How MeiliSearch works?

What's inside?

We have a songs database with *100 000* documents.

Those documents are stored under small internal ids.

Documents are composed of 3 fields:

- the song *title*
- the name of the *artist*
- the *release year*

Placeholder Search

That's easy, we pick the n **first internal ids from our database.**

We returns the documents associated with the internal ids to the user.

Faceted Search

Imagine that the user asked only for *John Lennon* songs released between *1957* and *1975*.

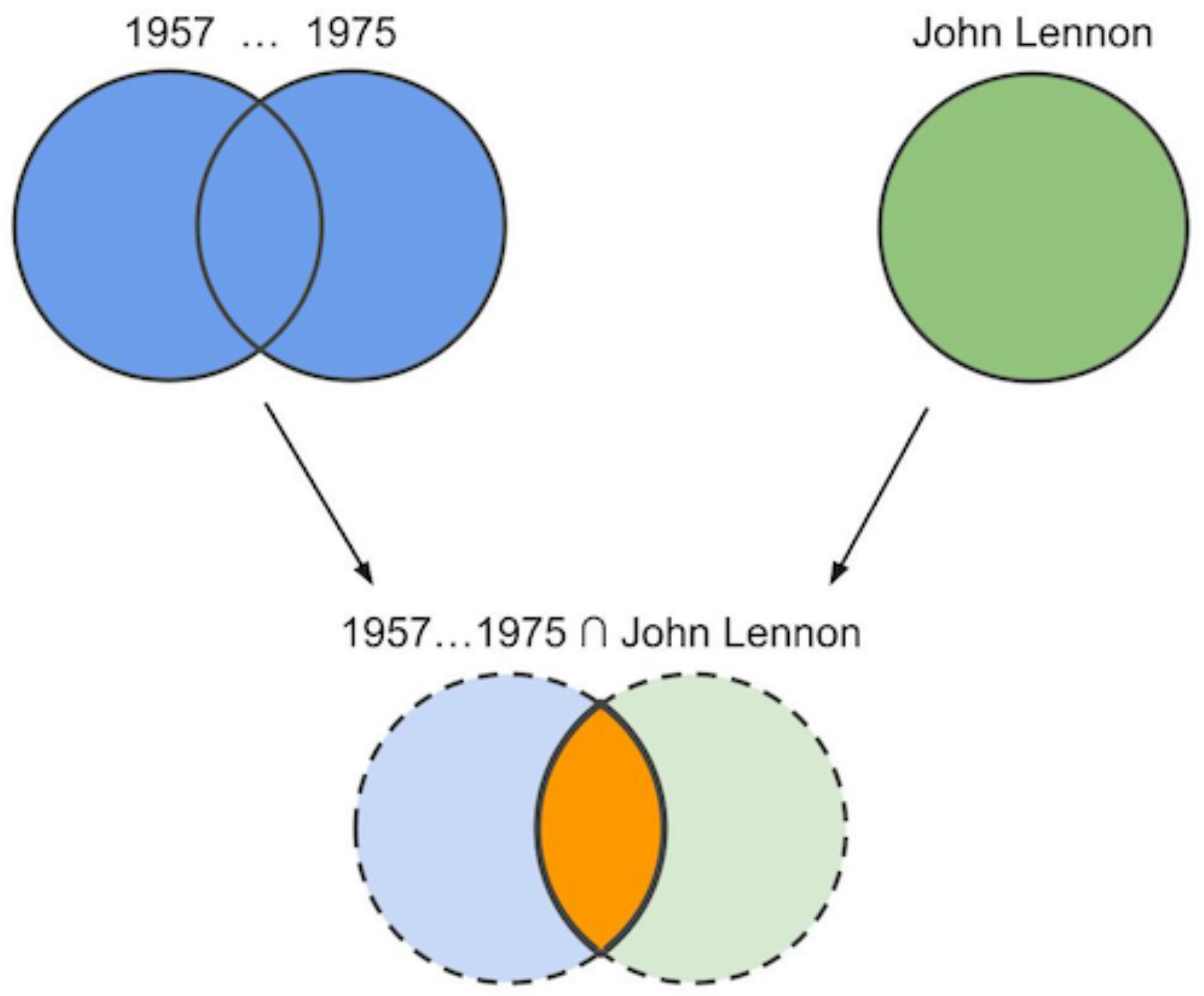
Because the *artist* and *release year* are faceted, that means that we have every facet values associated with the related internal ids.

