

Tema 5: Frameworks JS en el cliente

Parte I: Introducción

Tema 5: Frameworks JS en el cliente

5.1

¿Por qué *frameworks* en el cliente?

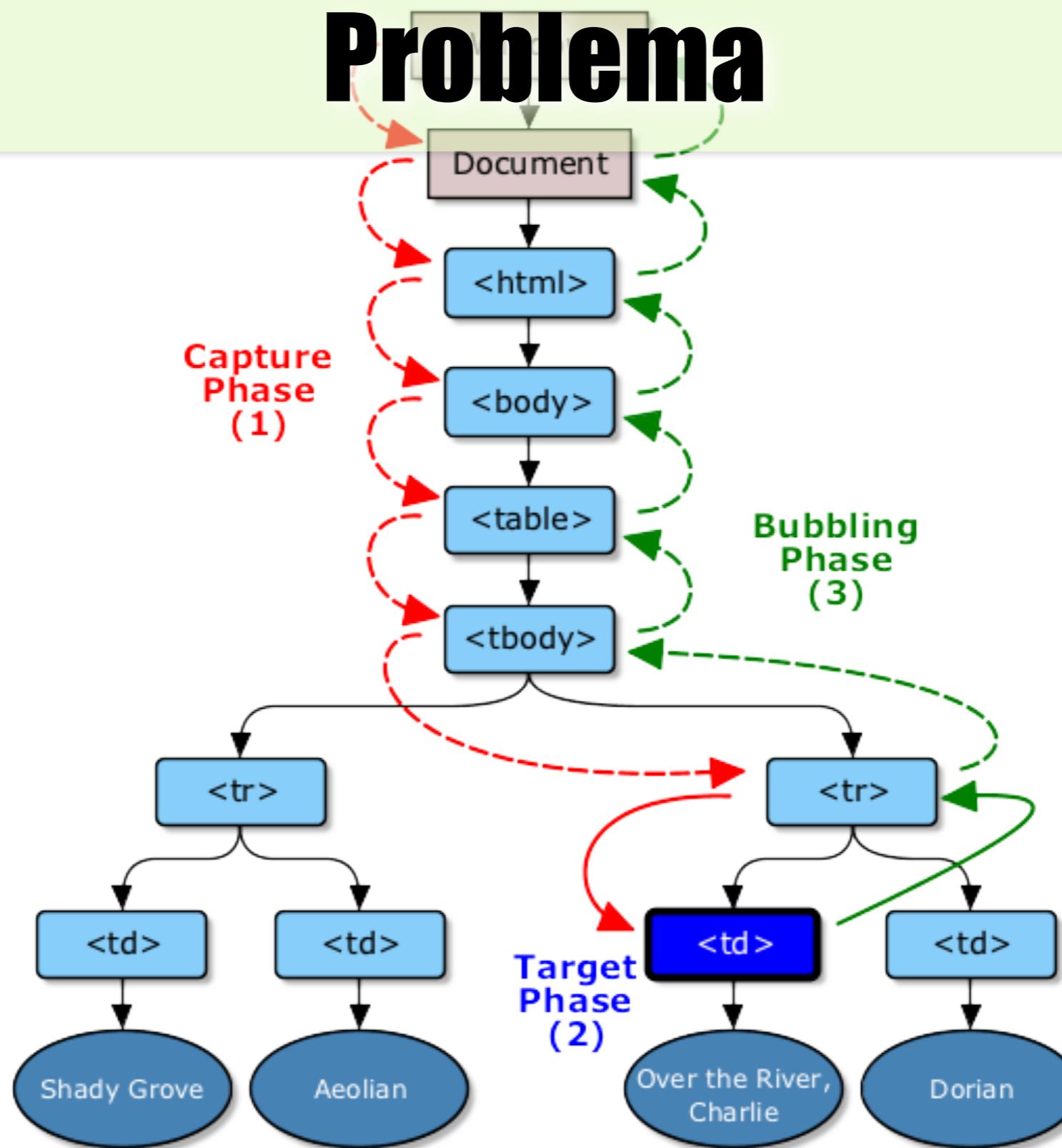
APIs estándar vs. frameworks

- Hasta ahora, en la parte de teoría hemos visto lo que se puede hacer con los APIs nativos del navegador
- Muchas **funcionalidades**
 - **Comunicación con el servidor:** fetch API
 - **Manipulación dinámica del HTML y del CSS:** DOM
 - **Almacenamiento local:** local storage, IndexedDB, ...
- ¿Por qué muchos desarrolladores **no los consideran suficientes?**. Hay razones de distintos tipos...

Volvamos por un momento a mediados de los 2000



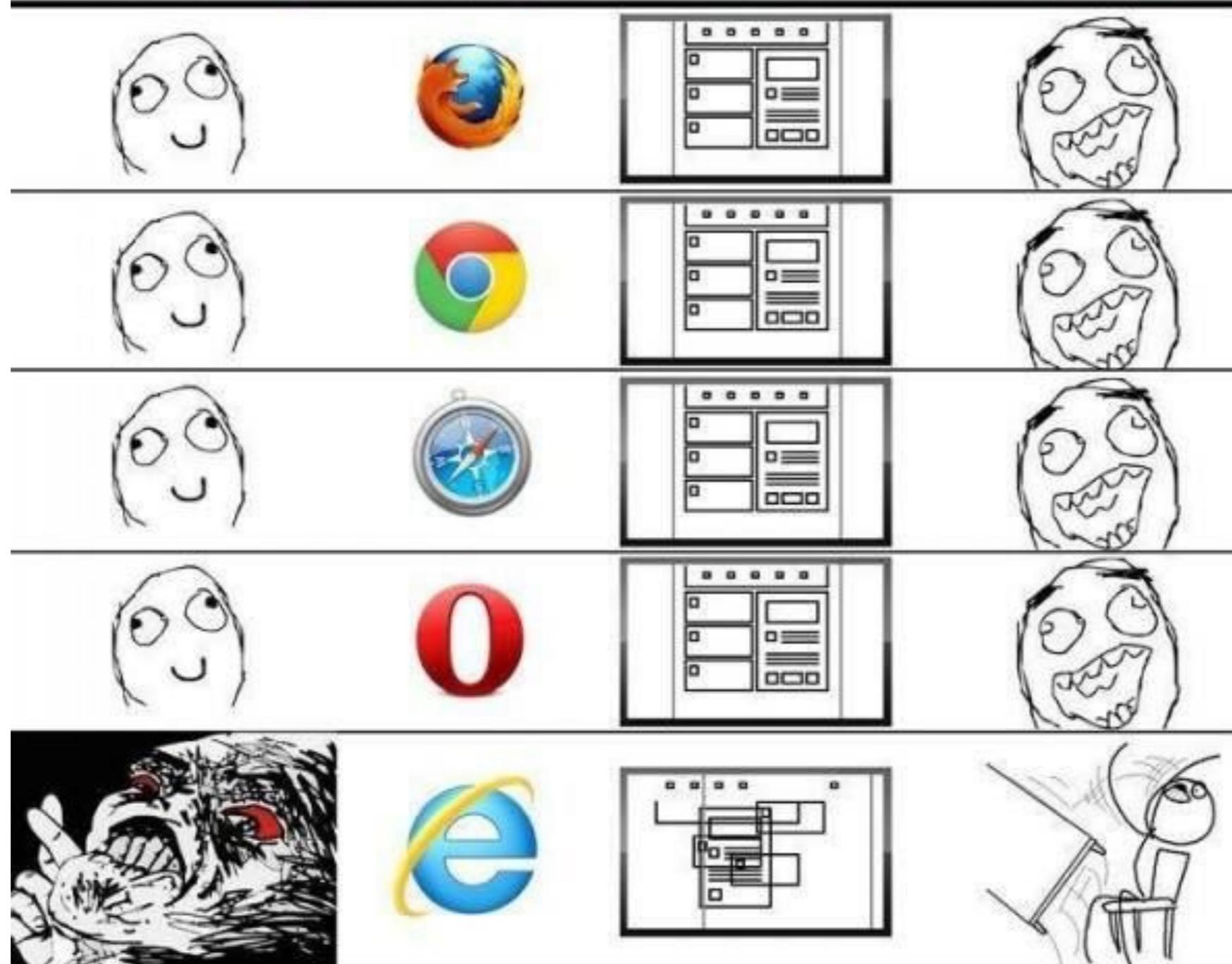
Problema



APIs web tediosos y complejos

Problema

Testing web pages on different browsers



We Know Meme

Incompatibilidades entre navegadores

2006



La sensación: jQuery

La era de las librerías y *toolkits* en el *frontend*



- Ofrecían un API más sencillo de usar y sin embargo más potente que el estándar
- Ofrecían elementos que no tenía el estándar (como widgets)
- Proporcionaban compatibilidad entre navegadores

