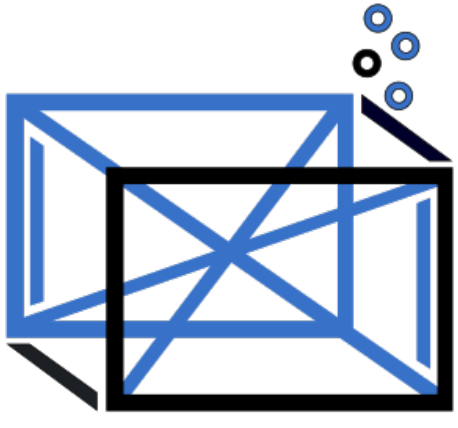
PostGIS and PostgreSQL: GIS Data, Queries, and Performance

SRIDs, `EXPLAIN (ANALYZE)`, and Hexes



RustProof Labs

bringing you data

SRID

Spatial Reference IDentifier

... it's complicated

SRID Functions

- `ST_SetSRID(geom, srid)`
- `ST_SRID(geom)`
- `ST_Transform(geom, srid)`

https://blog.rustprooflabs.com/static/docs/RustProofLabs-PostGIS-Function-Guide.pdf

Generic SRIDs

- 3857: Projected coordinate system (based on WGS84)
- 4326: Geographic coordinate system (WGS84)

Generic SRID

Units

- 3857: Meters
- 4326: Decimal Degrees

My Default: 3857

- Default of `osm2pgsql`
- Units in meters

Know your units

- PostGIS calculations use the SRID's units
- 1 decimal degree (4326) = 111 km (3857)
- Miles vs. Kilometers is difficult enough
- Decimal Degrees don't mentally jive for me

Be Consistent with SRIDs

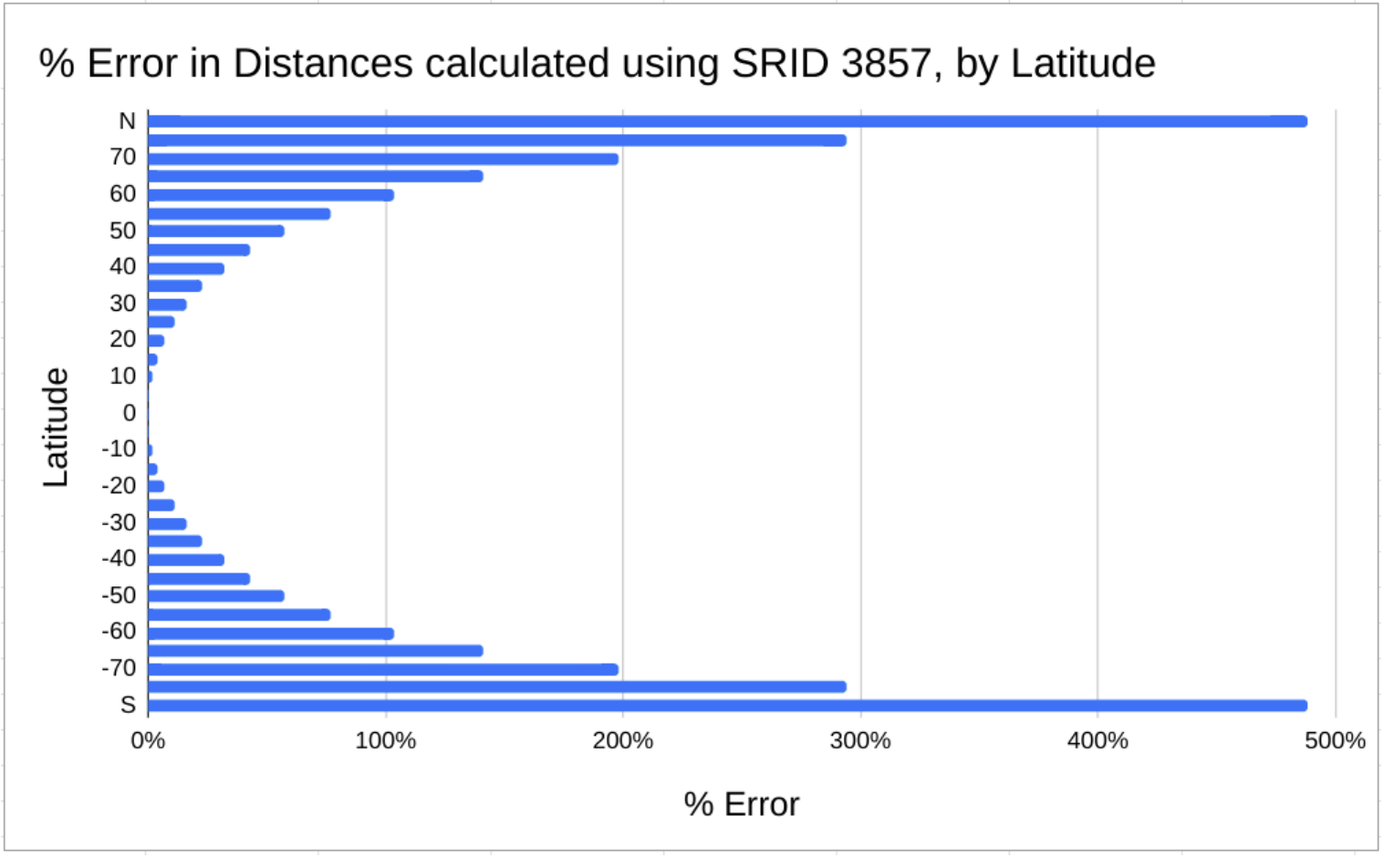
- Pick an SRID for your standard
- Be clear when you deviate from your standard

