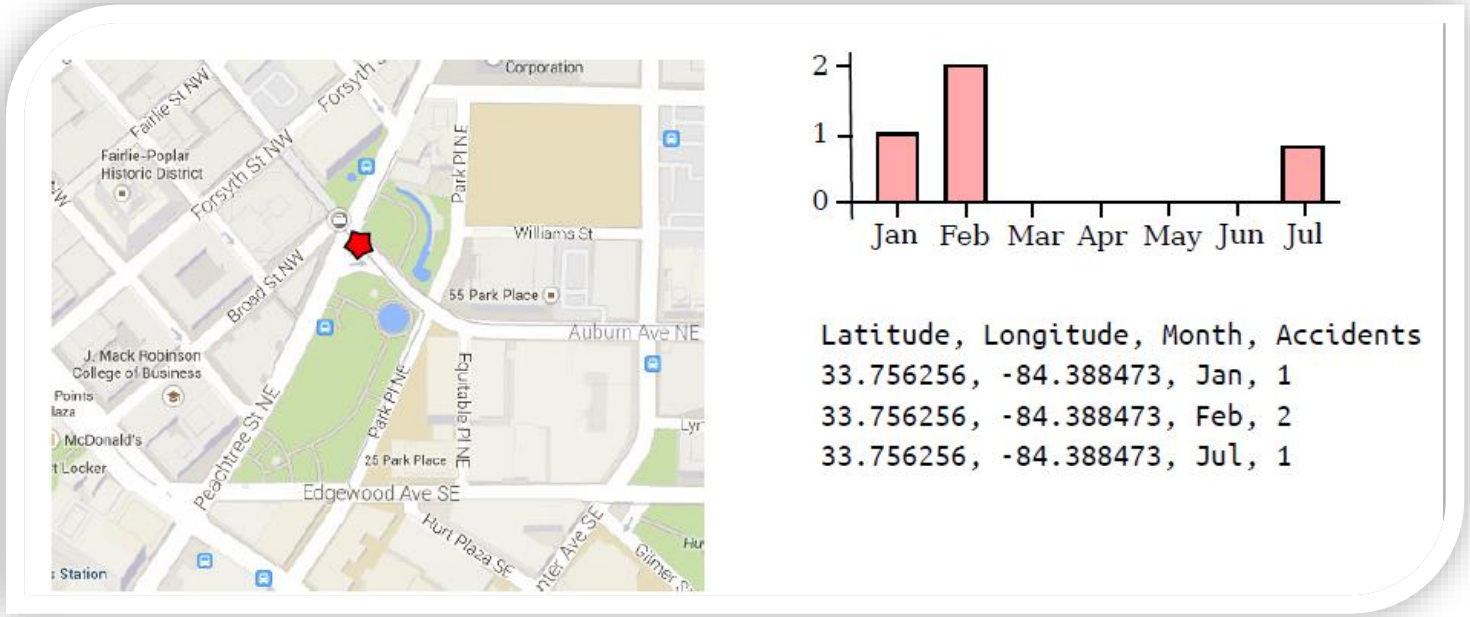
Introduction to d3.js

Shamal AL-Dohuki

Overview

- Data visualization is the presentation of data in a pictorial or graphical format.



Overview Cont.

- Reasons to use:
  - helps people see things that were not obvious to them before;
  - patterns can be spotted quickly and easily;
  - conveys information in a universal manner;
  - answer questions like “What would happen if we made an adjustment to that area?”.

What is d3.js

A JavaScript library which:

- Draws charts
- Visualizes data
- Doesn’t provide pre-defined charts
- Can be used to develop the real time dashboards
- Can’t be used to draw the 3-D charts

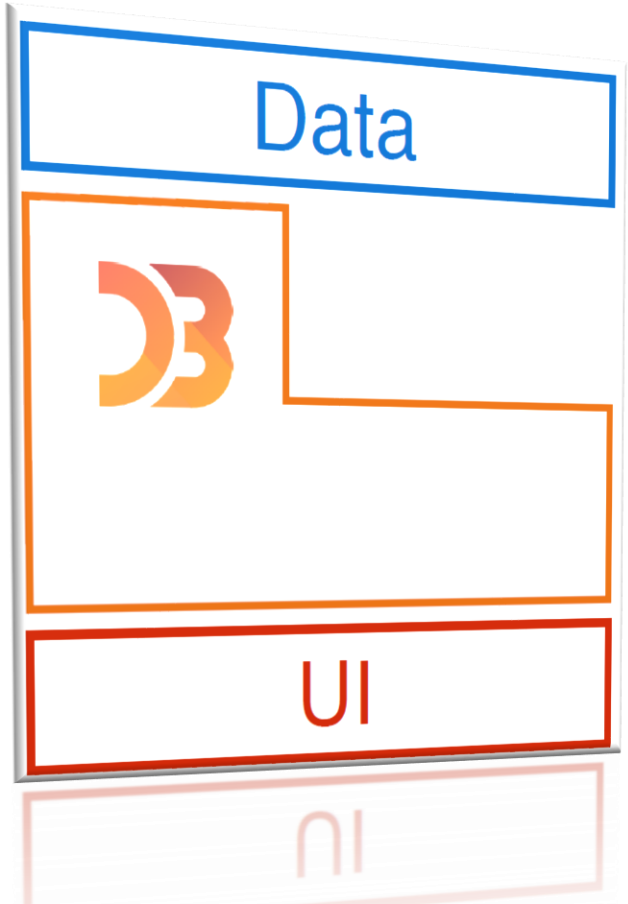
What is d3.js Cont.

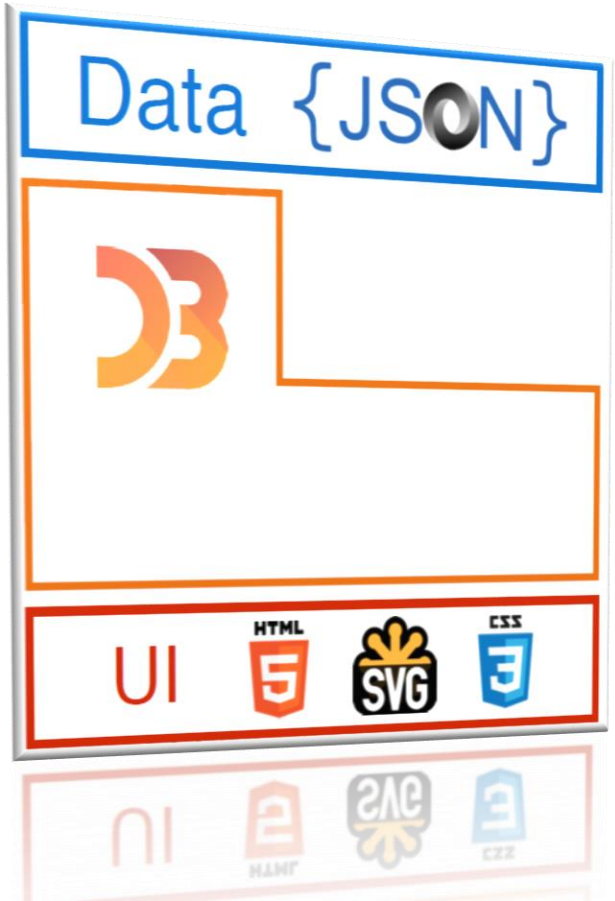
- D3 - Data-Driven Documents

- Data: Provided by you
- Driven: d3 connects data to documents
- Documents: web-based documents

