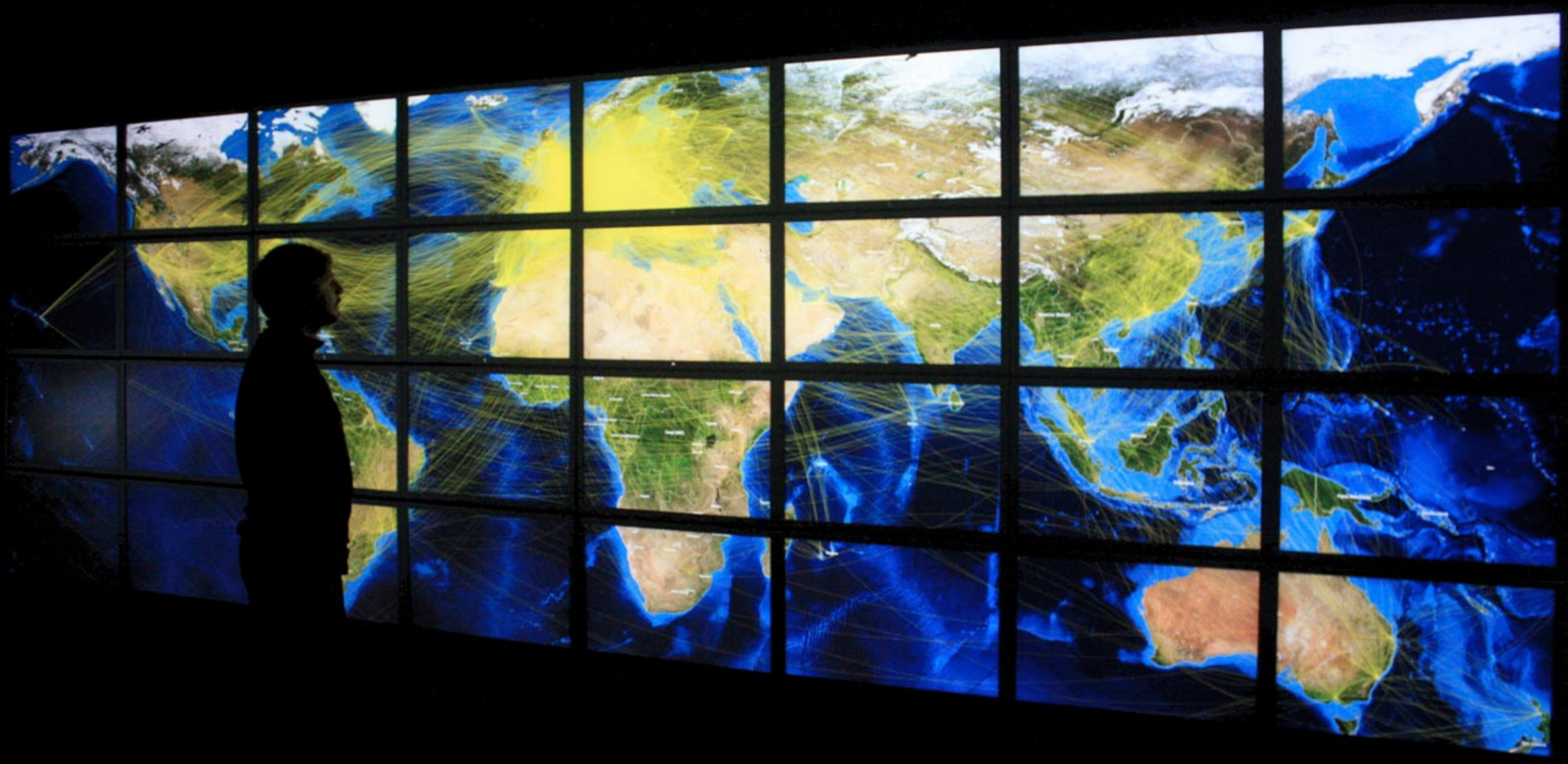
Data Visualization

MIDO - Data Visualization - Session 04 - exercices
Visualizing time-series with Vega-lite and D3



