Lucene Tutorial

Open Source IR Library

Overview

- Open source IR systems
- Lucene Intro
- Installation
- Lucene core classes
- Scoring
- Index format
- Useful resources.

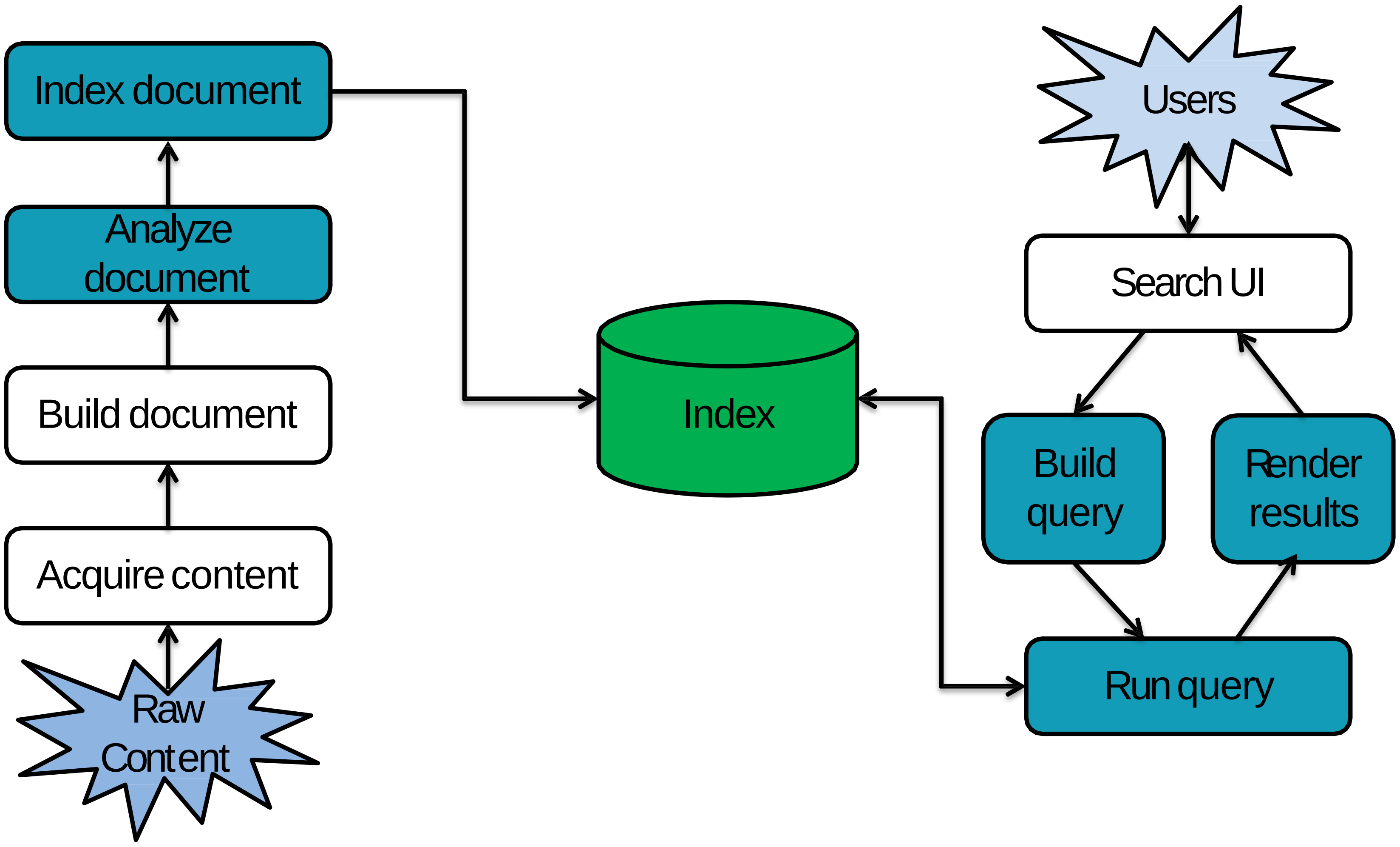
Open source IR systems

- IR systems to reduce information overload.
- Widely used academic systems
  - Terrier (Java, U. Glasgow) http://terrier.org
  - Indri/Galago/Lemur (C++ (& Java), U. Mass & CMU)
  - Zettair (RMIT U.)..
- Widely used non-academic open source systems
  - **Lucene**
    - Things built on it: Solr, ElasticSearch
  - A few others (Xapian, …)

