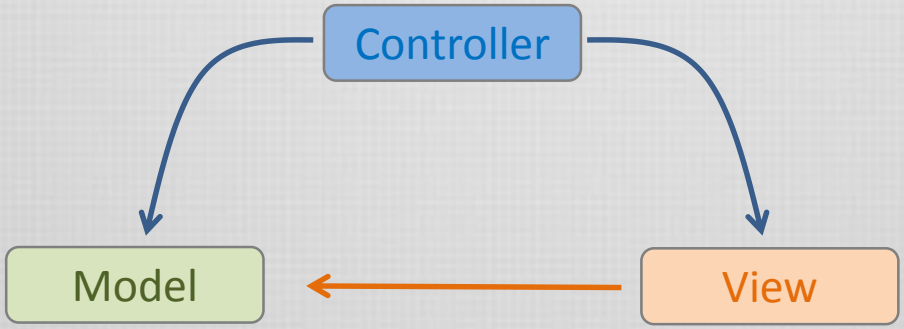
# Ingeniería de Aplicaciones Web

*Diego C. Martínez*

Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur

# Patrón general de diseño: MVC

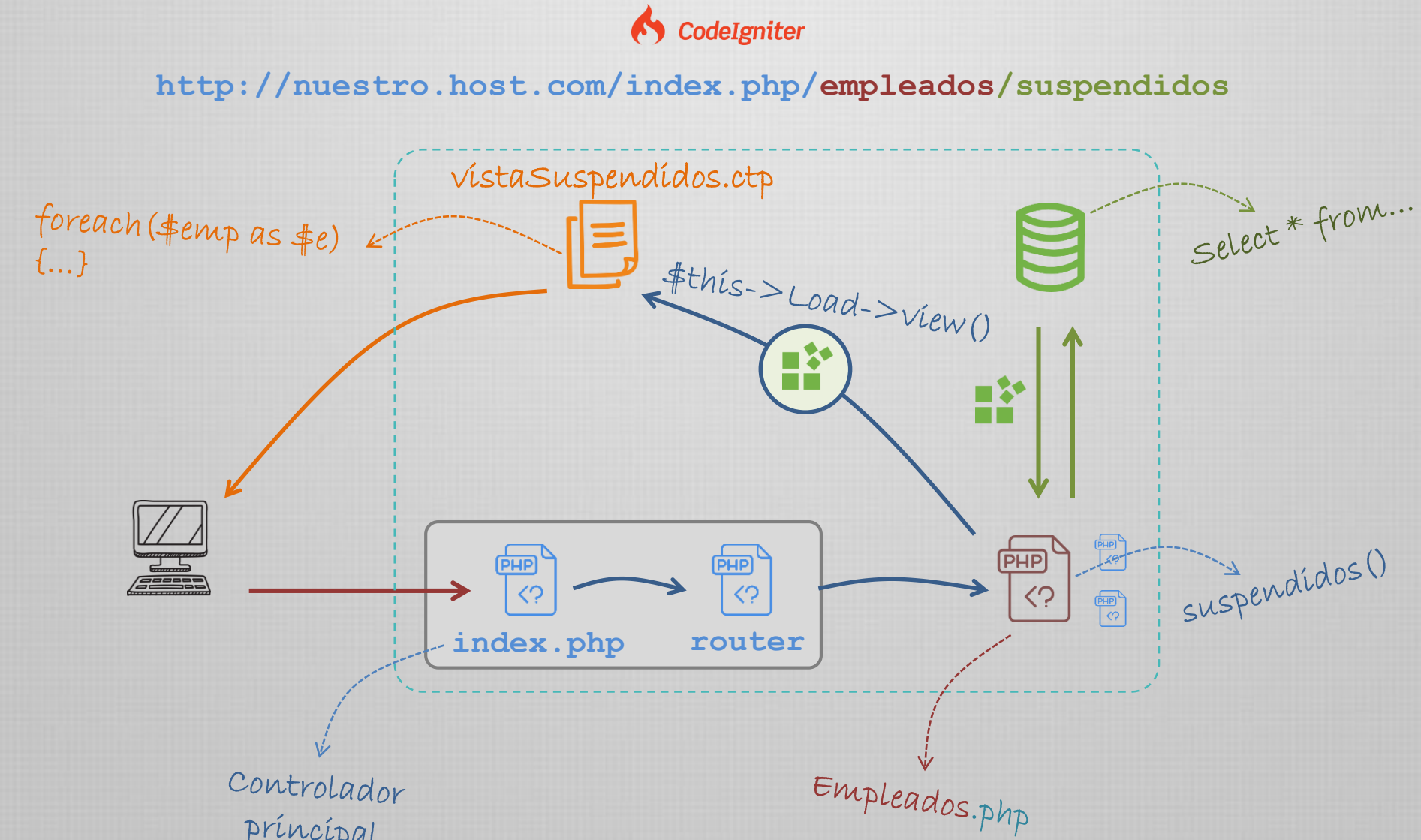
MVC es el patrón arquitectónico predominante en las aplicaciones web



