# Information Retrieval

CS 547/DS 547

Worcester Polytechnic Institute
Department of Computer Science
Instructor: Prof. Kyumin Lee

# A Little About Me

- Software Engineer at NHN from 2006-2008, South Korea
- Ph.D. at Texas A&M in 2013
- Research Internship at eBay Research Labs in 2011 and IBM Research in 2012
- Associate Professor at WPI
- Summer Research Fellow at Air Force Research Lab in 2022
- Director of Infolab: Information Retrieval, Machine Learning/AI, Natural Language Processing, Social Computing and AI for social good

# TA

Di You

# Now… Your turn

# What is Information Retrieval?

- …
- …
- …

# IR is Everywhere …

## Web search engines

Google bing Yandex Baidu 百度 YAHOO! Aol. Ask

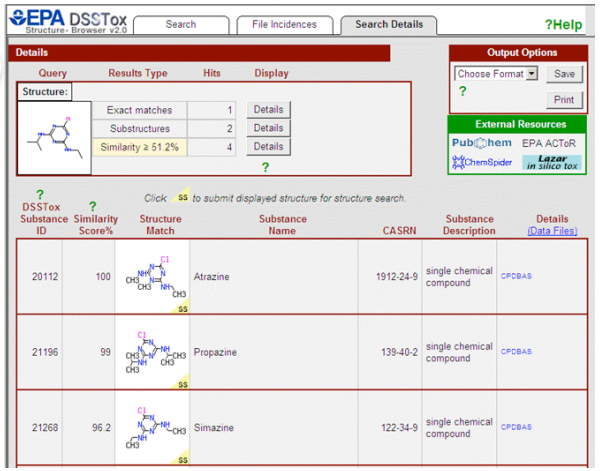
## Recommenders

amazon.com Spotify®

last·fm the social music revolution

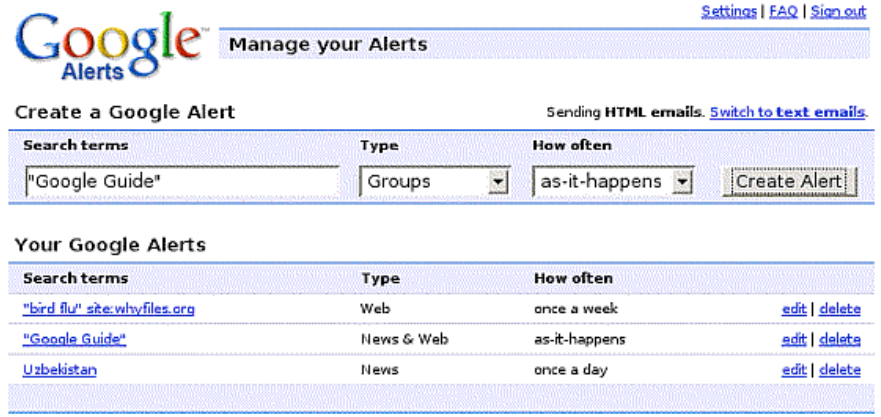
## Domain-specific search

Westlaw®




## Info filtering / classification

SpamAssassin

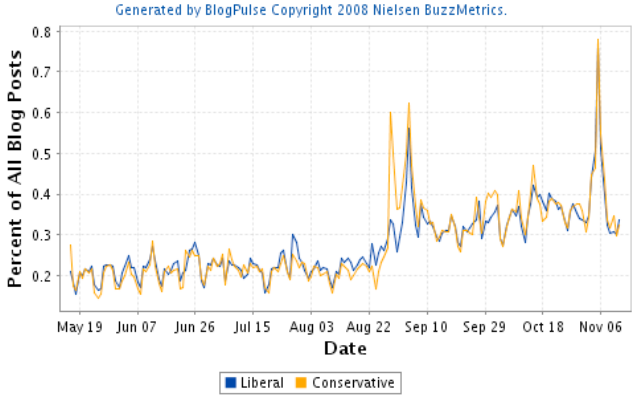


# IR is Everywhere …

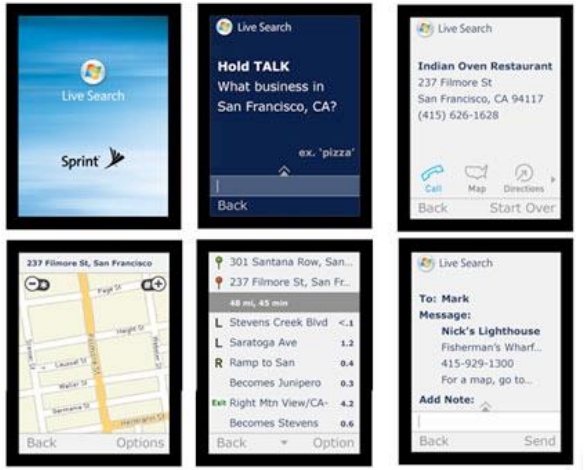
## Social search/Web




## Topic detection and tracking




## Mobile and location-based



# IR is Everywhere …

- Domain specific applications of information retrieval
  - Expert search finding
  - Genomic information retrieval
  - Geographic information retrieval
  - Information retrieval for chemical structures
  - Information retrieval in software engineering
  - Legal information retrieval
  - Vertical search (domain/topic specific search)

