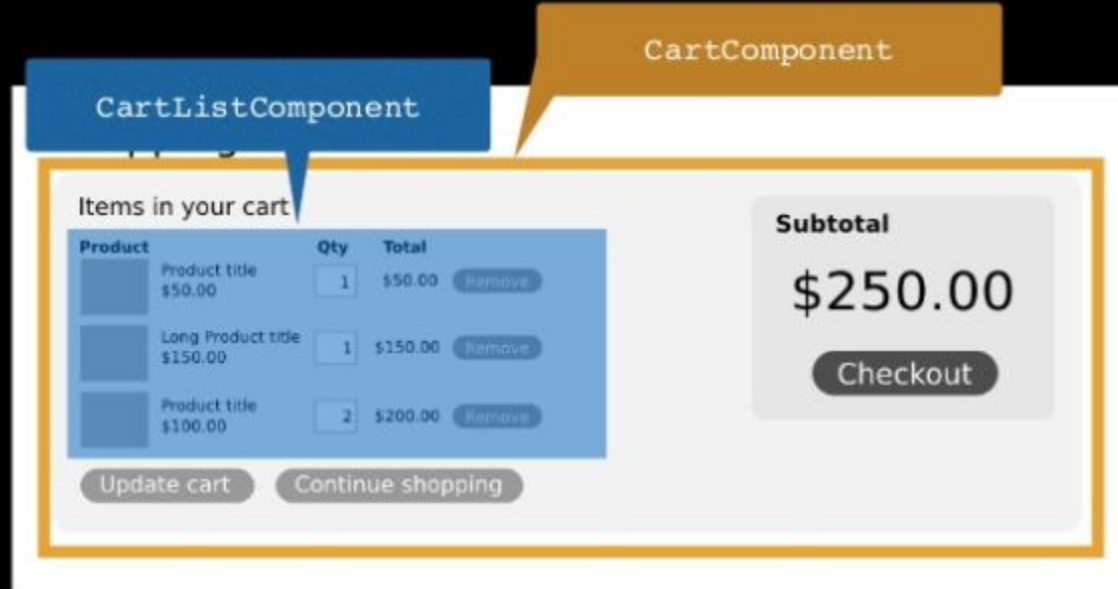
\$250.00

[Checkout](#)

What if we Separate in ANOTHER way?



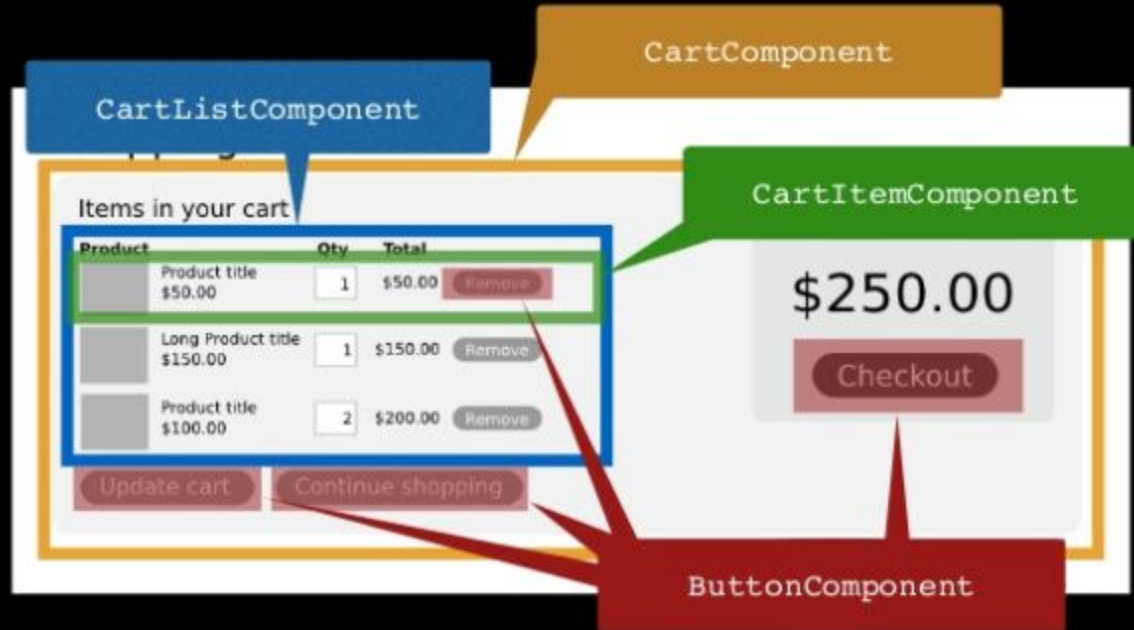
What if we Separate in ANOTHER way?



