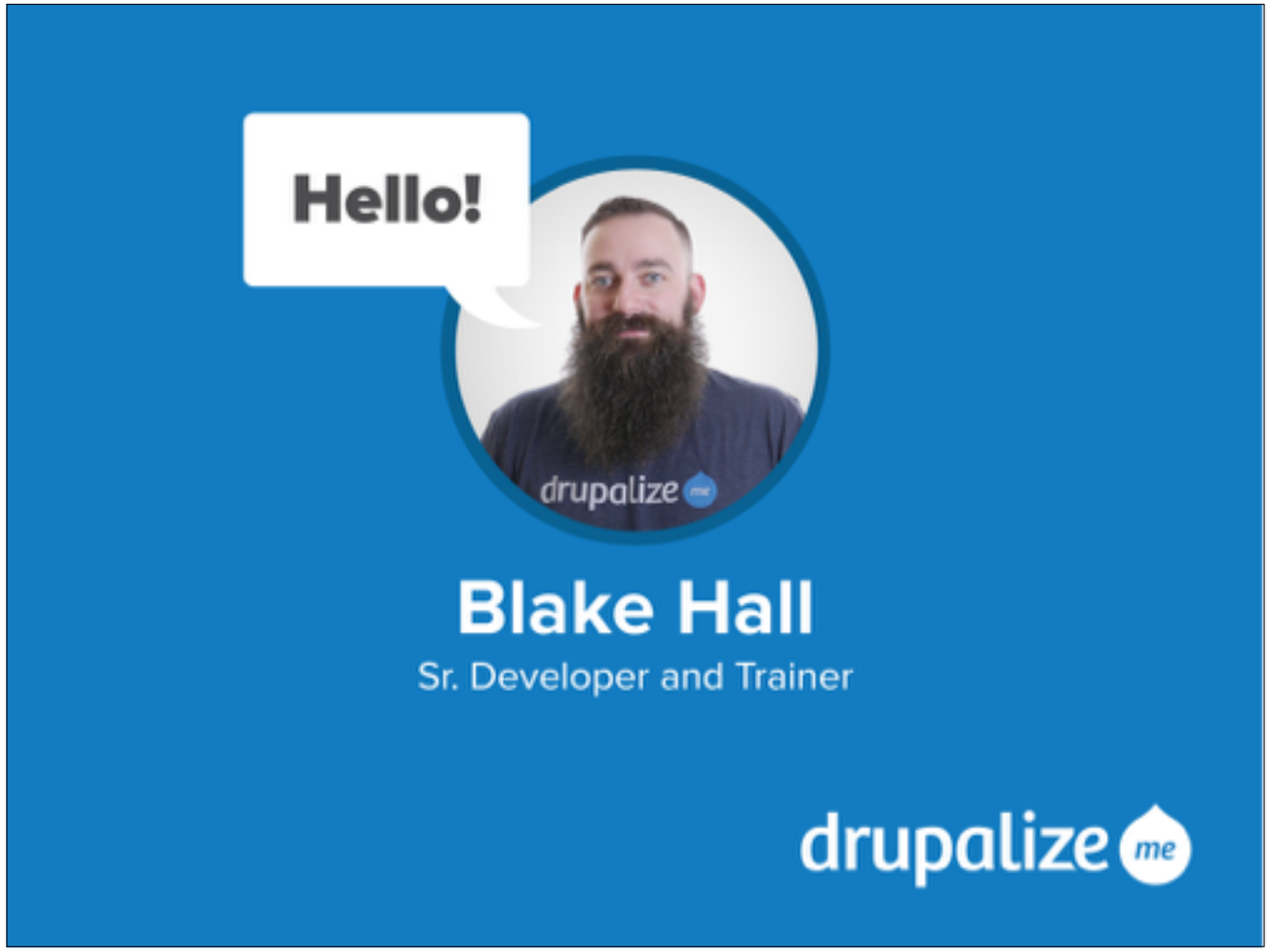
What’s the fuss with all this

Welcome!
We’ve got a lot to cover, so let’s get going.



Hi, I'm Blake.
I'm a Developer and Trainer on the Drupalize.Me team.
We have the largest collection of Drupal training anywhere, check us out.

Lullabot

Lullabot is an interactive strategy, design, and development company.

GAME PLAN

- What's the problem?
- Backbone.js
- Angular.js
- React.js
- Questions?

drupalize me

I spent a bunch of time last year attending events in the node & javascript community.
I just have to plug a couple of them quickly, because a tech conference with camping is AWESOME.

The last night at JS.Conf, our party included a bouncy castle, dunk tank, and I got to put on those giant boxing gloves and punch Eric Duran…

It was a good time.

so what?

Okay, what’s this have to do with anything?

Marco Rogers

polotek

Finding Patterns Across Front-end Frameworks

We're all building a client-side framework. And they're all different implementations of the same stuff. I want to compare the approaches that popular frameworks take to solving these patterns. If they even try to solve them at all. I also want to include info about the custom framework that Yammer uses. It'll be illustrative to see how all of these have different approaches to the same set of patterns.

Meet Marco

Marco is an engineer who loves building things for the web. He's been doing server-side javascript since before it was cool (and before it was node). But he spends most of his time in the browser these days as Front-end Engineering Lead at Yammer. You can find him as @polotek almost everywhere online. He also tweets occasionally.

Ryan Florence

ryanflorence

Embularactymerbone

Ember, Angular, React, Polymer, and Backbone. Following in jQuery's footsteps, these projects (and others) are helping to drive some future APIs in the browser.

While they all get lumped into the category "Frontend MVC", an intimate look at each reveals they are quite different. These stark and subtle differences matter when choosing one for your project.

In this session, you'll see each project's sweet spot, and where each struggles. You'll get a healthy dose of code as we explore the primary APIs. You will walk away with a better understanding of the goals and intended use-cases for each project.

Meet Ryan

I'm a software engineer from Salt Lake City, Utah. I have been creating websites since the early 90's and currently work as a software engineer @instructure

A couple of sessions either inspired this one,
or provided the source material to shamelessly steal from.

Both of these sessions are on youtube, and I definitely recommend checking them both out.

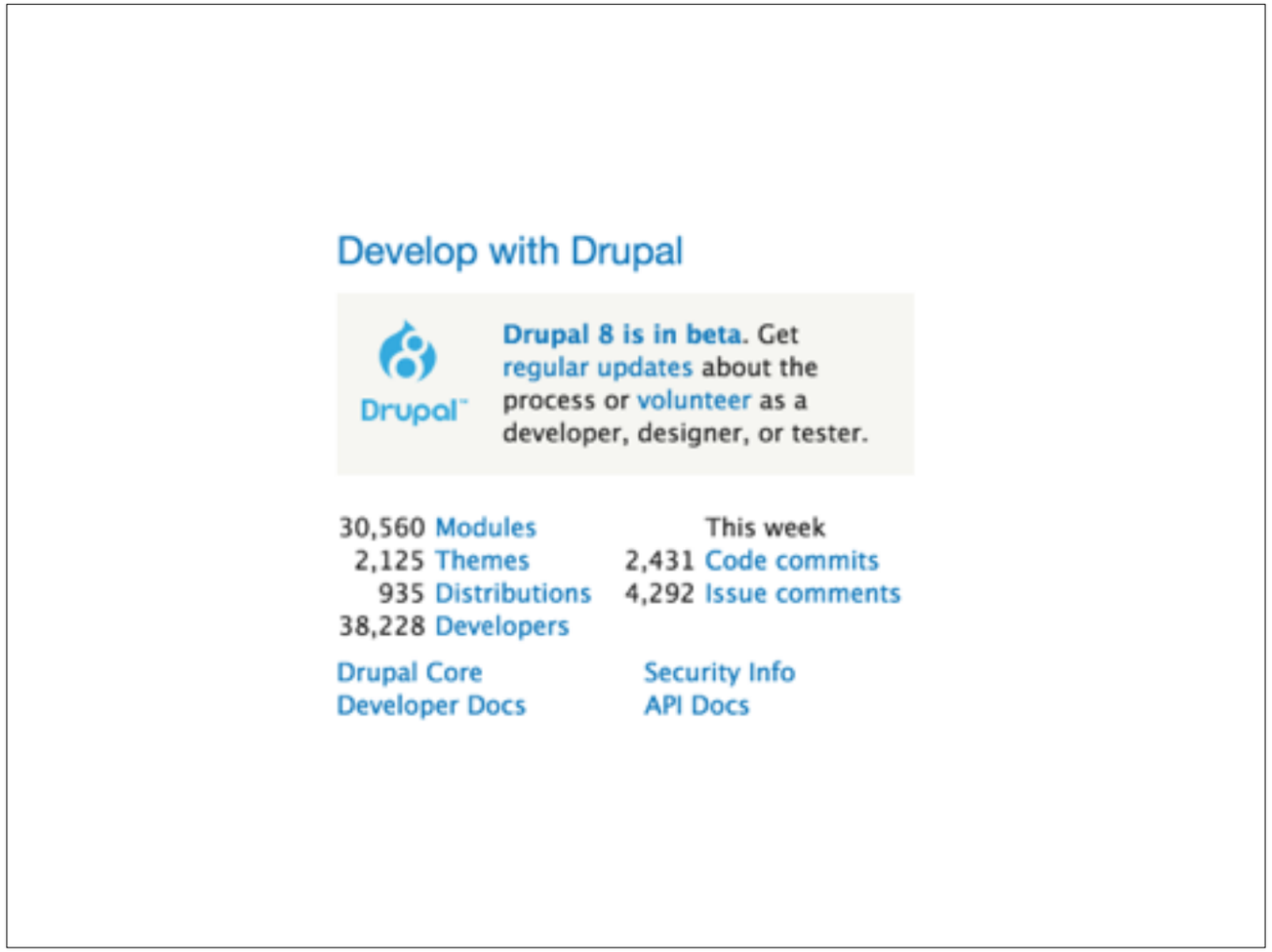
https://drupalize.me/blog/201504/backbonejs-and-underscorejs-drupal-8

Backbone.js and Underscore.js in Drupal 8

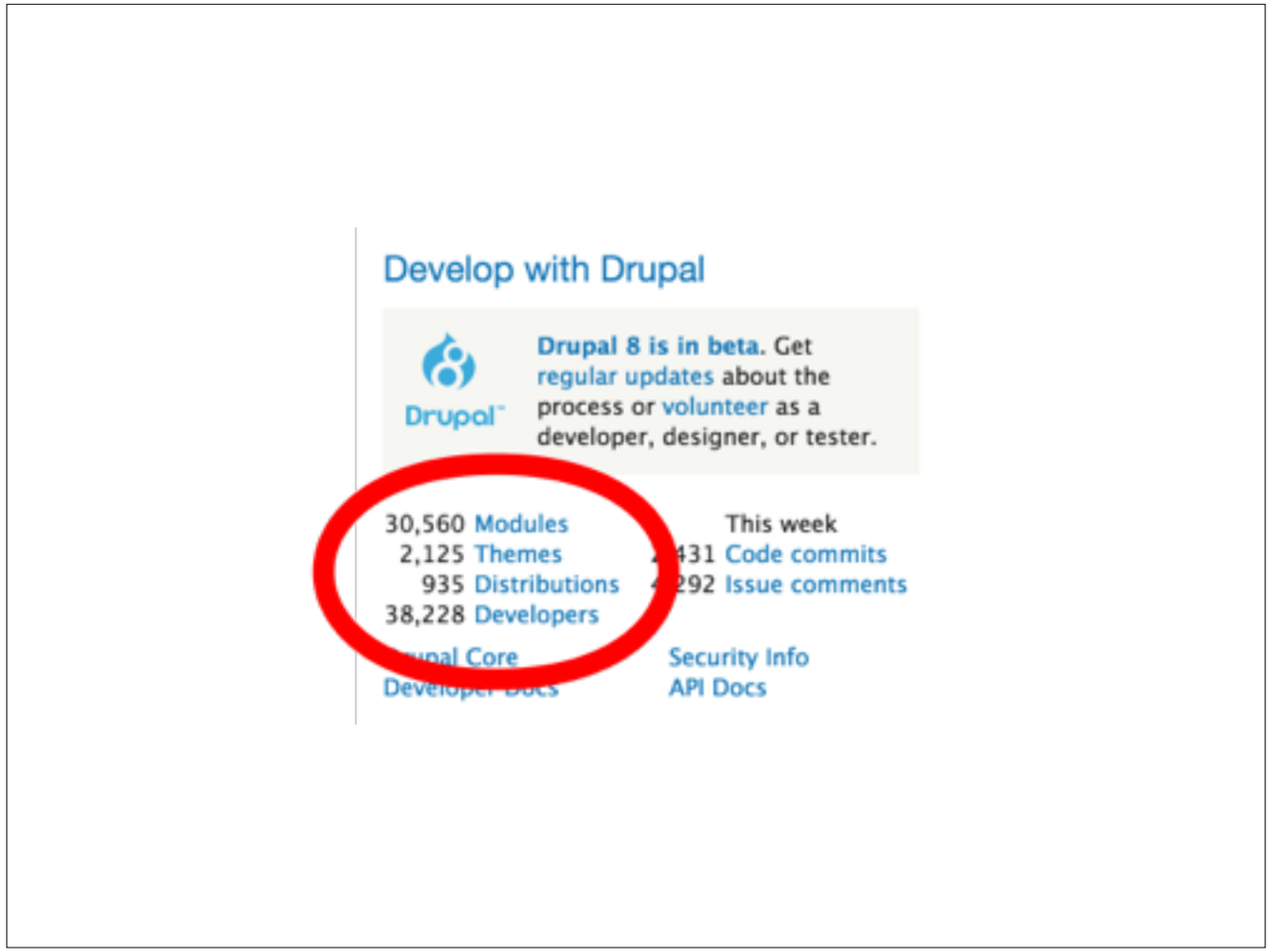
Posted on April 7, 2015 by Blake Hall

The Drupal 8 development cycle has definitely been a long one. There are several exciting features on the way, but the improvements to the authoring experience in Drupal 8 have definitely drawn a lot of attention. (I know Amber is clamoring for in-place editing for this blog.) The Spark project is the home to much of this work. Several

Meanwhile in the Drupal community…
I wrote a blog post about this about a month ago, but Backbone was committed to Drupal 8 ~3 years ago now!
So, like when jQuery was added, we have a new tool to take advantage of.
Front-end development moves fast though, How does this compare with newer frameworks that have hit the scene?



