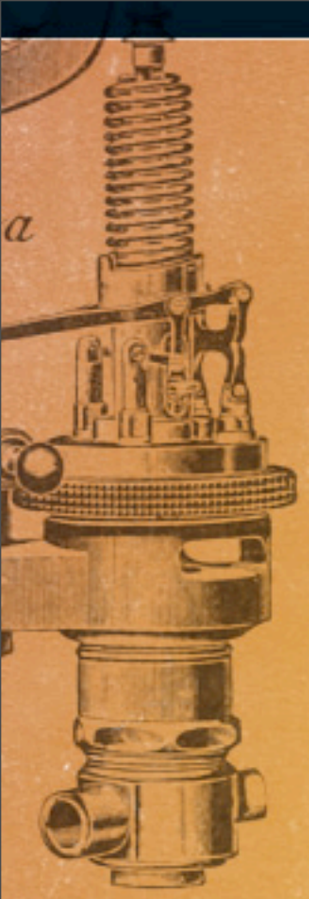
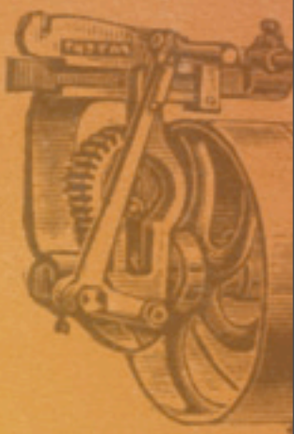
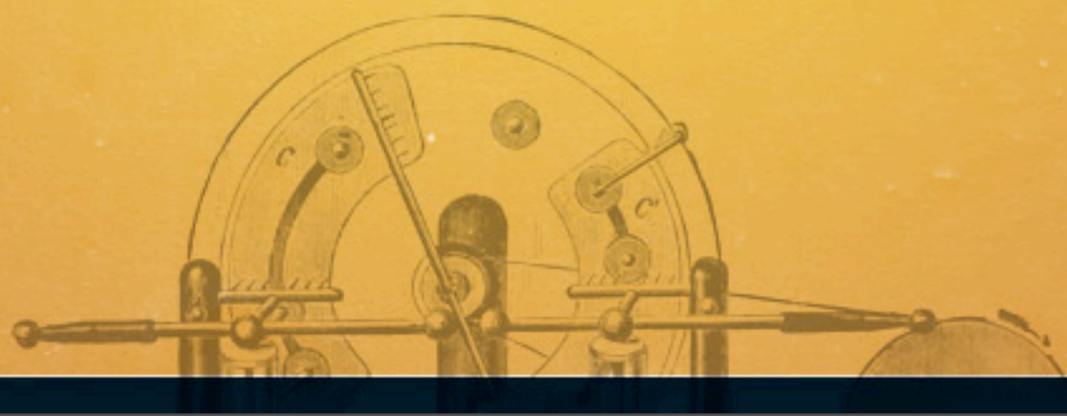




TOOLS FOR jQUERY APPLICATION ARCHITECTURE

A PRESENTATION BY ADDY OSMANI



"Always walk through life as if you have something new to learn and you will"

Who Am I?

- Member of the jQuery /(Bugs|Docs|Learning|Front-end)/ Subteams
- Software Engineer at AOL
- Writer (Essential JS Design Patterns, AddyOsmani.com, Script Junkie etc.)

AOL? You guys still exist?!

Yes, we do!

We actually actively use many of the tools
I'll be covering today.

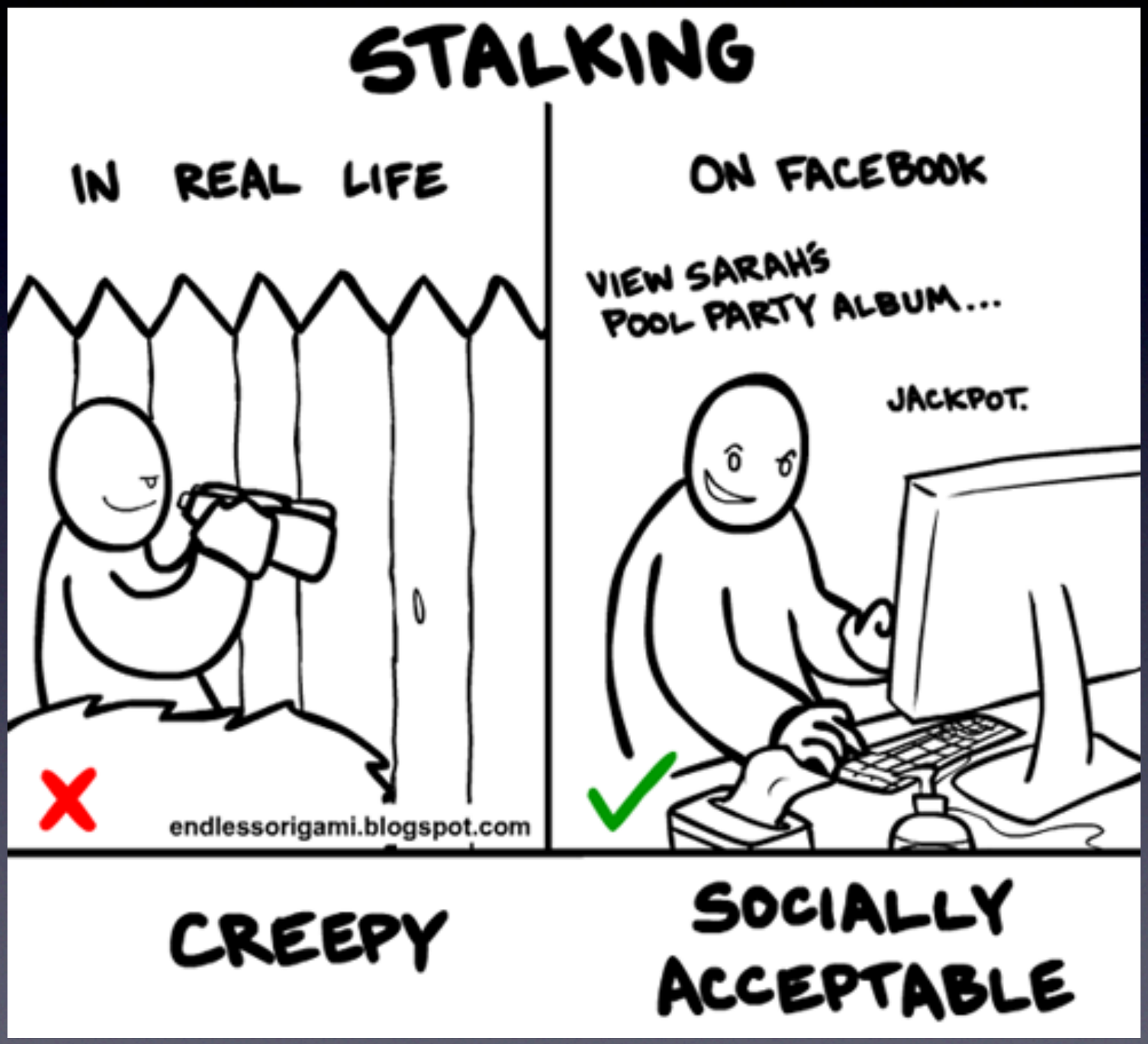

Let’s rock!

jQuery is great..but!

