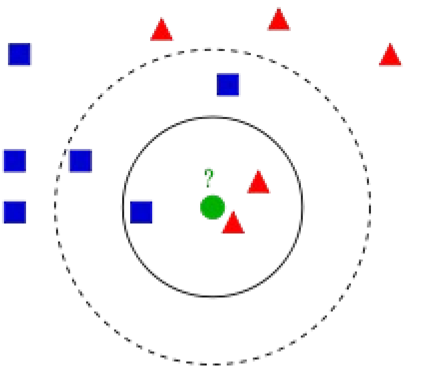
K-nearest neighbour search for PostgreSQL

Oleg Bartunov, Teodor Sigaev

Moscow University

Knn-search: The problem

- What are interesting points near Royal Oak pub in Ottawa ?
- What are the closest events to the May 20, 2009 in Ottawa ?
- Similar images – feature extraction, Hamming distance
- Classification problem (major voting)
- ....
- GIS, Science (high-dimensional data)



Knn-search: Existing solutions

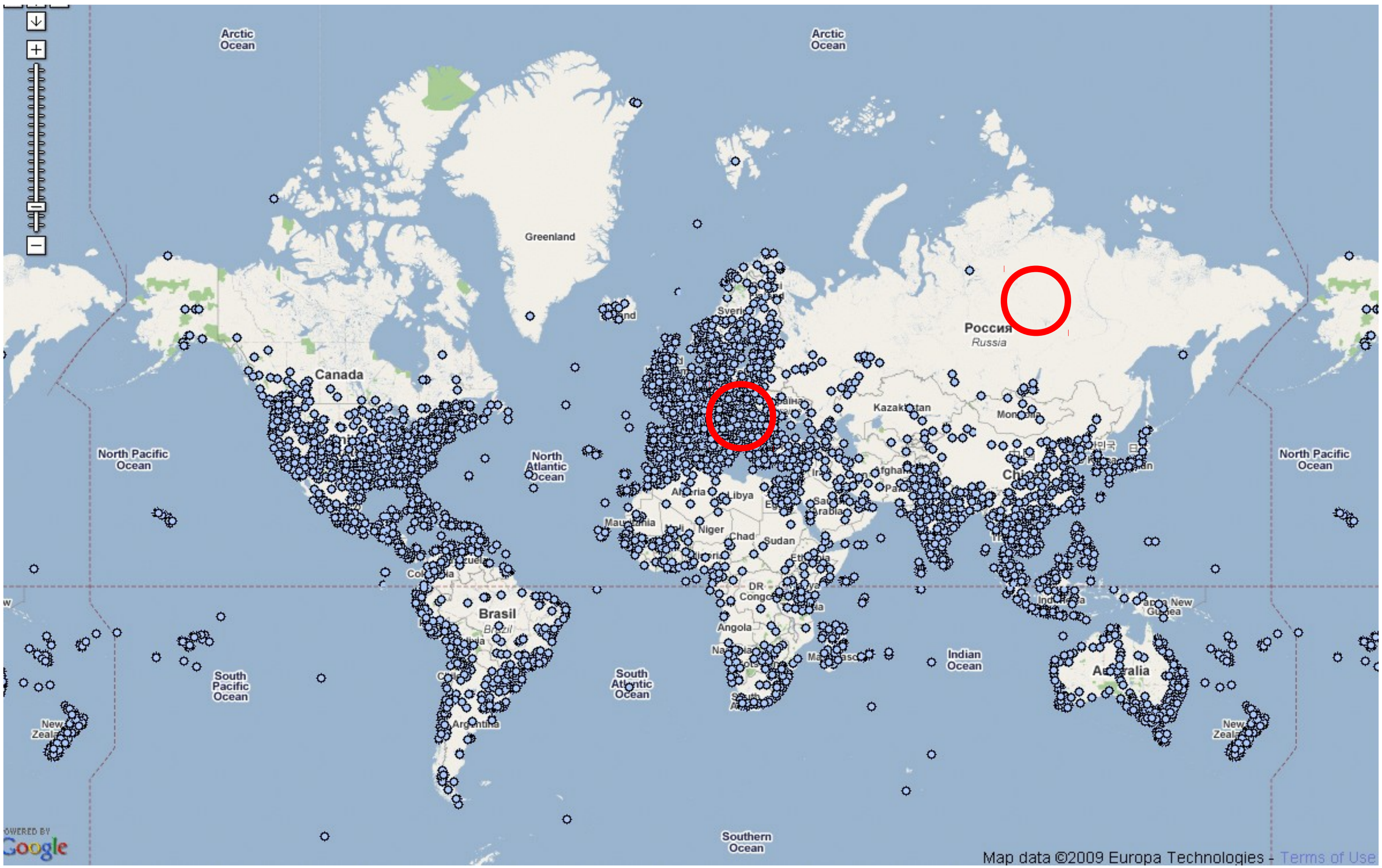
```
knn=# select id, date, event from events order by date <-> '1957-10-04'::date asc
limit 10;
   id   |    date    |                                 event
--------+------------+-----------------------------------------------------------------
  58137 | 1957-10-04 | U.S.S.R. launches Sputnik I, 1st artificial Earth satellite
  58136 | 1957-10-04 | "Leave It to Beaver," debuts on CBS
 117062 | 1957-10-04 | Gregory T Linteris, Demarest, New Jersey, astronaut, sk: STS 83
 117061 | 1957-10-04 | Christina Smith, born in Miami, Florida, playmate, Mar, 1978
 102670 | 1957-10-05 | Larry Saumell, jockey
  31456 | 1957-10-03 | Willy Brandt elected mayor of West Berlin
  58291 | 1957-10-05 | 12th Ryder Cup: Britain-Ireland, 7 -4 at Lindrick GC, England
  58290 | 1957-10-05 | 11th NHL All-Star Game: All-Stars beat Montreal 5-3 at Montreal
  58292 | 1957-10-05 | Yugoslav dissident Milovan Djilos sentenced to 7 years
 102669 | 1957-10-05 | Jeanne Evert, tennis player, Chris' sister
(10 rows)

Time: 115.548 ms
```

- Very inefficient:
  - Full table scan, classic B-tree index on date won't help.
  - Sort full table

Knn-search: Existing solutions

- Traditional way to speedup query
  - Constrain data space (range search)
    - Range search can use index
    - Incremental search → to many queries
    - Need to know in advance size of neighbourhood, how ?
      1Km is ok for Paris, but too small for Siberia
    - Maintain 'density map' ?

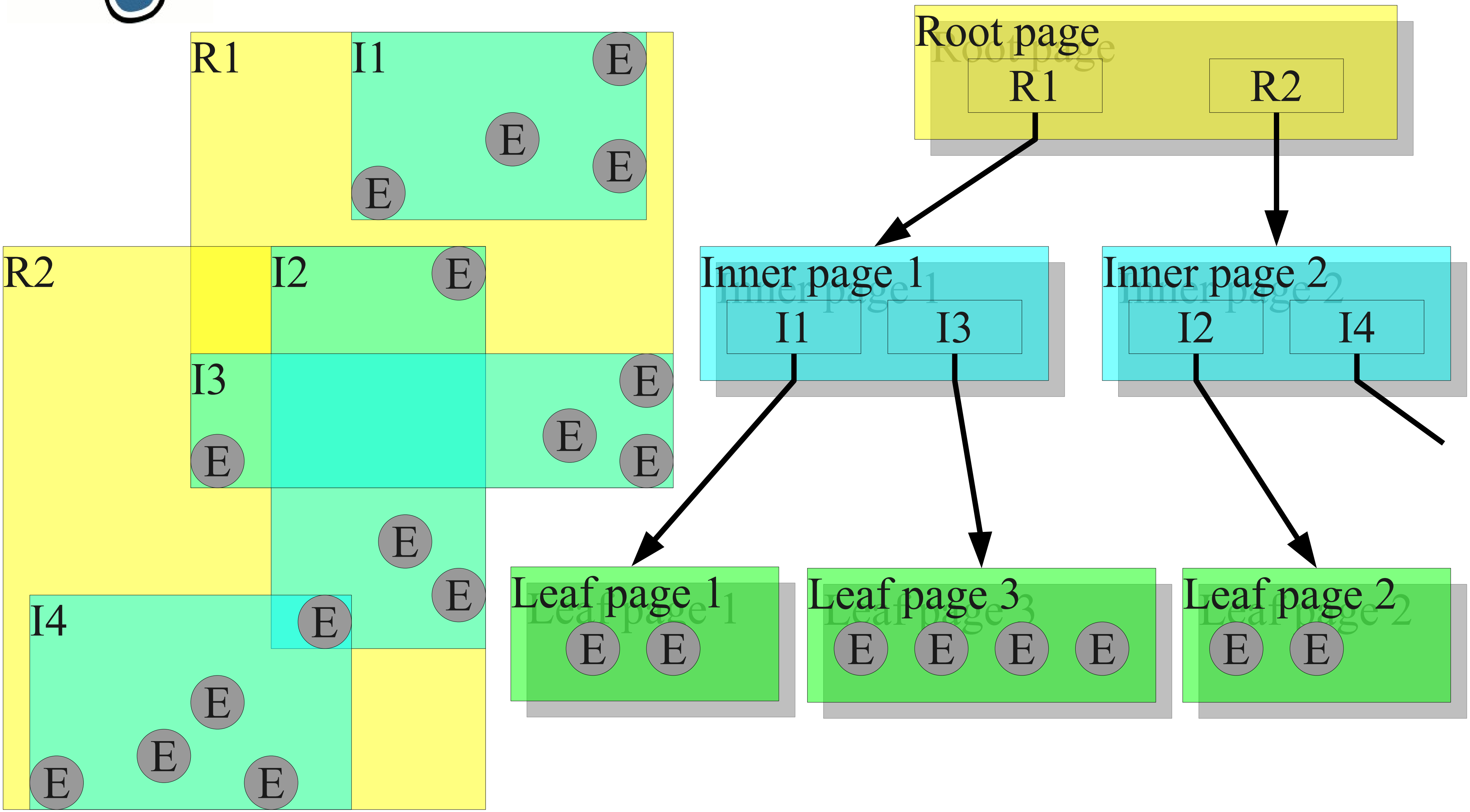
What's a neighbourhood ?



