Info-H-415 - Advanced Databases

Mobility (Moving Objects) Databases

Lesson 1: Introduction and Motivation

Alejandro Vaisman
avaisman@itba.edu.ar

Schedule and Program

- Lesson 1 (10/10/23) - Introduction – Mobility Data Science (lesson1.pdf)
- Lesson 2 (12/10/23) - T: The abstract model – Predicates (lesson2.pdf)
- Lesson 3 (16 /10/23) - T: The discrete model (Lesson 2.pdf)
- Practice 1 (19/10/23) - (Assignment 1 - mobilitydb - datamodel.pdf)
- Practice 2 (23/10/23) - (Assignment 2 - mobilitydb - AIS.pdf)
- Practice 3 (26/10/23) - (Assignment 3 - mobilitydb - OpenSky.pdf)

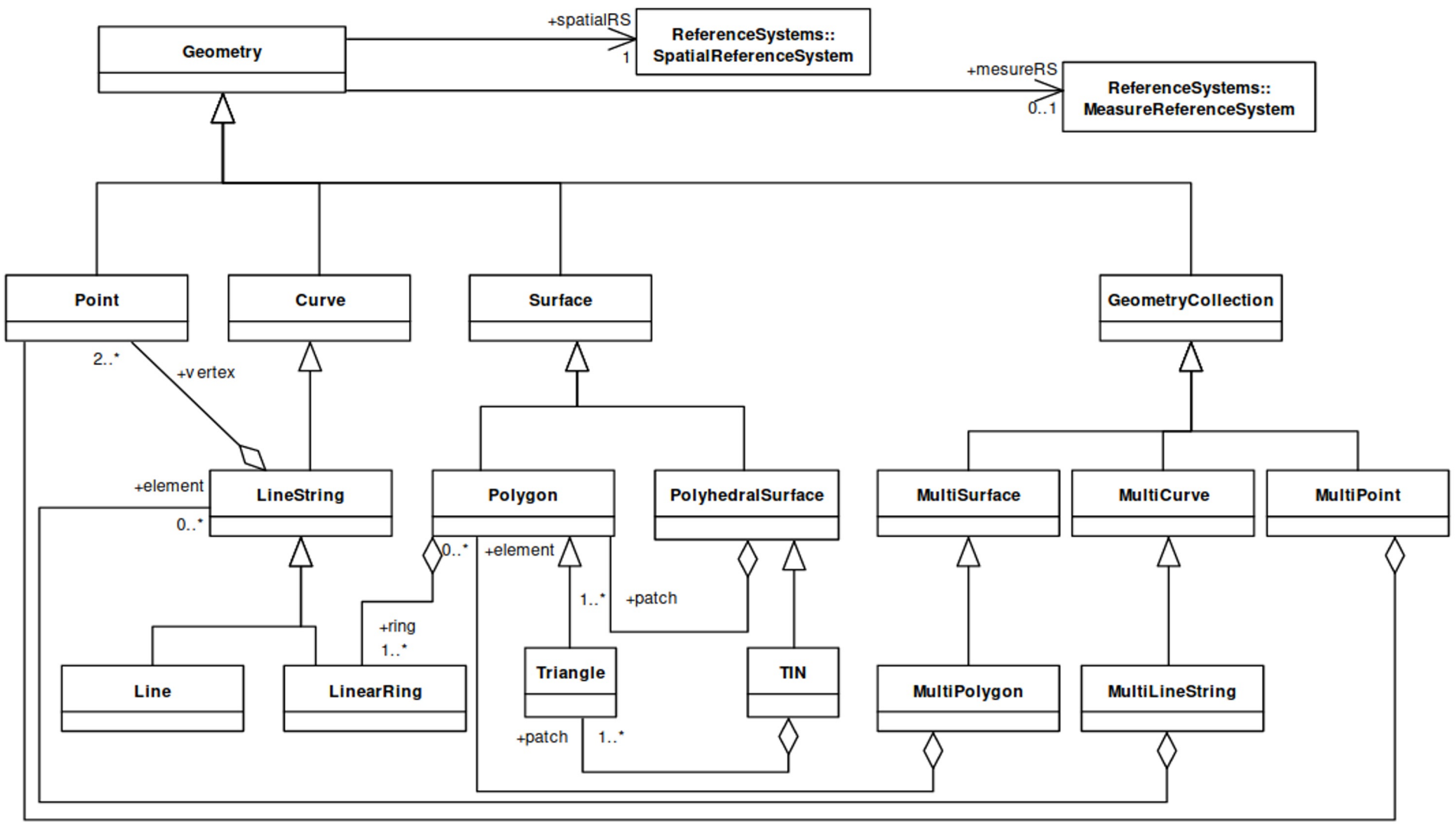
Note

- The slides in this course borrow material from the book “Moving Object Databases”, Ralph Güting and Markus Schneider, Morgan-Kaufmann, 2003. This material was kindly provided by Ralph Güting.
- Part of the slides have been developed in collaboration with Mahmoud Sakr and Esteban Zimanyi at ULB, and also used in the course “GeoSpatial and web technologies” lectured by Mahmoud Sakr at Université Libre de Bruxelles (ULB)

Spatial data

- Vector
  - Geometric objects representing the spatial features in the maps
  - Generated by Vectorization and Positioning technologies
- Raster
  - Precipitation, temperature
  - Satellite and other remote sensing
  - Elevation (altitude)
  - Forests, agriculture
- Thematic data
  - Attributes that give meaning to the spatial data, such as ‘restaurant’, opening-hours, reviews, etc.

Vector data


Geometry
+spatialRS
1
ReferenceSystems::
SpatialReferenceSystem
+mesureRS
0..1
ReferenceSystems::
MeasureReferenceSystem
Point
Curve
Surface
GeometryCollection
2..*
+vertex
+element
0..*
LineString
Polygon
PolyhedralSurface
MultiSurface
MultiCurve
MultiPoint
0..*
+element
1..*
+patch
+ring
1..*
Line
LinearRing
Triangle
TIN
MultiPolygon
MultiLineString
+patch
1..*

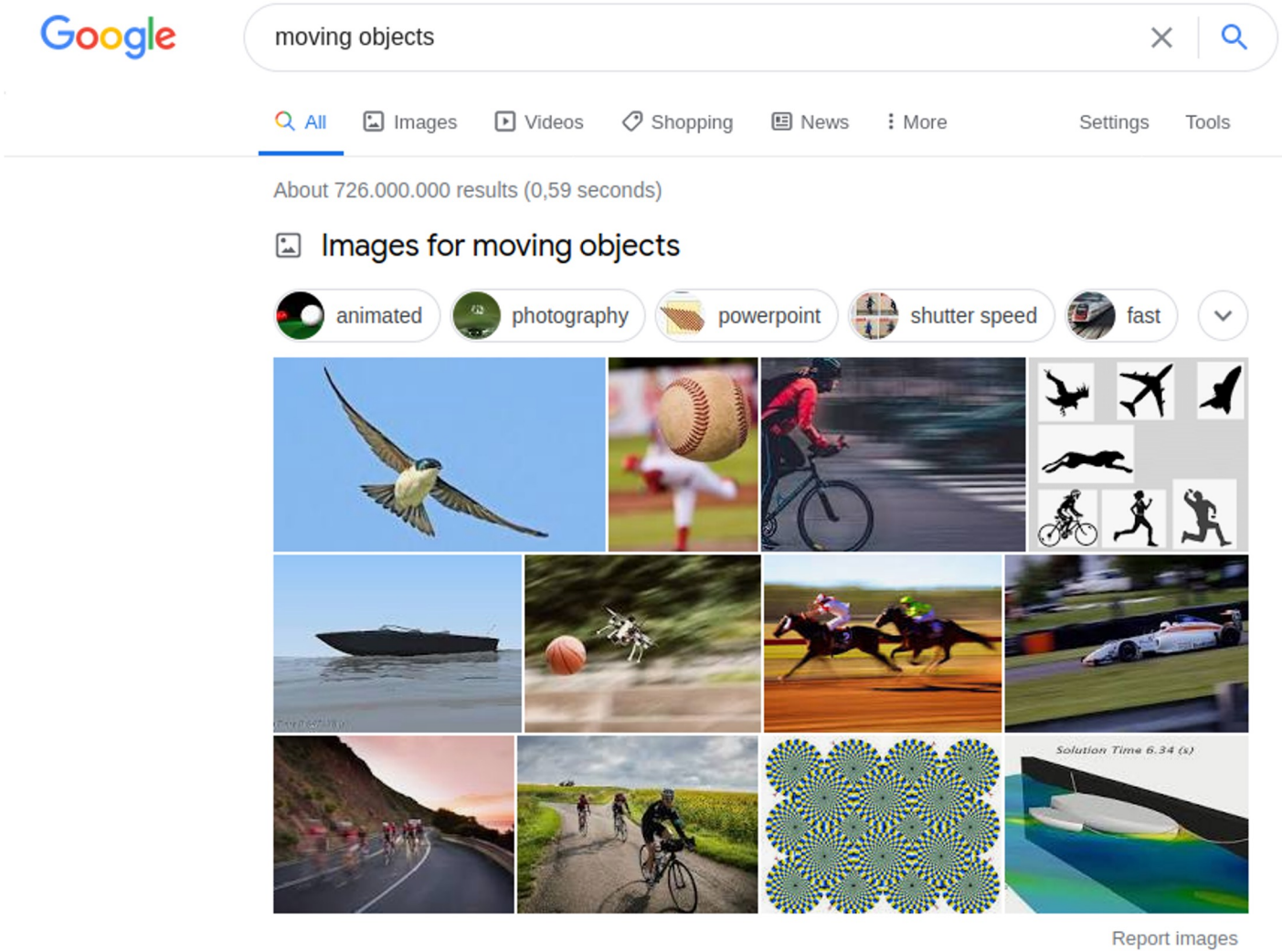
Temporal data

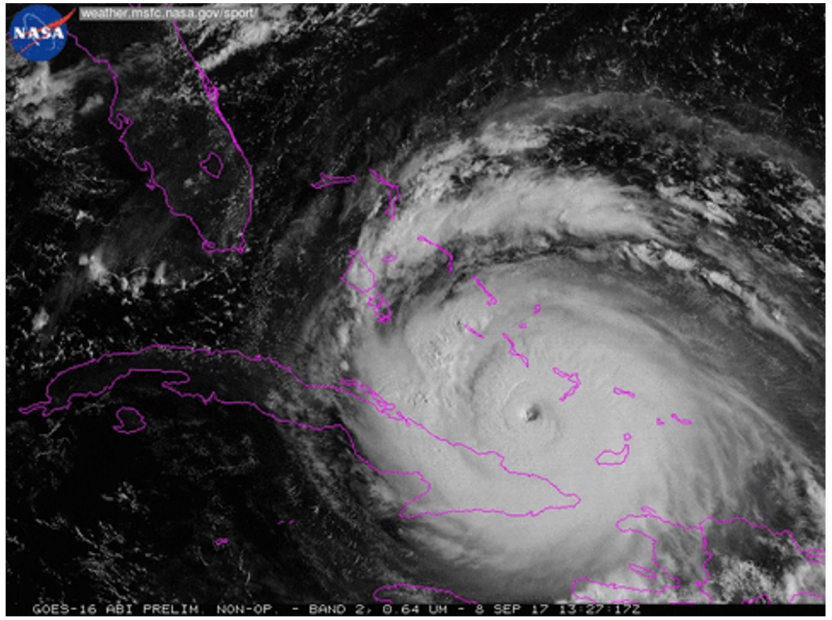
- Temporal database research deals with the problem of studying storing and querying temporal data
- Examples:
  - Evolution of salaries, population, etc., across time
- Temporal database modeling relevant concepts:
  - Valid time
  - Transaction time
  - 1NF and non-1NF models

