بسم الله الرحمن الرحيم

دورة "استرجاع المعلومات" باللغة العربية - صيف ٢٠٢١

Information Retrieval – Summer 2021

2. Indexing & Preprocessing

read

Tamer Elsayed
Qatar University

Feedback on Yesterday’s Lecture

- Mostly very positive!
  - detailed explanation, simplification
  - content well-organized, logical sequence, etc.
  - polls

- Speed, repetition
- Lab
- More material
- Taking questions

4

4

Google supports Boolean retrieval.

- Yes
- No

Boolean retrieval is called "exact-match" because ...

- it returns documents that exactly satisfy the Boolean query.
- it returns documents that exactly satisfy the information need.
- it divides the collection into exactly two subsets of documents.

When we change our query after seeing the search results, ...

- we are actually changing our information need.
- we are representing the same information need but in a different way.
- either of the above cases can happen.

Today’s Roadmap

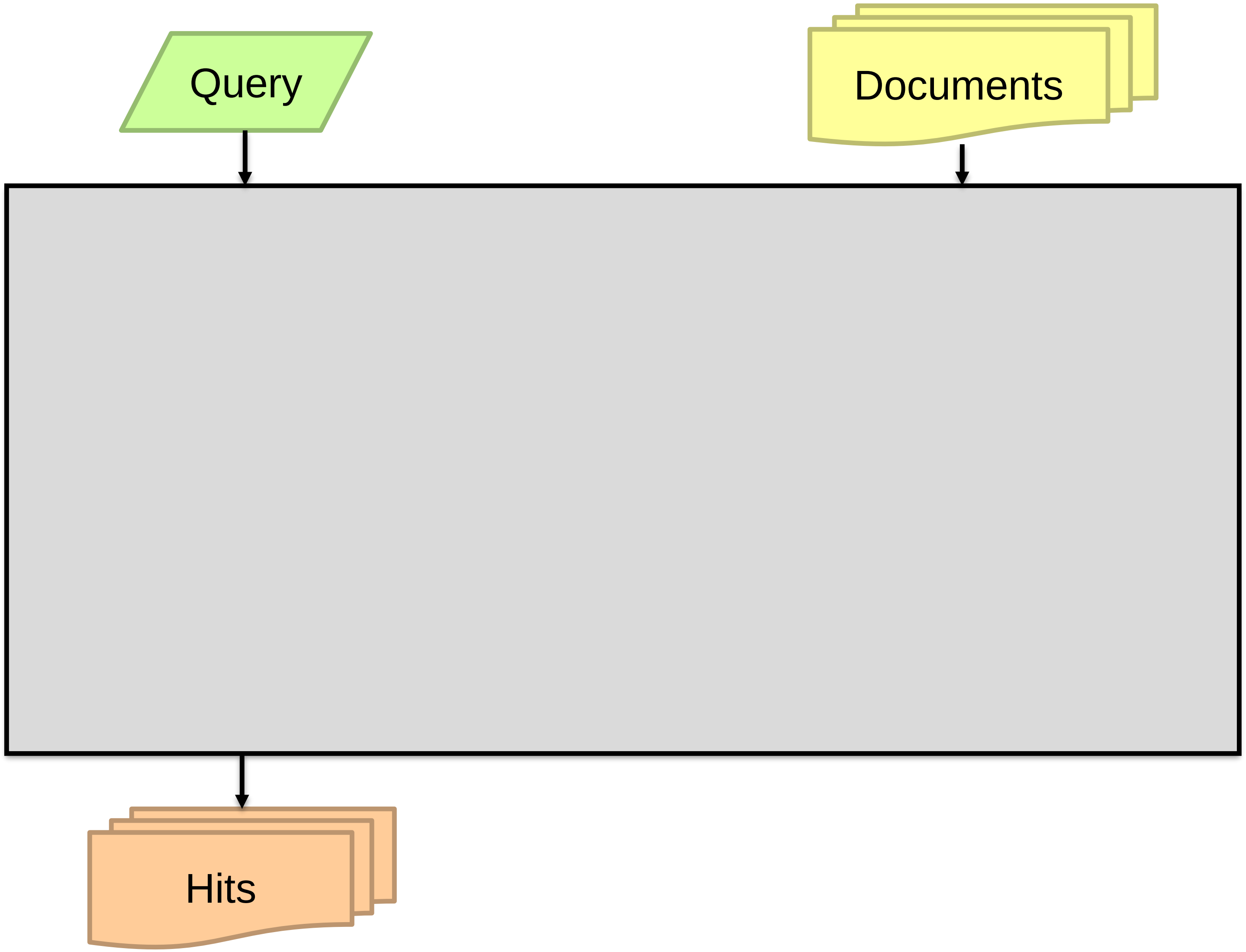
- The anatomy of a search engine

- Indexing

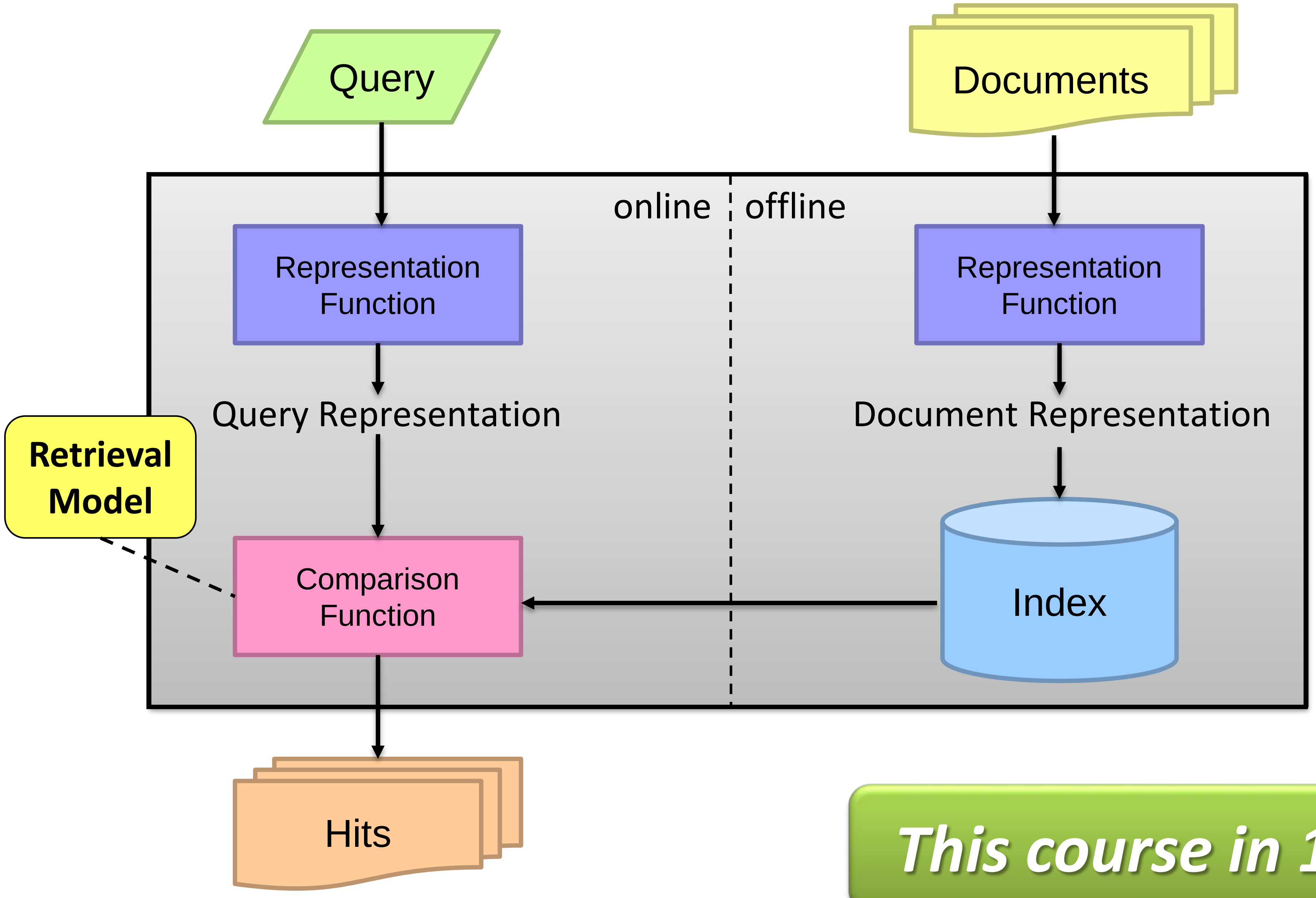
- Preprocessing

THE ANATOMY OF A SEARCH ENGINE

The IR Black Box

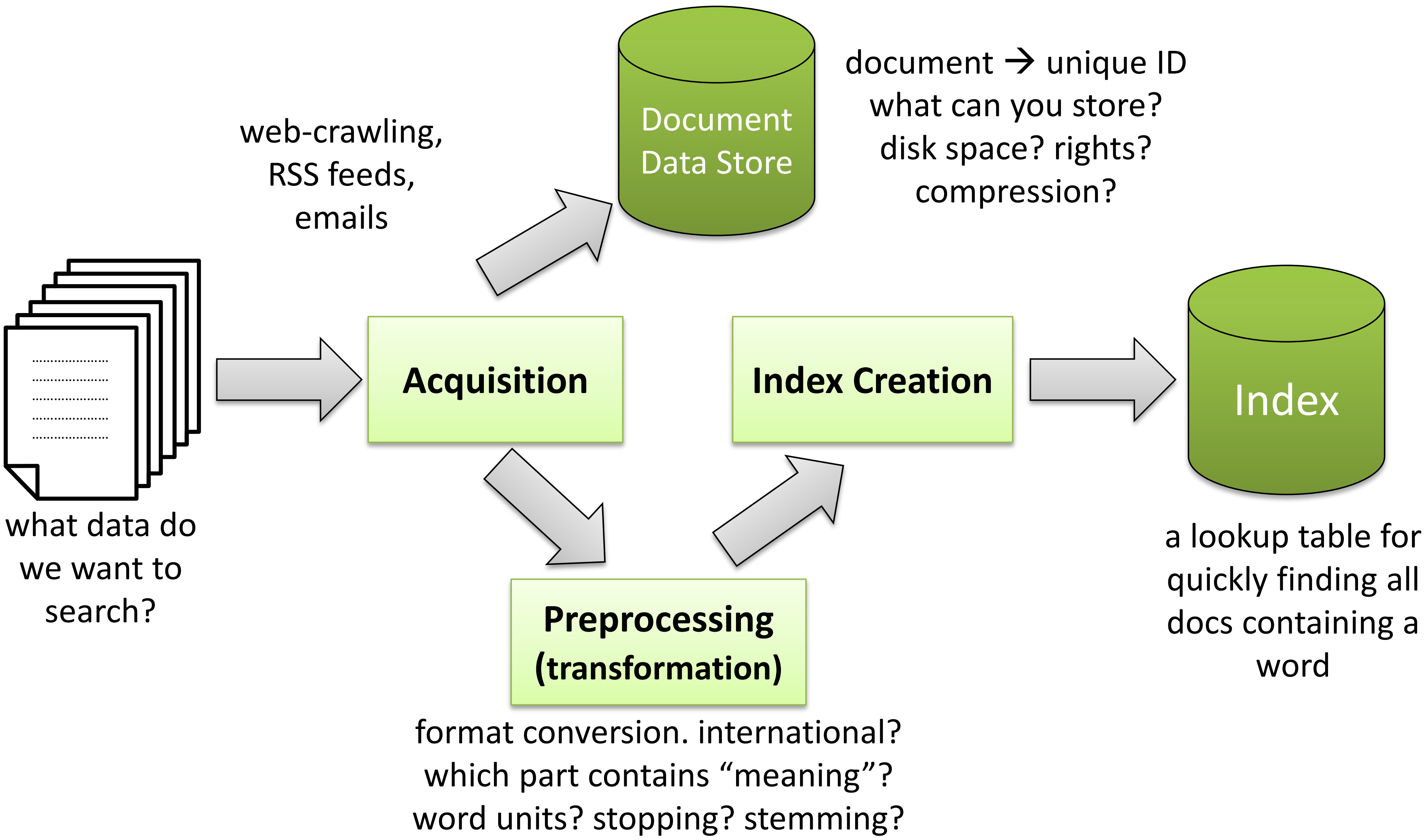


Inside the IR Black Box

