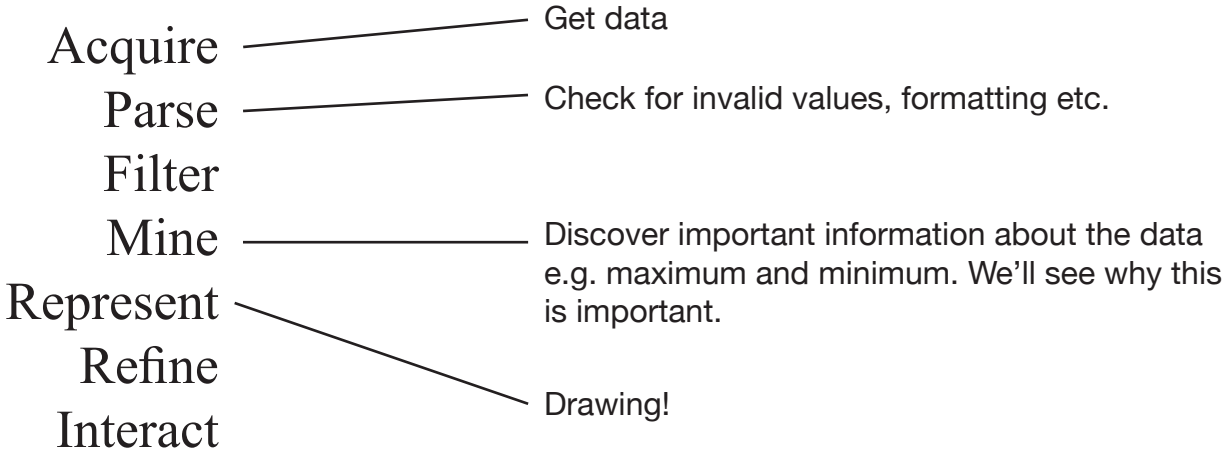
Week 6

Scatterplot Refined

Parsing, Append DOM, Scales

Generalizing the Data Viz Process

Ben Fry's visualization pipeline is a powerful conceptual framework for any data visualization problem.



Quick Review

From .csv to Data to DOM Elements

Tabular Data

Country	GDP	GDP Per Cap	% Internet Users
Afghanistan	20724663537	678.3	5.9
Albania	12903854876	4652.5	60.1
...	...	...	...

CSV

```
country,GDP,GDP_per_capita,internet_users_per_100
Afghanistan,20724663537,678.34,5.9
Albania,12903854876,4652.4,60.1
```

From .csv to Data to DOM Elements

CSV

```
country,GDP,GDP_per_capita,internet_users_per_100
Afghanistan,20724663537,678.3,5.9
Albania,12903854876,4652.4,60.1
```

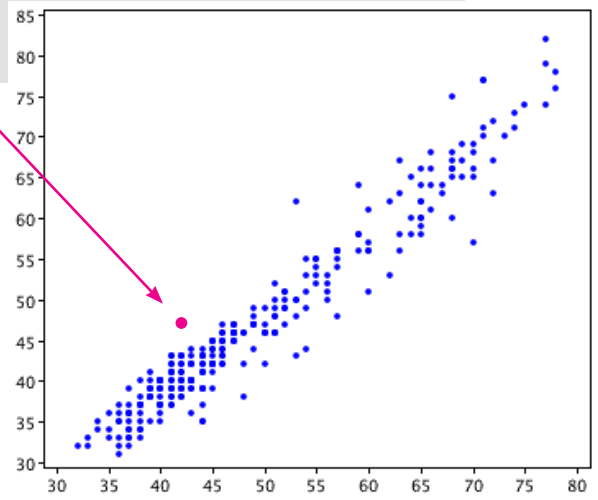
JavaScript array of objects

```
[
     {
          country: “Afghanistan”
          GDP: 20724663537,
          GDP_per_capita: 678.3,
          ...
     },
     {    country: “Albania”, GDP: 12903854876, ...}
]
```

