ADAM SPANNBAUER

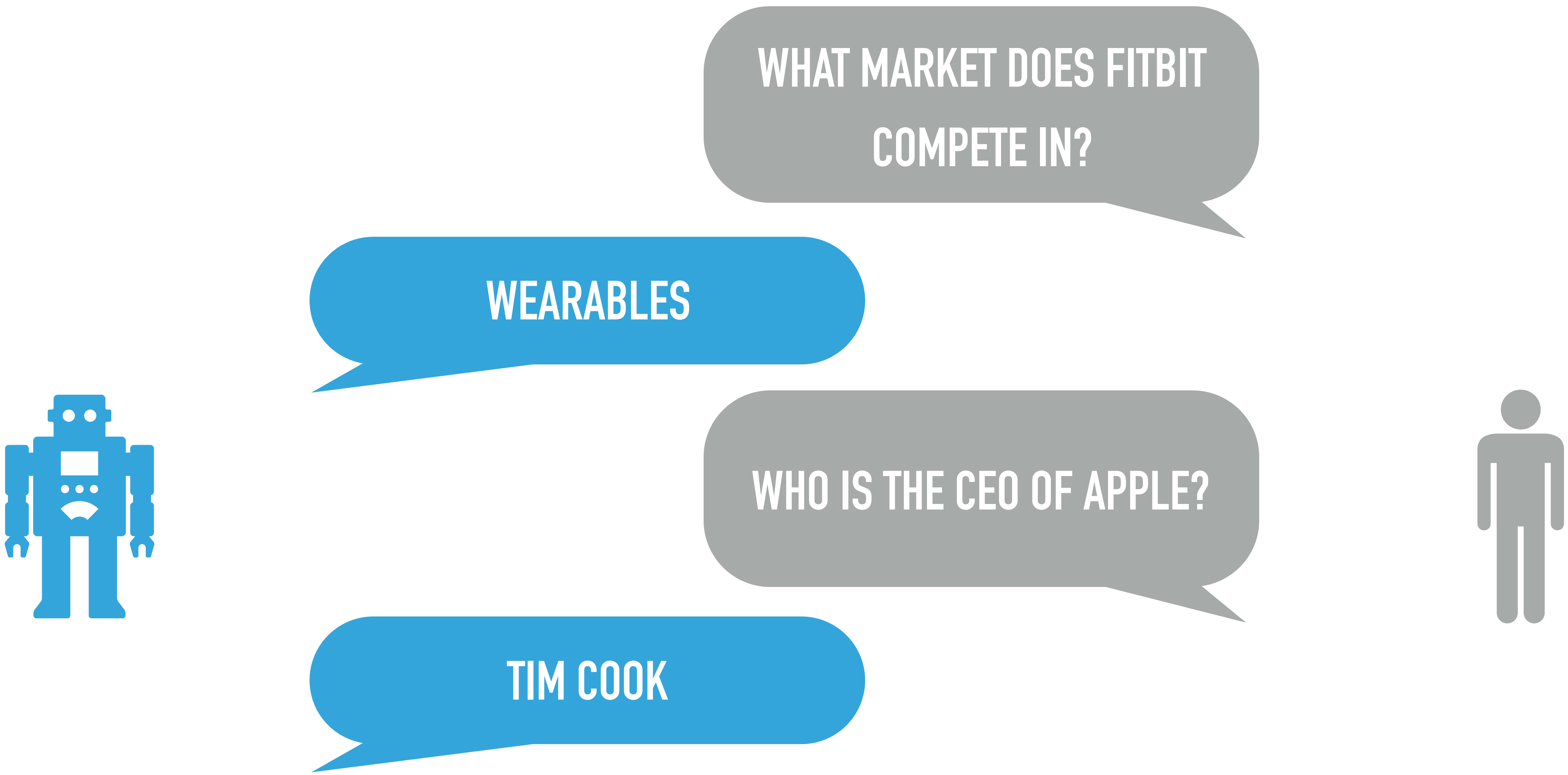
NATURAL LANGUAGE Q&A SEARCH ENGINE

MATERIALS

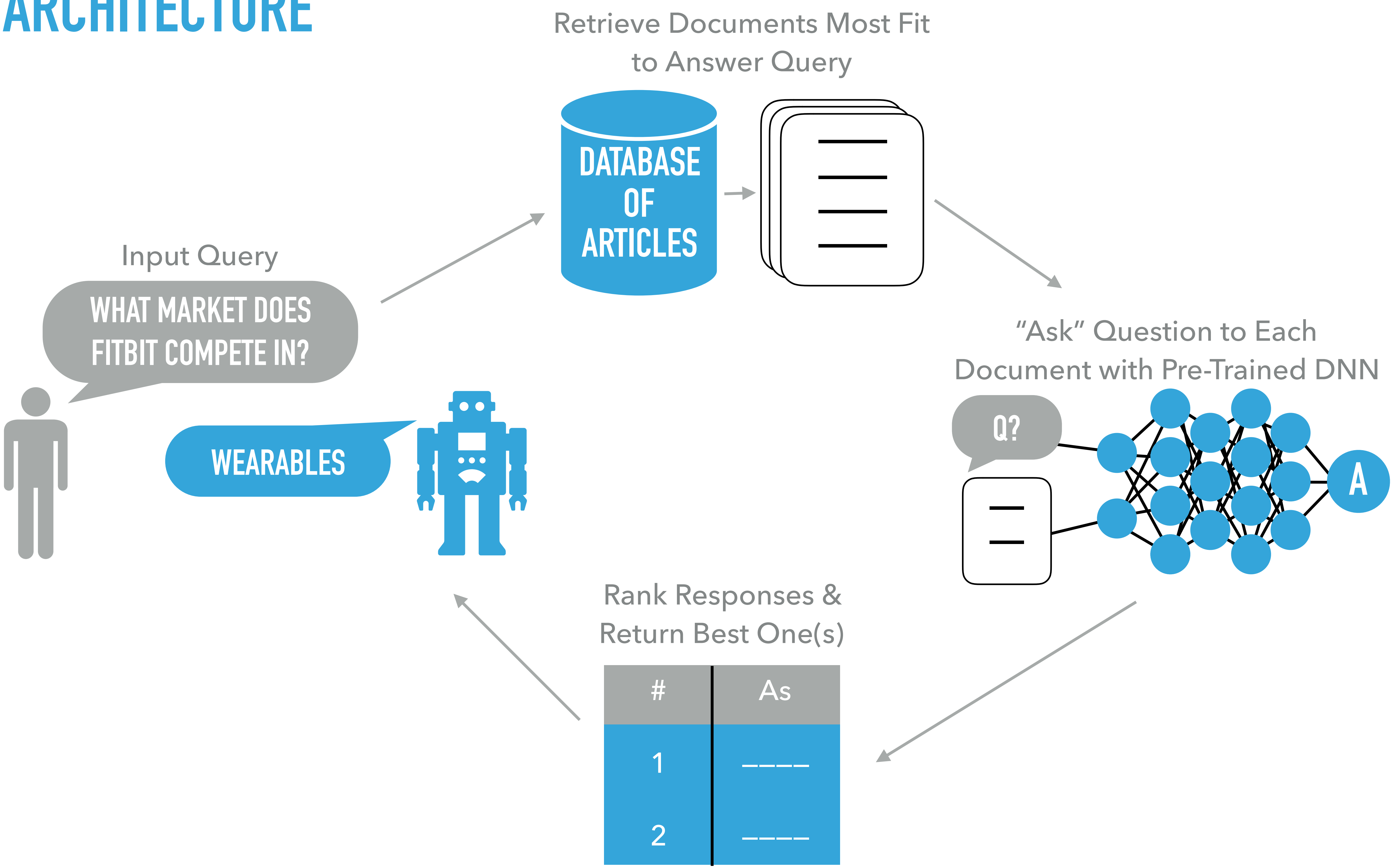
All data & source code used can be found at:

https://github.com/AdamSpannbauer/qa_query

GOAL OF PROJECT

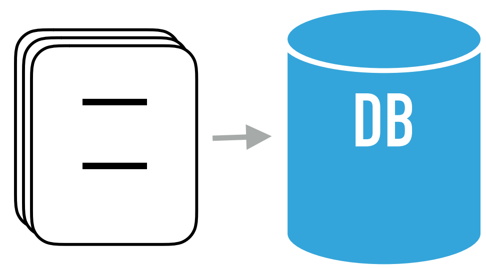


ARCHITECTURE



STEP 0

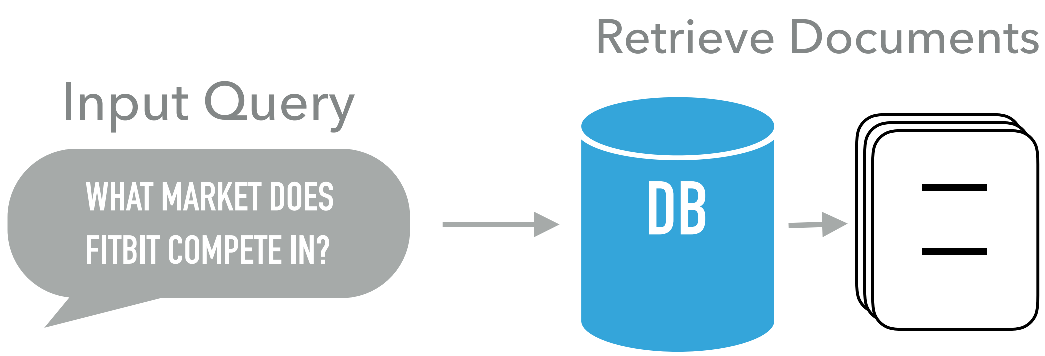
Populate Database with
Articles to Search



This won't be covered in the scope of this session.

Articles were scraped from Nasdaq; for simplicity these articles were stored as JSON files in this project's repo.

STEP 1 – OVERVIEW

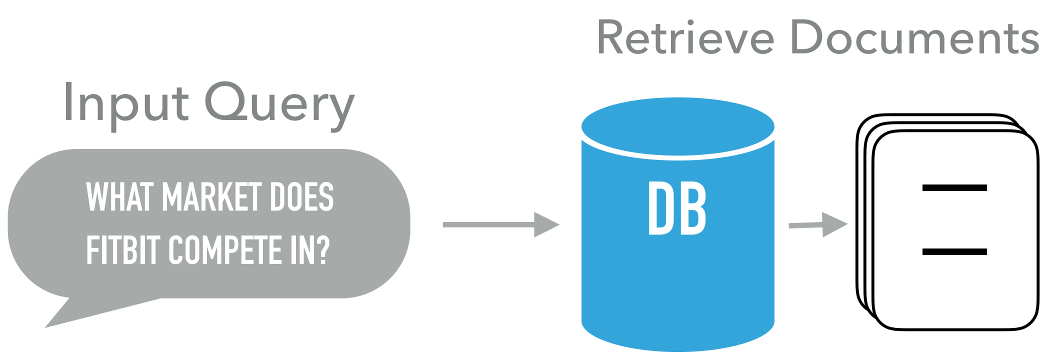


This problem is nothing new and is known as Information Retrieval (IR)

> *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).**

*Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, Introduction to Information Retrieval, Cambridge University Press. 2008

STEP 1 – TOOL SELECTION



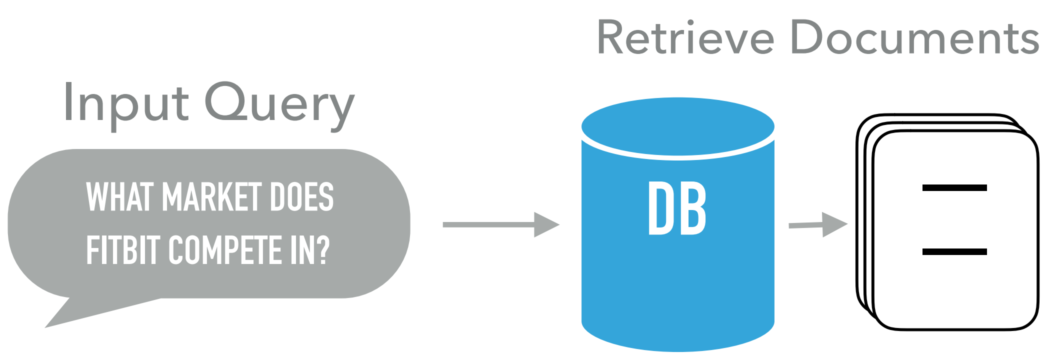
Whenever a problem is nothing new, a new solution might not be best solution for the problem.