What if we Separate in ANOTHER way?



What if we Separate in ANOTHER way?



Yay baby!



Separation of Components!

- Composable
- Reusable
- Maintainable
- Testable

IF AND ONLY IF, components are self-contained.

Data flow

- Pure JS - Any component can communicate with any other component
- Backbone - Publisher Subscriber model

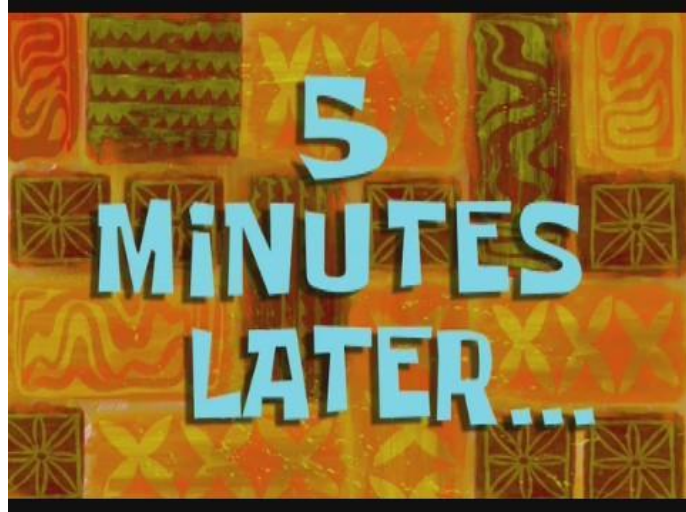
```
item.on('change:name', function() { ...
```

