Lecture 2: The term vocabulary and postings lists

Algorithm

Intersect($p1$, $p2$)
1 *answer* ←
2 **while** $p1$ = nil and $p2$ = nil
3 **do if** *doc ID*($p1$) = *doc ID*($p2$)
4 **then** Add(*answer*, *doc ID*($p1$))
5 $p1$ ← *next*($p1$)
6 $p2$ ← *next*($p2$)
7 **else if** *doc ID*($p1$) < *doc ID*($p2$)
8 **then** $p1$ ← *next*($p1$)
9 **else** $p2$ ← *next*($p2$)
10 **return** *answer*

Algorithm: for the intersection of two postings lists $p1$ and $p2$.

Recap of the previous lecture

- Basic inverted indexes:
  - Structure: Dictionary and Postings
  - Key step in construction: Sorting
- Boolean query processing
  - Simple optimization
  - Linear time merging
- Overview of course topics

Information Retrieval

Finding **unstructed data**

that satisfies an **information need**

- When we finding word in large untrusted text what we do?????

Example

Q: lawyer

Result

Q: lawyer

CHAPTER I

Mr. Sherlock Holmes

Mr. Sherlock Holmes, who was usually very late in the mornings, save upon those not infrequent occasions when he was up all night, was seated at the breakfast table. I stood upon the hearth-rug and picked up the stick which our visitor had left behind him the night before. It was a fine, thick piece of wood, bulbous-headed, of the sort which is known as a "Penang **lawyer**." Just under the head was a broad silver band nearly an inch across. "To James Mortimer, M.R.C.S., from his friends of the C.C.H.," was engraved upon it, with the date "1884." It was just such a stick as the old-fashioned family practitioner used to carry--dignified, solid, and reassuring.

"Well, Watson, what do you make of it?"

Holmes was sitting with his back to me, and I had given him no sign of my occupation.

"How did you know what I was doing? I believe you have eyes in the

Incidence Matrix

$$\text{contains}_{i,j} = \begin{cases} 1 & \text{if document j contains term i} \\ 0 & \text{if document j does not contain term i} \end{cases}$$

