`tt()`

Schedule

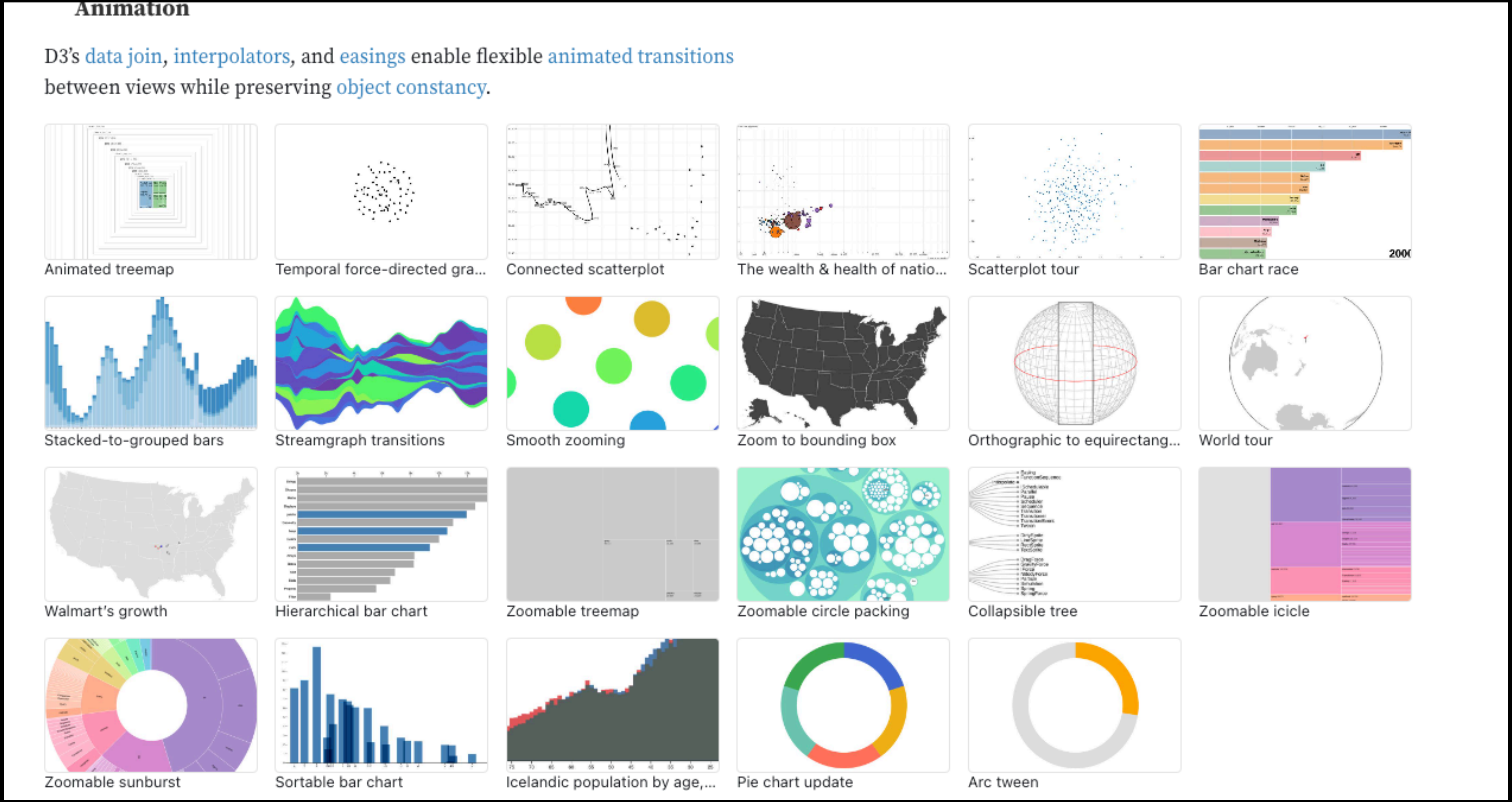
1. Data joins (enter, update exit)
2. Dynamic filtering
3. Events
4. All together!

Schedule

1. **Data joins (enter, update exit)**
2. Dynamic filtering
3. Events
4. All together!

Data joins

> Data joins allow you to **manage the dynamic aspects of data visualization.** Changing elements in your visualization based on changes in the dataset. […]


