Big Data Search

Kaushik Dutta
Information Systems and Decision Sciences
Muma College of Business
University of South Florida

Search Systems

- Lucene
- Solr (On top of Lucene)
- Elastic Search (AWS)

Apache Lucene

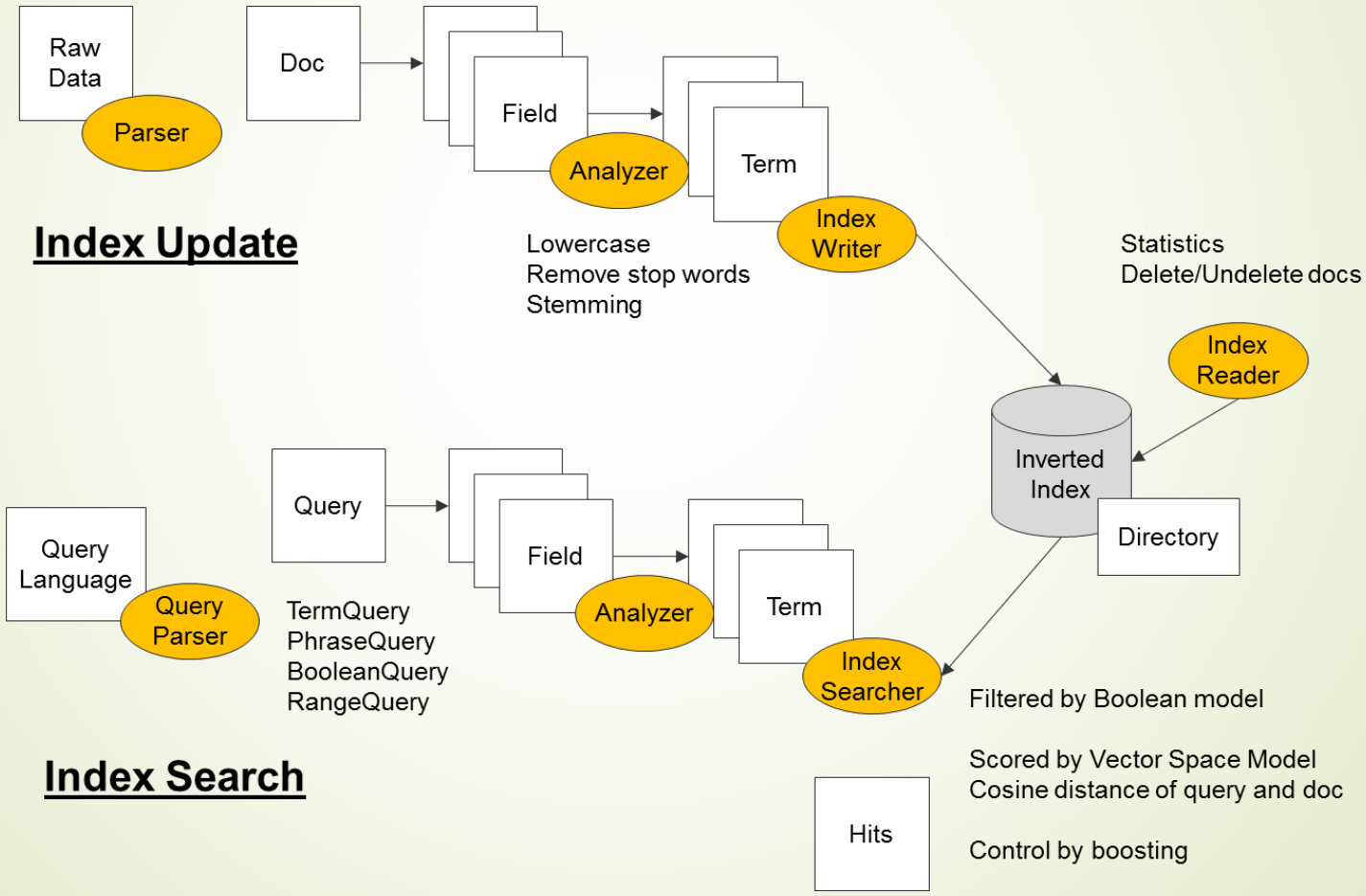
- Information Retrieval library
- Open source
- Initially developed by Doug Cutting (Also author of Hadoop)
- Indexing and Searching
- Inverted Index of documents
- High performance, scalable
- Provides advanced Search options like synonyms, stopwords, based on similarity, proximity.

Lucene

- Lucene Core, 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 JSON/Python/Ruby APIs, hit highlighting, faceted search, caching, replication, and a web admin interface.
- Open Relevance Project is a subproject with the aim of collecting and distributing free materials for relevance testing and performance.
- PyLucene is a Python port of the Core project.

Lucene Text Processing

- There are two main processing flow in the system ...
  - Document indexing: Given a document, add it into the index
  - Document retrieval: Given a query, retrieve the most relevant documents from the index.

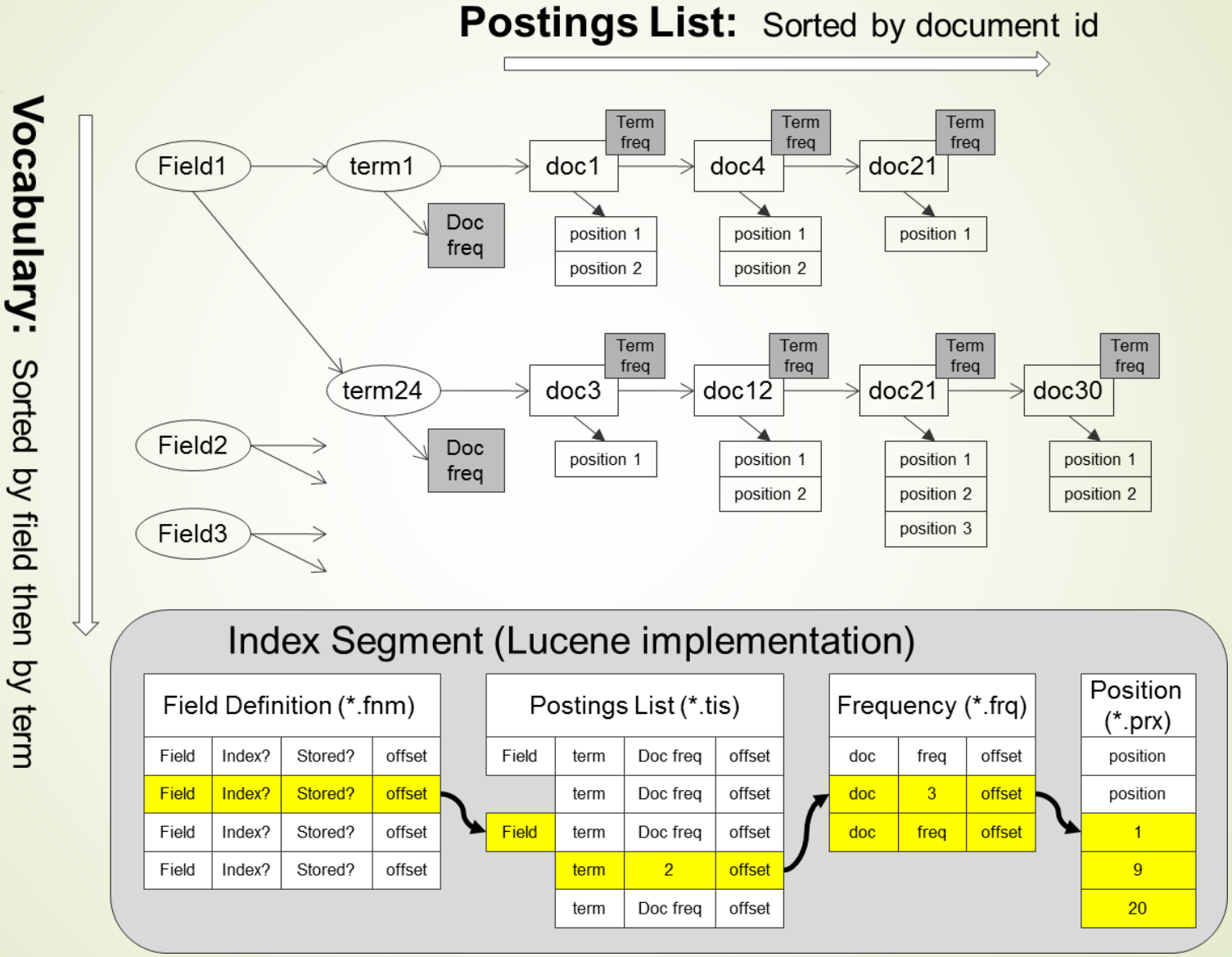


http://horicky.blogspot.com/2013/02/text-processing-part-2-inverted-index.html

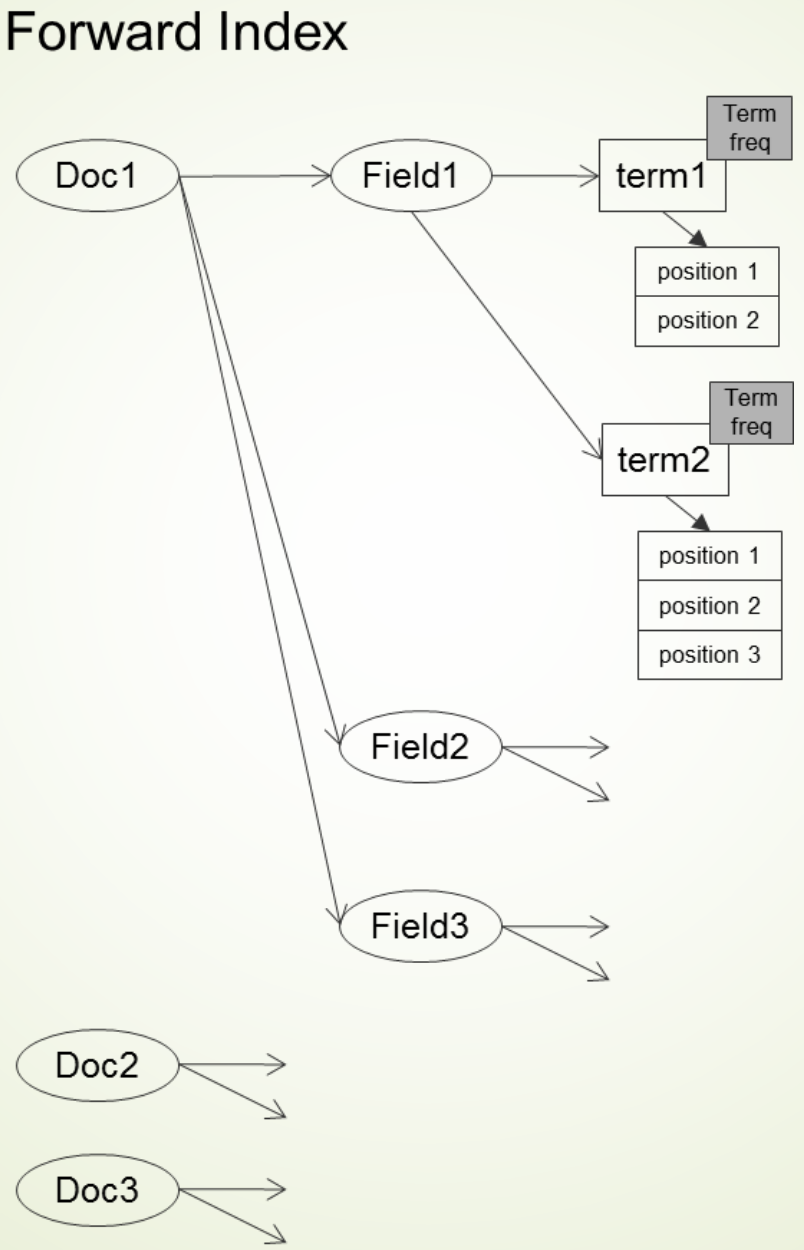
Lucene Index Structure

- Both documents and query is represented as a bag of words.
  - "Document" is the basic unit for storage and retrieval.
    - A "Document" contains multiple "Fields" (also call zones).
    - Each "Field" contains multiple "Terms" (equivalent to words).
- Document Indexing
  - Keyword (Not analyzed, Indexed, Stored)
  - Unindexed (Not analyzed, Not indexed, Stored)
  - Unstored (Analyzed, Indexed, Not stored)
  - Text (Analyzed, Indexed, Stored)
- Inverted Index
  - It is organized as an inverted manner from terms to the list of documents (which contain the term).