Some possible solutions for performing IR (not exhaustive):

- Elasticsearch
- MongoDB Text Index
- Whoosh*

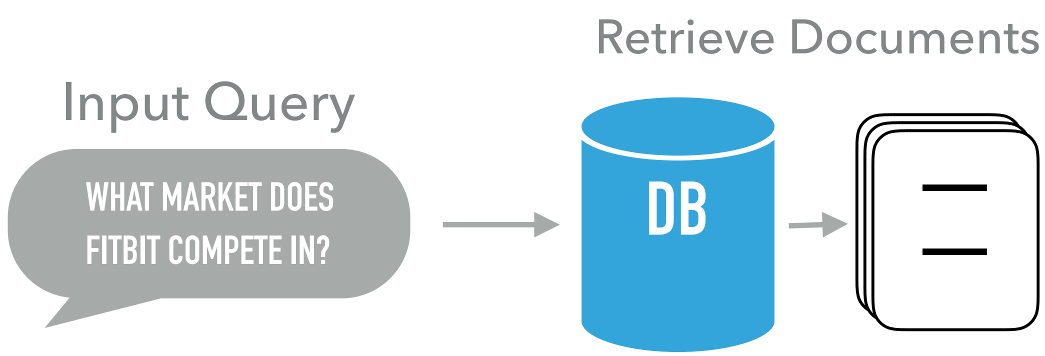
*pure Python and will be our solution today; this was chosen mostly based on simplicity

STEP 1 – GETTING STARTED WITH WHOOSH



> *Whoosh was created by Matt Chaput. It started as a quick and dirty search server for the online documentation of the Houdini 3D animation software package. Side Effects Software generously allowed Matt to open source the code in case it might be useful to anyone else who needs a very flexible or pure-Python search engine (or both!).*

STEP 1 – HOW WHOOSH WORKS

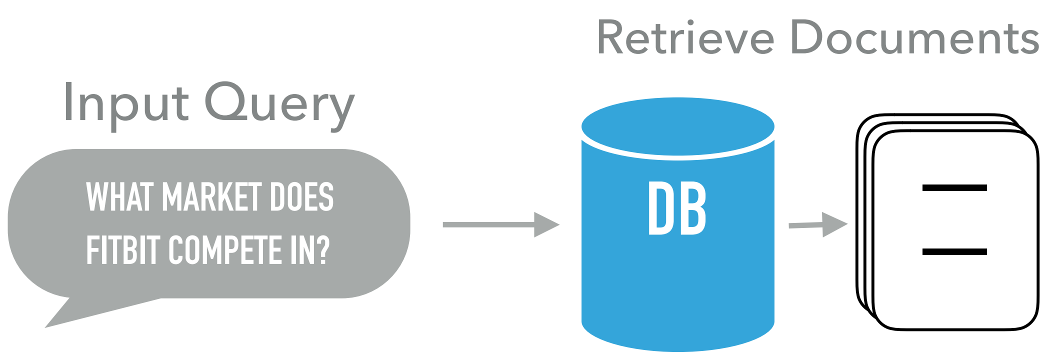


Whoosh uses Okapi BM25, to *estimate* and rank each document's relevance to an input query. The term estimate is used in this definition because BM25 is a probabilistic approach to scoring documents.

> *A relevance score, according to probabilistic information retrieval, ought to reflect the probability a user will consider the result relevant.**

*Doug Turnbull BM25 The Next Generation of Lucene Relevance

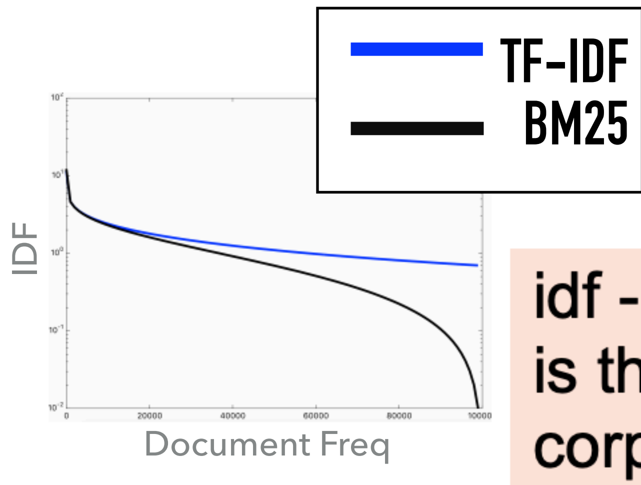
STEP 1 – OKAPI BM25 OVERVIEW



Main Components of Interest:

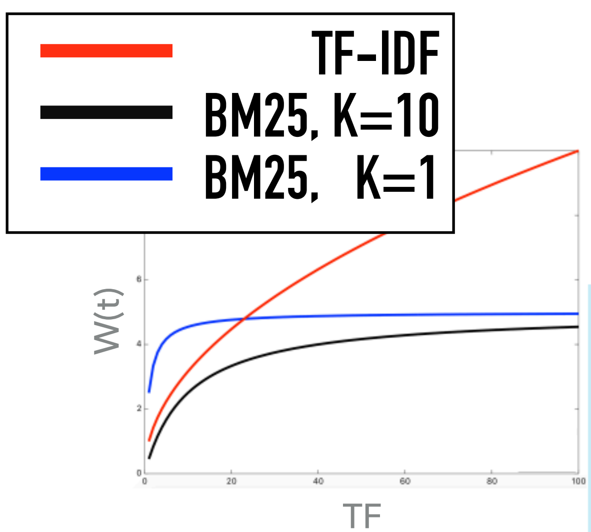
- Term Frequency (TF) - Words mentioned a lot in a document are more important
- Inverse Document Frequency (IDF) - Words mentioned too often in a corpus are less important (i.e. “the”, “a”, etc.)
- Document Length - A word appearing 20 times in a book isn’t as meaningful as a word appearing 20 times in a tweet

STEP 1 – OKAPI BM25 MATH



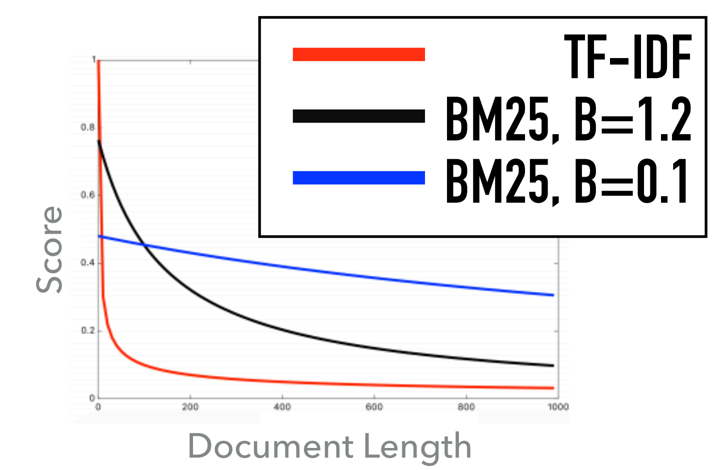
idf - how popular is the term in the corpus?