Knn-search: What do we want !

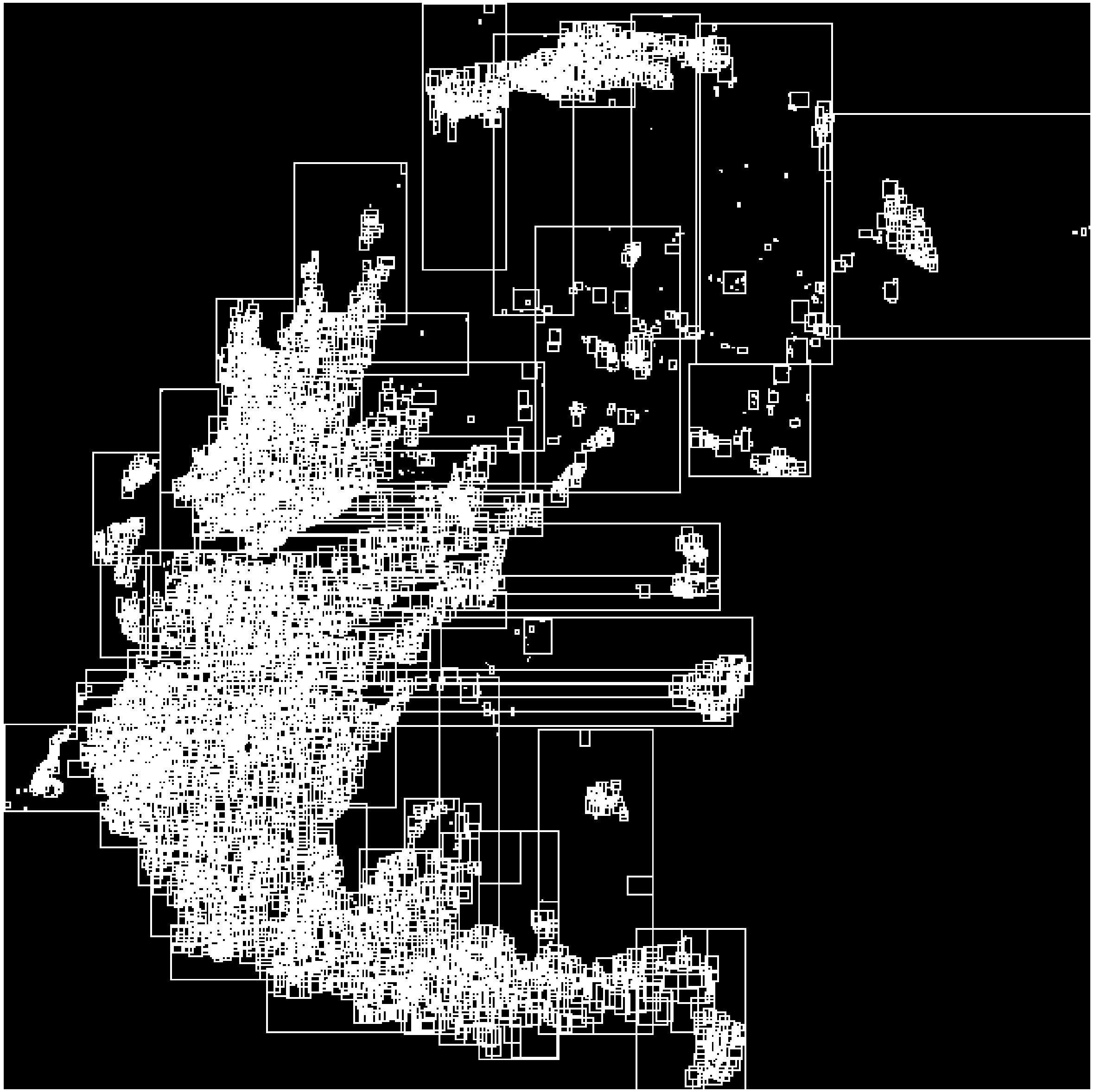
- We want to avoid full table scan – read only <right> tuples
  - So, we need index
- We want to avoid sorting – read <right> tuples in <right> order
  - So, we need special strategy to traverse index
- We want to support tuples visibility
  - So, we should be able to resume index traverse

R-tree index

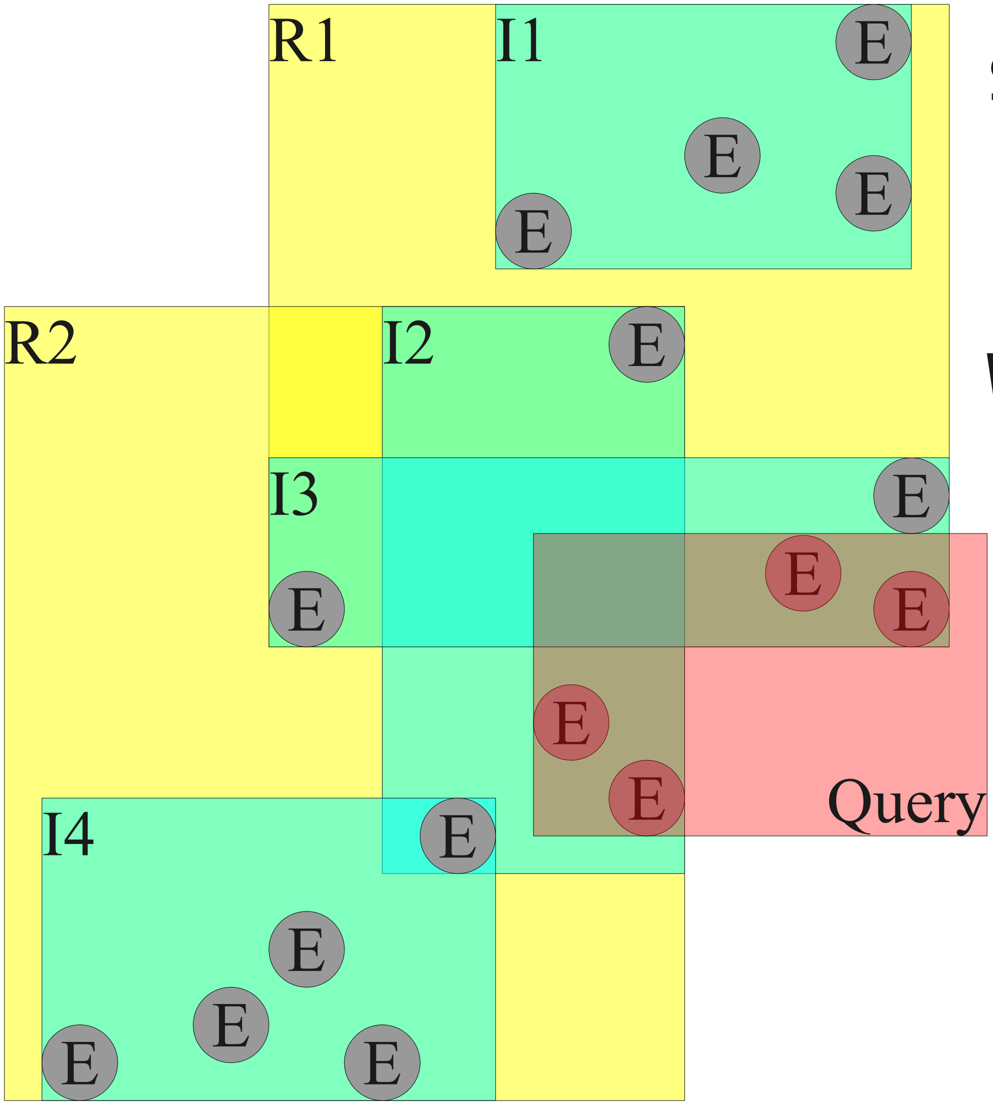

R1
I1
E
E
E
E
R2
I2
E
I3
E
E
E
E
E
E
I4
E
E
E
E
E
Root page
R1
R2
Inner page 1
I1
I3
Inner page 2
I2
I4
Leaf page 1
E E
Leaf page 3
E E E E
Leaf page 2
E E

R-tree index

- Visualization of R-tree index using Gevel.
- Greece (data from rtreeportal.org)



R-tree index




```
SELECT
   *
FROM
   events
WHERE
   events.coord <@ 'QUERY';
```

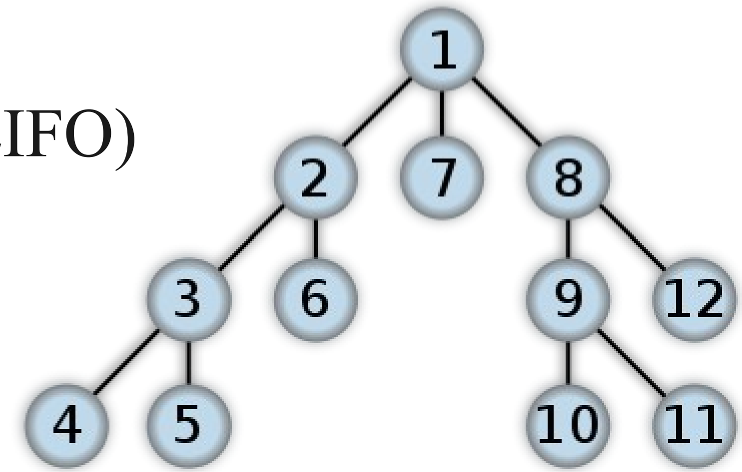
- Root page: R1, R2 keys
- Inner pages: I3, I2 keys
- Leaf pages: 4 points

- Very efficient for Search !

