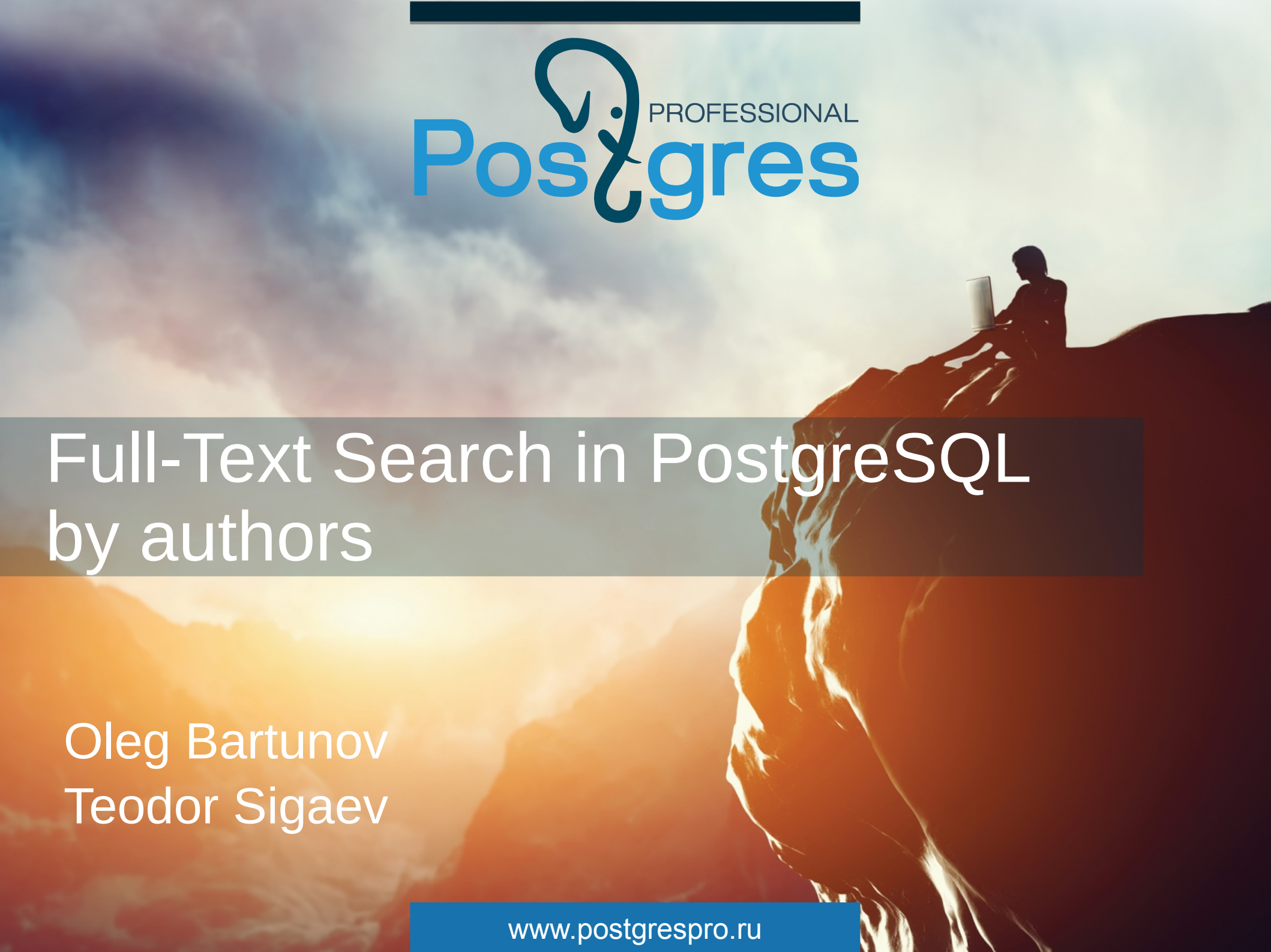


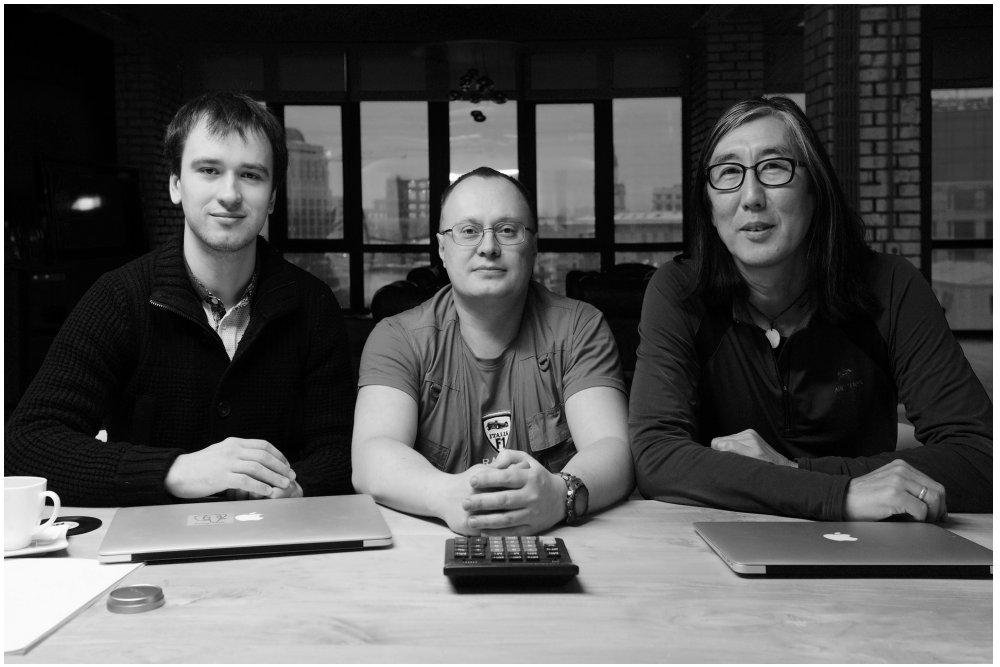
Full-Text Search in PostgreSQL by authors

Oleg Bartunov
Teodor Sigaev

www.postgrespro.ru

Russian PostgreSQL developers

Oleg Bartunov, Teodor Sigaev, Alexander Korotkov



PostgreSQL CORE

- Locale support
- PostgreSQL extendability:
- GiST(KNN), GIN, SP-GiST
- Full Text Search (FTS)
- NoSQL (hstore, jsonb)
- Indexed regexp search
- VODKA access method (WIP)

Расширения:

- Intarray
- Pg_trgm
- Ltree
- Hstore
- plantuner
- JsQuery

- Speakers at PGCon, PGConf: 20+ talks
- GSoC mentors
- PostgreSQL committers
- Conference organizers
- 50+ years of PostgreSQL expertise: development, audit and consulting
- Novartis, Raining Data, Heroku, Engine Yard, WarGaming, Rambler, Avito, 1c

Agenda

- Full text search in database
- Full text search in PostgreSQL
  - FTS types and operators
  - Parser, Dictionaries, Configurations
  - FTS indexes (GiST, GIN)
  - Addons, examples, tips
- Additional features
  - Phrase search
  - Millisecond FTS

What is a Full Text Search ?

- Full text search
  - Find documents, which match a query
  - Sort them in some order
- Typical Search
  - Find documents with **all words** from query
  - Return them sorted by similarity

Why FTS in Databases ?

- Feed database content to external search engines
  - They are fast !

  BUT
- They can't index all information, which could be totally virtual
- They don't have access to attributes - no complex queries
- They have to be maintained — headache for DBA
- Sometimes they need to be certified
- They don't provide instant search (need time to download new data and reindex)
- They don't provide consistency — search results can be already deleted from database

FTS in Databases

- **FTS requirements**
  - **Full integration with database engine**
    - Transactions
    - Concurrent access
    - Recovery
    - Online index
  - Configurability (parser, dictionary...)
  - Scalability

What is a document ?

- Arbitrary text attribute
- Combination of text attributes from the same or different tables (join) – virtual document

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

Text Search Operators

- Traditional text search operators
  (TEXT op TEXT, op - ~, ~*, LIKE, ILIKE)

```
=# select title  from apod where title ~* 'x-ray' limit 5;
                  title
-----------------------------------------
 The X-Ray Moon
 Vela Supernova Remnant in X-ray
 Tycho's Supernova Remnant in X-ray
 ASCA X-Ray Observatory
 Unexpected X-rays from Comet Hyakutake
(5 rows)

=# select title  from apod where title ilike '%x-ray%' limit 5;
                 title
-----------------------------------------
 The Crab Nebula in X-Rays
 X-Ray Jet From Centaurus A
 The X-Ray Moon
 Vela Supernova Remnant in X-ray
 Tycho's Supernova Remnant in X-ray
(5 rows)
```

What's wrong ?

- No linguistic support
  - What is a word ?
  - What to index ?
  - Word «normalization» ?
  - Stop-words (noise-words)
- No ranking
  - All documents are equally similar to query
- Slow, documents should be seq. scanned

  9.3+ index support of ~* (pg_trgm)

  select * from man lines where man line ~* '(?:(?:p(?:ostgres(?:ql)?|g?sql)|sql)) (?:(?:(?:mak|us)e|do|is))';

```
One of (postgresql,sql,postgres,pgsql,psql) space One of (do,is,use,make)
```

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 ???

FTS in PostgreSQL

- **tsvector** – data type for document optimized for search
  - Sorted array of lexems
  - Positional information
  - Structural information (importance)
- **tsquery** – textual data type for query with boolean operators & | ! ()
- **Full text search operator @@:** tsvector @@ tsquery
- Operators @>, <@ for tsquery
- **Functions**: to_tsvector, to_tsquery, plainto_tsquery, ts_lexize, ts_debug, ts_stat, ts_rewrite,ts_headline, ts_rank, ts_rank_cd, setweight
- **Indexes**: GiST, GIN

http://www.postgresql.org/docs/current/static/textsearch.html

FTS in PostgreSQL

What is the benefit ?

Document processed only once when inserting to table, no overhead in search

- Document parsed into tokens using pluggable parser
- Tokens converted to lexems using pluggable dictionaries
- Words coordinates and importance are stored and used for ranking
- Stop-words ignored

FTS in PostgreSQL

- Query processed in search time
  - Parsed into tokens
  - Tokens converted to lexems using pluggable dictionaries
  - Tokens may have weights
  - Stop-words removed from query
  - It's possible to restrict search area
    `'fat:ab & rats & ! (cats | mice)'`
  - Query can be rewritten «on-the-go»

Full-text search in PostgreSQL

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

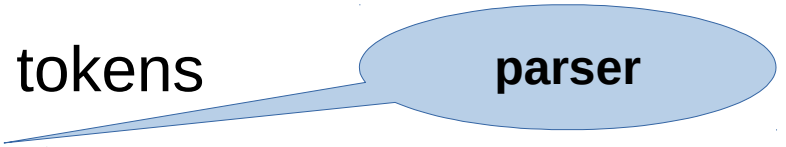
- **tsvector** – storage for document
- – sorted arrav of lexemes with optional positional and weight information
- **tsquery** – textual data type for query
- – Boolean operators - & | ! ()
- **FTS operator**

  tsvector @@ tsquery
- to_tsvector, to_tsquery, plainto_tsquery
- Indexes: GiST, GIN

'telefonsvarer' =>
'telefonsvarer' | 'telefon' & 'svar'

FTS in PostgreSQL

- Parser breaks document into tokens

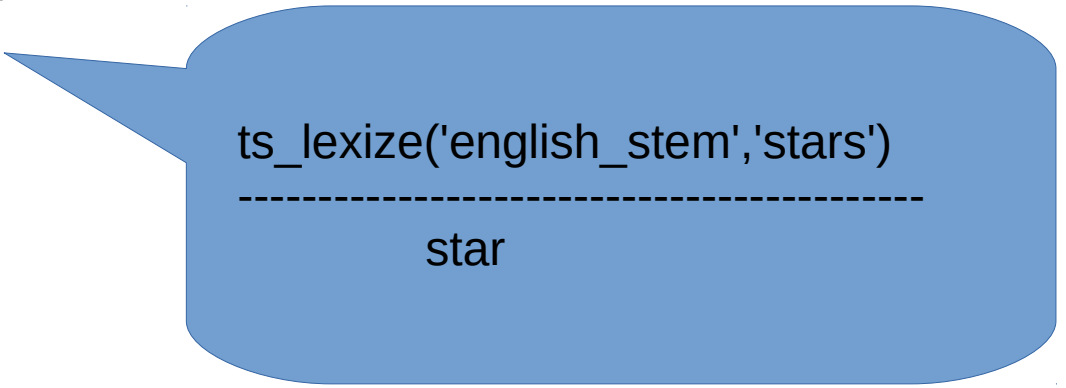



```
=# select * from ts_token_type('default');
 tokid |      alias      |                description
-------+-----------------+-----------------------------------------
     1 | asciiword       | Word, all ASCII
     2 | word            | Word, all letters
     3 | numword         | Word, letters and digits
     4 | email           | Email address
     5 | url             | URL
     6 | host            | Host
     7 | sfloat          | Scientific notation
     8 | version         | Version number
     9 | hword_numpart   | Hyphenated word part, letters and digits
    10 | hword_part      | Hyphenated word part, all letters
    11 | hword_asciipart | Hyphenated word part, all ASCII
    12 | blank           | Space symbols
    13 | tag             | XML tag
    14 | protocol        | Protocol head
    15 | numhword        | Hyphenated word, letters and digits
    16 | asciihword      | Hyphenated word, all ASCII
    17 | hword           | Hyphenated word, all letters
    18 | url_path        | URL path
    19 | file            | File or path name
    20 | float           | Decimal notation
    21 | int             | Signed integer
    22 | uint            | Unsigned integer
    23 | entity          | XML entity
(23 rows)
```

FTS in PostgreSQL

- Each token processed by a set of dictionaries

```
=# \dF+ russian
Text search configuration "pg_catalog.russian"
Parser: "pg_catalog.default"
      Token      | Dictionaries
-----------------+--------------
 asciihword      | english_stem
 asciiword       | english_stem
 email           | simple
 file            | simple
 float           | simple
 host            | simple
 hword           | russian_stem
 hword_asciipart | english_stem
 hword_numpart   | simple
 hword_part      | russian_stem
 int             | simple
 numhword        | simple
 numword         | simple
 sfloat          | simple
 uint            | simple
 url             | simple
 url_path        | simple
 version         | simple
 word            | russian_stem
```