$$\mathrm{bm25}(d) = \sum_{t \in q, f_{t,d}>0} \log\left(1 + \frac{N - df_t + 0.5}{df_t + 0.5}\right) \cdot \frac{f_{t,d}}{f_{t,d} + k \cdot \left(1 - b + b\frac{\mathrm{l}(d)}{avgdl}\right)}$$



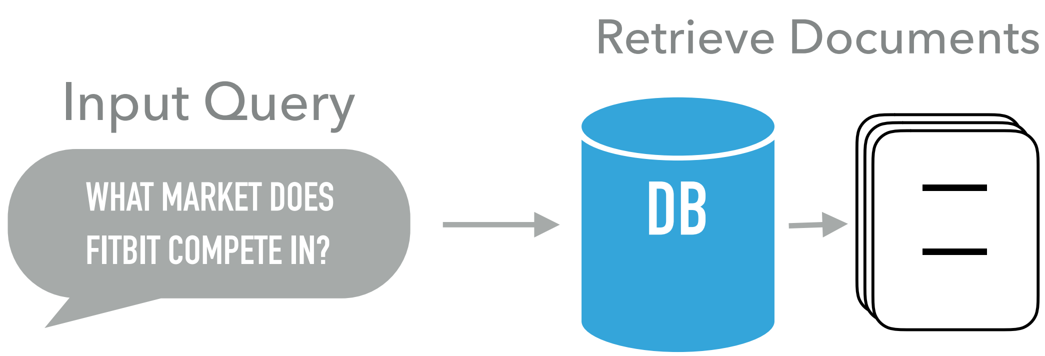
saturation curve - limit influence of tf on the score

length weighing - tweak influence of document length



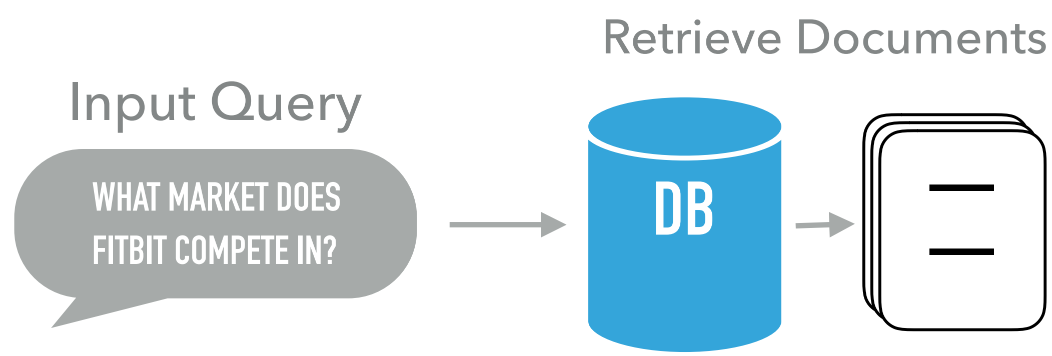
*Slide adapted from: BM25 Demystified by Britta Weber
(highly recommended for a deeper dive)

STEP 1 – OKAPI BM25 PRACTICAL APPLICATION KNOWLEDGE



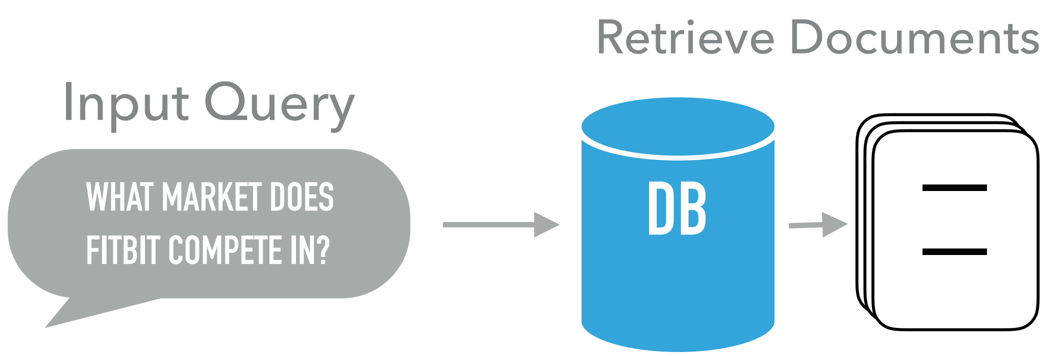
- Works best for smaller documents; think blog post-ish size and smaller
- Already implemented for us in Whoosh and other tools (i.e. Elasticsearch)

STEP 1 – WHOOSH DESIGN




```
{
  "url": "https://www.nasdaq.com/article/rcl-crosses-above-key-moving-average-level-cm1210547",
  "title": "RCL Crosses Above Key Moving Average Level - Nasdaq.com",
  "text": [
    "In trading on Wednesday, shares of Royal Caribbean Cruises Ltd (Symbol: RCL) crossed above their 200 day moving average of $114.09, changing hands as high as $114.38 per share. Royal Caribbean Cruises Ltd shares are currently trading down about 0.2% on the day. The chart below shows the one year performance of RCL shares, versus its 200 day moving average:     Looking at the chart above, RCL's low point in its 52 week range is $89.48 per share, with $133.60 as the 52 week high point - that compares with a last trade of $113.53.   Click here to find out which 9 other dividend   stocks recently crossed above their 200 day moving average \u00bb  ",
    "  Click here to find out which 9 other dividend   stocks recently crossed above their 200 day moving average \u00bb ",
    "\nThe views and opinions expressed herein are the views and opinions of the author and do not necessarily reflect those of Nasdaq, Inc.\n"
  ],
  "published_datetime": "2019-09-11 11:10:01"
}
```

