AI6122 Text Data Management & Analysis

Lecture: Introduction to Lucene

Partially based on https://web.stanford.edu/class/cs276/handouts/lecture-**lucene**.pptx

Open source IR systems

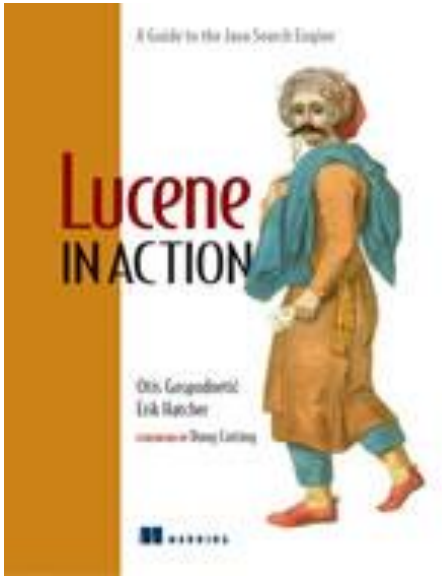
- Widely used academic systems
  - Terrier (Java, U. Glasgow) http://terrier.org
  - Indri/Galago/Lemur (C++ & Java, U. Mass & CMU)
  - Tail of others (Zettair, …)

- Widely used non-academic open source systems
  - **Lucene**
    - Things built on it: Solr, ElasticSearch
    - http://lucene.apache.org/solr/
    - https://www.elastic.co/
  - A few others (Xapian, …)

Lucene

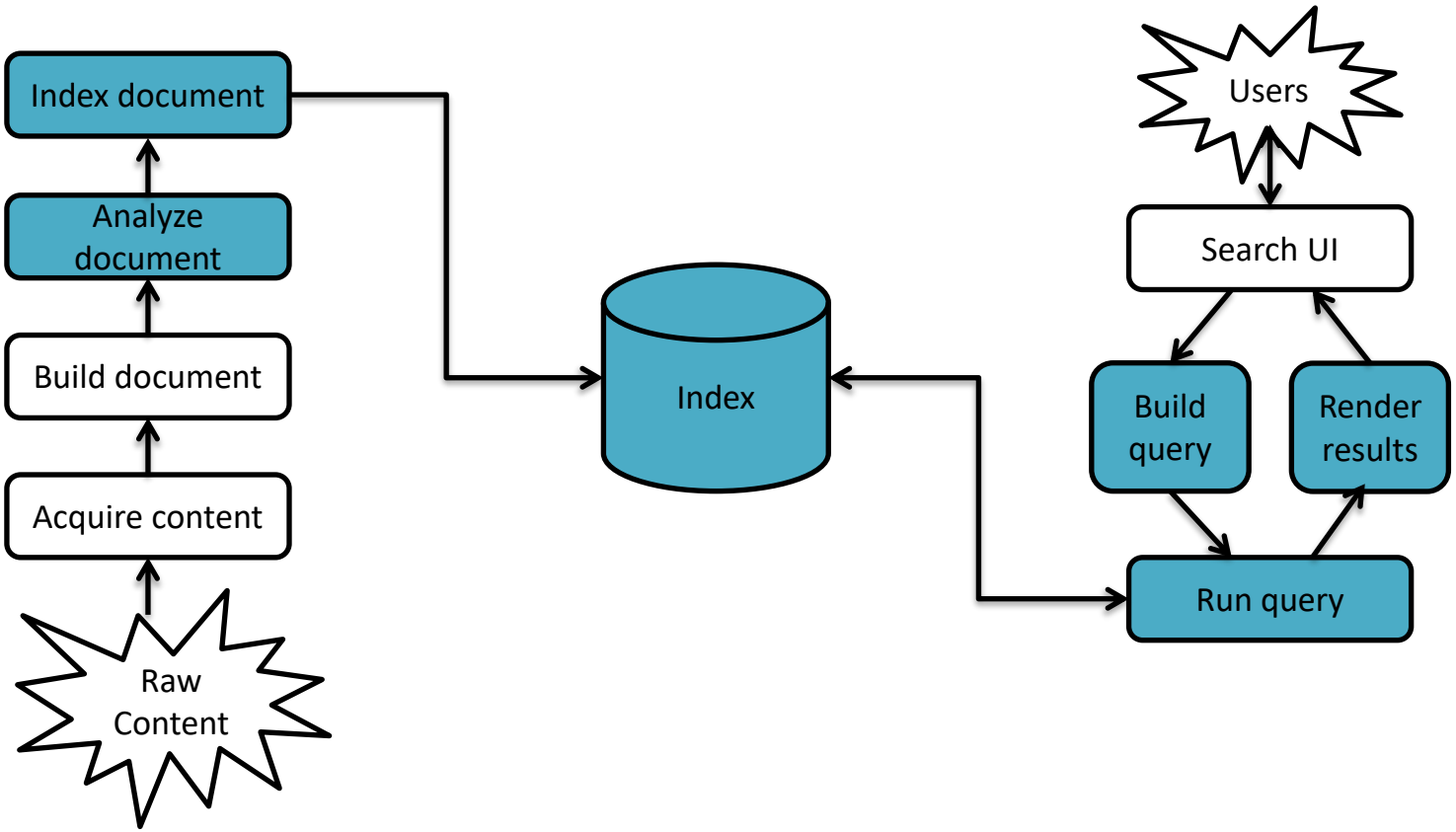
- Open source Java library for indexing and searching
  - Lets you add search to your application
  - Not a complete search system by itself
  - Written by Doug Cutting
- Used by: Twitter, LinkedIn, Zappos, CiteSeer, Eclipse, …
  - … and many more (see http://wiki.apache.org/lucene-java/PoweredBy)
-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

