FOSS4G.NA 2016

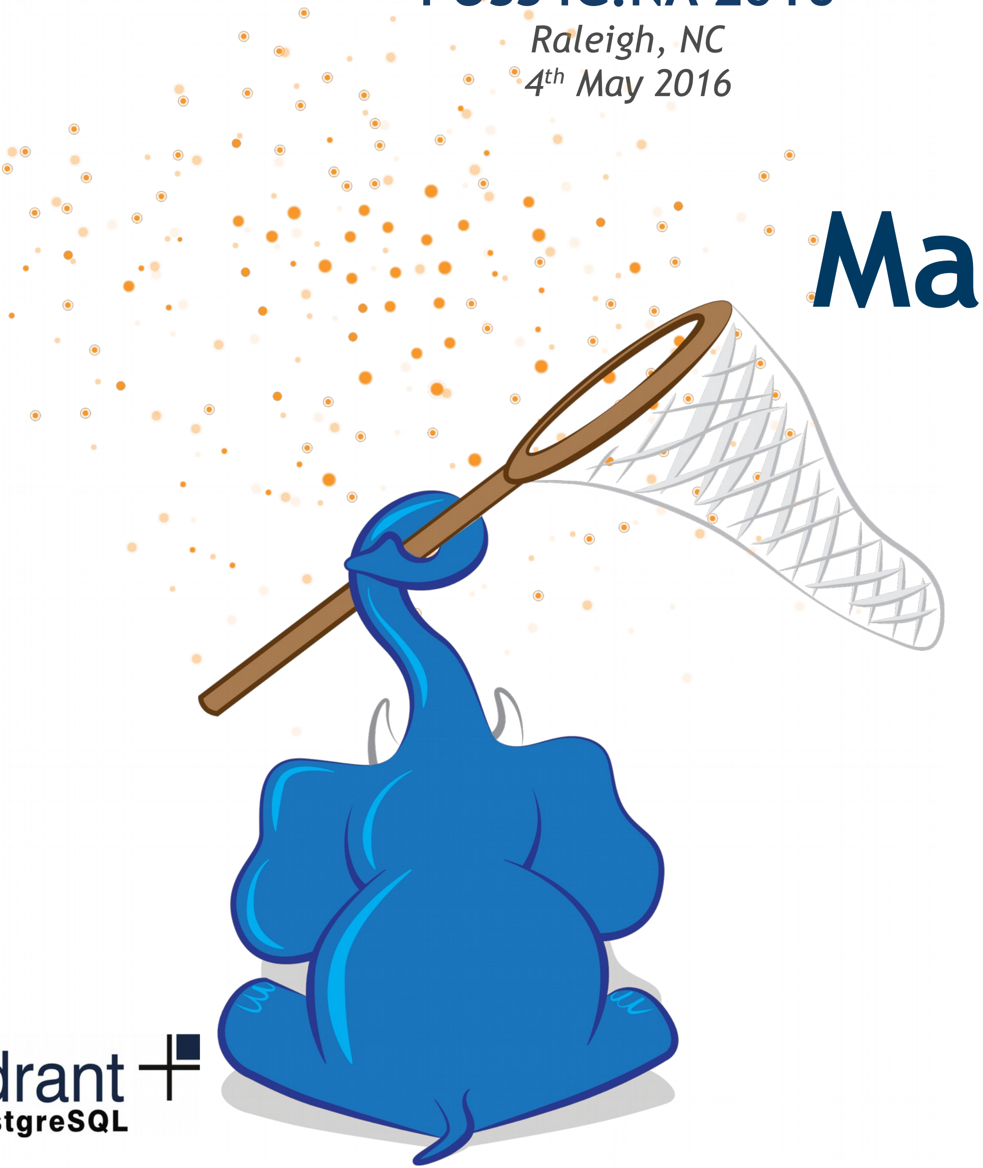
Raleigh, NC

4th May 2016

FOSS4G NORTH AMERICA 2016

Manage LiDAR data with PostgreSQL

is it possible?



2ndQuadrant

Professional PostgreSQL

www.2ndquadrant.com

2ndQuadrant Italia

Giuseppe Broccolo – giuseppe.broccolo@2ndquadrant.it

Is the relational approach valid for LiDAR?