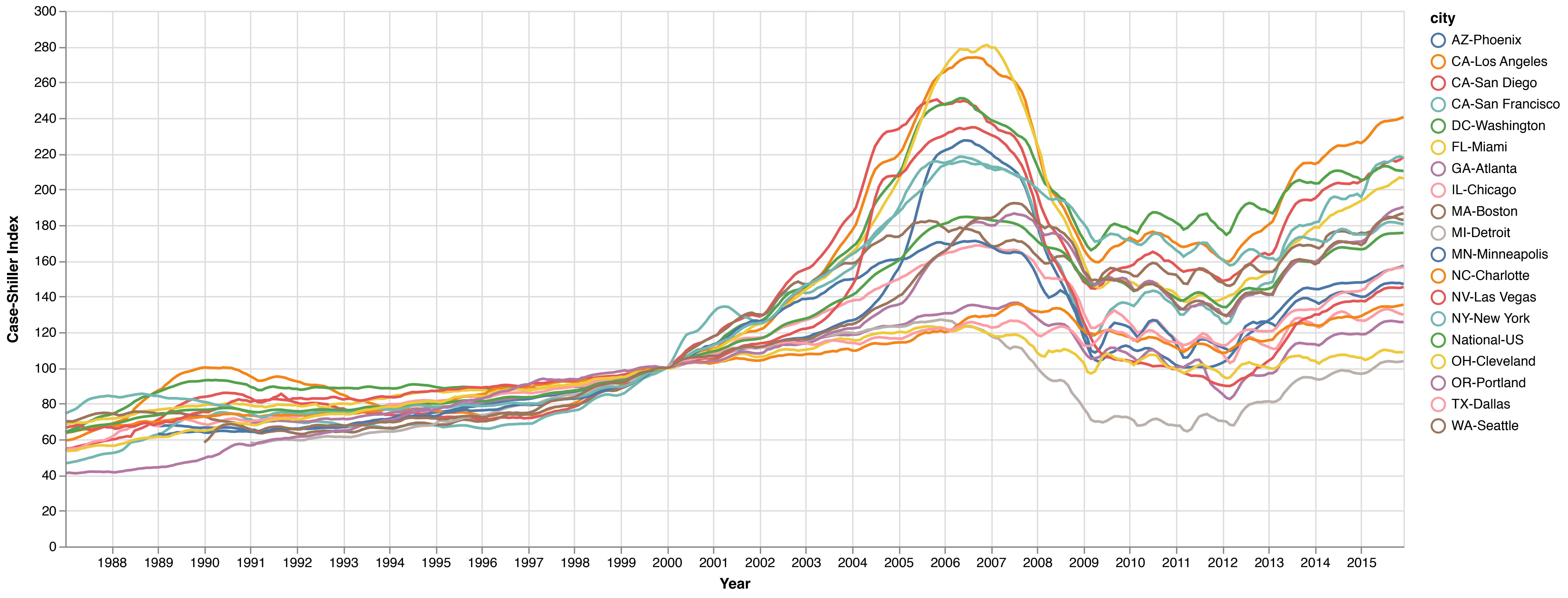
emmanuel.pietriga@inria.fr

Exercise s06a

- Visualization of the evolution of home price indices in the USA, for different metropolitan areas. Based on the Case–Shiller index.

  `https://en.wikipedia.org/wiki/Case%E2%80%93Shiller_index`

- We start with a basic Vega-lite line plot:



Exercise s06a

- First, we need to transform the data. It is not easily visualized the way it is structured in the JSON file:

```
[{“AZ-Phoenix": null, "CA-Los Angeles": 59.33, "CA-San Diego": 54.67, "CA-San Francisco": 46.61, "CO-
Denver": 50.2, "Composite-10": 62.82, "Composite-20": null, "DC-Washington": 64.11,  "FL-Miami": 68.5,
"FL-Tampa": 77.33, "GA-Atlanta": null, "IL-Chicago": 53.55,
    "MA-Boston": 70.04, "MI-Detroit": null, "MN-Minneapolis": null, "NC-Charlotte": 63.39, "NV-Las
Vegas": 66.36, "NY-New York": 74.42, "National-US": 63.75, "OH-Cleveland": 53.53, "OR-Portland": 41.05,
"TX-Dallas": null, "WA-Seattle": null, "Date": "1987-01-01"},

 {"AZ-Phoenix": null, "CA-Los Angeles": 59.65, "CA-San Diego": 54.89, "CA-San Francisco": 46.87, "CO-
Denver": 49.96, "Composite-10": 63.39, "Composite-20": null, "DC-Washington": 64.77, "FL-Miami": 68.76,
"FL-Tampa": 77.93, "GA-Atlanta": null, "IL-Chicago": 54.64,
    "MA-Boston": 70.08, "MI-Detroit": null, "MN-Minneapolis": null, "NC-Charlotte": 63.94, "NV-Las
Vegas": 67.03, "NY-New York": 75.43, "National-US": 64.15, "OH-Cleveland": 53.5, "OR-Portland": 41.28,
"TX-Dallas": null, "WA-Seattle": null, "Date": "1987-02-01"},


 …
]
```

- Use a Vega-lite `fold` transform to restructure the data. What we want, basically, is to:

Exercise s06a

- … move from this:

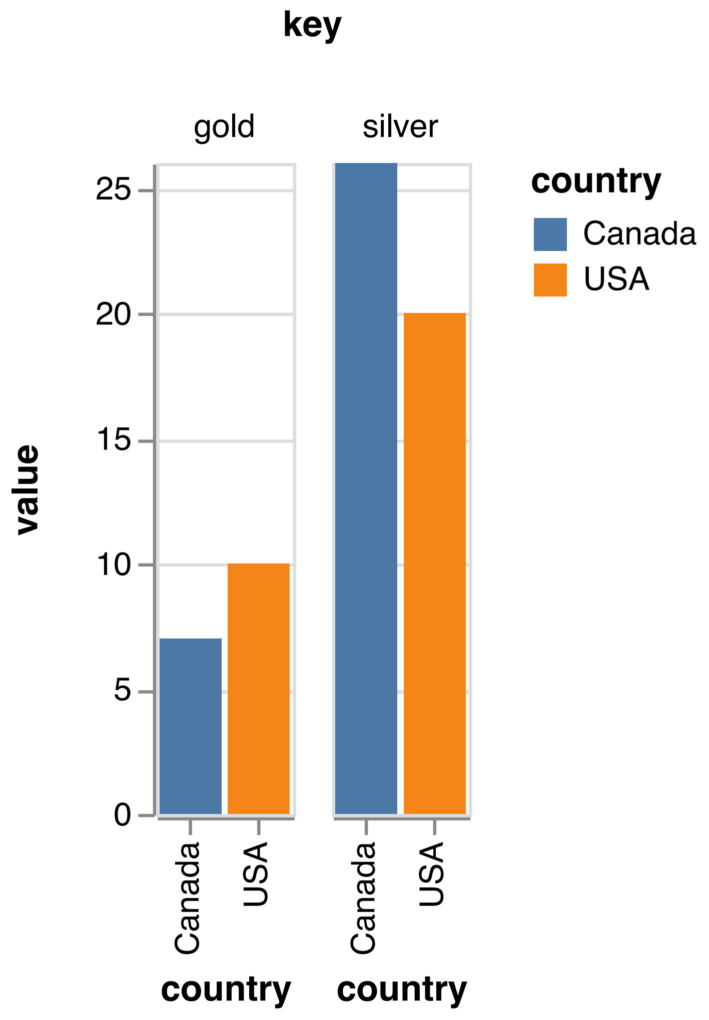
```
[{"AZ-Phoenix": null, "CA-Los Angeles": 59.33, ..., "Date": "1987-01-01"},
 {"AZ-Phoenix": null, "CA-Los Angeles": 59.65, ..., "Date": "1987-02-01"},
 ...,
]
```

- to this:

```
[{"city": "AZ-Phoenix", "cs_index": null, ..., "Date": "1987-01-01"},
 {"city": "CA-Los Angeles", "cs_index": 59.33, ..., "Date": "1987-01-01"},
 ...,
 {"city": "AZ-Phoenix", "cs_index": null, ..., "Date": "1987-02-01"},
 {"city": "CA-Los Angeles", “cs_index": 59.65, ..., "Date": "1987-02-01"},
 ...,
]
```