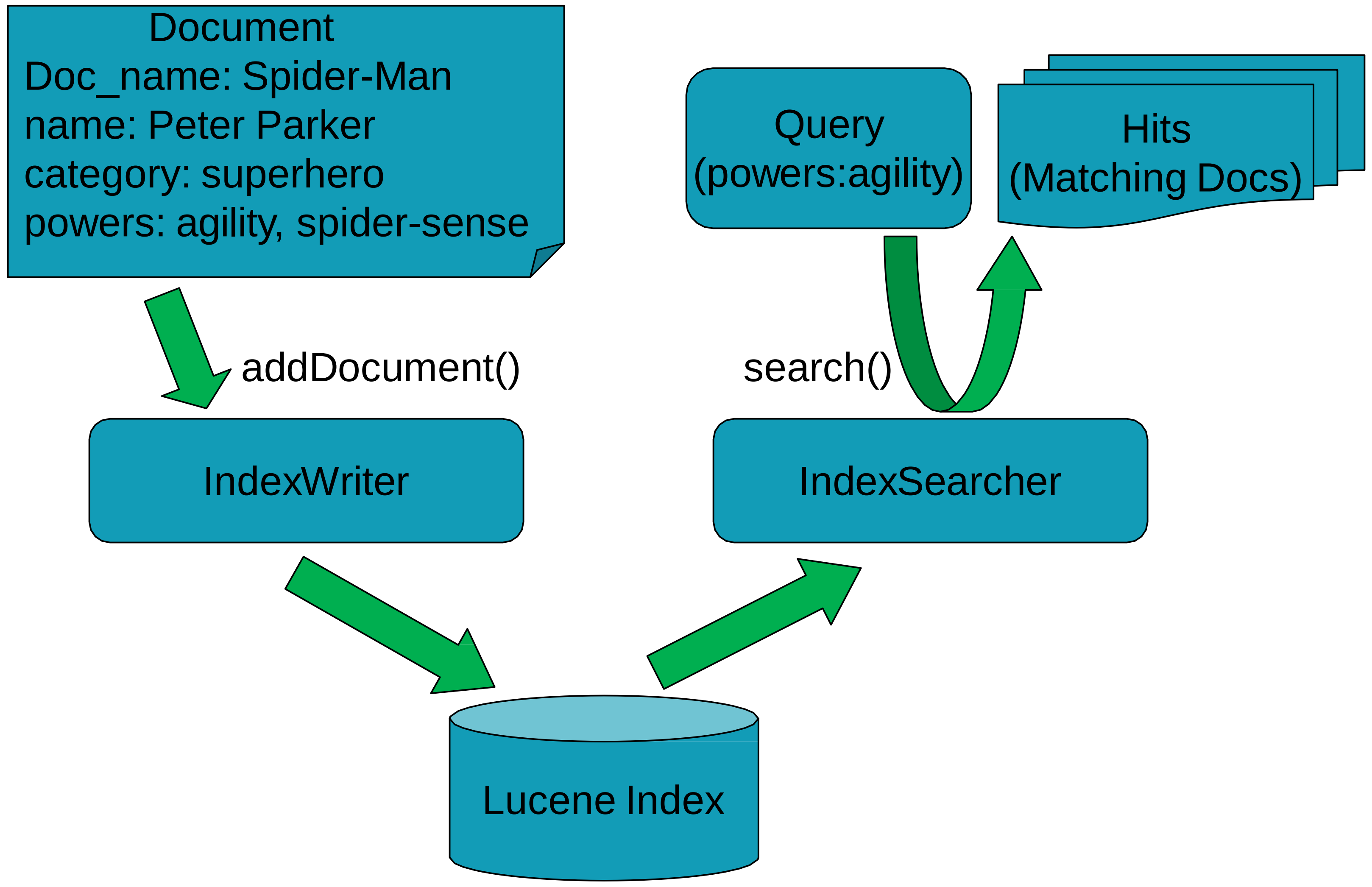
Lucene

- Open source Java library for indexing and searching
  - Lets you add search to your application
  - Not a complete search system by itself
  - Written by Doug Cutting in 1990
- Used by: Twitter, LinkedIn, Zappos, CiteSeer, Eclipse, …
  - …and many more (see http://wiki.apache.org/lucene-java/PoweredBy)
- Ports/integrations to other languages
  - C/C++, C#, Ruby, Perl, Python (Pylucene), PHP, …

Basic Application



Lucene in search systems



Installation

- Apache Lucene
  https://archive.apache.org/dist/lucene/java/3.6.2/
  after downloading extract the files to the desktop.
- JDK/JRE
- Eclipse

Add External Jar Files

- lucene-core-3.6.2

Demo Files

- LuceneConstants- provide various constants to be used across the sample application.
- TextFileFilter- Code is used as a **.txt file** filter.
- Indexer- Code to create a Lucene index.
- Searcher- Code to search a Lucene Index.
- LuceneTester- Code used to test the indexing and search capability of lucene library.

Set directory Paths

- IndexFiles

  set docsPath= “Documents folder path”

  set indexPath = “Index folder path”

- SearchFiles

  set index= “Index folder path”

- Write a query
- Top results

Core indexing classes

- IndexWriter
  - Central component that allows you to create a new index, open an existing one, and add, remove, or update documents in an index
  - Built on an IndexWriterConfig and a Directory
- Directory
  - Abstract class that represents the location of an index
- Analyzer
  - Extracts tokens from a text stream

Creating an IndexWriter

```
Import org.apache.lucene.analysis.Analyzer;
import org.apache.lucene.index.IndexWriter;
import org.apache.lucene.index.IndexWriterConfig;  import
org.apache.lucene.store.Directory;
...
private IndexWriter writer;
public Indexer(String dir) throws IOException {//this //directory will contain
    the indexes
      Directory indexDirectory =
         FSDirectory.open(new File(indexDirectoryPath));

      //create the indexer
      writer = new IndexWriter(indexDirectory,
         new StandardAnalyzer(Version.LUCENE_36),true,
         IndexWriter.MaxFieldLength.UNLIMITED);
   }
```

