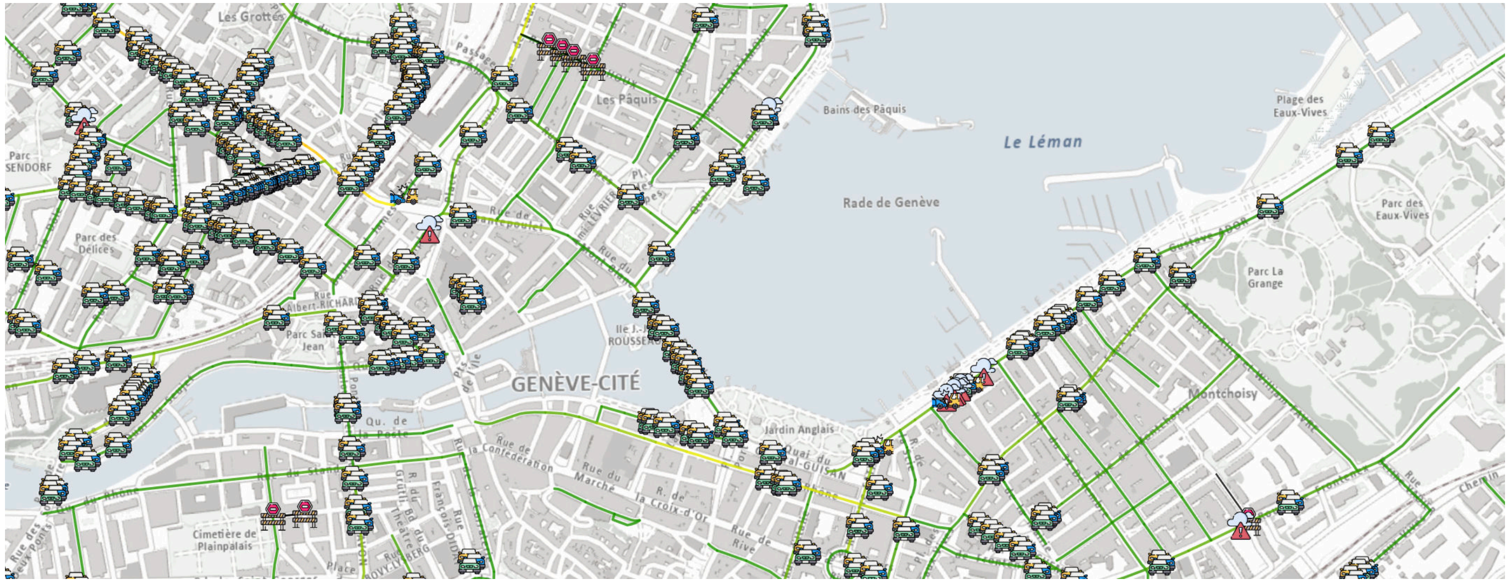
Making sense of Waze traffic data with Python, TimescaleDB-PostGIS and JavaScript

Alessandro Cerioni

Data Scientist @ Direction de l'Information du Territoire, État de Genève

June 27, 2025



Motivation

- Beneficiary: Geneva's Cantonal Office of Transport

- Goals of the project:

  i. get a better **knowledge** on road traffic

  - already exploited data sources: inductive loop counters, TomTom

  ii. assist the **decision-making** process

  - traffic regulation
  - interplay with the public transport
  - land use planning

What's Waze?

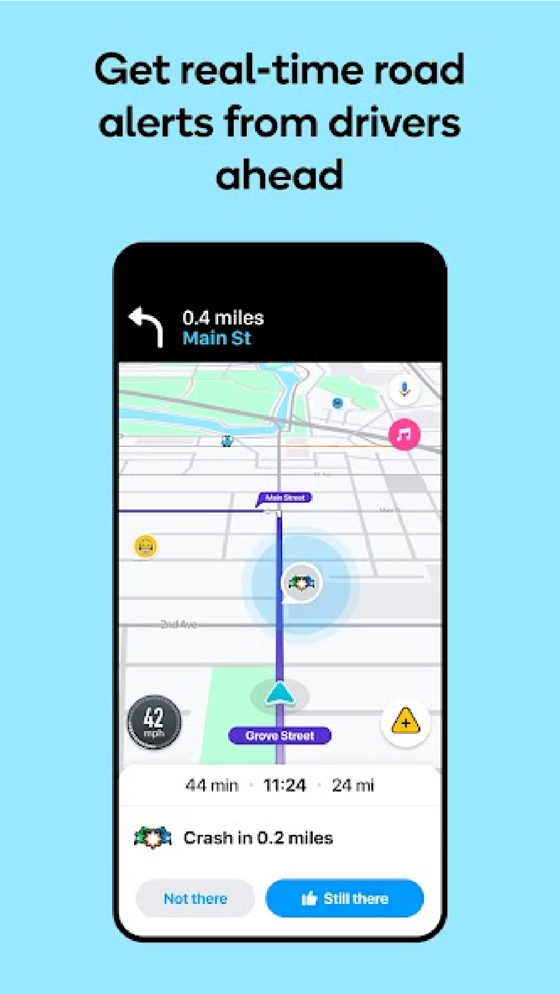


Image credit: Google Play

- free GPS navigation and live traffic app for Android, iOS
- users (aka "wazers") can report accidents, traffic jams, weather hazards, ...
- Waze collects travel times and traffic information from users
- acquired by Google in June 2013

