Data Visualization & Design

Week 6

1. Concepts in **Interactive Visualization**
2. Introduction to **D3.js**
3. D3.js **Studio**

1. Concepts in **Interactive Visualization**
2. Introduction to **D3.js**
3. D3.js **Studio**

Filtering

- Analysts rarely analyze the entirety of a dataset at once
- Typically, a presentation is comprised of visualizations that represent selected data dimensions
- Given an overview of dimensions, analysts often want to shift their focus to different subsets (ex. time slices, or particular categories)

Filtering — *Interaction techniques*

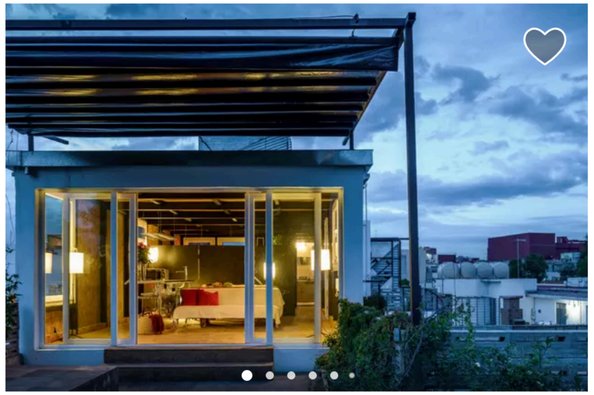
- **User interface elements** (“dynamic query widgets,” dropdowns, radio buttons, checkboxes, etc.)
- Lasso (direct selection)

Anywhere | Anytime | 1 guest

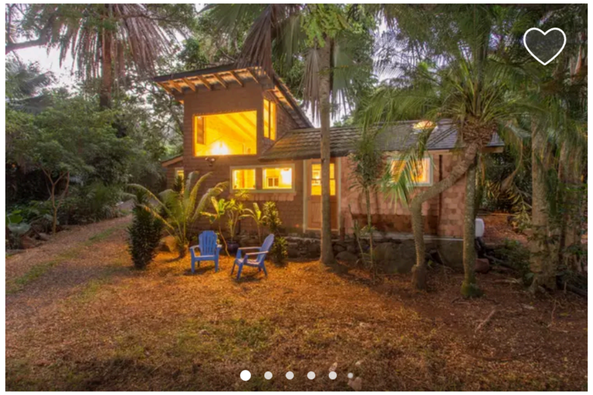
Trips | Messages | Help | Become a host

FOR YOU | HOMES | EXPERIENCES | RESTAURANTS

Room type | Price range | Instant Book | More filters



$84 Apartment 1 of 4 with green terrace in Roma Norte
Entire loft · 2 beds
★★★★★ 265 · Superhost



$175 Adorable Garden Gingerbread House
Entire cabin · 1 bed
★★★★★ 188 · Superhost

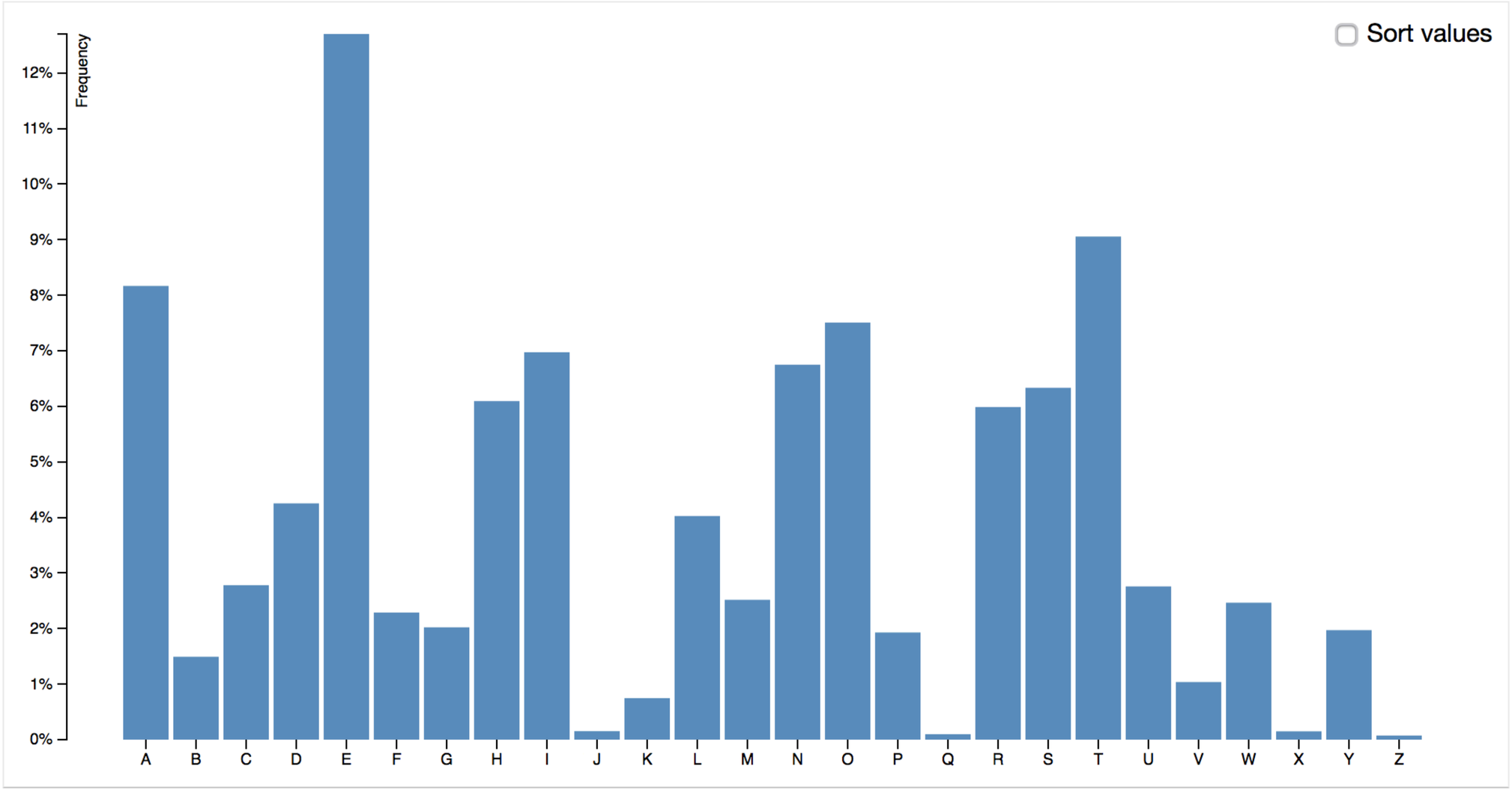
[Example: Airbnb]