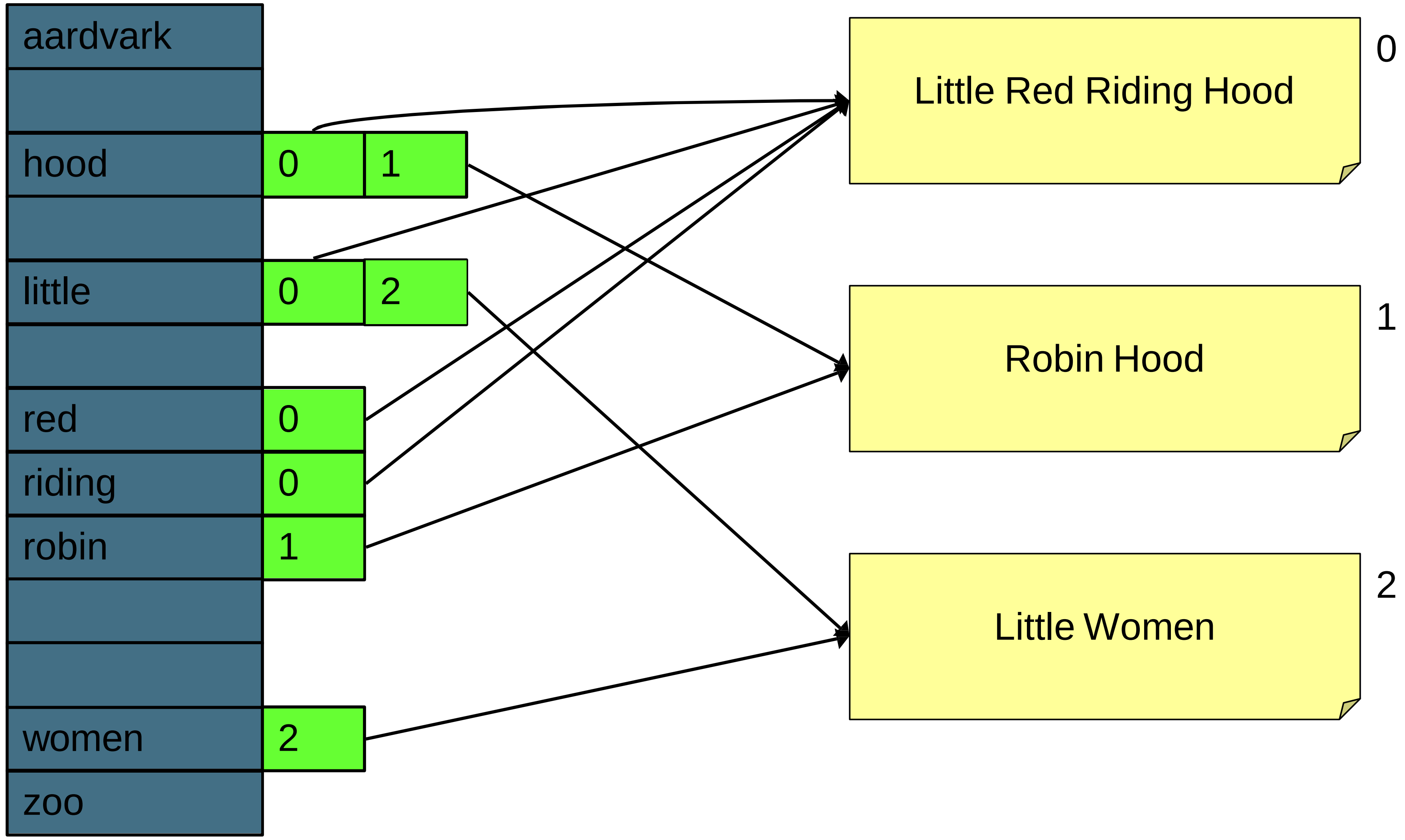
Index a Document with IndexWriter

```
private IndexWriter writer;
...
private void indexFile(File file) throws IOException {
    System.out.println("Indexing
        "+file.getCanonicalPath());
    Document document = getDocument(file);
    writer.addDocument(document);
  }
```

The Index

- The Index is the kind of inverted index we know and love
  - natural ordering of docIDs
  - encodes both term frequencies and positional information
- APIs to customize the codec