Waze data feed

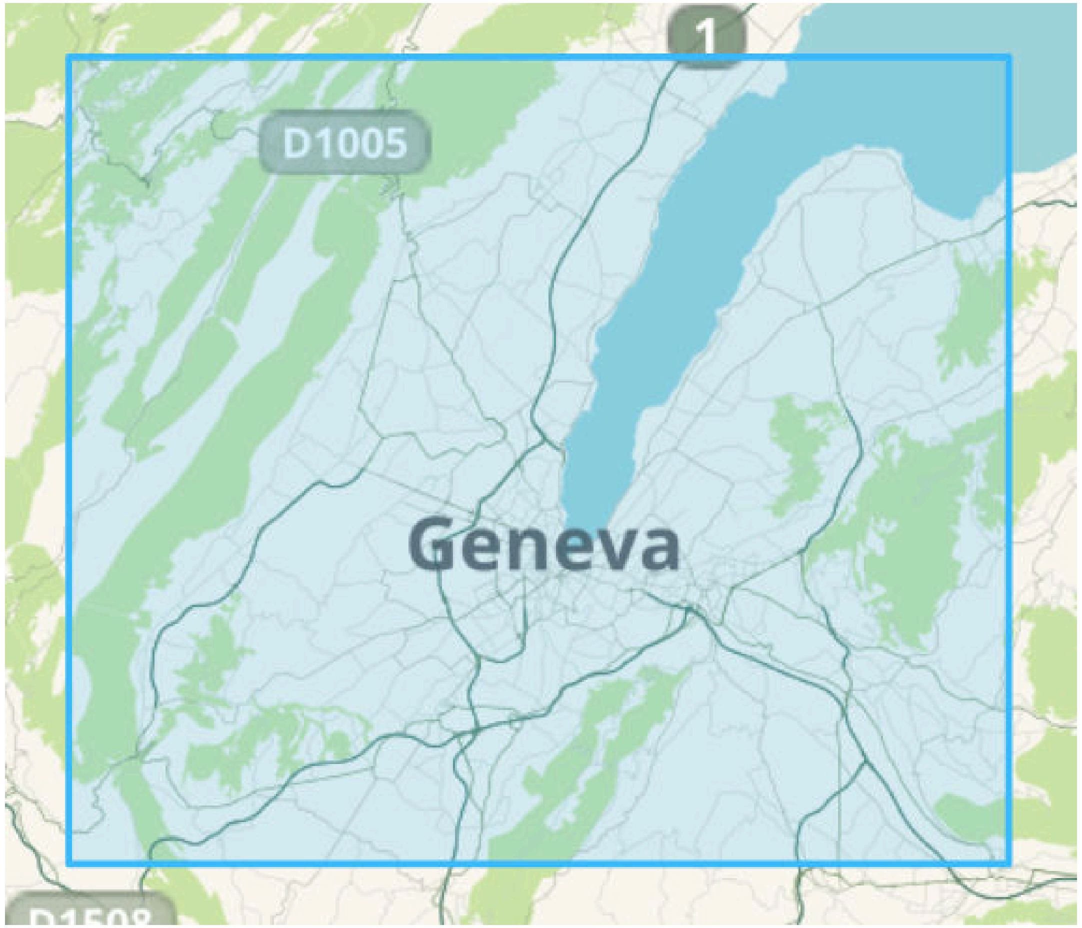
- **"Waze for Cities" program** - see https://www.waze.com/fr/wazeforcities
  "*available to authorities that manage traffic or public infrastructure*"

Dataset	Geometry
`alerts`	points
`jams`	lines
`irregularities`	lines

- access through an HTTP API
- refresh interval = 1 minute
- specs ➡️ https://tinyurl.com/35f28zj9

- terms - see https://sites.google.com/site/wazeccpattributionguidelines/membership-criteria
  - ✅ internal use, real-time incident notifications
  - ⛔ "*distribute or publish aggregated or historic Waze data or any analyses of the Waze data*"
    ("*except with Google's express prior written consent*")

Waze data collection



- data is being collected since March 2023
- Waze API --- **Python program** ---> S3
- the area of interest covers ~1 550 km^2 ("Greater Geneva")
- >8 GB JSON data / month
- >15 M records / month
- ⚠️ Waze API doesn't provide historical data

1st use case: congestion rate computation

- $\text{congestion rate} = \dfrac{\text{congested time}}{\text{observation time } (e.g.\ 1 \text{ day, } 1 \text{ month, ...})}$
- congestion if and only if

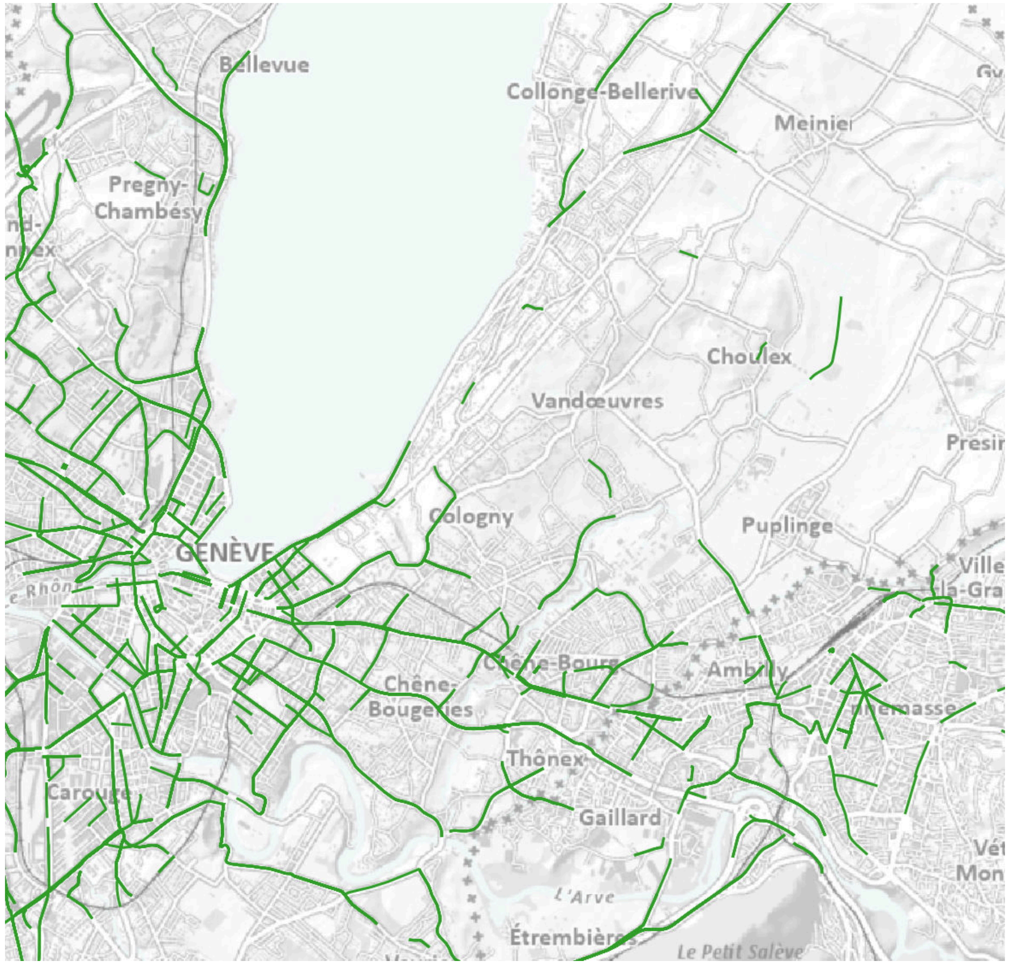
$$\text{travel time} > k \times (\text{free-flow travel time}), \ \ k \geq 1 \quad [1]$$

- $[1]$ can be reformulated as follows:

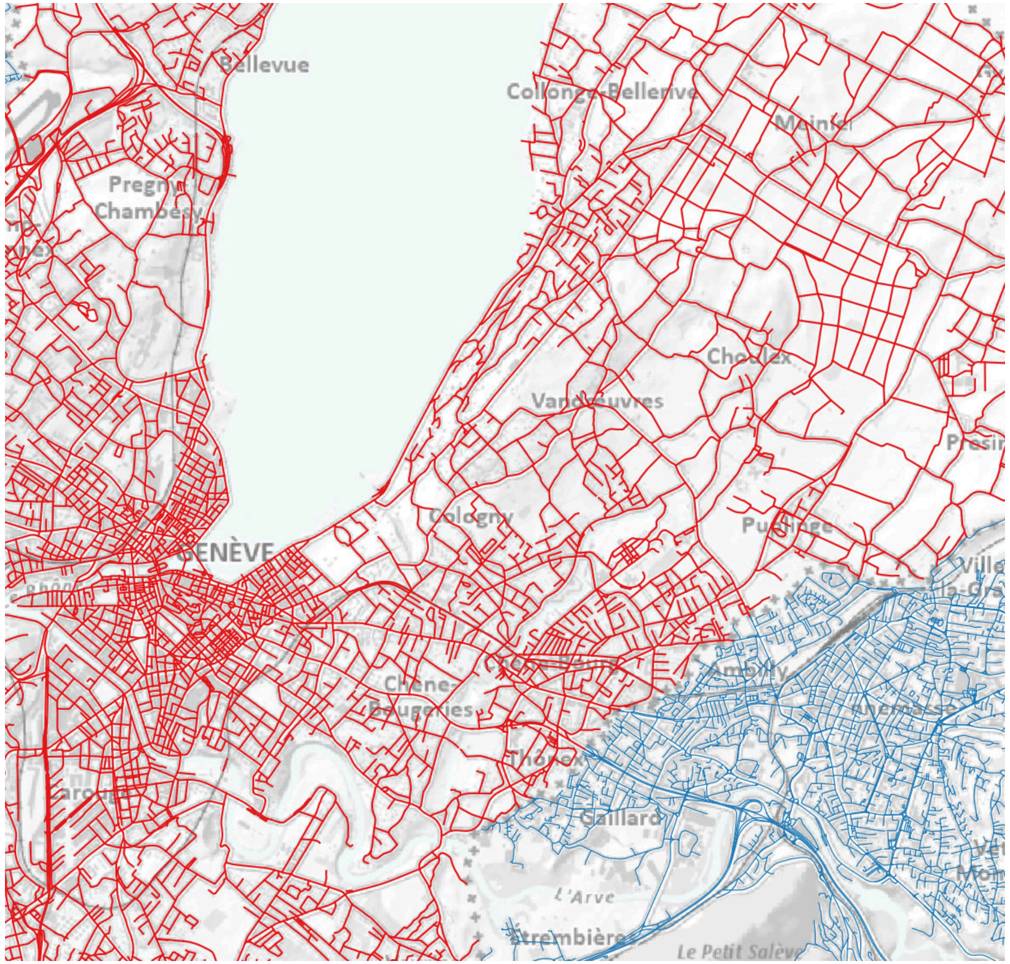
$$\frac{\text{delay}}{\text{length}/\text{speed}} > \frac{k-1}{k} \equiv \text{“minimum congestion coefficient”} \quad [2]$$

- the `jams` and `irregularities` datasets provide the speed, delay and length for each record (line geometry, Waze-specific)
- ⚠️ beneficiary requirement: **compute the congestion rate for every edge of the Greater Geneva's road network**