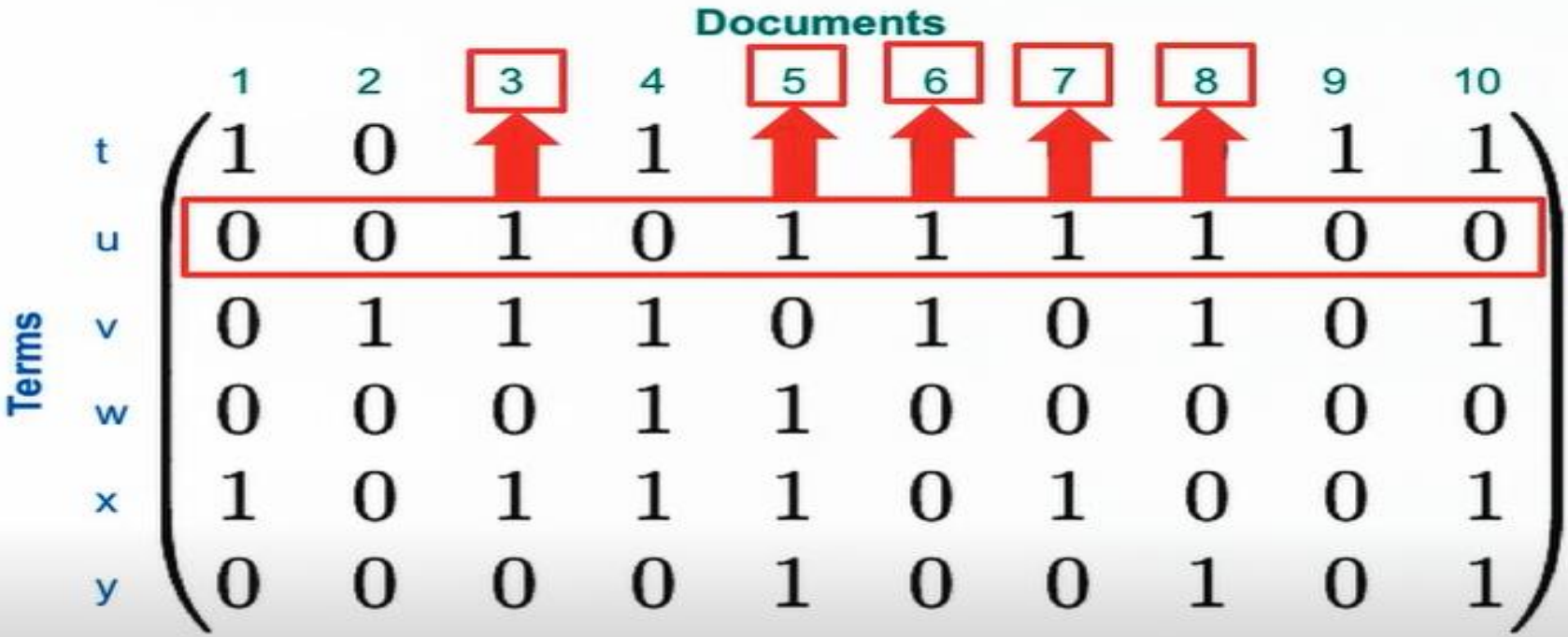
Incidence Matrix

find U

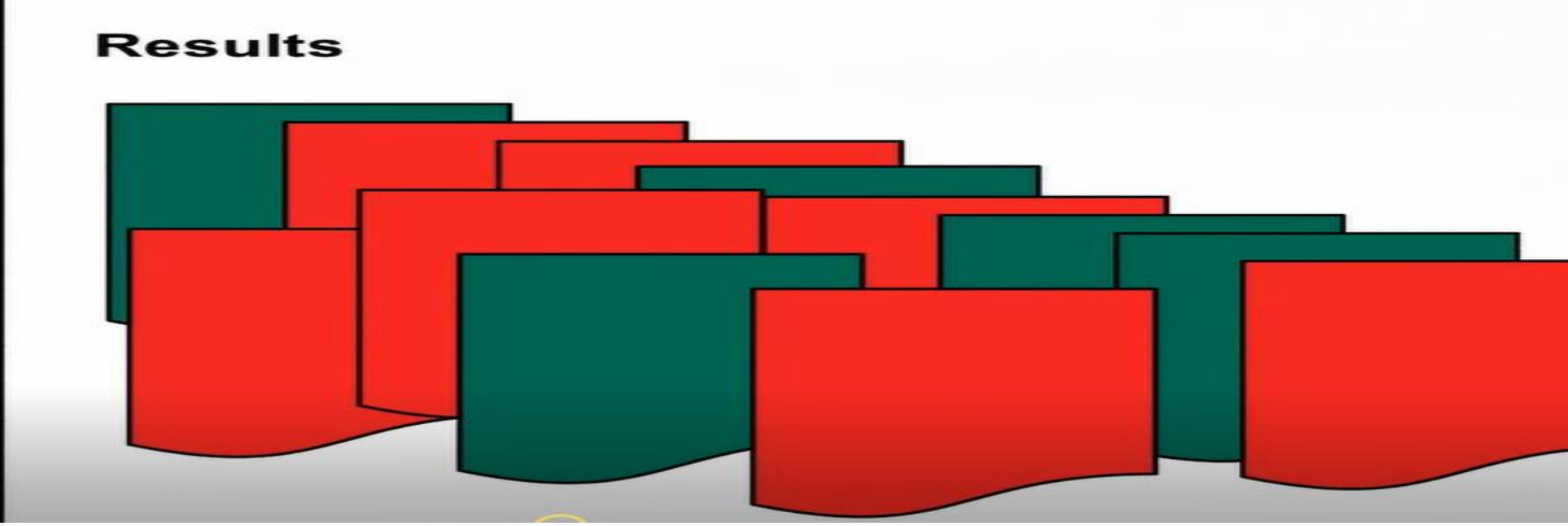
Terms \ Documents	1	2	3	4	5	6	7	8	9	10
t	1	0	1	1	1	0	0	0	1	1
u	0	0	1	0	1	1	1	1	0	0
v	0	1	1	1	0	1	0	1	0	1
w	0	0	0	1	1	0	0	0	0	0
x	1	0	1	1	1	0	1	0	0	1
y	0	0	0	0	1	0	0	1	0	1

$\text{contains}_{y,5} = 1$

Incidence Matrix

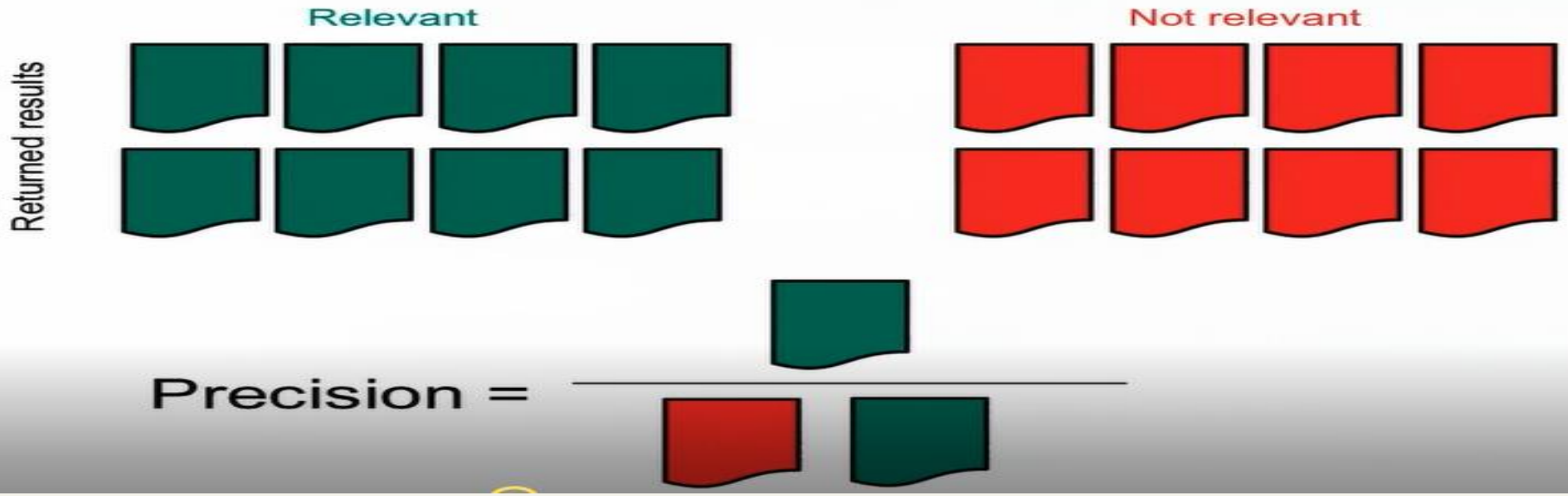


- If you have NOT U?


