Week 10

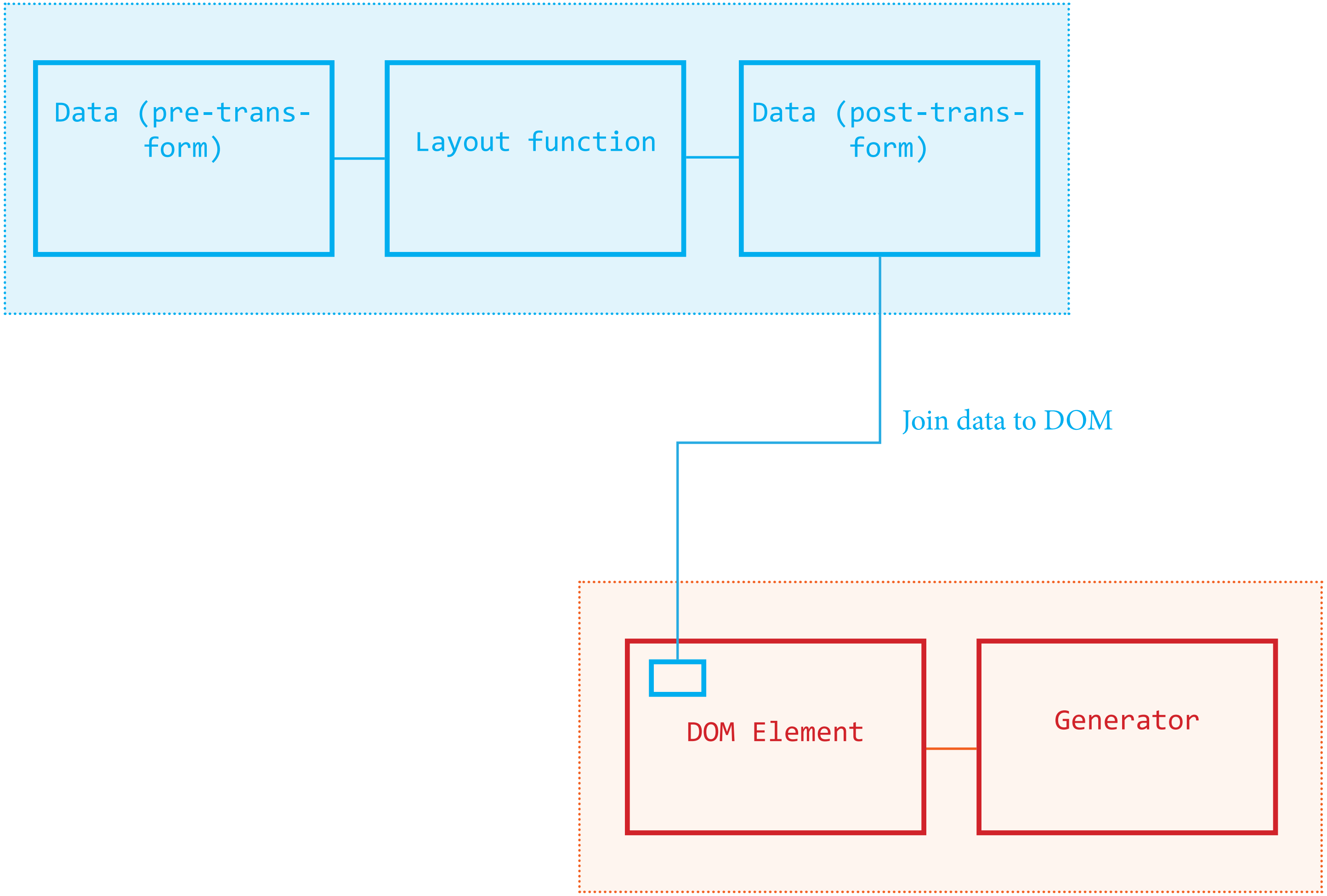
SPATIAL REPRESENTATION

WHAT ARE WE TRYING TO DO?

