



Pontificia Universidad Católica de Chile  
Escuela de Ingeniería  
Departamento de Ciencia de la Computación

# Client-side Frameworks

**IIC2513 - Tecnologías y Aplicaciones Web**

Sebastián Vicencio R.  
2do Semestre 2020

# Client-side Frameworks

Uso extendido de JavaScript  
en el cliente

# Client-side Frameworks

## UI aplicaciones web tradicionales



**Interacción del usuario**  
genera **nuevos requests HTTP**

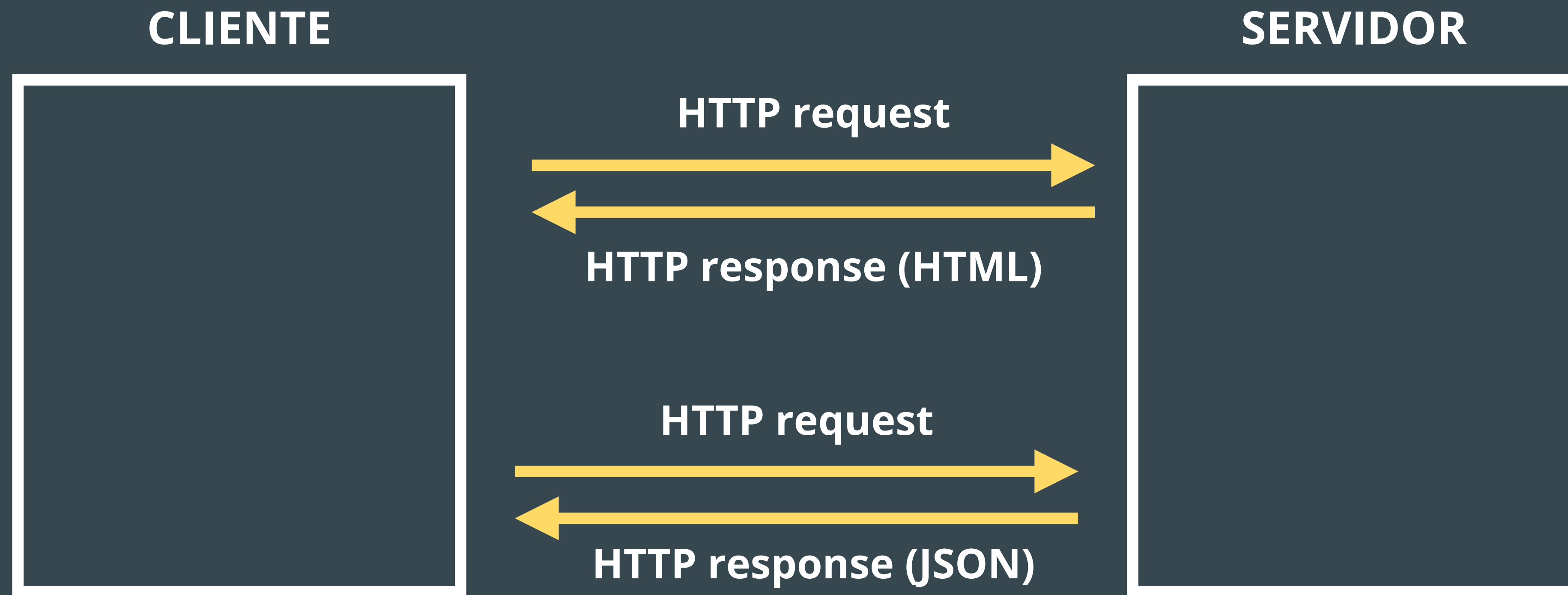
**Página en blanco** mientras  
se espera la respuesta

**Interacción del usuario** es  
constantemente **interrumpida**

# Client-side Frameworks

AJAX entra al juego

Interacción web **cambió con AJAX**



# Client-side Frameworks

AJAX entra al juego

Uso de **JavaScript en cliente aumentó mucho** en primera década del 2000

**Primeras aplicaciones** conocidas con **Ajax**

- Gmail (2004)
- Kayak.com (2004)
- Google Maps (2005)

# Client-side Frameworks

Segunda mitad de los 2000s

**Librerías** que permitían **manipular DOM y eventos**, y además con excelente **soporte para AJAX**



2005



2005



2006



2006



2007

# Single page applications

## Antecedentes

A medida que comenzó a **aumentar la cantidad de código JS en cliente**, su **complejidad también aumentó**

¿Era un **crecimiento sostenible**?

