Полнотекстовый поиск в PostgreSQL за миллисекунды

Олег Бартунов

Александр Коротков

Профессиональная конференция разработчиков высоконагруженных систем

HighLoad 2012

Full-text search in DB

- **Full-text search**
  - Find documents, which *satisfy* query
  - return results in some order (opt.)
- **Requirements to FTS**
  - **Full integration with DB core**
    - transaction support
    - concurrency and recovery
    - online index
  - **Linguistic support**
  - **Flexibility, Scalability**

What is a document ?

- Arbitrary textual attribute
- Combination of textual attributes
- Could be fully virtual
- It's a textual result of any SQL, for example – join of `doc` and `authors` tables

Title || Abstract || Keywords || Body || Author

Text Search Operators

- Traditional FTS operators for textual attributes
  ~, ~*, LIKE, ILIKE

Problems

– No linguistic support, no stop-words
– No ranking
– Sequential scan all documents, slow

Solution

– Preprocess document in advance
– Add index support

FTS in PostgreSQL

- set of rules how document and query should be transformed to their FTS representations – tsvector, tsquery
- set of functions to obtain tsvector, tsquery from textual data types
- FTS operators and indexes
- ranking functions, headline

FTS in PostgreSQL

```
=# select  'a fat cat sat on a mat and ate a fat rat'::tsvector
                                @@
                  'cat & rat':: tsquery;
```

- **tsvector** – storage for document, optimized for search
  - sorted array of lexemes
  - positional information
  - weights information
- **tsquery** – textual data type for query
  - Boolean operators - & | ! ()
- **FTS operator**

  tsvector @@ tsquery

FTS features

- Full integration with PostgreSQL
- 27 built-in configurations for 10 languages
- Support of user-defined FTS configurations
- Pluggable dictionaries (ispell, snowball, thesaurus), parsers
- Relevance ranking
- GiST and GIN indexes with concurrency and recovery support
- Rich query language with query rewriting support

FTS in PostgreSQL

- OpenFTS — 2000, Pg as a storage
- GiST index — 2000, thanks Rambler
- Tsearch — 2001, contrib:no ranking
- Tsearch2 — 2003, contrib:config
- GIN —2006, thanks, JFG Networks
- FTS — 2006, in-core, thanks,EnterpriseDB
- E-FTS — Enterprise FTS, thanks ???

ACID overhead is really big :(

- Foreign solutions: Sphinx, Solr, Lucene....
  - Crawl database and index (time lag)
  - No access to attributes
  - Additional complexity
  - BUT: **Very fast !**
- **Can we improve native FTS ?**

Can we improve native FTS ?

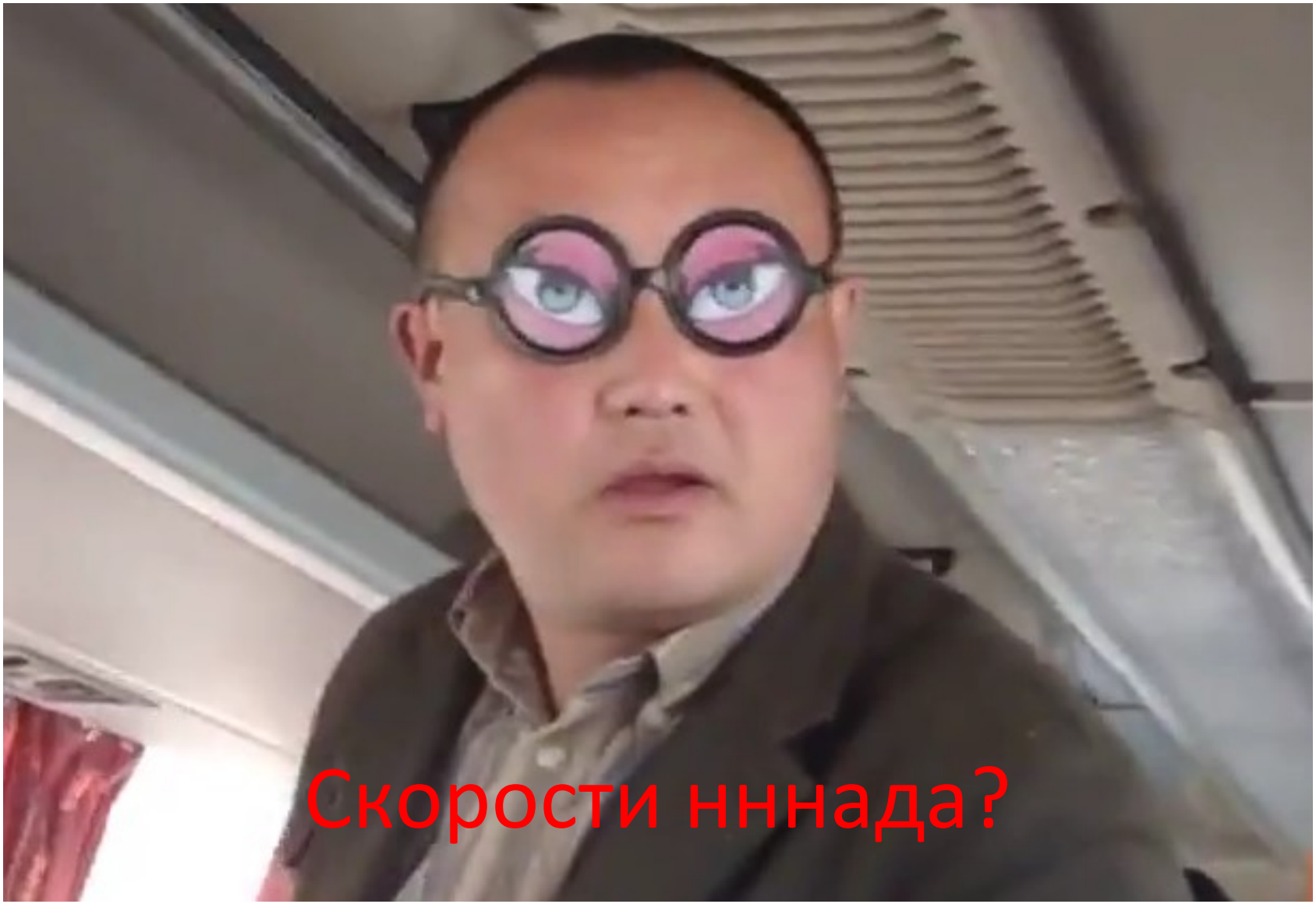
1. Find relevant documents
    1. Index scan — usually very fast
2. Calculate ranks for all founded documents
    1. Heap scan — usually slow
3. Sort documents

Can we improve native FTS ?