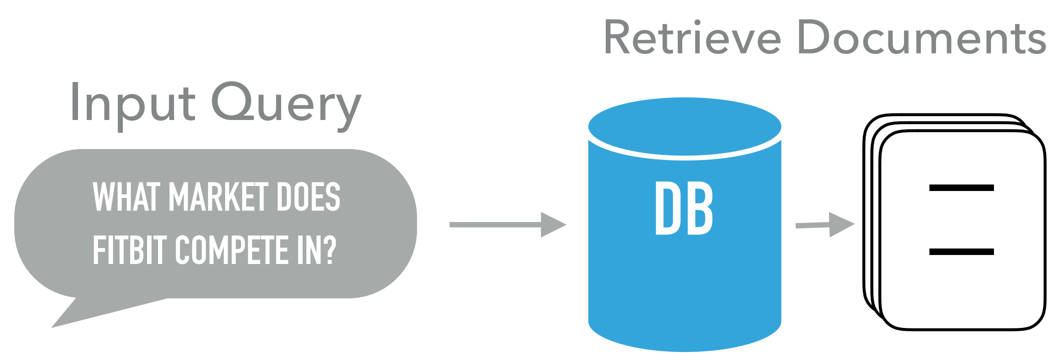
STEP 1 – WHOOSH DESIGN



Fields relevant for TEXT search


```
{
  "url": "https://www.nasdaq.com/article/rcl-crosses-above-key-moving-average-level-cm1210547",
  "title": "RCL Crosses Above Key Moving Average Level - Nasdaq.com",
  "text": [
    "In trading on Wednesday, shares of Royal Caribbean Cruises Ltd (Symbol: RCL) crossed above their 200 day moving average of $114.09, changing hands as high as $114.38 per share. Royal Caribbean Cruises Ltd shares are currently trading down about 0.2% on the day. The chart below shows the one year performance of RCL shares, versus its 200 day moving average:     Looking at the chart above, RCL's low point in its 52 week range is $89.48 per share, with $133.60 as the 52 week high point - that compares with a last trade of $113.53.   Click here to find out which 9 other dividend   stocks recently crossed above their 200 day moving average \u00bb  ",
    "  Click here to find out which 9 other dividend   stocks recently crossed above their 200 day moving average \u00bb ",
    "\nThe views and opinions expressed herein are the views and opinions of the author and do not necessarily reflect those of Nasdaq, Inc.\n"
  ],
  "published_datetime": "2019-09-11 11:10:01"
}
```

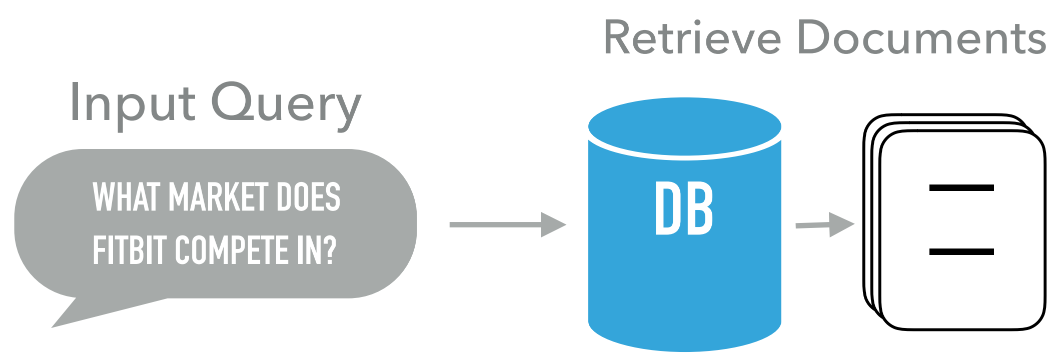
STEP 1 – WHOOSH DESIGN



ID fields


```
{
  "url": "https://www.nasdaq.com/article/rcl-crosses-above-key-moving-average-level-cm1210547",
  "title": "RCL Crosses Above Key Moving Average Level - Nasdaq.com",
  "text": [
    "In trading on Wednesday, shares of Royal Caribbean Cruises Ltd (Symbol: RCL) crossed above their 200 day moving average of $114.09, changing hands as high as $114.38 per share. Royal Caribbean Cruises Ltd shares are currently trading down about 0.2% on the day. The chart below shows the one year performance of RCL shares, versus its 200 day moving average:     Looking at the chart above, RCL's low point in its 52 week range is $89.48 per share, with $133.60 as the 52 week high point - that compares with a last trade of $113.53.   Click here to find out which 9 other dividend   stocks recently crossed above their 200 day moving average \u00bb  ",
    "  Click here to find out which 9 other dividend   stocks recently crossed above their 200 day moving average \u00bb ",
    "\nThe views and opinions expressed herein are the views and opinions of the author and do not necessarily reflect those of Nasdaq, Inc.\n"
  ],
  "published_datetime": "2019-09-11 11:10:01"
}
```

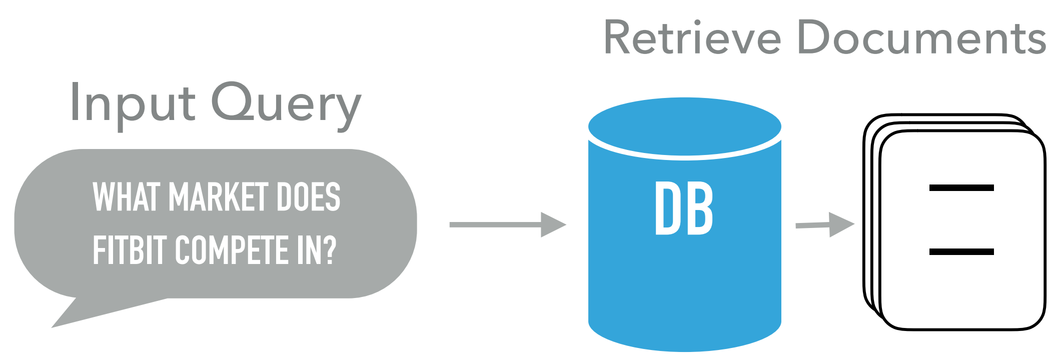
STEP 1 – WHOOSH SCHEMA




```
from whoosh.fields import Schema, TEXT, ID
from whoosh.analysis import StandardAnalyzer

schema = Schema(
        url=ID(stored=True, unique=True),
        published_datetime=ID(stored=True),
        title=TEXT(stored=True, analyzer=StandardAnalyzer()),
        article=TEXT(stored=True, analyzer=StandardAnalyzer()),
    )
```

- Fields to be searched are defined with `TEXT()` & others are defined with `ID()`.

