# Very brief introduction to Lucene (4.10)

**Related Projects**

Apache Hadoop
Apache ManifoldCF
Apache Lucene.Net
Apache Lucy
Apache Mahout
Apache Nutch
Apache OpenNLP
Apache Tika
Apache Zookeeper

[From yonik@apache.org, 02 May 2007]

Amsterdam, Netherlands, slides:
http://www.apache.org/~yonik

# What is Lucene

- High performance, scalable, full-text search library
- Focus: Indexing + Searching Documents
- 100% Java, no dependencies, no config files
- No crawlers or document parsing
- Users: Wikipedia, Technorati, Monster.com, Nabble, TheServerSide, Akamai, SourceForge, Twitter, LinkedIn, Hi5, ...
- Applications: Eclipse, JIRA, Roller, OpenGrok, Nutch, Solr, and many commercial products.

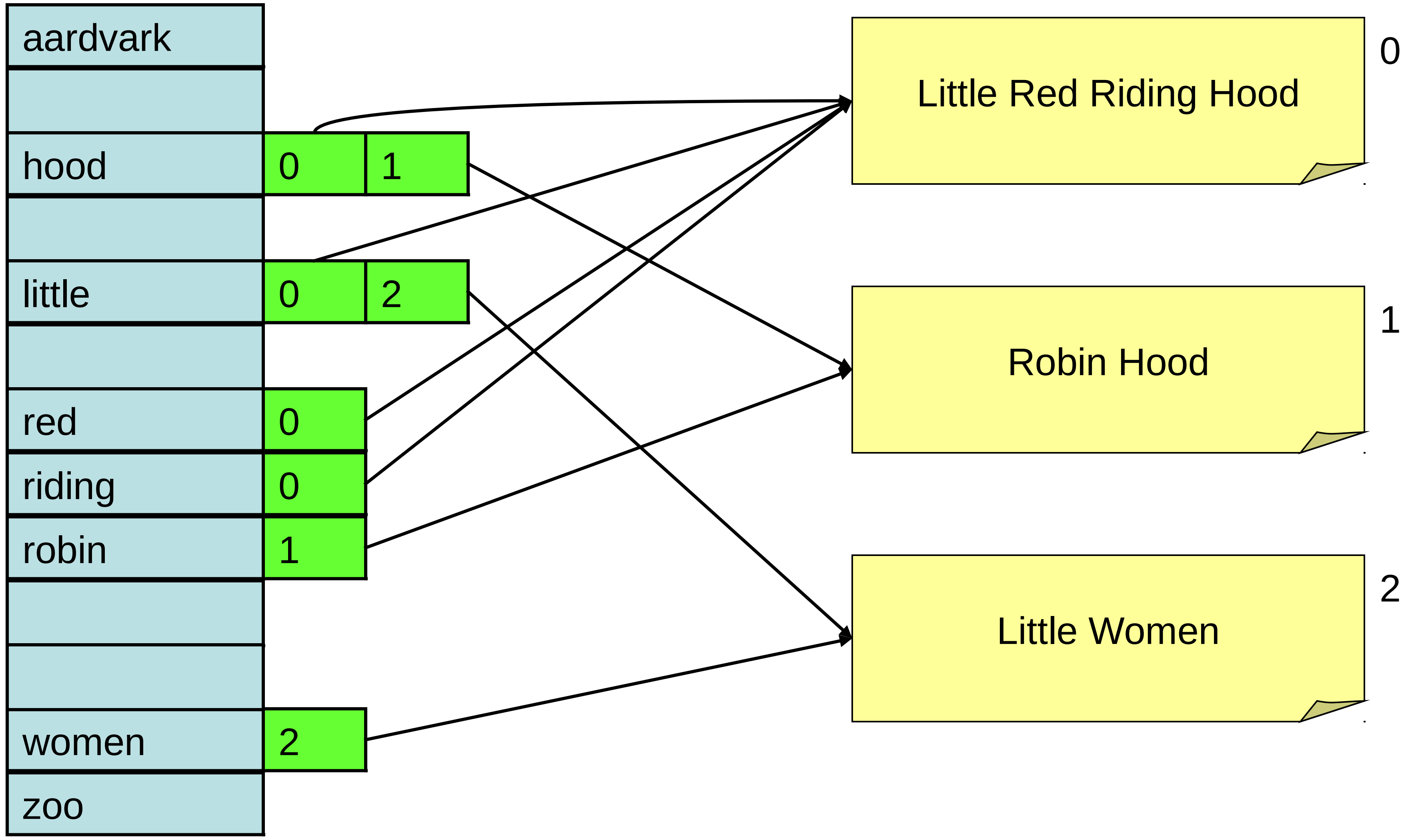
# Main feautures

**Scalable, High-Performance Indexing**
over 150GB/hour on modern hardware
small RAM requirements -- only 1MB heap
incremental indexing as fast as batch indexing
index size roughly 20-30% the size of text indexed

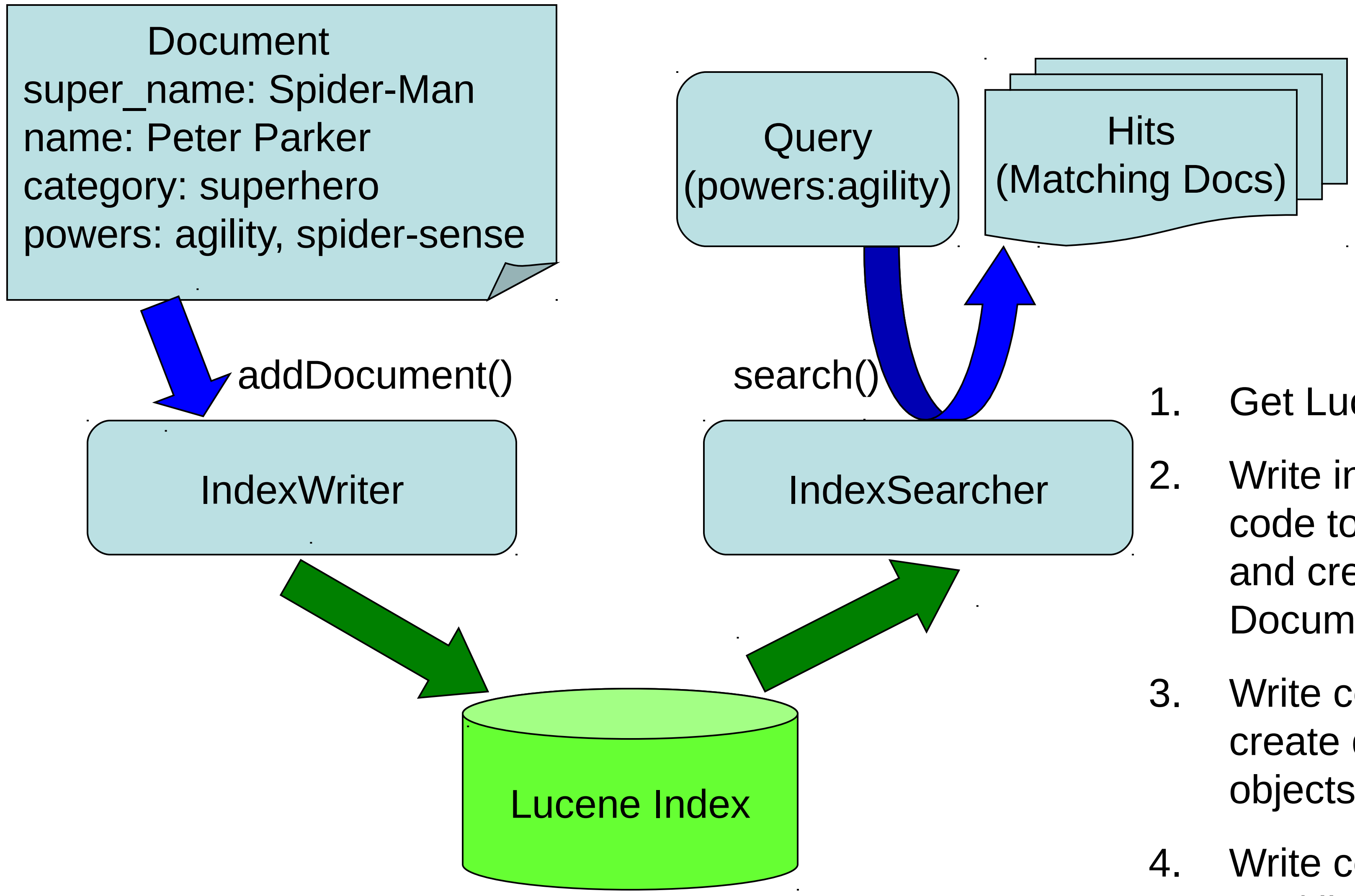
**Cross-Platform Solution**
Available as Open Source software under the Apache License which lets you use Lucene in both commercial and Open Source programs
100%-pure Java