I think it's important to understand the communities we're talking about here.
I noticed on the carpet this morning that drupal.org launched in 2001.
I started using Drupal almost 10 years ago, at a time where contrib consisted of about 350 modules and it was possible to try them all.



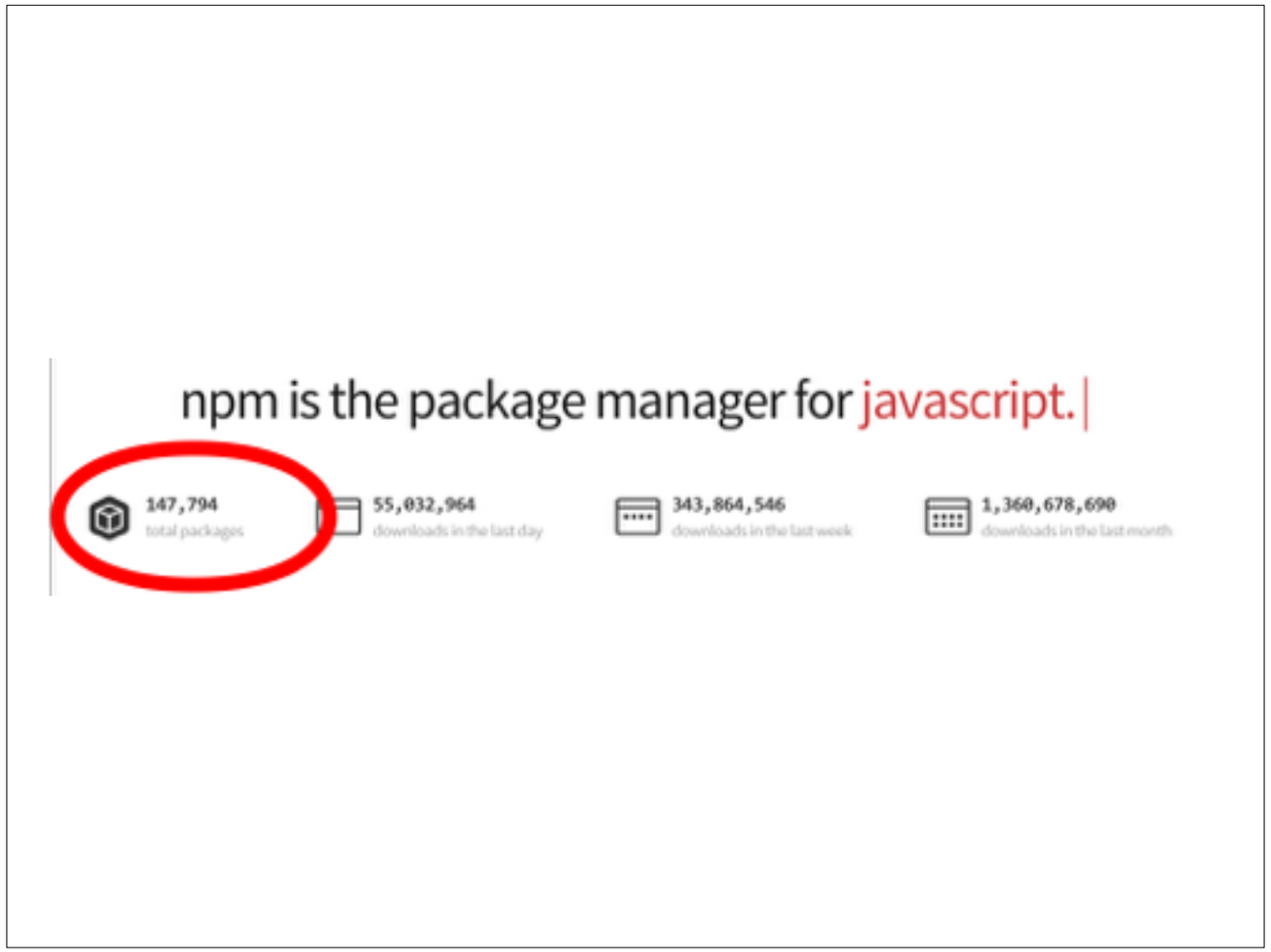
This screen cap was taken yesterday, and as a community we're just over 30K Modules!

It's worth noting that this number includes sandbox projects, but still…

I don’t know about you, but that blew my mind a bit.

and we’re just under 17K full projects

Meanwhile, in 2009 npm was released and has became the de-facto home for javascript libraries and frameworks.



147K packages…
Since 2009.

In 6 years, the javascript community has shared almost 5x as much code. Not apples to apples, but

Wow!

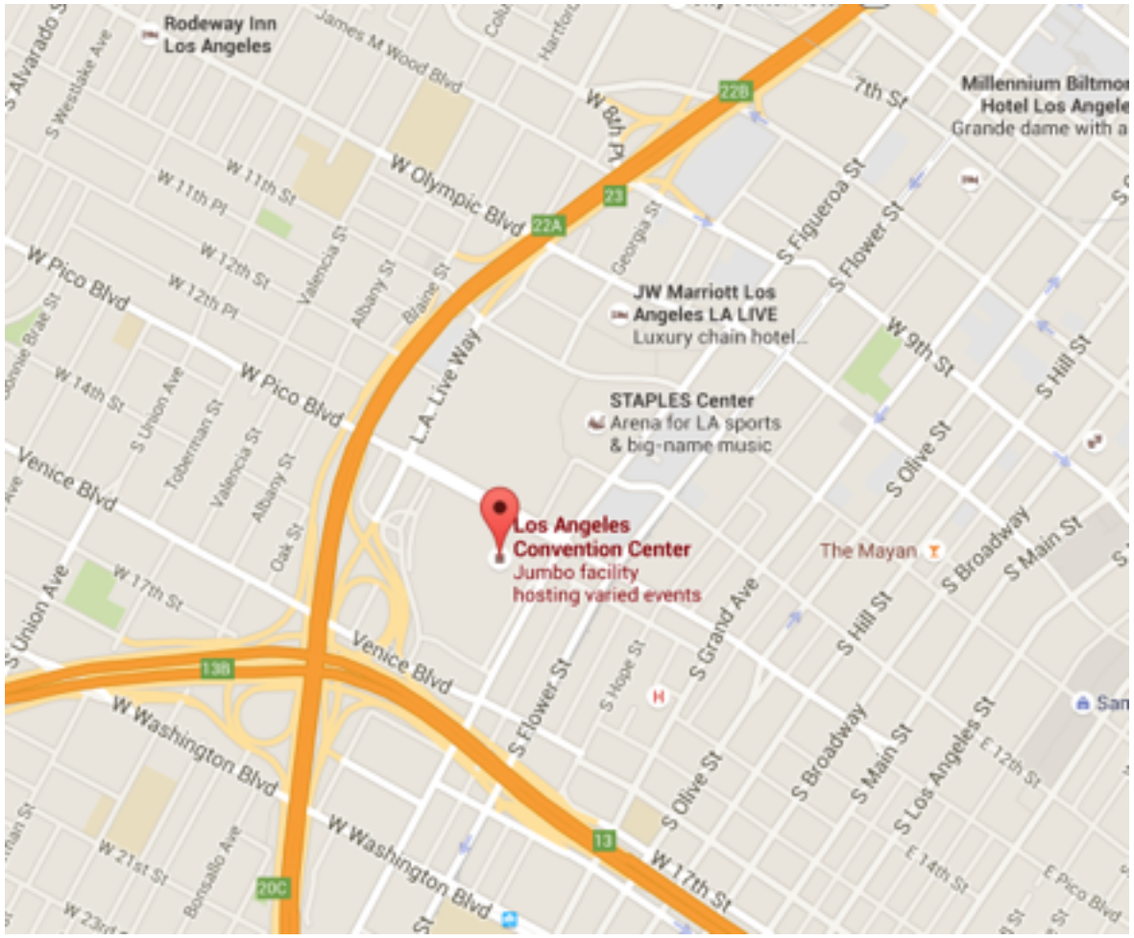
So where did all this come from?

Back in the old days…

(And by old days I mean 1996)

MapQuest launched to help us get directions from the internet.
As far as interactions go, this is where we came from:
a stylized series of forms with full page reloads on every click

Along comes Ajax in 2004/5 based on some work from google maps, gmail



While we're still doing AJAX -> it's often just naive html replacement happening via background requests

it's basically just a "transformed version of a series of forms" happening with background processing.

2010 comes along and backbone is released.
Model-View-Presenter pattern.
Allows developers to use traditional Object Oriented data modeling techniques in the browser to improve code organization
and make it easier to keep the client and server in sync with all the background AJAX requests that are happening in more complex applications.

Now, it’s kind of a running joke how many front-end frameworks their are to choose from.

What's a Framework?

- Models
- Views
- Routing
- RESTful API access

drupalize me

What actually constitutes a framework though?
Some form of data modeling
Some sort of presentation layer
Some way to handle state / urls / navigation
A way to make requests for additional data to display on the page.

todomvc.com is a pretty handy way to compare frameworks and grok what they're trying to do. The code samples we're going to walk through come from this project, and they're linked to and available on github if you want to revisit them later.

Here’s what the app actually looks like. While they’re all identical, this one is actually powered by “vanilla” jQuery.

We’ve using local storage for a todo list, All the basic functionality is here… Add, Delete, Filter, Log, etc…

We've already mentioned it a few times, and since it's in Drupal core and this is Drupalcon Let's start with Backbone.js

“What would the **ideal** [client side] webapp API look like?”

- Jeremy Ashkenas

drupalize me

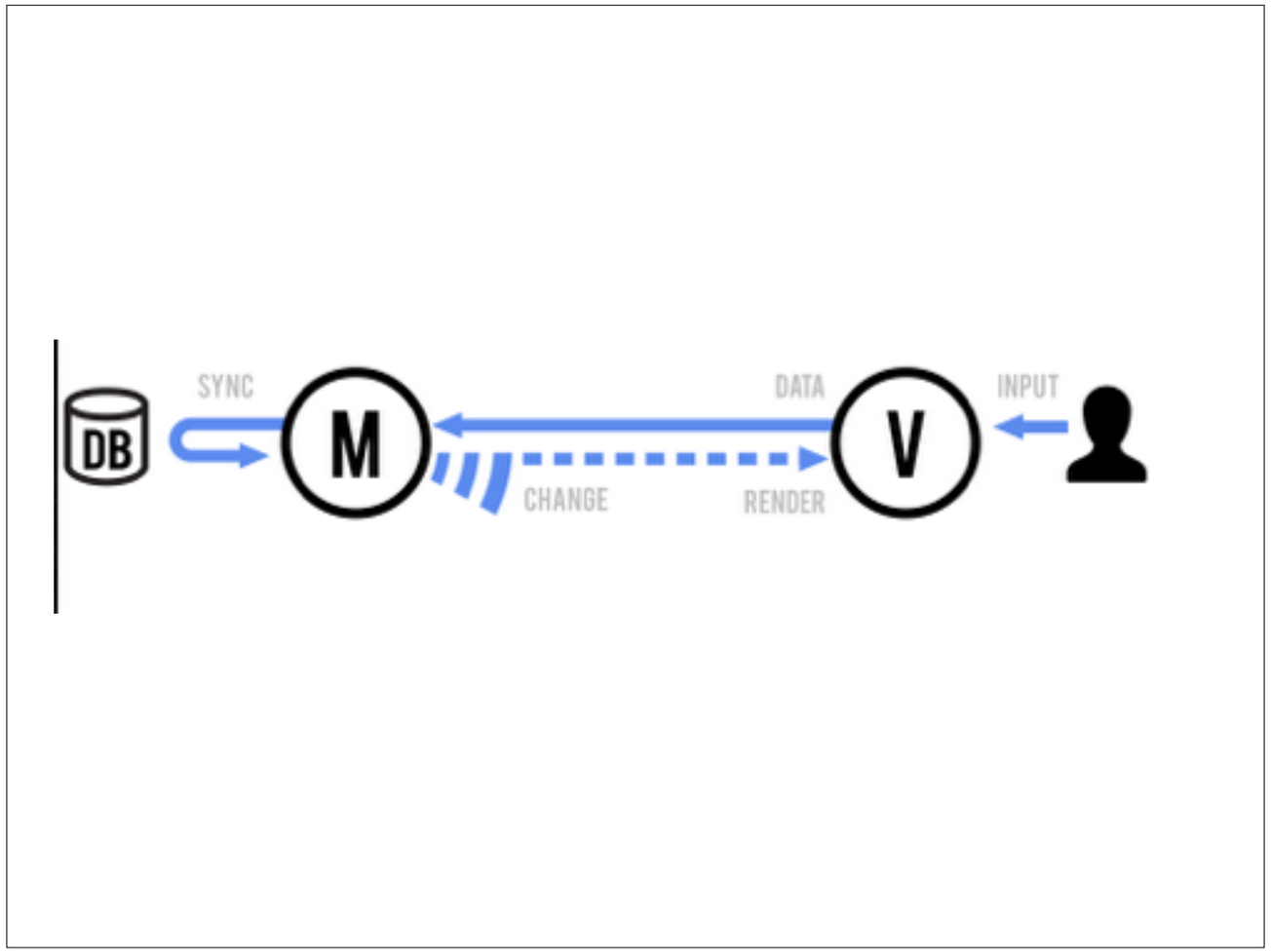
"Get the **truth** out of the DOM"
(finding & parsing data out of the page for business value)
"the DOM is a user interface not a data source”
Backbone.JS - JSConfUY 2014 https://www.youtube.com/watch?v=UAl_N62gKmM

