Javascript client & server architectures

Pedro Melo Pereira

07/05/14

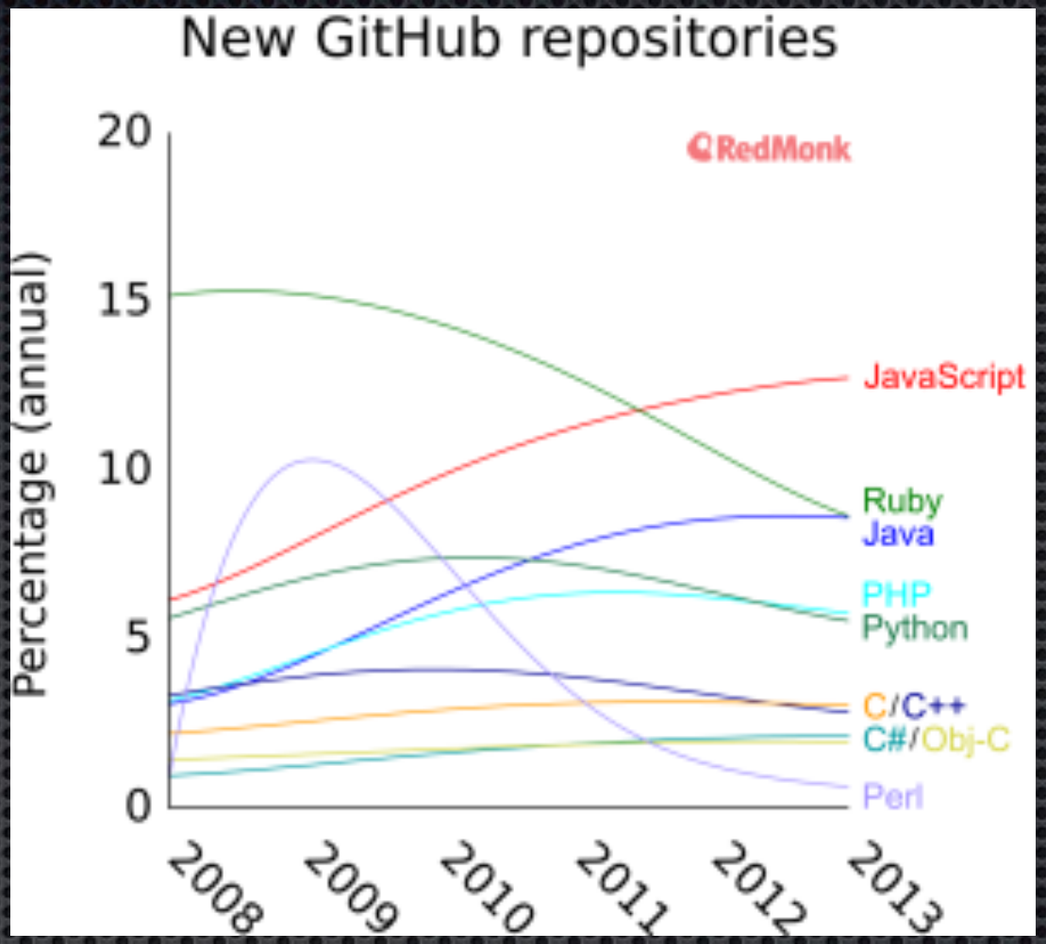
JS

“The World's Most Misunderstood Programming Language”

Douglas Crockford

- Name: It is not interpreted Java, and the “script” suggests it is not a real programming language
- It has Java / C like syntax but is is more in common with functional languages: Lisp / Scheme
- Real power comes from lambdas & closures
- Load & go delivery, dynamic typing and prototypal inheritance

trendy



- The rise of JS is undeniable
- Node.js is contributing to this hype
- Also, the number of JS plugins / frameworks is exploding

..but with problems

- Bad books in general. Too much focus on the DOM
- Design errors
- Early lousy implementations
- Bad reputation, viewed as an amateur’s web toy

History

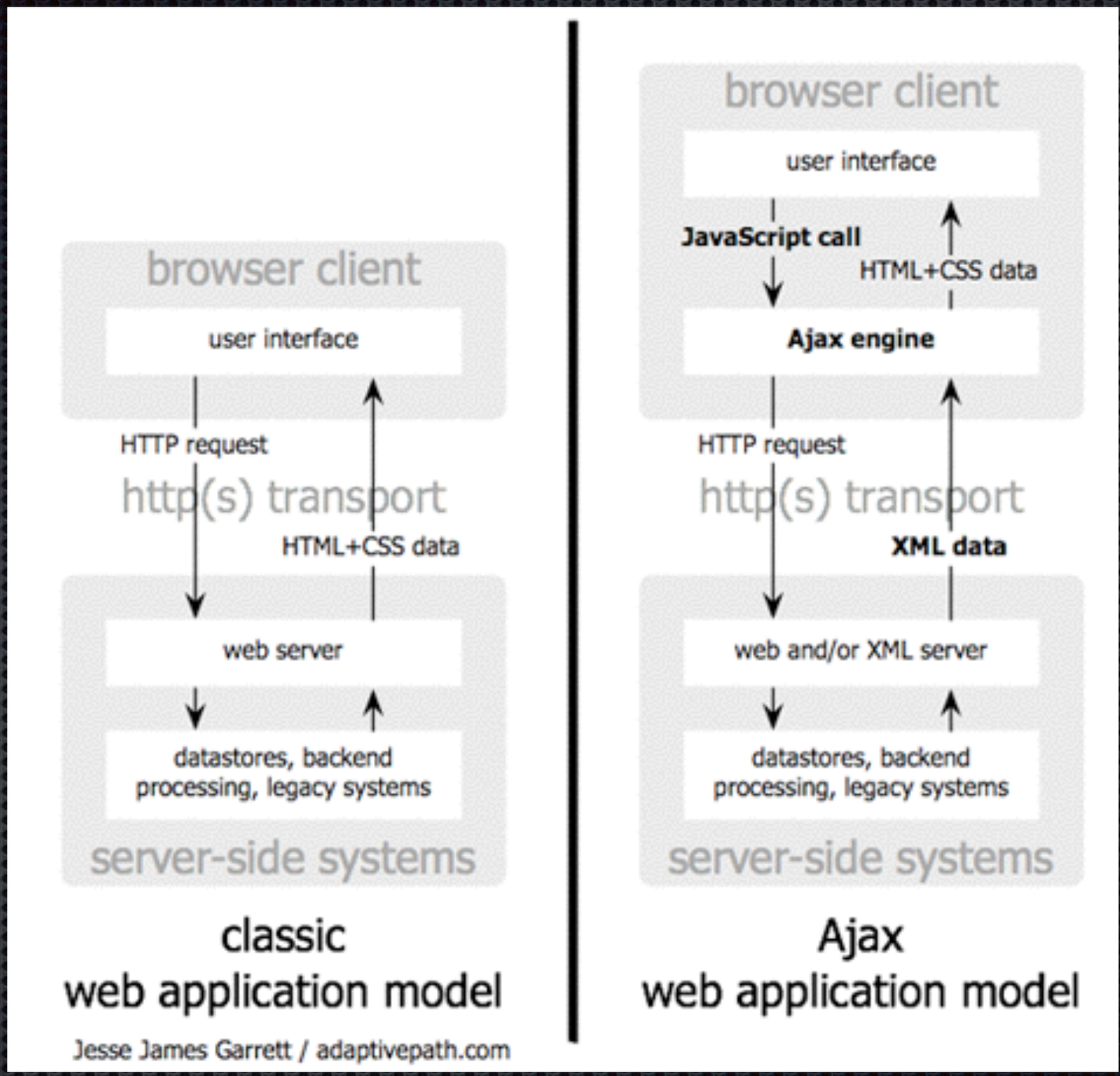
- 1992 - Oak, Jim Gosling @ Sun
- 1995
  - Sun: Oak becomes Java. HotJava web browser
  - Brendan Eich @ Netscape: Livescript, a lighter language for the web, it was to be a dialect of Scheme but adopted Java's syntax to compete with Sun
  - Alliance is formed between Sun and Netscape. Livescript is renamed to Javascript
- 1996 - JScript @ Microsoft, same language different implementation. Iframe tag introduced in IE to load content asynchronously
- 1998 - ECMAScript is born to prevent Microsoft from changing the language. Eich helps found Mozilla

