Searching and Indexing

Indexing

- *Indexing is the initial part of all search applications.*

- *Its goal is to process the original data into a highly efficient cross-reference lookup in order to facilitate rapid searching.*

- *The job is simple when the content is already textual in nature and its location is known.*

Indexing

- **Steps**:
- ***acquiring the content****.*
  - *This process gathers and* scopes the content that needs to be indexed.
- ***build documents***
  - *The raw content that* needs to be indexed has to be translated into the units (usually called documents) used by the search application.
- ***document analysis***
  - The textual fields in a document cannot be indexed directly. Rather, the text has to be broken into a series of individual atomic elements called tokens.
  - This happens during the *document analysis step. Each token corresponds roughly to a word in* the language, and the analyzer determines how the textual fields in the document are divided into a series of tokens.
- ***index the document***
  - The final step is to *index the document. During the indexing step, the document* is added to the index.

Lucene

- Lucene is a free, open source project implemented in Java.
- licensed under Apache Software Foundation.
- Lucene itself is a single JAR (Java Archive) file, less than 1 MB in size, and with no dependencies, and integrates into the simplest Java stand-alone console program as well as the most sophisticated enterprise application.
- Rich and powerful full-text search library.
- Lucene to provide full-text indexing across both database objects and documents in various formats (Microsoft Office documents, PDF, HTML, text, and so on).
- supporting full-text search using Lucene requires two steps:
- ***creating a lucence index***
  - *creating a lucence index* on the documents and/or database objects.
- ***Parsing looking up***
  - *parsing* the user query and *looking up* the prebuilt index to answer the query.

Architecture

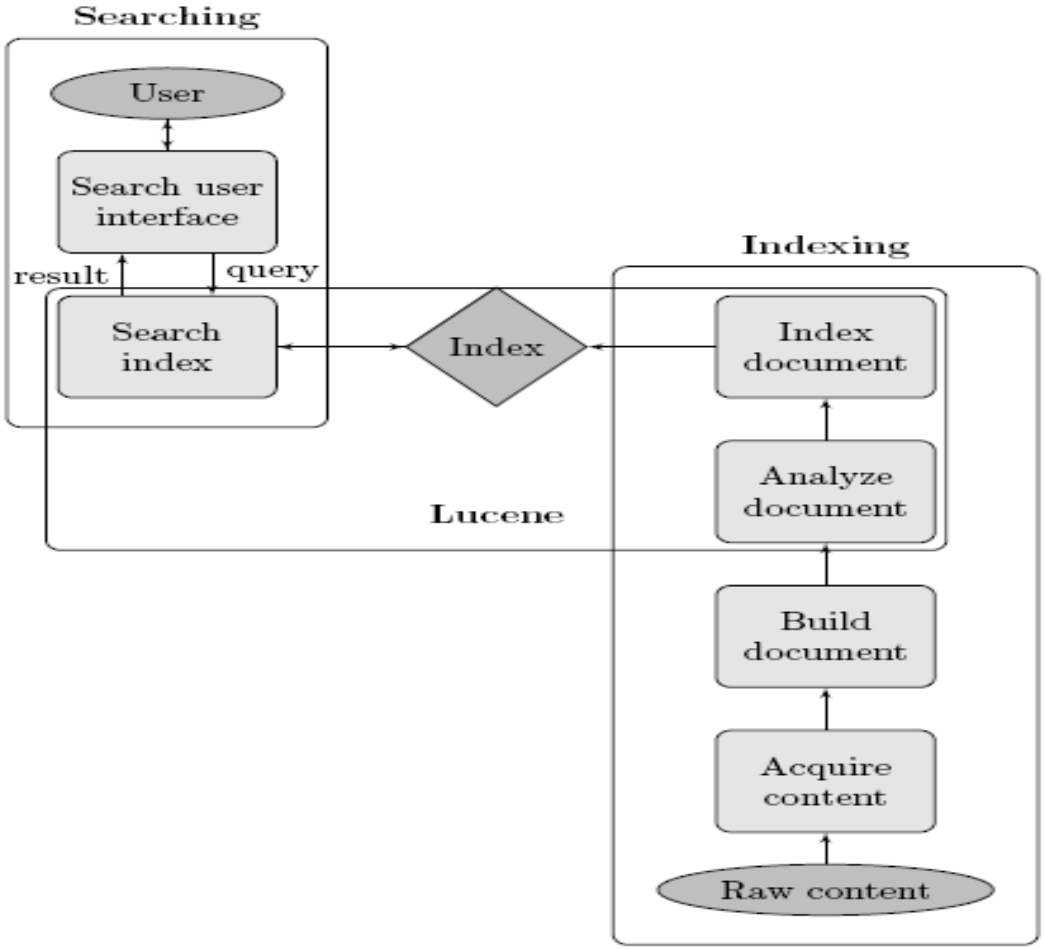
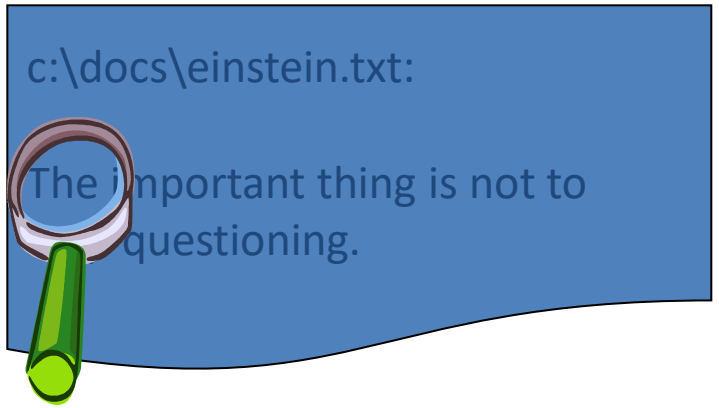