Waze jams

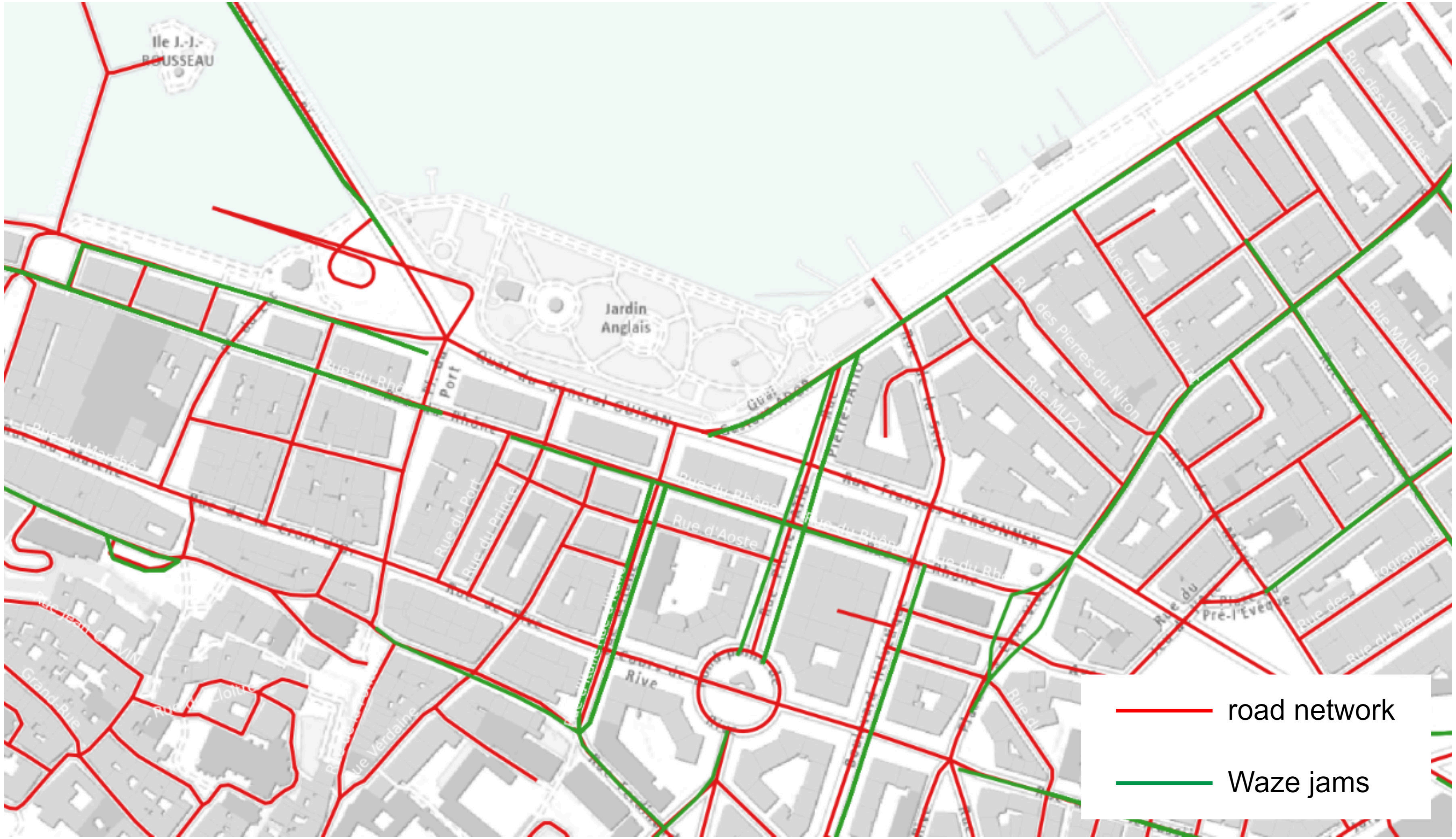


Greater Geneva's road network



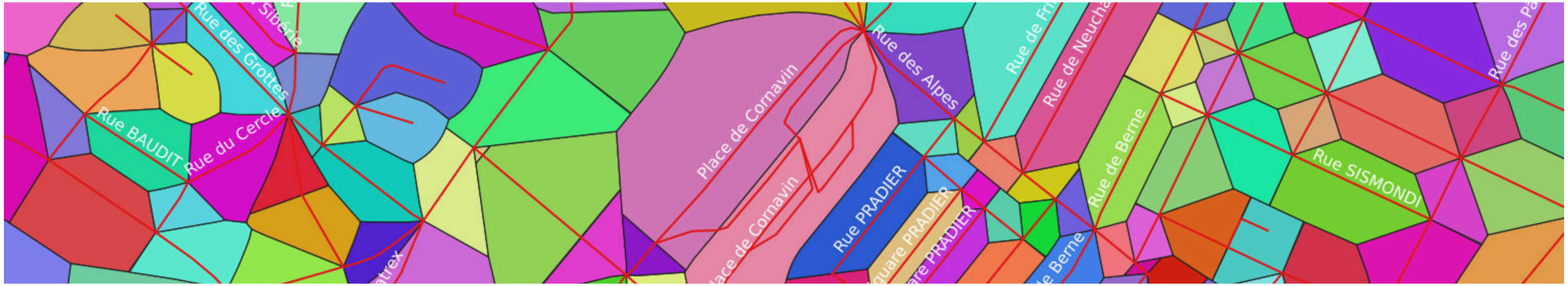
How to relate these two sets of geometries to each other?

How to relate Waze geometries to the road network?



How to relate Waze geometries to the road network?

1. a mosaic is generated out of the road network, using Voronoi cells / Thiessen polygons

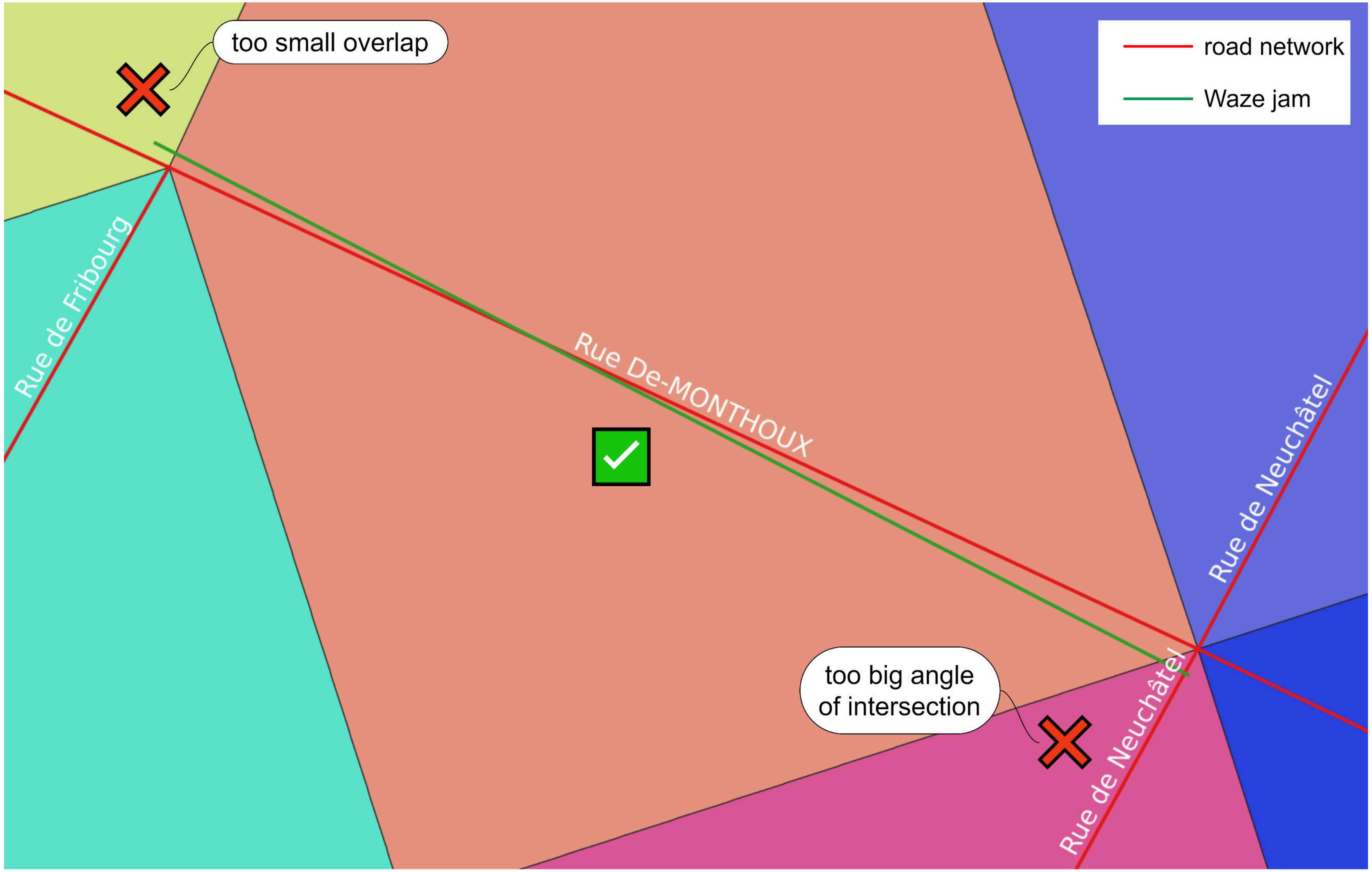


2. spatial ∩ between Waze geometries and the mosaicked road network => **candidate relations**

   ⚠️ some edges of the road network happen to be stacked along the z-axis (tunnels, ...)

