CSC 544
Data Visualization

Joshua Levine
josh@arizona.edu

Lecture 04
d3 Joins and Scales

Jan. 25, 2023

Today's Agenda

- Reminders:
  - A00 graded, check out D2L
  - A01 posted, questions?
  - P01 posted, due in 4 weeks: Feb. 22
- Goals for today: Continue discussing d3 concepts, scales, etc.

Project Milestone 01
Proposal

Assigned: Monday, January 25
Due: Monday, February 22, 4:59:59 pm

Recall: Selections + Binding Data

Selections

- Recall that `d3.select()` and `d3.selectAll()` return **selections**
- Given a selection, we can use:
  - `selection.append(name)`, `selection.remove()`, and `selection.text(value)` that directly *modify* the DOM
  - `selection.attr(name, value)`, `selection.style(name, value)` that modify DOM element *attributes*
  - In general, these methods accept anonymous functions that will be called once per selection element.

Binding Data

- Given a selection in d3, once can bind data to it using `.data()`
- This builds a mapping between each element in the selection and each data element
    - One can control this in lots of ways, but the default is sequential, element `i` is mapped to data at index `i`.

Accessing Bound Data

- Once bound, one can use the data to define attributes:

  - ```
    .style("width", function(d) {
        return d * 100;
    });
    ```

    This would set the width of each element in the selection to d*100.

  - Can also use `function(d,i)` if one wants to access the index `i` of the data element in addition to its value `d`
    ```

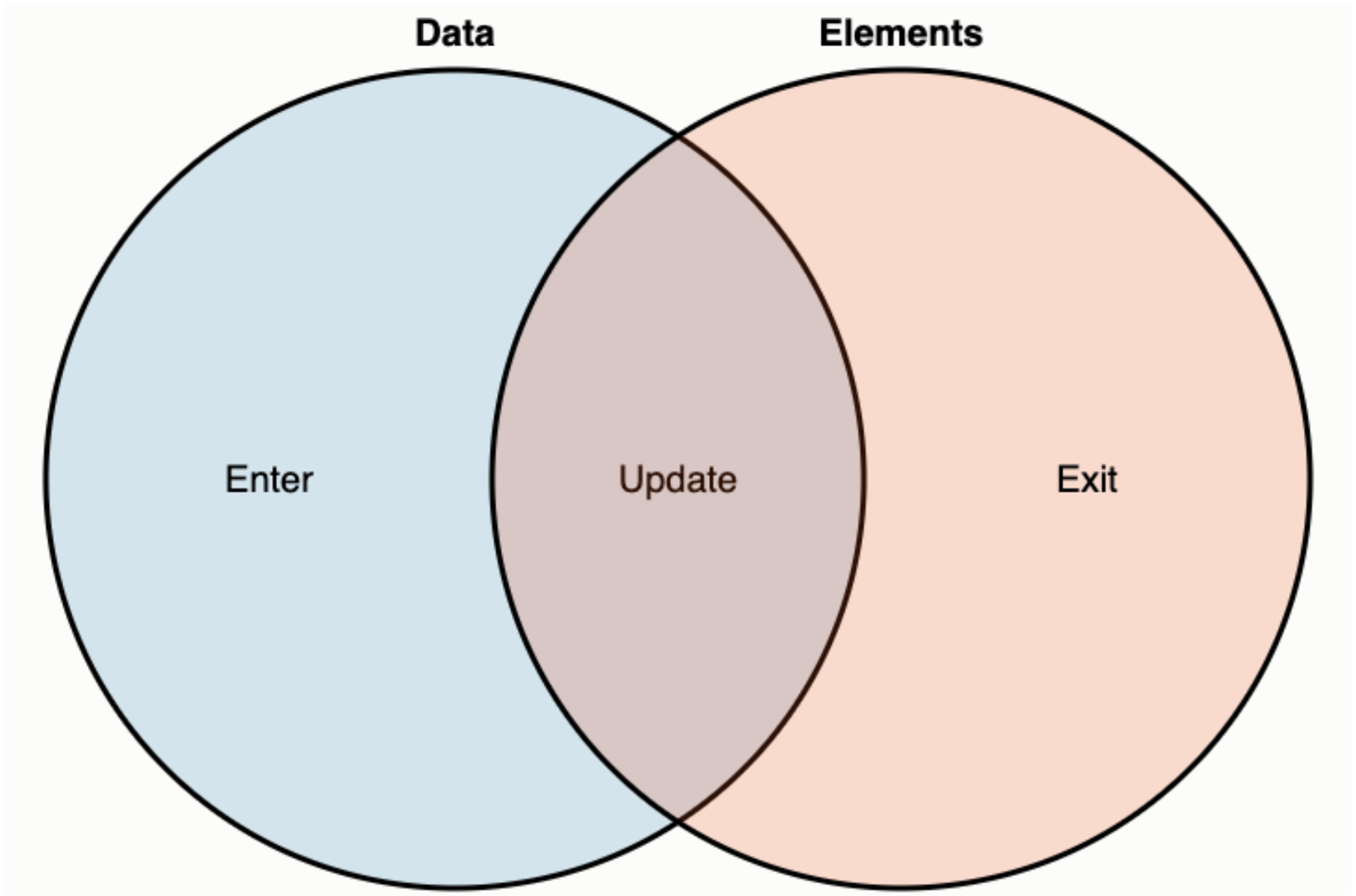
Data Joins

What Happens to Selections with Data?

- Three possibilities can happen once a selection is bound to data:
  - Update: Selection has the *same number* of elements as the number of values in data
  - Enter: Selection has *fewer* elements than data