MV*

drupalize me

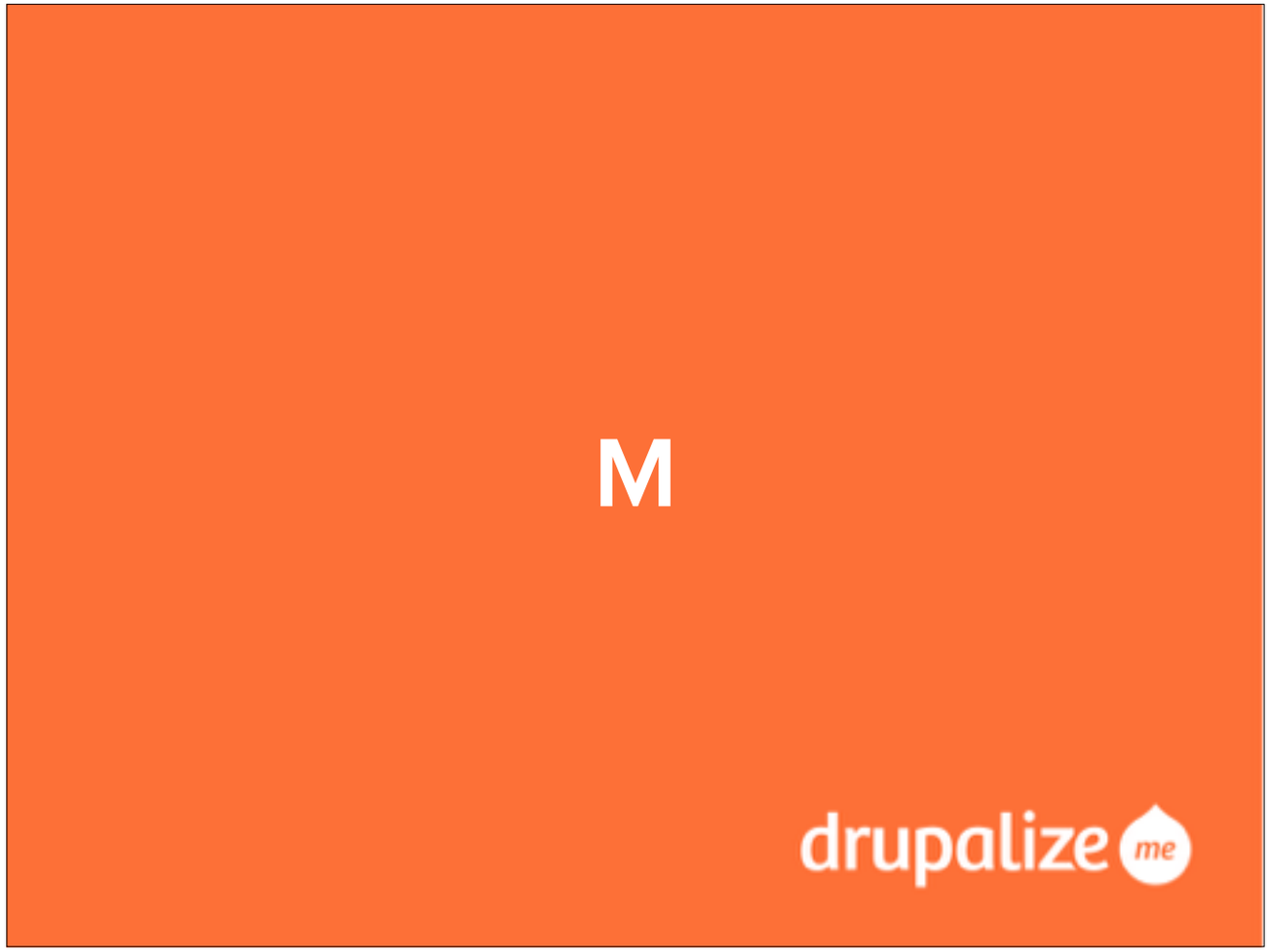
Rather than the (perhaps?) familiar MVC Pattern,
Backbone is based on the Model - View - Presenter paradigm

What exactly does that mean?



User input triggers events from the View class. These bubble up to the model.

Then, when the model changes (perhaps after an API request returns data), that change event bubbles back out to the view layer triggering a re-rendering of the view template.



Let take a look at a Backbone Model in our example Todo app

```
/*global Backbone */
var app = app || {};

(function () {
  'use strict';

  // Todo Model
  // ----------

  // Our basic **Todo** model has `title`, `order`, and `completed` attributes.
  app.Todo = Backbone.Model.extend({
    // Default attributes for the todo
    // and ensure that each todo created has `title` and `completed` keys.
    defaults: {
      title: '',
      completed: false
    },

    // Toggle the `completed` state of this todo item.
    toggle: function () {
      this.save({
        completed: !this.get('completed')
      });
    }
  });
})();
```

Backbone models extend a base class Backbone.Model.
Inside of your new base class you then set up defaults, and add any rich methods needed to provide functionality to the model. OOP 101…

So what does the View layer look like?

```
/*global Backbone, jQuery, _, ENTER_KEY, ESC_KEY */
var app = app || {};

(function ($) {
  'use strict';

  // Todo Item View
  // --------------

  // The DOM element for a todo item...
  app.TodoView = Backbone.View.extend({
    //... is a list tag.
    tagName:  'li',

    // Cache the template function for a single item.
    template: _.template($('#item-template').html()),

    // The DOM events specific to an item.
    events: {
      'click .toggle': 'toggleCompleted',
      'dblclick label': 'edit',
      'click .destroy': 'clear',
      'keypress .edit': 'updateOnEnter',
      'keydown .edit': 'revertOnEscape',
      'blur .edit': 'close'
    },

    // The TodoView listens for changes to its model, re-rendering. Since
    // there's a one-to-one correspondence between a **Todo** and a
    // **TodoView** in this app, we set a direct reference on the model for
    // convenience.
    initialize: function () {
      this.listenTo(this.model, 'change', this.render);
      this.listenTo(this.model, 'destroy', this.remove);
      this.listenTo(this.model, 'visible', this.toggleVisible);
    },
```

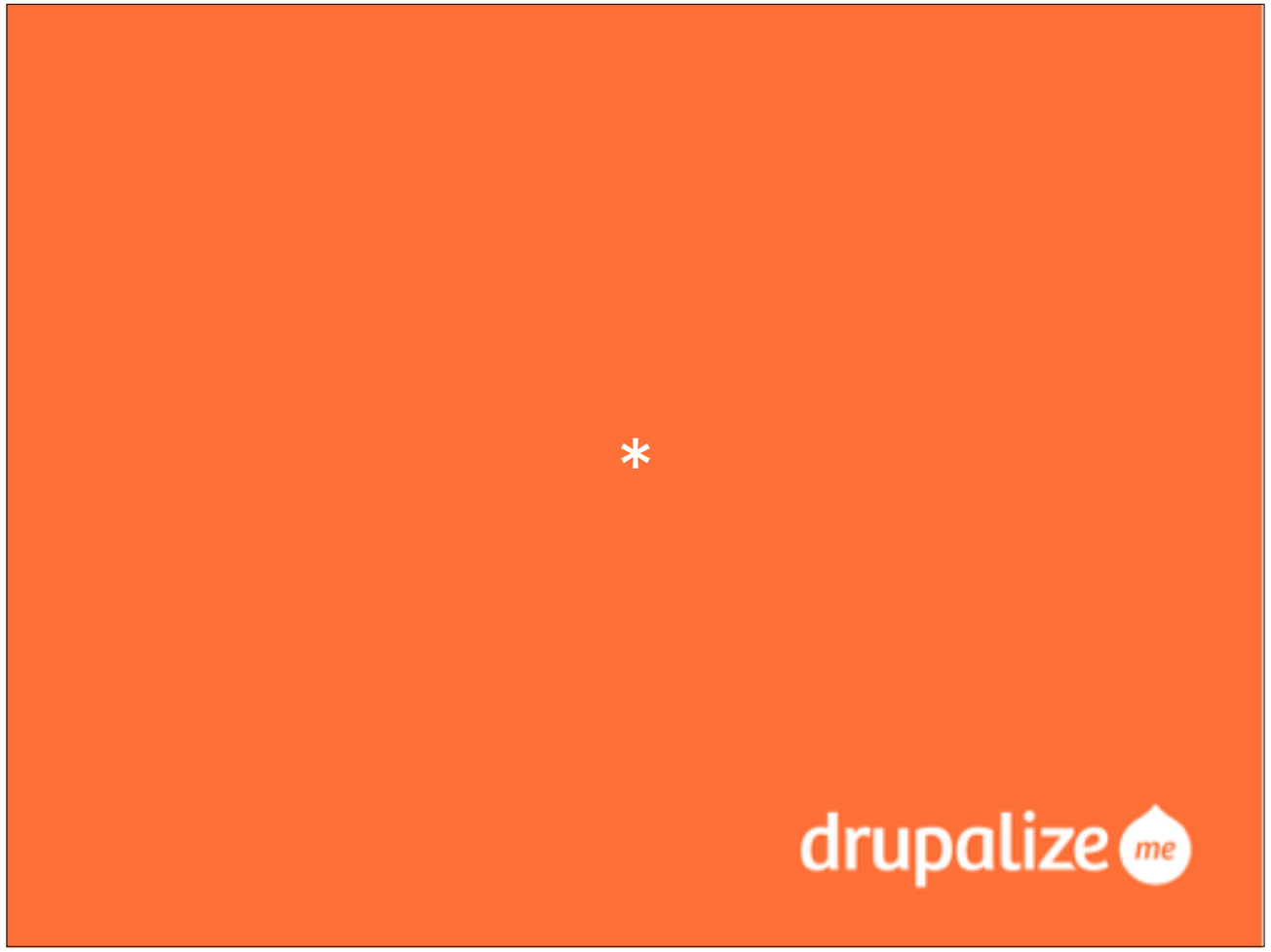
Again, we’re extending a base view class Backbone.View.

We’re passing in the template used to render this view, the DOM events we want to bind to and adding listeners to trigger custom methods based on specific types of user input.

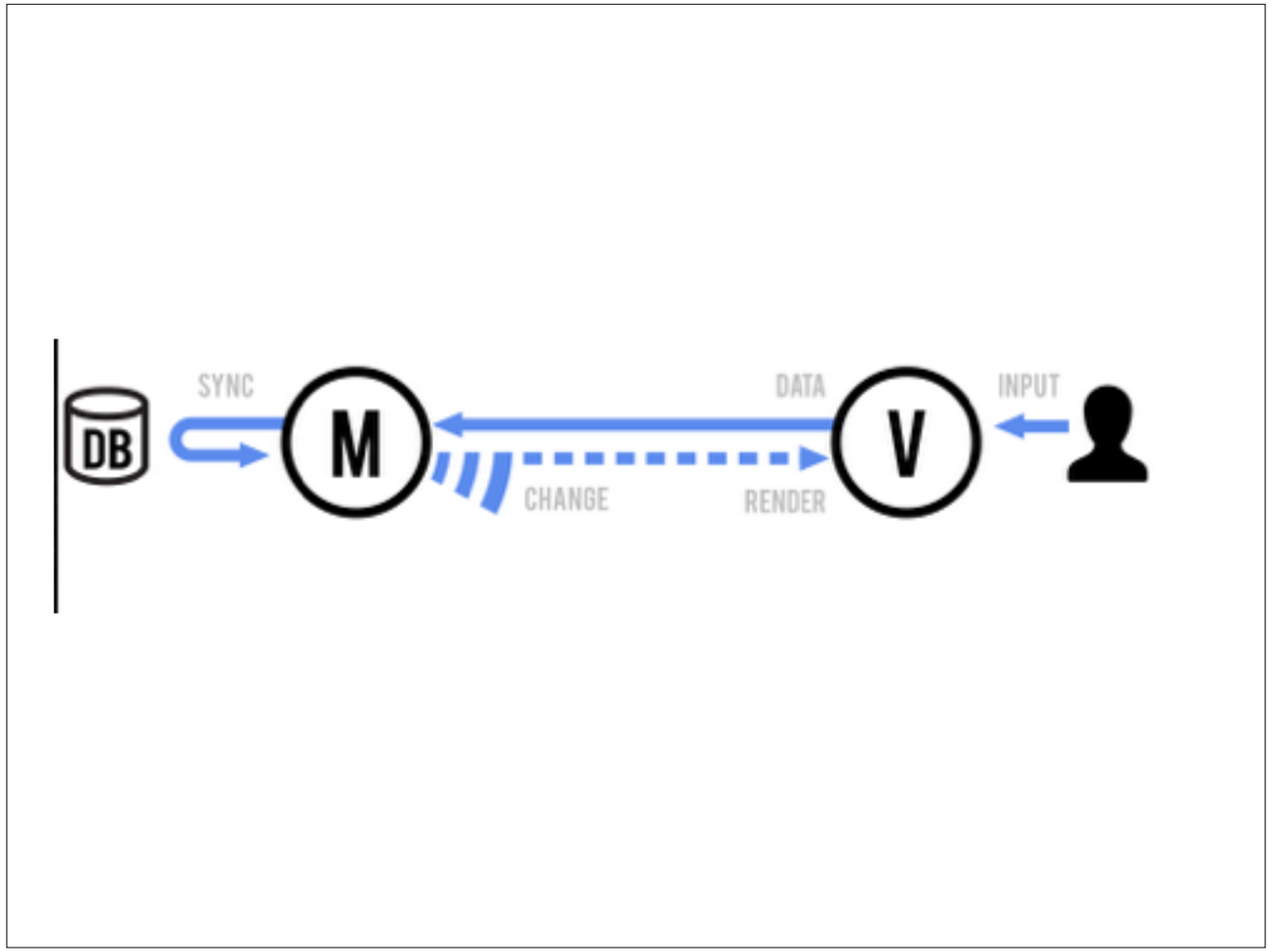
```
    // Re-render the titles of the todo item.
    render: function () {
      // Backbone LocalStorage is adding 'id' attribute instantly after
      // creating a model.  This causes our TodoView to render twice. Once
      // after creating a model and once on 'id' change.  We want to
      // filter out the second redundant render, which is caused by this
      // 'id' change.  It's known Backbone LocalStorage bug, therefore
      // we've to create a workaround.
      // https://github.com/tastejs/todomvc/issues/469
      if (this.model.changed.id !== undefined) {
        return;
      }

      this.$el.html(this.template(this.model.toJSON()));
      this.$el.toggleClass('completed', this.model.get('completed'));
      this.toggleVisible();
      this.$input = this.$('.edit');
      return this;
    },

    toggleVisible: function () {
      this.$el.toggleClass('hidden', this.isHidden());
    },

    isHidden: function () {
      return this.model.get('completed') ?
        app.TodoFilter === 'active' :
        app.TodoFilter === 'completed';
    },

    // Toggle the `"completed"` state of the model.
    toggleCompleted: function () {
      this.model.toggle();
    },
```

We also have to provide a render method since the default implementation is a no-op.
This method renders the view template from model data, and updates this.el with the new HTML.