Results

Results
Relevant
Not relevant
Returned results

Results

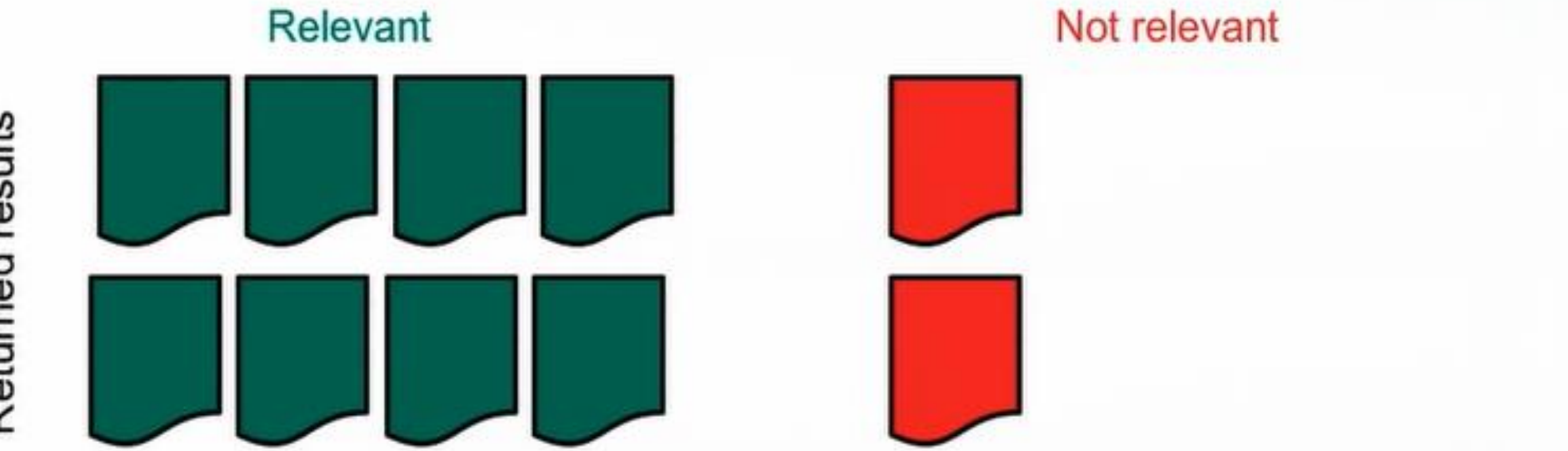


$$\text{Precision} = \frac{\blacksquare}{\blacksquare\ \blacksquare}$$

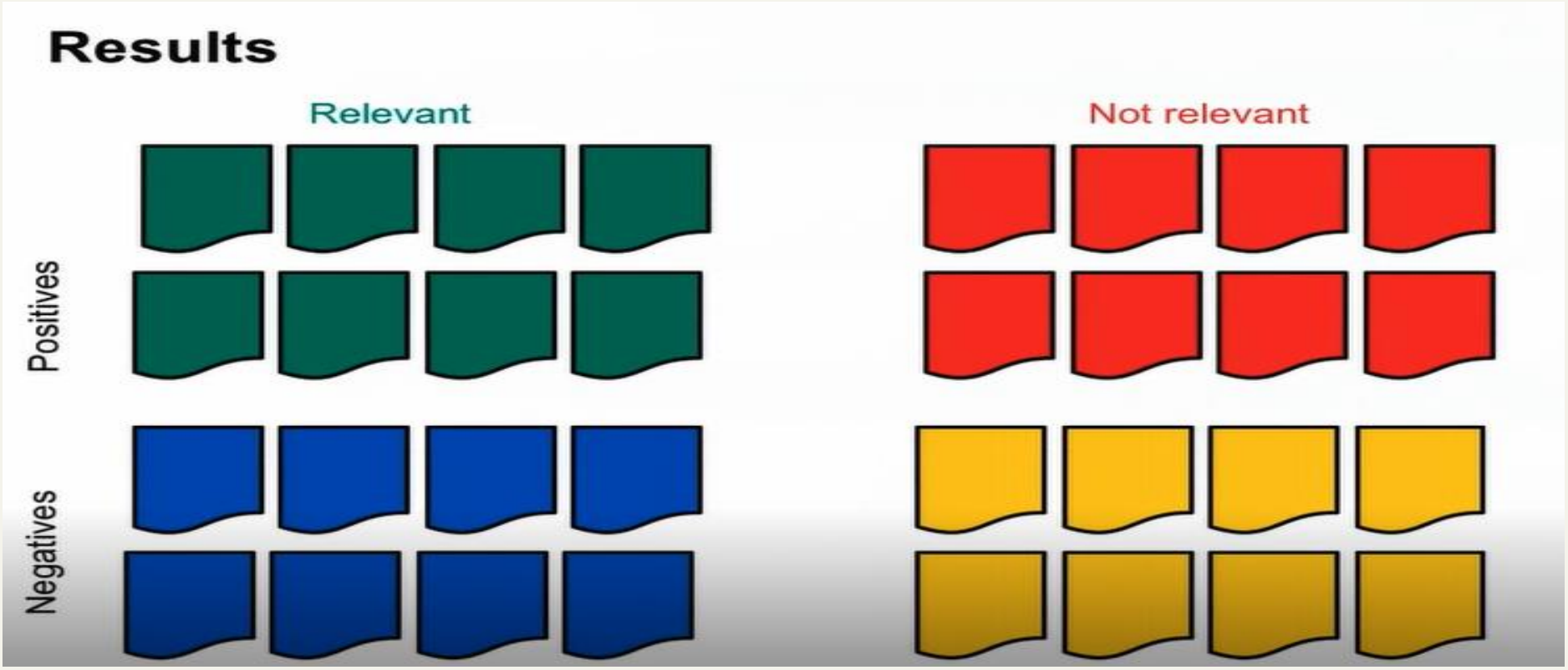
Results

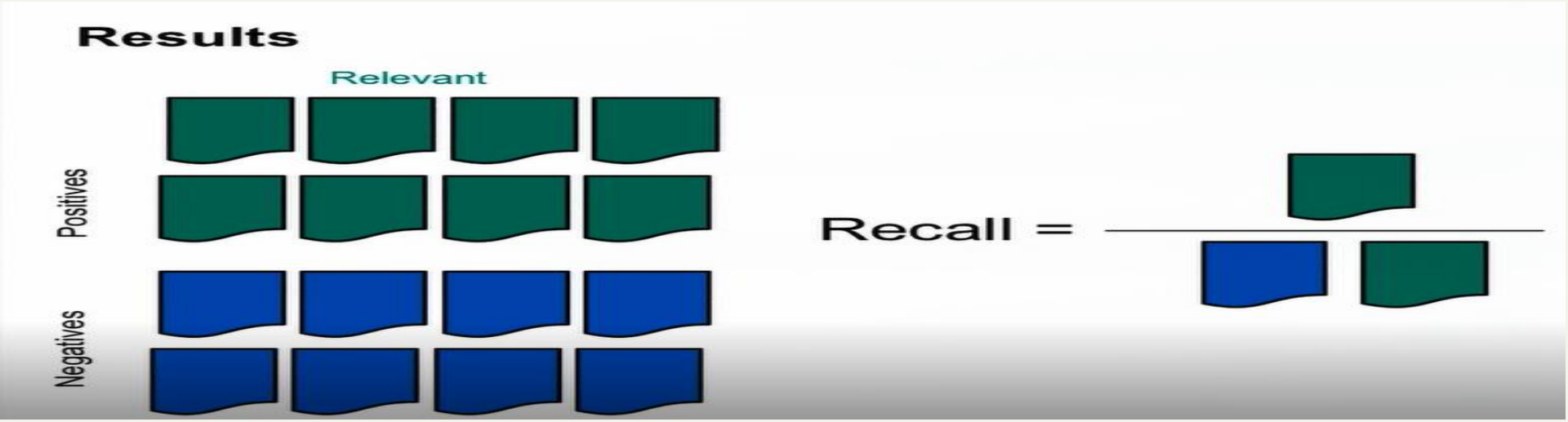
$$\text{Precision} = \frac{\text{true positives}}{\text{returned}}$$