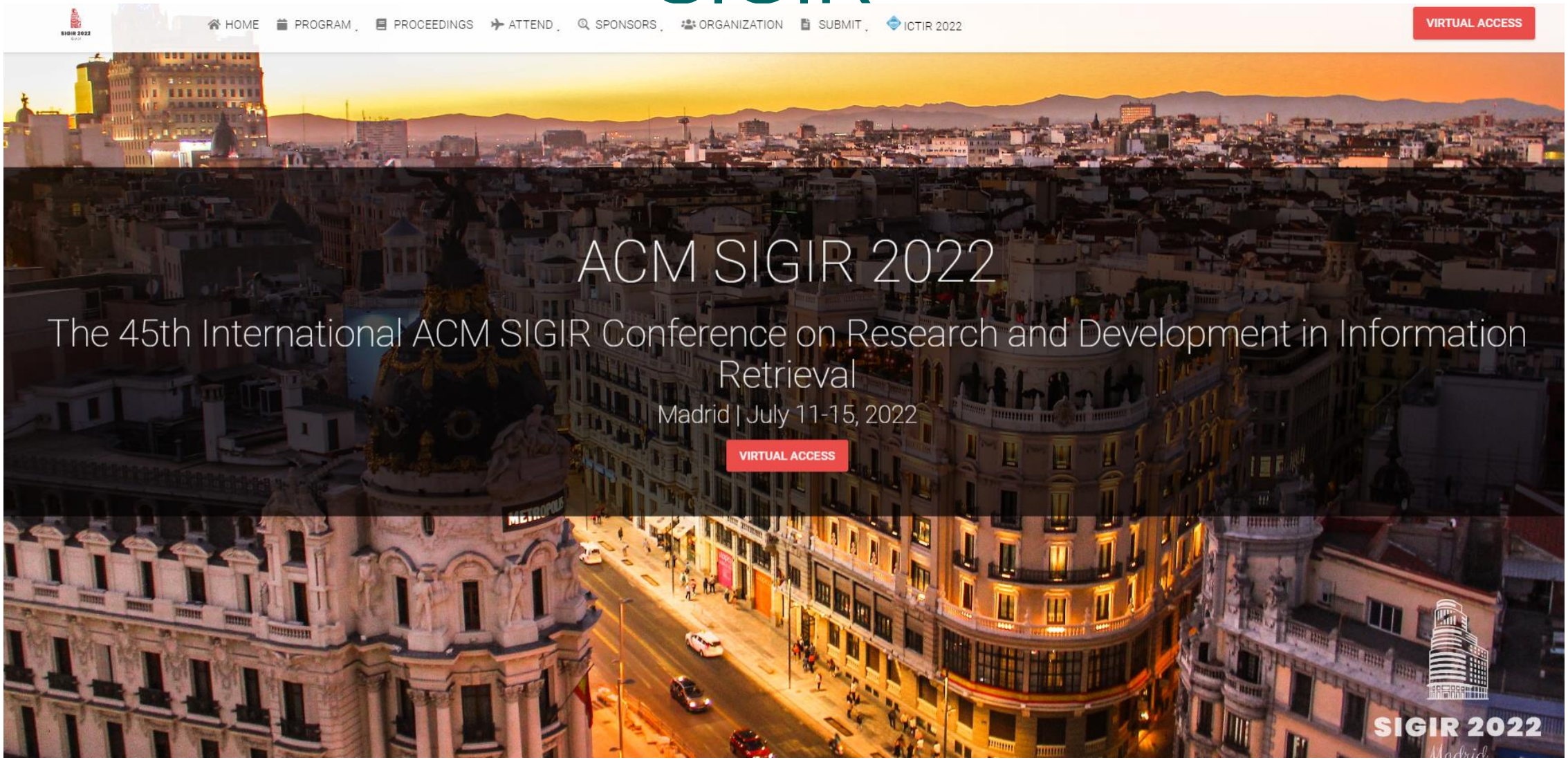
# IR is Everywhere …

- General applications of information retrieval
  - Digital Libraries
  - Information Filtering
    - Recommender Systems
  - Media Search
    - Blog, image, music, news, speech, video
  - Search engines
    - Desktop, enterprise, federated, mobile, social, Web search
  - Retrieval-augmented text generation

# IR is Everywhere …

- Other retrieval methods
  - Adversarial information retrieval
  - Automatic document summarization
  - Cross-lingual retrieval
  - Document classification
    - Spam filtering
  - Question answering
  - Structured document retrieval
  - Topic detection and tracking

# SIGIR




https://sigir.org/sigir2022/

https://sigir.org/sigir2022/program/schedu

# This course

- What makes a system like Google, Yahoo, Bing or Amazon?
  - How does it gather information?
  - What tricks does it use?
- How can those approaches be made better?
- What can we do to make things work more quickly?
- How do we decide whether it works well?
- ...

# So ... What is Information Retrieval?

- Information Retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).

This smells a bit like

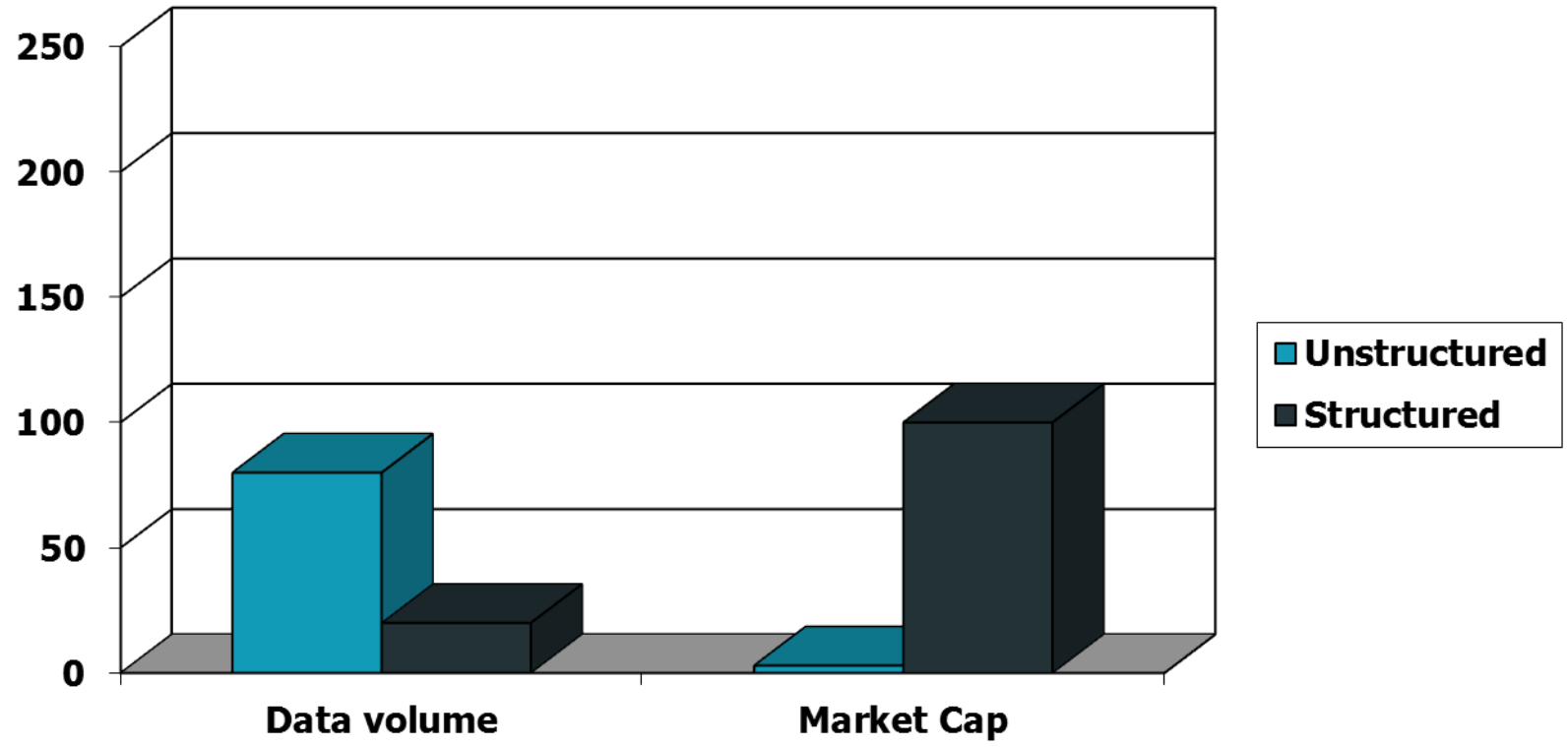
Databases ...

What’s the difference?