From .csv to Data to DOM Elements

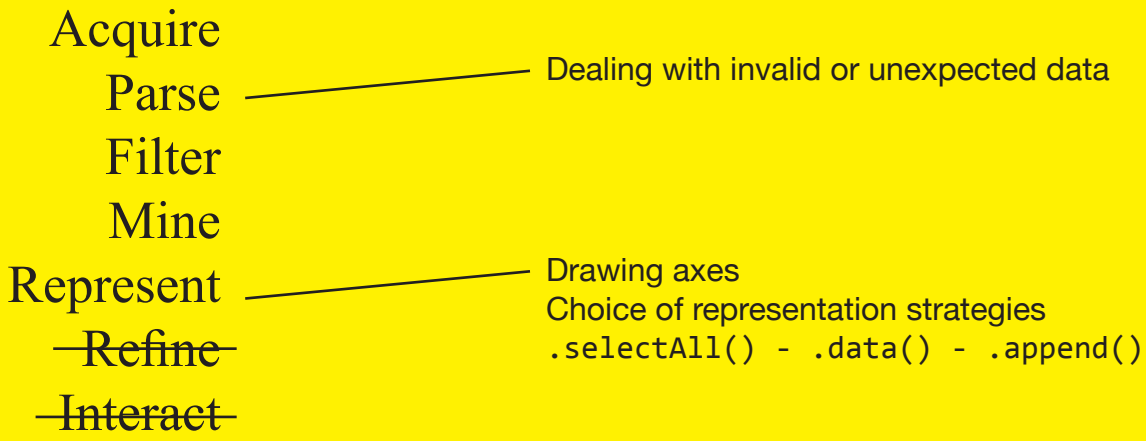
JavaScript array of objects

```
[
    {
        country: “Afghanistan”
        GDP: 20724663537,
        GDP_per_capita: 678.3,
        ...
    },
    {   country: “Albania”, GDP: 12903854876, ...}
]
```



Today's Task

Revisit the scatterplot example, and develop a deeper understanding of critical steps.



(Acquire) Review of Data Import

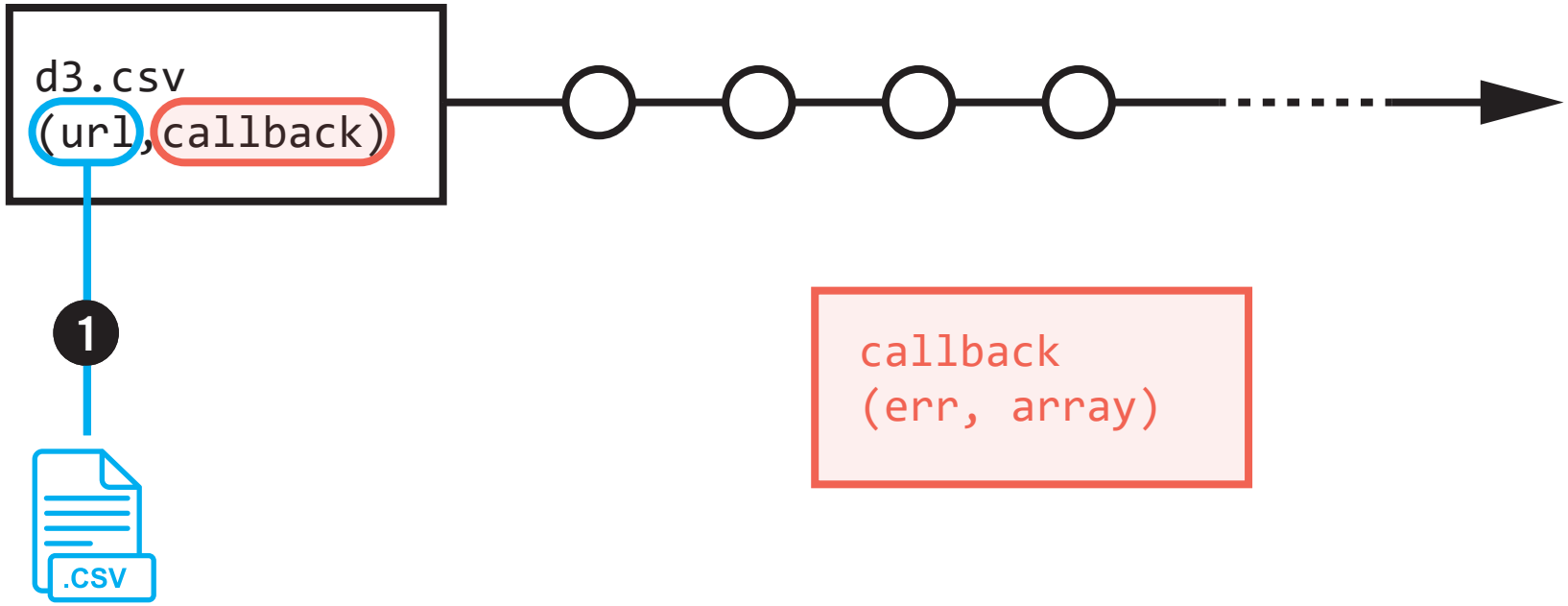
A d3 function allows us to import .csv data as an array of objects, and optionally allows parsing.