Spatio-temporal data

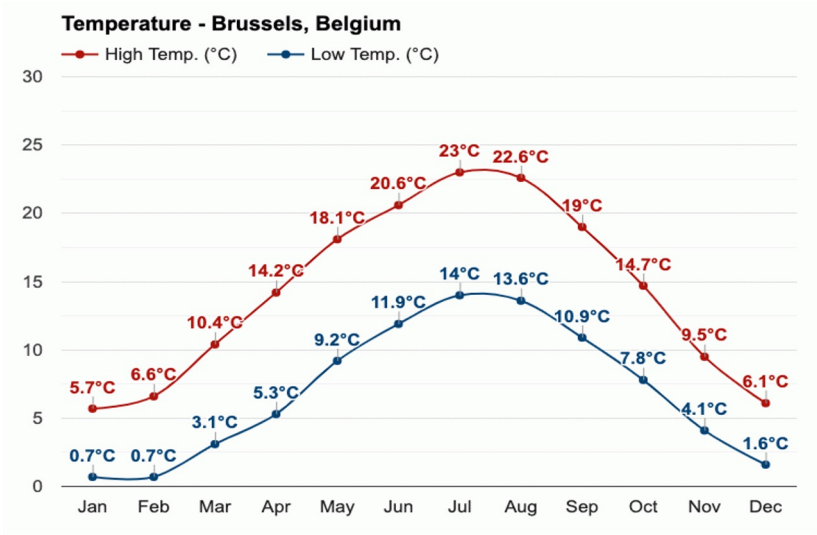
- Temporal + spatial data
- Objects can move in space and time (points, regions, evolve across time)
- Temporal database concepts applied to spatial data
- We will address temporal data in general (e.g., spatial and non-spatial data)
- We will also study MobilityDB, an extension to PostgreSQL and PostGIS for representing, managing and querying these kinds of data
- Let us start with the notion of moving objects……

Moving Objects





giphy.com



weather-atlas.com

Moving Objects - Points

- Extend database technology so that any kind of moving entity can be represented in a database
- High-level query languages allow formulating queries about such movement

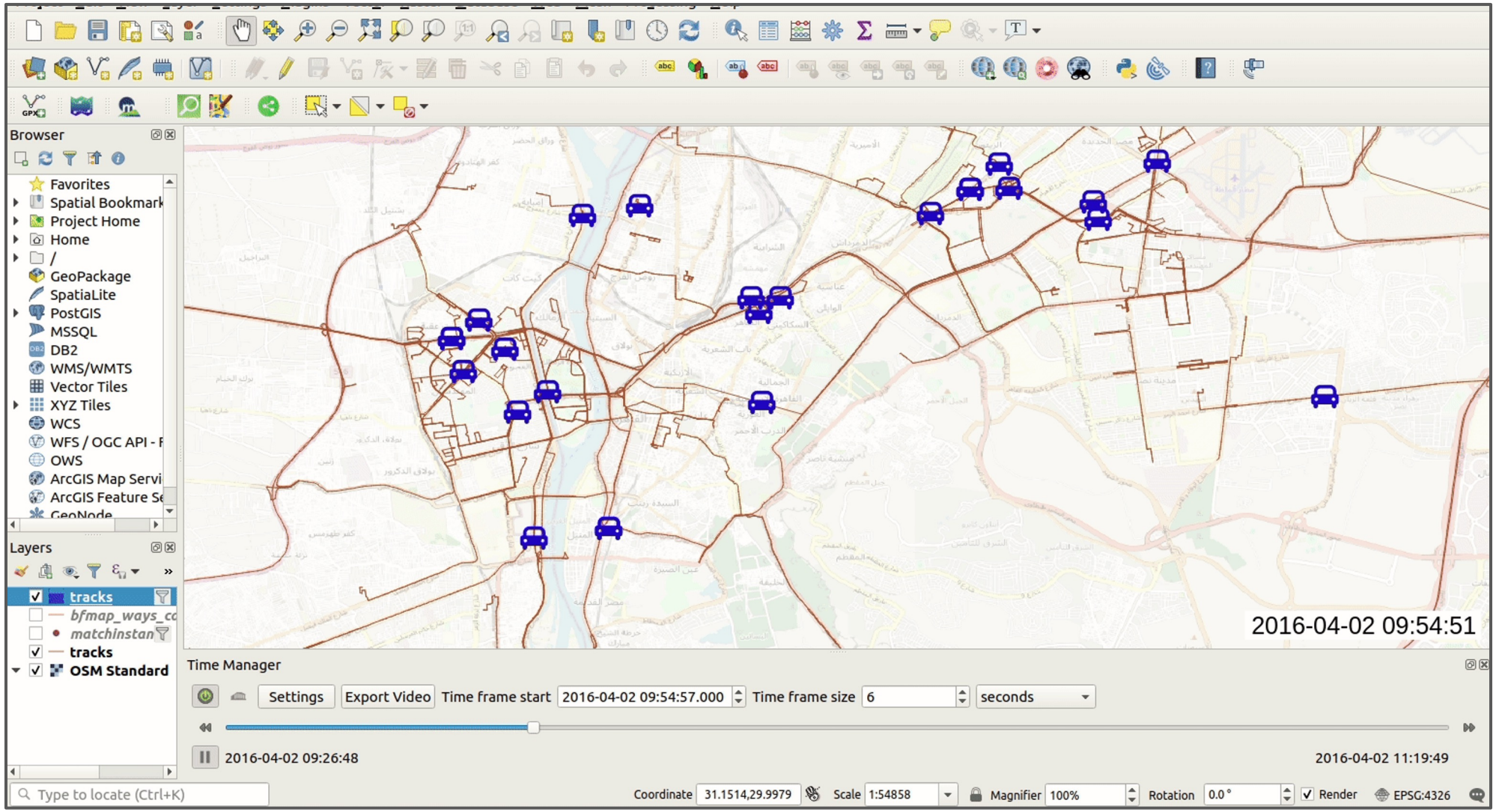
Animals	• Which distance do they traverse, at which speed? How often do they stop? • Determine trajectories of birds, whales, ... • Where are the whales now?
Cars: taxi-cabs, trucks	• Which taxi is closest to a passenger request position? • Which routes are used regularly by trucks? • Did the trucks with dangerous goods ever come close to a high risk facility?
Airplanes	• Are two planes heading towards each other (going to crash)? • Were any two planes close to a collision? • Did planes cross the air territory of a given country? • At what speed does this plane move? What is its top speed?

Moving Objects - Regions

- Extend database technology so that any kind of moving entity can be represented in a database
- High-level query languages allow formulating queries about such movement

Forests, lakes	• How fast is the Amazon rain forest shrinking? • Is the dead sea shrinking? • What is the minimal and maximal extent of river X during the year?
Glaciers	• Does the polar ice cap grow? Does it move? • Where must glacier X have been at time Y (backward projection)?
Diseases	• Show the area affected by mad cow disease for every month in 1998

