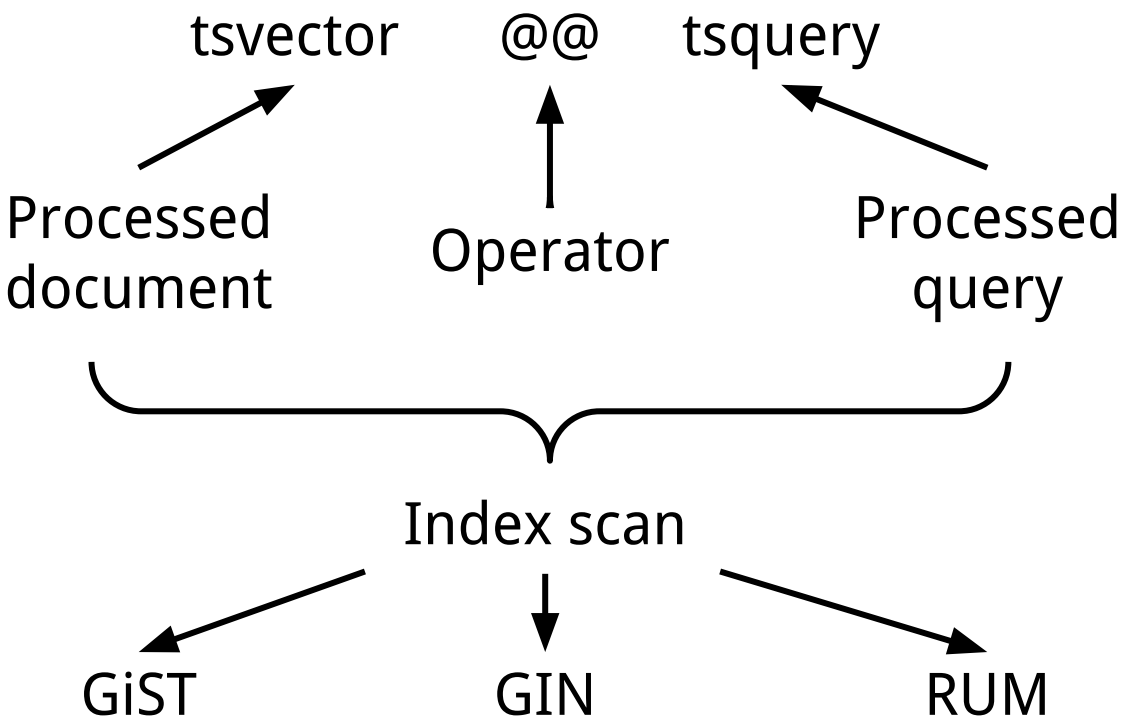
Flexible Full Text Search

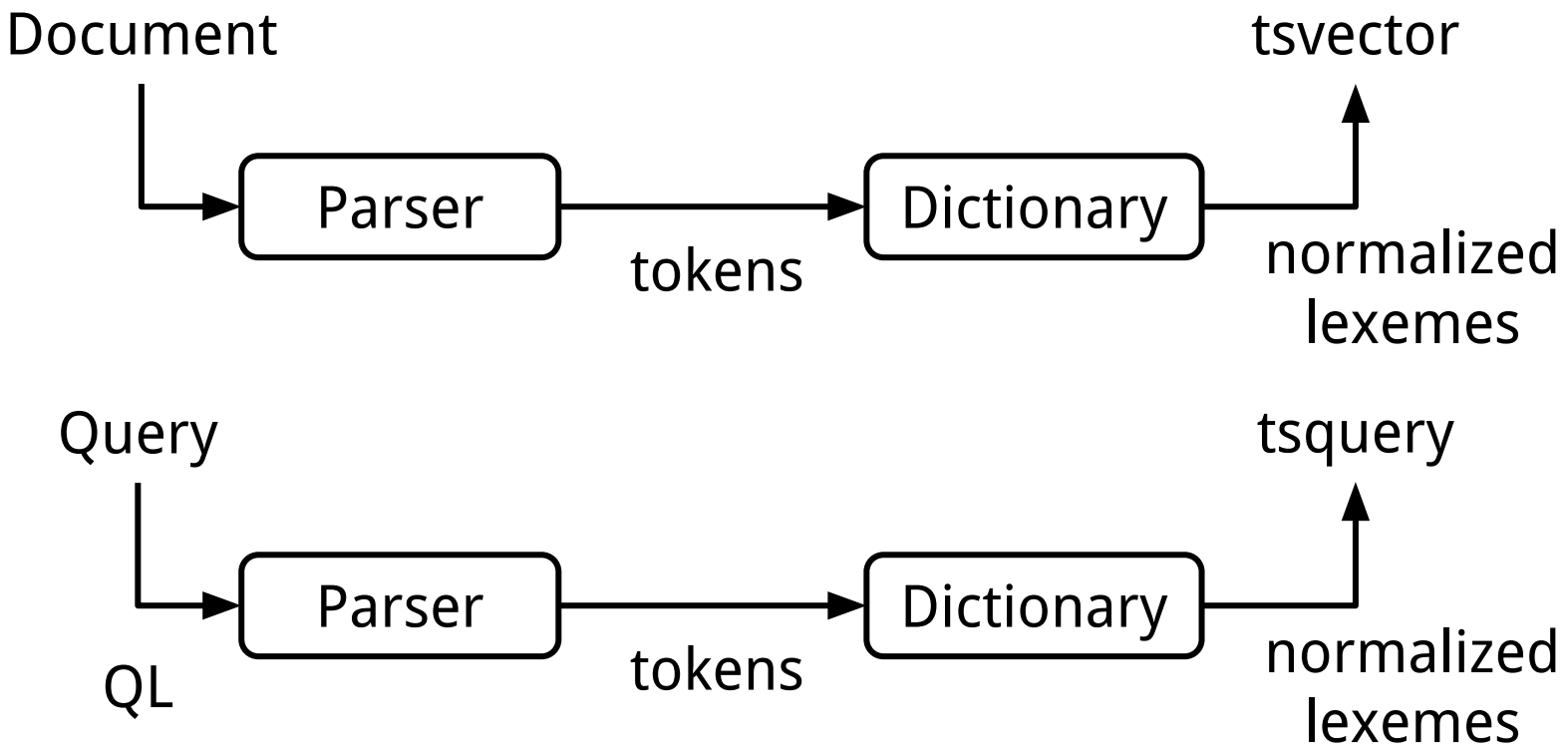
Aleksandr Parfenov
Arthur Zakirov

PGConf.EU-2017, Warsaw

FTS in PostgreSQL



Document and Query Preprocessing



tsvector

```
SELECT to_tsvector('english','The quick brown fox jumps
                              over the lazy dog');
-----------------------------------------------------
 'brown':3 'dog':9 'fox':4 'jump':5 'lazi':8 'quick':2
```

tsvector with labels:

```
SELECT setweight(to_tsvector('english','quick brown'),'A')
               || to_tsvector('english','lazy dog');
-----------------------------------------
 'brown':2A 'dog':4 'lazi':3 'quick':1A
```

tsquery

```
SELECT to_tsquery('english','quick & (fox | dog)');
------------------------------
 'quick' & ( 'fox' | 'dog' )
```

tsquery with labels:

```
SELECT to_tsquery('english','quick:AB & dog');
--------------------
 'quick':AB & 'dog'
```

tsquery for prefix search:

```
SELECT to_tsquery('english','quick & eleph:*');
----------------------
 'quick' & 'eleph':*
```

Inverse FTS

It is possible to index not only tsvector but tsquery as well.

Use cases:

- Find queries, which match given document (subscription)
- Automatic text classification

Inverse FTS Example

```
SELECT * FROM queries;
------------------+-----------
 'black' & 'hole' | astronomy
 'red' & 'hat'    | linux
 'black' & 'flag' | pirate

SELECT * FROM queries
WHERE to_tsvector('black holes never exist') @@ q;
------------------+-----------
 'black' & 'hole' | astronomy
```