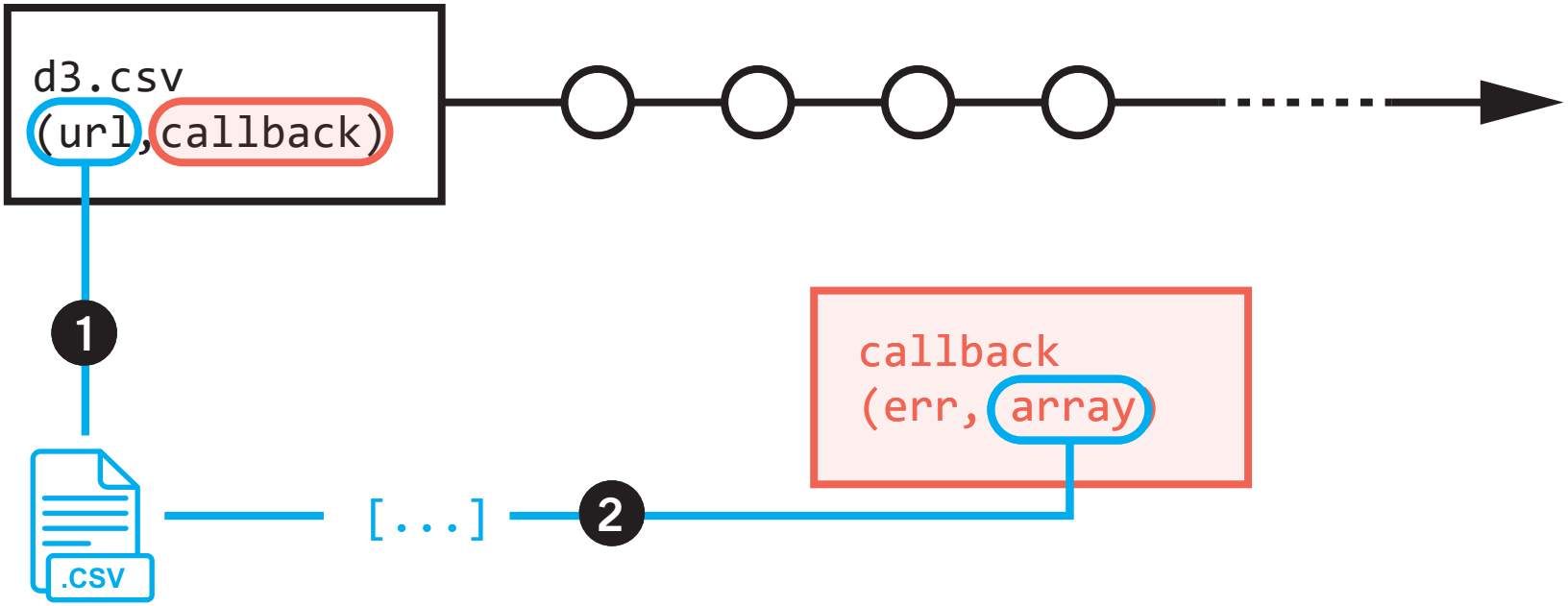
```
d3.csv(url, [accessor,] callback)
```

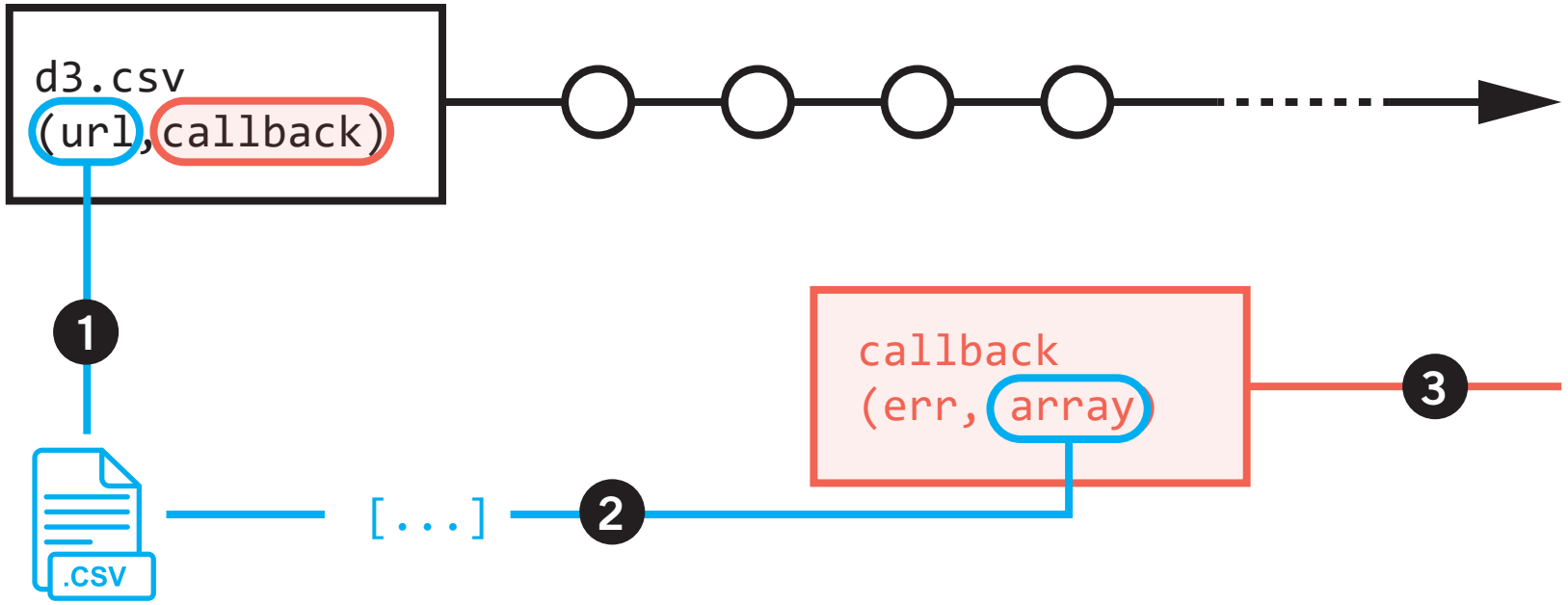
`url` is a **string**, and is the file path of the .csv file.

