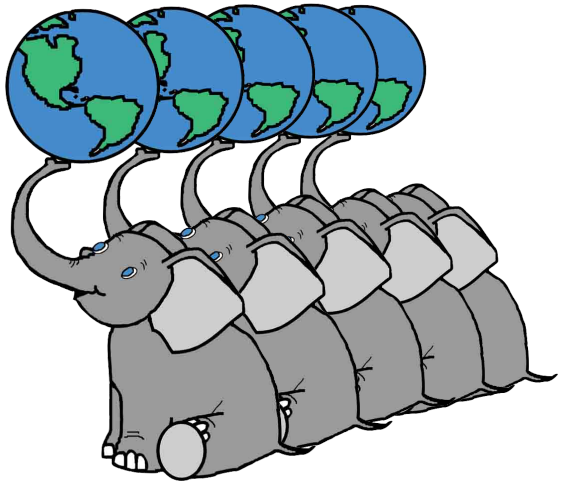
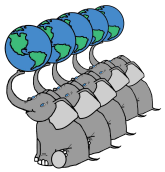


# Traitements spatiaux parallélisés pour les gros volumes de données

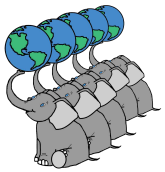
PostgreSQL Session #8, Lyon, 22 sept 2016

Nicolas Ribot – nicolas.ribot@gmail.com

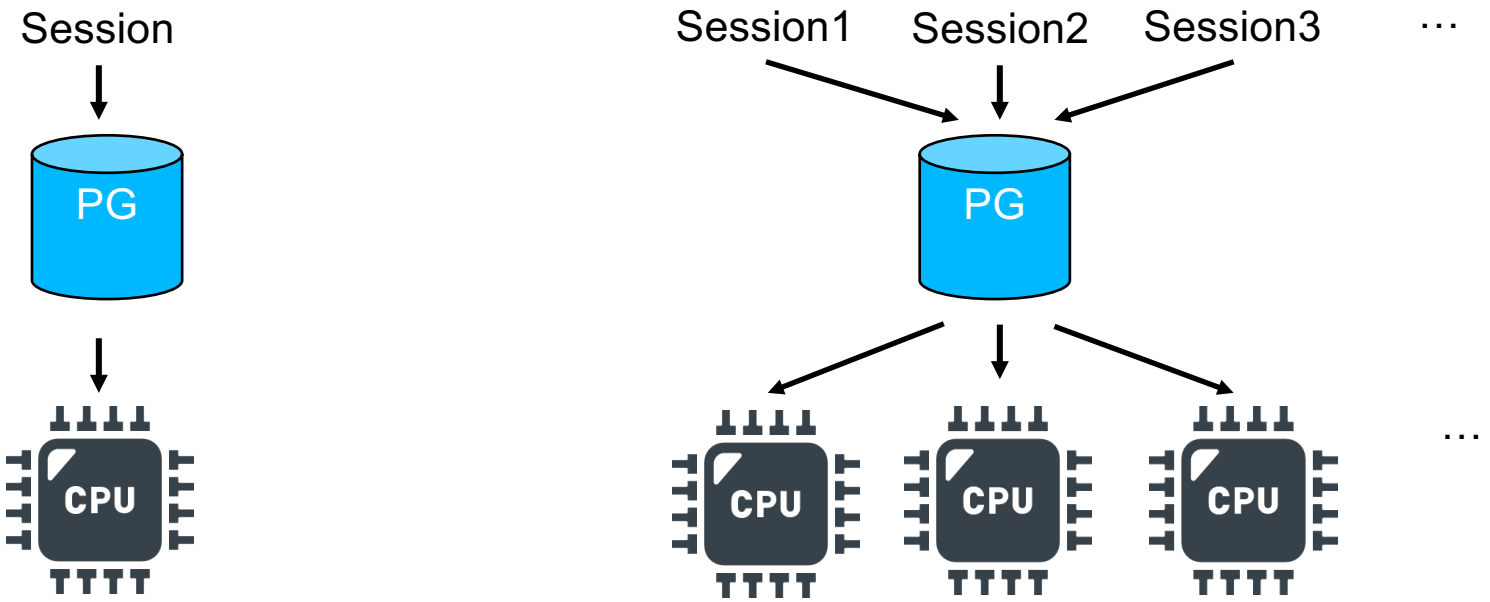


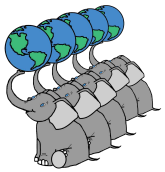
# Plan de la présentation

- Introduction
- Données de test
- Solutions testées
  - GNU Parallel: Fast Map Intersection/Par Psql
  - Postgresql Parallel Query
- Comparaisons



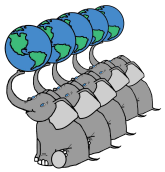
# Introduction



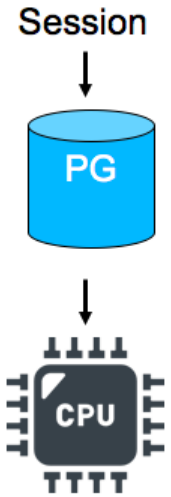


# Introduction

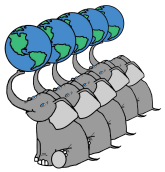
- PG: mono processeur
- Machines actuelles multi-processeur, multi-core
- Optimiser l'usage des CPU pour les traitements volumineux
- Map/reduce:
  - Découper la requête en parties traitées par des connexions différentes
  - Unir le résultat



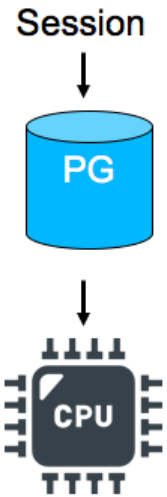
# Introduction

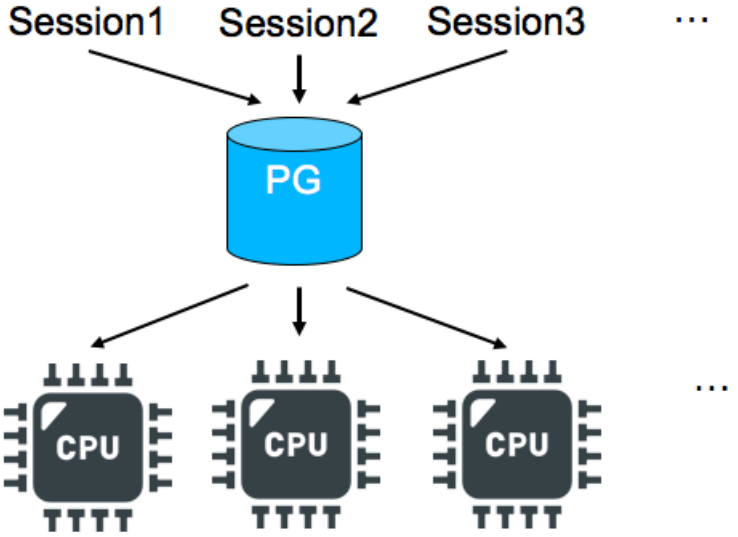



- PG: mono processeur
- Machines actuelles multi-processeur, multi-core
- Optimiser l'usage des CPU pour les traitements volumineux
- Map/reduce:
  - Découper la requête en parties traitées par des connexions différentes
  - Unir le résultat