Based on “Lucene in Action”



- By Michael McCandless, Erik Hatcher, Otis Gospodnetic
- The book covers Lucene 3.0.1.
- The current version is 8.4.1
  - http://lucene.apache.org/core/
  - Lots of changes to APIs (compared with Lucene 3)
  - Lots of changes because of Java changes
  - Current releases require Java 8 or greater.

Lucene in a search system



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
  - https://lucene.apache.org/core/6_4_1/core/org/apache/lucene/index/IndexWriter.html
  - Built on an `IndexWriterConfig` and a `Directory`

- `Directory`
  - **Abstract** class that represents the location of an index
  - A Directory is a flat list of files. Files may be written once, when they are created. Once a file is created it may only be opened for read, or deleted.
  - https://lucene.apache.org/core/6_4_1/core/org/apache/lucene/store/Directory.html

- `Analyzer`
  - Extracts tokens from a text stream

Creating an IndexWriter

```
public class QAIndexer {

        private IndexWriter writer = null;

        public QAIndexer(String dir) throws IOException {
                Directory indexDir = FSDirectory.open(Paths.get(dir));
                Analyzer analyzer = new StandardAnalyzer();
                IndexWriterConfig cfg = new IndexWriterConfig(analyzer);
                cfg.setOpenMode(OpenMode.CREATE);
                writer = new IndexWriter(indexDir, cfg);
        }
}
```

Core indexing classes

- `Document`
  - Documents are the **unit** of indexing and search.
  - A Document is a set of **fields**.
  - Each field has a **name** and a textual **value**. A field may be stored with the document, in which case it is returned with search hits on the document.
  - http://lucene.apache.org/core/6_4_1/core/org/apache/lucene/document/Document.html

- `Field`
  - Lucene `Field`s can represent both “fields” and “zones” as described in the class
  - Lots of Field classes

Field

- A field is a section of a Document.
  - Each field has three parts: **name**, **type** and **value**. Values may be text (`String`, `Reader` or pre-analyzed `TokenStream`), binary (byte[]), or numeric (a Number).
  - Fields are optionally stored in the index, so that they may be returned with hits on the document.