Architecture

D3 - **D**ata-**D**riven **D**ocuments





D3.js is good at

- Providing a way to map data to documents.
- Being a general purpose visualization library
- Handling data transformation
- Providing basic math & layout algorithms

D3.js alternatives

- Cytoscape.js
- C3.js
- Canvas.js
- Datacopia
- Panxpan
- See more: alternativeto.net

Before We Start

Before we start, we need the following components:

- D3.js library
- Editor
- Web browser
- Web server

Prerequisites

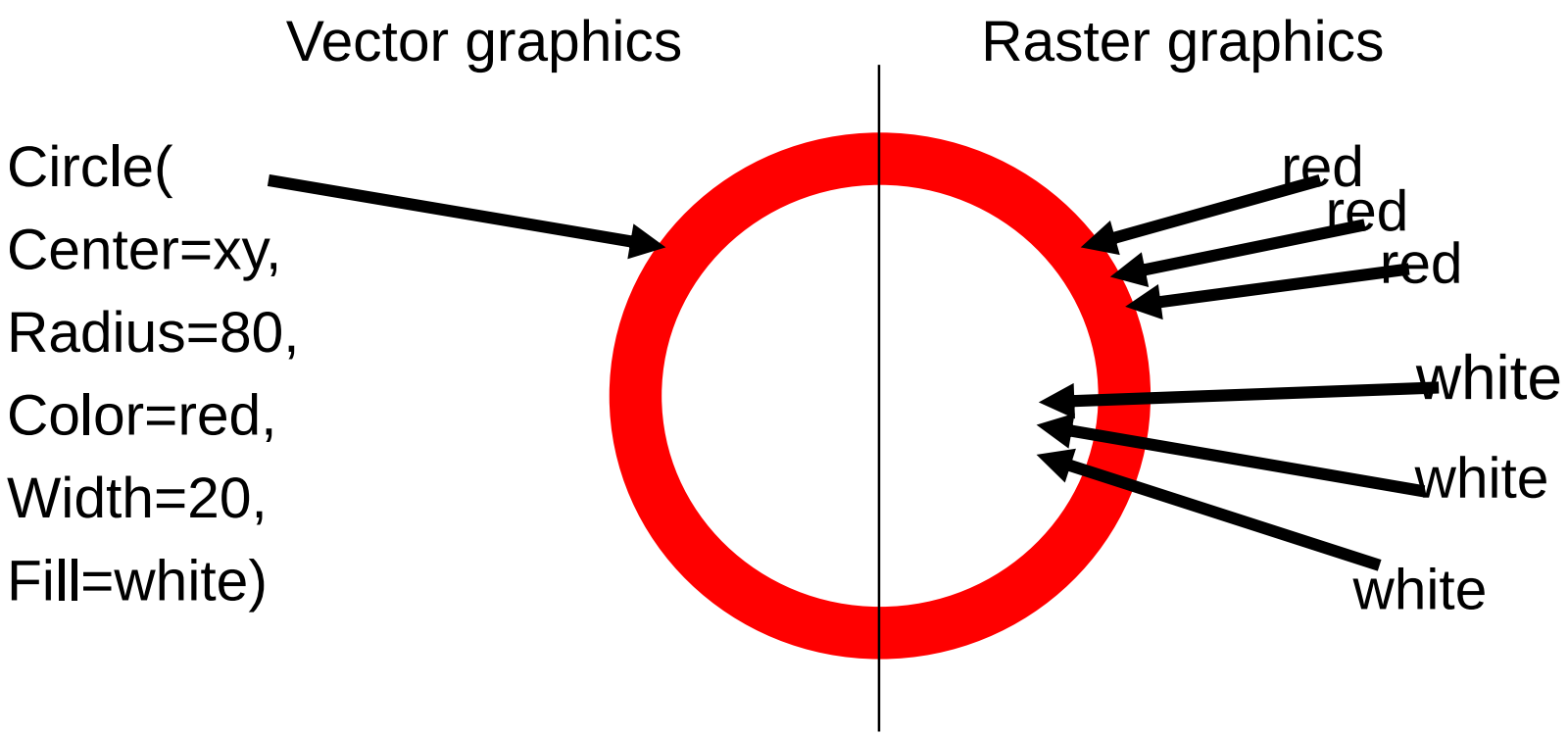
- HTML
- CSS
- Scalable Vector Graphics (SVG)
- Document Object Model (DOM)
- Some knowledge of JavaScript
- jQuery is a bonus

Scalable Vector Graphics (SVG)

- There are two main ways to represent graphics on the Web:
- Bitmapped Graphics - storing the RGB values of each pixel in the image.
- Vector Graphics - storing the coordinates of each vectors and the colors in which they are rendered.

(SVG) Cont.

- Vector and raster graphics



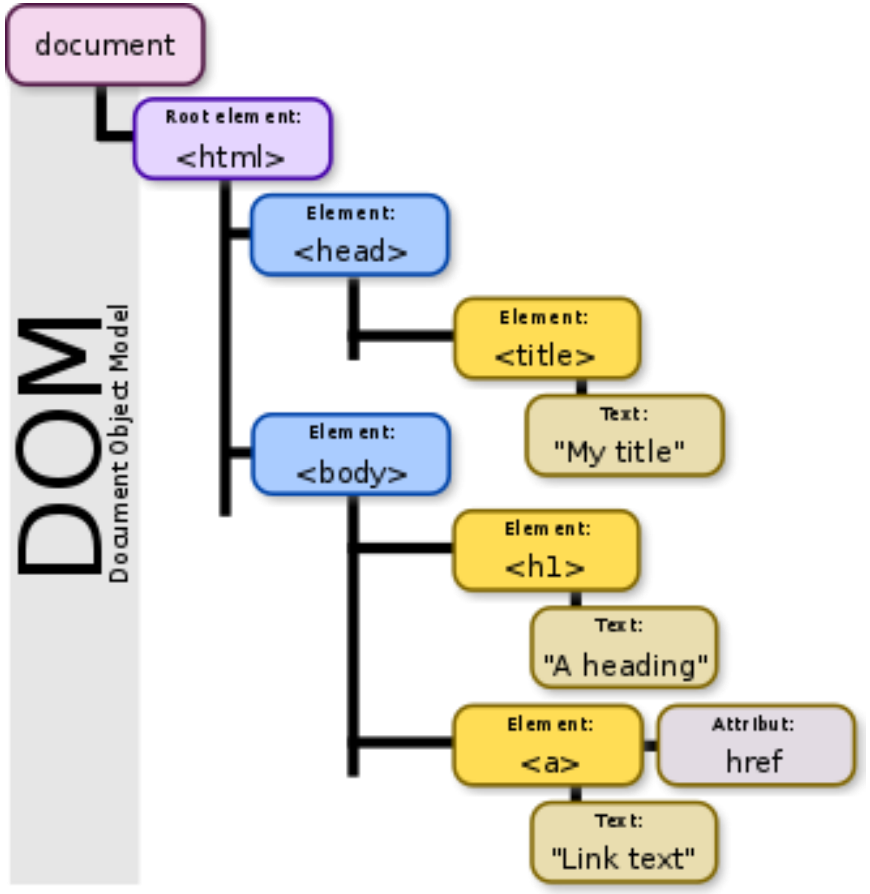
(SVG) Cont.

- SVG is a way to render images on the webpage.
- SVG is not a direct image, but is just a way to create images using text.
- It is a **Scalable Vector**. It scales itself according to the size of the browser, so resizing your browser will not distort the image.

(SVG) Cont.