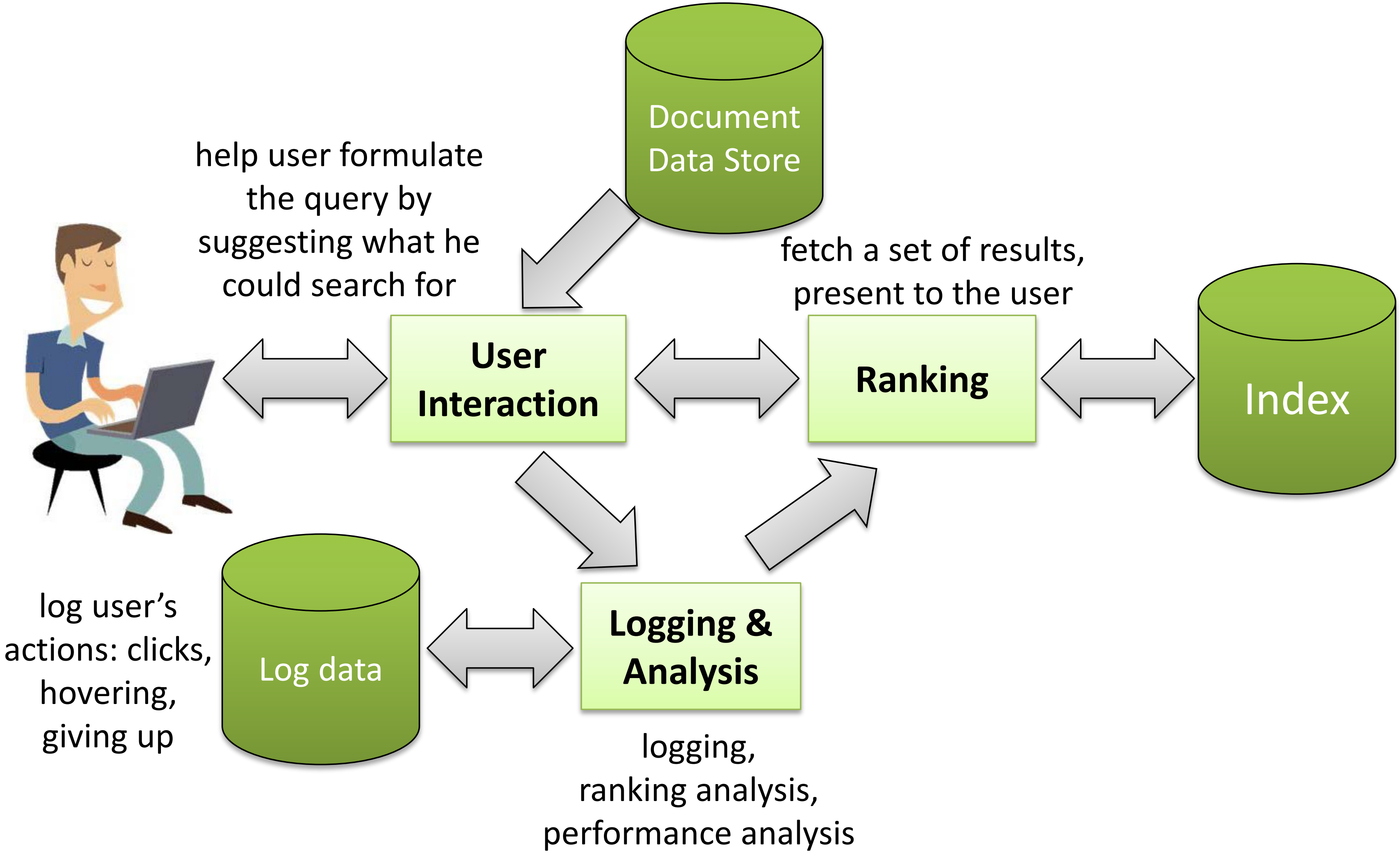


This course in 1 slide!

Indexing process (offline)



Search process (online)




5
P
O
L
L

5

Indexing is done at query time only.

- Yes
- No, it is done only offline
- No, it is done both offline and online

Ranking is done ...

- offline
- online
- both offline and online

(SIMPLE) INDEXING

Bigger Collections …

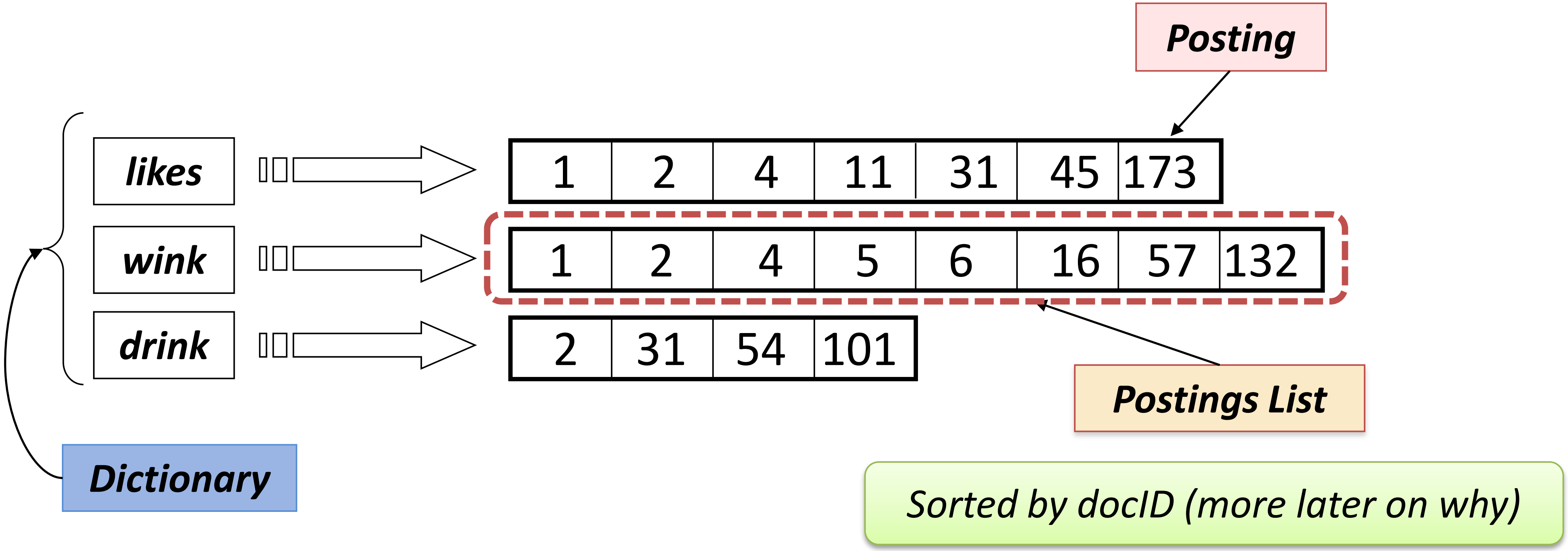
- Consider N = 1 million documents, each with about 1000 words.
- Say there are M = 500K *distinct* terms among these.
- 500K x 1M term-doc incidence matrix has half-a-trillion 0's and 1's.

