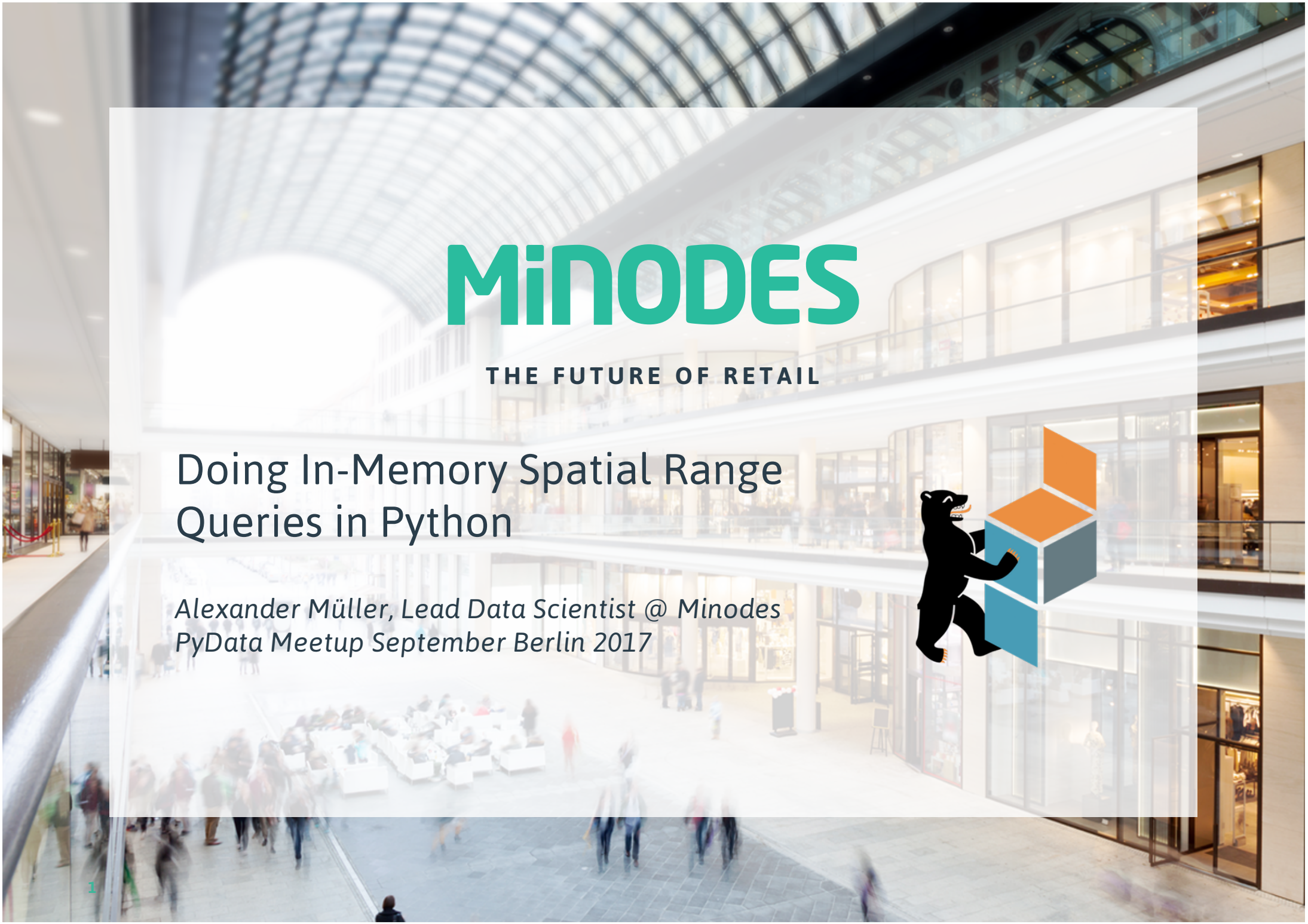
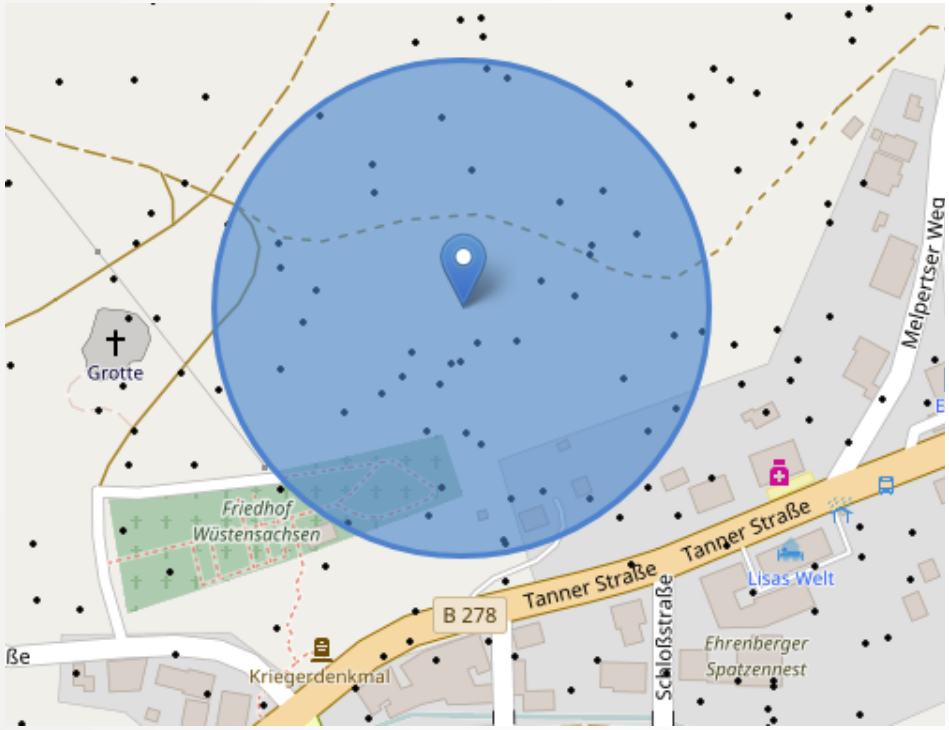