Sorting

- Fundamental operation within a visualization
- Enabling the user to **order** values allows for:
    - More easily identifying clusters or trends
    - Reconciling the data within familiar units of analysis (days of the week, financial quarters, etc.)

Sorting — *Interaction techniques*

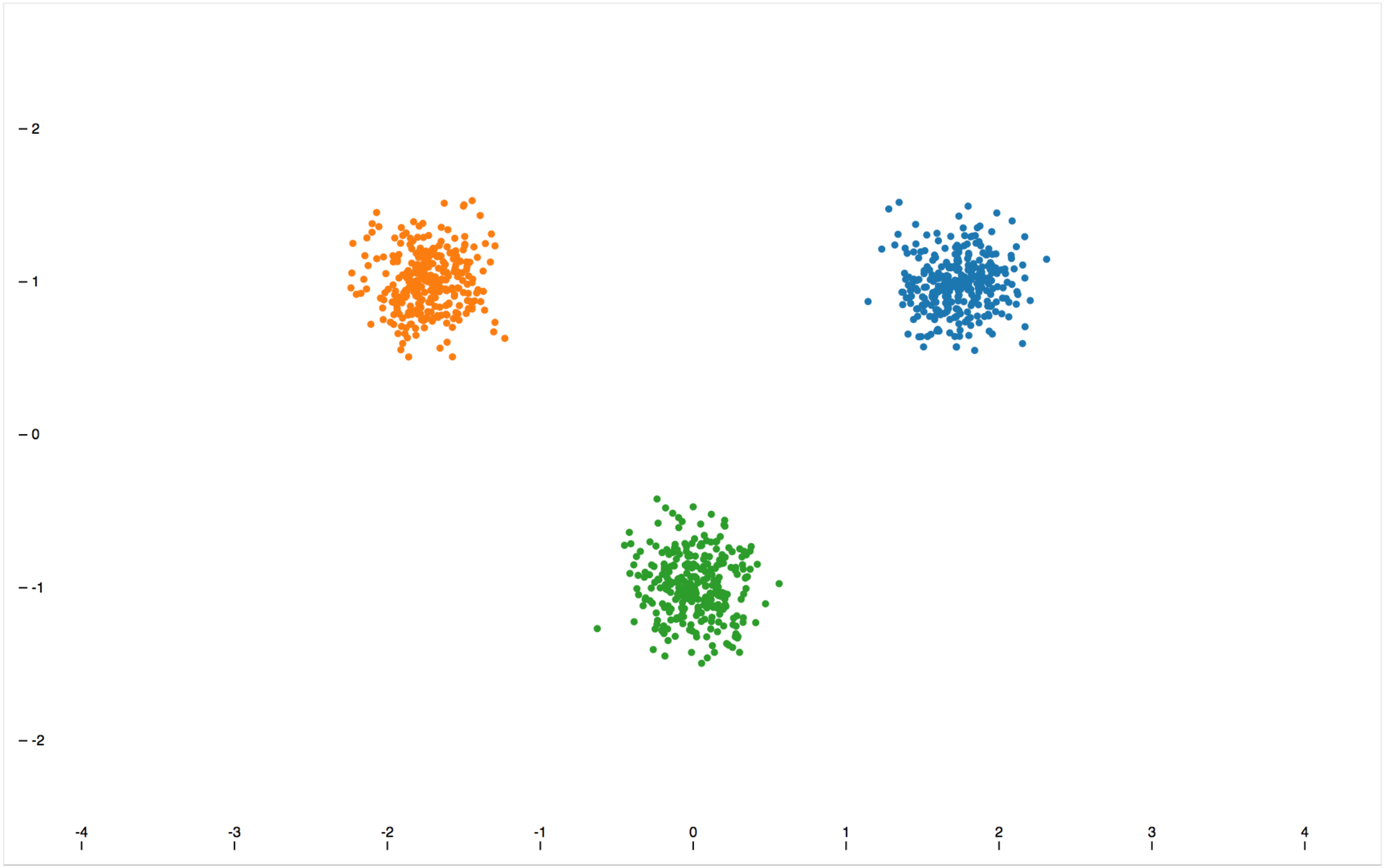
- Choices in a toolbar
- Clicking the header of a table
- User interface elements