(as a quick aside, returning this at the end of the method allows for chaining)



Presenter / Controller / Whatever glue pieces are needed to hold things together…

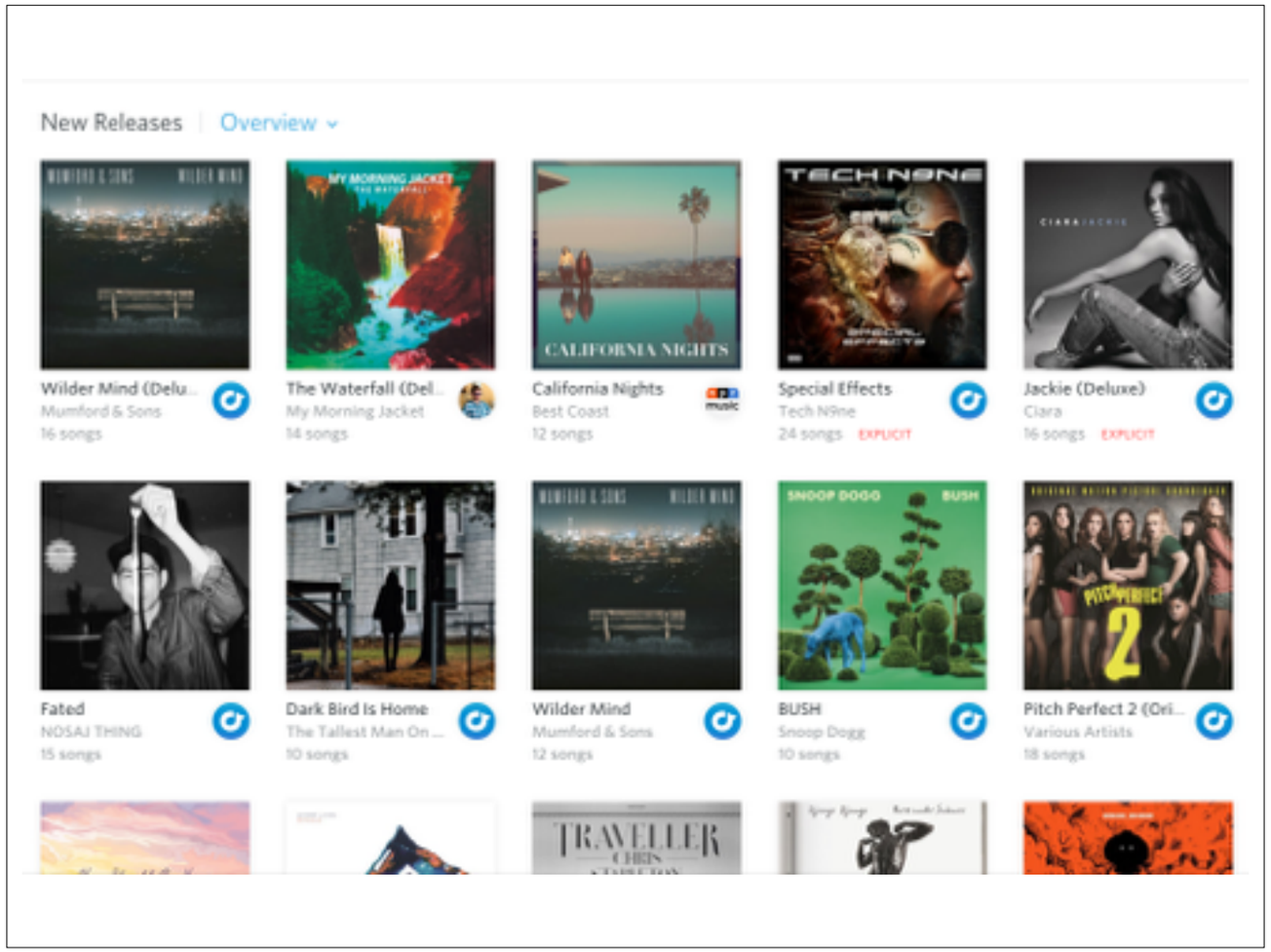


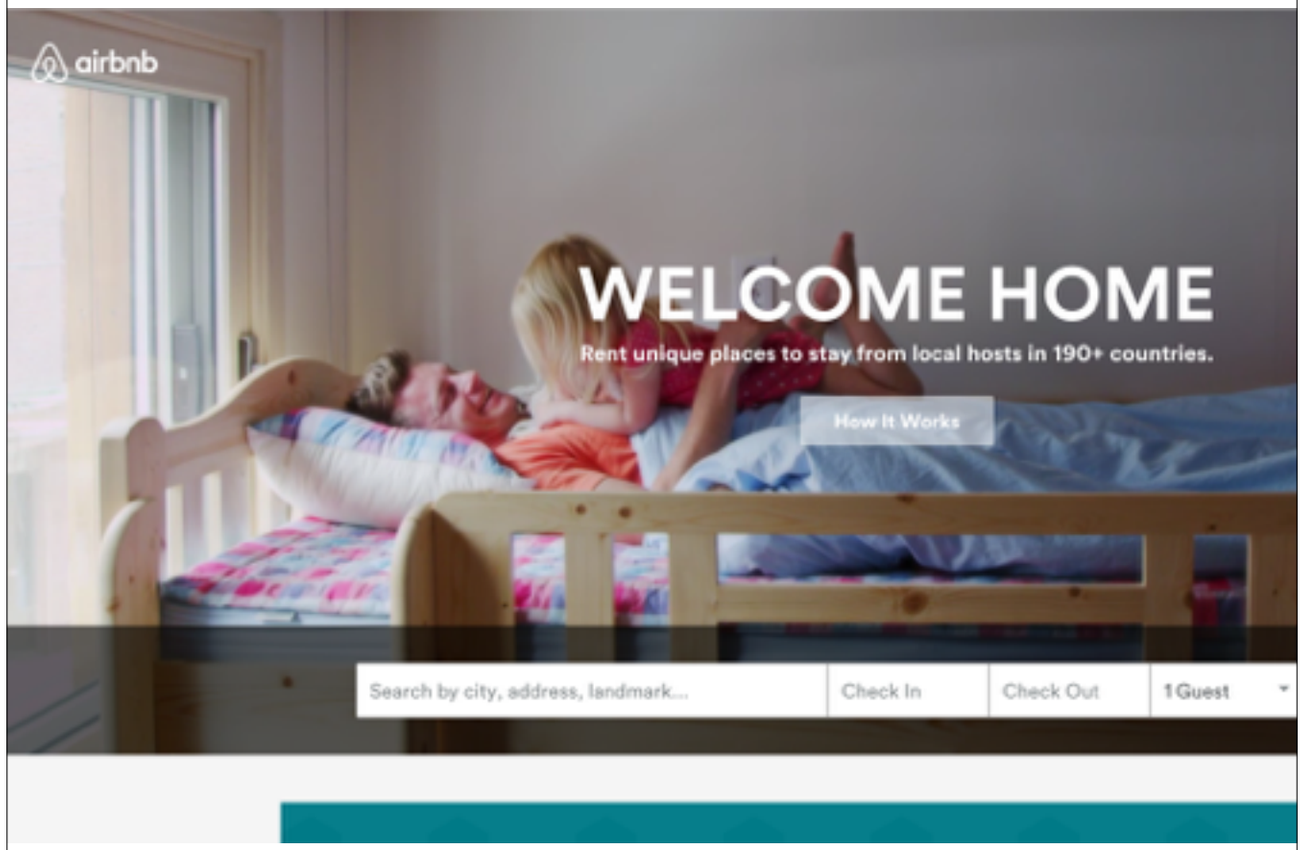
Observer Pattern

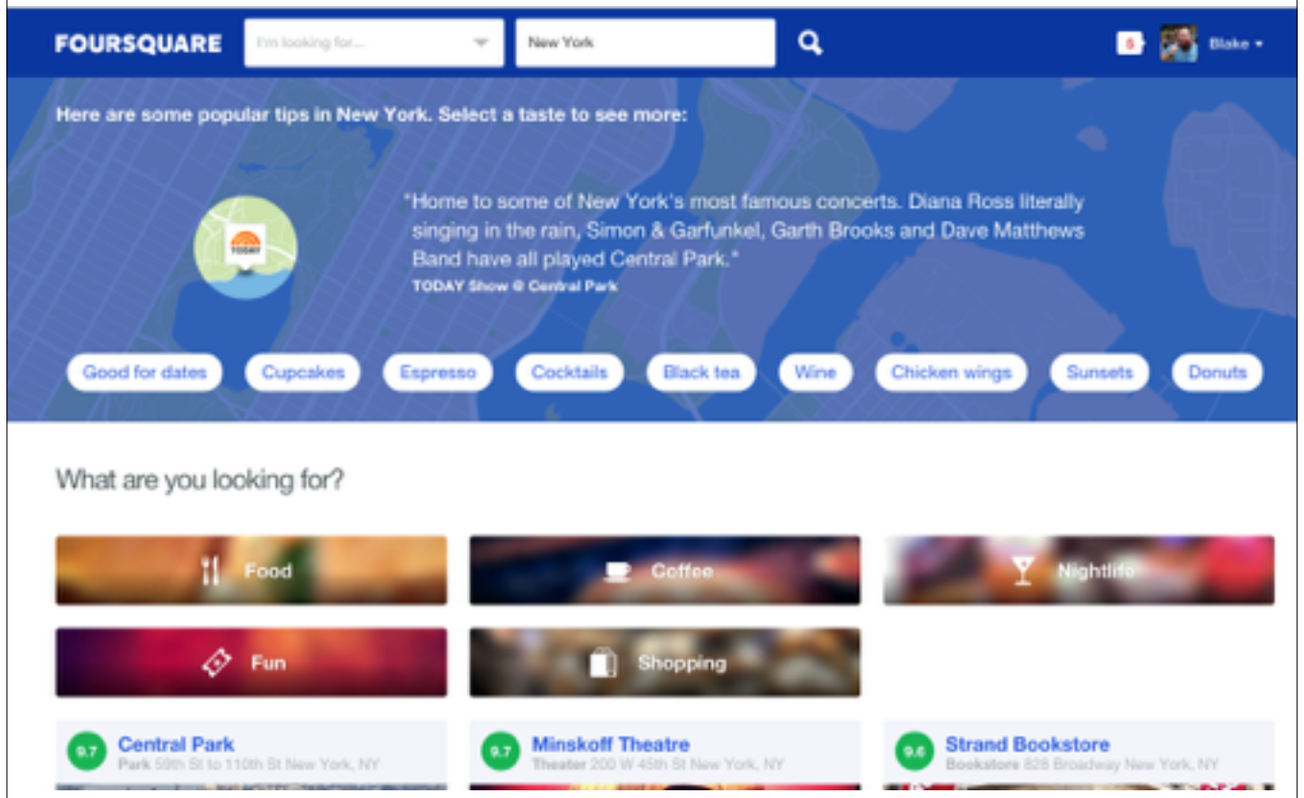
Key value observation -> cascading effects depending on how models & collections are set up.
(we'll revisit this later)

Who uses Backbone.js?

drupalize me


New Releases
Overview
Wilder Mind (Delu...
Mumford & Sons
16 songs
The Waterfall (Del...
My Morning Jacket
14 songs
California Nights
Best Coast
12 songs
Special Effects
Tech N9ne
24 songs EXPLICIT
Jackie (Deluxe)
Ciara
16 songs EXPLICIT
Fated
NOSAJ THING
15 songs
Dark Bird Is Home
The Tallest Man On ...
10 songs
Wilder Mind
Mumford & Sons
12 songs
BUSH
Snoop Dogg
10 songs
Pitch Perfect 2 (Ori...
Various Artists
18 songs


airbnb
WELCOME HOME
Rent unique places to stay from local hosts in 190+ countries.
How It Works
Search by city, address, landmark...
Check In
Check Out
1 Guest


FOURSQUARE
I'm looking for...
New York
Blake
Here are some popular tips in New York. Select a taste to see more:
"Home to some of New York's most famous concerts. Diana Ross literally singing in the rain, Simon & Garfunkel, Garth Brooks and Dave Matthews Band have all played Central Park."
TODAY Show @ Central Park
Good for dates
Cupcakes
Espresso
Cocktails
Black tea
Wine
Chicken wings
Sunsets
Donuts
What are you looking for?
Food
Coffee
Nightlife
Fun
Shopping
9.7
Central Park
Park 59th St to 110th St New York, NY
9.7
Minskoff Theatre
Theater 200 W 45th St New York, NY
9.6
Strand Bookstore
Bookstore 828 Broadway New York, NY

