React

and why it’s awesome

Andrew Hull
@aghull
normative.com

React First Impressions

```
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>
    );
  }
});
```

React First Impressions

```
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>
    );
  }
});
```

React First Impressions

```
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>
    );
  }
});
```

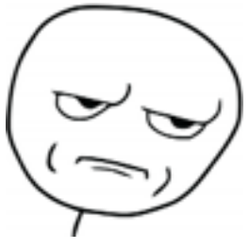
WTF?!

“Reaction” to React

```
  <p onClick={this.handleClick}>
    You {text} this. Click to toggle.
  </p>
```

“Reaction” to React

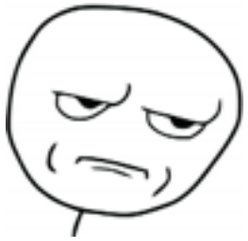
```
  <p onClick={this.handleClick}>
    You {text} this. Click to toggle.
  </p>
```



“Ugly.”

“Reaction” to React

```
  <p onClick={this.handleClick}>
    You {text} this. Click to toggle.
  </p>
);
```

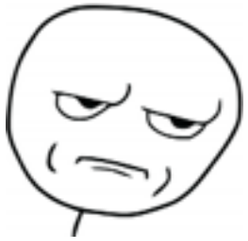


“Ugly.”

“Separation of concerns!!”

“Reaction” to React

```
  <p onClick={this.handleClick}>
    You {text} this. Click to toggle.
  </p>
);
```



“Ugly.”

“Separation of concerns!!”

“React is a templating language.”

“Reaction” to React

```
  <p onClick={this.handleClick}>
    You {text} this. Click to toggle.
  </p>
);
```

“Reaction” to React

```
  <p onClick={this.handleClick}>
    You {text} this. Click to toggle.
  </p>
);
```

- **Don’t sweat it**

“Reaction” to React

```
  <p onClick={this.handleClick}>
    You {text} this. Click to toggle.
  </p>
);
```

- **Don’t sweat it**
- **JSX**

“Reaction” to React

```
  <p onClick={this.handleClick}>
    You {text} this. Click to toggle.
  </p>
);
```

- **Don’t sweat it**
- **JSX**
- **It’s actually just Javascript**

“Reaction” to React

```
  <p onClick={this.handleClick}>
    You {text} this. Click to toggle.
  </p>
);
```

- **Don’t sweat it**
- **JSX**
- **It’s actually just Javascript**
- **It’s not a templating language**

“Reaction” to React

```
  <p onClick={this.handleClick}>
    You {text} this. Click to toggle.
  </p>
);
```

- **Don’t sweat it**
- **JSX**
- **It’s actually just Javascript**
- **It’s not a templating language**
- **If you don’t like it, you don’t hafta**

React Second Impressions

React Second Impressions

- Oh! There's 2-way data-binding, like Angular!

React Second Impressions

- Oh! There’s 2-way data-binding, like Angular!
- Oh! It can work with Backbone. How I do?

React Second Impressions

- Oh! There's 2-way data-binding, like Angular!
- Oh! It can work with Backbone. How I do?
- Oh! It can do animations and SVG!

React Second Impressions

- Oh! There [illegible] way data-bi[illegible]ng, like Angular!
- Oh! It can work [illegible]ckbone. How I do?
- Oh! It ca[illegible] animations and [illegible]!

Wait. Let’s back up.

Some Fundamentals

#1

Everything is a Component

React has no...

React has no...

... controllers

React has no...

... controllers

... directives

React has no...

... controllers

... directives

... templates

React has no...

... controllers
... directives
... templates
... global event listeners

React has no...

... controllers
... directives
... templates
... global event listeners
... models

React has no...

... controllers
... directives
... templates
... global event listeners
... models
... no view models

React has no...

... controllers
... directives
... templates
... global event listeners
... models
... no view models

Just

React has no...

... controllers
... directives
... templates
... global event listeners
... models
... no view models

Just Components

Shopping Cart

Items in your cart

Product	Qty	Total	
Product title $50.00	1	$50.00	Remove
Long Product title $150.00	1	$150.00	Remove
Product title $100.00	2	$200.00	Remove

Update cart Continue shopping

Subtotal

$250.00

Checkout

Shopping Cart

Items in your cart

Product	Qty	Total
Product title $50.00	1	$50.0
Long Product title $150.00	1	$150.0
Product title $100.00	2	$200.0

Update cart Continue sh

```
var CartController = new Controller({

  addItem: function(item) {
    ...
  },

  removeItem: function(item) {
    ...
  },

  changeQty: function(item, qty) {
    ...
  },

  ...

});
```

Shopping Cart

Items in your cart

Product	Qty	Total
Product title $50.00	1	$50.0

```
var CartController = new Controller({

  addItem: function(item) {
    ...
  },

  removeItem: function(item) {
    ...
  },

  changeQty: function(item, qty) {
```

```
<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"><img src="{{thumbnail}}"> {{name}}</li>
    {{/cart.items}}

  </ul>

</div>
```

```
var CartController = new Controller({

  addItem: function(item) {
    ...
  },
```

Shopping Cart

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


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

```
<div

  <h1

  <di

  <ul

    {{#cart.items}}
      <li class="cart-item"><img src="{{thumbnail}}"> {{name}}</li>
    {{/cart.items}}

  </ul>

</div>
```

```
var CartController = new Controller({

  addItem: function(item) {
    ...
  },
```

Shopping Cart

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

“Separation of concerns!!”

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

```
<d
  <h1
  <di
  <ul

    {{#cart.items}}
      <li class="cart-item"><img src="{{thumbnail}}"> {{name}}</li>
    {{/cart.items}}

  </ul>

</div>
```

```
var CartController = new Controller({

  addItem: function(item) {
    ...
  },
```

Shopping Cart

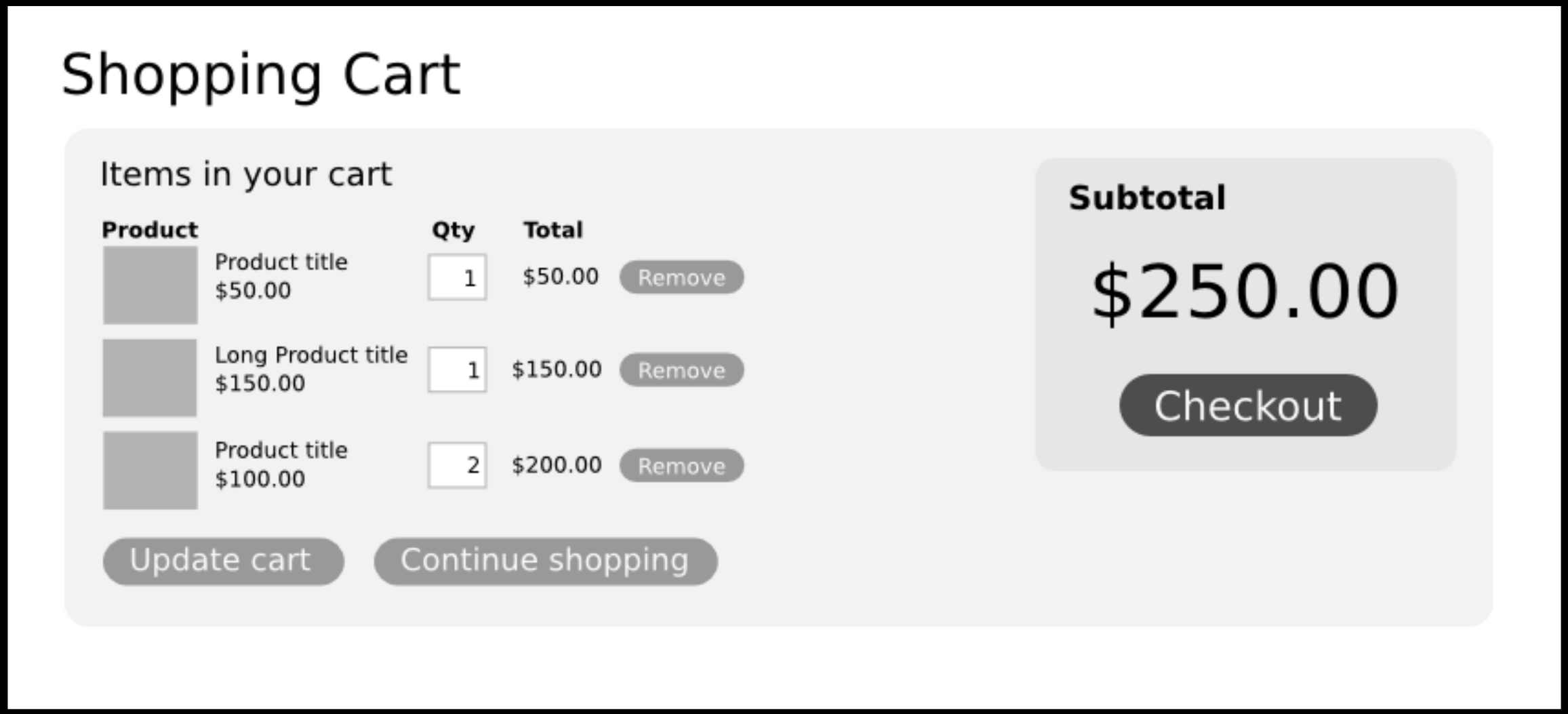
```
this.$el.find('#delete').on('click', function() {
  // call addItem
```

“Do you even MVC?”