156676 Wikipedia articles:

```
postgres=# explain analyze
SELECT docid, ts_rank(text_vector, to_tsquery('english', 'title')) AS rank
FROM ti2
WHERE text_vector @@ to_tsquery('english', 'title')
ORDER BY rank DESC
LIMIT 3;
```

```
Limit  (cost=8087.40..8087.41 rows=3 width=282) (actual time=433.750..433.752 rows=
  ->  Sort  (cost=8087.40..8206.63 rows=47692 width=282) (actual time=433.749..433
        Sort Key: (ts_rank(text_vector, '''titl'''::tsquery))
        Sort Method: top-N heapsort  Memory: 25kB
        ->  Bitmap Heap Scan on ti2  (cost=529.61..7470.99 rows=47692 width=282) (
              Recheck Cond: (text_vector @@ '''titl'''::tsquery)
              ->  Bitmap Index Scan on ti2_index  (cost=0.00..517.69 rows=47692 wi
                    Index Cond: (text_vector @@ '''titl'''::tsquery)
Total runtime: 433.787 ms
```


Скорости ннада?

Can we improve native FTS ?

156676 Wikipedia articles:

```
postgres=# explain analyze
SELECT docid, ts_rank(text_vector, to_tsquery('english', 'title')) AS rank
FROM ti2
WHERE text_vector @@ to_tsquery('english', 'title')
ORDER BY text_vector>< plainto_tsquery('english','title')
LIMIT 3;
```

What if we have this plan ?

```
Limit  (cost=20.00..21.65 rows=3 width=282) (actual time=18.376..18.427 rows=3 loop
   ->  Index Scan using ti2_index on ti2  (cost=20.00..26256.30 rows=47692 width=28
         Index Cond: (text_vector @@ '''titl'''::tsquery)
         Order By: (text_vector >< '''titl'''::tsquery)
 Total runtime: 18.511 ms
```

We'll be FINE !

We'll be FINE !

- Teach index (GIN) to calculate ranks and returns results in sorted order
  - Store positions in index — no need for tsvector column, use compression to keep index small
  - Change algorithms and interfaces

We'll be FINE !

- Additional benefit
  - T(rare_word & frequent_word) ~ T(rare_word)

Inverted Index

Report Index

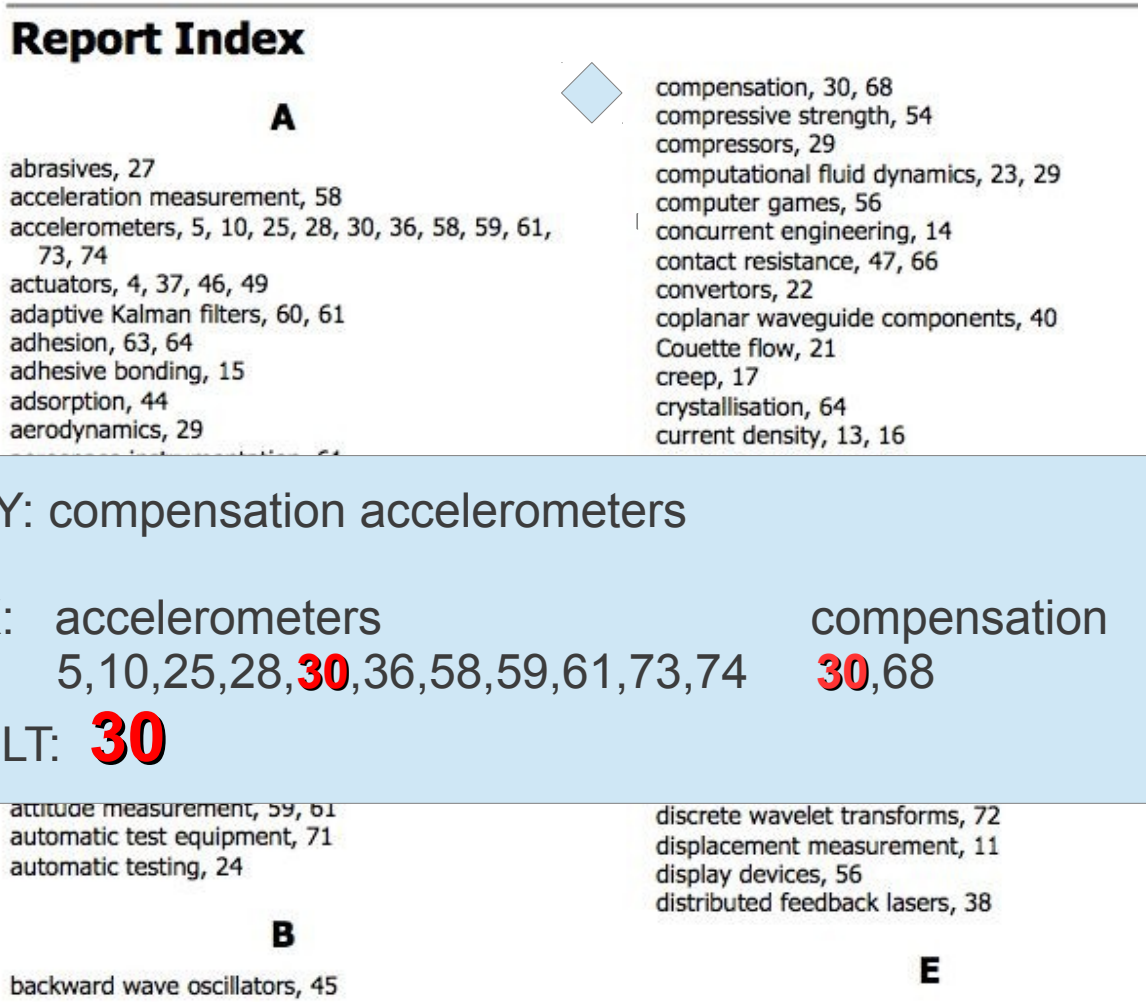
A

B

D

E

Inverted Index



Report Index

A

abrasives, 27
acceleration measurement, 58
accelerometers, 5, 10, 25, 28, 30, 36, 58, 59, 61, 73, 74
actuators, 4, 37, 46, 49
adaptive Kalman filters, 60, 61
adhesion, 63, 64
adhesive bonding, 15
adsorption, 44
aerodynamics, 29

compensation, 30, 68
compressive strength, 54
compressors, 29
computational fluid dynamics, 23, 29
computer games, 56
concurrent engineering, 14
contact resistance, 47, 66
convertors, 22
coplanar waveguide components, 40
Couette flow, 21
creep, 17
crystallisation, 64
current density, 13, 16

QUERY: compensation accelerometers

INDEX: accelerometers 5,10,25,28,**30**,36,58,59,61,73,74

compensation **30**,68

RESULT: **30**

attitude measurement, 59, 61
automatic test equipment, 71
automatic testing, 24

discrete wavelet transforms, 72
displacement measurement, 11
display devices, 56
distributed feedback lasers, 38

B

backward wave oscillators, 45

E

No positions in index !