- 1999: XHR introduced in IE5 through ActiveX control
- 2003: Single Page Applications - the next big thing
  - Serve a single page for your web application
  - Depart from full page reloads, allowing partial page updates. Provide richer experience
    - Pages were the old days of static web data. How to bring desktop-like apps to the web ?
- 2004: Google launches first cross browser XHR application - Gmail. Ruby on rails arrives at the scene.
- 2005: Ajax term coined based on Google's technique. Google Maps is launched

Asynchronous Javascript and Xml

- Getting a new page from the server on each interaction was when the web was just a hypertext medium
- With Ajax you only change parts of an existing page
- Pros:
  - More responsive sites. Saves page load time
- Cons:
  - Increased complexity. More code = less performance
  - Breaks the “back” button

Ajax life cycle



JQuery

- 2006: JQuery revolution.
  - Changed the way people write JS. Provides easy DOM manipulation via selectors, useful utilities and plugins
  - Today the most used feature of JQuery, the selector engine, was release on a separate open source project: Sizzle
  - What really changed the game was the cross browser support that freed the developer of worrying about all the different browser quirks

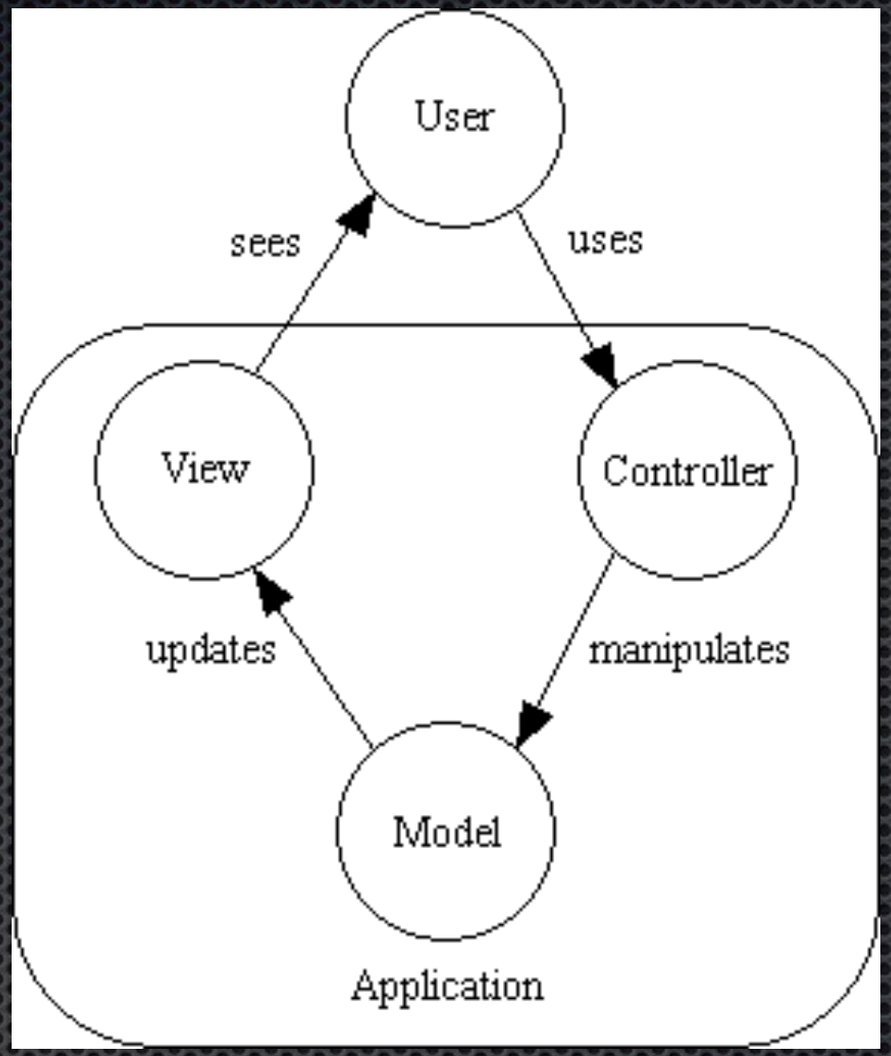
JQuery

- Pros
  - Large community. Good documentation / tutorials
  - Eliminate cross browser javascript issues
  - Easy DOM manipulation. Do better things in much less code
- Cons
  - There is always another magical plugin to use. It discourages attempts to understand the underlying problems and often introduces brittle dependencies into the codebase
  - Sometimes JQuery code could be done in vanilla JS with less overhead

