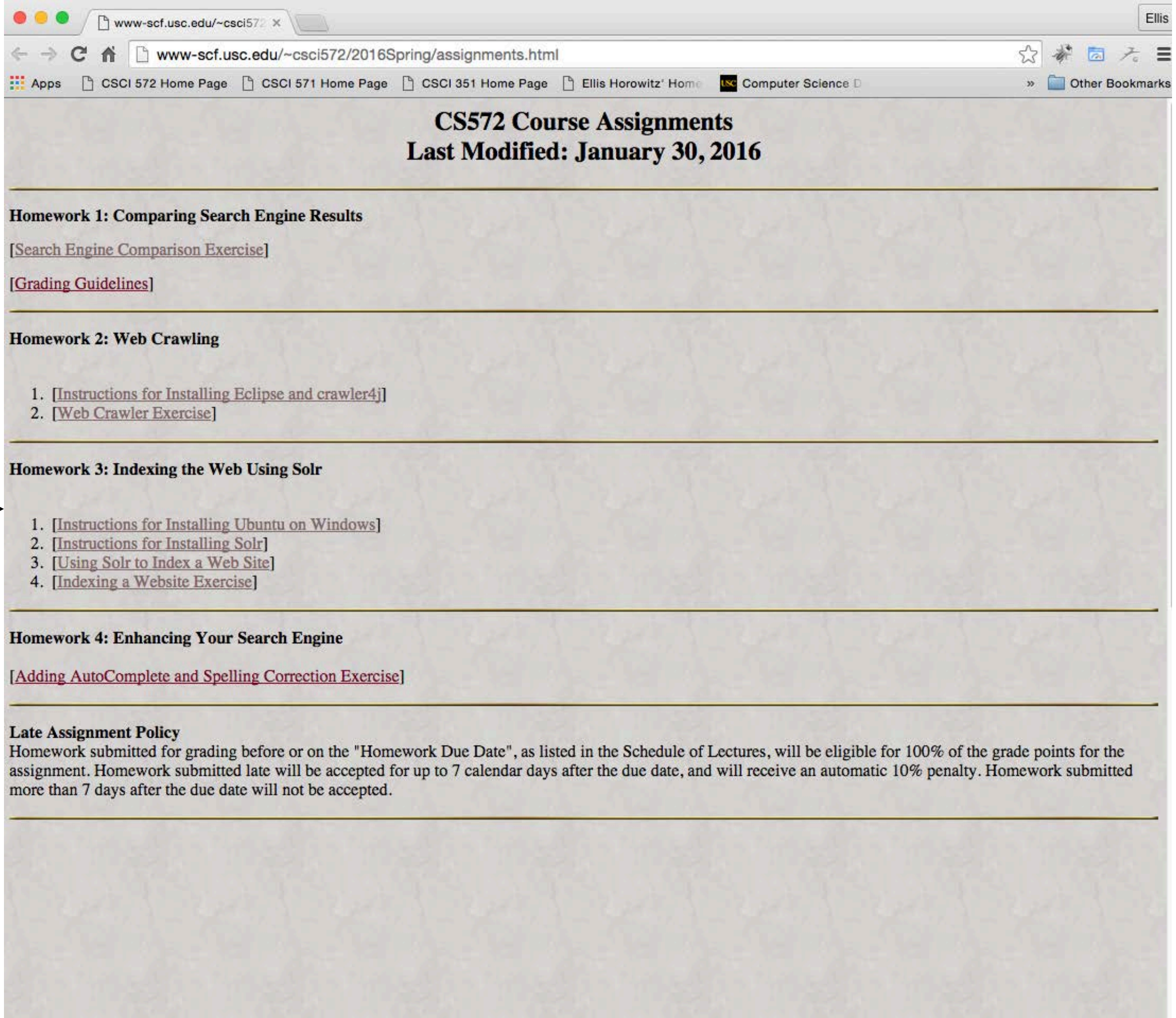
HW3 Overview

There are 4 components to this homework; you will possibly not need all of them;

1. Installing Ubuntu
2. Installing Solr
3. Using Solr to Index your downloaded web pages
4. the exercise - comparing ranking algorithms



CS572 Course Assignments
Last Modified: January 30, 2016

Homework 1: Comparing Search Engine Results

[Search Engine Comparison Exercise]

[Grading Guidelines]

Homework 2: Web Crawling

1. [Instructions for Installing Eclipse and crawler4j]
2. [Web Crawler Exercise]

Homework 3: Indexing the Web Using Solr

1. [Instructions for Installing Ubuntu on Windows]
2. [Instructions for Installing Solr]
3. [Using Solr to Index a Web Site]
4. [Indexing a Website Exercise]

Homework 4: Enhancing Your Search Engine

[Adding AutoComplete and Spelling Correction Exercise]

Late Assignment Policy
Homework submitted for grading before or on the "Homework Due Date", as listed in the Schedule of Lectures, will be eligible for 100% of the grade points for the assignment. Homework submitted late will be accepted for up to 7 calendar days after the due date, and will receive an automatic 10% penalty. Homework submitted more than 7 days after the due date will not be accepted.

Step 1: Ubuntu with VirtualBox

- **VirtualBox** is an open source, freely available Windows application (it also runs on other platforms) that lets you run multiple operating systems on your single machine
  - E.g. run Windows on a Mac, run Linux on Windows
  - Major supported operating systems include: Windows NT 4.0, Windows 2000, Windows 8, Windows 10, DOS Windows 3.x, Linux, Solaris, FreeBSD, OpenBSD
- **Ubuntu** is a Linux-based operating system distributed on personal computers, smartphones and network servers. It uses **Unity** as its default desktop environment
- **Solr requires a Unix environment** to run, so step 1 is required if you plan to use your Windows laptop

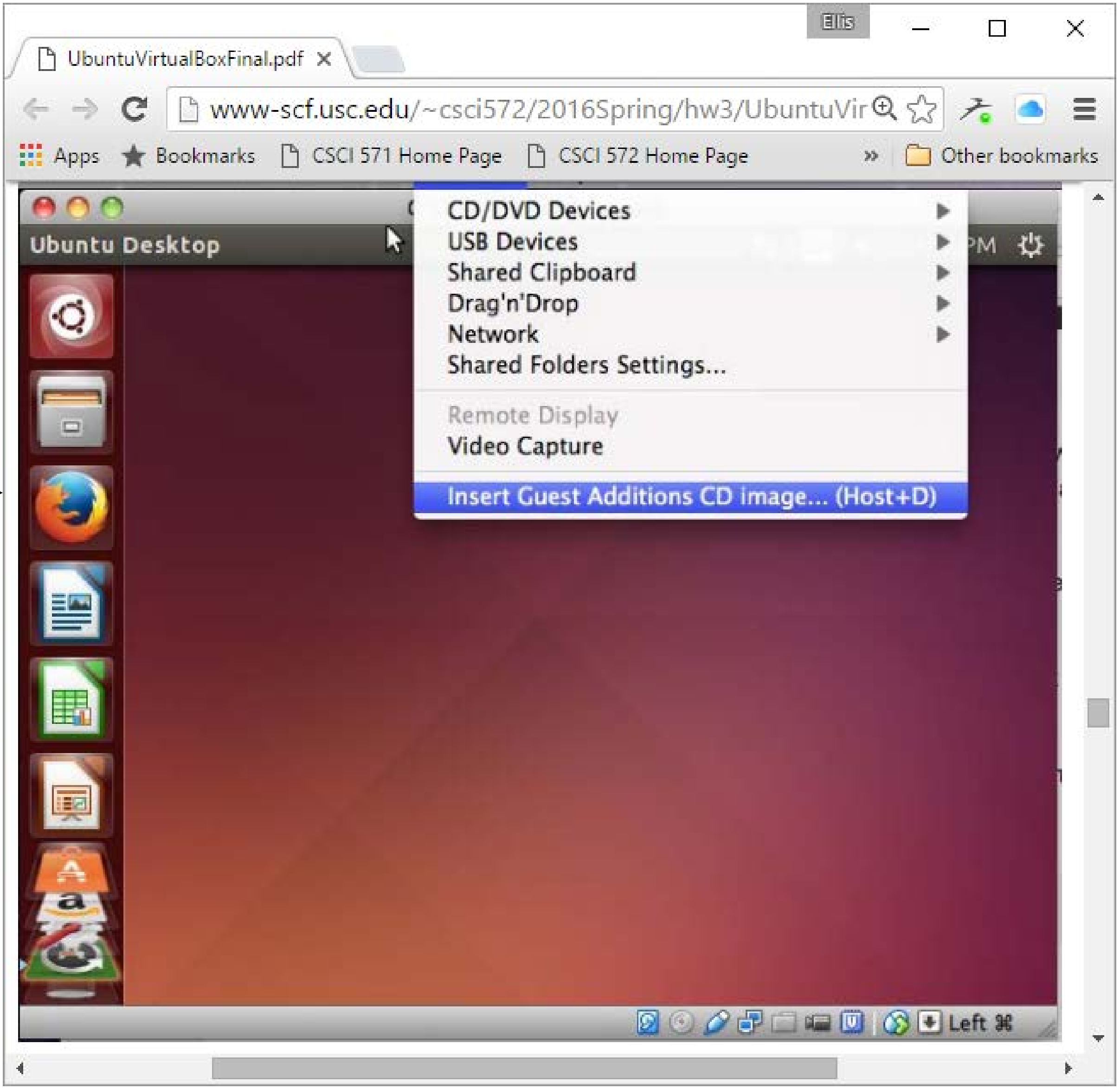
Step 1: Setting Up Ubuntu with VirtualBox

1. Download the free version of VirtualBox for Windows machines
   - Instructions can be found here

   http://www-scf.usc.edu/~csci572/Exercises/UbuntuVirtualBoxFinal.pdf
2. Download the Ubuntu 64-bit version
3. Run VirtualBox and select your Ubuntu version as the New Application
4. Set various parameters
5. Install Ubuntu and you should be ready to run

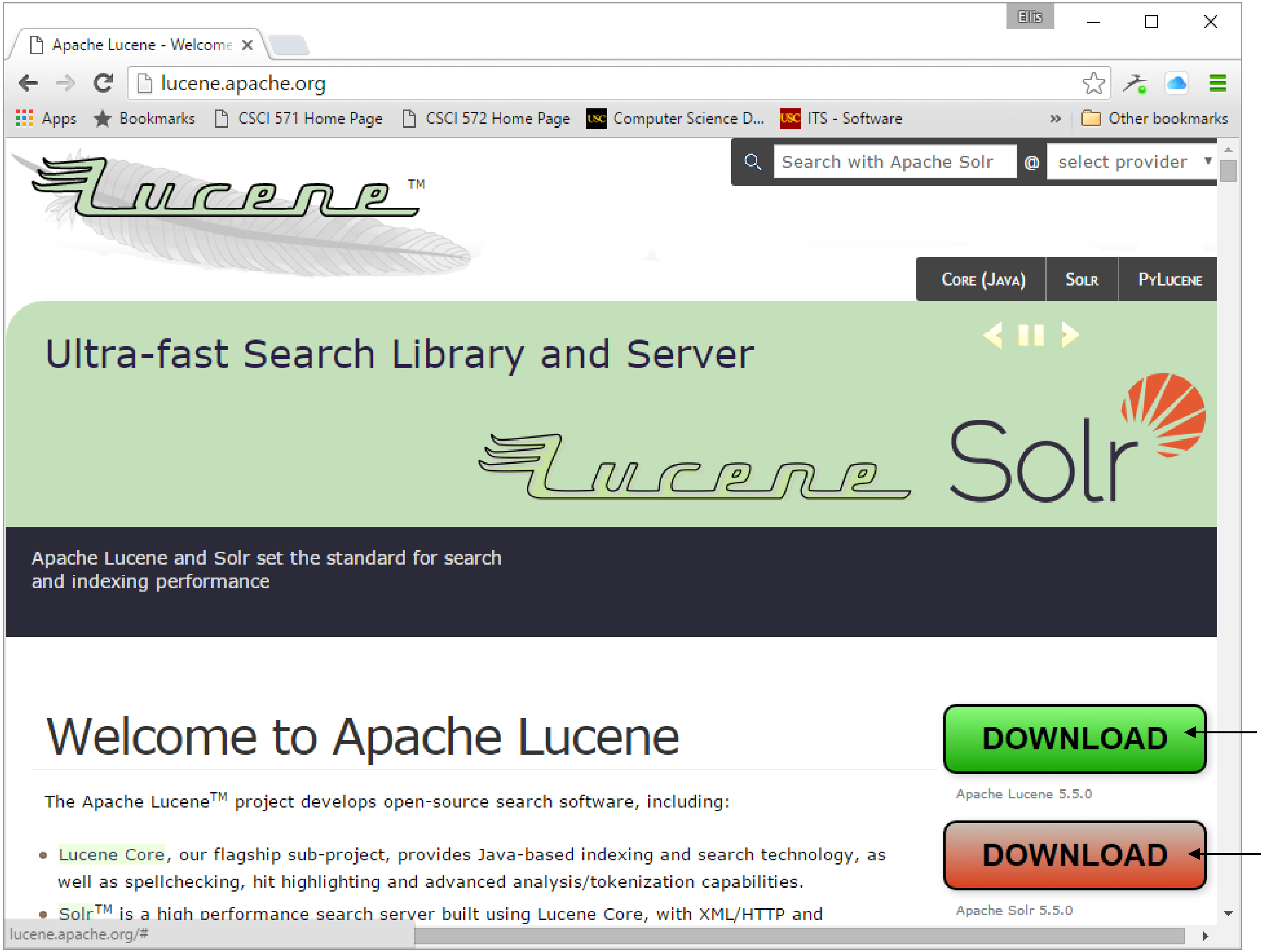
Your Ubuntu/Unity Desktop

Built-in applications
including
Firefox browser



Step 2: Installing Solr

- Solr is an open source enterprise search server based on the Lucene Java search library
- Instructions for downloading and installing Solr can be found here
  - http://www-scf.usc.edu/~csci572/2016Spring/hw3/SolrInstallation.pdf
- Fast, high performance, scalable search/Information Retrieval library
- Initially developed by Doug Cutting (Also author of Hadoop)
- it provides for Indexing and Searching of documents
- produces an Inverted Index of documents
- Provides advanced Search options like synonyms, stopwords, based on similarity, proximity.
- **http://lucene.apache.org/ is the main page for both Lucene and Solr**



Ultra-fast Search Library and Server

Apache Lucene and Solr set the standard for search and indexing performance

Welcome to Apache Lucene

DOWNLOAD

Apache Lucene 5.5.0

DOWNLOAD

Apache Solr 5.5.0

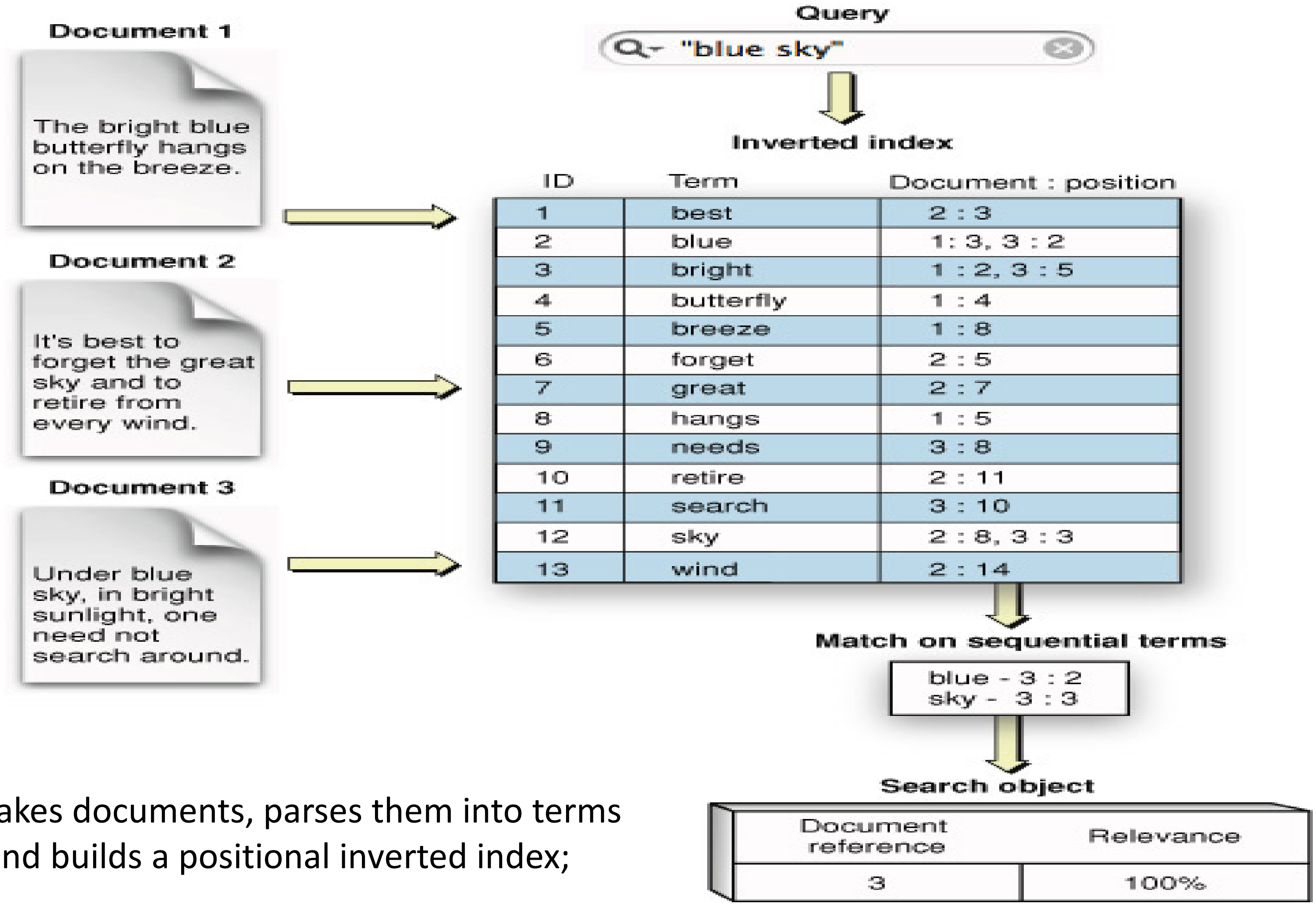
The Apache Lucene™ project develops open-source search software, including:

- Lucene Core, our flagship sub-project, provides Java-based indexing and search technology, as well as spellchecking, hit highlighting and advanced analysis/tokenization capabilities.
- Solr™ is a high performance search server built using Lucene Core, with XML/HTTP and

Download Lucene

Download Solr

Lucene Internals - Positional Inverted Index



Document 1

The bright blue butterfly hangs on the breeze.

Document 2

It's best to forget the great sky and to retire from every wind.

Document 3

Under blue sky, in bright sunlight, one need not search around.

Query

"blue sky"

Inverted index

ID	Term	Document : position
1	best	2 : 3
2	blue	1: 3, 3 : 2
3	bright	1 : 2, 3 : 5
4	butterfly	1 : 4
5	breeze	1 : 8
6	forget	2 : 5
7	great	2 : 7
8	hangs	1 : 5
9	needs	3 : 8
10	retire	2 : 11
11	search	3 : 10
12	sky	2 : 8, 3 : 3
13	wind	2 : 14

Match on sequential terms

blue - 3 : 2
sky - 3 : 3

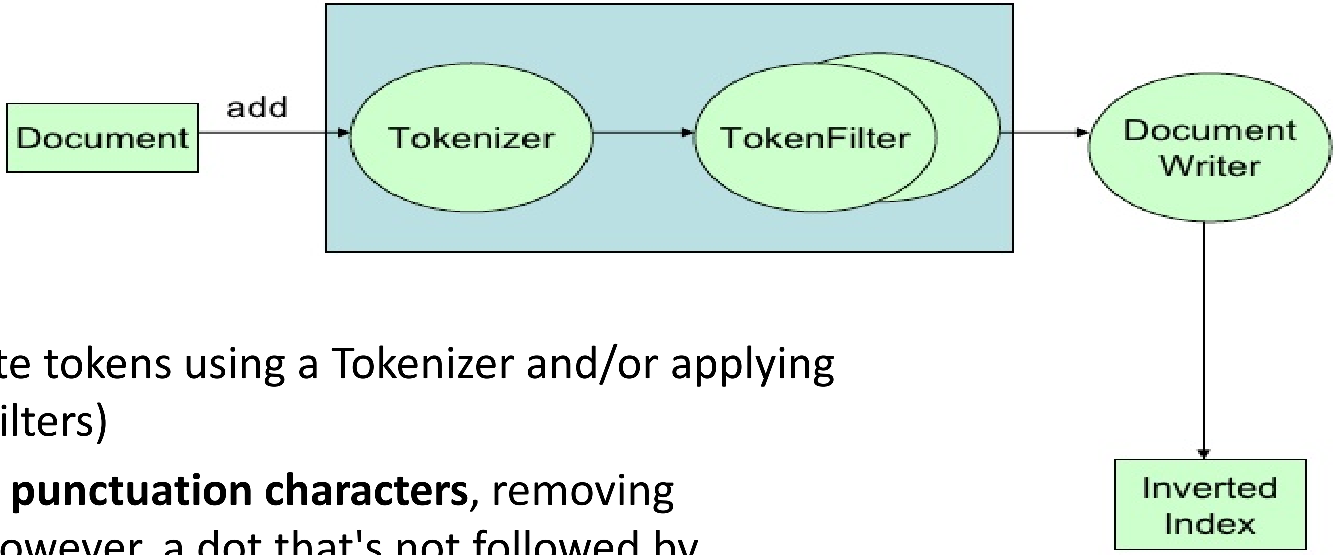
Search object

Document reference	Relevance
3	100%

takes documents, parses them into terms and builds a positional inverted index;

Lucene Indexing Pipeline

Analyzer



- Analyzer : create tokens using a Tokenizer and/or applying Filters (Token Filters)
- **Splits words at punctuation characters**, removing punctuation. However, a dot that's not followed by whitespace is considered part of a token.
- **Splits words at hyphens**, unless there's a number in the token, in which case the whole token is interpreted as a product number and is not split.
- **Recognizes** email addresses and internet hostnames as one token.

Analyzers, Tokenizers, and Filters

- Field analyzers are used both during ingestion, when a document is indexed, and at query time.
  - An analyzer examines the text of fields and generates a token stream.
- Tokenizers break field data into lexical units, or *tokens*.
- Filters examine a stream of tokens and keep them, transform or discard them, or create new ones.
- Analyzers: Tokenizers and filters may be combined to form pipelines, or *chains*, where the output of one is input to the next. Such a sequence of tokenizers and filters is called an *analyzer* and the resulting output of an analyzer is used to match query results or build indices.

Lucene Scoring Concepts

TF - IDF

Lucene scores using a combination of TF-IDF and vector closeness

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

TF-IDF

Term x within document y

$tf_{x,y}$ = frequency of x in y

df_x = number of documents containing x

N = total number of documents

TF - IDF = Term Frequency **X** Inverse Document Frequency

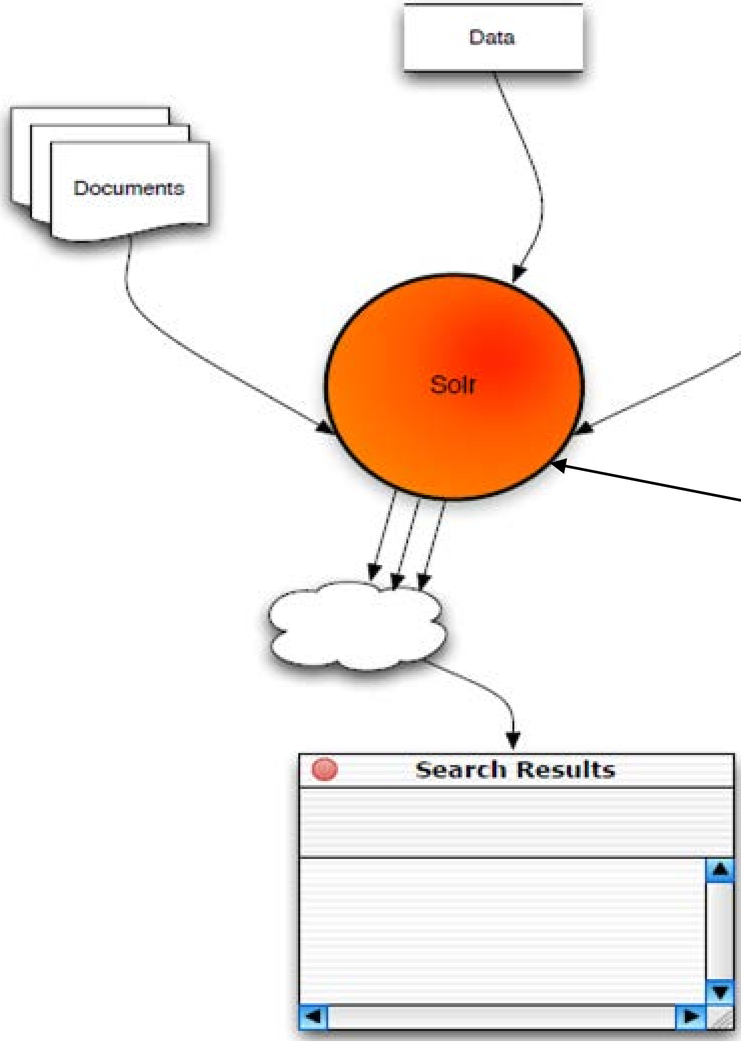
cosine-similarity(query_vector, document_vector) = V(q) * V(d)/|V(q)|*|V(d)|
where V(q)*V(d) is the dot product of the weighted vectors and |V(q)|, |V(d)|
are the Euclidean norms of the vectors (square root of the sum of squares)

for details see
https://lucene.apache.org/core/4_0_0/core/org/apache/lucene/search/similarities/TFIDFSimilarity.html

Apache Solr

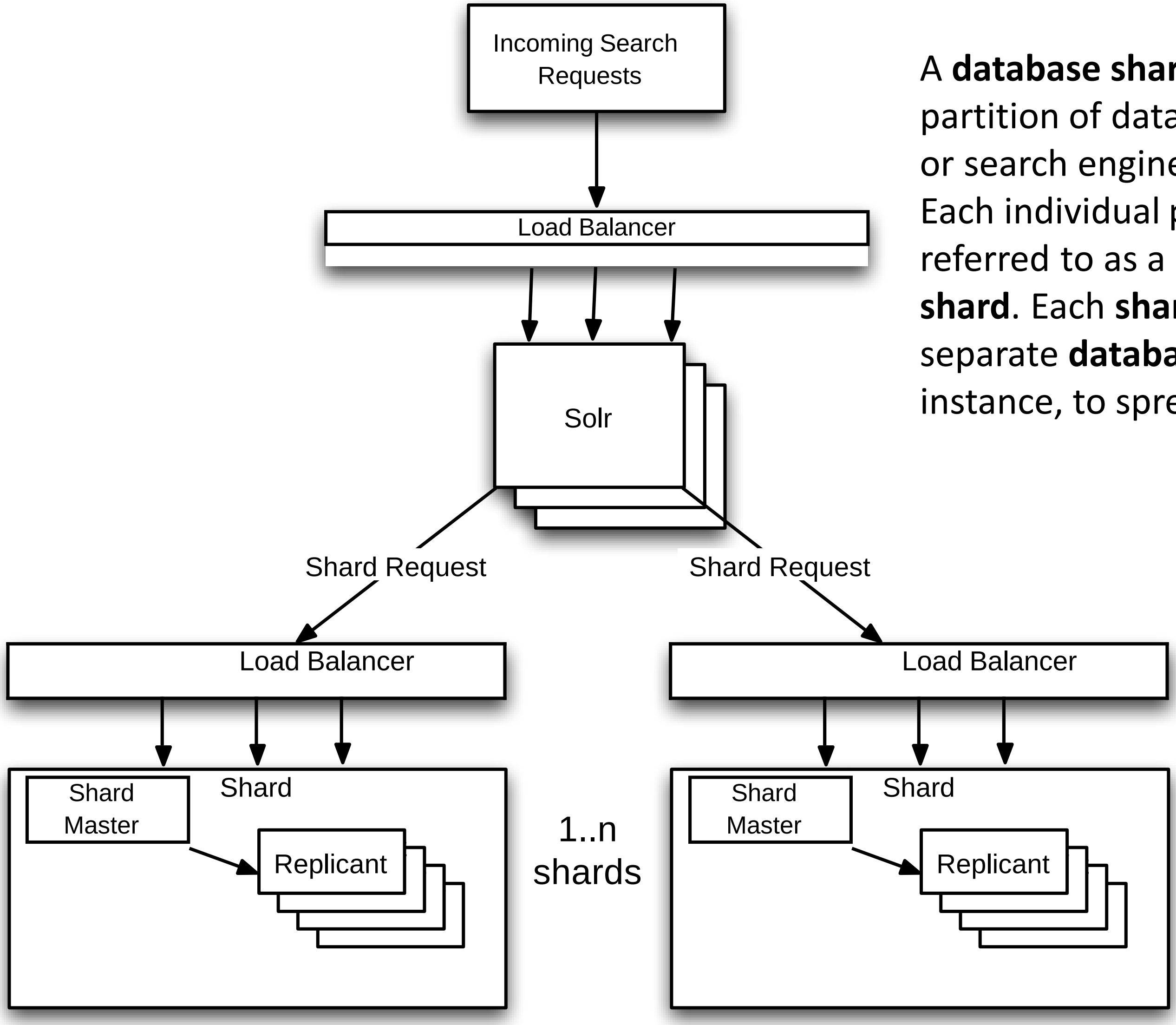
- Created by Yonik Seeley for CNET
- Enterprise Search platform for Apache Lucene
- Open source
- Highly reliable, scalable, fault tolerant
- Support distributed Indexing (SolrCloud), Replication, and load balanced querying
- **http://lucene.apache.org/solr**

High level overview



there is a Solr server which accepts queries in the form of http requests and returns results as JSON strings

Solr in Production



A **database shard** is a horizontal partition of data in a **database** or search engine.
Each individual partition is referred to as a
shard. Each **shard** is held on a separate **database** server instance, to spread load.

How to start

1. Start Solr

```
java -jar start.jar
```

2. Index your data

```
java -jar post.jar *.xml
```

3. Search **http://localhost:8983/solr**

localhost indicates the Solr server is running locally on port 8983

Complete installation instructions can be found at

http://www-scf.usc.edu/~csci572/2016Spring/hw3/SolrInstallation.pdf

Querying Data

HTTP GET or POST with parameters are used to specify queries

E.g. here are 4 sample queries, some with various parameters

http://solr/select?q=electronics

http://solr/select?q=electronics&sort=price+desc

http://solr/select?q=electronics&rows=50&start=50

http://solr/select?q=electronics&fl=name+price ← limit results to fields: name and price

Querying Data: Results

Canonical response format is XML, though JSON is often used as well

```
<response>
  <lst name="responseHeader">
    <int name="status">0</int>
    <int name="QTime">1</int>
  </lst>
  <result name="response" numFound="14" start="0">
    <doc>
      <arr name="cat">
        <str>electronics</str>
        <str>connector</str>
      </arr>
      <arr name="features">
         <str>car power adapter, white</str>
      </arr>
      <str name="id">F8V7067-APL-KIT</str>
      ...
```

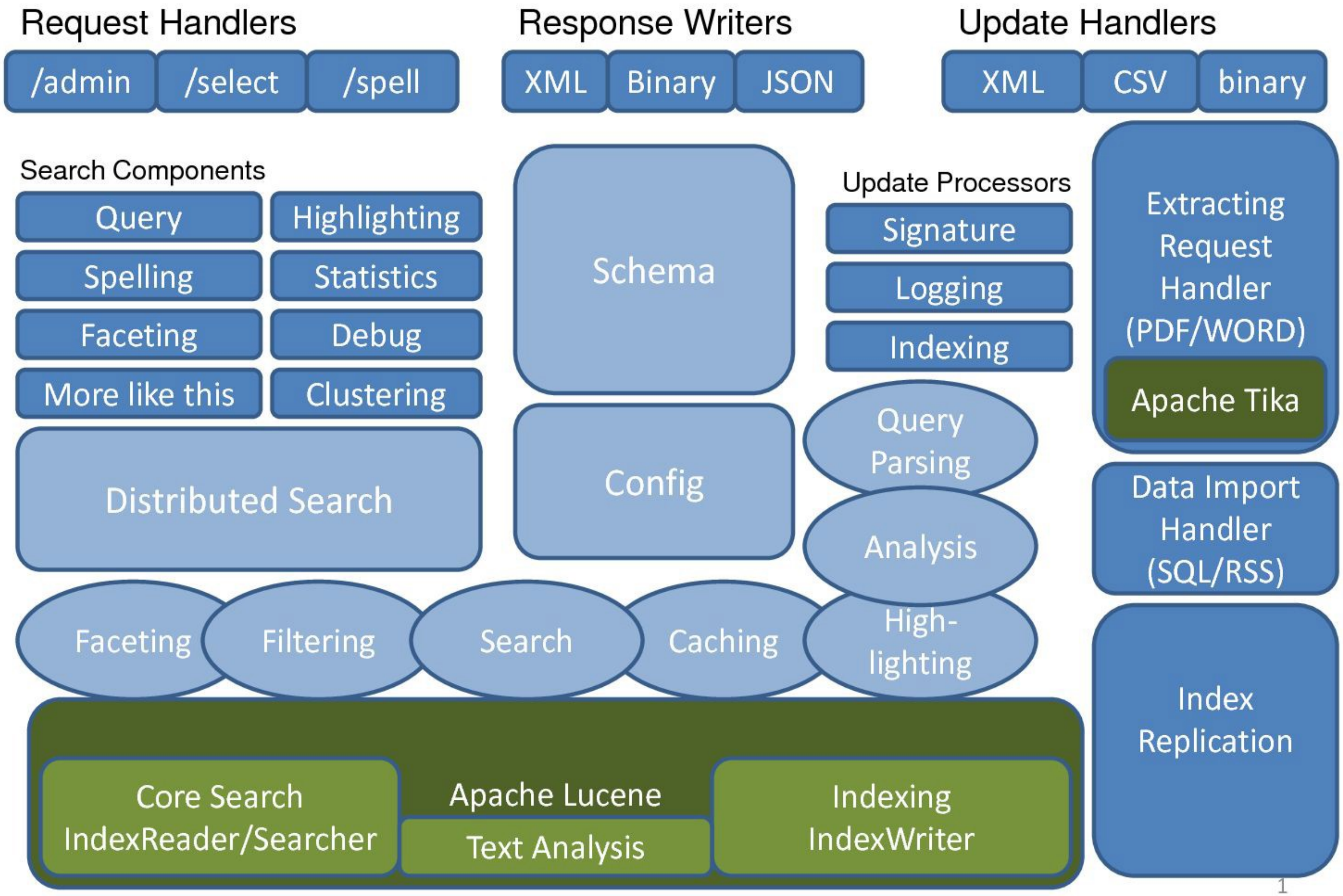
Query Types

- Single and multi term queries
  - ex fieldname:value or title: software engineer
- +, -, AND, OR NOT operators.
  - ex. title: (software AND engineer)
- Range queries on date or numeric fields,
  - ex: timestamp: [* TO NOW] or price: [1 TO 100]
- Boost queries:
  - e.g. title:Engineer ^1.5 OR text:Engineer
- Fuzzy search : is a search for words that are similar in spelling
  - e.g. roam~0.8 => noam
- Proximity Search : with a sloppy phrase query. The close together the two terms appear, higher the score.
  - ex “apache lucene”~20 : will look for all documents where “apache” word occurs within 20 words of “lucene”

Solr is Used by Many

- **Search Engine**
  - Yandex.ru, DuckDuckGo.com
- **News Paper**
  - Guardian.co.uk
- **Music/Movies**
  - Apple.com, Netflix.com
- **Events**
  - Stubhub.com, Eventbrite.com
- **Cloud Log Management**
  - Loggly.com
- **Others**
  - Whitehouse.gov
- **Jobs**
  - Indeed.com, Simplyhired.com, Naukri.com
- **Auto**
  - AOL.com
- **Travel**
  - Cleartrip.com
- **Social Network**
  - Twitter.com, LinkedIn.com, mylife.com

Lucene/Solr Architecture



Request Handlers

/admin /select /spell

Response Writers

XML Binary JSON

Update Handlers

XML CSV binary

Search Components

Query Highlighting

Spelling Statistics

Faceting Debug

More like this Clustering

Distributed Search

Schema

Config

Update Processors

Signature

Logging

Indexing

Query Parsing

Analysis

High-lighting

Extracting Request Handler (PDF/WORD)

Apache Tika

Data Import Handler (SQL/RSS)

Index Replication

Faceting Filtering Search Caching

Apache Lucene

Core Search IndexReader/Searcher

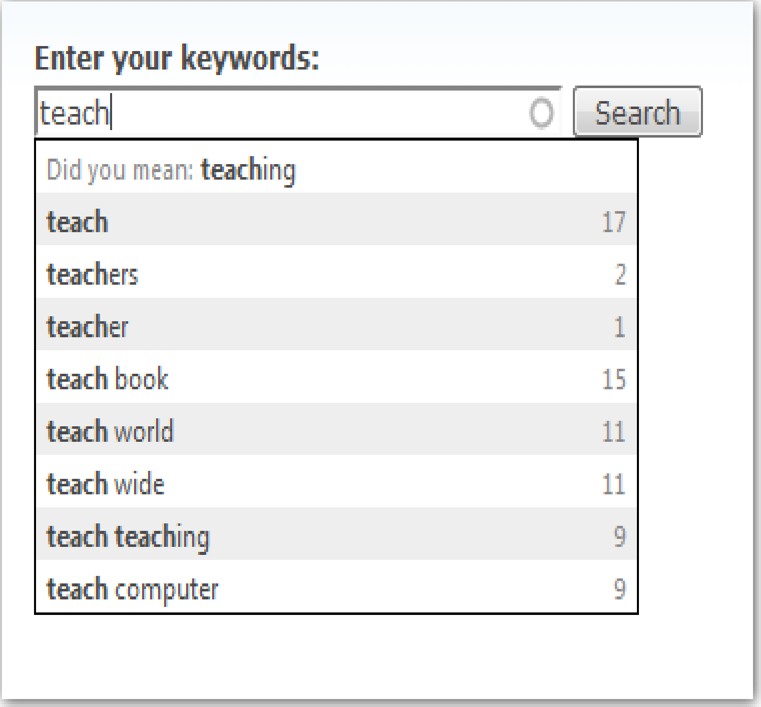
Text Analysis

Indexing IndexWriter

Solr Includes Spell Checking

- Not enabled by default, see example config to wire it in
- https://cwiki.apache.org/confluence/display/solr/Spell+Checking
- File or index-based dictionaries for spell correction
- Supports pluggable distance algorithms:
    - Levenstein alg: https://en.wikipedia.org/wiki/Levenshtein_distance
    - JaroWinkler alg: , https://en.wikipedia.org/wiki/Jaro%E2%80%93Winkler_distance
- http://wiki.apache.org/solr/SpellCheckComponent is a full discussion of the spell checking abilities of Solr

Solr Includes Autosuggestion

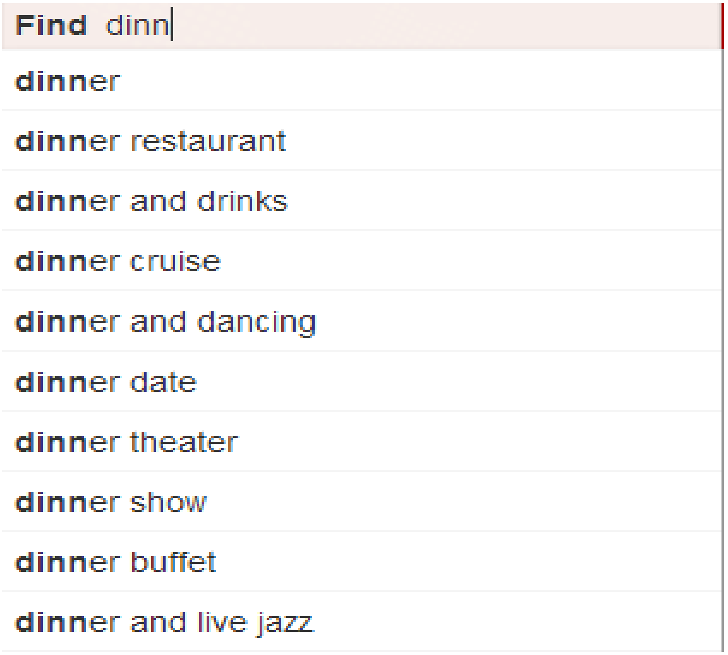


Enter your keywords:

teach

Did you mean: **teaching**

teach	17
teachers	2
teacher	1
teach book	15
teach world	11
teach wide	11
teach **teach**ing	9
teach computer	9



Find dinn

- **dinn**er
- **dinn**er restaurant
- **dinn**er and drinks
- **dinn**er cruise
- **dinn**er and dancing
- **dinn**er date
- **dinn**er theater
- **dinn**er show
- **dinn**er buffet
- **dinn**er and live jazz

https://wiki.apache.org/solr/Suggester

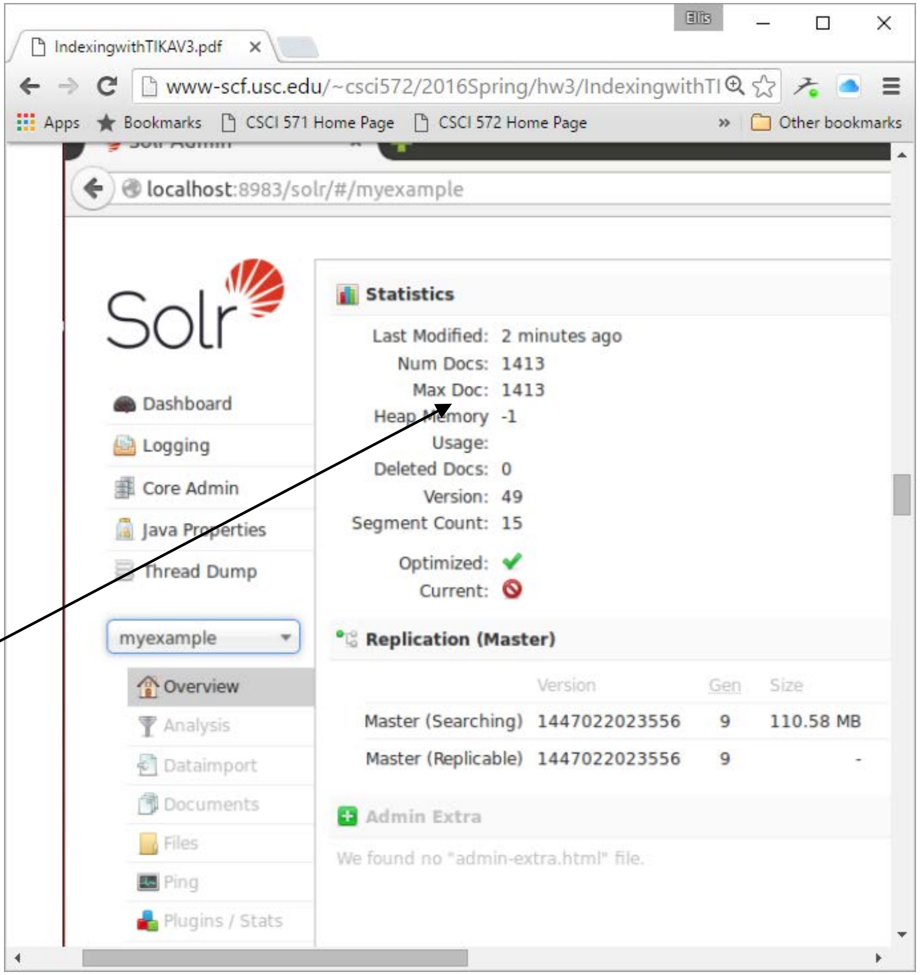
Step 3: Solr to Index a Web Site

- start the Solr server
- start a new Solr core
- Use Tika to import your saved files
- Use the Solr interface to check that the files have been properly indexed
- Note the URL:

localhost:8983/solr/#/myexample

- 1413 docs successfully

indexed



Statistics

```
      Last Modified:  2 minutes ago
           Num Docs:  1413
            Max Doc:  1413
  Heap Memory Usage:  -1
       Deleted Docs:  0
            Version:  49
      Segment Count:  15
          Optimized:  ✓
            Current:  ⊘
```

Replication (Master)

	Version	Gen	Size
Master (Searching)	1447022023556	9	110.58 MB
Master (Replicable)	1447022023556	9	-

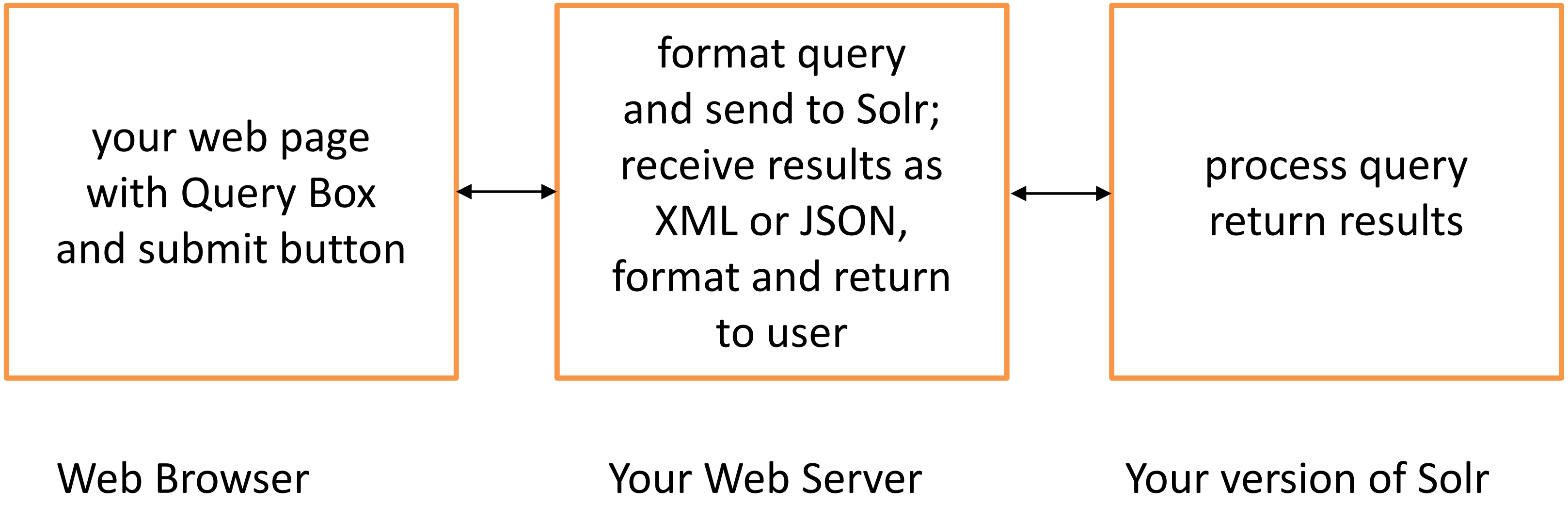
Admin Extra

We found no "admin-extra.html" file.

Step 4: The Actual Exercise

Comparing Search Engine Ranking Algorithms

1. From homework 2 you should have saved all of the HTML, etc. files you downloaded
2. You should install Solr as described previously
3. Take the pages you saved and index them in Solr, as described in Step 3
4. Build a front end to Solr that permits a visitor to enter a query and get matching results
5. Solr will return the results in JSON format; your server needs to take the results and format them for the user



your web page
with Query Box
and submit button

format query
and send to Solr;
receive results as
XML or JSON,
format and return
to user

process query
return results

Web Browser

Your Web Server

Your version of Solr

Step 4: The Actual Exercise

a PHP client that accepts input from the user in a HTML form, and sends the request to the Solr server. After the Solr server processes the query, it returns the results which is parsed by the PHP program and formatted for display

```
<?php
// make sure browsers see this page as utf-8 encoded HTML
header('Content-Type: text/html; charset=utf-8');
$limit = 10;
$query = isset($_REQUEST['q']) ? $_REQUEST['q'] : false;
$results = false;
if ($query)
{  require_once('Apache/Solr/Service.php');
 // create a new solr service instance - host, port, and corename
 // path (all defaults in this example)
 $solr = new Apache_Solr_Service('localhost', 8983, '/solr/core_name/');
 // if magic quotes is enabled then stripslashes will be needed
 if (get_magic_quotes_gpc() == 1)
 {    $query = stripslashes($query);   }
```

returning a web page

test for a query

this is the Solr client library

Solr runs on port 8983

handles quoting of special characters in query

PhP Program (2 of 3)

```
 try
 {    $results = $solr->search($query, 0, $limit);   }
 catch (Exception $e)
 {  die("<html><head><title>SEARCH EXCEPTION</title><body><pre>{$e->__toString()}</pre></body></html>");    }     }
?>
<html>   <head>   <title>PHP Solr Client Example</title>   </head>   <body>
 <form accept-charset="utf-8" method="get">
 <label for="q">Search:</label>
 <input id="q" name="q" type="text" value="<?php echo htmlspecialchars($query, ENT_QUOTES, 'utf-8'); ?>"/>
 <input type="submit"/>
 </form>
<?php
// display results
if ($results)
{  $total = (int) $results->response->numFound; $start = min(1, $total); $end = min($limit, $total);
?>
 <div>Results <?php echo $start; ?> - <?php echo $end;?> of <?php echo $total; ?>:</div>
 <ol>
```

send query to Solr

catch any exception

create web page output

create input text box

create submit button

end form

JSON result string

PhP Program (3 of 3)

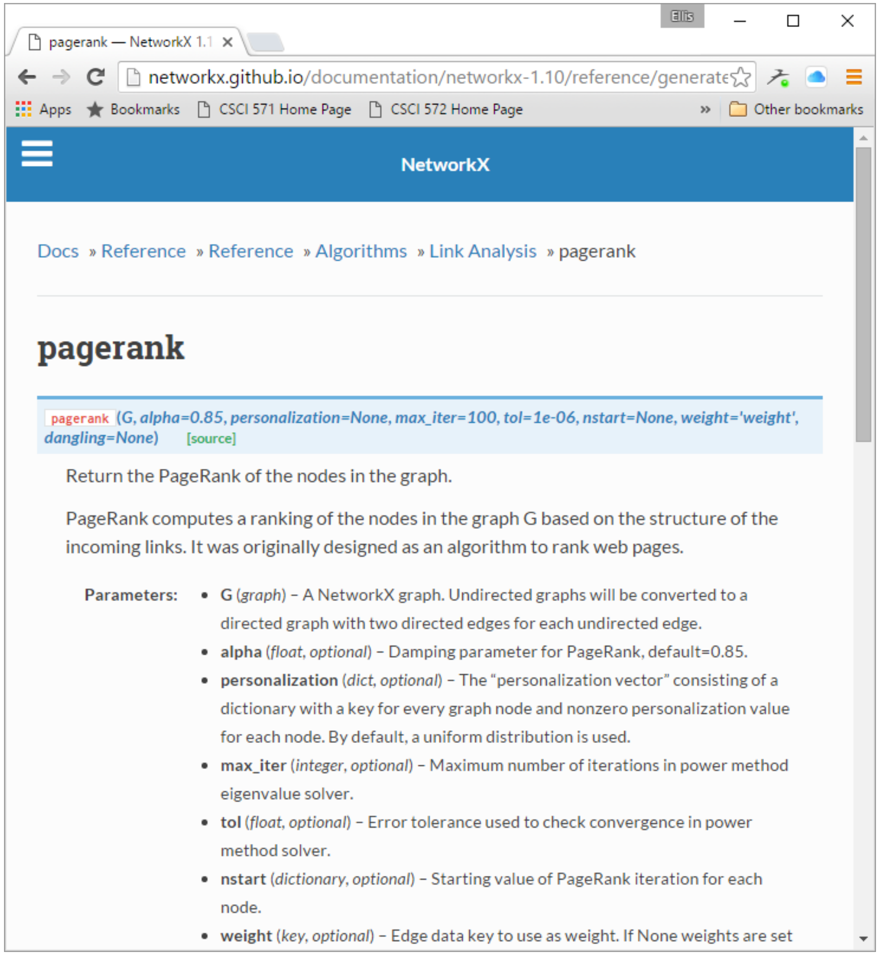
```
<?php
 // iterate result documents
 foreach ($results->response->docs as $doc)
 {   ?>   <li>
 <table style="border: 1px solid black; text-align: left">
<?php
 // iterate document fields / values
 foreach ($doc as $field => $value)
 {    ?>
 <tr>
 <th><?php echo htmlspecialchars($field, ENT_NOQUOTES, 'utf-8'); ?></th>
 <td><?php echo htmlspecialchars($value, ENT_NOQUOTES, 'utf-8'); ?></td>
 </tr>
<?php    }
?> </table>   </li>
<?php    }    ?>
 </ol>
<?php
}   ?> </body>  </html>
```

output consists of a set of field, value pairs

Comparing Ranking Algorithms

- we have already seen the built-in Solr ranking method
  - see slide 9
- Solr permits alternative ranking algorithms
  - we will use Page Rank as contained in an External file

http://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.algorithms.link_analysis.pagerank_alg.pagerank.html



Docs » Reference » Reference » Algorithms » Link Analysis » pagerank

pagerank

`pagerank` *(G, alpha=0.85, personalization=None, max_iter=100, tol=1e-06, nstart=None, weight='weight', dangling=None)* [source]

Return the PageRank of the nodes in the graph.

PageRank computes a ranking of the nodes in the graph G based on the structure of the incoming links. It was originally designed as an algorithm to rank web pages.

Parameters:

- **G** (*graph*) – A NetworkX graph. Undirected graphs will be converted to a directed graph with two directed edges for each undirected edge.
- **alpha** (*float, optional*) – Damping parameter for PageRank, default=0.85.
- **personalization** (*dict, optional*) – The "personalization vector" consisting of a dictionary with a key for every graph node and nonzero personalization value for each node. By default, a uniform distribution is used.
- **max_iter** (*integer, optional*) – Maximum number of iterations in power method eigenvalue solver.
- **tol** (*float, optional*) – Error tolerance used to check convergence in power method solver.
- **nstart** (*dictionary, optional*) – Starting value of PageRank iteration for each node.
- **weight** (*key, optional*) – Edge data key to use as weight. If None weights are set

- You are going to use an open source PageRank algorithm, located at URL above;
- You should have already stored your downloaded documents with all outgoing links in a file;
- You need to create a graph that the PageRank algorithm can work on

Important Parameters:

- a NetworkX graph
- a damping parameter (e.g. 0.85)
- maximum number of iterations
- error tolerance
- starting Page Rank value of nodes

Final Steps

- Input to the PageRank algorithm is a file containing every document ID and associated with each ID, the IDs that are pointed to by links withing the document ID
- Output from the PageRank algorithm is a file containing every document ID and its associated PageRank
- place this file is solr-5.3.1/server/solr/core_name, call the file external_pageRankFile.txt
- add the PageRank field to the schema.xml file

```
<fieldType name="external" keyField="id" defVal="0" class="solr.ExternalFileField" valtype="pfloat" />
<field name="pageRankFile" type="external" stored="false" indexed="false" />
```

- Once both ranking algorithms are working you should input the same queries as Exercise #1 and compare the results