Information Visualization Frameworks

SWE 432, Fall 2016

Design and Implementation of Software for the Web

Today

- How do we build an information visualization?
  - D3.js

For further reading:

https://d3js.org/ — Official documentation & tutorial

Series of tutorials explaining core concepts of d3:
https://bost.ocks.org/mike/bar/
https://bost.ocks.org/mike/bar/2/

Information visualization

- Technology has made data **pervasive**
  - health, finance, commerce, customer, travel, demographics, communications, …
  - some of it "**big**"
- Information visualization: the use of interactive visual representations to amplify cognition
  - e.g., discover insights, answer questions

Graphics is the visual means of resolving logical problems.

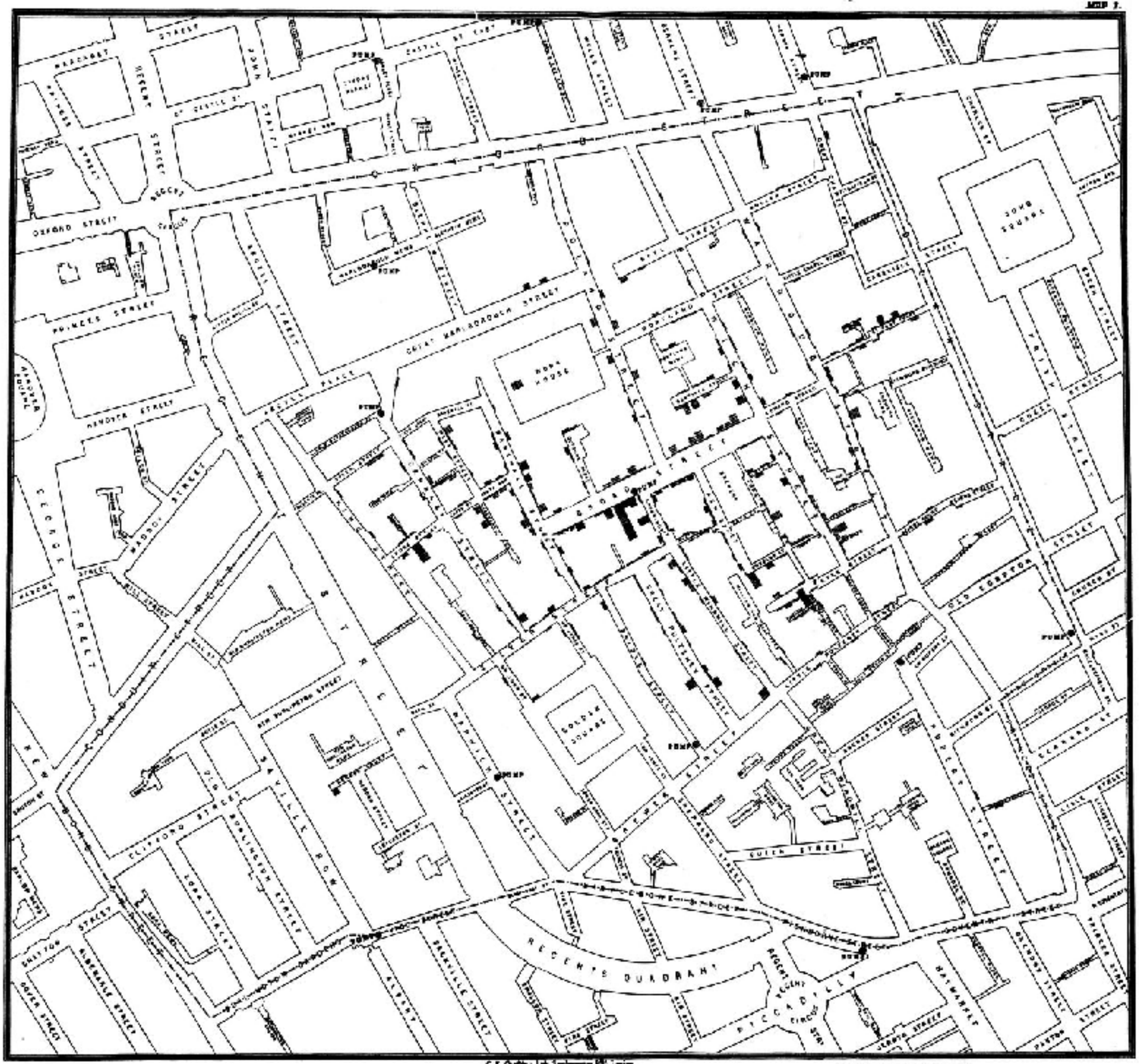
-Bertin (1977)

