# TEXT PROCESSING

6.S080 SOFTWARE SYSTEMS FOR DATA SCIENCE

TIM KRASKA

# CASE STUDY FOR THIS CLASS

You work at Nickelback Inc.

Nickelback Inc recently downloaded every song text ever written (TBs of data) to draw inspiration as they lately have trouble to produce a number 1 hit.

Now they want to create a system which enables them to search through this large collection of text and help them to write some songs.

# YOUR TASK

Task1: Design a system that efficiently finds all song texts contain certain keywords (e.g., "mountain" and "grass")

Task2: Create a simple ranking for the query results and enable that Nickelback can cluster the songs

Task3: Extend the system to allow search with sentiments (e.g., all happy songs, sad songs,…)

Task4: Extend the system to find songs with the right meaning of "grass" (FYI: Nickelback is a clean band)

Task5: Develop an assistant that helps Nickelback to write songs by predicting the next sentence

# GOAL: (EFFICIENT) TECHNIQUES TO PROCESS TEXT

**Basic queries:**

- How often does word X appear
- How often does word X and Y appear together
- …

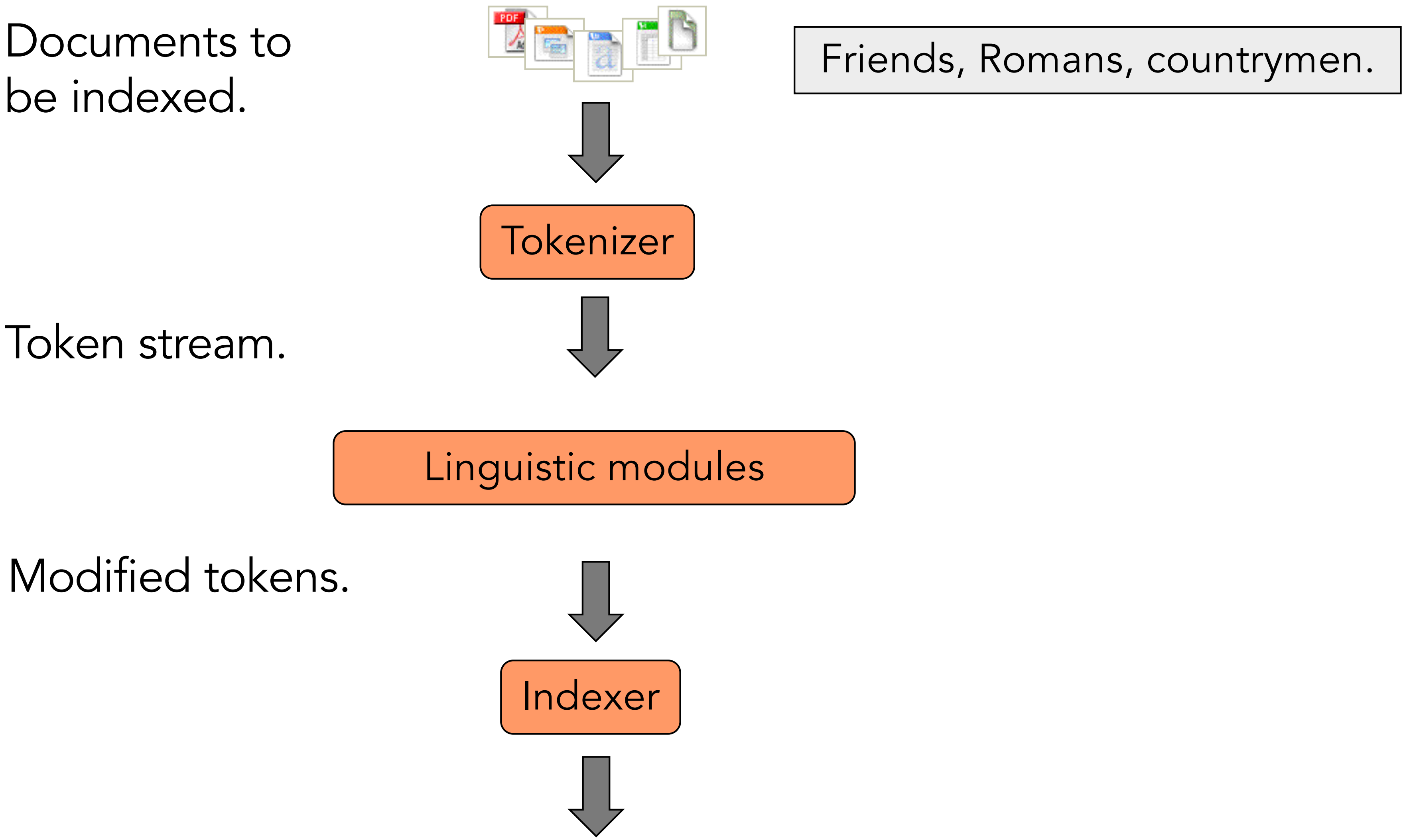
**Search engines:**

- Return the top 10 documents for a given query
- What news items are most relevant to me
- …

**Analytics:**

- What are trending topics on the web
- How to predict the unemployment rates of next month?
- How to predict Walmart´s sales numbers before they are released? (e.g., to make a buy or sell recommendation)
- …

# THE BASIC INDEXING PIPELINE



# TOKENIZATION

Input: "*Friends, Romans and Countrymen*"

Output: Tokens

- *Friends*
- *Romans*
- *and*
- *Countrymen*

A token is an instance of a sequence of characters

Each such token is now a candidate for an index entry, after further processing

But what are valid tokens to emit?

# CLICKER

Name 3 or more issues with tokenization, which could influence the search result?

# TOKENIZATION

## Issues in tokenization:

- Finland’s capital → Finland? Finlands? Finland’s?
- Hewlett-Packard → Hewlett and Packard as two tokens?
  - state-of-the-art, lowercase, lower-case, lower case ?
- San Francisco: one token or two?
  - How do you decide it is one token?

## Numbers/Dates

- Examples:
  - Date: 3/20/91 or Mar. 12, 1991 or 20/3/91 or 55 B.C.
  - Numbers: My PGP key is 324a3df234cb23e
  - Phone numbers: (800) 234-2333
- Older IR systems may not index numbers
  - But often very useful: think about things like looking up error codes/stacktraces on the web or finding a web-address

# TOKENIZATION: LANGUAGE ISSUES

## 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)

## 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)

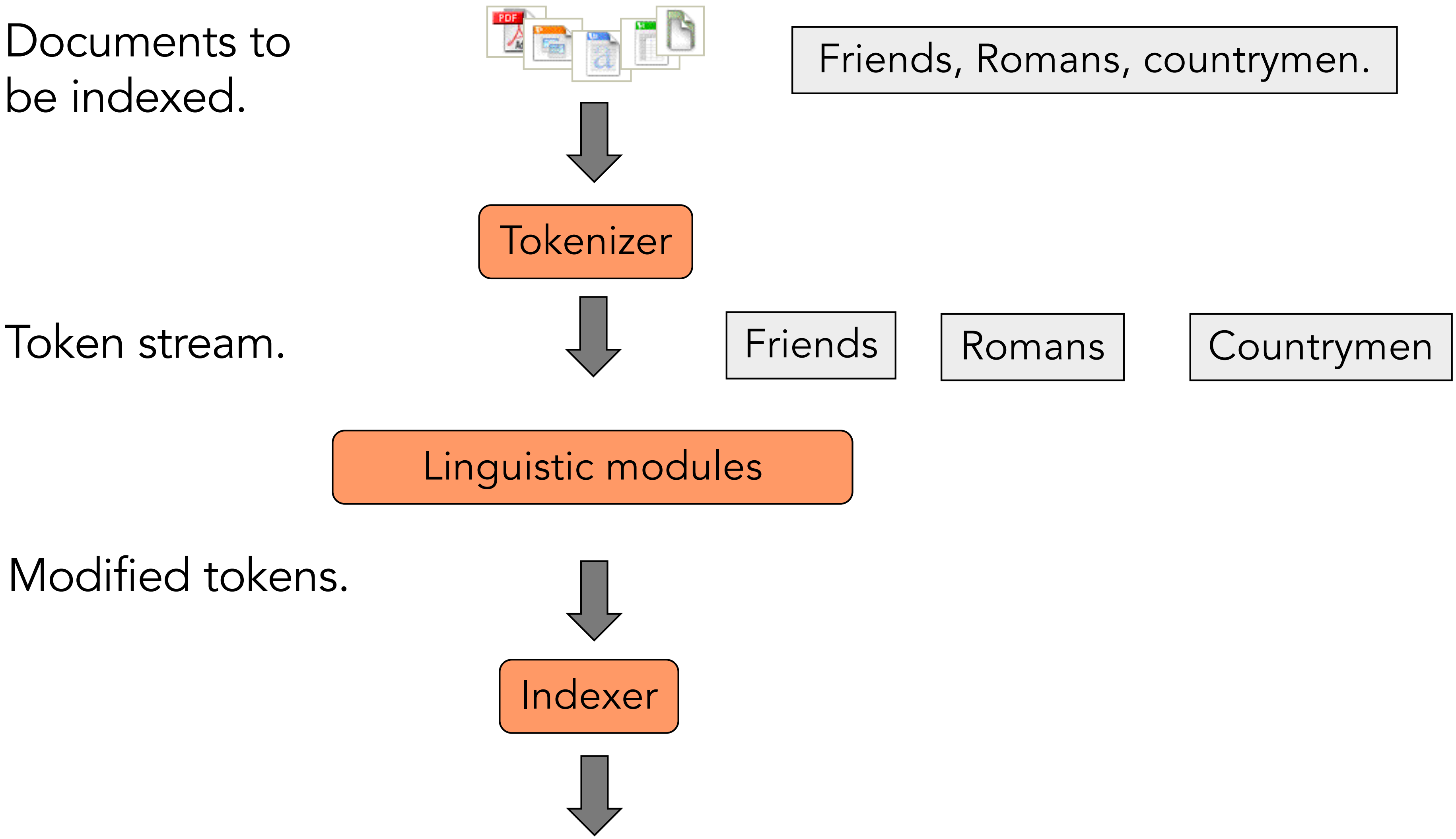
## Chinese and Japanese have no spaces between words:

- 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

## Arabic (or Hebrew) is basically written right to left, but with certain items like numbers written left to right

استقلت الجزائر في سنة 1962 بعد 132 عاما من الاحتلال الفرنسي.