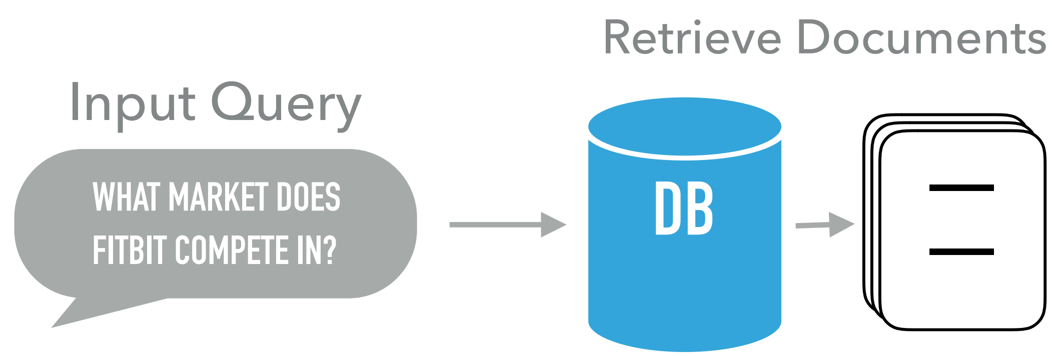
STEP 1 – WHOOSH SCHEMA




```
from whoosh.fields import Schema, TEXT, ID
from whoosh.analysis import StandardAnalyzer

schema = Schema(
        url=ID(stored=True, unique=True),
        published_datetime=ID(stored=True),
        title=TEXT(stored=True, analyzer=StandardAnalyzer()),
        article=TEXT(stored=True, analyzer=StandardAnalyzer()),
   )
```

- We can enforce a uniqueness constraint by setting `unique=True`

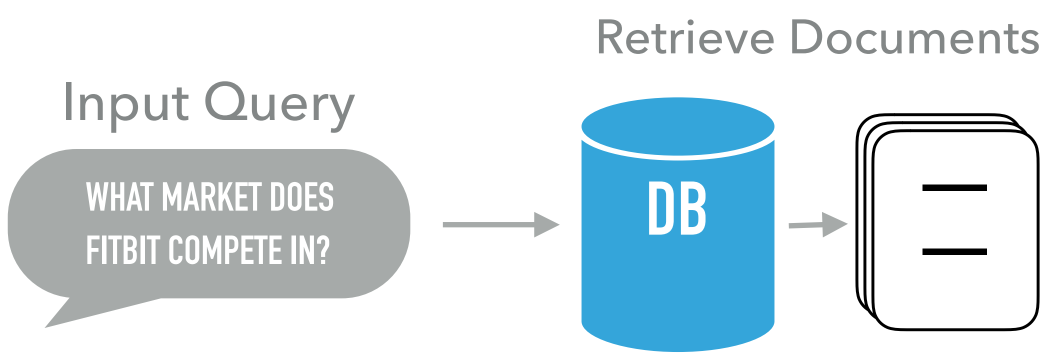
STEP 1 – WHOOSH SCHEMA




```
from whoosh.fields import Schema, TEXT, ID
from whoosh.analysis import StandardAnalyzer

schema = Schema(
        url=ID(stored=True, unique=True),
        published_datetime=ID(stored=True),
        title=TEXT(stored=True, analyzer=StandardAnalyzer()),
        article=TEXT(stored=True, analyzer=StandardAnalyzer()),
   )
```

- Indicating `stored=True` means that the raw version of the field will be saved in the index. For text fields this can be costly since the post-processed text will be much smaller.

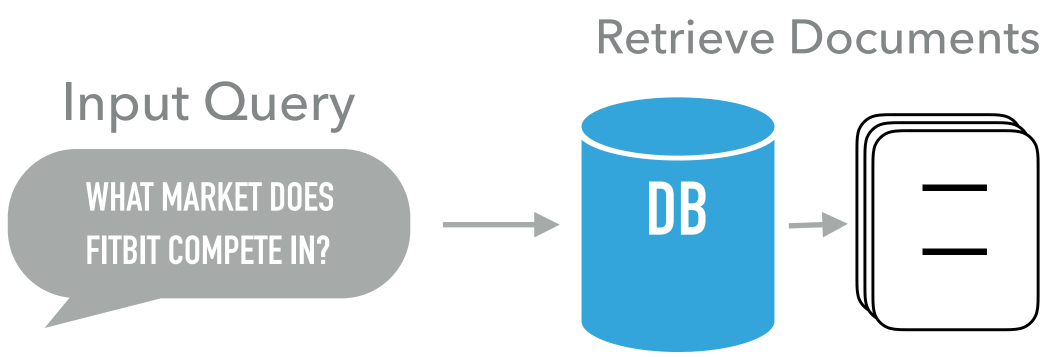
STEP 1 – WHOOSH SCHEMA




```
from whoosh.fields import Schema, TEXT, ID
from whoosh.analysis import StandardAnalyzer

schema = Schema(
        url=ID(stored=True, unique=True),
        published_datetime=ID(stored=True),
        title=TEXT(stored=True, analyzer=StandardAnalyzer()),
        article=TEXT(stored=True, analyzer=StandardAnalyzer()),
    )
```

- Whoosh comes with some pre-defined “analyzers” that will dictate how stored documents and input queries will be processed.

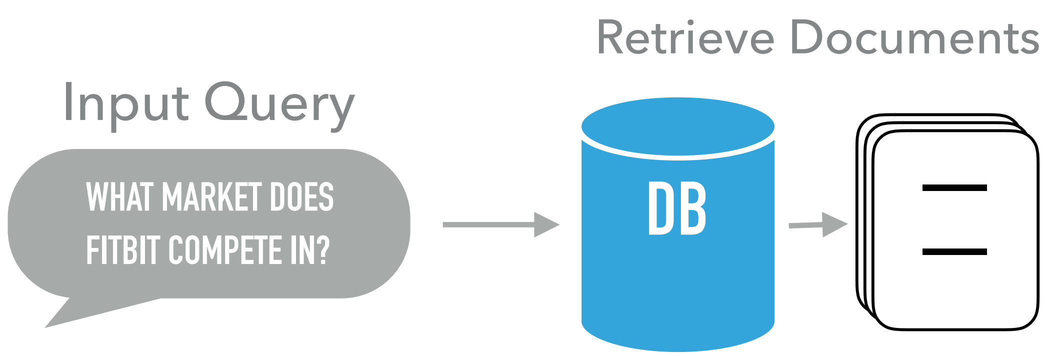
STEP 1 – WHOOSH ANALYZERS




```
>>> from whoosh.analysis import StandardAnalyzer, StemmingAnalyzer
>>>
>>> [token.text for token in StandardAnalyzer()("This is an example analysis")]
['example', ‘analysis']
>>> [token.text for token in StemmingAnalyzer()("This is an example analysis”)]
['exampl', ‘analysi’]
```