**Implementations**
in other programming languages available that are index-compatible

# Inverted Index



# Basic Application



1. Get Lucene jar file
2. Write indexing code to get data and create Document objects
3. Write code to create query objects
4. Write code to use/display results

# Indexing Documents

```
IndexWriter writer = new IndexWriter(directory, analyzer,
    true);
Document doc = new Document();
doc.add(new Field(“super_name", “Sandman",
     Field.Store.YES, Field.Index.TOKENIZED));
doc.add(new Field(“name", “William Baker",
     Field.Store.YES, Field.Index.TOKENIZED));
doc.add(new Field(“name", “Flint Marko",
     Field.Store.YES, Field.Index.TOKENIZED));
// [...]
writer.addDocument(doc);
writer.close();
```

## Field Options

- **Indexed**
  - Necessary for searching or sorting
- **Tokenized**
  - Text analysis done before indexing
- **Stored**
  - You get these back on a search “hit”
- **Compressed**
- **Binary**
  - Currently for stored-only fields

# Searching an Index

```
IndexSearcher searcher = new IndexSearcher(directory);
QueryParser parser = new QueryParser("defaultField",
    analyzer);
Query query = parser.parse(“powers:agility");
Hits hits = searcher.search(query);
System.out.println(“matches:" + hits.length());
Document doc = hits.doc(0); // look at first match
System.out.println(“name=" + doc.get(“name"));
searcher.close();
```

# 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

## Score Boosting

Lucene allows influencing search results by "boosting" in more than one level:

- Document level boosting - while indexing - by calling **document.setBoost()** before a document is added to the index.
- Document's Field level boosting - while indexing - by calling **field.setBoost()** before adding a field to the document (and before adding the document to the index).
- Query level boosting - during search, by setting a boost on a query clause, calling **Query.setBoost()**.

# Lucene Conceptual Scoring Formula

score(q,d) =

**coord-factor**(q,d) . **query-boost**(q) .

(V(q) · V(d) / |V(q)| ). **doc-len-norm**(d) . **doc-boost**(d)

http://lucene.apache.org/core/4_1_0/core/org/apache/lucene/search/similarities/TFIDFSimilarity.html

## Query Construction

**Lucene QueryParser**

- Example: *queryParser.parse(“name:Spider-Man");*
- Easy entered queries, debugging, IPC
- Does text analysis and constructs appropriate queries
- Not all query types supported

**Programmatic query construction**

- Example:

*new TermQuery(new Term(“name”,”Spider-Man”))*

- Explicit
- Does not do text analysis for you

## Query Examples

1. **justice league**
   - EQUIV: justice OR league
   - QueryParser default is “optional”
2. **+justice +league –name:aquaman**
   - EQUIV: justice AND league NOT name:aquaman
3. **“justice league” –name:aquaman**
4. **title:spiderman^10 description:spiderman**
5. **description:“spiderman movie”~10**