# THE BASIC INDEXING PIPELINE



# STOP WORDS

With a stop list, you exclude from the dictionary entirely the most common words. Intuition:

- They have little semantic content: *the, a, and, to, be*
- There are a lot of them: ~30% of postings for top 30 words

Clicker:

a) For search and analytical tasks always remove them
b) Only for search tasks remove them
c) Only for analytical tasks remove them
d) Never remove
e) Scooby-doo – do not pick this answer ☺

# STOP WORDS

**With a stop list, you exclude from the dictionary entirely the most common words. Intuition:**

- They have little semantic content: *the, a, and, to, be*
- There are a lot of them: ~30% of postings for top 30 words

**But the trend is away from doing this:**

- Good compression techniques means the space for including stopwords 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”

**In contrast for analytics: you often remove them. Why?**

# WHAT OTHER MODIFICATIONS CAN YOU THINK OF?

# NORMALIZATION TO TERMS

We need to "normalize" words in indexed text as well as query words into the same form

- We want to match *U.S.A.* and *USA*

Result is terms: a term is a (normalized) word type, which is an entry in our IR system dictionary

We most commonly implicitly define equivalence classes of terms by, e.g.,

- deleting periods to form a term
  - *U.S.A., USA* → *USA*
- deleting hyphens to form a term
  - *anti-discriminatory, antidiscriminatory* → *antidiscriminatory*

# NORMALIZATION: OTHER LANGUAGES

Accents: e.g., French *résumé* vs. *resume.*

Umlauts: e.g., German: *Tuebingen* vs. *Tübingen*

- Should be equivalent

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

- Often best to normalize to a de-accented term
  - *Tuebingen, Tübingen, Tubingen* → *Tubingen*

# CASE FOLDING

**Reduce all letters to lower case**

- exception: upper case in mid-sentence?
  - e.g., *General Motors*
  - *Fed* vs. *fed*
  - *Brown* vs. *brown*
- Often best to lower case everything, since users will use lowercase regardless of ‘correct’ capitalization…

**Google example:**

- Query *C.A.T.*
- Even today #1 result is for “cat” *not* Caterpillar Inc.

# NORMALIZATION TO TERMS

An alternative to equivalence classing is to do asymmetric expansion

An example of where this may be useful

- Enter: *window* Search: *window, windows*
- Enter: *windows* Search: *Windows, windows, window*
- Enter: *Windows* Search: *Windows*

Potentially more powerful, but less efficient

# 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

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

Common 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.*

Typical rules in porter:

- sses → ss
- ies → i
- ational → ate
- tional → tion
- Weight of word sensitive rules

  (m>1) EMENT →

  - replacement → replac
  - cement → cement

Other stemmers exist, e.g., Lovins stemmer

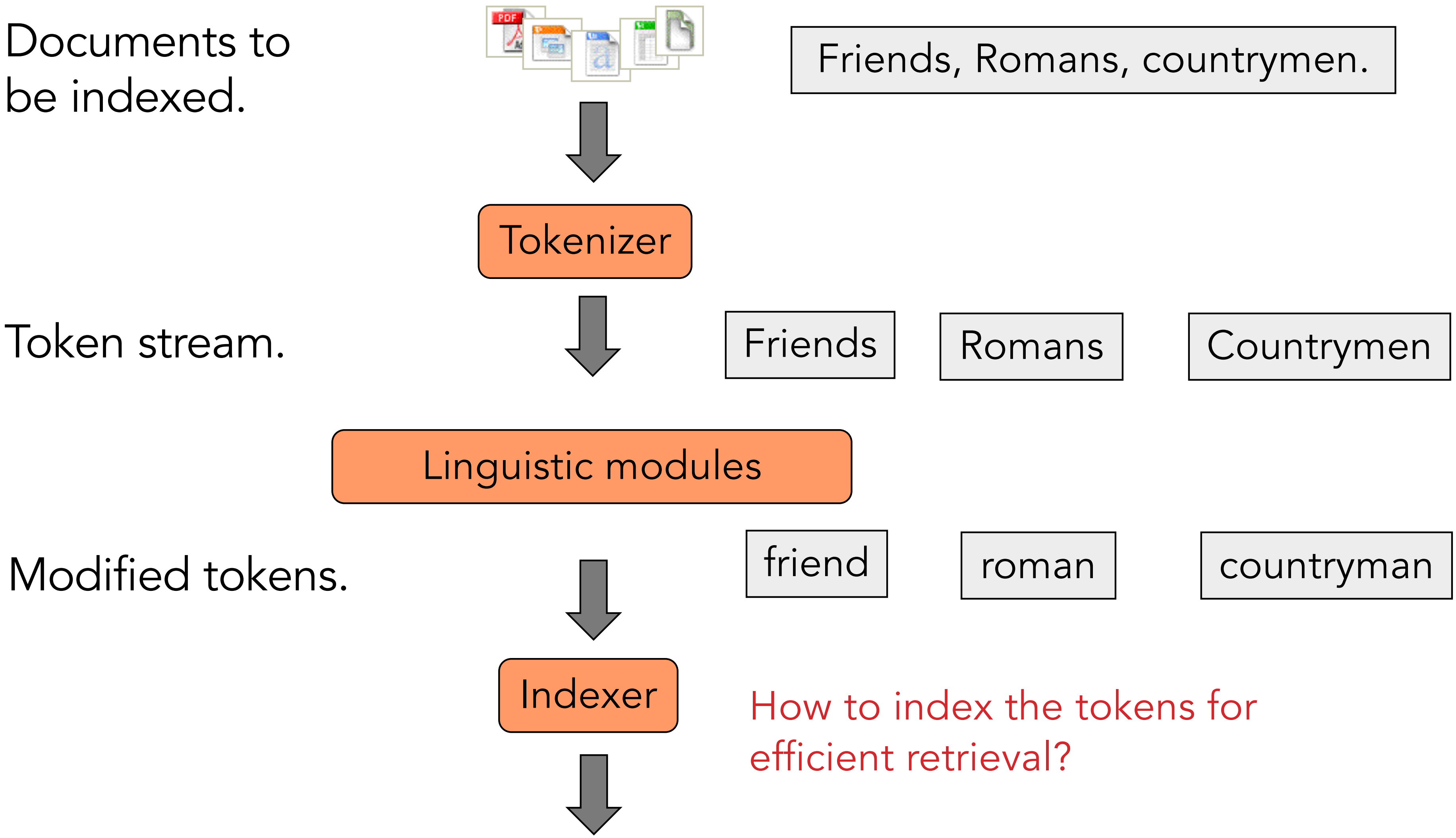
- http://www.comp.lancs.ac.uk/computing/research/stemming/general/lovins.htm
- Single-pass, longest suffix removal (about 250 rules)

# MAIN TAKE-AWAY

Be aware what your are indexing and how it is processed

→ Huge differences in recall/precision and performance

# THE BASIC INDEXING PIPELINE



# BIT-INDEX / TERM-DOCUMENT INCIDENCE

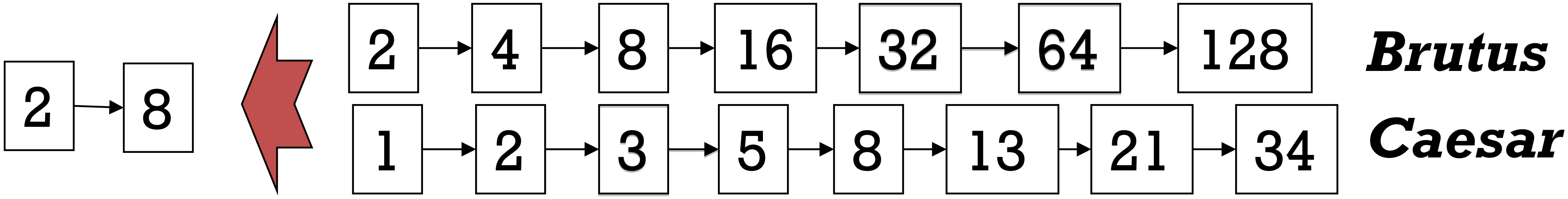
| | Anotony and Cleopatra | Julius Caesar | The Tempest | Hamlet | Othello | Macbeth |
|---|---|---|---|---|---|---|
| Antony | 1 | 1 | 0 | 0 | 0 | 1 |
| Brutus | 1 | 1 | 0 | 1 | 0 | 0 |
| Caesar | 1 | 1 | 0 | 1 | 1 | 1 |
| Calpurnia | 0 | 1 | 0 | 0 | 0 | 0 |
| Cleopatra | 1 | 0 | 0 | 0 | 0 | 0 |
| Mercy | 1 | 0 | 1 | 1 | 1 | 1 |
| Worser | 1 | 0 | 1 | 1 | 1 | 0 |

# INVERTED INDEX

| 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 |

How do we get all documents which include "Julius" and "Caesar"?

