INFORMATION RETRIEVAL

Luca Manzoni
lmanzoni@units.it

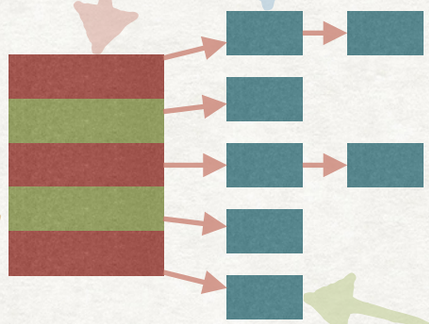
Lecture 2

LECTURE OUTLINE

*Now embellished with diagrams

Stemming & Lemmatization
Removing Stop Words

Arrays, linked lists,
and skip lists



Basic operations on
inverted indices

PRACTICAL PART
A PYTHON IMPLEMENTATION
OF A SIMPLE BOOLEAN
RETRIEVAL SYSTEM

Positional postings

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

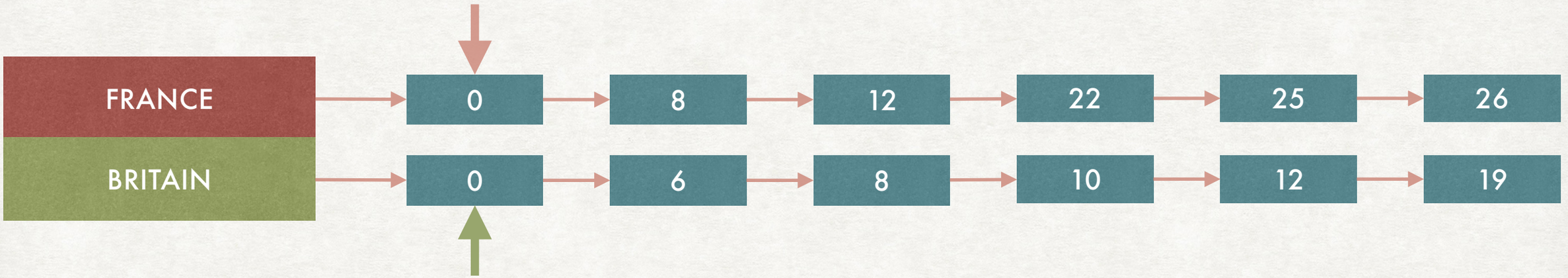
A SINGLE WORD QUERY

ANSWERING AN "AND" QUERY

NOW WITH TWO WORDS

QUERY	FRANCE	**AND**	BRITAIN

Now we need to compare the two lists of documents

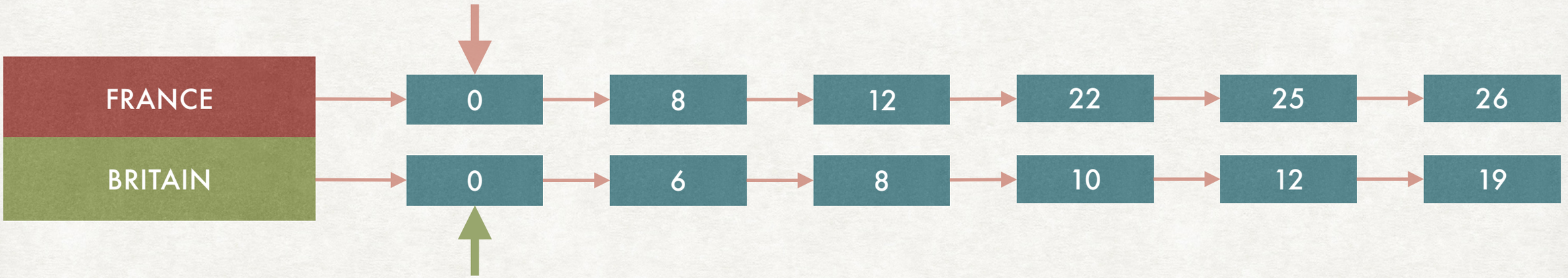


ANSWER

ANSWERING AN "AND" QUERY

NOW WITH TWO WORDS

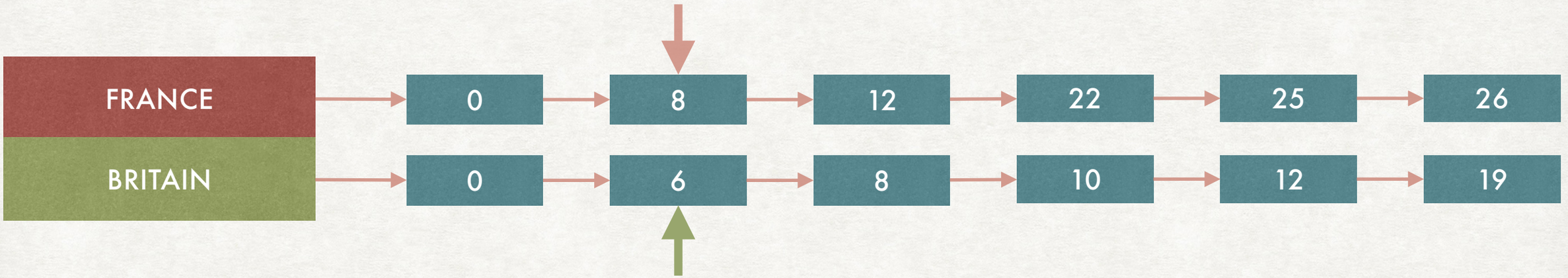
QUERY	FRANCE	**AND**	BRITAIN



ANSWERING AN "AND" QUERY

NOW WITH TWO WORDS

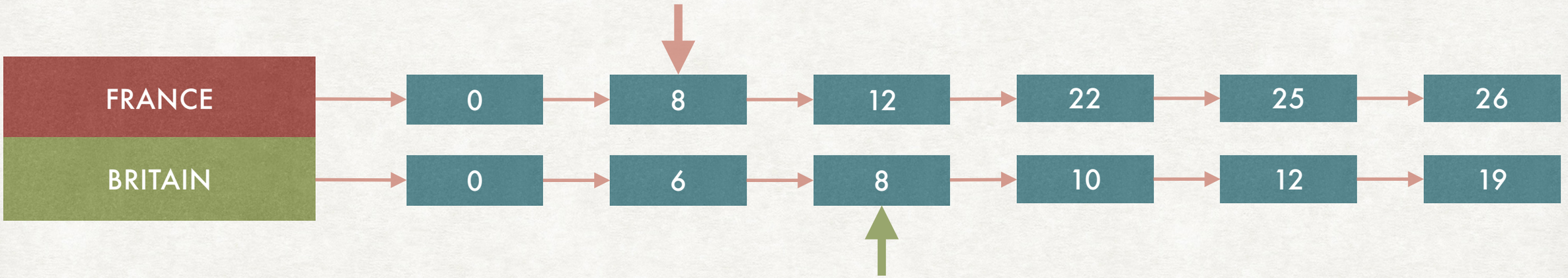
QUERY	FRANCE	**AND**	BRITAIN



ANSWERING AN "AND" QUERY

NOW WITH TWO WORDS

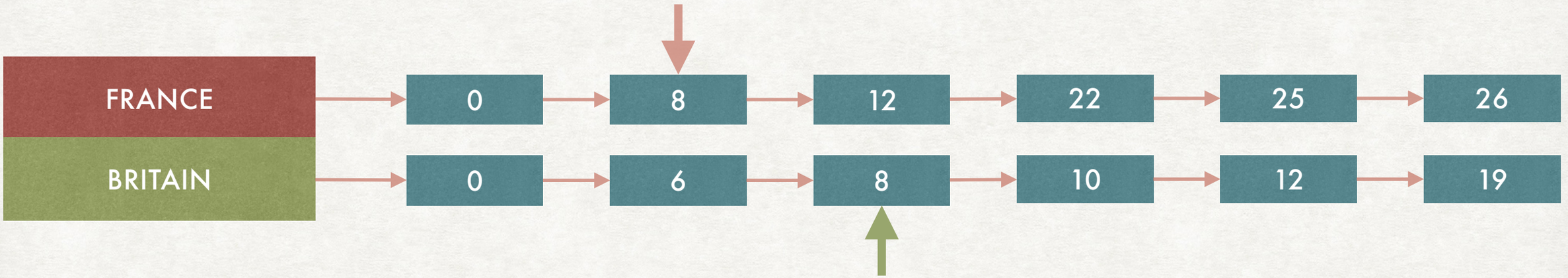
QUERY	FRANCE	**AND**	BRITAIN

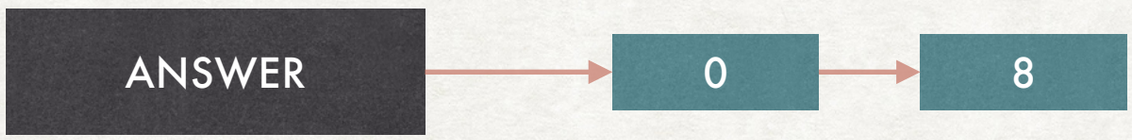


ANSWERING AN "AND" QUERY

NOW WITH TWO WORDS

QUERY	FRANCE	**AND**	BRITAIN

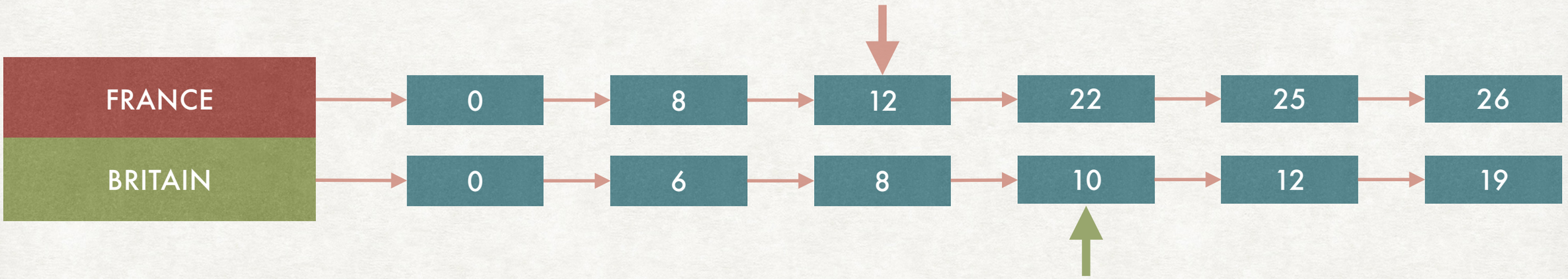


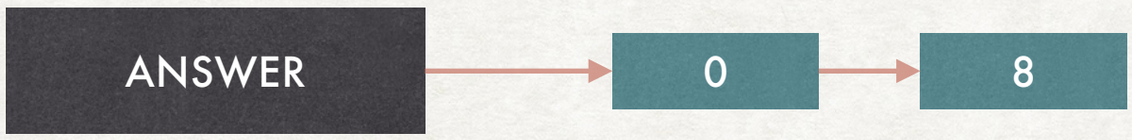


ANSWERING AN "AND" QUERY

NOW WITH TWO WORDS

QUERY	FRANCE	**AND**	BRITAIN

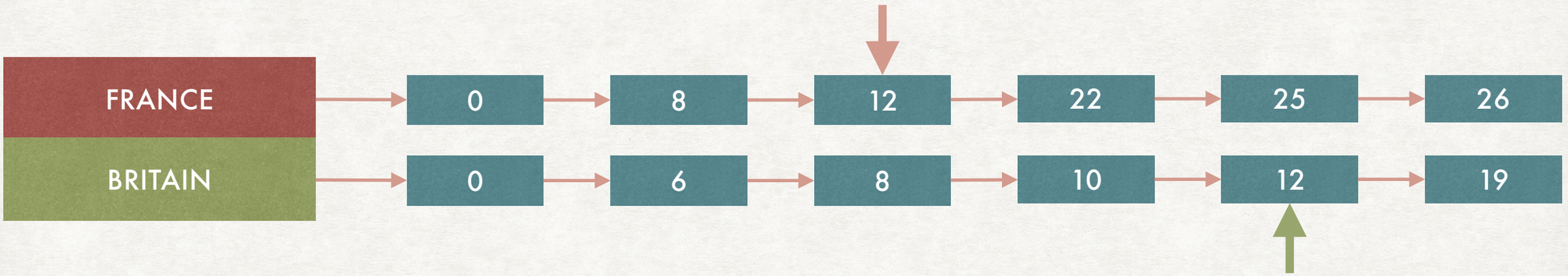


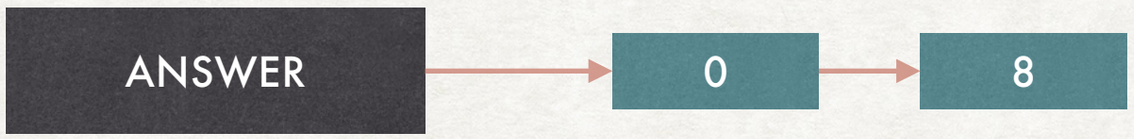


ANSWERING AN "AND" QUERY

NOW WITH TWO WORDS

QUERY	FRANCE	**AND**	BRITAIN

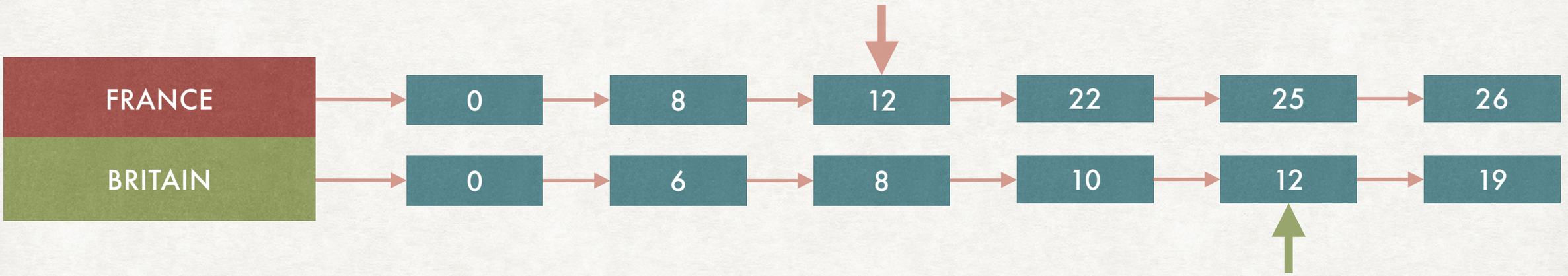




ANSWERING AN "AND" QUERY

NOW WITH TWO WORDS

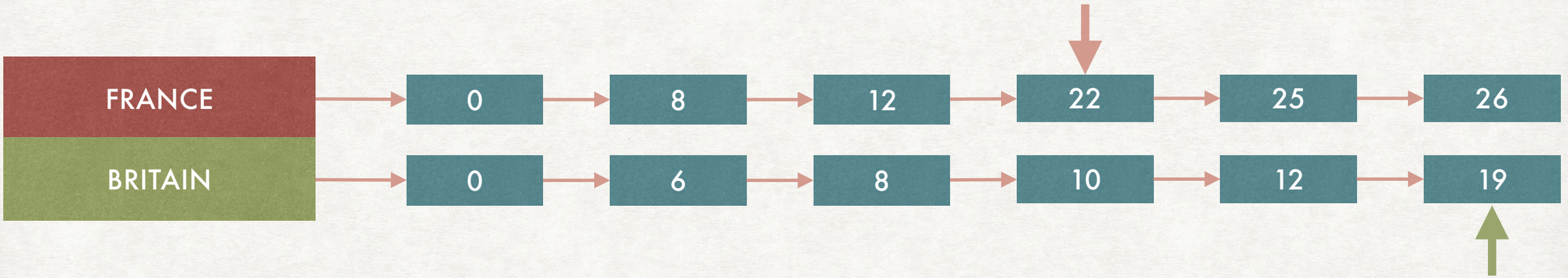
QUERY	FRANCE	**AND**	BRITAIN



ANSWERING AN "AND" QUERY

NOW WITH TWO WORDS

QUERY	FRANCE	**AND**	BRITAIN



Complexity: linear in the lengths of the lists

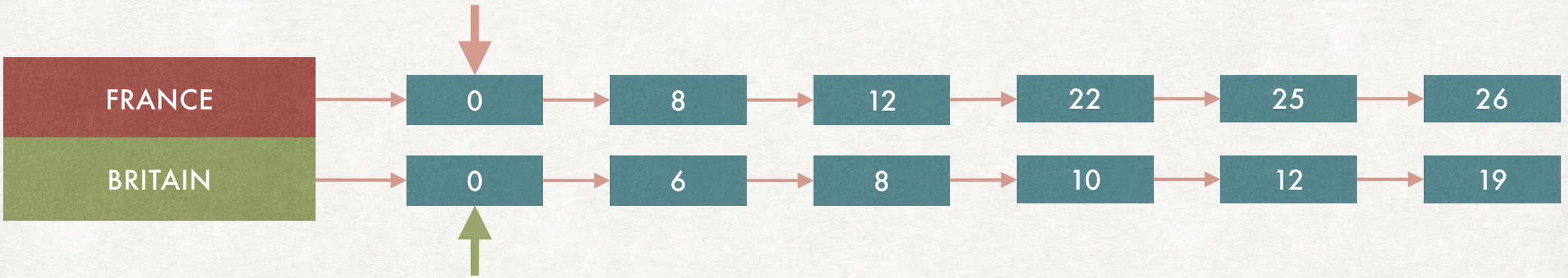
Size of the answer ≤ minimum of the lengths of the lists

ANSWERING A "OR" QUERY

WITHOUT DUPLICATES!

QUERY	FRANCE	OR	BRITAIN

We still need to compare the two lists of documents

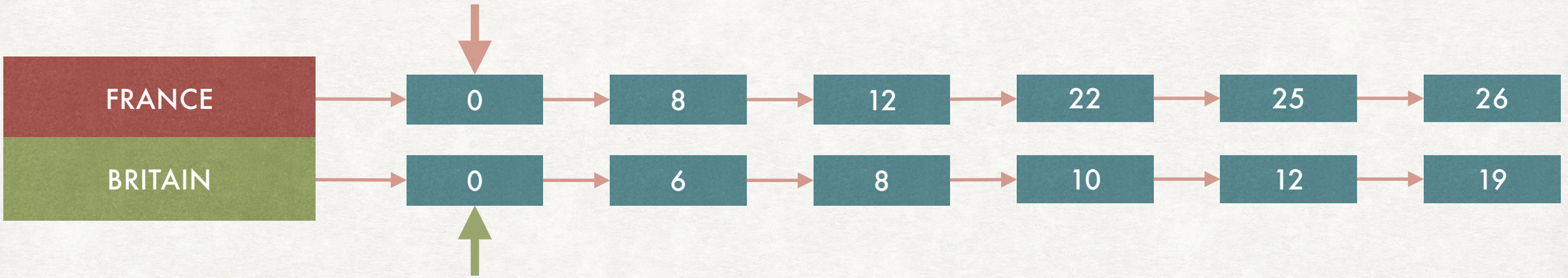


ANSWER

ANSWERING A "OR" QUERY

WITHOUT DUPLICATES!