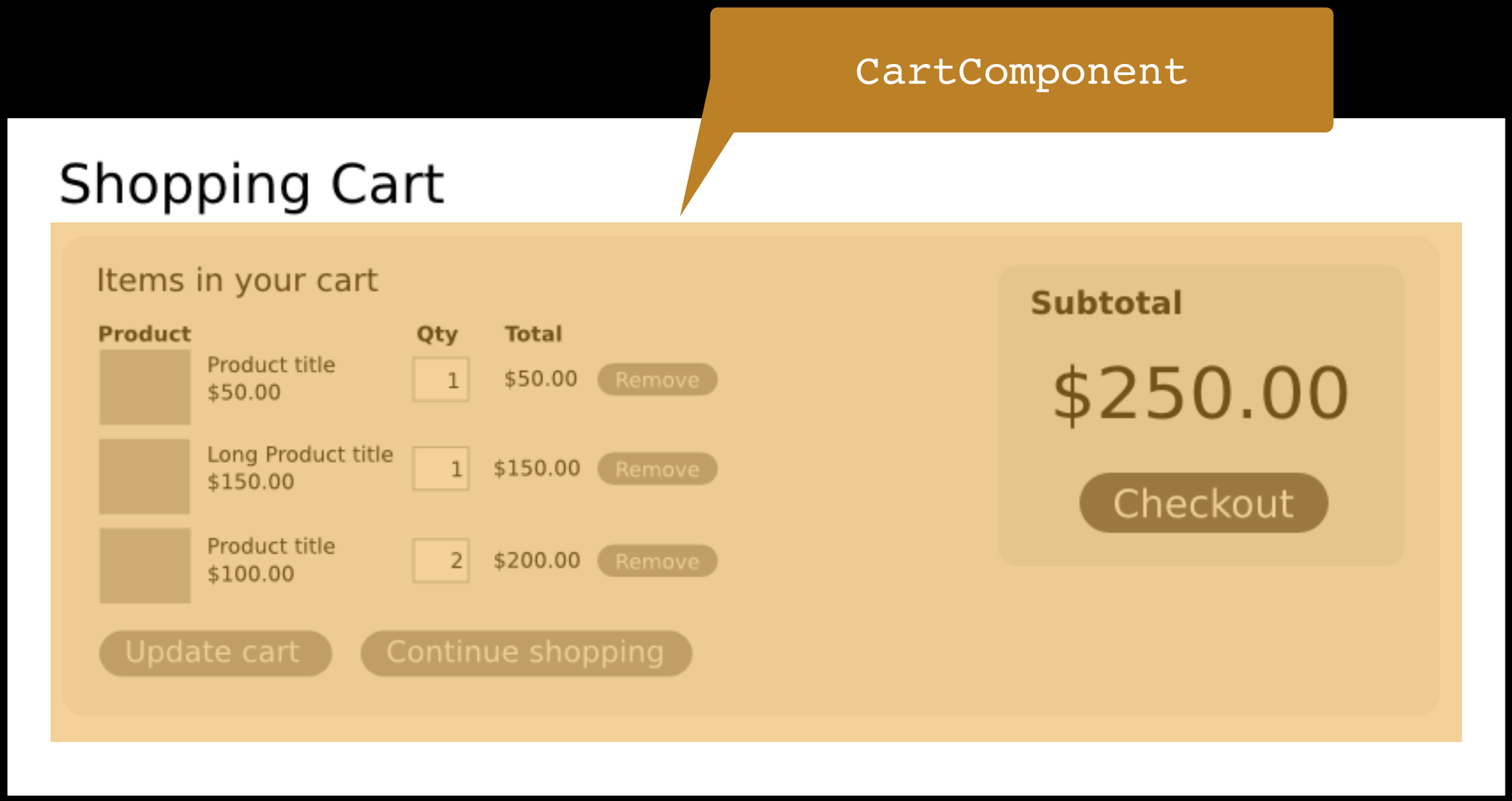
```
    // call changeQty
  });
```

```
<d
  <h1
  <div
  <ul

    {{#cart.items}}
      <li class="cart-item"><img src="{{thumbnail}}"> {{name}}</li>
    {{/cart.items}}

  </ul>

</div>
```

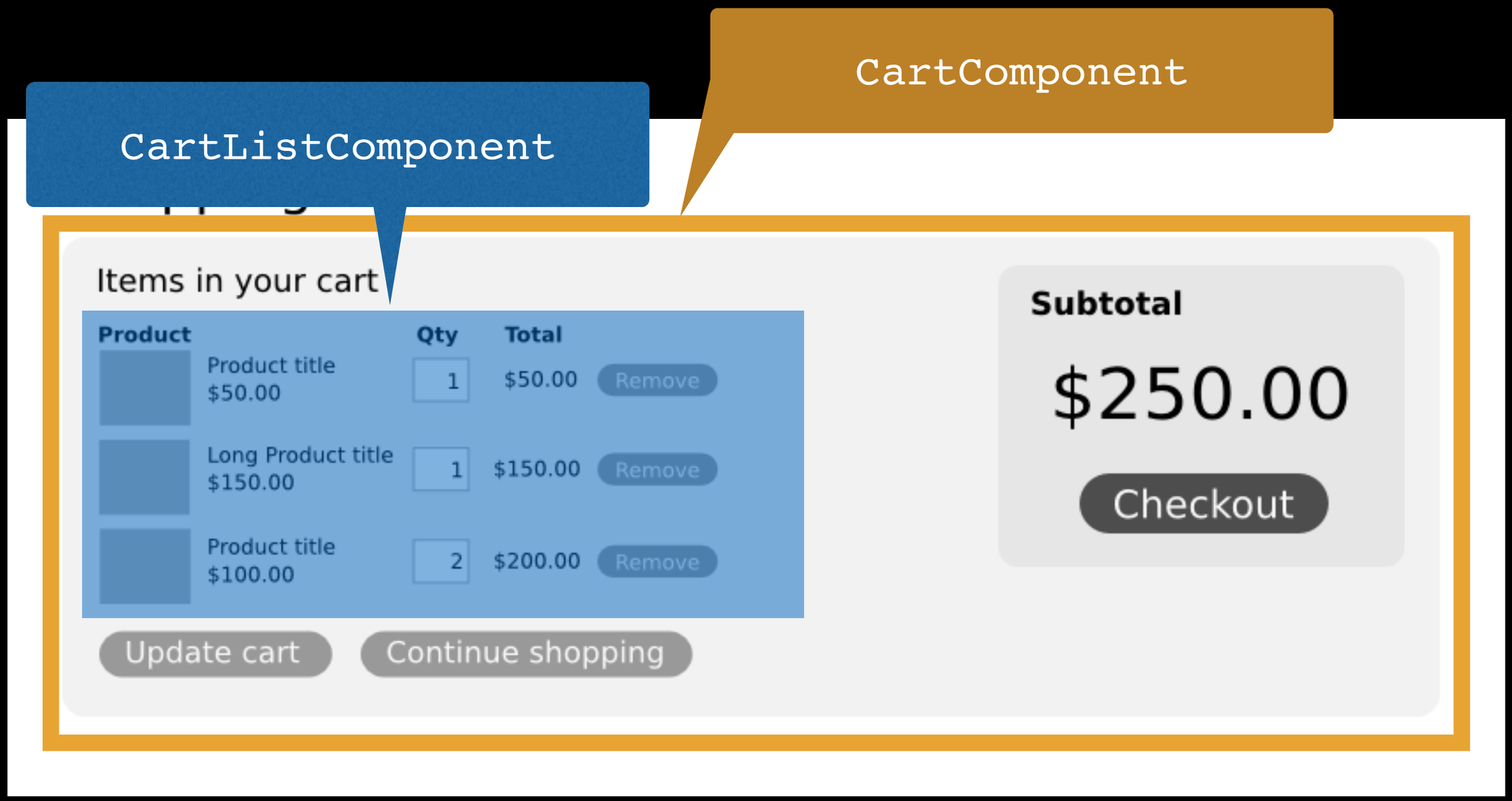
What if we “separate” another way?