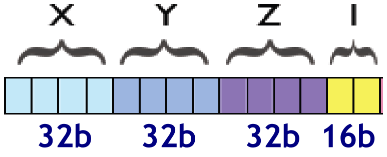
- The relational approach to the data:
  - data organized in ***tuples***
  - tuples are part of ***tables***
  - tables are related to each other through ***constraints*** (PK, FK, etc.)
- If the number of tuples grows:
  - ***indexes*** allow to reduce the complexity of a search to ~O(logN)...
  - ...but they must be contained in RAM!
  - OTHERWISE: the relational approach start to fail...

PostgreSQL & LiDAR data

- PostgreSQL is a relational DBMS with an extension for LiDAR data: **`pg_pointcloud`**

  `https://github.com/pgpointcloud/pointcloud`

- Part of the OpenGeo suite, completely compatible with PostGIS
  - two new datatype: **`pcpoint`**, **`pcpatch`** (compressed set of points)
    - N points (with all attributes from the survey) → 1 patch → 1 record



  - compatible with PDAL drivers to import data directly from **`.las`**

Relational approach to LiDAR data with PG

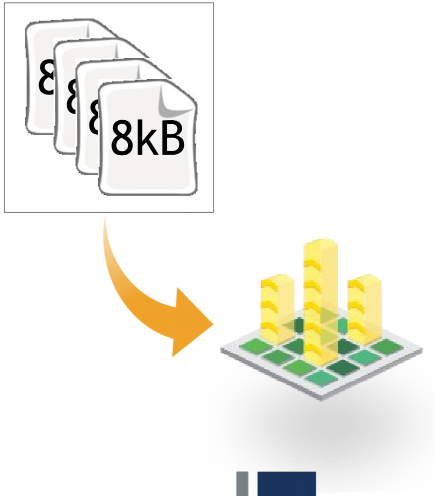
http://www.slideshare.net/GiuseppeBroccolo/gbroccolo-foss4-geugeodbindex

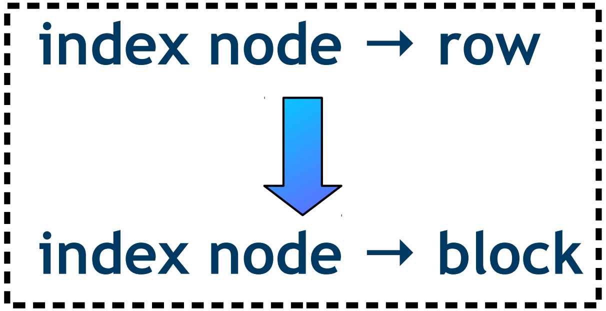
FOSS4G
EUROPE
COMO 2015

- GiST indexing in PostGIS
- GiST performances:
  - storage: 1TB RAID1, RAM 16GB, 8 CPU @3.3GHz, PostgreSQL9.3
  - index size ~O(table size)
  - Index was used:
    - up to ~300M points in bbox inclusion searches
    - up to ~10M points in kNN searches

LiDAR size: ~O($10^9 \div 10^{11}$) → few % can be properly indexed!

A new index in PG: Block Range INdexing

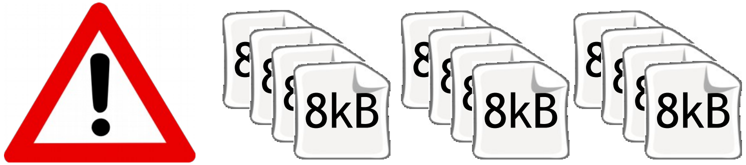




❗ less specific than GiST!

✓ Really small!