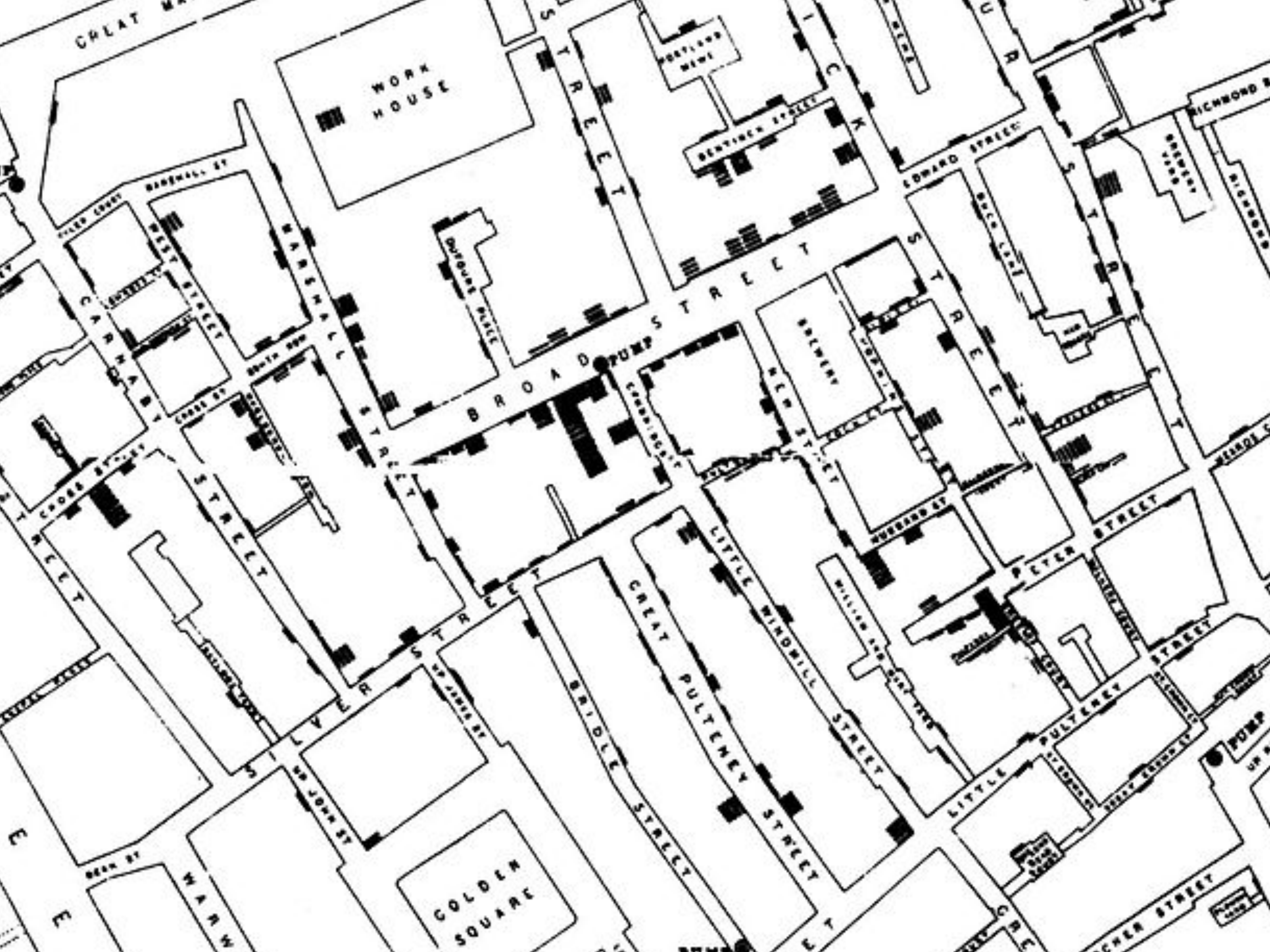
Cholera Epidemic in London, 1854

- >500 fatal attacks of cholera in 10 days
  - Concentrated in Broad Street area of London
  - Many died in a few hours
- Dominant theory of disease: caused by noxious odors
- Afflicted streets deserted by >75% inhabitants

John Snow

- Set out to investigate **cause**
- Suspected it might be due to water from community **pump**
- Tested water —> no obvious impurities
- What more evidence could there be?
  - Listed 83 deaths, plotted on map