Note that facet values are ordered.

Faceted Search

1. Get the internal ids stored under that *artist* equal to *John Lennon*.
2. Do an union of all the ids stored under the *release year* between *1957* and *1975*.
3. Do the intersection between ids found at step *1* and *2*.

We returns the documents associated with those internal ids to the user.



Search Query

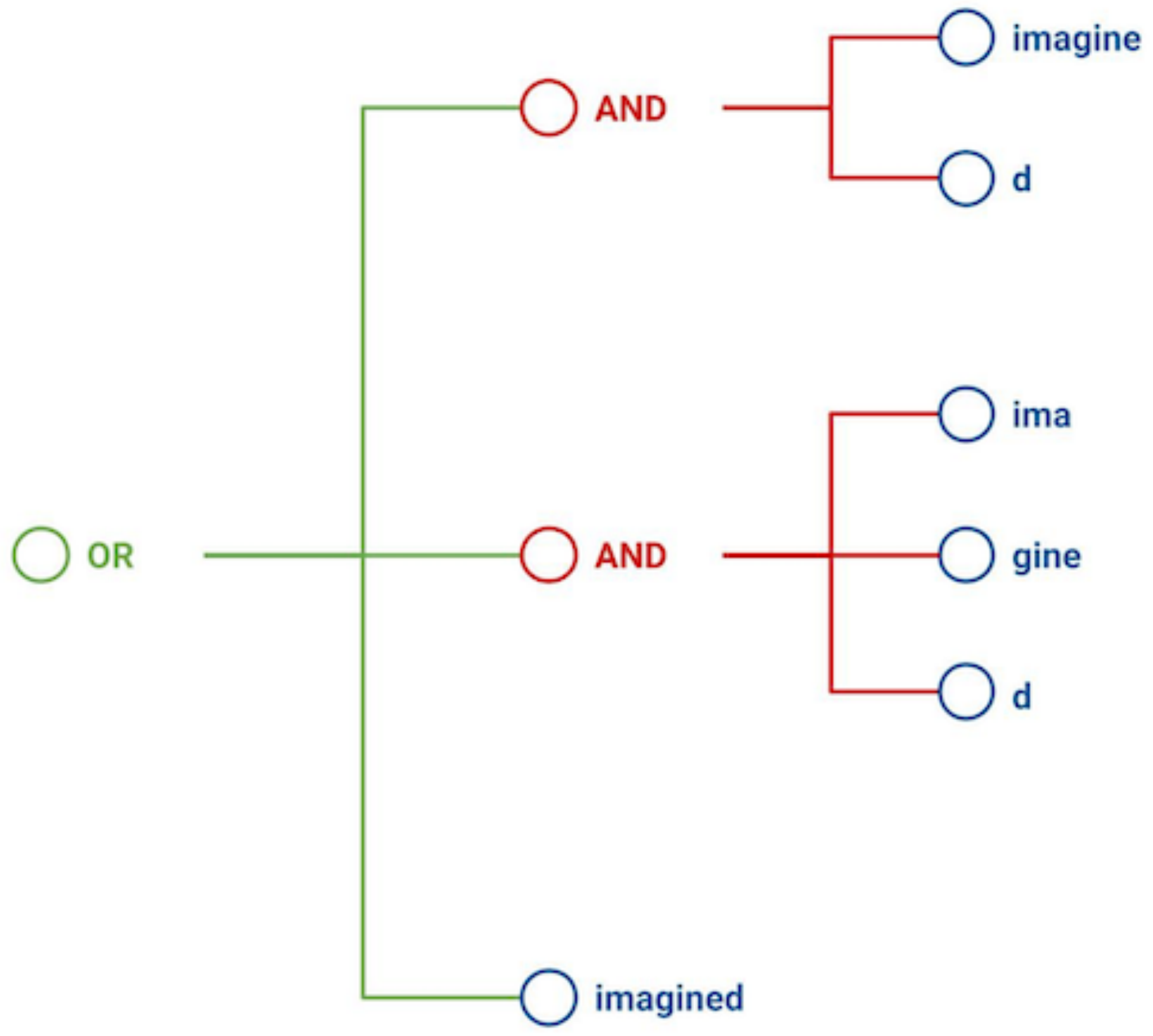
1. Extract the words from the query.
2. Construct a query tree from those words.
3. Construct the criteria list.
4. Run the criteria using the query tree.
5. Return the documents to the user. 🎉

Search Query

Construct the query tree

Imagine that a user try to search for *imagine d*. Where *d* is a prefix word.