# CLICKER: INTERSECTING TWO POSTINGS LISTS (A "MERGE" JOIN)




```
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
```

Fill in the missing operator

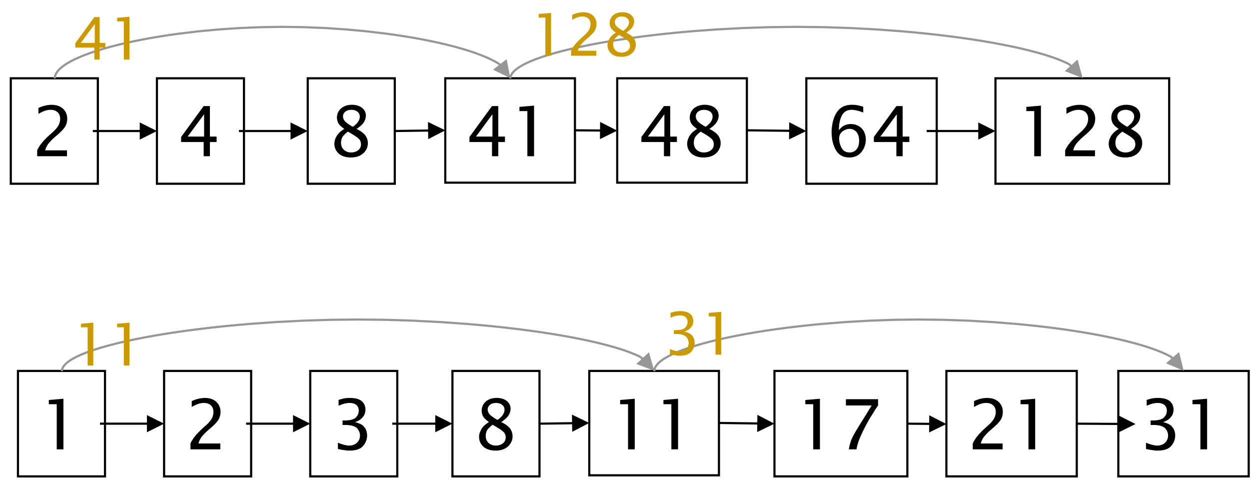
A) Scooby-doo
B) <
C) >
D) =

# CLICKER: INTERSECTING TWO POSTINGS LISTS (A “MERGE” JOIN)

```
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
```

Can you think of a way to speed-up the merge join?

# QUERY PROCESSING WITH SKIP POINTERS



# CLICKER

What is the best order for query processing?

Consider a query that is an *AND* of *n* terms.

For each of the *n* terms, get its postings, then *AND* them together.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Brutus* | 2 | 4 | 8 | 16 | 32 | 64 | 128 |
| *Caesar* | 1 | 2 | 3 | 5 | 8 | 16 | |
| *Calpurnia* | 13 | 16 | | | | | |

Query: *Brutus AND Calpurnia AND Caesar*

Clicker: What join order should you use?

A) Brutus join Caesar then join Culpurnia
B) Scooby-doo
C) Caesar join Calpurnia then join Brutus
D) Calpurnia join Brutus then join Caesar

# QUERY OPTIMIZATION EXAMPLE

Process in order of increasing freq:

- *start with smallest set, then keep cutting further.*

This is why we kept document freq. in dictionary

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| ***Brutus*** | 2 | 4 | 8 | 16 | 32 | 64 | 128 |
| ***Caesar*** | 1 | 2 | 3 | 5 | 8 | 16 | |
| ***Calpurnia*** | 13 | 16 | | | | | |

Execute the query as (***Calpurnia*** *AND* ***Brutus****)* *AND* ***Caesar***.

# SAME AS IN RELATIONAL MODEL

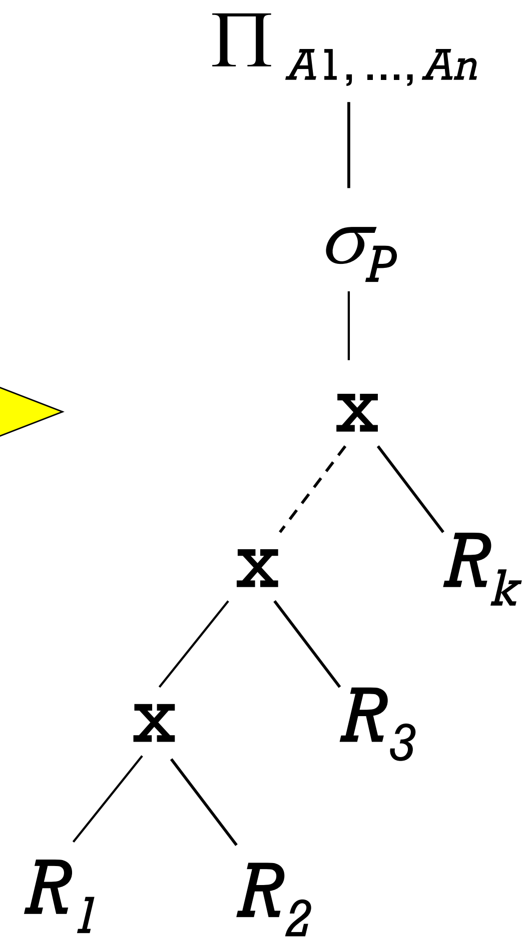
**Relational Algebra**

**SQL**

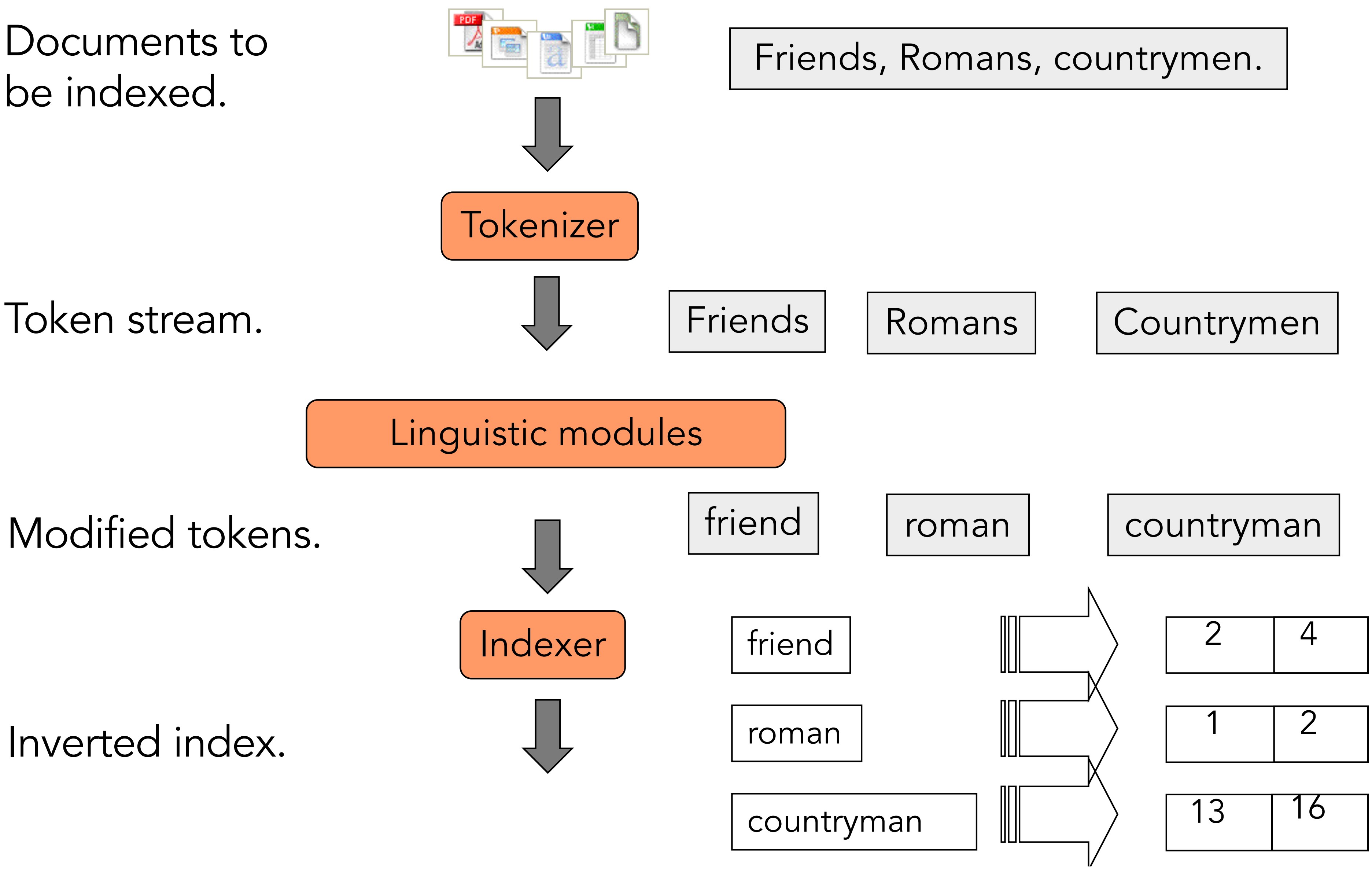
**select** $A_1, \ldots, A_n$

**from** $R_1, \ldots, R_k$

**where** $P$;



# THE BASIC INDEXING PIPELINE



# PHRASE QUERIES

Want to be able to answer queries such as "computer science" – as a phrase

Thus the sentence *"I worked on my science project on the computer"* 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*

For this, it no longer suffices to store only

<*term* : *docs*> entries

**Ideas???**

# A FIRST ATTEMPT: BIWORD INDEXES

Index every **consecutive pair of terms** in the text as a phrase

Longer phrases are processed as we do with wild-cards:

*Massachusetts Institute of Technology* can be broken into the Boolean query on biwords:

*"Massachusetts Institute" AND "Institute of" AND "of Technology"*

Clicker:

A) This strategy always works

B) Leads to less recall

C) Leads to less precision

D) Scooby-doo

# SOLUTION 2: POSITIONAL INDEXES

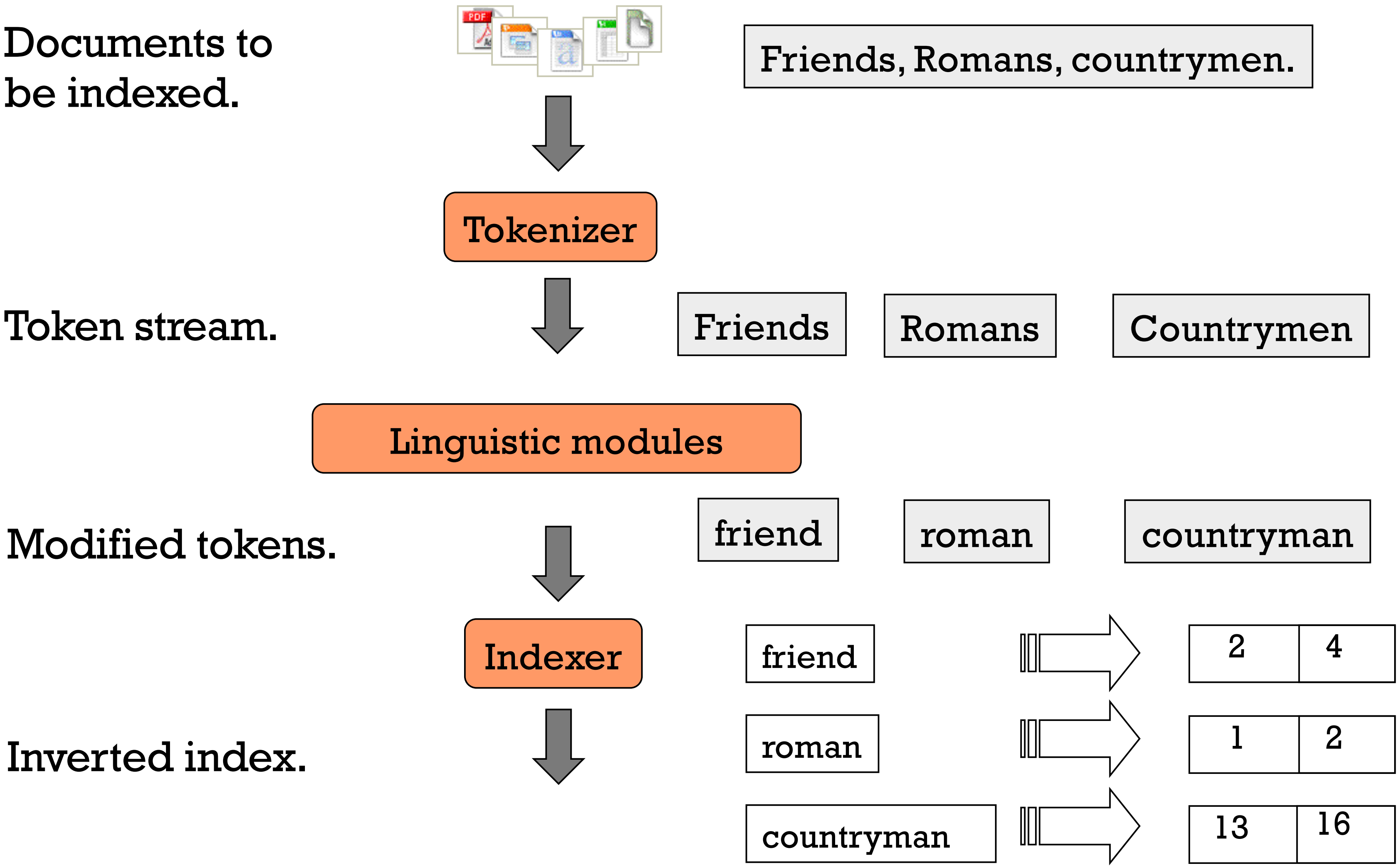
**In the postings, store, for each *term* the position(s) in which tokens of it appear:**

<***term***, number of docs containing ***term***;
*doc1*: position1, position2 … ;
*doc2*: position1, position2 … ;
etc.>

**An Example:**
**<*be*: 993427;**
***1*: 7, 18, 33, 72, 86, 231;**
***2*: 3, 149;**
***4*: 17, 191, 291, 430, 434;**
***5*: 363, 367, …;**

- Extended version of merge join can be used
- Allows for proximity or wildcard queries
- Rules of thumb
  - A positional index is 2–4 as large as a non-positional index
  - Positional index size 35–50% of volume of original text
  - Caveat: all of this holds for English-like” languages

# THE BASIC INDEXING PIPELINE



# CASE STUDY FOR THIS CLASS

You work at Nickelback Inc.

Nickelback Inc recently downloaded every song text ever written (TB of data) to draw inspiration as they lately have trouble to produce a number 1 hit.

Now they want to create a system which enables them to search through this large collection of text and help them to write some songs.

Your task:

Task1: Design a system that efficiently finds all song texts contain certain keywords (e.g., "mountain" and "grass")

Task2: Create a simple ranking for the query results and enable that Nickelback can cluster the songs

Task3: Extend the system to allow search with sentiments (e.g., all happy songs, sad songs,…)

Task4: Extend the system further to find songs with the right meaning of "grass" (the green stuff in the football stadium)

Task5: Develop an assistant that helps Nickelback to write songs by predicting the next sentence

# TERM-DOCUMENT COUNT INDICES

Idea: create a vector representation of the document and compare the vectors (e.g., cosine similarity)

Consider the number of occurrences of a term in a document:

- Each document is a count vector in $\mathbb{N}^v$: a column below

| | Antony and Cleopatra | Julius Caesar | The Tempest | Hamlet | Othello | Macbeth |
|---|---|---|---|---|---|---|
| Antony | 157 | 73 | 0 | 0 | 0 | 0 |
| Brutus | 4 | 157 | 0 | 1 | 0 | 0 |
| Caesar | 232 | 227 | 0 | 2 | 1 | 1 |
| Calpurnia | 0 | 10 | 0 | 0 | 0 | 0 |
| Cleopatra | 57 | 0 | 0 | 0 | 0 | 0 |
| mercy | 2 | 0 | 3 | 5 | 5 | 1 |
| worser | 2 | 0 | 1 | 1 | 1 | 0 |

Bag of words model

- Vector representation doesn't consider the ordering of words in a document
- *John is quicker than Mary* and *Mary is quicker than John* have the same vectors
- In a sense, this is a step back: The positional index was able to distinguish these two documents.

# TERM FREQUENCY TF

The term frequency $\mathrm{tf}_{t,d}$ of term *t* in document *d* is defined as the number of times that *t* occurs in *d*.

We want to use tf when computing query-document match scores. But how?

Raw term frequency is not what we want:

- A document with 10 occurrences of the term is more relevant than a document with 1 occurrence of the term.
- But not 10 times more relevant.

Relevance does not increase proportionally with term frequency.

$$w_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{tf}_{t,d}, & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases}$$

TF score $= \sum_{t \in q \cap d} (1 + \log \mathrm{tf}_{t,d})$

Clicker: Are we done?

A) Looks all good to me
B) Scooby-doo
C) Rare words are a problem
D) Large documents are a problem

# RECALL: IDF WEIGHT

Frequent terms are less informative than rare terms

$df_t$ is the document frequency of *t*: the number of documents that contain *t*

- $df_t$ is an inverse measure of the informativeness of *t*
- $df_t \leq N$

We define the idf (inverse document frequency) of *t* by

$$\mathrm{idf}_t = \log_{10}\left(N/\mathrm{df}_t\right)$$

- We use log ($N$/$df_t$) instead of $N$/$df_t$ to "dampen" the effect of idf.

# TF-IDF WEIGHTING

The tf-idf weight of a term is the product of its tf weight and its idf weight.

$$\mathrm{w}_{t,d} = (1 + \log \mathrm{tf}_{t,d}) \times \log_{10}(N / \mathrm{df}_t)$$

Best known weighting scheme in information retrieval

- Note: the “-” in tf-idf is a hyphen, not a minus sign!
- Alternative names: tf.idf, tf x idf

Increases with the number of occurrences within a document

Increases with the rarity of the term in the collection

# BINARY → COUNT → WEIGHT MATRIX

| | Antony and Cleopatra | Julius Caesar | The Tempest | Hamlet | Othello | Macbeth |
|---|---|---|---|---|---|---|
| Antony | 5.25 | 3.18 | 0 | 0 | 0 | 0.35 |
| Brutus | 1.21 | 6.1 | 0 | 1 | 0 | 0 |
| Caesar | 8.59 | 2.54 | 0 | 1.51 | 0.25 | 0 |
| Calpurnia | 0 | 1.54 | 0 | 0 | 0 | 0 |
| Cleopatra | 2.85 | 0 | 0 | 0 | 0 | 0 |
| mercy | 1.51 | 0 | 1.9 | 0.12 | 5.25 | 0.88 |
| worser | 1.37 | 0 | 0.11 | 4.15 | 0.25 | 1.95 |
| the | ? | ? | ? | ? | ? | ? |