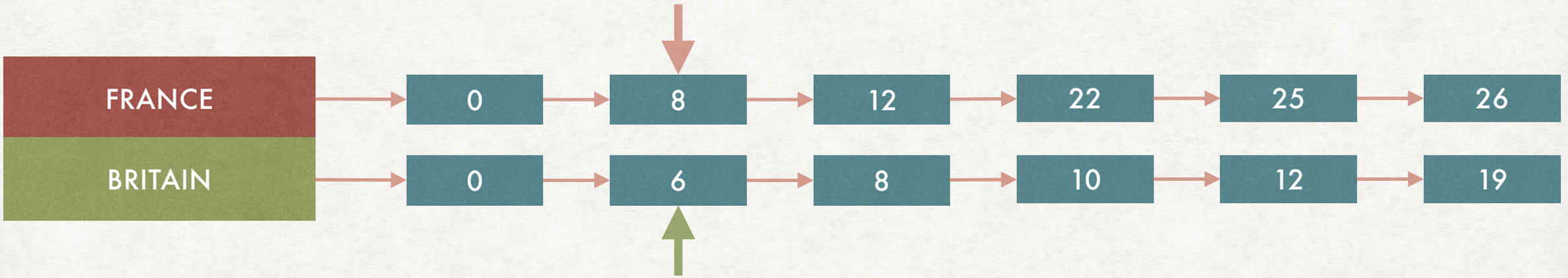
QUERY	FRANCE	OR	BRITAIN

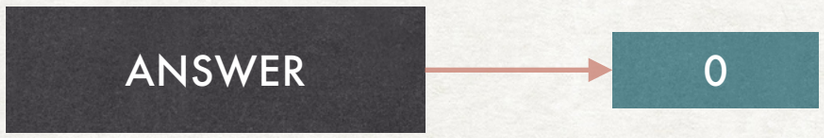


ANSWERING A "OR" QUERY

WITHOUT DUPLICATES!

QUERY	FRANCE	OR	BRITAIN

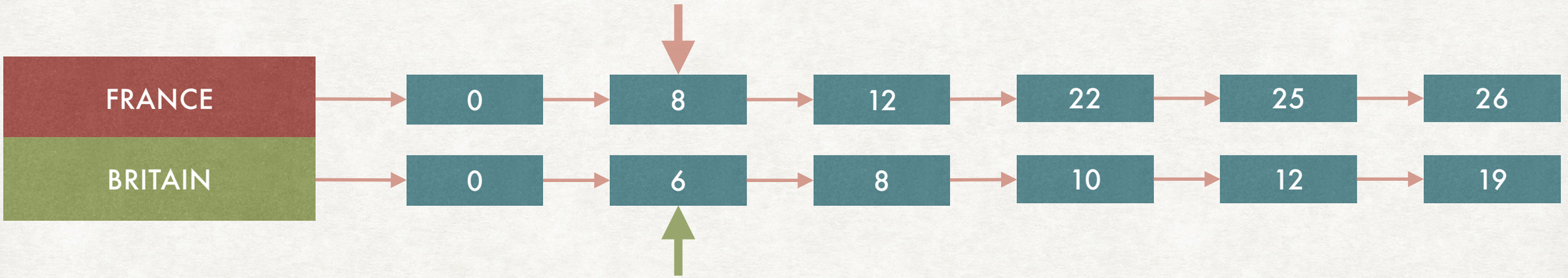


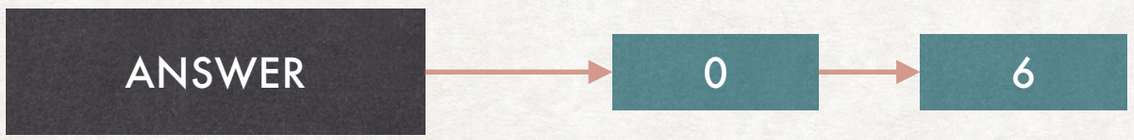


ANSWERING A "OR" QUERY

WITHOUT DUPLICATES!

QUERY	FRANCE	OR	BRITAIN

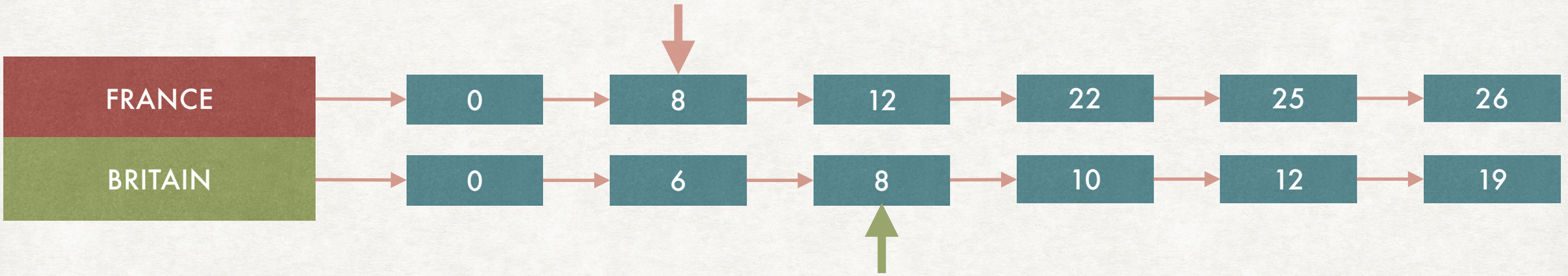


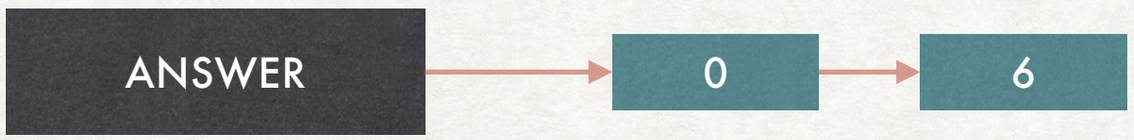


ANSWERING A "OR" QUERY

WITHOUT DUPLICATES!

QUERY	FRANCE	OR	BRITAIN

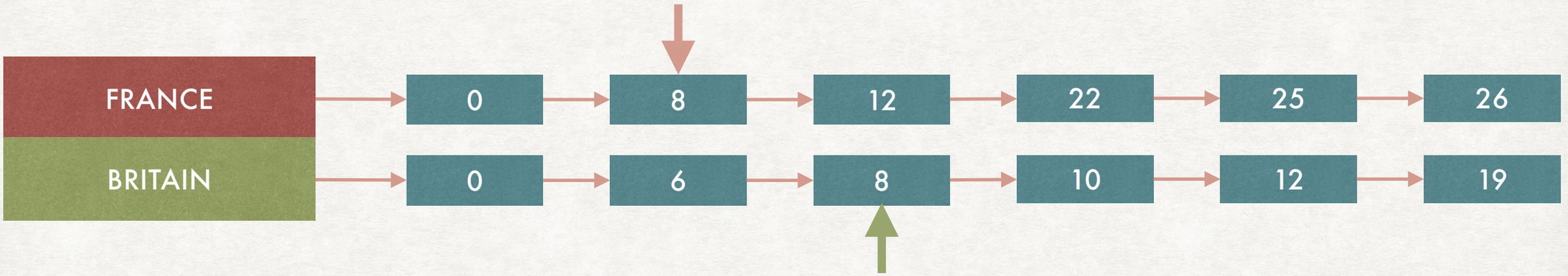




ANSWERING A "OR" QUERY

WITHOUT DUPLICATES!

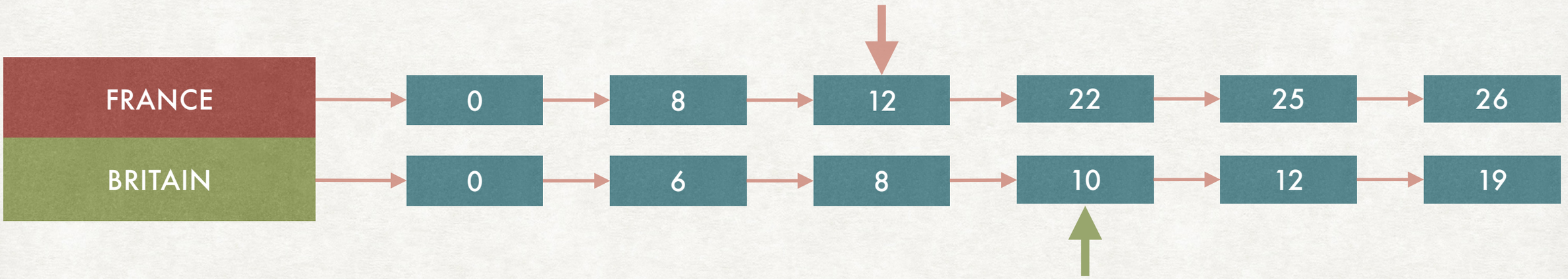
QUERY	FRANCE	OR	BRITAIN



ANSWERING A "OR" QUERY

WITHOUT DUPLICATES!

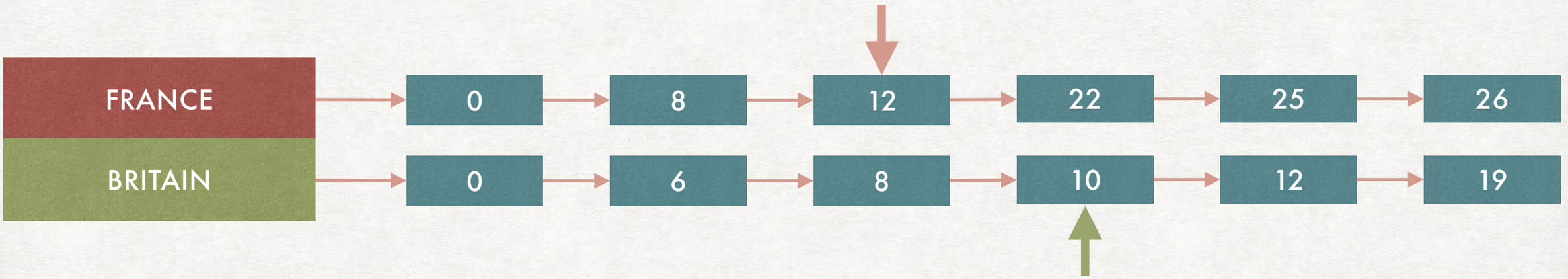
QUERY	FRANCE	OR	BRITAIN

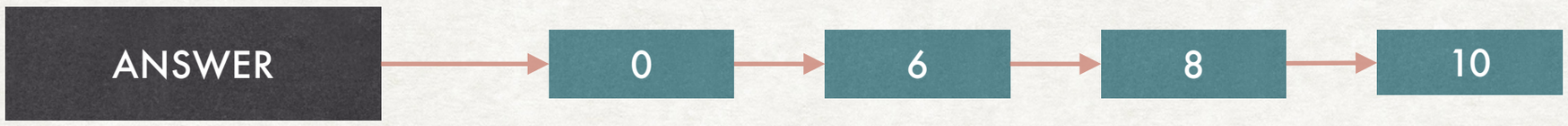


ANSWERING A "OR" QUERY

WITHOUT DUPLICATES!

QUERY	FRANCE	OR	BRITAIN

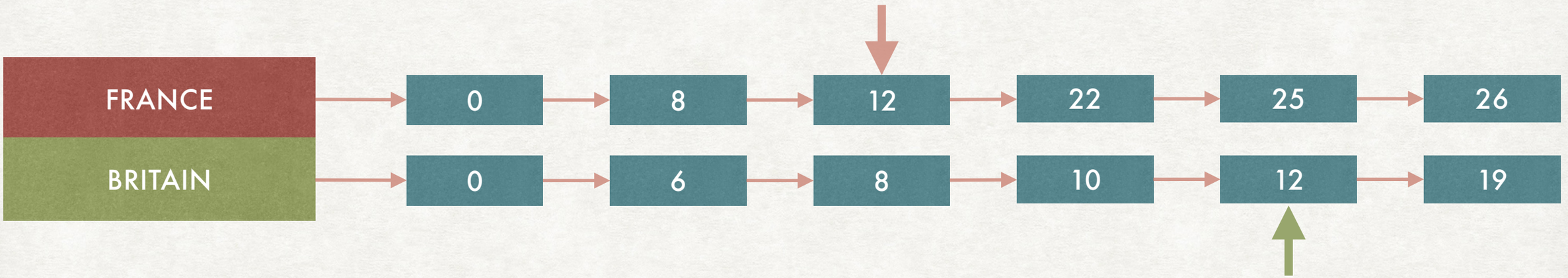


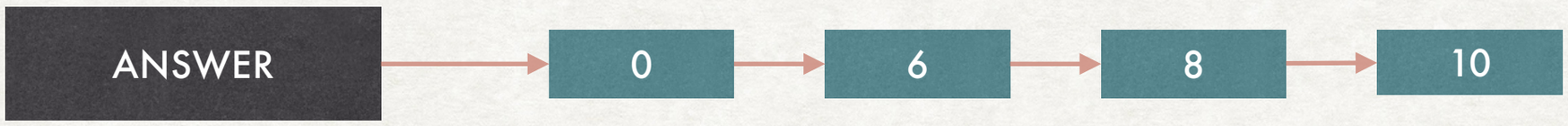


ANSWERING A "OR" QUERY

WITHOUT DUPLICATES!

QUERY	FRANCE	OR	BRITAIN

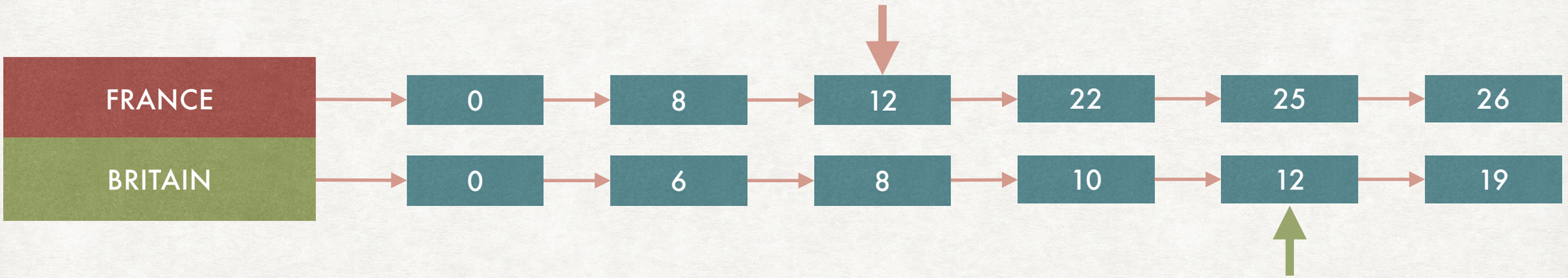




ANSWERING A "OR" QUERY

WITHOUT DUPLICATES!

QUERY	FRANCE	**OR**	BRITAIN

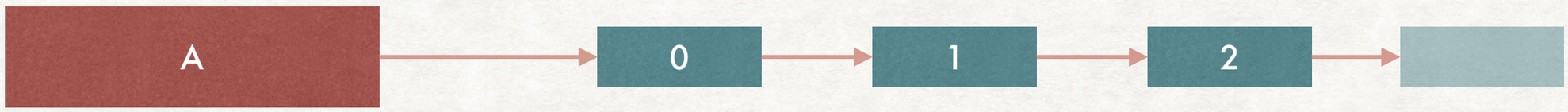


Complexity: linear in the lengths of the lists

Size of the answer ≤ 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.

<u>Example</u>

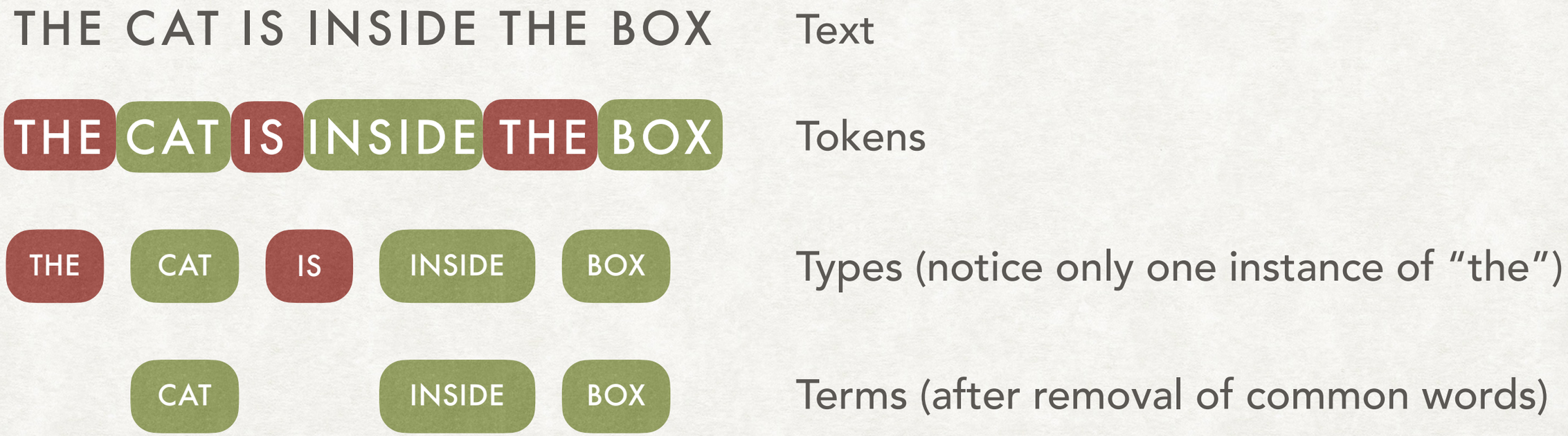
Do we really want to keep “CAR” and “CARS” separated?