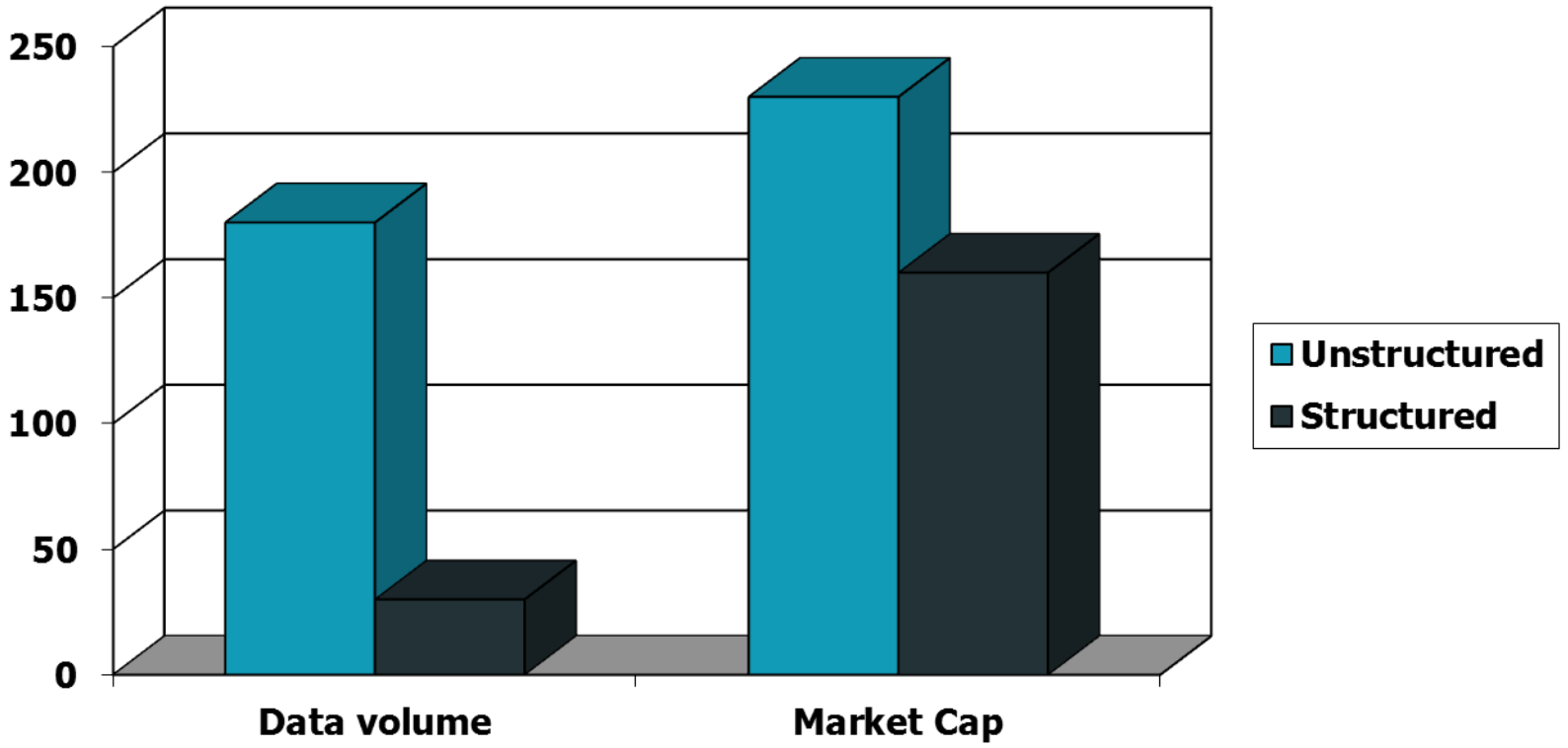
# Information Retrieval versus Databases

| | Databases | IR |
|---|---|---|
| Data | **Structured** | **Unstructured** |
| Fields | **Clear semantics** (SSN, age) | **No fields** (other than text) |
| Queries | **Defined** (relational algebra, SQL) | **Free text** (natural language, Boolean) |
| Recoverability | **Critical** (concurrency control, recovery, atomic operations) | **Downplayed** (though still an issue) |
| Matching | **Exact** (results are always "correct") | **Imprecise** (need to measure effectiveness) |

# Unstructured (text) vs. structured (database) data in the mid-nineties



# Unstructured (text) vs. structured (database) data today



# Course Objectives

- Introduce
  - theory, design, and implementation of text-based and Web-based information retrieval systems.

- Study
  - crawling, Indexing, vector space model, web search, link-based algorithms, recommender systems, and etc.

# Goal of the Class

- Understand the key concepts and models relevant to information retrieval, including efficient text indexing, vector space model, Web search.
- Design, implement, and evaluate the core algorithms underlying a fully functional IR system, including the indexing, retrieval, and ranking components.
- Identify the salient features and apply recent research results in information retrieval.

# Course Structure and Administrivia

# Course Information

- Instructor
  - Kyumin Lee
  - kmlee@wpi.edu
  - Office: Unity Hall 363
  - Office hours: W: 4:00-5:00 pm
- TA
  - Di You
  - dyou@wpi.edu
  - Office: Unity Hall 341
  - Office hours: T: 1:00-2:00 pm, and F: 2:30-3:30 pm
- Class hours:
  - 6:00-8:50 pm W
  - Classroom: Goddard Hall 227

# Course Information

- Course web page
  - Check the course Canvas page and Schedule page
  - https://canvas.wpi.edu/

- Communication
  - Will post important announcements via Canvas mailing list
  - Send us a question via email

# Course Materials

- Course readings will be drawn from the following resources:
  - Introduction to Information Retrieval (2008)
  - Mining of Massive Datasets (2020)
  - Introduction to Neural Information Retrieval (2019)
  - Research papers

# Course Communication

- Check schedule page to understand upcoming deadlines and topics
- I will post important announcements to the Canvas page and sometimes send messages via Canvas message service
  - Make sure to check the Canvas messages or get them via email.
  - https://community.canvaslms.com/t5/Question-Forum/How-do-I-get-my-emails-in-canvas-to-be-sent-to-my-normal-email/td-p/123356
- The best way to discuss general questions or share something cool stuff is to email it via Canvas mailing list.

# Class Structure

- Lectures
  - By instructor -- I'll teach information retrieval techniques
  - By us - Discussion and interaction in the class

- Your part
  - In-class discussion
  - Quizzes
  - Homework
    - 4 assignments
  - Exams
  - Project
    - Proposal writing&presentation, execution, workshop presentation

- Participation
  - Ask good questions

# Grading

- 5% Quizzes
- 24% Assignments
- 20% Midterm
- 20% Final
- 31% Project

# Quizzes

- 2 Quizzes
  - True/false, multiple choice, and/or short answer questions
  - Refer to the Schedule page on Canvas

  Or
  - https://web.cs.wpi.edu/~kmlee/cs547/schedule.htm

# Assignments

- 4 assignments
  - Be familiar with Python
- Submit your solution to Canvas
- Late day policy: look at the syllabus
  - http://web.cs.wpi.edu/~kmlee/cs547/syllabus.pdf

# Midterm and Final

# Exams

- The exams are closed book.
- You may bring one standard 8.5" by 11" piece of paper with any notes you think appropriate or significant (front and back).
- You may bring a calculator but no phone/laptop

# Project

# The Project

- 3~4 person team
- Project idea:
  - Propose anything you wish (related to IR and/or social systems)
  - You are encouraged to talk to me
- In the end of the semester, I will collect self and peer evaluation form.
- **31% of your final grade!!**

# Project Grading Criteria

- [7%] Project Proposal Writing: March 17 by 11:59pm
- [5%] Project Proposal Presentation: March 21
- [8%] Project website: April 25 by 11:59pm
- [11%] Project Workshop: April 26 in-class

# So far…

- Read syllabus
- Be familiar with Python for Assignments
- Great News!
  - Di You, the TA, will hold a python 101 session next Tuesday at 1pm at Unity Hall 520
- Form a team and notify the names of your team members by Jan 25.

