

D3 Introduction

Gracie Young and Vera Lin

Slides adapted from

Maneesh Agrawala

Jessica Hullman

Ludwig Schubert

Peter Washington

Alec Glassford and Zach Maurer

CS 448B: Visualization

Fall 2018

Topics

- 1) What is D3? Why do we use it?
- 2) D3 as DOM manipulation.
- 3) D3 as DataViz tool.
- 4) Running example.
 - Scales, Axes, Coordinates, Marks
 - Filtering, Updating, Interactivity

[Adapted from Mike Bostock's D3 Workshop]

What is D3?

Why do we use it?

Visualization with Web Standards

D3: "Data-Driven Documents"

Data visualization built on top of HTML, CSS, JavaScript, and SVG

Pros:

- Highly-customizable

- Developing and debugging tools

- Documentation, resources, community

- Integrates with the web!

Cons:

- Very "low-level".

What does D3 look like?

i.e. What does HTML, CSS, JS look like?

hello-world.html

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
  <meta charset="utf-8">
```

```
</head>
```

```
<!-- Your html spec here -->
```

```
</html>
```

hello-world.html

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
  <meta charset="utf-8">
```

```
</head>
```

```
<body>
```

```
  Hello, world!
```

```
</body>
```

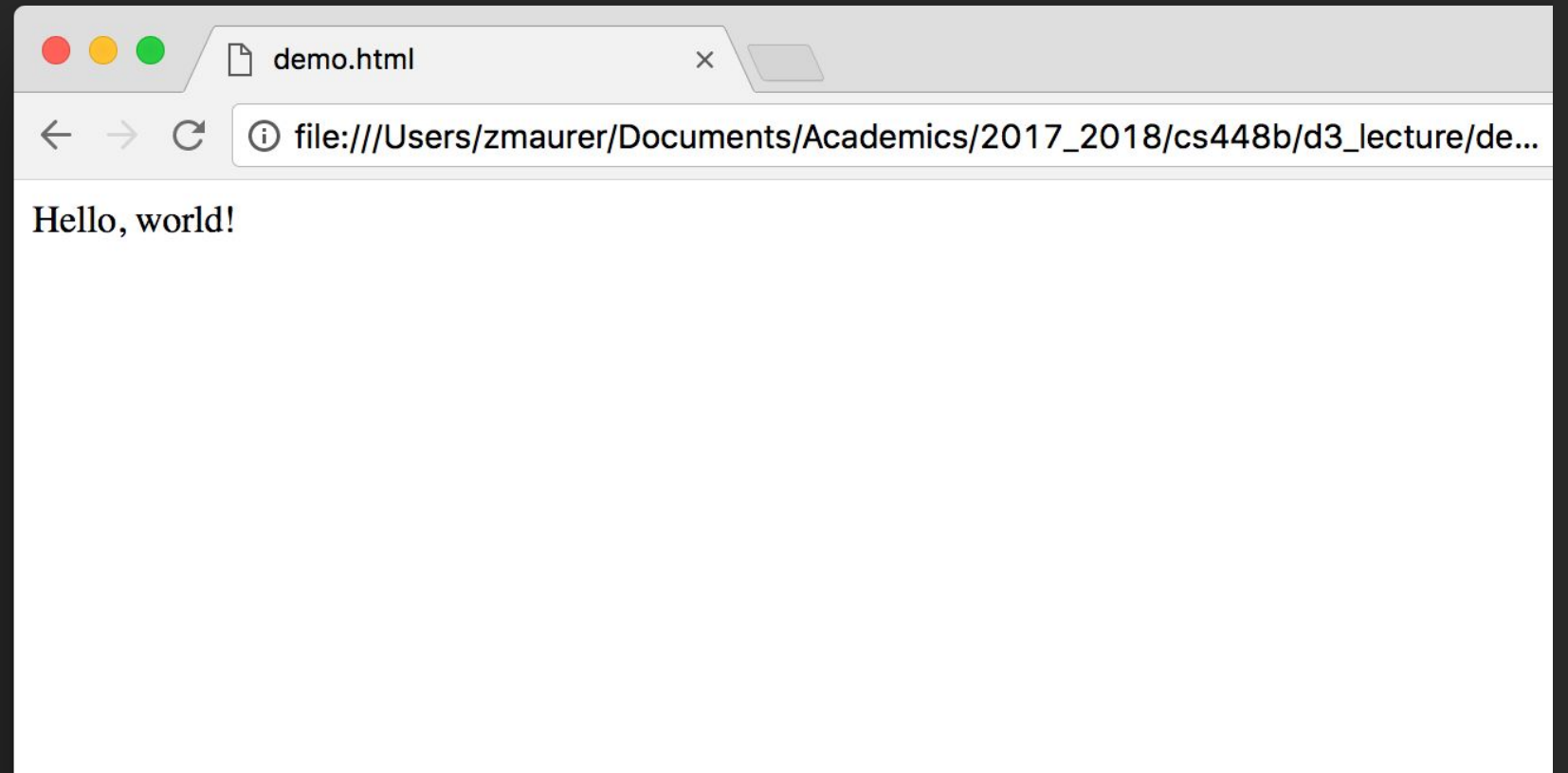
```
</html>
```

hello-world.html

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
</head>

<body>
  Hello, world!
</body>

</html>
```



hello-css.html

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">

  <style>
    body { background: steelblue; }
  </style>

</head>

<body>
  Hello, world!
</body>

</html>
```

hello-css.html

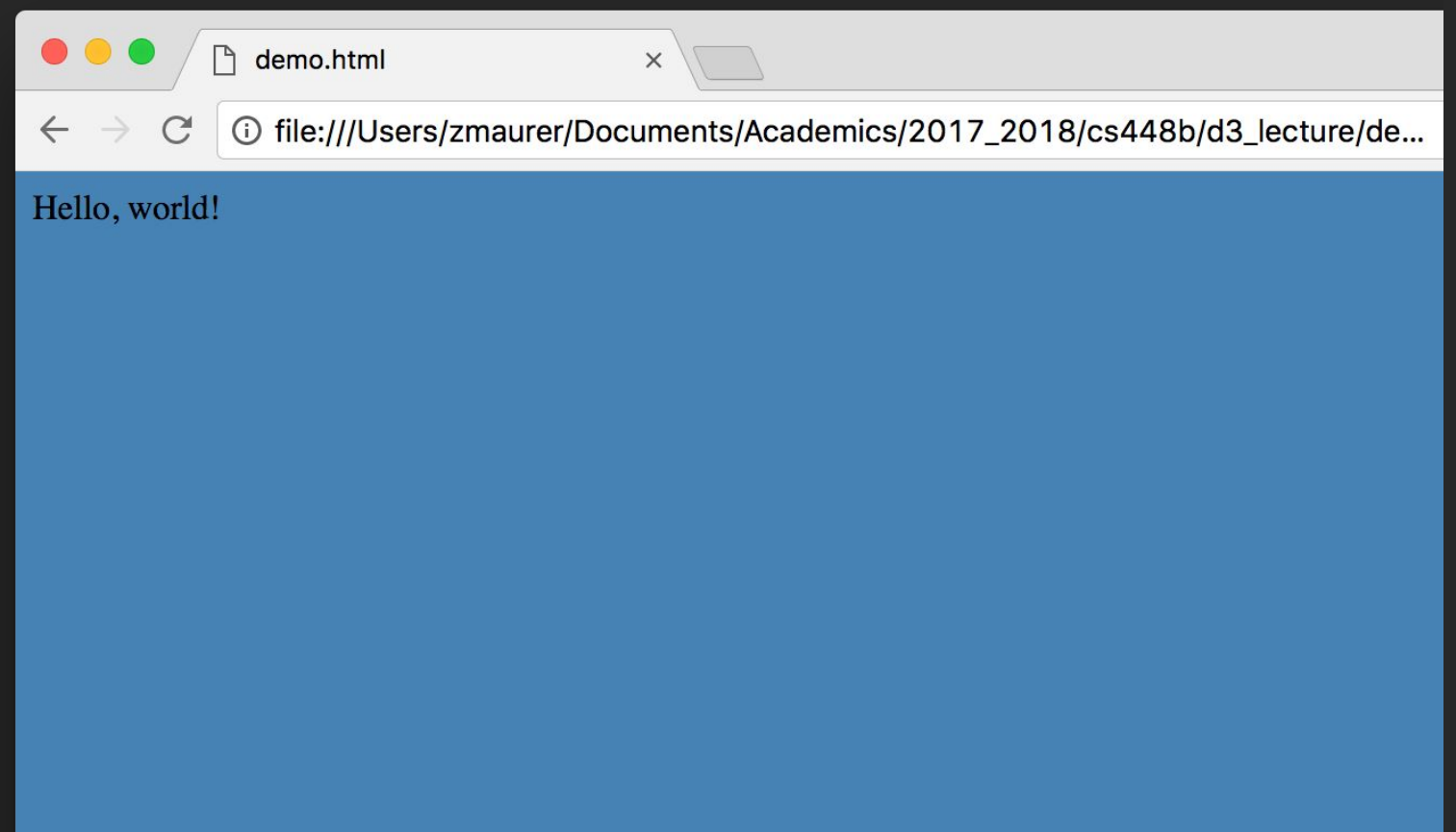
```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">

  <style>
    body { background: steelblue; }
  </style>

</head>

<body>
  Hello, world!
</body>

</html>
```



hello-svg.html

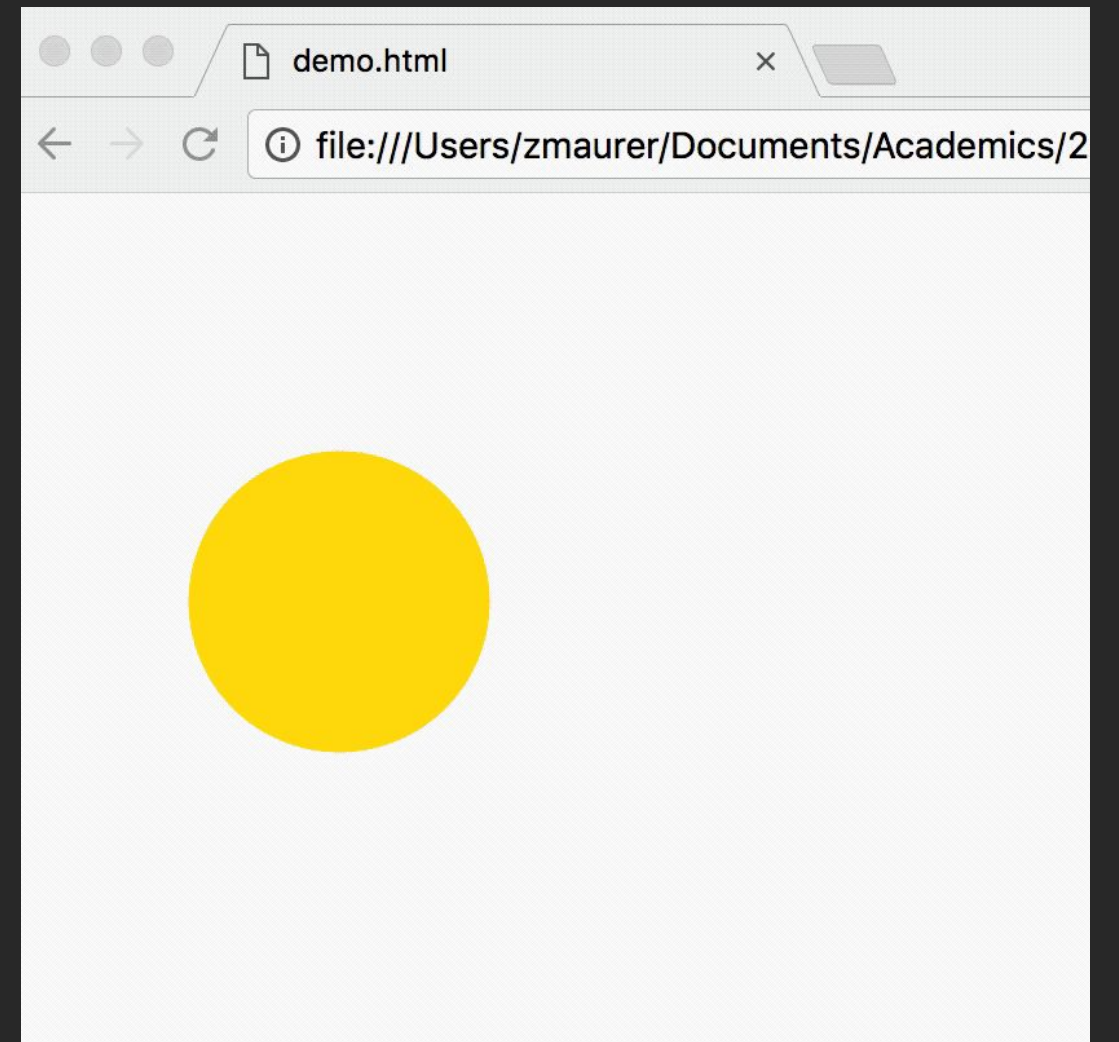
```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <style> /* CSS */ </style>
</head>

<body>
  <svg width="960" height="500">
    <circle cx='120' cy='150' r='60' style='fill: gold;'>
      <animate
        attributeName='r'
        from='2' to='80' begin='0' dur='3'
        repeatCount='indefinite' />
    </circle>
  </svg>
</body>
</html>
```

hello-svg.html

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <style> /* CSS */ </style>
</head>