- Angular - 2 way data binding & \$digest loop

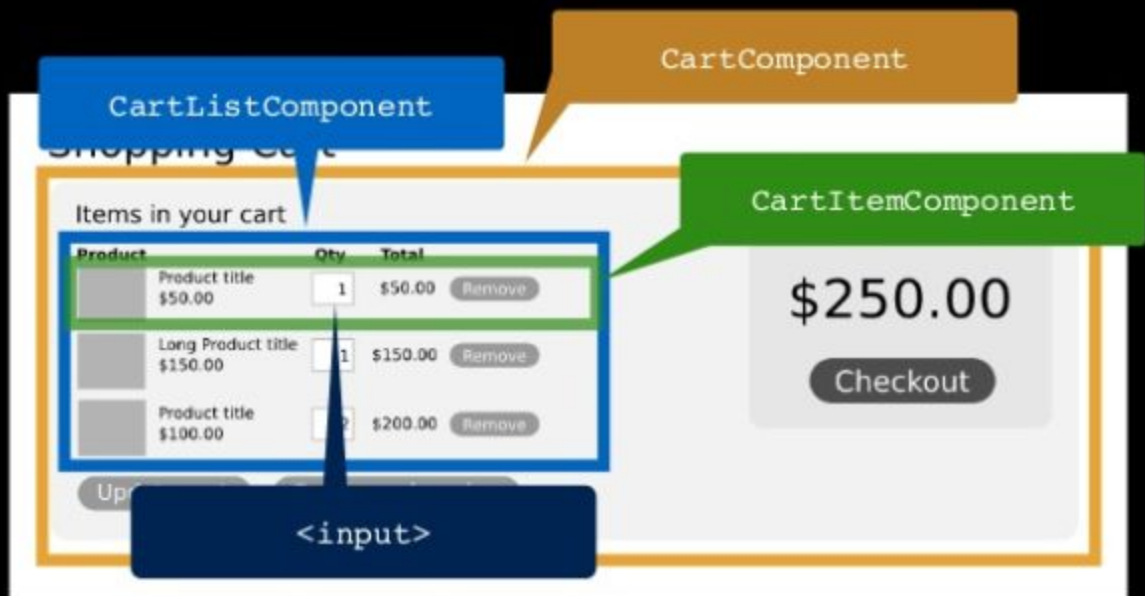
```
$scope.name = ...
```

- React - 1 way data “FLOW”

One Way data FLOW sounds GREAT!



That's NOT how real world works!






Data flows DOWN with handlers

CartComponent

```
<CartListComponent  
  list={this.state.list}  
  onChange={this.handleChange}/>
```

Shopping Cart

Items in your cart

Product	Qty	Total	
 Product title \$50.00	<input type="text" value="1"/>	\$50.00	<button>Remove</button>
 Long Product title \$150.00	<input type="text" value="1"/>	\$150.00	<button>Remove</button>
 Product title \$100.00	<input type="text" value="2"/>	\$200.00	<button>Remove</button>