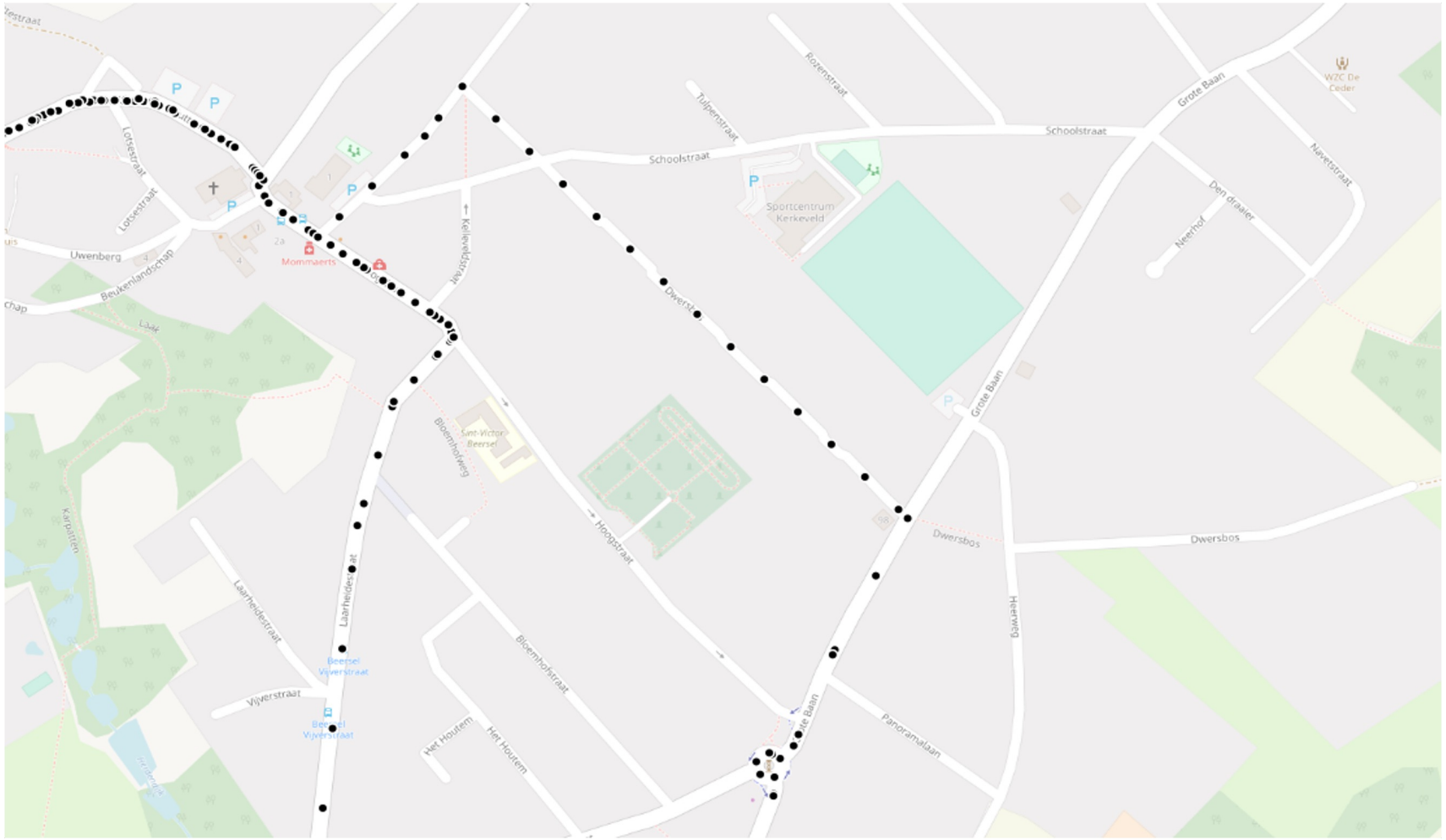
Moving Objects



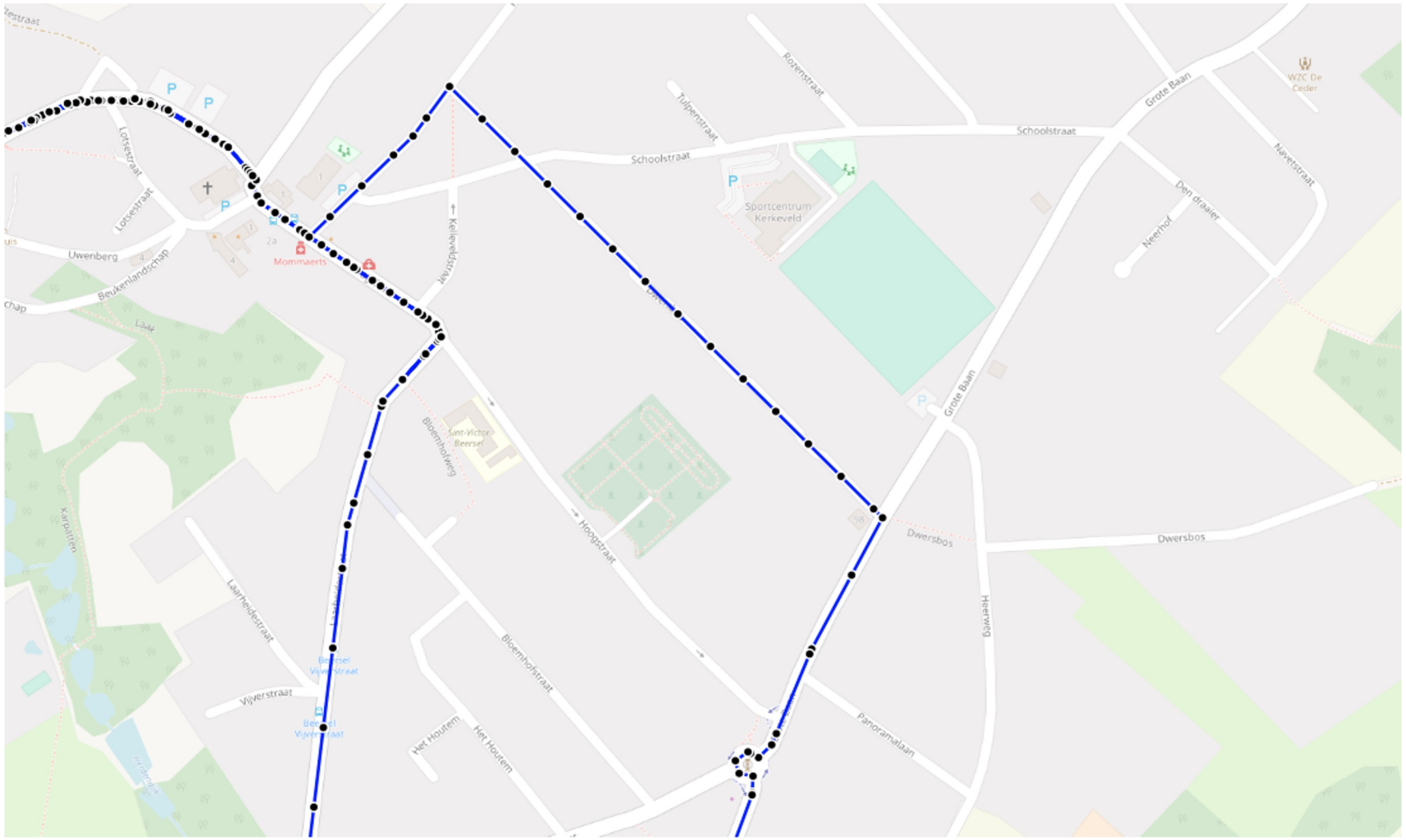
Mobility Data: GPS, DCM (data collection mechanism)

	vehicle integer	day date	seq integer	source bigint	target bigint	t timestamp with time zone	geom text
370	1	2020-06-01	1	16690	34728	2020-06-01 10:01:20.978+02	POINT(490985.514477288 6604283.85305868)
371	1	2020-06-01	1	16690	34728	2020-06-01 10:01:28.081029+02	POINT(490956.213031904 6604299.02553383)
372	1	2020-06-01	1	16690	34728	2020-06-01 10:01:30.978+02	POINT(490940.739620434 6604303.45266601)
373	1	2020-06-01	1	16690	34728	2020-06-01 10:01:34.491879+02	POINT(490921.971156538 6604308.82255297)
374	1	2020-06-01	1	16690	34728	2020-06-01 10:01:39.062744+02	POINT(490911.18429788 6604299.23736115)
375	1	2020-06-01	1	16690	34728	2020-06-01 10:01:40.978+02	POINT(490901.048941147 6604301.05718891)
376	1	2020-06-01	1	16690	34728	2020-06-01 10:01:42.592551+02	POINT(490892.504887324 6604302.5912944)
377	1	2020-06-01	1	16690	34728	2020-06-01 10:01:47.131132+02	POINT(490872.022101019 6604317.29566003)
378	1	2020-06-01	2	34728	16690	2020-06-01 17:53:26.791+02	POINT(490872.022101019 6604317.29566003)
379	1	2020-06-01	2	34728	16690	2020-06-01 17:53:31.929581+02	POINT(490892.504887324 6604302.5912944)
380	1	2020-06-01	2	34728	16690	2020-06-01 17:53:36.791+02	POINT(490910.008155174 6604299.44854041)
381	1	2020-06-01	2	34728	16690	2020-06-01 17:53:37.117666+02	POINT(490911.18429788 6604299.23736115)
382	1	2020-06-01	2	34728	16690	2020-06-01 17:53:39.828856+02	POINT(490921.971156538 6604308.82255297)
383	1	2020-06-01	2	34728	16690	2020-06-01 17:53:46.239706+02	POINT(490956.213031904 6604299.02553383)
384	1	2020-06-01	2	34728	16690	2020-06-01 17:53:46.791+02	POINT(490958.875710594 6604297.64678183)
385	1	2020-06-01	2	34728	16690	2020-06-01 17:53:56.791+02	POINT(491007.174416728 6604272.63740401)
386	1	2020-06-01	2	34728	16690	2020-06-01 17:54:06.791+02	POINT(491055.473122862 6604247.62802619)
387	1	2020-06-01	2	34728	16690	2020-06-01 17:54:10.336527+02	POINT(491072.597559528 6604238.76088375)

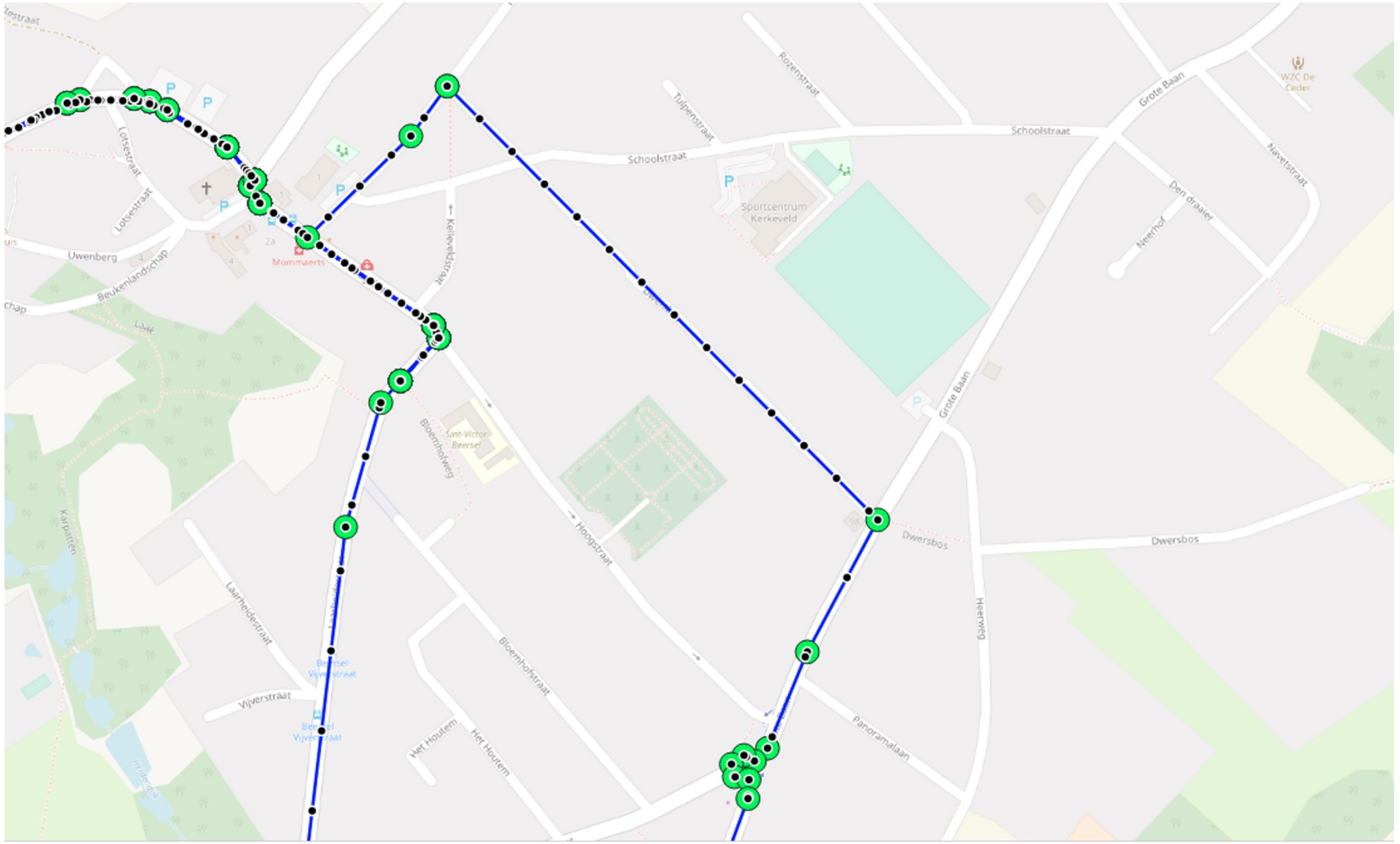
Mobility Data: GPS, DCM (data collection mechanism)



