Single Page Applications

Javier Espinosa, PhD
javier.espinosa@imag.fr

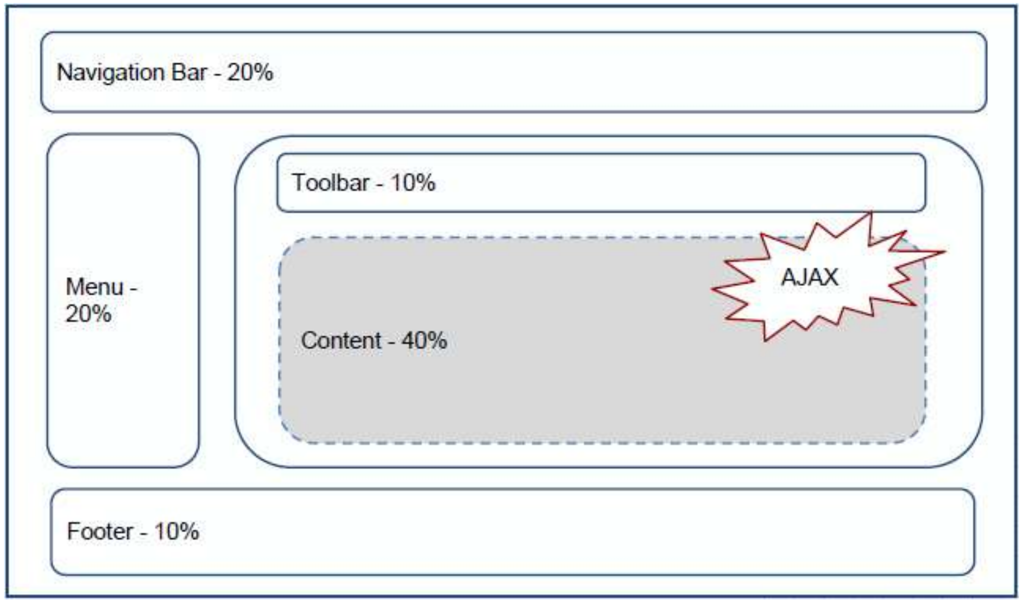
Outline

- **Single Page Applications**
  - AJAX in a nutshell
  - MVC pattern
  - Case study: AngularJS

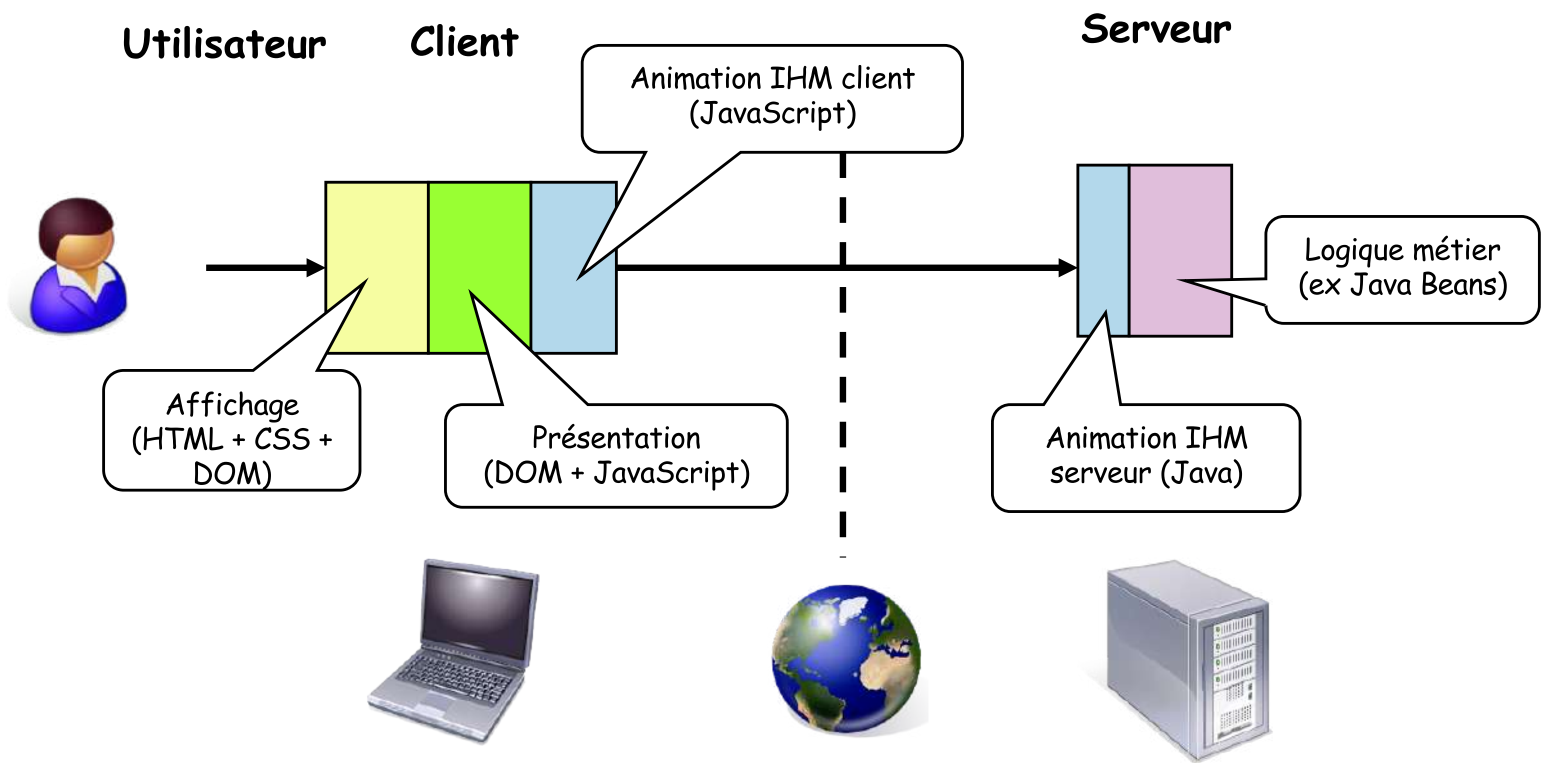
AJAX in a Nutshell

Single Page Applications (SPA)

- Resources are dynamically loaded and added to the page as necessary
- Inspired in native application