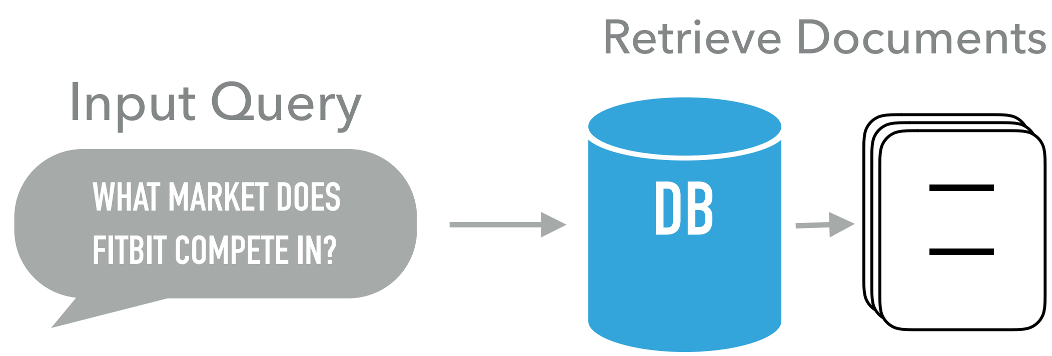
- The `StandardAnalyzer` performs tokenization, lowercasing, & stopword removal by default
- The `StemmingAnalyzer` adds stemming with the [Porter Stemming Algorithm]

STEP 1 – BUILDING THE INDEX



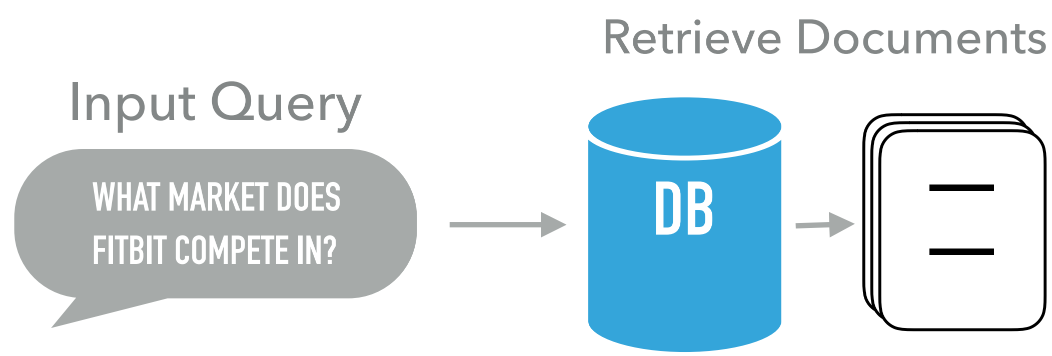

```
schema = Schema(
        url=ID(stored=True, unique=True),
        published_datetime=ID(stored=True),
        title=TEXT(stored=True, analyzer=StemmingAnalyzer()),
        article=TEXT(stored=True, analyzer=StemmingAnalyzer()),
    )

idx = whoosh.index.create_in("index_directory", schema=schema, indexname="nasdaq")

writer = idx.writer()
for i, row in text_df.iterrows():
    writer.update_document(
        url=row['url'],
        published_datetime=row['published_datetime'],
        title=row['title'],
        article=row['text'],
    )
writer.commit()
```

STEP 1 – QUERY PARSING




```
>>> import whoosh.index
... from whoosh.qparser import MultifieldParser, OrGroup, WildcardPlugin
...
... whoosh_idx = whoosh.index.open_dir('whoosh_idx', indexname='nasdaq')
... query_parser = MultifieldParser(['title', 'article'],
                                    schema=whoosh_idx.schema,
                                    group=OrGroup)
... query_parser.remove_plugin_class(WildcardPlugin)
...
... parsed_query = query_parser.parse('What market does FitBit compete in?')
>>> print(parsed_query)
Or([Term('title', 'what'), Term('article', 'what'), Term('title', 'market'),
Term('article', 'market'), Term('title', 'doe'), Term('article', 'doe'), Term('title',
'fitbit'), Term('article', 'fitbit'), Term('title', 'compet'), Term('article',
'compet')])
```

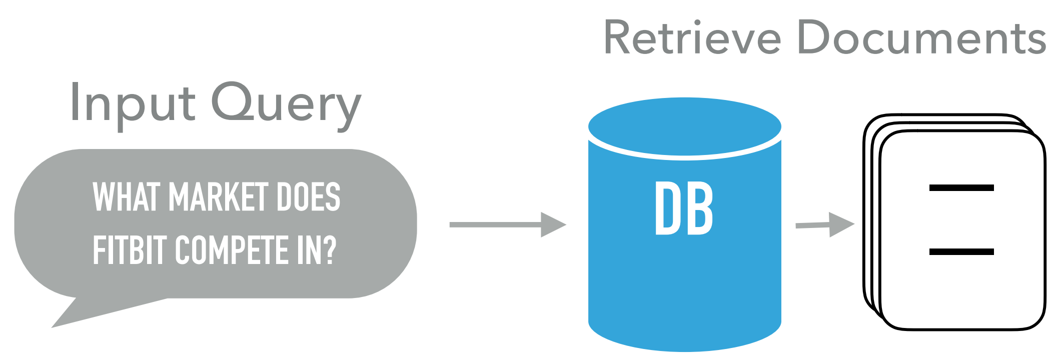
STEP 1 – QUERY PARSING




```
>>> import whoosh.index
... from whoosh.qparser import MultifieldParser, OrGroup, WildcardPlugin
...
... whoosh_idx = whoosh.index.open_dir('whoosh_idx', indexname='nasdaq')
... query_parser = MultifieldParser(['title', 'article'],
                                    schema=whoosh_idx.schema,
                                    group=OrGroup)
... query_parser.remove_plugin_class(WildcardPlugin)
...
... parsed_query = query_parser.parse('What market does FitBit compete in?')
>>> print(parsed_query)
Or([Term('title', 'what'), Term('article', 'what'), Term('title', 'market'),
Term('article', 'market'), Term('title', 'doe'), Term('article', 'doe'), Term('title',
'fitbit'), Term('article', 'fitbit'), Term('title', 'compet'), Term('article',
'compet')])
```

We're building a query parser specifically for questions. Maybe our stopword list should be revisited?

