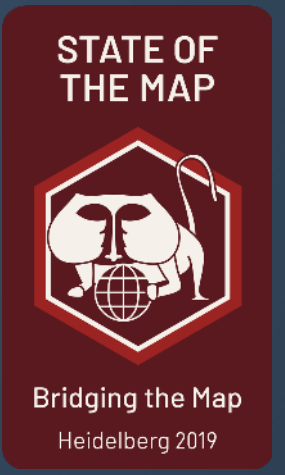
Spatial indexes for OSM in PostGIS

https://slides.com/fxku/sotm19

Felix Kunde

zalando



```
CREATE INDEX pts_spx
ON point_table
USING GIST (geom)
```

```
CREATE INDEX pts_spx
ON point_table
USING GIST (geom)
```

BRIN *< v2.3*

```
CREATE INDEX pts_spx
ON point_table
USING GIST (geom)
```

BRIN *< v2.3*

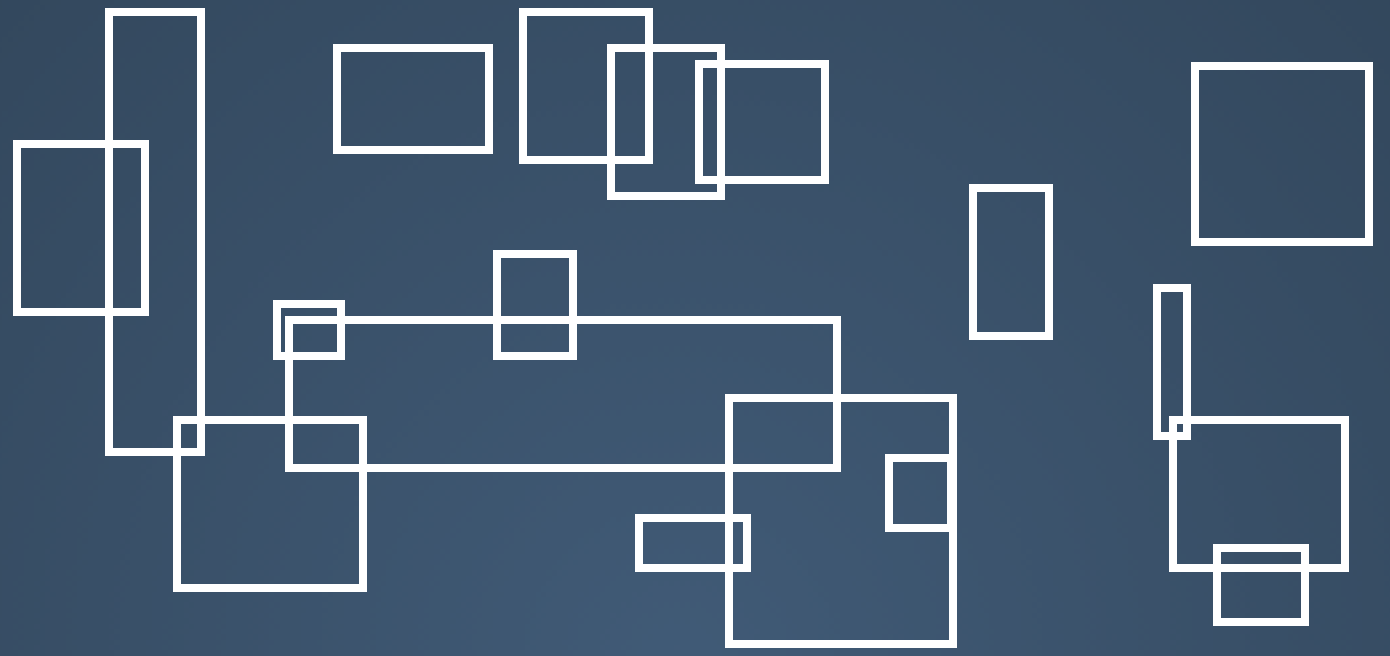
SPGIST *< v2.5*

GiST

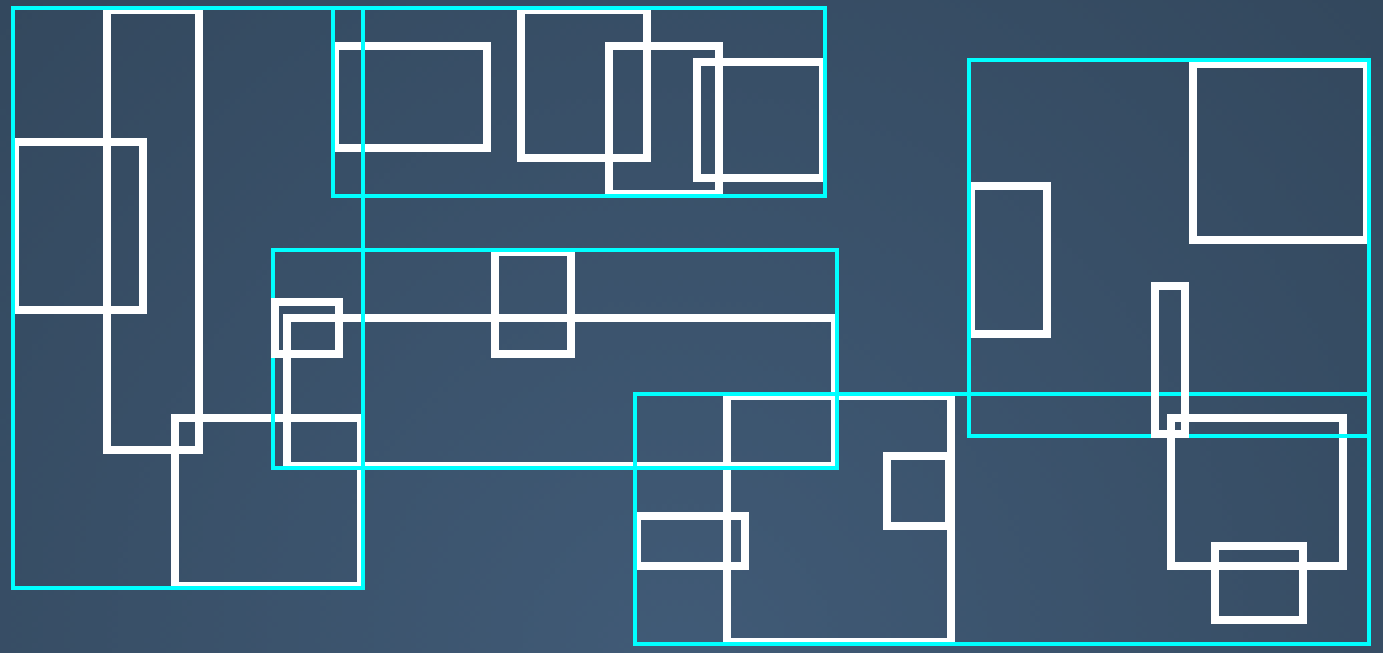
It's a framework

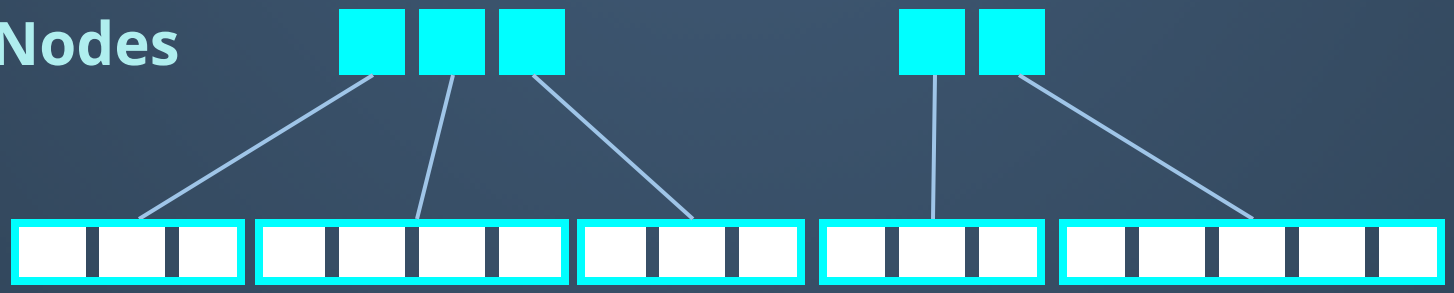
PostGIS > R-Tree

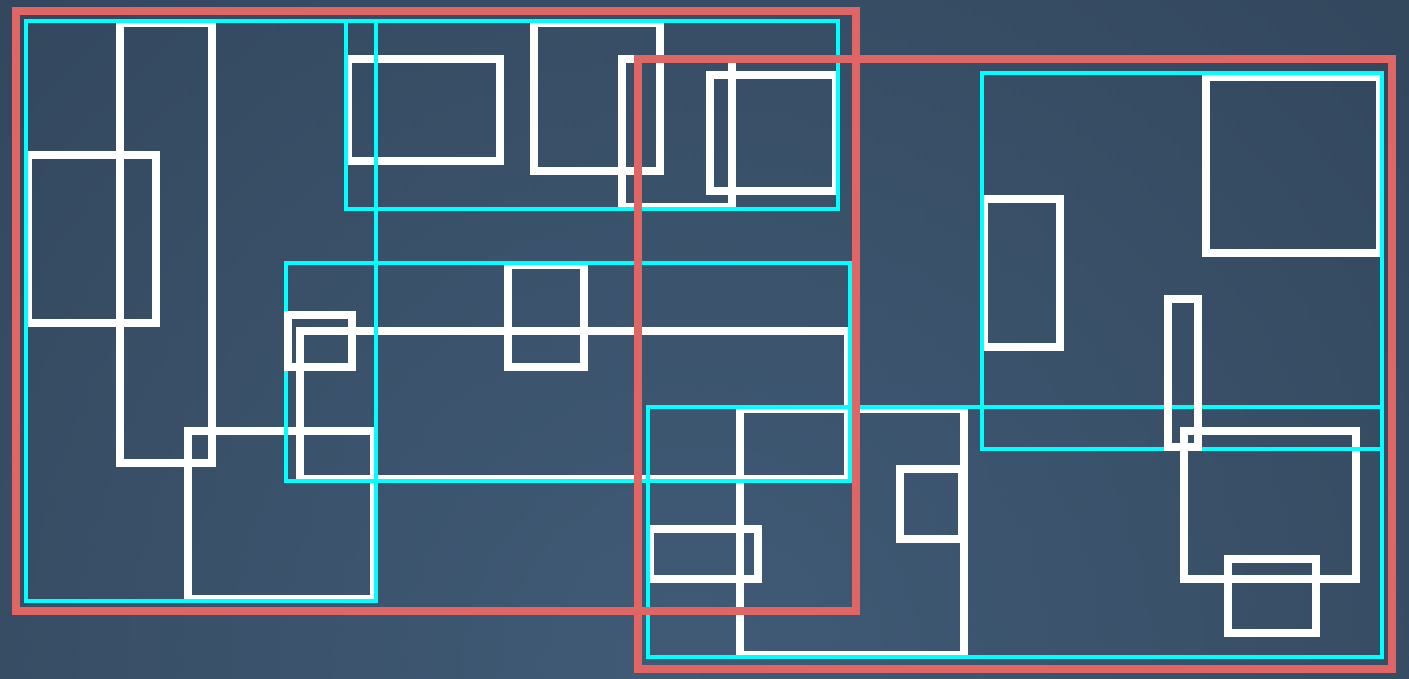
2-nD and kNN support

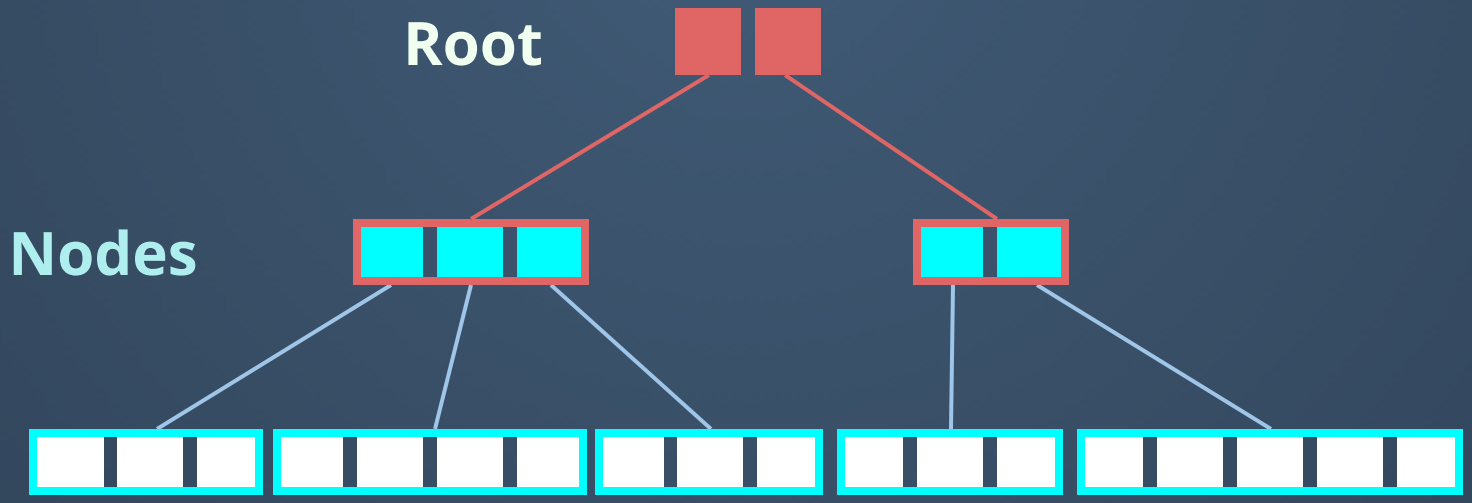


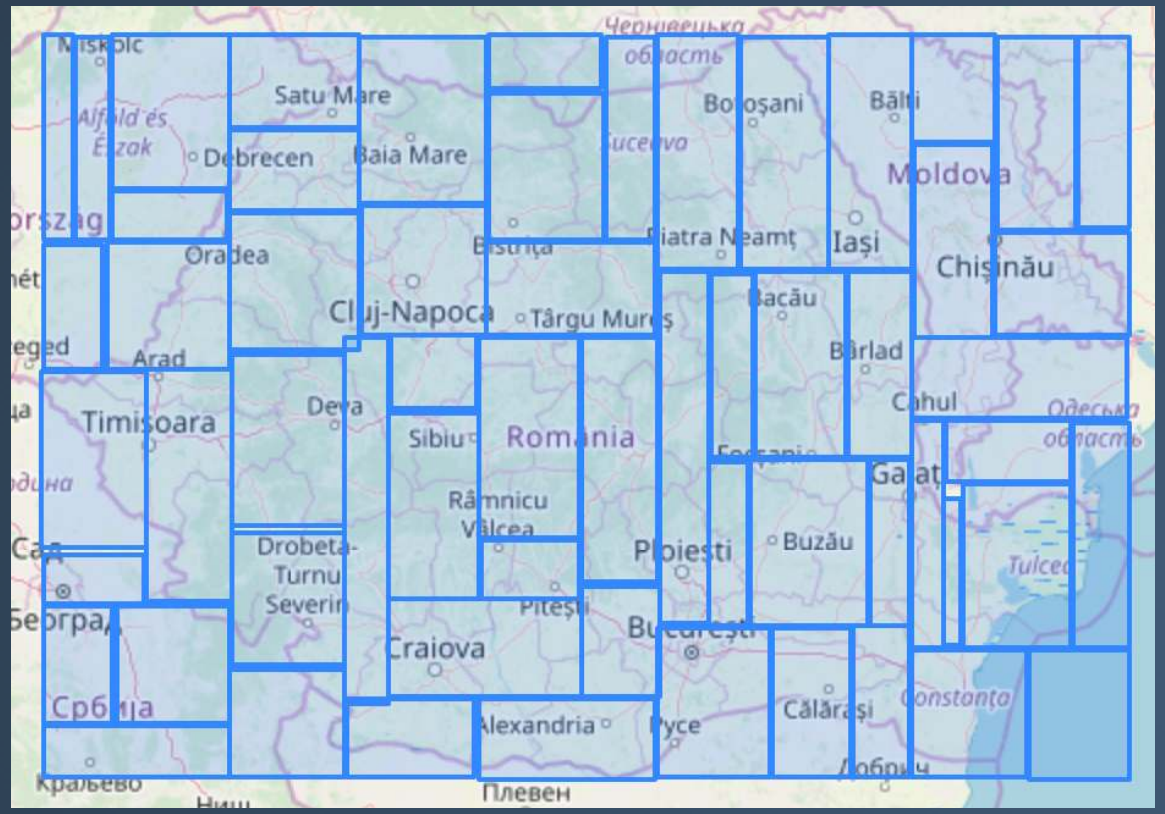
Leaves




Nodes




Root
Nodes



Level 1

Level 2

Produced with Gevel

ST_Intersects in ms

tests	100 k	1 Mio	10 Mio	100 Mio	1 Bn
no index	18.00	87.00	670	6473	135529
bulk	0.21	14.00	19	146	1568
online	0.18	15.00	29	163	1672
vacuum	0.16	0.87	18	145	1551
cluster	0.13	0.64	16	32	214

ST_Intersects in ms

tests	100 k	1 Mio	10 Mio	100 Mio	1 Bn