Doing In-Memory Spatial Range Queries in Python

Alexander Müller, Lead Data Scientist @ Minodes
PyData Meetup September Berlin 2017

OUR PROBLEM
GET THE STUFF THAT IS NEARBY



- **x00 Million** location events per day
- Assign to each location signal **Points of Interest (POI)** within a certain distance in meters
- Meaning we need be able to get for **any given point a list of POIS** within a given distance

WHAT OPTIONS TO WE HAVE?

PostgreSQL

+ PostGIS Indices

mongoDB

ADDING UP SOME NUMBERS (WORST CASE)

- **100 Million** spatial range queries per day
- Assumptions:
  - 50ms roundtrip time for postgres + postgis
  - 100ms roundtrip time for mongodb
- Comparing Naïve approaches
  - No batching, temp table etc.

	Postgres PostGIS	MongoDB
Query time in seconds	0.05	0.1
Amount of queries in million	100	100
Total querytime in hours	1388.9	2777.8
With Parallelism 40	34.7	69.4

We have a problem here since the day only has 24h

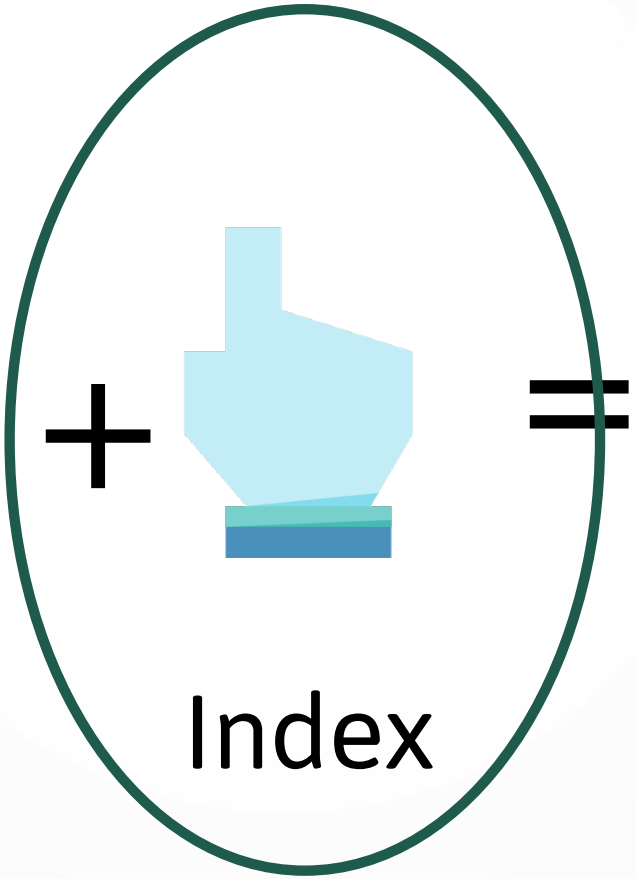
OK, LETS TRY TO DO IT BETTER THAN THIS

KEEP
CALM
AND TRUST ME
I´M AN
ENGINEER

INGREDIENTS TO BUILD YOUR OWN FAST RANGE QUERY TOOL

lat	long	Point of Interest
50.123	13.212	Mc Donalds
34.123	9.231	Burger King
34.123	9.231	Subway
54.123	19.231	Mustafas