1. We extract the words from the user query.
2. Compute and fetch the derivate words.
   — ngrams (*imagined*).
   — split (*ima gine*).
   — synonyms.

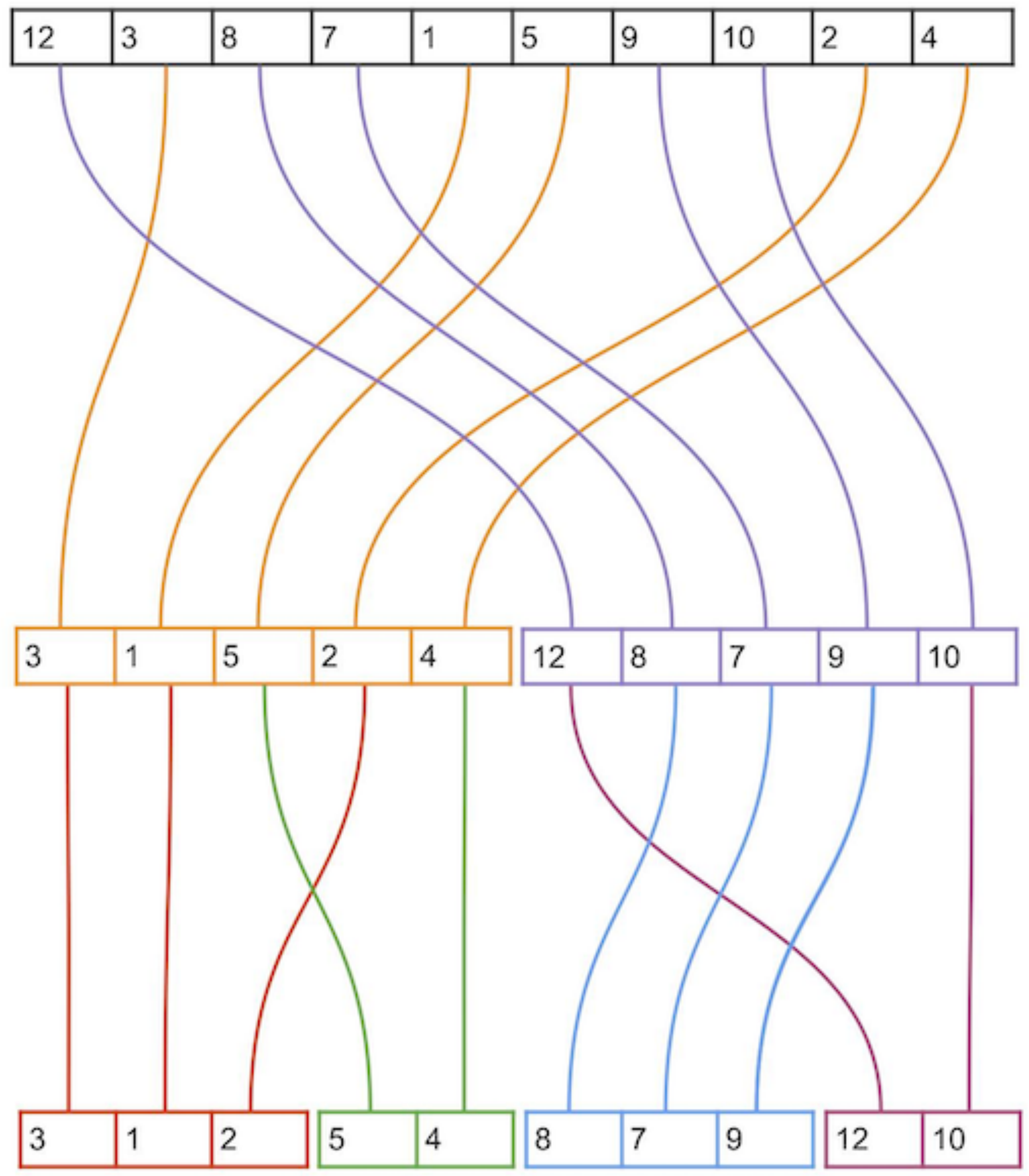


Search Query

Introduce what's a criterion

A criterion takes a bunch of internal ids as input, all those ids are considered equal.

The job of a criterion is to rank ids by returning subsets of those ids, in order.



