

---

# IIC2026

# Visualización de Información

— Hernán F. Valdivieso López —  
(2023 - 2 / Clase 08)

---

# Temas de la clase - Utilidades D3 II

1. Eventos.
2. Transiciones.
3. Data Join personalizado.

# Eventos

---

# Eventos

Anteriormente vimos que podemos conectar eventos del DOM con Javascript.

```
// Creamos un elemento con tag <h1>
```

```
const elemento = document.createElement("h1");
```

```
// Conectamos la ocurrencia de un evento en el elemento a una función
```

```
elemento.addEventListener(evento, funcion);
```

¿Qué evento puede ser?

- Click sobre el elemento ([click](#)).
- Pasar el mouse sobre el elemento ([mouseover](#)).
- Copiar el elemento ([copy](#)).
- Hacer scroll en el elemento ([scroll](#)).
- Muchos más: [https://www.w3schools.com/jsref/dom\\_obj\\_event.asp](https://www.w3schools.com/jsref/dom_obj_event.asp)

D3 también puede hacer lo mismo 🎉.

# Eventos

🤔 ¿Cómo se hace en D3?

1. Generar una selección de elementos.
2. Utilizar el método `.on(evento, function)` para indicar que todos los elementos de dicha selección gatillen dicho evento.

Por ejemplo:

```
// Seleccionamos todos los rect
const elementos = d3.selectAll("rect");

// Cada vez que hagamos click, imprime uwu
elementos.on("click", () => console.log("uwu"));
```

# Eventos

🤔 ¿Cómo se hace en D3?

```
elementos.on("click", (evento, data) => {  
    console.log(evento);  
    console.log(evento.currentTarget);  
    console.log(data);  
});
```

Los eventos en D3 le entregan 2 elementos a la función:

1. Un objeto evento. Aquí podemos ver que tipo de evento fue y acceder, por ejemplo, al elemento HTML que gatilló el evento.
2. El dato asociado al elemento HTML que gatilló el evento. Este solo existe si previamente se hizo un *data joins*, en otro caso será `undefined`.

# Eventos

En la clase de hoy veremos 3 eventos:

1. `selection.on('click', ...)`: cuando haces *click* en un elemento de la selección.
2. `selection.on('mouseenter', ...)`: cuando el mouse pasa por encima de un elemento de la selección.
3. `selection.on('mouseleave', ...)`: cuando el mouse deja de estar por encima de un elemento de la selección.

Existe más eventos que veremos más adelante, tales como:

1. Arrastrar un elemento por la pantalla.
2. Doble click en un elemento.
3. Hacer zoom con el mouse.

# Eventos

Vamos al código  



# Transiciones

---

# Transiciones

- Interfaz que entrega D3 para crear animaciones sobre elementos del DOM.
- Podemos crear diferentes tipos de animaciones:
  - Traslado de figuras.
  - Cambio de color.
  - Cambio de tamaño.

# Transiciones

## Formato

```
d3.select("rect").transition().duration(200).attr("x", 200).attr("fill", "red")
```

1. Definir una selección.
2. Agregar `transition()` para indicar que todos los cambios siguientes en la selección van a ser animados.
3. Agregar `duration()` para indicar cuantos milisegundos debe durar la animación.
4. Usar tantos `attr()` como uno desea para alternar los atributos de una forma animada.

# Transiciones

## Múltiples transiciones

```
d3.select("rect")  
  .transition().duration(200).attr("x", 200)  
  .transition().attr("fill", "red")  
  .transition().attr("y", 1000)
```

Podemos definir varias transiciones para que ocurra una después de otra.

# Transiciones

## Múltiples transiciones 🙄 🙄

```
const rects = d3.select("rect")  
rects.transition().duration(200).attr("x", 200).attr("fill", "red") // T1  
rects.transition().duration(200).attr("y", 300) // T2
```

- En este caso, la transición 2 (T2) se ejecutará y la 1 no.
- Esto ocurre porque T2 sobrescribe la transición actual.
- Todo se soluciona agregando nombres a las transiciones para indicar a D3 que son diferentes transiciones.

```
rects.transition("t1").duration(200).attr("x", 200).attr("fill", "red") // T1  
rects.transition("t2").duration(200).attr("y", 300) // T2
```

# Transiciones

## Cuidado con lo que retorna una transición 👁️ 👁️

```
const rects = d3.select("rect")  
rects.transition() != rects
```