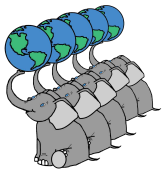


# Introduction



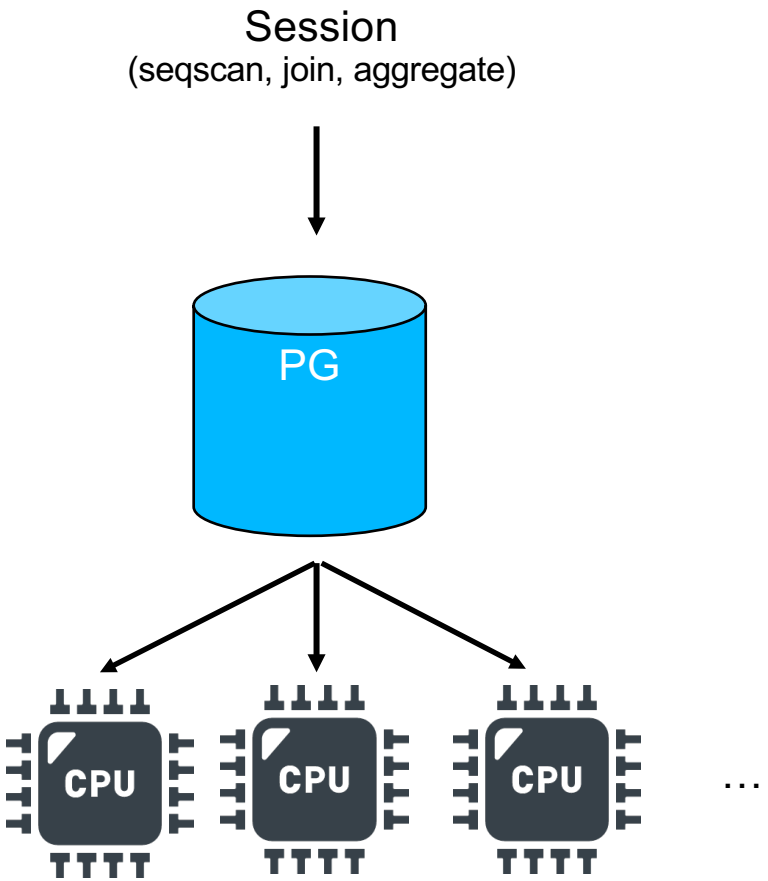



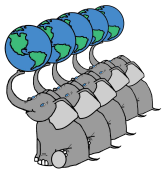

- PG: mono processeur
- Machines actuelles multi-processeur, multi-core
- Optimiser l'usage des CPU pour les traitements volumineux
- Map/reduce:
  - Découper la requête en parties traitées par des connexions différentes
  - Unir le résultat



# Introduction

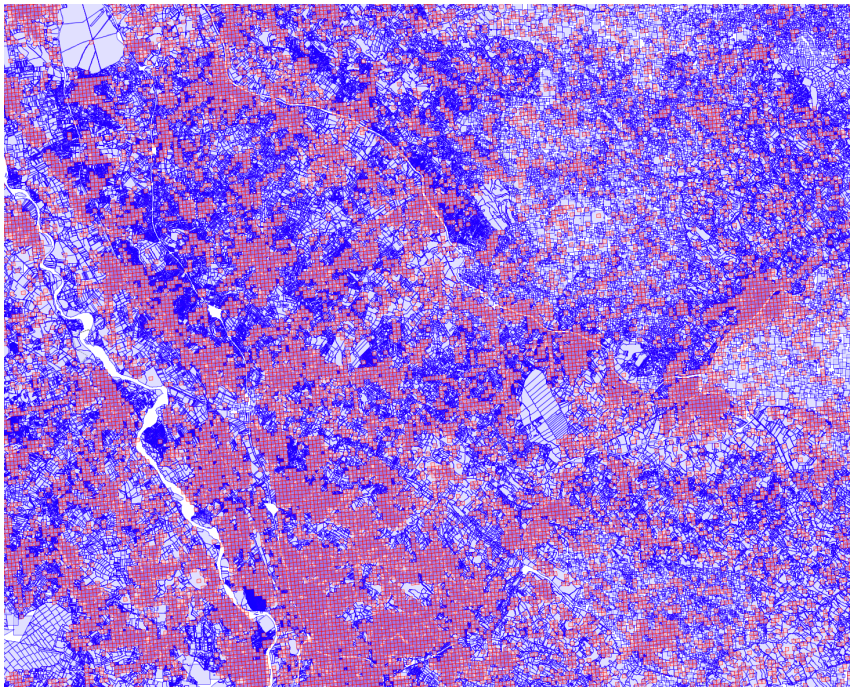
- PG 9.6: **pararell query mode** ☺

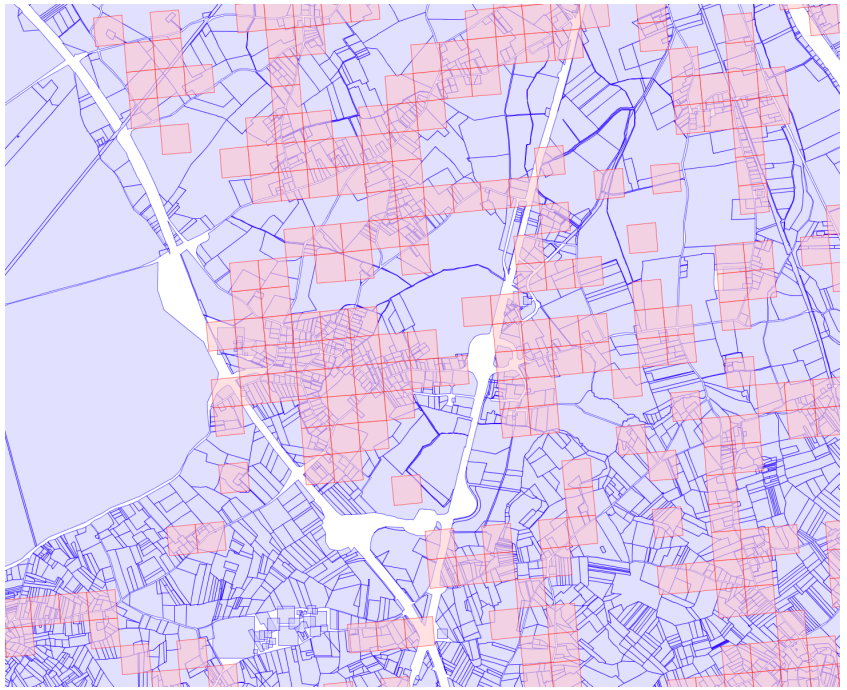


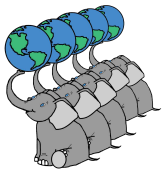


# Données de test

- Parcelles cadastrales sur plusieurs années:
  - 97M de MULTIPOLYGON, 1250M de coordonnées
  - 28 Go (40 Go avec index)
- Carroyage statistique INSEE sur la population:
  - 2.28M de POLYGON, 11.4M de coordonnées



