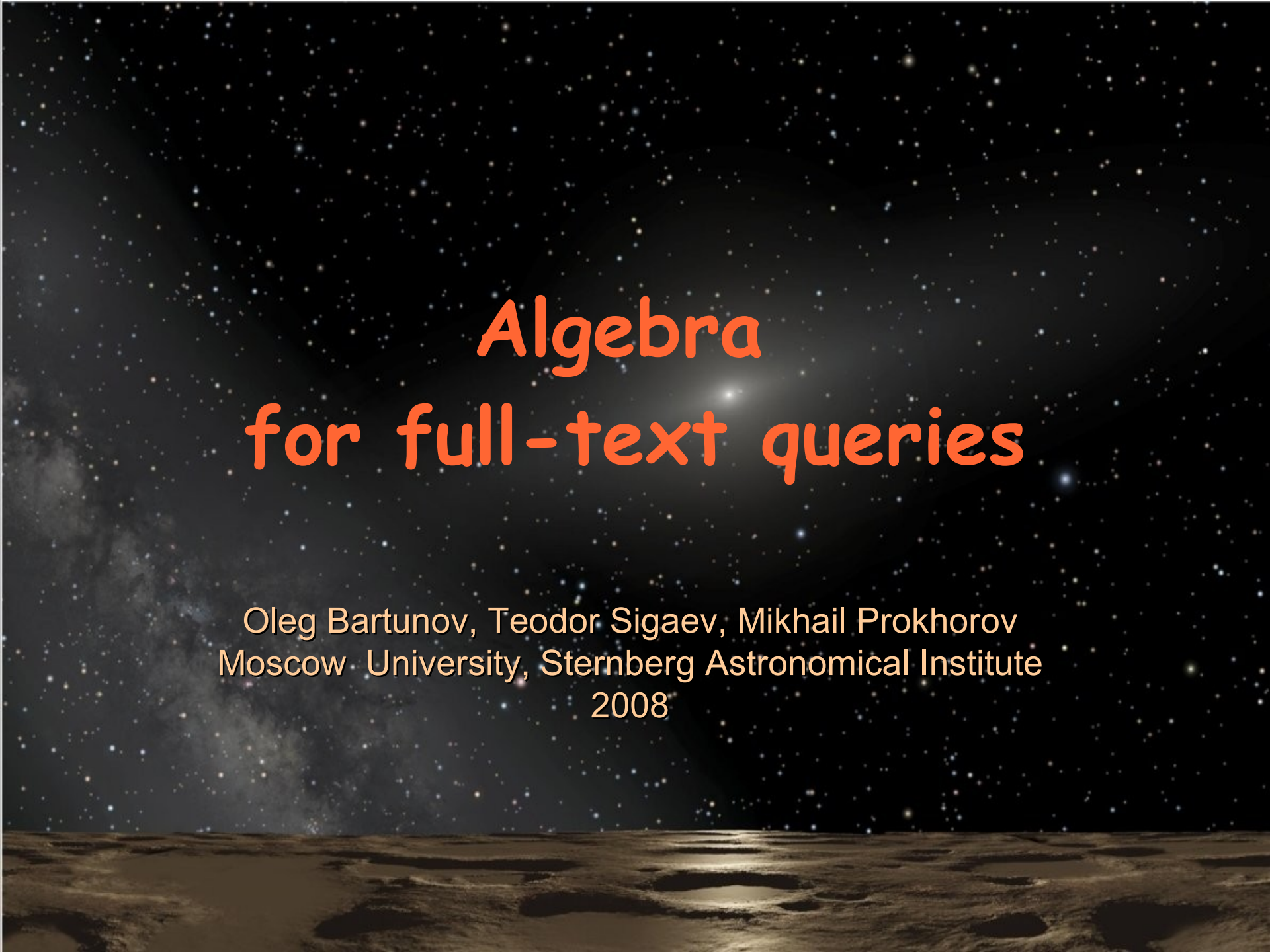

Algebra
for full-text queries
Oleg Bartunov, Teodor Sigaev, Mikhail Prokhorov
Moscow University, Sternberg Astronomical Institute
2008

