Data Visualization

INF552 - 2023 - Session 03 - exercices
Visualizing networks with D3

ÉCOLE POLYTECHNIQUE

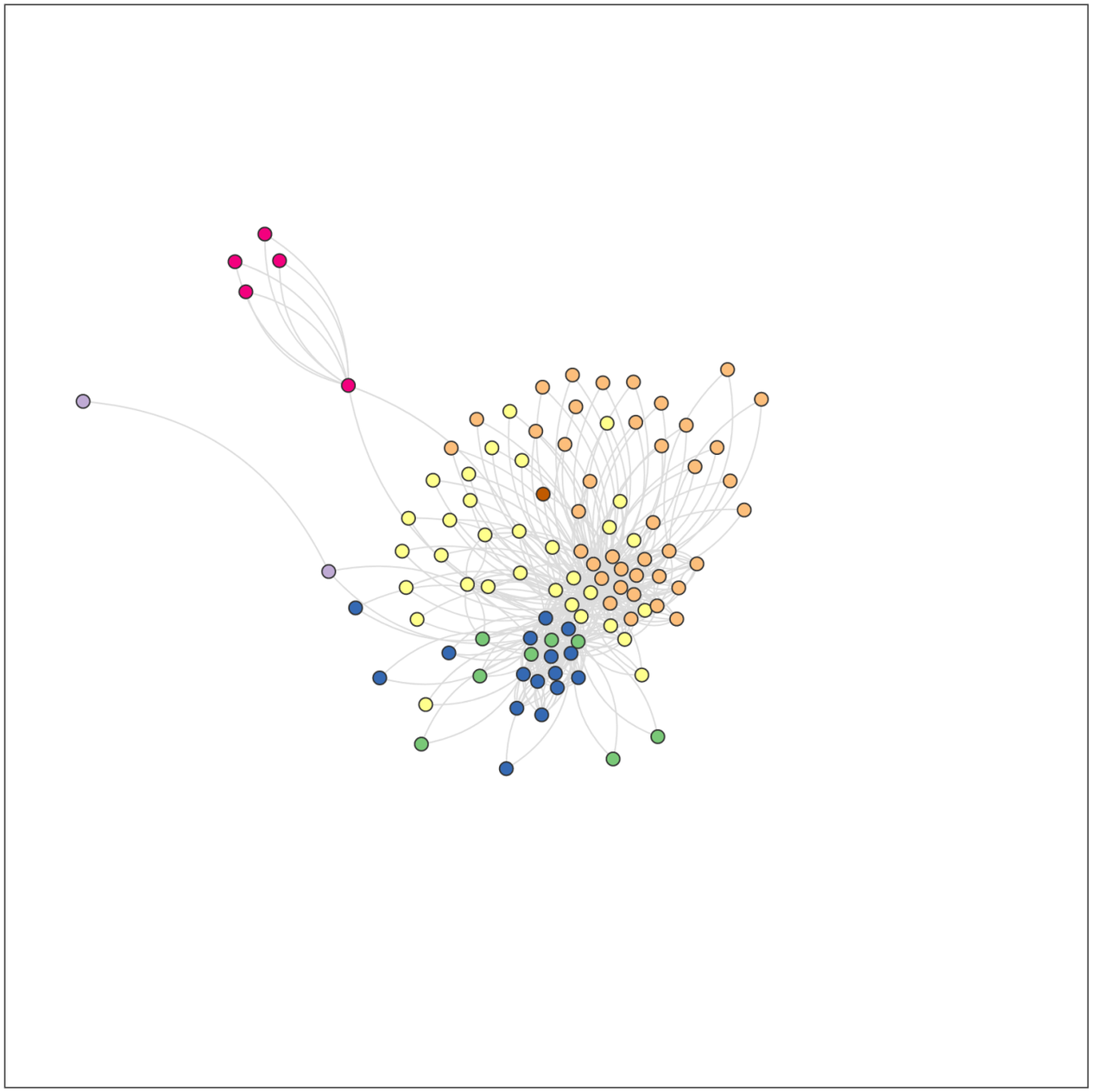
Inria

emmanuel.pietriga@inria.fr

olivier.gladin@inria.fr

PC s07

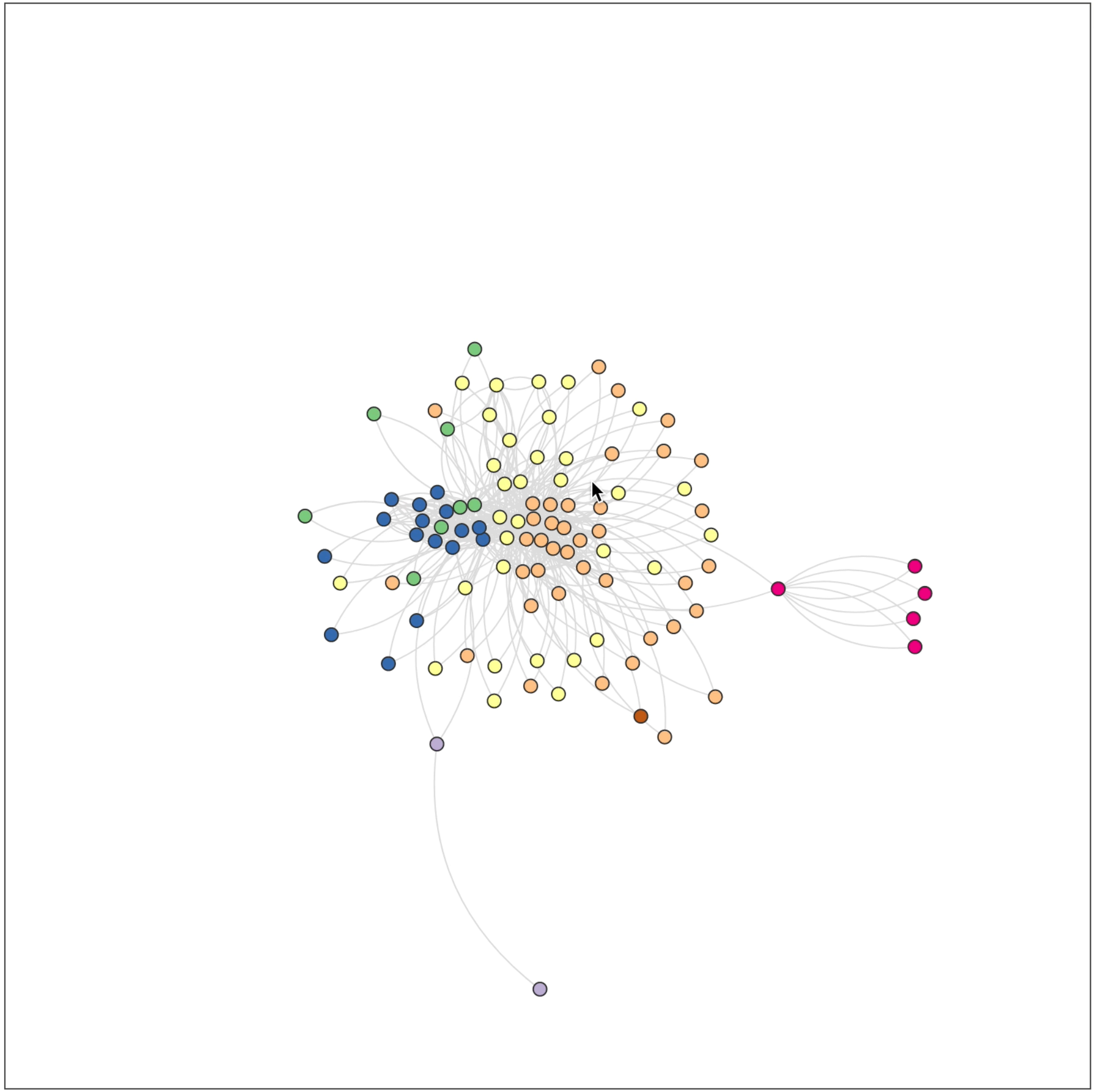
Visualization of an air traffic network.