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



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]

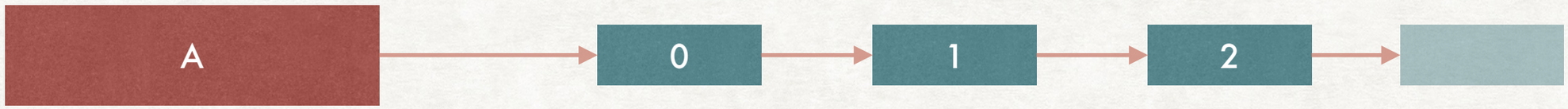
Possible (partial) solutions:

- 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:



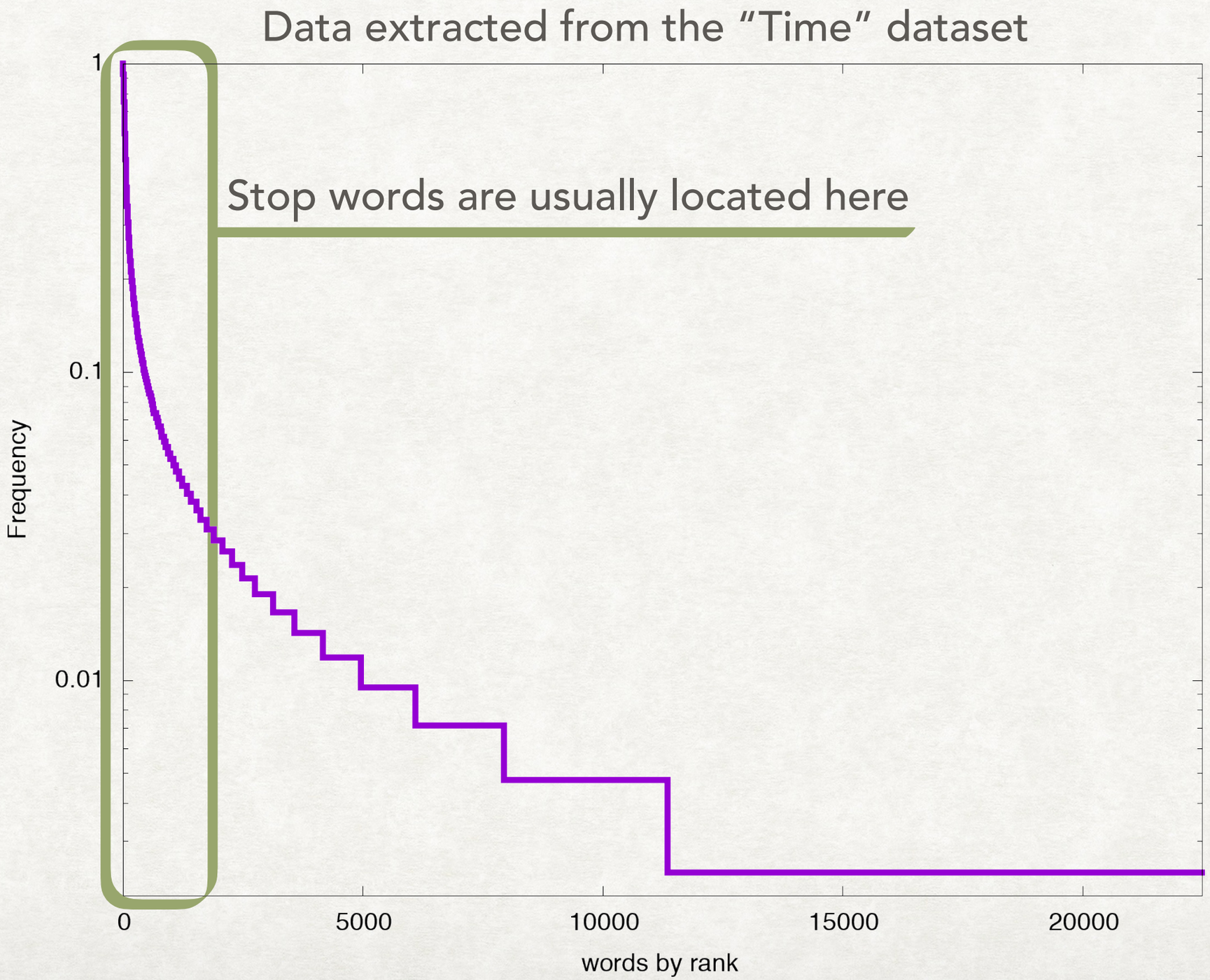
Some terms are not useful: “A” is in all the documents!

- **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



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

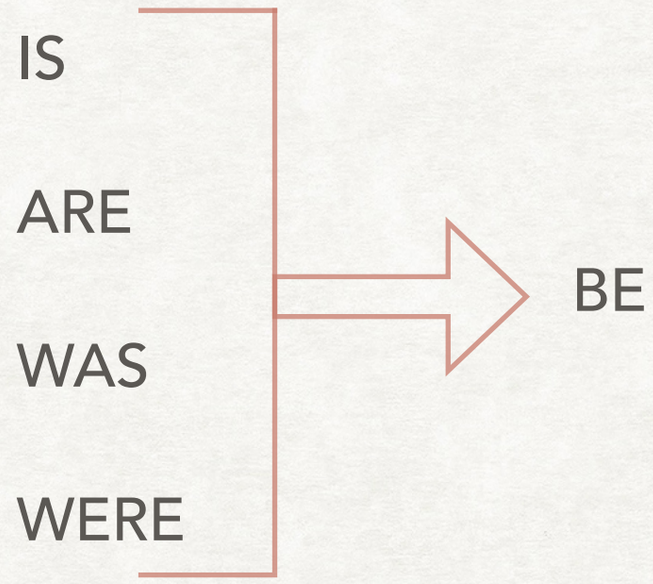
windows (can mean both the object and the OS)  Windows (the OS)

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

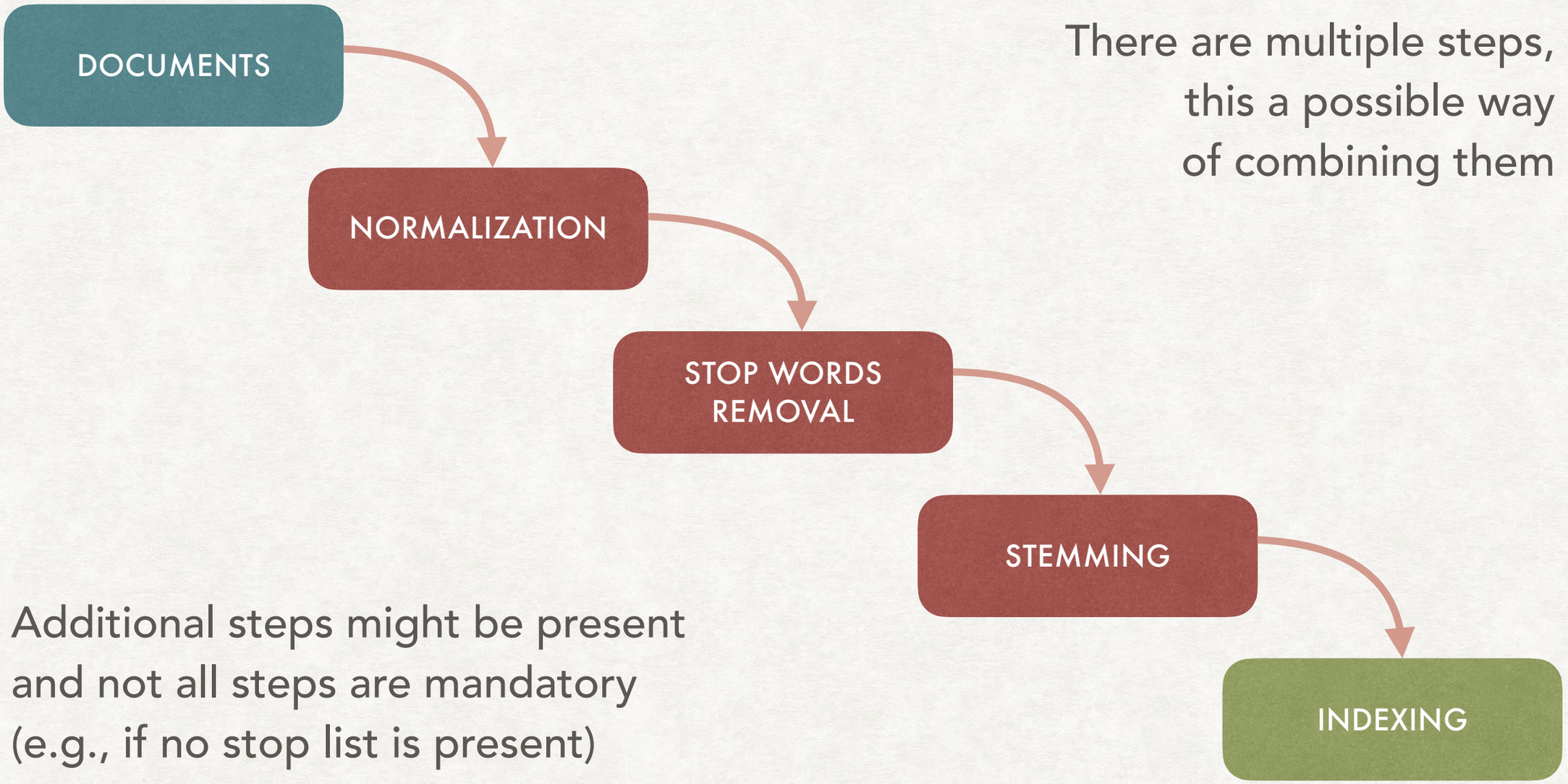
Five stages applied sequentially.

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

Rule	
SSES → SS	caressess → caress
IES → S	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



There are multiple steps,
this a possible way
of combining them

Additional steps might be present
and not all steps are mandatory
(e.g., if no stop list is present)

ANSWERING PHRASE QUERIES

OUR GOAL

EXTENDING THE QUERY LANGUAGE

- We want to be able to ask queries consisting of multiple consecutive words:
  - “calico cat”
  - “University of Trieste”
- A common syntax for this kind of queries is to enclose the words in double quotes.
- Two approaches shown: *biword indexes* and *positional indexes*.

BIWORD INDEXES

WORKING ON PAIRS OF WORDS

THE CAT IS INSIDE THE BOX — Text

THE CAT | CAT IS | IS INSIDE | INSIDE THE | THE BOX — Terms

- The terms are pairs of words
- Queries need to be “rewritten”:

“inside the box” → “inside the” AND “the box”

BIWORD INDEXES

POSSIBLE PROBLEMS

Text: INSIDE THE HOUSE THERE IS THE BOX

Original Query: “inside the box” No Match

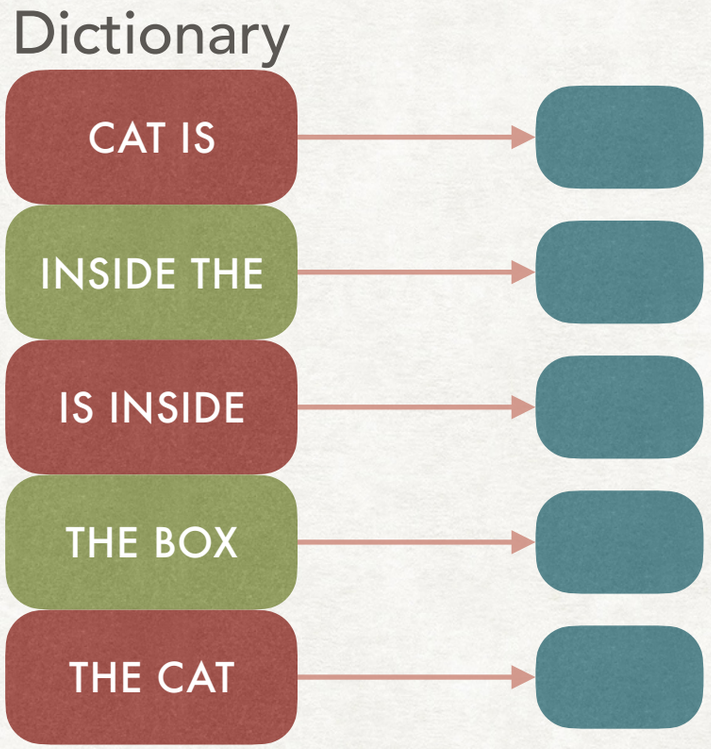
Rewritten Query: “inside the” AND “the box” Match

Rewriting the query might generate false positives
(but it works quite well in practice)

BIWORD INDEXES

POSSIBLE PROBLEMS

To answer the query
we would need to find
all terms containing “cat”



We also need an index of single-word terms!

BIWORD INDEXES

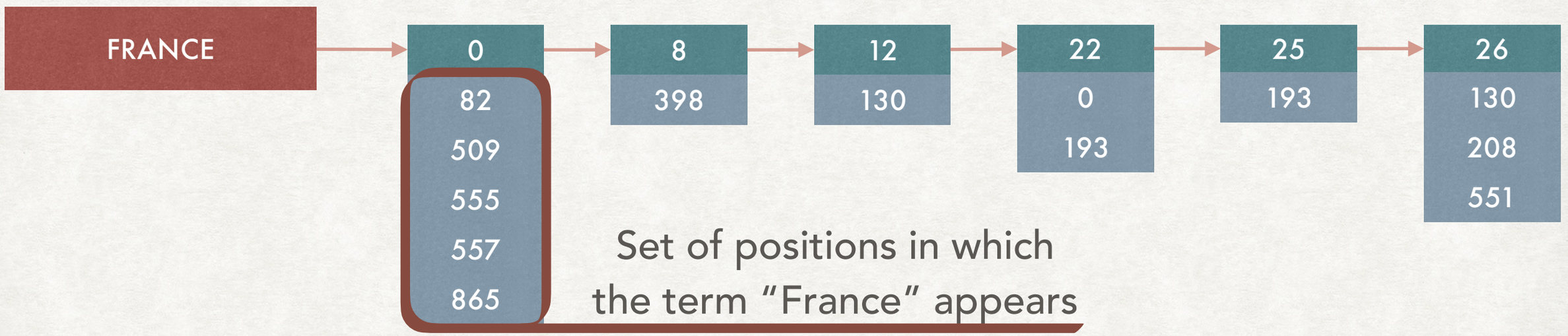
EXTENSIONS AND FURTHER OBSTACLES

- The idea of using pair of words as terms can be extended to any length, reducing the risk of false positives…
- …but increasing the amount of space needed.
- If the number of words in a term is variable it is called *phrase index*.
- It is also possible to “tag” the part of speech (i.e., names, verbs, articles, prepositions, etc.) to add pairs of names separated by articles and prepositions to the index.
  - E.g., in “door at the entrance”, “door entrance” is considered a term

POSITIONAL INDEXES

ADDING POSITIONS TO THE POSTINGS

One way to answer a phrase query is to add, for each posting, the set of positions in which the term appear in the document.

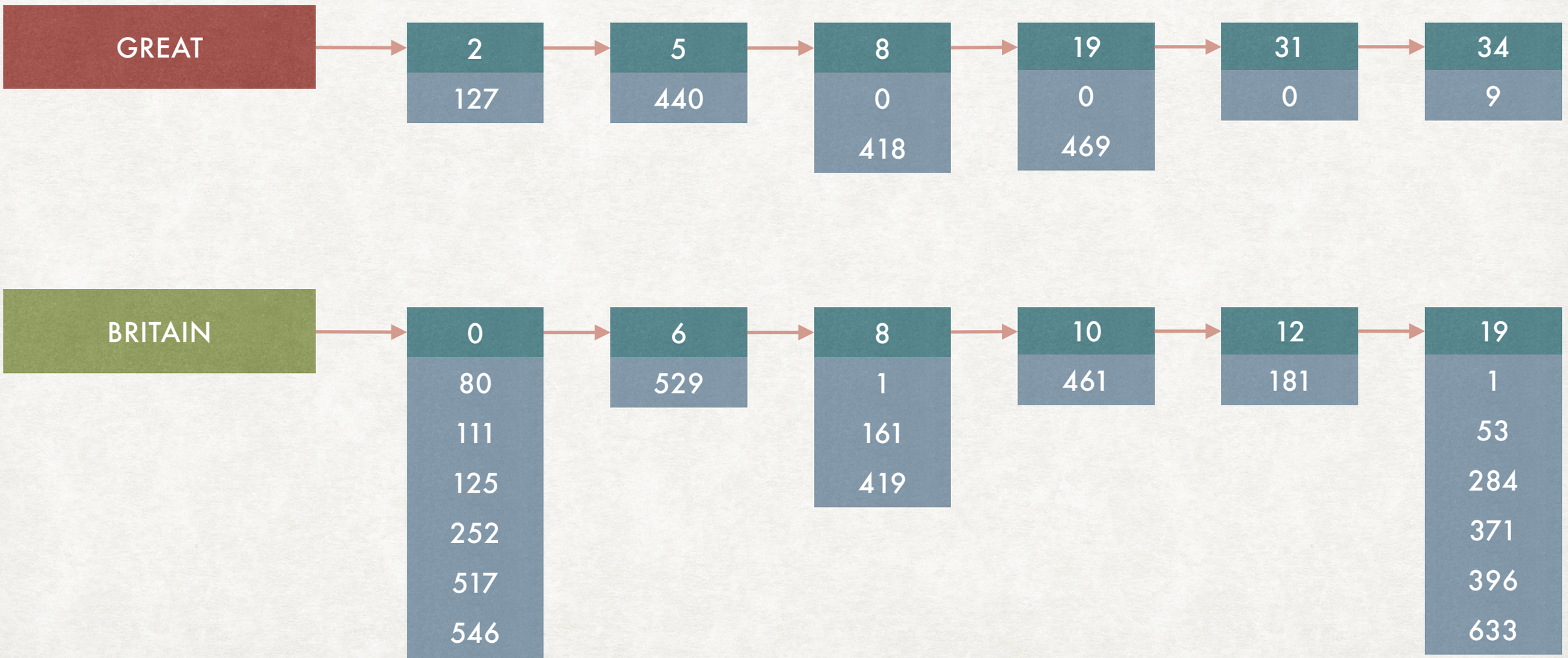


ANSWERING A PHRASE QUERY

WITH POSITIONAL INDEXING

QUERY	"GREAT BRITAIN"

