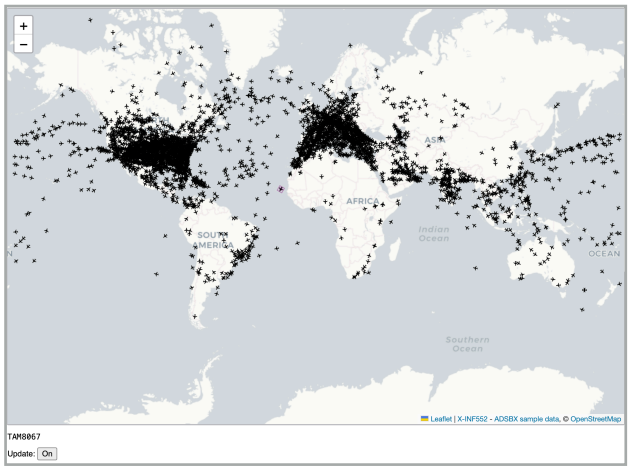
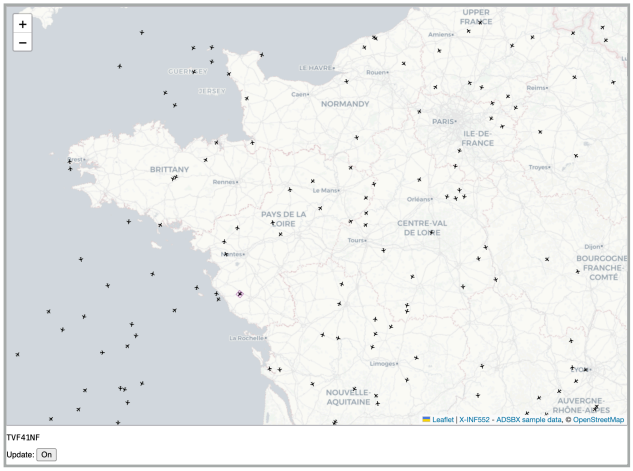
INF552 (2023-2024) - PC s09

Goal: visualize live air traffic from an ADS-B feed using D3 on a leaflet map:

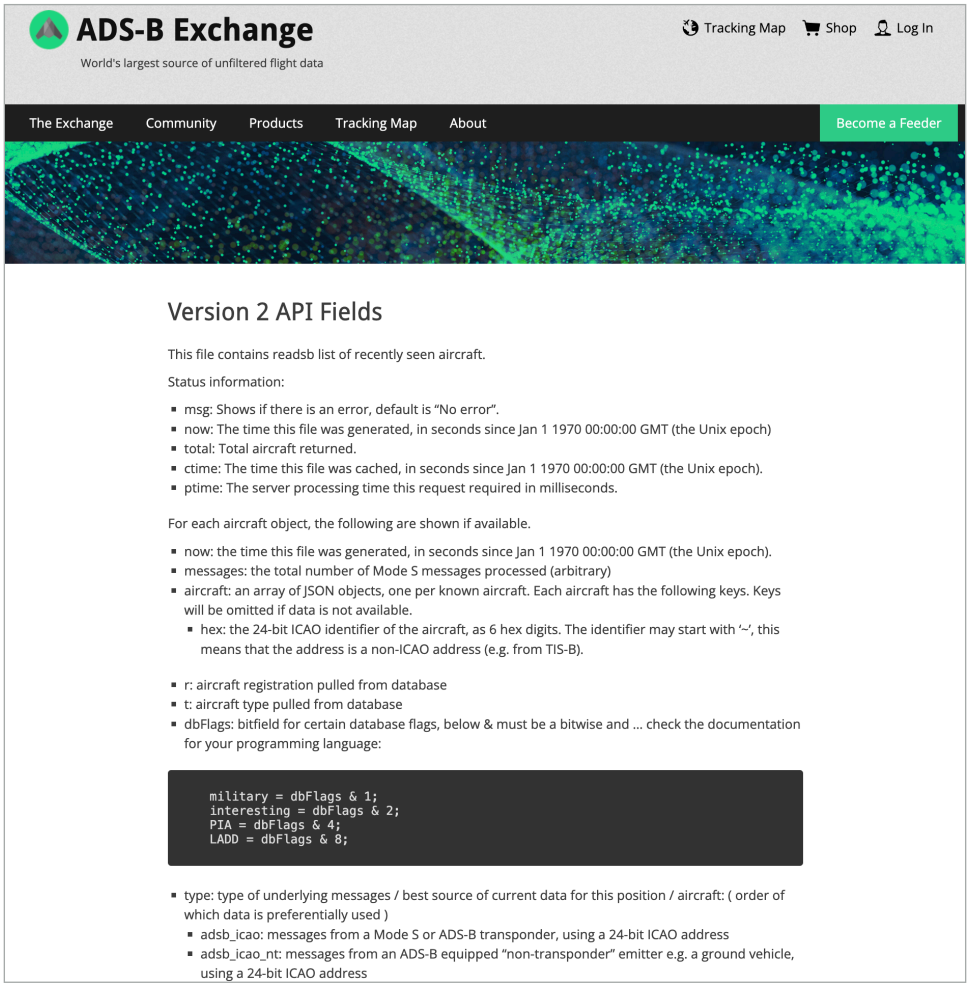




and then, extend this visualization. This second part (Section 3) is completely open-ended, and you can use either D3 or Vega-Lite (or both).

1. Data Structure

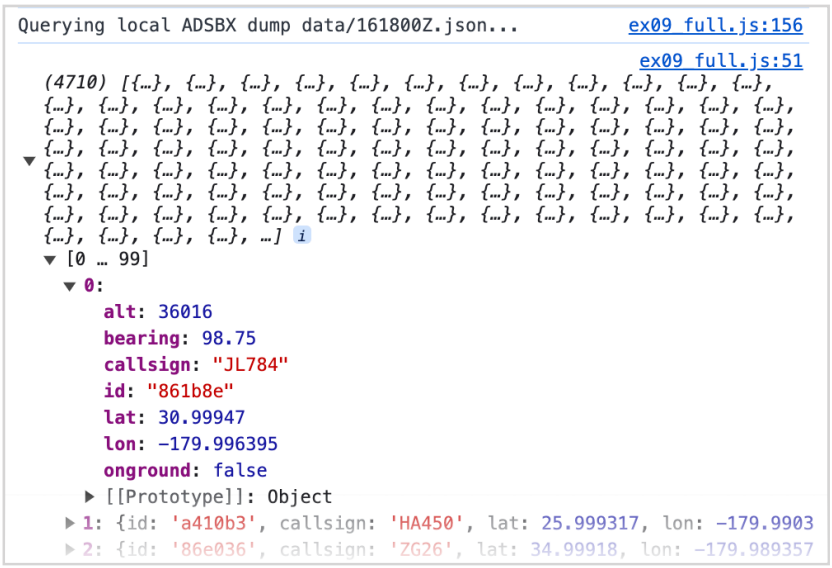
We could fetch live data from free APIs such as `https://opensky-network.org/api/states/all`. But that one is currently unstable. Instead, we use a local dump obtained from a commercial API to avoid service denial (if too many requests): `https://www.adsbexchange.com/version-2-api-wip/`



You start to work in function `loadPlanesFromLocalDump(...)`. That methods gets called at load time with the name of the first local data dump for 2023-11-01T16:18:00. The file is loaded using a `d3.json()` promise as usual. Your job is to transform the data as follows. We will deal with visualization in Section 2.

Filter planes to only keep those flying above `ctx.ALT_FLOOR` (set to 32,000ft). For each plane, create a new object, keeping only the following attributes from the ADS-B feed, and append it to array `ctx.filteredFlights`:

ADSBX API field name	**Name in our data structure**
hex	id
flight	callsign
lat	lat
lon	lon
track	bearing
alt_baro	alt



as illustrated here.