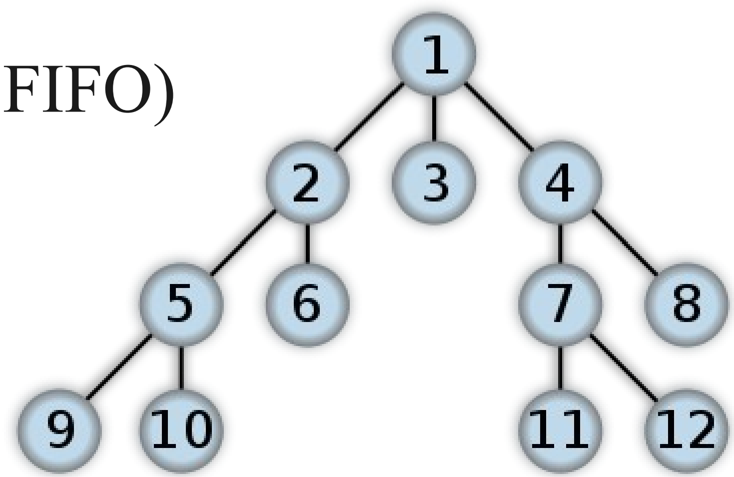
Knn-search: Index traverse

- Depth First Search (stack, LIFO)

  R-tree search



- Breadth First Search (queue, FIFO)



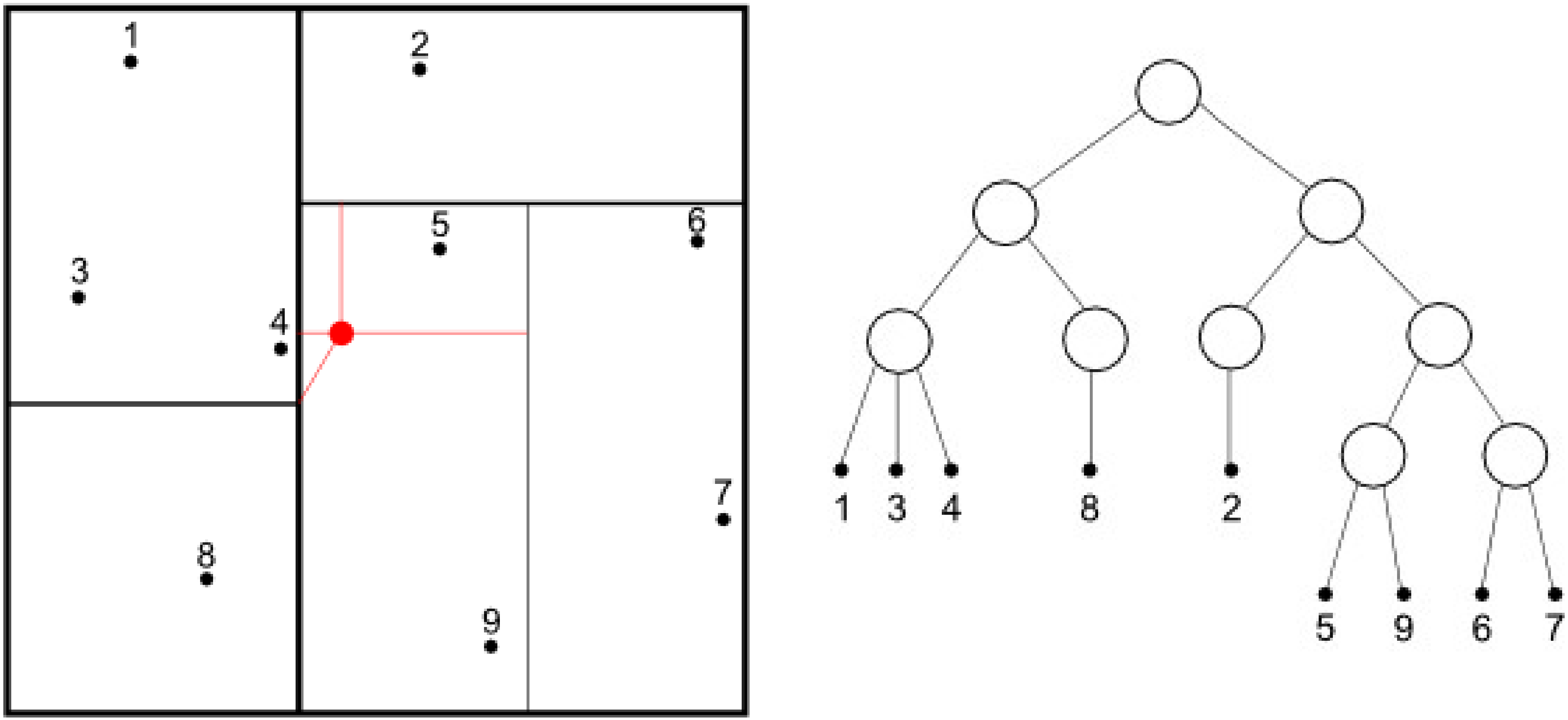
- Both strategies are not good for us – full index scan

Knn-search: Index traverse

- Best First Search (PQ, priority queue). Maintain order of items in PQ according their distance from given point
  - Distance to MBR (rectangle for Rtree) for internal pages – minimum distance of all items in that MBR
  - Distance = 0 for MBR with given point
  - Distance to point for leaf pages
- Each time we extract point from PQ we output it – it is next closest point ! If we extract rectangle, we expand it by pushing their children (rectangles and points), which match WHERE clause into the queue.
- We traverse index by visiting only interesting nodes !