<body>
  <svg width="960" height="500">
    <circle cx='120' cy='150' r='60' style='fill: gold;'>
      <animate attributeName='r'
        from='2' to='80' begin='0' dur='3'
        repeatCount='indefinite' />
    </circle>
  </svg>
</body>
</html>
```



hello-javascript.html

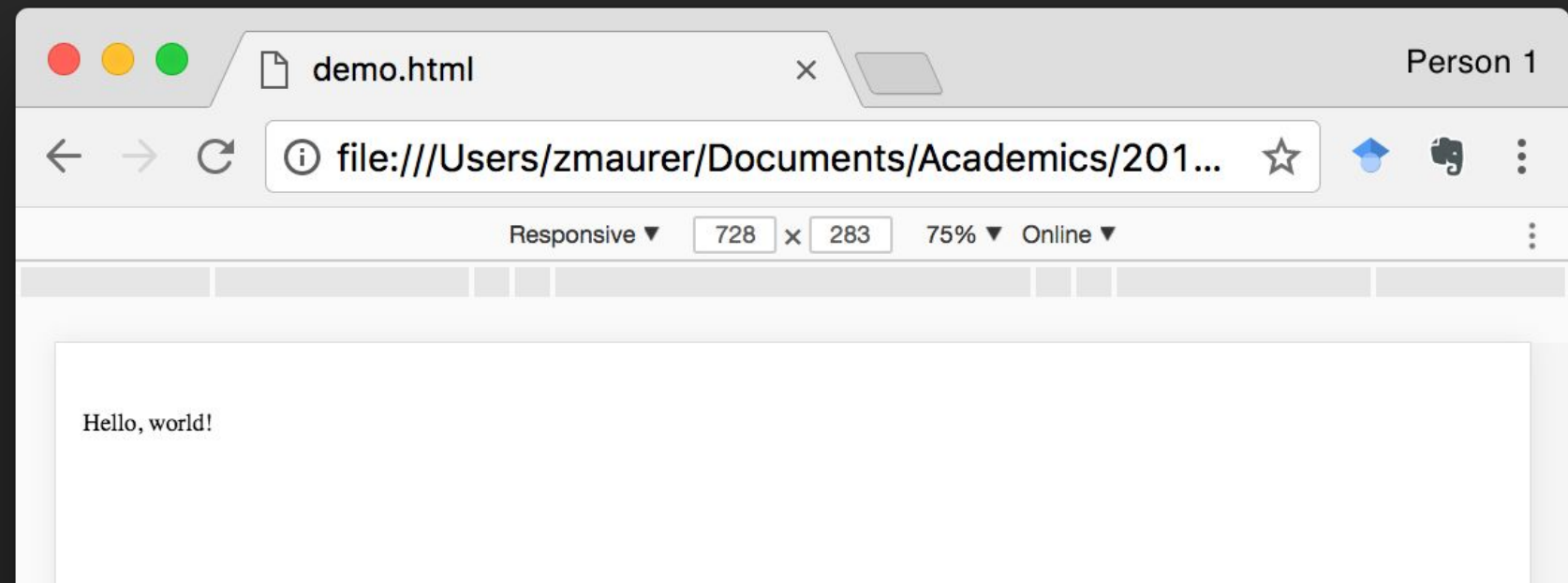
```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <style> /* CSS */ </style>
</head>

<body>
Hello, world!
  <script>
    console.log("Hello, world!");
    function add2(x) {
      return x + 2;
    }
    console.log("2 + 2 is " + add2(2));
  </script>
</body>
</html>
```

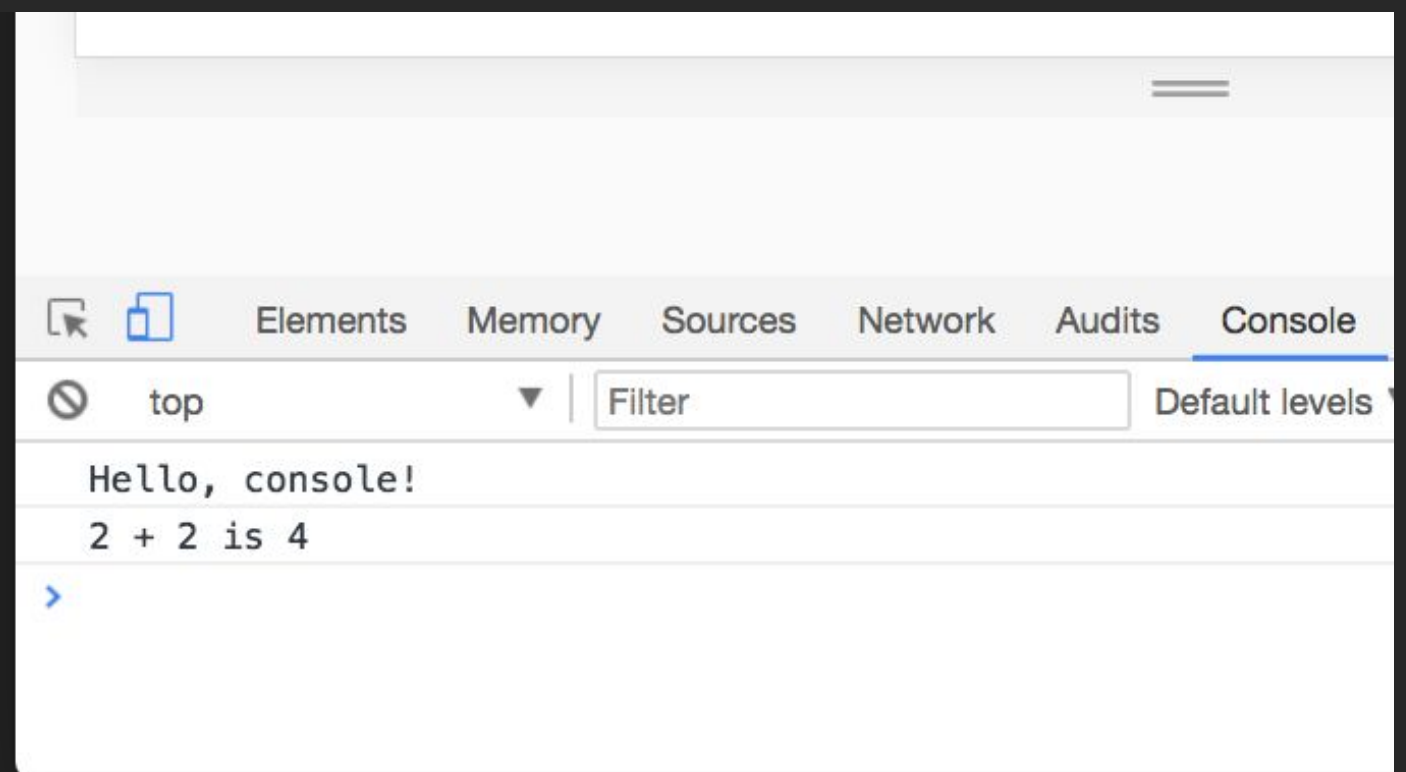
hello-javascript.html

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <style> /* CSS */ </style>
</head>

<body>
Hello, world!
  <script>
    console.log("Hello, world!");
    function add2(x) {
      return x + 2;
    }
    console.log("2 + 2 is " + add2(2));
  </script>
</body>
</html>
```



...



hello-d3.html

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
  <meta charset="utf-8">
```

```
  <style> /* CSS */ </style>
```

```
</head>
```

```
<body>
```

```
  <script src="https://d3js.org/d3.v5.min.js" ></script>
```

```
  <script>
```

```
    // JavaScript code that handles the logic of adding SVG elements
```

```
    // that make up the visual building blocks of your data visualization
```

```
  </script>
```

```
</body>
```

```
</html>
```

hello-d3.html

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

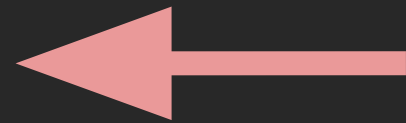
```
  <meta charset="utf-8">
```

```
  <style> /* CSS */ </style>
```

```
</head>
```

```
<body>
```

```
  <script src="https://d3js.org/d3.v5.min.js" ></script>
```



```
  <script>
```

```
    // JavaScript code that handles the logic of adding SVG elements
```

```
    // that make up the visual building blocks of your data visualization
```

```
  </script>
```

```
</body>
```

```
</html>
```


DOM Manipulation

What is the DOM?

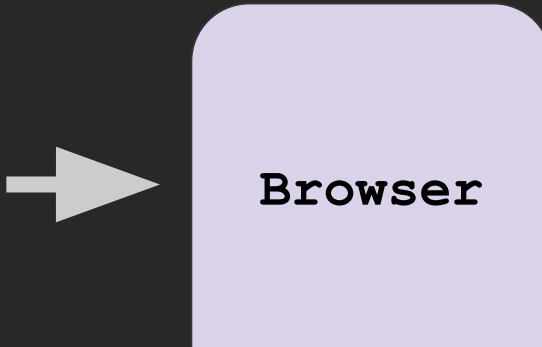
DOM: "Document Object Model"

```
<html>
  <head>
    <title></title>
  </head>
  <body>
    <h1></h1>
    <div>
      <p></p>
    </div>
  </body>
</html>
```

[Adapted from Victoria Kirst's cs193x [slides](#).]

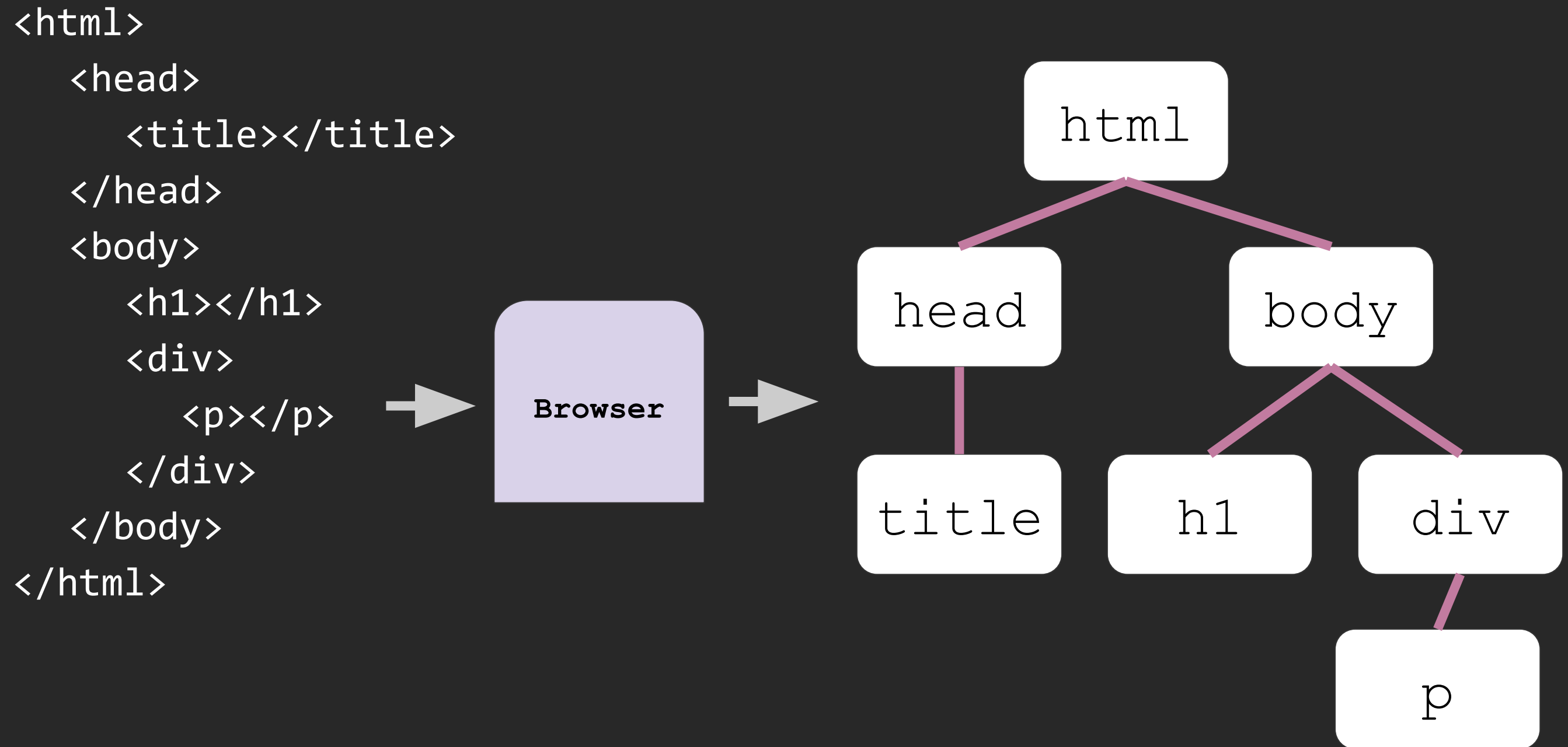
DOM: "Document Object Model"

```
<html>
  <head>
    <title></title>
  </head>
  <body>
    <h1></h1>
    <div>
      <p></p>
    </div>
  </body>
</html>
```



[Adapted from Victoria Kirst's cs193x [slides](#).]

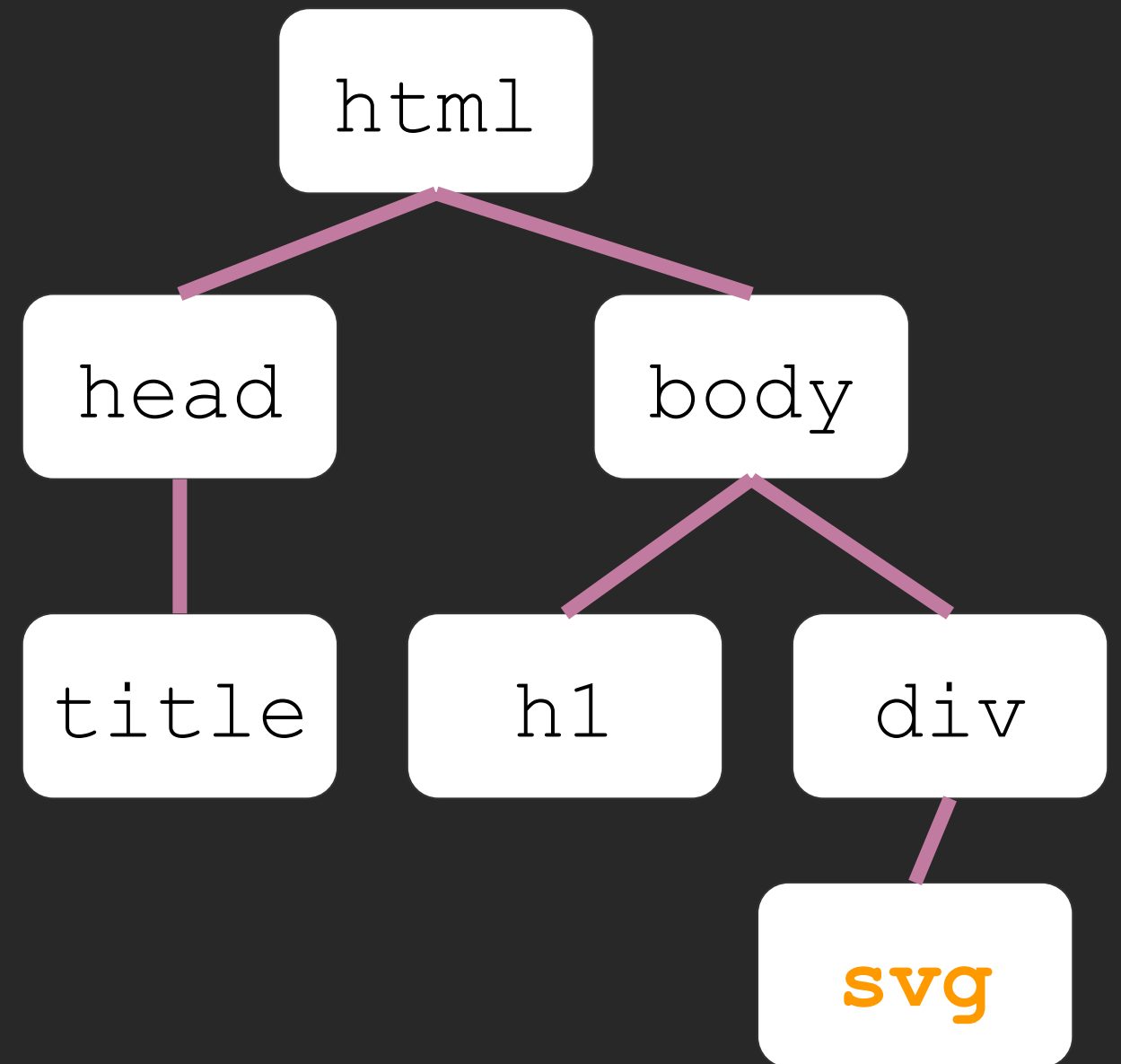
DOM: "Document Object Model"



[Adapted from Victoria Kirst's [cs193x slides](#).]

DOM: "Document Object Model"