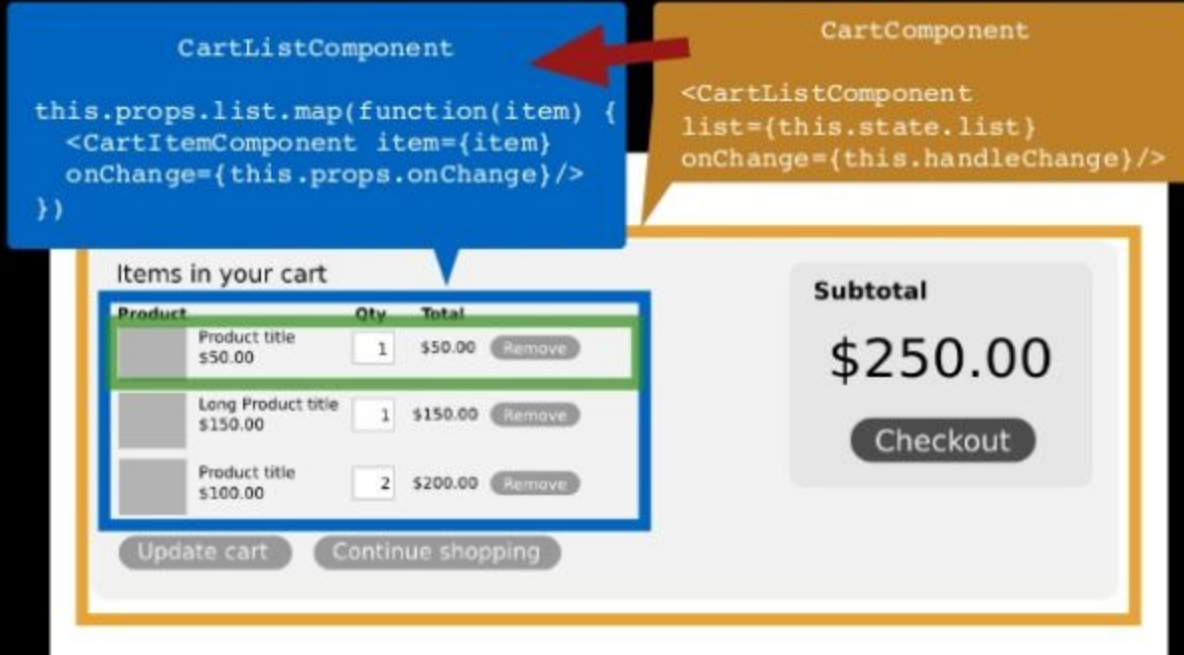
Update cart Continue shopping

Subtotal

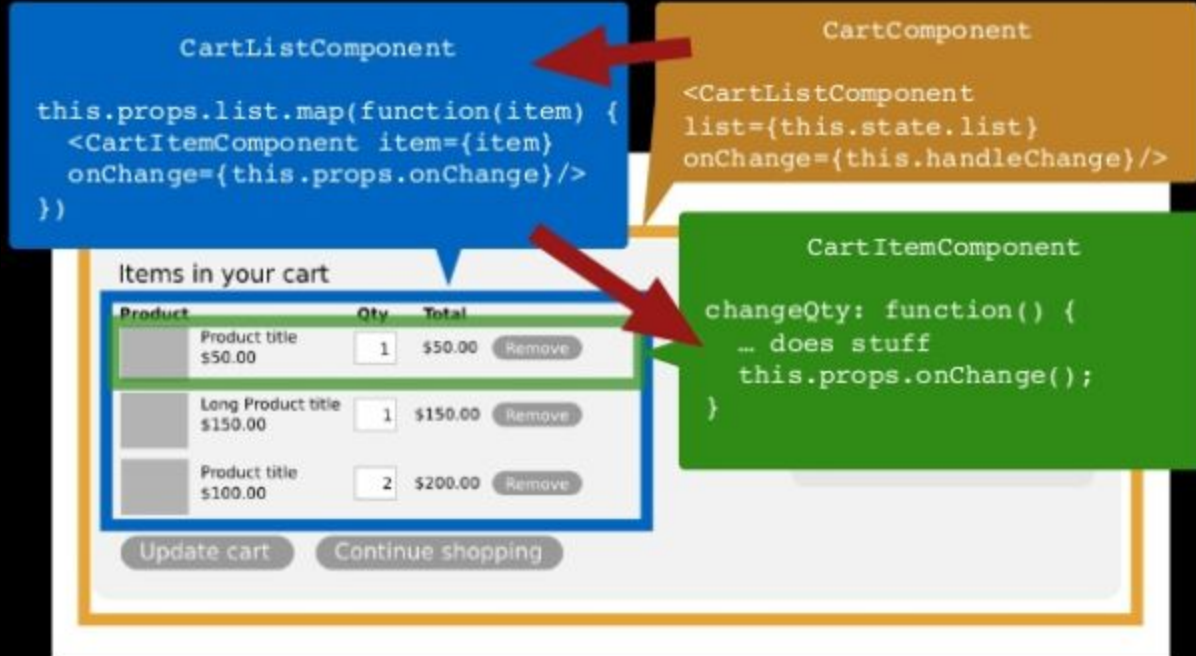
\$250.00

Checkout

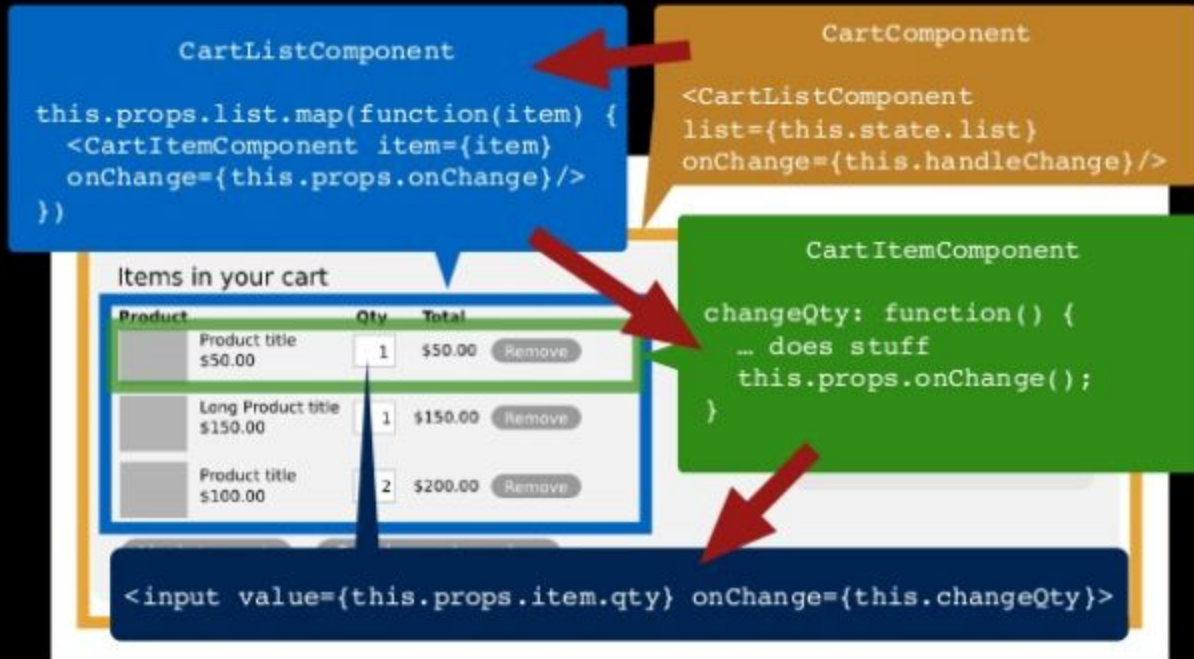
Data flows DOWN with handlers



Data flows DOWN with handlers



Data flows DOWN with handlers



Here we goooooo

CartListComponent

```
this.props.list.map(function(item) {  
  <CartItemComponent item={item}  
    onChange={this.props.onChange}/>  
})
```

CartComponent

```
<CartListComponent  
  list={this.state.list}  
  onChange={this.handleChange}/>
```

CartItemComponent

```
changeQty: function() {  
  ... does stuff  
  this.props.onChange();  
}
```

Items in your cart

Product	Qty	Total	