Search Query

The default set of criteria

1. The *Words* criterion: the *more words* the better.
2. The *Typo* criterion: the *less number of typos* the better.
3. The *Proximity* criterion: the *nearest the query words* are the better.
4. The *Desc(release year)* criterion: the *most recent* the documents the better.

Search Query

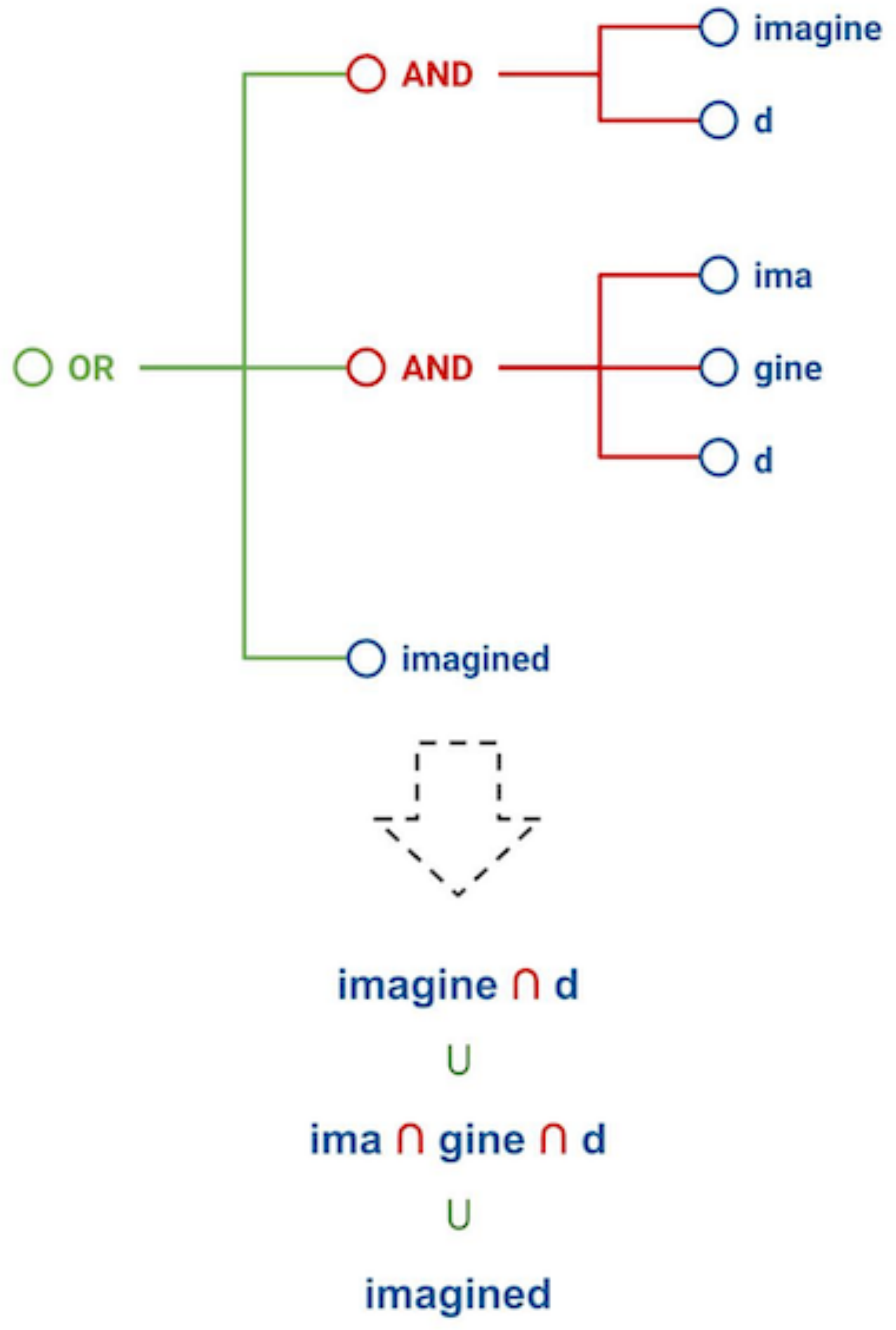
Run the criteria list

1. We ask documents the last criterion in the list.
2. This criterion must ask its parent the bucket of documents.
3. The answered documents are *sort* and returned in *buckets of equal score* using the query tree.