no index	18.00	87.00	670	6473	135529
bulk	0.21	14.00	19	146	1568
online	0.18	15.00	29	163	1672
vacuum	0.16	0.87	18	145	1551
cluster	0.13	0.64	16	32	214

ST_Intersects in ms

tests	100 k	1 Mio	10 Mio	100 Mio	1 Bn
no index	18.00	87.00	670	6473	135529
bulk	0.21	14.00	19	146	1568
online	0.18	15.00	29	163	1672
vacuum	0.16	0.87	18	145	1551
cluster	0.13	0.64	16	32	214

ST_Intersects in ms

tests	100 k	1 Mio	10 Mio	100 Mio	1 Bn
no index	18.00	87.00	670	6473	135529
bulk	0.21	14.00	19	146	1568
online	0.18	15.00	29	163	1672
vacuum	0.16	0.87	18	145	1551
cluster	0.13	0.64	16	32	214

BRIN

Block ranges. That's all.

Data should be sorted!

PostGIS > BBox per block

2-4D, no kNN

Block Range Index examples

...	time	geom
...	2019-03-13	POINT(13.8 50)
...	2019-03-13	POINT(13.8 50)
	...	...
	2019-03-14	POINT(13.8 51)
	2019-03-16	POINT(13.7 51)
	...	...
...	2019-03-16	POINT(13.7 51)
...	2019-03-16	POINT(13.7 51)

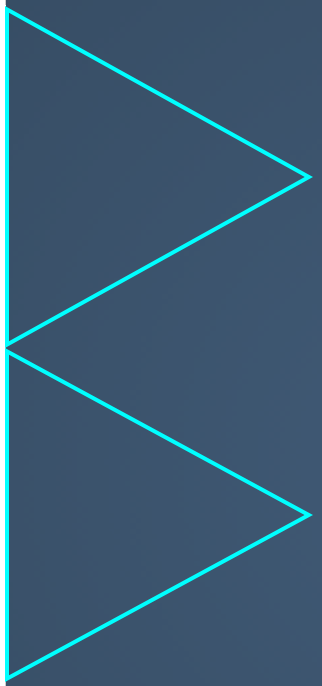
(2019-03-13 09:00:00 , 2019-03-14 11:00:00)

(A-Weg , Grunaer Str.)

