Terrier

Researching and Building IR applications using Terrier

Craig Macdonald & Ben He

Department of Computing Science

University of Glasgow

ECIR 2008 - 30th March

Terrier

Researching and Building IR applications using Terrier

Craig Macdonald & Ben He

Department of Computing Science
University of Glasgow

ECIR 2008 - 30th March

Tutorial Roadmap

- IR Science
- How to build large-scale IR systems in 2008
- Terrier as an illustration
  - How to use Terrier
  - How to extend Terrier
  - Looking to the future

And the Outline

- Motivations for an open source IR platform
- Indexing
  - Background
  - Terrier implementation
  - Terrier usage
  - Extending Terrier indexing

Outline (2)

- Retrieval
  - Document Weighting Models
  - Query Expansion
  - Terrier usage
  - Extending Terrier to your research ideas
- Wrap up

IR as an empirical science

- IR is about **thorough** and **large-scale** experimentation
  - – Just look at recent SIGIR/ECIR proceedings!

- Designing a new IR idea is about:
  - – Knowing what has gone before
  - – Having the idea, and implementing it
  - – Comparing to the current state-of-the-art

TREC

- Initiative for the cross-comparison of IR systems
  - Exemplifies IR as an empirical science
- TREC Paradigm:
  - Corpus of documents
  - Queries (known as topics)
  - Relevance assessments for each query
    - Which documents are relevant to the query?
- Test Collections are the basis for advancing knowledge in IR

TREC Collections

- Constantly Increasing in size

Collection	*Year*	*Docs*	*Size(GB)*
Disk 1&2	1992	740K	2.0
Disk 4&5	1998	500K	1.9
WT2G	1999	240k	2.0
WT10G	2000	1.6M	10.0
GOV	2002	1.8M	18.0
Blog06	2006	3M	13.0
GOV2	2004	25M	425.0

Experimenting in IR

- Recall Experimentation Outline:
  - Understand state-of-the-art
  - Design & implement new idea
  - Compare new idea to state-of-the-art
  - Draw conclusions
- Do you really want to implement the state-of-the art techniques?

Experimentation Timeline

OLD

- Read about state-of-the-art
- Implement state-of-the-art
- Design & implement new technique
- Experiments
- Paper, PhD

NEW

- Read about state-of-the-art
- Implement state-of-the-art
- Design & implement new technique
- Experiments
- Paper, PhD

GRAPHICAL?

Existing IR Platforms

- Academic:
  - Terrier
  - Zettair
  - Lemur/Indri
- Non-Academic:
  - Lucene/Nutch
  - Xapian

Terrier:

- Flexible & ideal for experimentation
- Rapid development of new ideas
- Not just one model
  - Implements various modern state-of-the-art IR models
- Proven effective retrieval

Learning Outcomes

- By the end of the tutorial:
  - Learn more about large-scale IR systems
  - How to use Terrier
    - Design and evaluate an IR experiment
  - Extend Terrier to your research ideas

- Join us in improving our Open Source platform ;-)

What is Terrier?

- Research project (2001-)
  - Part of its outcome is released as open source software
    - Latest release version 2.1 (19/03/08)
  - Researchers, projects, PhD students and undergraduates all involved
  - Funded by UK Engineering & Physical Sciences Research Council (GR/R90543/01); Leverhulme Trust (F/00179/S)

- Evaluation of Terrier
  - TREC Web, Robust, Terabyte, Enterprise, Blog tracks
  - CLEF Ad-hoc and Web tracks

- Platform to develop new research ideas for experimentation
  - Modern platform implementing various state-of-the-art techniques for indexing and retrieval

Open Source Terrier

- Why Open Source? Terrier is a community project
  - you use & **benefit**
  - you contribute
  - **Everyone** benefits

- Cross-OS developed in Java
  - runs on Windows, *nix, MacOS X

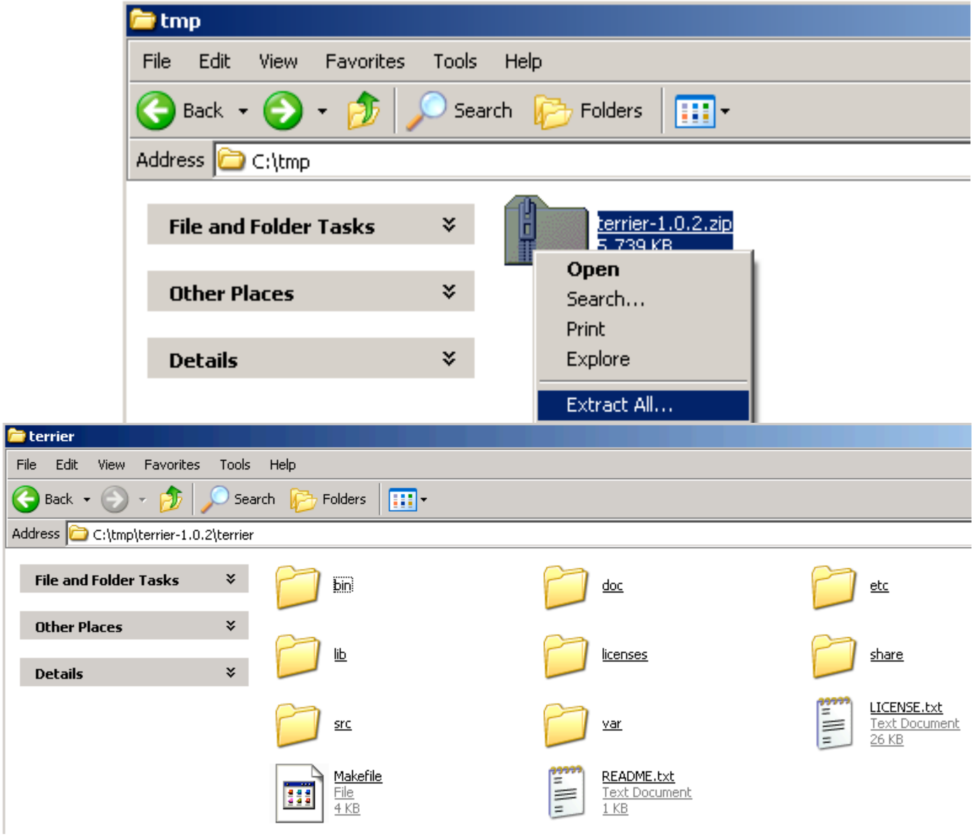
- Indexing and Querying APIs
  - Easy to extend – adapt for new applications
  - Modular architecture
  - Simple to start working with
  - Many configuration options

Installing Terrier

TODO new windows figures

- What do I need:
  - Sun Java 1.5 or newer
  - The package, e.g. terrier-2.1.tar.gz or terrier-2.1.zip
  - Linux / Windows operating system

```
[toto@boano tmp]$ ls
terrier-1.0.2.tar.gz
[toto@boano tmp]$ tar -xzf terrier-1.0.2.tar.gz
[toto@boano tmp]$ ls
terrier  terrier-1.0.2.tar.gz
[toto@boano tmp]$ cd terrier
[toto@boano terrier]$ ls
bin  etc  licenses    Makefile    share  var
doc  lib  LICENSE.txt  README.txt  src
[toto@boano terrier]$
```

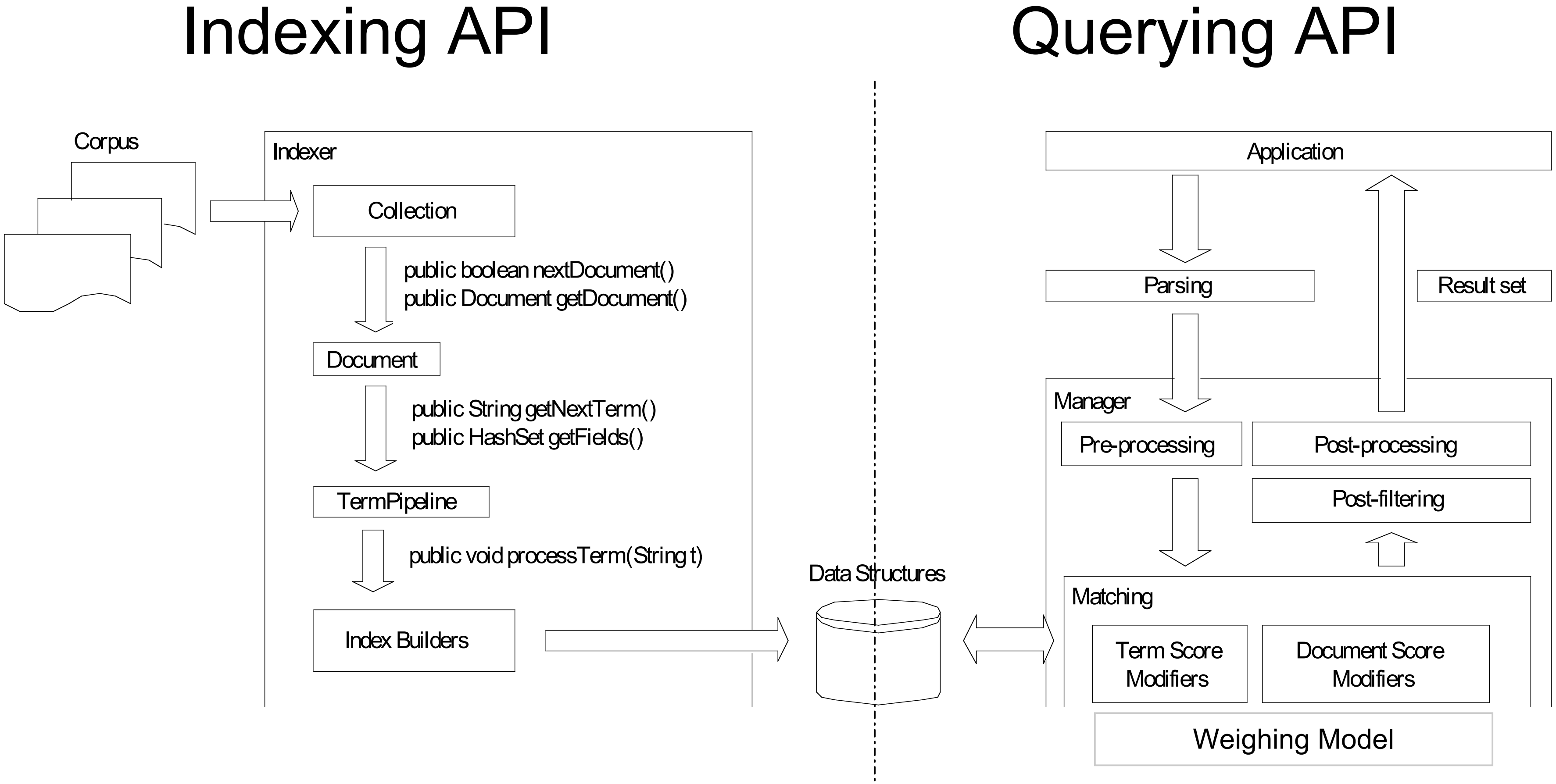


- Terrier is pre-compiled
  - ***No need to compile***

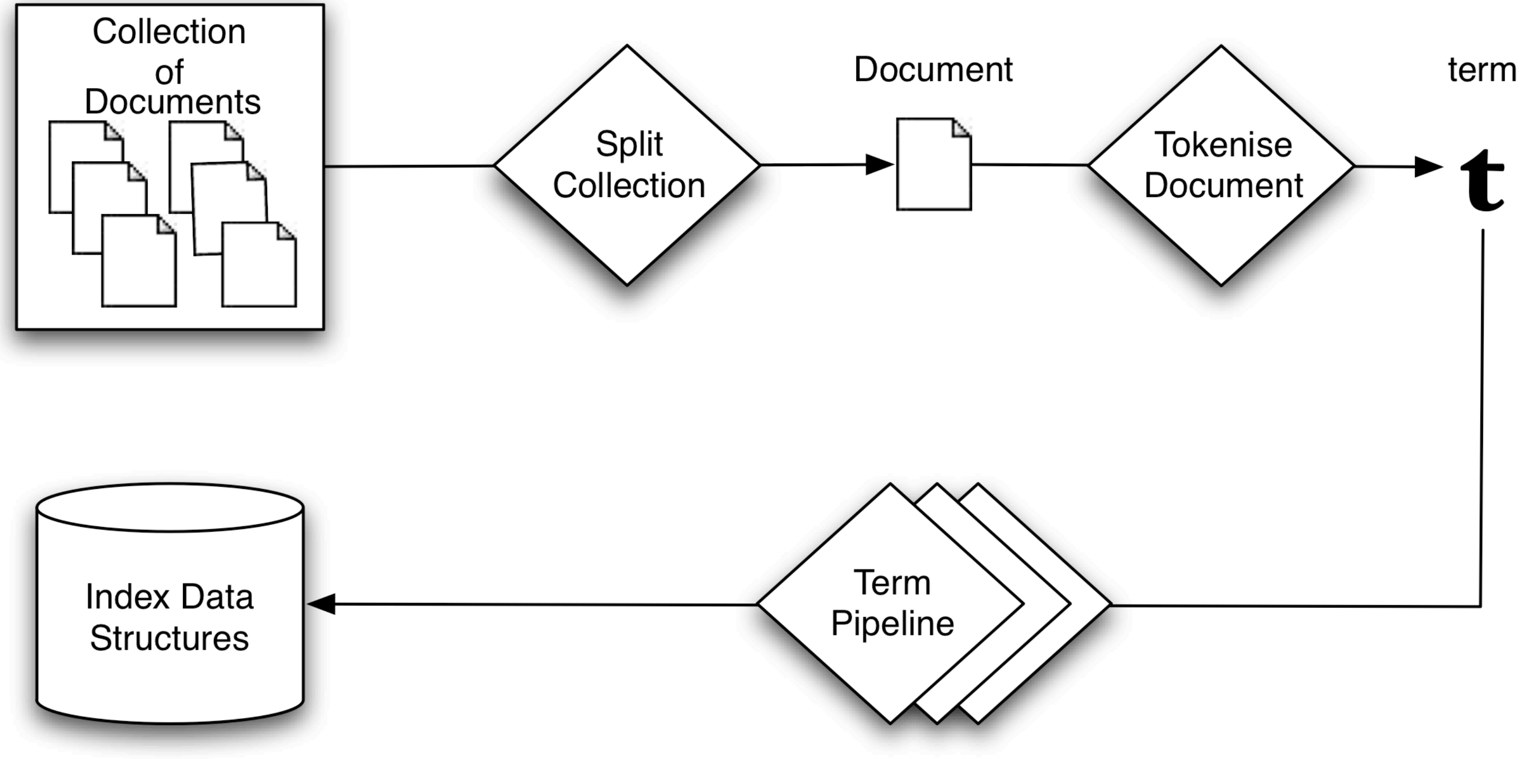
What’s in Terrier

bin/	Scripts to Start Terrier
doc/	Documentation
etc/	Terrier Configuration Files
lib/	Compiled Java files
share/	Stopwords & tests
src/	Java Source for Terrier
var/	
– index/	Create index
– results/	Output from Batch Retrieval Experiments

Terrier Architecture Overview



Indexing



eg:
Stemming
Stopword Removal

Indexing Large-scale Collections

- Requirements:
  - Parse (heterogeneous?) collections of documents
    - Web documents, Office files, Publications
  - Process tokens: stemming, stopword removal, synonymy
  - Create compressed index structures, quickly
  - Easily accessible index structures for efficient matching

Document Tokenisation (Lexical Analysis)

- The process of converting a stream of characters (the text of the documents) into a stream of words (the candidate words to be adopted as index terms)
  - i.e. identification of the words in the text
    - Recognition of spaces ?
      - Easy for English, French,…
      - Chinese?
  - treating digits, hyphens, punctuation marks, and the case of the letters.