Search Query

Interpret the query tree

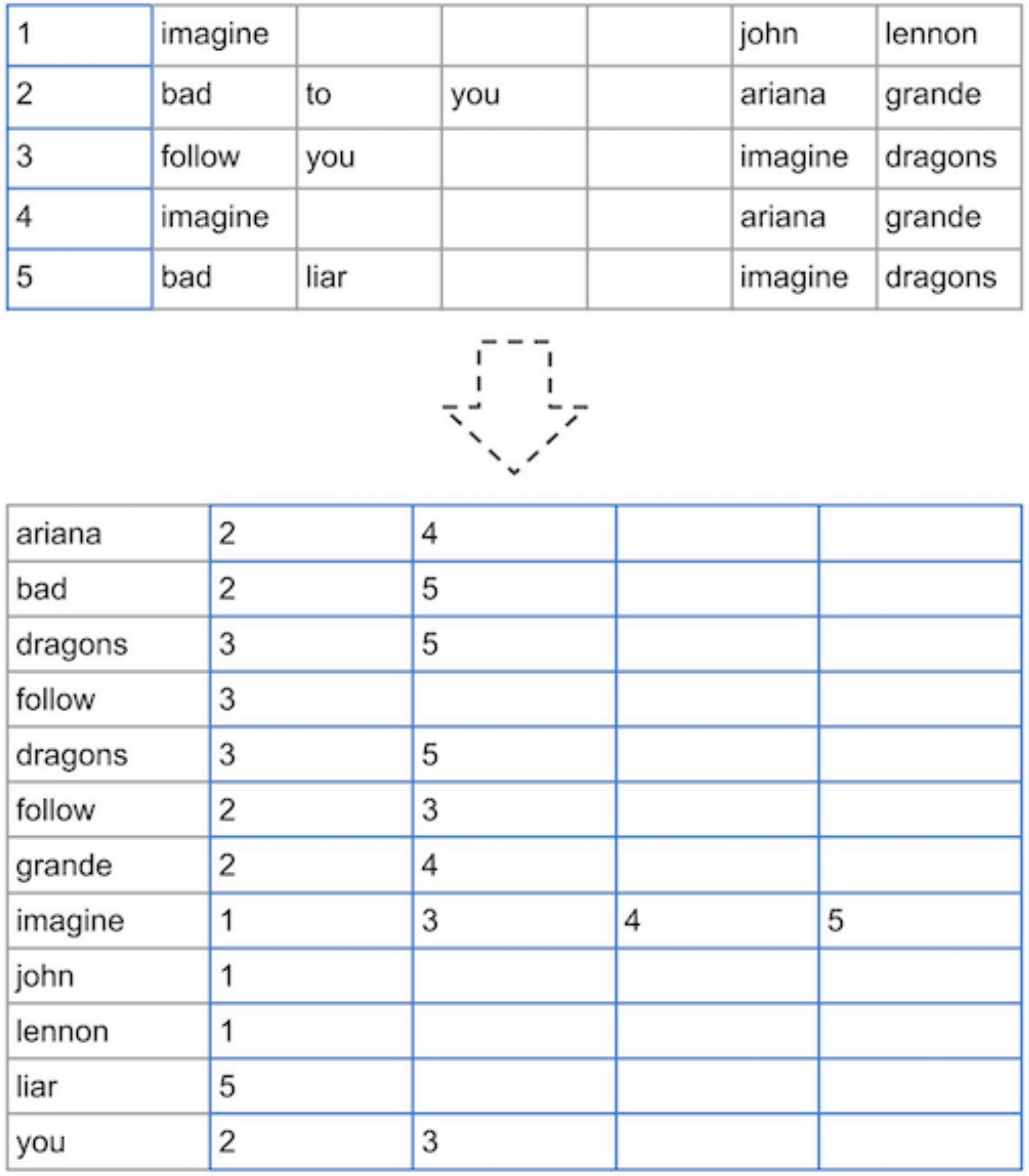
1. Fetch the internal ids associated with each query word.
2. Do the unions and intersection depending on the query tree shape.



Questions?

Data

Structures



1	imagine				john	lennon
2	bad	to	you		ariana	grande
3	follow	you			imagine	dragons
4	imagine				ariana	grande
5	bad	liar			imagine	dragons

ariana	2	4		
bad	2	5		
dragons	3	5		
follow	3			
dragons	3	5		
follow	2	3		
grande	2	4		
imagine	1	3	4	5
john	1			
lennon	1			
liar	5			
you	2	3		

The Inverted Index

What's that?