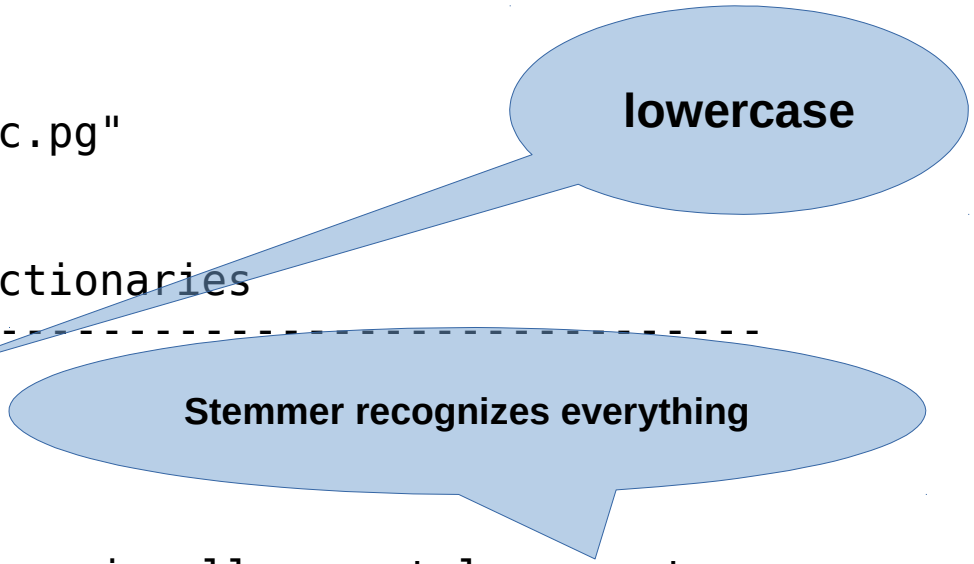


FTS in PostgreSQL

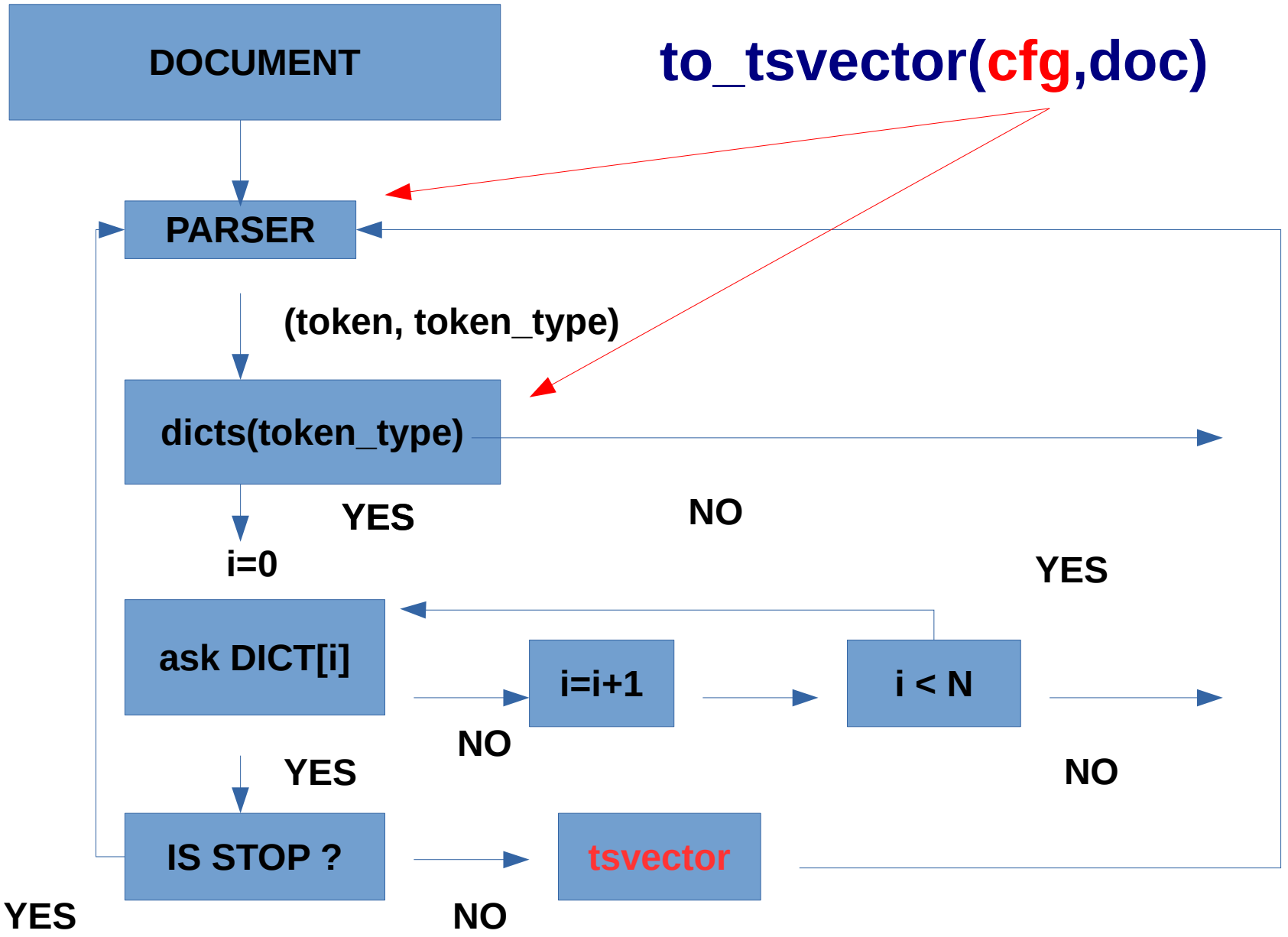
- Word processed by dictionaries until it recognized
- It is discarded, if it's not recognized

Rule: from «specific» dictionary to «common» dictionary

```
=# \dF+ pg
                 Configuration "public.pg"
Parser name: "pg_catalog.default"
Locale: 'ru_RU.UTF-8' (default)
    Token     |                      Dictionaries
--------------+-------------------------------------------------------
 file         | pg_catalog.simple
 host         | pg_catalog.simple
 hword        | pg_catalog.simple
 int          | pg_catalog.simple
 lhword       | public.pg_dict,public.en_ispell,pg_catalog.en_stem
 lpart_hword  | public.pg_dict,public.en_ispell,pg_catalog.en_stem
 Lword        | public.pg_dict,public.en_ispell,pg_catalog.en_stem
 nlhword      | pg_catalog.simple
 nlpart_hword | pg_catalog.simple
```



FTS PostgreSQL



FTS in PostgreSQL

- **Dictionary** – is a **program**, which accepts token on input and returns an array of lexems, NULL if token doesn't recognized and empty array for stop-word.
- Dictionary API allows to develop any custom dictionaries
  - Truncate too long numbers
  - Convert colors
  - Convert URLs to canonical way

    http://a.in/a/./index.html → http://a.in/a/index.html
- Built-in dictionary templates
  - ispell, myspell, hunspell
  - snowball stemmer
  - thesaurus
  - synonym
  - simple

Dictionaries

- Dictionary — is a program !

```
=# select ts_lexize('intdict', 11234567890);
  ts_lexize
-----------
 {112345}
=# select ts_lexize('roman', 'XIX');
 ts_lexize
------------
 {19}

=# select ts_lexize('colours','#FFFFFF');
ts_lexize
------------
{white}
```

Astronomical dictionary

Dictionary with regexp support (pcre library)

```
# Messier objects
(M|Messier)(\s|-)?((\d){1,3}) M$3
# catalogs
(NGC|Abell|MKN|IC|H[DHR]|UGC|SAO|MWC)(\s|-)?((\d){1,6}[ABC]?) $1$3
(PSR|PKS)(\s|-)?([JB]?)(\d\d\d\d)\s?([+-]\d\d)\d? $1$4$5
# Surveys
OGLE(\s|-)?((I){1,3}) ogle
2MASS twomass
# Spectral lines
H(\s|-)?(alpha|beta|gamma) h$2
(Fe|Mg|Si|He|Ni)(\s|-)?((\d)|([IXV])+) $1$3
# GRBs
gamma\s?ray\s?burst(s?) GRB
GRB\s?(\d\d\d\d\d\d)([abcd]?) GRB$1$2
```

Agglutinative Languages

German, norwegian, ...
http://en.wikipedia.org/wiki/Agglutinative_language

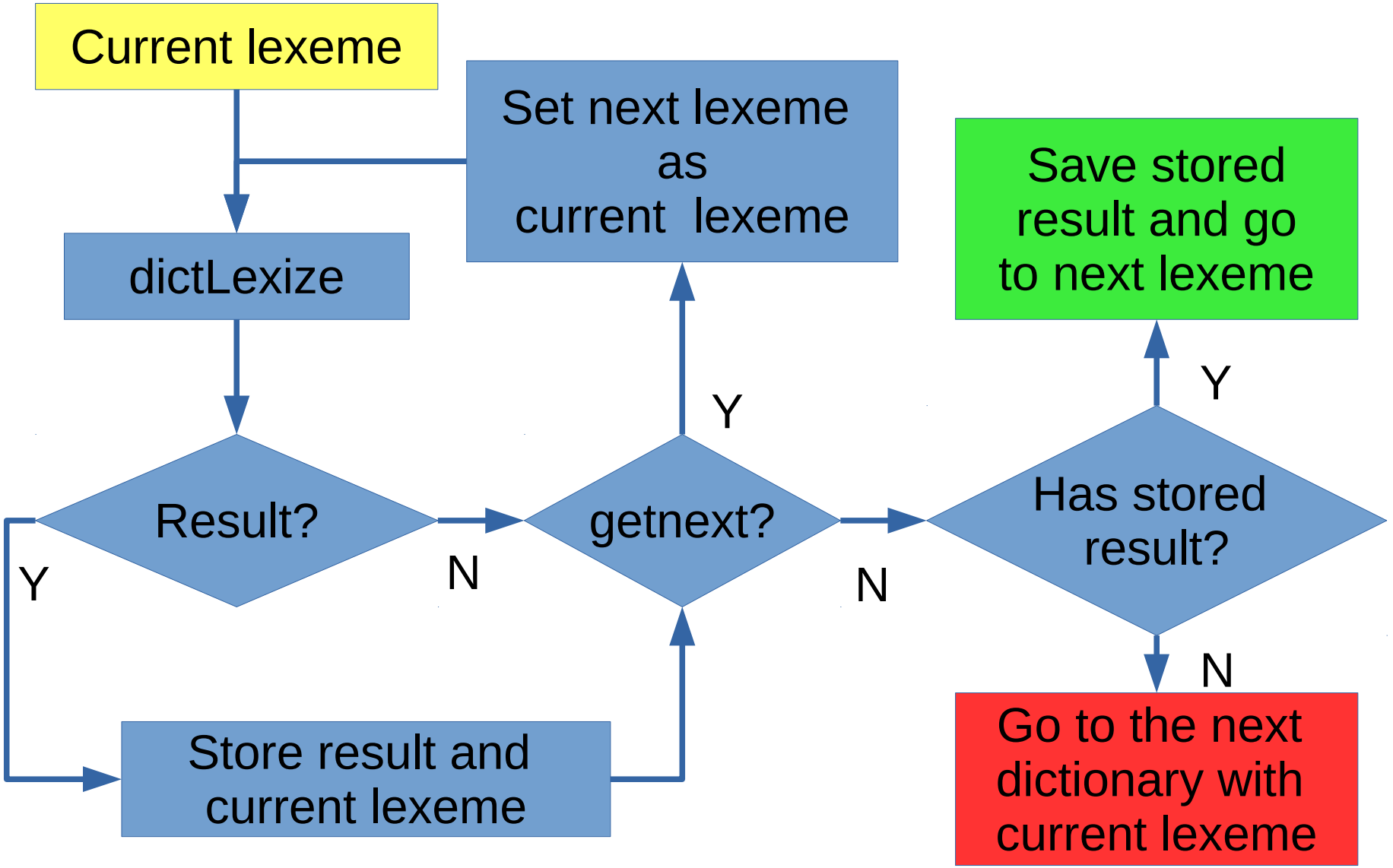
Concatenation of words without space

Query - Fotballklubber

How to find document 'Klubb on fotballfield'

Split words and build search query
'fotbalklubber' =>
' (fotball & klubb) | (fot & ball & klubb) '

Dictionaries – several words



Filter dictionary – unaccent

contrib/unaccent - unaccent text search dictionary and function to remove accents (suffix tree, ~ 25x faster *translate*() solution)

```
1. Unaccent dictionary does nothing and returns NULL.
(lexeme 'Hotels' will be passed to the next dictionary if any)

=# select ts_lexize('unaccent','Hotels') is NULL;
 ?column?
----------
 t

2. Unaccent dictionary removes accent and returns 'Hotel'.
(lexeme 'Hotel' will be passed to the next dictionary if any)

=# select ts_lexize('unaccent','Hôtel');
 ts_lexize
-----------
 {Hotel}
```

Filter dictionary - unaccent

```
CREATE TEXT SEARCH CONFIGURATION fr ( COPY = french );
ALTER TEXT SEARCH CONFIGURATION fr ALTER MAPPING FOR hword, hword_part, word
    WITH unaccent, french_stem;


=# select to_tsvector('fr','Hôtel de la Mer') @@ to_tsquery('fr','Hotels');
 ?column?
----------
 t



=# select ts_headline('fr','Hôtel de la Mer',to_tsquery('fr','Hotels'));
      ts_headline
------------------------
  <b>Hôtel</b> de la Mer
```

Synonym dictionary with prefix search support

```
cat $SHAREDIR/tsearch_data/synonym_sample.syn
postgres        pgsql
postgresql      pgsql
postgre pgsql
gogle   googl
indices index*

=# create text search dictionary syn
( template=synonym,synonyms='synonym_sample');
=# select ts_lexize('syn','indices');
 ts_lexize
-----------
 {index}
```

Synonym dictionary with prefix search support

```
=# create text search configuration tst ( copy=simple);
=# alter text search configuration tst alter mapping
      for asciiword with syn;

=# select to_tsquery('tst','indices');
 to_tsquery
------------
 'index':*
=# select 'indexes are very useful'::tsvector @@
                  to_tsquery('tst','indices');
 ?column?
----------
 t
```

dict_xsyn

- How to search for 'William' and any synonyms 'Will', 'Bill', 'Billy' ? We can:
  - Index only synonyms
  - Index synonyms and original name
  - Index only original name - replace all synonyms. Index size is minimal, but s*earch for specific name is impossible*.

Postgres PROFESSIONAL

dict_xsyn

- Old version of dict_xsyn can return only list of synonyms. It's possible to prepare synonym file to support other options:

```
William Will Bill Billy
Will William Bill Billy
Bill William Will Billy
Billy William Will Bill
```

- New dict_xsyn (Sergey Karpov) allows better control:

```
CREATE TEXT SEARCH DICTIONARY xsyn
(RULES='xsyn_sample', KEEPORIG=false|true,
mode='SIMPLE|SYMMETRIC|MAP');
```

dict_xsyn

- Mode SIMPLE - accepts the original word and returns all synonyms as OR-ed list. This is default mode.
- Mode SYMMETRIC - accepts the original word **or any** of its synonyms, and return all others as OR-ed list.
- Mode MAP - accepts any synonym and returns the original word.