FTS features

- Full integration with PostgreSQL
- 16 built-in configurations for 15 languages
- Support of user-defined FTS configurations
- Pluggable dictionaries (ispell, snowball, thesaurus) and parsers
- Full multibyte support (UTF-8)
- Relevance ranking
- Two types of indexes – GiST and GIN, both supports concurrency and recovery
- Rich query language with query rewriting support

FTS in PostgreSQL

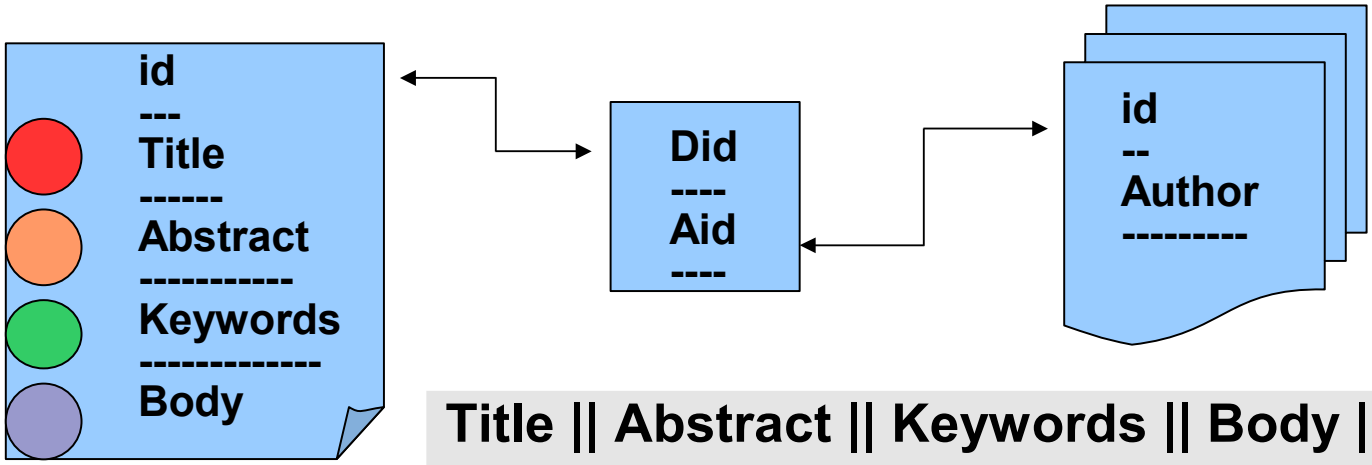
```
=# select  'a fat cat sat on a mat and ate a fat rat'::tsvector
                                @@
                  'cat & rat':: tsquery;
```

- **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

What is a Document ?

- Any textual attributes or their combination
- Should have unique id
- Textual result of any SQL command
  - Can be virtual entity



Грамматика (BNF)

- http://en.wikipedia.org/wiki/Backus-Naur_Form
- <lexeme> ::=
  <alphanumeric>{ <alphanumeric> }
  | "'"<character>{ <character> }"'"
- <character> ::=
  <alphanumeric>
  | "\" "'" /* single quote */
  | "'" "'" /* single quote */
  | <non-alphanumeric>
- <number> ::= <digit>{ <digit> }
- <weight> ::= A|B|C|D

Грамматика - tsvector

- <tsvector> ::=
  <entry>{ <space> <entry> }
  | /* empty */
- <entry> ::=
  <lexeme>[":" <positions>]
- <positions> ::=
  <position>{ "," <position> }
- <position>:
  <number>[<weight>]

Грамматика - tsquery

- <tsquery> ::=
  <expression>
  | /* empty */
- <expression> ::=
  <expression> "&" <expression>
  | <expression> "|" <expression>
  | "!" <expression>
  | "(" <expression> ")"
  | <lexeme>[:<modifiers>]
- <modifiers> ::=
  "*"
  | <weight>{ <weight> }["*"]