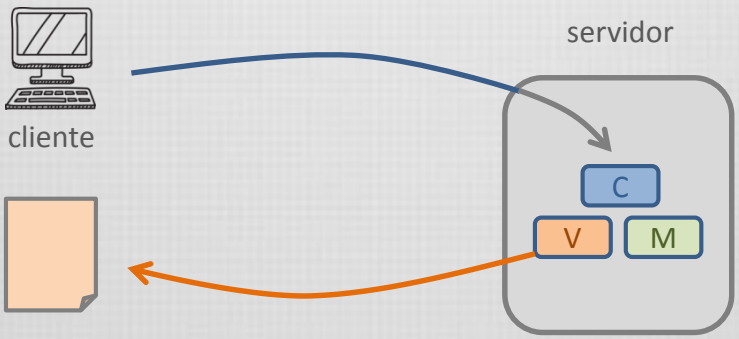

*El Controlador administra el Modelo y la Vista.*

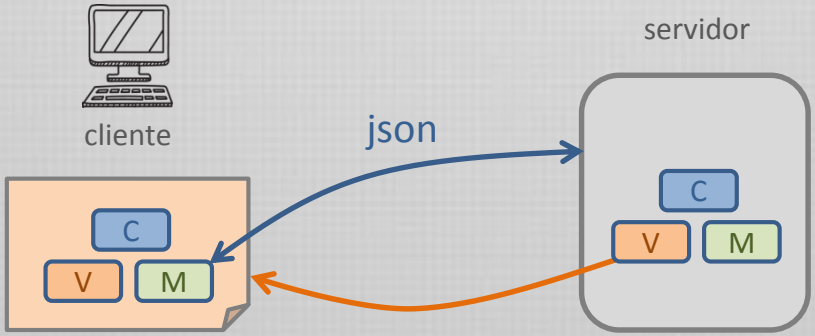
*La Vista es responsable de observar el Modelo para exteriorizar los datos.*

