

INFORMATION RETRIEVAL

Luca Manzoni

lmanzoni@units.it

WHO AM I

AND SOME INFORMATION ABOUT THE COURSE

Luca Manzoni

email: lmanzoni@units.it

office: C5, 2nd floor, room 3.20

Information retrieval & data visualisation

Luca Manzoni

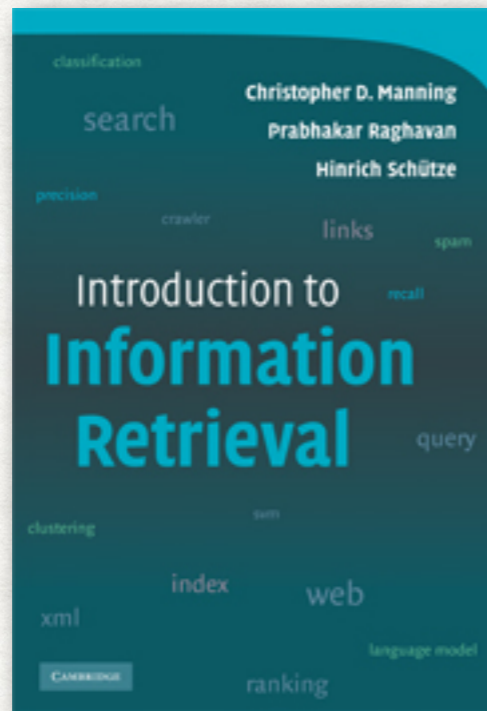
November and (early) December

Tea Tušar

(you already did it)

All the material of the course will be available online on Moodle and Teams

BOOK



Christopher D. Manning
Prabhakar Raghavan
Hinrich Schütze

Introduction to Information Retrieval
Cambridge University Press. 2008.

Freely available at: <https://nlp.stanford.edu/IR-book/>

THE EXAM (1)

How

Project + presentation of the project (with questions)

When

Mid December and Mid January (we can discuss to fix a day).

The projects will be assigned by the end of November

Is the project the same for everybody?

No, each student will have a different project

How can we select a project?

There will be a selection of project to choose from
or you can propose your own

THE EXAM (2)

What kind of project?

Code or report (approx. 10 pages)

Can I use language X?

Generally yes, but please ask if it is not in the following list:
Python, C, C++, Java, R, Julia, Common Lisp, Clojure, Haskell

I want to do a project on X

We can discuss and prepare a personalised
project on X

How is the final mark computed?

It is the average of the marks in the two parts

OUTLINE OF THE COURSE

INFORMATION RETRIEVAL PART

- Introduction to information retrieval (IR)
- Data structures for IR
- Models for IR:
 - Boolean
 - Vector space
 - Probabilistic
- Evaluation of IR
- IR on the Web
- Recommender Systems

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)

*Manning, Raghavan, Schütze
Introduction to Information Retrieval*

(IR) part of computer science which studies
the retrieval of **information** (not data)
from a **collection of written documents**.
The retrieved documents aim at satisfying
a **user information need**
usually expressed in natural language.

Baeza-Yates, Ribeiro-Neto
Modern Information Retrieval

EXAMPLES OF INFORMATION RETRIEVAL

- Web search
- Searching emails in your email client
- Searching documents using
Spotlight/Windows Desktop Search/Tracker/Nepomuk
- Search inside a knowledge base

A BRIEF HISTORY OF IR

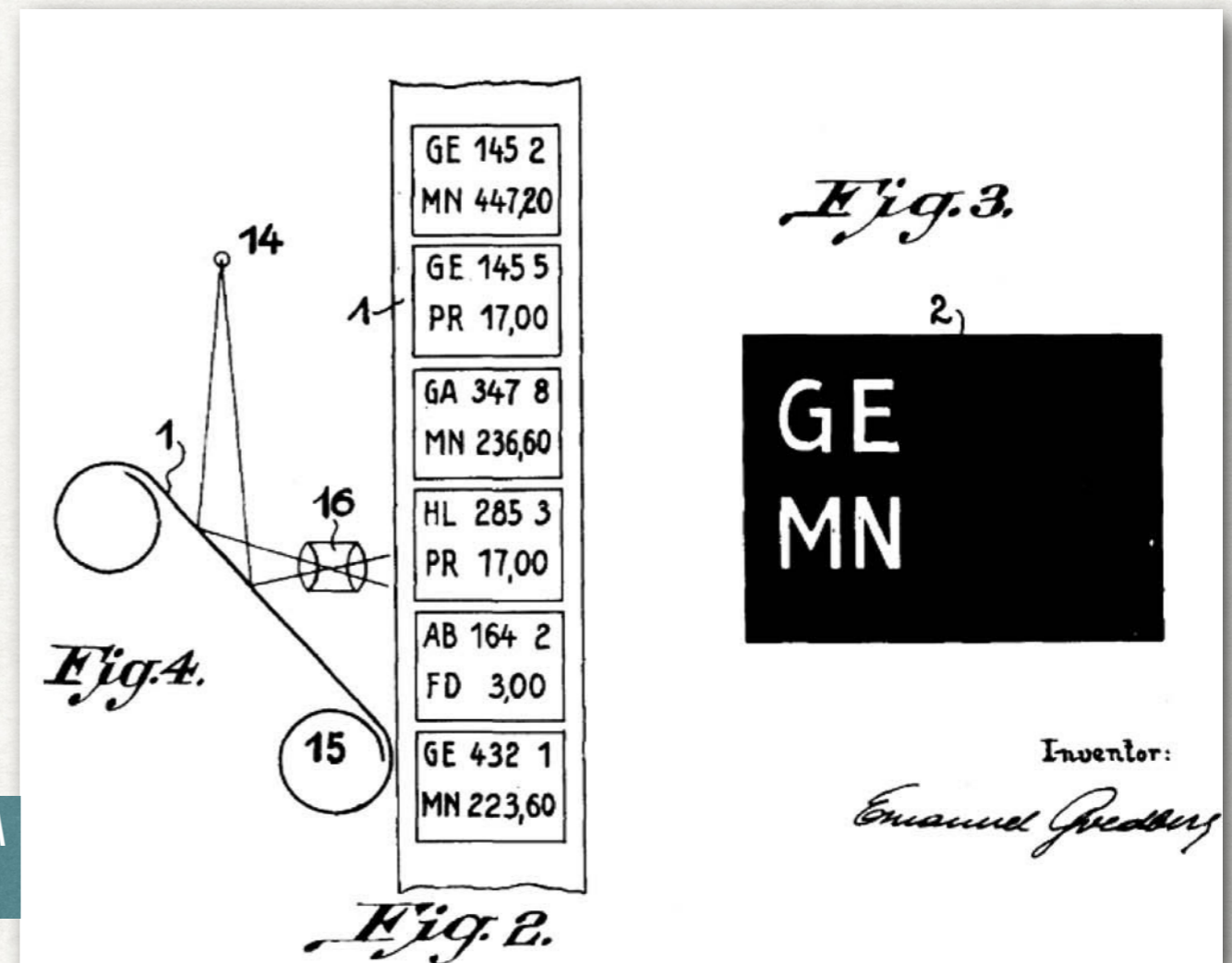
BEFORE COMPUTERS

The origin of IR is in the cataloguing of books in libraries

Each books has one or more topics associated to it

Multiple machines were invented to help librarians in retrieving books

MACHINE FOR RETRIEVAL BASED ON MICROFILM AND A PHOTOCELL. GOLDBERG, 1931



Sanderson, Mark, and W. Bruce Croft.

"The history of information retrieval research."

Proceedings of the IEEE 100.Special Centennial Issue (2012): 1444-1451.

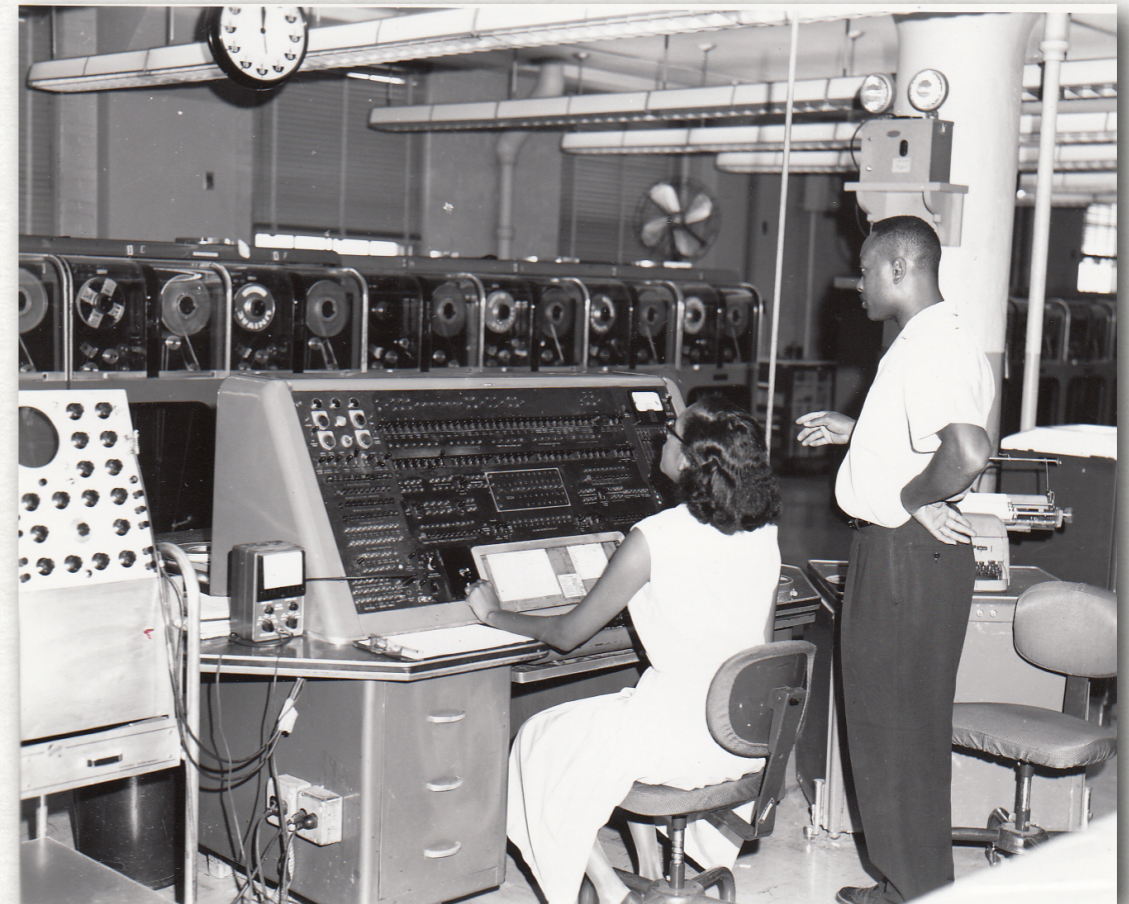
A BRIEF HISTORY OF IR

EARLY USE OF COMPUTERS

The use of computer allowed to speed up the traditional search methods, like searching across 10^6 records in 15 hours using a UNIVAC computer