# So far…

- Information Retrieval (IR) is finding material (usually documents) of an **unstructured** nature (usually text) that satisfies an **information need** from within **large collections** (usually stored on computers).

# Next

Boolean Retrieval

# Unstructured data in 1680

- 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)
    - The **key feature** of modern search engines

# Term-document incidence

| | 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 |

1 if play contains word, 0 otherwise

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

# Term-document incidence

| | 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 |

1 if play contains word, 0 otherwise

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

# 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.

Q: Which plays of Shakespeare contain the words ***Brutus*** *AND* ***Caesar*** but *NOT* ***Calpurnia***?

# Answers to query

- ## Antony and Cleopatra, Act III, Scene ii

*Agrippa* [Aside to DOMITIUS ENOBARBUS]: Why, Enobarbus,
When Antony found Julius ***Caesar*** dead,
He cried almost to roaring; and he wept
When at Philippi he found ***Brutus*** slain.

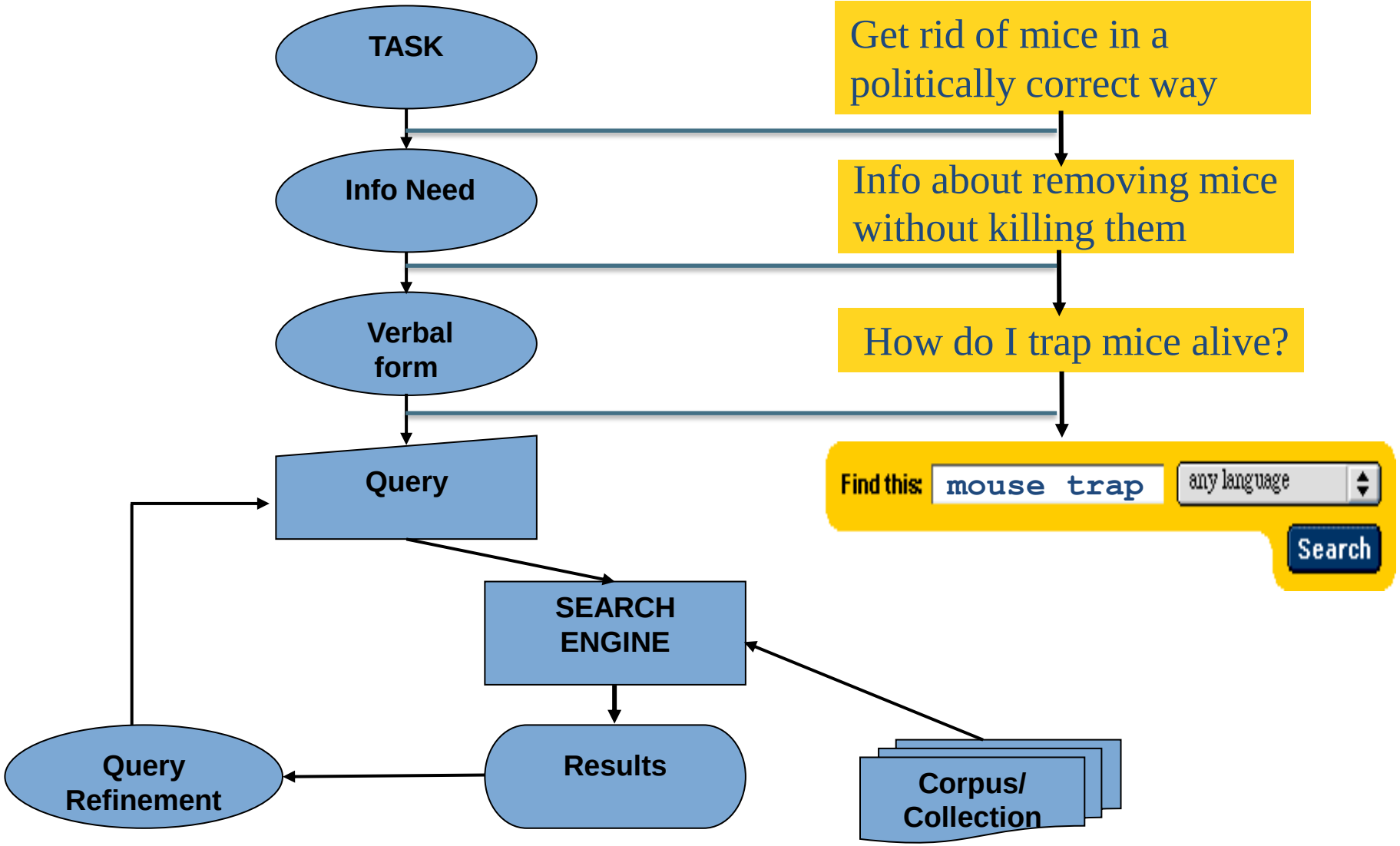
- ## Hamlet, Act III, Scene ii

*Lord Polonius:* I did enact Julius ***Caesar*** I was killed i' the Capitol; ***Brutus*** killed me.

# Basic assumptions of Information Retrieval

- Collection: Fixed set of documents
- Goal: Retrieve documents with information that is relevant to the user's information need and helps the user complete a task

# The classic search model



# How good are the retrieved docs?

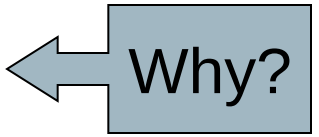
- *Precision* : Fraction of retrieved docs that are relevant to user's information need
- *Recall* : Fraction of relevant docs in collection that are retrieved
- More precise definitions and measurements to follow in later lectures