Airports: 103, Flights:582

Airports: 103, Flights:582

switch



Airports: 103, Flights:582

d3-force

https://github.com/d3/d3-force/blob/master/README.md

Simulates physical forces on particles. Can be used for graph layout, collision detection, simple physical simulations.

Usage:

- create a simulation and set the nodes it applies to:

```
var simulation = d3.forceSimulation(myNodeArray);
```

- set forces
  (in our case attraction forces of the graph's edges + node repulsion force + weak attraction force at the center)

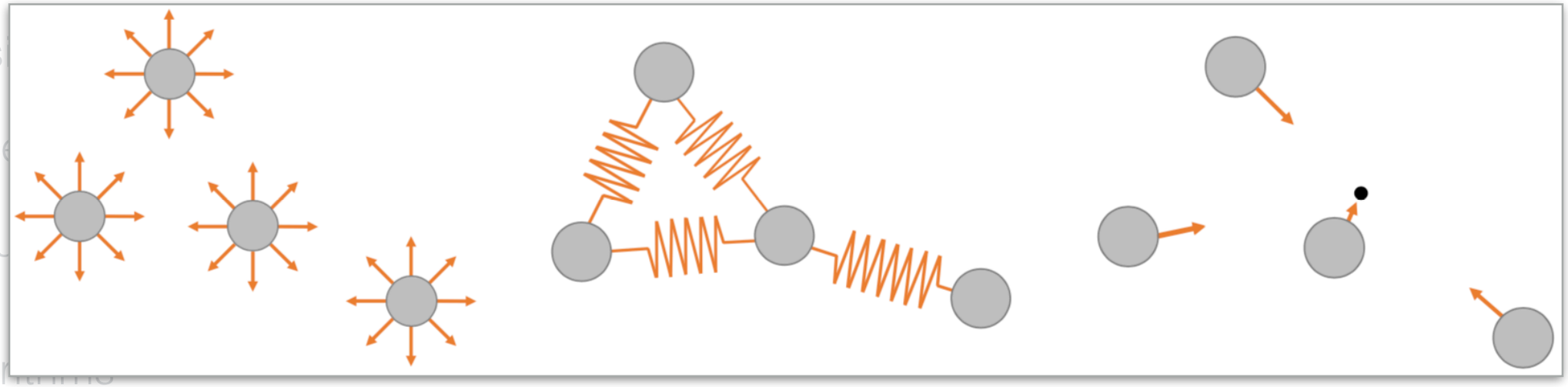
```
simulation.force("link", ...);
```
```
simulation.force("charge", ...);
```
```
simulation.force("center", ...);
```

- listen to tick events (callback triggered at each new simulation step), move nodes and link endpoints accordingly

```
simulation.on("tick", mySimCallback);

function mySimCallback(){
    d3.selectAll("line")
      .attr("x1", (d) => (d.source.x))
      .attr("y1", (d) => (d.source.y))
      .attr("x2", (d) => (d.target.x))
      .attr("y2", (d) => (d.target.y));
    d3.selectAll(“#nodes circle”).attr(“cx”, (d) => (d.x))
                                 .attr(“cy", (d) => (d.y));
}
```

Node-Link Diagrams / Force-directed Layout

- Metaphor of physical forces: attraction (springs, gravity) + repulsion (charged particles)



```
var simulation = d3.forceSimulation()
                   .force("charge", d3.forceManyBody())
                   .force("link", d3.forceLink().id(function(d) { return d.id; }).distance(5).strength(0.08))
                   .force("center", d3.forceCenter(ctx.w / 2, ctx.h / 2));
```

- Physi
  - ite
  - ru
- Algorithms
  - GEM
  - ForceAtlas
  - LinLog
- Complexity
  - basic approaches are $O(|V|^2)$
  - Barnes & Hut's hierarchical force-calculation algorithm is $O(|E| + |V| \times log(|V|))$

A note on SVG `path`

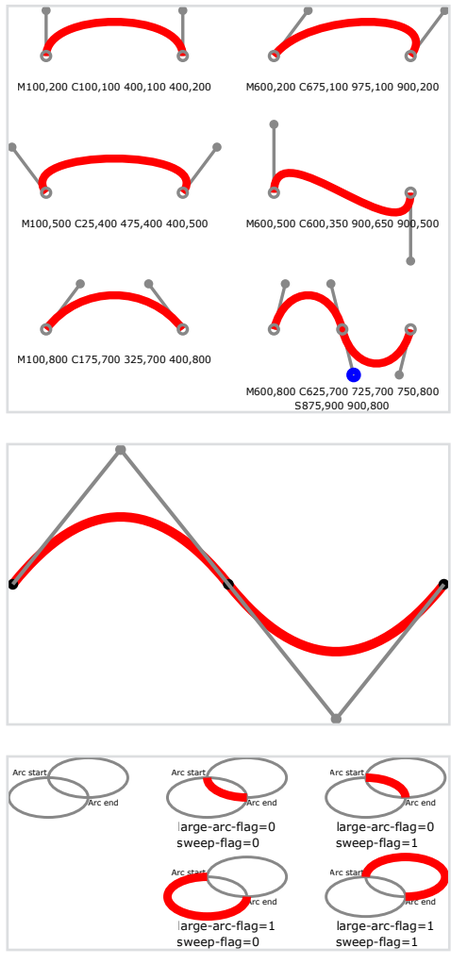
- `d` holds values like:

```
<path d="M10,315L110,215A36,60 0 0,1 150.71,170.29L172.55,152.45A30,50 -45 0,1 215.1,109.9L315,10"/>
```