- Most users should use one of the sugar subclasses:
  - `TextField`: Reader or String **indexed for full-text search**
  - `StringField`: String indexed verbatim **as a single token**
  - `IntPoint/LongPoint/FloatPoint/DoublePoint`: int/long/float/double indexed for **exact/range** queries.
  - `SortedDocValuesField`: byte[] indexed column-wise for **sorting/faceting**
  - `SortedSetDocValuesField`: SortedSet<byte[]> indexed column-wise for **sorting/faceting**
  - `NumericDocValuesField`: long indexed column-wise for sorting/faceting
  - `SortedNumericDocValuesField`: SortedSet<long> indexed column-wise for **sorting/faceting**
  - `StoredField`: Stored-only value for retrieving in summary results

Creating a document for indexing

```
protected Document getDocument(File f) throws Exception {
        Document doc = new Document();
        doc.add(new TextField("contents", new FileReader(f), Field.Store.YES));
        doc.add(new StringField("filename", f.getName(), Field.Store.YES));
        return doc;
}
```

```
protected Document getDocument(String question, String answer) throws Exception {
        Document doc = new Document();
        doc.add(new TextField("question", question, Field.Store.YES));
        doc.add(new TextField("answer", answer, Field.Store.YES));
        return doc;
}
```

Index a document, and close IndexWriter

```
private void indexFile(File f) throws Exception {
        Document doc = getDocument(f);
        writer.addDocument(doc);
}

-------------------------------------------------------------------------------

while (in.hasNextLine()) {
        try {
                jLine = in.nextLine().trim();
                JSONObject jObj = new JSONObject(jLine);
                String question = jObj.getString("question");
                String answer = jObj.getString("answer");
                Document doc = getDocument(question, answer);
                writer.addDocument(doc);
        } catch (Exception e) { }
}
```

```
public void close() throws IOException
{
        writer.close();
}
```

The Index

- The **Index** is the kind of inverted index we have studied

- The default codec is:
  - variable-byte and fixed-width encoding of delta values
  - multi-level skip lists
  - natural ordering of docIDs
  - encodes both term frequencies and positional information

- Summary
  - https://lucene.apache.org/core/6_4_1/core/org/apache/lucene/codecs/lucene62/package-summary.html

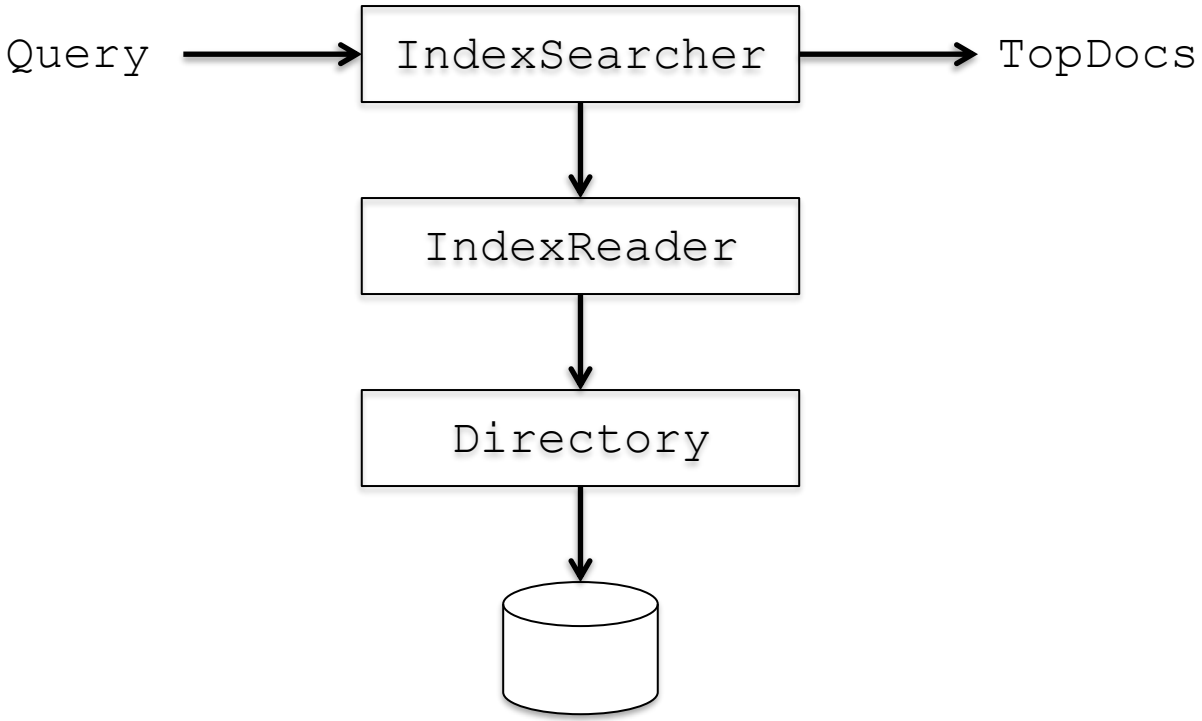
Core searching classes

- **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, QueryBuilder**
  - Parses a textual representation of a query into a Query instance

