

# **Mobile Web Applications Development with HTML5**



## **Lecture 3: Templating and MVC**

**Claudio Riva  
Aalto University March 2012**

# **LECTURE 3: TEMPLATING AND MVC**

**A QUICK RECAP**

**CLIENT-SIDE TEMPLATING**

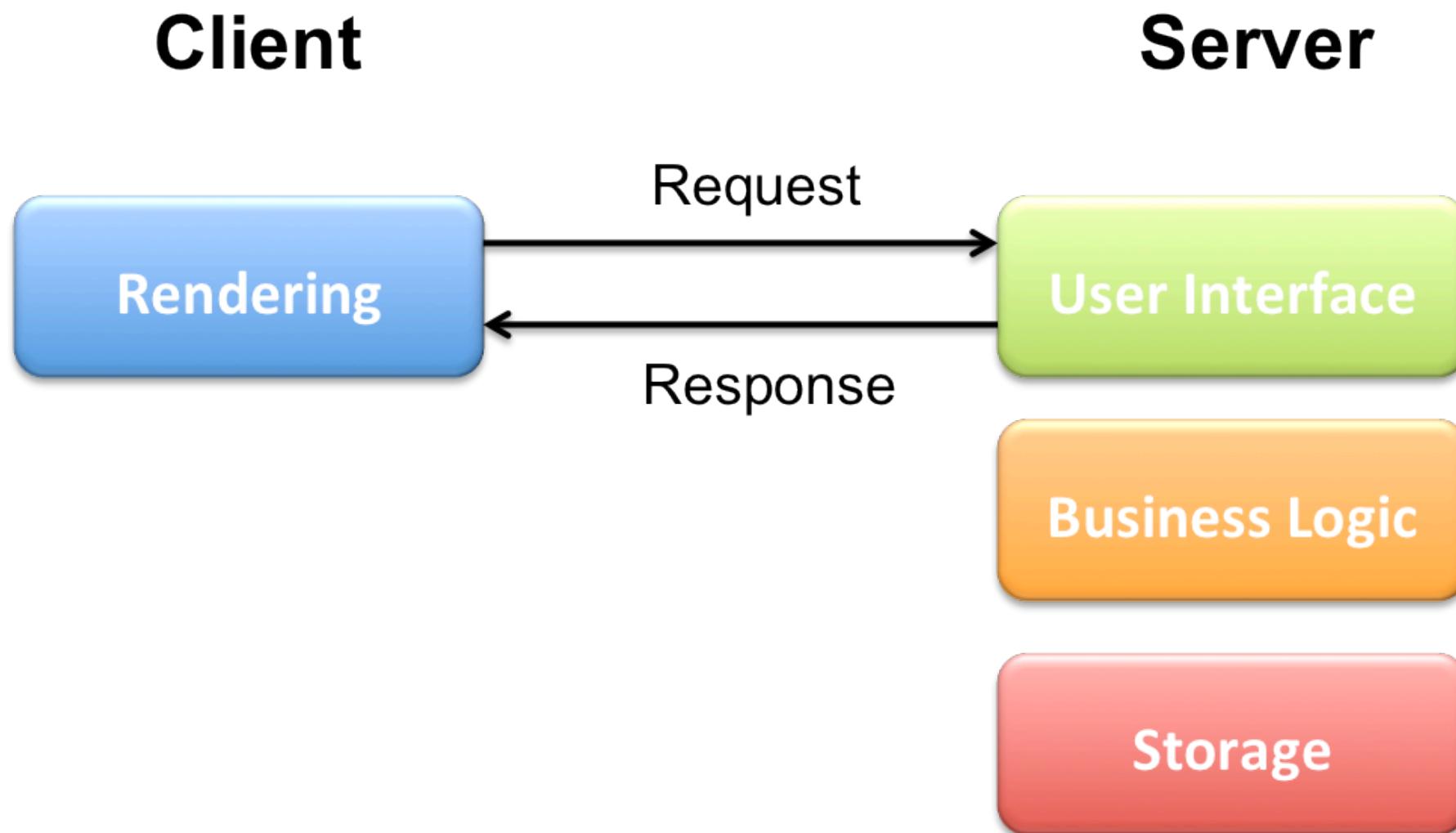
**HANDLERBAR.JS**

**MODEL-VIEW-CONTROLLER**

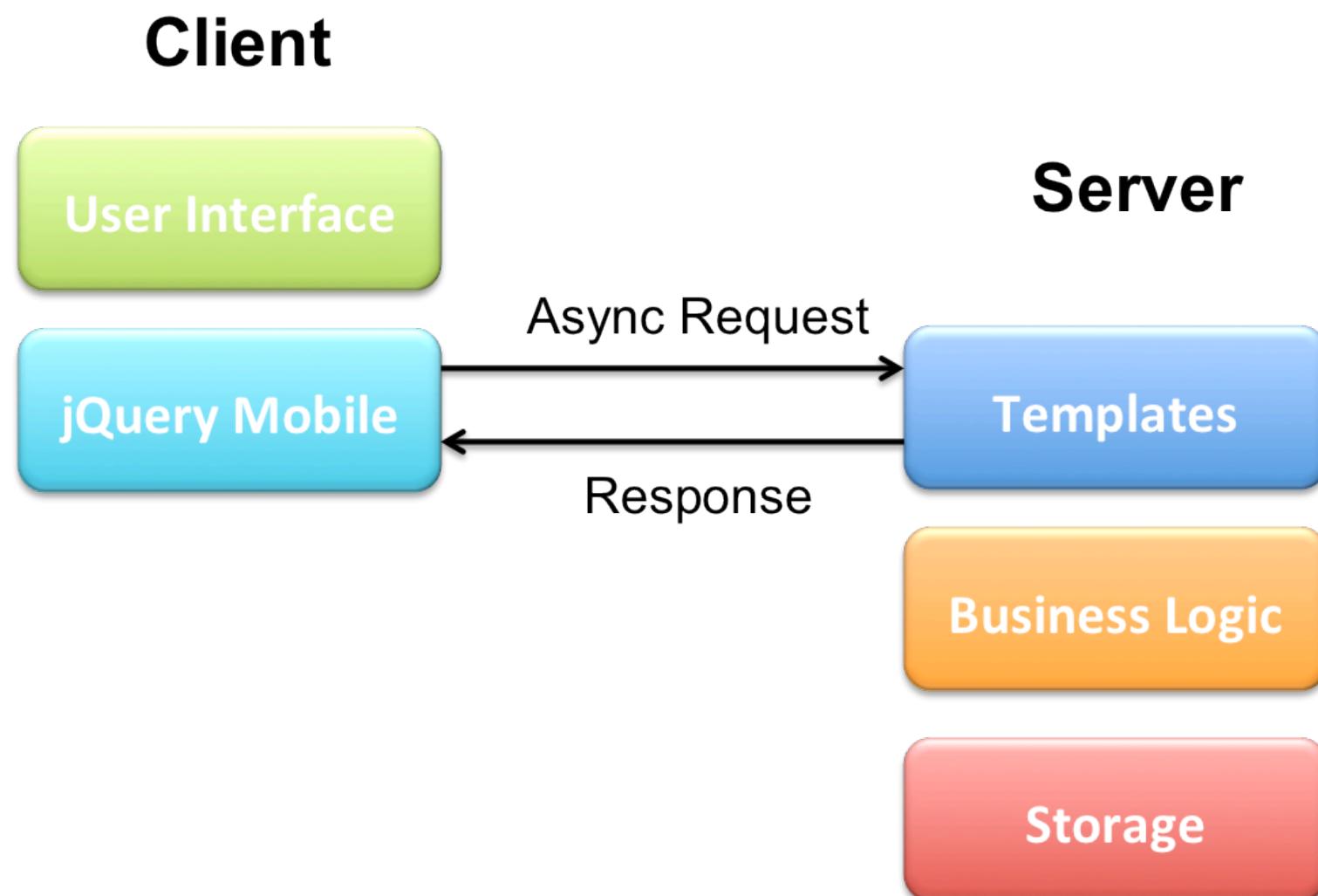
**BACKBONE.JS**

**EXAMPLES**

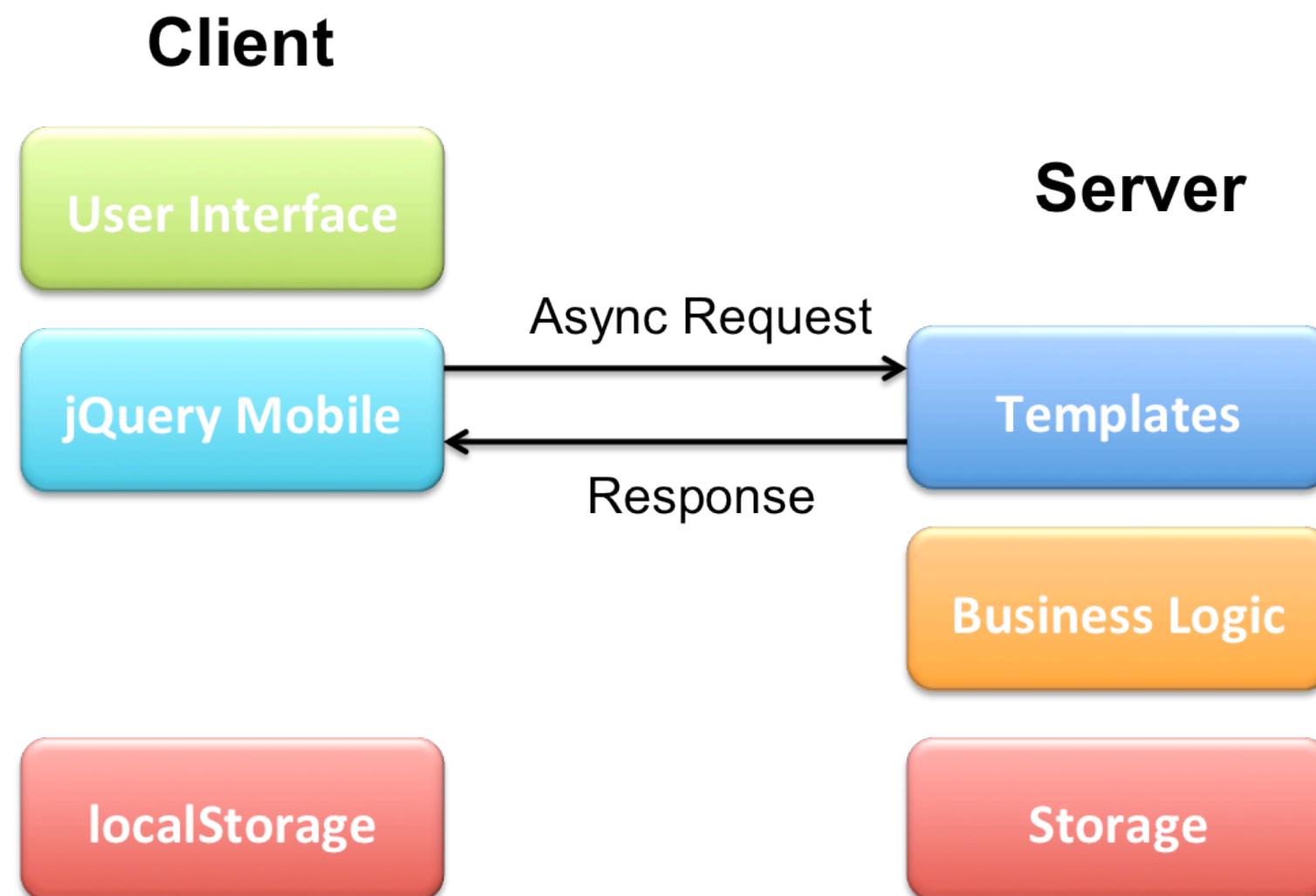
We started here



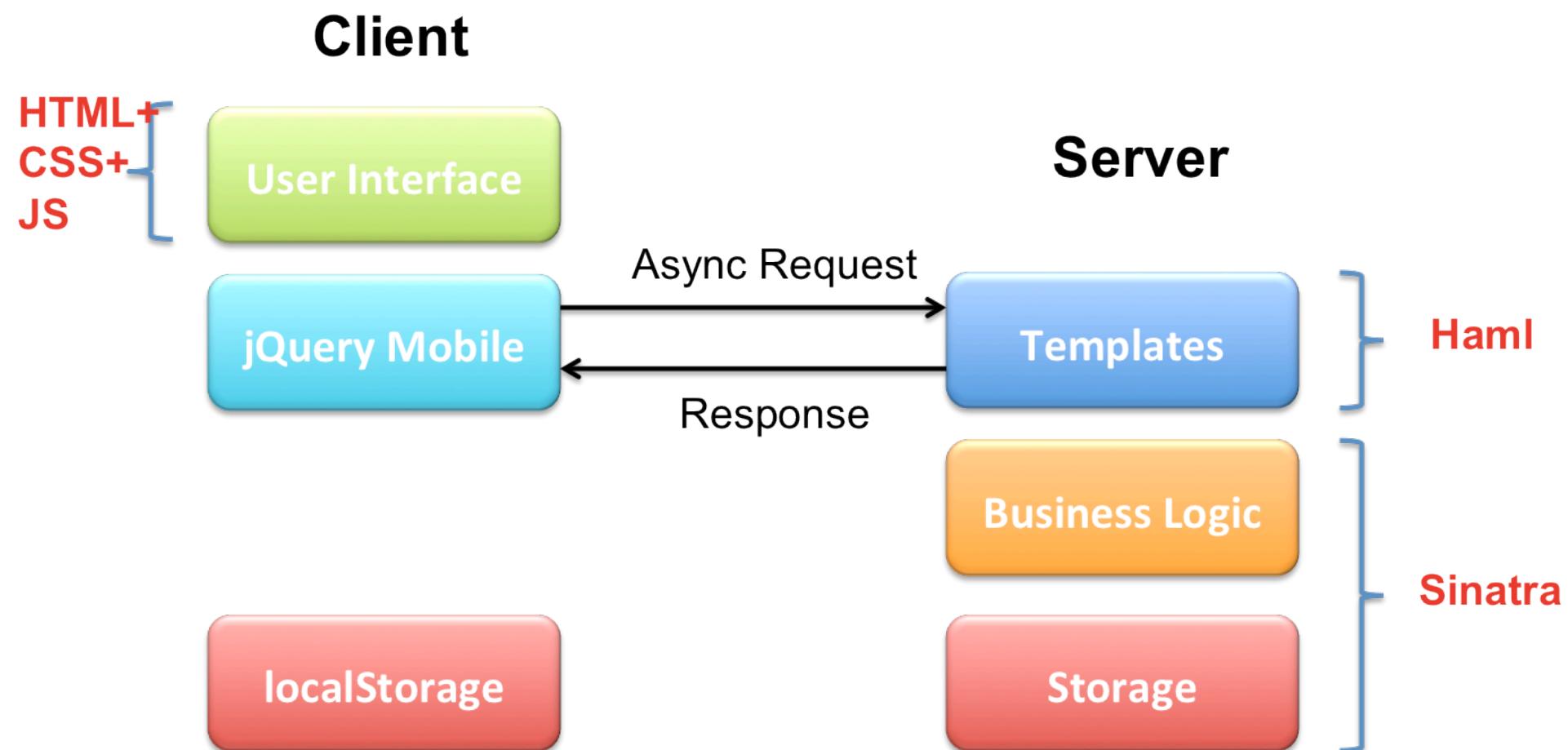
# We added jQuery Mobile



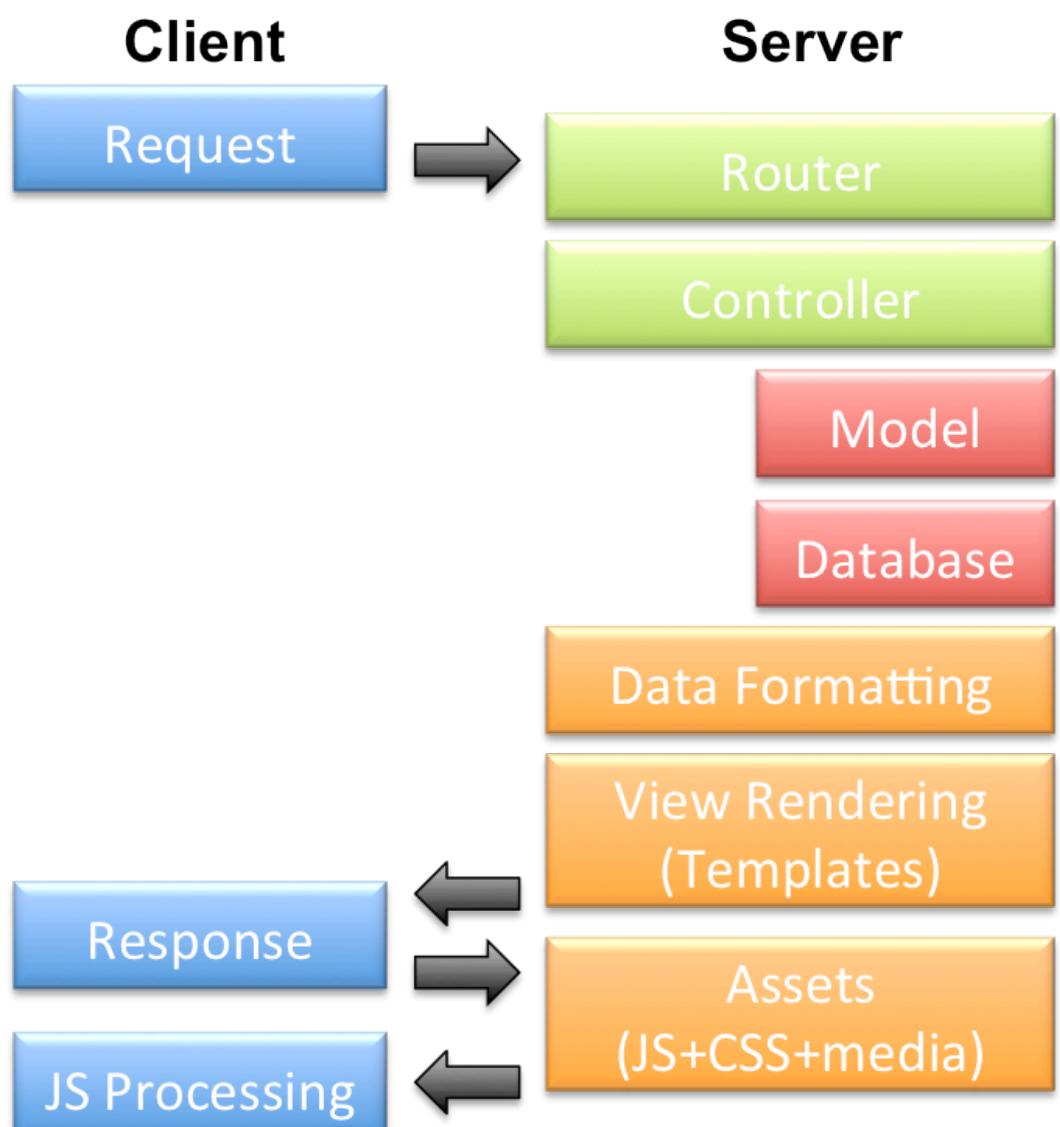
# We added local storage



# Server-side templates with Haml



# Traditional Sequence in a Web Request



## Model

```
$articles = [{ :title => "Welcome", :content => "My first post",  
  :email => "hello@blog.com", :timestamp => "1.1.2012 10:20:30"}]
```

## Router + Controller

```
get '/' do  
  haml :index, :layout => :layout  
end
```

## Template

```
%div(data-role="content")  
  %ul(data-role = "listview")  
    - $articles.each_with_index do |a, i|  
      %li  
        %p.ui-li-aside  
          =a[:timestamp]  
        %h3(contenteditable="true" data-name="title" data-id="#{i}")  
          =a[:title]  
        %p  
          %strong  
            =a[:email]  
          %p(contenteditable="true" data-name="content" data-id="#{i}")  
            =a[:content]
```

# Traditional Response

```
<div data-role='content'>
  <ul data-role='listview'>
    <li>