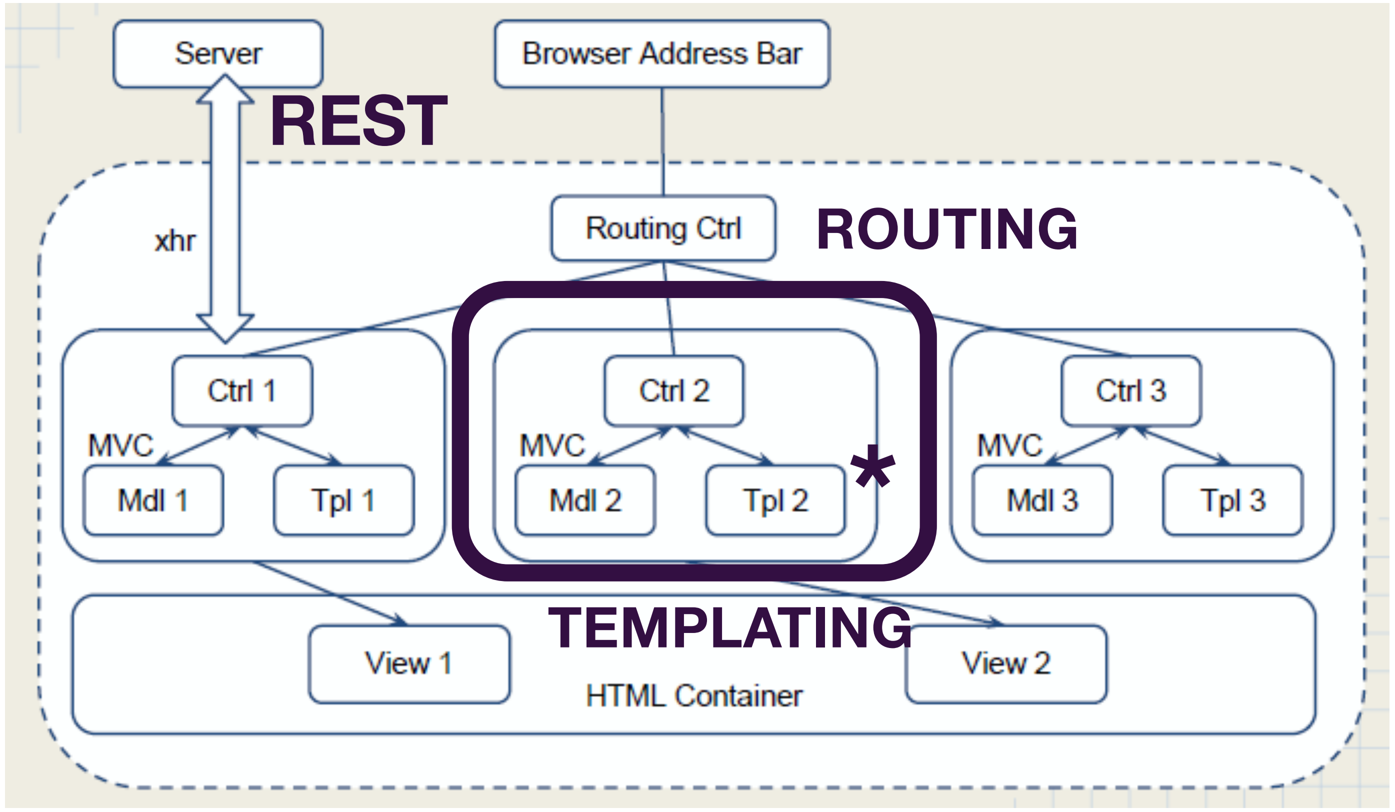
Single Page Applications (SPA)


Utilisateur
Client
Serveur
Animation IHM client (JavaScript)
Logique métier (ex Java Beans)
Affichage (HTML + CSS + DOM)
Présentation (DOM + JavaScript)
Animation IHM serveur (Java)

New problems and solutions

- Problems
  1. Mimic static addresses (http://mysite.com/...) and manage browser history
  2. Mix HTML strings and Javascript
  3. Handle Ajax callbacks
- Solutions
  1. Routing (http://mysite.com/#/...)
  2. Templating (Javascript-HTML)
  3. Providers + REST

SPA architecture



MVC Pattern

- Architectural pattern for implementing a interactive applications
- Introduced in the 1970s as part of Smalltalk
- Classifies objects based on their roles in the application
  - **Model**: object(s) representing the application domain
  - **View**: objects presenting the model to a user (graphic part)
  - **Controller**: glue between models and views

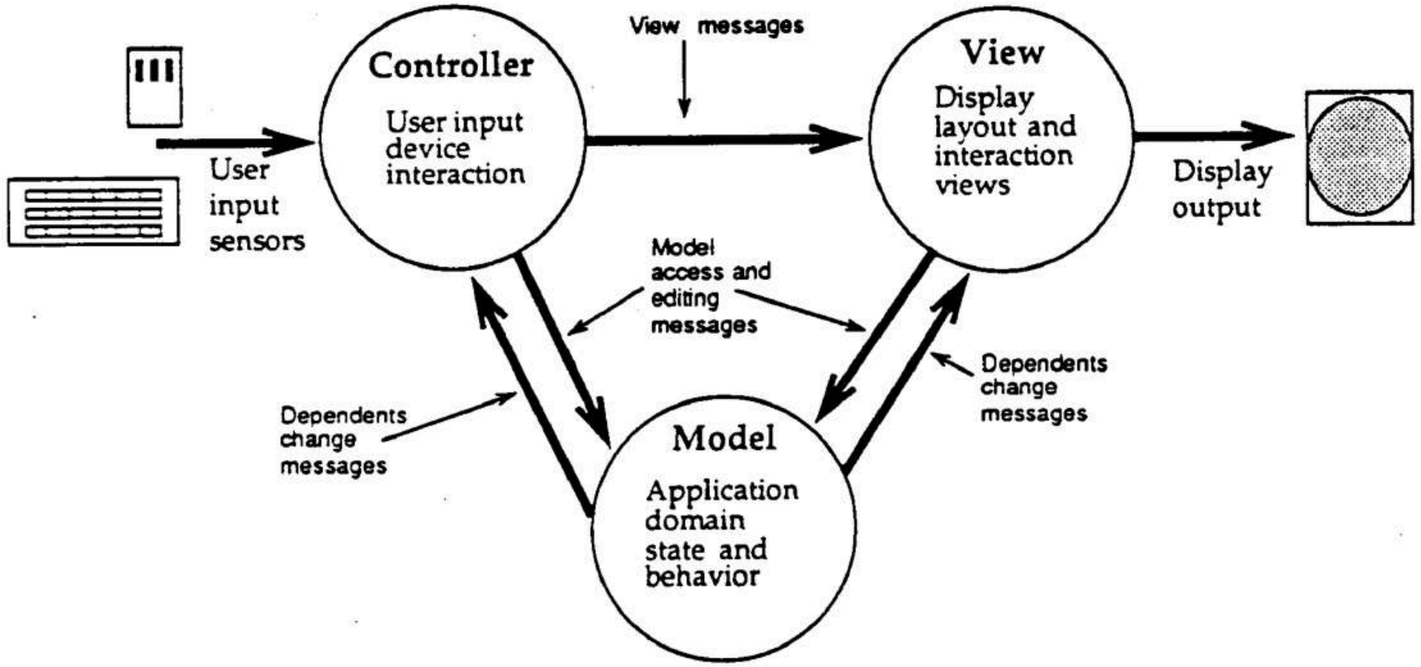
MVC Pattern Benefits

- Organization
- Rapid Application Development
- Reusing Code
- Parallel development
- The views and application behavior should reflect the manipulations of the data immediately