# MVC – Lado Servidor - Ejemplo

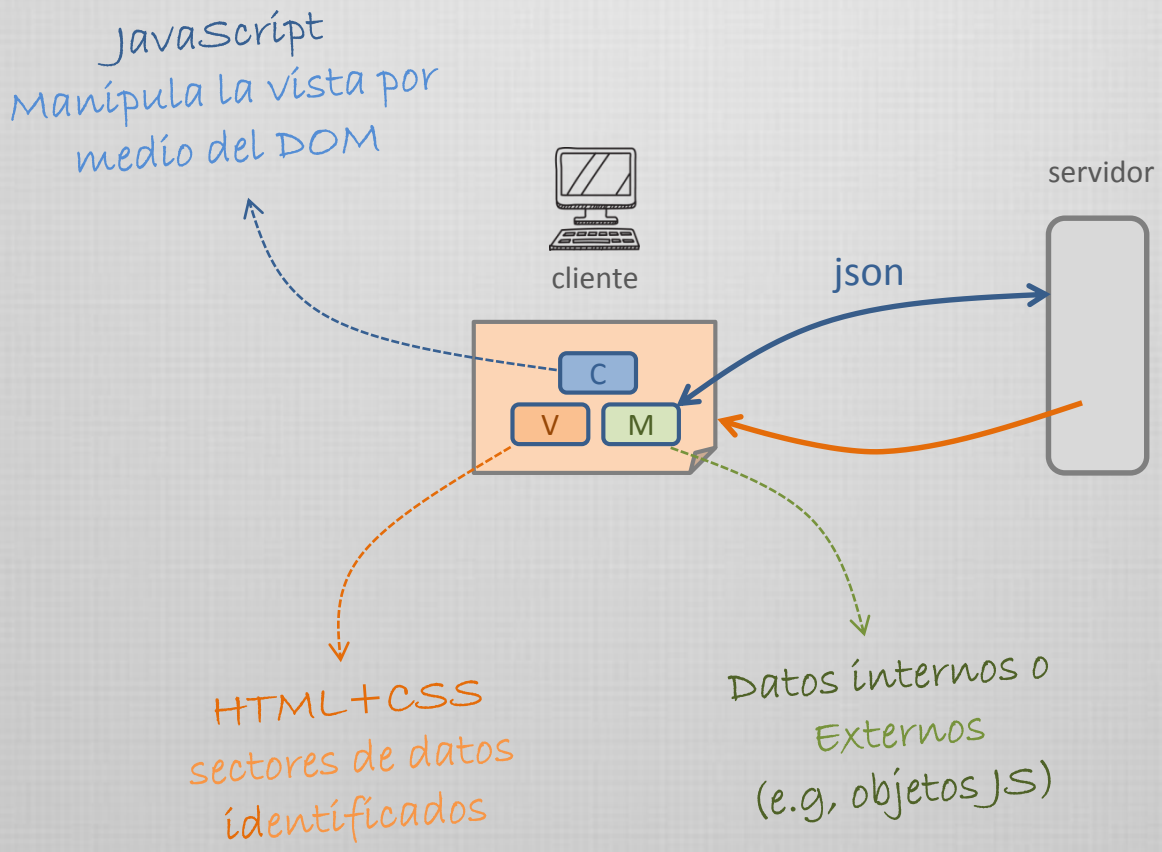


# MVC – Clientes y Servidores



# MVC del lado cliente



# MPA vs SPA

## Multi-page Applications


Una página completa en cada pedido

Mapeo directo entre URL y resultado obtenido

Ideal para Search Engine Optimization

Muchas soluciones enlatadas en el Mercado

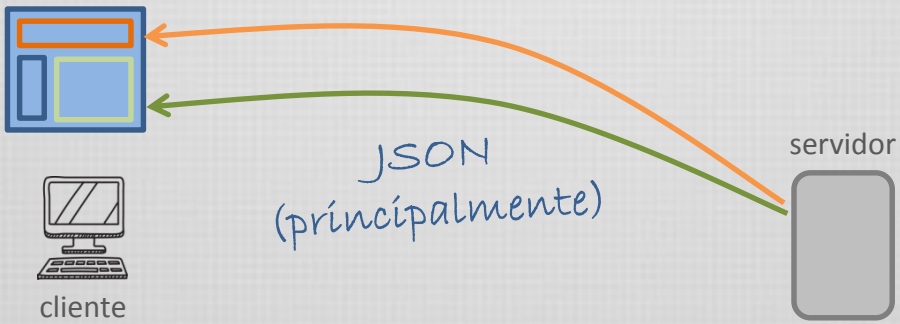
Desarrollo simple *per-page*

Mayor acoplamiento entre el front-end y el back-end

Requiere mayor adaptación al scenario móvil

# MPA vs SPA

## Single-page Applications




Una página fija, con mucha interacción *background* con el servidor

Mayor velocidad en el uso

Adecuado para aplicaciones móviles

Desarrollo ágil

Debug localizado en el navegador

Menor seguridad

Inadecuado para Search Engine Optimization

# MPA vs SPA

## Multi-page Applications


THE WALL STREET JOURNAL.



## Single-page Applications

# MVC - Alternativas

Hay variaciones al modelo MVC tradicional, especialmente del lado cliente

MV* → El concepto de controlador dedicado no es mandatorio.

MVP → *Model-View-Presenter*.
La diferencia es la codificación del Presenter, que se comunica con la Vista por medio de una interfaz.
Usualmente no hay vinculo entre el Modelo y la Vista

MVVM → *Model-View-ViewModel*
Impulsado por Microsoft para su estructura de interfaces WPF-Silverlight
El ViewModel es una abstracción de la vista e implementa su comportamiento.

MVW → *Model-View-Whatever*
Whatever works

# MVP

Persigue el mismo objetivo de separación de responsabilidades

La principal diferencia es quién controla los *inputs* del usuario


Variaciones

## Passive View

La vista no tiene conocimiento del modelo.
El Presentador actualiza la vista reflejando cambios en el modelo

## Supervising Controller

La vista interactúa con el modelo.
*<click!> → "Save"*
El Presentador actualiza el modelo y manipula la vista sólo si es necesario.