Lucene – Inverted Index



Lucene Forward Index



Steps of Document Indexing

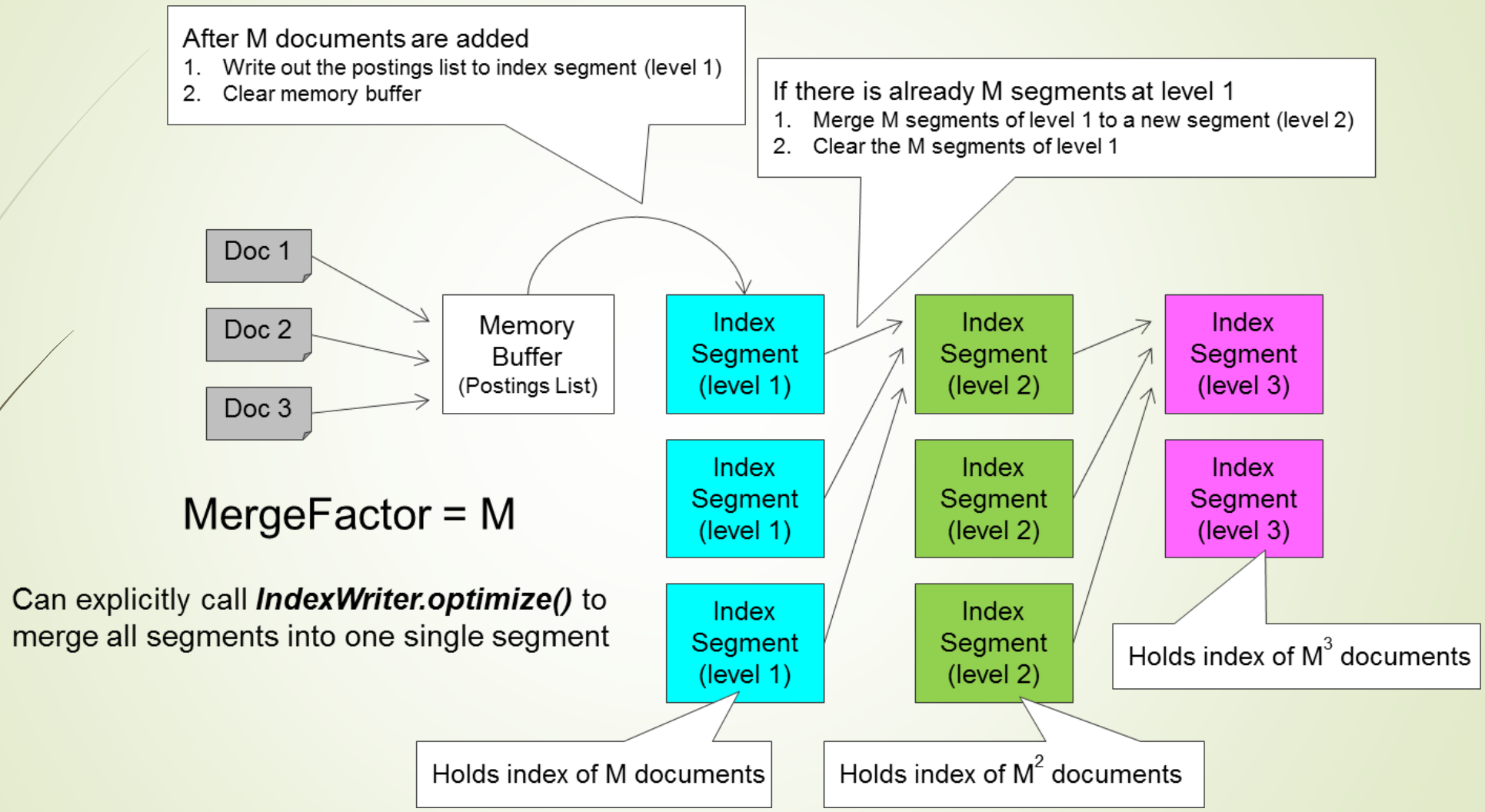
- Tokenize the document (breakdown into words)
- Lowercase each word (to make it non-case-sensitive, but need to be careful with names or abbreviations)
- Remove stop words (take out high frequency words like "the", "a", but need to careful with phrases)
- Stemming (normalize different form of the same word, e.g. reduce "run", "running", "ran" into "run")
- Synonym handling. This can be done in two ways. Either expand the term to include its synonyms (ie: if the term is "huge", add "gigantic" and "big"), or reduce the term to a normalized synonym (ie: if the term is "gigantic" or "huge", change it to "big")

- doc = [term1, term2 ...].
  - Optionally, terms can be further combined into n-grams.
  - For example, in a bi-gram expansion, the document will become ...
    doc1 -> {term1: 5, term2: 8, term3: 4, term1_2: 3, term2_3:1}

Document Addition in the Index

- **Insert**: When this is a document insertion, it goes through the normal indexing process (as I described above) to analyze the document and build an inverted list in RAM.
- **Delete**: When this is a document deletion (the client request only contains the doc id), it fetches the forward index to extract the document content, then goes through the normal indexing process to analyze the document and build the inverted list. But in this case the doc object in the inverted list is labeled as "deleted".
- **Update**: When this is a document update (the client request contains the modified document), it is handled as a deletion followed by an insertion, which means the system first fetch the old document from the forward index to build an inverted list with nodes marked "deleted", and then build a new inverted list from the modified document. (e.g. If doc1 = "A B" is update to "A C", then the posting list will be {A:doc1(deleted) -> doc1, B:doc1(deleted), C:doc1}. After collapsing A, the posting list will be {A:doc1, B:doc1(deleted), C:doc1}

Lucene Index Structure



Document Retrieval

- Given a query Q containing terms [t1, t2]
  - For each term t1, t2 in query, we identify all the corresponding posting lists.
  - We walk each posting list concurrently to return a sequence of documents (ordered by doc id). Notice that each return document contains at least one term but can also also contain multiple terms.
  - We compute the dynamic score which is dot product of the query to document vector.
  - We also look up the static score which is purely based on the document (but not the query). The total score is a linear combination of static and dynamic score.
  - Although the score we used in above calculation is based on computing the cosine distance between the query and document, we are not restricted to that. We can plug in any similarity function that make sense to the domain. (e.g. we can use machine learning to train a model to score the similarity between a query and a document).
  - After we compute a total score, we insert the document into a heap data structure where the top K scored document is maintained.

http://horicky.blogspot.com/2013/02/text-processing-part-2-inverted-index.html

Basic Concepts

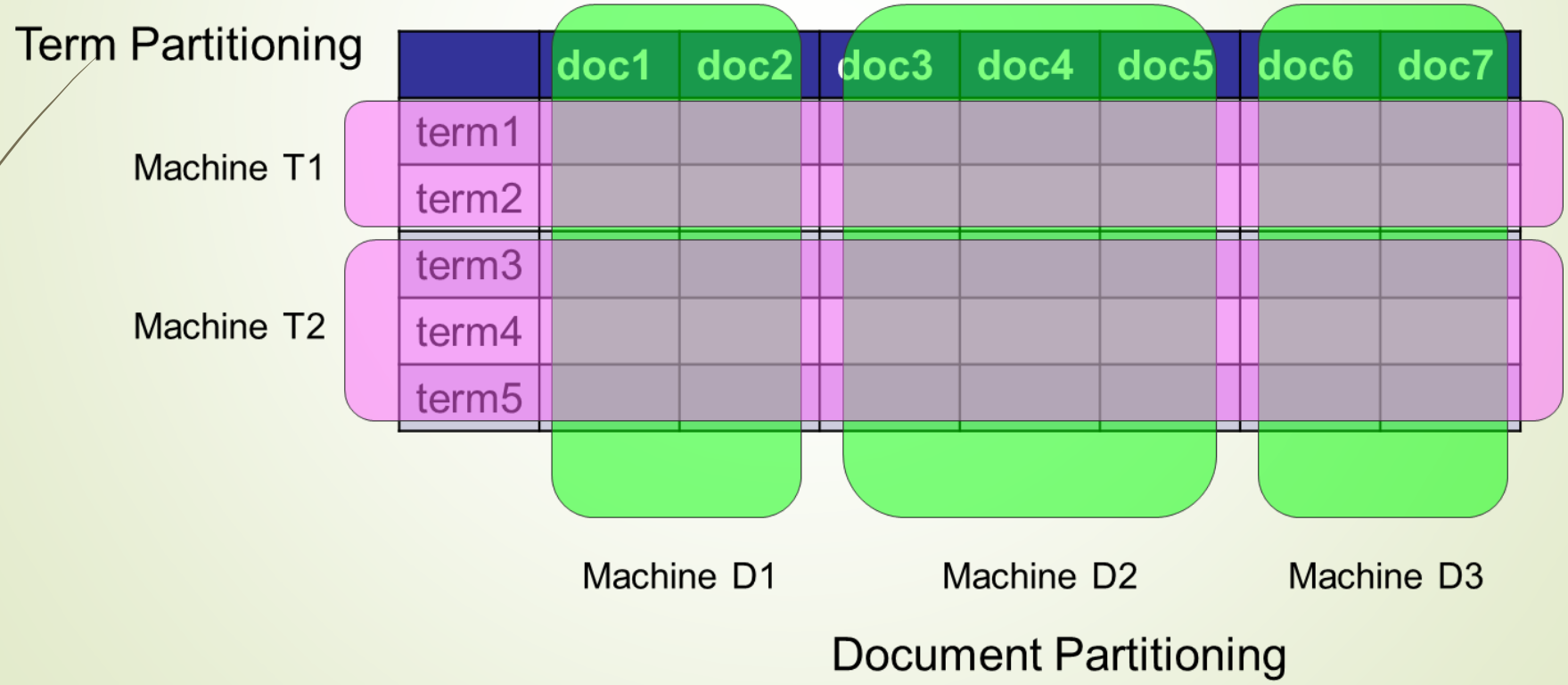
- **tf** (t in d) : term frequency in a document
  - measure of how often a term appears in the document
  - the number of times term *t* appears in the currently scored document *d*
- **idf** (t) : inverse document frequency
  - measure of whether the term is common or rare across all documents, i.e. how often the term appears across the index
  - obtained by dividing the total number of documents by the number of documents containing the term, and then taking the logarithm of that quotient.
- **coord** : coordinate-level matching
  - number of terms in the query that were found in the document,
  - e.g. term 'x' and 'y' found in doc1 but only term 'x' is found in doc2 so for a query of 'x' **OR** 'y' doc1 will receive a higher score.
- **boost (index)** : boost of the field at index-time
- **boost (query)** : boost of the field at query-time

$$\text{TF-IDF}_{(n,d)} = \text{TF}_{(n,d)} \times \text{IDF}_{(n)}$$

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

Distributed Index – Elastic Search

- Lucene is Single machine system
  - Distributed version is Elastic Search
- Term partitioning
- Document partitioning