- It has its own mini-syntax, with the following commands:

(cp = control point)

Command	Action	Syntax
L	line	`Lx,y`
H	horizontal line	`Hx`
V	vertical line	`Vy`
M	move (jump, without drawing)	`Mx,y`
Z	close path (straight line)	`Z`
C	cubic bezier curve (2 control points)	`Ccp1x,cp1y cp2x,cp2y x,y`
Q	quadratic bezier curve (1 control point)	`Qcpx,cpy x,y`
S	same as C where first cp is a reflection (central symmetry) of the last cp from the previous C or S cmd	`Scp2x,cp2y x,y`
T	same as Q where first cp is a reflection (central symmetry) of the cp from the previous Q or T cmd	`Tx y`
A	arc (ellipse with dimensions rX,rY,rotation)	`A rX,rY rotation large-arc-flag,sweep-flag x,y`



- Same commands using lowercase letters will interpret the coordinate pairs as relative coordinates rather than absolute ones.

[Image source: Scalable Vector Graphics 2 W3C Candidate Recommendation 07 August 2018]

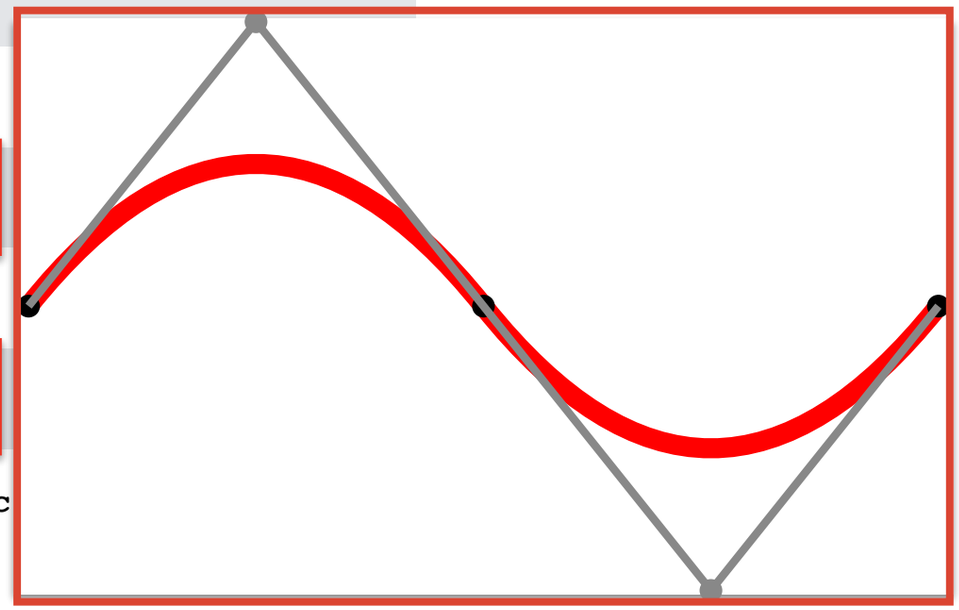
A note on SVG `path`

- `d` holds values like:

```
<path d="M10,315L110,215A36,60 0 0,1 150.71,170.29L172.55,152.45A30,50 -45 0,1 215.1,109.9L315,10"/>
```

- It has its own mini-syntax, with the following commands: (cp = control point)

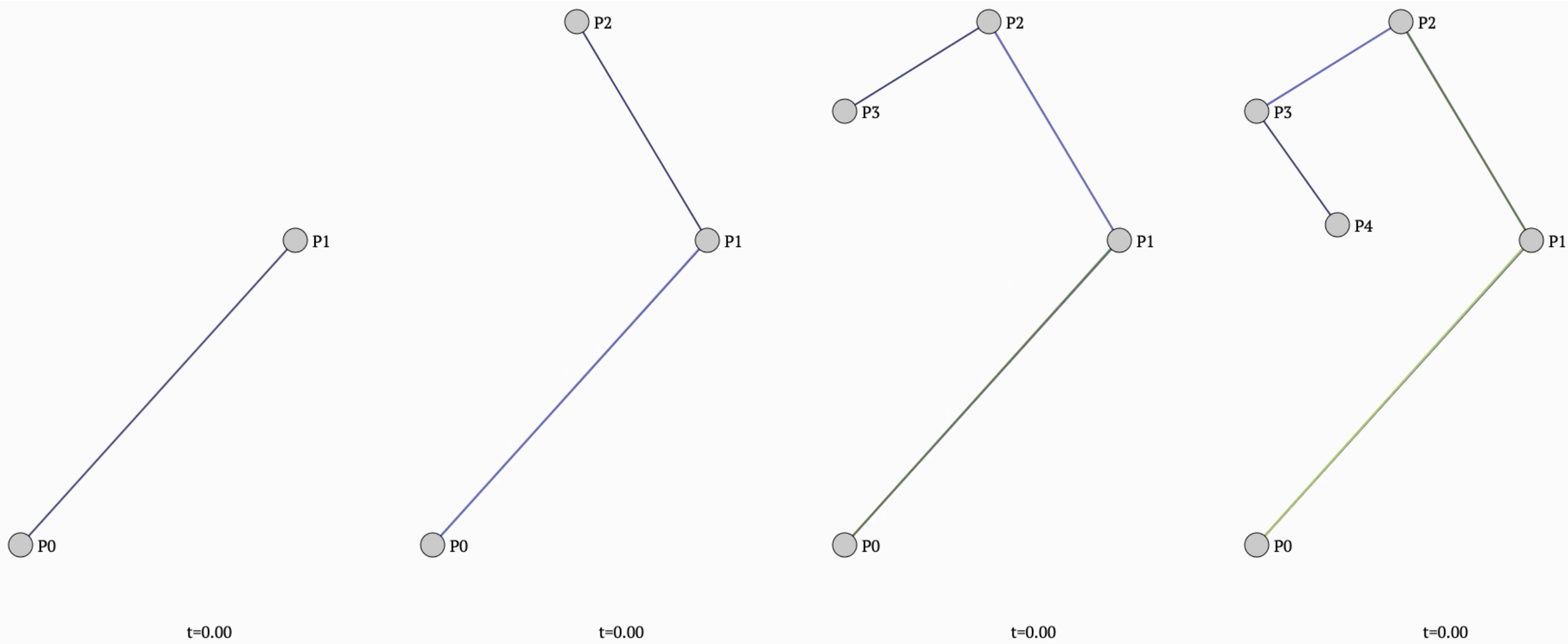
Command	Action	Syntax
L	line	`Lx,y`
H	horizontal line	`Hx`
V	vertical line	`Vy`
M	move (jump, without drawing)	`Mx,y`
Z	close path (straight line)	`Z`
C	cubic bezier curve (2 control points)	`Ccp1x,cp1y cp2x,cp2y x,y`
Q	quadratic bezier curve (1 control point)	`Qcpx,cpy x,y`
S	same as C where first cp is a reflection (central symmetry) of the last cp from the previous C or S cmd	`Scp2x,cp2y x,y`
T	same as Q where first cp is a reflection (central symmetry) of the cp from the previous Q or T cmd	`Tx y`
A	arc (ellipse with dimensions rX,rY,rotation)	`A rX,rY rotation large-arc`



- Same commands using lowercase letters will interpret the coordinate pairs as relative coordinates rather than absolute ones.