MVC

Model

- Manages data
- Defines domain logic
- Independent



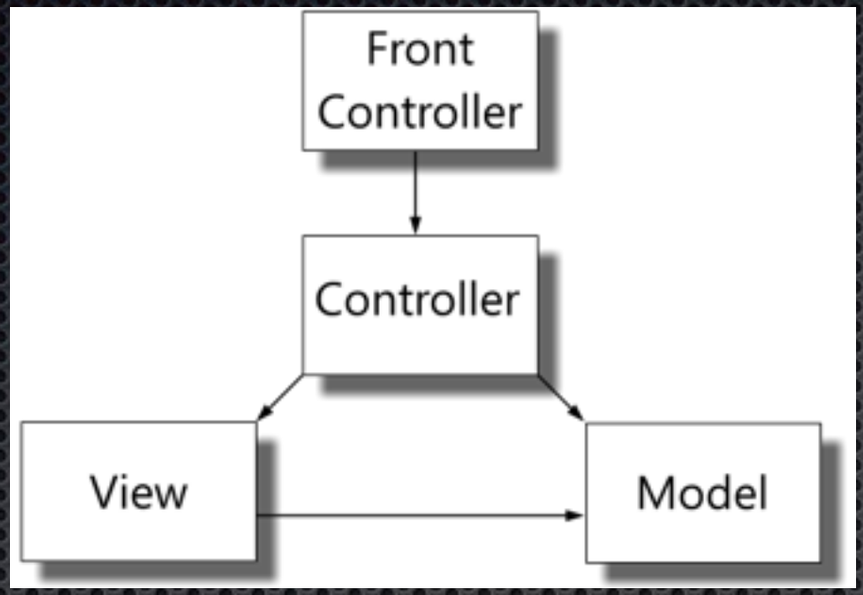
View

- Interface to view and modify the data
- Sends user actions to the controller
- Depends on the model

Controller

- Interprets user actions. Provides model data to the view
- Defines application logic
- Depends on the view and the model

Web-based MVC



- This pattern emerged naturally as object-oriented design was applied to the stateless nature of the HTTP protocol
- The front controller handles common infrastructure concerns (ex: security, session state management, dependency injection) and dispatches requests to individual controllers
- New needs:
  - Routing inbound requests
  - Server-side input processing
  - Views as pre-processed server side templates or objects that encapsulate content
  - Manual data-binding from the model to the view emerged

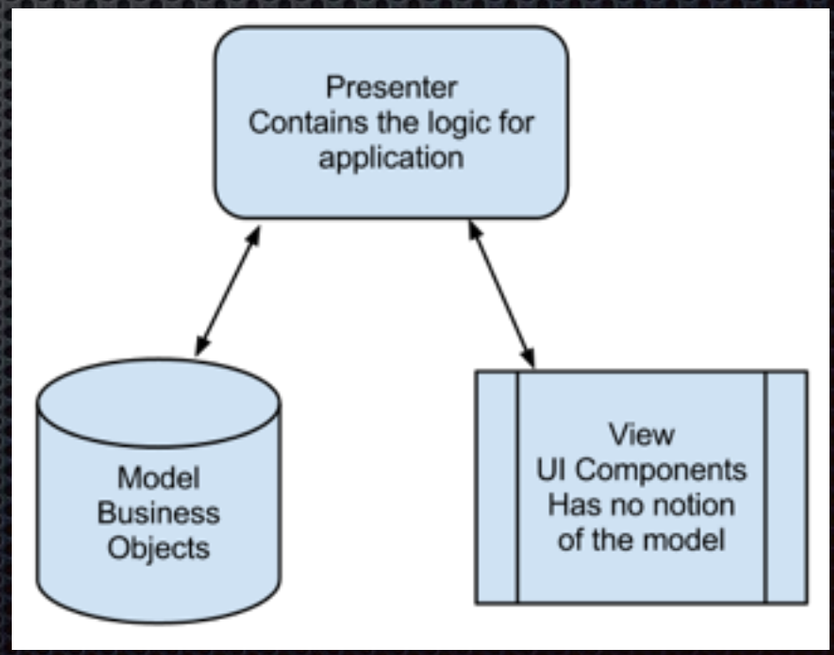
Model View Presenter

This led to an evolution of the classical MVC

- Key ideas
  - While in MVC all three components can talk to each other in MVP there are 4 components whose responsibilities are clearly separated: view / view interface / presenter / model
  - By isolating each components it is easier to unit-test
  - The presenter is responsible for binding the model to the view

MVP has 2 variants:

- Passive view
  - The view is independent from the model
- Supervising controller
  - Like MVC, the view depends on the model



Model View ViewModel

- MVVM originated from Microsoft and it was targeted at UI development platforms that support event-driven programming in WPF and Silverlight on the .NET framework using XAML
- Technically different but similar, this pattern is available in HTML5 through Angular.js, Knockout.js and in Java through the ZK Framework

- MVC differences
  - The ViewModel mediates between the model and the view by defining a model for the view which acts as a target for view data bindings
  - The controller is replaced by a framework binder, whose role is to free the developer from having to write boilerplate code to synchronize the model from the view
  - MVVM facilitates the separation from the development of the GUI from the backend

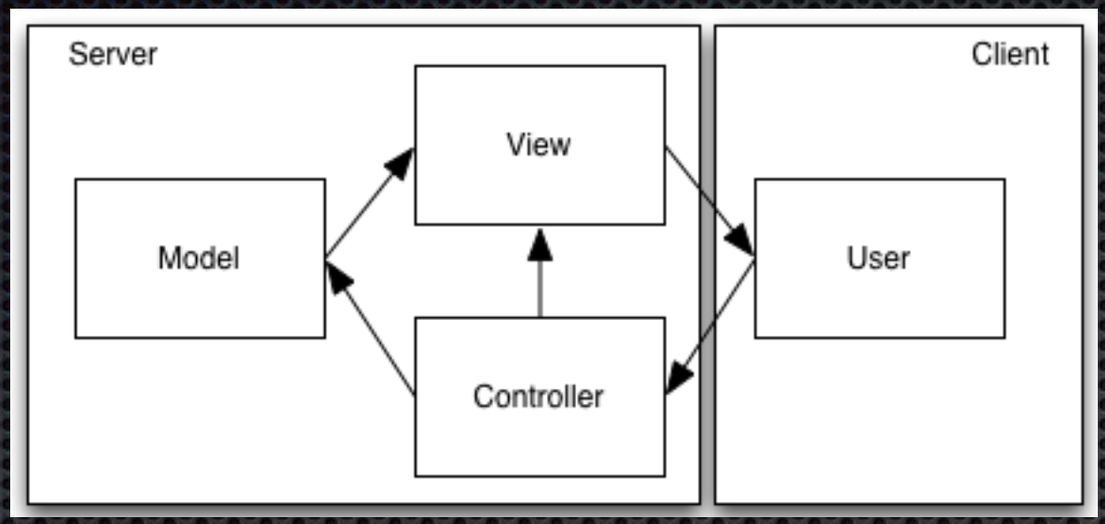
Recent history

- 2008
  - SPA's were all the rage
  - Lots of JS libraries fighting for the pole position: JQuery, Mootols, Prototype, Dojo, YUI
  - Quite some tension between the SOAP and REST camps
  - JSON was still a new thing, XML was the norm
  - Ruby on Rails reached maximum hype, everyone was writing their own rails-like framework in their favorite language
  - IE6 was still a plague but rapidly vanishing
  - Websockets were forecasted as futuristic, not really for adoption in years to come
  - Relational databases were the only option