Inverted Index in PostgreSQL

ENTRY TREE

Report Index

A

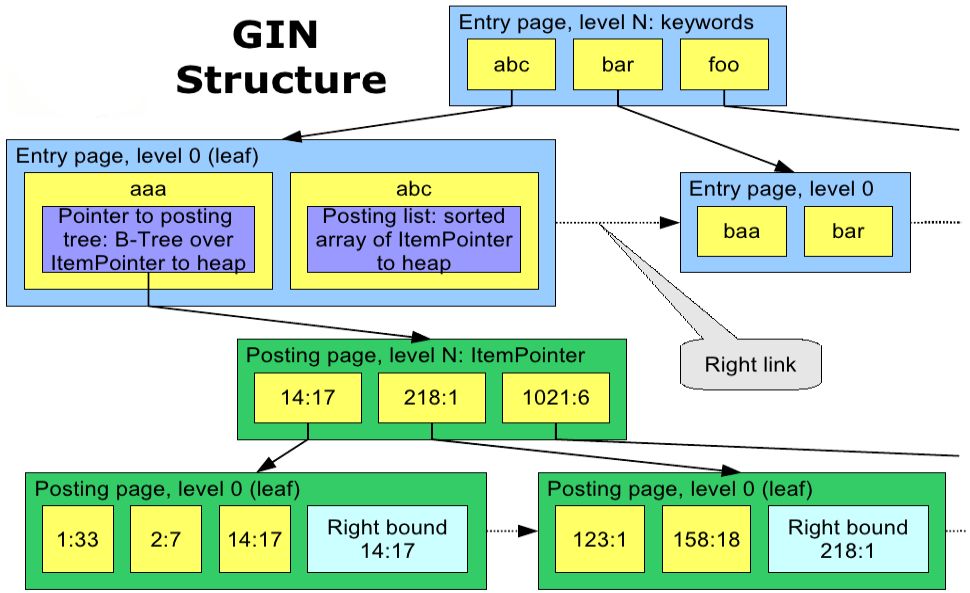
abrasives, 27
acceleration measurement, 58
accelerometers, 5, 10, 25, 28, 30, 36, 58, 59, 61, 73, 74
actuators, 4, 37, 46, 49
adaptive Kalman filters, 60, 61
adhesion, 63, 64
adhesive bonding, 15
adsorption, 44
aerodynamics, 29
aerospace instrumentation, 6
aerospace propulsion, 52
aerospace robotics, 68
aluminium, 17
amorphous state, 67
angular velocity measurement
antenna phased arrays, 41, 4
argon, 21
assembling, 22
atomic force microscopy, 13,
atomic layer deposition, 15
attitude control, 60, 61
attitude measurement, 59, 61
automatic test equipment, 71
automatic testing, 24

B

backward wave oscillators, 45

compensation, 30, 68
compressive strength, 54
compressors, 29
computational fluid dynamics, 23, 29
computer games, 56
concurrent engineering, 14
contact resistance, 47, 66
convertors, 22
coplanar waveguide components, 40
Couette flow, 21
creep, 17
crystallisation, 64

Posting list
Posting tree



HighLoad 2012
Профессиональная конференция разработчиков высоконагруженных систем

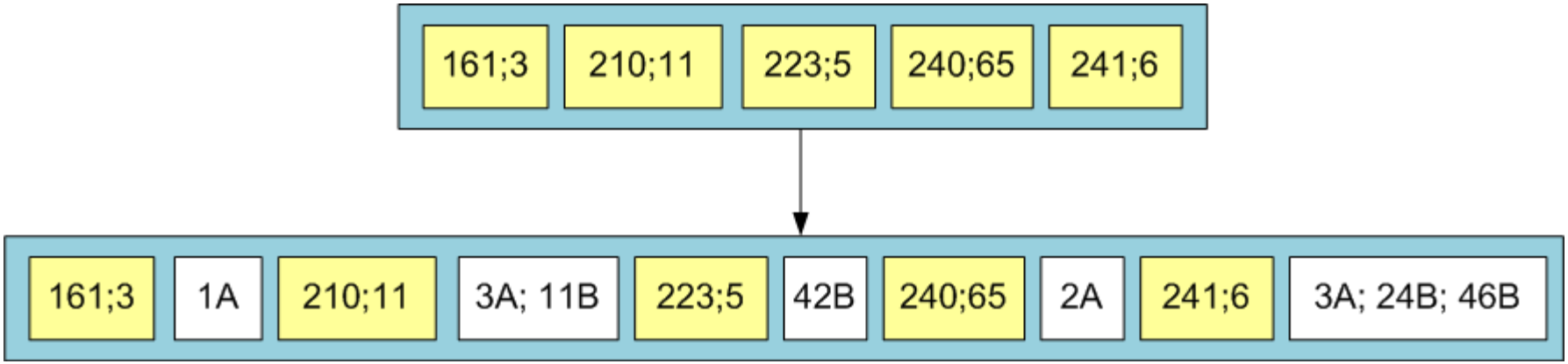
Summary of changes

- GIN
  - storage
  - search
  - ORDER BY
  - interface
- planner

GIN structure changes

hl++ HighLoad 2012
Профессиональная конференция разработчиков высоконагруженных систем

Add additional information (word positions)



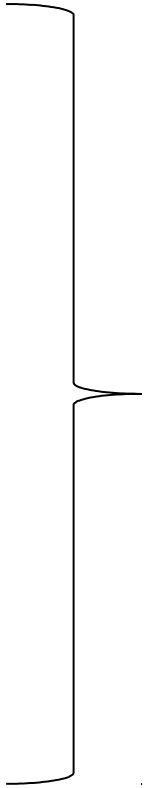
ItemPointer

```
typedef struct ItemPointerData
{
   BlockIdData ip_blkid;
   OffsetNumber ip_posid;
}

typedef struct BlockIdData
{
   uint16    bi_hi;
   uint16    bi_lo;
} BlockIdData;
```