Examples

- tsvector

  'star':1B,3**A**,15,26B,32B,44 'supernovae':2**A**,25B,31B
  'star':1B,3A,15,26B,32B,44 'supernovae':25B,31B

- tsquery

  'supernovae':A & 'star':A* (match 1st tsvector)

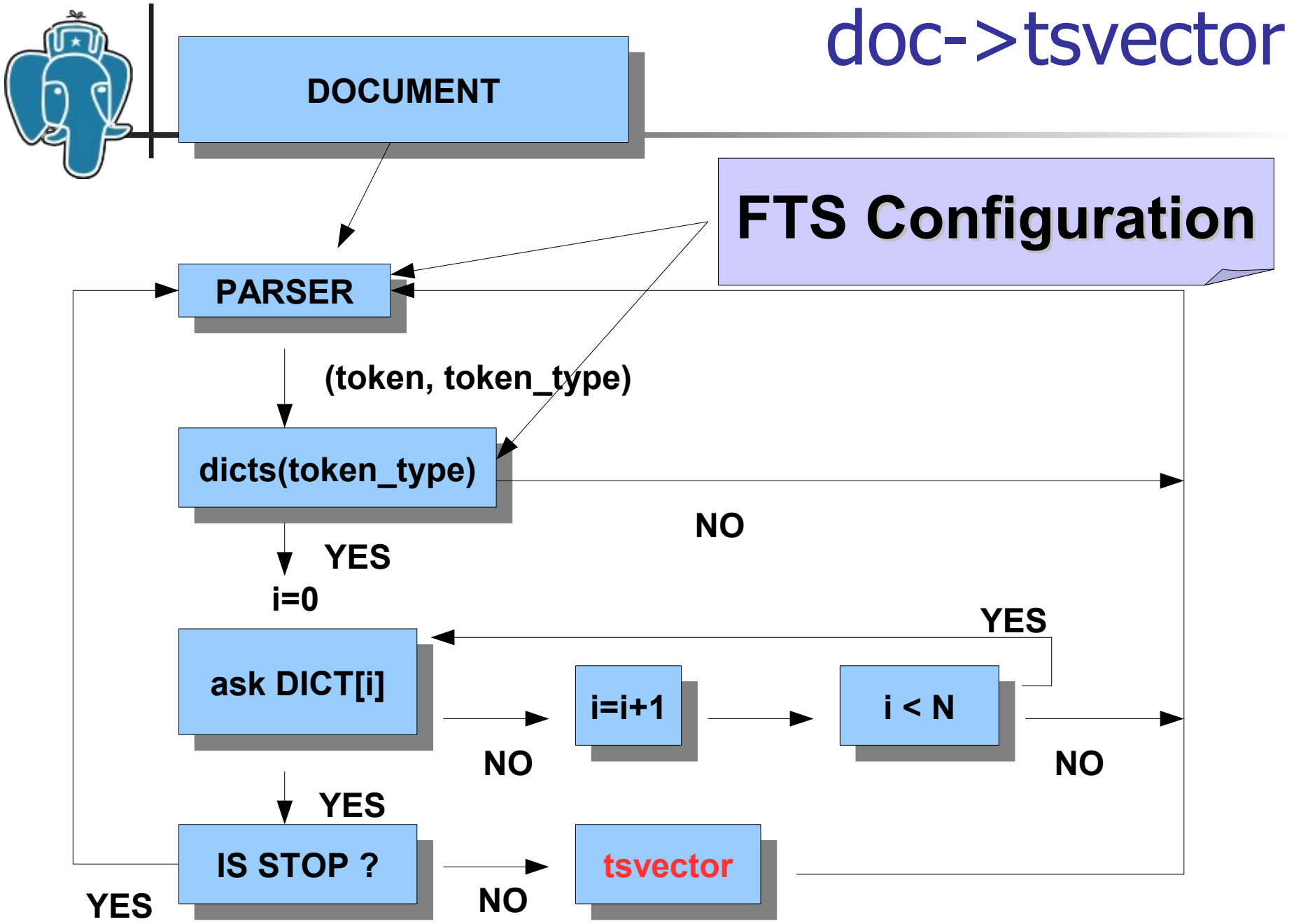
Functions

- text -> tsvector
  - to_tsvector([conf], text)
- text -> tsquery
  - to_tsquery([conf],text)
  - plainto_tsquery([conf],text)
- Configuration (conf) defines rules for document->text (parser, dict<->word map)
- word -> lexeme
  - ts_lexize(dict,text)