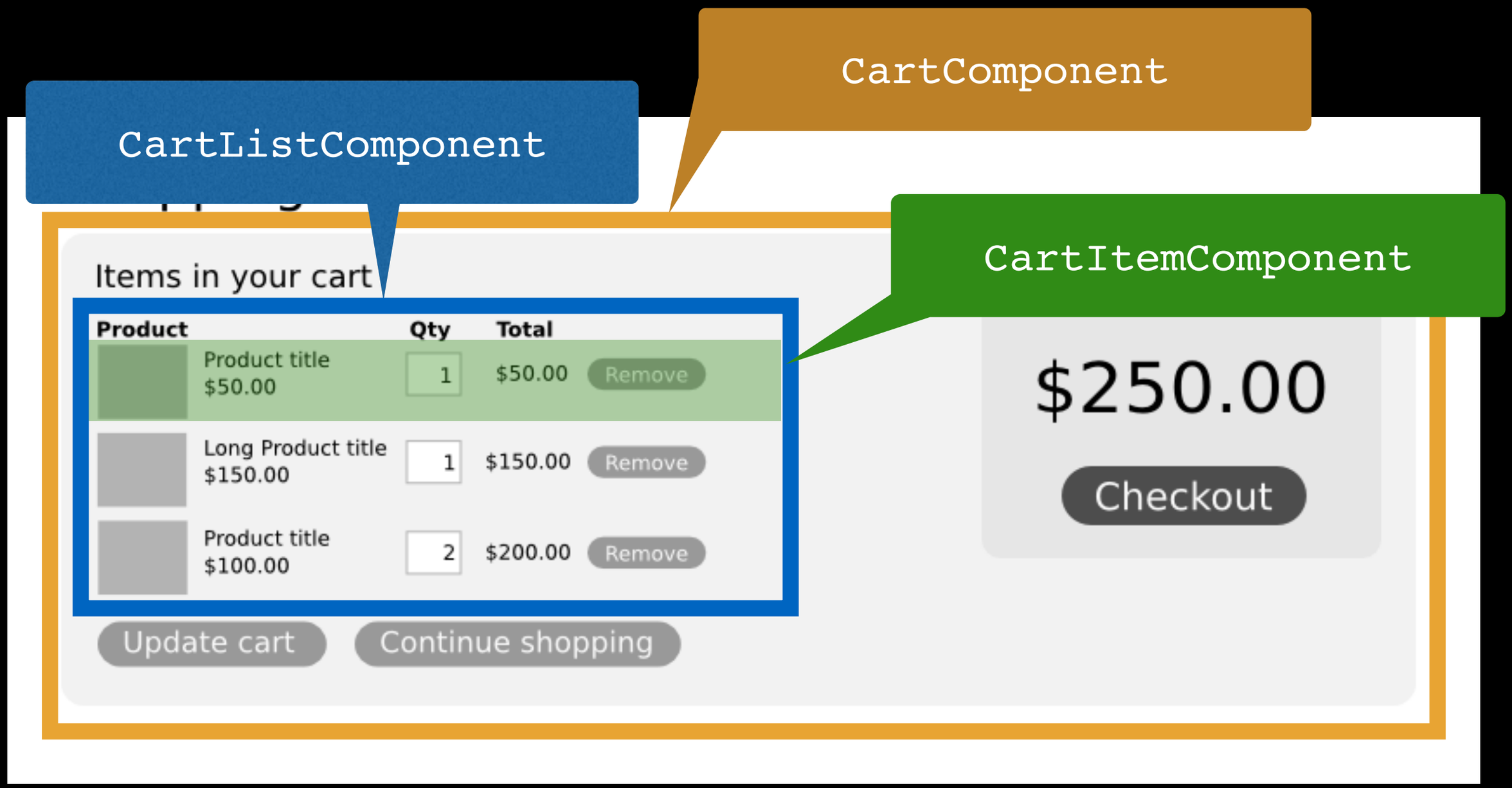
What if we "separate" another way?



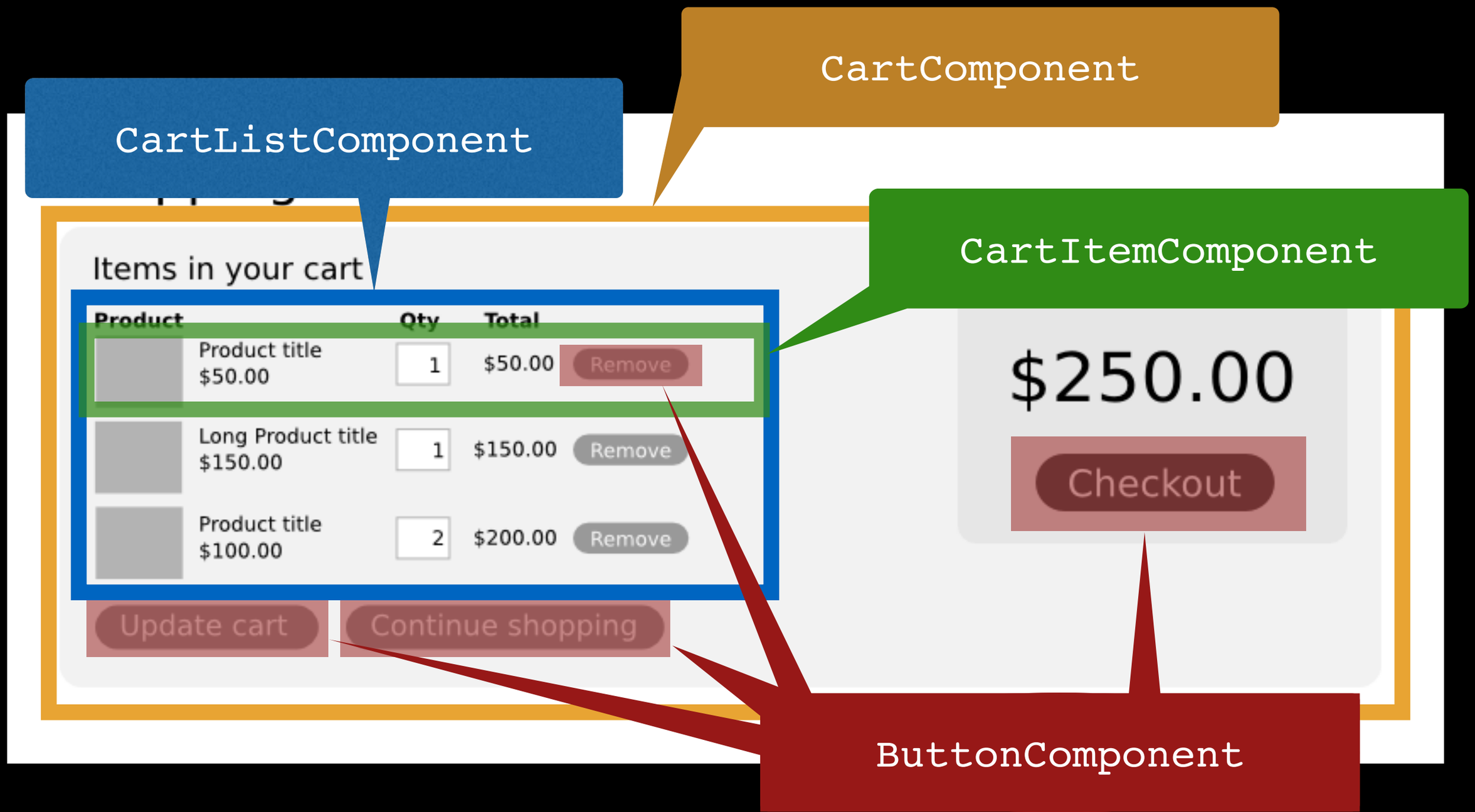
What if we "separate" another way?



What if we "separate" another way?



What if we "separate" another way?