- But it has no more than one billion 1's. **?**
  - matrix is extremely sparse.

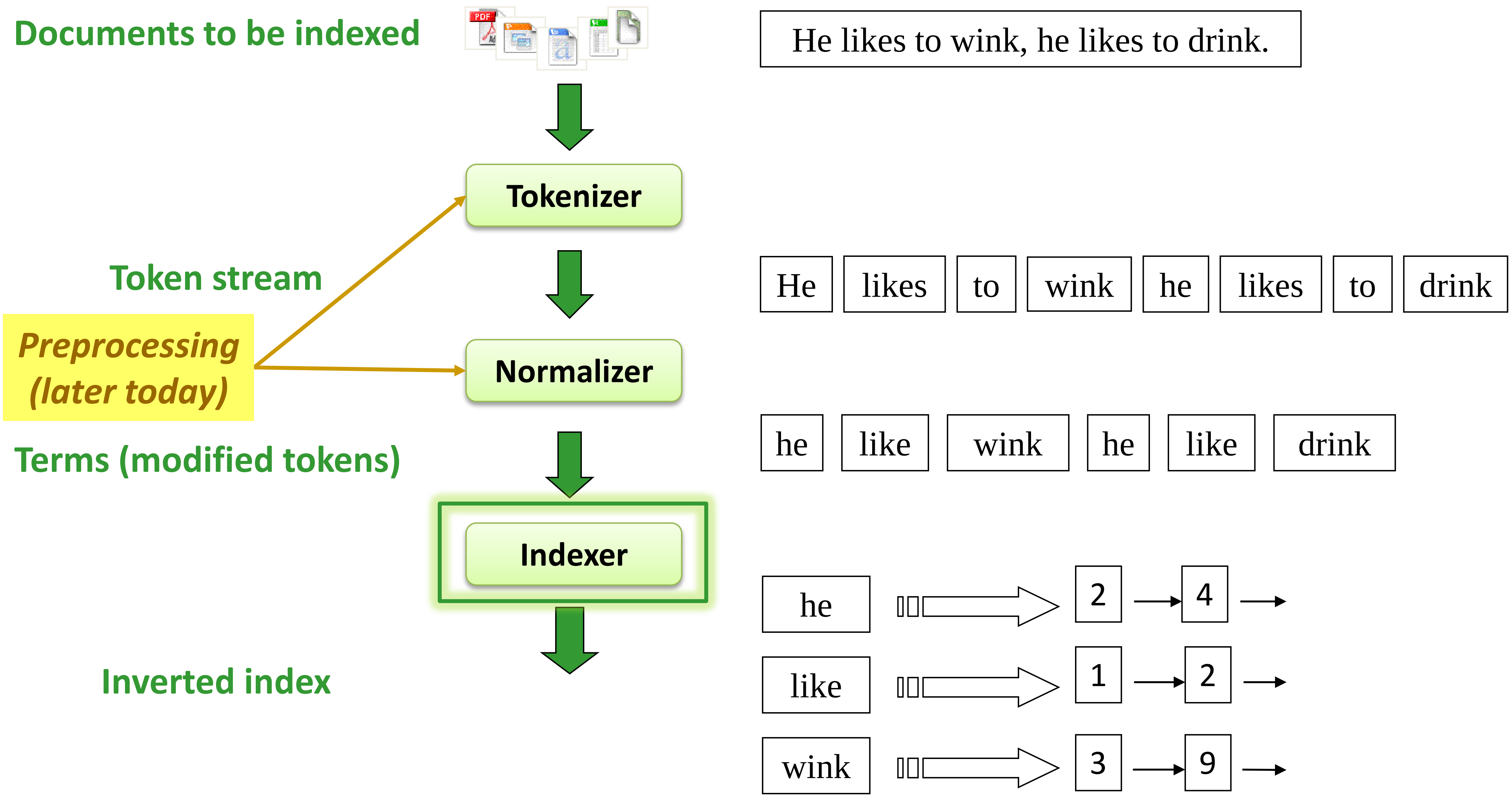
What's a better representation?

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



Inverted Index Construction



Step 1: Term Sequence

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

Preprocess

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

Sequence of (term, Doc ID) pairs

Step 2: Sorting

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

Preprocess

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

Sequence of (term, Doc ID) pairs

Core indexing step

Sort by term then DocID

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

Sorted Sequence of (term, Doc ID) pairs

Step 3: Dictionary & Postings

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

Preprocess

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

Sequence of (term, Doc ID) pairs

Core indexing step

Sort by term then DocID

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

Sorted Sequence of (term, Doc ID) pairs

Dictionary & Postings

***df* information is added**

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

Inverted Index

Indexing

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

Preprocess

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
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Sequence of (term, Doc ID) pairs

Sort by term <u>then</u> DocID

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
the	1
the	
told	
you	2
was	1
was	2
with	2