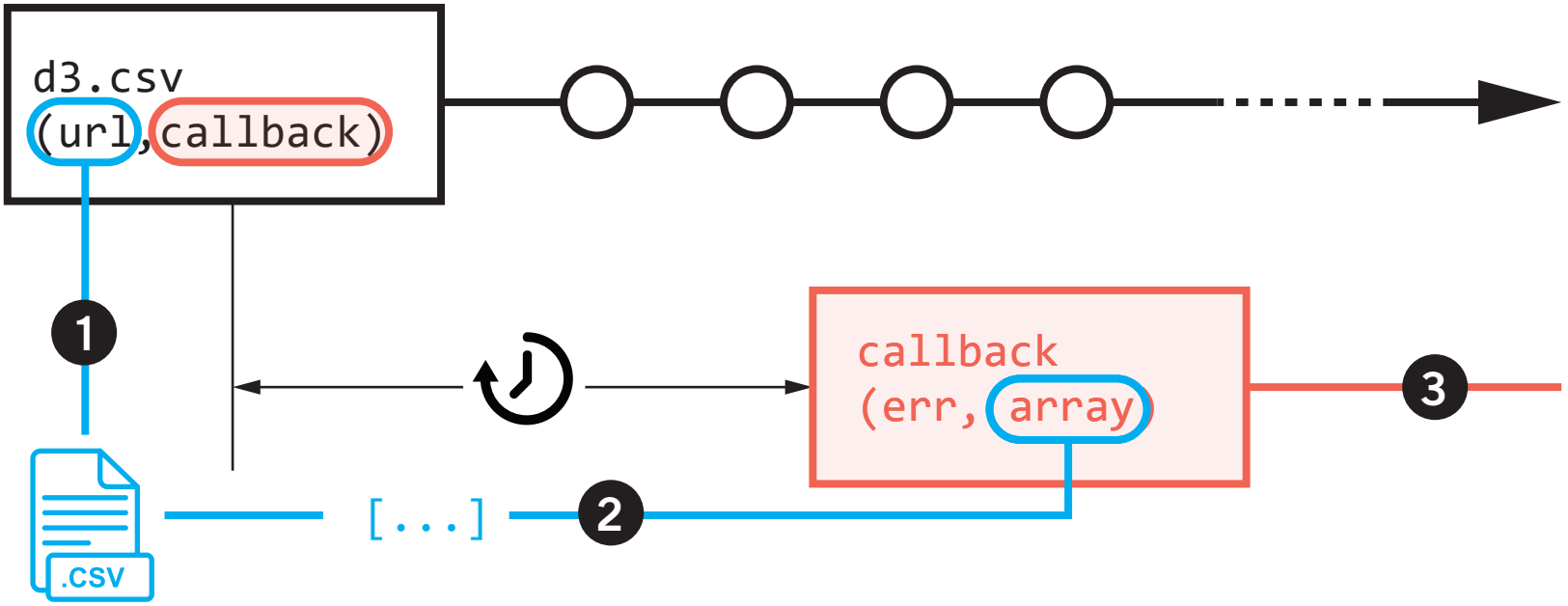
`accessor` is a **function**. It receives 1 argument, which is the pre-parse data of a single row.