But new techniques were also being developed:

First move to indexing based on words instead of assigning topics to document (*Uniterm system, 1952*)



How to rank the results: position based on the frequency of the words in the documents (*term frequency weighting, 1958*)

A BRIEF HISTORY OF IR

PRE-INTERNET ('60–EARLY '90)

Clustering of documents
with similar contents

Representing documents as vectors:
the **vector space mode**

Relevance feedback:
the user feedback can be used
to improve the query results

First private companies focused
on information retrieval

tf-idf: less common words tend
to refer to more specific concepts,
which were more important in retrieval

Text REtrieval Conference (**TREC**)

A BRIEF HISTORY OF IR

WEB SEARCH

1993: ~100 websites

2019: ~2 billions websites

Instead of using an authoritative source
now the web must be scraped to get information

The links between the pages
are a source of additional information



Altavista:
first fully-searchable
index of the web (1995)



Google:
currently the most used
web search engine (1997)

SOME TERMINOLOGY

DOCUMENTS AND COLLECTIONS

- **Document:** individual unit on which we build an IR system
 - Books, chapters, webpages, scientific papers, etc.
 - Images, videos, Music, etc.
- **Collection (also called Corpus):** the group of documents on which we perform the search
 - All Shakespeare plays, the emails in your mailbox, all the products on an online shop, the web
 - Some collections are static, while others are dynamic

PROPERTIES OF UNSTRUCTURED DOCUMENTS

DOCUMENT

- Significant text contents
- Some structure might be present (e.g., title, author)
- The semantics of the document is not well-defined
- Example query:
"find all stories about holidays in Ireland"

DATABASE RECORD

- Predefined structure
- Usually the semantics of the fields is well defined
- Finding matches is (usually) done by comparing specific fields
- Example Query: *"select all products that cost at most 50€"*

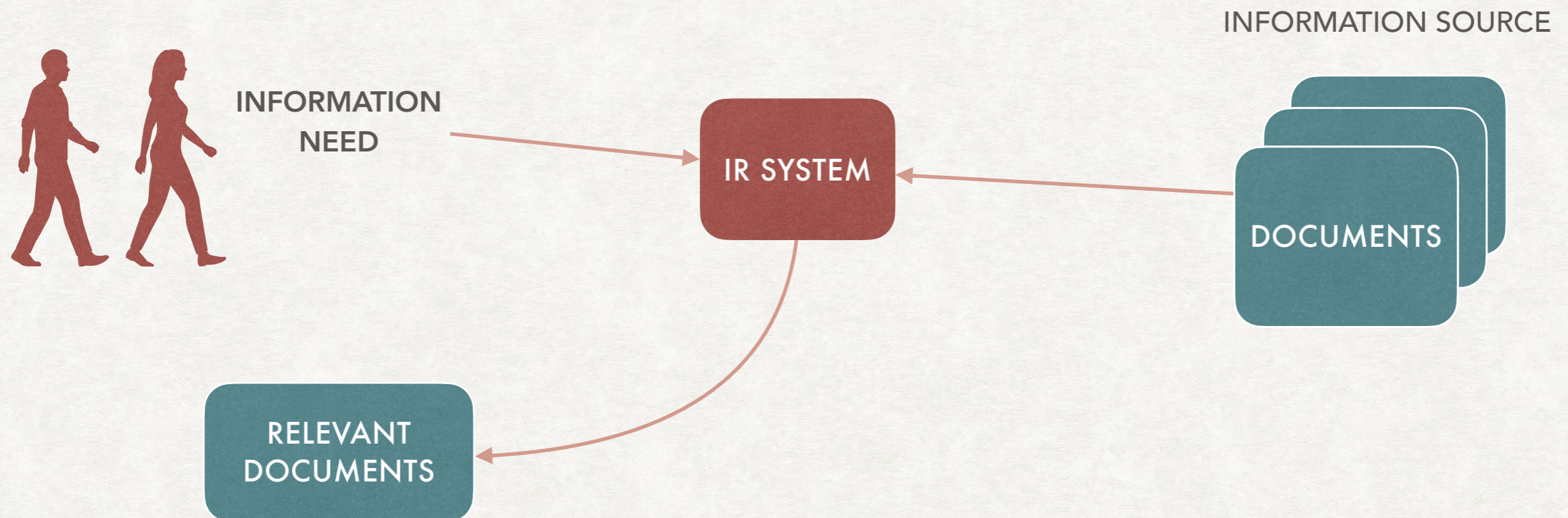
SOME TERMINOLOGY (2)

FROM INFORMATION NEED TO QUERIES

- **Information need:** a topic about which the user what to know more
- **Query:** the way the user formulate his/her information need to the IR system
 - For the same information need the users might formulate different queries
 - E.g., what query would you use to know which is the current record holder in Tetris?

GOAL OF AN IR SYSTEM

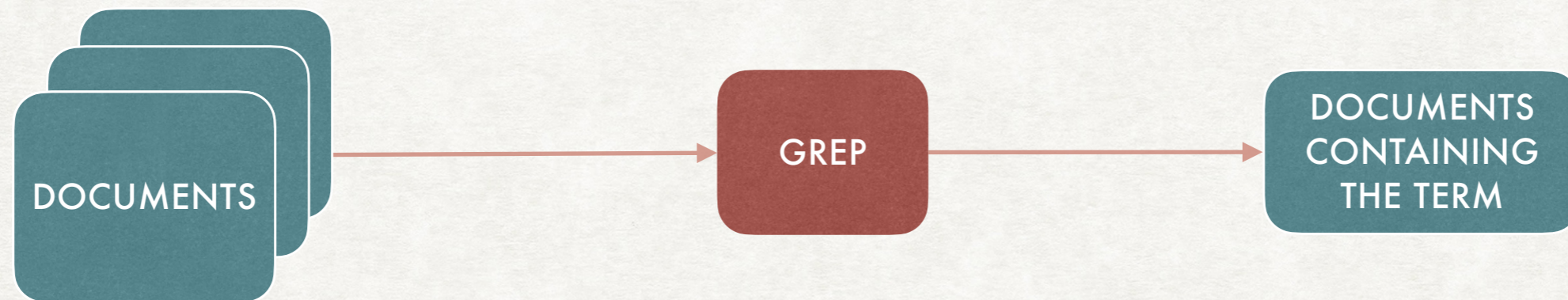
An IR System must interpret the information needs of the user and estimate the relevance of the documents with respect to it.



IS AN IR SYSTEM JUST "GREP"?

HINT: NO

```
> grep term_to_search corpus
```



- Searching across all the text does not scale to large collections
- We might want a more flexible query language
- We might want *ranked* retrieval (i.e., more relevant documents first)

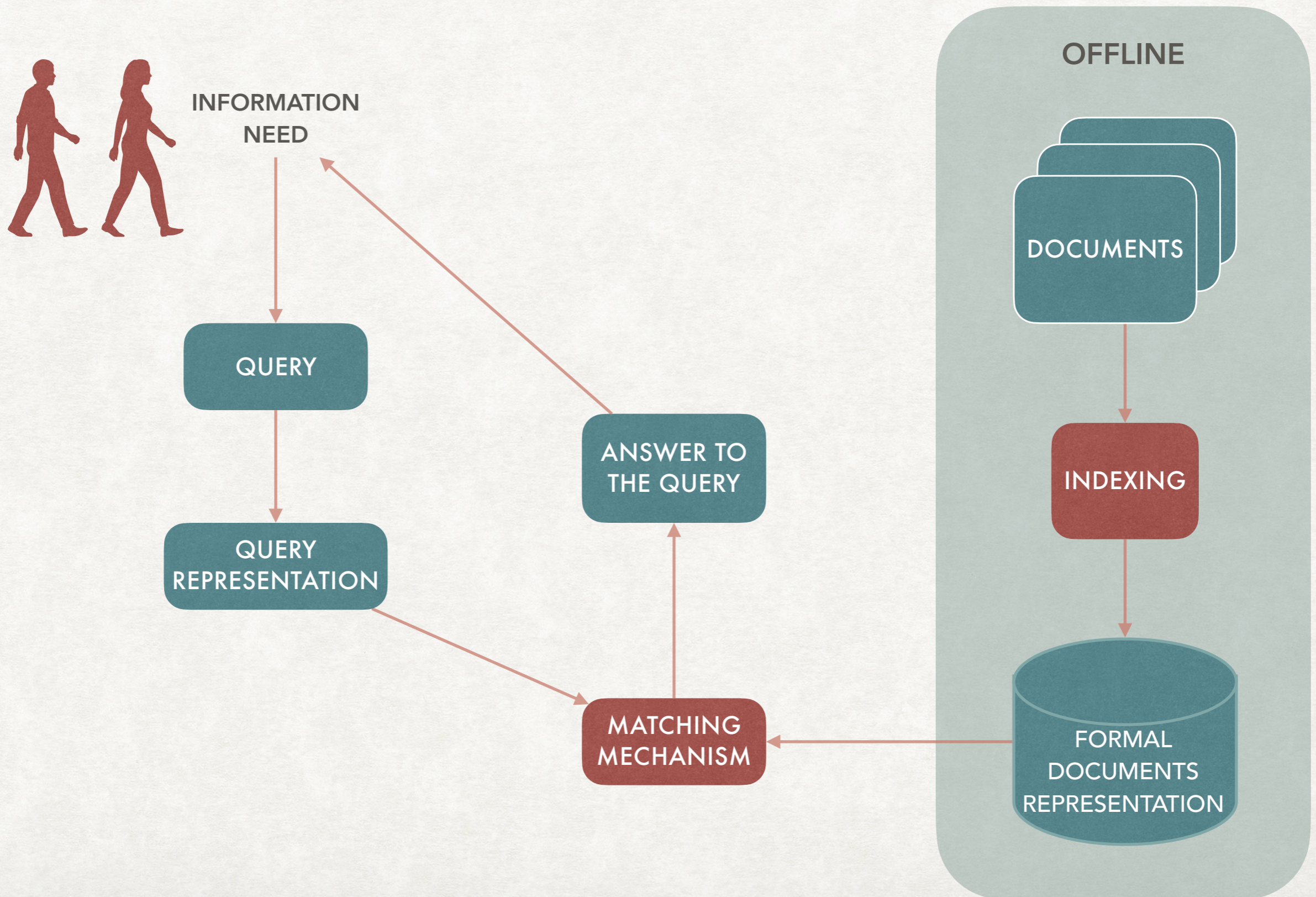
INDEXING

HOW TO AVOID GREPPING

- It is unfeasible to scan the entire corpus for each query...
- ...therefore, the corpus is scanned once* and an **index** is built
- An index will have for each **term** (e.g., a word) all documents containing that term
- In this way we can avoid a linear scan of all documents

*assuming it never changes

STRUCTURE OF AN IR SYSTEM



MAIN COMPONENTS

A FORMAL REPRESENTATION
OF THE QUERIES

Exact matching
The document is either
relevant or not relevant

A WAY OF MATCHING
THE QUERY AND DOCUMENTS REPRESENTATIONS
AND A WAY TO MEASURE RELEVANCE

Partial matching
The document that are
"sufficiently similar"
to the query are recovered

A FORMAL REPRESENTATION
OF THE DOCUMENTS

DIFFERENT KINDS OF REPRESENTATIONS

QUERIES AND DOCUMENTS

Query

BOOLEAN
FORMULAE

+ EXTENSIONS

cat AND dogs

FREE FORM QUERIES

common house pets

IMAGES



Document

SET OF TERMS

POINT IN
A VECTOR SPACE

BINARY VECTOR

THE TWO ASPECTS OF IR

We need to manage technical and a semantic aspects

Technical aspects

- How to represent information in a computer?
- How to retrieve the information fast enough?

