Visual-Analytics

Introduction to D3.js

Personal info

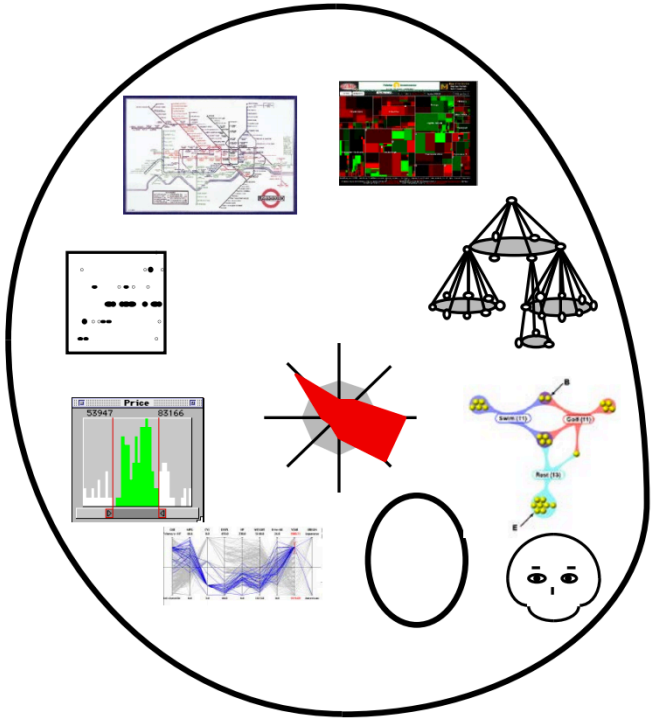
Name: Marco Angelini

E-mail: angelini@diag.uniroma1.it

Office hour:

- no fixed time (does not mean no office hour)
- Send an e-mail for appointment
- room B113, 1st floor (not for this semester)

What have you NOT seen so far (almost…)



Techniques

change blindness

derived structure

perceptual issues

popping out

visibility

derived values

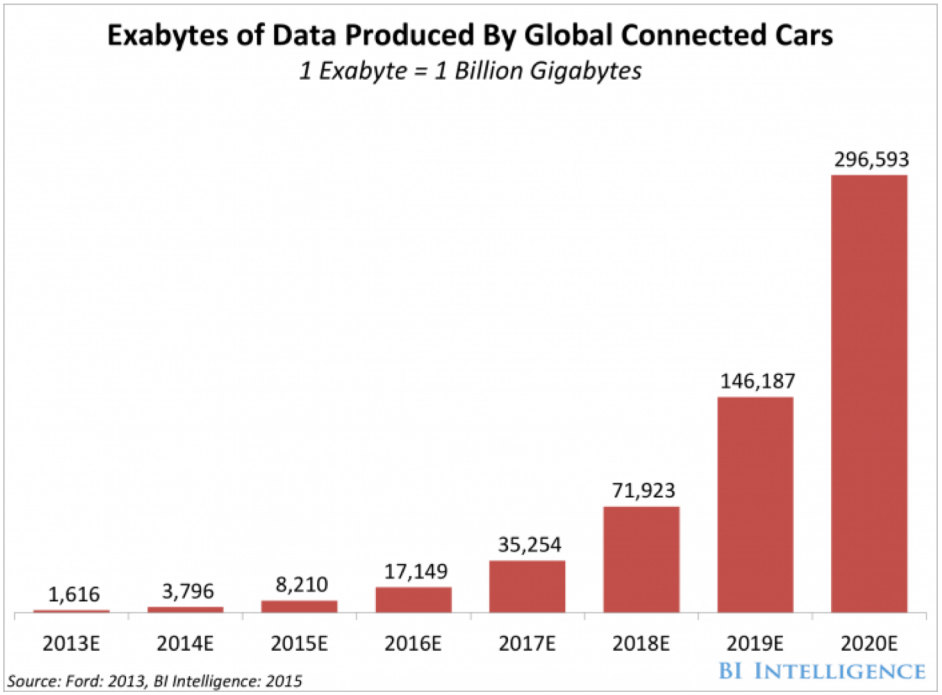
brushing

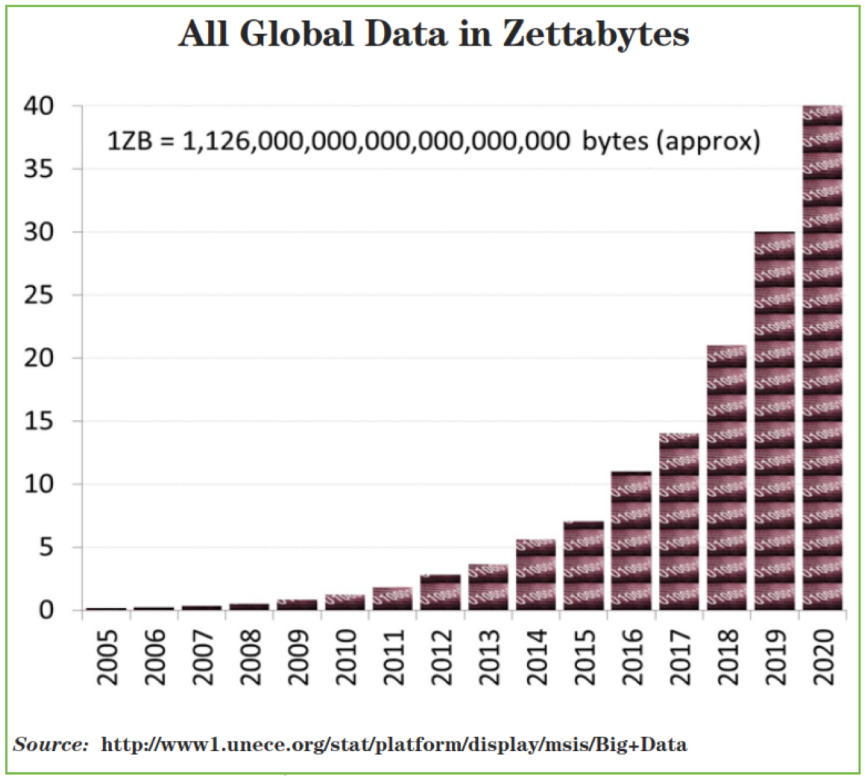
Concepts

Representation

Perception

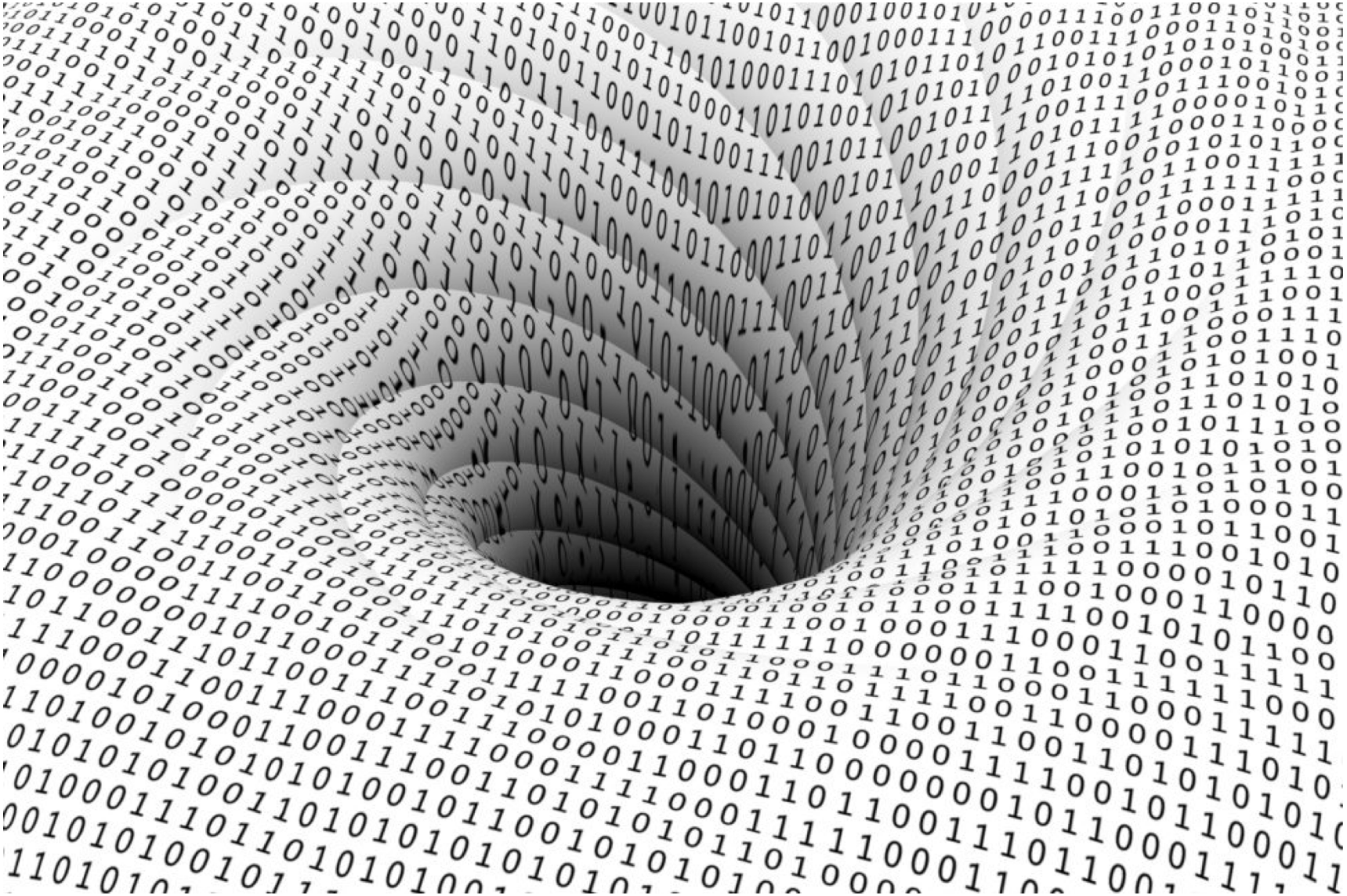
Data





- Apart from the very well-known sectors:
  - Helthcare data
  - Sports & E-sports analytics data
  - Machine Learning & A.I. data
  - Data science
  - …