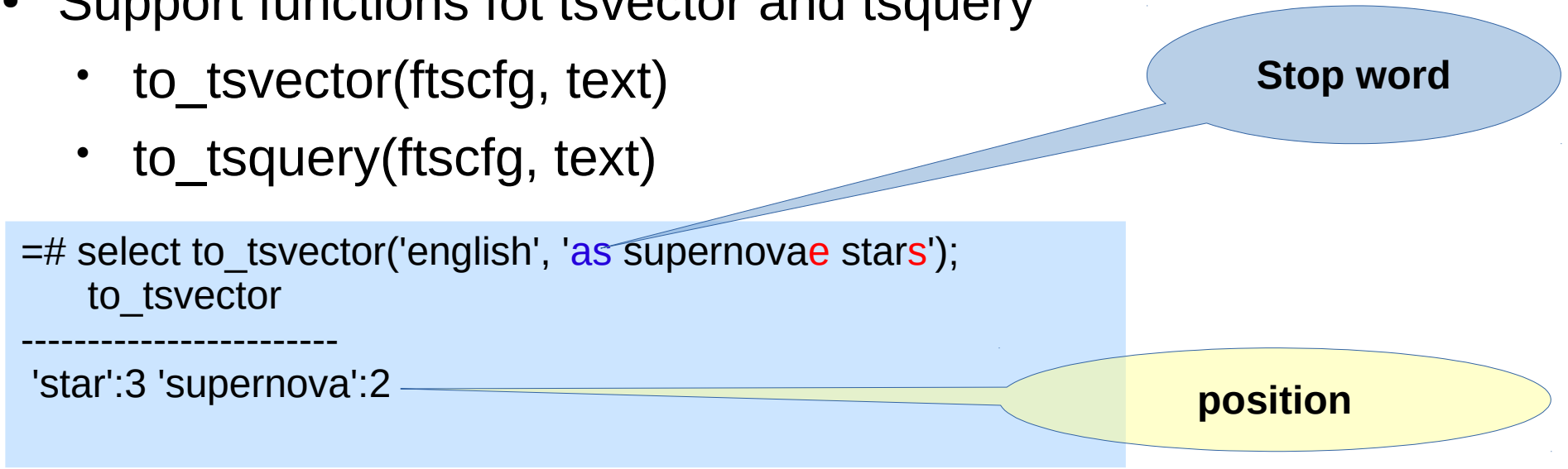
dict_xsyn

EXAMPLES:

```
=# ALTER TEXT SEARCH DICTIONARY xsyn (RULES='xsyn_sample',
KEEPORIG=false, mode='SYMMETRIC');

=# select ts_lexize('xsyn','Will') as Will,
          ts_lexize('xsyn','Bill') as Bill,
          ts_lexize('xsyn','Billy') as Billy;

         will          |         bill          |        billy
-----------------------+-----------------------+----------------------
 {william,bill,billy} | {william,will,billy} | {william,will,bill}

Mode='MAP'

   will    |   bill    |   billy
-----------+-----------+-----------
 {william} | {william} | {william}
```

FTS in PostgreSQL

- Support functions fot tsvector and tsquery
  - to_tsvector(ftscfg, text)
  - to_tsquery(ftscfg, text)




```
=# select * from ts_debug('english', 'a supernovae stars');
   alias   |   description   |   token    |  dictionaries  |  dictionary  |   lexemes
-----------+-----------------+------------+----------------+--------------+-------------
 asciiword | Word, all ASCII | a          | {english_stem} | english_stem | {}
 blank     | Space symbols   |            | {}             |              |
 asciiword | Word, all ASCII | supernovae | {english_stem} | english_stem | {supernova}
 blank     | Space symbols   |            | {}             |              |
 asciiword | Word, all ASCII | stars      | {english_stem} | english_stem | {star}
(5 rows)
```

FTS configuration

- FTS configuration defined:
  - Parser to produce tokens
  - How tokens processed by dictionaries
- SQL commands

```
{CREATE | ALTER | DROP} TEXT SEARCH {CONFIGURATION | DICTIONARY | PARSER}
```

- FTS configurarion supports schema
- Psql commands

  \dF{,d,p}[+] [PATTERN]

- Default FTS configuration- SET `default_text_search_config`

```
SHOW default_text_search_config;
 default_text_search_config
----------------------------
 pg_catalog.russian
(1 row)
```

FTS configuration

- Default FTS configuration-

  SET default_text_search_config
- Assign default_text_search_config to database

  ALTER DATABASE name

    SET default_text_search_config
- SHOW default FTS configuration

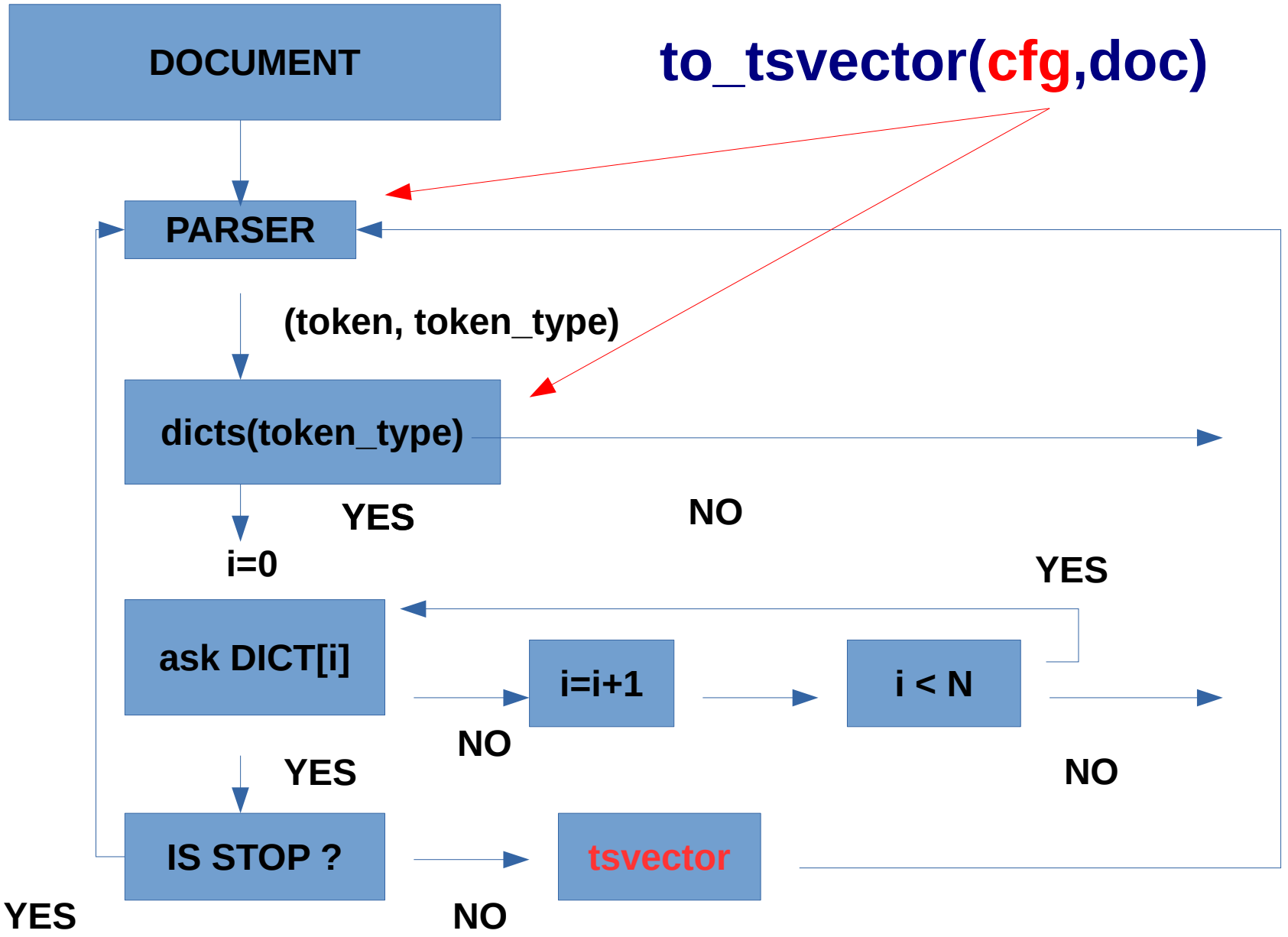
```
SHOW default_text_search_config;
 default_text_search_config
----------------------------
 pg_catalog.russian
(1 row)
```

FTS configuration

16 configurations for 15 languages

```
=# \dF
                   List of text search configurations
   Schema   |    Name    |                Description
------------+------------+-------------------------------------
 pg_catalog | danish     | configuration for danish language
 pg_catalog | dutch      | configuration for dutch language
 pg_catalog | english    | configuration for english language
 pg_catalog | finnish    | configuration for finnish language
 pg_catalog | french     | configuration for french language
 pg_catalog | german     | configuration for german language
 pg_catalog | hungarian  | configuration for hungarian language
 pg_catalog | italian    | configuration for italian language
 pg_catalog | norwegian  | configuration for norwegian language
 pg_catalog | portuguese | configuration for portuguese language
 pg_catalog | romanian   | configuration for romanian language
 pg_catalog | russian    | configuration for russian language
 pg_catalog | simple     | simple configuration
 pg_catalog | spanish    | configuration for spanish language
 pg_catalog | swedish    | configuration for swedish language
 pg_catalog | turkish    | configuration for turkish language
(16 rows)
```

FTS PostgreSQL



Why tsquery

- We want to write complex queries, for example:
  supernovae or white dwarf
  or
  crab nebulae

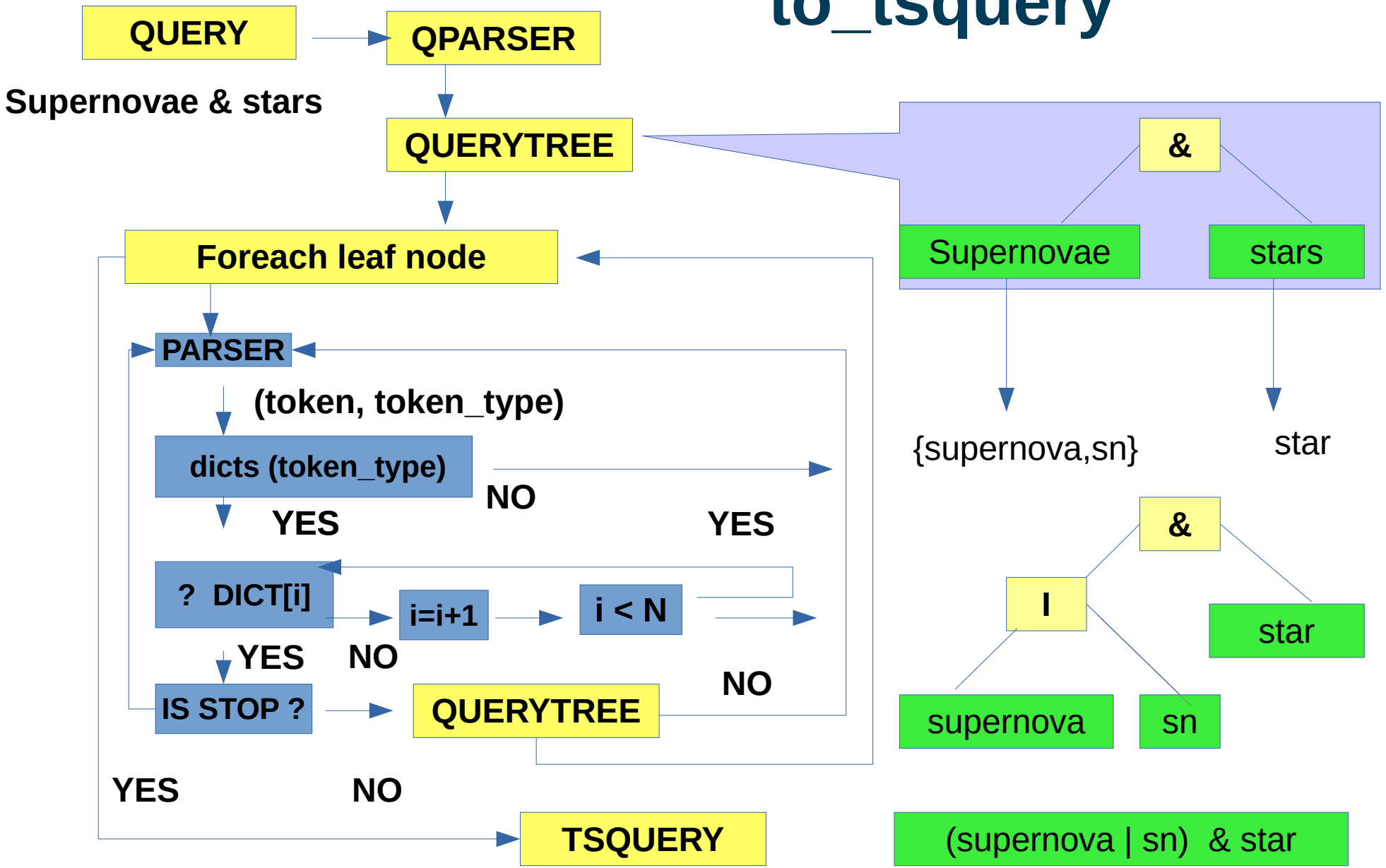
- It's difficult to express in SQL
- Tsquery (text search query) provides compact way

```
'(supernovae | (white & dwarf)) | (crab & nebulae)'
```

- plainto_tsquery() for AND-ed query

```
'white dwarf', 'crab nebulae'
```

FTS PostgreSQL
to_tsquery



Simple Example (no stop words)

```
\dF+ english
Text search configuration "pg_catalog.english"
Parser: "pg_catalog.default"
      Token      | Dictionaries
-----------------+--------------
 asciihword      | english_stem
 asciiword       | english_stem
 email           | simple
 file            | simple
 float           | simple
 host            | simple
 hword           | english_stem
 hword_asciipart | english_stem
 hword_numpart   | simple
 hword_part      | english_stem
 int             | simple
 numhword        | simple
 numword         | simple
 sfloat          | simple
 uint            | simple
 url             | simple
 url_path        | simple
 version         | simple
 word            | english_stem
```

New FTS configuration no stop words

```
CREATE TEXT SEARCH CONFIGURATION english_ns
( COPY = english);

CREATE TEXT SEARCH DICTIONARY en_stem_ns
(TEMPLATE=snowball,LANGUAGE = 'english',
 STOPWORDS='');

SELECT ts_lexize('english_stem', 'the');
 ts_lexize
-----------
 {}
(1 row)
SELECT ts_lexize('en_stem_ns', 'the');
 ts_lexize
-----------
 {the}
(1 row)

ALTER TEXT SEARCH CONFIGURATION english_ns
ALTER MAPPING FOR asciihword,asciiword,hword,
hword_asciipart,hword_numpart,hword_part,word
WITH en_stem_ns;
```

```
\dFd+ english_stem
List of text search dictionaries
-[ RECORD 1 ]+-------------------------------------------
Schema       | pg_catalog
Name         | english_stem
Template     | pg_catalog.snowball
Init options | language = 'english', stopwords = 'english'
Description  | snowball stemmer for english language
```

Simple Example (no stop words)

```
\dF+ english_ns
Text search configuration "public.english_ns"
Parser: "pg_catalog.default"
      Token      | Dictionaries
-----------------+--------------
 asciihword      | en_stem_ns
 asciiword       | en_stem_ns
 email           | simple
 file            | simple
 float           | simple
 host            | simple
 hword           | en_stem_ns
 hword_asciipart | en_stem_ns
 hword_numpart   | en_stem_ns
 hword_part      | en_stem_ns
 int             | simple
 numhword        | simple
 numword         | simple
 sfloat          | simple
 uint            | simple
 url             | simple
 url_path        | simple
 version         | simple
 word            | en_stem_ns
```

