# Visualization Frameworks

Hua Guo

Some slides borrowed from Heer and Munzner. References at back.

# Road Map

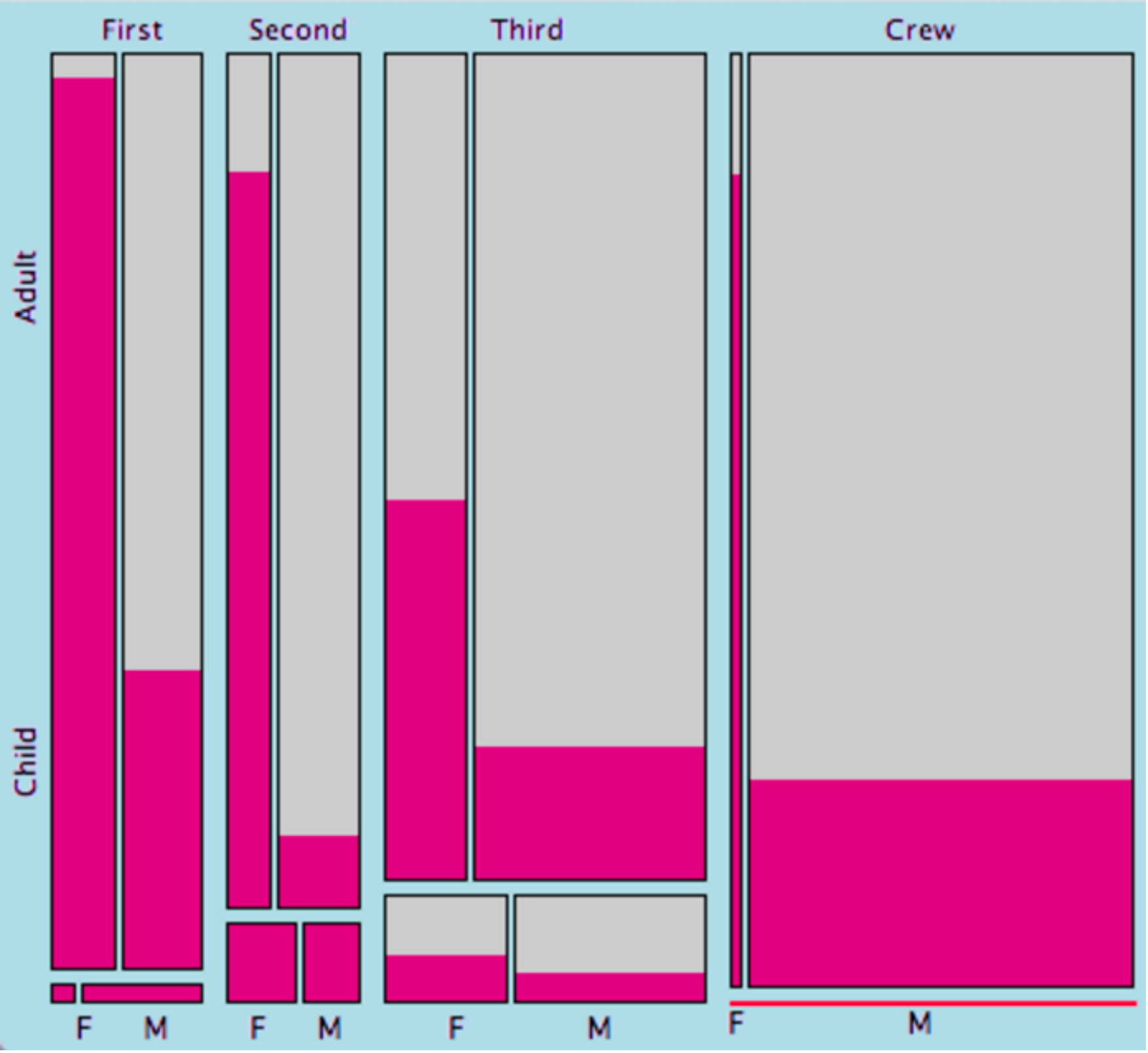
- What is a visualization framework?
- Popular visualization frameworks
  - Different approaches to creating visualizations
  - Pros and cons
- A D3 primer
  - Data binding
  - General update pattern
  - Scales, animations, and interactions
  - Additional libraries and Vega

# What is a visualization framework? (and why we need them)

# Two visualizations of the Titanic

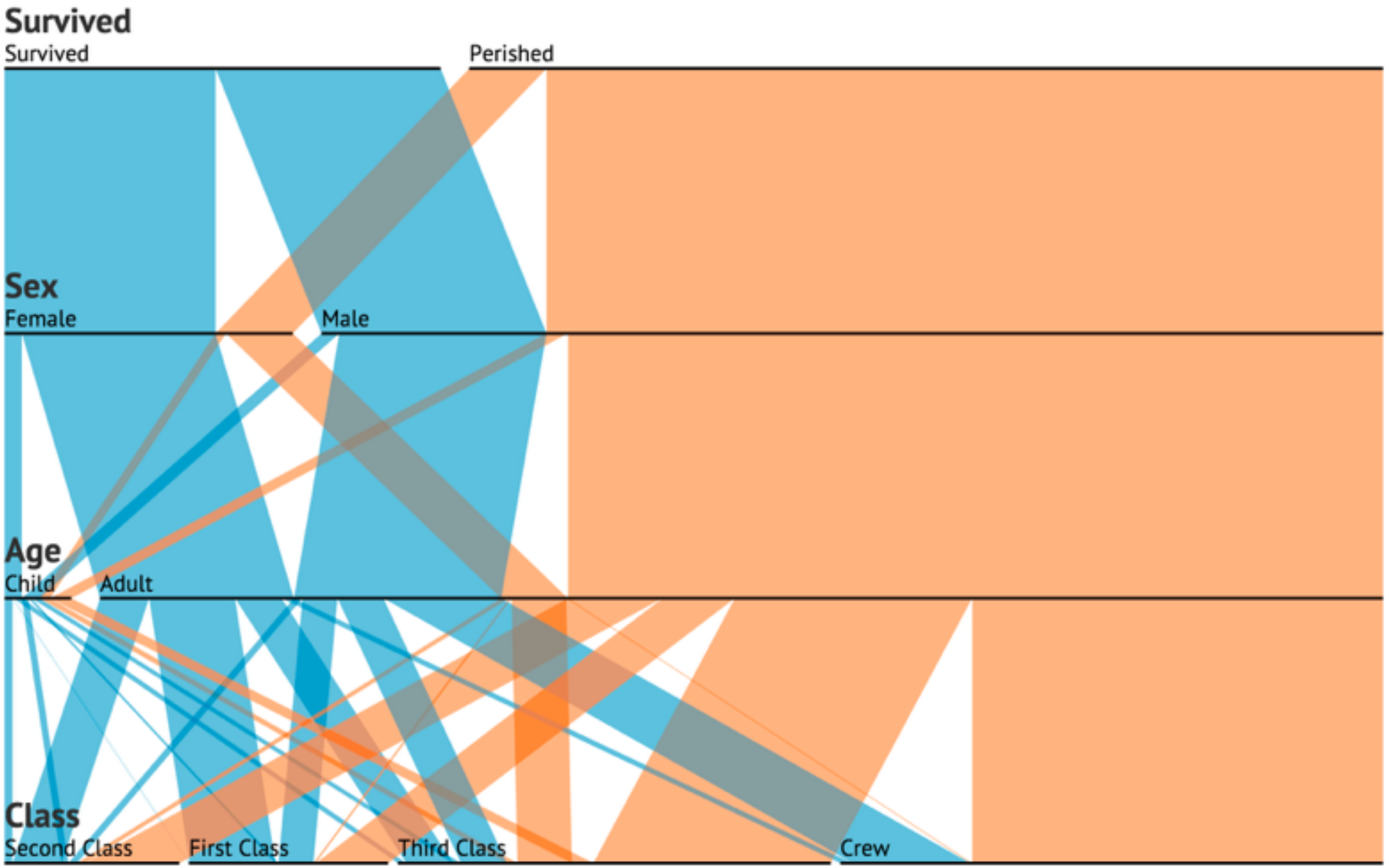
# Titanic Dataset

| survived | pclass | sex | age |
|---|---|---|---|
| 0 | 3 | male | 22 |
| 1 | 1 | female | 38 |
| 1 | 3 | female | 26 |
| 1 | 1 | female | 35 |
| 0 | 3 | male | 35 |
| 0 | 3 | male | |
| 0 | 1 | male | 54 |
| 0 | 3 | male | 2 |
| 1 | 3 | female | 27 |
| 1 | 2 | female | 14 |
| 1 | 3 | female | 4 |
| 1 | 1 | female | 58 |
| 0 | 3 | male | 20 |




# Mosaic plot

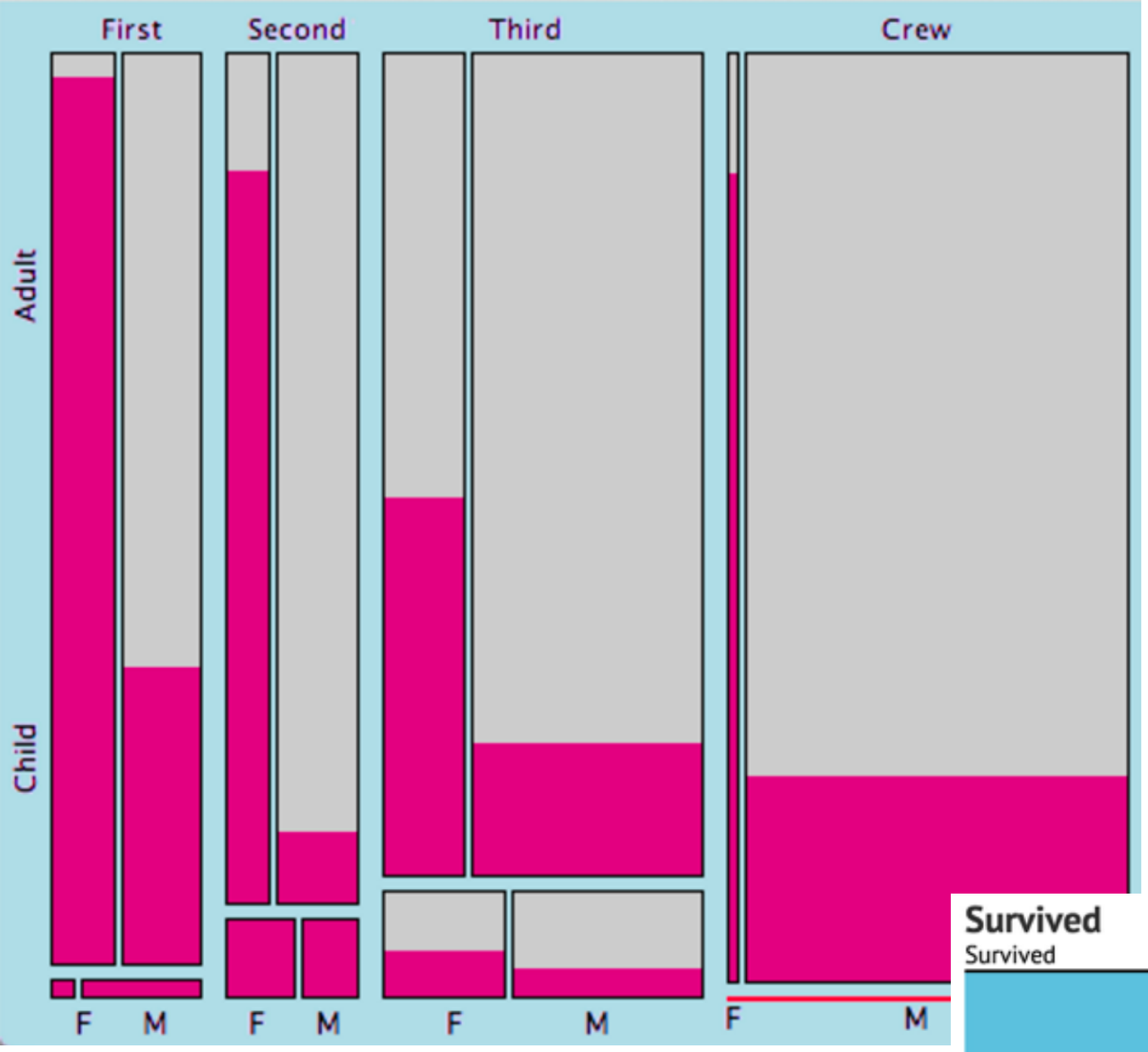
http://www.theusrus.de/blog/titanic-disaster-visualization-insights-100-years-later/



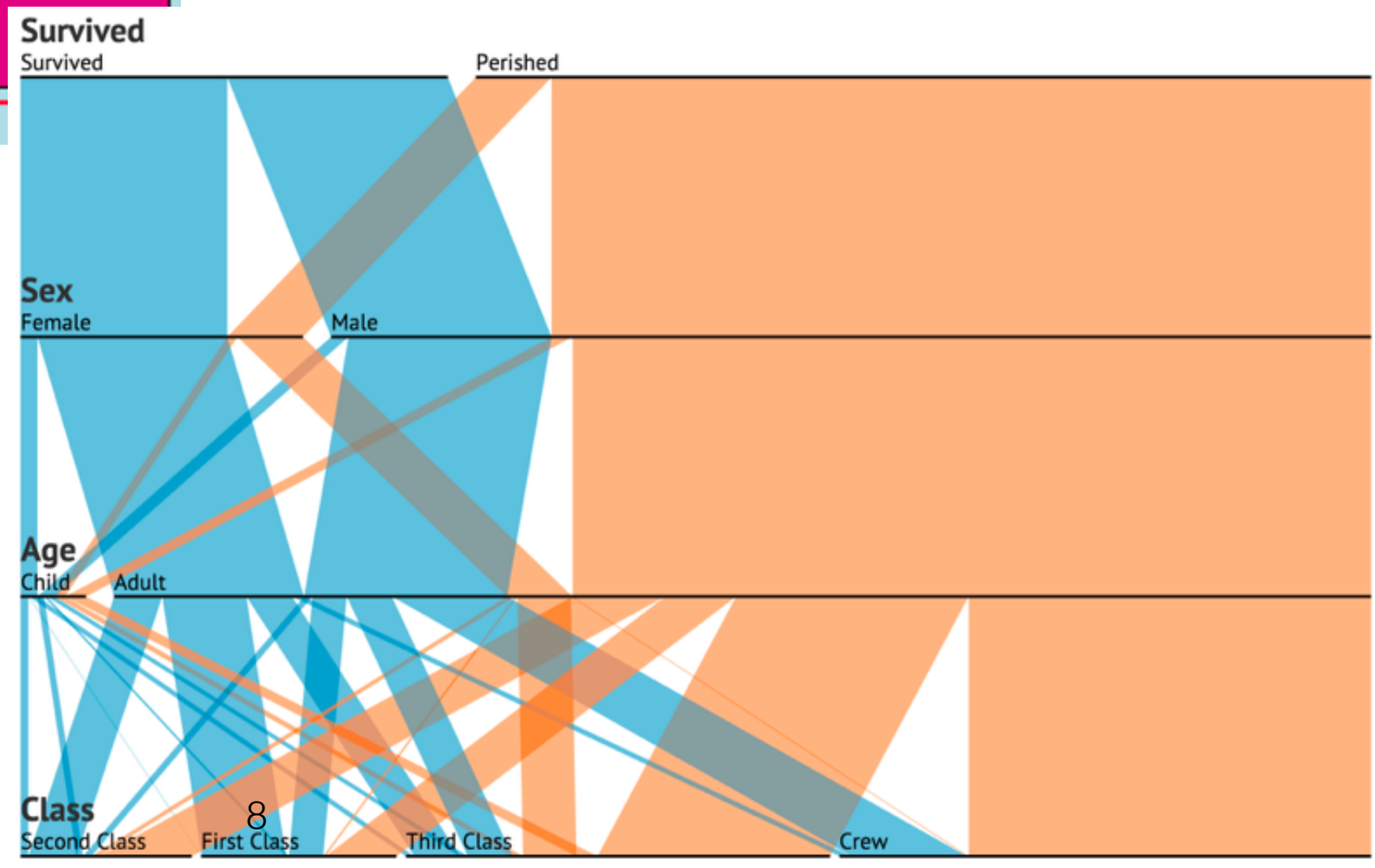

# Parallel set

https://www.jasondavies.com/parallel-sets/

http://www.theusrus.de/blog/titanic-disaster-visualization-insights-100-years-later/




# Mosaic plot



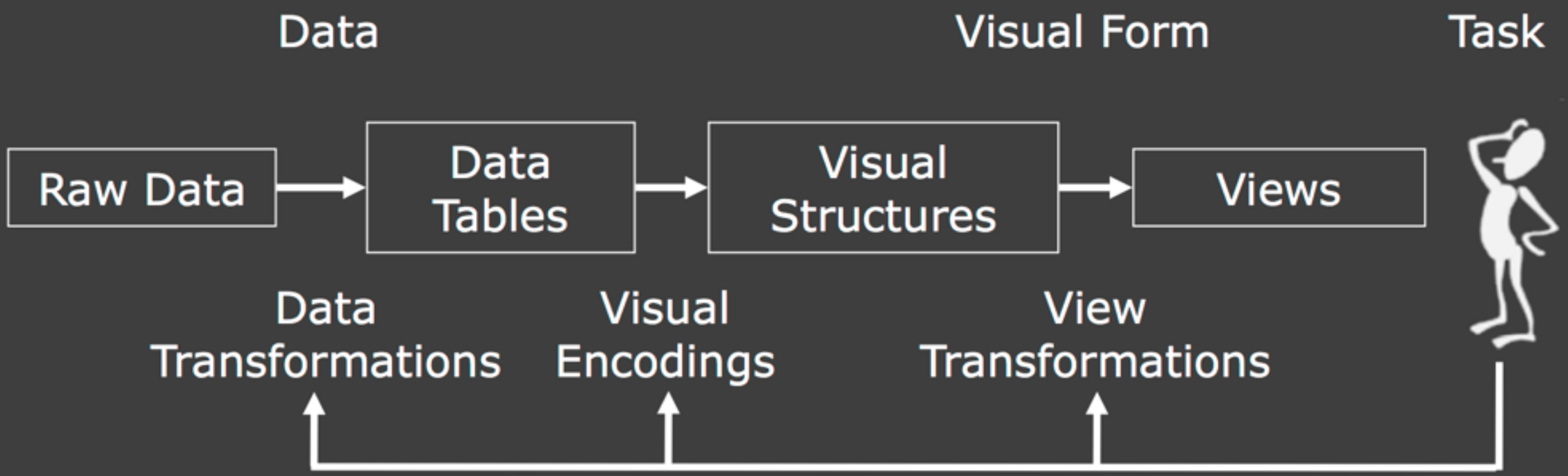

# Parallel set

https://www.jasondavies.com/parallel-sets/

# Visualizing the Titanic Dataset

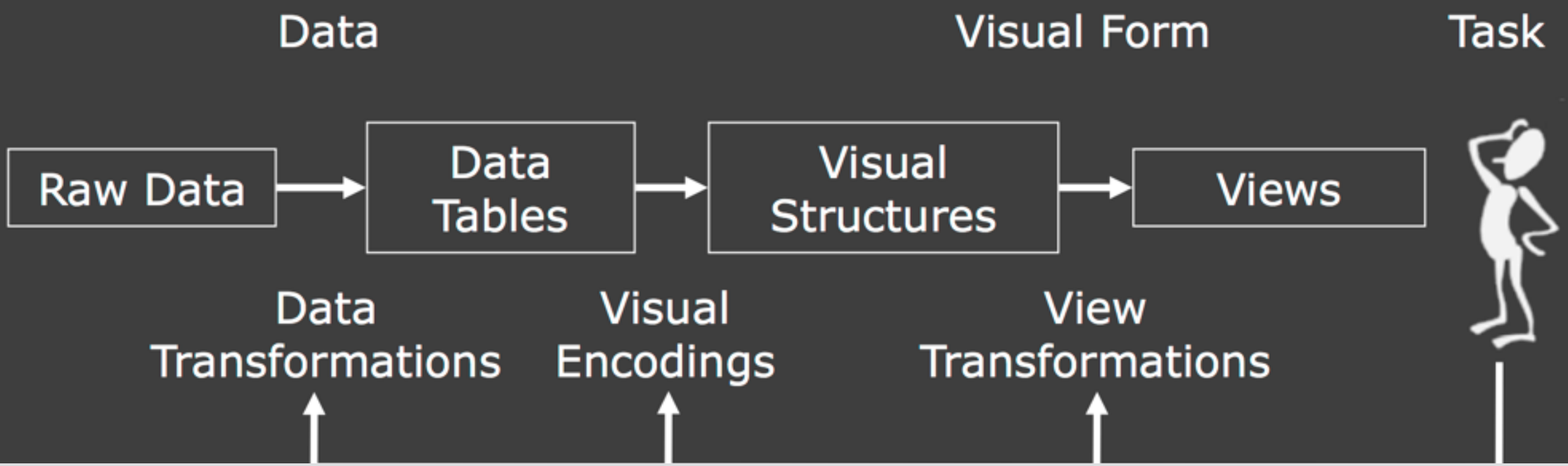
- Same data fields
- Many possible visual specifications
  - Color
  - Mark
  - Coordinate
- Lots of choices in data processing
  - Sum, avg
  - Merge dimensions

# Information Visualization Pipeline




[Redrawn Fig 1.23. Card, Mackinlay, and Shneiderman. Readings in Information Visualization: Using Vision To Think, Chapter 1. Morgan Kaufmann, 1999. Heer Lecture.]

