

Information Visualization Frameworks

SWE 437, Fall 2019

Design and Implementation of Software for the Web

Quiz

Go to:

b.socrative.com, Click student login

Room name: SWE432

Student Name: Your G-number (Including the G)

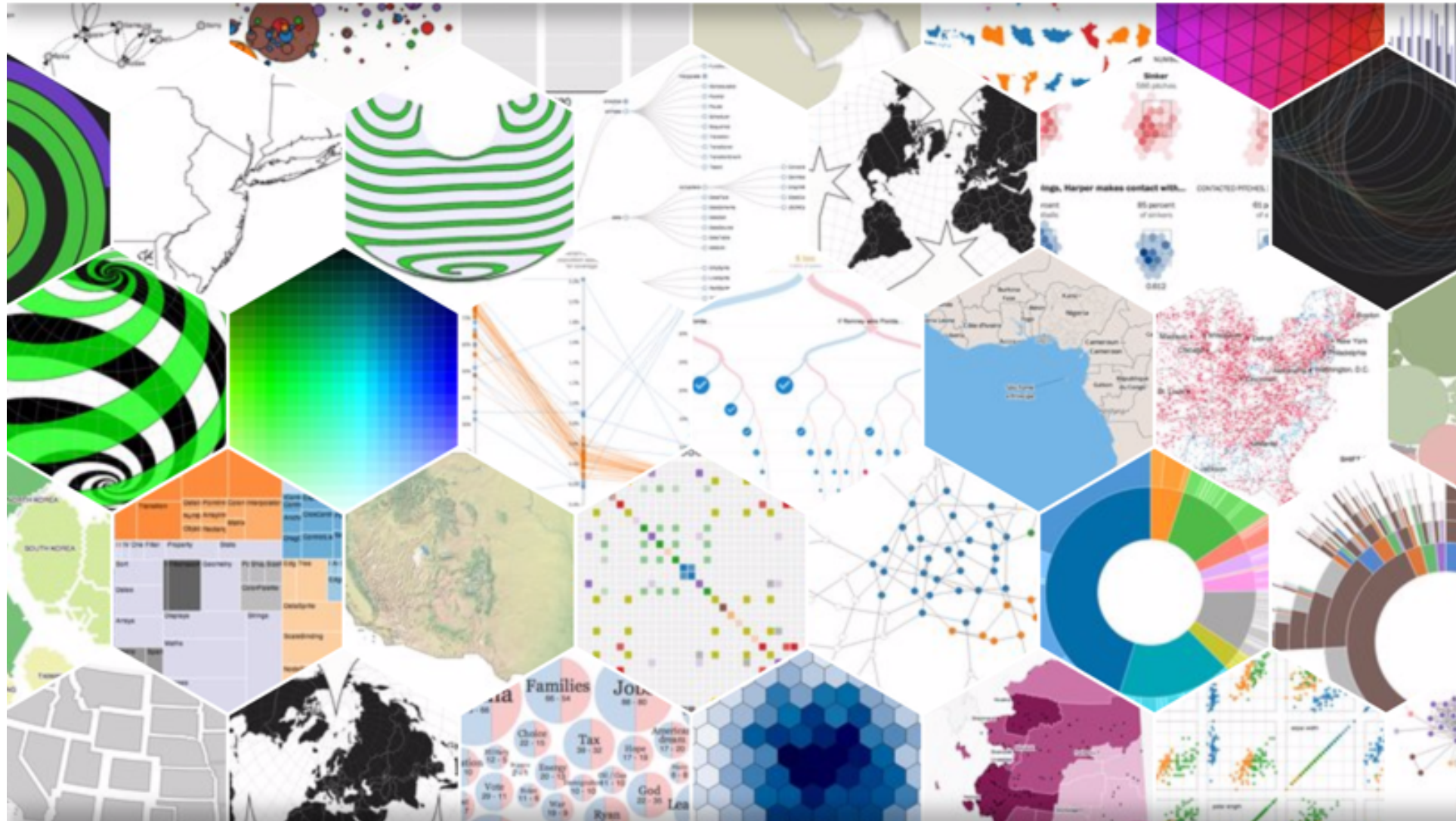
Reminder: Survey can only be completed if you are in class. If you are not in class and do it you will be referred directly to the honor code board, no questions asked, no warning.

Today

- How do information visualization frameworks help build information visualizations?
- How can you build an information visualization with d3.js?
- How is binding data to elements similar but also different than in React?