New FTS configuration no stop words

```
SELECT to_tsvector('english','the pink elephant');
    to_tsvector
--------------------
 'eleph':3 'pink':2
(1 row)


SELECT to_tsvector('english_ns','the pink elephant');
        to_tsvector
---------------------------
 'eleph':3 'pink':2 'the':1
(1 row)
```

```
\dFd+ en_stem_ns
List of text search dictionaries
-[ RECORD 1 ]+-----------------------------------
Schema       | public
Name         | en_stem_ns
Template     | pg_catalog.snowball
Init options | language = 'english', stopwords = ''
Description  | english stemmer without stop words
```

Simple Example (ispell dictionary)

- *spell dictionaries from Libreoffice can be used for normalization
- Download dictionaries from
  http://extensions.libreoffice.org/extension-center?getCategories=Dictionary
  Let's use en_US (en_US.dict, en_US.affix)

```
CREATE TEXT SEARCH DICTIONARY en_spell
(TEMPLATE=ispell,DictFile=en_US, AFFfile=en_US,
StopWords='english');

SELECT ts_lexize('en_spell','elephant');
 ts_lexize
------------
 {elephant}
(1 row)
```

Simple Example (ispell dictionary)

```
\dF+ english
Text search configuration "pg_catalog.english"
Parser: "pg_catalog.default"
      Token      |     Dictionaries
-----------------+----------------------
 asciihword      | en_spell,english_stem
 asciiword       | en_spell,english_stem
 email           | simple
 file            | simple
 float           | simple
 host            | simple
 hword           | en_spell,english_stem
 hword_asciipart | en_spell,english_stem
 hword_numpart   | en_spell,english_stem
 hword_part      | en_spell,english_stem
 int             | simple
 numhword        | simple
 numword         | simple
 sfloat          | simple
 uint            | simple
 url             | simple
 url_path        | simple
 version         | simple
 word            | en_spell,english_stem
```

Add en_spell dictionary to english config

```
SELECT to_tsvector('english','the pink elephant');
    to_tsvector
--------------------
 'eleph':3 'pink':2
(1 row)

SELECT to_tsvector('english_ns','the pink elephant');
        to_tsvector
----------------------------
 'eleph':3 'pink':2 'the':1
(1 row)

ALTER TEXT SEARCH CONFIGURATION english
ALTER MAPPING FOR asciihword,asciiword,hword,
hword_asciipart,hword_numpart,hword_part,word
WITH en_spell, english_stem;

SELECT to_tsvector('english','the pink elephant');
      to_tsvector
-----------------------
 'elephant':3 'pink':2
(1 row)
```

Default Configuration name

```
SET default_text_search_config = 'english';
SELECT to_tsvector('the pink elephant');
      to_tsvector
-----------------------
 'elephant':3 'pink':2
(1 row)
```

Compound words

Concatenation of words without space:
Rindfleischetikettierungsüberwachungsaufgabenübertragungsgesetz
"beef labelling supervision duty assignment law"

Query - Fotballklubber
Document - `Klubbfotballen`

How to find document ?

Split words and build search query
'fotbalklubber' =>
' (fotball & klubb) | (fot & ball & klubb) '

Dictionaries - TSLexeme->nvariant

Compound words have several variants of splitting:
Word 'foobarcom' (imaginary)

Lexeme	nvariant	Comment
foo	1	
bar	1	
com	1	
foob	2	-a- is an affix (interfix)
rcom	2	

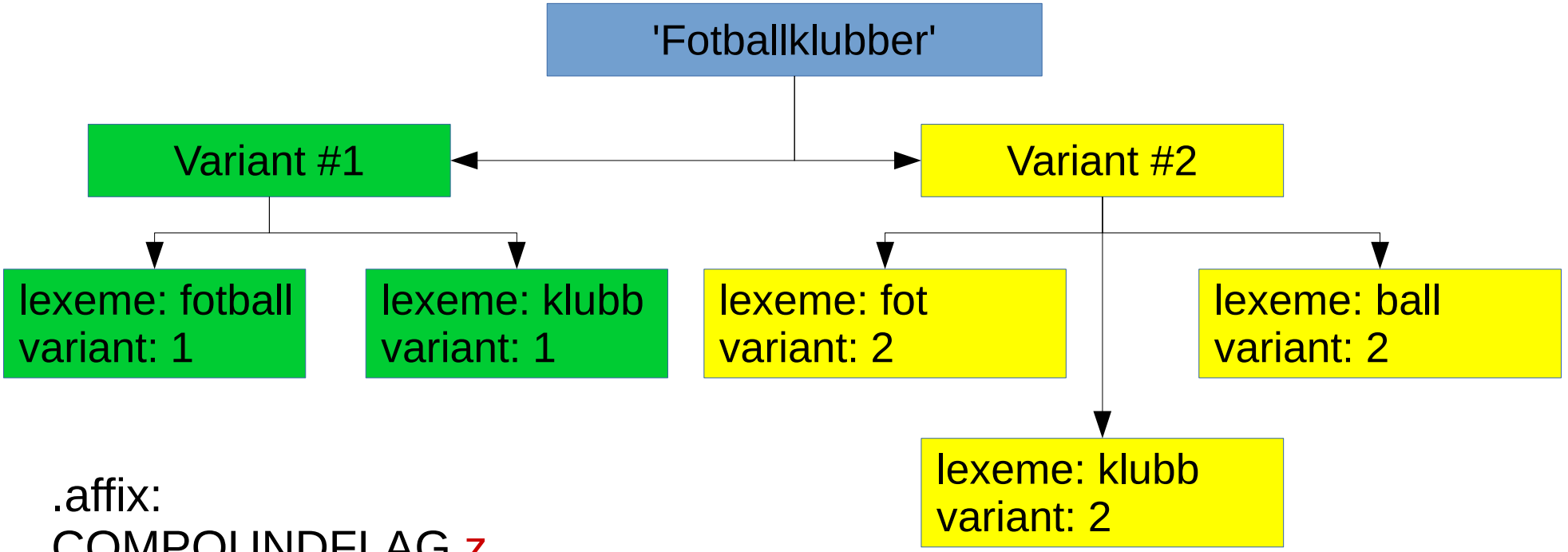
```
tsvector: 'bar:1 com:1 foo:1 foob:1 rcom:1'
tsquery: '(foob & rcom) | (foo & bar & com)'
```

Example: Compound words

- Norwegian language:
  - How to match document 'Klubbfotballen' with query 'Fotballklubber ?
  - 'Klubbfotballen' used also as 'Klubb' and 'fotballen', 'Fotballklubber' — 'Fotball' and 'klubber'.
  - Thanks to FLAG z in ispell we can do the same

Hunspell. Compound words

Lexemes generation scheme:



.affix:
COMPOUNDFLAG z

.dict:
ball/ABCDFVz
fot/ACDJVz
fotball/ADFz
klubb/ABCDEFGVz

Example: Compound words

- Norwegian language:
  - Download from
    http://extensions.libreoffice.org/extension-center?getCategories=Dictionary
  - unzip dictionary-no-no-1.0.oxt
  - Convert to UTF8 !!!

```
iconv -f ISO8859-1 -t utf8 nn_NO.aff > nn_NO.affix
iconv -f ISO8859-1 -t utf8 nn_NO.dic > nn_NO.dict
```

  - CREATE dictionary nn_spell

```
CREATE TEXT SEARCH DICTIONARY nn_spell
(TEMPLATE=ispell,DictFile=nn_NO, AFFfile=nn_NO,
StopWords='norwegian');
SELECT ts_lexize('nn_spell', 'Fotballklubber');
           ts_lexize
---------------------------------
 {fotball,klubb,fot,ball,klubb}
(1 row)
```

Example: Compound words

- Norwegian language
  - ADD nn_spell to norwegian configuration

```
ALTER TEXT SEARCH CONFIGURATION norwegian ALTER MAPPING
FOR asciihword,asciiword,hword,hword_asciipart,
hword_numpart,hword_part,word WITH nn_spell, norwegian_stem;


SELECT to_tsquery('norwegian', 'Fotballklubber');
                    to_tsquery
-----------------------------------------------
 'fotball' & 'klubb' | 'fot' & 'ball' & 'klubb'
(1 row)
```

Example: Compound words

- Norwegian language

```
SELECT to_tsvector('norwegian', 'Klubbfotballen');
                            to_tsvector
-------------------------------------------------------------------
 'ball':1 'balle':1 'fot':1 'fotball':1 'klubb':1 'klubbfotball':1


WOW !!!

SELECT to_tsvector('norwegian', 'Klubb fotballen') @@
       to_tsquery('norwegian', 'Fotballklubber');
 ?column?
----------
 t
(1 row)
```

FTS works, but it's SLOW...

- Huge ACID overhead
  - Results should be consistent
  - Transactions support
- Sequential Scan of the whole table

Index — silver bullet !

the only weapon that is effective against a werewolf, witch, or other monsters.

Indexes !

- Index is a search tree with tuple pointers in the leaves
- Index has no visibility information (MVCC !)
- Indexes used only for accelerations
- Index scan should produce the same results as sequence scan with filtering
- Indexes can be: **partial** (where price > 0.0), **functional** (to_tsvector(text)), **multicolumn** (timestamp, tsvector)
- Indexes not always useful !
  - Low selectivity
  - Maintainance overhead

PostgreSQL extendability

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.

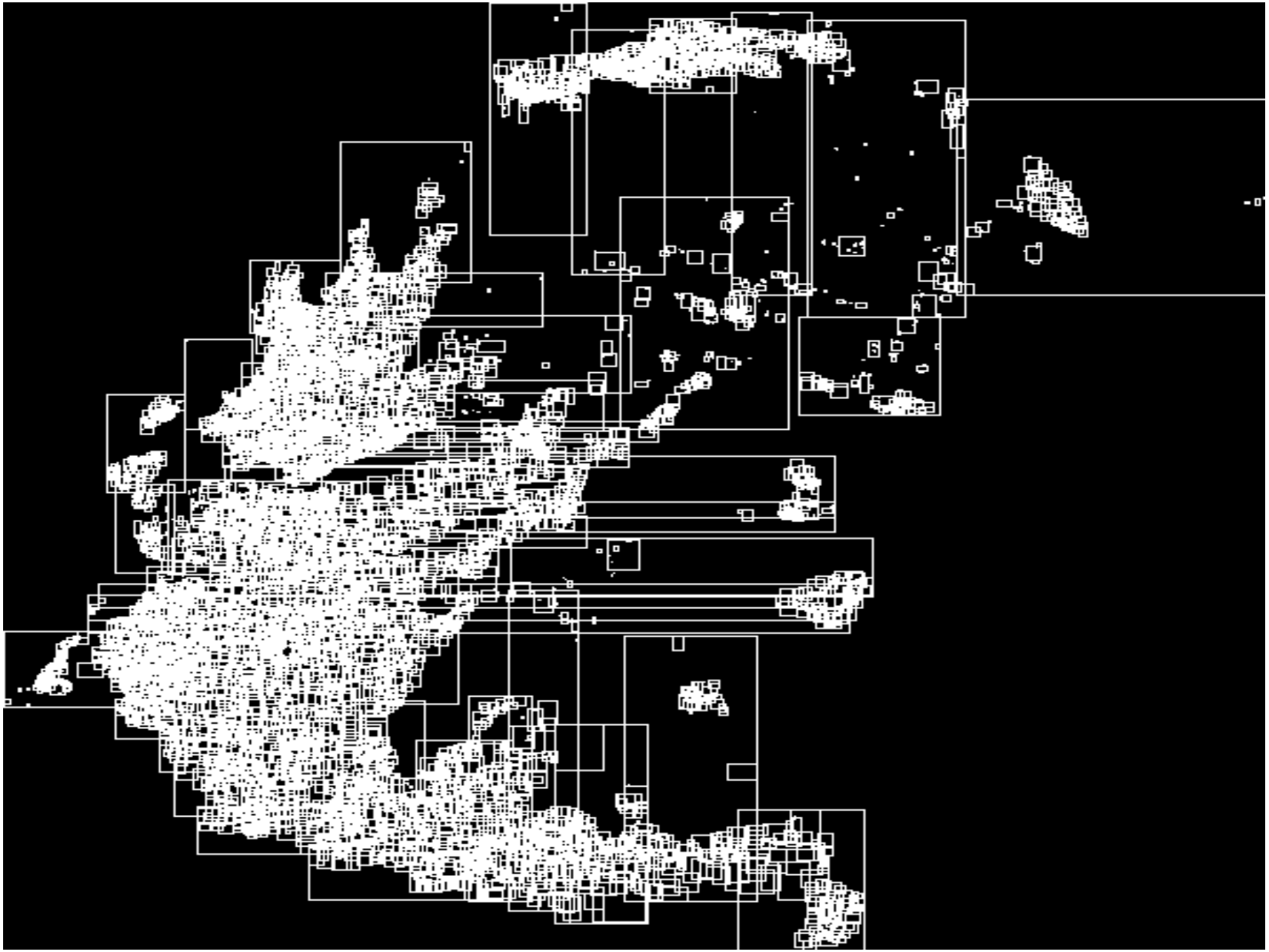
Extendability in POSTGRES , IEEE Data Eng. Bull. 10 (2) pp.16-23, 1987

GiST

- GiST — Generalized Search Tree

  J. M. Hellerstein, J. F. Naughton, and Avi Pfeffer. "Generalized search trees for database systems.", VLDB 21, 1995
- Containment hierarchy
  - Parent node contains all its childrens
- No order within pages
- Key ranges of pages can overlap
- GiST provides standard methods
  - Tree navigation (also, knn)
  - Recovery and Concurrency
- Easy to write custom acces methods

GiST — Rtree example (Greece)



Understanding GiST (array example)

- Intarray -Access Method for array of integers
  - Operators overlap, contains

S1 = {1,**2**,3,5,6,**9**}
S2 = {1,**2**,5}
S3 = {0,5,6,**9**}
S4 = {1,4,5,8}
S5 = {0,9}
S6 = {3,5,6,7,8}
S7 = {4,7,**9**}