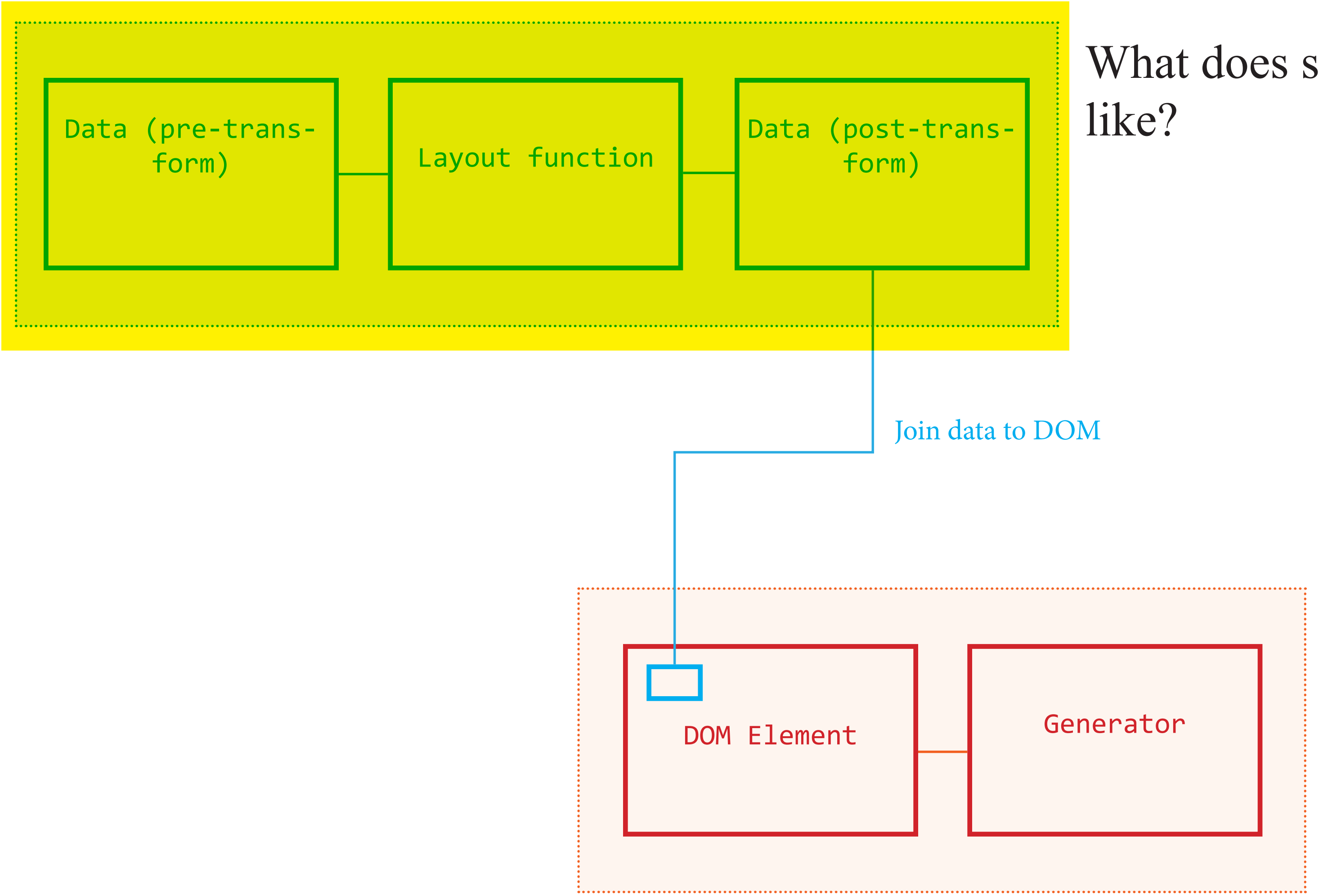
“Mapping” is a huge and vague topic. In this class, we’ll focus on building a couple of key capabilities:

- Represent geographic features (points, lines, and polygon features) visually;
- Integrate thematic data into geographic representation i.e. **thematic mapping**;
- Alternative spatial representations, such as cartograms.

CONCEPTUALLY...



SPATIAL DATA



What does spatial data look like?

SPATIAL DATA

Spatial data comes in very specific formats:

Shapefiles (.shp)
KML
GeoJSON (.json)

The .json Format

You are actually already very familiar with .json data, which is an open-standard format that transmits data objects using **attribute-value pairs**.

```
{
  class: “ARTG5330”,
  graduateLevel: true,
  numStudents: 8,
  students: [
     {name: “Lia Petronio”, id:2334233},
     {name: “Ashley Treni”, id:3433322},
     ...
  ],
  instructor: {
     name: “Siqi Zhu”,
     id: 4333444,
     courses:[“ARTG5330”]
  }
}
```