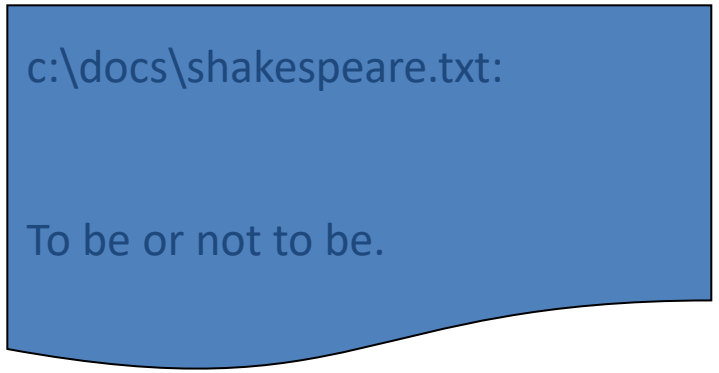
Figure 3.1: Typical components of search application architecture with Lucene components highlighted

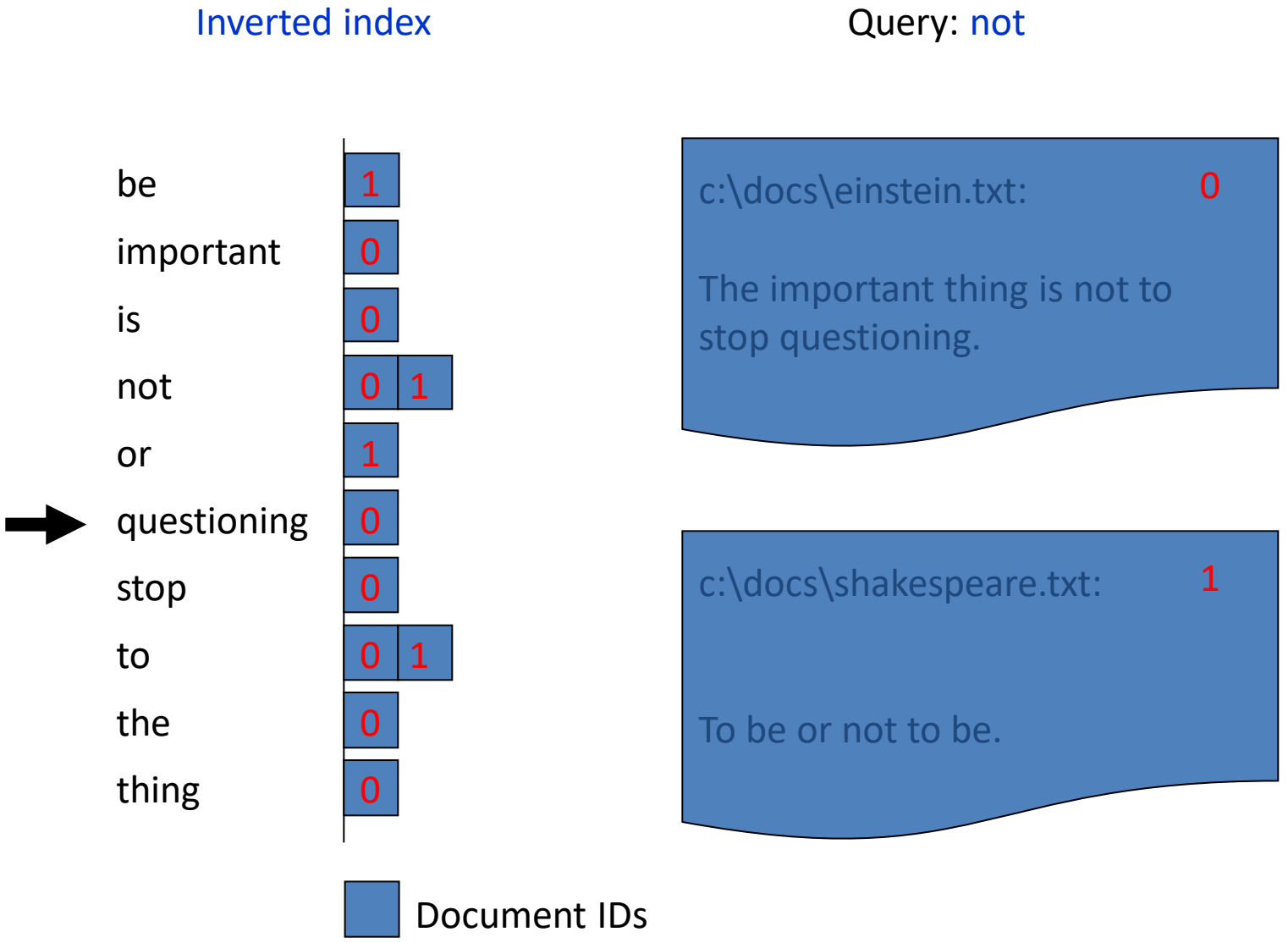
Query: not

String comparison slow!