MVC Pattern Implementations

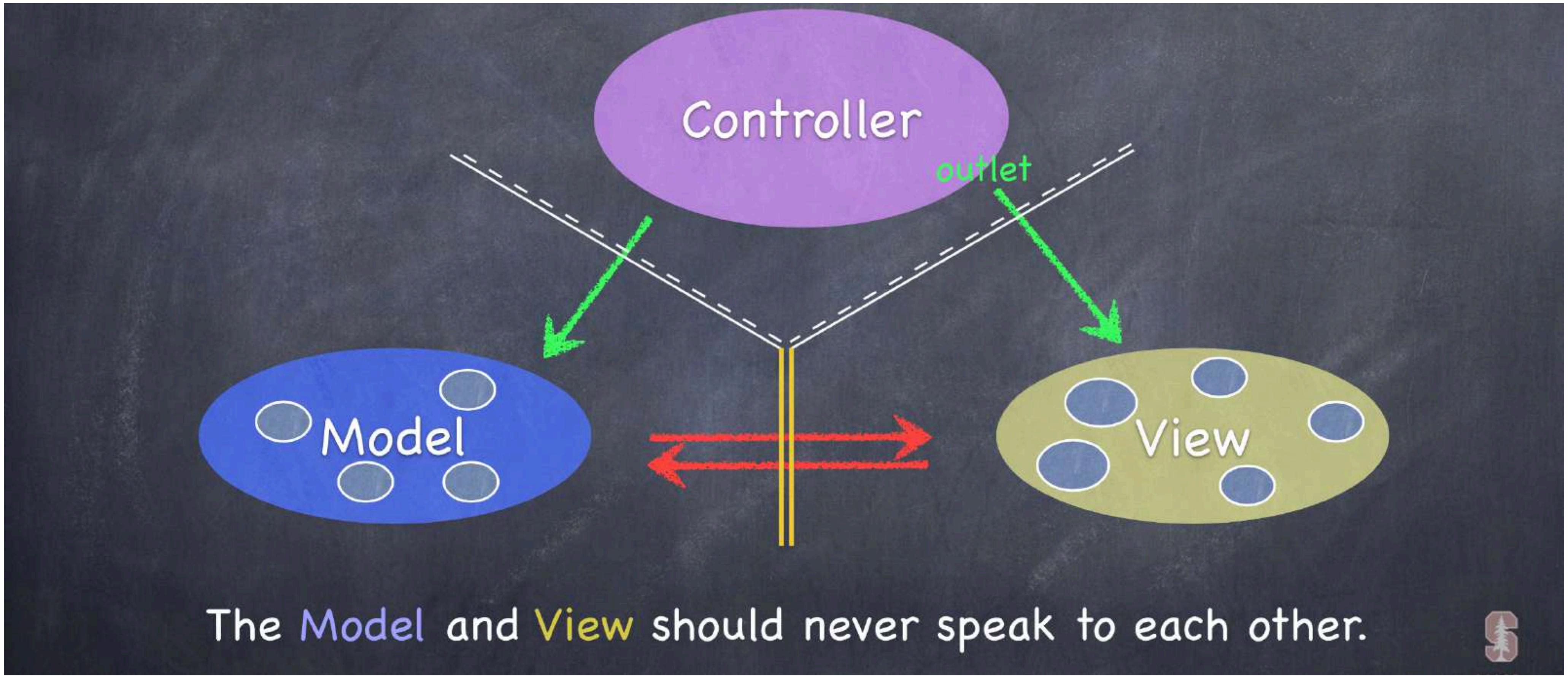
- Popular JS frameworks
  - Angular
  - Backbone
  - Ember
  - Knockout
- Note*: the role of controller greatly varies in frameworks*
- Other MVC like patterns
  - **MVVM** (Model-View-ViewModel)
  - **MVP** (Model-View-Presentation)

Original MVC Interaction Pattern

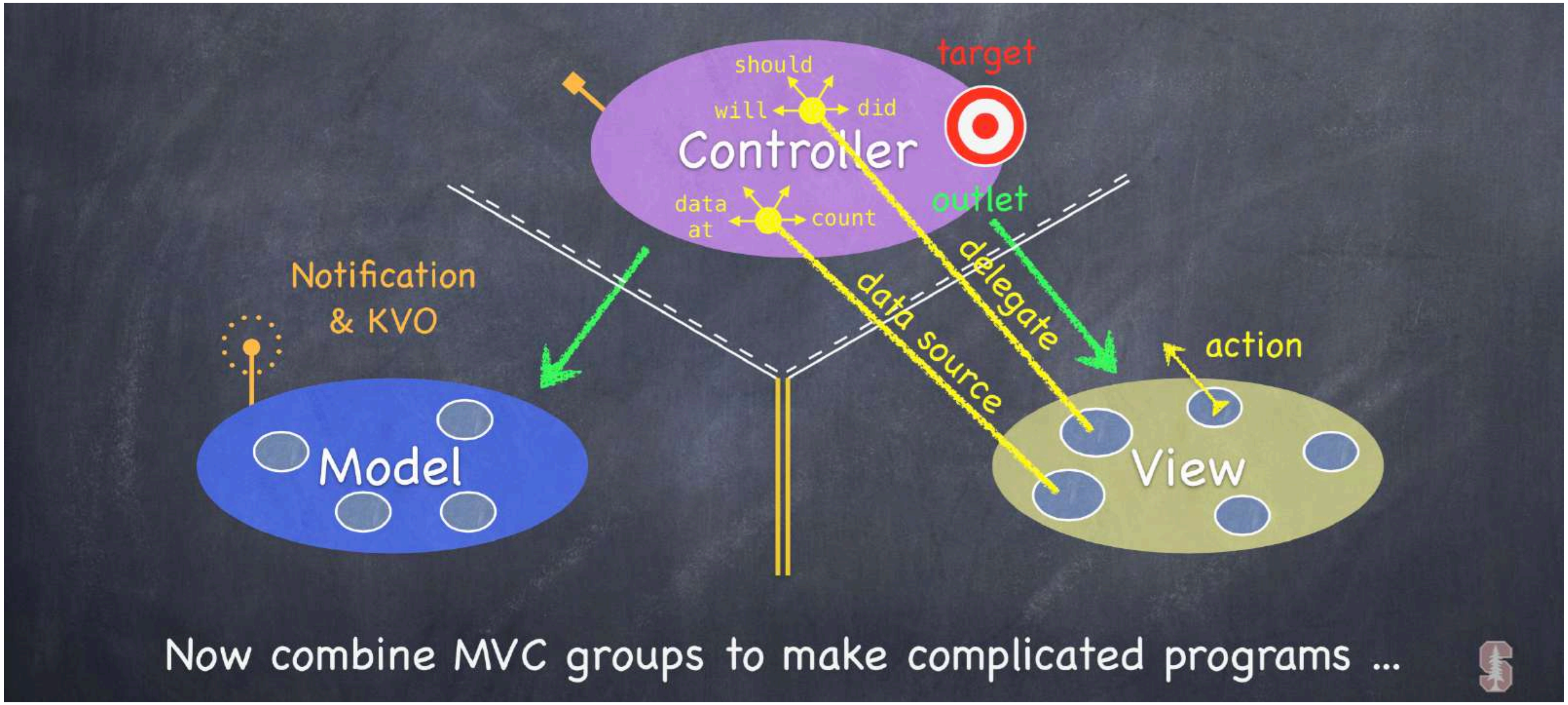


Controller
User input device interaction
View messages
View
Display layout and interaction views
Display output
User input sensors
Model access and editing messages
Dependents change messages
Dependents change messages
Model
Application domain state and behavior