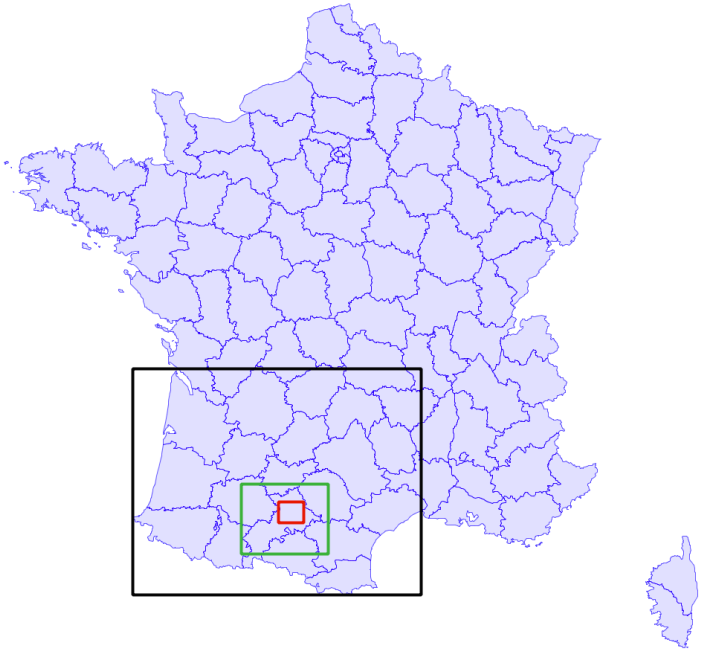


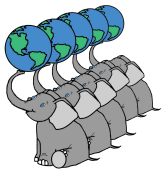


# Données de test

- Extraits spatiaux de différents volumes:
  - Sample0: 22 125 334 pg / 545 280 pg
  - Sample1: 2 892 689 pg / 87 376 pg
  - Sample2: 376 131 pg / 14 354 pg

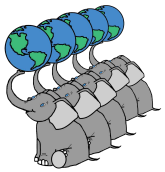
- OS X, 32 Go ram,
- Intel Core i7 3.5 Ghz (4 coeurs)
- Data, système: 2 SSD
- Shared_buffers: 3128Mo
- Work_mem: 20 Mo
- max_wall_size: 20 Go





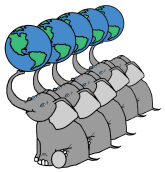
# Fast Map Intersection

- Projet GIT:https://github.com/gbb/fast_map_intersection
- Auteur: Graeme Bell
- But:
  - découper une intersection entre couches en plusieurs requêtes lancées en //
  - Union des résultats intermédiaires



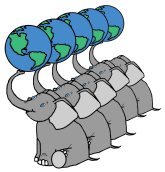
# Par Psql

- Projet GIT: https://github.com/gbb/par_psql
- Auteur: Graeme Bell
- But: ajouter un mot-clé permettant de lancer des requêtes en // depuis un script SQL
- Lance les requêtes commentées avec « `--&` » en //
- Synchronise quand « `--&` » n ’est plus présent



# Par Psql, FMI

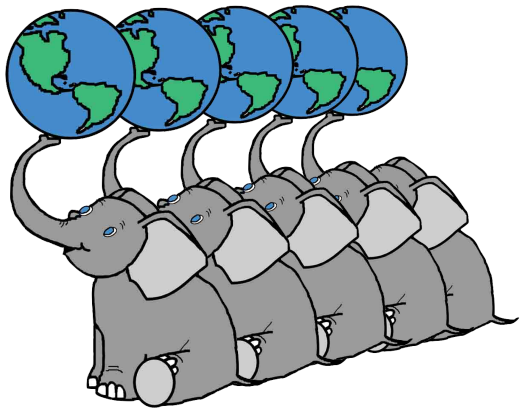
- Présentation FOSS4g Côme 2015
- http://graemebell.net/foss4gcomo.pdf

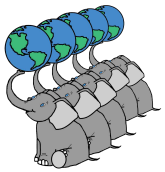


# PG parallel query

- PostgreSQL 9.6: Parallel Query !
  - Parallel seqscan, join, aggregate
- Nouveaux paramètres:
  - max_worker_processes
  - max_parallel_workers_per_gather
  - parallel_workers (table creation)
- Fonctions agrégées déclarées PARALLEL SAFE

- PostGIS 2.3beta1: support des requêtes //





# PG parallel query: Limitations

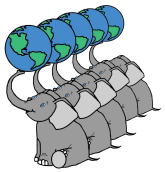
- Requêtes en écriture non supportées

```
create table as SELECT...
```

- Utilisation de fonction PARALLEL SAFE (UDF=UNSAFE)
  - Fonctions systèmes PARALLEL SAFE
- Requêtes imbriquées dans une requête parallélisée
- Agrégats spatiaux non supportés

```
Schema     | Name | Result data type | Argument data types | Type | Volatility | Parallel |
-----------+------+------------------+---------------------+------+------------+----------+
pg_catalog | sum  | numeric          | bigint              | agg  | immutable  | safe     |
pg_catalog | sum  | double precision | double precision    | agg  | immutable  | safe     |
```

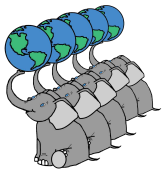
```
Schema     |   Name    | Result data type | Argument data types | Type | Volatility | Parallel |
-----------+-----------+------------------+---------------------+------+------------+----------+
public     | st_union  | geometry         | geometry            | agg  | immutable  | unsafe   |
public     | st_accum  | geometry[]       | geometry            | agg  | immutable  | unsafe   |
public     | st_extent | box2d            | geometry            | agg  | immutable  | unsafe   |
```



# PG parallel query: create table

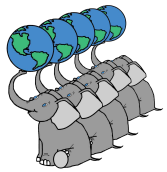
- Fonction PL/PGSQL permettant un **`create table`** // à partir d'une requête
- Utilise COPY et psql
- COPY la table depuis la requête lancée par psql:

```
copy tc FROM PROGRAM $$psql -A -t \
  -c "select p.id, c.gid,
        st_intersection(p.geom, c.geom) as geom
      from parcelle p
      join carreau c on st_intersects(p.geom, c.geom)"$$
with (DELIMITER '|');
```



# PG parallel query: create table

```
create or replace function create_table_parallel(
  table_name text,
  query text,
  program text DEFAULT 'psql -A -t -c',
  num_workers int DEFAULT 0,
  drop_table boolean DEFAULT false) returns text as $$
DECLARE
  v_set_workers text :=format('set max_parallel_workers_per_gather = %s;',
   num_workers);
BEGIN
  if drop_table then
    execute format('drop table if exists %I', table_name);
  END IF;
  execute format('create table %I as %s LIMIT 0', table_name, query);
  execute format('copy %I FROM PROGRAM ''%s "%s %s"'' with (DELIMITER ''|'')',
   table_name, program, v_set_workers, query);
  return format('%I created', table_name);
END;
$$ LANGUAGE plpgsql parallel safe;
```

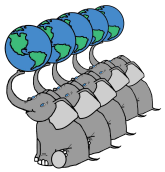


# PG parallel query: create table

- Utilisation:

```
select * from create_table_parallel(
    'inter_pgpar1', -- new table name
    'select p.id, c.gid,
        clock_timestamp() AS creation_time,
        st_intersection(p.geom, c.geom) as geom
      from parcelle p
      join carreau c on st_intersects(p.geom, c.geom)',
    '/usr/local/pgsql-9.6/bin/psql -A -t -p 5439 -d nicolas -c',
    8, -- number of workers
    true - to drop table first.
);
```