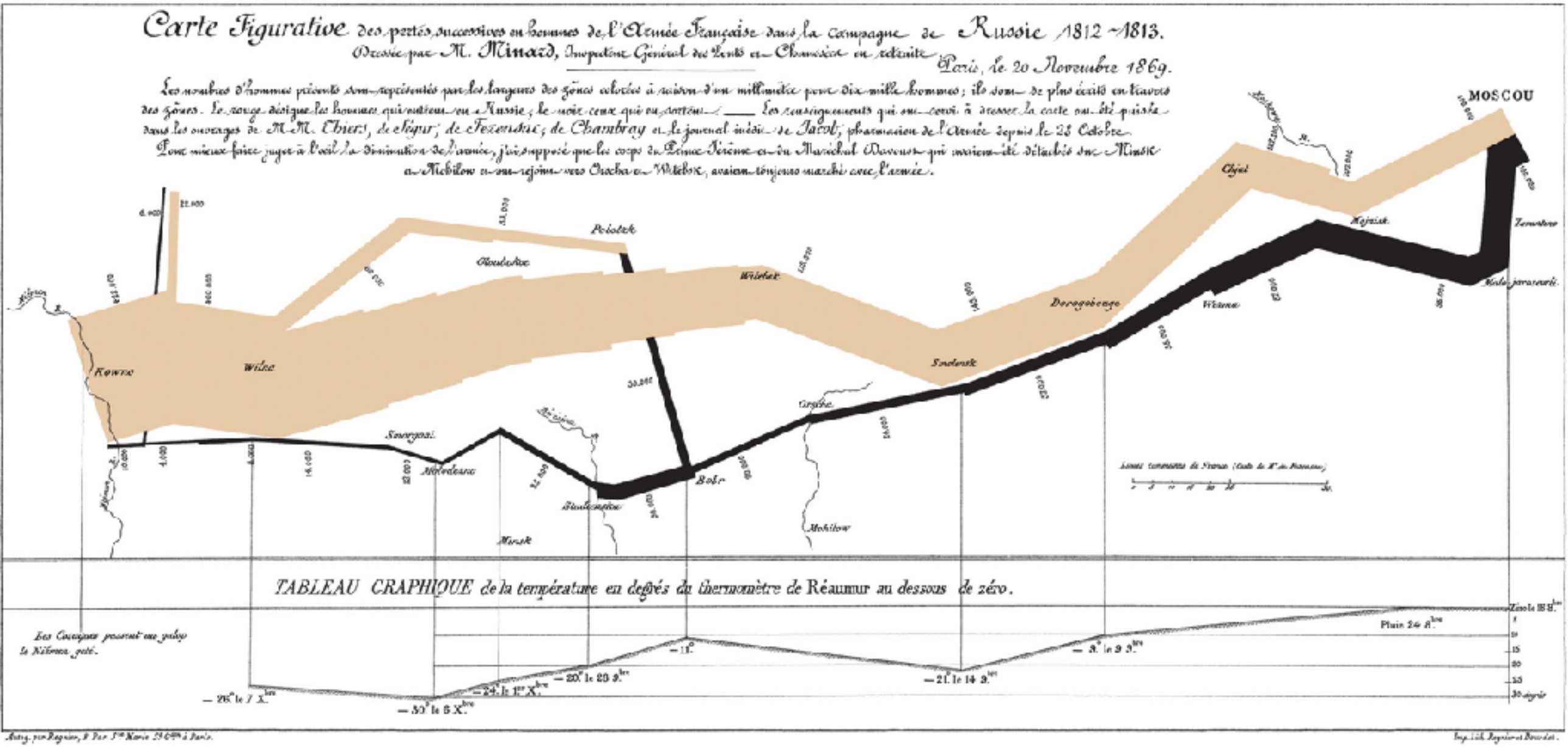

MAP 1.
OXFORD STREET
REGENT STREET
PRINCES STREET
GREAT MARLBOROUGH STREET
ARGYLL PLACE
NEWPORT STREET
CHARING CROSS ROAD
HAYMARKET
JOHN STREET
CASTLE ST EAST
BROAD STREET
POLAND STREET
BERNERS STREET
NOEL STREET
WARDOUR STREET
SOHO HOUSE
SOHO SQUARE
GOLDEN SQUARE
OLD COMPTON STREET
CLIFFORD STREET
SAVILLE ROW
BURLINGTON GARDENS
BURLINGTON STREET
NEW BURLINGTON STREET
REGENT'S QUADRANT
PICCADILLY CIRCUS
PICCADILLY
REGENT STREET
GLASSHOUSE STREET
BRUNSWICK HOUSE
ARGYLL HOUSE
PUMP
C.F. Griffin, Ltd, Southampton Row, London
SCALE 50 INCHES TO A MILE.


WORK HOUSE
PORTLAND MEWS
BENTINCK STREET
EDWARD STREET
BROAD STREET
PUMP
CARNABY
MARSHALL ST
WEST STREET
DUFOURS PLACE
BREWERY
NEW STREET
HUBBARD ST
PETER STREET
LITTLE WINDMILL STREET
GREAT PULTENEY STREET
BRIDLE STREET
SILVER STREET
GOLDEN SQUARE
LITTLE PULTENEY STREET
CROSS STREET
ARCHER STREET