Solution: Inverted index

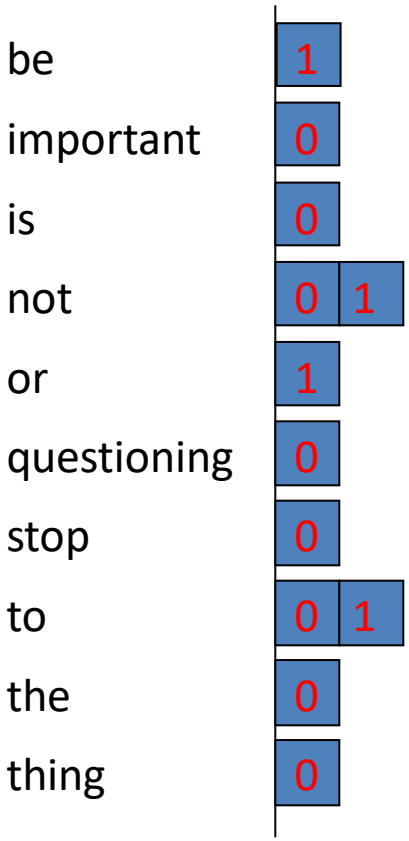


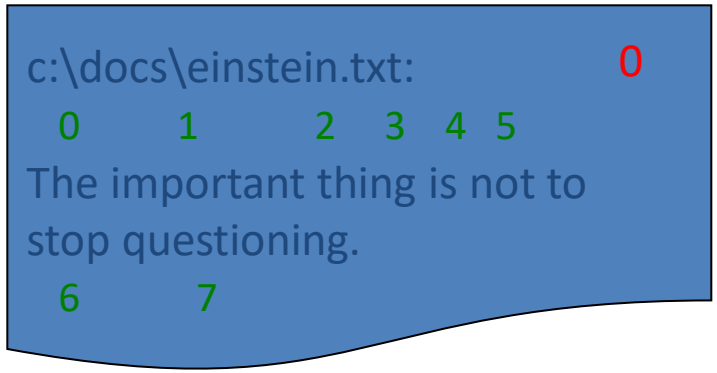


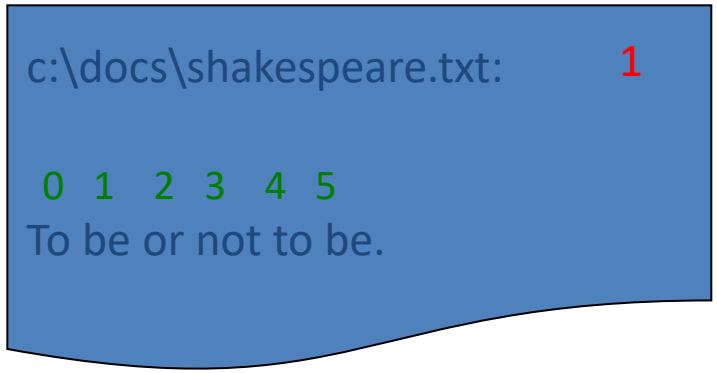

Inverted index
Query: not
be
important
is
not
or
questioning
stop
to
the
thing
1
0
0
0 1
1
0
0
0 1
0
0
Document IDs
c:\docs\einstein.txt:
0
The important thing is not to stop questioning.
c:\docs\shakespeare.txt:
1
To be or not to be.

Inverted index

Query: ”not to”



