SP-GiST – a new indexing framework for PostgreSQL

Space-partitioning trees in PostgreSQL

Oleg Bartunov, Teodor Sigaev

Moscow University

PostgreSQL extensibility

- „The world's most advanced open source database“ from www.postgresql.org

It is imperative that a user be able to construct new access methods to provide efficient access to instances of nontraditional base types

Michael Stonebraker, Jeff Anton, Michael Hirohama.

Extendability in POSTGRES , IEEE Data Eng. Bull. 10 (2) pp.16-23, 1987

- User data types are „first class citizens“
- Adding new extensions on-line without restarting database

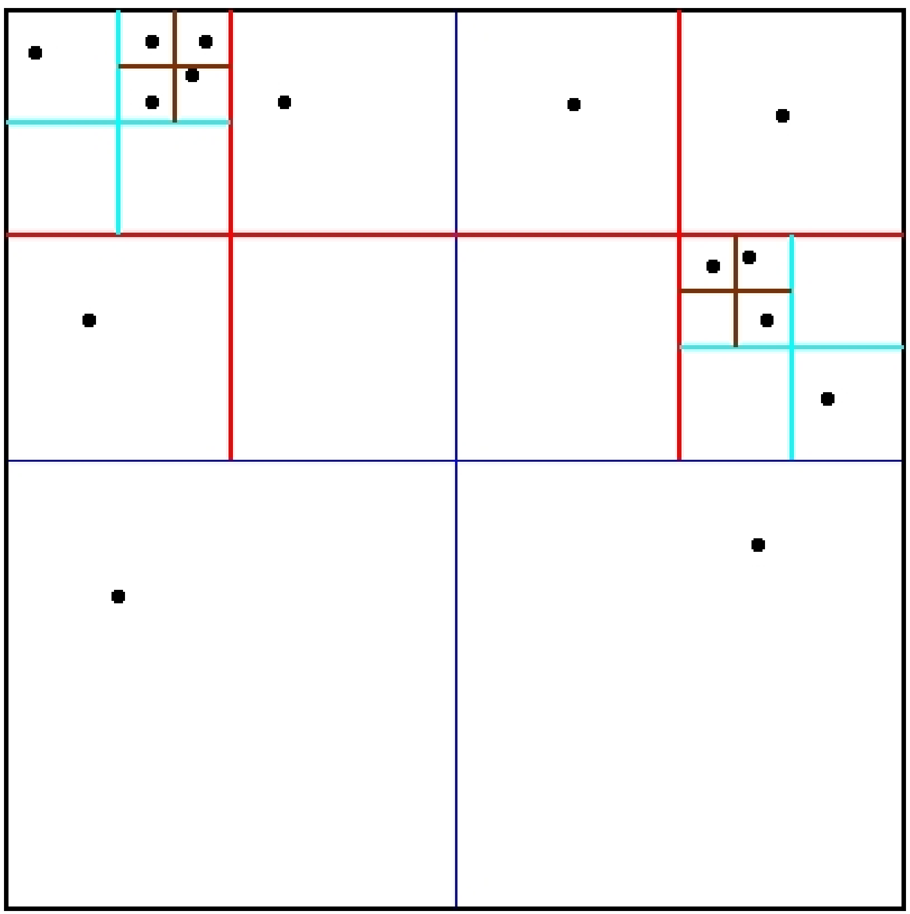
PostgreSQL extensibility

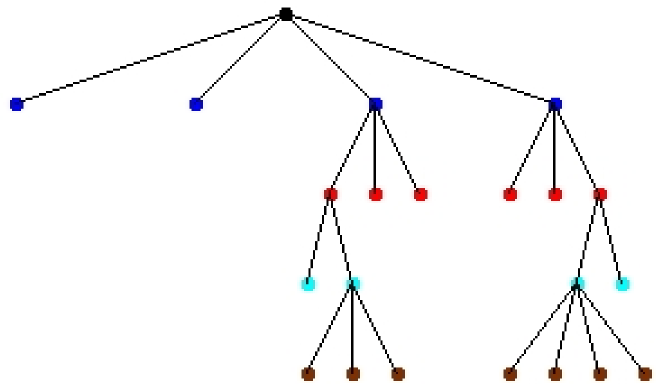
- B-tree – limited set of comparison operators (<,>,=,<=,=>)
  - All buil-in data types
- GiST – Generalized Search Tree used in many extensions
  - Ltree, hstore, pg_trgm, full text search, intarray, PostGIS
  - Many other extensions ......
- GIN – Generalized Inverted Index
  - Hstore, pg_trgm, full text search, intarray
  - Many other extensions
- Why do we talk about new indexing framework ?