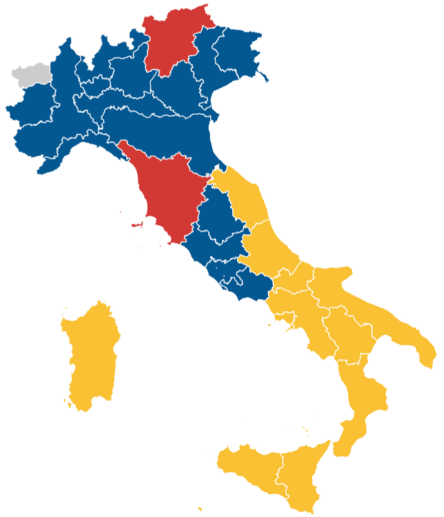
Data



Looking at data…

DATAELEZIO	NUMLISTA	CIRCOSCRIZI	PROVINCIA	COMUNE	LISTA	VOTILISTA
24/02/13	1	PIEMONTE 1	TORINO	AGLIE'	SINISTRA EC	38
24/02/13	2	PIEMONTE 1	TORINO	AGLIE'	CENTRO DEN	0
24/02/13	3	PIEMONTE 1	TORINO	AGLIE'	PARTITO DEI	368
24/02/13	4	PIEMONTE 1	TORINO	AGLIE'	RIVOLUZION	23
24/02/13	5	PIEMONTE 1	TORINO	AGLIE'	MIR - MODE	1
24/02/13	6	PIEMONTE 1	TORINO	AGLIE'	FRATELLI D'I	34
24/02/13	7	PIEMONTE 1	TORINO	AGLIE'	PARTITO PEI	17
24/02/13	8	PIEMONTE 1	TORINO	AGLIE'	IL POPOLO D	350
24/02/13	9	PIEMONTE 1	TORINO	AGLIE'	LEGA NORD	79
24/02/13	10	PIEMONTE 1	TORINO	AGLIE'	LA DESTRA	4
24/02/13	11	PIEMONTE 1	TORINO	AGLIE'	FARE PER FE	16
24/02/13	12	PIEMONTE 1	TORINO	AGLIE'	CASAPOUND	5
24/02/13	13	PIEMONTE 1	TORINO	AGLIE'	FIAMMA TRI	0
24/02/13	14	PIEMONTE 1	TORINO	AGLIE'	MOVIMENTO	377
24/02/13	15	PIEMONTE 1	TORINO	AGLIE'	FUTURO E LI	8
24/02/13	16	PIEMONTE 1	TORINO	AGLIE'	SCELTA CIVIC	139
24/02/13	17	PIEMONTE 1	TORINO	AGLIE'	UNIONE DI C	25
24/02/13	18	PIEMONTE 1	TORINO	AGLIE'	FORZA NUO'	4
24/02/13	1	PIEMONTE 1	TORINO	AIRASCA	SINISTRA EC	80
24/02/13	2	PIEMONTE 1	TORINO	AIRASCA	CENTRO DEN	8
24/02/13	3	PIEMONTE 1	TORINO	AIRASCA	PARTITO DEI	519
24/02/13	4	PIEMONTE 1	TORINO	AIRASCA	RIVOLUZION	53
24/02/13	5	PIEMONTE 1	TORINO	AIRASCA	MIR - MODE	2
24/02/13	6	PIEMONTE 1	TORINO	AIRASCA	FRATELLI D'I	40
24/02/13	7	PIEMONTE 1	TORINO	AIRASCA	PARTITO PEI	14
24/02/13	8	PIEMONTE 1	TORINO	AIRASCA	IL POPOLO D	389
24/02/13	9	PIEMONTE 1	TORINO	AIRASCA	LEGA NORD	40
24/02/13	10	PIEMONTE 1	TORINO	AIRASCA	LA DESTRA	4
24/02/13	11	PIEMONTE 1	TORINO	AIRASCA	FARE PER FE	7
24/02/13	12	PIEMONTE 1	TORINO	AIRASCA	CASAPOUND	2
24/02/13	13	PIEMONTE 1	TORINO	AIRASCA	FIAMMA TRI	5
24/02/13	14	PIEMONTE 1	TORINO	AIRASCA	MOVIMENTO	821
24/02/13	15	PIEMONTE 1	TORINO	AIRASCA	FUTURO E LI	9
24/02/13	16	PIEMONTE 1	TORINO	AIRASCA	SCELTA CIVIC	195
24/02/13	17	PIEMONTE 1	TORINO	AIRASCA	UNIONE DI C	57
24/02/13	18	PIEMONTE 1	TORINO	AIRASCA	FORZA NUO'	9
24/02/13	1	PIEMONTE 1	TORINO	ALA DI STUR	SINISTRA EC	10

http://dait.interno.gov.it/documenti/camera_2013_liste_italia.csv



https://elezioni.repubblica.it/2018/cameradeideputati

What we'll see:

DATA

Data manipulation & transformation

Data analysis

VISUALIZATION

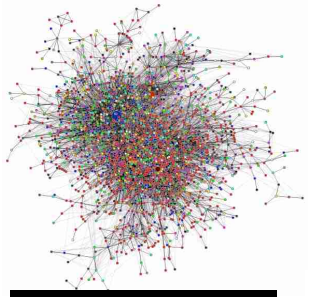
Data representation

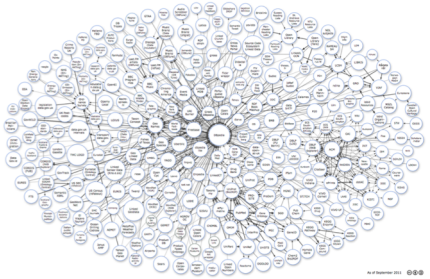
Presentation & Perception

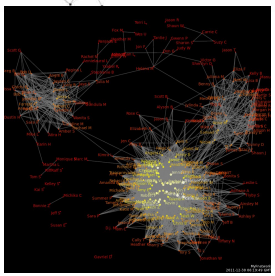
Interaction

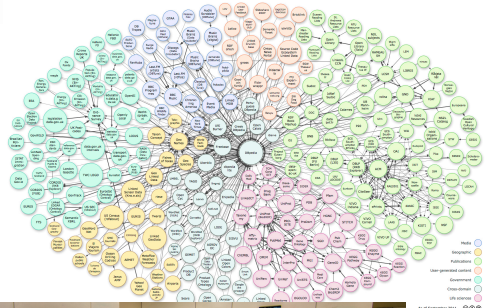
Interpolated Values

	Temp														
Time	68.0	68.5	69.0	69.5	70.0	70.5	71.0	71.5	72.0	72.5	73.0	73.5	74.0	74.5	75.0
0.025	2504.08	2638.15	2707.32	2750.09	2784.91	2851.19	2911.62	2940.67	2961.40	2983.17	3000.06	3006.32	3041.01	3125.78	3026.85
0.05	2507.26	2635.76	2704.79	2746.66	2779.96	2846.35	2907.00	2934.96	2955.07	2976.69	2993.64	2999.35	3034.49	3126.43	3036.68
0.075	2510.83	2633.45	2702.58	2743.62	2775.40	2841.84	2902.75	2929.64	2949.08	2970.51	2987.50	2992.60	3027.38	3126.97	3046.32
0.1	2513.93	2631.34	2700.70	2740.99	2771.27	2837.66	2898.88	2924.66	2943.43	2964.66	2981.67	2986.08	3021.49	3127.39	3055.77
0.125	2515.14	2629.60	2699.17	2738.77	2767.61	2833.83	2895.40	2920.07	2938.14	2959.14	2976.16	2979.83	3015.06	3127.71	3065.02
0.15	2514.31	2628.58	2698.02	2736.99	2764.49	2830.38	2892.31	2915.87	2933.23	2953.97	2970.99	2973.86	3008.70	3127.95	3074.06
0.175	2511.84	2628.88	2697.25	2735.68	2762.00	2827.31	2889.53	2912.08	2928.72	2948.17	2966.17	2968.21	3002.47	3128.11	3082.93
0.2	2508.10	2629.91	2696.87	2734.79	2760.22	2824.68	2887.26	2908.72	2924.62	2944.75	2961.71	2962.89	2996.39	3128.21	3091.57
0.225	2503.37	2631.32	2696.88	2734.37	2759.24	2822.57	2885.29	2905.80	2920.96	2940.73	2957.65	2957.93	2990.50	3128.25	3099.99
0.25	2497.84	2632.93	2697.28	2734.42	2758.10	2821.05	2883.68	2903.34	2917.76	2937.13	2953.97	2953.36	2984.86	3128.24	3108.15
0.275	2491.66	2634.64	2698.05	2734.91	2758.76	2820.23	2882.43	2901.33	2915.02	2933.97	2950.71	2949.20	2979.52	3128.18	3116.14
0.3	2484.92	2636.35	2699.18	2735.85	2761.12	2820.16	2881.55	2899.79	2912.78	2931.26	2947.88	2945.48	2974.53	3128.07	3123.83
0.325	2477.71	2638.00	2700.64	2737.22	2763.09	2820.81	2881.06	2898.72	2911.04	2929.03	2945.47	2942.21	2969.96	3127.90	3131.26
0.35	2470.07	2639.54	2702.41	2738.91	2765.59	2822.11	2880.97	2898.13	2909.82	2927.29	2943.52	2939.43	2965.89	3127.66	3138.38
0.375	2462.06	2640.93	2704.45	2741.19	2768.54	2823.98	2881.29	2898.00	2909.13	2926.05	2942.01	2937.16	2962.39	3127.30	3145.19
0.4	2453.70	2642.15	2706.75	2743.75	2771.89	2826.33	2882.03	2898.34	2908.97	2925.33	2940.96	2935.42	2959.55	3126.79	3151.66
0.425	2445.03	2643.15	2709.26	2746.67	2775.62	2829.13	2883.20	2899.16	2909.34	2925.14	2940.37	2934.25	2957.45	3126.07	3157.75
0.45	2436.07	2643.94	2711.97	2749.92	2779.68	2832.32	2884.78	2900.44	2910.23	2925.48	2940.24	2933.67	2956.16	3125.09	3163.42
0.475	2426.82	2644.48	2714.84	2753.48	2784.06	2835.88	2886.78	2902.19	2911.63	2926.34	2940.57	2933.71	2955.74	3123.85	3168.63
0.5	2417.31	2644.77	2717.84	2757.32	2788.73	2839.78	2889.19	2904.40	2913.52	2927.71	2941.36	2934.34	2956.22	3122.46	3173.31
0.525	2407.54	2644.80	2720.95	2761.44	2793.67	2844.01	2891.99	2907.04	2915.89	2929.57	2942.61	2935.55	2957.60	3121.27	3177.39
0.55	2397.51	2644.56	2724.14	2765.79	2798.87	2848.55	2895.19	2910.11	2918.72	2931.90	2944.30	2937.30	2959.85	3120.68	3180.74
0.575	2387.24	2644.05	2727.39	2770.37	2804.31	2853.38	2898.77	2913.60	2921.99	2934.68	2948.43	2939.57	2962.89	3121.69	3183.27
0.6	2376.71	2643.25	2730.67	2775.14	2809.97	2858.49	2902.71	2917.48	2925.67	2937.89	2948.99	2942.35	2966.66	3123.41	3184.53