Fast forward to today

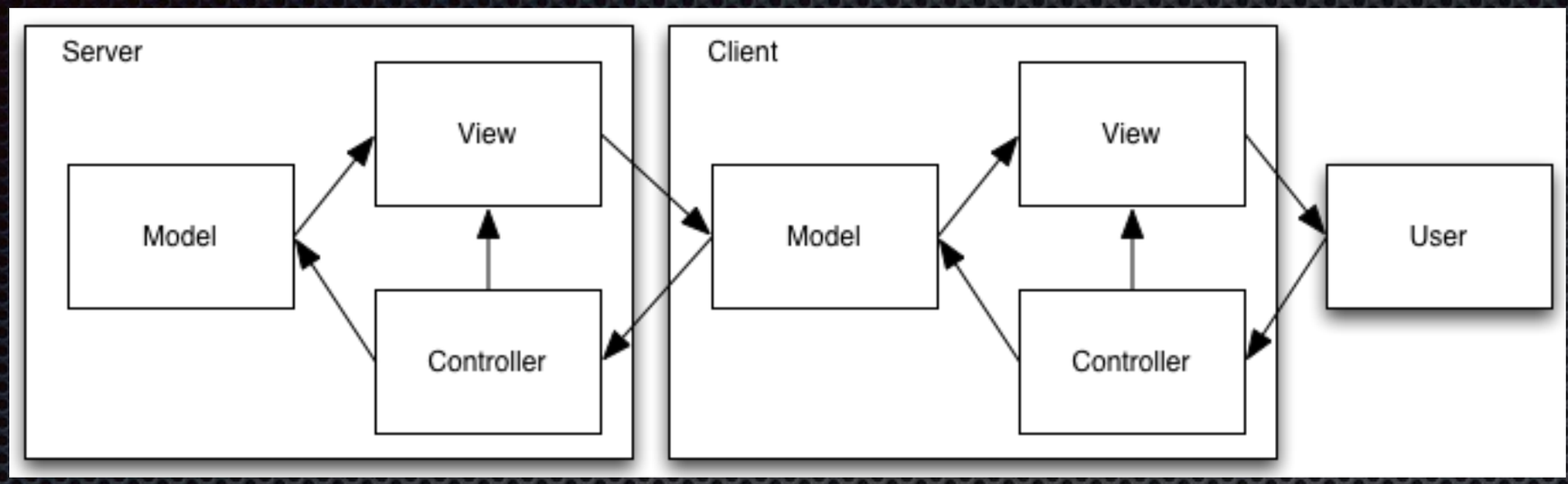
- 2014
  - JQuery won
  - JSON won
  - REST won
  - Most SPA's are all thin server now
  - IE doesn't matter like it used to
  - Javascript is the new hype. Full stack JS is possible
  - Preprocessors are not to be ignored: Coffeescript, Dart, Typescript
  - Node.js ecosystem is huge
  - Geeks are all using NoSql

Thin Client Architecture - Server-side MVC



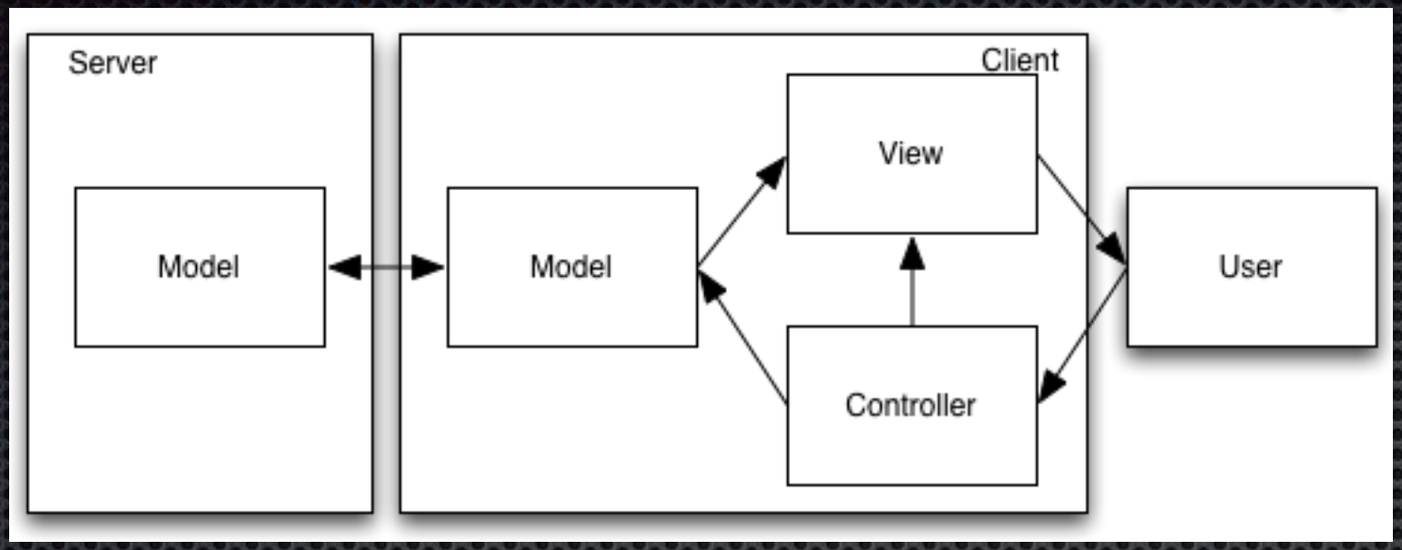
- Initial SPA's placed the entire MVC on the server
- The view, either pages (ex: JSP) or templates (ex: Velocity, Freemarker) is processed on the server which generates HTML that is shipped to the client
- Even if Ajax is used, the experience is not optimal because some things are asynchronous while others require a new page load. We are always dependent on the server for rendering, hence the full post back - render cycle
- Lots of ugly JS code everywhere will bring a maintainability nightmare

Thickening Clients - Client / Server MVC



- By using a Javascript MVC framework, we optimize the JS code. Meaning that it becomes more cohesive and maintainable
- No more manual data binding is needed because view updates are automatically binded to the client model
- Why complicate ? Shouldn’t we use the server for what is really needed ? Does it make any sense to still do Server-side view rendering ?