[Image source: Scalable Vector Graphics 2 W3C Candidate Recommendation 07 August 2018]

A note on SVG `path`



[Image source: Jason Davis, 2010]

A note on SVG `path`

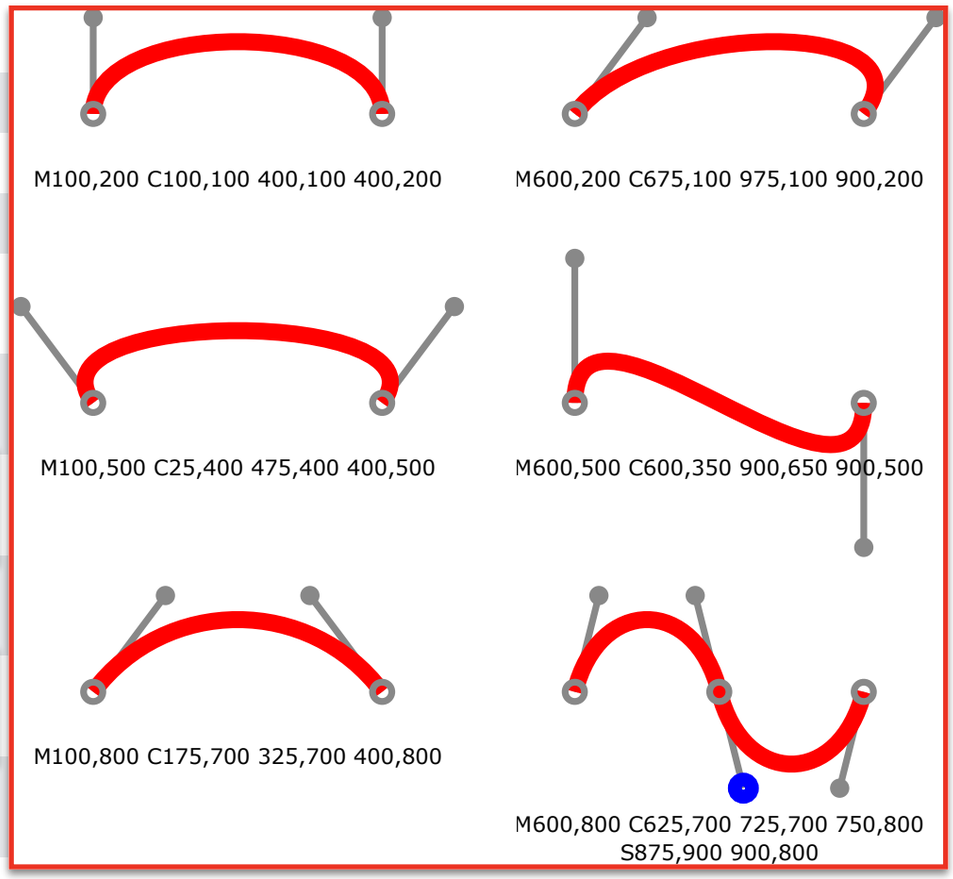
- `d` holds values like:

```
<path d="M10,315L110,215A36,60 0 0,1 150.71,170.29L172.55,152.45A30,50 -45 0,1 215.1,109.9L315,10"/>
```

- It has its own mini-syntax, with the following commands:

Command	Action	Syntax
L	line	`Lx,y`
H	horizontal line	`Hx`
V	vertical line	`Vy`
M	move (jump, without drawing)	`Mx,y`
Z	close path (straight line)	`Z`
C	cubic bezier curve (2 control points)	`Ccp1x,cp1y cp2x,cp2y x,y`
Q	quadratic bezier curve (1 control point)	`Qcpx,cpy x,y`
S	same as C where first cp is a reflection (central symmetry) of the last cp from the previous C or S cmd	`Scp2x,cp2y x,y`
T	same as Q where first cp is a reflection (central symmetry) of the cp from the previous Q or T cmd	`Tx y`
A	arc (ellipse with dimensions rX,rY,rotation)	`A rX,rY rotation large-arc-flag,sweep-flag x,y`

(cp = control point)



- Same commands using lowercase letters will interpret the coordinate pairs as relative coordinates rather than absolute ones.

[Image source: Scalable Vector Graphics 2 W3C Candidate Recommendation 07 August 2018]

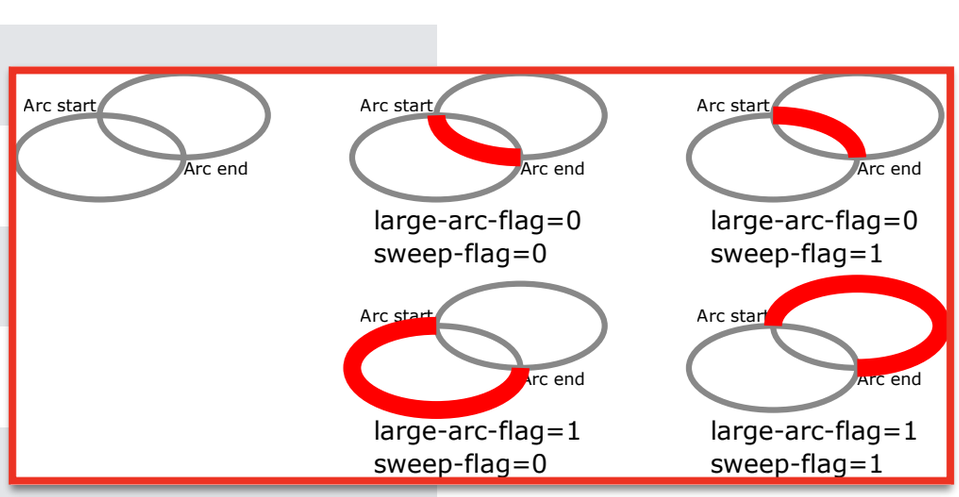
A note on SVG `path`

- `d` holds values like:

```
<path d="M10,315L110,215A36,60 0 0,1 150.71,170.29L172.55,152.45A30,50 -45 0,1 215.1,109.9L315,10"/>
```