# Single page applications

## Antecedentes

**Actualizar la UI** cuando los datos cambian requiere una cantidad de líneas de código no menor

- Manipular el DOM es **verboso**
- Código se hace **poco mantenible**
- **Spaghetti code**: código difícil de seguir, ¿dónde hacer cambios?



# Single page application

¿Qué es?

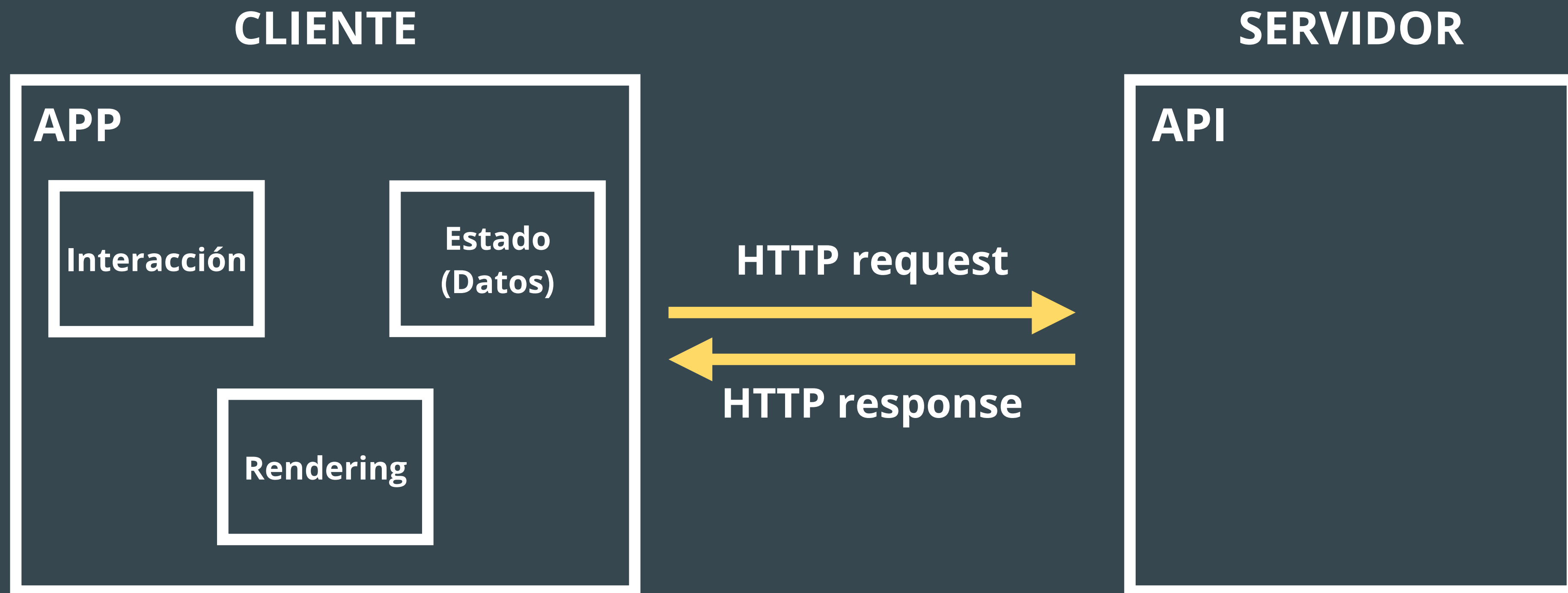
Una aplicación que consiste en **un documento HTML** base, **estilos** asociados, y **un script JS** que contiene toda la **lógica de frontend** de la aplicación

Recursos **se cargan una sola vez**, en la llamada **"primera carga"**

# Single page application

¿Qué es?

El script JS **contiene una aplicación** a cargo de 3 tareas



# Single page applications

Surgimiento frameworks MV\*

Desde el 2010 comenzaron a aparecer **nuevas librerías y frameworks** de más **alto nivel**

Permitían desarrollar **aplicaciones más complejas** de forma **más simple**

# Single page applications

Surgimiento frameworks MV\*

Basados en **paradigma MVC** con variaciones: **MV\***  
(MVC, MVVM, MVP)



BACKBONE.JS

2010 (pionero)



ANGULARJS

2010

Knockout.