Animation

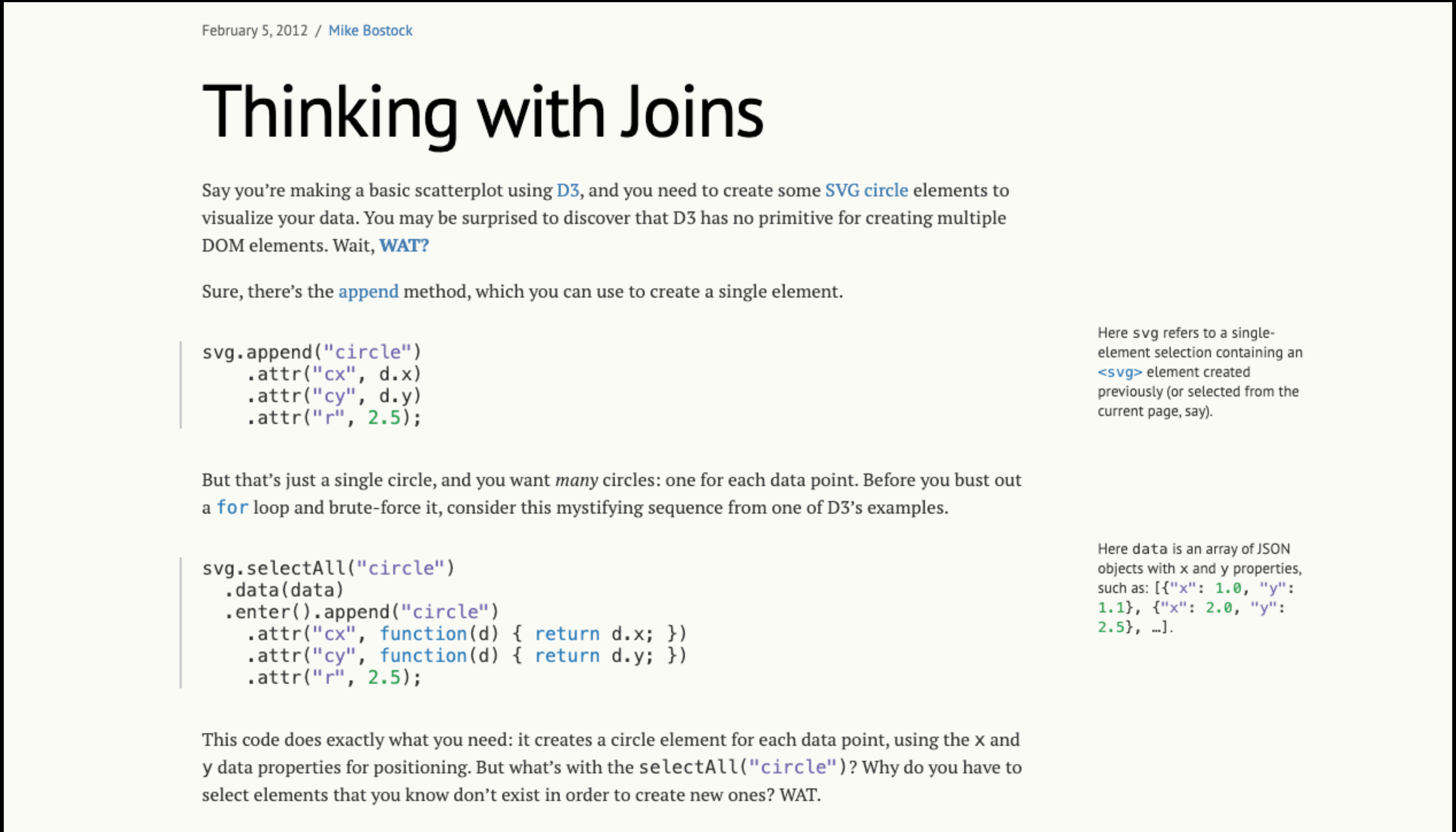
D3's data join, interpolators, and easings enable flexible animated transitions between views while preserving object constancy.

e.g. filter dropdown, timelines etc.

Data joins

This allows developers to *create*, *update*, or *remove* elements in the DOM based on changes in data. This concept is commonly known as **data binding.**

Data binding



February 5, 2012 / Mike Bostock

Thinking with Joins

Say you're making a basic scatterplot using D3, and you need to create some SVG circle elements to visualize your data. You may be surprised to discover that D3 has no primitive for creating multiple DOM elements. Wait, **WAT?**

Sure, there's the append method, which you can use to create a single element.

```
svg.append("circle")
    .attr("cx", d.x)
    .attr("cy", d.y)
    .attr("r", 2.5);
```

Here `svg` refers to a single-element selection containing an `<svg>` element created previously (or selected from the current page, say).

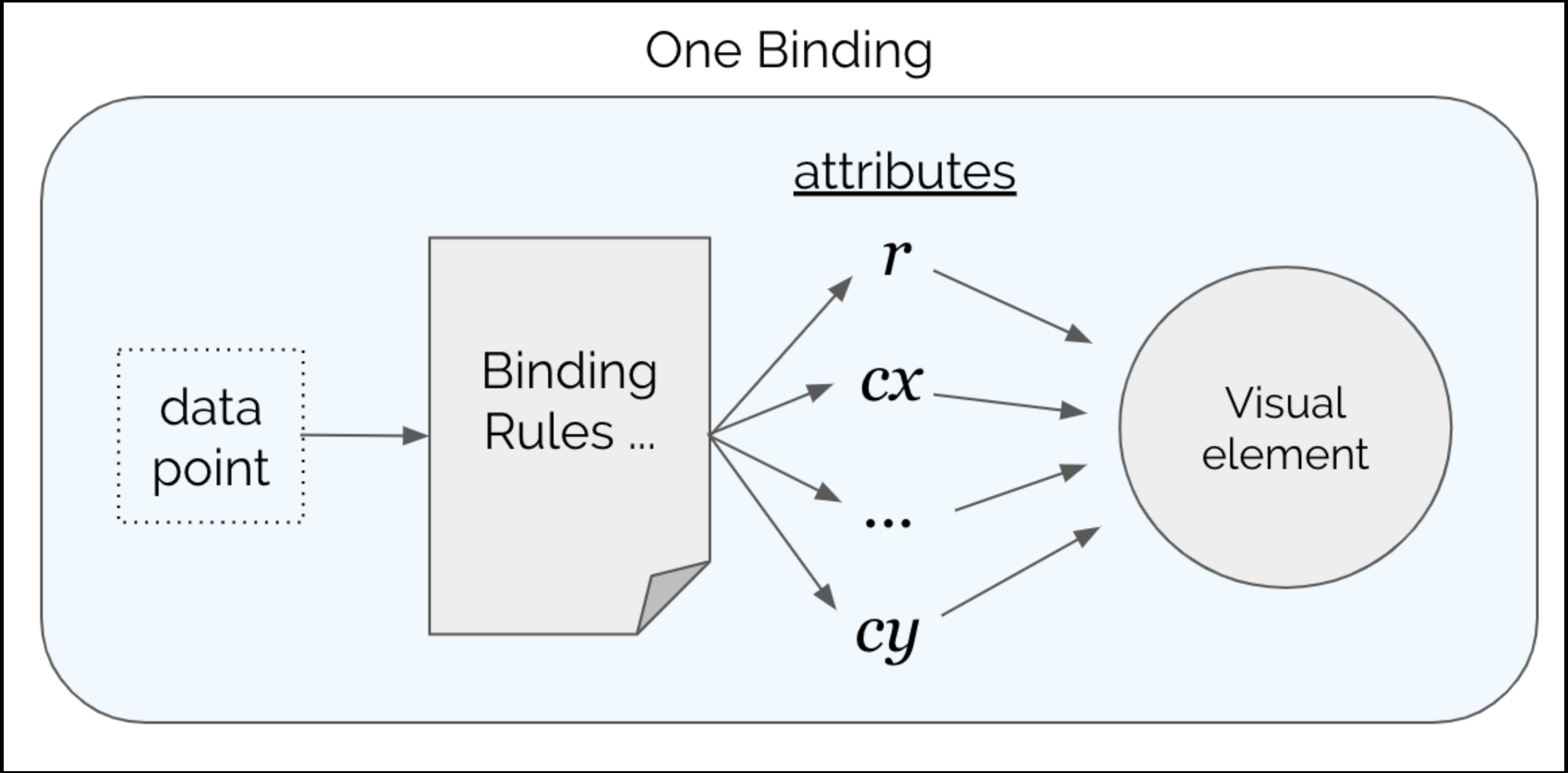
But that's just a single circle, and you want *many* circles: one for each data point. Before you bust out a `for` loop and brute-force it, consider this mystifying sequence from one of D3's examples.

```
svg.selectAll("circle")
  .data(data)
  .enter().append("circle")
    .attr("cx", function(d) { return d.x; })
    .attr("cy", function(d) { return d.y; })
    .attr("r", 2.5);
```