```
<html>
  <head>
    <title></title>
  </head>
  <body>
    <h1></h1>
    <div>
      <svg></svg>
    </div>
  </body>
</html>
```



[Adapted from Victoria Kirst's cs193x [slides](#).]

D3: Selecting & manipulating DOM

```
<html>
```

```
...
```

```
<svg width="960" height="500">
```

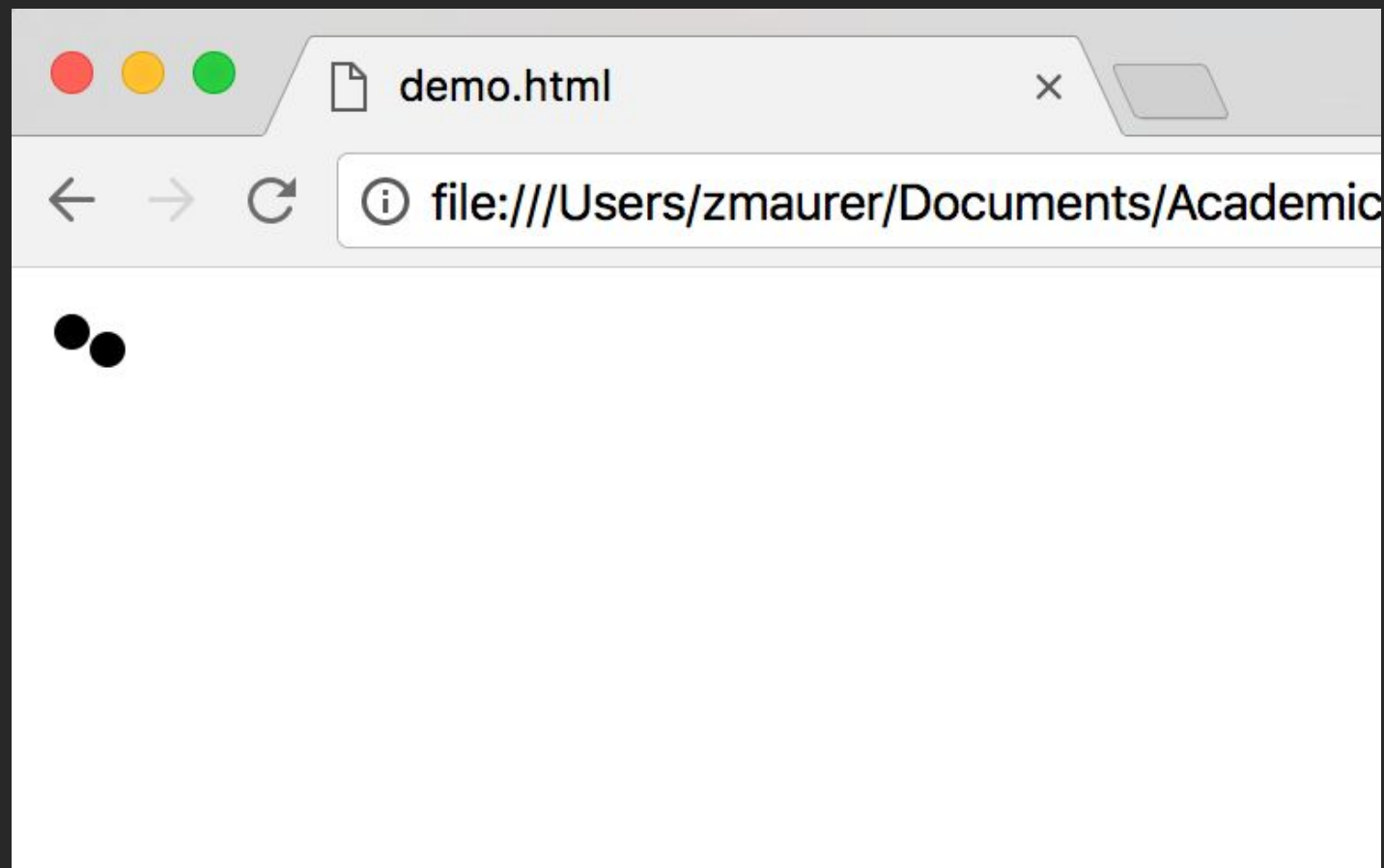
```
  <circle cx="10" cy="10" r="5"></circle>
```

```
  <circle cx="20" cy="15" r="5"></circle>
```

```
</svg>
```

D3: Selecting & manipulating DOM

```
<html>
...
<svg width="960" height="500">
  <circle cx="10" cy="10" r="5" >/circle>
  <circle cx="20" cy="15" r="5" >/circle>
</svg>
```



D3: Selecting & manipulating DOM

```
<html>
```

```
...
```

```
<svg width="960" height="500">
```

```
  <circle cx="10" cy="10" r="5"></circle>
```

```
  <circle cx="20" cy="15" r="5"></circle>
```

```
</svg>
```

```
<script>
```

```
// select all SVG circle elements
```

```
var circles =
```

```
  d3.selectAll("circle");
```

```
</script>
```


D3: Selecting & manipulating DOM

```
<html>
```

```
...
```

```
<svg width="960" height="500">
```

```
  <circle cx="10" cy="10" r="5" ></circle>
```

```
  <circle cx="20" cy="15" r="5" ></circle>
```

```
</svg>
```

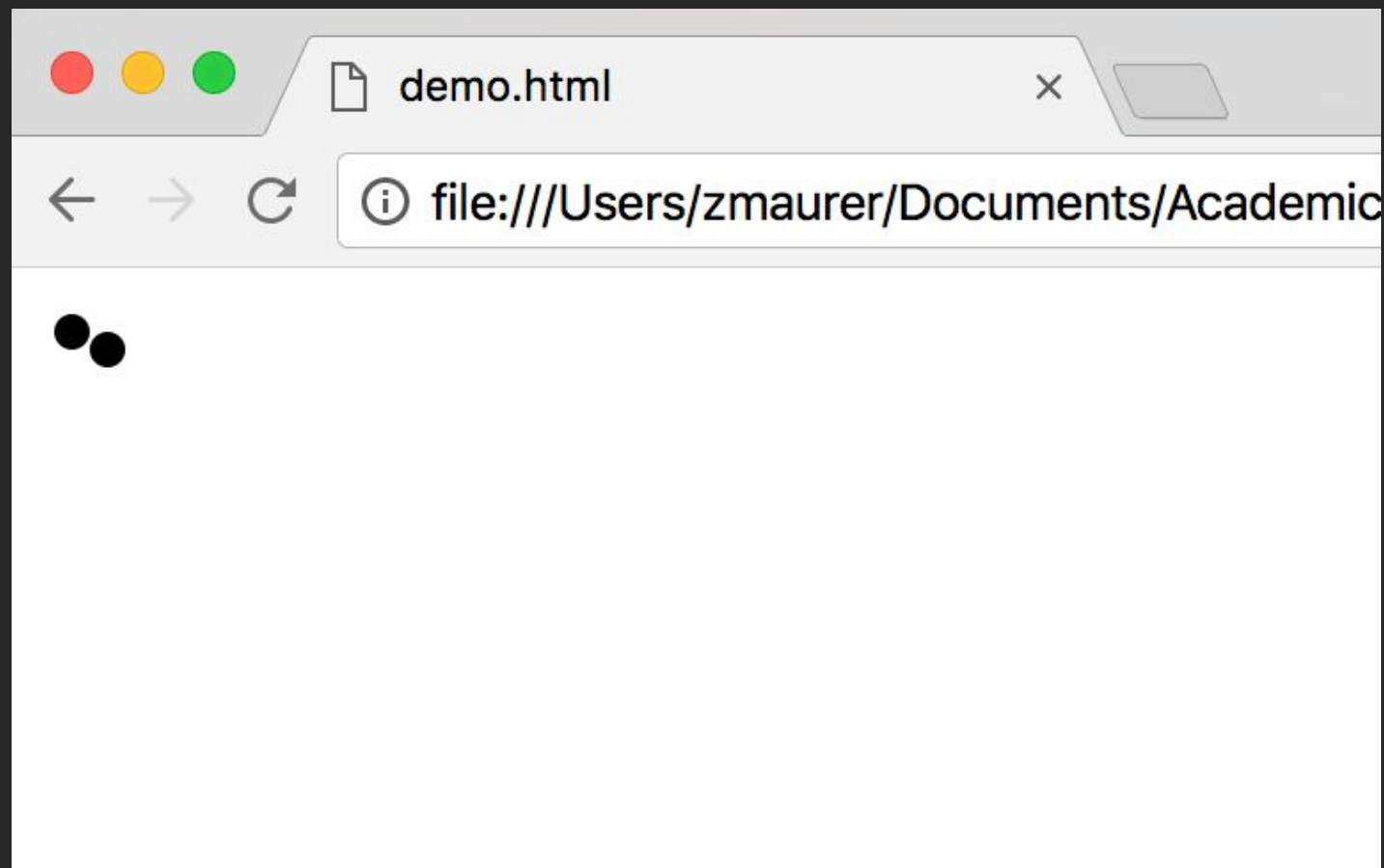
```
<script>
```

```
// select all SVG circle elements
```

```
var circles =
```

```
  d3.selectAll("circle");
```

```
</script>
```



D3: Selecting & manipulating DOM

```
<html>
```

```
...
```

```
<svg width="960" height="500">
```

```
  <circle cx="10" cy="10" r="5"></circle>
```

```
  <circle cx="20" cy="15" r="5"></circle>
```

```
</svg>
```

```
<script>
```

```
// select all SVG circle elements
```

```
var circles =
```

```
  d3.selectAll("circle");
```

```
// set attributes and styles
```

```
circles.attr("cx", 40);
```

```
circles.attr("cy", 50);
```

```
circles.attr("r", 24);
```

```
circles.style("fill", "red");
```

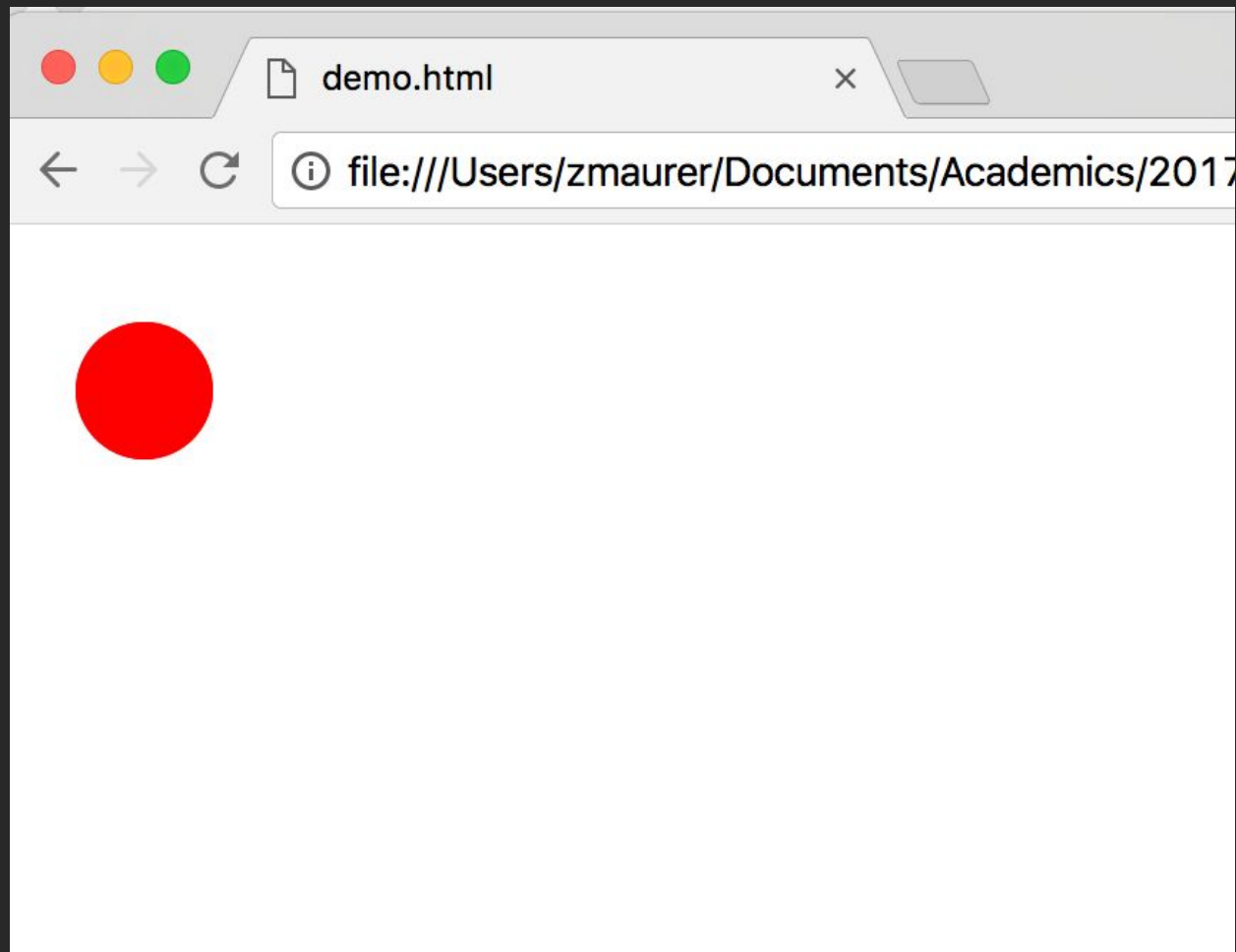
```
</script>
```

D3: Selecting & manipulating DOM