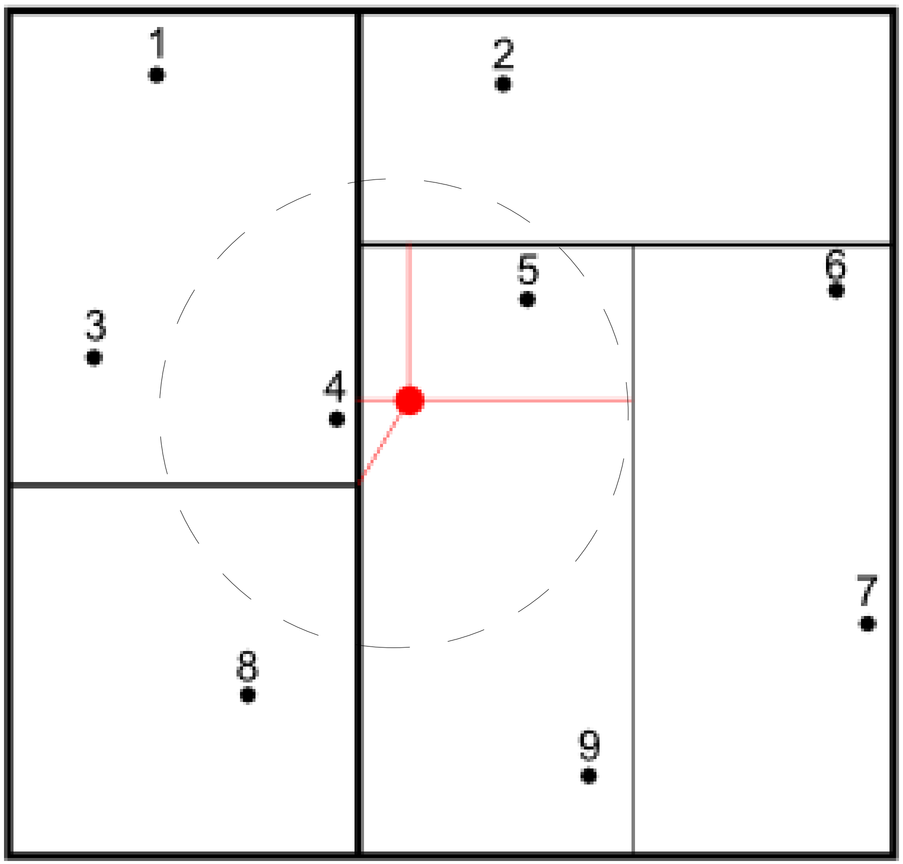
Knn-search: Index traverse

- Simple example – non-overlapped partitioning



Knn-search: Index traverse

- Simple example – non-overlapped partitioning

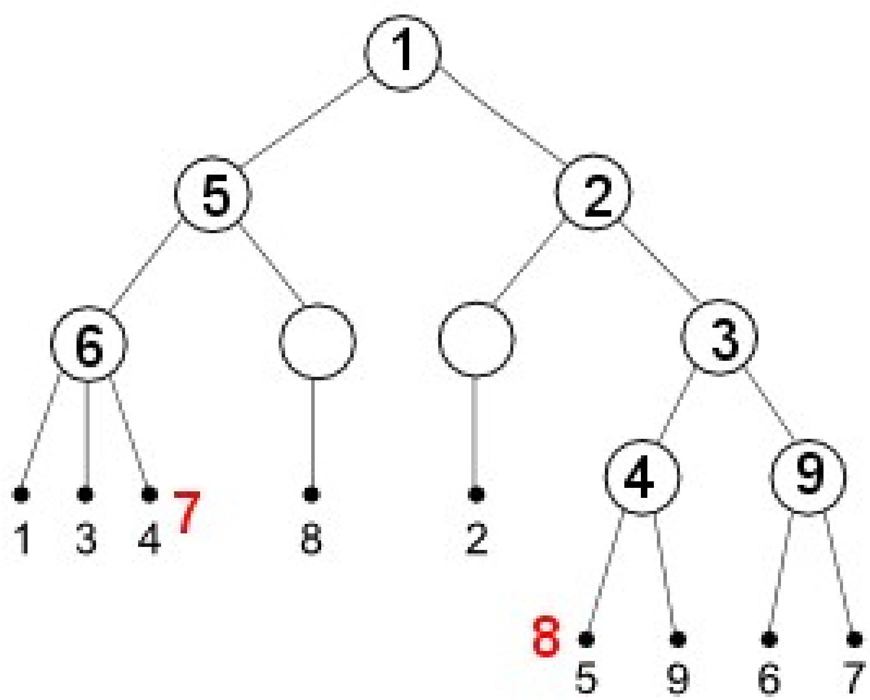


- Priority Queue
- 1: {1,2,3,4,5,6,7,8,9}
- 2: {2,5,6,7,9}, {1,3,4,8}
- 3: {5,6,7,9}, {1,3,4,8}, {2}
- 4: {5,9}, {1,3,4,8}, {2}, {6,7}
- 5: {1,3,4,8}, 5, {2}, {6,7}, 9
- 6: {1,3,4}, {8}, 5, {2}, {6,7}, 9
- 7: 4, {8}, 5, {2}, {6,7}, 3, 1, 9

  we can output 4 without visit other rectangles !
- 8: 5, {2}, {6,7}, 3, 8, 1, 9
- 9: {6,7}, 3, 2, 8, 1, 9
- 10: 3, 2, 8, 1, 9, 6, 7

Knn-search: Index traverse

- Simple example – non-overlapped partitioning



- Priority Queue
- 1: {1,2,3,4,5,6,7,8,9}
- 2: {2,5,6,7,9}, {1,3,4,8}
- 3: {5,6,7,9}, {1,3,4,8}, {2}
- 4: {5,9}, {1,3,4,8}, {2}, {6,7}
- 5: {1,3,4,8}, 5, {2}, {6,7}, 9
- 6: {1,3,4}, {8}, 5, {2}, {6,7}, 9
- **7:** 4, {8}, 5, {2}, {6,7}, 3, 1, 9
- 8: 5, {2}, {6,7}, 3, 8, 1, 9

Knn-search: Performance

- SEQ (no index) – base performance
  - Sequentually read full table + Sort full table (can be very bad, work_mem !)

- BFS – the best for small k !
  - Partial index scan + Random read k-records
    - T(index scan) ~ Height of Search tree ~ log(n)
  - T(BFS) ~ k, for small k. The more rows, the more benefit !
  - Can still win even for k=n (for large tables) - no sort !

Knn-search: What do we want !

- + We want to avoid full table scan – read only <right> tuples
  - So, we need index
- + We want to avoid sorting – read <right> tuples in <right> order
  - So, we need special strategy to traverse index
- + We want to support tuples visibility
  - So, we should be able to resume index traverse
- We want to support many data types
  - So, we need to modify GiST

Knn-search: modify GiST

- GiST – Generalized Search Tree, provides
  - API to build custom disk-based search trees (any tree, where key of internal page is a Union of keys on children pages)
  - Recovery and Concurrency
  - Data type and query extendability
- GiST is widely used in GIS (PostGIS), text search,...
- Current strategy of search tree traverse is DFS
  - Not good for knn-search
  - We need to add Best First Search strategy for knn-search
  - Retain API compatibility

Knn-search: syntax

Knn-query uses ORDER BY clause

```
SELECT … FROM … WHERE …
ORDER BY p <-> '(0.0, 0.0)'::point
LIMIT k;
```

<-> - distance operator, should be provided for data type

GiST interface

- compress/decompress
- same
- union
- penalty
- picksplit
- **Consistent** – controls search tree traverse

GiST changes

```
! bool consistent(
                      Datum key,
                      Datum query,
                      StrategyNumber strategy,
                      Oid subtype /* unused */,
                      bool *recheck );

--- XXX, YYY ---

! double consistent(
                      Datum key,
                      Datum query,
                      StrategyNumber strategy,
                      Oid subtype /* unused */,
                      bool *recheck );
```

Return value of consistent

- < 0 - query doesn't match WHERE clause. Forbidden for ORDER BY clause
- = 0 - exact match for WHERE clause or zero distance for ORDER BY clause
- > 0 - distance for ORDER BY clause
- „wrapper“ for old consistent method:
  false => -1
  true => 0

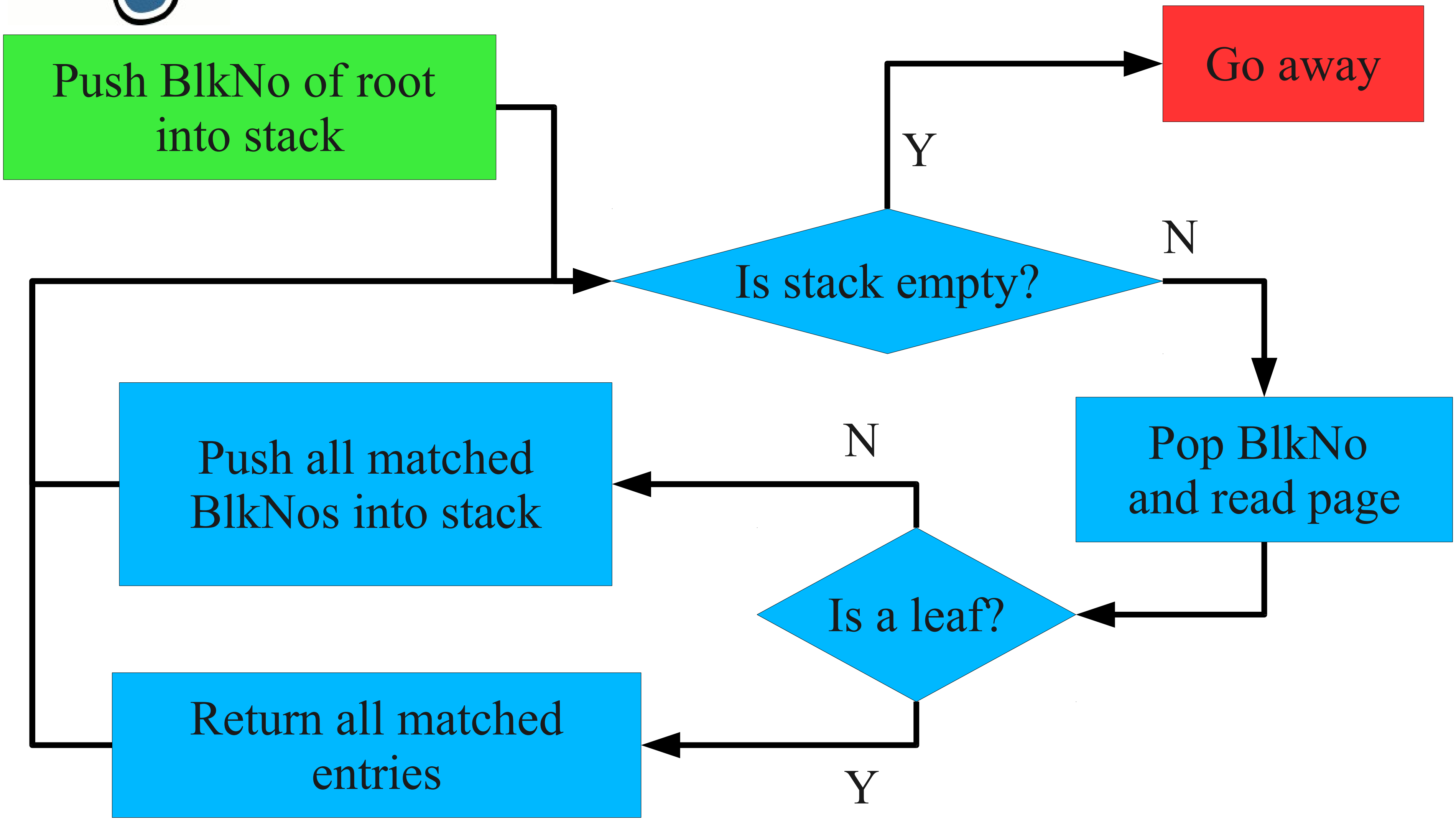
Consistent interface

- GiST's traverse algorithm treats WHERE and ORDER BY clauses in uniform way.
- Consistent from strategy number knows data types of query and WHERE/ORDER BY clauses.
- Consistent should not return `recheck = true` for ORDER BY clause – how to order data, which need recheck ?