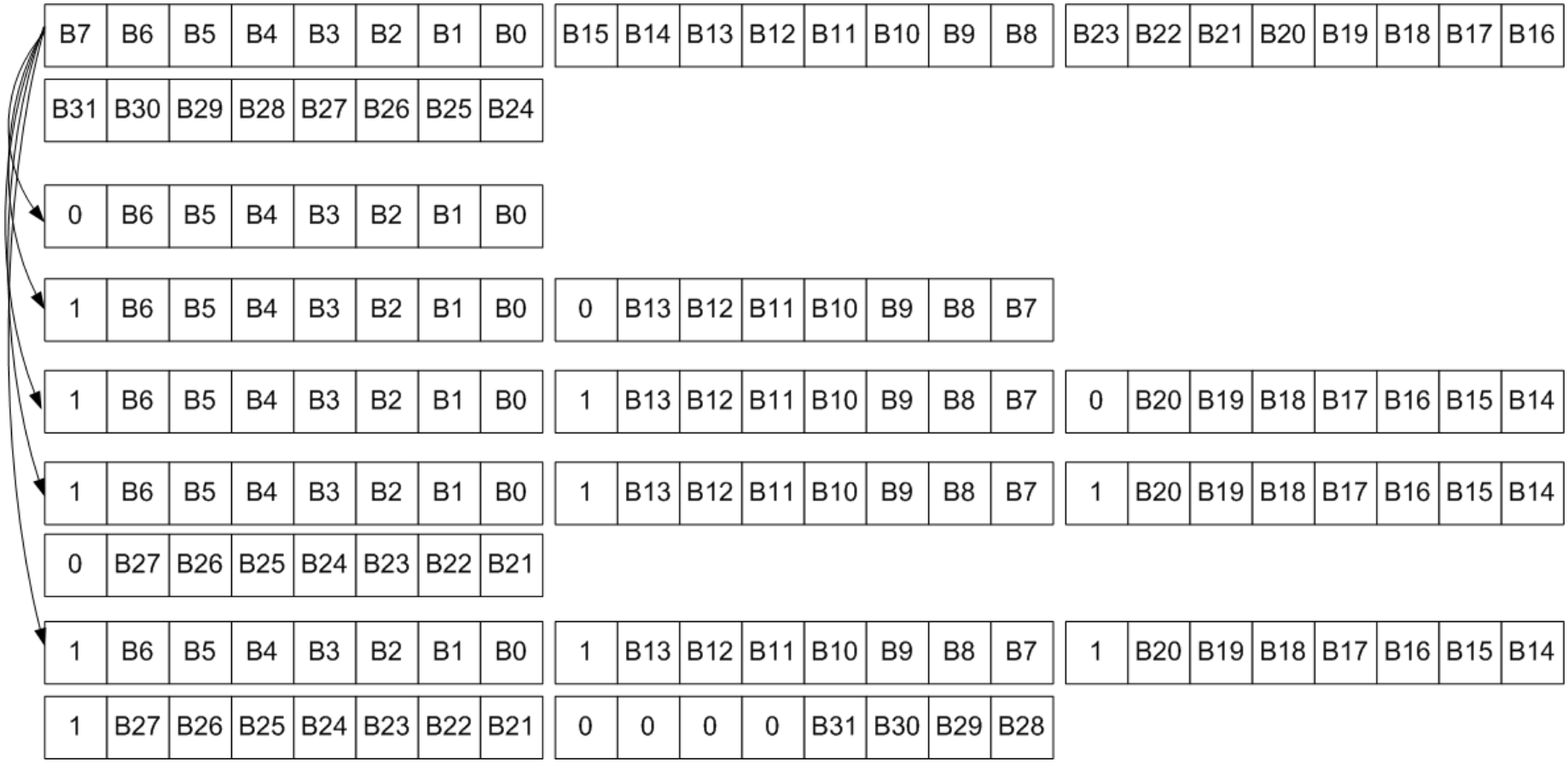
6 bytes

WordEntryPos

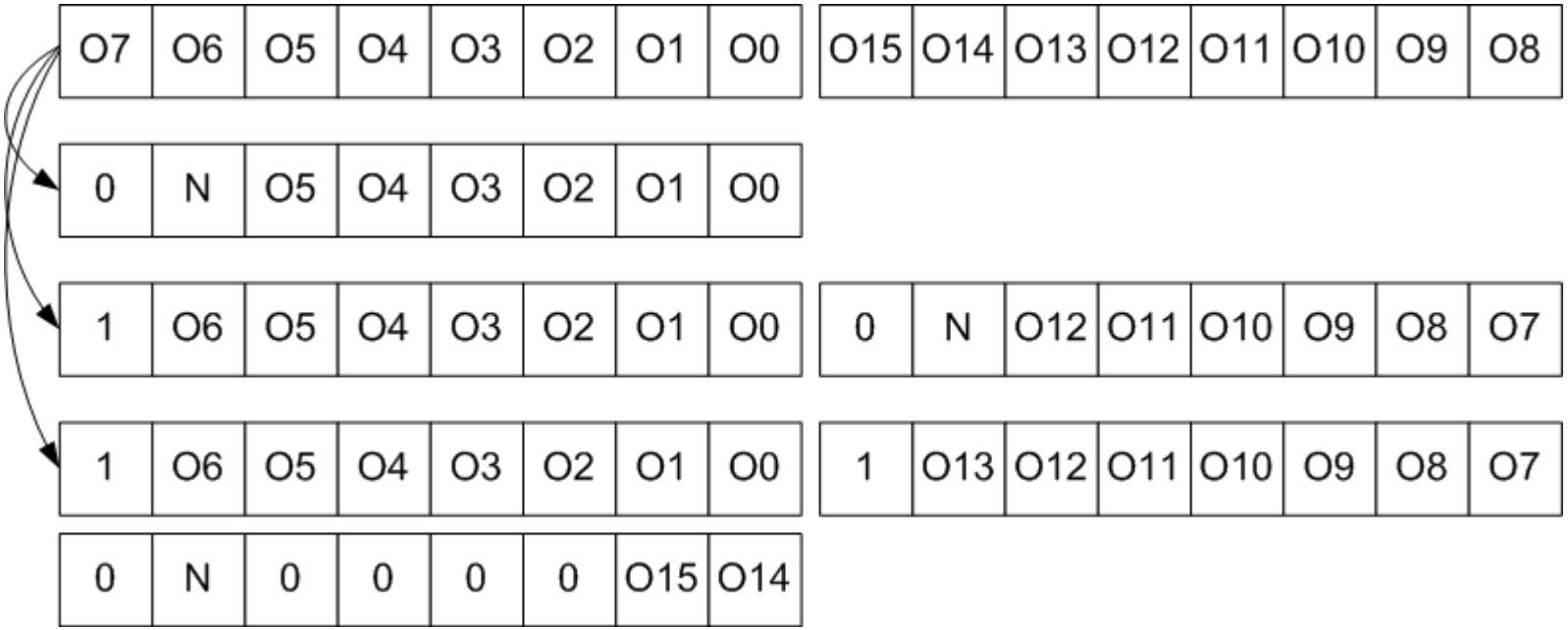
```
/*
 * Equivalent to
 * typedef struct {
 *   uint16
 *      weight:2,
 *      pos:14;
 * }
 */

typedef uint16 WordEntryPos;
```