  - Exit: Selection has *more* elements

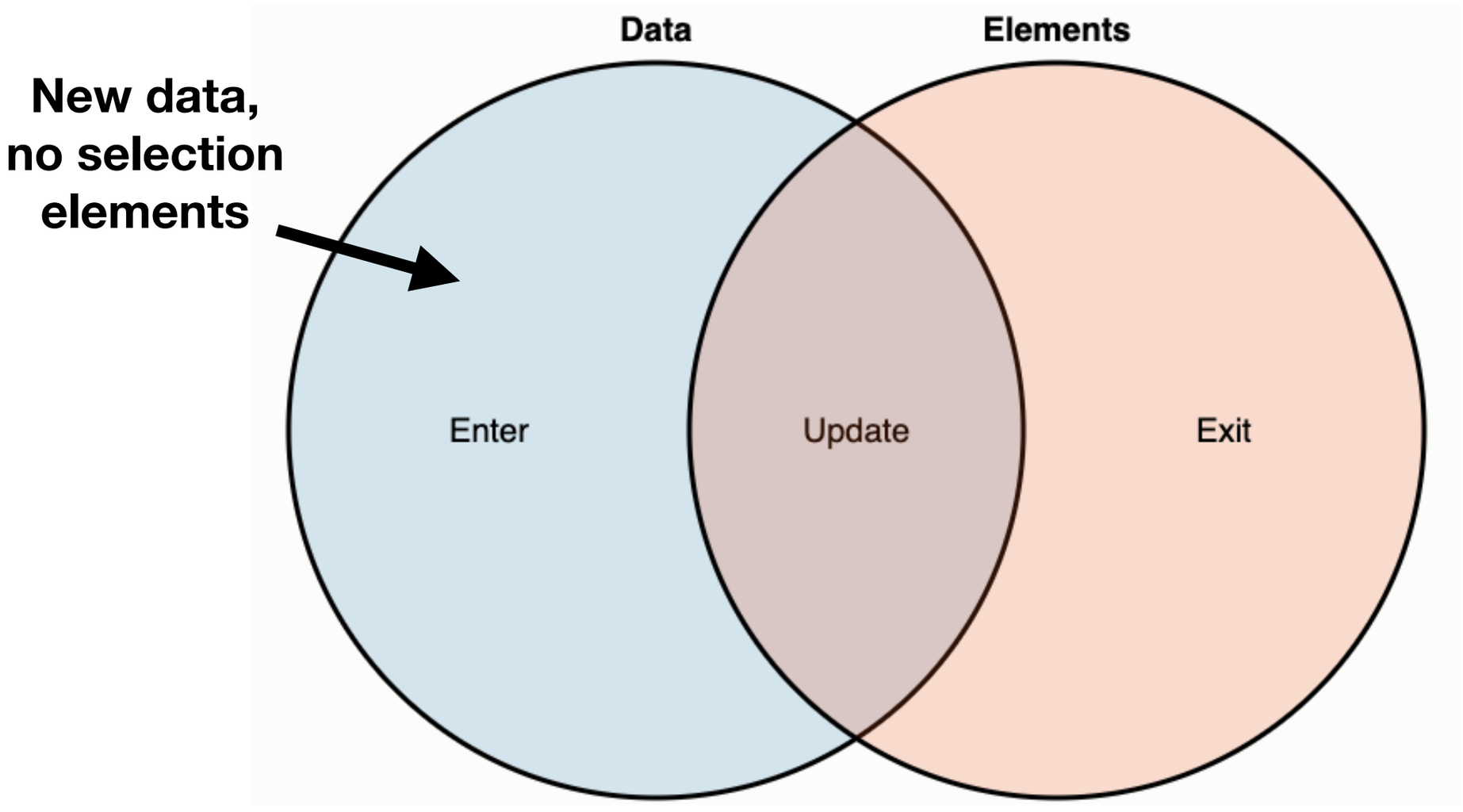
Three Parts to a Data Join

- Once bound to data, there are 3 possibilities for a selection (https://bost.ocks.org/mike/join/):



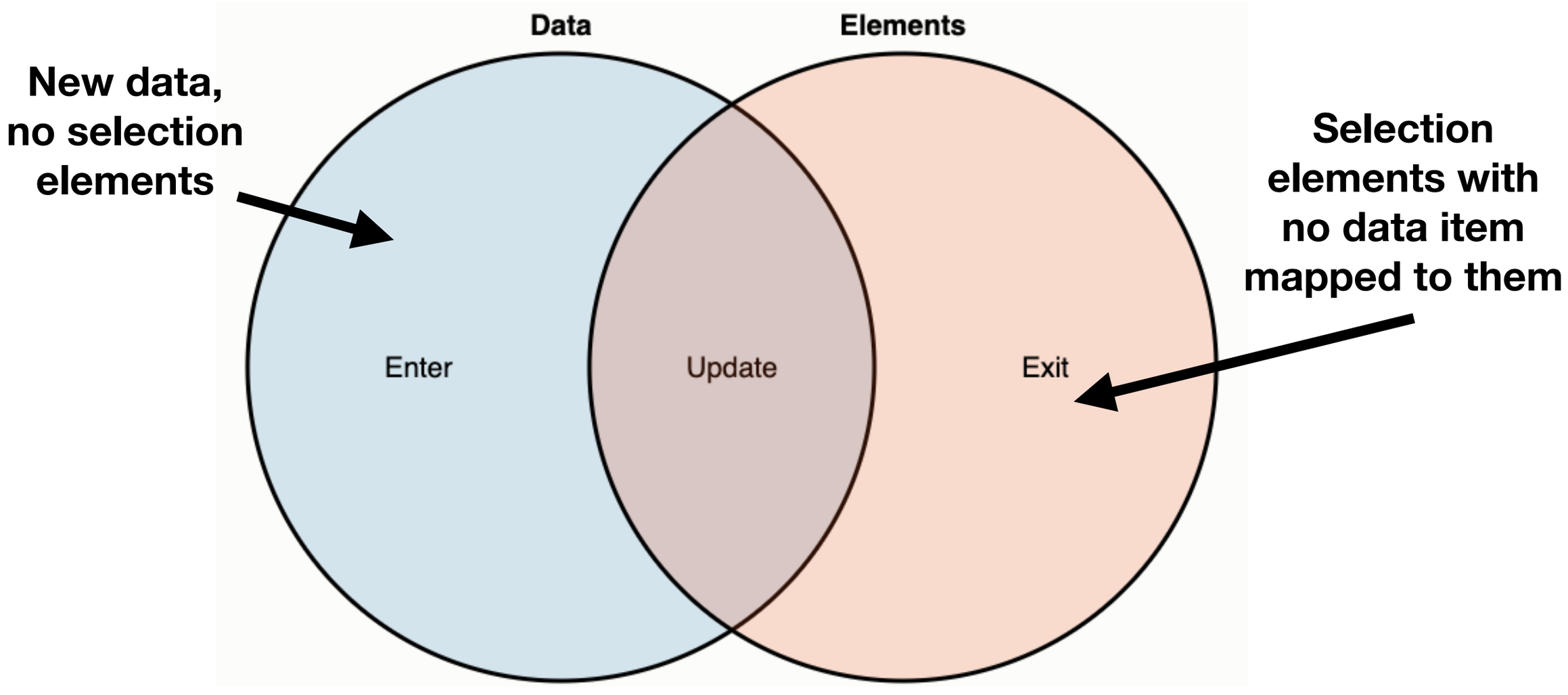
Three Parts to a Data Join

- Once bound to data, there are 3 possibilities for a selection (https://bost.ocks.org/mike/join/):



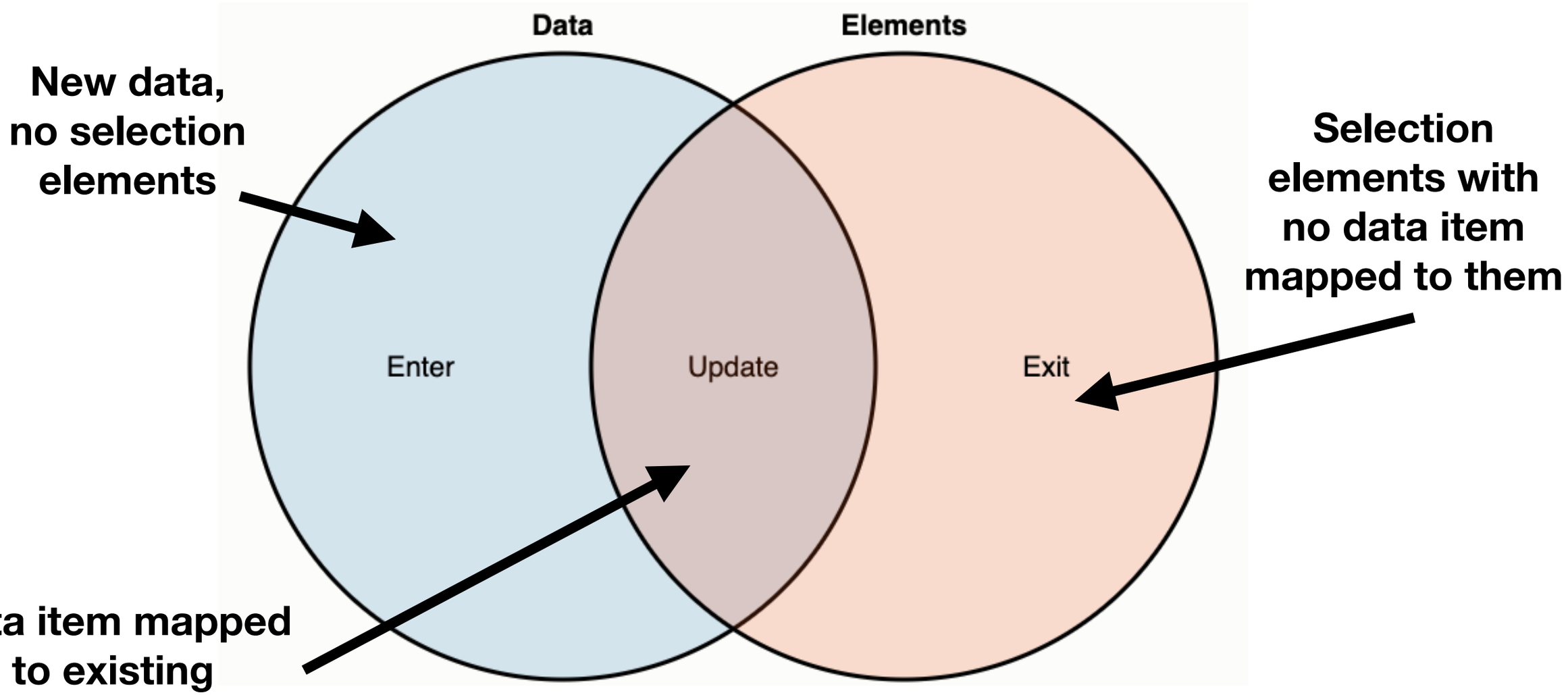
Three Parts to a Data Join

- Once bound to data, there are 3 possibilities for a selection (https://bost.ocks.org/mike/join/):



Three Parts to a Data Join

- Once bound to data, there are 3 possibilities for a selection (https://bost.ocks.org/mike/join/):



.enter(): Creating New Elements

- In many instances, there may not be enough elements for the number of data elements.

- `.enter()` returns the **enter selection**, one entry per an unmapped data element.

- Typically then used to `.append()` new elements

.exit(): Removing Extraneous Elements

- In some instances, there may already be too many elements for the number of data elements.

- `.exit()` returns the **exit selection**, one entry per an unmapped html element.

- Typically then used to `.remove()` existing elements one no longer needs to draw.

Break for d3.js Demo

How Does Data Binding Work?

- Bound data is **not** a property of the selection, it's a property of the its **elements**.
    - D3 actually stores this in the DOM for you
- As a result, bound data *persists*, even though selections are *transient*: if you reselect elements from the DOM they'll still be aware of their bound data
- See https://bost.ocks.org/mike/selection/ for more details

Data Ordering w/ Keys

- When binding with `.data()`, can also pass an anonymous function to define the **key** for each data element