- Cuando se aplica una transition, ya no se retorna la selección. Se retorna un objeto tipo transición.
- Cuando hagan *data join* asegurarse de no retornar una transición. Sino todo puede llegar a fallar 😭

# Transiciones

Vamos al código 

# Transiciones - extra

- Podemos retrasar las transiciones con el comando `delay()`.
- También podemos cambiar la velocidad de la transición o dar efectos a las transiciones. Todo esto con el comando `ease()`.
- Encontrarán ejemplos de estos comandos en: [Transitions | D3 in Depth](#)



# Data joins personalizado

---

# Data joins personalizado

Imaginemos los siguientes elementos y datos

```
[{"id": "A", "valor": 4}, {"id": "B", "valor": 15}, {"id": "C", "valor": 15}, {"letra": "D", "valor": 99}, {"letra": "E", "valor": 102}]
```

Elementos

<rect>

<rect>

<rect>

<rect>

<rect>

Datos

4

15

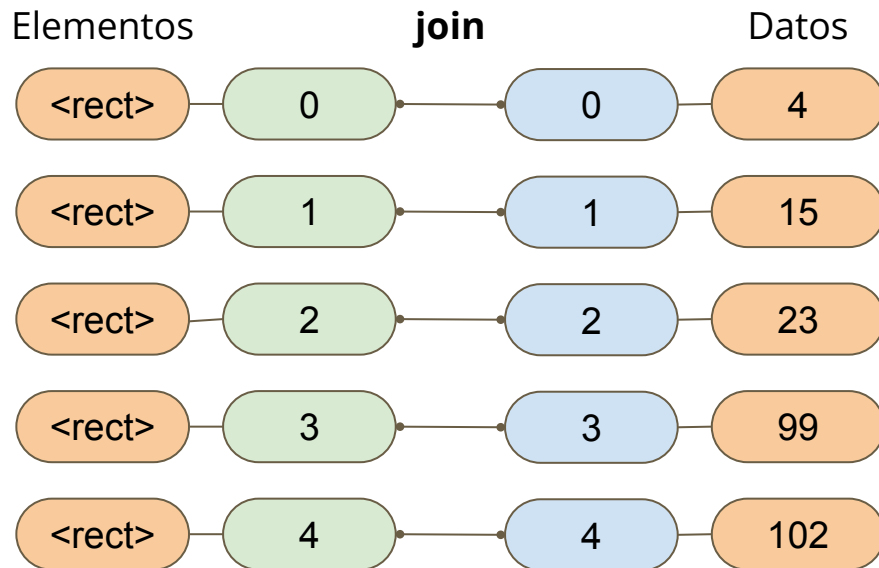
23

99

102

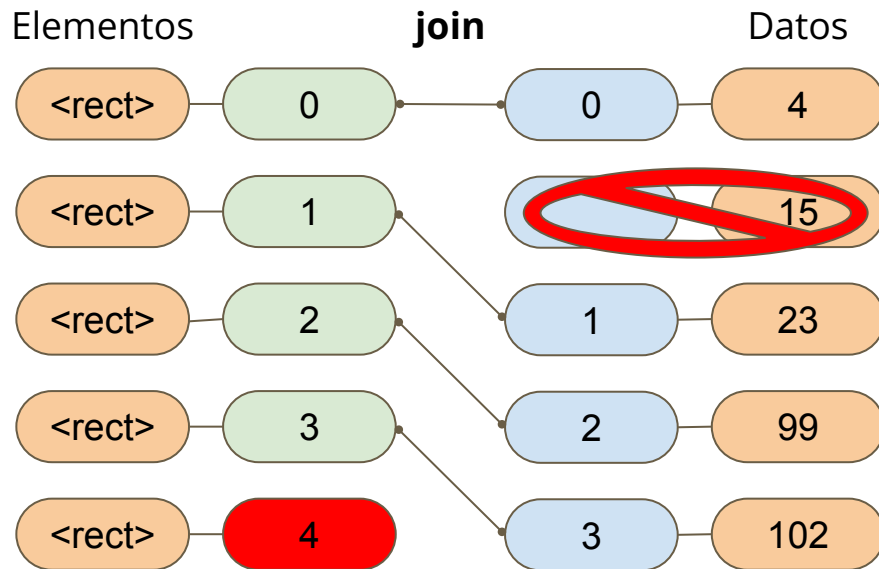
# Data joins personalizado

Hacemos `.data(datos)`. Por lo tanto, D3, **usa el índice de los elementos y el índice de los datos como llave.**