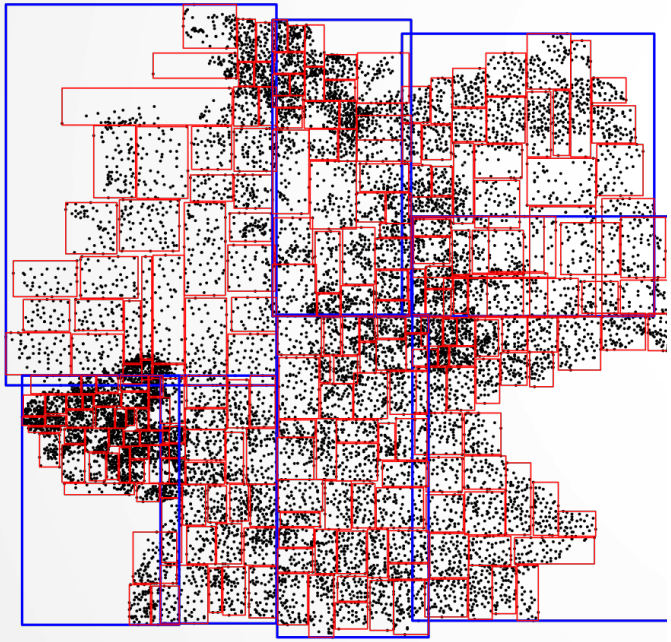
Data



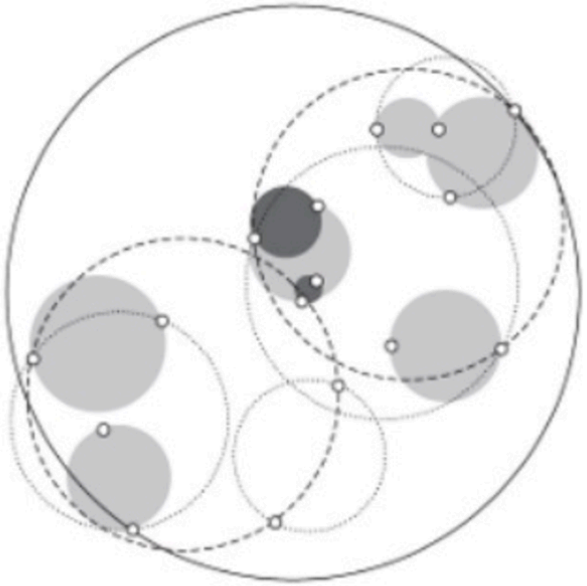
Index

Millisecond Query Time

WHAT KIND OF INDICES ARE OUT THERE



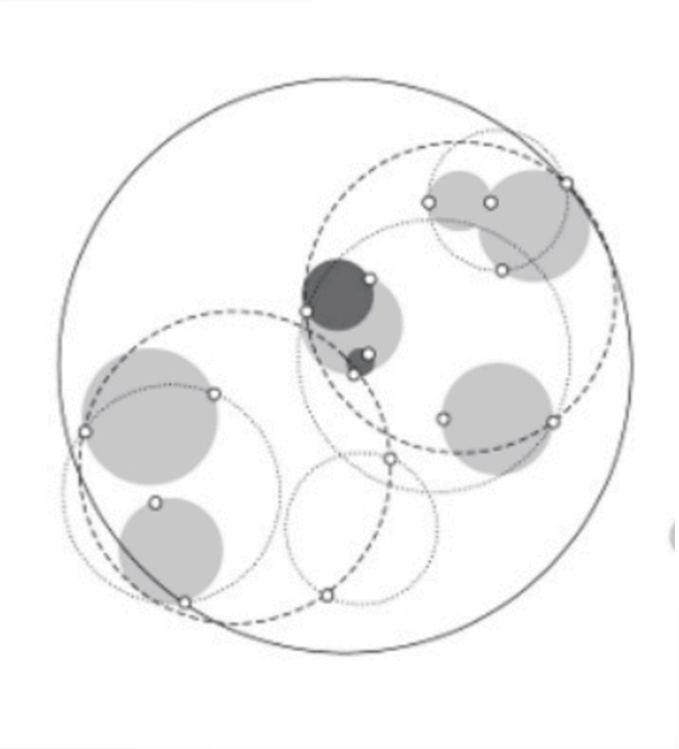
R-Trees



Ball Trees

(Geohashes)