```
let imdbResult = [
  {
    title: "Gravity",
    id: "tt1454468",
    year: 2013,
    gross: 274084951
  }, ... ];
selection.data(imdbResult,
  function(d) { return d.id; })
```

- If a new dataset comes in, elements will be mapped according to the `id` key

- See D3 Drills 2-3 from https://cscheid.net/projects/d3-drills/

Other Useful Methods on Selections

- `selection.classed(name, value)`, which sets the class to `name` for all elements in the selection if `value` is truthy, e.g.

```
selection.classed('important',
function(d) { return d.value > 100; })
```

Other Useful Methods on Selections

- `selection.classed(name, value)`, which sets the class to `name` for all elements in the selection if `value` is truthy, e.g.

```
selection.classed('important',
    function(d) { return d.value > 100; })
```

- `selection.call(function)`, which calls `function` once per selection. Very convenient for setting multiple attributes

Other Useful Methods on Selections

- `selection.classed(name, value)`, which sets the class to `name` for all elements in the selection if `value` is truthy, e.g.

```
selection.classed('important',
    function(d) { return d.value > 100; })
```

- `selection.call(function)`, which calls `function` once per selection. Very convenient for setting multiple attributes

- `selection.filter(function)`, which returns a new subselection for all elements for which `function` returns true.

Nested Selections

- `.data()` can also accept anonymous functions as the data!

- This is particularly useful for nesting selections on multidimensional arrays and in combination with DOM groups or tables.

```
let D = [[1,2,3],[4,5,6]];
d3.selectAll(“g”)
  .data(D)
  .enter().append(“g”)       //creates 2 groups
  .selectAll(“circle”)
  .data(function(row) { return row; })
  .enter().append(“circle”) //creates 3 circles
  .attr(“r”, function(d) { return d; })…
```

- See D3 Drills 2-2 from https://cscheid.net/projects/d3-drills/

General Update Pattern

- Want to (1) remove exiting elements, (2) update existing elements, and (3) append new elements, all in one go:

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

