Open-Source Search Engines and Lucene/Solr

UCSB 293S, Fall 2020. Tao Yang

Slides are based on Y. Seeley,
S. Das, C. Hostetter

Open Source Search Engines

- **Why?**
  - Low cost: No licensing fees
  - Source code available for customization
  - Good for modest or even large data sizes
- **Challenges:**
  - Performance, Scalability
  - Maintenance

Open Source Search Engines: Examples

- **Lucene**
  - A full-text search library with core indexing and search services based on Java
- **Solr**
  - based on the Lucene Java search library with XML/HTTP APIs
  - caching, replication, and a web administration interface.
- **ElasticSearch**
- **Lemur/Indri**
  - C++ search engine from CMU/U. Mass

Lucene

- **Developed by Doug Cutting initially**
  - – Java-based. Created in 1999, Donated to Apache in 2001
- **Features**
  - No crawler, No document parsing, No "PageRank"

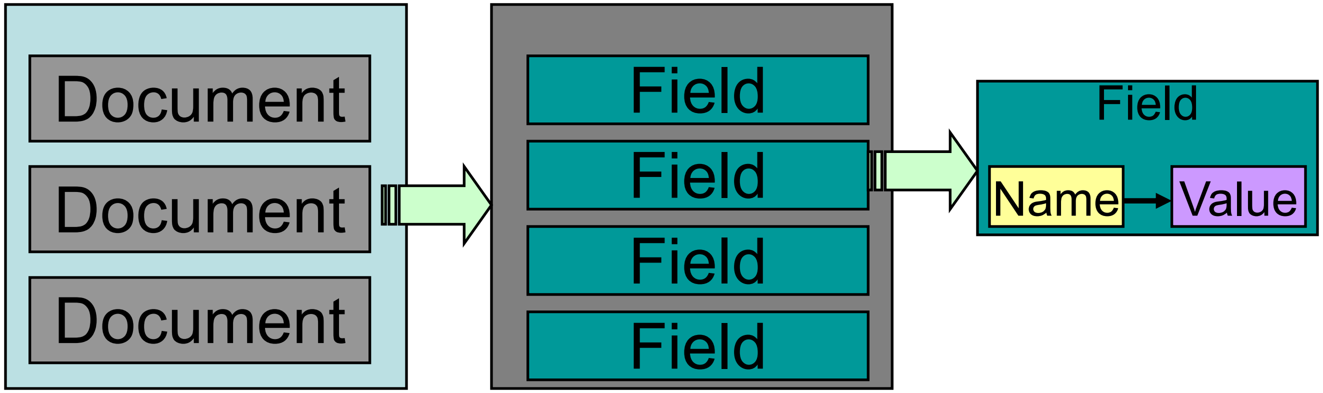
- **Powered by Lucene**
  - – IBM Omnifind Y! Edition, Technorati
  - – Wikipedia, Internet Archive, LinkedIn, monster.com
- **Add documents to an index via IndexWriter**
  - A document is a collection of fields
  - Flexible text analysis – tokenizers, filters

- **Search for documents via IndexSearcher**

  Hits = search(Query,Filter,Sort,topN)

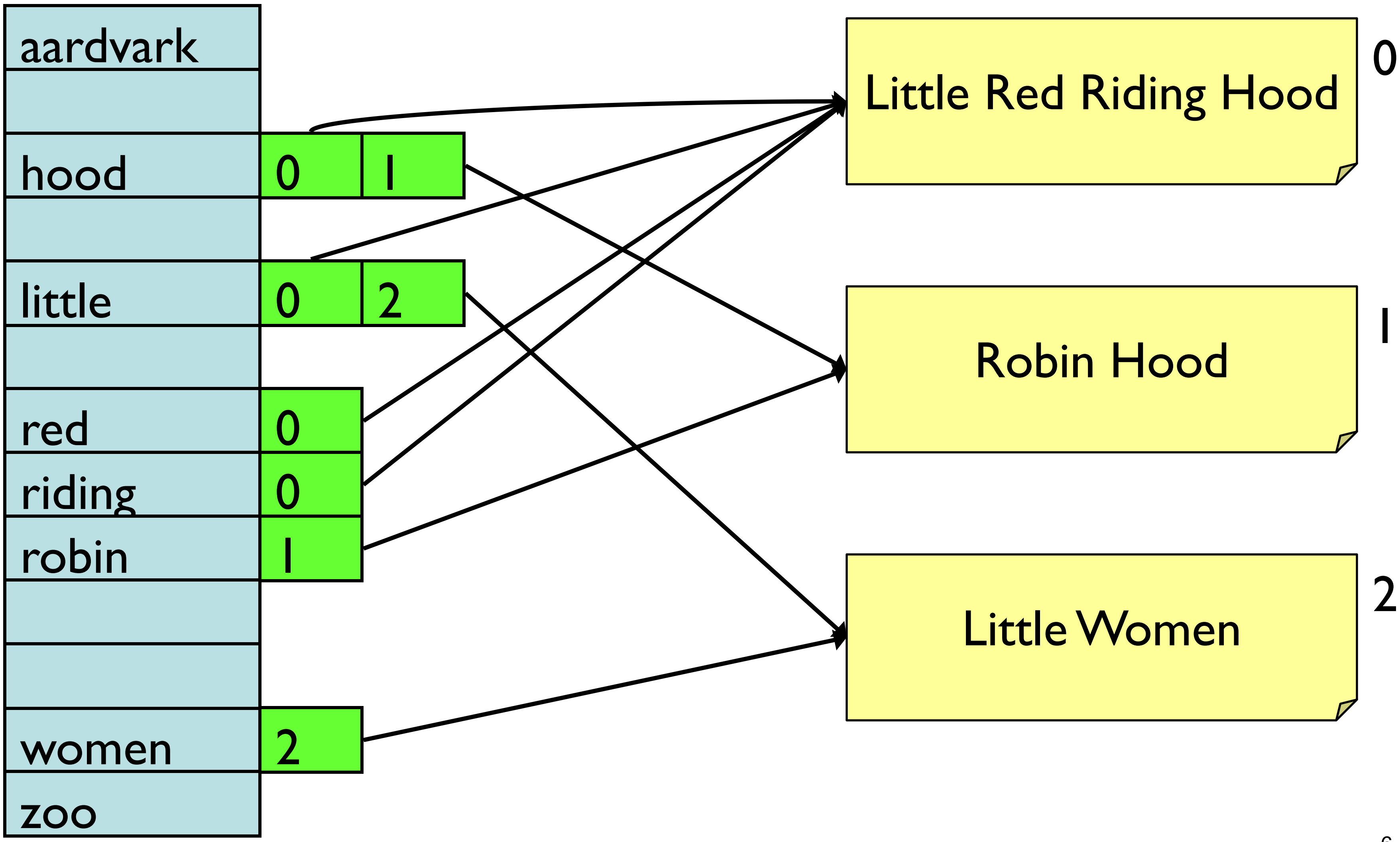
- **Ranking based on tf * idf similarity with normalization**

Lucene's input content for indexing



- **Logical structure**
  - Documents are a collection of fields
    - Stored – Stored verbatim for retrieval with results
    - Indexed – Tokenized and made searchable
  - Indexed terms stored in inverted index
- **Physical structure of inverted index**
  - Multiple documents stored in segments
- **IndexWriter is interface object for entire index**

Example of Inverted Indexing



DESKTOPS

Faceted Search/Browsing Example

You found 1045 items for System type: Budget desktop system

Too few results? Click a link above to remove that filter, or remove all filters.

Find by price
- Less than $400 (76)
- $400 to $699 (337)
- $700 to $999 (468)
- $1000 to $1299 (5)

Find by manufacturer
- Dell, Inc. (43)
- Lenovo (490)
- HP (342)
- Acer America Corp. (28)
- Cyberpower Inc (22)
- See all manufacturers

Find by processor manufacturer
- Intel (804)
- AMD (122)
- Motorola (1)

Or find by
- Clock speed
- Graphics processor
- RAM installed
- Hard drive size
- OS provided
- See all

Sort by: Product name | Lowest price | Editors' rating | **Review date**

Check products to Compare

Reviewed on 05/05/2006

Dell Dimension B110 Desktop Computer for Home (Cel-D 2.53GHz/160GB/512MB)

Dell's entry-level Dimension B110 series features aging technology and a dated design, but its members will suffice as second PCs for basic tasks.

Specs: Celeron D (2.53 GHz), 512 MB, 160 GB, 15 in, Microsoft Windows XP Home Edition

Add to my products **New!** What is this?