BBOX

Block Range Index examples

...	time	geom
...	2019-03-13	POINT(13.8 50)
...	2019-03-13	POINT(13.8 50)
	...	...
	2019-03-14	POINT(13.8 51)
	2019-03-16	POINT(13.7 51)
	...	...
...	2019-03-16	POINT(13.7 51)
...	2019-03-16	POINT(13.7 51)



(2019-03-13 09:00:00 , 2019-03-14 11:00:00)

(A-Weg , Grunaer Str.)

BBOX

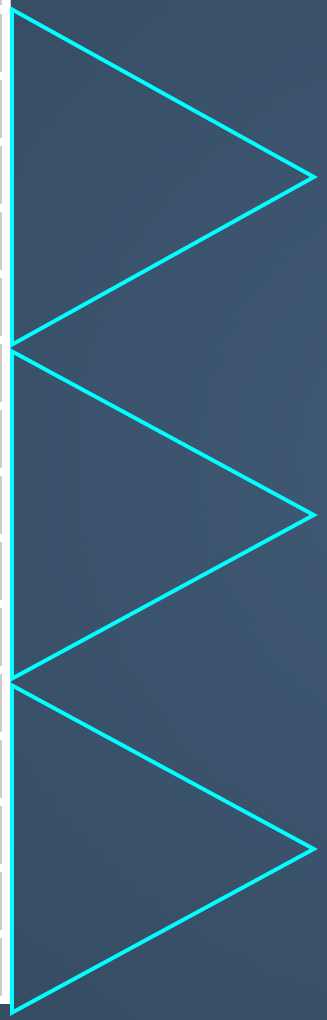
(2019-03-14 11:15:00 , 2019-03-15 18:00:00)

(Grunaer Weg, Nürnberger Str.)

BBOX

Block Range Index examples

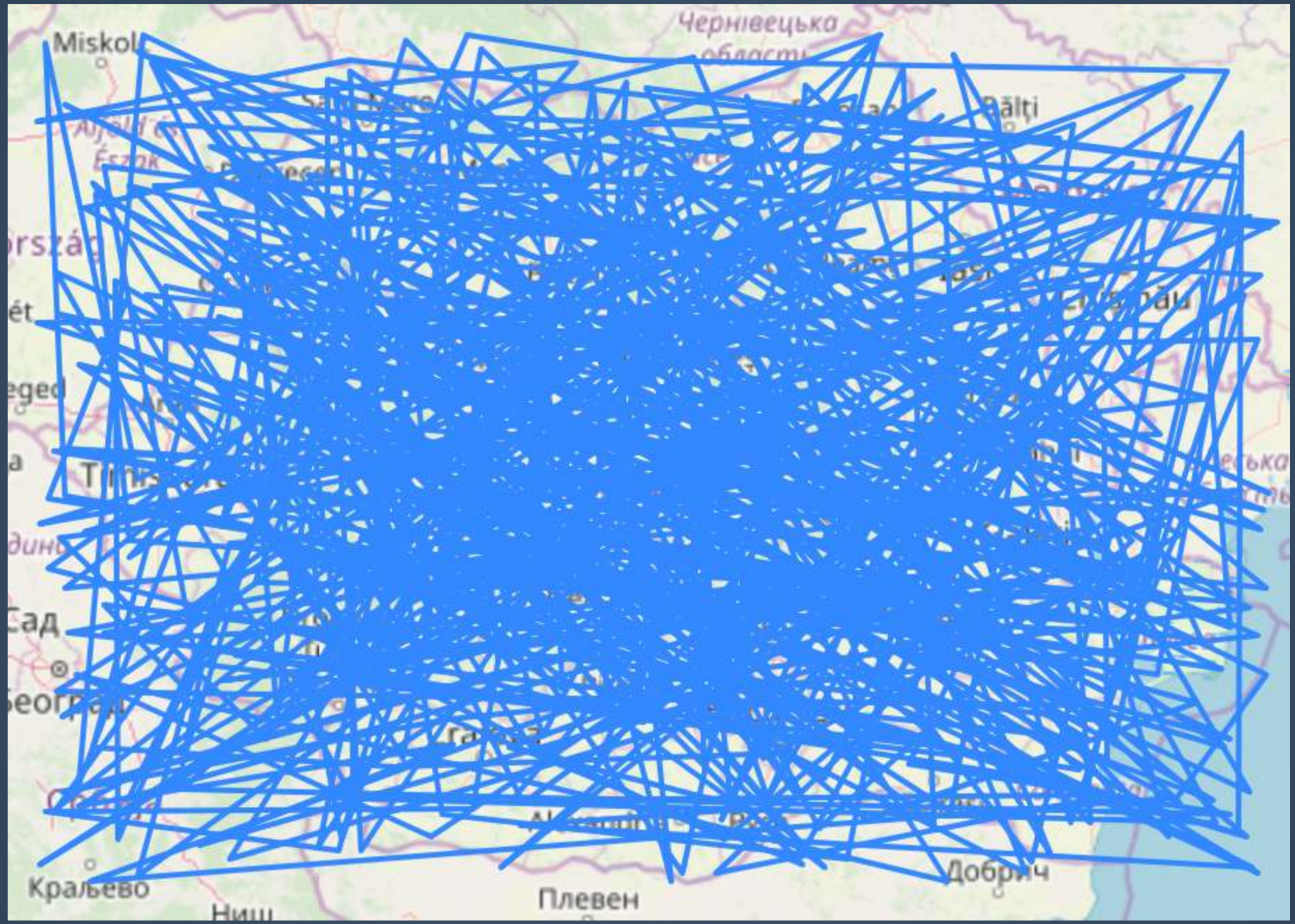
...	time	geom
...	2019-03-13	POINT(13.8 50)
...	2019-03-13	POINT(13.8 50)
	...	...
	2019-03-14	POINT(13.8 51)
	2019-03-16	POINT(13.7 51)
	...	...
...	2019-03-16	POINT(13.7 51)
...	2019-03-16	POINT(13.7 51)



(2019-03-13 09:00:00 , 2019-03-14 11:00:00)
(A-Weg , Grunaer Str.)
BBOX

(2019-03-14 11:15:00 , 2019-03-15 18:00:00)
(Grunaer Weg, Nürnberger Str.)
BBOX

(2019-03-16 09:00:00 , 2019-03-16 17:30:00)
(Oberauer Str. , Zwinglistr.)
BBOX


Miskol
Чернівецька
Plevén
Кральево
Добрич

ORDER BY geom

(PostGIS v3.0 uses Hilbert Curve)

ORDER BY
ST_GeoHash(geom)

```
SET enable_seqscan = false;
```

(discourage Postgres' query planner to use seq scans)

BRIN vs. GiST

tests	100 k	1 Mio	10 Mio	100 Mio	1 Bn
create gist	700 ms	8 sec	2 min	23 min	6 hrs
create brin	24 ms	0.2 sec	2 sec	18 sec	90 sec
size gist	5 MB	50 MB	500 MB	5 GB	50 GB
size brin	24 KB	24 KB	48 KB	376 KB	3,6 MB
duration	x25	x23	x1,4	x1.6	x1.1