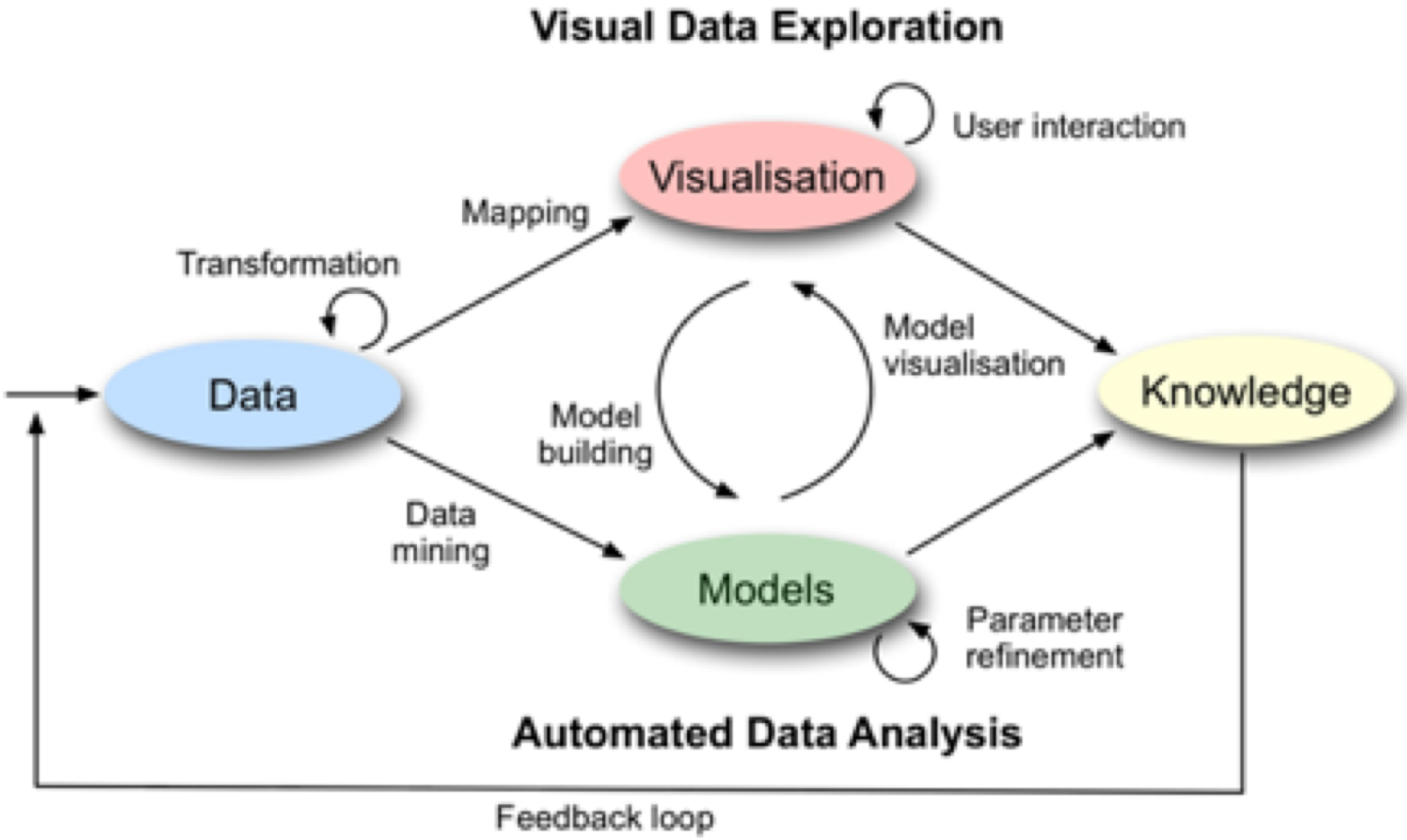


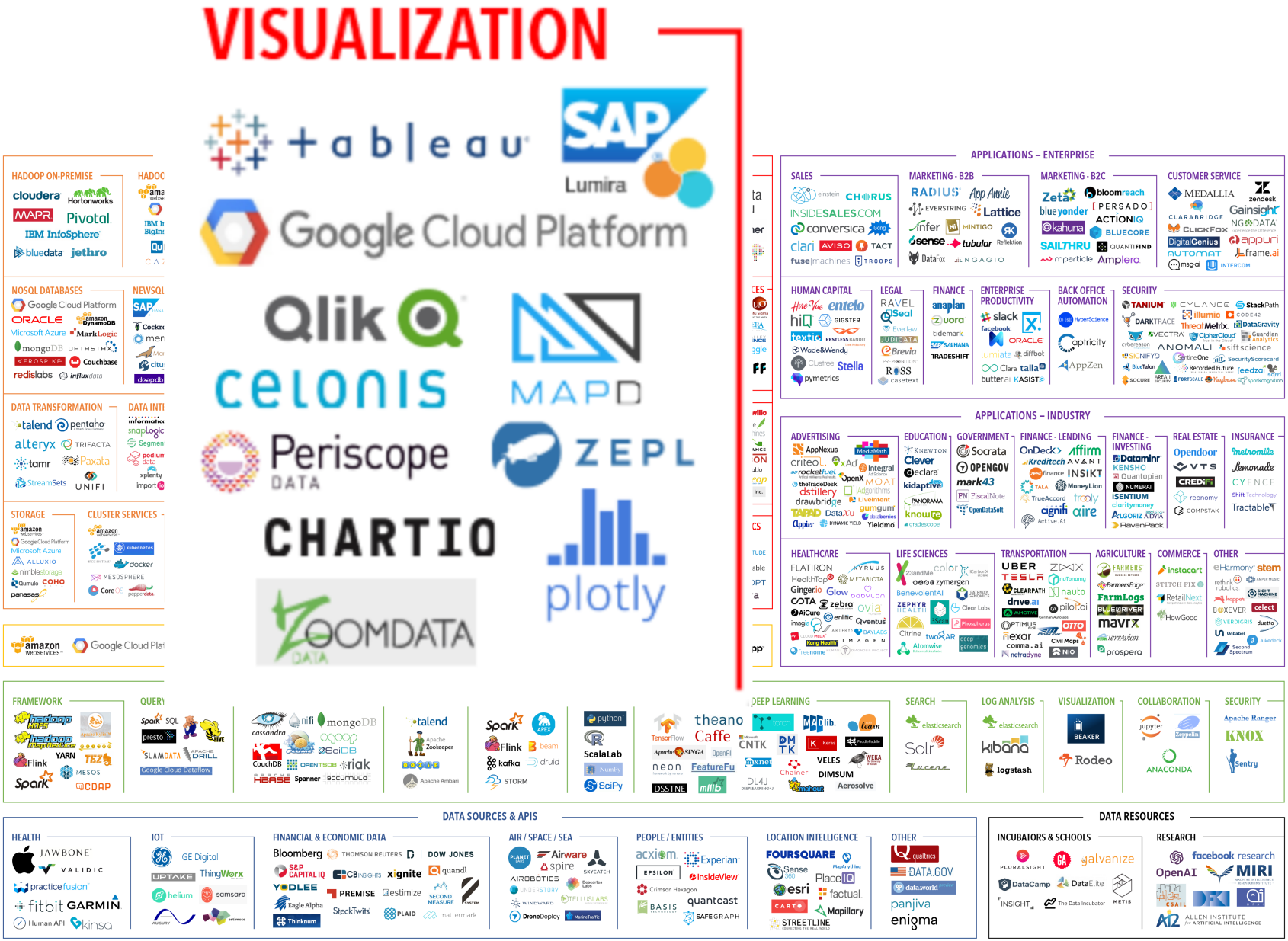




A.A. 2020/21

VISUAL ANALYTICS





V2 – Last updated 5/3/2017

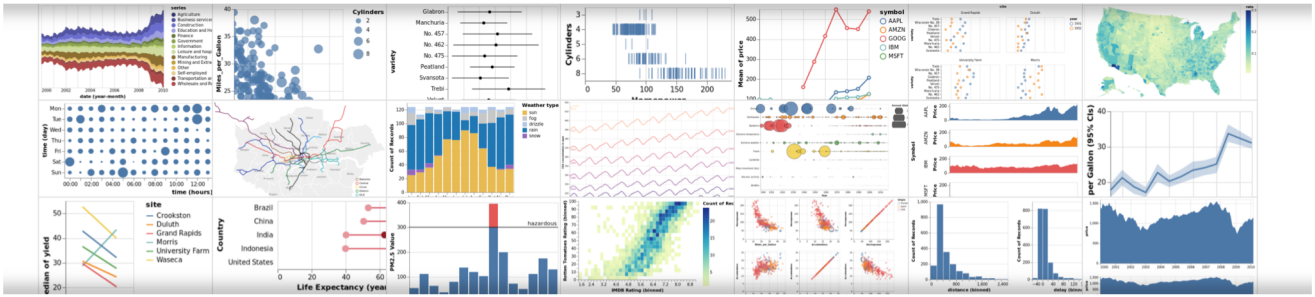
© Matt Turck (@mattturck), Jim Hao (@jimrhao), & FirstMark (@firstmarkcap) mattturck.com/bigdata2017

FIRSTMARK EARLY STAGE VENTURE CAPITAL

A.A. 2020/21

Programming based tools for Visualization

Vega-Lite – A Grammar of Interactive Graphics



https://vega.github.io/vega-lite/

Altair

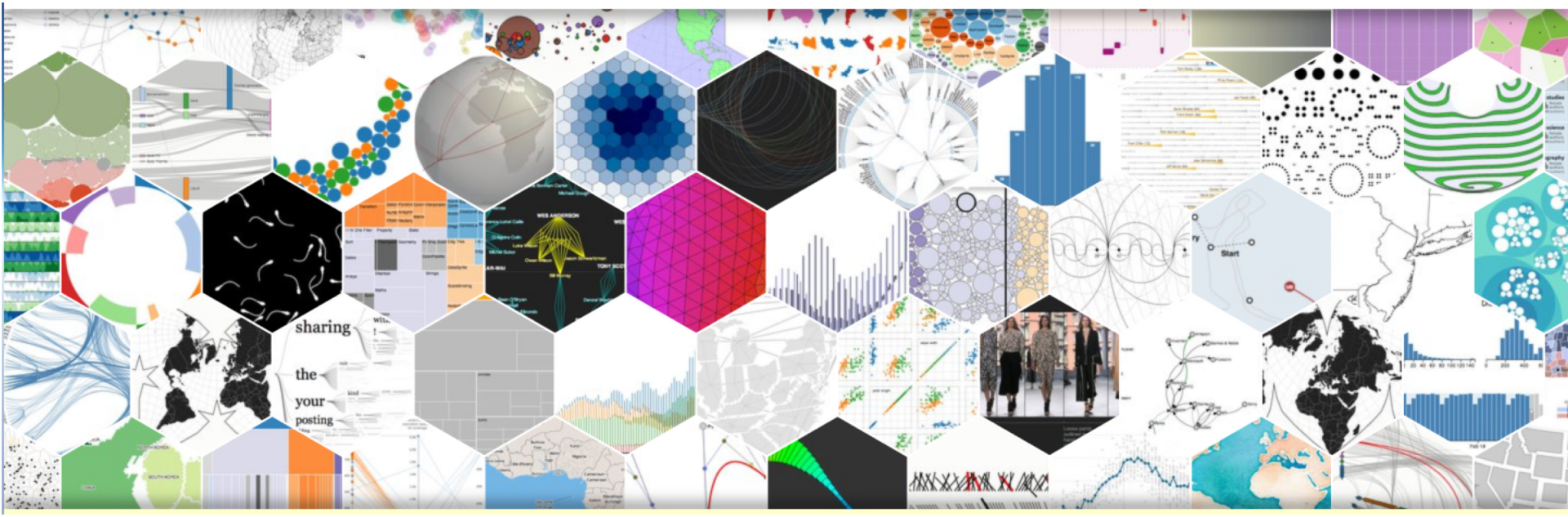
https://altair-viz.github.io/

bokeh

https://bokeh.org/

WebGL™

D3



What is D3.js?