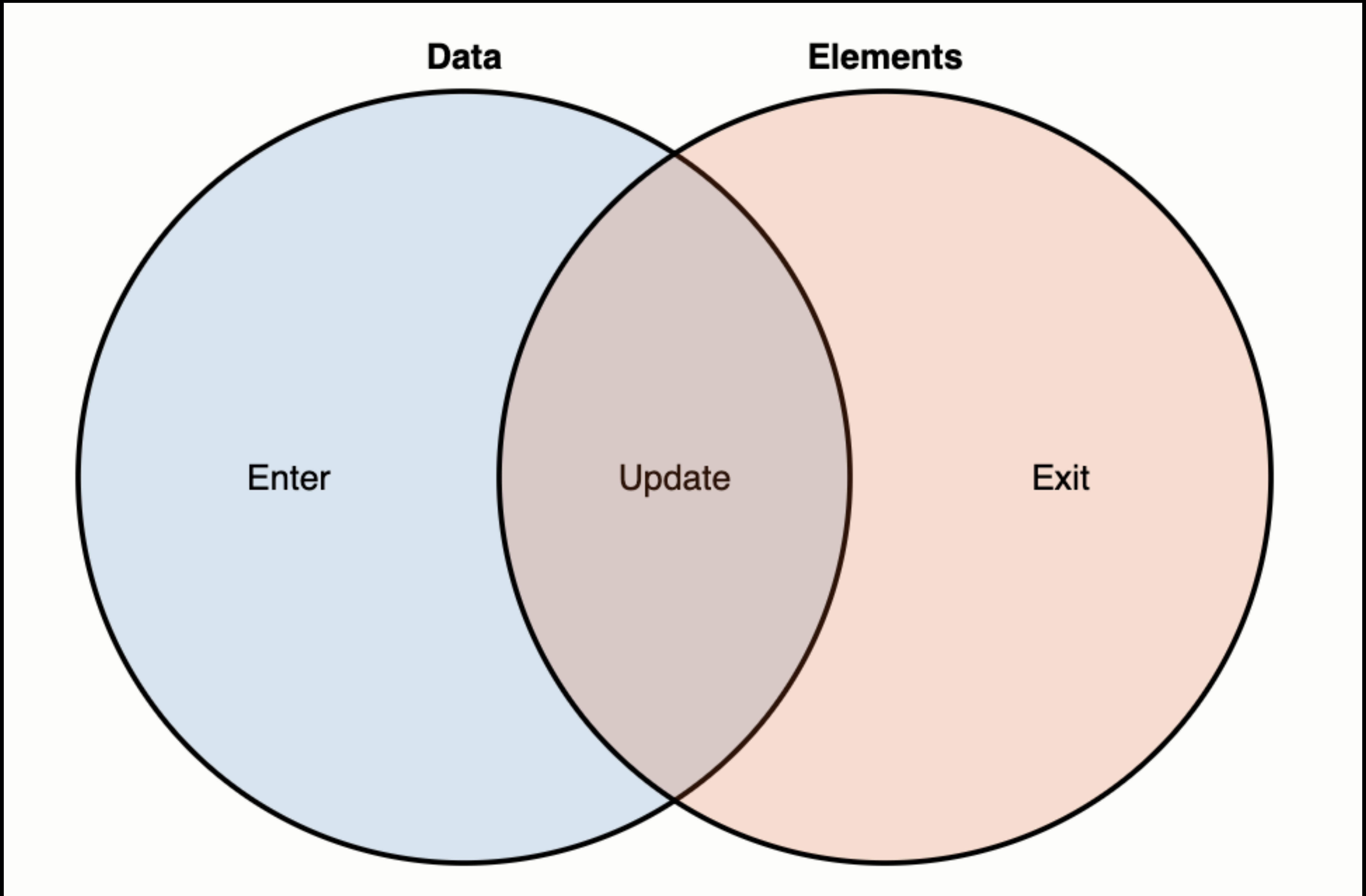
Here `data` is an array of JSON objects with x and y properties, such as: `[{"x": 1.0, "y": 1.1}, {"x": 2.0, "y": 2.5}, …]`.