Atlassian Bitbucket

Features Pricing Log in Get started

Your team, your code, connected

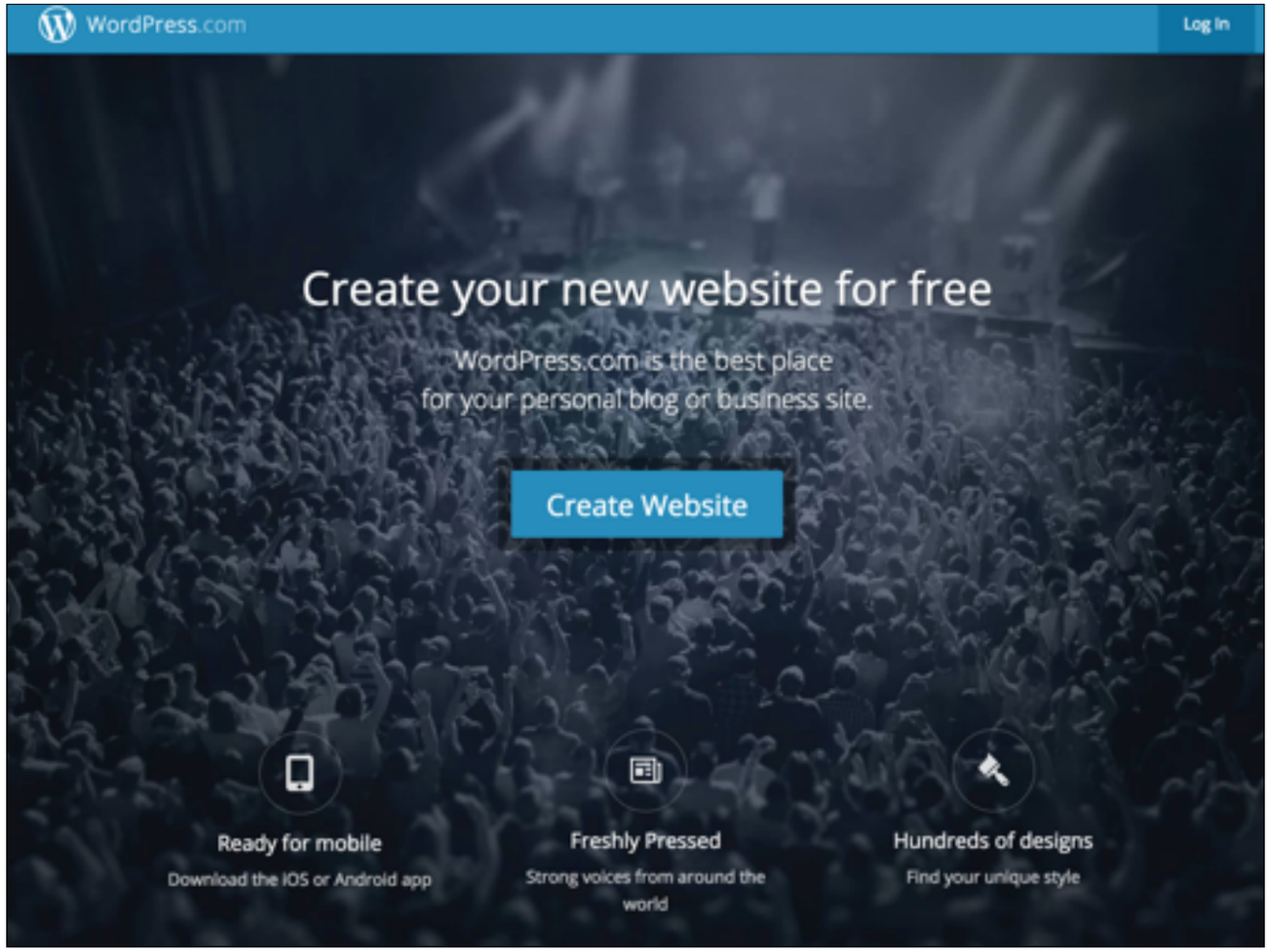
Git and Mercurial code management for teams

Get started for free

Free for 5 users + Unlimited private repositories

More than 330,000 teams and 2.5 million developers love Bitbucket

salesforce TESLA The New York Times PayPal DHL


WordPress.com
Log In
Create your new website for free
WordPress.com is the best place
for your personal blog or business site.
Create Website
Ready for mobile
Download the iOS or Android app
Freshly Pressed
Strong voices from around the world
Hundreds of designs
Find your unique style

A little closer to home…
The tonight show starring Jimmy Fallon is actually a decoupled Drupal site, using backbone to render the front-end. Built by the NBC team right here in Los Angeles, with some assistance from Lullabot.

Switching gears, next up is Angular.js

originally targeted at non-devs
the way html (and the browser)_should_ work (according to angular developers, at google)

SPA

drupalize me

Single Page Application framework pain points

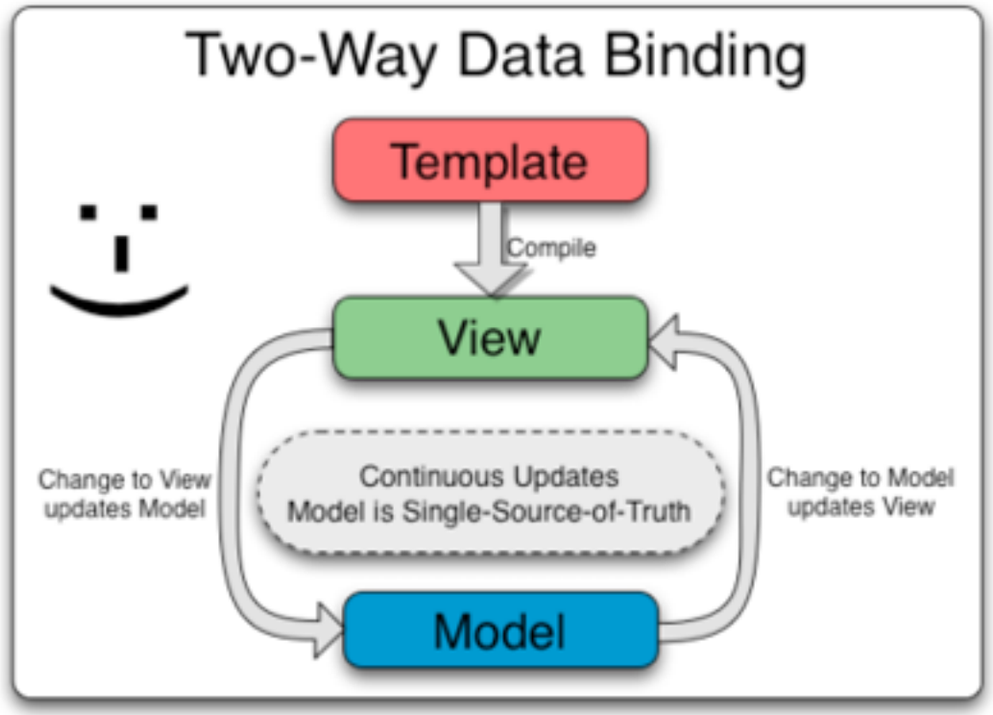
Declarative programming — ie: template driven

Data Binding & Directives

drupalize me

The two core features of Angular.js we're going to take a quick look at are it's two-way Data Binding

and something called Directives



https://docs.angularjs.org/guide/databinding

Angular requires us to build a lot of the application's structure into our templates.
The controller then loads data from our Models, and Angular handles keeping the models and views in sync.
The result is that our templates are continuously reloaded, and update immediately to data changes.

(performance implications of dirty checking)

```
<script type="text/ng-template" id="todomvc-index.html">
  <section id="todoapp">
    <header id="header">
      <h1>todos</h1>
      <form id="todo-form" ng-submit="addTodo()">
        <input id="new-todo" placeholder="What needs to be done?" ng-model="newTodo"
      </form>
    </header>
    <section id="main" ng-show="todos.length" ng-cloak>
      <input id="toggle-all" type="checkbox" ng-model="allChecked" ng-click="markAll(
      <label for="toggle-all">Mark all as complete</label>
      <ul id="todo-list">
        <li ng-repeat="todo in todos | filter:statusFilter track by $index"
          ng-class="{completed: todo.completed, editing: todo == editedTodo}">
          <div class="view">
            <input class="toggle" type="checkbox" ng-model="todo.completed" ng-change
            <label ng-dblclick="editTodo(todo)">{{todo.title}}</label>
            <button class="destroy" ng-click="removeTodo(todo)"></button>
          </div>
          <form ng-submit="saveEdits(todo, 'submit')">
            <input class="edit" ng-trim="false" ng-model="todo.title" todo-escape="re
            ng-blur="saveEdits(todo, 'blur')" todo-focus="todo == editedTodo">
          </form>
        </li>
      </ul>
    </section>
```

Here’s the angular template for our todo app.

You can see addTodo() is called as a new todo item is submitted, and the ng-repeat directive loops over a variable called todos to print out our list of tasks.

```
32      $scope.addTodo = function () {
33        var newTodo = {
34          title: $scope.newTodo.trim(),
35          completed: false
36        };
37
38        if (!newTodo.title) {
39          return;
40        }
41
42        $scope.saving = true;
43        store.insert(newTodo)
44          .then(function success() {
45            $scope.newTodo = '';
46          })
47          .finally(function () {
48            $scope.saving = false;
49          });
50      };
```

Here is the portion of the todo controller responsible for adding a new todo item.
It does this by inserting it into the store object (which is mapped to $scope.todos elsewhere)

```
8   angular.module('todomvc')
9     .controller('TodoCtrl', function TodoCtrl($scope, $routeParams, $filter, store)
10      'use strict';
11
12      var todos = $scope.todos = store.todos;
13
```

Here is another portion of the controller, which sets store.todos = $scope.todos.

This $scope.todos value is what gets passed to the template we looked at earlier.

Directives

a DSL for your HTML

drupalize me

Directives allow us to define our own custom HTML elements, with complex behavior.

```
24          <ul id="todo-list">
25            <li ng-repeat="todo in todos | filter:statusFilter track by $index"
26              ng-class="{completed: todo.completed, editing: todo == editedTodo}">
27              <div class="view">
28                <input class="toggle" type="checkbox" ng-model="todo.completed" ng-cl
29                <label ng-dblclick="editTodo(todo)">{{todo.title}}</label>
30                <button class="destroy" ng-click="removeTodo(todo)"></button>
31              </div>
32              <form ng-submit="saveEdits(todo, 'submit')">
33                <input class="edit" ng-trim="false" ng-model="todo.title" todo-escape
                  ng-blur="saveEdits(todo, 'blur')" todo-focus="todo == editedTodo">
34              </form>
35            </li>
36          </ul>
```

One example we saw in the TODO app was ng-repeat. This directive loops over it's contents for each data item that is passed into it. Let's take a look at another (simpler) example.

```
<share-buttons link="http://example.com">
</share-buttons>
```

We're going to build a simple directive that takes this new HTML tag, <share-buttons> and provides links to our favorite social sites.

```
angular.module('share-buttons', [])
  .directive('shareButtons', function() {
    return {
      restrict: 'E',
      scope: {
        link: '='
      },
      templateUrl: 'share_buttons.html'
    }
  });

```

Here’s all the code necessary to register an Angular directive. There’s some name normalization going on, here it’s camelCase whereas in the html it’s hyphenated.

We’re also setting up a link variable, which will be available in our template, and passing in a template file

```
<div class="share-buttons">
  <!-- Facebook (url) -->
  <a href="http://www.facebook.com/sharer/sharer.php?u={{link}}">
    <img src="facebook.png" />
  </a>
  <!-- Twitter (url, text, @mention) -->
  <a href="http://twitter.com/share?url={{link}}">
    <img src="twitter.png" />
  </a>
  <!-- Google Plus (url) -->
  <a href="https://plus.google.com/share?url={{link}}">
    <img src="google.png" />
  </a>
</div>
```

The template file contains the markup that Angular will use to replace our Directive during compilation.

In this case we’re dropping in links to Facebook, Twitter and Google Plus, formatted for easy sharing.

And here’s the final output on the site using the new <share-buttons> HTML tag.

Angular 2.0 is coming, sometime?

It’s really quite different, and rapidly evolving. It looks like they’ll be doing away with a bit of two way data binding in order to improve the performance of dirty checking.

Who uses Angular?