(Data Driven Documents)

D3.js is a small, free javascript library for manipulating documents based on data

Latest version (v 6.2.0) of the library downloadable at:

https://github.com/d3/d3/releases/download/v6.2.0/d3.zip

Why D3.js?
(Data Driven Documents)

- **Web based**
- **Easy integration in most data analysis pipelines**
- **Eventually deployable both on client side or server side (Node.js)**
- **Versatile:**
  - **Recreate easily known visual paradigm**
  - **Create new ones**
  - **Gives full control on the visualization design process**

Features of D3.js

- D3 allows to bind arbitrary data to a **Document Object Model** (**DOM**), and then apply data-driven transformations to the document.
- Is possible to use D3 to generate a basic HTML table from an array of numbers, or use the same data to create an interactive SVG bar chart with smooth transitions and interaction.
- D3 solves the crux of the problem: **efficient manipulation of documents based on data**. This gives D3 extraordinary flexibility, exposing the full capabilities of underlying technologies such as CSS3, HTML5 and SVG. (Transformations, not Representations)
- D3 is extremely fast, supporting **large datasets** and dynamic behaviors for interaction and animation.
- For those common needs, D3's functional style allows **code reuse** through a diverse collection of optional modules.

DOM

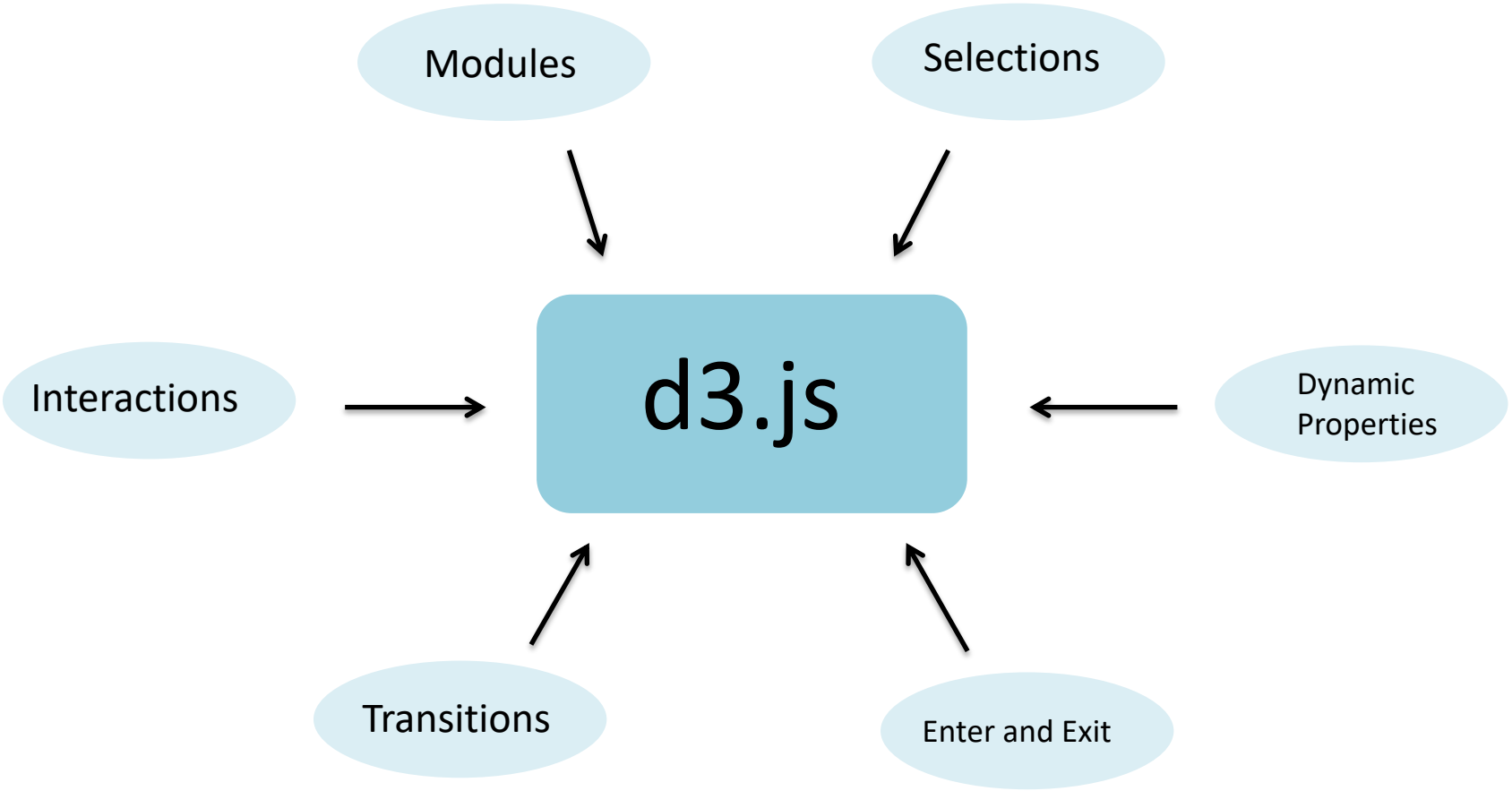
- **DOM (Document Object Model)** is a hierarchical representation of the content of an HTML document

- Each element of the document is represented as a node:
  - Linked to the parent element (node) to which it belong
  - Linked with as many children as the elements it hierarchically contains

SVG

- **SVG** (Scalable Vector Graphics) is a language for two-dimensional vector graphics based on XML.
- Most important features about SVG:
  - **Textual** : efficient compression which encourages the use of SVG in Web scope.
  - **Vectorial**: is possible to scale and zoom an SVG image keeping its quality.
  - **Open**
  - **Interactive:** by using a scripting language is possible to have an interactive SVG image
  - **Dynamic:** it is possible to create animations by using SMIL (Synchronized Multimedia Integration Language) animation language.

d3.js Components



d3.js Components

- Selections
- Dynamic Properties
- Enter and Exit
- Transitions
- Interactions
- Modules

Selections

- A **selection** is an array of elements pulled from the current document.
- D3 uses **CSS3** to select elements.
- After selecting elements, you apply **operators** to them to do stuff.
- These operators can get or set **attributes**, **styles**, **properties**, **HTML** and t**ext** content.

Selections/2

- D3 provides two top-level methods for selecting elements: **select** and **selectAll**.
  - the former selects only the first matching element,
  - the latter selects *all* matching elements in document traversal order.
- These methods accept selector strings or nodes.

Selecting Elements

#d3.**select**(selector)

- Selects the first element that matches the specified selector string, returning a single-element selection.
- If no elements in the current document match the specified selector, returns the empty selection.
- If multiple elements match the selector, only the first matching element (in document traversal order) will be selected.

#d3.**select**(node)

- Selects the specified node. This is useful if you already have a reference to a node, such as **d3.select(this)** within an event listener, or a global such as **document.body.**

Selecting Elements/2

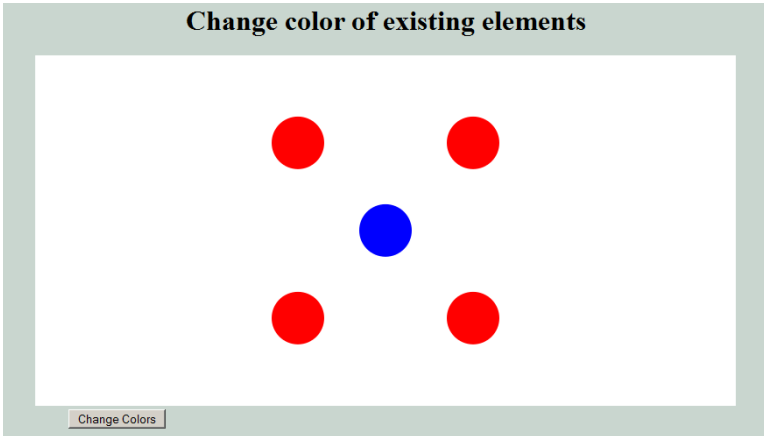
#d3.**selectAll**(selector)

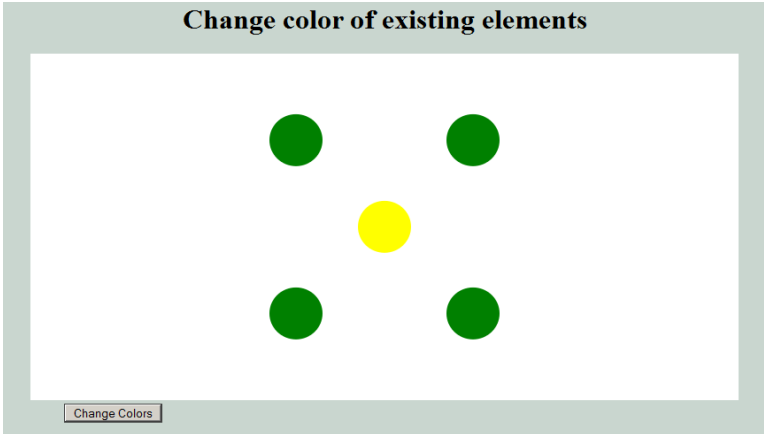
- Selects all elements that match the specified selector.
- The elements will be selected in document traversal order (top-to-bottom).
- If no elements in the current document match the specified selector, returns the empty selection.

#d3.**selectAll**(nodes)

- Selects the specified array of elements.
- Is useful if you already have a reference to nodes, such as **d3.select(this.childNodes)** within an event listener, or a global such as **document.links**.