text -> tsquery

- 'Йцукен & ФЫВА'::tsquery
  никакого преобразования as is
- Входной язык запросов
  to_tsquery('большую & Лопату') =>
  'большая & лопата'
  применились словари
- Естественный язык
  plainto_tsquery('большую Лопату') =>
  'большая & лопата'
  применились словари и операция &

doc->tsvector



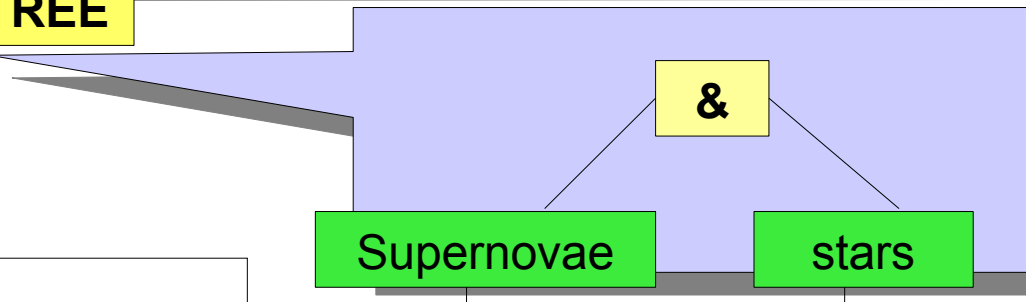
text->tsquery

QUERY

QPARSER

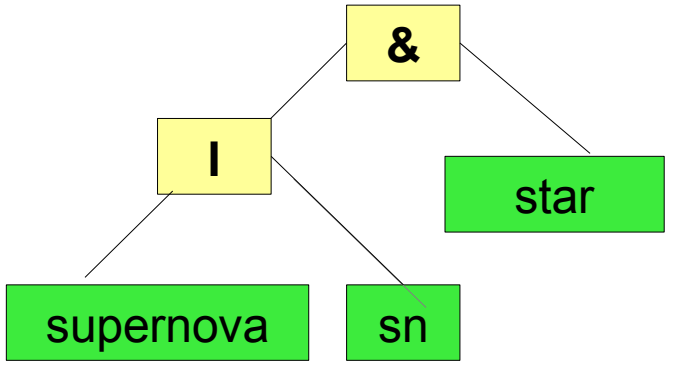
Supernovae & stars

QUERYTREE



{supernova,sn}

star



(supernova | sn) & star

Foreach leaf node

PARSER

(token, token_type)

dicts (token_type)

NO

YES

i=0

YES

? DICT[i]

i=i+1

i < N

NO

NO

YES

IS STOP ?

QUERYTREE

YES

NO

TSQUERY

Dictionaries

- **Dictionary** – is a program, which accepts token and returns
  - an array of lexemes, if it is known and not a stop-word
  - void array, if it is a stop-word
  - NULL, if it's unknown
- API for developing specialized dictionaries
- Built-in dictionary-templates :
  - ispell (works with ispell, myspell, hunspell dicts)
  - snowball stemmer
  - synonym, thesaurus
  - simple

Dictionaries:examples

- Intdict
  - '123456789' -> '123456'
- Roman
  - 'XIX' -> '19'
- Colours
  - 'FFFFFF' -> 'white'
- Regexp
  - H(\s|-)?(alpha|beta|gamma) h$2 — spectral lines

Интерфейс словарей

- `void* init(List *options)`
  Инициализация
- `TSLexeme* lexize(void *dict, char *lexeme, int32 lexeme_len, DictSubState *state)`

  Морфологизация. Параметр state – опциональный, для сложных словарей (тезаурус). Возврат NULL – слово не известно.