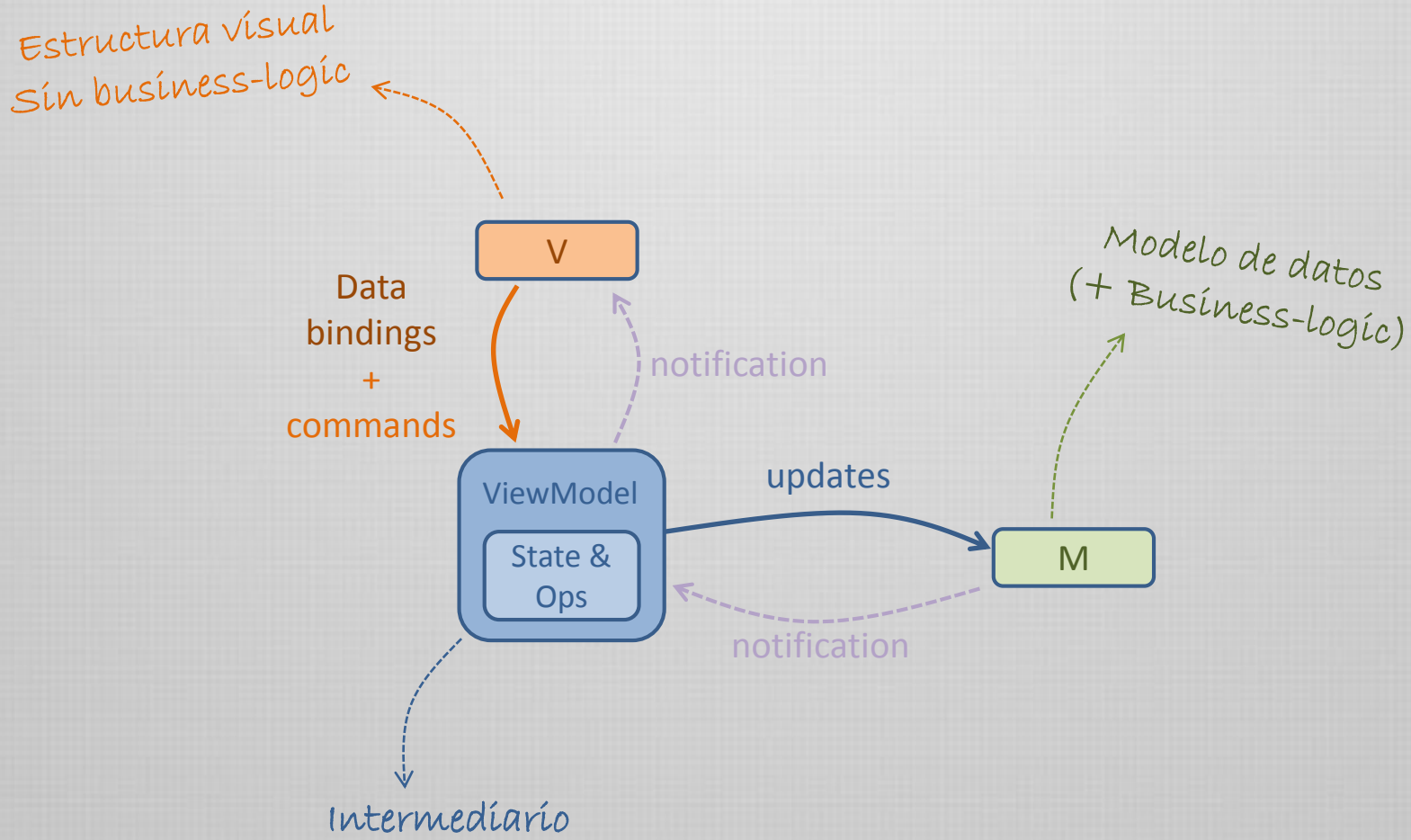
# MVP vs MVC

| MVC | MVP |
| --- | --- |
| Separación de responsabilidades | Separación de responsabilidades |
| El controlador maneja los inputs del usuario y gestiona el modelo | La vista maneja los inputs del usuario e invoca al presentador si es necesario |
| La vista puede consultar al modelo directamente | La vista es completamente pasiva |
| No favorece *unit testing* | Favorece *unit testing* |

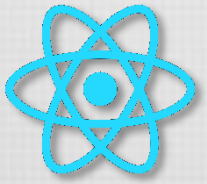
Es un patrón más apropiado para aplicaciones *pesadas* del lado cliente