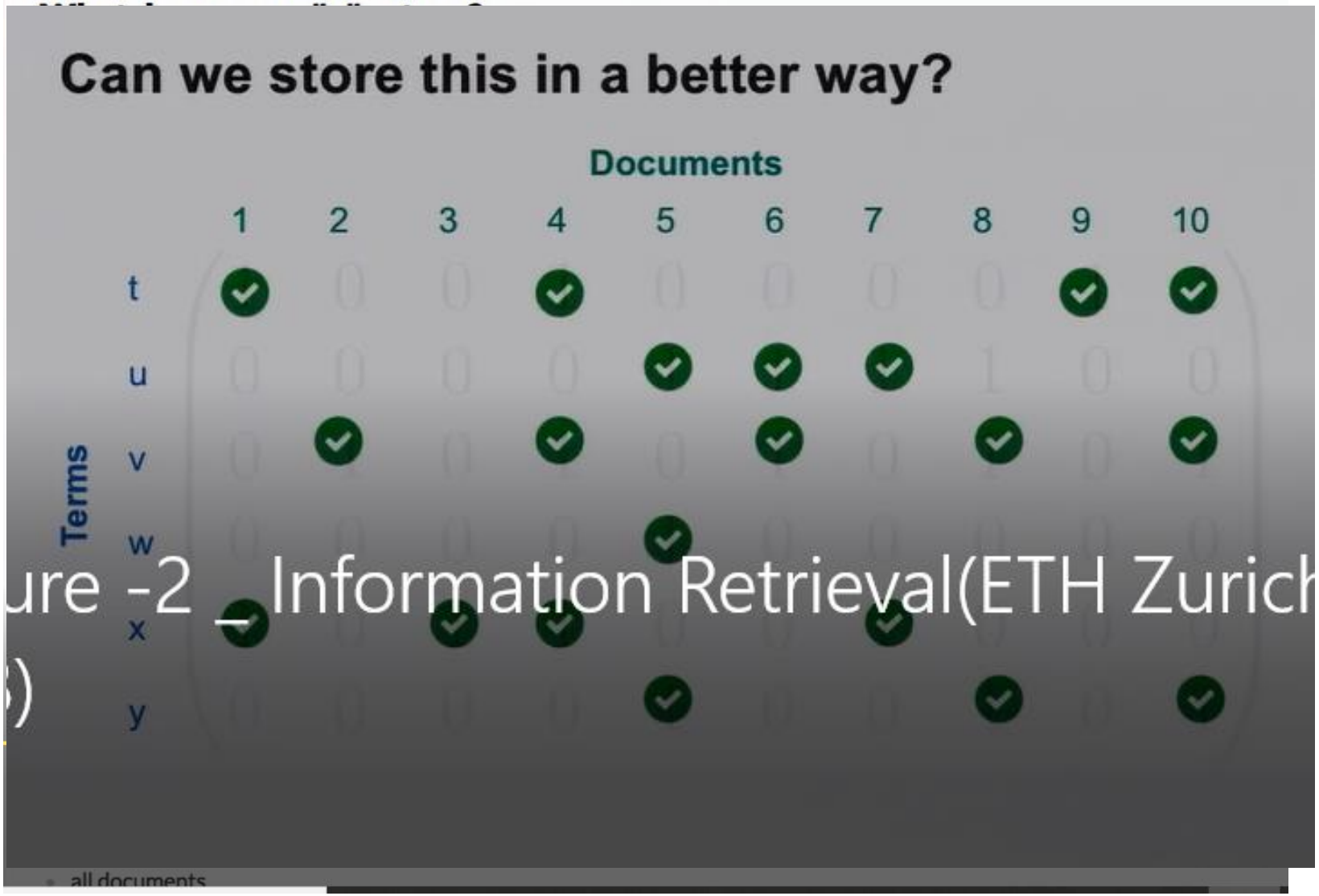
Results



Precision = 80%


Results
Relevant
Not relevant
Positives
Negatives


Results
Relevant
Positives
Negatives
Recall =



ure -2 _ Information Retrieval(ETH Zurich

)

all documents

Tokenize

You come most carefully upon your hour

Take fair thy hour Laertes time be thine

My hour is almost come

Possess it merely That it should come to this

Linguistic pre-processing

you come most carefully upon your hour

take fair thy hour Laertes time be thine

my hour is almost come

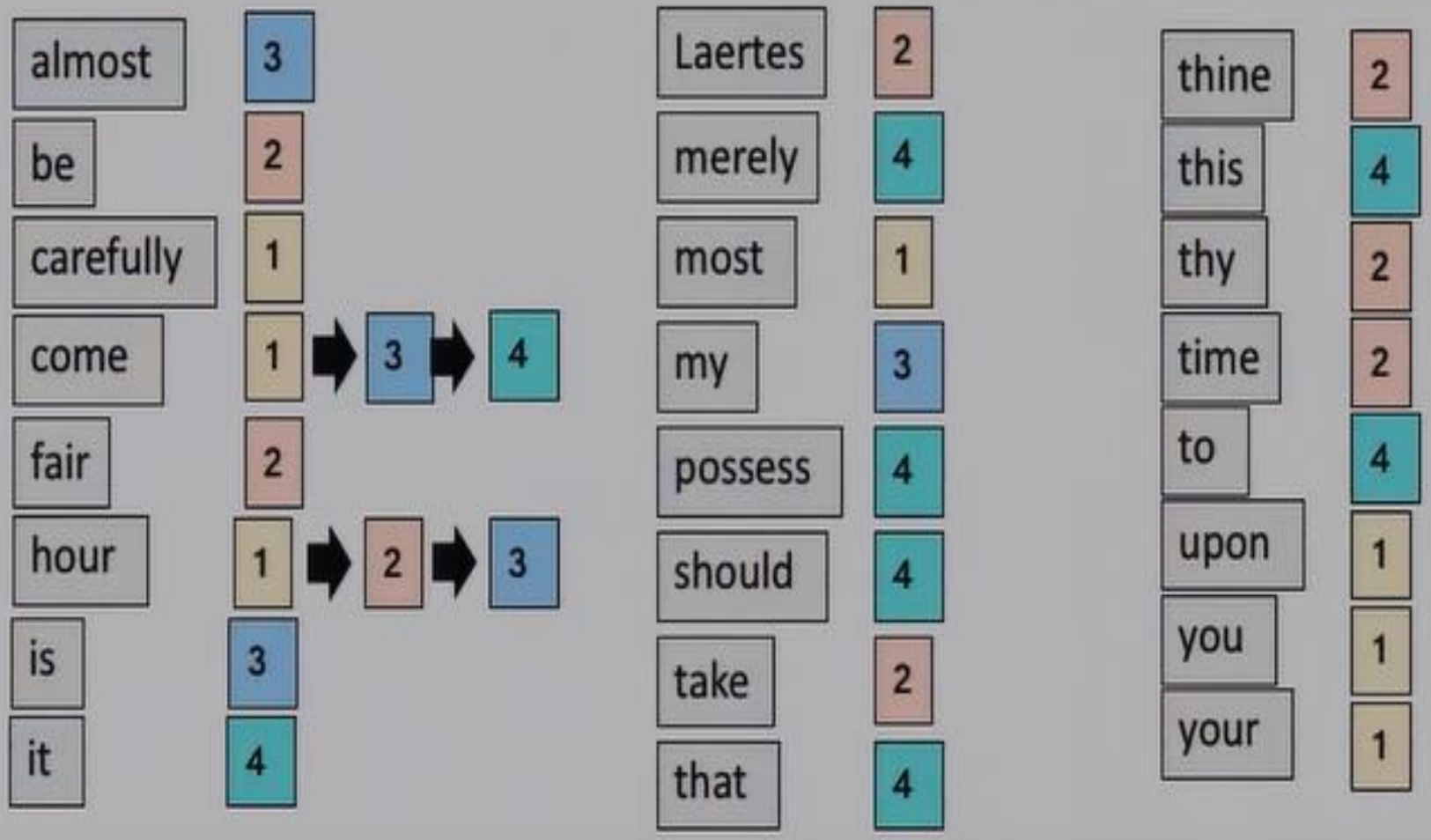
Possess it merely that it should come to this

Assign document IDs

Sort


almost 3
be 2
carefully 1
come 1
come 4
come 3
fair 2
hour 3
hour 1
hour 2
is 3
it 4
it 4
Laertes 2
merely 4
most 1
my 3
possess 4
should 4
take 2
that 4
thine 2
this 4
thy 2
time 2
to 4
upon 1
you 1
your 1

