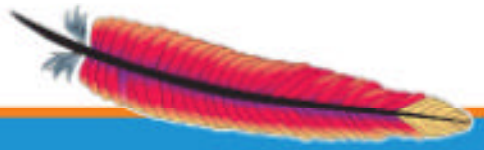
Full-Text Search with Lucene

Yonik Seeley
yonik@apache.org

02 May 2007
Amsterdam, Netherlands

slides: http://www.apache.org/~yonik

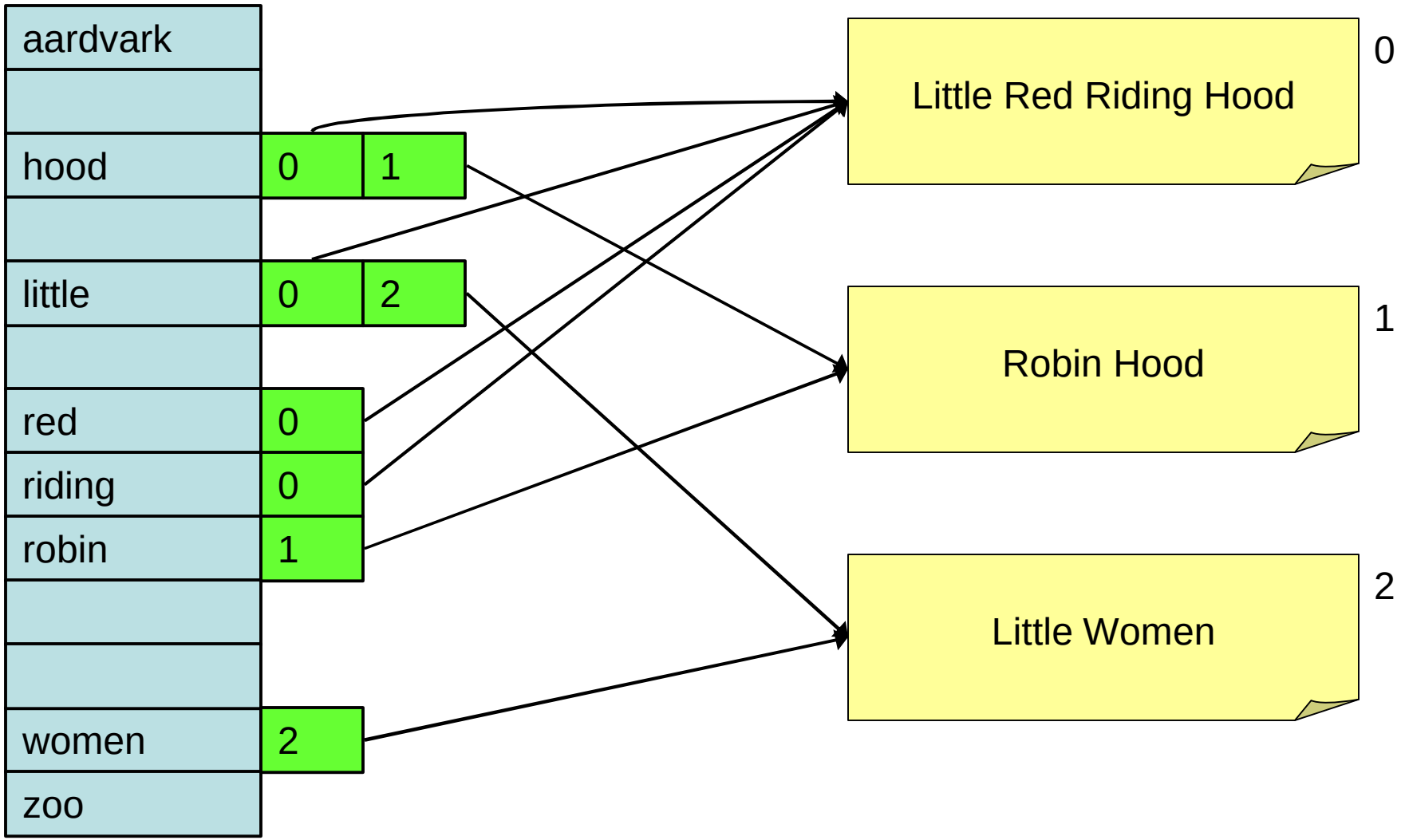


ApacheCon
EU 2007

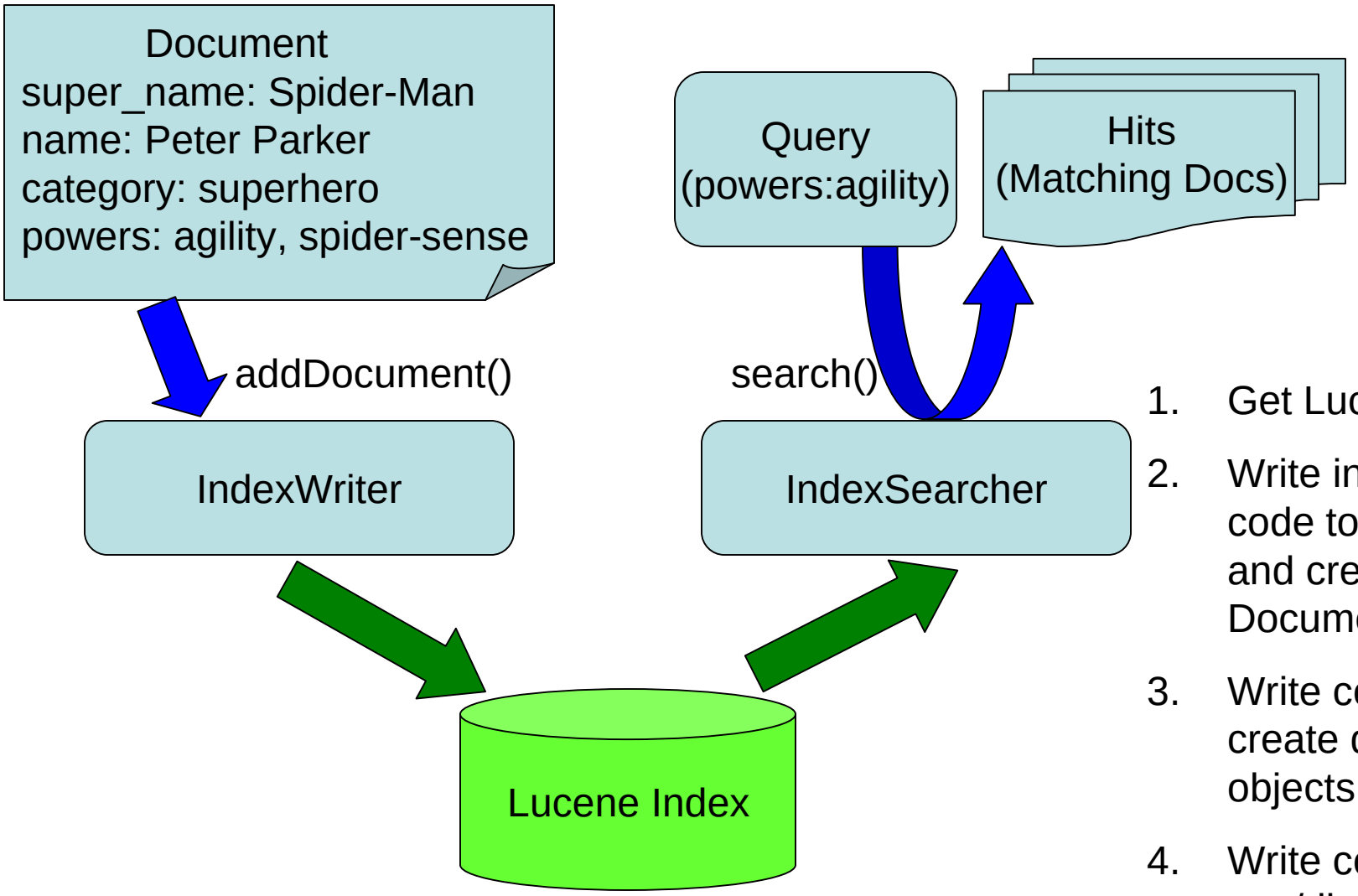
What is Lucene

- 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
- Applications: Eclipse, JIRA, Roller, OpenGrok, Nutch, Solr, many commercial products

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
- document boost
- query clause boost

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

Query Construction

Lucene QueryParser

- Example: queryParser.parse(“name:Spider-Man");
- good human 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, no escaping necessary
- 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