- It has its own mini-syntax, with the following commands: (cp = control point)

Command	Action	Syntax
L	line	`Lx,y`
H	horizontal line	`Hx`
V	vertical line	`Vy`
M	move (jump, without drawing)	`Mx,y`
Z	close path (straight line)	`Z`
C	cubic bezier curve (2 control points)	`Ccp1x,cp1y cp2x,cp2y x,y`
Q	quadratic bezier curve (1 control point)	`Qcpx,cpy x,y`
S	same as C where first cp is a reflection (central symmetry) of the last cp from the previous C or S cmd	`Scp2x,cp2y x,y`
T	same as Q where first cp is a reflection (central symmetry) of the cp from the previous Q or T cmd	`Tx y`
A	arc (ellipse with dimensions rX,rY,rotation)	`A rX,rY rotation large-arc-flag,sweep-flag x,y`

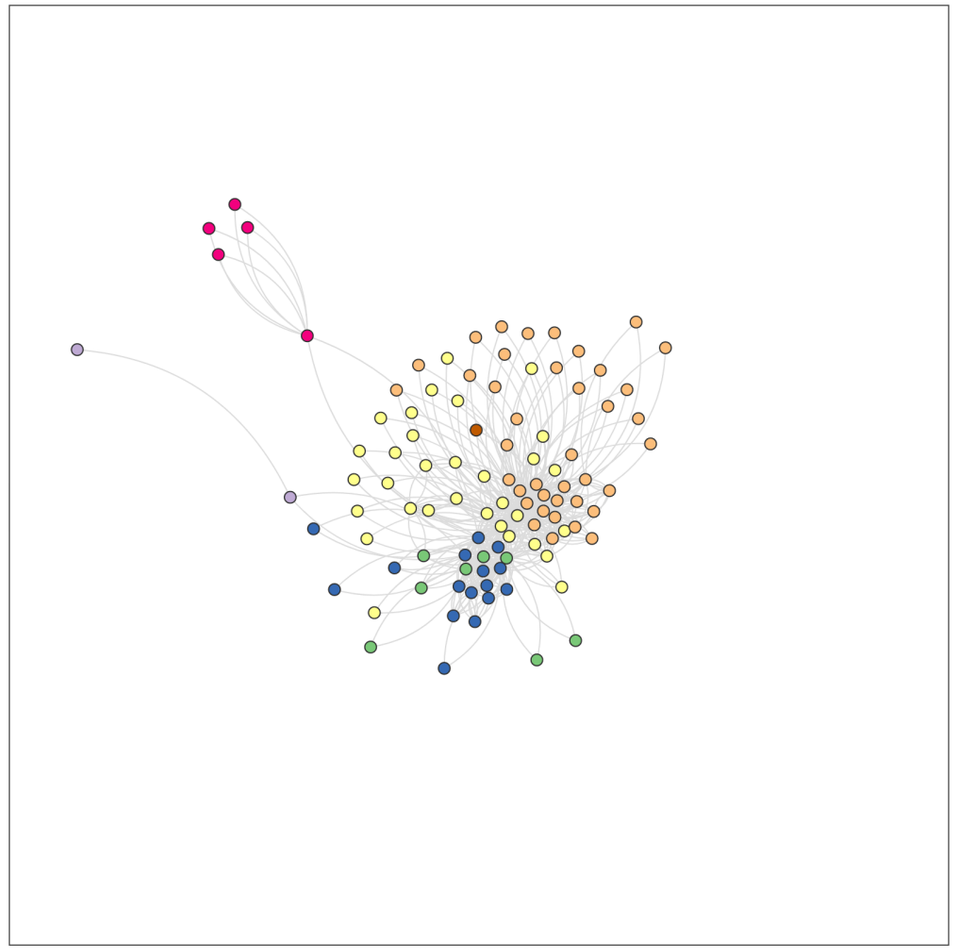


- Same commands using lowercase letters will interpret the coordinate pairs as relative coordinates rather than absolute ones.

[Image source: Scalable Vector Graphics 2 W3C Candidate Recommendation 07 August 2018]

PC s07

- Visualization of an air traffic network.
- The network consists of airports (nodes) in the USA connected by flights (edges):
  - the network is hierarchical: each airport belongs to a state (`AL, MA, CA,` *etc.*);
  - edges are weighted using a numerical attribute (representing the passenger traffic on each edge).



PC s07

- The input data is split in four files:
  - **`airports.json`** contains data about the airports only;
  - **`flights.json`** contains data about the flights between those airports;
  - **`states_tz.csv`** contains data about which time zone all 50 states are in;
  - **`us-states.geojson`** contains data to draw the US states map.
- Load all files, using Javascript function **`Promise.all()`** to handle the async' calls.

```
d3.json(...) // like d3.csv(), returns a Promise.
             //  We called then(...) directly on that promise in ex02.

// Here we want to load multiple resources before proceeding to the creation
// of the visualization. To wait for multiple promises to be completed, use:
Promise.all(...).then(function(data){/* Callback code */})

// This runs all promises given as input (array) to all(),
// and executes the code in then() only once
// all promises have been completed
// (in our case, fetching and parsing all CSV and JSON files)
```

https://github.com/d3/d3-fetch

https://developer.mozilla.org/docs/Web/JavaScript/Reference/Global_Objects/Promise/all

PC s07

- We will first visualize this network using a classic force-directed graph layout.
- Transform the input data to match the data structure expected by this D3 component.
- The graph is quite large and noisy. Filter the data given as input to the graph layout algorithm:
  - ignore airports with an IATA code starting with a number, such as, *e.g.*, `06C`;
  - ignore flights whose `count < 3000`;
  - ignore airports that are not connected (possibly as a result of the above flight filtering).
- Add a group attribute to nodes (airports), whose value is the parent state's time zone.
- Use the trafic volume (`count`) as the edges' weight (attribute `value`).
- Add the state and city to node properties, as we will also display this information on demand.

```
{nodes:[{id: "ATL", group:"EST", state: “GA”, city: “Atlanta”, latitude:…, longitude:…},
        {id: "ABE", group:"EST", state: “PA”, city: “Allentown”, latitude:…, longitude:…},
        …],
 links:[{source:"PHX", target:”ABQ”, value: 5265.0},
        …]
}
```

ssic force-directed graph layout.

tructure expected by this D3 component.