drupalize me

doubleclick by Google

Home Publishers Advertisers/Agencies Networks Insights

Welcome to DoubleClick

The Ad Exchange, Evolved

Brand Activate initiative

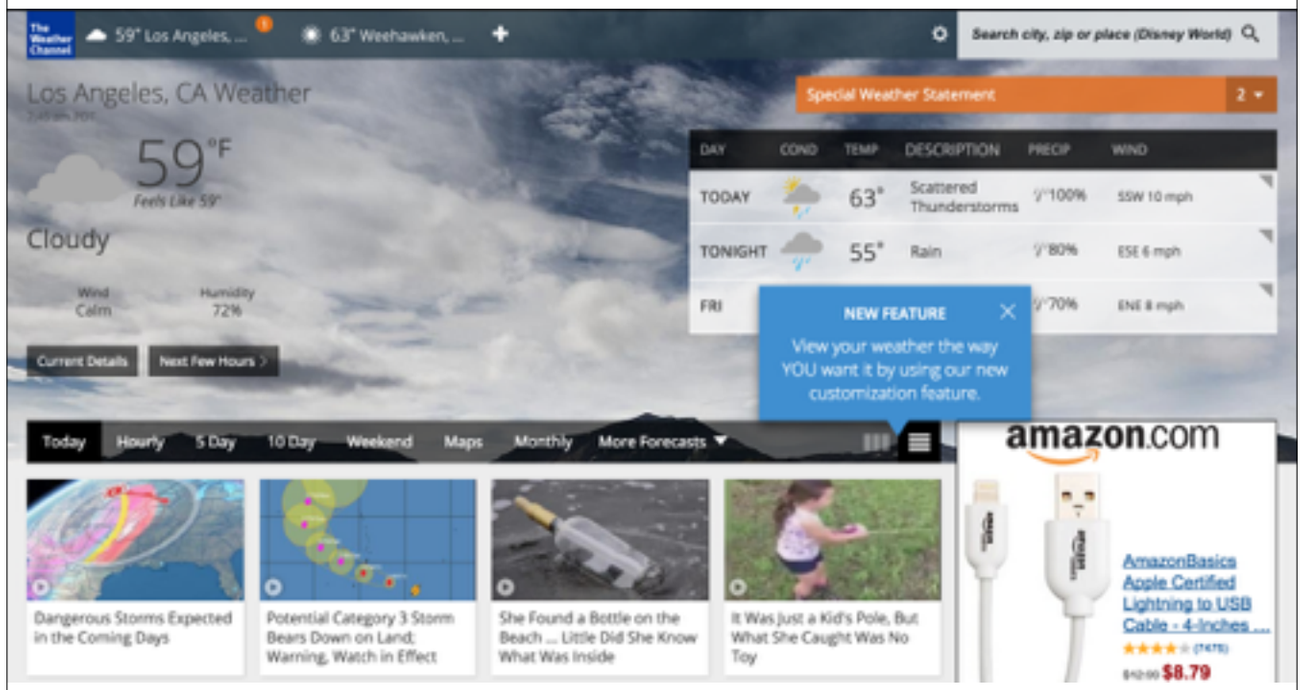
Publisher trends

DoubleClick at CES 2015

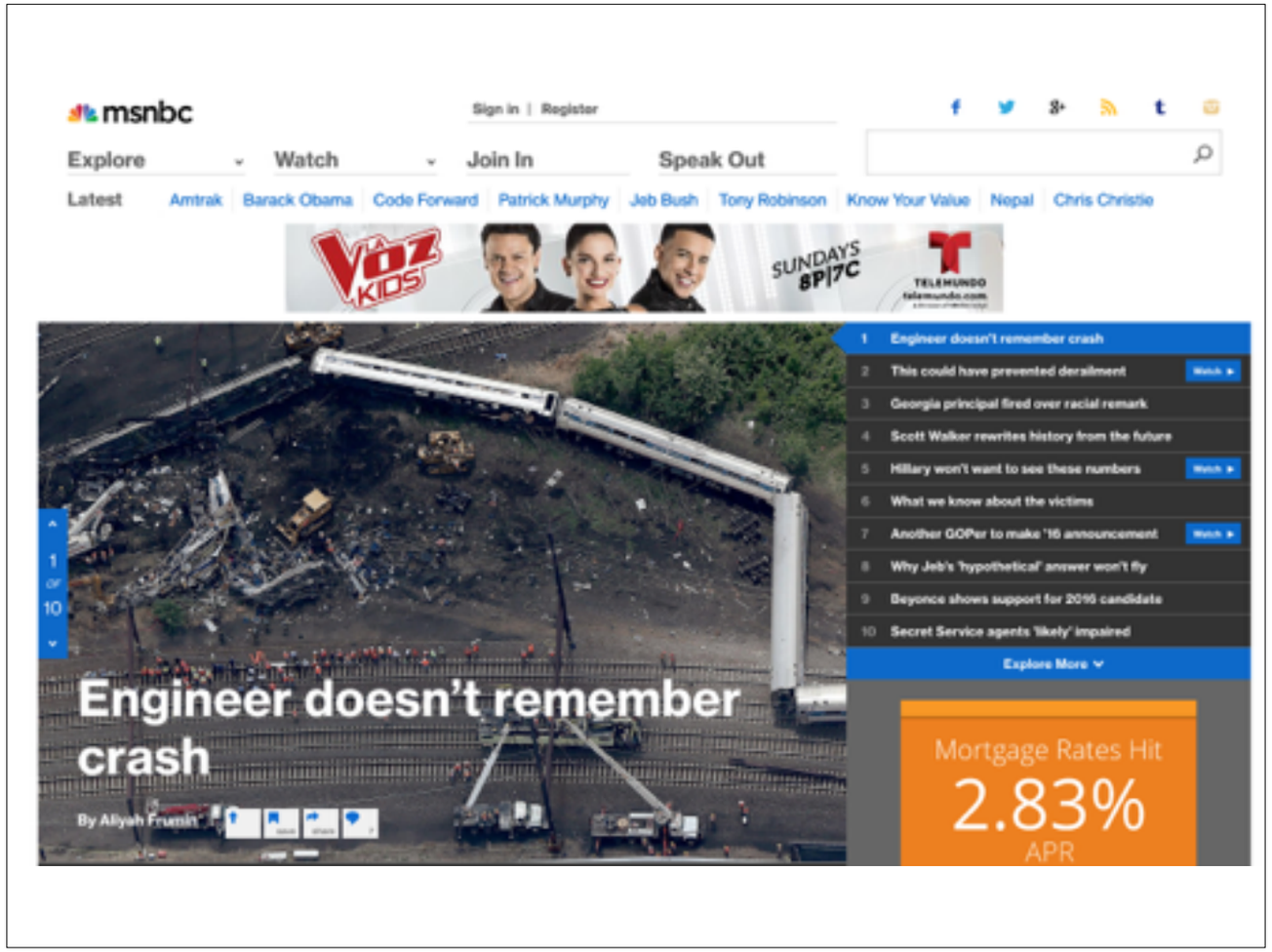
Powering digital advertising globally

DoubleClick is the ad technology foundation to create, transact, and manage digital advertising for the world's buyers, creators and sellers.

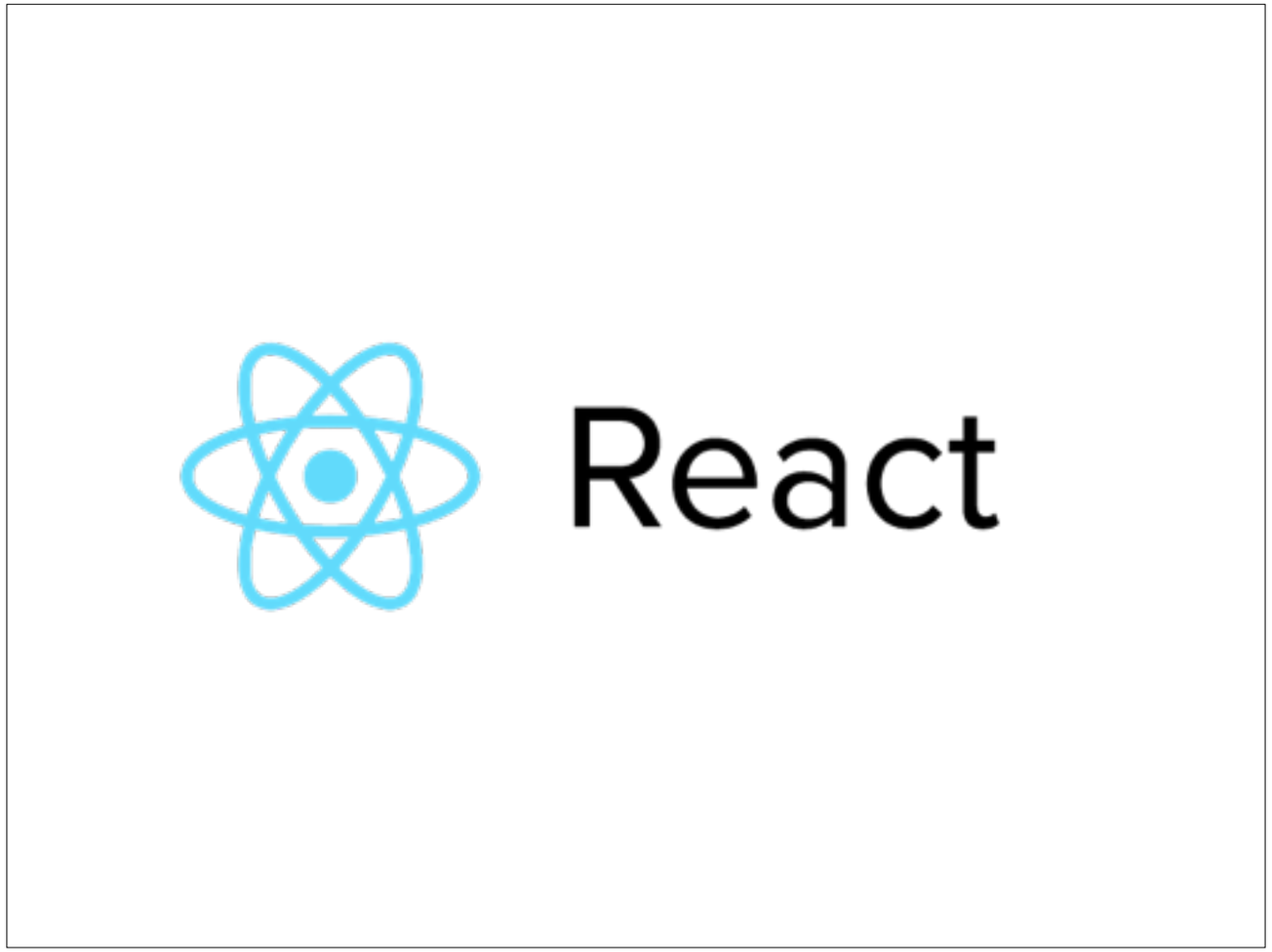
Publishers Advertisers/Agencies Networks



The weather channel (a drupal site)



MSNBC, another Lullabot client.



2013. Quite a bit different from the others.

Just the V

drupalize me

Because it’s really just a templating library by itself.

```
166     function render() {
167       React.render(
168         <TodoApp model={model}/>,
169         document.getElementById('todoapp')
170       );
171     }
```

React uses something called Components to handle rendering portions of a page.
The components each have a render method, that defines their output.
Here’s the main component for the Todo sample app.
Given a model, it drops the TodoApp component into the DOM element with the ID of todoapp

```
    render: function () {
      return (
        <li className={React.addons.classSet({
          completed: this.props.todo.completed,
          editing: this.props.editing
        })}>
          <div className="view">
            <input
              className="toggle"
              type="checkbox"
              checked={this.props.todo.completed}
              onChange={this.props.onToggle}
            />
            <label onDoubleClick={this.handleEdit}>
              {this.props.todo.title}
            </label>
            <button className="destroy" onClick={this.props.onDestroy} />
          </div>
          <input
            ref="editField"
            className="edit"
            value={this.state.editText}
            onBlur={this.handleSubmit}
            onChange={this.handleChange}
            onKeyDown={this.handleKeyDown}
          />
        </li>
      );
    }
  });
```

The ToDo item components' render method, then contains the actual markup displayed for each todo item as well as the input form.

```
76      render: function () {
77        return (
78          <li className={React.addons.classSet({
79            completed: this.props.todo.completed,
80            editing: this.props.editing
81          })}>
82            <div className="view">
83              <input
84                className="toggle"
85                type="checkbox"
86                checked={this.props.todo.completed}
87                onChange={this.props.onToggle}
88
89              <label onDoubleClick={this.handleEdit}>
90                {this.props.todo.title}
91              </label>
92              <button className="destroy" onClick={this.props.onDestroy} />
93            </div>
94            <input
95              ref="editField"
96              className="edit"
97              value={this.state.editText}
98              onBlur={this.handleSubmit}
99              onChange={this.handleChange}
100             onKeyDown={this.handleKeyDown}
101           />
102         </li>
103       );
104     }
105   });
```

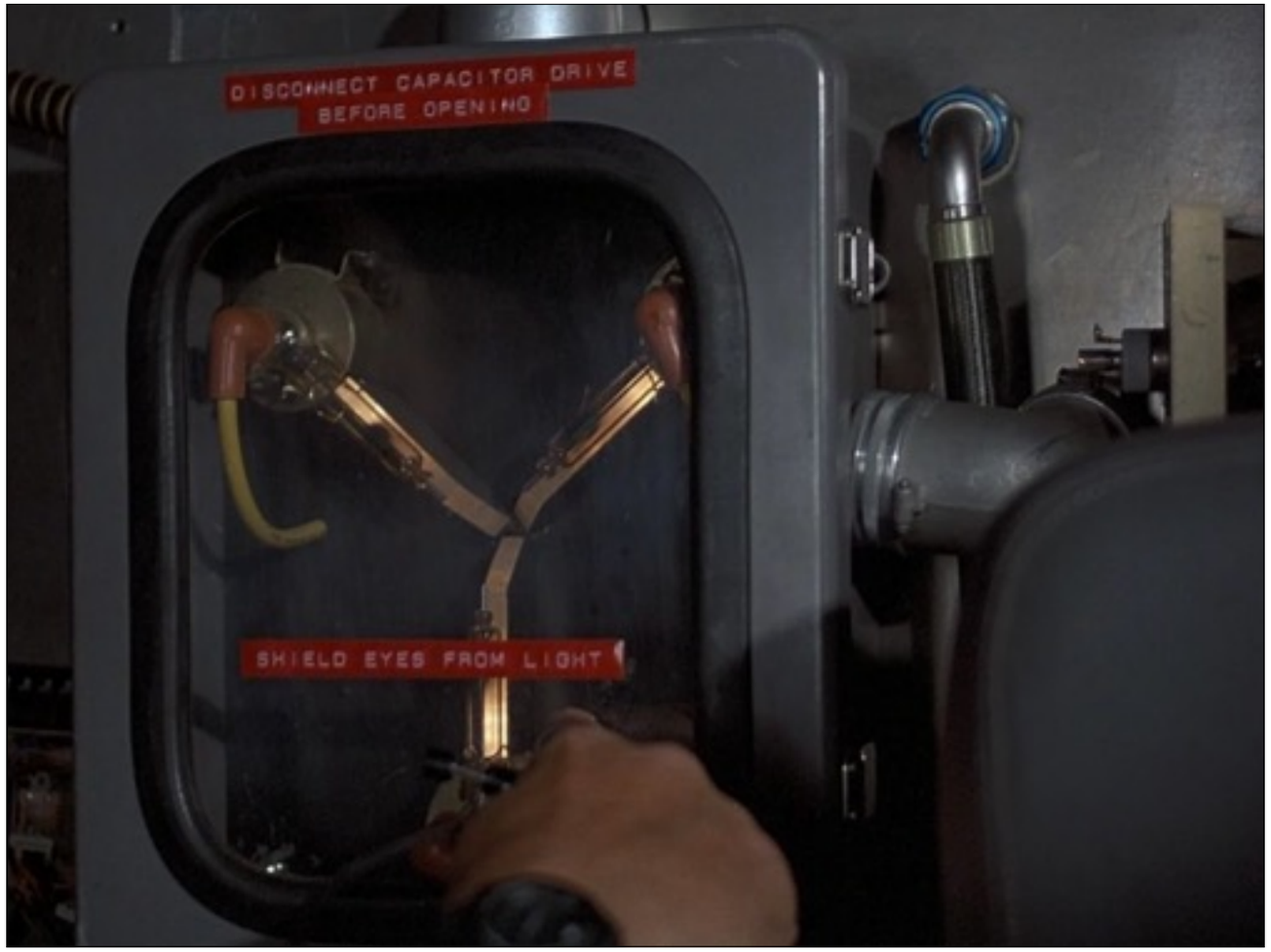
Within the component we map actions to component methods responsible for handling the user interaction.

```
25      handleEdit: function () {
26        this.props.onEdit();
27        this.setState({editText: this.props.todo.title});
28      },
29
```

then, in the event handler we use setState to update the data model of our component.