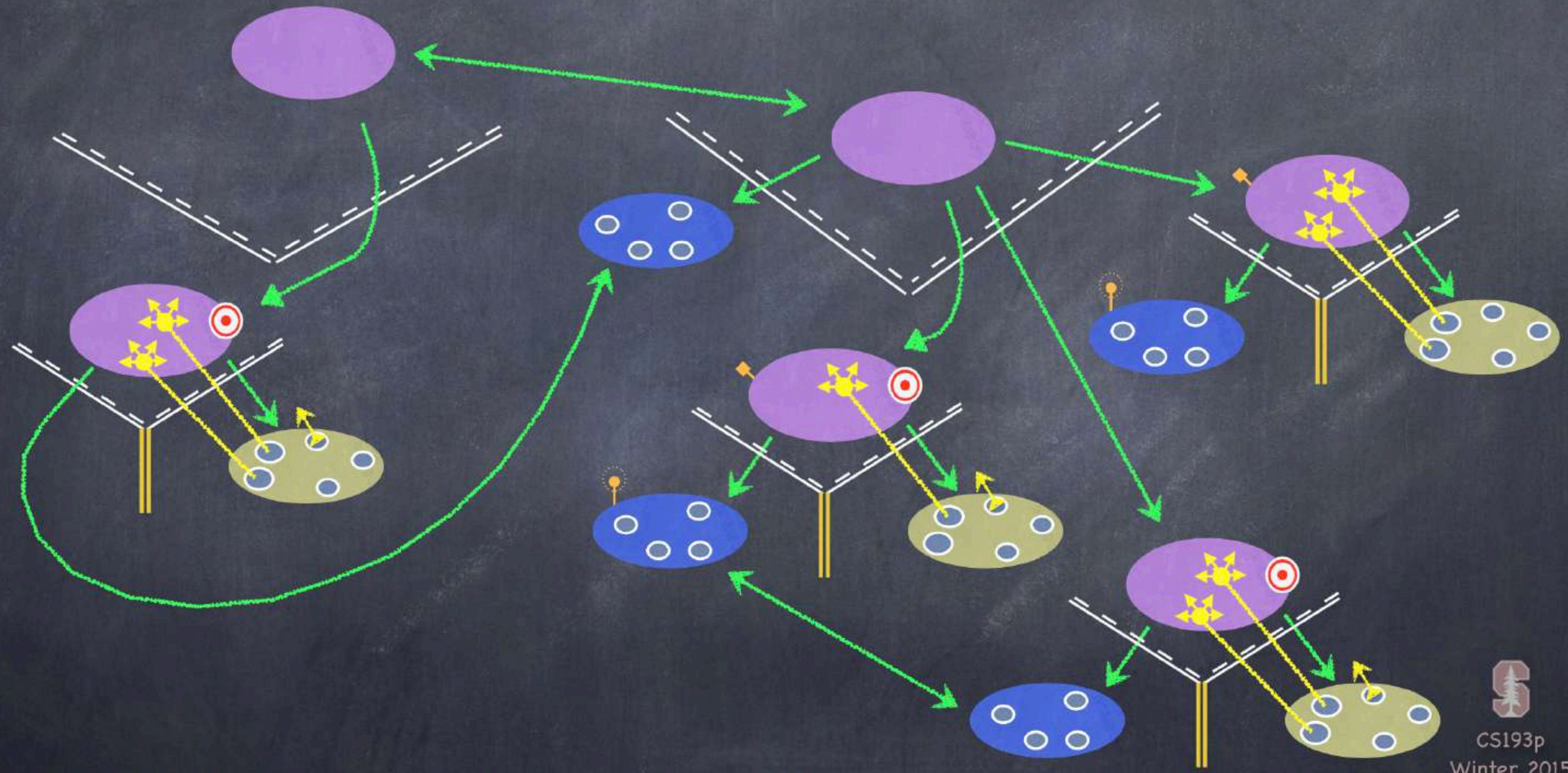
Apple MVC Interaction Pattern



Apple MVC Interaction Pattern

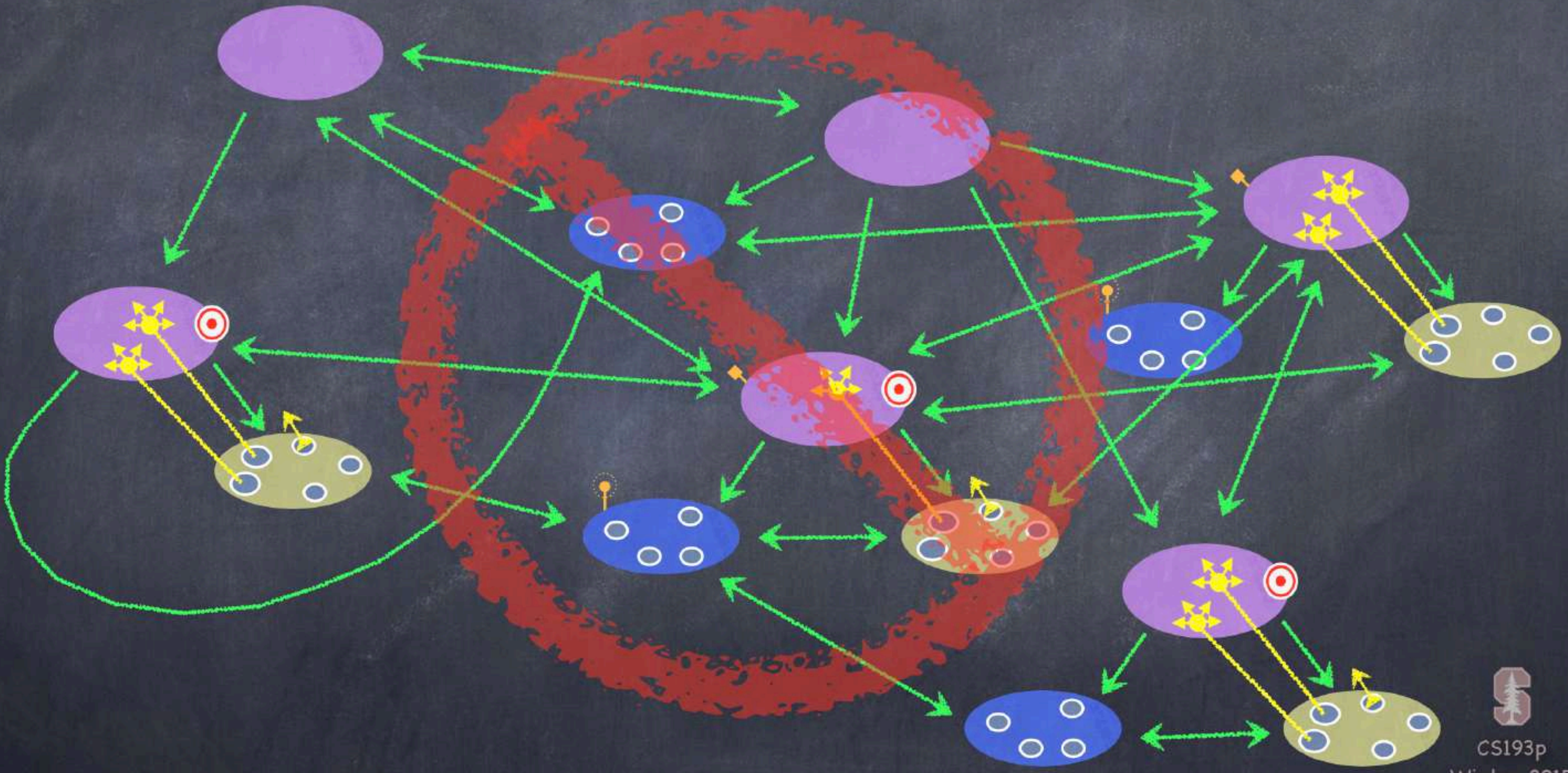


MVCs working together



CS193p
Winter 2015

MVCs not working together



CS193p
Winter 2015

MVC Methodology

- **Step 1: Models**
  Define the classes that would embody the special application domain specific information
  - It can be as simple as an integer or string
- **Step 2: Views**
  Define a user interface to the model by laying out a composite view (window) by "*plugging in*" instances taken from pre-defined UI classes
  - They request data from their model
- **Step 3: Controllers**
  - Define associations between a model and a view and the situations of interest