Separation of ~~Concerns~~ Components

Separation of ~~Concerns~~ Components

- composable

Separation of ~~Concerns~~ Components

- composable
- reusable

Separation of ~~Concerns~~ Components

- composable
- reusable
- maintainable

Separation of ~~Concerns~~ Components

- composable
- reusable
- maintainable
- testable

Separation of ~~Concerns~~ Components

- composable
- reusable
- maintainable
- testable

If Components are truly self-contained

#2
Single Source of Truth

Traditional data flows

Traditional data flows

No framework: Any component can communicate with any other component

Traditional data flows

No framework: Any component can communicate with any other component

Backbone: Pub-sub

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

Traditional data flows

No framework: Any component can communicate with any other component

Backbone: Pub-sub

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

Angular: 2-way data binding and $digest loop

```
$scope.name = …
```

Traditional data flows

No framework: Any component can communicate with any other component

Backbone: Pub-sub

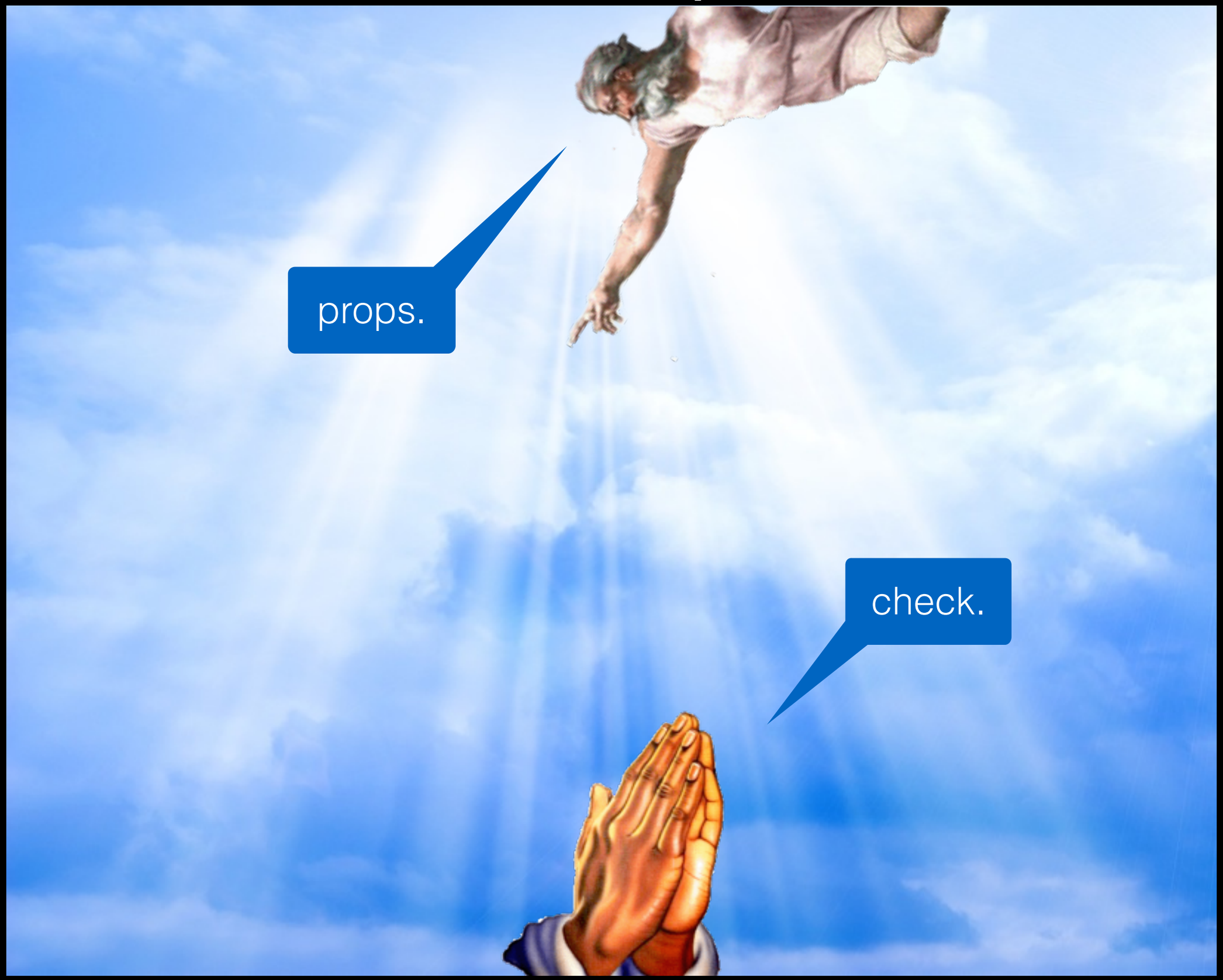
```
item.on('change:name', function() {…
```

Angular: 2-way data binding and $digest loop

```
$scope.name = …
```

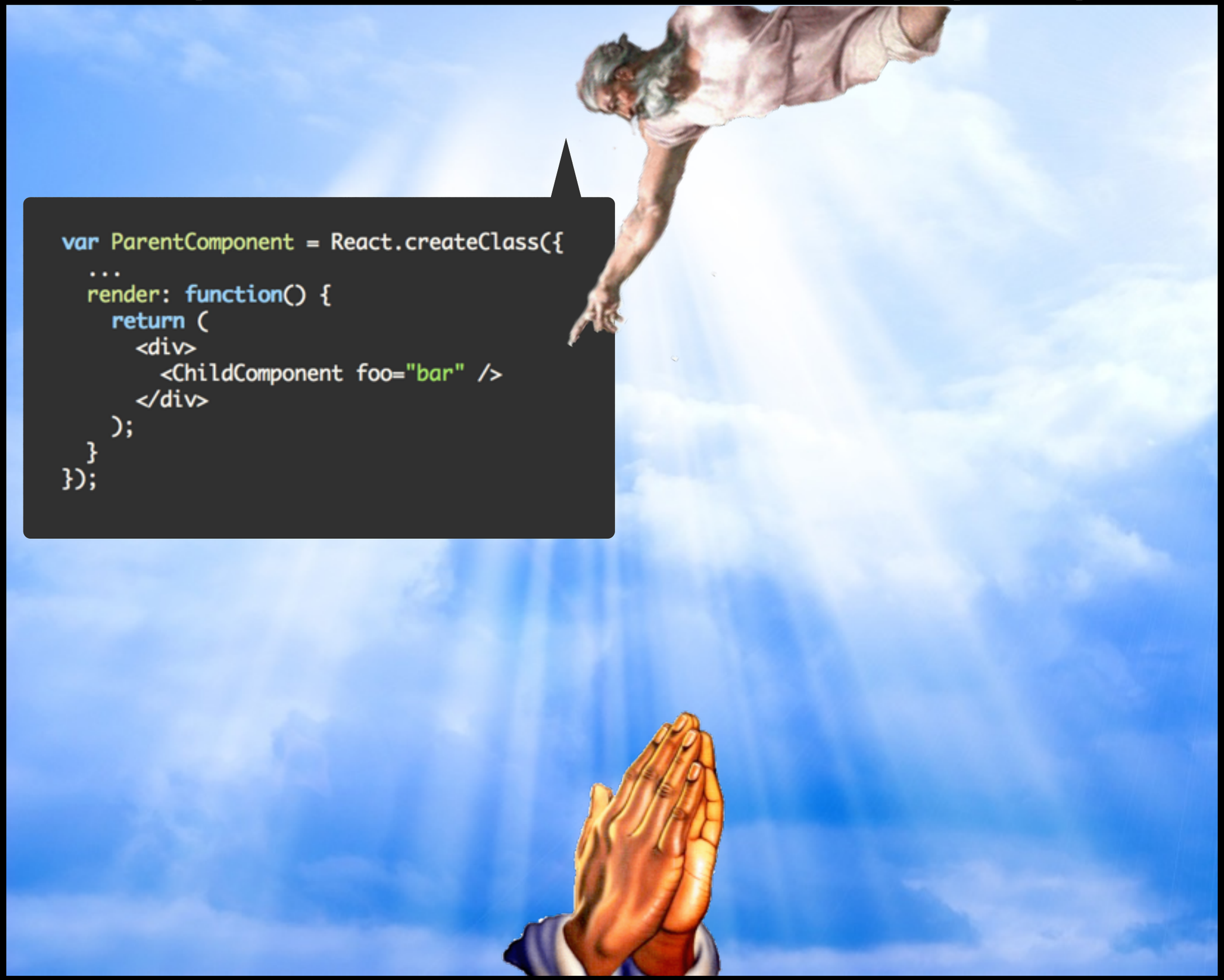
React: 1-way data flow

Data handed from parent to child



Props accessed on this.props

Props accessed on this.props



```
var ParentComponent = React.createClass({
  ...
  render: function() {
    return (
      <div>
        <ChildComponent foo="bar" />
      </div>
    );
  }
});
```

Props accessed on this.props

```
var ParentComponent = React.createClass({
  ...
  render: function() {
    return (
      <div>
        <ChildComponent foo="bar" />
      </div>
    );
  }
});
```

Props accessed on this.props

Props accessed on this.props

Props are immutable

Props are immutable

```
var ChildComponent = React.createClass({
  ...
  doStuff: function() {
    this.props.foo = 'something else'
  }
});
```

Props are immutable

Props are immutable

State is mutable

```
var ChildComponent = React.createClass({
  ...
  makeActive: function() {
    this.setState({ active: true });
  },

  render: function() {
    return <input type="checkbox" checked={this.state.active} />
  }
});
```

State is mutable

```
var ChildComponent = React.createClass({
  ...
  makeActive: function() {
    this.setState({ active: true });
  },

  render: function() {
    return <input type="checkbox" checked={this.state.active} />
  }
});
```