ata given as input to the graph layout algorithm:

th a number, such as, *e.g.*, `06C`;

ose value is the parent state's time zone.

eight (attribute `value`).

ve will also display this information on demand.


```
[{"city": "Bay Springs", "country": "USA", "iata": "00M",
"latitude": 31.95376472, "longitude": -89.23450472, "name":
"Thigpen", "state": "MS"},
 {"city": "Livingston", "country": "USA", "iata": "00R",
"latitude": 30.68586111, "longitude": -95.01792778, "name":
"Livingston Municipal", "state": "TX"},
 {"city": "Colorado Springs", "country": "USA", "iata":
"00V", "latitude": 38.94574889, "longitude": -104.5698933,
"name": "Meadow Lake", "state": "CO"},
 {"city": "Perry", "country": "USA", "iata": "01G",
"latitude": 42.74134667, "longitude": -78.05208056, "name":
"Perry-Warsaw", "state": "NY"},
 {"city": "Hilliard", "country": "USA", "iata": "01J",
"latitude": 30.6880125, "longitude": -81.90594389, "name":
"Hilliard Airpark", "state": "FL"},
```

```
[{"count": 853.0, "destination": "ATL", "origin": "ABE"},
 {"count": 1.0, "destination": "BHM", "origin": "ABE"},
 {"count": 805.0, "destination": "CLE", "origin": "ABE"},
 {"count": 465.0, "destination": "CLT", "origin": "ABE"},
 {"count": 247.0, "destination": "CVG", "origin": "ABE"},
 {"count": 997.0, "destination": "DTW", "origin": "ABE"},
 {"count": 3.0, "destination": "JFK", "origin": "ABE"},
 {"count": 9.0, "destination": "LGA", "origin": "ABE"},
 {"count": 1425.0, "destination": "ORD", "origin": "ABE"},
 {"count": 2.0, "destination": "PHL", "origin": "ABE"},
 {"count": 2660.0, "destination": "DFW", "origin": "ABI"},
 {"count": 368.0, "destination": "AMA", "origin": "ABQ"},
 {"count": 1067.0, "destination": "ATL", "origin": "ABQ"},
 {"count": 433.0, "destination": "AUS", "origin": "ABQ"},
```

```
State,TimeZone
AL,CST
AK,AKST
AZ,MST
AR,CST
CA,PST
CO,MST
CT,EST
DE,EST
FL,EST
GA,EST
HI,HST
ID,MST
IL,CST
IN,EST
IA,CST
```


```
{nodes:[{id: "ATL", group:"EST", state: “GA",
         city: “Atlanta”, latitude:…, longitude:…},
        {id: "ABE", group:"EST", state: "PA",
         city: “Allentown”, latitude:…, longitude:…},
        …],
 links:[{source:"PHX", target:”ABQ”,
         value: 5265.0},
        …]
}
```

PC s07

- We will first visualize this network using a classic force-directed graph layout.
- Transform the input data to match the data structure expected by this D3 component.
- The graph is quite large and noisy. Filter the data given as input to the graph layout algorithm:
  - ignore airports with an IATA code starting with a number, such as, *e.g.*, `06C`;
  - ignore flights whose **`count < 3000`**;
  - ignore airports that are not connected (possibly as a result of the above flight filtering).
- Add a group attribute to nodes (airports), whose value is the parent state's time zone.
- Use the trafic volume (**`count`**) as the edges' weight (attribute **`value`**).
- Add the state and city to node properties, as we will also display this information on demand.

```
{nodes:[{id: "ATL", group:"EST", state: “GA”, city: “Atlanta”, latitude:…, longitude:…},
        {id: "ABE", group:"EST", state: “PA”, city: “Allentown”, latitude:…, longitude:…},
        …],
 links:[{source:"PHX", target:”ABQ”, value: 5265.0},
        …]
}
```

PC s07

- Once the input data has been transformed to match the structure expected by D3's forceSimulation, populate the SVG canvas:

  - append `<g class="links">` and `<g class="nodes">` to the `<svg>` element;

  - and bind elements from the `nodes` and `links` arrays to graphical marks:

    - using `d3.selectAll(…).data(…).enter()…`, populate the first group with `<line>` elements mapped to items in the `links` array in the input data structure, and set the opacity of all lines to `0.2`;

    - similarly, populate the second group with `<circle>` elements mapped to items in the `nodes` array in the input data structure, and set the radius of all circles to `5`.

PC s07

- We use d3-force to visualize the network as a node-link diagram:
  `https://github.com/d3/d3-force/blob/master/README.md`

- The code initialising the layout algorithm is already provided in **`ex07.js`**

```
var simulation = d3.forceSimulation()
                   .force("link", d3.forceLink().id(function(d) { return d.id; }).distance(5).strength(0.08))
                   .force("charge", d3.forceManyBody())
                   .force("center", d3.forceCenter(ctx.w / 2, ctx.h / 2));
```

- Associate the previously-created nodes and links to this simulation:

```
{nodes:[{id: "iata4foo", group:"PST", state: "ST", city: “foo”},…],
 links:[{source:"iata4foo", target:"iata4bar", value: 853.0},…]
}
```

```
// input data structure created earlier is in ctx.nodes + ctx.links:

simulation.nodes(ctx.nodes)
          .on("tick", simStep);

simulation.force("link")
          .links(ctx.links);

function simStep(){
    // code run at each iteration of the simulation
    // updating the position of nodes and links
    d3.selectAll(“#links line”).attr(“x1”, (d) => (d.source.x))
                               .attr(“y1", (d) => (d.source.y))
                               .attr("x2", (d) => (d.target.x))
                               .attr(“y2", (d) => (d.target.y));
    d3.selectAll(“#nodes circle”).attr(“cx”, (d) => (d.x))
                                 .attr(“cy", (d) => (d.y));
}
```

PC s07

- We use d3-force to visualize the network as a node-link diagram:
  https://github.com/d3/d3-force/blob/master/README.md
- The code initialising the layout algorithm is already provided in `ex07.js`
- Associate the simulation to nodes and links to this simulation:

reconciling nodes and links

```
var simulation = d3.forceSimulation()
                   force("link", d3.forceLink().id(function(d) { return d.id; }).distance(5).strength(0.08))
                   force("charge", d3.forceManyBody())
                   force("center", d3.forceCenter(ctx.w / 2, ctx.h / 2));
```