Sort values
Frequency
12%
11%
10%
9%
8%
7%
6%
5%
4%
3%
2%
1%
0%
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Example: D3

Brushing and linking

- Address the **connections** between two or more views of the same data
- **Linking —** A change in one view produces a change in the other (i.e. changing filter parameters)
- **Brushing —** Highlighting data to focus a single view or highlight that data in other views



Example: D3

Selection

- Closely related to filtering
- Determines a set of objects to be manipulated or examined further

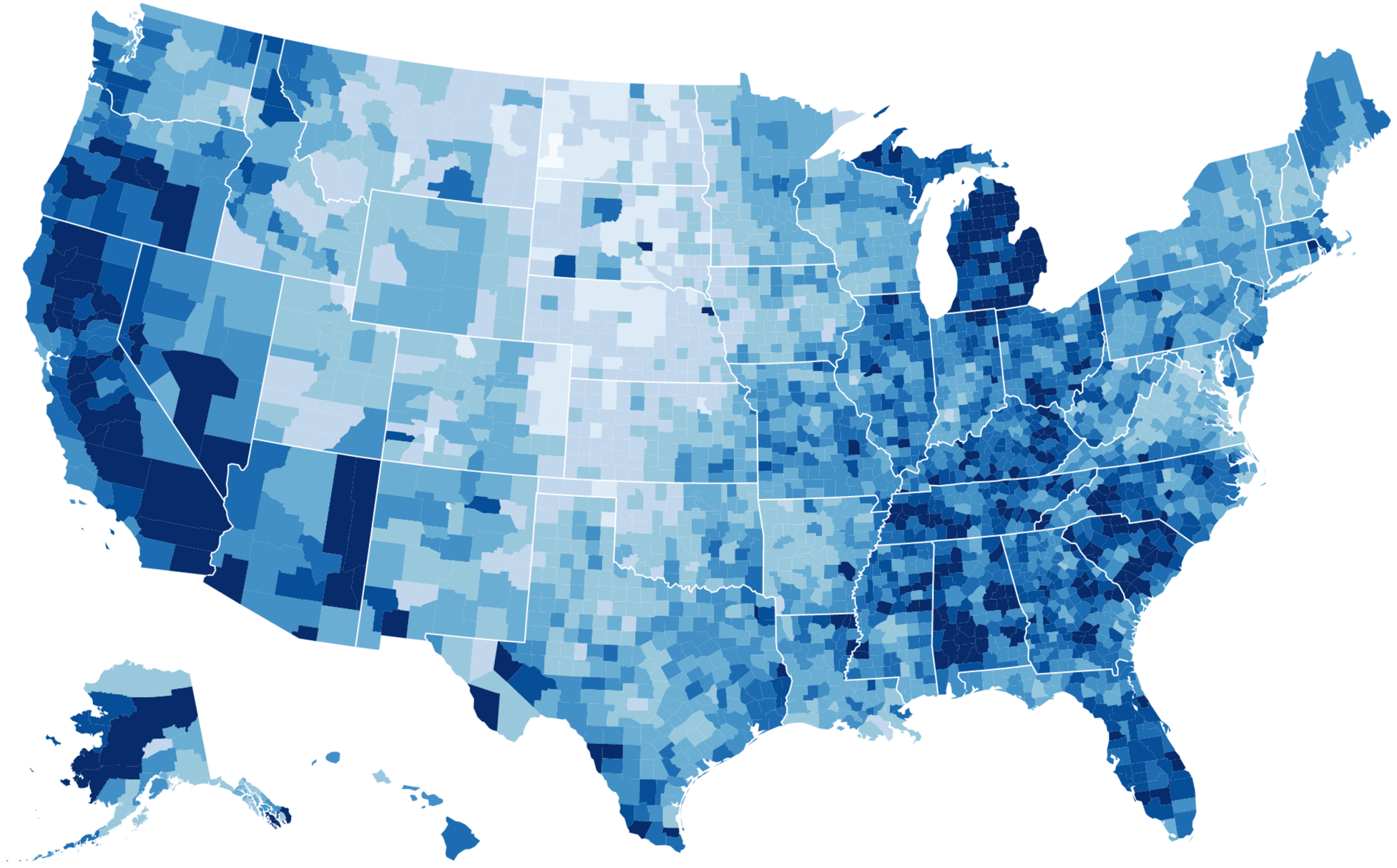
1. Concepts in **Interactive Visualization**
2. Introduction to **D3.js**
3. D3.js **Studio**