Exercise s06a

```
{
  "$schema": "https://vega.github.io/schema/vega-lite/v3.json",
  "data": {
    "values": [
      {"country": "USA", "gold": 10, "silver": 20},
      {"country": "Canada", "gold": 7, "silver": 26}
    ]
  },
  "transform": [{"fold": ["gold", "silver"], "as": ["key", "value"]}],
  "mark": "bar",
  "encoding": {
    "column": {"field": "key", "type": "nominal"},
    "x": {"field": "country", "type": "nominal"},
    "y": {"field": "value", "type": "quantitative"},
    "color": {"field": "country", "type": "nominal"}
  },
  "config": {"axisY": {"minExtent": 30}}
}
```

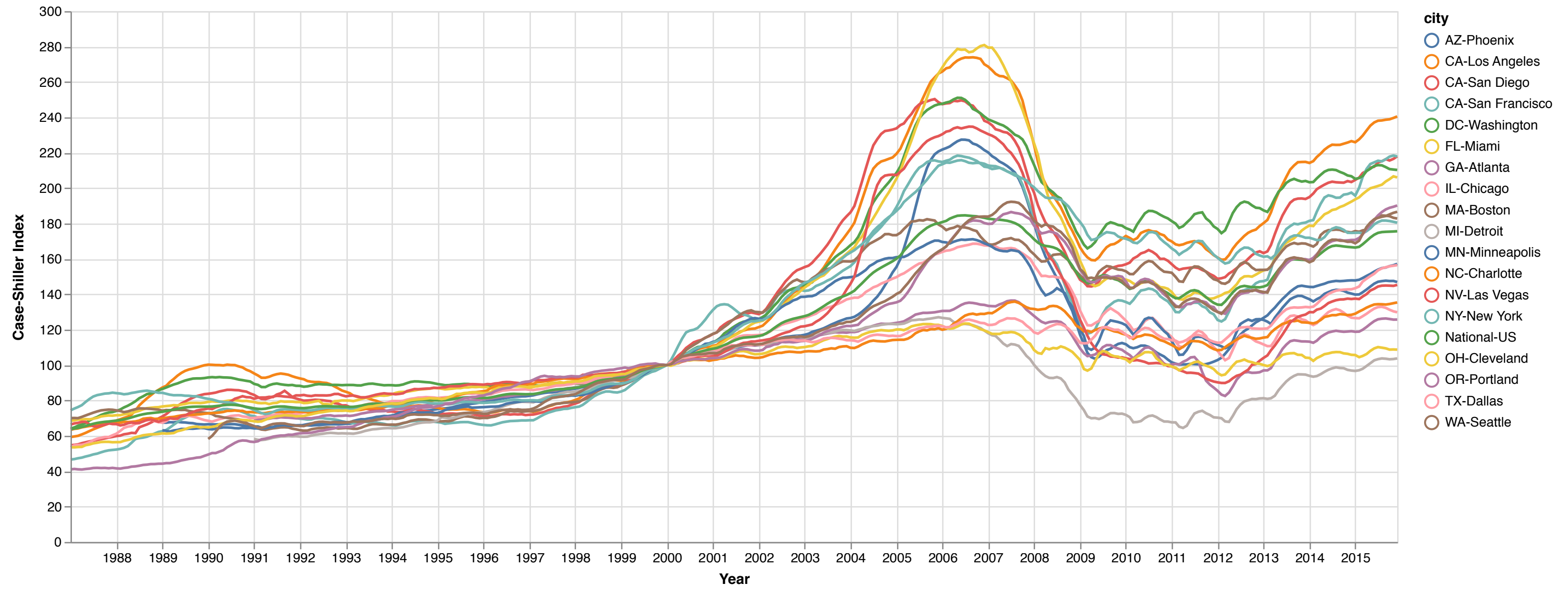


```
[
  {"country": "USA", "gold": 10, "silver": 20},
  {"country": "Canada", "gold": 7, "silver": 26}
]
```

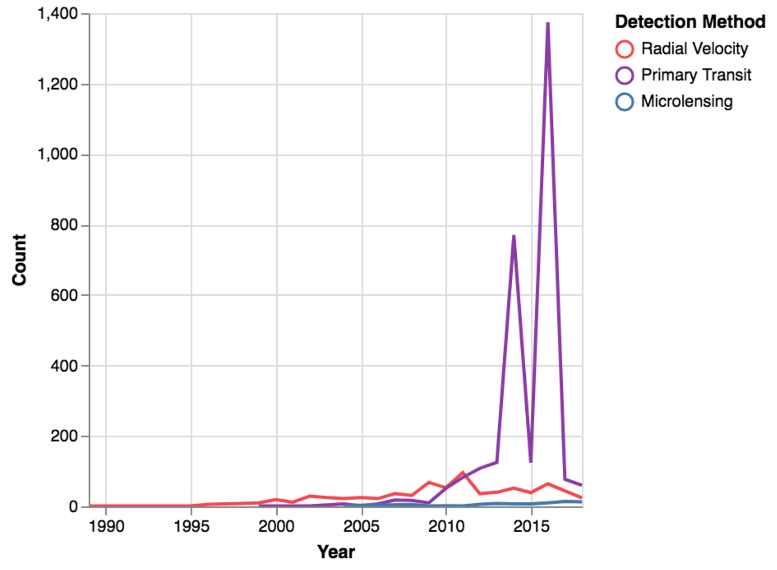
```
[
  {"key": "gold", "value": 10, "country": "USA", "gold": 10, "silver": 20},
  {"key": "silver", "value": 20, "country": "USA", "gold": 10, "silver": 20},
  {"key": "gold", "value": 7, "country": "Canada", "gold": 7, "silver": 26},
  {"key": "silver", "value": 26, "country": "Canada", "gold": 7, "silver": 26}
]
```

Exercise s06a

Based on the newly-created **`fold`** fields, create the following plot:

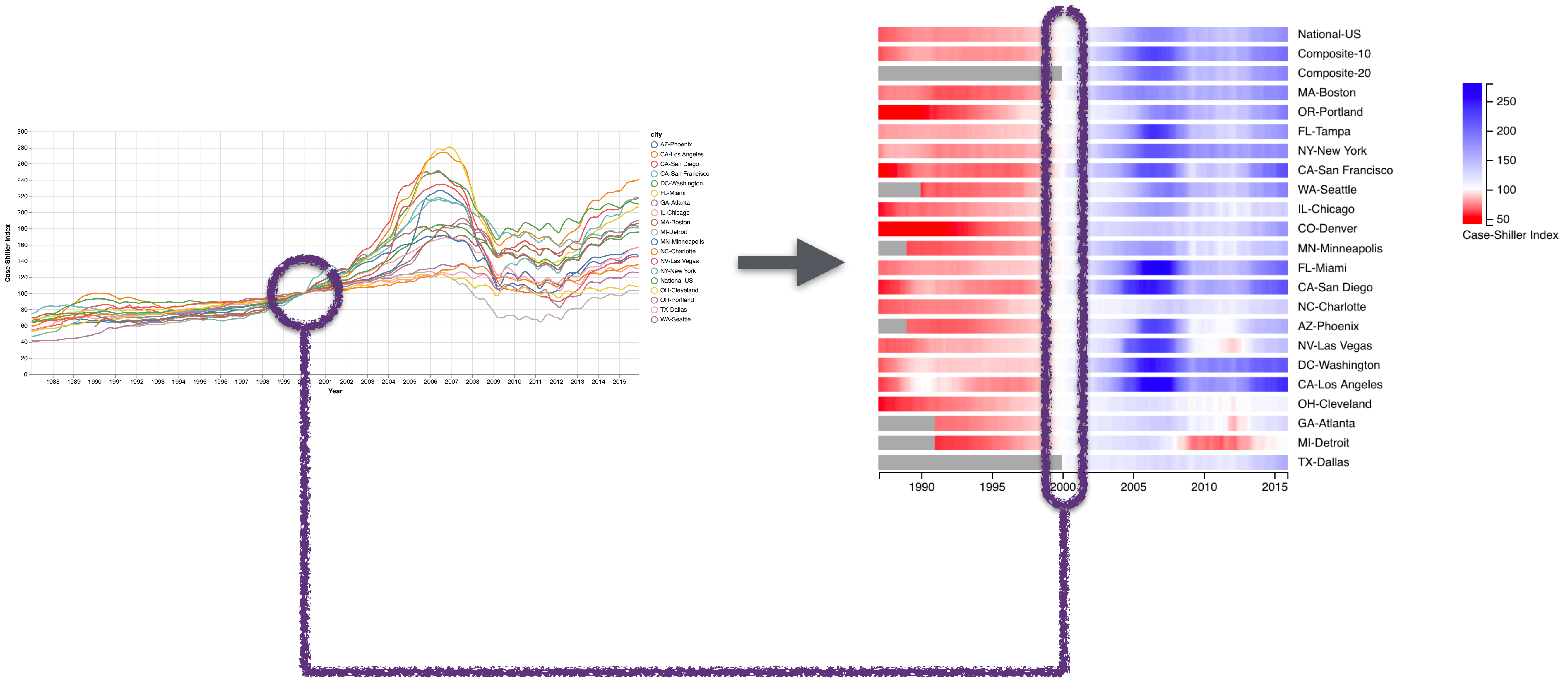


Take inspiration from the previous TD, in which we did something very similar, plotting the number of exoplanets discovered over time by different detection methods as one of the visualizations:



Exercise s06b

- We now switch to an alternative representation of the same data, this time using D3: we use a color map encoding, based on a diverging color scale.



base index (2000)

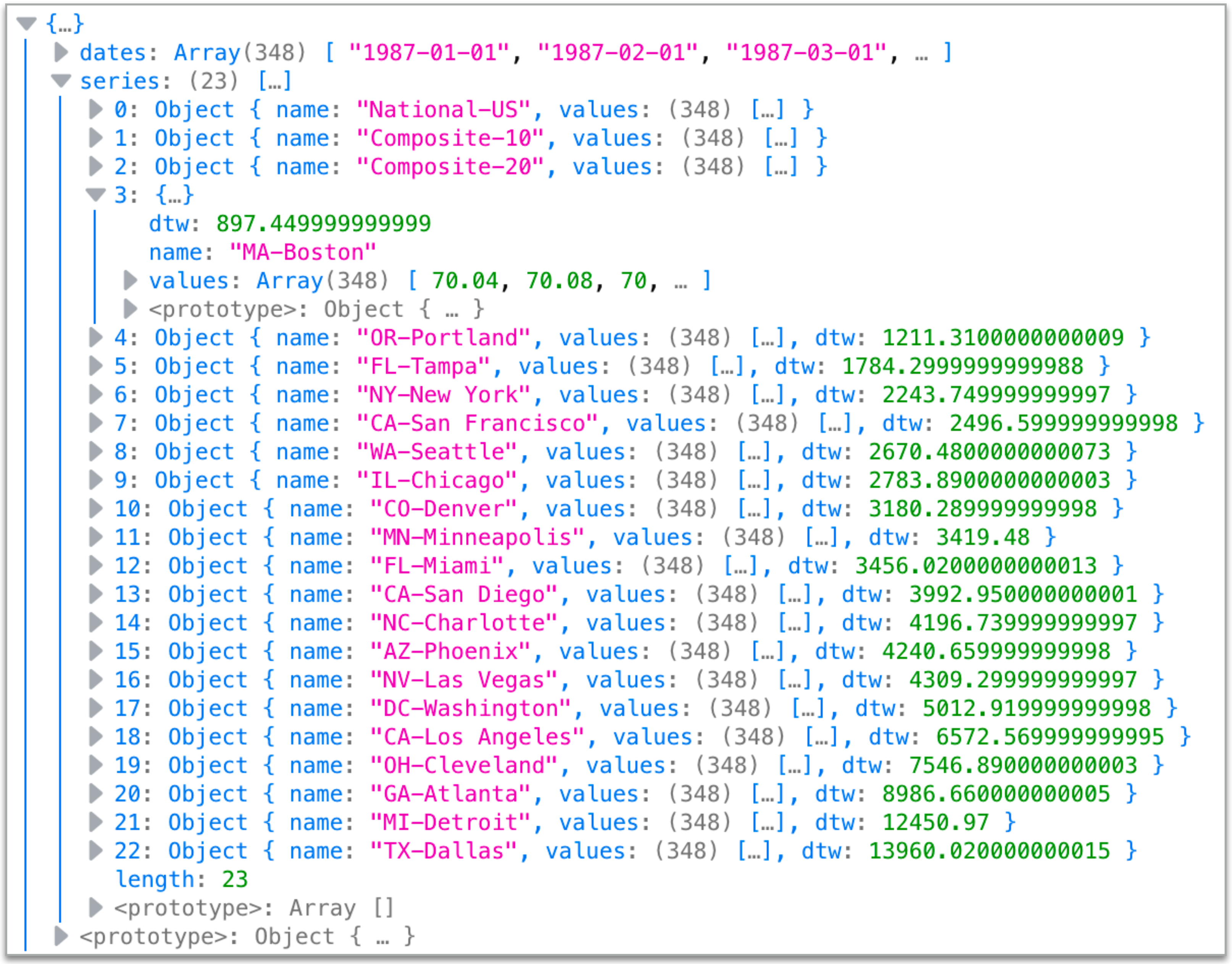
Exercise s06b

- Load the data with `d3.json`, and restructure it to get:

dates: an array of all Date attributes

series: an array of objects as follows:

- one object for each city/group
- `name`: name of the city/group
- `values`: time-series for that city/group
  (tip: populate arrays of values for each city, all stored in a hashmap, as you iterate through input data items)
- `dtw`: see next slide

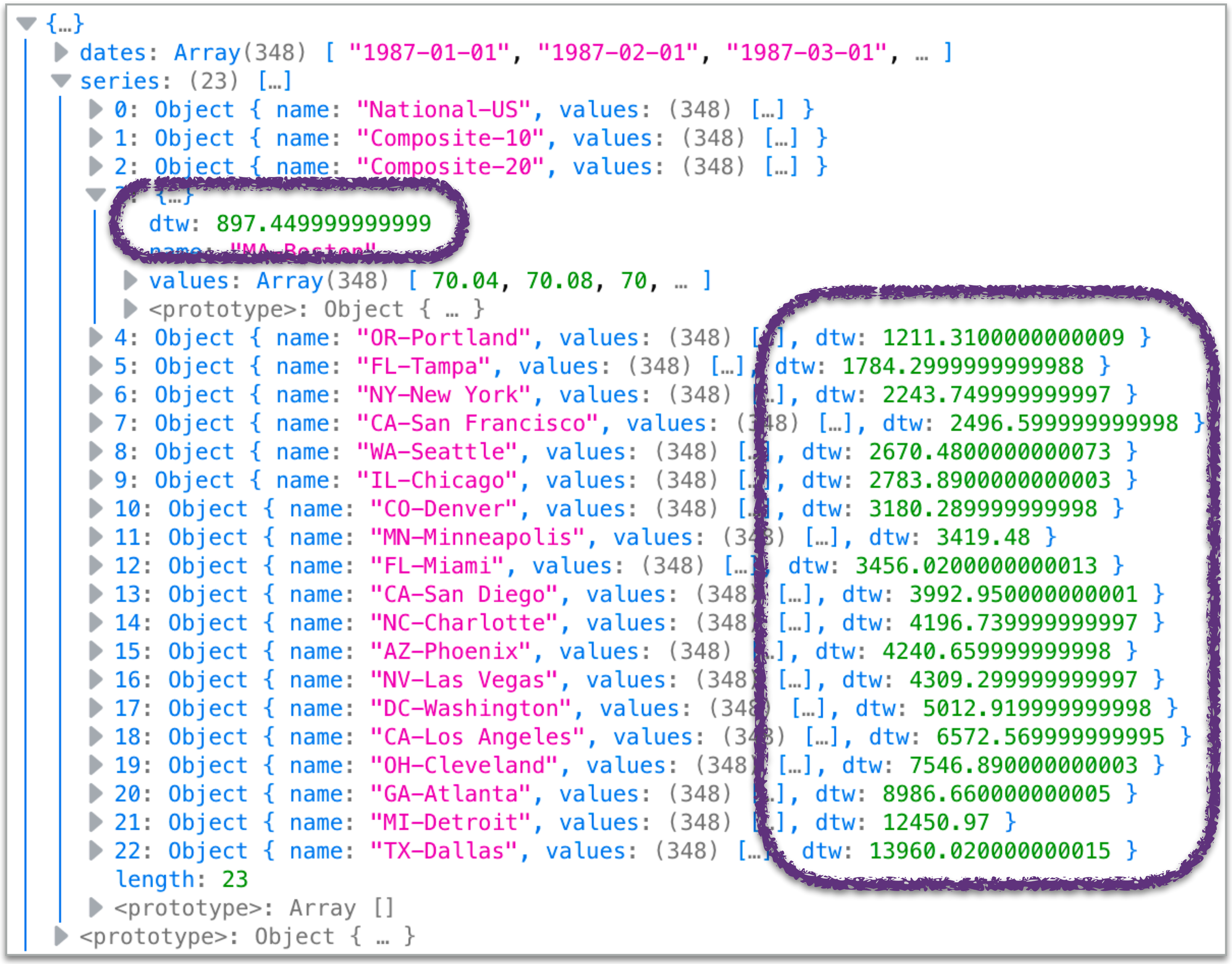


```
▼ {…}
  ▶ dates: Array(348) [ "1987-01-01", "1987-02-01", "1987-03-01", … ]
  ▼ series: (23) […]
    ▶ 0: Object { name: "National-US", values: (348) […] }
    ▶ 1: Object { name: "Composite-10", values: (348) […] }
    ▶ 2: Object { name: "Composite-20", values: (348) […] }
    ▼ 3: {…}
        dtw: 897.449999999999
        name: "MA-Boston"
      ▶ values: Array(348) [ 70.04, 70.08, 70, … ]
      ▶ <prototype>: Object { … }
    ▶ 4: Object { name: "OR-Portland", values: (348) […], dtw: 1211.3100000000009 }
    ▶ 5: Object { name: "FL-Tampa", values: (348) […], dtw: 1784.2999999999988 }
    ▶ 6: Object { name: "NY-New York", values: (348) […], dtw: 2243.749999999997 }
    ▶ 7: Object { name: "CA-San Francisco", values: (348) […], dtw: 2496.599999999998 }
    ▶ 8: Object { name: "WA-Seattle", values: (348) […], dtw: 2670.4800000000073 }
    ▶ 9: Object { name: "IL-Chicago", values: (348) […], dtw: 2783.8900000000003 }
    ▶ 10: Object { name: "CO-Denver", values: (348) […], dtw: 3180.289999999998 }
    ▶ 11: Object { name: "MN-Minneapolis", values: (348) […], dtw: 3419.48 }
    ▶ 12: Object { name: "FL-Miami", values: (348) […], dtw: 3456.0200000000013 }
    ▶ 13: Object { name: "CA-San Diego", values: (348) […], dtw: 3992.950000000001 }
    ▶ 14: Object { name: "NC-Charlotte", values: (348) […], dtw: 4196.739999999997 }
    ▶ 15: Object { name: "AZ-Phoenix", values: (348) […], dtw: 4240.659999999998 }
    ▶ 16: Object { name: "NV-Las Vegas", values: (348) […], dtw: 4309.299999999997 }
    ▶ 17: Object { name: "DC-Washington", values: (348) […], dtw: 5012.919999999998 }
    ▶ 18: Object { name: "CA-Los Angeles", values: (348) […], dtw: 6572.569999999995 }
    ▶ 19: Object { name: "OH-Cleveland", values: (348) […], dtw: 7546.890000000003 }
    ▶ 20: Object { name: "GA-Atlanta", values: (348) […], dtw: 8986.660000000005 }
    ▶ 21: Object { name: "MI-Detroit", values: (348) […], dtw: 12450.97 }
    ▶ 22: Object { name: "TX-Dallas", values: (348) […], dtw: 13960.020000000015 }
      length: 23
    ▶ <prototype>: Array []
  ▶ <prototype>: Object { … }
```