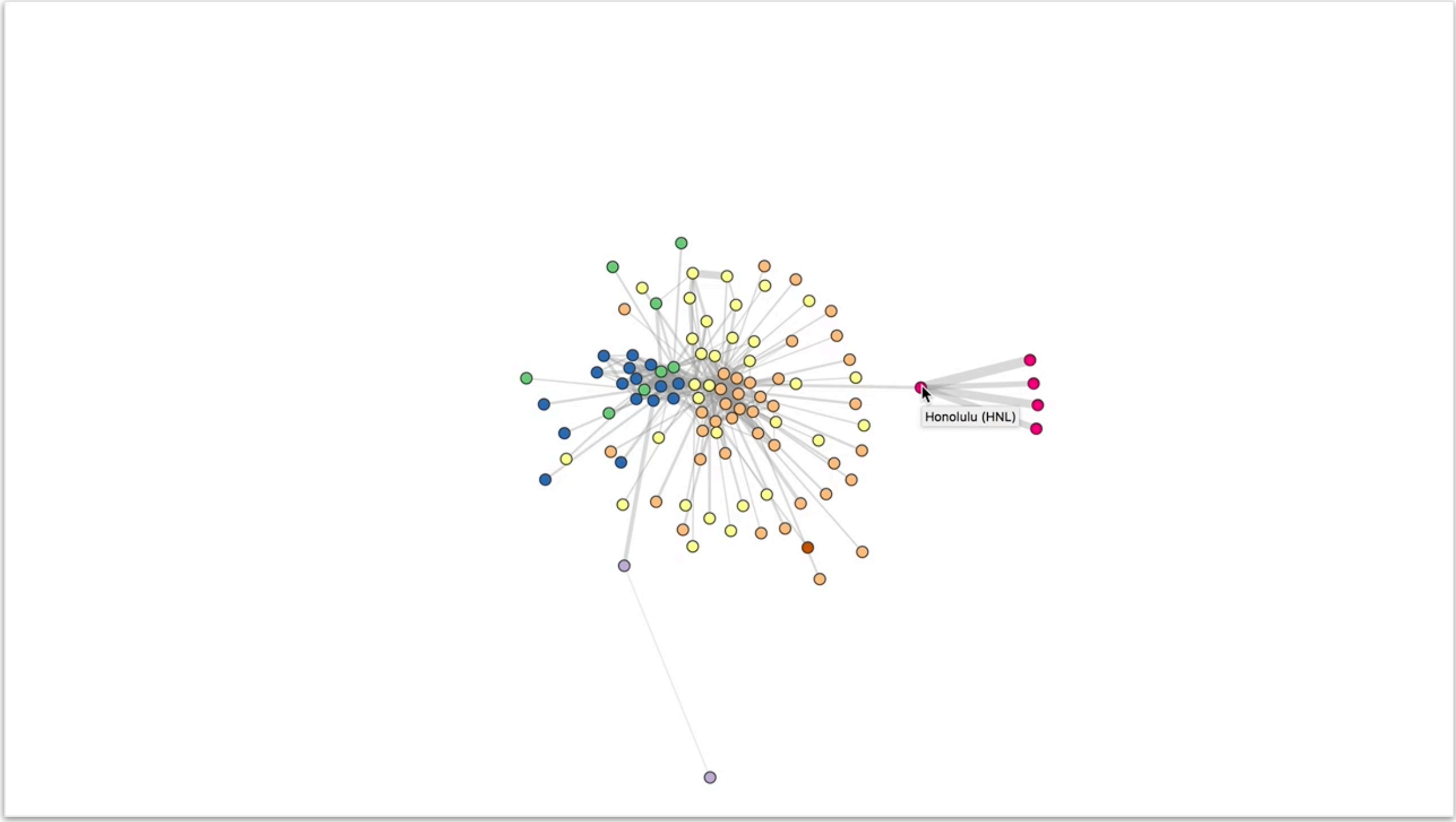
3 different forces relevant to graph layout

```
{nodes:[{id: "iata4                                    ty: “foo”},…],
 links:[{source:"ia                                    853.0},…]
}
```

```
// input data structure created earlier is in ctx.nodes + ctx.links:

simulation.nodes(ctx.nodes)
          .on("tick", simStep);

simulation.force("link")
          .links(ctx.links);

function simStep(){
    // code run at each iteration of the simulation
    // updating the position of nodes and links
    d3.selectAll(“#links line”).attr(“x1”, (d) => (d.source.x))
                               .attr(“y1", (d) => (d.source.y))
                               .attr("x2", (d) => (d.target.x))
                               .attr(“y2", (d) => (d.target.y));
    d3.selectAll(“#nodes circle”).attr(“cx”, (d) => (d.x))
                                 .attr(“cy", (d) => (d.y));
}
```