We list all the words from the documents in the database.

We associate each of these words to the list of documents ids that contains them.

The Inverted Index

Where do we use this?

This is the kind of data structure that we use for facets.

This is also used when we need to fetch the documents associated with the words in the query tree.

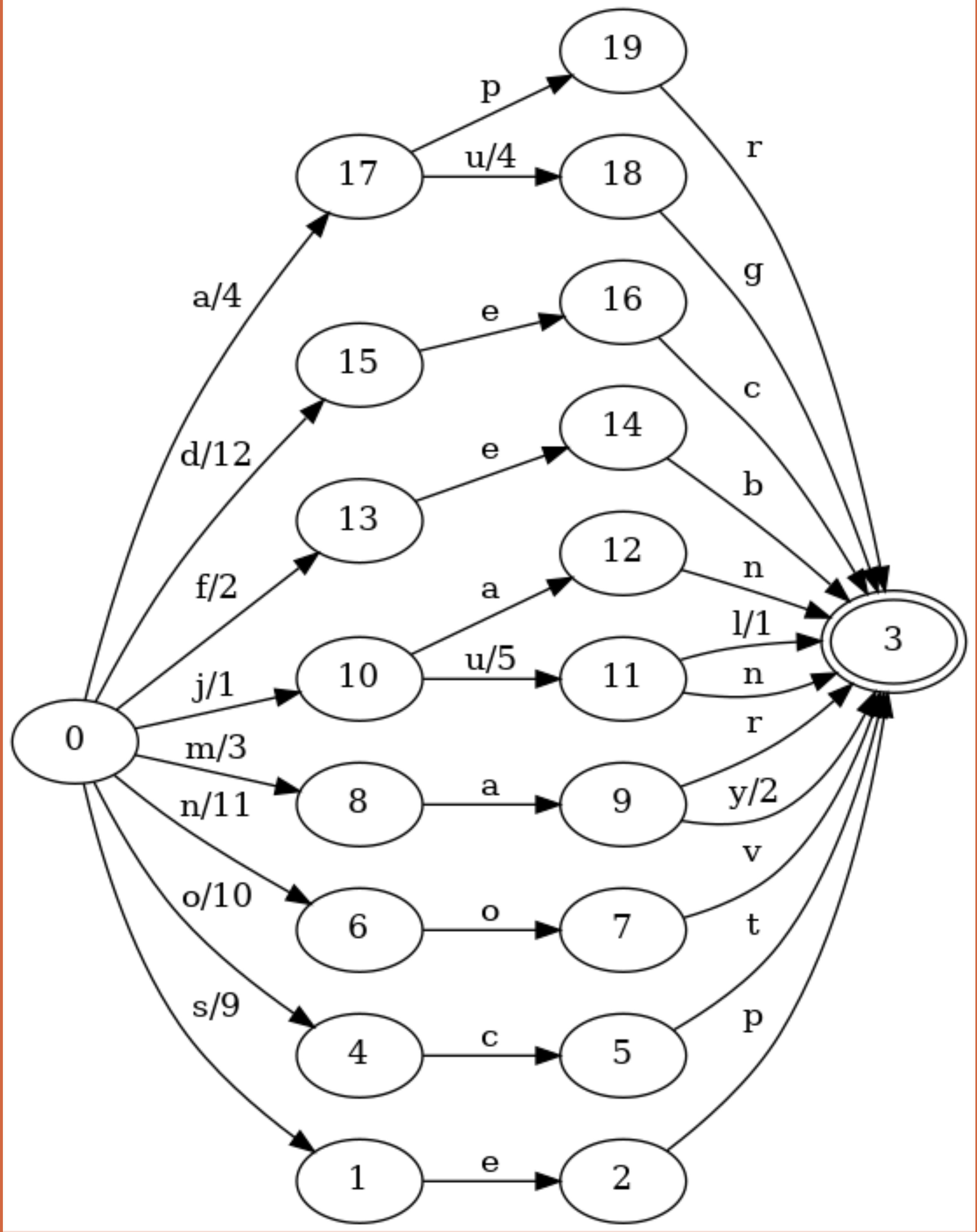
The Word Dictionnary

What's that?

This is the list of all the words in the whole database.