Exercise s06b

Dynamic Time Warping

- used to compute the distance between two series; we will sort the array according to it;
- for each series, except the 3 aggregates: `National-US`, `Composite-10` and `Composite-20`, store the distance between a city's series and `National-US`;
- order the series array according to attribute `dtw`, putting the 3 aggregates (which do not have a `dtw` value) first;
- `dynamic-time-warping.min.js` is already imported.

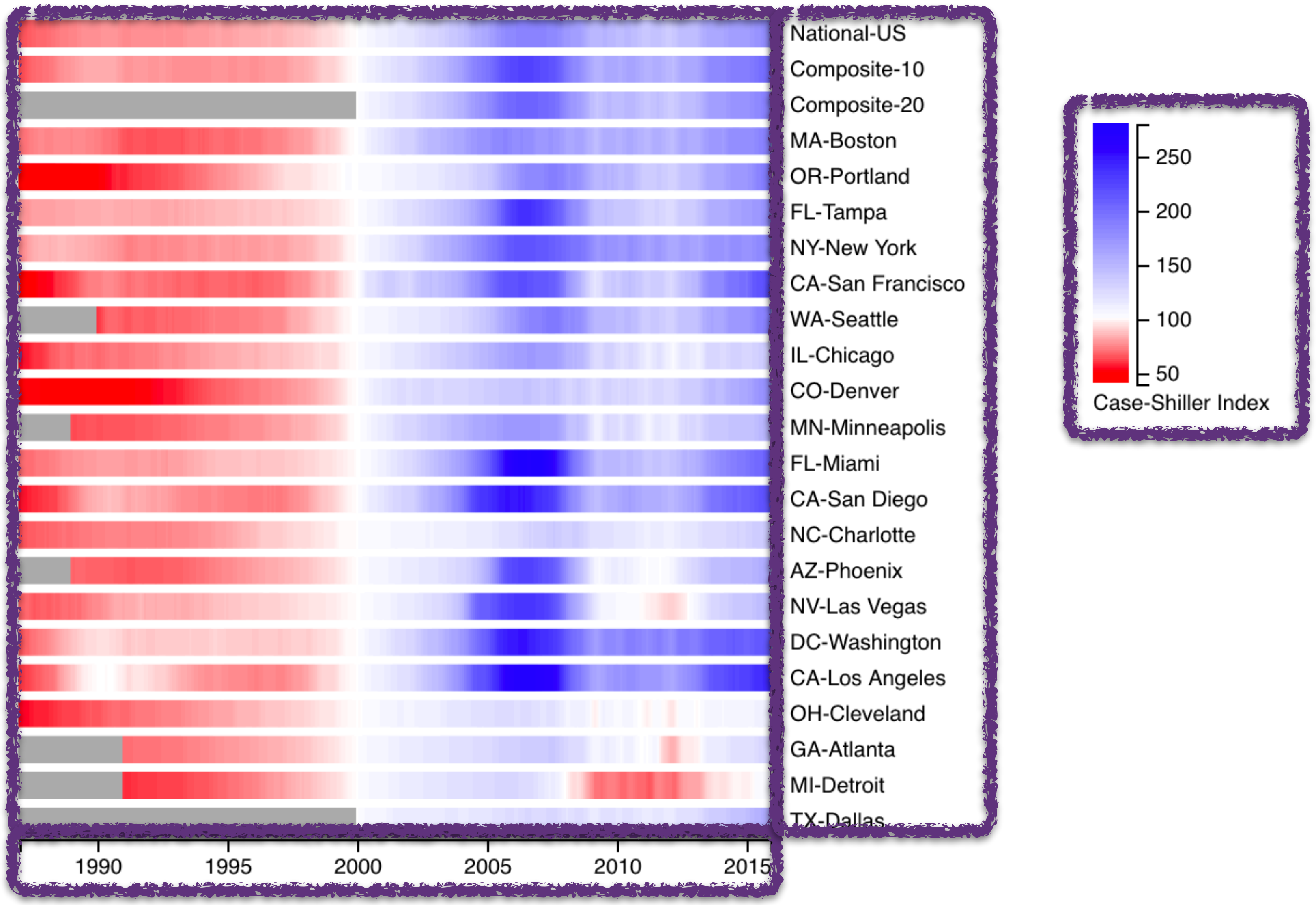


```
▼ {…}
  ▶ dates: Array(348) [ "1987-01-01", "1987-02-01", "1987-03-01", … ]
  ▼ series: (23) […]
    ▶ 0: Object { name: "National-US", values: (348) […] }
    ▶ 1: Object { name: "Composite-10", values: (348) […] }
    ▶ 2: Object { name: "Composite-20", values: (348) […] }
    ▼ 3: {…}
        dtw: 897.449999999999
        name: "MA-Boston"
      ▶ values: Array(348) [ 70.04, 70.08, 70, … ]
      ▶ <prototype>: Object { … }
    ▶ 4: Object { name: "OR-Portland", values: (348) […], dtw: 1211.3100000000009 }
    ▶ 5: Object { name: "FL-Tampa", values: (348) […], dtw: 1784.2999999999988 }
    ▶ 6: Object { name: "NY-New York", values: (348) […], dtw: 2243.749999999997 }
    ▶ 7: Object { name: "CA-San Francisco", values: (348) […], dtw: 2496.599999999998 }
    ▶ 8: Object { name: "WA-Seattle", values: (348) […], dtw: 2670.4800000000073 }
    ▶ 9: Object { name: "IL-Chicago", values: (348) […], dtw: 2783.8900000000003 }
    ▶ 10: Object { name: "CO-Denver", values: (348) […], dtw: 3180.289999999998 }
    ▶ 11: Object { name: "MN-Minneapolis", values: (348) […], dtw: 3419.48 }
    ▶ 12: Object { name: "FL-Miami", values: (348) […], dtw: 3456.0200000000013 }
    ▶ 13: Object { name: "CA-San Diego", values: (348) […], dtw: 3992.950000000001 }
    ▶ 14: Object { name: "NC-Charlotte", values: (348) […], dtw: 4196.739999999997 }
    ▶ 15: Object { name: "AZ-Phoenix", values: (348) […], dtw: 4240.659999999998 }
    ▶ 16: Object { name: "NV-Las Vegas", values: (348) […], dtw: 4309.299999999997 }
    ▶ 17: Object { name: "DC-Washington", values: (348) […], dtw: 5012.919999999998 }
    ▶ 18: Object { name: "CA-Los Angeles", values: (348) […], dtw: 6572.569999999995 }
    ▶ 19: Object { name: "OH-Cleveland", values: (348) […], dtw: 7546.890000000003 }
    ▶ 20: Object { name: "GA-Atlanta", values: (348) […], dtw: 8986.660000000005 }
    ▶ 21: Object { name: "MI-Detroit", values: (348) […], dtw: 12450.97 }
    ▶ 22: Object { name: "TX-Dallas", values: (348) […], dtw: 13960.020000000015 }
      length: 23
    ▶ <prototype>: Array []
  ▶ <prototype>: Object { … }
```