sp-GiST

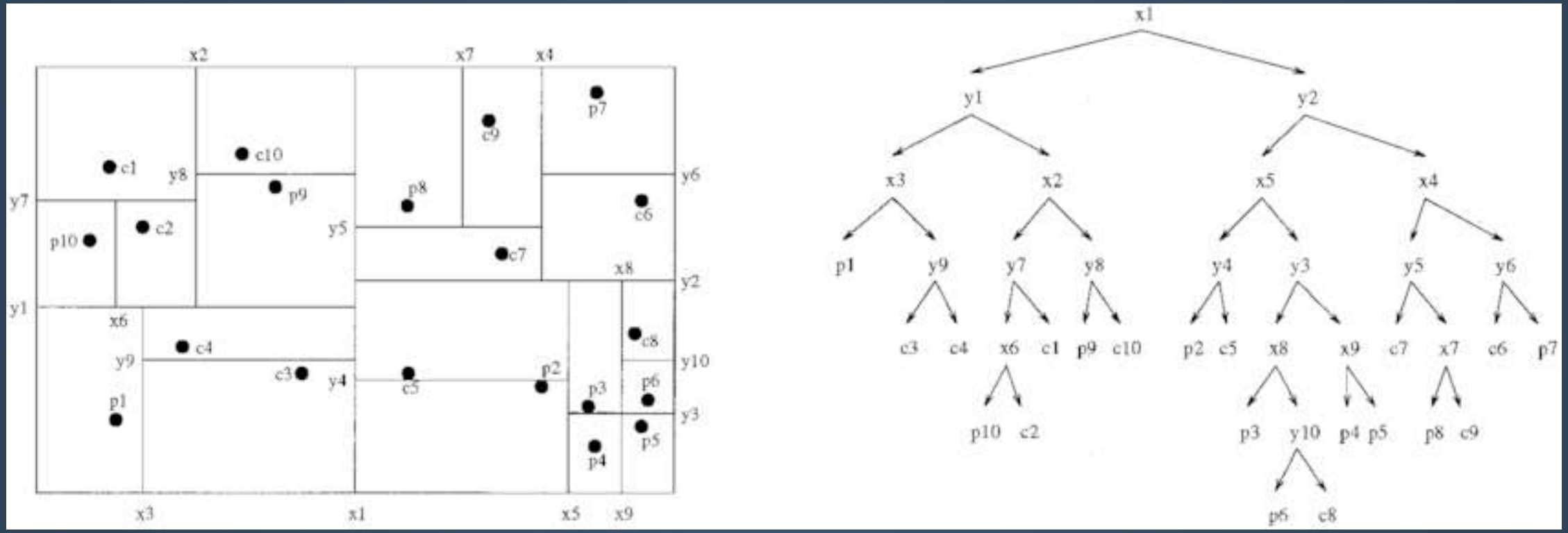
Framework like GIST

Unbalanced tree

No overlaps & prefixes

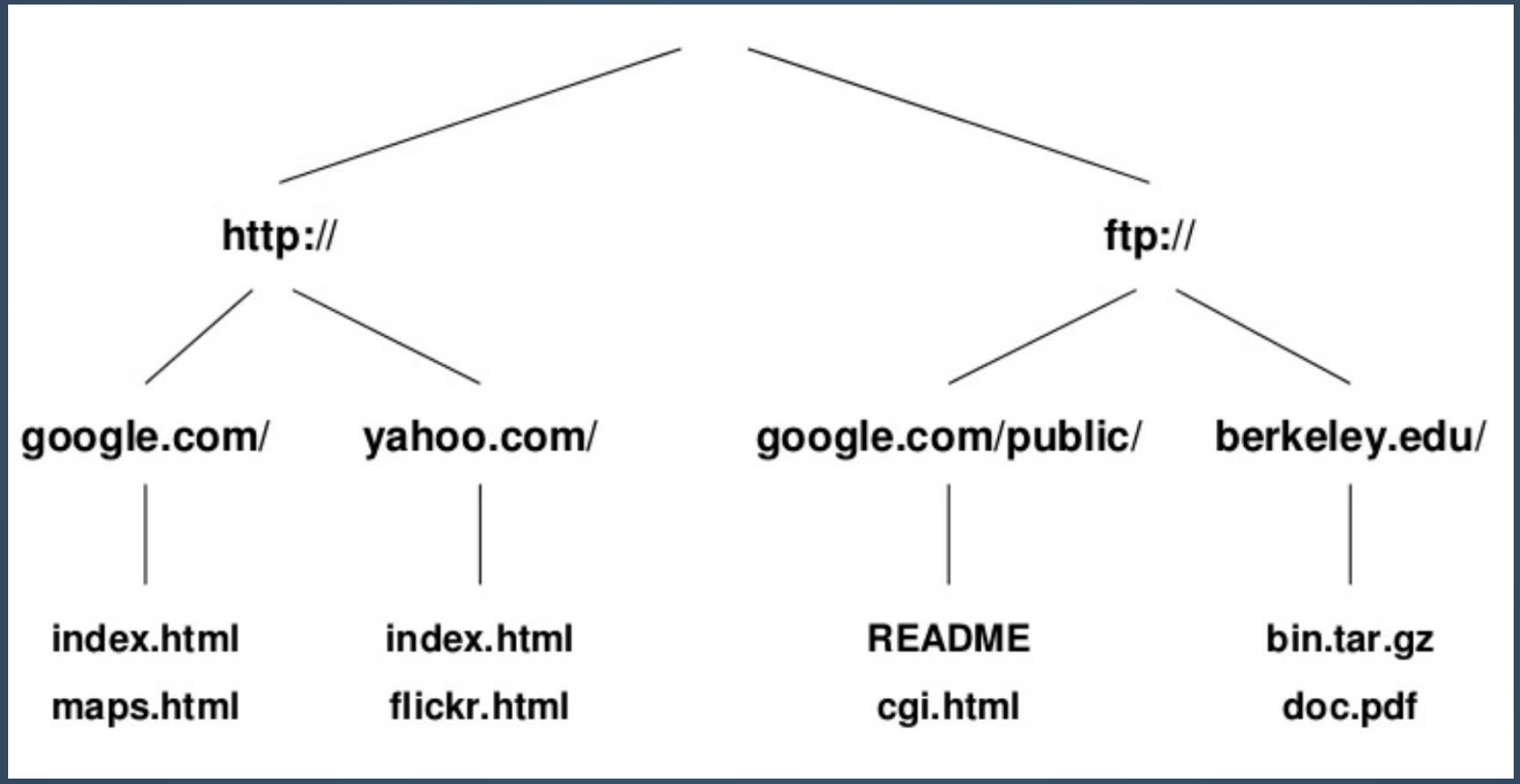
2-nD, no kNN, yet

kd-Tree, Quadtree



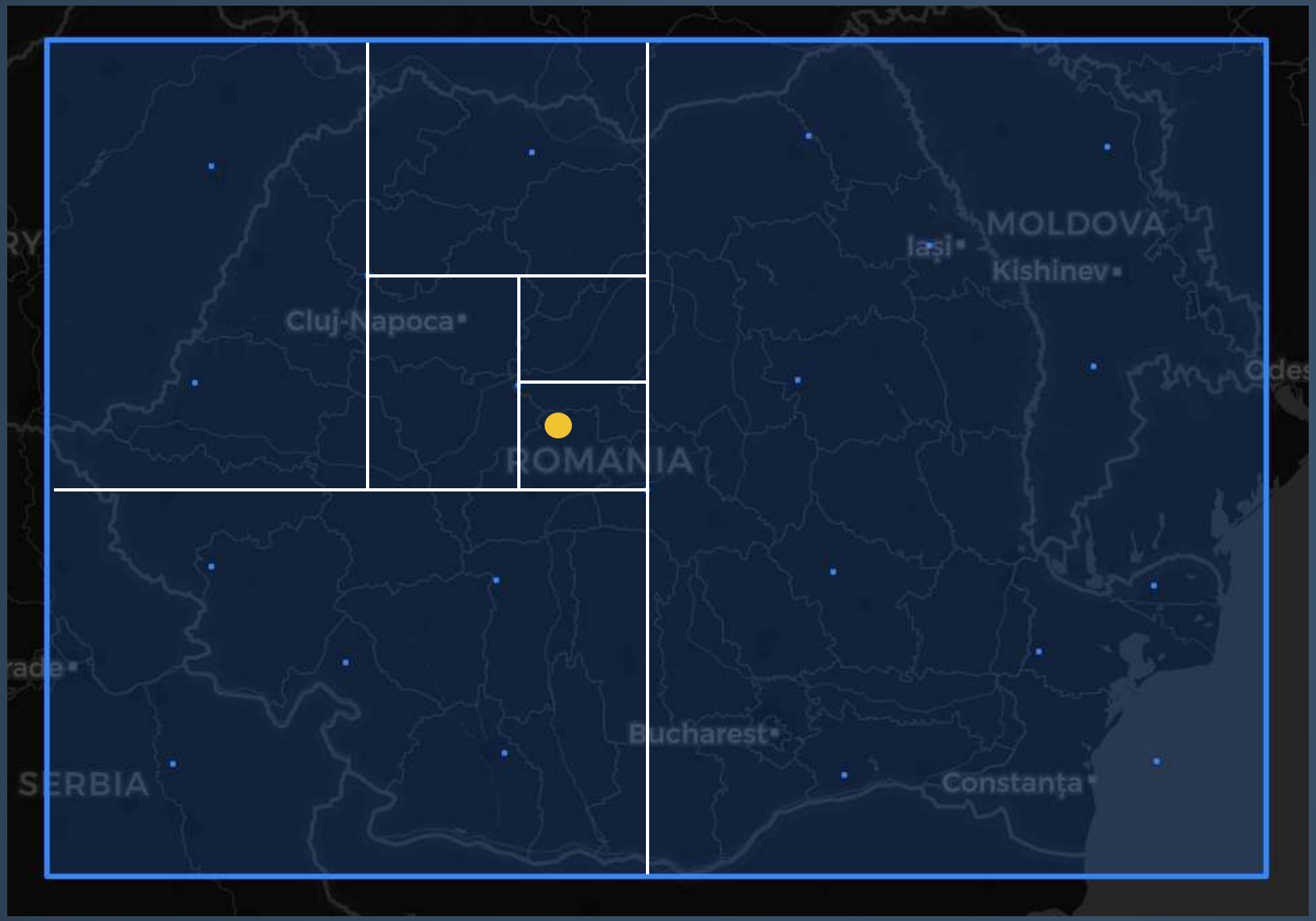
https://www.researchgate.net/figure/Adaptive-k-d-tree_fig9_2334587

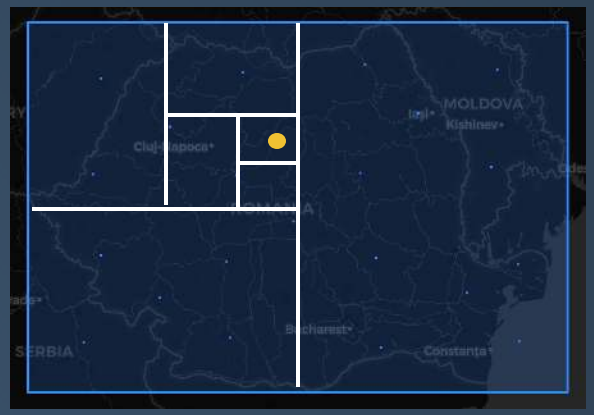
Split values between index and leaf nodes



Momjian 2019: Flexible Indexing with Postgres [PDF]

Trick: No Overlap via more dimensions





Each point you see on the map are in fact 4 bounding boxes which are the prefixes of the sp-GiST tree defining the bounds of child quadrants

sp-GiST vs. GiST

tests	100 k	1 Mio	10 Mio	100 Mio	1 Bn
create gist	700 ms	8 sec	2 min	23 min	6 hrs
create spgist	344 ms	3,7 sec	50 sec	11 min	8 hrs
size gist	5 MB	50 MB	500 MB	5 GB	50 GB
size spgist	4,5 MB	44 MB	440 MB	4.3 GB	43 GB
duration	x0.85	x1	x1	x1	x1.1

Thematic filters

Multi-column index

```
CREATE INDEX idx ON planet_osm_point
USING GIST (way, power);
```

Requires btree_gist extension

Spatial column first as it is more selective

Less read from heap

Index will be bigger, combine with ...