Sorted Sequence of (term, Doc ID) pairs

Dictionary & Postings

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

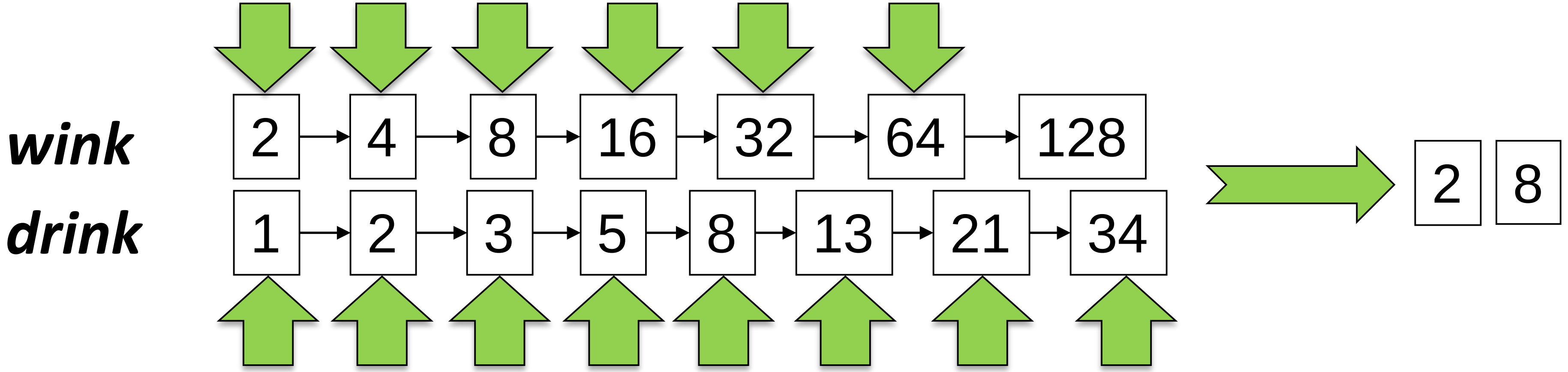
Inverted Index

How do we index efficiently?

In IR2 course ☺

Query Processing: AND

- Consider processing the query: ***wink*** *AND* ***drink***
  1. Locate ***wink*** in the Dictionary, Retrieve its postings
  2. Locate ***drink*** in the Dictionary, Retrieve its postings
  3. "Merge" the two postings lists



- Complexity ?
- <u>Crucial</u>: postings sorted by docID.

Intersecting Two Postings Lists: (a “merge” algorithm)

```
INTERSECT(p1, p2)
answer ← ⟨ ⟩
while p1 ≠ NIL and p2 ≠ NIL
do if docID(p1) = docID(p2)
      then ADD(answer, docID(p1))
           p1 ← next(p1)
           p2 ← next(p2)
      else if docID(p1) < docID(p2)
              then p1 ← next(p1)
              else p2 ← next(p2)
return answer
```

Document-at-a-time

How to modify for OR?

Boolean Queries: More General Merges

- Adapt the merge for the queries:

  wink *AND NOT* ***drink***

  wink *OR NOT* ***drink***

- What about an arbitrary Boolean formula?

  (wink *OR* ***drink)*** *AND* ***(like*** *OR NOT* ***ink)***

6

6

In inverted index, we can get efficiently ...

- ➢ what terms appear in a specific document
- ➢ what documents have a specific term
- ➢ both of the above

One posting belongs to ...

- ➢ one term
- ➢ one document
- ➢ one term in one document

PROXIMITY QUERIES

Proximity Queries

- If 2 words are "near" each other in a document ***d***, they might be more related than further words ➔ ***d*** might be "*more relevant*"

- Ex: Find ***Gates*** *NEAR/3* ***Microsoft***.

How can we support it?

Positional Indexes

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

<***likes***: 9347;

1: 7, 18, 33, 72, 86, 231;

2: 3, 149;

4: 17, 191, 291, 430, 434;

5: 363, 367, …>

What's the biggest problem?

Positional Index Size

- You can compress position values/offsets
- Nevertheless, a positional index **expands postings storage *substantially***
- Nevertheless, a positional index is **now standardly used** because of the power and usefulness of phrase and proximity queries … whether used explicitly or implicitly in a ranking retrieval system.

Phrase Queries

- Want to be able to answer queries such as “***qatar university”*** – as a phrase
- Thus the sentence *“I went to university in Qatar”* 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*

How can we support it using positional index?