Downside to generic SRIDs

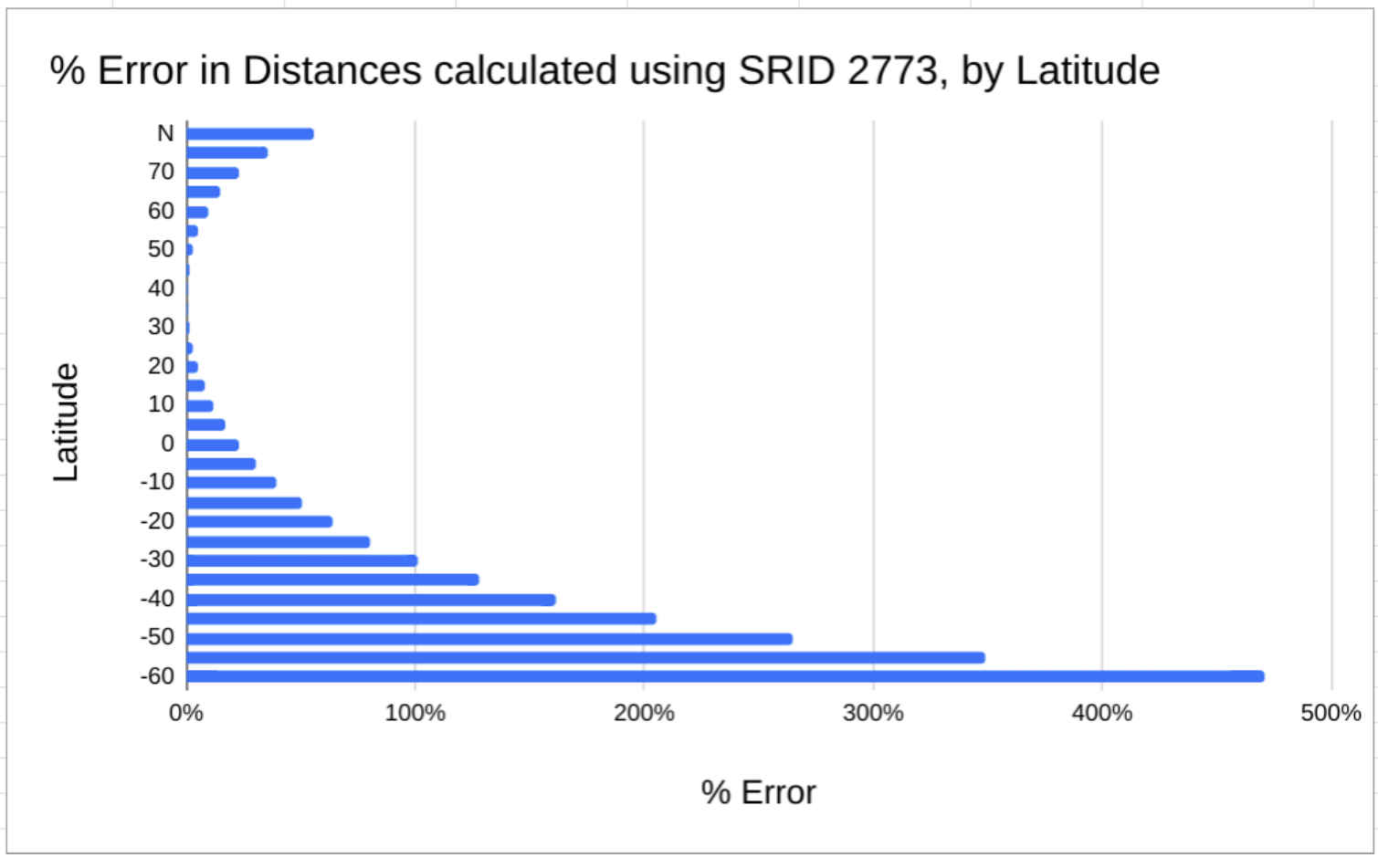
e.g. 3857 and 4326

- Inaccurate calculations in most of the world
- `ST_Area()`, `ST_Distance()`, etc