# Query Examples 2

1. **releaseDate:[2000 TO 2007]**
   - Range search: lexicographic ordering, so beware of numbers
2. **Wildcard searches**: sup?r, su*r, super*
3. **spider~**
   - Fuzzy search: Levenshtein distance
   - Optional minimum similarity: spider~0.7
4. (Superman AND “Lex Luthor”) OR (+Batman +Joker)

# Deleting Documents

Deleting with IndexWriter

- **deleteDocuments**(Term term): *Deletes the document(s) containing term.*
- **updateDocument**(Term term, Iterable<? extends IndexableField> doc): *Updates a document by first deleting the document(s) containing term and then adding the new document.*
- Deleting does not immediately reclaim space

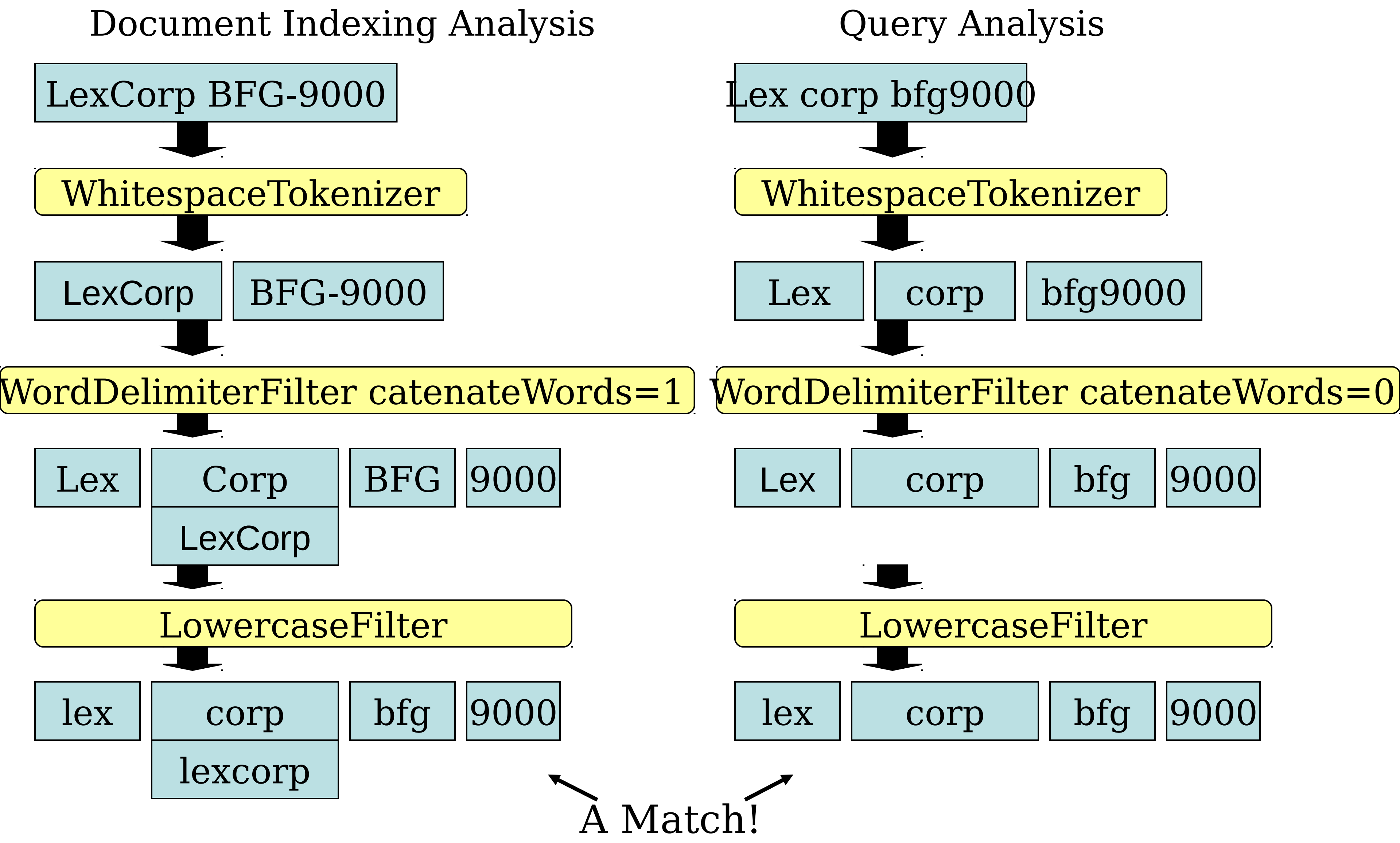
# Performance

- **Indexing Performance**
  - Index documents in batches
  - Raise merge factor
  - Raise maxBufferedDocs
- **Searching Performance**
  - Reuse IndexSearcher
  - Optimize
  - Use cached filters (see QueryFilter)

    ‘+superhero +lang:english’

    ‘superhero’ filtered by ‘lang:english’

# Analysis & Search Relevancy



## Tokenizers

Tokenizers break field text into tokens

- **StandardTokenizer**
  - source string: “full-text lucene.apache.org”
  - “full”   “text”   “lucene.apache.org”
- **WhitespaceTokenizer**
  - “full-text”   “lucene.apache.org”
- **LetterTokenizer**
  - “full”   “text”   “lucene”   “apache”   “org”

## TokenFilters

- LowerCaseFilter
- StopFilter
- ISOLatin1AccentFilter
- SnowballFilter
  - stemming: reducing words to root form
  - rides, ride, riding => ride
  - country, countries => countri
- contrib/analyzers for other languages
- SynonymFilter (from Solr)
- WordDelimiterFilter (from Solr)