2010

ember

2011

Entre otros...

# Single page applications

MUCHOS frameworks MV\*

## YAJS

Yet Another (JavaScript)  
Framework Syndrome

Más información

## TodoMVC

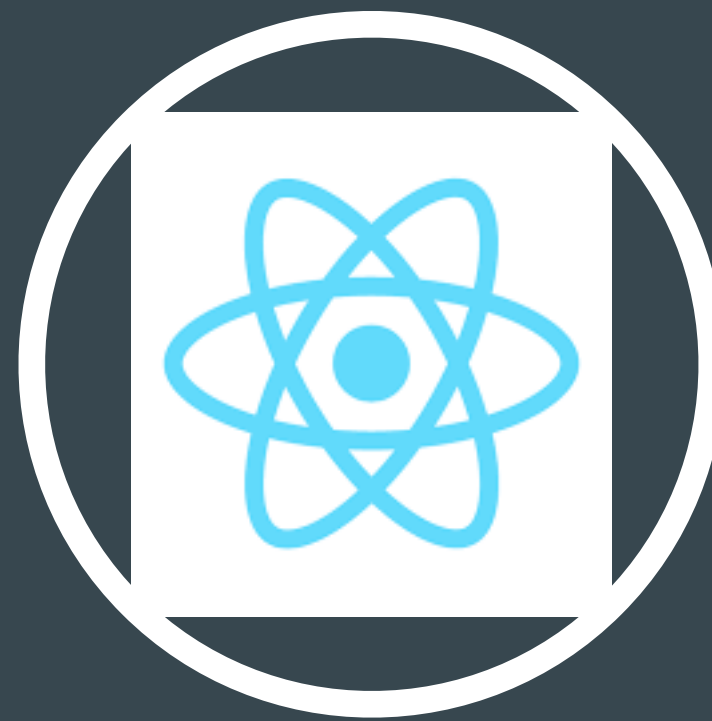
Implementación de una  
aplicación con diferentes  
frameworks

Afortunadamente, hoy en día ya está **más estable**

# Single page applications

Situación actual: frameworks populares

## Nivel de satisfacción



React

89% (2013)



Vue.js

87% (2014)



Angular

38% (2015)



Ember

31% (2011)

Fuente: State of JS 2019

# React

**Librería para construir  
interfaces de usuario**

# React

## Características

Utilizada para crear **pequeños componentes**, **partes de una UI**, o también la **UI completa** de una aplicación

- **Declarativa**: describir cómo se verá la UI
- Basada en **componentes**



# React

## Component-based architecture

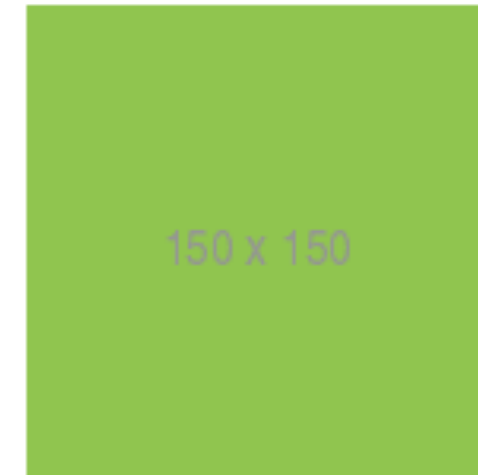
**Componente:** trozo de código que **describe** una **porción de la UI**

- **Reusable** y fácil de mantener
- Componentes se pueden **componer entre ellos**

