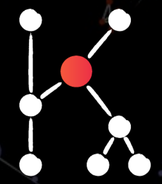


INTERACTIVE D3

KIRELL BENZI, PH.D

@KirellBenzi

www.kirellbenzi.com

Scales

Critical if we want map abstract data to a visual representation

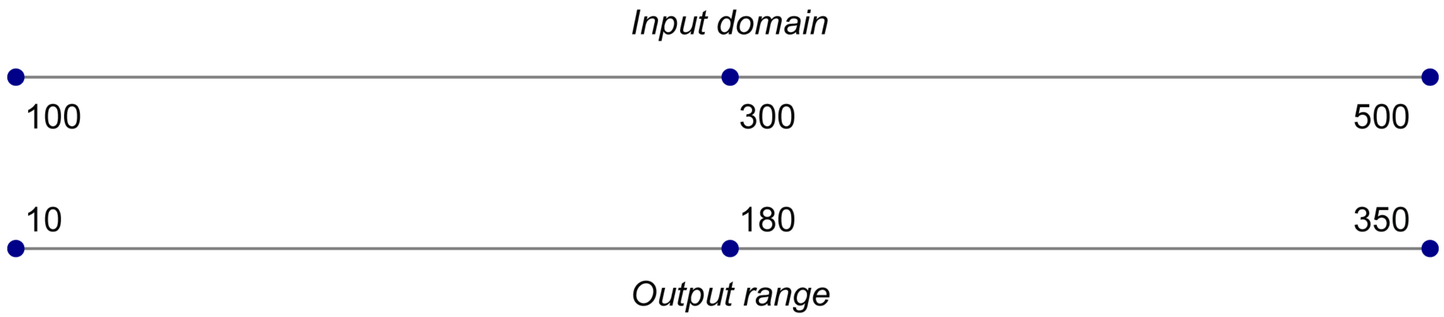
Map values from an ***input domain*** to an ***output range***

≈ Take an interval and transform it into a new interval

```
const myScale = d3.scaleLinear()
                  .domain([100, 500])
                  .range([10, 350]);

myScale(100); // 10
myScale(300); // 180
myScale(5000); // 4175

myScale.clamp(true); // cannot overshoot bounds
myScale(5000); // 350
```



More scales

D3 has a lot of different scales grouped into quantitative and ordinal scales

Under the hood, d3 uses interpolators

API Reference

- Continuous (Linear, Power, Log, Identity, Time)
- Sequential
- Quantize
- Quantile
- Threshold
- Ordinal (Band, Point, Category)

```
const colorScale = d3.scaleLinear()
                     .domain([100, 500])
                     .range(['steelblue', 'crimson']);
console.log(colorScale(100)); // "rgb(70, 130, 180)"
console.log(colorScale(233)); // "rgb(120, 93, 140)"
```

More on scales

d3.max and d3.min

When working with a real dataset you need to be able to change a scale dynamically.

Finding the max and min of your data is easy

```
const dataset = [
  [5, 20],
  [480, 90],
  [250, 50],
  [100, 33],
  [330, 95],
  [410, 12],
  [475, 44],
  [25, 67],
  [85, 21],
  [220, 88]
];
```

```
xMax = d3.max(dataset, d => d[0]); // 475
yMin = d3.min(dataset, d => d[1]); // 12
```

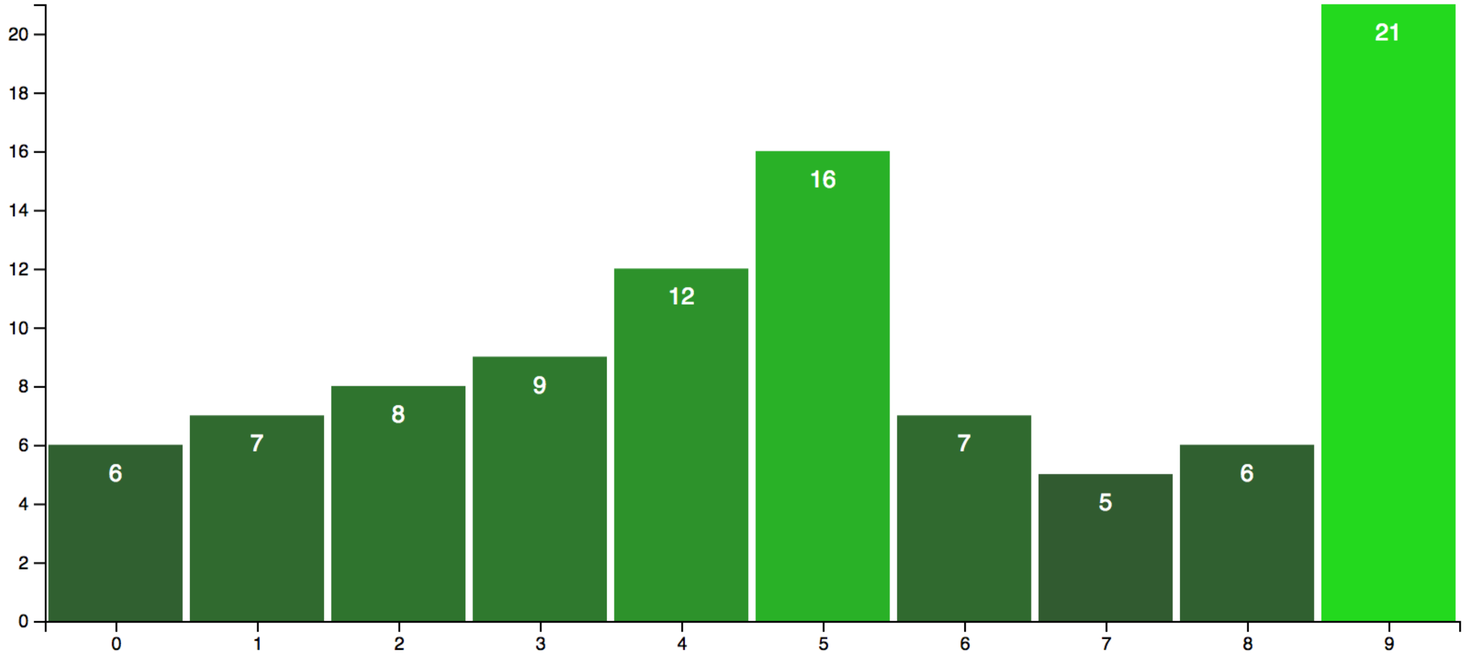
Axes

Axes are mandatory for data viz, yet extremely tedious to create.