# Information Visualization Pipeline




Visualization frameworks reduce the amount of work needed going through this process

…but in different ways

[Redrawn Fig 1.23. Card, Mackinlay, and Shneiderman. Readings in Information Visualization: Using Vision To Think, Chapter 1. Morgan Kaufmann, 1999. Heer Lecture.

# What Differentiates Visualization Frameworks?

- Support for data transformation?
- How to specify visual encodings?
- How much flexibility in view transformations?

# Popular visualization framework

Tableau, ggplot2

d3.js

Processing

Efficiency

Tableau, ggplot2

Pre-defined chart / geometry types

d3.js

Declarative data binding to visual primitives

Processing

Procedural creation of visual primitives

Expressiveness



[Adapted from Heer Lecture.]

Efficiency

Tableau, ggplot2

Pre-defined chart / geometry types

d3.js

Declarative data binding to visual primitives

Processing

Procedural creation of visual primitives

Expressiveness

# Processing

- Layer on top of Java / OpenGL
- Procedural creation of visual primitives

# Processing

- Layer on top of Java / OpenGL
- **Procedural creation of visual primitives**

# Processing: Hello World

```
background(255);
noStroke();

// Bright red
fill(255, 0, 0);
ellipse(20, 20, 16, 16);

// Dark red
fill(127, 0, 0);
ellipse(40, 20, 16, 16);

// Pink (pale red)
fill(255, 200, 200);
ellipse(60, 20, 16, 16);
```

http://learningprocessing.com/

# Processing: a Pie Chart

```
int[] angles = { 30, 10, 45, 35, 60, 38, 75, 67 };

void setup() {
  size(640, 360);
  noStroke();
  noLoop();  // Run once and stop
}

void draw() {
  background(100);
  pieChart(300, angles);
}

void pieChart(float diameter, int[] data) {
  float lastAngle = 0;
  for (int i = 0; i < data.length; i++) {
    float gray = map(i, 0, data.length, 0, 255);
    fill(gray);
    arc(width/2, height/2, diameter, diameter, lastAngle, lastAngle+radians(angles[i]));
    lastAngle += radians(angles[i]);
  }
}
```

https://processing.org/examples/piechart.html

# Processing: a Pie Chart

```
int[] angles = { 30, 10, 45, 35, 60, 38, 75, 67 };

void setup() {
  size(640, 360);
  noStroke();
  noLoop();  // Run once and stop
}

void draw() {
  b
  p
}

voi
  float lastAngle = 0;
  for (int i = 0; i < data.length; i++) {
    float gray = map(i, 0, data.length, 0, 255);
    fill(gray);
    arc(width/2, height/2, diameter, diameter, lastAngle, lastAngle+radians(angles[i]));
    lastAngle += radians(angles[i]);
  }
}
```

Need to specify the position and color of each slice explicitly

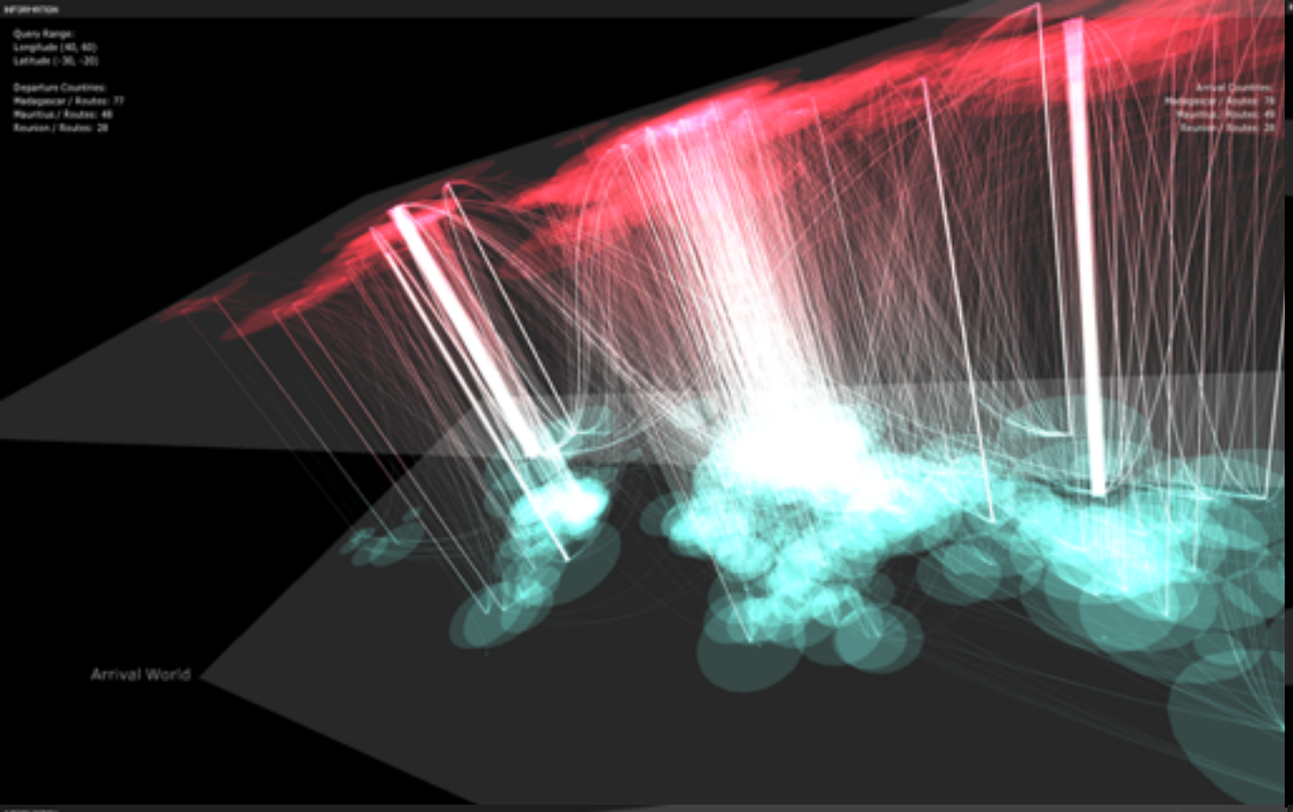
No built-in notion of “pie chart”

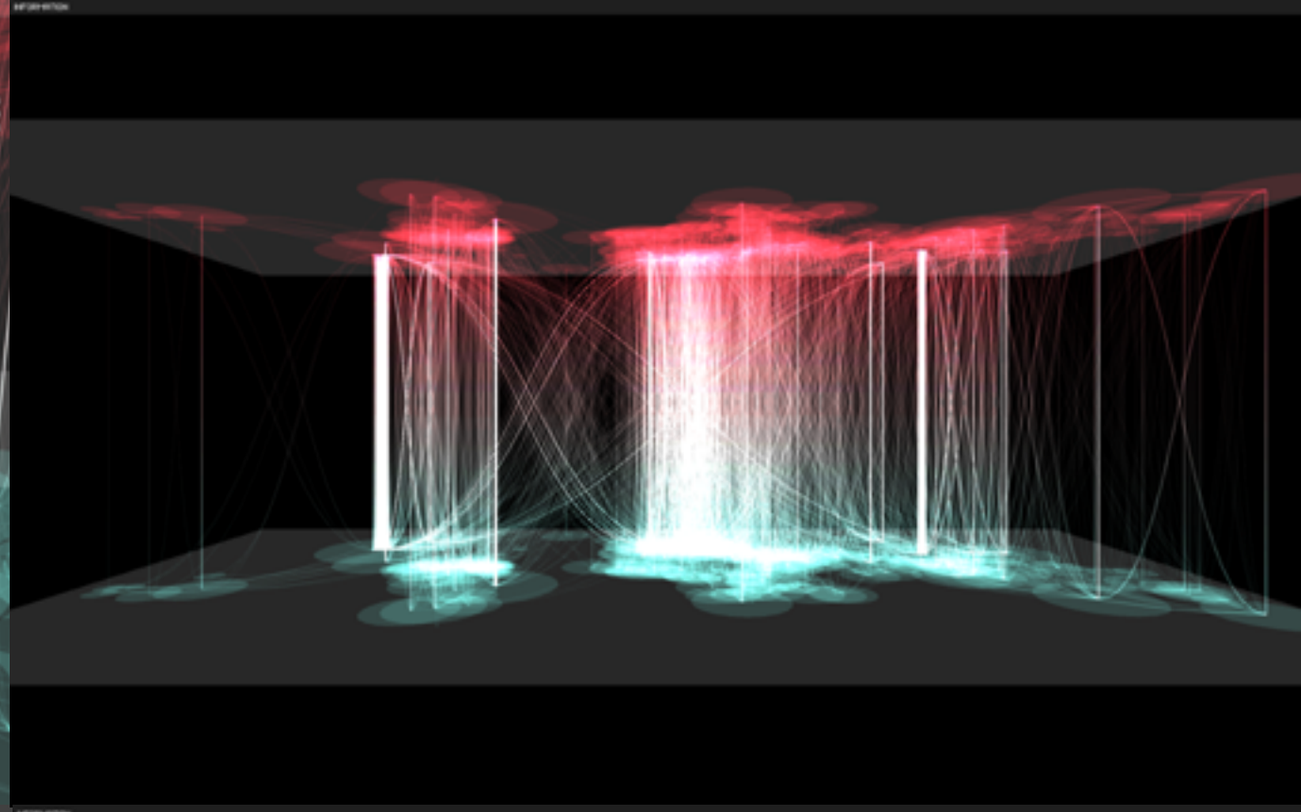
https://processing.org/examples/piechart.html

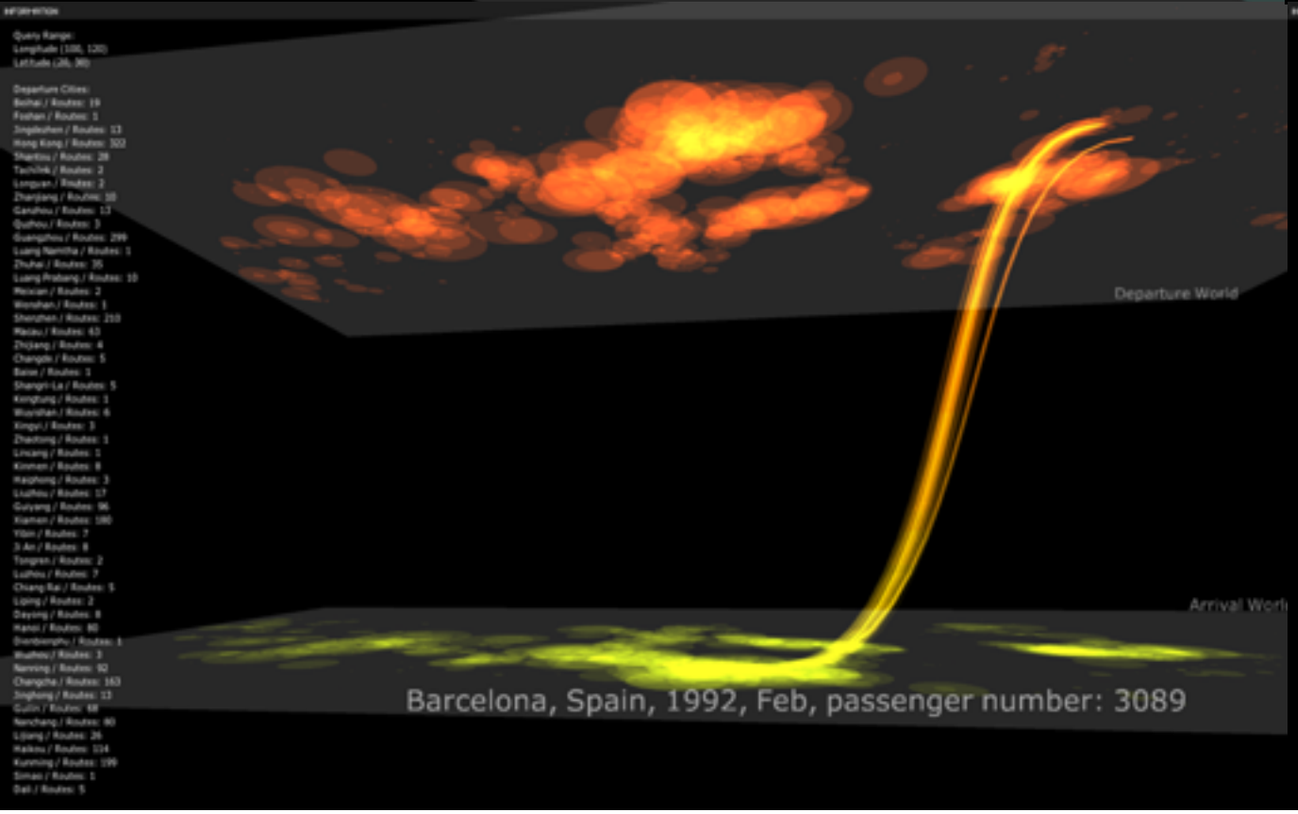
US Flight Patterns. Aaron Koblin. http://www.aaronkoblin.com/work/flightpatterns/interactiveMap.html

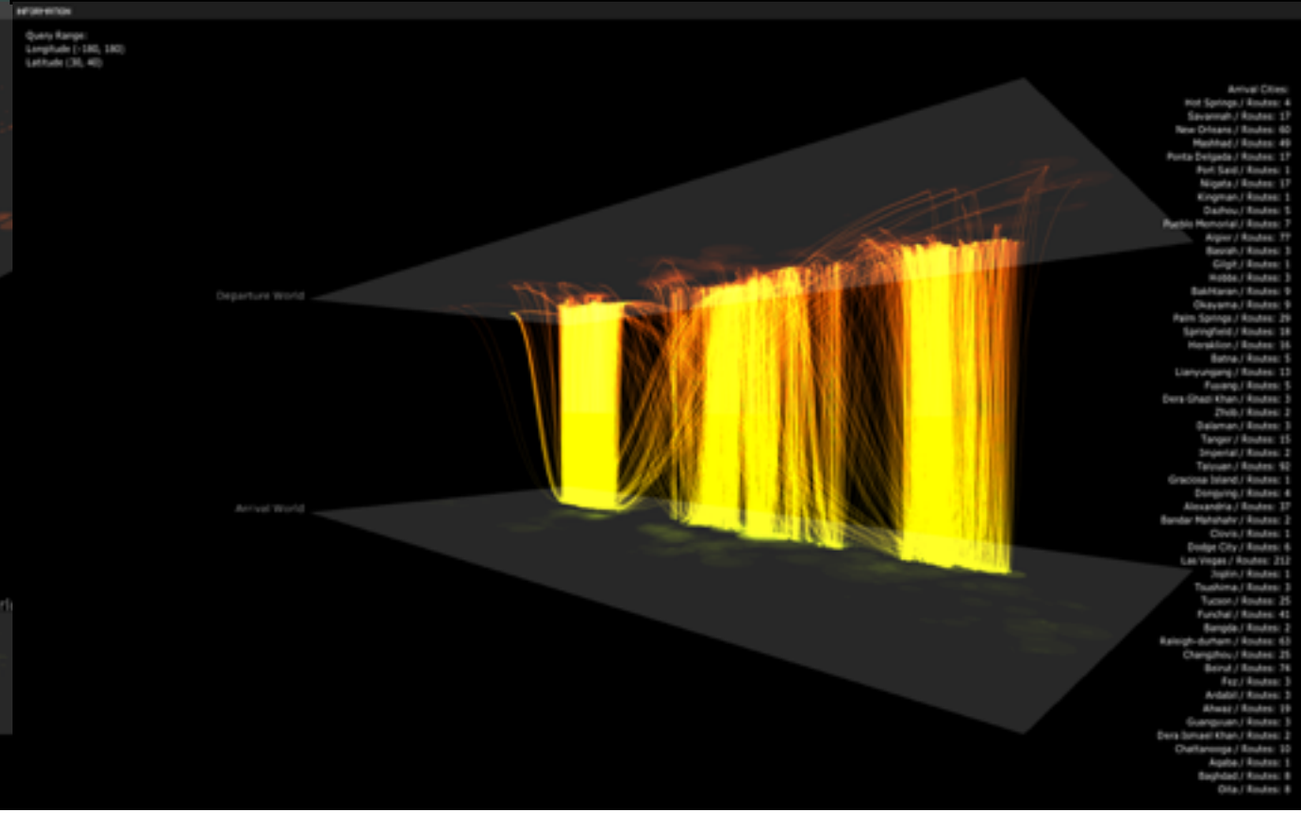
# Advanced Processing

- interactivity
- 3D
- lights, camera
- textures
- shaders
- music and video






Global Flight Pattern. Ian Liu and Yun Teng. http://www.mat.ucsb.edu/qian/index.php/project/take-flight/

# Processing

- Pros
  - Highly flexible; good for storytelling
  - Huge user community, many of which are artists / designers
  - Available in multiple languages (p5.js, Processing.py)
- Cons
  - No built-in visualization layouts, color palettes etc (but external libraries are available, e.g. giCentre Utilities)