PostgreSQL extensibility

- There are many interesting data structures not available
  - K-D-tree, Quadtree and many variants
    - CAD, GIS, multimedia
  - Tries, suffix tree and many variants
    - Phone routing, ip routing, substring search
- Common features:
  - Decompose space into disjoint partitions
    - Quadtree – 4 quadrants
    - Suffix tree – 26 regions (for english alphabet)
  - Unbalanced trees

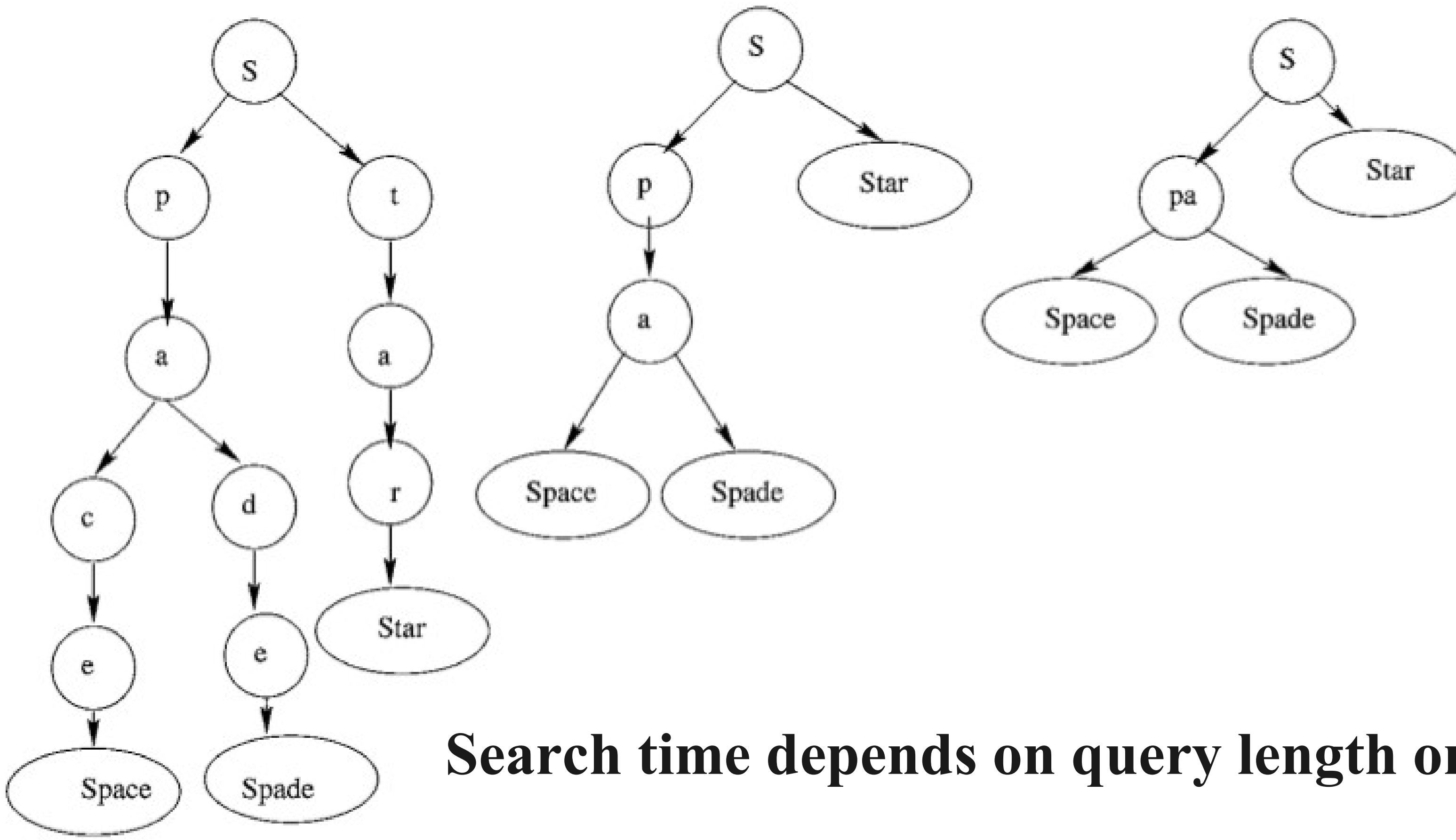
Quadtree





Suffix tree

•



(a) (b) (c)

Search time depends on query length only !