Thin Server Architecture - Rich Internet Applications



- Move page rendering logic on to the client
- By using REST we take advantage of proxied requests
- Drastically reduce server load
- Essentially, keep only the database-centric tasks on the server: Persistence, Authentication, Authorization, Validation
- Exploit HTML5 api's for client side storage
- Solution: Initial page load render on the server, all further renders on the client. The server returns JSON to the client

JS MVC Framework Jungle

TodoMVC

Helping you **select** an MV* framework

Download (1.2) | View on GitHub | Blog

Introduction

Developers these days are spoiled with choice when it comes to selecting an **MV* framework** for structuring and organizing their JavaScript web apps.

Backbone, Ember, AngularJS... the list of new and stable solutions continues to grow, but just how do you decide on which to use in a sea of so many options?

To help solve this problem, we created TodoMVC - a project which offers the same Todo application implemented using MV* concepts in most of the popular JavaScript MV* frameworks of today.

Follow | Tweet 2,055 | +1

JavaScript Apps

Backbone.js R	Polymer R	PureMVC R	SAPUI5 R
AngularJS R	React R	Olives	Exoskeleton R
Ember.js R	cujoJS	PlastronJS R	Atma.js R
KnockoutJS R	Montage	Dijon	Ractive.js
Dojo R	Sammy.js R	rAppid.js R	ComponentJS R
YUI R	Stapes R	DeftJS + ExtJS	Vue.js R
Agility.js R	Epitome R	Aria Templates R	React + Backbone.js R
Knockback.js R	soma.js	Enyo + Backbone.js R	
CanJS R	DUEL		
Maria R	Kendo UI R	AngularJS (optimized) R	

Why JS MVC Frameworks ?

- Bring structure and organization to your projects, establishing a maintainable foundation right from the start
- If you are developing using just JQuery its very easy to create a JS app that ends up a tangled mess of selectors and callbacks, all desperately trying to keep data in sync between the HTML, the logic in your JS and calls to your API for data
- Avoid callback chaos

What is Angular.js ?

- Open source Javascript Framework, actively maintained by Google and the community
- A structural framework for dynamic web apps (SPA's, RIA's)
- Augments web apps with MVC capabilities
- Initial release was in 2009

Philosophy

- Built around the belief that declarative programming should be used for building web interfaces and wiring software components, while imperative programming is excellent for business logic

Who is using angular ?

- Google - Doubleclick
- Youtube - Leanback (also PS3)
- Alcatel / Lucent - Cloudband
- Goodfilms mobile site
- Plunker
- Localytics
- Lamborghini
- Mini
- NSNBC
- Netflix

Angular design goals

- Decouple DOM manipulation from application logic
- Regard app testing as equal importance to app writing. Testing difficulty is dramatically affected by the way the code is structured
- Decouple the client side of an app from the server side. This allows development work to progress in parallel and allows for reuse of both sides
- Guide developers through the entire journey of building an app, from designing the UI, through writing the business logic, to testing

Punchline

- Encourages loose coupling between presentation, data, and logic components. Using DI, Angular brings traditional server-side services, such as view-dependent controllers, to client-side web applications. Consequently, much of the burden of the backend is reduced, leading to much lighter web apps

Angular key features

- Lets you extend HTML's syntax to express behavior in a declarative way. HTML is the template
- Frees you from having to register callbacks
- Has two-way data binding that allows for the automatic synchronization of models and views
- Has dependency injection to help modularize JS code
- Supports form validation, routing, built-in testing, support for mocks

Angular structure

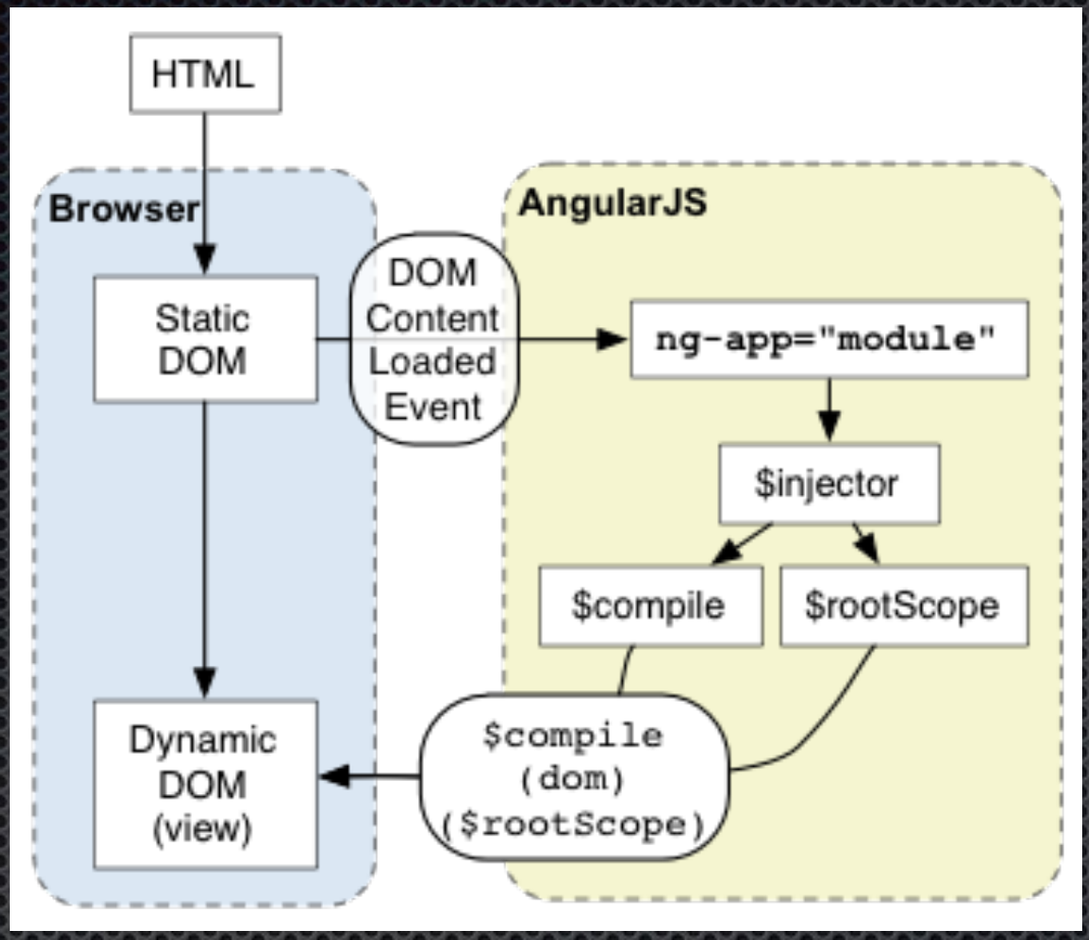
Architecture

- Angular is close to the MVVM pattern.
- It uses POJO objects for the model
- Views are binded to a $scope variable that corresponds to the view model