```
<html>
...
<svg width="960" height="500">
  <circle cx="10" cy="10" r="5"></circle>
  <circle cx="20" cy="15" r="5"></circle>
</svg>
```

```
<script>
```

```
// select all SVG circle elements
var circles =
  d3.selectAll("circle");
```



```
// set attributes and styles
circles.attr("cx", 40);
circles.attr("cy", 50);
circles.attr("r", 24);
circles.style("fill", "red");
```

```
</script>
```

D3: Selecting & manipulating DOM

```
<html>
```

```
...
```

```
<svg width="960" height="500">
```

```
  <circle cx="10" cy="10" r="5" ></circle>
```

```
  <circle cx="20" cy="15" r="5" ></circle>
```

```
</svg>
```

```
<script>
```

```
// select all SVG circle elements  
var circles = d3.select("circle");
```

```
// set attributes and styles  
circles.attr("cx", 40);  
circles.attr("cy", 50);  
circles.attr("r", 24);  
circles.style("fill", "red");
```

```
</script>
```

D3: Selecting & manipulating DOM

```
<html>
```

```
...
```

```
<svg width="960" height="500">
```

```
  <circle cx="10" cy="10" r="5"></circle>
```

```
  <circle cx="20" cy="15" r="5"></circle>
```

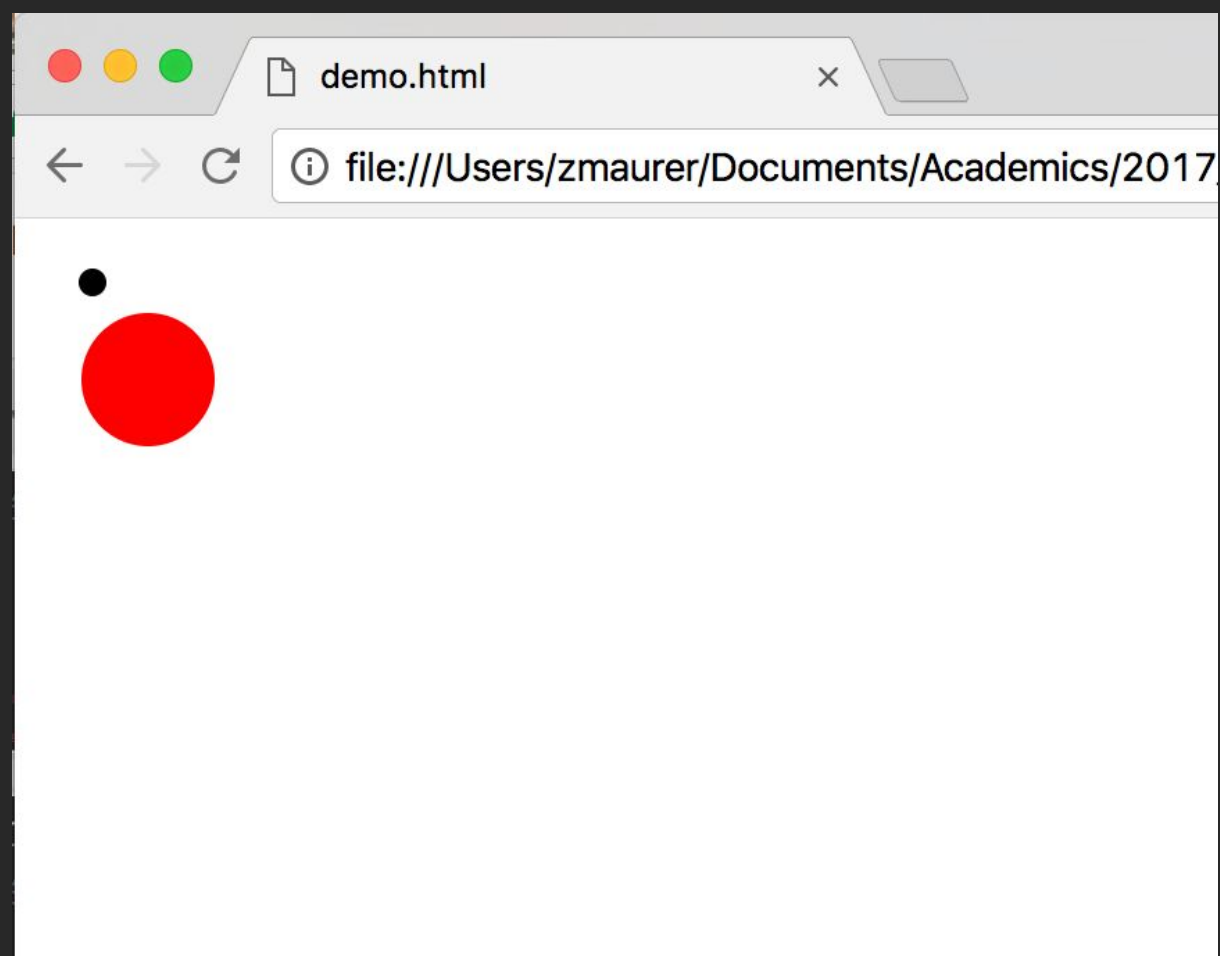
```
</svg>
```

```
<script>
```

```
// select all SVG circle elements  
var circles = d3.select("circle");
```

```
// set attributes and styles  
circles.attr("cx", 40);  
circles.attr("cy", 50);  
circles.attr("r", 24);  
circles.style("fill", "red");
```

```
</script>
```



D3: Selecting & manipulating DOM

```
<html>
```

```
...
```

```
<svg width="960" height="500">
```

```
  <circle cx="10" cy="10" r="5" ></circle>
```

```
  <circle cx="20" cy="15" r="5" ></circle>
```

```
</svg>
```

```
<script>
```

```
// all together!!
```

```
d3.select("circle")
```

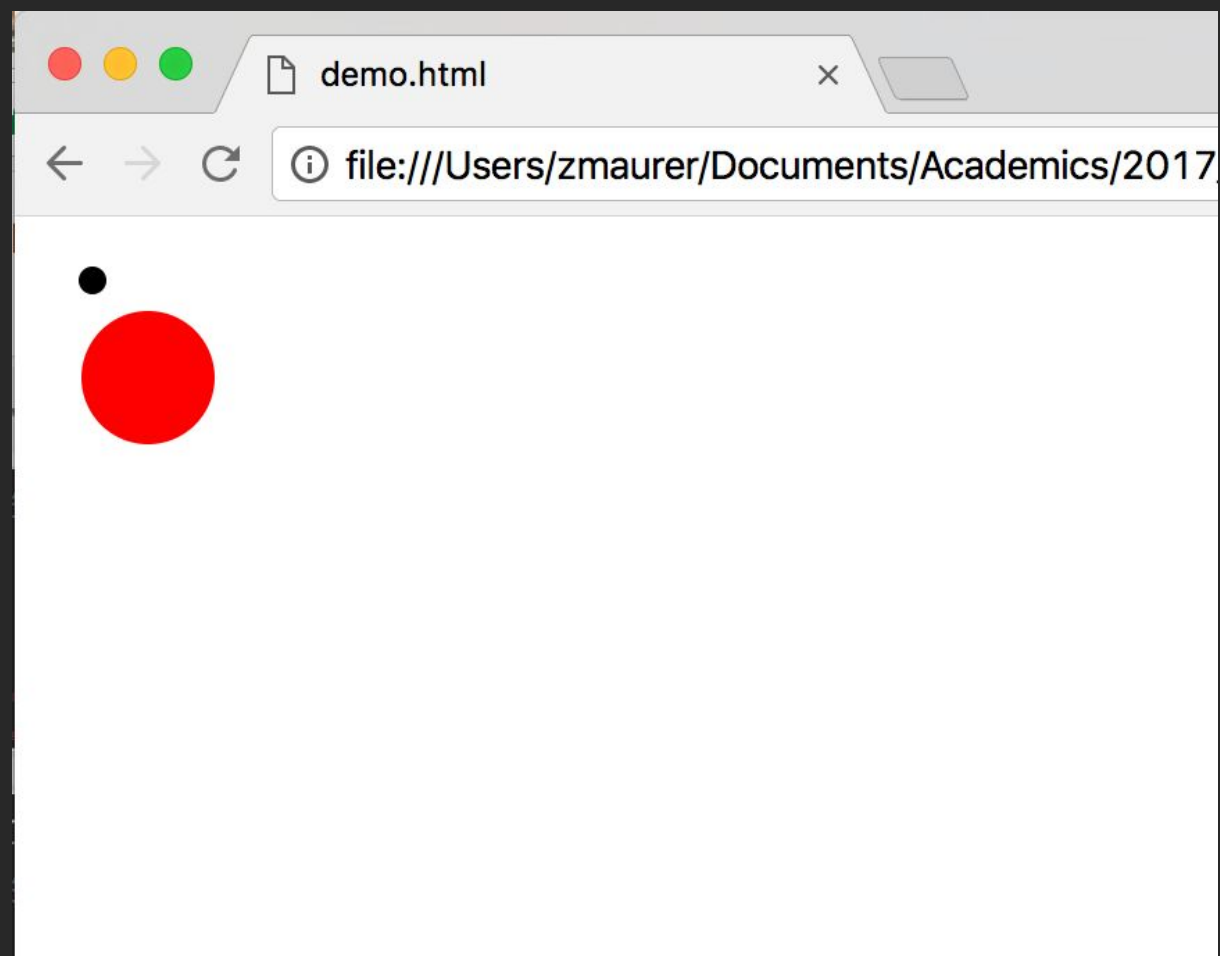
```
  .attr("cx", 40)
```

```
  .attr("cy", 50)
```

```
  .attr("r", 24)
```

```
  .style("fill", "red");
```

```
</script>
```



Where do I learn about SVG attributes and D3.js?

SVG Attribute Reference

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

D3 API Reference v5.0

<https://github.com/d3/d3/blob/master/API.md>

Information Visualization: Programming with D3.js

<https://www.coursera.org/learn/information-visualization-programming-d3js/home/welcome>

Why do I want to learn about D3?

Because it's cool and useful!

And also...

Assignment 3: Dynamic Queries

Create a **small** interactive dynamic query application similar to Homefinder, but for SF Restaurant data

1. Implement interface and produce final writeup
2. Submit the application and a final writeup on canvas



Can work alone or in pairs
Due before class on **Oct 29, 2018**

Assignment 3 Dataset

- **business_id**: A unique integer for each restaurant.
- **business_name**: A string describing the **name of the restaurant**.
- **business_address**: A string describing the **address** of the restaurant.
- **business_postal_code**: A string describing the postal code of the restaurant.
- **business_latitude**: Float describing latitude of the restaurant.
- **business_longitude**: Float describing longitude of the restaurant.
- **business_location**: A string that contains both the latitude and the longitude.
- **business_phone_number**: A string describing the restaurant's phone number.
- **inspection_id**: A unique string for each inspection.
- **inspection_date**: A **datetime timestamp** in the format of **mm/dd/yy**.
- **inspection_score**: An integer describing the **inspection score** of the restaurant.
- **violation_id**: A string describing the id of the violation. Could be empty if the restaurant is free of violation.
- **violation_description**: A string **describing the violation**. Could be empty if the restaurant is free of violation.
- **risk_category**: A string **describing the risk level**. Could be empty if the restaurant is free of violation.

D3 as a DataViz tool.

Data

Selections

Mappings

Dynamic JS
Object

```
d3.selectAll(..)  
  .data(myData)
```

```
.enter()  
  .append(..)  
  .style(..)  
  .text(..)
```

Data

Selections

Mappings

Dynamic JS
Object

```
d3.selectAll(..)  
  .data(myData)
```

```
.enter()  
  .append(..)  
  .style(..)  
  .text(..)
```

Data elements are
bound to nodes in the
DOM.

Data

Selections

Mappings

Dynamic JS
Object

```
d3.selectAll(..)  
  .data(myData)
```

```
.enter()  
  .append(..)  
  .style(..)  
  .text(..)
```

Data elements are
bound to nodes in the
DOM.

Bound elements are
'drawn' with SVG / HTML
attributes and CSS.

Let's make a scatter plot 🐱 🐶

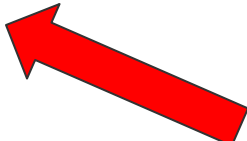
```
id,animal,weight,height,name
1,cat,10,3,phyllis
2,cat,3,3,oreo
3,cat,9,9,sam
4,cat,3,5,dog
5,cat,6,5,fred
6,cat,5,6,jane
7,cat,1,8,esmerelda
8,dog,9,2,garfield
9,dog,8,9,alpha
10,dog,7,7,omega
11,dog,2,3,zeta
12,dog,8,3,cupcake
```

Steps

1. **Setup** (start server, html boilerplate, link D3 src)
2. **Define canvas, scales, axes**
3. **Load data**
4. **Bind data**
5. **Draw marks**
6. **Interactive update with enter/exit**

Start with HTML structure & basic CSS

```
<html>
  <head>
    <meta charset="utf-8">
    <title>🐱 vs. 🐶 stats</title>
    <style>
      /*←— Write CSS here →*/
    </style>
    <script src="https://d3js.org/d3.v4.min.js"></script>
  </head>
  <body>
    ←— Write some HTML here →
    <svg id="animal-viz"></svg>
    <script>
      // Write some JavaScript here!
    </script>
  </body>
</html>
```



This is where
D3 will append
elements

Start with HTML structure & basic CSS

```
<html>
  <head>
    <meta charset="utf-8">
    <title>🐱 vs. 🐶 stats</title>
    <style>
      h1 {
        font-size: 50px;
        font-family: Helvetica, sans-serif;
      }
      button#cats-only {
        background-color: steelblue;
      }
      button#dogs-only {
        background-color: salmon;
      }
      button#both {
        background-color: grey;
      }
    </style>
    <script src="https://d3js.org/d3.v4.min.js"></script>
  </head>
  <body>
    <section>
      <h1>Height vs. Weight for Cats & Dogs</h1>
      <span> Fancy filters: </span>
      <button id="cats-only" data-filter="cat">Cats Only</button>
      <button id="dogs-only" data-filter="dog">Dogs Only</button>
      <button id="both" data-filter="both">Both</button>
    </section>
    <svg id="animal-viz"></svg>
    <script>
      // Write some JavaScript here!
    </script>
  </body>
</html>
```

Height vs. Weight for Cats & Dogs

Fancy filters:

How do I get this to show up?

Run it on a local server! Use your browser to render the html.

> cd path/to/your/project

> python -m SimpleHTTPServer

[if using Python2.x]

> python -m http.server

[if using Python 3.x]