SP-GiST

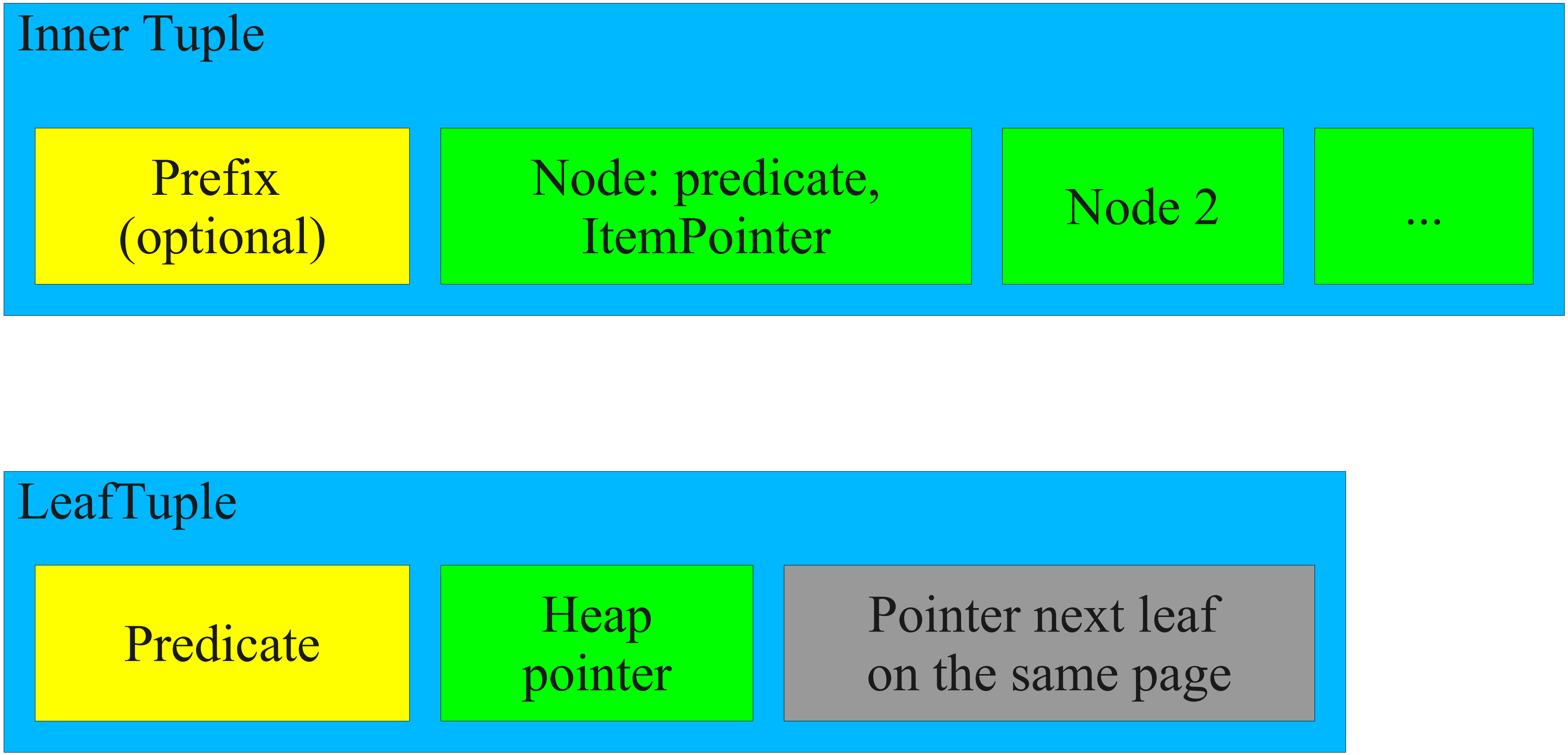
- GiST is inspired by R-tree and doesn't supports unbalanced trees
- So, we need new indexing framework for Spatial Partitioning trees:
  - Provide internal methods, which are common for whole class of space partitioning trees
  - Provide API for implementation specific features of data type