Partial index

```
CREATE INDEX idx ON planet_osm_point
USING GIST (way)
WHERE power = 'pole';
```

Smaller index, less IO overhead

Very specific to a certain task

OSM people do this

Covering index

```
CREATE INDEX idx ON planet_osm_point
USING GIST (way)
INCLUDE (power);
```

"Payload" only in the leaves (not sorted)

No big speed difference to multi-column index

GiST support in Postgres 12

Btw: Index-only scans?

Not possible due to compression

Requires new index tuple format with SRID

For small geoms (Points, BBox, ..?)

Breaking all GiSTs out there?

GeoHash index

```
CREATE INDEX idx ON planet_osm_point
USING BTREE (
  ST_GeoHash(ST_Transform(way,4326)), power
)
WHERE power IS NOT NULL;
```

BTREE benefits (parallel build, index-only-scan)

Spatial filtering tricky (GeoHash grid)

Slower, so only useful for table clustering

Radix tree

```
CREATE INDEX idx ON planet_osm_point
USING SPGIST (
  ST_GeoHash(ST_Transform(way,4326)) text_ops
)
WHERE power IS NOT NULL;
```

GeoHash and SPGIST seem like a great fit

LIKE not supported, but can use ranges

Slow (only two prefixes on small dataset)

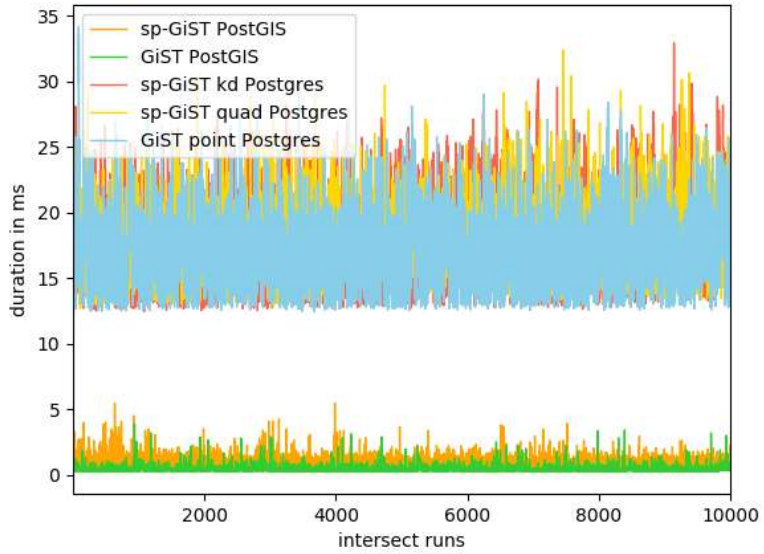
Postgres geometry

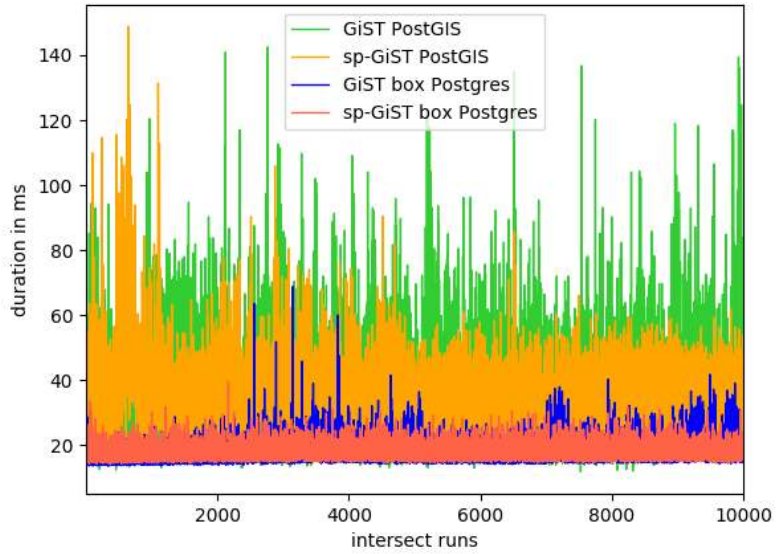
```
CREATE INDEX idx ON planet_osm_point
USING SPGIST (
  CAST (way AS point) quad_point_ops
)
WHERE power IS NOT NULL;
```

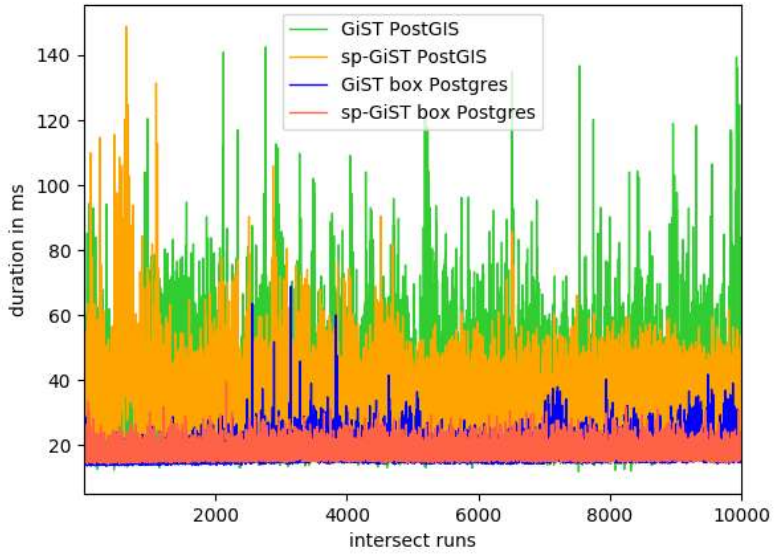
Queries need casting but it works

And performance is good (smaller tuple layout)

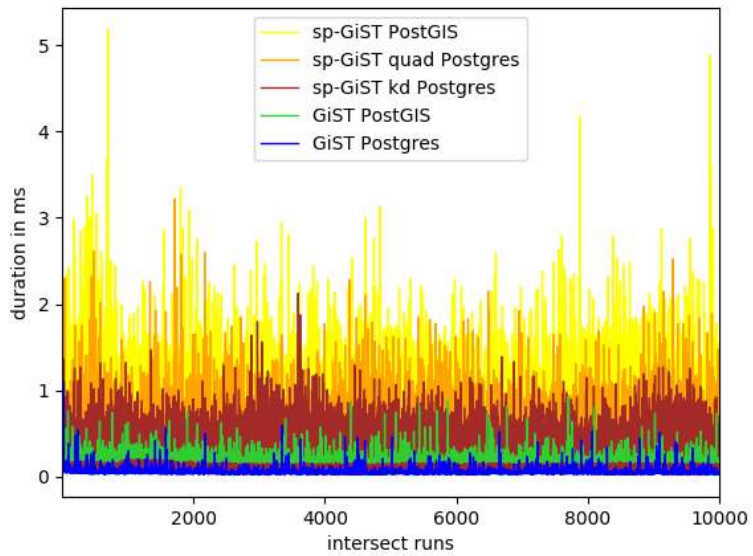
```
SELECT 1 FROM planet_osm_[point|line|polygon]
 WHERE CAST(way AS point) <@ [filter-box]
   AND _ST_Intersects(way, [filter-buffer]);
```

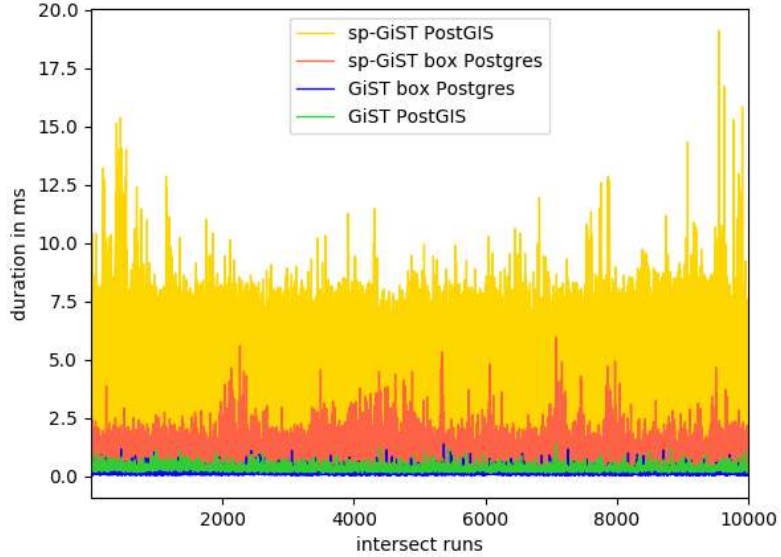


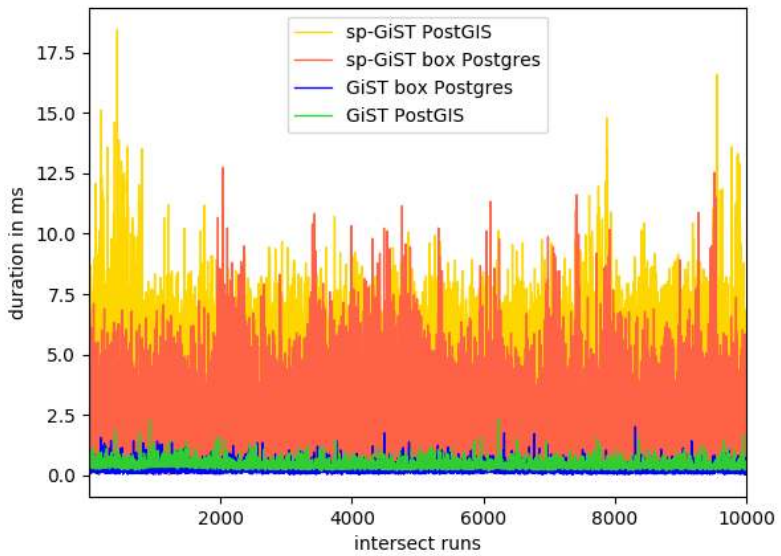




```
SELECT 1 FROM planet_osm_[point|line|polygon]
 WHERE CAST(way AS point) <@ [filter-box];
```