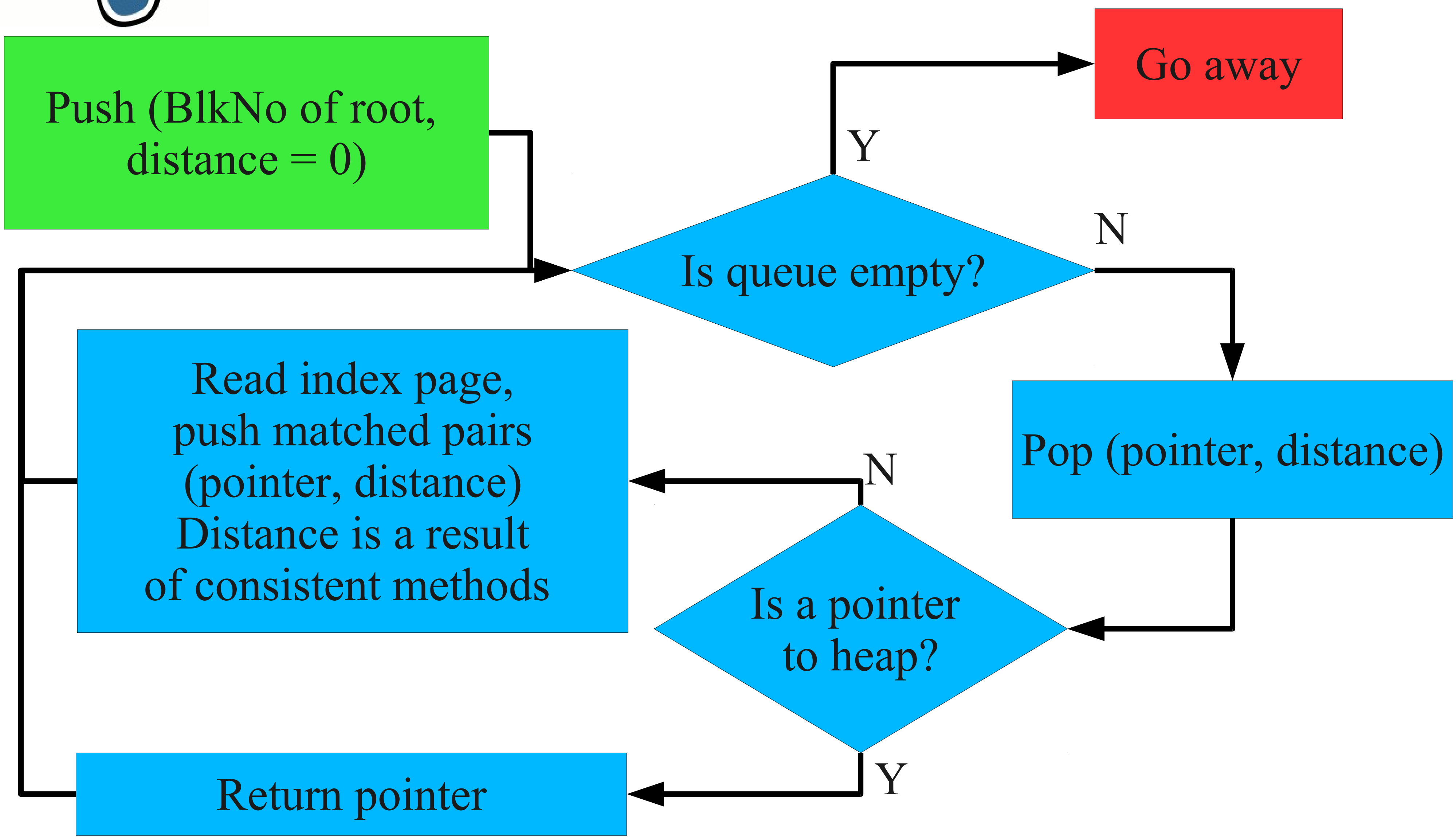
The problem

- We need to recognize if operator is from ORDER BY clause – different work with NULL values
  - For WHERE clause strict operator should discard NULL
  - For ORDER BY assume distance is infinity (ASC NULLS LAST)
- Currently, we do this by operation's returned value – non-bool type
- Option 1: add flag to pg_amop to indicate, that operator used in ORDER BY clause
  - bool returned operator could be duplicated in operator family → too many work to allow index support for boolean distance
- Option 2:   if operator returns DOUBLE – it's knn-search

GiST + Depth First Search

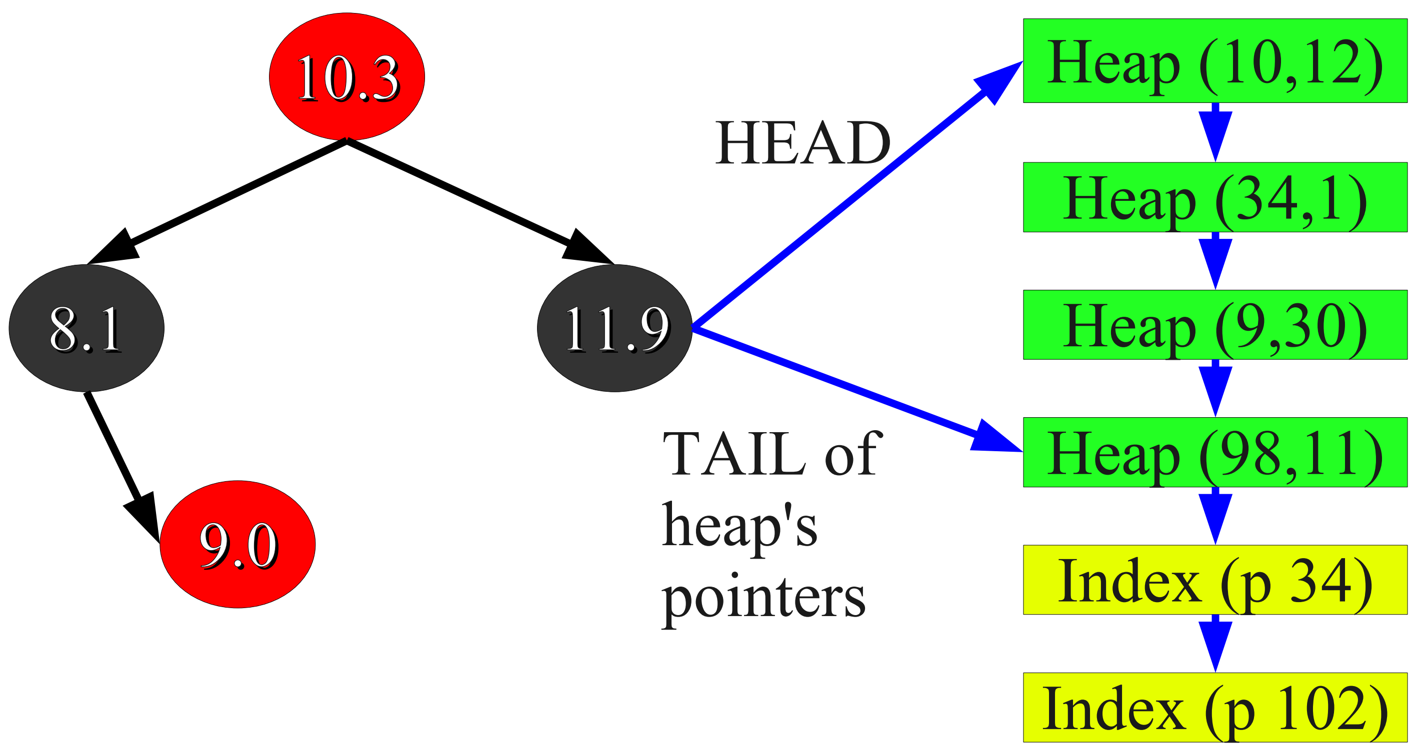


KNN-search: GiST + Priority Queue



GiST: Technical details

- Priority queue is implemented as a RB-tree (Red-Black tree)
- Each node of RB-tree contains a list of pointers - pointers to internal pages follow pointers to heap.



GiST: Technical details

Depth First Search

```
push Stack, Root;
While Stack {
   If p is heap {
             output p;
   else {
             children = get_children(p);
             push Stack, children;
        }
}
```

Best First Search

```
push PQ, Root;
While PQ {
   If p is heap {
             output p;
   else {
             Children = get_children(p);
             push PQ, children;
        }
}
```

- For non-knn search all distances are zero, so PQ => Stack and BFS => DFS
- We can use only one strategy for both – normal search and knn-search !

Knn-search: What do we want !