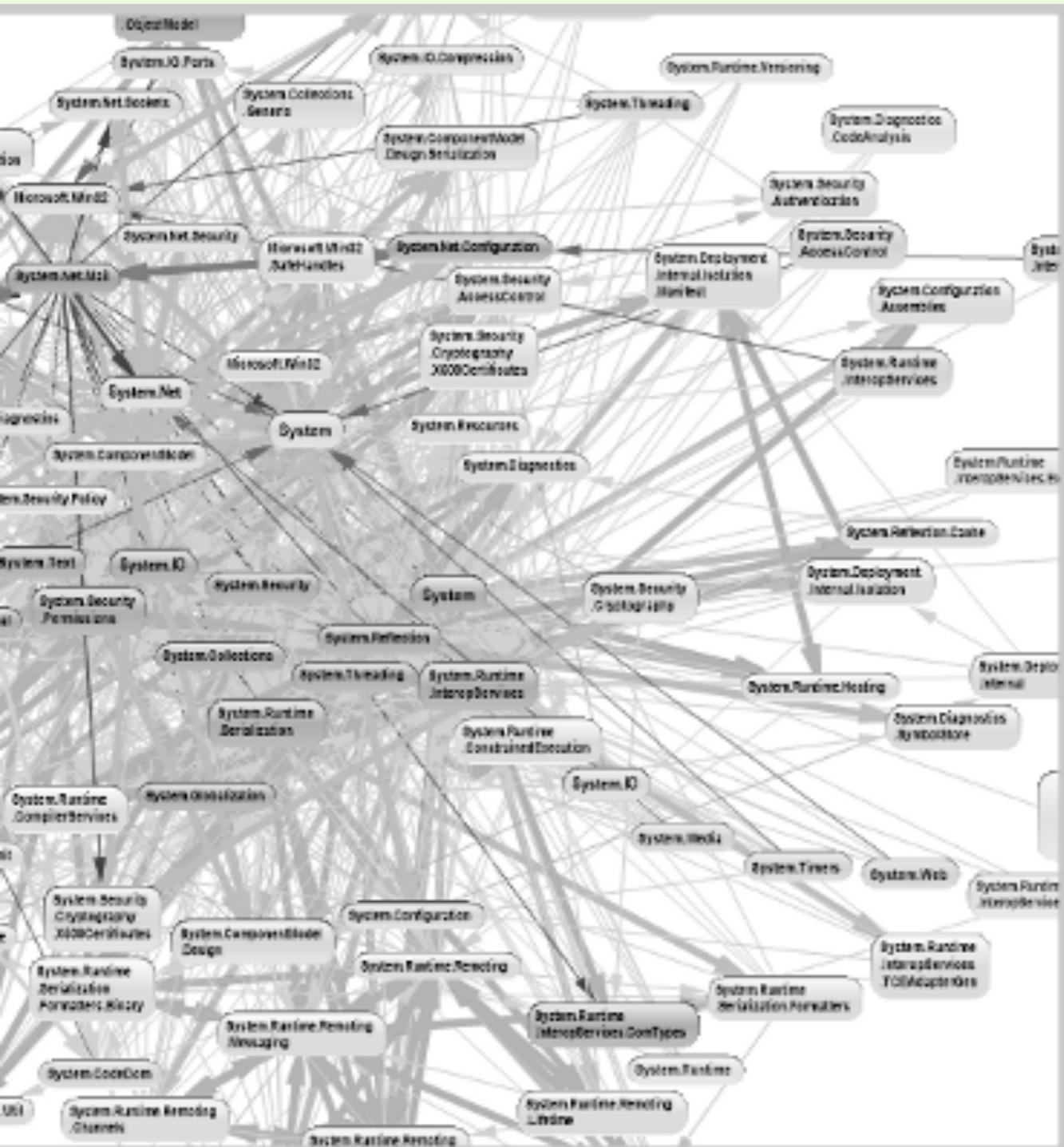
Modernización de los APIs estándar

En los últimos años se han modernizado/simplificado/mejorado muchos APIs estándar

- `fetch` vs `XMLHttpRequest`
- `document.querySelector()` está claramente “inspirado” en jQuery
- `innerHTML` vs. la forma de trabajar nativa del DOM 1