State is mutable

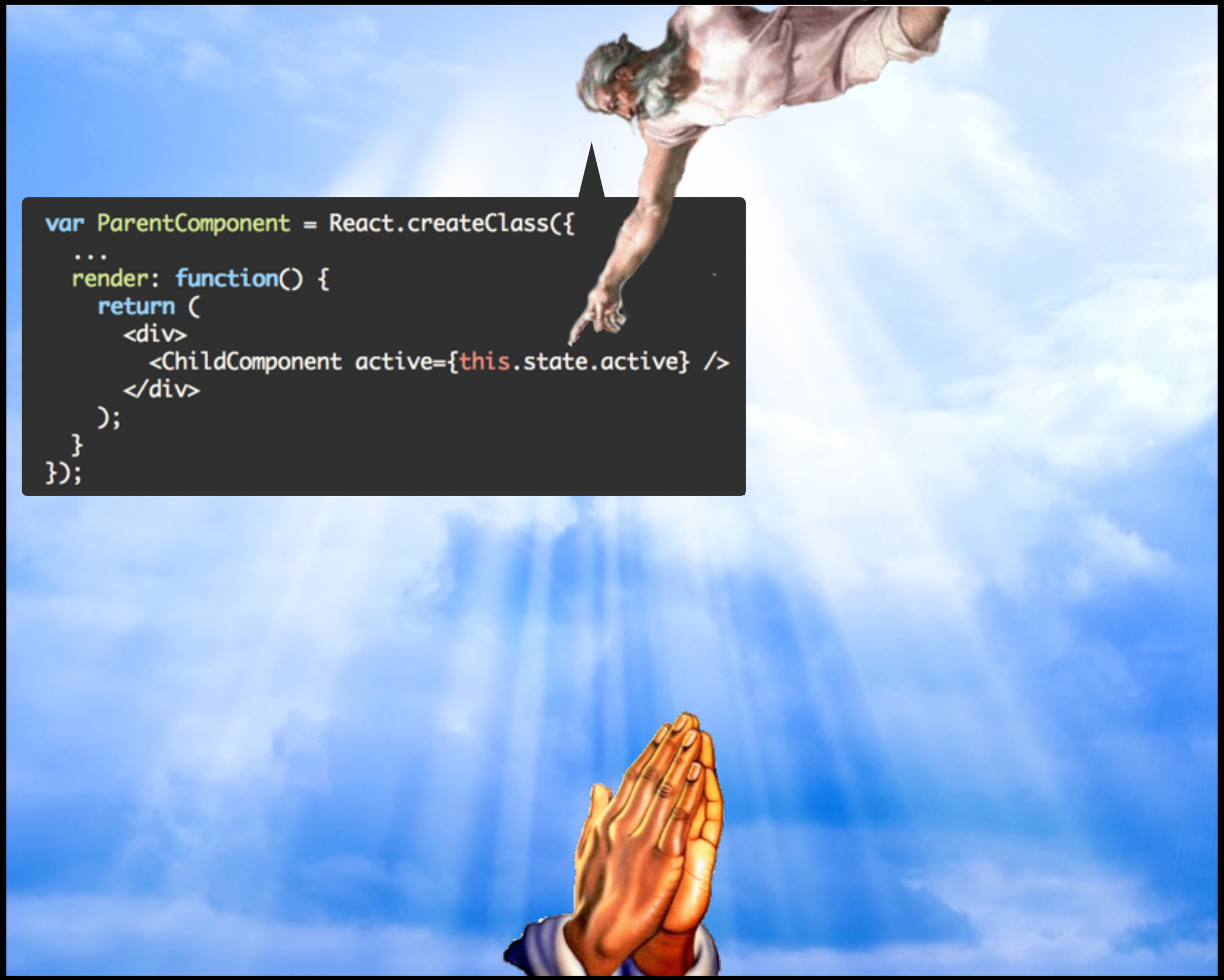
```
var ChildComponent = React.createClass({
  ...
  makeActive: function() {
    this.setState({ active: true });
  },

  render: function() {
    return <input type="checkbox" checked={this.state.active} />
  }
});
```

State is mutable


```
var ChildComponent = React.createClass({
  ...
  makeActive: function() {
    this.setState({ active: true });
  },

  render: function() {
    return <input type="checkbox" checked={this.state.active} />
  }
});
```