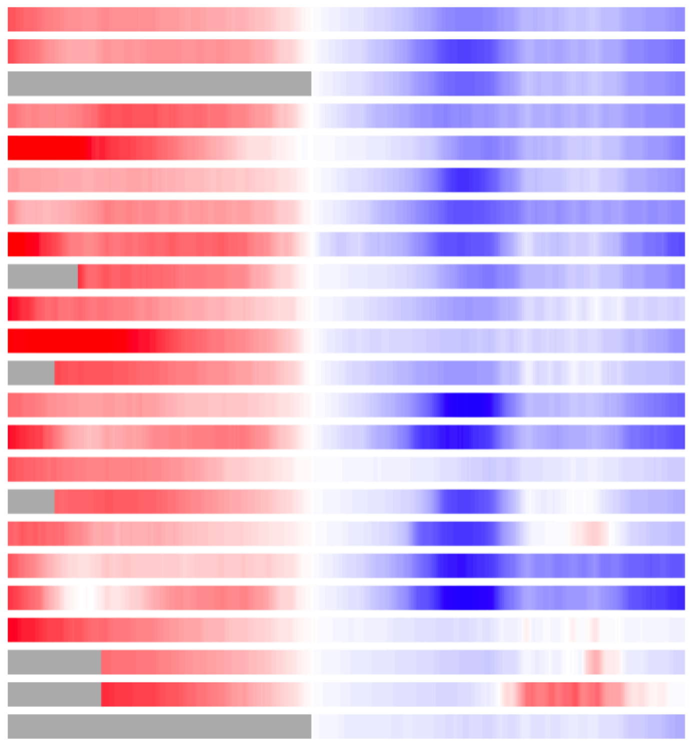
https://github.com/GordonLesti/dynamic-time-warping

Exercise s06b

- The data structure is now ready. We can create the different parts of the visualization:



Exercise s06b



- We start by drawing each series on a separate band, adding vertical SVG `<line>` elements.

- Each of the 23 bands consists of 348 lines (1px wide).

- For each band:

  - Create a `<g>` and translate it vertically using an SVG affine transform, so that bands don't overlap, filling the entire vertical, only leaving about 20px at the bottom of the SVG canvas free for the time axis (added later, see next slide).

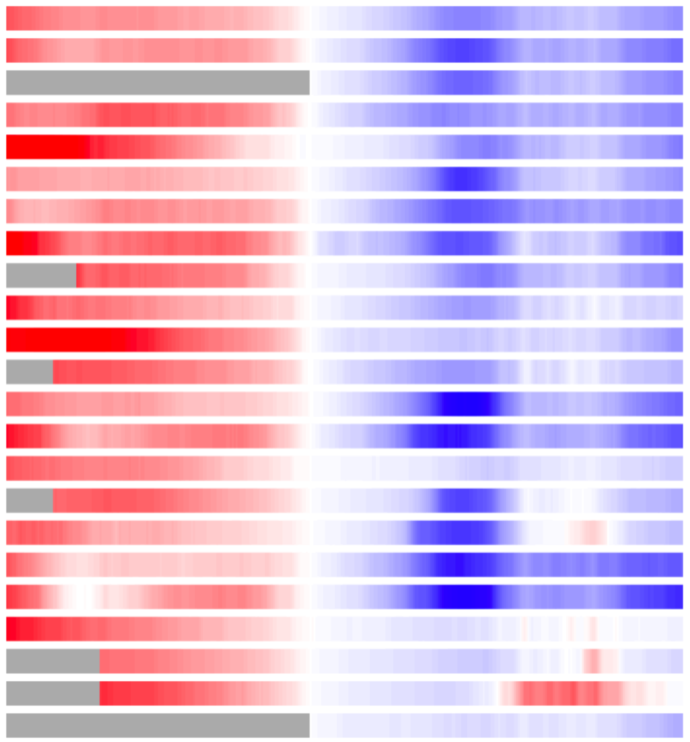
  - Inside each <g>, bind the corresponding series' `values` array to `<line>` elements, setting their x1,y1,x2,y2 attributes (line endpoint coordinates) so as to eventually create color strips as shown above. Reminder: `.attr(”foo”, function(d,i){return …;})`

    iterator

  - The color of each individual 1px-wide `<line>` depends on the actual value for that year in the series. Create a *diverging* color scale that creates this visual mapping (see hints on next slide), and use that scale to map a `stroke` color to each `<line>` element.