Preferido en aplicaciones con mucha interacción del usuario en la vista

# MVMM

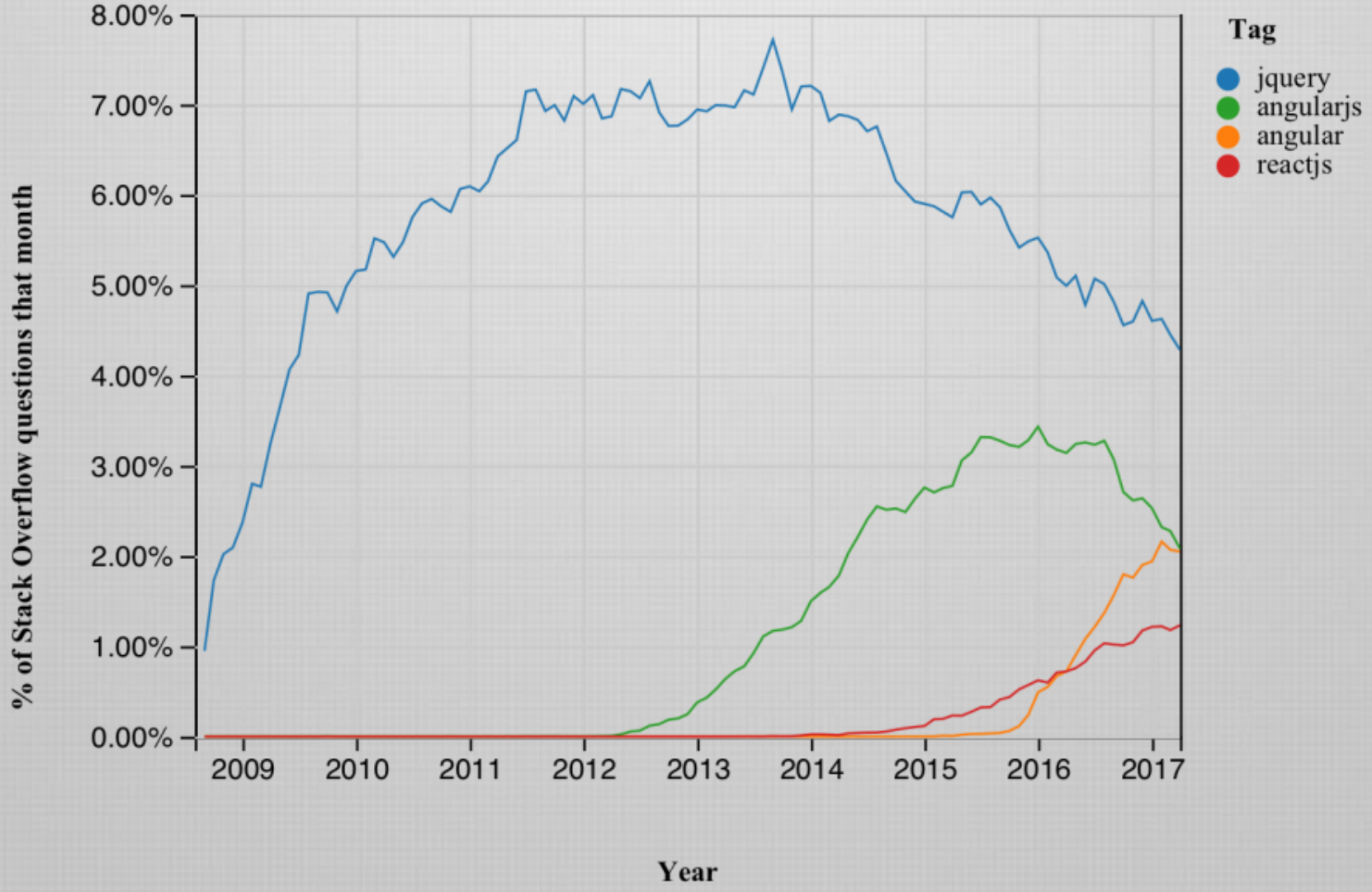

Estructura visual
Sin business-logic
V
Data
bindings
+
commands
notification
Modelo de datos
(+ Business-logic)
ViewModel
State &
Ops
updates
M
notification
Intermediario
Recupera datos del modelo y los ofrece a la vista
La comunicación bidireccional se da por medio de
notificaciones (PropertyChanged)

