Full-text search in PostgreSQL in milliseconds

Oleg Bartunov (thanks 1C for support)

Alexander Korotkov

FTS in PostgreSQL

- 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

It's cool, but we want faster FTS !

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 ?

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;

HEAP IS SLOW
400 ms !

```
 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..433.749 rows=3 loops=1)
         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)
(actual time=15.094..423.452 rows=47855 loops=1)
               Recheck Cond: (text_vector @@ '''titl'''::tsquery)
               ->  Bitmap Index Scan on ti2_index  (cost=0.00..517.69 rows=47692 wi
(actual time=13.736..13.736 rows=47855 loops=1)
                     Index Cond: (text_vector @@ '''titl'''::tsquery)
 Total runtime: 433.787 ms
```

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
(actual time=18.375..18.425 rows=3 loops=1)
         Index Cond: (text_vector @@ '''titl'''::tsquery)
         Order By: (text_vector >< '''titl'''::tsquery)
 Total runtime: 18.511 ms vs 433.787 ms
```

We'll be FINE !

6.7 mln classifieds

	Without patch	With patch	With patch functional index	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.**

WOW !!!

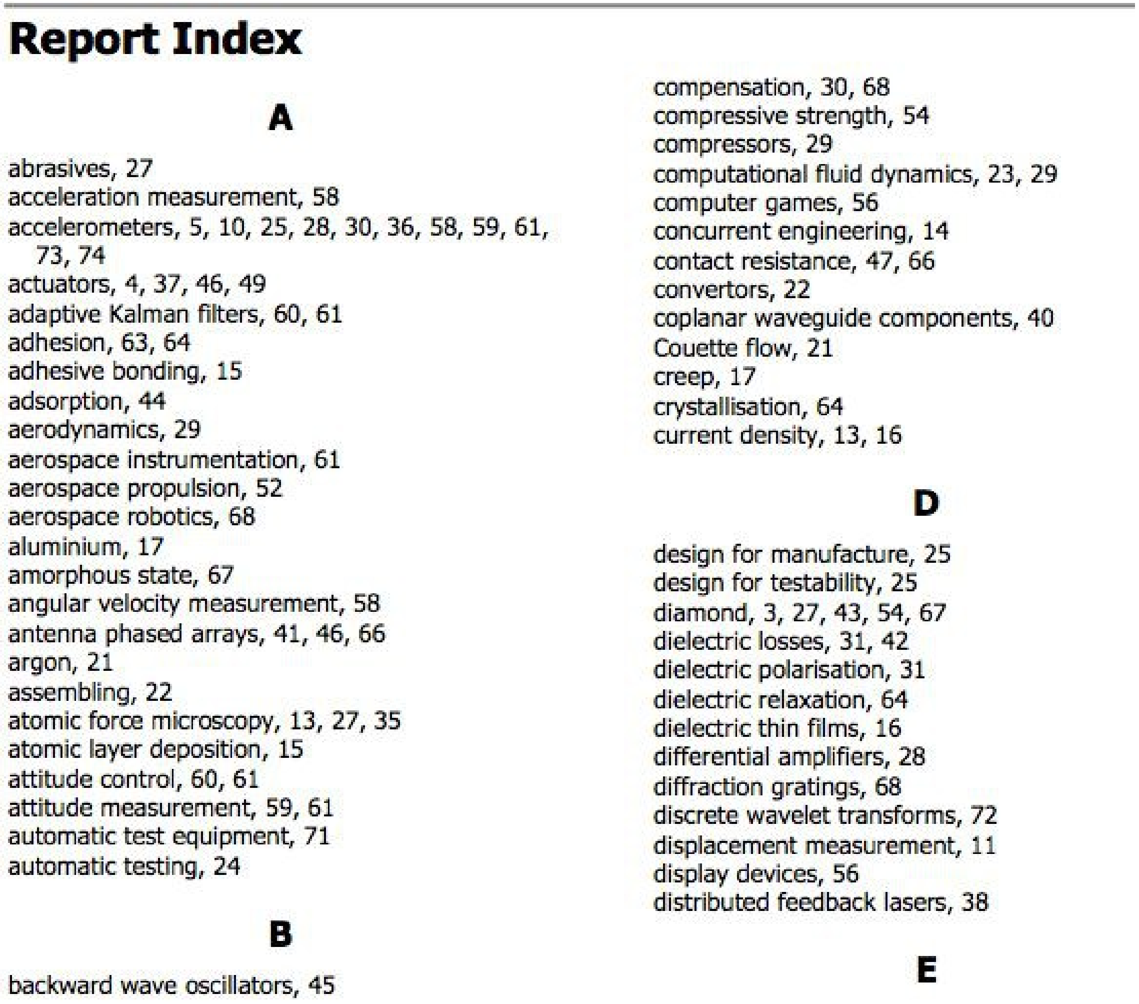
20 mln descriptions

	Without patch	With patch	With patch functional index	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.**

WOW !!!

GIN improvements

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
aerospace instrumentation, 61
aerospace propulsion, 52
aerospace robotics, 68
aluminium, 17
amorphous state, 67
angular velocity measurement, 58
antenna phased arrays, 41, 46, 66
argon, 21
assembling, 22
atomic force microscopy, 13, 27, 35
atomic layer deposition, 15
attitude control, 60, 61
attitude measurement, 59, 61
automatic test equipment, 71
automatic testing, 24

B

backward wave oscillators, 45

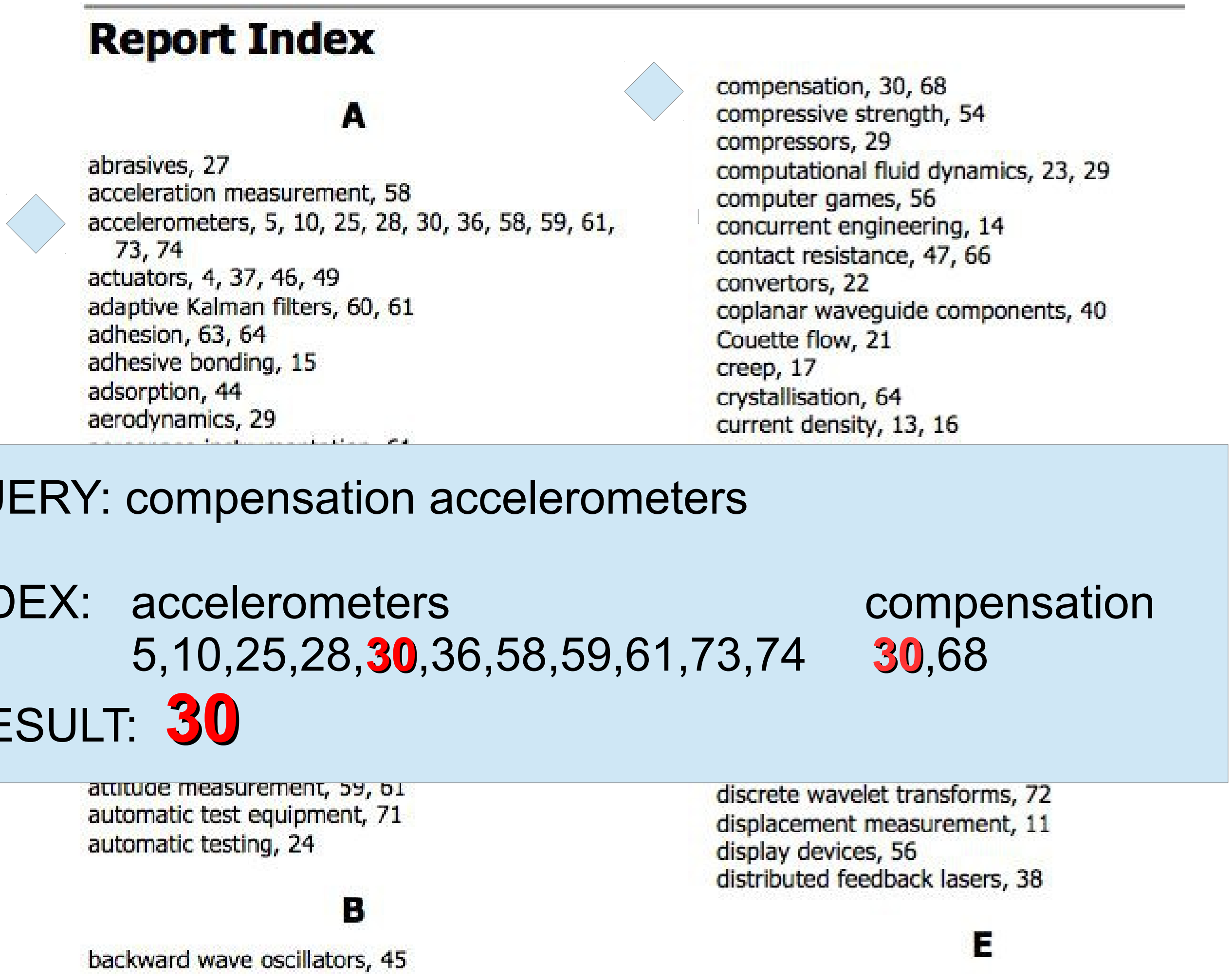
compensation, 30, 68
compressive strength, 54
compressors, 29
computational fluid dynamics, 23, 29
computer games, 56
concurrent engineering, 14
contact resistance, 47, 66
convertors, 22
coplanar waveguide components, 40
Couette flow, 21
creep, 17
crystallisation, 64
current density, 13, 16

D

design for manufacture, 25
design for testability, 25
diamond, 3, 27, 43, 54, 67
dielectric losses, 31, 42
dielectric polarisation, 31
dielectric relaxation, 64
dielectric thin films, 16
differential amplifiers, 28