```
[$ python -m http.server  
Serving HTTP on 0.0.0.0 port 8000 ...  
█
```

In your browser, visit: `http://localhost:<port>` [see number above]

Note: The server you start in your project directory looks for a file called "index.html" to render when you visit localhost:<port>

An easy way to start is to put all your JS, HTML, CSS in this file.
[Not recommended for big projects.]

Let's write some D3.js!

Note: all code is contained within the
`<script></script>` tags

Selections

```
// Select the `<svg id="animal-viz"></svg>` DOM
let wholeChart = d3.select('#animal-viz');

// Set size of the plot and spacing around it (margin)
let plotWidth = 500;
let plotHeight = 500;
let plotMargin = 50;
let outerWidth = plotWidth + 2 * plotMargin;
let outerHeight = plotHeight + 2 * plotMargin;

// Set the size of the whole chart
// We could have done this in CSS too,
// since it's not dependent on our data
wholeChart
  .attr('width', outerWidth)
  .attr('height', outerHeight);
```

- Can select by any CSS selector
- Chain methods to set attributes
- <https://github.com/d3/d3-selection>
- <https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelector>

Height vs. Weight for Cats & Dogs

Fancy filters: Cats Only Dogs Only Both

SVG Coordinates

```
// Append a `g` element to our SVG: we'll work in this for our plot  
// It has margins  
let plot = wholeChart.append('g')  
  .attr('transform', `translate(${plotMargin},${plotMargin})`);
```

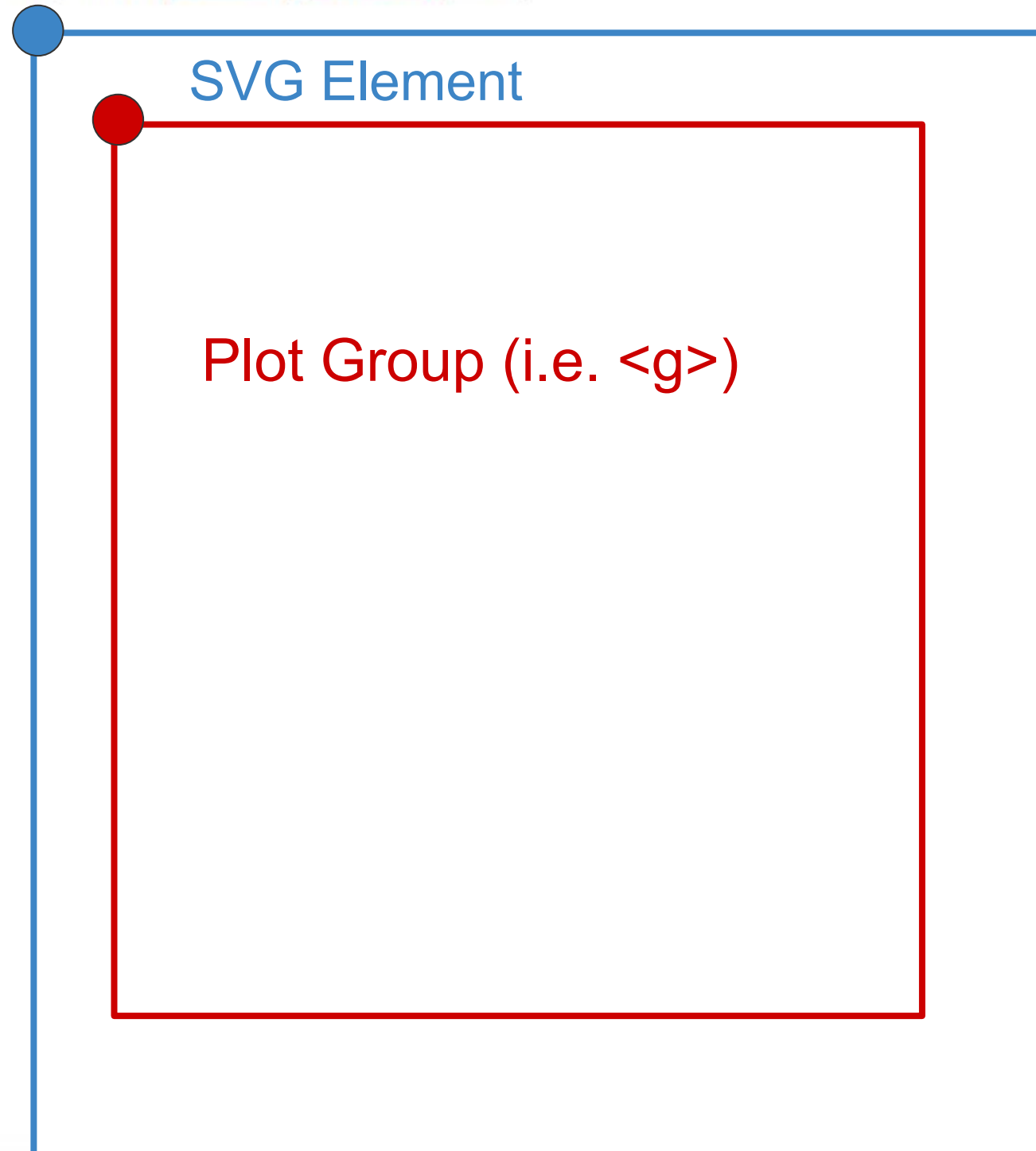


Use transforms on `<g>` to define a new origin
(e.g., plotting area)

SVG Coordinates

Height vs. Weight for Cats & Dogs

Fancy filters: Cats Only Dogs Only Both



Scales

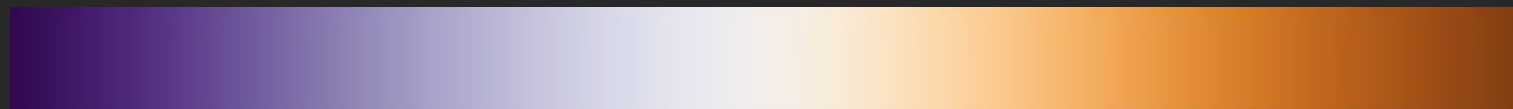
```
let xScale = d3.scaleLinear()
  .domain([0, 10])
  .range([0, plotWidth]);
let yScale = d3.scaleLinear()
  .domain([0, 10])
  .range([plotHeight, 0]); // s
```

- A scale is a function.
- This function maps your data “space” to the encoding “space”.
- E.g. 10lbs → 253 px from origin
- Can also make nominal/ordinal scales, map to colors, interpolate between colors, ...
- <https://github.com/d3/d3-scale>

Data / Domain: [10, 11, 12, 13, 14, 15 ..., 26]

scale(data) ↓

Encoding / Range:



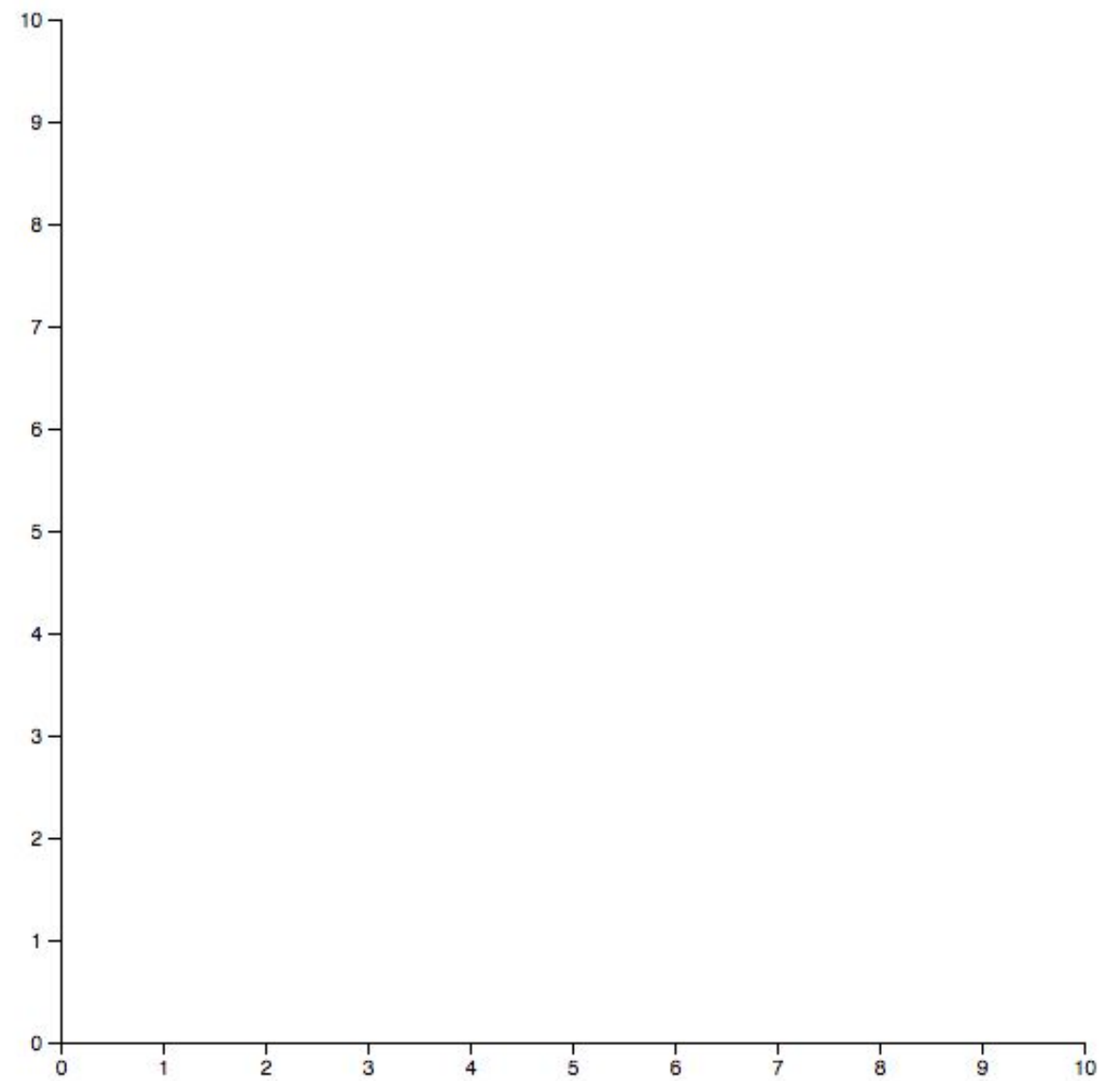
Drawing axes

```
// Draw our axes based on xScale and yScale
let xAxis = plot.append('g')
  .attr('transform', `translate(0,${plotHeight})`)
  .call(d3.axisBottom(xScale));
let yAxis = plot.append('g')
  .call(d3.axisLeft(yScale));
```

- `d3.axisBottom(xscale)` returns a function
- We create a new 'g' node, then apply that function to it.
- D3 draws it for us. ✨
- <https://github.com/d3/d3-axis>

Height vs. Weight for Cats & Dogs

Fancy filters: Cats Only Dogs Only Both



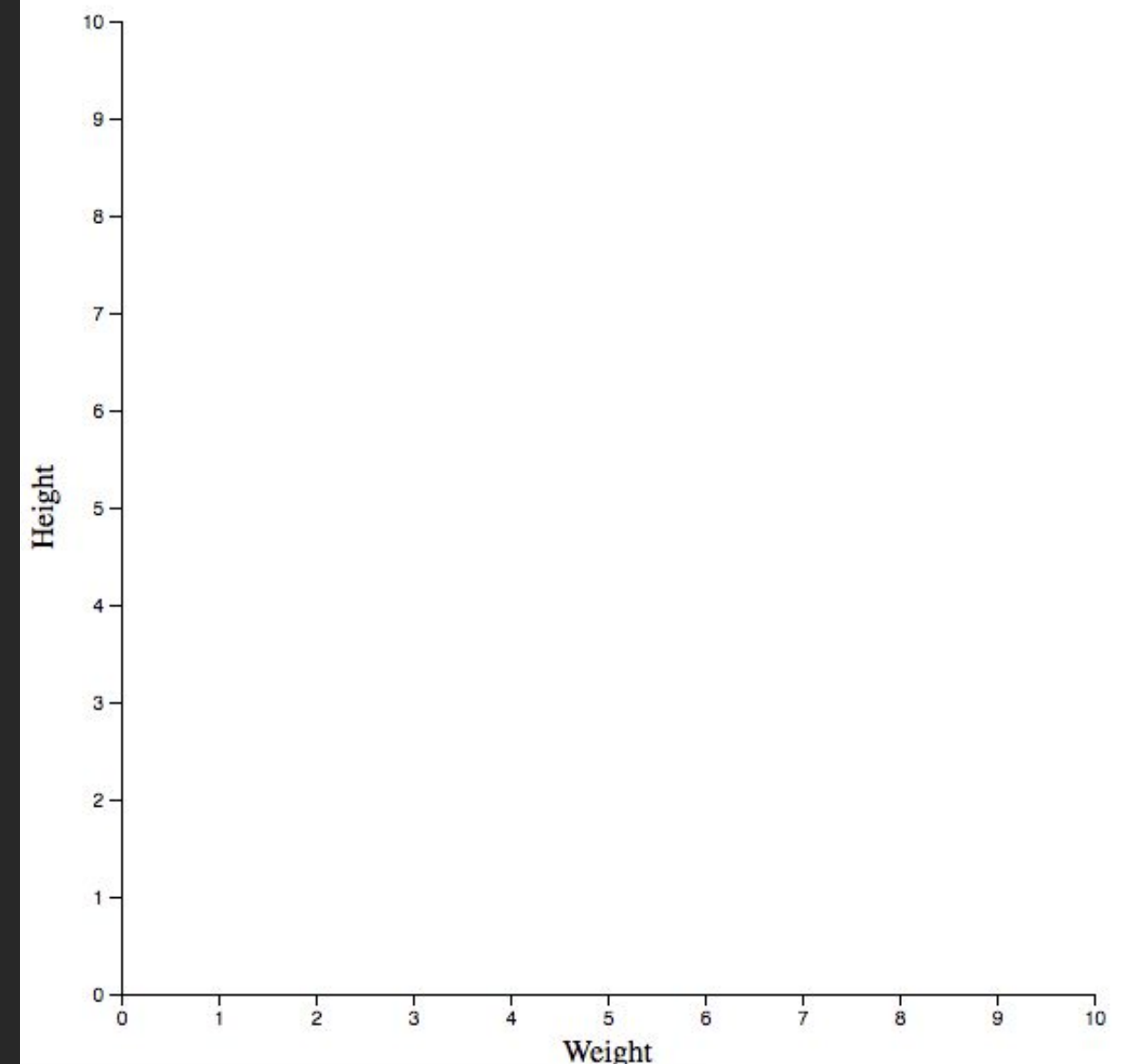
Drawing labels: More svg...