$479

at 1 store

Check prices

COMPARE

Dell Dimension B110 Desktop Computer for Home (Cel-D 2.53GHz/80GB/256MB)

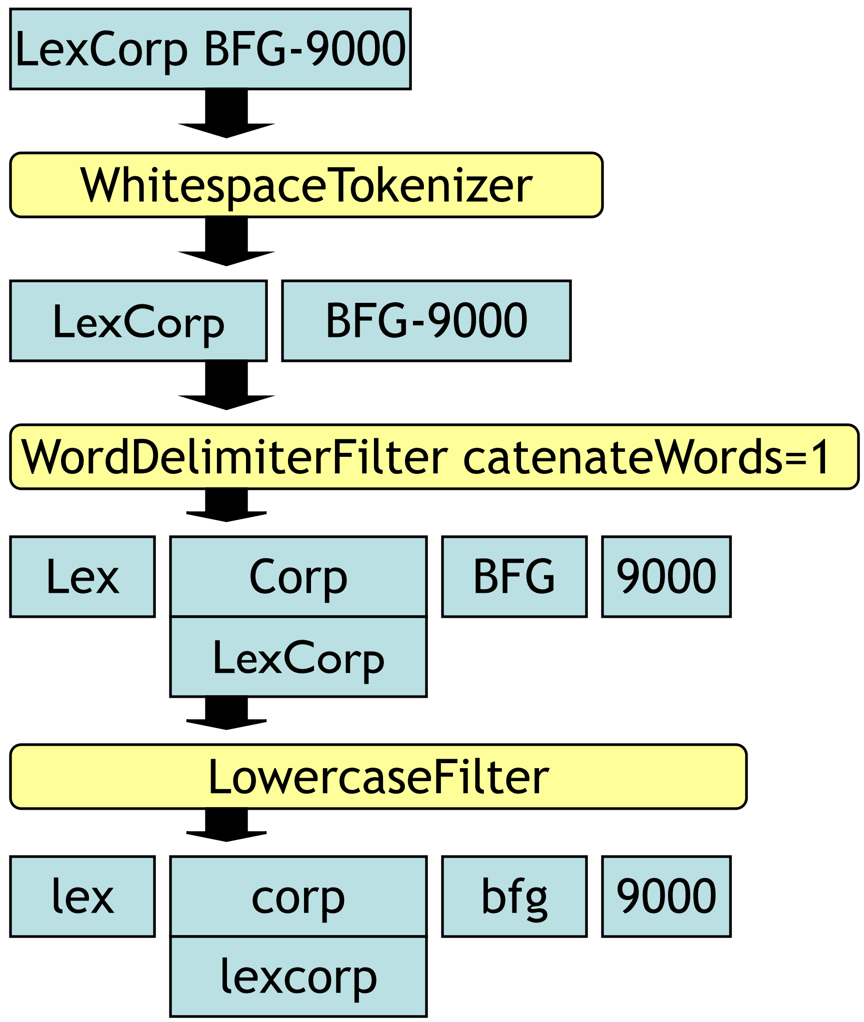
Dell's entry-level Dimension B110 series features aging

$349

at 1 store

COMPARE

Indexing Flow

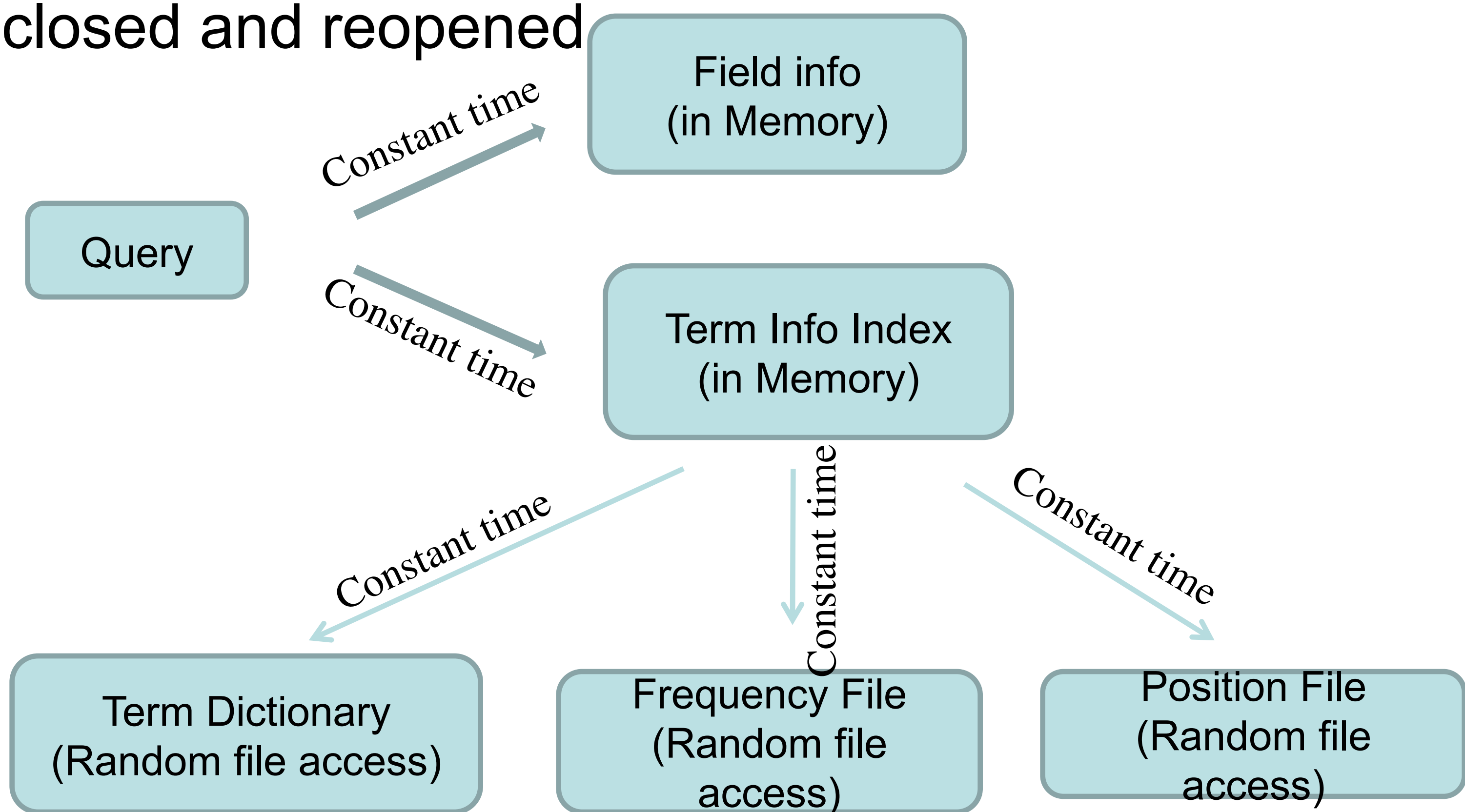


Analyzers specify how the text in a field is to be indexed

- Options in Lucene
  - WhitespaceAnalyzer
    - divides text at whitespace
  - SimpleAnalyzer
    - divides text at non-letters
    - convert to lower case
  - StopAnalyzer
    - SimpleAnalyzer
    - removes stop words
  - StandardAnalyzer
    - good for most European Languages
    - removes stop words
    - convert to lower case
  - Create you own Analyzers

Query Processing and Index Files Involved

- **Concurrent search query handling:**
  - Multiple searchers at once. Thread safe
- **Additions or deletions to index are not reflected in already open searchers**
  - Must be closed and reopened



Lucene Index Files: Field infos file (.fnm)

Enumerate a list of fields

Format:	FieldsCount, <FieldName, FieldBits>
FieldsCount	the number of fields in the index
FieldName	the name of the field in a string
FieldBits	a byte and an int where the lowest bit of the byte shows whether the field is indexed, and the int is the id of the term

Example: A data set with 1 field called “content”

1, <content, 0x01>

http://lucene.apache.org/core/3_6_2/fileformats.html

Lucene Index Files: Term Dictionary file (.tis)

List all terms

Format:	TermCount, TermInfos	
	TermInfos	<Term, DocFreq>
	Term	<PrefixLength, Suffix, FieldNum>
This file is sorted by Term. Terms are ordered first lexicographically by the term's field name, and within that lexicographically by the term's text		
TermCount	the number of terms in the documents	
Term	Term text prefixes are shared. The PrefixLength is the number of initial characters from the previous term which must be pre-pended to a term's suffix in order to form the term's text. Thus, if the previous term's text was "bone" and the term is "boy", the PrefixLength is two and the suffix is "y".	
FieldNumber	the term's field, whose name is stored in the .fnm file	

Example with 4 terms:

4,<<0,football,1>,2> <<0,penn,1>, 1> <<1,layers,1>,1> <<0,state,1>,2>

<1, layers, 1> means Prefix = ”p”. Suffix=“layers”, Field Num=1. Thus term=“players”

Document Frequency can be obtained from this file.