Dependency Injection

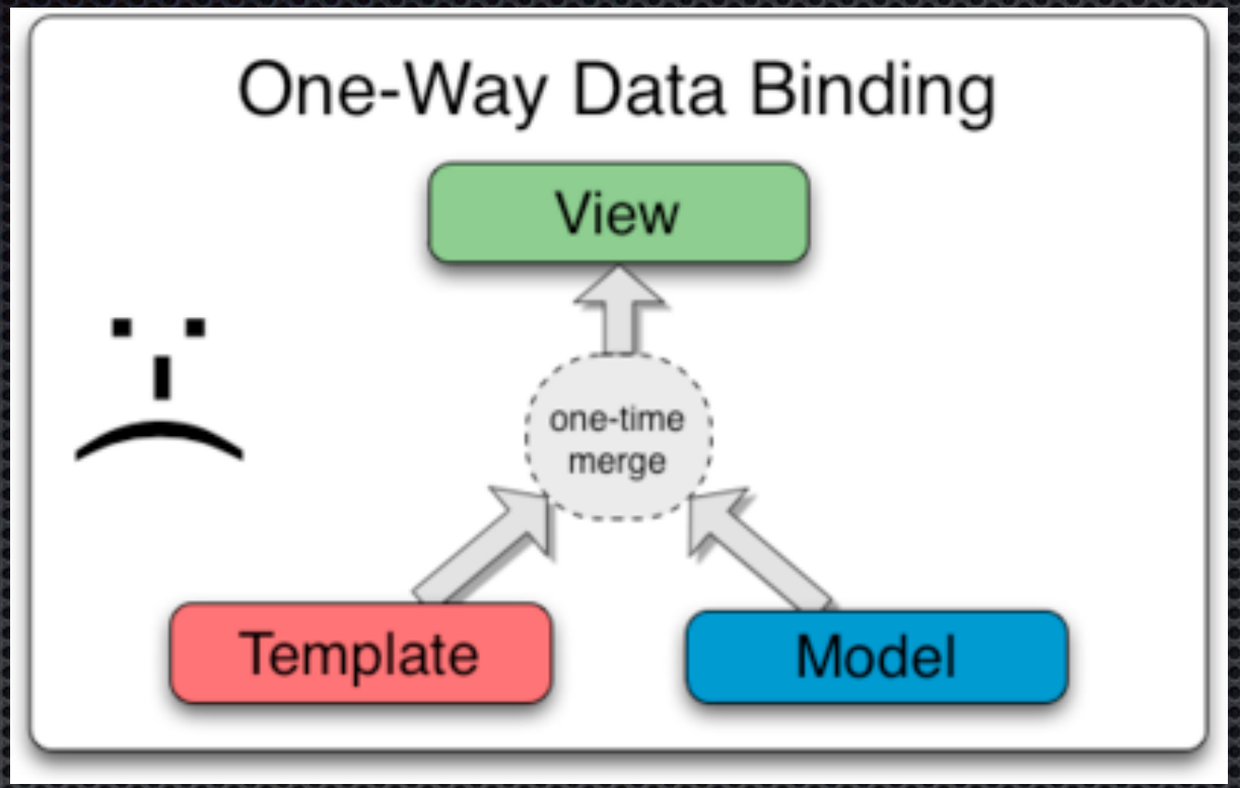
- Angular uses name-based injection, where an argument to a function is replaced by an instance of an object. Example: injecting services into controllers

Angular Initialization



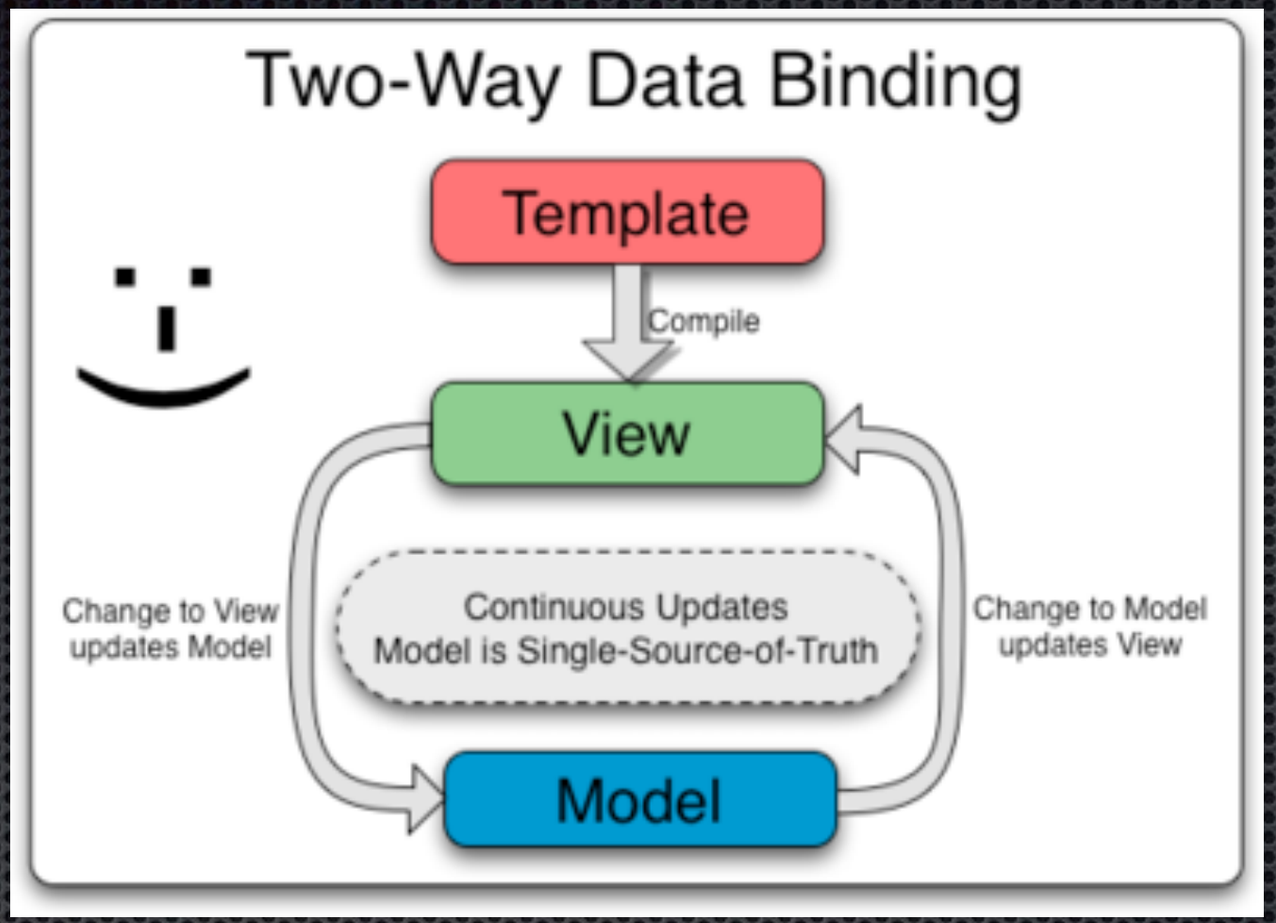
- Angular looks for the ng-app directive which designates your app root, then it :
  - loads the module associated to the directive
  - creates the app injector
  - compiles the DOM, treating the ng-app as the root of the compilation. This allows to treat a portion of the DOM as the angular app