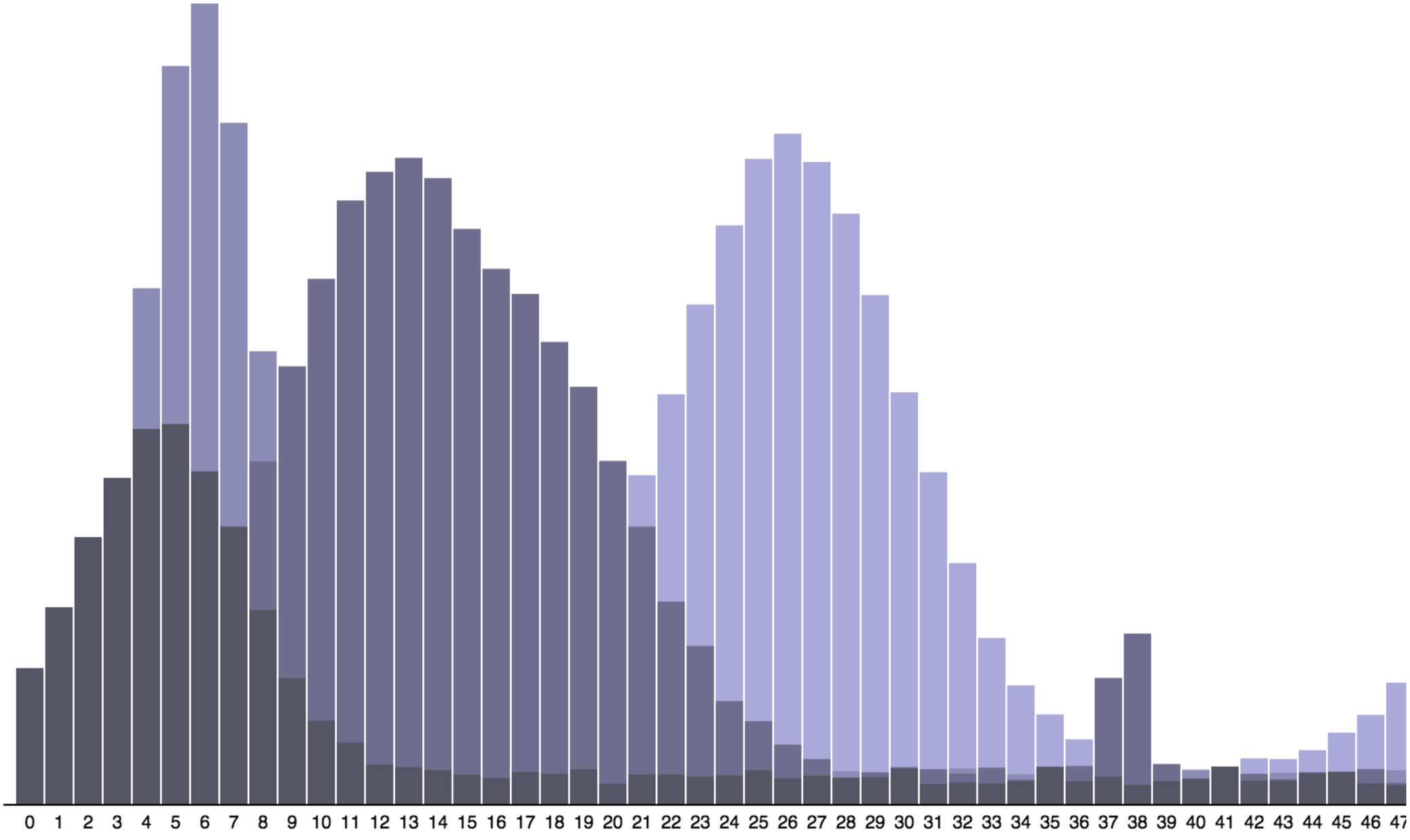
What is D3?