TSLexeme

```
typedef struct {
    uint16 nvariant;
    uint16 flags;
    char   lexeme; /* C-string */
} TSLexeme;
```

- lexize() возвращает массив структур, последняя структура содержит NULL-строчку (TSLexeme->lexeme). Пустой массив - стоп-слово.

Русский Ispell

- 'туши' => 'тушить | туша | туш | тушь'
-
```
nvariant | lexeme
---------+----------
       1 | тушить
       2 | туша
       3 | туш
       4 | тушь
```
-
- Простое перечисление – номер варианта (TSLexeme->nvariant), соединение вариантов по ИЛИ

Agglutinative languages (норвежский, германский...)

- http://en.wikipedia.org/wiki/Agglutinative_language
- Соединение слов без пробела
- Footbalklubber – футбольный судья
- Klubber on footbalfield – судья на футбольном поле
- По первому слову надо найти документ со второй фразой. Нужна разбивка слова на составные части и соответствующее построение запроса

Норвежский Ispell простой вариант

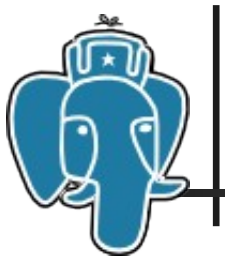
- 'foobar' => 'foo & bar'
-
```
nvariant | lexeme
---------+----------
       1 | foo
       1 | bar
```
- Номер варианта – постоянен, слова с одним номером объединяются по И. Все слова варианта должны содержаться в документе.

Норвежский Ispell

- 'footbalklubber' =>
  ' (football & klubber) | (foot & ball & clubber) '
-
```
nvariant | lexeme
---------+----------
       1 | football
       1 | klubber
       2 | foot
       2 | ball
       2 | klubber
```
- Подмножества с разными номерами соединяются по ИЛИ, слова с одним номером – по И