Problema



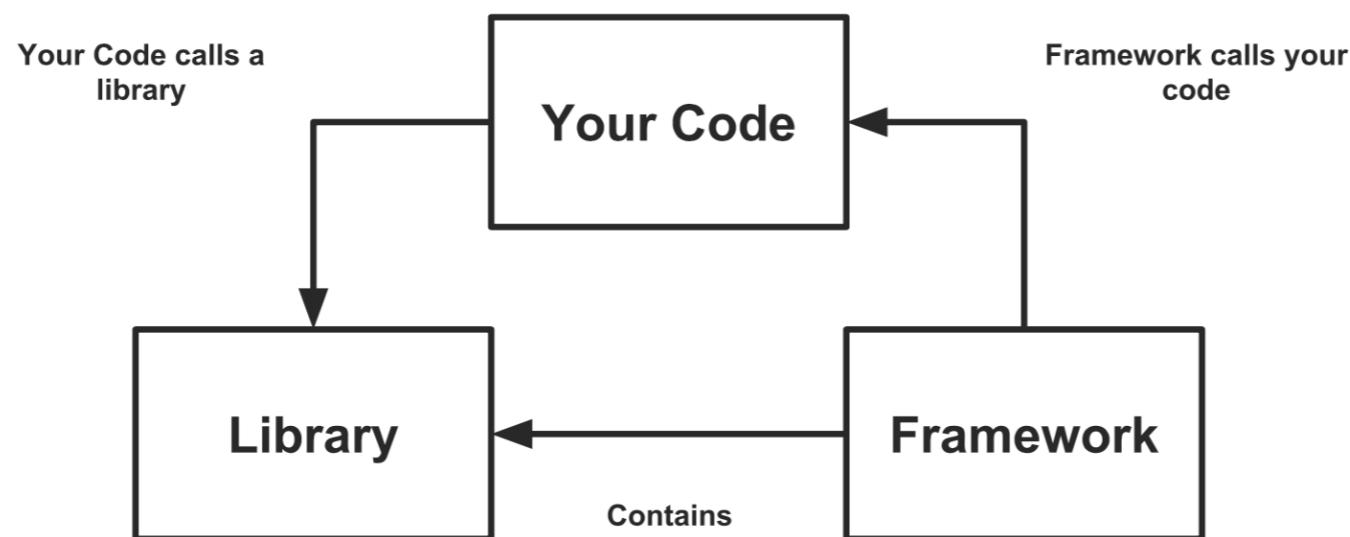
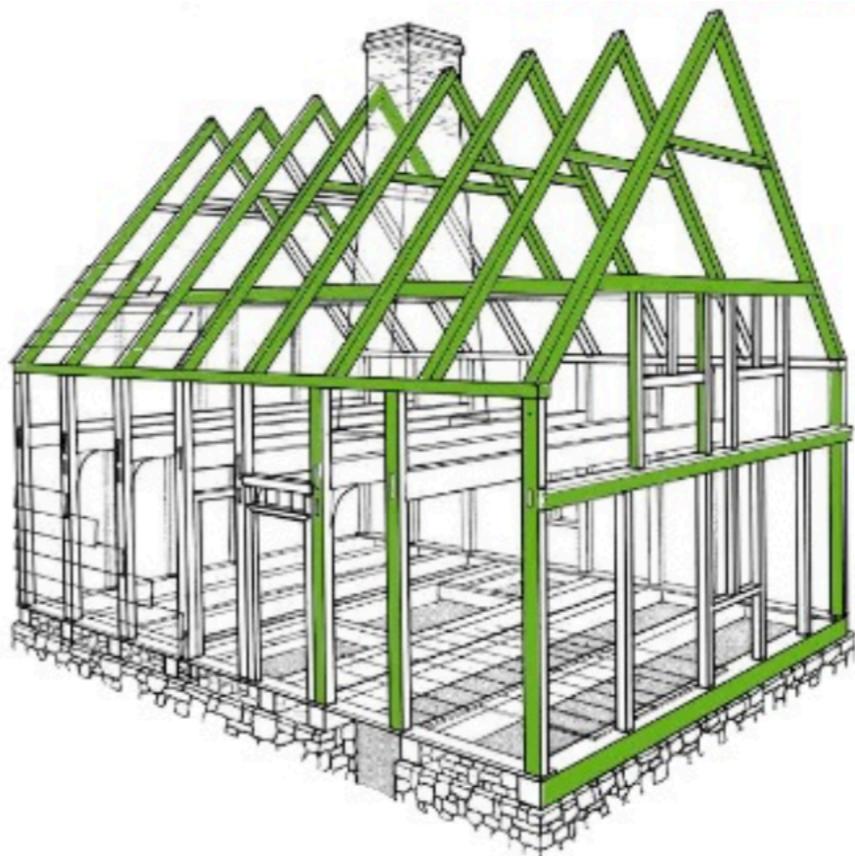
Evitar el “big ball of mud”

Código poco estructurado

Que mezcla gestión de interfaz y lógica

```
auth.onAuthStateChanged(async function(user) {
  if (user!=null) {
    //ocultar login, mostrar lista
    document.getElementById('login-form').style.display = "none"
    document.getElementById('lista-widget').style.display = "block"
    //listar items
    await listarItems()
  }
  else {
    //nadie autentificado
    document.getElementById('login-form').style.display = "block"
    document.getElementById('lista-widget').style.display = "none"
  }
})
```

Los frameworks nos dan
**una guía de cómo
estructurar y organizar
nuestro código**



Del libro Frameworkless Frontend Development, Francesco Strazzullo

Problema

The screenshot shows a Gmail inbox with the following layout:

- Compose** button (red)
- Inbox (7)** link
- Starred**, **Drafts**, **Sent Mail** links
- Search people...** input field
- Primary** tab selected
- Social** tab (3 new): Google+, YouTube, Emi...
- Promotions** tab (2 new): Google Offers, Zagat
- Updates** tab (2 new): Shoehop, Blitz Air
- More** button
- Gmail** dropdown menu
- 1-100 of 2,067** link

The inbox list contains the following items:

- Google+**: You were tagged in 3 photos on Google+ - Google+ You were tagged in three photos in an album titled [REDACTED]
- YouTube**: LauraBlack just uploaded a video. - Jess, have you seen the video LauraBlack uploaded...
- Emily Million (Google+)**: [Knitting Club] Are we knitting tonight? - [Knitting Club] Are we knitting tonight?
- Sean Smith (Google+)**: Photos of the new pup - Sean Smith shared an album with you. View album be thoughtful about who
- Google+**: Kate Baynham shared a post with you - Follow and share with Kate by adding her to a circle. Don't know
- Google+**: Danielle Hoodhood added you on Google+ - Follow and share with Danielle by adding her to a circle.
- YouTube**: Just for You From YouTube: Daily Update - Jun 19, 2013 - Check out the latest videos from your char
- Google+**: You were tagged in 3 photos on Google+ - Google+ You were tagged in three photos in an album titled [REDACTED]
- Hilary Jacobs (Google+)**: Check out photos of my new apt - Hilary Jacobs shared an album with you. View album be thoughtful
- Google+**: Kate Baynham added you on Google+ - Follow and share with Kate by adding her to a circle. Don't know

Desarrollo de Single Page Apps

¿Qué necesitamos para implementar una SPA?

- Código modular y bien estructurado
- Gestionar de forma lo más sencilla posible la sincronización entre el HTML y los datos
- Comunicación sencilla con APIs externos
- Navegación entre páginas/componentes de la app
-

A blurry, high-contrast image of a person's face and hands holding a smartphone. The person appears to be wearing a dark cap and a light-colored shirt. The phone screen displays a large, stylized, multi-layered geometric logo, possibly a hexagon or a similar shape, with white outlines against a dark background.