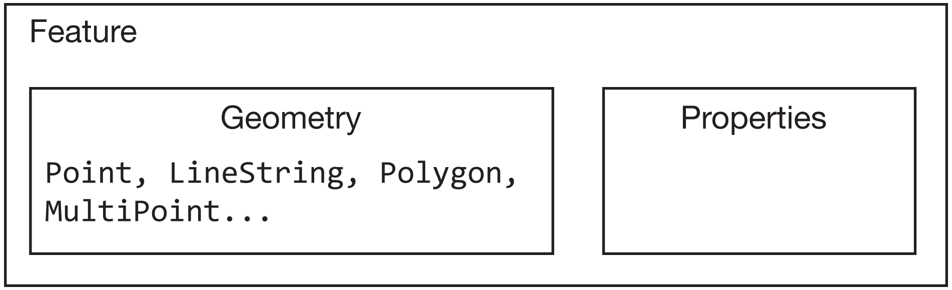
The .json Format

You are actually already very familiar with .json data, which is an open-standard format that transmits data objects using **attribute-value pairs**.

```
{ attribute   value
  class: "ARTG5330",comma separation btw pairs
  graduateLevel: true,
  numStudents: 8,
  students: [
     {name: "Lia Petronio", id:2334233},
     {name: "Ashley Treni", id:3433322},
     ...
  ],
  instructor: {
     name: "Siqi Zhu",
     id: 4333444,
     courses:["ARTG5330"]
  }
}
```

GeoJSON Is a Subset of .json

GeoJSON data is a subset of JSON, with attributes that specifically describe geometries and their properties.