WHAT KIND OF INDICES ARE OUT THERE

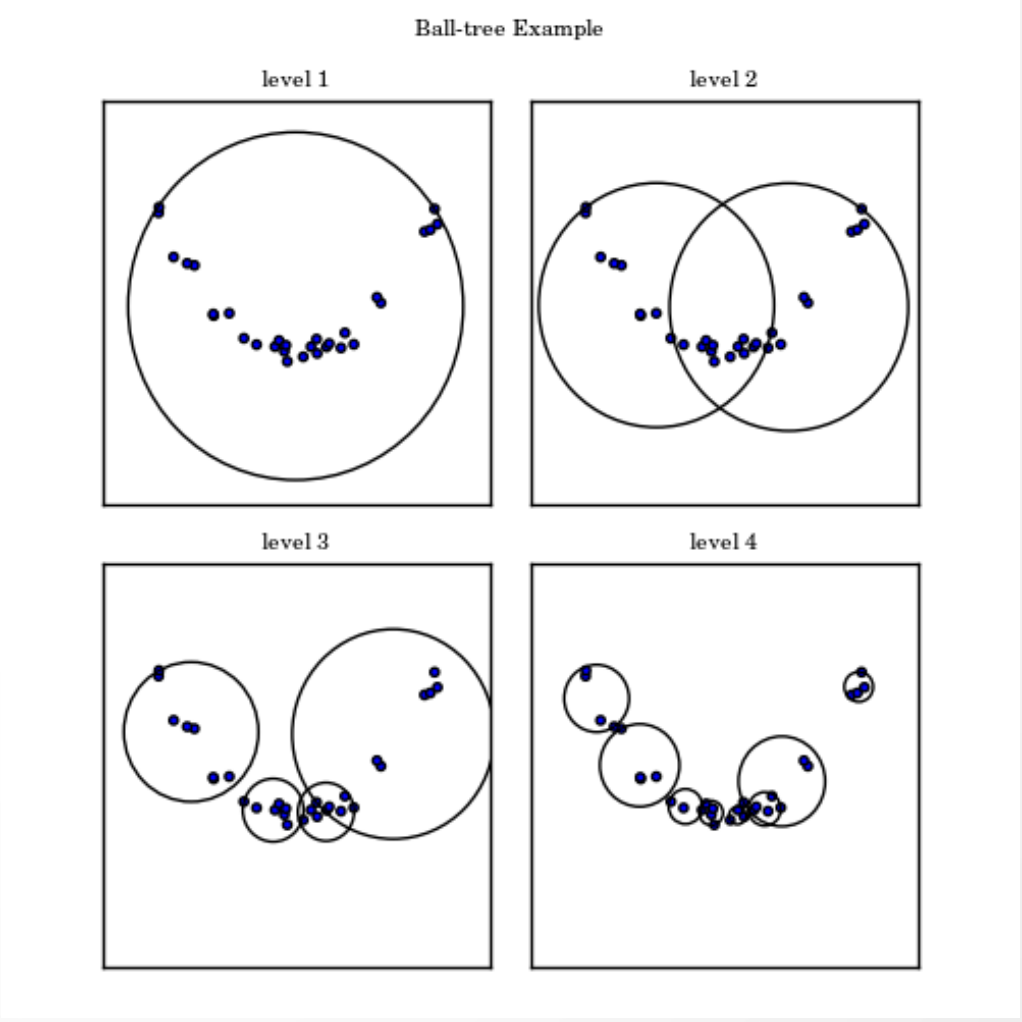


Ball Trees

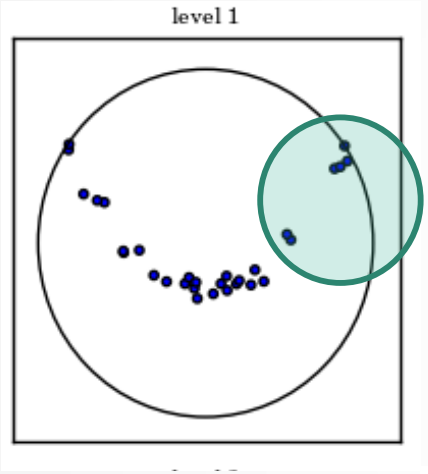
Geohashes

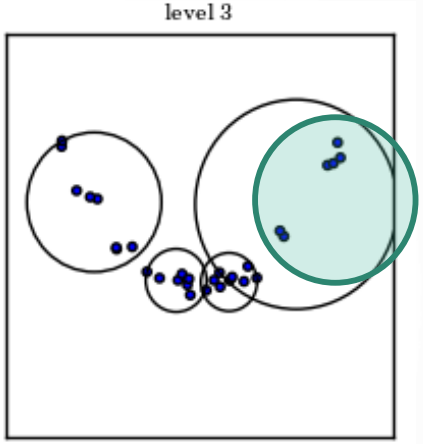
BALL TREE 101

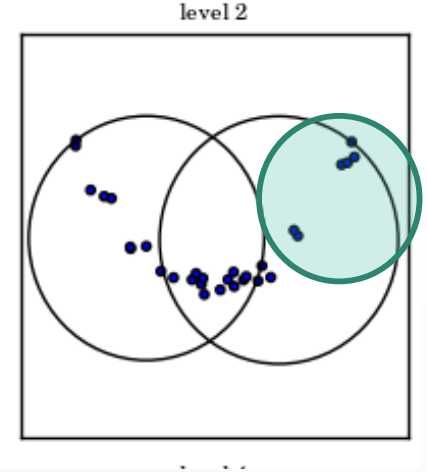
- Binary Tree-Based data partitioner
- Partitions data into nested sets of spheres (Balls)
- Due to the spherical property very well suited for range queries