2010

La sensación: Frameworks MVC

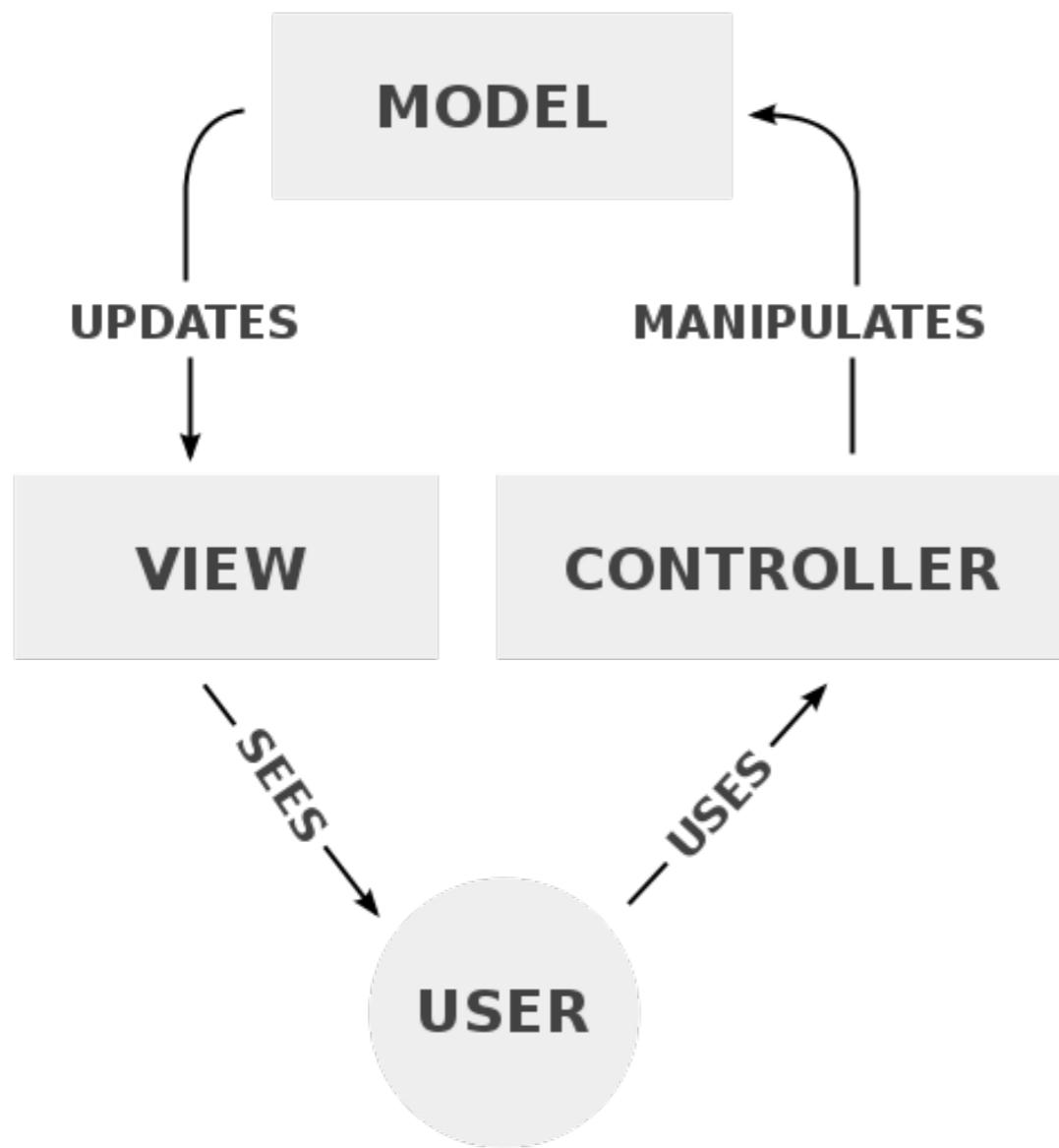
Tema 3: Frameworks JS en el cliente

5.2

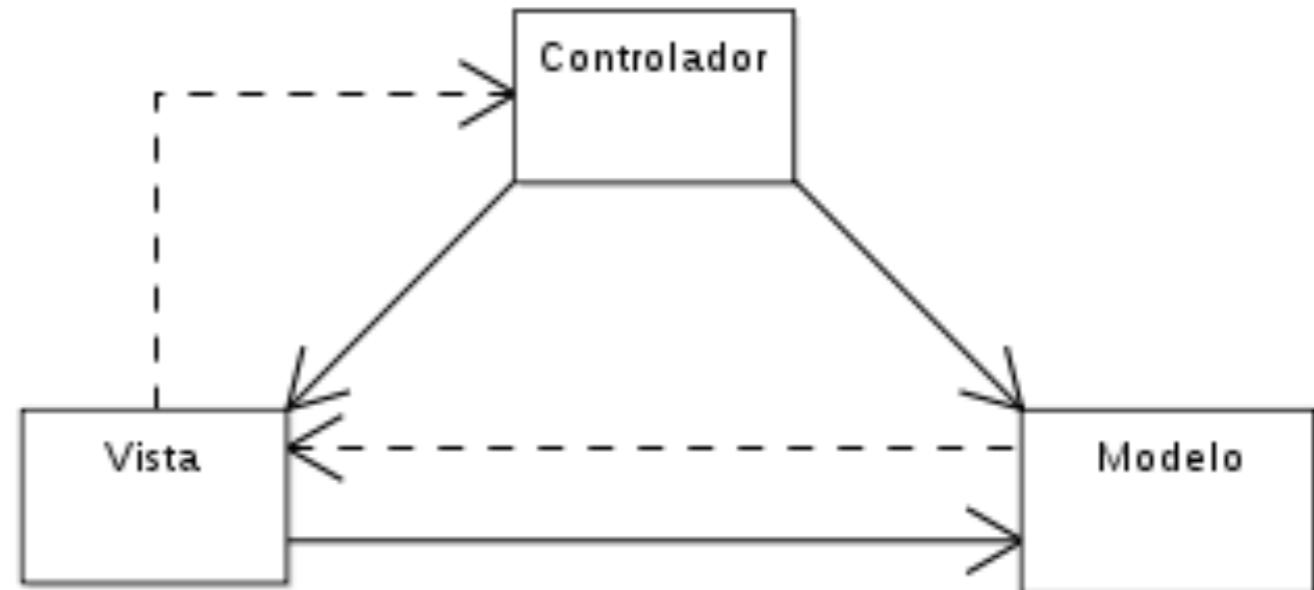
Frameworks MVC (auge y caída)

MVC

Patrón de diseño arquitectónico que **separa el modelo** del dominio **de su representación** en la interfaz (**vista**)



<http://en.wikipedia.org/wiki/Model-view-controller>



<http://es.wikipedia.org/wiki/Modelo-vista-controlador>

MVC en el servidor web

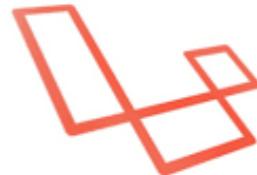
MVC lleva siendo usado con éxito en aplicaciones web **en el lado del servidor** desde hace +20 años



Symfony



Struts



laravel



Si MV* funciona tan bien en el servidor...¿por qué no llevarlo también al **cliente**?

2010-2015: La explosión de los frameworks MVC

Spine



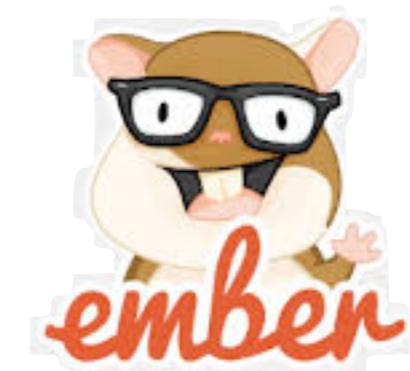
Maria

Knockout.