# Client-side Frameworks

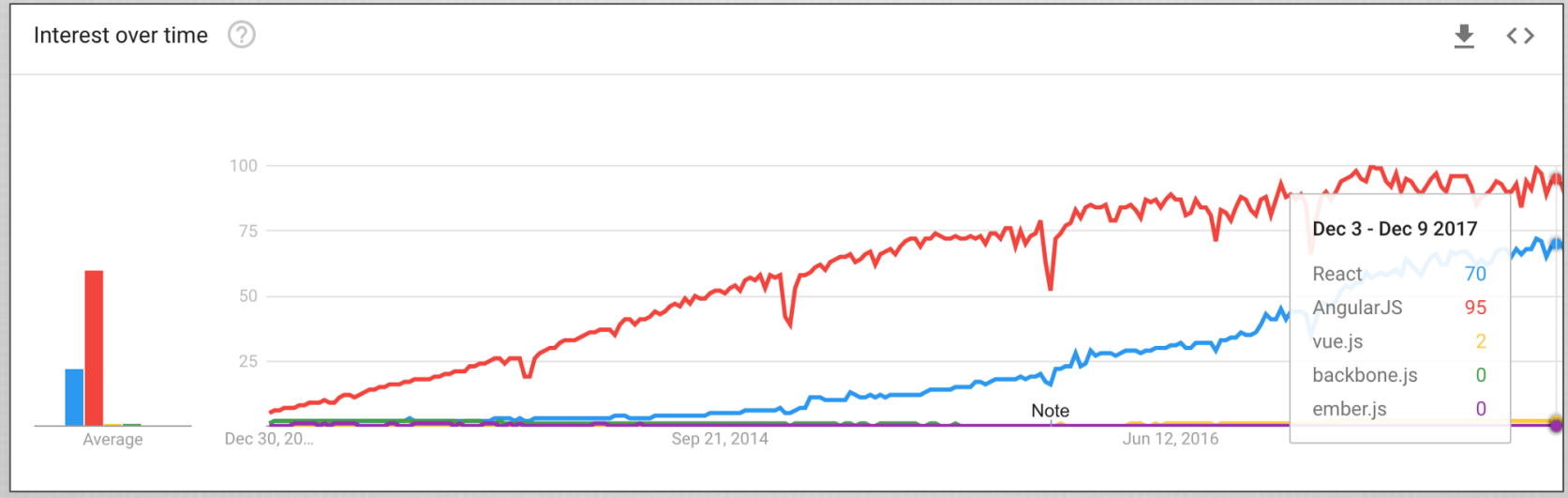



Knockout.