IMPORTANT: write your code so that repeated calls to the method that fetches data from the ADS-B stream will clear & replace the values in `ctx.filteredFlights` with the new ones. **Just remove the old ones and replace them by the new ones.** D3 will handle the data binding update for you, as detailed in Section 2.1.

2. Chart Planes on the Map

The following base map is already drawn for you in function create`Map(...)`:

2.1. Planes

- Put all planes in `<g#planes>`, which has already been created for you as an SVG overlay above the map, binding data to SVG elements using D3 as usual. In this case the SVG elements are of type `<image>`:
  - set their position and orientation by setting the `<image>` element's `transform` attribute's value with `getPlaneTransform(d)`, `d` being the plane datum bound to that `<image>`
  - set their `width` and `height` to `8`
  - set attribute `xlink:href` to `plane_icon.png`
  - set attribute `id` to `p-<callsign>` where `<callsign>` is the actual callsign of that plane

2.2. Live Updates

At this point we have loaded flight data from one file, that corresponds to one time stamp. Now we want to simulate a live data stream. Uncomment the call to `startPlaneUpdater()` in function `loadFlights()`. This will trigger the loading of data for the next time stamp, iteratively, every 5 seconds (for 12 time stamps, at which point it resets back to 2023-11-01T16:18:00).

At each update, you need to handle three cases:

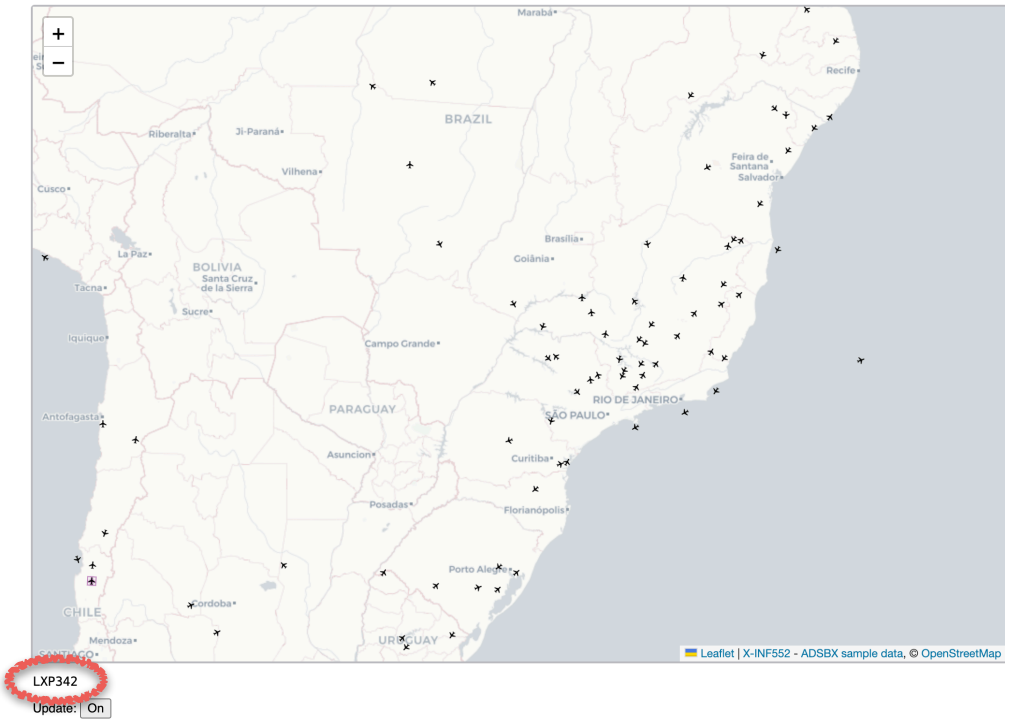
- planes that **were already there** in the previous step, whose datum is already bound to an SVG element:
- planes that **were not there in the previous step**, whose datum is thus not bound to any SVG element;
- planes that **are no longer there** in the new step, which are still bound to an SVG element that should be removed from the DOM.