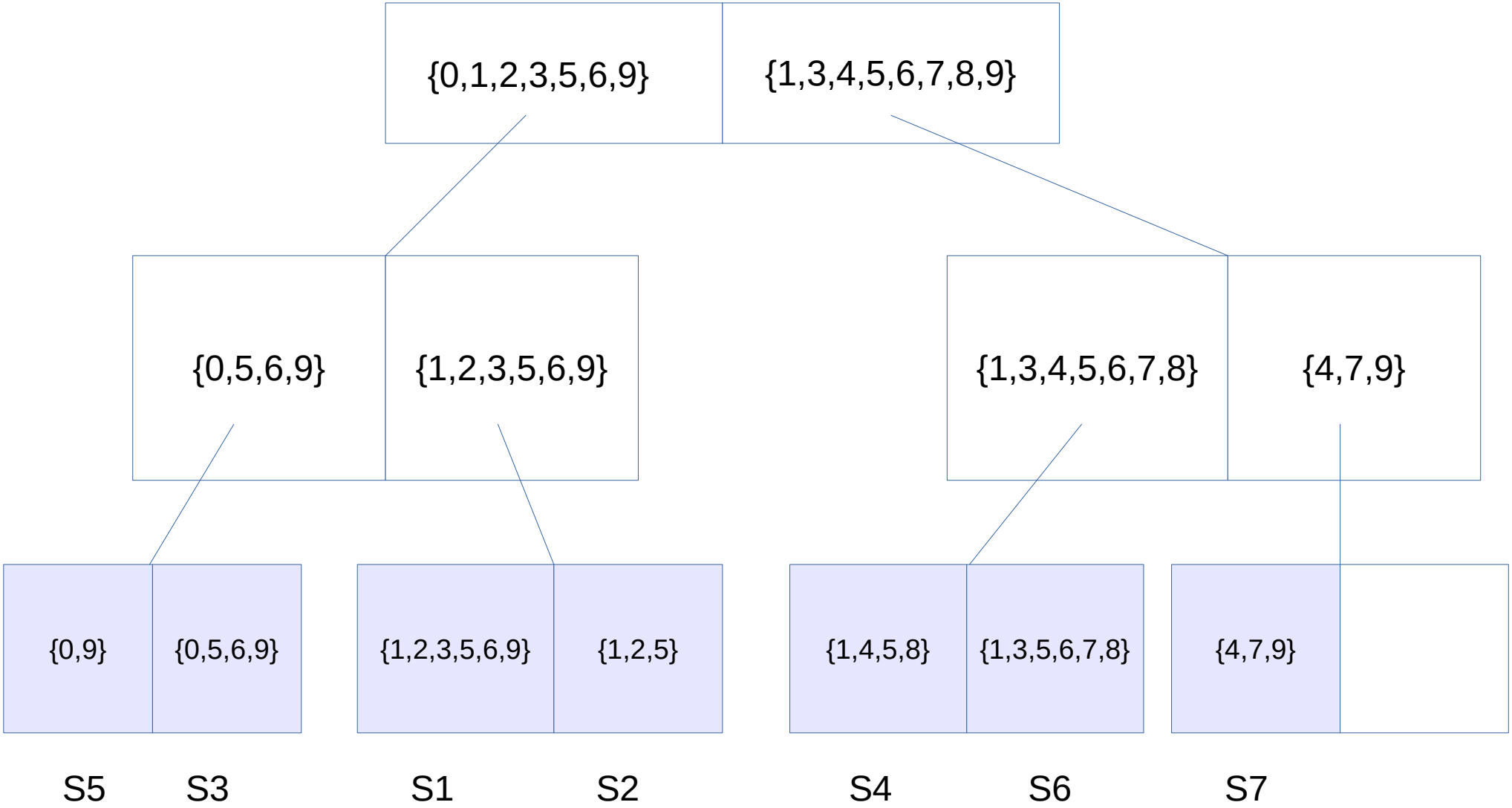
Q = {**2,9**}

Russian Doll Tree

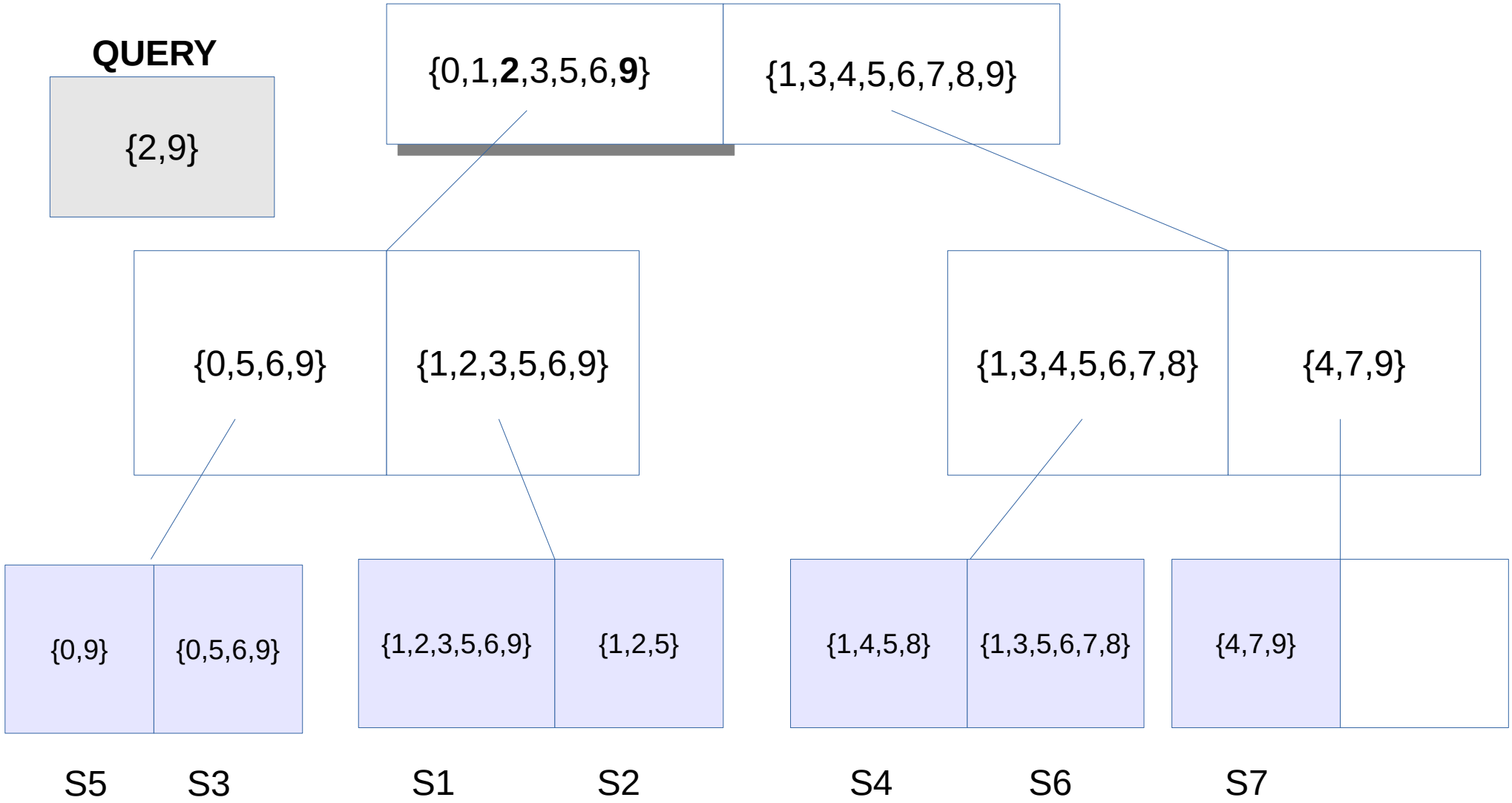
"THE RD-TREE: AN INDEX STRUCTURE FOR SETS", Joseph M. Hellerstein

RD-Tree

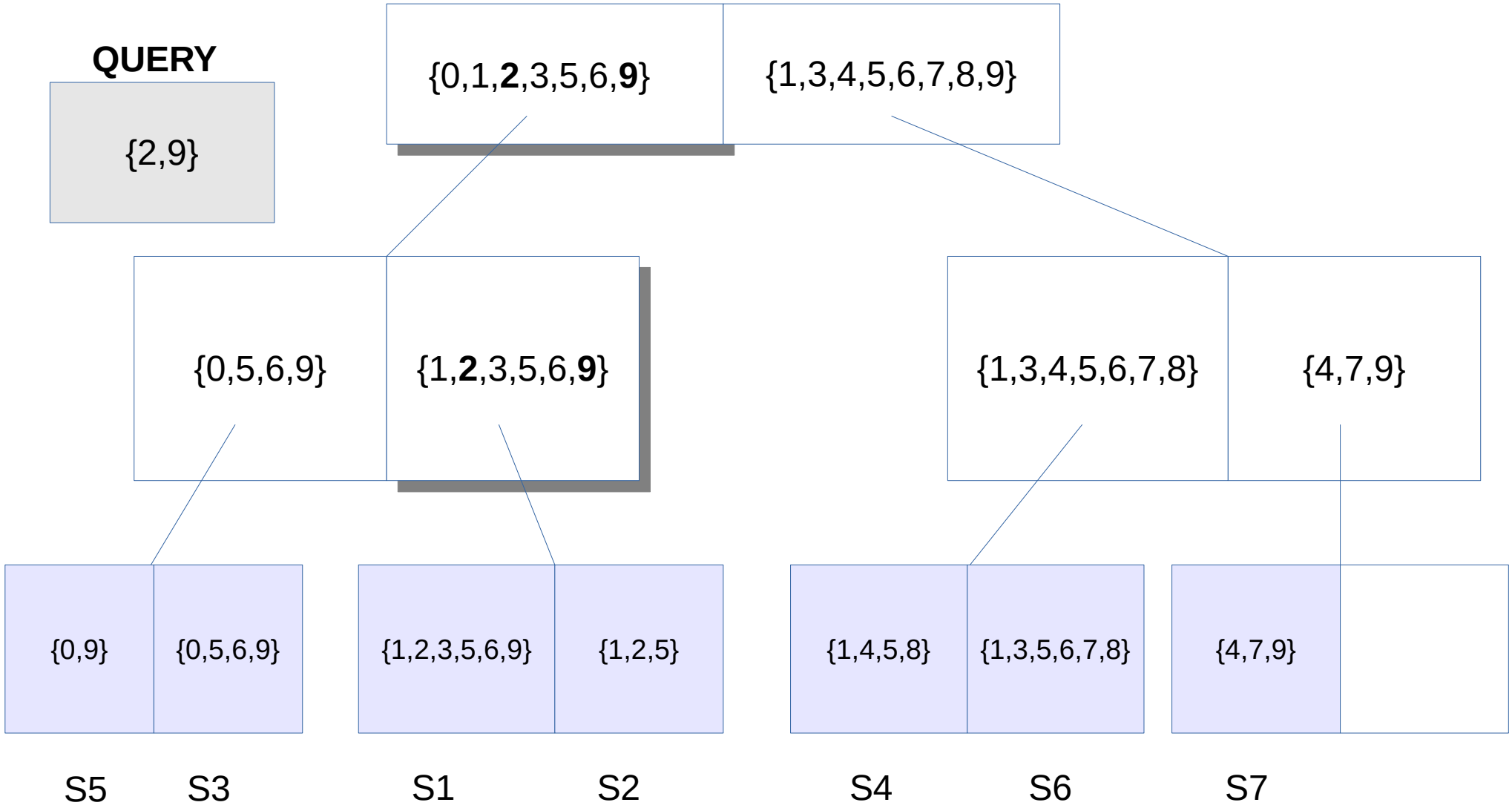
Containment Hierarchy



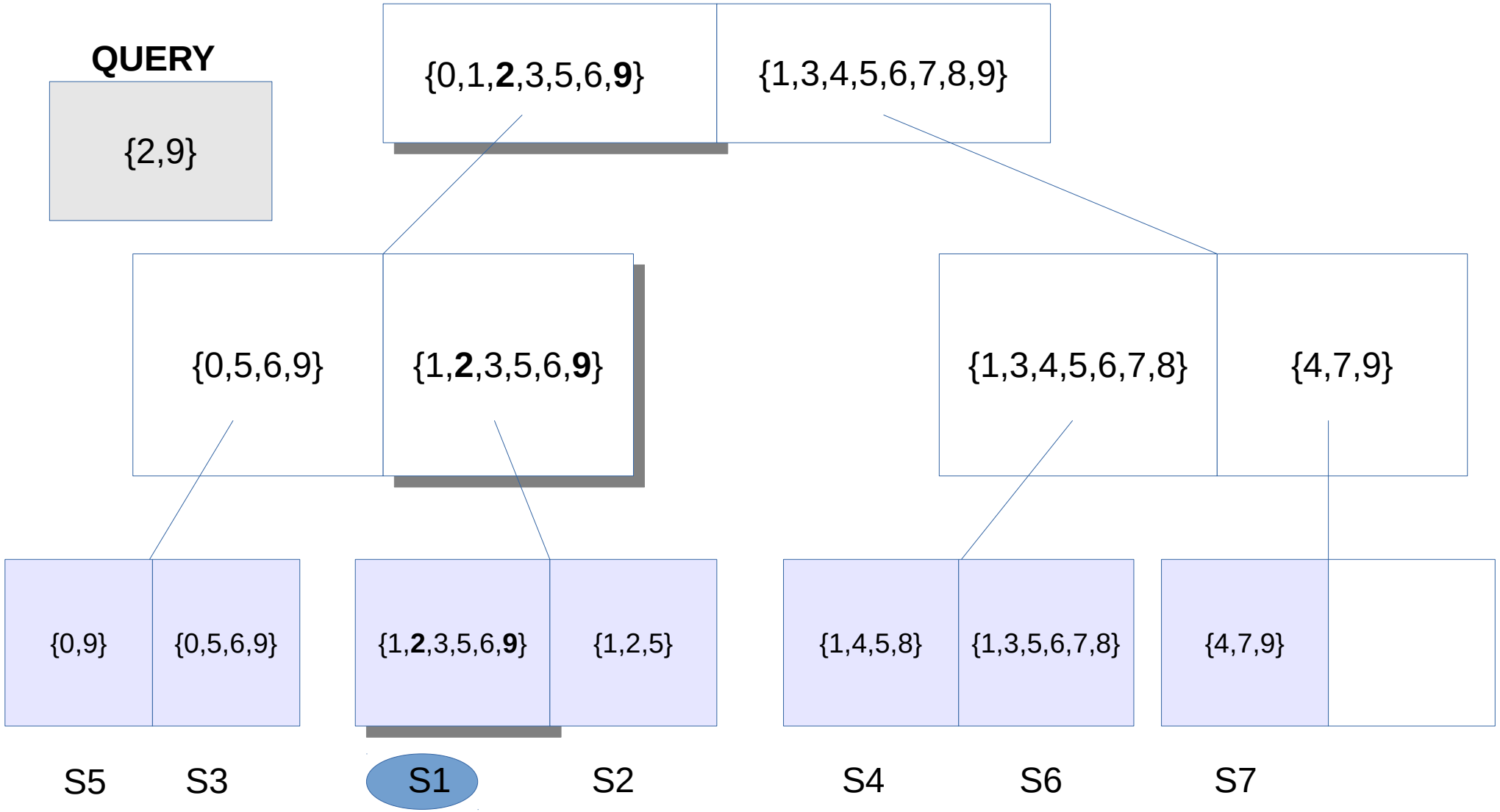
RD-Tree


QUERY
{2,9}
{0,1,2,3,5,6,9}
{1,3,4,5,6,7,8,9}
{0,5,6,9}
{1,2,3,5,6,9}
{1,3,4,5,6,7,8}
{4,7,9}
{0,9}
{0,5,6,9}
{1,2,3,5,6,9}
{1,2,5}
{1,4,5,8}
{1,3,5,6,7,8}
{4,7,9}
S5
S3
S1
S2
S4
S6
S7

RD-Tree


QUERY
{2,9}
{0,1,2,3,5,6,9}
{1,3,4,5,6,7,8,9}
{0,5,6,9}
{1,2,3,5,6,9}
{1,3,4,5,6,7,8}
{4,7,9}
{0,9}
{0,5,6,9}
{1,2,3,5,6,9}
{1,2,5}
{1,4,5,8}
{1,3,5,6,7,8}
{4,7,9}
S5
S3
S1
S2
S4
S6
S7

RD-Tree


QUERY
{2,9}
{0,1,2,3,5,6,9}
{1,3,4,5,6,7,8,9}
{0,5,6,9}
{1,2,3,5,6,9}
{1,3,4,5,6,7,8}
{4,7,9}
{0,9}
{0,5,6,9}
{1,2,3,5,6,9}
{1,2,5}
{1,4,5,8}
{1,3,5,6,7,8}
{4,7,9}
S5
S3
S1
S2
S4
S6
S7