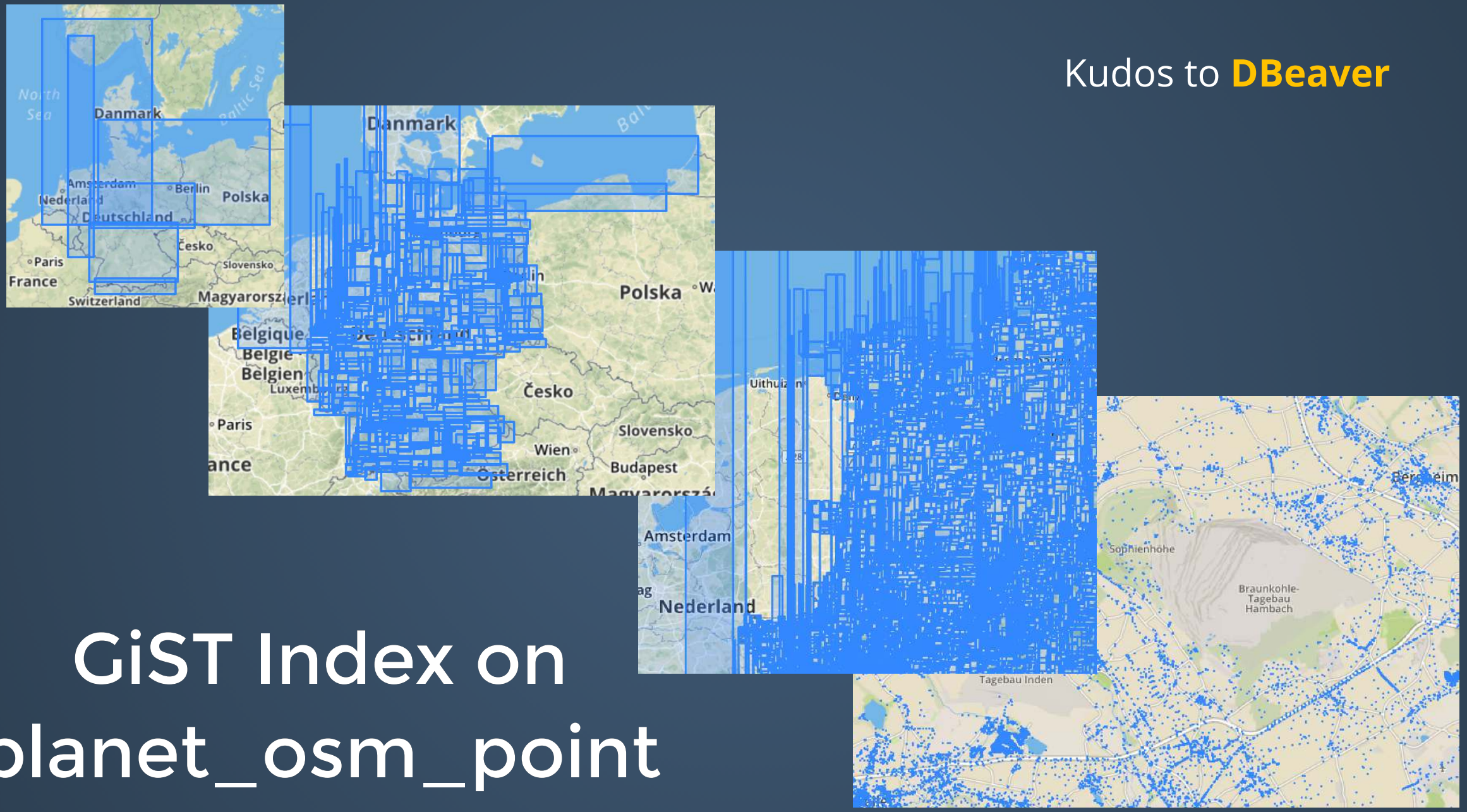
OSM Experiments

			
Points	0.5 Mio, 128 MB, **12%** 132, 53, 0.2 MB	1.7 Mio, 0.4 GB, **13%** 96, 92, 0.5 MB	11.7 Mio, 3 GB, **11%** 630, 600, 0.1 MB
Lines	0.5 Mio, 250 MB, **43%** 132, 53, 0.2 MB	2 Mio, 1 GB, **44%** 230, 93, 0.6 MB	14.37 Mio, 7 GB, **42%** 1500, 600, 0.2 MB
Polygons	1.2 Mio, 533 MB, **38%** 132, 53, 0.2 MB	5 Mio, 2.3 GB, **39%** 560, 220, 0.2 MB	35.6 Mio, 16 GB, **39%** 3800, 1500, 0,5 MB