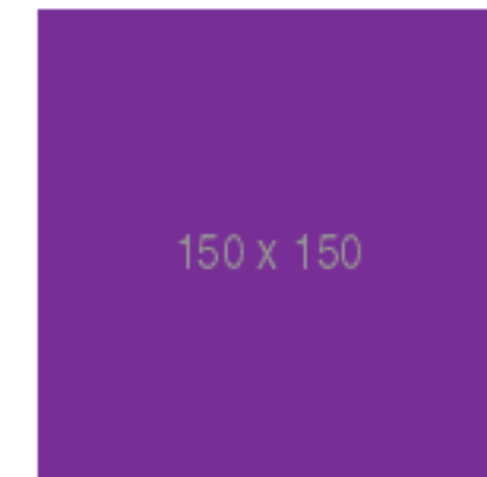
### React Photos

Load from endpoint

accusamus beatae ad  
facilis cum similique  
qui sunt



reprehenderit est  
deserunt velit ipsam



officia porro iure quia  
iusto qui ipsa ut modi

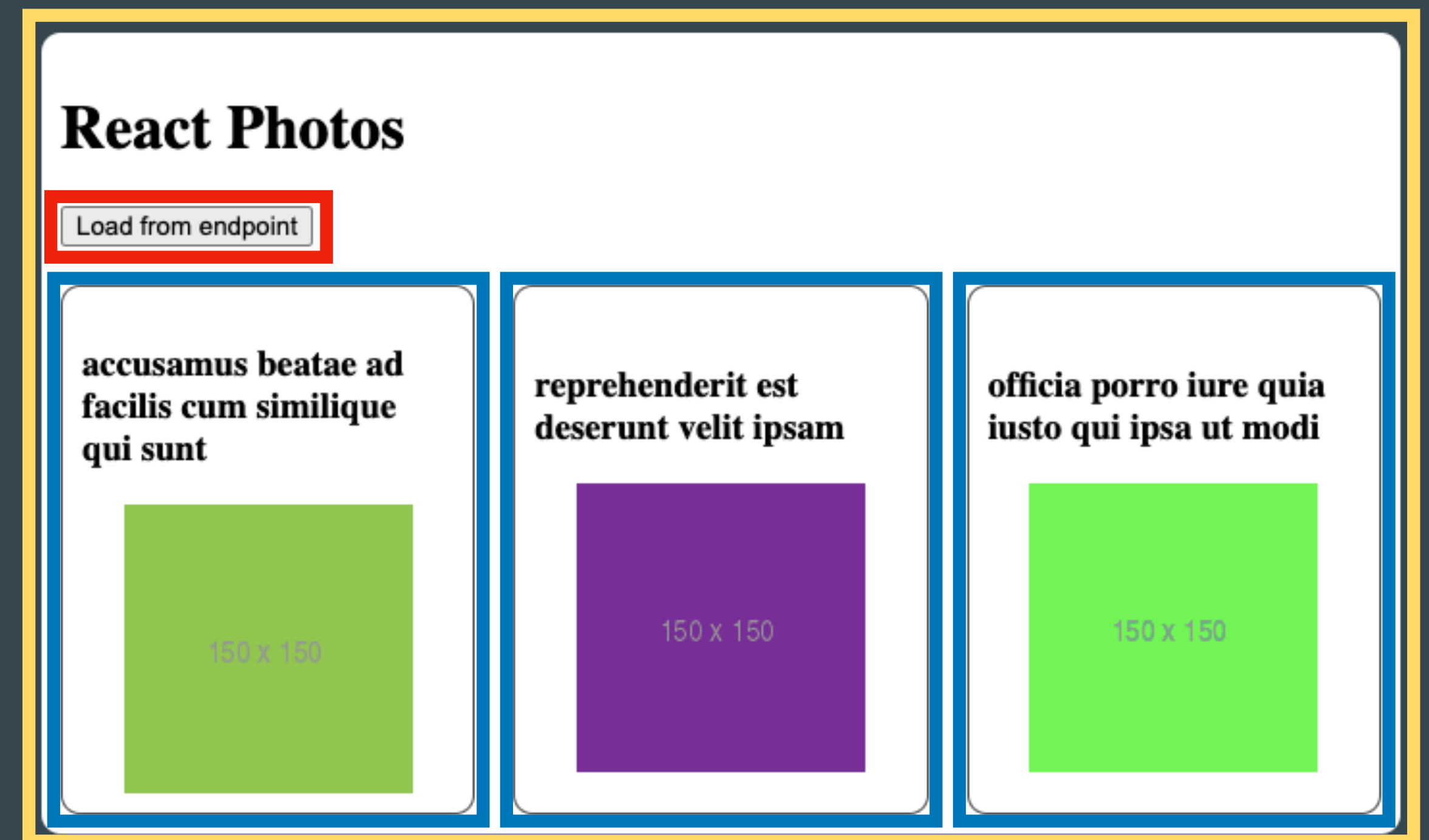


# React

## Component-based architecture

**Componente:** trozo de código que **describe** una **porción de la UI**

- **Reusable** y fácil de mantener
- Componentes se pueden **componer entre ellos**