- + We want to avoid full table scan – read only <right> tuples
  - So, we need index
- + We want to avoid sorting – read <right> tuples in <right> order
  - So, we need special strategy to traverse index
- + We want to support tuples visibility
  - So, we should be able to resume index traverse
- + We want to support many data types
  - So, we need to modify GiST

Knn-search: Examples

- Synthetic data – randomly distributed points

```
create table qq ( id serial, p point, s int4);

insert into qq (p,s)  select point( p.lat, p.long),
(random()*1000)::int
from ( select  (0.5-random())*180 as lat, random()*360 as
long
       from ( select generate_series(1,1000000) ) as t
) as p;
create index qq_p_s_idx on qq using gist(p);
analyze qq;
```

- Query - find k-closest points to (0,0)

```
set enable_indexscan=on|off;
explain (analyze on, buffers on)
  select * from qq order by (p <-> '(0,0)') asc limit 10;
```

Knn-search: Examples

- postgresql.conf:

  shared_buffers = 512MB #32MB
  work_mem = 32MB #1MB
  maintenance_work_mem = 256MB #16MB
  checkpoint_segments = 16
  effective_cache_size = 1GB #128MB

- Index statistics (n=1000,000)

```
Number of levels:          3
Number of pages:           8787
Number of leaf pages:      8704
Number of tuples:          1008786
Number of invalid tuples:  0
Number of leaf tuples:     1000000
Total size of tuples:      44492028 bytes
Total size of leaf tuples: 44104448 bytes
Total size of index:       71983104 bytes
```

Knn-search: Examples

k=1, n=1,000,000

```
 Limit  (cost=0.00..0.08 rows=1 width=24) (actual time=0.104..0.104
rows=1 loops=1)
   Buffers: shared hit=4
   ->  Index Scan using qq_p_idx on qq  (cost=0.00..82060.60 rows=1000000
width=24) (actual time=0.104..0.104 rows=1 loops=1)
         Sort Cond: (p <-> '(0,0)'::point)
         Buffers: shared hit=4
 Total runtime: 0.117 ms
```

4000 times faster !

```
--------------------------
Limit  (cost=24853.00..24853.00 rows=1 width=24) (actual time=469.129..469.130
rows=1 loops=1)
   Buffers: shared hit=7353
   ->  Sort  (cost=24853.00..27353.00 rows=1000000 width=24) (actual
time=469.128..469.128 rows=1 loops=1)
         Sort Key: ((p <-> '(0,0)'::point))
         Sort Method:  top-N heapsort  Memory: 25kB
         Buffers: shared hit=7353
         ->  Seq Scan on qq  (cost=0.00..19853.00 rows=1000000 width=24)
(actual time=0.007..241.539 rows=1000000 loops=1)
               Buffers: shared hit=7353
 Total runtime: 469.150 ms
```

Knn-search: Examples

```
n=1000,000
```

k	:hit	:knn	: seq	:sortmem
1	:4	:0.117	:469.150	: 25
10	:17	:0.289	:471.735	: 25
100	:118	:0.872	:468.244	: 32
1000	:1099	:7.107	:473.840	: 127
10000	:10234	:31.629	:525.557	: 1550
100000	:101159	:321.182	:994.925	: 13957

Knn-search: Examples

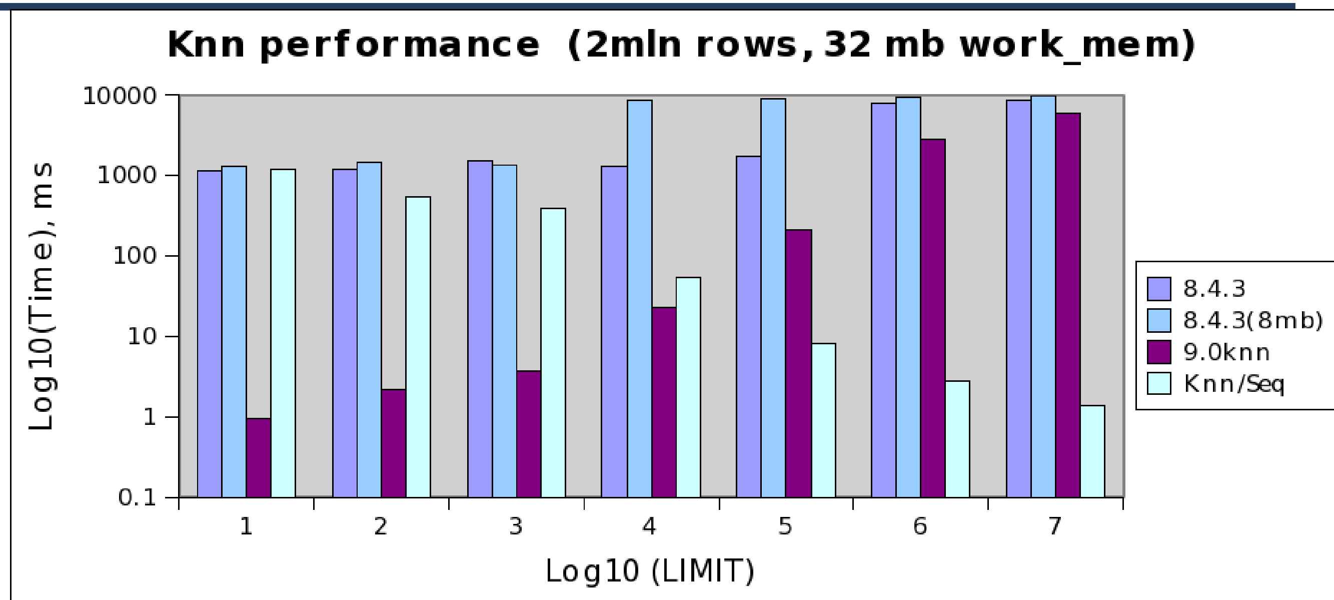
```
n=10,000
```

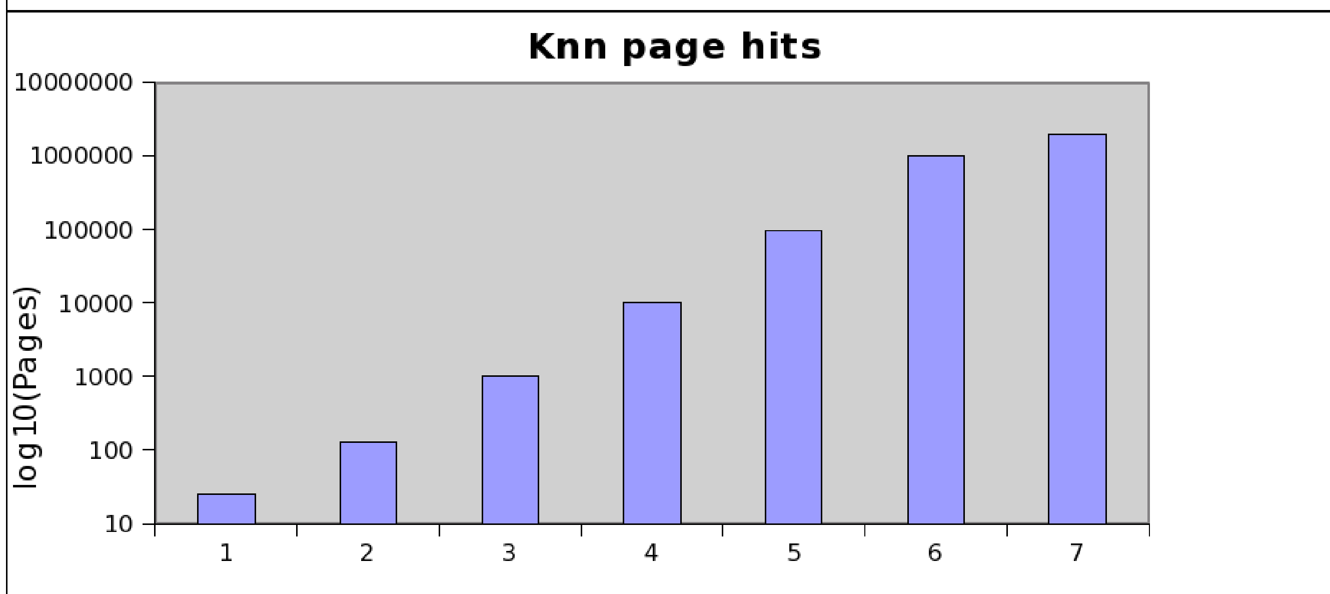
K	:hit	:knn	: seq
1	:3	:0.117	:6.072
10	:13	:0.247	:5.014
100	:103	:0.295	:6.381
1000	:996	:1.605	:8.670
10000	:9916	:16.487	:14.706

-> knn lose if k=n, n is small

Knn-search: Examples

- Real data
  2 mln points
  US, geonames





Knn-search: Examples

- Query: find 10 closest points in US with 'mars' in names to the point (5,5) - create composite index:

```
create index pt_fts_idx on geo using gist(point, to_tsvector('english',asciiname));

=# explain (analyze on, buffers on) select asciiname,point, (point <->
'5.0,5.0'::point) as dist from geo  where to_tsvector('english', asciiname)
@@ to_tsquery('english','mars') order by dist asc limit 10;
                                                            QUERY PLAN
---------------------------------------------------------------------------------------
 Limit  (cost=0.00..33.55 rows=10 width=35) (actual time=0.452..0.597 rows=10 loops=1)
   Buffers: shared hit=56
   ->  Index Scan using pt_fts_idx on geo  (cost=0.00..34313.91 rows=10227 width=35)
(actual time=0.452..0.592 rows=10 loops=1)
         Index Cond: (to_tsvector('english'::regconfig, (asciiname)::text) @@
'''mar'''::tsquery)
         Sort Cond: (point <-> '(5,5)'::point)
         Buffers: shared hit=56
 Total runtime: 0.629 ms
(7 rows)
```