- It doesn't provide all the tools needed to build serious JavaScript web applications
- It does give us easy access to DOM manipulation, ajax , animation, events etc.
- But how do we fill in the gaps for everything else?

Let's evaluate what we're missing by building a hypothetical application

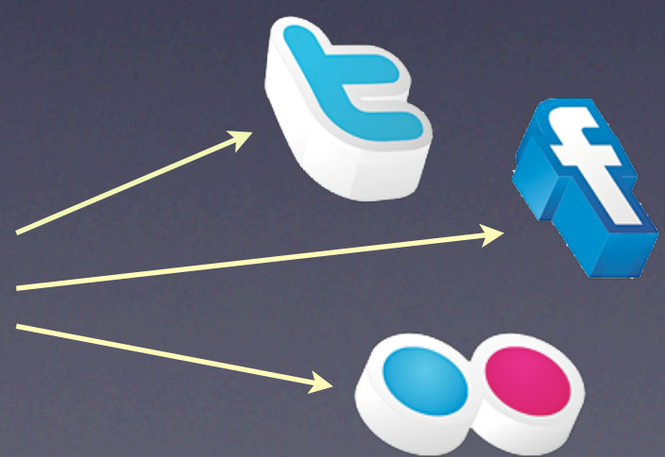
StalkBook!


STALKING
IN REAL LIFE
ON FACEBOOK
VIEW SARAH'S POOL PARTY ALBUM...
JACKPOT.
endlessorigami.blogspot.com
CREEPY
SOCIALLY ACCEPTABLE

What does it do?

- Enter a friend's name to find accounts that match it on other social networks
- Allows sharing different views of results easily
- Remembers user's last known configuration

Jon Stewart



btw fool..it also has to..

- Be quick and support easily changing behaviour later
- Offer multiple views of the data, but require no hard page refresh
- Run without any database on our end
- Be well tested & perform optimally