```
// Label the x axis, halfway along the  
let xAxisLabel = plot.append('text')  
  .attr('x', plotWidth / 2)  
  .attr('y', plotHeight + 35)  
  .style('text-anchor', 'middle')  
  .text('Weight');
```

```
// Label the y axis,  
let yAxisLabel = plot.append('text')  
  .attr('transform', 'rotate(-90)')  
  .attr('y', -35) // Actually moving left  
  .attr('x', - (plotHeight / 2)) // Moving  
  .style('text-anchor', 'middle')  
  .text('Height');
```

Height vs. Weight for Cats & Dogs

Fancy filters: Cats Only Dogs Only Both



Recap: What do we have so far?

```

// Select the `<svg id="animal-viz"></svg>` DOM node
let wholeChart = d3.select('#animal-viz');

// Set size of the plot and spacing around it (for axes and labels)
let plotWidth = 500;
let plotHeight = 500;
let plotMargin = 50;
let outerWidth = plotWidth + 2 * plotMargin;
let outerHeight = plotHeight + 2 * plotMargin;

// Set the size of the whole chart
// We could have done this in CSS too,
// since it's not dependent on our data
wholeChart
  .attr('width', outerWidth)
  .attr('height', outerHeight);

// Append a `g` element to our SVG: we'll work in this for our plot
// It has margins
let plot = wholeChart.append('g')
  .attr('transform', `translate(${plotMargin},${plotMargin})`);

// Create our scales,
// each of which will map data from 0-10 to the size of our plot
let xScale = d3.scaleLinear()
  .domain([0, 10])
  .range([0, plotWidth]);
let yScale = d3.scaleLinear()
  .domain([0, 10])
  .range([plotHeight, 0]); // SVG has its origin in the top left,

// xScale and yScale are functions:
// xScale(0) ⇒ 0; xScale(10) ⇒ 500; xScale(2) ⇒ 100
// yScale(0) ⇒ 500; yScale(10) ⇒ 0; yScale(2) ⇒ 400

// Draw our axes based on xScale and yScale
let xAxis = plot.append('g')
  .attr('transform', `translate(0,${plotHeight})`)
  .call(d3.axisBottom(xScale));
let yAxis = plot.append('g')
  .call(d3.axisLeft(yScale));

// Label the x axis, halfway along the width of the plot, 35 px up
let xAxisLabel = plot.append('text')
  .attr('x', plotWidth / 2)
  .attr('y', plotHeight + 35)
  .style('text-anchor', 'middle')
  .text('Weight');

// Label the y axis,
let yAxisLabel = plot.append('text')
  .attr('transform', 'rotate(-90)')
  .attr('y', -35) // Actually moving left, since we rotated
  .attr('x', -(plotHeight / 2)) // Move vertically down halfway
  .style('text-anchor', 'middle')
  .text('Height');

```



```

// Select the `<svg id="animal-viz"></svg>` DOM node
let wholeChart = d3.select('#animal-viz');

// Set size of the plot and spacing around it (for axes and labels)
let plotWidth = 500;
let plotHeight = 500;
let plotMargin = 50;
let outerWidth = plotWidth + 2 * plotMargin;
let outerHeight = plotHeight + 2 * plotMargin;

// Set the size of the whole chart
// We could have done this in CSS too,
// since it's not dependent on our data
wholeChart
  .attr('width', outerWidth)
  .attr('height', outerHeight);

// Append a `g` element to our SVG: we'll work in this for our plot
// It has margins
let plot = wholeChart.append('g')
  .attr('transform', `translate(${plotMargin},${plotMargin})`);

// Create our scales,
// each of which will map data from 0-10 to the size of our plot
let xScale = d3.scaleLinear()
  .domain([0, 10])
  .range([0, plotWidth]);
let yScale = d3.scaleLinear()
  .domain([0, 10])
  .range([plotHeight, 0]); // SVG has its origin in the top left,

// xScale and yScale are functions:
// xScale(0) ⇒ 0; xScale(10) ⇒ 500; xScale(2) ⇒ 100
// yScale(0) ⇒ 500; yScale(10) ⇒ 0; yScale(2) ⇒ 400

// Draw our axes based on xScale and yScale
let xAxis = plot.append('g')
  .attr('transform', `translate(0,${plotHeight})`)
  .call(d3.axisBottom(xScale));
let yAxis = plot.append('g')
  .call(d3.axisLeft(yScale));

// Label the x axis, halfway along the width of the plot, 35 px up
let xAxisLabel = plot.append('text')
  .attr('x', plotWidth / 2)
  .attr('y', plotHeight + 35)
  .style('text-anchor', 'middle')
  .text('Weight');

// Label the y axis,
let yAxisLabel = plot.append('text')
  .attr('transform', 'rotate(-90)')
  .attr('y', -35) // Actually moving left, since we rotated
  .attr('x', -(plotHeight / 2)) // Move vertically down halfway
  .style('text-anchor', 'middle')
  .text('Height');

```

Define the chart container.

Define the plot dimensions.

Set the chart container's attributes.

Add a group (<g>) for the actual plot.

Define the scales.

Draw the axes on the plot group.

Draw the axis labels.

Data loading & joining

📢 *This is important!* 📢

Loading the data from an external file

```
// Load data using Promises.  
// Parsing rows lets you format data and reduces size  
// of data points if there are unneeded attributes.  
d3.csv("animals.csv", parseInputRow).then(loadData);  
  
// Convert weight and height from strings to numbers  
function parseInputRow(d) {  
  return {  
    id: +d.id,  
    animal: d.animal,  
    weight: +d.weight,  
    height: +d.height,  
    name: d.name  
  };  
}  
  
function loadData(animalData) {  
  // animal data looks like [  
  // {animal: "cat", weight: 10, height: 3},  
  // ..., ]  
  
  // Draw the initial scatter plot  
  drawScatterPlot(animalData);  
}
```

- Load the CSV, then pass the data through 2 callback functions.
- First function runs on each row, parsing it.
- <https://github.com/d3/d3-request#csv>

Drawing the scatter plot

```
function drawScatterPlot(animalData) {  
  // Create a selection of circles in our plot (empty on the first go)  
  let circles = plot.selectAll('circle');  
  
  // Bind our animal data to the circles, using the "id" field as our key  
  let updatedCircles = circles.data(animalData, d => d.id);  
  
  // We'll use "enter" to make circles for new datapoints  
  let enterSelection = updatedCircles.enter();  
  let newCircles = enterSelection.append('circle')  
    .attr('r', 20)  
    .attr('cx', function (d) { return xScale(d.weight); })  
    .attr('cy', function (d) { return yScale(d.height); })  
    .style('fill', function(d) {  
      return d.animal === 'cat' ? 'steelblue' : 'salmon';  
    });  
  
  // Now we'll select all the circles that no longer  
  // have any corresponding data after the data join  
  let unselectedCircles = updatedCircles.exit();  
  // And we'll remove those nodes from the DOM - poof!  
  updatedCircles.exit().remove();  
}
```


Selection + data

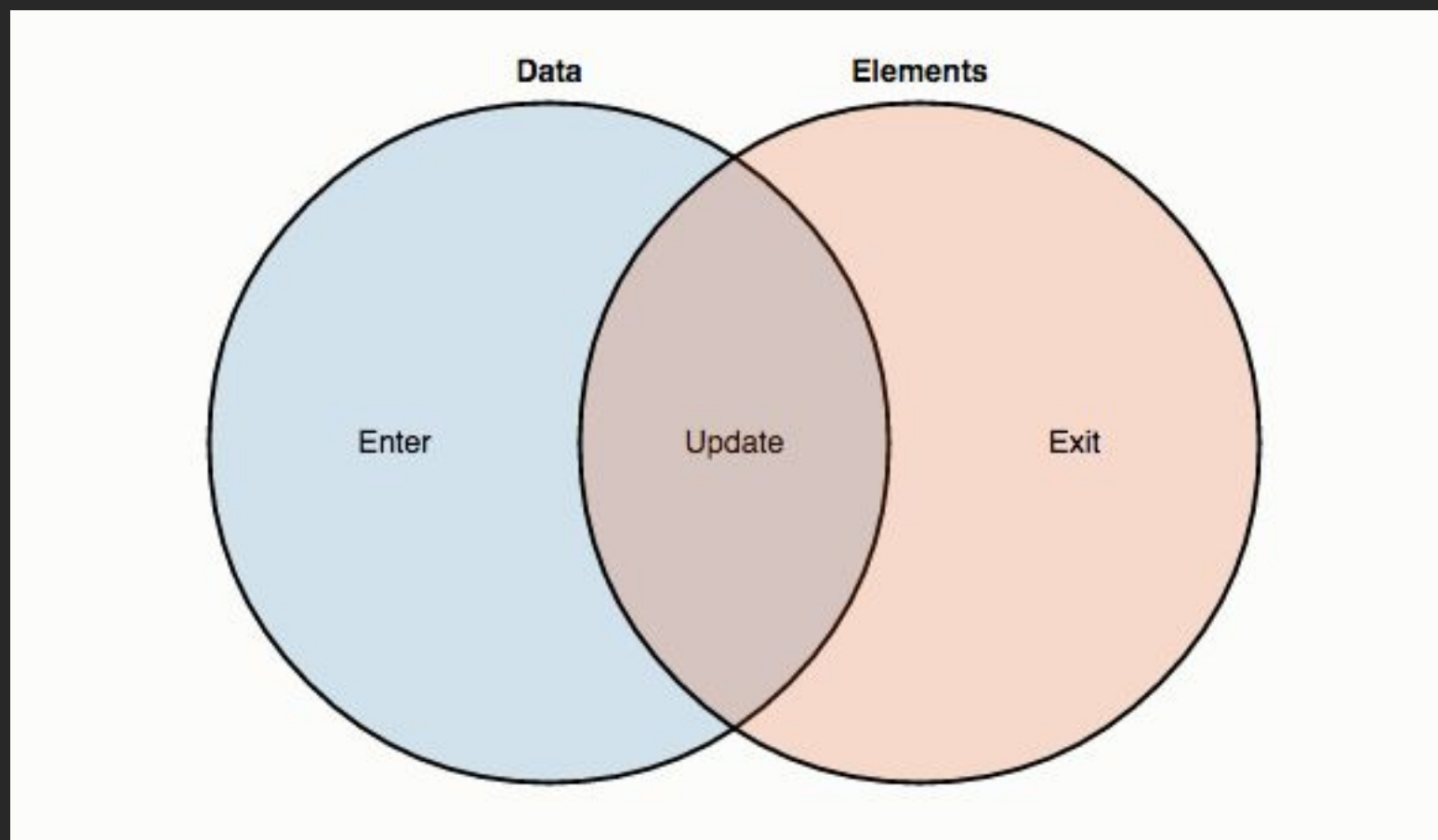
```
function drawScatterPlot(animalsData) {  
  // Create a selection of circles in our plot (empty on the first go)  
  let circles = plot.selectAll('circle');  
  
  // Bind our animal data to the circles, using the "id" field as our key  
  let updatedCircles = circles.data(animalsData, d => d.id);  
}
```

- `.data()` takes an array of data, and joins it to a selection
 - Returns a new selection, called the "update" selection
 - Note: `.data()` does not draw anything!
- 2nd arg is a "key function"
 - Tells D3 how to match the data with the elements in the selection
 - By default, uses index in array as key

What's happening when you join data?



What's happening when you join data?



3 sub-selections:

- *Enter*: New data, missing elements
- *Update*: Data points joined to existing elements
- *Exit*: Leftover unbound elements

Enter selection + appending circles

```
// We'll use "enter" to make circles for new datapoints
let enterSelection = updatedCircles.enter();
let newCircles = enterSelection.append('circle')
  .attr('r', 20)
  .attr('cx', function (d) { return xScale(d.weight); })
  .attr('cy', function (d) { return yScale(d.height); })
  .style('fill', function(d) {
    return d.animal === 'cat' ? 'steelblue' : 'salmon';
  });
```

- `enter()`
joins the update data with the data that is already bound.
- `append()`
draws circles for the data in the join.
- Canonical Explanation:
https://github.com/d3/d3-selection#selection_enter :

Exit selection + removing nodes

```
// Now we'll select all the circles that no longer  
// have any corresponding data after the data join  
let unselectedCircles = updatedCircles.exit();  
// And we'll remove those nodes form the DOM - poof!  
updatedCircles.exit().remove();  
}
```

- `exit()`
selects the data that is not contained by the join.
- `remove()`
removes the circles for that data.
- Doesn't do anything now, but it will once we add interactive filtering...

Learn more about data-joining and “general update pattern”



Chris Given's Block 32d4c53f19aea6e528faf10bfe4f3da9
Updated July 29, 2017

Popular / About

Interactive General Update Pattern