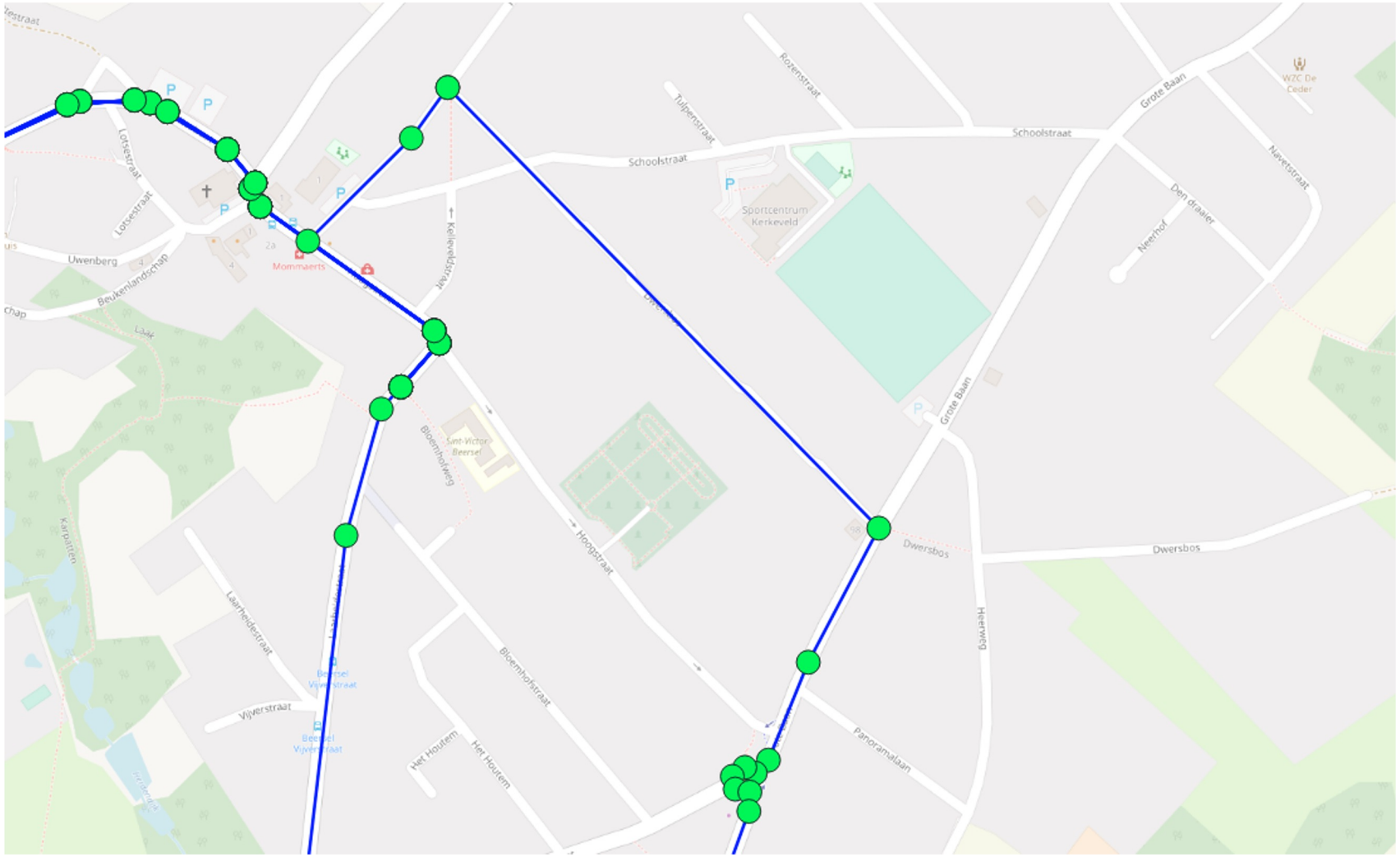
Building trajectories



Compact Format (MobilityDB)



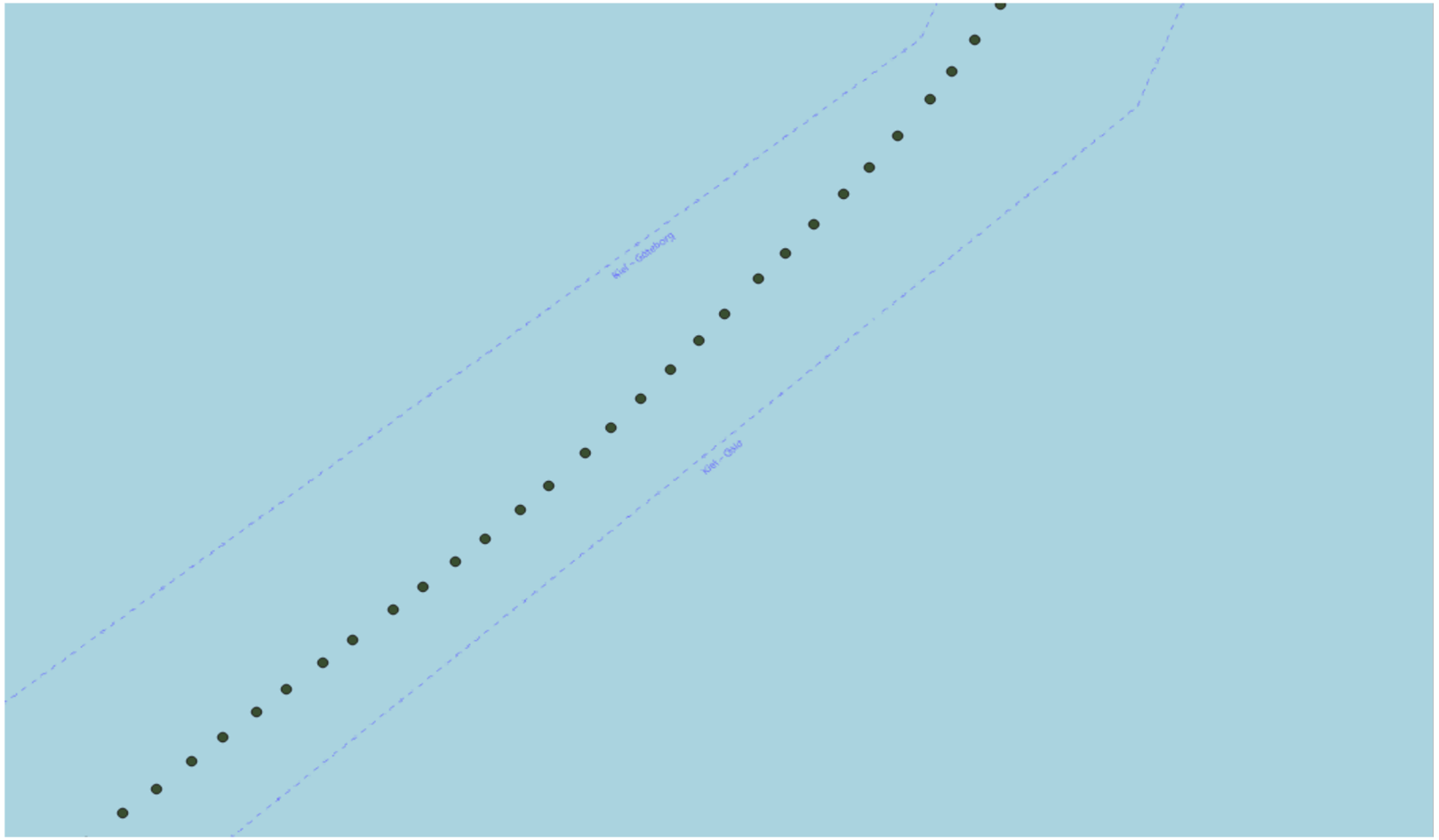
Compact Format (MobilityDB)



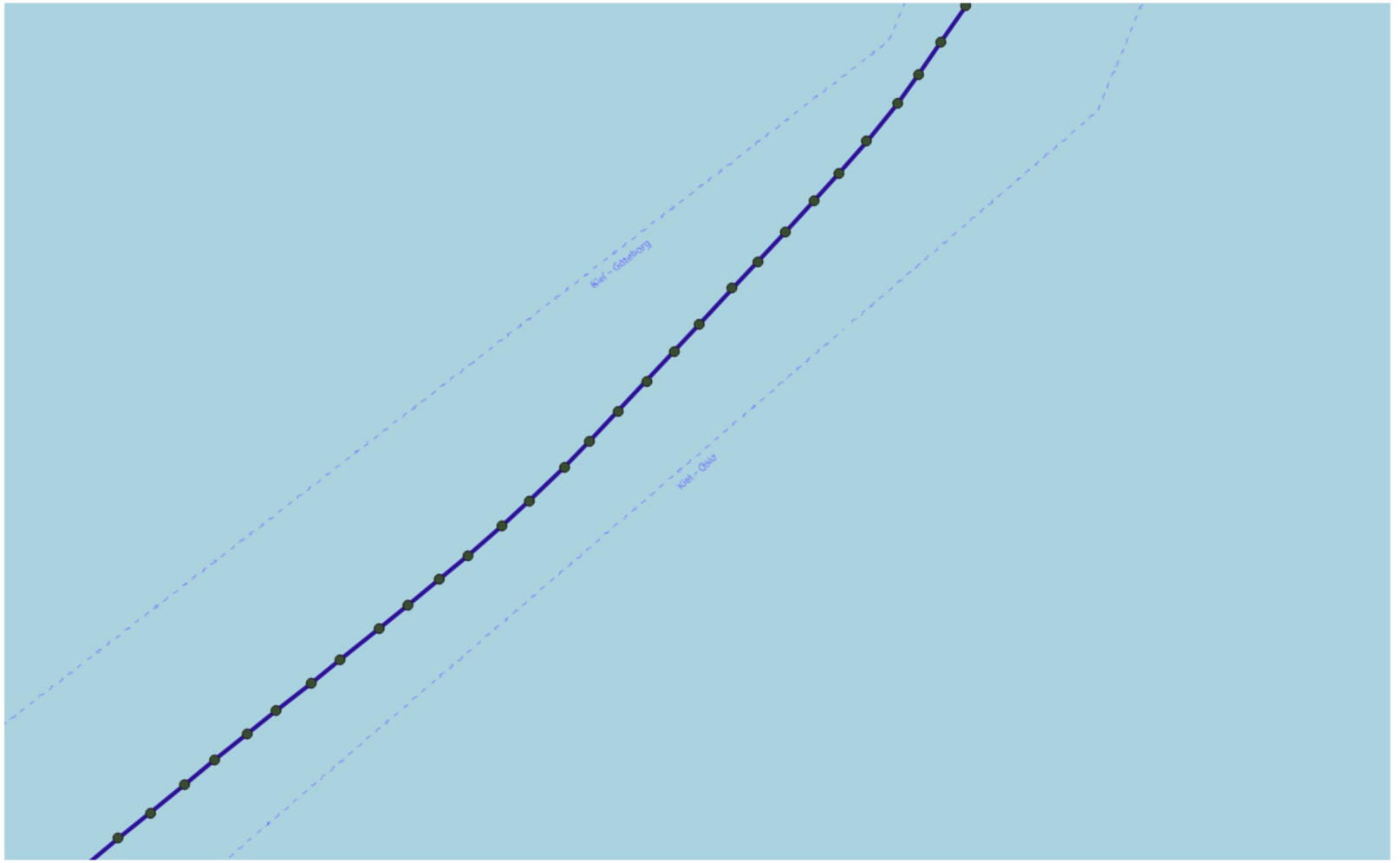
Another example: Mobility Data: AIS (Automatic Identification System)

	t	mmsi	latitude	longitude	cog	sog	sizea
1	2021-10-29 01:40:29.000000	538009363	55.344313	14.46848	219.1	11.7	149
2	2021-10-29 01:11:11.000000	538009363	55.417545	14.573578	219.5	11.3	149
3	2021-10-29 01:46:20.000000	538009363	55.329398	14.447498	216.8	11.7	149
4	2021-10-29 02:11:09.000000	538009363	55.264513	14.359782	219.2	11.7	149
5	2021-10-29 01:10:39.000000	538009363	55.418805	14.575425	221.1	11.3	149
6	2021-10-29 00:14:31.000000	538009363	55.517727	14.844517	239	11.9	149
7	2021-10-29 01:45:11.000000	538009363	55.332392	14.451415	217.8	11.7	149
8	2021-10-29 00:52:40.000000	538009363	55.453145	14.655757	241.7	11.8	149
9	2021-10-29 02:00:01.000000	538009363	55.293733	14.399363	217.7	11.9	149
10	2021-10-29 00:17:49.000000	538009363	55.512148	14.827812	239.9	11.9	149
11	2021-10-29 01:49:30.000000	538009363	55.321232	14.436467	217.1	11.7	149
12	2021-10-29 00:03:00.000000	538009363	55.537168	14.903507	241	12.1	149
13	2021-10-29 02:00:39.000000	538009363	55.292023	14.397023	217.8	11.9	149
14	2021-10-29 02:13:39.000000	538009363	55.257973	14.351725	216	11.3	149
15	2021-10-29 00:00:30.000000	538009363	55.54139	14.916162	238.4	12	<null>
16	2021-10-29 02:05:01.000000	538009363	55.280447	14.381715	217.4	11.9	149
17	2021-10-29 00:53:40.000000	538009363	55.451593	14.650627	241.6	11.8	149
18	2021-10-29 00:41:11.000000	538009363	55.472212	14.713127	239.4	11.8	149
19	2021-10-29 02:10:41.000000	538009363	55.265733	14.361543	220.3	11.8	149
20	2021-10-29 01:42:29.000000	538009363	55.339198	14.461327	219.4	11.7	149
21	2021-10-29 01:38:11.000000	538009363	55.350108	14.476952	219.3	11.7	149
22	2021-10-29 00:17:20.000000	538009363	55.512967	14.83034	240.2	11.9	149
23	2021-10-29 01:56:29.000000	538009363	55.303013	14.411842	218.6	11.9	149
24	2021-10-29 00:32:30.000000	538009363	55.48698	14.755772	238.2	11.6	149