Efficiency

Tableau, ggplot2

Pre-defined chart / geometry types

d3.js

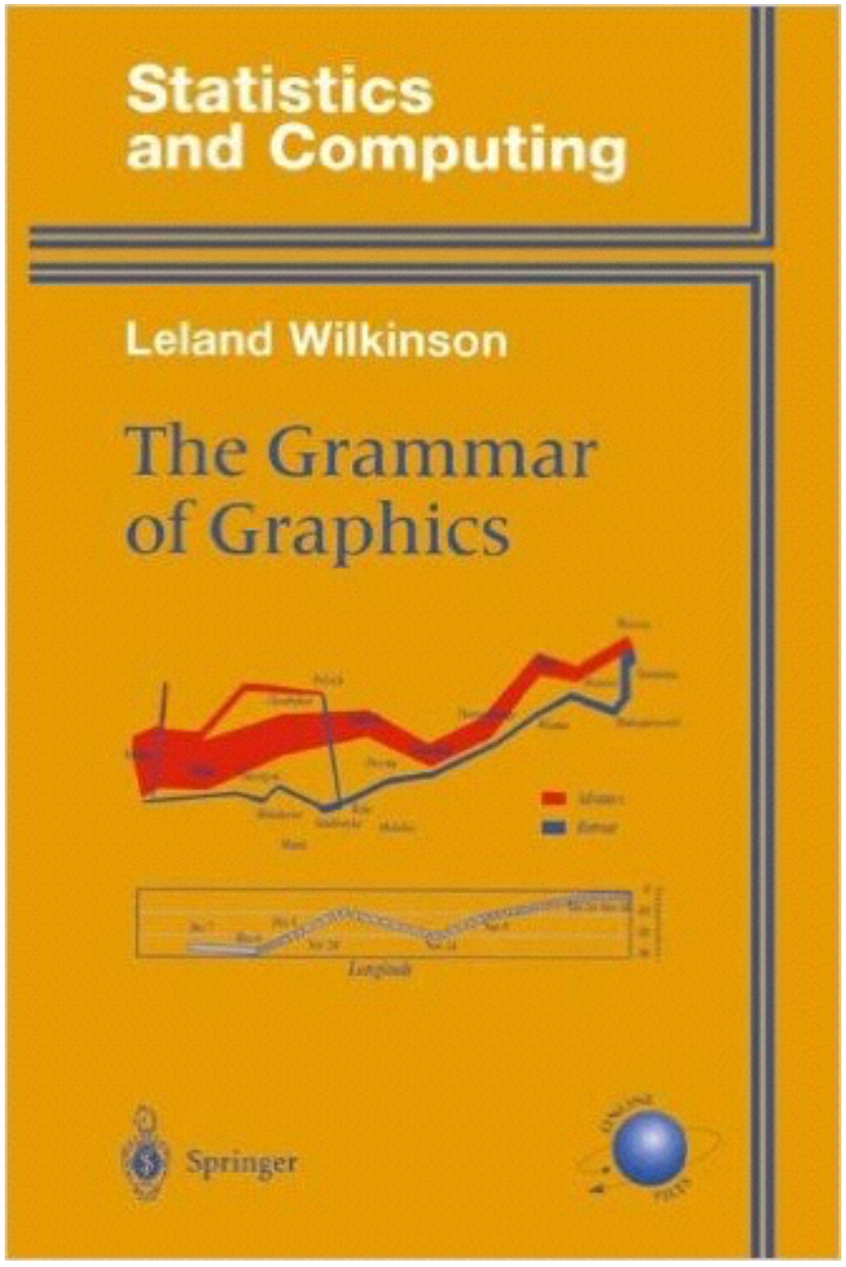
Declarative data binding to visual primitives

Processing

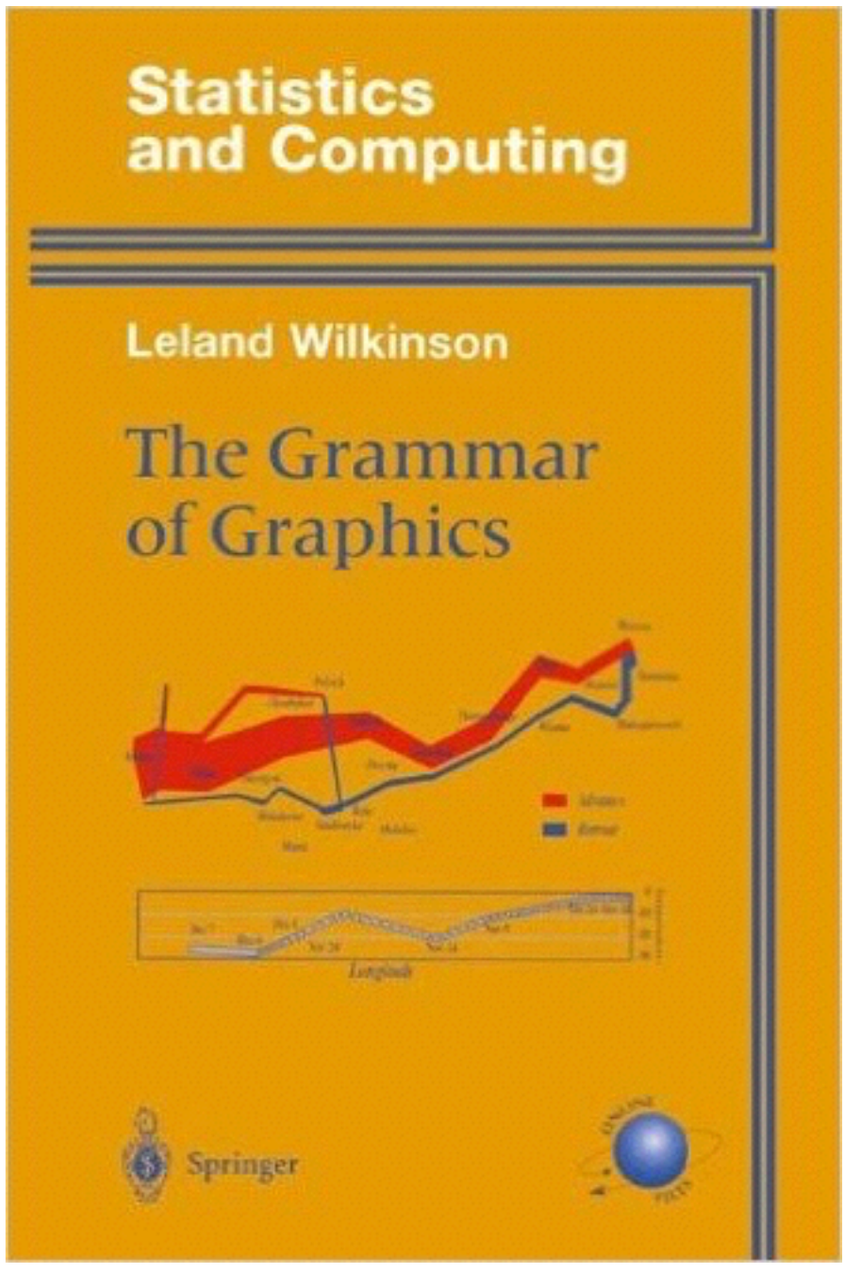
Procedural creation of visual primitives

Expressiveness

# Grammar of Graphics



# Grammar of Graphics




Visualization ⟷ Layers

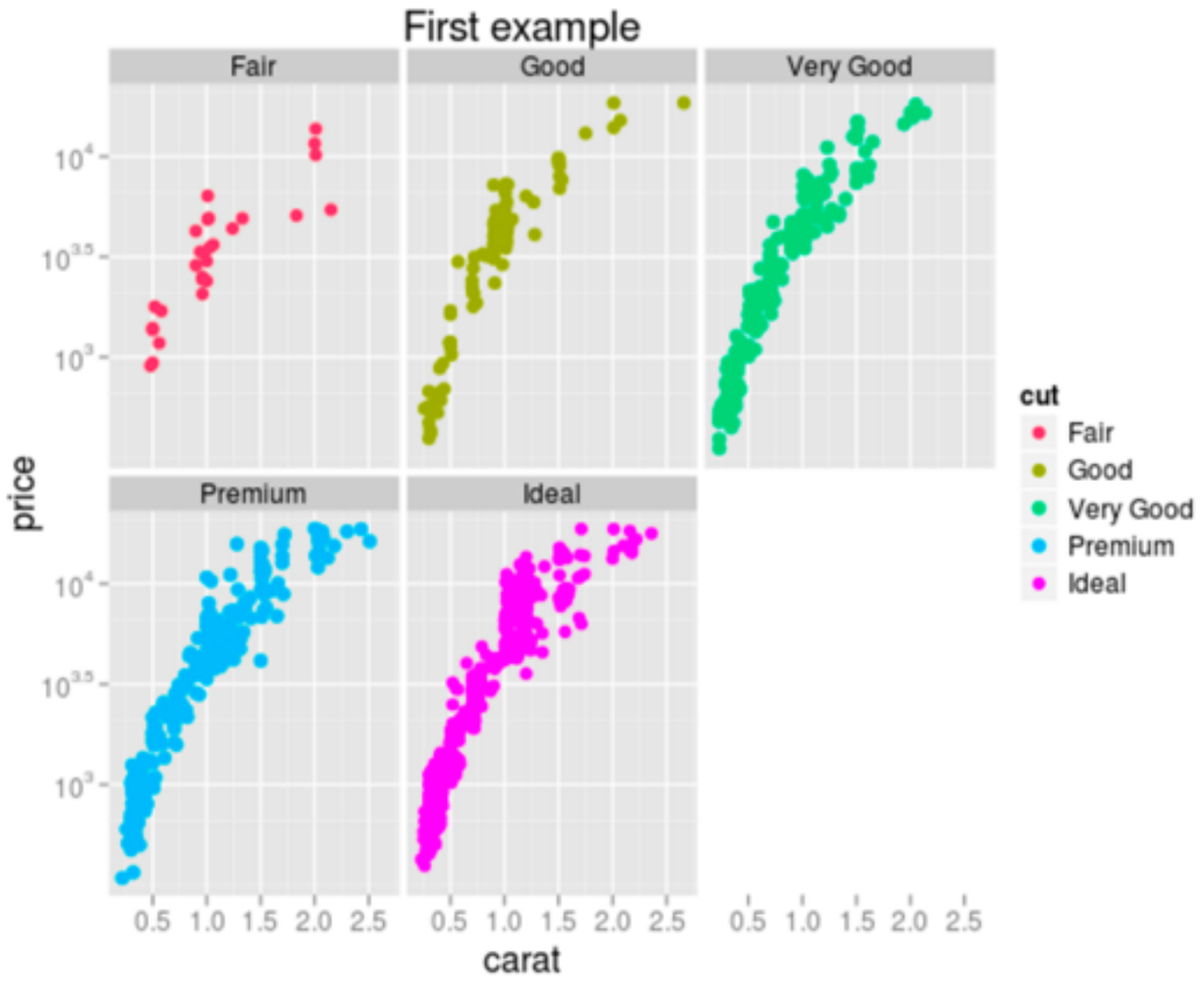
↕ ↕

Sentence ⟷ Parts of speech

# Grammar of Graphics: six layers

| | |
|---|---|
| **Data** | Input data to visualize |
| **Operators** | Grouping, statistics, layout |
| **Coordinates** | Cartesian & polar coordinates |
| **Scales** | Map data values to visual values |
| **Guides** | Axes & legends visualize scales |
| **Marks** | Geometric primitives |

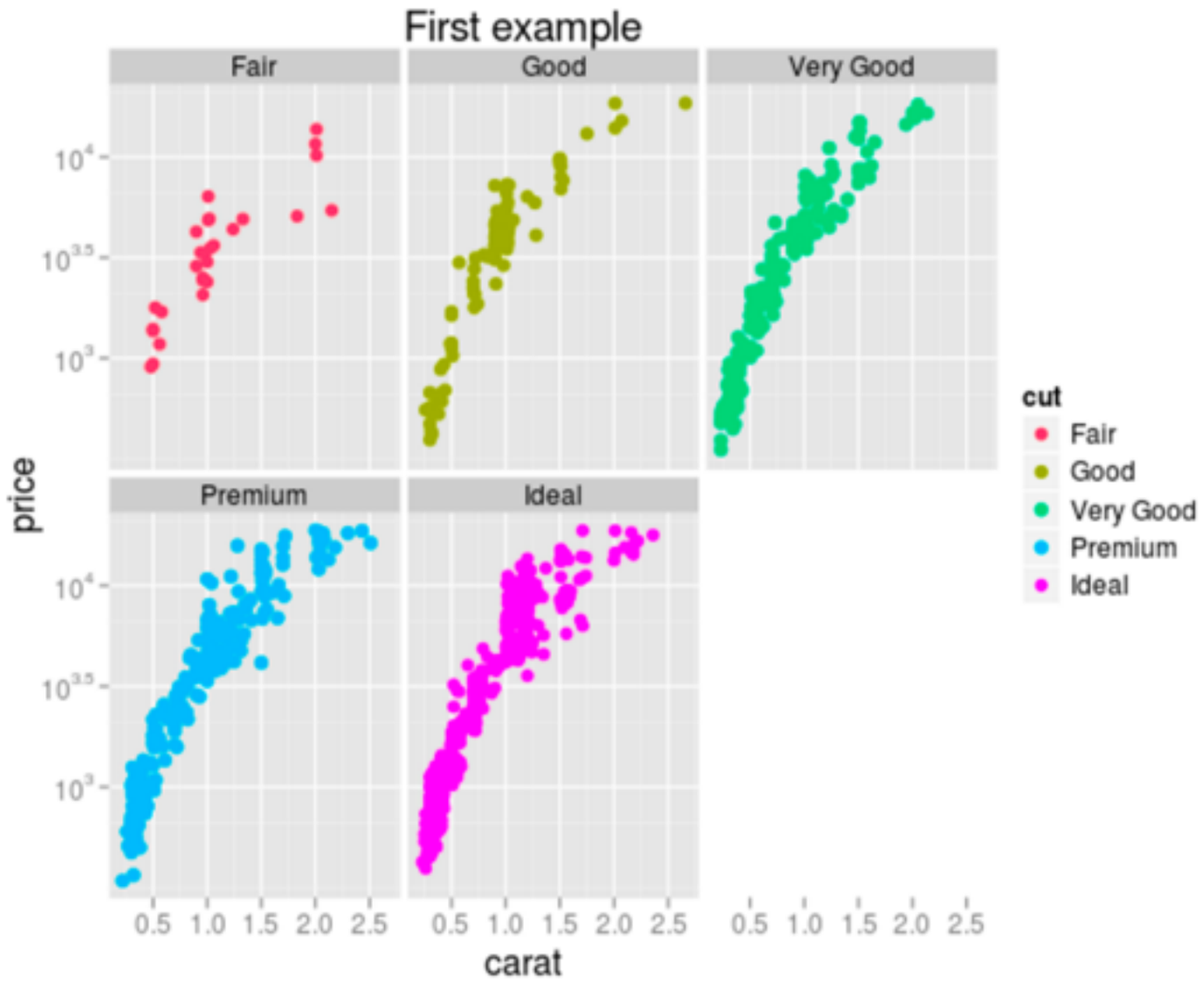
# ggplot2 (gg -> “Grammar of Graphics”)




```
ggplot(small)
        +geom_point(aes(x=carat,y=price,colour=cut))
                +scale_y_log10()
                        +facet_wrap(~cut)
                                +ggtitle("First example")
```

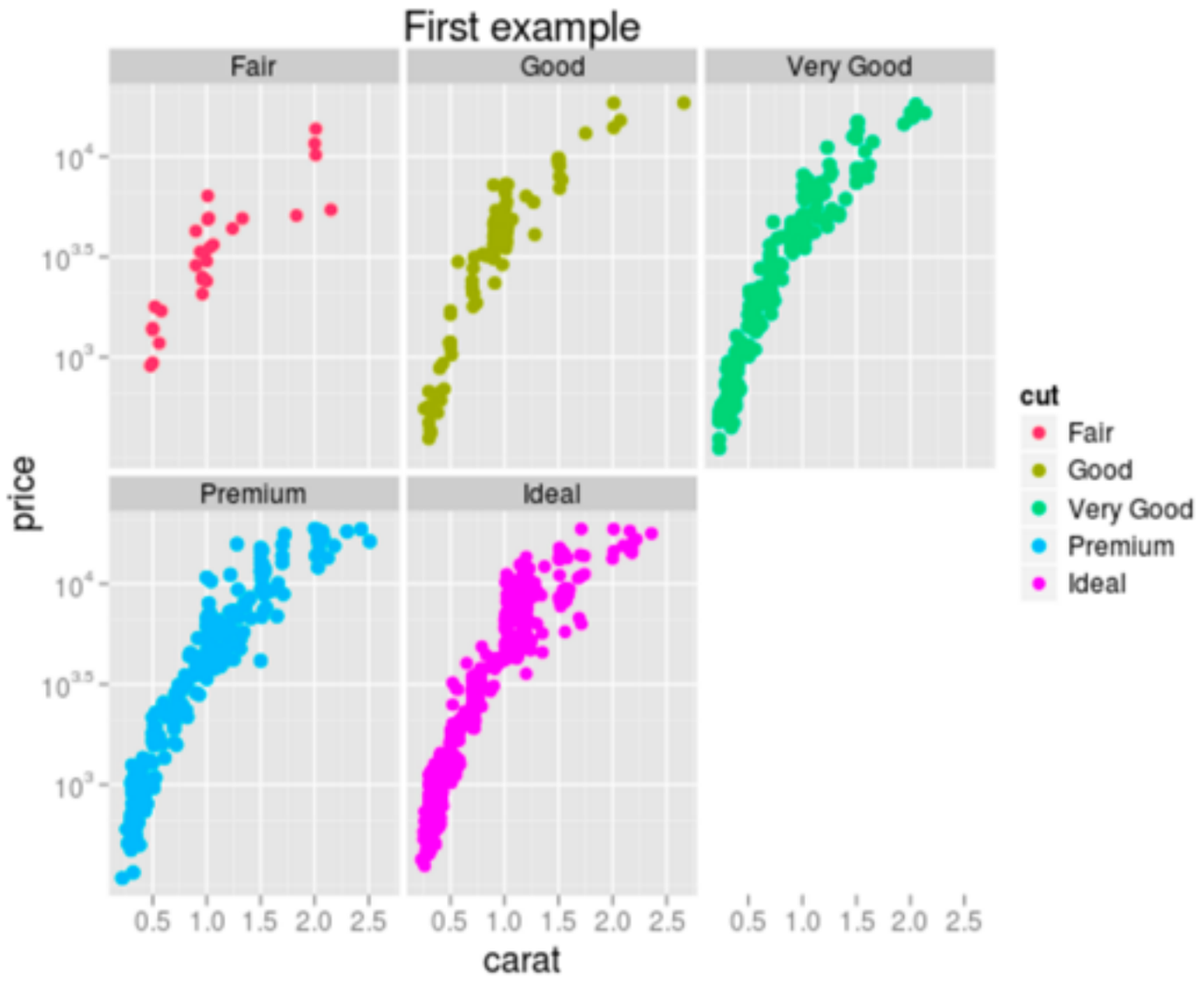
http://chrisladroue.com/extra/ggplot2Intro/introductionGGplot2/