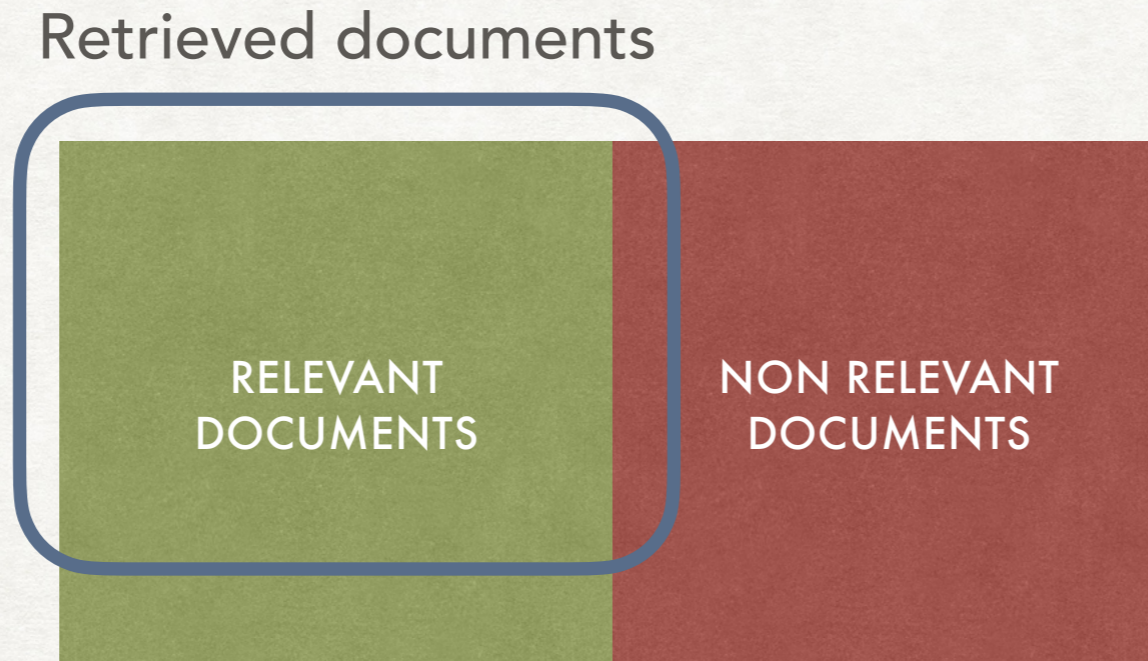
↓
EFFICIENCY

Semantic aspects

- How can we store information in a way that preserves its meaning?
- How can we retrieve the relevant documents?

↓
EFFECTIVENESS

MEASURING EFFECTIVENESS



$$\text{precision} = \frac{\text{relevant} \cap \text{retrieved}}{\text{retrieved}}$$

Which fraction
of the retrieved documents
is relevant

$$\text{recall} = \frac{\text{relevant} \cap \text{retrieved}}{\text{relevant}}$$

Which fraction
of the relevant documents
has been retrieved

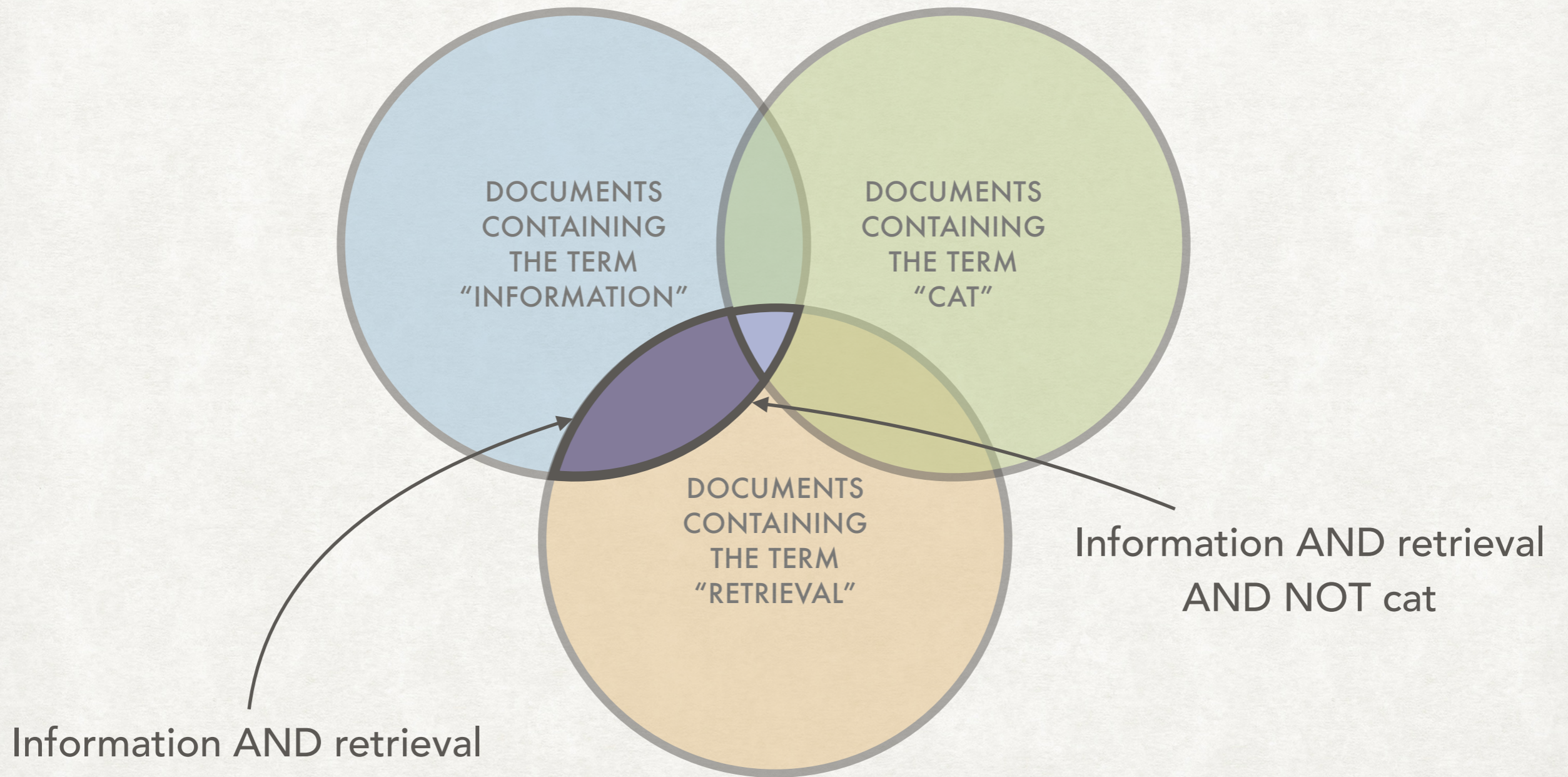
BOOLEAN RETRIEVAL

WHAT IS BOOLEAN RETRIEVAL?

- Only exact matching: either a document is relevant or not
- The query is expressed like a Boolean formula:
 - E.g., (dog OR cat) AND box
- We can ask for the inclusion (or exclusion) of certain terms
- Going forward, we can decide to extend this model to allow more powerful queries
- For now we ignore the issue of ranking the results according to relevance.

A VISUAL EXAMPLE

BOOLEAN QUERIES AND SETS



LINKING BOOLEAN QUERIES AND SETS

$$d_1 = \{t_1, t_3, t_4\}$$

$$d_2 = \{t_1, t_2, t_5\}$$

$$d_3 = \{t_2, t_3, t_5\}$$

Documents
as sets of terms

$$t_1 = \{d_1, d_2\}$$

$$t_2 = \{d_2, d_3\}$$

$$t_3 = \{d_1, d_3\}$$

$$t_4 = \{d_1\}$$

$$t_5 = \{d_2, d_3\}$$

Terms as sets
of documents

Queries

Answers

$$q_1 = t_2 \longrightarrow \{d_2, d_3\}$$

$$q_2 = t_2 \wedge t_3 \longrightarrow \{d_2, d_3\} \cap \{d_1, d_3\} = \{d_3\}$$

$$q_3 = t_2 \vee t_3 \longrightarrow \{d_2, d_3\} \cup \{d_1, d_3\} = \{d_1, d_2, d_3\}$$

HOW TO ASSOCIATE DOCUMENTS AND TERMS

Let us consider as a corpus a set of ~400 articles from "Time"¹ in the '60s:

0. THE ALLIES AFTER NASSAU IN DECEMBER 1960, THE U.S. FIRST [...]
1. RUSSIA WHO'S IN CHARGE HERE ? IT WAS IN 1954 THAT NIKITA [...]
2. BERLIN ONE LAST RUN HANS WEIDNER HAD BEEN HOPING FOR MONTHS TO [...]
3. THE ROAD TO JAIL IS PAVED WITH NONOBJECTIVE ART SINCE THE [...]
-

We can build an incidence matrix of documents and terms

¹ Available at http://ir.dcs.gla.ac.uk/resources/test_collections/time/

THE INCIDENCE MATRIX

	Article 0	Article 1	Article 2	Article 3	Article 4	Article 5
A	1	1	1	1	1	1
AACHEN	0	0	0	0	0	0
ABABA	0	0	0	0	0	0
ABABAS	0	0	0	0	0	0
ABACK	0	0	0	0	0	0
ABADAN	0	0	0	0	0	0
ABANDON	0	0	0	0	0	0
ABANDONED	0	0	0	0	0	0
ABANDONING	0	0	0	0	0	0
ABANDONMENT	0	0	0	0	0	0