(S. Riggs, A. Herrera)

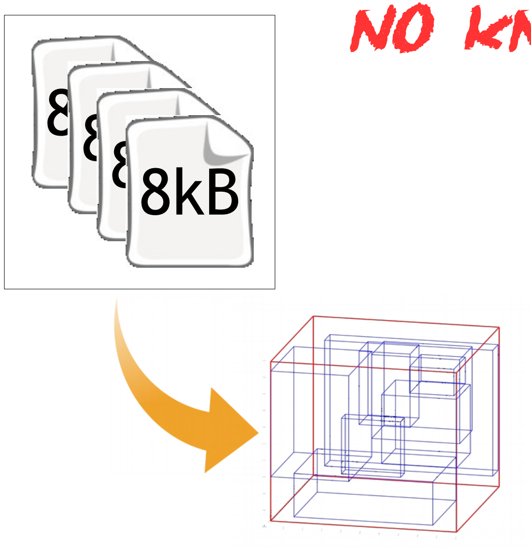


data must be physically sorted on disk!

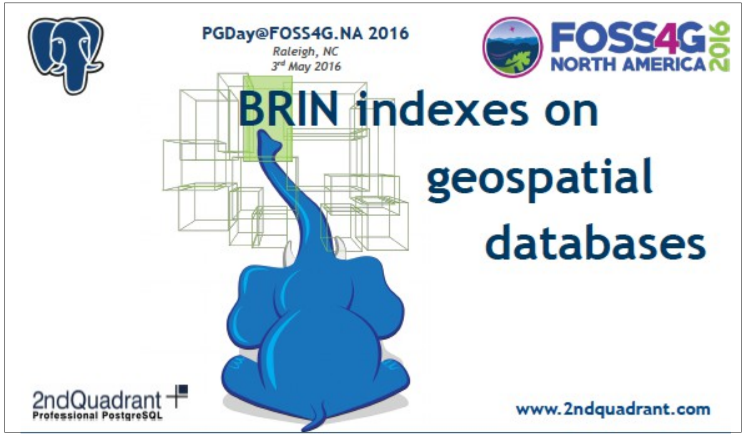
BRIN support for PostGIS datatypes

WARNING

NO KNN SUPPORT!!



3rd May, PGDay @ FOSS4G.NA


PGDay@FOSS4G.NA 2016
Raleigh, NC
3rd May 2016

FOSS4G NORTH AMERICA 2016

BRIN indexes on geospatial databases

2ndQuadrant
Professional PostgreSQL

www.2ndquadrant.com

G. Broccolo, J. Rouhaud, R. Dunklau

The LiDAR dataset: the ahn2 project

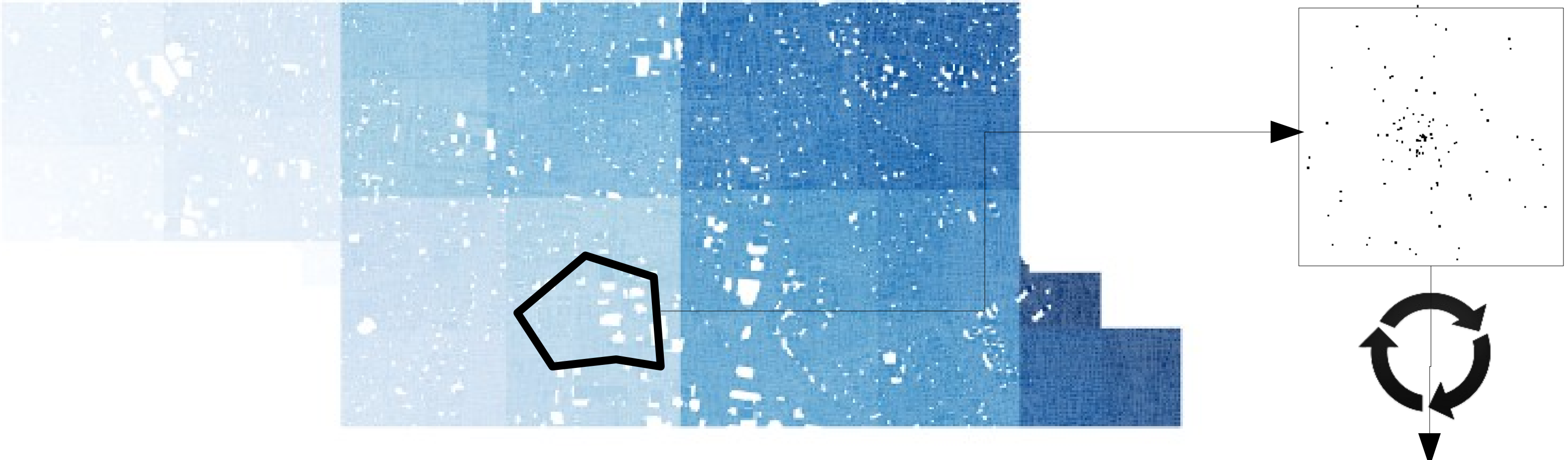
Geodan (thanks to Tom Van Tilburg)