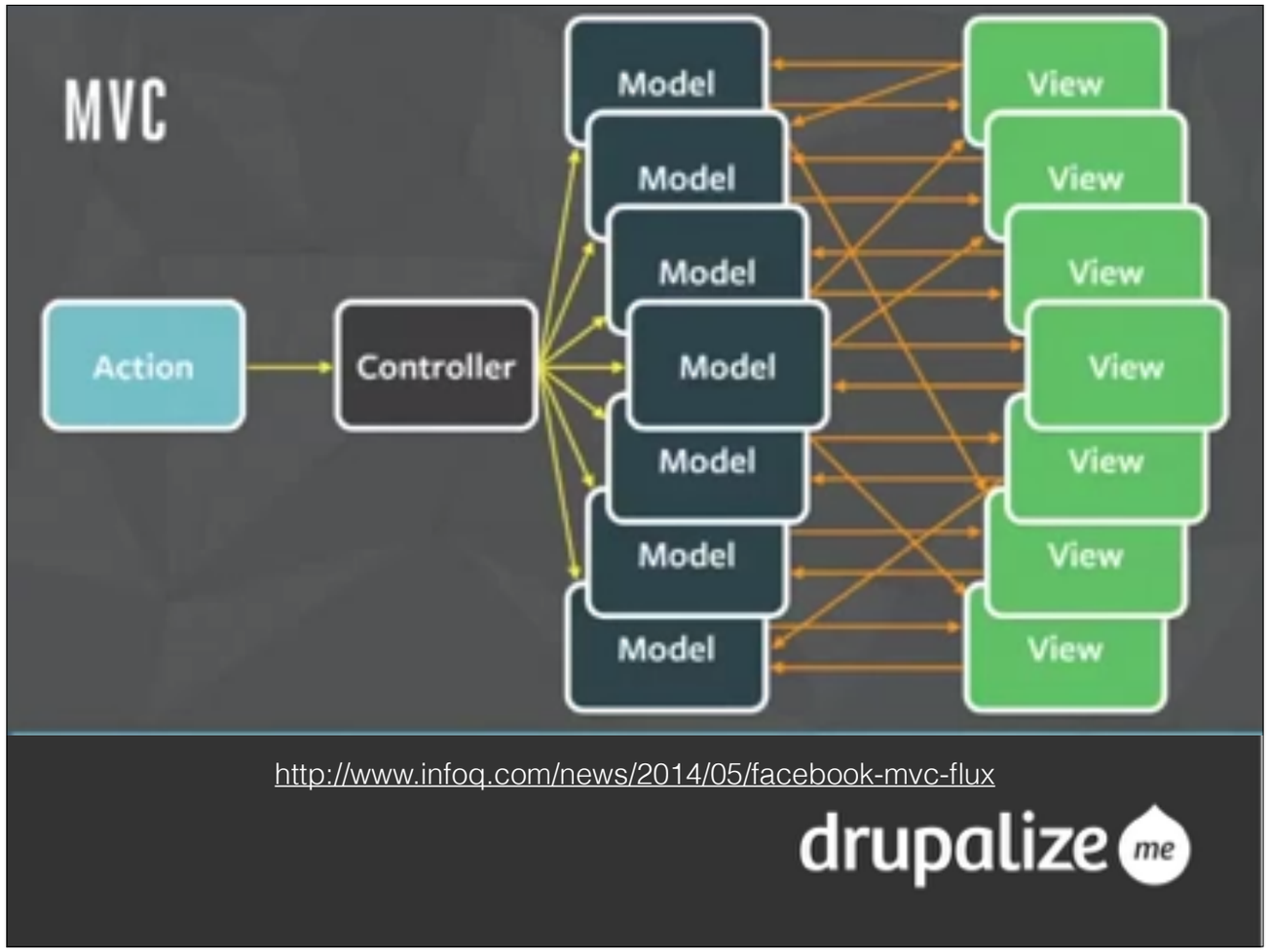
```
76      render: function () {
77        return (
78          <li className={React.addons.classSet({
79            completed: this.props.todo.completed,
80            editing: this.props.editing
81          })}>
82            <div className="view">
83              <input
84                className="toggle"
85                type="checkbox"
86                checked={this.props.todo.completed}
87                onChange={this.props.onToggle}
88              />
89              <label onDoubleClick={this.handleEdit}>
90                {this.props.todo.title}
91              </label>
92              <button className="destroy" onClick={this.props.onDestroy} />
93            </div>
94            <input
95              ref="editField"
96              className="edit"
97              value={this.state.editText}
98              onBlur={this.handleSubmit}
99              onChange={this.handleChange}
100             onKeyDown={this.handleKeyDown}
101           />
102         </li>
103       );
104     }
105   });
```

The components' render method is using this same state to display the UI, and react ensures it's re-rendered when there has been a data change.

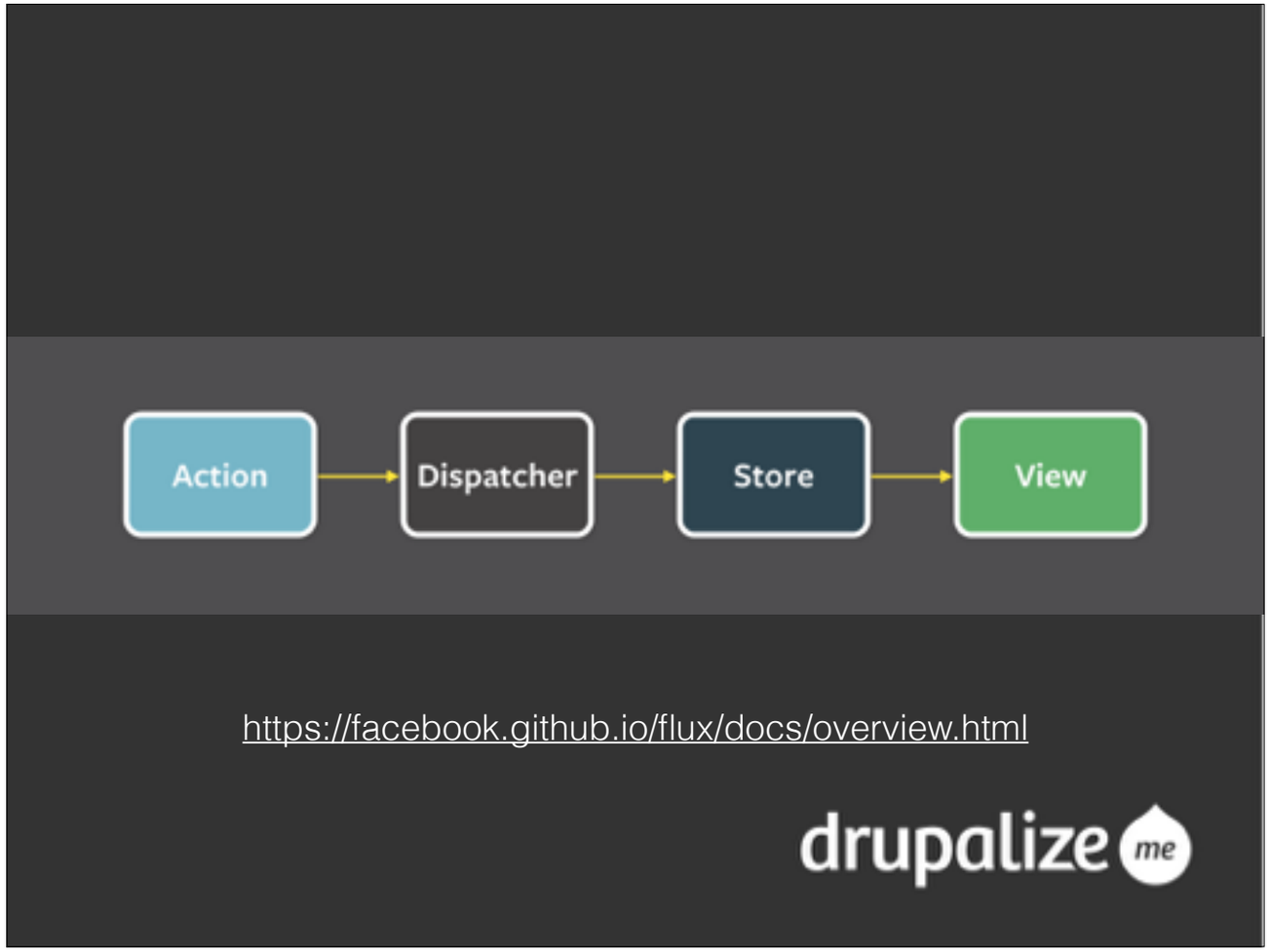


Let’s talk a bit about a related technology / approach:

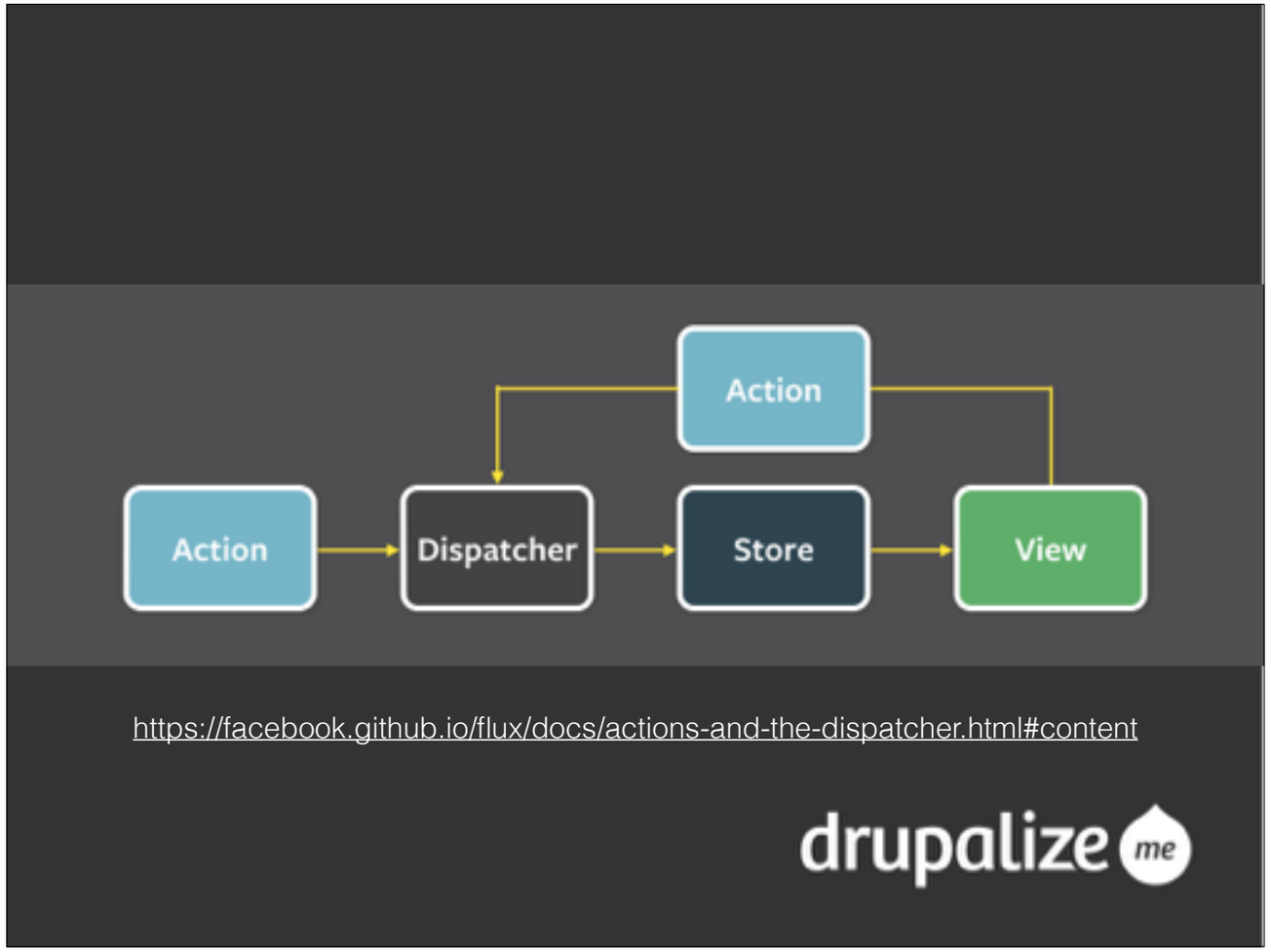
Something called Flux



In a traditional MVC framework, like backbone or angular, it can be difficult to reason about the cascading effects of data changes across models.



In flux, the event and data flow is unidirectional.

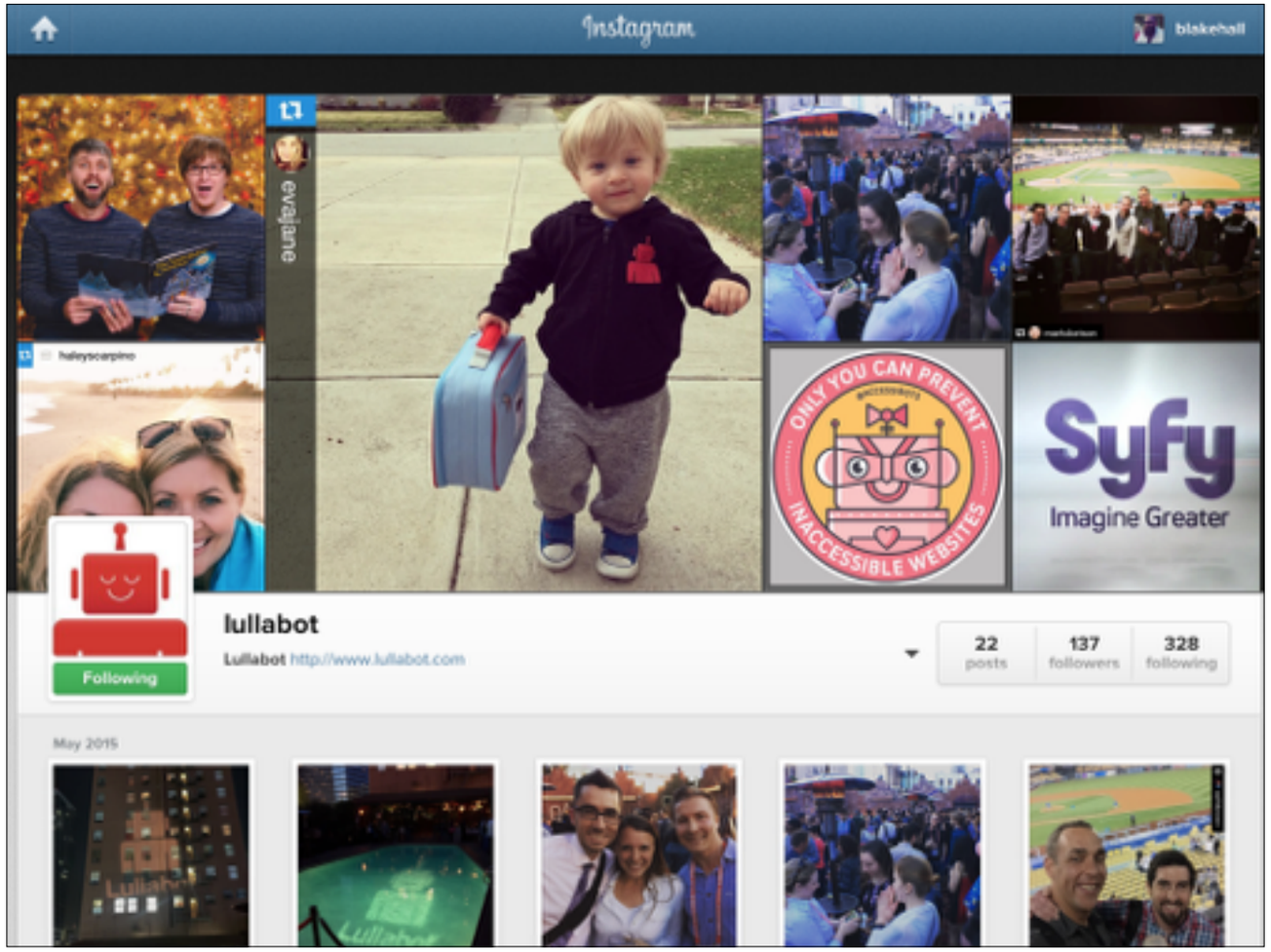


Contrast this approach to what we saw with Backbone.