One-way data binding



- Classical template systems bind in one direction, they merge
- After the merge, changes to the model are not automatically reflected in the view or vice-versa
- Code has to be written in order to manually sync the two

Two-way data binding



- Angular is different, it creates a live view by compiling the HTML template
- After that, changes in the view automatically update the model and vice-versa
- The view becomes a projection of the model, and contributes to separate concerns from the controller, which improves testability

Angular Concepts

- Directives
  - Provide a way to extend HTML vocabulary to support new constructs
  - Only place where DOM manipulation is permitted and thus behavior is defined
  - Attached to: new tags, attributes, comments or even css classes
  - One of the most complex and powerful features of Angular

Angular Concepts

- Controllers
  - A controller is attached to the DOM via an ng-directive
  - Those DOM elements will share a common scope object
  - No presentation logic, controlled should contain only the business logic required for the view
  - Nested controllers have access to the parent scope

Angular Concepts

- Services
  - Organize and share code across an Angular app
  - Singletons
  - Lazily instantiated
  - Used in the app through DI

Angular Concepts

- Routes
  - A clientside router is present in which we activate html partial views based on the url
- Filters
  - Used to format a value from an expression
- Out of the box services
  - $http - promise-based access to HTTP requests
  - $resource - high level abstraction to access REST services
  - $location - parses URL and reflects changes in browser navigation
  - and many more..

Angular Tools

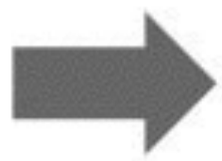
create a new webapp

yo webapp

OR
yo angular
yo angular:controller

handle dependencies

bower search
bower install



preview, test, build

grunt server
grunt test
grunt

Resources

http://ngmodules.org

What is Backbone.js ?

- A lightweight JS library, rather than a framework, that adds structure to your client-side code
- Provides models with key-value binding and custom events
- Collections with an API of enumerable functions
- Views with declarative event handling

Who is using Backbone ?

- Sony Entertainment Network
- SnagFilms
- CakeMail mobile
- Trello
- Gojee
- ZenPayroll
- DropTask
- Stitcher
- Groupon
- 37 Signals - Basecamp
- AirBnb
- Pandora
- USA Today
- Hulu
- Quartz
- Gawker Media
- Wordpress
- Foursquare
- Bitbucket
- Disqus
- Pitchfork
- Nokia - Profiles
- Corbis Crave
- Walmart Mobile

Backbone Concepts

- Backbone does not have controllers, it's aligned to the MVP pattern
- Views contain controller logic
- Supports event-driven communication between views and models
- Data binding is done through manual events or a separate key-value observing library
- Underscore.js templating is used by default

Backbone Structure

- Models
  - define data, validation and business rules
  - if you want to receive a notification when a model changes you can bind a listener to the model for its change event. Individual attribute change listeners are also possible
- Views
  - In Backbone, views are not the markup but instead the presentation behavior. They are defined as JS objects that are associated to a model and a template
- Collections
  - They are sets of models

What is Ember.js ?

- A JS framework based on the MVC pattern
- Sits in the middle between Backbone and Angular
- More opinionated than Backbone, provides you with automatic data binding

Who is using ember ?

- Zendesk
- Thoughtbot
- Square
- Yahoo
- Groupon
- Boxee
- Code school
- Yapp
- Discourse
- Twitch
- Basho
- Addepar

Ember Concepts

- Uses handlebars.js templates, based on mustache but with support for logic operators inside the templates
- REST-oriented. You define router objects that are associated to urls that poll data from models.
- Relies heavily on naming conventions to perform automatic binding between: router, model and template

JS MVC Framework Comparison

Fw	Pros	Cons
Angular	Loaded with features Easy to get started Fast development / Less boilerplate Testing easy Strong community	Takes longer to learn Has not been proven in as many projects as Backbone
Backbone	Relatively mature, proven framework Strong community Many extensions Flexible, works fine for new projects or for existing ones	Lacks data-binding Require large amounts of boilerplate Dangerous if developers do not follow conventions
Ember	Can do almost all that Angular can Claims to be faster Does not deviate from standard JS Newer framework	Less known Less mature Less libraries

DIRT applications

- Stands for Data Intensive Real Time applications
- With its V8 engine, Chrome has initiated a browser war with JS performance a primary battlefield
- The browser has become sufficiently powerful to allow for interesting real time applications
- Rise of the asynchronous and evented server-side architectures

which leads us to…

What is Node.js ?

- Platform built on top of Google Chrome's V8 JS runtime for easily building fast, scalable network applications
- Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient
- Single threaded, event-loop based architecture
  - Perfect for IO bound data-intensive real-time applications that run across distributed devices
  - Not well suited for CPU bound applications

Who is using node ?

LinkedIn

- “On the server side, our entire mobile software stack is completely built in Node. One reason was scale. The second is Node showed us huge performance gains”

YAHOO!