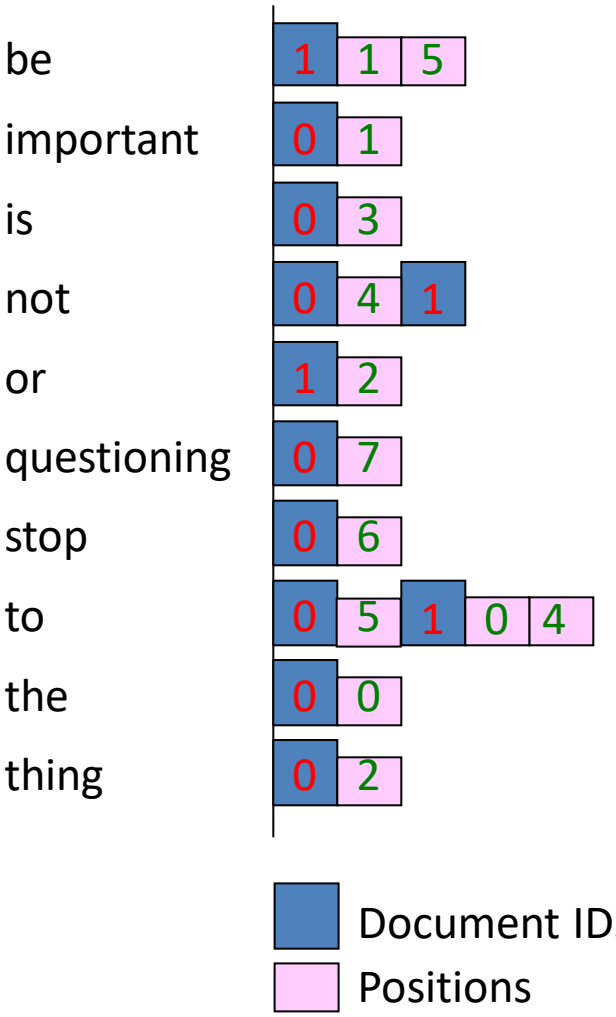


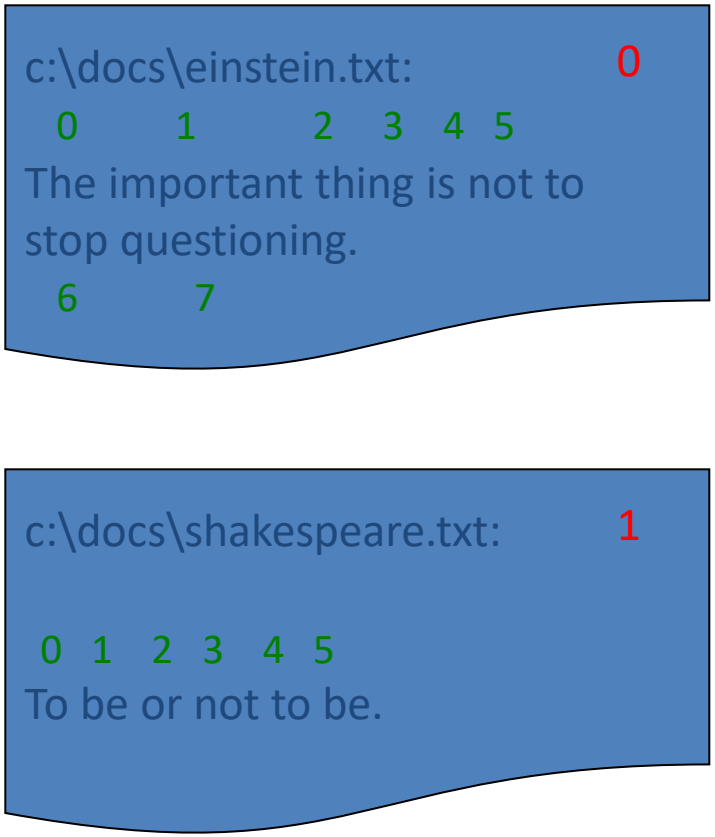