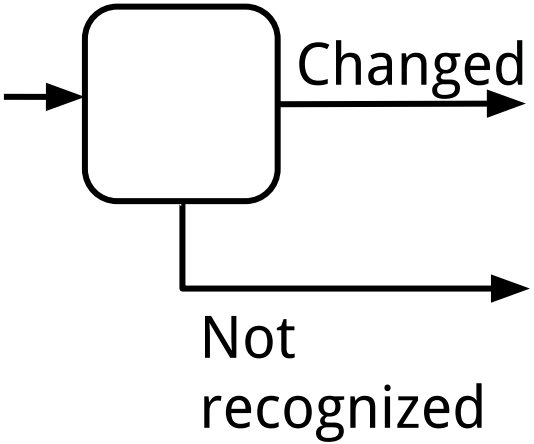
FTS Parser

- Splits text into tokens
- Determines type of each token

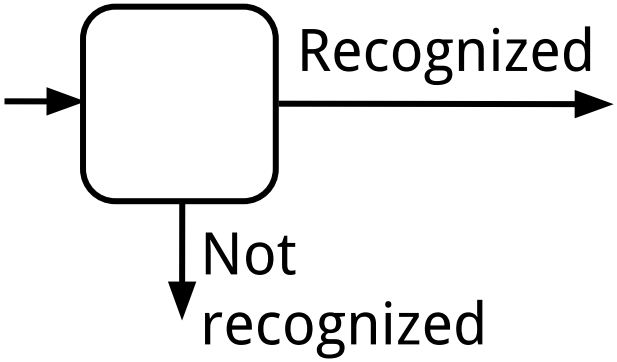
```
SELECT alias AS "token type", token
FROM ts_debug('simple', '100500 fox postgresql.org');
 token type |      token
------------+-----------------
 uint       | 100500
 blank      |
 asciiword  | fox
 blank      |
 host       | postgresql.org
```

FTS Dictionary Types

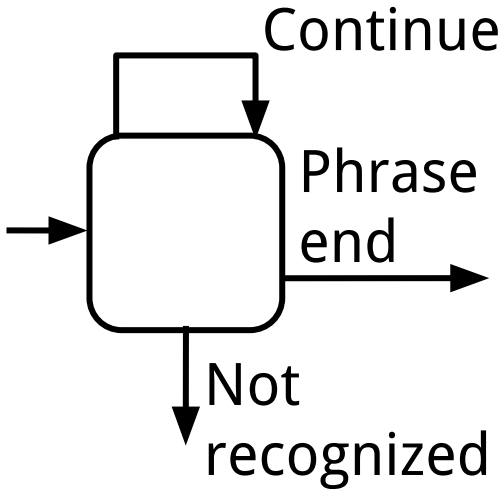
Filtering



Regular



Phrase



Postgres PROFESSIONAL

FTS Dictionary Types

- **Regular dictionaries** - return token if it recognized, otherwise transfer control to next dictionary (Examples: *ispell, simple, synonym, snowball*)
- **Filtering dictionaries** - change token if it recognized, always transfer control to next dictionary (Example: *unaccent*)
- **Phrase dictionaries** - same as regular but can recognize more than one token, hold control until the end of phrase processing (Examples: *thesaurus*)

FTS Dictionaries

simple:

```
SELECT to_tsvector('simple','Best database');
------------------------
 'best':1 'database':2
```

synonym:

```
SELECT to_tsvector('synonym_sample','Best database');
----------------------------
 'database':2 'wonderful':1
```

thesaurus:

```
SELECT to_tsvector('thesaurus_sample','Best database');
-----------------
 'postgresql':1
```

FTS Dictionaries

snowball:

```
SELECT to_tsvector('english','quick elephants');
----------------------
 'eleph':2 'quick':1
```

ispell:

```
SELECT to_tsvector('english_hunspell','quick elephants');
-------------------------
 'elephant':2 'quick':1
```

FTS Dictionaries

Hunspell dictionaries for several languages

github.com/postgrespro/hunspell_dicts

Ispell dictionary stored in shared memory

github.com/postgrespro/shared_ispell

- Consumes less memory
- First call of ispell dictionary per connection is faster