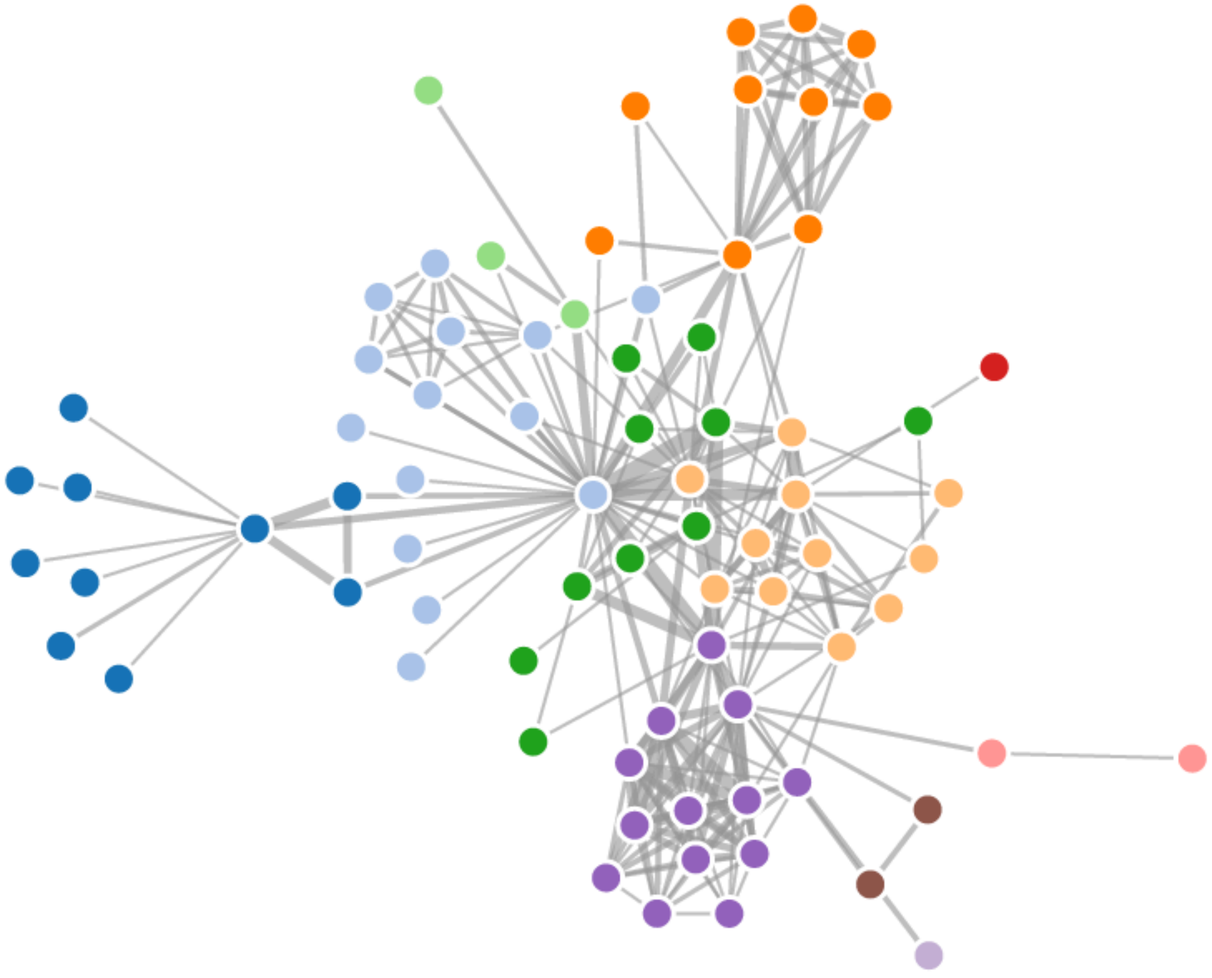
A **general-purpose visualization library** for HTML and SVG.

Efficiently **transform data into elements** in a browser.