They are made up of path, line, and text SVG elements that we group together using the <g> tag.

As such, they can be styled by assigning a CSS class to the whole group

```
.axis line,
.axis path {
  fill: none;
  stroke: #000;
  shape-rendering: crispEdges;
}
.axis text {
  font-family: sans-serif;
  font-size: 11px;
}
```



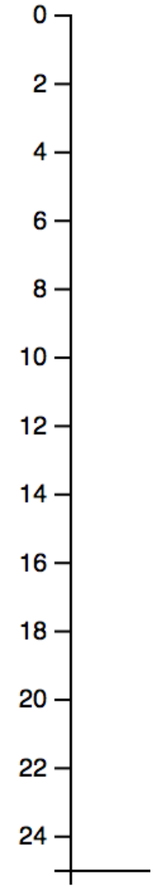
d3 Axis

To generate the elements we need an axis, a scale and an orientation to draw the tick marks and labels correctly.

```
const yScale = d3.scaleLinear()
                 .domain([0, d3.max(dataset)])
                 .range([0, height]);

const yAxis = d3.axisLeft(yScale);

svg.append("g")
   .attr('class', 'axis')
   .call(yAxis); // creates elements from the current selection
```



Reversing an axis

The top corner of the screen is always coordinate (0,0)

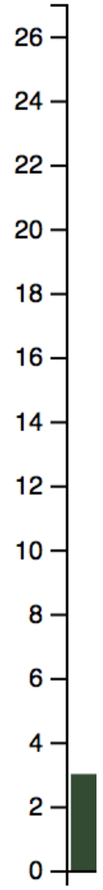
Y values are thus pushed down our screen

To invert the labels, we rely on the output range and swap min and max value

```
const yScale = d3.scaleLinear()
                 .domain([0, d3.max(dataset)])
                 .range([height, 0]);

const yAxis = d3.axisLeft(yScale);

svg.append("g")
   .attr('class', 'axis')
   .call(yAxis); // creates elements from the current selection
```