Search: from Query to TopDocs



Create a Searcher, and Search

```
public QASearcher(String dir) {
        try {
                lReader = DirectoryReader.open(FSDirectory.open(Paths.get(dir)));
                lSearcher = new IndexSearcher(lReader);
        } catch (Exception e) {
                e.printStackTrace();
        }
}
```

```
public ScoreDoc[] search(String field, String keywords, int numHits) {
        QueryBuilder builder = new QueryBuilder(new StandardAnalyzer());
        Query query = builder.createBooleanQuery(field, keywords);
        ScoreDoc[] hits = null;
        try {
                TopScoreDocCollector collector = TopScoreDocCollector.create(numHits);
                lSearcher.search(query, collector);
                hits = collector.topDocs().scoreDocs;
        } catch (Exception e) {
                e.printStackTrace();
        }
        return hits;
}
```

Lucene supports many query types

- TermQuery
- BooleanQuery
- WildcardQuery
- PhraseQuery
- PrefixQuery
- MultiPhraseQuery
- FuzzyQuery
- RegexpQuery
- TermRangeQuery
- PointRangeQuery
- ConstantScoreQuery
- DisjunctionMaxQuery
- MatchAllDocsQuery

http://lucene.apache.org/core/6_4_1/core/org/apache/lucene/search/Query.html

Core searching classes

- `TopDocs`
  - Contains references to the top documents returned by a search

- `ScoreDoc`
  - Represents a single search result

```
public void printResult(ScoreDoc[] hits) {
        for (ScoreDoc hit : hits) {
                System.out.println("\nResult " + i + "\tDocID: " + hit.doc + "\t Score: " + hit.score);
                try {
                        System.out.println("Q: " + lReader.document(hit.doc).get("question"));
                        System.out.println("A: " + lReader.document(hit.doc).get("answer"));
                } catch (Exception e) { }
        }
}
```

Closing `IndexSearcher`

```
public void close() {
        try {
                if (lReader != null) {
                        lReader.close();
                }
        } catch (Exception e) {
                e.printStackTrace();
        }
}
```

```
public QASearcher(String dir) {
        try {
                lReader = DirectoryReader.open(FSDirectory.open(Paths.get(dir)));
                lSearcher = new IndexSearcher(lReader);
        } catch (Exception e) {
                e.printStackTrace();
        }
}
```

Analyzer

- Tokenizes the input text

- Common `Analyzer`s
  - `WhitespaceAnalyzer`
    Splits tokens on whitespace
  - `SimpleAnalyzer`
    Splits tokens on non-letters, and then lowercases
  - `StopAnalyzer`
    Same as `SimpleAnalyzer`, but also removes stop words
  - `StandardAnalyzer`
    Most sophisticated analyzer that knows about certain token types, lowercases, removes stop words, ...

Analysis example

- Example sentence: “The quick brown fox jumped over the lazy dog”

- `WhitespaceAnalyzer`
  - [The] [quick] [brown] [fox] [jumped] [over] [the] [lazy] [dog]

- `SimpleAnalyzer`
  - [the] [quick] [brown] [fox] [jumped] [over] [the] [lazy] [dog]

- `StopAnalyzer`
  - [quick] [brown] [fox] [jumped] [over] [lazy] [dog]

- StandardAnalyzer
  - [quick] [brown] [fox] [jumped] [over] [lazy] [dog]

Another analysis example

- “XY&Z Corporation – xyz@example.com”