> **IMPORTANT:** Appendix A (p.7) explains how to do this with `join()` or `enter()` / `exit()`.

In addition, **be careful**, the ordering of planes in the data array is **not guaranteed**. You need to tell D3 how to join data items with SVG elements. As explained in Appendix B, you should specify a unique identifier of your data items to make the binding consistent across updates. Here, choose a data attribute that uniquely identifies each place.

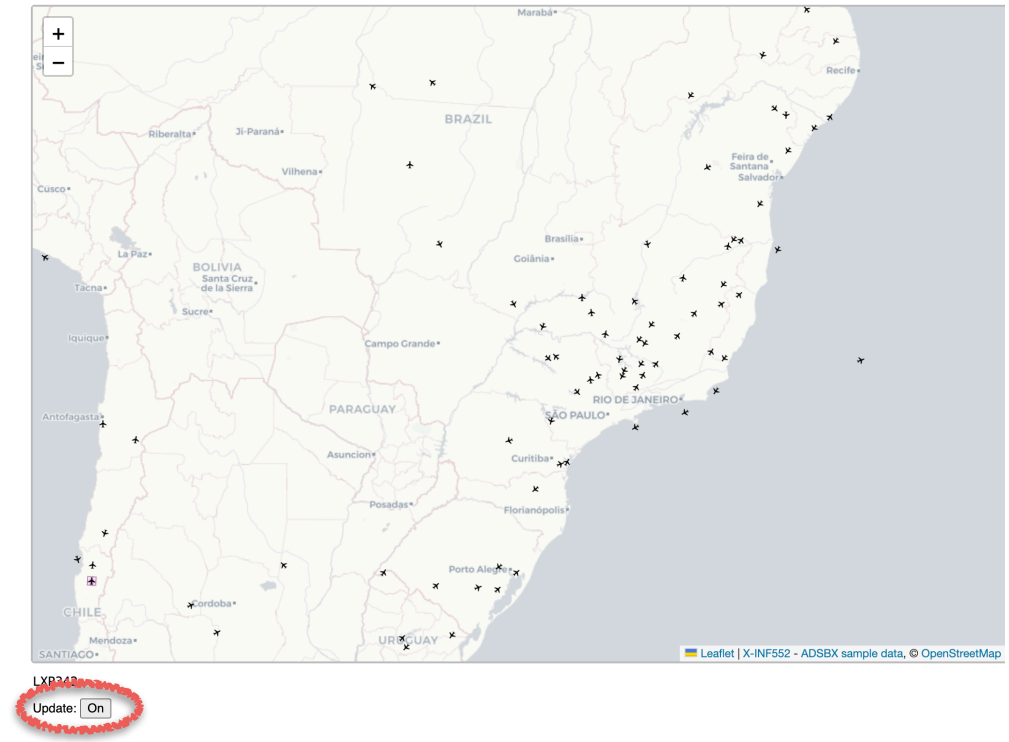
2.3. Tweaks

- Show callsign in `<div#info>` when hovering a plane on the map by writing the contents of function `showDetails(...)`, that gets called whenever users click on the map (the closest plane gets selected).