diffraction gratings, 68
discrete wavelet transforms, 72
displacement measurement, 11
display devices, 56
distributed feedback lasers, 38

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

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
current density, 13, 16

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

E

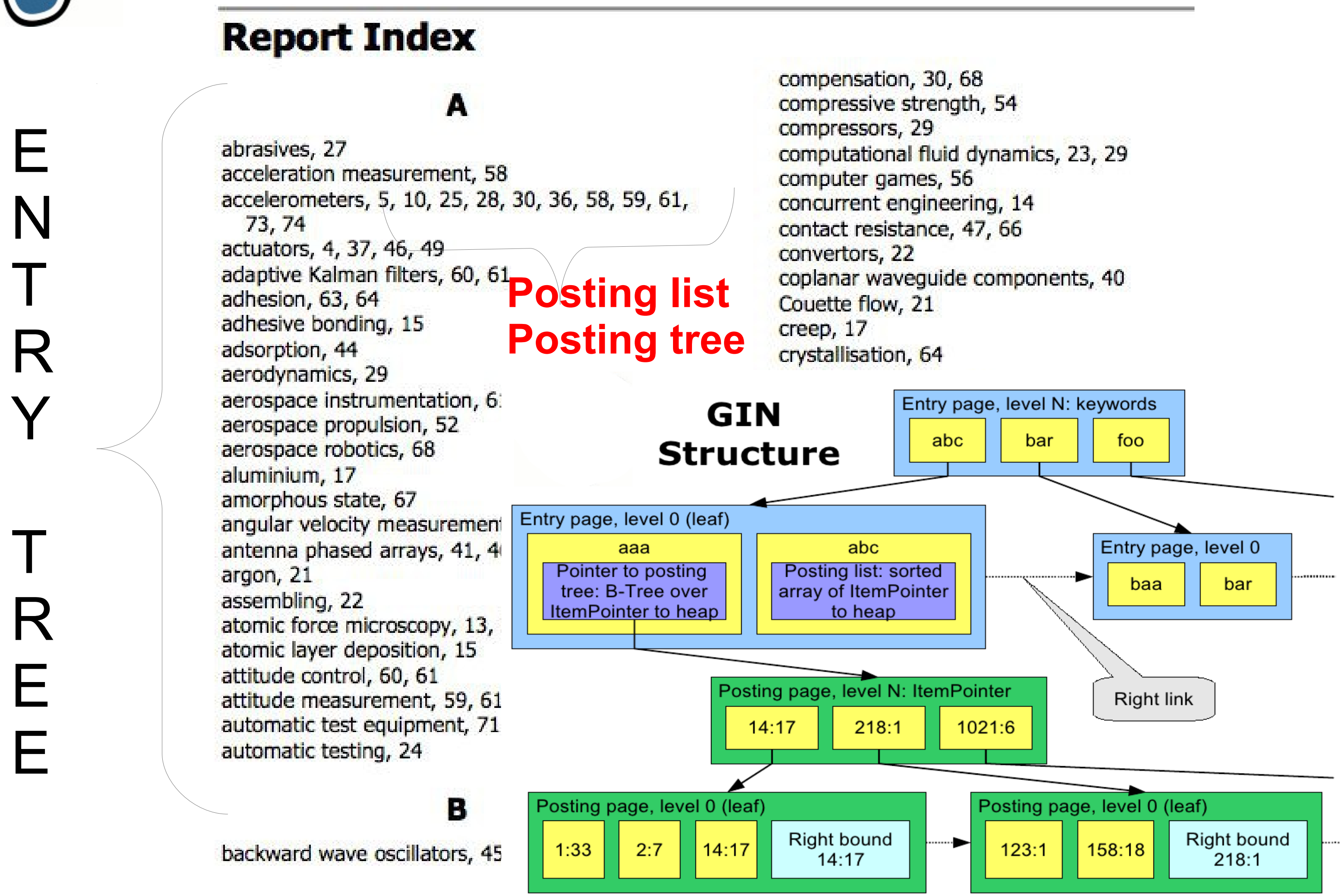
QUERY: compensation accelerometers

INDEX:	accelerometers	compensation
	5,10,25,28,**30**,36,58,59,61,73,74	**30**,68

RESULT: **30**

No positions in index !

Inverted Index in PostgreSQL



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 measuremen
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

ENTRY TREE

Posting list
Posting tree

GIN Structure

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

Summary of changes

- Compressed storage with additional information
- Optimized search («frequent_entry & rare_entry» case)
- Return ordered results by index (ORDER BY optimization)

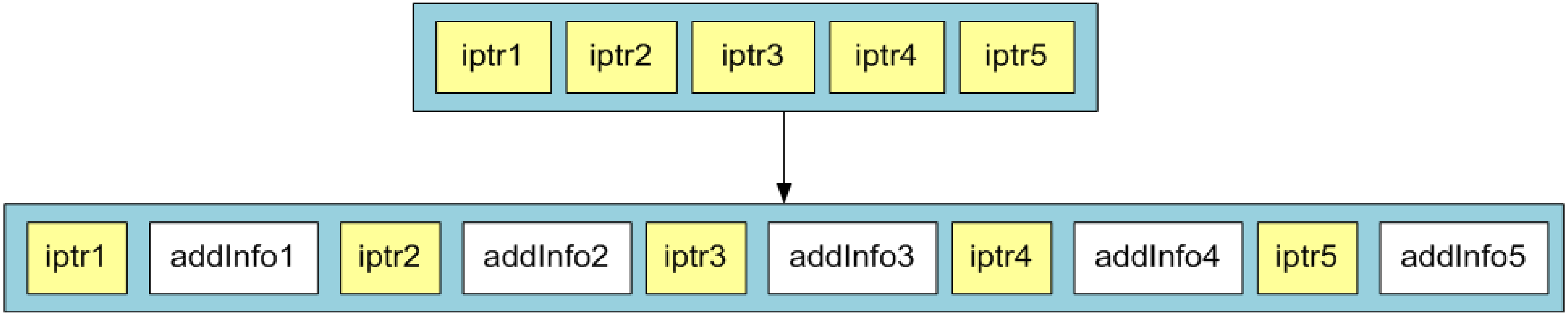
interface changes needs for all this stuff

Every GIN application can have a benefit

- Fulltext search: store word positions, get results in relevance order.
- Trigram indexes: store trigram positions, get results in similarity order.