Feature

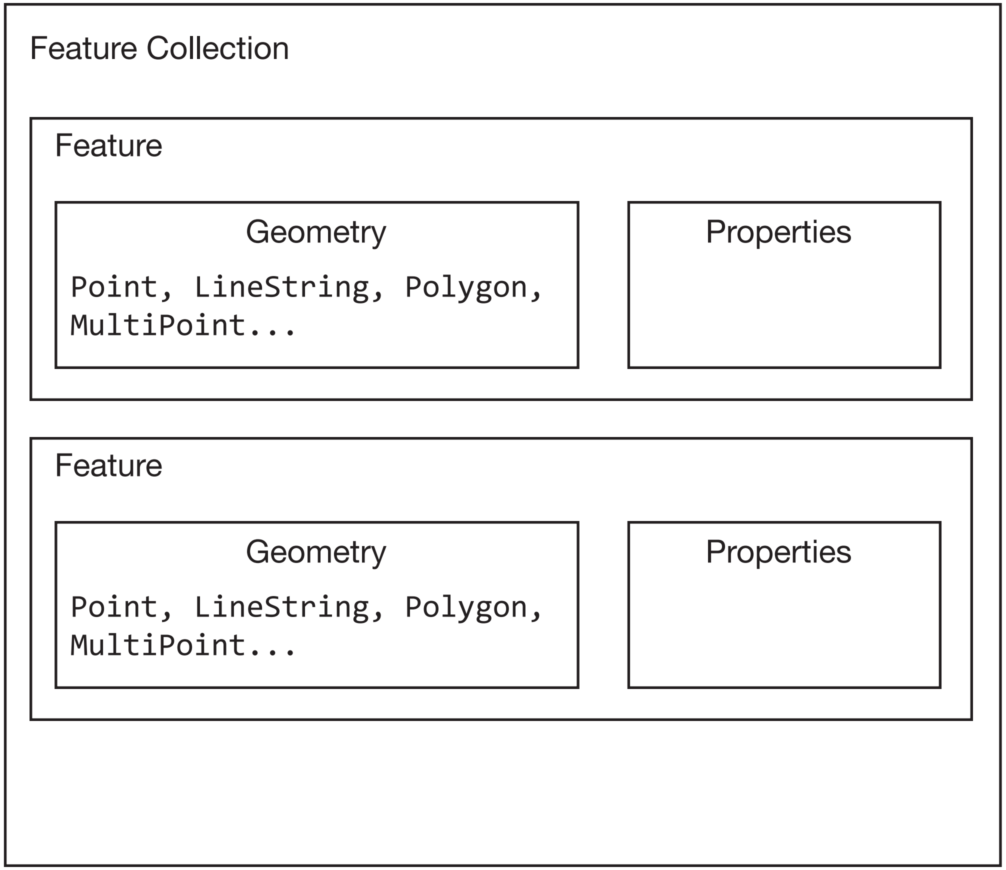
Geometry

`Point, LineString, Polygon, MultiPoint...`

Properties

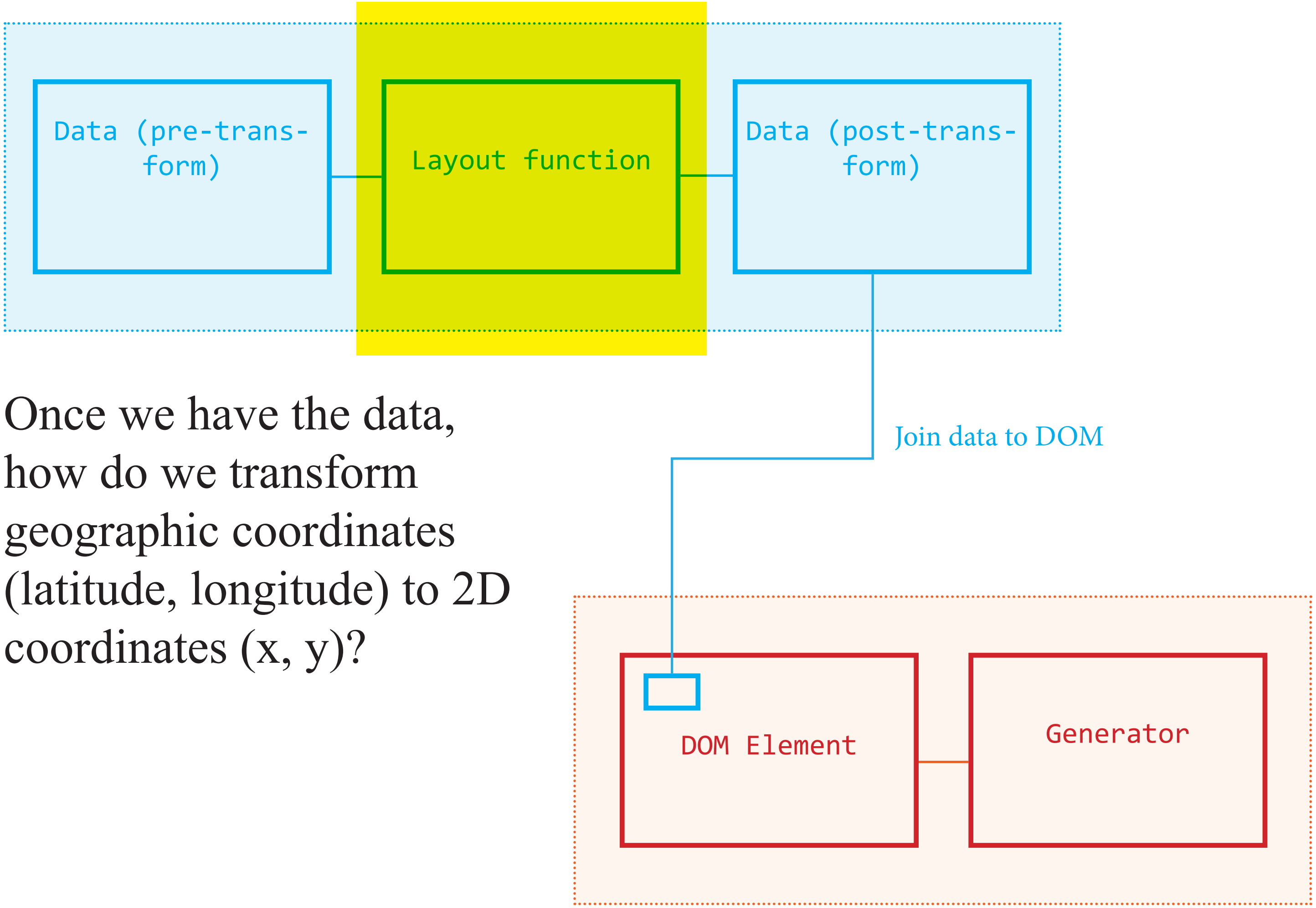
GeoJSON Is a Subset of .json

GeoJSON data is a subset of JSON, with attributes that specifically describe geometries and their properties.