Lucene Index Files: Term Info index (.tii)

Format:	IndexTermCount, IndexInterval, TermIndices	
	TermIndices	<TermInfo, IndexDelta>
This contains every IndexInterval th entry from the .tis file, along with its location in the "tis" file. This is designed to be read entirely into memory and used to provide random access to the "tis" file.		
IndexDelta	determines the position of this term's TermInfo within the .tis file. In particular, it is the difference between the position of this term's entry in that file and the position of the previous term's entry.	

Example with 4 terms

4,<football,1> <penn,3><layers,2> <state,1>

Term info positions in .tis file: 1, 4, 5, 6

Lucene Index Files: Frequency file (.frq)

Format:	<TermFreqs>	
	TermFreqs	TermFreq
	TermFreq	DocDelta, Freq?
TermFreqs are ordered by term (the term is implicit, from the .tis file). TermFreq entries are ordered by increasing document number.		
DocDelta	determines both the document number and the frequency. In particular, DocDelta/2 is the difference between this document number and the previous document number (or zero when this is the first document in a TermFreqs). When DocDelta is odd, the frequency is one. When DocDelta is even, the frequency is read as the next Int. For example, the TermFreqs for a term which occurs once in document seven and three times in document eleven would be the following sequence of Ints: 15, 8, 3	

Example with 2 documents: Doc 7 with freq 1 and Doc 11 with Freq 3

[7, 1] [11, 3] → [DocIDDelta = 7, Freq = 1] [DocIDDelta = 4 (11-7), Freq = 3]
→(7 << 1) | 1 = 15 and (4 << 1) | 0 = 8
→[DocDelta = 15] [DocDelta = 8, Freq = 3]
http://hackerlabs.org/blog/2011/10/01/hacking-lucene-the-index-format/

Lucene Index Files: Position file (.prx)

Format:	<TermPositions>	
	TermPositions	<Positions>
	Positions	<PositionDelta >
TermPositions are ordered by term (the term is implicit, from the .tis file). Positions entries are ordered by increasing document number (the document number is implicit from the .frq file).		
PositionDelta	the difference between the position of the current occurrence in the document and the previous occurrence (or zero, if this is the first occurrence in this document). For example, the TermPositions for a term which occurs as the fourth term in one document, and as the fifth and ninth term in a subsequent document, would be the following sequence of Ints: 4, 5, 4	

Example of a term that appears in two documents.
Positions in these two docs:
[4] [5 9] → [4] [5 4] → 4 5 4

Query Syntax and Examples

- **Terms with fields and phrases**
  - Title:right and text: go
  - Title:right and go   (go appears in default field “text”)
  - Title: “the right way” and go
- **Proximity**
  - “quick fox”~4
- **Wildcard**
  - pla?e (plate or place or plane)
  - practic* (practice or practical or practically)
- **Fuzzy (edit distance as similarity)**
  - planting~0.75 (granting or planning)
  - roam~ (default is 0.5)

Query Syntax and Examples

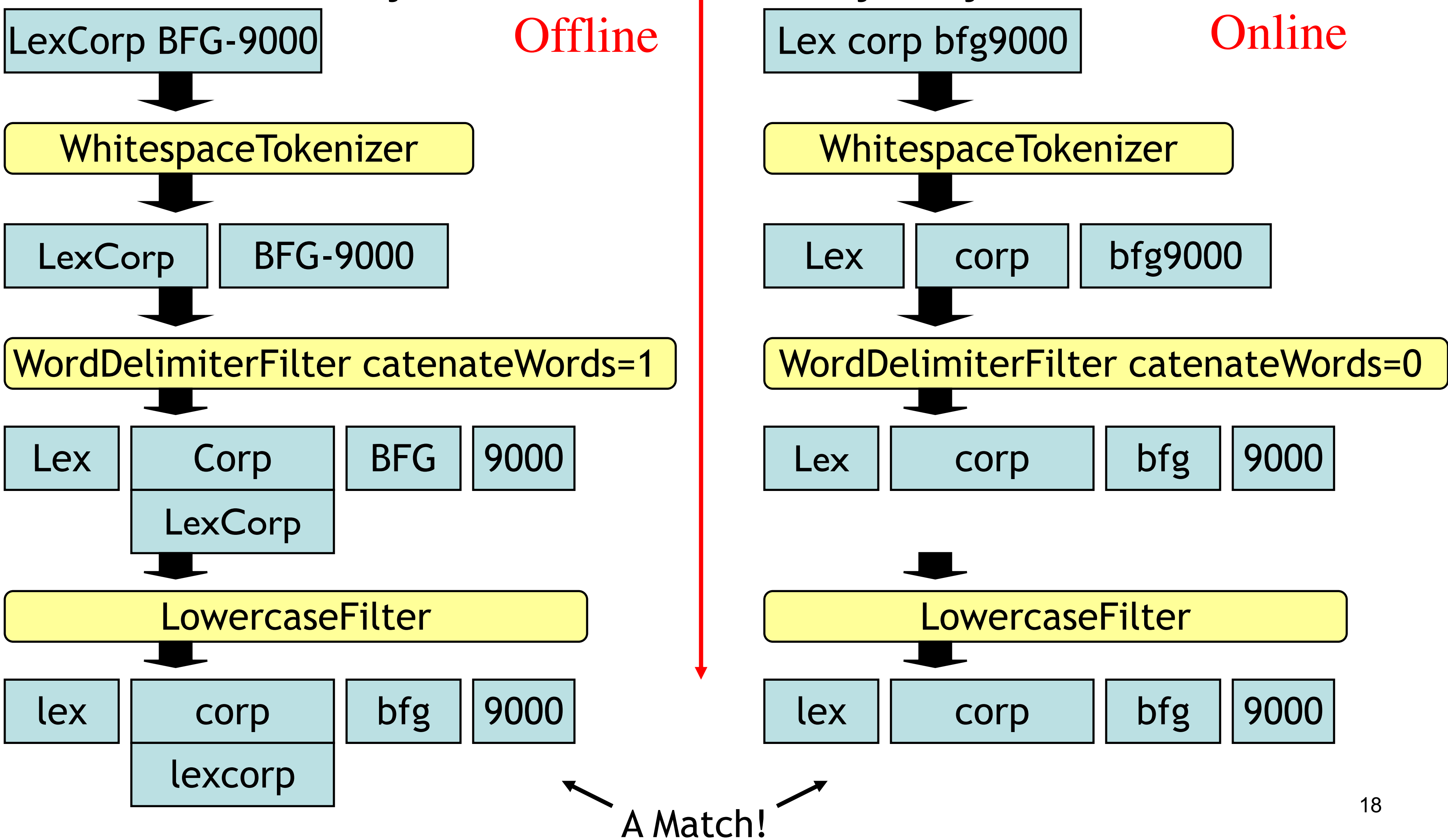
- **Range**
  - date:[05072007 TO 05232007] (inclusive)
  - author: {king TO mason} (exclusive)
- **Ranking weight boosting ^**
  - title:“Bell” author:“Hemmingway”^3.0
  - Default boost value 1. May be <1 (e.g 0.2)

- **Boolean operators: AND, "+", OR, NOT and "-"**
  - “Linux OS” AND system
  - Linux OR system, Linux system
  - +Linux system
  - +Linux –system
- **Grouping**
  - Title: (+linux +”operating system”)
- **http://lucene.apache.org/core/2_9_4/queryparsersyntax.html**

Consistency in Indexing and Searching

- **Document analysis**

Query analysis



Ranking Factors for Lucene's Scoring

- **tf** = term frequency in document = measure of how often a term appears in the document
- **idf** = inverse document frequency = measure of how often the term appears across the index
- **coord** = number of terms in the query that were found in the document
- **Document length Norm** = measure of the importance of a term according to the total number of terms in the field
- **Query Norm** = normalization factor so that queries can be compared
- **boost (index)** = boost of the field at index-time
- **boost (query)** = boost of the field at query-time
- http://lucene.apache.org/core/3_6_2/scoring.html

http://www.lucenetutorial.com/advanced-topics/scoring.html

Scoring Function is specified in schema.xml

- **Similarity**

  $$score(Q,D) = coord(Q,D) \cdot queryNorm(Q) \cdot \sum_{t \text{ in } Q} (tf(t \text{ in } D) \cdot idf(t)^2 \cdot t.getBoost() \cdot norm(D))$$

- **term-based factors**
  - *tf(t in D)* : term frequency of term t in document d
    - Sqrt of raw term frequency
  - *idf(t)*: inverse document frequency of term t in the entire corpus
    - $Ln[N_{docs}/(docFreq +1)] +1$

Default Scoring Functions for query Q in matching document D

- ***coord(Q,D)*** *= overlap between Q and D / maximum overlap*
  Maximum overlap is the maximum possible length of overlap between Q and D

- ***queryNorm(Q)*** *= 1/sum of square weight*$^{1/2}$
  sum of square weight $= q.getBoost()^2 \cdot \sum_{t \in Q} (idf(t) \cdot t.getBoost())^2$

  If $t.getBoost() = 1$, and $q.getBoost() = 1$
  Then, *sum of square weight* $= \sum_{t \in Q} (idf(t))^2$
  thus, $queryNorm(Q) = 1/(\sum_{t \in Q} (idf(t))^2)^{1/2}$

- ***norm(D)*** *= 1/number of terms*$^{1/2}$ (This is the normalization by the total number of terms in a document. Number of terms is the total number of terms appeared in a document *D*.)

Example: $score(Q,D) = coord(Q,D) \cdot queryNorm(Q) \cdot \sum_{t \text{ in } Q} (tf(t \text{ in } D) \cdot idf(t)^2 \cdot t.getBoost() \cdot norm(D))$

- **D1: hello, please say hello to him.**
- **D2: say goodbye**
- **Q: you say hello**
  - *coord(Q, D) = overlap between Q and D / maximum overlap*
    - *coord(Q, D1) = 2/3, coord(Q, D2) = 1/2,*
  - *queryNorm(Q) = 1/sum of square weight*$^{1/2}$
    - *sum of square weight* $= q.getBoost()^2 \cdot \sum t \text{ in } Q\ (idf(t) \cdot t.getBoost())^2$
    - *t.getBoost() = 1, q.getBoost() = 1*
    - *sum of square weight* $= \sum t \text{ in } Q\ (idf(t))^2$
    - $queryNorm(Q) = 1/(0.5945^2+1^2)^{1/2} = 0.8596$
  - $tf(t \text{ in } d) = frequency^{1/2}$
    - tf(you,D1) = 0, tf(say,D1) = 1, tf(hello,D1) = $2^{1/2}$ =1.4142
    - tf(you,D2) = 0, tf(say,D2) = 1, tf(hello,D2) = 0
  - $idf(t) = \ln\ (N/(n_j+1)) + 1$
    - idf(you) = 0, idf(say) = ln(2/(2+1)) + 1 = 0.5945, idf(hello) = ln(2/(1+1)) +1 = 1
  - $norm(D) = 1/number\ of\ terms^{1/2}$
    - norm(D1) = $1/6^{1/2}$ =0.4082, norm(D2) = $1/2^{1/2}$ =0.7071
  - Score(Q, D1) = $2/3*0.8596*(1*0.5945^2+1.4142*1^2)*0.4082=0.4135$
  - Score(Q, D2) = $1/2*0.8596*(1*0.5945^2)*0.7071=0.1074$

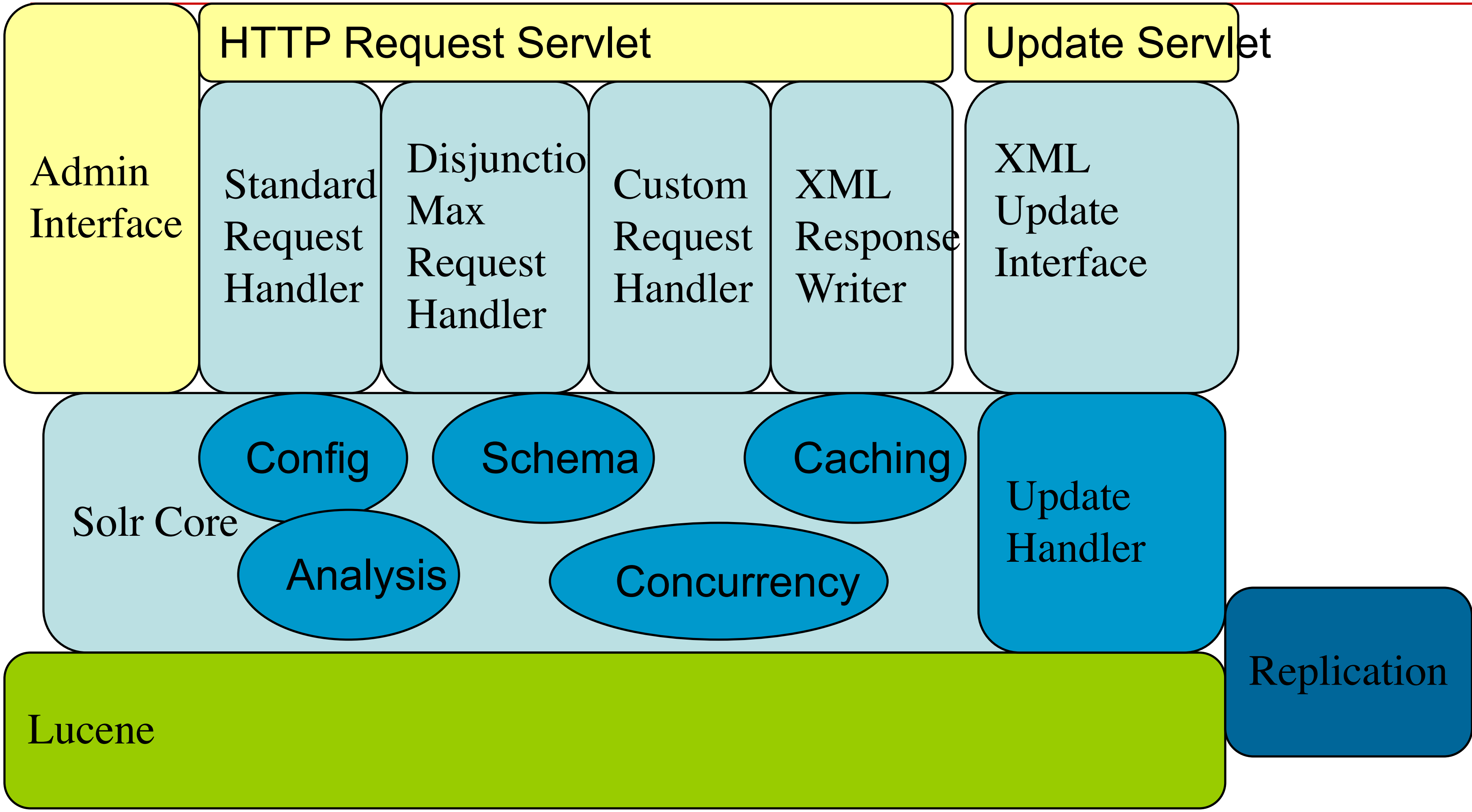
Lucene Sub-projects or Related

- **Nutch**
  - Web crawler with document parsing
- **Hadoop**
  - Distributed file systems and data processing
  - Implements MapReduce
- **Solr**
- **Elasticsearch**
- **Zookeeper**
  - Centralized service (directory) with distributed synchronization

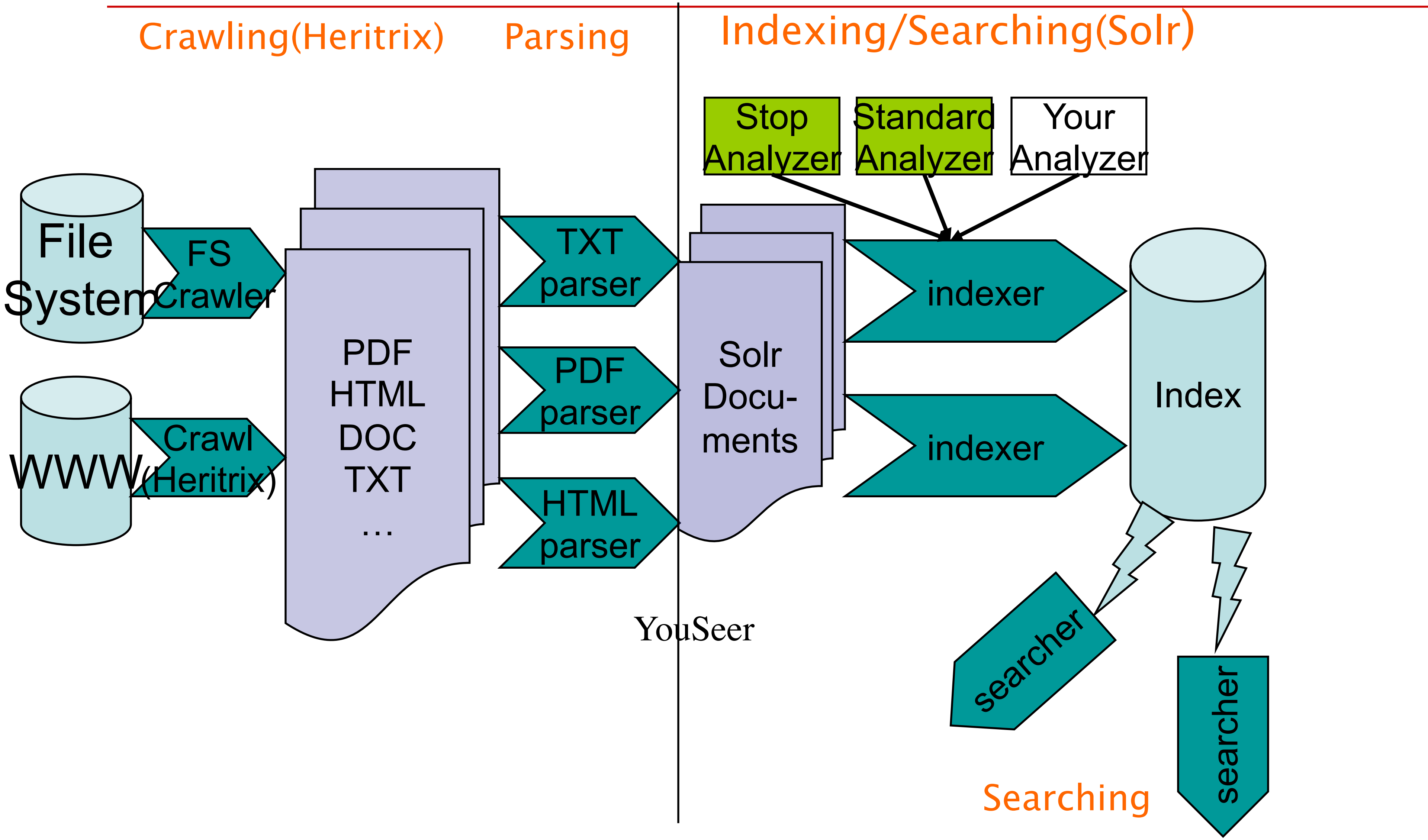
Solr

- Developed by Yonik Seeley at CNET. Donated to Apache in 2006
- **Features**
  - Servlet, Web Administration Interface
  - XML/HTTP, JSON Interfaces
  - Faceting, Schema to define types and fields
  - Highlighting, Caching, Index Replication (Master / Slaves)
  - Pluggable. Java
- **Powered by Solr**
  - Netflix, CNET, Smithsonian, GameSpot, AOL:sports and music
  - Drupal module

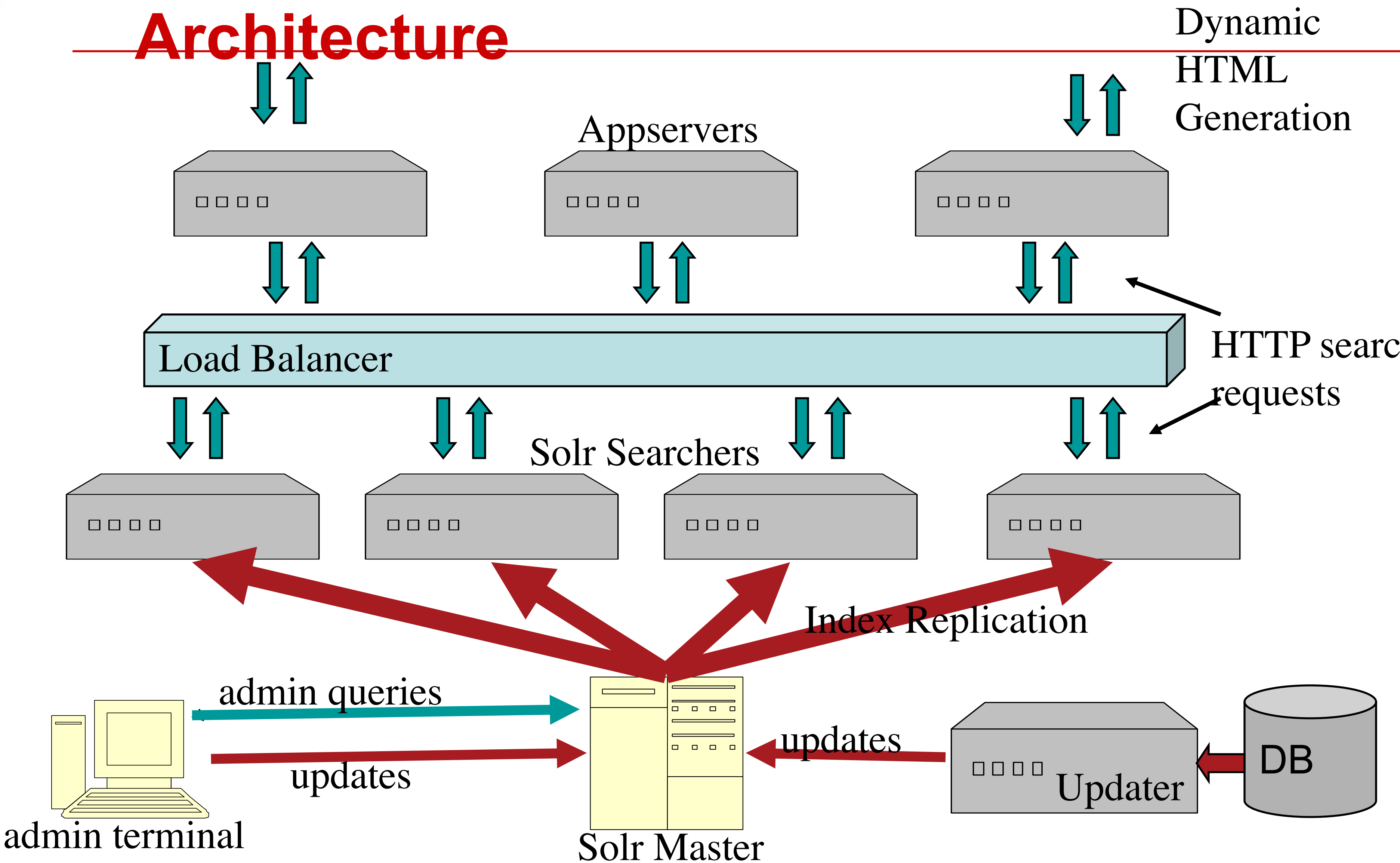
Architecture of Solr



Application usage of Solr: YouSeer search [PennState]

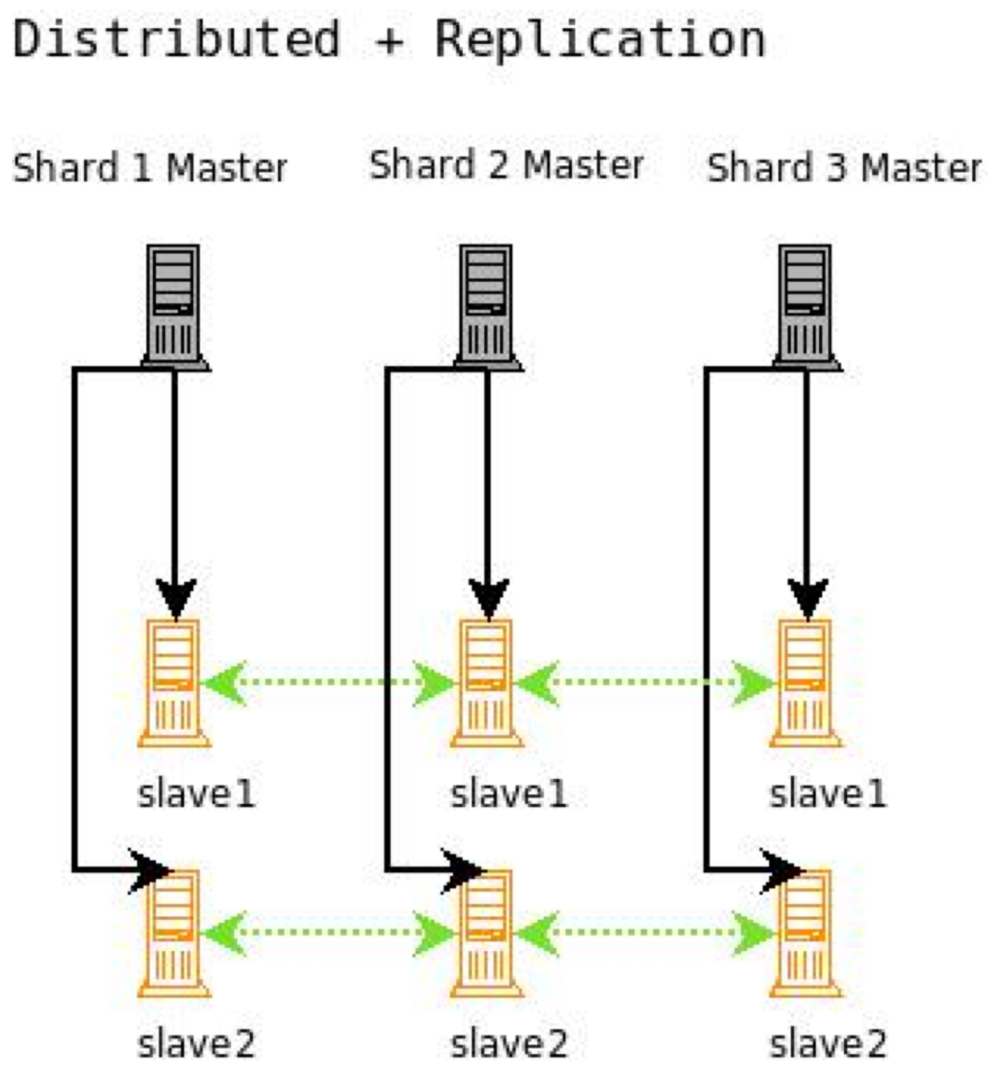


High Availability with Distributed Architecture



Distribution+Replication

- Index is divided into 3 shards.
- Each shard is replicated with 2 copies
- Copies of shards are distributed to different machines for parallel/distributed processing



Four Types of Caching are Supported

IndexSearcher's view of an index is fixed

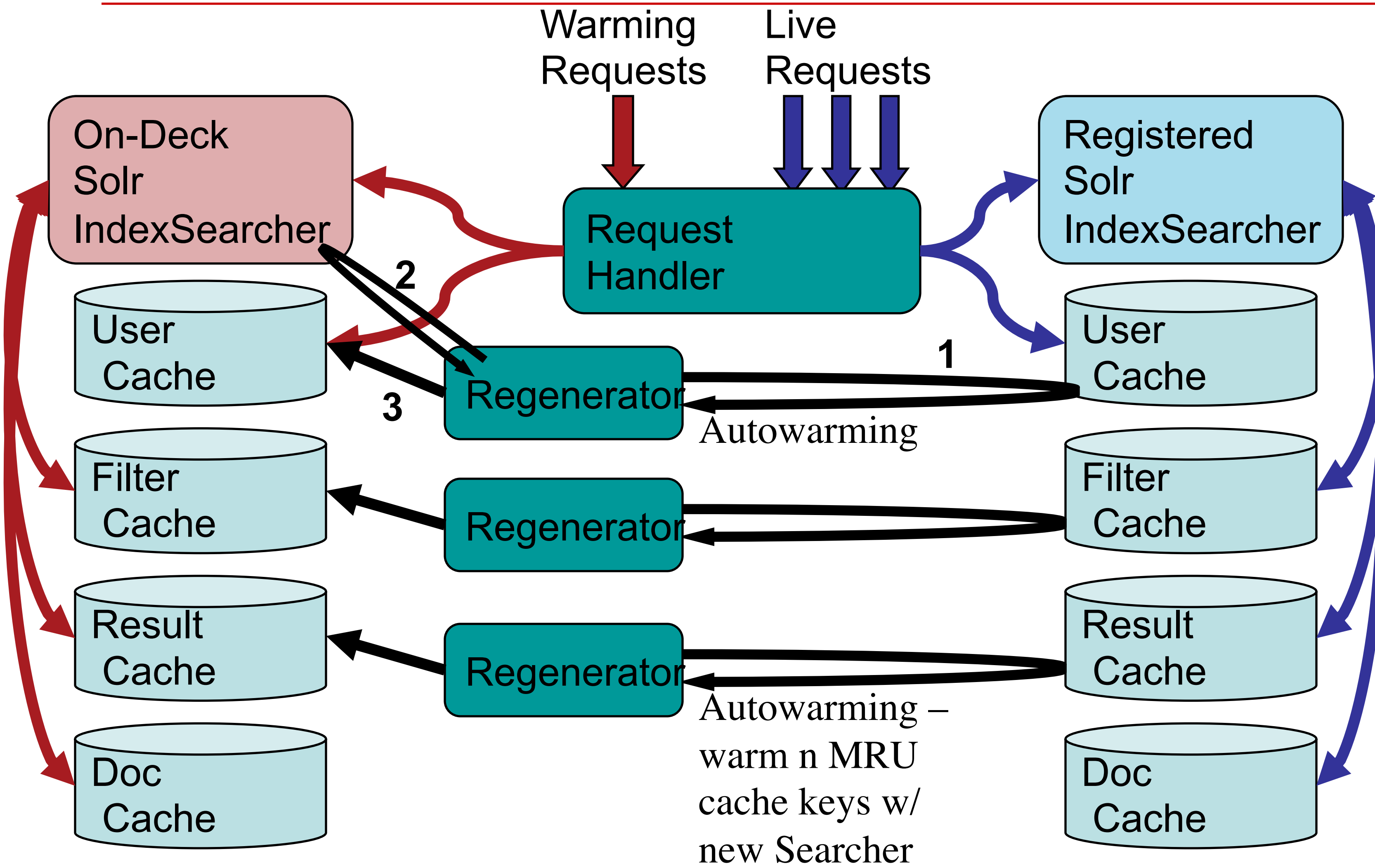
- Aggressive caching possible
- Consistency for multi-query requests

- **filterCache –** unordered set of document ids matching a query. key=Query, val=DocSet
- **resultCache –** ordered subset of document ids matching a query. key=(Query,Sort,Filter), val=DocList
- **documentCache –** the stored fields of documents. key=docid, val=Document
- **userCaches –** application specific, custom query handlers. key=Object, val=Object

Cache Warming

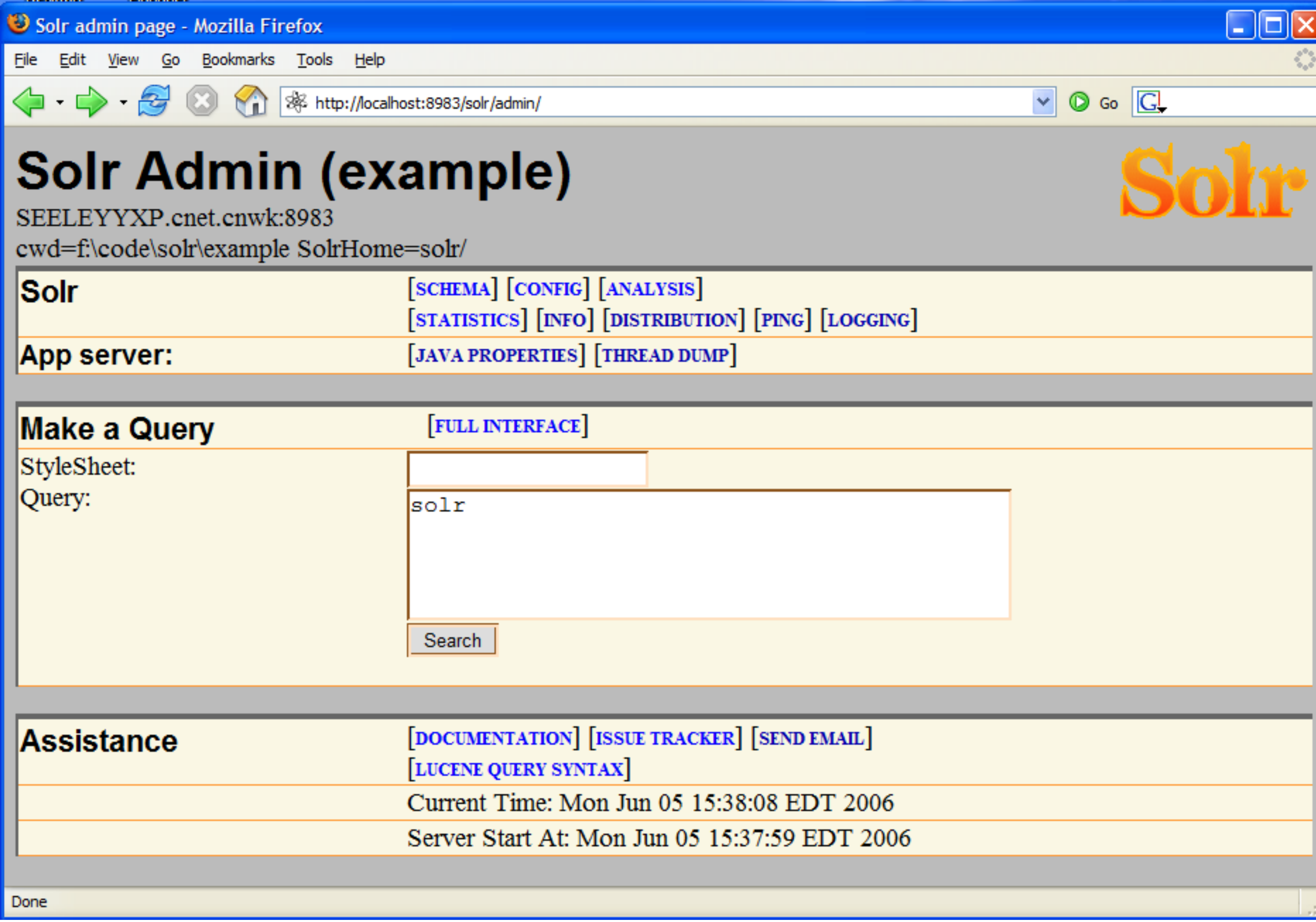
- **Lucene IndexReader warming**
  - field norms, FieldCache, tii – the term index
- **Static Cache warming**
  - Configurable static requests to warm new Searchers
- **Smart Cache Warming (autowarming)**
  - Using MRU items in the current cache to pre-populate the new cache
- **Warming in parallel with live requests**

Architecture View of Smart Cache Warming



Web Admin Interface

- **Show Config, Schema, Distribution info**
- **Query Interface**
- **Statistics**
  - Caches: lookups, hits, hitratio, inserts, evictions, size
  - RequestHandlers: requests, errors
  - UpdateHandler: adds, deletes, commits, optimizes
  - IndexReader, open-time, index-version, numDocs, maxDocs,
- **Analysis Debugger**
  - Shows tokens after each Analyzer stage
  - Shows token matches for query vs index


Solr admin page - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:8983/solr/admin/
Go
Solr Admin (example)
SEELEYYXP.cnet.cnwk:8983
cwd=f:\code\solr\example SolrHome=solr/
Solr
Solr
[SCHEMA] [CONFIG] [ANALYSIS]
[STATISTICS] [INFO] [DISTRIBUTION] [PING] [LOGGING]
App server:
[JAVA PROPERTIES] [THREAD DUMP]
Make a Query
[FULL INTERFACE]
StyleSheet:
Query:
solr
Search
Assistance
[DOCUMENTATION] [ISSUE TRACKER] [SEND EMAIL]
[LUCENE QUERY SYNTAX]
Current Time: Mon Jun 05 15:38:08 EDT 2006
Server Start At: Mon Jun 05 15:37:59 EDT 2006
Done

Adding Documents in Solr

HTTP POST to /update

```
<add><doc boost=“2”>
  <field name=“article”>05991</field>
  <field name=“title”>Apache Solr</field>
  <field name=“subject”>An intro...</field>
  <field name=“category”>search</field>
  <field name=“category”>lucene</field>
  <field name=“body”>Solr is a full...</field>
</doc></add>
```

Updating/Deleting Documents

- **Inserting a document with already present uniqueKey will erase the original**
- **Delete by uniqueKey field (e.g Id)**

<delete><id>05591</id></delete>

- **Delete by Query (multiple documents)**

```
<delete>
  <query>manufacturer:microsoft</query>
</delete>
```

Commit

- **<commit/> makes changes visible**
  - closes IndexWriter
  - removes duplicates
  - opens new IndexSearcher
    - newSearcher/firstSearcher events
    - cache warming
    - “register” the new IndexSearcher
- **<optimize/> same as commit, merges all index segments.**

Default Query Syntax

Lucene Query Syntax

1. mission impossible; releaseDate desc
2. +mission +impossible –actor:cruise
3. “mission impossible” –actor:cruise
4. title:spiderman^10 description:spiderman
5. description:“spiderman movie”~10
6. +HDTV +weight:[0 TO 100]
7. Wildcard queries: te?t, te*t, test*

Default Parameters

Query Arguments for HTTP GET/POST to /select

param	default	description
q		The query
start	0	Offset into the list of matches
rows	10	Number of documents to return
fl	*	Stored fields to return
qt	standard	Query type; maps to query handler
df	(schema)	Default field to search

Search Results

http://localhost:8983/solr/select?q=video&start=0&rows=2&fl=name,price

<response><responseHeader><status>0</status>
<QTime>1</QTime></responseHeader>
<result numFound="16173" start="0">
<doc>
<str name="name">Apple 60 GB iPod with Video</str>
<float name="price">399.0</float>
</doc>
<doc>
<str name="name">ASUS Extreme N7800GTX/2DHTV</str>
<float name="price">479.95</float>
</doc>
</result>
</response>

Schema

- **Lucene has no notion of a schema**
  - Sorting - string vs. numeric
  - Ranges - val:42 included in val:[1 TO 5] ?
  - Lucene QueryParser has date-range support, but must guess.
- **Defines fields, their types, properties**
- **Defines unique key field, default search field, Similarity implementation**

Field Definitions

- **Field Attributes: name, type, indexed, stored, multiValued, omitNorms**

```
<field name="id“ type="string" indexed="true" stored="true"/>
<field name="sku“ type="textTight” indexed="true" stored="true"/>
<field name="name“ type="text“ indexed="true" stored="true"/>
<field name=“reviews“ type="text“ indexed="true“ stored=“false"/>
<field name="category“ type="text_ws“ indexed="true" stored="true“
    multiValued="true"/>
```

Stored means retrievable during search

- **Dynamic Fields, in the spirit of Lucene!**

```
<dynamicField name="*_i" type="sint“ indexed="true" stored="true"/>
<dynamicField name="*_s" type="string“ indexed="true"
    stored="true"/>
<dynamicField name="*_t" type="text“ indexed="true" stored="true"/>
```

Schema: Analyzers

```
<fieldtype name="nametext" class="solr.TextField">
   <analyzer class="org.apache.lucene.analysis.WhitespaceAnalyzer"/>
</fieldtype>

<fieldtype name="text" class="solr.TextField">
   <analyzer>
        <tokenizer class="solr.StandardTokenizerFactory"/>
        <filter class="solr.StandardFilterFactory"/>
        <filter class="solr.LowerCaseFilterFactory"/>
        <filter class="solr.StopFilterFactory"/>
        <filter class="solr.PorterStemFilterFactory"/>
   </analyzer>
</fieldtype>

<fieldtype name="myfieldtype" class="solr.TextField">
   <analyzer>
        <tokenizer class="solr.WhitespaceTokenizerFactory"/>
        <filter class="solr.SnowballPorterFilterFactory"
   language="German" />
   </analyzer>
</fieldtype>
```

More example

```
<fieldtype name="text" class="solr.TextField">
 <analyzer>
   <tokenizer class="solr.WhitespaceTokenizerFactory"/>
   <filter class="solr.LowerCaseFilterFactory"/>
   <filter class="solr.SynonymFilterFactory"
          synonyms="synonyms.txt“/>
   <filter class="solr.StopFilterFactory“
          words=“stopwords.txt”/>
   <filter class="solr.EnglishPorterFilterFactory"
          protected="protwords.txt"/>
 </analyzer>
</fieldtype>
```