cujoJS

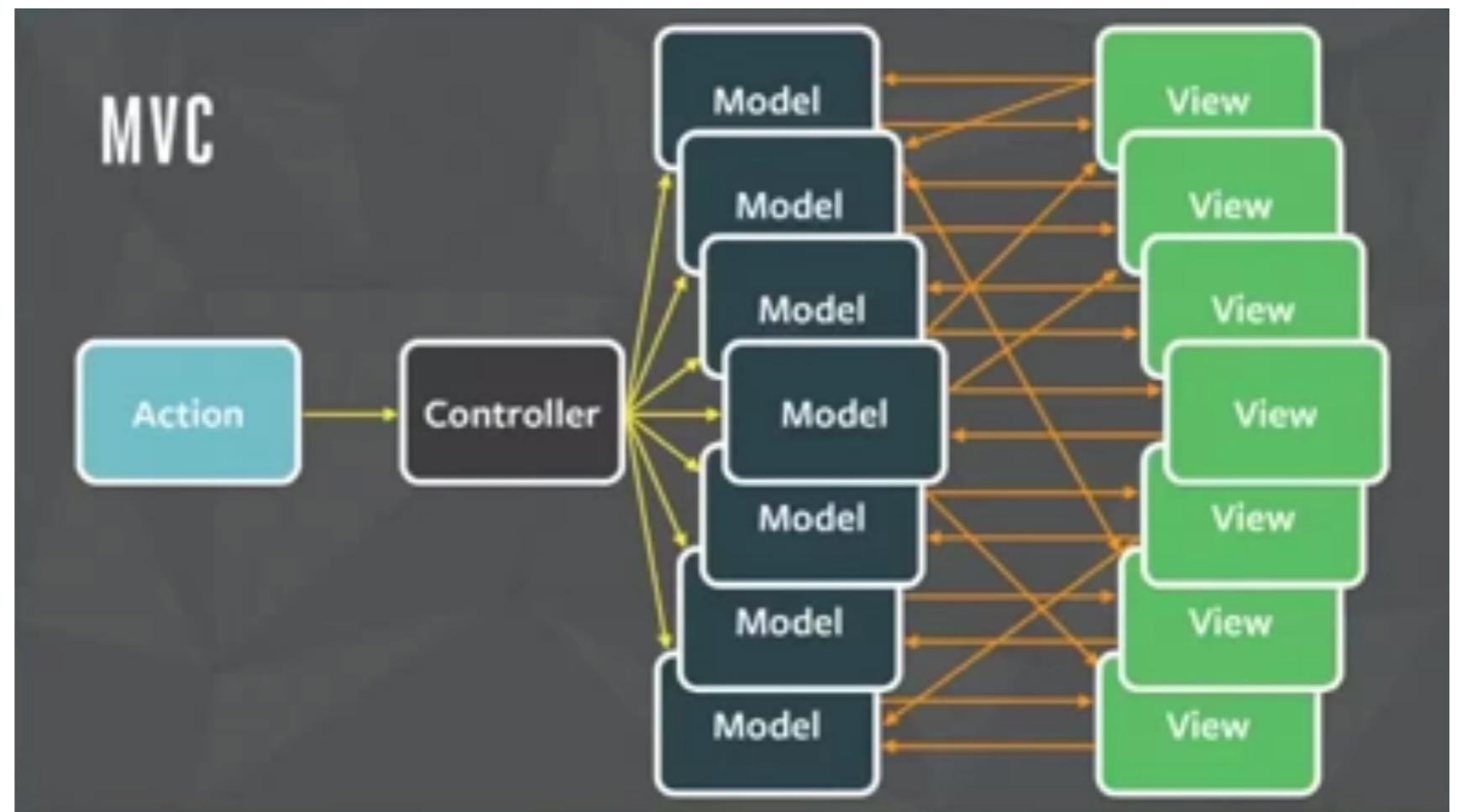
canjs



 **AGILITY.JS**

¿Por qué “cayó” MVC en el frontend?

- Nunca estuvo muy claro cómo resolver la parte “C”
 - ¿Cuál es el papel exacto del controlador?
 - ¿Dónde colocamos la lógica de negocio? ¿Y la conexión con APIs externos?
- Tampoco estaba claro cuántos M, V y C debería tener una página
- Escalabilidad
- ...¿Moda?



A blurry, low-light photograph of a person's hands and face. The person is wearing a virtual reality headset and holding a VR controller. The background is dark and out of focus.

2015

La sensación: Componentes

Tema 3: Frameworks JS en el cliente

5.3

*Frameworks basados
en componentes*

React popularizó varias ideas, entre ellas la de **Componentes**
(aunque ni siquiera lo ponían en su web como algo diferenciado)

React

A JAVASCRIPT LIBRARY FOR BUILDING USER INTERFACES

Get Started

Download React v0.12.1

JUST THE UI

Lots of people use React as the V in MVC. Since React makes no assumptions about the rest of your technology stack, it's easy to try it out on a small feature in an existing project.

VIRTUAL DOM

React uses a *virtual DOM* diff implementation for ultra-high performance. It can also render on the server using Node.js — no heavy browser DOM required.

DATA FLOW

React implements one-way reactive data flow which reduces boilerplate and is easier to reason about than traditional data binding.

Conceptos comunes

Ideas presentes en la actualidad en la **mayoría** de frameworks, aunque siempre hay variaciones

- **Componentes** como unidad básica de abstracción
- App como **jerarquía** de componentes
- “**Data Down, Actions Up**”
- Gestión **centralizada** del **estado**
- **Routing**: asociar URLs a componentes
- **Reactividad**

Gestión del estado

Gestionar el estado de la app de manera efectiva, organizada y con código sencillo de depurar

- Datos que estamos mostrando en pantalla
- Datos que guardamos en local
- Estado de la interfaz (en qué pantalla estamos, qué cosas están plegadas/desplegadas)

Lo veremos más adelante

Routing

La web está ***basada en las URLs***. Queremos mapear la URL que ve el usuario con los componentes que ve pintados



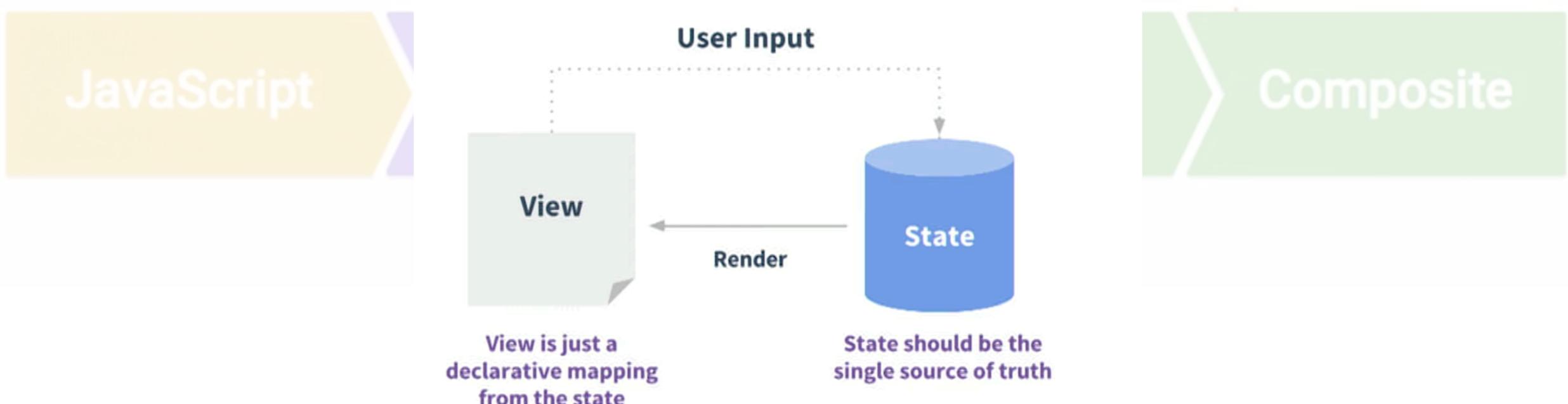
REACT ROUTER INTRODUCTION

```
<Switch>
  <Route
    path="/about"
    render={props => <About {...props} admin="Bean" />}
  />
  <Route path='/:user' component={User} />
  <Route component={NoMatch} />
</Switch>
```

Reactividad

Modificar el HTML automáticamente para **reflejar el estado actual** de la app de forma **sencilla para el desarrollador** y además **eficiente**