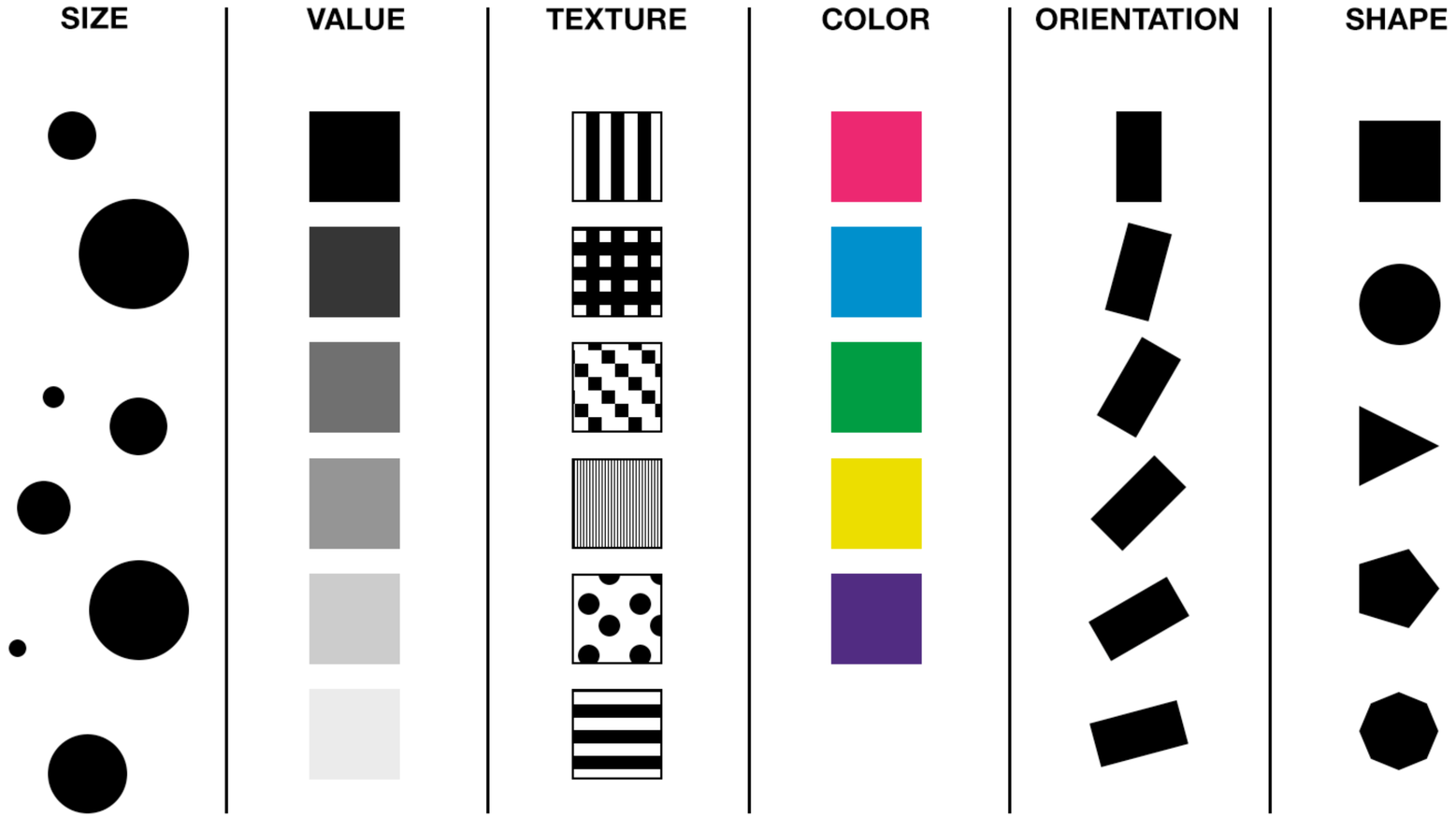
Visualizations are comprised of **basic graphical elements**…




0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47



. . . with **attributes**.



Jacques Bertin: six visual variables

SVG is a **DOM for graphics**.

(Retina-ready, too.)