FTS Index (GiST): RD-Tree

- Word signature — words hashed to the specific position of '1'

  w1 -> S1: 01000000 Document: w1 w2 w3

  w2 -> S2: 00010000

  w3 -> S3: 10000000

- Document (query) signature — superposition (bit-wise OR) of signatures

  S: 11010000

- Bloom filter

  Q1: 00000001 – exact not

  Q2: 01010000 - may be contained in the document, **false drop**

- Signature is a lossy representation of document
  - + fixed length, compact, + fast bit operations
  - - lossy (false drops), - saturation with #words grows

FTS Index (GiST): RD-Tree

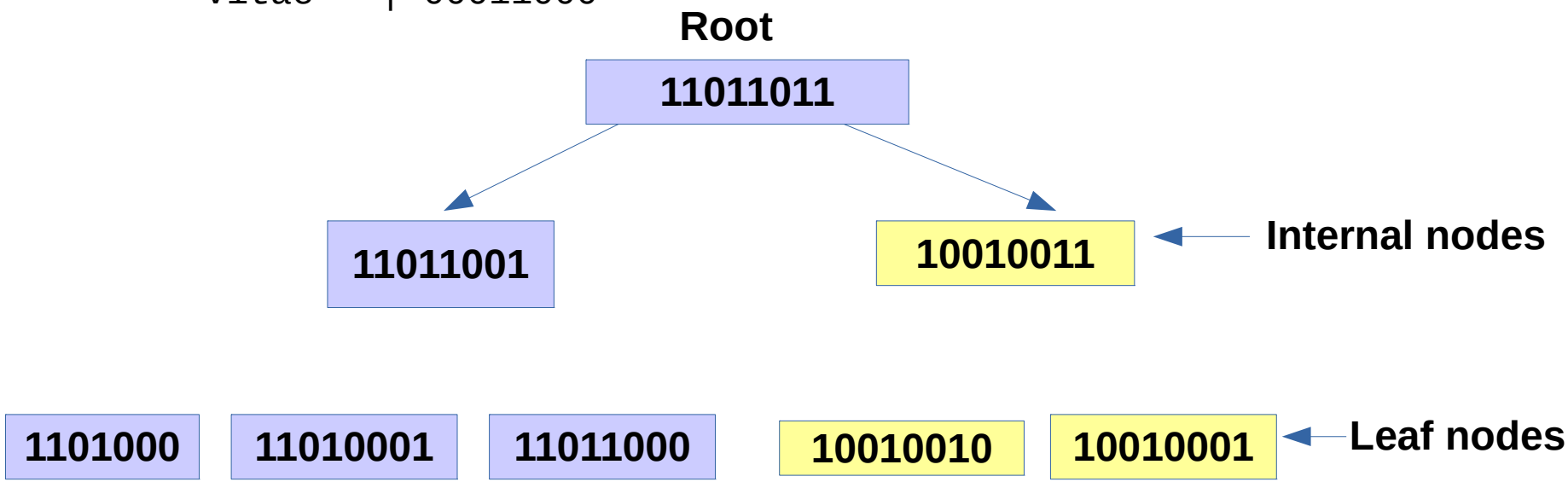
- Latin proverbs

```
 id |           proverb
----+--------------------------
  1 | Ars longa, vita brevis
  2 | Ars vitae
  3 | Jus vitae ac necis
  4 | Jus generis humani
  5 | Vita nostra brevis
```

FTS Index (GiST): RD-Tree

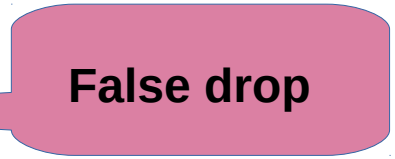
word	signature
ac	00000011
ars	**11000000**
brevis	00001010
generis	01000100
humani	00110000
jus	00010001
longa	00100100
necis	01001000
nostra	10000001
vita	01000001
vitae	00011000

QUERY



RD-Tree (GiST)

id	proverb	signature
1	Ars longa, vita brevis	11101111
2	Ars vitae	11011000
3	Jus vitae ac necis	01011011
4	Jus generis humani	01110101
5	Vita nostra brevis	11001011



RD-Tree (GiST)

- Problems
  - Not good scalabilty with increasing of cardinality of words and records.
  - Index is lossy, need check for false drops

    (Recheck в EXPLAIN ANALYZE)

GIN
Generalized Inverted Index

PROFESSIONAL Postgres

GIN

Report Index

A

B

D

E

GIN

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

B

backward wave oscillators, 45

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

E

PROFESSIONAL Postgres

No positions in index !

Inverted Index in PostgreSQL

ENTRY TREE

Report Index

A

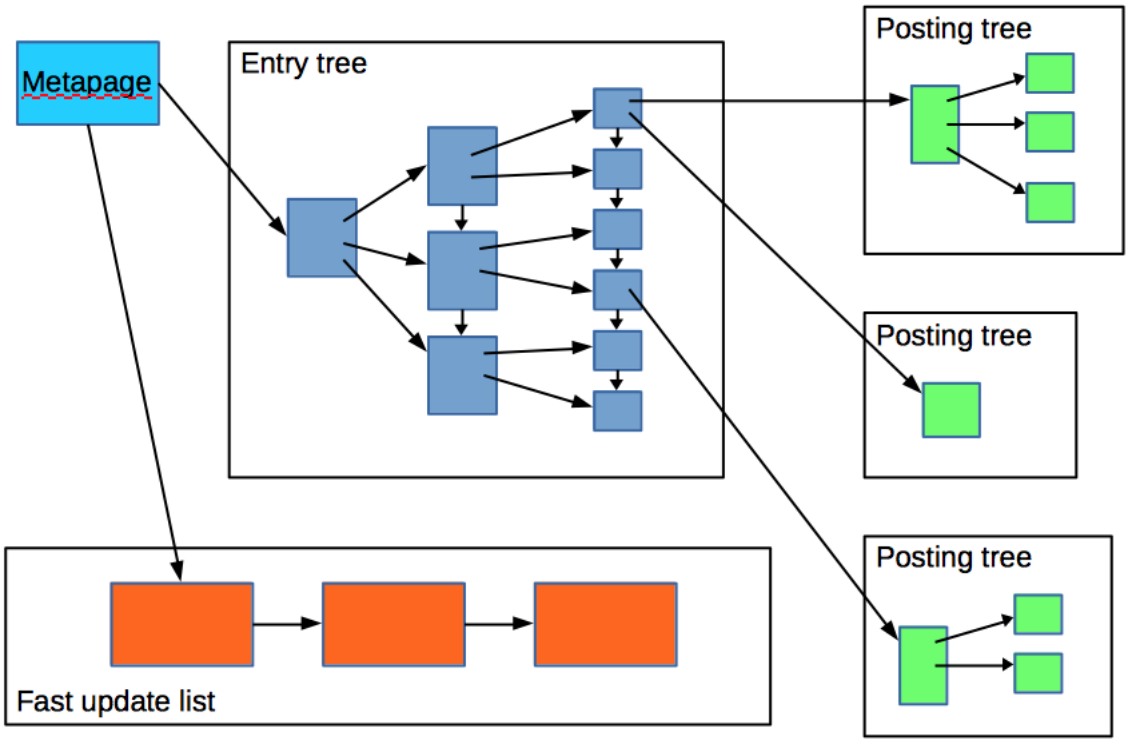
abrasives, 27
acceleration measurement, 58
accelerometers, 5, 10, 25, 28, 30, 36, 58, 59, 61, 73, 74
actuators, 4, 37, 46, 49
adaptive Kalman filters, 60, 61
adhesion, 63, 64
adhesive bonding, 15
adsorption, 44
aerodynamics, 29
aerospace instrumentation, 61
aerospace propulsion, 52
aerospace robotics, 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

Posting list
Posting tree

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

B

backward wave oscillators, 45



GIN

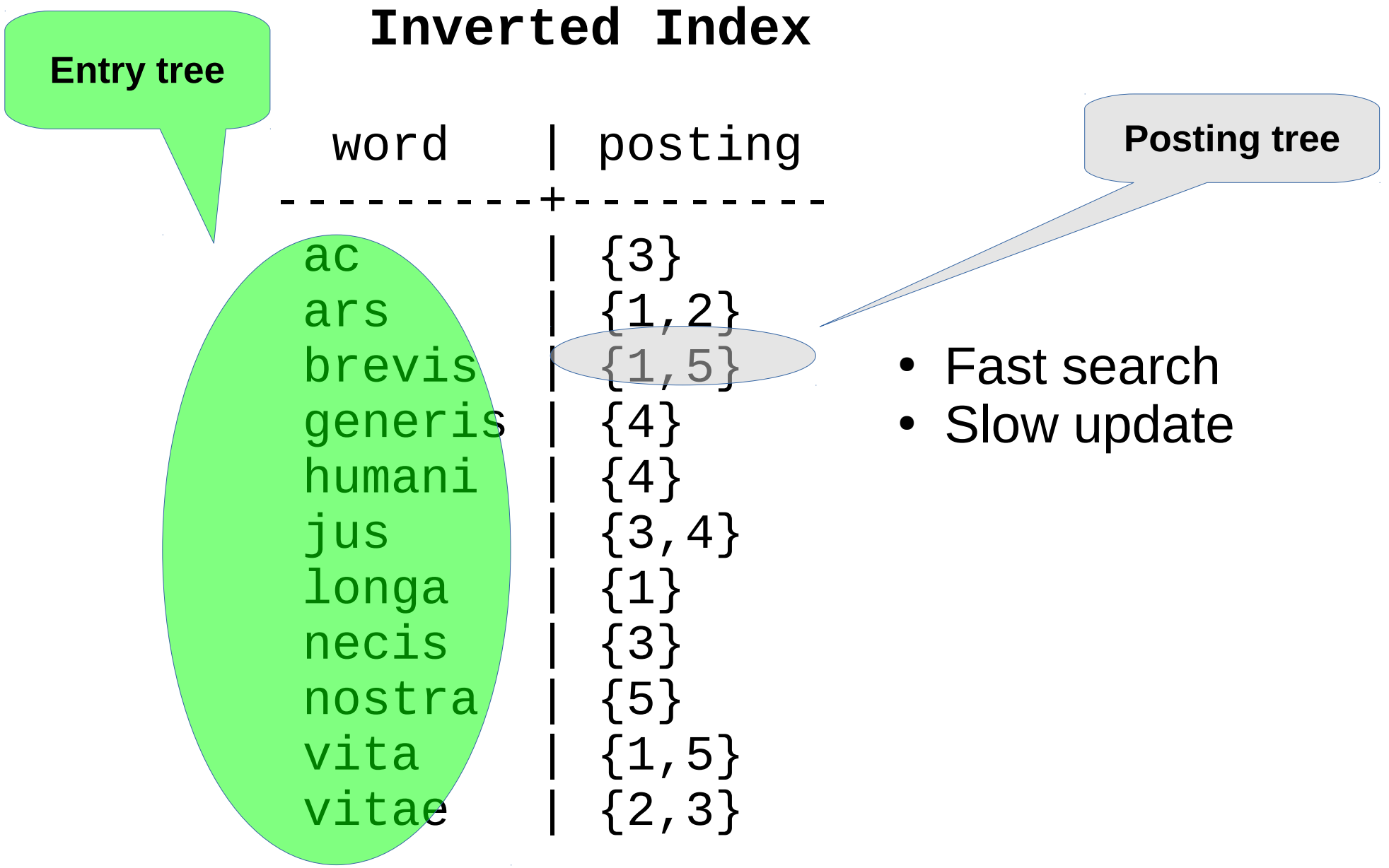
- Internal structure is basically just a B-tree
  - Optimized for storing a lot of duplicate keys
  - Duplicates are ordered by heap TID
- Interface supports indexing more than one key per indexed value
  - Full text search: “foo bar” → “foo”, “bar”
- Bitmap scans only

GIN Index

Demo collections – latin proverbs

```
 id |          proverb
----+--------------------------
  1 | Ars longa, vita brevis
  2 | Ars vitae
  3 | Jus vitae ac necis
  4 | Jus generis humani
  5 | Vita nostra brevis
```

GIN Index


Inverted Index
Entry tree
Posting tree
word | posting
ac | {3}
ars | {1,2}
brevis | {1,5}
generis | {4}
humani | {4}
jus | {3,4}
longa | {1}
necis | {3}
nostra | {5}
vita | {1,5}
vitae | {2,3}
• Fast search
• Slow update

GiST, GIN applications

- Array of integers (GiST, GIN)
- Full text search (GiST, GIN)
- Hierarchical data (GiST)
- Similarity search (GiST, GIN)
- Rtree (GiST)
- PostGIS (postgis.org) (GiST) — spatial index
- BLASTgres (GiST) — bioinformatics
- Cube (GiST)
- Many other .....

Postgres PROFESSIONAL

Full-text search tips

- Stable to_tsquery
- Find documents with specific token type
- Getting words from tsvector
- Confuse with text search
- Antimat constraint
- Prefix search
- APOD example (ts_headline, query rewriting)
- FTS without tsvector column
- Strip tsvector
- Fast approximated statistics
- To be or not to be
- Inverse FTS
- Refined Search

Stable to_tsquery

Result of to_tsquery() can't be used as a cache key,
since to_tsquery() doesn't preserves an order,
which isn't good for cacheing.

Little function helps:

```
CREATE OR REPLACE FUNCTION stable_ts_query(tsquery)
RETURNS tsquery AS
$$
   SELECT ts_rewrite( $1 , 'dummy_word', 'dummy_word');
$$
LANGUAGE SQL RETURNS NULL ON NULL INPUT IMMUTABLE;
```

Note: Remember about text search
configuraton to have really good cache key !