Rowcount, Tablesize, Geom. size to table size

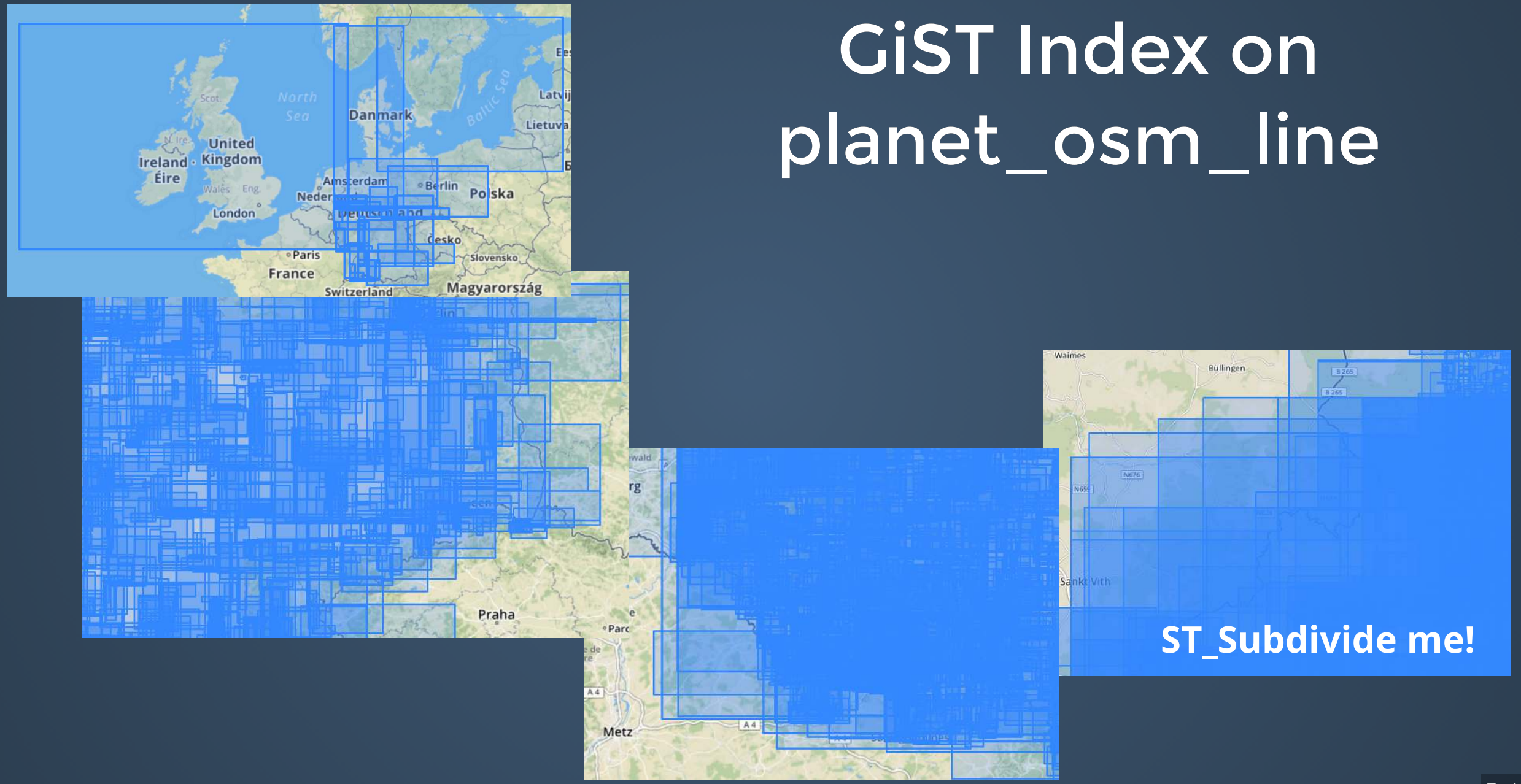
sizes of GiST, sp-GiST and BRIN

Kudos to **DBeaver**

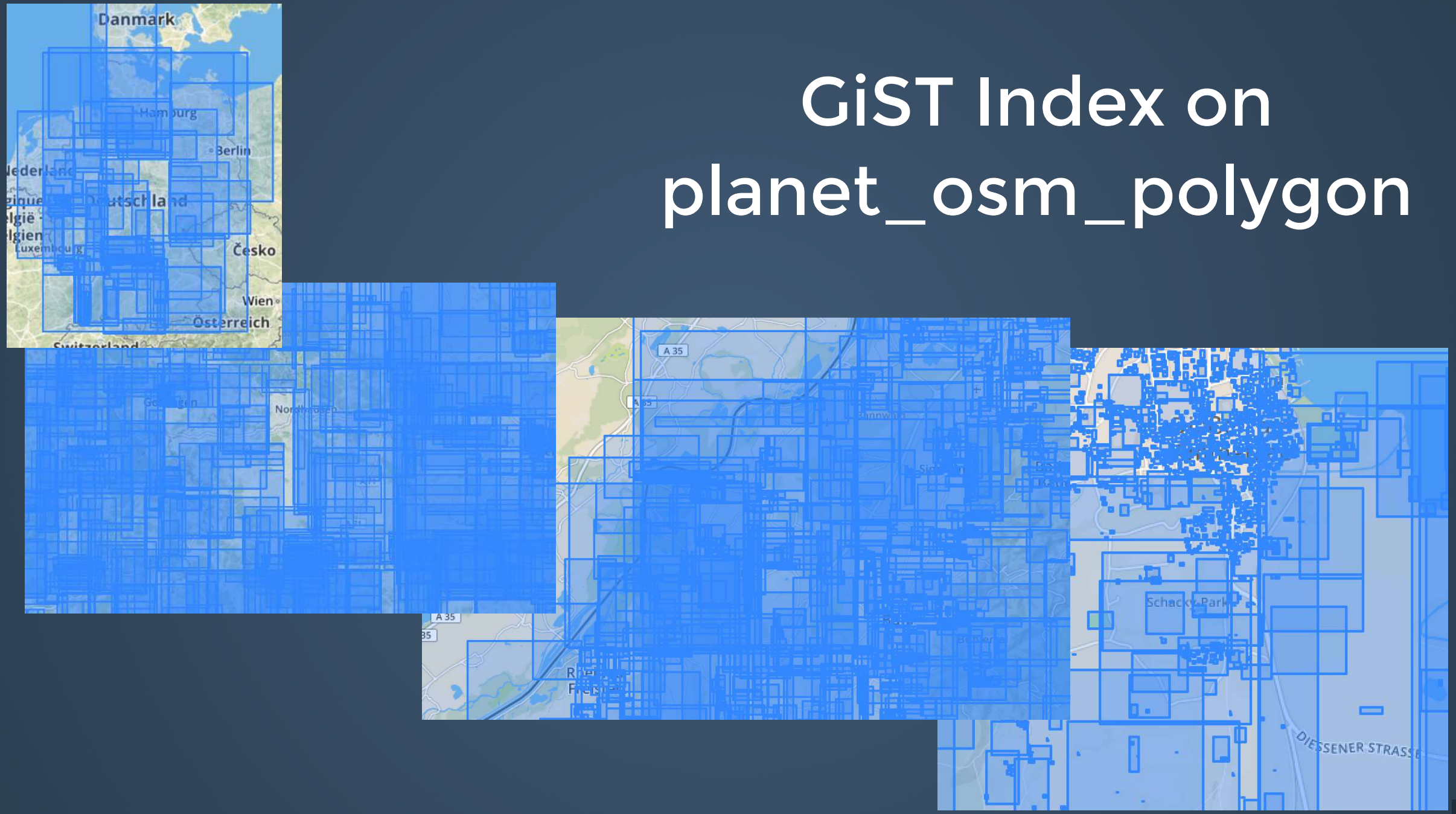


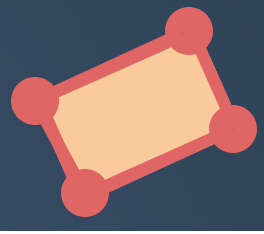
GiST Index on planet_osm_point

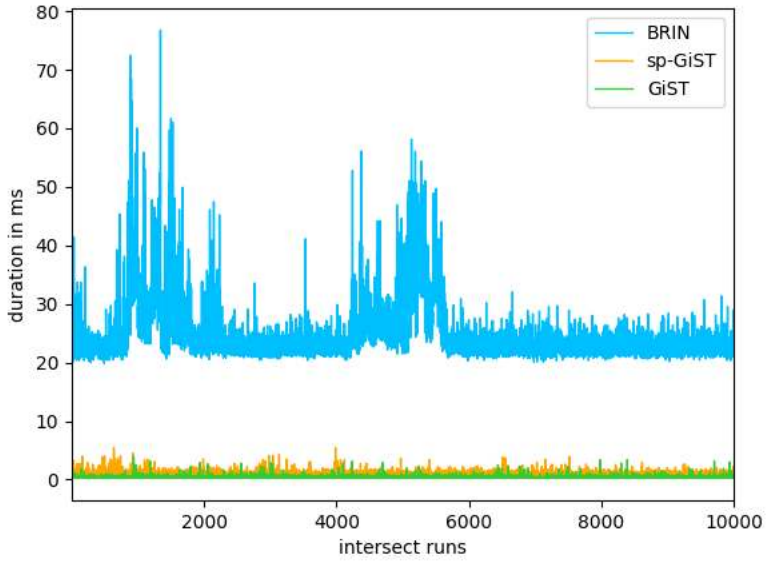
GiST Index on planet_osm_line

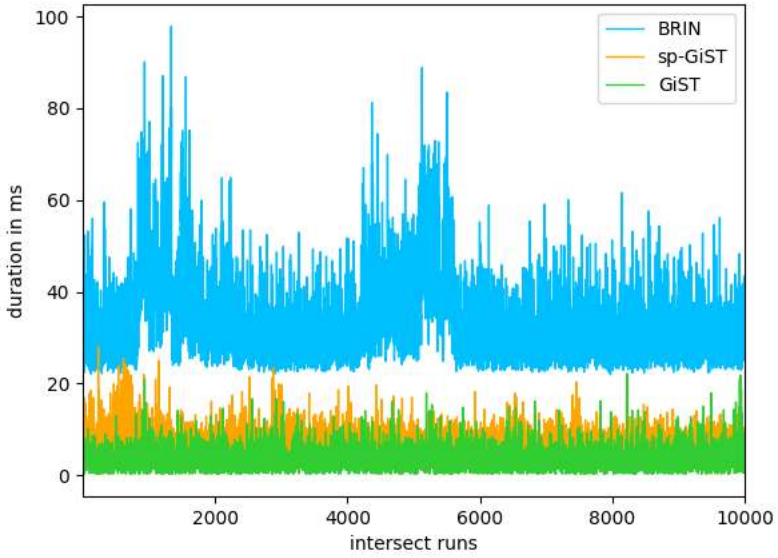


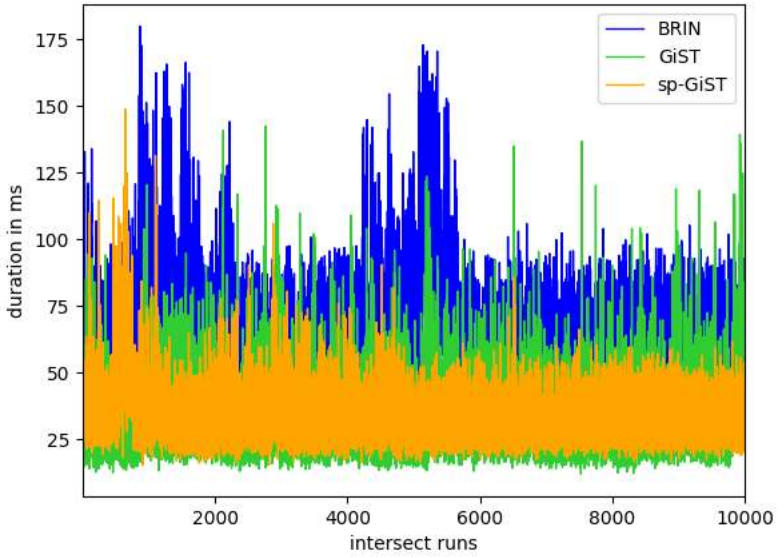
GiST Index on planet_osm_polygon







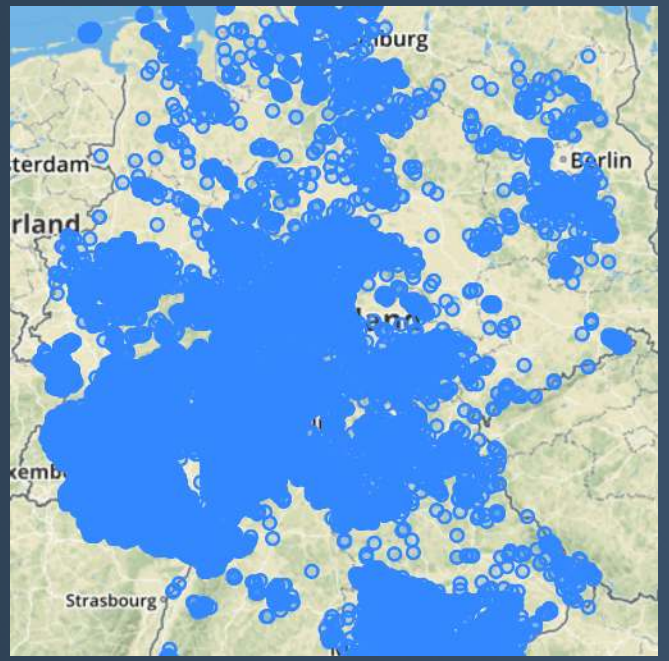




The more data, the closer the gap to BRIN

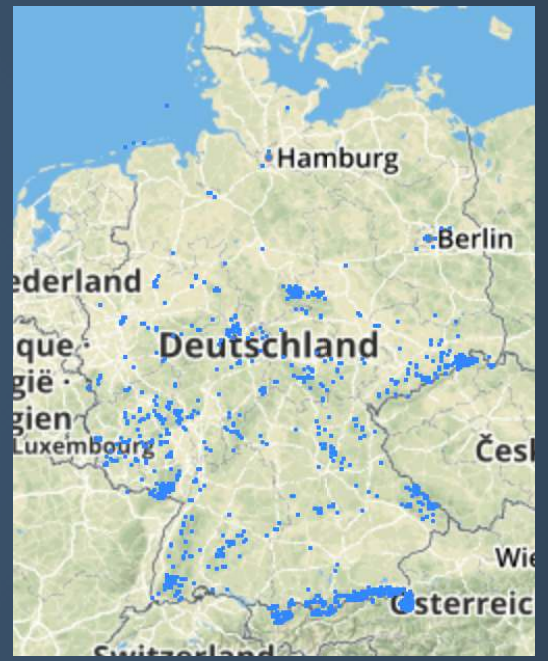
GiST and sp-GiST comparable on Polygons

highway:
emergency_access_point
~ 25.000



highway: motorway
~ 65.000

natural: bare_rock
~ 25.000



sp-GiST slightly faster for points and poylgons

Higher hit-rate of shared buffers (when is this bad?)

```
CREATE INDEX highway_eap_gist ON planet_osm_point USING GIST (way) WHERE highway = 'emergency_access_point';
CREATE INDEX highway_eap_spgist ON planet_osm_point USING SPGIST (way) WHERE highway = 'emergency_access_point';

EXPLAIN (ANALYZE, Buffers)
SELECT 1 FROM planet_osm_point, random_points
 WHERE ST_Intersects(way,ST_Buffer(geom,1000))
   AND highway = 'emergency_access_point';

GIST

Nested Loop  (cost=0.28..29261.90 rows=83000 width=4) (actual time=0.685..452.657 rows=817 loops=1)
  Buffers: shared hit=21756
  ->  Seq Scan on random_points  (cost=0.00..94.00 rows=10000 width=32) (actual time=0.011..1.977 rows=10000 loops=1)
        Buffers: shared hit=84
  ->  Index Scan using highway_eap_gist on planet_osm_point  (cost=0.28..2.92 rows=1 width=32) (actual time=0.018..0.019 rows=0 loops=10000)
        Index Cond: (way && st_buffer(random_points.geom, '1000'::double precision))
        Filter: _st_intersects(way, st_buffer(random_points.geom, '1000'::double precision))
        Rows Removed by Filter: 0
        Buffers: shared hit=21672
Planning Time: 0.201 ms
Execution Time: 452.791 ms

sp-GIST

Nested Loop  (cost=0.28..29274.00 rows=83000 width=4) (actual time=1.267..362.863 rows=817 loops=1)
  Buffers: shared hit=39917
  ->  Seq Scan on random_points  (cost=0.00..94.00 rows=10000 width=32) (actual time=0.019..1.690 rows=10000 loops=1)
        Buffers: shared hit=84
  ->  Index Scan using highway_eap_spgist on planet_osm_point  (cost=0.28..2.92 rows=1 width=32) (actual time=0.011..0.012 rows=0 loops=10000)
        Index Cond: (way && st_buffer(random_points.geom, '1000'::double precision))
        Filter: _st_intersects(way, st_buffer(random_points.geom, '1000'::double precision))
        Rows Removed by Filter: 0
        Buffers: shared hit=39833
Planning Time: 0.347 ms
Execution Time: 363.027 ms
```

```
CREATE INDEX motorway_gist ON planet_osm_line USING GIST (way) WHERE highway = 'motorway';