Document IDs

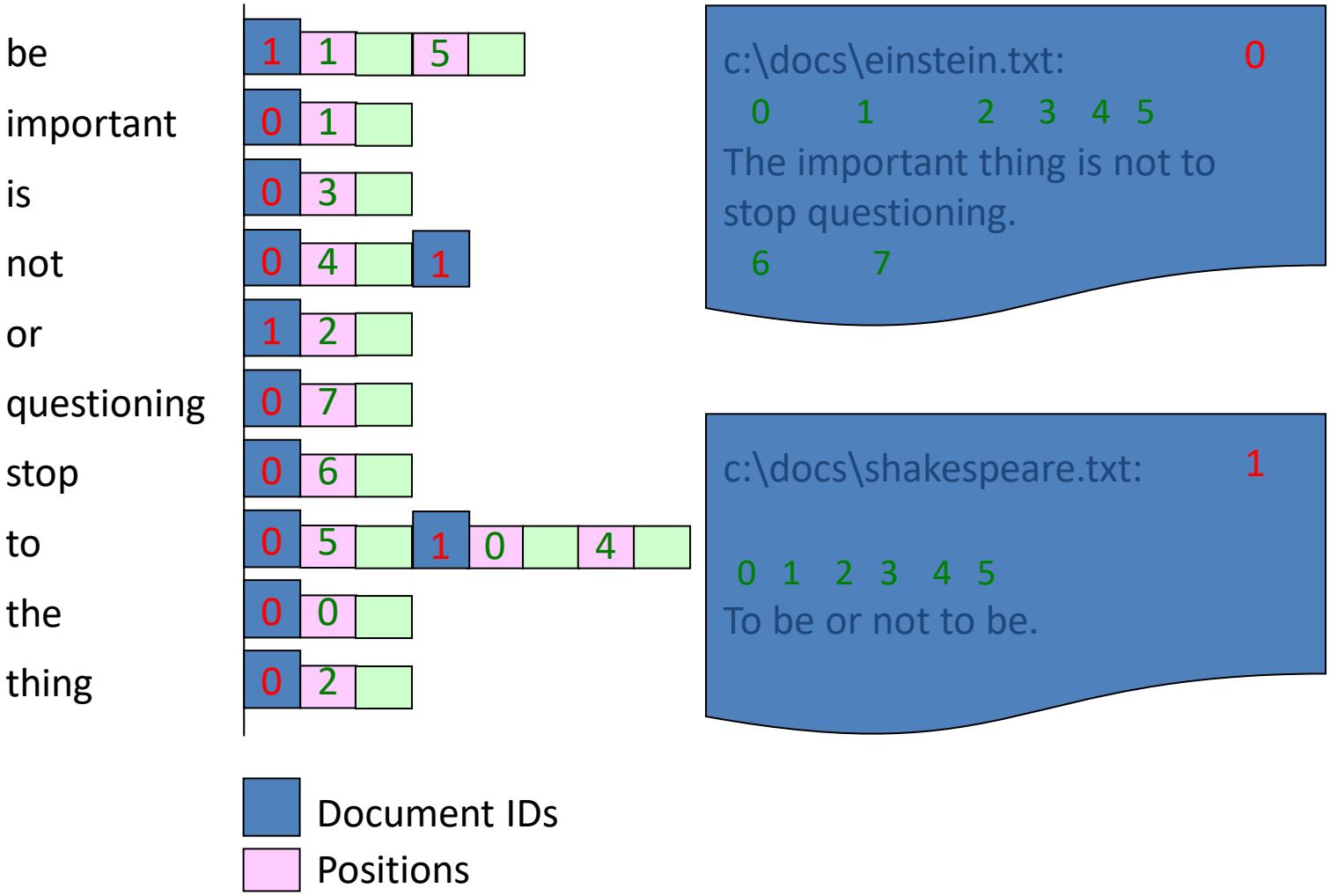
Inverted index

Query: ”not to”