Document Object Model (DOM)

- The Document Object Model (DOM) is a cross-platform and language-independent application programming interface that treats an HTML, XHTML, or XML document as a tree structure wherein each node is an object representing a part of the document.

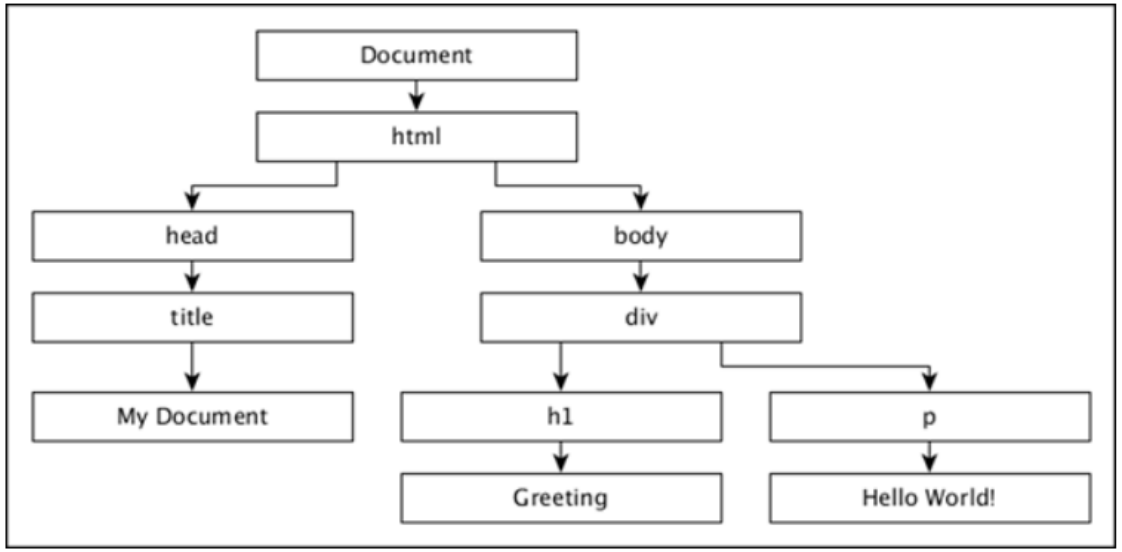


Example of DOM hierarchy in an HTML document

DOM Example

```
<!DOCTYPE html>
<html lang = "en">
   <head>
      <title>My Document</title>
   </head>

   <body>
      <div>
         <h1>Greeting</h1>
         <p>Hello World!</p>
      </div>
   </body>
</html>
```

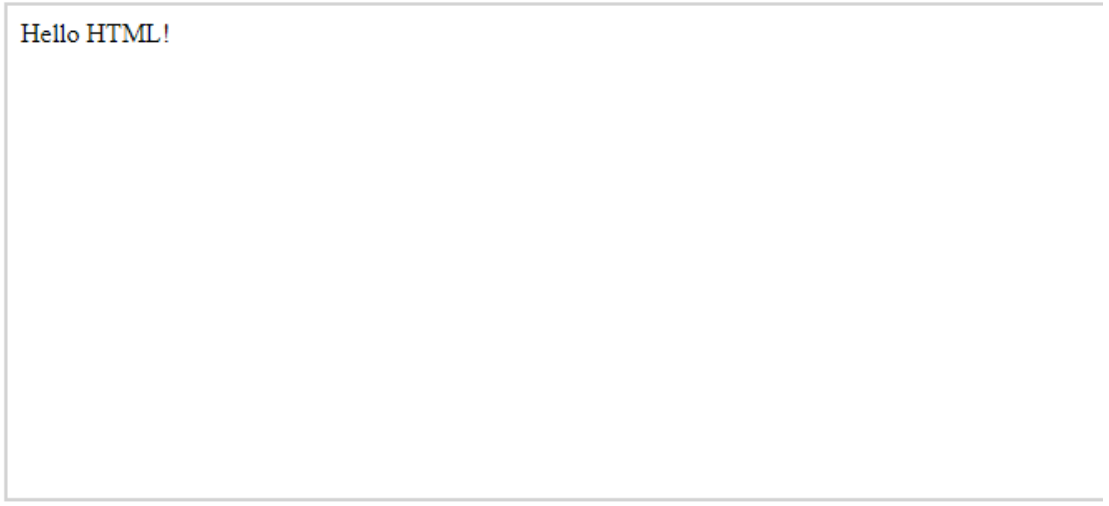
The document object model of the above HTML document is as follows,



Hello HTML

index.html

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Basic HTML Page</title>
  </head>
  <body>
    Hello HTML!
  </body>
</html>
```



Hello SVG

index.html

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>SVG Example</title>
  </head>
  <body>

    <svg>
      <rect width="100" height="100"></rect>
    </svg>

  </body>
</html>
```

Hello SVG Cont.

```
<svg width="400" height="200">
    <circle cx="100" cy="100" r="10"></circle>
    <circle cx="200" cy="100" r="30" fill="orange"></circle>
    <circle cx="300" cy="100" r="20" fill="olivedrab"></circle>
</svg>
```

How D3 Works

```
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>D3 Examples</title>
  <script src="http://d3js.org/d3.v3.min.js">
  </script>
</head>
<body>
</body>
</html>
```

How D3 Works Cont.

```
<!DOCTYPE html>
<html lang = "en">
    <head>
        <script src = "https://d3js.org/d3.v4.min.js"></script>
    </head>

    <body>
        <script>
            // write your d3 code here..
        </script>
    </body>
</html>
```

Core D3 ideas: Selecting elements

- **Selecting elements:** **d3.select()** and **d3.selectAll()** can be used to access DOM elements by name, class, id, or many other css selectors.
- **d3.select()** selects only the first element that matches the css selectors while **d3.selectAll()** selects all matched elements.

Selecting Elements

- A selection is an array of elements pulled from the current document.
- After selecting elements, you apply operators to them to do stuff.
- These operators can get or set attributes, styles, properties, HTML and text content.

```
▼<svg class="chart" width="500" height="500">
  ▶<circle r="5" cx="60" cy="50.636363636363626">…</circle>
  ▶<circle r="5" cx="136" cy="489.70545454545453">…</circle>
  ▶<circle r="5" cx="212" cy="472.8181818181818">…</circle>
  ▶<circle r="5" cx="288" cy="35">…</circle>
  ▶<circle r="5" cx="364" cy="113.18181818181819">…</circle>
  </svg>
```

```
> d3.select(".chart")
< [▼Array[1] i ]
    ▶0: svg
      length: 1
    ▶parentNode: html
    ▶__proto__: Array[0]
```

```
> d3.select("svg")
< [▼Array[1] i ]
    ▶0: svg
      length: 1
    ▶parentNode: html
    ▶__proto__: Array[0]
```