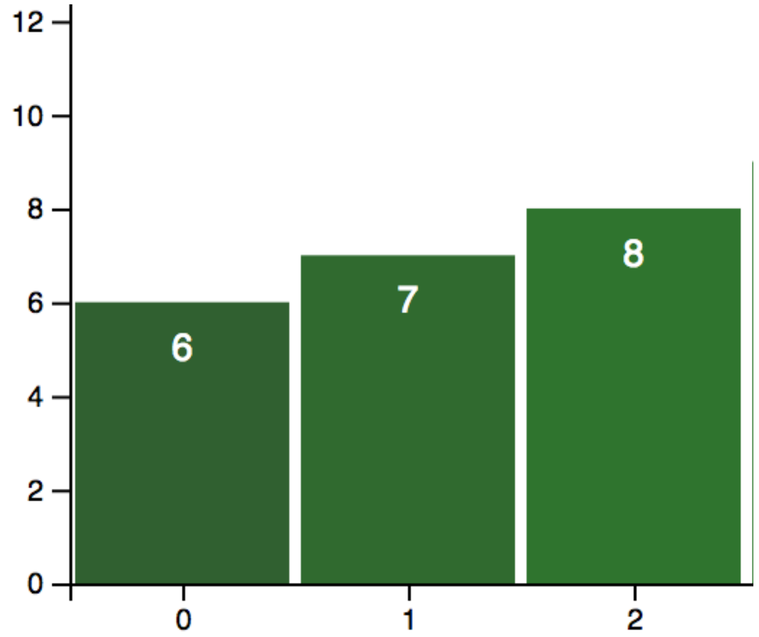


Moving things around

All axes are also drawn at the top-left corner of the screen

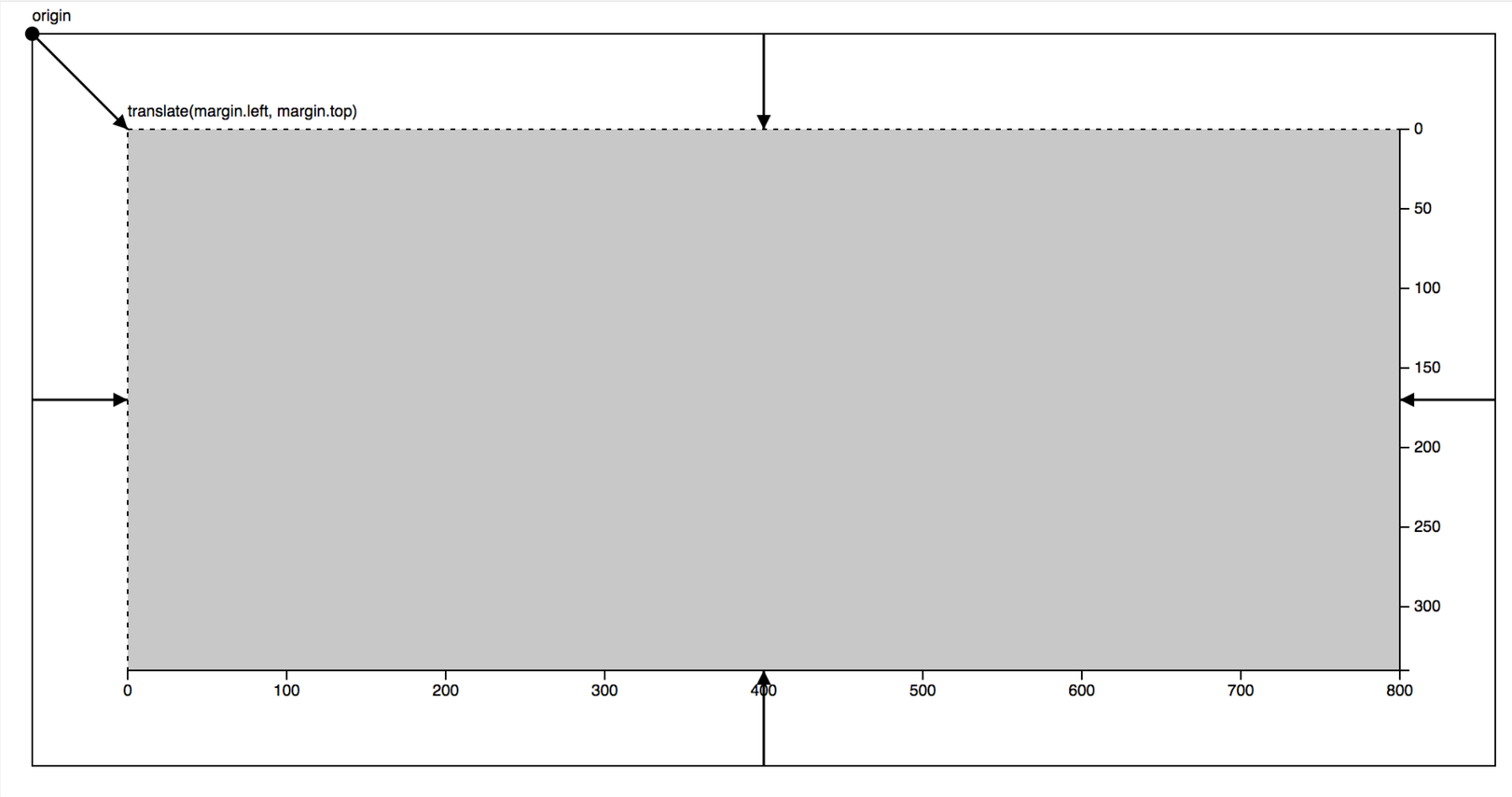
To be able to choose to put our axes where we want, we rely (again) on the SVG transform function

```
const height = 1000;
const xAxis = d3.axisBottom(xScale);
 svg.append("g")
    .attr('class', 'axis')
    .attr('transform', `translate(0, ${height})`)
    .call(xAxis);
```



Margin convention

To remove any issues with our labels, D3's creator proposes a margin convention




```
let svg = d3.select("body").append("svg")
    .attr("width", width + margin.left + margin.right)
    .attr("height", height + margin.top + margin.bottom)
  .append("g")
    .attr("transform", `translate(${margin.left}, ${margin.top})`);
```

DOM events

To interact with your data viz, we need a way to capture our user's intentions

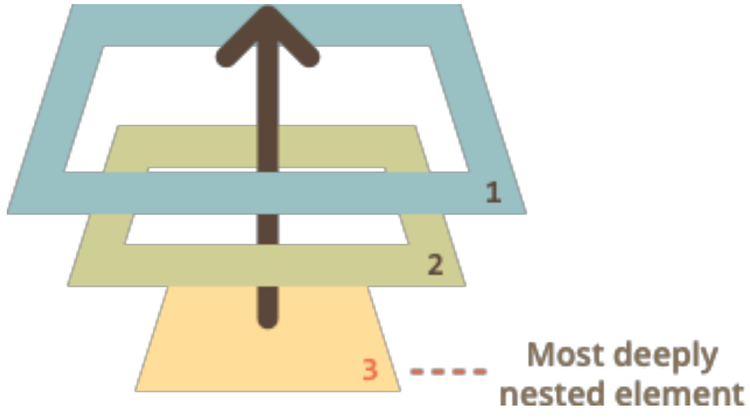
Each DOM element can be used to register functions as *handlers* for specific events

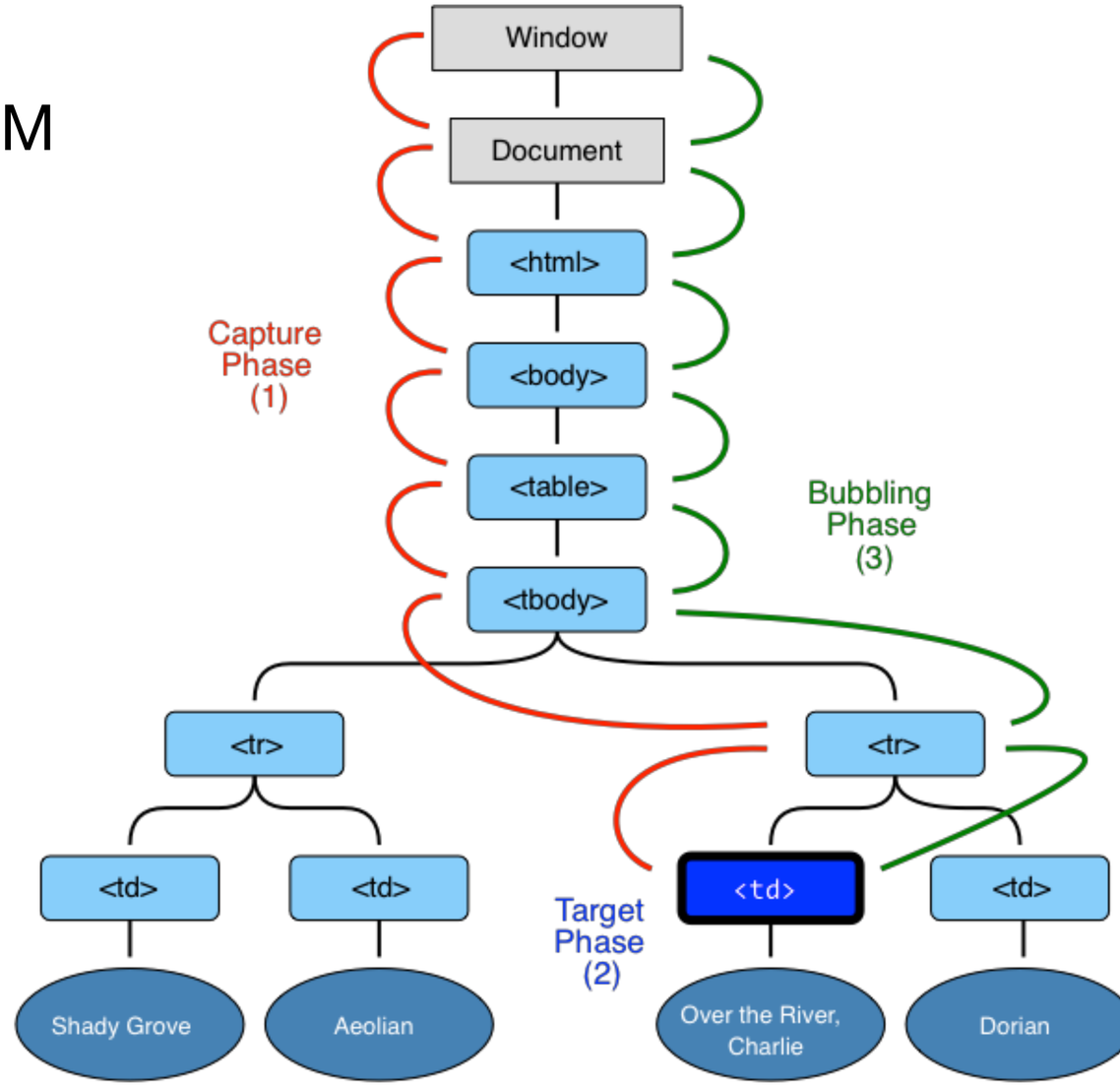
addEventListener allows to react on an event, ***removeEventListener*** removes it from the DOM element's queue

```
let button = document.querySelector("button");
function once() {
  console.log("Yo!");
  button.removeEventListener("click", once);
}
button.addEventListener("click", once);
```