```
var circle = svg.selectAll(“circle")
  .data(data);

circle.exit().remove();

circle.enter().append("circle")
    .attr("r", 2.5)
  .merge(circle)
    .attr("cx", function(d) { return d.x; })
    .attr(“cy", function(d) { return d.y; });
```

General Update Pattern, Written Poorly (on Purpose!) to be Easily Understood

```
//probably best not to do this!
let circle = svg.selectAll(“circle”).data(data);

circle.call(update);      //circle ONLY contains the changed elements
circle.exit().call(exiting);    //circle.exit() is exiting elements
circle.enter().call(entering);  //circle.enter() is entering elements

function update(selection) {
  selection
    .attr("cx", function(d) { return d.x; })
    .attr(“cy", function(d) { return d.y; });
}

function exiting(selection) {
  selection.remove();
}

function entering(selection) {
  selection.append("circle")
    .attr("r", 2.5)
    .call(update);  //this ensures entering elements get update props
}
```

.join(): General Update Pattern Revisited

- Newer method .join() handles this a bit more compactly:

```
var circle = svg.selectAll(“circle")
  .data(data);

circle.join(
  enter => enter.append(…),
  update => update.attr(…),
  exit => exit.remove()
);
```

- See https://observablehq.com/@d3/selection-join

Transitions

.transition(): Animating Change

- One can use transitions to animate the process of data joining, and to help highlight changes.

- d3 (mostly) takes care of smoothing out the transition for you.

- `selection.transition().attr(…target…)` will animate from whatever the current attributes are to the targets