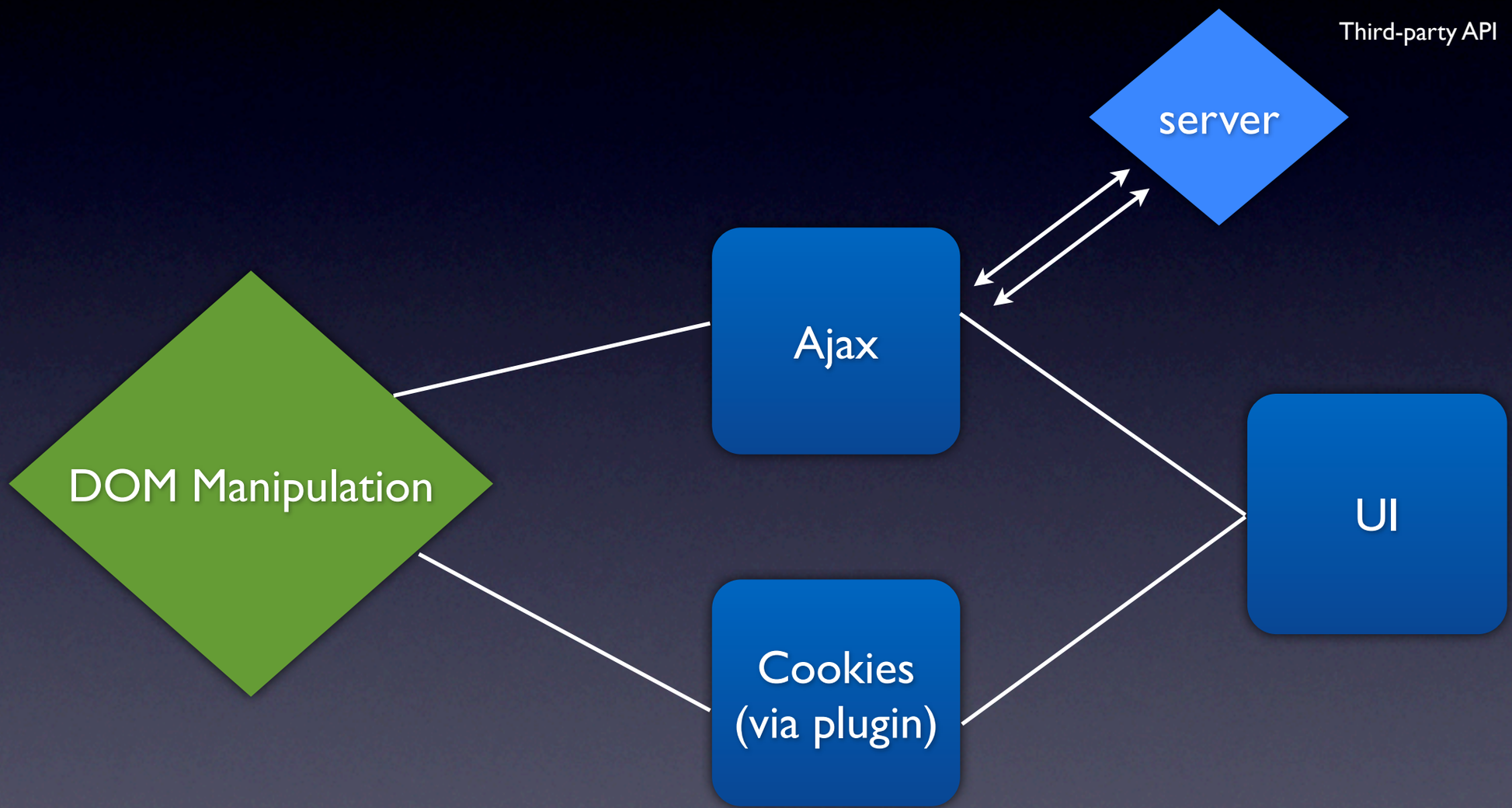
Mr. T, Project manager

Some implementation notes

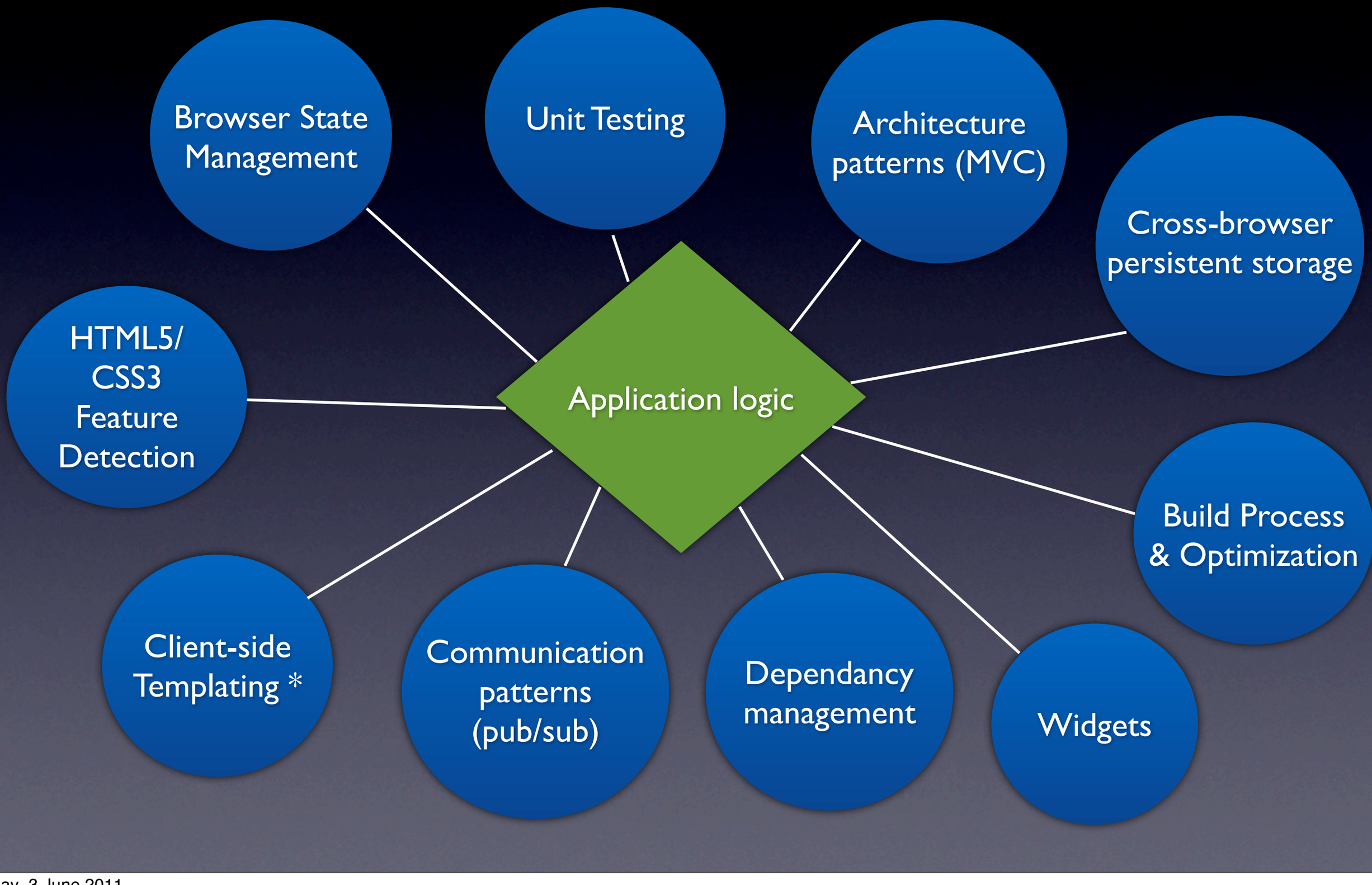
- Could be a single-page application
- Separate application concerns & keep the code decoupled
- Store persistent configuration information
- Make best use of modern technologies to improve the user experience
- Work cross-browser with fallbacks for IE

What do we have to work with?

jQuery can give us this



But we're missing *this*




Let’s fill in the gaps, bro!

Code Organization:
Design Patterns

- Reusable solutions to commonly occurring problems in software design
- Proven approaches that reflect experience and insights of developers over time
- Highly flexible & can be adapted to our own needs easily

I can haz architecture?

- Architecture patterns describe patterns for broader systems with a larger scope.
- Certain patterns can be applied to both the server-side and client-side
- Let's look at one you know

The MVC Pattern

- Separates objects into three main concerns: models, views and controllers
- Traditionally heavily used on the server-side
- Excellent for code-organisation in general
- Implementations can vary quite significantly

Models, Views & Controllers

- Models represent knowledge and data (eg. get and set methods). Isolated from Views and Controllers
- Views can be considered the UI. Generally ‘dumb’. Don’t need to perform validation logic
- Controller sits between Models and Views. Performs business logic, data manipulation

MVC Options

- JavaScriptMVC
- Backbone.js + Underscore
- Spine.js
- SproutCore
- Sammy.js

JavaScriptMVC

Lead: Justin Meyer

- JMVC can really be considered two things:
  - Repeatable MVC architecture
  - All-in-one integrated development tools
- Benefit is that it provides a clear path to adding functionality
- Does a lot of the things you should be doing in your project anyway

JMVC Components

- JMVC is broken down into 4 projects:
  - jQueryMVC - MVC extensions for jQuery
  - StealJS - Dependency management, build process, code generators
  - FuncUnit - A web testing framework
  - DocumentJS - A JavaScript docs framework

JMVC's MVC

- **Model:** classical model layer. Effectively, a way to package and organize Ajax requests and service data
- **Controller:** jQuery widget factory
- **View:** client side templating (eg. jQuery.tmpl, Micro)
- **Class:** a JavaScript-compatible class system

JMVC Class

```
$.Class("Speaker",{
    talk: function(){]
        console.log("I just said something");
    }
});

//usage
var douglasCrockford = new Speaker();
douglasCrockford.talk();

//we can take this further as follows:
Speaker("Organizer",{
    announce: function(){
        console.log("please take your seats");
    }
});

//usage
var johnAlsopp = new Organizer;
johnAlsopp.announce();
johnAlsopp.talk();
```

JMVC Model

```
$.Class("Model",{
   //get a speaker's slides
   getSlides:function(eventName, success){
    $.get('/slidesApi/'+ eventName, this.callback(function(q){
       success(new this(q));
      })
    },'json')
  }
},{
   init:function(q){
    $.extend(this, q)
  }
});

Model("Crockford", {
   getRecentSlidesUrl: function(){
      return "My most recent slides are at:" + this.slideUrl;
   }
});

Crockford.getSlides('webDirections', function(crock){
   console.log(crock.getRecentSlidesUrl());
});
```

JMVC Controller

```
$.Controller('ModalWindow',{
  //sets up the widget
  init: function(){
    this.find('.modal').hide();
  },
  ".newModal keyup": function(el, ev){
    if(ev.keyCode == 13){
     new Modal({
         title : el.attr('title'),
         href: el.attr('href'),
         complete : false
     }).save(this.callback('created'));
    }
  },
  ".modal .close click" : function(el){
    el.closest('.modal').model().destroy();
  }
});
```

JMVC View

```
//Views essentially use client-side templating
//Here's a simple example

<script type='text/ejs' id='slidesEJS'>
<% for(var i=0; i < slides.length; i++){ %>
    <li><%=slides[i].name %> were talked about at <%=slides
[i].eventName %>.

A preview of the talk can be found below:<img src="<%=slides
[i].thumbnailUrl %>"/></li>
<%} %>
</script>

//Which can be easily rendered using..
$("#slides").html('slidesEJS', slideData);
```

Why JMVC?

- Maturity & includes testing, dependency management, build tools, client side templates.
- Perfect for large applications where an all-in-one solution is required.
- Scales well. It was most recently used by Grooveshark in their application re-write.