This code does exactly what you need: it creates a circle element for each data point, using the `x` and `y` data properties for positioning. But what's with the `selectAll("circle")`? Why do you have to select elements that you know don't exist in order to create new ones? WAT.

Data binding



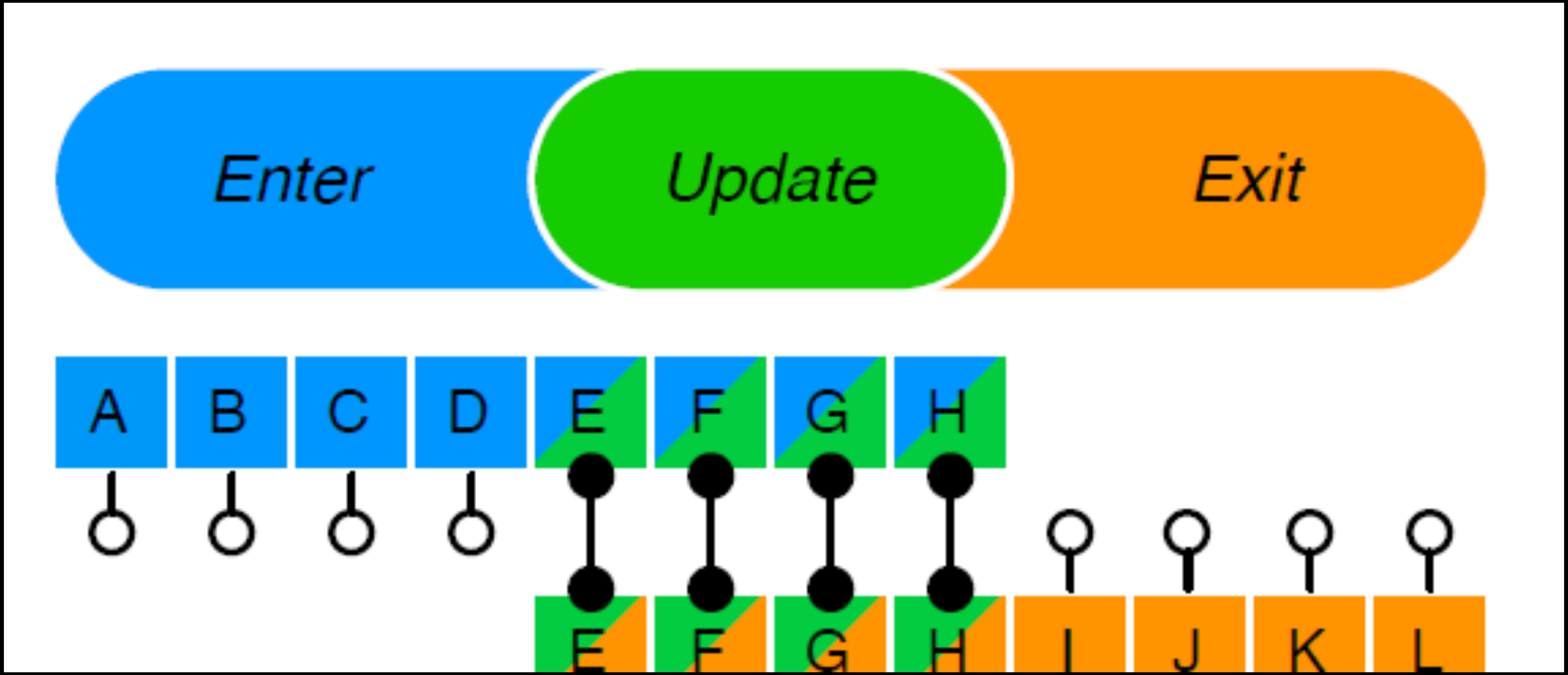
Data binding