# ggplot2 (gg -> “Grammar of Graphics”)



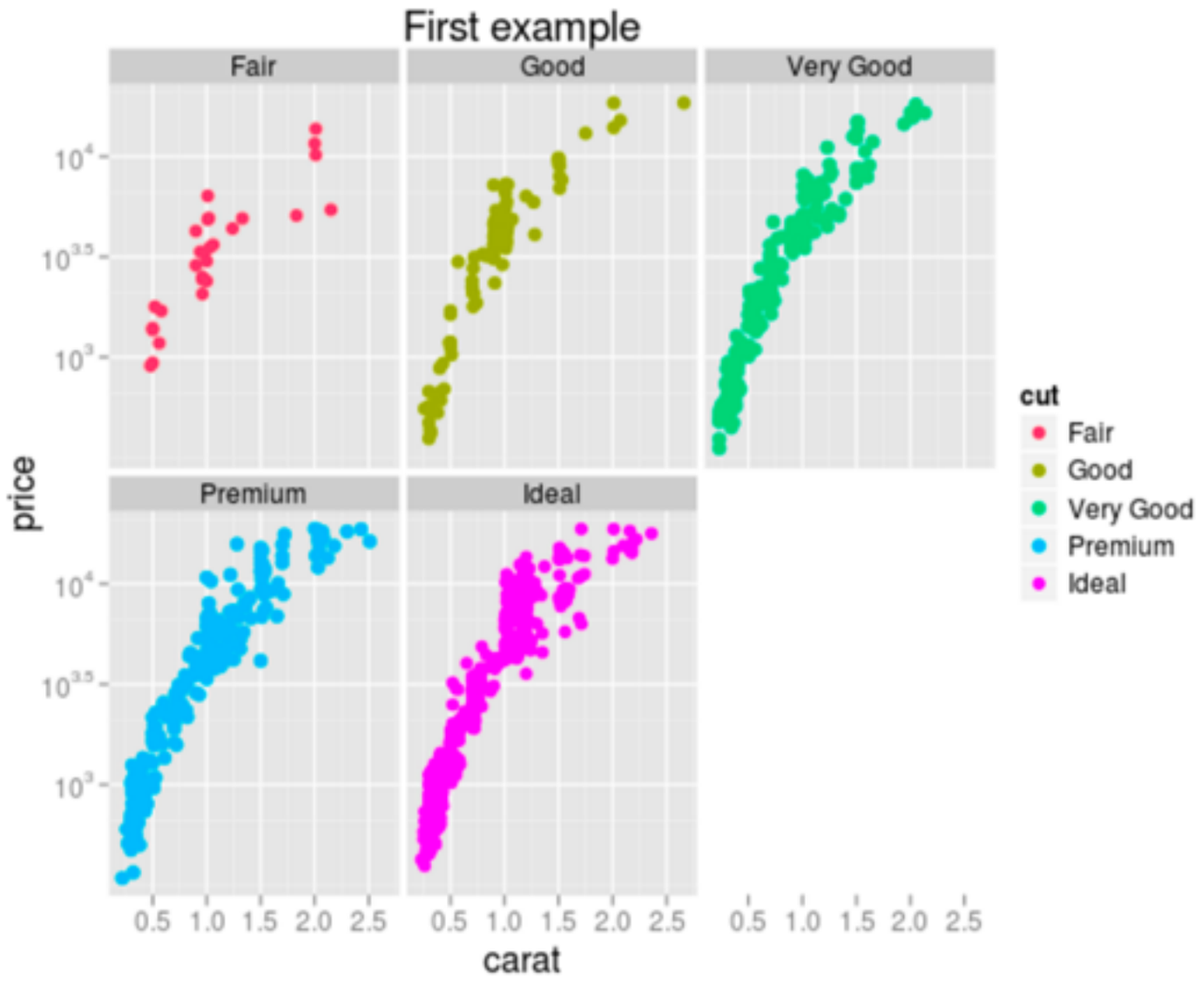

```
ggplot(small) data
        +geom_point(aes(x=carat,y=price,colour=cut))
              +scale_y_log10()
                    +facet_wrap(~cut)
                          +ggtitle("First example")
```

# ggplot2 (gg -> “Grammar of Graphics”)




```
ggplot(small)  data
        +geom_point(aes(x=carat,y=price,colour=cut))  mark
                +scale_y_log10()
                        +facet_wrap(~cut)
                                +ggtitle("First example")
```

# ggplot2 (gg -> “Grammar of Graphics”)



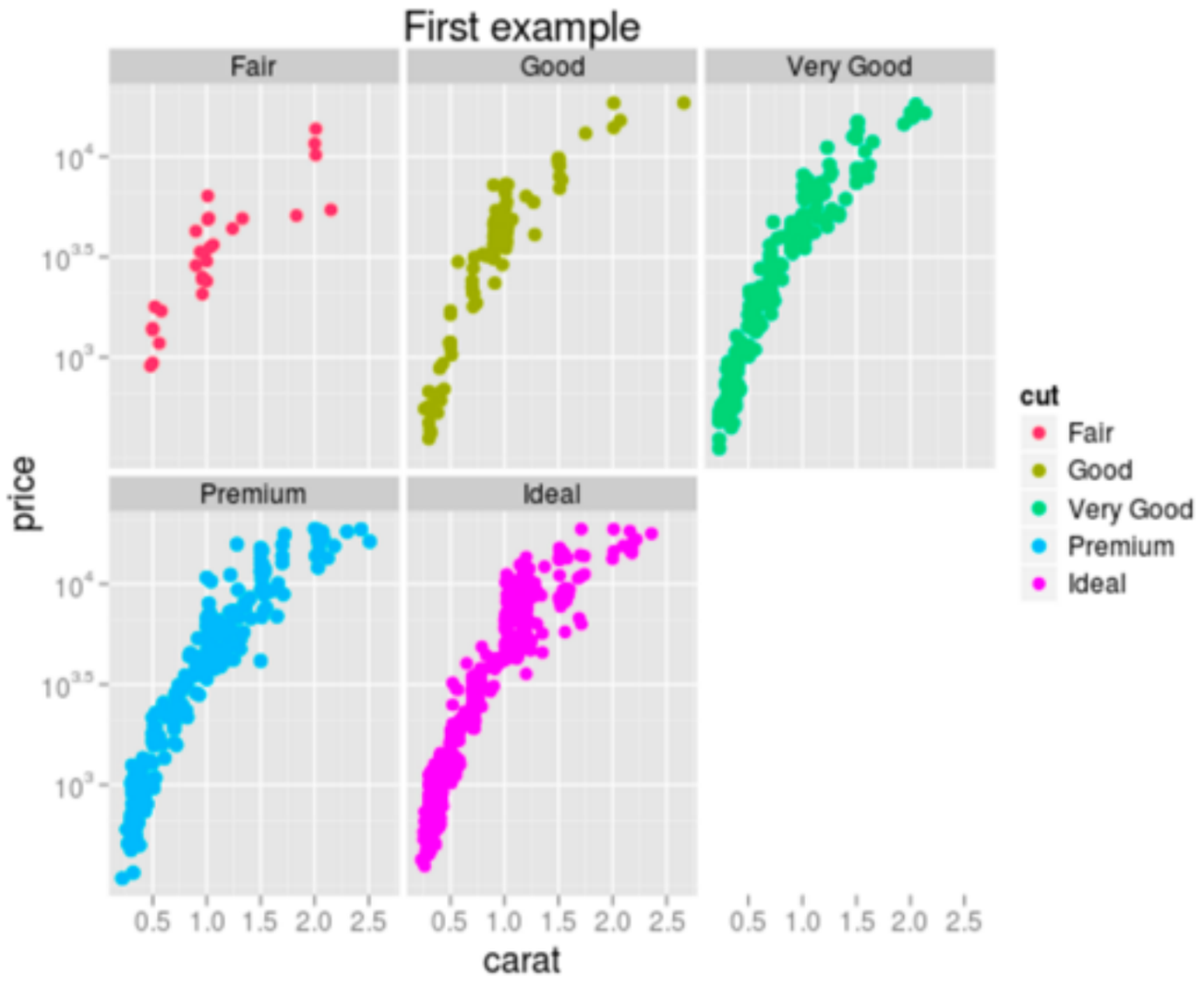

```
ggplot(small)
    +geom_point(aes(x=carat,y=price,colour=cut))
        +scale_y_log10()
            +facet_wrap(~cut)
                +ggtitle("First example")
```

data

mark

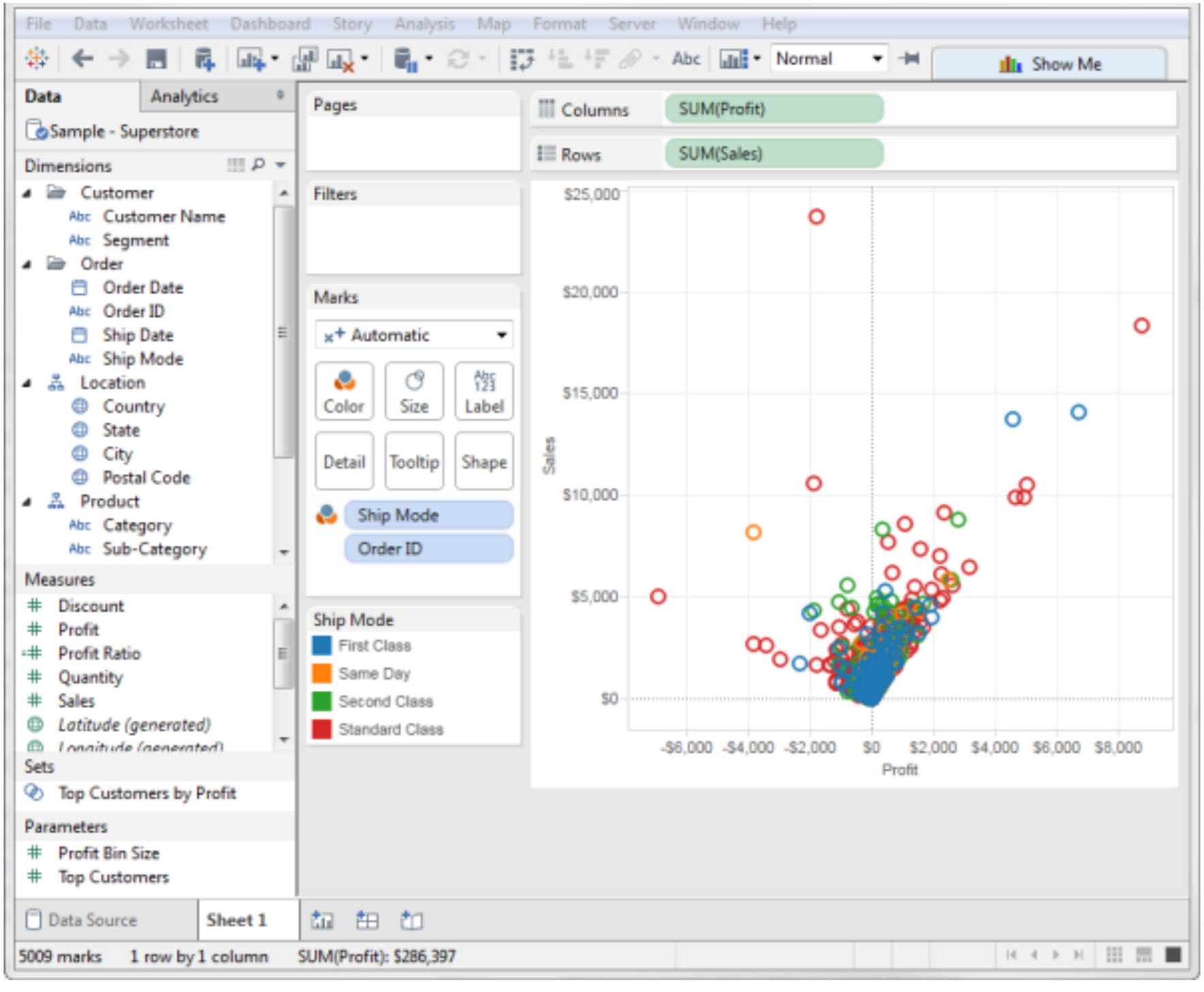
scale

# ggplot2 (gg -> “Grammar of Graphics”)




```
ggplot(small)                                    data
    +geom_point(aes(x=carat,y=price,colour=cut))  mark
        +scale_y_log10()                         scale
            +facet_wrap(~cut)                    operator (grouping)
                +ggtitle("First example")
```

# Tableau


File Data Worksheet Dashboard Story Analysis Map Format Server Window Help
Normal
Show Me
Data
Analytics
Sample - Superstore
Dimensions
Customer
Customer Name
Segment
Order
Order Date
Order ID
Ship Date
Ship Mode
Location
Country
State
City
Postal Code
Product
Category
Sub-Category
Measures
Discount
Profit
Profit Ratio
Quantity
Sales
Latitude (generated)
Sets
Top Customers by Profit
Parameters
Profit Bin Size
Top Customers
Pages
Filters
Marks
Automatic
Color
Size
Label
Detail
Tooltip
Shape
Ship Mode
Order ID
Ship Mode
First Class
Same Day
Second Class
Standard Class
Columns
SUM(Profit)
Rows
SUM(Sales)
$25,000
$20,000
$15,000
$10,000
$5,000
$0
Sales
-$6,000 -$4,000 -$2,000 $0 $2,000 $4,000 $6,000 $8,000
Profit
Data Source
Sheet 1
5009 marks
1 row by 1 column
SUM(Profit): $286,397

# Polaris / Tableau

- Polaris: research at Stanford by Stolte, Tang, and Hanrahan, and later evolved into Tableau
- Simultaneously specify both database queries and visualization
  - Tailored for tabular data
- Data first, visualization follows
  - Drag & drop to specify data dimensions
  - Smart defaults / recommendations for graph types
  - Automate visualization design

# Polaris: Visualization Described Using Visual Specifications

- Table configuration for visualization (algebra)
  - Operands: the database fields (ordinal and quantitative fields are treated differently)
  - Operators: concatenation (+), cross product (x), nest (/)
- Use table configuration to infer appropriate visual specification
  - Type of graphic in each pane
  - Encoding of data as visual properties (color, size, shape, …) of marks
  - Data transformations and queries



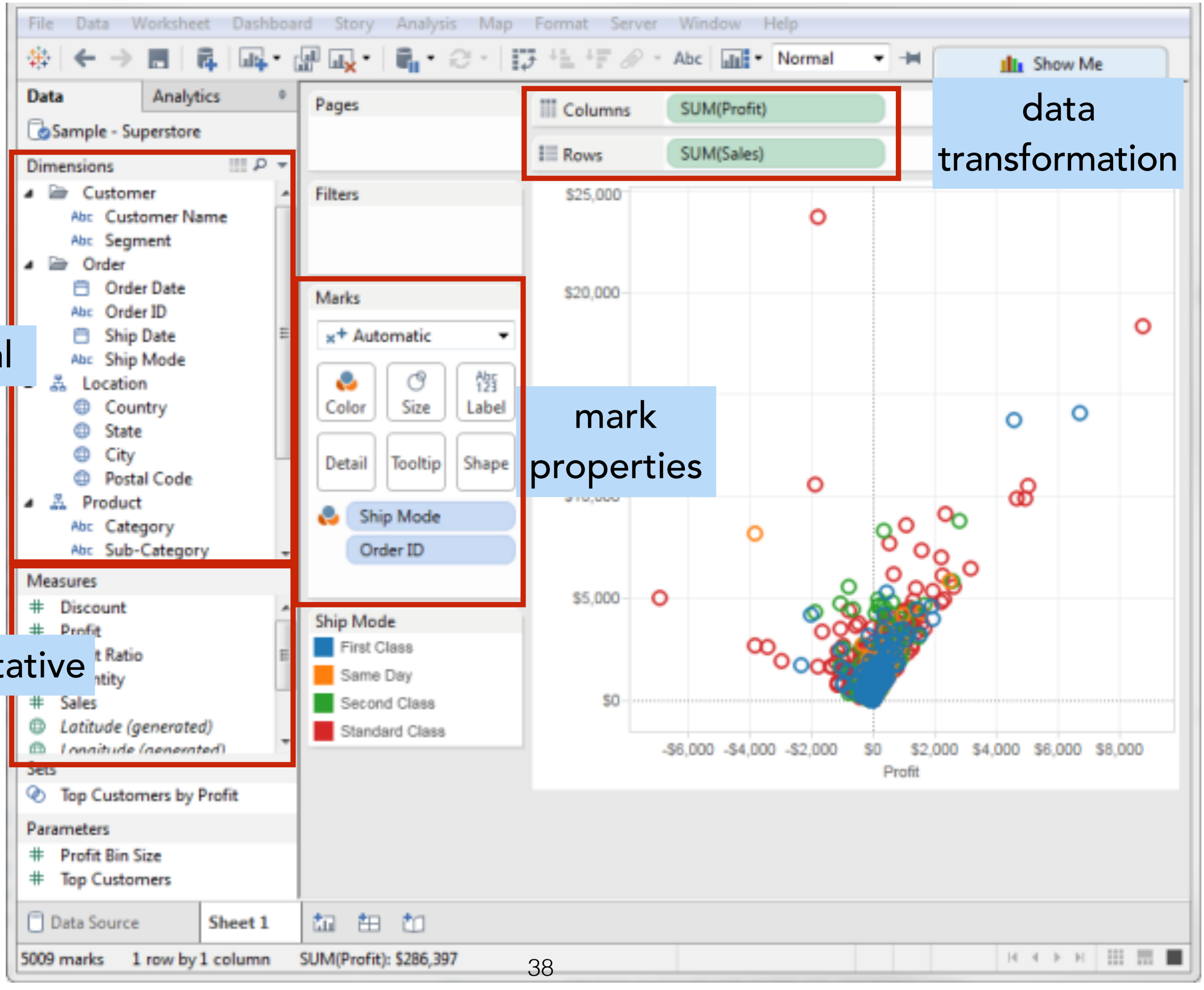
[Adapted from Heer Lecture.]

ordinal

quantitative

data transformation

mark properties



# Table Algebra: Operands

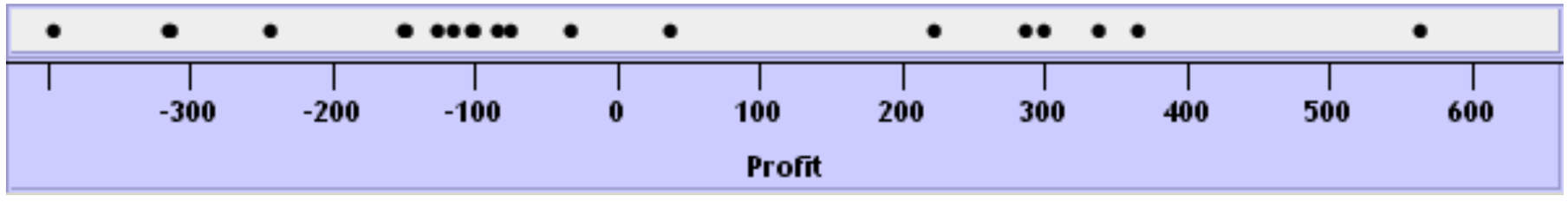
- Ordinal fields: interpret domain as a set that partitions table into rows and columns:

  Quarter = {(Qtr1),(Qtr2),(Qtr3),(Qtr4)}

| Qtr1 | Qtr2 | Qtr3 | Qtr4 |
|---|---|---|---|
| 95892 | 101760 | 105282 | 98225 |

- Quantitative fields: treat domain as single element set and encode spatially as axes:

  Profit = {(Profit[-410,650])}




Stolte. Polaris: Query, Analysis, and Visualization of Large Hierarchical Relational Databases