GrooveShark

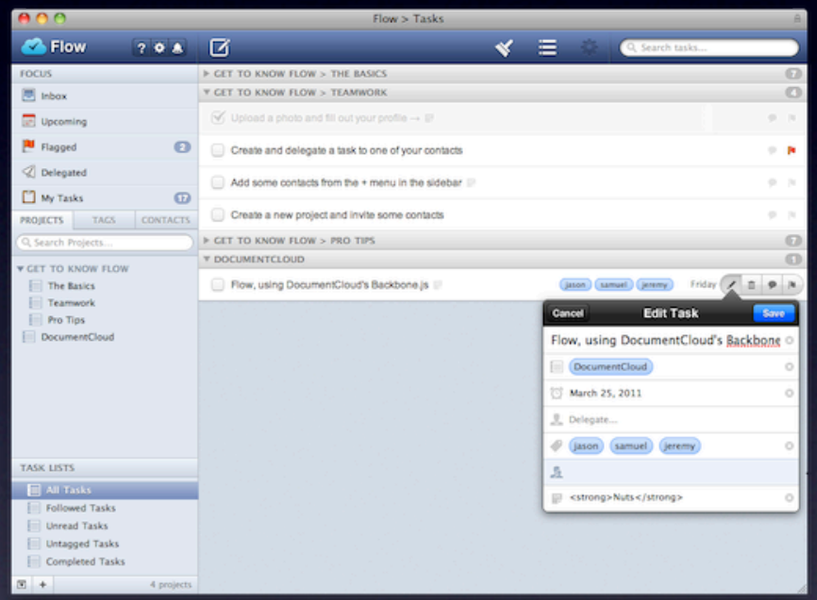
Backbone.js

Lead: Jeremy Ashkenas

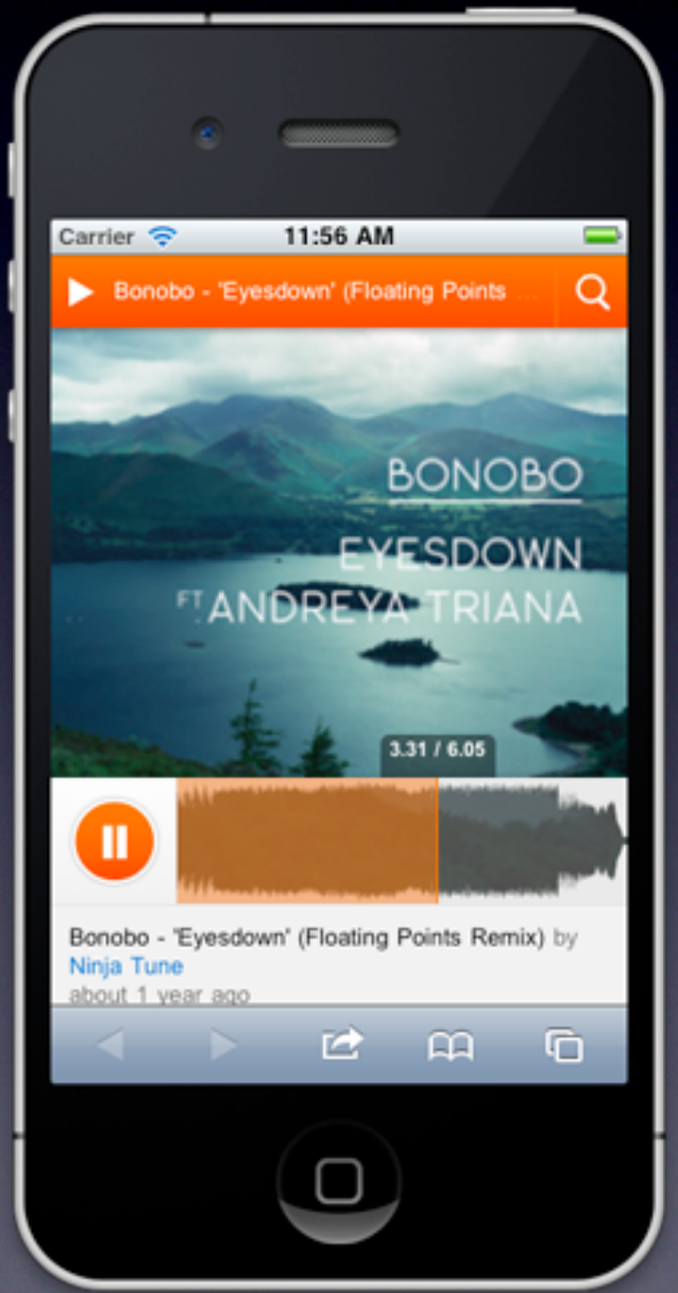
- Backbone provides the 'backbone' you need to organize your code using the MVC pattern.
- Excellent for a lightweight solution where you select the additional components you feel work best for your project.
- Works best for SPAs (single-page apps)

Who uses it?

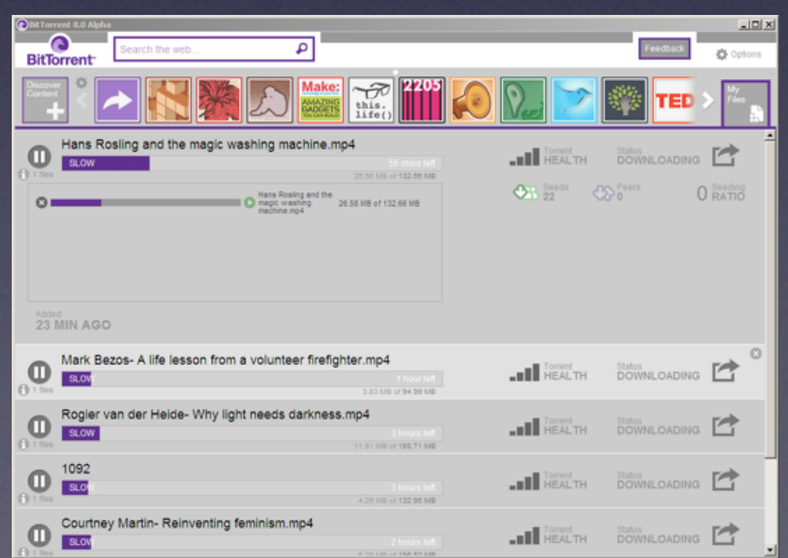
MetaLab



SoundCloud



BitTorrent



Backbone's MVC

- Models: represent data that can be created, validated, destroyed & listened to for changes. Collections are sets of models.
- Views: display the model's data / provide the user interface layer
- Controller: methods for routing client-side URL fragments

Backbone Model

```
window.Note = Backbone.Model.extend({
  // Default attributes for the note
  defaults: {
    content: "empty note",
    done: false
  },

  // Make sure each note has default 'content'
  initialize: function() {
    if (!this.get("content")) {
      this.set({"content": this.defaults.content});
    }
  },

  // Toggle the `done` state of this note
  toggle: function() {
    this.save({done: !this.get("done")});
  },

  // Remove this note from local-storage.
  clear: function() {
    this.destroy();
    this.view.remove();
  }
});
```

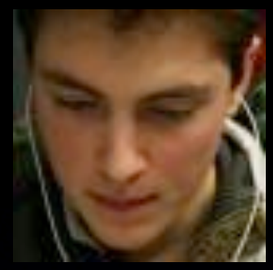
Backbone Collection

```
window.StickyNotes = Backbone.Collection.extend({

  // Reference to this collection's model.
  model: Note,

  // Save all of the note items under the `"notes"` namespace.
  localStorage: new Store("notes"),

  // Filter down the list of all notes that are finished.
  done: function() {
    return this.filter(function(note){ return note.get('done'); });
  },

  // Filter down the list to only note items that are still not finished.
  remaining: function() {
    return this.without.apply(this, this.done());
  }
});
```