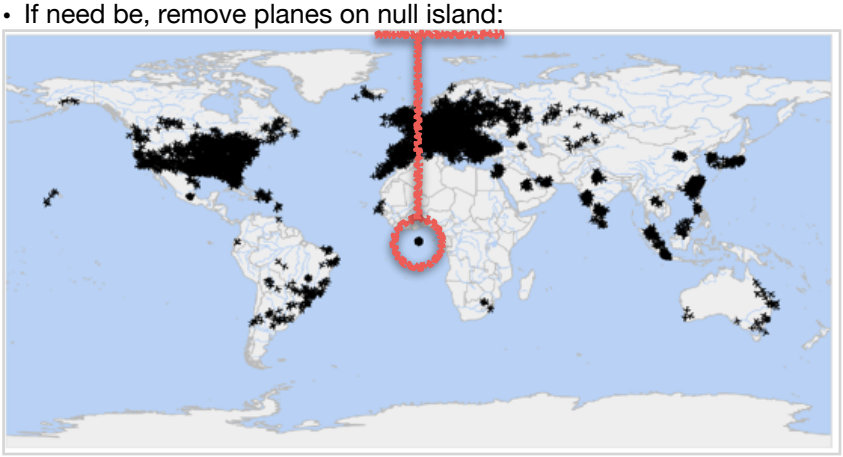


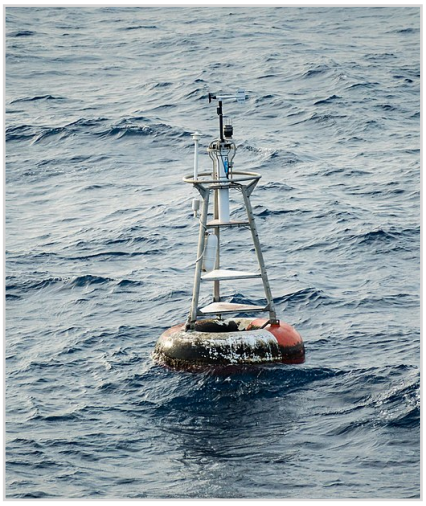
- Make the on/off button actually toggle auto-update every 5 seconds, using JavaScript's `setInterval()` and `clearInterval()` methods.
  See https://www.w3schools.com/jsref/met_win_setinterval.asp

  The toggle button's behavior is already implemented in `toggleUpdate()`



- If need be, remove planes on null island:





More information: https://en.wikipedia.org/wiki/Null_Island

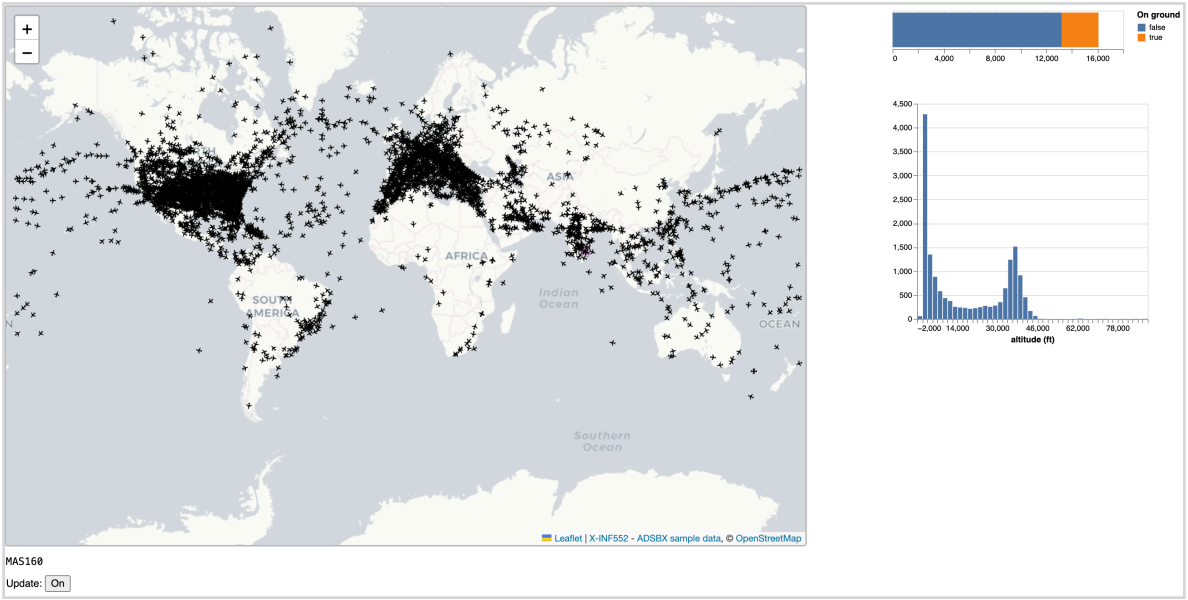
- Animate plane position/orientation updates with a simple `transition()`.