# Table Algebra: Concatenation Operator

- Ordered union of two sets

  Quarter + ProductType

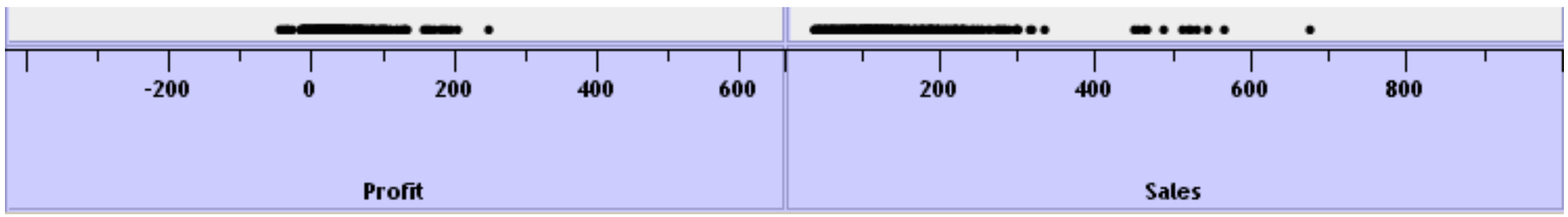
  = {(Qtr1),(Qtr2),(Qtr3),(Qtr4)}+{(Coffee),(Espresso)}

  = {(Qtr1),(Qtr2),(Qtr3),(Qtr4),(Coffee),(Espresso)}

| Qtr1 | Qtr2 | Qtr3 | Qtr4 | Coffee | Espresso |
|---|---|---|---|---|---|
| 48 | 59 | 57 | 53 | 151 | 21 |

Profit + Sales

= {(Profit[-310,620]),(Sales[0,1000])}




Stolte. Polaris: Query, Analysis, and Visualization of Large Hierarchical Relational Databases

# Table Algebra: Cross Operator

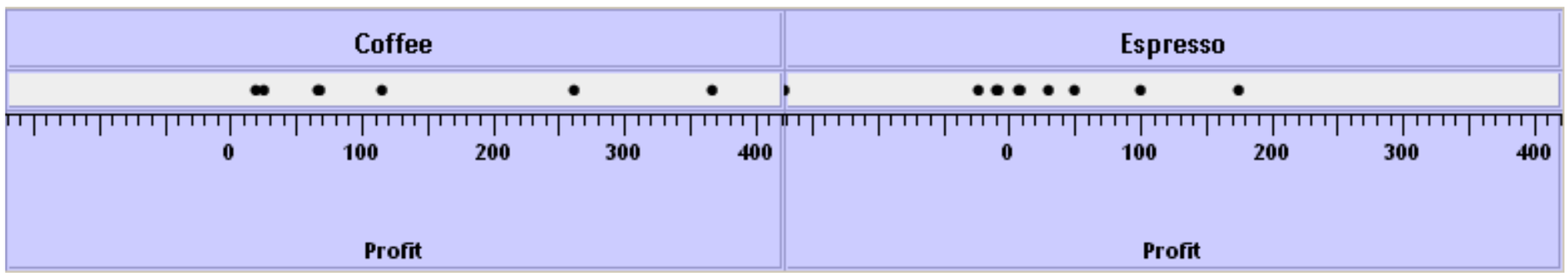
- Direct-product of two sets

Quarter x ProductType =

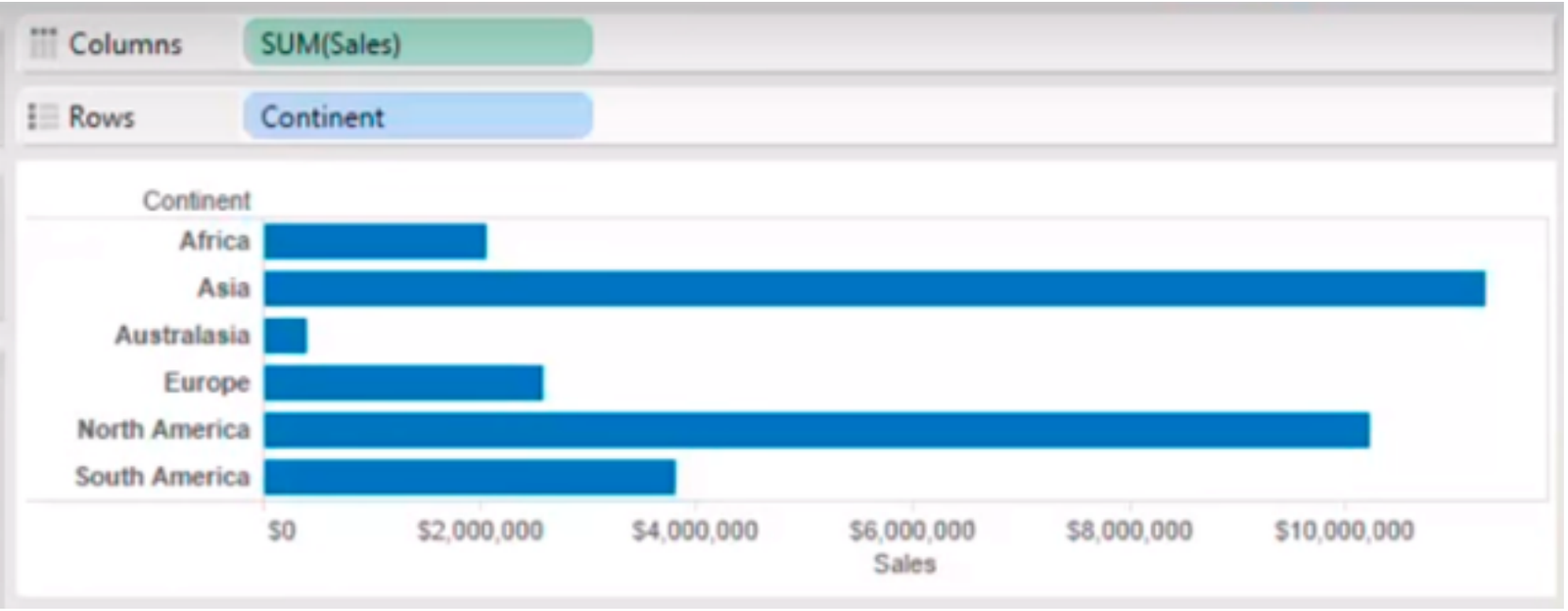
{(Qtr1,Coffee), (Qtr1, Tea), (Qtr2, Coffee), (Qtr2, Tea),
(Qtr3, Coffee), (Qtr3, Tea), (Qtr4, Coffee), (Qtr4,Tea)}

| Qtr1 | | Qtr2 | | Qtr3 | | Qtr4 | |
|---|---|---|---|---|---|---|---|
| Coffee | Espresso | Coffee | Espresso | Coffee | Espresso | Coffee | Espresso |
| 131 | 19 | 160 | 20 | 178 | 12 | 134 | 33 |

ProductType x Profit =




Stolte. Polaris: Query, Analysis, and Visualization of Large Hierarchical Relational Databases



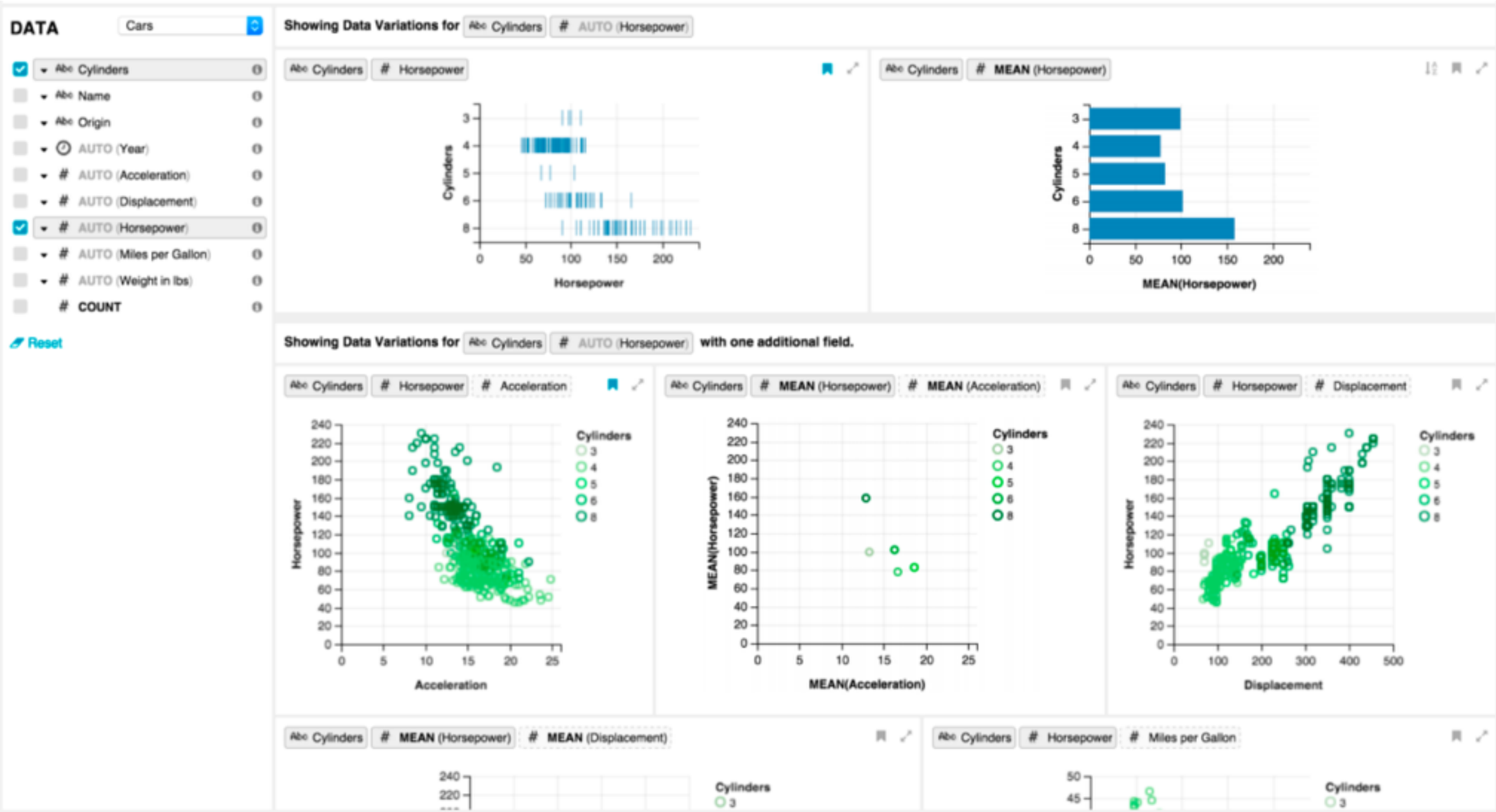

Columns: Quantitative

Rows: Ordinal


Columns: Quantitative

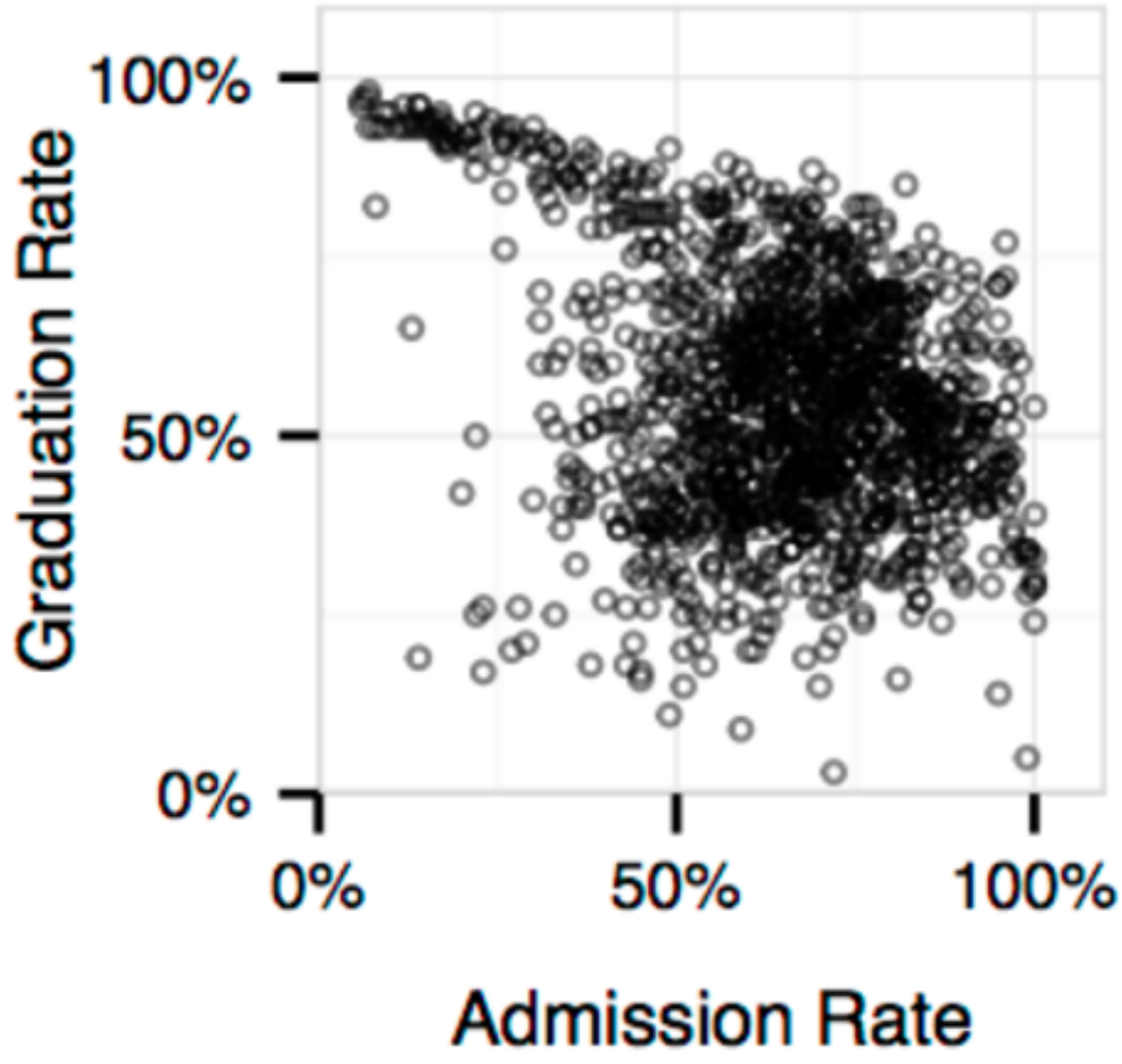
Rows: Ordinal x Ordinal

# Recommendation of data fields to explore



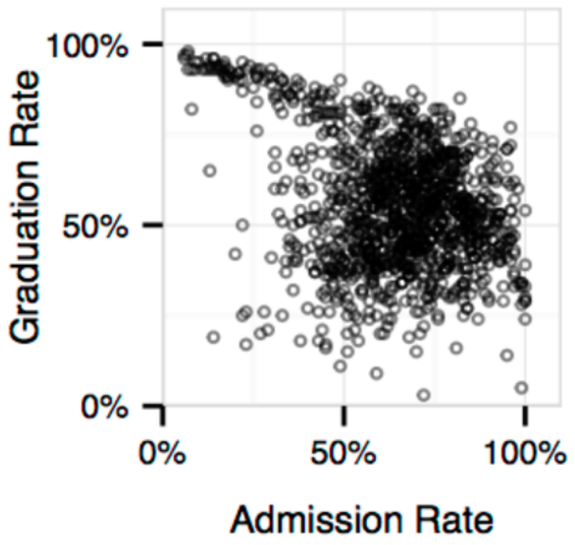

[Fig 1. Wongsuphasawat et al. Voyager: Exploratory Analysis via Faceted Browsing of Visualization Recommendations. ]

# Automatic data partition

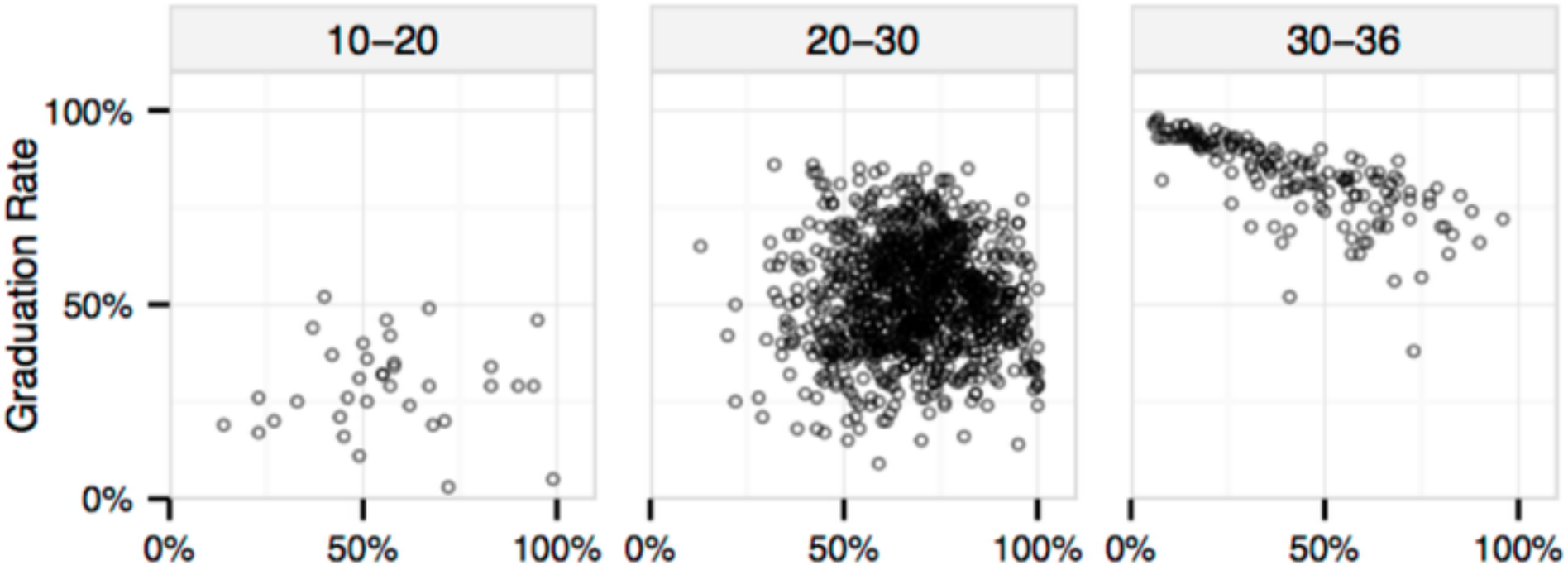



[Fig 1. Anand and Talbot. Automatic Selection of Partitioning Variables for Small Multiple Displays.]