SP-GiST

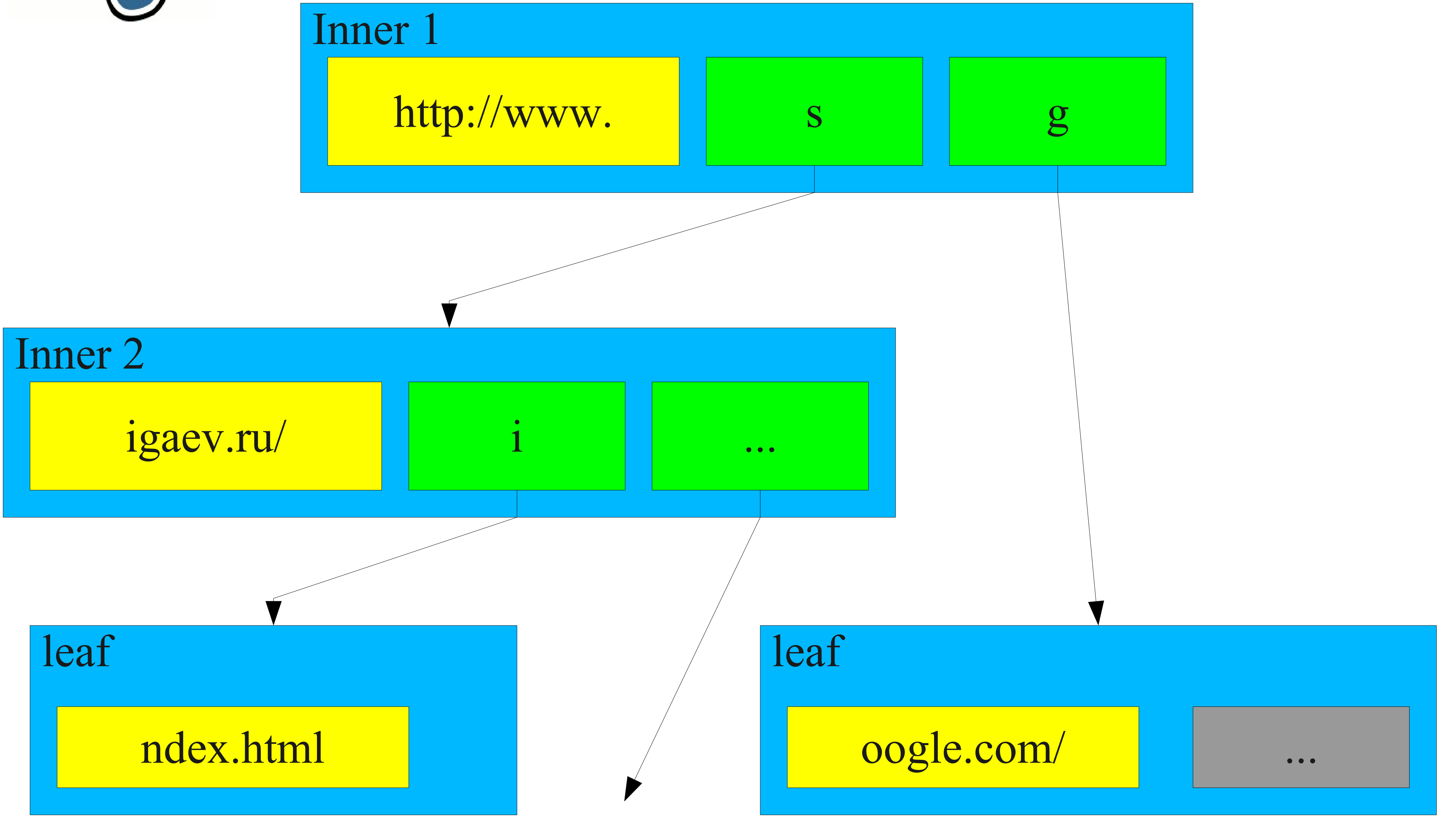
- Big Problem – Space Partitioning trees are in-memory structures and not suitable for page-oriented storage
- Several approaches:

1. Adapt structure for disk storage – difficult and not generalized
2. Introduce non-page oriented storage in Postgres - No way !
3. **Add node clustering to utilize page space on disk and preserve locality (path nodes stored close)**