- Avantage: création de tables avec plan // dans un script SQL



# PG parallel query: exemples

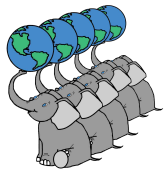
- // agregate

```
explain select sum(st_npoints(geom)), min(st_npoints(geom)),
  max(st_npoints(geom)), avg(st_npoints(geom)) from parcelle;


set max_parallel_workers_per_gather = 0;
```

```
Aggregate  (cost=15492093.56..15492093.57 rows=1 width=48)
     ->  Seq Scan on parcelle  (cost=0.00..4694615.88 rows=98158888 width=253)
```

```
set max_parallel_workers_per_gather = 6;
```

```
Finalize Aggregate  (cost=5677205.47..5677205.48 rows=1 width=48)
    ->  Gather  (cost=5677204.80..5677205.41 rows=6 width=48)
          Workers Planned: 6
          ->  Partial Aggregate  (cost=5676204.80..5676204.81 rows=1 width=48)
                ->  Parallel Seq Scan on parcelle  (cost=0.00..3876625.15
  rows=16359815 width=253)
```



# PG parallel query: exemples

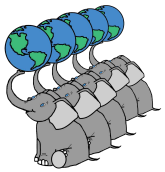
- // spatial JOIN

```
explain select p.id, c.gid
from parcelle p join carreau c on st_intersects(p.geom, c.geom);

set max_parallel_workers_per_gather = 0;
```

```
Nested Loop  (cost=0.41..21389689299.18 rows=74542277448 width=12)  ->  Seq Scan on
   parcelle p  (cost=0.00..4694615.88 rows=98158888 width=261)  ->  Index Scan
   using carreau_geom_idx on carreau c  (cost=0.41..217.10 rows=76 width=36)
   Index Cond: (p.geom && geom)        Filter: _st_intersects(p.geom, geom)
```

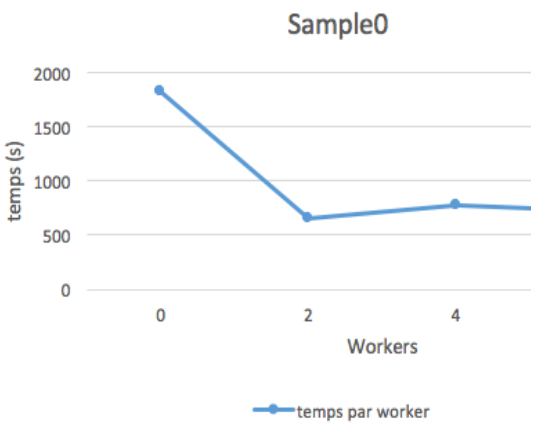
```
set max_parallel_workers_per_gather = 6;
```

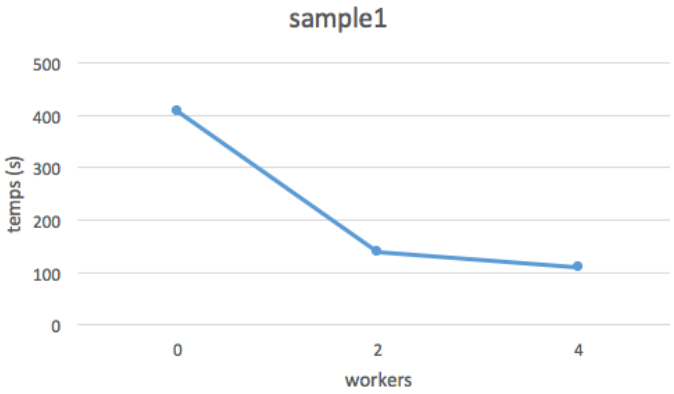
```
Gather  (cost=1000.41..022271223.12 rows=74542277448 width=12)  Workers Planned: 6
   ->  Nested Loop  (cost=0.41..3568042478.32 rows=74542277448 width=12)        ->
   Parallel Seq Scan on parcelle p  (cost=0.00..3876625.15 rows=16359815 width=261)
   ->  Index Scan using carreau_geom_idx on carreau c  (cost=0.41..217.10 rows=76
   width=36)              Index Cond: (p.geom && geom)              Filter:
   _st_intersects(p.geom, geom)
```

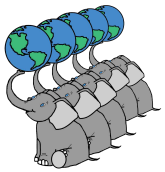


# PG parallel query: exemples

- Gain de performance
  - dataset (97M x 2.2M): **4220s vs 15122s (x 3.6)**
  - Sample0 (22M x 545k): **740s vs 2403s (x 3.7)**
  - Sample1 (2.9M x 87k) : **105s vs 406s (x 3.9)**
  - Sample2 (380k x 14k) : **22s vs 82s (x 3.8)**



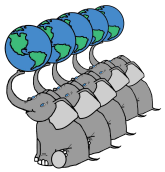



# PG parallel query: exemples

- Sample0: 0-2 workers: changement de plan de requête
  - Table carreau_sample0 plus petite que parcelle

```
Workers Launched: 4
   ->  Nested Loop
         ->  Parallel Seq Scan on parcelle_sample0 p
         ->  Index Scan using carreau_sample0_geom_gist on
carreau_sample0 c


Workers Launched: 2
   ->  Nested Loop
         ->  Parallel Seq Scan on carreau_sample0 c
         ->  Index Scan using parcelle_sample0_geom_gist on
parcelle_sample0 p c
```

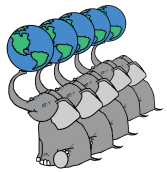
# Comparaison des solutions



# Comparaison des solutions

- Création d'une couche d'intersection entre les deux tables test

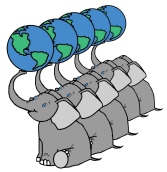
```
select p.id as idparc, c.gid as idcarreau, p.annee,
  clock_timestamp() AS creation_time,
  st_intersection(p.geom, c.geom) as geom
from parcelle p join carreau c
on st_intersects(p.geom, c.geom);
```



# Comparaison des solutions

- FMI: script shell à éditer:

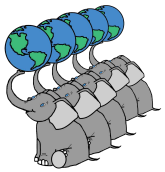
```
WORK_MEM=200
SPLIT=6
JOBS=8
...
CREATE UNLOGGED TABLE ${4}_${1}_${2} AS
select p.id as idparc, c.gid as idcarreau, p.annee,
    clock_timestamp() AS creation_time,
    st_intersection(p.geom, c.geom) as geom
from parcelle p join carreau c
on st_intersects(p.geom, c.geom)
  WHERE p.id%$3=${1}
  AND  c.gid%$3=${2};
...
```



# Comparaison des solutions

- Par psql: script SQL avec commentaires spéciaux:

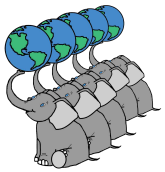
```
CREATE UNLOGGED TABLE inter_pp_1 as
  select p.id as idparc, c.gid as idcarreau, p.annee,
     st_intersection(p.geom, c.geom) as geom, clock_timestamp() as creation_time
  from parcelle p join carreau c on st_intersects(p.geom, c.geom)
  where p.id%8=0; --&
CREATE UNLOGGED TABLE inter_pp_2 as
  select p.id as idparc, c.gid as idcarreau, p.annee,
  st_intersection(p.geom, c.geom) as geom,clock_timestamp() as creation_time
  from parcelle_sample p join carreau_sample c on st_intersects(p.geom, c.geom)
  where p.id%8=1; --&

...
--partie synchronisée
CREATE TABLE inter_pp as
  select * from inter_pp_1
  UNION ALL
  select *from inter_pp_2
...
```