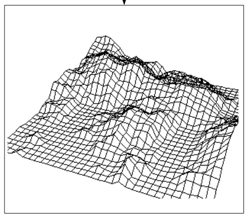
- 3D point cloud, coverage: almost the whole Netherlands
  - EPSG: 28992, ~8 points/m^2
- **1.6TB, ~250G points in ~560M patches (compression: ~10x)**
  - PDAL driver - **`filter.chipper`**
- available RAM: **16GB**
- the point structure:

X	Y	Z	scan	LAS	time	RGB	chipper
32b	32b	32b	40b	16b	64b	48b	32b

the “indexed” part (can be converted to PostGIS datatype)

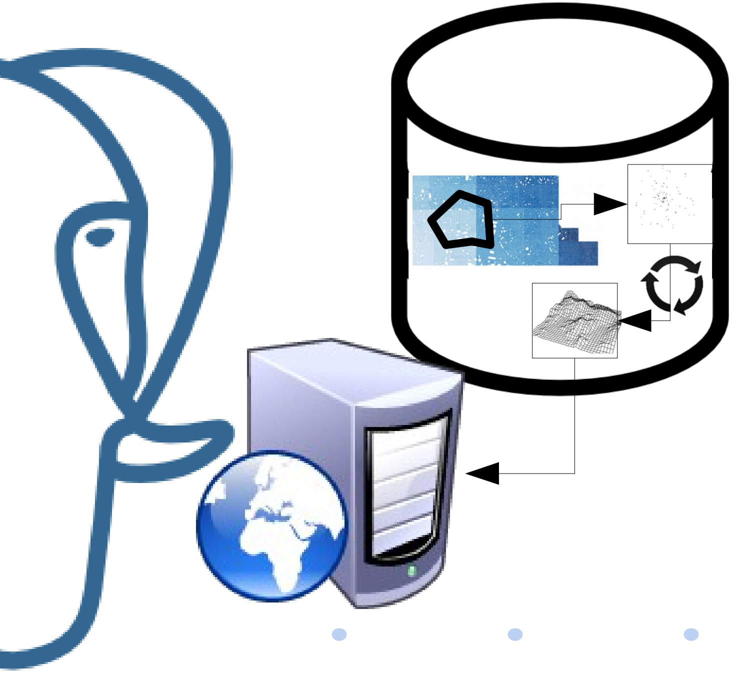
Typical searches on ahn2 - `x3d_viewer`





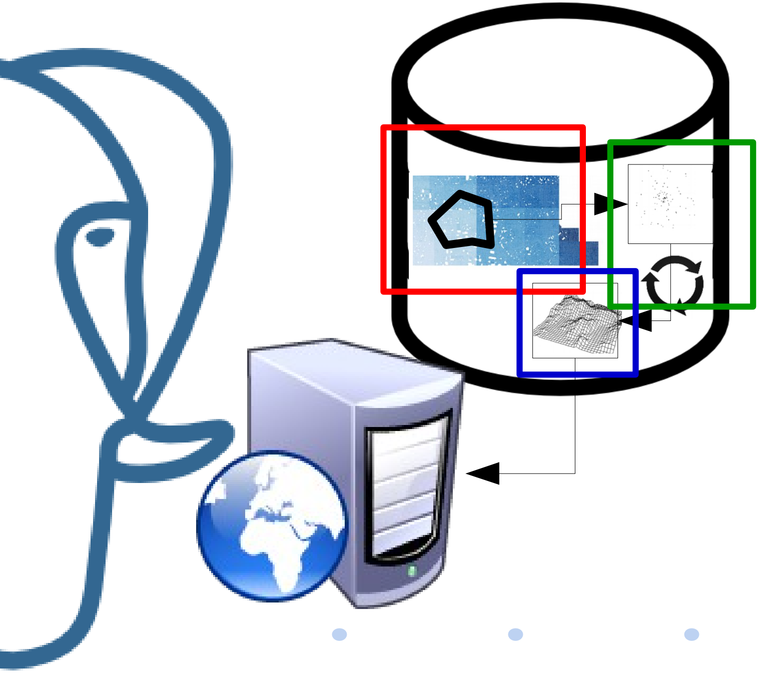
- Intersection with a polygon (*PostGIS*)
  - *the part that lasts longer - <u>need indexes here</u>!*
- Patch “explosion” + NN sorting (*pg_PointCloud+PostGIS*)
- *constrained* Delaunay Triangulation (*SFCGAL*)