# Automatic data partition




Algorithmically ranking partitioning variables (e.g. ACT score) based on quality measures for visual patterns




Partition by ACT scores

# A D3 Primer

# Data-Driven Documents

# Overview of D3

- Research at Stanford by Jeff Heer and Mike Bostock
- Declarative toolkit in Javascript

# Overview of D3

- Research at Stanford by Jeff Heer and Mike Bostock
- Declarative toolkit in Javascript
- Visualization via binding data to DOM (usually SVG elements)
  - DOM (Document Object Model): HTML nodes (div, h1, button etc) organized in a tree structure
  - SVG: XML-based vector image format (visual primitives on the web)

# Data Binding: an Example

[1, 1.2, 1.7, 1.5, 0.7] ⟶ bar chart

# Data Binding: an Example

[1, 1.2, 1.7, 1.5, 0.7] ⟶ bar chart

Need to compute:

- bar height
- bar position

# Data Binding: an Example

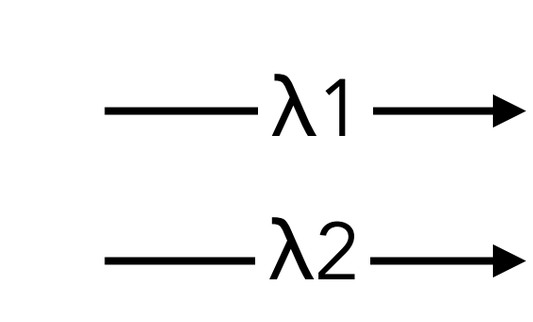
[1, 1.2, 1.7, 1.5, 0.7] ⟶ bar chart

Given data, we have:

- element value
- element index

Need to compute:

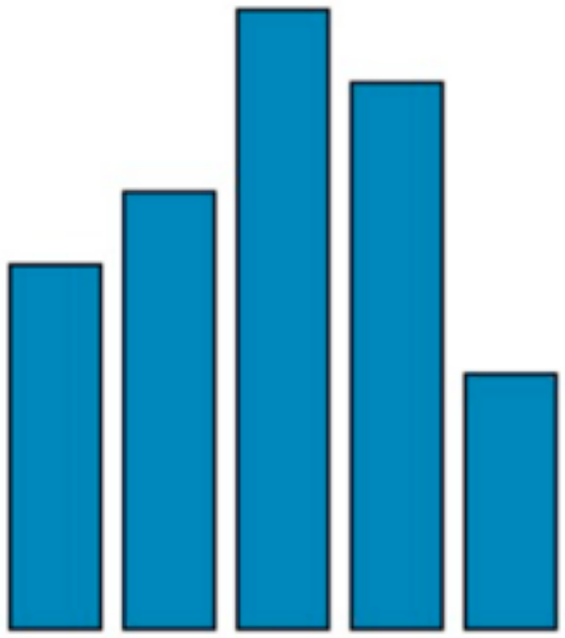
- bar height
- bar position




Visualization as transformation of data properties!

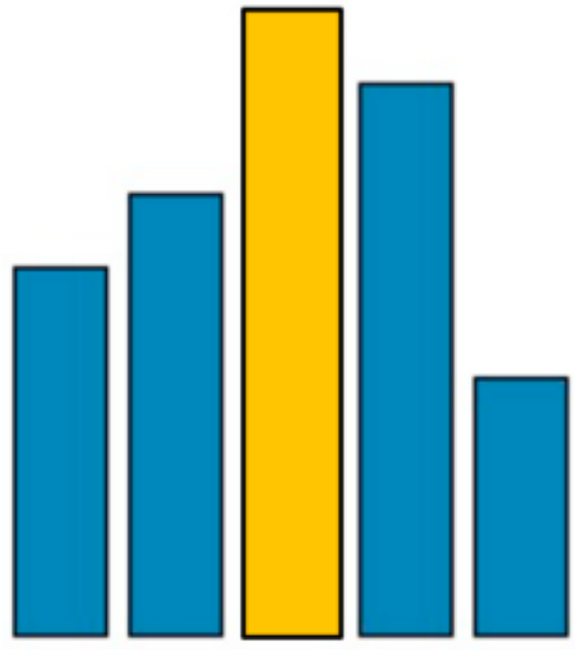
**BAR** $\lambda: D \rightarrow R$

| data | 1 | 1.2 | 1.7 | 1.5 | 0.7 |
|---|---|---|---|---|---|
| visible | true | | | | |
| left | λ: index * 25 | | | | |
| bottom | 0 | | | | |
| width | 20 | | | | |
| height | λ: datum * 80 | | | | |
| fillStyle | blue | | | | |
| strokeStyle | black | | | | |
| lineWidth | 1.5 | | | | |
| ... | ... | | | | |



**BAR** $\lambda: D \rightarrow R$

| data | 1 | 1.2 | 1.7 | 1.5 | 0.7 |
|---|---|---|---|---|---|
| visible | true | | | | |
| left | 2 * 25 | | | | |
| bottom | 0 | | | | |
| width | 20 | | | | |
| height | 1.7 * 80 | | | | |
| fillStyle | blue | | | | |
| strokeStyle | black | | | | |
| lineWidth | 1.5 | | | | |
| ... | ... | | | | |

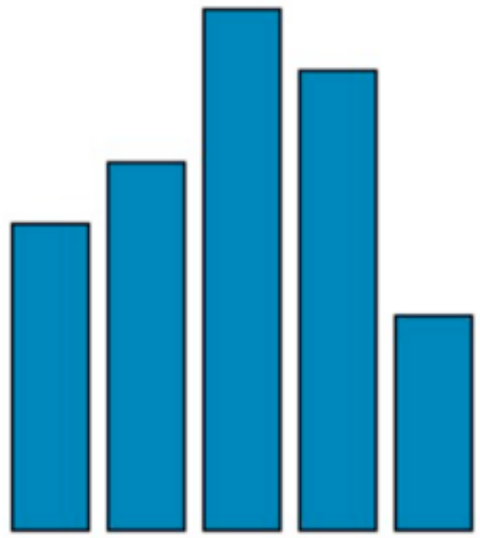




[Adapted from Heer Lecture.]

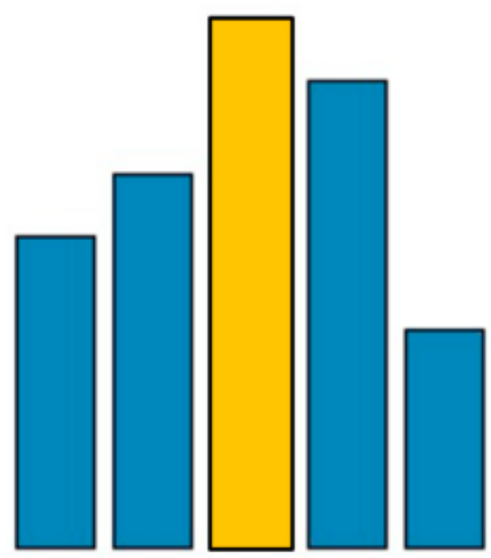
# Specifying the visual transformations in code

| BAR | λ: D → R | | | | |
|---|---|---|---|---|---|
| data | 1 | 1.2 | 1.7 | 1.5 | 0.7 |
| visible | true | | | | |
| left | λ: index * 25 | | | | |
| bottom | 0 | | | | |
| width | 20 | | | | |
| height | λ: datum * 80 | | | | |
| fillStyle | blue | | | | |
| strokeStyle | black | | | | |
| lineWidth | 1.5 | | | | |
| ... | ... | | | | |



```
d3.selectAll('rect')
  .data([1,1.2,1.7,1.5,0.7])
  .enter()
  .append('rect')
  .attr('fill','steelblue')
  .attr('transform',function(d,i) {
    var left = i*25;
    var top = 400-d*80;
    return 'translate(' + left +
    ',' + top +')';
})
.attr('width',20)
.attr('height',function(d) {
    return d * 80;
})
```

| BAR | λ: D → R | | | | |
|---|---|---|---|---|---|
| data | 1 | 1.2 | 1.7 | 1.5 | 0.7 |
| visible | true | | | | |
| left | 2 * 25 | | | | |
| bottom | 0 | | | | |
| width | 20 | | | | |
| height | 1.7 * 80 | | | | |
| fillStyle | blue | | | | |
| strokeStyle | black | | | | |
| lineWidth | 1.5 | | | | |
| ... | ... | | | | |

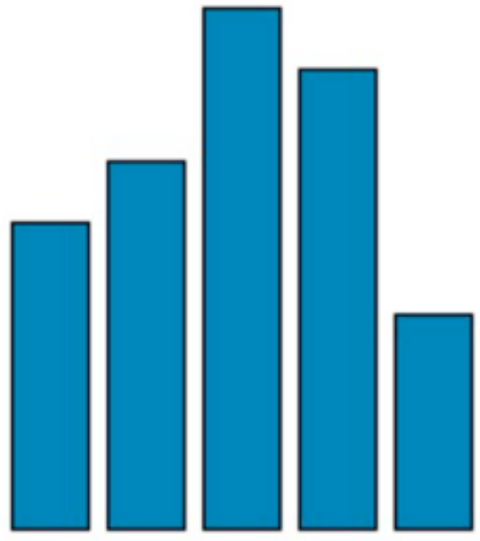


.....

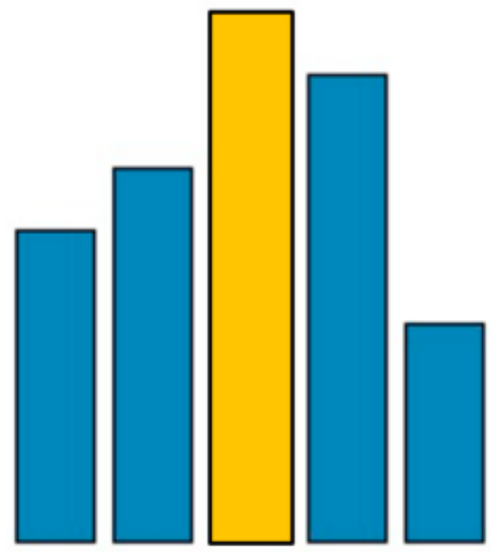
See https://github.com/mbostock/d3/wiki/SVG-Shapes to learn more about SVG attributes

# *Method chaining*

| BAR | λ: D → R | | | | |
|---|---|---|---|---|---|
| data | 1 | 1.2 | 1.7 | 1.5 | 0.7 |
| visible | true | | | | |
| left | λ: index * 25 | | | | |
| bottom | 0 | | | | |
| width | 20 | | | | |
| height | λ: datum * 80 | | | | |
| fillStyle | blue | | | | |
| strokeStyle | black | | | | |
| lineWidth | 1.5 | | | | |
| ... | ... | | | | |



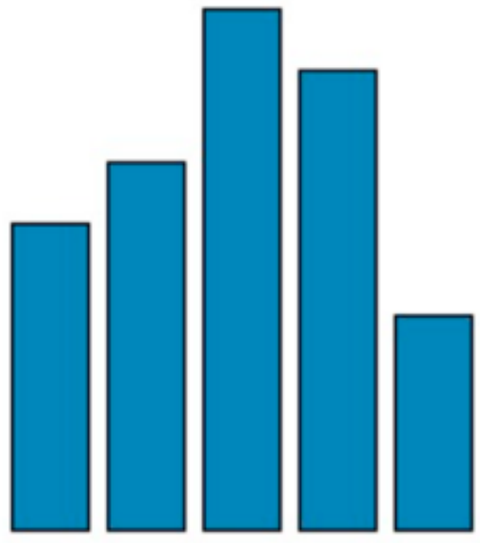
| BAR | λ: D → R | | | | |
|---|---|---|---|---|---|
| data | 1 | 1.2 | 1.7 | 1.5 | 0.7 |
| visible | true | | | | |
| left | 2 * 25 | | | | |
| bottom | 0 | | | | |
| width | 20 | | | | |
| height | 1.7 * 80 | | | | |
| fillStyle | blue | | | | |
| strokeStyle | black | | | | |
| lineWidth | 1.5 | | | | |
| ... | ... | | | | |



```
d3.selectAll('rect')
  .data([1,1.2,1.7,1.5,0.7])
  .enter()
  .append('rect')
  .attr('fill','steelblue')
  .attr('transform',function(d,i) {
    var left = i*25;
    var top = 400-d*80;
    return 'translate(' + left +
    ',' + top +')';
})
.attr('width',20)
.attr('height',function(d) {
    return d * 80;
})
……
```

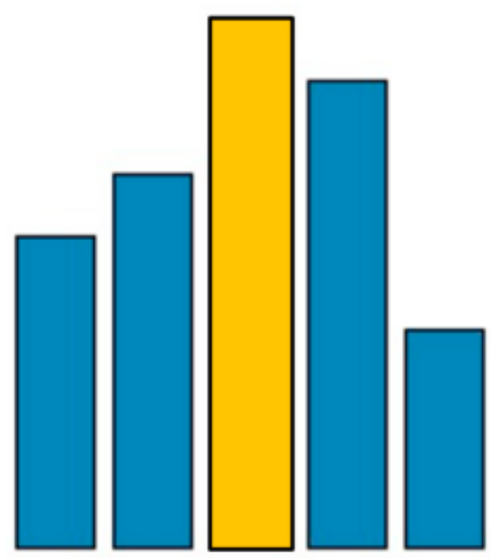
# Same code w/o method chaining

| BAR | λ: D → R |
|---|---|
| data | 1 1.2 1.7 1.5 0.7 |
| visible | true |
| left | λ: index * 25 |
| bottom | 0 |
| width | 20 |
| height | λ: datum * 80 |
| fillStyle | blue |
| strokeStyle | black |
| lineWidth | 1.5 |
| ... | ... |



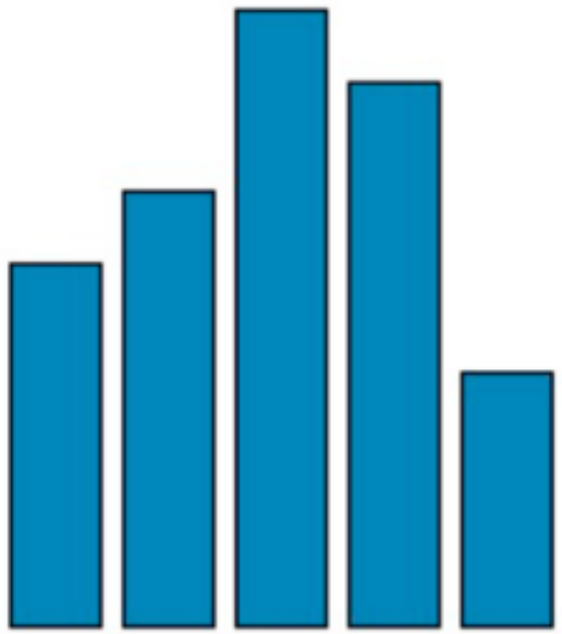
```
var rects = d3.selectAll('rect')
  .data([1,1.2,1.7,1.5,0.7])
  .enter()
  .append(‘rect’);

 rects.attr(‘fill’,’steelblue');
```

| BAR | λ: D → R |
|---|---|
| data | 1 1.2 1.7 1.5 0.7 |
| visible | true |
| left | 2 * 25 |
| bottom | 0 |
| width | 20 |
| height | 1.7 * 80 |
| fillStyle | blue |
| strokeStyle | black |
| lineWidth | 1.5 |
| ... | ... |



```
rects.attr(‘transform',function(d
,i) {
  var left = i*25;
  var top = 400-d*80;
  return 'translate(' + left +
  ',' + top +')';
});
……
```

# Drawing Marks through Document Object Model (DOM)

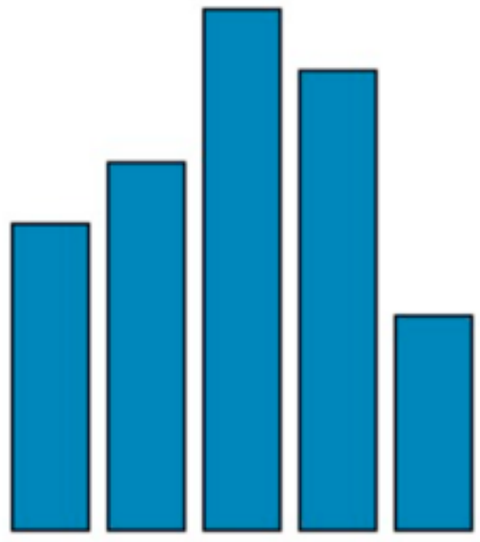


```
d3.selectAll('rect')
  .data([1,1.2,1.7,1.5,0.7])
  .enter()
  .append('rect')
  .attr('fill','steelblue')
  .attr('transform',function(d,i) {
    var left = i*25;
    var top = 400-d*80;
    return 'translate(' + left + ',' + top
    +')';
  })
  .attr('width',20)
  .attr('height',function(d) {
      return d * 80;
})
……
```

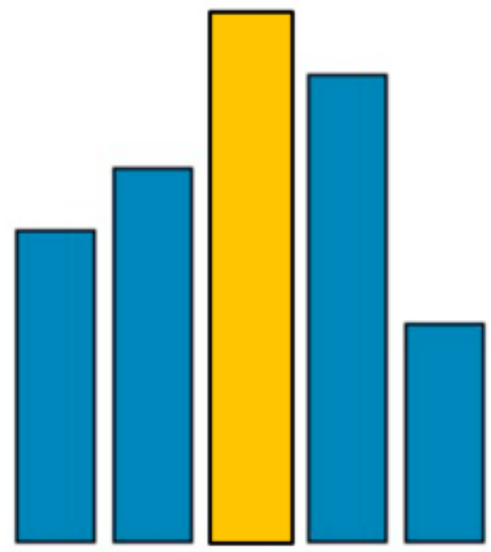
```
▼ <html>
  ▶ <head>…</head>
  ▼ <body>
    ▼ <svg id="canvas" style="width:420;height:420">
        <rect width="20" height="80" fill="steelblue" transform="translate(0,320)"></rect>
        <rect width="20" height="96" fill="steelblue" transform="translate(25,304)"></rect>
        <rect width="20" height="136" fill="steelblue" transform="translate(50,264)"></rect>
        <rect width="20" height="120" fill="steelblue" transform="translate(75,280)"></rect>
        <rect width="20" height="56" fill="steelblue" transform="translate(100,344)"></rect>
      </svg>
    </body>
  </html>
```

# Data Binding

| BAR | λ: D → R | | | | |
|---|---|---|---|---|---|
| data | 1 | 1.2 | 1.7 | 1.5 | 0.7 |
| visible | true | | | | |
| left | λ: index * 25 | | | | |
| bottom | 0 | | | | |
| width | 20 | | | | |
| height | λ: datum * 80 | | | | |
| fillStyle | blue | | | | |
| strokeStyle | black | | | | |
| lineWidth | 1.5 | | | | |
| ... | ... | | | | |