Inverted index with Payloads



Creating an Index (IndexWriter Class)

- *The first step in implementing full-text searching with Lucene is to build an index.*

- *To create an index, the first thing that need to do is to create an IndexWriter object.*

  - *The IndexWriter object is used to create the index and to add new index entries (i.e., Documents) to this index. You can create an IndexWriter as follows*

    - *IndexWriter indexWriter = new IndexWriter("index-directory", new StandardAnalyzer(), true);*

Parsing the Documents (Analyzer Class)

- The job of Analyzer is to "parse" each field of your data into indexable "tokens" or keywords.
- Several types of analyzers are provided out of the box. Table 1 shows some of the more interesting ones.

- **StandardAnalyzer**
  - A sophisticated general-purpose analyzer.
- **WhitespaceAnalyzer**
  - A very simple analyzer that just separates tokens using white space.
- **StopAnalyzer**

  Removes common English words that are not usually useful for indexing.
- **SnowballAnalyzer**

  An interesting experimental analyzer that works on word roots (a search on *rain* should also return entries with *raining*, *rained*, and so on).

Adding a Document/object to Index (Document Class**)**

- .To index an object, we use the Lucene Document class, to which we add the fields that you want indexed.

- Document doc = new Document();
- doc.add(new Field("description", hotel.getDescription(), Field.Store.YES, Field.Index.TOKENIZED));

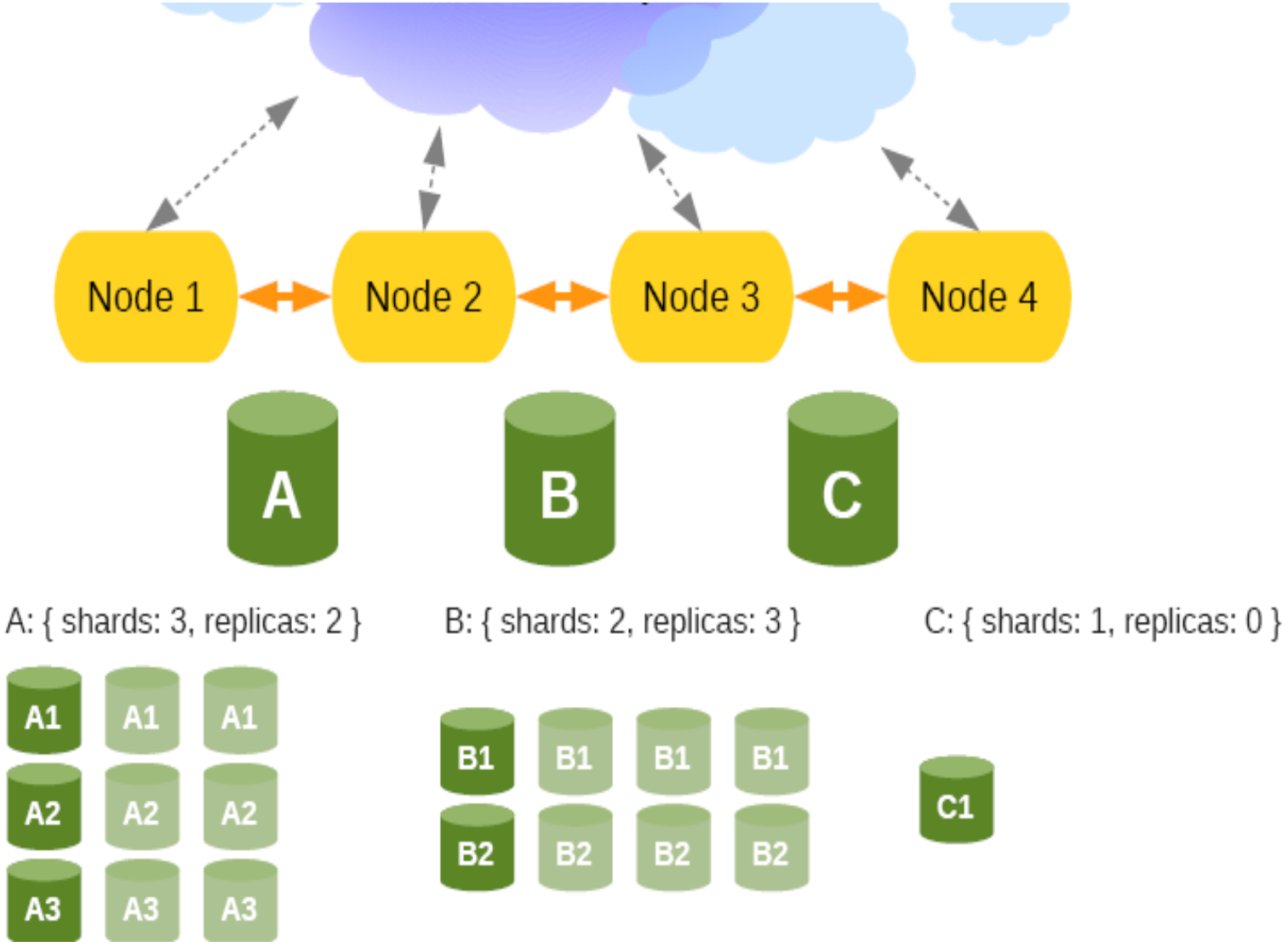
Elasticsearch (NO-SQL)