Some challenges in information visualization frameworks

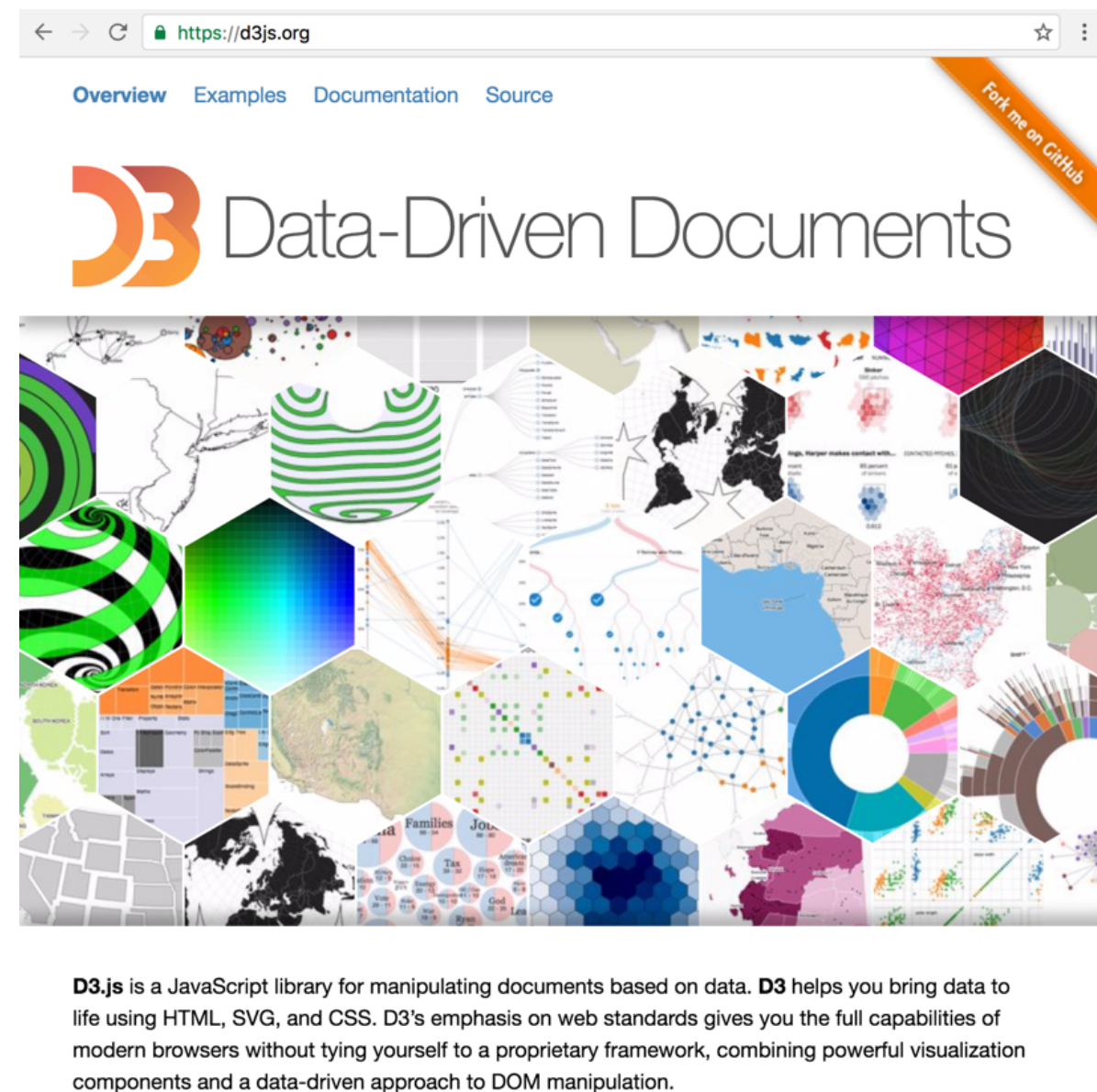
- Data binding
 - You have data. How do you create corresponding visual elements?
 - How do you update the visual elements if the data changes?
 - Or the user updates what they want to see...
- Scales
 - How do data values correspond to position, size, color, etc. of visual elements?
- Transitions
 - How do you smoothly animate changes between visual states?



D3.js is a JavaScript library for manipulating documents based on data. **D3** helps you bring data to life using HTML, SVG, and CSS. D3's emphasis on web standards gives you the full capabilities of modern browsers without tying yourself to a proprietary framework, combining powerful visualization components and a data-driven approach to DOM manipulation.