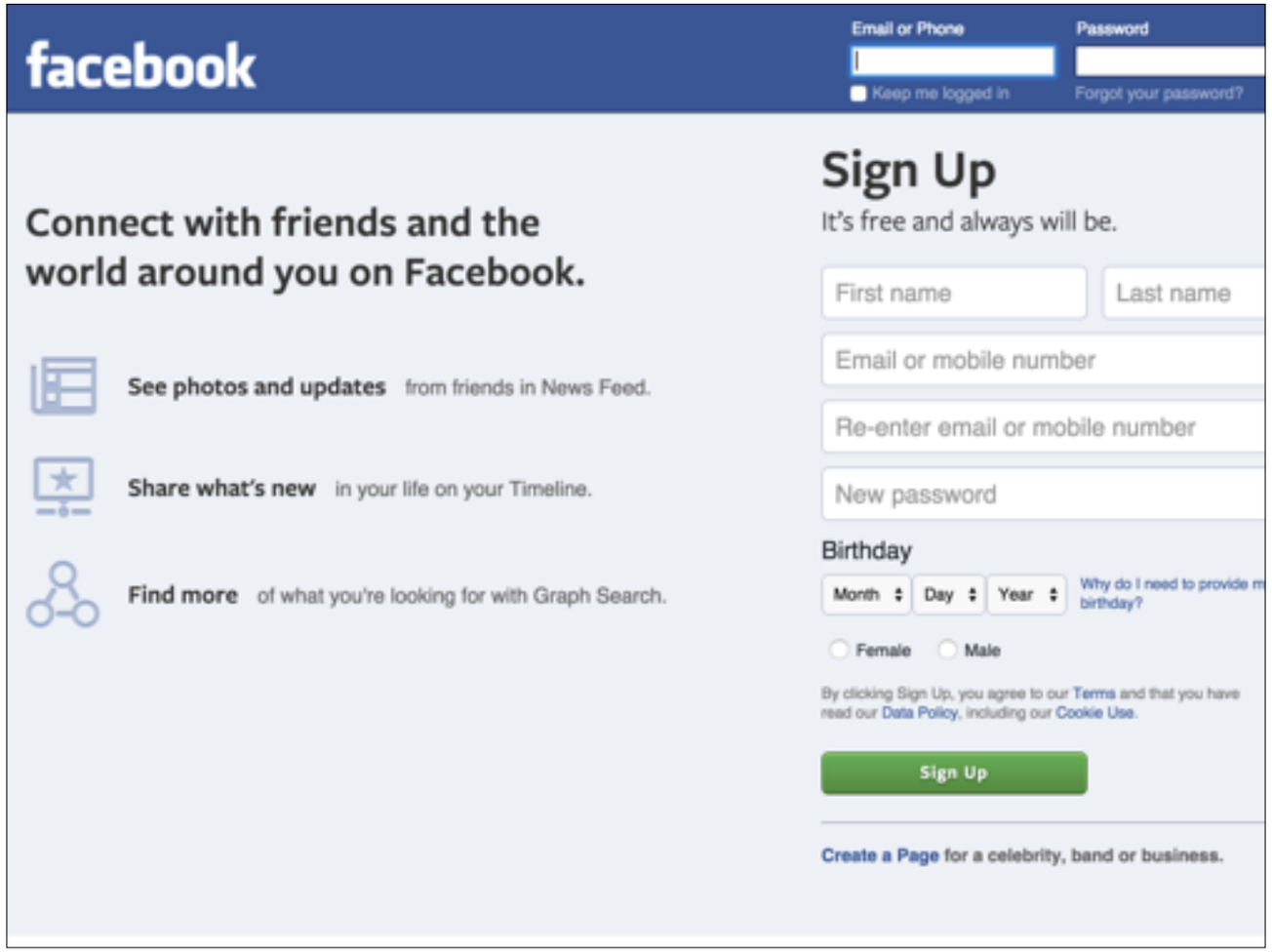
In this case, data is only moving one direction making keeping things in sync a much easier task.
This comes at the expense of completely re-rendering components. It turns out, doing that is pretty performant thanks to something React uses called the Virtual DOM.

Who uses React?

drupalize me

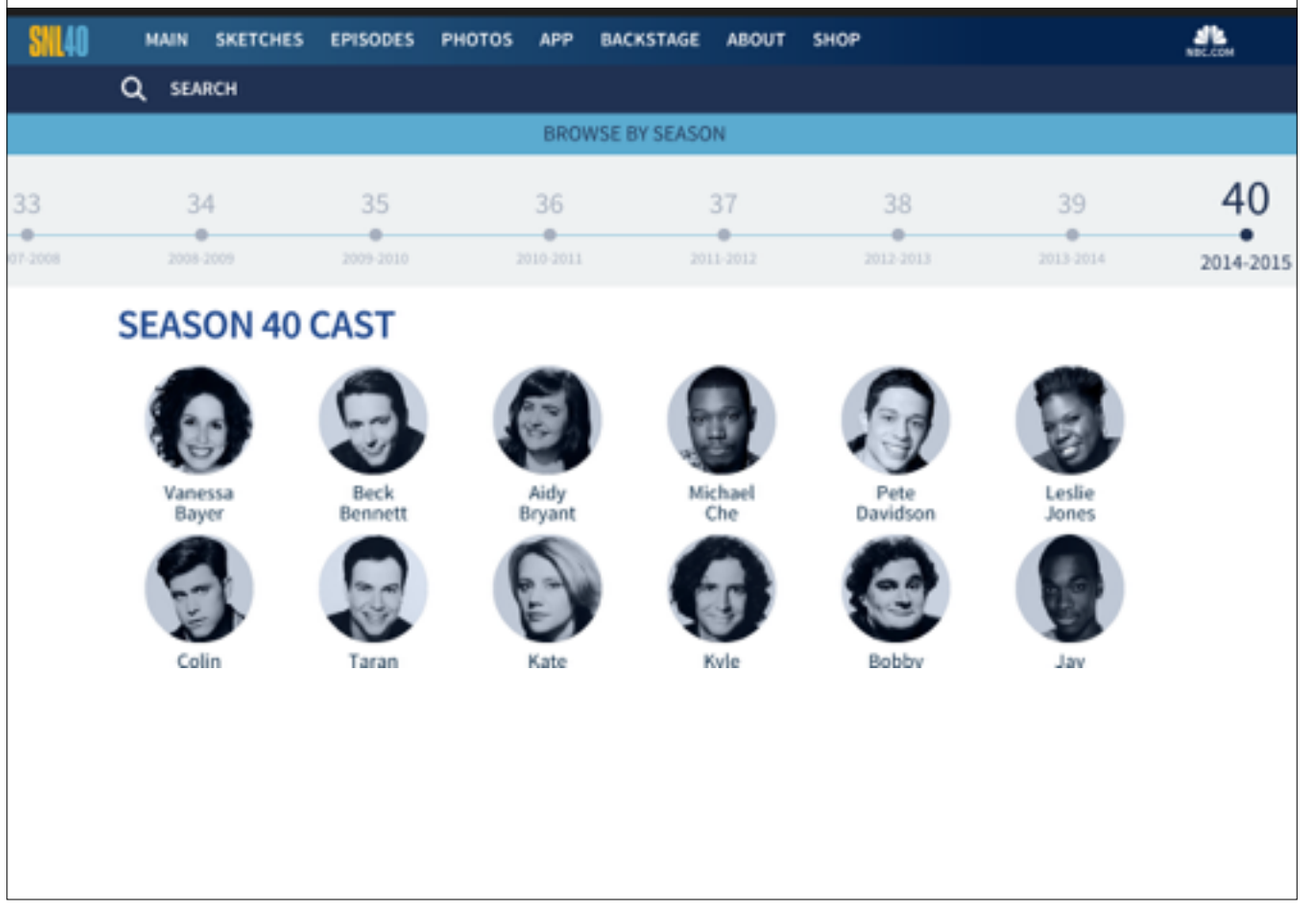


Initial versions of React were developed by the folks at Instagram.

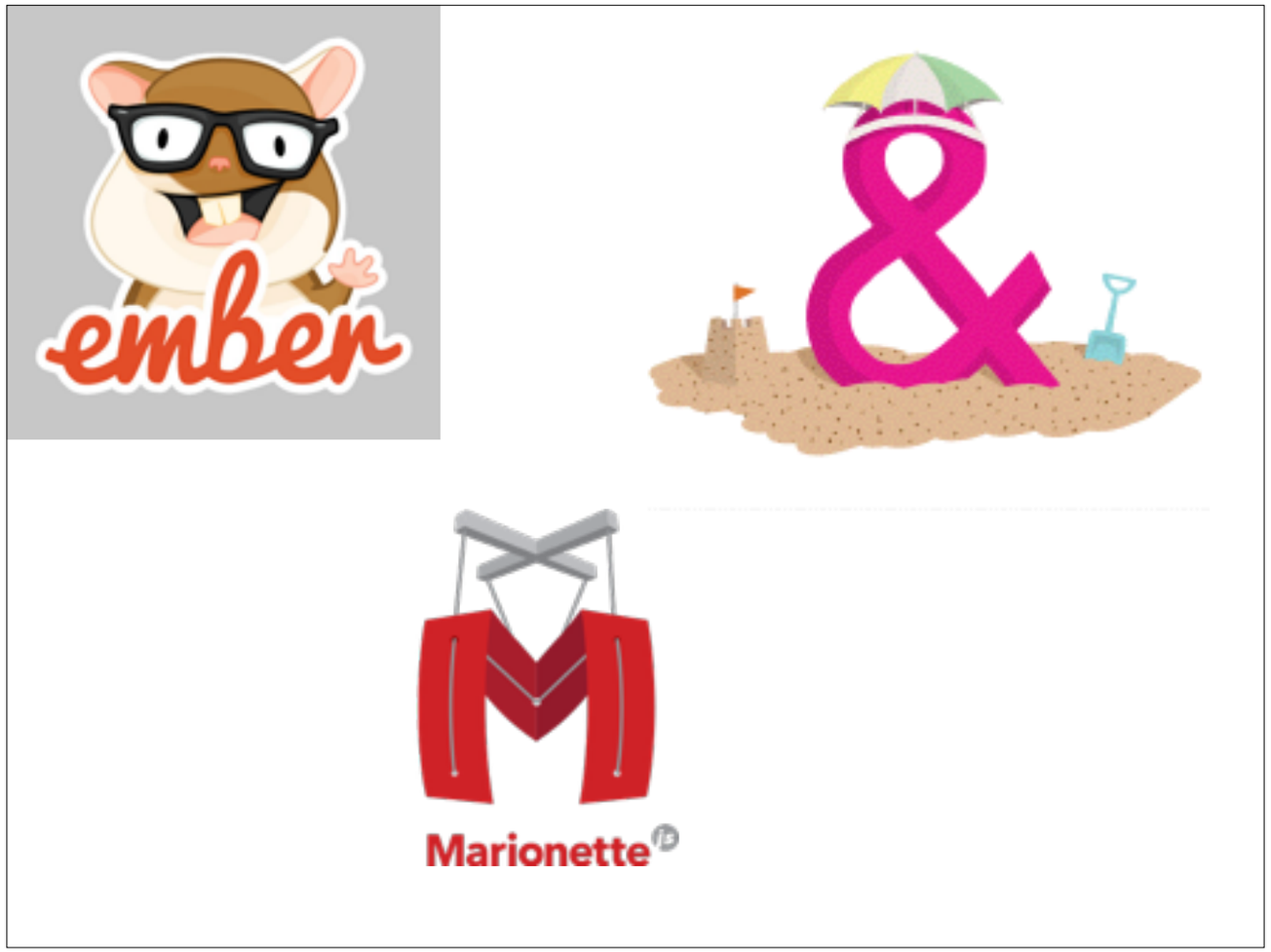


I mentioned that React is constantly re-rendering entire components based on data changes.

It's working well enough for these folks.



And another, recently launched decoupled drupal site (that I helped out with earlier this year).



Other frameworks…

John Hannah
Front-end Developer

Want John Hannah to speak at your event? Contact us with the details and we'll be in touch soon.

by John Hannah on April 9, 2015 // Short URL

Choosing the Right JavaScript Framework for the Job

Weighing the pros and cons of popular front-end frameworks

7 Comments // 157 Tweets // Share

If you've been following web development over the past few years, you will no doubt have noticed that JavaScript frameworks are an increasingly popular way to build web applications. Although there are many frameworks out there,

Read this https://www.lullabot.com/blog/article/choosing-right-javascript-framework-job!

Questions?

blake@lullabot.com
@blakehall

drupalize me

Also come find me at the Drupalize.Me booth this afternoon. #407 & 411

DRUPALCON ★ LOS ANGELES

WHAT DID YOU THINK?

EVAULATE THIS SESSION - LOSANGELES2015.DRUPAL.ORG/SCHEDULE

THANK YOU!