Apache Solr

- Initially Developed by Yonik Seeley
- Enterprise Search platform for Apache Lucene
- Open source
- Highly reliable, scalable, fault tolerant
- Support distributed Indexing (SolrCloud), Replication, and load balanced querying

Solr

- Advanced full-text search
- Near real-time indexing
- Standards-based open interfaces like XML, JSON and HTTP
- Comprehensive HTML administration interfaces
- Server statistics exposed over JMX for monitoring
- Linearly scalable, auto index replication, auto failover and recovery
- Flexible and adaptable, with XML configuration

What is SOLR

- Open source enterprise search platform based on Apache Lucene project.
- REST-like HTTP/XML and JSON APIs
- Powerful full-text search, hit highlighting, faceted search
- Database integration, and rich document (e.g., Word, PDF) handling
- Dynamic clustering, distributed search and index replication
- Loose Schema to define types and fields
- Written in Java5, deployable as a WAR

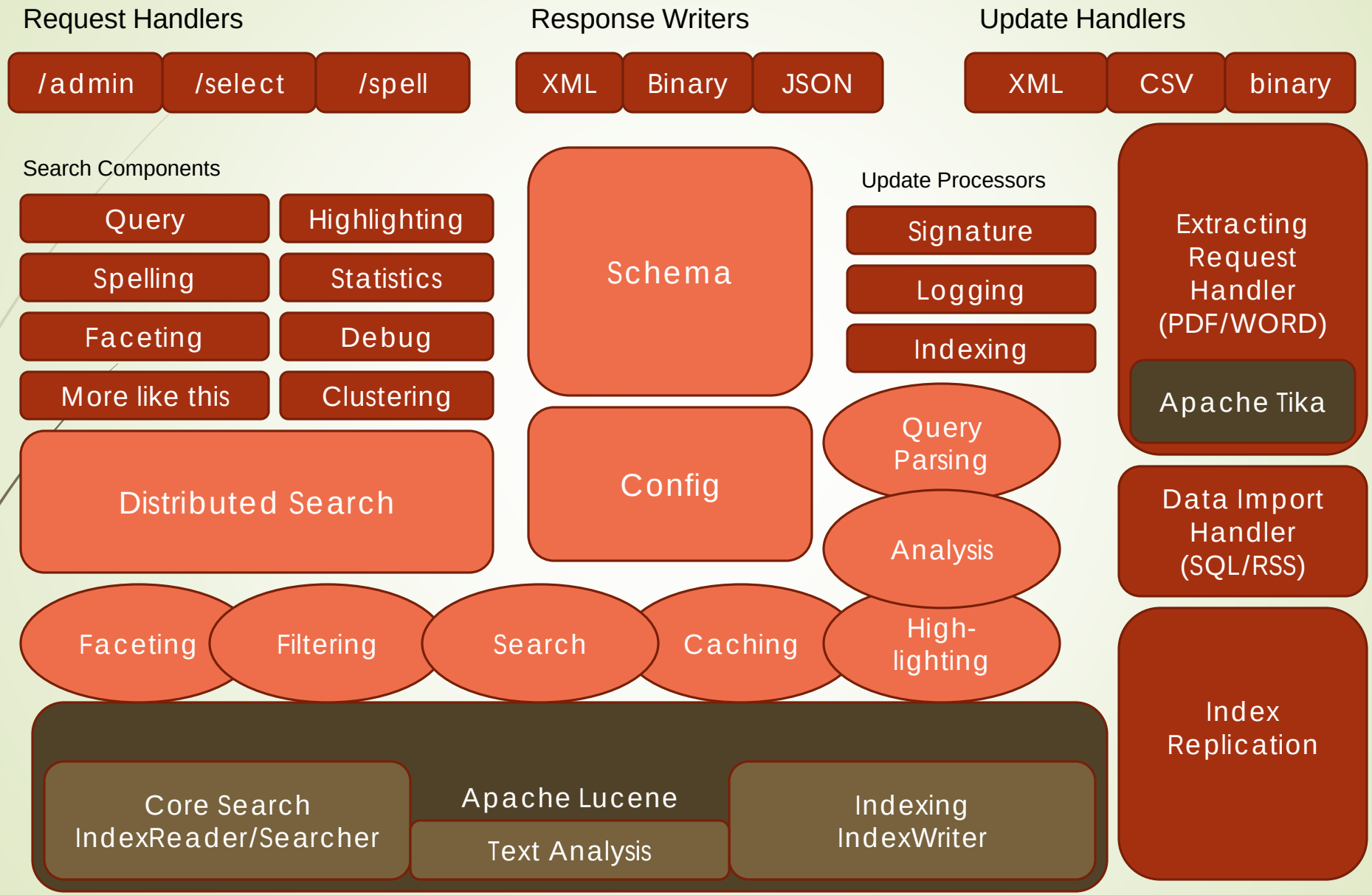
Apache Solr - Features

- full-text search
- hit highlighting
- faceted search (similar to GroupBy clause in RDBMS)
- near real-time indexing
- dynamic clustering (e.g. Cluster of most frequent words, tagCloud)
- database integration
- rich document (e.g., Word, PDF) handling
- geospatial search

Solr Use Cases

- **Instagram**: Instagram (a Facebook company) is one of the famous sites, and it uses Solr to power its geosearch API
- WhiteHouse.gov: The Obama administration's website is inbuilt in Drupal and Solr
- **Netflix**: Solr powers basic movie searching on this extremely busy site
- **Internet archive**: Search this vast repository of music, documents, and video using Solr
- StubHub.com: This ticket reseller uses Solr to help visitors search for concerts and sporting events.
- **The Smithsonian Institution**: Search the Smithsonian's collection of over 4 million items

Lucene/Solr Architecture



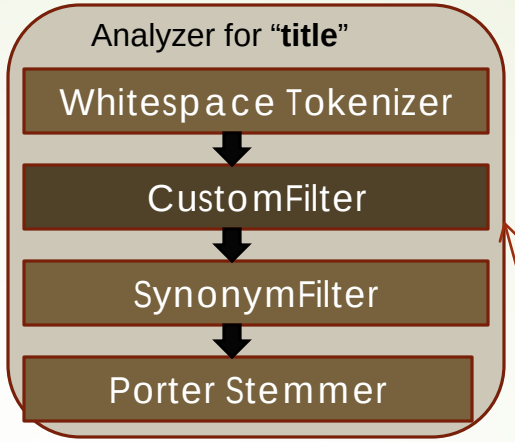
Lucene/Solr plugins

- RequestHandlers – handle a request at a URL like /select
- SearchComponents – part of a SearchHandler, a componentized request handler
  - Includes, Query, Facet, Highlight, Debug, Stats
  - Distributed Search capable
- UpdateHandlers – handle an indexing request
- Update Processor Chains – per-handler componentized chain that handle updates
- Query Parser plugins
  - Mix and match query types in a single request
  - Function plugins for Function Query
- Text Analysis plugins: Analyzers, Tokenizers, TokenFilters
- ResponseWriters serialize & stream response to client

Lucene/Solr Query Plugin Architecture

Declarative Analysis per-field
- Tokenizer to split text
- TokenFilter to transform tokens
- Analyzer for completely custom
- Separate query / index analyzer

QParser plugins
- Support different query syntaxes
- Support different query execution
- Function Query supports pluggable custom functions
- Excellent support for nesting/mixing different query types in the same request.



schema.xml

```
// declaratively defines  types
// and analyzers for fields
<fieldType name="text1">
   <filter="whitespace">
   <filter="customFilter" …>
   <filter="synonyms" file=..>
   <filter="porter" except=..>

<field name="title" type="text1"

<field name="cust1" class=…
```

Analyzer for "**cust1**" (potentially completely custom architecture not using tokenizer/filters)

solrconfig.xml

```
< index configuration />
< caching configuration />
< request handler config />
< search component config />
< update processor config />
< misc –  HTTP cache, JMX >

<parser name="mycustom" …

<func name="custom" class=…
```