...and another 22484 rows

INCIDENCE MATRIX

STORAGE REQUIREMENTS

The size needed to store an incidence matrix is $\#terms \times \#documents$

Hence the size requirements make this data structure impractical

A simple example

The Oxford English Dictionary has over 171000 words in common use

In 2019 the English Wikipedia contains over 5.9×10^6 articles

The resulting incidence matrix would have more than 10^{12} entries

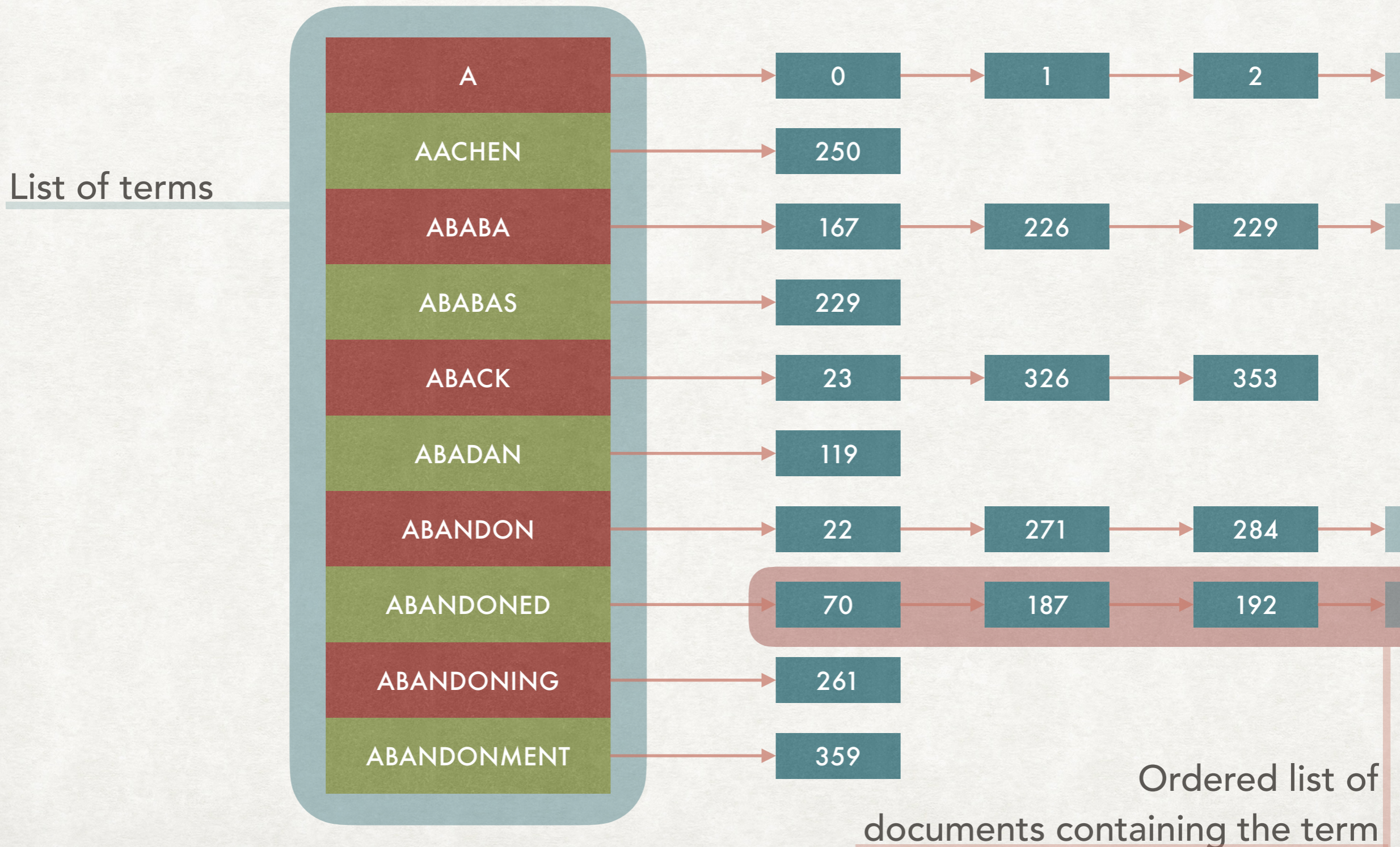
THE INVERTED INDEX

A MORE COMPACT DATA STRUCTURE

- We want to avoid storing rows that will be mostly empty
- We store, for each term, the list of documents containing it
- This is similar to the difference between adjacency *matrices* and adjacency *lists* for graphs.
- We keep the list ordered to improve performances for union and intersection
- Why **inverted** index? Because it is not from documents to terms but from terms to documents. Apart from that, it is a "normal" index

THE INVERTED INDEX

A VISUAL REPRESENTATION



SOME TERMINOLOGY (3)

THIS TIME FOR INVERTED INDICES

- **DocID** (Document Identifier): a unique number associated to a document. E.g., consecutive integers.
- **Dictionary**: the collection of all terms that we have in the inverted index
- **Posting list**: list of DocIDs associated to a term
- **Posting**: element of the list (different from a simple DocID because it is associated to a term)

BUILDING AN INVERTED INDEX

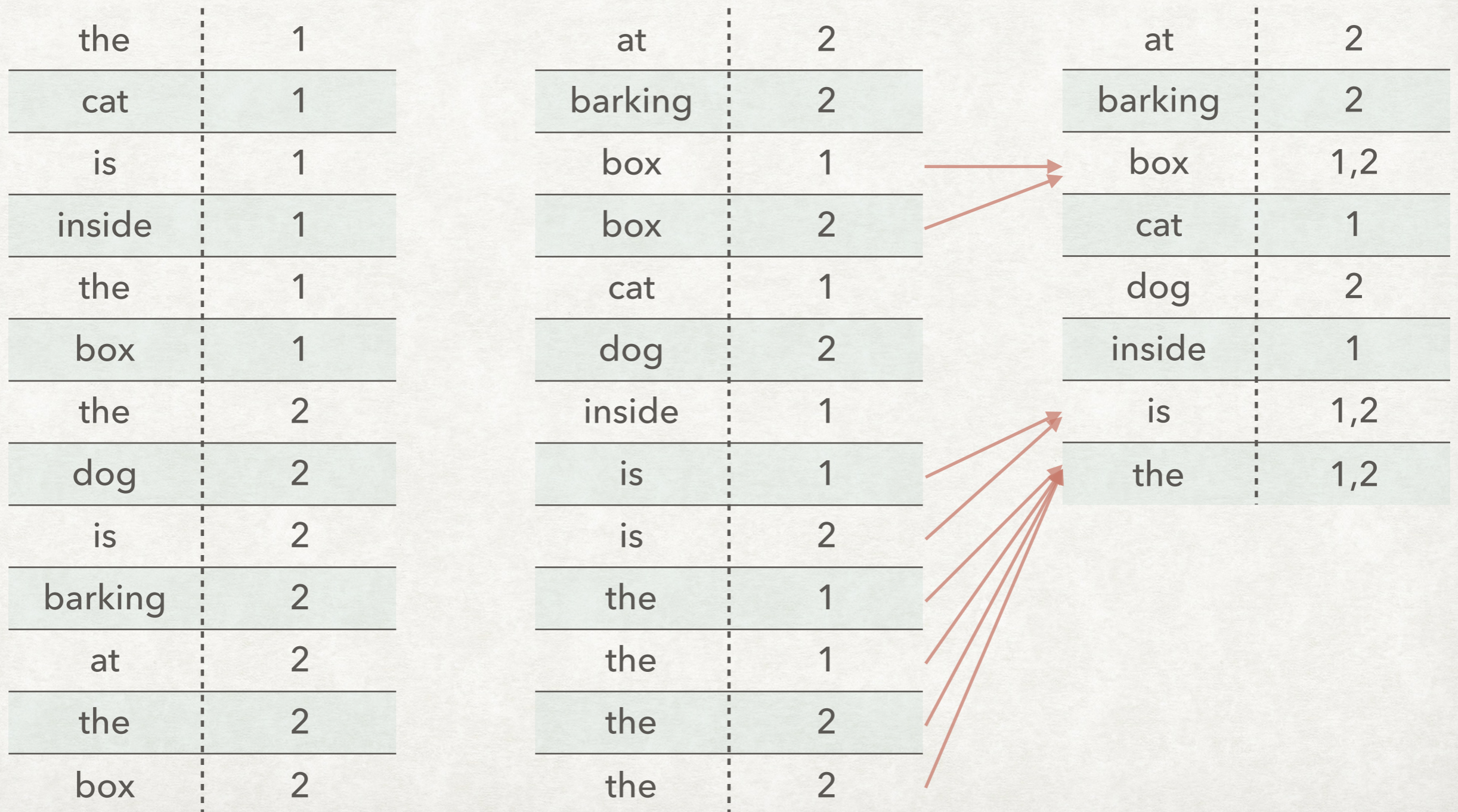
SORTING AND GROUPING

- For each document we extract the sequence of terms
- We tag each term with the corresponding DocID
- We sort the list of terms extracted from all the documents
- We group together equal terms and we “merge” the posting lists of the two terms

EXAMPLE OF SORTING AND GROUPING

Document 1: the cat is inside the box

Document 2: the dog is barking at the box



WHAT ABOUT THE SPACE REQUIREMENTS?

WORST-CASE BOUNDS AND THE REAL WORLD

- But... the space occupied by an inverted index is not necessarily lower than the one of the incidence matrix.
- In the worst case (each document contains all the terms) they both occupy $O(\#terms \times \#documents)$ space.
- In practice most document contains only a small subset of the terms.
- The same reasoning applies to the time complexity of the operations (intersection and union) performed on the set of documents.