ANSWER

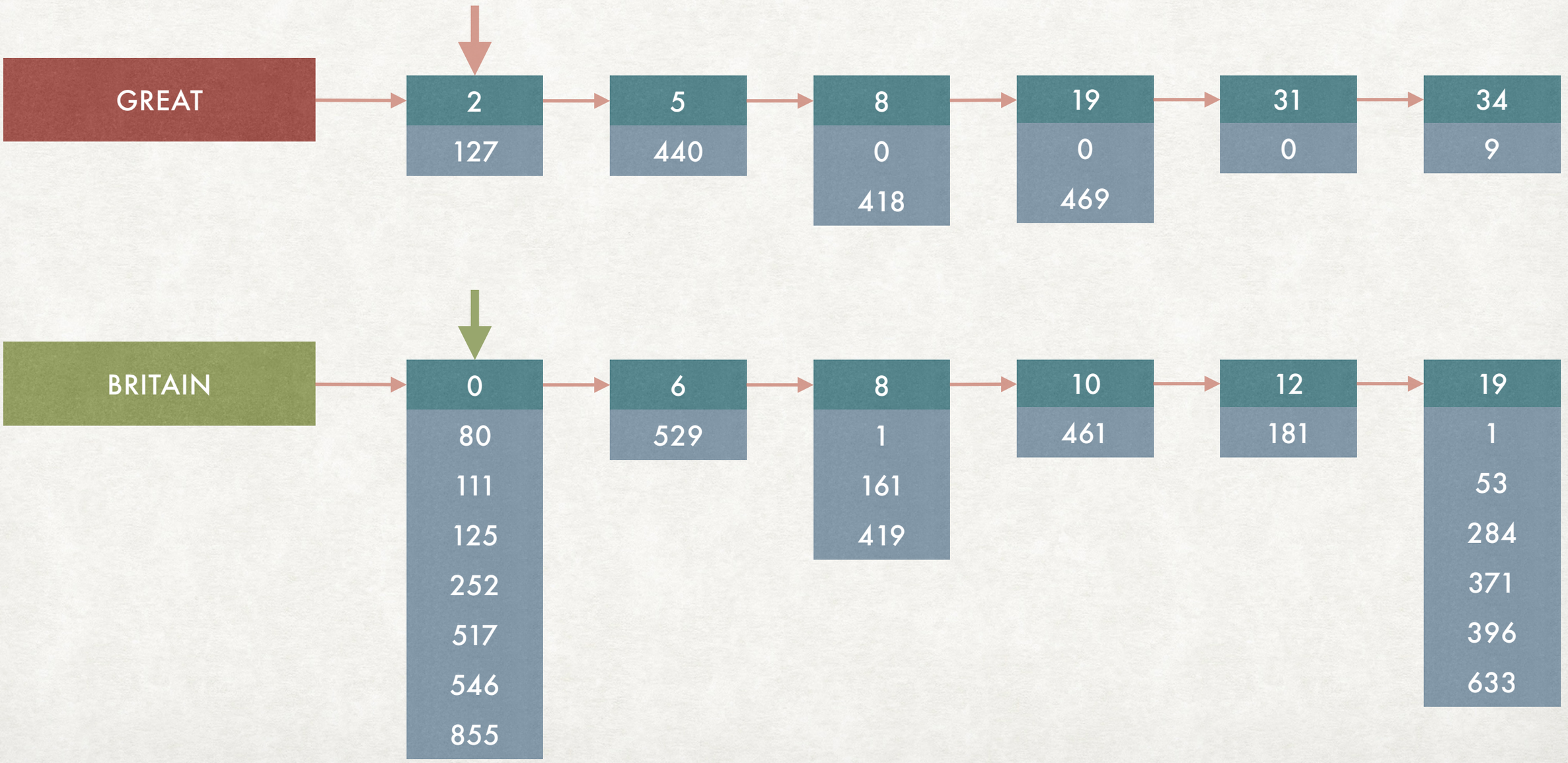


ANSWERING A PHRASE QUERY

WITH POSITIONAL INDEXING

QUERY | "GREAT BRITAIN"

ANSWER

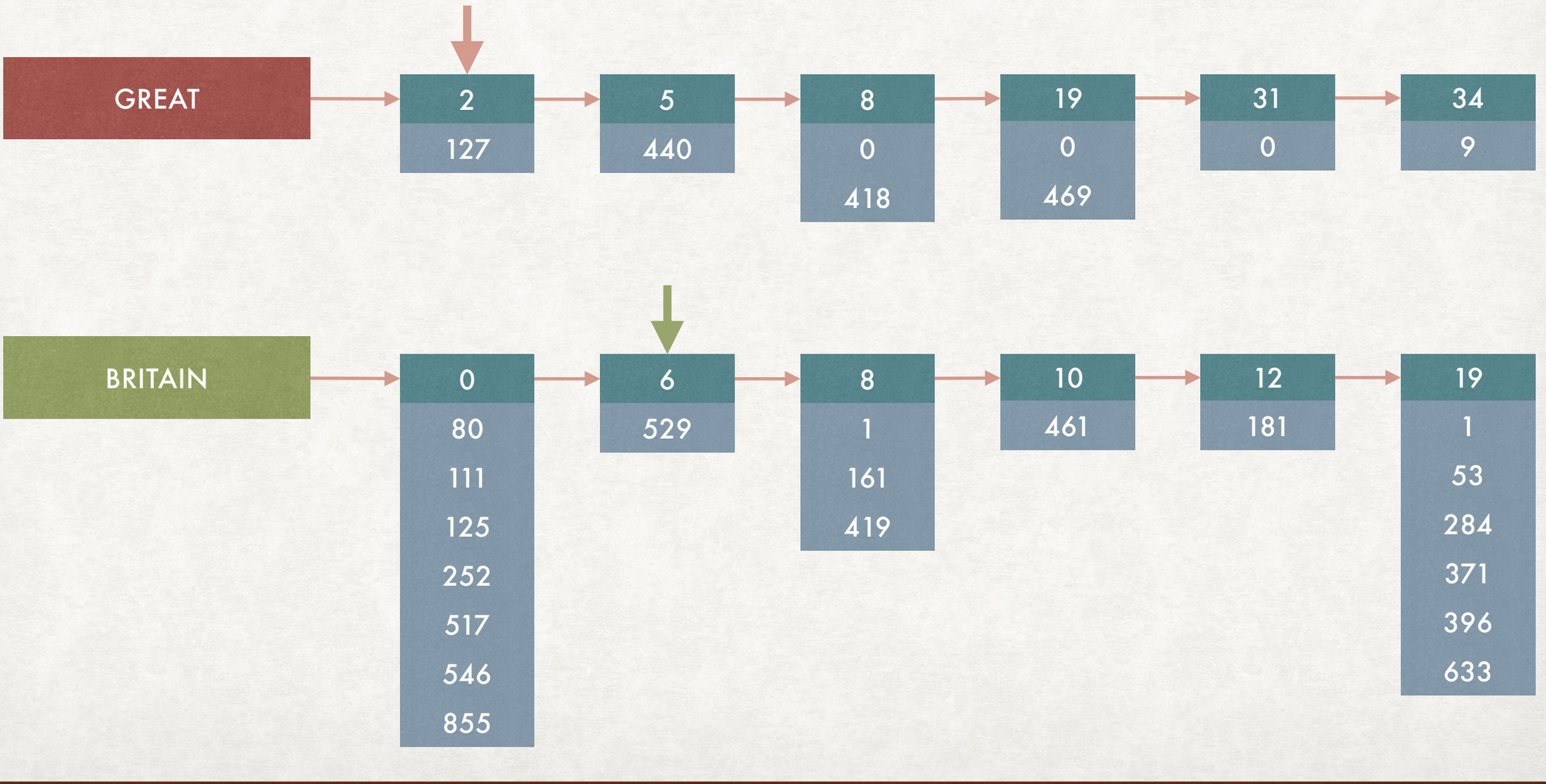


ANSWERING A PHRASE QUERY

WITH POSITIONAL INDEXING

QUERY | "GREAT BRITAIN"

ANSWER

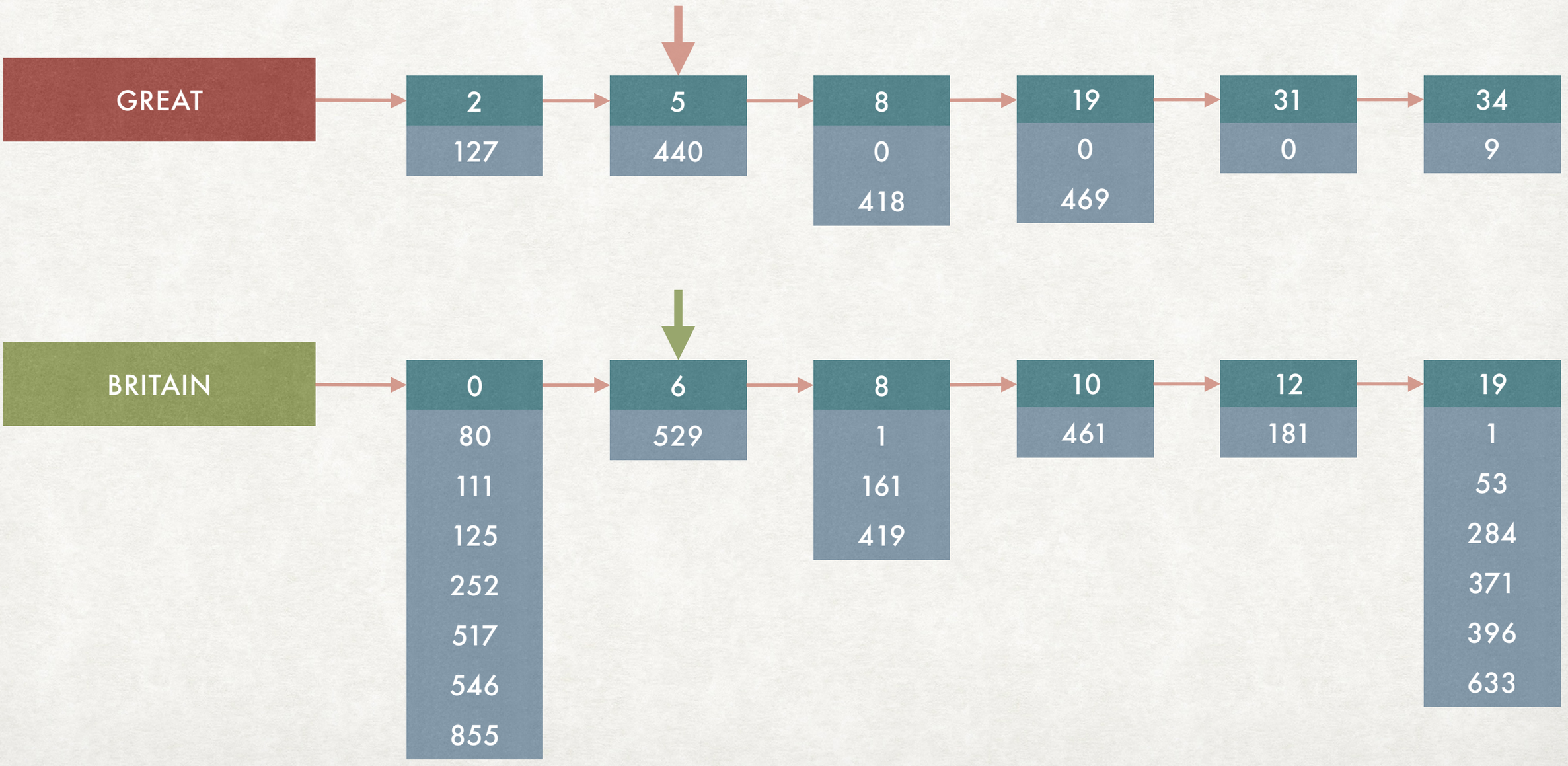


ANSWERING A PHRASE QUERY

WITH POSITIONAL INDEXING

QUERY | "GREAT BRITAIN"

ANSWER

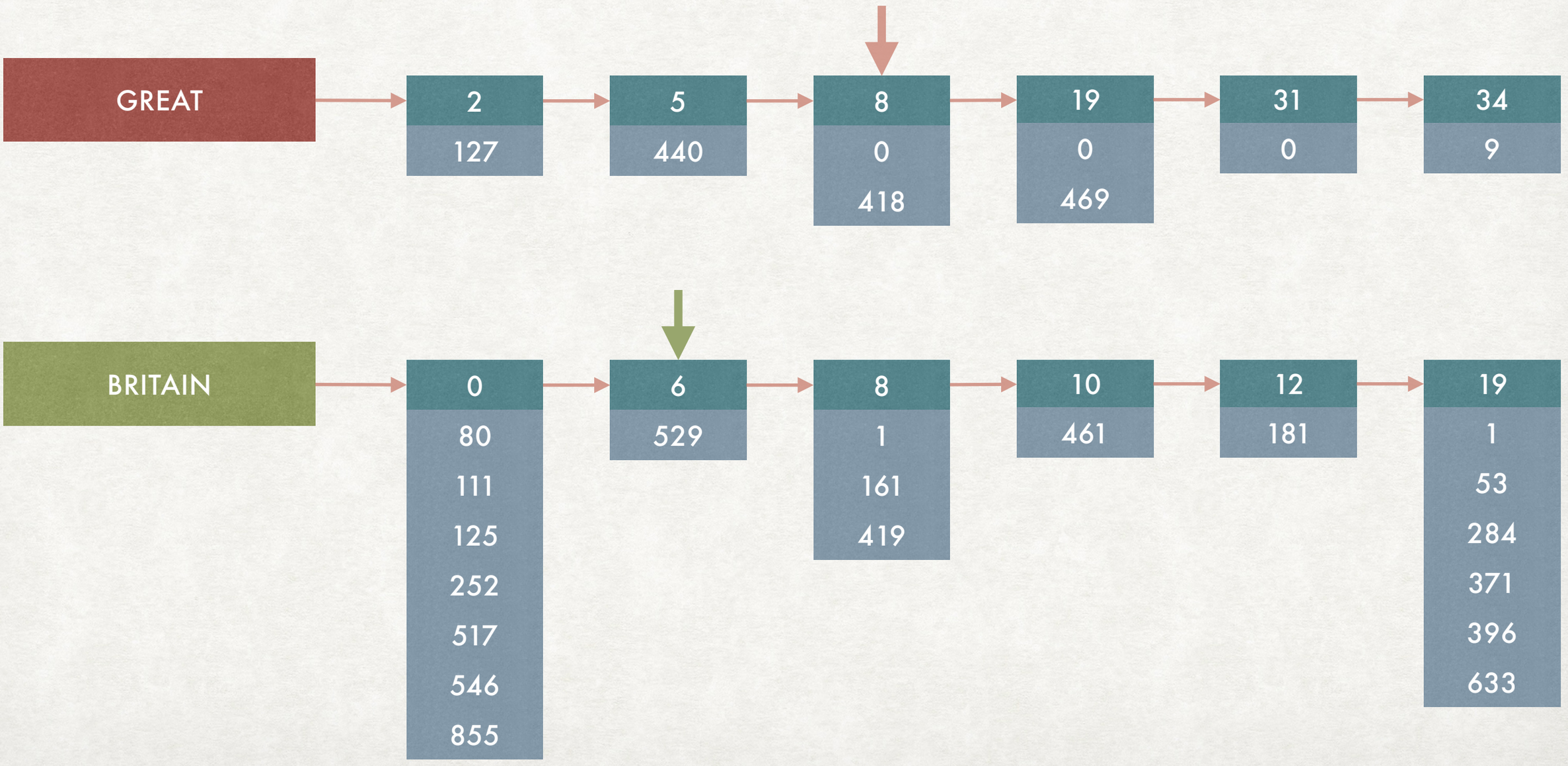


ANSWERING A PHRASE QUERY

WITH POSITIONAL INDEXING

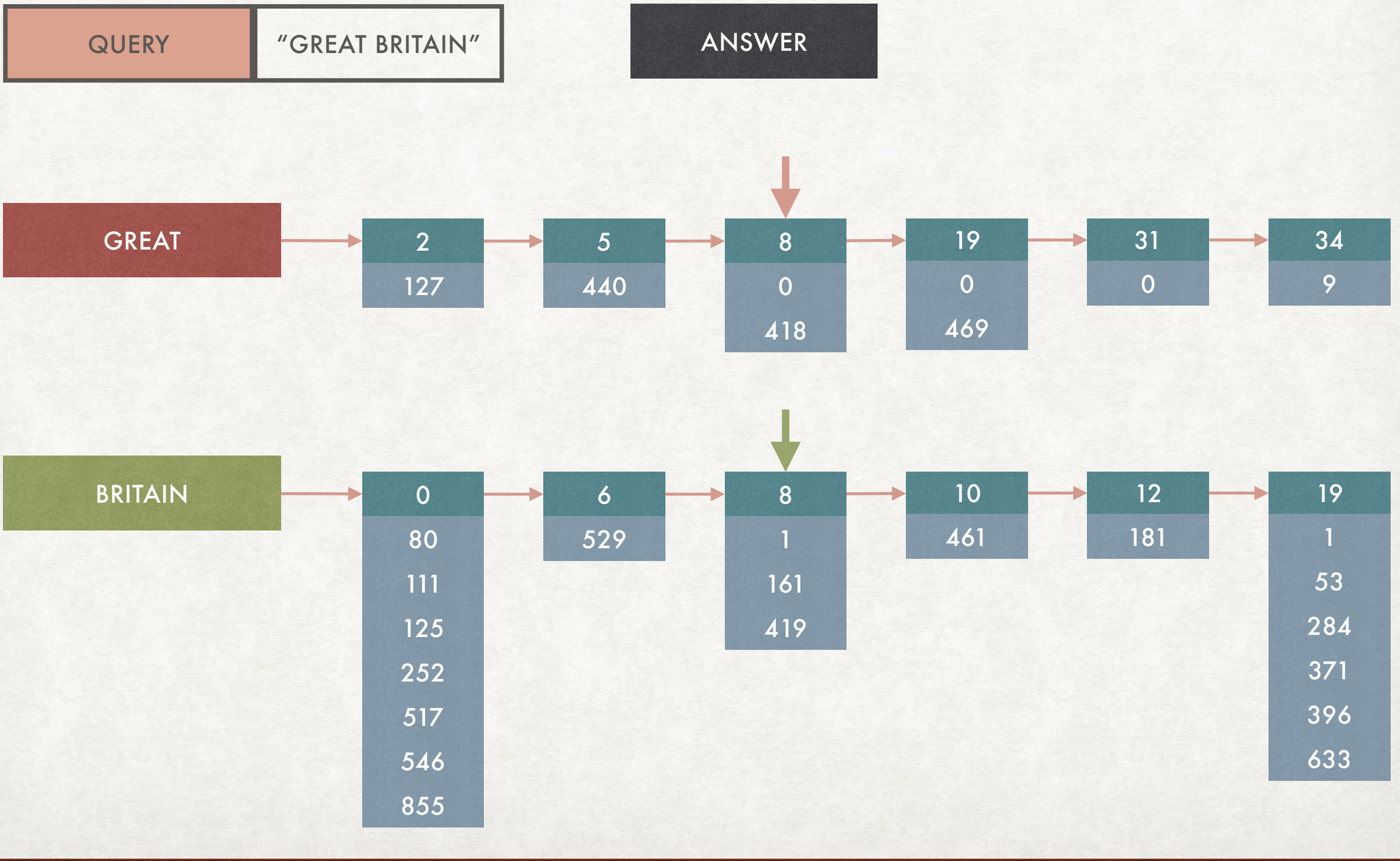
QUERY | "GREAT BRITAIN"