BlockIdData compression



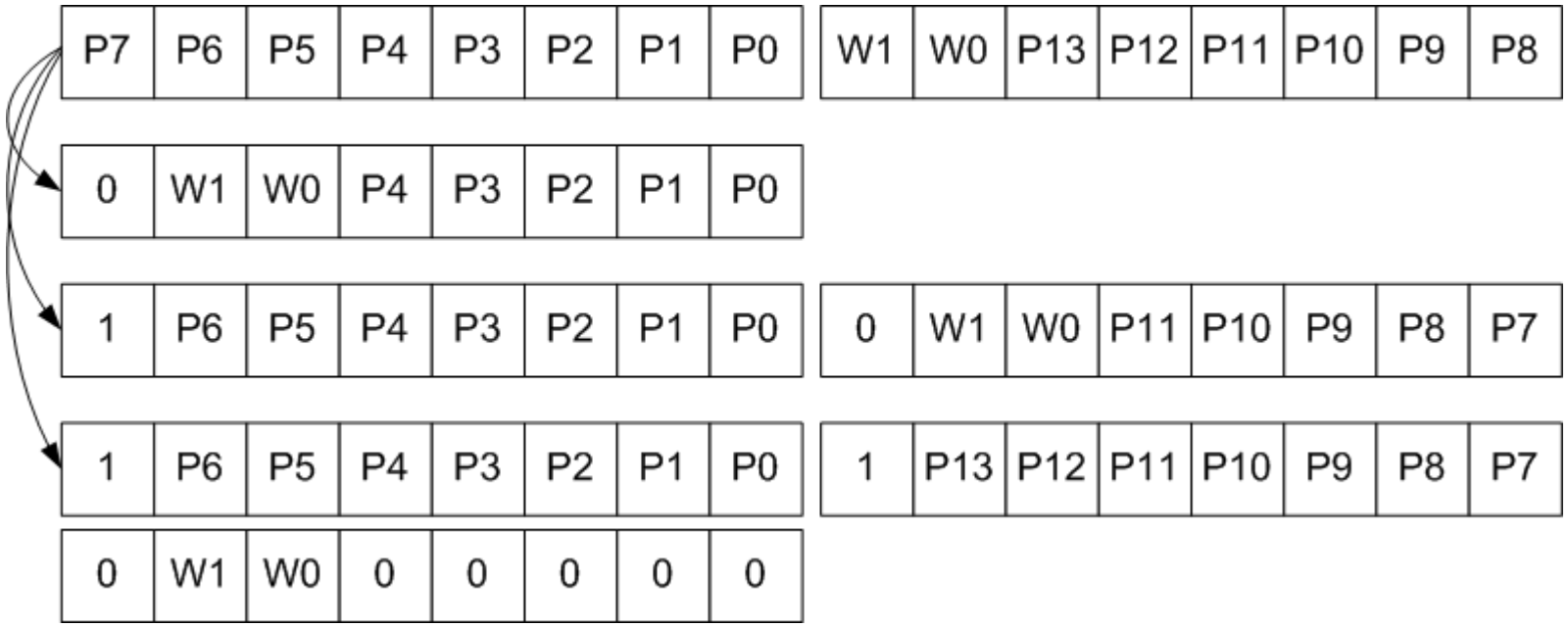
OffsetNumber compression



O0-O15 – OffsetNumber bits

N – Additional information NULL bit

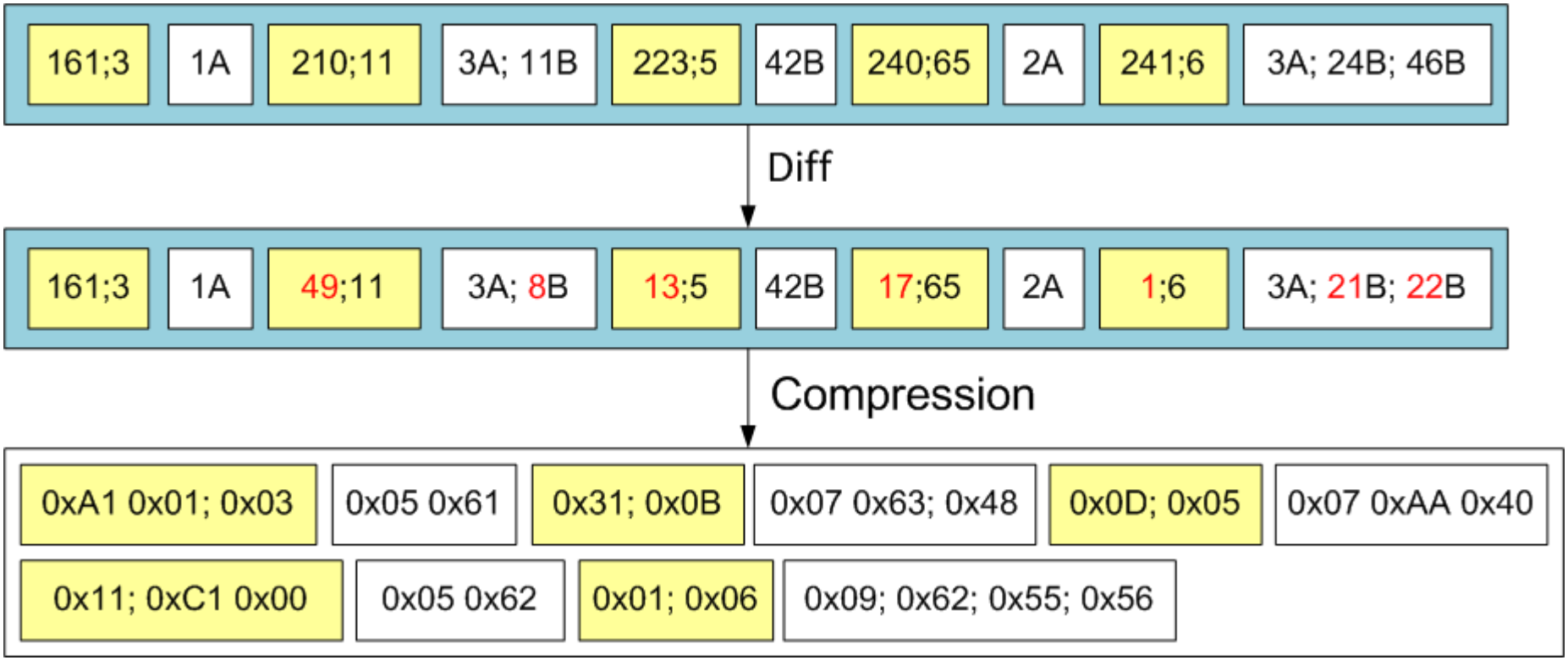
WordEntryPos compression



P0-P13 – position bits

W0,W1 – weight bits

Example


161;3
1A
210;11
3A; 11B
223;5
42B
240;65
2A
241;6
3A; 24B; 46B
Diff
161;3
1A
49;11
3A; 8B
13;5
42B
17;65
2A
1;6
3A; 21B; 22B
Compression
0xA1 0x01; 0x03
0x05 0x61
0x31; 0x0B
0x07 0x63; 0x48
0x0D; 0x05
0x07 0xAA 0x40
0x11; 0xC1 0x00
0x05 0x62
0x01; 0x06
0x09; 0x62; 0x55; 0x56

GIN algorithm changes

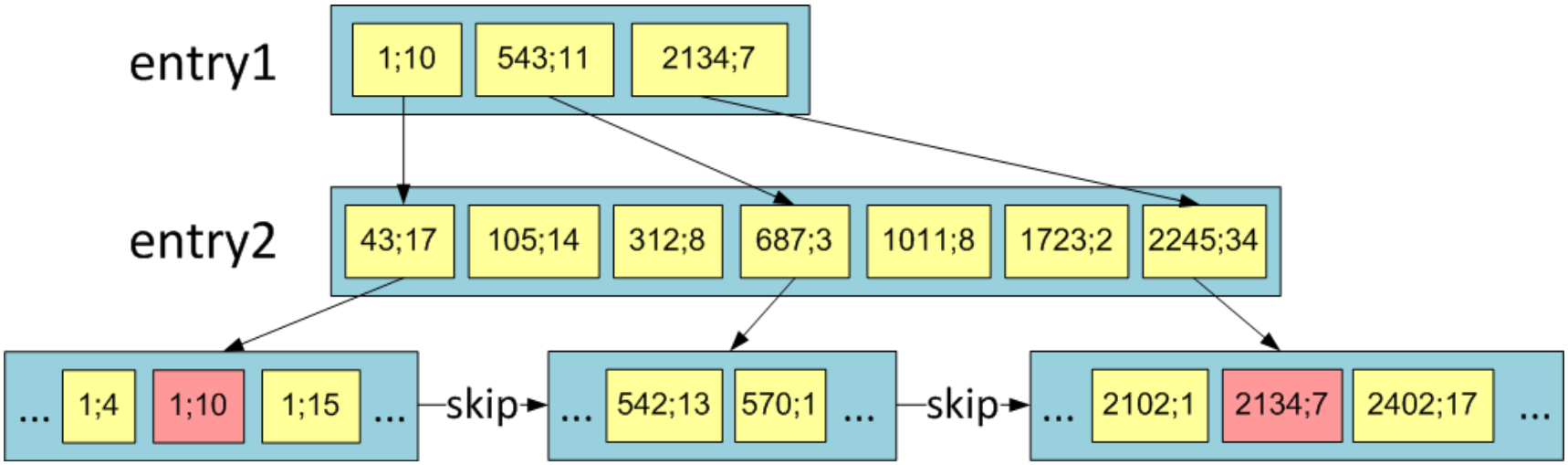
hl++ HighLoad 2012
Профессиональная конференция разработчиков высоконагруженных систем

Top-N queries

1. Scan + calc rank
2. Sort
3. Return using gingettuple one by one

Fast scan

entry1 && entry2



GIN interface changes

hl++ HighLoad 2012
Профессиональная конференция разработчиков высоконагруженных систем

extractValue

```
Datum *extractValue
(
    Datum itemValue,
    int32 *nkeys,
    bool **nullFlags,
    Datum *addInfo,
    bool *addInfoIsNull
)
```

extractQuery

```
Datum *extractValue
(
   Datum query,
   int32 *nkeys,
   StrategyNumber n,
   bool **pmatch,
   Pointer **extra_data,
   bool **nullFlags,
   int32 *searchMode,
   ???bool **required???
)
```

consistent

```
bool consistent
(
    bool check[],
    StrategyNumber n,
    Datum query,
    int32 nkeys,
    Pointer extra_data[],
    bool *recheck,
    Datum queryKeys[],
    bool nullFlags[],
    Datum addInfo[],
    bool addInfoIsNull[]
)
```

calcRank

```
float8 calcRank
(
   bool check[],
   StrategyNumber n,
   Datum query,
   int32 nkeys,
   Pointer extra_data[],
   bool *recheck,
   Datum queryKeys[],
   bool nullFlags[],
   Datum addInfo[],
   bool addInfoIsNull[]
)
```

???joinAddInfo???

```
Datum joinAddInfo
(
   Datum addInfos[]
)
```

Planner optimization

HighLoad 2012
Профессиональная конференция разработчиков высоконагруженных систем