- Cases to be considered with care
  - Numbers (e.g. 1999 vs. 510B.C)
  - Hyphens (e.g state-of-the-art vs. B-49)
  - Punctuation (e.g. 510B.C vs. list.id)
  - Case of letters (e.g. Bank vs. bank)

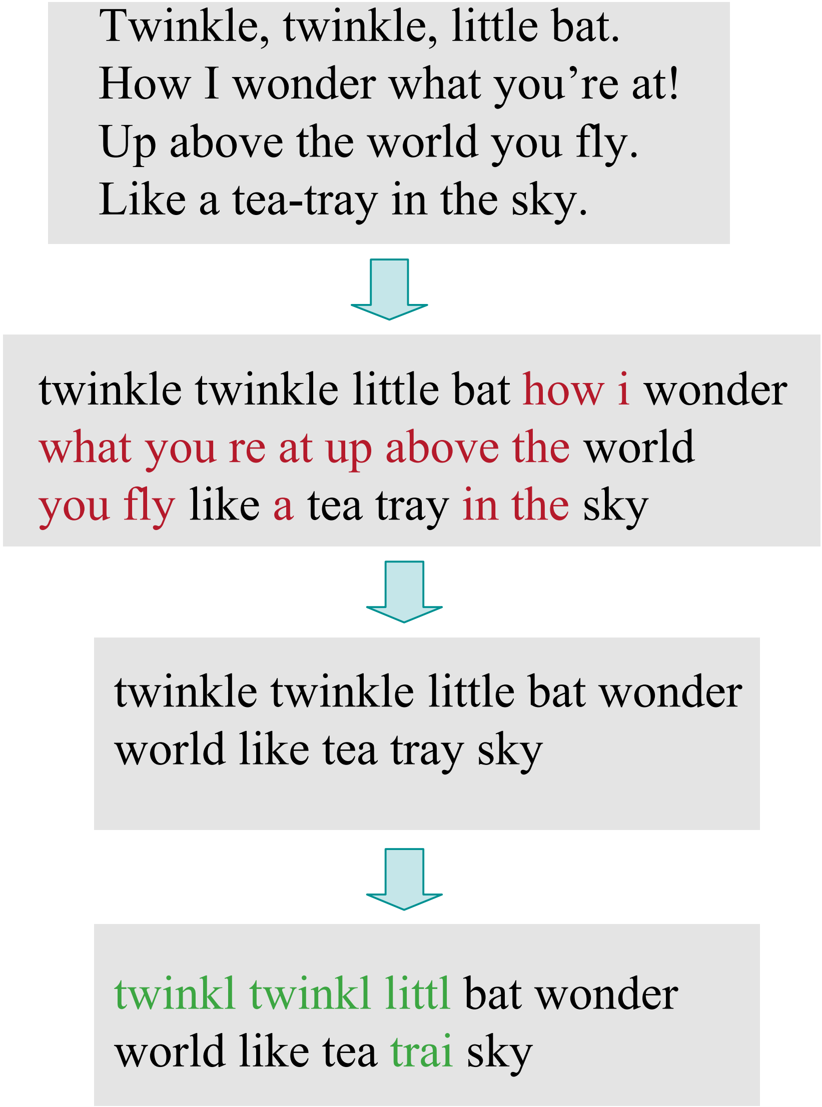
SEE ALSO: Lexical Analysis and stoplists
by E. Fox, In Information Retrieval - Data Structures & Algorithms by Frakes & Yates, Prentice-Hall

Term Pipelining

- In Terrier, each token from a Document is passed through the Term Pipeline
- Each Term Pipeline stage can either:
  - Transform the term
  - Drop the term
- Why?
  - Stemming, ala Porter’s English stemming etc.
  - Stopword removal
- Flexibility
  - Various chains of stages
  - Language specific stemming
  - Synonymy, ontologies, etc.

Example

- Original Text
- Tokenisation
- Stopword removal
- Stemming



Twinkle, twinkle, little bat.
How I wonder what you're at!
Up above the world you fly.
Like a tea-tray in the sky.

twinkle twinkle little bat how i wonder what you re at up above the world you fly like a tea tray in the sky

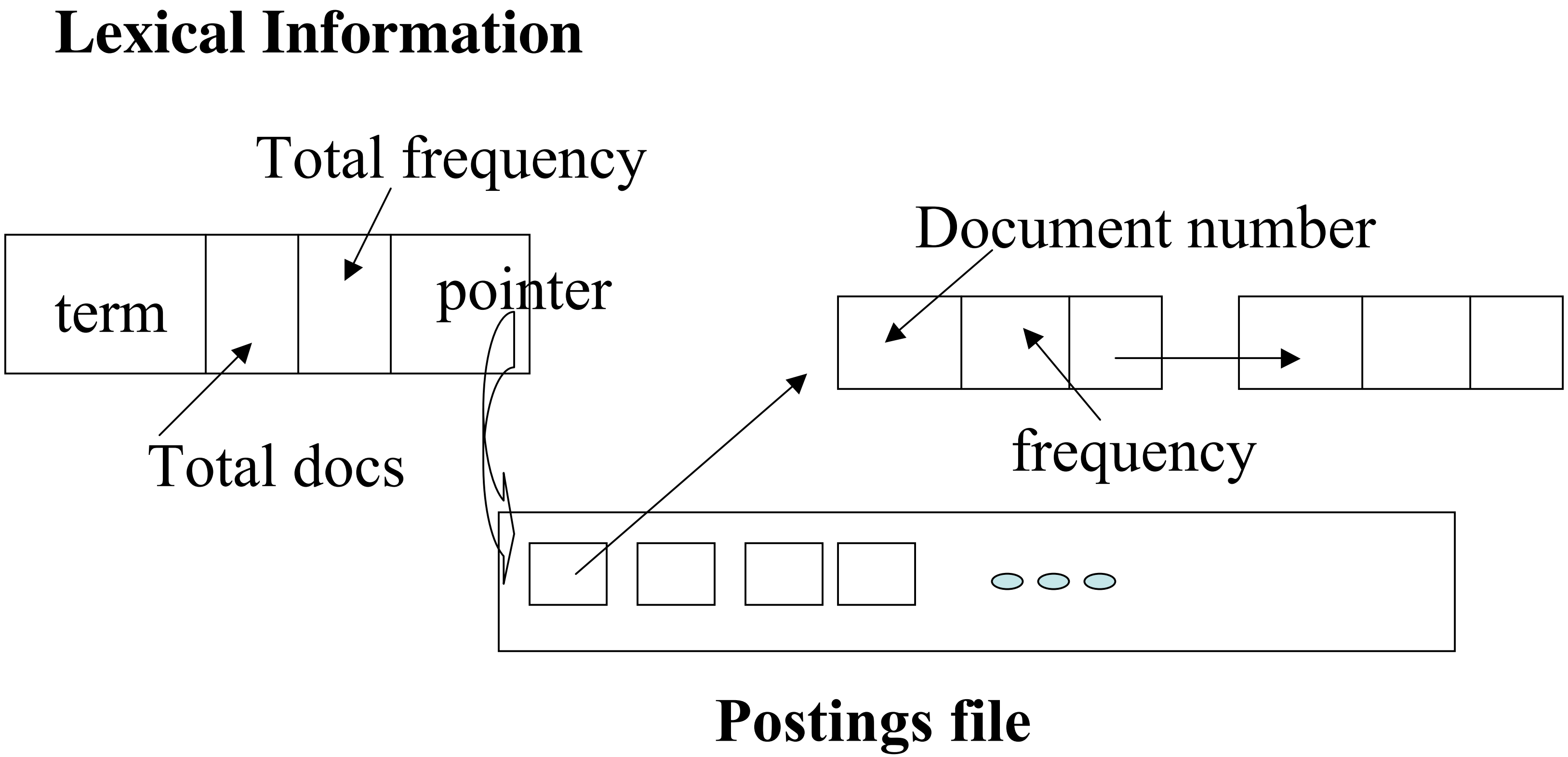
twinkle twinkle little bat wonder world like tea tray sky

twinkl twinkl littl bat wonder world like tea trai sky

Basic Index Data Structures

- Lexicon
  - Information about each term: Frequencies
  - Offset in Inverted Index
- Inverted Index
  - Posting lists for each term: <docid, tf>
  - Posting list might also contain
    - the (eg HTML) tags (known as fields) that the term occurs in
    - Positions of occurrences in document: for “phrase matching”
- Document Index
  - Length of each document

Components of an inverted file



Direct Index

- An inverted index stores for a term, the documents that term occurs in
  - For fast retrieval of documents given a query
- A **direct index** stores for a document, the terms that occurred in the document
- Direct Index facilitates
  - Query Expansion (Pseudo-relevance feedback) ✓
  - Document clustering
  - Document classification
  - Document-document similarity

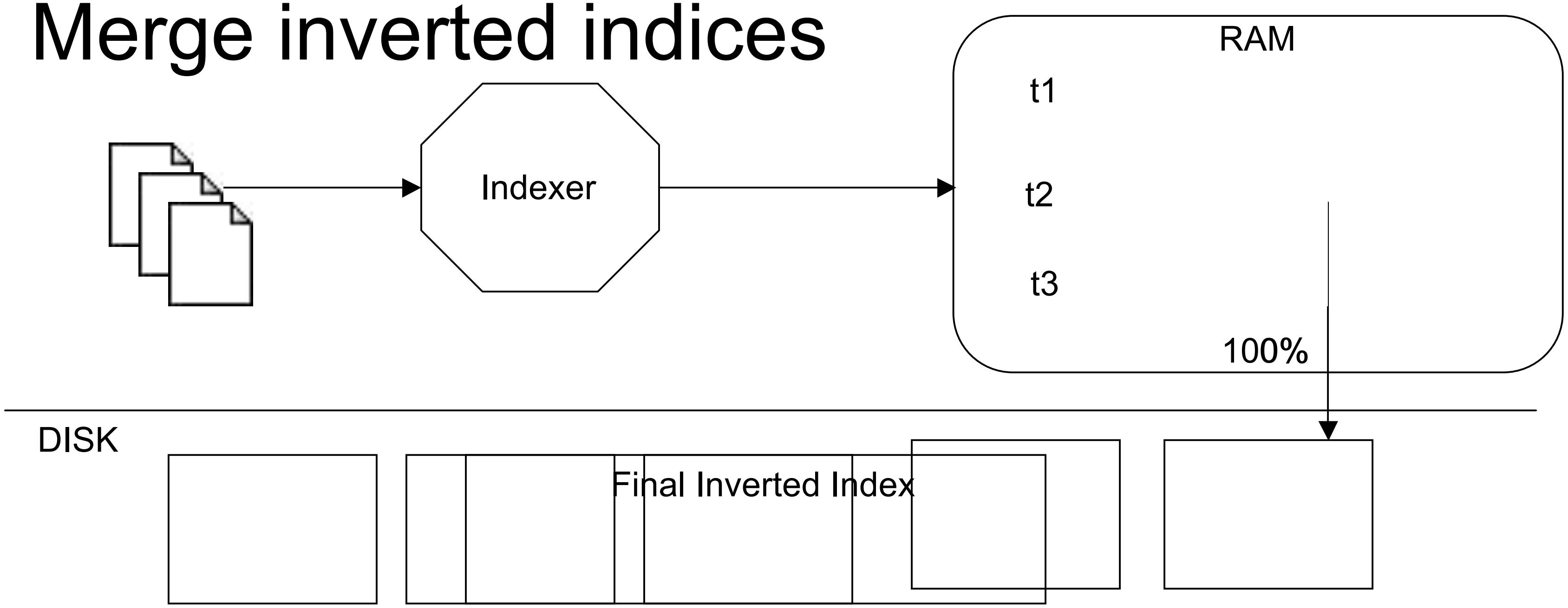
How to Index (1): Direct file Indexing with Sort based Inversion

- Iterate through collection, recording IDs for each new terms PROBLEM: MEMORY
  - Write out to Direct Index each document (records terms for each document)
- Inversion:
  - Scan entire direct index, building posting lists for each term in memory PROBLEM: MEMORY
  - Write out inverted postings once you know all documents each term occurs in

TODO: ANIMATION

How to Index (2): Single Pass Indexing

- Parse collection, building inverted index postings in memory
  - – Write out to disk when memory is exhausted
- Merge inverted indices



However, no direct index by default

Index Compression

- Indices can be compressed: this reduces disk IO, making indexing and retrieval faster

SEE ALSO: Managing Gigabytes: Witten, Moffat & Bell
Morgan Kaufmann Publishing 1999

- Compression Examples:
  - Variable Byte Encoding
  - Elias-Unary Encoding
  - Elias-Gamma Encoding

Posting Compression

t1: <1, 5> <5,4> <19,3>

Record only gaps

t1: <1,5> <4,4> <14,3>

Unary ,

the

Gamm (

1. Sep (
   and
2. Enc
3. App

Comp !

Integer Encoding (32 bits each= 24 bytes)

0000000000000000000000000000001

Unary Encoding (4 bytes)

10000100001000100000000000001001

0000000000000000000000000000100

0000000000000000000000000001110

Gamma,Unary Encoding (2.7 bytes) 1

10010100100010001110

Where in Terrier?

- Tokenisation:
  - `uk.ac.gla.terrier.indexing.*Document`
- Term Pipelines:
  - `uk.ac.gla.terrier.terms.*`
- Compression:
  - Gamma compression for docids
  - Unary compression for tf
  - `uk.ac.gla.terrier.compression.*`
- Two-phase indexing:
  - `uk.ac.gla.terrier.indexing.(Basic|Block)Indexer`
  - `uk.ac.gla.terrier.structures.indexing.*`
- Single-pass indexing:
  - `uk.ac.gla.terrier.indexing.(Basic|Block)SinglePass Indexer`
  - `uk.ac.gla.terrier.structures.indexing.singlepass.*`

Let’s try in Terrier

- Specify collection to index
- Indexing collection
- Advanced Indexing Options
- Extending Indexing

Indexing with Terrier (1)

- Terrier can readily index tagged and TREC formatted test collections

TREC AP Collection

```
<DOC>
<DOCNO> AP890101-0002 </DOCNO>
<FILEID>AP-NR-01-01-89 2359EST</FILEID>
<FIRST>r a PM-FutureFactory     01-01 0872</FIRST>
<SECOND>PM-Future Factory,0897</SECOND>
<HEAD>University Erects A Factory Of The
    Future</HEAD>
<HEAD>Eds: Also in Monday AMs report.</HEAD>
<BYLINE>By DONNA BRYSON</BYLINE>
<BYLINE>Associated Press Writer</BYLINE>
<DATELINE>ROLLA, Mo. (AP) </DATELINE>
<TEXT>For students working in a miniature factory
    at the University of Missouri-Rolla, the
    future of American business is now...</TEXT>
</DOC>
<DOC>...
```

TREC .GOV2 Collection

```
<DOC>
<DOCNO>GX010-60-0164440</DOCNO>
<DOCHDR>
http://www.emsc.nysed.gov/repcrd2003/links/sg29.html
HTTP/1.1 200 OK
Server: Netscape-Enterprise/3.6 SP1
Date: Wed, 10 Dec 2003 08:52:41 GMT
Content-type: text/html
Last-modified: Mon, 19 May 2003 20:49:43 GMT
Content-length: 17183
Accept-ranges: bytes
Connection: close
</DOCHDR>
<html>
<head>
<title>Similar Schools Group #29 for 2001-2002</title>
</head>
<body bgcolor="#FFFFFF">
<p align="center"><img src="../images/emscban.gif"
    width="476" height="111" alt="Office of
    Elementary, Middle, Secondary and Continuing
    Education"></p>...
</DOC>
<DOC>...
```

Indexing with Terrier (2)

- Setup Terrier with:

```
bin/trec_setup.sh /path/to/collection
```

- This will:
  - Make a default configuration properties file *etc/terrier.properties*
  - Create an *etc/collection.spec* that contains a list of files to index
    - *Before proceeding, it's worth checking that the etc/collection.spec file contains only the files you want to index*

Indexing with Terrier (3): Specify Collection to Index

```
[toto@boano terrier]$ bin/trec_setup.sh /path/to/collection/
Setting TERRIER_HOME to /users/toto/tmp/terrier
Setting JAVA_HOME to /local/java/linux/jdk1.5.0
Creating collection.spec file.
Creating trec.qrels file.
Creating topics file.
Creating models file.
Creating query expansion models (qemodels) file.
Creating terrier.properties file.
#add the files to index
/path/to/collection/AP890103.gz
/path/to/collection/AP890104.gz
...
/path/to/collection/AP891231.gz
/path/to/collection/AP890101.gz
/path/to/collection/AP890102.gz
/path/to/collection/README.gz
Updated collection.spec file. Please check that it contains
all and only all the files to be indexed, or create it manually.
[toto@boano terrier]$
```

Indexing with Terrier (4): Direct index based indexing with Inversion

- Index the collection with `bin/trec_terrier.sh -i`

```
[toto@boano terrier]$ bin/trec_terrier.sh -i
Setting TERRIER_HOME to /users/toto/tmp/terrier
Setting JAVA_HOME to /local/java/linux/jdk1.5.0
TRECCollection read collection specification
Processing /path/to/collection/AP890103.gz
Processing /path/to/collection/AP890104.gz
Processing /path/to/collection/AP890105.gz
Processing /path/to/collection/AP890106.gz
Processing /path/to/collection/AP890107.gz
Processing /path/to/collection/AP890108.gz
...
Finished building the inverted index...
Time elapsed for inverted file: 239.298
Time elapsed: 1119.203 seconds.
[toto@boano terrier]$
```

Indexing with Terrier (5): Single Pass Indexing

v2.0

Index the collection with `bin/trec_terrier.sh -i -j`

```
[toto@boano terrier]$ bin/trec_terrier.sh -i -j
INFO - TRECCollection read collection specification (20 files)
INFO - Processing /path/to/collection/AP890103.gz
Starting building the inverted file...
INFO - creating the data structures data_1
INFO - Creating IF (no direct file)..
INFO - Collection #0 took 20 seconds to
 build the runs for 210 documents
INFO - Merging 1 runs...
INFO - Collection #0 took 21 seconds to merge

INFO - Collection #0 total time 21
Finished building the inverted index...
Time elapsed for inverted file: 0.655
Time elapsed: 0.732 seconds.
```

Why use Single Pass Indexing?

- Advantages:
  - (Much) faster
  - less memory problems with larger collections
  - Reduced storage required, as direct index is not built

Collection	Two-phase	Single-pass
Disk1&2	13.5min	8.65min
Disk4&5	11.7min	7.63min
WT2G	9.95min	7.52min
WT10G	1 hr 2.5min	34.68min
.GOV	1 hr 11min	47.1min
Blog06	4hr 40min	3hour 3min

- Disadvantage: No direct index is created
  - (use uk.ac.gla.terrier.structures.indexing.singlepass.Inverted2DirectIndexBuilder to fix)

Terrier Index files

The indexing process generates files in directory var/index:

1. Document index (data.docid) — set $\{doc_i\}$
2. Vocabulary/Lexicon (data.lex) — set $\{kw_j\}$
3. Direct index (data.df) — $doc_i \xrightarrow{about} \{kw_j\}$
   - (Not for single-pass indexing)
4. Inverted index (data.if) — $kw_j \xrightarrow{describes} \{doc_i\}$
5. Index properties (data.properties)
   - •Specifies Index classes
   - •Index statistics

But that’s not all….

- This is basic indexing
- Terrier can do much more
  - Allowing flexibility

- Examples:
  - Which stemmer?
  - Non-English settings
  - Term positions
  - Fields

Configuring Terrier (1)

- You can configure Terrier to your needs by editing the file ***etc/terrier.properties***

```
#directory names
terrier.home=/users/toto/tmp/terrier
...
#stop-words file
stopwords.filename=stopword-list.txt
...
#the processing stages a term goes through
termpipelines=Stopwords,PorterStemmer
```

- Look at ***etc/terrier.properties.sample*** for examples.
  - **Documentation** contains pointers to specific properties
  - The **Javadoc** for each class lists properties that affect it
  - **doc/properties.html** lists all known properties in Terrier
- Eg: I want to create/open an index at /path/to/index
  - `terrier.index.path=/path/to/index`

Configuring Terrier (2)

- Use a different stemmer by changing the Term Pipeline

```
termpipelines=Stopwords,WeakPorterStemmer
```

  - {PorterStemmer,WeakPorterStemmer, *SnowballStemmer,<blank>}
- Disable removing of stopwords

```
termpipelines=SpanishPorterStemmer
```

  - {Stopwords,<blank>}
- Show terms in pipeline

```
termpipelines=DumpTerm,PorterStemmer,DumpTerm
```

Indexing with Blocks

- Save exact positions of terms in order to do “phrase search” or proximity search

```
block.indexing=true
block.size=1
max.blocks=1000
```

- Increases indexing time
  - But less marked for single-pass indexing

Collection	Two-phase	Two-phase + Blocks	Single-pass	Single-pass + Blocks
Disk1&2	13.5min	32.6min	8.65min	12.1min
Disk4&5	11.7min	25.0min	7.63min	10.2min
WT2G	9.95min	23.6min	7.52min	10.8min
WT10G	1 hour 2.5min	2hour 18min	34.68min	53.1min
.GOV	1hour 11 min	2hour 45min	47.1min	1hour 11 min
Blog06	4hr 40min	10hour 36min	3hour 3min	4hour 19min

Configuring Terrier (3)

- Indexing fields
  - Save whether a term appears within a particular tag
  - HTML tags
  - collection specific tags
  - Tags indicating the language of a document

- Index **fields**

```
FieldTags.process=TITLE,H1
```

- Specify **which tags** will be indexed

```
TrecDocTags.doctag=DOC
TrecDocTags.idtag=DOCNO
TrecDocTags.skip=DOCHDR
```

- Index only the titles of documents

```
TrecDocTags.doctag=DOC
TrecDocTags.idtag=DOCNO
TrecDocTags.process=TITLE
```

```
<DOC>
<DOCNO>DOC-X1</DOCNO>
<DOCHDR>
. . .
</DOCHDR>
<TITLE>. . .</TITLE>
. . .
<H1>. . .</H1>
</DOC>
```

Configuring Terrier for Non-English Documents

- Set `string.use_utf=true` to support Unicode characters in the Lexicon **v1.1.0**
- Use `trec.collection.class=TRECUTFCollection` to parse TREC-like Collections
- Use non-English stemmers: **v1.1.1**

DanishSnowballStemmer, DutchSnowballStemmer, EnglishSnowballStemmer, FinnishSnowballStemmer, FrenchSnowballStemmer, GermanSnowballStemmer, HungarianSnowballStemmer, ItalianSnowballStemmer, NorwegianSnowballStemmer, PortugueseSnowballStemmer, RomanianSnowballStemmer, RussianSnowballStemmer, SpanishSnowballStemmer, SwedishSnowballStemmer, TurkishSnowballStemmer

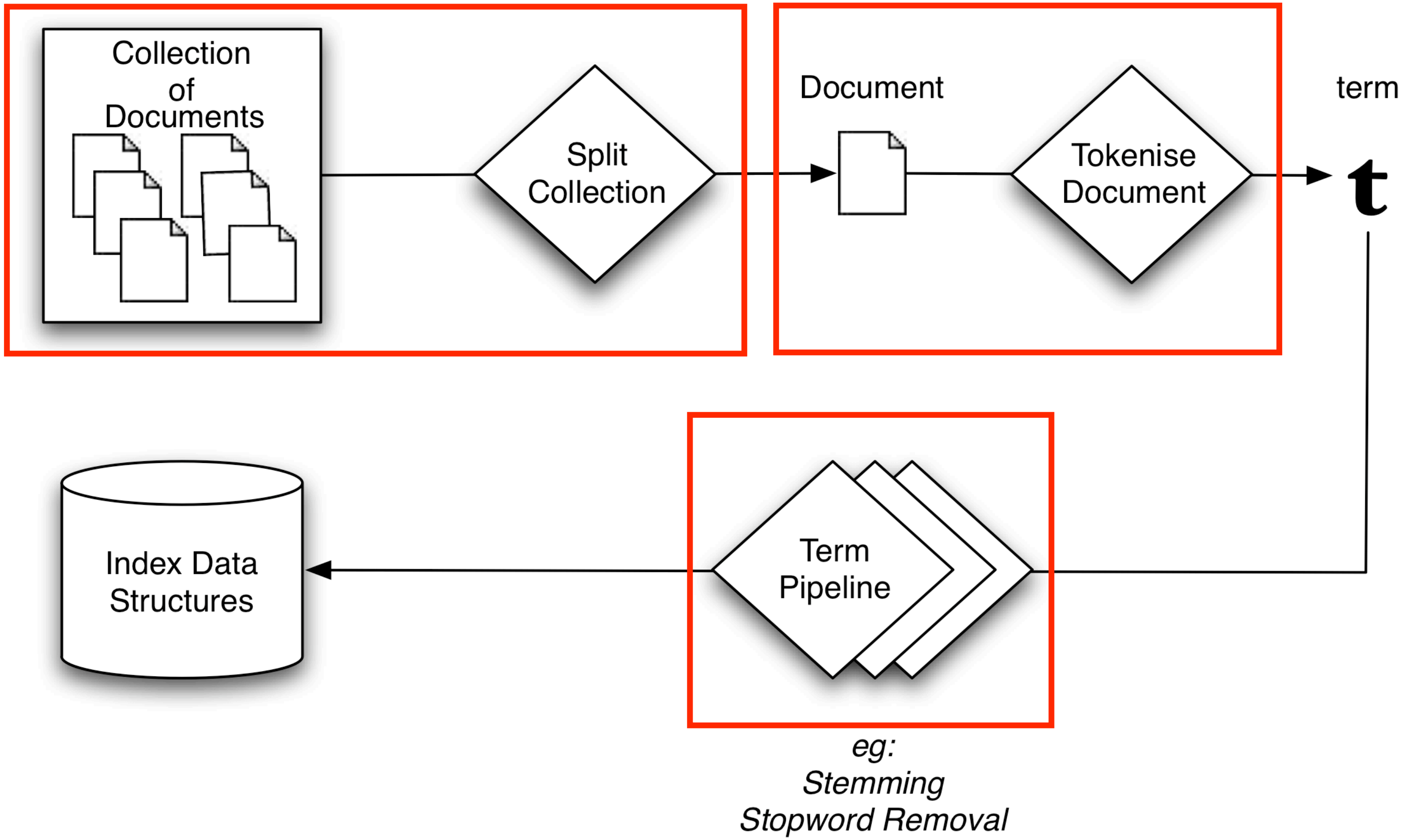
Extending Indexing: *"But Terrier doesn't do…X?"*

- Terrier has been designed to make it simple to add support for X

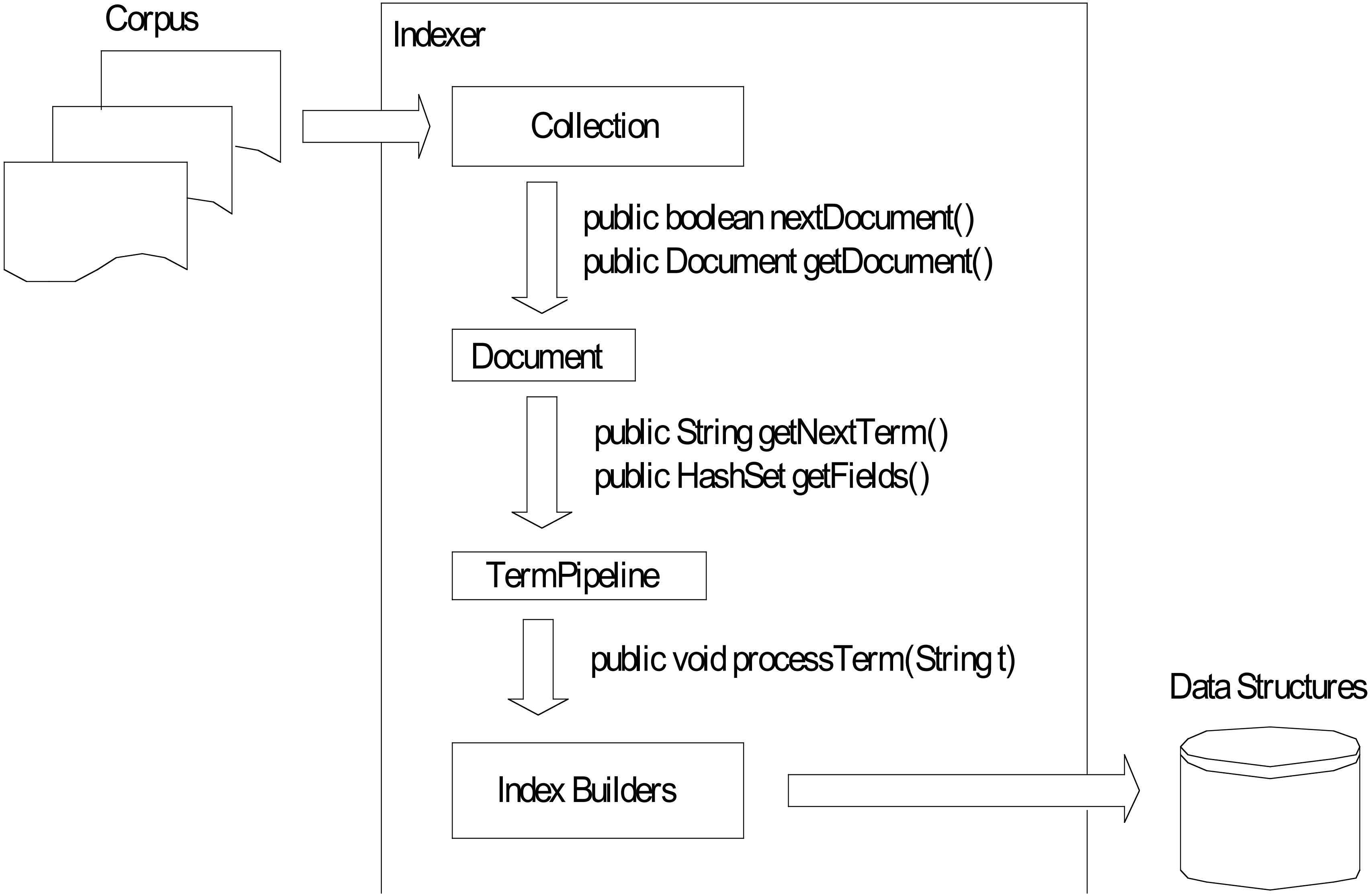
- How do I index a new type of Collection?
  - Database, email, RSS feeds
  - Implement Collection & Document interfaces
- How can I expand a document using Wordnet to contain synonyms?
  - Implement Term Pipeline interface

- And contribute back to platform for next release… ;-)

Indexing Architecture



Indexing API



Corpus

Indexer

Collection

public boolean nextDocument()
public Document getDocument()

Document

public String getNextTerm()
public HashSet getFields()

TermPipeline

public void processTerm(String t)

Index Builders

Data Structures

Collection and Document

- If you want to parse your own collection, you need to:
  - implement the `Collection` interface for obtaining documents from the collection

```
public Document getDocument();
public boolean nextDocument();
public String getDocid();
public boolean endOfCollection();
```

  - implement the `Document` interface for parsing the documents

```
public String getNextTerm();
public boolean endOfDocument();
```

- See also:
  - doc/indexing.html
  - doc/javadoc/uk/ac/gla/terrier/indexing/Collection.html
  - doc/javadoc/uk/ac/gla/terrier/indexing/Document.html
- Examples in package `uk.ac.gla.terrier.indexing`:
  - TRECCollection -> TRECDocument
  - TRECUTFCollection -> TRECUTFDocument
  - SimpleFileCollection -> FileDocument; HTMLDocument; PDFDocument; MSWordDocument
  - SimpleXMLCollection

Case Study: Indexing RSS Feeds

- Brief: Create a News Search + Aggregation System
- Problem: I want to download and index a list of RSS feeds
- Solution:
  - Terrier can fetch files directly from HTTP **v2.1**
  - Use ROME RSS/Atom parser: https://rome.dev.java.net
  - Implement Collection and Documents RSS objects (see Handout)

Zoom on RSS code

- Problem: Download RSS feeds
  - Use Files class - read/writes files **and** HTTP URLs
  - `Files.openFileReader("http://rss.bbc.co.uk/…");` **v2.1**
- Problem: Parse RSS feed
  - Use ROME parser
- Problem: Tokenise Text from RSS feed
  - Subclass TRECDocument, pass text from ROME to TRECDocument

Term Pipeline

- When terms are indexed, they are passed through the TermPipeline
  - You can implement your own TermPipeline objects
  - Alter/remove/add terms as they pass through the term pipeline

- Examples found in package `uk.ac.gla.terrier.terms`
  - Stemming, Removing stopwords, Noun phrase extraction, etc etc

```
public class DumpTerm implements TermPipeline {
  TermPipeline next = null;
  public DumpTerm(TermPipeline next) {
    this.next = next;
  }
  public void processTerm(String t) {
    if (t == null)
      return;
    System.err.println("term: "+t); //display term
    next.processTerm(t); //pass onto next term pipeline object
  }
}
```

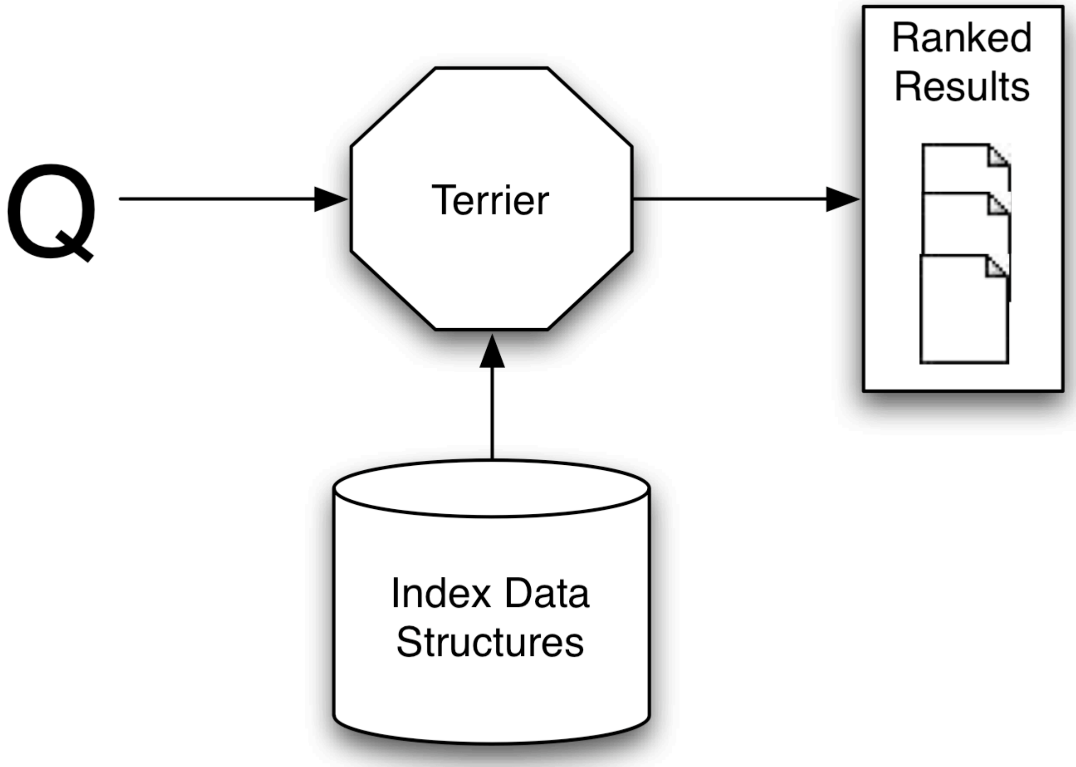
End of Part 1

Coffee is on Level 5 upstairs

We resume at 4pm

Retrieval in IR

- Retrieving documents:



Retrieval Overview

- Background:
  - Document Weighting Models
  - Query expansion (QE)
- Experimenting and Research with Terrier
  - TREC experimentation
- Extending Terrier retrieval
  - Changing the ranking
  - Getting statistics
- Retrieval examples
  - Document Priors
  - Opinionated Document Retrieval

Ranking in IR

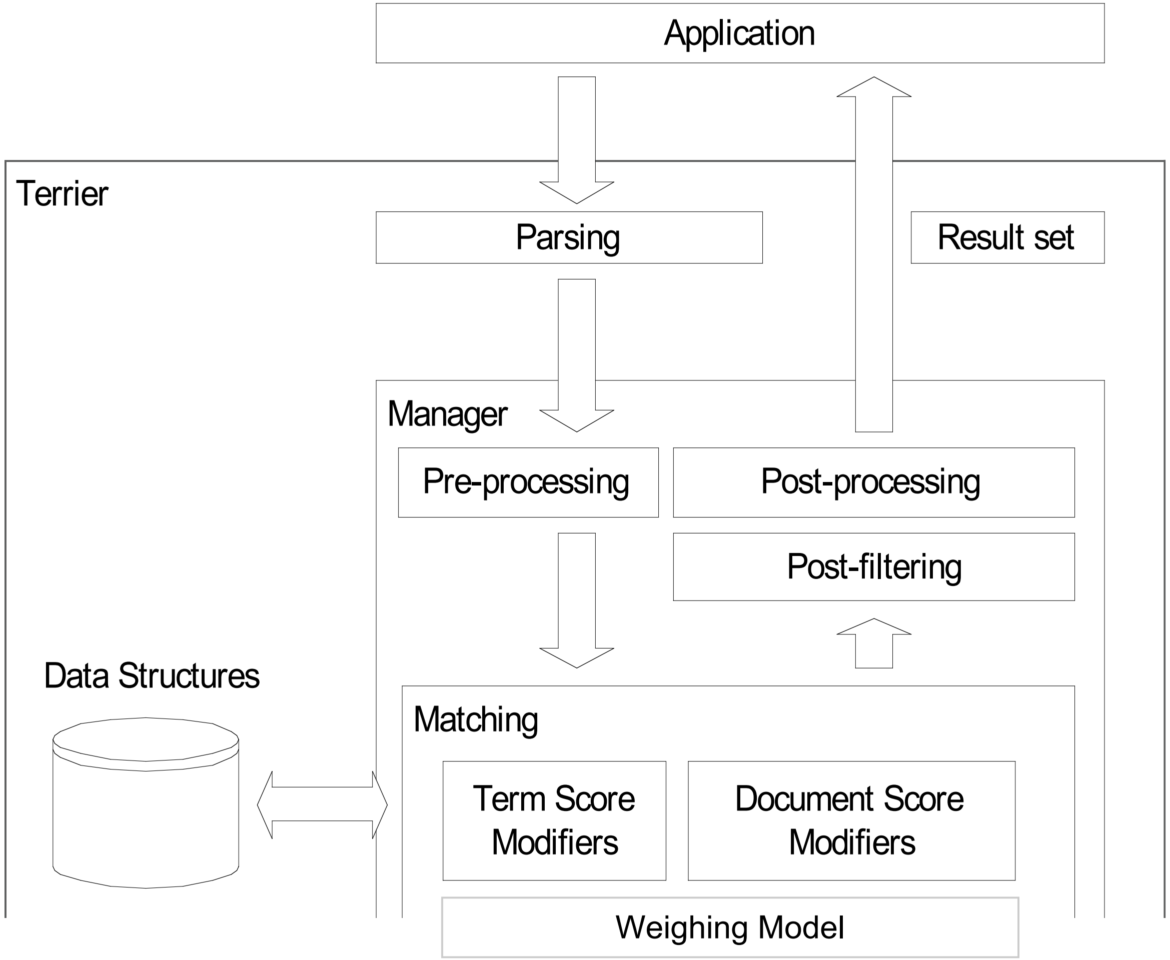
- The IR system ranks documents in response to a query
  - The documents are ranked in decreasing order of predicted relevance to the user's query
- Query term occurrences in documents are scored to obtain the relevance score value of the document to the query:

$$Score(d,Q)$$

Retrieval: More Details

Query

1. Parsing (Tokenisation)
2. Stemming/stopwords
3. Matching & scoring
4. Post-processing/filtering
5. Application rendering



Ranking Process

- Tokenise the user’s query
- Retrieve documents matching query terms using Inverted Index
- Score retrieved documents, and sort by decreasing score
- Present results to the user

Scoring Documents

- A simple model of scoring documents to a query is TF.IDF:

$$score(d,Q) = \sum_{t \in Q} tf \cdot \log_2 \frac{N}{N_t}$$

- Also Language Modelling (Hiemstra)

$$p(d \mid Q) \propto p(Q \mid d) p(d)$$

=> $score(d,Q) = \sum_{t \in Q} w(t,d) = \sum_{t \in Q} \log_2 \left(1 + \frac{\lambda \cdot tf \cdot T_c}{(1-\lambda) \cdot TF \cdot l_d} \right)$

Weighting Models in Terrier

- Terrier provides many state-of-the-art document weighting models:
  - TF-IDF (with length normalisation, aka BM11)
  - Lemur’s TF-IDF
  - Okapi BM25
  - Hiemstra and Ponte&Croft Language Models
  - Various Divergence from Randomness (DFR) models

Divergence from Randomness (DFR)

- The DFR paradigm is a generalisation of Harter's 2-poisson indexing model

- The DFR approach is based on a simple idea:
  - *"The more the divergence of the within-document term-frequency from its frequency within the collection, the more the information carried by the word t in the document d"*

Document Urn
Collection Urn

- A query term *w(t,d)* is scored by how different its term distribution in the document *d* is, compared to the whole collection (where the distribution of the term is assumed to be random)
  - w(t,d) is calculated using various probabilistic divergence measures

Weighting Models in Terrier

- All in `uk.ac.gla.terrier.matching.models`
- Example Weighting Model:

```
class SimpleTFIDF extends WeightingModel
{
 public double score(tf, doclength)
 {
  return tf * Math.log(
     numberOfDocuments/documentFrequency)
     /Math.log(2);
 }
}
```

What is Query Expansion (QE)?

- Relevance feedback
  - Taking evidence on relevant and irrelevant document to reformulate the query
  - Can be interactive (with a user), or blind [pseudo-relevance fedback] without the user
- Terrier's QE is a pseudo-relevance fedback technique that
  - Expands the query by adding new query terms
  - Re-weights the query terms

- It re-formulates the user query, so that to achieve a better retrieval performance

Expanding the query

- The added query terms are meant to be related to the topic
- QE brings more information to the query
- It helps to retrieve more relevant documents

    BUT it can also bring noise

Illustrative Example

- TREC Query: *Scottish highland games*

- What are the possible expanded query terms?

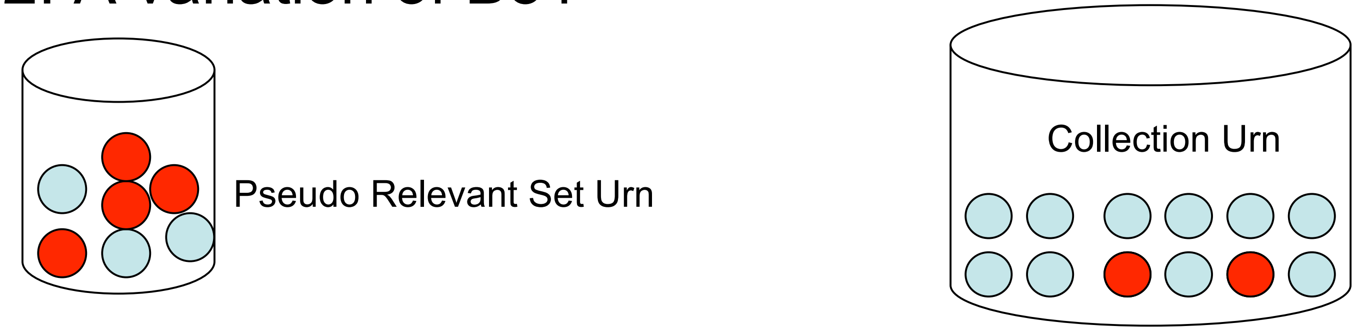
- The expanded query (Using one of Terrier's QE mechanisms and Weak Stemming):
  - *Scottish highland games* Ligonier kilt caber clan toss Scot tartan grandfather artist heavy tradition dance Celtic dancer athlete heather competitor

- In the expanded query (using the relevance assessment)
  - These terms are helpful: Ligonier kilt caber clan toss Scot tartan
  - These terms bring noise: grandfather artist heavy
  - The rest of the added query terms are neutral, e.g. dancer, tradition

Terrier's QE Models

- Basic idea: infer how informative a term is by the divergence of the term's distribution in the pseudo relevance set from a random distribution
  - Identical concept to document weighting

- Terrier deploys three QE models:
  - KL: Kullback-Leibler divergence
  - Bo1: Bose-Einstein statistics
  - Bo2: A variation of Bo1



Problems With QE

1. Can be detrimental if the pseudo relevance set is poor
   - The added query terms are unlikely to be related to the topic
2. May not be rewarding if the number of relevant documents is small
   - Adding more query terms cannot bring many relevant documents
   - e.g. query *Homepage of Terrier platform* has only a unique relevant document. It is not helpful to expand the query

SEE ALSO: Learning to Estimate Query Difficulty with Applications to Missing Content Detection and Distributed Information Retrieval. Yom-Tov et al. SIGIR 2005.

Practical Retrieval using Terrier

Terrier has two retrieval applications:

- Interactive Retrieval
- TREC-like batch retrieval for experimentation

Interactive Retrieval with Terrier

- Run the script `bin/interactive_terrier.sh`

```
[toto@boano terrier]$ bin/interactive_terrier.sh
Setting TERRIER_HOME to /users/toto/tmp/terrier
Setting JAVA_HOME to /local/java/linux/jdk1.5.0
time to intialise indexes : 0.269
Please enter your query: cellular
1 : cellular
weighting model: PL2c1.0
1: cellular with 451 documents (TF is 1216).
number of retrieved documents: 451

        Displaying 1-451 results
0 AP900725-0227 210549 9.964763620794955
1 AP900523-0277 196850 9.912967288227696
...
449 AP901009-0235 225761 1.7048076030739692
450 AP900808-0105 213356 1.4994280197490206
Please enter your query: <return>
[toto@boano terrier]$
```

More Interactive Terrier

- To choose a weighting model, specify property

```
interactive.model=TF_IDF
```

v2.1

- Interactive Terrier is not designed for experimentation
- It is a small application, ideal for
  - Debugging
  - Example Querying code
  - Using as a base to your IR application powered by Terrier

Query Language (1)

- Terrier has an advanced query language with the following operators

```
t1 t2 : retrieves documents with either t1 or t2
t1^2.3: the weight of t1 is boosted to 2.3
+t1 -t2: retrieve docs with t1 but not t2

“t1 t2”: retrieve docs with the phrase 't1 t2'
“t1 t2”~n: retrieve docs where the terms t1, t2
  appear within the given distance
```

  - Requires indexing with blocks

Query Language (2)

- More query language operators
  - `+(t1 t2): both terms t1 and t2 are required`
  - `field:t1 : retrieves docs where t1 appears in the specified field`
  - `control:on/off : enables or disables a given control`
    - → like properties, but for query settings
    - → enable query expansion with `qe:on`
- Controls are used to control the querying process on a per-query basis

```
querying.default.controls=c:1.0,start:0,end:999
```

- To avoid potential security problems, a list of allowed control is defined as follows:

```
querying.allowed.controls=c,scope,qe,qemodel,start,end
```

Experimentation in IR

- Is a new technique/weighting model any good?
  - Evaluate using standard test collection

- Terrier is **ideal** for TREC-like experimentation and evaluation
  - Generate runs: the retrieved set of documents for a pre-defined set of queries

Checklist: Running an experiment

✔ 1. Index your test corpus
2. Select test collection: topics + relevance assessments
3. Do baseline ‘run’ and evaluate
4. Implement and enable new technique
5. Do new ‘run’
6. Has retrieval performance improved?

Batch Retrieval with Terrier

- Example TREC topic

```
<top>
<head> Tipster Topic Description
<num> Number:  051
<dom> Domain:  International Economics
<title>   Airbus Subsidies
<desc> Description:
Document will discuss government assistance to Airbus Industrie, or mention a trade
   dispute between Airbus and a U.S. aircraft producer over the issue of subsidies.
<narr> Narrative:
A relevant document will cite or discuss assistance to Airbus Industrie by the French,
   German, British or Spanish government(s), or will discuss a trade dispute between
   Airbus or the European governments and a U.S. aircraft producer, most likely Boeing Co.
   or McDonnell Douglas Corp., or the U.S. government, over federal subsidies to Airbus.
</top>
```

- Specify the topics file in *etc/trec.topics.list*

```
echo /path/to/topics > etc/trec.topics.list
```

OR

```
trec.topics = /path/to/topics
```

Configuring Batch Retrieval (1)

- In the properties file, specify whether to use short, normal, or long queries

```
#short: title only
TrecQueryTags.doctag=TOP
TrecQueryTags.idtag=NUM
TrecQueryTags.process=TOP,NUM,TITLE
TrecQueryTags.skip=DESC,NARR

#normal: title + description
TrecQueryTags.doctag=TOP
TrecQueryTags.idtag=NUM
TrecQueryTags.process=TOP,NUM,TITLE,DESC
TrecQueryTags.skip=NARR

#long: title + description + narrative
TrecQueryTags.doctag=TOP
TrecQueryTags.idtag=NUM
TrecQueryTags.process=TOP,NUM,TITLE,DESC,NARR
```

```
<top>
<num> Number: TOPIC-X1
<title> . . . </title>
<desc> . . . </desc>
<narr> . . . </narr>
</top>
```

Configuring Batch retrieval (2)

- Set the weighting models to use
  - Divergence From Randomness (DFR) framework models, such as PL2
  - Classical models, such as *tf-idf*, BM25

```
echo PL2 > etc/trec.models
```

- You can specify more than one weighting models in *etc/trec.models*

```
echo PL2 > etc/trec.models
echo In_expB2 >> etc/trec.models
echo Hiemstra_LM >> etc/trec.models
```

Let's (batch) retrieve!

- Using trec_terrier.sh script to retrieve all queries

```
[toto@boano terrier]$ bin/trec_terrier.sh -r
 Setting TERRIER_HOME to /users/toto/tmp/terrier
 Setting JAVA_HOME to /local/java/linux/jdk1.5.0
 time to intialise indexes : 0.226
 Extracting queries from 51-200.topics
 051 : airbus subsidies
 processing query 051
 time to process query: 0.262
 ...
```

- Runs are stored in the folder var/results/, numbered…
  - E.g. InL2_c1.0_0.res then InL2_c1.0_1.res etc
  - InL2_c1.0_0.settings contains the properties and other settings used by Terrier, to help recreate runs later

Runs with Query Expansion

- Automatically extracts informative terms from top ranked documents and adds them to the query

- Use query expansion when batch querying

```
bin/trec_terrier.sh -r -q
```

- How to specify the query expansion model?

```
echo Bo1 > etc/qemodels
```

- Available models: Bo1, Bo2 and KL
- Number of top-ranked documents: `expansion.documents`
- Number of terms to extract: `expansion.terms`

Evaluation

- *How well did the system perform?*

- Specify the qrels file with the relevance assessments to use in *etc/trec.qrels*

```
echo /path/to/qrels > etc/trec.qrels
```

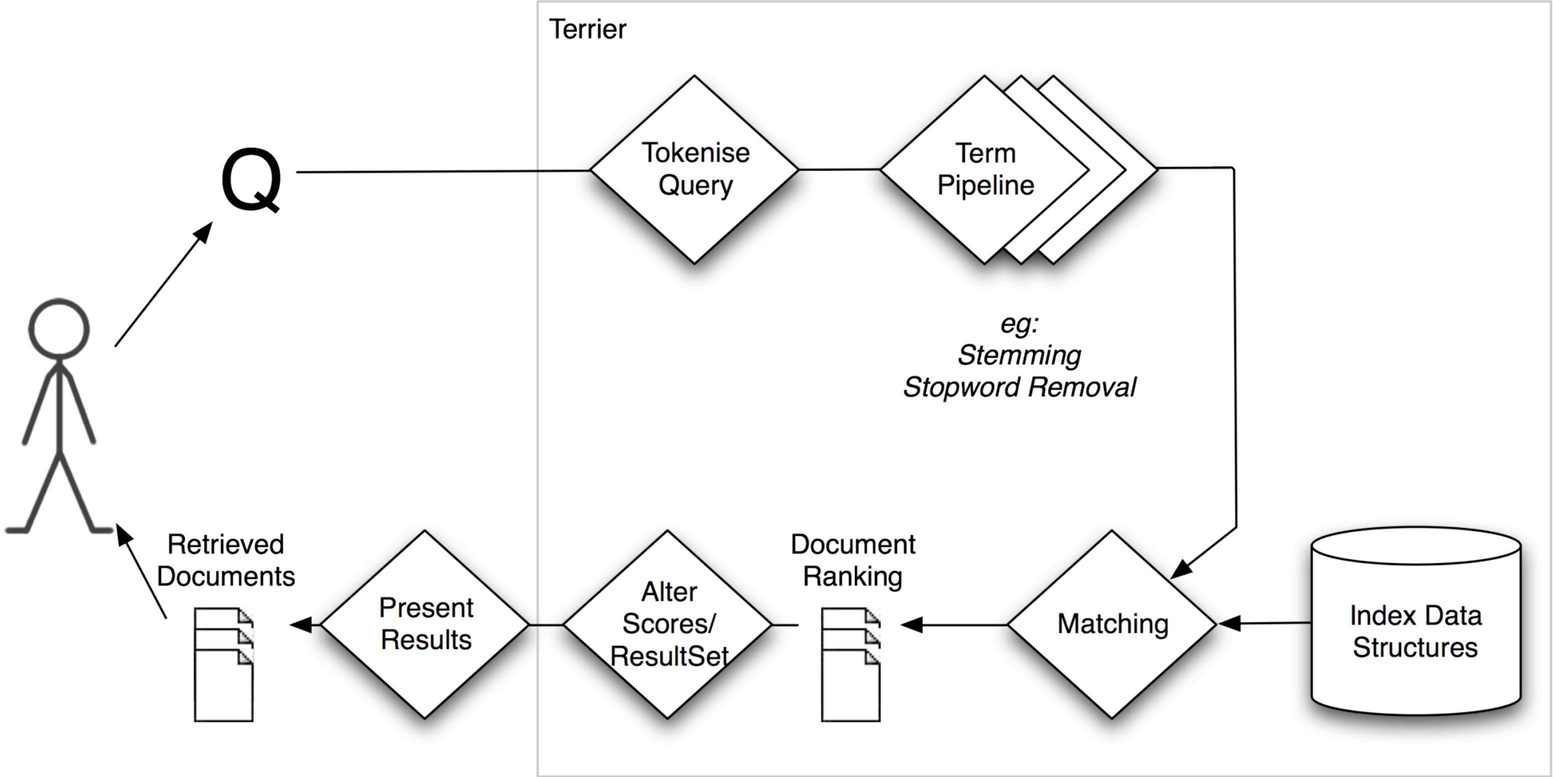
- Evaluate all the result files in the var/results directory

```
[toto@boano]$ bin/trec_terrier -e
Setting TERRIER_HOME to /users/toto/tmp/terrier
/users/toto/tmp/terrier/var/results/InL2c1.0_0.res
Average Precision: 0.0806
Time elapsed: 0.26 seconds
```

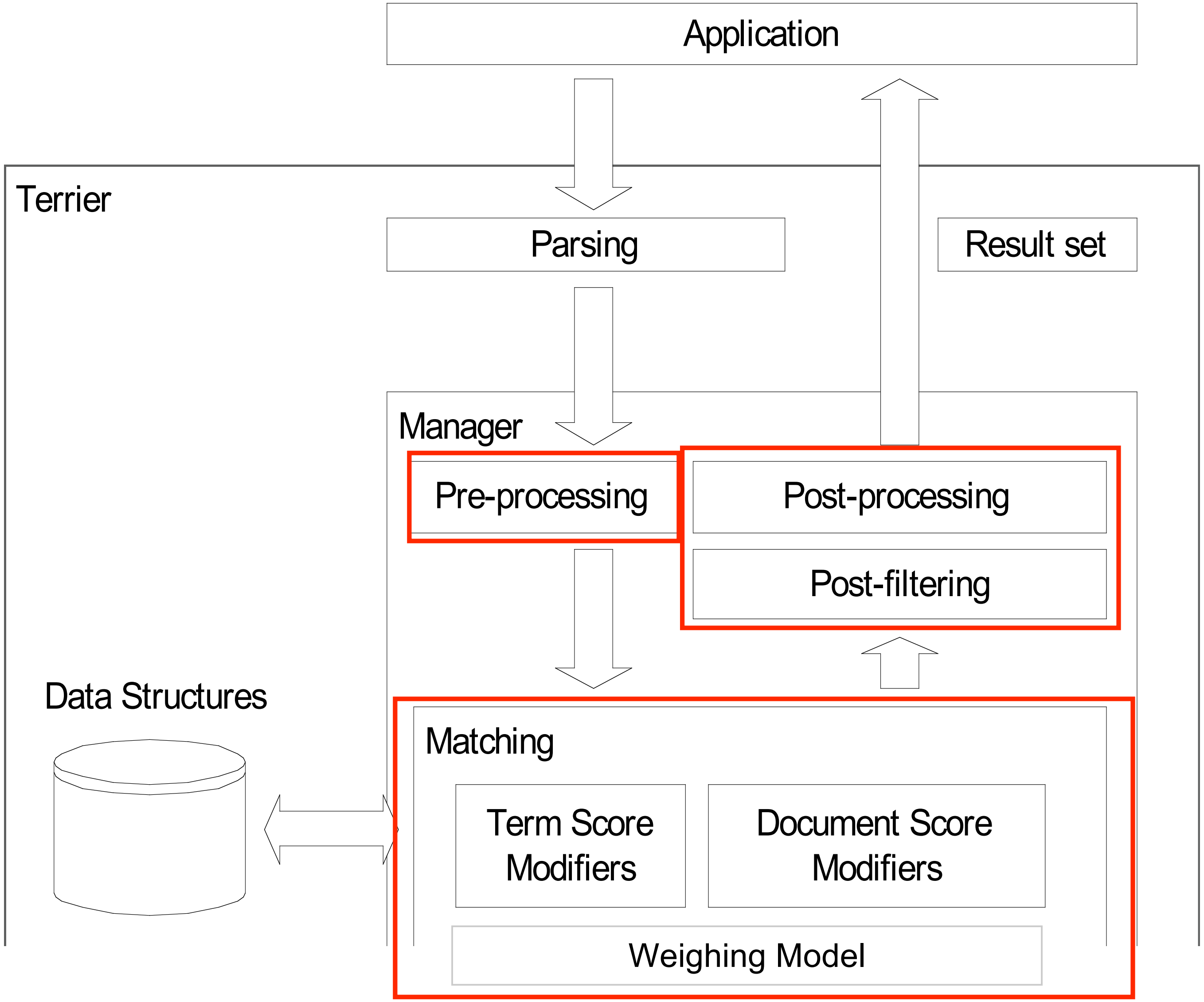
- `InL2c1.0_0.eval` contains usual evaluation measures, P@10 P@20 etc.
  - This is not TREC_EVAL though

Extending Terrier: how I do implement X?

- Terrier has a rich API that allows the result set of documents for a query to be altered
  - – In various ways
  - – At various phases of the retrieval



Retrieving API



Application

Terrier

Parsing

Result set

Manager

Pre-processing

Post-processing

Post-filtering

Data Structures

Matching

Term Score Modifiers

Document Score Modifiers

Weighing Model

Before Matching

- Parsing the query
- Pre-processing the parsed query for matching (TermPipeline)
  - Recall from indexing
  - Usually Stopword removal, Stemming

Matching

- The class Matching takes as input:
  - A query
  - The data structures
  - Weighting model
- Retrieves and ranks documents according to a weighting model
  - Returns a ResultSet
- Found in package `uk.ac.gla.terrier.matching`
- The output of the matching can be modified by applying Term Score Modifiers and Document Score Modifiers

Weighting Models for Matching

- Abstract class Model
  - WeightingModel
    - BB2, IFB2, I(n_e)C2, I(n_e)B2, InL2, PL2, DLH, DFR_BM25
    - BM25
    - tf-idf
    - Hiemstra_LM, LemurTF_IDF

```
public class MyModel extends WeightingModel {
  . . .
  public final String getInfo() { return "MyModel"; }
  public double score(double tf, double length) { ...; return score; }
  public double score(double tf,
                      double length,
                      double n_t,
                      double F_t,
                      double keyFrequency) { ...; return score; }
}
```

- Found in package `uk.ac.gla.terrier.matching.models`

Score Modification during Matching

- Take the query `title:t1 “t2 t3”`
  - Retrieve documents where t1 must occur in title field, and “t2 t3” occur as a phase
- Two forms of score modification occur during matching for this query
  - TermScoreModifier: For each document retrieved, does t1 occur in title field?
  - DocumentScoreModifier: For each document retrieved, does t2 & t3 both occur in it, and as a phase?
- We can also use score modification to implement advanced ranking functionality
  - e.g Prior integration

Term Score Modifiers

- Alters the given score to a term in a retrieved document
- Query specific TSMs *(enabled automatically when required for a query):*
  - TermInFieldModifier
  - RequiredTermModifier

  The query operators `field:term`, `+term` result in applying the term score modifiers `TermInFieldModifier` and `RequiredTermModifier` respectively

- Static TSMs: when you want to include additional evidence:
  - Specify the term score modifiers to apply with the following property

    `matching.tsms=FieldScoreModifier`

- Found in package `uk.ac.gla.terrier.matching.tsms`

Document Score Modifiers

- Alters the given scores to a retrieved document
- Query specific DSMs *(enabled automatically when required for a query):*
  - PhraseScoreModifier

  The query `“t1 t2”` returns only documents that match the phrase, by applying the document score modifier `PhraseScoreModifier` as a filter

- Static DSMs: To change the retrieval scores of the retrieved documents, Specify the document score modifiers to apply with the following property:

  ```
  matching.dsms=BooleanFallback
  ```

  ```
  matching.dsms=BooleanScoreModifier
  ```

- Found in package `uk.ac.gla.terrier.matching.dsms`

ResultSet

- Matching returns the ResultSet
  - The documents returned by Matching for a query

- The ResultSet contains
  - Array of scores
  - Array of document ids (numerical document identifiers)
  - Array of flags that denote whether a query term occurred in a document

- Found in package `uk.ac.gla.terrier.matching`

After Matching

- Driven by user applications, e.g.:
  - Controlling the type/location of the document, (filetype:pdf, site:gla.ac.uk)
  - Running QE
  - Two phases: Post Processing & Post Filtering
- Post-processing
  - Alters the result set after matching has finished
  - e.g. Query expansion expands the query, then runs matching again with the new query
- Post-filtering
  - Optional (last-ditch) filtering of documents
- Found in package `uk.ac.gla.terrier.querying`

Extending Retrieval Use Cases: Document Priors

- Aim: retrieve high quality as well as relevant documents
- Assumption: You have a file containing PageRank scores for each document in the collection
- Integrate with retrieval score as

$$score(d,Q) = score(d,Q) * prior(d)$$

- How: Use a DocumentScoreModifier
  - Modify retrieval scores at end of Matching

Example: Prior Integration

```
class IntegrateStaticScore implements DocumentScoreModifier
{
    //populated one for each document in collection
    double prior[] = new double[];
    public boolean modifyScores(Index I,
       MatchingQueryTerms mqt, ResultSet r)
    {
        double[] scores = r.getScores();
        int[] docids = r.getDocids();
        for(int i=0;i<scores.length;i++)
        {
            scores[i] = scores[i] * prior[docids[i]];
        }
        return true;
    }
}
```

Extending Retrieval: Opinionated Document Retrieval

- Aim: retrieve not just relevant, but opinionated documents
  - Cf. TREC 2006-2008 Blog Tracks
- Approach:
  - Offline: Score all documents in the collection using a large query containing a list of opinionated terms
  - Use these document opinionated scores as a prior, as before

Extending Retrieval: Working with Index Structures

- The Index object provides access to all index structures
  - Index.createIndex(); //load an existing index
  - index.getInvertedIndex(); //returns inverted index
- You can add more index structures to an index:
  - index.addIndexStructure(name, objects); index.flush();
  - index.getIndexStructure(name);

V2.0

Statistics Examples

How many documents does term X occur in?

```
Index index = Index.createIndex();
Lexicon lex = index.getLexicon();
LexiconEntry le =
   lex.getLexiconEntry("X");
if (le != null)
   System.out.println("Term X occurs
   in "+ le.n_t + " documents");
else
   System.out.println("Term X does
   not occur");
Double probabilityX = (le == null)
 ? 0.0d
 : le.TF /
   index.getCollectionStatistics().g
   etNumberOfTokens()
```

What terms occur in the 10th document?

```
Index index = Index.createIndex();
DirectIndex di =
   index.getDirectIndex();
Lexicon lex = index.getLexicon();
int[][] postings = di.getTerms(10);
for(int i=0;i<postings[0].length; i++)
{
   LexiconEntry le =
   lex.getLexiconEntry(
   postings[0][i]);
   System.out.print(le.term + " with
   frequency "+ postings[1][i]);
}
```

Statistic Examples (2)

- What documents does term Z occur in?

```
Index index = Index.createIndex();
InvertedIndex di = index.getInvertedIndex();
DocumentIndex doi = index.getDocumentIndex();
Lexicon lex = index.getLexicon();
LexiconEntry le = lex.getLexiconEntry( "Z" );
int[][] postings = ii.getDocuments(le);
for(int i=0;i<postings[0].length; i++)
{
   System.out.println(doi.getDocumentNumber(postings[0][i])
        + " with frequency "+ postings[1][i]);
}
```

Data Structures Builders

- Builders for the 4 main data structures
  - Lexicon and Lexicon index : stores the vocabulary
  - Document Index : stores information about documents
  - Direct File (used for fast query expansion) : stores the terms for each document
  - Inverted File : stores the postings lists
- Found in package `uk.ac.gla.terrier.structures.indexing`

* Includes the size of a lexicon with global statistics

Lexicon

- Stores information about the vocabulary – which terms are in collection

```
public boolean findTerm(int termId)
public boolean findTerm(String term)

public String getTerm()
public int getTermId()
public int getTF()
public int getNt()
```

```
public long getStartOffset()
public byte getStartBitOffset()
public long getEndOffset()
public byte getEndBitOffset()

public int getNumberOfLexiconEntries()
```

- Found in package `uk.ac.gla.terrier.structures`
- Using lexicon as a random access file
  - `Lexicon`
- Using lexicon as a stream
  - `LexiconInputStream`
  - `LexiconOutputStream`

Lexicon	**Term** (20 bytes), **Term id** (4 bytes), **Document frequency** (4 bytes), **Term Frequency** (4 bytes), **End byte offset in inverted file** (8 bytes), **End bit offset in inverted file** (1 byte)
Lexicon Index	**Offset of an entry in the lexicon** (8 bytes)

Document Index

- Stores information about documents

```
public String getDocumentNumber(int docid)
public int getDocumentId(String docno)
public int getDocumentLength(int docid)
public int getDocumentLength(String docno)

public byte getStartBitOffset()
public byte getEndBitOffset()
public long getStartOffset()
public long getEndOffset()

Public int getNumberOfDocuments()
```

- Found in package `uk.ac.gla.terrier.structures`
- Using document index as a random access file
  - `DocumentIndex`
  - `DocumentIndexInMemory`
  - `DocumentIndexEncoded`
- Using document index as a stream
  - `DocumentIndexInputStream`

Document Index	**Document id** (4 bytes), **Document Length** (4 bytes), **Document number** (20 bytes), **End byte offset in direct file** (8 bytes), **End bit offset in direct file** (1 byte)

Direct Index

- Useful for fast query expansion or clustering
- Stores the terms that are contained in each document

```
public int[][] getTerms(int docid)
```

- The method `getTerms` returns a two dimensional array:

```
int[][] terms = getTerms(docid);
terms[0] //contains term identifiers
terms[1] //contains term frequencies in the document
terms[2] //is null, or contains field information if fields are indexed
```

- If blocks are indexed

```
terms[4] //contains the number of blocks in which a term appears
terms[5] //contains the block identifiers
```

- (The length of terms[5] is different from the length of terms[4])
- Found in package `uk.ac.gla.terrier.structures`
- Using direct index as a random access file

```
DirectIndex, BlockDirectIndex
```

- Using direct index as an input stream

```
DirectIndexInputStream, BlockDirectIndexInputStream
```

Direct Index	**Term id gap** (gamma code), **Term frequency** (unary code), **Fields** (# of fields bits), **Block frequency** (unary code), **[Block id gap** (gamma code)**]**

Inverted Index

- Stores the posting lists

```
public int[][] getDocuments(int termId)
```

- The method `getDocuments` returns a two dimensional array:

```
int[][] postings = getDocuments(termId);
postings[0] //contains document identifiers
postings[1] //contains term frequencies in the document
postings[2] //is null, or contains field information if fields are indexed
```

- If blocks are indexed

```
postings[4] //contains the number of blocks in which a term appears
postings[5] //contains the block identifiers
```

- The length of postings[5] is different from the length of postings[4]
- Found in package `uk.ac.gla.terrier.structures`
- Using inverted index as a random access file

```
InvertedIndex
BlockInvertedIndex
```

Inverted Index	**Document id gap** (gamma code), **Term frequency** (unary code), **Fields** (# of fields bits), **Block frequency** (unary code), **[Block id gap** (gamma code)**]**

Compiling Terrier

- To use your code with Terrier, add your jar file or your class folder to the CLASSPATH environment variable

- If you do need to alter the code in Terrier, then you have to recompile.

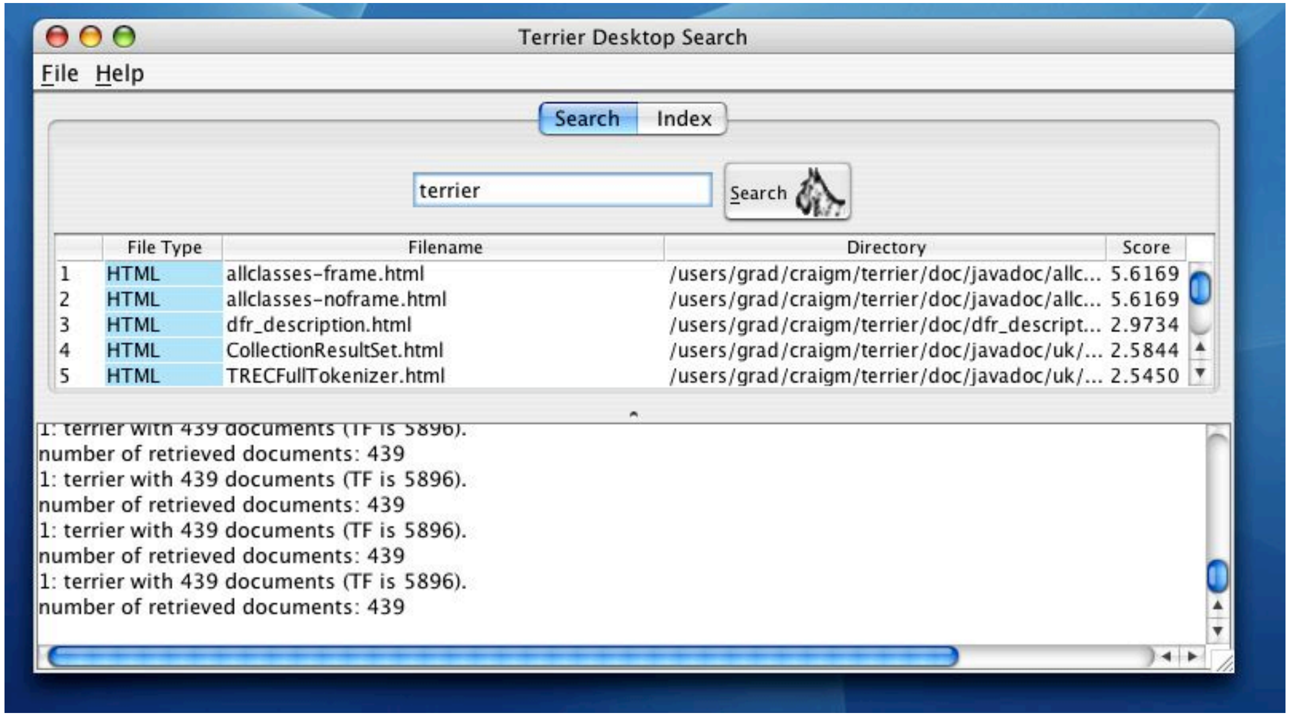
- `bin/compile.sh`
- `bin/compile.bat`
- `make clean compile`

- In Eclipse, you will need the Antlr plugin to compile Terrier

Putting it altogether

- Searching your desktop:
  - Terrier Desktop Search
- Java Swing GUI
- Comes with Terrier

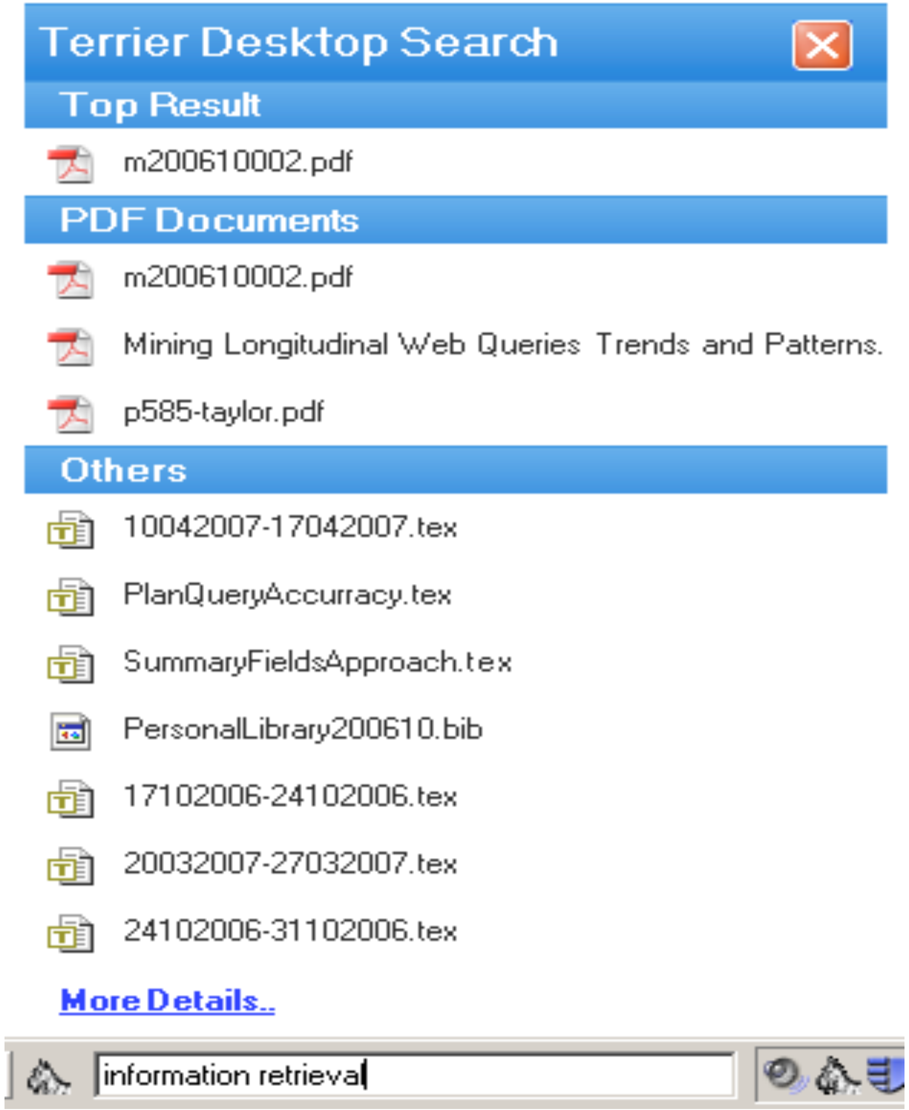
- SimpleFileCollection
  - FileDocument, PDFDocument, WordDocument, etc



	File Type	Filename	Directory	Score
1	HTML	allclasses-frame.html	/users/grad/craigm/terrier/doc/javadoc/allc...	5.6169
2	HTML	allclasses-noframe.html	/users/grad/craigm/terrier/doc/javadoc/allc...	5.6169
3	HTML	dfr_description.html	/users/grad/craigm/terrier/doc/dfr_descript...	2.9734
4	HTML	CollectionResultSet.html	/users/grad/craigm/terrier/doc/javadoc/uk/...	2.5844
5	HTML	TRECFullTokenizer.html	/users/grad/craigm/terrier/doc/javadoc/uk/...	2.5450

```
1: terrier with 439 documents (TF is 5896).
number of retrieved documents: 439
1: terrier with 439 documents (TF is 5896).
number of retrieved documents: 439
1: terrier with 439 documents (TF is 5896).
number of retrieved documents: 439
1: terrier with 439 documents (TF is 5896).
number of retrieved documents: 439
```

Improved Desktop Search



Terrier Desktop Search

Top Result

m200610002.pdf

PDF Documents

m200610002.pdf

Mining Longitudinal Web Queries Trends and Patterns.

p585-taylor.pdf

Others

10042007-17042007.tex

PlanQueryAccurracy.tex

SummaryFieldsApproach.tex

PersonalLibrary200610.bib

17102006-24102006.tex

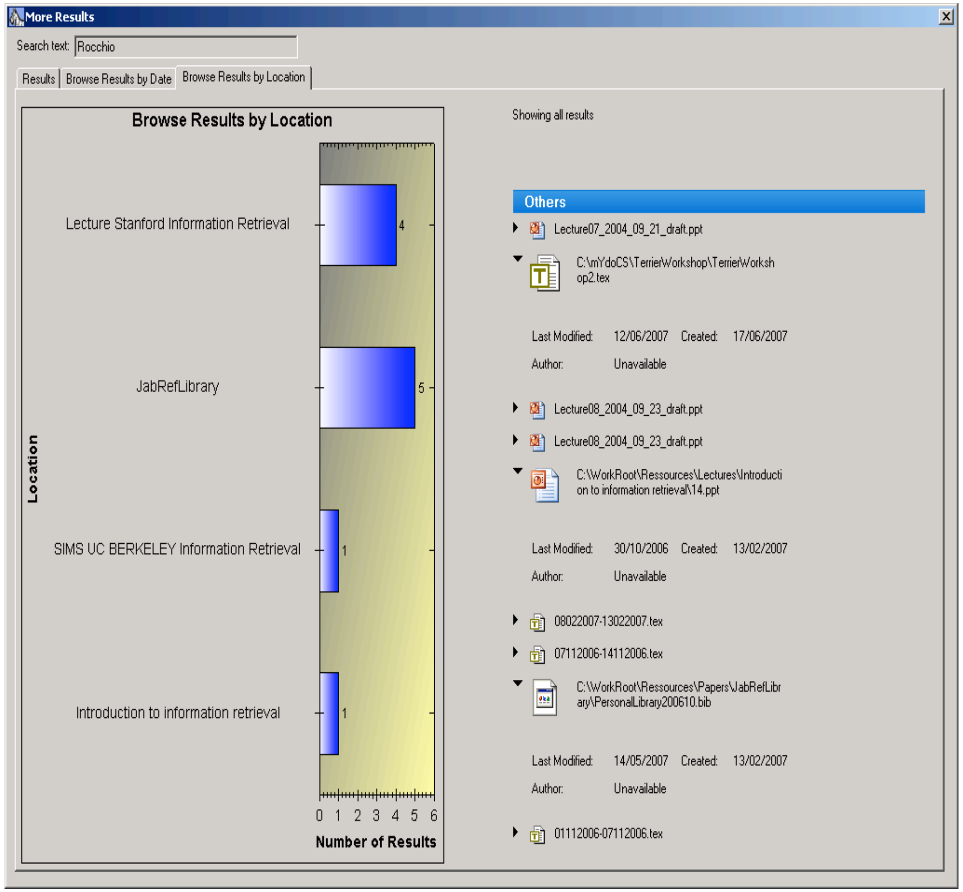
20032007-27032007.tex

24102006-31102006.tex

More Details..

information retrieval

- Improved Desktop Search built on Terrier
  - Integrated into Windows UI



More Results

Search text: Rocchio

Results | Browse Results by Date | Browse Results by Location

Browse Results by Location
Lecture Stanford Information Retrieval
4
JabRefLibrary
5
SIMS UC BERKELEY Information Retrieval
1
Introduction to information retrieval
1
Location
0 1 2 3 4 5 6
Number of Results

Showing all results

Others

Lecture07_2004_09_21_draft.ppt

C:\mYdoCS\TerrierWorkshop\TerrierWorkshop2.tex

Last Modified: 12/06/2007 Created: 17/06/2007

Author: Unavailable

Lecture08_2004_09_23_draft.ppt

Lecture08_2004_09_23_draft.ppt

C:\WorkRoot\Ressources\Lectures\Introduction to information retrieval\14.ppt

Last Modified: 30/10/2006 Created: 13/02/2007

Author: Unavailable

08022007-13022007.tex

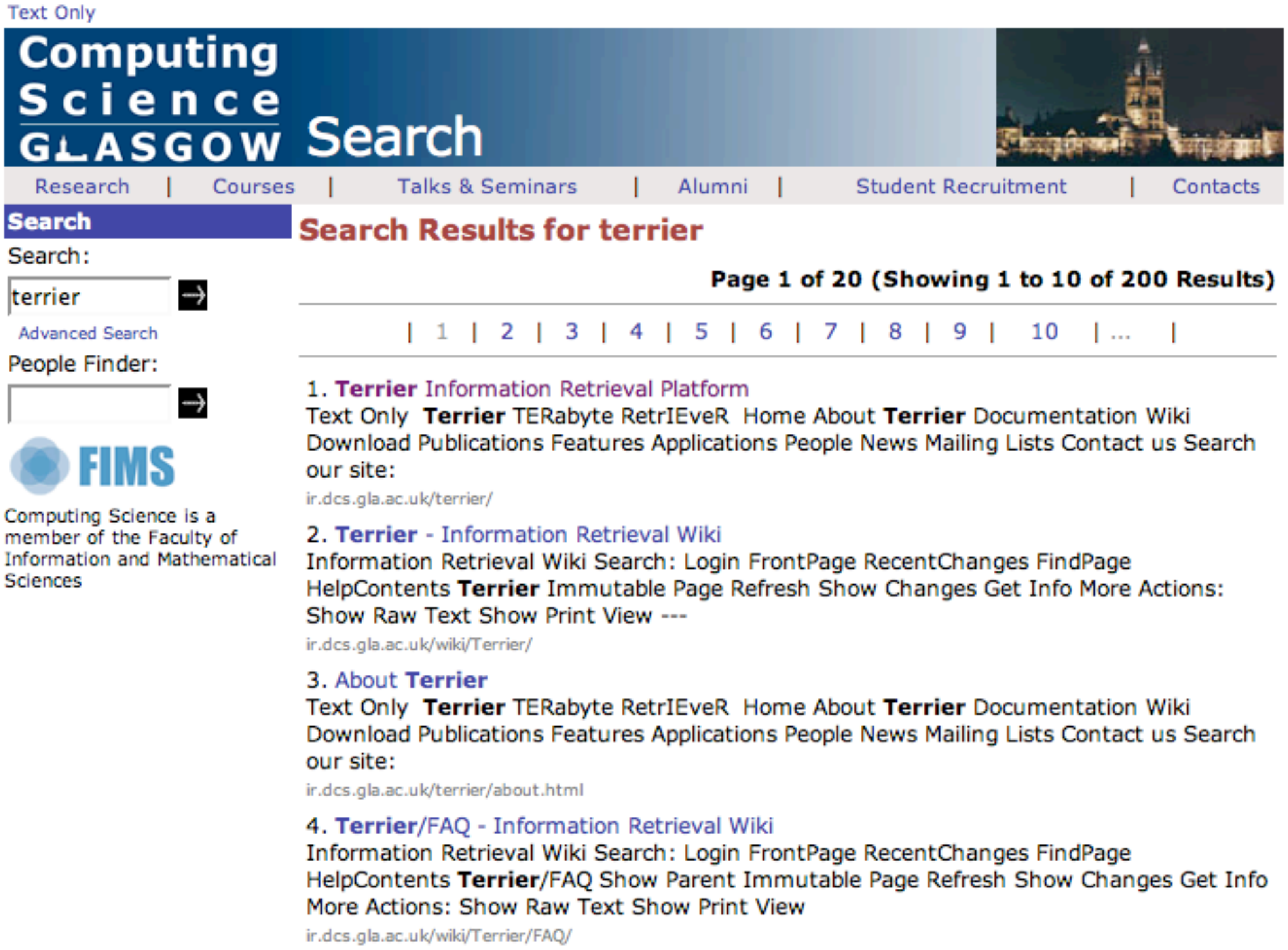
07112006-14112006.tex

C:\WorkRoot\Ressources\Papers\JabRefLibrary\PersonalLibrary200610.bib

Last Modified: 14/05/2007 Created: 13/02/2007

Author: Unavailable

01112006-07112006.tex



Text Only

Computing Science GLASGOW Search

Research | Courses | Talks & Seminars | Alumni | Student Recruitment | Contacts

- **Search**

Search:

terrier

Advanced Search

People Finder:

FIMS

Computing Science is a member of the Faculty of Information and Mathematical Sciences

Search Results for terrier

Page 1 of 20 (Showing 1 to 10 of 200 Results)

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | ... |

1. **Terrier** Information Retrieval Platform
Text Only **Terrier** TERabyte RetrIEveR Home About **Terrier** Documentation Wiki Download Publications Features Applications People News Mailing Lists Contact us Search our site:
ir.dcs.gla.ac.uk/terrier/

2. **Terrier** - Information Retrieval Wiki
Information Retrieval Wiki Search: Login FrontPage RecentChanges FindPage HelpContents **Terrier** Immutable Page Refresh Show Changes Get Info More Actions: Show Raw Text Show Print View ---
ir.dcs.gla.ac.uk/wiki/Terrier/

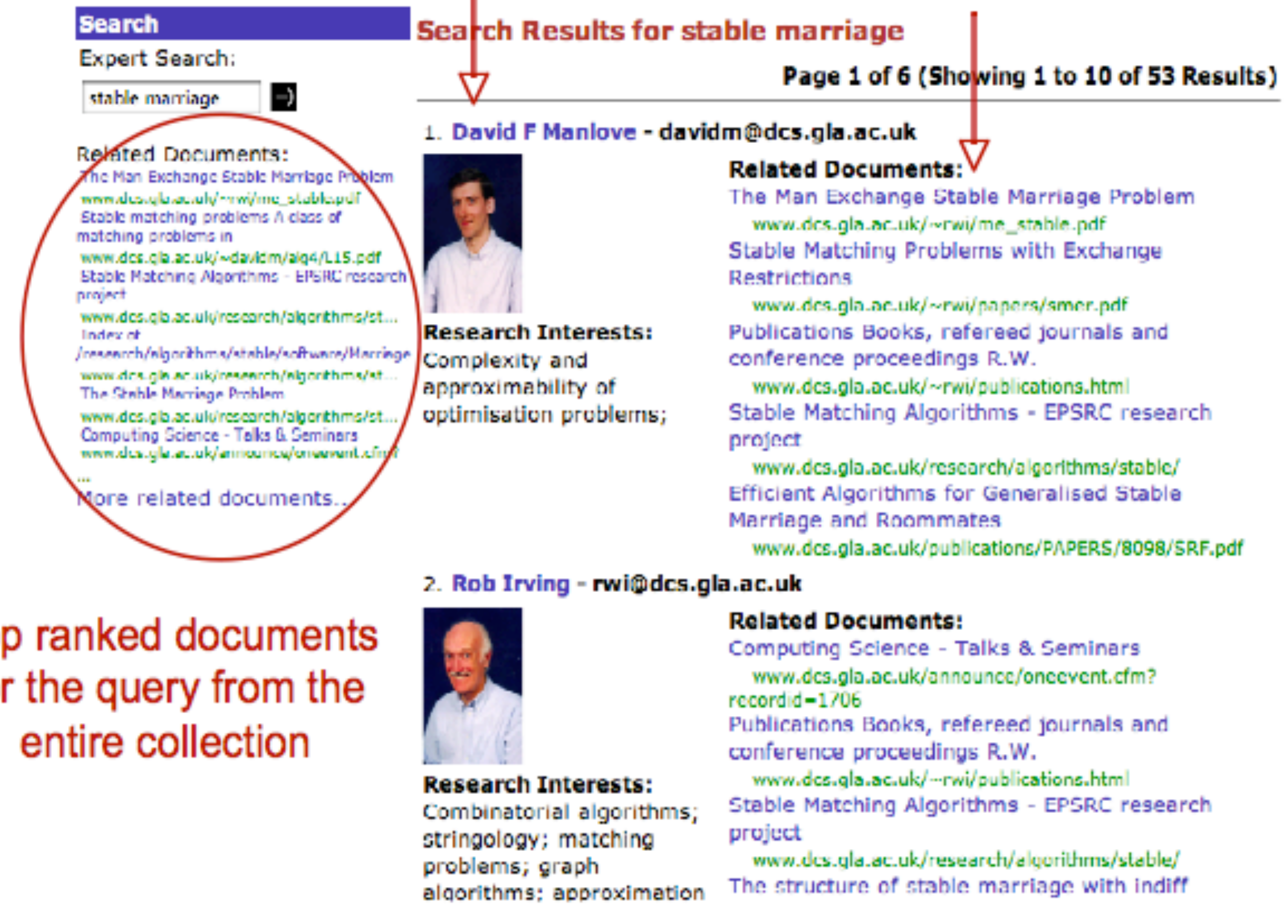
3. About **Terrier**
Text Only **Terrier** TERabyte RetrIEveR Home About **Terrier** Documentation Wiki Download Publications Features Applications People News Mailing Lists Contact us Search our site:
ir.dcs.gla.ac.uk/terrier/about.html

4. **Terrier**/FAQ - Information Retrieval Wiki
Information Retrieval Wiki Search: Login FrontPage RecentChanges FindPage HelpContents **Terrier**/FAQ Show Parent Immutable Page Refresh Show Changes Get Info More Actions: Show Raw Text Show Print View
ir.dcs.gla.ac.uk/wiki/Terrier/FAQ/

Expert Search Engine

Ranking of experts

Supporting evidence for each candidate



Search

Expert Search:

stable marriage

Related Documents:

The Man Exchange Stable Marriage Problem
www.dcs.gla.ac.uk/~rwi/me_stable.pdf
Stable matching problems A class of matching problems in
www.dcs.gla.ac.uk/~davidm/alg4/L15.pdf
Stable Matching Algorithms - EPSRC research project
www.dcs.gla.ac.uk/research/algorithms/st...
Index of /research/algorithms/stable/software/Marriage
www.dcs.gla.ac.uk/research/algorithms/st...
The Stable Marriage Problem
www.dcs.gla.ac.uk/research/algorithms/st...
Computing Science - Talks & Seminars
www.dcs.gla.ac.uk/announce/oneevent.cfm?
...
More related documents...

Search Results for stable marriage

Page 1 of 6 (Showing 1 to 10 of 53 Results)

1. David F Manlove - davidm@dcs.gla.ac.uk

Research Interests:
Complexity and approximability of optimisation problems;

Related Documents:
The Man Exchange Stable Marriage Problem
www.dcs.gla.ac.uk/~rwi/me_stable.pdf
Stable Matching Problems with Exchange Restrictions
www.dcs.gla.ac.uk/~rwi/papers/smer.pdf
Publications Books, refereed journals and conference proceedings R.W.
www.dcs.gla.ac.uk/~rwi/publications.html
Stable Matching Algorithms - EPSRC research project
www.dcs.gla.ac.uk/research/algorithms/stable/
Efficient Algorithms for Generalised Stable Marriage and Roommates
www.dcs.gla.ac.uk/publications/PAPERS/8098/SRF.pdf

2. Rob Irving - rwi@dcs.gla.ac.uk

Research Interests:
Combinatorial algorithms; stringology; matching problems; graph algorithms; approximation

Related Documents:
Computing Science - Talks & Seminars
www.dcs.gla.ac.uk/announce/oneevent.cfm?recordid=1706
Publications Books, refereed journals and conference proceedings R.W.
www.dcs.gla.ac.uk/~rwi/publications.html
Stable Matching Algorithms - EPSRC research project
www.dcs.gla.ac.uk/research/algorithms/stable/
The structure of stable marriage with indiff

Top ranked documents for the query from the entire collection

Figure 2: A ranking of suggested experts for the query stable marriage.

Recent Improvements

- 2.1
  - Various bug fixes
  - FileSystem abstraction layer
- 2.0
  - Single-pass indexing
  - Better non-English support
  - New index format (backward compatabile)
- 1.1.x
  - Non-English index support (UTF)

Useful Links

- Terrier Website
  http://ir.dcs.gla.ac.uk/terrier/

- Terrier Forum - (very active recently)
  http://ir.dcs.gla.ac.uk/terrier/forum/

- Terrier Documentation
  - Contents http://ir.dcs.gla.ac.uk/terrier/doc/
  - TREC Experiment Examples http://ir.dcs.gla.ac.uk/terrier/doc/trec_examples.html
  - All properties http://ir.dcs.gla.ac.uk/terrier/doc/properties.html

- Terrier Publications:
  - http://ir.dcs.gla.ac.uk/terrier/publications.html

Summarising

- Open source IR platform since 2004
- Ideal for
  - Building IR applications
  - Rapid development of new research ideas
  - Large-scale experimentation
- Come participate on the forum!
- Cite us when use Terrier in your papers
  - Ounis et al. Terrier: A High Performance and Scalable Information Retrieval Platform.In *Proceedings of ACM SIGIR'06 Workshop on Open Source Information Retrieval*.

Achieved Outcomes

- Learn more about large-scale IR systems
  - Indexing strategies
  - Index compression
  - Document weighting models + QE
- How to use Terrier
  - Indexing & configuring indexing
  - Retrieval & configuring retrieval
- Design and evaluate an IR experiment
  - Collection, Topics, Qrels, Evaluate
- Extend Terrier to your research ideas
  - Indexing: Collection, Documents, TermPipelines
  - Retrieval: DSMs, TSMs, Post Proceses/Filters

Hopeful future release might have…

- More query language constructs:
  - Polysemy
  - Prior integration
- Collection annotation, e.g. POS
- Query Performance Prediction & DIR Resource Selection
- Web Search UI

Perhaps with your help!