# 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.

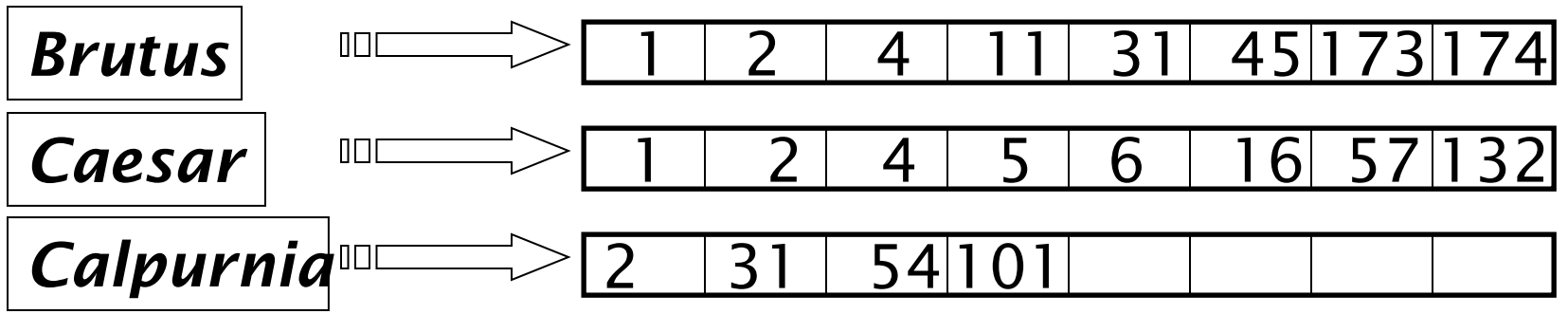
# Can’t build the matrix

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



# Inverted index

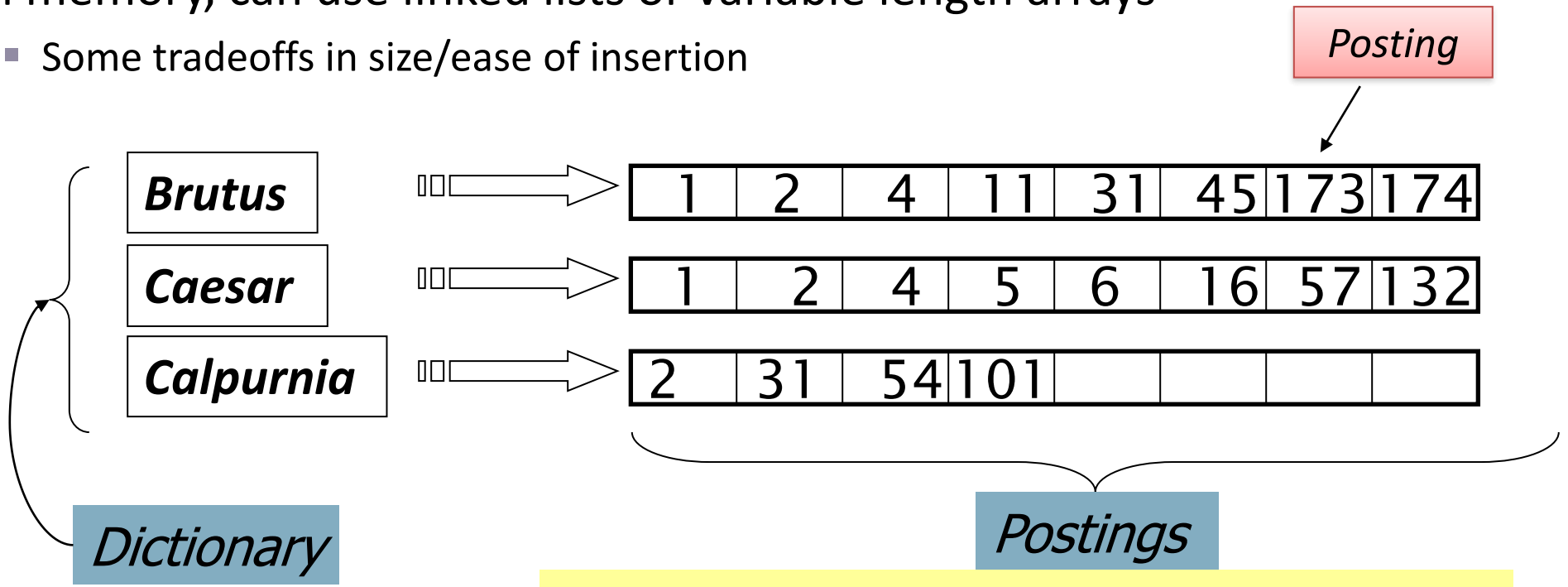
- For each term *t*, we must store a list of all documents that contain *t*.
  - Identify each by a **docID**, a document serial number
- Can we use fixed-size arrays for this?




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

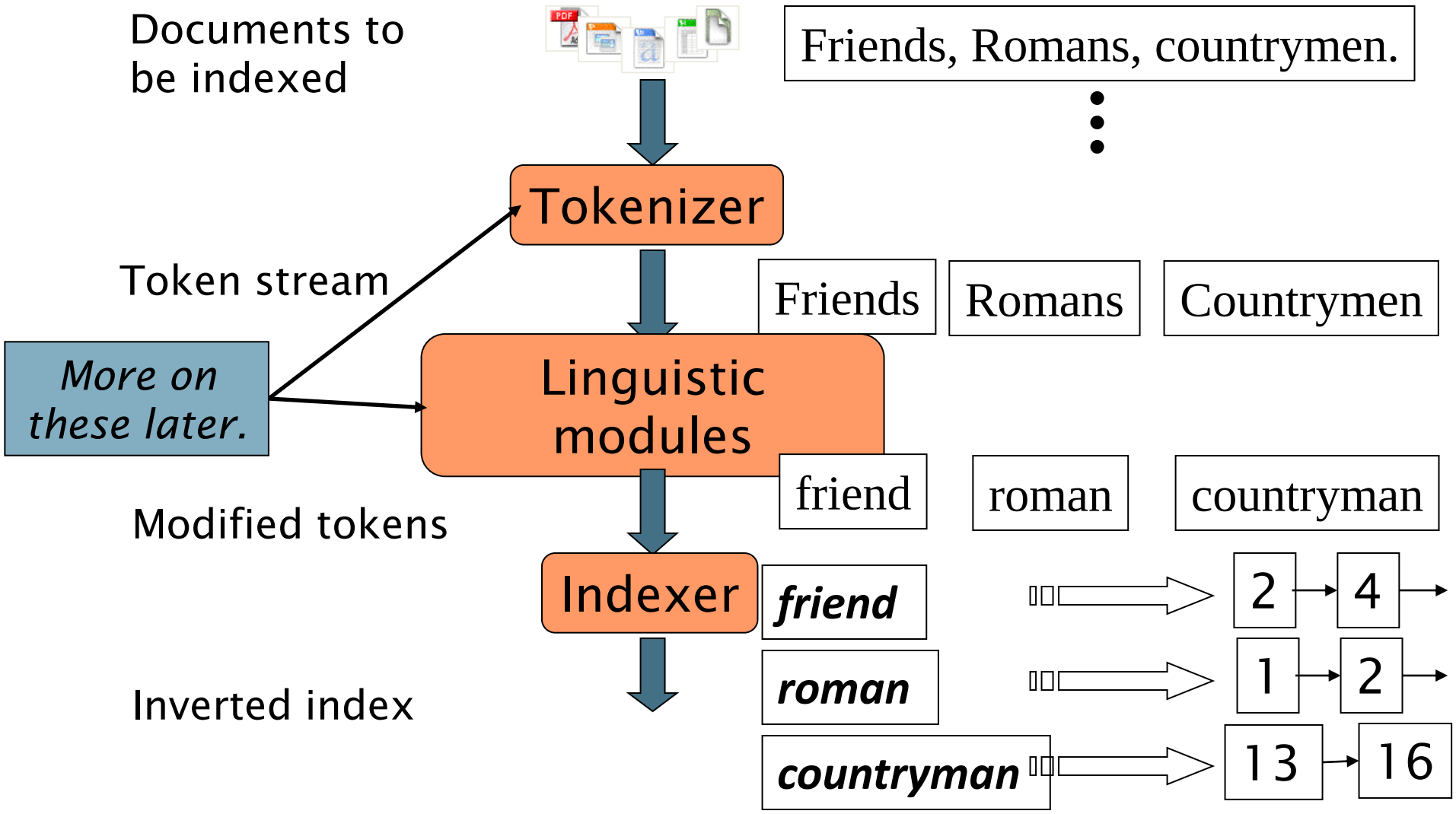
# Inverted index

- 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
    - Some tradeoffs in size/ease of insertion




Sorted by docID (more later on why).