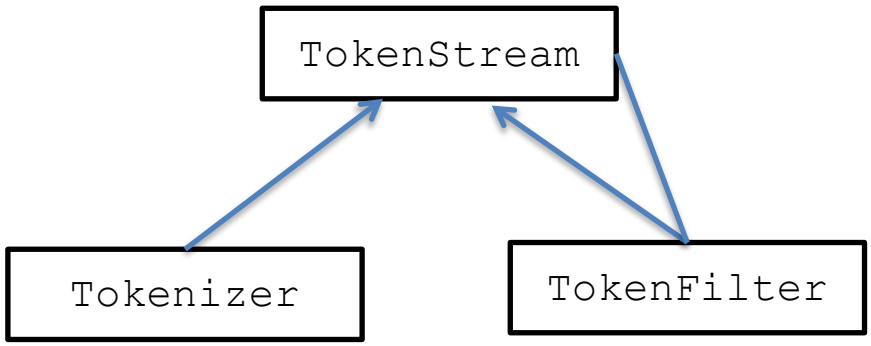
- `WhitespaceAnalyzer`
  - [XY&Z] [Corporation] [-] [xyz@example.com]

- `SimpleAnalyzer`
  - [xy] [z] [corporation] [xyz] [example] [com]

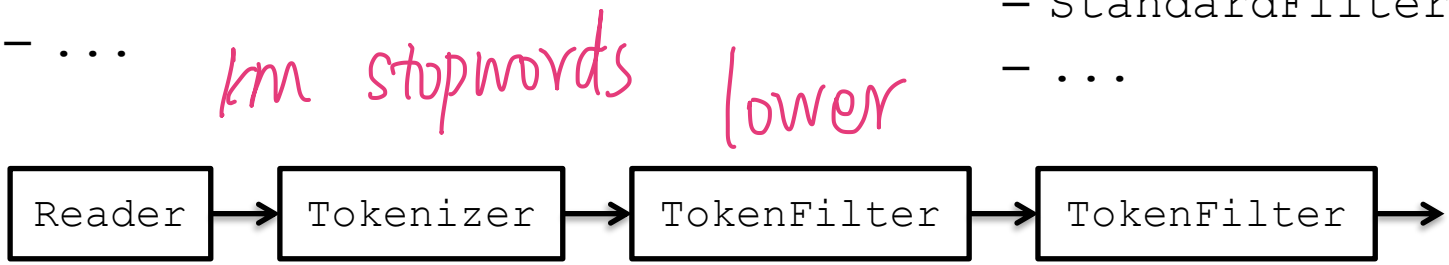
- `StopAnalyzer`
  - [xy] [z] [corporation] [xyz] [example] [com]

- StandardAnalyzer
  - [xy&z] [corporation] [xyz@example.com]

What's inside an Analyzer?



- Tokenizer
  - WhitespaceTokenizer
  - KeywordTokenizer
  - LetterTokenizer
  - StandardTokenizer
  - ...
- TokenFilter
  - LowerCaseFilter
  - StopFilter
  - PorterStemFilter
  - ASCIIFoldingFilter
  - StandardFilter
  - ...



http://lucene.apache.org/core/6_4_1/core/org/apache/lucene/analysis/Analyzer.html

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`
- You can explicitly invoke `forceMerge()` to merge segments

Lucene query syntax examples

Query expression	Document matches if…
java	Contains the term *java* in the default field
java junit java OR junit	Contains the term *java* or *junit* or both in the default field (*the default operator can be changed to* AND)
+java +junit java AND junit	Contains both *java* and *junit* in the default field
title:ant	Contains the term *ant* in the title field
title:extreme –subject:sports	Contains *extreme* in the title and not *sports* in subject
(agile OR extreme) AND java	Boolean expression matches
title:”junit in action”	Phrase matches in title
title:”junit action”~5	Proximity matches (within 5) in title
java*	Wildcard matches
java~	Fuzzy matches
lastmodified:[1/1/09 TO 12/31/09]	Range matches

Scoring

- Original scoring function uses basic tf-idf scoring with
  - Programmable boost values for certain fields in documents
  - Length normalization
  - Boosts for documents containing more of the query terms
  - http://lucene.apache.org/core/6_4_1/core/org/apache/lucene/search/similarities/TFIDFSimilarity.html