7

7

Phrase queries are special case of proximity queries

- Yes
- No

Proximity queries are .... Boolean queries

- more expensive than
- less expensive then
- of equal cost to

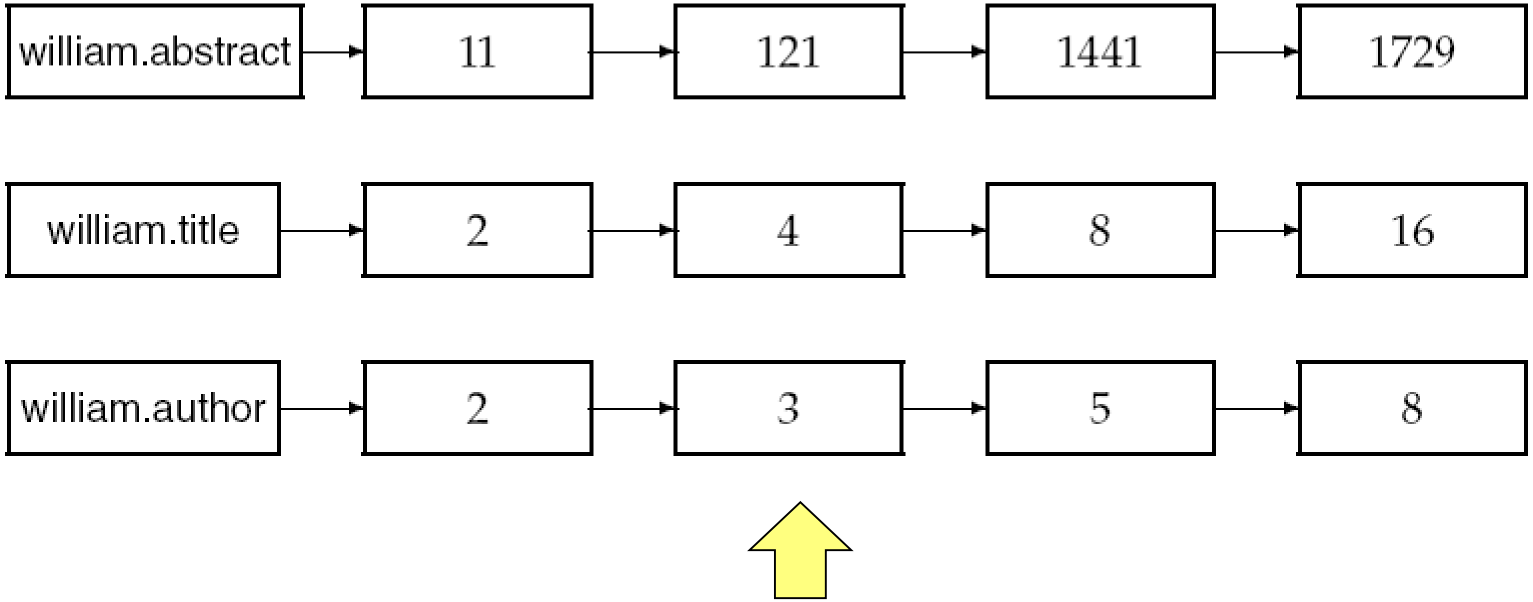
ZONES

Zone

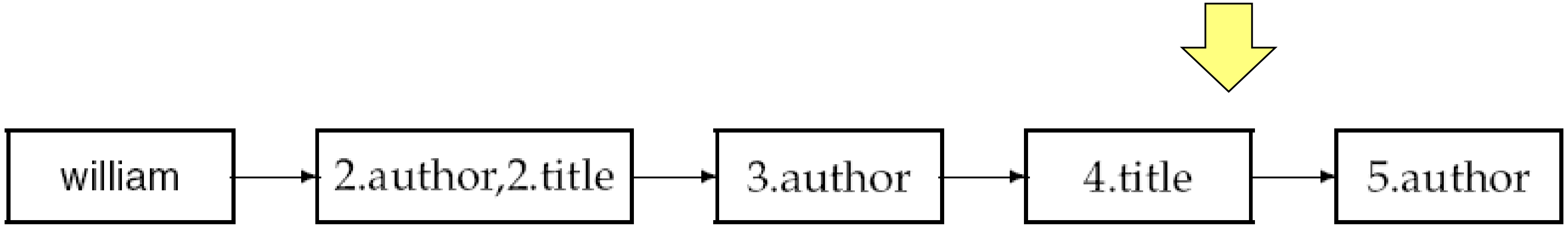
- A zone is a region of the doc that can contain an arbitrary amount of **text** e.g.,
  - Title
  - Abstract
  - References …

- Build inverted indexes on zones as well to permit querying
  - e.g., find docs with *merchant* in the title zone and “*gentle rain” in the* body.

Example Zone Indexes



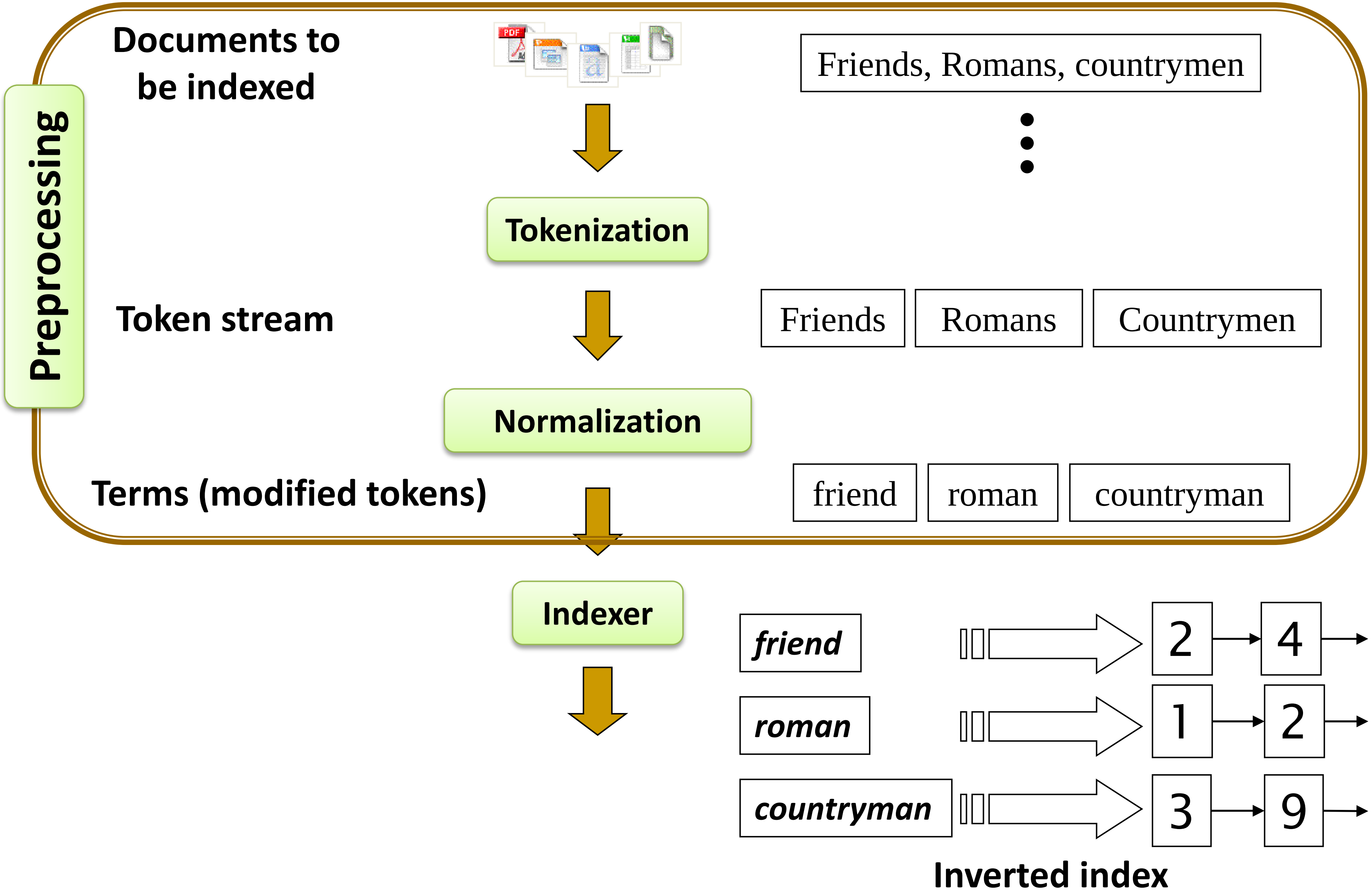
Encode zones in dictionary vs. postings.