Lucene QParser

XML QParser

DisMax QParser

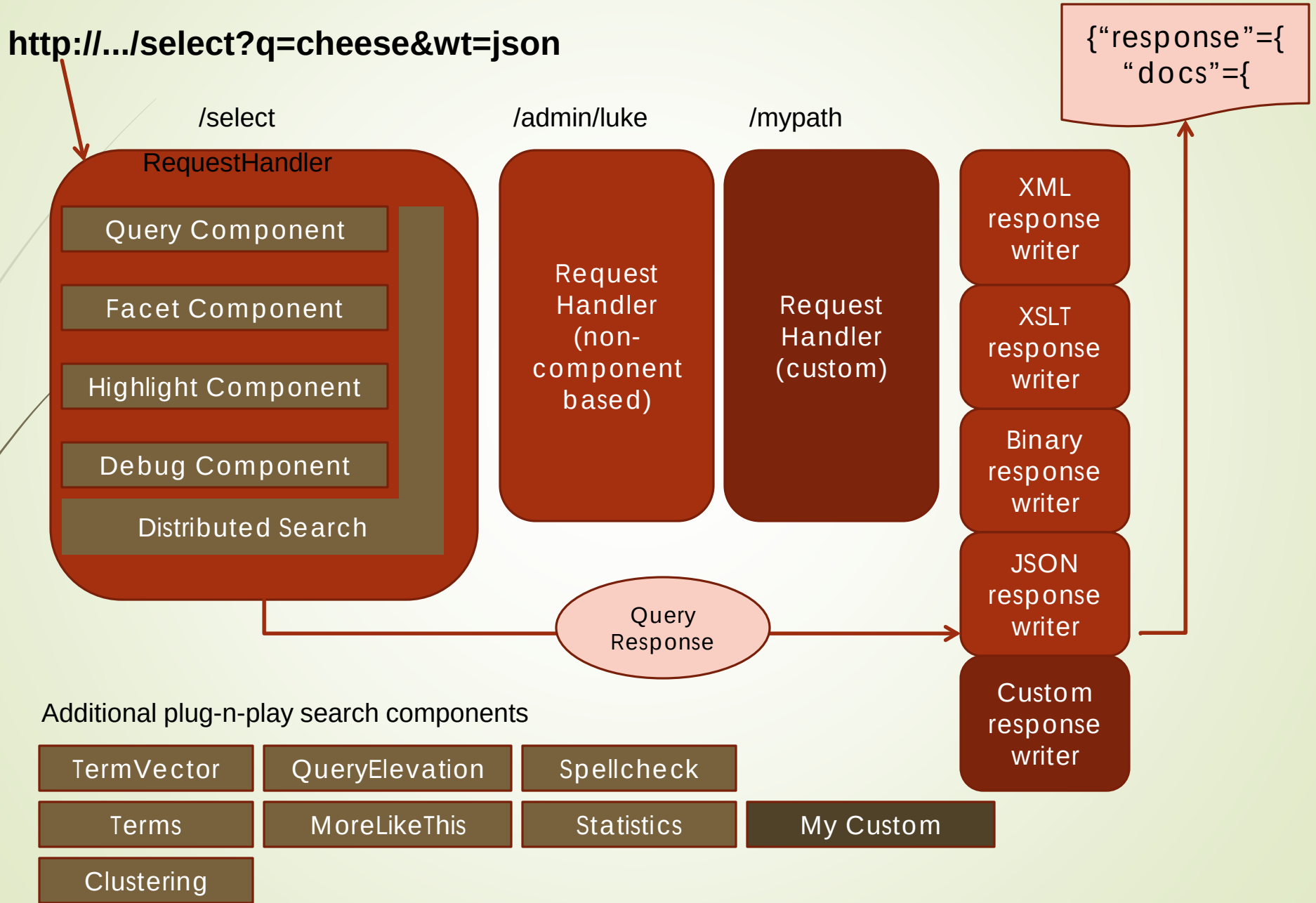
Function Range Q

MyCustom QParser

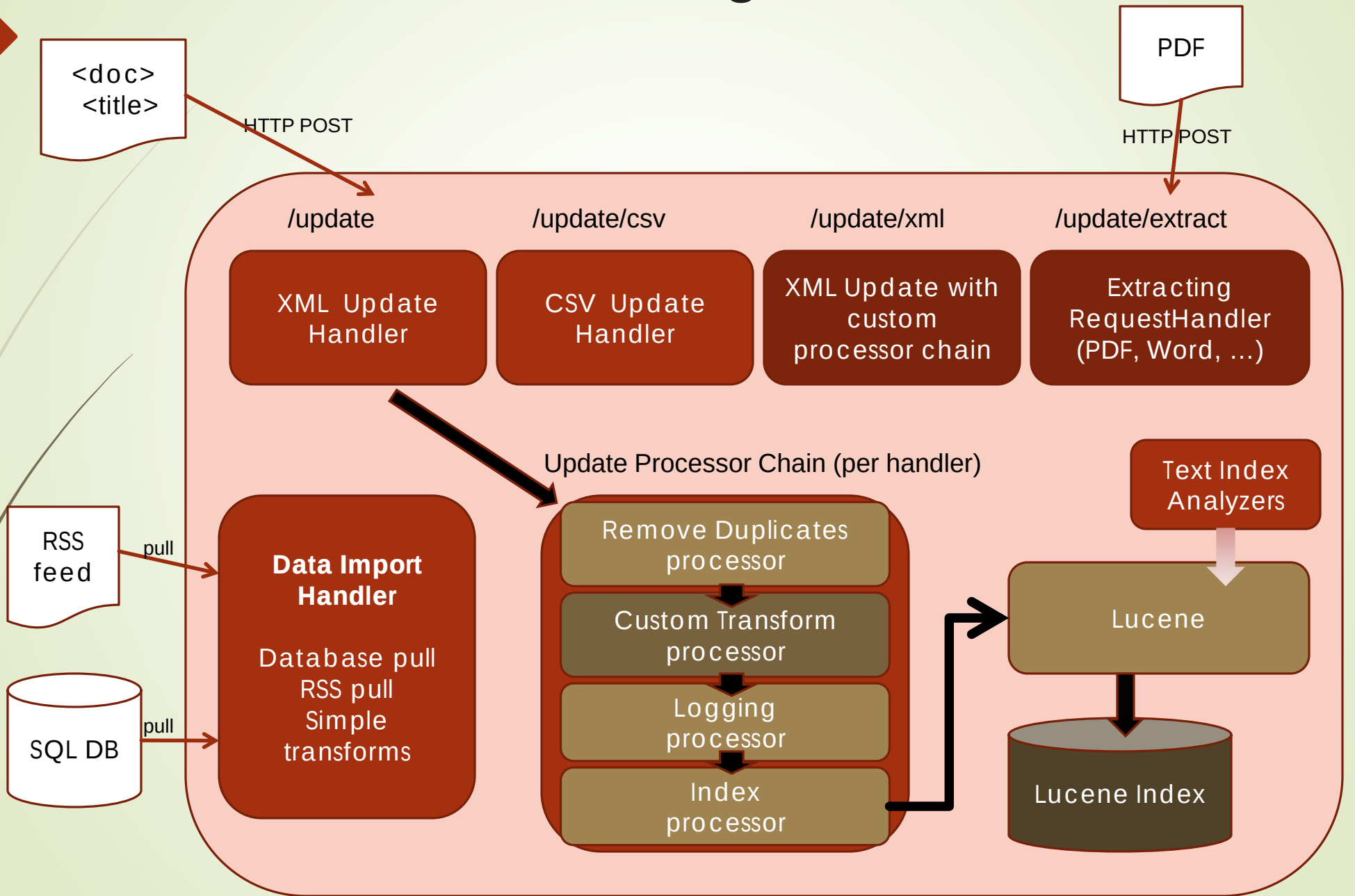
Function QParser

sum | max | pow | log | sqrt | custom

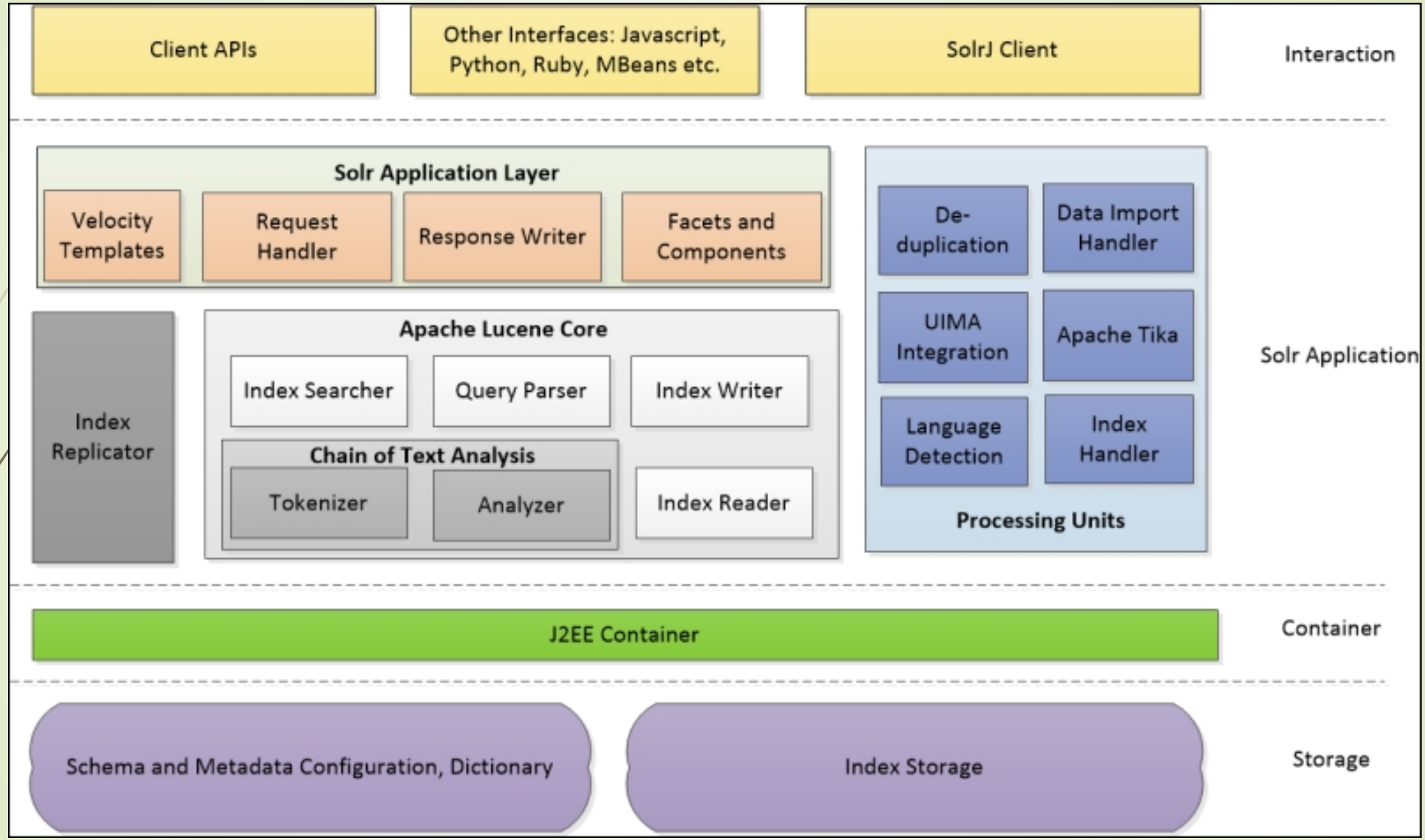
Lucene/Solr Request Plugins



Lucene/Solr Indexing



Solr Architecture



How Solr Sees the World

- An index is built of one or more Documents
- A Document consists of one or more Fields
- Documents are composed of fields
- A Field consists of a name, content, and metadata telling Solr how to handle the content.
- You can tell Solr about the kind of data a field contains by specifying its field type

Solr – schema.xml

- Types with index and query Analyzers - similar to data type
- Fields with name, type and options
- Unique Key
- Dynamic Fields
- Copy Fields

Solr – Content Analysis

- Defines documents Model
- Index contains documents.
- Each document consists of fields.
- Each Field has attributes.
  - What is the data type (FieldType)
  - How to handle the content (Analyzers, Filters)
  - Is it a stored field (stored="true") or Index field (indexed="true")

Solr – Content Analysis

- Field Attributes
  - Name : Name of the field
  - Type : Data-type (FieldType) of the field
  - Indexed : Should it be indexed (indexed="true/false")
  - Stored : Should it be stored (stored="true/false")
  - Required : is it a mandatory field (required="true/false")
  - Multi-Valued : Would it will contains multiple values e.g. text: pizza, food (multiValued="true/false")

e.g. <field name="id" type="string" indexed="true" stored="true" required="true" multiValued="false" />