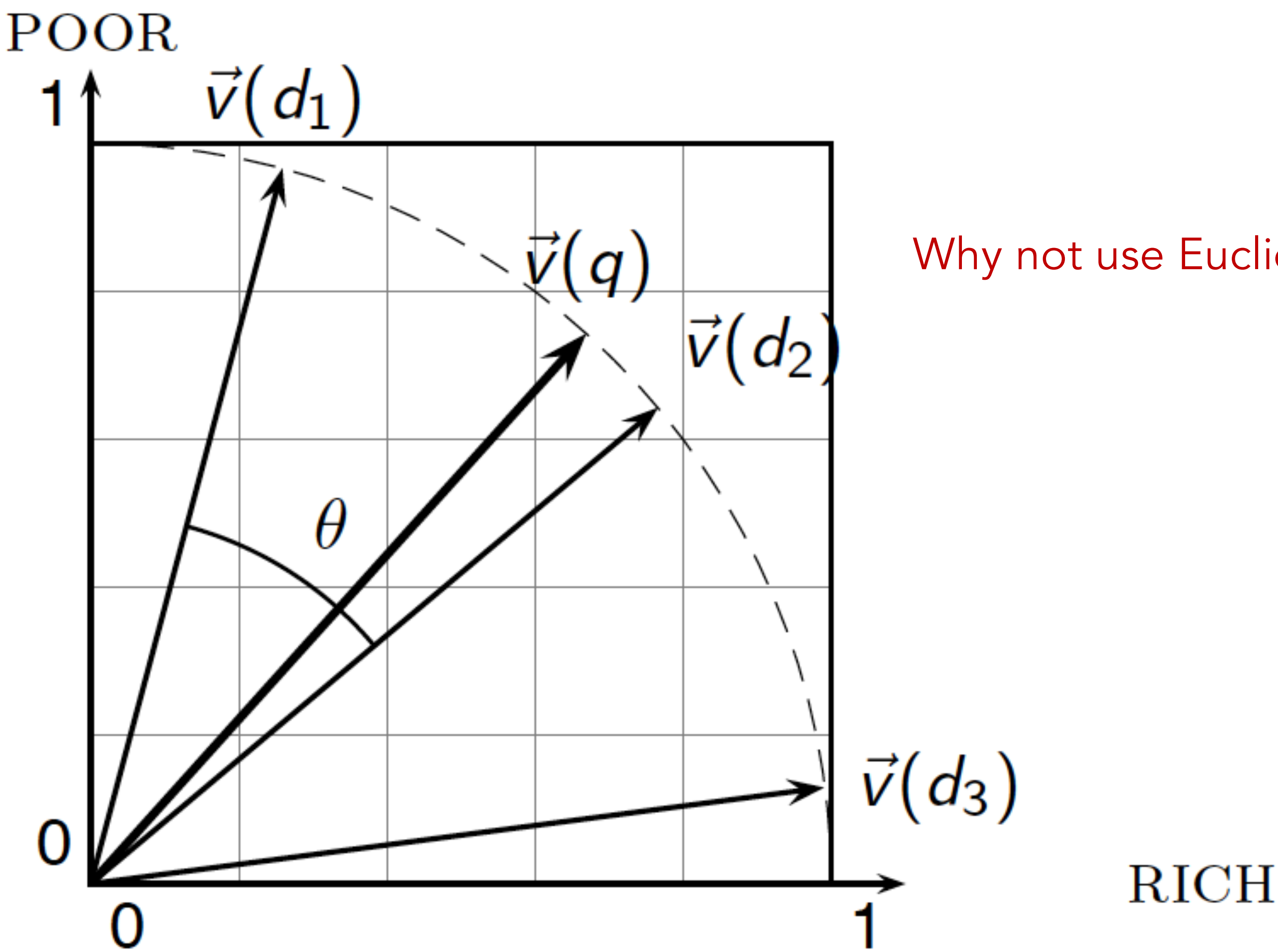
Each document is now represented by a real-valued vector of tf-idf weights $\in \mathbb{R}^{|V|}$

Clicker: What value do you suspect for the last row?

a) All 0
b) Elmo and Bert
c) All 1
d) All values > 1

$$w_{t,d} = (1 + \log \mathrm{tf}_{t,d}) \times \log_{10}(N / \mathrm{df}_t)$$

# COSINE SIMILARITY




Why not use Euclidean distance?

# CASE STUDY FOR THIS CLASS

You work at Nickelback Inc.

Nickelback Inc recently downloaded every song text ever written (TB of data) to draw inspiration as they lately have trouble to produce a number 1 hit.

Now they want to create a system which enables them to search through this large collection of text and help them to write some songs.

Your task:

Task1: Design a system that efficiently finds all song texts contain certain keywords (e.g., "mountain" and "grass")

Task2: Rank the query results based on relevance and be able to find and cluster/similar song texts

**Task3: Extend the system to allow search with sentiments (e.g., all happy songs, sad songs,…)**

Task4: Extend the system further to find songs with the right meaning of "grass"

Task5: Develop an assistant that helps Nickelback to write songs by predicting the next sentence

# LEVERAGE WORDNET

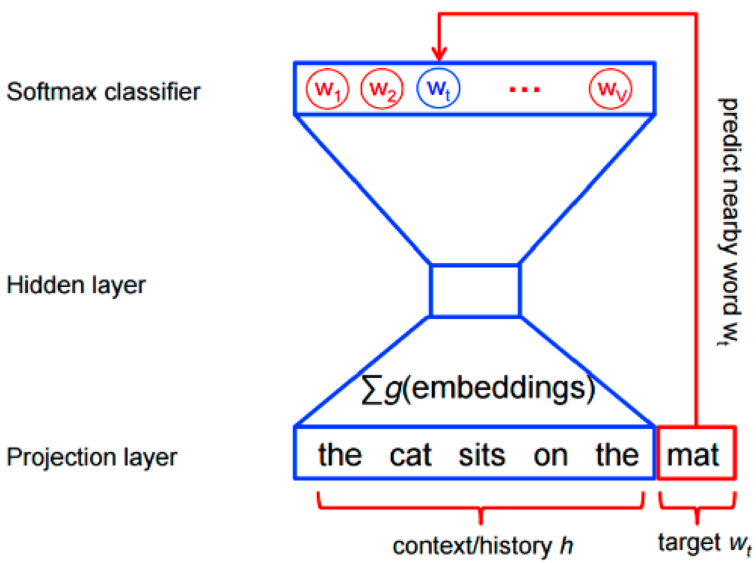
**Unsupervised**: Wordnet affect or similar

**Supervised**: train classifier – but how should we encode the words?

# SUPERVISED

# WHAT DOES WORD2VEC DO?

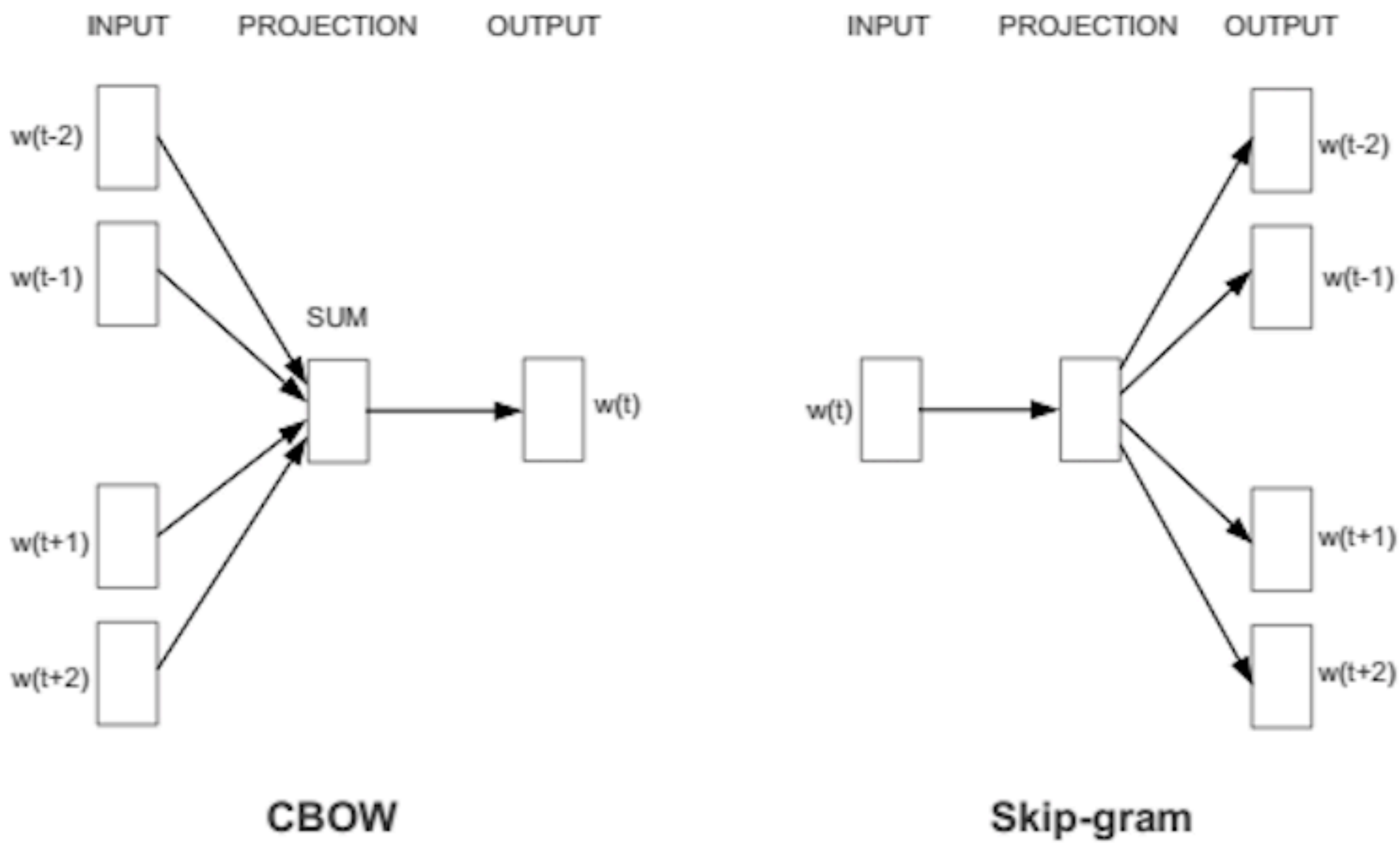
https://github.com/eclipse/deeplearning4j-examples/blob/master/dl4j-examples/src/main/java/org/deeplearning4j/examples/recurrent/word2vecsentiment/Word2VecSentimentRNN.java