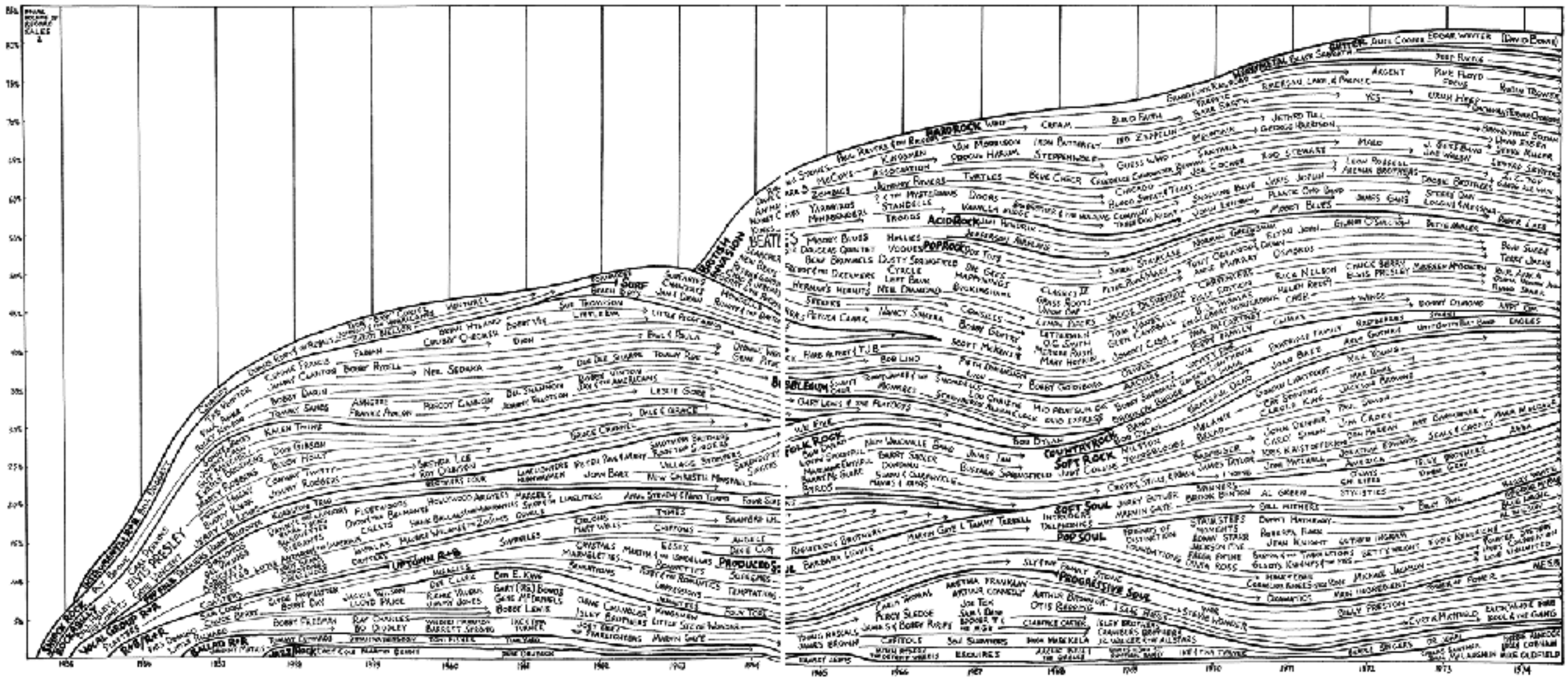
Investigation and aftermath

- Based on **visualization**, did case by case investigation
- Found that **61 / 83** positive identified as using well water from Broad Street pump
- Board ordered pump-handle to be removed from well
- Epidemic soon **ended**
- Solved centuries old question of how cholera spread

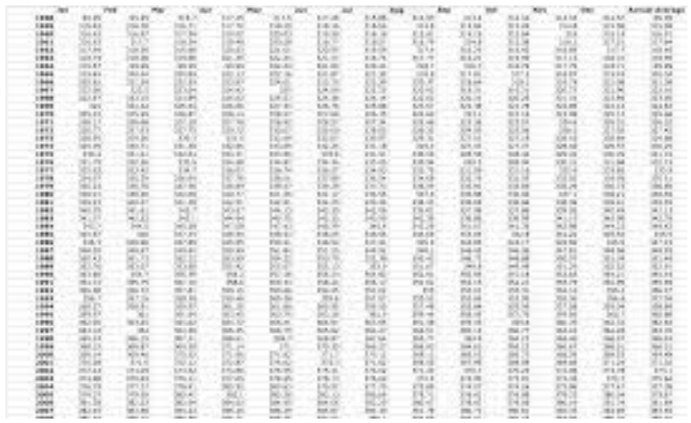
Charles Minard's Map of Napoleon's Russian Campaign of 1812

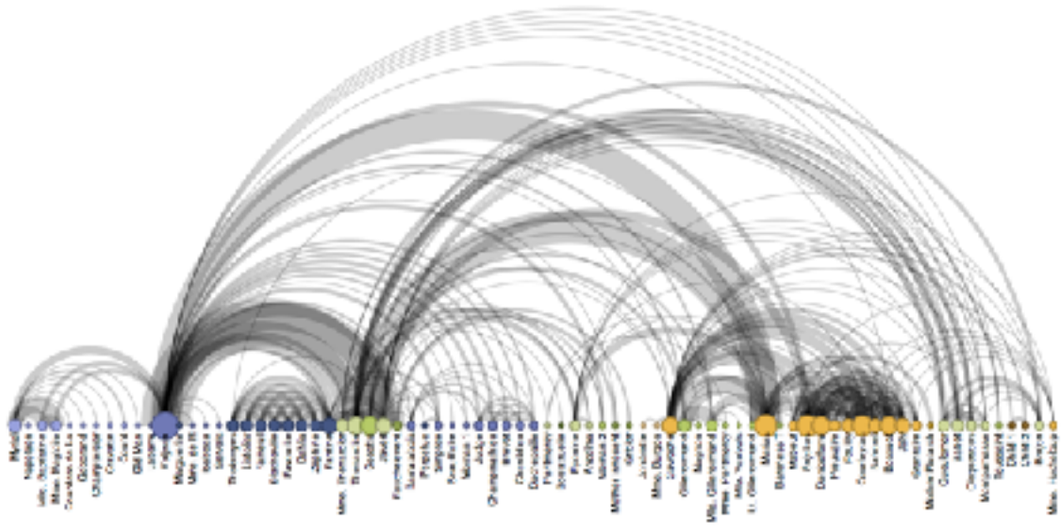


Chapel & Garofalo, Rock 'N Roll is Here to Pay: The History and Politics of the Music Industry



What is an information visualization?