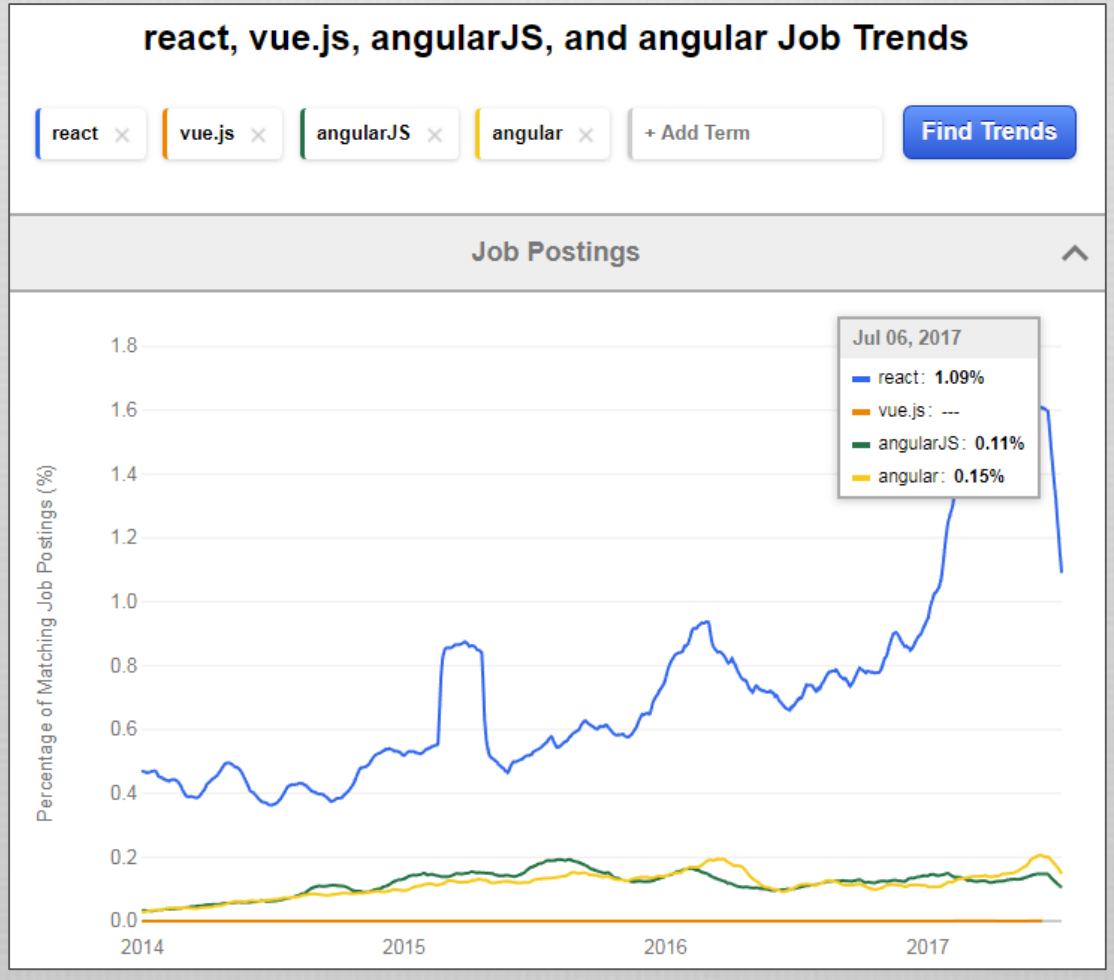

# Client-side Frameworks



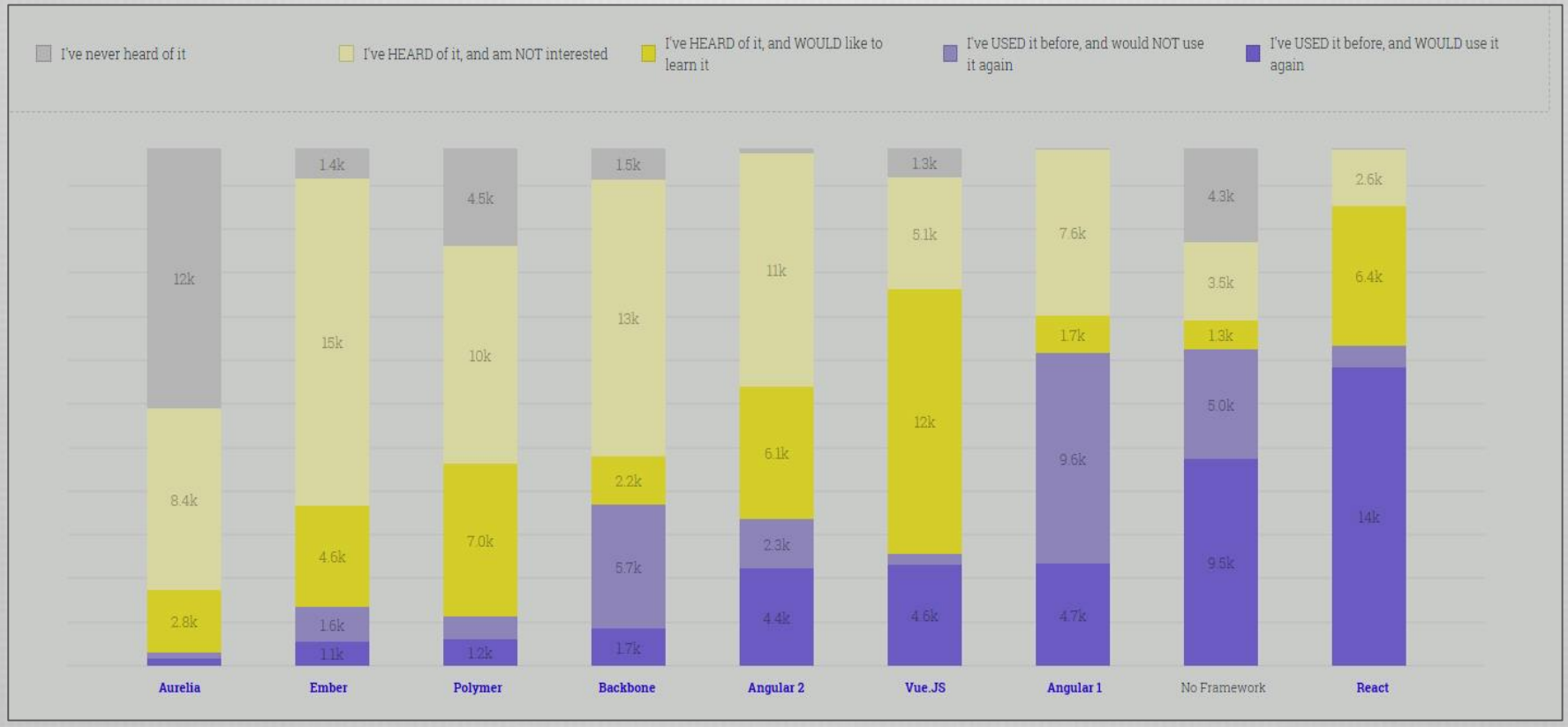
# Client-side Frameworks



# Client-side Frameworks



# Client-side Frameworks



# Angular.JS


Extiende la sintaxis de HTML para expresar componentes

Separación de la manipulación del DOM de la lógica de la application

importancia(app testing) = importancia(app writing)

Separación client-side / server-side

Acompañamiento integral: *diseño de UI, codificación de business-logic, testing*

"common trasks trivials, difficult tasks possible"

Inicializa y nombra la aplicación

Declara y crea el controlador

```
<div ng-app="invoice1" ng-controller="InvoiceController as invoice">
  <b>Invoice:</b>
  <div>
    Quantity: <input type="number" min="0" ng-model="invoice.qty" >
  </div>
  <div>
    Costs: <input type="number" min="0" ng-model="invoice.cost">
  </div>
  <div>
    <b>Total:</b> {{ invoice.total() | currency }}
  </div>
      <button class="btn" ng-click="invoice.pay()">Pay</button>
</div>
```