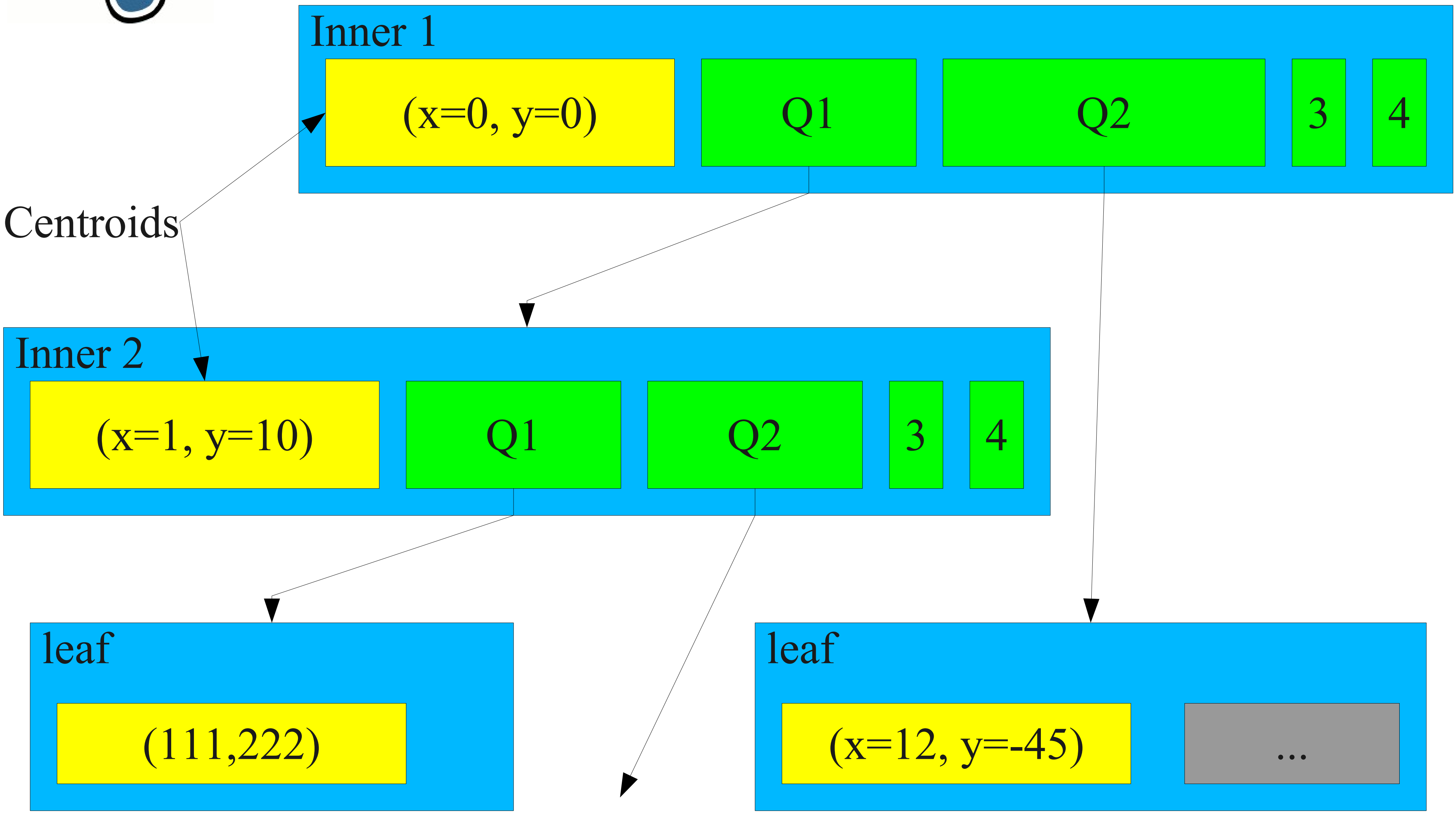
SP-GiST tuples