Before

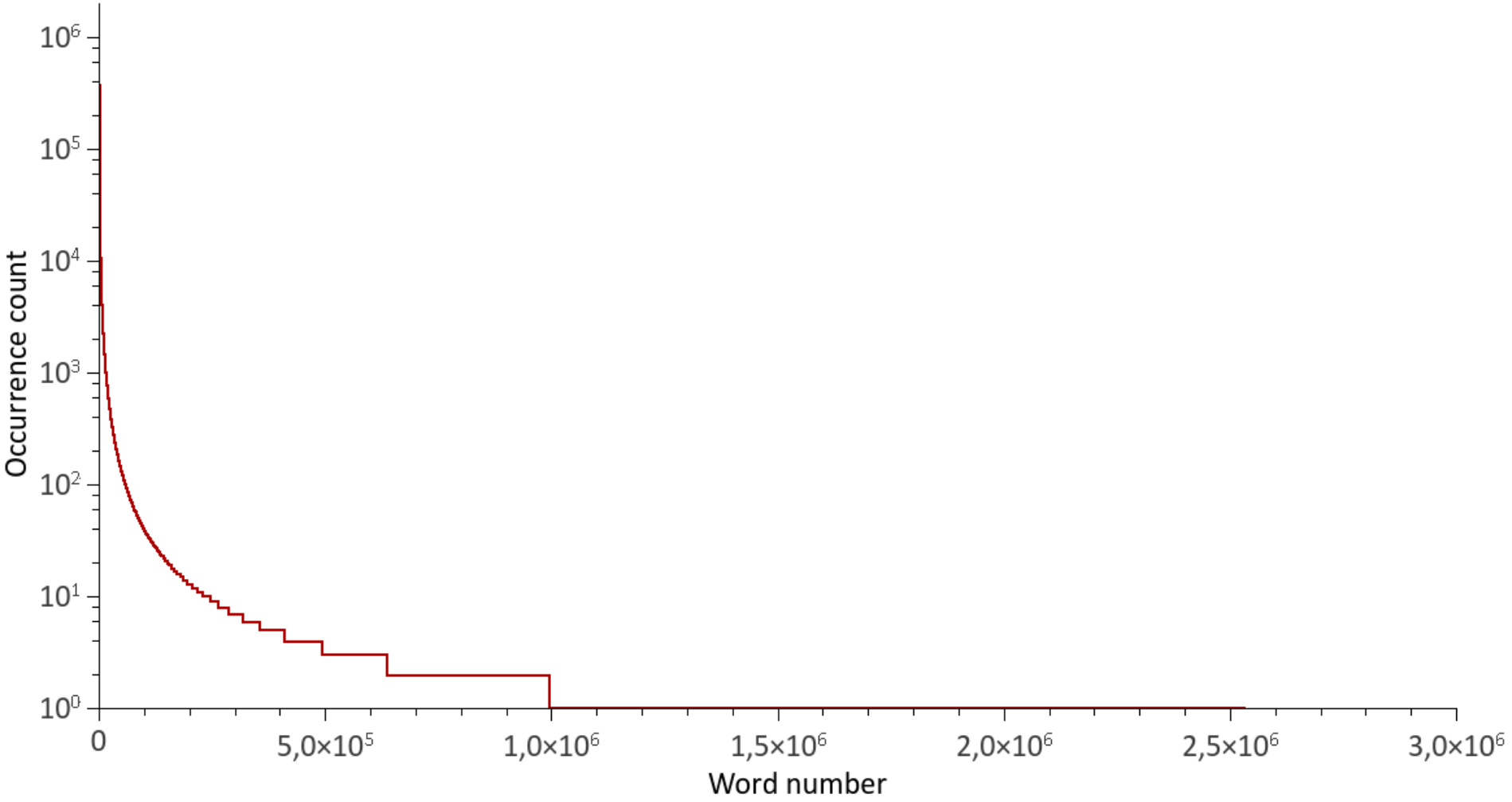
```
test=# EXPLAIN (ANALYZE, VERBOSE) SELECT * FROM test ORDER BY slow_func(x,y)
LIMIT 10;
                                                                  QUERY PLAN

-----------------------------------------------------------------------------------------------------------------------
 Limit  (cost=0.00..3.09 rows=10 width=16) (actual time=11.344..103.443 rows=10
loops=1)
   Output: x, y, (slow_func(x, y))
   ->  Index Scan using test_idx on public.test  (cost=0.00..309.25 rows=1000 width=16)
(actual time=11.341..103.422 rows=10 loops=1)
         Output: x, y, slow_func(x, y)
 Total runtime: 103.524 ms
(5 rows)
```

After

```
test=# EXPLAIN (ANALYZE, VERBOSE) SELECT * FROM test ORDER BY slow_func(x,y)
LIMIT 10;
                                                                  QUERY PLAN

-----------------------------------------------------------------------------------------------------------------------
 Limit  (cost=0.00..3.09 rows=10 width=16) (actual time=0.062..0.093 rows=10 loops=1)
   Output: x, y
   ->  Index Scan using test_idx on public.test  (cost=0.00..309.25 rows=1000 width=16)
(actual time=0.058..0.085 rows=10 loops=1)
         Output: x, y
 Total runtime: 0.164 ms
(5 rows)
```

Testing results

avito.ru: 6.7 mln. docs



With tsvector column

```
SELECT
    itemid, title
FROM
    items
WHERE
    fts @@ plainto_tsquery('russian', 'угловой
шкаф')
ORDER BY
    ts_rank(fts, plainto_tsquery('russian',
'угловой шкаф')) DESC
LIMIT
    10;
```

With tsvector column without patch,

```
Limit  (cost=2341.92..2341.94 rows=10 width=398) (actual time=38.532..38.5
   Buffers: shared hit=12830
   ->  Sort  (cost=2341.92..2343.37 rows=581 width=398) (actual time=38.53
         Sort Key: (ts_rank(fts, '''углов'' & ''шкаф'''::tsquery))
         Sort Method: top-N heapsort  Memory: 26kB
         Buffers: shared hit=12830
         ->  Bitmap Heap Scan on items  (cost=48.50..2329.36 rows=581 widt
               Recheck Cond: (fts @@ '''углов'' & ''шкаф'''::tsquery)
               Buffers: shared hit=12830
               ->  Bitmap Index Scan on fts_idx  (cost=0.00..48.36 rows=58
                     Index Cond: (fts @@ '''углов'' & ''шкаф'''::tsquery)
                     Buffers: shared hit=116
 Total runtime: 38.569 ms
```

With tsvector column with patch,

```
 Limit  (cost=40.00..80.28 rows=10 width=400) (actual
time=11.528..11.536
   Buffers: shared hit=374
   ->  Index Scan using fts_idx on items
(cost=40.00..2863.77 rows=701
       Index Cond: (fts @@ '''углов'' & ''шкаф'''::tsquery)
         Order By: (fts >< '''углов'' & ''шкаф'''::tsquery)
         Buffers: shared hit=374
 Total runtime: 11.561 ms
```

Without tsvector column

```
SELECT itemid, title
FROM items2
WHERE (setweight(to_tsvector('russian'::regconfig,
title), 'A'::"char") ||
setweight(to_tsvector( 'russian'::regconfig,
description), 'B'::"char")) @@
plainto_tsquery('russian', 'угловой шкаф')
ORDER BY
ts_rank((setweight(to_tsvector('russian'::regconfig,
title), 'A'::"char") ||
setweight(to_tsvector('russian'::regconfig,
description), 'B'::"char")), plainto_tsquery('russian',
'угловой шкаф')) DESC
LIMIT 10;
```