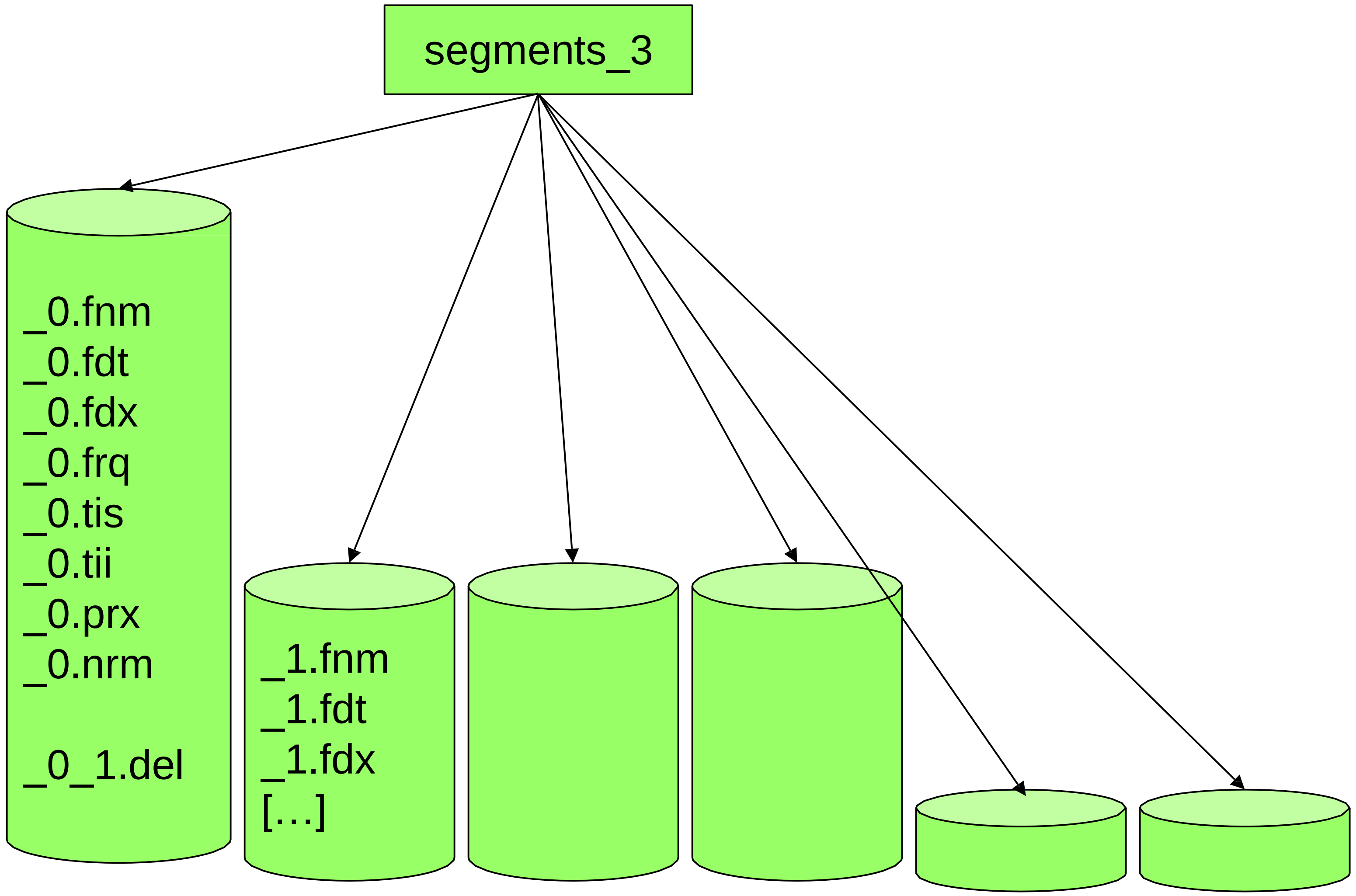
Inverted Index



Index format

- Each Lucene index consists of one or more segments
  - A segment is a standalone index for a subset of documents
  - All segments are searched
  - A segment is created whenever IndexWriter flushes adds/deletes
- Periodically, IndexWriter will merge a set of segments into a single segment
  - Policy specified by a MergePolicy