Find documents by token type

How to find documents, which contain emails ?

```
CREATE OR REPLACE FUNCTION document_token_types(text)
RETURNS _text AS
$$

SELECT ARRAY (
    SELECT
        DISTINCT alias
    FROM
        ts_token_type('default') AS tt,
        ts_parse('default', $1) AS tp
    WHERE
        tt.tokid = tp.tokid
    );
$$ LANGUAGE SQL immutable;
```

Find documents by token type

```
=# SELECT document_token_types(title) FROM papers
        LIMIT 10;
                        document_token_types
-----------------------------------------------------------------
 {asciihword,asciiword,blank,hword_asciipart}
 {asciiword,blank}
 {asciiword,blank}
 {asciiword,blank}
 {asciiword,blank}
 {asciiword,blank,float,host}
 {asciiword,blank}
 {asciihword,asciiword,blank,hword_asciipart,int,numword,uint}
 {asciiword,blank}
 {asciiword,blank}
(10 rows)

CREATE INDEX fts_types_idx ON papers USING
                          gin( document_token_types (title) );
```

Find documents by token type

How to find documents, which contain emails ?

```
SELECT comment FROM papers
WHERE document_token_types(title) && '{email}';
```

The list of available token types:

```
SELECT * FROM ts_token_type('default');
```

Getting words from tsvector

```
CREATE OR REPLACE FUNCTION ts_stat(tsvector, OUT word text,
OUT ndoc integer, OUT nentry integer)
RETURNS SETOF record AS $$
SELECT ts_stat('SELECT ' || quote_literal( $1::text )
                                || '::tsvector');
$$ LANGUAGE SQL RETURNS NULL ON NULL INPUT IMMUTABLE;

SELECT id, (ts_stat(fts)).* FROM apod WHERE id=1;
 id |    word     | ndoc | nentry
----+-------------+------+--------
  1 | 1           |    1 |      1
  1 | 2           |    1 |      2
  1 | io          |    1 |      2
  1 | may         |    1 |      1
  1 | new         |    1 |      1
  1 | red         |    1 |      1
  1 | two         |    1 |      1
```

Confuse with text search

One expected **true** here, but result is disappointing **false**

```
=# select to_tsquery('ob_1','inferences') @@
                to_tsvector('ob_1','inference');
 ?column?
----------
 f
```

Use ts_debug() to understand the problem

```
'inferences':
{french_ispell,french_stem} | french_stem   | {inferent}

'inference':
{french_ispell,french_stem} | french_ispell | {inference}
```

Confuse with text search

- Use synonym dictionary as a first dictionary {synonym,french_ispell,french_stem}

  with rule '`inferences inference`'
  - Don't forget to reindex !
- Use ts_rewrite()
  - Don't need to reindex

Antimat constraint

```
CREATE TABLE nomat (i int, t text,
   CHECK (NOT (to_tsvector(t) @@ 'f.ck'::tsquery))
);
=# INSERT INTO nomat(i,t) VALUES(1,'f.ck him');
ERROR: new row for relation "nomat" violates check
constraint "nomat_t_check"
DETAIL: Failing row contains (1, f.ck him).
=# INSERT INTO nomat(i,t) VALUES(1,'f.cking him');
ERROR: new row for relation "nomat" violates check
constraint "nomat_t_check"
DETAIL: Failing row contains (1, f.cking him).
=# INSERT INTO nomat(i,t) VALUES(1,'kiss him');
INSERT 0 1
```

Prefix full-text search support

- Prefix full-text search support
  - to_tsquery('supernov:*') - match all documents, which contains words with prefix 'supernov'
  - to_tsquery('supernov:ab*') - the same,but only in titles (weight 'a') and keywords (weight 'b')
  - Can use new GIN partial match feature to speedup search
  - Can be useful if there is no stemmer available

APOD example

http://www.astronet.ru/db/apod.html

- curl -O http://www.sai.msu.su/~megera/postgres/fts/apod.dump.gz
- zcat apod.dump.gz | psql postgres
- psql postgres

```
postgres=# \d apod
        Table "public.apod"
  Column  |    Type    | Modifiers
----------+------------+-----------
 id       | integer    | not null
 title    | text       |
 body     | text       |
 sdate    | date       |
 keywords | text       |

postgres=# show default_text_search_config;
    default_text_search_config
-------------------------------
 pg_catalog.russian
```

APOD example: FTS configuration

```
=# \dF+ russian
Text search configuration
"pg_catalog.russian"
Parser: "pg_catalog.default"
      Token      | Dictionaries
-----------------+--------------
 asciihword      | english_stem
 asciiword       | english_stem
 email           | simple
 file            | simple
 float           | simple
 host            | simple
 hword           | russian_stem
 hword_asciipart | english_stem
 hword_numpart   | simple
 hword_part      | russian_stem
 int             | simple
 numhword        | simple
 numword         | simple
 sfloat          | simple
 uint            | simple
 url             | simple
 url_path        | simple
 version         | simple
 word            | russian_stem
```

APOD example: FTS index

```
postgres=# alter table apod add column fts tsvector;
postgres=# update apod  set fts=
setweight( coalesce( to_tsvector(title),''),'B')          ||
setweight( coalesce( to_tsvector(keywords),''),'A') ||
setweight( coalesce( to_tsvector(body),''),'D');
```

NULL || nonNULL => NULL

```
postgres=# create index apod_fts_idx on apod using gin(fts);
postgres=# vacuum analyze apod;

postgres=# select title from apod where fts  @@
                  plainto_tsquery('supernovae stars') limit 5;
                   title
-----------------------------------------
 Runaway Star
 Exploring The Universe With IUE 1978-1996
 Tycho Brahe Measures the Sky
 Unusual Spiral Galaxy M66
 COMPTEL Explores The Radioactive Sky
```

APOD example: Search

```
SELECT title,ts_rank_cd(fts, q) AS rank FROM apod,
to_tsquery('supernovae & x-ray') q
WHERE fts  @@ q ORDER BT rank DESC LIMIT 5;
```

title	rank
Supernova Remnant E0102-72 from Radio to X-Ray	1.59087
An X-ray Hot Supernova in M81	1.47733
X-ray Hot Supernova Remnant in the SMC	1.34823
Tycho's Supernova Remnant in X-ray	1.14318
Supernova Remnant and Neutron Star	1.08116

(5 rows)

Time: 1.965 ms

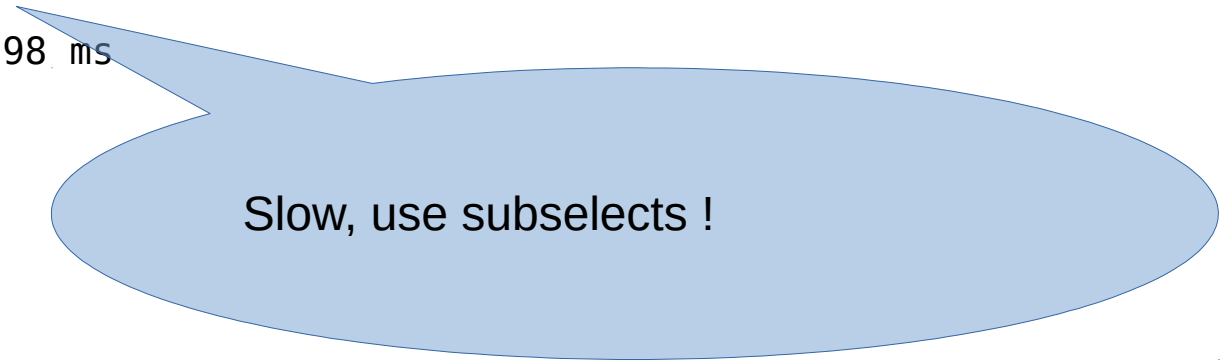
ts_rank_cd uses only local information

0 < rank/(rank+1) < 1

ts_rank_cd('{0.1, 0.2, 0.4, 1.0 }',fts, q)
D C B A

APOD example: headline

```
SELECT ts_headline(body,q,'StartSel=<,StopSel=>,MaxWords=10,MinWords=5'),
ts_rank_cd(fts, q) as rank from apod, to_tsquery('supernovae & x-ray') q
WHERE fts  @@  q ORDER By rank DESC LIMIT 5;

                                 headline                                  | rank
---------------------------------------------------------------------------+----------
 <supernova> remnant  E0102-72, however, is giving astronomers a clue      | 1.59087
 <supernova> explosion. The picture was taken in  <X>-<rays>               | 1.47733
  <X>-<ray> glow is produced by  multi-million degree                      | 1.34823
  <X>-<rays> emitted by this shockwave made by a telescope                 | 1.14318
  <X>-<ray> glow. Pictured is the <supernova>                              | 1.08116
(5 rows)

Time: 39.298 ms
```



APOD example

- Different searches with one index
  - Search only in titles (label 'b')

```
=# SELECT title,ts_rank_cd(fts, q) AS rank FROM apod,
to_tsquery('supernovae:b & x-ray') q
WHERE fts  @@ q ORDER BT rank_cd DESC LIMIT 5;

                    title                          | rank
---------------------------------------------------+----------
 Supernova Remnant E0102-72 from Radio to X-Ray | 1.59087
 An X-ray Hot Supernova in M81                     | 1.47733
 X-ray Hot Supernova Remnant in the SMC            | 1.34823
 Tycho's Supernova Remnant in X-ray                | 1.14318
 Supernova Remnant and Neutron Star                | 1.08116
(5 rows)
```

to_tsquery('supernovae:ab') - search in titles and keywords

FTS without tsvector column

- Use functional index (GiST or GiN)
  - no ranking, use other ordering

```
create index gin_text_idx on test using gin (
( coalesce(to_tsvector(title),'') || coalesce(to_tsvector(body),'') )
);
```

```
apod=# select title from test where
(coalesce(to_tsvector(title),'') || coalesce(to_tsvector(body),'') ) @@
to_tsquery('supernovae') order by sdate desc limit 10;
```

FTS tips

- ts_headline() is slow functions, use **subselect**

790 times

```
select id,ts_headline(body,q),ts_rank(fts,q) as rank
from apod, to_tsquery('stars') q
where fts @@ q order by rank desc limit 10;

Time: 723.634 ms
```

10 times !

```
select id,ts_headline(body,q),ts_rank from (
  select id,body,q, rank(fts,q) as rank from apod,
to_tsquery('stars') q
  where fts @@ q order by rank desc limit 10
) as foo;

Time: 21.846 ms
```

```
=#select count(*)from apod where fts @@ to_tsquery('stars');
 count
-------
   790
```

FTS tips – Query rewriting

- Rewriting query online
  - Query expanding
    - synonyms (new york => Gottham, Big Apple, ...)
  - Query narrowing
    - Курск => submarine Курск
- Like thesaurus dictionary, but doesn't require reindexing

FTS tips – Query rewriting

ts_rewrite (tsquery, tsquery, tsquery)

ts_rewrite (ARRAY[tsquery,tsquery,tsquery]) from aliases

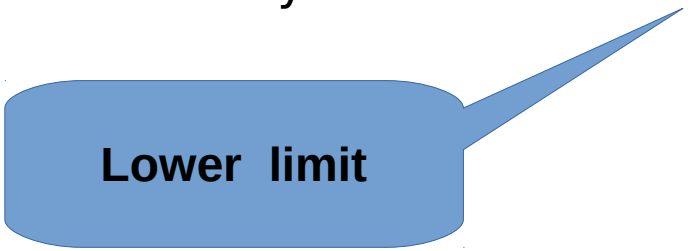
ts_rewrite (tsquery,'select tsquery,tsquery from aliases')

```
create table aliases( t tsquery primary key, s tsquery);

insert into aliases values(to_tsquery('supernovae'),
to_tsquery('supernovae|sn'));

apod=# select ts_rewrite(to_tsquery('supernovae'),
'select * from aliases');

      ts_rewrite
--------------------
 'supernova' | 'sn'
```

FTS tips – Query rewriting

```
apod=# select title, coalesce(ts_rank_cd(fts,q,1),2) as rank
from apod, to_tsquery('supernovae') q
where fts @@ q  order by rank desc limit 10;
```

title	rank
The Mysterious Rings of Supernova 1987A	0.669633
Tycho's Supernova Remnant in X-ray	0.598556
Tycho's Supernova Remnant in X-ray	0.598556
Vela Supernova Remnant in Optical	0.591655
Vela Supernova Remnant in Optical	0.591655
Galactic Supernova Remnant IC 443	0.590201
Vela Supernova Remnant in X-ray	0.589028
Supernova Remnant: Cooking Elements In The LMC	0.585033
Cas A Supernova Remnant in X-Rays	0.583787
Supernova Remnant N132D in X-Rays	**0.579241**



FTS tips – Query rewriting

```
apod=# select id, title, coalesce(ts_rank_cd(fts,q,1),2) as rank
from apod, ts_rewrite(to_tsquery('supernovae'), 'select * from aliases') q
where fts @@ q  order by rank desc limit 10;
   id    |                    title                     |   rank
---------+----------------------------------------------+----------
 1162701 | The Mysterious Rings of Supernova 1987A      |  0.90054
 1162717 | New Shocks For Supernova 1987A               | 0.738432
 1163673 | Echos of Supernova 1987A                     | 0.658021
 1163593 | Shocked by Supernova 1987a                   | 0.621575
 1163395 | Moving Echoes Around SN 1987A                | 0.614411
 1161721 | Tycho's Supernova Remnant in X-ray           | 0.598556
 1163201 | Tycho's Supernova Remnant in X-ray           | 0.598556
 1163133 | A Supernova Star-Field                       |  595041
 1163611 | Vela Supernova Remnant in Optical            | 0.  1655
 1161686 | Vela Supernova Remnant in Optical            | 0.5   55
```

```
apod=# select  title, coalesce(rank_cd(fts,q,1),2) as rank
from apod, to_tsquery('supernovae') q
where fts @@ q and id=1162717;
```

```
            title             |   rank
------------------------------+----------
 New Shocks For Supernova 1987A | 0.533312
```

Old rank

new document

FTS tips — strip tsvector

- Positions in tsvector used only for ranking ! Strip them from tsvector (less size) if ranking is not needed. Save size of tsvector.

```
postgres=# select to_tsvector('w1 w3 w1 w3');
     to_tsvector
--------------------
 'w1':1,3 'w3':2,4
(1 row)

Time: 0.268 ms
postgres=# select strip(to_tsvector('w1 w3 w1 w3'));
   strip
-----------
 'w1' 'w3'
(1 row)
```

Fast approximated statistics

- Gevel extension — GiST/GIN indexes explorer (http://www.sai.msu.su/~megera/wiki/Gevel)
- **Fast** — uses only GIN index (no table access)
- **Approximated** — no table access, which contains visibility information, approx. for long posting lists
- For mostly **read-only** data error is small

Fast approximated statistics

- Top-5 most frequent words (463,873 docs)

```
=# SELECT * FROM gin_stat('gin_idx') as t(word text, ndoc int)
 order by ndoc desc limit 5;

  word  |  ndoc
--------+--------
 page   | 340858
 figur  | 240366
 use    | 148022
 model  | 134442
 result | 129010
(5 rows)
Time: 520.714 ms
```

Fast approximated statistics

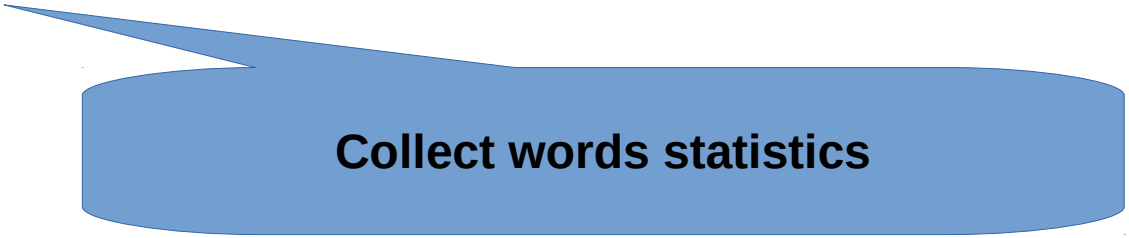
- gin_stat() vs ts_stat()

```
=# select * into stat from ts_stat('select fts from papers') order by ndoc
desc, nentry desc,word;

...wait.... 68704,182 ms

=# SELECT a.word, b.ndoc as exact, a.estimation as estimation,
round ( (a.estimation-b.ndoc)*100.0/a.estimation,2)||'%' as error
FROM (SELECT * FROM gin_stat('gin_x_idx') as t(word text, estimation int)
order by estimation desc limit 5 ) as a, stat b
WHERE a.word = b.word;

  word  | exact  | estimation | error
--------+--------+------------+-------
 page   | 340430 |     340858 | 0.13%
 figur  | 240104 |     240366 | 0.11%
 use    | 147132 |     148022 | 0.60%
 model  | 133444 |     134442 | 0.74%
 result | 128977 |     129010 | 0.03%
(5 rows)
Time: 550.562 ms
```