It has a lot of interresting properties:

1. **It is *compressed* and *lazily uncompressed*.**
2. **It can return a subset of words matching a *levenshtein* rule.**
3. **It can return all the words that**



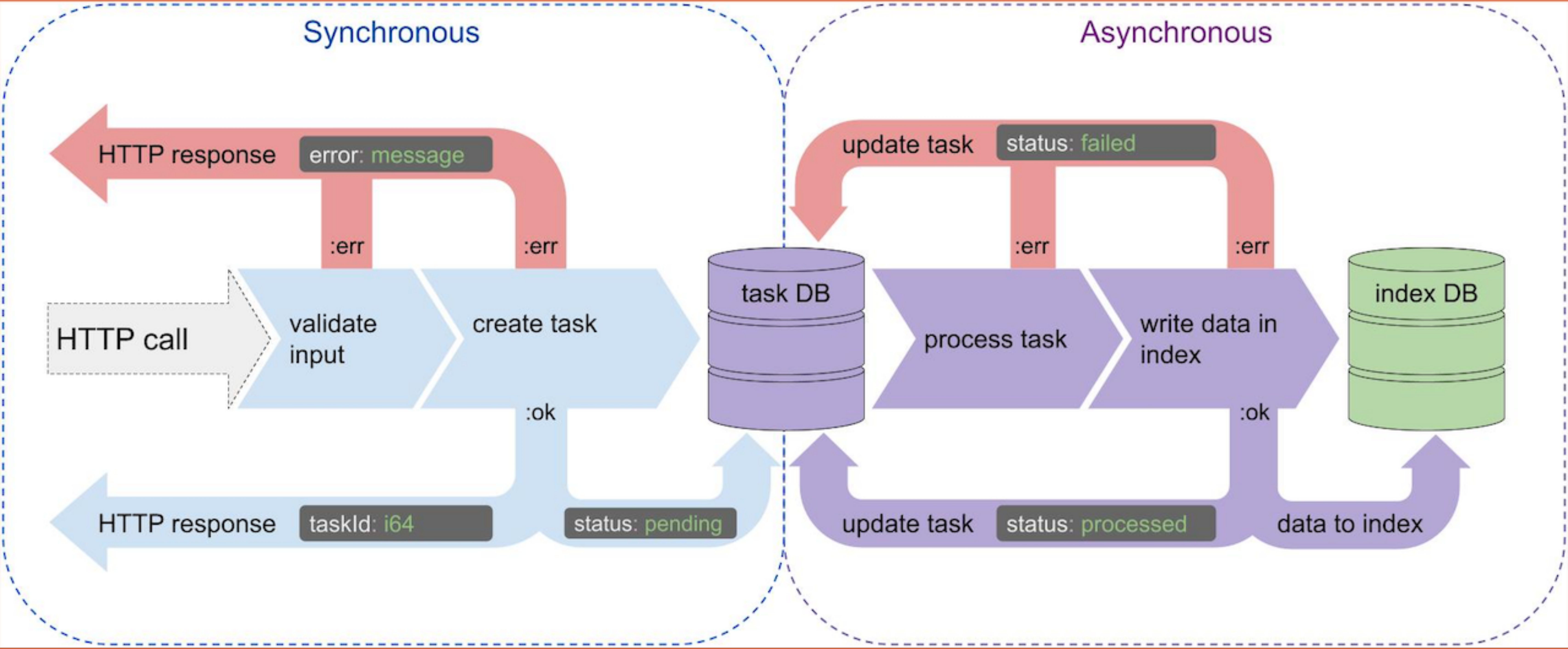
The Word Dictionnary

Where do we use this?

We use it anytime we need to fetch the derivate words from the query tree.