ANSWER



ANSWERING A PHRASE QUERY

WITH POSITIONAL INDEXING

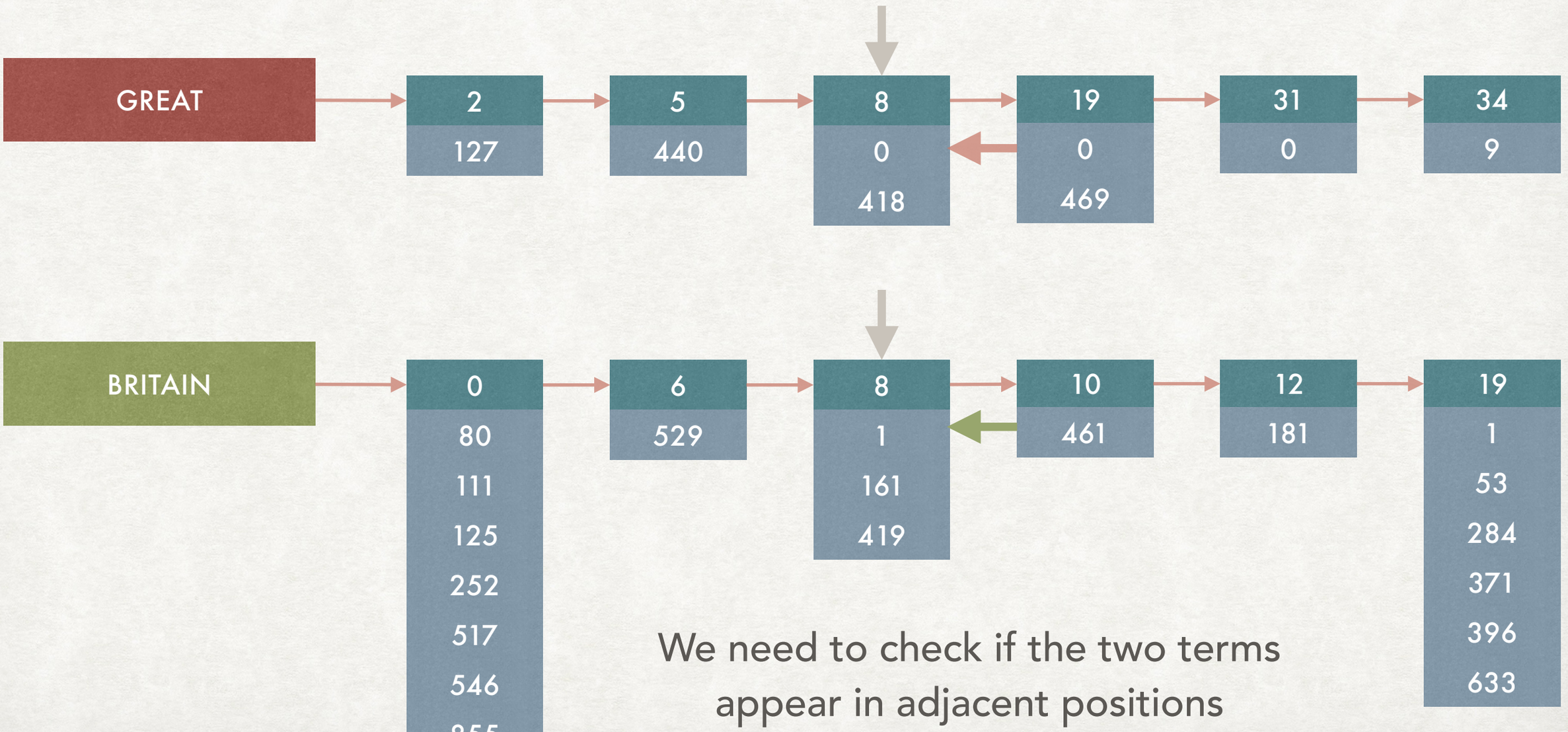


ANSWERING A PHRASE QUERY

WITH POSITIONAL INDEXING

QUERY | "GREAT BRITAIN"