# Inverted index construction



# 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

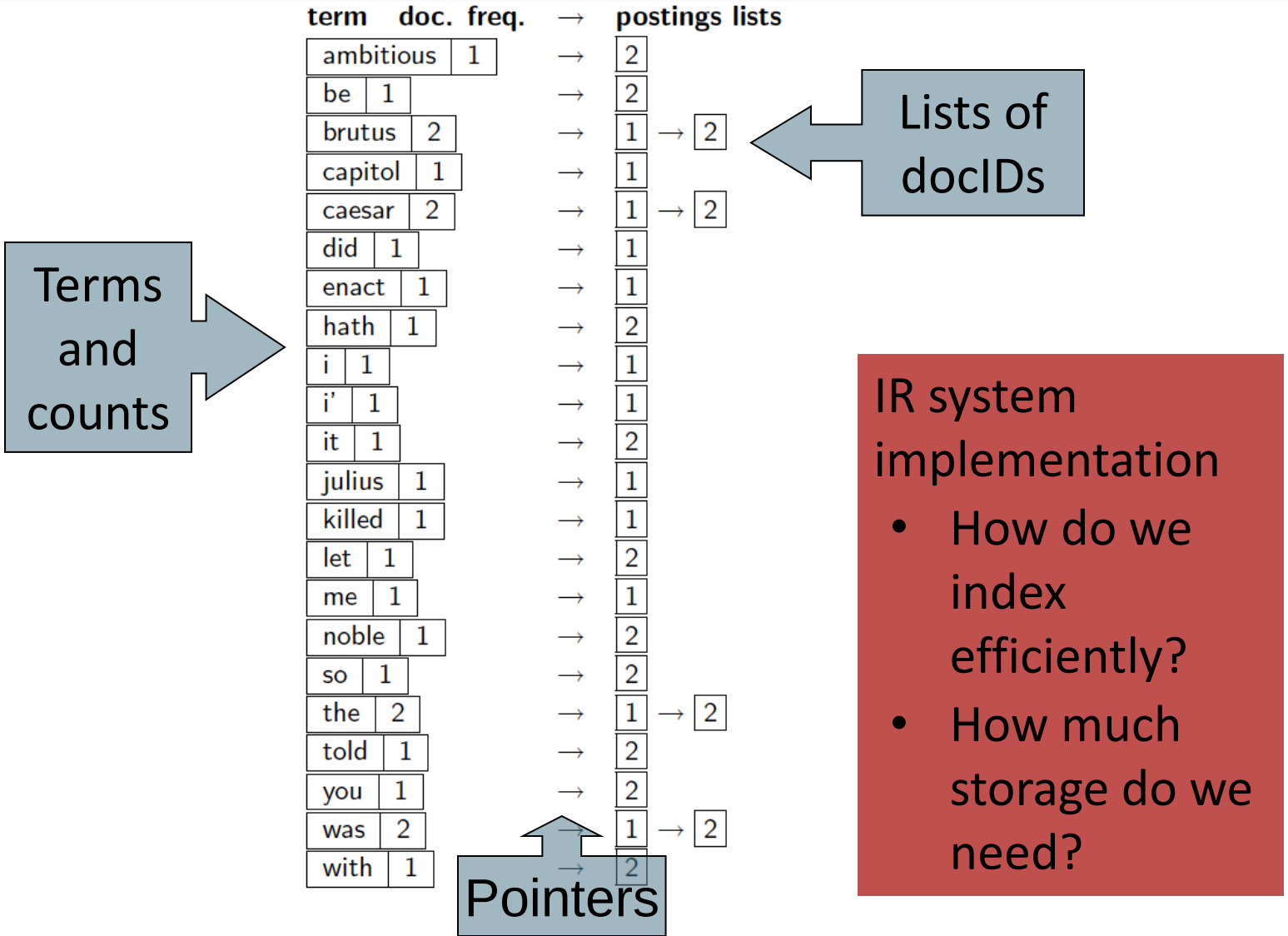
- Multiple term entries in a single document are merged.
- Split into Dictionary and Postings
- Doc. frequency information is added.

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 |

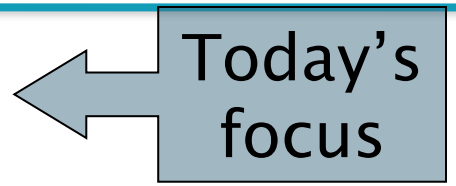
| term | doc. freq. | → | postings lists |
|---|---|---|---|
| ambitious | 1 | → | 2 |
| be | 1 | → | 2 |
| brutus | 2 | → | 1 → 2 |
| capitol | 1 | → | 1 |
| caesar | 2 | → | 1 → 2 |
| did | 1 | → | 1 |
| enact | 1 | → | 1 |
| hath | 1 | → | 2 |
| i | 1 | → | 1 |
| i' | 1 | → | 1 |
| it | 1 | → | 2 |
| 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?



# The index we just built

- How do we process a query?
  - Later - what kinds of queries can we process?

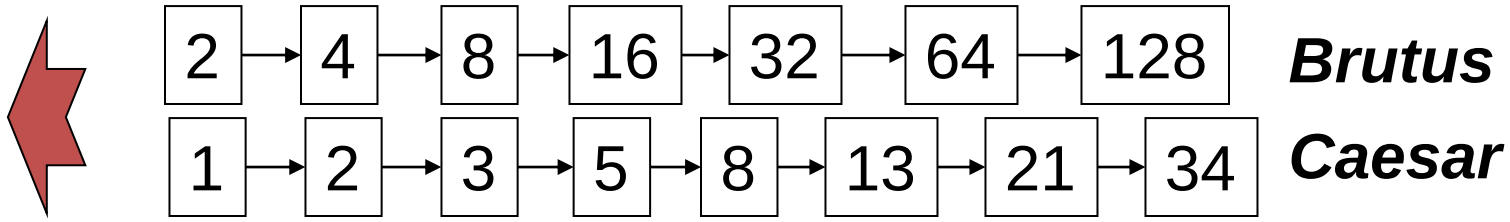


# Query processing: AND

- Consider processing the query:

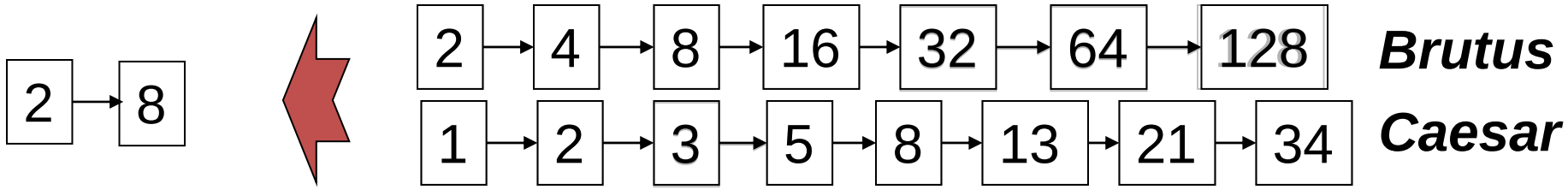
  ***Brutus*** *AND* ***Caesar***

  - Locate ***Brutus*** in the Dictionary;
    - Retrieve its postings.
  - Locate *Caesar* in the Dictionary;
    - Retrieve its postings.
  - "Merge" the two postings:



# The merge

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




If list lengths are *x* and *y*, merge takes O(*x*+*y*) operations.
Crucial: postings sorted by docID.

# Intersecting two postings lists (a “merge” algorithm)

```
INTERSECT(p1, p2)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then ADD(answer, docID(p1))
 5             p1 ← next(p1)
 6             p2 ← next(p2)
 7        else if docID(p1) < docID(p2)
 8                then p1 ← next(p1)
 9                else p2 ← next(p2)
10  return answer
```

# Boolean queries: Exact match

- The Boolean retrieval model is being able to ask a query that is a Boolean expression:
  - Boolean Queries use *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 Spotlight

# Example: WestLaw http://www.westlaw.com/

- Largest commercial (paying subscribers) legal search service (started 1975; ranking added 1992)
- Majority of users *still* use boolean queries
- Example query:
  - What is the statute of limitations in cases involving the federal tort claims act?
  - LIMIT! /3 STATUTE ACTION /S FEDERAL /2 TORT /3 CLAIM
    - /3 = within 3 words, /S = in same sentence

# Boolean queries: More general merges

- Exercise: Adapt the merge for the queries:

  ***Brutus*** *AND NOT* ***Caesar***

  ***Brutus*** *OR NOT* ***Caesar***

Can we still run through the merge in time O(*x*+*y*)?

What can we achieve?

# Exercise Solution

- Brutus AND NOT Caesar
  - Time is O(x+y). Instead of collecting documents that occur in both postings lists, collect those that occur in the first one and not in the second

- Brutus OR NOT Caesar
  - Time is O(N) (where N is the total number of documents in the collection) assuming we need to return a complete list of all documents satisfying the query. This is because the length of the result list is only bounded by N, not by the length of the postings lists.

# Merging

What about an arbitrary Boolean formula?

***(Brutus** OR **Caesar)** AND NOT*

***(Antony** OR **Cleopatra)***

- Can we always merge in “linear” time?
  - Linear in what?
- Can we do better?

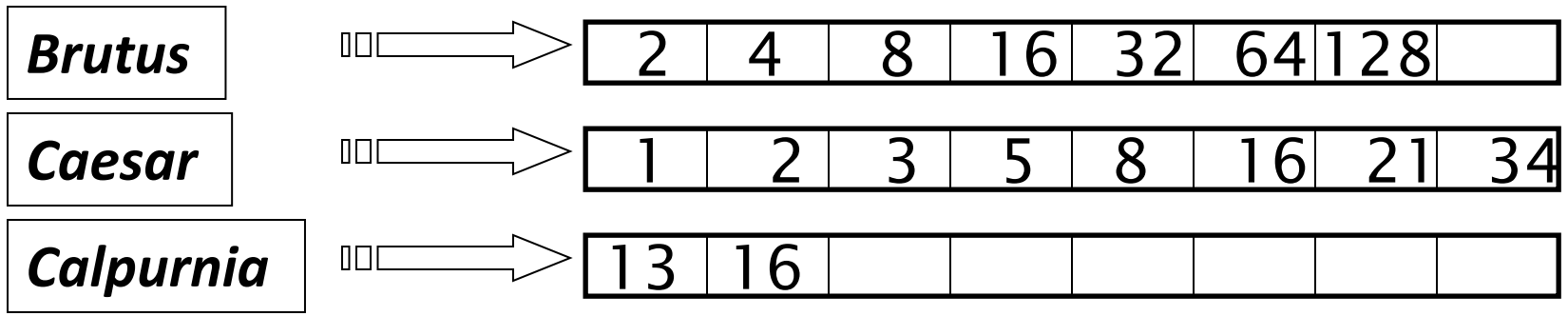
# Solution

- We can always intersect in O(qN) where q is the number of query terms and N the number of documents, so the intersection time is linear in the number of documents and query terms. Since the tightest bound for the size of the result list is N, the number of documents, one cannot do better than O(N).

- But… still we can reduce computation time even though time complexity is still O(N). How?

# Query optimization

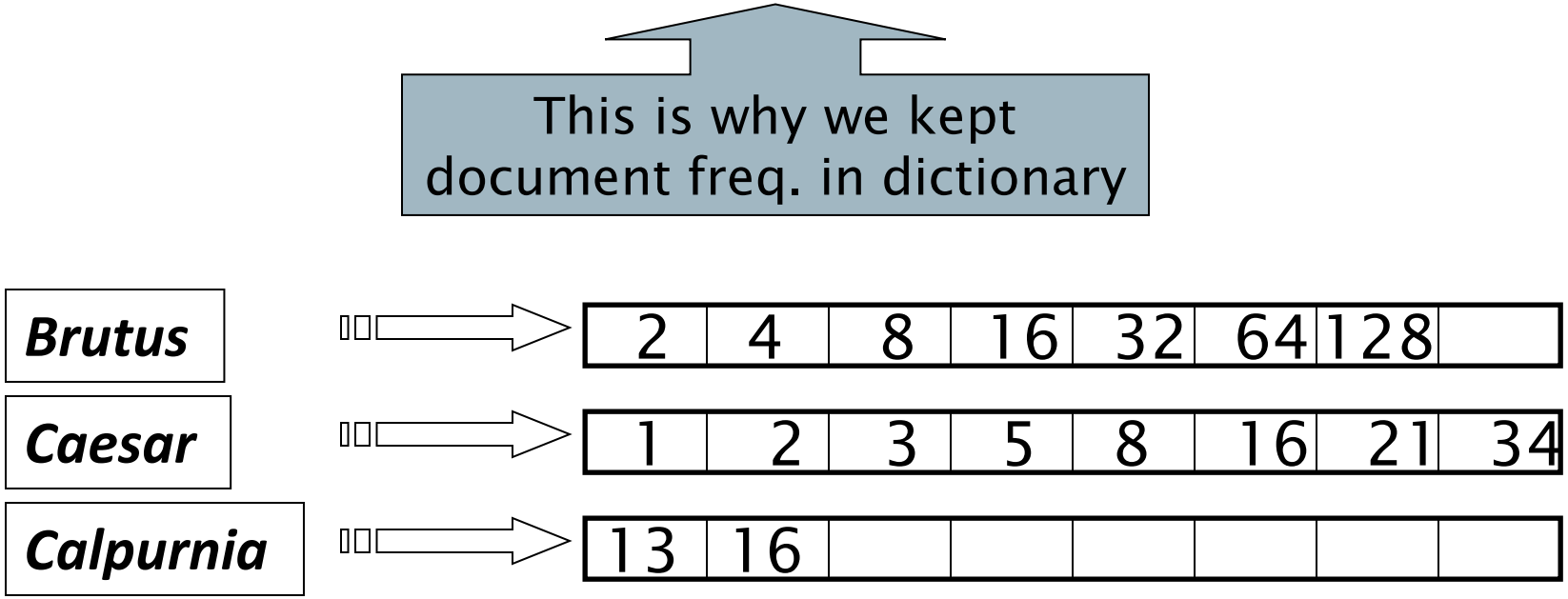
- Consider a query that is an *AND* of *n* terms.
- For each of the *n* terms, get its postings, then *AND* them together.
- What is the best order for query processing?