# React

## Conceptos

# Virtual DOM

Copia del DOM en memoria

- Cambio de UI en componente: comparación Virtual DOM con DOM real
- Diff utilizado para actualizar DOM de forma óptima

# React

## Conceptos

# JSX

JavaScript and XML

Lenguaje similar a HTML que puede ser escrito con código JavaScript

```
<h1>Un título</h1>
```

Requiere herramienta que convierta JSX en JavaScript: Babel

# React

## Conceptos

### Elemento

`<h1>Un título</h1>`

- **Unidad mínima** de una aplicación React
- Se **traduce** en un **elemento del DOM**

### Render en DOM

Una aplicación React se **renderiza** **sobre un elemento** existente en el **DOM**

```
const rootNode = document.getElementById('root');
```

```
ReactDOM.render(<h1>Un título</h1>, rootNode);
```

# Componentes

**Encapsular porciones de UI**

# Componentes en React

Trozos de código que describen una porción de la UI

- **Reciben input y retornan** lo que debiese aparecer **visualmente** (como una función)
- Está compuesto de **elementos**

# Componentes en React

Trozos de código que describen una porción de la UI

**Dos tipos** de componentes

## Funcional

```
function MyComponent(props) {  
  return (  
    <h1>Un título</h1>  
  );  
}
```

## De clase

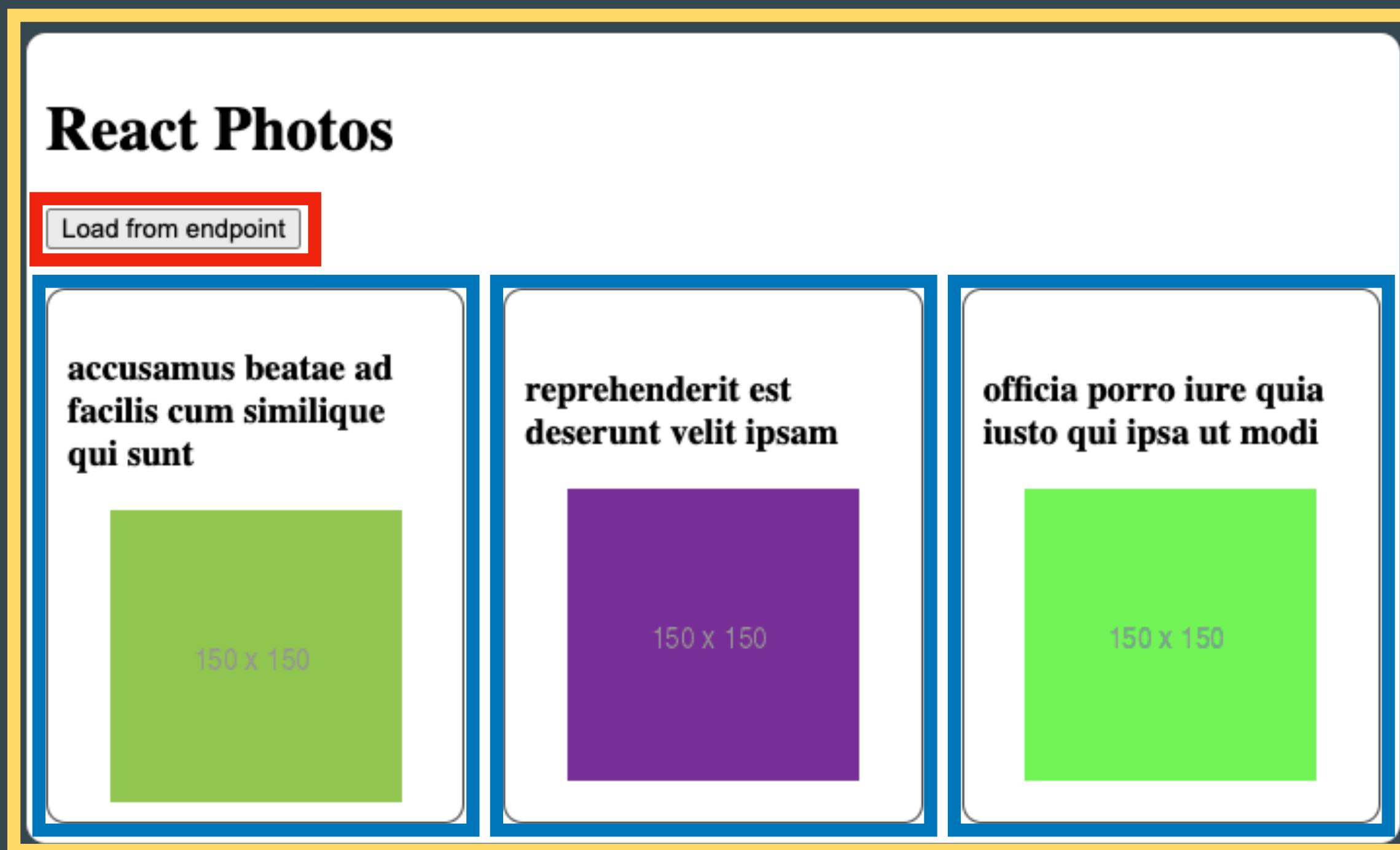
```
class MyComponent extends React.Component {  
  constructor(props) {  
    super(props);  
  }  
  
  render() {  
    return (  
      <h1>Un título</h1>  
    );  
  }  
}
```