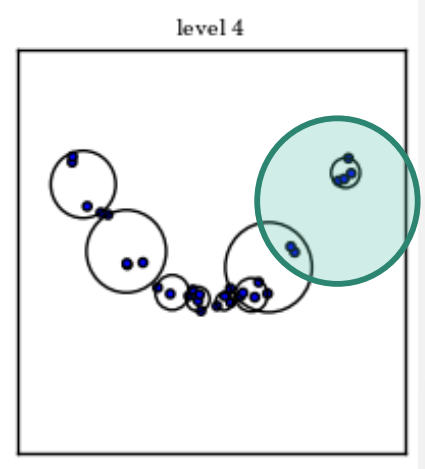


QUERY TIME
EFFICIENT SEARCH SPACE PRUNING AND LESS OPERATIONS









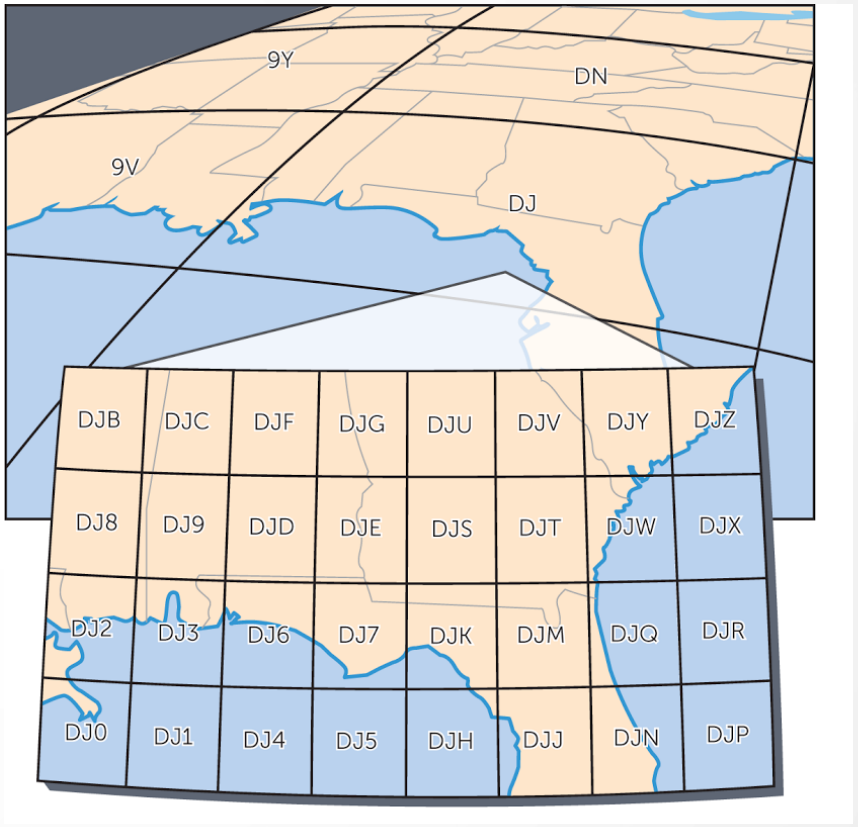
IMPLEMENTATION USING SKLEARN

Ball-Tree-based indexer

In [19]:
```
class BallTreeIndex:
    def __init__(self,lat_longs):
        self.lat_longs = np.radians(lat_longs)
        self.ball_tree_index =BallTree(self.lat_longs, metric='haversine')

    def query_radius(self,query,radius):
        radius_km = radius/1e3
        radius_radiant = radius_km / RADIANT_TO_KM_CONSTANT
        query = np.radians(np.array([query]))
        indices = self.ball_tree_index.query_radius(query,
                                                    r=radius_radiant)
        return indices[0]
```

GEOHASH 101

- A Geo-hash is a hierarchical structure that subdivides space into buckets of grid shape
- Each geohash is defined by a base32 value
- Very neat properties to drill up/down and get nearby geohashes



USE GEOHASHES TO ENABLE RANGE QUERIES

1. Create an index that maps the geohash to POIs
2. Get for the query point and its radius the geohashes that fit this space
3. Retrieve all candidate POIs
4. Perform a brute-force search on the set of candidates