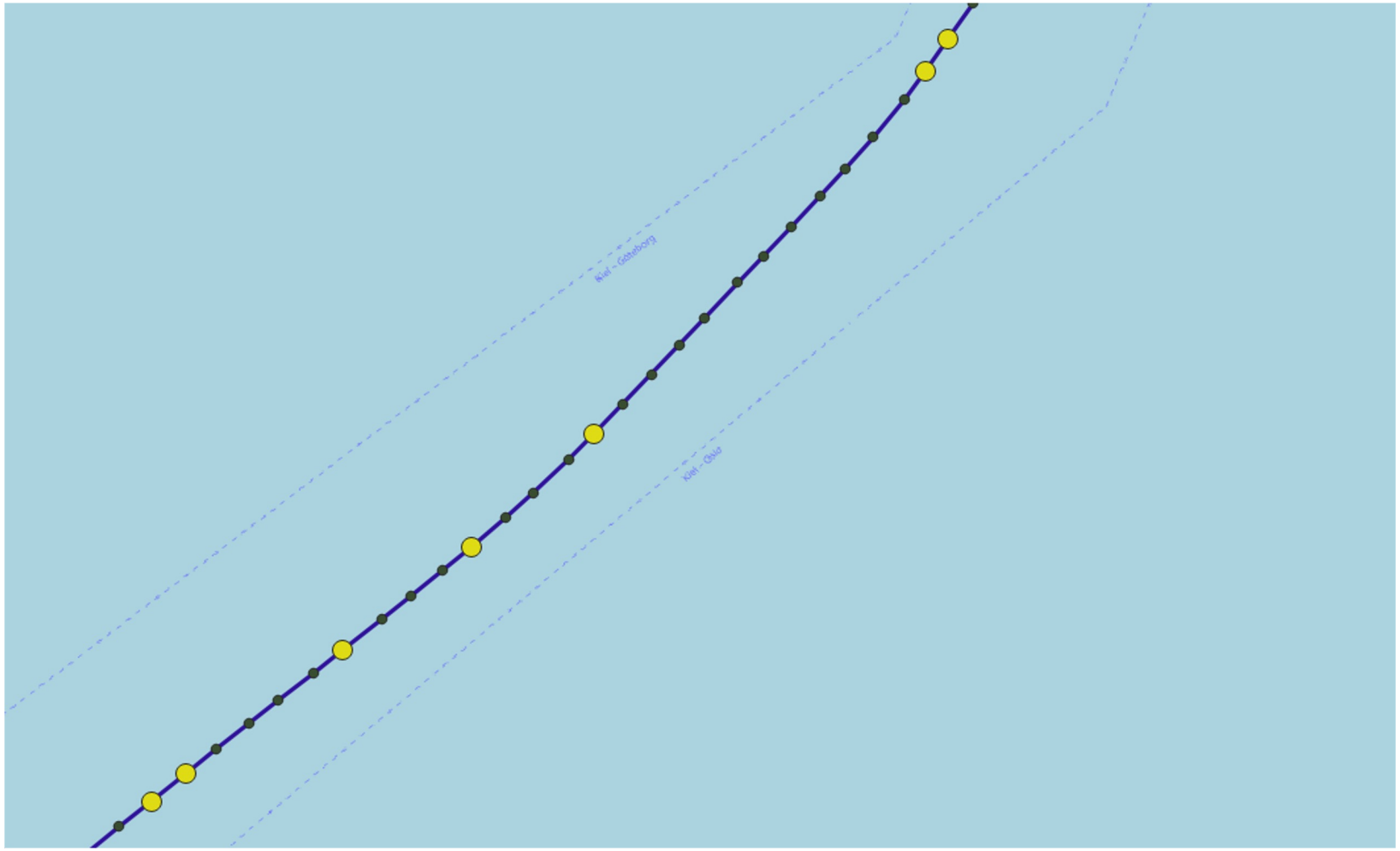
Mobility Data: AIS



MobilityDB Compact Format



MobilityDB Compact Format



PostGIS -> MobilityDB migration

44384	2015-04-06 06:38:00	POINT(37.3826816 55.7937783)
44384	2015-04-06 06:38:30	POINT(37.3826816 55.7937783)
44384	2015-04-06 06:39:00	POINT(37.3826816 55.7937783)
44384	2015-04-06 06:39:30	POINT(37.3826816 55.7937783)
44384	2015-04-06 06:40:00	POINT(37.3826816 55.7937783)
44384	2015-04-06 06:40:30	POINT(37.3826816 55.7937783)
44384	2015-04-06 06:41:00	POINT(37.3826816 55.7937783)
44384	2015-04-06 06:41:30	POINT(37.3826816 55.7937783)
44384	2015-04-06 06:42:00	POINT(37.3826816 55.7937783)
44384	2015-04-06 06:42:30	POINT(37.3826816 55.7937783)

44384	[POINT(37.3826816 55.7937783)@2015-04-06 06:38:00+03, POINT(37.3826816 55.7937783)@2015-04-06 0...
44399	[POINT(37.6126166 55.7274032)@2015-04-06 07:14:29+03, POINT(37.6118683 55.7274732)@2015-04-06 0...
44399	[POINT(37.6127783 55.7265099)@2015-04-06 05:32:14+03, POINT(37.6127783 55.7265099)@2015-04-06 0...
62736	[POINT(37.6078283 55.7158566)@2015-04-06 05:35:17+03, POINT(37.607475 55.71504)@2015-04-06 05:35:...
62771	[POINT(37.6124233 55.7264416)@2015-04-06 05:07:57+03, POINT(37.6124233 55.7264416)@2015-04-06 0...
67756	[POINT(37.608135 55.7163933)@2015-04-06 04:47:23+03, POINT(37.60777 55.7153983)@2015-04-06 04:47:...
67762	[POINT(37.6093483 55.7190449)@2015-04-06 16:58:07+03, POINT(37.6094966 55.7188982)@2015-04-06 1...
67762	[POINT(37.6099266 55.7209099)@2015-04-06 04:41:30+03, POINT(37.60921 55.7190516)@2015-04-06 04:4...