- Array indexes: store array length, get results in similarity order.

Store additional information

See Appendix 1 for more details



iptr1 iptr2 iptr3 iptr4 iptr5

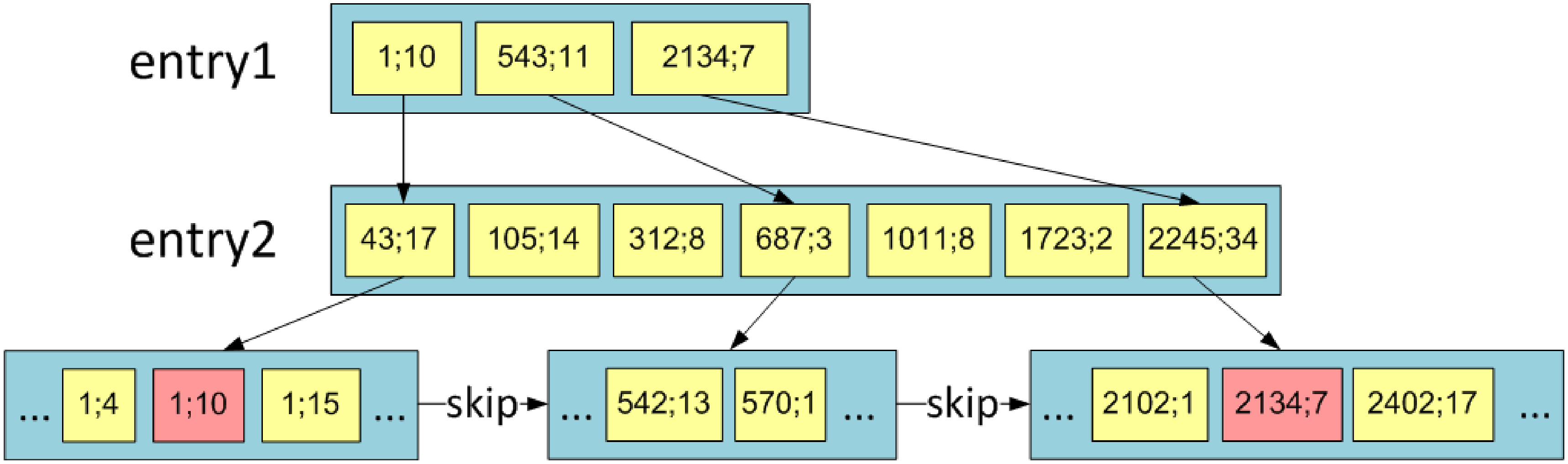
iptr1 addInfo1 iptr2 addInfo2 iptr3 addInfo3 iptr4 addInfo4 iptr5 addInfo5

Use increments and variable byte encoding to keep index small

1034, 1036, 1038 (12 bytes) => 1034, 2, 2 (4 bytes)

Fast scan

entry1 && entry2

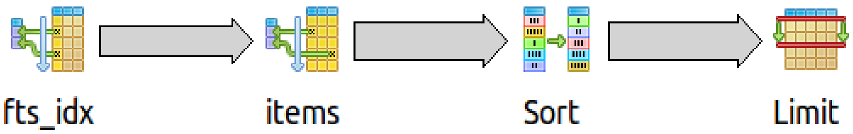


Visiting 3 pages instead of 7

ORDER BY using index

Before

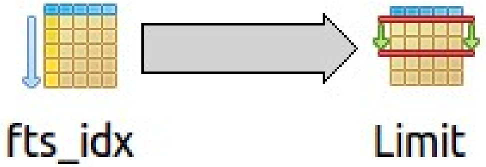
SELECT itemid, title
FROM items
WHERE fts @@ to_tsquery('english', 'query')
ORDER BY
ts_rank(fts, to_tsquery('english', 'query')) DESC
LIMIT 10;



Ranking and sorting are outside the fulltext index

After

SELECT itemid, title
FROM items
WHERE fts @@ to_tsquery('english', 'query')
ORDER BY
fts **><** to_tsquery('english', 'query')
LIMIT 10;



Index returns data ordered by rank. Ranking and sorting are inside.

368 ms vs 13 ms

Example: frequent entry (30%)

Before:

node type	count	sum of times	% of query
Bitmap Heap Scan	1	367.687 ms	94.6 %
Bitmap Index Scan	1	6.570 ms	1.7 %
Limit	1	0.001 ms	0.0 %
Sort	1	14.465 ms	3.7 %

388 ms

After:

node type	count	sum of times	% of query
Index Scan	1	13.346 ms	100.0 %
Limit	1	0.001 ms	0.0 %

13 ms

Example: rare entry (0.08%)

Before:

node type	count	sum of times	% of query
Bitmap Heap Scan	1	0.959 ms	93.4 %
Bitmap Index Scan	1	0.027 ms	2.6 %
Limit	1	0.001 ms	0.1 %
Sort	1	0.040 ms	3.9 %

1.1 ms

After:

node type	count	sum of times	% of query
Index Scan	1	0.052 ms	98.1 %
Limit	1	0.001 ms	1.9 %

0.07 ms

Example: frequent entry (30%) & rare entry (0.08%)

Before:

node type	count	sum of times	% of query
Bitmap Heap Scan	1	1.547 ms	23.0 %
Bitmap Index Scan	1	5.151 ms	76.7 %
Limit	1	0.000 ms	0.0 %
Sort	1	0.022 ms	0.3 %

6.7 ms

After:

node type	count	sum of times	% of query
Index Scan	1	0.998 ms	100.0 %
Limit	1	0.000 ms	0.0 %

1.0 ms

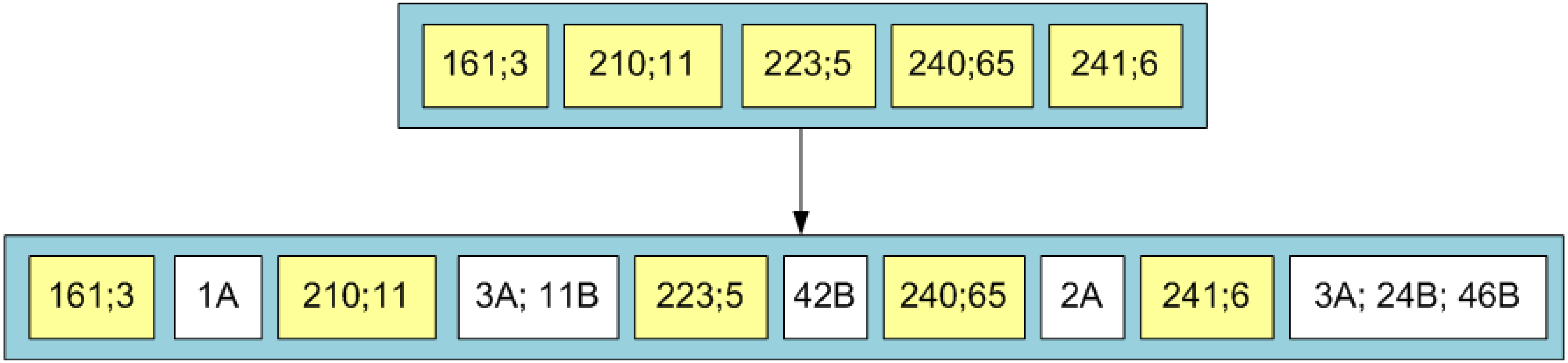
Sponsors are welcome!

- 150 Kb patch for 9.3
- Todo:
  - Fix everything we broke :(
  - Fast scan interface
  - Accelerate index build
  - Partial match support
- Datasets and workloads are welcome

Appendix 1

Compressed storage of additional information in GIN

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

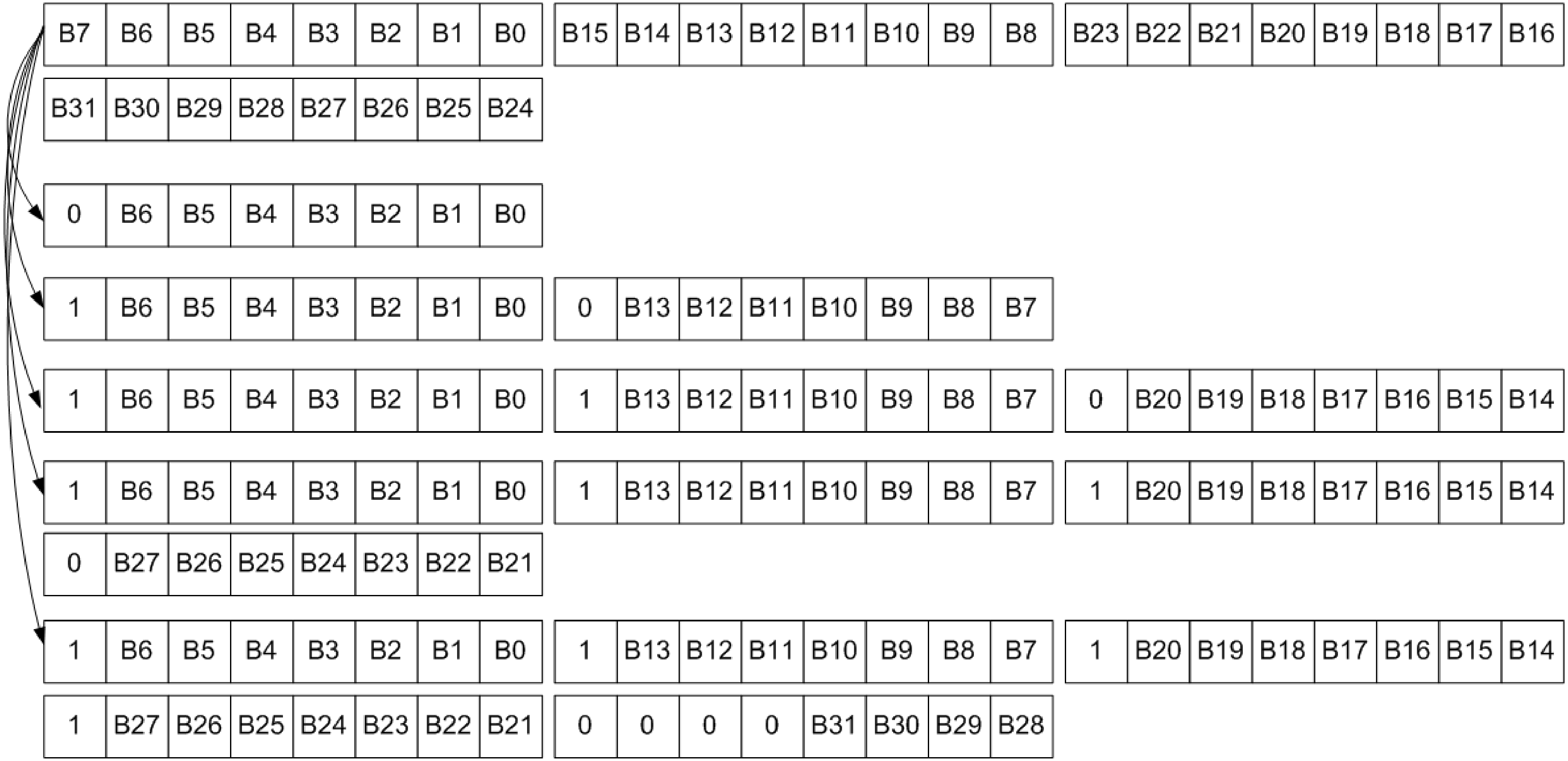
WordEntryPos

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

typedef uint16 WordEntryPos;
```