3. Extensions

Extend this visualization at will. This can be:

- either by showing additional information in the map view (additional features, more elaborate visual mappings);
- or designing additional views that provide different perspectives on the data (aggregate views, views showing other attributes, *etc.*);
- or both.

For instance, you can create aggregate visualizations and think of possible coordination between the views. In the basic example below, clicking somewhere/making a selection in one view updates the other views accordingly (*e.g.*, only show planes on the ground by clicking on the orange rectangle).



Useful resources:
https://d3js.org/d3-selection/events#selection_on
https://vega.github.io/vega/docs/api/view/#view_addEventListener

More suggestions on the next page...

Additional information about individual planes is available for that day (2023-11-01) in individual JSON trace files, made available to you as follows:

- sample dataset (all planes with ICAO ending with `16`) at:
  https://moodle.polytechnique.fr/mod/resource/view.php?id=448957
- full dataset (all planes) on demand (2.54GB from a USB stick).

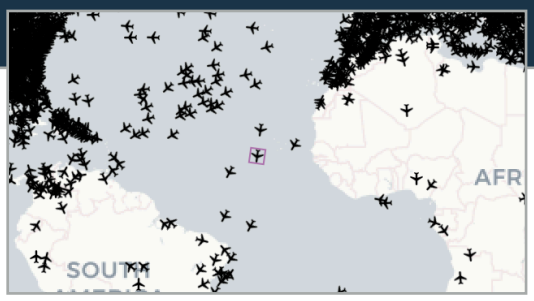
Within those archives, the trace for a plane whose ICAO is `0123ac` (hex) will be located in:
`2023/11/01/ac/trace_full_0123ac.json`

You can think about visualizations of these additional attributes. See https://www.adsbexchange.com/version-2-api-wip/ (see **Trace File Fields** for more detail).

```
{
    icao: "0123ac", // hex id of the aircraft
    timestamp: 1609275898.495, // unix timestamp in seconds since epoch (1970)
    trace: [
        [ seconds after timestamp,
            lat,
            lon,
            altitude in ft or "ground" or null,
            ground speed in knots or null,
            track in degrees or null, (if altitude == "ground", this will be true heading instead of track)
            flags as a bitfield: (use bitwise and to extract data)
                (flags & 1 > 0): position is stale (no position received for 20 seconds before this one)
                (flags & 2 > 0): start of a new leg (tries to detect a separation point between landing and takeoff that separates fligths)
                (flags & 4 > 0): vertical rate is geometric and not barometric
                (flags & 8 > 0): altitude is geometric and not barometric
             ,
            vertical rate in fpm or null,
            aircraft object with extra details or null (see aircraft.json documentation, note that not all fields are present as lat and lon for example
arlready in the values above),
            // the following fields only in files generated 2022 and later:
            type / source of this position or null,
            geometric altitude or null,
            geometric vertical rate or null,
            indicated airspeed or null,
            roll angle or null
        ],
        [next entry like the one before],
        [next entry like the one before],
    ]
}
```