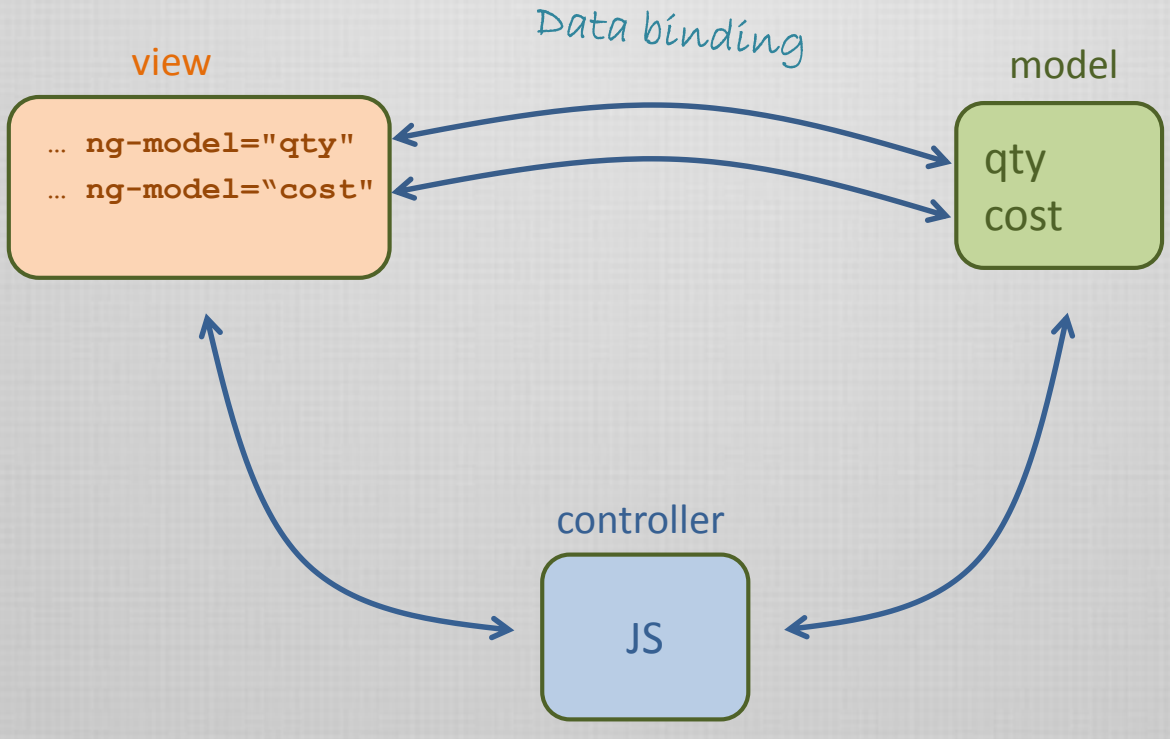
Invocación al controlador

Controller.js

```
angular.module('invoice1', [])
.controller('InvoiceController', function() {
  this.qty = 1;
  this.cost = 2;

  this.total = function total() {
    return (this.qty * this.cost);
  };


  this.pay = function pay() {
    window.alert("Thanks!");
  };
});
```


Data binding
view
model
... ng-model="qty"
... ng-model="cost"
qty
cost
controller
JS

```
import React, { Component } from 'react';

class Saludo extends React.Component {
  render() {
    return (
      <div className=misaludo">
        <h1>Hola {this.props.name}!</h1>
      </div>
    );
  }
}

export default Saludo;
```

```
import Saludo from ".\App";

ReactDOM.render(
  <Saludo name="Homero" />,document.getElementById('root')
);
```

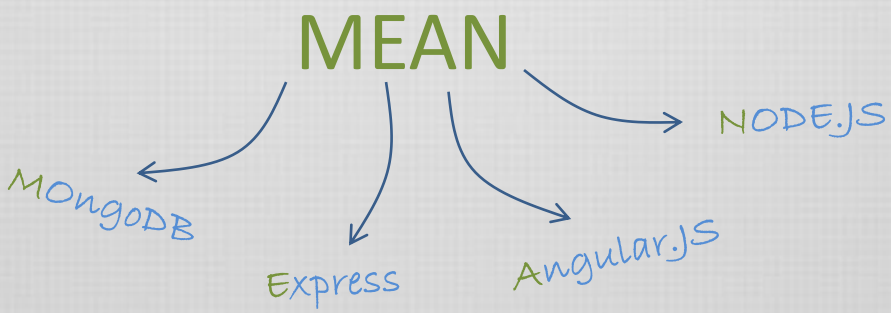
# React

```
<div id="miapp">
  {{ message }}
</div>
```

```
var app = new Vue({
  el: '#miapp',
  data: {
    message: 'Hello Vue!'
  }
})
```

# Solution Stacks

Angular se usa con frecuencia en un paquete de soluciones (stack) denominado MEAN





JavaScript

100% JavaScript

# Balance cliente - servidor