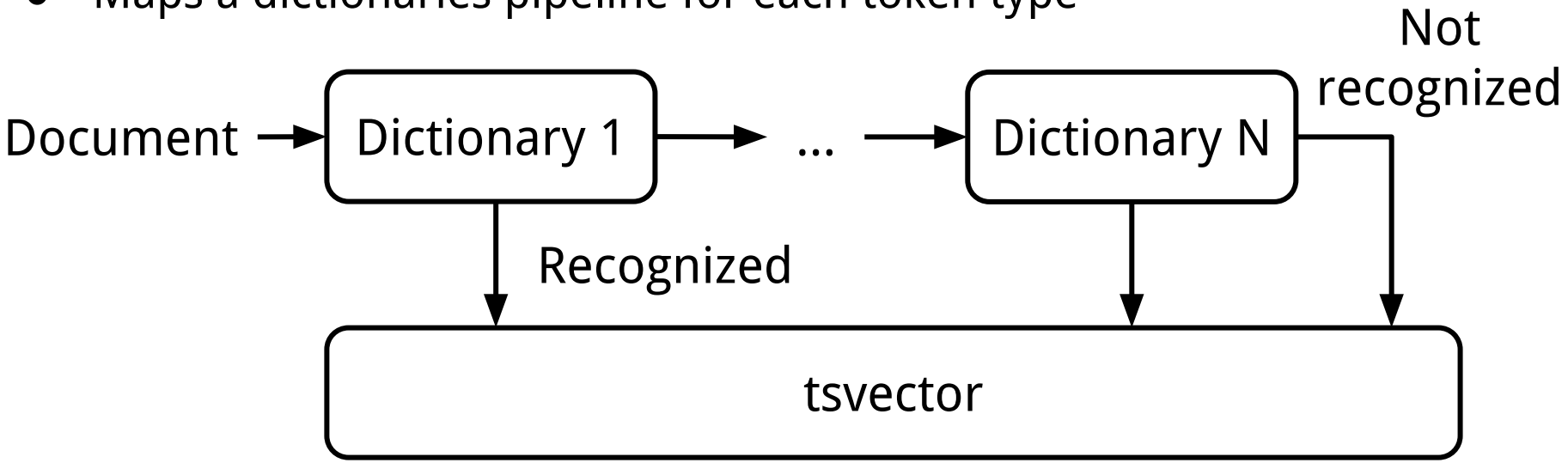
FTS Dictionaries: contrib/unaccent

```
SELECT ts_lexize('unaccent', 'brown');
--------
 (null)

SELECT ts_lexize('unaccent', 'träge');
---------
 {trage}
```

unaccent returns a lexeme with **TSL_FILTER** flag.

FTS Configuration

- Connection point for parser and dictionaries
- Defines how text should be processed
- Maps a dictionaries pipeline for each token type

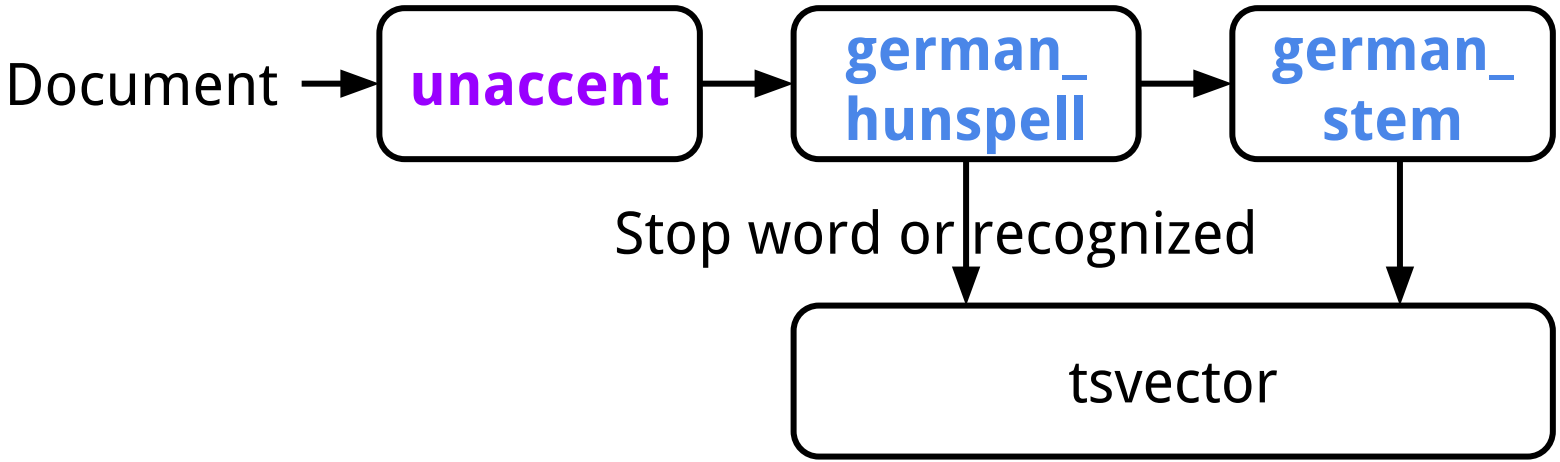


FTS Configuration Example

Configuration with unaccent dictionary:

```
CREATE EXTENSION hunspell_de_de;
CREATE EXTENSION unaccent;
CREATE TEXT SEARCH CONFIGURATION de_conf (copy='simple');

ALTER TEXT SEARCH CONFIGURATION de_conf
ALTER MAPPING FOR asciiword, asciihword, hword_asciipart,
                  word, hword, hword_part
WITH unaccent, german_hunspell, german_stem;
```