All just in the DB...



```
WITH patches AS (
 SELECT patches FROM ahn2
 WHERE patches && ST_GeomFromText(‘POLYGON(...)’)
), points AS (
 SELECT ST_Explode(patches) AS points
 FROM patches
), sorted_points AS (
 SELECT points,
 ST_DumpPoints(ST_GeomFromText(‘POLYGON(...)’))).geom AS poly_pt
 FROM points ORDER BY points <#> poly_pt LIMIT 1;
), sel AS (
 SELECT points FROM sorted_points
 WHERE points && ST_GeomFromText(‘POLYGON(...)’)
)
SELECT ST_Dump(ST_Triangulate2DZ(ST_Collect(points))) FROM sel;
```

...and with just one query!



```
WITH patches AS (
 SELECT patches FROM ahn2
 WHERE patches && ST_GeomFromText(‘POLYGON(...)’)
), points AS (
 SELECT ST_Explode(patches) AS points
 FROM patches
), sorted_points AS (
 SELECT points,
 ST_DumpPoints(ST_GeomFromText(‘POLYGON(...)’))).geom AS poly_pt
 FROM points ORDER BY points <#> poly_pt LIMIT 1;
), sel AS (
 SELECT points FROM sorted_points
 WHERE points && ST_GeomFromText(‘POLYGON(...)’)
)
SELECT ST_Dump(ST_Triangulate2DZ(ST_Collect(points))) FROM sel;
```

patches && polygons - GiST performance

index building

GiST	2 d

index size

GiST	26GB

searches based on GiST

polygon size	timing
~O(m)	~40ms
~O(km)	~50s
~O(10km)	hours

index not contained in RAM anymore →

(~5G points → ~3%)

patches && polygons - BRIN performance

index building

BRIN	4 h

index size

BRIN	15MB

searches based on BRIN

polygon size	timing
~O(m)	~150s

patches && polygons - BRIN performance

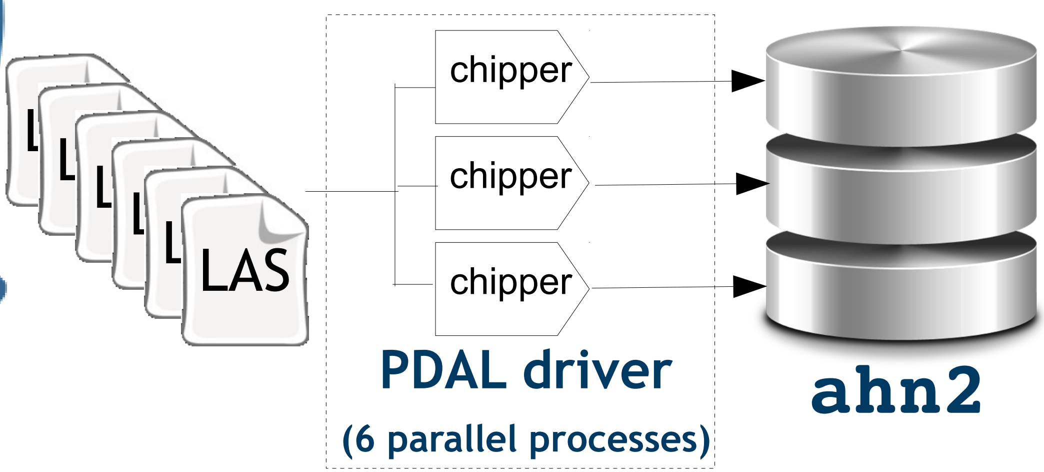
index building

BRIN	4 h

index size

BRIN	15MB

How data was inserted...