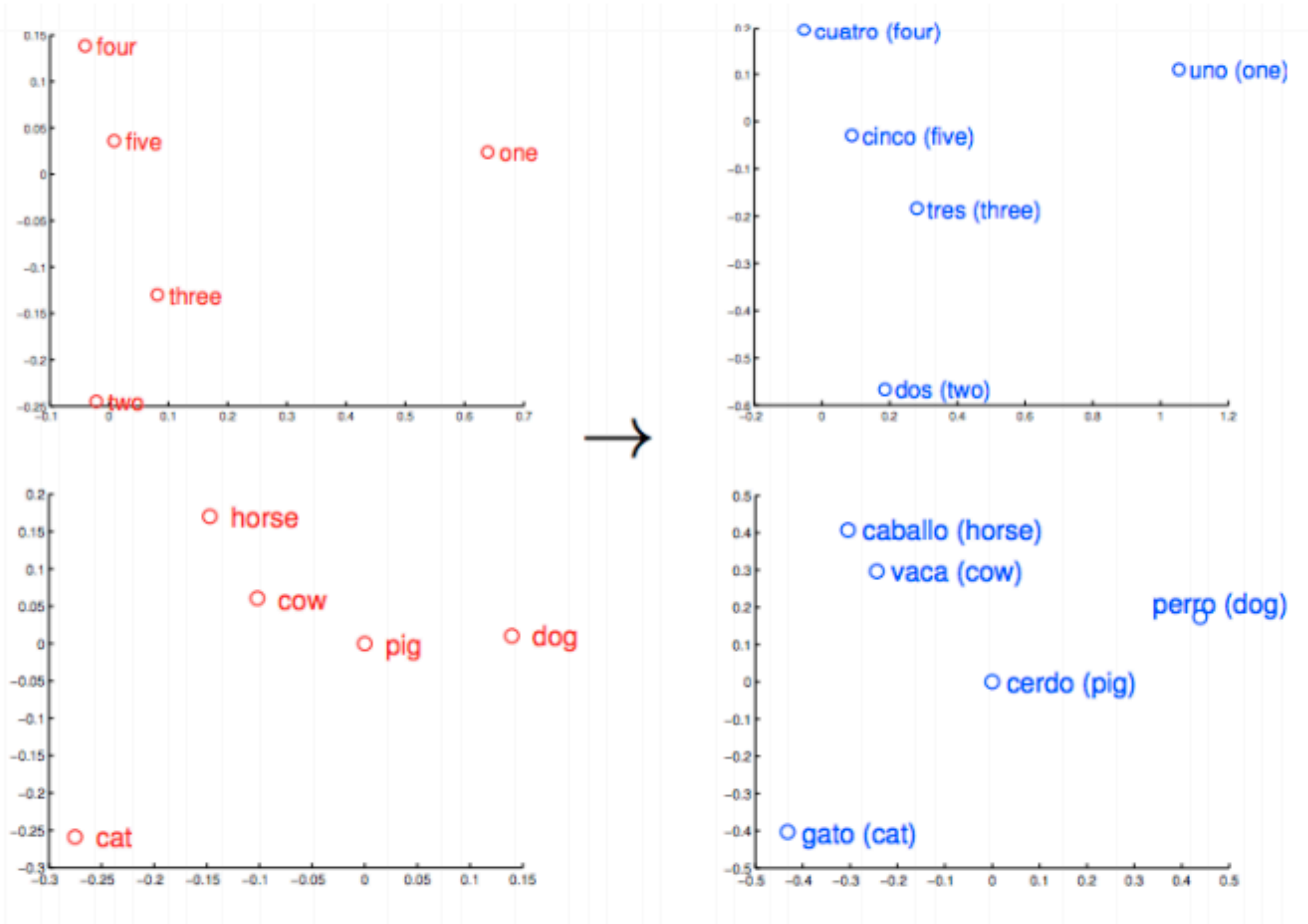

1. Take a 3 layer neural network. (1 input layer + 1 hidden layer + 1 output layer)
2. Feed it a word and train it to predict its neighboring word.
3. Remove the last (output layer) and keep the input and hidden layer.
4. Now, input a word from within the vocabulary. The output given at the hidden layer is the 'word embedding' of the input word.

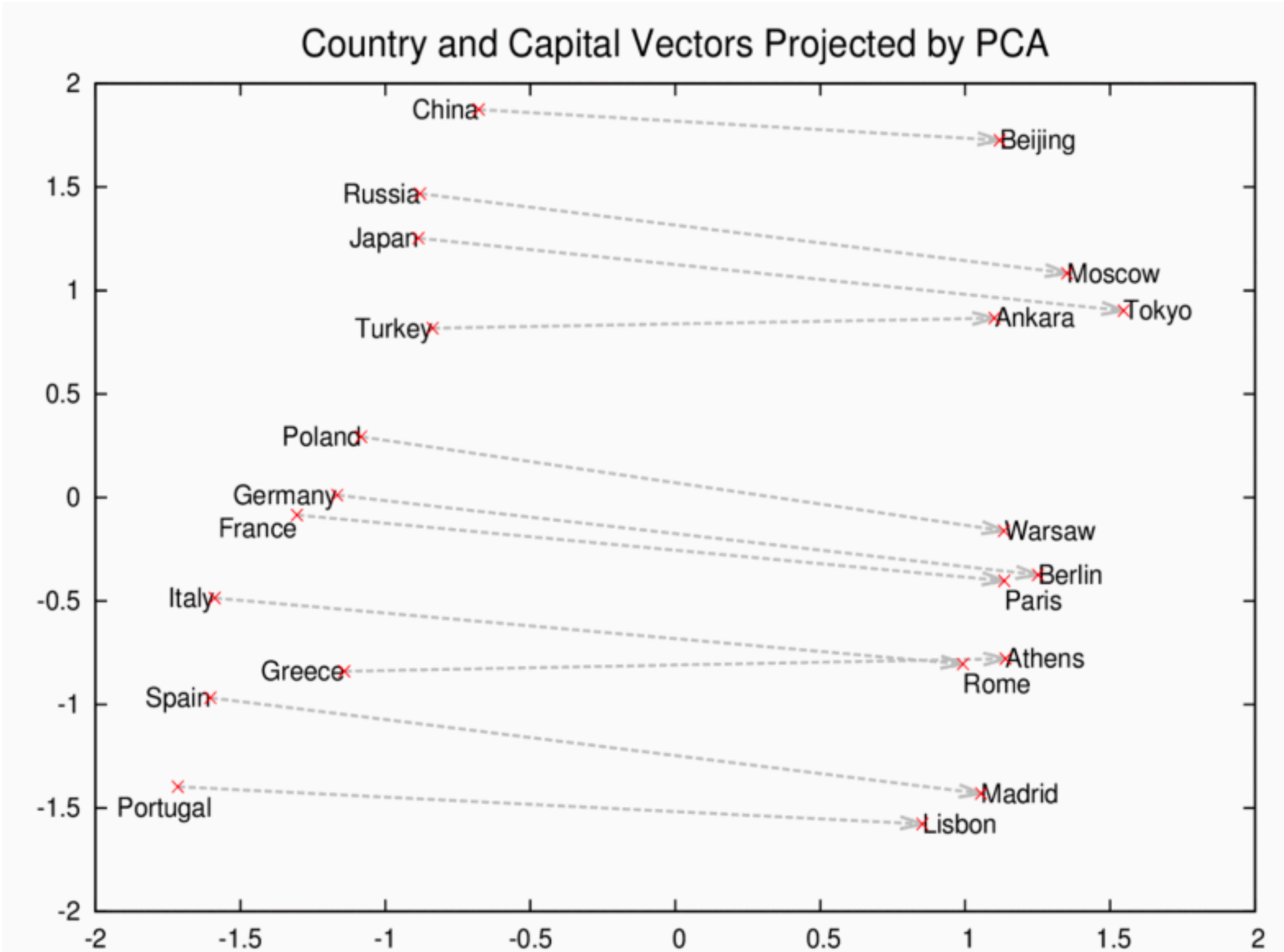
Other optimization: negative sampling, etc.

# IDEA BEHIND WORD2VEC




(continuous bag of words)


four
five
one
three
two
cuatro (four)
uno (one)
cinco (five)
tres (three)
dos (two)
horse
cow
pig
dog
cat
caballo (horse)
vaca (cow)
perro (dog)
cerdo (pig)
gato (cat)




Rome -Italy + China would return Beijing (same distance in vector space)

# CASE STUDY FOR THIS CLASS

You work at Nickelback Inc.

Nickelback Inc recently downloaded every song text ever written (TB of data) to draw inspiration as they lately have trouble to produce a number 1 hit.

Now they want to create a system which enables them to search through this large collection of text and help them to write some songs.

Your task:

Task1: Design a system that efficiently finds all song texts contain certain keywords (e.g., "mountain" and "grass")

Task2: Create a simple ranking for the query results and enable that Nickelback can cluster the songs

Task3: Extend the system to allow search with sentiments (e.g., all happy songs, sad songs,…)

Task4: Extend the system further to find songs with the right meaning of "grass" (the green stuff in the football stadium)

Task5: Develop an assistant that helps Nickelback to write songs by predicting the next sentence

# WHAT IS THE PROBLEM WITH WORD EMBEDDINGS?

The mountain has a lot of **grass**

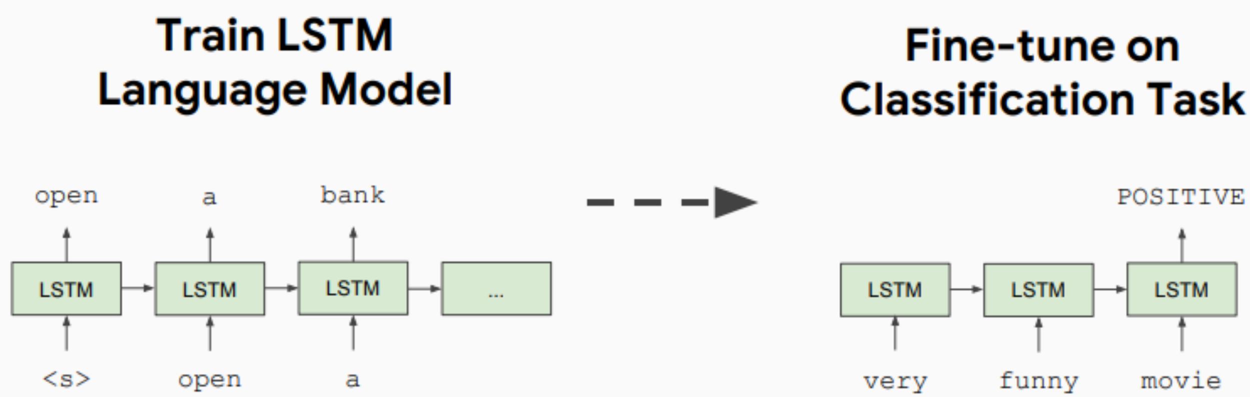
You should never smoke **grass**

same word embedding [0.99, 0.8, …]