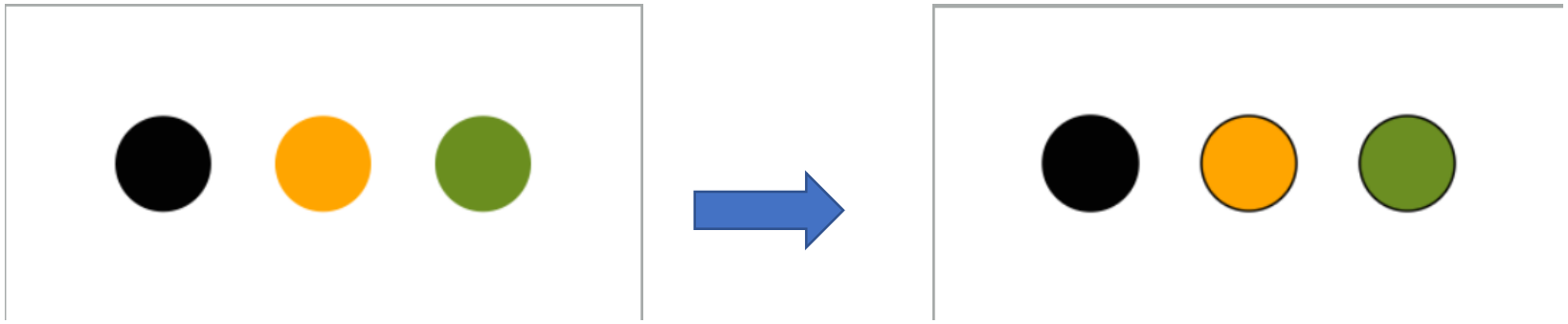
```
> d3.selectAll("circle")
< [▼Array[5] i ]
    ▶0: circle
    ▶1: circle
    ▶2: circle
    ▶3: circle
    ▶4: circle
      length: 5
    ▶parentNode: html
    ▶__proto__: Array[0]
```

Selecting Elements: Example

```
<svg width="400" height="200">
    <circle cx="100" cy="100" r="10"></circle>
    <circle cx="200" cy="100" r="30" fill="orange"></circle>
    <circle cx="300" cy="100" r="20" fill="olivedrab"></circle>
</svg>
<script>
    var circles = d3.selectAll("circle");
    circles.attr("r", 30);
</script>
```

Selecting Elements: Example Cont.

```
<script>
    var circles = d3.selectAll("circle");
    circles
        .attr("r", 30)
        .attr("stroke", "black")
        .attr("stroke-width", 1.5);
</script>
```



Select with jQuery

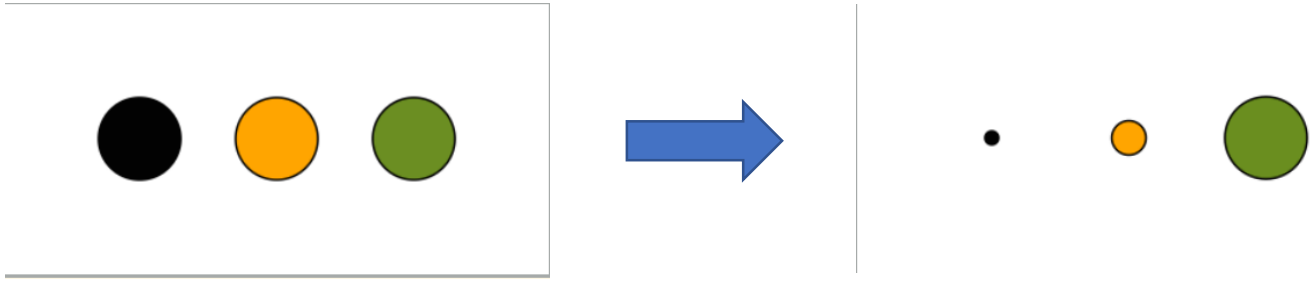
```
<script>
  var circles = $("circle");
  circles
    .attr("r", 30)
    .attr("stroke", "black")
    .attr("stroke-width", 1.5);
</script>
```

Core D3 ideas: Data Binding

- **Data Binding:** We can use the **data()** function to **bind** data to a **selection**.
- We can also use **data()** or **datum()** to access the data that belong to a selection.

Data Binding: Example

```
<script>
   var circles = d3.selectAll("circle");
   var sizes = [10, 25, 60];
   circles
      .data(sizes)
      .attr("r", function(size)
      { return size / 2; });
</script>
```



External data files

- **d3.csv** - request a comma-separated values (CSV) file.
- **d3.html** - request an HTML document fragment.
- **d3.json** - request a JSON blob.
- **d3.text** - request a text file.
- **d3.tsv** - request a tab-separated values (TSV) file.
- **d3.xhr** - request a resource using XMLHttpRequest.
- **d3.xml** - request an XML document fragment.