Solr – Content Analysis

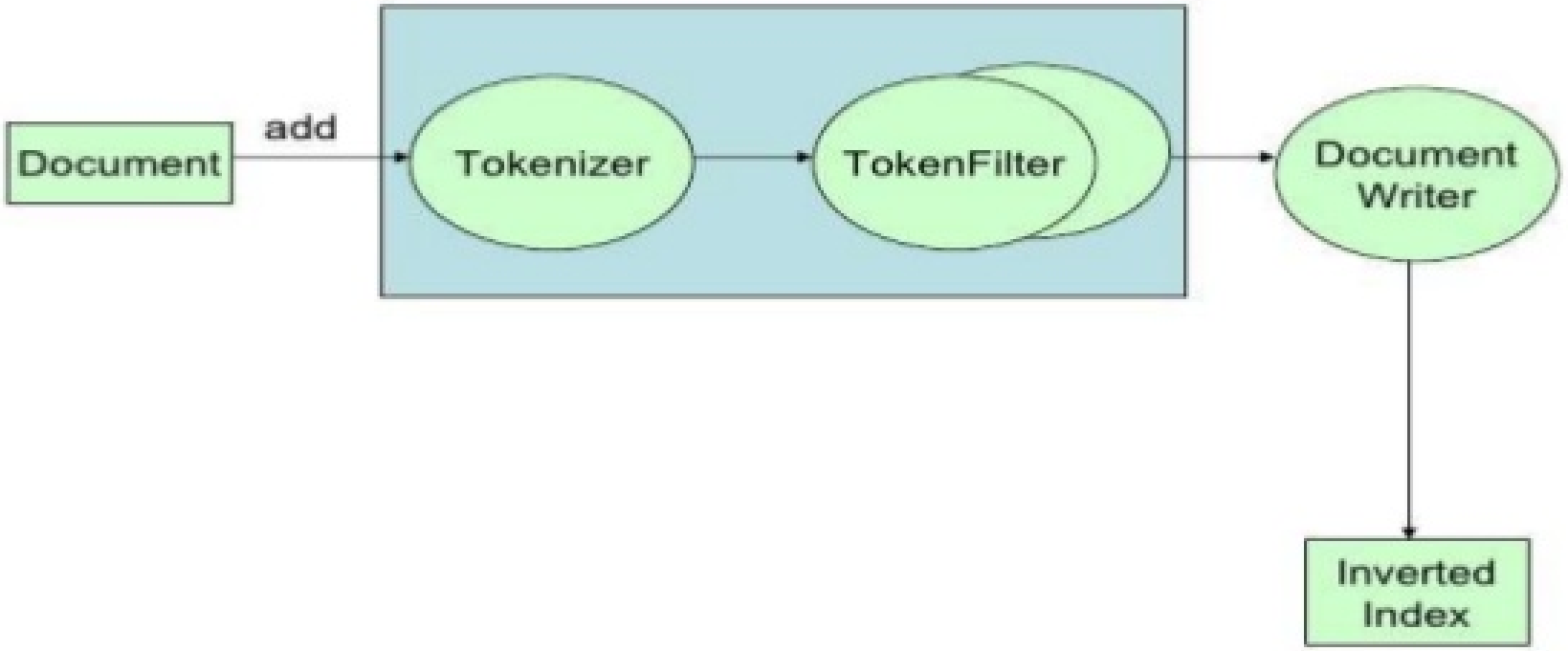
- FieldType can be
  - StrField : String Field
  - TextField : Similar to StrField but can be analyzed
  - BoolField : Boolean Field
  - IntField : Integer Field
  - Trie Based
    - TrieIntField
    - TrieLongField
    - TrieDateField
    - TrieDoubleField
  - Few more….

e.g.

```
<fieldType name="string" class="solr.StrField" sortMissingLast="true" omitNorms="true"/>
<fieldType name="boolean" class="solr.BoolField" sortMissingLast="true" omitNorms="true"/>
<fieldType name="tint" class="solr.TrieIntField" precisionStep="8" positionIncrementGap="0" omitNorms="true"/>
<fieldType name="tfloat" class="solr.TrieFloatField" precisionStep="8" positionIncrementGap="0" omitNorms="true"/>
<fieldType name="tlong" class="solr.TrieLongField" precisionStep="8" positionIncrementGap="0" omitNorms="true"/>
<fieldType name="tdouble" class="solr.TrieDoubleField" precisionStep="8" positionIncrementGap="0" omitNorms="true"/>
<fieldType name="tdate" class="solr.TrieDateField" precisionStep="6" positionIncrementGap="0" omitNorms="true"/>
```

Check for more Field Types @ https://cwiki.apache.org/confluence/display/solr/Field+Types+Included+with+Solr

Indexing Pipeline



- Analyzer : create tokens using a Tokenizer and/or applying Filters (Token Filters)
- Each field can define an Analyzer at index time/query time or the both at same time.

Credit : http://www.slideshare.net/otisg/lucene-introduction

Solr – Content Analysis

- Commonly used tokenizers:
  - StandardTokenizerFactory
  - WhitespaceTokenizerFactory
  - KeywordTokenizerFactory
  - LowerCaseTokenizerFactory
  - PatternTokenizerFactory
  - LetterTokenizerFactory
  - ClassicTokenizerFactory
  - UAX29URLEmailTokenizerFactory

Solr – Content Analysis

- Commonly used filters:
  - ClassicFilterFactory
  - LowerCaseFilterFactory
  - CommonGramsFilterFactory
  - EdgeNGramFilterFactory
  - TrimFilterFactory
  - StopFilterFactory
  - TypeTokenFilterFactory
  - PatternCaptureGroupFilterFactory
  - PatternReplaceFilterFactory

Field Analysis

- 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. Analyzers may be a single class or they may be composed of a series of tokenizer and filter classes.
- Tokenizers break field data into lexical units, or tokens
- Example:
  - Setting all letters to lowercase
  - Eliminating punctuation and accents, mapping words to their stems, and so on
  - "ram", "Ram" and "RAM" would all match a query for "ram"

Filter explanation

- StopFilterFactory: Tokenize on whitespace, then removed any common words
- WordDelimiterFilterFactory: Handle special cases with dashes, case transitions, etc.
- LowerCaseFilterFactory: lowercase all terms.
- EnglishPorterFilterFactory: Stem using the Porter Stemming algorithm.

E.g: “runs, running, ran” ➔ its elemental root "run"

- RemoveDuplicatesTokenFilterFactory: Remove any duplicates:

Schema.xml

- schema.xml file located in ../solr/conf
- schema file starts with <schema> tag
- Solr supports one schema per deployment
- The schema can be organized into three sections:
  - Types
  - Fields
  - Other declarations

Field Attributes

- Indexed:
  - Indexed Fields are searchable and sortable.
  - You also can run Solr 's analysis process on indexed Fields, which can alter the content to improve or change results.
- Stored:
  - The contents of a stored Field are saved in the index.
  - This is useful for retrieving and highlighting the contents for display but is not necessary for the actual search.
  - For example, many applications store pointers to the location of contents rather than the actual contents of a file.

Indexing data

- Using curl to interact with Solr: http://curl.haxx.se/download.html
- Here are different data formats:
  - Solr's native XML
  - CSV (Character Separated Value)
  - Rich documents through Solr Cell
  - JSON format
  - Direct Database and XML Import through Solr's *DataImportHandler*

Add / Update documents

HTTP POST to add / 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>
```

Delete documents

- Delete by Id

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

- Delete by Query (multiple documents)

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

Commit / Optimize

- <commit/> tells Solr that all changes made since the last commit should be made available for searching.
- <optimize/> same as commit.
  - Merges all index segments. Restructures Lucene 's files to improve performance for searching.
  - Optimization is generally good to do when indexing has completed
  - If there are frequent updates, you should schedule optimization for low-usage times
  - An index does not need to be optimized to work properly. Optimization can be a time-consuming process.

Index XML documents

- Use the command line tool for POSTing raw XML to a Solr
- Other options:
  - -Ddata=[files|args|stdin]
  - -Durl=http://localhost:8983/solr/update
  - -Dcommit=yes

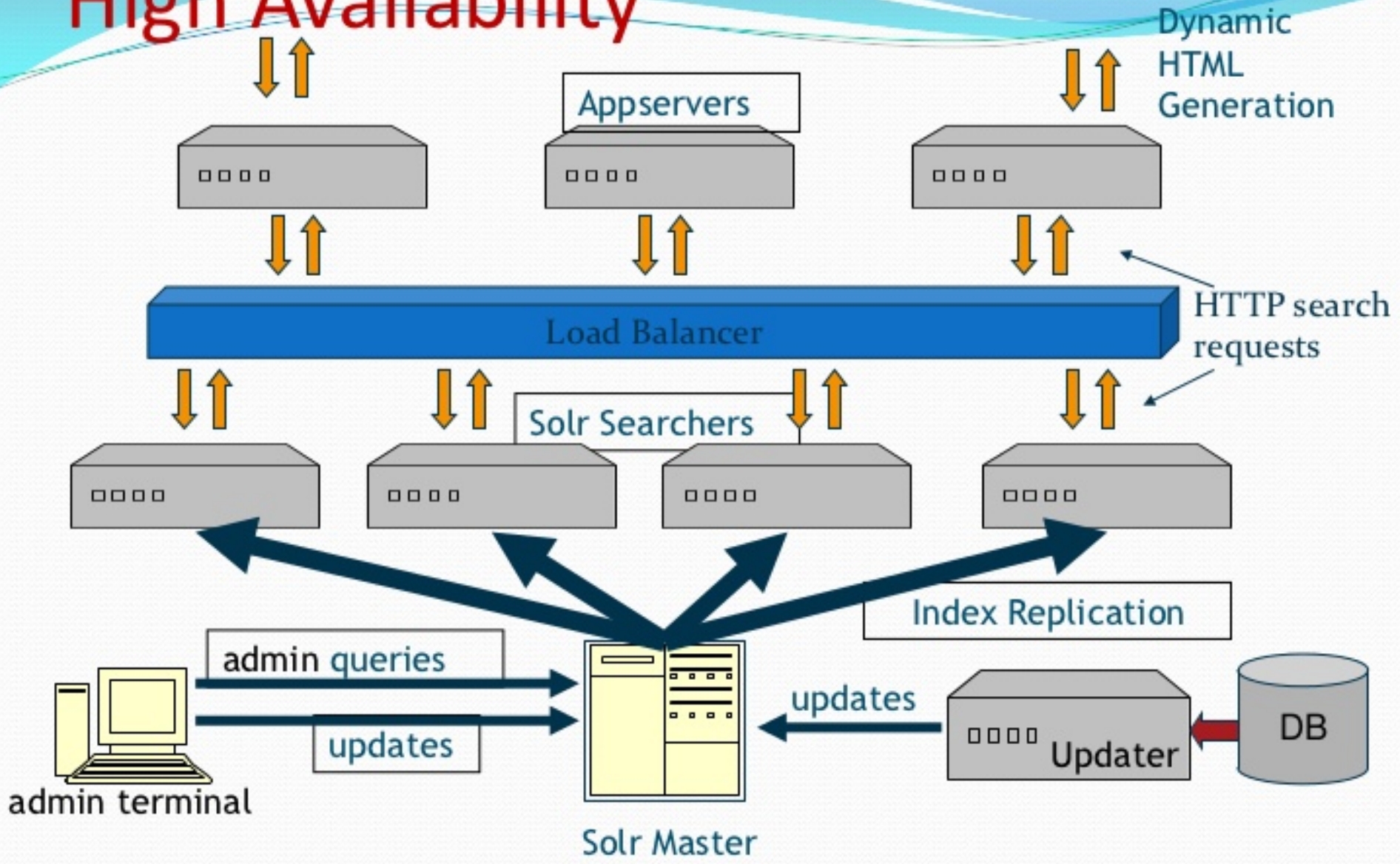
(Option default values are in red)

- Example:
  - java -jar post.jar *.xml
  - java -Ddata=args -jar post.jar "<delete><id>42</id></delete>"
  - java -Ddata=stdin -jar post.jar
  - java -Dcommit=no -Ddata=args -jar post.jar "<delete><query>*:*</query></delete>"

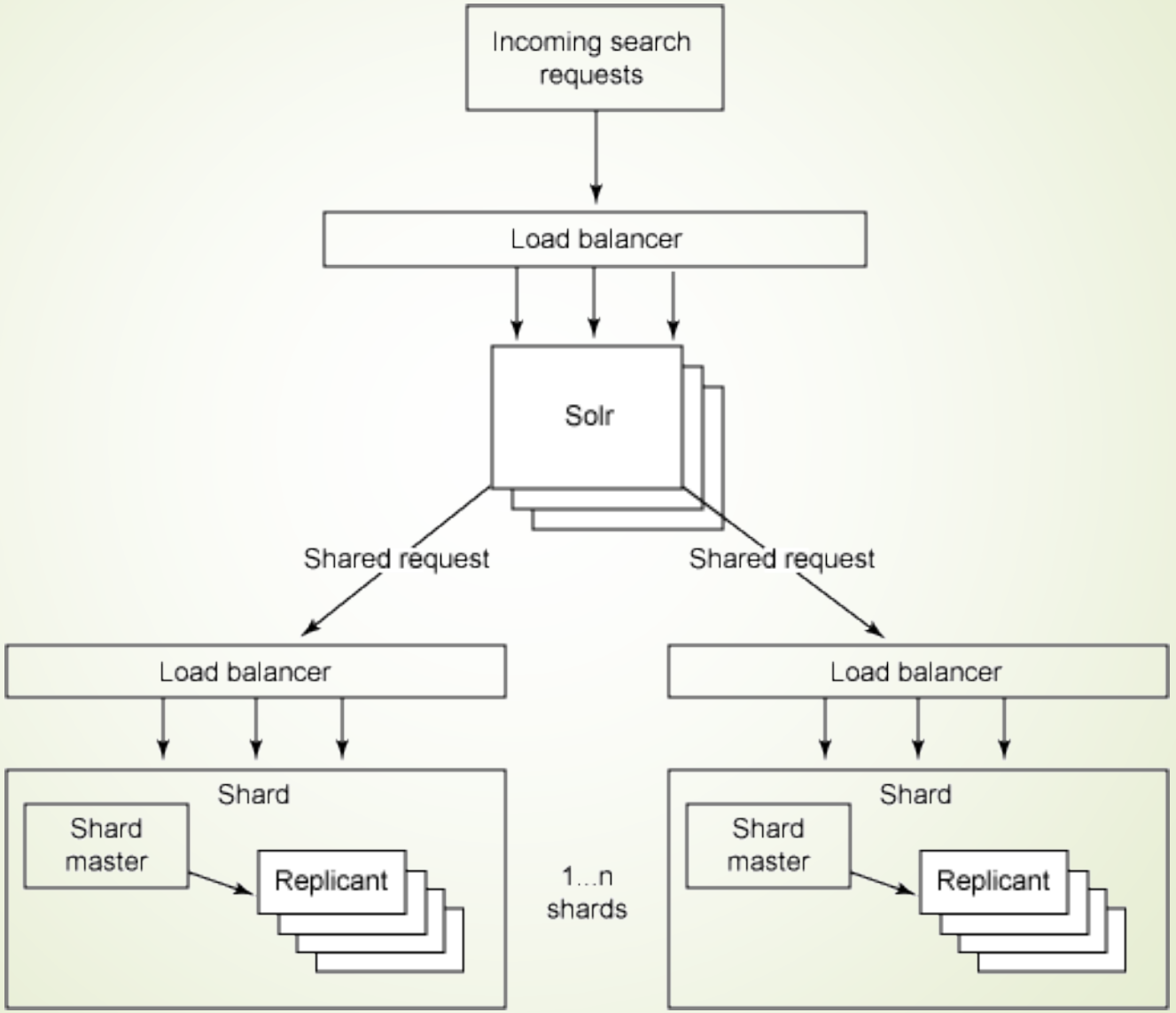
Index CSV file using HTTP POST

- *curl* command does this with *data-binary* and an appropriate content-type header reflecting that it's XML.
- Example: using HTTP-POST to send the CSV data over the network to the Solr server:
  - curl http://localhost:9090/solr/update -H "Content-type:text/xml;charset=utf-8" --data-binary @ipod_other.xml

High Availability



Distributed and Replicated Solr Architecture



Snippet of SolrBaseDAO - Contains code to get instance of Solr server