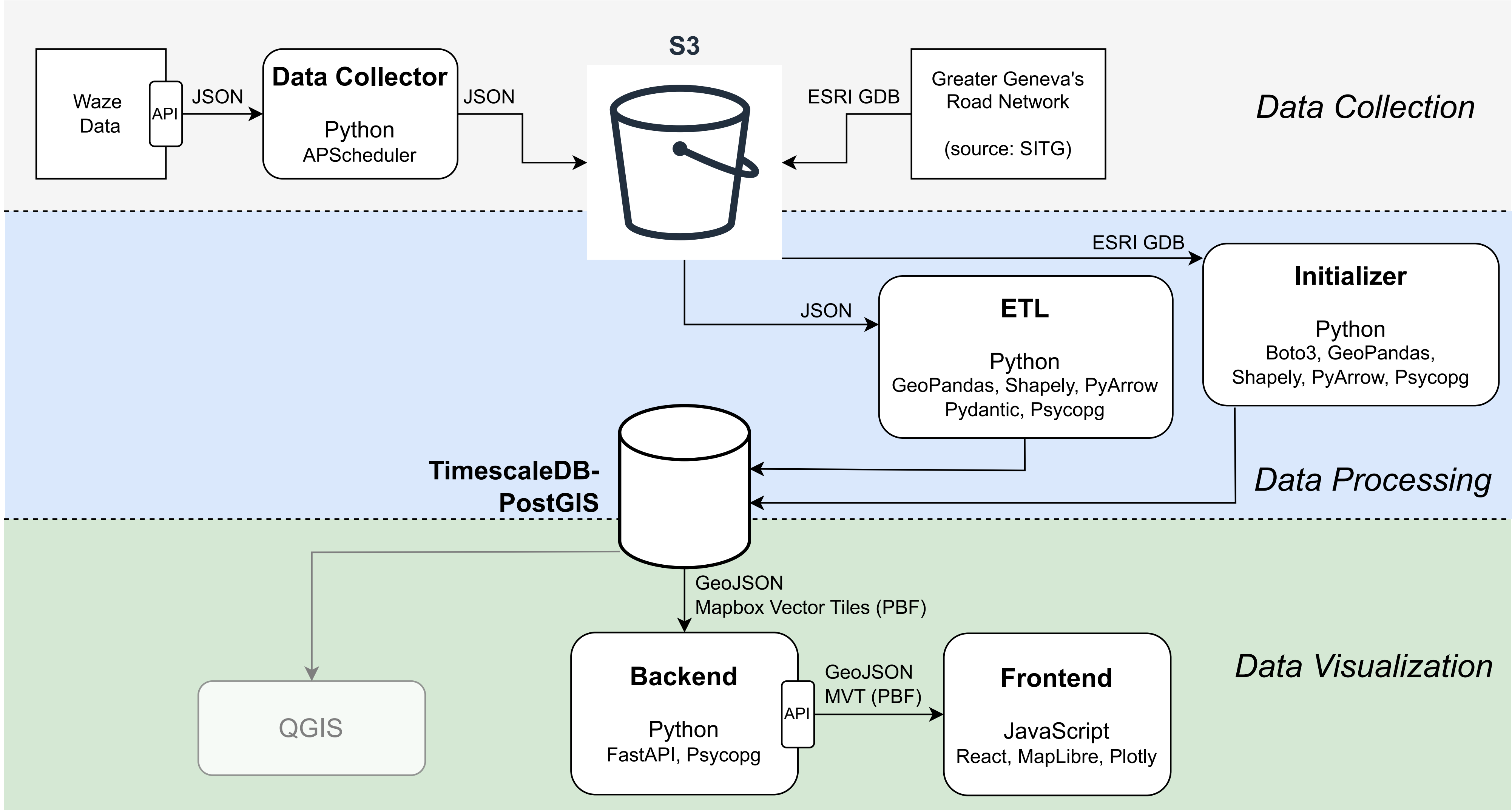
3. some of the **candidate relations** are filtered out depending on
   - the angle of intersection (maximum threshold *i.e.* `max_angle`)
   - the overlap (minimum threshold *i.e.* `min_overlap`)

How to relate Waze geometries to the road network?



Demo...

Architecture v1



Mapbox Vector Tiles generation

```
WITH
bbox AS (
    SELECT ST_TileEnvelope(%(z)s, %(x)s, %(y)s) AS geom
),
[...]
road_network AS (
    SELECT [...], ST_AsMVTGeom(A.geom, bbox.geom, 4096, 256, true) AS geom
    FROM rn_with_counts A, bbox, date_bins B
),
[...]
tiles AS (
    SELECT ST_AsMVT(road_network, 'road_network', 4096, 'geom', 'id') AS mvt from road_network
    UNION
    SELECT ST_AsMVT(closed_roads, 'closed_roads', 4096, 'geom', 'id') AS mvt FROM closed_roads
)
SELECT string_agg(mvt, '') from tiles;
```

Filters

```
WITH
bbox AS (
    SELECT ST_TileEnvelope(%(z)s, %(x)s, %(y)s) AS geom
),
[...]
filtered_join_table AS (
    SELECT jam_geometry_md5, road_segment_geometry_md5
    FROM waze.jams__road_network
    WHERE road_segment_geometry_md5 IN (SELECT geometry_md5 FROM road_network_within_bbox) AND
      angle_deg <= %(max_angle_degrees)s AND overlap >= %(min_overlap)s
),
time_filtered_jams AS (
    SELECT geometry_md5, time, _fid, delay_seconds, _congestion_coefficient
    FROM waze.jams_events
    WHERE time >= %(from)s::timestamptz AND time < %(to)s::timestamptz
),
[...]
spacetime_filtered_jams AS (
    SELECT geometry_md5, _fid, delay_seconds, time_filtered_jams_with_geometry.geom AS geom
    FROM time_filtered_jams_with_geometry, bbox
    WHERE time_filtered_jams_with_geometry.geom && bbox.geom
),
[...]
```