**INVERTED INDEX:
UNION AND INTERSECTION**

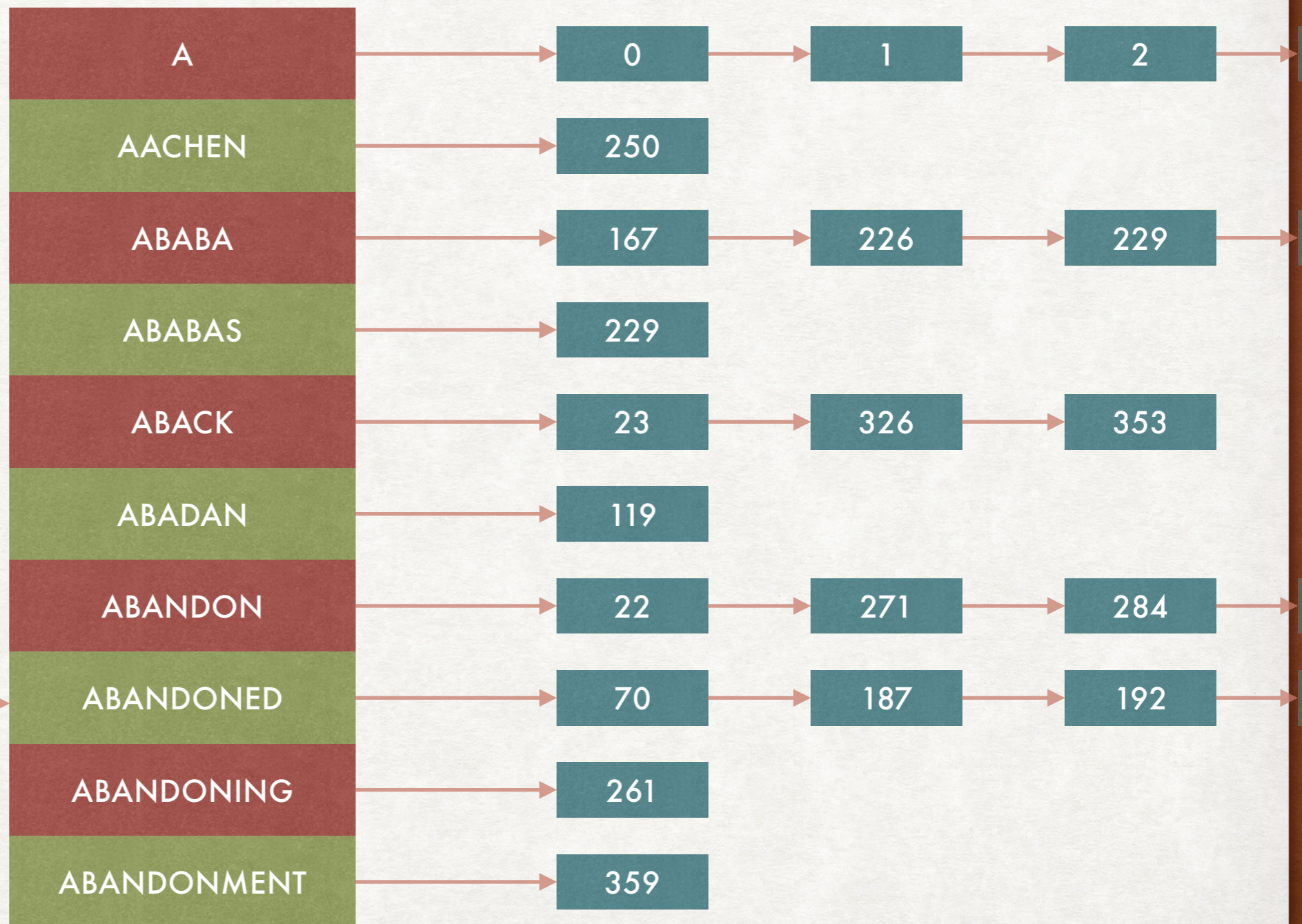
HOW TO IMPLEMENT AN INVERTED INDEX

BASIC IMPLEMENTATION AND OTHER IMPROVEMENTS

- We will spend some time in discussing how to implement and improve the inverted index
- Basic functionality: answer queries of the form
 - term1 AND term2
 - term1 OR term2
- Additional functionalities:
term1 NEAR term2, "term1 term2", term1* (wildcards), etc.
- How to compress the index, how to update it, etc.

ANSWERING A SIMPLE QUERY

A SINGLE WORD QUERY



We find the term
in the list of terms

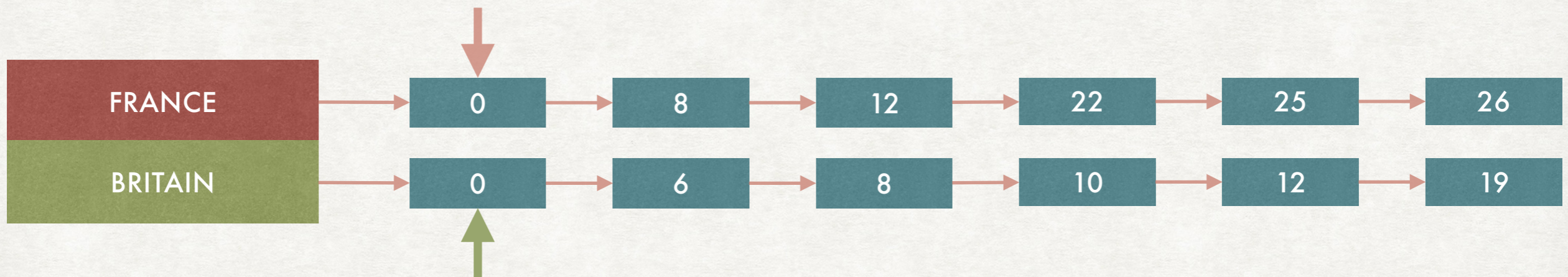
We return the
associated list of terms

ANSWERING AN "AND" QUERY

NOW WITH TWO WORDS



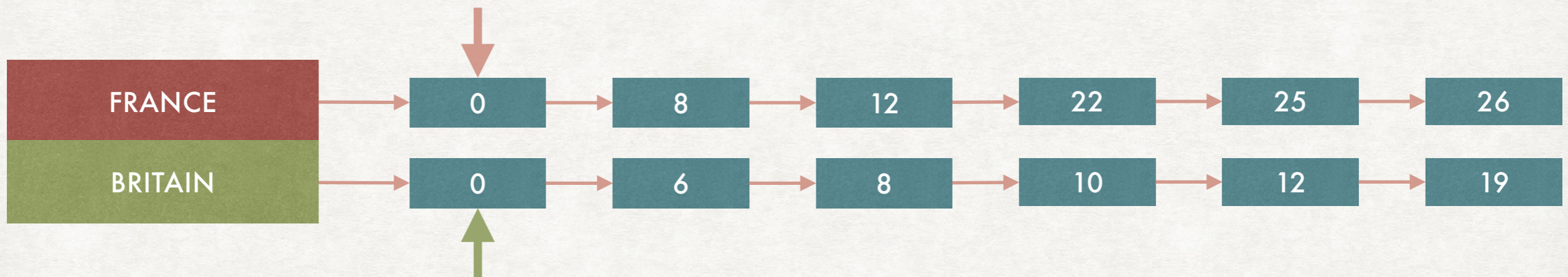
Now we need to compare the two lists of documents



ANSWER

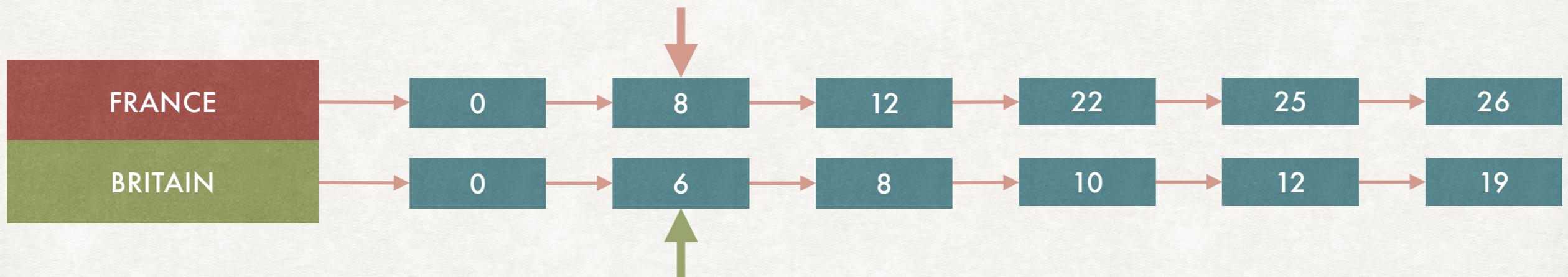
ANSWERING AN "AND" QUERY

NOW WITH TWO WORDS



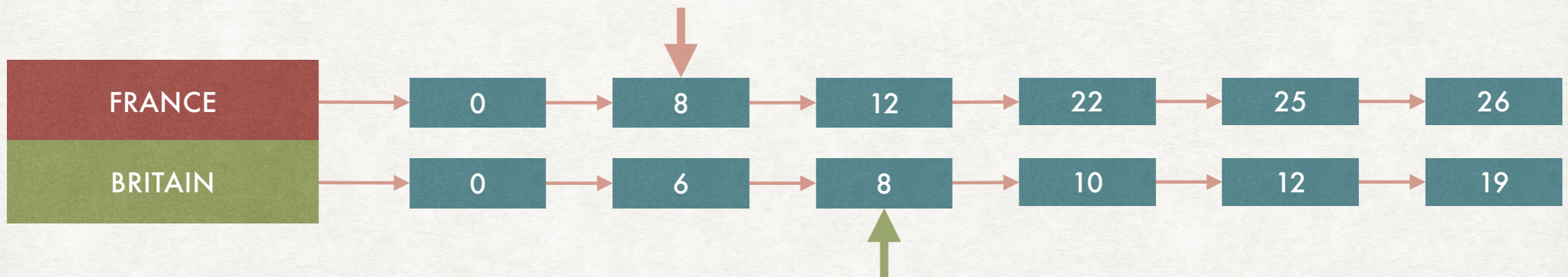
ANSWERING AN "AND" QUERY

NOW WITH TWO WORDS



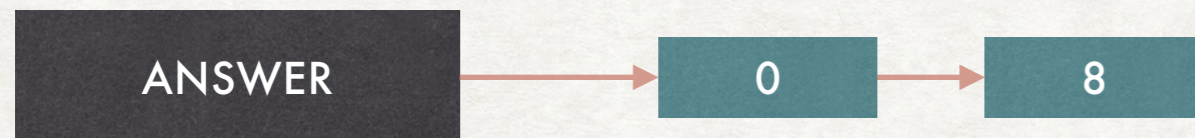
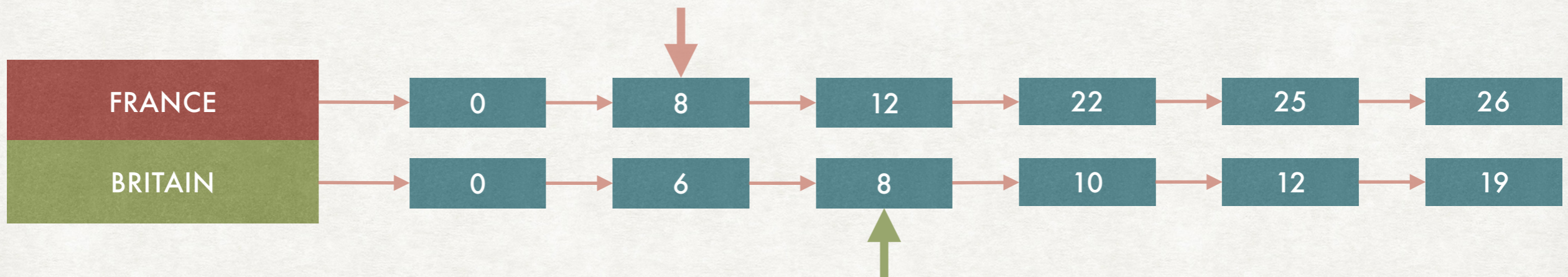
ANSWERING AN "AND" QUERY