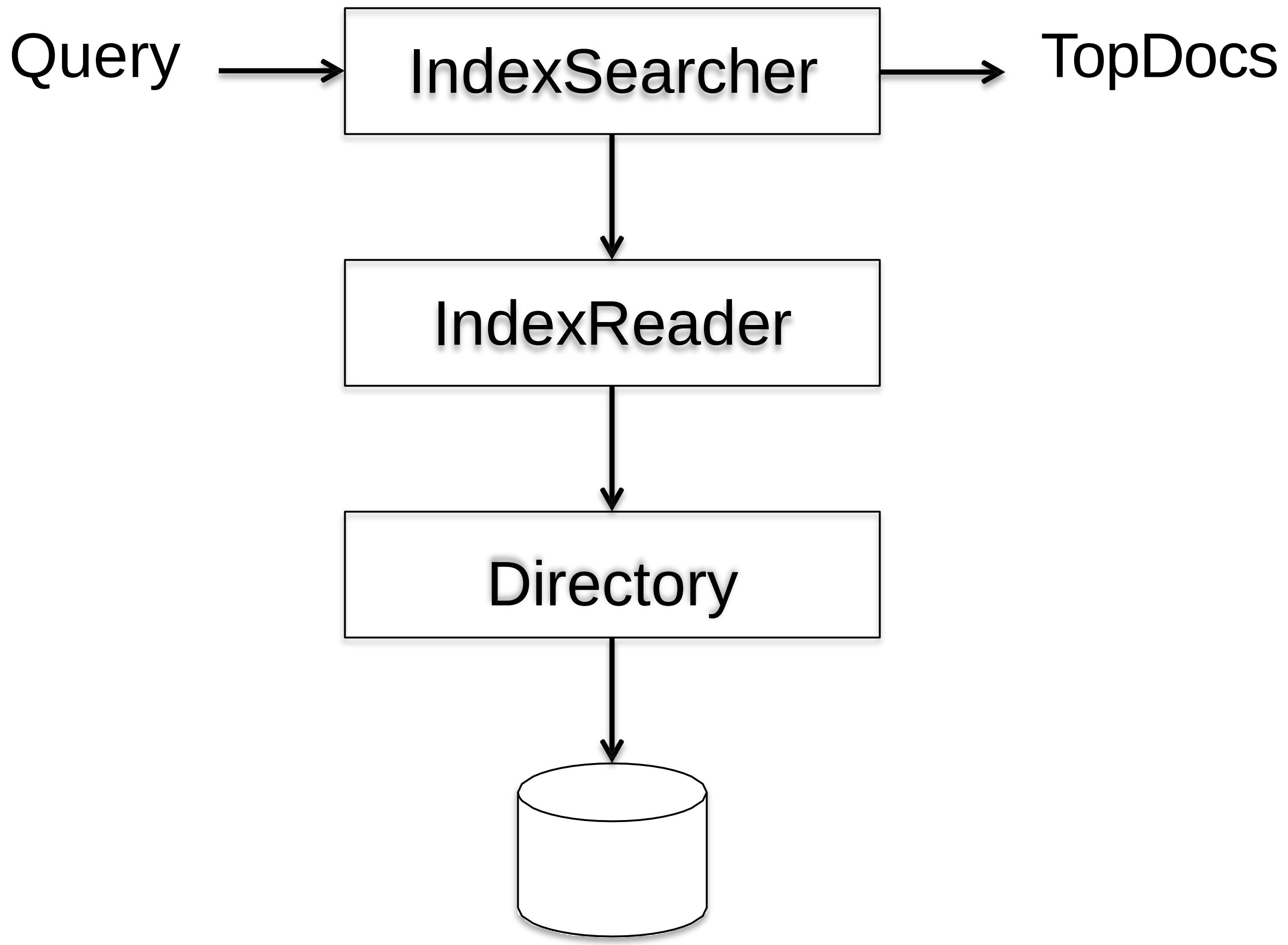
Index Structure



Core searching classes

- IndexSearcher
  - Central class that exposes several search methods on an index
  - Accessed via an IndexReader
- Query
  - Abstract query class. Concrete subclasses represent specific types of queries, e.g., matching terms in fields, boolean queries, phrase queries, …
- QueryParser
  - Parses a textual representation of a query into a Query instance