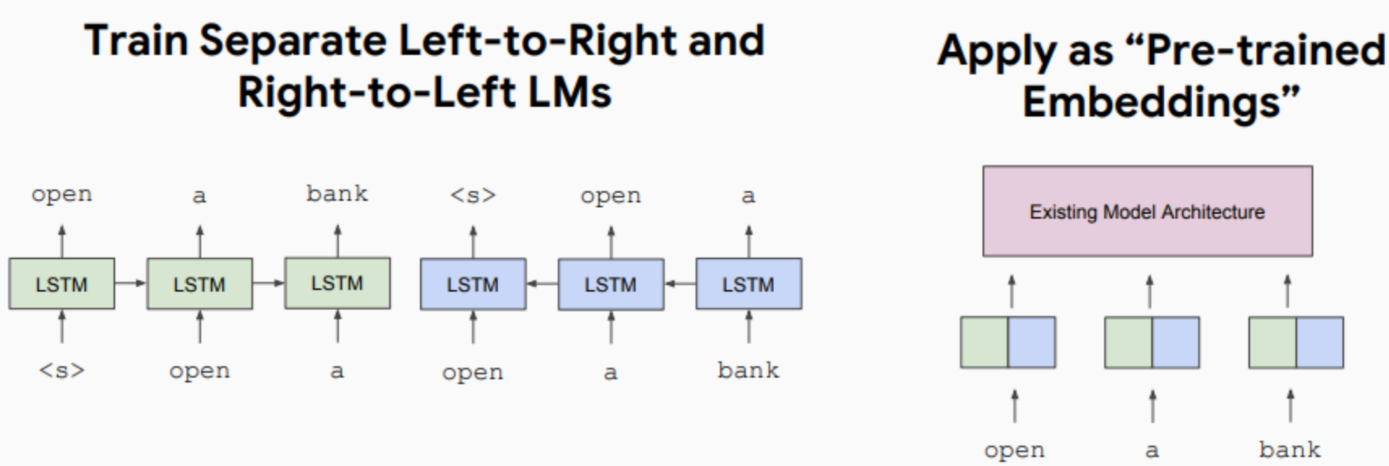
Solution: Train contextual representations on text corpus

# LITTLE HISTORY

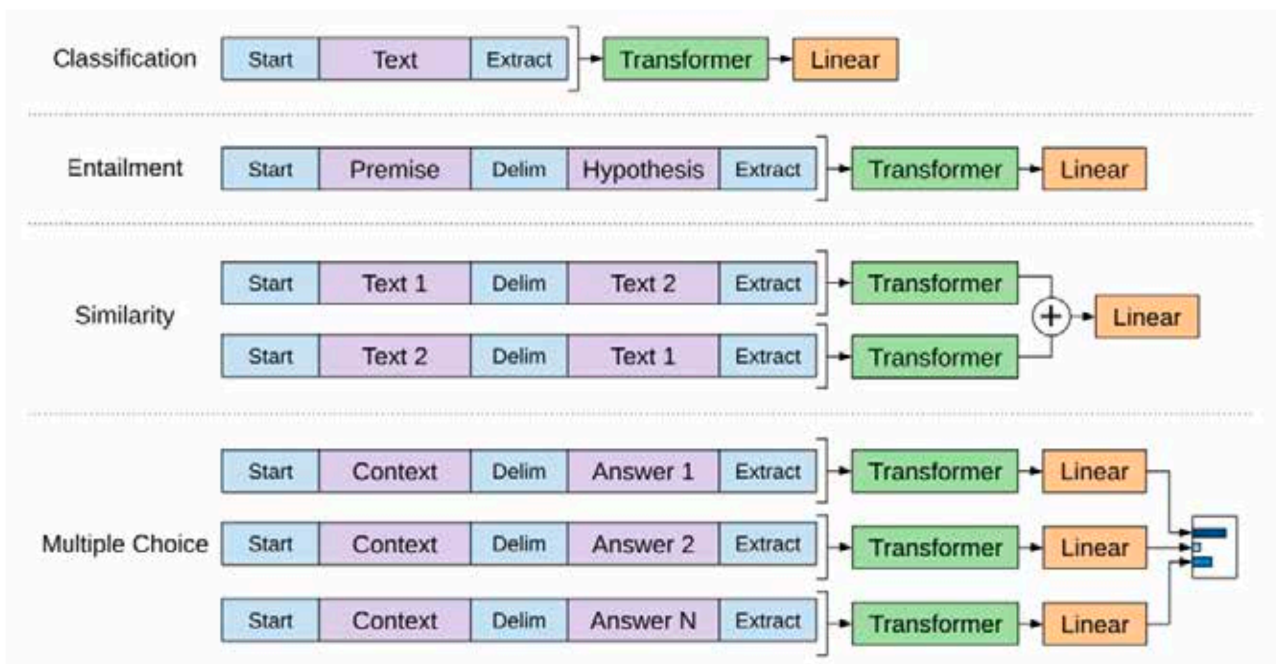
Semi-Supervised Sequence Learning, Google, 2015




ELMo: Deep Contextual Word Embeddings, AI2 &
University of Washington, 2017



# Improving Language Understanding by Generative Pre-Training, OpenAI, 2018 – Based on transformers/attention from "Attention is All You Need" Vaswani et al

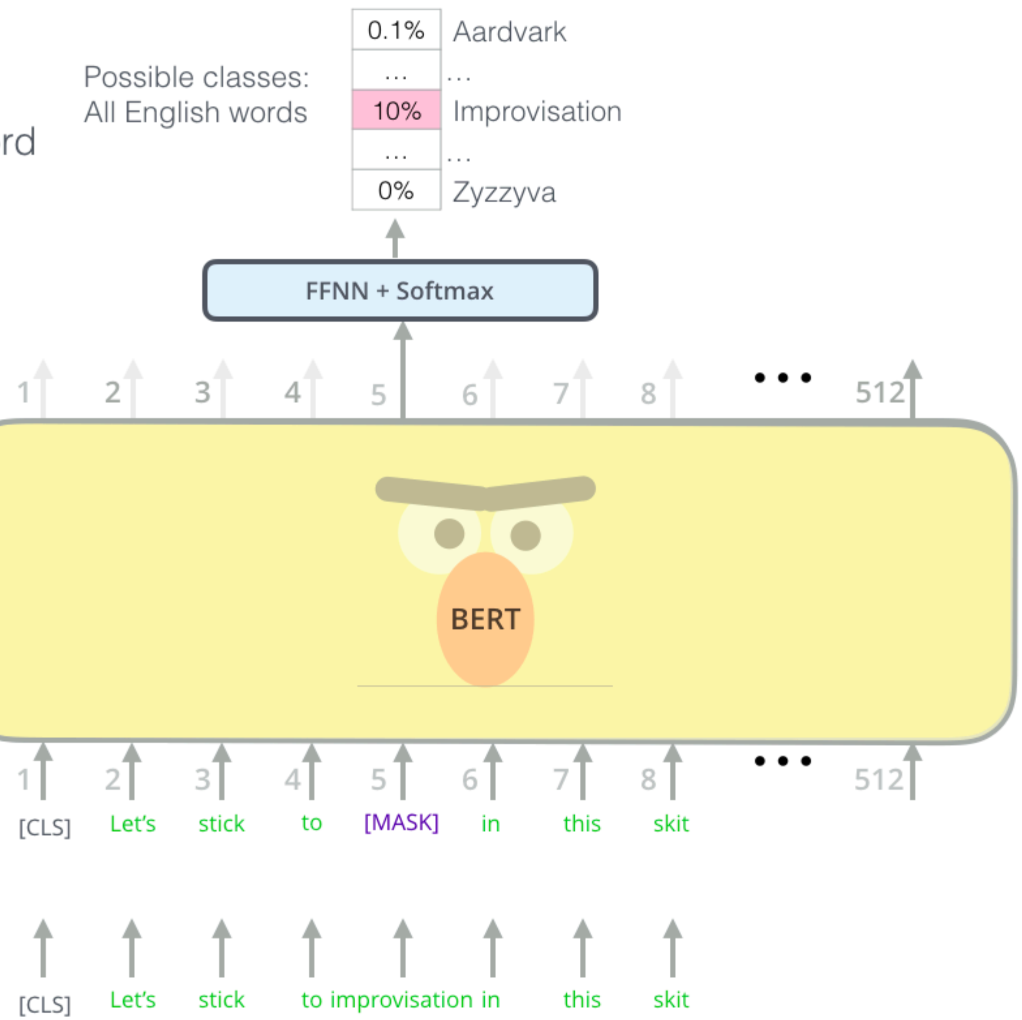



# BERT

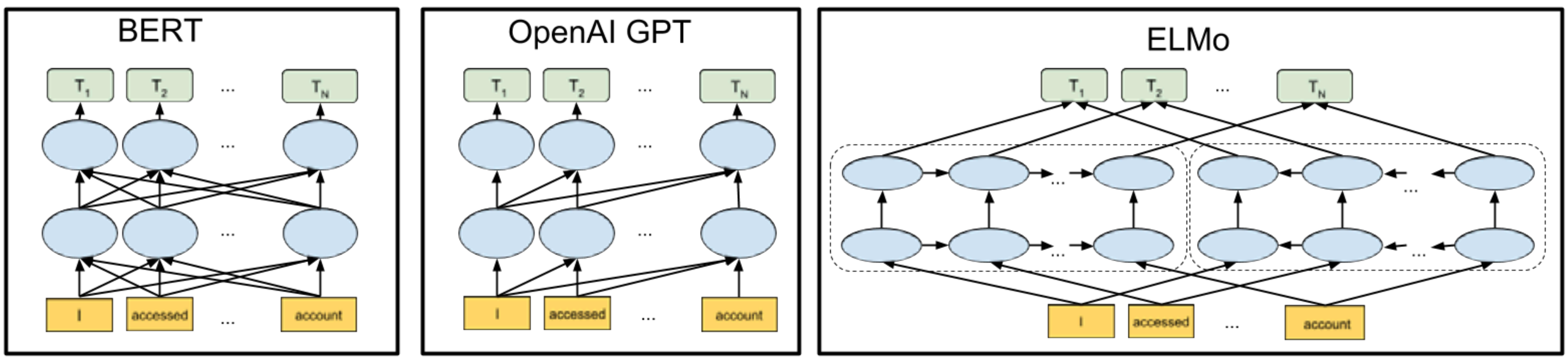
Use the output of the
masked word's position
to predict the masked word