NOW WITH TWO WORDS



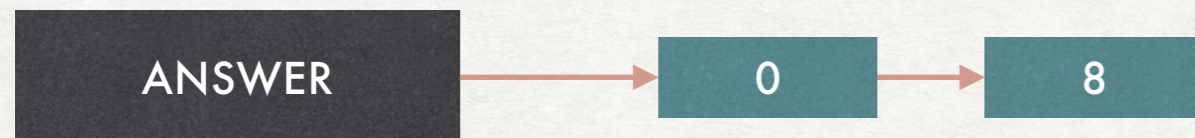
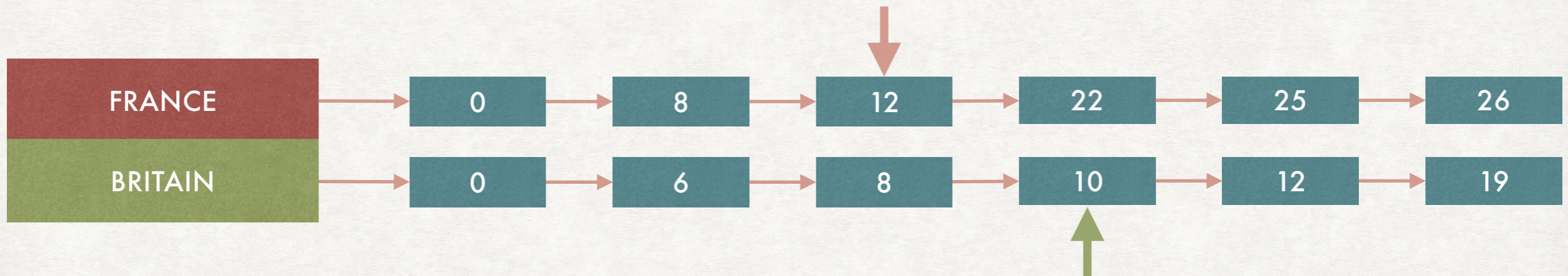
ANSWERING AN "AND" QUERY

NOW WITH TWO WORDS



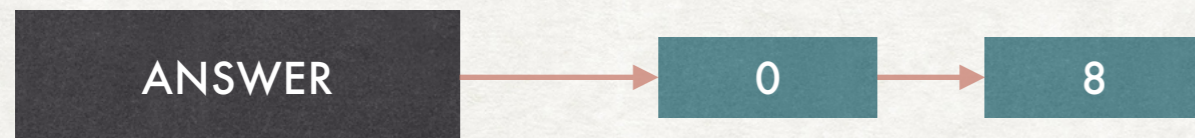
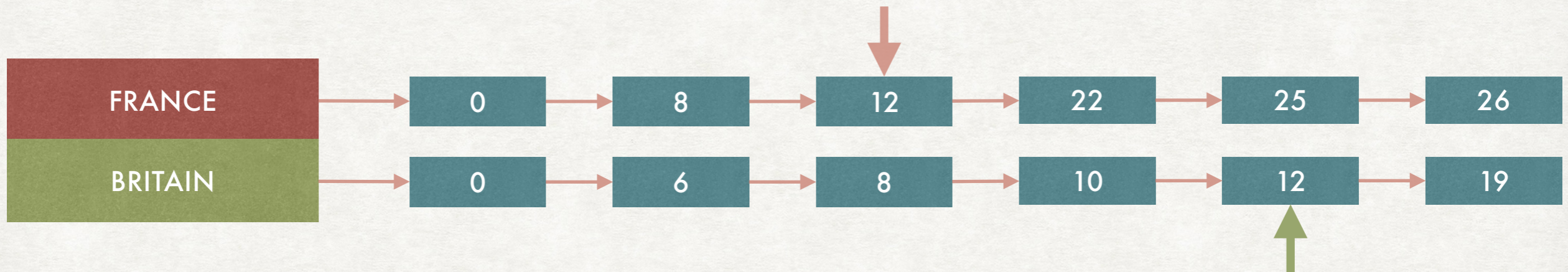
ANSWERING AN "AND" QUERY

NOW WITH TWO WORDS



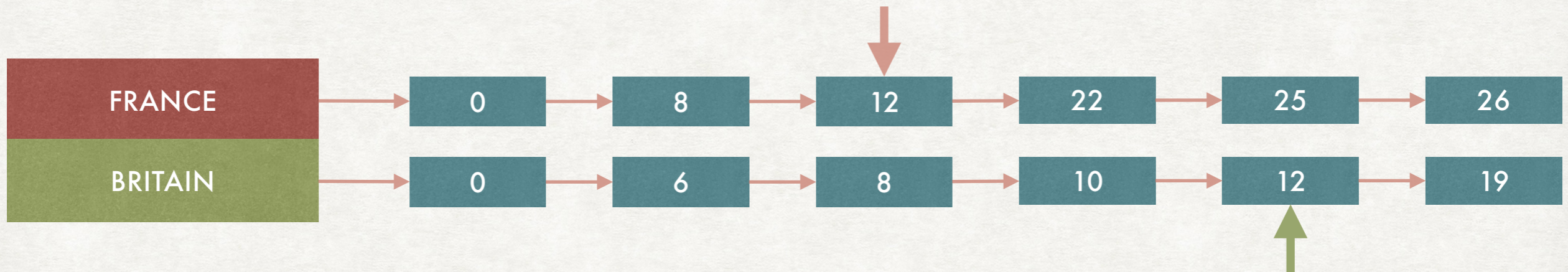
ANSWERING AN "AND" QUERY

NOW WITH TWO WORDS



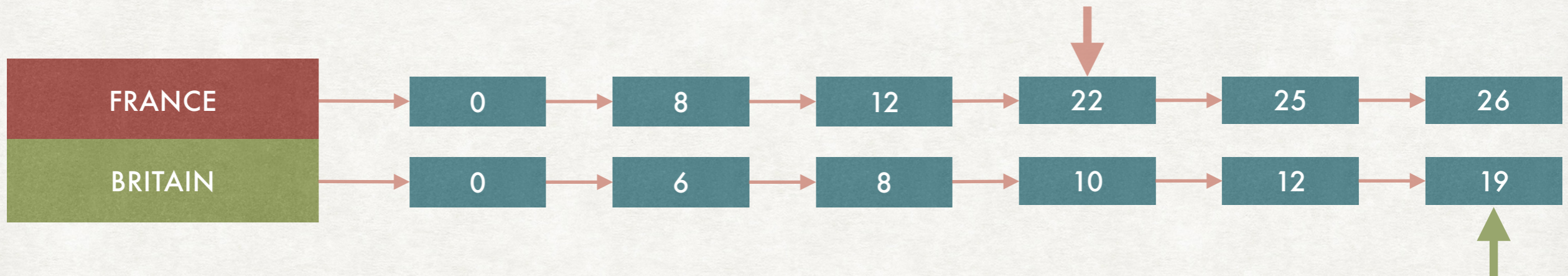
ANSWERING AN "AND" QUERY

NOW WITH TWO WORDS



ANSWERING AN "AND" QUERY

NOW WITH TWO WORDS



Complexity: linear in the lengths of the lists

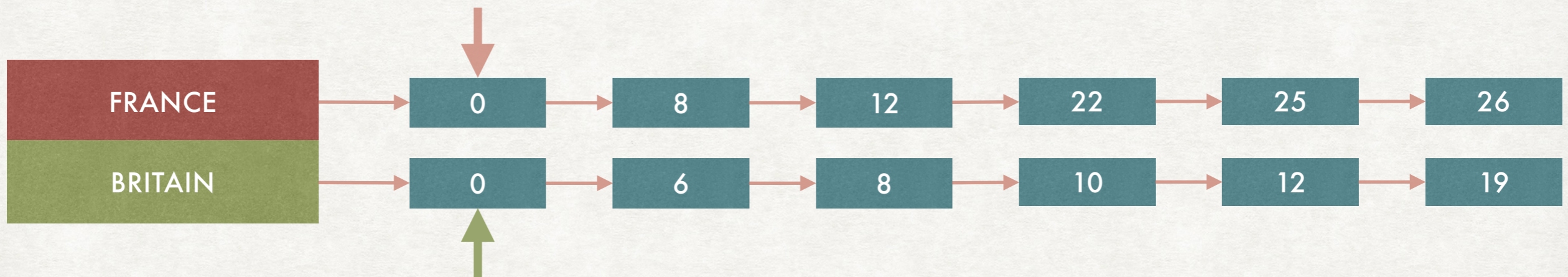


Size of the answer \leq minimum of the lengths of the lists

ANSWERING A "OR" QUERY WITHOUT DUPLICATES!

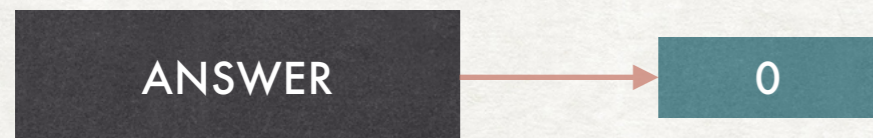
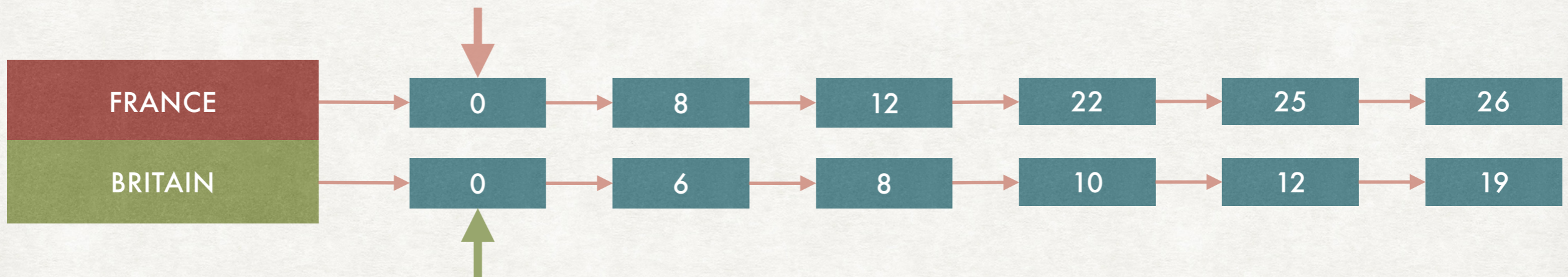