Generic SRID Error Rates

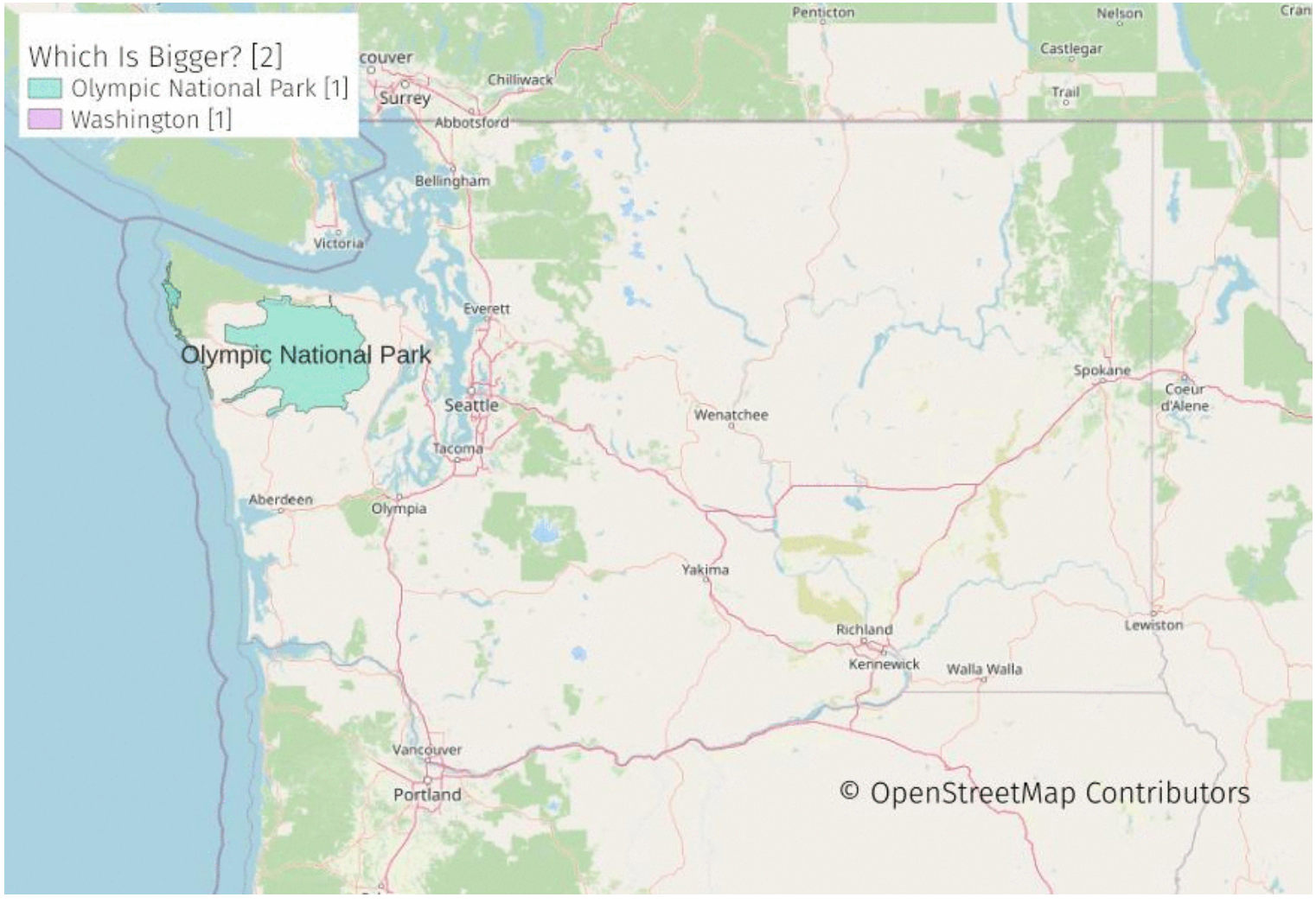


https://blog.rustprooflabs.com/2023/04/postgis-geometry-accuracy



https://blog.rustprooflabs.com/2023/04/postgis-geometry-accuracy

Inaccurate calculations okay when comparing in region



Finding Local SRIDs

- SRID Bounding Box project
- Provides data and view `public.srid_units`

https://github.com/rustprooflabs/srid-bbox

Read more: https://blog.rustprooflabs.com/2020/11/postgis-find-local-srid

DEMO - `03 - a.sql`

SRID Questions?