Knn-search: Existing solutions

```
knn=# select id, date, event from events order by date <-> '1957-10-04'::date asc limit 10;
   id   |    date    |                                 event
--------+------------+-----------------------------------------------------------------
  58137 | 1957-10-04 | U.S.S.R. launches Sputnik I, 1st artificial Earth satellite
  58136 | 1957-10-04 | "Leave It to Beaver," debuts on CBS
 117062 | 1957-10-04 | Gregory T Linteris, Demarest, New Jersey, astronaut, sk: STS 83
 117061 | 1957-10-04 | Christina Smith, born in Miami, Florida, playmate, Mar, 1978
 102670 | 1957-10-05 | Larry Saumell, jockey
  31456 | 1957-10-03 | Willy Brandt elected mayor of West Berlin
  58291 | 1957-10-05 | 12th Ryder Cup: Britain-Ireland, 7 -4 at Lindrick GC, England
  58290 | 1957-10-05 | 11th NHL All-Star Game: All-Stars beat Montreal 5-3 at Montreal
  58292 | 1957-10-05 | Yugoslav dissident Milovan Djilos sentenced to 7 years
 102669 | 1957-10-05 | Jeanne Evert, tennis player, Chris' sister
(10 rows)

Time: 115.548 ms
```

- Very inefficient:
  - Full table scan, btree index on date won't help.
  - Sort full table

Knn-search: Existing solutions

contrib/btree_gist

```
knn=# select id, date, event from events order by date <-> '1957-10-04'::date asc
limit 10;
   id   |    date    |                                 event
--------+------------+-----------------------------------------------------------------
  58137 | 1957-10-04 | U.S.S.R. launches Sputnik I, 1st artificial Earth satellite
  58136 | 1957-10-04 | "Leave It to Beaver," debuts on CBS
 117062 | 1957-10-04 | Gregory T Linteris, Demarest, New Jersey, astronaut, sk: STS 83
 117061 | 1957-10-04 | Christina Smith, born in Miami, Florida, playmate, Mar, 1978
 102670 | 1957-10-05 | Larry Saumell, jockey
  31456 | 1957-10-03 | Willy Brandt elected mayor of West Berlin
  58291 | 1957-10-05 | 12th Ryder Cup: Britain-Ireland, 7 -4 at Lindrick GC, England
  58290 | 1957-10-05 | 11th NHL All-Star Game: All-Stars beat Montreal 5-3 at Montreal
  58292 | 1957-10-05 | Yugoslav dissident Milovan Djilos sentenced to 7 years
 102669 | 1957-10-05 | Jeanne Evert, tennis player, Chris' sister
(10 rows)

Time: 0.590 ms
```

- Very inefficient:
  - 8 index pages read + 10 tuples read
  - NO sorting
  - About 200 times faster !

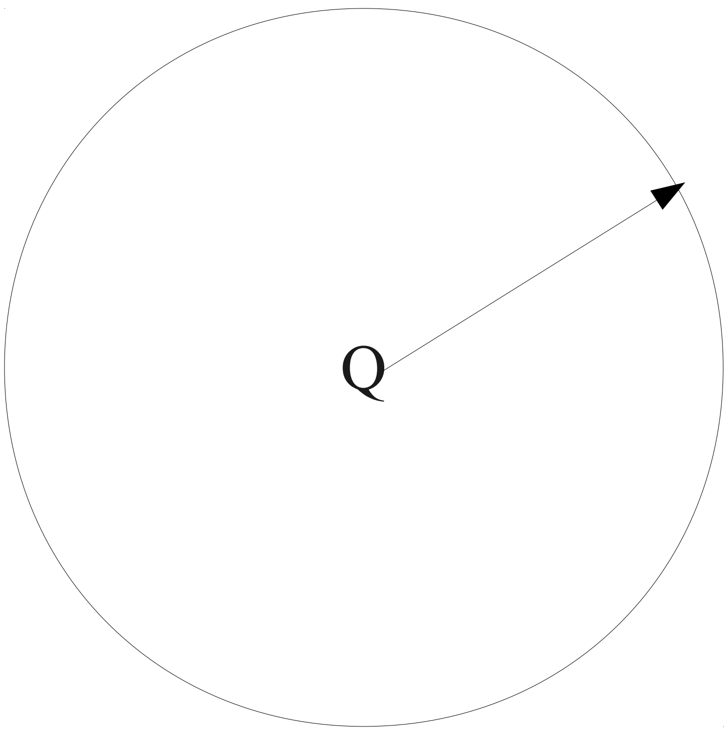
Knn-search: Examples

- pg_trgm support – distance = 1 – Similarity

```
knn=# select date, event, ('jeorge ewashington'  <-> event ) as dist
from events order by dist asc limit 10;

    date    |                      event                      |   dist
------------+-------------------------------------------------+----------
 1732-02-11 | George Washington                               | 0.458333
 1792-12-05 | George Washington re-elected U.S. pres          | 0.674419
 1811-02-23 | George Washington Hewitt, composer              |    0.675
 1753-08-04 | George Washington becomes a master mason        | 0.697674
 1941-07-19 | Jennifer Dunn, Rep-R-Washington                 | 0.710526
 1945-05-12 | Jayotis Washington, rocker                      | 0.714286
 1817-05-05 | George Washington Julian, MC, Union, died in 1899 |  0.72549
 1789-08-25 | Mary Ball Washington, mother of George, dies    | 0.729167
 1844-01-12 | George Washington Cable, American Novelist      | 0.729167
 1925-01-31 | George Washington Cable, American Novelist      | 0.729167
(10 rows)

Time: 187.604 ms
```

Knn-search: Examples

- Corner case for knn-search - all data are on the same distance from point Q !



Knn-search: Examples

- Corner case for Best First Strategy - all data are on the same distance from point Q !

```
create table circle (id serial, p point, s int4);
insert into circle (p,s)
    select point( p.x, p.y), (random()*1000)::int
    from ( select t.x, sqrt(1- t.x*t.x) as y
          from ( select random() as x, generate_series(1,1000000) ) as t
) as p;
create index circle_p_idx on circle using gist(p);
analyze circle;
```

Number of levels: 3
Number of pages: 8266
Number of leaf pages: 8201

Knn-search: Examples

- Corner case for knn-search - all data are on the same distance from point Q !

```
=# explain (analyze on, buffers on) select * from circle
                           order by (p <-> '(0,0)') asc limit 10;


 Limit  (cost=0.00..0.80 rows=10 width=24) (actual time=226.907..226.924
rows=10 loops=1)
   Buffers: shared hit=8276
   ->  Index Scan using circle_p_idx on circle  (cost=0.00..79976.58
rows=1000000 width=24) (actual time=226.905..226.921 rows=10 loops=1)
         Sort Cond: (p <-> '(0,0)'::point)
         Buffers: shared hit=8276 - read all index
 Total runtime: 230.885 ms
```

- Still 2 times faster than SEQ (454.331 ms) because of sorting

Bloom index
(prototype)

Bloom index (prototype)

- Data with many attributes
- Too many indexes to support queries, which uses aribitrary combinations of attributes – (a,b,c), (b,c,a), (c,a,b), (c,b,a)...
  - Space usage
  - Slow update
- Equality queries (a = 2)
- Idea - hash all attributes to a bit-signature of fixed sized
  - Store signatures in a file
  - To search read full file (sequentually)
  - Search performance is constant O(N), insert O(1)

Bloom index

```
 Id  | nick   | email              | name   | age | ....
-----+--------+--------------------+--------+-----+-----
 122 | teodor | teodor@sigaev.ru   | Teodor | 37  | ....
```

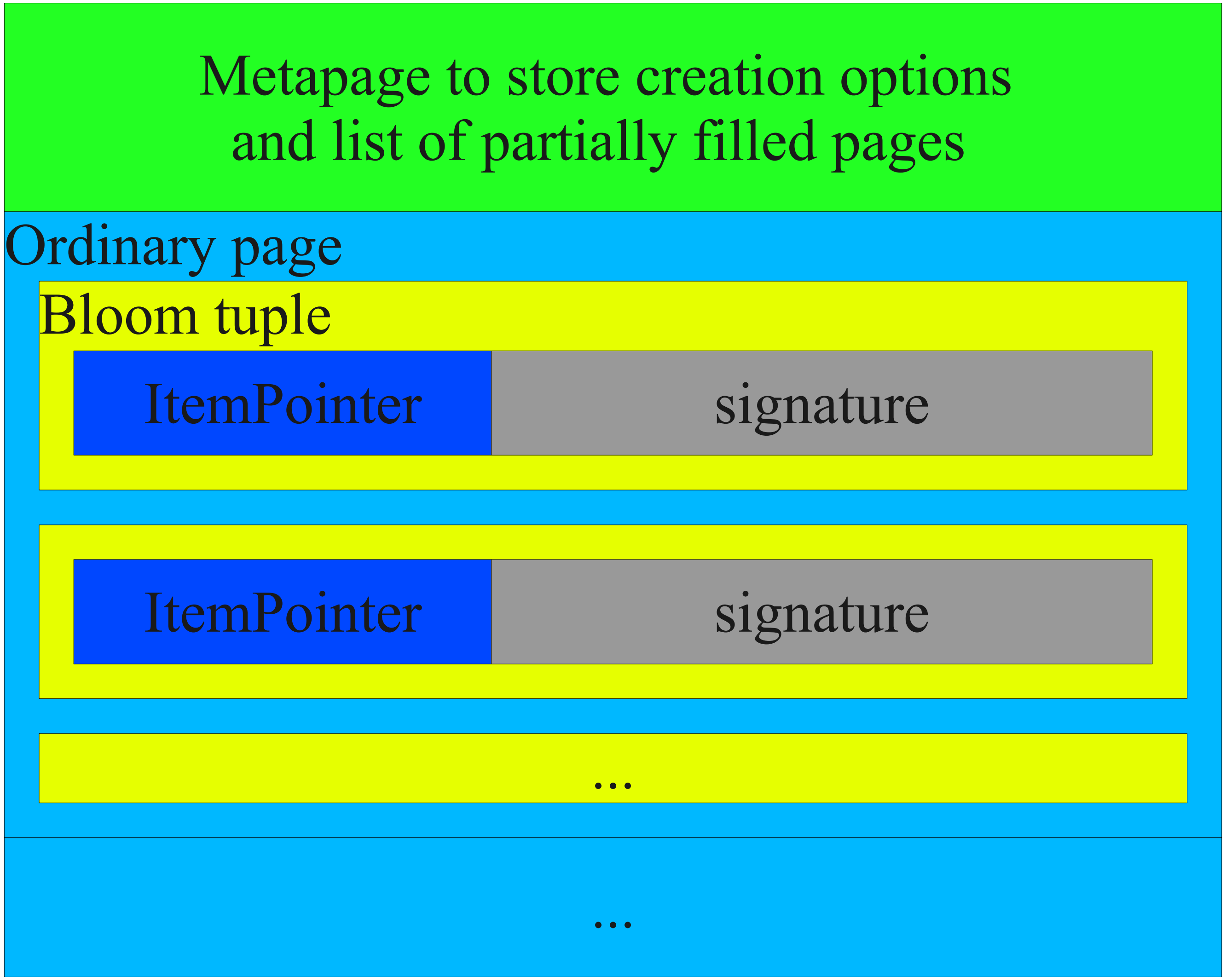
St = 010010001000000010101000100000000010101