Selecting Elements (Example)

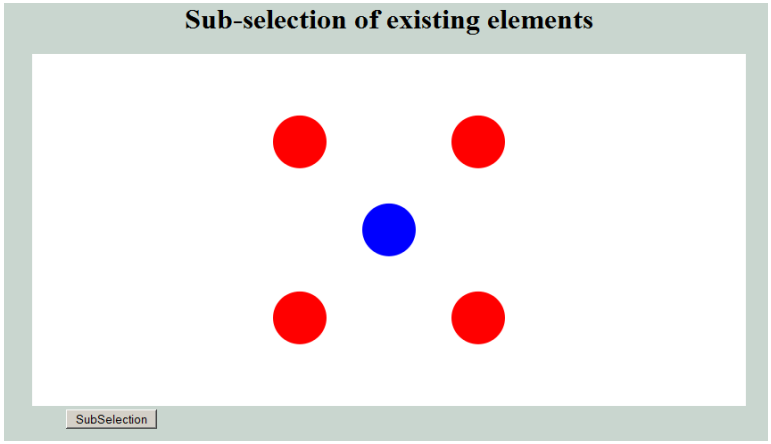


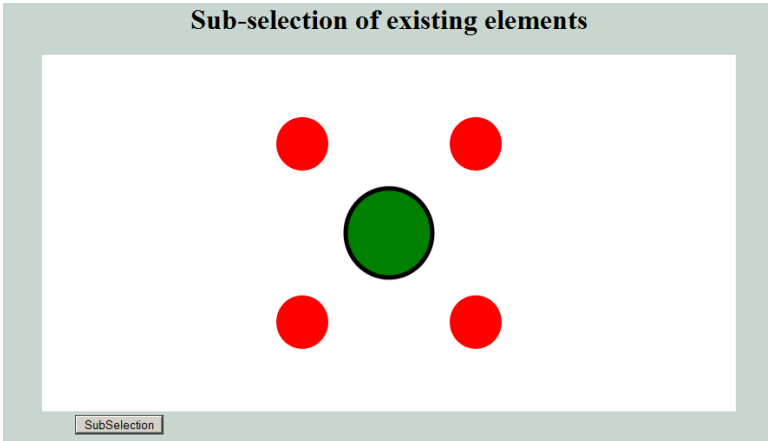



```
<script>
    function changeColor(){
        var reds=d3.selectAll(".red")
            .attr("class","green");
        var blue=d3.select("#blue")
            .attr("fill","yellow");
    }
</script>
```

Code: selection.html

Sub-selecting Elements (Example)





The first **selectAll** selects all circles and the second one selects only the circle with ID blue

```
<script>

    function subSelection(){
        var reds=d3.selectAll("circle")
            .selectAll("#blue")
            .attr("fill","green")
            .attr("stroke","black")
            .attr("stroke-width","5")
            .attr ("r","50");
    }

</script>
```

Code: [sub_selection.html]

d3.js Components

- Selections
- Dynamic Properties
- Enter and Exit
- Transitions
- Interactions
- Modules

Dynamic Properties

- D3 allows you to bind data to a selection; this data is available when computing properties.

- The data is specified as an array of arbitrary values (whatever you want), and each value to be passed as the first argument (**d**) to property functions.

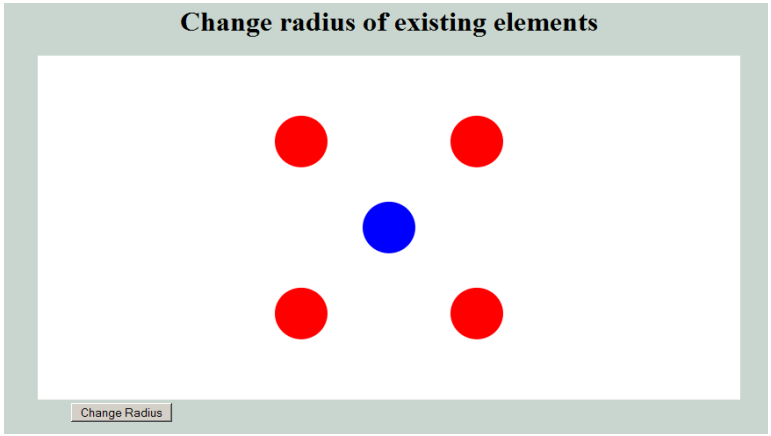
- The first element in the data array is passed to the first node in the selection, the second element to the second node, and so on.

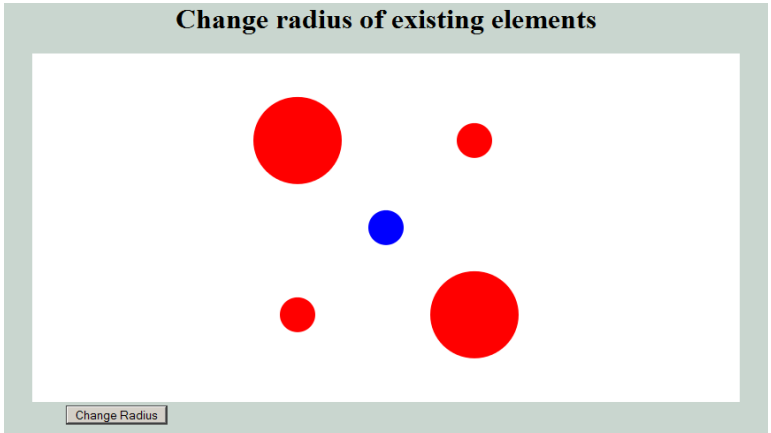
Dynamic Properties/2

#selection.**data**([*values*[, *key*]])

- Joins the specified array of data with the current selection.
- The specified *values* is an array of data values, such as an array of numbers or objects, or a function that returns an array of values.
- If a *key* function is not specified, then the first datum in the specified array is assigned to the first element in the current selection, the second datum to the second selected element, and so on.
- When data is assigned to an element, it is stored in the property **__data__**, thus making the data available on re-selection.

Dynamic Properties (Example)





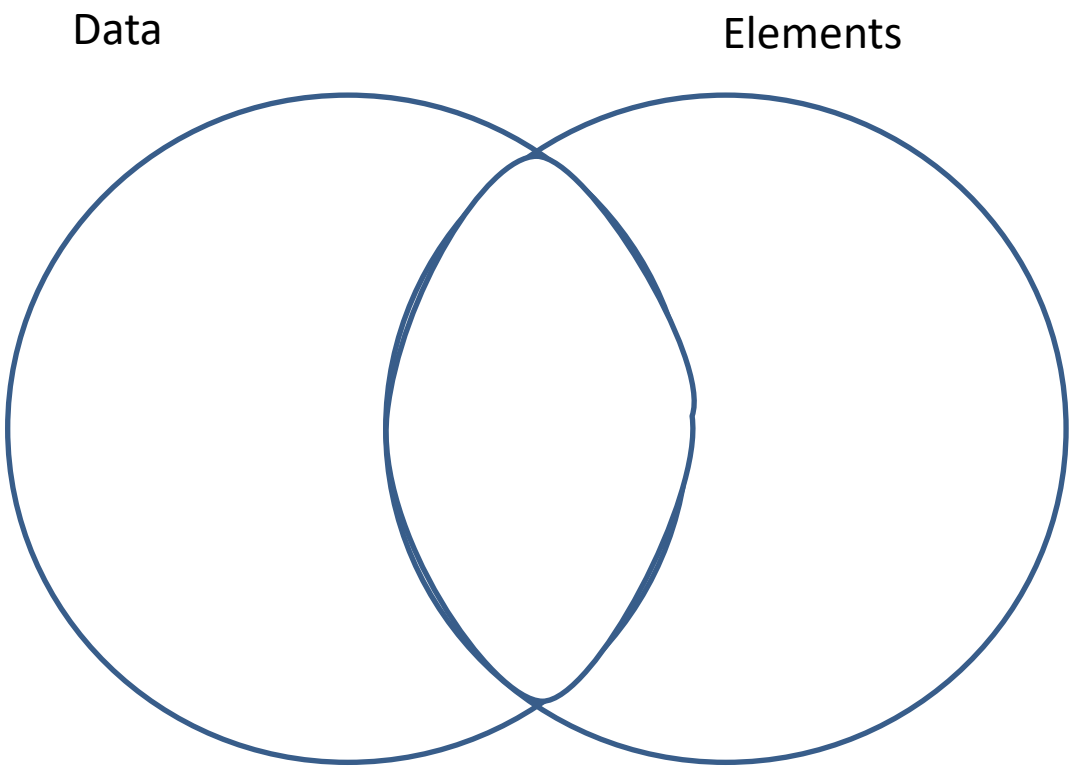

```
<script>

    function changeRadius(){
        var circle=d3.selectAll("circle")
            .data([50,20,50,20,20])
            .attr("r",function(d){
                return d;
                });
    }

</script>
```

Code: [dynamic_properties.html]

d3.js Components

- Selections
- Dynamic Properties
- Enter and Exit
- Transitions
- Interactions
- Modules

Data & Elements

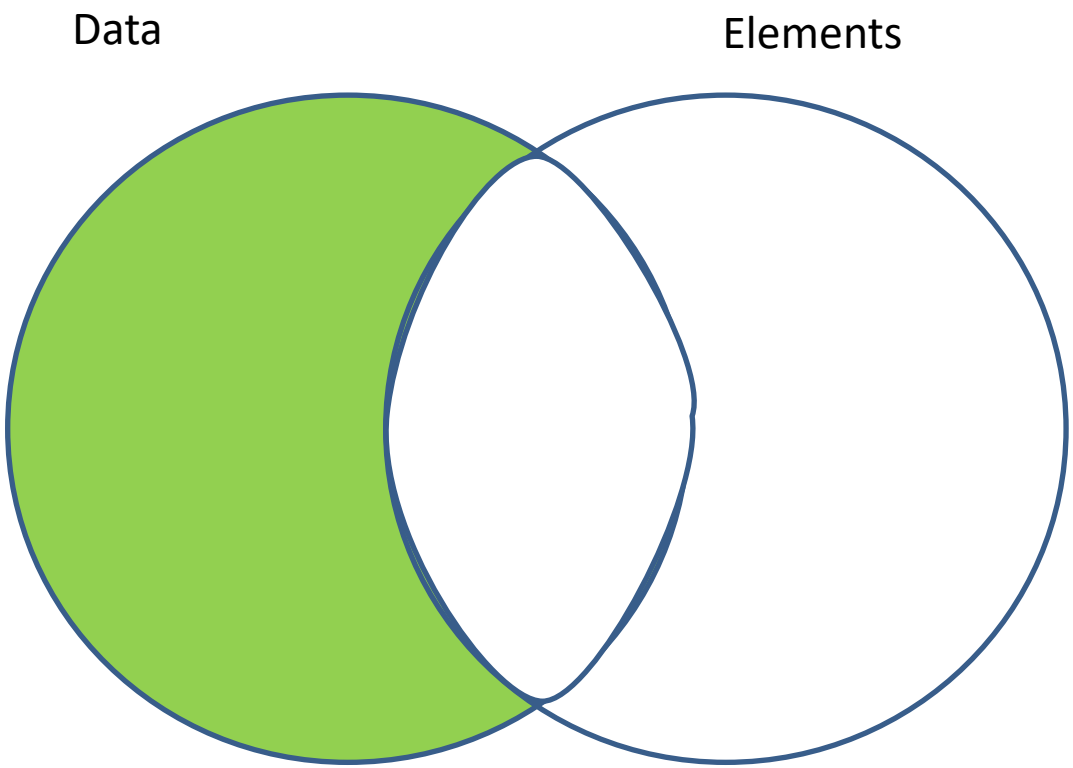


Enter and Exit Operators

#selection.**enter()**

- Returns the entering selection: placeholder nodes for each data element for which no corresponding existing DOM element was found in the current selection.
- This method is only defined on a selection returned by the **data** operator.
- The entering selection only defines **append** and **insert** operators; you must use these operators to instantiate the entering nodes before modifying any content.