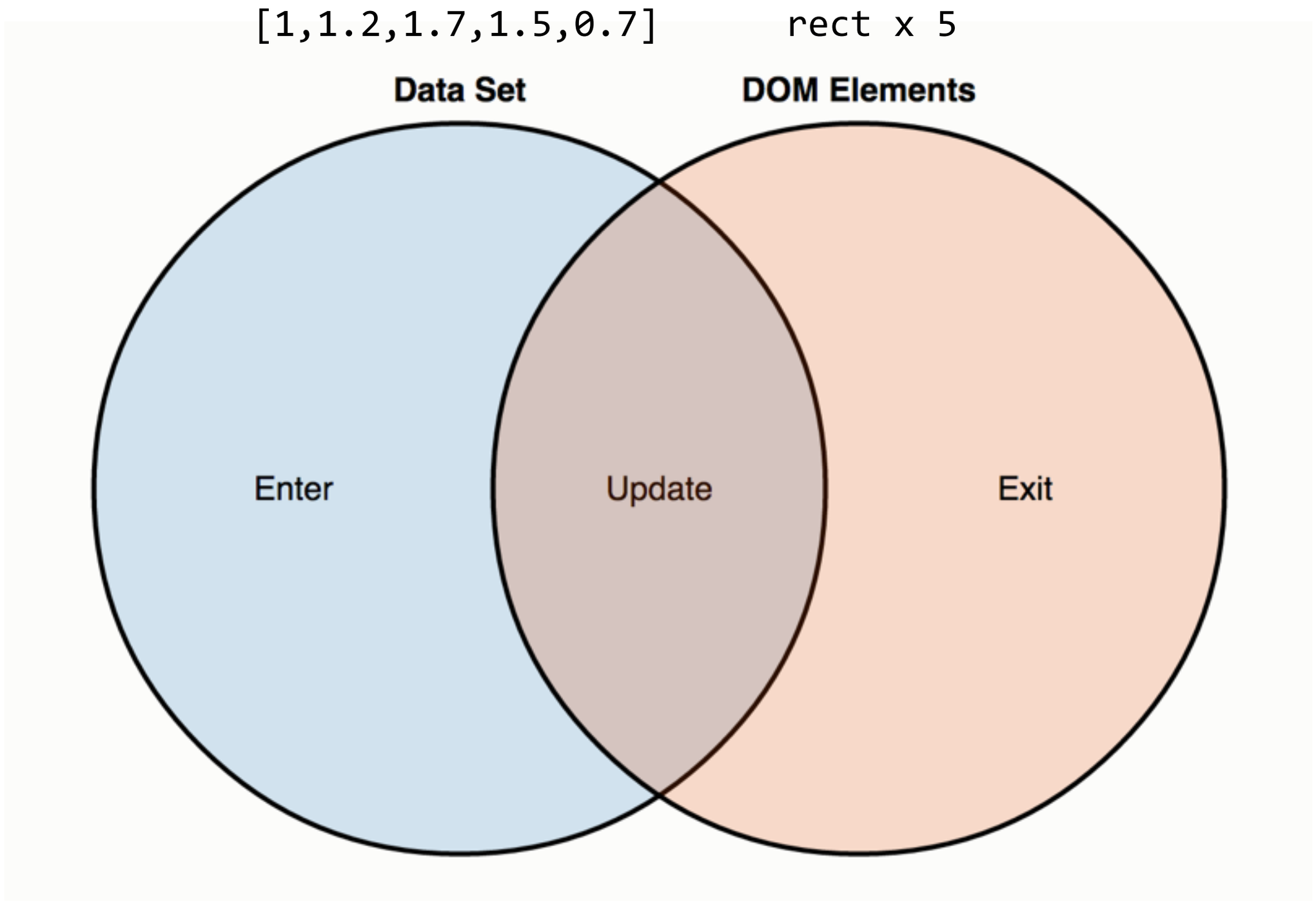


| BAR | λ: D → R | | | | |
|---|---|---|---|---|---|
| data | 1 | 1.2 | 1.7 | 1.5 | 0.7 |
| visible | true | | | | |
| left | 2 * 25 | | | | |
| bottom | 0 | | | | |
| width | 20 | | | | |
| height | 1.7 * 80 | | | | |
| fillStyle | blue | | | | |
| strokeStyle | black | | | | |
| lineWidth | 1.5 | | | | |
| ... | ... | | | | |



```
d3.selectAll('rect')
  .data([1,1.2,1.7,1.5,0.7])
  .enter()
  .append('rect')
  .attr('fill','steelblue')
  .attr('transform',function(d,i) {
    var left = i*25;
    var top = 400-d*80;
    return 'translate(' + left +
    ',' + top +')';
})
.attr('width',20)
.attr('height',function(d) {
   return d * 80;
})
……
```

# Data Binding



# General Update Pattern

Problem: How to deal with dynamic data?

Example: Iris dataset, add one species at a time

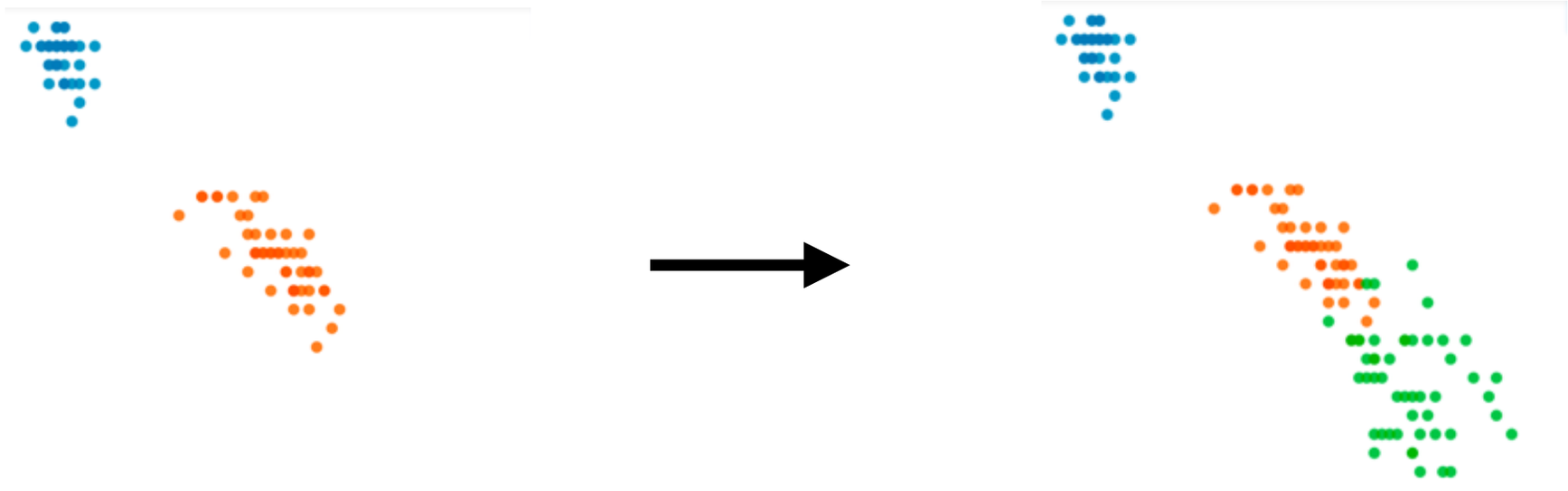
Data fields: species, petal length, petal width

Setosa

Setosa & Versicolor

All three species

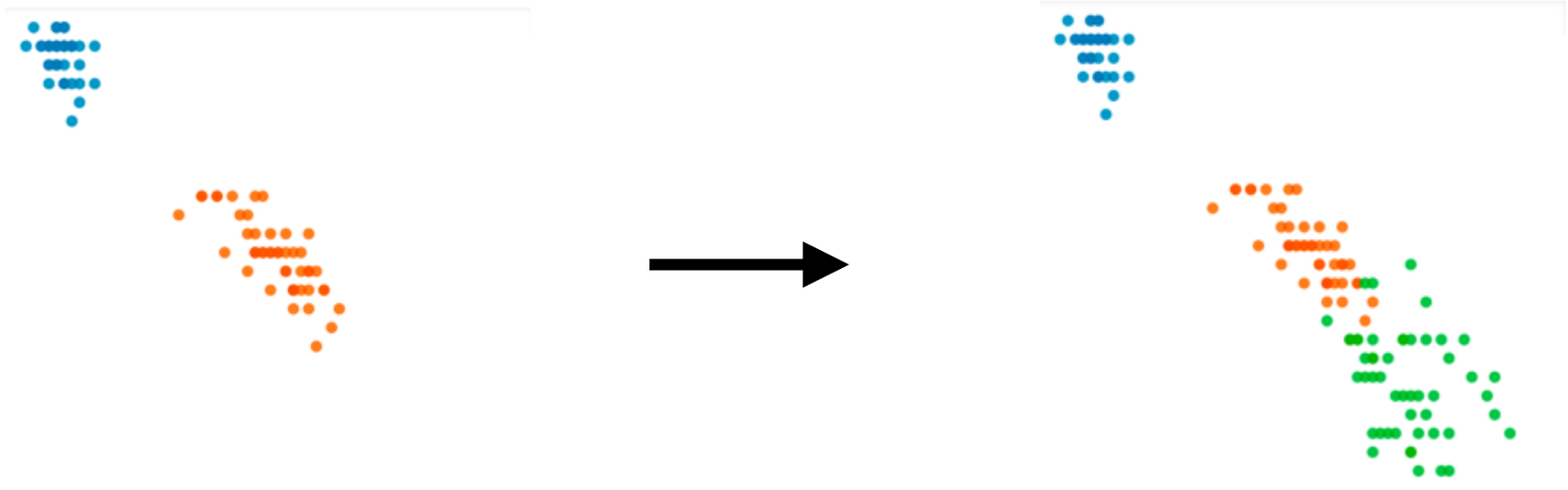
# General Update Pattern - Iris Example



```
visibleData = updateData(userSelection);
var circles = d3.select('#canvas').selectAll('circle')
  .data(visibleData);
circles.enter().append('circle');
circles.exit().remove();
configCircle(circles, iris);
```

Plunker full code example and live preview

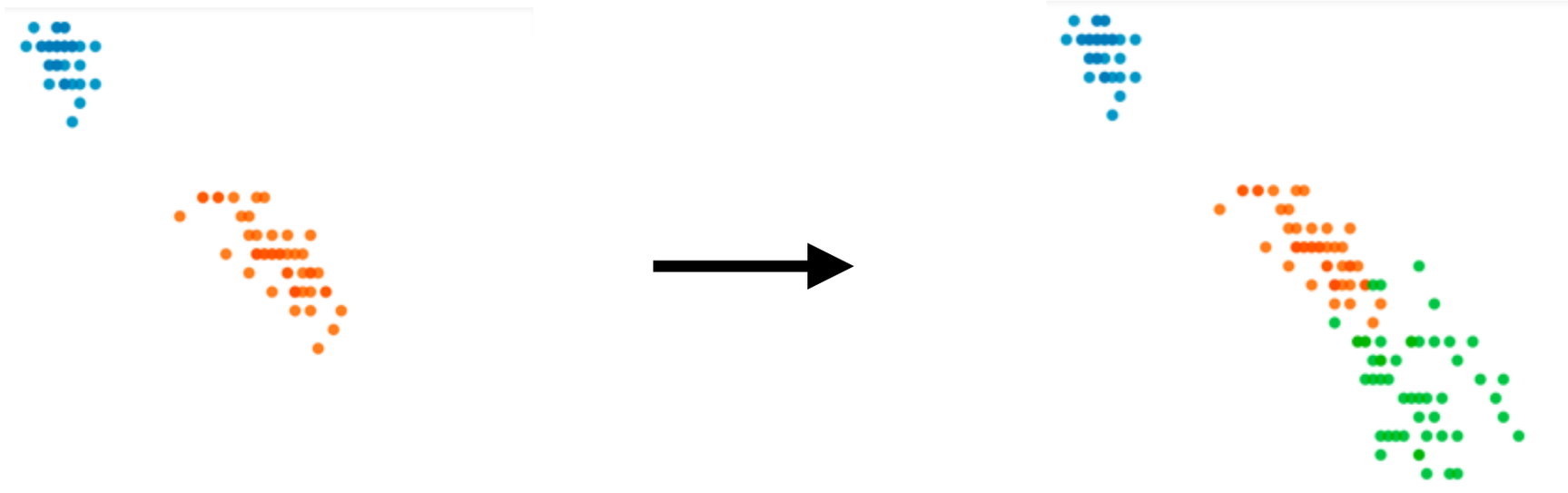
# General Update Pattern - Iris Example



```
visibleData = updateData(userSelection);
var circles = d3.select('#canvas').selectAll('circle')
  .data(visibleData);
circles.enter().append('circle');
circles.exit().remove();
configCircle(circles, iris);
```

Add more data into the data array

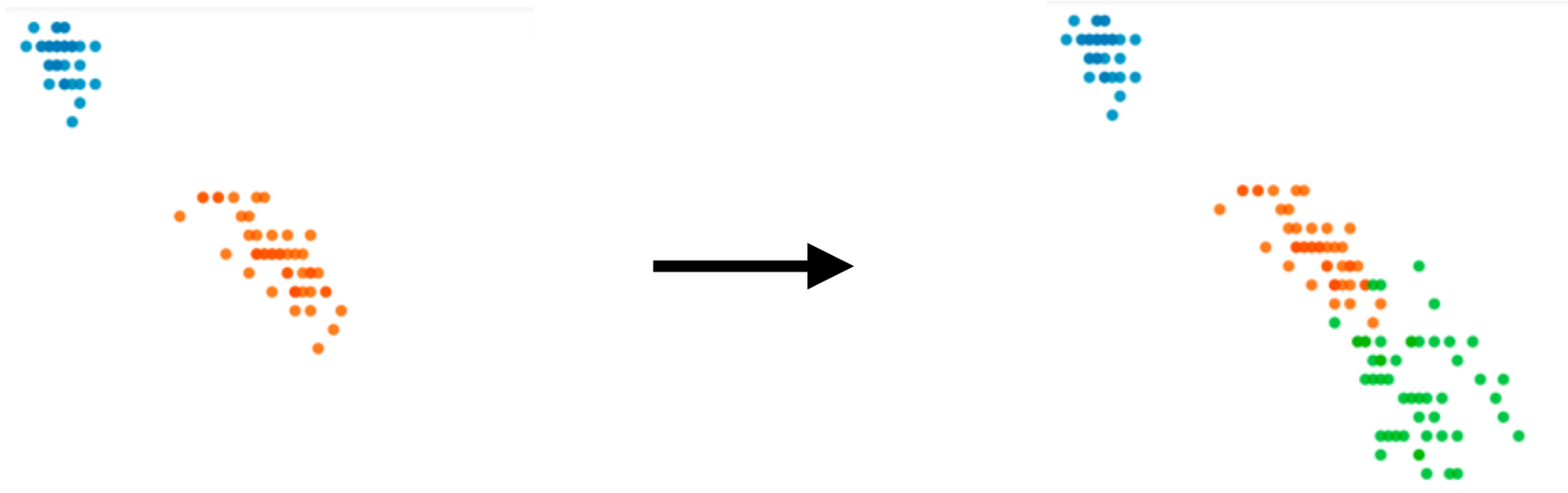
# General Update Pattern - Iris Example



```
visibleData = updateData(userSelection);
var circles = d3.select('#canvas').selectAll('circle')
  .data(visibleData);
circles.enter().append('circle');
circles.exit().remove();
configCircle(circles, iris);
```

Bind the updated data array to existing circles

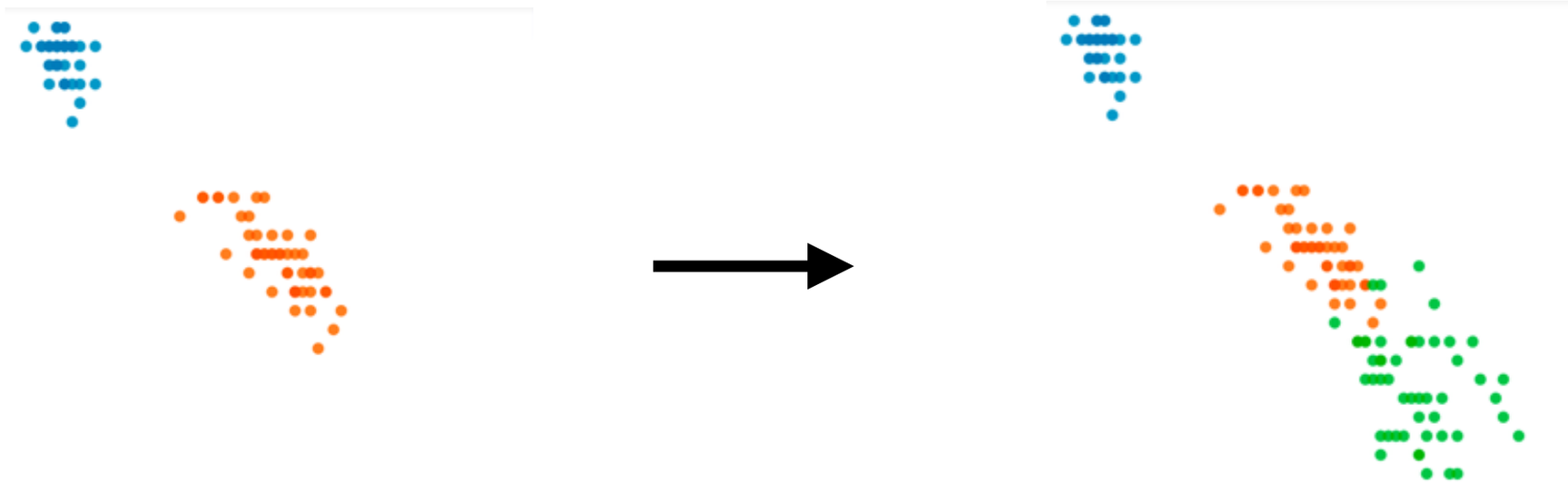
# General Update Pattern - Iris Example



```
visibleData = updateData(userSelection);
var circles = d3.select('#canvas').selectAll('circle')
  .data(visibleData);
circles.enter().append('circle'); ←—— green dots added
circles.exit().remove();
configCircle(circles, iris);
```

Specify actions for new data and removed data

# General Update Pattern - Iris Example

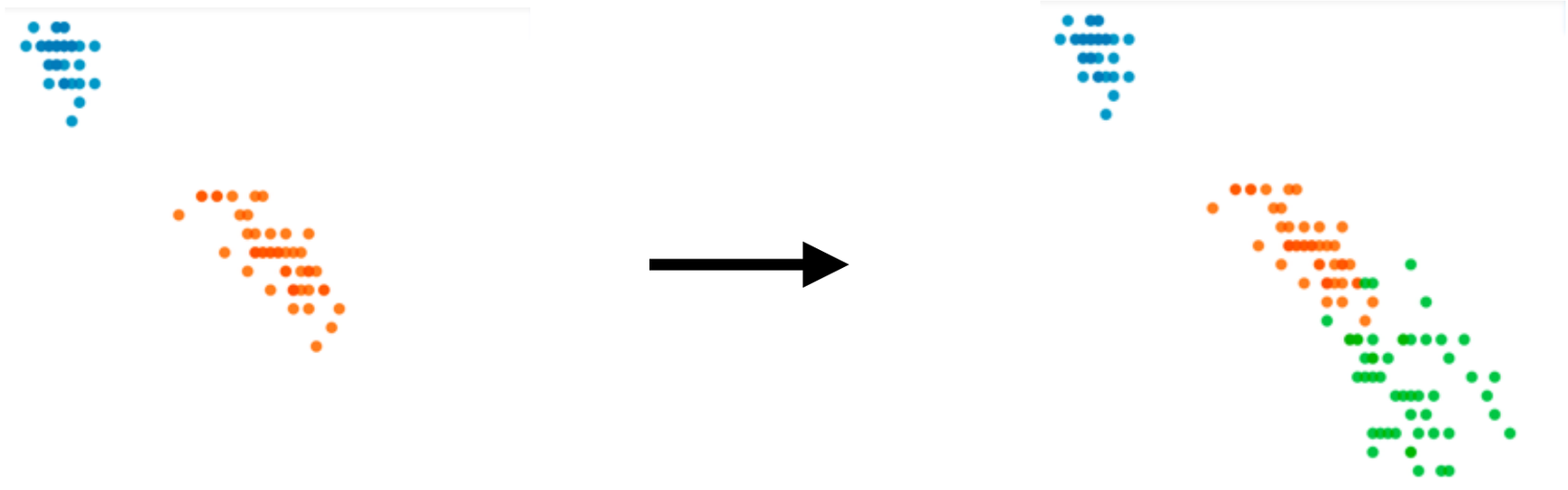


```
visibleData = updateData(userSelection);
var circles = d3.select('#canvas').selectAll('circle')
  .data(visibleData);
circles.enter().append('circle');
circles.exit().remove();
configCircle(circles, iris);
```

now we have one-to-one mapping between the data array and the selected DOM elements