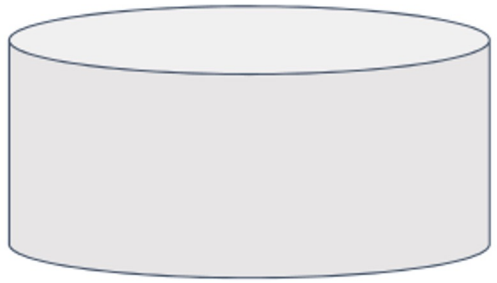
10 billion rows a day

> 500 MB per day

~ 300 GB per year

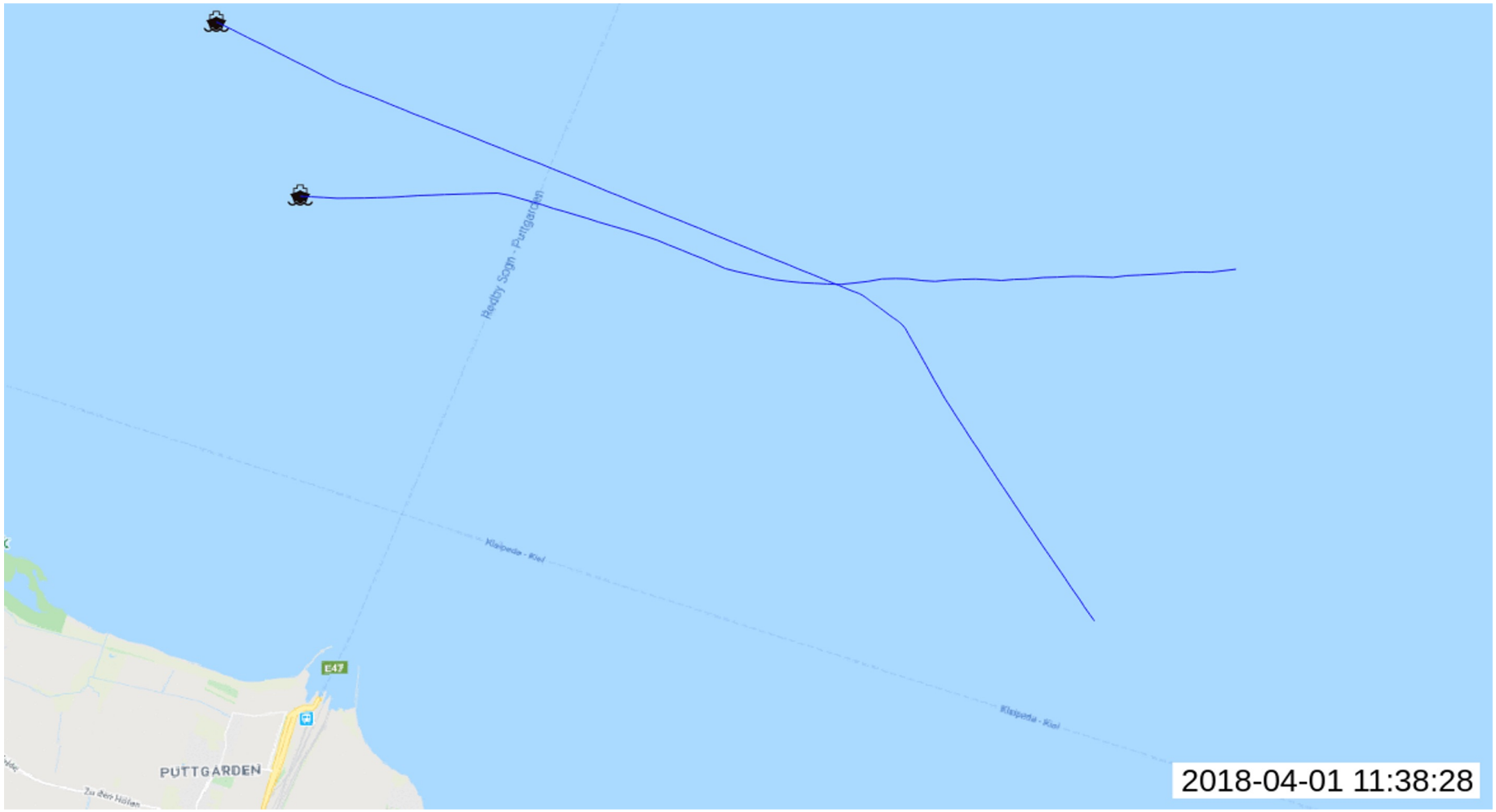
15 thousand rows

~ 5 MB per day



2GB per year

Spatiotemporal Proximity: Marine Data

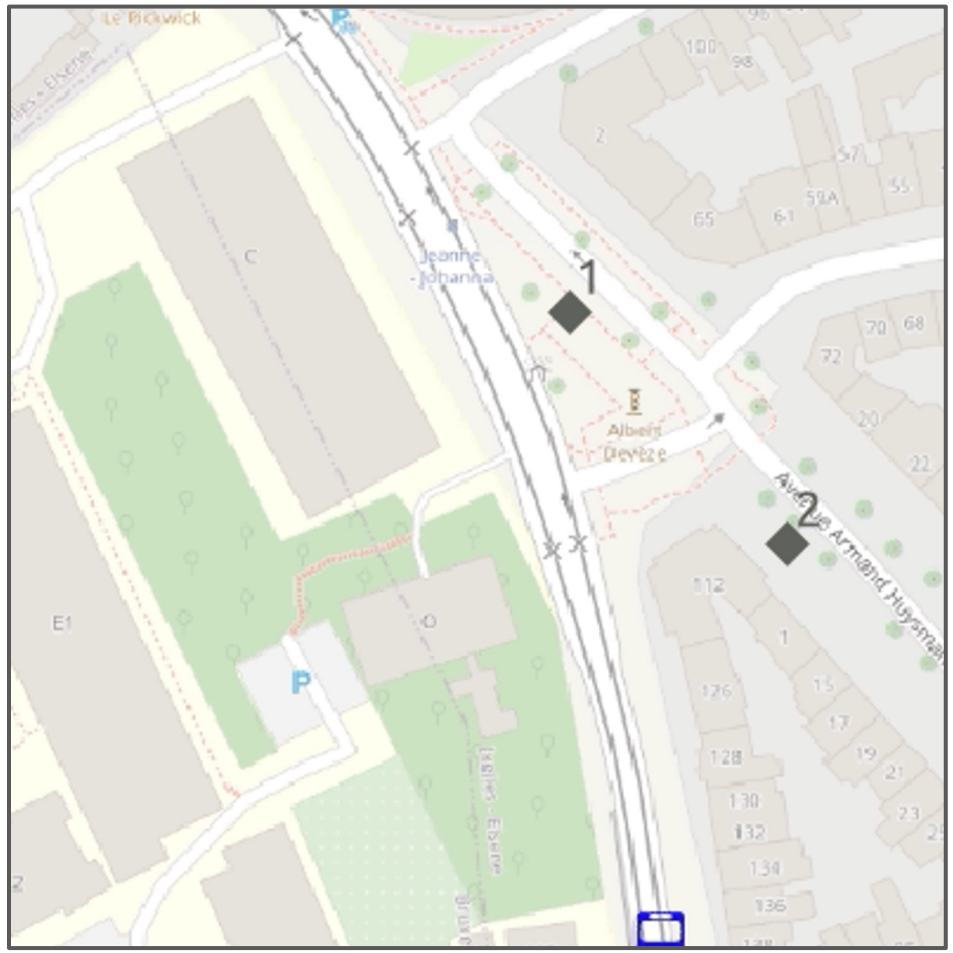


Spatiotemporal Proximity: Marine Data

The question is: Is GPS (static) sampling data enough?

Let’s look at another example

Why continuous trajectories ?



- A geospatial trajectory of a public transport bus that goes nearby an advertising billboard.
- We want to study the visibility of the billboard to the passengers in the bus
- We want to extract interesting insight for advertising agencies to price the billboards, and for advertisers who need to optimize their campaigns
- In this animated gif, we simply assume that if the bus is within 30 meters to the billboard, then it is visible to its passengers (indicated in the animation by the yellow flash around the billboard)

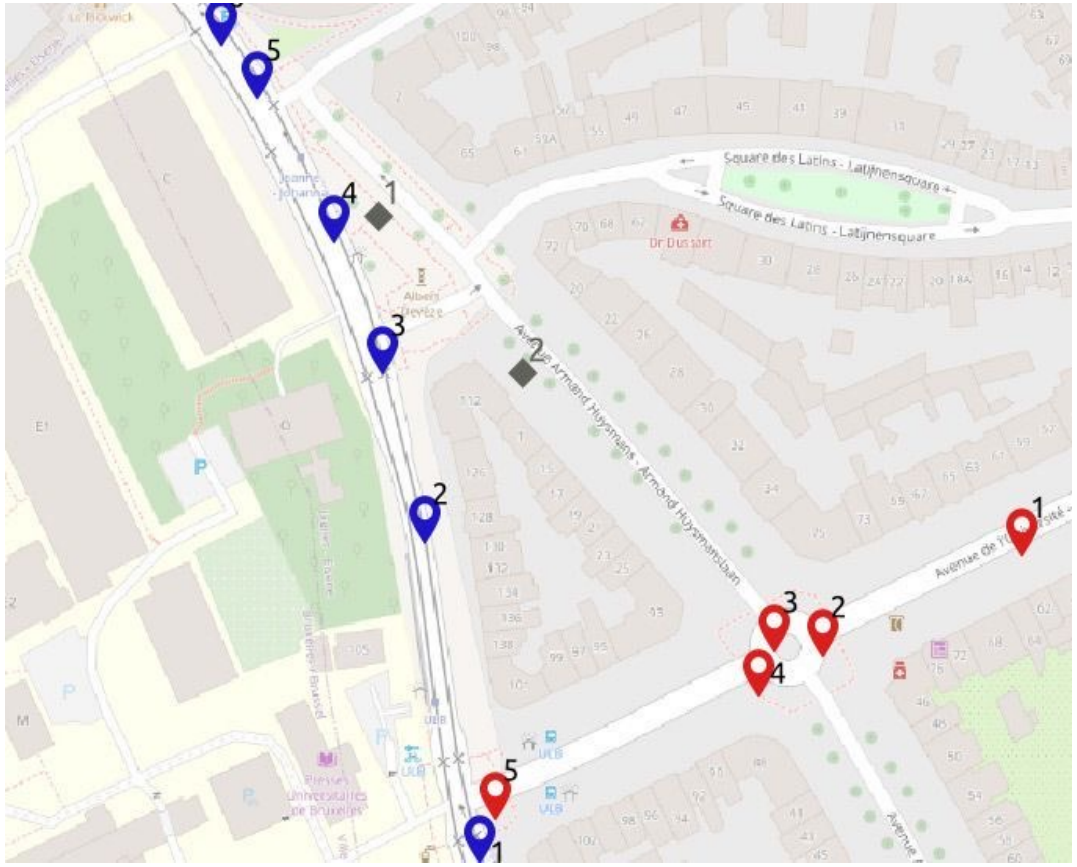
A traditional database solution

```
CREATE TABLE gpsPoint (tripID int, pointID int, t timestamp, geom geometry(Point, 3812));

CREATE TABLE billboard(billboardID int, geom geometry(Point, 3812));

INSERT INTO gpsPoint Values
(1, 1, '2020-04-21 08:37:27', 'SRID=3812;POINT(651096.993815166 667028.114604598)'),
(1, 2, '2020-04-21 08:37:39', 'SRID=3812;POINT(651080.424535144 667123.352304597)'),
(1, 3, '2020-04-21 08:38:06', 'SRID=3812;POINT(651067.607438095 667173.570340437)'),
(1, 4, '2020-04-21 08:38:31', 'SRID=3812;POINT(651052.741845233 667213.026797244)'),
(1, 5, '2020-04-21 08:38:49', 'SRID=3812;POINT(651029.676773636 667255.556944161)'),
(1, 6, '2020-04-21 08:39:08', 'SRID=3812;POINT(651018.401101238 667271.441380755)'),
(2, 1, '2020-04-21 08:39:29', 'SRID=3812;POINT(651262.17004873 667119.331513367)'),
(2, 2, '2020-04-21 08:38:36', 'SRID=3812;POINT(651201.431447782 667089.682115196)'),
(2, 3, '2020-04-21 08:38:43', 'SRID=3812;POINT(651186.853162155 667091.138189286)'),
(2, 4, '2020-04-21 08:38:49', 'SRID=3812;POINT(651181.995412783 667077.531372716)'),
(2, 5, '2020-04-21 08:38:56', 'SRID=3812;POINT(651101.820139904 667041.076539663)');

INSERT INTO billboard Values
 (1, 'SRID=3812;POINT(651066.289442793 667213.589577551)'),
 (2, 'SRID=3812;POINT(651110.505092035 667166.698041233)');
```



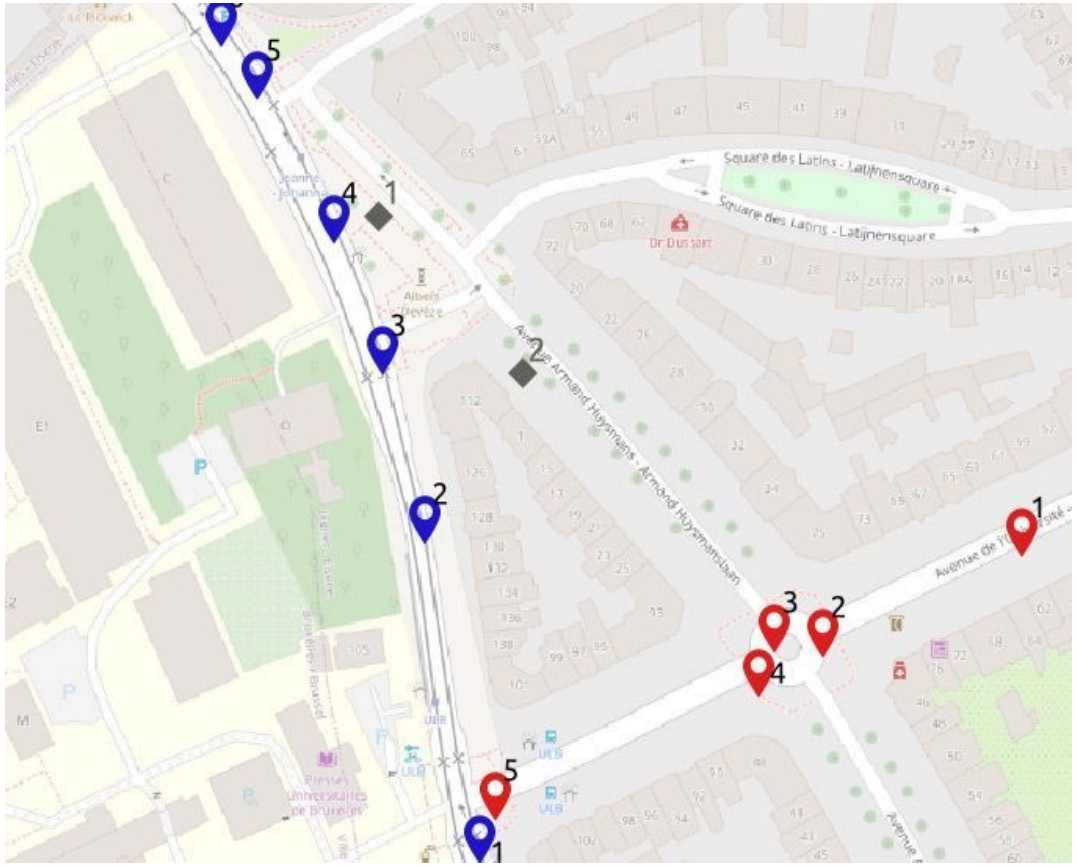
A traditional database solution

Let’s have a look at this query:

```
SELECT tripID, pointID, billboardID
FROM gpsPoint a, billboard b
WHERE st_dwithin(a.geom, b.geom, 30);
```

RESULT: < 1, 4, 1 >

Does the query solve our problem. Why?