Event propagation

If the same event is attached to several DOM elements, events bubble up from the most specific to the least specific one





Event target

event.target represents the most deeply nested element that triggered the event

event.target doesn’t change through the bubbling process

event.currentTarget (== this) is the current element, the one that has a currently running handler on it

Stop event

A paragraph with a button.

Console

"Button handler."

"Paragraph handler"

We can stop the propagation of an event with event.stopPropagation()

Beware, there are only a few valid reason to do this

```
let para = document.querySelector("p");
let button = document.querySelector("button");
para.addEventListener("mousedown", () => console.log("Paragraph handler"));
button.addEventListener("mousedown", (event) => {
  console.log("Button handler.");
  if (event.which == 3) // Check if right button
    event.stopPropagation();
});
```

Common events

Category	Type	Attribute	Description	Bubbles	Cancelable
Mouse	click	onclick	Fires when the pointing device button is clicked over an element. A click is defined as a mousedown and mouseup over the same screen location. The sequence of these events is: • mousedown • mouseup • click	Yes	Yes
	dblclick	ondblclick	Fires when the pointing device button is double-clicked over an element	Yes	Yes
	mousedown	onmousedown	Fires when the pointing device button is pressed over an element	Yes	Yes
	mouseup	onmouseup	Fires when the pointing device button is released over an element	Yes	Yes
	mouseover	onmouseover	Fires when the pointing device is moved onto an element	Yes	Yes
	mousemove	onmousemove	Fires when the pointing device is moved while it is over an element	Yes	No
	mouseout	onmouseout	Fires when the pointing device is moved away from an element	Yes	Yes
Keyboard	keydown	onkeydown	Fires before keypress, when a key on the keyboard is pressed.	Yes	Yes
	keypress	onkeypress	Fires after keydown, when a key on the keyboard is pressed.	Yes	Yes
	keyup	onkeyup	Fires when a key on the keyboard is released	Yes	Yes

More on events: http://eloquentjavascript.net/14_event.html

D3 events

d3 selection offers the same declarative functional interface to add (or remove) event listeners to a selection

```
d3.selectAll("div")
  .on("mouseover", function(d, i) {
      console.log(d, i); // datum if any, current index
      d3.select(this)
        .style("background-color", "orange");
    };
```

Use regular function and not arrow func to have the correct *this*

Live example: https://codepen.io/kikohs/pen/dzVvvj?editors=1111

Animations

Tweening (Inbetweening) is the process of generating intermediate frames between two images to give the appearance that the first image evolves smoothly into the second image [wikipedia]

Animations are thus need to **smoothly interpolate values through time** to create the illusion

The **easing function** refers to a mechanism for defining the physics of the transition between two animation states, i.e., the linearity of a tween

Animations on the browser

Browsers do not update their display while a JavaScript program is running, nor do they allow any interaction with the page.