We still need to compare the two lists of documents

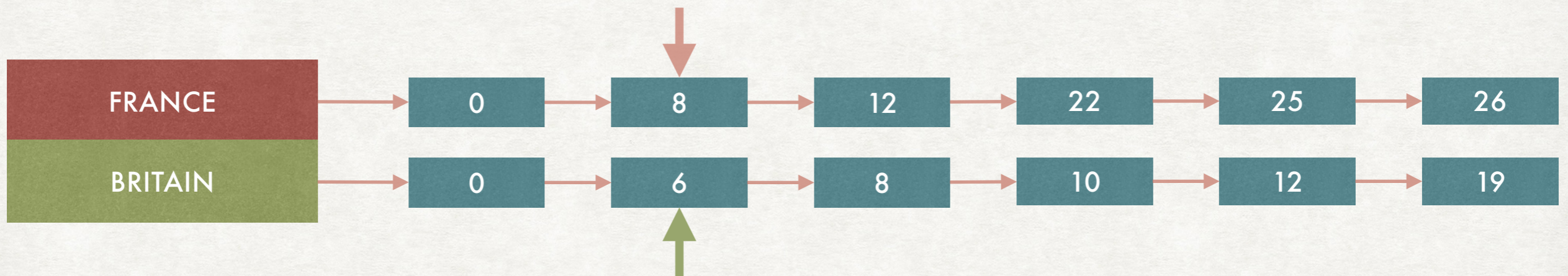


ANSWER

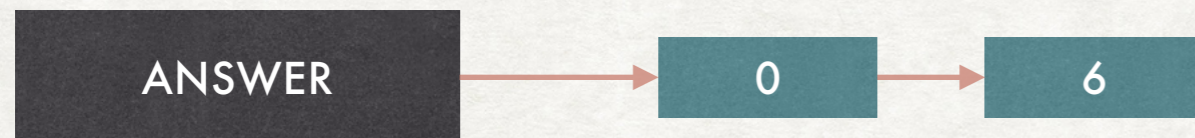
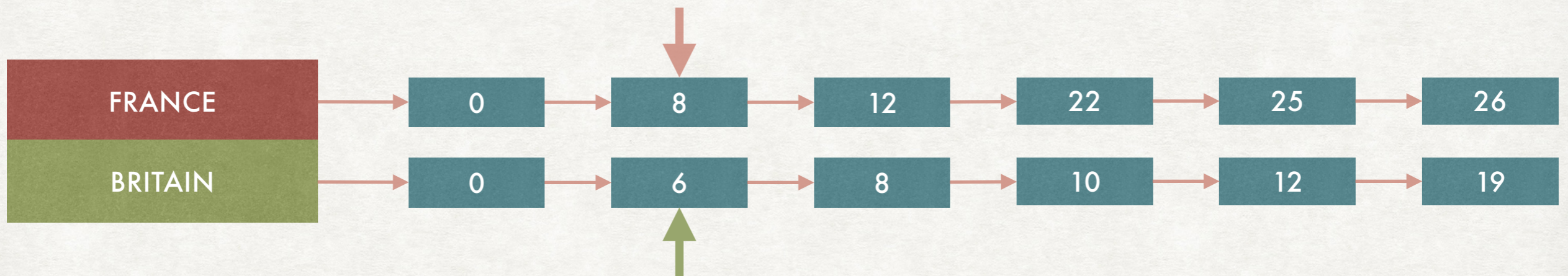
ANSWERING A "OR" QUERY WITHOUT DUPLICATES!



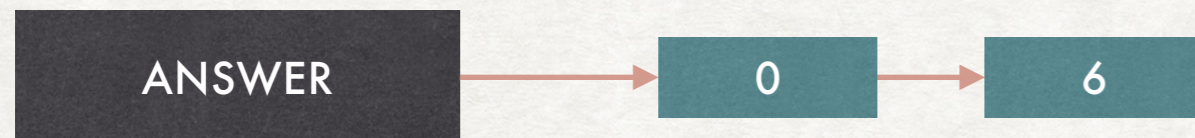
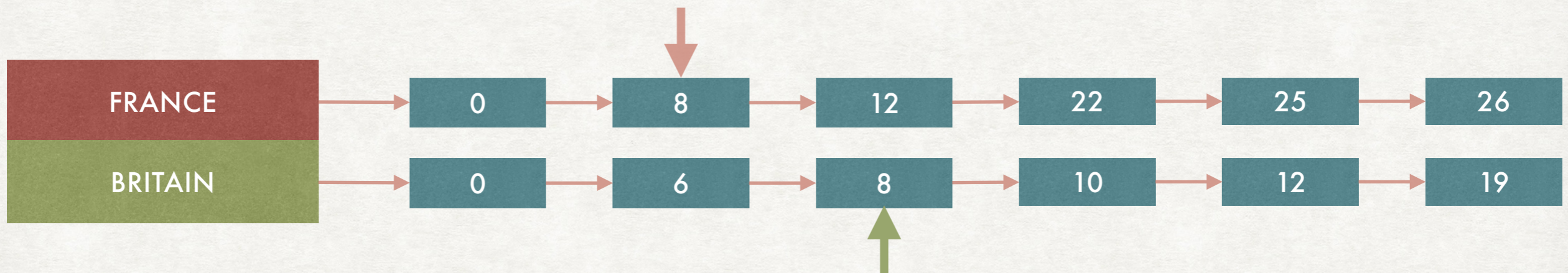
ANSWERING A "OR" QUERY WITHOUT DUPLICATES!



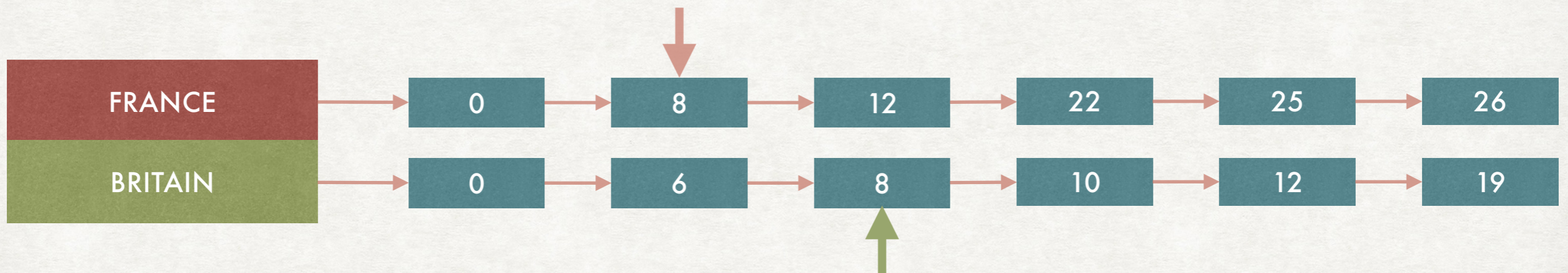
ANSWERING A "OR" QUERY WITHOUT DUPLICATES!



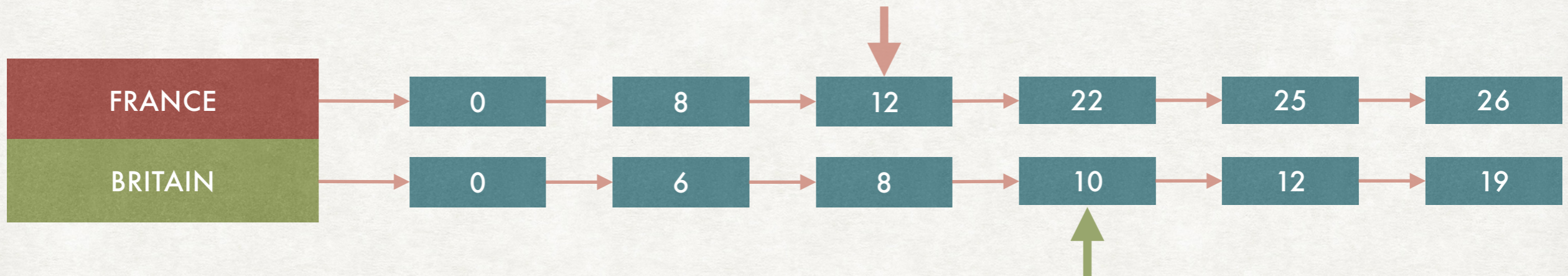
ANSWERING A "OR" QUERY WITHOUT DUPLICATES!



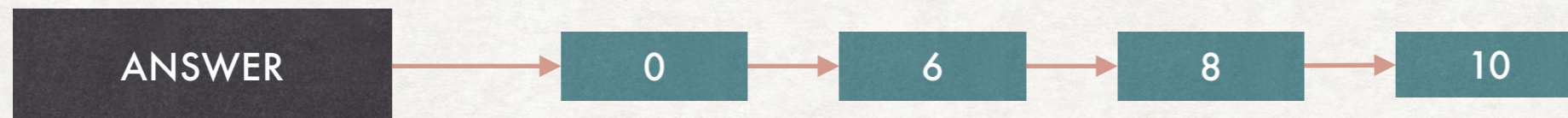
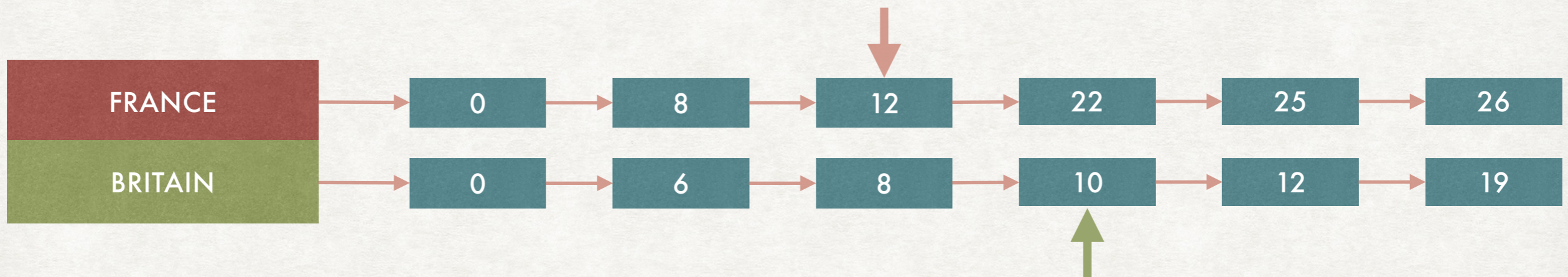
ANSWERING A "OR" QUERY WITHOUT DUPLICATES!