2 bytes

BlockIdData compression



B7	B6	B5	B4	B3	B2	B1	B0	B15	B14	B13	B12	B11	B10	B9	B8	B23	B22	B21	B20	B19	B18	B17	B16
B31	B30	B29	B28	B27	B26	B25	B24																

0	B6	B5	B4	B3	B2	B1	B0																

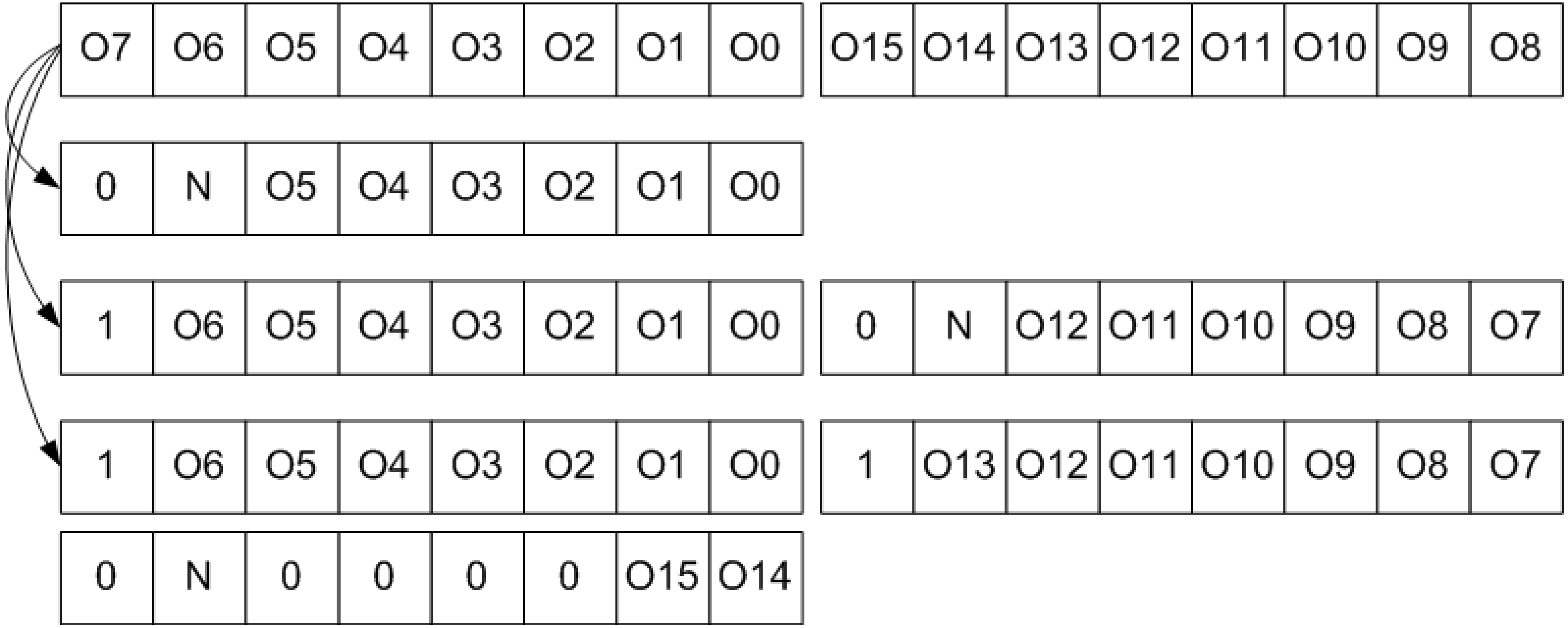
1	B6	B5	B4	B3	B2	B1	B0	0	B13	B12	B11	B10	B9	B8	B7								