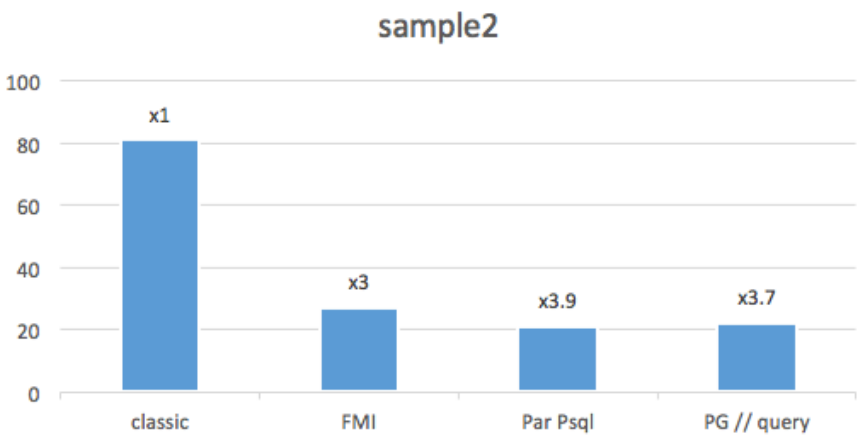
# Comparaison des solutions

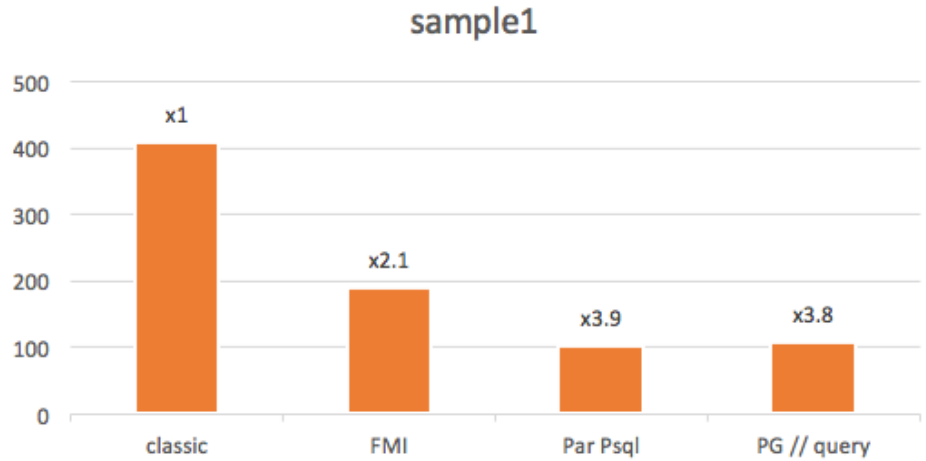
- PG parallel query: fonction avec requête classique:

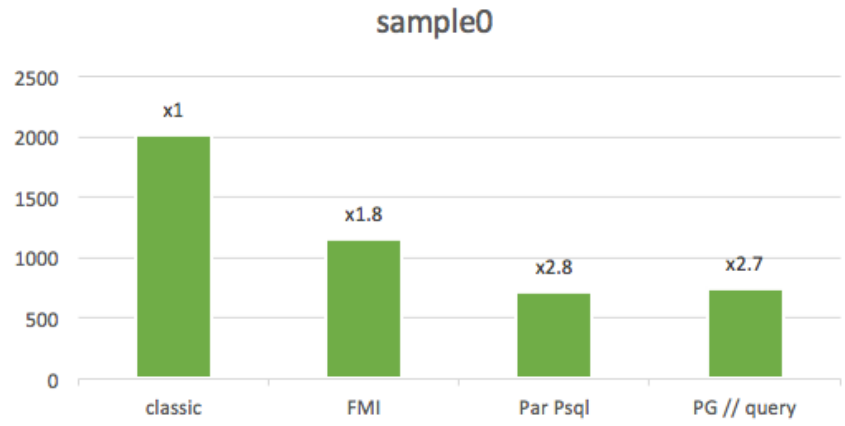
```
select * from create_table_parallel(
    'inter_pgpar1', -- new table name
    'select p.id, c.gid,
        clock_timestamp() AS creation_time,
        st_intersection(p.geom, c.geom) as geom
      from parcelle p
      join carreau c on st_intersects(p.geom, c.geom)',
    '/usr/local/pgsql-9.6/bin/psql -A -t -p 5439 -d nicolas -c',
    8, -- number of workers
    true - to drop table first.
);
```

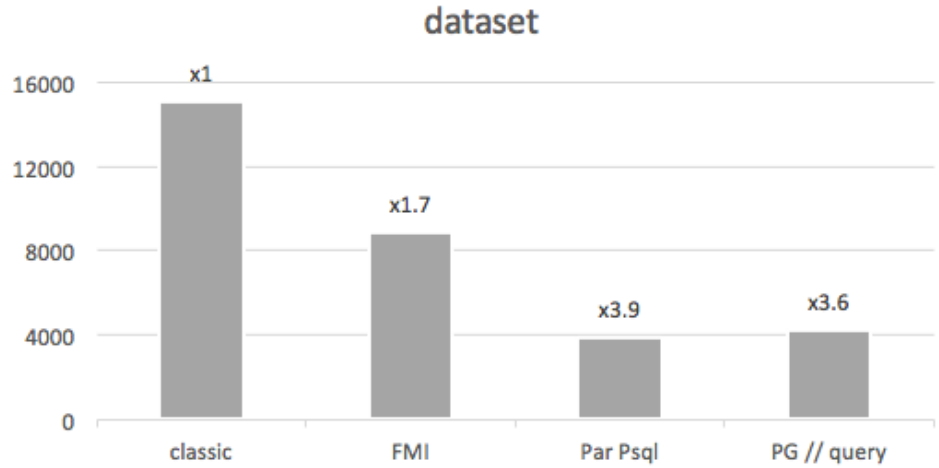


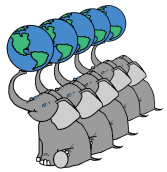
# Comparaison des solutions



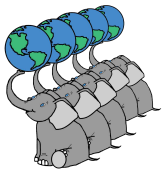





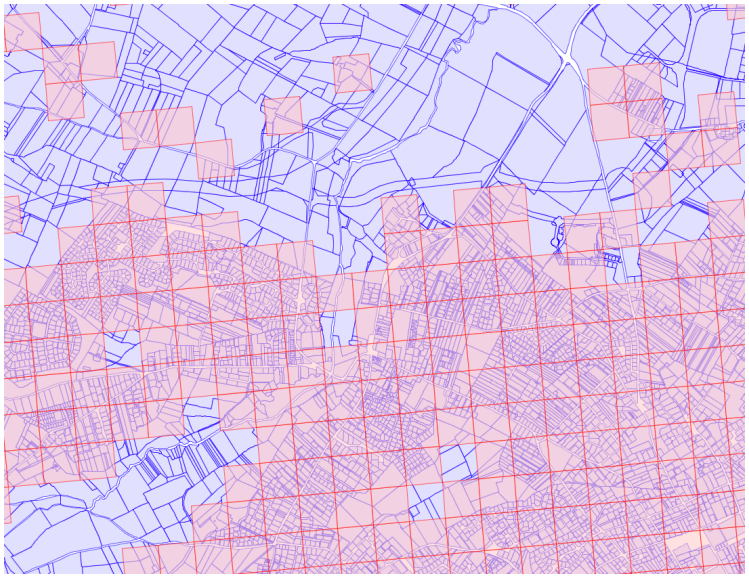
# Comparaison des solutions

- Fast Map Intersection
  - ☹ Limité au croisement de 2 couches spatiales
  - ☹ Script shell à éditer
  - ☺ Découpage automatique des tables
- Par Psql
  - ☹ Découpage manuel des tables
  - ☺ Parallélisation de tout type de requêtes
  - ☺ Utilisation directe en script SQL
- PostgreSQL // query
  - ☹ Ecriture // directe impossible
  - ☺ Parallélisation de plans seqscan, join, aggregate
  - ☺ Pas de découpage de la requête