- Data —> Visual representation
  - Rows in data table —> elements in data visualization
    - e.g., historical person —> circle in visualization
  - Columns of data —> visual variables
    - e.g., relationship to another person —> edge in network visualization

Some challenges in information visualizations

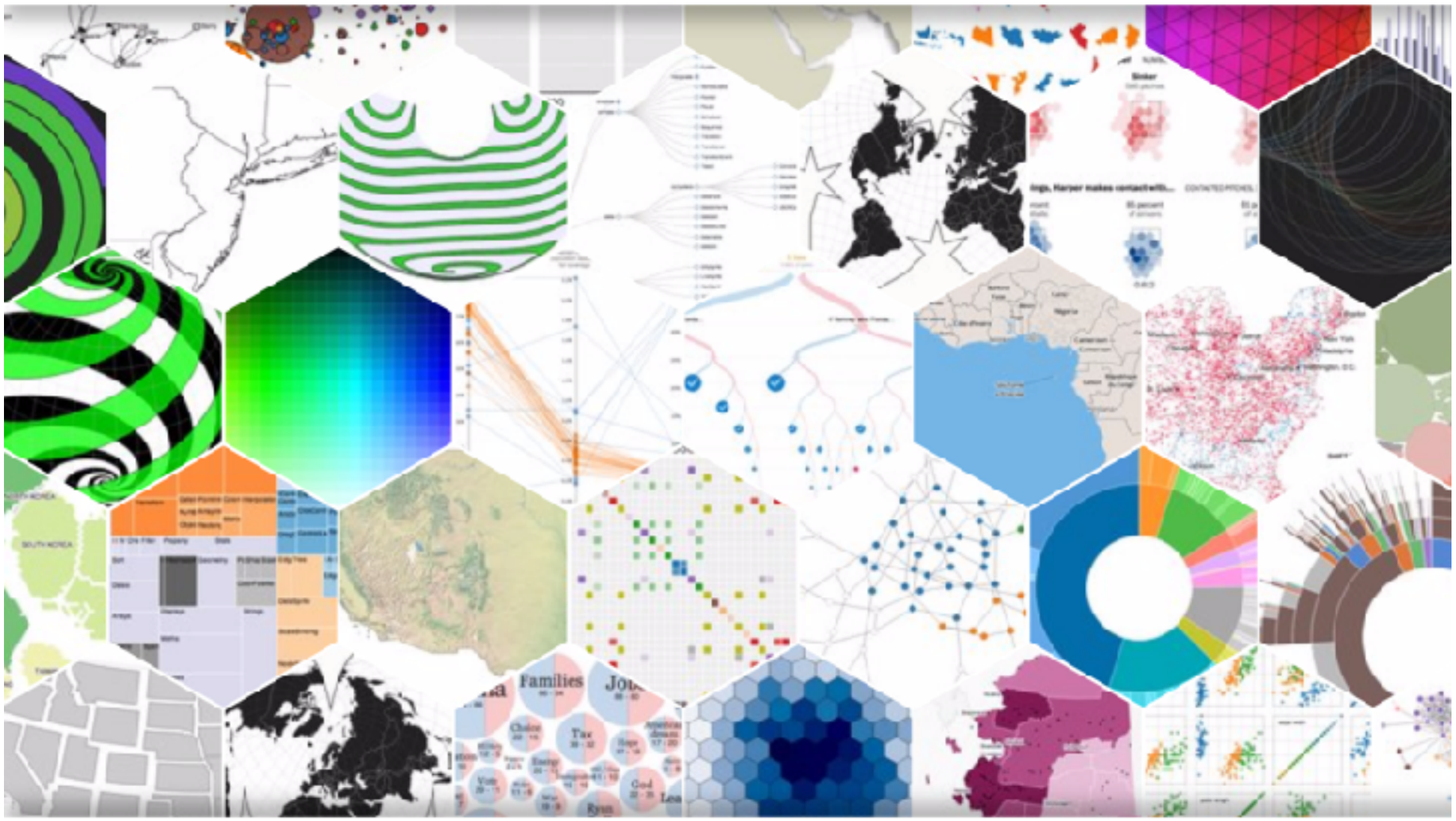
- Data binding
  - You have data. How do you create corresponding visual elements?
  - How do you update the visual elements if the data changes?
    - Or the user updates what they want to see…
- Scales
  - How do data values correspond to position, size, color, etc. of visual elements?
- Transitions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