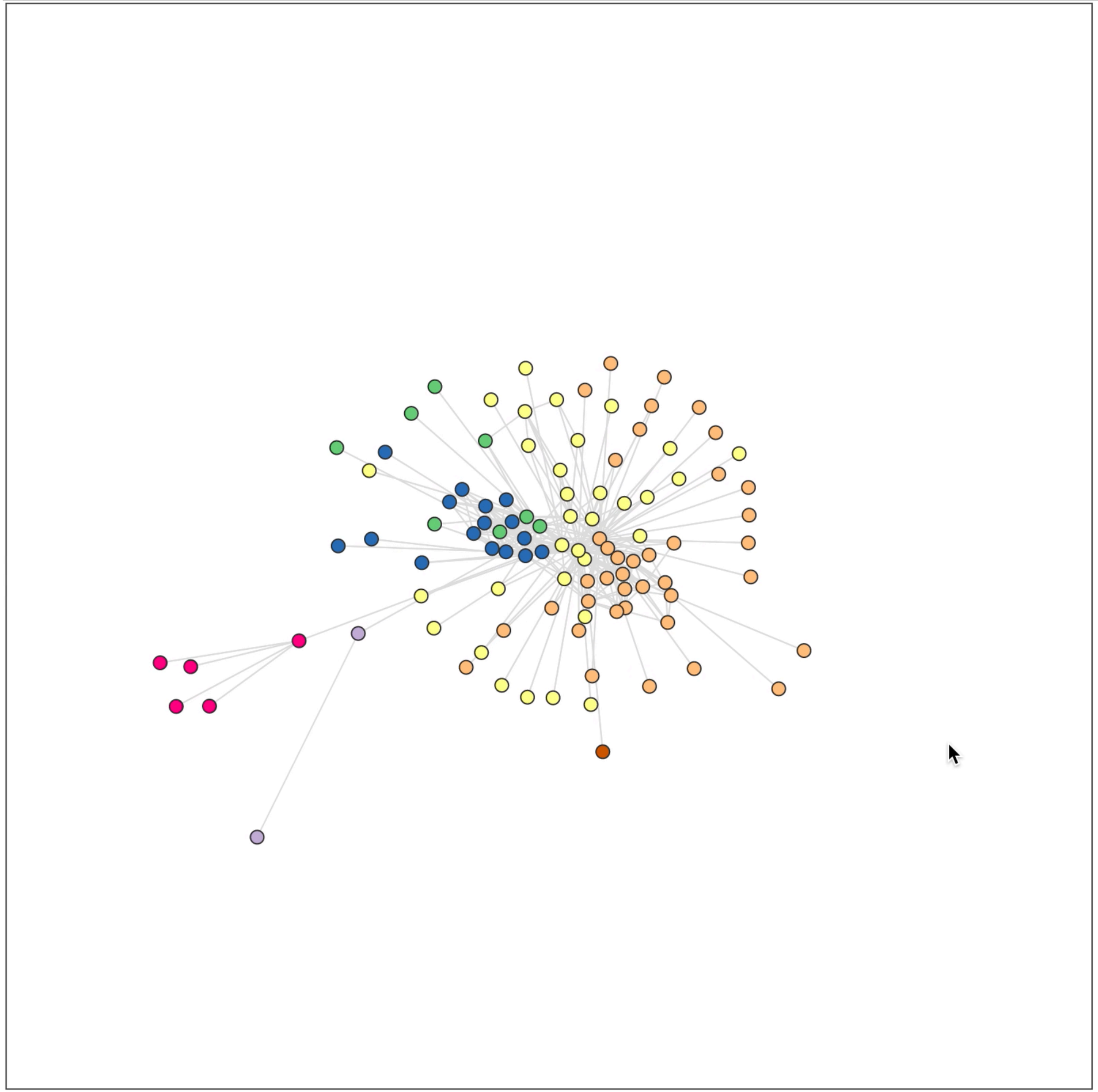
PC s07

Append a **`title`** to each node's **`circle`**. That title should be a string made of the city and airport IATA code. It will be displayed when hovering the node with the mouse cursor.



- Now we want to enable a smooth transition to a representation of this network on a map when pressing the `switch` button.

- All airport nodes have lat/lon coordinates. Draw a map of the US states with the Albers projection, project those airport coordinates on that map using the same projection, smoothly animate the transition from the node-link diagram to the map, and conversely.

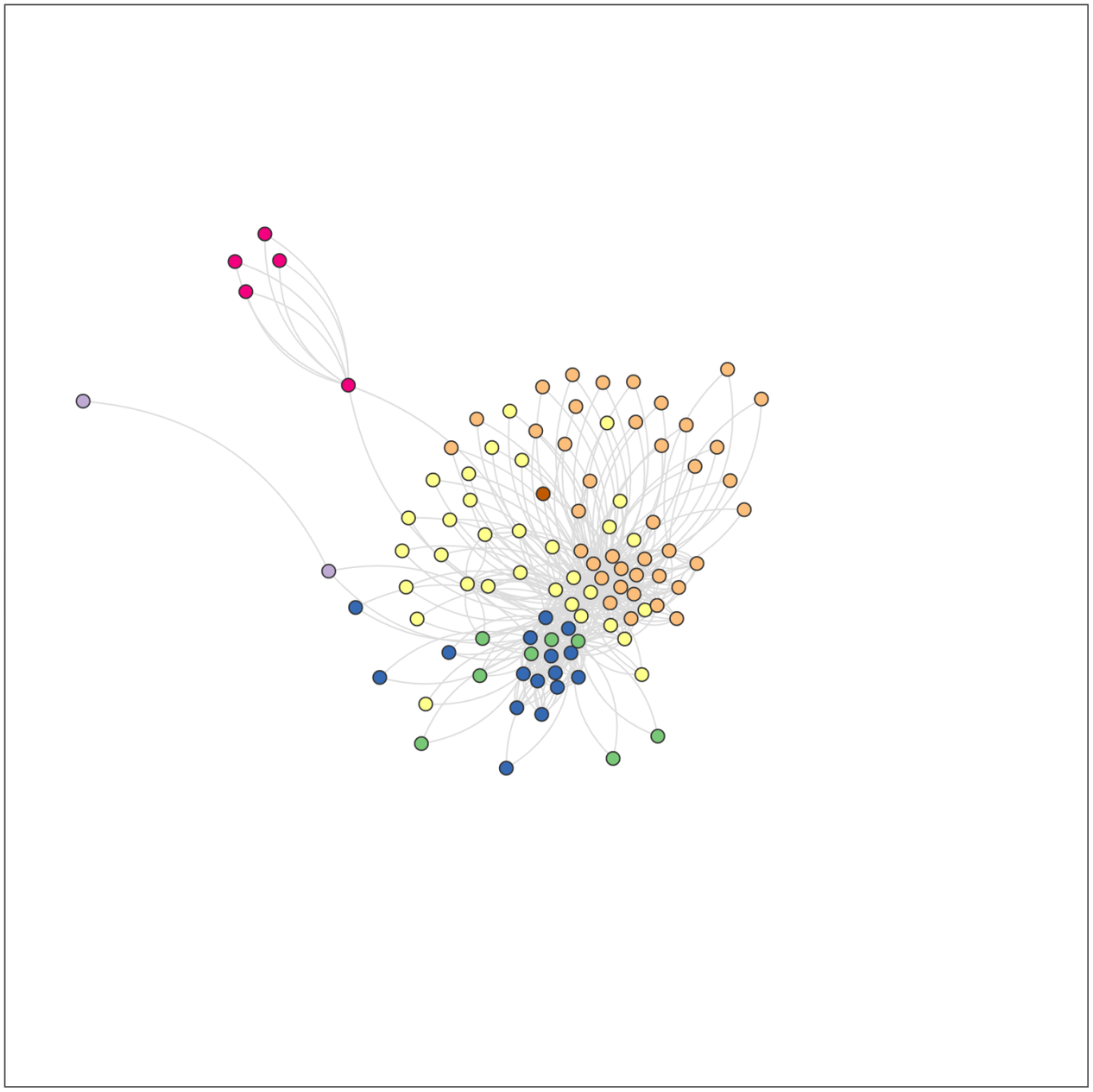


Airports: 103, Flights:582

PC s07

where we see that we are actually dealing with a multi-graph

Replace **<line>** elements by **<path>** elements consisting of a single quadratic bézier curve:



Airports: 103, Flights:582

Airports: 103, Flights:582

A solution for the quad curve control point:

$$\begin{cases} x_{cp} = x_1 + \rho \cdot cos(\alpha + \frac{\pi}{6}) \\ y_{cp} = y_1 + \rho \cdot sin(\alpha + \frac{\pi}{6}) \end{cases} \quad where \quad \begin{cases} \rho = \frac{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}}{2} \\ \alpha = arctan2(\frac{y_2 - y_1}{x_2 - x_1}) \end{cases}$$