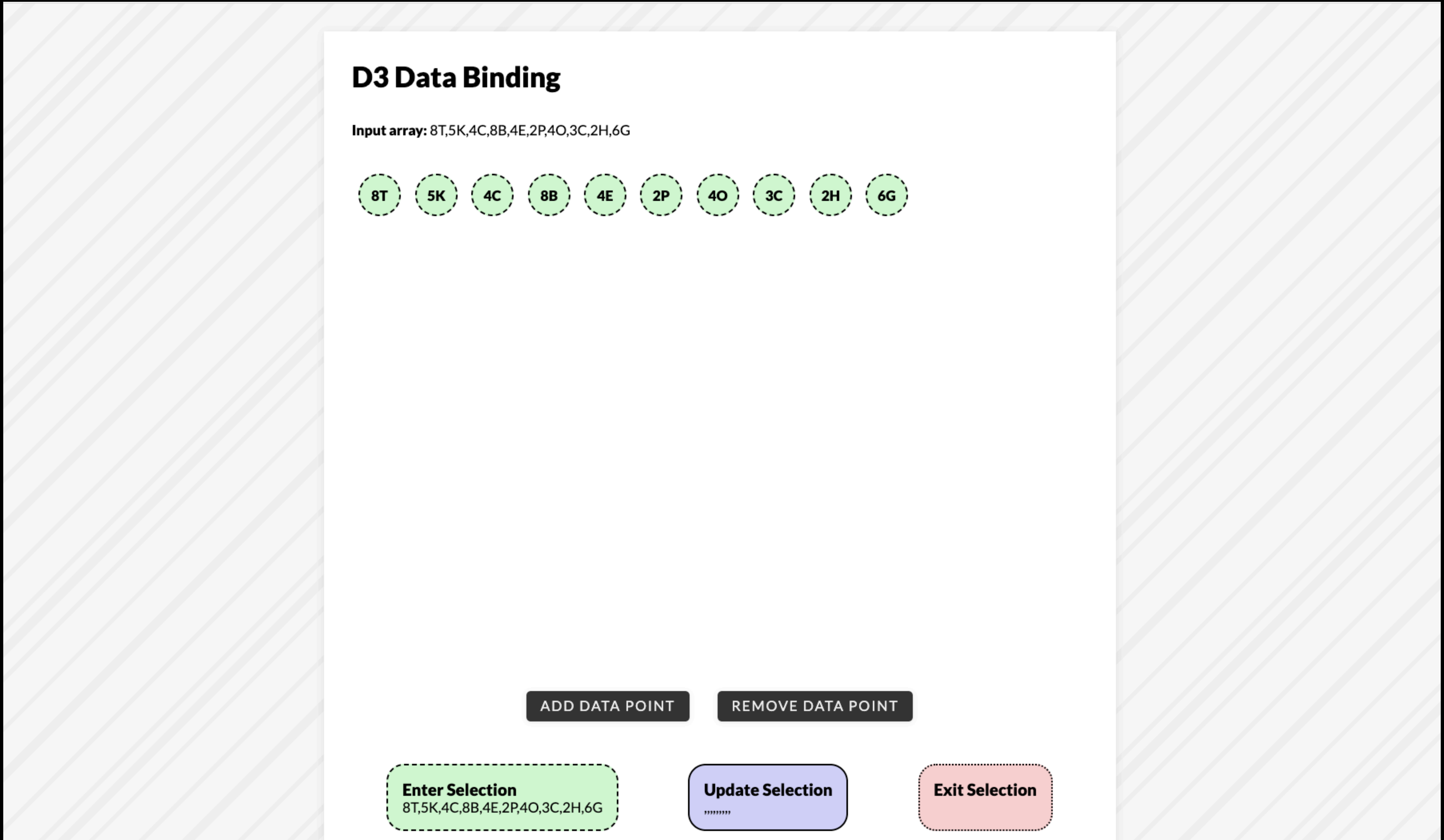
Data joins

1. enter() - elements that need to be *created*
2. update() - elements that already *selected*
3. exit() - elements that need to be *removed*

Data binding



Data binding – example



D3 Data Binding

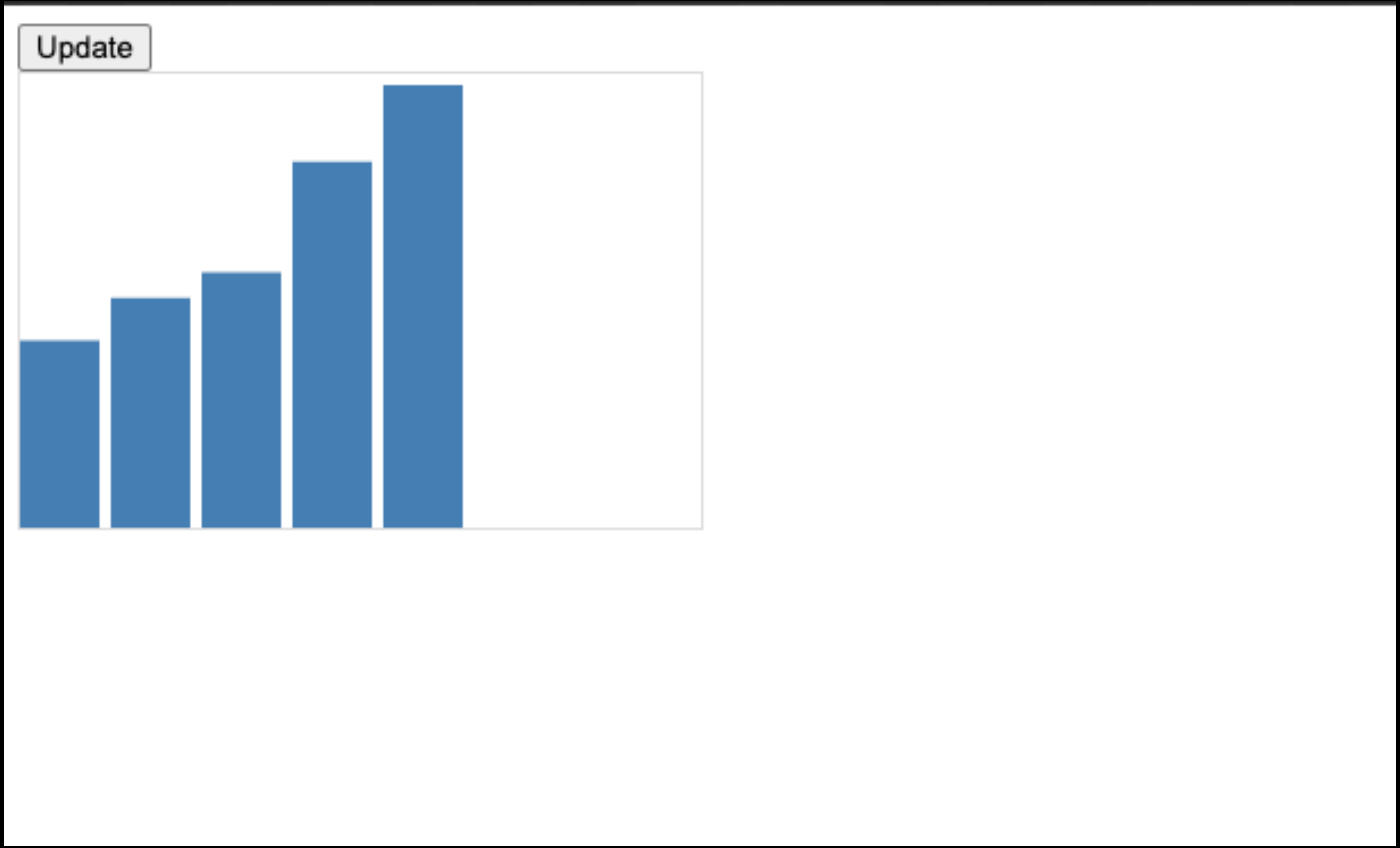
Input array: 8T,5K,4C,8B,4E,2P,4O,3C,2H,6G