Флаги

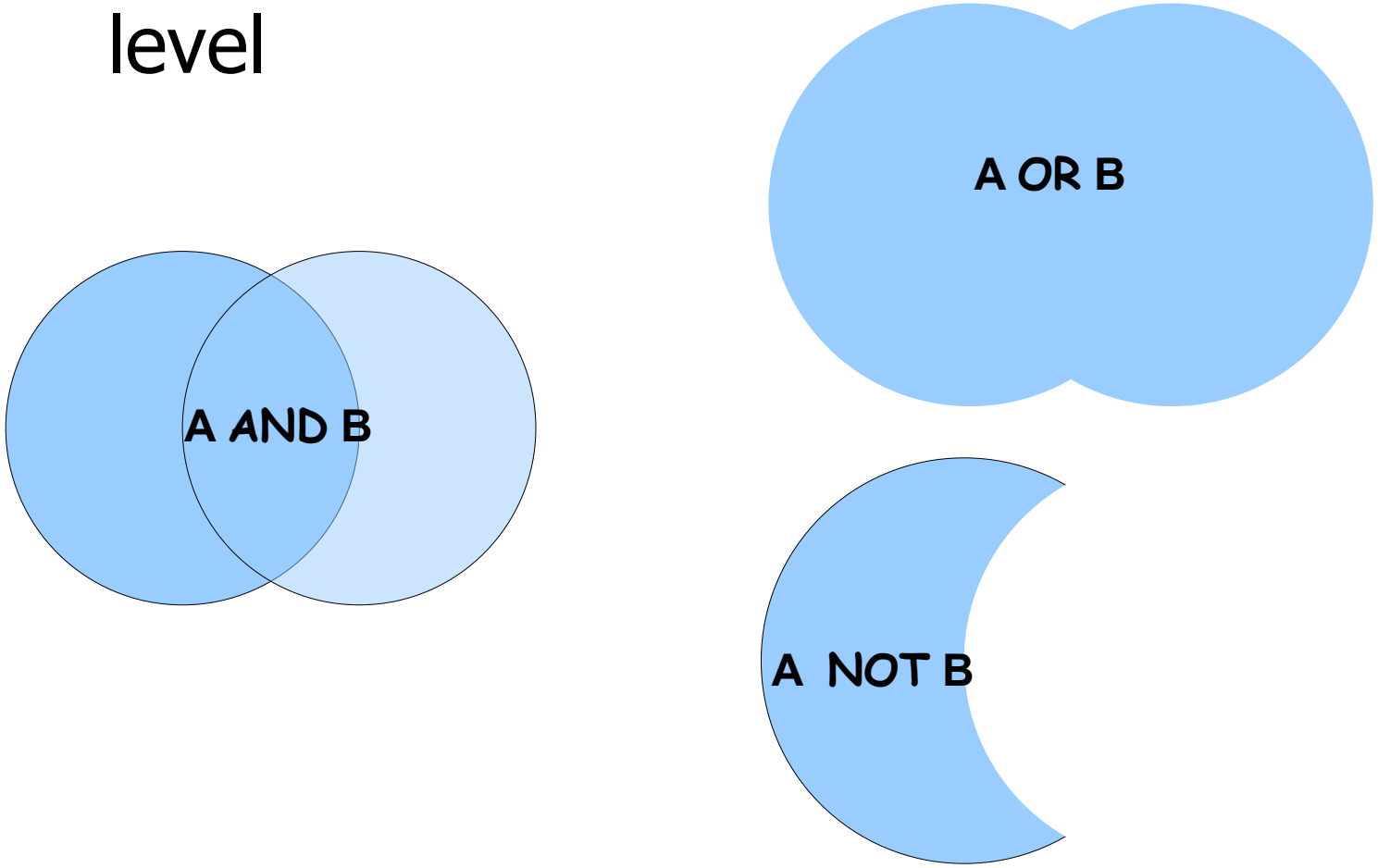
- TSLexeme->flags
- TSL_ADDPOS
  Инкремент позиции слова (тезаурус)
- TSL_PREFIX
  Префиксный поиск (8.4, пока такие словари неизвестны)



Motivation for Algebra

Motivation for Algebra

- Existing operators defined at *document* level



Motivation for Algebra

- Phrase Search requires operation at *lexeme* level — operator BEFORE ($)
- Different semantics - A AND NOT B
  - Phrase search:
    «A $ X» (X is anything, except B)
- Phrase can be very complex
  - Even simplest phrase can be transformed to complex expression.

to_tsquery('nb', '**telefonsvarer**') =>
'telefonsvarer' | 'telefon' & 'svar'

Motivation for Algebra

- Phrase can be constructed programmatically, or manually (SMTH::tsquery)

  «A $ (B $!(C $!D))»

- We need well-defined algebra for operations: & | ! $
- Backward compatibility !