D3.js

- Most popular information visualization framework for the web
- Designed by Mike Bostock as part of his PhD
- Transform data into a visual representation
 - e.g., build HTML elements for elements in an array
- Based on web standards, including HTML, CSS, SVG

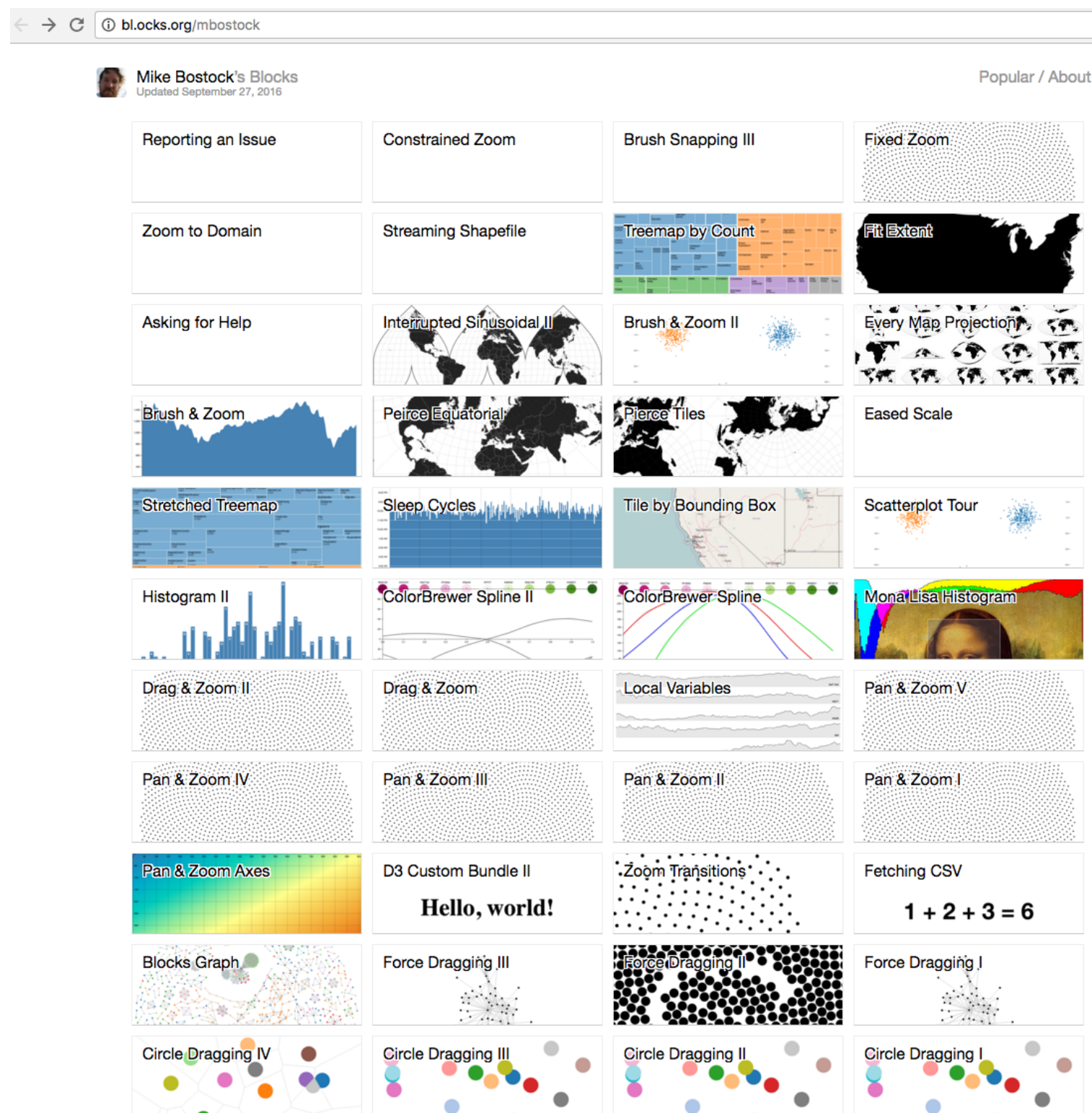


Installing D3.js

```
npm install d3
```

Learning D3

- Many tutorials
- Many, **many** examples
- Frequent pattern: copy similar visualization, customize for your needs
- But... be careful you use d3 v4
- Current version



Key concepts we'll cover today

- Selections
- Dynamic properties
- Data joins (a.k.a. data binding)
- Scales
- SVG
- Loading data

Selections

```
var paragraphs = document.getElementsByTagName("p");  
for (var i = 0; i < paragraphs.length; i++) {  
    var paragraph = paragraphs.item(i);  
    paragraph.style.setProperty("color", "white", null);  
}
```