Without tsvector column, without patch

```
Limit  (cost=2596.31..2596.33 rows=10 width=372) (actual time=862.520..86
  Buffers: shared hit=6875
  ->  Sort  (cost=2596.31..2597.91 rows=642 width=372) (actual time=862.5
        Sort Key: (ts_rank((setweight(to_tsvector('russian'::regconfig,
        Sort Method: top-N heapsort  Memory: 26kB
        Buffers: shared hit=6875
        ->  Bitmap Heap Scan on items2  (cost=48.98..2582.44 rows=642 wid
              Recheck Cond: ((setweight(to_tsvector('russian'::regconfig,
              Buffers: shared hit=6875
              ->  Bitmap Index Scan on fts_idx2  (cost=0.00..48.82 rows=6

                  Index Cond: ((setweight(to_tsvector('russian'::regconfi
                  Buffers: shared hit=116
Total runtime: 862.551 ms
```

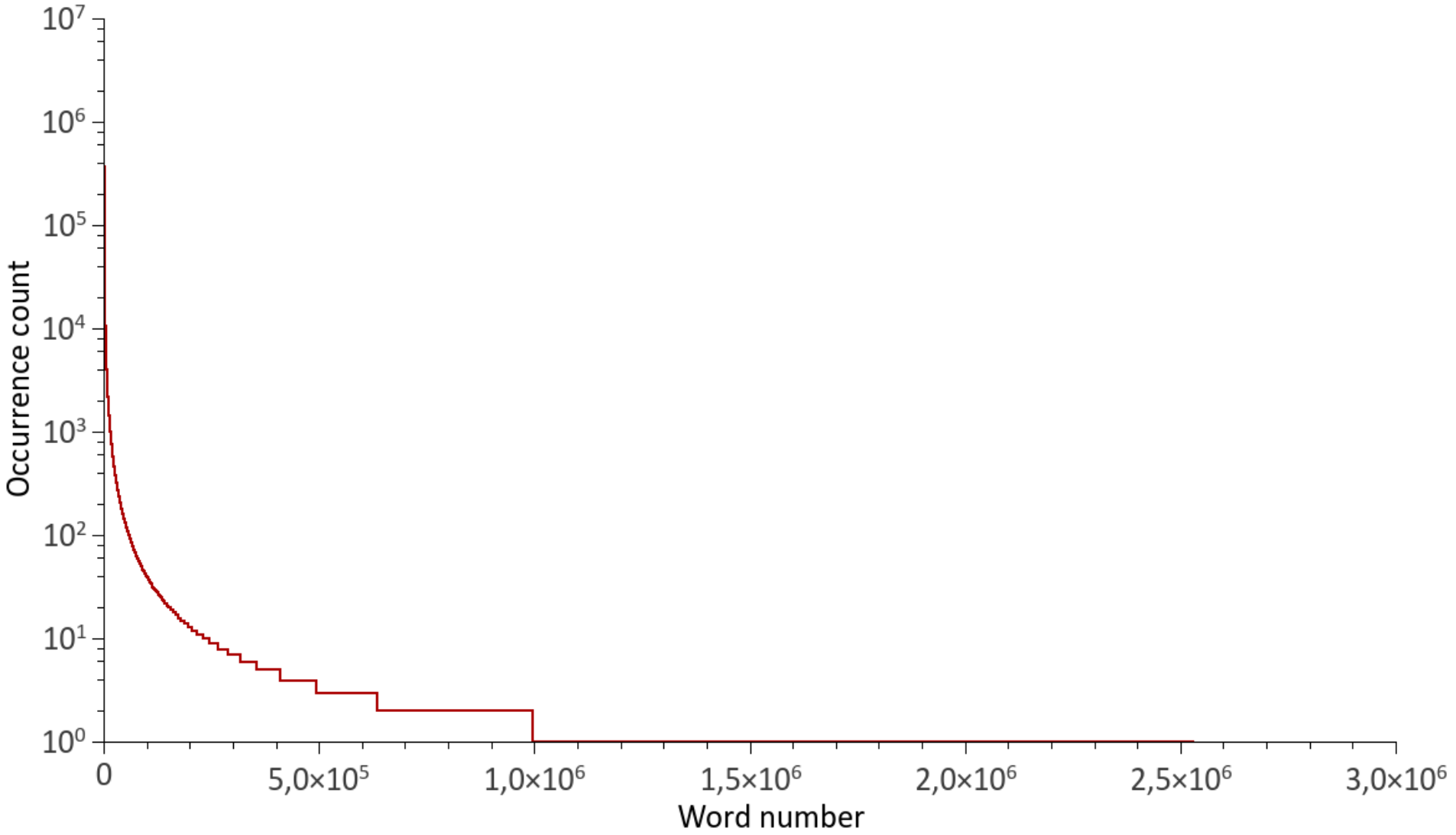
Without tsvector column, with patch

```
 Limit  (cost=40.02..80.43 rows=10 width=373) (actual
time=11.298..11.304
   Buffers: shared hit=374
   ->  Index Scan using fts_idx2 on items2
(cost=40.02..2771.68 rows=676
         Index Cond:
((setweight(to_tsvector('russian'::regconfig, title),
           Order By:
((setweight(to_tsvector('russian'::regconfig, title),
           Buffers: shared hit=374
 Total runtime: 11.321 ms
```

avito.ru: tests

	Without patch	With patch	With patch without tsvector	Sphinx
Table size	6.0 GB	6.0 GB	2.87 GB	-
Index size	1.29 GB	1.27 GB	1.27 GB	1.12 GB
Index build time	216 sec	303 sec	718sec	180 sec*
Queries in 8 hours	**3,0 mln.**	**42.7 mln.**	**42.7 mln.**	**32.0 mln.**

Anonymous source: 18 mln. docs



Anonymous source: tests

	Without patch	With patch	With patch without tsvector	Sphinx
Table size	18.2 GB	18.2 GB	11.9 GB	-
Index size	2.28 GB	2.30 GB	2.30 GB	3.09 GB
Index build time	258 sec	684 sec	1712 sec	481 sec*
Queries in 8 hours	**2.67 mln.**	**38.7 mln.**	**38.7 mln.**	**26.7 mln.**


Пуляет!!!

Status & Availability

- 150 Kb patch for 9.3
- Datasets and workloads are welcome

Plans & TODO

- Fix recovery
- Fix fastupdate
- Fast scan interface
- Accelerate index build if possible
- Partial match support

Thanks!
Sponsors are welcome!