Merge

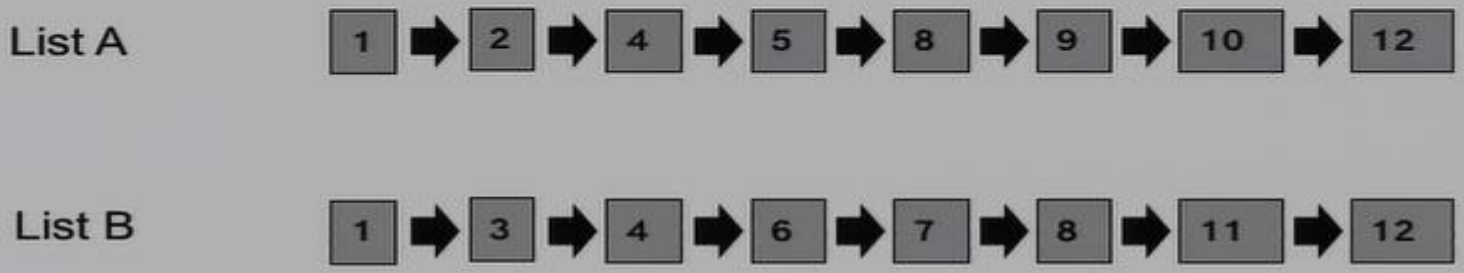

almost
3
be
2
carefully
1
come
1
3
4
fair
2
hour
1
2
3
is
3
it
4
Laertes
2
merely
4
most
1
my
3
possess
4
should
4
take
2
that
4
thine
2
this
4
thy
2
time
2
to
4
upon
1
you
1
your
1