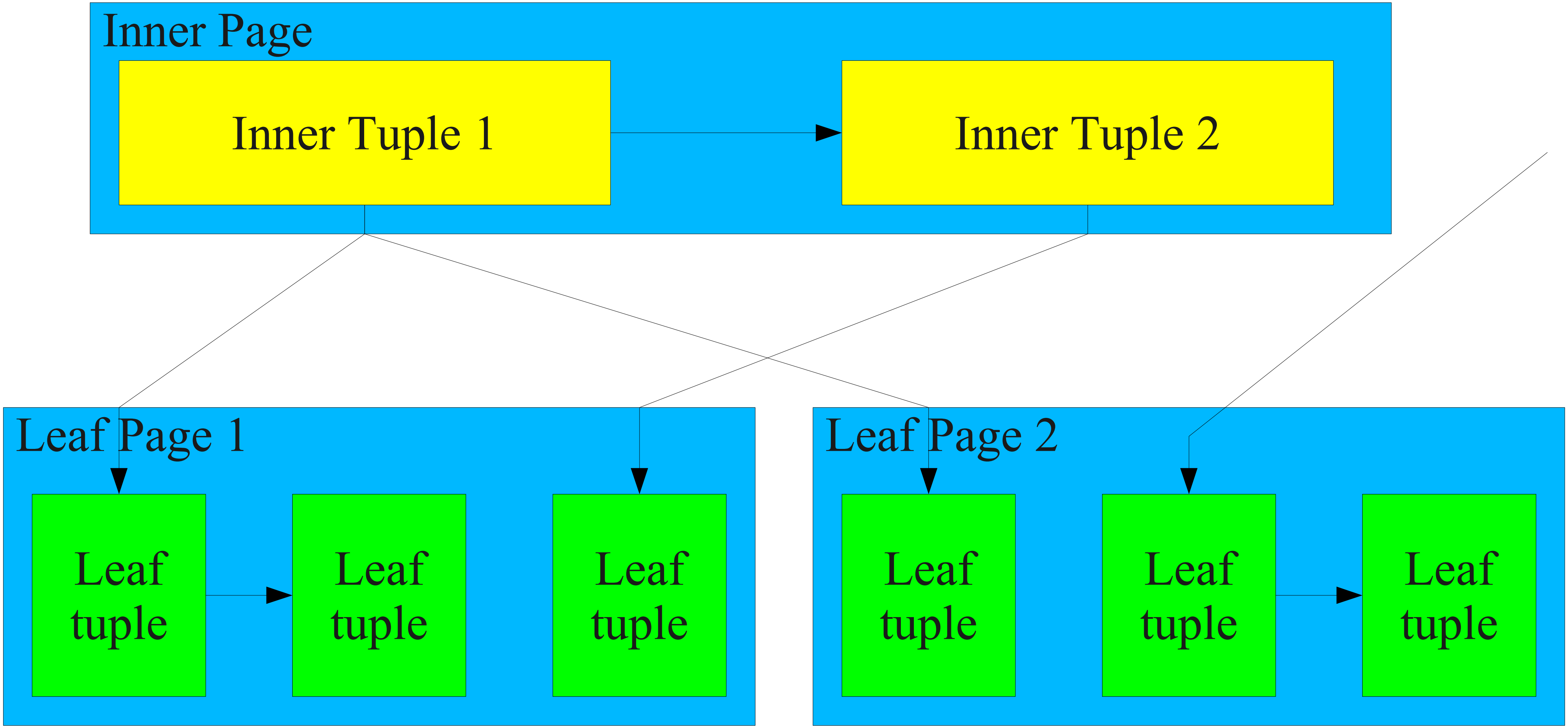
SP-GiST (suffix tree)



SP-GiST (quadtree)