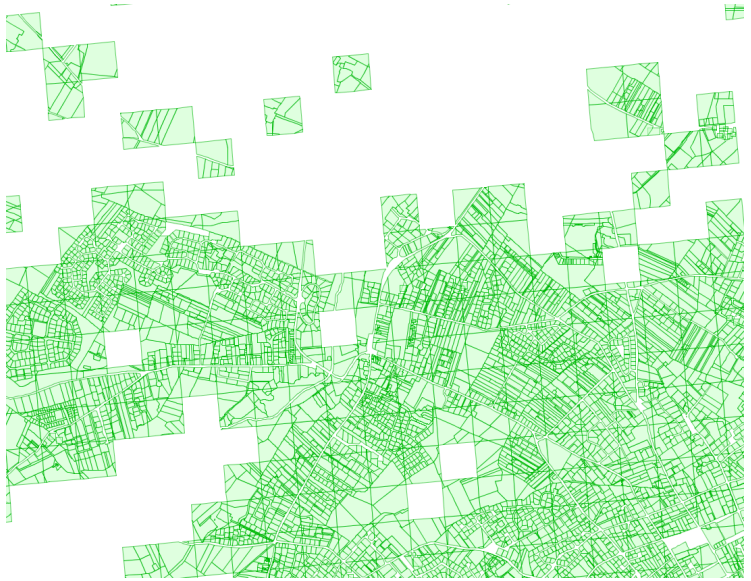


# Comparaison des solutions

Solution efficace pour les traitements volumineux



=>



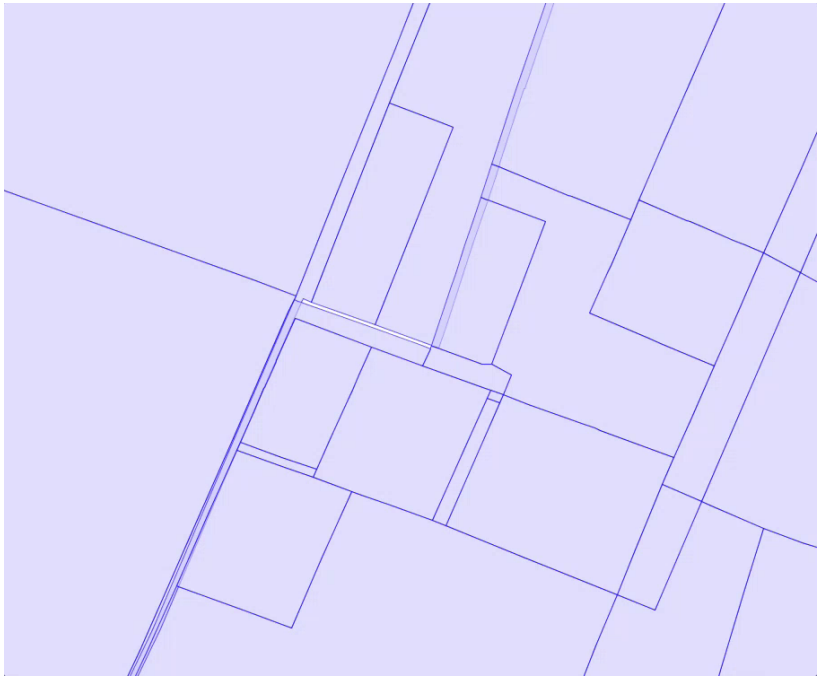
**PG //** pour les requêtes parallelisables

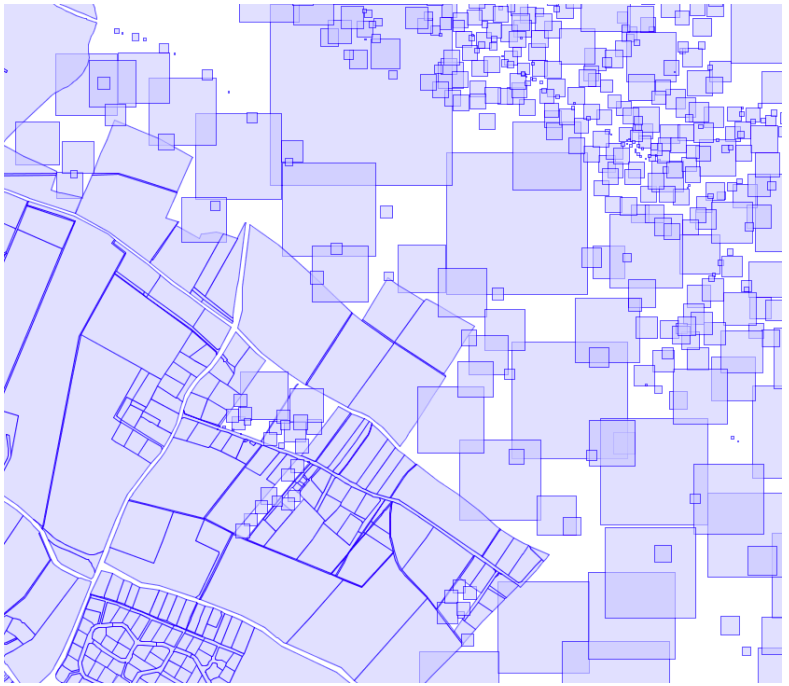
+

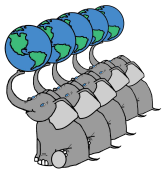
script **par_pgsql** pour paralléliser les parties non parellisables par PG