```
public HttpSolrServer getSolrConnection() throws Exception {
HttpSolrServer solrServer = null;
try {
// configure a server object with actual solr values.
    if (solrServer == null) {
solrServer = new HttpSolrServer("http://localhost:8080/solr");
solrServer.setParser(new XMLResponseParser());
solrServer.setSoTimeout(5000);
solrServer.setConnectionTimeout(5000);
     }
} catch (Exception exc) {
     logger.error(" Exception in getting Solr Connection: "
+ exc.getMessage());
  exc.printStackTrace();
}
return solrServer;
}
```

Snippet of SolrDataDAO.java – query data.

```
/**
 * Get Data from Solr
 *
 * @param start
 * @param rows
 * @param query
 * @return
 * @throws Exception
 */
public QueryResponse queryData(int start, int rows, String query)
throws Exception {
ModifiableSolrParams solrParams = new ModifiableSolrParams();
solrParams.set("q", query);
solrParams.set("start", start);
solrParams.set("rows", rows);
return server.query(solrParams);
}
```

Snippet of SolrDataDAO.java – add data.

```
/**
 * Add data into Solr
 *
 * @param id
 * @param name
 * @throws Exception
 */
public void addData(int id, String name) throws Exception {
SolrInputDocument doc = new SolrInputDocument();
doc.addField("id", id);
doc.addField("name", name);
server.add(doc);
server.commit();
}
```

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 Query

- **Keyword Matching**
  - Search for word "foo" in the title field.
    - title:foo
  - Search for phrase "foo bar" in the title field.
    - title:"foo bar"
  - Search for phrase "foo bar" in the title field AND the phrase "quick fox" in the body field.
    - title:"foo bar" AND body:"quick fox"
  - Search for either the phrase "foo bar" in the title field AND the phrase "quick fox" in the body field, or the word "fox" in the title field.
    - (title:"foo bar" AND body:"quick fox") OR title:fox
  - Search for word "foo" and not "bar" in the title field.
    - title:foo -title:bar

Solr Query

- **Wildcard matching**
  - Search for any word that starts with "foo" in the title field.
    - title:foo*
  - Search for any word that starts with "foo" and ends with bar in the title field.
    - title:foo*bar
- **Proximity matching**
  - Search for "foo bar" within 4 words from each other.
    - "foo bar"~4
- **Range Searches**
  - Range Queries allow one to match documents whose field(s) values are between the lower and upper bound specified by the Range Query. Range Queries can be inclusive or exclusive of the upper and lower bounds. Sorting is done lexicographically.
    - mod_date:[20020101 TO 20030101]
- **Boosts**
  - Query-time boosts allow one to specify which terms/clauses are "more important". The higher the boost factor, the more relevant the term will be, and therefore the higher the corresponding document scores.
    - (title:foo OR title:bar)^1.5 (body:foo OR body:bar)

Solr/Lucene Use-cases

- Search
- Analytics
- NoSQL datastore
- Auto-suggestion / Auto-correction
- Recommendation Engine (MoreLikeThis)
- Relevancy Engine
- Solr as a White-List
- Spatial based Search

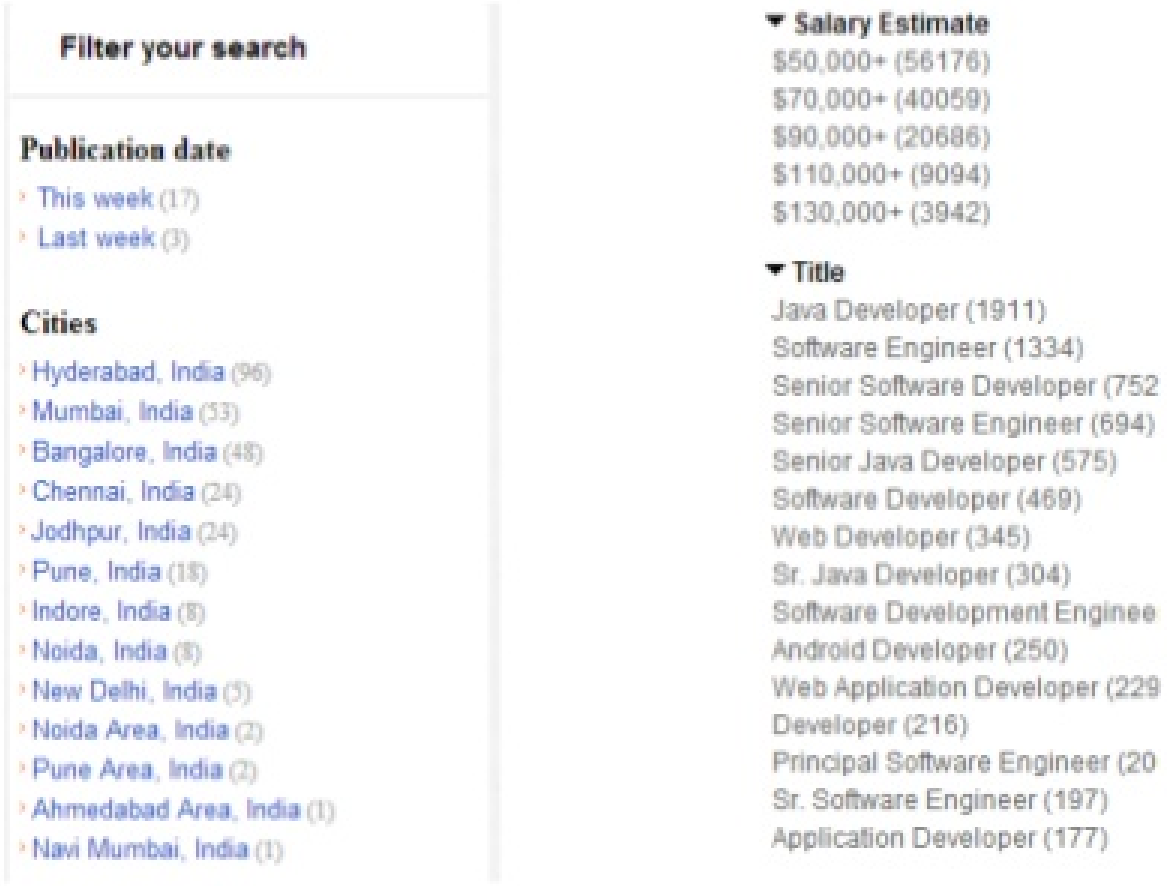
Search

- Application
  - Eclipse, Hibernate search
- E-Commerce :
  - Flipkart.com, Infibeam.com, Buy.com, Netflix.com, ebay.com
- Jobs
  - Indeed.com, Simplyhired.com, Naukri.com, Shine.com,
- Auto
  - AOL.com
- Travel
  - Cleartrip.com
- Social Network
  - Twitter.com, LinkedIn.com, mylife.com

Search (Contd.)

- 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

Results Grouping (using facet)

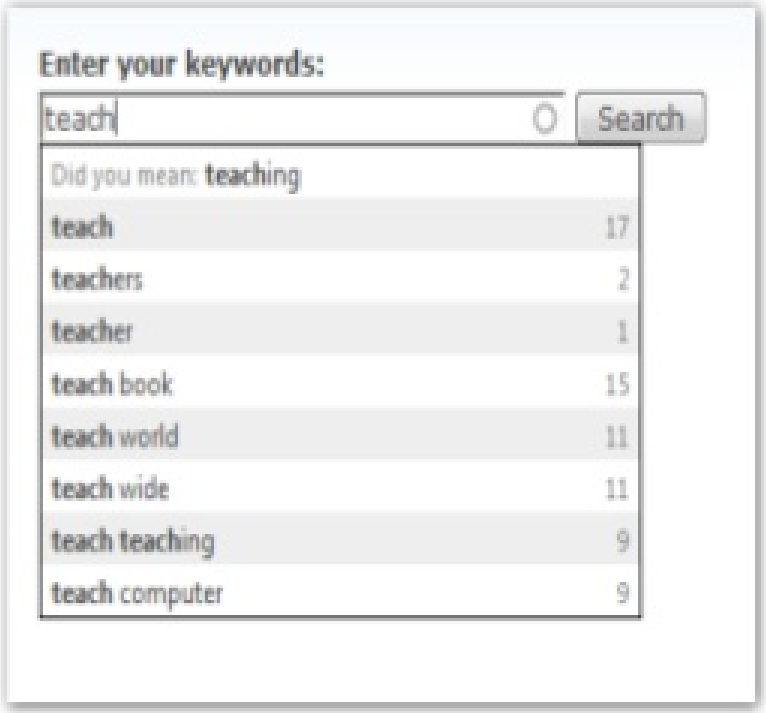


Source: www.career9.com, www.indeed.com

Analytics

- Analytics source : Kibana.org based on ElasticSearch and Logstash
- Image Source : http://semicomplete.com/presentations/logstash-monitorama-2013/#/8

Autosuggestion



Source: www.drupal.org , www.yelp.com

Integration

- Clustering (Solr – Carrot2)
- Named Entity extraction (Solr-UIMA)
- SolrCloud (Solr-Zookeeper)
- Stanbol EntityHub
- Parsing of many Different File Formats (Solr-Tika)

Elasticsearch

Elasticsearch

- Powered by Apache Lucene
- Open-source
- Rapid growth
- High profile users world-wide

GitHub

SOUNDCLOUD

theguardian

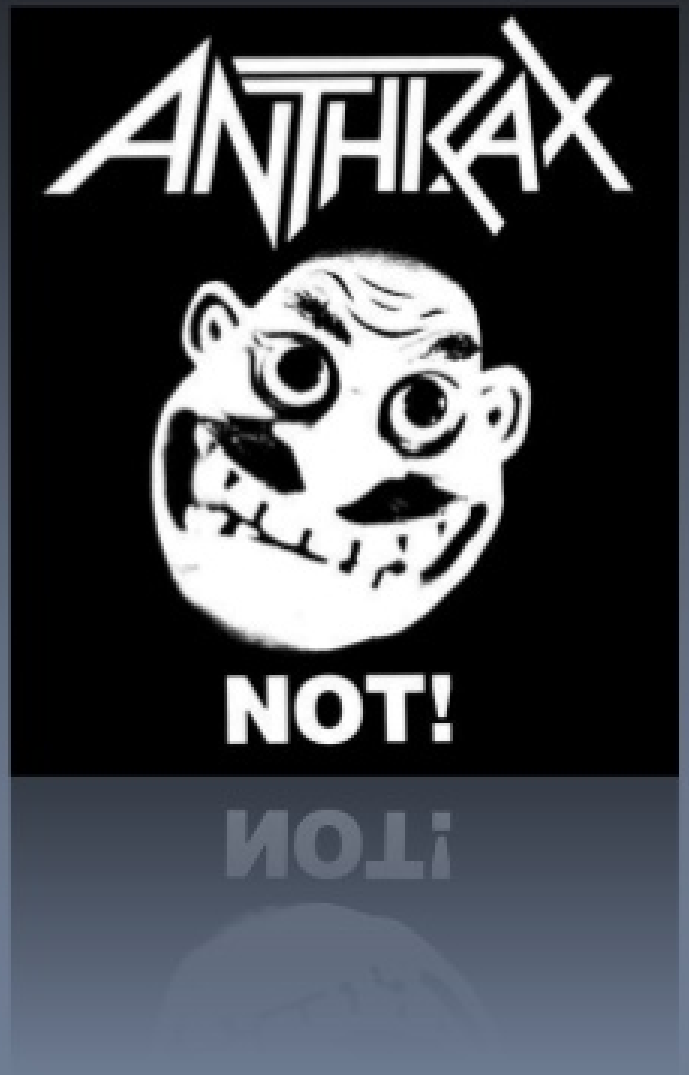
OpenTable

Elasticsearch is…

- Search and Analytics engine
- Document Store
  - Every field is indexed/searchable
- Distributed

What Elasticsearch is not

- Key-Value Store
  - Redis, Riak
- Column Family Store
  - C*, HBase
- Graph Database
  - Neo4J



ElasticSearch in a Nutshell

- Based on Apache Lucene
- Distributed
- Document-Oriented
- Schema free
- HTTP + JSON
- (Near) Real-time search
- Ecosystem
  - Hosting, Monitoring Apps, Clients (SDK)

Where can I get it?

- Free and Open Source
- https://www.elastic.co/
- https://github.com/elastic/elasticsearch
- Backed by a Company, Elastic
  - Training
  - Support
  - Auth/AuthZ
  - Marvel for Monitoring

How do I run it?

- Download it
  - https://www.elastic.co/downloads
- bin/elasticsearch
- http://localhost:9200