Postgres `EXPLAIN`

```
EXPLAIN (ANALYZE, BUFFERS, COSTS, WAL,
         SETTINGS, VERBOSE, FORMAT JSON)
<your query here>
;
```

https://www.postgresql.org/docs/current/sql-explain.html

Help your stats

```
ANALYZE;
```

```
ANALYZE != EXPLAIN (ANALYZE)
```

Tools for `EXPLAIN (ANALYZE)`

https://explain.depesz.com/

https://www.pgmustard.com/

Configurables RE using `EXPLAIN`

```
track_io_timing = on
compute_query_id = on
```

Configurables RE performance

- `work_mem = 10MB` is typically safe starting point
- `shared_buffers = 1GB` (start @ 25% of total)
- `max_parallel_workers_per_gather = 2` (keep under 50% of total)
- `random_page_cost = 1.1`
- `jit = off`

Less obvious (but often important)

- `max_wal_size = 10GB` Keep headroom on disk, should always have more than this available
- `maintenance_work_mem = 1GB` (assuming at least 8GB instance)

Configurables for Performance

Just In Time

```
jit = off
```

Configurables for Performance

Work Memory

- Don't bother with formulas
- Establish reasonable baseline
- Log / monitor temp files

```
work_mem = 10MB
log_temp_files = 0
```

Tune `work_mem` per login/group role

```
ALTER ROLE ryanlambert SET work_mem = '1GB';
ALTER ROLE webapp SET work_mem = '4MB';
ALTER ROLE analytics_group SET work_mem = '50MB';
```

In [4]:

```
sql_raw = """
SHOW work_mem;
"""
pd.read_sql(sql_raw, get_db_conn(in_docker=False))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[4], line 4
      1 sql_raw = """
      2 SHOW work_mem;
      3 """
----> 4 pd.read_sql(sql_raw, get_db_conn(in_docker=False))

NameError: name 'pd' is not defined
```

Logging config

```
log_checkpoints = on
log_connections = on
log_disconnections = on
log_duration = on
log_hostname = on
log_statement = 'all'
log_line_prefix = '%t [%p]: [%l-1]
user=%u,db=%d,app=%a,client=%h,query_id=%Q  '
```

Approach to performance tuning

Tempting: Why isn't the planner choosing to use `<some specific algo/sort/hash/something>`?

- Postgres does not allow forcing plans.
- Avoid the temptation of `enable_seqscan = off`!

Approach to performance tuning

> *The alogrithm / plan is a symptom*

Approach to performance tuning

> *The alogrithm / plan is a symptom*

The Real Causes for What Happens

- Statistics
- Your actual data and query
- Configuration

Approach to performance tuning

- Solve the root cause
- It read a lot of data from disk? How can we reduce that?
- This approach is more involved, but produces better results

Approach to performance tuning

How can we think like the planner?

Approach to performance tuning

How can we think like the planner?

Spoiler: Check the statistics!

DEMO - `03 - b.sql`

Row estimates with Spatial Joins

- Tricky
- No statistic for "how many of these things are in this place"

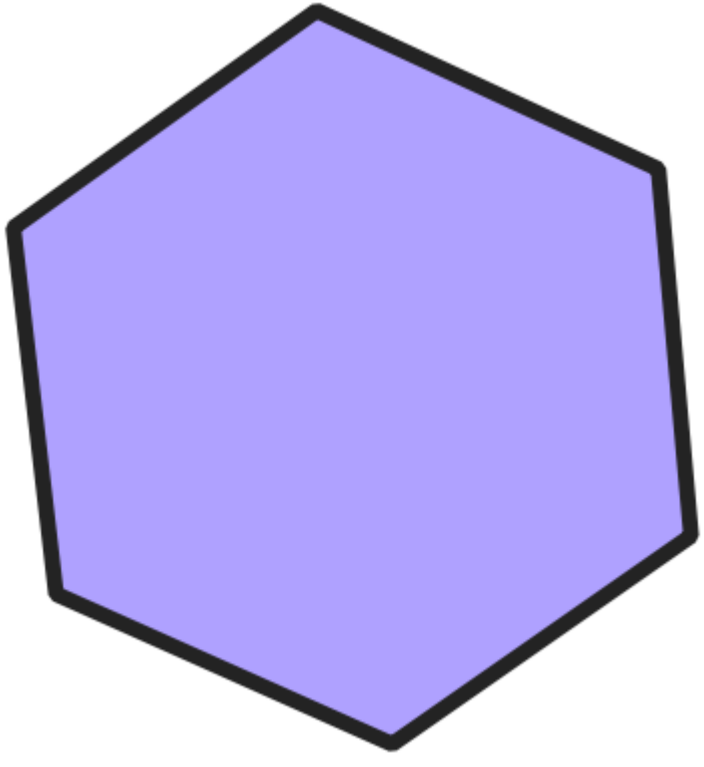
Analyze / Track Logs

- Log temp files
- pgBadger
- `auto_analyze`

Questions about `EXPLAIN`?

Not these hexes

These hexes!



Why use Hexes?