CREATE INDEX motorway_spgist ON planet_osm_line USING SPGIST (way) WHERE highway = 'motorway';

EXPLAIN (ANALYZE, Buffers)
SELECT 1 FROM planet_osm_line, random_points
 WHERE ST_Intersects(way,ST_Buffer(geom,1000))
   AND highway = 'motorway';

GIST

Nested Loop  (cost=0.28..93512.50 rows=229587 width=4) (actual time=3.490..457.310 rows=3023 loops=1)
  Buffers: shared hit=28119
  ->  Seq Scan on random_points  (cost=0.00..94.00 rows=10000 width=32) (actual time=0.007..1.564 rows=10000 loops=1)
        Buffers: shared hit=84
  ->  Index Scan using motorway_gist on planet_osm_line  (cost=0.28..9.34 rows=2 width=218) (actual time=0.016..0.024 rows=0 loops=10000)
        Index Cond: (way && st_buffer(random_points.geom, '1000'::double precision))
        Filter: _st_intersects(way, st_buffer(random_points.geom, '1000'::double precision))
        Rows Removed by Filter: 0
        Buffers: shared hit=28035
Planning Time: 0.278 ms
Execution Time: 457.651 ms

sp-GIST

Nested Loop  (cost=0.28..93497.10 rows=229587 width=4) (actual time=8.248..653.375 rows=3023 loops=1)
  Buffers: shared hit=294815
  ->  Seq Scan on random_points  (cost=0.00..94.00 rows=10000 width=32) (actual time=0.010..1.836 rows=10000 loops=1)
        Buffers: shared hit=84
  ->  Index Scan using motorway_spgist on planet_osm_line  (cost=0.28..9.34 rows=2 width=218) (actual time=0.032..0.041 rows=0 loops=10000)
        Index Cond: (way && st_buffer(random_points.geom, '1000'::double precision))
        Filter: _st_intersects(way, st_buffer(random_points.geom, '1000'::double precision))
        Rows Removed by Filter: 0
        Buffers: shared hit=294731
Planning Time: 0.345 ms
Execution Time: 653.781 ms
```

Conclusion & Outlook

Stick with GiST

But give BRIN a try

Use partial indexes

sp-GiST for unbalanced themes

Keep your table stats up-to-date

Parallel queries!

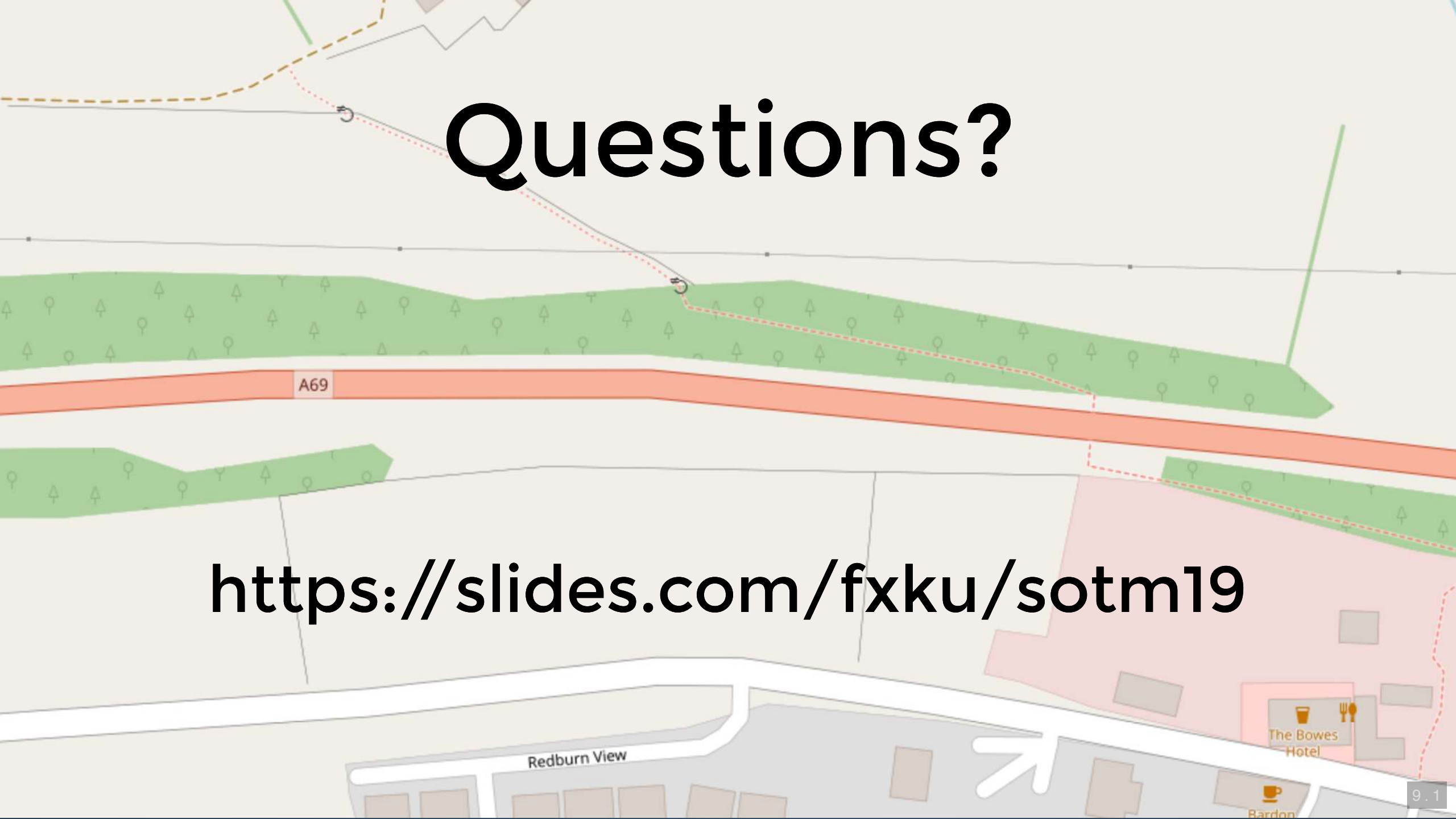
Faster index building!

Index-only scans?

New table access methods?

Questions?

https://slides.com/fxku/sotm19



Used hardware

Tuxedo Infinity Book 13
Intel i7-8550U CPU 1.80GHz
Quadcore, 8 CPUs
32 GB RAM
500GB SSD disk

Repo

github.com/FxKu/postgis_indexing

PostgreSQL config

PostgreSQL 11 & PostGIS 2.5
shared_buffers = 16 GB
work_mem = 128 MB
maintenance_work_mem = 4 GB
min/max_wal_level = 16/4 GB
checpoint_timeout = 30 min
checkpoint_completion_target = 0.9
random_page_cost = 1.1
cpu_tuple_cost = 0.001
cpu_index_tuple_cost = 0.001
effective_cache_size = 24 GB
default_statistics_target = 500

More links

Pettus 2019: Look it up - Practical PostgreSQL indexing [PDF]

Rogov 2019: Indexes in PostgreSQL - sp-GiST (Blog)

Katz 2018: Why covering indexes are helpful (Blog)

Albe 2019: Optimizer support for functions (Blog)

Wienand 2019: Be inclusive (Covering indexes) [PDF]