Feature Collection
Feature
Geometry
Point, LineString, Polygon, MultiPoint...
Properties
Feature
Geometry
Point, LineString, Polygon, MultiPoint...
Properties

```
{ “type”: “FeatureCollection”,
    “features”: [
      { “type”: “Feature”,
        “geometry”: {“type”: “Point”, “coordinates”: [102.0, 0.5]},
        “properties”: {“prop0”: “value0”}
      },
      { “type”: “Feature”,
        “geometry”: {
          “type”: “LineString”,
          “coordinates”: [[102.0, 0.0], [103.0, 1.0], [104.0, 0.0], [105.0, 1.0]]
          },
        “properties”: {
          “prop1”: 0.0
          }
      },
      { “type”: “Feature”,
         “geometry”: {
           “type”: “Polygon”,
           “coordinates”: [
             [ [100.0, 0.0], [101.0, 0.0], [101.0, 1.0],
               [100.0, 1.0], [100.0, 0.0] ] ]
         },
         “properties”: {
           “prop1”: {“this”: “that”}
           }
         }
```

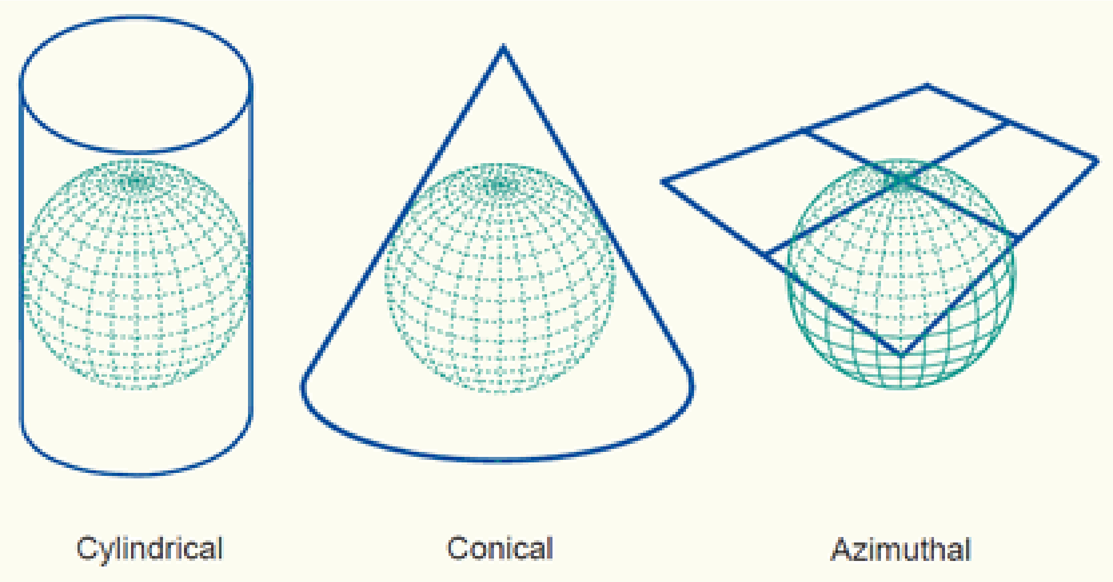
From Data to x-y Coordinates



Once we have the data, how do we transform geographic coordinates (latitude, longitude) to 2D coordinates (x, y)?

From Data to x-y Coordinates

Not as simple as you think!



From Data to x-y Coordinates

Map projection is the process whereby **longitude, latitude coordinates** on the surface of sphere are transformed into **cartesian coordinates** on a plane.

Conceptually, map projection should be a function, where

```
x-y coordinates = projectionFunction([longitude,
latitude])
```

PROJECTION IN d3

`d3.geo.projection()` constructs a new projection function, for which you can specify a number of key attributes

```
var projectionFunction = d3.geo.projection()
    .center([lng, lat]) //0,0 by default
    .translate([x, y])
    .scale(); //150 by default

//screen coordinates to geographic coordinates
projectionFunction.invert([100,100]);

//geographic to screen coordinates
projectionFunction([-120,42]);
```