SVG primitives—

Rectangle

Circle

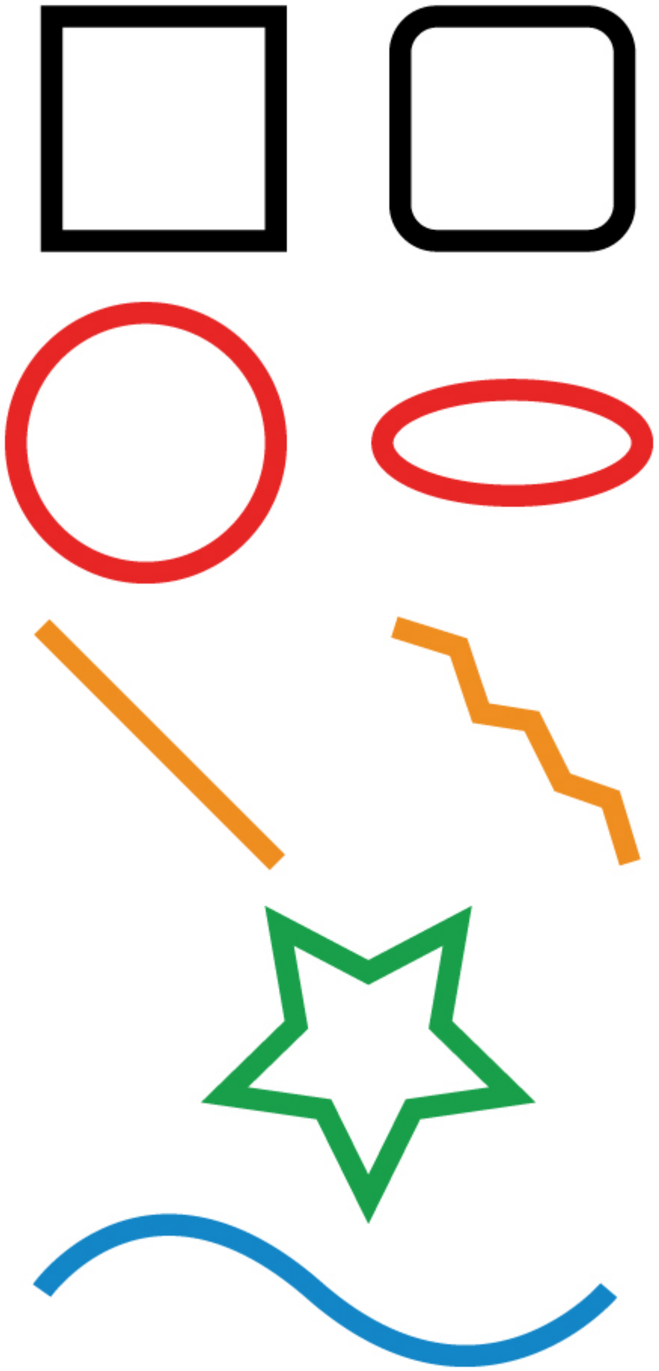
Ellipse

Line

Polyline

Polygon

Path



When is D3 appropriate?

- **Web-based** visualizations
- **Highly custom** visualizations (i.e. cannot be replicated by a charting library)
- Visualizations that need to **update frequently**

JavaScript, data types, & JSON

JavaScript (JS) is a **programming language** that enables you to add interactive features to your website.

HTML skeleton, CSS skin, JS brain.

String

Number

Boolean

Array

Object

String

A series of characters enclosed in double- or single-quotes.

```
var x = "hello”
var x = "34"
```

Number

Integer or float.

```
var x = 34
var x = 34.00
```

Boolean

True or false.

```
var x = true
var x = false
```

Array

Square brackets. Items separated by commas.

```
var x = [1, 2, 3]
var x = ["this", "that"]
```

Object

Curly brackets. Properties are written as name/value pairs, separated by commas.

```
var x = {
  "name":"bob",
  "age":35,
  "location":"NY"
}
```

JSON is a syntax for **data storage and exchange**.

JSON syntax is **derived** from JavaScript syntax.

JSON data is written in **name/value pairs**.

```
"city":"new york"
```

JSON **objects** are stored inside **curly brackets**...

```
{"city":"new york"}
```

…which can contain multiple name/value pairs.

```
{"city":"new york", "state":"NY"}
```

JSON **arrays** are written inside **square brackets**.

```
"locations":[
   {"city":"new york", ”state":"NY"},
   {"city":"los angeles", ”state":"CA"},
   {"city":"chicago", ”state":"IL"}
]
```

To work with JSON in JS, create an array and assign data to it.

```
var locations = [
   {"city":"new york", ”state":"NY"},
   {"city":"los angeles", ”state":"CA"},
   {"city":"chicago", ”state":"IL"}
];
```

In **Excel**, that data would look like this.

	A	B	C
1	**city**	**state**	
2	new york	NY	
3	los angeles	CA	
4	chicago	IL	
5			
6			
7			
8			
9			

Elements in an array can be accessed using their **position** (index).

```
locations[0]
{"city":"new york", ”state":"NY"}

locations[2]
{"city":"chicago", ”state":"IL”}
```

D3: A **closer look**

What makes D3.js **unique**?

- Pairs a **data object with an element**
- Keeps track of **new and old objects**
- Lets you **animate** differences between new and old

Pie chart example:

http://bl.ocks.org/dbuezas/9572040

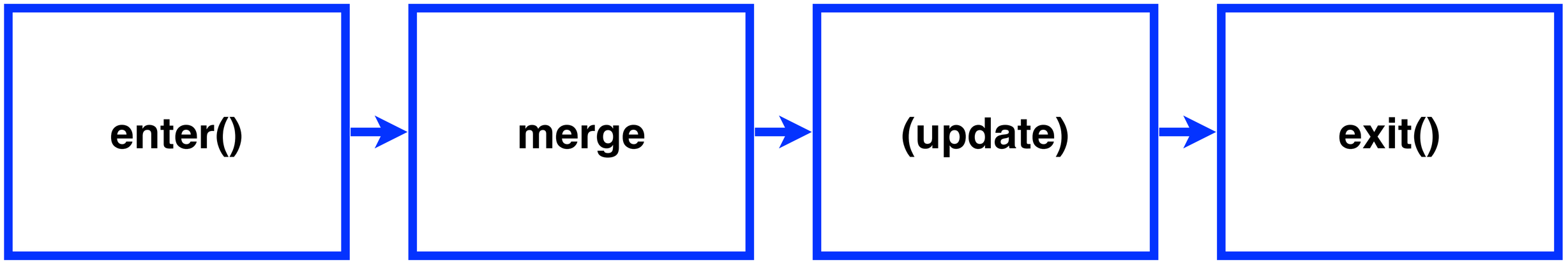
enter, update, exit

Our *data*:

```
[1,2,3]
```

Our *goal*:

Draw *one circle* for every data point.