Enter

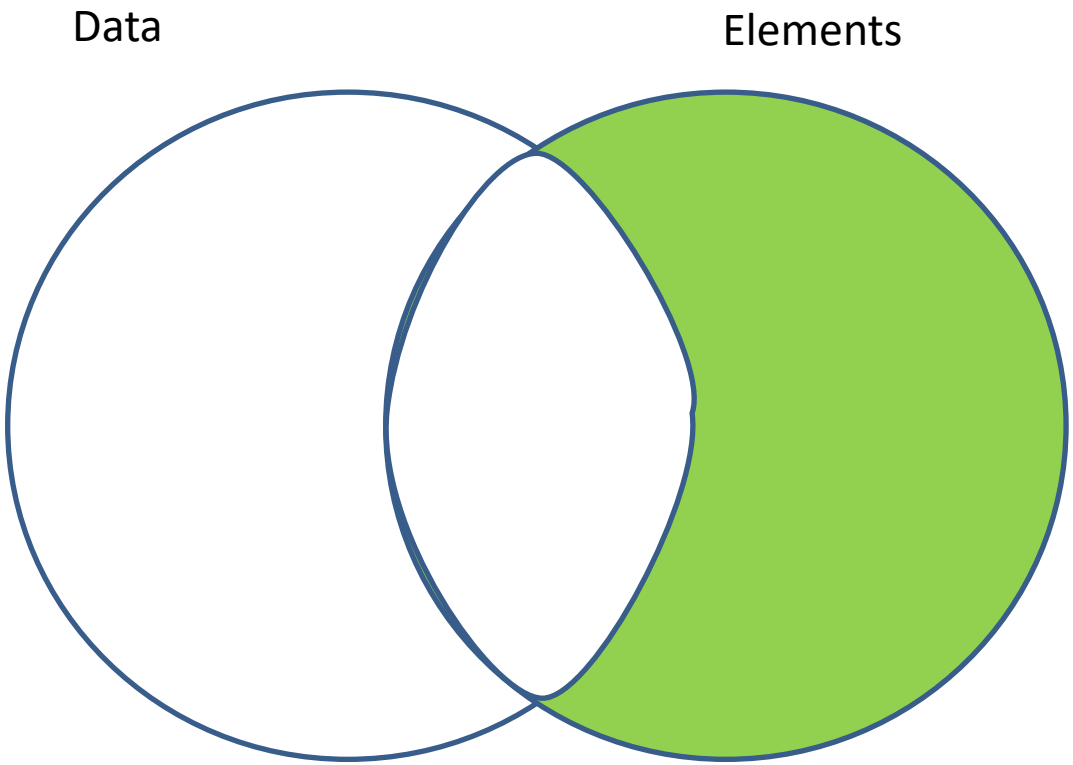


Enter and Exit Operators/2

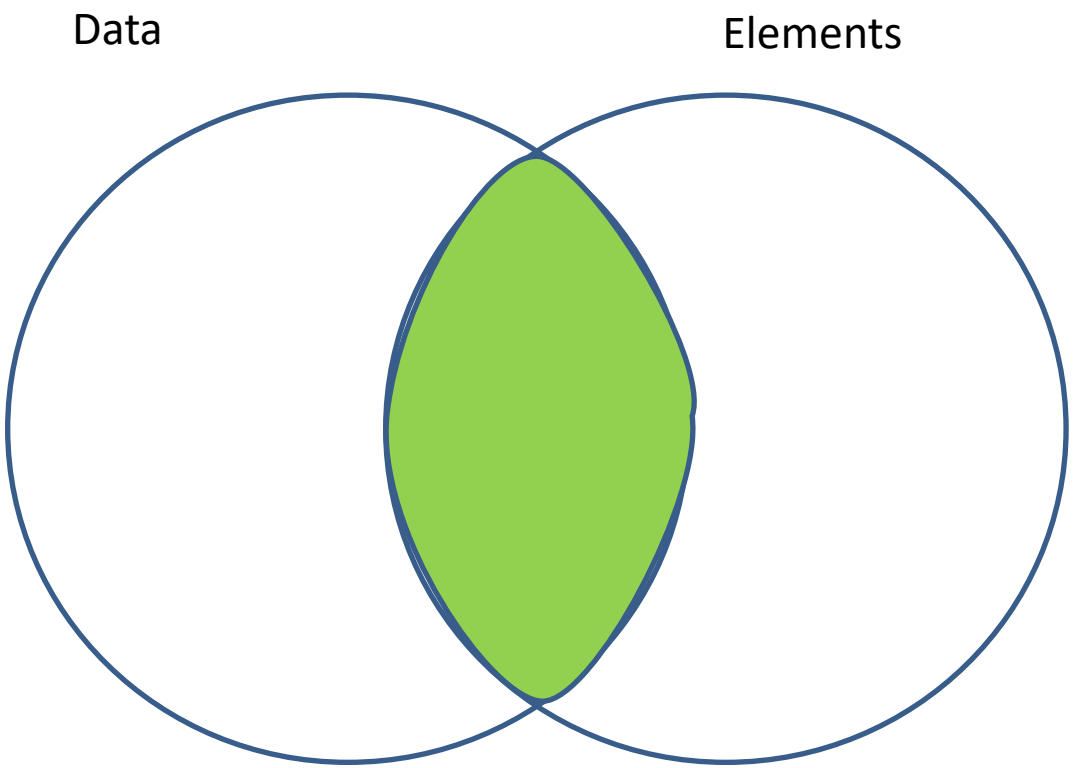
#selection.**exit()**

- Returns the exiting selection: existing DOM elements in the current selection for which no new data element was found.
- This method is only defined on a selection returned by the **data** operator.
- The exiting selection defines all the normal operators, though typically the main one you'll want to use is **remove**.
- The other operators exist primarily so you can define an exiting transition as desired.
- Note that the **exit** operator merely returns a reference to the exiting selection, and it is up to you to remove the new nodes.

Exit



UPDATE



append & remove

#selection.**append**(*name*)

- Appends a new element with the specified *name* as the last child of each element in the current selection.
- Returns a new selection containing the appended elements. Each new element inherits the data of the current elements, if any, in the same manner as **select** for subselections.
- The name must be specified as a constant, though in the future we might allow appending of existing elements or a function to generate the name dynamically.

append & remove/2

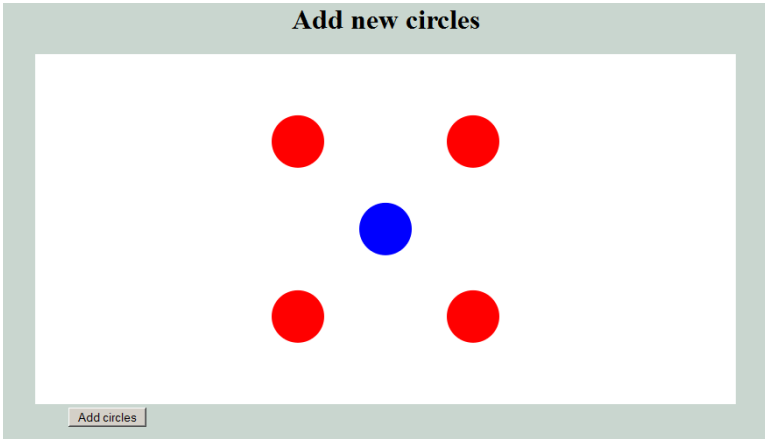
#selection.**remove**()

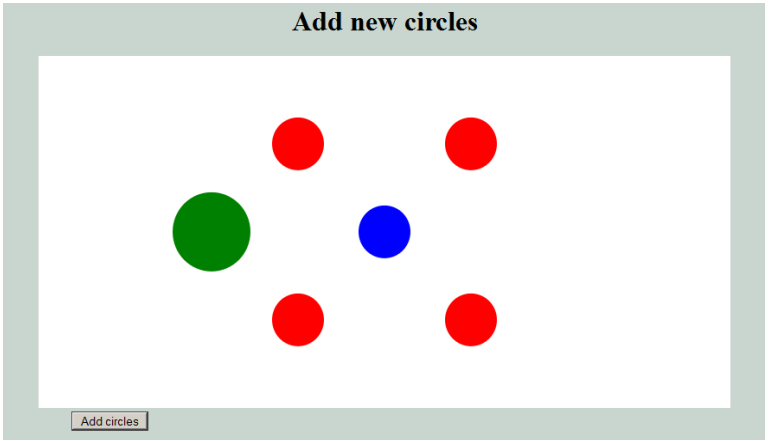
- Removes the elements in the current selection from the current document.
- Generally speaking, you should stop using selections once you've removed them, because there's not currently a way to add them back to the document.

Enter Operator

- Using the ***enter*** selection, you can add new nodes to match your data.
- When data is bound to a selection of nodes, each element in the data array is paired with the corresponding node in the selection.
- If there are fewer nodes than data, the extra data elements form the ***enter*** selection, which you can instantiate using the enter operator.
- A common pattern is to break the initial selection into three parts: the updating nodes to modify, the entering nodes to add, and the exiting nodes to remove.

Enter Operator (Example)





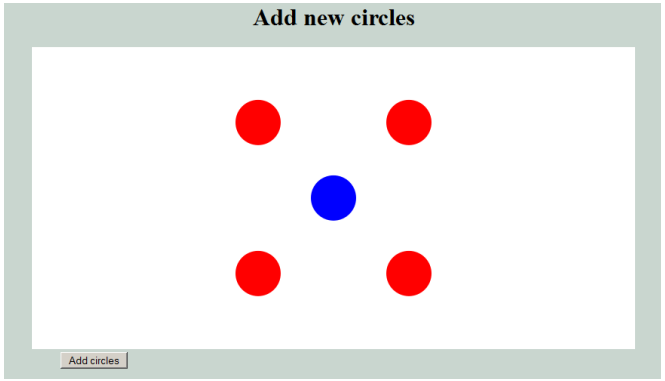

```
<script>
    function addCircle(){
        var svg=d3.select("svg")
        svg.selectAll("circle")
            .data([30,30,30,30,30,45])
            .enter()
            .append("circle")
                .attr("class","green")
                .attr("cx","200")
                .attr("cy","200")
                .attr("r",function(d){
                    return d;
                    });

    }
</script>
```

Code: [enter_operator.html]

Enter Operator (Example)/2

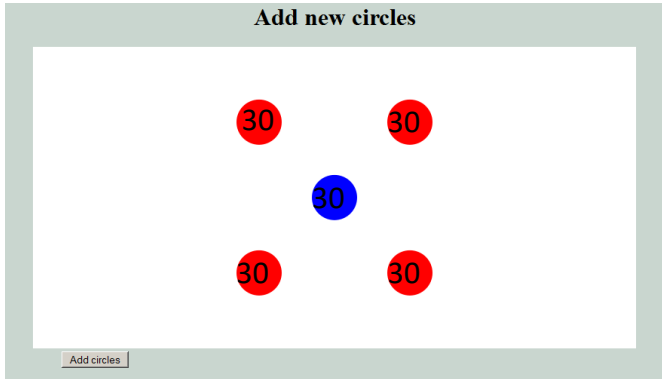
Data Matching

30	30	30	30	30	45




```
<script>
    function addCircle(){
        var svg=d3.select("svg")
        svg.selectAll("circle")
            .data([30,30,30,30,30,45])
            ......
</script>
```