Randomly mask
15% of tokens

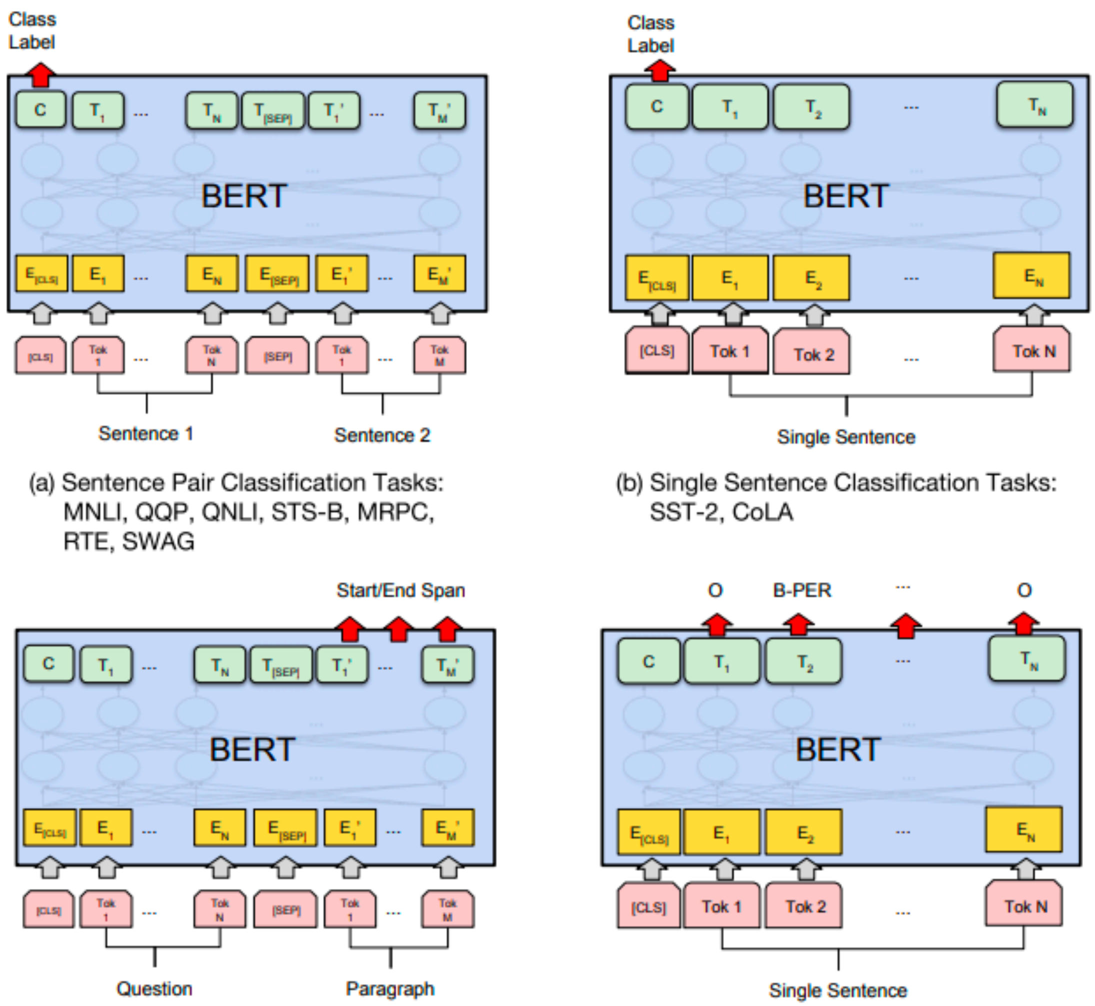
Input



# BERT VS OPENAI GPT VS ELMO



# TASKS




(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

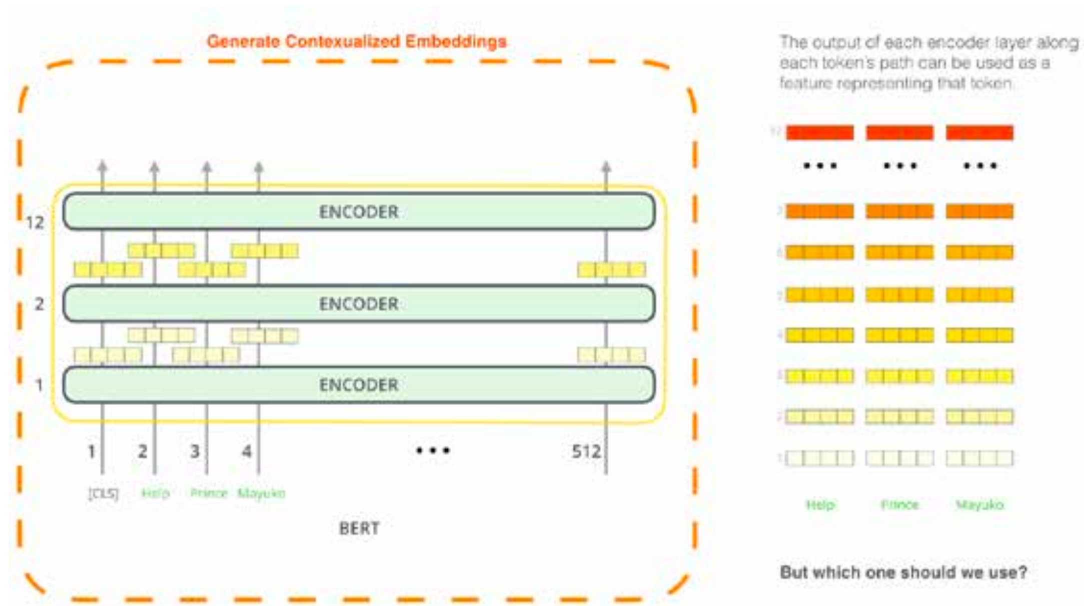
(b) Single Sentence Classification Tasks: SST-2, CoLA

(c) Question Answering Tasks: SQuAD v1.1

(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

http://www.msmarco.org/leaders.aspx

# BERT FOR FEATURE EXTRACTION




# MICROSOFT MARCO DATASETS

## KeyPhrase Extraction(10/18/2019) ranked by F1 @3 on Eval

| Rank | Model | Submission Date | Precision @1,@3,@5 | Recall @1,@3,@5 | F1 @1,@3,@5 |
|---|---|---|---|---|---|
| 1 | **BERT (Base) Sequence Tagging** Si Sun (Tsinghua University), Chenyan Xiong (MSR AI), Zhiyuan Liu (Tsinghua University) [Code] | November 5th, 2019 | 0.484, 0.312, 0.227 | 0.255, 0.469, 0.563 | 0.321, **0.361**, 0.314 |
| 2 | **Baseline finetuned on Bing Queries** MSMARCO Team | October 19th, 2019 | 0.397, 0.249, 0.149 | 0.215, 0.391, 0.391 | 0.267, **0.292**, 0.209 |
| 3 | **Baseline** MSMARCO Team | October 19th, 2019 | 0.365, 0.237, 0.142 | 0.196, 0.367, 0.367 | 0.244, **0.277**, 0.198 |

## Passage Retrieval(10/26/2018-Present) ranked by MRR on Eval

| Rank | Model | Ranking Style | Submission Date | MRR@10 On Eval | MRR@10 On Dev |
|---|---|---|---|---|---|
| 1 | **Enriched BERT base + AOA index + CAS** Ming Yan of Alibaba Damo NLP | Full Ranking | August 20th, 2019 | 0.393 | 0.408 |
| 2 | **W-Index retrieval + BERT-F re-rank** Zhuyun Dai of Carnegie Mellon University | Full Ranking | September 12th,2019 | 0.388 | 0.394 |
| 3 | **Enriched BERT base + AOA index V1** Ming Yan of Alibaba Damo NLP | Full Ranking | May 13th, 2019 | 0.383 | 0.397 |

## Q&A Task(03/01/2018-Present)

| Rank | Model | Submission Date | Rouge-L | Bleu-1 |
|---|---|---|---|---|
| 1 | **Multi-doc Enriched BERT** Ming Yan of Alibaba Damo NLP | June 20th, 2019 | 0.540 | 0.565 |
| 2 | **Human Performance** | April 23th, 2018 | 0.539 | 0.485 |
| 3 | **BERT Encoded T-Net** Y. Zhang, C. Wang, X.L. Chen | August 5th, 2019 | 0.526 | 0.539 |

## Q&A + Natural Language Generation Task(03/01/2018-Present)

| Rank | Model | Submission Date | Rouge-L | Bleu-1 |
|---|---|---|---|---|
| 1 | **Human Performance** | April 23th, 2018 | 0.632 | 0.530 |
| 2 | **Masque NLGEN Style** NTT Media Intelligence Laboratories [Nishida et al. '19] | January 3rd, 2019 | 0.496 | 0.501 |
| 3 | **BERT+ Multi-Pointer-Generator** Tongjun Li of the ColorfulClouds Tech and BUPT | June 11th,2019 | 0.495 | 0.476 |

# GOOGLE IS NOW USING BERT