- Whereas
  ```
  selection
     .attr(…start…)
     .transition()
     .attr(…target…)
  ```

  will animate from the start to the target

.duration() and .delay(): Controlling Timing

- `.duration()` can be used to specify the amount of time that the transition will take

- `.delay()` can be used to specify the amount of time before the transition begins

- Both can accept constants, but also both can accept anonymous functions for per-element delays/durations

Break for Questions

D3 Scales

Scales Explained

- In visualization, we will frequently need to map one space to another.
    - Example: in A03, you need to map the ACT scores to the y-axis in your scatterplot
- D3 provides a convenient set of mechanisms for this called **scales**
    - Bostock: *“Scales are functions that map from an input domain to an output range”*

Exercise

- Task:

Exercise

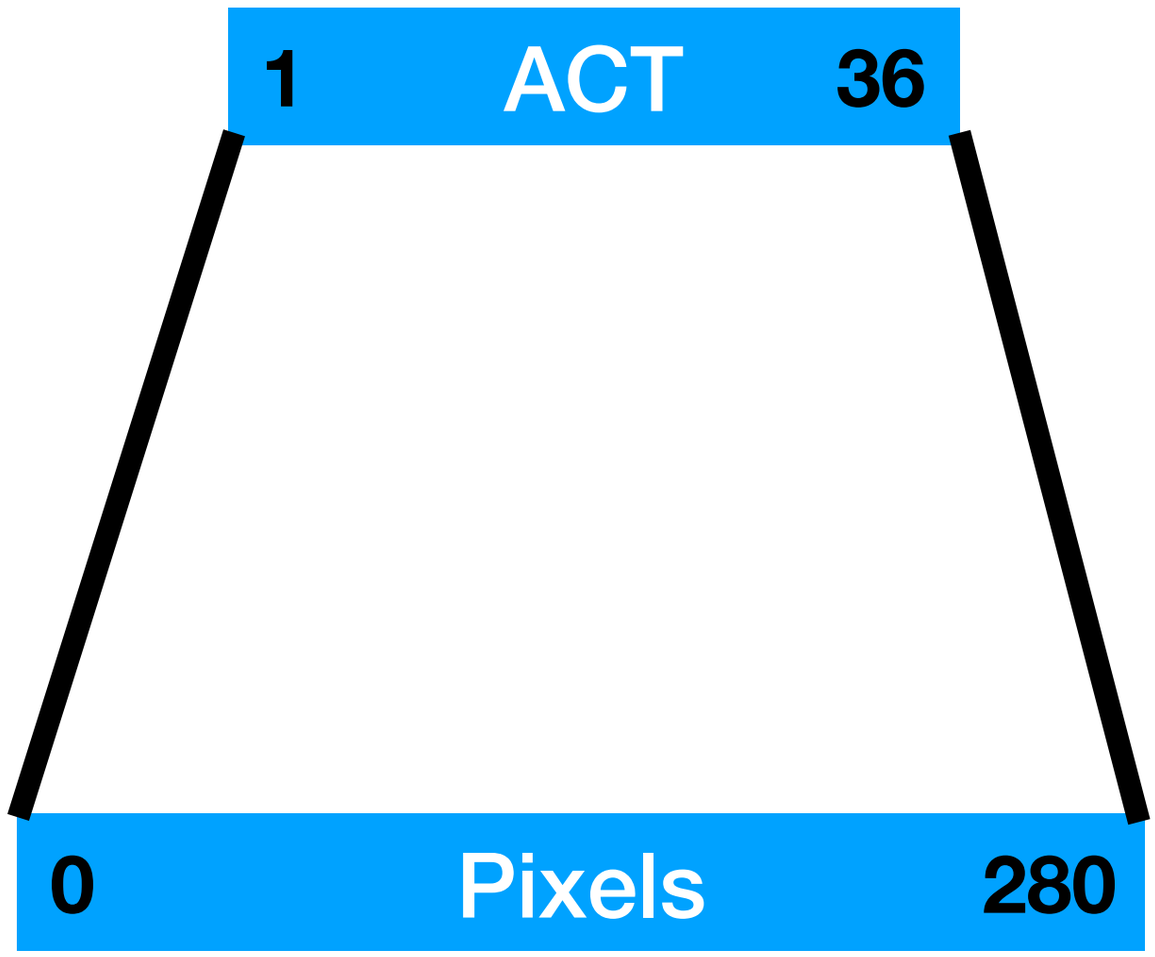
- Task:
  - Map ACT scores (from [1,36])…

Exercise

- Task:
  - Map ACT scores (from [1,36])…
  - …to positions on an SVG canvas (from [0,280])

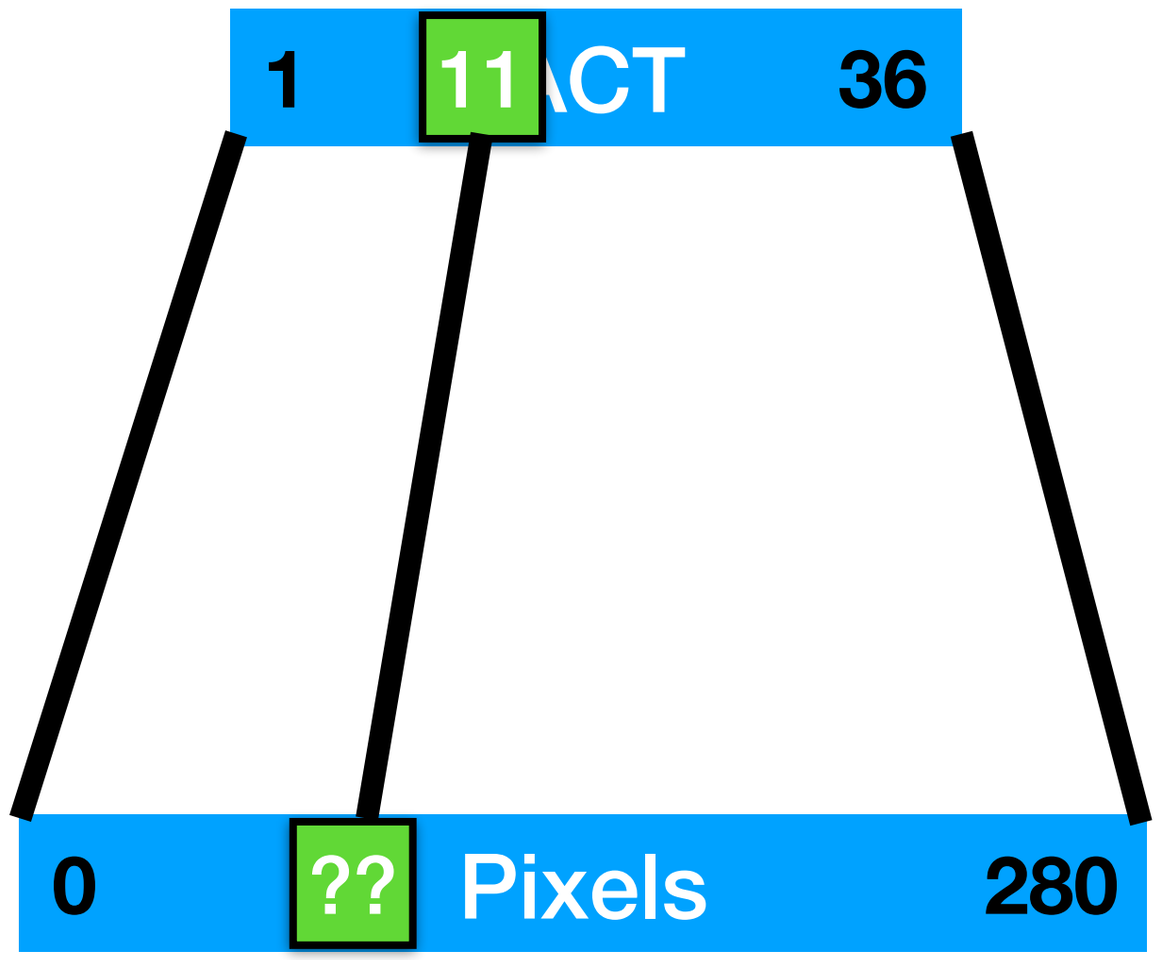
Exercise

- Task:
  - Map ACT scores (from [1,36])…
  - …to positions on an SVG canvas (from [0,280])
  - So that, given an arbitrary ACT score (e.g. 11) you can report what y position it should have: ??



Exercise

- Task:
  - Map ACT scores (from [1,36])…
  - …to positions on an SVG canvas (from [0,280])
  - So that, given an arbitrary ACT score (e.g. 11) you can report what y position it should have: ??



Simple Solution: “Normalization”