Vincular variables JS con fragmentos de HTML, o unas variables con otras





🔗 FRONT-END FRAMEWORKS RATIOS OVER TIME

Introducción

T-shirt

Demografía

Features

Librerías

> Front-end frameworks

Ratios Over Time

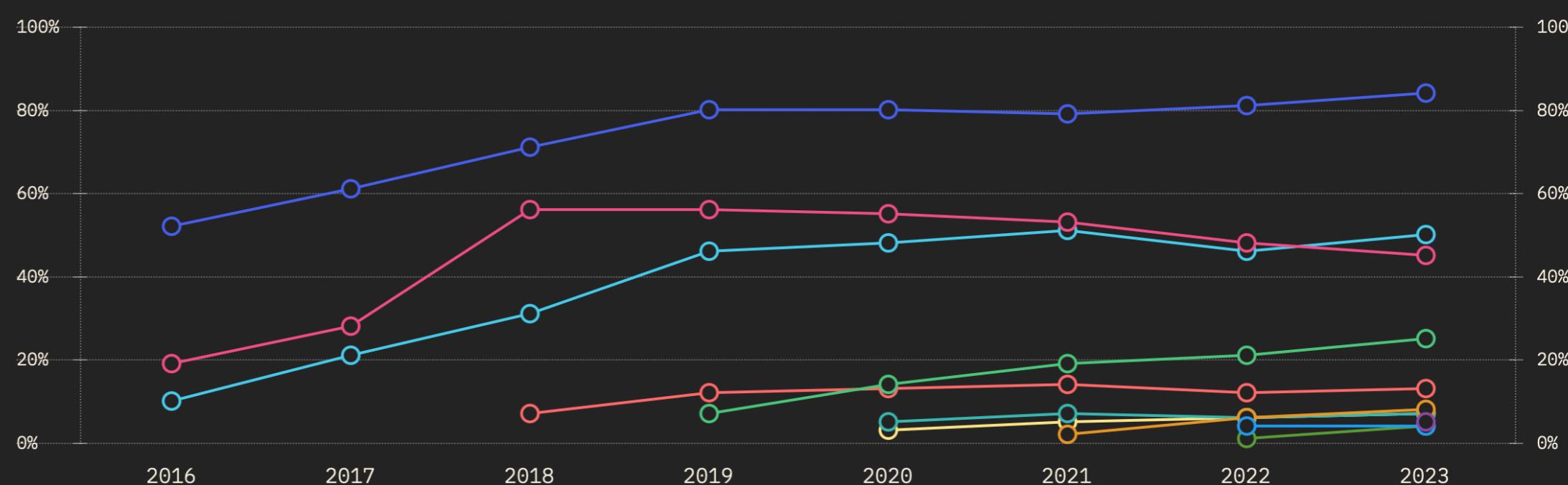
Todos los que han respondido

Add Filters...

React Vue.js Angular Preact Svelte Alpine.js Lit
Solid Qwik Stencil HTMX

Mode: Value Rank

View: Usage Awareness Interest Retention Positivity



“The State of JS”, una encuesta anual hecha a desarrolladores de todo el mundo donde se recogen las tendencias de uso de herramientas, librerías, frameworks, etc del mundo JS

<https://2023.stateofjs.com/es-ES/libraries/front-end-frameworks/>

Componente

- **vista** + manejo de **eventos** + **lógica/estado** propio
- Además de todo esto, es probable que tengamos un manejo de eventos + lógica/estado “**globales**”
- Curiosamente, por su naturaleza un componente **mezclará HTML,CSS** (vista) + **JS** (lógica/estado), lo que hasta entonces se había considerado mala práctica

```
<script>
  let count = 0;

  function increment() {
    count += 1;
  }
</script>

<h1>Contador: {count}</h1>
<button on:click={increment}>
  Incrementar
</button>

<style>
  button {
    margin: 5px;
    padding: 10px;
    font-size: 1rem;
  }
</style>
```

Un componente en Svelte

Componente en Angular

```
@Component({
  selector: 'app-tarjeta-visita',
  templateUrl: './tarjeta-visita.component.html',
  styleUrls: ['./tarjeta-visita.component.css'],
})
export class TarjetaVisitaComponent {

  nombre = ""
  apellidos = ""
  puesto = ""

  constructor(private puestoService : PuestoService ) {}

  get nombreCompleto() {
    return this.nombre + " " + this.apellidos
  }

  generarPuesto() {
    this.puestoService.generarPuesto().subscribe(datos=> this.puesto = datos)
  }
}
```