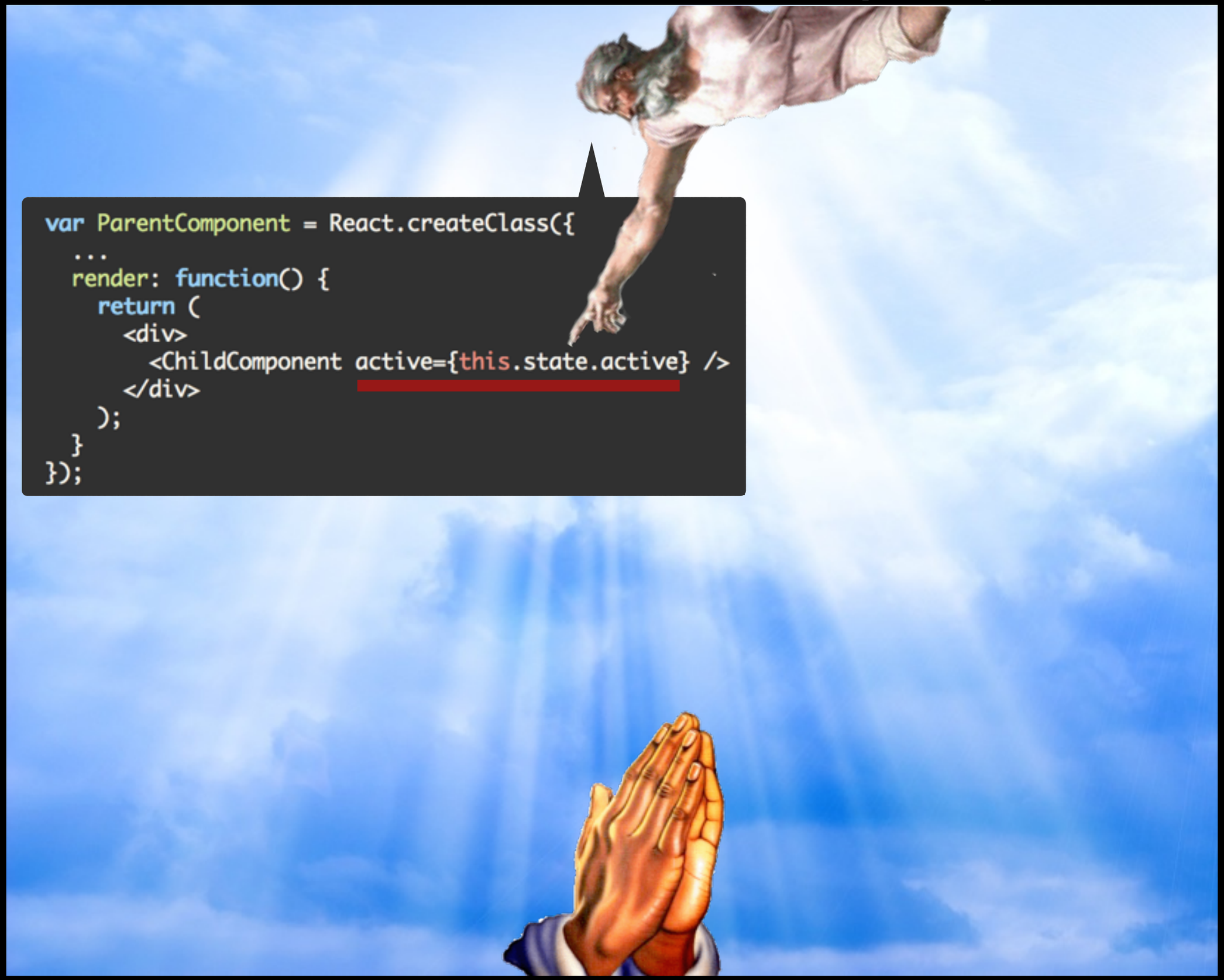
State can become props

State can become props



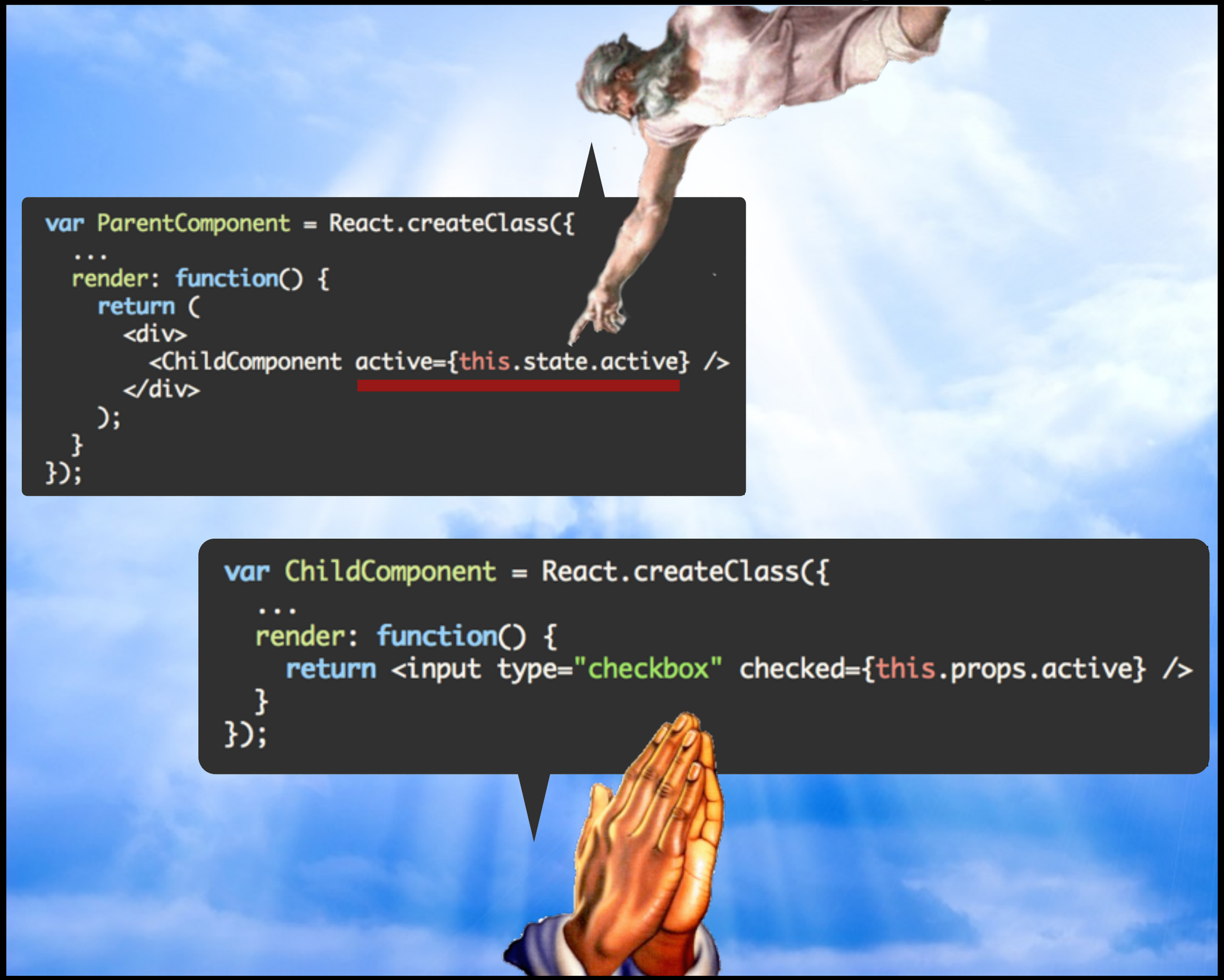
```
var ParentComponent = React.createClass({
  ...
  render: function() {
    return (
      <div>
        <ChildComponent active={this.state.active} />
      </div>
    );
  }
});
```

State can become props



```
var ParentComponent = React.createClass({
  ...
  render: function() {
    return (
      <div>
        <ChildComponent active={this.state.active} />
      </div>
    );
  }
});
```

State can become props



```
var ParentComponent = React.createClass({
  ...
  render: function() {
    return (
      <div>
        <ChildComponent active={this.state.active} />
      </div>
    );
  }
});
```

```
var ChildComponent = React.createClass({
  ...
  render: function() {
    return <input type="checkbox" checked={this.props.active} />
  }
});
```

State can become props

```
var ParentComponent = React.createClass({
  ...
  render: function() {
    return (
      <div>
        <ChildComponent active={this.state.active} />
      </div>
    );
  }
});
```

```
var ChildComponent = React.createClass({
  ...
  render: function() {
    return <input type="checkbox" checked={this.props.active} />
  }
});
```

Data only flows one way

Data only flows one way

Sure, sounds good...

Data only flows one way

Sure, sounds good...

(5 minutes later) Wait!

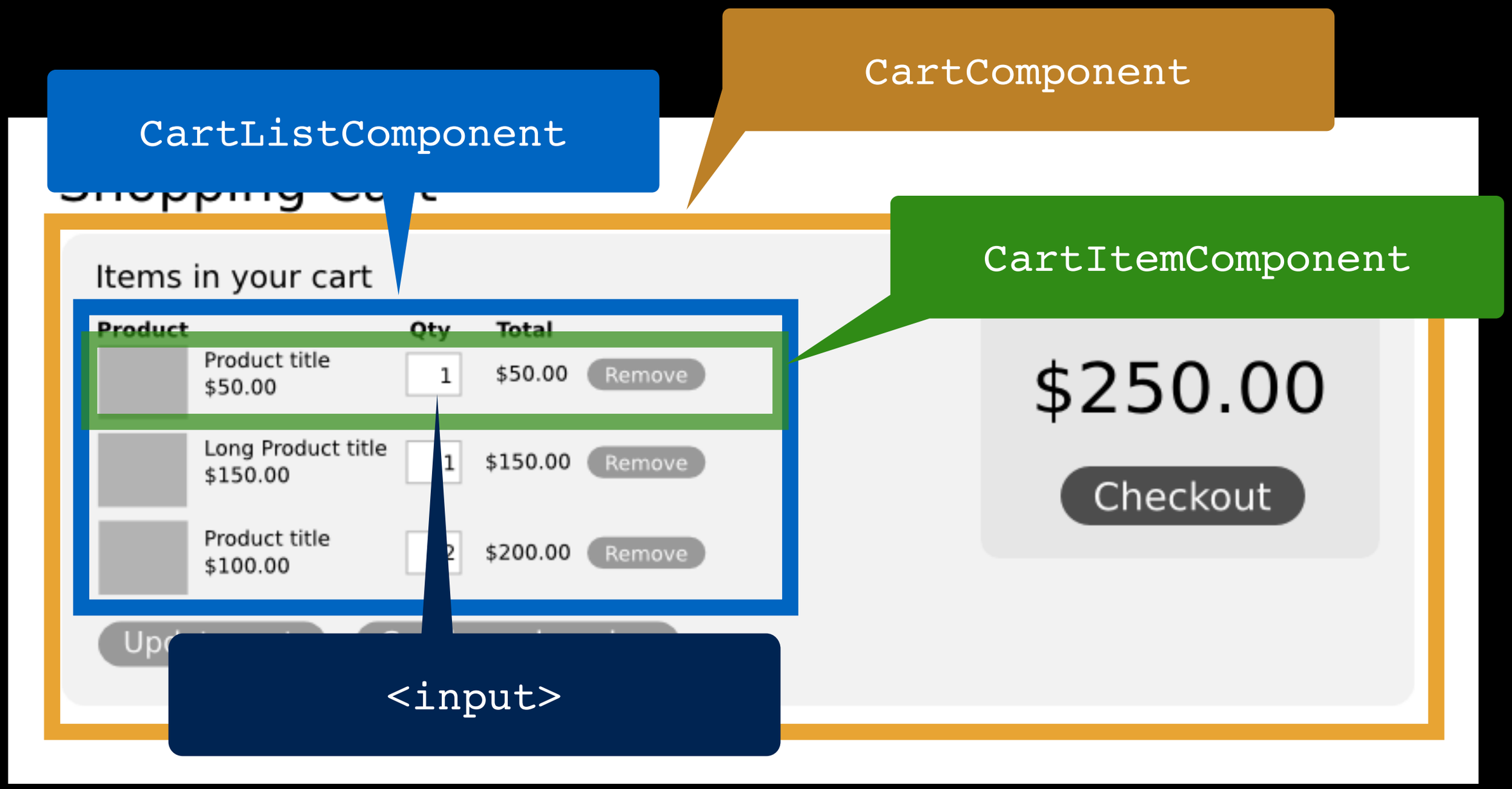
Data only flows one way

Sure, sounds good...

(5 minutes later) Wait!

That’s not how the real world works!