Enter Selection
8T,5K,4C,8B,4E,2P,4O,3C,2H,6G

Update Selection
,,,,,,,,,

Exit Selection

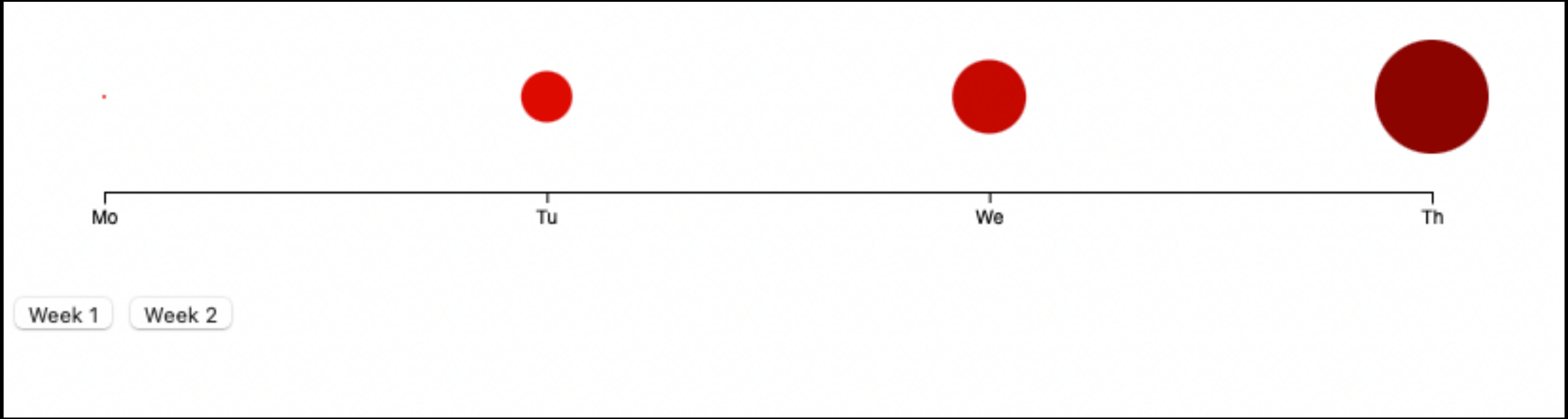
Data binding – example



Schedule

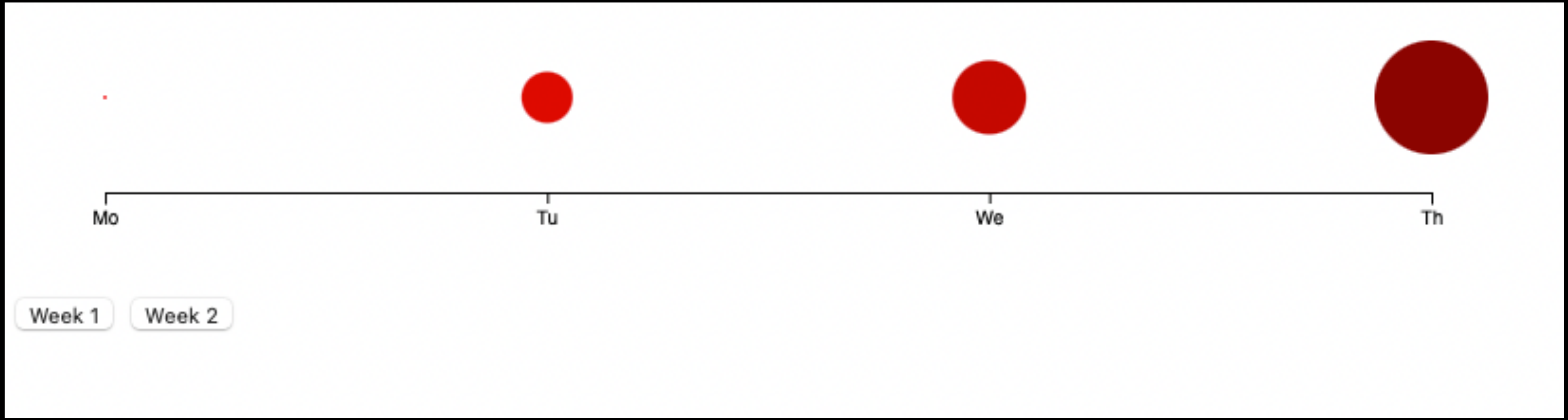
1. Data joins (enter, update exit)
2. **Dynamic filtering**
3. Events
4. All together!