# Data joins personalizado

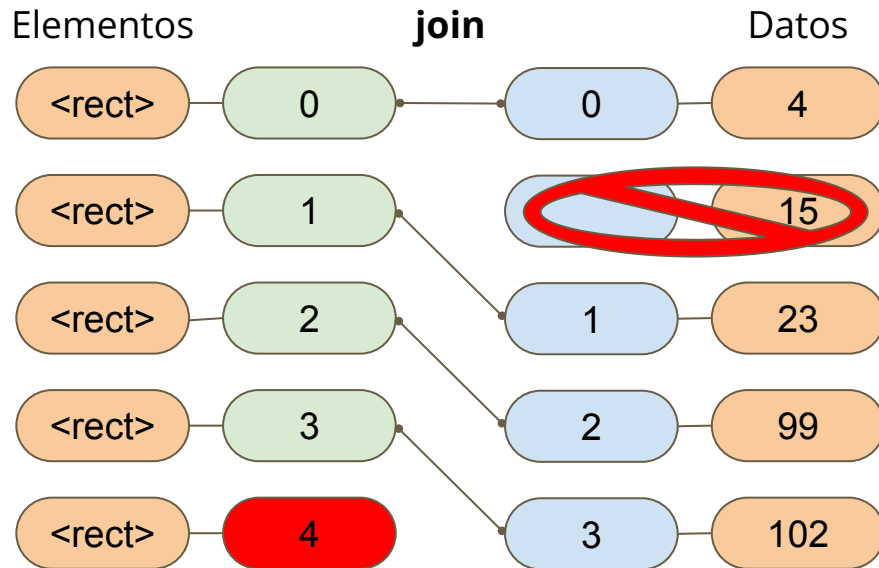
Eliminamos el dato 15. Esto hace que el índice de los siguientes datos se actualice. Por lo tanto, el data join conecta cada rect con su nuevo dato y deja el último rect sin datos.



¿Está bien eso?

# Data joins personalizado

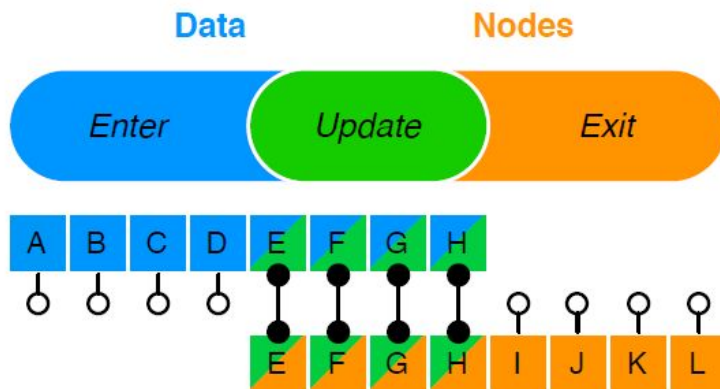
Eliminamos el dato 15. Esto hace que el índice de los siguientes datos se actualice. Por lo tanto, el data join conecta cada rect con su nuevo dato y deja el último rect sin datos.



¿Está bien eso?  
👤 NO

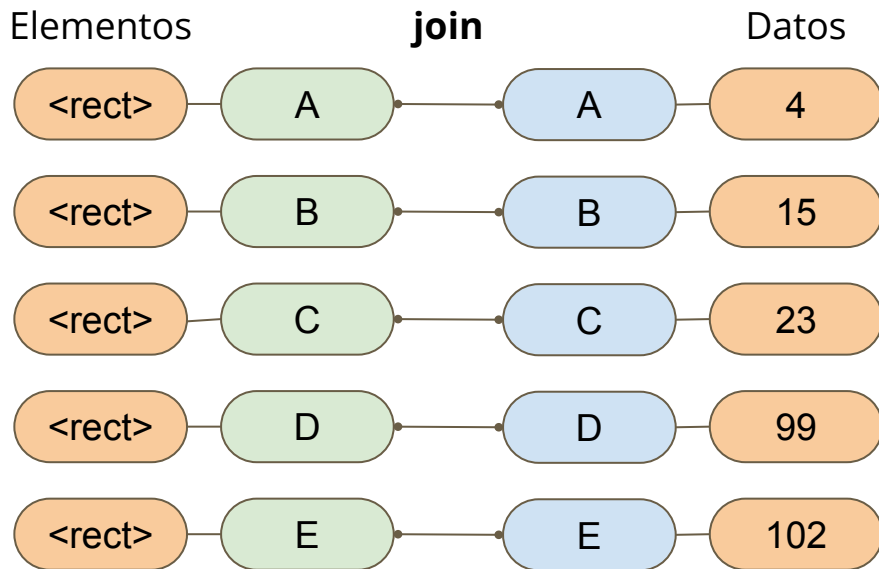
# Data joins personalizado - ¿por qué ocurre esto? 🤔