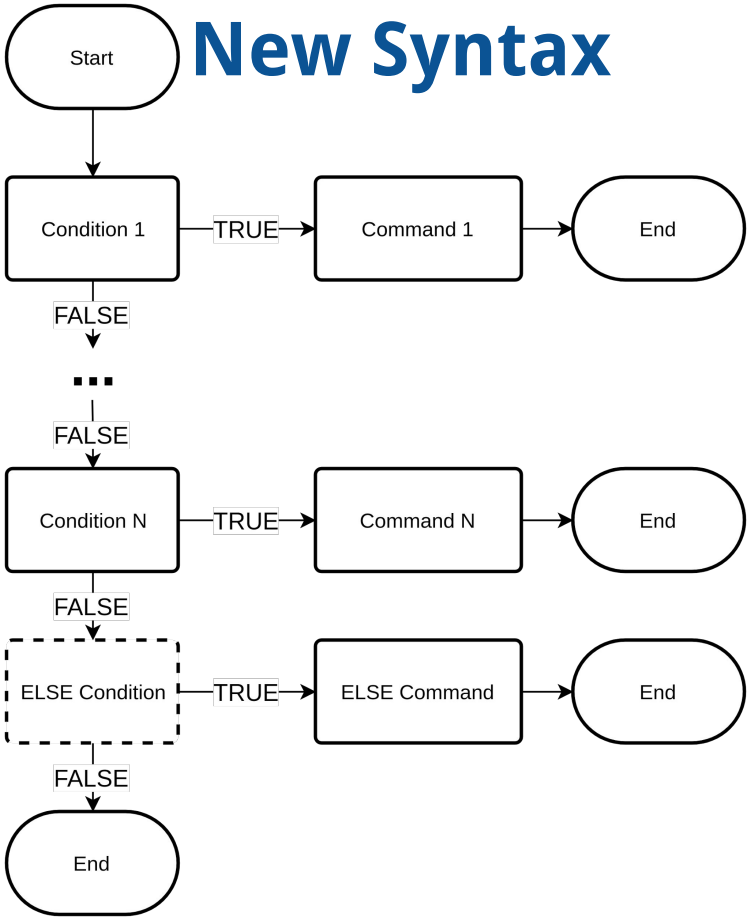
FTS Configuration Example



FTS Configuration New Syntax

- Based on *CASE/WHEN/THEN/ELSE* syntax
- Separate selection of dictionaries and building lexemes set
- Filtering dictionaries work as regular
- Condition is a logical expression on dictionaries output
- Command is a set expression on dictionaries output

github.com/postgrespro/postgres/tree/flexible-fts



FTS Configuration: Condition Expression

- AND, OR and NOT operators
- Dictionary output can be casted to boolean via *IS [NOT] NULL* and *IS [NOT] STOPWORD* clauses

  `german_hunspell IS NOT NULL` *OR* `english_hunspell IS NOT NULL`

- Dictionary name without clauses interpreted as:
  `dictionary IS NOT NULL` *AND* `dictionary IS NOT STOPWORD`

FTS Configuration: *MAP BY* Operator

- Used for dictionary pipeline to connect dictionaries between each other
- If output of the dictionary is not *NULL* it is passed as an input to next dictionary
- May be used with any dictionary
- May be used in both: condition and command
- Example:
  Old syntax: `unaccent, english_stem`
  New syntax: `english_stem MAP BY unaccent`

FTS Multilingual Search

- Data: set of documents in different languages
- No markers for languages of the document
- Old solution:
  Separate configuration for each languages and separate *tsvector* and *tsquery*

```
=# SELECT * FROM apod_en_de WHERE
to_tsvector('english', text) @@
to_tsquery('english', 'query')
  OR
to_tsvector('german', text) @@
to_tsquery('german', 'query');
```

FTS Multilingual Search (new)