Filtering demo



Filtering exercise

https://codepen.io/dandevri/pen/KKOGJev



Schedule

1. Data joins (enter, update exit)
2. Dynamic filtering
3. **Events**
4. All together!

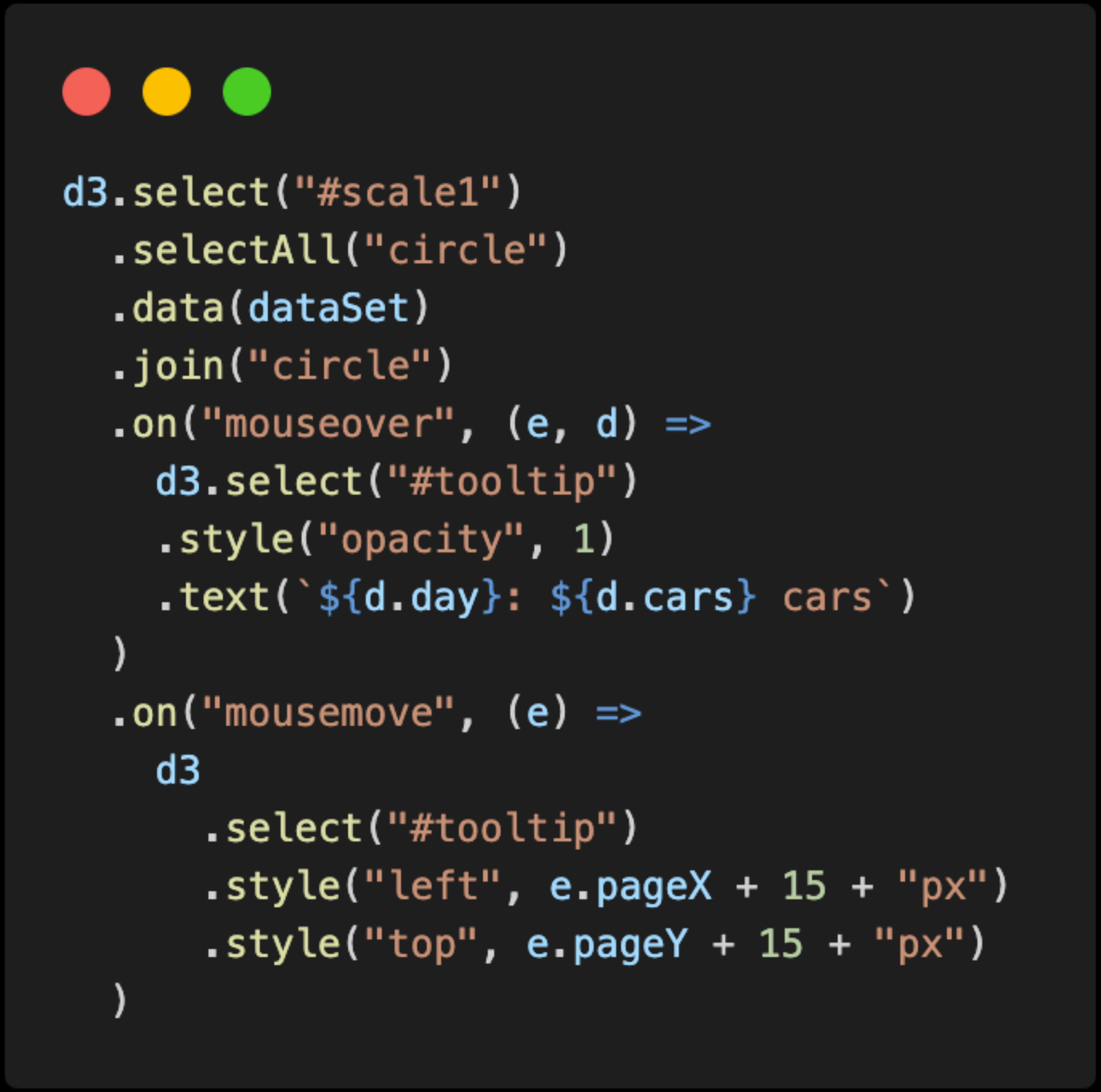
Events

[…] refer to interactions or actions which events allow developers to add **interactivity to data visualizations** by responding to user actions.

Events

1. Mouse events (clicks, mouseover etc.)
2. Keyboard events (keypress etc.)
3. Touch events (longpress etc.)
4. Drag & Zoom events (drag etc.)

Event Binding



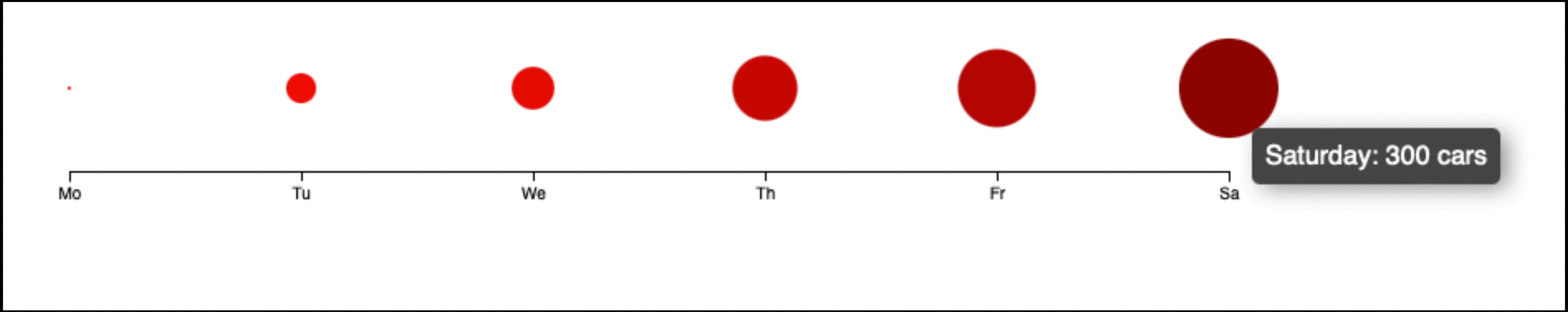
```
d3.select("#scale1")
  .selectAll("circle")
  .data(dataSet)
  .join("circle")
  .on("mouseover", (e, d) =>
    d3.select("#tooltip")
    .style("opacity", 1)
    .text(`${d.day}: ${d.cars} cars`)
  )
  .on("mousemove", (e) =>
    d3
      .select("#tooltip")
      .style("left", e.pageX + 15 + "px")
      .style("top", e.pageY + 15 + "px")
  )
```