- Esto se debe al funcionamiento **interno** del data joins.
- El vínculo de "elemento visual - dato" se hace mediante una llave. Si no le damos esa llave, usa el índice de los datos como llave.
- Podemos verlo como: *"Yo {elemento visual} observaré siempre al dato con llave {inserte llave}. Si esa llave desaparece, yo desaparezco. Si los datos de esa llave cambian, yo actualizo mi información visual"*.



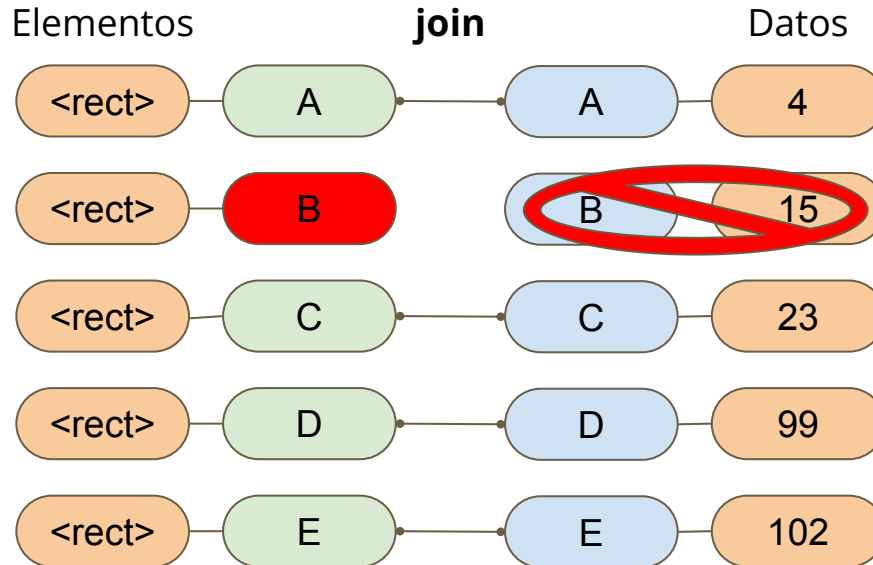
# Data joins personalizado

Tenemos que hacer `.data(datos, d => d.id)` para usar una llave (una categoría única) por dato. Por lo tanto, ahora cada elemento visual queda vinculado a una letra y solo se modifica si dicha letra le ocurre algo.



# Data joins personalizado

Eliminamos el dato 15, pero la llave ya no es el índice, sino una letra. Los demás datos no cambian su llave. Con esto hacemos que el rect conectado al 15 sea eliminado.





# Data joins personalizado - Ejemplo 2

Imaginemos los siguientes elementos y datos

```
[{"id": "A", "valor": 4}, {"id": "B", "valor": 15}, {"id": "C", "valor": 15}, {"letra": "D", "valor": 99}]
```