==

```
d3.selectAll("p").style("color", "white");
```

Dynamic properties

```
<p>P1</p>  
<p>P2</p>  
<p>P3</p>  
<p>P4</p>
```

```
d3.selectAll("p").style("color", function() {  
  return "hsl(" + Math.random() * 360 + ",100%,50%)";  
});
```

P1

P2

P3

P4

Dynamic properties

```
<p>P1</p>  
<p>P2</p>  
<p>P3</p>  
<p>P4</p>
```

```
d3.selectAll("p").style("color", function(data, index) {  
  return index % 2 ? "black" : "gray";  
});
```

P1
P2
P3
P4

Dynamic properties

```
<p>P1</p>  
<p>P2</p>  
<p>P3</p>  
<p>P4</p>
```

P1

P3

P4

```
d3.selectAll("p")  
  .style("font-size", function(d) { return Math.random() * 50 + "px"; });
```


Styling elements

- `selection.attr` - get or set an attribute.
- `selection.classed` - get, add or remove CSS classes.
- `selection.style` - get or set a style property.
- `selection.property` - get or set a (raw) property.
- `selection.text` - get or set the text content.
- `selection.html` - get or set the inner HTML.

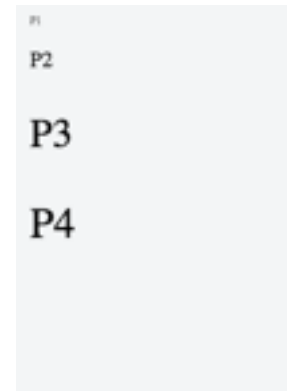
Data binding

- We can style elements dynamically based on data.
- But...
 - usually we have a dataset (e.g., time-series data of temperature readings)
 - and we want to directly associate it with some visual elements
 - and it'd be great if we could automatically create elements based on the data.
 - and delete or update the visual elements when the data changes.

Data binding

```
<p>P1</p>  
<p>P2</p>  
<p>P3</p>  
<p>P4</p>
```

```
d3.selectAll("p")  
  .data([4, 8, 15, 16, 23, 42])  
  .style("font-size", function(d) { return d + "px"; });
```



P1
P2
P3
P4

- Bind *data* with visual element.

Data binding is persistent

```
<p>P1</p>  
<p>P2</p>  
<p>P3</p>  
<p>P4</p>
```

```
var p = d3.selectAll("p")  
    .data([4, 8, 15, 16, 23, 42])  
    .style("font-size", function(d) { return d + "px"; });  
  
p.style("color", "blue");
```

P1
P2
P3
P4

- D3 uses cascade pattern, returning element set.
- By default, visual elements persist once created.
- Can update style without binding to data again

How do we deal with
changing data?

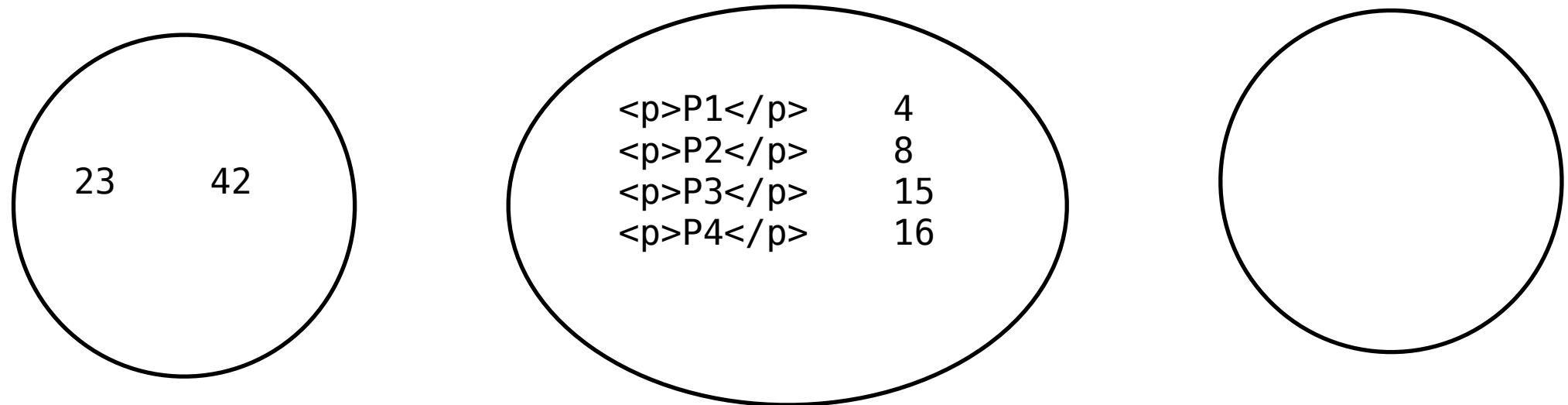
Handling Changing Data

- React:
 - Make components, bind components to state, update state
- D3:
 - Need to provide more control to rendering
 - E.g.: What if I want to highlight data that is new?