PROJECTION IN d3

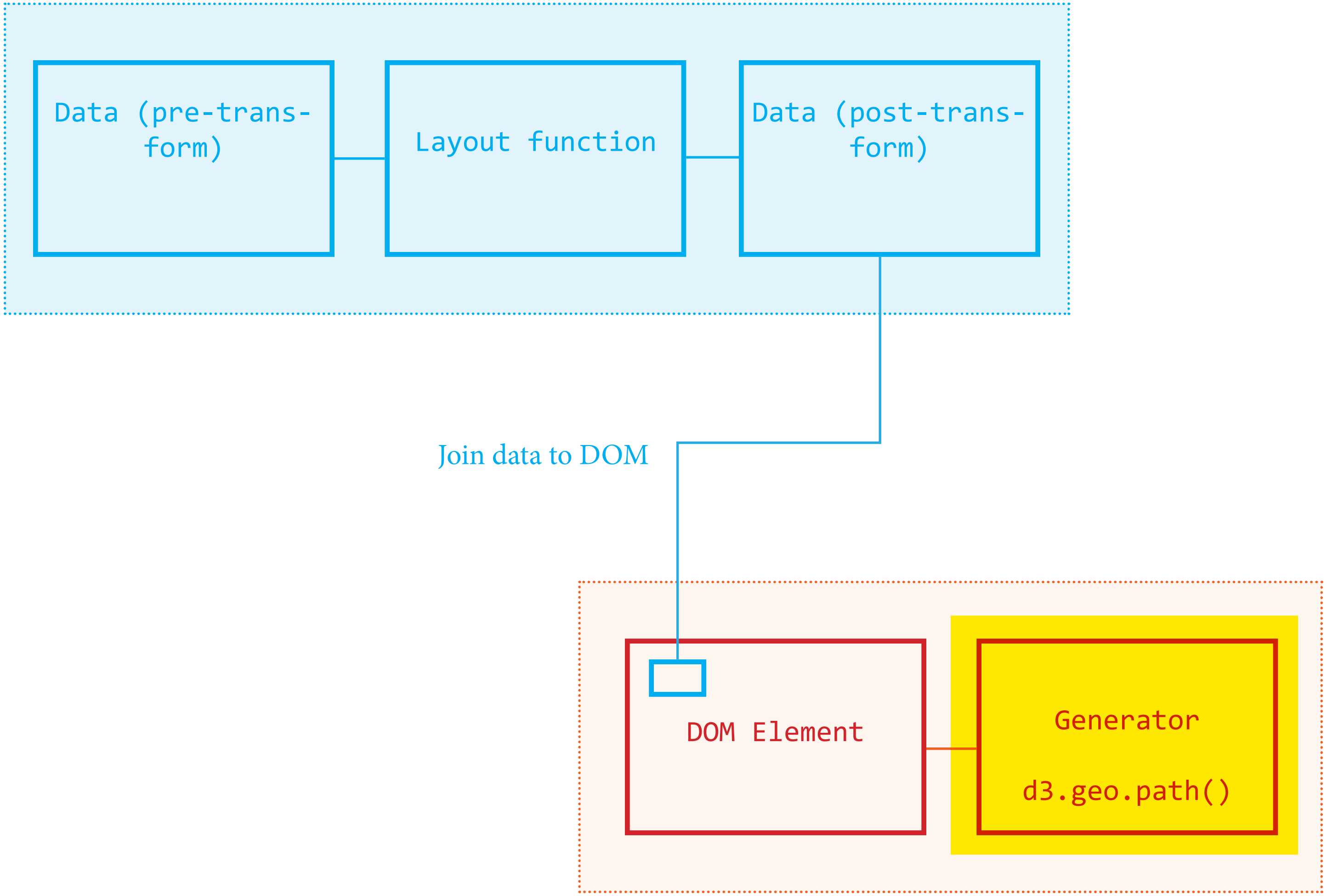
d3 has some pre-built projection functions that we can use off the shelf:

```
d3.geo.albers()
d3.geo.albersUsa()
```

...

https://github.com/mbostock/d3/wiki/Geo-Projections

GENERATING SVG



`d3.geo.path()`

Similar to other SVG generator functions, like `d3.svg.line()`, `d3.geo.path()` takes data and generates path attributes for SVG paths.

`d3.geo.path()` tightly interfaces with GeoJSON.

`d3.geo.path()` **depends on a projection function.**

d3.geo.path()