https://d3js.org

Overview Examples Documentation Source

Fork me on GitHub

Data-Driven Documents



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

D3.js

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

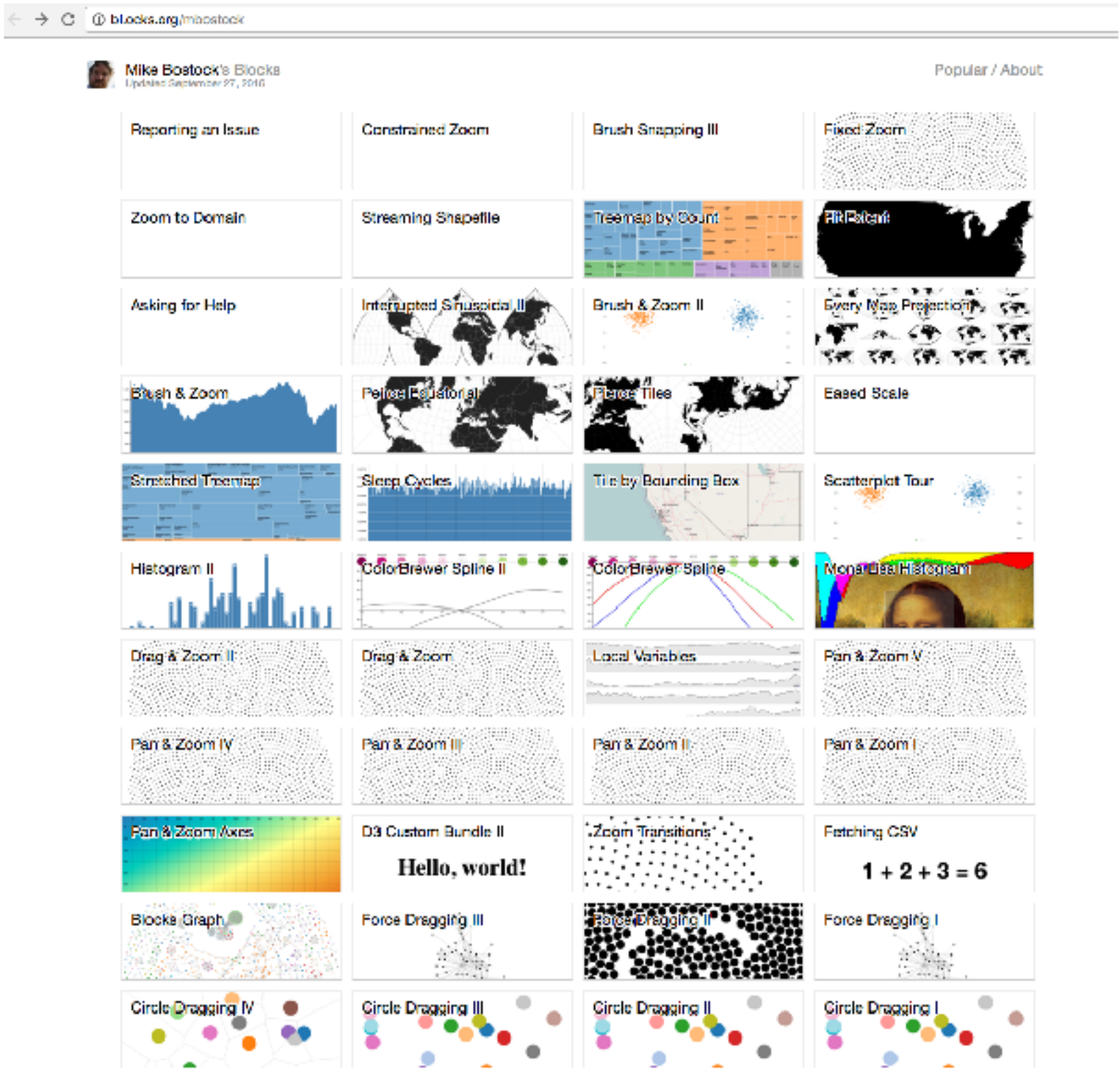
Using D3.js

```
<script src="https://d3js.org/d3.v4.min.js"></script>
```

- Or it works with NPM too

Learning D3

- Many tutorials
- Many, **many** examples
  - Ok to copy and paste IF you cite source
  - Frequent pattern: copy similar visualization, customize for your needs
- But… be careful you use d3 v4
  - Current version



Key concepts we'll cover today

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

Selections

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

==

```
$(“p”).css(“color”, “white”);
```

==

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

Dynamic properties

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

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

P1

P2

P3

P4

Dynamic properties

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

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

P1

P2

P3

P4

Dynamic properties

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

P1

P3

P4

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

Styling elements

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

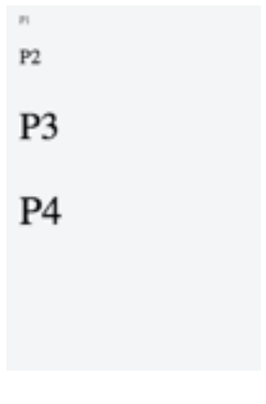
Data binding

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

Data binding

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

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



- Bind *data* with visual element.