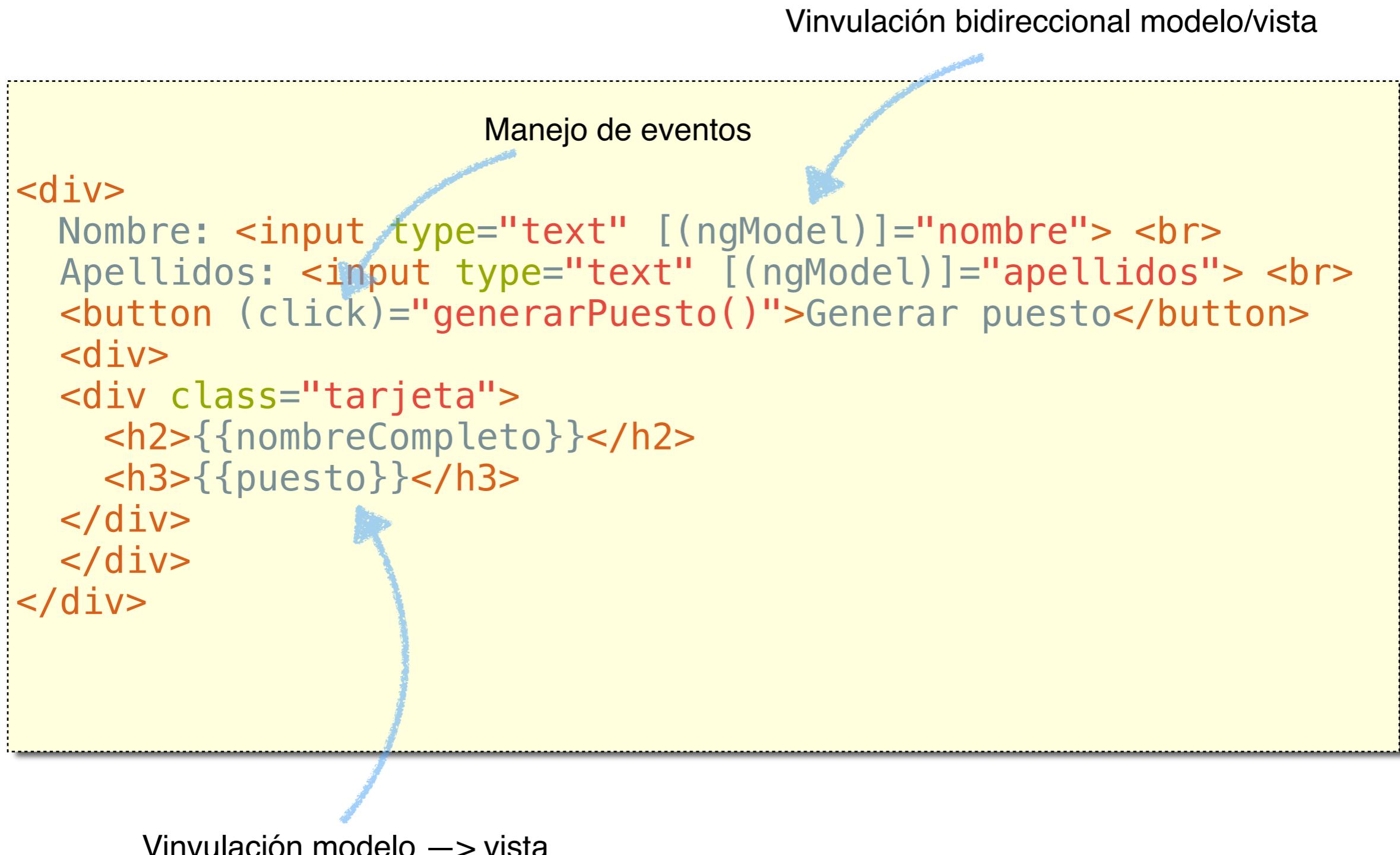
Vista: HTML+CSS

Estado

Lógica

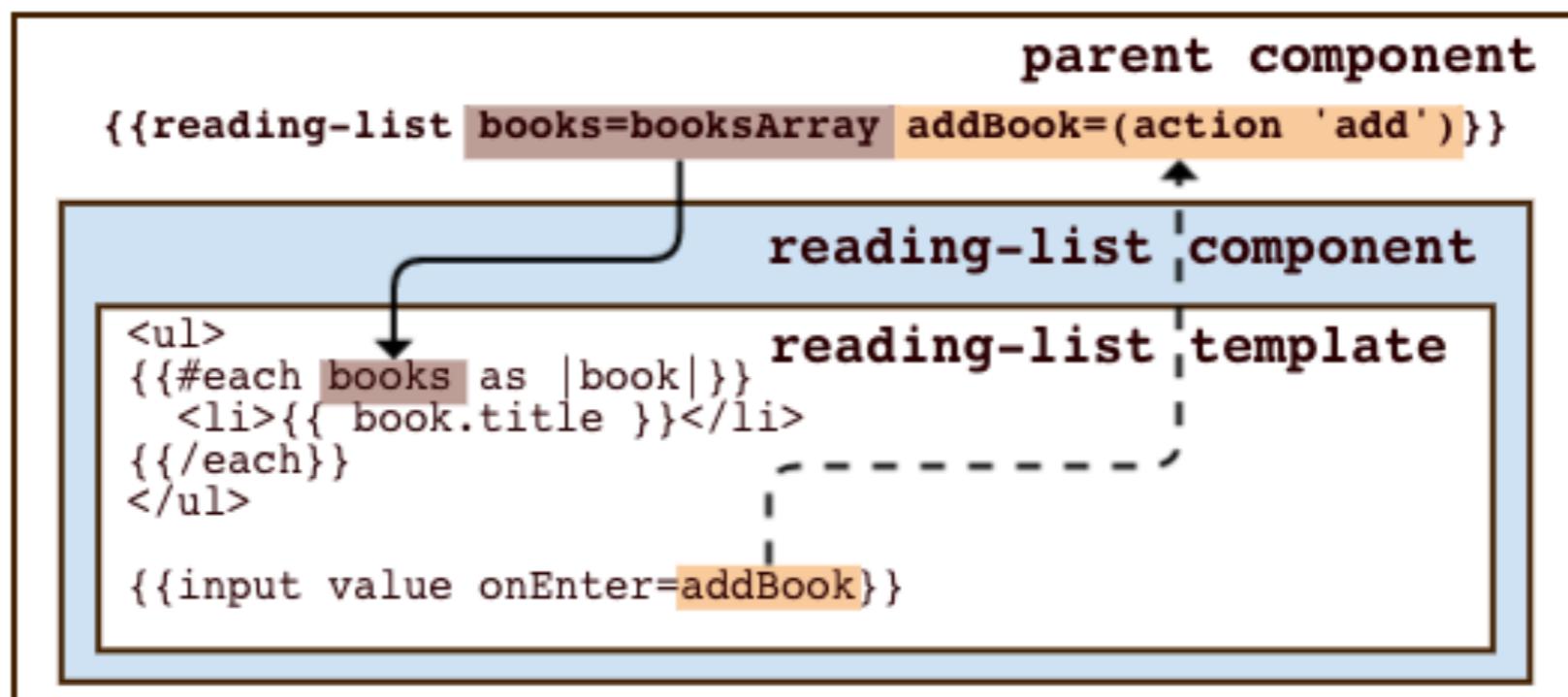
<https://stackblitz.com/edit/angular-zwu7ad>

Template del componente



“Data Down, Actions Up”

- Los **datos** pasan de componentes padres a hijos (**down**) y las **acciones** suben (**up**) hasta el componente capaz de procesarlas
- La frase del título es del framework **Ember** (<https://www.emberjs.com/>) pero también es así en **Angular, Vue, React...**
- En Angular/Vue las acciones se modelan como eventos que suben, en React se pasa una función que luego llama el hijo



De un tutorial de cómo hacer una app de una lista de libros pendientes en Ember siguiendo la idea de Data Down Actions Up

Data down

- Los componentes padres pasan a los hijos los datos necesarios para el *rendering*

ListaCompra

Lista

Item

Item

Item

```
export class ListaCompra {  
  items = [  
    {nombre: 'Leche', comprado: true },  
    {nombre: 'Pan', comprado: false },  
    {nombre: 'Huevos', comprado: false }  
]
```

```
<h2>  
  La lista de la compra  
</h2>  
<lista [items] = "items" />
```

```
export class Lista {  
  @Input() items;  
  ...
```

```
<ul *ngFor="let item of items;  
  index as i">  
  <li> <item [datos] = "item" </li>  
</ul>
```

```
export class Item {  
  @Input() datos;  
  ...
```

```
<input type="checkbox"  
  [checked] = "datos.comprado" />  
<span [class.tachado] = "datos.comprado" >  
  {{datos.nombre}} </span>
```

<https://stackblitz.com/edit/stackblitz-starters-83ynxg>

(El código de gestión de eventos se ha eliminado para simplificar)

Actions up

```
export class ListaCompra implements OnInit {
  ...
  items: any[] = [
    { nombre: 'Leche', comprado: true },
    { nombre: 'Pan', comprado: false },
    { nombre: 'Huevos', comprado: false }
  ];
  toggleComprado(index: number) {
    this.items[index].comprado = !this.items[index].comprado;
    console.log(this.items);
  }
}
```

⑦ Como ya estamos en el componente “dueño” de los datos, el evento ya no sigue subiendo sino que se procesa, cambiando el estado del componente. **El “Data Down” vuelve a comenzar**

```
<div>
  <h2>La lista de la compra</h2>
  <lista [items]="items"
    (toggleComprado)="toggleComprado($event)"></div>
```

⑥ El evento lo captura el nivel inmediatamente superior.

```
export class Lista implements OnInit {
  @Input() items!: any[];
  @Output() toggleComprado = new EventEmitter<number>()
```

⑤ se genera otro evento llamado también “toggleComprado”, ahora con un parámetro

```
<ul *ngFor="let item of items; index as i">
  <li>
    <item [datos]="item"
      (toggleComprado)="toggleComprado.emit(i)">
    </li>
  </ul>
```

④ El evento lo captura el nivel inmediatamente superior.

① El usuario hace clic

```
export class Item implements OnInit {
  @Input() datos: any;
  @Output() toggleComprado = new EventEmitter<void>();
```