      <p class='ui-li-aside'>1.1.2012 10:20:30</p>
      <h3 contenteditable='true' data-id='0' data-name='title'>Welcome</h3>
      <p>
        <strong>hello@blog.com</strong>
      </p>
      <p contenteditable='true' data-id='0' data-name='content'>My first post</p>
    </li>
    <li>
      <p class='ui-li-aside'>04.02.2012 12:28:31</p>
      <h3 contenteditable='true' data-id='1' data-name='title'>Test</h3>
      <p>
        <strong>hello@blog.com</strong>
      </p>
      <p contenteditable='true' data-id='1' data-name='content'>This is test content</p>
    </li>
  </ul>
</div>
```

# Important Data in Response

```
<div data-role='content'>
  <ul data-role='listview'>
    <li>
      <p class='ui-li-aside'>1.1.2012 10:20:30</p>
      <h3 contenteditable='true' data-id='0' data-name='title'>Welcome</h3>
      <p>
        <strong>hello@blog.com</strong>
      </p>
      <p contenteditable='true' data-id='0' data-name='content'>My first post</p>
    </li>
    <li>
      <p class='ui-li-aside'>04.02.2012 12:28:31</p>
      <h3 contenteditable='true' data-id='1' data-name='title'>Test</h3>
      <p>
        <strong>hello@blog.com</strong>
      </p>
      <p contenteditable='true' data-id='1' data-name='content'>This is test content</p>
    </li>
  </ul>
</div>
```

# Client-side templating

```
%ul(data-role = "listview")
{{#articles}}
%li
  %p.ui-li-aside
    {{timestamp}}
  %h3(contenteditable="true" data-name="title" data-id='{{id}}')
    {{title}}