Thinking in Joins

<p>P1</p>
<p>P2</p>
<p>P3</p>
<p>P4</p>

```
var p = d3.selectAll("p")  
    .data([4, 8, 15, 16, 23, 42])
```



`.enter(...)`

`// update (default)`

`.exit(...)`

Stuff not on left

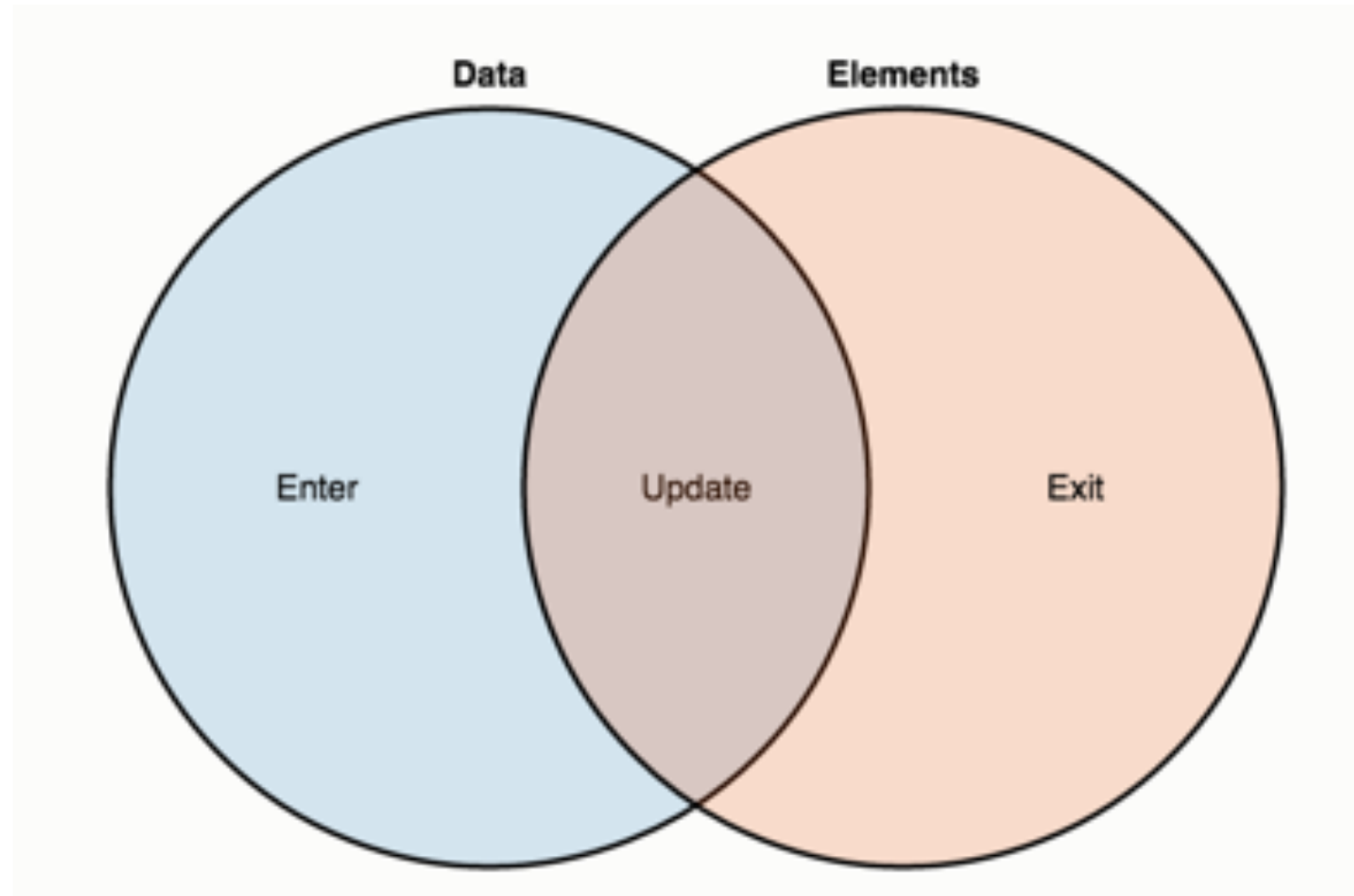
Stuff in both

Stuff not on right

- Elements in selection set undergo data join with elements in data

<https://bost.ocks.org/mike/join/>

Thinking in joins

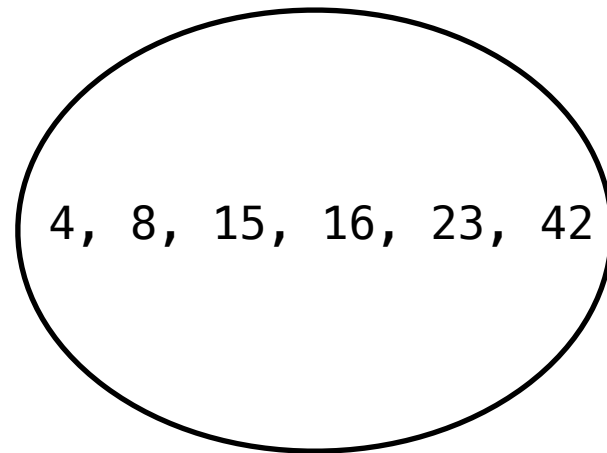