Aggregations

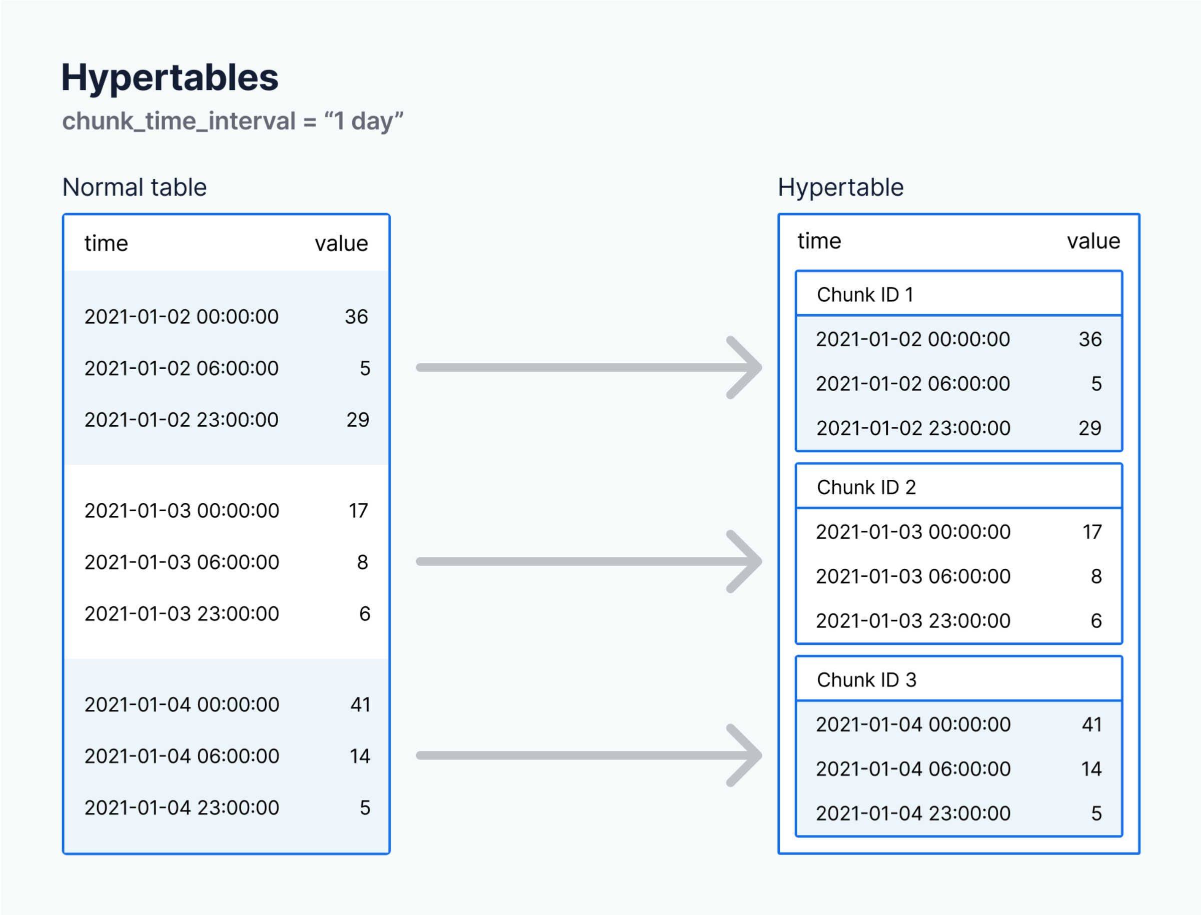
```
[...]
date_bins AS (
    SELECT COUNT(*) AS no_date_bins
    FROM generate_series(%(from)s::timestamptz, %(to)s::timestamptz, %(bin_width_minutes)s)
),
jams_joined_with_road_network AS (
    SELECT road_segment_geometry_md5,
           date_bin(%(bin_width_minutes)s, time, TIMESTAMP WITH TIME ZONE '1970-01-01T00:00:00Z') AS _date_bin
    FROM spacetime_filtered_jams_without_road_closed A JOIN filtered_join_table B ON A.geometry_md5 = B.jam_geometry_md5
),
jams_joined_with_road_network_relevant_rows AS (
    SELECT DISTINCT ON (_date_bin, road_segment_geometry_md5) road_segment_geometry_md5
    FROM jams_joined_with_road_network
),
counts AS (
    SELECT road_segment_geometry_md5, count(*) AS cnt
    FROM jams_joined_with_road_network_relevant_rows
    GROUP BY road_segment_geometry_md5
),
road_network_with_counts AS (
    SELECT A._fid, A.geometry_md5, A.id_gm_troncon, B.cnt, A.geom
    FROM road_network_within_bbox A JOIN counts B ON A.geometry_md5 = B.road_segment_geometry_md5
),
[...]
```

Indexes

Several fields are indexed:

- angle of intersection, overlap between Waze events and GE's road network
- congestion coefficient
- geometry (thanks to PostGIS 👏)
- last but not least: **time**
  - note that Waze events are time partitioned by TimescaleDB

TimescaleDB: hypertables and chunks



Hypertables

chunk_time_interval = “1 day”

Normal table

time	value
2021-01-02 00:00:00	36
2021-01-02 06:00:00	5
2021-01-02 23:00:00	29
2021-01-03 00:00:00	17
2021-01-03 06:00:00	8
2021-01-03 23:00:00	6
2021-01-04 00:00:00	41
2021-01-04 06:00:00	14
2021-01-04 23:00:00	5

Hypertable

time	value
Chunk ID 1	
2021-01-02 00:00:00	36
2021-01-02 06:00:00	5
2021-01-02 23:00:00	29
Chunk ID 2	
2021-01-03 00:00:00	17
2021-01-03 06:00:00	8
2021-01-03 23:00:00	6
Chunk ID 3	
2021-01-04 00:00:00	41
2021-01-04 06:00:00	14
2021-01-04 23:00:00	5