Backbone View

```
window.AppView = Backbone.View.extend({
    el: $("#notesapp"),
    events: {
      "keypress #new-note":  "createOnEnter",
      "click .note-clear a": "clearCompleted"
    },
   initialize: function() {
      _.bindAll(this, 'addOne', 'addAll', 'render');
      this.input    = this.$("#new-note");
      StickyNotes.bind('add',     this.addOne);
      StickyNotes.bind('reset',   this.addAll);
      StickyNotes.bind('all',     this.render);
      StickyNotes.fetch();
    },
    render: function() {
      this.$('#notes-info').html({ /* template logic could go here*/ });
    },
    addOne:function(note){
      //add a single note
      var view = new NoteView({model: todo});
      this.$("#note-list").append(view.render().el);
    }
    addAll: function() {
      //add everything to the stickynotes collection at once
      StickyNotes.each(this.addOne);
    },
});
```

Spine.js



Lead: Alex MacCaw

- A lightweight MVC framework with a focus on providing an inheritance model through classes and extension.
- Based on Backbone's API so developers may find getting started a little easier than expected
- Personally found it a nice alternative

Spine's MVC

- Classes: traditional 'classes' powered by an emulated version of Object.create
- Models: your application's data storage models as well as any logic associated with this data. Based on classes.
- Controllers: Similar to 'Views' - each controller has an element associated with it. Also based on classes.

Spine vs. Backbone

- Spine's Controllers are effectively the same concept as Backbone's Views.
- In terms of inheritance, Backbone uses constructor functions and prototypes whilst Spine uses an emulated version of Object.create for a simulated class system
- Spine doesn't require underscore as a dependancy

Sammy.js

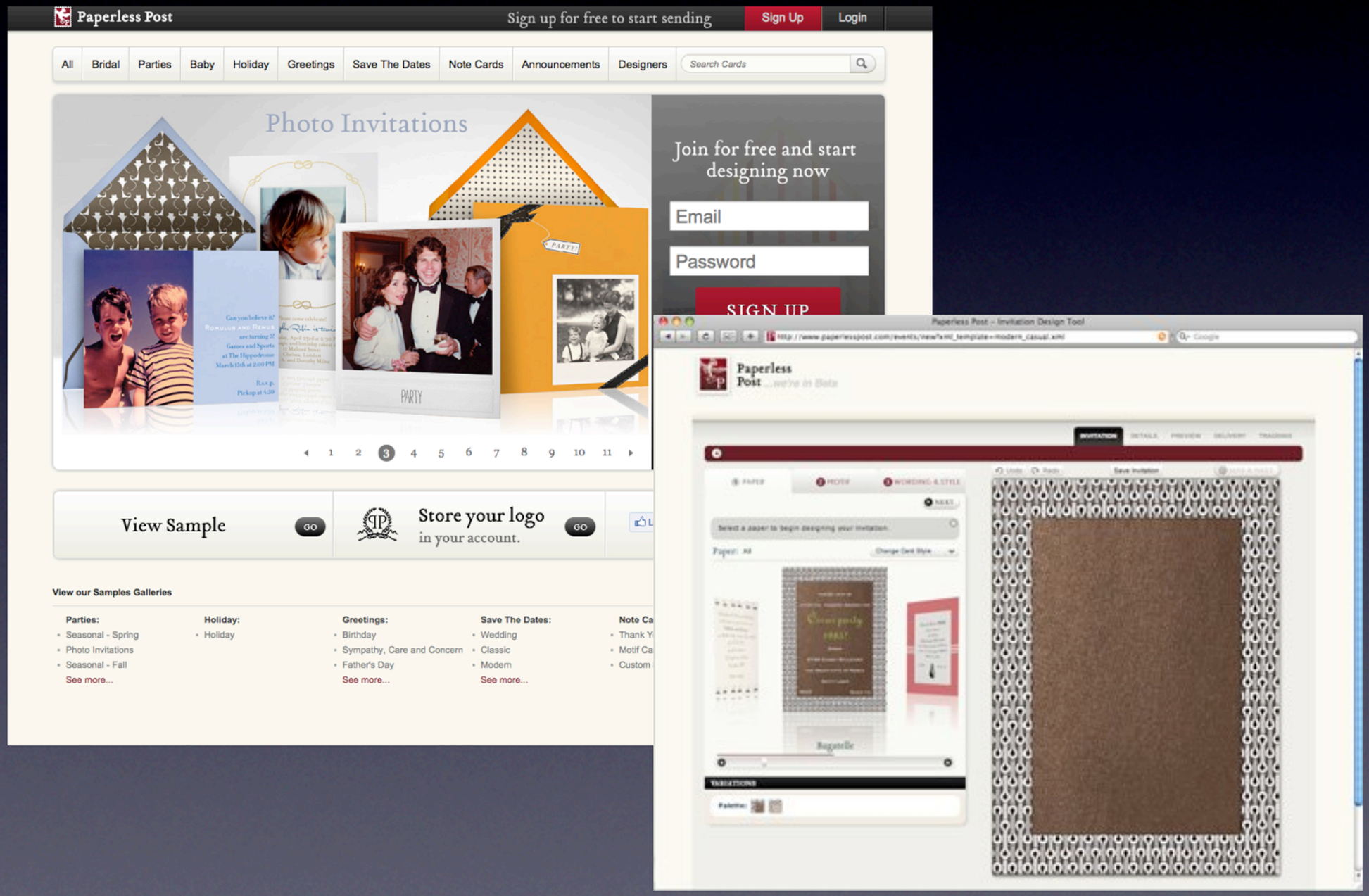
Lead: Aaron Quint

- A popular lightweight framework that allows you to easily define route based applications.
- Some consider it the best controller framework but it doesn't provide the Model and View aspects itself.
- Good for SPAs requiring code organization.
- Also used by apps such as PaperlessPost

Sammy.js Example

```
//initialize the application
var app = $.sammy(function() {

    //set the plugin for providing create/render client side templates
    this.use(Sammy.Template);

    //handler for a page that doesn't exist or can't be located
    this.notFound = function(verb, path) {
        this.runRoute('get', '#/404');
    };

    //define a route for 'about'
    this.get('#/about', function() {
        //partial calls template() because of the file extension
        this.partial('templates/about.template', {}, function(html) {
            $('#page').html(html);
        });
    });

    //define a route for '404' pages
    this.get('#/404', function() {
        this.partial('templates/404.template', {}, function(html) {
            $('#page').html(html);
        });
    });

});
```

PaperlessPost



SproutCore

Lead: Charles Jolley

- Recommended for applications wishing to have a desktop-like ‘richness’.
- Data binding framework makes updating views easy
- Extends MVC to include a server interface, a display that 'paints' your interface and responders for controlling application state.
- Often used for larger enterprise apps, but may be a little overkill for smaller ones.
- Apple used SproutCore for MobileMe