- Extra data $\rightarrow$ enter set
- Matched data with elements $\rightarrow$ update set
- Extra elements $\rightarrow$ exit set

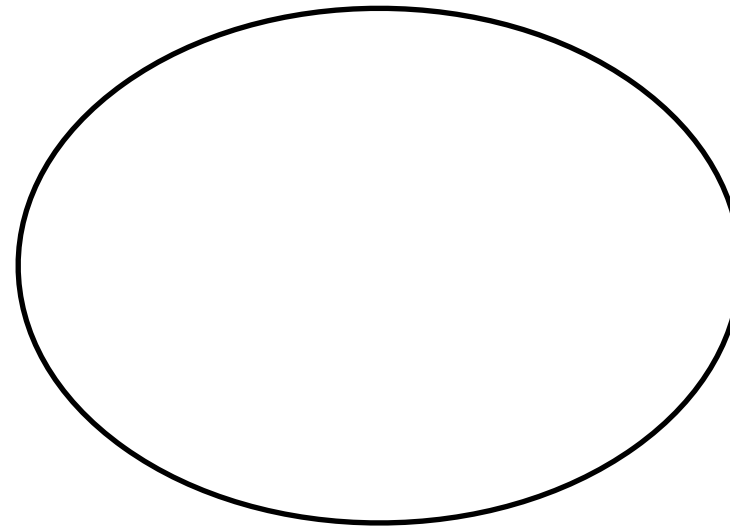
Thinking in Joins

<!-- no p elements -->

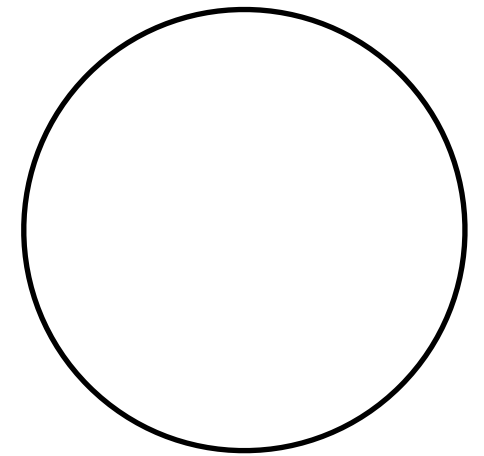
```
var p = d3.selectAll("p")  
  .data([4, 8, 15, 16, 23, 42])
```



.enter(...)



// update (default)

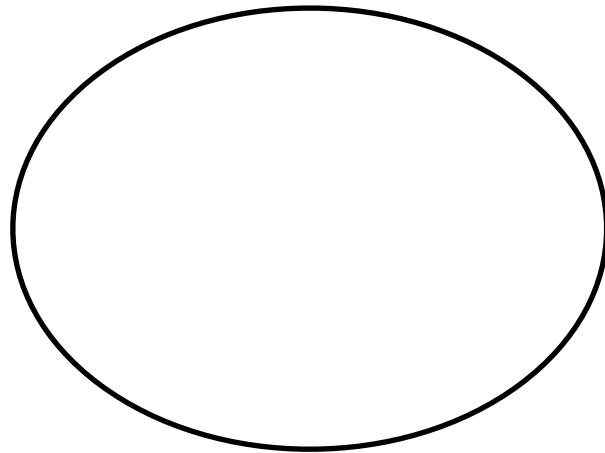


.exit(...)