```
{
  status: 200,
  name: "Tesla",
  cluster_name: "elasticsearch_royrusso",
  version: {
    number: "1.4.2",
    build_hash: "927caff6f05403e936c20bf4529f144f0c89fd8c",
    build_timestamp: "2014-12-16T14:11:12Z",
    build_snapshot: false,
    lucene_version: "4.10.2"
  },
  tagline: "You Know, for Search"
}
```

Elasticsearch requires Java. *

* You have 5 seconds to whine about it and then shutup.

Elasticsearch is a Document Store

Document Store

- Like MongoDB and CouchDB
- Document DBs:
  - JSON documents
  - Collections of key-value collections
  - Nesting
  - Versioned

Elasticsearch is Distributed

Terminology

MySQL	Elasticsearch
Database	Index
Table	Type
Row	Document
Column	Field
Schema	Mapping
Index	(Everything is indexed)
SQL	Query DSL

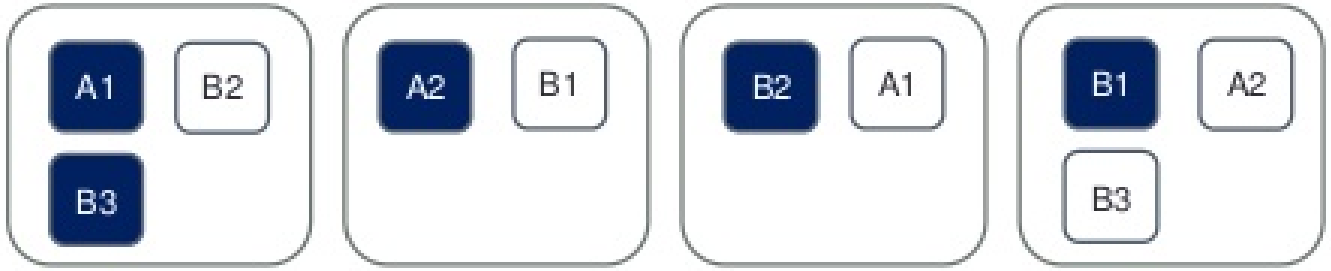
- Cluster: 1..N Nodes w/ same Cluster Name
- Node: One ElasticSearch instance (1 java proc)
- Shard = One Lucene instance
  - 0 or more replicas

High Availability

- No need for load balancer
- Different Node Types
- Indices are Sharded
- Replica shards on different Nodes
- Automatic Master election & failover

Cluster Topology

4 Node Cluster



Index A: 2 Shards & 1 Replica
Index B: 3 Shards & 1 Replica

Discovery

- Nodes discover each other using multicast.
  - Unicast is an option
- Each cluster has an elected master node
  - Beware of split-brain

```
discovery.zen.ping.multicast.enabled: false
discovery.zen.ping.unicast.hosts: ["host1", "host2:port", "host3"]
```

Nodes

- Master node handles cluster-wide (Meta-API) events:
  - Node participation
  - New indices create/delete
  - Re-Allocation of shards
- Data Nodes
  - Indexing / Searching operations
- Client Nodes
  - REST calls
  - Light-weight load balancers

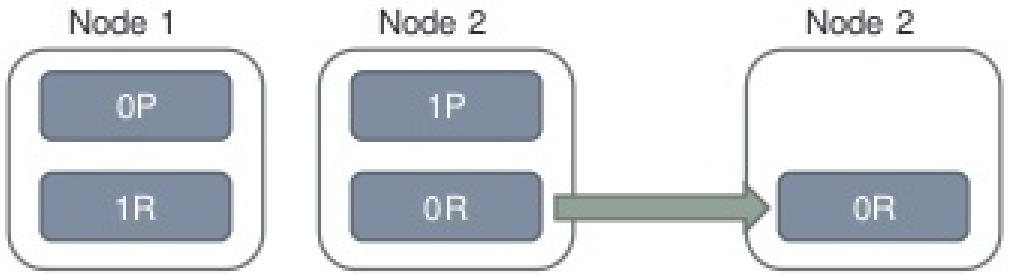
node.data | node.master

The Basics - Shards

- Primary Shard:
  - First time Indexing
  - Index has 1..N primary shards (default: 5)
  - # Not changeable once index created
- Replica Shard:
  - Copy of the primary shard
  - # Can be changed later
  - Each primary has 0..N replicas
  - HA:
    - Promoted to primary if primary fails
    - Get/Search handled by primary||replica

Shard Auto-Allocation

Add a Node: Shards Relocate



- Shard Phases:
  - Unassigned
  - Initializing
  - Started
  - Relocating

Talking to Elasticsearch

REST

- HTTP Verbs: GET, POST, PUT, DELETE
- JSON
- _cat API

```
% curl '192.168.56.10:9200/_cat/health?v&ts=0'
cluster status nodeTotal nodeData shards pri relo init unassign
foo     green         3         3      3   3    0    0        0
```

How Searching Works

- How it works:
  - Search request hits a node
  - Node broadcasts to every shard in the index
  - Each shard performs query
  - Each shard returns metadata about results
  - Node merges results and scores them
  - Node requests documents from shards
  - Results merged, sorted, and returned to client.

REST API - Search

- Free Text Search
  - URL Request
- Complex Query