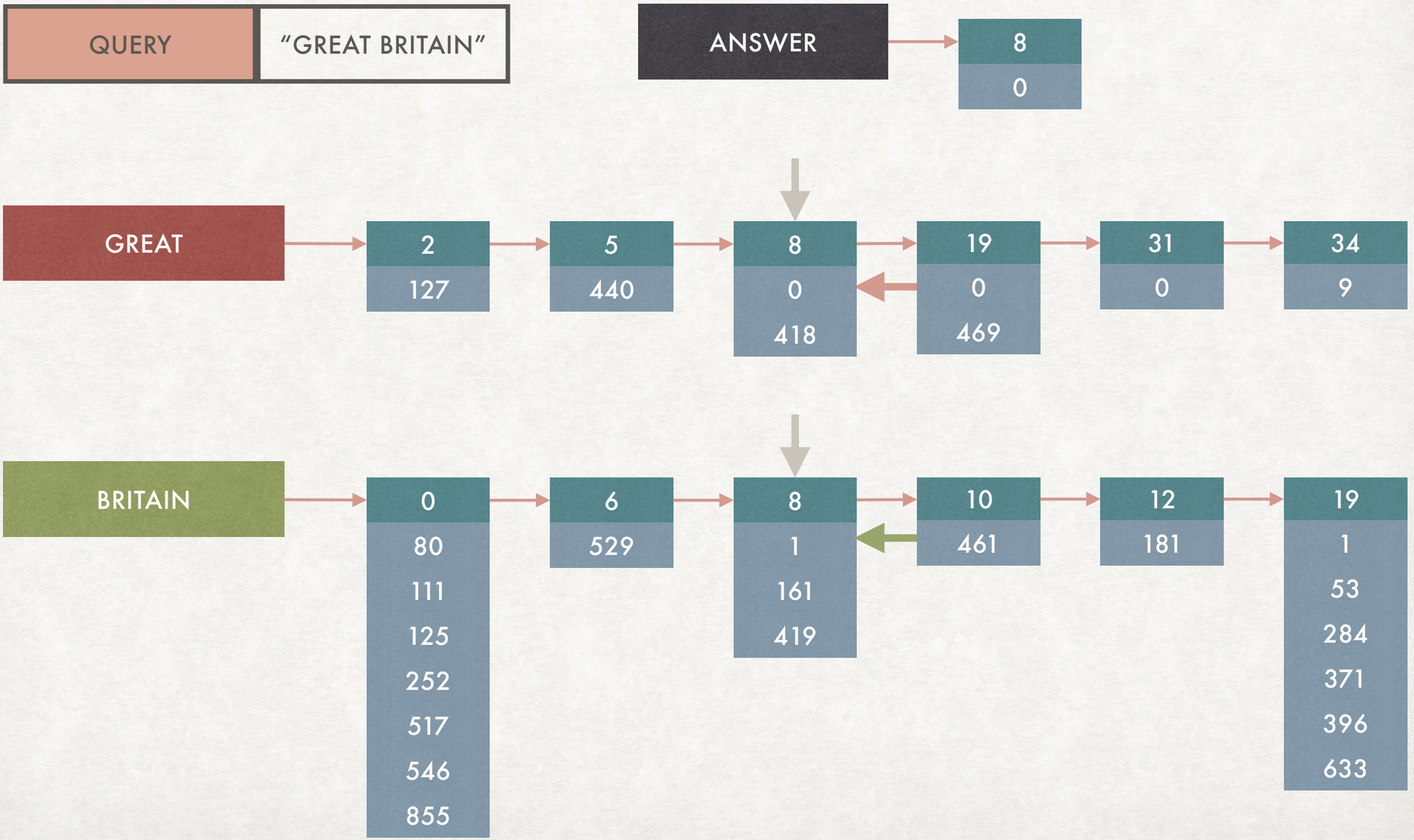
ANSWER



We need to check if the two terms appear in adjacent positions

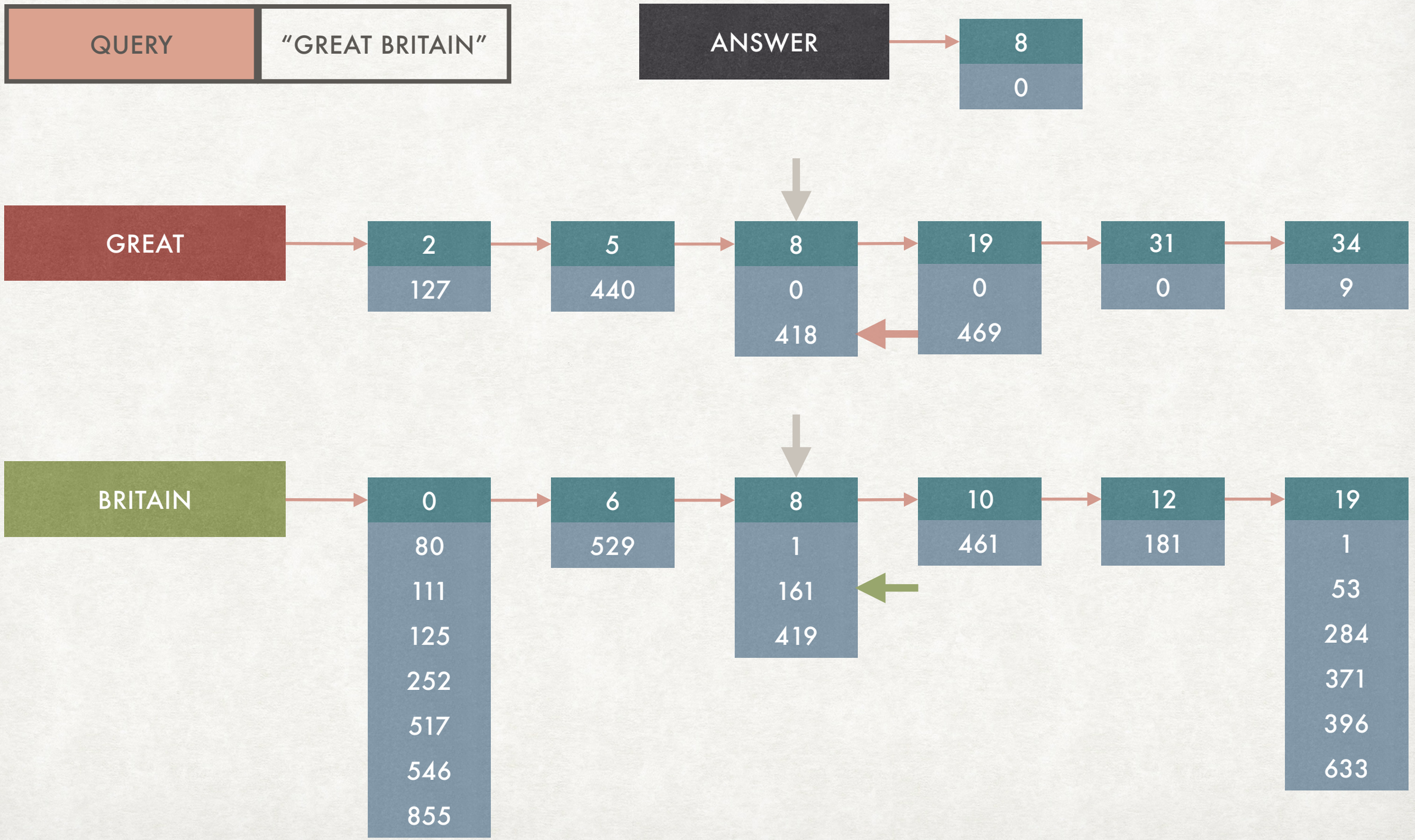
ANSWERING A PHRASE QUERY

WITH POSITIONAL INDEXING



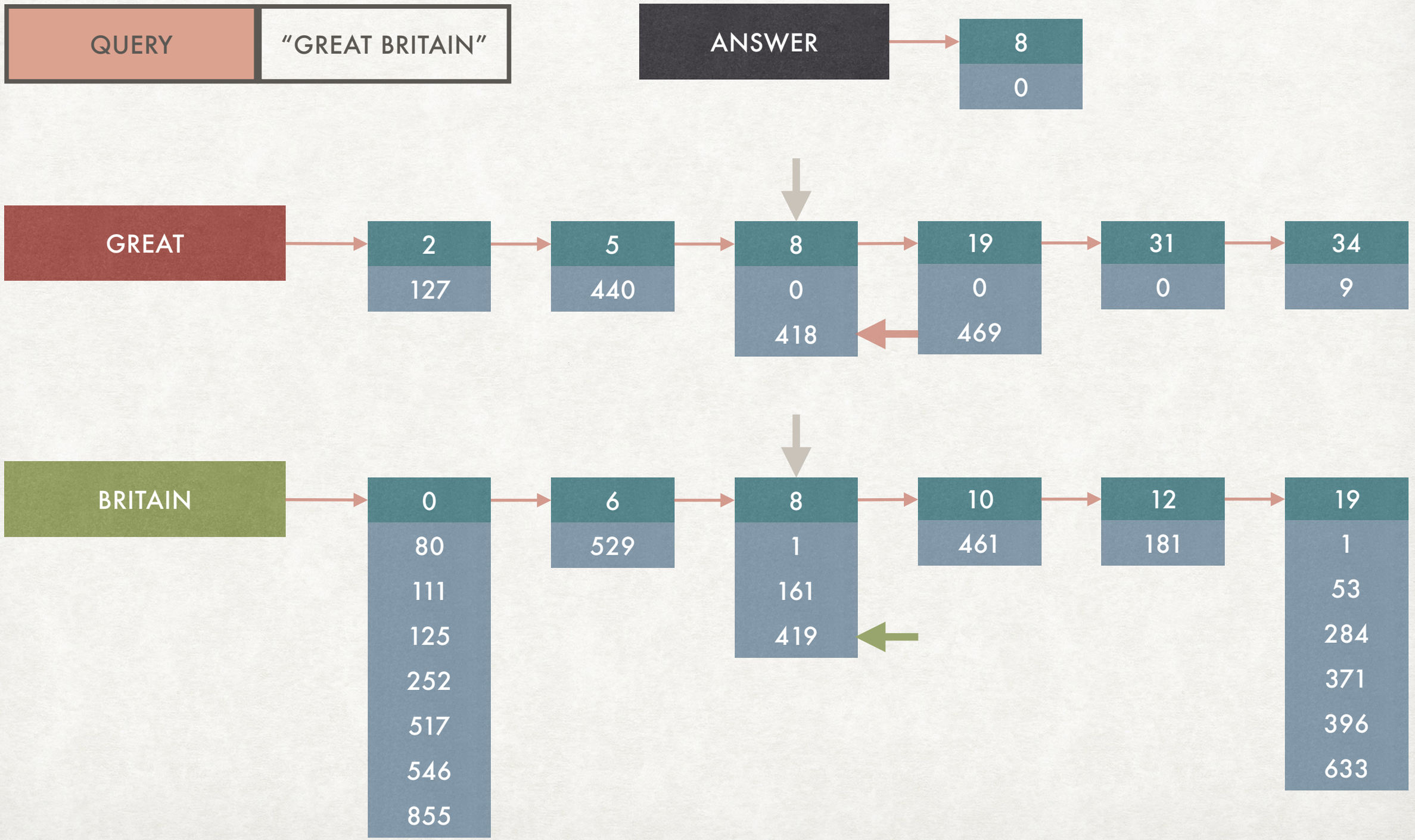
ANSWERING A PHRASE QUERY

WITH POSITIONAL INDEXING



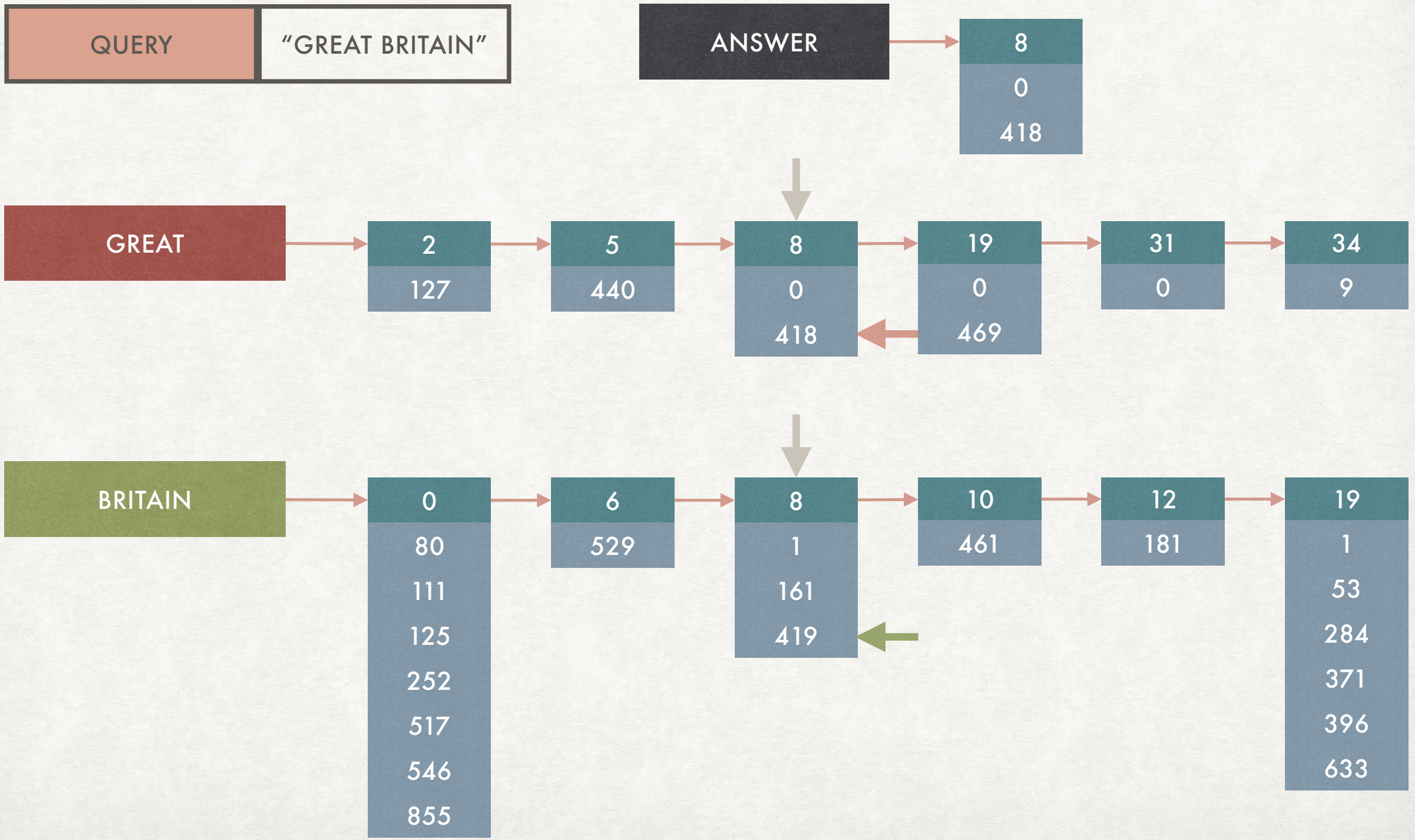
ANSWERING A PHRASE QUERY

WITH POSITIONAL INDEXING



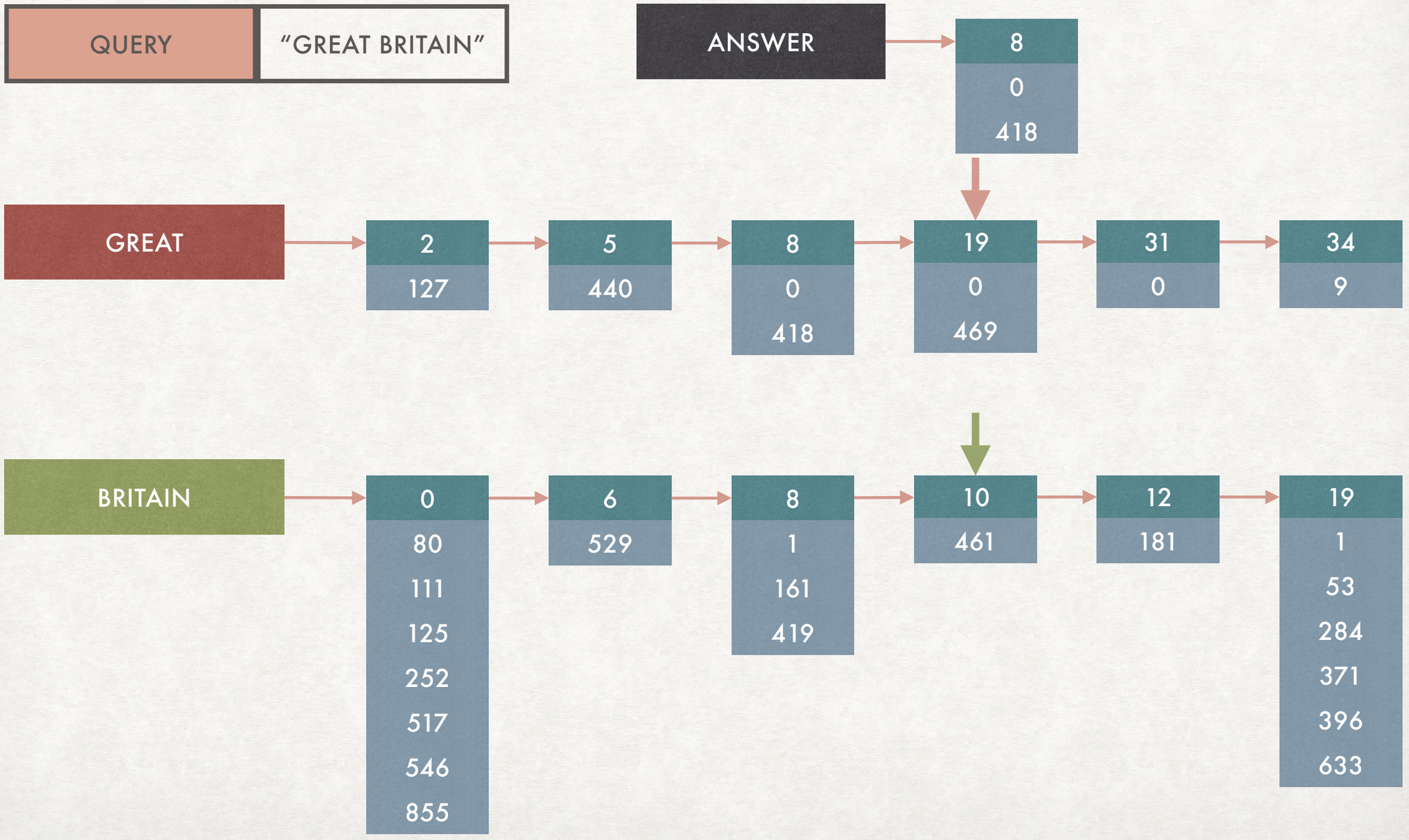
ANSWERING A PHRASE QUERY

WITH POSITIONAL INDEXING



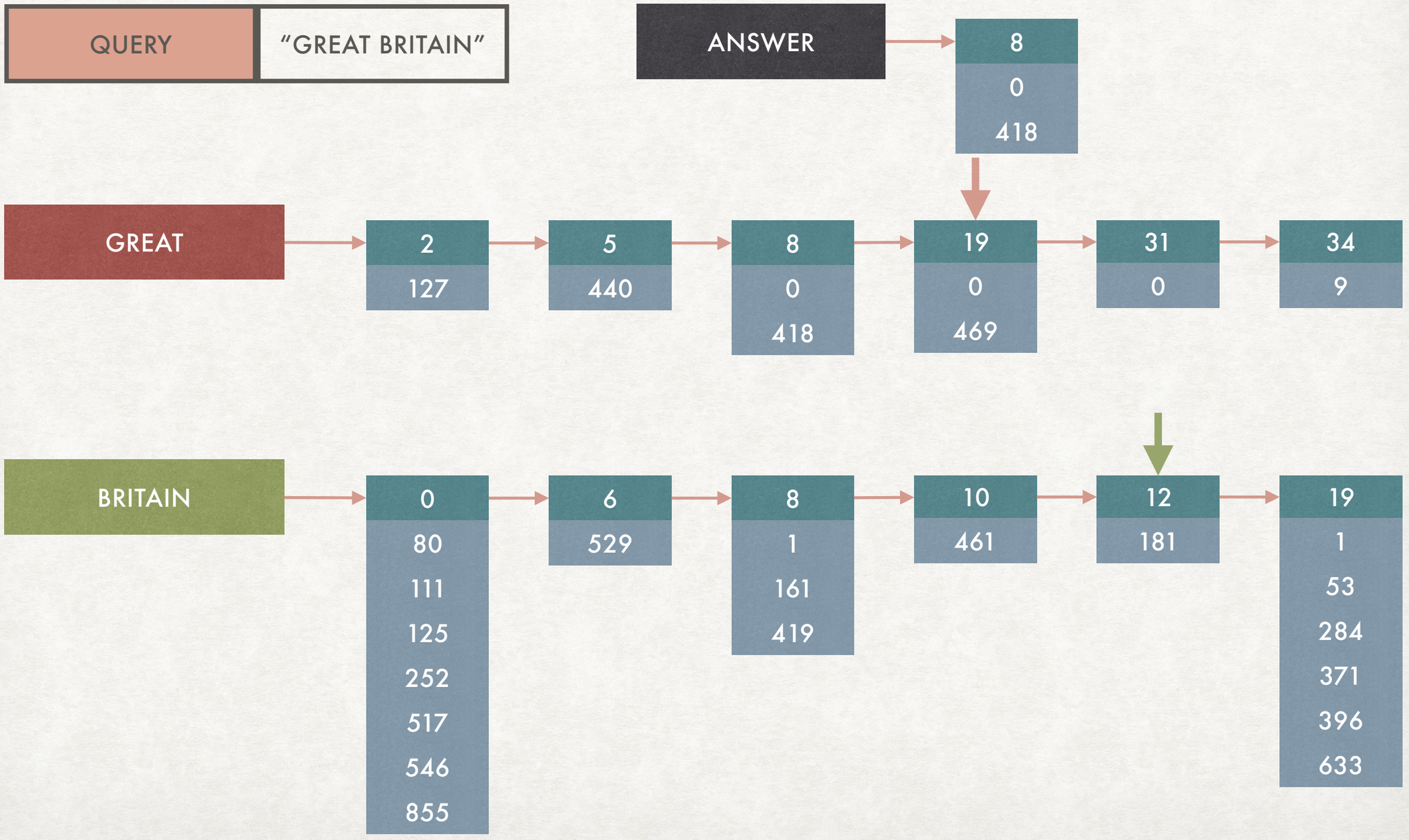
ANSWERING A PHRASE QUERY

WITH POSITIONAL INDEXING



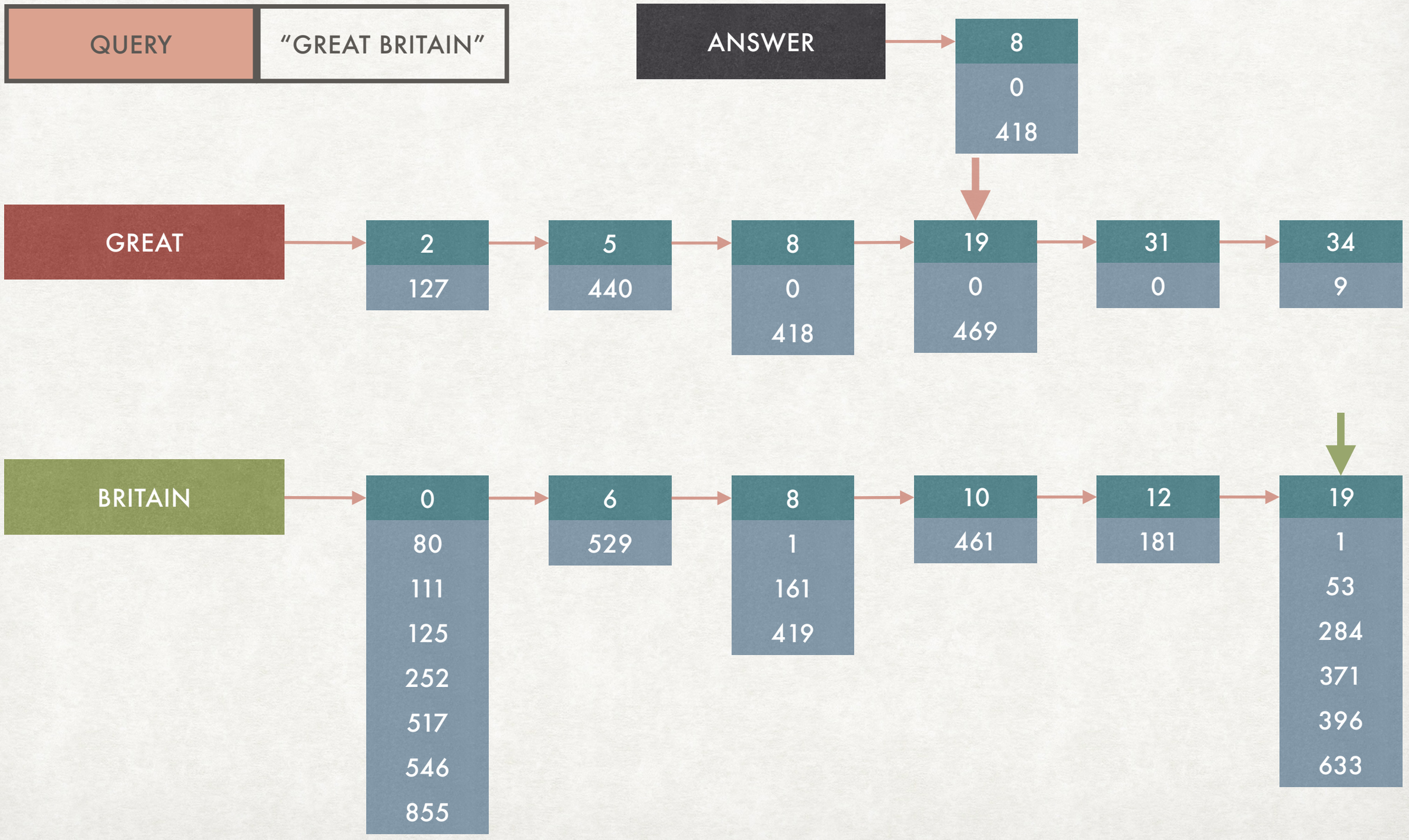
ANSWERING A PHRASE QUERY

WITH POSITIONAL INDEXING



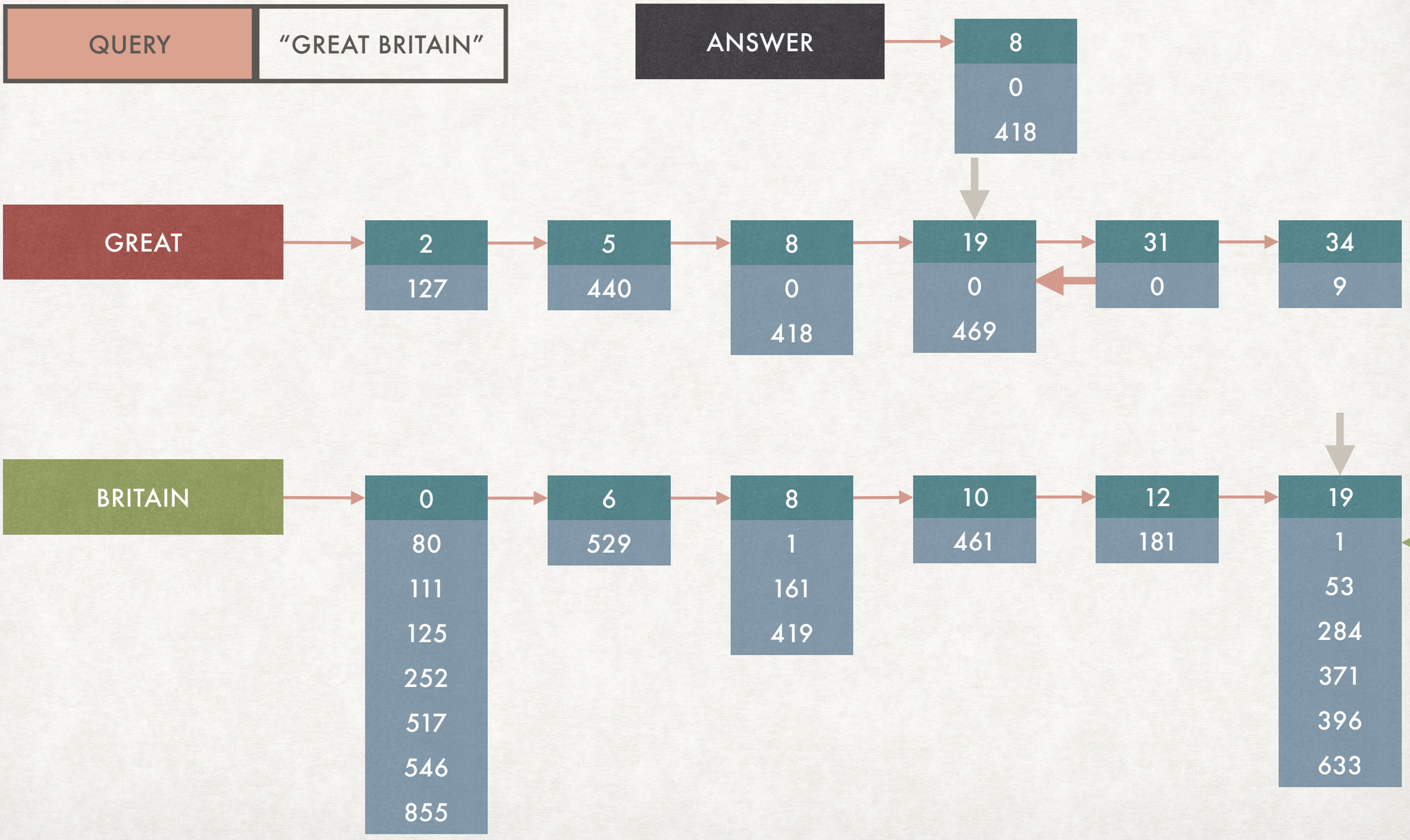
ANSWERING A PHRASE QUERY

WITH POSITIONAL INDEXING



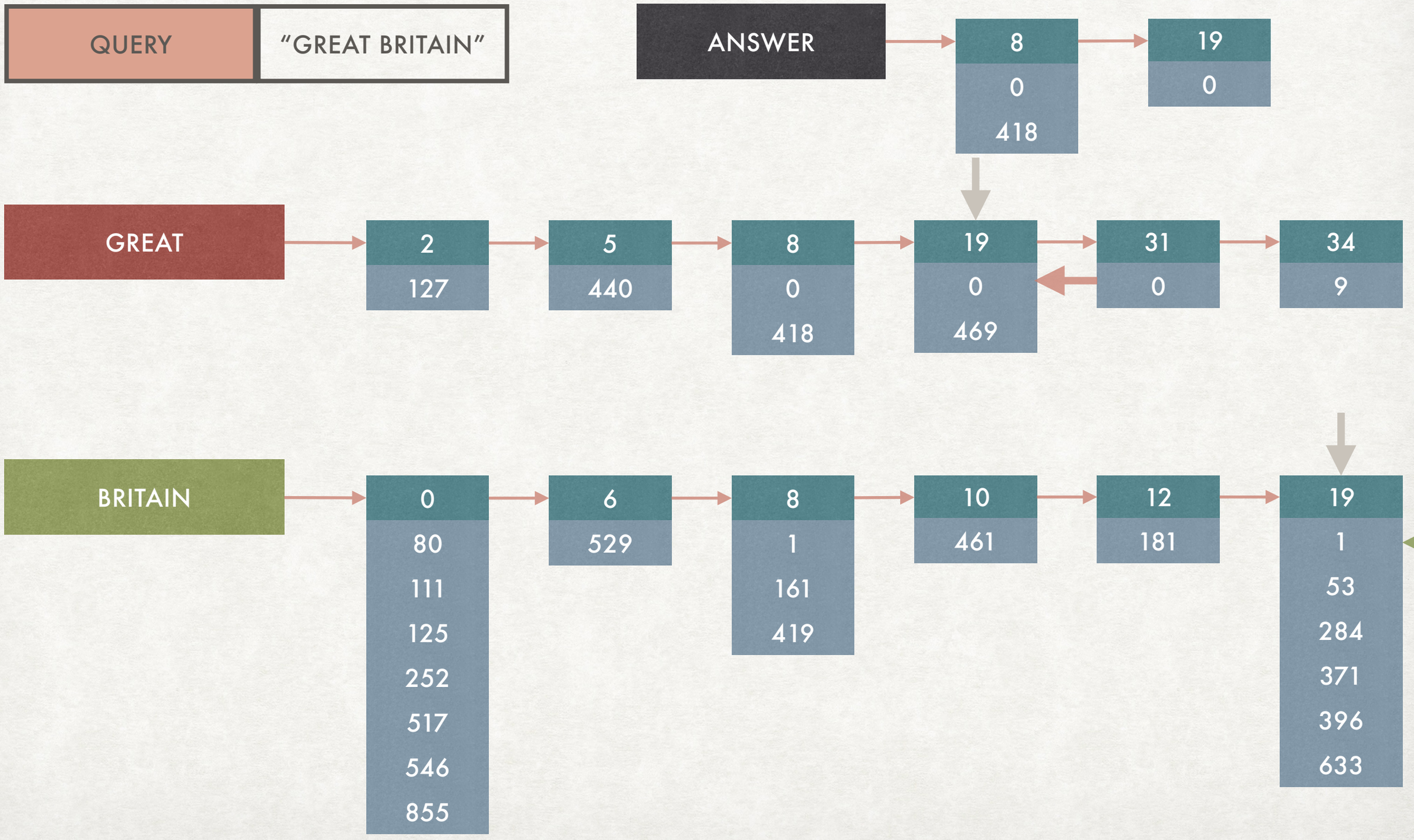
ANSWERING A PHRASE QUERY

WITH POSITIONAL INDEXING



ANSWERING A PHRASE QUERY

WITH POSITIONAL INDEXING

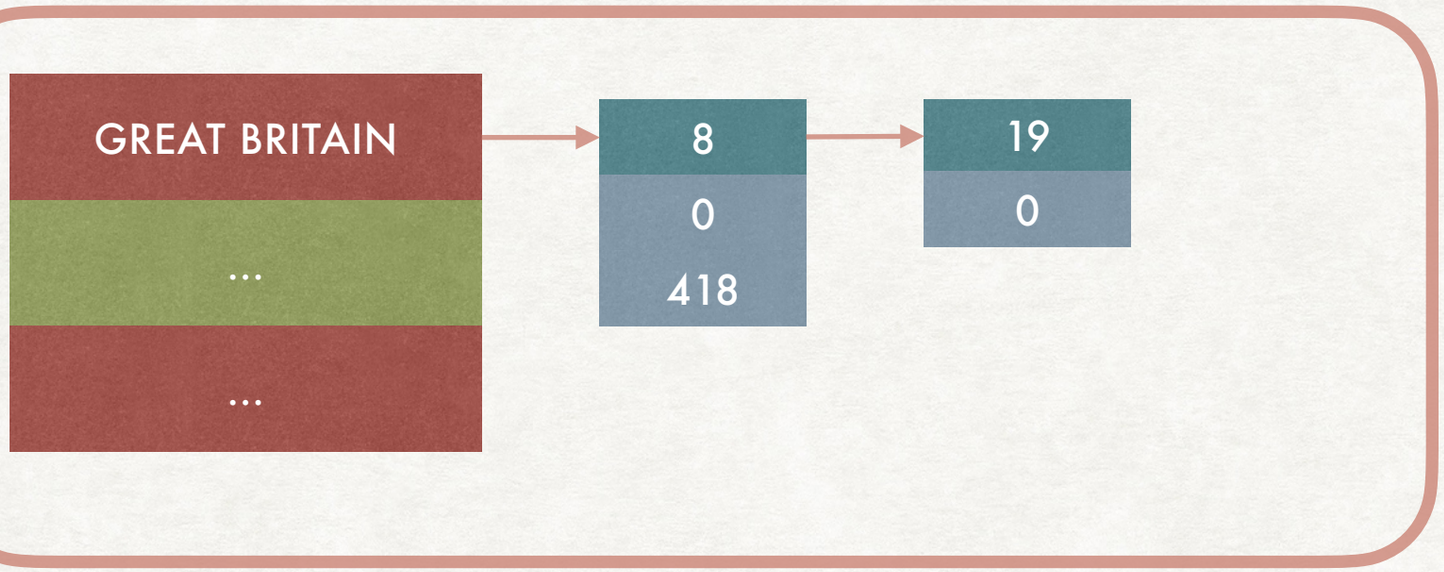


POSITIONAL INDEXING: SUMMARY

THE GOOD, THE BAD, AND THE UGLY

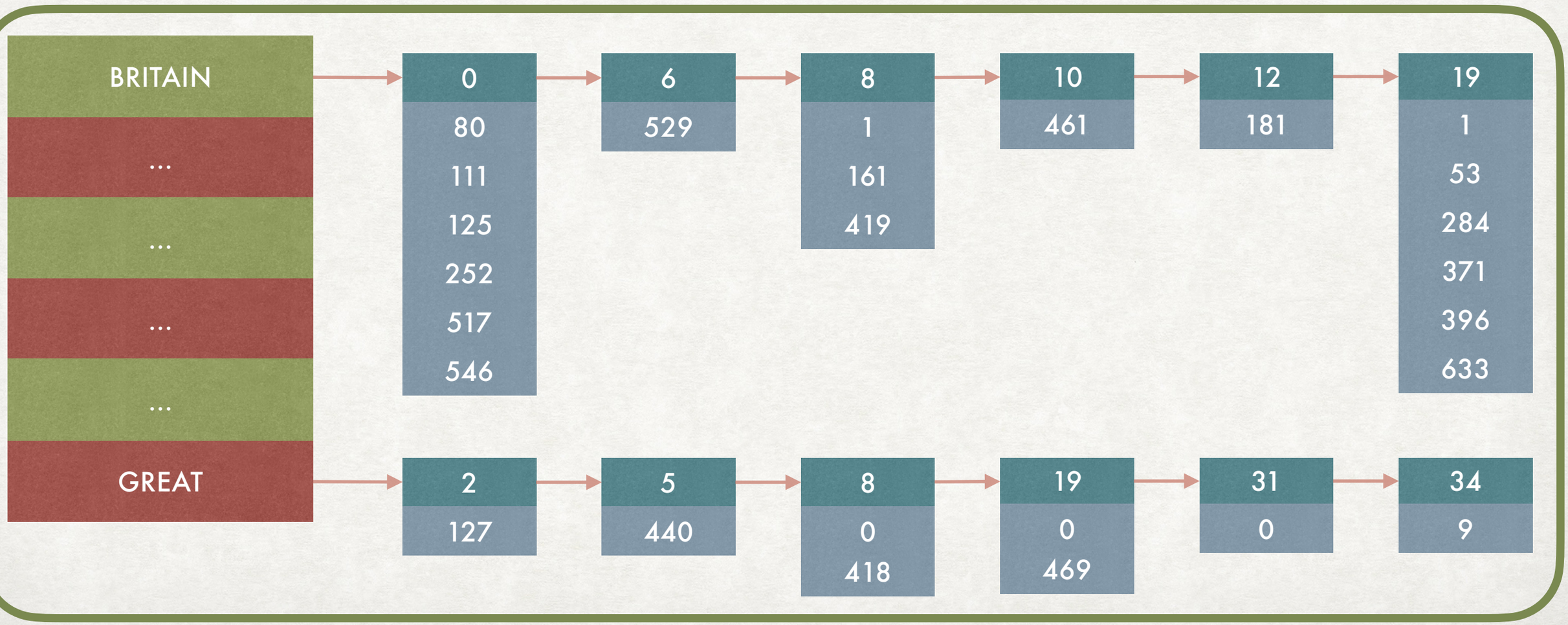
- The positional index can be used to support the operators of the form "$term_1$ /k $term_2$" with k an integer indicating the maximum number of words that can be between $term_1$ and $term_2$.