- ✅ Analysis
- ✅ Visualization
- ✅ Privacy
- 🤔 Performance

Hexes for Analysis

Hexagons have many advantages

- Grid is roughly circular
- Nest from small to large scale, enabling proper scaling
- Equal area (ish)

- https://medium.com/swlh/spatial-data-analysis-with-hexagonal-grids-961de90a220e
- https://www.mdpi.com/2220-9964/10/9/576
- https://ica-abs.copernicus.org/articles/3/140/2021/ica-abs-3-140-2021.pdf

Hexes for Analysis

vs. Geopolitical Boundaries

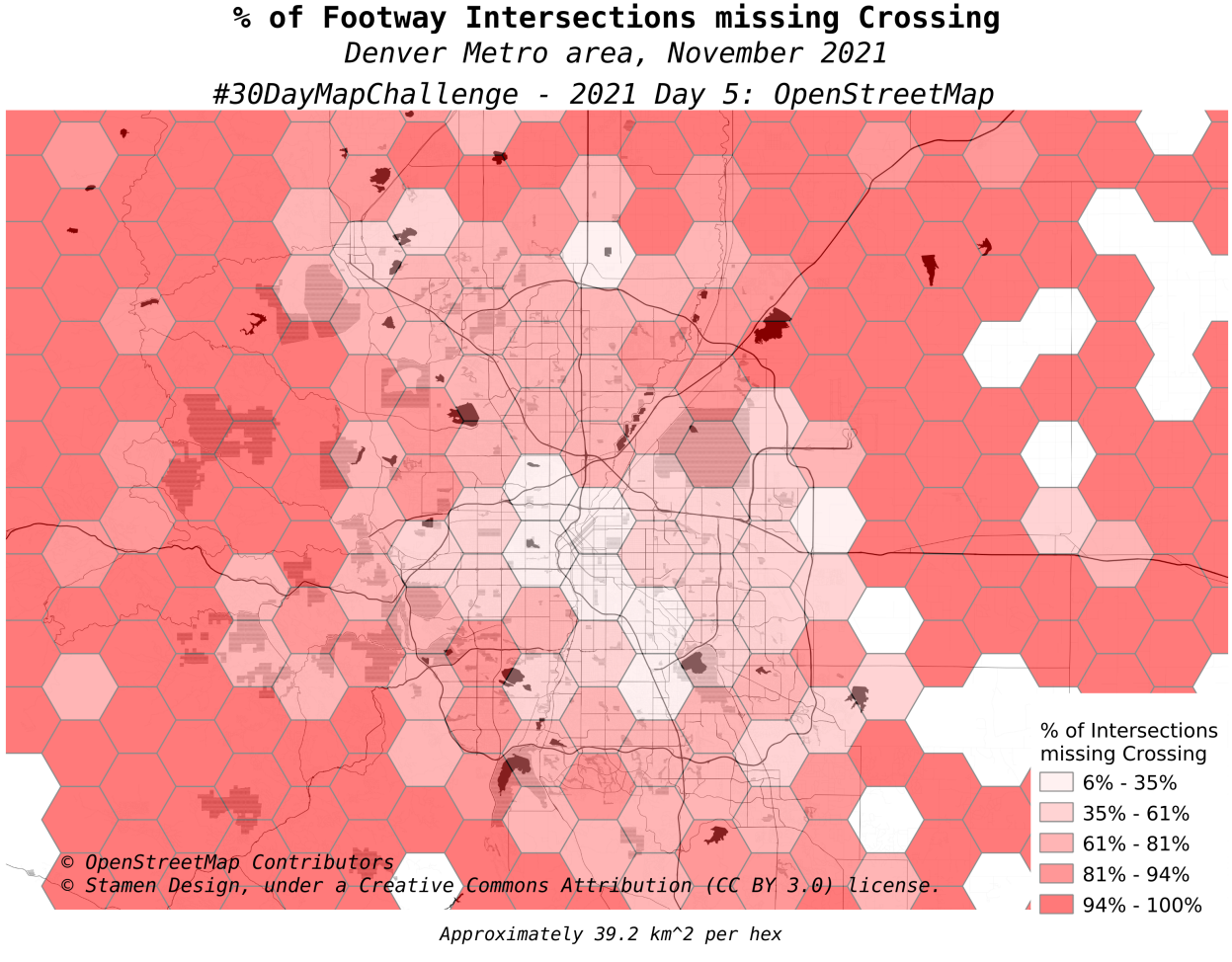
Geopolitical boundaries are generally not the best option for geospatial analysis

- Widely different sizes
- Do not scale from local to regional
- Tricky (or impossible) to use for trends

- https://carto.com/blog/zip-codes-spatial-analysis
- https://atcoordinates.info/2020/05/11/the-trouble-with-zip-codes-solutions-for-data-analysis-and-mapping/

Hexes for Visualization

Hexes Built-in to PostGIS



https://blog.rustprooflabs.com/2021/11/postgis-find-openstreetmap-missing-crossing