`callback` is a **function**. It receives 2 argument. The first is an error object. The second is the array of objects.


d3.csv
(url,callback)
1
.csv
callback
(err, array)


d3.csv
(url,callback)
1
.csv
[...]
2
callback
(err, array)


d3.csv
(url, callback)
1
.csv
[...]
2
callback
(err, array)
3


d3.csv
(url, callback)
1
callback
(err, array)
3
.csv
[...]
2

(Parse) Using the `accessor` Function

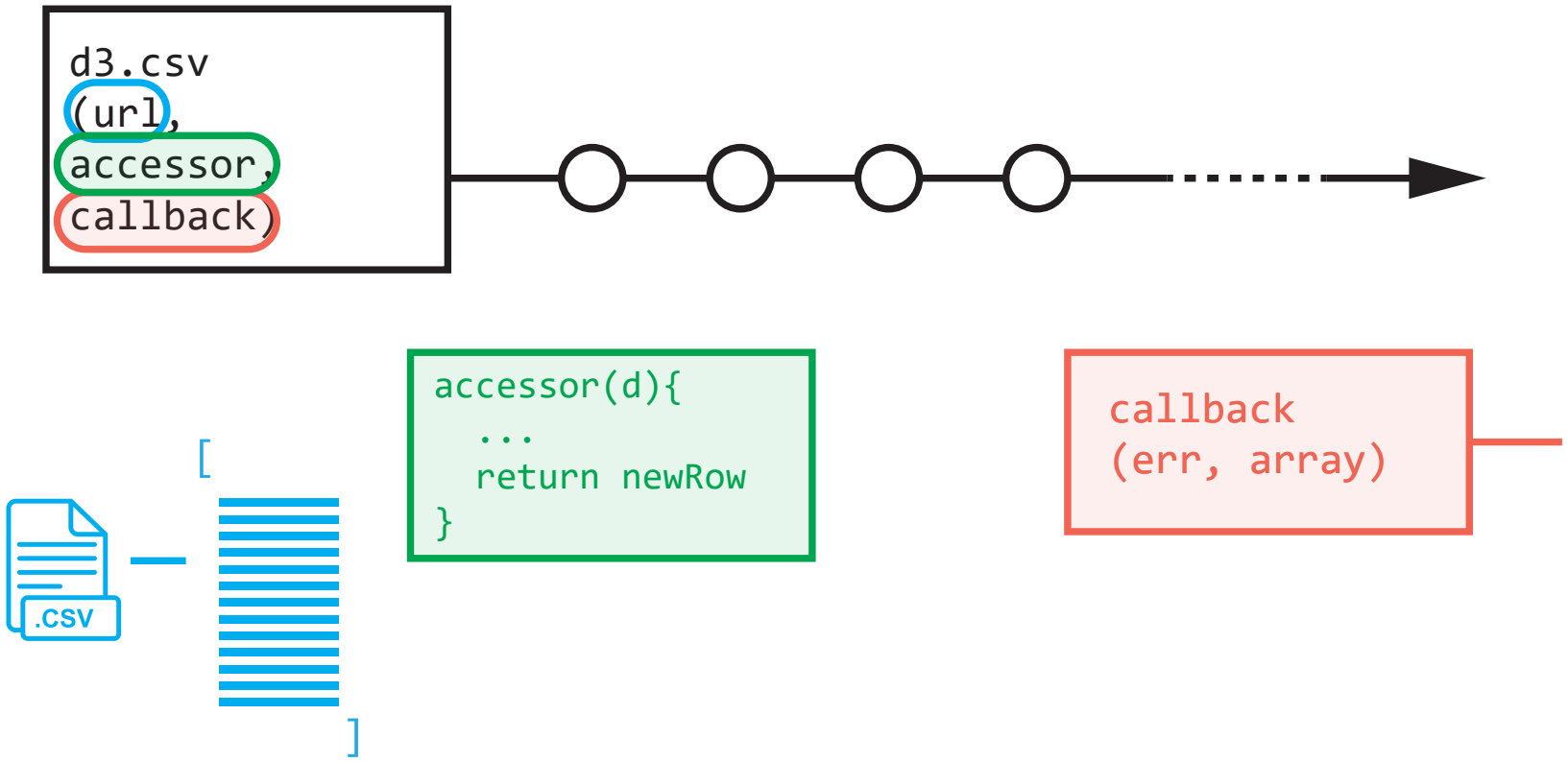
Let's pay particular attention to the `accessor` argument.