The New York Times

- “A news organization must be responsive, both to its readers and to a fast-paced flow of information. Node provides a level of flexibility we haven’t found anywhere else - and enables us to deliver performant apps that can be easily adjusted”

PayPal™

- “Node.js powers our web applications and has allowed our teams to move much faster in bringing their designs to life. We’ve happily embraced the power of Javascript”

ebay®

- “Node’s evented I/O model freed us from worrying about locking and concurrency issues that are common with multithreaded async I/O”

Node history

- 2008
  - Chrome is released, JS performance improved at an incredible rate due to heavy competition
    - V8 is essentially a javacript engine written in C++ that gives a huge boost in performance because it cuts out the middleman, preferring straight compilation into native machine code over executing bytecode using an interpreter
- 2009
  - Node.js is released

Node concepts

- Scalability
  - Node never sleeps because it does asynchronous IO via callbacks, so a slow IO operation will never block others
- Efficiency
  - Thread-based networking is relatively inefficient and difficult to use. Node will show better memory efficiency under high loads than systems which allocate 2mb stacks for each connection

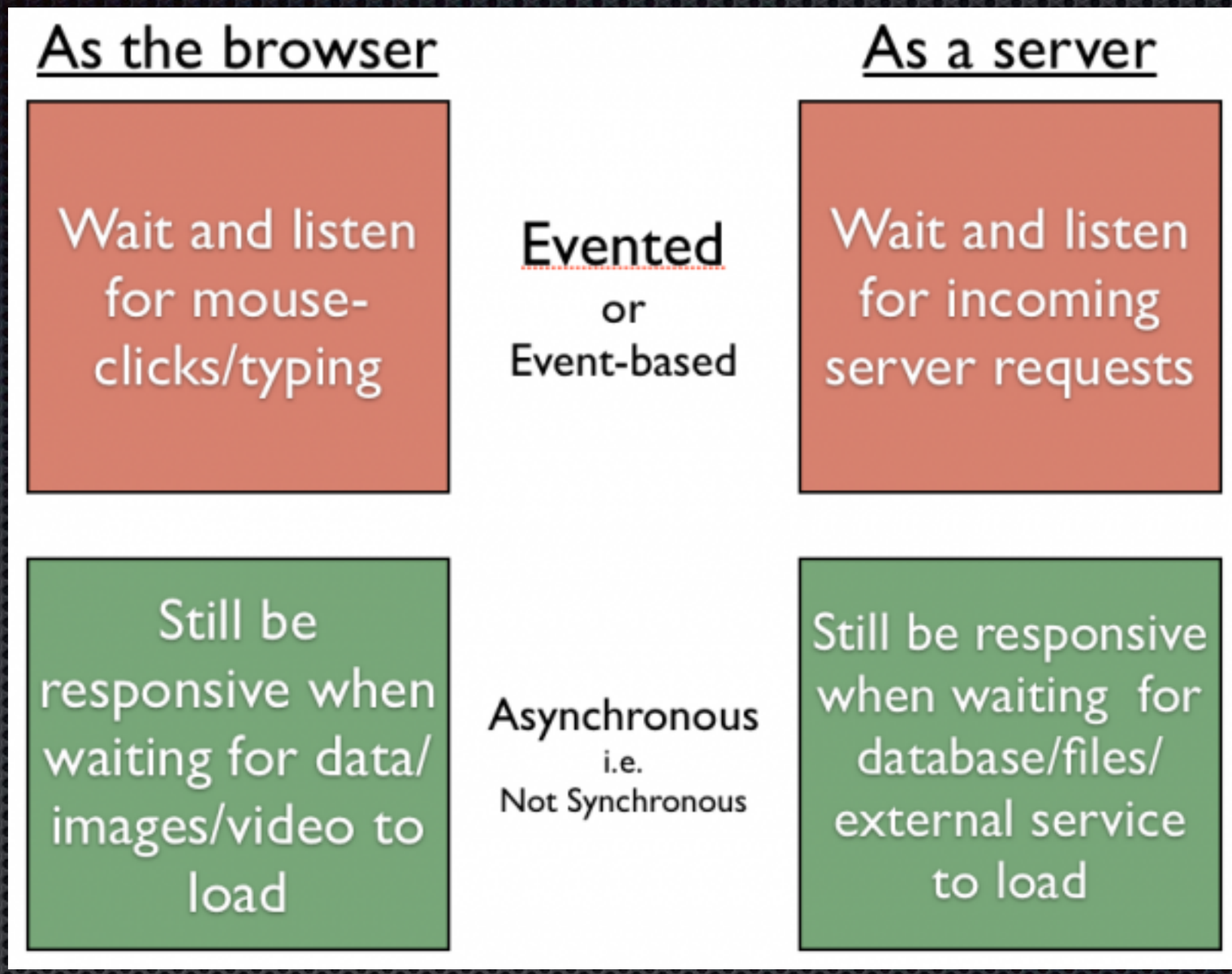
Node apps

- Applications
  - Perfect for building servers
  - Provides an easy way to build scalable network apps
  - Easier to develop concurrency and parallelism
    - No mutexes, semaphores, locks… all via callbacks
- Modules
  - Node's standard library can be extended with modules
  - Huge list in npmjs.org

Node event loop

- Node presents the event loop as a language construct instead of as a library
- Node simply enters the event loop after executing the input script and exits when there are no more callbacks to perform

Node concepts



- Turns out the same properties that make JS good for interacting with a web page are also what is needed for a good web server: Asynchronous and event-based

Mean stack

- Full stack JS framework which provides a good starting point for: MongoDb, Node.js, Express and Angular.js based applications
- When set up, you get a fully functional blog engine application that you can tweak to your needs

Mongo.db

- NoSql, Document database that provides high performance, high availability and easy scalability
- For integrating with a node app, I recommend the mongoose library that is an ODM (Object Document Mapper)

Express.js

- This is a web application framework for node apps, with builtin support for:
  - Jade templates
  - Session management
  - Authentication
  - Routing
  - Caching
- In Node, your application is in fact the web-server, for web sites better to base it on a proven one like express than to build your own from scratch

Javascript Fears

- How to protect Intellectual Property
  - Obfuscation is used to convert code into equivalent one that is difficult to reverse engineer, it's not impossible to de-obfuscate but the code will be unreadable
- How to keep chaos away
  - Avoid using unnecessary libraries
  - Adhere to unobstrusive javascript
  - DRY code
  - Explore the functional nature of JS
  - Use an object-oriented approach
  - Refactor often


That's all Folks!