Web Search Engine Using Java

COURSE: 60-654-01

INSTRUCTOR: DR LUIS RUEDA

Group Members:

- Dhawal Rank
- Mustafa Ali Misri
- Shehraj Gupta
- Siddharth Khobare

Web Search Engines

Google

YAHOO!®

AOL®™

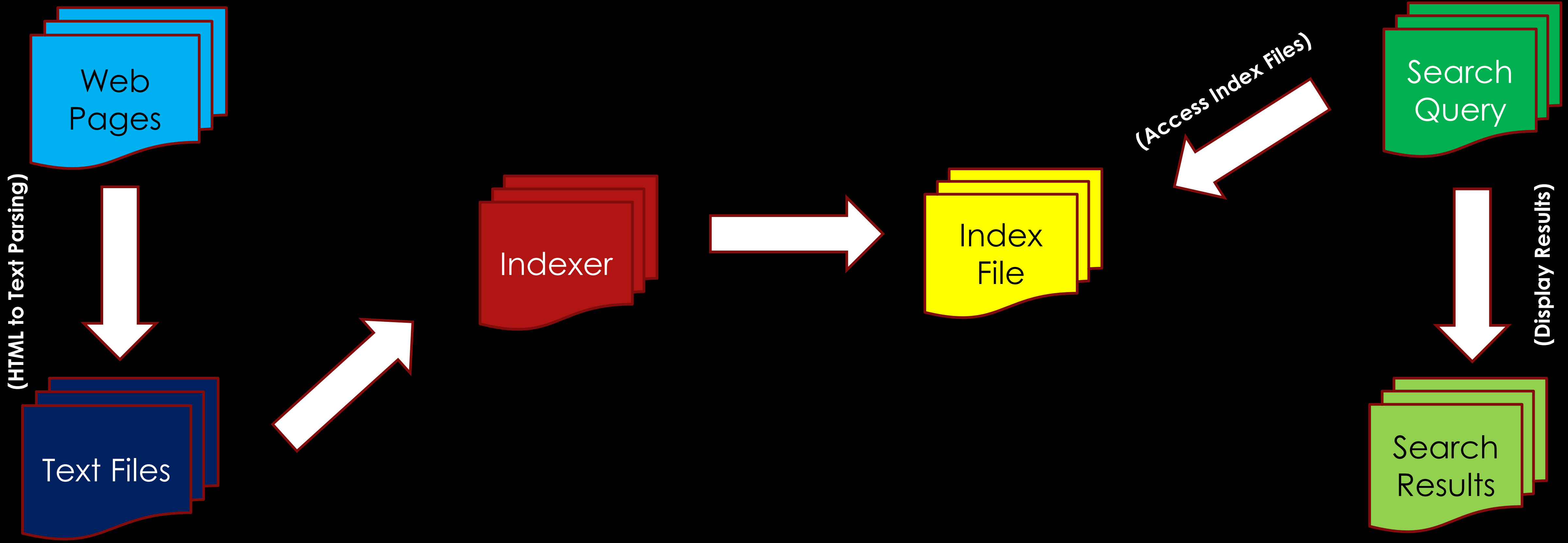
bing

Ask

Proposal:

As part of our term project, a search engine will be developed which will take search string and outputs the number of occurrence-sorted list of web pages in which particular keyword can be found.

Architecture



Lucene Scoring

- Lucene scoring is the reason why all of us prefer Apache Lucene.
- It is blazingly fast and it hides almost all of the complexity from the user.
- In Lucene, the objects we are scoring are Documents. A Document is a collection of Fields. Each Field has semantics about how it is created and stored (i.e. tokenized, untokenized, raw data, compressed, etc.)
- It is important to note that Lucene scoring works on Fields and then combines the results to return Documents.
- This is important because two Documents with the exact same content, but one having the content in two Fields and the other in one Field will return different scores for the same query due to length normalization.
- Scoring is very much dependent on the way documents are indexed, so it is important to understand indexing

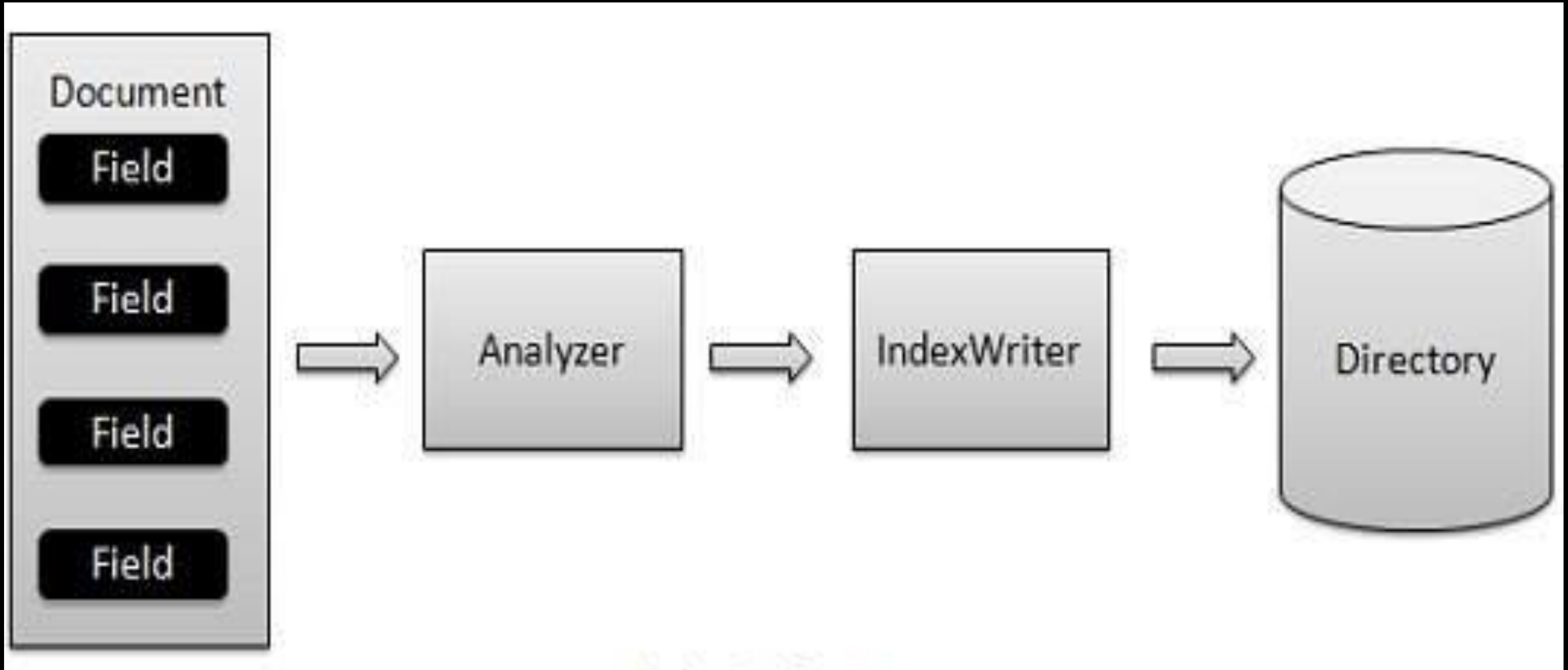
Indexer

- Apache Lucene core and libraries will be used used for Indexing files.
- Indexer collects the set of text files converted from HTML files and then indexes them.

Benefits:

- ranked searching — best results returned first.
- many powerful query types: phrase queries, wildcard queries, proximity queries, range queries.
- fielded searching: in our case (URL, contents, title).
- sorting by any field.

Indexing



Indexing Process

Indexing [Continued..]

- Analyzer Class: It creates an instance of the Standard Analyzer class, which is in charge of extracting tokens out of text to be indexed.
- Index Writer Class: It creates an instance of the Index Writer class, which is a key component in the indexing process. This class can create a new index or open an existing index and add documents to it.
- Here, the constructor accepts three parameters. The first parameter specifies the directory that stores the index files; the second parameter specifies the analyzer that will be used in the indexing process; the last parameter is a Boolean variable. If true, the class creates a new index; if false, it opens an existing index.
- We also have used optimize() method for the fastest available search.

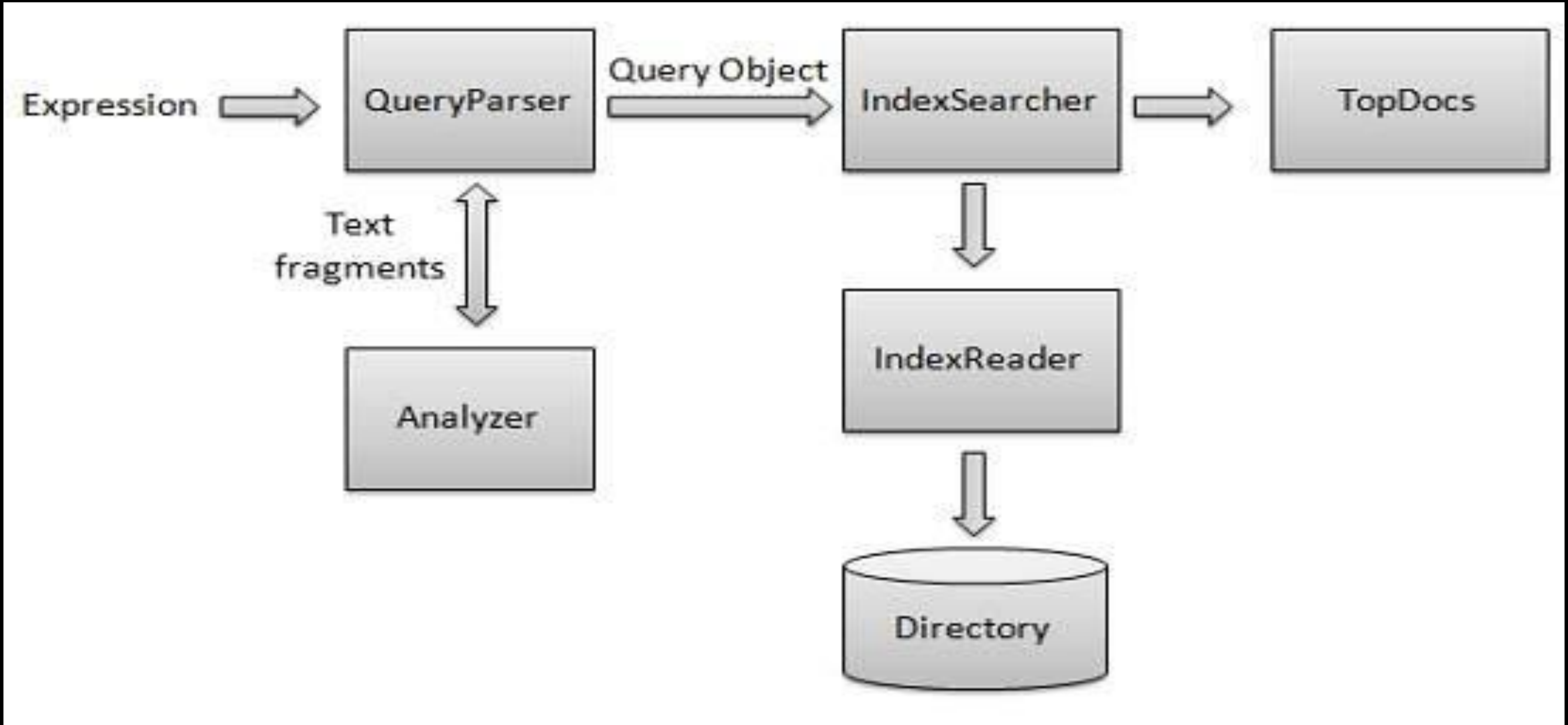
Indexing [Continued..]

```
Document document = new Document();

document.add(new Field(FIELD_PATH, path,Field.Store.YES, Field.Index.UN_TOKENIZED));
document.add(new Field(FIELD_CONTENTS, reader));
indexWriter.addDocument(document);
```

- The first line creates an instance of the Document class, which consists of a collection of fields.
- The second and third lines add two fields to the document. Each field contains a field name and the content. This example adds two fields named "content" and "path", which store the content and the path of the text file, respectively.
- The last line adds the prepared documents to the index.

Searching



Searching Process

Searching [Continued..]

- QueryParser class parses the user entered input into lucene understandable format query.
- Initialize the QueryParser object created with a standard analyzer having version information and index name on which this query is to run.
- IndexSearcher class acts as a core component which searches for indexes created during indexing process.
- IndexReader class is where we give Index directory as the input.

Searching [Continued..]

- Create a lucene directory which should point to location where indexes are to be stored.
- To start search, create a Query object by parsing search expression through QueryParser.
- Make search by calling IndexSearcher.search() method, where the parameter is the query.
- The best thing is that the results are already ranked by Apache Lucene, hiding the complexities and saving the time of users.

Conclusion

- In a nutshell, when lucene indexes a document it breaks it down into a number of terms.
- It then stores the terms in an index file where each term is associated with the documents that contains it.
- You could think of it as a bit like a hashtable. When a query is issued it is processed through the same analyzer that was used to build the index and then used to look up the matching terms in the index, that provides a list of documents that match the query.

Thanks for being good listeners.