The actual interest of TimescaleDB in this application is yet to be confirmed…

Image taken from https://docs.tigerdata.com/use-timescale/latest/hypertables/

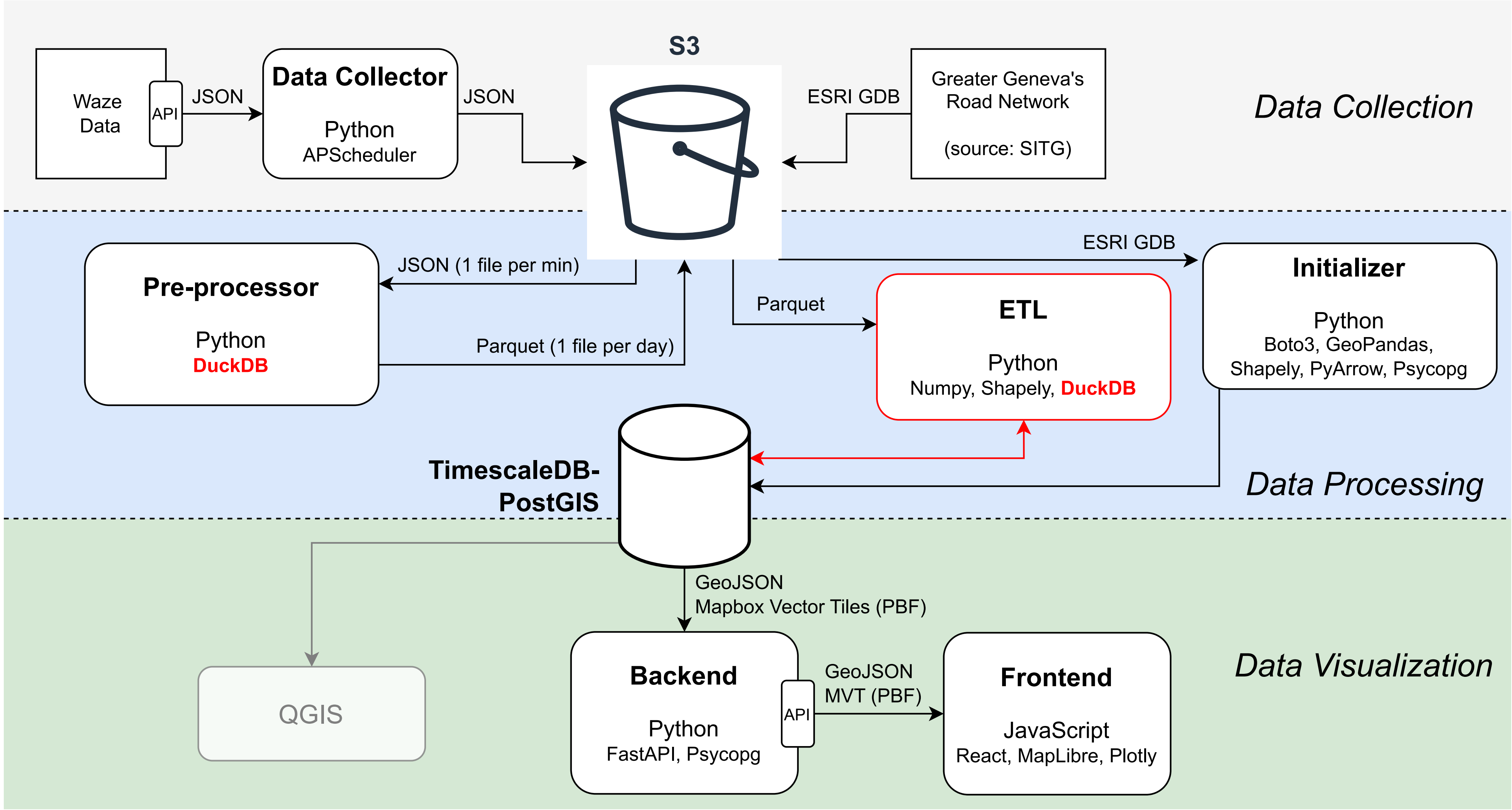
Architecture v1 - issues

- it takes more than 1 second to ETL one source JSON file (4k bogomips CPU)
  ➡️ **more than 12h to ETL one month's worth of data!**
- what if:
  - the end-user requires new features?
  - we want to use another (version of the) road network?
  - ...
- at first, it seemed relevant to push ALL data attributes to PG,
  in order to be ready to perform all kind of analyses
  - what if Waze's data model evolves? (it already happened!)
- can a migration tool like postgres_migrator be helpful?

DuckDB saved my day!

- an open-source (MIT) column-oriented RDBMS which supports the SQL
- first released in 2019, latest stable release on June 16, 2025 (v1.3.1)
- already very popular: 30.5k stars on GitHub
- in-process => high-speed data transfer to and from the DB
- scalable, fast, designed to support analytical query workloads (OLAP)
- no external dependencies => extremely portable
- extensible:
  - core extensions, in particular:
    - PostgreSQL Extension
    - Spatial Extension
  - community extensions

Architecture v2



Architecture v2

- ~45 min to ETL a month's worth of data, **more than 10x faster than v1**
  - ~8 GB JSON data --> ~100 MB Parquet data (zstd compression)
- clearer separation of concerns:
  - DuckDB is used for data processing and analytics
    - developer-friendly access to source data
  - PG is used to perform on-the-fly aggregations / filtering / vector tiles generation / ...
    - the front-end app dictates the data model to the back-end,
      which in turn dictates the data model to the DB
- scaling-up with DuckDB seems to be easier than with PG: **(where) am I wrong**?
- functions can be implemented in Python
  (which I am much more familiar with than with PL/pgSQL 🫣)

`get_slope` user-defined function - PL/pgSQL

```sql
CREATE FUNCTION waze.get_slope(
    geometry public.geometry
) RETURNS double precision
LANGUAGE plpgsql IMMUTABLE STRICT PARALLEL SAFE
AS $$
    DECLARE
        slope double precision;
    BEGIN

        SELECT regr_slope(y, x) INTO slope FROM (
            SELECT ST_X(a.geom) as x, ST_Y(a.geom) as y FROM (
                SELECT (ST_DumpPoints(ST_Segmentize(geometry, 0.5))).geom
            ) a
        ) b;

        RETURN slope;
    END;
$$;
```

`get_slope` user-defined function - Python

```python
import numpy as np
import shapely

def get_slope(geom: bytes) -> float:

    _geom = shapely.from_wkb(geom)
    _geom = shapely.segmentize(_geom, 0.5)

    coords = shapely.get_coordinates(_geom).tolist()
    xy = [item for sublist in coords for item in sublist]

    x = xy[::2]   # Elements at even indices
    y = xy[1::2]  # Elements at odd indices

    slope, _ = np.polyfit(x, y, 1)

    return slope
```

Lessons learned

- data collection: S3-based Data Lake 👍
- PG or DuckDB?

  Answer: PG AND DuckDB!
- no "one size fits all": analytics != application development
- PG ~~can be~~ IS scary!