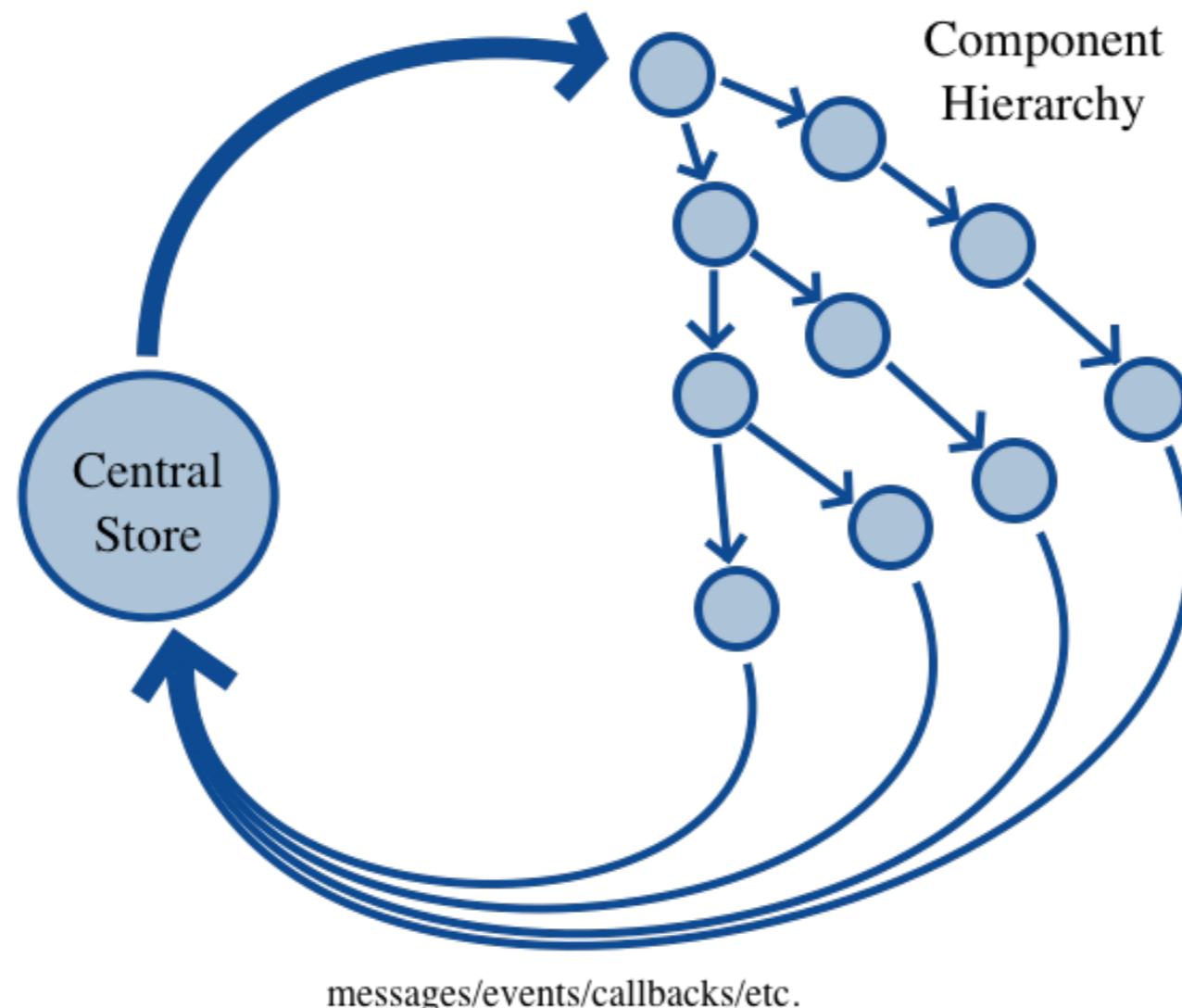
③ se genera un evento “toggleComprado” definido por el desarrollador, no es del navegador

```
<input type="checkbox"
  [checked]="datos.comprado"
  (change)="toggleComprado.emit()">
<span [class.tachado] = "datos.comprado">
  {{datos.nombre}}
</span>
```

② El manejador del evento “change” llama a un “emisor de eventos” del componente

Gestión centralizada del estado

- Ya hemos visto la filosofía de subir el estado a los componentes de nivel superior
- Ir un paso más allá: sacar el estado a un “almacén externo”



Web components

- Un **estándar** para desarrollar componentes en la web
- **Conjunto de tecnologías:** custom elements + shadow DOM + templates
- **Carecen de algunas funcionalidades** que sí suelen tener los frameworks como la reactividad o la gestión del estado

-

0

+

Ejemplo componente contador de [webcomponents.dev](#)



webcomponents.org: comunidad donde se comparten y documentan web components

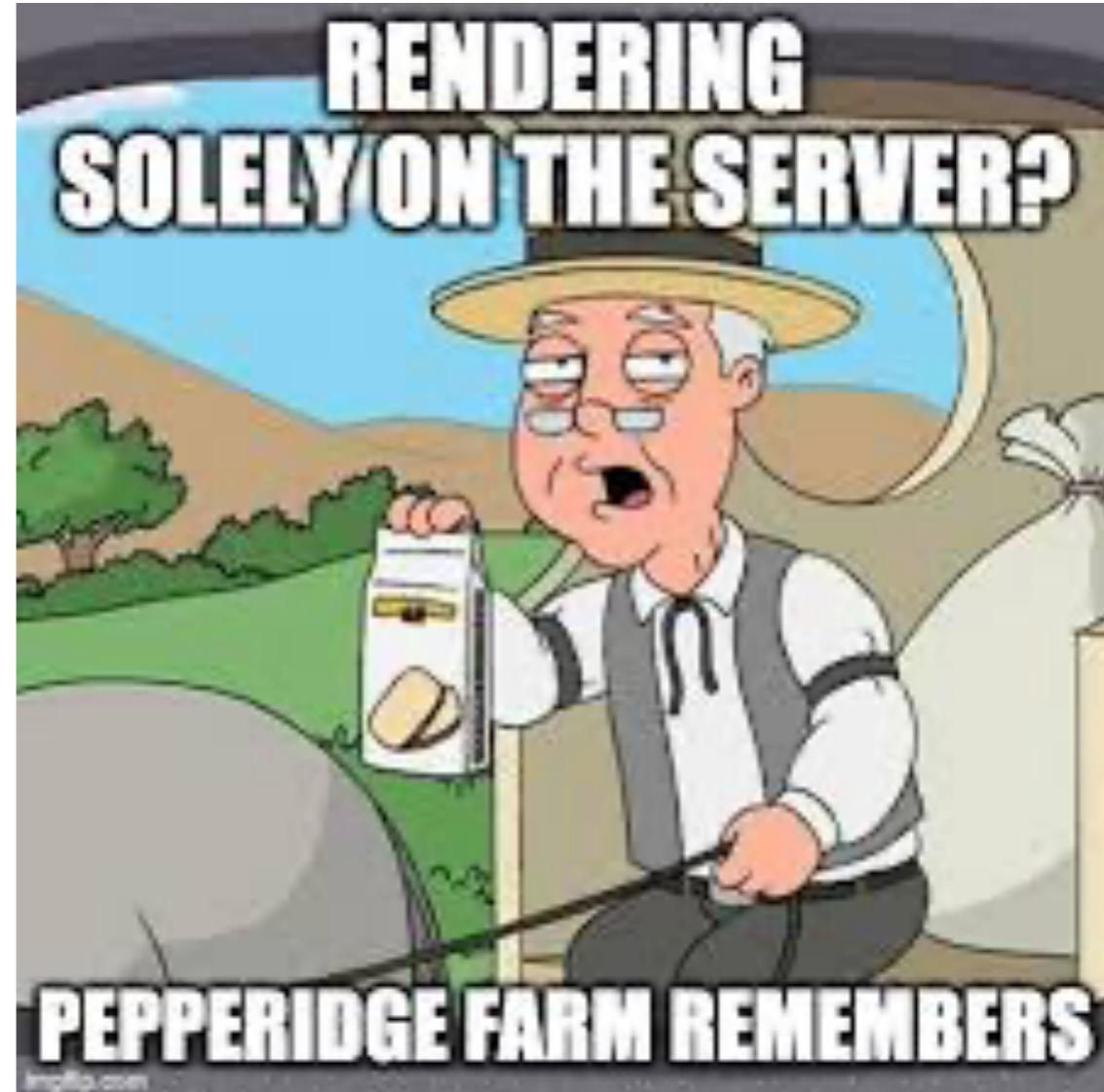
A blurry, colorful background image showing a close-up of a person's face and shoulder area. The person has dark hair and appears to be wearing a white shirt. The colors are somewhat muted and out of focus.

2020s

**La sensación
SSR (Server Side Rendering)**

Server Side Rendering

- Las SPAs tienen el problema de que el rendimiento no es siempre el mejor... depende del dispositivo cliente
- Si el servidor envía el HTML ya montado, la carga inicial será más rápida
- Esto se parece sospechosamente a la web “de toda la vida”, aunque como veremos no es una vuelta exactamente a lo mismo



Moralejas de hoy

- Hasta cierto punto todos los frameworks JS se basan en las mismas ideas
- Corolario: **no hay un framework mejor**, dependerá de la tarea para la que queramos dedicarlo, la experiencia de los desarrolladores, la escala del proyecto, el gusto personal,....
- Los *frameworks* JS ofrecen una serie de **ventajas de productividad y organización de código**, el **coste** a cambio es la curva de **aprendizaje**
- Si vais a dedicaros al *frontend* tened en cuenta que cada 5 años cambian los paradigmas...