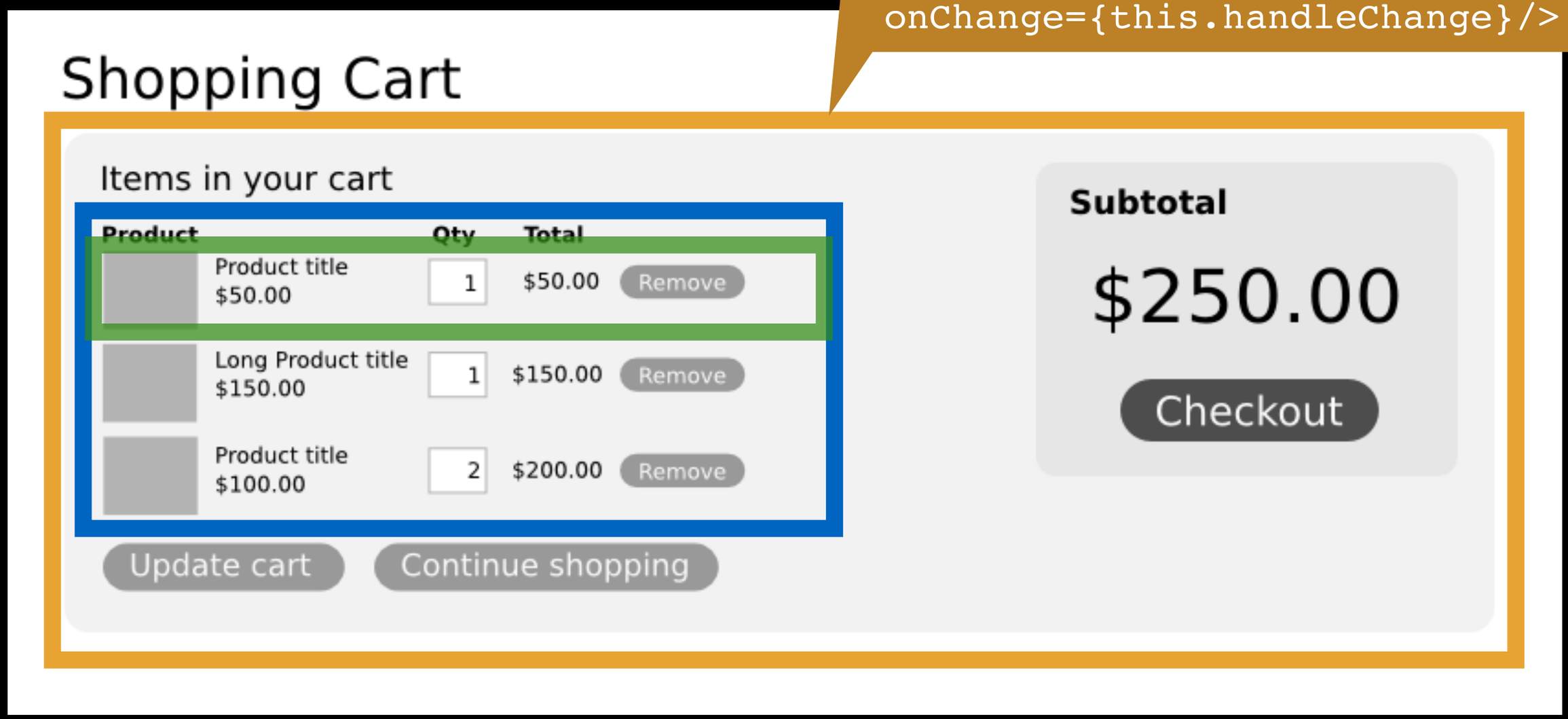
Data flows up?



Data flows down with handlers

CartComponent

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



Data flows down with handlers

CartListComponent

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

CartComponent

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

Items in your cart

Product	Qty	Total	
Product title $50.00	1	$50.00	Remove
Long Product title $150.00	1	$150.00	Remove
Product title $100.00	2	$200.00	Remove

Update cart Continue shopping

Subtotal

$250.00

Checkout

Data flows down with handlers

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	1	$50.00	Remove
Long Product title $150.00	1	$150.00	Remove
Product title $100.00	2	$200.00	Remove

Update cart Continue shopping

Data flows down with handlers

CartListComponent

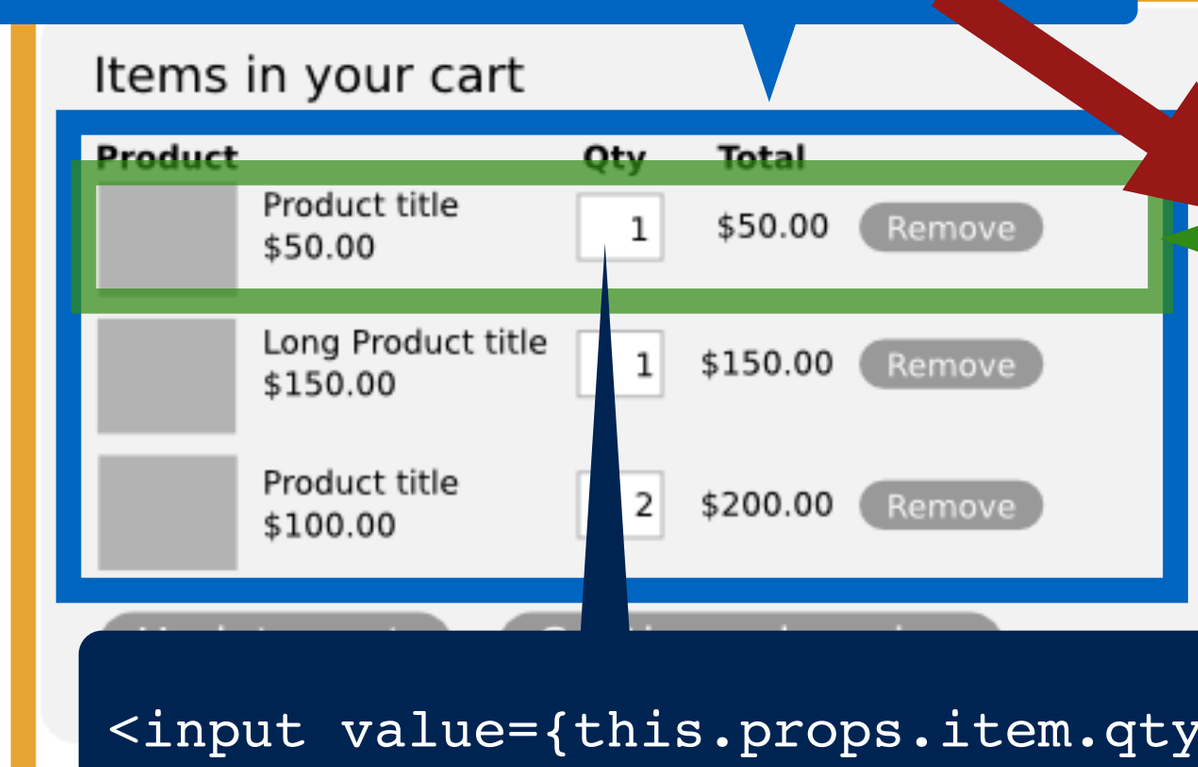
```
this.props.list.map(function(item) {
  <CartItemComponent item={item}
  onChange={this.props.onChange}/>
})
```

CartComponent

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

CartItemComponent

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




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

Data flows down with handlers

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	1	$50.00	Remove
Long Product title $150.00	1	$150.00	Remove
Product title $100.00	2	$200.00	Remove

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

Data flows down with handlers

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	1	$50.00	Remove
Long Product title $150.00	1	$150.00	Remove
Product title $100.00	2	$200.00	Remove

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

Data flows down with handlers

CartListComponent

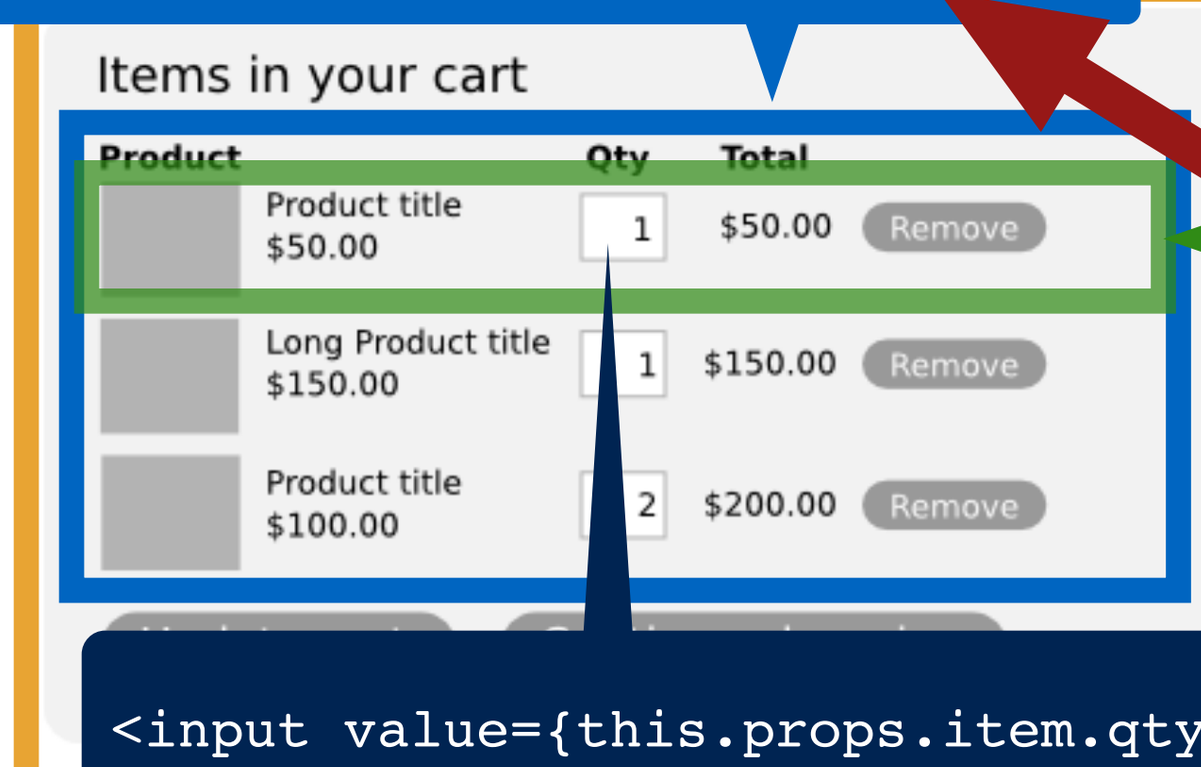
```
this.props.list.map(function(item) {
  <CartItemComponent item={item}
  onChange={this.props.onChange}/>
})
```