  becoming at ease with PG is not that obvious, at least for me 🫢

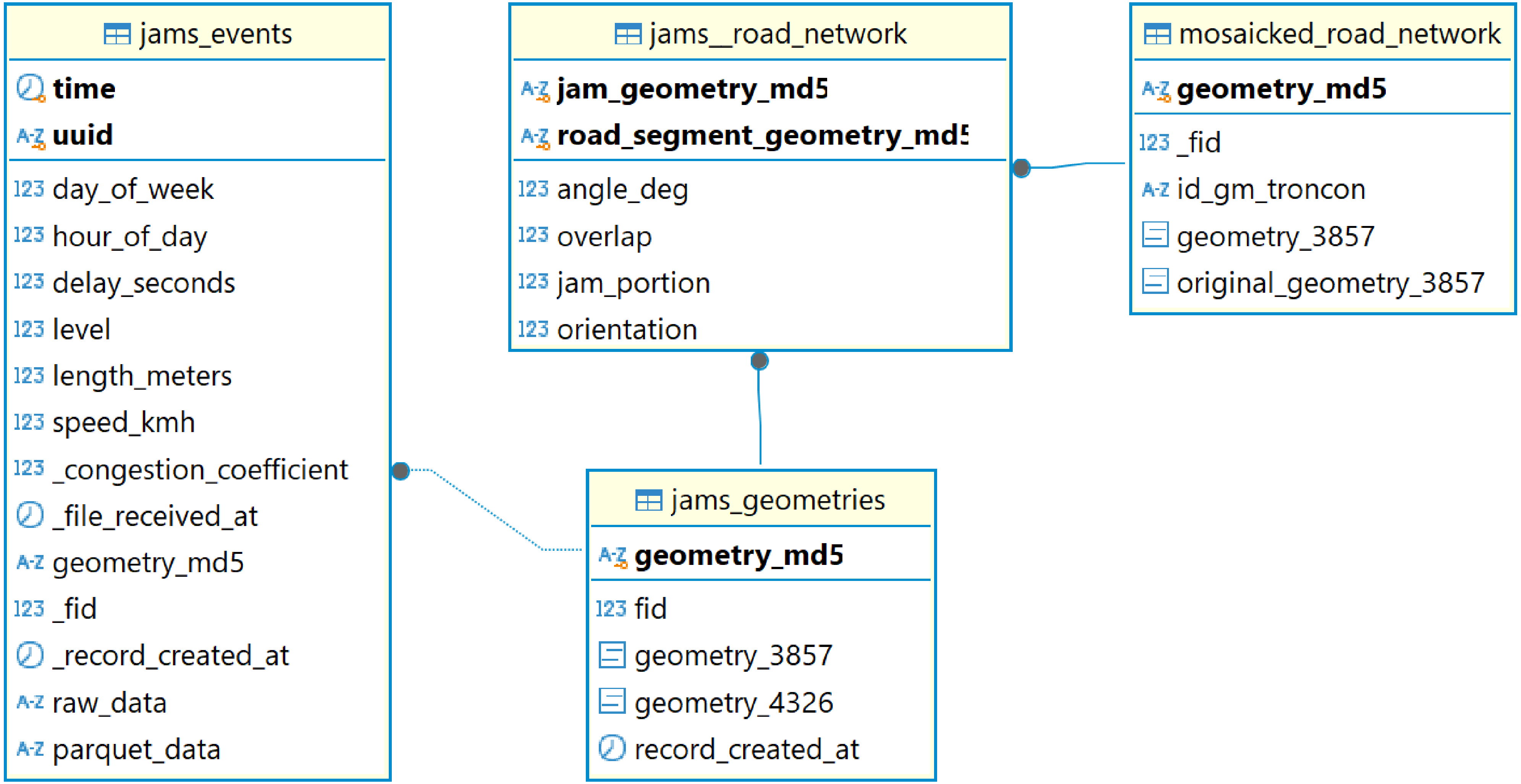
Outlook on future developments

1. reach the "**minimal viable product**" milestone
    - compute the congestion rate on specific days of week and time ranges
    - periodic (monthly, yearly, ...) report generation
    - hot spot analysis, pattern recognition, ...
2. **collaborate with other parties** (federal offices, cantons, municipalities, ...)
    - release the code under Open Source terms
    - use the OpenStreeMap road network
3. do some more **data analysis**
    - cross-analysis with weather data, public transport data
    - impact assessment (*ex ante*, *ex post*)

Thanks for your attention, questions and feedback are welcome!

alessandro.cerioni@etat.ge.ch

Annex: Relational Data Model



jams_events

time
uuid
day_of_week
hour_of_day
delay_seconds
level
length_meters
speed_kmh
_congestion_coefficient
_file_received_at
geometry_md5
_fid
_record_created_at
raw_data
parquet_data

jams__road_network

jam_geometry_md5
road_segment_geometry_md5
angle_deg
overlap
jam_portion
orientation

mosaicked_road_network

geometry_md5
_fid
id_gm_troncon
geometry_3857
original_geometry_3857

jams_geometries

geometry_md5
fid
geometry_3857
geometry_4326
record_created_at

Annex: Web Mapping / Tiling

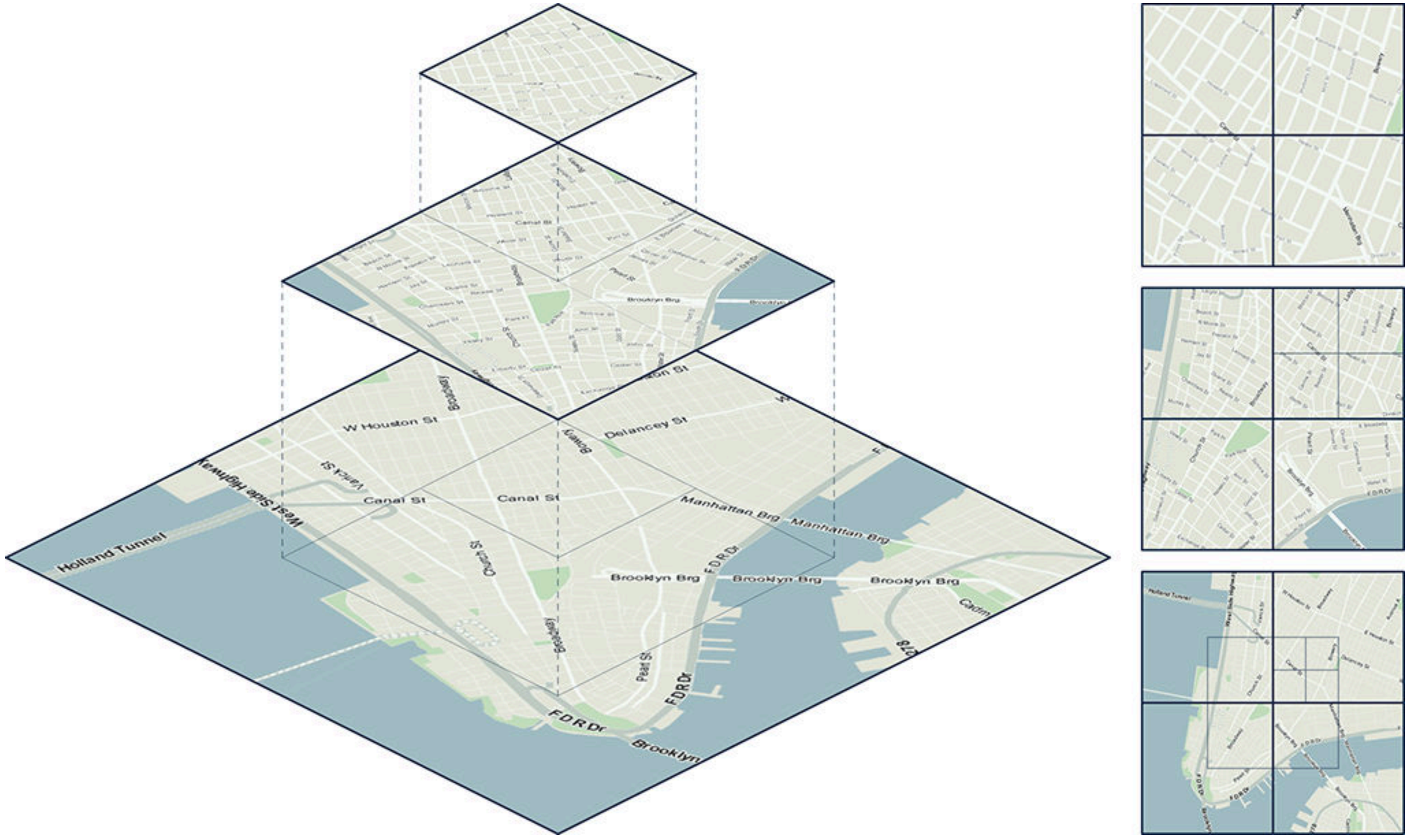


Image credit: https://avantgeo.com/generar-cartografia-offline-para-aplicaciones-moviles/