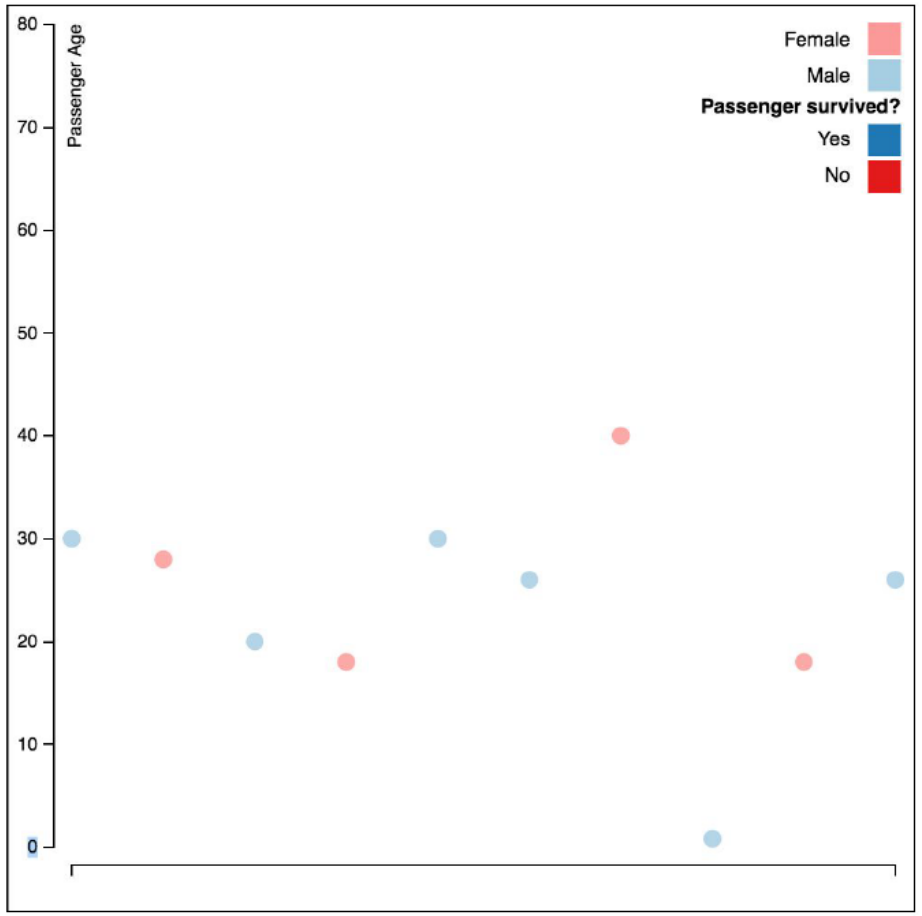
Example

Titanic Passengers

Example: Dataset

pclass	survived	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked	boat	body	home.dest
1	1	Allen, Miss. Elisabeth Walton	female	29	0	0	24160	211.3375	B5	S	2		St Louis, MO
1	1	Allison, Master. Hudson Trevor	male	0.92	1	2	113781	151.55	C22 C26	S	11		Montreal, PQ / Chesterville, ON
1	0	Allison, Miss. Helen Loraine	female	2	1	2	113781	151.55	C22 C26	S			Montreal, PQ / Chesterville, ON
1	0	Allison, Mr. Hudson Joshua Creighton	male	30	1	2	113781	151.55	C22 C26	S		135	Montreal, PQ / Chesterville, ON
1	0	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25	1	2	113781	151.55	C22 C26	S			Montreal, PQ / Chesterville, ON
1	1	Anderson, Mr. Harry	male	48	0	0	19952	26.55	E12	S	3		New York, NY
1	1	Andrews, Miss. Kornelia Theodosia	female	63	1	0	13502	77.9583	D7	S	10		Hudson, NY
1	0	Andrews, Mr. Thomas Jr	male	39	0	0	112050	0	A36	S			Belfast, NI
1	1	Appleton, Mrs. Edward Dale (Charlotte Lamson)	female	53	2	0	11769	51.4792	C101	S	D		Bayside, Queens, NY
1	0	Artagaveytia, Mr. Ramon	male	71	0	0	PC 17609	49.5042		C		22	Montevideo, Uruguay
1	0	Astor, Col. John Jacob	male	47	1	0	PC 17757	227.525	C62 C64	C		124	New York, NY
1	1	Astor, Mrs. John Jacob (Madeleine Talmadge Force)	female	18	1	0	PC 17757	227.525	C62 C64	C	4		New York, NY
1	1	Aubart, Mme. Leontine Pauline	female	24	0	0	PC 17477	69.3	B35	C	9		Paris, France
1	1	Barber, Miss. Ellen "Nellie"	female	26	0	0	19877	78.85		S	6		
1	1	Barkworth, Mr. Algernon Henry Wilson	male	80	0	0	27042	30	A23	S	B		Hessle, Yorks
1	0	Baumann, Mr. John D	male		0	0	PC 17318	25.925		S			New York, NY
1	0	Baxter, Mr. Quigg Edmond	male	24	0	1	PC 17558	247.5208	B58 B60	C			Montreal, PQ
1	1	Baxter, Mrs. James (Helene DeLaudeniere Chaput)	female	50	0	1	PC 17558	247.5208	B58 B60	C	6		Montreal, PQ
1	1	Bazzani, Miss. Albina	female	32	0	0	11813	76.2917	D15	C	8		
1	0	Beattie, Mr. Thomson	male	36	0	0	13050	75.2417	C6	C	A		Winnipeg, MN
1	1	Beckwith, Mr. Richard Leonard	male	37	1	1	11751	52.5542	D35	S	5		New York, NY
1	1	Beckwith, Mrs. Richard Leonard (Sallie Monypeny)	female	47	1	1	11751	52.5542	D35	S	5		New York, NY
1	1	Behr, Mr. Karl Howell	male	26	0	0	111369	30	C148	C	5		New York, NY
1	1	Bidois, Miss. Rosalie	female	42	0	0	PC 17757	227.525		C	4		
1	1	Bird, Miss. Ellen	female	29	0	0	PC 17483	221.7792	C97	S	8		
1	0	Birnbaum, Mr. Jakob	male	25	0	0	13905	26		C		148	San Francisco, CA
1	1	Bishop, Mr. Dickinson H	male	25	1	0	11967	91.0792	B49	C	7		Dowagiac, MI
1	1	Bishop, Mrs. Dickinson H (Helen Walton)	female	19	1	0	11967	91.0792	B49	C	7		Dowagiac, MI
1	1	Bissette, Miss. Amelia	female	35	0	0	PC 17760	135.6333	C99	S	8		
1	1	Bjornstrom-Steffansson, Mr. Mauritz Hakan	male	28	0	0	110564	26.55	C52	S	D		Stockholm, Sweden / Washington, DC
1	0	Blackwell, Mr. Stephen Weart	male	45	0	0	113784	35.5	T	S			Trenton, NJ
1	1	Blank, Mr. Henry	male	40	0	0	112277	31	A31	C	7		Glen Ridge, NJ
1	1	Bonnell, Miss. Caroline	female	30	0	0	36928	164.8667	C7	S	8		Youngstown, OH
1	1	Bonnell, Miss. Elizabeth	female	58	0	0	113783	26.55	C103	S	8		Birkdale, England Cleveland, Ohio

Example: Visualization



Example: Getting Started

- **index.html**
- We can start by defining a simple web page, which has a header (h1) and an svg element that will hold our visualization. In the style tags, we can add CSS styling for both elements defined in the HTML.

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

<title>D3 Example</title>
<style>
h1 {
  font-family: sans-serif;
  text-align: center;
}

svg {
  display: block;
  margin-left: auto;
  margin-right: auto;
  border: 1px solid black;
}

</style>
<script src="https://d3js.org/d3.v4.min.js" charset="utf-8"></script>
</head>
```

CSS Style

```
<body>
<h1>D3 Example</h1>

<svg class="chart" width="500px" height="500px">
</svg>



</body>
```

HTML

Example: Adding Elements

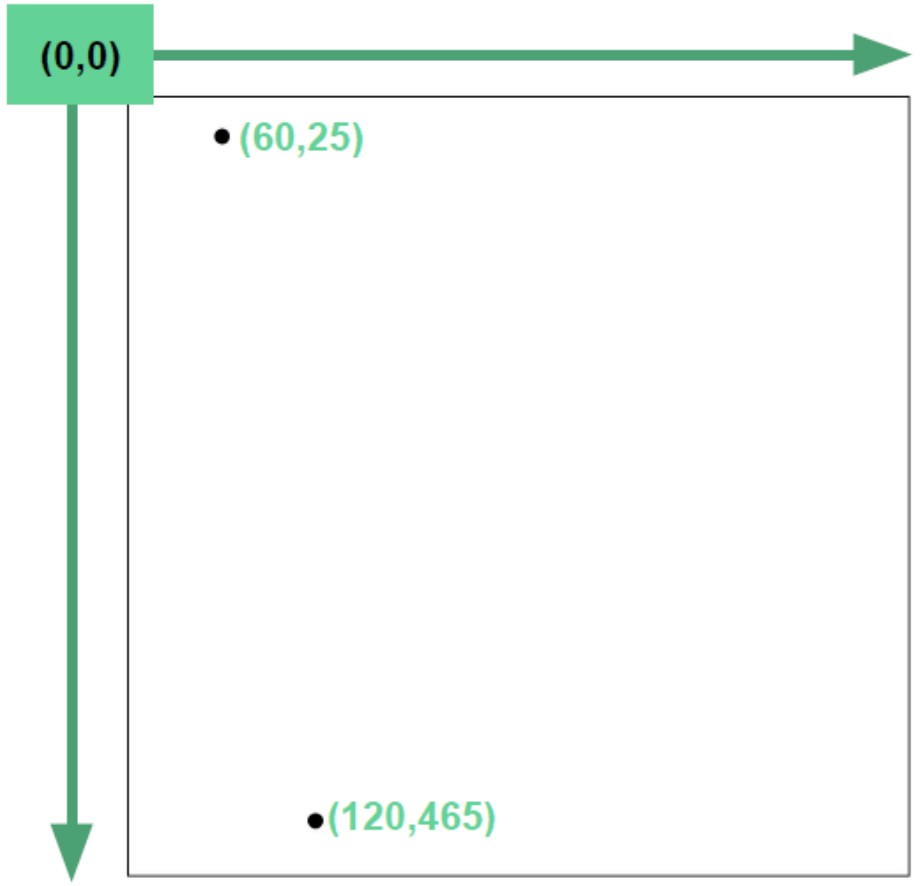
- **Manually specifying elements**
- We can manually add elements to the DOM, and specify properties such as the x and y position, and the title (which appears as a tooltip on hover).

```
<circle cx="60px" cy="25" r="5">
<title>Allen, Miss. Elisabeth Walton</title>
</circle>

<circle cx="120px" cy="465" r="5">
<title>Allison, Master. Hudson Trevor</title>
</circle>
```

Example: Adding Elements Cont.

- **Positioning Elements**
- Keep in mind that the origin for positioning elements is the upper left corner.

```
<circle cx="60px" cy="25" r="5">
<title>Allen, Miss. Elisabeth Walton</title>
</circle>

<circle cx="120px" cy="465" r="5">
<title>Allison, Master. Hudson Trevor</title>
</circle>
```



Example: Selections

- **Selecting elements**
- **d3.select()** and **d3.selectAll()** can be used to access DOM elements by name, class, id, or many other css selectors. **d3.select()** selects only the first element that matches the css selectors while **d3.selectAll()** selects all matched elements.

Example: Selections Cont.

```
▼<svg class="chart" width="500" height="500">
  ▶<circle r="5" cx="60" cy="50.636363636363626">…</circle>
  ▶<circle r="5" cx="136" cy="489.70545454545453">…</circle>
  ▶<circle r="5" cx="212" cy="472.8181818181818">…</circle>
  ▶<circle r="5" cx="288" cy="35">…</circle>
  ▶<circle r="5" cx="364" cy="113.18181818181819">…</circle>
  </svg>
```

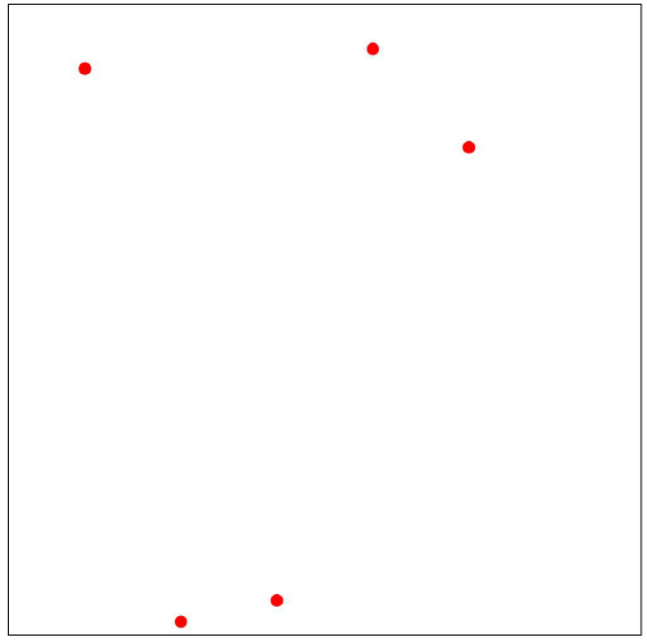
```
> d3.select(".chart")
<- [▼Array[1] i ]
    ▶0: svg
     length: 1
    ▶parentNode: html
    ▶__proto__: Array[0]
```

```
> d3.select("svg")
<- [▼Array[1] i ]
    ▶0: svg
     length: 1
    ▶parentNode: html
    ▶__proto__: Array[0]
```

```
> d3.selectAll("circle")
<- [▼Array[5] i ]
    ▶0: circle
    ▶1: circle
    ▶2: circle
    ▶3: circle
    ▶4: circle
     length: 5
    ▶parentNode: html
    ▶__proto__: Array[0]
```

Example: Selections Cont.

- **Modifying selected elements**
- You can use access and modify the properties of selections with **attr()**, **text()**, **style()**, and other **operators**. Most D3 selection methods return the selection, allowing us to chain the operator calls.



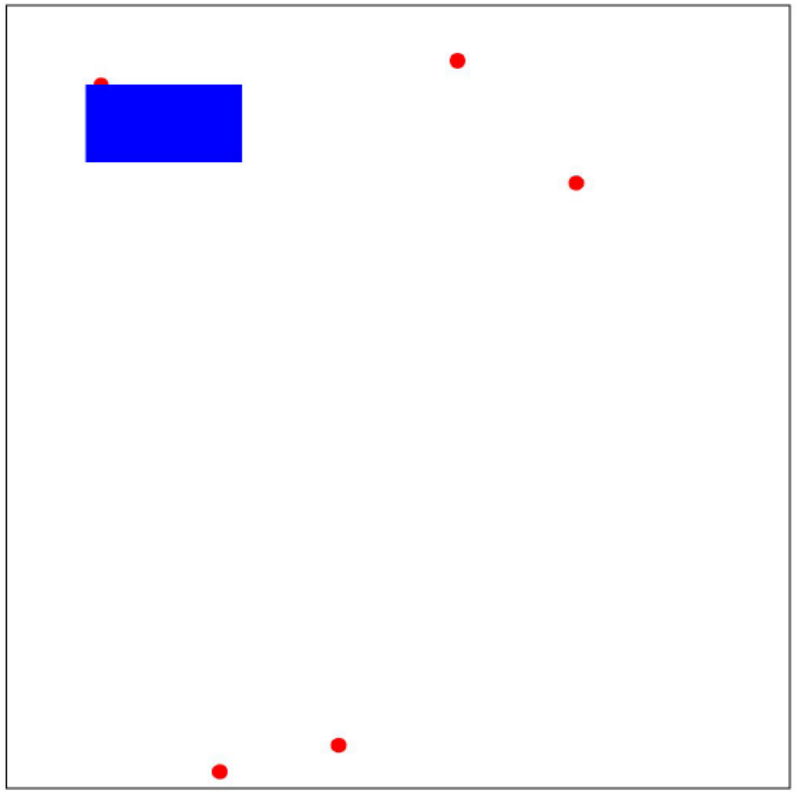
```
> d3.selectAll("circle").attr("fill","#ff0000")
> d3.select("h1").text("Hello World")
```

Example: Selections Cont.

- **Appending elements**
- Through **append()**, we can add new elements anywhere in the DOM.
- We can then use operators or CSS to set the properties of the element.
- We can also get rid of elements with **remove()**.
- Finally, we can store selections in variables for future use.

Example: Selections Cont.

```
> d3.select("svg").append("rect").attr("x",50).attr("y",50).attr("width",100).attr("height",50).attr("fill","#0000ff")
```



Example: Data Binding

- **Binding**
- We can use the **data()** function to **bind** data to a **selection**.
- We can also use **data()** or **datum()** to access the data that belong to a selection.

Example: Data Binding Cont.

```
var sampleData = [
            {
            "pclass": 1,
            "survived": 1,
            "name": "Allen, Miss. Elisabeth Walton",
            "sex": "female",
            "age": 29,
            "sibsp": 0,
            "parch": 0,
            "ticket": 24160,
            "fare": 211.3375,
            "cabin": "B5",
            "embarked": "S",
            "boat": 2,
            "body": 0,
            "home.dest": "St Louis, MO"
            },
            {
            "pclass": 1,
            "survived": 1,
            "name": "Allison, Master. Hudson Trevor",
            "sex": "male",
            "age": 0.92,
            "sibsp": 1,
            "parch": 2,
            "ticket": 113781,
            "fare": 151.55,
            "cabin": "C22 C26",
            "embarked": "S",
            "boat": 11,
            "body": 0,
            "home.dest": "Montreal, PQ / Chesterville, ON"
            },
```

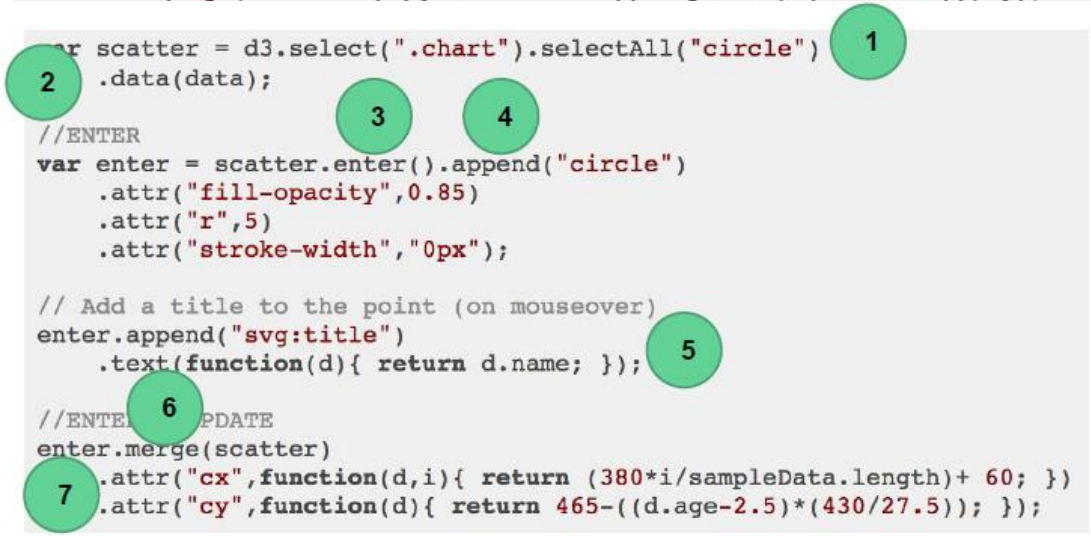
```
> d3.selectAll("circle").data()
< [▼Object i                                    , ▶Object, ▶Object, ▶Object, ▶Object]
    age: 29
    boat: 2
    body: 0
    cabin: "B5"
    embarked: "S"
    fare: 211.3375
    home.dest: "St Louis, MO"
    name: "Allen, Miss. Elisabeth Walton"
    parch: 0
    pclass: 1
    sex: "female"
    sibsp: 0
    survived: 1
    ticket: 24160
  ▶__proto__: Object
```

Example: Data Binding Cont.

selectAll().data().enter().append()

1. **Select** all of our circles (currently we don't have any).
2. **Bind** our data (in this case, 5 rows worth)
3. **Enter** each new datum from our selection.
4. **Append** a new DOM element. There are now 5 new elements, each with their own unique data.
5. **Append** titles to the new elements.
6. **Merge** our new elements into our original selections.
7. **Set** attributes with operators, using anonymous functions.

```
var scatter = d3.select(".chart").selectAll("circle")
    .data(data);

//ENTER
var enter = scatter.enter().append("circle")
    .attr("fill-opacity",0.85)
    .attr("r",5)
    .attr("stroke-width","0px");

// Add a title to the point (on mouseover)
enter.append("svg:title")
    .text(function(d){ return d.name; });

//ENTER + UPDATE
enter.merge(scatter)
    .attr("cx",function(d,i){ return (380*i/sampleData.length)+ 60; })
    .attr("cy",function(d){ return 465-((d.age-2.5)*(430/27.5)); });
```



Example: Scales

- **Specifying scales**
- To position the dots, we can manually specify the x and y position attributes, but this process can be tedious and error prone for complex attributes:

```
enter.merge(scatter)
    .attr("cx",function(d,i){ return (380*i/sampleData.length)+ 60; })
    .attr("cy",function(d){ return 465-((d.age-2.5)*(430/27.5)); });
```

Example: Scales Cont.

- **Specifying scales**
- **Scales** are functions that map from a domain to a range.
- Anonymous functions can be used to parameterize the element's attributes using the element's data. Anonymous functions can have two parameters **d** (our bound datum) and **i** (the index of our datum).

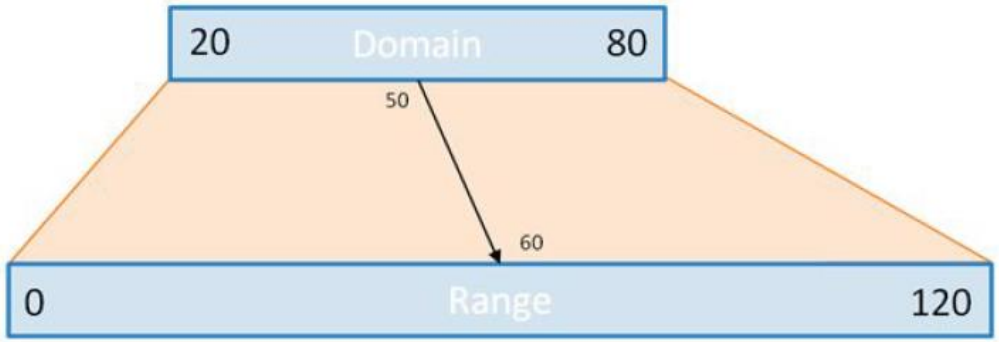
```
x = d3.scaleLinear()
    .domain([0,data.length-1])
    .range([60,440]);
```

Example: Scales Cont.

- **Specifying scales**
- Given a value from the domain, returns the corresponding value from the range.

```
var x = d3.scaleLinear()
    .domain([10, 130])
    .range([0, 960]);

x(20); // 80
x(50); // 320
```

Example: Scales Cont.



Using a scale:

```
enter.merge(scatter)
     .attr("cx",function(d,i){ return x(i); })
     .attr("cy",function(d){ return y(d.age); });
```

Manual specification:

```
enter.merge(scatter)
    .attr("cx",function(d,i){ return (380*i/sampleData.length)+ 60; })
    .attr("cy",function(d){ return 465-((d.age-2.5)*(430/27.5)); });
```

Example: Scales Cont.

- **More scale types**

- **d3.scaleLinear** create a linear mapping. You can also have **d3.scaleLog**, **d3.scaleSqrt**, and so on.
- You can also specify **ordinal** (which include nominal data types) and **temporal** scales.
- Note that the **range()** does not have to be a set of numbers; it can also be colors or strings.

```
// color
c = d3.scaleOrdinal()
    .domain(["male","female"])
    .range(["#a6cee3","#fb9a99"]);
```

Example: Scales Cont.

```
const data = [1, 2, 3, 4, 5];

const scaleLinear = d3.scaleLinear()
  .domain([0, Math.max(...data)]).range([1, 100]);

const scaleOrdinal = d3.scaleOrdinal()
  .domain(data).range(['one', 'two', 'three', 'four', 'five']);
```

Now we start calling them to see the result:

```
scaleLinear(1); //20
scaleOrdinal(1); //one

scaleLinear(2); //40
scaleOrdinal(2); //two

scaleLinear(5); //100
scaleOrdinal(5); //five
```

Example: Axes & Legends

- **Creating axes**
- Axes can be generated based on the scales in your visualization. Axes are defined based on their position using **d3.axisTop**, **d3.axisBottom**, **d3.axisRight**, or **d3.axisLeft**.
- Note: each of these constructors is a function; to create our axis, we create or select the element where we want to place it, and then use **call()** to apply the function to it.

Example: Axes & Legends Cont.

Scale:

```
y = d3.scaleLinear()
    .domain(d3.extent(sampleData,function(d){ return d.age; }))
    .range([465,10]);
```

Specify axis:

```
yAxis = d3.axisLeft()
      .scale(y);
```

Draw axis:

```
var yAxisGroup = canvas.append("g")
    .attr("class","axis")
    .attr("transform","translate(25,0)")
    .call(yAxis);
```

Example: Axes & Legends Cont.

- **Labeling axes**
- Labels can be added to your visualization by adding text marks. As with any other mark, you can programmatically specify both HTML attributes and CSS styles.