IndexSearcher



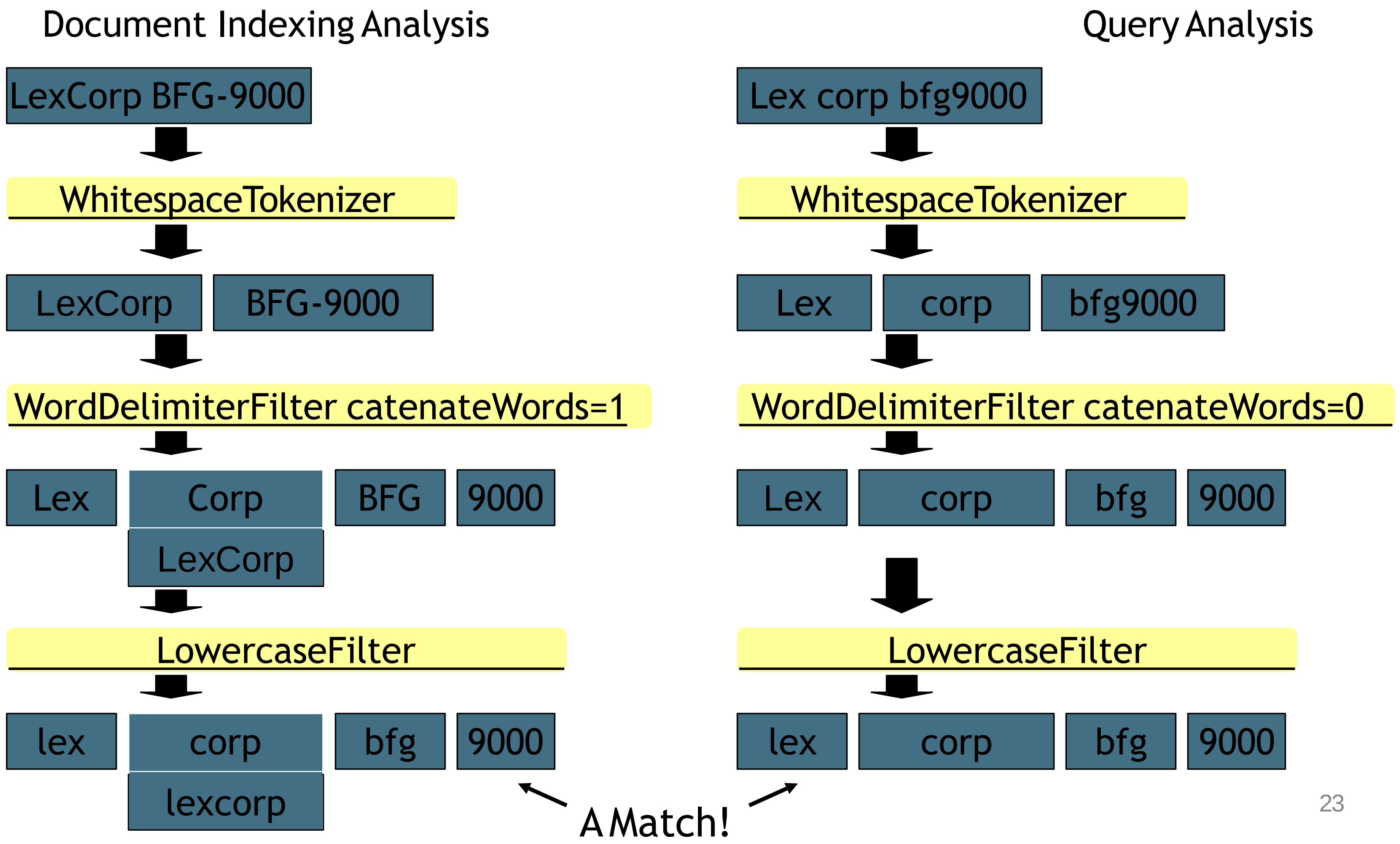
Creating an IndexSearcher

```
import org.apache.lucene.search.IndexSearcher;
...
public Searcher(String indexDirectoryPath) throws IOException {
      Directory indexDirectory = FSDirectory.open(new
File(indexDirectoryPath));
 indexSearcher = new IndexSearcher(indexDirectory);
}
```

Query and QueryParser and search() returns TopDocs

```
import org.apache.lucene.queryParser.QueryParser;
import org.apache.lucene.search.Query;
public static void search(String indexDir, String q) throws IOException,
ParseException {
    ...
queryParser = new
QueryParser(Version.LUCENE_36,LuceneConstants.CONTENTS,
      new StandardAnalyzer(Version.LUCENE_36));
}
public TopDocs search( String searchQuery) throws IOException,
ParseException {
    query = queryParser.parse(searchQuery);
    return indexSearcher.search(query, LuceneConstants.MAX_SEARCH);
  }
```

Analysis & Search Relevancy



Scoring

- VSM – Vector Space Model
- tf – term frequency: numer of matching terms in field
- lengthNorm – number of tokens in field
- idf – inverse document frequency
- coord – coordination factor, number of matching terms
- Boosting techniques

http://lucene.apache.org/java/docs/scoring.html

Useful Resources

- https://lucene.apache.org/core/7_0_1/index.html
- https://lucene.apache.org/core/3_5_0/scoring.html
- https://lucene.apache.org/core/2_9_4/fileformats.pdf
- https://lucene.apache.org/core/7_2_1/demo/overview-summary.html
- https://www.youtube.com/watch?v=pVDVURw_AJQ