# PG Parallel: process complet

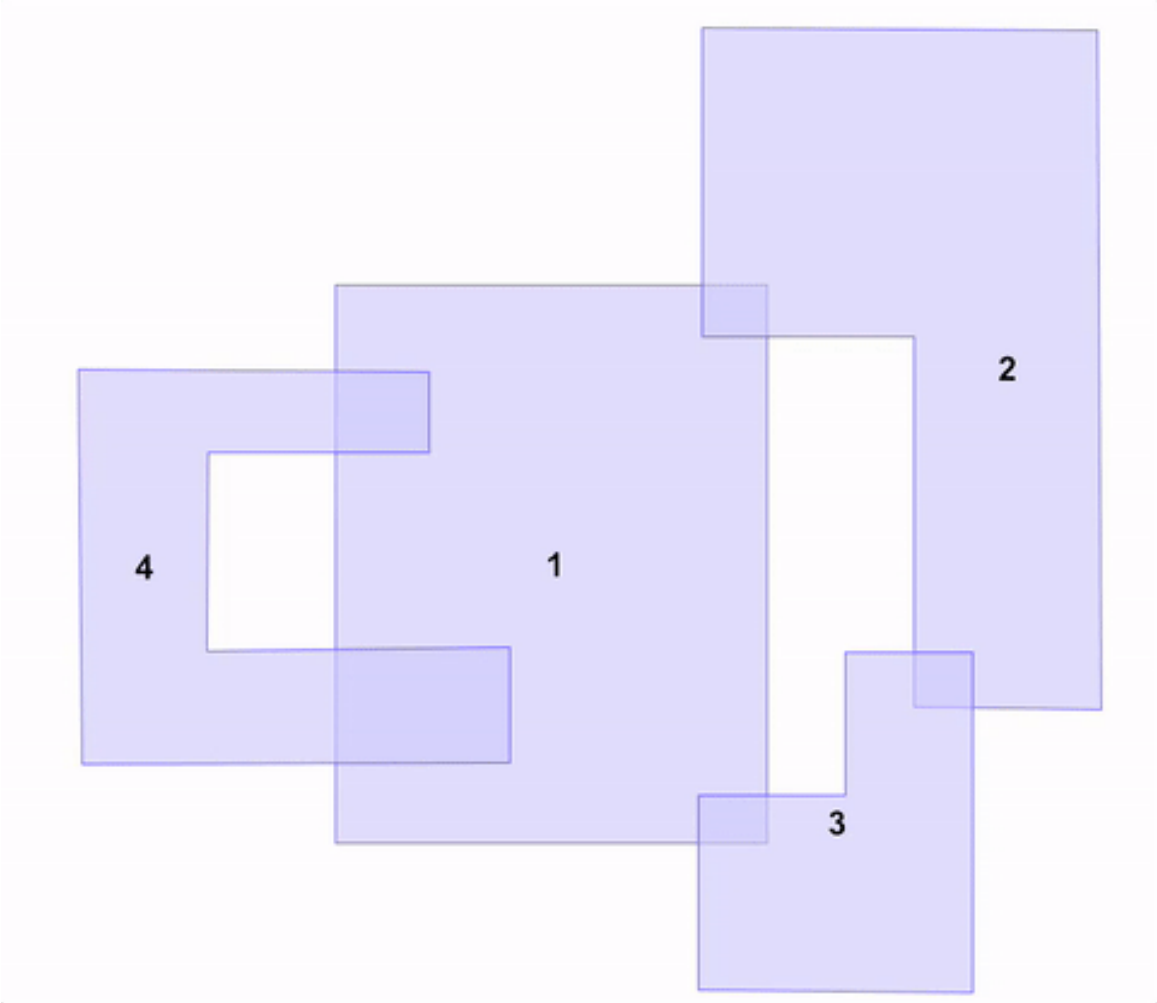
- Nettoyage de la couche parcelle:
  - Micro-intersections



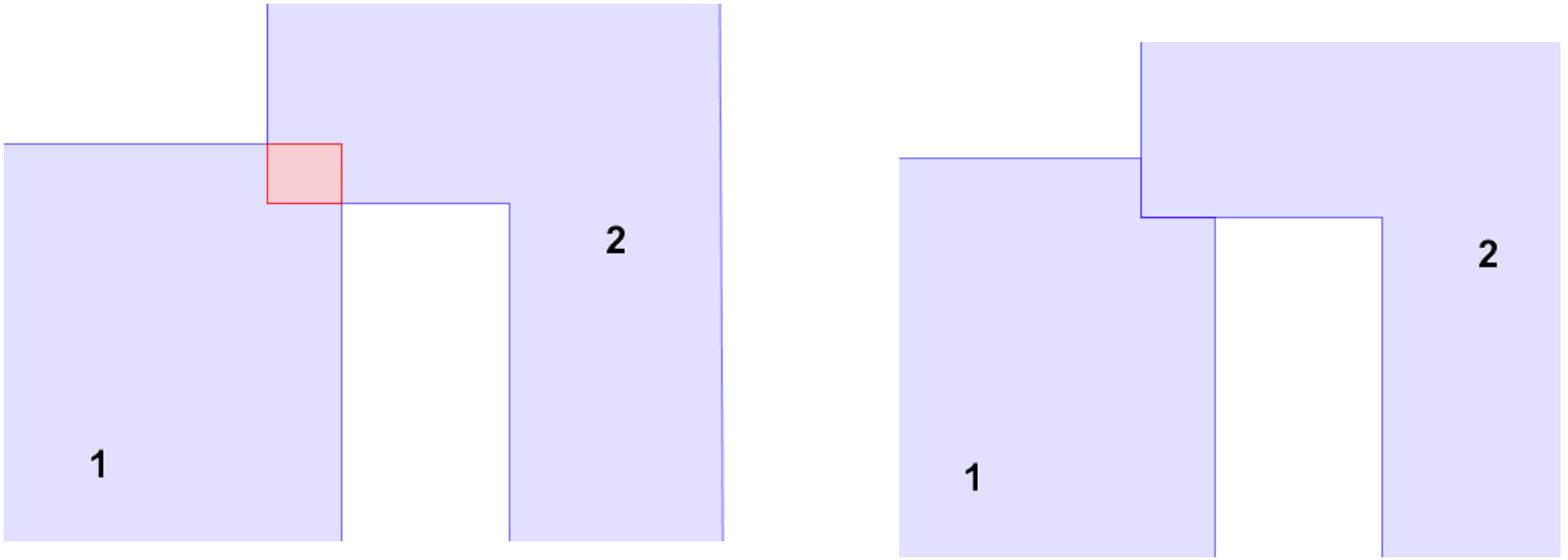


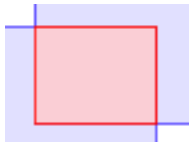


# PG Parallel: process complet



# PG Parallel: process complet



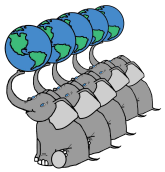



=> A unir au PG 2

=> A soustraire au PG 1

2 opérations:

- op1: union des parcelles avec intersections
- op2: différence des parcelles avec intersections



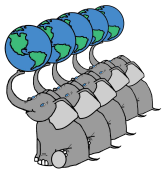
# PG Parallel: process complet

- On affecte à chaque parcelle la liste des polygones à fusionner (op1) et des polygones à soustraire (op2)

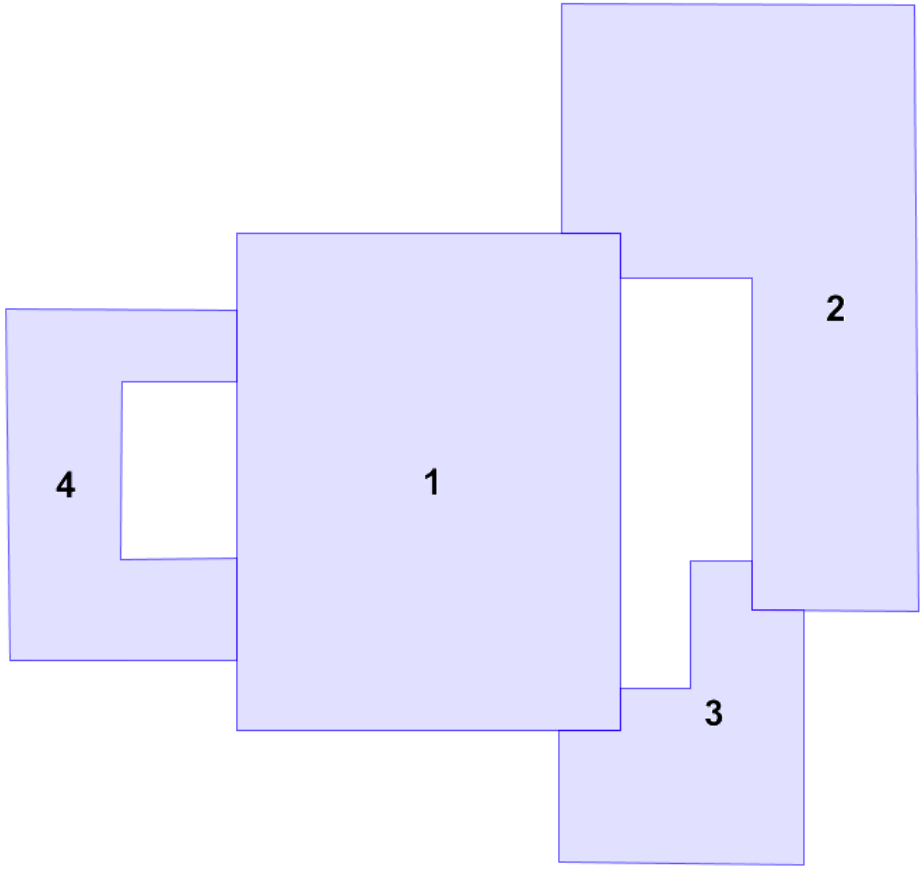
```
SELECT
      unnest(ARRAY [(p1.id, 1)::pgop_type, (p2.id, 2)::pgop_type])
  as geom_ope,
      st_intersection(p1.geom, p2.geom) AS intergeom
    FROM parcelle_sample4 p1 JOIN parcelle_sample4 p2 ON
  st_overlaps(p1.geom, p2.geom) AND p1.id < p2.id
```