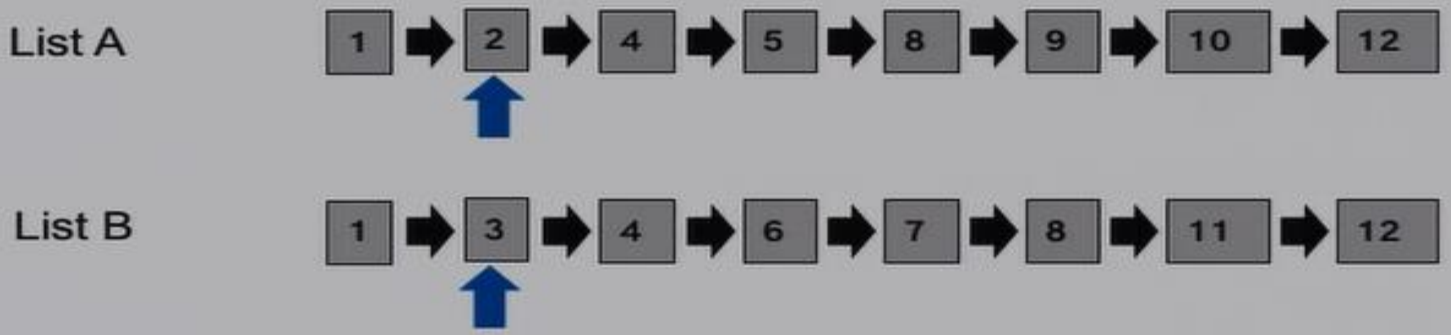
Add document frequency

Single word query

Single word query

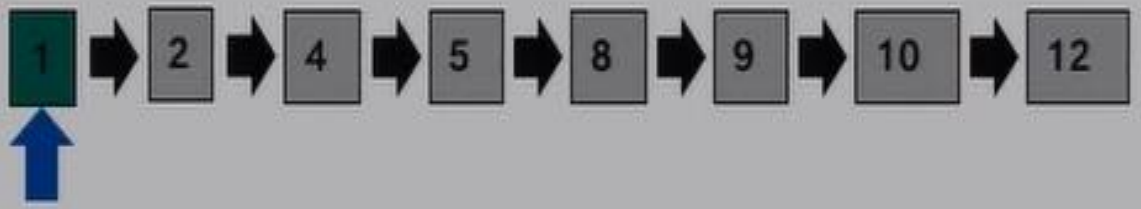
Intersection algorithm





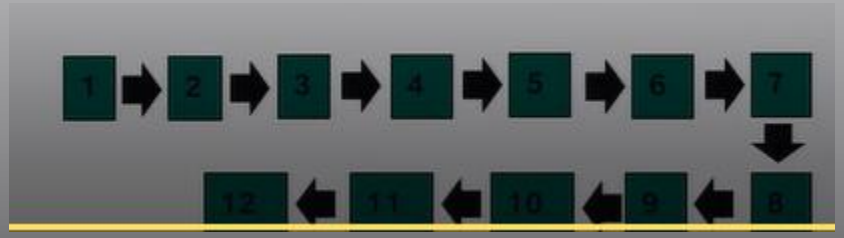
Union algorithm

List A



List B

Union of A and B

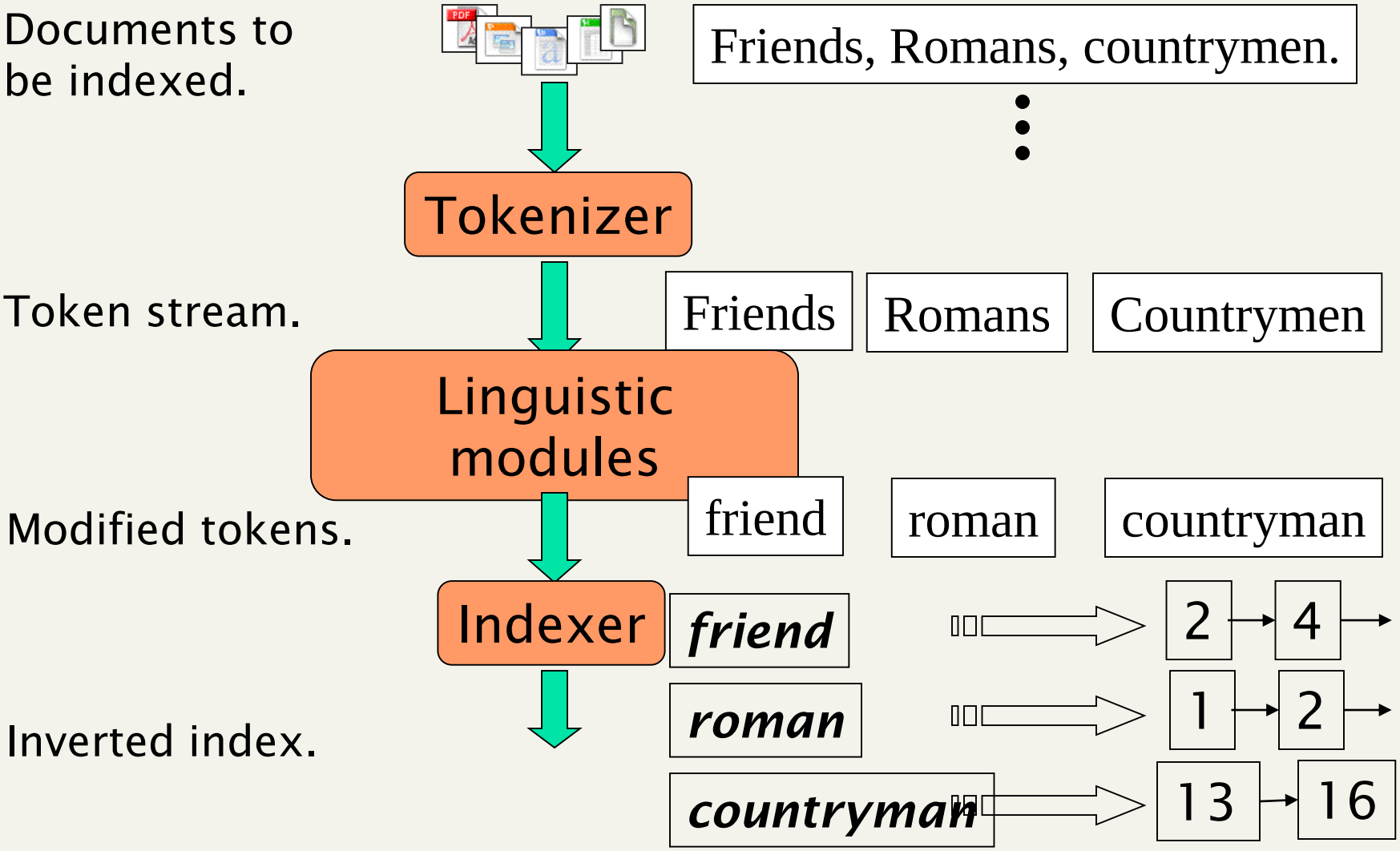


Plan for this lecture

Elaborate basic indexing

- Preprocessing to form the term vocabulary
  - Documents
  - Tokenization
  - What *terms* do we put in the index?
- Postings
  - Faster merges: skip lists
  - Positional postings and phrase queries

Recall basic indexing pipeline



Parsing a document

- What format is it in?
  - pdf/word/excel/html?
- What language is it in?
- What character set is in use?

Each of these is a classification problem, which we will study.

But these tasks are often done heuristically …

Complications: Format/language

- Documents being indexed can include docs from many different languages
  - A single index may have to contain terms of several languages.
- Sometimes a document or its components can contain multiple languages/formats
  - French email with a German pdf attachment.
- What is a unit document?
  - A file?
  - An email? (Perhaps one of many in an mbox.)
  - An email with 5 attachments?
  - A group of files (PPT or LaTeX as HTML pages)

Tokens and Terms

Tokenization

- Input: "***Friends, Romans and Countrymen***"
- Output: Tokens
  - ***Friends***
  - ***Romans***
  - ***Countrymen***
- Each such token is now a candidate for an index entry, after further processing
  - Described below
- But what are valid tokens to emit?

Tokenization

- Issues in tokenization:
  - ***Finland's capital*** → ***Finland? Finlands? Finland's***?
  - ***Hewlett-Packard*** → ***Hewlett*** and ***Packard*** as two tokens?
    - ***state-of-the-art***: break up hyphenated sequence.
    - ***co-education***
    - ***lowercase***, ***lower-case***, ***lower case*** ?
    - It's effective to get the user to put in possible hyphens
  - ***San Francisco***: one token or two? How do you decide it is one token?

- For example, if the document to be indexed is **to sleep perchance to dream**, then there are five tokens, but only four types (because there are two instances of to). However, if to is omitted from the index (as a stop word; then there are only three terms: sleep, perchance, and dream.).
- The major question of the tokenization phase is what are the correct tokens to use? In this example, it looks fairly trivial: you chop on whitespace and throw away punctuation characters.
- Example.
- Mr. O'Neill thinks that the boys' stories about Chile's capital aren't amusing.

neill

oneill

o'neill

o' neill

o neill ?