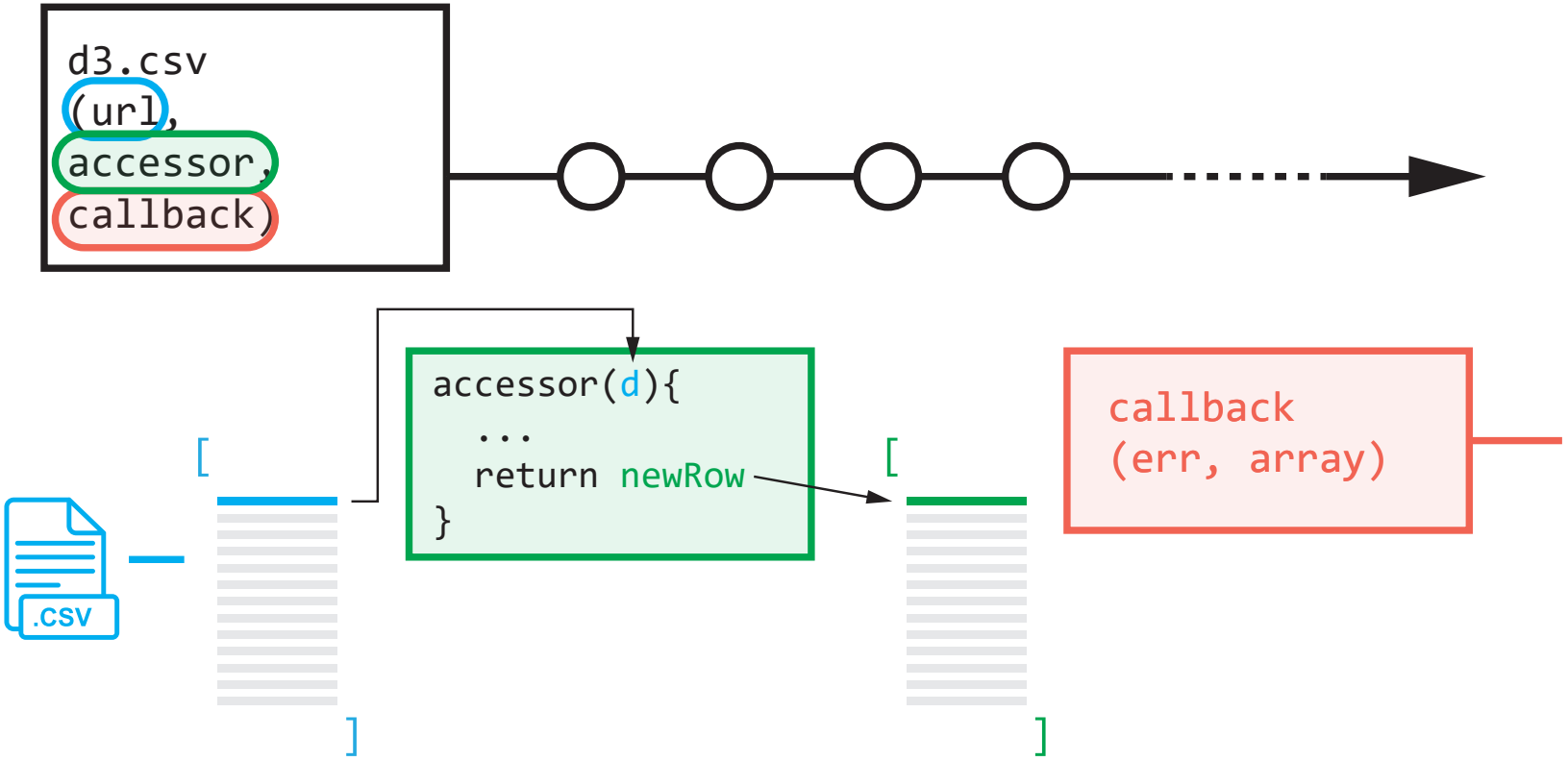
```
d3.csv(url, accessor, callback)
```

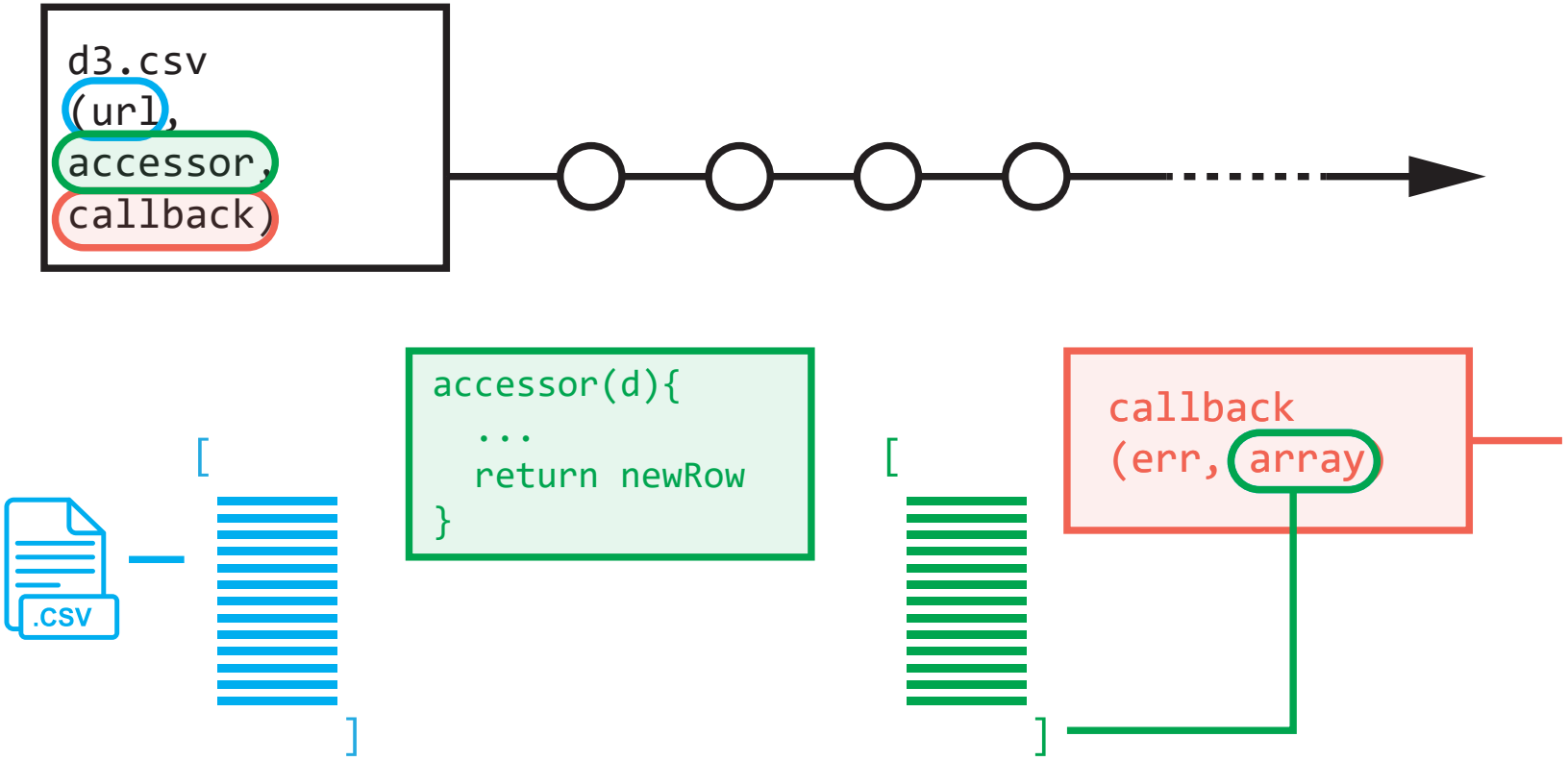
This accessor function runs for every row in the .csv data, parses it, then returns the parsed row.

It receives one argument -- the unparsed row.

It should return an object -- the parsed row.


d3.csv
(url,
accessor,
callback)
accessor(d){
...
return newRow
}
callback
(err, array)
[
.csv
]


d3.csv
(url,
accessor,
callback)
accessor(d){
...
return newRow
}
callback
(err, array)
[
]
[
]
.csv


d3.csv
(url,
accessor,
callback)
accessor(d){
...
return newRow
}
callback
(err, array)
.csv
[
]
[
]

(Parse) Dealing with Space in Objects

```
var someObject = {
 property1: “Hello”,
 “some other prop”: “world”
}

someObject.property1 // “Hello”
someObject[“property1”] //“Hello”
someObject[“some other prop”] //“world”
```

(Parse) What If We Have Unexpected Values?

The ternary operator is a useful shorthand:

```
(boolean value) ? a : b
```

If boolean value is true, return a; otherwise, return b.

```
(88 > 90) ? “Apple” : “Orange”
(88) ? “Apple” : “Orange”
(0) ? “Apple” : “Orange”
(undefined) ? “Apple” : “Orange”
```

(Mine) Mining for max and min

One last issue: I know what my range is (the dimension of my `<div>` element), but what is the domain?