1	B6	B5	B4	B3	B2	B1	B0	1	B13	B12	B11	B10	B9	B8	B7	0	B20	B19	B18	B17	B16	B15	B14

1	B6	B5	B4	B3	B2	B1	B0	1	B13	B12	B11	B10	B9	B8	B7	1	B20	B19	B18	B17	B16	B15	B14
0	B27	B26	B25	B24	B23	B22	B21																

1	B6	B5	B4	B3	B2	B1	B0	1	B13	B12	B11	B10	B9	B8	B7	1	B20	B19	B18	B17	B16	B15	B14
1	B27	B26	B25	B24	B23	B22	B21	0	0	0	0	B31	B30	B29	B28								

OffsetNumber compression

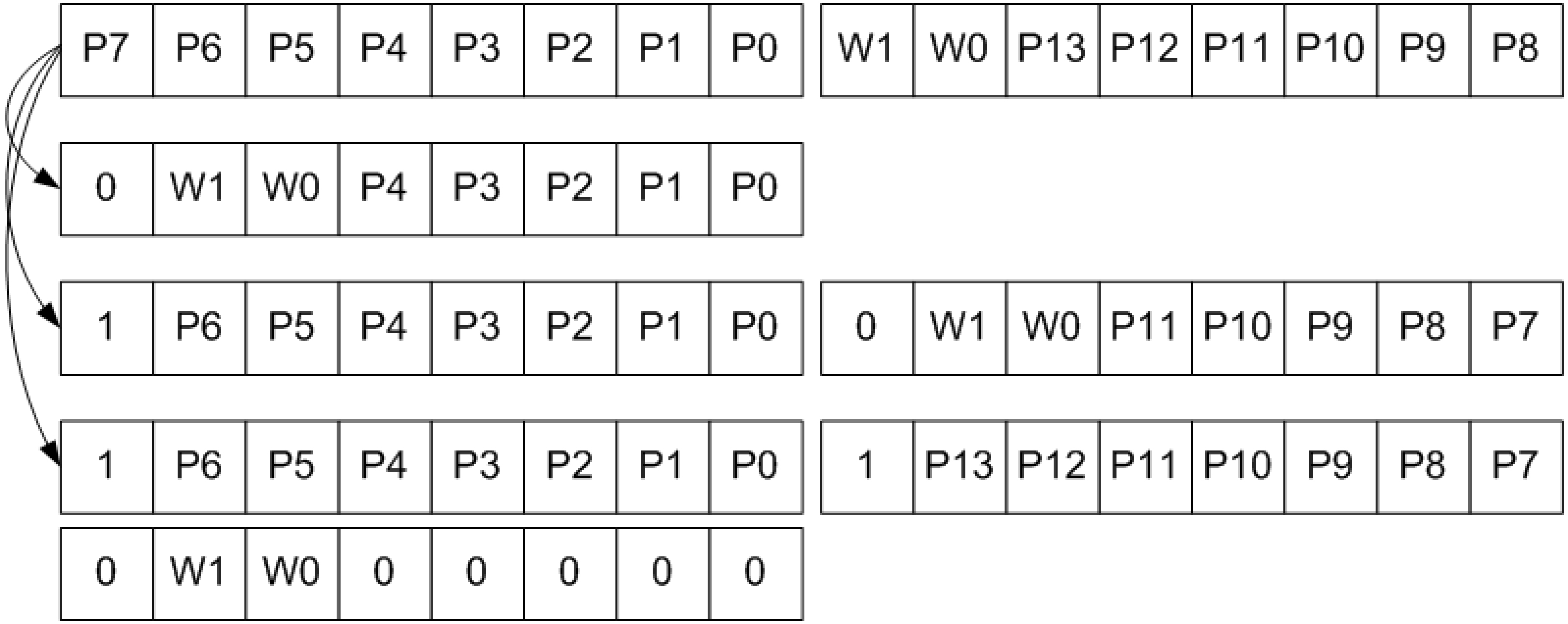


O7	O6	O5	O4	O3	O2	O1	O0	O15	O14	O13	O12	O11	O10	O9	O8
0	N	O5	O4	O3	O2	O1	O0								
1	O6	O5	O4	O3	O2	O1	O0	0	N	O12	O11	O10	O9	O8	O7
1	O6	O5	O4	O3	O2	O1	O0	1	O13	O12	O11	O10	O9	O8	O7
0	N	0	0	0	0	O15	O14								