Elementos

<rect>

<rect>

<rect>

<rect>

Datos

4

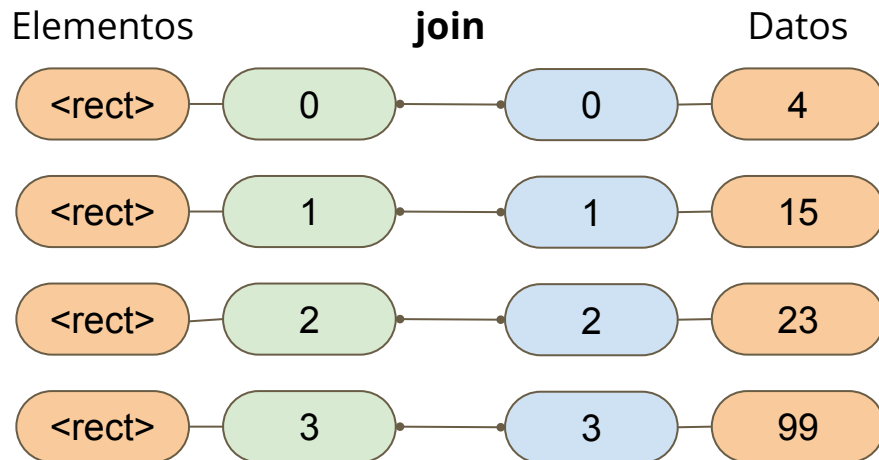
15

23

99

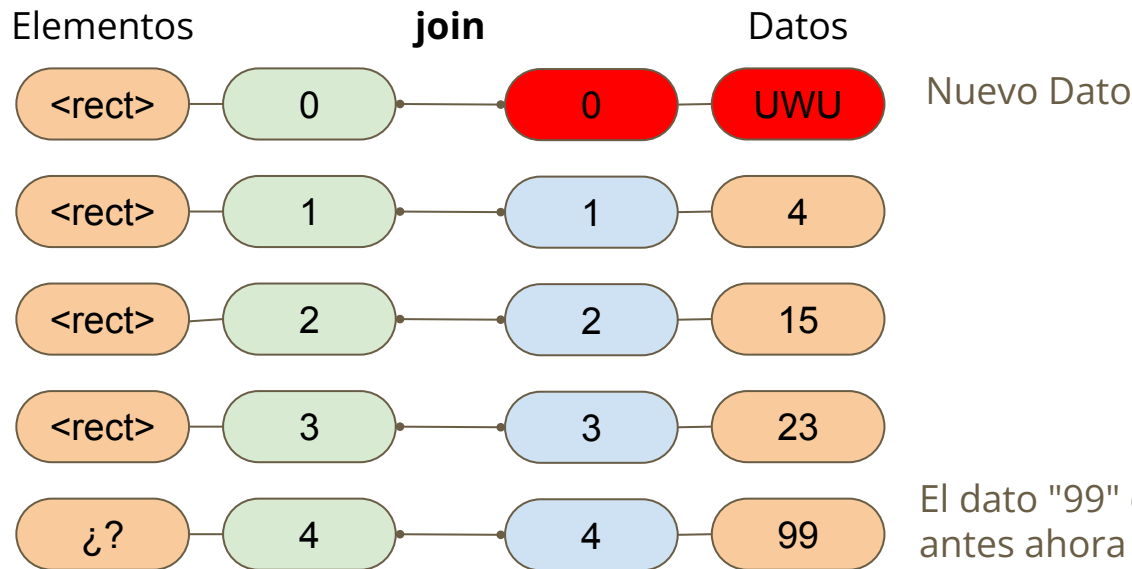
# Data joins personalizado - Ejemplo 2

Hacemos `.data(datos)` sin dar una llave. Por lo tanto, D3, **usa el índice de los elementos y el índice de los datos como llave.**



# Data joins personalizado - Ejemplo 2

Vamos a insertar un nuevo dato al inicio de nuestra lista (`{"id": "E", "valor": "UWU"}`) y hacemos `join. .data(datos)`