Hexes for Privacy

Hexes for Privacy

⚠️ Location is PII ⚠️

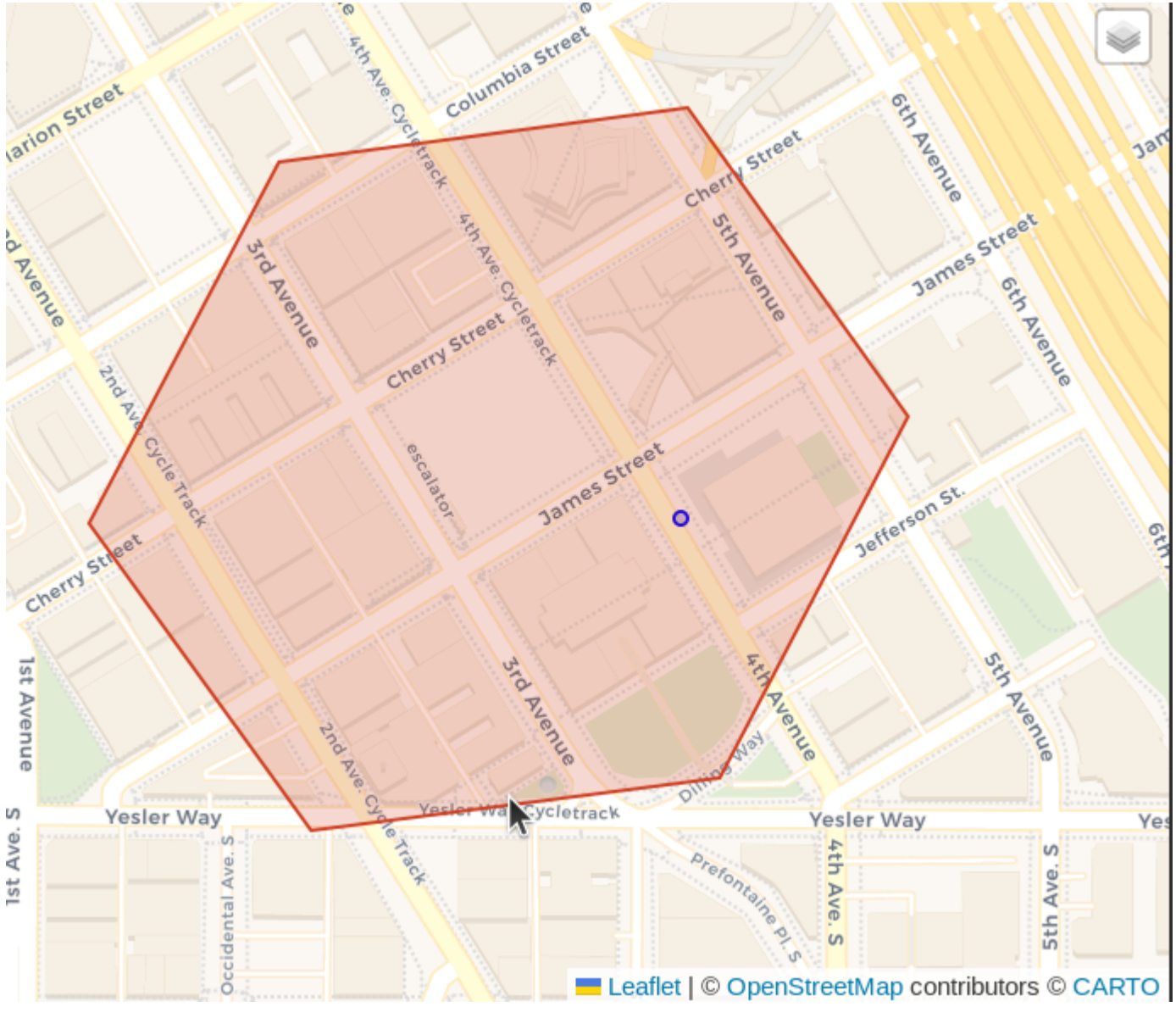
Hexes for Privacy

> *Do you **really** need to save exact customer locations?*

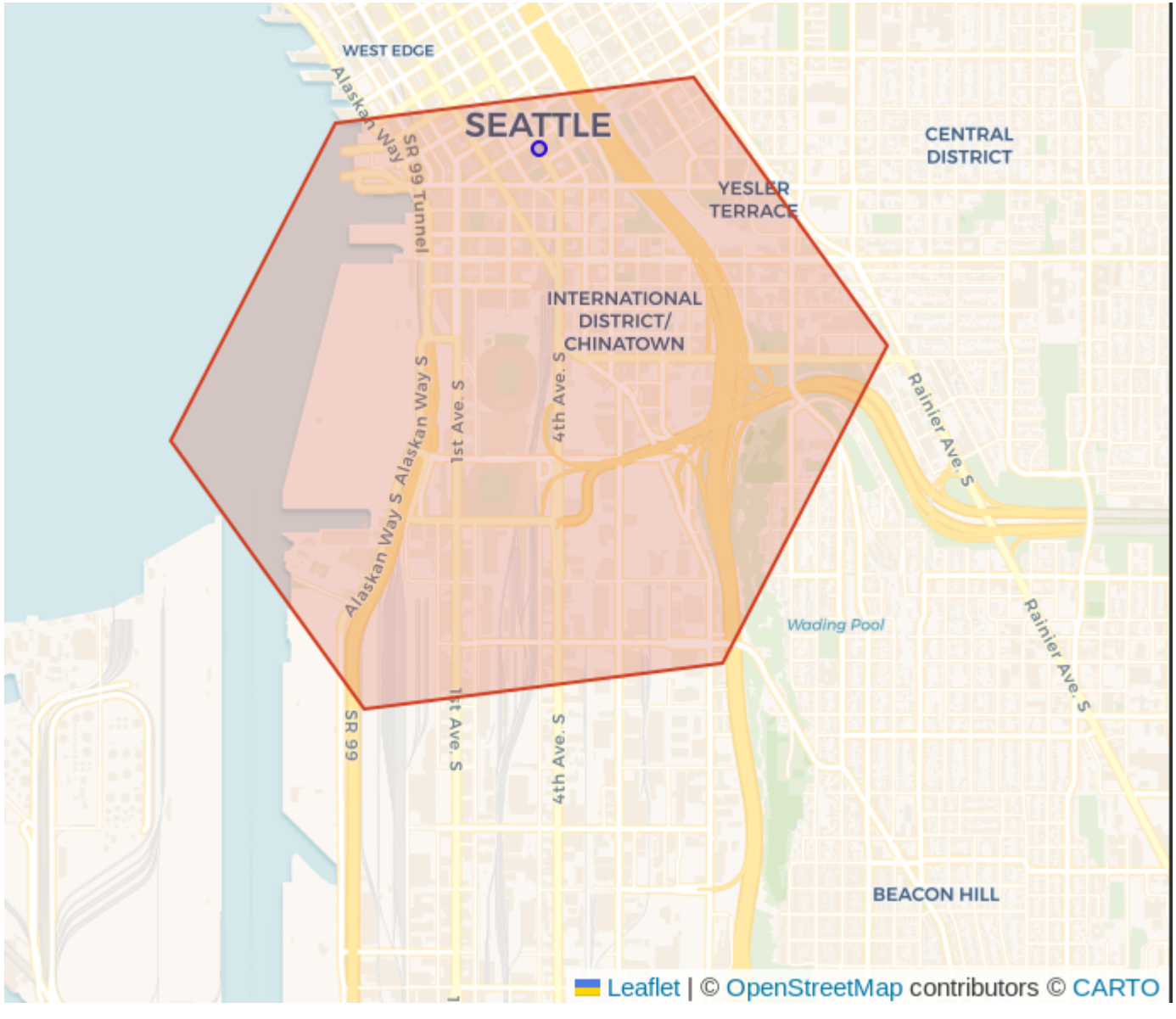
Data Incidents Happen

> *It's not **if**, it's **when***

"Customer" point vs. H3 Hex (res 9)



Customer point vs. H3 Hex (res 7)



Performance with H3 indexes

Performance with H3 indexes

- Nearest neighbor searches performed 73% - 77% faster
- Standard deviation of execution time is almost always lower compared to spatial join

https://blog.rustprooflabs.com/2022/06/h3-indexes-on-postgis-data

🤔 Performance with H3 indexes

🤔 Performance with H3 indexes

- Blog posts highlighting features often use ~~contrived~~ targeted examples

🤔 Performance with H3 indexes

- Blog posts highlighting features often use ~~contrived~~ targeted examples

Many real-world spatial queries...

- Do not run faster with H3 indexes
- Cannot be **easily** approximated with H3 indexes

Hexagon sources

- Internal PostGIS
- H3

Internal PostGIS Hexes

- `ST_HexagonGrid()`
- Abritrary grid based on input geometry

https://postgis.net/docs/en/ST_HexagonGrid.html

`ST_HexagonGrid()` covering region

```
CREATE TEMP TABLE my_hexes AS
WITH h as (
SELECT (ST_HexagonGrid(5000, ST_Envelope(ST_Collect(geom)))).*
    FROM osm_wa.place_polygon
    WHERE admin_level = 6
)
SELECT row_number() over(ORDER BY i, j) AS id,
        h.*
    FROM h
;
ALTER TABLE my_hexes ADD CONSTRAINT pk_hexes PRIMARY KEY (id);
```