```
yAxisGroup.append("text")
    .text("Passenger Age")
    .attr("transform", "rotate(-90)")
    .attr("y", 15)
    .attr("dx", -10)
    .style("text-anchor", "end");
```

Example: Axes & Legends Cont.

- **Legends**
- Legends can be constructed just like the other elements of your visualization: by creating a new set of marks and using scales to style the attributes.
- In addition to the rect for the legend mark, we can append text to create the legend labels.

```
legend.append("rect")
    .attr("x", 475)
    .attr("y", 9)
    .attr("width", 18)
    .attr("height", 18)
    .style("fill", c);

legend.append("text")
    .attr("x", 465)
    .attr("y", 18)
    .attr("dy", ".35em")
    .style("text-anchor", "end")
    .text(function(d) {
      return d.charAt(0).toUpperCase()+d.slice(1);
    });
```

Example: Events & Transitions

- **Reacting to events**
- Event listeners can be added to marks to react to events on the underlying selection using the **on()** method. The **on()** method takes the event name and a callback function that is triggered every time the specified event happens.
- An anonymous function can be used as the callback for the event listener. The input to the function **d** represents the underlying data of the mark. The scope, **this**, corresponds to the DOM element.

Example: Events & Transitions Cont.

```
.on("mouseover", function(d){
  d3.select(this)
      .attr("stroke-width", "5px")
      .attr("r", r(d.fare));
})
.on("mouseout", function(){
  d3.select(this)
      .attr("stroke-width","0px")
      .attr("r",5);
});
```

Example: Loading Files

- **Loading data from external files**
- Data can be loaded from many types of external files using commands such as **d3.csv**, **d3.json**, **d3.tsv**.
- The D3 functions additionally support callback functions for dealing with the resulting data or error cases.

Example: Loading Files Cont.

What to do per row:
(Including creating aliases or specifying data type.

Callback function

Error handling

What to do with all returned rows (including sorting, filtering, or

```
d3.csv("titanic passenger list.csv",function(row,i){
  return {
    name: row.name,
    survived: (row.survived==1) ? "Yes": "No",
    sex: row.sex,
    age: +row.age,
    fare: +row.fare,
  };
}, function(error,rows){
  if(error){
    console.log(error);
  }
  rows.sort(function(a,b) { return (a.name).localeCompare(b.name); });
  allData = rows;
  makeChart(rows.slice(index,index+10));
});
```

Example: Enter/Update/Exit

- **Rebinding**
- Three things can happen when we call **data()**:
- **Update:** We want to change the elements we already have.
- **Enter:** We have new data.
- **Exit:** We have data that is no longer bound.

Example: Enter/Update/Exit Cont.

Rebinding

Good practice to have an update function.

1. **Bind** or rebind data
2. Perform **update** operations
3. Perform operations on **enter** set
4. Perform operations on **update+enter** sets
5. Perform **exit** operations

```
//BIND DATA
var scatter = d3.select(".chart").selectAll("circle")
    .data(data,key);
```

```
//UPDATE
scatter.attr("stroke-width","5px");
```

```
//ENTER
var enter = scatter.enter().append("circle")
    .attr("fill-opacity",0.85)
    .attr("r",5)
    .attr("cx",function(d,i){ return x(i); })
    .attr("cy",function(d){ return y(d.age); })
    .attr("stroke",function(d){ return s(d.survived); })
  .on("mouseover",function(d){
    d3.select(this).transition()
      .attr("stroke-width","5px")
      .attr("r",r(d.fare));
  })
  .on("mouseout",function(){
    d3.select(this).transition()
      .delay(1000)
      .attr("stroke-width","0px")
      .attr("r",5);
  });

// Add a title to the point (on mouseover)
enter.append("svg:title")
    .text(function(d){ return d.name;});
```

```
//ENTER + UPDATE
enter.merge(scatter).transition().duration(1000)
    .attr("cx",function(d,i){ return x(i); })
    .attr("cy",function(d){ return y(d.age); })
    .attr("fill",function(d){ return c(d.sex); })
    .attr("stroke",function(d){ return s(d.survived); })
    .attr("stroke-width","0px");
```

```
//EXIT
scatter.exit().transition().duration(1000)
    .attr("cx",0)
    .attr("fill-opacity",0)
    .remove();
```

Example: Enter/Update/Exit Cont.

1. Update

- Things I want to happen to all of our data, whenever the function is called. Potentially overwritten by later steps.

```
//UPDATE
scatter.attr("stroke-width","5px");
```

Example: Enter/Update/Exit Cont.

2. Enter

- Things I want to happen to all new data
- Can use **append()** to make new elements for new data.

```
//ENTER
var enter = scatter.enter().append("circle")
    .attr("fill-opacity",0.85)
    .attr("r",5)
    .attr("cx",function(d,i){ return x(i); })
    .attr("cy",function(d){ return y(d.age); })
    .attr("stroke",function(d){ return s(d.survived); })
  .on("mouseover",function(d){
    d3.select(this).transition()
      .attr("stroke-width","5px")
      .attr("r",r(d.fare));
  })
  .on("mouseout",function(){
    d3.select(this).transition()
      .delay(1000)
      .attr("stroke-width","0px")
      .attr("r",5);
  });

// Add a title to the point (on mouseover)
enter.append("svg:title")
    .text(function(d){ return d.name;});
```

Example: Enter/Update/Exit Cont.

3. Enter+Update

- Things I want to set initially. Can use transitions to have attributes fade in after creation.

```
//ENTER + UPDATE
enter.merge(scatter).transition().duration(1000)
    .attr("cx",function(d,i){ return x(i); })
    .attr("cy",function(d){ return y(d.age); })
    .attr("fill",function(d){ return c(d.sex); })
    .attr("stroke",function(d){ return s(d.survived); })
    .attr("stroke-width","0px");
```

Example: Enter/Update/Exit Cont.

4. Exit

- Things I want to happen to old data
- Can use transitions to make old data fade away
- Can use **remove()** to keep only elements that are bound to our current data.

```
//EXIT
scatter.exit().transition().duration(1000)
    .attr("cx",0)
    .attr("fill-opacity",0)
    .remove();
```

Example: Enter/Update/Exit Cont.

Key binding

- With only one argument, binding will only keep track of the amount of data we have.
- If we always have the same *amount* of data, then nothing will “exit.”
- Can use a argument to specify unique identifiers for data, to define whether data should enter or exit.
- Here, our key is the index (row number) of the data in our original csv. Passenger name is not unique, and so would not make a good key.

Example: Enter/Update/Exit Cont.

```
var index = 0;

d3.csv("titanic passenger list.csv",function(row,i){
  return {
    name: row.name,
    survived: (row.survived==1) ? "Yes": "No",
    sex: row.sex,
    age: +row.age,
    fare: +row.fare,
    key: i
  };
},function(error,rows){
  if(error){
    console.log(error);
  }
  rows.sort(function(a,b) { return (a.name).localeCompare(b.name);});
  allData = rows;
  makeChart(rows.slice(index,index+10));
});

var key = function(d){ return d.key; };
```

```
//BIND DATA
var scatter = d3.select(".chart").selectAll("circle")
    .data(data,key);
```