 Product title \$50.00	<input type="text" value="1"/>	\$50.00	<button>Remove</button>
 Long Product title \$150.00	<input type="text" value="1"/>	\$150.00	<button>Remove</button>
 Product title \$100.00	<input type="text" value="2"/>	\$200.00	<button>Remove</button>

```
<input value={this.props.item.qty} onChange={this.changeQty}>
```

Event Flows UP

CartListComponent

```
this.props.list.map(function(item) {  
  <CartItemComponent item={item}  
    onChange={this.props.onChange}/>  
})
```

CartComponent

```
<CartListComponent  
  list={this.state.list}  
  onChange={this.handleChange}/>
```

CartItemComponent

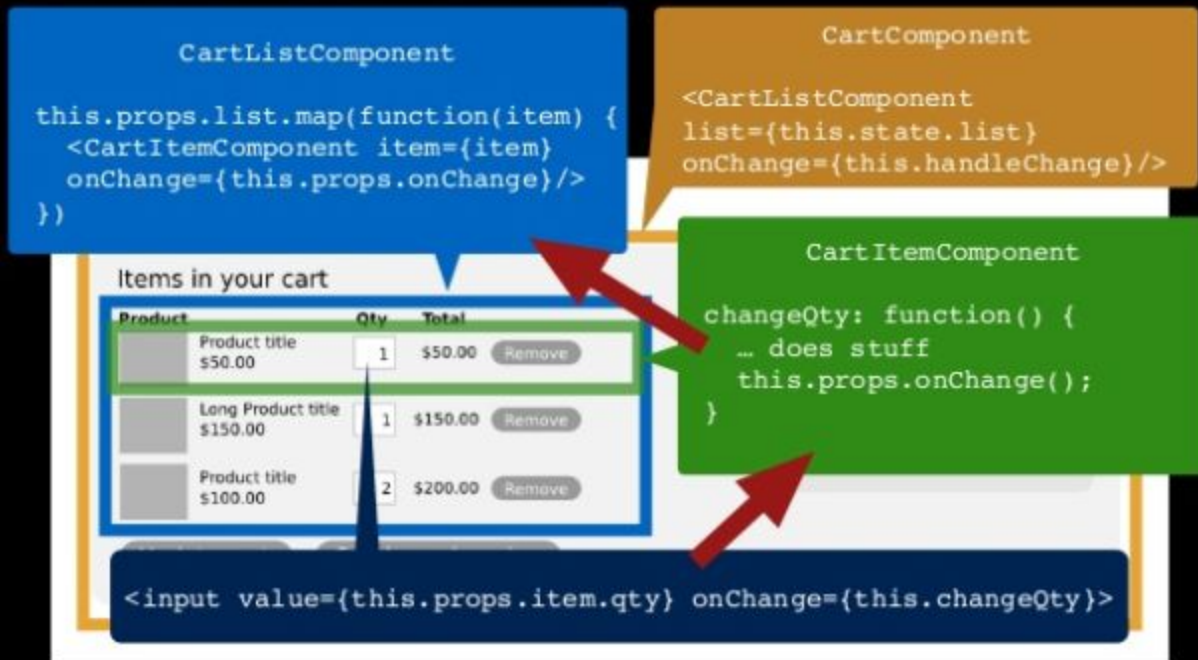
```
changeQty: function() {  
  ... does stuff  
  this.props.onChange();  
}
```