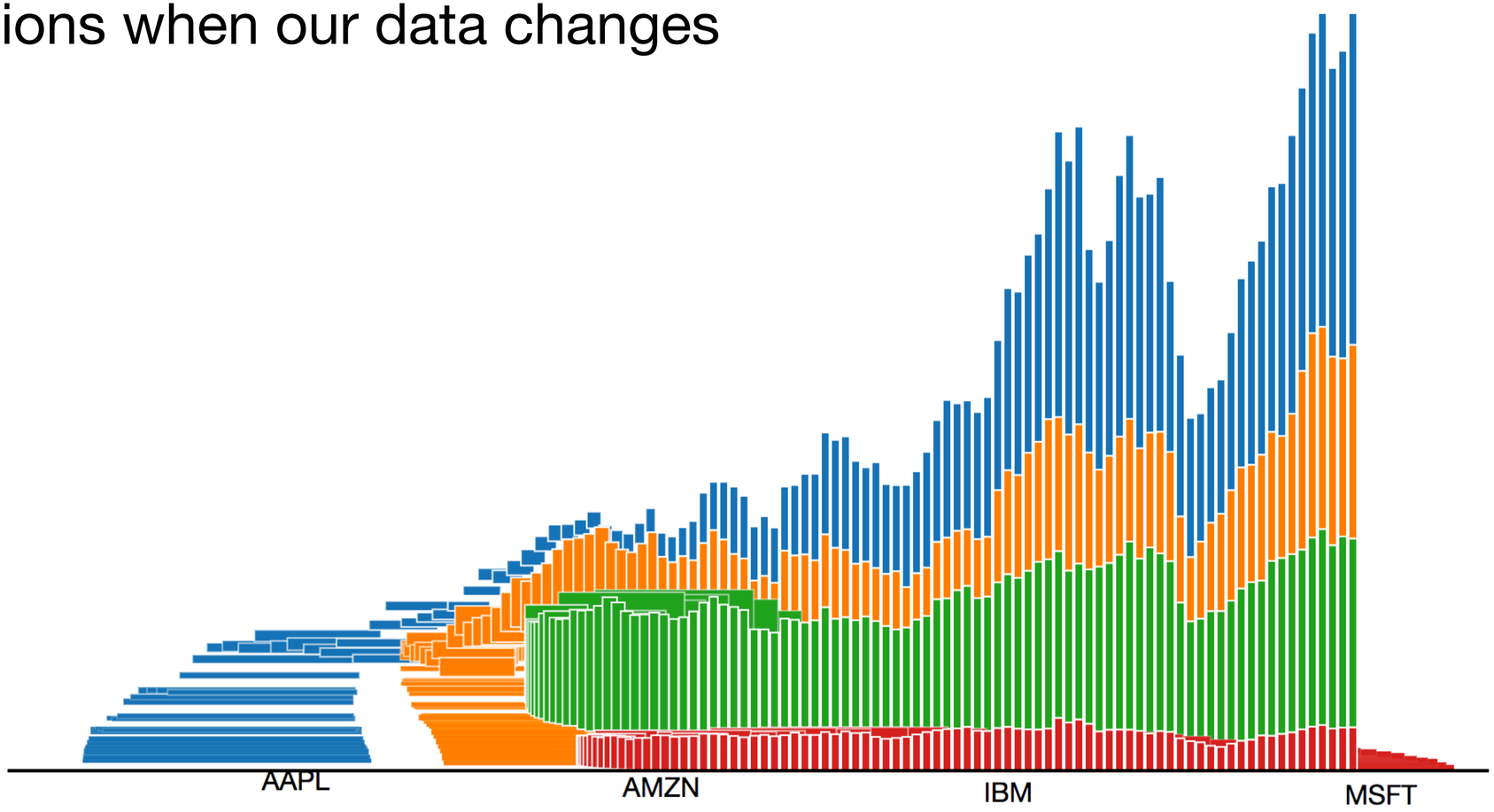
requestAnimationFrame schedules our animation function to run whenever the browser is ready to repaint the screen.

Once the values have changed, we call our animation function again

```
function animate(time) {
 // interpolate values
 requestAnimationFrame(animate);
}
requestAnimationFrame(animate);
```

D3 to the rescue

Fortunatelly, D3 abstracts a lot of the complexity for creating smooth transitions when our data changes