Data binding is persistent

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

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

p.style("color", "blue");
```

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

How do we deal with changing data?

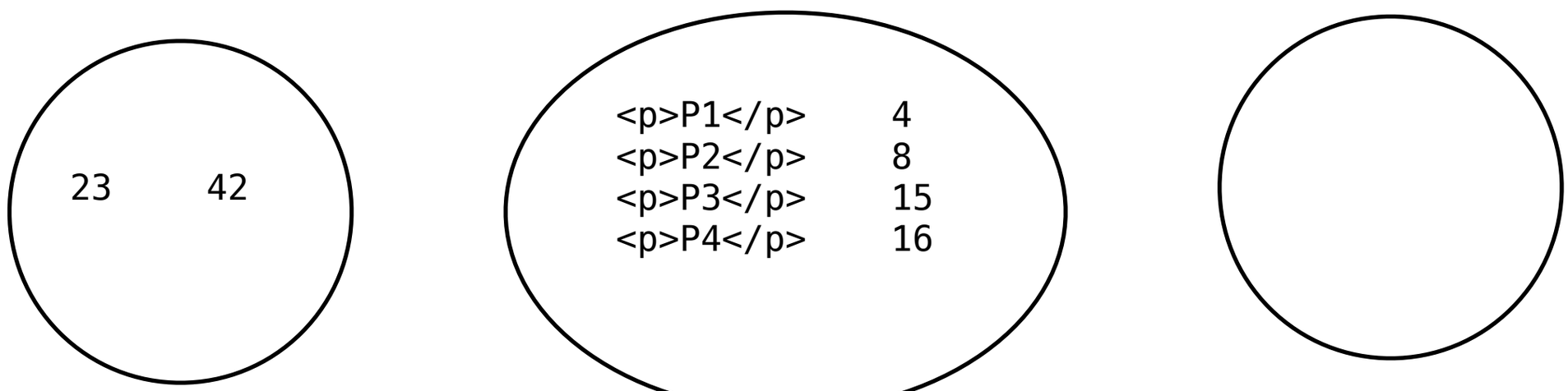
Handling Changing Data

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

Thinking in Joins

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

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



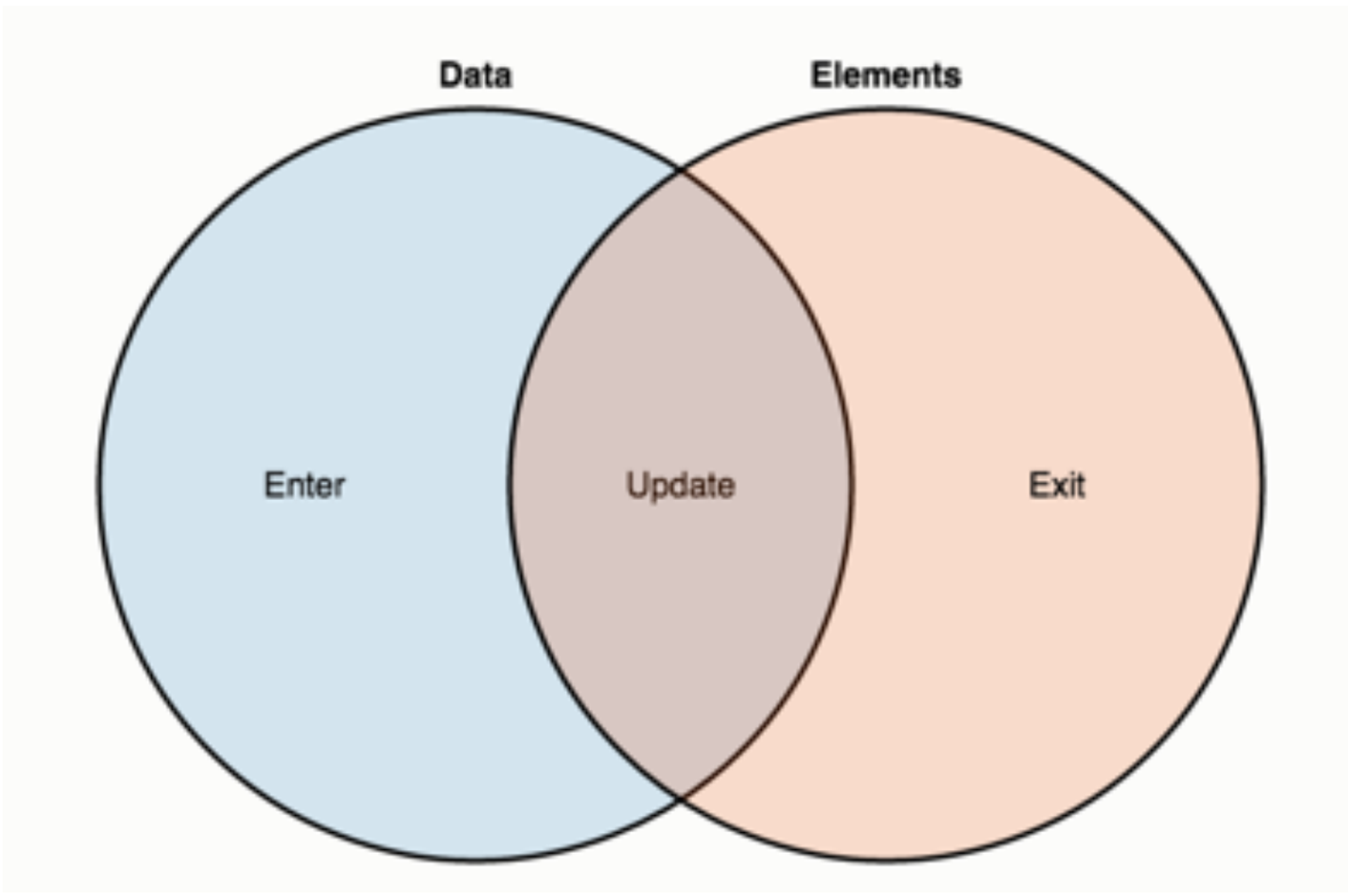