  %p
    %strong
      {{email}}
  %p(contenteditable="true" data-name="content" data-id='{{id}}')
    {{content}}
{{/articles}}
```

```
{
  articles: [
    title: "Welcome",
    content: "My first post",
    email: "hello@blog.com",
    timestamp: "1.1.2012 10:20:30",
    id: 0
  ]
}
```

# Handlebars.js

Handlebars.js provides the tools for building **semantic templates** that can be compiled and instantiated on the client-side

## Template

```
<script id="#entry" type="text/x-handlebars-template">
  <div>
    <h1>{{title}}</h1>
    <div class="content">
      {{content}}
    </div>
  </div>
</script>
```

## Instance

```
<div>
  <h1>My New Post</h1>
  <div class="content">
    This is my first post!
  </div>
</div>
```

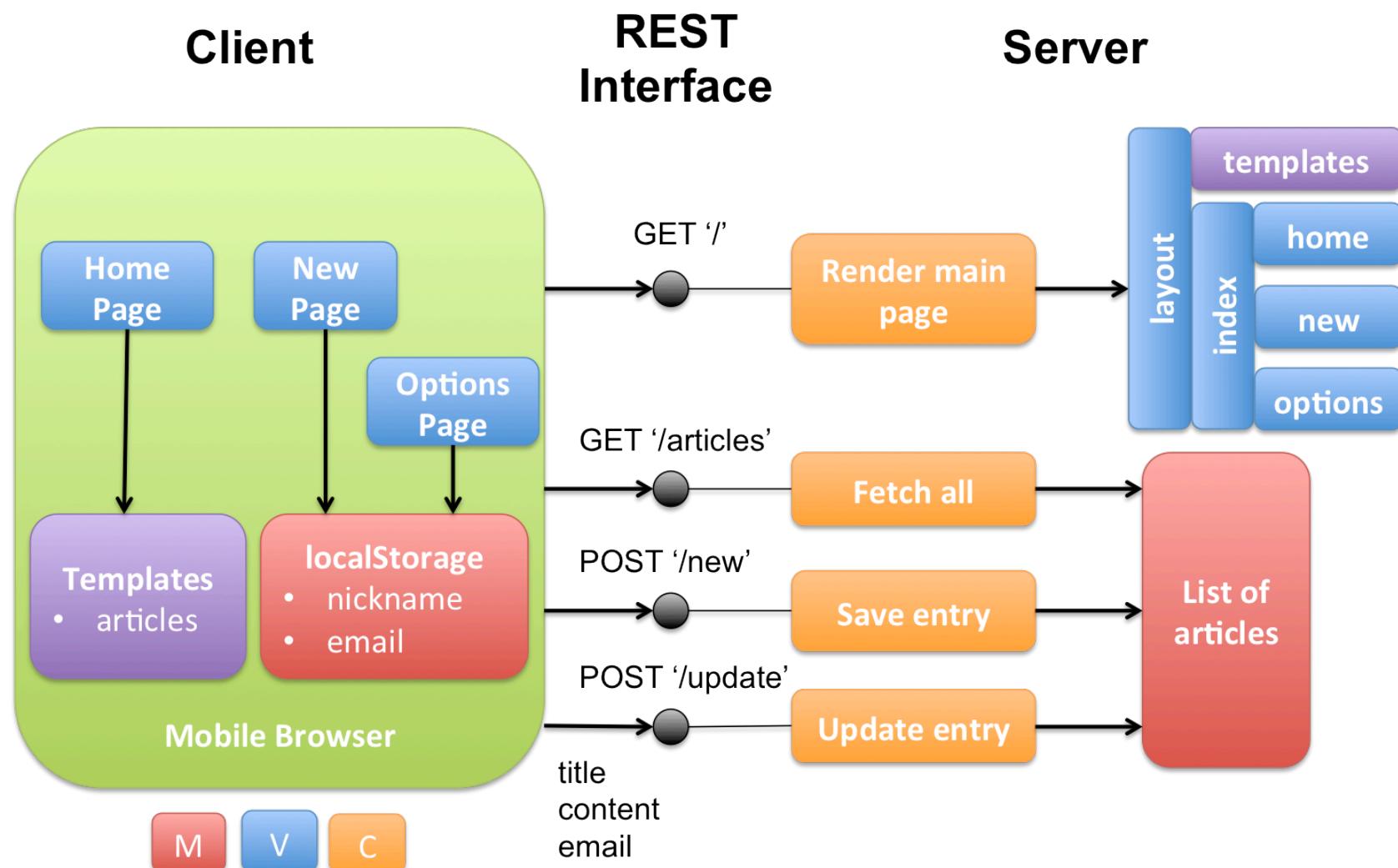
```
var source = $("#entry").html();
var template = Handlebars.compile(source);
var context = {title: "My New Post", body: "This is my first post!"}
var html = template(context);
```

# Blog with Templates templateHandlebarsBlog

views/templates.haml

```
%script(id="articles" type="text/x-handlebars-template")
  %ul(data-role = "listview")
    {{#articles}}
      %li
        %p.ui-li-aside
          {{timestamp}}
        %h3(contenteditable="true" data-name="title" data-id='{{id}}')
          {{title}}
        %p
          %strong
            {{email}}
        %p(contenteditable="true" data-name="content" data-id='{{id}}')
          {{content}}
    {{/articles}}
```

# Design with Templates



# Blog with Templates

templateHandlebarsBlog

## views/layout.haml

```
%body  
  = haml :templates  
  = yield
```

## views/index.haml

```
%div(data-role="page" id="home")  
  = haml :header  
  
  %div(data-role="content" id="articlesList")  
  
  = haml :footer  
  
%div(data-role="page" id="new")  
  = haml :header  
  
  %div(data-role="content")  
    = haml :new  
  
  =haml :footer  
  
%div(data-role="page" id="options")  
  =haml :options  
  =haml :footer
```

# Blog with Templates templateHandlebarsBlog

## server.rb

```
get '/articles' do
  content_type :json
  {articles => $articles}.to_json
end
```

## views/layout.haml

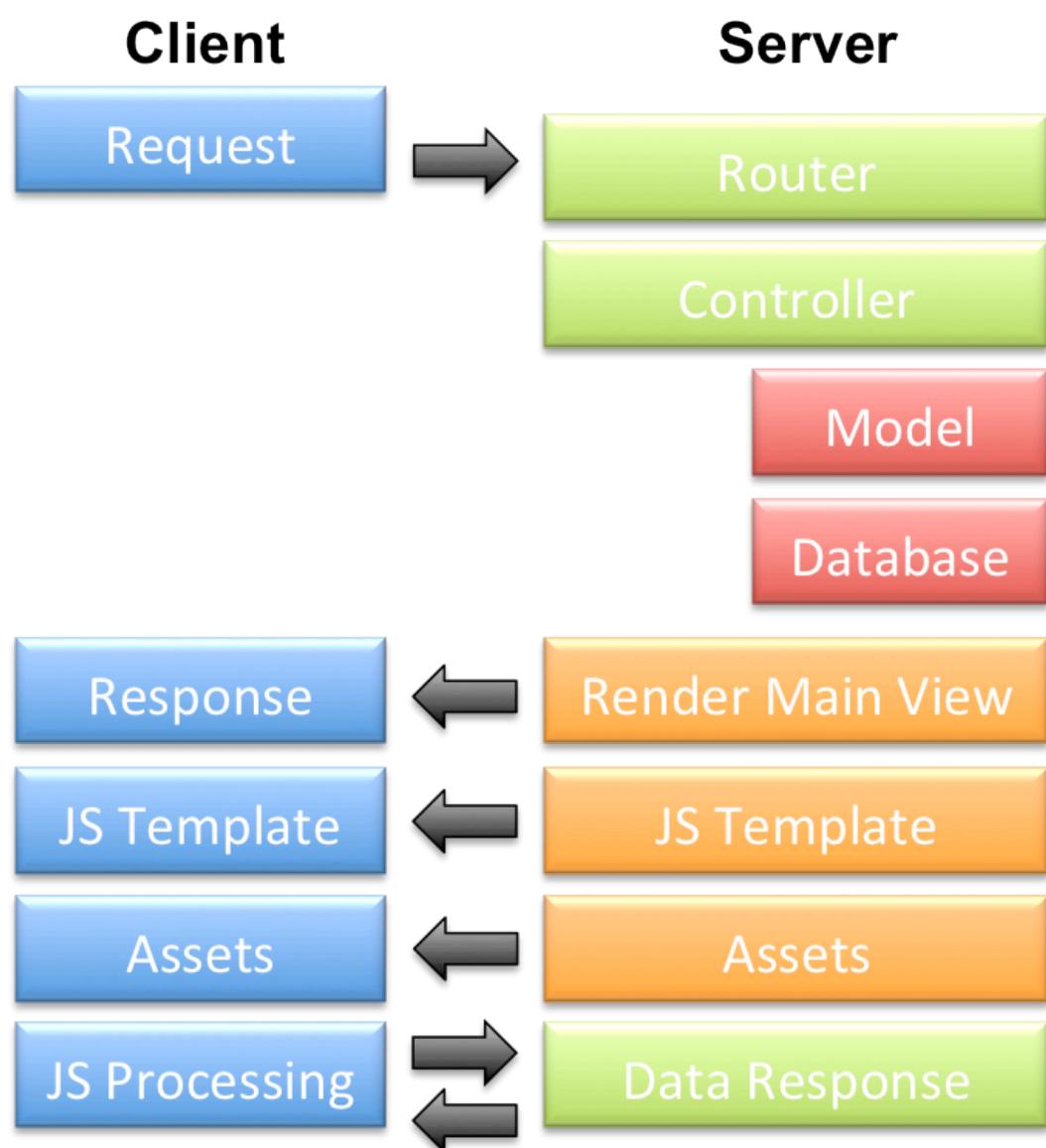
```
var Templates = {};

function loadArticles () {
  $.getJSON('/articles', function(json) {
    var content = Templates.articles(json);
    $("#articlesList").html(content).find("ul").listview();
  });
}

$(function() {
  $('script[type="text/x-handlebars-template"]').each(function () {
    Templates[this.id] = Handlebars.compile($(this).html());
  });

  loadArticles();
  ...
});
```

# Achievements

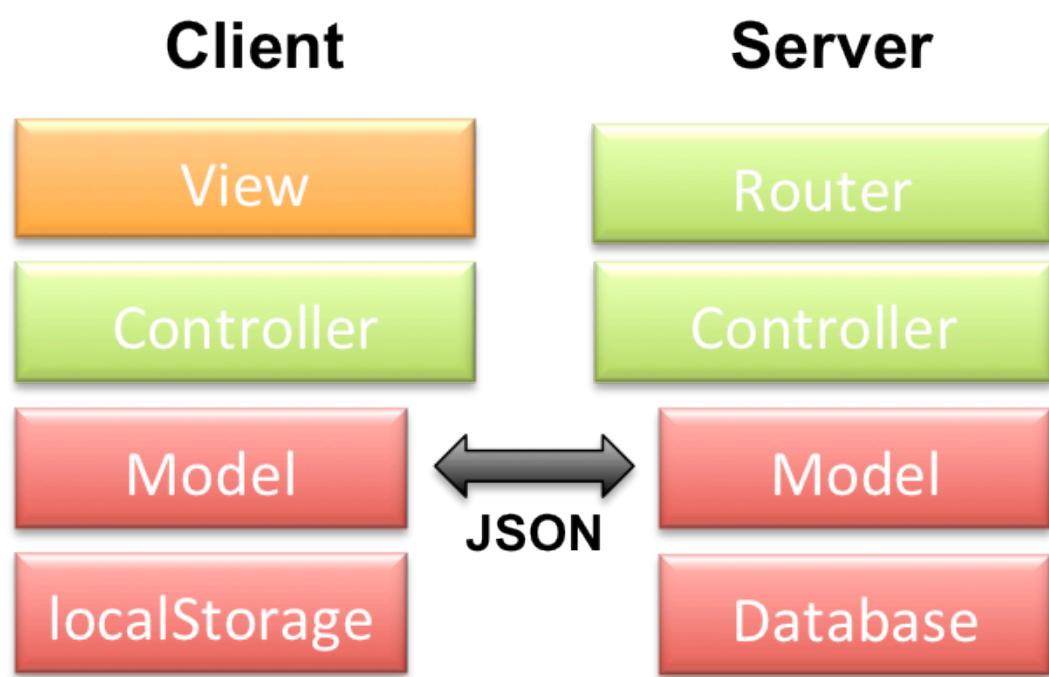


- Controller provides a clear **interface**
- Server-side view prepares **formatted output** (as a **decorator**)
- Client-side receives a decorated **JSON** and a **template** in HTML/JS
- Client **instantiates** the template with the JSON data and **renders** it
- Less work on server-side means **more requests per second**
- Views become API customers, **unifying the data interface**

# Templating Tools

- **Mustache** : Generic logic-less templates for HTML, config files, code, anything.
- **Mustache.js** : Mustache implementation in Javascript
- **Haml-js** : Haml implemenation in Javascript (popular with node.js)
- **Eco** : Embed CoffeScript logic in markup (popular with node.js)
- **ICanHaz.js** : JS helper for client-side templating.

# Towards a Client-Side MVC Architecture



Various MVC frameworks:

- [Backbone.js](#)
- [Sproutcore](#)
- [Cappuccino](#)
- [KnockoutJS](#)
- [JavaScriptMVC](#)
- [Sammy.js](#)
- [Ember.js](#)

# Backbone.js

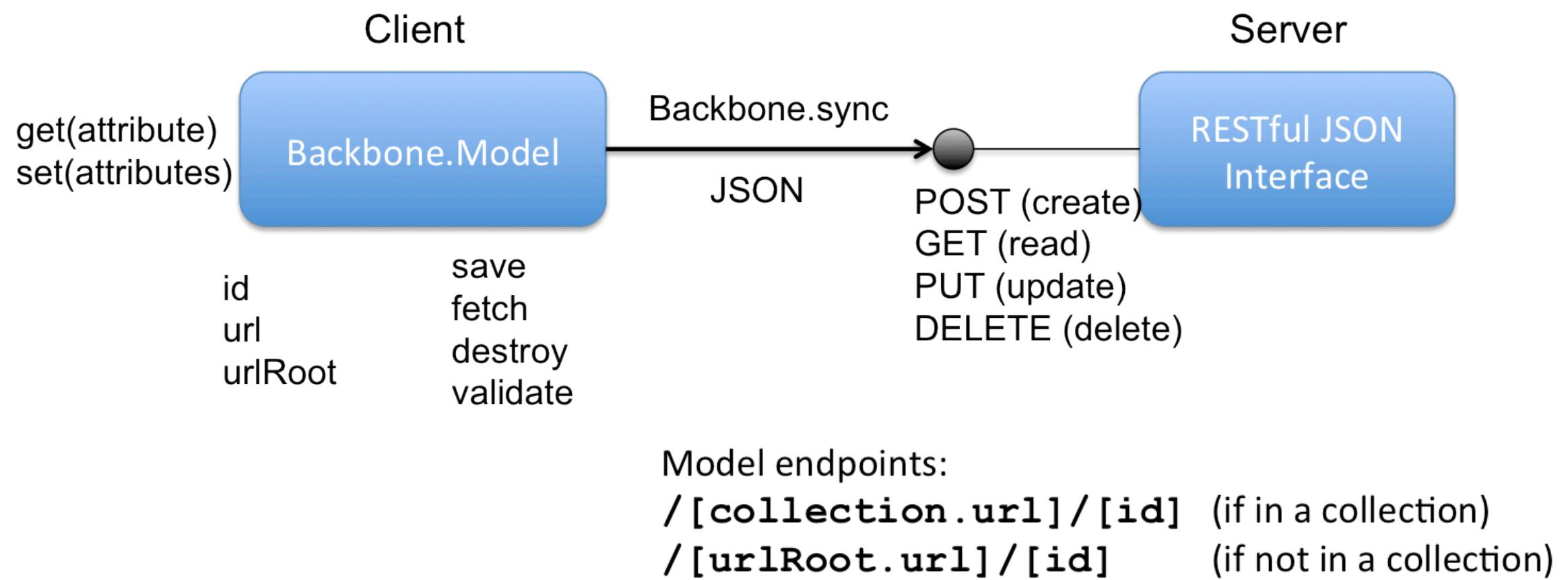
Backbone.js supplies structures to web applications by providing:

- **models** with key-value binding and custom events
- **collections** with a rich API for enumerable functions
- **views** with declarative handling
- **routers** for connecting client-side pages to actions and events
- **RESTful JSON interface** for connecting to existing applications

## Backbone.js - Benefits

- No assumptions about the UI
- Flexible about the data persistence possibilities
- Support any HTML templating engine
- Only hard dependency is with Underscore.js
- Lightweight: no UI widgets
- Backend agnostic by using RESTful JSON endpoints

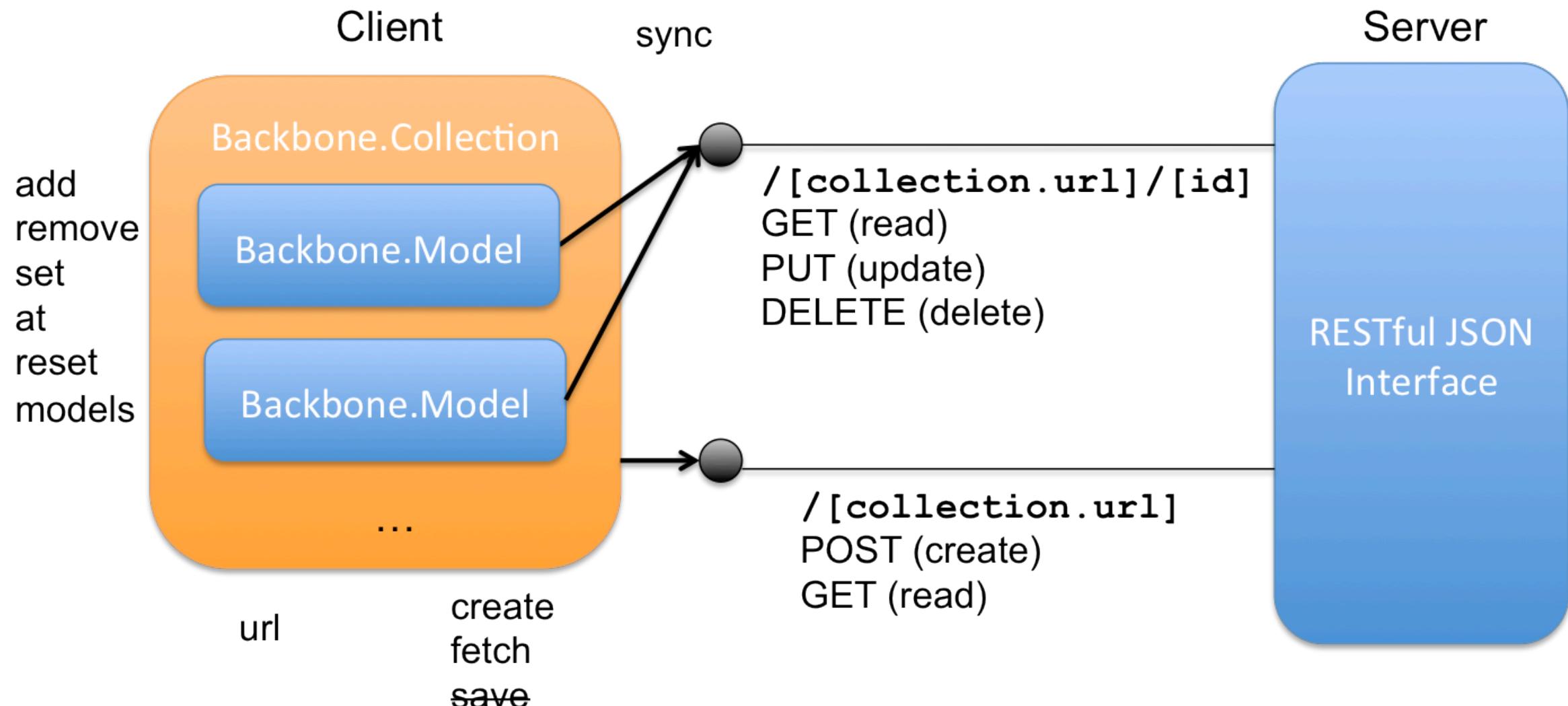
# Backbone.js - Models



## Let's try it in the Javascript console

```
var Book = Backbone.Model.extend({});  
var book = new Book;  
//try book, book.id  
book.set('title', 'Hello');  
book.set({author: "someone"});  
book.get('title'); //book.attributes  
bookisNew();  
book.save();  
book.url();  
book.urlRoot;  
book.urlRoot = "/books";  
book.save();  
book.fetch();
```

# Backbone.js - Collections



# Let's try it in the Javascript console

```
var Book = Backbone.Model.extend({});  
var Library = Backbone.Collection.extend({model: Book});  
var library = new Library;  
var book = library.create({title: "Hello", author: "myself"});  
library.url = "/books";  
var book = library.create({title: "Hello", author: "myself"});  
var book = library.create({title: "Stories", author: "myself"}, {wait: true});  
library.models;  
JSON.stringify(library.toJSON);  
library.fetch();  
library.reset();  
//Add and Remove don't trigger Backbone.sync  
book = new Book({title: "HTML5", author: "myself"});  
library.add(book);  
//Saving the model will trigger the Backbone.sync  
book.save();  
//The model is only removed from the collection but not from the server  
library.remove(book);  
//To remove from the server user either:  
library.remove(book).destroy();  
//or:  
book.destroy(); //deletes from all collections as well
```

# Backbone Model for Blog

mvcBackboneBlog

## Read collection

```
get '/articles' do
  content_type :json
  $articles.to_json
end
```

## Read item

```
get '/articles/:id' do
  content_type :json
  $articles[params[:id].to_i].to_json
end
```

## Create item

```
post '/articles' do
  data = JSON.parse(request.body.string)
  article = {}
  [:title, :content, :email, :name].each do |k|
    article[k] = data[k.to_s] || ""
  end
  article[:timestamp] = timestamp
  article[:id] = $articles.length
  $articles.unshift(article)
  article.to_json
end
```

## Update item

```
put '/articles/:id' do
  data = JSON.parse(request.body.string)
  article = {}
  [:title, :content, :email, :name].each do |k|
    article[k] = data[k.to_s] || ""
  end
  article[:timestamp] = timestamp
  article[:id] = params[:id].to_i
  $articles[article[:id].to_i].merge!(article)
  content_type :json
  $articles[params[:id].to_i].to_json
end
```

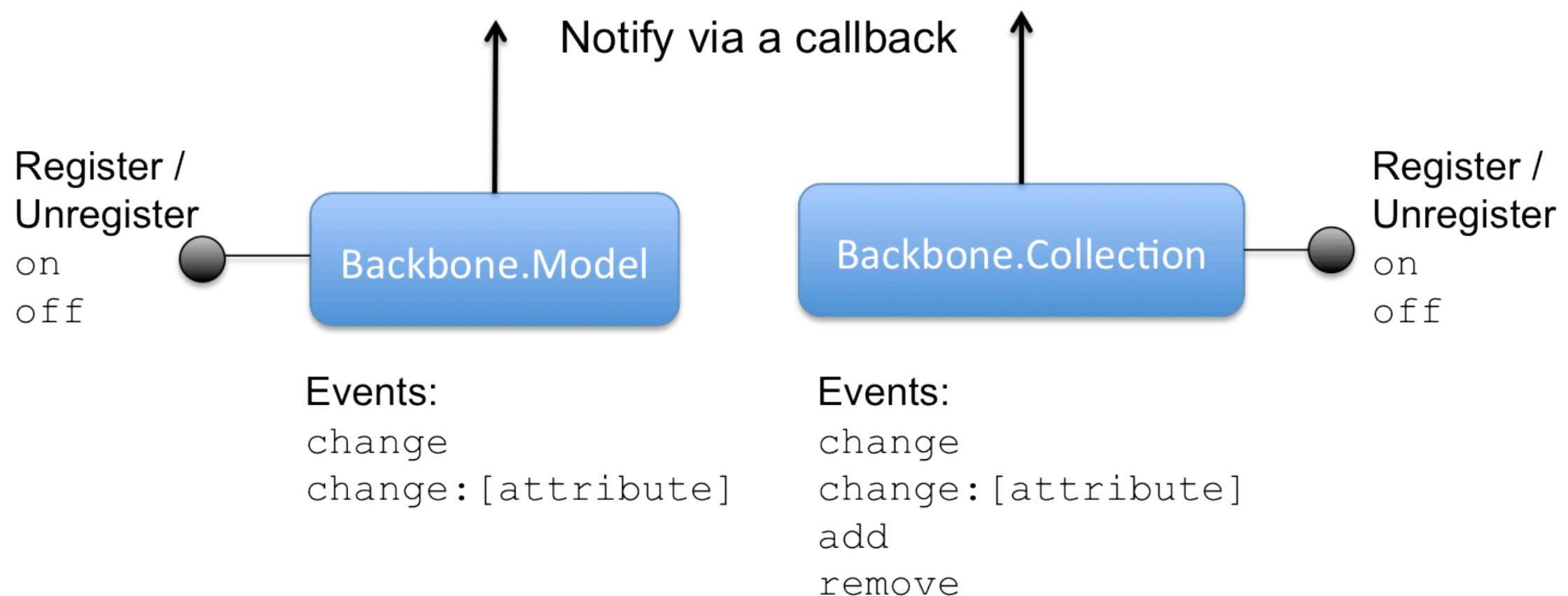
## Destroy item

```
delete '/articles/:id' do
  $articles.delete_at(params[:id].to_i)
end
```

## Let's test the model in the Javascript console

```
articles.fetch();
articles.length;
articles.add([{title: "First Post"}, {title: "Second post"}]);
articles.reset();
articles.create({title: "First Post", content: "The content goes here"});
articles.reset();
articles.fetch();
articles.at(2).set({title: "Something", content: "different"}).save();
articles.at(1).remove();
```

## Backbone.js - Events



# Let's test the events in the Javascript console

```
articles.fetch();
m=articles.at(0);
//Event for all changes of the model
m.on('change', function(model) { alert(JSON.stringify(model.changedAttributes())); } );
m.set('title', 'a new value');
//Event only for changes of the content
m.on('change:content', function(model) { alert(JSON.stringify(model.changedAttributes())); } );
//Events can be set on the collection as well.
articles.on('change', function(model) {alert(JSON.stringify(model.changedAttributes()))});
```

# Backbone.js - View

A view handles two duties fundamentally:

- Listen to events thrown by the DOM and models/collections.
- Represent the application's state and data model to the user.

```
var View = Backbone.View.extend ({  
  tagName: "div",           //tag of the generated element  
  className: "item",         //class of the generated element  
  id: "home",                //id of the generated element  
  el: $('#homeContainer'),  //reference to the DOM element  
  
  //function for rendering the view  
  render: function() { this.el.innerHTML = ... ; return this; }  
});  
  
var v = new View(); //Instantiate a view  
v.el; // DOM element of the view  
v.$el; //Cached jQuery object for the view's element  
v.render(); //Renders the view into the element  
v.render().el; //DOM element of the rendered view
```

# Inserting a view in the DOM

```
var itemView = Backbone.View.extend ({
  tagName: "li",
  className: "item",
  render: function() { this.el.innerHTML = 'Hello'; return this; }
});

var i = new itemView();
i.el;
i.render().el;
$("#articlesList").html(i.render().el);
```

# Attaching the View to an Existing DOM Element

```
var itemView1 = Backbone.View.extend ({
  el: $("#articlesList"),
  render: function() { $(this.el).html('Hello'); return this; }
});

var i1 = new itemView1();
i1.render();

var itemView2 = Backbone.View.extend ({
  el: $("#articlesList"),
  render: function() { $(this.el).html('Hello World'); return this; }
});

var i2 = new itemView2();
i2.render();
i2.$el.html('Cheers');
i1.render();
i2.render();
```

# Attaching a Model to a View

```
var itemView = Backbone.View.extend ({
  el: $("#articlesList"),
  render: function() { $(this.el).html(this.model.get('title') + "," + this.model.get('content'));
  return this; }
});

var m = new Article({title: "Hello World", content: "not much to say"});
var v = new itemView({model: m});

v.render();
```

# Binding the View to the Model

```
var itemView = Backbone.View.extend ({
  el: $("#articlesList"),
  initialize: function() { this.model.on('change', this.render, this); },
  render: function() { $(this.el).html(this.model.get('title') + "," + this.model.get('content'));
  return this; }
});

var m = new Article({title: "Hello World", content: "not much to say"});
var v = new itemView({model: m});

v.render();
m.set('title','Good morning');
```

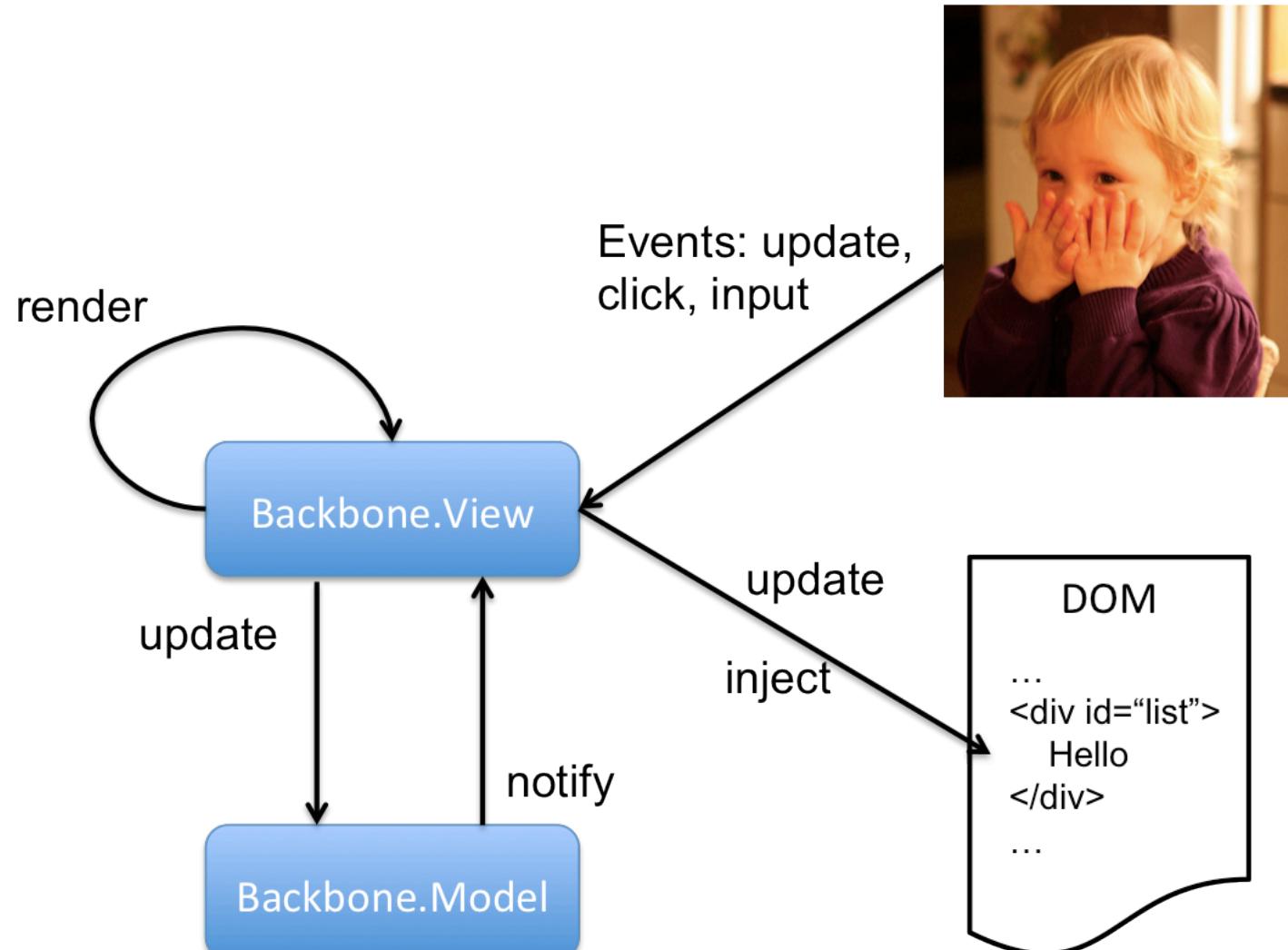
# Binding to the View's Events

```
var itemView = Backbone.View.extend ({
  el: $("#articlesList"),
  events: {'click': 'handleClick'},
  initialize: function() { this.model.on('change', this.render, this); },
  render: function() { $(this.el).html(this.model.get('title') + "," + this.model.get('content'));
  return this; },
  handleClick: function() { this.$el.fadeOut().fadeIn(); }
});

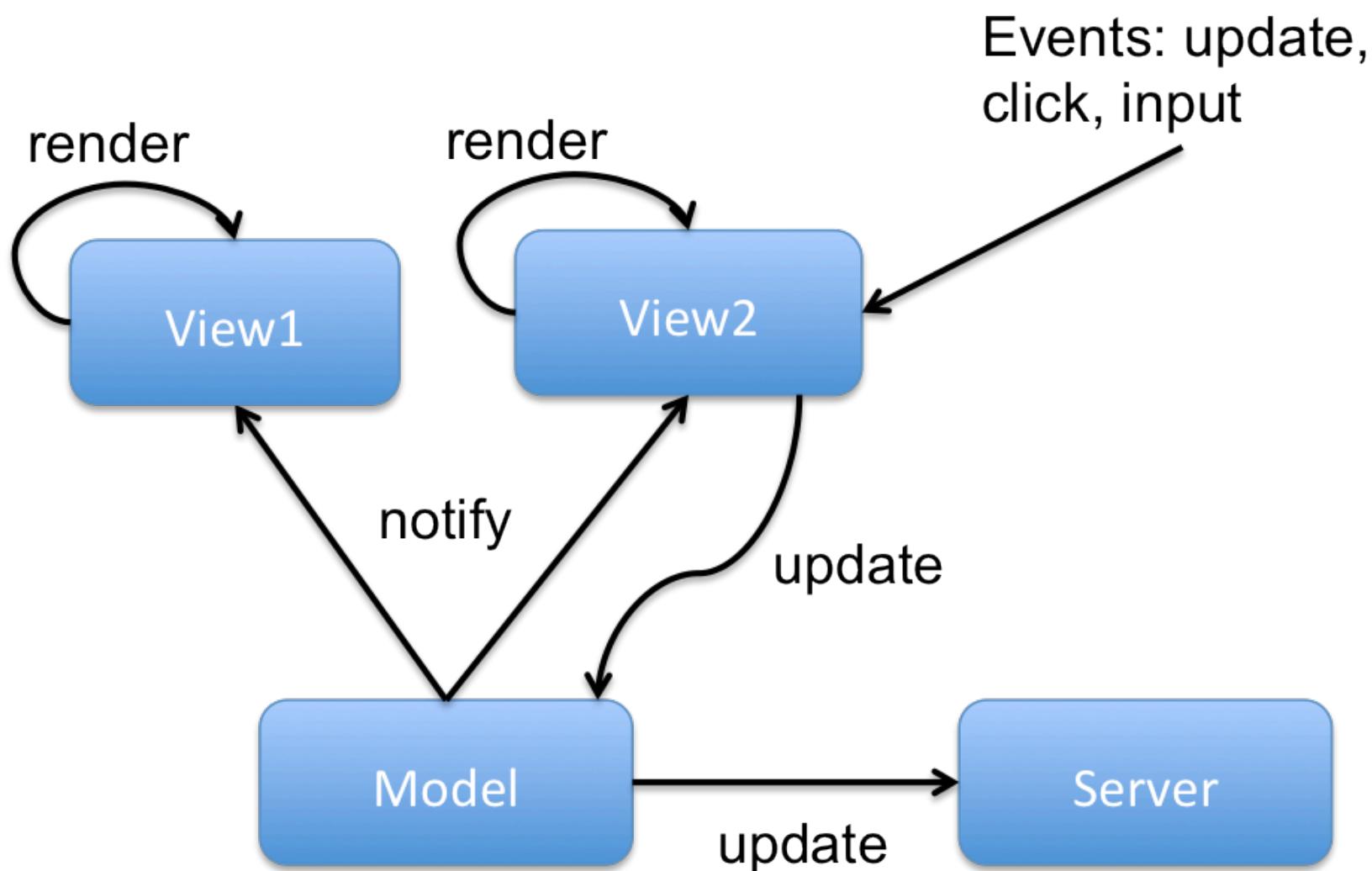
var m = new Article({title: "Hello World", content: "not much to say"});
var v = new itemView({model: m});

v.render();
```

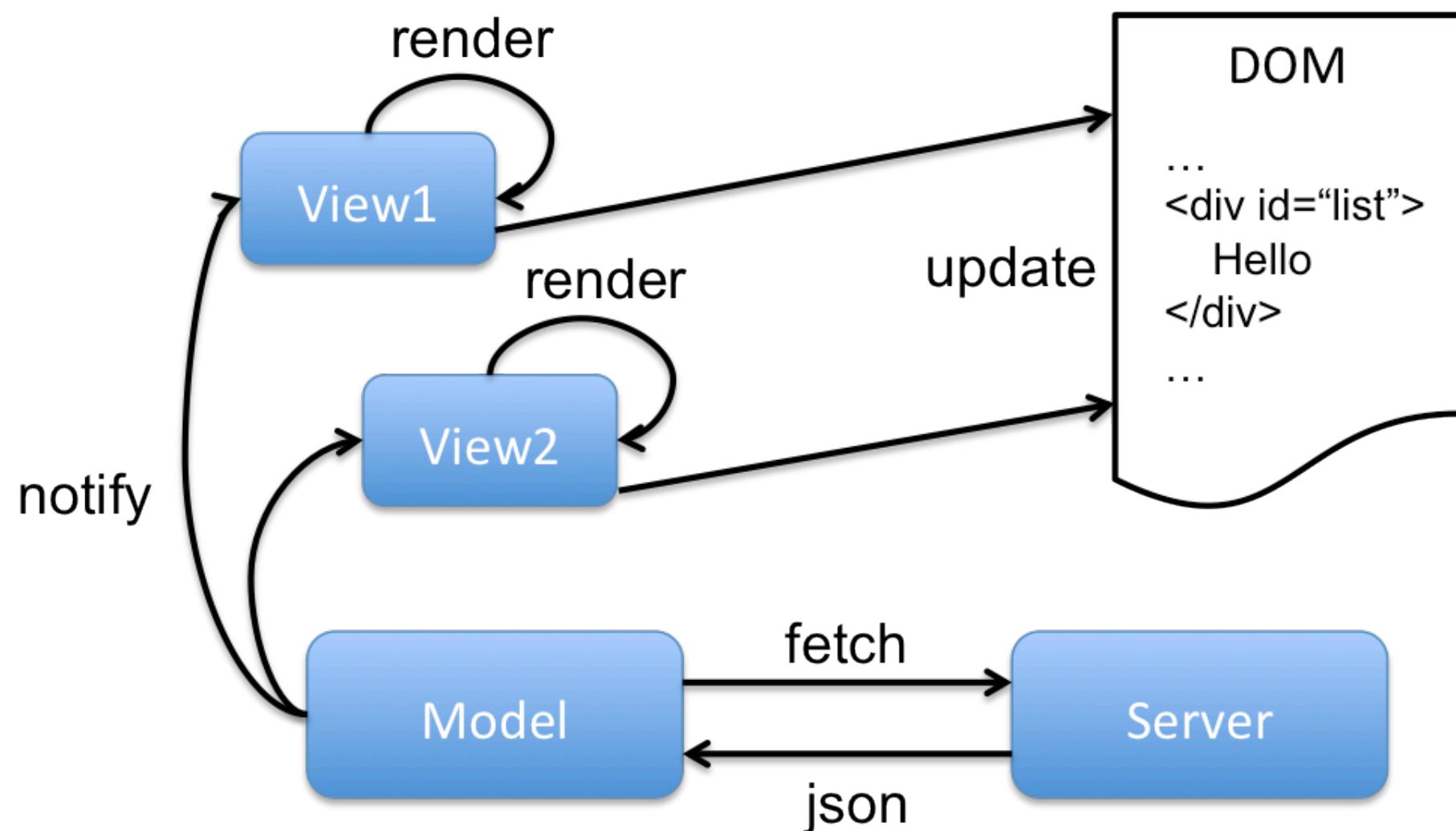
# Backbone.js - View



# Backbone.js - User Triggers an Update to the Server



# Backbone.js - Server's Changes trigger an Update



## Backbone Blog - Model mvcBackboneBlog

```
var Article = Backbone.Model.extend ({  
});  
  
var Articles = Backbone.Collection.extend ({  
  model: Article,  
  url: '/articles'  
});
```

## Backbone Blog - Item View [mvcBackboneBlog](#)

```
//View for rendering one entry of the blog
var ItemView = Backbone.View.extend ({
  tagName: "li",
  events: {
    "blur [contenteditable)": "saveValues"
  },
  initialize: function() {
    this.model.bind('change', this.render, this);
    this.template = Templates.article;
  },
  render: function() {
    $(this.el).html( this.template(this.model.toJSON()) );
    return this;
  },
  saveValues: function() {
    this.model.save({
      title: this.$("[data-name='title']").html(),
      content: this.$("[data-name='content']").html()
    },{silent: true});
  }
});
```

# Backbone Blog - List View [mvcBackboneBlog](#)

```
//View for rendering the list of entries
var ListView = Backbone.View.extend ({
  el: $("#articlesList"),
  events: {
  },
  initialize: function() {
    this.collection.bind('reset', this.render, this);
    this.collection.bind('all', this.render, this);
  },
  render: function() {
    var el = this.$el;
    el.empty();
    this.collection.each(function(item) {
      var itemView = new ItemView({model: item});
      el.append(itemView.render().el);
    });
    this.$el.listview('refresh');
    return this;
  },
});
```

# Backbone Blog - New View mvcBackboneBlog

```
//View for creating a new entry
var NewView = Backbone.View.extend({
  el: $("#new"),
  events: { "click #postEntry": "createNew" },
  initialize: function() {
    this.title = this.$("#title");
    this.content = this.$("#content");
  },
  createNew: function() {
    this$(".invalid").removeClass("invalid");
    if (this$(":invalid").length) {
      this$(":invalid").addClass("invalid");
      return false;
    }
    this.collection.create({
      title: this.title.val(),
      content: this.content.val(),
      email: localStorage.email,
      name: localStorage.name
    }, {at: 0});
    this.title.val("");
    this.content.val("");
  }
});
```

# Backbone Blog - Options View mvcBackboneBlog

```
//View for editing the options
var OptionsView = Backbone.View.extend({
  el: $("#options"),
  events: {
    "click #saveOpt": "saveOptions"
  },
  initialize: function() {
    this.name = this.$("#optName");
    this.email = this.$("#optEmail");

    // Assign the options from the local storage
    this.name.val(localStorage.name);
    this.email.val(localStorage.email);
  },
  saveOptions: function() {
    this$(".invalid").removeClass("invalid")
    if (this$(".:invalid").length) {
      this$(".:invalid").addClass("invalid");
      return false;
    }
    localStorage.name = this.name.val();
    localStorage.email = this.email.val();
  }
});
```

# Backbone Blog - Initialization

mvcBackboneBlog

```
var Templates = {};
var articles;

$(function() {
    //Load the templates and store them in a global variable
    $('script[type="text/x-handlebars-template"]').each(function () {
        Templates[this.id] = Handlebars.compile($(this).html());
    });

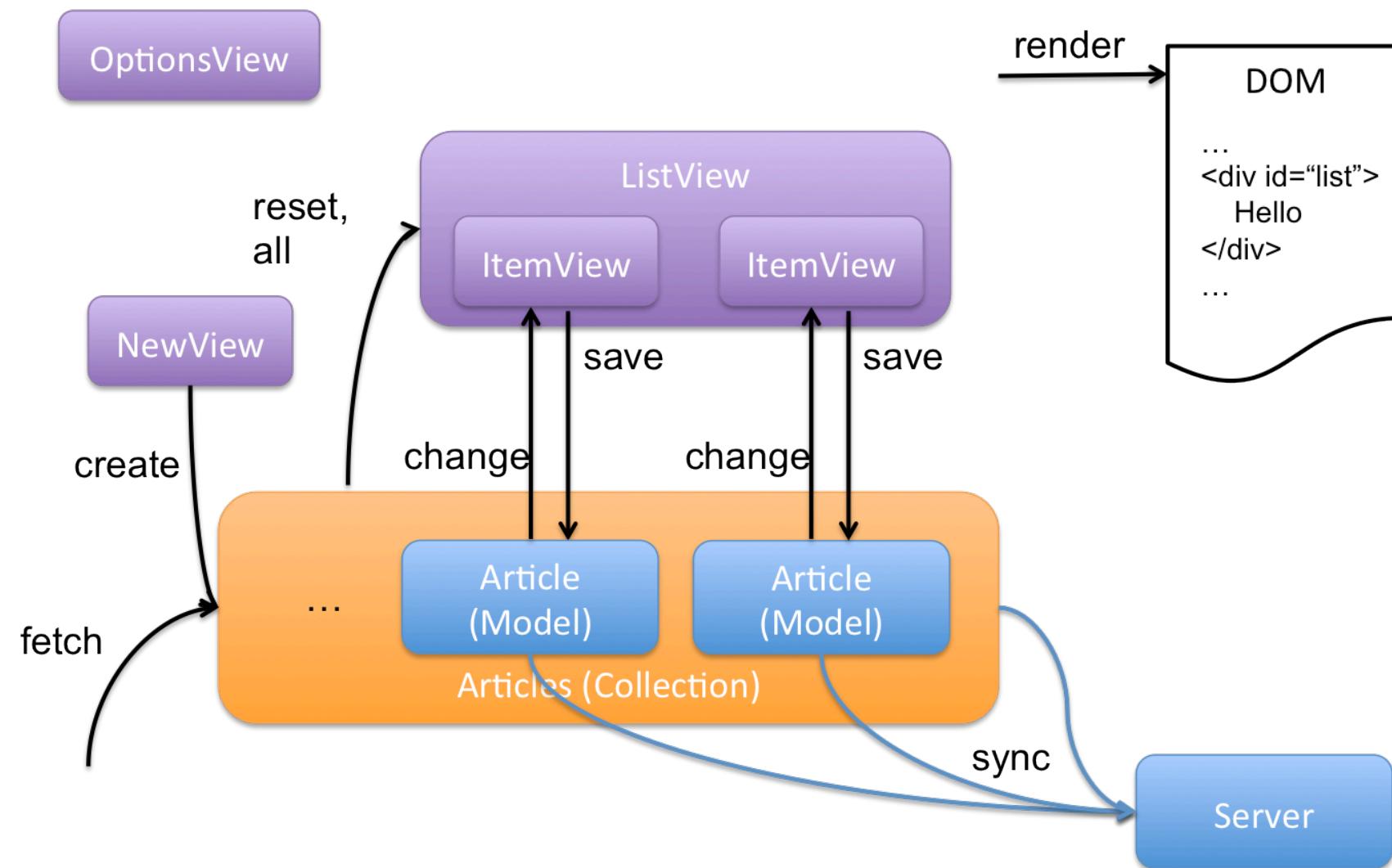
    //Trigger an update of the articles collection
    $("#refresh").live('click',function () {
        articles.fetch();
    });

    //Instantiate the collection of articles
    var articles = new Articles();

    //Instantiate the views
    var listView = new ListView({collection: articles});
    var newView = new NewView({collection: articles});
    var optionsView = new OptionsView();

    //Fetch the latest articles and trigger an update of the views
    articles.fetch();
})
```

# Backbone Blog - Diagram mvcBackboneBlog



# Backbone.js - Summary

- MVC Pattern
  - User or server triggers data change, never update view directly
  - Data update triggers change event
  - Views can subscribe to change events and re-render
- Models are clearly separated from view logic. One HTTP Sync request can cause many UI updates
- Support for templates (default from Underscore)
- Persistence comes from Backbone.sync
  - The default is RESTful JSON
  - You can override sync() to use XML or LocalStorage
- Optimized for simplicity and speed

## Backbone.js - Local Storage

- The extension `backbone.localStorage.js` provides the persistence with local storage
- The bad news is that the extension disables the communication with the server (rewrites `Backbone.sync` )
- Original `Backbone.sync` is available at `Backbone.ajaxSync`

# Blog with localStorage

[mvcBackboneLocalBlog](#)

## layout.haml

```
...
%script{type="text/javascript" src="backbone.js"}
%script{type="text/javascript" src="backbone.localStorage.js"}
...
```

## JS

```
...
var Articles = Backbone.Collection.extend ({
  model: Article,
  localStorage: new Backbone.LocalStorage("Articles")
});
...
```

# Pre-compilation of the assets

- If you are developing a standalone application (without a server) you can compile the Haml files to HTML (e.g. using a [Rakefile](#) )
- If you are developing a server-based webapp there are tools for precompiling the assets (including the Handlebars.js templates):
  - [Handlebar.js compilation](#)
  - [Sprockets](#)
- [Static Web Site Generation](#)

# Synchronization of models

Racer but not ready yet

Backbone.js on the server

Backbone and local storage