```
http://localhost:9200/imdb/movie/_search?q=scar*

http://localhost:9200/imdb/movie/_search?q=scarface+OR+star

http://localhost:9200/imdb/movie/_search?q=(scarface+OR+star)+AND+year:[1981+TO+1994]
```

Solr vs. Elasticsearch

- http://solr-vs-elasticsearch.com/

SOLR vs ElasticSearch

- SOLR
  - Well-known, many tools, extensions
  - Feels clunky to configure
  - Manual document to lucene mapping
  - Replication and indexing in a cluster non-trivial
- Old school ;-)

- ElasticSearch
  - New kid on the block
  - Very easy to configure
  - Handles document to lucene mapping
  - Horizontally scalable
    - Easy replication
    - But: shard key
- New school

search evolution

- Custom indexers

- Inverted index
- Segment merges

- Custom analyzers
- Faceting

- Configuration of analyzers
- Faceting, Geospatial

- Document mapping
- Sub-document queries
- Replication

- JSON document input
- Faceting, complex queries just work

Under the Hood

- ElasticSearch 0.20
  - Apache Lucene 3.6.1
- Apache Solr 4.0
  - Apache Lucene 4.0

sematext

Copyright 2012 Sematext Int'l. All rights reserved

ElasticSearch Cluster Architecture

- Distributed
- Fault tolerant
- Only ElasticSearch nodes
- Single leader
- Automatic leader election

sematext

Copyright 2012 Sematext Int'l. All rights reserved

SolrCloud Cluster Architecture

- Distributed
- Fault tolerant
- Apache Solr + ZooKeeper ensemble
- Leader per shard
- Automatic leader election

sematext

Copyright 2012 Sematext Int'l. All rights reserved

Collection vs Index

- Collection – Solr main logical index
- Index – ElasticSearch main logic structure
- Collections and Indices can be spread among different nodes in the cluster

sematext

Copyright 2012 Sematext Int'l. All rights reserved

Multiple Document Types in Index

- ElasticSearch - multiple document types in a single index
- Apache Solr - multiple document types in a single collection – shared schema.xml

sematext

Copyright 2012 Sematext Int'l. All rights reserved

Shards and Replicas

- Index / Collection can have many shards
- Each shard can have 0 or more replicas
- Replicas are automatically updated
- Replicas can be promoted to leaders when a leader shard goes off-line

sematext

Copyright 2012 Sematext Int'l. All rights reserved

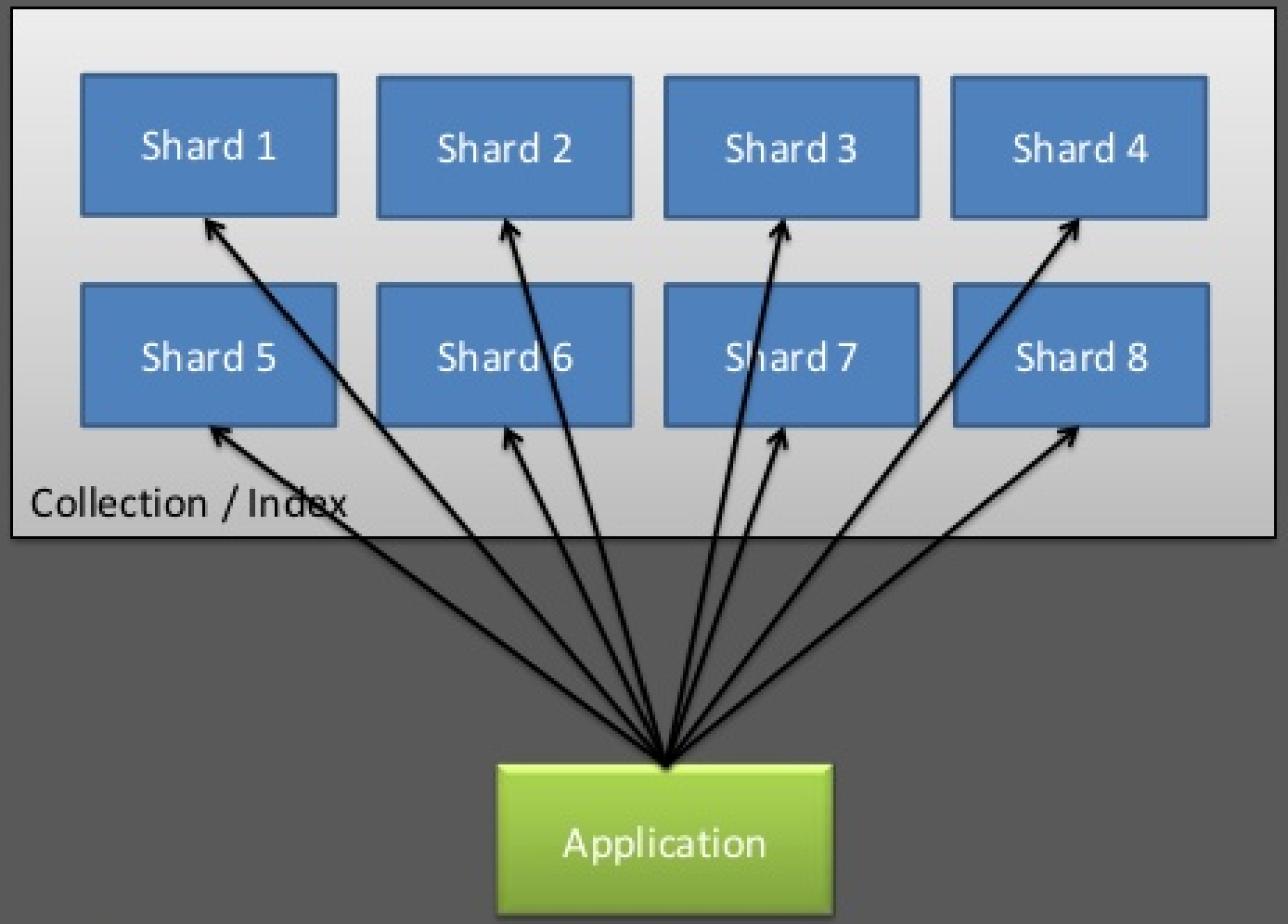
Index and Query Routing

- Control where documents are going
- Control where queries are going
- Manual data distribution

sematext

Copyright 2012 Sematext Int'l. All rights reserved

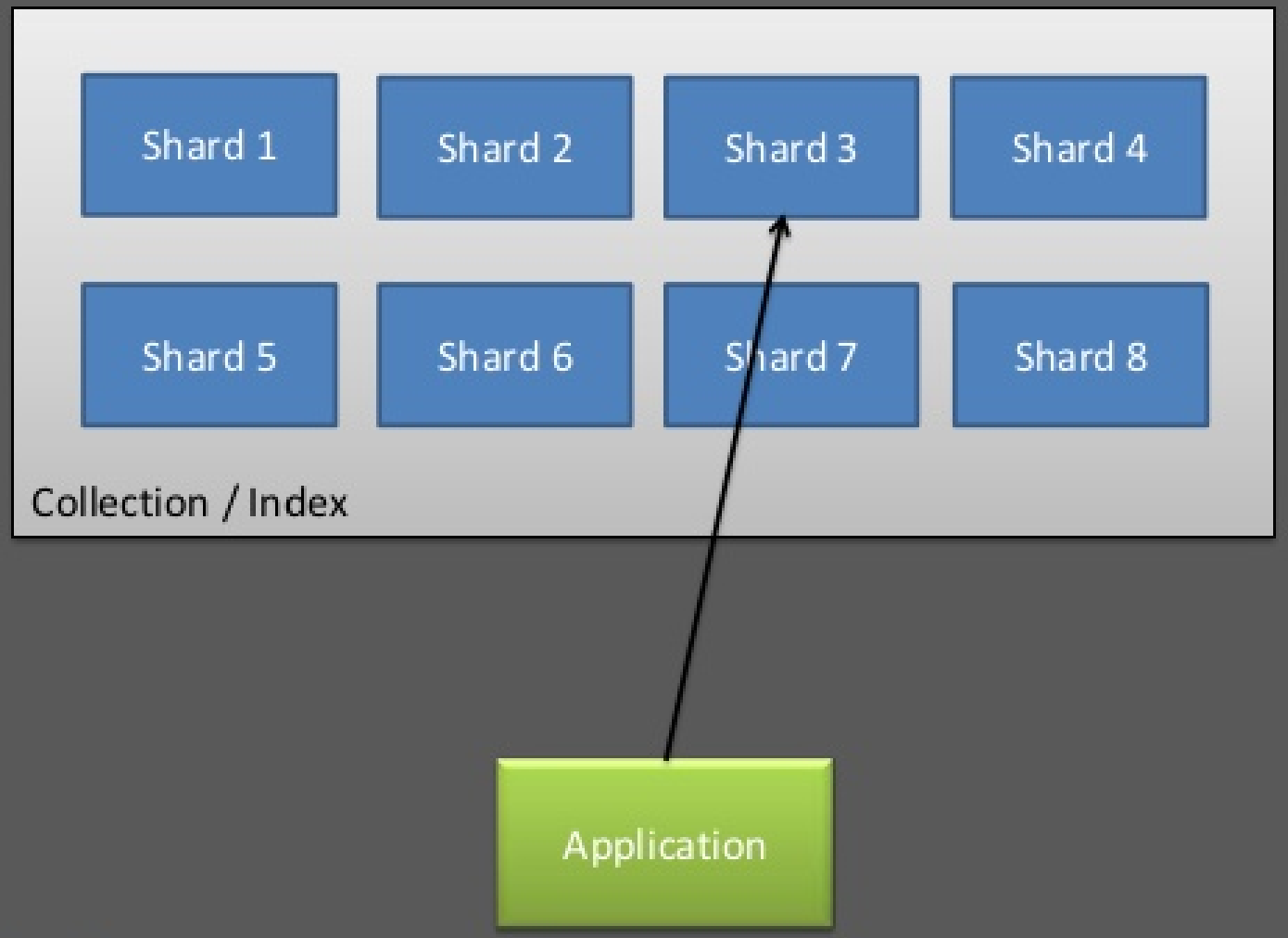
Querying Without Routing



sematext

Copyright 2012 Sematext Int'l. All rights reserved

Query With Routing



sematext

Copyright 2012 Sematext Int'l. All rights reserved

Routing Docs and Queries in Solr

- Requires some effort
- Defaults to hash based on document identifiers
- Can be turned off using `solr.NoOpDistributingUpdateProcessorFactory`

```
<updateRequestProcessorChain>
  <processor class="solr.LogUpdateProcessorFactory" />
  <processor class="solr.RunUpdateProcessorFactory" />
  <processor class="solr.NoOpDistributingUpdateProcessorFactory" />
</updateRequestProcessorChain>
```

sematext

Copyright 2012 Sematext Int'l. All rights reserved

Routing Docs and Queries - ElasticSearch

- `routing` parameter controls target shard which document/query will be forwarded to
- defaults to document identifiers
- can be changed to any value

```
curl -XPUT localhost:9200/sematext/test/1?routing=1234 -d '{
  "title" : "Test routing document"
}'

curl –XGET localhost:9200/sematext/test/_search/?q=*&routing=1234
```

sematext

Copyright 2012 Sematext Int'l. All rights reserved

Apache Solr Index Structure

- Field types defined in schema.xml file
- Fields defined in schema.xml file
- Allows automatic value copying
- Allows dynamic fields
- Allows custom similarity definition

sematext

Copyright 2012 Sematext Int'l. All rights reserved

ElasticSearch Index Structure

- Schema - less
- Analyzers and filters defined with HTTP API
- Fields defined with an HTTP request
- Multi – field support
- Allows nested documents
- Allows parent – child relationship
- Allows structured data

sematext

Copyright 2012 Sematext Int'l. All rights reserved

Discovery

- Apache Solr uses ZooKeeper
- ElasticSearch uses Zen Discovery

sematext

Copyright 2012 Sematext Int'l. All rights reserved

ElasticSearch Zen Discovery

- Allows automatic node discovery
- Provides multicast and unicast discovery methods
- Automatic master detection
- Two - way failure detection

sematext

Copyright 2012 Sematext Int'l. All rights reserved

Apache Solr & Apache ZooKeeper

- Requires additional software
- ZooKeeper ensemble with 1+ ZooKeeper instances
- Prevents split – brain situations
- Holds collections configurations
- Solr needs to know address of one of the ZooKeeper instances

sematext

Copyright 2012 Sematext Int'l. All rights reserved

Data Handling

- Solr
  - Multiple formats allowed as input
  - Can return results in multiple formats
- ElasticSearch
  - JSON in / JSON out

sematext

Copyright 2012 Sematext Int'l. All rights reserved

Single or Batch

- Solr
  - Single or multiple documents per request
- ElasticSearch
  - Single document with a standard indexing call
  - `_bulk` end – point exposed for batch indexing
  - `_bulk` UDP end – point can be exposed for low latency batch indexing

sematext

Copyright 2012 Sematext Int'l. All rights reserved

And The Winner Is ?

The Users

sematext

Copyright 2013 Sematext Group. Inc. All rights reserved