SP-GiST – tuples and pages



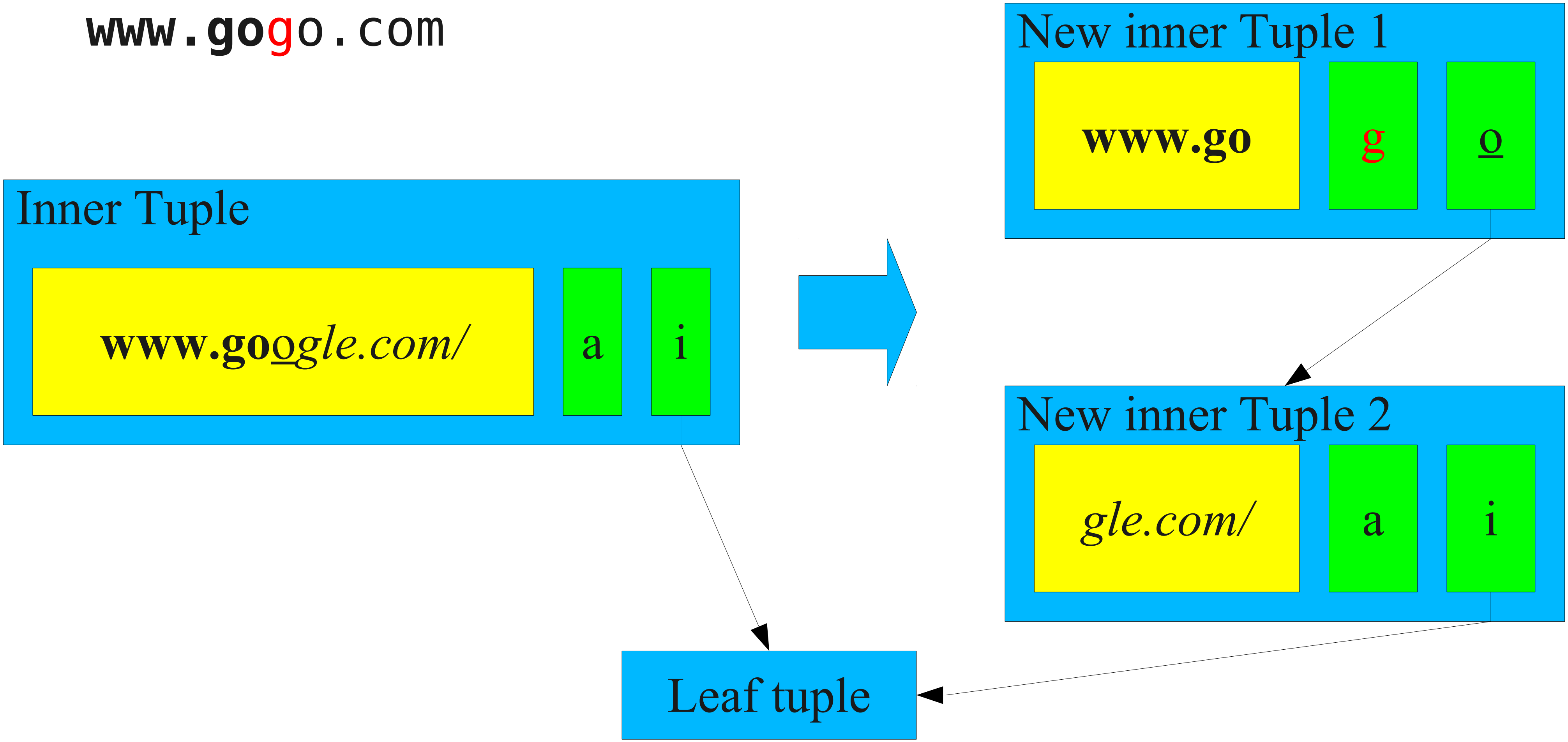
SP-GiST - interface

ConfigFn()	- returns 3 oids of data types: prefix, predicates of node and leaf tuple
ChooseFn()	- accepts content of inner node, returns one of action: - Match node - Add node to inner tuple - Split inner tuple (prefix split)
SplitFn()	- makes inner tuple from leaf page
InnerConsistentFn()	- accepts content of inner node and query, returns nodes to follow
LeafConsistentFn()	- test leaf tuple for query

Notes: all functions accepts level and full indexed value

SP-GiST ChooseFn:Split

Insert:
www.gogo.com



SP-GiST – insert algorithm

```
   Start with first tuple on root

loop:
  if (page is leaf) then
        if (enough space)
              insert
        else
              call splitFn() and resume insert from
              current place
        end if
    else
         switch by chooseFn
              case MatchNode – go by pointer and loop
                                 again
              case AddNode   – add node and insert
              case Split     – split inner tuple and
                               resume insert from current
                               place
    end if
```

Quadtree implementation

- Prefix and leaf predicate are points, node predicate is short number
- SplitFn() - just form a centroid and 4 nodes (quadrants)
- ChooseFn() - choose a quadrant (no AddNode, no split tuple)
- InnerConsistentFn() - choose quadrant(s)
- LeafConsistentFn – simple equality
- 179 lines of code

Quadtree

- Table geo (points) : 2045446 points from US geonames
  Size: 293363712

```
knn=# explain (analyze on, buffers on) select point from geo
where point ~= '(34.34898,-92.82934)';
                Abco (Arkansas,County of Hot Spring)

 Seq Scan on geo  (cost=0.00..36626.31 rows=10228 width=16)
(actual time=0.027..286.088 rows=1 loops=1)
   Filter: (point ~= '(34.34898,-92.82934)'::point)
   Buffers: shared hit=11057
 Total runtime: 286.118 ms
(4 rows)

Time: 286.659 ms
```

Quadtree

- Table geo (points) : 2045446 points from US geonames
- GiST

```
knn=# create index pt_gist_idx on geo using gist(point);
CREATE INDEX
Time:  36672.283 ms
Size: 153,124,864
```

- SP-GiST

```
knn=# create index pt_spgist_idx on geo using spgist(point)
CREATE INDEX
Time: 12805.530 ms  ~ 3 times faster !
Size: 153,788,416     ~ the same size
```

Quadtree

- GiST

```
knn=# explain (analyze on, buffers on) select point from geo where point
~= '(34.34898,-92.82934)';

 Bitmap Heap Scan on geo  (cost=456.26..11872.18 rows=10227 width=16)
(actual time=0.188..0.188 rows=1 loops=1)
   Recheck Cond: (point ~= '(34.34898,-92.82934)'::point)
   Buffers: shared hit=12
   ->  Bitmap Index Scan on pt_gist_idx  (cost=0.00..453.70 rows=10227
width=0) (actual time=0.179..0.179 rows=1 loops=1)
         Index Cond: (point ~= '(34.34898,-92.82934)'::point)
         Buffers: shared hit=11
 Total runtime: 0.235 ms
```