A traditional database solution

Let’s have a look at this query:

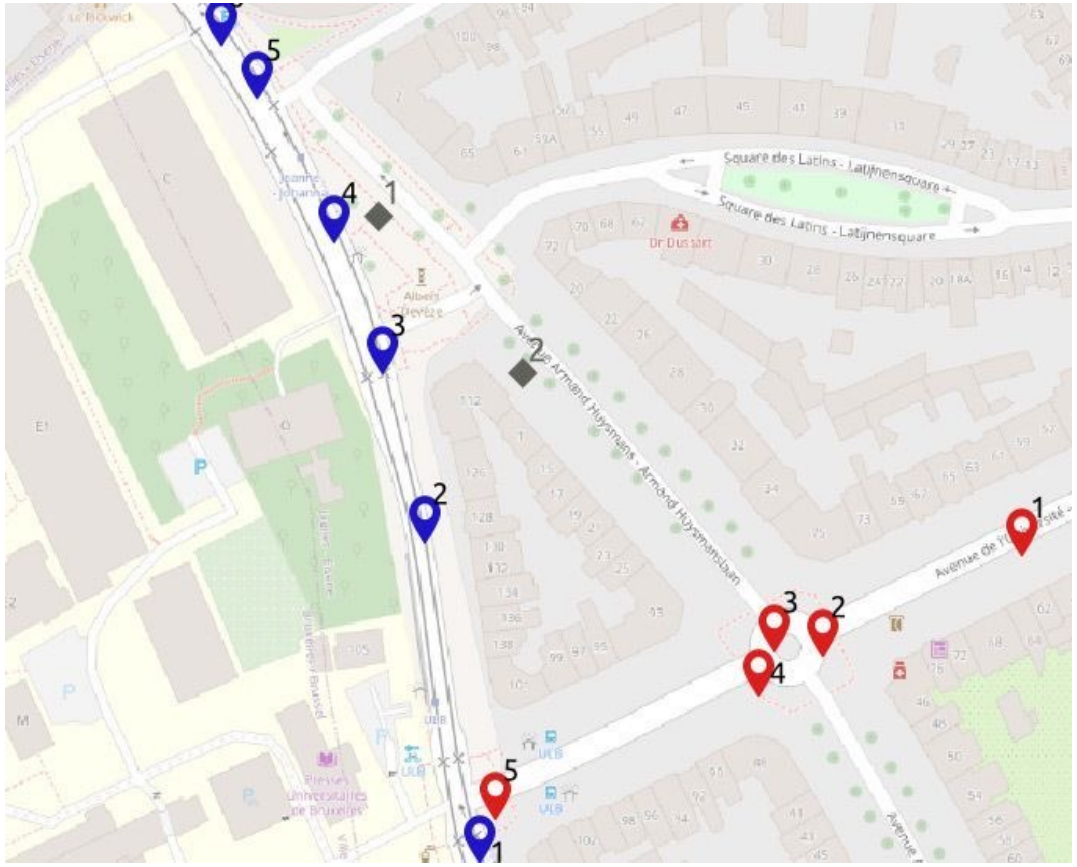
```
SELECT tripID, pointID, billboardID
FROM gpsPoint a, billboard b
WHERE st_dwithin(a.geom, b.geom, 30);
```

RESULT: < 1, 4, 1 >

Does the query solve our problem. Why?

The condition in the WHERE clause finds the GPS points that are within 30 meters from a billboard. But the query does not tell the duration of this event.

If point 4 in trip 1 (the blue trip) is not given, the query returns null => the query does not deal with the continuity of the bus trip

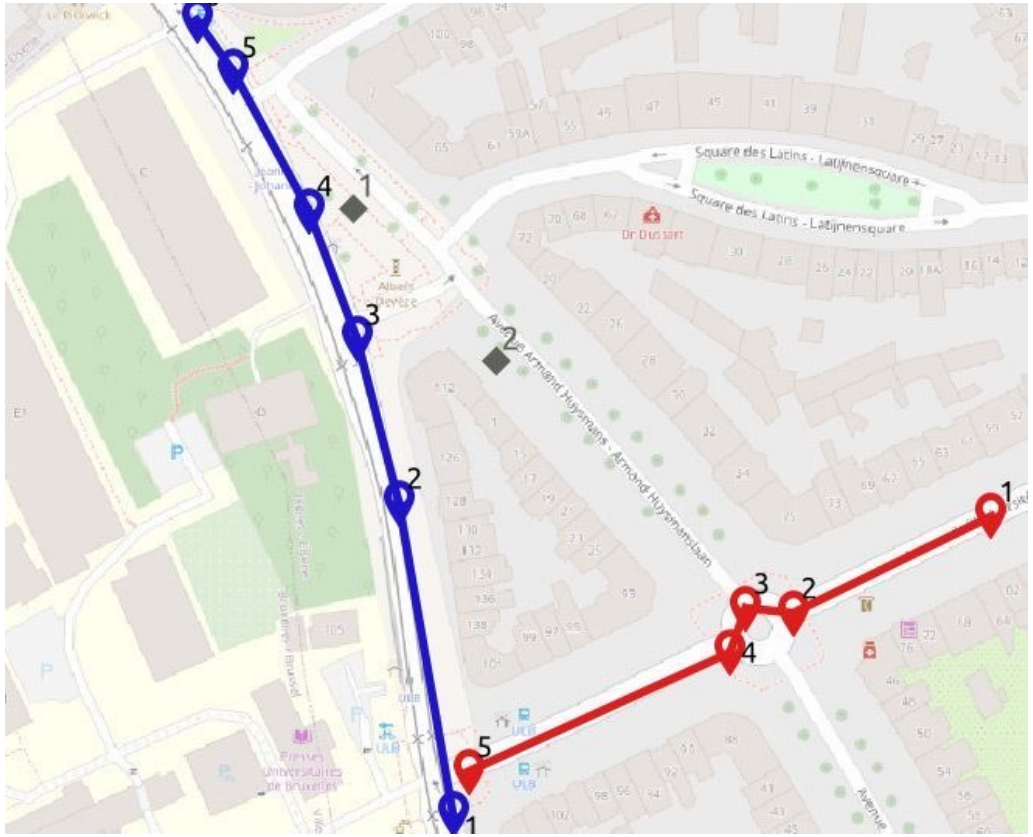


A traditional database solution

We need to reconstruct the continuous movement trajectory from the given GPS points

Query that computes *the locations of the billboard's visibility to the passengers, and how long the billboard was visible to them:*

```
WITH pointPair AS (
    SELECT tripID, pointID AS p1, t AS t1, geom AS geom1, lead(pointID, 1) OVER
    (PARTITION BY tripID ORDER BY pointID) p2, lead(t, 1) OVER (PARTITION BY tripID ORDER
     BY pointID) t2, lead(geom, 1) OVER (PARTITION BY tripID ORDER BY pointID) geom2
     FROM gpsPoint
), segment AS (
    SELECT tripID, p1, p2, t1, t2,  st_makeline(geom1, geom2) geom
     FROM pointPair
     WHERE p2 IS NOT NULL
-- So far, the two  continuous lines are returned.
), approach AS(
    SELECT tripID, p1, p2, t1, t2, a.geom,  st_intersection(a.geom, st_exteriorRing
    (st_buffer(b.geom, 30))) AS visibilityTogglePoint
    FROM segment a, billboard b  WHERE st_dwithin(a.geom, b.geom, 30)
)
SELECT tripID, p1, p2, t1, t2, geom, visibilityTogglePoint,  (st_lineLocatePoint(geom, visibilityTogglePoint) * (t2 - t1))
+ t1 visibilityToggleTime
FROM approach;
```



This is the result of the CTEs

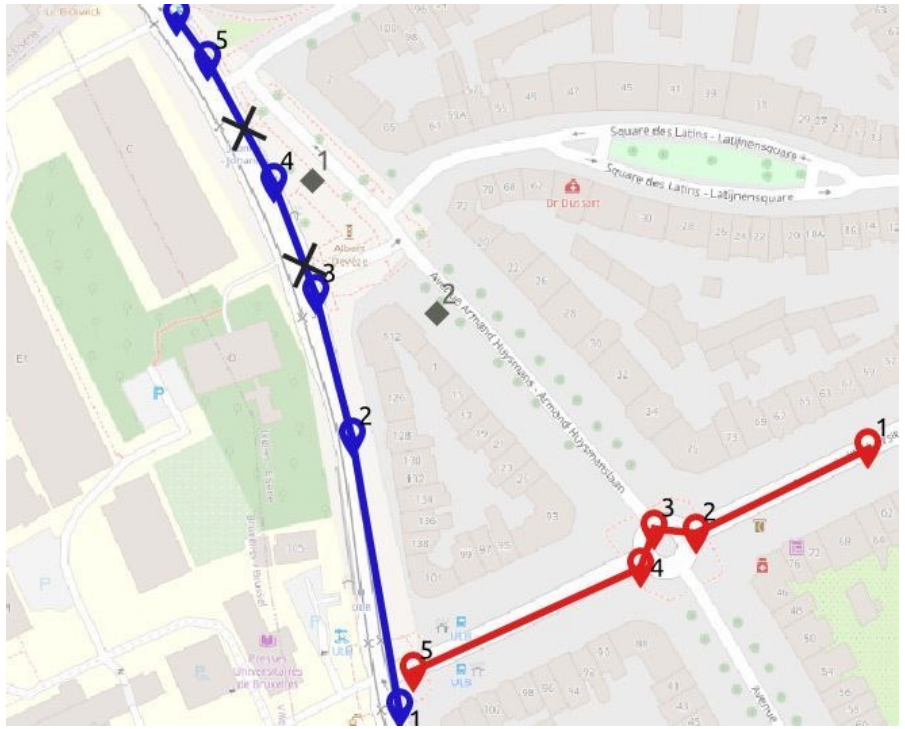
- The **first** CTE, **pointPair** uses the window function, ***lead***, to pack every pair of consecutive points that belong to the same bus trip, into one tuple.
- The **second** CTE, **segment,** connects the two points with a line segment (i.e., a linear interpolation of the path between every two GPS points).