Sq = 000000000000000000001000100000000010101

SELECT … WHERE name = 'Teodor' AND age = 37

St & Sq == Sq

Bloom index


Metapage to store creation options
and list of partially filled pages
Ordinary page
Bloom tuple
ItemPointer
signature
ItemPointer
signature
...
...

Bloom index

- Index scan is a sequentual scan of index
- Index is rather small
- Insert ~ O(1), Search ~ O(N)

Bloom index

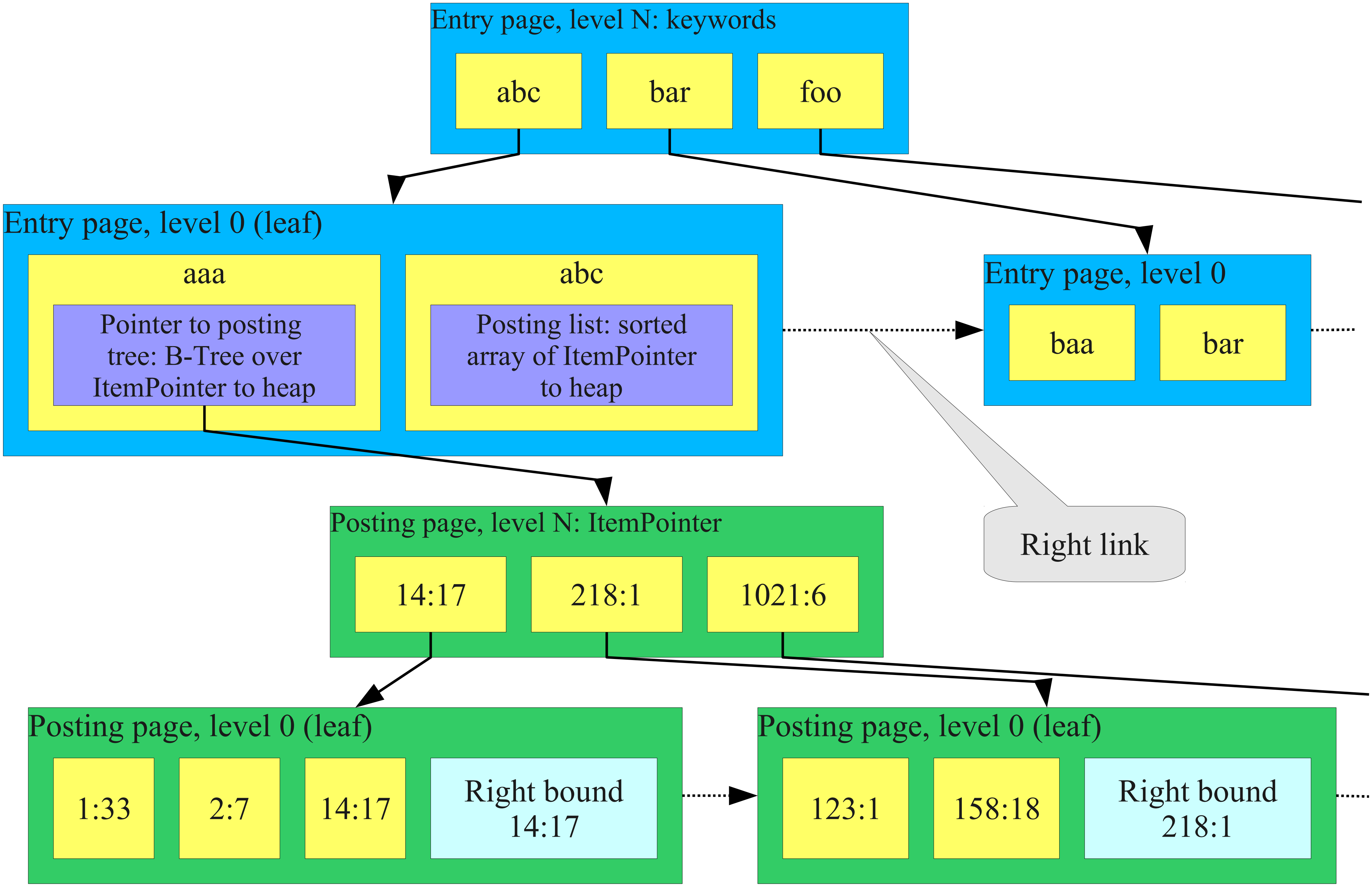
```
CREATE INDEX bloomidx ON tbloom(i1,i2,i3)
       WITH (length=5, col1=2, col2=2, col3=4);
```

- `length` – number of uint16 in signature (ItemPointer is 6 bytes long, so just an aligment)
- `colN` – number of bits for column N
- It's a Prototype!

Better cost estimation of GIN index scan

gincostestimate

- The problem:
  http://archives.postgresql.org/pgsql-performance/2009-10/msg00393.php
  - planner chooses sequential scan instead of index scan, which hurt fts users
  - Current cost of GIN index scan is very over-estimated selectivity * pg_class.relpages
  - GIN index is different from normal indexes (it's inverted index) and consists of
    - ENTRY Tree – store entries
    - POSTING List or Tree – store ItemPointers to heap


Entry page, level N: keywords
abc
bar
foo
Entry page, level 0 (leaf)
aaa
Pointer to posting tree: B-Tree over ItemPointer to heap
abc
Posting list: sorted array of ItemPointer to heap
Entry page, level 0
baa
bar
Right link
Posting page, level N: ItemPointer
14:17
218:1
1021:6
Posting page, level 0 (leaf)
1:33
2:7
14:17
Right bound 14:17
Posting page, level 0 (leaf)
123:1
158:18
Right bound 218:1

gincostestimate

- `SELECT … WHERE ts @@ 'foo & bar'::tsquery`
- Search query should be processed (by extractQuery) to get entries. For each entry:
  - Calculate cost of search in ENTRY tree
  - Calculate cost of reading POSTING list or tree

gincostestimate

Cost of search in entry tree

- Need to know depth of tree, could be estimated using number of pages in entry tree (`pg_class.relentrypages`)
- Partial match (prefix search for tsquery 'foo:*') :(But it doesn't need to search – just a scan on leaf pages

gincostestimate

Cost of posting lists/trees reading (never search)

- No stats per entry, estimate DataPageEstimate as
  `(pg_class.relpages - pg_class.relentrypages) / pg_class.relentries`
- For partial match multiply this estimation by constant (100)
- For frequent entry DataPageEstimate can be under-estimated

  Hack:
  DataPageEstimate = max(
  `selectivity * (pg_class.relpages-pg_class.relentrypages),`
  DataPageEstimate)

Gincostestimate: problems

- Where to store relentrypages & relentries, in pg_class ?
- How to update them
  - VACUUM and CREATE INDEX – ok
  - INSERT has no interface to update pg_class
  - INSERT doesn't produces dead tuples, so vacuum will do nothing with indexes

References

- KNN (patch -l)
  - http://www.sigaev.ru/misc/builtin_knngist_itself-0.7.gz
  - http://www.sigaev.ru/misc/builtin_knngist_contrib_btree_gist-0.6.gz
  - http://www.sigaev.ru/misc/builtin_knngist_contrib_pg_trgm-0.6.gz
  - http://www.sigaev.ru/misc/builtin_knngist_planner-0.6.gz
  - http://www.sigaev.ru/misc/builtin_knngist_proc-0.6.gz
- Bloom index
  - http://www.sigaev.ru/misc/bloom-0.3.tar.gz
- GIN cost estimate
  - http://www.sigaev.ru/misc/gincostestimate-0.17.gz