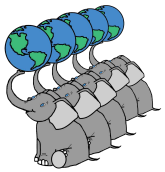
- Pour les parcelles non concernées par une opération:
  - PG intersection = GEOMETRYCOLLECTION EMPTY
  - St_union(geom, 'GEOMETRYCOLLECTION EMPTY'::geometry) = geom
- Pour chaque parcelle concernée:

```
select t.gid,
 st_difference(
   st_union(t.geom,t.uniondiffgeom[1]),t.uniondiffgeom[2]
 ) as geom  from tmp1 t;
```



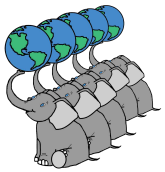
# PG Parallel: process complet



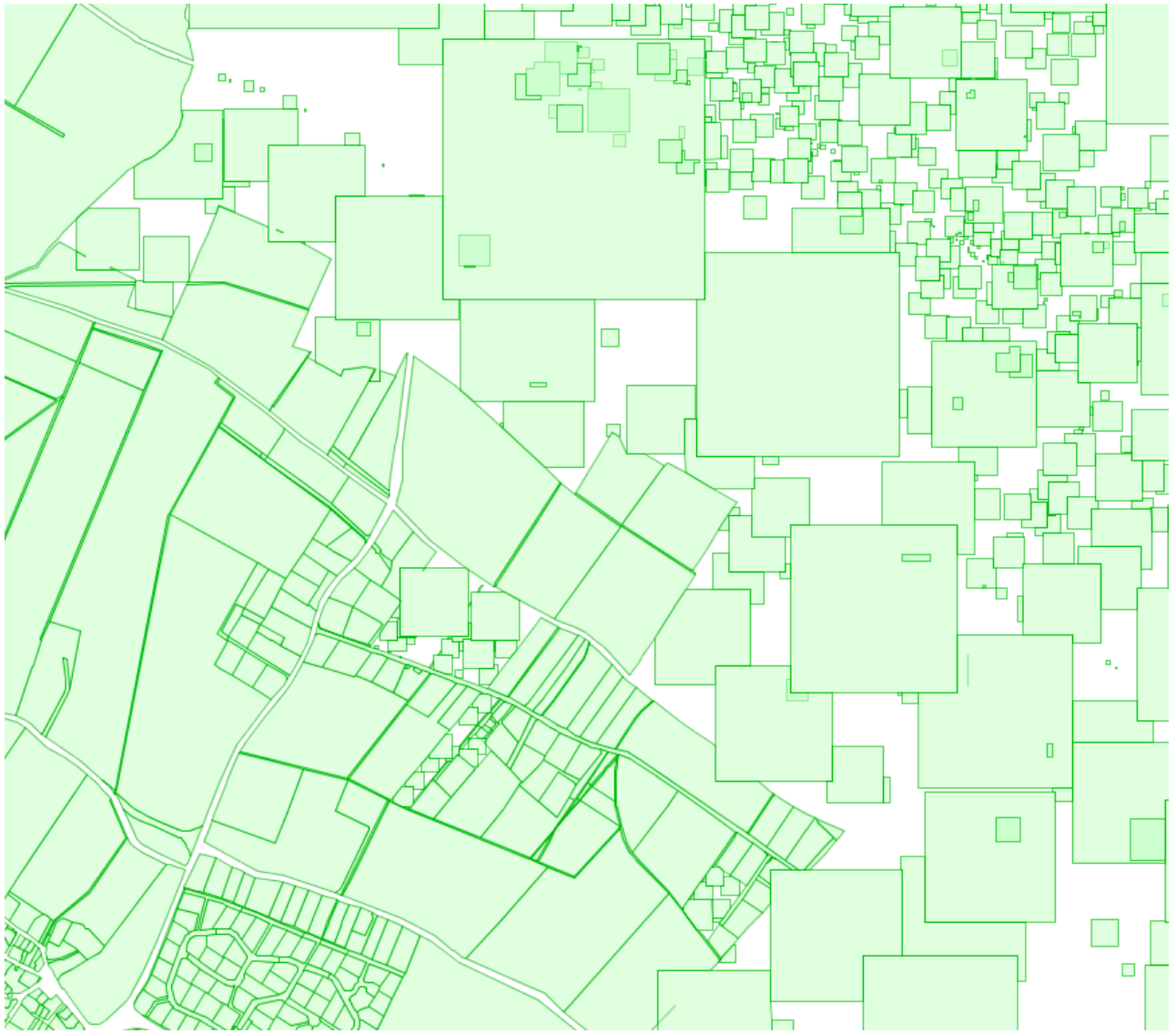


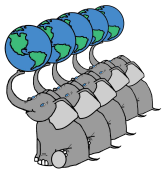
# PG Parallel: process complet

```
select * from create_table_parallel('inter1',
                'SELECT
    unnest(ARRAY [(p1.id, 1)::pgop_type, (p2.id, 2)::pgop_type]) as geom_ope,
    st_intersection(p1.geom, p2.geom) AS intergeom
  FROM parcelle_sample2 p1 JOIN parcelle_sample2 p2 ON st_overlaps(p1.geom,
   p2.geom) AND p1.id < p2.id',
                '/usr/local/pgsql-9.6/bin/psql -A -t -p 5439 -d nicolas -c',
                8, true);
create UNLOGGED table inter2_1 as (
  select (t.geom_ope).gid, (t.geom_ope).op, st_union(intergeom) as intergeom
  FROM inter1 t
  where (t.geom_ope).gid%8=0
  GROUP BY (t.geom_ope).gid, (t.geom_ope).op
); --&

create UNLOGGED table inter2_2 as (
  select (t.geom_ope).gid, (t.geom_ope).op, st_union(intergeom) as intergeom
  FROM inter1 t
  where (t.geom_ope).gid%8=1
  GROUP BY (t.geom_ope).gid, (t.geom_ope).op
); --&
```



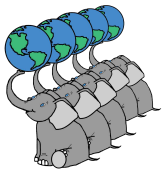
# PG Parallel: process complet





# PG Parallel: process complet

- Sample2: 175 s vs 50s => **x3.5**
- Sample1: 1814 vs 513s => **x3.5**
- Sample0: 10150 vs 3628s => **x2.8**



# PG Parallel

Questions