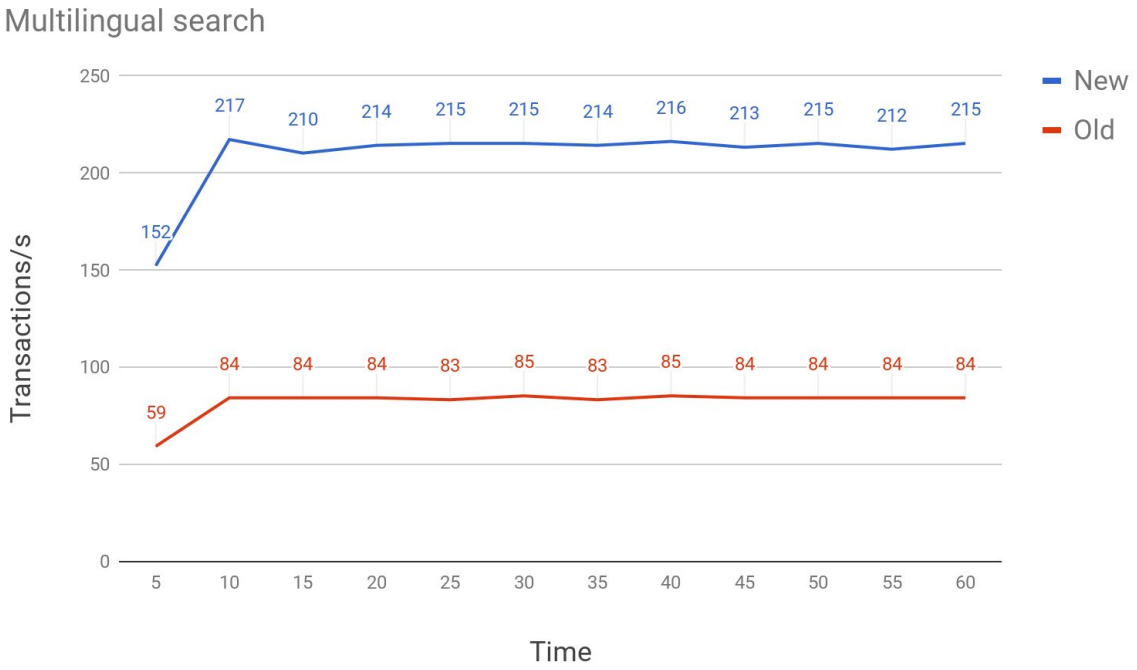
```
ALTER TEXT SEARCH CONFIGURATION multi
ALTER MAPPING FOR asciiword, asciihword, word, hword
                  hword_asciipart, hword_part WITH
   CASE
      WHEN english_hunspell AND german_hunspell THEN
           english_hunspell UNION german_hunspell
      WHEN english_hunspell THEN english_hunspell
      WHEN german_hunspell THEN german_hunspell
      ELSE german_stem UNION english_stem
   END;

SELECT * FROM apod_en_de WHERE
to_tsvector('multi', text)@@to_tsquery('multi', 'query')
```

FTS Multilingual Search (comparison)

	Old	New
tsvector size	EN (in EN/DE): 3769MB DE: 3722MB Sum (EN+DE): 7491MB	Union (EN+DE): 4110MB (54% of before patch size)
GIN Index size	EN (in EN/DE): 1417MB DE: 1388MB Sum (EN+DE): 2805MB	Union (EN+DE): 1449MB (52% of before patch size)

FTS Multilingual Search (comparison)

- Based on English and German APOD dump dataset
- Use *hunspell* as a main dictionaries with snowball as last dictionary in list
- ~2.5 times faster due to search on one shared index



Exact and Morphological Search

- Morphological search: for different forms of the word
- Exact search: for exact form of the word
- Goal: combine morphological and exact search in one query

- Old solution:
  Separate searches for each morphological and exact part of the query and smart combination of the results.