```
function update(data) {  
  var update = svg.selectAll('circle')  
    .data(data, function (d) { return d })  
  var enter = update.enter()  
    .append('circle')  
  var exit = update.exit()  
  update.style('fill', 'black')  
  enter.style('fill', 'green')  
  exit.style('fill', 'red')  
    .remove()  
  update.merge(enter)  
    .call(pulse)  
}
```

data = ["a", "b", "c", "d", "f", "g"]

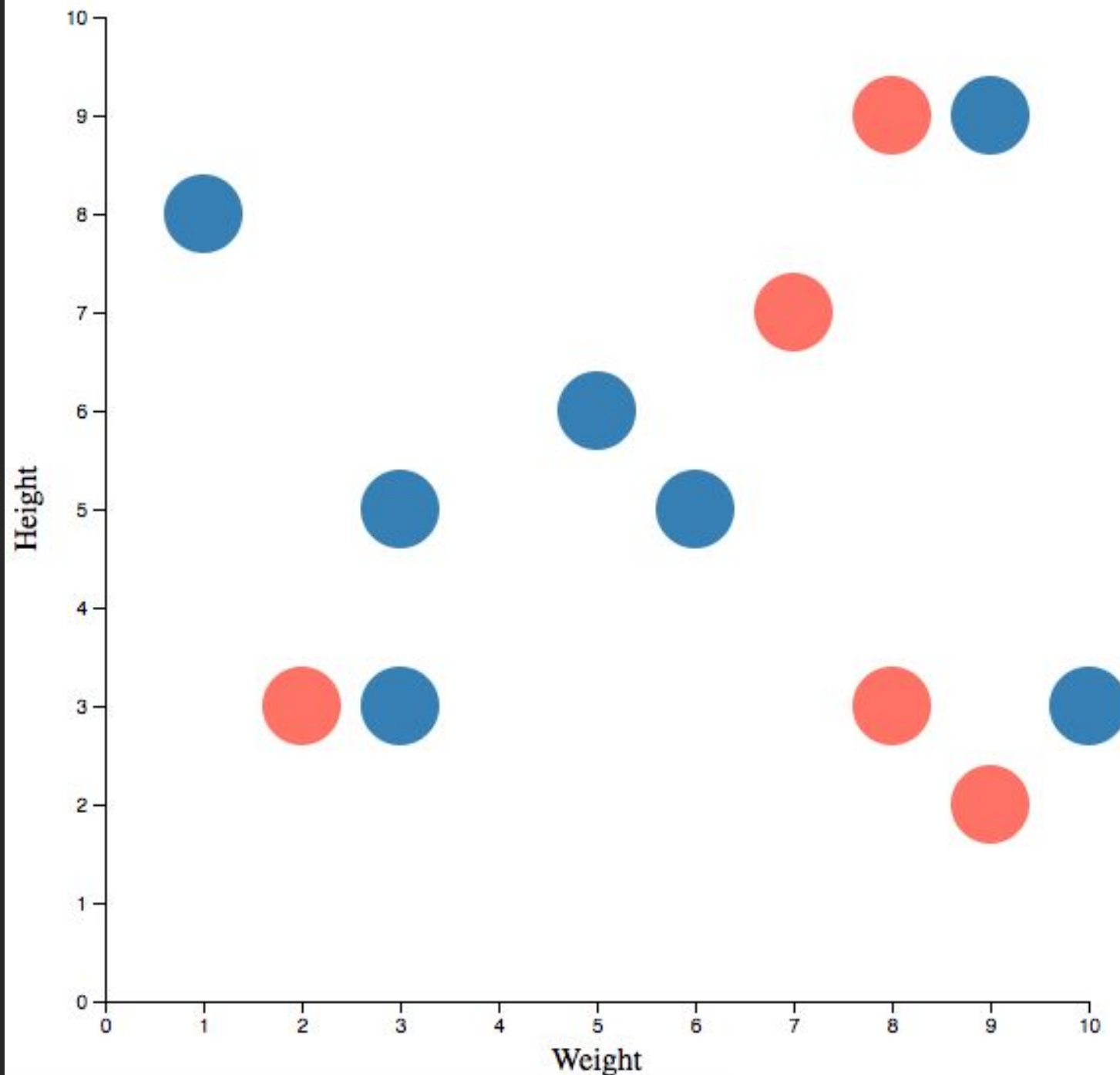
.enter() Although we're showing letters in the top left to illustrate that D3 knows about them, circles that correspond to these letters don't yet exist. By calling `.enter()` on the update selection, we select these non-existent circles, so that...

<https://bl.ocks.org/cmgiven/32d4c53f19aea6e528faf10bfe4f3da9>

Circles!

Height vs. Weight for Cats & Dogs

Fancy filters: Cats Only Dogs Only Both



Drawing the scatter plot

```
function drawScatterPlot(animalData) {  
  // Create a selection of circles in our plot (empty on the  
  let circles = plot.selectAll('circle');  
  
  // Bind our animal data to the circles, using the "id" field  
  let updatedCircles = circles.data(animalData, d => d.id);  
  
  // We'll use "enter" to make circles for new datapoints  
  let enterSelection = updatedCircles.enter();  
  let newCircles = enterSelection.append('circle')  
    .attr('r', 20)  
    .attr('cx', function (d) { return xScale(d.weight); })  
    .attr('cy', function (d) { return yScale(d.height); })  
    .style('fill', function(d) {  
      return d.animal === 'cat' ? 'steelblue' : 'salmon';  
    });  
  
  // Now we'll select all the circles that no longer  
  // have any corresponding data after the data join  
  let unselectedCircles = updatedCircles.exit();  
  // And we'll remove those nodes from the DOM - poof!  
  updatedCircles.exit().remove();  
}
```

Select all the circles.

Bind the data.
(i.e. update)

Join the data.

Draw the data.

Get the unmatched data.

Remove it.

Interactivity

```
let buttons = d3.selectAll('button');
buttons.on('click', function() {
  // When you write a function for a D3 selection with multiple nodes
  // `this` refers to the current DOM node
  let chosenAnimal = this.dataset.filter; // value of `data-filter` attr
  let filteredData;
  if (chosenAnimal === 'both') {
    filteredData = animalData;
  } else {
    filteredData = animalData.filter( d => d.animal === chosenAnimal );
  }
  drawScatterPlot(filteredData);
});
```

- In our script tag, add event listeners to the buttons.
- When you click on a button,
 - filter the dataset
 - redraw the plot
- D3 has an interface for this, but you can also use plain JavaScript `addEventListener()`

Interactivity

- Because `drawScatterPlot` uses `id` as a key function, it knows what is entering/exiting when we filter, then rebind the data

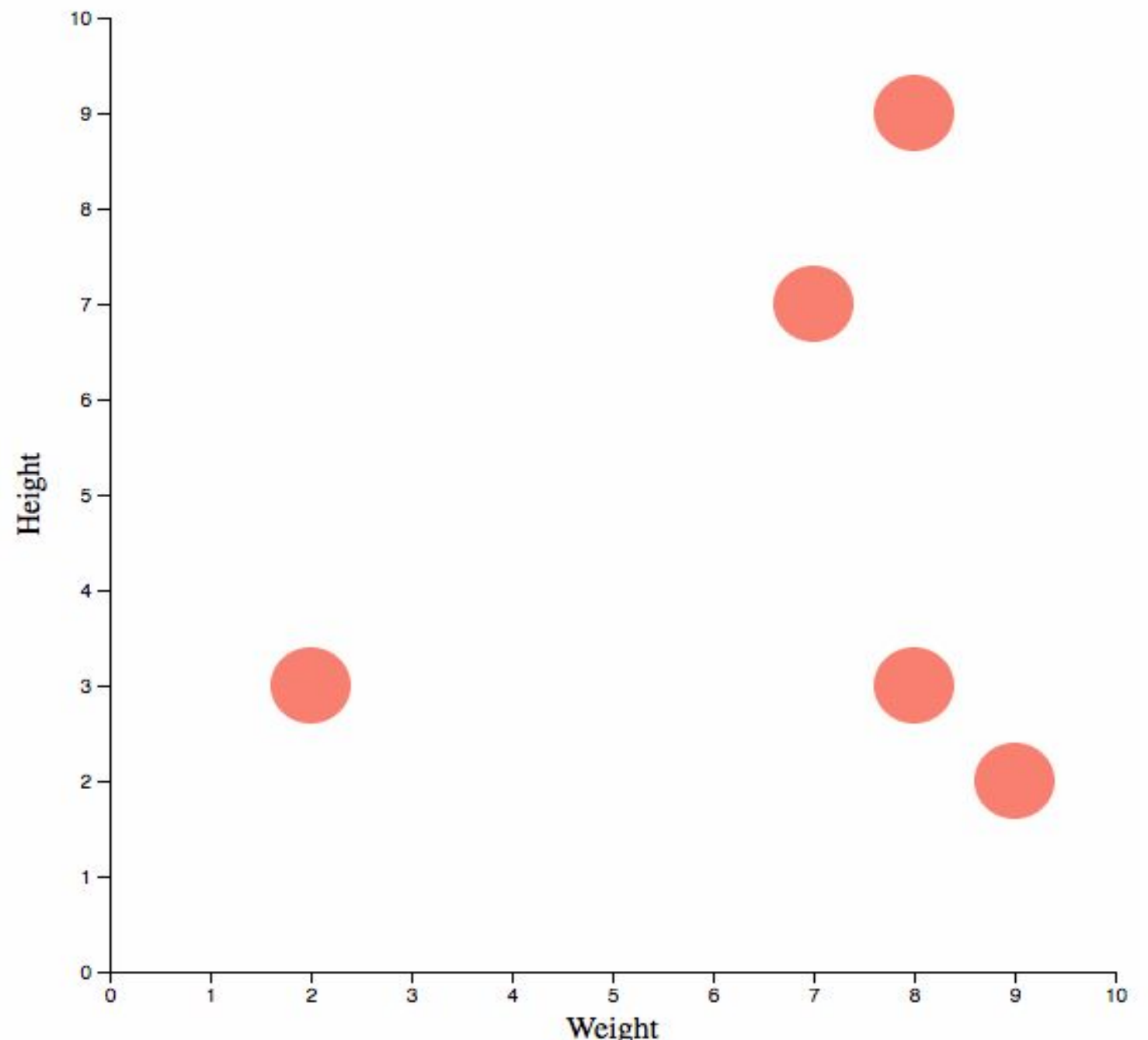
```
// Bind our animal data to the circles, using the "id" field as our key  
let updatedCircles = circles.data(animalData, d => d.id);
```

- You *could* just remove all circles, then redraw new ones based on your filtered data.
- But the strength of data joining is that D3 can see what's changed between your existing selection's data and the new data.
- ... and operate efficiently, only on the elements that have changed.

Height vs. Weight for Cats & Dogs

Fancy filters:

Cats Only Dogs Only Both



Tips

- Browser showing old code?
cmd-shift-r, hard refresh and clear the cache.
- Use the inspector console.
`console.log()` is your friend,
- Reset your server, if all else fails.
- Avoid using a framework.
Unnecessary complexity.
- Use HTML inputs as needed
- CSS can simplify simple interactions

```
.circle:hover {  
  fill: yellow;  
}
```

Other things D3 can do for you

Different scales

- Ordinal scale (discrete domain and range)
- Colors as range
 - Interpolation between colors for diverging scales
 - Predefined discrete color schemes
- <https://github.com/d3/d3-scale/blob/master/README.md>

d3.schemeCategory10 <>



An array of ten categorical colors represented as RGB hexadecimal strings.

d3.schemeCategory20 <>



An array of twenty categorical colors represented as RGB hexadecimal strings.

d3.schemeCategory20b <>



An array of twenty categorical colors represented as RGB hexadecimal strings.

d3.schemeCategory20c <>

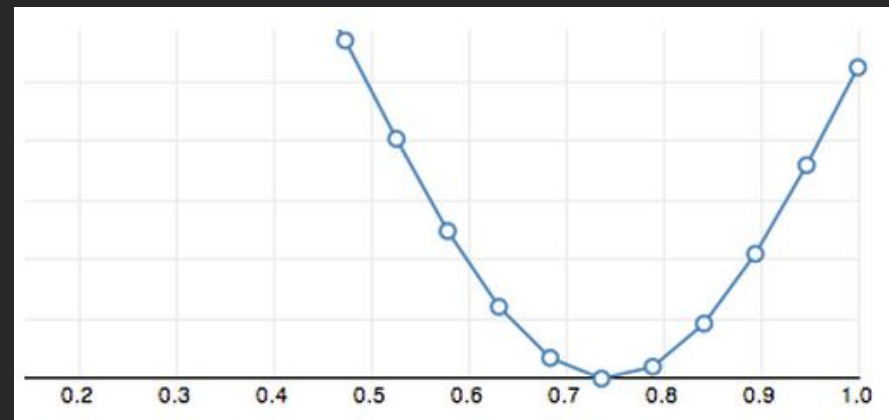


An array of twenty categorical colors represented as RGB hexadecimal strings. This color scale includes color specifications and designs developed by Cynthia Brewer (colorbrewer2.org).

Paths and areas

d3.line

<https://github.com/d3/d3-shape/blob/master/README.md#line>

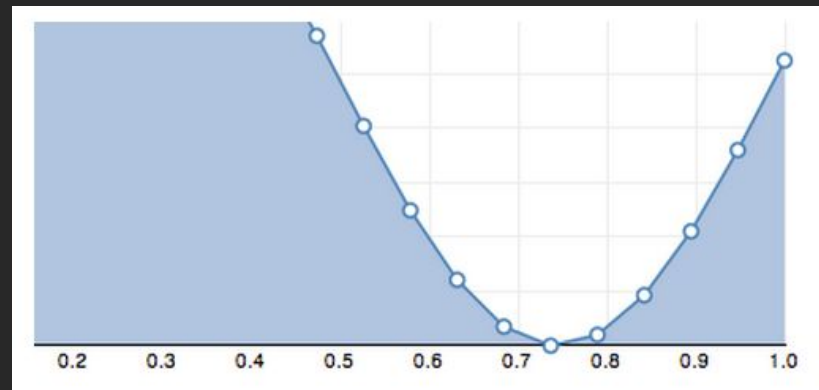