## Specify actions for new data and removed data

# General Update Pattern - Iris Example



```
visibleData = updateData(userSelection);
var circles = d3.select('#canvas').selectAll('circle')
  .data(visibleData);
circles.enter().append('circle');
circles.exit().remove();
configCircle(circles, iris);
```

Specify visual transformations of data properties

# Data Binding on Nested Arrays

```
var groups = d3.select('body')
  .append('svg')
  .selectAll('g')
  .data([['Alice','Bob','Chris'], [27,19,24], ['RI','MA','CT']])
  .enter()
  .append('g')
  .attr('transform', function(d, i) {
    return 'translate(0,' + (i+1) * 20 + ')';
  });

groups.selectAll('text')
  .data(function(group) { return group; })
```

# Data Binding on Nested Arrays

```
var groups = d3.select('body')
  .append('svg')
  .selectAll('g')
  .data([['Alice','Bob','Chris'], [27,19,24], ['RI','MA','CT']])
  .enter()
  .append('g')
  .attr('transform', function(d, i) {
    return 'translate(0,' + (i+1) * 20 + ')';
  });

groups.selectAll('text')
  .data(function(group) { return group; })
  .enter()
  .append('text')
  .text(function(d) { return d; })
  .attr('transform', function(d, j) {
    return 'translate(' + j * 20 + ', 0)';
  });
```

# General Update Pattern - Iris Example

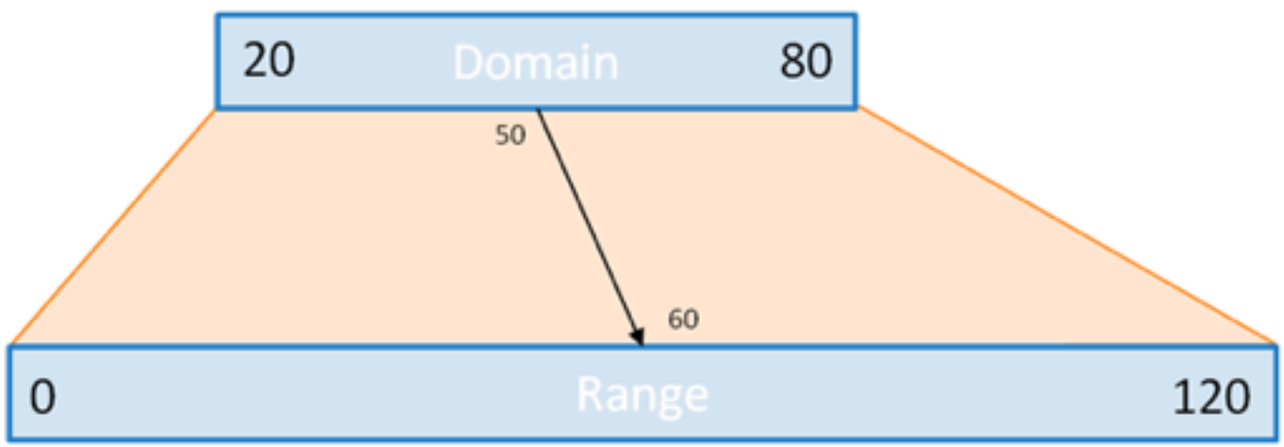
```
function configCircle(circles, data) {
  var positionScales = getScales(data);
  var colorScale = d3.scale.category10()
    .domain(['setosa', 'versicolor', 'virginica']);

  circles
    .attr('cx', function(d) {
      return positionScales.x(d.petal_length);
    })
    .attr('cy', function(d) {
      return positionScales.y(d.petal_width);
    })
    .attr('r', 5)
    .attr('fill', function(d) {
      return colorScale(d.species);
    })
    .attr('opacity', 0.8);
}
```

# General Update Pattern - Iris Example

```
function configCircle(circles, data) {
  var positionScales = getScales(data);
  var colorScale = d3.scale.category10()
    .domain(['setosa', 'versicolor', 'virginica']);

  circles
    .attr('cx', function(d) {
      return positionScales.x(d.petal_length);
    })
    .attr('cy', function(d) {
      return positionScales.y(d.petal_width);
    })
    .attr('r', 5)
    .attr('fill', function(d) {
      return colorScale(d.species);
    })
    .attr('opacity', 0.8);
}
```

# Scales: data value space ⟶ pixel space

```
function getScales(data) {
  var scales = {};
  scales.x = d3.scale.linear()
    .domain([
      d3.min(data, function(d) { return d.petal_length; }),
      d3.max(data, function(d) { return d.petal_length; })
    ])
    .range([10, 400]); ⟵ range for the x-axis in pixel space
  ……
  return scales;
}
```



# Scales: continous domain

- d3.scale.quantize()

```
var q=d3.scale.quantize().domain([0,10]).range([0,2,8]);
q(0); // 0
q(3.33); // 0
q(3.34); // 2
```

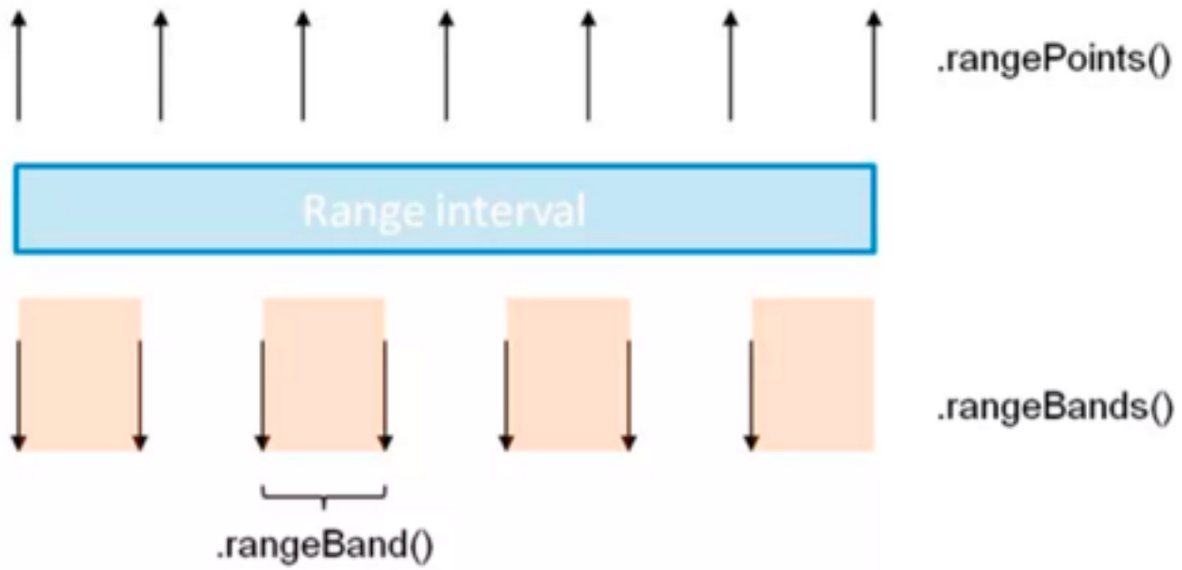
- d3.scale.quantile()

```
var q=d3.scale.quantile()
  .domain([0,1,5,6,2,4,6,2,4,6,7,8])
  .range([0,100]);

q.quantiles(); // [4.5], only one quantile - the median
q(4); // 0
q(4.499); // 0
q(4.5); // 100 - over the median
q(5); // 100
```

Examples from http://www.jeromecukier.net/blog/2011/08/11/d3-scales-and-color/. For more explanations and examples, see D3 documentation on scales.

# Scales: discrete domain

- d3.scale.ordinal()




```
var x=d3.scale.ordinal()
  .domain([“Sunday","Monday","Tuesday","Wednesday","Thur
sday","Friday","Saturday"]);

x.rangePoints([0,120]);
x("Saturday"); // 120
x.rangeBands([0,120]);
x("Saturday"); // 102.85714285714286
x("Saturday")+x.rangeBand(); // 120
```

Examples from http://www.jeromecukier.net/blog/2011/08/11/d3-scales-and-color/. For more explanations and examples, see D3 documentation on scales.

# Animations

## Animating objects with one line

```
circles
  .attr('fill', 'steelblue')
  .attr('r', function(d) { return d * 2; });
  .attr('cx', 200)
  .attr('cy', 200)

circles
  .transition()
  .attr('cx', function(d) {
    return random(10, 300);
  })
  .attr('cy', function(d) {
    return random(10, 300)
  })
```

Declaration of the static properties

Initialization of the dynamic properties

Update of the dynamic properties

Plunker full code example and live preview

# Interactions

## Interaction via DOM event listeners

```
circles
  .on(‘click', function(d) {
   highlightCircle();
  })
  .on('mouseover', function(d) {
   showTooltip();
  });
```

## Common mouse events

click, mouseover, mouseout, mouseenter, mouseleave,

mousedown, mouseup

For difference between mouseover/mouseout and mouseenter/mouseleave: https://bl.ocks.org/mbostock/5247027

Full list of DOM events: https://developer.mozilla.org/en-US/docs/Web/Events#Standard_events

# Common Pitfalls

- Make sure that the parent container is initialized before performing data binding
  - e.g. d3.select('#canvas') might return empty selection if called before DOM is fully loaded
  - Use $(document).ready(callback) as a safety net if not loading data from file (requires jQuery)
- d3.selectAll('…').attr(…) v.s. d3.selectAll('…').style(…)
  - equivalent for SVG objects; specify styles via CSS is the same as .style(); just don't mix and match!
- Javascript behaviors
  - Arrays and objects are passed by reference by default
  - Asynchronous loading

# Additional libraries

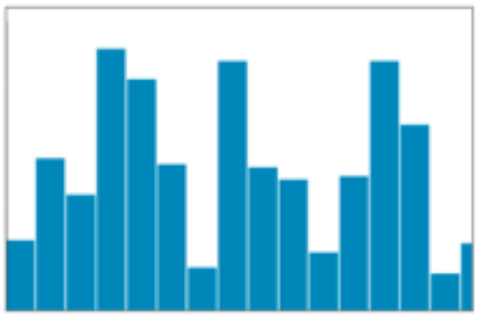
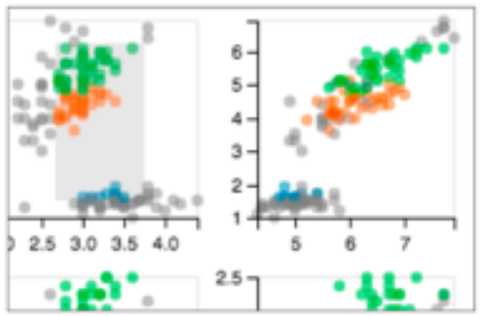
- D3-based chart libraries
  - NVD3
  - C3.js
  - MetricsGraphics.js
- Data processing
  - Crossfilter - data grouping & filtering
- D3 development
  - D3Kit

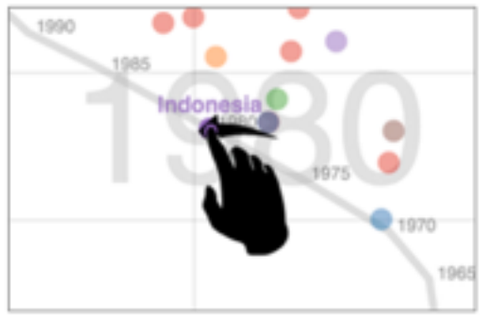
# D3 Pros and Cons

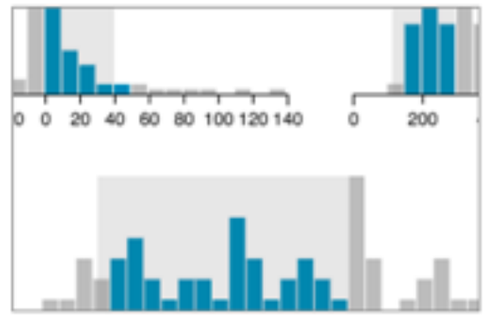
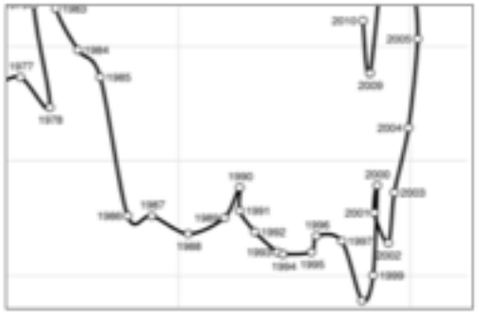
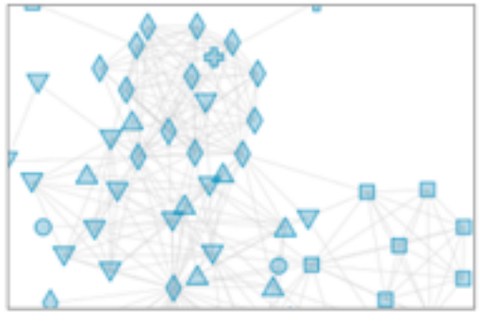
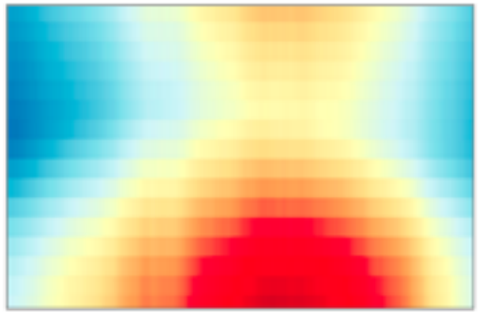
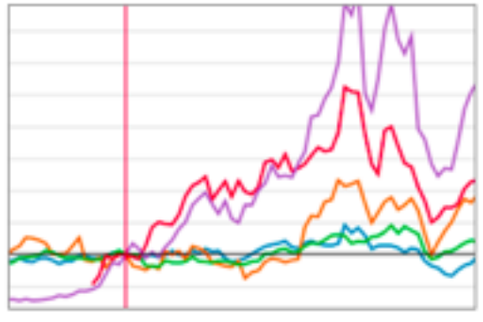
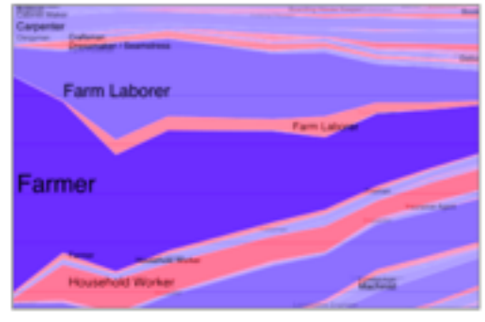
- Pros
  - seamless interoperability with Web
  - explicit transformation of scene
  - active user community with lots of examples
- Cons
  - limited scalability (dataset size and # of visual marks) - but you can use Canvas instead of SVG in d3 v4!

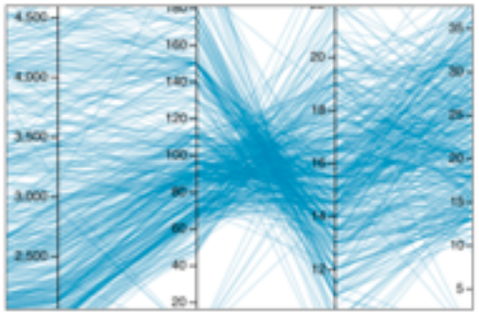
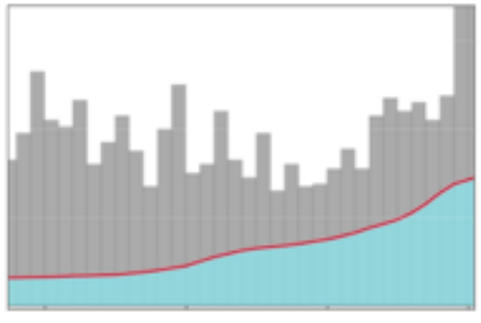
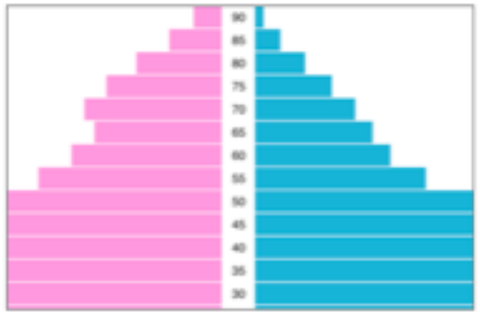
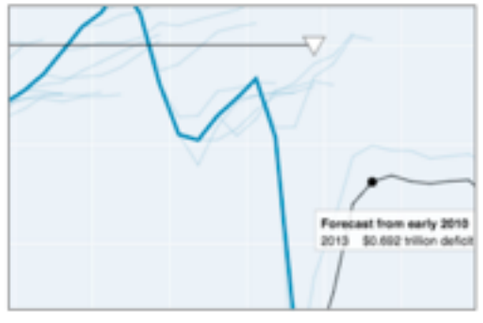
# vega

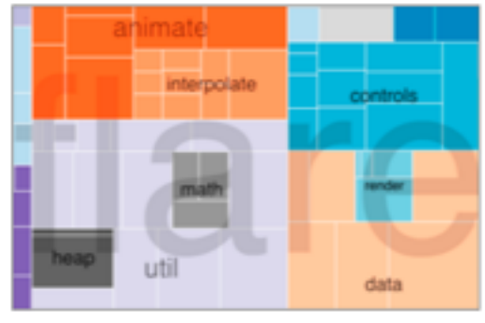
vega.min.js
JSON Schema
GitHub





Efficiency

Tableau, ggplot2, **Vega**

d3.js

Processing

Expressiveness

# Vega

- Declarative grammar, great for quick prototyping
- Pros
  - Web-based
  - little coding needed
- Cons
  - Limited expressiveness

```
{
  "name": "pie", "width": 150, "height": 150,
  "data": [{
    "name": "table",
    "values": [1, 1.2, 1.7, 1.5, 0.7],
    "transform": [{"type": "pie", "value": "data"}]
  }],
  "scales": [
    {"name": "color", "type": "ordinal", "range": "category20"}
  ],
  "marks": [{
    "type": "arc",
    "from": {"data": "table"},
    "properties": {
      "enter": {
        "x": {"group": "width", "mult": 0.5},
        "y": {"group": "height", "mult": 0.5},
        "startAngle": {"field": "startAngle"},
        "endAngle": {"field": "endAngle"},
        "innerRadius": {"value": 0},
        "outerRadius": {"value": 70},
        "fill": {"scale": "color", "field": "index"}
      }
    }
  }]
}
```

# Further Readings and References

- Visualization frameworks
  - Heer lecture: http://courses.cs.washington.edu/courses/cse512/14wi/lectures/CSE512-Tools.pdf
  - Munzner lecture: http://www.cs.ubc.ca/~tmm/courses/533-11/slides/toolkits.pdf
- D3
  - http://blockbuilder.org/ - fork and learn from D3.js examples
  - https://bost.ocks.org/mike/circles/
  - http://christopheviau.com/d3_tutorial/
  - https://www.dashingd3js.com/table-of-contents