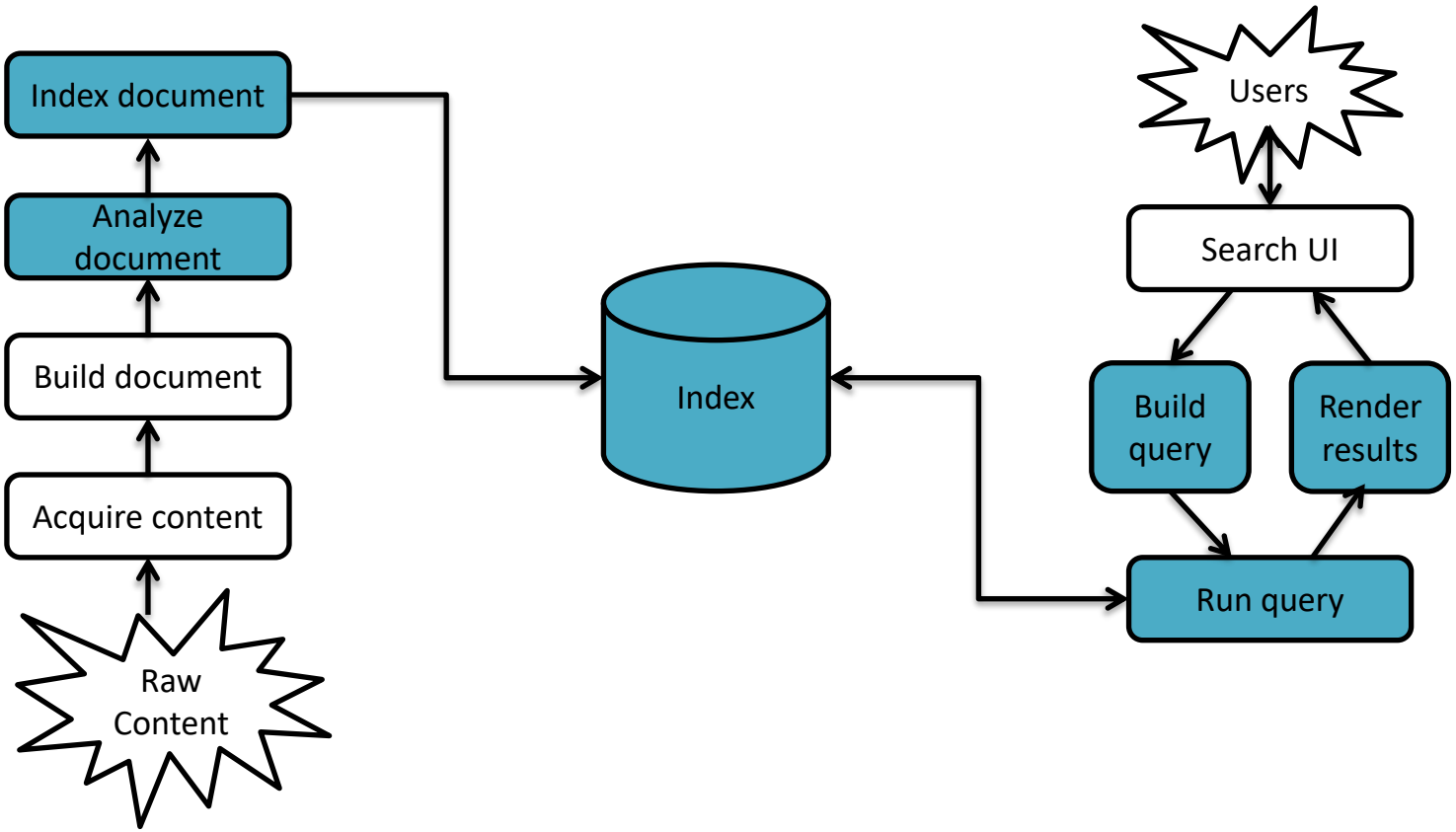
- `IndexSearcher` provides an `explain()` method that explains the scoring of a document

Lucene Scoring

- More than traditional tf.idf vector space model
  - BM25
  - DRF (Divergence From Randomness)
  - LM (Language Modeling similarities)
  - http://lucene.apache.org/core/6_4_1/core/org/apache/lucene/search/similarities/Similarity.html
  - http://lucene.apache.org/core/6_4_1/core/org/apache/lucene/search/similarities/SimilarityBase.html

```
indexSearcher.setSimilarity(new BM25Similarity());
BM25Similarity custom =new BM25Similarity(1.2, 0.75); // k1, b
indexSearcher.setSimilarity(custom);
```

Lucene in a search system



https://lucene.apache.org/core/8_4_1/index.html