FTS tips — misprints (pg_trgm)

```
=# select show_trgm('supyrnova');
                        show_trgm
-----------------------------------------------
 {"  s"," su",nov,ova,pyr,rno,sup,upy,"va ",yrn}
```

```
=# select * into apod_words from ts_stat('select fts from apod') order by ndoc desc,
     nentry desc,word;

=# \d apod_words
  Table "public.apod_words"
 Column |  Type   | Modifiers
--------+---------+-----------
 word   | text    |
 ndoc   | integer |
 nentry | integer |
```




```
=# create index trgm_idx on apod_words using gist(word gist_trgm_ops);

=# select word, similarity(word, 'supyrnova') AS sml
from apod_words where word % 'supyrnova' order by sml desc, word;

   word    |   sml
-----------+----------
 supernova | 0.538462
```

FTS tips:to be or not to be

- Problem
  - Different searches using one tsvector
    - With stop words
    - Without stop words
- Solution
  - Use two FTS configurations
    - Index with 'no_stop' FTS configuration (include all words)
    - Use different FTS configurations for search

FTS tips:to be or not to be

```
SELECT to_tsvector('no_stop', 'to be or not to be. This is my document.')
      @@ plainto_tsquery('english', 'to be or not to be');
NOTICE:  text-search query contains only stop words or doesn't contain lexemes, ignored
 ?column?
----------
 f
(1 row)


SELECT to_tsvector('no_stop', 'to be or not to be. This is my document.')
       @@ plainto_tsquery('no_stop', 'to be or not to be');
 ?column?
----------
 t
(1 row)
```

FTS tips: Inverse FTS (FQS)

- Find queries, which match given document
  - Automatic text classification

```
SELECT * FROM queries;
                 q                  |  tag
------------------------------------+-------
 'supernova' & 'star'               | sn
 'black'                            | color
 'big' & 'bang' & 'black' & 'hole'  | bang
 'spiral' & 'galaxi'                | shape
 'black' & 'hole'                   | color
(5 rows)


SELECT * FROM queries WHERE
 to_tsvector('black holes never exists before we think about them')
@@ q;
        q          |  tag
-------------------+-------
 'black'           | color
 'black' & 'hole'  | color
(2 rows)
```

FTS tips: Refined search

- Search for wide query, rank using narrow query
  - Using narrow query can be risky — zero or little results
  - Better to refine search when ranking

```
SELECT title,ts_rank_cd(fts, to_tsquery('x-ray & russian')) AS rank
FROM apod WHERE fts  @@ to_tsquery('x-ray&russian') ORDER BY rank
DESC LIMIT 5;
                 title                    |   rank
------------------------------------------+-----------
 The High Energy Heart Of The Milky Way | 0.0240938
(1 row)
SELECT title,ts_rank_cd(fts, to_tsquery('x-ray & russian')) AS rank
FROM apod WHERE fts  @@ to_tsquery('x-ray') ORDER BY rank DESC LIMIT 5;
                 title                    |   rank
------------------------------------------+-----------
 The High Energy Heart Of The Milky Way | 0.0240938
 X-Ray Jet From Centaurus A               |         0
 Barnacle Bill And Sojourner              |         0
 The Crab Nebula in X-Rays                |         0
 M27: The Dumbbell Nebula                 |         0
(5 rows)
```

Postgres PROFESSIONAL

Full-text search in PostgreSQL in milliseconds

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=3 loops=
   ->  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 width=0)
(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 loops=1)
   ->  Index Scan using ti2_index on ti2  (cost=20.00..26256.30 rows=47692 width=282)
(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 !

GIN Improvements

- Compress +
- Store additional information (compression)
  (positions for FTS, array length for arrays...)
  Calculate ranking in index !
- Output **ordered** results from index (no heap scan!)
- Optimize execution of (rare & frequent) query +
  Before:
  T(freq & rare)=T(rare & freq) ~ T(freq)
  Now:
  T(freq & rare)>>T(rare & freq) ~ T(rare)

Postgres PROFESSIONAL

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.**

Postgres PROFESSIONAL

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.**

Phrase Search

- Queries 'A & B'::tsquery и 'B & A'::tsquery produce the same result
- Sometimes it's needed to search phrase (preserve order of words in a query)
- New operator $ (A $ B): word 'A' followed by 'B'
  - A & B (the same priority)
  - exists at least one pair of positions P_B, P_A , so that $0 \le P_B - P_A \le 1$ (distance condition)

  A $[n] B: $0 \le P_B - P_A \le n$

  A $ B $\neq$ B $ A

Phrase search - definition

A $ B: word 'A' followed by 'B':

- A & B (the same priority)
- exists at least one pair of positions P_B, P_A , so that $0 \le P_B - P_A \le 1$ (distance condition)

A $[n] B: $0 \le P_B - P_A \le n$

Result of operation:

- false
- true and array of positions of left argument which satisfy distance condition (without positional information $ is equivalent to &)

$ is very similar to & except: A $ B ≠ B $ A

Phrase search - properties

'A $[n] B $[m] C' → '(A $[n] B) $[m] C' →
matched phrase length ≤ max(n, m)
Note: 'A C B' matched by '(A $[2] B) $ C'

'A $[n] (B $[m] C') →
matched phrase length ≤ n + m
Note: Order is preserved for any n, m

'A $[0] B' matches the word with two different forms (infinitives)

```
=# SELECT ts_lexize('ispell','bookings');
   ts_lexize
---------------
 {booking,book}
to_tsvector('bookings') @@ 'booking $[0] book'::tsquery
```

Phrase search - practice

Phrase:

- `'A B C' → 'A $ (B $ C)'`
- `'A B C' → '(A $ B) $[2] C'`
- `TSQUERY phraseto_tsearch([CFG,] TEXT)`

Stop-words: `'A the B' → 'A $[2] B'`

What shall we do with complex queries?

`A $ ( B & ( C | ! D ) →` ???

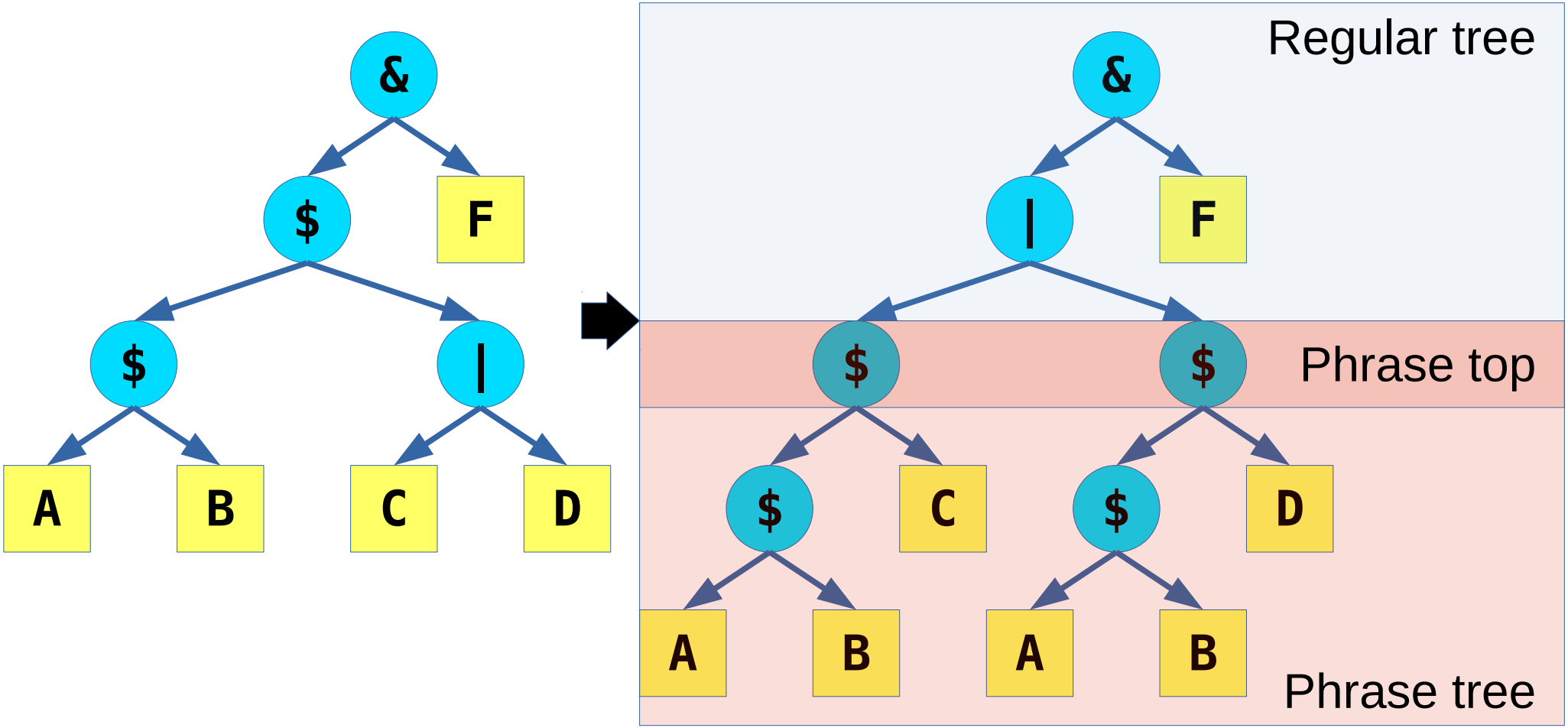
Phrase search - internals

Phrase search has overhead, since it requires access and operations on posting lists

To avoid slowdown of existing tsearch, executor of tsquery should not access positions without necessity. To facilitate this, any $ operations pushed down in query tree, so tsearch executor can call special phrase executor for the top $ operation, which will work only with query tree containing only $ operations.

Phrase search - transformation

```
( (A $ B) $ (C | D) ) & F
```



Phrase search - push down

```
a $ (b&c) => (a$b) & (a$c)
(a&b) $ c => (a$c) & (b$c)
a $ (b|c) => (a$b) | (a$c)
(a|b) $ c => (a$c) | (b$c)
a $ !b    => a & !(a$b)
```

there is no position of A followed by B

```
!a $ b    => b & !(a$b)
```

there is no position of B precedenced by A

Phrase search - transformation

```
# select '( A | B ) $ ( D | C )'::tsquery;
                        tsquery
  ----------------------------------------------
   'A' $ 'D' | 'B' $ 'D' | 'A' $ 'C' | 'B' $ 'C'

# select 'A $ ( B & ( C | ! D ) )'::tsquery;
                             tsquery
  ------------------------------------------------------
   ( 'A' $ 'B' ) & ( 'A' $ 'C' | 'A' & !( 'A' $ 'D' ) )
```

Phrase search - example

'PostgreSQL can be extended by the user in many ways' ->

```
# select phraseto_tsquery('PostgreSQL can be extended
                          by the user in many ways');
                             phraseto_tsquery
  ------------------------------------------------------------------
'postgresql' $[3] ( 'extend' $[3] ( 'user' $[2] ( 'mani' $ 'way' ) ) )
```

Can be written by hand:

```
'postgresql' $[3] extend $[6] user $[8] mani $[9] way
```

Difficult to modify, use phraseto_tsquery() function !

Phrase search - TODO

Ranking functions

Headline generation

Rewrite subsystem

Concatenation of two tsquery by $ operation: $$?
- like other concatenations: &&, || and !!
- distance $$[2] !, functional interface ?

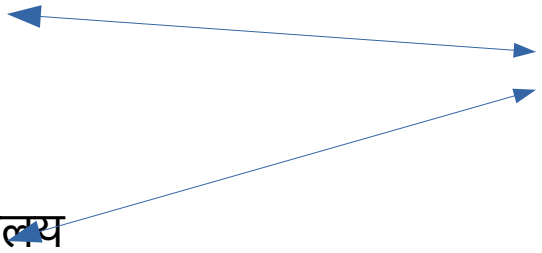
Need testing for agglutinative languages (norwegian, german, etc)

Devanagari script support

PostgreSQL 8.3- has problem with Devanagari script (http://en.wikipedia.org/wiki/Devanagari - script for Hindi, Marathi, Nepali, Sanscrit,...).

```
select * from ts_parse('default','मदन पुरस्कार पुस्तकालय');
```

```
2     मदन                       Madan Puraskar Pustakalaya
12
2     पुरस
12     ◌
2     कार
12
2     पुस
12     ◌
2     तकालय
```



Virama sign (modifier, suppresses inherent vowel) – *punct* in np_NP locale. Breaks all parsers, which use locale.

Devanagari script support

मदन पुरस्कार पुस्तकालय (Madan Puraskar Pustakalaya)

character	byte	UTF-32	encoded as	glyph	name
0	0	00092E	E0 A4 AE	म	DEVANAGARI LETTER MA
1	3	000926	E0 A4 A6	द	DEVANAGARI LETTER DA
2	6	000928	E0 A4 A8	न	DEVANAGARI LETTER NA
3	9	000020	20		SPACE
4	10	00092A	E0 A4 AA	प	DEVANAGARI LETTER PA
5	13	000941	E0 A5 81	◌ु	DEVANAGARI VOWEL SIGN U
6	16	000930	E0 A4 B0	र	DEVANAGARI LETTER RA
7	19	000938	E0 A4 B8	स	DEVANAGARI LETTER SA
8	22	00094D	E0 A5 8D	◌्	DEVANAGARI SIGN VIRAMA
9	25	000915	E0 A4 95	क	DEVANAGARI LETTER KA
10	28	00093E	E0 A4 BE	◌ा	DEVANAGARI VOWEL SIGN AA
11	31	000930	E0 A4 B0	र	DEVANAGARI LETTER RA
12	34	000020	20		SPACE
13	35	00092A	E0 A4 AA	प	DEVANAGARI LETTER PA
14	38	000941	E0 A5 81	◌ु	DEVANAGARI VOWEL SIGN U
15	41	000938	E0 A4 B8	स	DEVANAGARI LETTER SA
16	44	00094D	E0 A5 8D	◌्	DEVANAGARI SIGN VIRAMA
17	47	000924	E0 A4 A4	त	DEVANAGARI LETTER TA
18	50	000915	E0 A4 95	क	DEVANAGARI LETTER KA
19	53	00093E	E0 A4 BE	◌ा	DEVANAGARI VOWEL SIGN AA
20	56	000932	E0 A4 B2	ल	DEVANAGARI LETTER LA
21	59	00092F	E0 A4 AF	य	DEVANAGARI LETTER YA

Devanagari script support

8.4+ knows Virama signs

```
=# select * from ts_parse('default','मदन पुरस्कार पुस्तकालय');
 tokid |  token
-------+---------
     2 | मदन
    12 |
     2 | पुरस्कार
    12 |
     2 | पुस्तकालय
(5 rows)
```

Thanks to Dibyendra Hyoju and Bal Krishna Bal for testing and valuable discussion

- FTS in PostgreSQL is a flexible search engine, but it is more than a complete solution

- It is a «collection of bricks» you can build your search engine with
  - Custom parser
  - Custom dictionaries
  - + All power of SQL (FTS+Spatial+Temporal)
  - Use tsvector as a custom storage

  - For example, instead of textual documents consider chemical formulas or genome string

THANKS !

www.postgrespro.ru