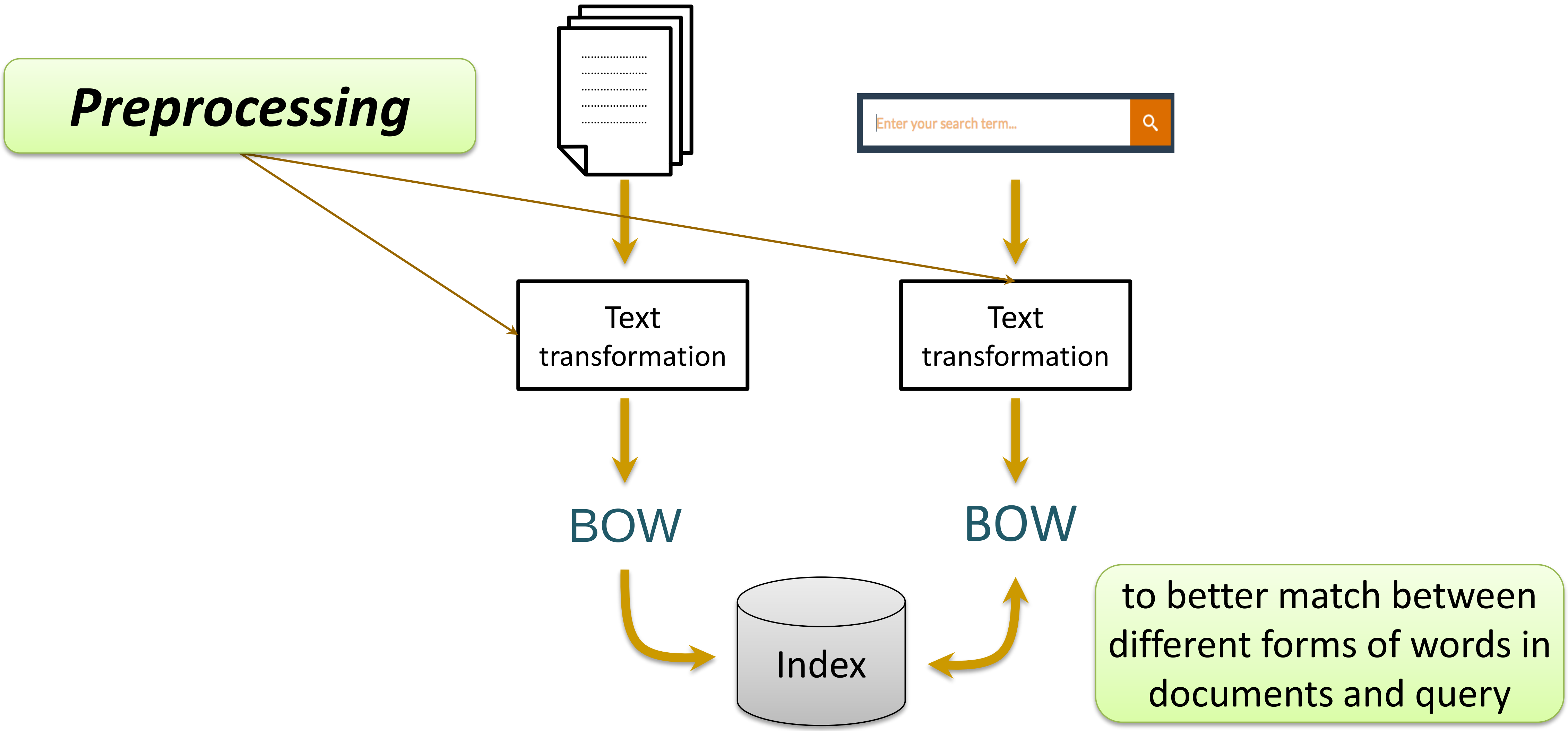
Today's Roadmap

- The anatomy of a search engine
- Indexing
- Preprocessing

The Basic Indexing Pipeline



Preprocessing



Preprocessing Steps

1. Tokenization
2. Stopping
3. Stemming

***Objective**: identify the optimal form of the term to be indexed to achieve the best retrieval performance.*

Before Tokenization …

- **Encoding & Parsing a Document**
  - Which encoding/character set?
  - What format? pdf/word/excel/html?
  - What language?
  - Each is classification problem
  - BUT often done heuristically, by user selection, or by metadata

Byte sequence → Character sequence

- **What is a Unit Document?**
  - A file? An email? A group of files (PPT)?
    A book (a chapter/paragraph/sentence)?
  - Understand collection, user, and usage patterns

Where to Stop?

TOKENIZATION

Tokenization

- Sentence → tokenization (splitting) → tokens
- Input: "***Friends, Romans and Countrymen***"
- Output: Tokens
  - ***Friends***
  - ***Romans***
  - ***and***
  - ***Countrymen***

But what are valid tokens to emit?

- A **token** is an **instance** of a sequence of characters
- Each such token is now a candidate for an index entry (**term**), after further processing.

Issues in Tokenization

- ***Finland's capital*** $\rightarrow$ ***Finland? Finlands? Finland's***?
- ***Hewlett-Packard*** $\rightarrow$ one token or two?
  - ***state-of-the-art***: break up hyphenated sequence.
  - ***co-education***
  - ***lowercase***, ***lower-case***, ***lower case*** ?
  - It can be effective to get the user to put in possible hyphens
- ***San Francisco***: one token or two?
  - How do you decide it is one token?
- **Numbers?**
  - *3/20/91 Mar. 12, 1991 20/3/91*
  - This course code is CMPT621
  - *(800) 234-2333*

Issues in Tokenization

- **URLs:**
  - *http://www.bbc.co.uk*
  - *http://www.bbc.co.uk/news/world-europe-41376577*
- **Social Media**
  - *Black lives matter*
  - *#Black_lives_matter*
  - *#BlackLivesMatter*
  - *#blacklivesmatter*
  - *@blacklivesmatter*

Language-dependent Issues

- French
  - ***L'ensemble*** → one token or two?
    - ***L*** ? ***L'*** ? ***Le*** ?
    - Want ***l'ensemble*** to match with ***un ensemble***
      - Until at least 2003, it didn't on Google
- German noun compounds are not segmented
  - ***Lebensversicherungsgesellschaftsangestellter***
  - 'life insurance company employee'
  - German retrieval systems benefit greatly from a **compound splitter** module
    - Can give a 15% performance boost for German
- Chinese and Japanese have no spaces between words:
  - 莎拉波娃现在居住在美国东南部的佛罗里达。
  - Tokenization → Segmentation

Tokenization: common practice

- Just split at non-letter characters
- Add special cases if required
- Some applications have special setup
  - Social media: hashtags/mentions handled differently
  - URLs: no split, split at domain only, remove entirely!
  - Medical: proteins & diseases names

STOP WORD REMOVAL

Stopping (stop words removal)

- ~~This is a very~~ exciting lecture ~~on the~~ technologies ~~of~~ text
- **Stop words**: the most common words in collection
  → the, a, is, he, she, I, him, for, on, to, very, …
- They have little semantic contribution
- They appear a lot ≈ 30-40% of text
- New stop words appear in specific domains
  - e.g., "RT" in Tweets: *"RT @realDonalTrump Mexico will …"*
- Stop words
  - influence on sentence structure
  - less influence on topic (aboutness)

Stopping: always apply?