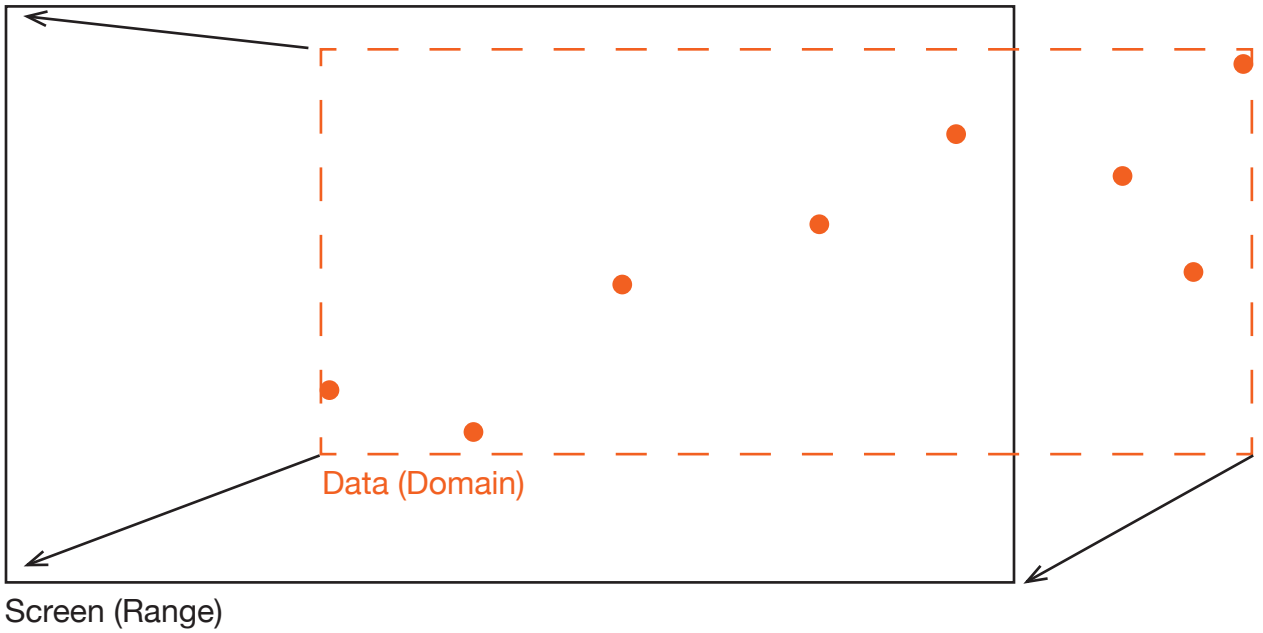
We need to manually find the max and min of the data in both the x- and y-directions! For that, we use:

```
d3.min(array, accessor)
```

Where `accessor` is a function that iterates over the array.

(Represent) Scaling Data

Often, we need to map data from one space to another. This is referred to as mapping from "domain" to "range".



(Represent) Using `d3.scale`

d3 provides a convenience function for us to map data from domain to range: `d3.scale`

First, we need to define a new scale, and define its domain and range:

```
var newScale = d3.scale.linear()
       .domain([100,600])
       .range([0,1200]);
```

This returns a scaling function. Later, we can convert any number from domain space to range space by calling this function.

```
newScale(350); //returns 600
```

(Represent) Using `d3.scale`

What are some other possible scales? When should we consider using them?

(Represent) Representing Chosen Scale with `d3.svg.axis`

What are some other possible scales? When should we consider using them?

The Next Part Is New and Super-Important

How do we draw the circles representing these countries?

Acquire
Parse
~~Filter~~
Mine
Represent
~~Refine~~
~~Interact~~

`.selectAll() - .data() - .enter()`

This is arguably the most important pattern in D3.

It does two things:
1. It creates the corresponding DOM element for each element of data;

2. It "**joins**" that data element to the DOM element, so that there is a one-to-one relationship.

.selectAll() - .data() - .enter()

```
canvas.selectAll(‘.node’)
        .data(rows)
        .enter()
        .append(‘circle’)
        .attr(‘class’,’node’);
```

.selectAll() - .data() - .enter()

```
canvas.selectAll(‘.node’)
        .data(rows)
        .enter()
        .append(‘circle’)
        .attr(‘class’,’node’)
        .attr(‘cx’, function(d){
             //what is “d”?

          });
```

Quick Recap

We revisited the scatterplot problem through the lens of the visualization pipelines

	Concept	Implementation and Technique
Acquire	`d3.csv()`	
Parse	How to use accessor function	Dealing with invalid values (ternary operator) Dealing with spaces in object notation
~~Filter~~		
Mine		`d3.max(), d3.min(), d3.range()` Construct corresponding axes with `d3.svg.axis()`
Represent	Using visual variables to represent data dimensions **!!! Joining data to DOM elements !!!**	
~~Refine~~		
~~Interact~~		