AngularJS

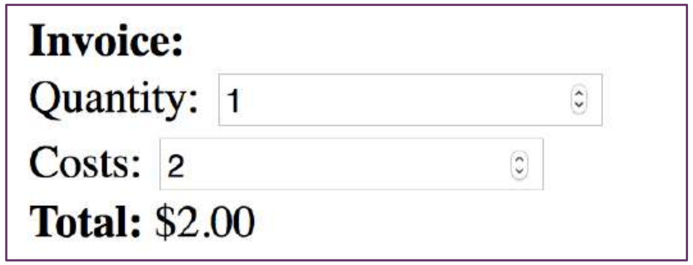
AngularJS Overview (i)

- Framework for building single page applications using MVC
- Extends HTML with declarative expressions for defining application's components (views, models)
  - Angular is what HTML would have been if it had been designed for applications
- Angular teaches the browser new tricks through directives
  - Data binding
  - Support for forms and form validation
  - DOM control structures for repeating, showing and hiding DOM fragments
- Conceived with testability in mind

AngularJS Overview (ii)

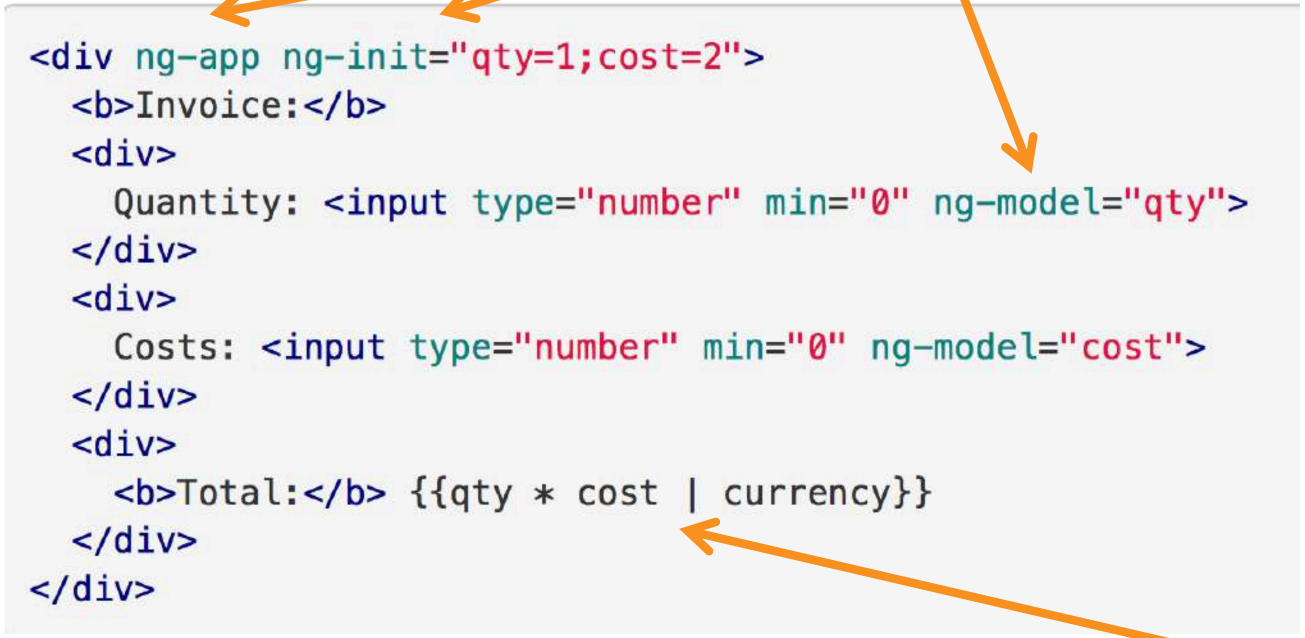
- Simplifies application development by presenting a higher level of abstraction to the developer
  - You don not manipulate the DOM directly
- Built with CRUD (*Create/Read/Update/Delete*) application in mind
  - Data-binding, form validation, reusable components, unit-testing, end-to-end testing
  - (**!!**) The majority of web applications are CRUD
- Not a good fit for Games and GUI editors
  - Intensive and tricky DOM manipulation
  - Use a library with a lower level of abstraction (e.g, jQuery)

Angular App Example


Invoice:
Quantity: 1
Costs: 2
Total: $2.00

Index.html

Directives



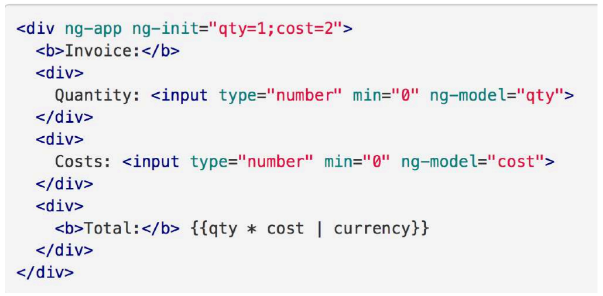
```
<div ng-app ng-init="qty=1;cost=2">
  <b>Invoice:</b>
  <div>
    Quantity: <input type="number" min="0" ng-model="qty">
  </div>
  <div>
    Costs: <input type="number" min="0" ng-model="cost">
  </div>
  <div>
    <b>Total:</b> {{qty * cost | currency}}
  </div>
</div>
```

Template

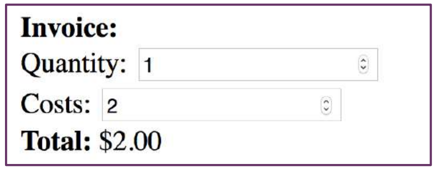
Expressions

Compilation Process

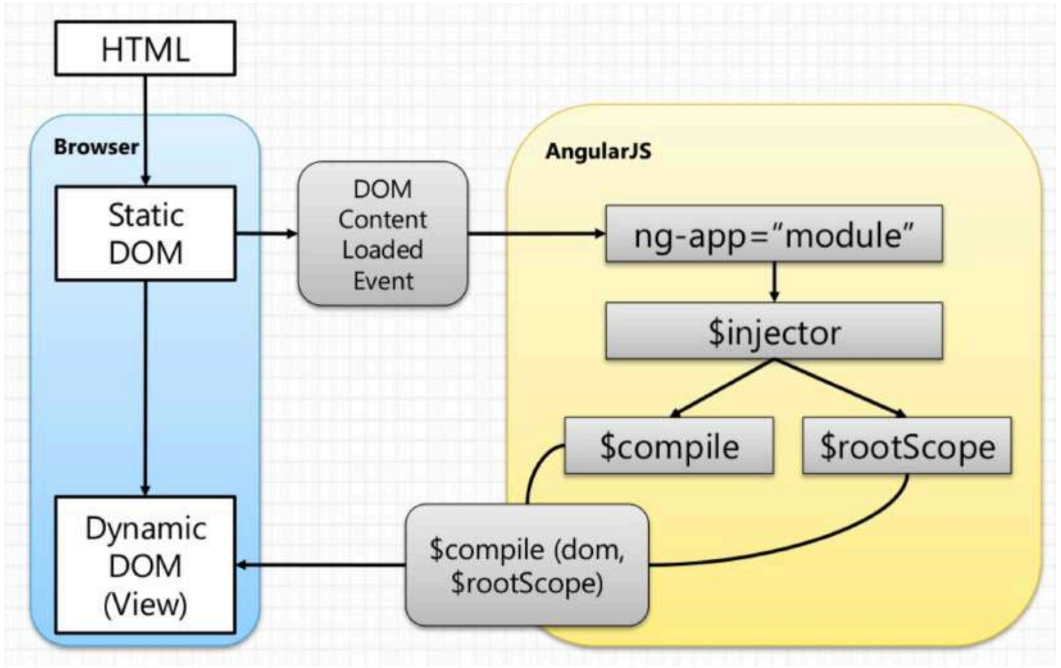
- When Angular initialize, it compiles (*parses and processes*) the template for producing a **view**



```
<div ng-app ng-init="qty=1;cost=2">
  <b>Invoice:</b>
  <div>
    Quantity: <input type="number" min="0" ng-model="qty">
  </div>
  <div>
    Costs: <input type="number" min="0" ng-model="cost">
  </div>
  <div>
    <b>Total:</b> {{qty * cost | currency}}
  </div>
</div>
```