Motivation for Algebra

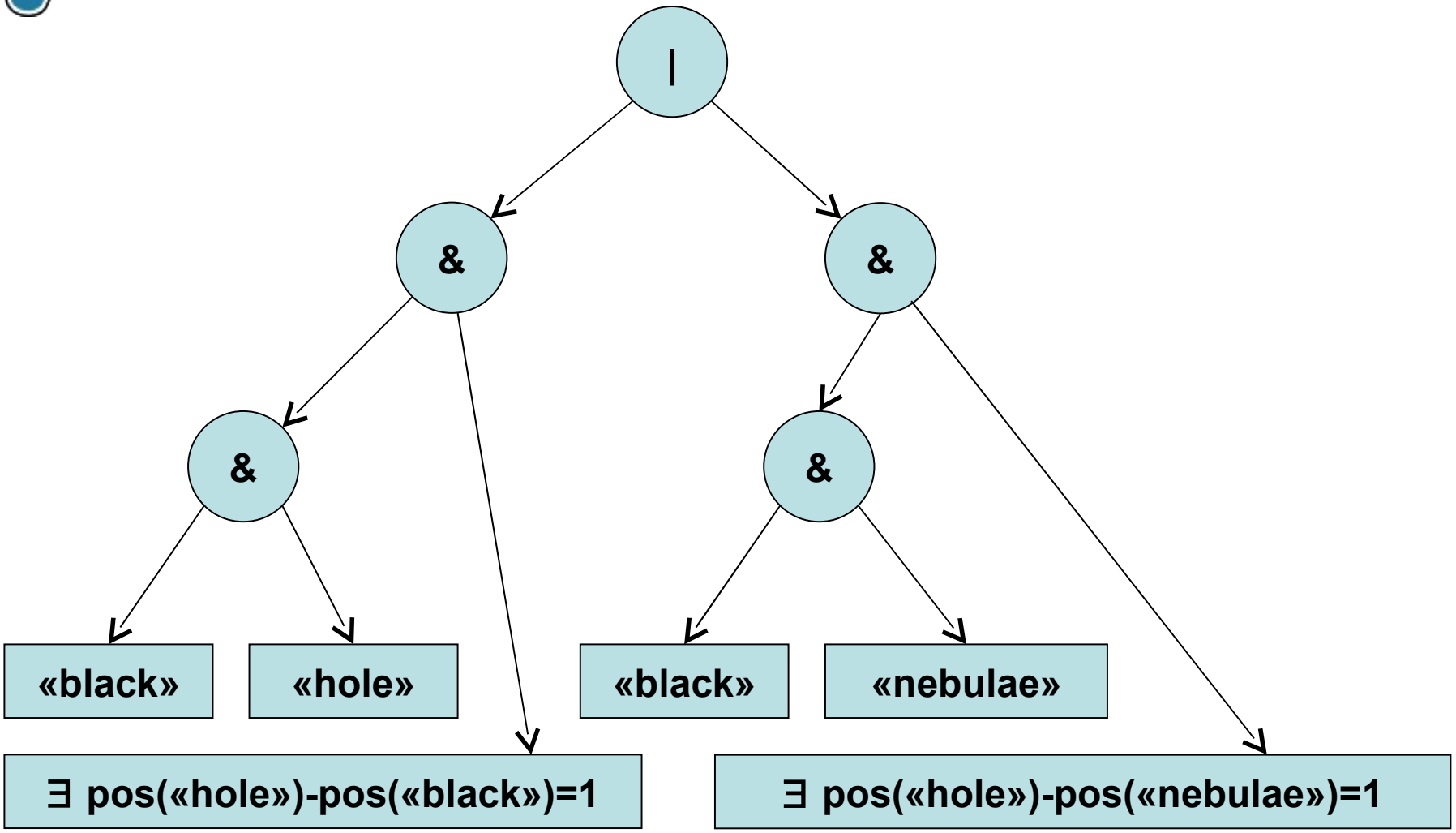
- We consider generalized phrase

  a $[n] b

- Operator BEFORE ($n) guarantee
  - *order* of operands — a BEFORE b
  - Distance between operands, default is 1

a $[n] b == a & b & (∃ i,j : $pos(b)_i - pos(a)_j = n$)

Query: «black» $ («hole» | «nebulae») ==>
«black» $ «hole» | «black» $ «nebulae»