Items in your cart

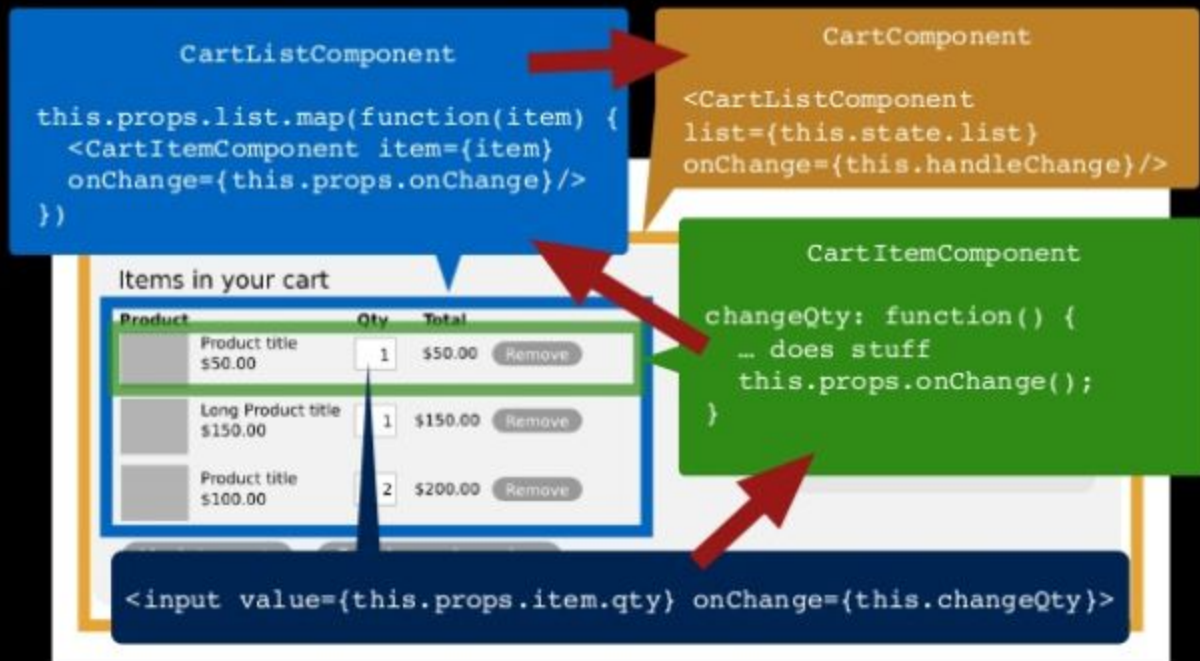
Product	Qty	Total	
 Product title \$50.00	<input type="text" value="1"/>	\$50.00	<button>Remove</button>
 Long Product title \$150.00	<input type="text" value="1"/>	\$150.00	<button>Remove</button>
 Product title \$100.00	<input type="text" value="2"/>	\$200.00	<button>Remove</button>

```
<input value={this.props.item.qty} onChange={this.changeQty}>
```

Event Flows UP



Event Flows UP



Data flows DOWN

Event Flows UP

Sounds familiar?