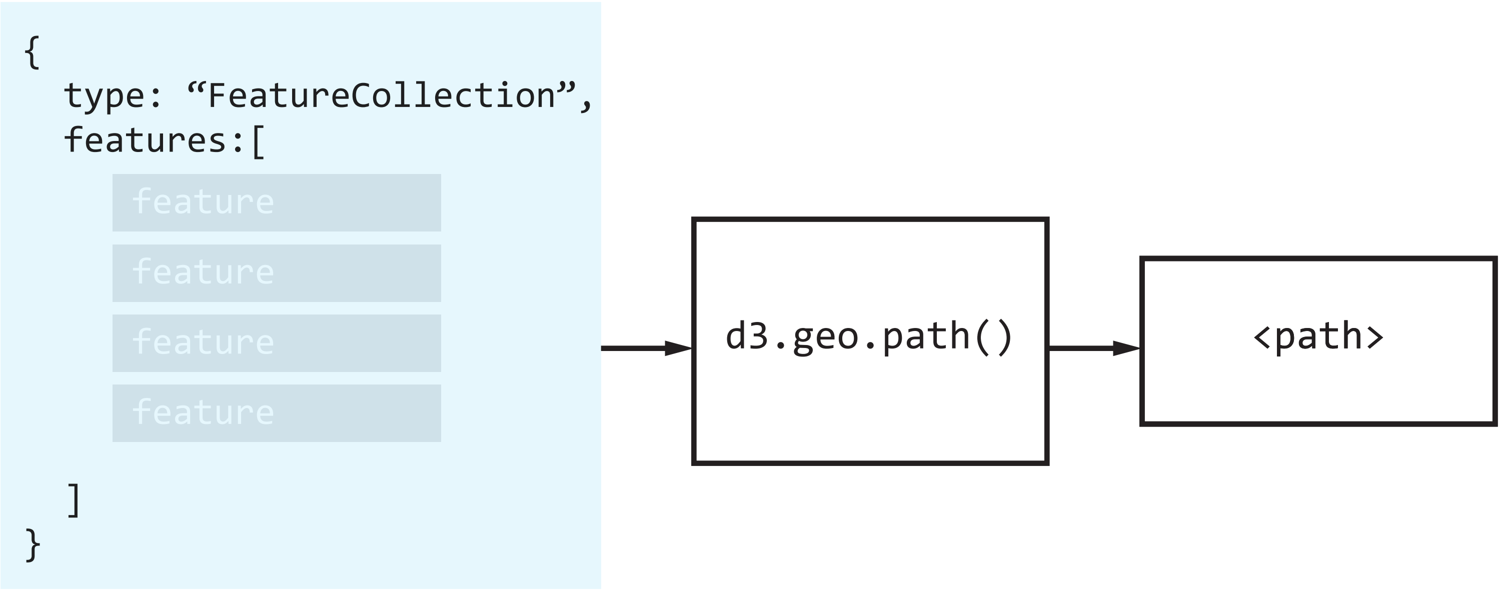
```
var projectionFunc = ... //some projection function

var geopath = d3.geo.path()
     .projection(projectionFunc);

...

svg.selectAll(‘.country’)
     .data(...)
     .enter()
     .append(‘path’)
     .attr(‘class’, ‘country’)
     .attr(‘d’, geopath);
```

LET’S DRAW A MAP OF THE US!

d3.geo.path()

Path generator functions work with both the entire feature collection and individual features.

Feature Collection



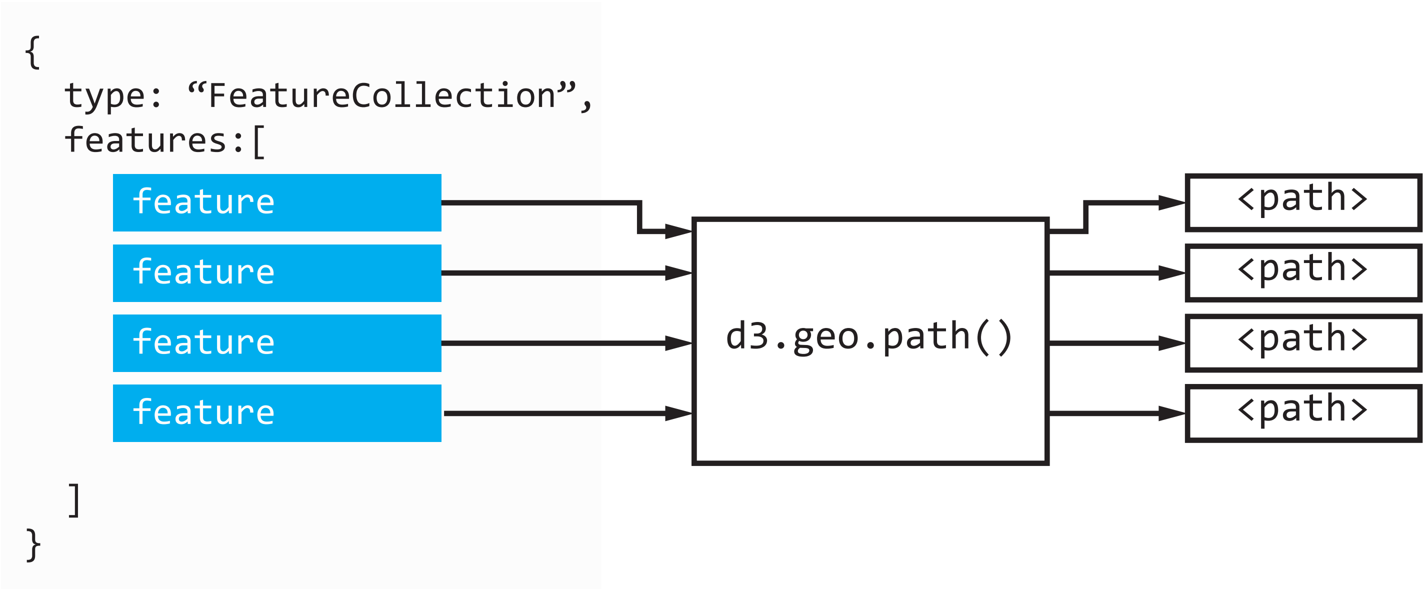
```
{
  type: “FeatureCollection”,
  features:[
      feature
      feature
      feature
      feature

  ]
}
```