Exercise s06b

- Create a piecewise scale, that maps shades of red to index values < 100, white to 100, and shades of blue to values > 100.



- Look here for an example of piecewise mapping to a diverging color scheme:

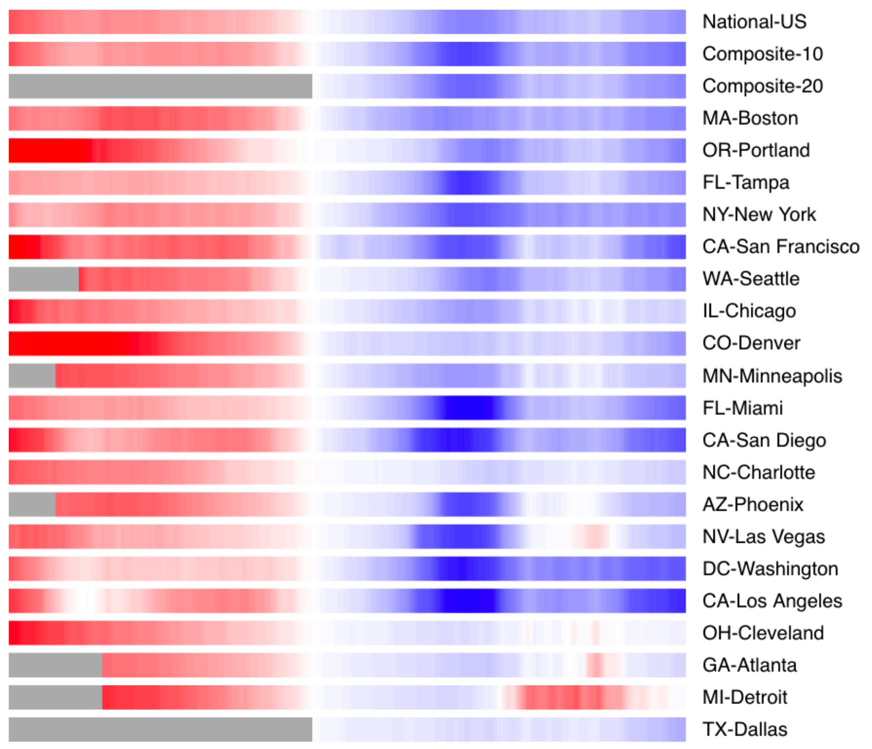
`https://github.com/d3/d3-scale#continuous_domain`

- The scale's **`domain`** should be the min and max values over *all* time-series. Remember: functions such as **`d3.min(), d3.max()`** are your friends.

- Display **`null`** values in time-series by painting the corresponding **`<line>`** elements gray. (Handle this case outside of the color scale).

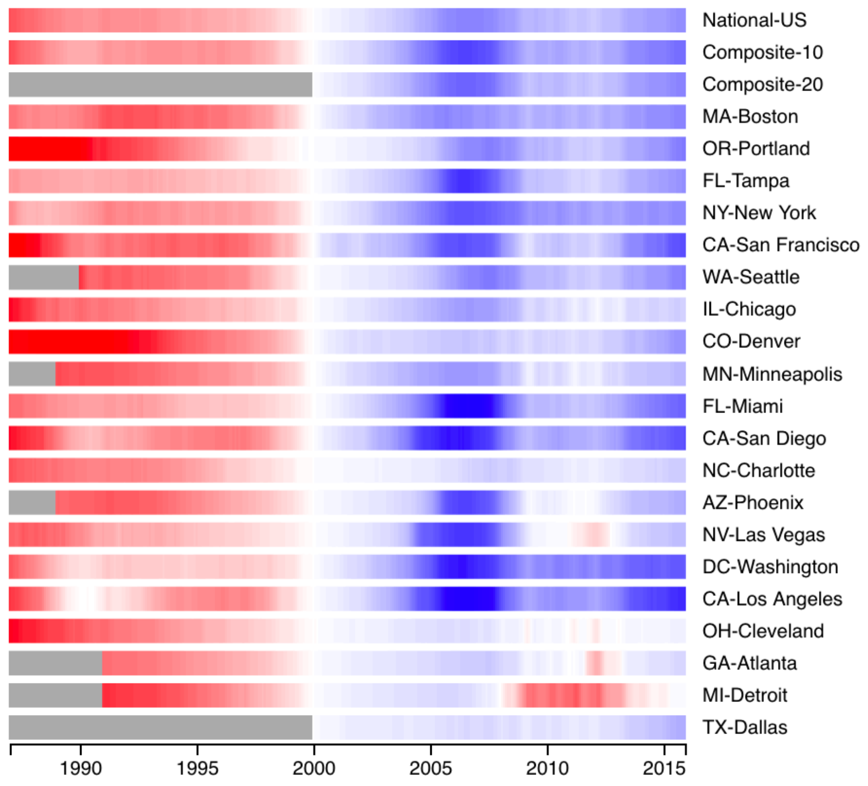
Exercise s06b

- Add `<text>` elements next to each time-series' color strip, showing the series' name.
- Put them in the same `<g>` as the `<line>` elements, to benefit from the affine transform already set up for vertical layout.



Exercise s06b

- Add a time scale in the remaining 20px or so that you have reserved for it at the bottom of the SVG canvas.



- We did that already for the scatterplot in TD s#02:

```
ctx.xScale = d3.scaleLinear().domain([0, maxStarMass])
                             .range([60, ctx.w-20]);

d3.select("#bkgG").append("g")
  .attr("transform", "translate(0,"+ (ctx.h - 50) +")")
  .call(d3.axisBottom(ctx.xScale).ticks(10));
```

- The principle is the same here. Create a **`d3.scaleTime()`**, set its domain to the extent of dates, using **`ctx.timeParser()`** to get **`Date`** objects from the first and last strings in that array. Set **`ticks()`** every 5 years. Adapt the **`transform`** to put the axis in the right place.

https://github.com/d3/d3-scale#time-scales

https://github.com/d3/d3-scale#time_ticks