- First map ACT space to [0,1]:
  - normACT(x) = (x - min(ACT)) / (max(ACT) - min(ACT)
  - normACT(x) = (ACT - 1) / (36 - 1)
- Then map normalized ACT space to pixels:
  - pixACT(x) = normACT(x) * (max(PIX) - min(PIX)) + min(PIX)
  - pixACT(x) = normACT(x) * (280-0) + 0
- pixACT(11) = normACT(11) * 280 = (10/35) * 280 = 80

Linear Scales

- This method of normalization produces linear scales (which linearly interpolate values)

- Of course, we could write functions to do this over and over, and then use then as anonymous functions when we set attributes.

- d3 makes this easier with `d3.scaleLinear()`

- Exact syntax:

```
let actToPixelScale = d3.scaleLinear()
  .domain([1,36])
  .range([0,280]);

selection.attr("cy", function(d) {
  return actToPixelScale(d.ACT);
});
```

Linear Scales Work for Many Types

- Can also used non-numeric values, provided they can be coerced into numbers:

```
let actToColorScale = d3.scaleLinear()
  .domain([1,36])
  .range(["brown", "steelblue"]);

selection.attr(“fill”, function(d) {
  return actToColorScale(d.ACT);
});
```

Other Types of Scales

- `d3.scaleSqrt()`, power scale with exponent 0.5 (`d3.scalePow()` for general power scales)

- `d3.scaleLog()`, logarithmic scale

- `d3.scaleOrdinal()`, uses discrete domains, e.g. for mapping categories to colors

- Many many variants! See `.clamp()`, `.nice()` for adjustments to specific scales

- See https://cscheid.net/projects/d3-scale-playground/

Lec05 Reading

- Munzner, Chapter 1, 6.10

Reminder
Assignment 01

Assigned: Monday, January 23
Due: Monday, February 6, 4:59:59 pm