- The complexity of performing a query is not bounded anymore by the number of documents, but by the number of terms
- The size of the index now depends on the average document size.

COMBINING BIWORD AND POSITIONAL INDEXES



Phrase index for
frequently asked queries

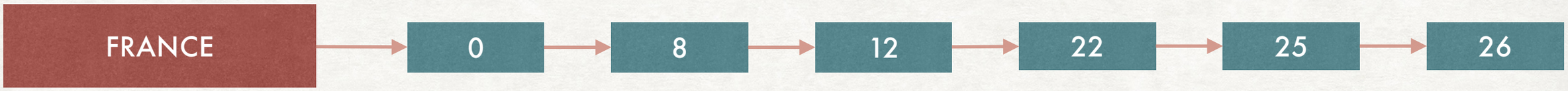
Positional index for
all other queries



IMPROVING THE INVERTED INDEX

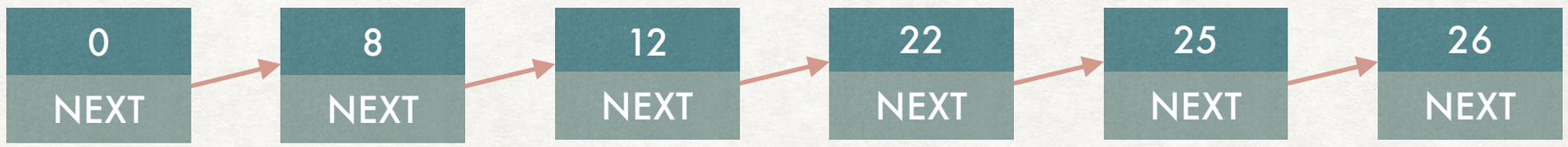
ARRAYS OR LINKED LISTS?

WHAT TO USE FOR THE POSTING LISTS?



Which data structures should we actually use for the postings list?

Singly linked lists



cheap insertion and updates

pointer overhead, poor memory locality (pointers chasing)

Variable length arrays

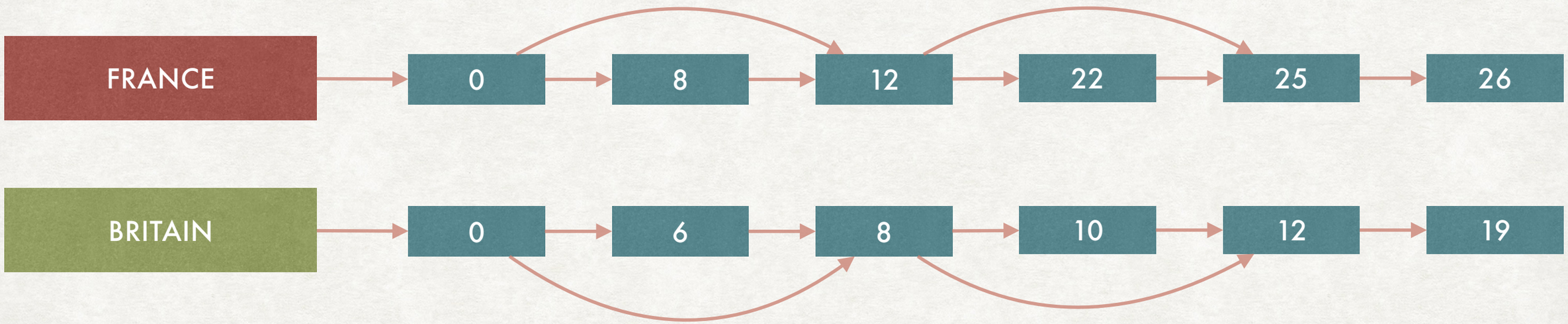
no pointers overhead, contiguous memory

difficult to update

SKIP LISTS

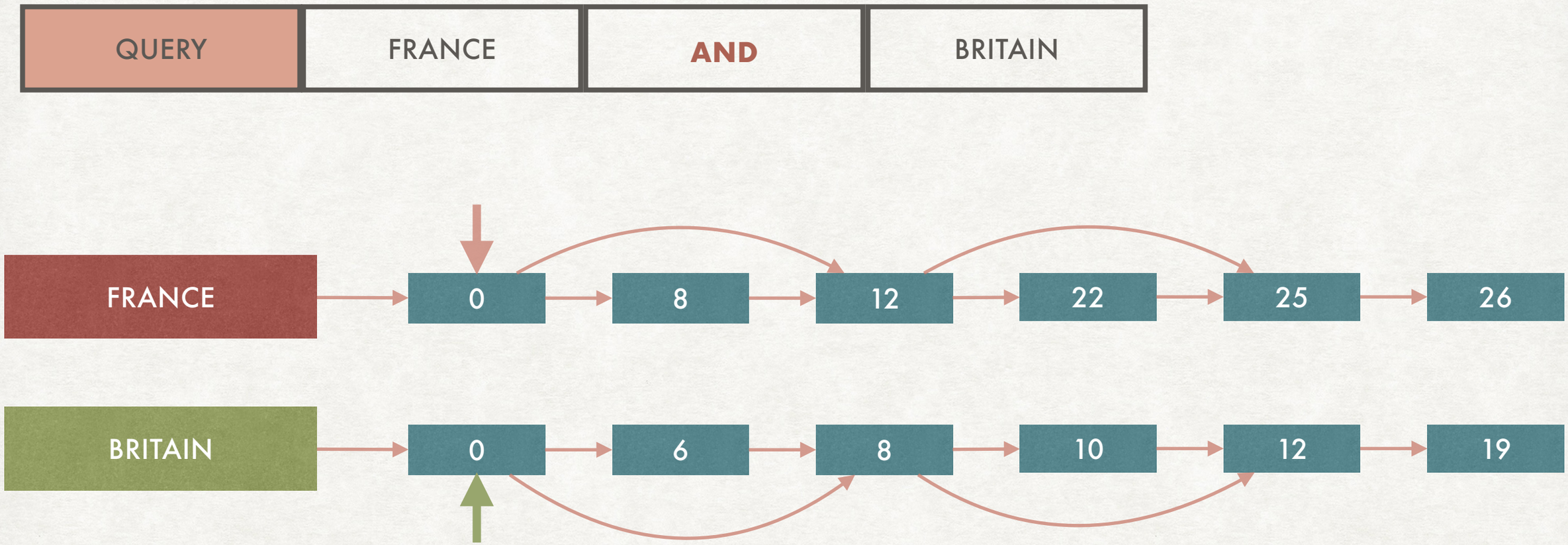
FASTER INTERSECTION

- We add additional forward pointers every k postings inside a list. The forward pointer "skips" a certain number of postings.
- A rule of thumb is, for a postings list of P postings to use $\sqrt{P}$ evenly spaced skip pointers