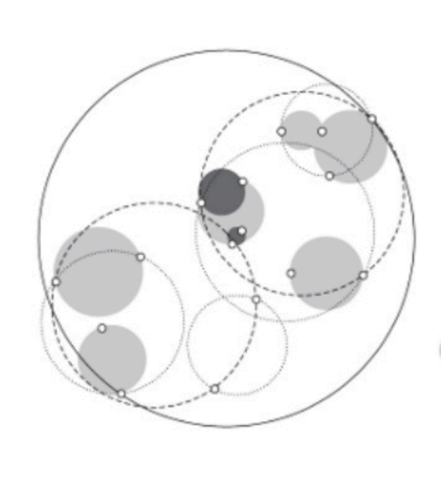
IMPLEMENTATION USING GEOHASH

Geohash-based indexer

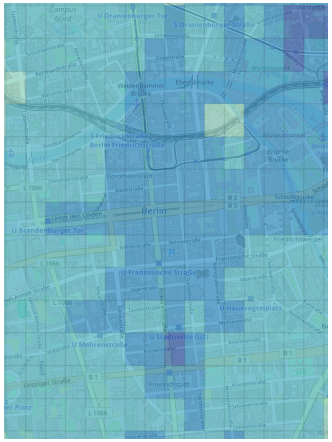
In [11]:

```
class GeoHashIndexer:

    def __init__(self,precision,lat_longs):
        self.index = defaultdict(list)
        # build the index
        for lat_long in lat_longs:
            lat = lat_long[0]
            long = lat_long[1]
            geo_hash = pgh.encode(lat, long, precision=precision )
            lat_long_radian = np.radians(np.array(lat_long))
            self.index[geo_hash].append(lat_long_radian)
        self.precision = precision
        self.haversine =DistanceMetric.get_metric('haversine')
        self.lat_longs = lat_longs

    def query_radius(self,query,radius):
        candidate_geohashes = proximitypyhash.get_geohash_radius_approximat
        candidate_points = []
        for geohash in candidate_geohashes:
            if geohash in self.index.keys():
                candidate_points.extend(self.index[geohash])

        if not candidate_points:
            return []

        query = np.radians(np.array([query]))
        result=self.haversine.pairwise(candidate_points,query)
        # convert radiant radius to meters
        radius_km = radius/1e3
        radius_radiant = radius_km / RADIANT_TO_KM_CONSTANT
        result = result[result<radius_radiant]
        return result*RADIANT_TO_KM_CONSTANT*1000 # get meters again
```

BENCHMARKING DIFFERENT APPROACHES



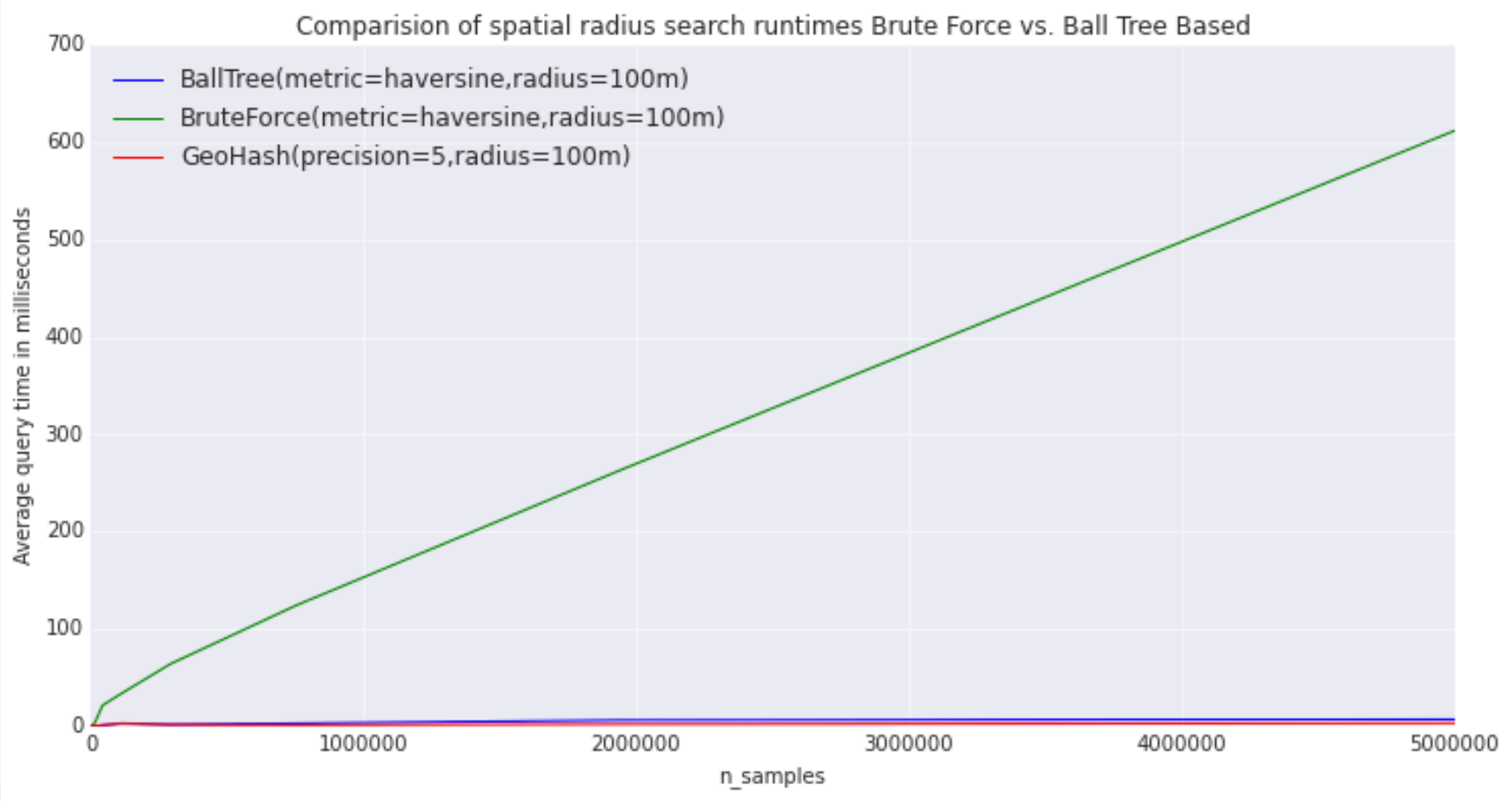
Vs.