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

# Query optimization example

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




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.

# Exercise

- Recommend a query processing order for

***(tangerine** OR **trees)** AND*
***(marmalade** OR **skies)** AND*
***(kaleidoscope** OR **eyes)***

| Term | Freq |
|---|---|
| eyes | 213312 |
| kaleidoscope | 87009 |
| marmalade | 107913 |
| skies | 271658 |
| tangerine | 46653 |
| trees | 316812 |

# Exercise Solution

- Using the conservative estimate of the length of unioned postings lists, the recommended order is: (kaleidoskope OR eyes) (300,321) AND (tangerine OR trees) (363,465) AND (marmalade OR skies) (379,571)

# What's ahead in IR?
# Beyond term search

- What about phrases?
  - ***Worcester Polytechnic Institute***
- Proximity: Find ***Musk*** *NEAR* ***Tesla***.
  - Need index to capture position information in docs.
- Zones in documents: Find documents with (*author* = ***Ullman***) *AND* (text contains ***automata***).

# Evidence accumulation

- 1 vs. 0 occurrence of a search term
  - 2 vs. 1 occurrence
  - 3 vs. 2 occurrences, etc.
  - Usually more seems better
- Need term frequency information in docs

# Ranking search results

- Boolean queries give inclusion or exclusion of docs.
- Often we want to rank/group results
  - Need to measure proximity from query to each doc.
  - Need to decide whether docs presented to user are singletons, or a group of docs covering various aspects of the query.

# Clustering, classification and ranking

- Clustering: Given a set of docs, group them into clusters based on their contents.
- Classification: Given a set of topics, plus a new doc *D*, decide which topic(s) *D* belongs to.

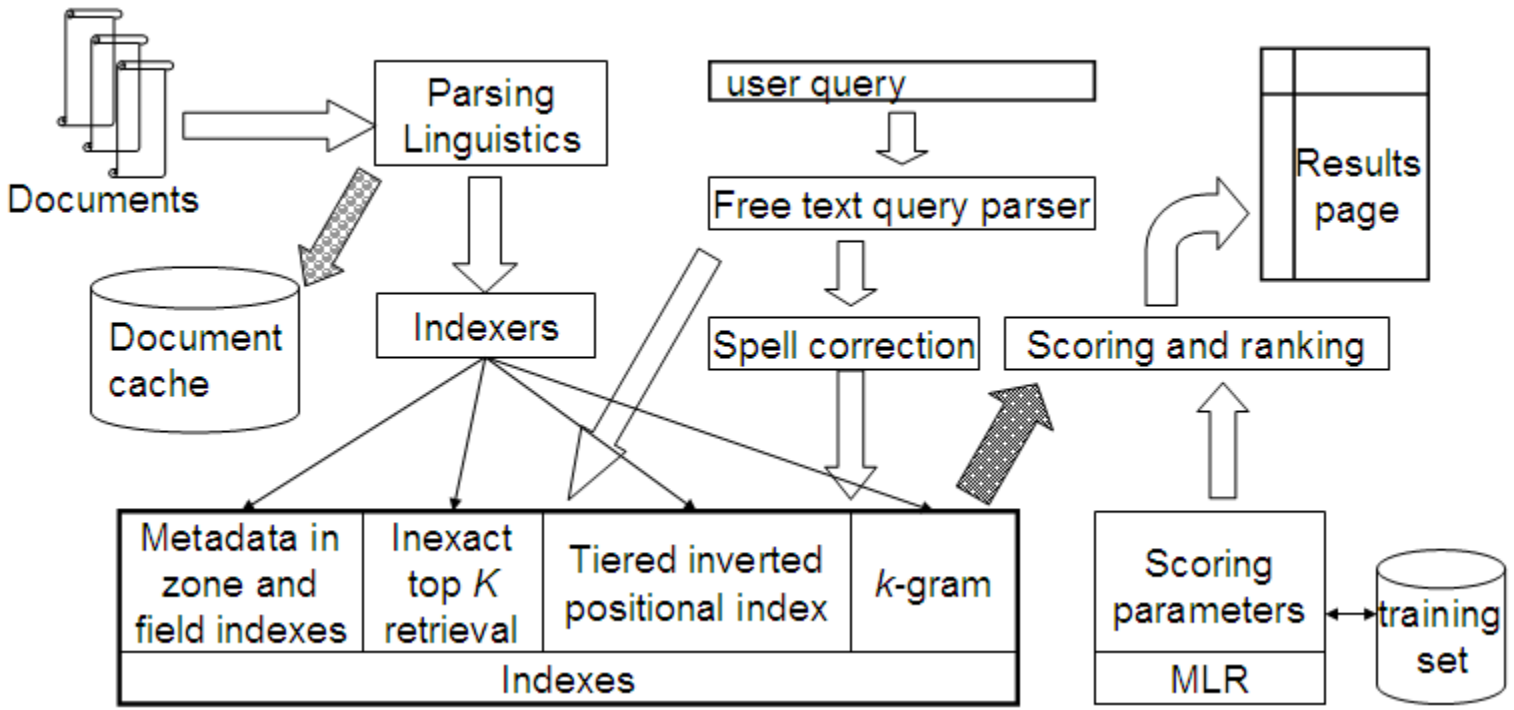
- Ranking: Can we learn how to best order a set of documents, e.g., a set of search results

# Exercise: Build Inverted Index and Run Boolean Retrieval

- Doc1: The winning ticket in Florida was sold at a Publix Supermarket.
- Doc2: The winning numbers were 08, 27, 34, 04 and 19, and the Powerball was 10.
- Doc3: With three winning tickets, the lump sum will be $187.2 million.

- Query1: winning AND ticket
- Query2: winning OR ticket

# What's next …


Documents
Parsing
Linguistics
Document
cache
Indexers
user query
Free text query parser
Spell correction
Scoring and ranking
Results
page
Metadata in
zone and
field indexes
Inexact
top K
retrieval
Tiered inverted
positional index
k-gram
Indexes
Scoring
parameters
MLR
training
set

# Any Questions?