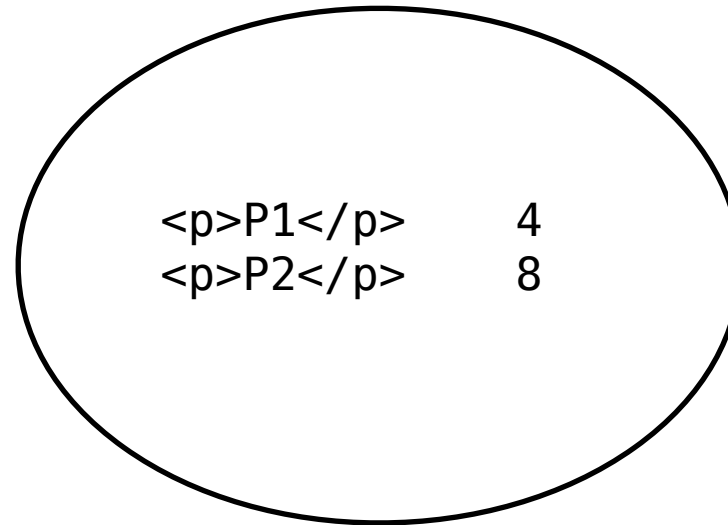
Thinking in Joins

```
<p>P1</p>  
<p>P2</p>  
<p>P3</p>  
<p>P4</p>
```

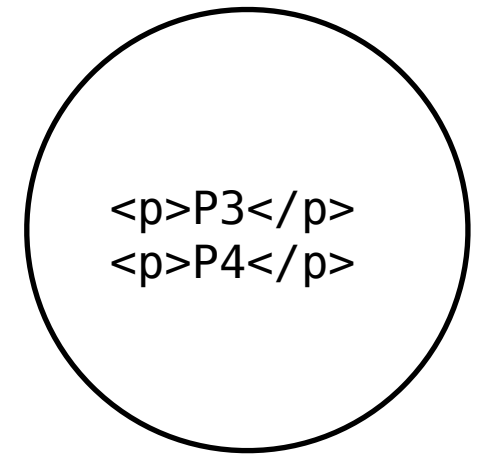
```
var p = d3.selectAll("p")  
    .data([4, 8]);
```



`.enter(...)`



`// update (default)`



`.exit(...)`

Creating elements

- We really don't want to hardcode html elements.
- How can we use data joins to create these automatically?

```
d3.select("body")
  .selectAll("p")
  .data([4, 8, 15, 16, 23, 42])
  .enter().append("p")
    .text(function(d) { return "I'm number " + d + "!"; });
```

- Selects data that are not yet bound to an element using enter
- Creates elements using append
- Sets text property using text

Creating elements

```
d3.select("body")
  .selectAll("p")
  .data([4, 8, 15, 16, 23, 42])
  .enter().append("p")
    .text(function(d) { return "I'm number " + d + "!"; });
```

- Note that we have to first select elements that do not exist!
- `selectAll("p")`
- Need this to specify what will eventually exist in future

Putting it together

```
// Update...
var p = d3.select("body")
    .selectAll("p")
    .data([4, 8, 15, 16, 23, 42])
    .text(function(d) { return d; });

// Enter...
p.enter().append("p")
    .text(function(d) { return d; });

// Exit...
p.exit().remove();
```

- Common pattern on data change is to rebind data to elements and separately handle
 - existing elements that should have new visual style (update)
 - new elements that should be created
 - existing elements that should be deleted

Demo: Really Simple Bar Chart

<http://jsbin.com/pivupuheta/edit?css,js,output>

Loading data

- What is data?
 - Anything that is an array
 - `.data()` just cares that it is an array of elements
 - Could be array of numbers, strings, JSON objects
 - If you have a dataset that is an array of JSON objects, pass it to `data` and you are done

```
.data([ { "a": 5 }, { "a": 3 }, { "a": 7 } ])  
  .text(function(d) { return d.a - 1; });
```


Scaling to fit data

```
.style("width", function(d) { return d * 10 + "px"; });
```

- 10 is a magic number
 - Transforms number in data scale to number in visual representation (“range”) scale
 - Every “1” unit in data should correspond to some unit in output coordinate system
- We’d like to automatically generate reasonable sizes, sizing data to take up all of the space based on range of data, etc.

Scales

```
var x = d3.scale.linear()  
    .domain([0, d3.max(data)])  
    .range([0, 420]);
```

$x(4) = 40;$
 $x(2) = 20;$

- Different types of scales that map domain values (data space) to range values (display space)
- Linear scale uses linear function (e.g., $ax + b$) to create range value from domain value
- Use:
 - Specify min and max of data
 - Specify min and max of range (output)
 - Generates a function (e.g., x) that will compute range value for a domain value

Shapes and paths

- We can use HTML boxes if all we care about is shapes that are rectangular (or almost rectangular)
- But what about a visualization with a line? Or a curve? Or a complex path?
- We need a new way to specify complex shapes!