You add events by calling d3.on(). D3 will call your event function with two paramenters:

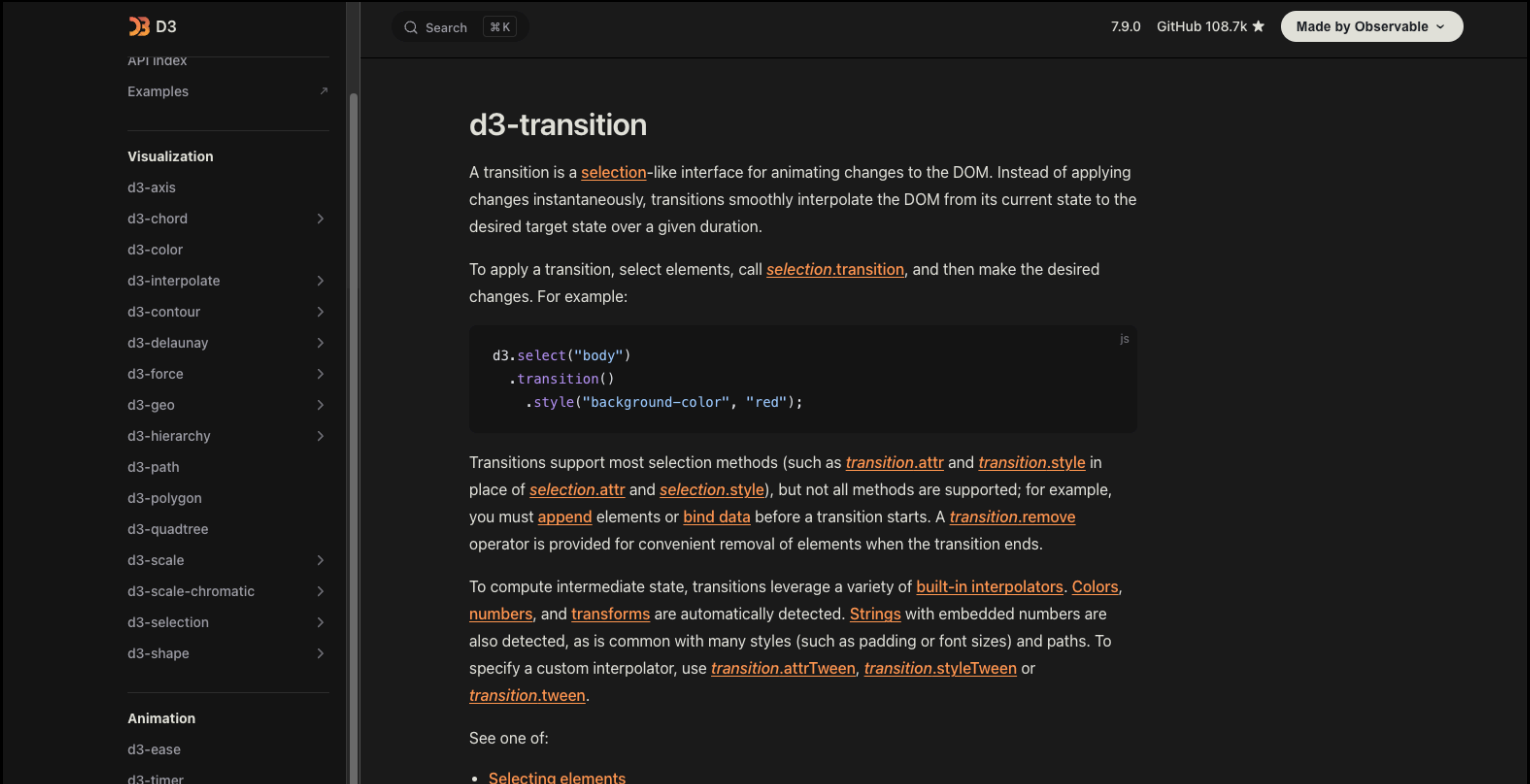
1. Event data
2. Object data used during d3.join()

Events

https://codepen.io/vijnv/pen/RwJKBeO



Events (transitions)



d3-transition

A transition is a selection-like interface for animating changes to the DOM. Instead of applying changes instantaneously, transitions smoothly interpolate the DOM from its current state to the desired target state over a given duration.

To apply a transition, select elements, call *selection*.transition, and then make the desired changes. For example:

```js
d3.select("body")
  .transition()
    .style("background-color", "red");
```

Transitions support most selection methods (such as *transition*.attr and *transition*.style in place of *selection*.attr and *selection*.style), but not all methods are supported; for example, you must append elements or bind data before a transition starts. A *transition*.remove operator is provided for convenient removal of elements when the transition ends.

To compute intermediate state, transitions leverage a variety of built-in interpolators. Colors, numbers, and transforms are automatically detected. Strings with embedded numbers are also detected, as is common with many styles (such as padding or font sizes) and paths. To specify a custom interpolator, use *transition*.attrTween, *transition*.styleTween or *transition*.tween.

See one of:

- Selecting elements

Schedule

1. Data joins (enter, update exit)
2. Dynamic filtering
3. Events
4. **All together!**

```
Uncaught SyntaxError
Unexpected end of input
```