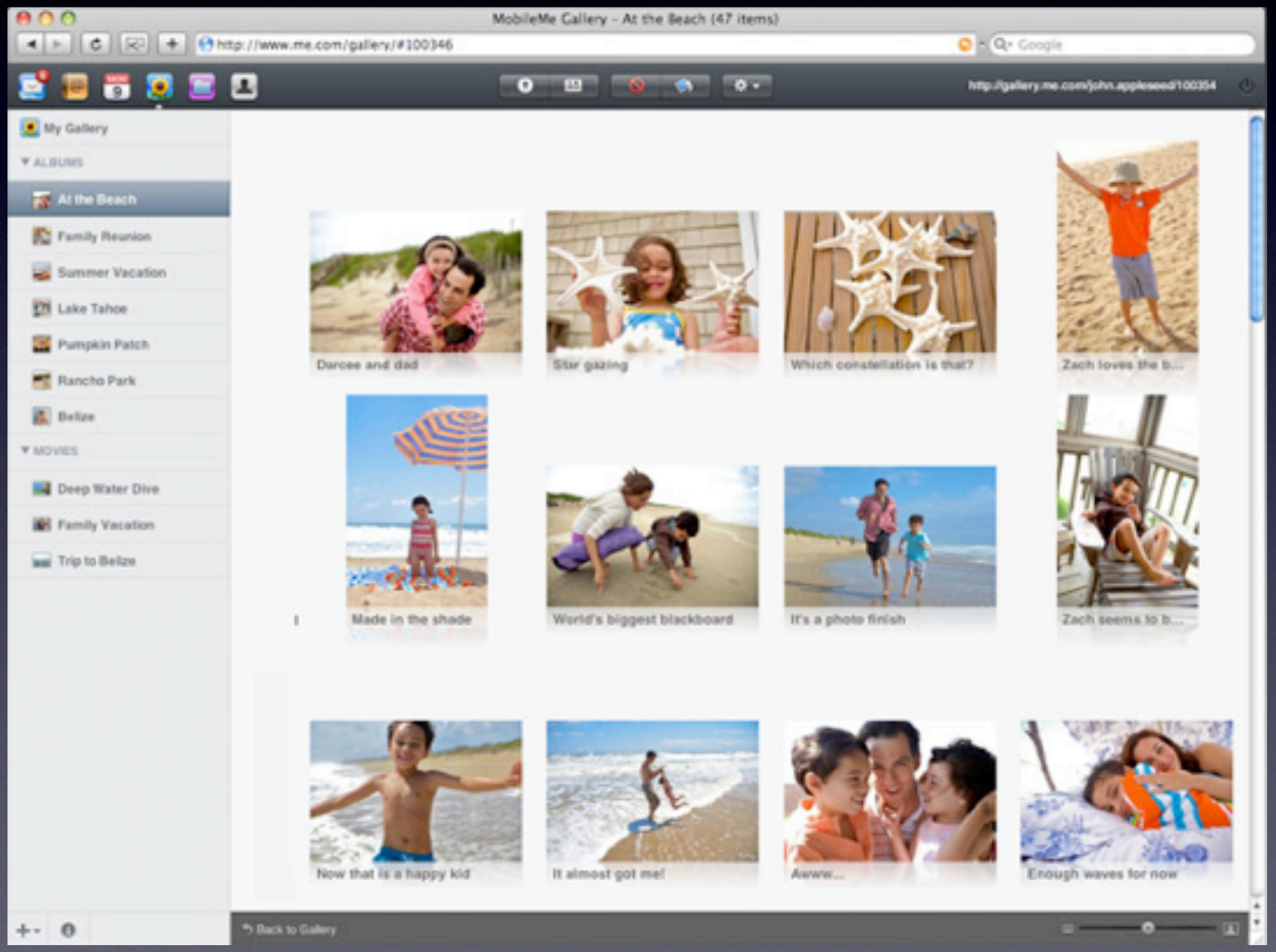
SproutCore Class + View

```
//Root object for the application. SC.Application is usually needed if you
//wish to use SC.Responder later to route events.
Todos = SC.Application.create();

//SC.Object is the root class for most classes defined by SproutCore.
Todos.Todo = SC.Object.extend({
    title: null, isDone: false
});

//The Todo App View
Todos.CreateTodoView = SC.TextField.extend({
    insertNewline: function() {
        var value = this.get('value');
        if (value) {
            Todos.todoListController.createTodo(value);
            this.set('value', '');
        }
    }
});

//The 'completed' View
Todos.MarkDoneView = SC.Checkbox.extend({
    titleBinding: '.parentView.content.title', valueBinding: '.parentView.content.isDone'
});
```

SproutCore Controller

```
//Our Todo Controller (for the todo list)
Todos.todoListController = SC.ArrayController.create({
    // Initialize the array controller with an empty array.
    content: [],
    // Creates a new todo with the passed title, then adds it to the array
    createTodo: function(title) {
        var todo = Todos.Todo.create({
            title: title
        });
        this.pushObject(todo);
    }
});

SC.ready(function() {
    Todos.mainPane = SC.TemplatePane.append({
        layerId: 'todos',
        templateName: 'todos'
    });
});
```

MobileMe


MobileMe Gallery - At the Beach (47 items)
http://www.me.com/gallery/#100346
Google
http://gallery.me.com/john.appleseed/100354
My Gallery
ALBUMS
At the Beach
Family Reunion
Summer Vacation
Lake Tahoe
Pumpkin Patch
Rancho Park
Belize
MOVIES
Deep Water Dive
Family Vacation
Trip to Belize
Darcee and dad
Star gazing
Which constellation is that?
Zach loves the b...
Made in the shade
World's biggest blackboard
It's a photo finish
Zach seems to b...
Now that is a happy kid
It almost got me!
Awww...
Enough waves for now
Back to Gallery

MVVM (Model View-ViewModel) pattern

- Architectural pattern based on the Presentation Model pattern
- Influenced by MVC but targeted at User-interface developers specifically
- ViewModel responsible for exposing data from Models, but considered more Model than View.

KnockoutJS

Lead: Steve Sanderson

- Knockout catered to those using JS for user interfaces, perfect for MVVM usage.
- Provides elegant dependency management, templating and works well with jQuery.
- Uses Observables (variation of pub/sub)
- May require a learning curve to get around the heavy use of data-binding.

Knockout Example

```
//A Simple Todo Application

<!--Markup with inline data-binding (Your View)-->
<form data-bind="submit: addItem">
    New todo: <input data-bind='value: itemToAdd, valueUpdate: "afterkeydown"' />
    <button type="submit" data-bind="enable: itemToAdd().length > 0">Add</button>
    <select multiple="multiple" width="50" data-bind="options: items"> </select>
</form>
```

```
//JavaScript: Model
var viewModel = {};
viewModel.items = ko.observableArray(["Get Groceries", "Wash Car"]);
viewModel.itemToAdd = ko.observable("");
viewModel.addItem = function () {
    if (viewModel.itemToAdd() != "") {
        //Adds the todo item. Modifying the "items" observableArray causes any
       //associated UI components to be updated.
        viewModel.items.push(viewModel.itemToAdd());
       // Clears the text box as it's bound to the "itemToAdd" observable
        viewModel.itemToAdd("");
    }
}

ko.applyBindings(viewModel);
```

Backbone? Spine? I'm worried..

Don’t be! They’re easier to use than you think.

Look, it’s a freakin rainbow!

How about patterns for your implementation code?

Code Design Patterns

Creational

Deal with object creation mechanisms (eg. singleton)

Behavioural

Identify common communication patterns between objects (eg. pub/sub)

Structural

Ease design by identifying ways to realise relationships between entities (eg. facade)

What's commonly used?