AN EXAMPLE QUERY

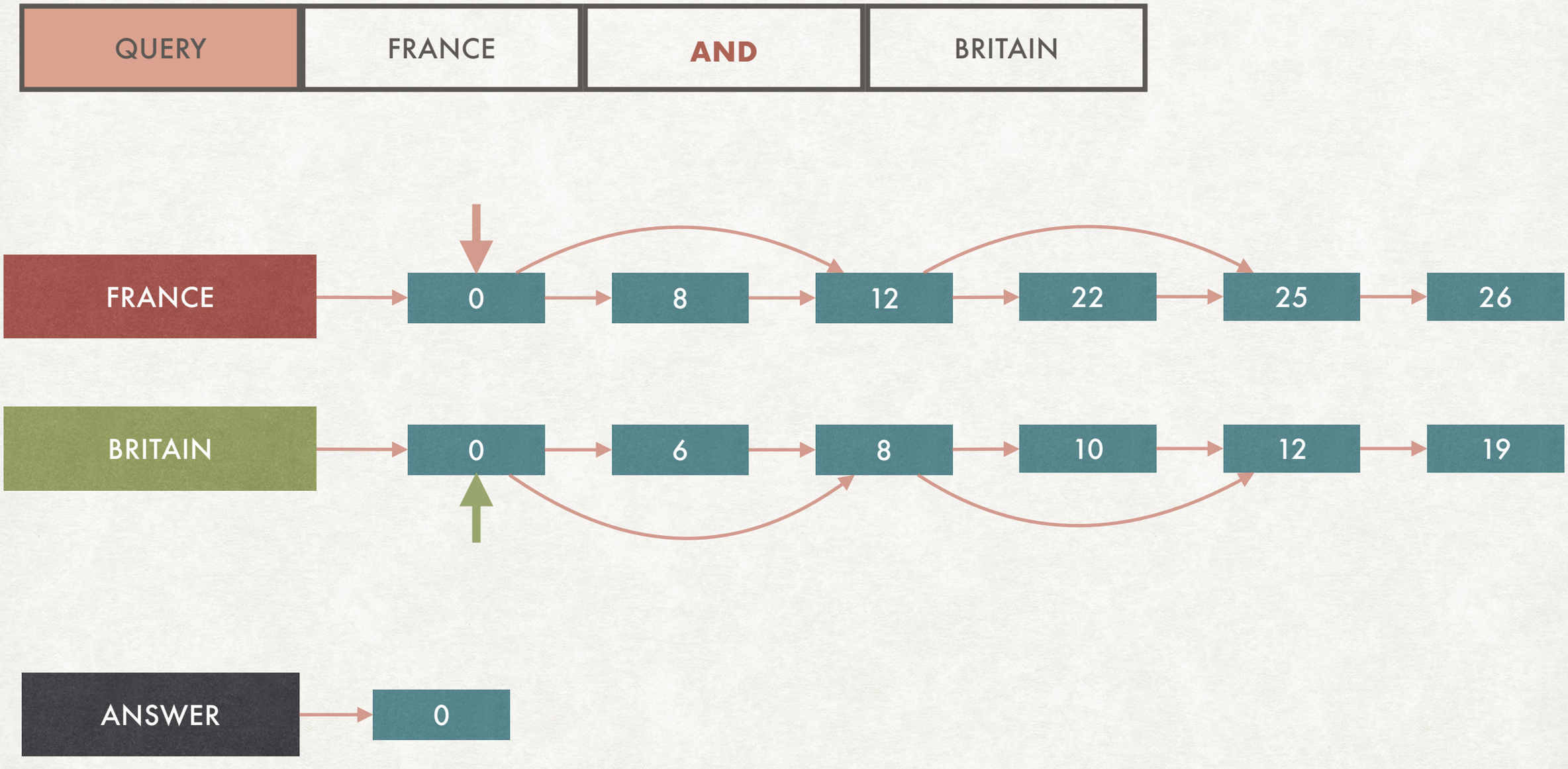
SAVING TIME WITH SKIP LISTS



ANSWER

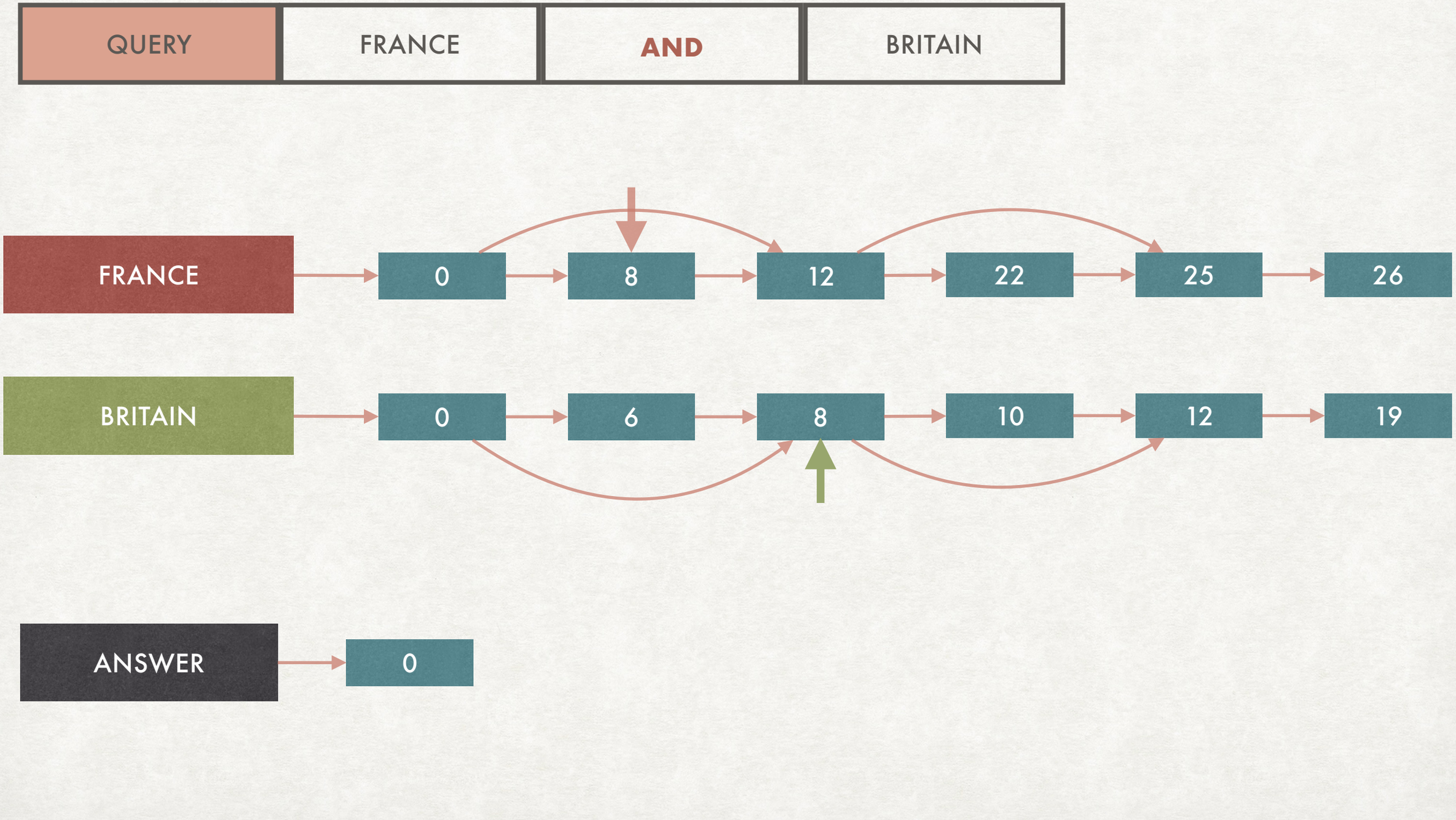
AN EXAMPLE QUERY

SAVING TIME WITH SKIP LISTS



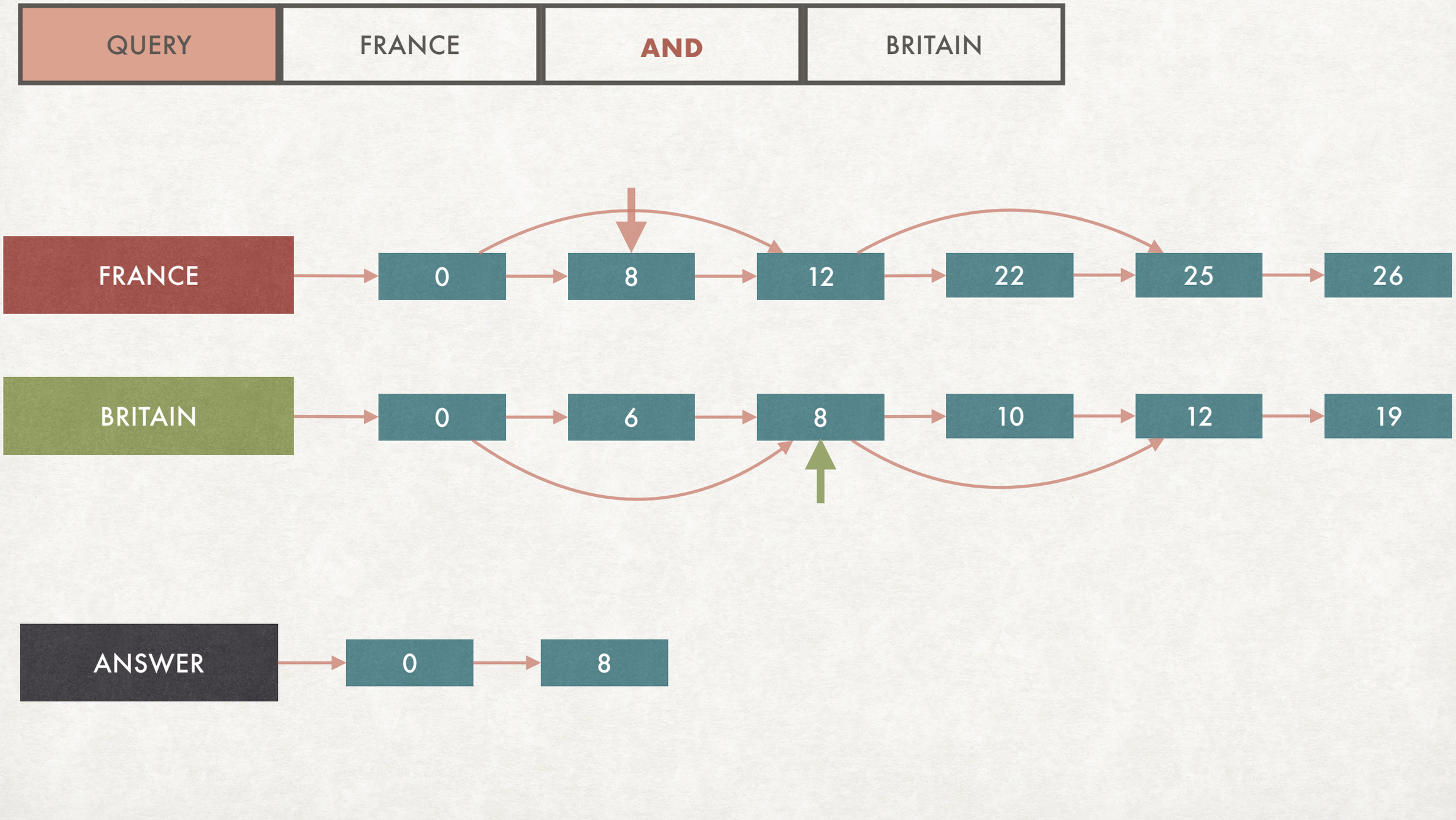
AN EXAMPLE QUERY

SAVING TIME WITH SKIP LISTS



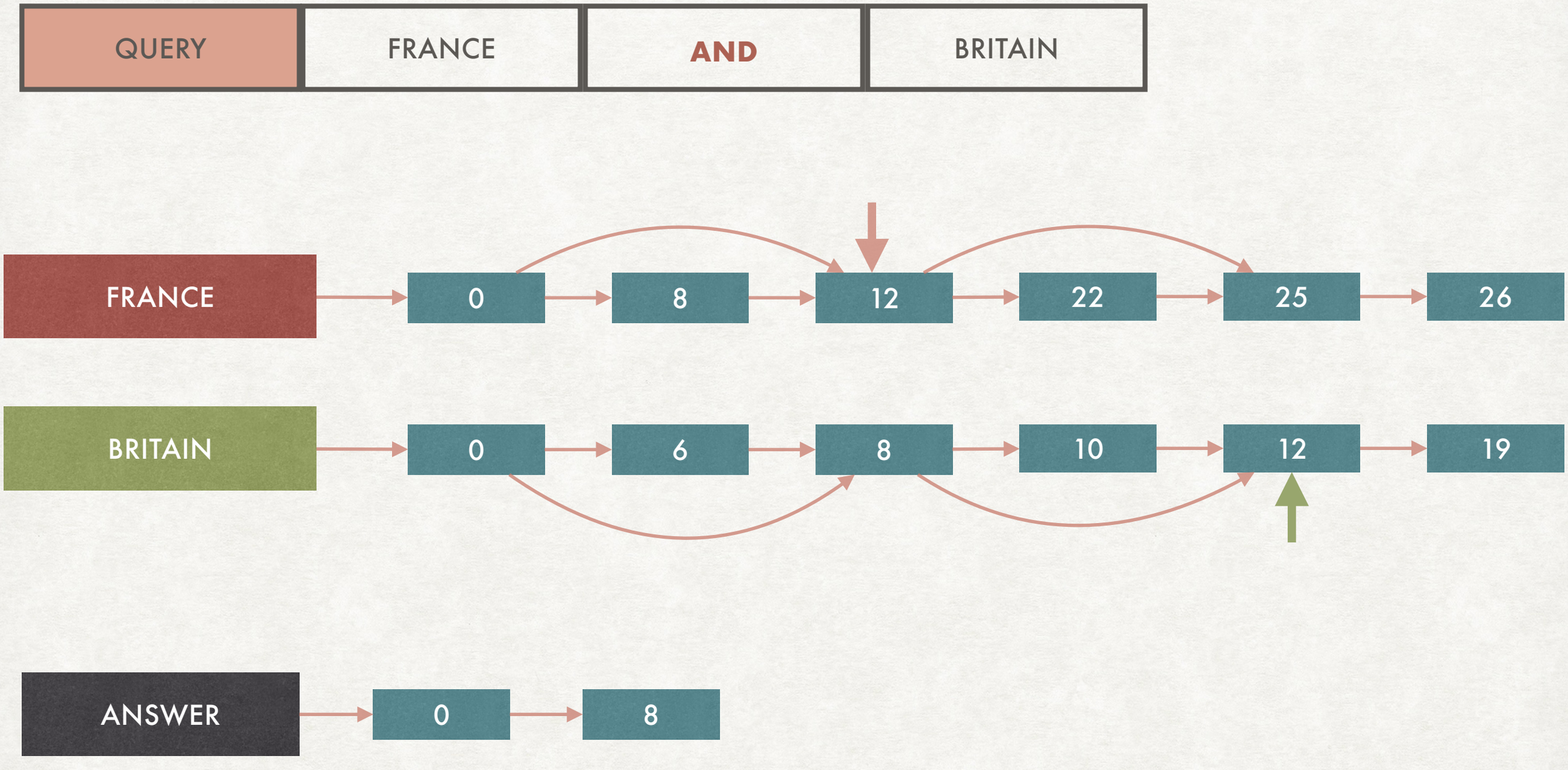
AN EXAMPLE QUERY

SAVING TIME WITH SKIP LISTS



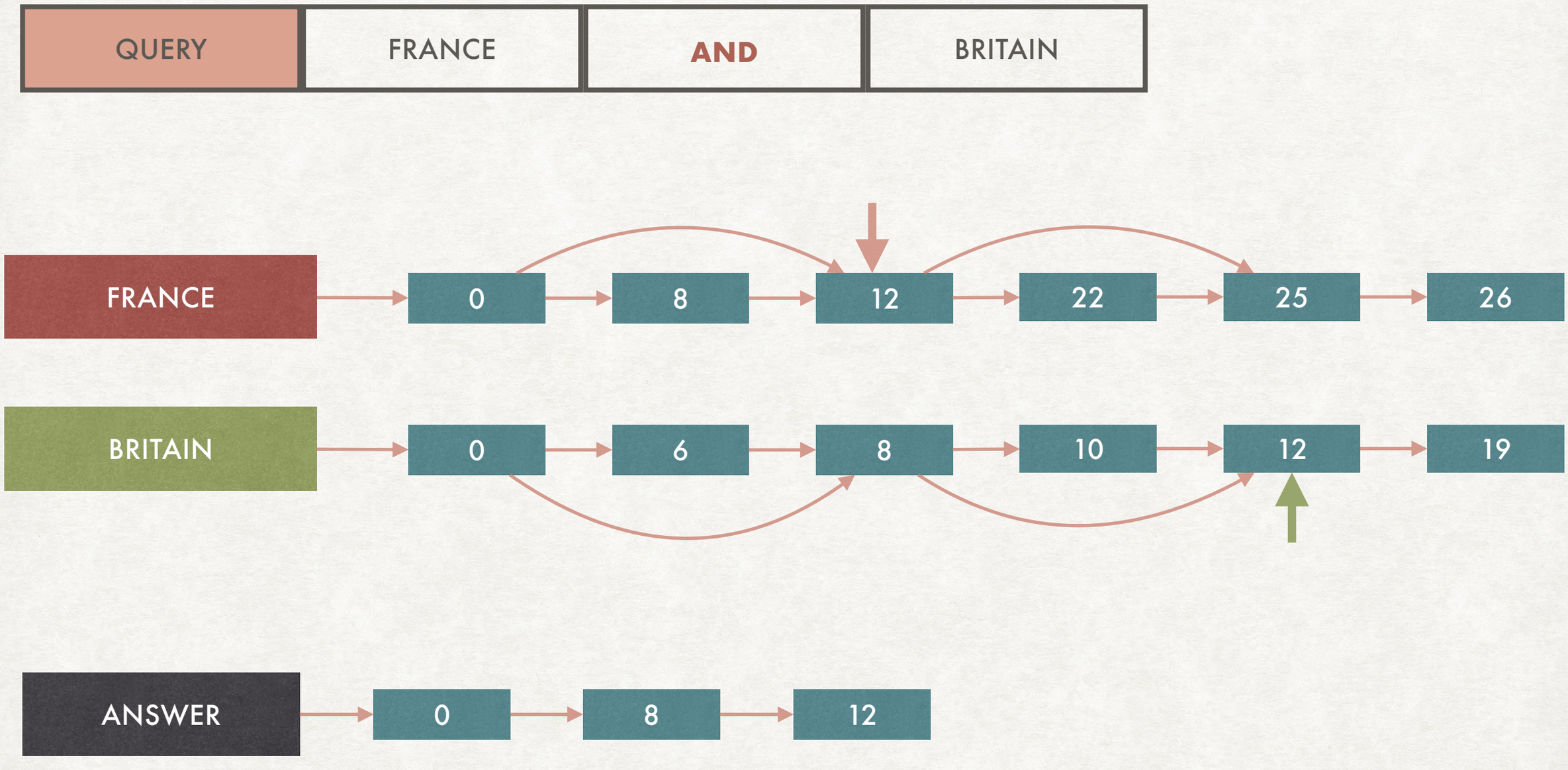
AN EXAMPLE QUERY

SAVING TIME WITH SKIP LISTS



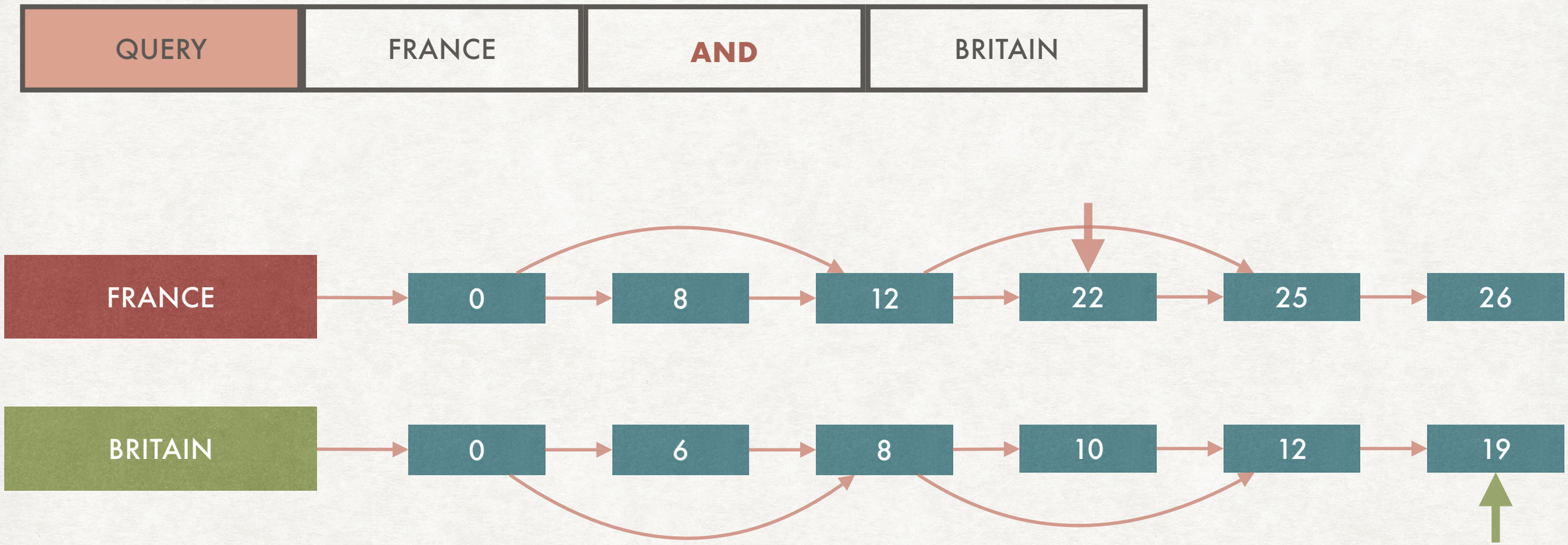
AN EXAMPLE QUERY

SAVING TIME WITH SKIP LISTS



AN EXAMPLE QUERY

SAVING TIME WITH SKIP LISTS



In some situations
we might only need
$O\left(\sqrt{P}\right)$ steps
to traverse a list

EXERCISES

THE PRACTICAL PART

- We are going to implement some of the algorithms and data structure described in this course
- We use Python 3, but you can follow along with any other programming language
- While IR systems must be efficient, we will sometimes allow for inefficiencies for the sake of more readable code
- Dataset that we use: http://www.cs.cmu.edu/~ark/personas/, more than 42k movie descriptions