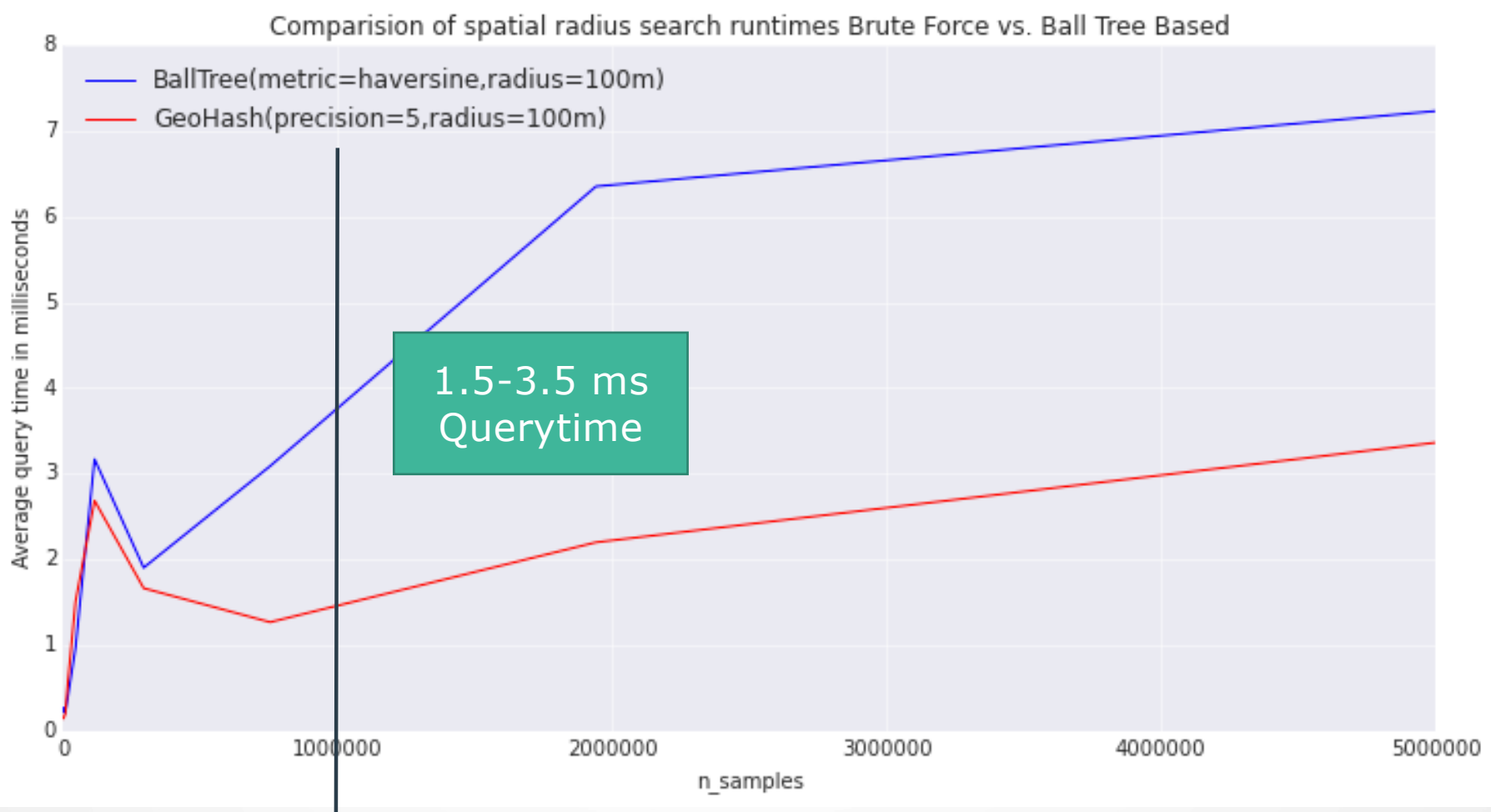
Vs.

Ball Tree Geohash Brute Force

BENCHMARKS


Comparision of spatial radius search runtimes Brute Force vs. Ball Tree Based
BallTree(metric=haversine,radius=100m)
BruteForce(metric=haversine,radius=100m)
GeoHash(precision=5,radius=100m)
Average query time in milliseconds
n_samples
0
100
200
300
400
500
600
700
0
1000000
2000000
3000000
4000000
5000000

BENCHMARKS


Comparision of spatial radius search runtimes Brute Force vs. Ball Tree Based
BallTree(metric=haversine,radius=100m)
GeoHash(precision=5,radius=100m)
Average query time in milliseconds
n_samples
0
1000000
2000000
3000000
4000000
5000000
0
1
2
3
4
5
6
7
8
1.5-3.5 ms
Querytime

RESULTS/ DISCUSSION

- Range queries can be performed very fast **in-memory in python**
- Only suitable for **long-lived objects** (like celery workers)
- **Scikit-learn** can be used for a bunch use cases
- **Big data = big problems**, because standard solutions often fail
- Enable you to use databases that don't have **GIS Features**