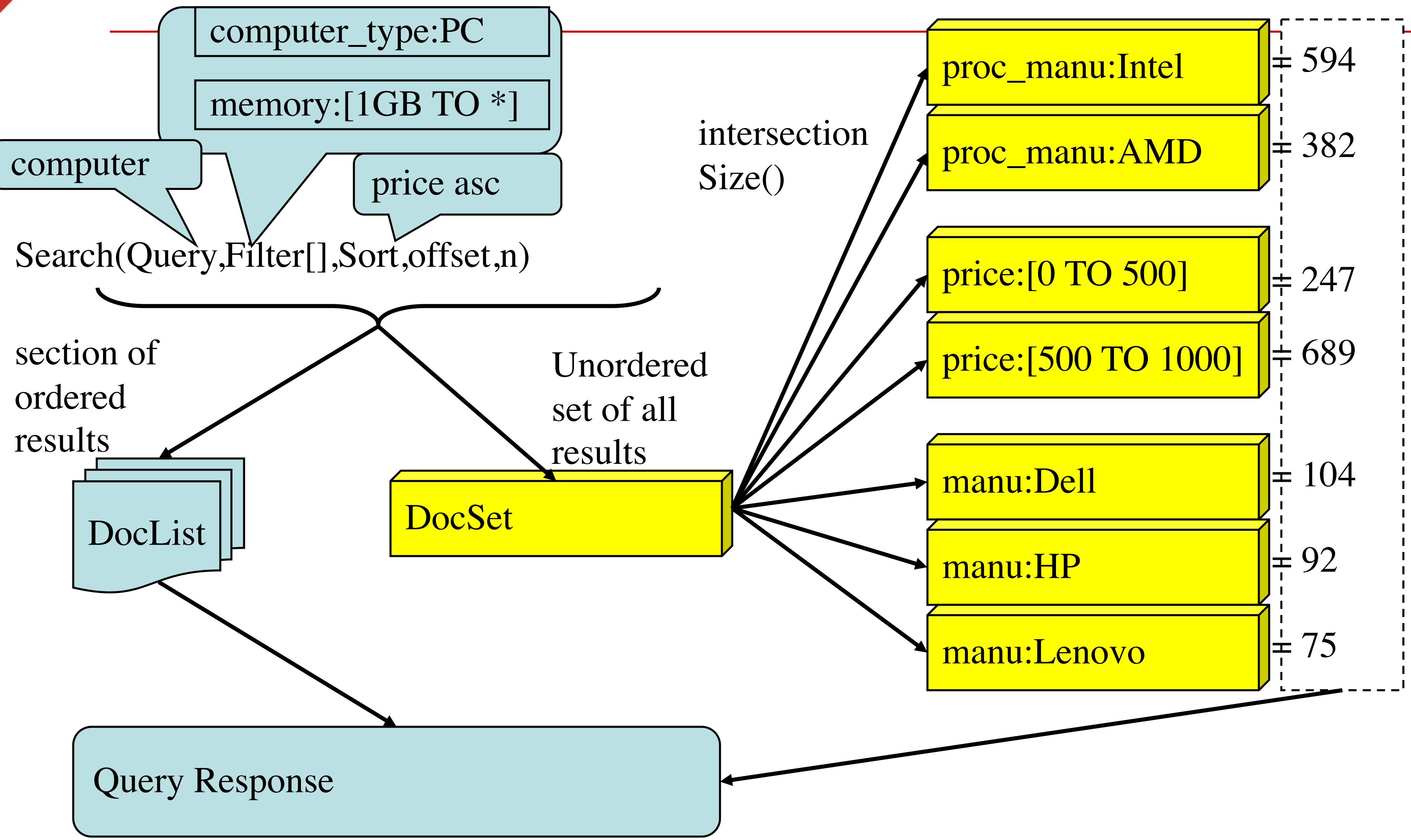
copyField

- **Copies one field to another at index time**
- **Usecase: Analyze same field different ways**
  - copy into a field with a different analyzer
  - boost exact-case, exact-punctuation matches
  - language translations, thesaurus, soundex

```
<field name=“title” type=“text”/>
<field name=“title_exact” type=“text_exact” stored=“false”/>
<copyField source=“title” dest=“title_exact”/>
```

- **Usecase: Index multiple fields into single searchable field**

Faceted Search/Browsing



References

- http://lucene.apache.org/
- http://lucene.apache.org/core/3_6_2/gettingstarted.html
- http://lucene.apache.org/solr/
- http://people.apache.org/~yonik/presentations/

A Comparison of Open Source Search Engines

- Middleton/Baeza-Yates 2010 (Modern Information Retrieval. Text book)

Search Engine	Storage[f]	Increm. Index	Results Excerpt	Results Template	Stop words	Filetype[e]	Stemming	Fuzzy Search	Sort[d]	Ranking	Search Type[c]	Indexer Lang.[b]	License[a]
Datapark	2	■	■	■	■	1,2,3	■	■	1,2	■	2	1	4
ht://Dig	1	■	■	■	■	1,2	■	■	1	■	2	1,2	4
Indri	1	■	■	■	■	1,2,3,4	■	■	1,2	■	1,2,3	2	3
IXE	1	■	■	■	■	1,2,3	□	■	1,2	■	1,2,3	2	8
Lucene	1	■	□	□	■	1,2,4	■	■	1	■	1,2,3	3	1
MG4J	1	■	■	■	■	1,2	■	□	1	■	1,2,3	3	6
mnoGoSearch	2	■	■	■	■	1,2	■	■	1	■	2	1	4
Namazu	1	■	■	■	□	1,2	□	□	1,2	■	1,2,3	1	4
Omega	1	■	□	■	■	1,2,4,5	■	□	1	■	1,2,3	2	4
OmniFind	1	■	■	■	■	1,2,3,4,5	■	■	1	■	1,2,3	3	5
OpenFTS	2	■	□	□	■	1,2	■	■	1	■	1,2	4	4
SWISH-E	1	■	□	□	■	1,2,3	■	■	1,2	■	1,2,3	1	4
SWISH++	1	■	□	□	■	1,2	■	□	1	■	1,2,3	2	4
Terrier	1	□	□	□	■	1,2,3,4,5	■	■	1	■	1,2,3	3	7
WebGlimpse	1	■	■[g]	■[g]	□	1,2	□	■	1[e]	■	1,2,3	1	8,9
XMLSearch	1	■	□	□	■	3	□	■	3	□	1,2,3	2	8
Zettair	1	■	■	□	■	1,2	■	□	1	■	1,2,3	1	2

[a] 1:Apache,2:BSD,3:CMU,4:GPL,5:IBM,6:LGPL,7:MPL,8:Comm,9:Free
[b] 1:C, 2:C++, 3:Java, 4:Perl, 5:PHP, 6:Tcl
[c] 1:phrase, 2:boolean, 3:wild card.
[d] 1:ranking, 2:date, 3:none.
[e] 1:HTML, 2:plain text, 3:XML, 4:PDF, 5:PS.
[f] 1:file, 2:database.
[g] Commercial version only.

■ Available
□ Not Available

Storage type: database vs files.
File type: HTML, plain text, XML, PDF
Incremental index.
Index lang: c/c++/Java
Stopword handling.
Stemming.
Ranking based sorting, date based.
Result summary.
Search type: phrase, boolean, wild card.

A Comparison of Open Source Search Engines for 1.69M Pages

- Middleton/Baeza-Yates 2010 (Modern Information Retrieval)

Search Engine	Indexing Time (h:m:s)		Index Size (%)		Searching Time (ms)		Answer Quality P@5	
ht://Dig	(7)	0:28:30	(10)	104	(6)	32		-
Indri	(4)	0:15:45	(9)	63	(2)	19	(2)	0.2851
IXE	(8)	0:31:10	(4)	30	(2)	19	(5)	0.1429
Lucene	(10)	1:01:25	(2)	26	(4)	21		-
MG4J	(3)	0:12:00	(8)	60	(5)	22	(4)	0.2480
Swish-E	(5)	0:19:45	(5)	31	(8)	45		-
Swish++	(6)	0:22:15	(3)	29	(10)	51		-
Terrier	(9)	0:40:12	(7)	52	(9)	50	(3)	0.2800
XMLSearch	(2)	0:10:35	(1)	**22**	(1)	**12**		-
Zettair	(1)	**0:04:44**	(6)	33	(6)	32	(1)	**0.3240**

Table 6.1: Ranking of search engines, comparing their indexing time, index size, and the average searching time (for the 2.7GB collection), and the Answer Quality for the engines that parsed the WT10g. The number in parentheses corresponds to the relative position of the search engine.

A Comparison of Open Source Search Engines

- July 2009, Vik's blog (http://zooie.wordpress.com/2009/07/06/a-comparison-of-open-source-search-engines-and-indexing-twitter/)

Platform	License	Lang.	Docs	Ranking	Users	Support	Parallel	Scale
Lucene	Apache	Java	Many	Flexible	Amazon	5/5	Yes	TB
zettair	BSD like	C	HTML, TREC, TXT	Flexible	Research	1/5	No	TB
Indri	BSD like	C++	Many	Very Flexible	Research	1.5/5	Yes	TB
Sphinx	GPL	C++	Many	Flexible	craigslist	4/5	Yes	TB
RDBMS	BSD, GPL	C	SQL Text	Limited	-	3/5	Maybe	GB
Xapian	GPL	C++	Many	Flexible	gmane	3/5	Yes	TB

A Comparison of Open Source Search Engines

- [Vik's blog(http://zooie.wordpress.com/2009/07/06/a-comparison-of-open-source-search-engines-and-indexing-twitter/](http://zooie.wordpress.com/2009/07/06/a-comparison-of-open-source-search-engines-and-indexing-twitter/))

TREC Filtering OHSUMED Data Set
63 Topics = Queries ("37 yr old man with sickle cell disease"); Avg. Len: 6.7; OR'ed
196,403 Medical Results (**300MB** Indexable Text)
Judgement Data: (Topic, Result, 2 or 1 or 0 Rating)
Relevancy: **DCG 10**

Platform	Index Peak Memory	Index Time	Index Size	Search Peak Memory	Search Time	Relevancy
Lucene 2.4.1	37 MB	2m15s	**91 MB**	18 MB	0.02168s (1.366s)	**1.0449**
zettair 0.9.3	22 MB	**0m29.34s**	122 MB	9 MB	0.02609s (1.644s)	0.8299
sphinx 0.9.8.1	19 MB	0m42.35s	201 MB	16 MB	**0.00803s** (0.506s)	0.7690
sqlite 3.6.11	**8 MB**	1m54.91s	474 MB	7 MB	0.91451s (54.614s)	0.0166
Xapian 1.0.13	48 MB	6m38.17s	339 MB	**1 MB**	0.02286s (1.440s)	1.0162