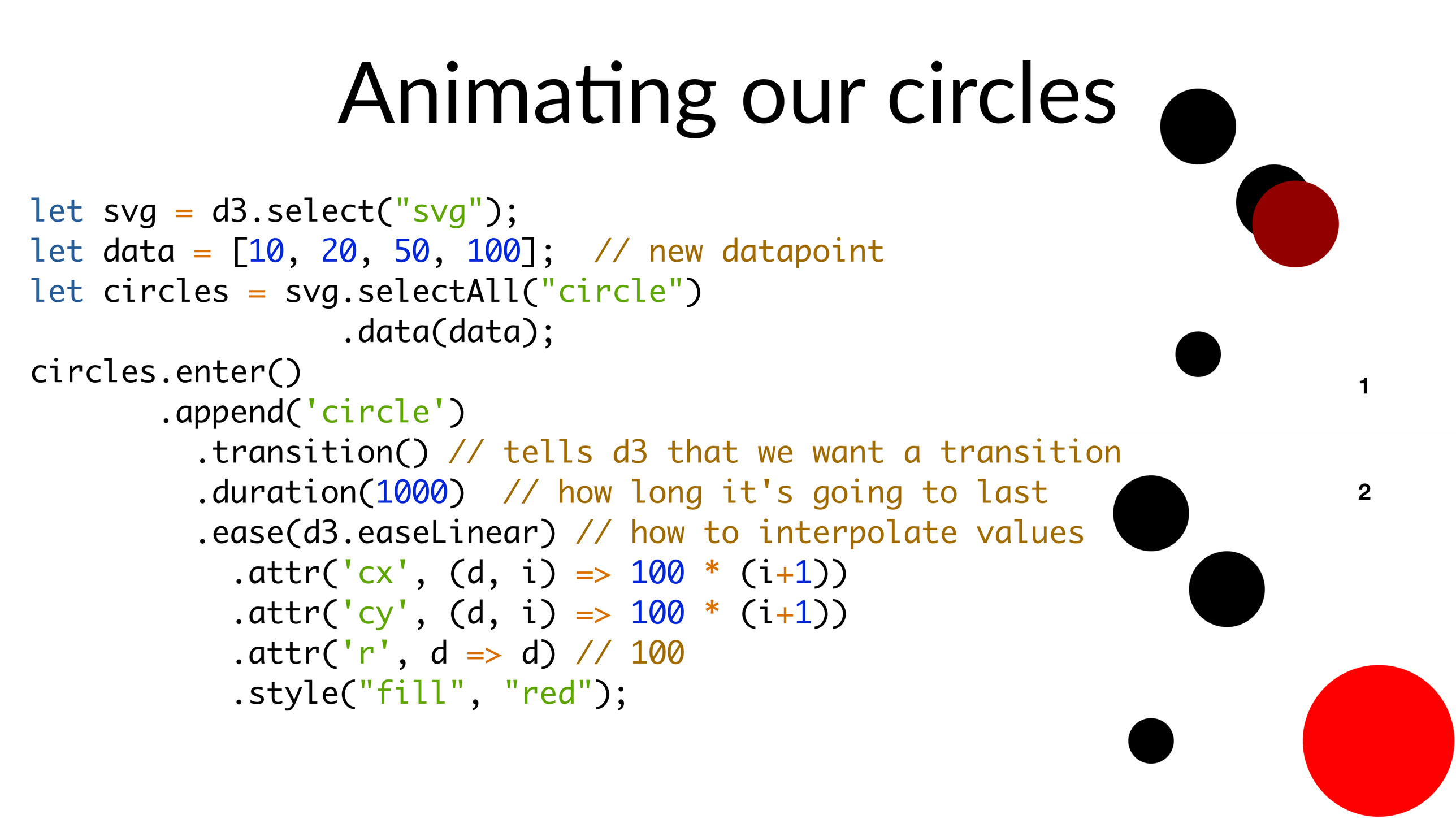


Animating our circles

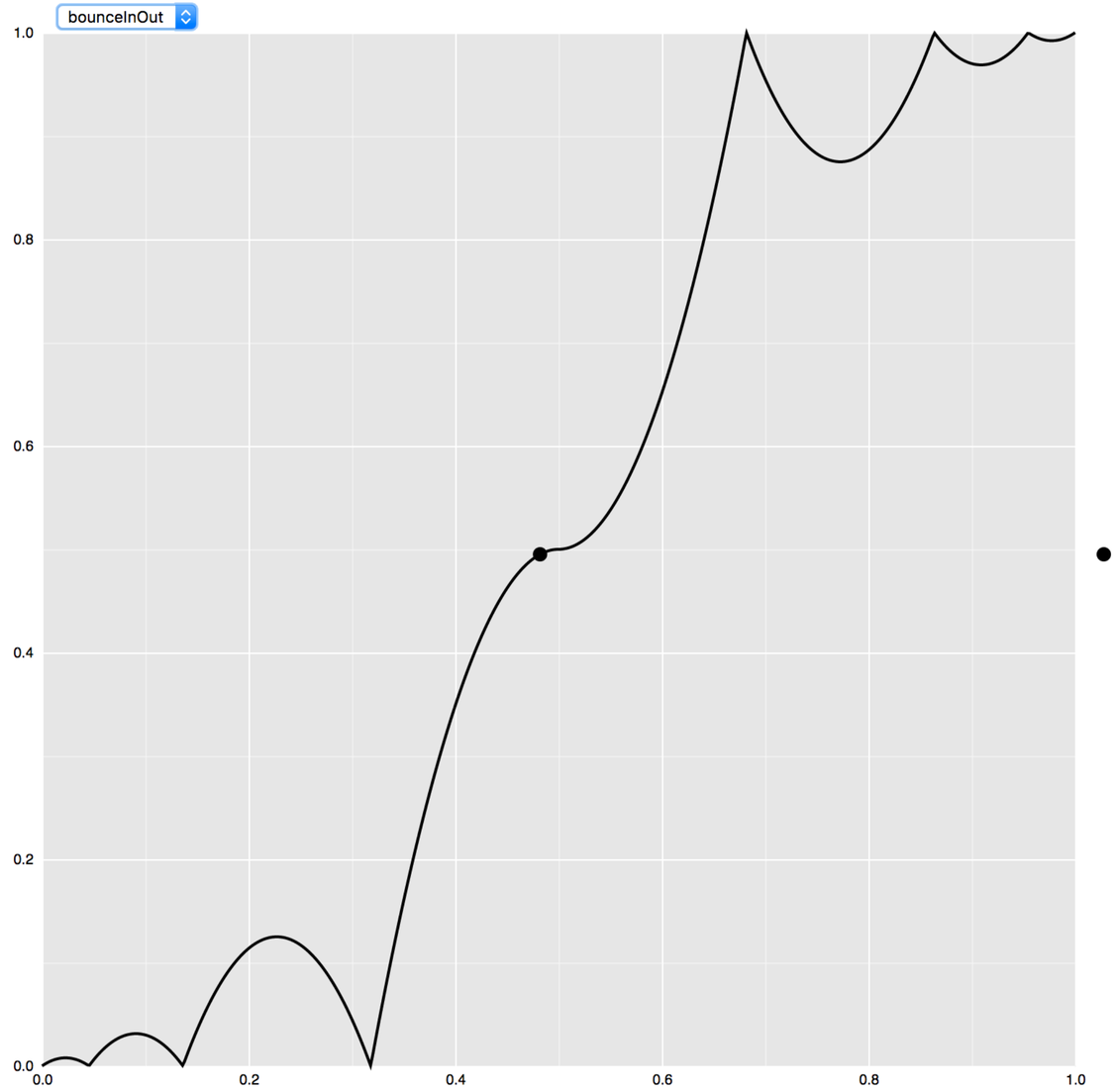
```
let svg = d3.select("svg");
let data = [10, 20, 50, 100];  // new datapoint
let circles = svg.selectAll("circle")
                 .data(data);
circles.enter()
       .append('circle')
         .transition() // tells d3 that we want a transition
         .duration(1000)  // how long it's going to last
         .ease(d3.easeLinear) // how to interpolate values
           .attr('cx', (d, i) => 100 * (i+1))
           .attr('cy', (d, i) => 100 * (i+1))
           .attr('r', d => d) // 100
           .style("fill", "red");
```