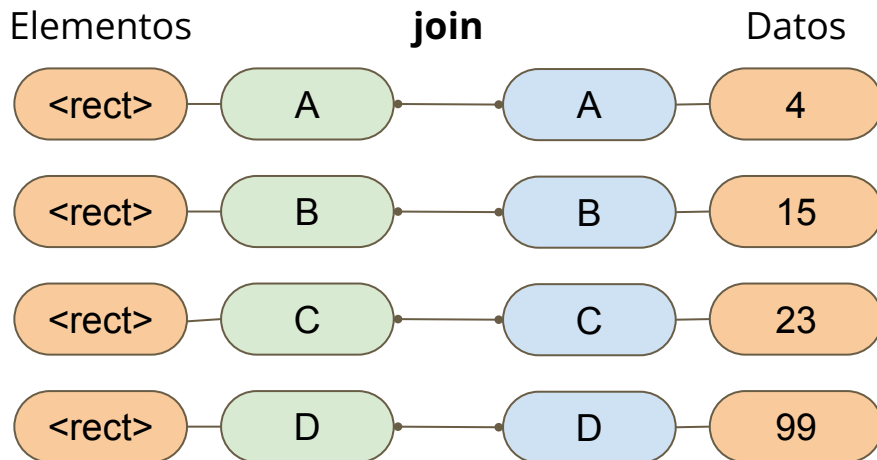


Para D3, el nuevo elemento visual es el conectado a 99

El dato "99" que ya existía antes ahora no tiene un <rect> asociado

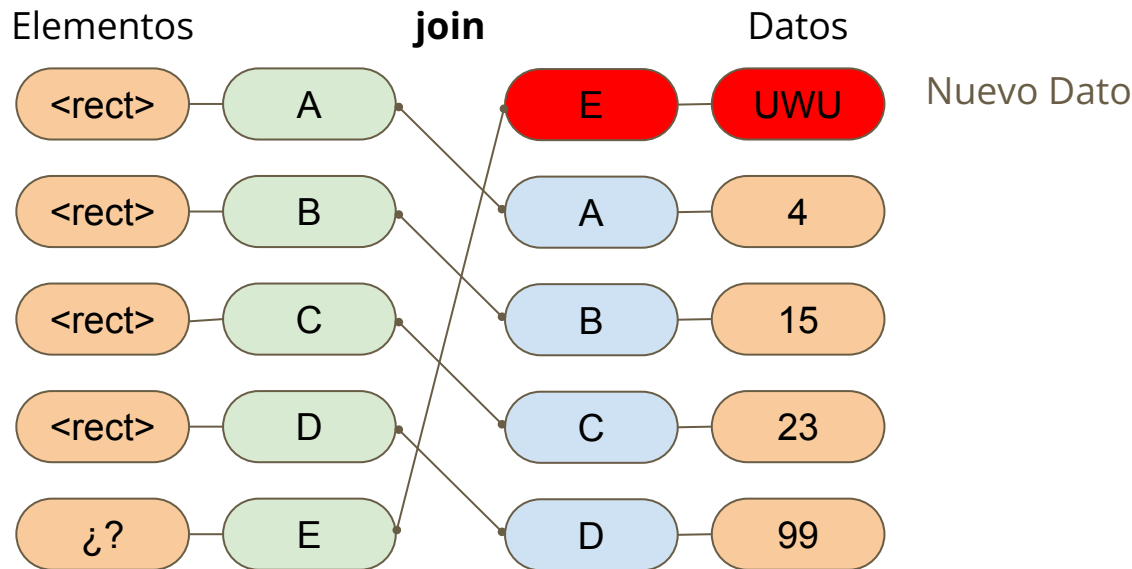
# Data joins personalizado - Ejemplo 2

Si hubiéramos hecho `.data(datos, d => d.id)` usando una categoría única por dato. Por ejemplo, letras.



# Data joins personalizado - Ejemplo 2

Vamos a insertar un nuevo dato al inicial (`{"id": "E", "valor": "UWU"}`) y hacer join. Como la llave "E" nunca la había visto, sabe que el nuevo *rect* debe conectarse a ese dato, sin importar donde está en la lista.



Para D3, el nuevo elemento visual es el conectado a UWU

# Data joins personalizado

Vamos al código  

# Data joins personalizado

🤔 ¿En qué casos es **imperante** utilizar *data join personalizado*?

1. Cuando la visualización permite filtros
2. Cuando la visualización permite ordenar los elementos visuales
3. Cuando usamos transiciones al momento que aparece un nuevo dato o desaparece uno en particular.
4. TLDR: cuando hay manipulación o algún cambio en el tiempo.

**Recomendación personal:** siempre que los datos tengan una llave (*id* por ejemplo), usen data join personalizado.

# Data joins personalizado

🤔 ¿En qué casos es **imperante** utilizar *data join personalizado*?


1. Cuando la visualización permite filtros
2. **Cuando la visualización permite ordenar los elementos visuales**
3. Cuando usamos transiciones al momento que aparece un nuevo dato o desaparece uno en particular.
4. TLDR: cuando hay manipulación o algún cambio en el tiempo.

Se subió "*ejemplo\_desordenar.js*" que reordena aleatoriamente los datos cada vez que se hace click en el SVG. Estudien en casa este caso y prueben sacando el *data joins* personalizado para ver qué sucede.



# Próximos eventos

## Próxima clase

- Layout tabulares en D3.
- Traigan notebook 

## Tarea 2

Se sube el viernes. Hacer dos visualizaciones artística con transiciones, eventos, data join personalizado, etc.

## Viernes

Ayudantía donde harán un ejercicio **muy similar** a la tarea 2.

---

# IIC2026

# Visualización de Información

— Hernán F. Valdivieso López —  
(2023 - 2 / Clase 08)

---