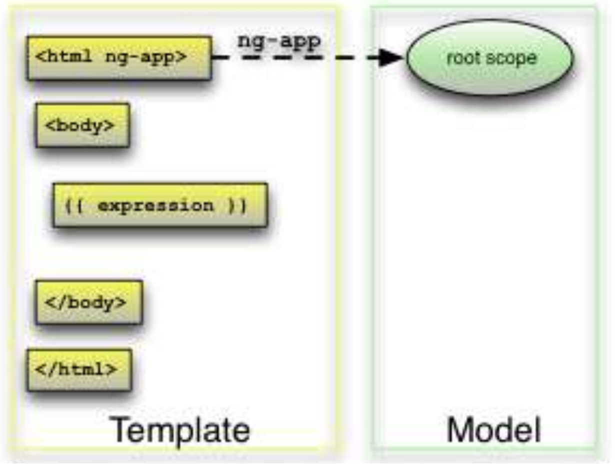
Invoice:
Quantity: 1
Costs: 2
Total: $2.00

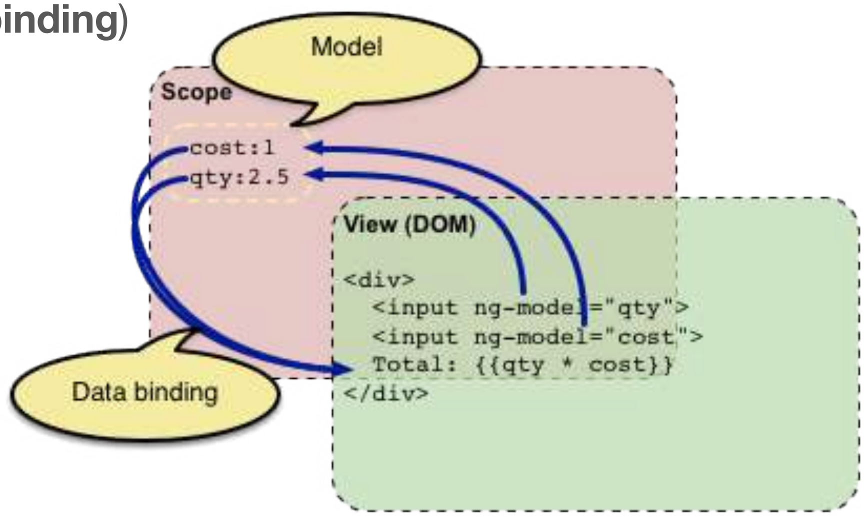


Compilation Phases

1. **Compilation**: traverse the DOM and collect all of the directives
   - The result is a linking function.

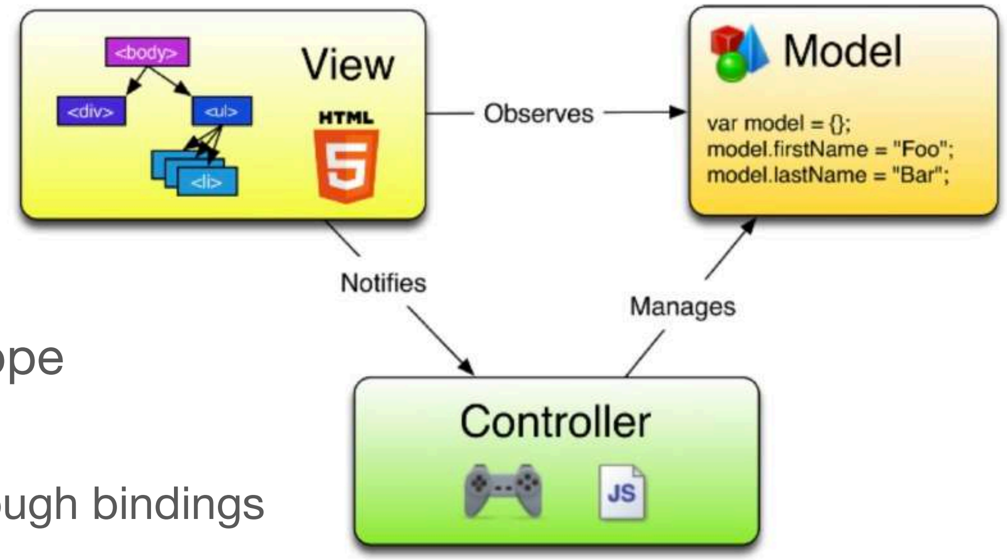
2. **Linking**: combine the directives with a scope and produce a *live view*
   - Any changes in the scope model are reflected in the view, and any user interactions with the view are reflected in the scope model (**2-way binding**)





```
<div>
  <input ng-model="qty">
  <input ng-model="cost">
  Total: {{qty * cost}}
</div>
```

AngularJS MVC



- Models are the properties of a scope
  - Scopes are attached to the DOM
  - Scope properties are accessed through bindings
- Views are the template (HTML with data bindings) that is presented to the user
- Controllers contains the business logic behind the application to decorate the scope with functions and values