Example (`2023/11/01/16/trace_full_484416.json`):


```
{"icao":"484416",
"r":"PH-BQM",
"t":"B772",
"dbFlags":0,
"desc":"BOEING 777-200",
"timestamp": 1698796800.000,
"trace":[
[7.13,53.091663,-99.354935,35000,504.8,77.1,0,0,null,"other",34000,null,null,null],
[26.57,53.104791,-99.259697,35000,504.8,77.1,0,0,{"type":"adsb_icao","flight":"KLM678
","alt_geom":34000,"track":77.06,"baro_rate":0,"squawk":"1317","emergency":"none","category":"A5","nav_qnh":1012.8,"nav_altitude_m
cp":35008,"nav_heading":57.66,"nic":8,"rc":186,"version":2,"nic_baro":1,"nac_p":10,"nac_v":2,"sil":3,"sil_type":"perhour","gva":2,
"sda":2,"alert":0,"spi":0},"adsb_icao",34000,null,null,null],
[46.43,53.115172,-99.184570,35000,504.8,77.1,0,0,null,"adsb_icao",34000,null,null,null],
[66.16,53.125414,-99.110574,35000,505.6,77.2,0,0,null,"adsb_icao",33975,null,null,null],
[83.61,53.134415,-99.044652,35000,505.6,77.2,0,0,null,"adsb_icao",33975,null,null,null],
[95.21,53.140457,-99.000157,35000,505.3,77.3,0,0,{"type":"adsb_icao","flight":"KLM678
```

A. D3's enter/update/exit pattern

A.1. General Update Pattern

`d3.selectAll(…).data(…).enter()`: selects items that are not yet bound:

```
d3.selectAll("circle")
  .data(someArray)
  .enter()
  .append("circle")
  .attr("cx", 200)
  .attr("cy", (d,j) => (j*42))
  .attr("r", 10)
  .attr("fill", (d) => (someScale(d)));
```

`d3.selectAll(…).data(…).exit()`: selects items that are no longer bound:

```
d3.selectAll("circle")
  .data(smallerArray)
  .exit()
  .remove();
```

actually removes the element from the DOM

Be careful: enter/update/exit subsets will vary if you call d3.selectAll(...).data(...) independently for each of them in sequence. You will likely want to work on those different subsets with the same initial selection. You should thus first store the selection itself:

```
var myCircles = d3.selectAll("circle")
                  .data(newArray);
```

and then work with the different subsets from that selection. For instance:

```
myCircles.attr(...)
         .style(...);

myCircles.enter(...)
         .append("circle")
         .attr(...);

myCircles.exit()
         .remove();
```

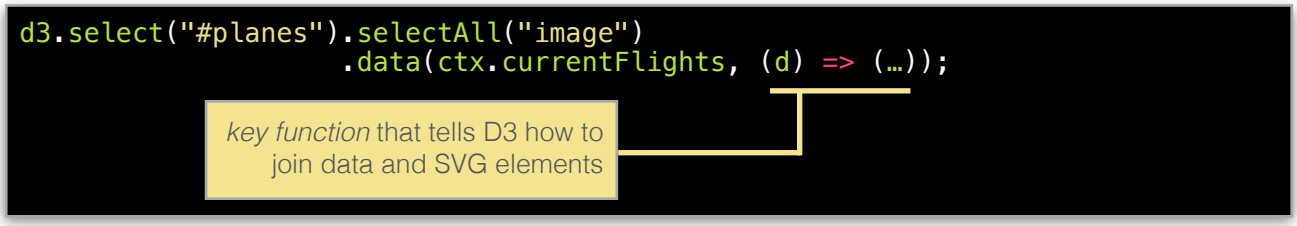
A.2. Selection Join

Instead of the general update pattern presented above, it is possible to use `d3-selection.join()`, as described at https://d3js.org/d3-selection/joining#selection_join

```
d3.selectAll("circle")
  .data(someArray)
  .join(
     enter => (
        enter.append("circle")
             .attr(...)
     ),
     update => (
        update.attr(...).style(...)
     ),
     exit => (
        exit.remove()
     )
  );
```

B. Overriding how D3 joins data items and SVG elements

When the ordering of items in the data array is not guaranteed, you should join data by key, not by index.



The arrow function should return an identifier unique to each item in the array.