Data flows DOWN

Event Flows UP

That's how **DOM** works!

Virtual DOM

- React uses Virtual DOM
- 2 DOM's will co-exist - Real and Virtual
- What worse that having 2 DOM's? Having 1 DOM!

Why not DOM?

- Inconsistent
- Hard to test
- Brittle
- Expensive operation

Back to JSX

```
var LikeButton = React.createClass({
  getInitialState: function() {
    return {liked: false};
  },
  handleClick: function(event) {
    this.setState({liked: !this.state.liked});
  },
  render: function() {
    var text = this.state.liked ? 'like' : 'unlike';
    return (
      <p onClick={this.handleClick}>
        You {text} this. Click to toggle.
      </p>
    );
  }
});
```

Compiled JSX

```
var LikeButton = React.createClass({
  getInitialState: function() {
    return {liked: false};
  },
  handleClick: function(event) {
    this.setState({liked: !this.state.liked});
  },
  render: function() {
    var text = this.state.liked ? 'like' : 'unlike';
    return (
      React.DOM.p( {onClick:this.handleClick},
        "You ", text, " this. Click to toggle."
      )
    );
  }
});
```


Virtual DOM

- Pure JS, in-memory representation of DOM
- render() fires when there is a change in state
- React takes care of modifying the Real DOM to match the Virtual DOM
- It's fast
- It's pure
- It just works

Virtual DOM - How's?

How do I access the actual DOM?
How do I know when render() is done?

Lifecycle Method



```
componentDidMount: function() {  
  var $el = $(this.getDOMNode());  
  this.setState({ width : $el.width() });  
}
```

The diagram shows a code block with a red arrow pointing from the text 'Lifecycle Method' to the `componentDidMount` property. Another red arrow points from the text 'Actual DOM Node' to the `getDOMNode()` method call within the function.

Actual DOM Node

Declarative Syntax

Run jQuery code after AngularJS completes rendering HTML

CAREERS 2.0
by stackoverflow



>



Easily apply for your dream job
No formatting needed!



30



8

In controller I get some JSON data using \$http or \$resource services. Then I write this data in \$scope and AngularJS updates HTML structure of the page. My problem is that I need to know what is the new size (width and height) of the list (I mean, HTML DOM element) that is filled with Angular ng-repeat directive. Consequently, I have to run javascript code right after Angular finishes updating DOM structure. What is the proper way to do it? I have searched internet over the last four hours but I couldn't find any solution to my problem.

This is how I receive JSON data:

```
var tradesInfo = TradesInfo.get({}, function(data){
    console.log(data);
    $scope.source.profile = data.profile;
    $scope.trades = $scope.source.profile.trades;
    $scope.activetrade = $scope.trades[0];
    $scope.ready = true;


    init(); //I need to call this function after update is complete
});
```

Sounds legit!

28

Actually in this case the angular way is not the easy way but the only right way :)

You have to write a directive and attach to the element you want to know the height of. And from the controller you \$broadcast an event, the directive'll catch the event and there you can do the DOM manipulation. NEVER in the controller.



```
var tradesInfo = TradesInfo.get({}, function(data){
  console.log(data);
  $scope.source.profile = data.profile;
  ...

  $scope.$broadcast('dataloaded');
});
```

Run jQuery code after AngularJS completes rendering HTML

```
directive/... {
  link($scope, element, attrs) {
    $scope.$on('dataloaded', function () {
      $timeout(function () { // You might need this timeout to be sure its run after DOM render.
        element.width()
        element.height()
      }, 0, false);
    });
  }
};
```

wut.

Why React is awesome?

- One way dataflow keeps complexity under-control
- You really know this changed because of that! Blame game!
- Easy to debug - Self contained components
- Library doesn't dictate too much!
- Simple and easy to learn