45

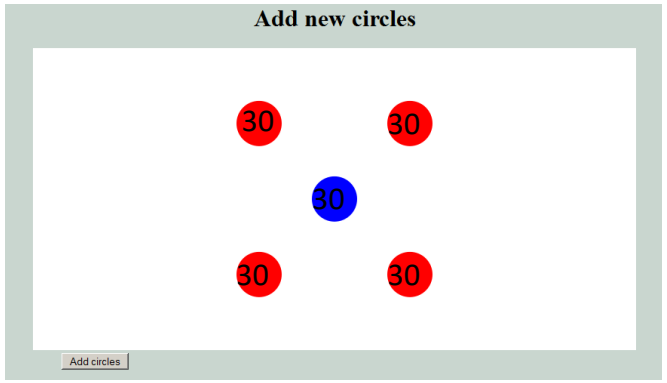


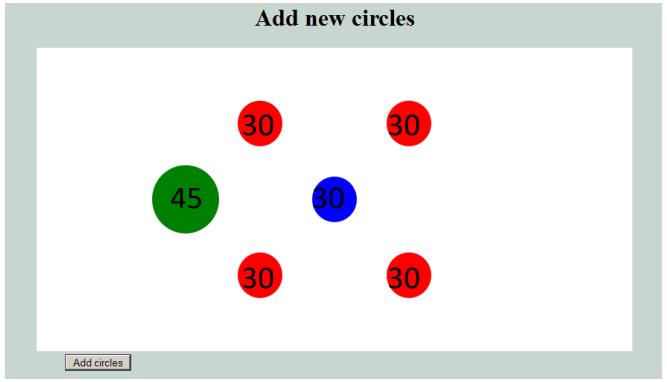
- Elements in the array macth with existing circles.
- The last element doesn't match: not exist its corresponding circle.

Enter Operator (Example)/3

Data Matching

45





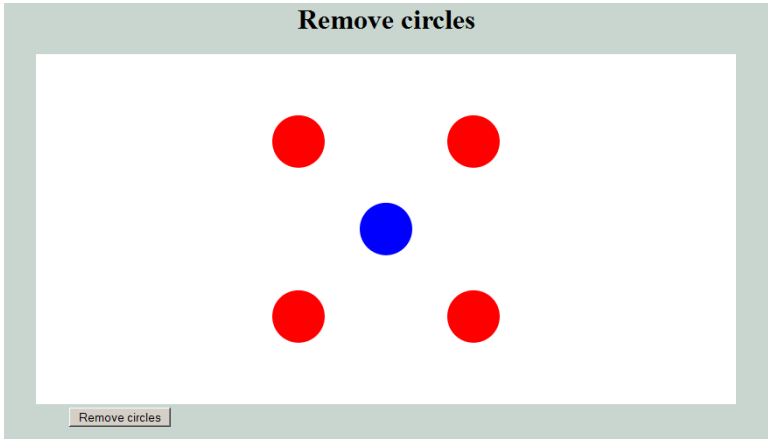

```
<script>
    function addCircle(){
        var svg=d3.select("svg")
        svg.selectAll("circle")
            .data([30,30,30,30,30,45])
            .enter()
            .append("circle")
                .attr("class","green")
                .attr("cx","200")
                .attr("cy","200")
                .attr("r",function(d){
                    return d;
                    });
    }
</script>
```

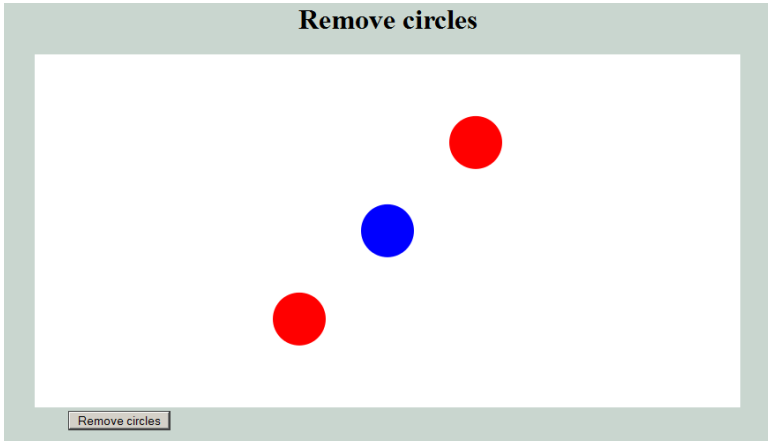
- Using **enter** operator another circle is created.

Exit Operator

- Using ***exit*** selection, you can remove nodes that are no longer needed.
- When data is bound to a selection of nodes, each element in the data array is paired with the corresponding node in the selection.
- If there are more nodes than data, the extra nodes elements,using the ***exit*** operator, will be removed.
- A common pattern is to break the initial selection into three parts: the updating nodes to modify, the entering nodes to add, and the exiting nodes to remove.

Exit Operator (Example)





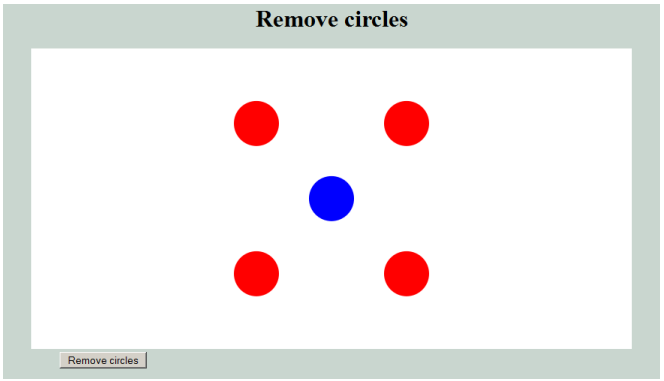

```
<script>
    function removeCircle(){
        var svg=d3.select("svg")
        svg.selectAll("circle")
            .data([30,30,30])
            .exit()
            .remove();
    }
</script>
```

Code: exit_operator.html

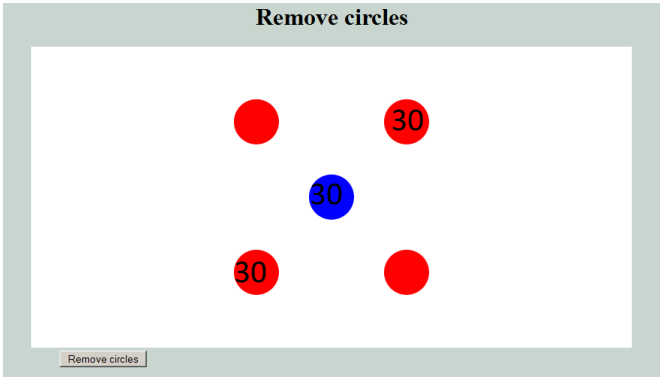
Exit Operator (Example)/2

Data Matching

30	30	30



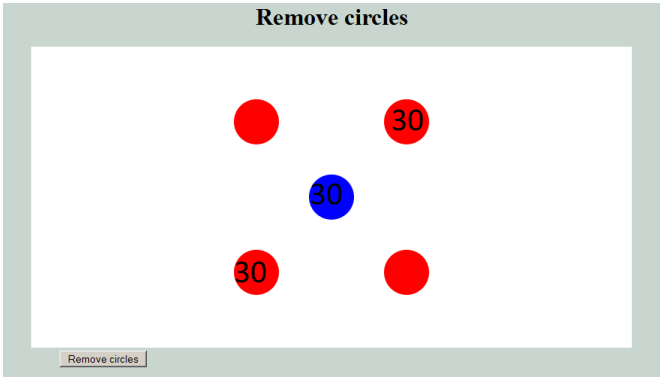

```
<script>
    function removeCircle(){
        var svg=d3.select("svg")
        svg.selectAll("circle")
            .data([30,30,30])
            ....
    }
</script>
```

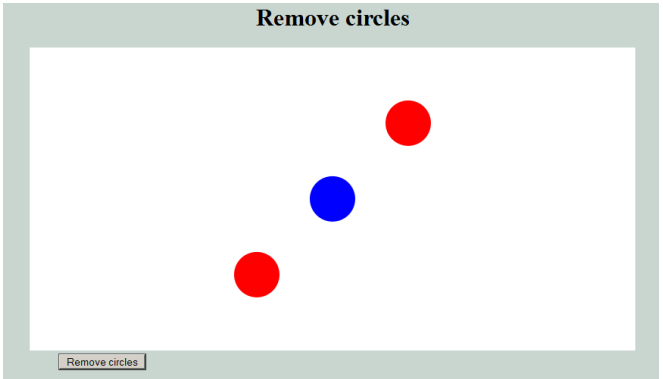


- Each element in the array matches with an existing circle; there are extra circles.

Exit Operator (Example)/3

Data Matching






```
<script>
    function removeCircle(){
        var svg=d3.select("svg")
        svg.selectAll("circle")
            .data([30,30,30])
            .exit()
            .remove();
    }
</script>
```

- Using **exit** operator extra circles are removed.

d3.js Components

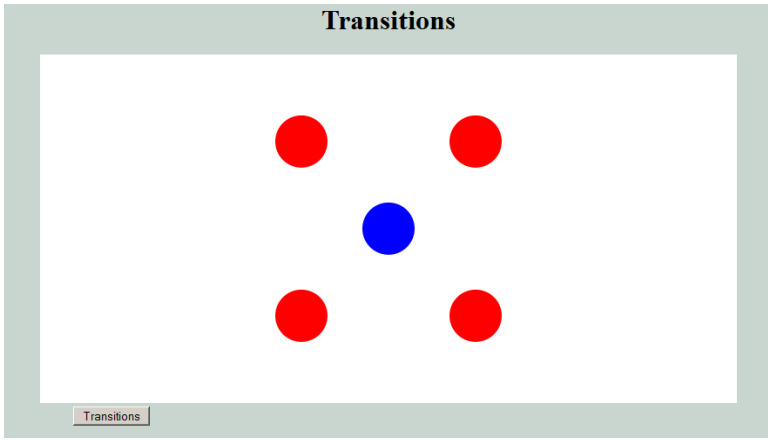
- Selections
- Dynamic Properties
- Enter and Exit
- **Transitions**
- Interactions
- Modules

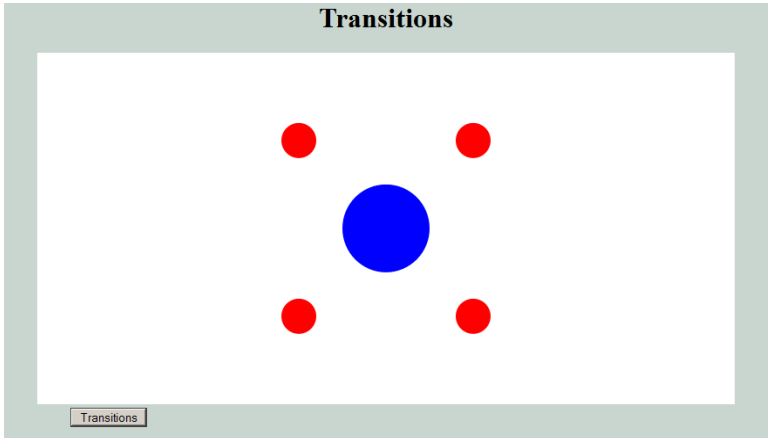
Transitions

selection.**transition**()

- Starts a **transition** for the current selection. Transitions behave much like selections, except operators animate smoothly over time rather than applying instantaneously.