Exact and Morphological Search

```
ALTER TEXT SEARCH CONFIGURATION exact_and_morph
ALTER MAPPING FOR asciiword, asciihword, word, hword,
                  hword_asciipart, hword_part WITH
CASE
   WHEN english_hunspell THEN
        english_hunspell UNION simple
   ELSE english_stem UNION simple
END;
```

May cause false positive results

Special Stop Words Processing

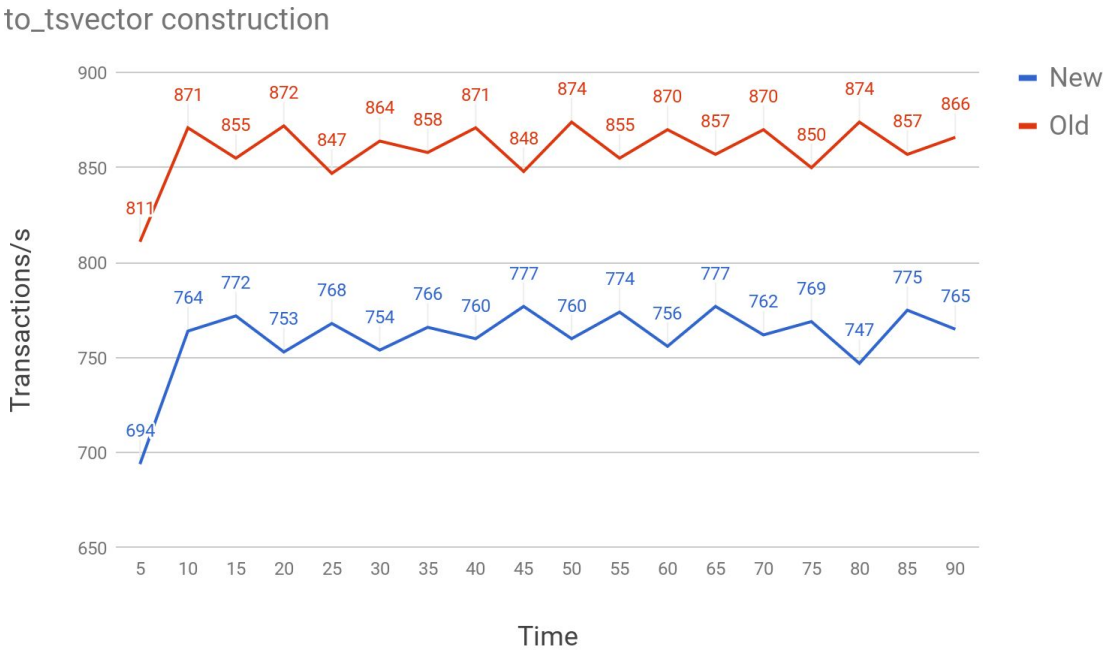
- Separate stopword detection and words normalization
- Get rid of legacy in mixing two functions of dictionaries

- Solution:

```
ALTER TEXT SEARCH CONFIGURATION stopwords
ALTER MAPPING FOR asciiword, asciihword, word, hword,
                  hword_asciipart, hword_part WITH
   CASE
      WHEN stopwords IS NOT STOPWORD THEN ispell
   END;
```

to_tsvector Construction Performance

- Based on APOD dataset
- 10 `to_tsvector` calls per transaction
- Use `english_hunpell` dictionary
- ~12% slowdown due to more complex parsing logic



PostgreSQL FTS Roadmap

- FTS for JSON and JSONB (done in PostgreSQL 10, Dmitry Dolgov)
- Remove *tsvector* size limit (commitfest.postgresql.org/15/1221/, Ildus Kurbangaliev)
- Google-like query language (commitfest.postgresql.org/15/1202/, Victor Drobny)
- Index-only count(*) for indexes (commitfest.postgresql.org/15/1117/, Alexander Kuzmenkov)
- Get rid of false positive hits in morph/exact search
- Range distance operator in phrase search
- ispell dictionary in shared memory extension core

FTS for JSON and JSONB

```
SELECT to_tsvector('english','{"type":"quick brown",
                                "animal":"fox"}'::jsonb);
-----------------------------
 'brown':2 'fox':4 'quick':1

SELECT to_tsvector('english','{"type":"quick brown",
                                "animal":"fox"}'::jsonb)
    @@ to_tsquery('english','quick <-> brown');
---
 t
```

Google-like Query Language

New function *queryto_tsquery([regconfig,] text)* with human-friendly query language

- “quick brown” - phrase search
- OR - logical clause
- -word - negation of word presents in document
- AROUND(N) - maximum distance between words/phrases

Google-like Query Language (example)

```
SELECT queryto_tsquery('english','quick "brown fox"');
------------------------------
 'quick' & 'brown' <-> 'fox'

SELECT queryto_tsquery('english',
                       'quick AROUND(3) fox -dog');
-----------------------------------
 'quick' AROUND(3) 'fox' & !'dog'
```

Thank you!

Feedback is welcome

2017.pgconf.eu/f