1

2



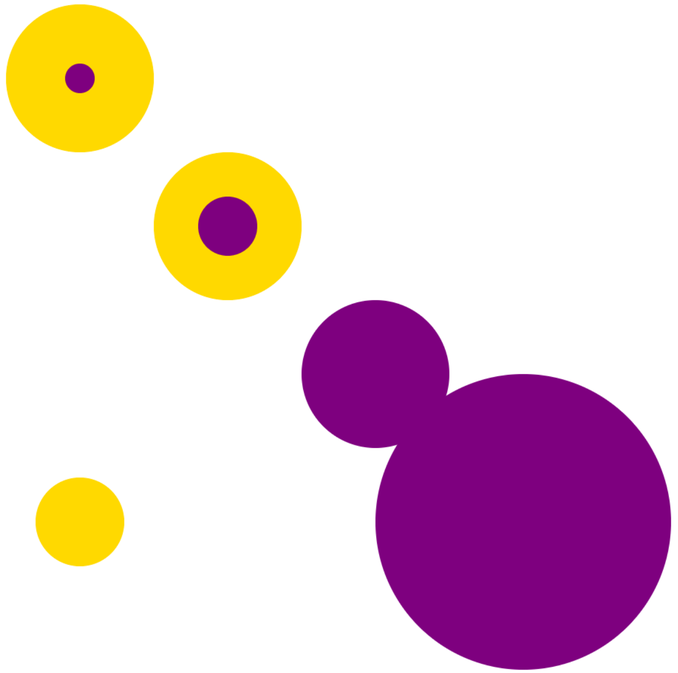
D3 easing functions



Adding a key to the data-join

```
let svg = d3.select("svg");
let data = [10, 20, 50, 100];
let circles = svg.selectAll("circle")
                 .data(data, d => d); // map circle to data value
circles.enter()
       .append('circle')
         .transition()
         .duration(10000)
           .attr('cx', (d, i) => 100 * (i+1))
           .attr('cy', (d, i) => 100 * (i+1))
           .attr('r', d => d) // 100
           .style("fill", "purple");

circles.exit()
       .transition()
       .delay((d, i) => i * 500)
       .duration(300)
       .style("fill", "gold");
```



General update pattern 3

Crossfilter

Fast Multidimensional Filtering for Coordinated Views

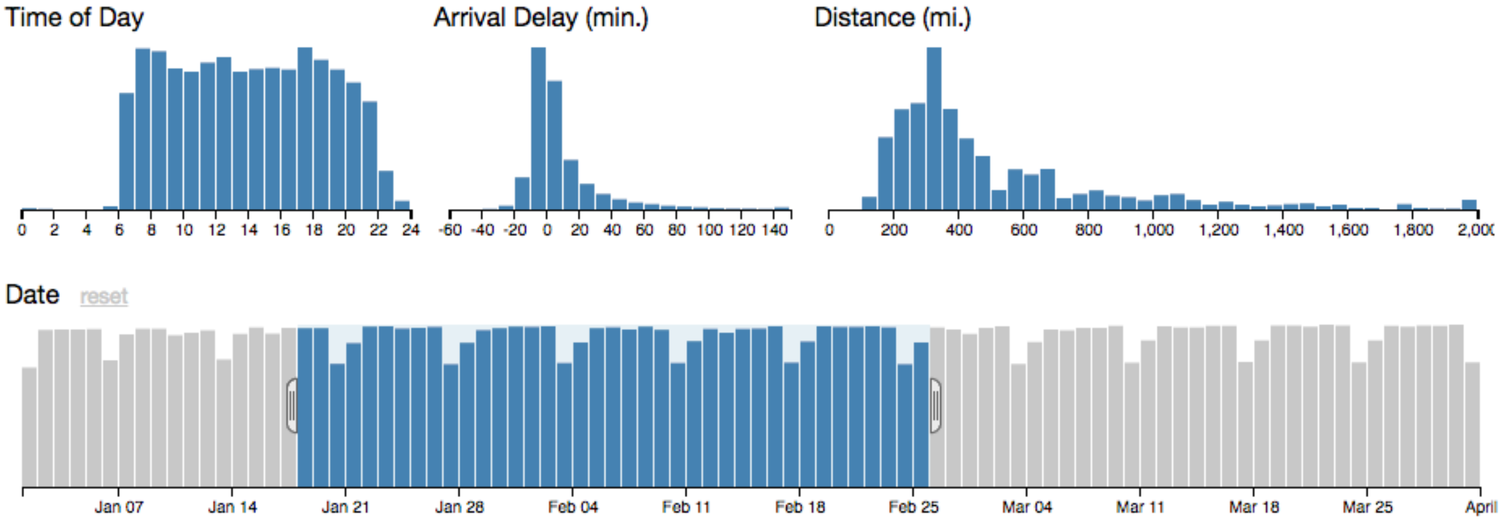
Crossfilter is a JavaScript library for exploring large multivariate datasets in the browser.

Supports extremely fast (<30ms) interaction with coordinated views, even with datasets containing a million or more records

Based on incremental filtering and reducing because interactions mostly involves a single dimension => significantly faster than starting from scratch.

https://github.com/crossfilter/crossfilter/wiki/API-Reference

Live example



February 25, 2001

99,793 of 231,083 flights selected.

11:58 PM	LAX	OAK	337 mi.	+204 min.
11:56 PM	LAS	SLC	368 mi.	+94 min.
11:52 PM	SJC	SAN	417 mi.	+132 min.
11:47 PM	MDW	STL	251 mi.	+158 min.
11:47 PM	SMF	LAS	397 mi.	+116 min.
11:43 PM	PHX	ABQ	328 mi.	+175 min.
11:40 PM	LAX	LAS	236 mi.	+310 min.
11:40 PM	LAS	ABQ	487 mi.	+125 min.
11:40 PM	LAX	LAS	236 mi.	+124 min.

Crossfilter in practice

```
const payments = crossfilter([
  {date: "2011-11-14T16:17:54Z", quantity: 2, total: 190, tip: 100, type: "tab"},
  {date: "2011-11-14T16:20:19Z", quantity: 2, total: 190, tip: 100, type: "tab"},
  {date: "2011-11-14T16:28:54Z", quantity: 1, total: 300, tip: 200, type: "visa"},
  {date: "2011-11-14T16:30:43Z", quantity: 2, total: 90, tip: 0, type: "tab"},
  {date: "2011-11-14T16:48:46Z", quantity: 2, total: 90, tip: 0, type: "tab"},
  {date: "2011-11-14T16:53:41Z", quantity: 2, total: 90, tip: 0, type: "tab"},
  {date: "2011-11-14T16:54:06Z", quantity: 1, total: 100, tip: 0, type: "cash"},
  {date: "2011-11-14T16:58:03Z", quantity: 2, total: 90, tip: 0, type: "tab"},
  {date: "2011-11-14T17:07:21Z", quantity: 2, total: 90, tip: 0, type: "tab"},
  {date: "2011-11-14T17:22:59Z", quantity: 2, total: 90, tip: 0, type: "tab"},
  {date: "2011-11-14T17:25:45Z", quantity: 2, total: 200, tip: 0, type: "cash"},
  {date: "2011-11-14T17:29:52Z", quantity: 1, total: 200, tip: 100, type: "visa"}
]);
```

```
let paymentsByTotal = payments.dimension(d => d.total);
paymentsByTotal.filter([100, 200]); // selects payments whose total is between 100 and
200
paymentsByTotal.filter(120); // selects payments whose total equals 120
paymentsByTotal.filter(d => d % 2); // selects payments whose total is odd
paymentsByTotal.filter(null); // selects all payments
```