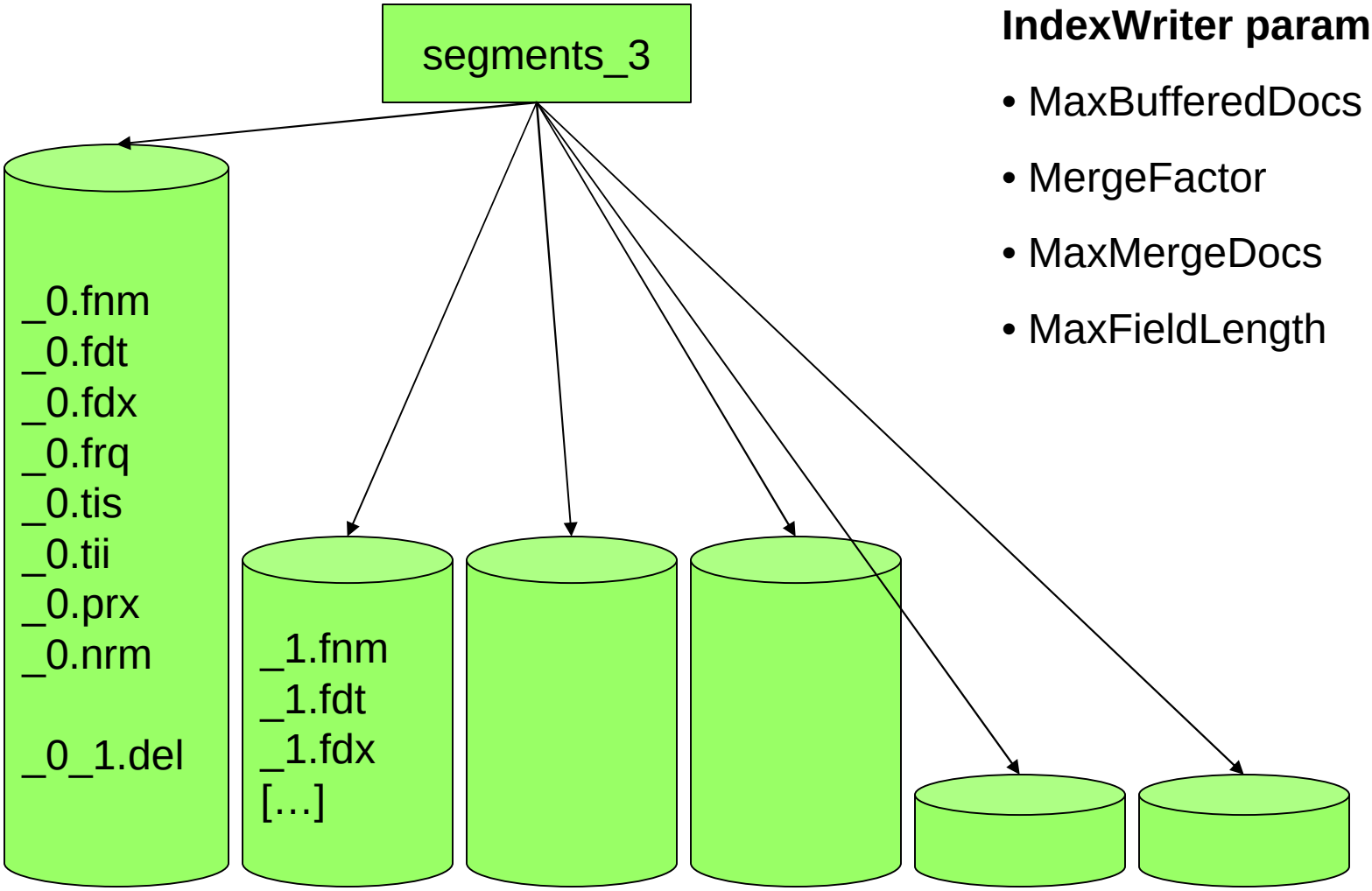
Query Examples2

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. *:*
5. (Superman AND “Lex Luthor”) OR (+Batman +Joker)

Deleting Documents

- IndexReader.deleteDocument(int id)
  - exclusive with IndexWriter
  - powerful
- Deleting with IndexWriter
  - deleteDocuments(Term t)
  - updateDocument(Term t, Document d)
- Deleting does not immediately reclaim space