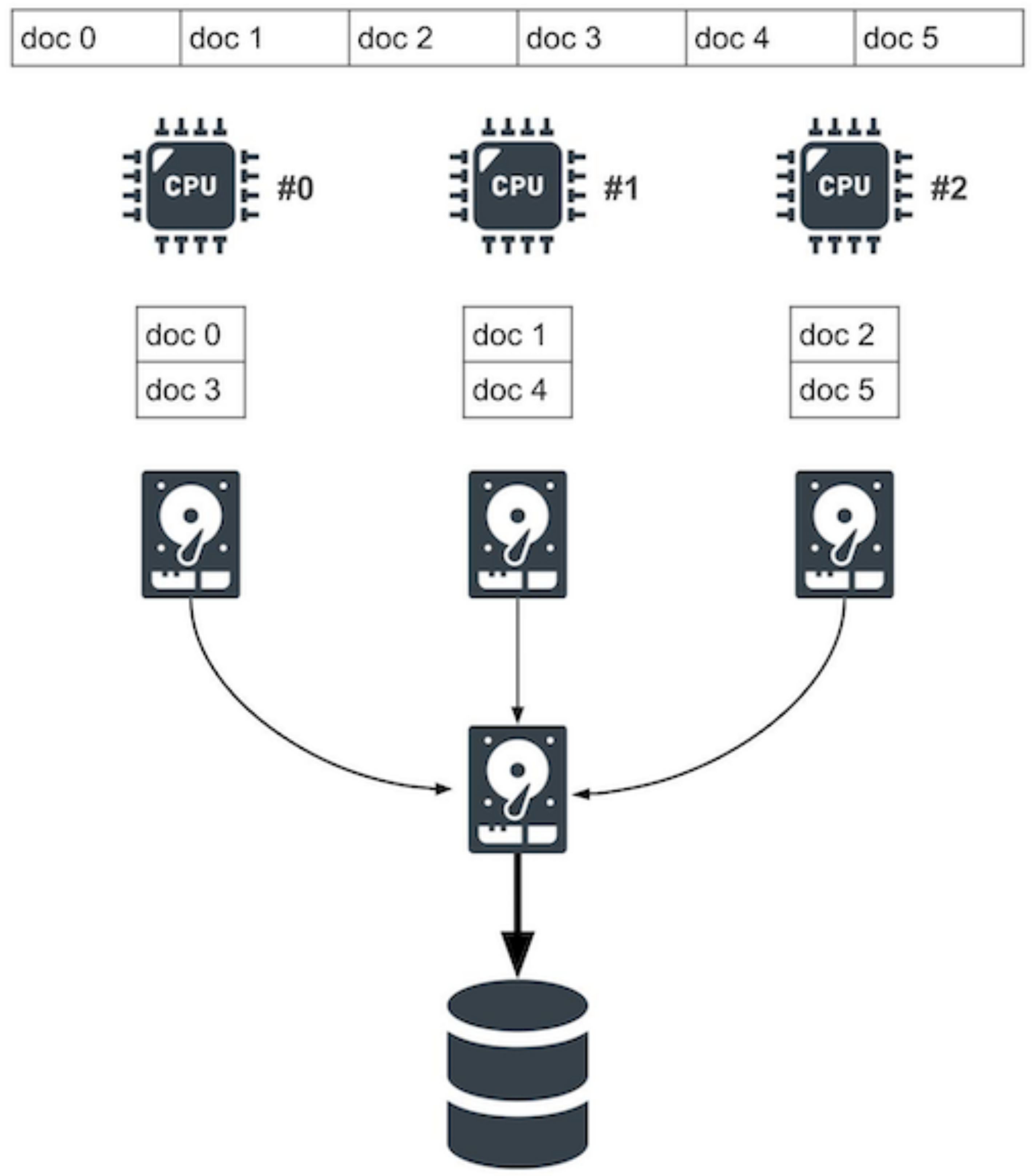
Mainly used in the *Typo* criterion to fetch typo derivations using the *levenshtein* algorithm.

Indexing Documents


Synchronous
Asynchronous
HTTP response
error: message
:err
:err
HTTP call
validate input
create task
:ok
HTTP response
taskId: i64
status: pending
task DB
update task
status: failed
:err
:err
process task
write data in index
index DB
:ok
update task
status: processed
data to index

Process Task

1. Each thread is assigned a list of documents to index.
2. Data extracted from the documents is written to disk, in parallel.
3. Once the indexing process is finished, all the disk files are merged into a final one.
4. This final file is written to the database.



The

End

The *Words* criterion

1. Fetch the internal ids associated with each query word.
2. Do the unions and intersection depending on the query tree shape.
3. If not enough documents are found, remove the last query word.

The *Typo* criterion

1. Fetch the internal ids associated with each query word without any typo.
2. Do the unions and intersection depending on the query tree shape.
3. If not enough documents are found, rerun step 1 and accept one more typo.

The *Proximity* criterion

1. For each AND operation we find the documents with all those words at a proximity of one.
2. Do the unions and intersection depending on the query tree shape.
3. If not enough documents are found, rerun step 1 and search with a bigger proximity.

The *Desc(release year)* criterion

1. Do a simple sort by *release year* of the internal ids.
2. Return each bucket in order.