What is ElasticSearch ?

- Open Source (No-sql DB)
- Distributed (cloud friendly)
- Highly-available
- Designed to speak JSON (JSON in, JSON out)

Highly available

- For each index you can specify:
- **Number of shards**
  – Each index has fixed number of shards
- **Number of replicas**
  – Each shard can have 0-many replicas, can be changed dynamically


Node 1
Node 2
Node 3
Node 4
A
B
C
A: { shards: 3, replicas: 2 }
B: { shards: 2, replicas: 3 }
C: { shards: 1, replicas: 0 }
A1
A1
A1
A2
A2
A2
A3
A3
A3
B1
B1
B1
B1
B2
B2
B2
B2
C1

Admin API

- Indices
  - Status
  - CRUD operation
  - Mapping, Open/Close, Update settings
  - Flush, Refresh, Snapshot, Optimize
- Cluster
  - Health
  - State
  - Node Info and stats
  - Shutdown

Rich query API

- There is rich Query DSL for search, includes:
- Queries
  - Boolean, Term, Filtered, MatchAll, ...
- Filters
  - And/Or/Not, Boolean, Missing, Exists, ...
- Sort

  -order:asc, order:desc
- Facets

  Facets allows to provide aggregated data for the search request

  –Terms, Term, Terms_stat, Statistical,….

Scripting support

- There is a support for using scripting languages
- mvel (default)
- JS
- Groovy
- Python

ES_Query syntax

```
{
    “fields”:[“name”,”Id”]
}
----------------------------------------
{
    “from”:0,
    “size”:100,
    “fields”:[“name”,”Id”]
}
```

ES_Query syntax

```
{
“fields”:[“name”,”Id”]
      “query”:{
            }
}
----------------------------------------
{“from”:0, “size”:100, “fields”:[“name”,”Id”],
      “query”:{
            “filtered”:{
                  “query”:{
                        “match_all”:{}
                  },
                        “filter”:{
                              “terms”:{
                                    “name”:[“ABC”,”XYZ”]
                                    }
                              }
                        }
            }
}
```

Filter_syntax

```
{“filter”:”terms”:{
    “name”:[“ABC”,”XYZ”]
}
----------------------
{
“filter”:
    {
    “range”:
        {
        “from”:0,
        “to”:10
        }
    }
}
```

Facets_Syntax

```
{
"facets":{
        "facet_name":{
                        "terms":{
                        "name":["ABC", "XYZ"]
                                }
                        }
                }
        }
```

Facets_Syntax

```
{
“facets”:{
        “facet_name”:{
                “term_stats”:{
                “key_field”:”name”
                “value_field”:”salary”
                        }
                }
                }
        }
```

Assignment

- *Indexing and Searching with LUCENE*