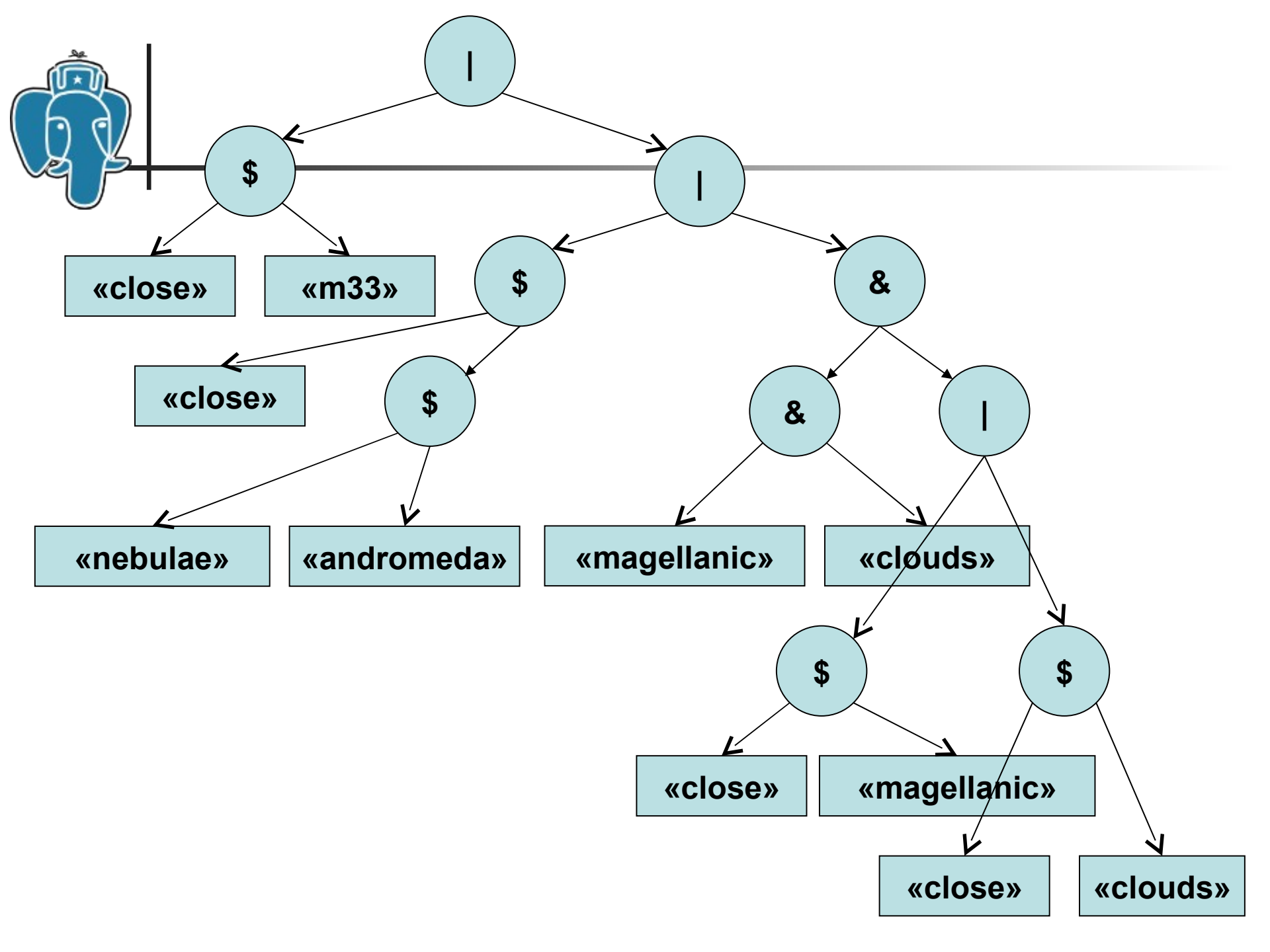


Example

Query: «close» $ «galaxies»

After dictionary: «close» $ («m33» |
(«andromeda» $ «nebulae» | («magellanic» & «clouds»))

Phrase: «close» $ «m33» |
(«close» $ («andromeda» $ «nebulae»)) |
(«magellanic» & «clouds» &
(«close» $ «magellanic» | «close» $ «clouds»)
)


|
$
|
«close»
«m33»
$
&
«close»
$
&
|
«nebulae»
«andromeda»
«magellanic»
«clouds»
$
$
«close»
«magellanic»
«close»
«clouds»

Phrase Search

- Possible extensions
  - #[n] — soft $[n], order doesn't important
  - a<$[n] b — at most n words between operands
  - a $[n]> b — at least n words between operands
  - And so on ...