## Analyzers

```
class MyAnalyzer extends Analyzer {
  private Set myStopSet =
    StopFilter.makeStopSet(StopAnalyzer.ENGLISH_STOP_WORDS);

  public TokenStream tokenStream(String fieldname, Reader reader) {
    TokenStream ts = new StandardTokenizer(reader);
    ts = new StandardFilter(ts);
    ts = new LowerCaseFilter(ts);
    ts = new StopFilter(ts, myStopSet);
    return ts;
  }
}
```

## Analysis Tips

- Use *PerFieldAnalyzerWrapper*

  This analyzer is used to facilitate scenarios where different fields require different analysis techniques.
- Add same field more than once, analyze differently
  - Boost exact case matches
  - Boost exact tense matches
  - Query with or without synonyms
  - Soundex for sounds-like queries
- Use *explain(Query q, int docid)* for debugging

# Nutch

- Open source web search application
- Crawlers
- Link-graph database
- Document parsers (HTML, word, pdf, etc)
- Language + charset detection
- Utilizes Hadoop (DFS + MapReduce) for massive scalability

## Solr

SolrTM is the popular, blazing fast open source enterprise search platform from the Apache LuceneTM project.

– REST XML/HTTP, JSON APIs
– Faceted search
– Flexible Data Schema
– Hit Highlighting
– Configurable Advanced Caching
– Replication
– Web admin interface

# Running Toy Application

- http://www.lucenetutorial.com/code/TextFileIndexer.java
- LUCENE 4.5 http://mirror.sdunix.com/apache/lucene/java/4.5.0/
- Importing external JARs
  - ../../lucene-4.5.0/core/lucene-core-4.5.0.jar
  - ../../lucene-4.5.0/analysis/common/lucene-analyzers-common-4.5.0.jar
  - ../../lucene-4.5.0/queryparser/lucene-queryparser-4.5.0.jar