PDAL driver

(6 parallel processes)

ahn2

searches based on BRIN

polygon size	timing
~O(m)	~150s

patches && polygons - BRIN performance

```
CREATE INDEX patch_geohash ON ahn2_subset
USING btree (ST_GeoHash(ST_Transform(Geometry(patch), 4326), 20));

CLUSTER ahn2_subset USING patch_geohash;
```

(http://geohash.org/)

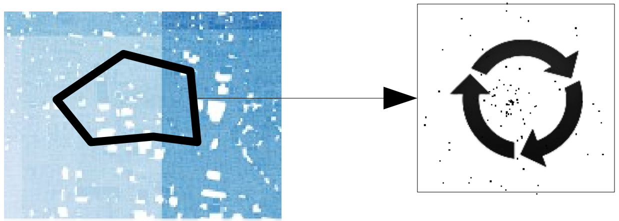
~150s → ~800ms [radius ~O(m)]

- x20 slower than GiST searches
- x200 faster than Seq searches
  - (x1000 faster in ~O(100m) searches)

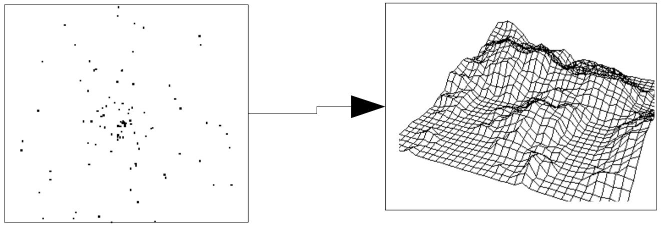
is the drop in performance acceptable?

-

  BRIN searches = x20 GiST searches

- 

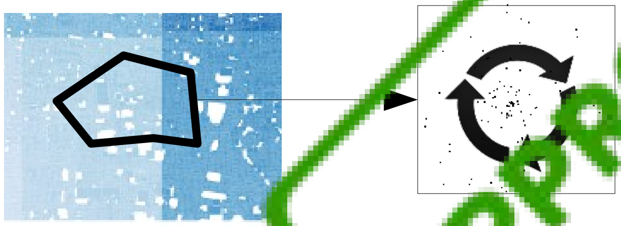
  patch explosion + NN sorting = x10÷x100 GiST searches

- 

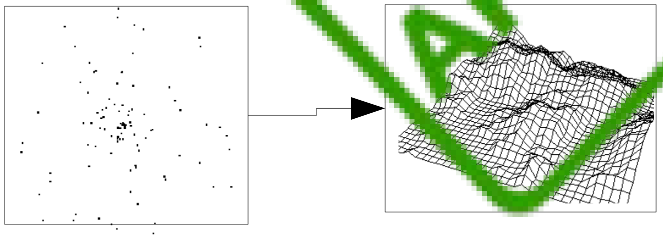
  constrained triangulation = x10÷x100 GiST searches

is the drop in performance acceptable?

- BRIN searches = x20 GiST searches



- patch explosion + NN sorting = x10÷x100 GiST searches



- constrained triangulation = x10÷x100 GiST searches

APPROVED

Conclusions

Can be the relational approach to LiDAR data valid with PostgreSQL?

- Yes!
  - GiST indexes are fast, but can manage just a real small portion of the dataset
  - BRINs are quite slower, but generally do not represent a real bottleneck
    - Make sure that data has the same sequentiality of **`.las`**

~$ whoami

2ndQuadrant Professional PostgreSQL

Giuseppe Broccolo, PhD

PostgreSQL & PostGIS consultant

@giubro in gbroccolo7 gbroccolo gemini__81

giuseppe.broccolo@2ndquadrant.it

Creative Commons license