A traditional database solution

We need to reconstruct the continuous movement trajectory from the given GPS points

Query that computes *the locations of the billboard's visibility to the passengers, and how long the billboard was visible to them:*

```
WITH pointPair AS (
    SELECT tripID, pointID AS p1, t AS t1, geom AS geom1, lead(pointID, 1) OVER
    (PARTITION BY tripID ORDER BY pointID) p2, lead(t, 1) OVER (PARTITION BY tripID ORDER
     BY pointID) t2, lead(geom, 1) OVER (PARTITION BY tripID ORDER BY pointID) geom2
     FROM gpsPoint
), segment AS (
    SELECT tripID, p1, p2, t1, t2,  st_makeline(geom1, geom2) geom
     FROM pointPair
     WHERE p2 IS NOT NULL
-- So far, the two  continuous lines are returned.
), approach AS(
    SELECT tripID, p1, p2, t1, t2, a.geom,  st_intersection(a.geom, st_exteriorRing
    (st_buffer(b.geom, 30))) AS visibilityTogglePoint
    FROM segment a, billboard b  WHERE st_dwithin(a.geom, b.geom, 30)
)
SELECT tripID, p1, p2, t1, t2, geom, visibilityTogglePoint,  (st_lineLocatePoint(geom, visibilityTogglePoint) * (t2 - t1))
+ t1 visibilityToggleTime
FROM approach;
```



This is the result of the third CTE

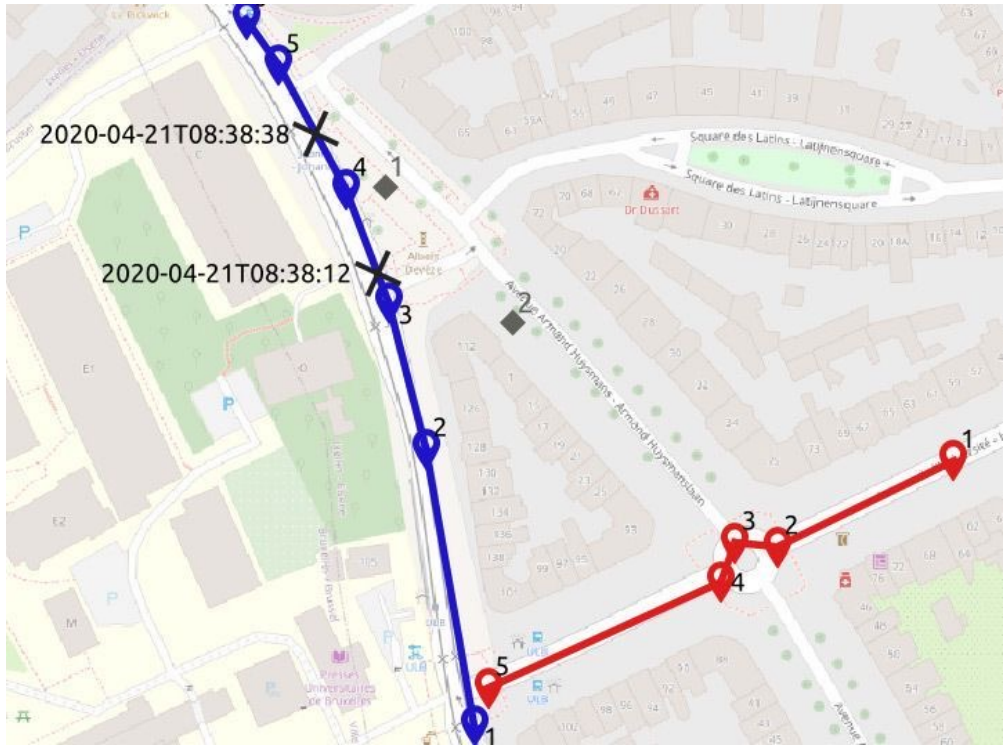
The **third** CTE, **approach**, finds the locations where the bus starts/ends to be within 30 meters from the billboard. It draws a circular ring with 30 meters diameter around the billboard and intersects it with the segments of the bus trajectory. We get the two points marked with the black cross.

A traditional database solution

We need to reconstruct the continuous movement trajectory out of the given GPS points

Query that computes *the locations of the billboard's visibility to the passengers, and how long the billboard was visible to them:*

```
WITH pointPair AS (
    SELECT tripID, pointID AS p1, t AS t1, geom AS geom1, lead(pointID, 1) OVER
    (PARTITION BY tripID ORDER BY pointID) p2, lead(t, 1) OVER (PARTITION BY tripID ORDER
     BY pointID) t2, lead(geom, 1) OVER (PARTITION BY tripID ORDER BY pointID) geom2
     FROM gpsPoint
), segment AS (
    SELECT tripID, p1, p2, t1, t2,  st_makeline(geom1, geom2) geom
     FROM pointPair
     WHERE p2 IS NOT NULL
-- So far, the two  continuous lines are returned.
), approach AS(
    SELECT tripID, p1, p2, t1, t2, a.geom,  st_intersection(a.geom, st_exteriorRing
    (st_buffer(b.geom, 30))) visibilityTogglePoint
    FROM segment a, billboard b  WHERE st_dwithin(a.geom, b.geom, 30)
)
SELECT tripID, p1, p2, t1, t2, geom, visibilityTogglePoint,  (st_lineLocatePoint(geom, visibilityTogglePoint) * (t2 -
t1)) + t1 visibilityToggleTime
FROM approach;
```



This is the result of the query

The **last step** computes the time at these two points using linear referencing, i.e., assuming a constant speed per segment

A moving object approach: MobilityDB

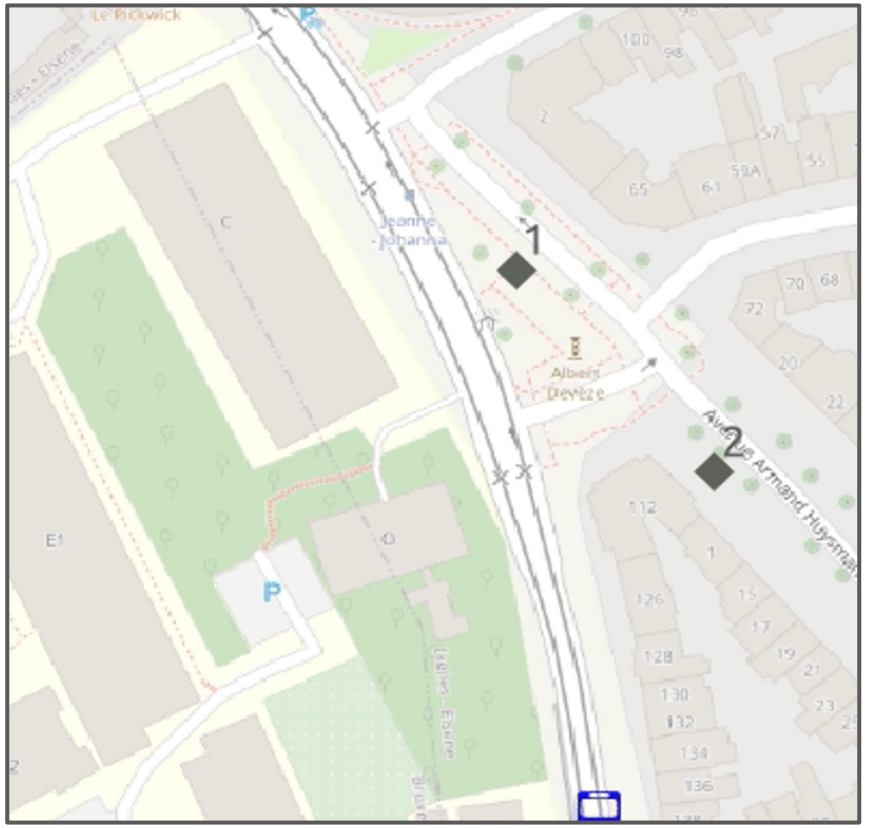
The PostGIS query is not trivial, because it must deal with:

Continuous movement trajectory: GPS data is discrete => we must reconstruct the continuous movement trajectory.

Spatiotemporal proximity: the continuous movement trajectory was used to find the location and time when the bus was within 30 meters from the billboard.

A MOD approach makes it easier to analyze these trajectories

MobilityDB (mobilitydb.com) is an extension of PostgreSQL and PostGIS that implements these spatiotemporal concepts as custom types and functions in Postgres.



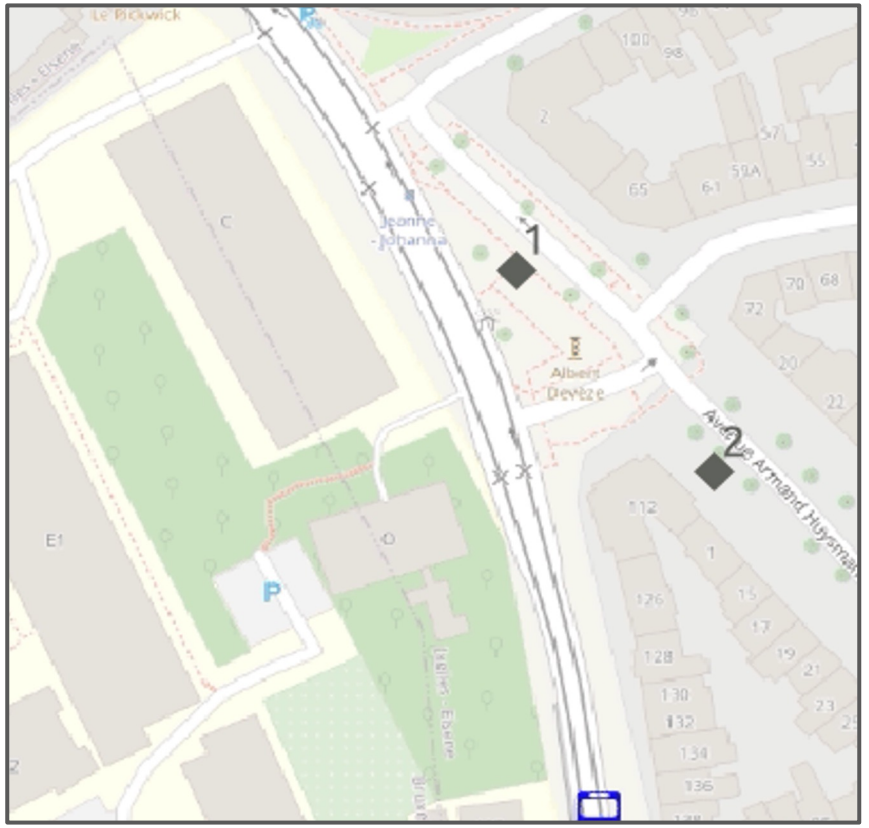
A moving object approach: MobilityDB

In MobilityDB the query above is expressed as:

```
SELECT astext(atperiodset(trip, getTime(atValue(tdwithin(a.trip, b.geom, 30), TRUE))))
FROM busTrip a, billboard b
WHERE dwithin(a.trip, b.geom, 30)
```

The result is:

{[POINT(651063.737915354 667183.840879818)@2020-04-21 08:38:12.507515+02,
POINT(651052.741845233 667213.026797244)@2020-04-21 08:38:31+02,
POINT(651042.581085347 667231.762425657)@2020-04-21 08:38:38.929465+02]}



A moving object approach: MobilityDB

In MobilityDB the query above is expressed as:

```
SELECT astext(atperiodset(trip, getTime(atValue(tdwithin(a.trip, b.geom, 30), TRUE))))
FROM   busTrip a, billboard b
WHERE dwithin(a.trip, b.geom, 30)
```

- Table busTrip has the attribute trip of type tgeompoint (the MobilityDB type for storing a moving point)
- The nesting tdwithin->atValue->getTime returns the time periods during which a bus trip has been within a distance of 30 meters from a billboard.
- The function atperiodset returns the bus trip to these time periods.
- The PostGIS function astext converts the output to textual format.

- The result is the part of the bus trip that starts at 2020-04-21 08:38:12.507515+02 and ends at 08:38:38.929465+02.