Transitions (Example)






```
<script>
    function transition(){

            var svg=d3.selectAll(".red")
            .transition()
            .duration(750)
            .delay(0,005)
            .attr("r", "20");

            var svg= d3.selectAll("#blue")
            .transition()
            .duration(750)
            .delay(0,1)
            .attr("r", "50");

    }
</script>
```

Code: [transition.html]

Interactions

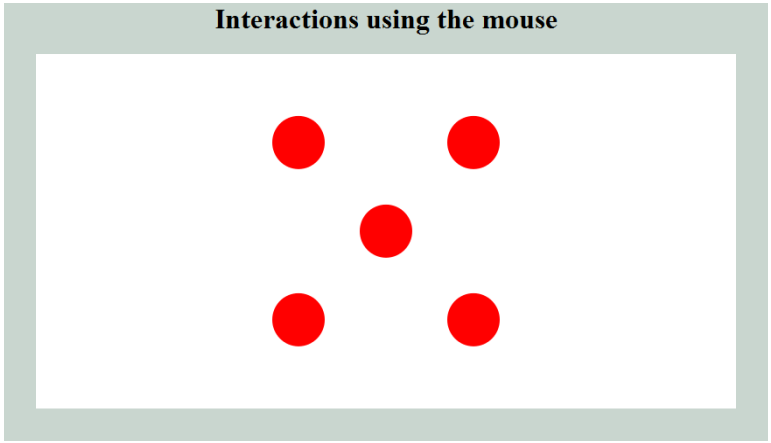
- Selections
- Dynamic Properties
- Enter and Exit
- Transitions
- **Interactions**
- Modules

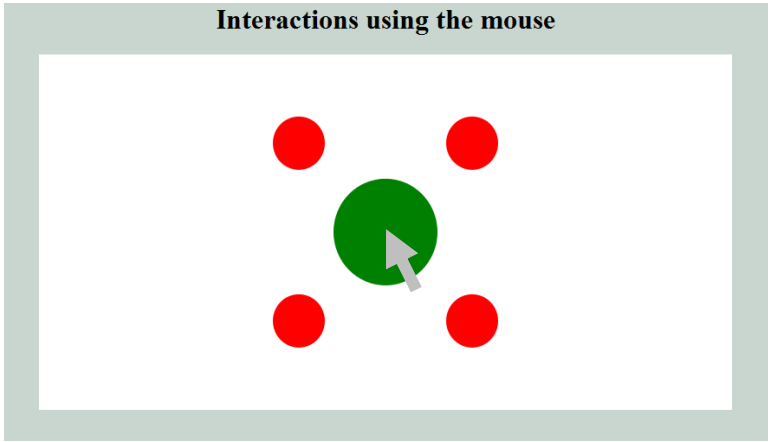
Interactions

#selection.**on**(*type*[, *listener*[, *capture*]]

- Adds or removes an event *listener* to each element in the current selection, for the specified *type*.
- The *type* is a string event type name, such as "click", "mouseover", or "submit".
- The specified *listener* is invoked in the same manner as other operator functions, being passed the current datum **d** and index **i**, with the this context as the current DOM element.

Interactions (Example)






```
<script>
    var svg=d3.select("svg");
    svg.selectAll("circle")
        .on("mouseover",
            function(){
                d3.select(this)
                    .attr("class","green")
                    .attr("r","60");
            }
        );
    svg.selectAll("circle")
        .on("mouseout",
            function(){
                d3.select(this)
                    .attr("class","red")
                    .attr("r","30");
            }
        );
</script>
```

Code: interaction.html

Modules

- Selections
- Dynamic Properties
- Enter and Exit
- Transitions
- Interactions
- Modules

Modules

- D3 is highly extensible, with optional modules available as needed, without bloating the core library.
- The only required feature of D3 is the selection implementation, along with transitions.
- For convenience, the default d3.js file also includes standard SVG shape generators and utilities, such as **scales** and data transformations.
- Several additional modules are available that are not included in the default build.

Modules/2

- The geo module adds support for geographic data, such as translating GeoJSON into SVG path data. The Albers equal-area projection is included in this module, as it is well-suited to choropleth maps.
- The geom module includes several computational geometry utilities, such as algorithms for Voronoi diagrams and convex hulls.
- The **CSV** module supports reading and writing comma-separated values, a common alternative to **JSON**.
- Lastly, the layout module includes various reusable visualization layouts, such as force-directed graphs, treemaps, and chord diagrams.

JSON

- **JSON** (JavaScript Object Notation) is a lightweight data-interchange format.
- It is easy for humans to read and write and for machines to parse and generate.
- JSON is a text format that is completely language independent.
- JSON is built on two structures:
  - A collection of name/value pairs. In various languages, this is realized as an *object*, record, struct, dictionary, hash table, keyed list, or associative array.
  - An ordered list of values. In most languages, this is realized as an *array*, vector, list, or sequence.

JSON/2

- In JSON data structures can have these forms:
  - An *object* is an unordered set of name/value pairs. An object begins with **{** (left brace) and ends with **}** (right brace). Each name is followed by **:** (colon) and the name/value pairs are separated by **,** (comma).
  - E.g. **{“name”: “Marco”, ”age”: 18}**
  - An *array* is an ordered collection of values. An array begins with **[** (left bracket) and ends with **]** (right bracket). Values are separated by **,** (comma).
  - **[{“name”: “Marco”, ”age”: 18}, {“name”: “Flavia”, ”age”: 23}]**

CSV

- **CSV** (Comma Separated Values) is a common file type used to import data from one software application to another, with commas separating the values in each field.
- CSV is often used to transfer data between databases or spreadsheet tables, and most financial software is a collection of databases or spreadsheets, making CSV a common method of transferring data between financial software or between financial software and spreadsheets or databases.

CSV-Functions

#d3.**csv** (url).then(*function(data)*)

- D3 now uses Promises instead of asynchronous callbacks to load data. Promises simplify the structure of asynchronous code, especially in modern browsers that support async and await.
- A Promise is an object representing the eventual completion or failure of an asynchronous operation.
- When the CSV data is available, the specified *function* will be invoked with the **parsed data** as the argument.
- Note that you don't need to rethrow an error—the promise will reject automatically, and you can *promise*.catch if desired

Async & Await

Async

```
1  async function f() {
2    return 1;
3  }
```

```
1  async function f() {
2    return Promise.resolve(1);
3  }
4
5  f().then(alert); // 1
```

The word “async” before a function means one simple thing: a function always returns a promise.

Await

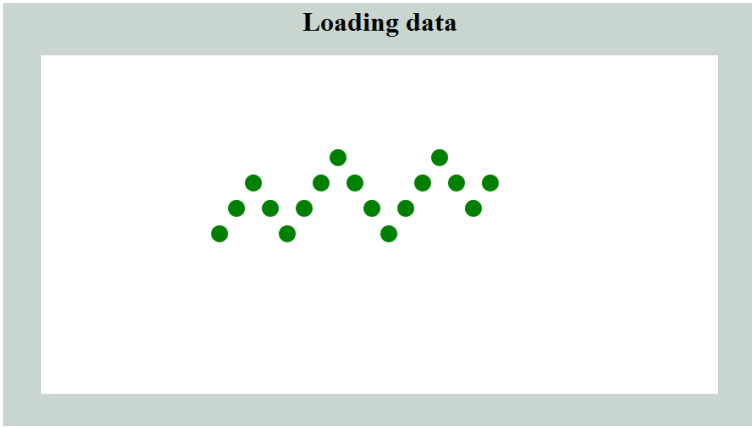
```
1  // works only inside async functions
2  let value = await promise;
```

The keyword **await** makes JavaScript wait until that promise settles and returns its result.

CSV-Loading Data (Example)

	A	B	C
1	x,y		
2	211,211		
3	231,181		
4	251,151		
5	271,181		
6	291,211		
7	311,181		
8	331,151		
9	351,121		
10	371,151		
11	391,181		
12	411,211		
13	431,181		
14	451,151		
15	471,121		
16	491,151		
17	511,181		
18	531,151		
19			

```
<script type="text/javascript">

    var data=[];
    d3.csv("coordinates.csv").then(function(rows){
                    var svg=d3.select("svg");
                    svg.selectAll("circle").data(rows).enter()
                        .append("circle")
                            .attr("cx",function(d){return d.x;})
                            .attr("cy",function(d){return d.y;})
                            .attr("r","10")
                            .attr("fill","green");
                }
            );
</script>
```



Code: [loading_data.html]

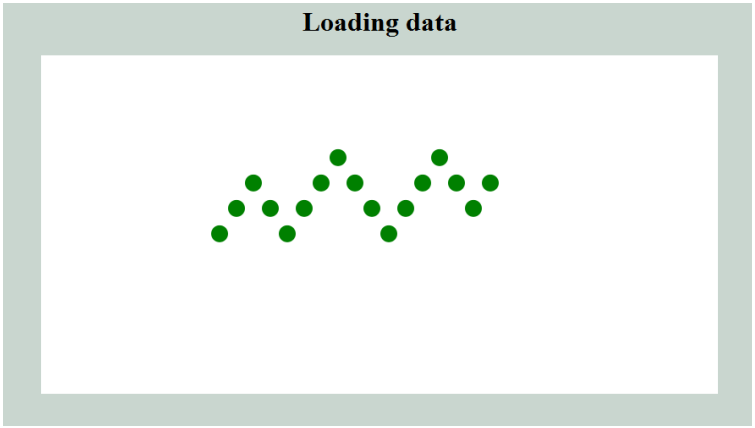
CSV-Loading Data (Example)

	A	B	C
1	x,y		
2	211,211		
3	231,181		
4	251,151		
5	271,181		
6	291,211		
7	311,181		
8	331,151		
9	351,121		
10	371,151		
11	391,181		
12	411,211		
13	431,181		
14	451,151		
15	471,121		
16	491,151		
17	511,181		
18	531,151		
19			

Alternative version:

```
const data = await d3.csv("file.csv");
console.log(data);
```

...



References

D3 official website:
http://d3js.org/

Complete API:
https://github.com/d3/d3/blob/master/API.md

Tutorials:
https://github.com/d3/d3/wiki/Tutorials