d3.geo.path()

<path>

d3.geo.path()

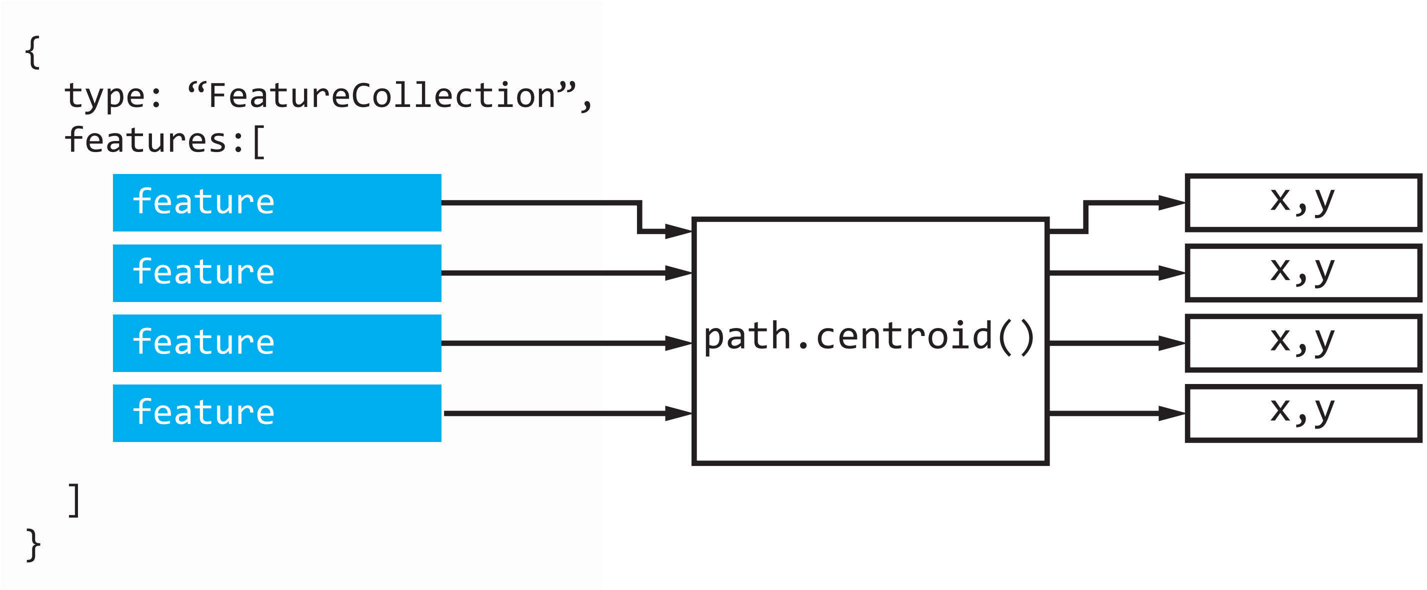
Path generator functions work with both the entire feature collection and individual features.



```
{
  type: “FeatureCollection”,
  features:[
     feature
     feature
     feature
     feature

  ]
}
```

d3.geo.path()

Path generator functions work with both the entire feature collection and individual features.



```
{
  type: “FeatureCollection”,
  features:[
```

```
  ]
}
```

WHERE TO FIND GEOSPATIAL DATA?

For U.S. administrative boundaries:
https://www.census.gov/geo/maps-data/data/tiger.html

For open-source world shapefiles:
http://www.naturalearthdata.com/

Open Street Maps

This tool converts .shp files to GeoJSON format:
http://www.gdal.org/ogr2ogr.html