- Sometimes very important:
  - Phrase queries: “Let it be”, “To be or not to be”
  - Relational queries:
    - flights to Doha from London
    - flights from Doha to London
- In Web search, trend is to keep them:
  - Good compression techniques means the space for including stop words in a system is small.
  - Good query optimization techniques mean you pay little at query time for including stop words.

Stopping: common practice

- Common practice in many applications
  → remove stop words
- There are common stop words list for each language
  - NLTK (Python)
  - Lucene (Java)
  - http://members.unine.ch/jacques.savoy/clef/index.html
- There are special stop words list for some applications

How to create your own list?

8

8

Can tokenization affect retrieval effectiveness?

- Yes
- No

Stop words should usually have very high document frequency

- Yes
- No

NORMALIZATION

Normalization

- **Objective** → make words with different surface forms look the same
- Document: "there are few CARS!!"
  Query: "car"
  should "car" match "CARS"?
- Sentence → tokenization → **tokens** → normalization → **terms** to be indexed (vocabulary/dictionary).

Case Folding

- "A" & "a" are different strings for computers
- Case folding: convert all letters to lower case

- CAR, Car, caR → car
- Windows → windows
  - should we do that?
  - Usually yes, users are so lazy
- Upper case in mid-sentence?
  - I bought it from ***General Motors***
  - ***Black*** vs. ***black***

Thesauri and Soundex

- Do we handle synonyms?
  - e.g., by hand-constructed equivalence classes
    - ***car*** = ***automobile*** ***color*** = ***colour***
  - We can rewrite to form equivalence-class terms
    - When the document contains ***automobile***, index it under ***car-automobile*** (and vice-versa)
  - Or we can expand a query
    - When the query contains ***automobile***, look under ***car*** as well

- What about spelling mistakes?
  - One approach is soundex, which forms equivalence classes of words based on phonetic heuristics

Lemmatization

- Lemmatization implies doing “proper” reduction to the “base” or dictionary form, called **lemma**.
  - Morphological analysis

- Reduce inflectional/variant forms to base form
- e.g.,
  - *am, are, is* $\rightarrow$ *be*
  - *saw* $\rightarrow$ see
  - *car, cars, car's, cars'* $\rightarrow$ *car*

Stemming

- Search for: “play”
  should it match: “plays”, “played”, “playing”, “player”?
- Many morphological variations of words
  - *inflectional* (plurals, tenses)
  - *derivational* (making verbs nouns, etc.)
- In most cases, aboutness does not change.
- Stemmers attempt to reduce morphological variations of words to a common **stem.**

Stemming

- "Stemming" suggests crude affix chopping
  - language dependent
  - e.g., ***automate, automates, automatic, automation*** all reduced to ***automat***.

for example compressed and compression are both accepted as equivalent to compress.

for exampl compress and compress ar both accept as equival to compress

Porter Stemmer

- Most common algorithm for stemming English
- Conventions + 5 phases of reductions
  - phases applied sequentially
  - each phase consists of a set of commands
- Example convention: *Of the rules in a compound command, select the one that applies to the longest suffix.*
- Example rules
  - *sses* $\rightarrow$ *ss* (processes $\rightarrow$ process)
  - *y* $\rightarrow$ *i* (reply $\rightarrow$ repli)
  - *ies* $\rightarrow$ *i* (replies $\rightarrow$ repli)
  - *tional* $\rightarrow$ *tion* (international $\rightarrow$ internation)
  - *(m>1)ement* $\rightarrow$ (replacement $\rightarrow$ replac), (cement $\rightarrow$ cement)

Stemming: is it really useful?

- Usually, it achieves 5-10% improvement in retrieval effectiveness, e.g. English.
- For highly inflected languages, it is more critical:
  - 30% improvement in Finnish IR
  - 50% improvement in Arabic IR

They are Ahmad’s **children**	هؤلاء **أبناء** أحمد
The **children** behaved well	**الأبناء** تصرفوا جيدا
Her **children** are cute	**أبناءها** لطاف
My **children** are funny	**أبنائي** ظرفاء
We have to save our **children**	علينا أن نحمي **أبناءنا**
Patents and **children** are happy	الآباء **والأبناء** سعداء
He loves his **children**	هو يحب **أبناءه**
His **children** loves him	**أبناؤه** يحبونه

Stemmed words are misspelled ?!

- repli, replac, suppli, inform retriev, anim
- These are not **words** anymore, these are **terms**.
- These terms are not seen by the user, but just used by the IR system (search engine).
- These represent the optimal form for a better match between different surface forms of a word.
  - e.g. replace, replaces, replaced, replacing, replacer, replacers, replacement, replacements → replac.

9

9

Same tokenization/normalization steps should be applied to documents and queries.

- Yes, always!
- No, they can be different of course

The dictionary in the index includes ...

- words
- tokens
- terms
- all of the above


SEARCH
INFORMATION
RETRIEVAL
GOOGLE
CORPUS
QUERY
RANK
DATA
DOCUMENTS
DIFFICULT
CONFUSING
MATHEMATICAL
WEB
INDEX
CRAWL
USEFUL
DATABASE
DOCUMENT
METADATA
RETRIEVE
LARGE
IMPORTANT
COOL
AMAZING
COMPLEX
EFFICIENT
SLOW
SCIENCE
PAGE
DISTRIBUTED
GATHER
SEARCHING
CRAWLER
FAST
FUN
LIST
TEXT

OVERALL

Preprocessing: common practice

- Tokenization: split at non-letter characters
  - For tweets, you might want to keep “#” and “@”.
- Remove stop words
  - find a common list, and filter these words out.
- Apply case folding
- Apply Porter stemmer (or others for other languages)
  - Other stemmers are available, but Porter is the most famous with many implementations available in different programming languages.

Summary

- Pre-processing:
  - Tokenization → Stopping → Stemming

This is an **exampl**e **sentenc**e of how the **pre-process**ing is **appli**ed to **text** in **inform**ation **retriev**al. It **includ**es: **Token**ization, **Stop Word**s **Remov**al, and **Stem**ming

↓

exampl sentenc pre process appli text inform retriev includ token stop word remov stem

***How can we know
if a search engine is "good" or "bad"?***