```
.enter(…)        // update (default)        .exit(…)
```

Stuff not on left | Stuff in both | Stuff not on right

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

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

Thinking in joins

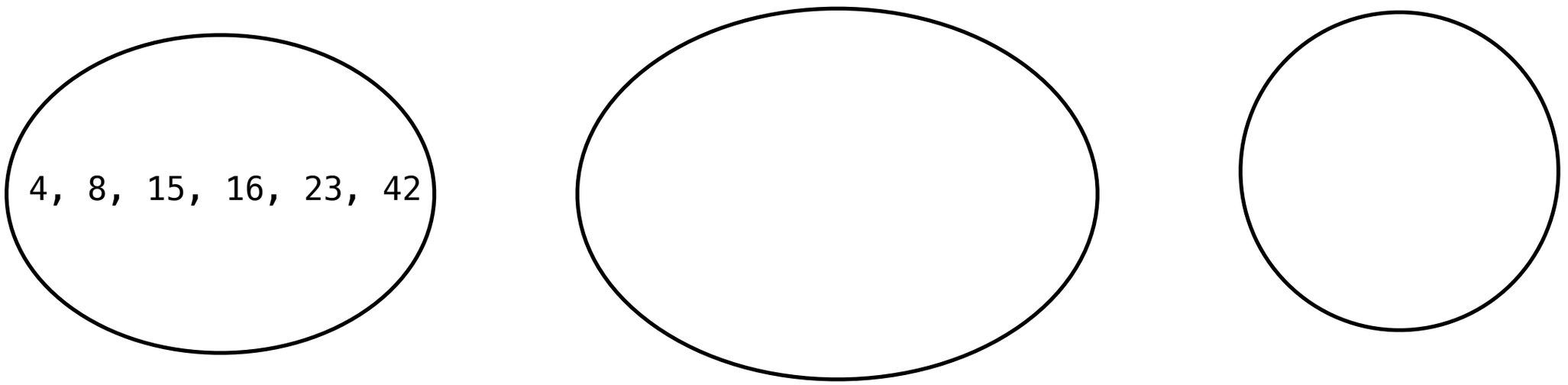


- Extra data —> enter set
- Matched data with elements —> update set
- Extra elements —> exit set

Thinking in Joins

```
<!– no p elements –>
```

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




```
.enter(…)
```

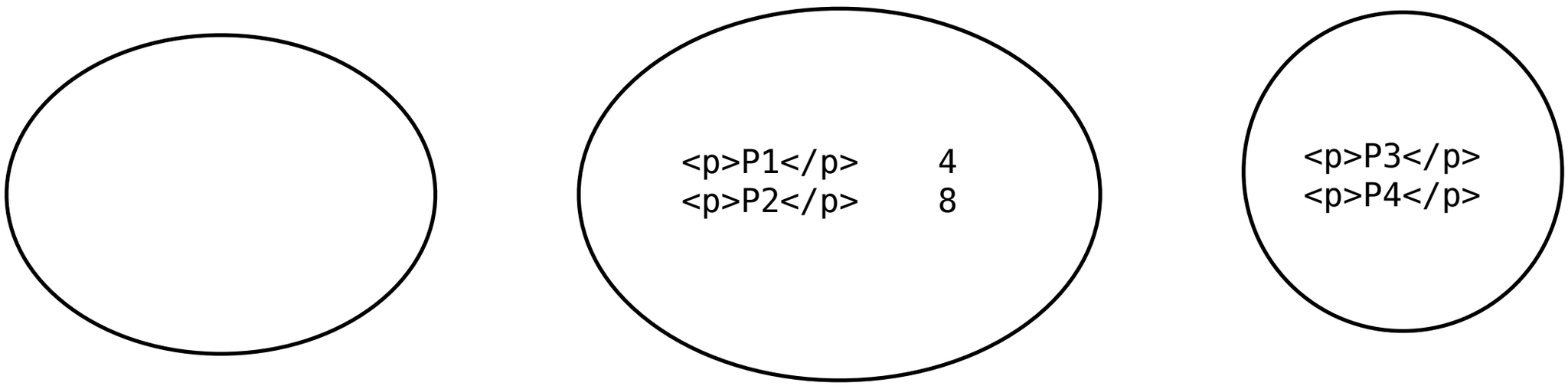
```
// update (default)
```

```
.exit(…)
```

Thinking in Joins

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

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



Creating elements

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

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

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

Creating elements

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

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

Putting it together

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

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

// Exit…
p.exit().remove();
```

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

Demo: Really Simple Bar Chart

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

Loading data

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

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

Scaling to fit data

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

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

Scales

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

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

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

Shapes and paths

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

SVG: Scalable Vector Graphics

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

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

SVG: Example

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

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

Demo: Static SVG Bar Chart

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

Demo: Generated SVG Bar Chart

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

Transitions

```
d3.selectAll("circle").transition()
    .duration(750)
    .delay(function(d, i) { return i * 10; })
    .attr("r", function(d) { return Math.sqrt(d * scale); });
```

- Transitions, just like CSS transitions, specify the animation by which new visual style appears
- Examples of what can be described
  - duration: how long is transition
  - delay: how long before transition starts
  - attr, text, style, etc.: what property should be set

Some other D3 features

- Layout
  - Computes position for elements (e.g., network visualization)
  - Usually just reuse an existing layout
- Interpolators
  - Take a parameter in domain space, produce output
  - Sounds like scale…
    - But can use it for arbitrary data types (colors, objects, …)
- Zooming
- Lots of functionality specialized for a specific set of visualizations

- But remember, it's built directly on HTML / CSS / JS / SVG.
  - Can use as much (or as little) of the D3 abstractions as desired
  - Only need to use D3 abstractions to the extent that they help

Using D3

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

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