SVG: Scalable Vector Graphics

- W3C standard adopted in 1999
- HTML for specifying visual shapes
 - Natively supported by browsers
- Just like HTML
 - Create it using a `<svg></svg>` tag
 - Shows up in DOM like normal DOM elements
 - Can be styled with css (but different property names...)
- Not like HTML
 - Elements inside *always* positioned relative to top left of container
 - Creates a coordinate system for elements within container

<https://developer.mozilla.org/en-US/docs/Web/SVG>

SVG: Example

```
<svg class="chart" width="420" height="120">
  <g transform="translate(0,0)">
    <rect width="36" height="19"></rect>
    <text x="37" y="9.5" dy=".35em">4</text>
  </g>
</svg>
```

- g: container element, like div
 - Enables specifying new coordinate system (i.e., where to start drawing)
- Rect: rectangle element
- Text: text element

Demo: Static SVG Bar Chart

<http://jsbin.com/xipexatodu/edit?html,css,output>

Demo: Generated SVG Bar Chart

<http://jsbin.com/baqeyovaho/edit?html,js,output>

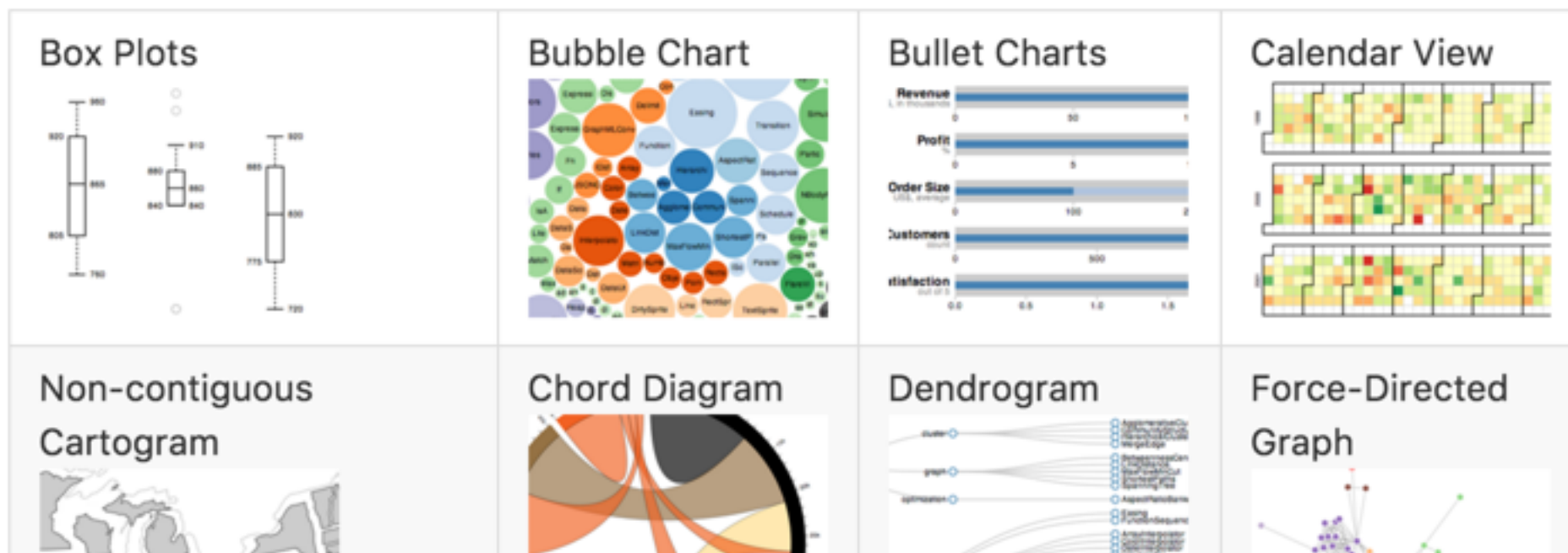
Using D3

- Best place to start
 - Example code of similar visualization
 - Don't need to understand *everything*, just enough to make it work

<https://github.com/d3/d3/wiki/Gallery>

 GitHub, Inc. [US] | <https://github.com/d3/d3/wiki/Gallery>

Visual Index



In Class Activity

- Goto D3 visualization gallery
- Find an information visualization
- Tinker with it, editing it to visualize data in a different way