Quadtree

- SP-GiST

```
knn=# explain (analyze on, buffers on) select point from geo where point
~= '(34.34898,-92.82934)';

 Bitmap Heap Scan on geo  (cost=576.50..11992.42 rows=10227 width=16)
(actual time=0.041..0.041 rows=1 loops=1)
   Recheck Cond: (point ~= '(34.34898,-92.82934)'::point)
   Buffers: shared hit=6
   ->  Bitmap Index Scan on pt_spgist_idx  (cost=0.00..573.94 rows=10227
width=0) (actual time=0.033..0.033 rows=1 loops=1)
         Index Cond: (point ~= '(34.34898,-92.82934)'::point)
         Buffers: shared hit=5
 Total runtime: 0.083 ms ~ 6 times faster !
```

Quadtree

- Page space utilization

```
knn=# select spgstat('pt_spgist_idx');
           spgstat
------------------------------
 totalPages:  18772          +
 innerPages:  803            +
 leafPages:   17969          +
 emptyPages:  32             +
 usedSpace:   64340.80 kbytes+
 freeSpace:   85321.91 kbytes+
 FillRatio:     42.99%       +
 leafTuples:  2045446        +
 innerTuples: 5982
(1 row)
```

Quadtree

- Conclusions
  - Index creation is fast (3 times faster than GiST) even in prototype.
  - Current page utilization is ~ 40% ! Index size can be improved using better clustering technique
  - Search is very fast (~ 3 times faster than GiST) for ~= operation. Need to implement other operations.

Suffix tree implementation

- Prefix and leaf predicate are texts, node predicate is char (byte)
- Interface functions are quite complex because of prefix support
- Interface functions takes into account current level in tree
- 329 lines of code (not so much!)

Suffix tree

- 4 mln urls from uk domain (10-20 url from each server)
- Btree (Size=396,730,368), create index ~ 19 sec

```
test=# explain (analyze on, buffers on) select * from t1
where t = 'http://0-2000webhosting.co.uk/super-submit.htm';
                                                    QUERY PLAN
------------------------------------------------------------------------------------
 Index Scan using t1_bt_idx on t1  (cost=0.00..10.20 rows=1 width=72)
(actual time=0.095..0.096 rows=1 loops=1)
   Index Cond: (t = 'http://0-2000webhosting.co.uk/super-submit.htm'::text)
   Buffers: shared hit=6
 Total runtime: 0.126 ms
```

Suffix tree

- 4 mln urls from uk domain (10-20 url from each server)
- SP-GiST (Size=1,797,554,176), create index ~ 28 sec

```
test=# explain (analyze on, buffers on) select * from t1
where t = 'http://0-2000webhosting.co.uk/super-submit.htm';
                                                     QUERY PLAN
--------------------------------------------------------------------------------
 Bitmap Heap Scan on t1  (cost=13.03..17.05 rows=1 width=72)
(actual time=0.030..0.030 rows=1 loops=1)
   Recheck Cond: (t = 'http://0-2000webhosting.co.uk/super-submit.htm'::text)
   Buffers: shared hit=4
   ->  Bitmap Index Scan on t1_spg_idx  (cost=0.00..13.03 rows=1 width=0)
(actual time=0.021..0.021 rows=1 loops=1)
         Index Cond: (t = 'http://0-2000webhosting.co.uk/super-submit.htm'::text
         Buffers: shared hit=3
 Total runtime: 0.075 ms  ~ 4 times faster !
```

Suffix tree

- Page space utilization

```
test=# select spgstat('t1_spg_idx');
            spgstat
--------------------------------
 totalPages:  219427           +
 innerPages:  4965             +
 leafPages:   214462           +
 emptyPages:  0                +
 usedSpace:   228026.99 kbytes +
 freeSpace:   1521389.05 kbytes+
 fillRatio:     13.03%         +
 leafTuples:  4000000          +
 innerTuples: 44144
(1 row)
```

Suffix tree

- Conclusions
  - Index creation is slower than Btree (28 sec vs 19 sec)
  - Current page utilization is ~ 13% ! Index size ~4 times bigger than Btree, can be ½ of Btree index if 100% utilization.
  - Search is very fast (~ 4 times faster than Btree) for = operation. Need to implement other operations.

SP-GiST TODO

- Improve page utilization (Clustering)
- Concurrency
- WAL
- Vacuum
- Spggettuple()
- Amcanorder
- Add operations
- K-d-tree? Btree emulation? Something else?
- KNN ? (amcanorderbyop)

SP-GiST links

- SP-GiST publications
  - http://www.cs.purdue.edu/spgist/
- Downloads

  `http://www.sigaev.ru/misc/spgist-0.37.tgz`