```
<path d="M152.64962091501462,320.5600780855698L133.88913955606318,325.4363177123538L134.96890954443046,330.37917634921996L131.19348249532786,331.158393614812L98.56681109628815,335.53933807857004L91.14450799488135,333.79662025279L72.1880101321918,333.74733970068166L69.51723455785742,332.8569681440152L62.37313911354066,333.2100666843387L62.248334309137434,335.3677272708405L58.843440998888326,335.0574959605036L53.97667317214221,331.36075125633175L56.30952738
```

Paths and areas

d3.area

<https://github.com/d3/d3-shape/blob/master/README.md#area>



Paths and areas

d3.geoPath

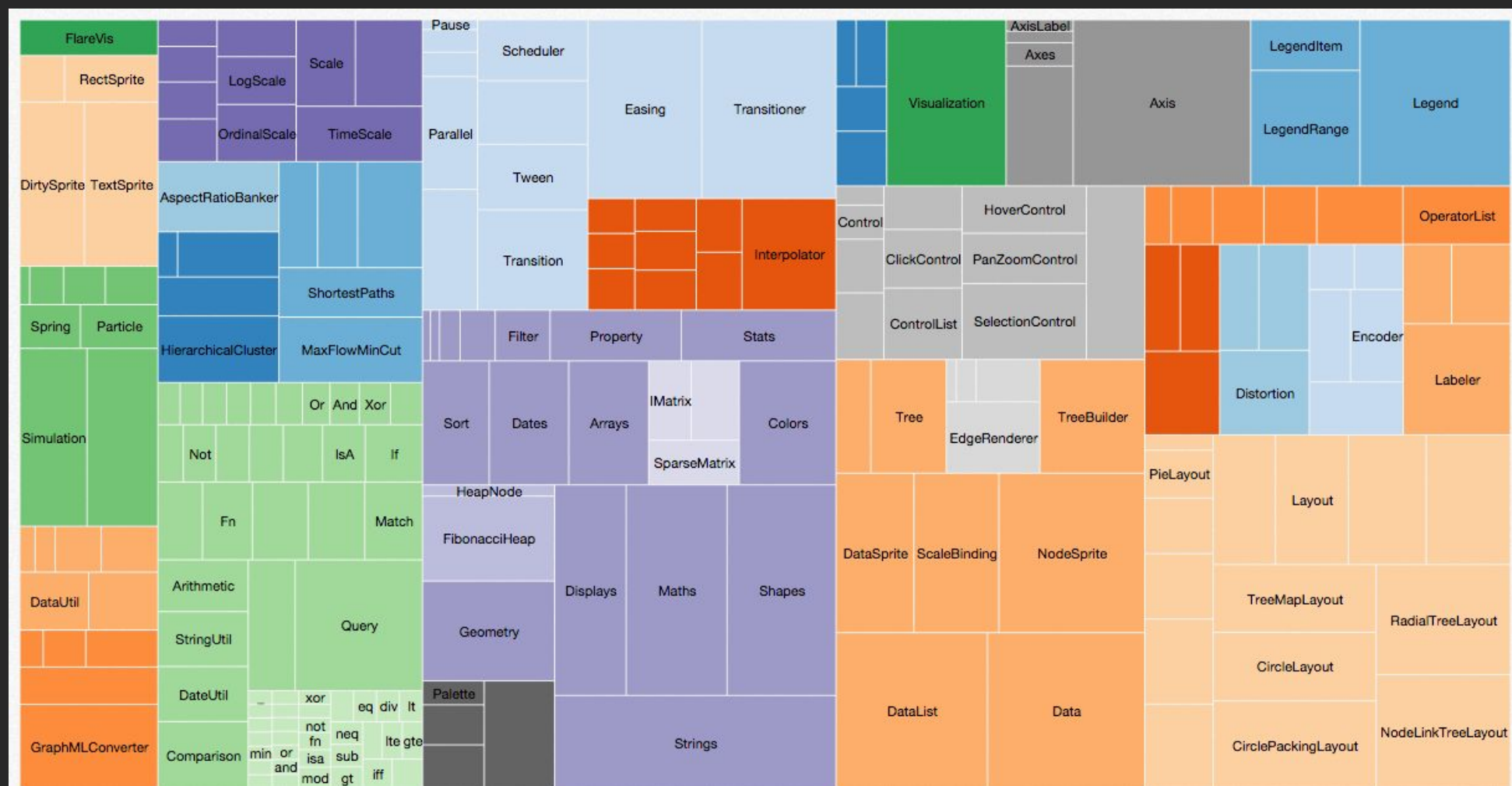
<https://github.com/d3/d3-geo/blob/master/README.md#geoPath>



Hierarchical layouts

d3.treemap

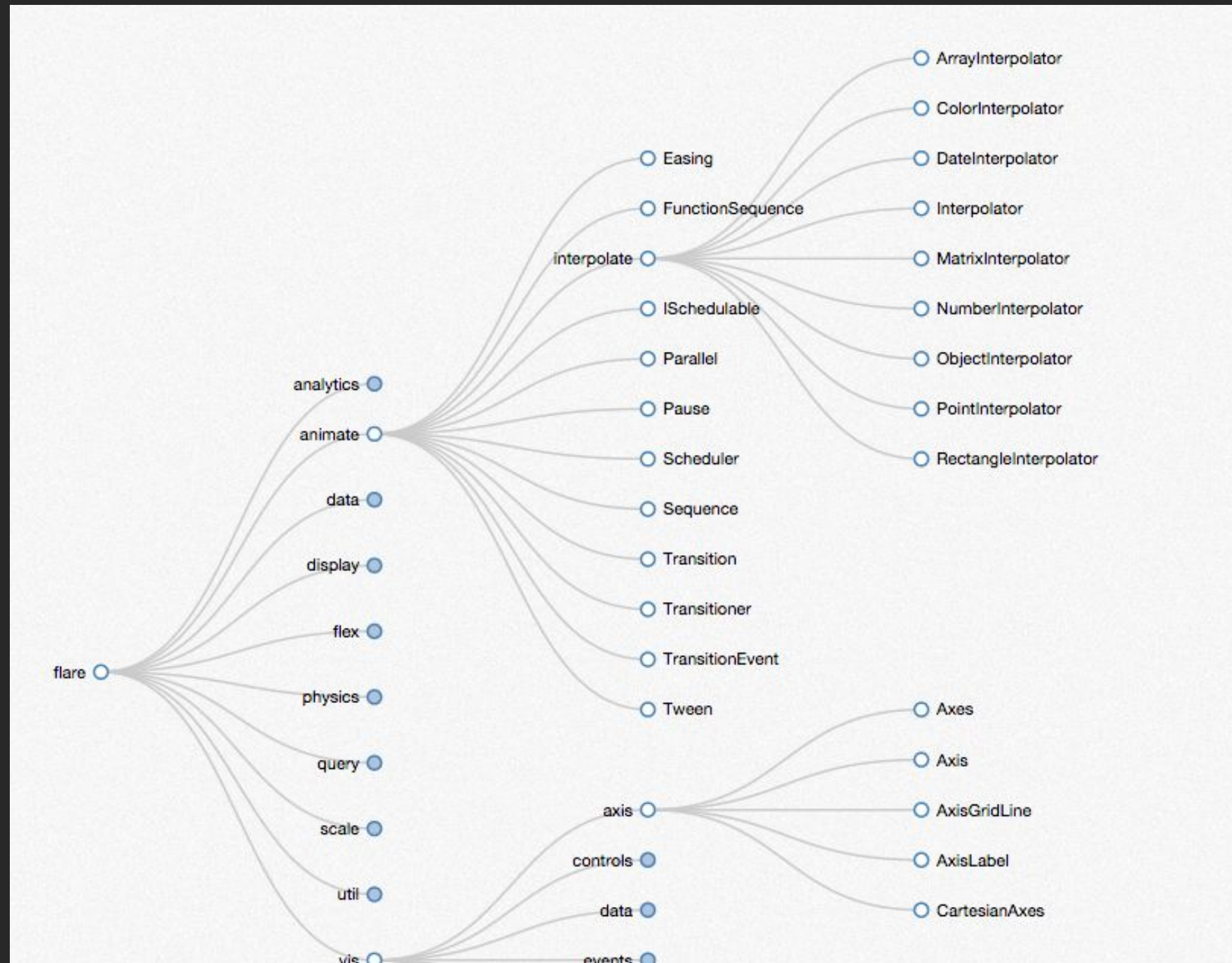
<https://github.com/d3/d3-hierarchy/blob/master/README.md#treemap>



Hierarchical layouts

d3.tree

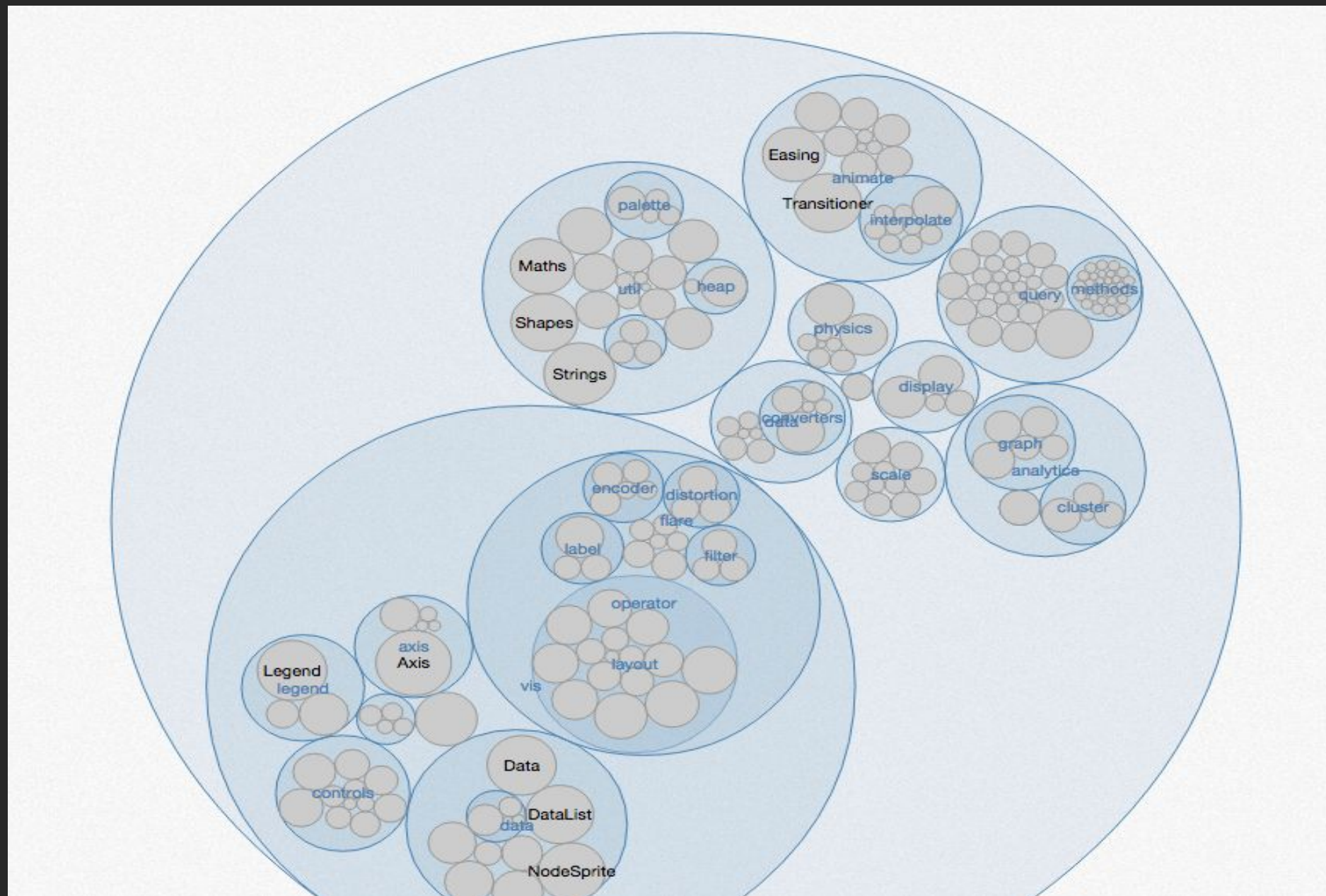
<https://github.com/d3/d3-hierarchy/blob/master/README.md#tree>



Hierarchical layouts

d3.pack

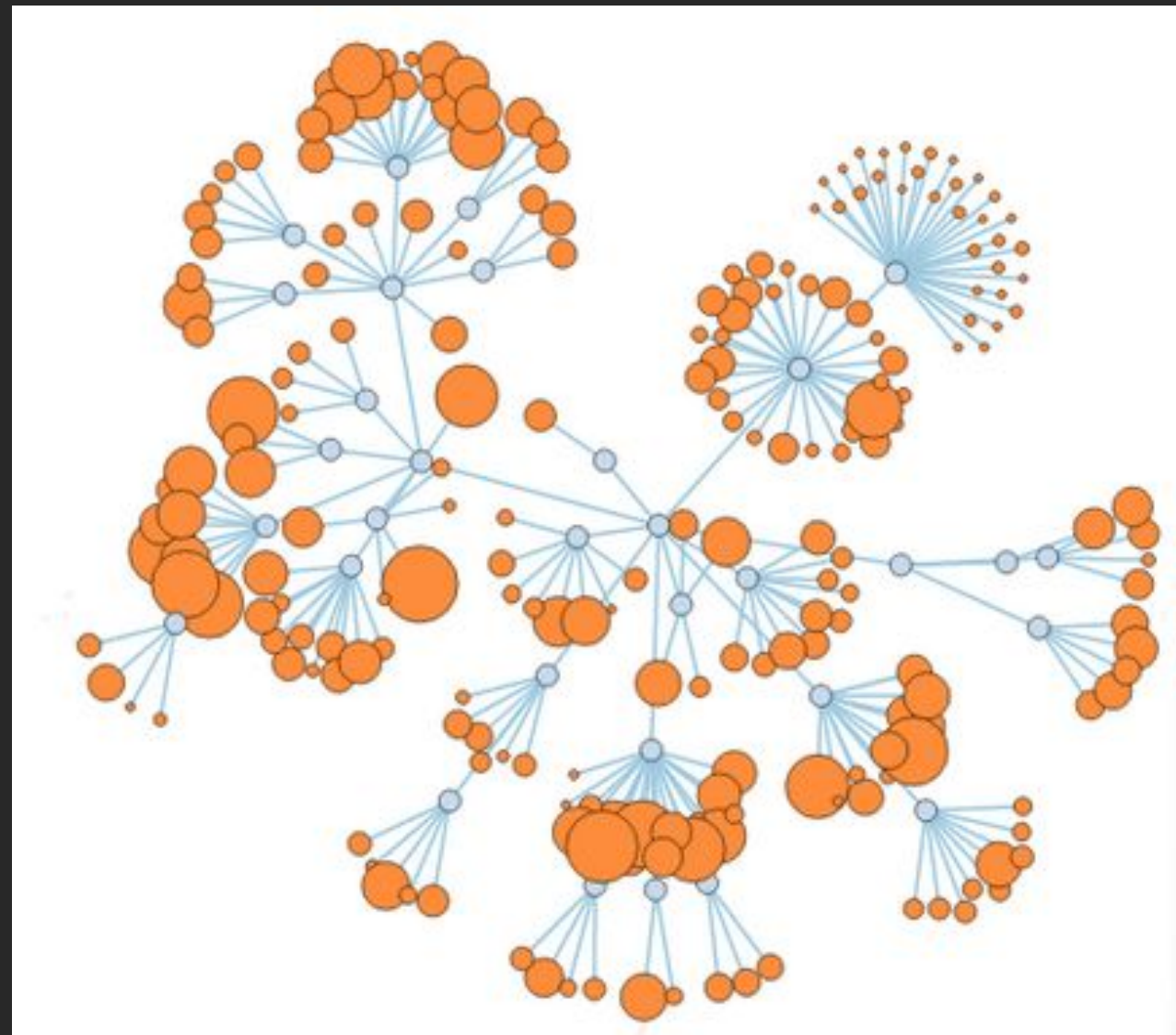
<https://github.com/d3/d3-hierarchy/blob/master/README.md#pack>



Network layout

d3.forceSimulation

<https://github.com/d3/d3-force>



Resources

D3 API Documentation at

<https://github.com/d3/d3/blob/master/API.md>

Example code

- Mike Bostock

- Drag + zoom:

- <https://bl.ocks.org/mbostock/3127661b6f13f9316be745e77fdfb084>

- Also Scott Murray, Jerome Cukier

Slack team: <https://d3-slackin.herokuapp.com/>

❖ More resources on the website homepage and Piazza!