Index Structure



IndexWriter params

- MaxBufferedDocs
- MergeFactor
- MaxMergeDocs
- MaxFieldLength

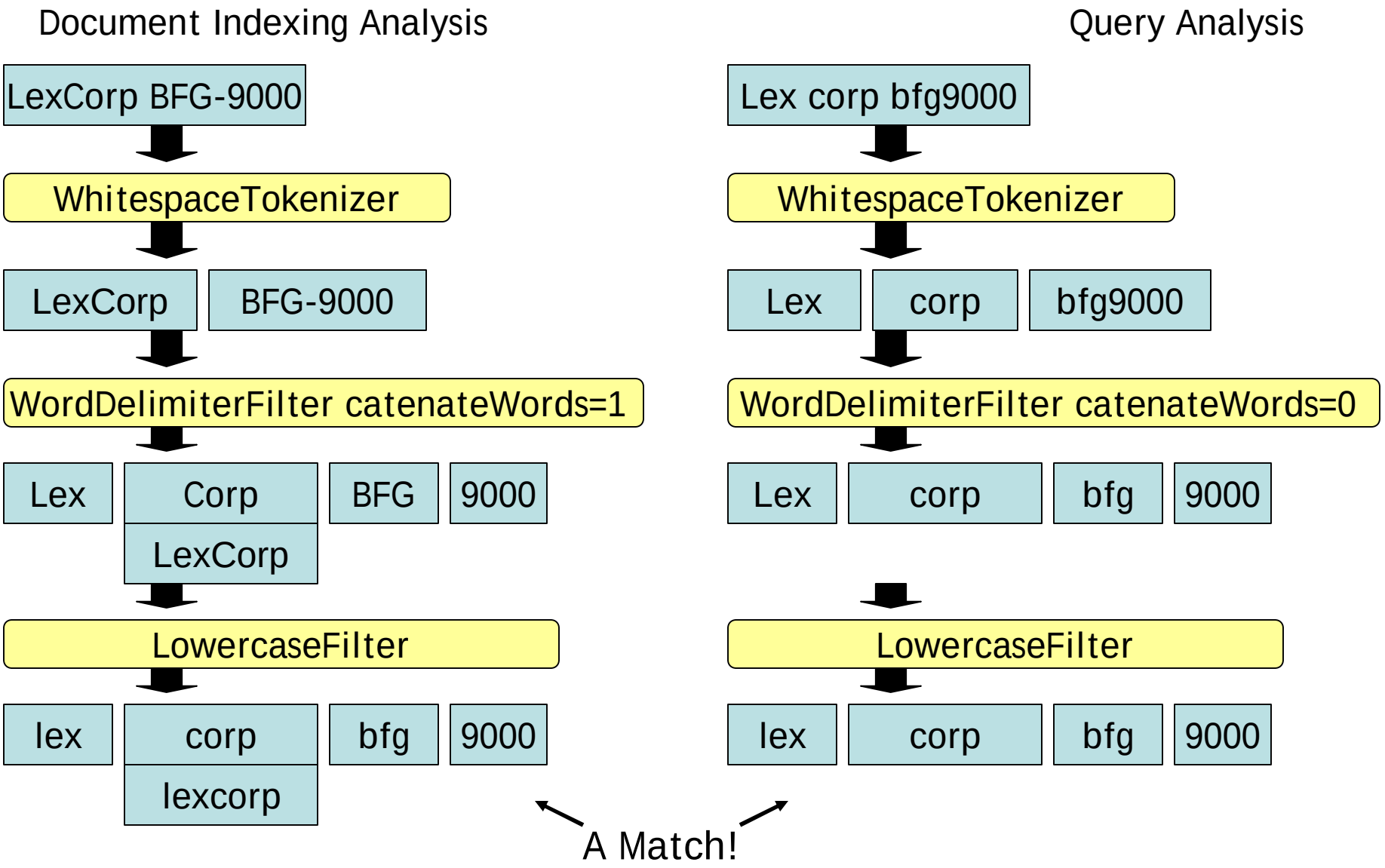
Performance

- Indexing Performance
  - Index documents in batches
  - Raise merge factor
  - Raise maxBufferedDocs
- Searching Performance
  - Reuse IndexSearcher
  - Lower merge factor
  - 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
- Use NumberTools for numbers
- 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

- REST XML/HTTP, JSON APIs
- Faceted search
- Flexible Data Schema
- Hit Highlighting
- Configurable Advanced Caching
- Replication
- Web admin interface
- Solr Flare: Ruby on Rails user interface

Het Eind

java-user-subscribe@lucene.apache.org

nutch-user-subscribe@lucene.apache.org

solr-user-subscribe@lucene.apache.org

Other Lucene Presentations

- Advanced Lucene (stay right here!)
- Beyond full-text searches with Solr and Lucene (Thursday 14:00)
- Introduction to Hadoop (Thursday 15:00)

This presentation: http://www.apache.org/~yonik