And for *aren't*, is it:

aren't

arent

are n't

aren t ?

A simple strategy is to just split on all non-alphanumeric characters, but although o neill looks okay, aren t looks intuitively bad.

In how many cases would a query of o'neill and capital match

generally by processing queries with the same Tokenizer. This guarantees that a sequence of characters in a text will always match the same sequence typed in a query.

Numbers

- *3/12/91* *Mar. 12, 1991*
- *55 B.C.*
- *B-52*
- *My PGP key is 324a3df234cb23e*
- *(800) 234-2333*
  - Often have embedded spaces
  - Often, don't index as text
    - But often very useful: think about things like looking up error codes/stacktraces on the web
    - (One answer is using n-grams: Lecture 3)
  - Will often index "meta-data" separately
    - Creation date, format, etc.

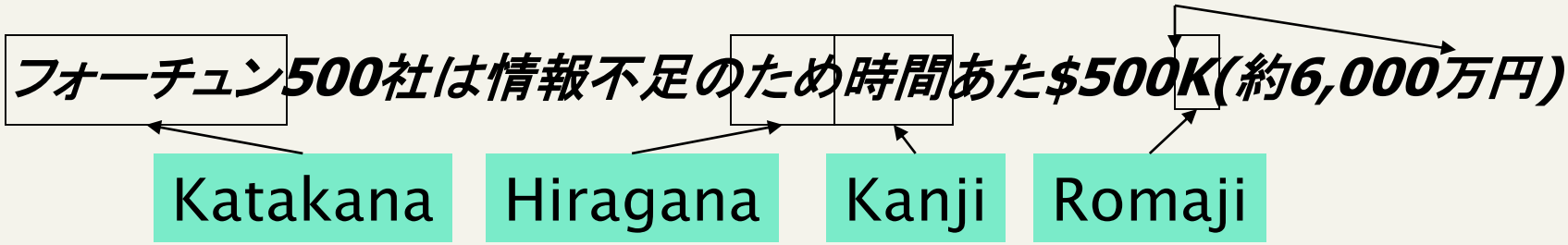
Tokenization: language issues

- French
  - ***L'ensemble*** → one token or two?
    - ***L*** ? ***L'*** ? ***Le*** ?
    - Want ***l'ensemble*** to match with ***un ensemble***

- German noun compounds are not segmented
  - ***Lebensversicherungsgesellschaftsangestellter***
  - 'life insurance company employee'
  - German retrieval systems benefit greatly from a **compound splitter** module

Tokenization: language issues

- 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。
  - Not always guaranteed a unique tokenization
- Further complicated in Japanese, with multiple alphabets intermingled
  - Dates/amounts in multiple formats



End-user can express query entirely in hiragana!

Stop words

- With a stop list, you exclude from dictionary entirely the commonest words. Intuition:
  - They have little semantic content: *the, a, and, to, be*
  - There are a lot of them: ~30% of postings for top 30 wds
- But the trend is away from doing this:
  - Good compression techniques means the space for including stop words in a system is very small
  - Good query optimization techniques mean you pay little at query time for including stop words.
  - You need them for:
    - Phrase queries: “King of Denmark”
    - Various song titles, etc.: “Let it be”, “To be or not to be”
    - “Relational” queries: “flights to London”

Normalization

- Need to “normalize” terms in indexed text as well as query terms into the same form
  - We want to match ***U.S.A.*** and ***USA***
- We most commonly implicitly define equivalence classes of terms
  - e.g., by deleting periods in a term
- Alternative is to do asymmetric expansion:
  - Enter: ***window*** Search: ***window, windows***
  - Enter: ***windows*** Search: ***Windows, windows, window***
  - Enter: ***Windows*** Search: ***Windows***
- Potentially more powerful, but less efficient

Normalization: other languages

- Accents: ***résumé*** vs. ***resume***.
- Most important criterion:
  - How are your users like to write their queries for these words?
- Even in languages that standardly have accents, users often may not type them
- German: ***Tuebingen*** vs. ***Tübingen***
  - Should be equivalent

Case folding

- Reduce all letters to lower case
  - exception: upper case in mid-sentence?
    - e.g., ***General Motors***
    - ***Fed*** vs. ***fed***
    - ***SAIL*** vs. ***sail***
  - Often best to lower case everything, since users will use lowercase regardless of 'correct' capitalization…

- Aug 2005 Google example:
  - ***C.A.T.*** → Cat Fanciers website *not* Caterpiller Inc.

Stemming and lemmatization

- The goal of both stemming and lemmatization is to reduce inflectional
- forms and sometimes derivationally related forms of a word to a common
- base form. For instance
- e.g., ***car*** = ***automobile***
  - ***color*** = ***colour***
- Rewrite to form equivalence classes
- Index such equivalences
  - When the document contains ***automobile***, index it under ***car*** as well (usually, also vice-versa)

Lemmatization

- Reduce inflectional/variant forms to base form
- E.g.,
  - *am, are, is* → *be*
  - *car, cars, car's, cars'* → *car*
- *the boy's cars are different colors* → *the boy car be different color*
- Lemmatization implies doing "proper" reduction to dictionary headword form

Stemming

- Reduce terms to their “roots” before indexing
- “Stemming” suggest crude affix chopping
  - language dependent
  - e.g., ***automate(s), 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's algorithm

- Commonest algorithm for stemming English
  - Results suggest it's at least as good as other stemming options
- Conventions + 5 phases of reductions
  - phases applied sequentially
  - each phase consists of a set of commands
  - sample convention: *Of the rules in a compound command, select the one that applies to the longest suffix.*