- Module pattern
- Observer (pub/sub) pattern
- Inheritance pattern
- Prototype pattern
- Sandbox pattern

Module Pattern

- Allows us to have particular methods & variables which are only accessible from within the module
- In JavaScript, it's used to emulate the concept of classes so we can include public/ private methods and variables inside a single object
- Shields functionality from the global scope

Example

The **counter** variable is actually fully shielded from our global scope so it acts just like a private variable would - its existence is limited to within the module's closure. We can call incrementCounter() or resetCounter() but can't directly access **counter**

```
var testModule = (function(){
    var counter = 0;
    return {
        incrementCounter: function() {
            return counter++;
        },
        resetCounter: function() {
            console.log('counter value prior to reset:' + counter);
            counter = 0
        }
    }
})();

/*test*/
testModule.incrementCounter();
testModule.resetCounter();
```

Pub/Sub (Observer) Pattern

- General idea is the promotion of loose coupling
- Objects subscribe to a specific task or activity of another object and are notified when it occurs
- Subscribers are observers and we refer to the object being observed as the publisher
- Publishers notify subscribers when events occur.

jQuery Pub/Sub Example

```
$.subscribe( "/search/query", function( term, mode, order ) {
  console.log( term + mode + order );
  //perform a search request
})

$.publish( "/search/query", [ "cats", "images", "recent" ] );
// logs: 'cats images recent'

//unsubscribe handler
$.unsubscribe( "/search/query", handle );
```

* uses Ben Alman's jQuery pub/sub implementation

Pub/Sub implementations

- AmplifyJS
- PubSubJS
- Ben Alman's implementation
- Peter Higgin's version
- jsSignals
- OpenAjaxHub
- jQuery custom events *

For more implementation patterns read EJDP - it's free!

- Available at addyosmani.com
- Downloaded over 103,000 since release
- Compact - Less than 50 pages.

It's Tool Time!

Tools & Micro-frameworks

- Routing / Browser state management
- Cross-browser storage
- Client-side templating
- Feature detection
- Widgets/Components
- Dependancy management

Browser State Management
The problem

- Bookmarkable URLs traditionally only possible with a hard page-refresh
- For modern apps we want to preserve this without any refresh involved
- When the URL changes application state needs to as well and visa versa.
- Users want unique URLs that can be shared

Browser State Management
The solution

- Solution is to change the URLs
  location.hash (eg. http://twitter.com/#!/jquery)
- Lots of tools that can assist in adding manageable routing to your applications via this technique
- Make URLs SEO-friendly via the Google AJAX Crawlable specs

Routing Options

- History.js
- RouteMap
- jQuery BBQ and jQuery.hashchange
- jQuery Routes
- Even an @140bytes version by @jed

jQuery.BBQ

Lead: Ben Alman

```
$("a").click(function(e){
    e.preventDefault();
    var href = $(this).attr( "href" );
    // Push this URL "state" onto the history hash.
    $.bbq.pushState({ url: href });
  });

$(window).bind( "hashchange", function(e) {
    var url = $.bbq.getState( "url" );
    //additional logic depending on state
});

$(window).trigger( "hashchange" );
```

RouteMap

```
var routes = window.RouteMap, rules, rule;
$(window).bind('hashchange', routes.handler);

rules = {
        load_main:      {route: '/',          method: 'load'},
        load_item:      {route: '/item/:id',  method: 'load_item'}
    };

for (rule in rules){
    if (rules.hasOwnProperty(rule)){
        routes.add(rule);
    }
}
```

jQuery Routes

```
$.routes({
  “/”:              “Index#home”,
  “/contact”: “Index#contact”,
  “/about/(:page_name)”: function(params){
  $(‘.about’).slideDown();
 }
});
```

Cross-browser Persistent Storage

- Store.js (localStorage, globalStorage, userData)
- AmplifyJS (localStorage, sessionStorage, globalStorage, userData)
- PersistJS (local, session, global, userData, gears)
- jQuery Offline plugin

Store.js Example

```
//store key/value pairs
store.set('country','new zealand');

//get the value of the 'country' entry
store.get('country');

//remove the 'country' entry
store.remove('country');

//clear all of the keys
store.clear();

//storing object literals
store.set('profile', { username: 'johnsmith', website:'http://
johnsmith.com'});

//we can then get the stored object values
var userProfile = store.get('profile');
console.log(userProfile.username + ' has a website as ' +
userProfile.website);
```

Client-side Templating

- Mark-up with expressions where the template can be applied to data objects or arrays and is rendered into the DOM.
- More readable than traditional string concatenation
- Make it significantly cleaner to define a 'template' for data you are outputting to the browser

Benefits

- Rendered and cached without an HTTP request to the server - very fast
- Separation of concerns
- More intelligent layouts supporting conditions, variables and more.

Templating Options

- Mustache.js
- Handlebars.js
- Dust.js
- jQuery.tmpl plugin
- Underscore Micro-templating
- PURE

jQuery.tmpl + pub/sub example

```
<script id="userTemplate" type="text/x-jquery-tmpl">
    <li>${user}</li>
</script>

<script type="text/javascript">
(function($) {
var userList  = [];
    $.subscribe( "/new/user", function(userName){
        if(userName.length){
            userList.push({user: userName});
            $( "#userTemplate" ).tmpl( userList ).appendTo
( "#users" );
        }
    });
    $('#add').bind('click', function(){
        var strUser    = $("#username").val();
        $.publish('/new/user', strUser );
    });
})(jQuery);
</script>
```

Feature Detection

Libraries that help detect compatibility for emerging web technologies (HTML5, CSS3)

- Modernizr + yepnope
- has.js
- Head.JS

Modernizr

Lead: Paul Irish

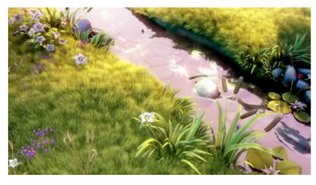
- Detects browser support for CSS3 features like @font-face, border-radius and HTML5 features like audio, video etc.
- Easily test for feature support via JavaScript
- Adds CSS classes for each feature detected so you can style based on support as well
- Widely used in the industry

Modernizr <video> test

```
if (Modernizr.video && Modernizr.video.ogg){
        // preload ogg video assets
}else{
        //  load flash fallback
}
```

has.js equivalent

```
(function(has, addtest, cssprop){

    var video = document.createElement("video");

    addtest("video", function(){
        return has.isHostType(video, "canPlayType");
    });

    addtest("video-ogg-theora", function(){
        return has("video") &&
          video.canPlayType('video/ogg; codecs="theora, vorbis"');
    });

})(has, has.add, has.cssprop);

if(has("video-ogg-theora")){
   // load ogg assets
}else{
  // load flash fallback
}
```



yepnope.js


Leads: Alex Sexton, Ralph Holzmann

```
// conditional polyfill loading (readup on Prefixes too!)
yepnope({
  test: Modernizr.geolocation,
  yep: 'geolocation.js',
  nope: ['geolocation-polyfill.js'],
  callback: function (url, result, key) {
  }
});


// if the ie version matches, load the patch
yepnope({
  load: ['regularapp.js', 'ie7!ie8!patch.js']
});
```