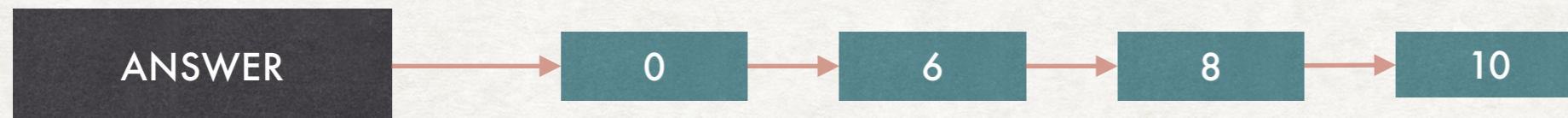
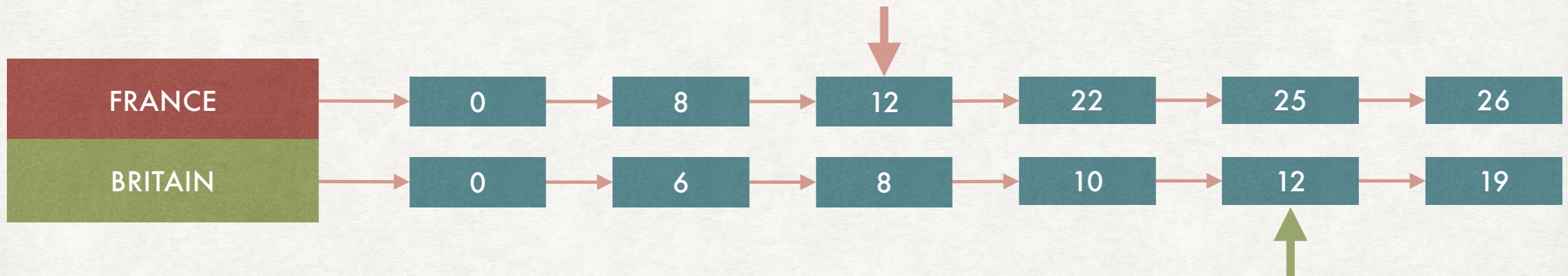
ANSWERING A "OR" QUERY WITHOUT DUPLICATES!



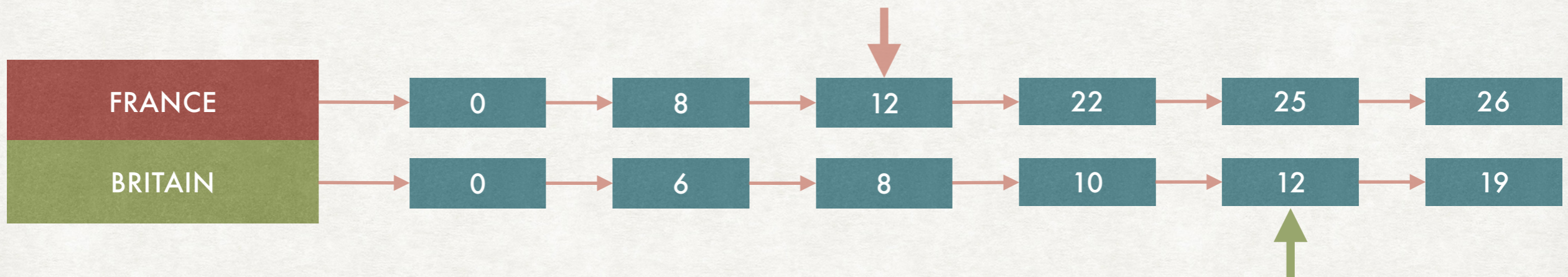
ANSWERING A "OR" QUERY WITHOUT DUPLICATES!



ANSWERING A "OR" QUERY WITHOUT DUPLICATES!



ANSWERING A "OR" QUERY WITHOUT DUPLICATES!



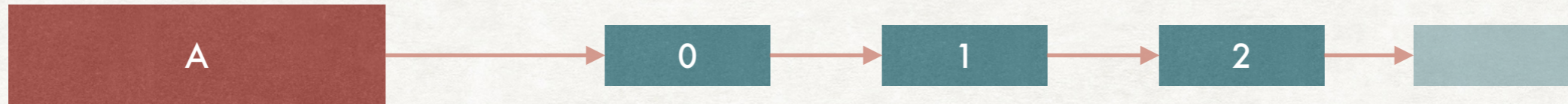
Complexity: linear in the lengths of the lists



Size of the answer \leq sum of the lengths of the lists

IS THAT ALL?

HINT: NO



Some terms are not useful: "A" is in all the documents!



Some terms are very similar semantically.

Example

Do we really want to keep "CAR" and "CARS" separated?

IMPROVING THE QUALITY OF RETRIEVAL

TERMINOLOGY (4)

THIS TIME FOR TOKENIZATION

- **Token:** instance of a sequence of characters
- **Type:** collection of all tokens with the same character sequence
- **Term:** a type that is inserted into the dictionary

THE CAT IS INSIDE THE BOX

Text

THE CAT IS INSIDE THE BOX

Tokens

THE CAT IS INSIDE BOX

Types (notice only one instance of "the")

CAT INSIDE BOX

Terms (after removal of common words)

TOKENIZATION

SPLITTING THE TEXT IN WORDS

- First step in the indexing process is to decide what is the granularity of the indexing (i.e., return chapters or paragraphs instead of entire books).
- The second step is to split a text sequence into tokens.
- In some cases deciding where to split the text sequence is simple...
- ...but in many others it is not, even in English.
- For others languages it might not even be clear where a word ends and the next one starts.

EXAMPLES OF PROBLEMATIC TOKENIZATION

Text	Possible tokenizations
New York	[New] [York]
File-system	[File] [system], [File-system]
555-1234 567	[555] [1234] [567], [555-1234] [567], [555-1234 567]
Upper case	[Upper] [case]
Uppercase	[Uppercase]
O'Hara	[O] [Hara], [O'Hara]
Aren't	[Aren][t], [Aren't]

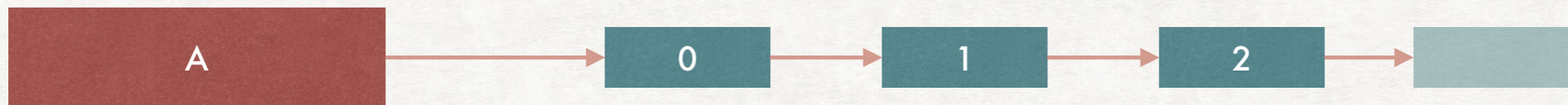
Possible (partial) solutions:

- use the same tokeniser for the documents and the queries
- use a collection of heuristics to decide where to split words

STOP WORDS

DROPPING COMMON TERMS

As anticipated before:



Some terms are not useful: "A" is in all the documents!

- **Stop words:** common words that do not help in selecting a document. They are discarded from the indexing and querying processes
- **Stop list:** list of stop words. Specific for a language/corpus. Usually consists of the most frequent words, curated for their semantic.

DISTRIBUTION OF WORDS

FREQUENCIES OF WORDS IN A CORPUS

Data extracted from the "Time" dataset



STOP WORDS FOR THE ENGLISH LANGUAGE

AND STOP WORDS FOR SPECIFIC TOPICS

- You can find multiple lists of stop words for the English language. They usually include words like:
 - a, about, above, after, again...
 - ... the, their, theirs, ... , your, yours, yourself, yourselves.
- The list of stop words is language specific: stop words in Italian are different (additional challenge: you might need to infer the language of a document).
- Stop lists can be specific by topic. E.g., in a "books on cats" corpus, the word "cat" might be a stop word.

PROBLEMS WITH STOP WORDS

SOMETIMES STOP WORDS ARE USEFUL

- You now have a IR system that removes all stop words.
- You receive the queries:
 - ~~To be or not to be~~
 - Dr ~~Who~~
 - ~~Do it yourself~~
 - ~~Let it be~~
- Removing stop words can reduce the *recall*.

PROBLEMS WITH STOP WORDS

SOMETIMES STOP WORDS ARE USEFUL

- A single stop word alone can usually be removed...
- ...but in a *phrase search* it might be important
- The trend is to have small (7-12 terms) or no stop word list but:
 - Use compression techniques to reduce the storage requirements
 - Use weighting to limit the impact of stop words
 - Use specific algorithms to limit the runtime impact of stop words

NORMALIZATION

REMOVING SUPERFICIAL DIFFERENCES

- The same word can be written in different ways and it must be normalized to allow the matching to occur.
- The idea is to define equivalence classes of terms, for example:
 - By ignoring capitalization (e.g., "HOME", "home", "HoMe").
 - By removing accents and diacritics (e.g., cliché is considered the same as cliche).
 - Other normalization steps specific to the language, like ignoring spelling differences (e.g., "colors" vs "colours").

RELATIONS BETWEEN UNNORMALIZED TOKENS

AN ALTERNATIVE TO EQUIVALENCE CLASSES

Sometimes capitalization and other features are important

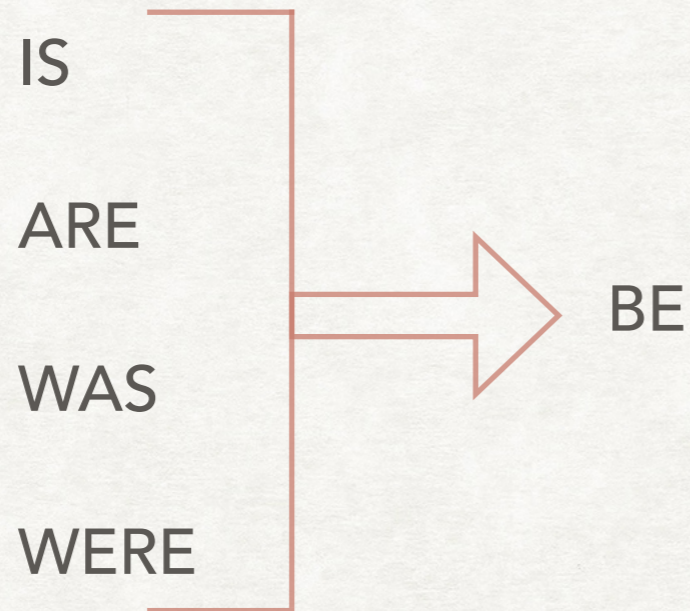
windows (can mean both the object and the OS)  Windows (the OS)

This can be solved by saving (possibly asymmetric) relations between token

Query Term	Equivalent terms
Windows	Windows
windows	Windows, windows, window
window	windows, window

STEMMING AND LEMMATIZATION

REDUCE WORDS TO A COMMON BASE FORM



Idea

reduce all variants of a word to a "common root"

Two main ways: stemming and lemmatization

Based on heuristics

Uses a vocabulary and morphological analysis

PORTER STEMMER

MOST USED STEMMER FOR THE ENGLISH LANGUAGE

Invented in 1979 (published 1980) by Martin Porter,
it is one of the most common stemmers for the English language

Five stages applied sequentially.

Each stage consists of a series of rewriting rules for words,
an example is given here

Rule	
SSES → SS	caressess → caress
IES → I	poinies → poni
SS → SS	caress → caress
S →	cats → cat

Porter Stemmer implementations: <https://tartarus.org/martin/PorterStemmer/>
(or you can read the original paper and the BCLP implementation)

THE "PREPROCESSING" PIPELINE