https://github.com/crossfilter/crossfilter/wiki/API-Reference

Crossfilter groups

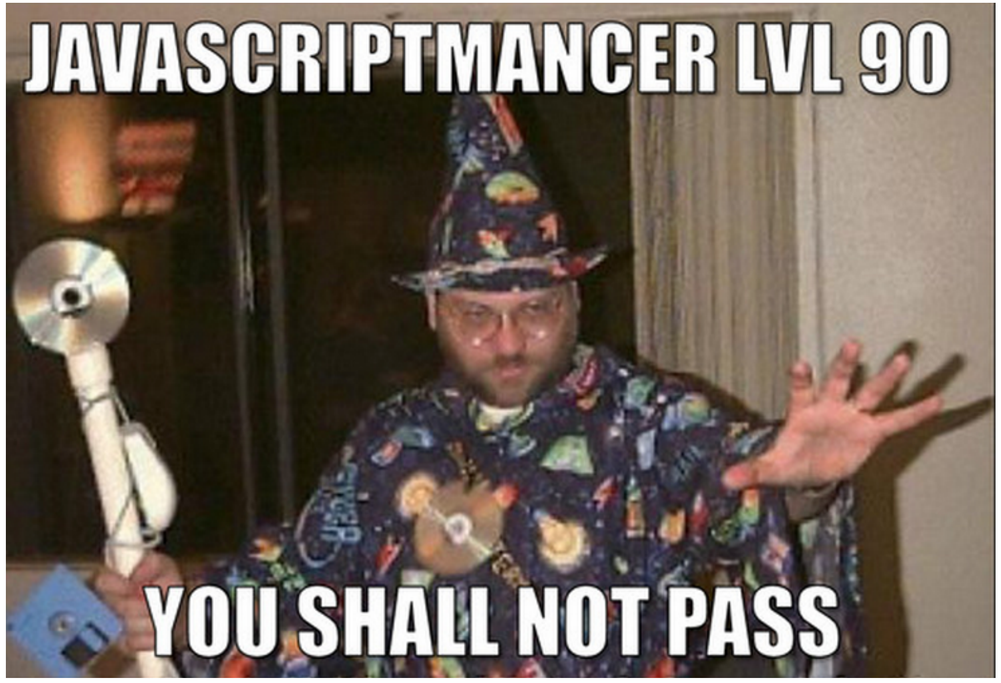
Constructs a new grouping for the given dimension, according to the specified groupValue function (default to identity), which takes a dimension value as input and returns the corresponding rounded value.

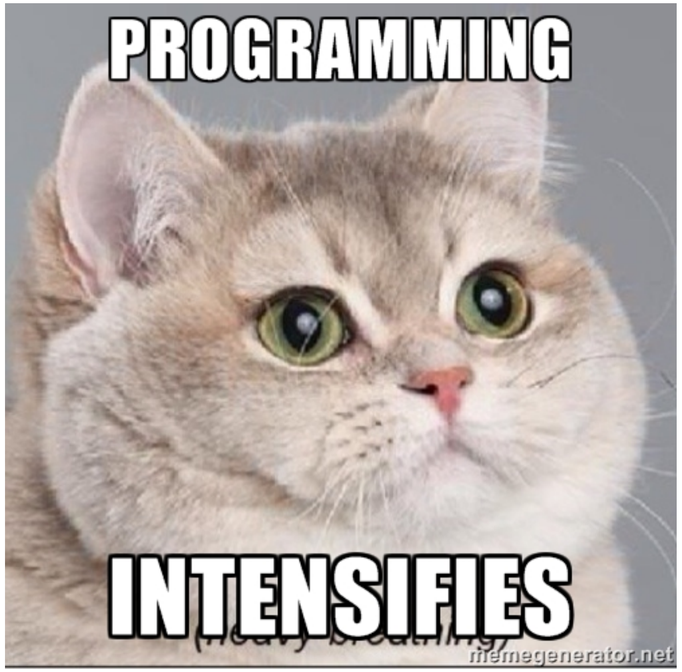
```
const paymentsByType = payments.dimension(d => d.type);
const paymentVolumeByType = paymentsByType.group().reduceSum(d => d.total);
const topTypes = paymentVolumeByType.top(1);
topTypes[0].key; // the top payment type (e.g., "tab")
topTypes[0].value; // the payment volume for that type (e.g., 920)
```

https://github.com/crossfilter/crossfilter/wiki/API-Reference

The last Javascript slide

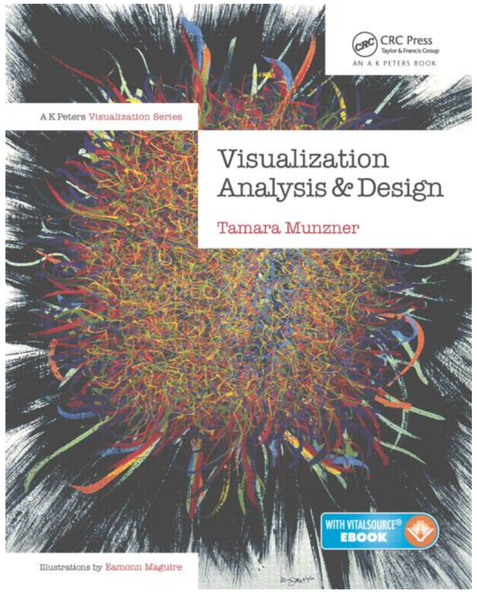
“Insert inspiring quote”





Homework

Read Visualization Analysis and Design
Chapter 6.8 & 11, 12, 13



Read Interactive Data Visualization for the Web
chapter 7 to 11 included