	Result
Query time in seconds	0.003
Amount of queries in million	100
Total querytime in hours	83.3
With Parallelism 40	2.1

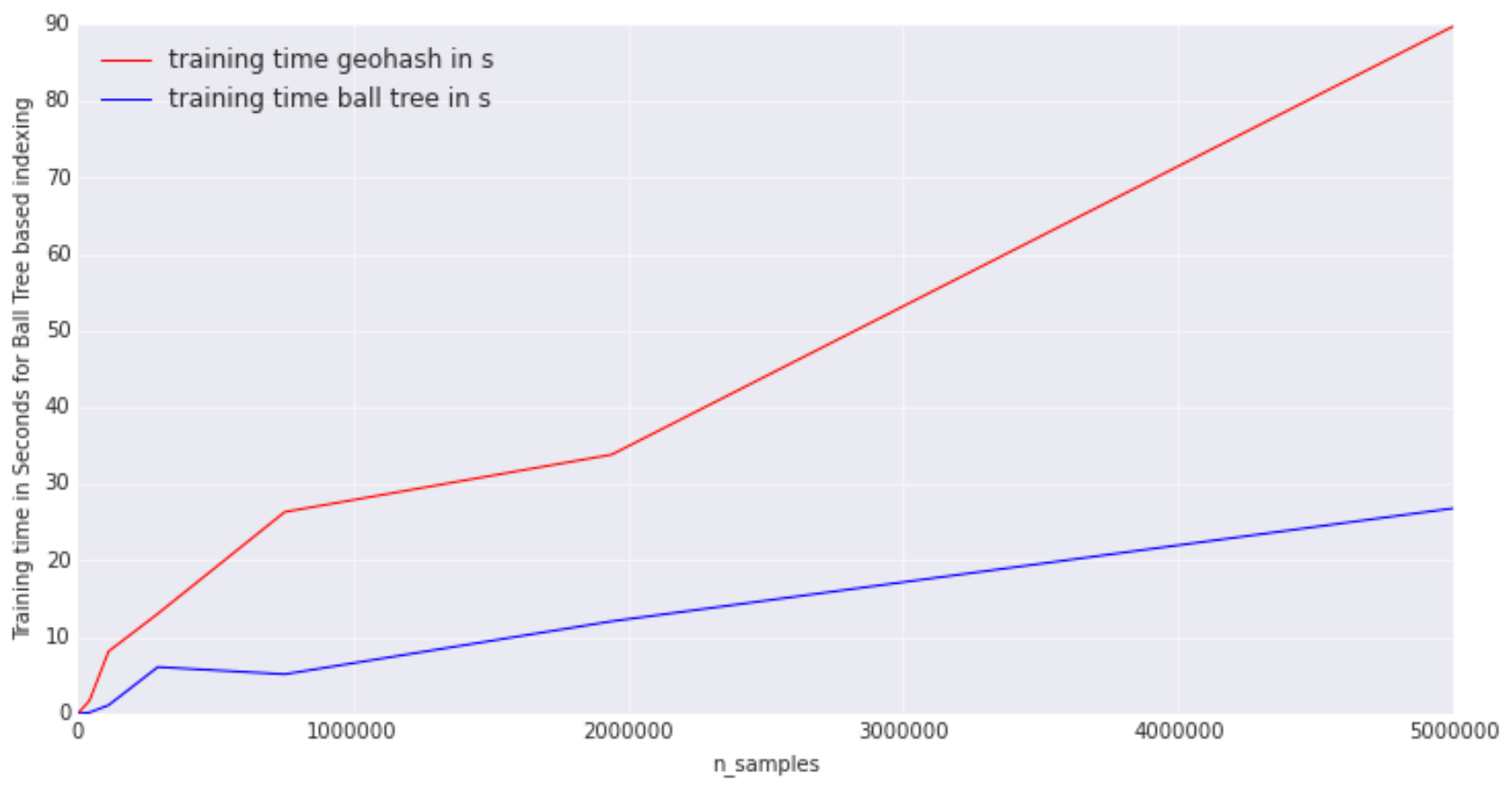
THANK YOU!

We’re hiring!

https://www.minodes.com/jobs

Backup slides

BENCHMARK TRAINING TIMES



RESOURCES

- CODE:
  - https://gist.github.com/dice89/e0c1bbf87021b0241ceb5fc6872720a1
- Original Blog Article
  - https://tech.minodes.com/experiments-with-in-memory-spatial-radius-queries-in-python-e40c9e66cf63
- https://en.wikipedia.org/wiki/R-tree
- http://geohash.org/
- https://www.factual.com/blog/how-geohashes-work
- https://en.wikipedia.org/wiki/Ball_tree
- https://en.wikipedia.org/wiki/World_Geodetic_System
- http://doi.ieeecomputersociety.org/10.1109/MCC.2016.65
- http://www.astroml.org/book_figures/chapter2/fig_balltree_example.html