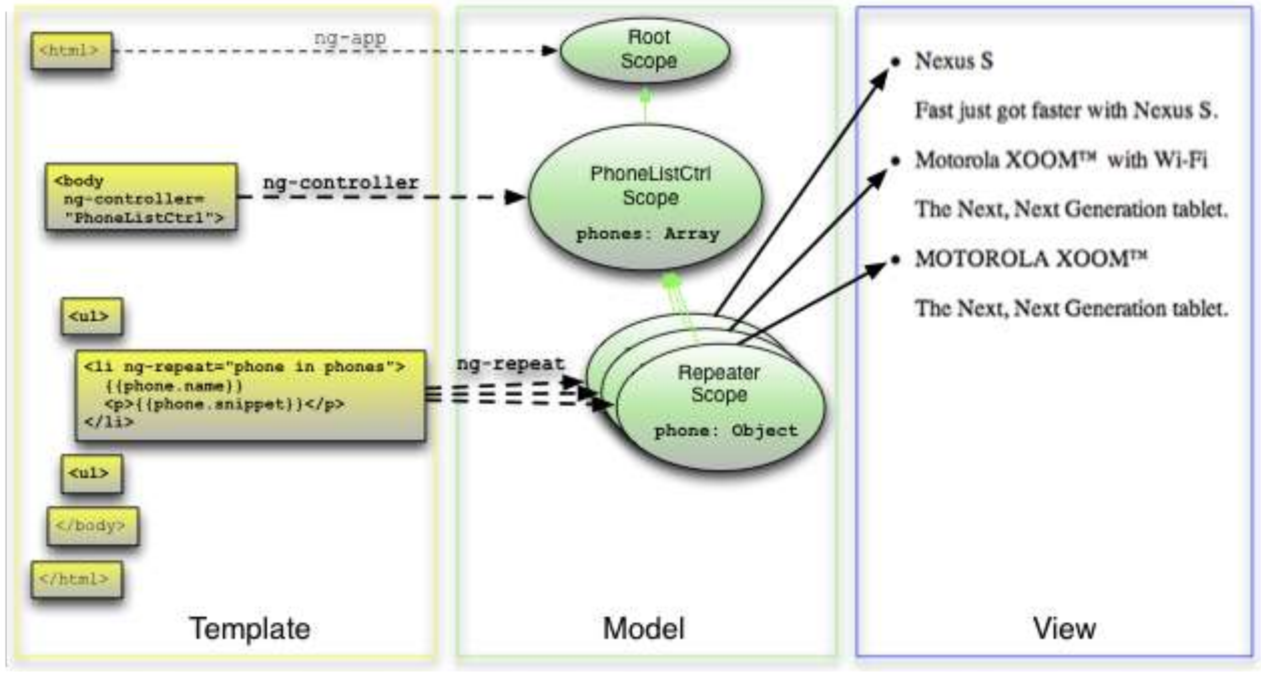
Directives

- Annotations on DOM elements (e.g. attribute, element name, comment or CSS class)
- Tell the compiler (*$compile*) to attach a specified behavior to that DOM element
- Examples:
  - **ng-app**: specifies that the HTML element will be manage by angular
  - **ng-repeat**: instantiates a template once per item from a collection
  - **ng-hide**: shows/hides an HTML element based on the evaluation of an expression
  - **ng-src**: loads an image based on an expression

Directives

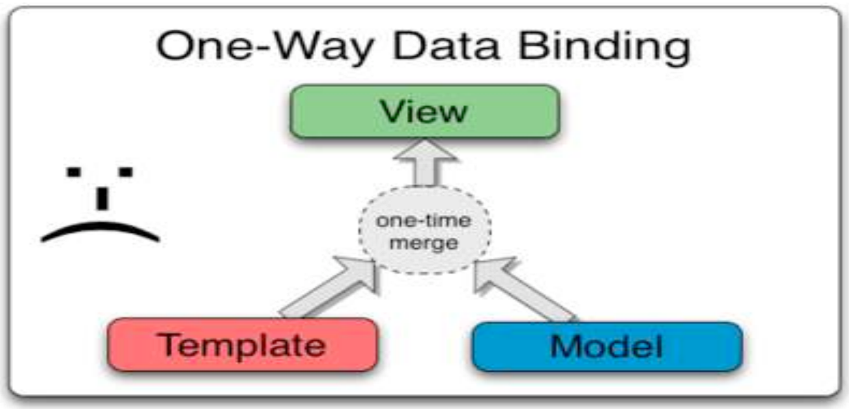
- **ng-repeat** example

```
<ul>
  <li ng-repeat="phone in phones">
    <span>{{phone.name}}</span>
    <p>{{phone.snippet}}</p>
  </li>
</ul>
```

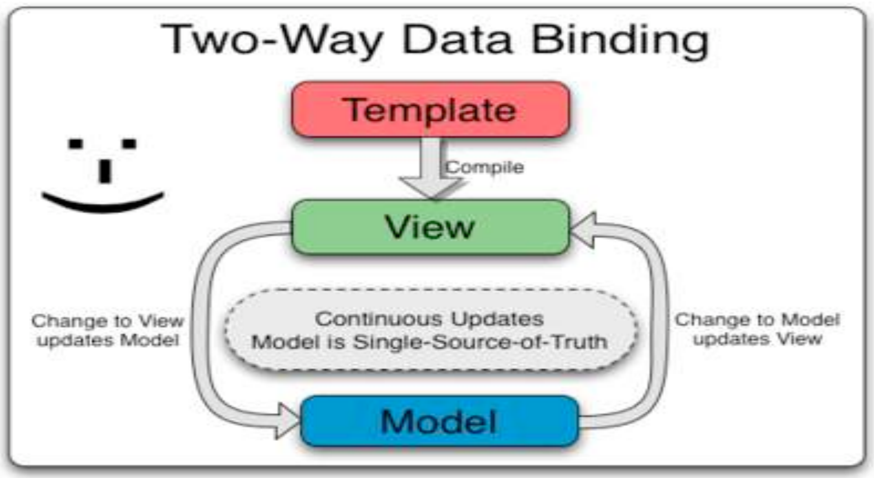


Data Binding

- Automatic synchronization of data between the model and view components



Most template systems



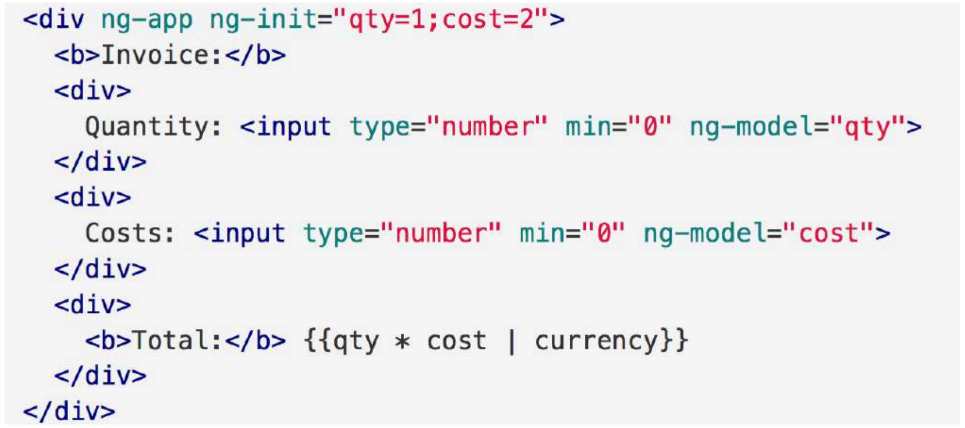
Two-Way Data Binding
Template
Compile
View
Change to View updates Model
Continuous Updates Model is Single-Source-of-Truth
Change to Model updates View
Model

Changes immediately reflected
(Live view)

Expressions

- JavaScript-like code snippet in a template that allows to read and write variables
  - Syntax: {{ expression | filter }}

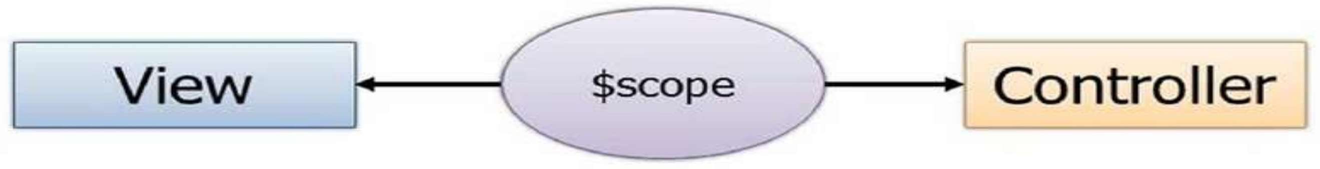
- Expressions bind the view and the model
  - Angular provides a scope for the variables accessible to expressions



```html
<div ng-app ng-init="qty=1;cost=2">
  <b>Invoice:</b>
  <div>
    Quantity: <input type="number" min="0" ng-model="qty">
  </div>
  <div>
    Costs: <input type="number" min="0" ng-model="cost">
  </div>
  <div>
    <b>Total:</b> {{qty * cost | currency}}
  </div>
</div>
```

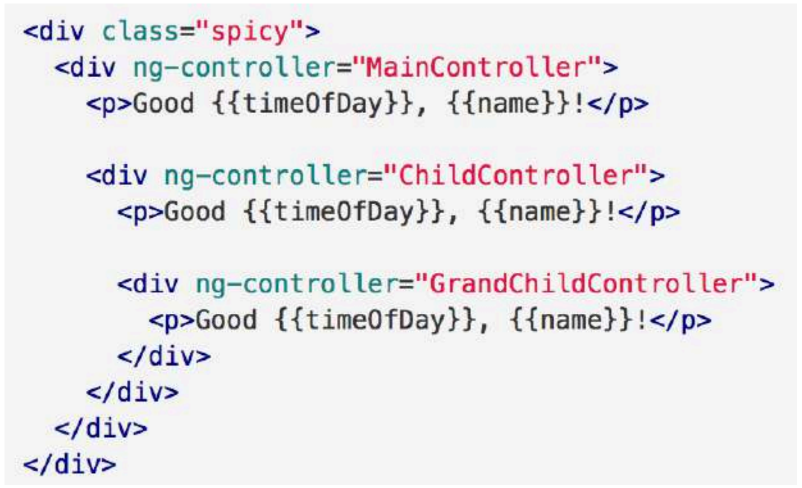
Controllers (i)

- The controller drives things:
  - Controls what data gets bound into the view (*i.e. prepares data for the view*)
  - Define the business logic needed by a single view
- A controller is implemented via a JavaScript function that is used to augment the Angular Scope with data and logic



Controllers (ii)

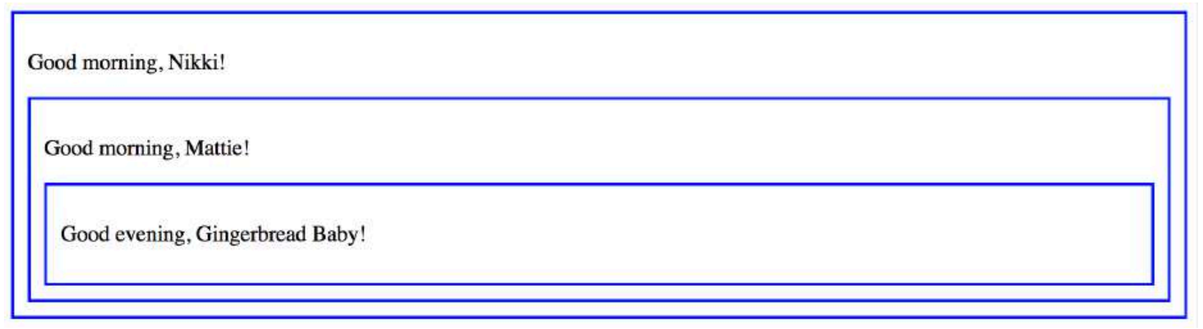
- Controller are attached to the DOM via the ng-controller directive
- When angular finds a ng-controller directive, it instantiates a new Controller object creating a new child scope
  - Scopes are arranged in hierarchical structure



```
<div class="spicy">
  <div ng-controller="MainController">
    <p>Good {{timeOfDay}}, {{name}}!</p>

    <div ng-controller="ChildController">
      <p>Good {{timeOfDay}}, {{name}}!</p>

      <div ng-controller="GrandChildController">
        <p>Good {{timeOfDay}}, {{name}}!</p>
      </div>
    </div>
  </div>
</div>
```

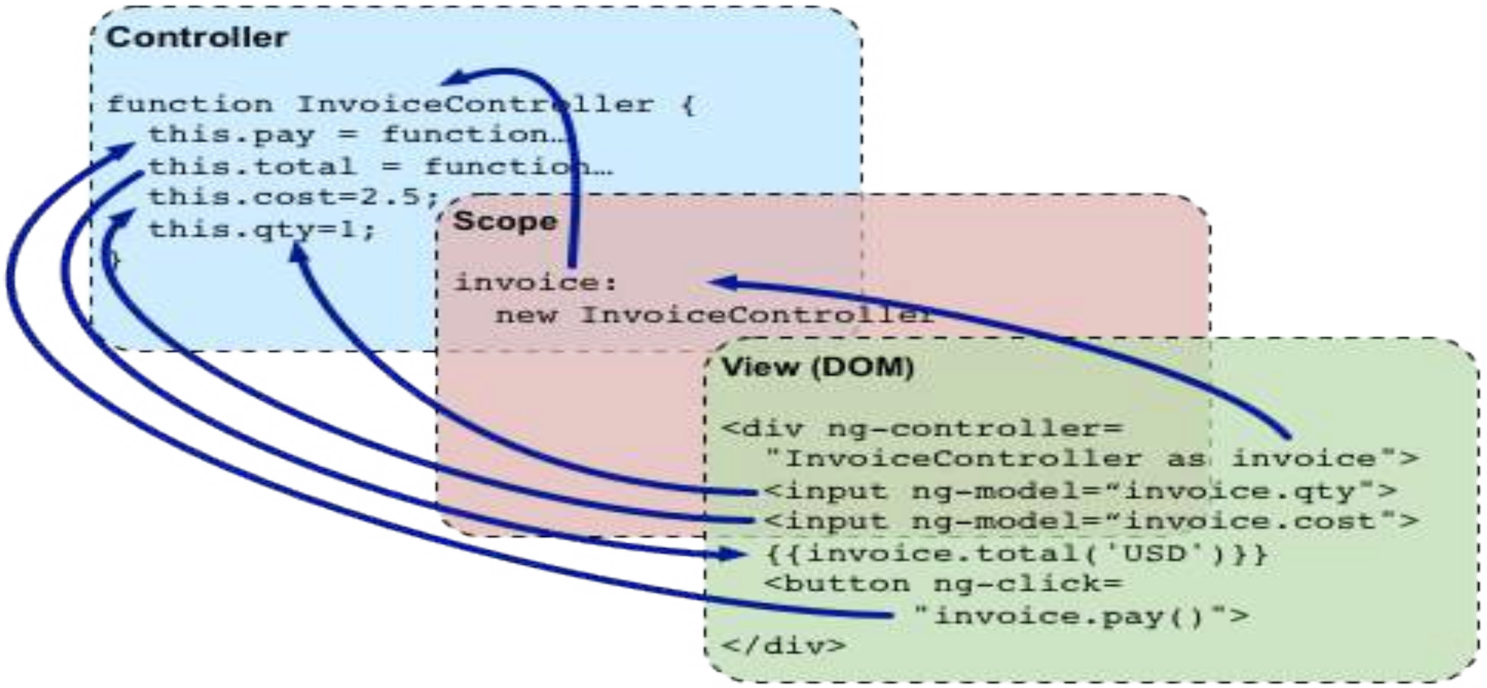


Good morning, Nikki!

Good morning, Mattie!

Good evening, Gingerbread Baby!

Controller-Scope-View



```
function InvoiceController {
  this.pay = function…
  this.total = function…
  this.cost=2.5;
  this.qty=1;
}
```

```
invoice:
  new InvoiceController
```

```
<div ng-controller=
  "InvoiceController as invoice">
  <input ng-model="invoice.qty">
  <input ng-model="invoice.cost">
  {{invoice.total('USD')}}
  <button ng-click=
       "invoice.pay()">
</div>
```

HANDS ON

?