# Componentes en React

¿Cómo comenzar?

Una forma es escribiendo un **componente estático**



```
function Photos(props) {  
  return (  
    <div>  
      <h1>React Photos</h1>  
      <Button />  
      <ul>  
        <SinglePhoto />  
        <SinglePhoto />  
        <SinglePhoto />  
      </ul>  
    </div>  
  );  
}
```

# Componentes en React

render

render

- Define **cómo se verá** el componente **en un determinado momento**
- Es el **retorno** de un componente funcional
- Función **render** en componente de clase

# Componentes en React

## props y state

### props

- **Input necesario** para hacer **render** del componente
- Sólo se debe **leer**
- Input **entregado** por **componente padre**

### state

- **Propiedad interna** de un componente que puede **cambiar en el tiempo**
- **useState** (funcional)
- **this.state / setState** (de clase)

En detalle más adelante...

Cada vez que **cambien props o state** de un componente, se hará **render nuevamente** de este

# Componentes en React

props

Input necesario para hacer render del componente

Entregado por componente padre

`<SinglePhoto photo={photo} />`



```
function SinglePhoto(props) {  
  const { photo } = props;  
  return (  
    <li className="photo">  
      <h3>{photo.title}</h3>  
      <img src={photo.thumbnailUrl} />  
    </li>  
  );  
}
```

# Componentes en React

state

**Propiedad interna** de un componente que puede **cambiar en el tiempo**

**Hook** `useState`

```
const [photos, setPhotos] = React.useState([]);
```



Valor actual  
state



Función para  
actualizar state



Valor inicial  
state

# Componentes en React

## Event handling

**Eventos en React** se manejan de manera similar a **eventos en el DOM**

(Similar a eventos inline 🤔)

```
<button onClick={handleClick}>Click me</button>
```



camelCase



Función handler  
(no string)

Caso de uso: **Actualizar state dentro de handler**

# Componentes

**Ciclo de vida**

# Ciclo de vida de un componente

¿Cómo funciona?

A un componente le ocurren 3 eventos en su vida



Fuente: [donavon's Github](#)



# Ciclo de vida de un componente


## Side effects en ciclo de vida

### Hook `useEffect`

```
useEffect(() => {  
  // Side effect code  
}, []);
```



Array  
dependencias



Función que se ejecuta  
en mount o update

# Referencias

- MDN - "Introduction to client-side frameworks"
- Flavio Copes - "What is a Single Page Application?"
- Angular University - "Angular SPA: Why Single Page Applications?"
- Ryan Mackey-Paulsen - "Web Component Based Architecture"
- React - Página oficial
- React - Documentación oficial (main concepts)
- React - Tutorial práctico oficial
- React - "Thinking in React"
- Egghead - "The Beginner's Guide to React" (curso online "corto")