`ST_HexagonGrid()`

Advantages

- Built in w/ PostGIS
- Easy to use
- Good for localized data

Disadvantages

- Every grid is custom to each input area
- Hard to share across projects

https://postgis.net/docs/manual-dev/ST_HexagonGrid.html

External Hexes

External Hexes

H3: `pg-h3` Extension

https://blog.rustprooflabs.com/2022/04/postgis-h3-intro

https://blog.rustprooflabs.com/2023/05/postgis-h3-v4-refresh

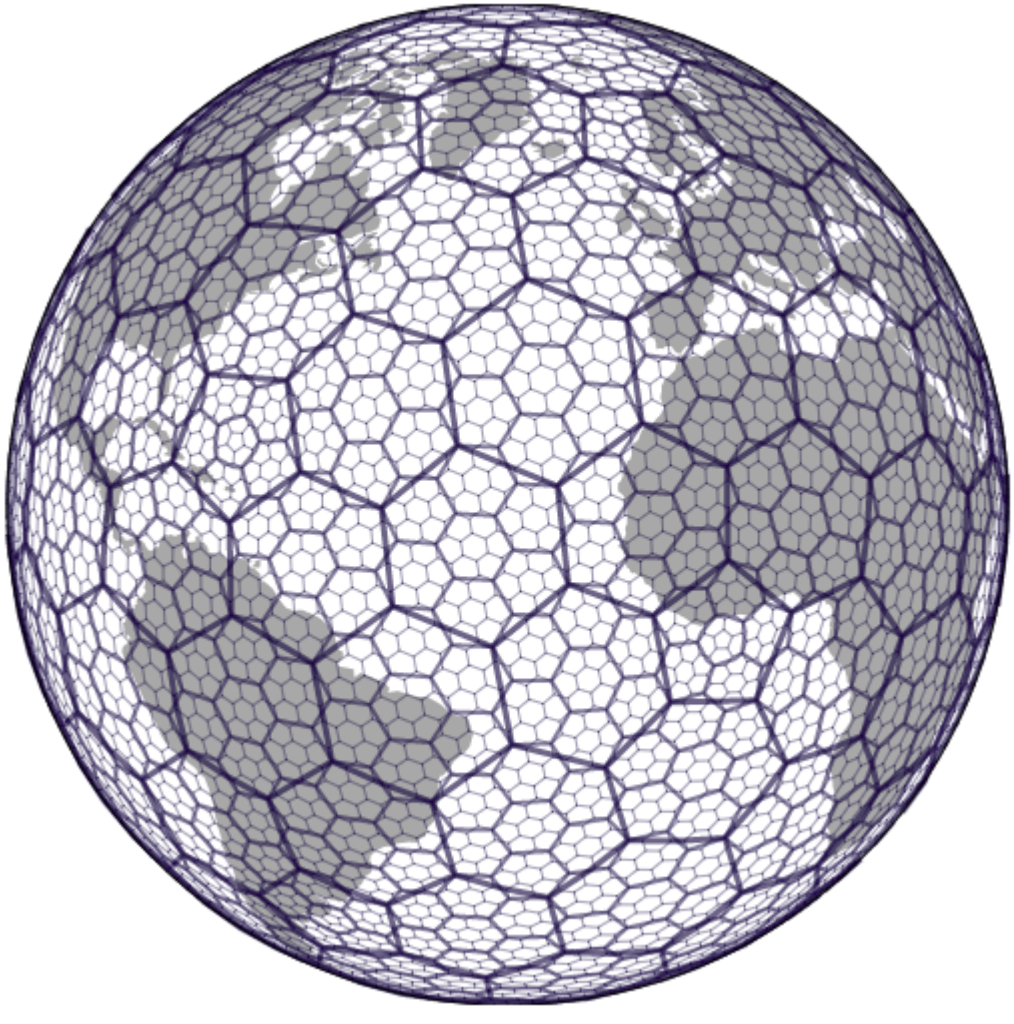
https://www.uber.com/blog/h3/

https://github.com/zachasme/h3-pg

https://h3geo.org/

Uber's H3

Source: https://www.uber.com/blog/h3/



How to plan for H3 indexes

- Is there ONE resolution you can always join on?
- What is the highest resolution you **really** need?
- Persist hexes to provide full coverage with `LEFT JOIN`
- Add `h3index` column to data (geocoded addresses)

H3 Resolution Sizes

Resolution 2: Seattle, plus a good chunk of Washington and into Canada `8228d7fffffffff`

Resolution 4: Wider Seattle area, ~ 600 sq. miles `8428d55ffffffff`

Resolution 6: Bellvue, 12.4 sq. miles `8628d5437ffffff`

H3 Resolution Sizes

Resolution 8: Part of Bridal Trails State Park, 0.25 sq. miles `8828d54143fffff`

Resolution 9: A few blocks for Bermerton, 0.04 sq. miles `8928d50a663ffff`

```
WHERE ix IN ('8228d7fffffffff', '8428d55fffffff',
                 '8628d5437ffffff', '8828d54143fffff',
                 '8928d50a663ffff')
```

H3 Resolution Sizes

Beware: H3 Uses SRID 4326

Beware: H3 Uses SRID 4326

Accidentally using 3857 or other SRIDs...

- Causes vague errors...
- ... or creates invalid Hexes
- https://github.com/zachasme/h3-pg/issues/130

Demo

- Creating H3 indexes - Generated column
- More `EXPLAIN (ANALYZE...)`

`03 - c .sql`

In [ ]: