IIIT-H

Web Mining
Lecture 1: Text Indexing and Crawling

Manish Gupta

31st July 2013

Many slides borrowed from
http://www.cse.iitb.ac.in/~soumen/mining-the-web/slides2/02_TextIndexing/TextIndexing.pdf
http://www.stanford.edu/class/cs276/handouts/lecture1-intro.ppt
http://www.cse.iitb.ac.in/soumen/mining-the-web/slides2/WebCrawlingAndSampling/WebCrawlingAndSampling.pdf

Today’s Agenda

- Administrative information
- Course introduction and preview
- Text Indexing
- Crawling
- A 15-minute quiz

Today's Agenda

- **Administrative information**
- Course introduction and preview
- Text Indexing
- Crawling
- A 15-minute quiz

Administrative Information (1)

- Basic Information
  - Instructor: Manish Gupta (manishg.iitb@gmail.com)
    - http://dais.cs.uiuc.edu/manish/
    - Applied Researcher at Bing, Microsoft
    - PhD from Univ of Illinois at Urbana Champaign
    - MTech from IIT Bombay
  - Time: Wed/Sat 8:30am-9:55am
  - Location: 102, Himalaya Building
  - Office Hours
    - On email
    - Meet the instructor after class on Saturdays
  - Time is important both yours and mine. Please be on time.

Administrative Information (2)

- Grading Policy
  - Quizzes (objective) - Surprise short 5-minute in-class quizzes (based on the content taught in the previous class). Six (Best Four. 5% each) - 20%
  - Assignments (Best 4 - 5% each) - 20%
  - Course project- 20%
    - Project idea/spec - 8%
    - Design - 3%
    - Code assessment - 3%
    - Running as per spec - 3%
    - Presentation - 3%
  - Midsem - 10%
  - End exam - 30%
  - 1 point will be awarded for each student who points out a mistake (first student only) or a very innovative idea related to any of the papers we study. This will be completely at the discretion of the instructor.

Administrative Information (3)

- Tentative Assignments Schedule

Assignment Title	Date Posted	Submission Date
Understanding Hadoop and a very brief introduction to Pig and Hive	14-Aug	21-Aug
Assignment on content covered until Aug 31	28-Aug	4-Sep
Studying various features of Lingpipe	18-Sep	25-Sep
Studying various ways of using Lemur	9-Oct	23-Oct
Assignment on content covered until Nov 6	14-Aug	21-Aug

- All assignments are due at 11:59pm on dates mentioned below.
- Assignments must be submitted by email to the instructor and cc-ed to the TAs.
- Honesty is the best policy. Refrain from copying/cheating etc. Severe penalty if caught.
- **Late Assignment Submission Policy**
  - 0-1 day late: Student can get a maximum of 50% marks.
  - 1-2 day late: Student can get a maximum of 25% marks.
  - >2 days late: Student will not get any marks.

Administrative Information (4)

- Course Project
  - The project can be done in groups of 2 or 3.
  - The instructor will post some sample project topics by week 5.
  - By week 7, the teams will have to either choose a topic from the sample set or propose their own project topics related to any area within web mining.
  - Students will have to start the project topic from week 10.
  - In the last week, the teams will have to send the poster+ project code + documentation + instructions to run the code to the instructor via email. Finally, in the last week, we will also have poster + project presentations (5 min per team).

Today’s Agenda

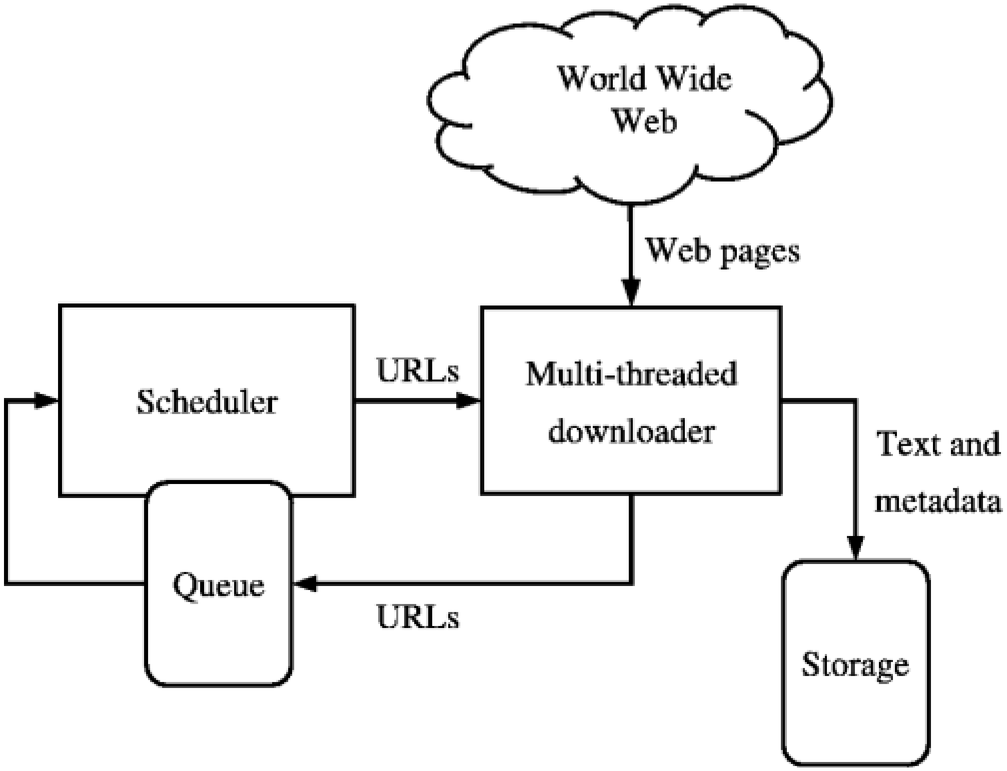
- Administrative information
- **Course introduction and preview**
- Text Indexing
- Crawling
- A 15-minute quiz

Course Outline

- Text indexing, Crawling
- Ranking, Scoring Techniques
- Similarity Search
- Ranking and link analysis
- Topic Models
- Recommender Systems
- Social Networks
- Social Influence Analysis
- Micro-blogging
- Computational Advertising
- Mining Structured Information from the Web
- Mining Structured Information from the Web for Entities (Entity Mining)
- Web Search Query Log Mining
- Crowdsourcing

Indexing, Crawling

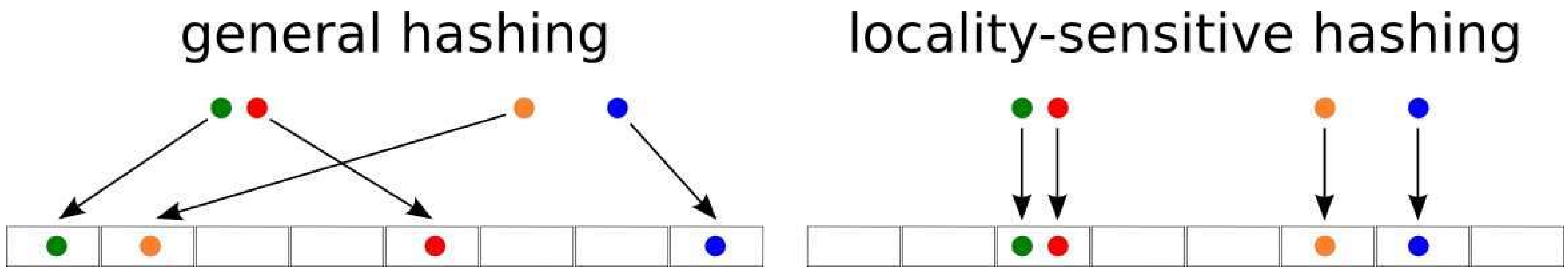
term	doc. freq.	→	postings lists
ambitious	1	→	2
be	1	→	2
brutus	2	→	1 → 2
capitol	1	→	1
caesar	2	→	1 → 2
did	1	→	1
enact	1	→	1
hath	1	→	2
i	1	→	1
i’	1	→	1
it	1	→	2
julius	1	→	1
killed	1	→	1
let	1	→	2
me	1	→	1
noble	1	→	2
so	1	→	2
the	2	→	1 → 2
told	1	→	2
you	1	→	2
was	2	→	1 → 2
with	1	→	2



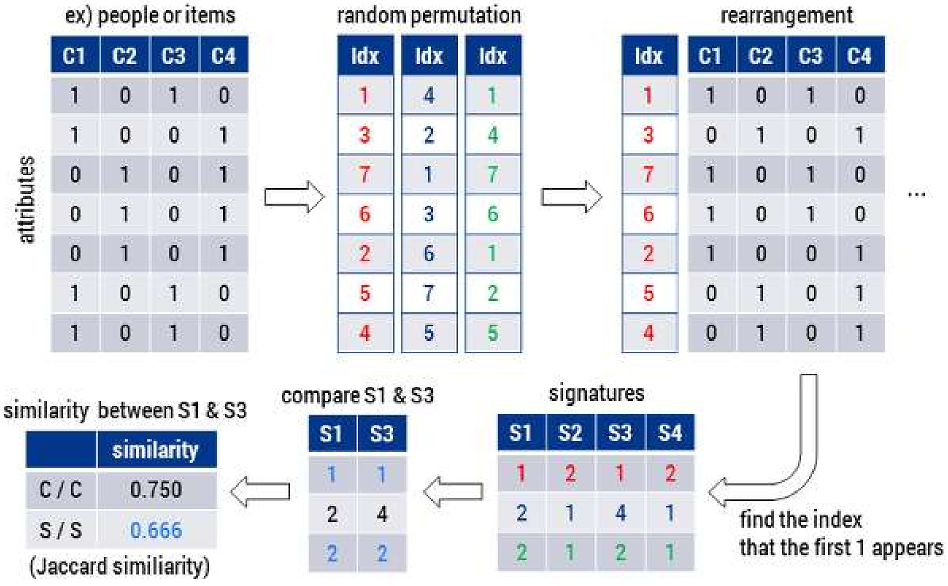
Ranking, Scoring Techniques

- Recall, precision, interpolated precision, F
- MAP, MRR, ROC, AUC
- NDCG
- Kappa measure
- A/B testing
- Term frequency, collection/corpus frequency, document frequency, TFIDF
- BM25
- Fagin's threshold algorithm

Similarity Search



Min-hashing



ex) people or items

C1	C2	C3	C4
1	0	1	0
1	0	0	1
0	1	0	1
0	1	0	1
0	1	0	1
1	0	1	0
1	0	1	0

random permutation

Idx	Idx	Idx
1	4	1
3	2	4
7	1	7
6	3	6
2	6	1
5	7	2
4	5	5

rearrangement

Idx	C1	C2	C3	C4
1	1	0	1	0
3	0	1	0	1
7	1	0	1	0
6	1	0	1	0
2	1	0	0	1
5	0	1	0	1
4	0	1	0	1

find the index
that the first 1 appears

signatures

S1	S2	S3	S4
1	2	1	2
2	1	4	1
2	1	2	1

compare S1 & S3

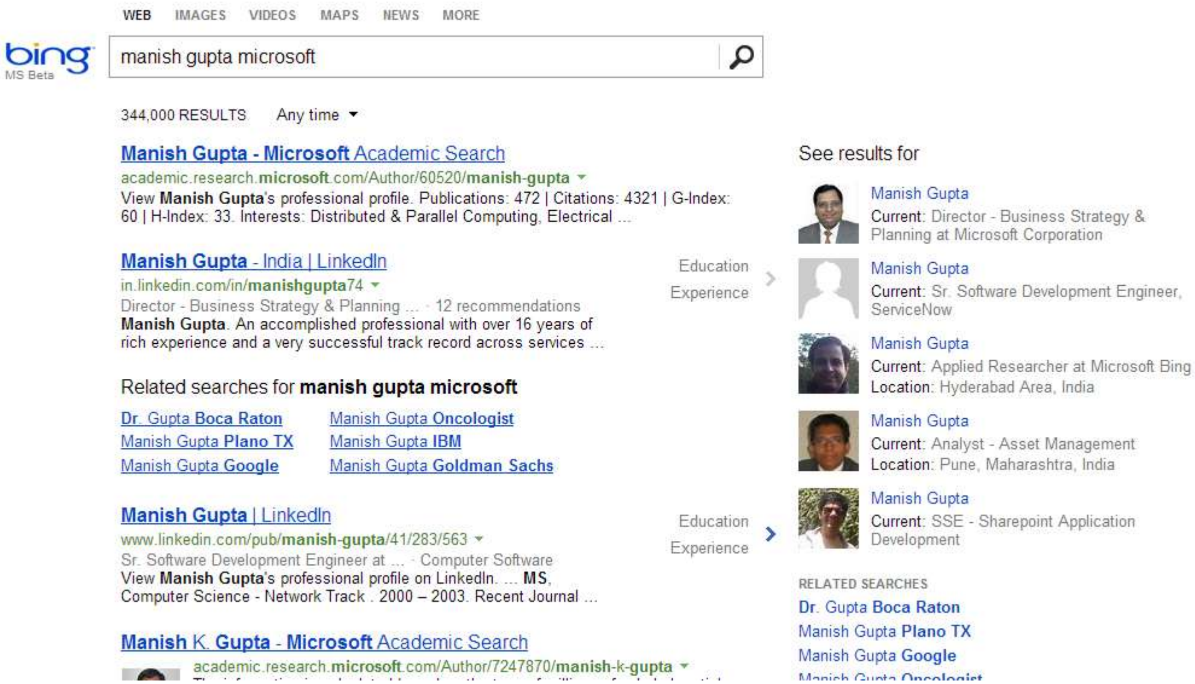
S1	S3
1	1
2	4
2	2

similarity between S1 & S3

	similarity
C / C	0.750
S / S	0.666

(Jaccard similiarity)

Ranking and Link Analysis



manish gupta microsoft

344,000 RESULTS Any time

Manish Gupta - Microsoft Academic Search
academic.research.**microsoft**.com/Author/60520/**manish-gupta**
View **Manish Gupta**'s professional profile. Publications: 472 | Citations: 4321 | G-Index: 60 | H-Index: 33. Interests: Distributed & Parallel Computing, Electrical ...

Manish Gupta - India | LinkedIn
in.linkedin.com/in/**manishgupta**74
Director - Business Strategy & Planning ... · 12 recommendations
Manish Gupta. An accomplished professional with over 16 years of rich experience and a very successful track record across services ...

Education
Experience

Related searches for **manish gupta microsoft**

Dr. Gupta **Boca Raton**
Manish Gupta **Plano TX**
Manish Gupta **Google**
Manish Gupta **Oncologist**
Manish Gupta **IBM**
Manish Gupta **Goldman Sachs**

Manish Gupta | LinkedIn
www.linkedin.com/pub/**manish-gupta**/41/283/563
Sr. Software Development Engineer at ... · Computer Software
View **Manish Gupta**'s professional profile on LinkedIn. ... **MS**, Computer Science - Network Track . 2000 – 2003. Recent Journal ...

Education
Experience

Manish K. **Gupta** - **Microsoft** Academic Search
academic.research.**microsoft**.com/Author/7247870/**manish-k-gupta**

See results for

Manish Gupta
Current: Director - Business Strategy & Planning at Microsoft Corporation

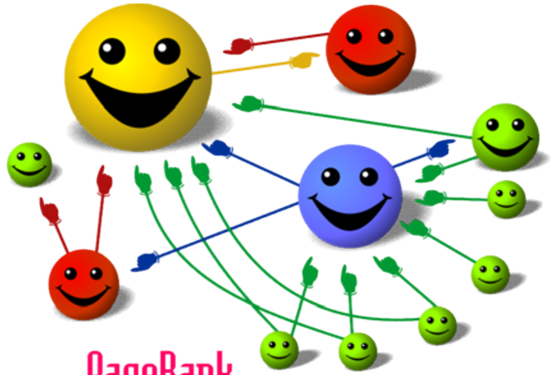
Manish Gupta
Current: Sr. Software Development Engineer, ServiceNow

Manish Gupta
Current: Applied Researcher at Microsoft Bing
Location: Hyderabad Area, India

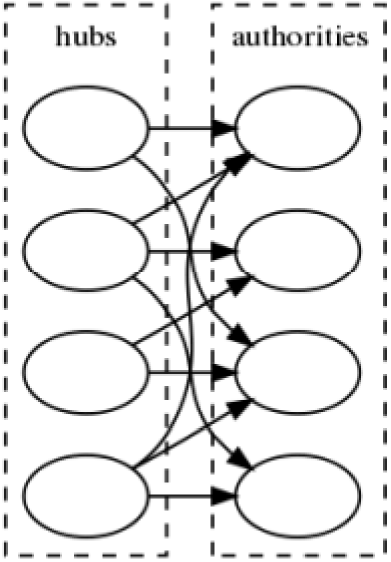
Manish Gupta
Current: Analyst - Asset Management
Location: Pune, Maharashtra, India

Manish Gupta
Current: SSE - Sharepoint Application Development

RELATED SEARCHES
Dr. Gupta **Boca Raton**
Manish Gupta **Plano TX**
Manish Gupta **Google**
Manish Gupta **Oncologist**

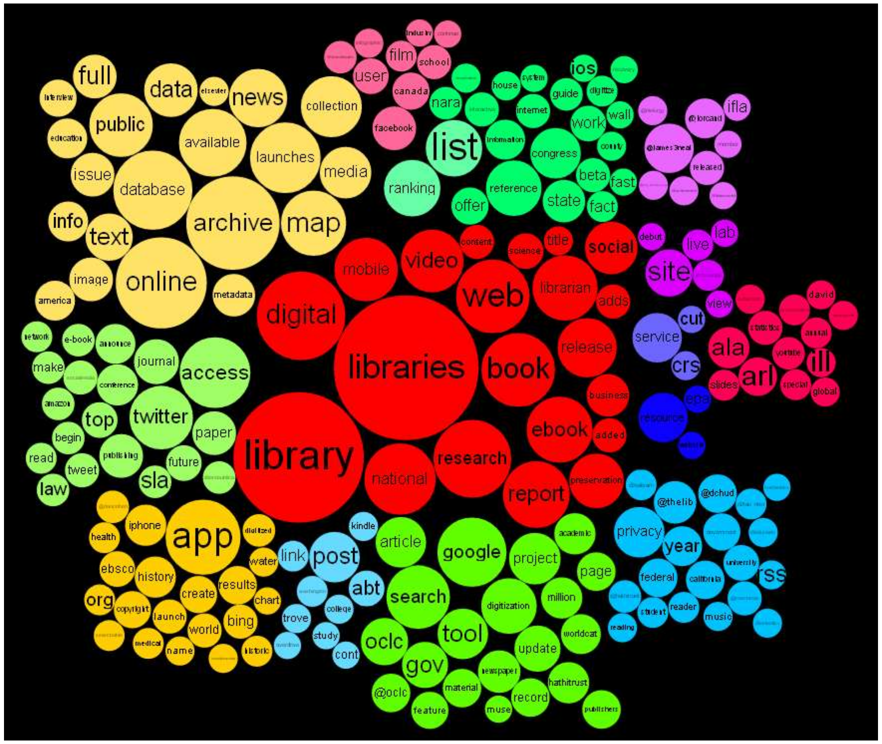


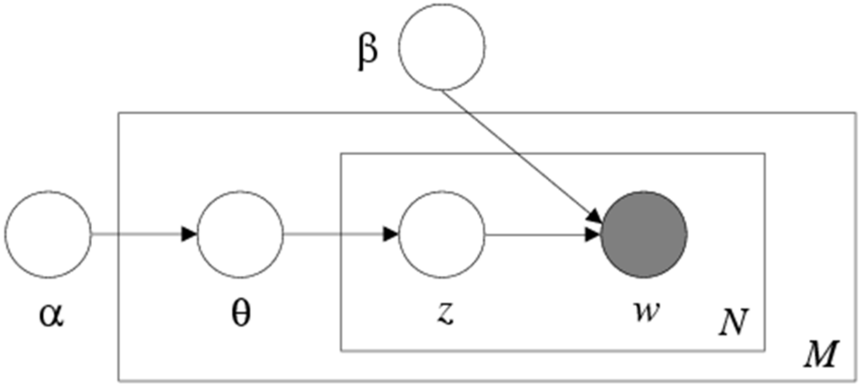
PageRank



HITS

Topic Models

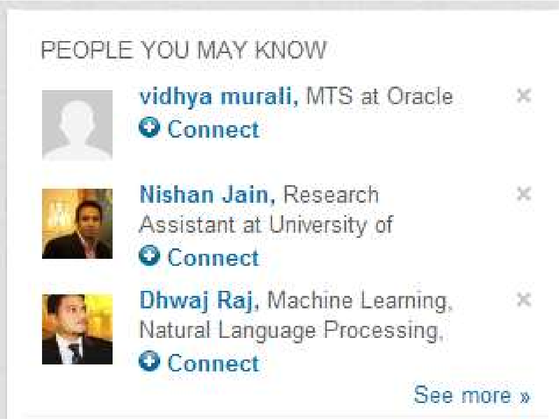




Latent Dirichlet Allocation (LDA)

Recommender Systems

LinkedIn Recommendations



PEOPLE YOU MAY KNOW

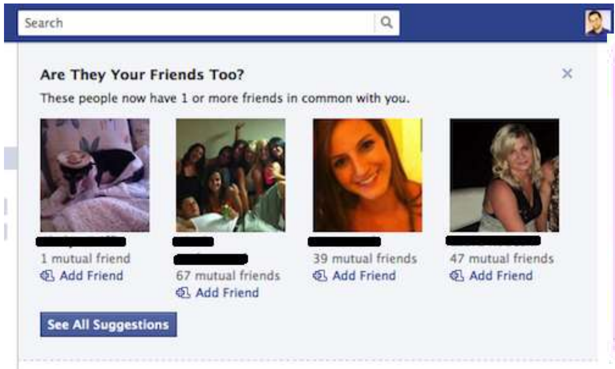
vidhya murali, MTS at Oracle
Connect

Nishan Jain, Research Assistant at University of
Connect

Dhwaj Raj, Machine Learning, Natural Language Processing,
Connect

See more »

Facebook Recommendations



Search

Are They Your Friends Too?

These people now have 1 or more friends in common with you.

[redacted]
1 mutual friend
Add Friend

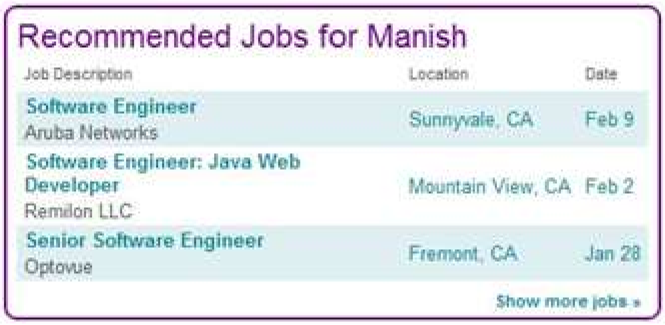
[redacted]
67 mutual friends
Add Friend

[redacted]
39 mutual friends
Add Friend

[redacted]
47 mutual friends
Add Friend

See All Suggestions

Job Recommendations

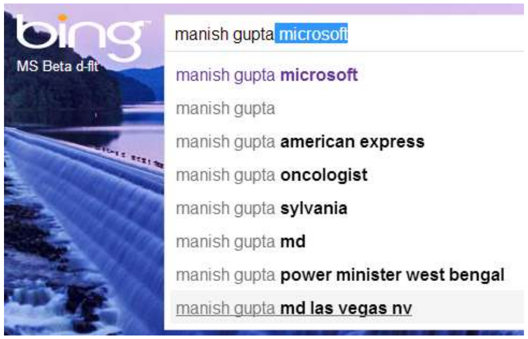


Recommended Jobs for Manish

Job Description	Location	Date
Software Engineer Aruba Networks	Sunnyvale, CA	Feb 9
Software Engineer: Java Web Developer Remilon LLC	Mountain View, CA	Feb 2
Senior Software Engineer Optovue	Fremont, CA	Jan 28

Show more jobs »

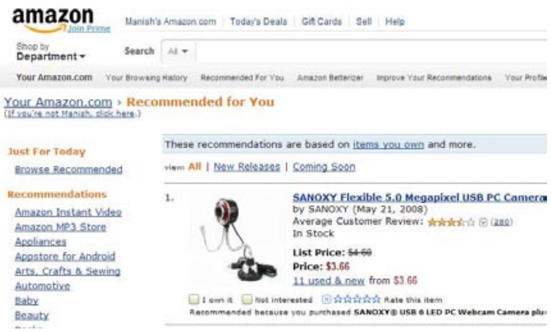
Query Recommendations



manish gupta microsoft

manish gupta microsoft
manish gupta
manish gupta american express
manish gupta oncologist
manish gupta sylvania
manish gupta md
manish gupta power minister west bengal
manish gupta md las vegas nv

Product Recommendations



Your Amazon.com › Recommended for You
(If you're not Manish, click here.)

Just For Today
Browse Recommended

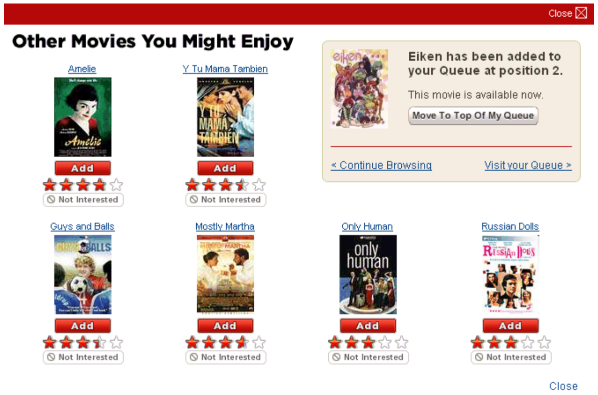
Recommendations
Amazon Instant Video
Amazon MP3 Store
Appliances
Appstore for Android
Arts, Crafts & Sewing
Automotive
Baby
Beauty

These recommendations are based on items you own and more.

view: All | New Releases | Coming Soon

1. SANOXY Flexible 5.0 Megapixel USB PC Camera
by SANOXY (May 21, 2008)
Average Customer Review: ★★★½☆ (280)
In Stock
List Price: ~~$4.60~~
Price: $3.66
11 used & new from $3.66
□ I own it □ Not interested ☆☆☆☆☆ Rate this item
Recommended because you purchased SANOXY® USB 6 LED PC Webcam Camera plu

Netflix Movie Recommendations



Close ☒

Other Movies You Might Enjoy

Amelie
Add
★★★★☆
Not Interested

Y Tu Mama Tambien
Add
★★★½☆
Not Interested

Eiken has been added to your Queue at position 2.
This movie is available now.
Move To Top Of My Queue
< Continue Browsing
Visit your Queue >

Guys and Balls
Add
★★★½☆
Not Interested

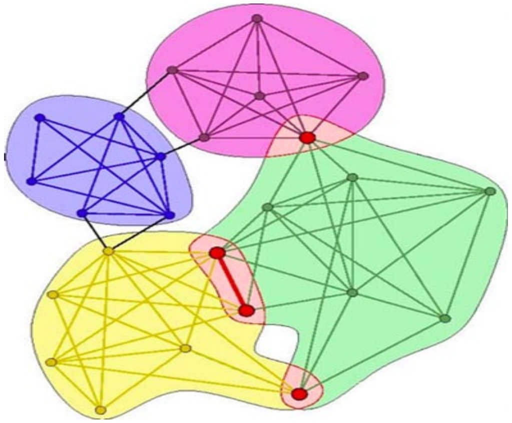
Mostly Martha
Add
★★★½☆
Not Interested

Only Human
Add
★★★☆☆
Not Interested

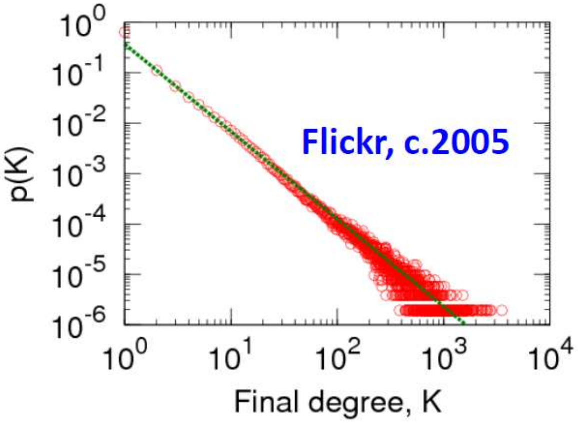
Russian Dolls
Add
★★★☆☆
Not Interested

Close

Social Networks



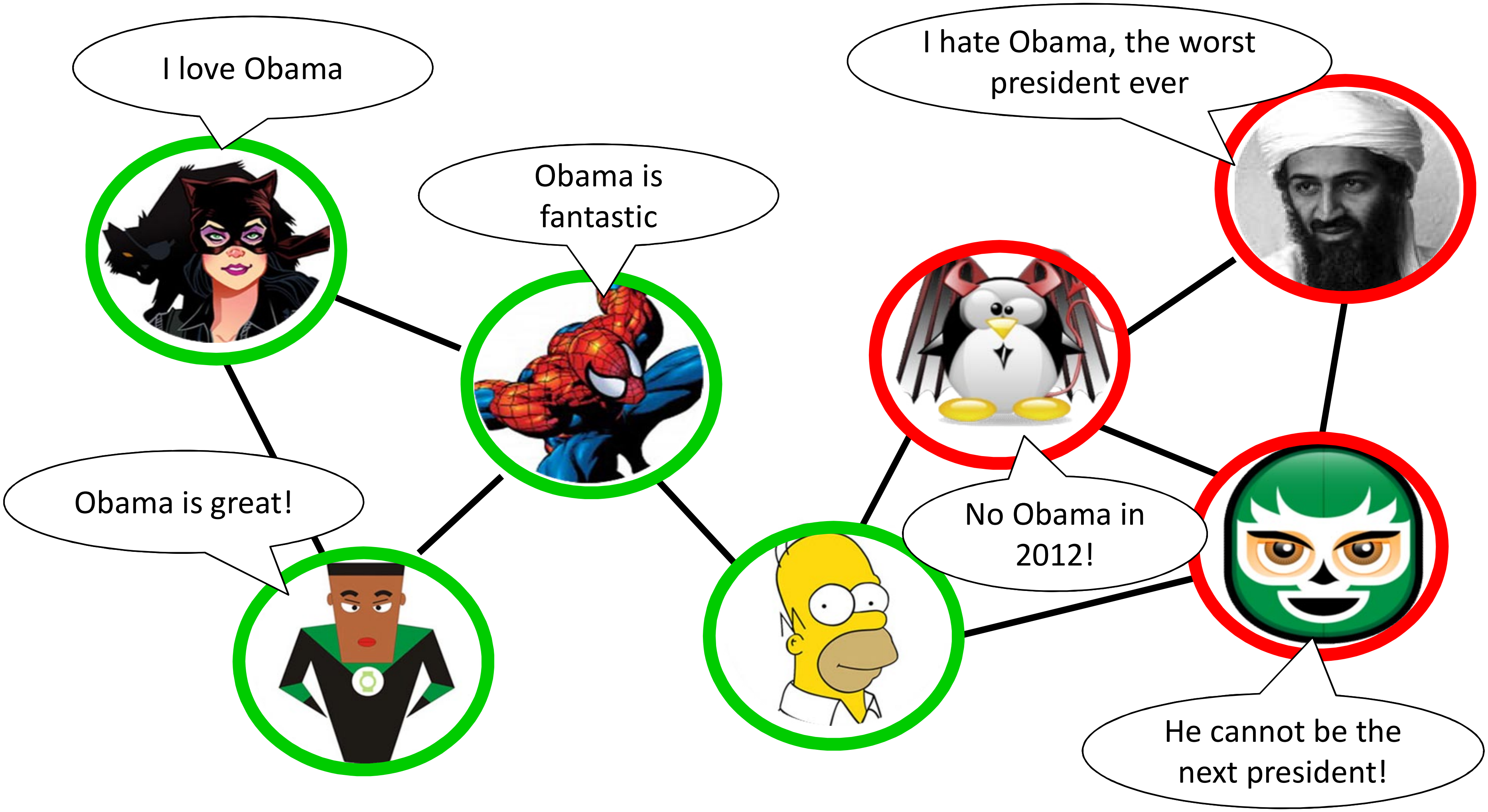
Community Detection



Link Prediction

- Network generation models
  - Random graph model
  - Preferential attachment
  - Copying model
  - Forest fire model …

Social Influence Analysis



I love Obama

Obama is fantastic

Obama is great!

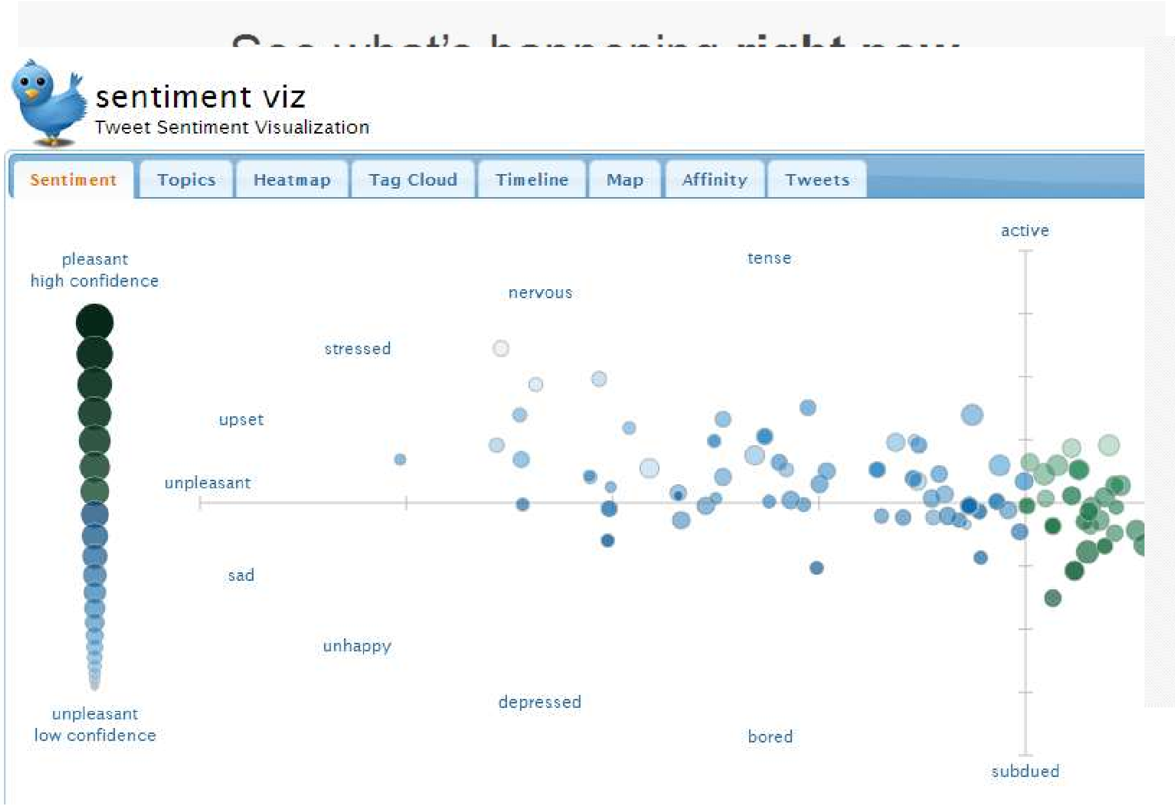
I hate Obama, the worst president ever

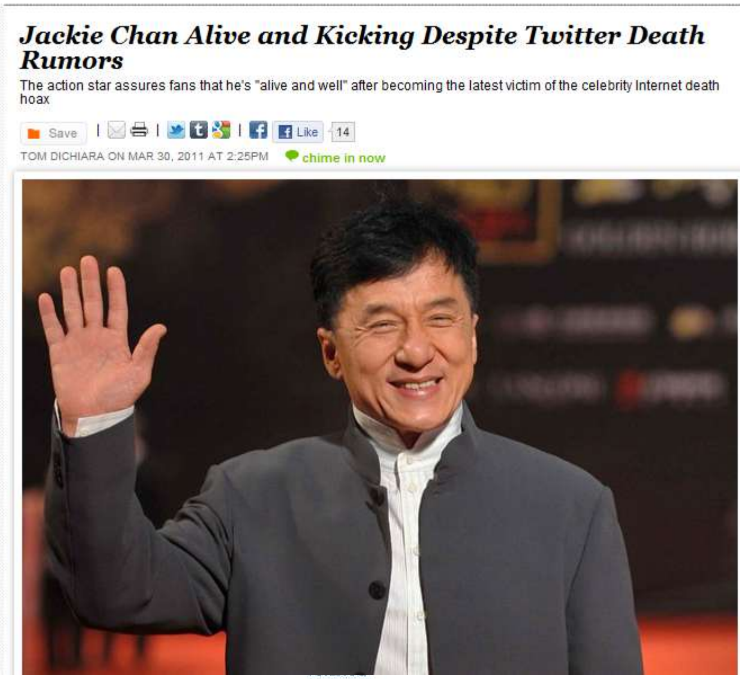
No Obama in 2012!

He cannot be the next president!

🟢 Positive 🔴 Negative

Micro-blogging





Jackie Chan Alive and Kicking Despite Twitter Death Rumors

The action star assures fans that he's "alive and well" after becoming the latest victim of the celebrity Internet death hoax

TOM DICHIARA ON MAR 30, 2011 AT 2:25PM

Computational Advertising

shaadi.com The World's Largest Matrimonial Service
Bride
18-24
Search Now
Best Matches | Most Responses | Fully Secure

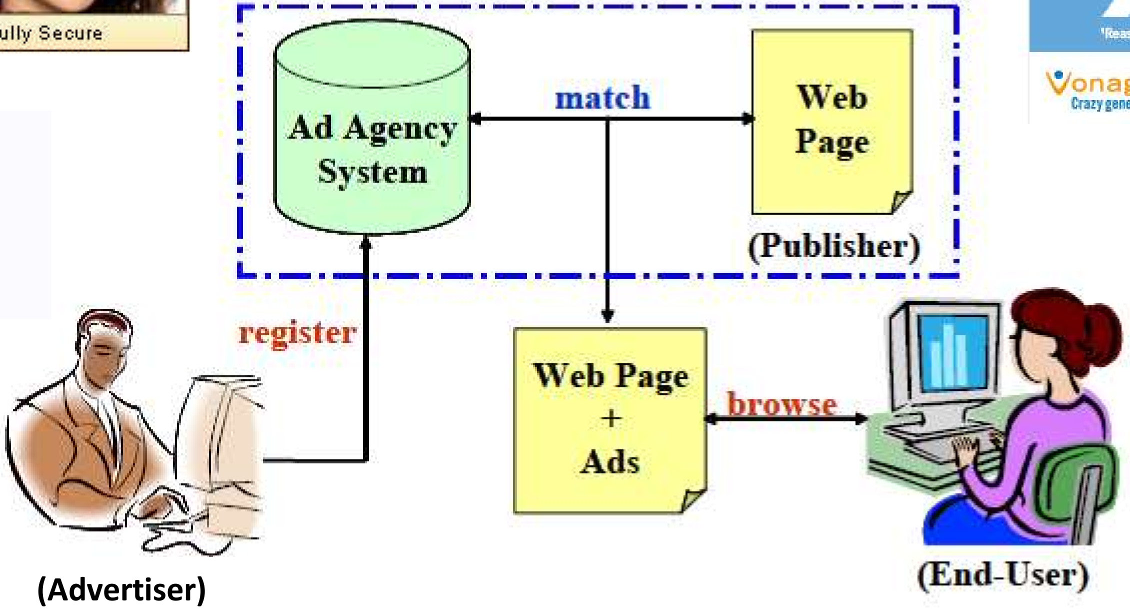
Get unlimited¹ calling to landlines & mobiles in India.
VONAGE WORLD®
$9.99 /MO.
for the first 3 months then $25.99/mo. Plus taxes & fees with 1-year agreement
¹Reasonable Use Policy and Important Details Apply.
Vonage Crazy generous.
START SAVING NOW

Ads

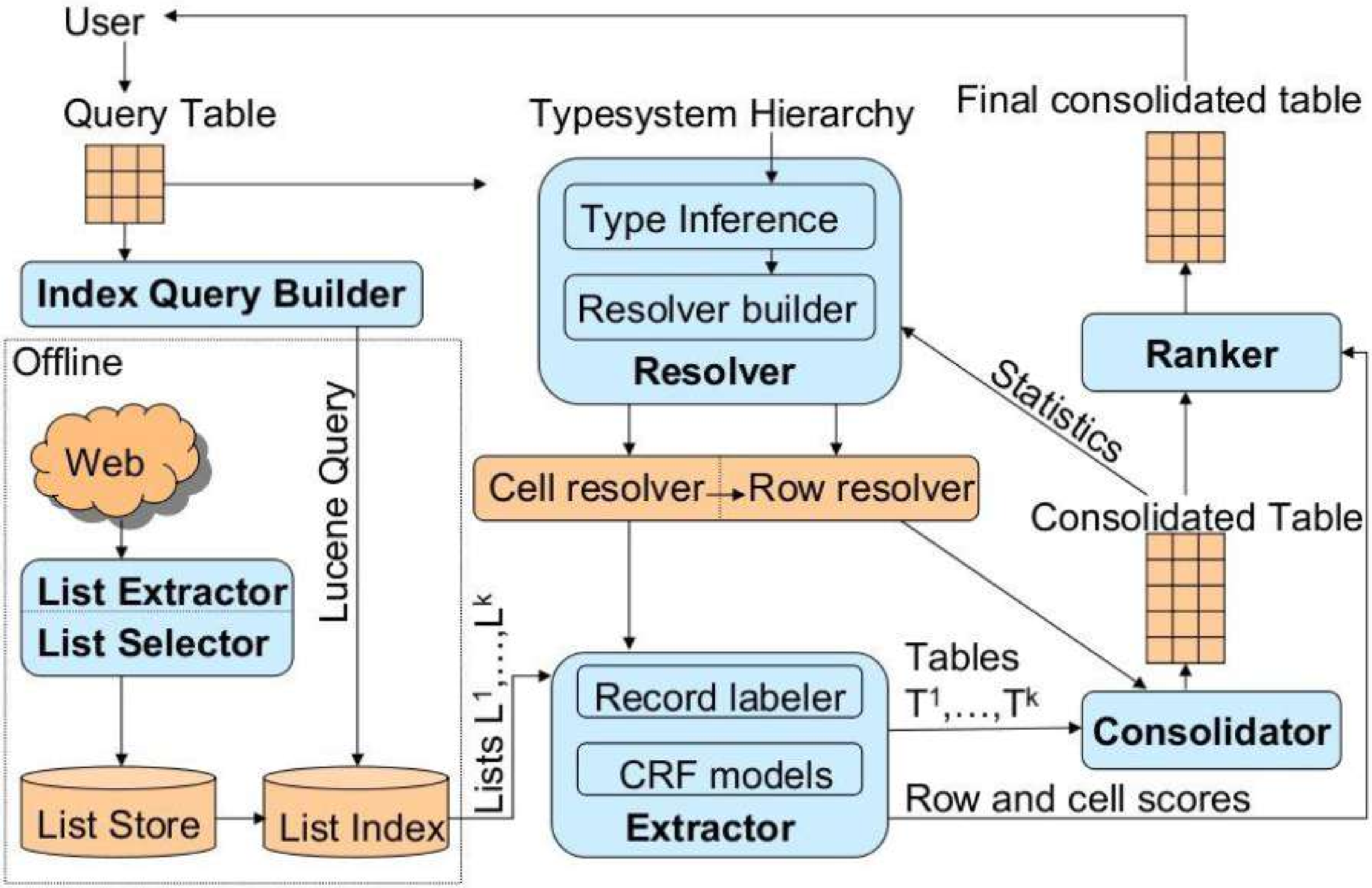
How To Fix **Java** Errors

FixError.com/Microsoft

If Your PC Has **Java** Issues Then Download This Fix.



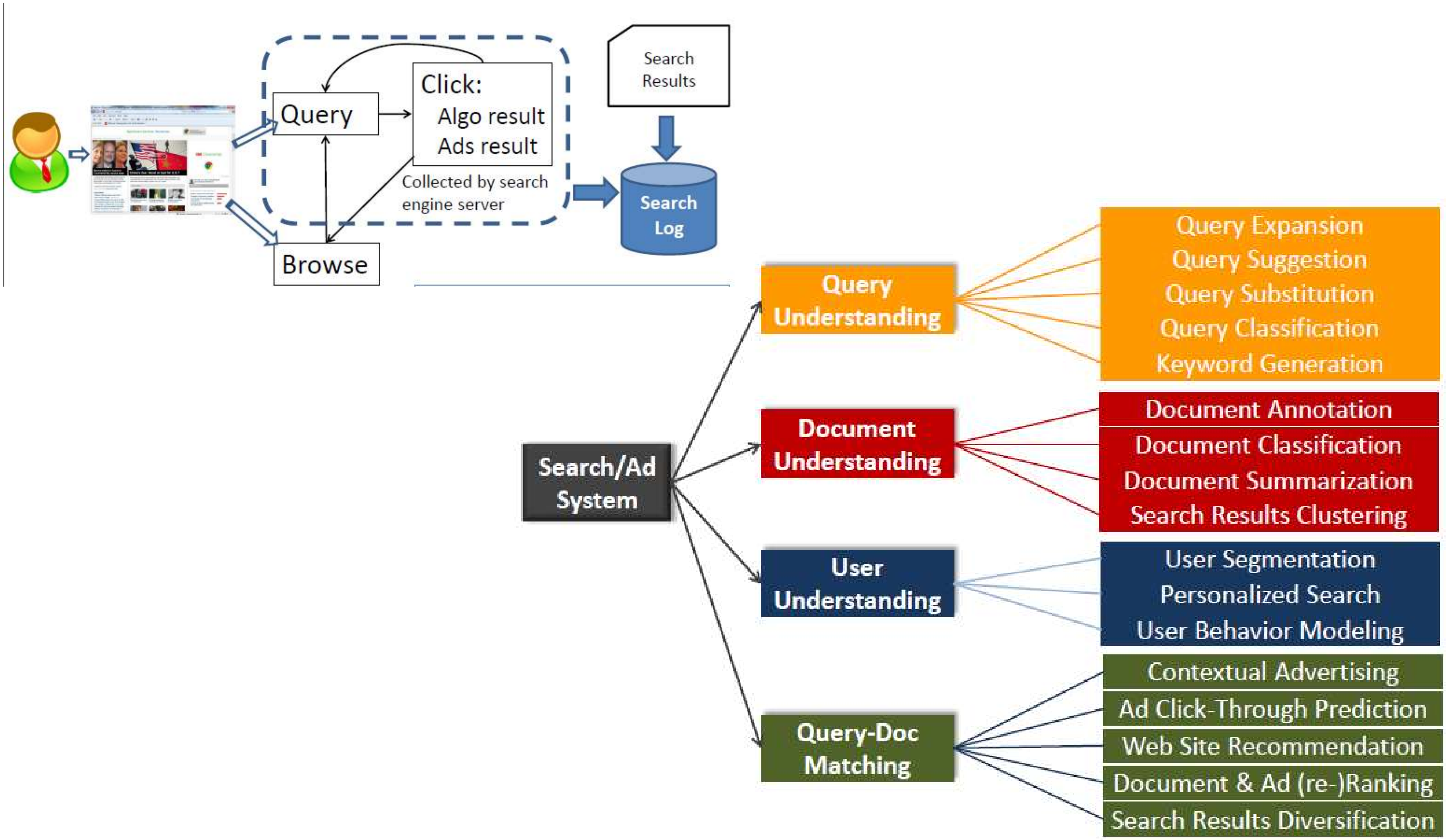
Mining Structured Information from the Web



Mining Structured Information from the Web for Entities (Entity Mining)

- Given an entity (or set of entities), what are other ways it is referred to?
  - Entity Synonyms
- Given a set of entities, what are the interesting attributes of these entities?
  - Entity Attribute Discovery
- Given a set of entities and an attribute, what are values of the entities on that attribute?
  - Entity Augmentation
- Given a set of entities and a text corpus, what are the semantic mentions of the entity in the corpus?
  - Entity Linking
- Given a set of entities, find phrases (“tags”) describing those entities?
  - Entity Tagging

Web Search Query Log Mining

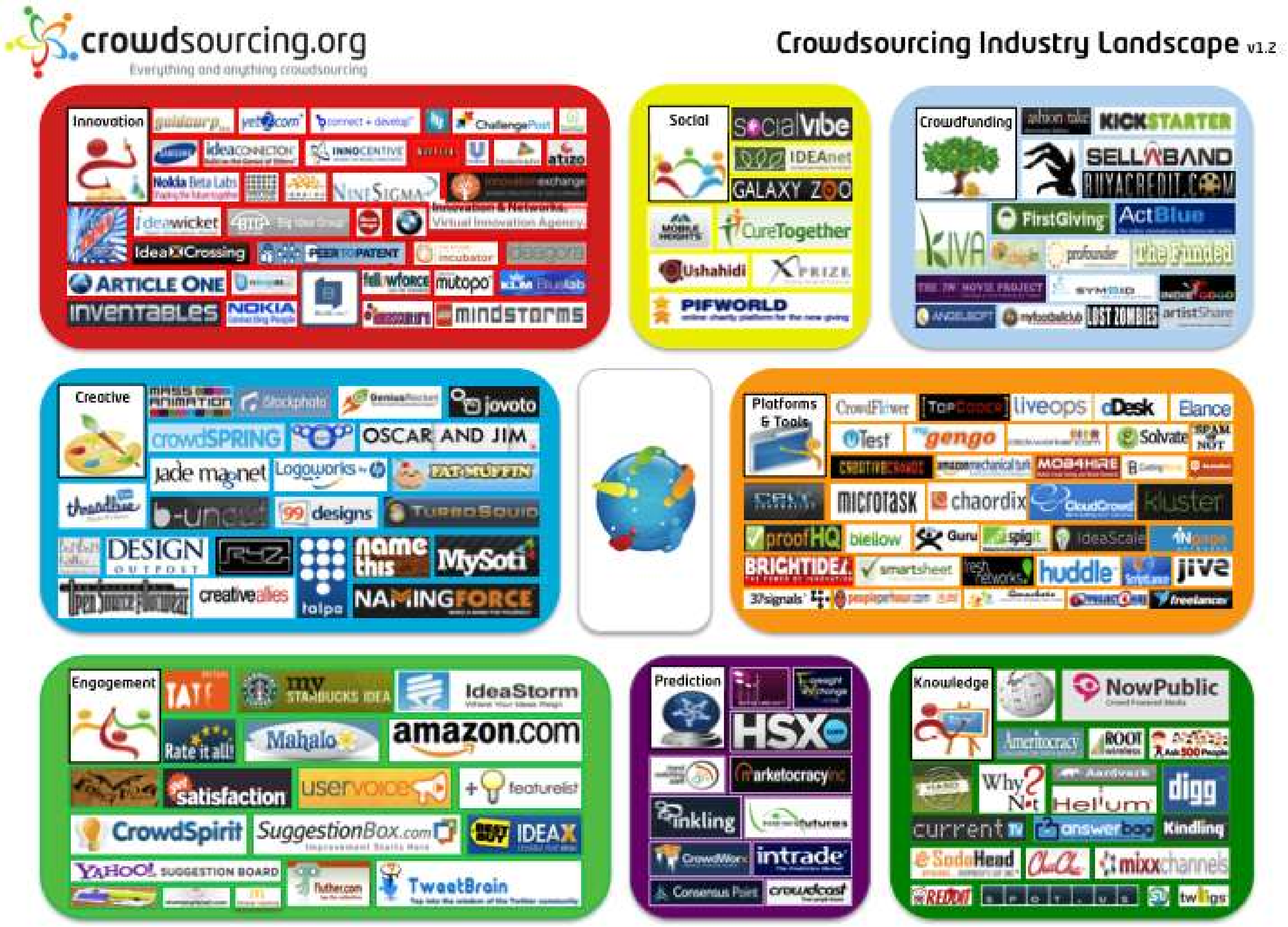


Search/Ad System

- Query Understanding
  - Query Expansion
  - Query Suggestion
  - Query Substitution
  - Query Classification
  - Keyword Generation
- Document Understanding
  - Document Annotation
  - Document Classification
  - Document Summarization
  - Search Results Clustering
- User Understanding
  - User Segmentation
  - Personalized Search
  - User Behavior Modeling
- Query-Doc Matching
  - Contextual Advertising
  - Ad Click-Through Prediction
  - Web Site Recommendation
  - Document & Ad (re-)Ranking
  - Search Results Diversification

Crowdsourcing

amazonmechanical turk
Artificial Artificial Intelligence



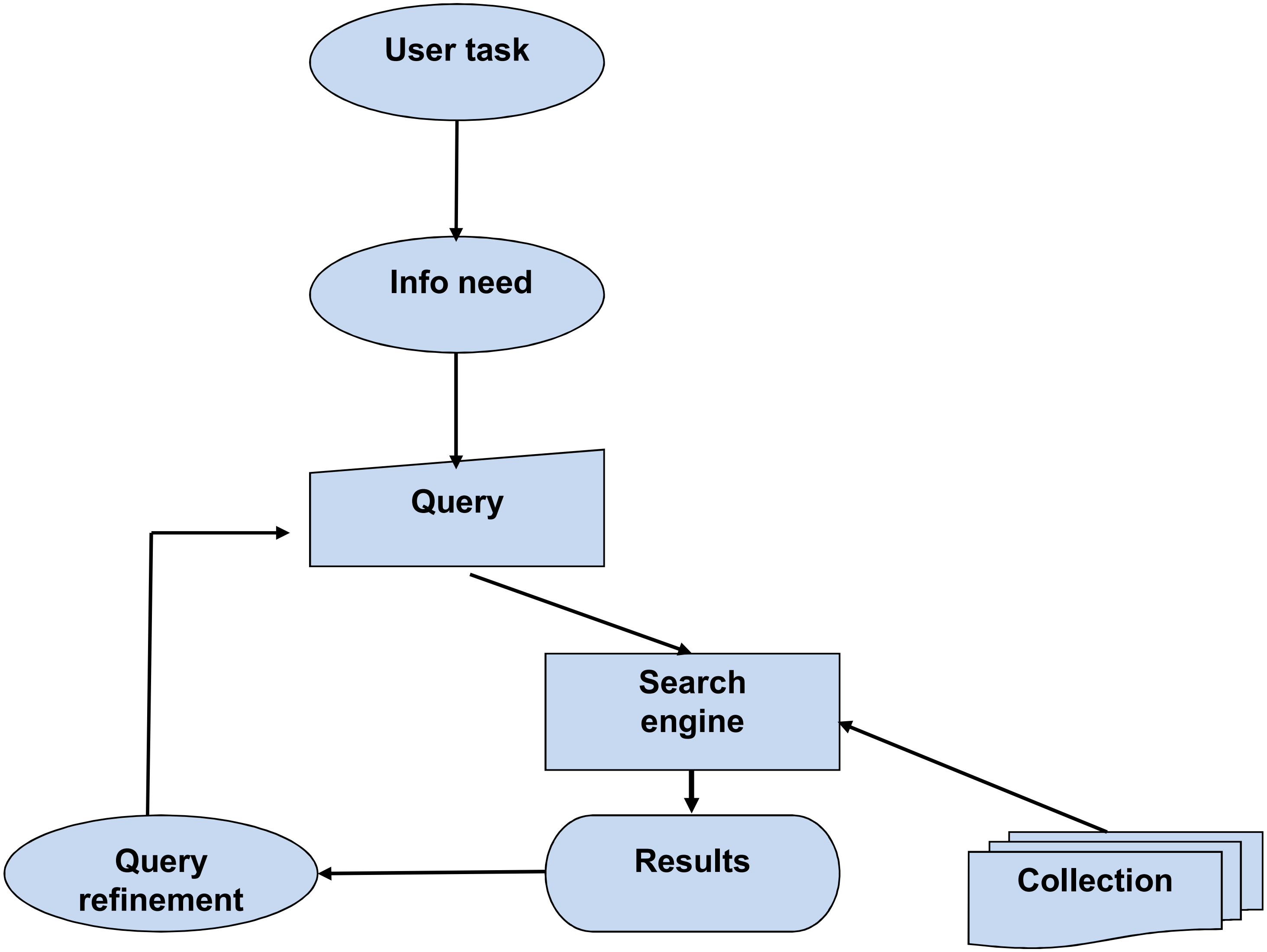
© Crowdsourcing LLC 2010 – Published under a Creative Commons Attribution Share-A-Like 2.5 License

September 2010 · Created by Carl Esposti / +1 (310) 948-1258 / carl@massolution.com

Today’s Agenda

- Administrative information
- Course introduction and preview
- **Text Indexing**
- Crawling
- A 15-minute quiz

The Classic Search Model



Query on Unstructured Data

- Which plays of Shakespeare contain the words ***Brutus*** *AND* ***Caesar*** but *NOT* ***Calpurnia***?
- One could `grep` all of Shakespeare's plays for ***Brutus*** and ***Caesar,*** then strip out lines containing ***Calpurnia***?
- Why is that not the answer?
  - Slow (for large corpora)
  - *NOT* ***Calpurnia*** is non-trivial
  - Other operations (e.g., find the word ***Romans*** near ***countrymen***) not feasible
  - Ranked retrieval (best documents to return)

Term-Document Incidence Matrices

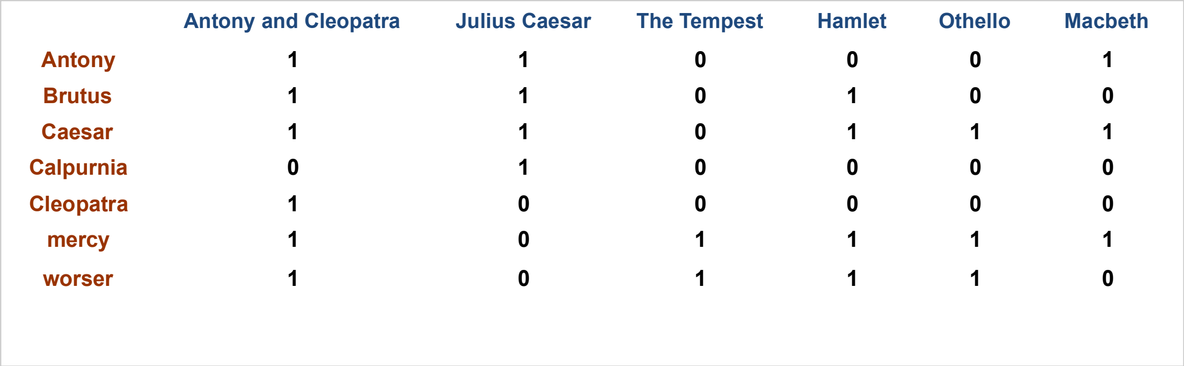
	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

Brutus** AND **Caesar** BUT NOT **Calpurnia

1 if play contains word, 0 otherwise

Incidence Vectors

- So we have a 0/1 vector for each term.
- To answer query: take the vectors for ***Brutus, Caesar*** and ***Calpurnia*** (complemented) ➔ bitwise *AND*.
  - 110100 *AND* 110111 *AND* 101111 = **100100**



	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

Bigger Collections

- Consider *N* = 1 million documents, each with about 1000 words.
- Avg 6 bytes/word including spaces/punctuation
  - 6GB of data in the documents.
- Say there are *M* = 500K *distinct* terms among these.

Can’t Build the Matrix

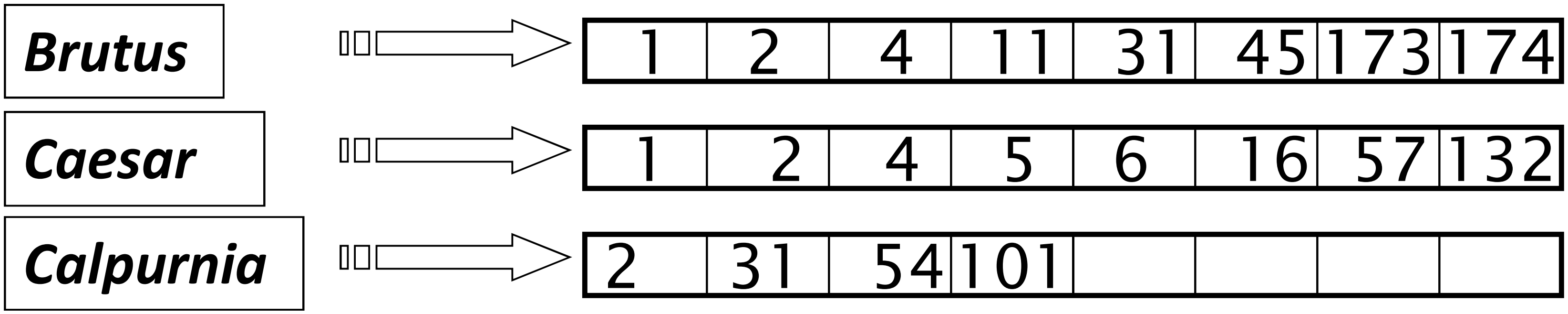
- 500K x 1M matrix has half-a-trillion 0’s and 1’s.
- But it has no more than one billion 1’s.
  - matrix is extremely sparse.

Why?

- What’s a better representation?
  - We only record the 1 positions.

Inverted Index

- For each term *t*, we must store a list of all documents that contain *t*.
  - Identify each doc by a **docID**, a document serial number
- Can we used fixed-size arrays for this?

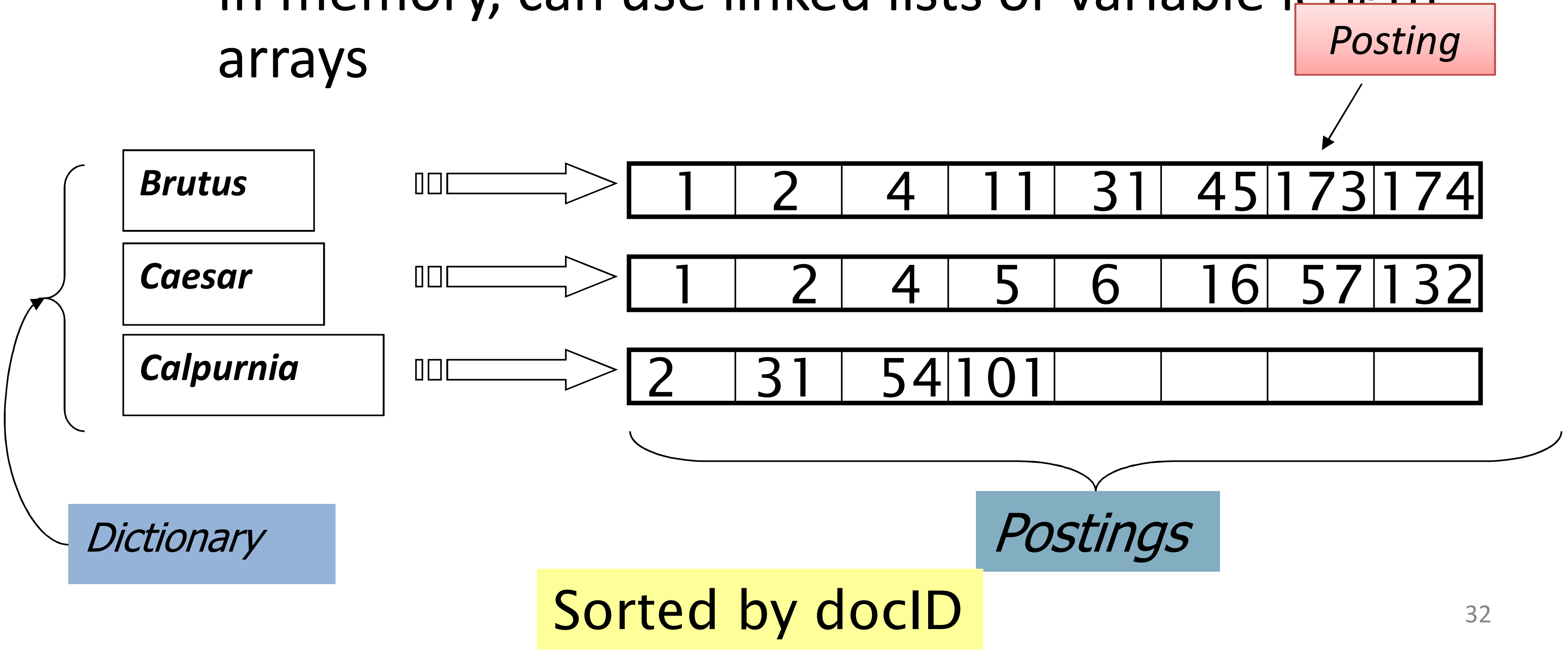


Brutus	1	2	4	11	31	45	173	174
Caesar	1	2	4	5	6	16	57	132
Calpurnia	2	31	54	101				

What happens if the word ***Caesar*** is added to document 14?

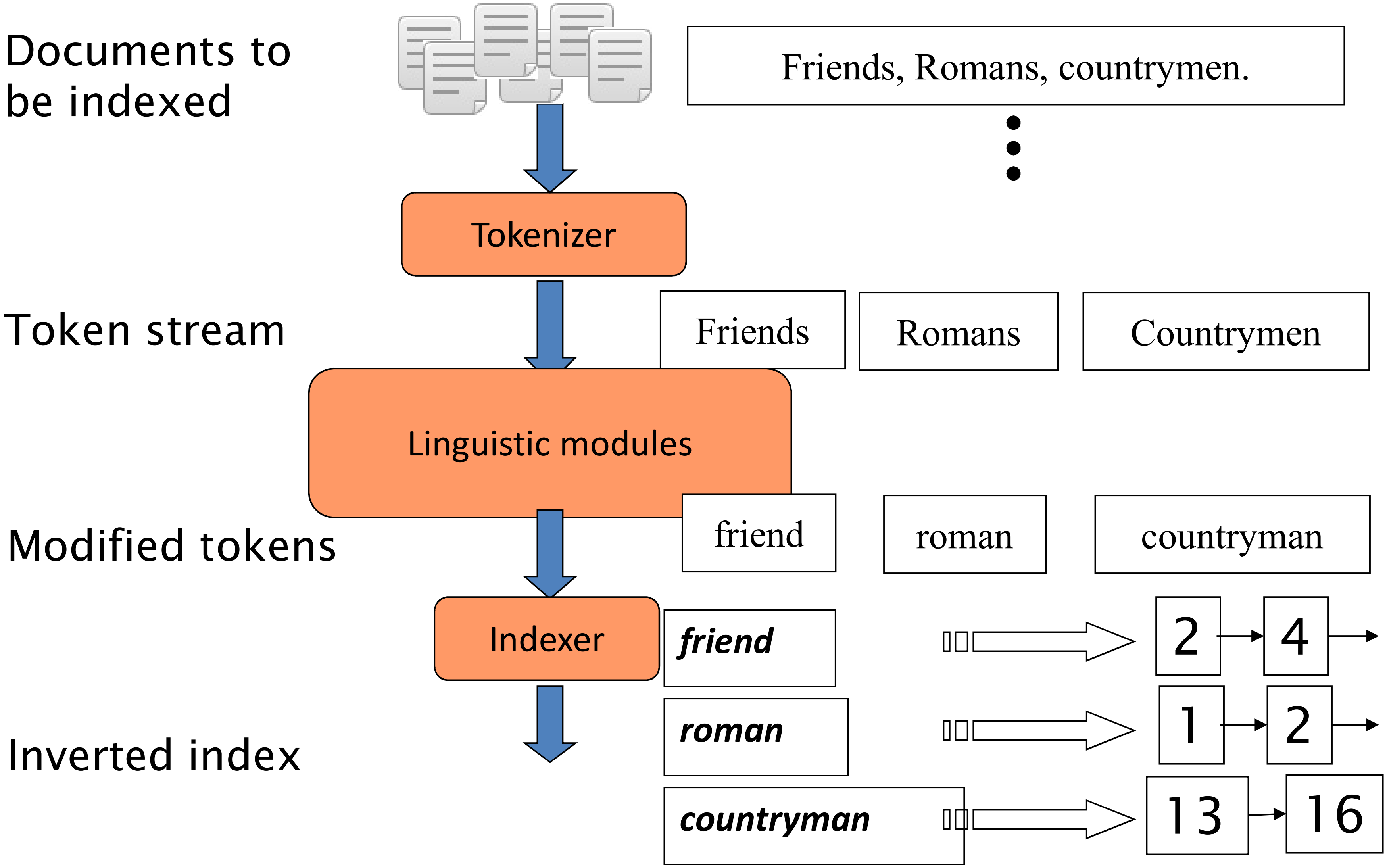
Inverted Index

- We need variable-size postings lists
  - On disk, a continuous run of postings is normal and best
  - In memory, can use linked lists or variable length arrays



Brutus	1	2	4	11	31	45	173	174
Caesar	1	2	4	5	6	16	57	132
Calpurnia	2	31	54	101				

Inverted Index Construction



Documents to be indexed

Friends, Romans, countrymen.

Tokenizer

Token stream

Friends | Romans | Countrymen

Linguistic modules

Modified tokens

friend | roman | countryman

Indexer

Inverted index

Initial Stages of Text Processing

- Tokenization
  - Cut character sequence into word tokens
    - Deal with ***“John’s”***, ***a state-of-the-art solution***
- Normalization
  - Map text and query term to same form
    - You want ***U.S.A.*** and ***USA*** to match
- Stemming
  - We may wish different forms of a root to match
    - ***authorize***, ***authorization***
- Stop words
  - We may omit very common words (or not)
    - ***the, a, to, of***

Indexer Steps: Token Sequence

- Sequence of (Modified token, Document ID) pairs.

Doc 1

I did enact Julius
Caesar I was killed
i’ the Capitol;
Brutus killed me.

Doc 2

So let it be with
Caesar. The noble
Brutus hath told you
Caesar was ambitious

Term	docID
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Indexer Steps: Sort

- Sort by terms
  - And then docID

Core indexing step

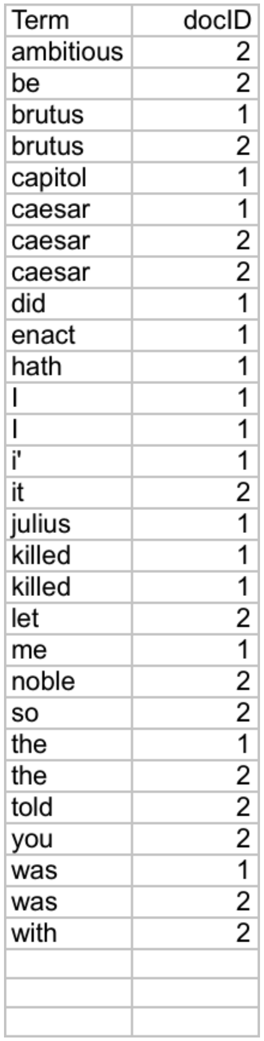
Term	docID
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

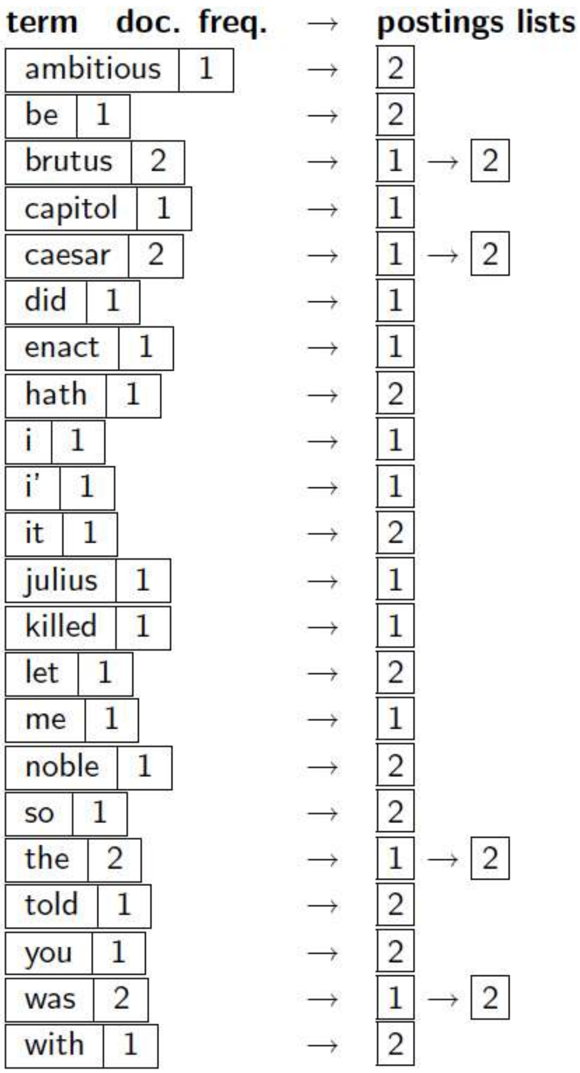
Indexer Steps: Dictionary & Postings

- Multiple term entries in a single document are merged.
- Split into Dictionary and Postings
- Doc. frequency information is added.

Why frequency?
Will discuss later.

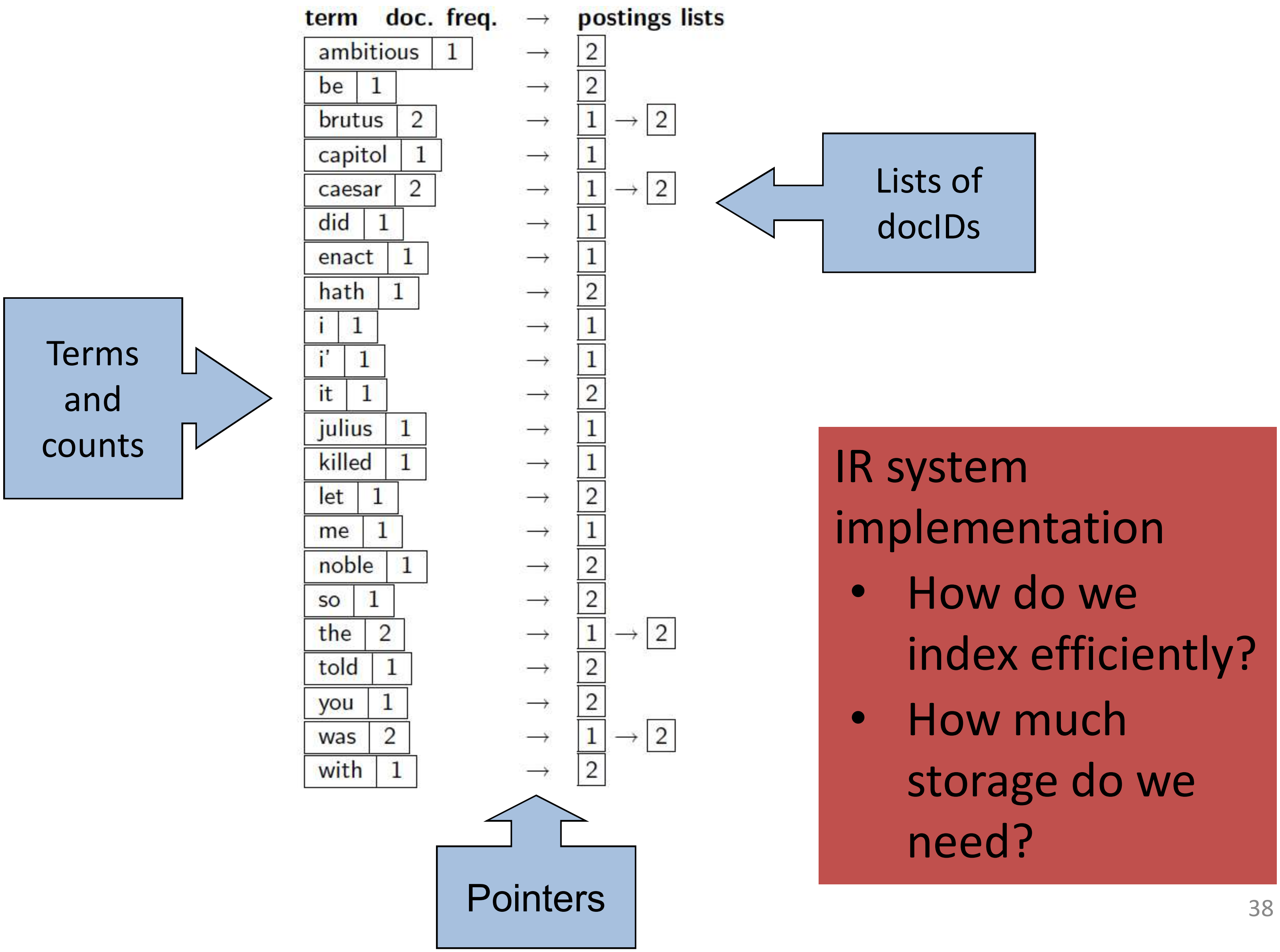


Term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2



term	doc. freq.	→	postings lists
ambitious	1	→	2
be	1	→	2
brutus	2	→	1 → 2
capitol	1	→	1
caesar	2	→	1 → 2
did	1	→	1
enact	1	→	1
hath	1	→	2
i	1	→	1
i'	1	→	1
it	1	→	2
julius	1	→	1
killed	1	→	1
let	1	→	2
me	1	→	1
noble	1	→	2
so	1	→	2
the	2	→	1 → 2
told	1	→	2
you	1	→	2
was	2	→	1 → 2
with	1	→	2

Where do we pay in Storage?



term	doc. freq.	→	postings lists
ambitious	1	→	2
be	1	→	2
brutus	2	→	1 → 2
capitol	1	→	1
caesar	2	→	1 → 2
did	1	→	1
enact	1	→	1
hath	1	→	2
i	1	→	1
i'	1	→	1
it	1	→	2
julius	1	→	1
killed	1	→	1
let	1	→	2
me	1	→	1
noble	1	→	2
so	1	→	2
the	2	→	1 → 2
told	1	→	2
you	1	→	2
was	2	→	1 → 2
with	1	→	2

Terms and counts

Lists of docIDs

Pointers

IR system implementation

- How do we index efficiently?
- How much storage do we need?

Query Processing: AND

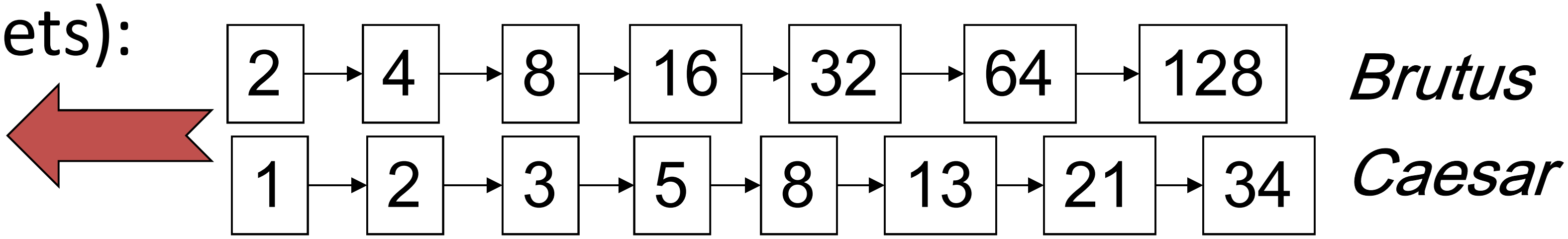
- Consider processing the query:

  Brutus *AND* ***Caesar***

  – Locate ***Brutus*** in the Dictionary;
    - Retrieve its postings.

  – Locate ***Caesar*** in the Dictionary;
    - Retrieve its postings.

  – "Merge" the two postings (intersect the document sets):

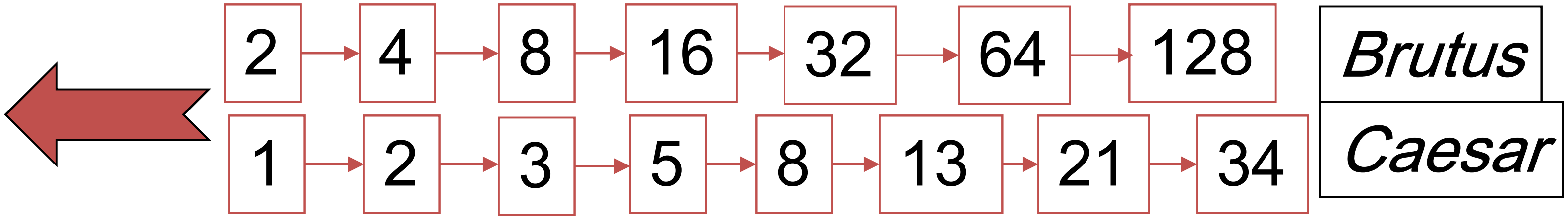


2 → 4 → 8 → 16 → 32 → 64 → 128 *Brutus*

1 → 2 → 3 → 5 → 8 → 13 → 21 → 34 *Caesar*

The Merge

- Walk through the two postings simultaneously, in time linear in the total number of postings entries



If the list lengths are *x* and *y*, the merge takes O(*x*+*y*) operations.

Crucial: postings sorted by docID.

Intersecting Two Postings Lists
(A “Merge” Algorithm)

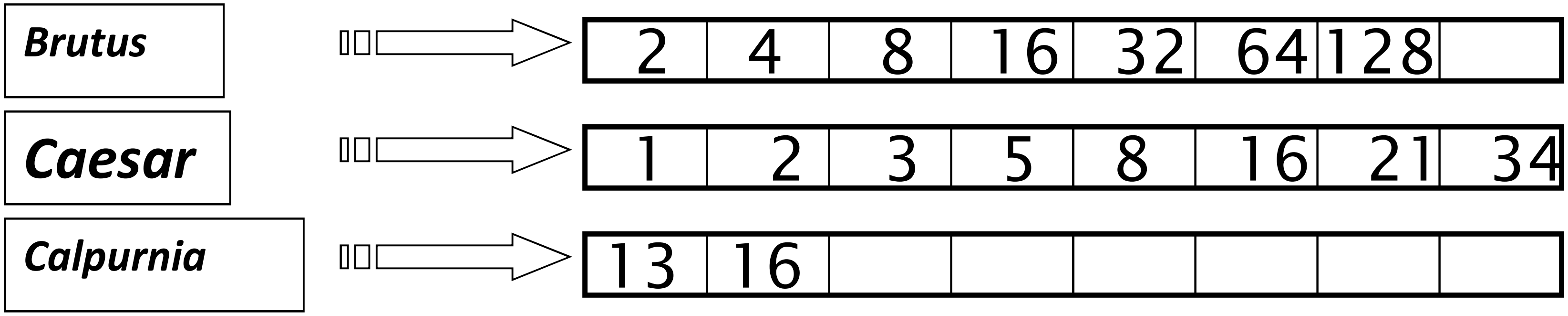
```
$\textsc{Intersect}(p_1, p_2)$
  $answer \leftarrow \langle\ \rangle$
  while $p_1 \neq \text{NIL}$ and $p_2 \neq \text{NIL}$
  do if $docID(p_1) = docID(p_2)$
          then $\textsc{Add}(answer, docID(p_1))$
               $p_1 \leftarrow next(p_1)$
               $p_2 \leftarrow next(p_2)$
          else if $docID(p_1) < docID(p_2)$
                  then $p_1 \leftarrow next(p_1)$
                  else $p_2 \leftarrow next(p_2)$
  return $answer$
```

Boolean Queries: Exact Match

- The Boolean retrieval model is being able to ask a query that is a Boolean expression:
  - – Boolean Queries are queries using *AND, OR* and *NOT* to join query terms
    - Views each document as a set of words
    - Is precise: document matches condition or not.
  - – Perhaps the simplest model to build an IR system on
- Primary commercial retrieval tool for 3 decades.
- Many search systems you still use are Boolean:
  - – Email, library catalog, Mac OS X Spotlight

Query Optimization

- What is the best order for query processing?
- Consider a query that is an *AND* of *n* terms.
- For each of the *n* terms, get its postings, then *AND* them together.



Brutus → 2 | 4 | 8 | 16 | 32 | 64 | 128

Caesar → 1 | 2 | 3 | 5 | 8 | 16 | 21 | 34

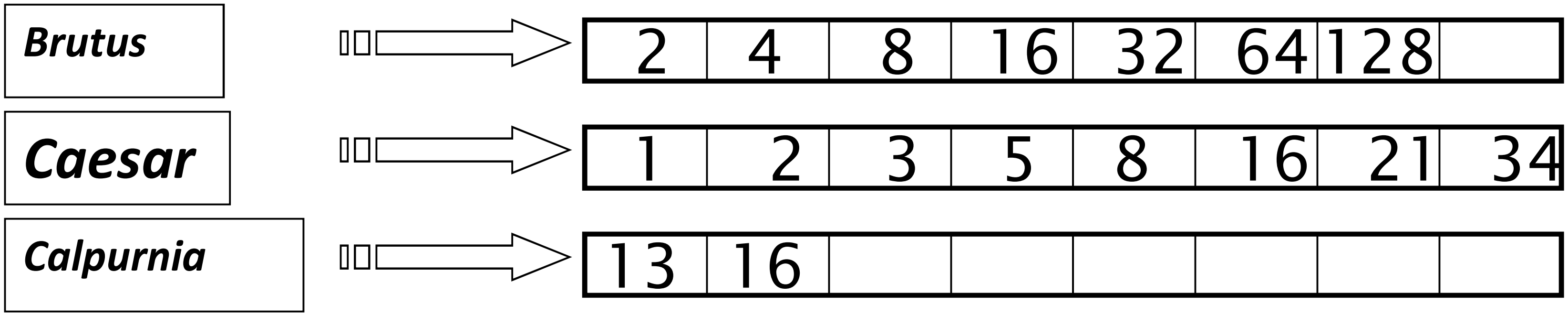
Calpurnia → 13 | 16

Query: ***Brutus*** *AND* ***Calpurnia*** *AND* ***Caesar***

Query Optimization Example

- Process in order of increasing freq:
  - *start with smallest set, then keep cutting further.*

This is why we kept document freq. in dictionary



Brutus →	2	4	8	16	32	64	128	
Caesar →	1	2	3	5	8	16	21	34
Calpurnia →	13	16						

Execute the query as (***Calpurnia*** *AND* ***Brutus)*** *AND* ***Caesar***.

More General Optimization

- e.g., *(**madding** OR **crowd**) AND (**ignoble** OR **strife**)*
- Get doc. freq.'s for all terms.
- Estimate the size of each *OR* by the sum of its doc. freq.'s (conservative).
- Process in increasing order of *OR* sizes.

Phrase Queries

- We want to be able to answer queries such as ***"stanford university"*** – as a phrase
- Thus the sentence *"I went to university at Stanford"* is not a match.
  - The concept of phrase queries has proven easily understood by users; one of the few "advanced search" ideas that works
  - Many more queries are *implicit phrase queries*
- For this, it no longer suffices to store only

  <*term* : *docs*> entries

A First Attempt: Bi-word Indexes

- Index every consecutive pair of terms in the text as a phrase
- For example the text "Friends, Romans, Countrymen" would generate the biwords
  - ***friends romans***
  - ***romans countrymen***
- Each of these bi-words is now a dictionary term
- Two-word phrase query-processing is now immediate.

Longer Phrase Queries

- Longer phrases can be processed by breaking them down
- ***stanford university palo alto*** can be broken into the Boolean query on biwords:

stanford university *AND* ***university palo*** *AND* ***palo alto***

Without the docs, we cannot verify that the docs matching the above Boolean query do contain the phrase.

Can have false positives!

Issues for Bi-word Indexes

- False positives, as noted before
- Index blowup due to bigger dictionary
  - Infeasible for more than bi-words, big even for them

- Bi-word indexes are not the standard solution (for all bi-words) but can be part of a compound strategy

Solution 2: Positional Indexes

- In the postings, store, for each ***term*** the position(s) in which tokens of it appear

<***term**,* number of docs containing ***term***;

doc1: position1, position2 … ;

doc2: position1, position2 … ;

etc.>

Positional Index Example

- For phrase queries, we use a merge algorithm recursively at the document level
- But we now need to deal with more than just equality

<***be***: 993427;
1: 7, 18, 33, 72, 86, 231;
2: 3, 149;
4: 17, 191, 291, 430, 434;
5: 363, 367, …>

Which of docs 1,2,4,5 could contain "***to be or not to be***"?

Processing a Phrase Query

- Extract inverted index entries for each distinct term: ***to, be, or, not.***
- Merge their *doc:position* lists to enumerate all positions with "***to be or not to be***".
  - ***to****:*
    - *2*:1,17,74,222,551; *4*:8,16,190,429,433; *7*:13,23,191; ...
  - ***be****:*
    - *1*:17,19; *4*:17,191,291,430,434; *5*:14,19,101; ...
- Same general method for proximity searches

Proximity Queries

- LIMIT! /3 STATUTE /3 FEDERAL /2 TORT
  - Again, here, /*k* means "within *k* words of".
- Clearly, positional indexes can be used for such queries; bi-word indexes cannot.

Positional Index Size

- A positional index expands postings storage *substantially* even though indices can be compressed
- Need an entry for each occurrence, not just once per document
- Index size depends on average document size
  - Average web page has <1000 terms
  - SEC filings, books, even some epic poems … easily 100,000 terms
- Consider a term with frequency 0.1%

Document size	Postings	Positional postings
1000	1	1
100,000	1	100

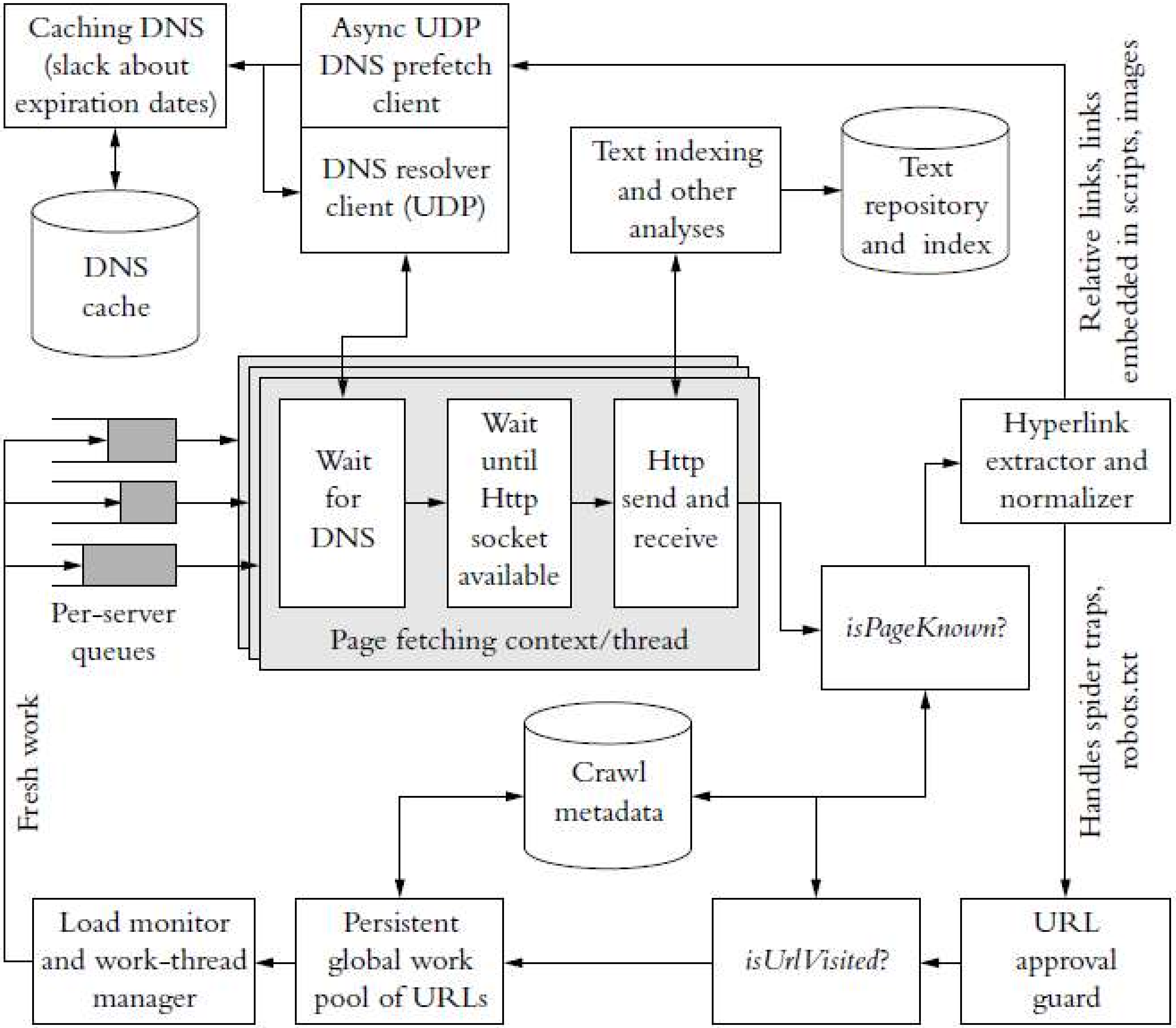
Today's Agenda

- Administrative information
- Course introduction and preview
- Text Indexing
- **Crawling**
- A 15-minute quiz

Crawling Basics

- HTTP request and status codes
- HTML and hyperlinks
- Infinite Web graph, changes as you crawl
- Performance issues in crawling
- Thousands of concurrent fetch threads per PC
- Duplicate URL elimination (hasBeenVisited)
- URL frontier for priority control
- Shared/distributed data structures
- Scaling up as crawling volume increases
- Designing frontier priority schemes

Web Crawler Block Diagram


Caching DNS (slack about expiration dates)
Async UDP DNS prefetch client
DNS resolver client (UDP)
DNS cache
Text indexing and other analyses
Text repository and index
Relative links, links embedded in scripts, images
Wait for DNS
Wait until Http socket available
Http send and receive
Page fetching context/thread
Hyperlink extractor and normalizer
Per-server queues
isPageKnown?
Handles spider traps, robots.txt
Fresh work
Crawl metadata
Load monitor and work-thread manager
Persistent global work pool of URLs
isUrlVisited?
URL approval guard

Web Crawler Issues

- DNS caching and prefetching
- Multithreading vs. socket selects
- Link extraction and normalization
- Eliminating already-visited URLs
- Avoiding link expansion from (near-) duplicate pages
- Spider traps
- Distributed, fault-tolerant data repository
- Robot exclusion
- Politeness and per-server job queue
- Basic frontier priority queue maintenance
- Crawler threads write append-style “log entries”

Webpage Processing during Crawling

- Features observed before crawling a page
  - Current inlink count (number of hrefs to page)
  - Anchor text from links into page
  - Current location/site-based aggregate stats
  - Pagerank in currently collected Web graph
- Features observed after crawling a page
  - Text, similarity with driving queries/profiles
  - Number of outlinks, same/different sites
  - and many more.

Today’s Agenda

- Administrative information
- Course introduction and preview
- Text Indexing
- Crawling
- **A 15-minute quiz**

15-minute Quiz

Put in your IIITH email id on answer-sheet

Results will be sent over email

Take-away Messages

- Web Mining is an exciting field
  - Not just to know how things work on Internet
  - But also to do research
  - And improve the current systems.
- The web is growing rapidly
  - in terms of data volume
  - in terms of varieties of problems
  - in terms of complexity of solutions
- We begun our journey towards understanding how things work on the web today

Preview of Lecture 2: Relevance Ranking

- Need for Relevance Ranking
- TF and IDF
- Vector Space Model
- Evaluation Metrics for Ranking

Thanks!

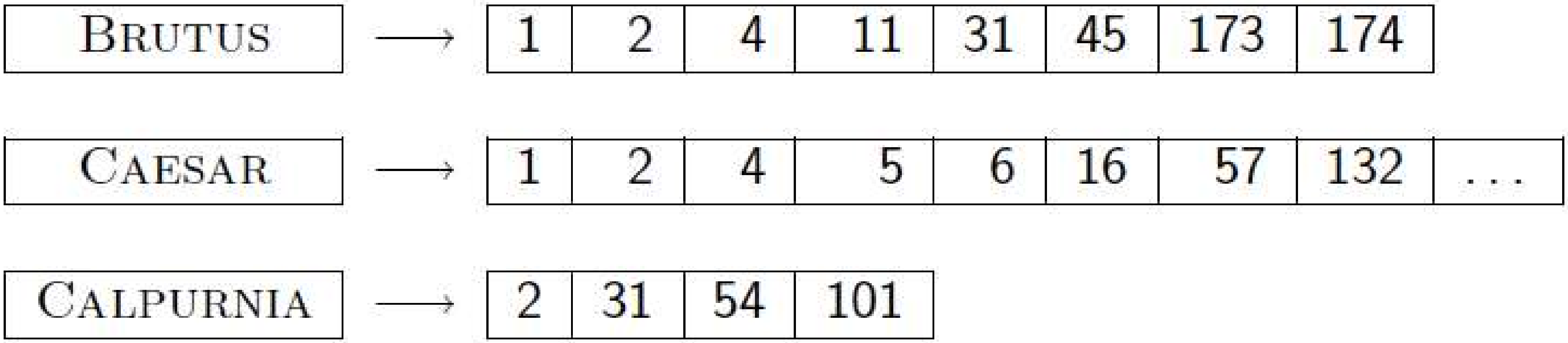
Additional Slides

Why Compression for Inverted Indexes?

- Dictionary
  - Make it small enough to keep in main memory
  - Make it so small that you can keep some postings lists in main memory too
- Postings file(s)
  - Reduce disk space needed
  - Decrease time needed to read postings lists from disk
  - Large search engines keep a significant part of the postings in memory.
    - Compression lets you keep more in memory
- We will devise various IR-specific compression schemes

Index Compression

- Compressing postings
  - For starters consider non-positional postings
- Compressing the dictionary



BRUTUS → 1 | 2 | 4 | 11 | 31 | 45 | 173 | 174

CAESAR → 1 | 2 | 4 | 5 | 6 | 16 | 57 | 132 | ...

CALPURNIA → 2 | 31 | 54 | 101

Postings Compression

- The postings file is much larger than the dictionary, factor of at least 10.
- Key desideratum: store each posting compactly.
- A posting for our purposes is a docID.
- For Reuters (800,000 documents), we would use 32 bits per docID when using 4-byte integers.
- Alternatively, we can use $\log_2 800{,}000 \approx 20$ bits per docID.
- Our goal: use far fewer than 20 bits per docID.

Postings: Two Conflicting Forces

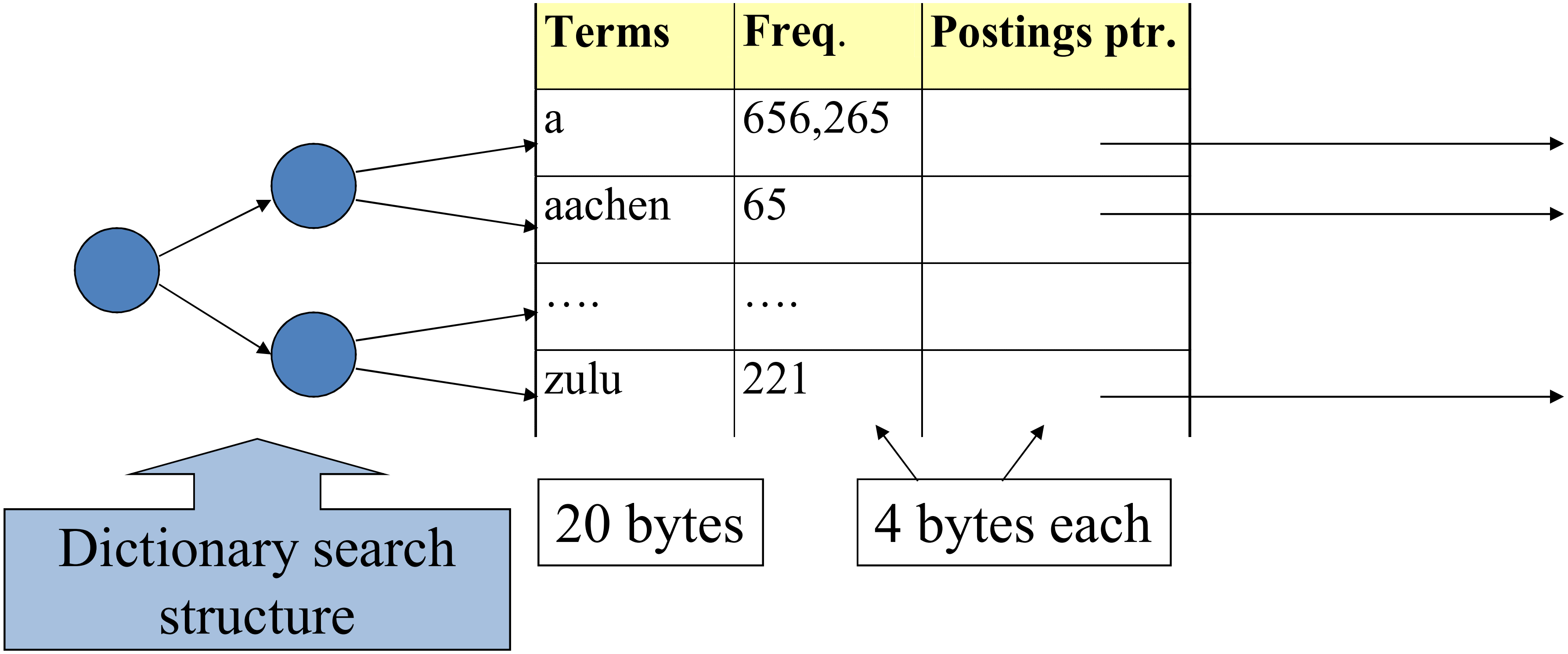
- A term like ***arachnocentric*** occurs in maybe one doc out of a million – we would like to store this posting using $\log_2$ 1M ~ 20 bits.
- A term like ***the*** occurs in virtually every doc, so 20 bits/posting is too expensive.
  - Prefer 0/1 bitmap vector in this case

Postings File Entry

- We store the list of docs containing a term in increasing order of docID.
  - ***computer***: 33,47,154,159,202 …
- Consequence: it suffices to store *gaps*.
  - 33,14,107,5,43 …
- Hope: most gaps can be encoded/stored with far fewer than 20 bits.
- Heads we win, tails they lose
  - Either a word is rare
  - Or the gaps are small

Dictionary Storage - First Cut

- Array of fixed-width entries
  - ~400,000 terms; 28 bytes/term = 11.2 MB.



Terms	Freq.	Postings ptr.
a	656,265	
aachen	65	
….	….	
zulu	221	

Dictionary search structure

20 bytes

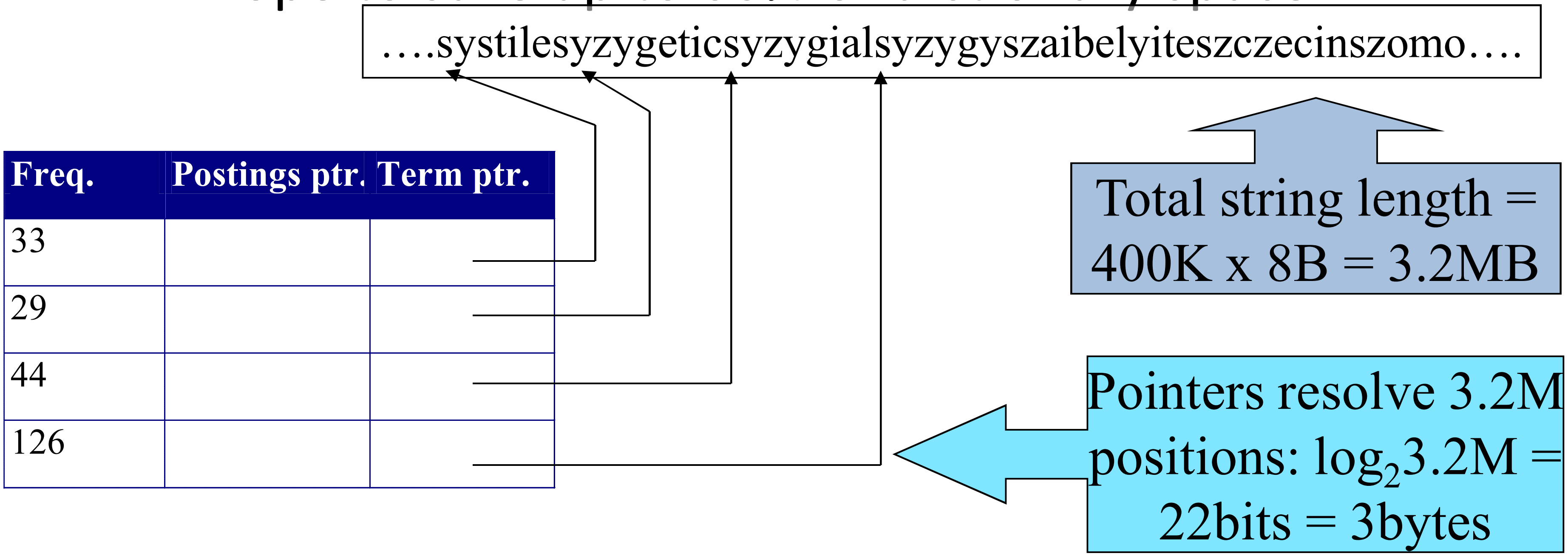
4 bytes each

Fixed-Width Terms are Wasteful

- Most of the bytes in the **Term** column are wasted – we allot 20 bytes for 1 letter terms.
  - And we still can't handle *supercalifragilisticexpialidocious* or *hydrochlorofluorocarbons.*
- Written English averages ~4.5 chars/word
  - Exercise: Why is/isn't this the number to use for estimating the dictionary size?
- Ave. English dictionary word: ~8 characters
  - How do we use ~8 characters per dictionary term?

Compressing the Term List: Dictionary-as-a-String

- Store dictionary as a (long) string of characters:
  - Pointer to next word shows end of current word
  - Hope to save up to 60% of dictionary space.



….systilesyzygeticsyzygialsyzygyszaibelyiteszczecinszomo….

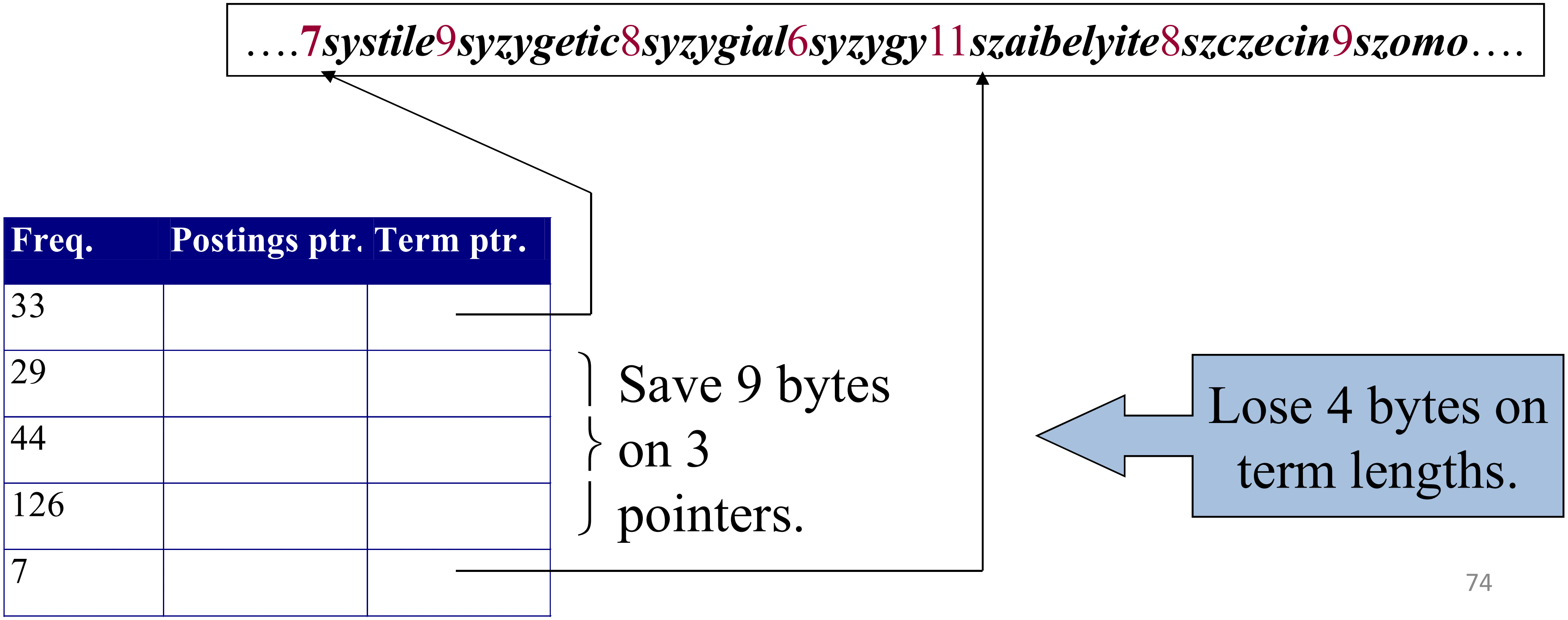
Freq.	Postings ptr.	Term ptr.
33		
29		
44		
126		

Total string length = 400K x 8B = 3.2MB

Pointers resolve 3.2M positions: $\log_2 3.2M$ = 22bits = 3bytes

Blocking

- Store pointers to every *k*th term string.
  - Example below: *k*=4.
- Need to store term lengths (1 extra byte)



….7***systile***9***syzygetic***8***syzygial***6***syzygy***11***szaibelyite***8***szczecin***9***szomo***….

Freq.	Postings ptr.	Term ptr.
33		
29		
44		
126		
7		

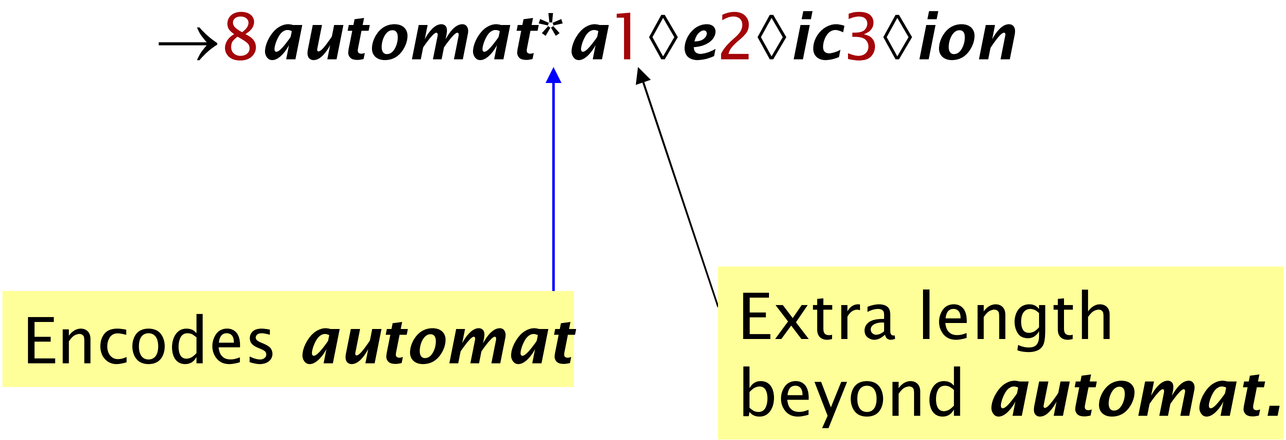
Save 9 bytes on 3 pointers.

Lose 4 bytes on term lengths.

Front Coding

- Sorted words commonly have long common prefix – store differences only
- (for last *k-1* in a block of *k*)

8***automata***8***automate***9***automatic***10***automation***



→8***automat**** ***a***1◊***e***2◊***ic***3◊***ion***

Encodes ***automat***

Extra length beyond ***automat.***

Begins to resemble general string compression.

RCV1 dictionary compression summary

Technique	Size in MB
Fixed width	11.2
Dictionary-as-String with pointers to every term	7.6
Also, blocking *k* = 4	7.1
Also, Blocking + front coding	5.9

Disclaimers

- This course represents opinions of the instructor only. It does not reflect views of Microsoft or any other entity (except of authors from whom the slides have been borrowed).
- Algorithms, techniques, features, etc mentioned here might or might not be in use by Microsoft or any other company.
- Lot of material covered in this course is borrowed from slides across many universities and conference tutorials. These are gratefully acknowledged.

Thanks!