O0-O15 – OffsetNumber bits

N – Additional information NULL bit

WordEntryPos compression



P7	P6	P5	P4	P3	P2	P1	P0	W1	W0	P13	P12	P11	P10	P9	P8

0	W1	W0	P4	P3	P2	P1	P0

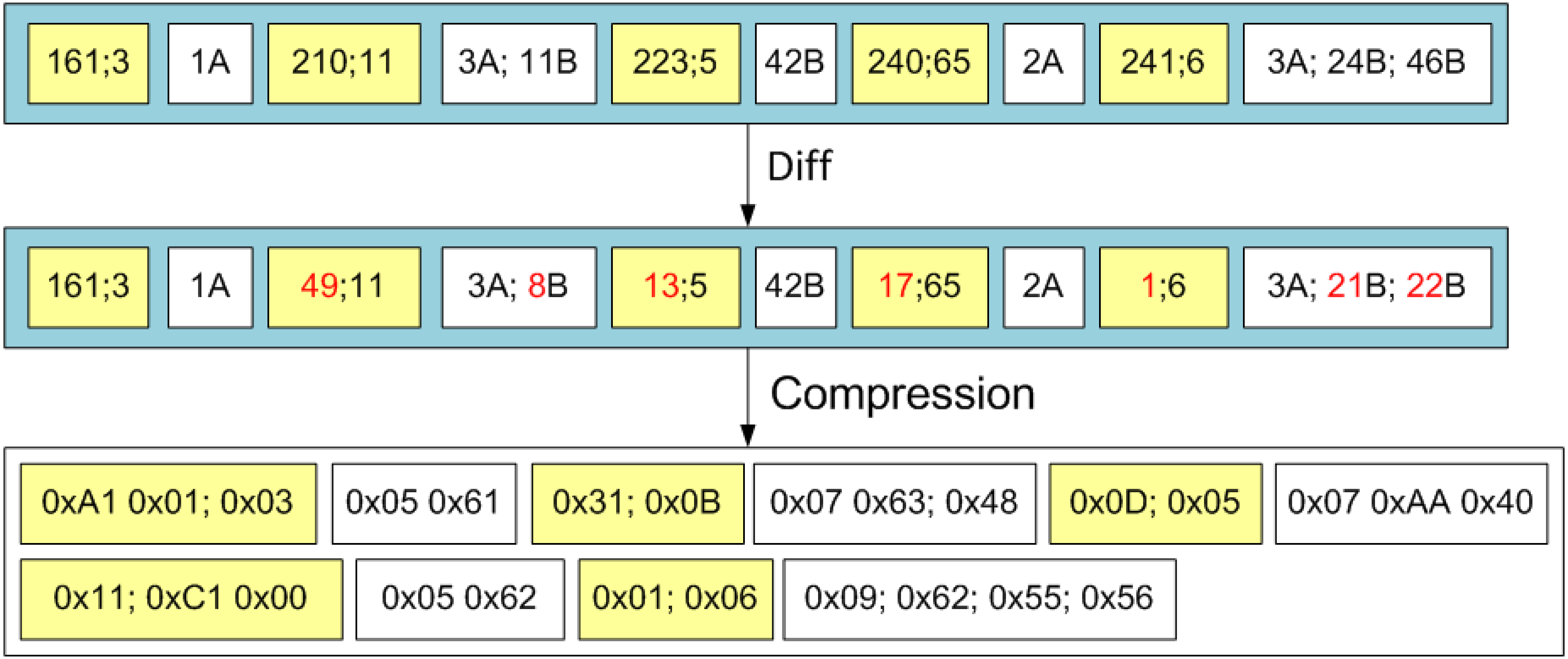
1	P6	P5	P4	P3	P2	P1	P0	0	W1	W0	P11	P10	P9	P8	P7

1	P6	P5	P4	P3	P2	P1	P0	1	P13	P12	P11	P10	P9	P8	P7
0	W1	W0	0	0	0	0	0								

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 interface changes

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 addInfo[]
)
```

Planner optimization

Remove unused targets when ORDER BY uses index

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)
```