Create elements to correspond to data

Combine old and new elements

For each element in selection, update attributes

Clean up unneeded elements

Start with a **selection**.

```
//this is an empty selection
//looks for instantiations of data

var myCircles = d3.selectAll('circle')
```

Join selected elements with data items.

```
var myCircles = d3.selectAll('circle')
   .data([1,2,3]);
```

enter()

For every part of the data that does not correspond to an existing element, add an element.

```
myCircles.enter().append('circle')
```

merge

Combine the enter and update groups.

```
.merge(myCircles)
```

(update)

For each element in the selection,

update attributes.

```
.attr('fill','red');
```

exit()

Remove elements that no longer correspond to the data.

```
myCircles.exit().remove();
```

→ data bound to the DOM.

A closer look at **update()**…

```
// d3 has a few different
// functions that set stuff

.text()
.property()
.style()
.attr()
```

```
// each takes a function

.attr('foo', function() { })
```

```
// and that function gets data
// from your .data()

.attr('foo', function(d) {
  return d.foo;
})
```

You can **chain** these, as well as use them to **pull an existing value**.

1. Concepts in **Interactive Visualization**
2. Introduction to **D3.js**
3. D3.js **Studio**

Let’s draw some shapes!

https://**beta.observablehq.com/**

@emilyfuhrman/d3-js-template

This row defines the **width** of your drawing space.

```
width
800
width = 800
```

This row defines the **height** of the drawing space.

```
height 200
height = 200
```

This row **includes the D3.js library**.

```
undefined
```

```
d3 = require("https://d3js.org/d3.v5.min.js")
```

SVG circles have a **center point** position
and a **radius**.

`(cx, cy, r)`

```
//full code

var my_circles = svg.selectAll("circle")
   .data([10,20,30]);
my_circles.enter()
   .append("circle")
   .merge(my_circles)
      .attr("cx",500)
      .attr("cy",100)
      .attr("r",function(d){
         return d;
      });
my_circles.exit().remove();
```

```
//create an empty selection
//this looks for instantiations of data

var my_circles = svg.selectAll("circle")
```

```
//this is data, which
//would be bound to a selection

var my_circles = svg.selectAll("circle")
   .data([10,20,30]);
```

```
//ENTER: for every time we see data,
//but do not see a corresponding element

var my_circles = svg.selectAll("circle")
   .data([10,20,30]);
my_circles.enter()
```

```
//append an element

var my_circles = svg.selectAll("circle")
   .data([10,20,30]);
my_circles.enter()
   .append("circle")
```

```
//merge enter and update selections

var my_circles = svg.selectAll("circle")
   .data([10,20,30]);
my_circles.enter()
   .append("circle")
   .merge(my_circles)
```

```
//set x-position and y-position

var my_circles = svg.selectAll("circle")
   .data([10,20,30]);
my_circles.enter()
   .append("circle")
   .merge(my_circles)
      .attr("cx",500)
      .attr("cy",100)
```

```
//finally, set circle radius
//based on value from the array

var my_circles = svg.selectAll("circle")
   .data([10,20,30]);
my_circles.enter()
   .append("circle")
   .merge(my_circles)
      .attr("cx",500)
      .attr("cy",100)
      .attr("r",function(d){
         return d;
      });
```

```
//and clean everything up

var my_circles = svg.selectAll("circle")
   .data([10,20,30]);
my_circles.enter()
   .append("circle")
   .merge(my_circles)
      .attr("cx",500)
      .attr("cy",100)
      .attr("r",function(d){
         return d;
      });
my_circles.exit().remove();
```

```
//full code

var my_circles = svg.selectAll("circle")
   .data([10,20,30]);
my_circles.enter()
   .append("circle")
   .merge(my_circles)
      .attr("cx",500)
      .attr("cy",100)
      .attr("r",function(d){
         return d;
      });
my_circles.exit().remove();
```

Only one?

```
//add a function to ‘cx’

var my_circles = svg.selectAll("circle")
    .data([10,20,30]);
my_circles.enter()
    .append("circle")
    .merge(my_circles)
        .attr(“cx”,function(d,i){
          return i*300;
        })
        .attr("cy",100)
        .attr("r",function(d){
           return d;
        });
my_circles.exit().remove();
```

```
//tweak it

var my_circles = svg.selectAll("circle")
    .data([10,20,30]);
my_circles.enter()
    .append("circle")
    .merge(my_circles)
        .attr("cx",function(d,i){
          return i*300 + 25;
        })
        .attr("cy",100)
        .attr("r",function(d){
           return d;
        });
my_circles.exit().remove();
```

Challenges

Challenge—

Give the circles a red stroke.

Challenge—

Make the data drive the stroke-width of the circles.

Challenge—

Make the circles rectangles.

SVG rectangles have an x-position, a y-position, width, and height.

`(x, y, width, height)`

Challenge—

Make the dataset into an array of objects to set the radii and color of the circles.