CartComponent

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

CartItemComponent

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




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

Data flows down with handlers

CartListComponent

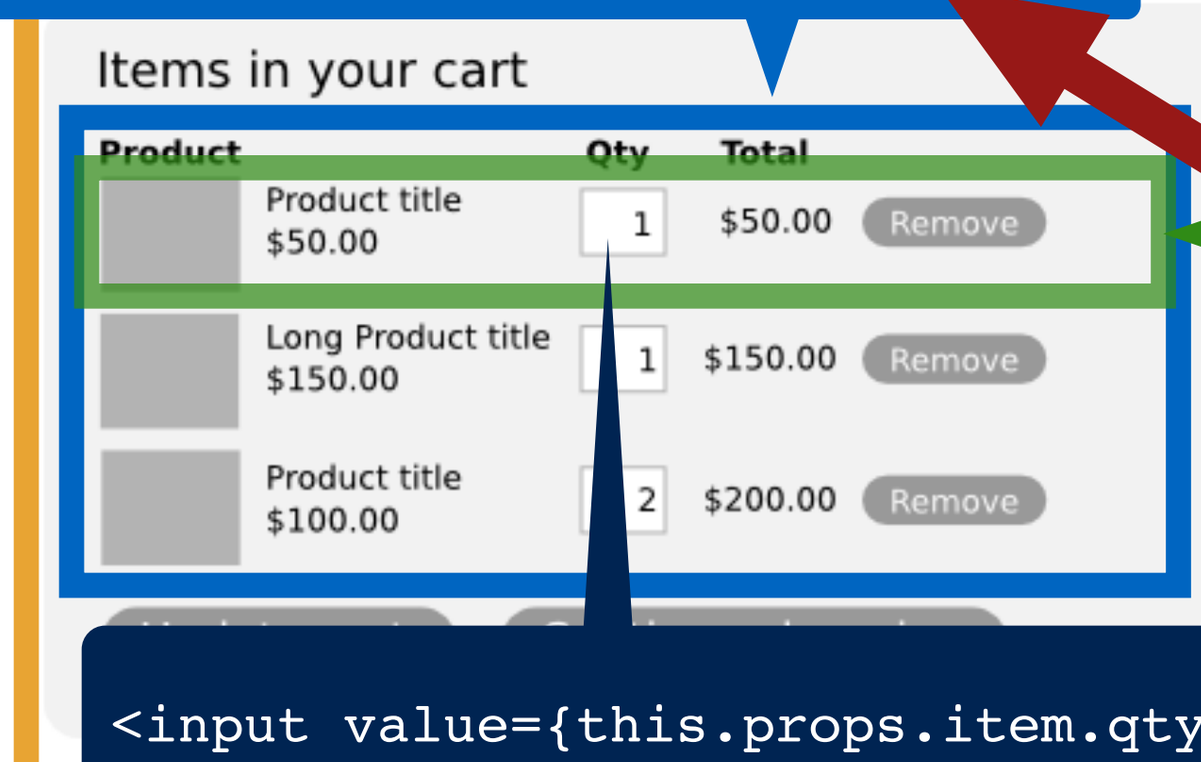
```
this.props.list.map(function(item) {
  <CartItemComponent item={item}
  onChange={this.props.onChange}/>
})
```

CartComponent

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

CartItemComponent

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




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

Events flow up, data flows down

Events flow up, data flows down

Does this sound familiar?

Events flow up, data flows down

Does this sound familiar?
Just like the DOM.

#3
Virtual DOM

What’s worse than having state in two places at once?

What’s worse than having state in two places at once?

Having state in the DOM.

Touching the DOM is evil

Touching the DOM is evil

- It’s inconsistent

Touching the DOM is evil

- It’s inconsistent
- It’s hard to test

Touching the DOM is evil

- It’s inconsistent
- It’s hard to test
- It’s brittle

Touching the DOM is evil

- It’s inconsistent
- It’s hard to test
- It’s brittle
- It’s EXPENSIVE!

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>
    );
  }
});
```

JSX Compiled

```
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."
      )
    );
  }
});
```

It's just Javascript

```
var CartListComponent = React.createClass({

  render: function() {
    return (
      <ul className="cartlist">
        <li className="cartlist-header">
          <span className="col1">Product</span>
          <span className="col2">Qty</span>
          <span className="col3">Total</span>
        </li>
        {this.props.list.map(function(item) {
          return (
            <li>
              <CartItemComponent key={item.id} item={item}/>
            </li>
          );
        })}
      </ul>
    );
  }
});
```

It's just Javascript

```
var CartListComponent = React.createClass({

  render: function() {
    var items = this.props.list.map(function(item) {
      return (
        <li><CartItemComponent key={item.id} item={item}/></li>
      );
    });

    return (
      <ul className="cartlist">
        <li className="cartlist-header">
          <span className="col1">Product</span>
          <span className="col2">Qty</span>
          <span className="col3">Total</span>
        </li>
        {items}
     </ul>
    );
  }
});
```

Why learn yet another template language?

3

2

i want to use ng-repeat in Angular, while i only want to output some elements of the array. something like:

```
ng-repeat"item in items | filter:($index%3 == 0)"
```

but absolutely it cannot work. please tell me how to do this, only output exact index of elements.

javascript angularjs ng-repeat

share | improve this question

asked Dec 31 '13 at 13:49
Frankjs
42 1 7

add a comment

6 Answers

active oldest votes

10

In your code, filter apply on 'items' array, not on each array item, that's why it does not work as you expect.

Instead, you can use ng-show (or ng-if):

```
<ul>
    <li ng-repeat="item in items" ng-show="$index % 3 == 0">{{item}}</li>
</ul>
```

What’s actually happening in render() ?

```
render: function() {
  return <p>Hello {this.props.world}</p>
}
```

What’s actually happening in render() ?

```
render: function() {
  return <p>Hello {this.props.world}</p>
}
```

What does p() return?

What’s actually happening in render() ?

```
render: function() {
  return <p>Hello {this.props.world}</p>
}
```

What does p() return?

When do I call render()

Virtual DOM

Virtual DOM

- It’s a pure Javascript, in-memory representation of the DOM

Virtual DOM

- It’s a pure Javascript, in-memory representation of the DOM
- render() fires whenever something changes

Virtual DOM

- It’s a pure Javascript, in-memory representation of the DOM
- render() fires whenever something changes
- React modifies the real DOM to match

Virtual DOM

- It’s a pure Javascript, in-memory representation of the DOM
- render() fires whenever something changes
- React modifies the real DOM to match
- It’s FAST

Virtual DOM

- It’s a pure Javascript, in-memory representation of the DOM
- render() fires whenever something changes
- React modifies the real DOM to match
- It’s FAST
- It’s pure

Virtual DOM

- It’s a pure Javascript, in-memory representation of the DOM
- render() fires whenever something changes
- React modifies the real DOM to match
- It’s FAST
- It’s pure
- It just works

… Except when it doesn’t

… Except when it doesn’t

How do I access the actual DOM?

… Except when it doesn’t

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

… Except when it doesn't

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

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

… Except when it doesn't

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() });
}
```

… Except when it doesn't

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() });
}
```

Actual DOM Node

Yay! Declarative Templates!

Run jQuery code after AngularJS completes rendering HTML

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

});
```

Declarative Templates suck!

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');
});


directive('heightStuff', ['$timeout', function ($timeout) {
    return {
        link: function ($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);
            })
        }
    };
}]);
```

Declarative Templates suck!

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');
});


directive('heightStuff', ['$timeout', function ($timeout) {
    return {
        link: function ($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.

Declarative Templates suck!

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');
});


directive('h                                    ion ($timeout) {

                ion ($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);
            })
        }
    };
}]);
```

Run jQuery code after AngularJS completes rendering HTML

wut.

Declarative Templates suck!

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');
});


directive('h                                    ion ($timeout) {

                ion ($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);
            })
        }
    };
}]);
```

Run jQuery code after AngularJS completes rendering HTML

wut.

Why all this is awesome

Why all this is awesome

- One-way data flow keeps complexity under control

Why all this is awesome

- One-way data flow keeps complexity under control
- Easy to debug self-contained components

Why all this is awesome

- One-way data flow keeps complexity under control
- Easy to debug self-contained components
- Library doesn’t dictate too much

Why all this is awesome

- One-way data flow keeps complexity under control
- Easy to debug self-contained components
- Library doesn’t dictate too much
- Ridiculous potential

A small demo

http://www.emergent.info/iphone-hairgate

Where to go from here?

Where to go from here?

Official React docs

http://facebook.github.io/react/

Where to go from here?

Official React docs

http://facebook.github.io/react/

TodoMVC

git@github.com:tastejs/todomvc.git

Thank You