Widgets/UI Components

- jQuery UI (grid, spinner, menus coming)
- Wijmo (open + commercial widgets)
- jQuery Tools (not updated as regularly)
- KnockoutUI
- NinjaUI (new, not as wide a collection as the others)
- Roll your own

If you're after templated widgets it might be worth looking at dojo as it addresses this quite well

Dependancy Management

- JS files loaded with the *script* tag can be blocking in nature
- Few modern browsers support reliable parallel loading
- For apps requiring many files, there's no easy way to manage dependancies

Script Loader Benefits

- Very useful for loading scripts in a specific order or dynamically depending on need
- Can ensure script dependancies have loaded before executing behaviour that relies on it being present
- Some can assist with more (eg. structure, conditional polyfill loading)

Script Loader Options

- LAB.js (Kyle Simpson)
- RequireJS (James Burke)
- StealJS (Justin Meyer)
- ControlJS (Steve Souders)
- yepnope.js (conditional polyfill loading)
- HeadJS

LabJS

Lead: Kyle Simpson

- Works best where scripts just need to be efficiently loaded in a particular order
- More lightweight than RequireJS
- It's a mature solution that also includes options for loading local scripts via Ajax, preserving specific order and more.

Example

```
<script>
    $LAB
    .setOptions({AlwaysPreserveOrder:true}) //optional
    .script("jquery-min.js").wait()
    .script("jquery.plugin.js")
    .wait(function(){
        $(function(){
         myplugin.init();
        });
    });
</script>
```

RequireJS

Lead: James Burke

- Recommend for modular code
- Modules attempt to limit their impact on the global namespace and be more explicit about mentioning their immediate dependencies.
- RequireJS also comes with an optimization tool that allows you to combine and group your scripts into a smaller set of minified scripts that load quickly **

Example 1: Basic Usage

```
<script data-main="scripts/main" src="scripts/require-jquery.js"></script>

<script>

//main.js (/main above) contains our require calls
require(["jquery", "jquery.thickbox","jquery.tmpl"], function($) {
    //plugins loaded, execute additional behaviour
    $(function() {
        $('.items').doStuff();
    });
});

</script>
```

Example 2: Modules

```
//modules are well-defined objects that avoid polluting the global namespace.
//my/boxers.js has the dependancies shoppingcart.js and stockcheck.js

define(["./shoppingcart", "./stockcheck"], function(shoppingcart, stockcheck)
{
        //return an object to define the "my/boxers" module.
        return {
            color: "pink",
            pattern: “plaid”,
            size: "extra-large",
            addToCart: function() {
                inventory.decrement(this);
                cart.add(this);
            }
        }
    }
);
```

Testing? If you're not sure about something, rub it against a piece of paper.

Testing

- Unit testing
- Ajax, Browser and coverage testing
- Scripting environment

Unit Testing

- Essential for automated testing of any serious web application.
- Manual functional testing is great from a user-interface perspective
- But..unit testing allows stress-testing to discover if all the inner-workings of it perform as intended.

Unit Testing options

- QUnit
- Jasmine (BDD)
- FuncUnit
- Crosscheck
- JSSpec (BDD)
- jsTestDriver

QUnit Example

```
//For below, remember to include markup for your QUnit test output template along
with jquery, qunit.js and qunit.css as these are required.

$(function(){
  test("A very basic test", function() {
    ok( true, "this test passes fine" );
    var value = "hello world";
    equals( "hello world", value, "We expect value to be hello world" );
  });

  module("Module A");
  test("first test within a module", function() {
    ok( true, "all pass" );
  });

  module("Module B");
  test("Another test", function() {
    expect(2);
    equals( true, false, "failing test" );
    equals( true, true, "passing test" );
  });
});
```

Ajax, Browser and Coverage testing

- MockJax - intercept and simulate ajax requests with a minimal impact on changes to production code
- WebDriver + Watir + jsTestDriver - automated browser testing
- JSCoverage - useful for discovering areas of your code which may either be broken or simply not being correctly

Scripting Environment

- Zombie.js - lightweight framework for testing client-side JavaScript code in a simulated environment
- Envjs - provides a JS implementation of the browser as a usable scripting environment
- Bumblebee - combines Rhino, JSpec, Envjs and Ant to provide an "out of the box" JavaScript testing toolkit

Pack it all up!

Build Process

- Quick intro to Ant
- Concatenation
- Minification

Ant

- Excellent Java library that assists with defining build files and targets
- Supports adding in many ready-made task-types (easily loop through directories, define filters, concatenate etc.)
- Can be used with Google's Closure compiler, YUI compressor and others.
- Used by many open-source projects

Ant: JSLint/Hint Validation

```
<target depends="-init" name="-js.lint">
    <pathconvert pathsep=" " property="jsfiles">
        <fileset dir="${build.dir}/js/">
            <include name="*.js"/>
        </fileset>
    </pathconvert>
    <exec dir="${build.dir}/js/" executable="java"
failonerror="true">
    <arg line="-jar ${js.jar} ${jslint.js} ${jsfiles}"/>
    </exec>
    <echo>Finished</echo>
  </target>
```

Ant: Concatenation

```
<target name="-js.concatenate" depends="-js.lint"
        description="Concatenates specified JavaScript files">
        <concat destfile="${build.dir}/js/concat.js">
            <fileset
                dir="${src.js.dir}"
                includes="*.js"
                excludes="first.js, second.js"/>
        </concat>
        <echo>Finished</echo>
</target>
```

Ant: Minification

```
<target name="-js.minify" depends="-js.concatenate"
        description="Minifies JavaScript files">
        <apply executable="java" parallel="false" dest="${build.dir}/js">
            <fileset
                dir="${build.dir}/js"
                includes="concat.js"/>
            <arg line="-jar"/>
            <arg path="${yui.dir}"/>
            <srcfile/>
            <arg line="-o"/>
            <mapper type="glob" from="*.js" to="*-min.js"/>
            <targetfile/>
        </apply>
        <echo>Finished</echo>
</target>
```

Reminder

- Check out the HTML5Boilerplate build file
- It's awesome and is a perfect example of how a front-end project should be built
- Find it on GitHub https://github.com/paulirish/html5-boilerplate/wiki/Build-script

Concatenation

- Web applications typically use a number of JavaScript files (app, libraries, plugins etc)
- Concatenation provides you a single file which can then be minified
- Scripts can be concatenated in any order
- Ant supports I/O operations like appending arbitrary files (eg. licenses) and string replacement as well

Concatenation Options

- Jake
- Sprockets
- Closure Compiler
- Smasher
- YUI Compressor
- Minify

Minification

- It's a critical part of the build process for any large JavaScript application.
- Every extra byte counts, which is why you should ideally minify your code rather than just including the unminified version
- The less time it takes to load, the quicker your page will be ready to use

Minification Options

- Closure Compiler
- YUI Compressor
- UglifyJS *
- JSMin
- Microsoft Minifier
- Dojo's ShrinkSafe

And that's it!

Conclusions

- jQuery is awesome
- These tools can help make your development process significantly easier
- Most are well-tested, quick wins to use.
- Spend more time on the project, less on architecture.

For more on me

- @addyosmani on Twitter or GitHub
- http://addyosmani.com for JS + jQuery articles, screencasts and resources

It's the final slide!