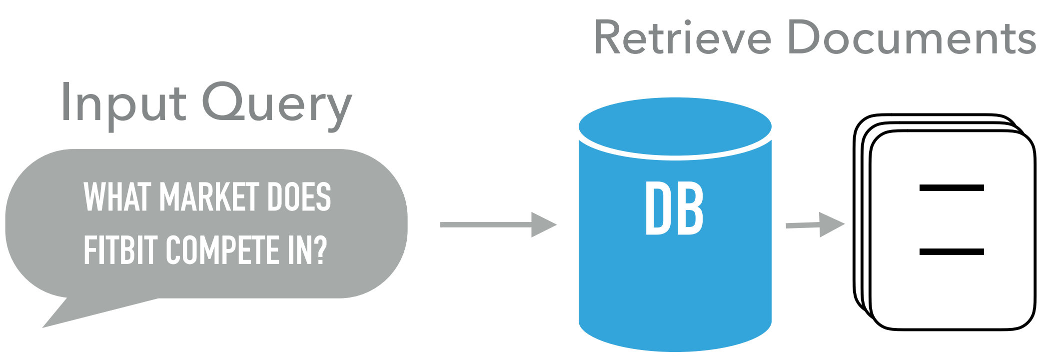
STEP 1 – QUERY PARSING




```
>>> import whoosh.index
... from whoosh.qparser import MultifieldParser, OrGroup, WildcardPlugin
...
... whoosh_idx = whoosh.index.open_dir('whoosh_idx', indexname='nasdaq')
... query_parser = MultifieldParser(['title', 'article'],
                                    schema=whoosh_idx.schema,
                                    group=OrGroup)
... query_parser.remove_plugin_class(WildcardPlugin)
...
... parsed_query = query_parser.parse('What market does FitBit compete in?')
>>> print(parsed_query)
Or([Term('title', 'what'), Term('article', 'what'), Term('title', 'market'),
Term('article', 'market'), Term('title', 'doe'), Term('article', 'doe'), Term('title',
'fitbit'), Term('article', 'fitbit'), Term('title', 'compet'), Term('article',
'compet')])
```

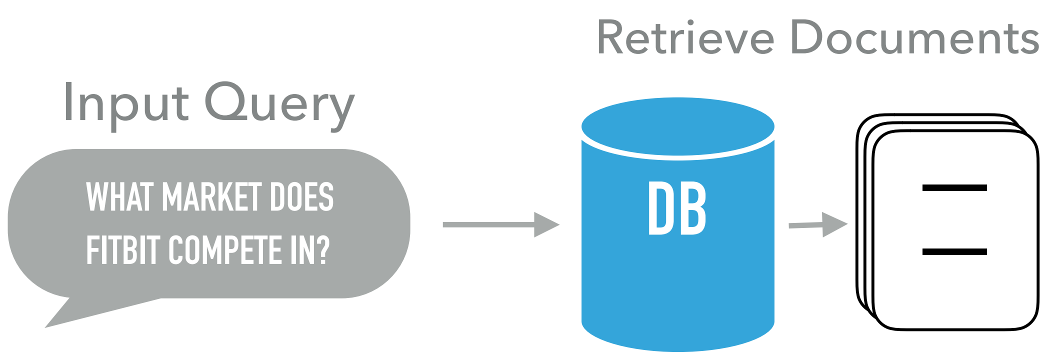
Companies and other "Named Entities" are likely key to a user's queries. Maybe we should give them more importance?

STEP 1 – PERFORMING A SEARCH




```
>>> import whoosh.index
... from whoosh.qparser import MultifieldParser, OrGroup, WildcardPlugin
...
... whoosh_idx = whoosh.index.open_dir('whoosh_idx', indexname='nasdaq')
... query_parser = MultifieldParser(['title', 'article'],
                                    schema=whoosh_idx.schema,
                                    group=OrGroup)
... query_parser.remove_plugin_class(WildcardPlugin)
...
... parsed_query = query_parser.parse('What market does FitBit compete in?')
...
... with whoosh_idx.searcher() as searcher:
...     search_results = searcher.search(parsed_query, limit=1)
...     [print(sr['title']) for sr in search_results]

Where Are Wearables and Smart Speakers in 2019? - Nasdaq.com
```

STEP 1 – CUSTOMIZING STOPWORDS



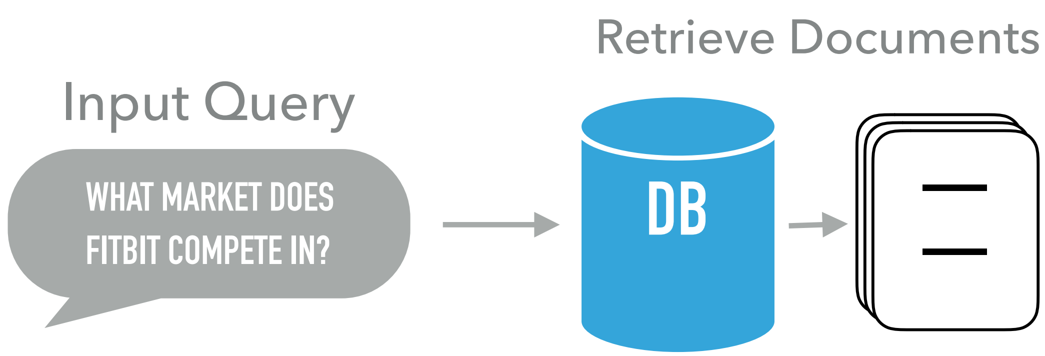

```
import nltk.corpus


NLTK_STOPWORDS = set(nltk.corpus.stopwords.words('english'))
QUESTION_STOPWORDS = {'who', 'what', 'where', 'when', 'why', 'how'}
QA_STOPWORDS = frozenset(QUESTION_STOPWORDS | NLTK_STOPWORDS)
```

Whoosh's stopword list is fairly minimal, here `nltk` is used as a base list and augmented with a custom list of common question words we'd expect.

This new list can be used with our analyzer like so:

```
StemmingAnalyzer(stoplist=QA_STOPWORDS)
```

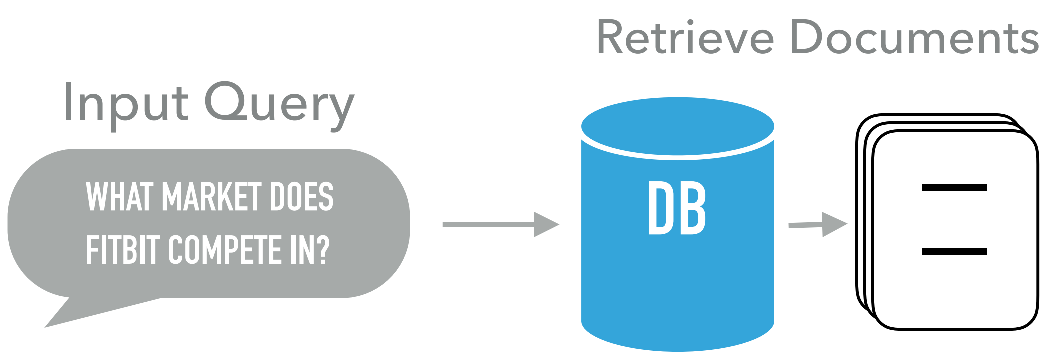
STEP 1 – NAMED ENTITY RECOGNITION



> *Named Entity Recognition (NER) labels sequences of words in a text which are the names of things, such as person and company names…**

*[Stanford Named Entity Recognizer](#)

STEP 1 – NAMED ENTITY RECOGNITION

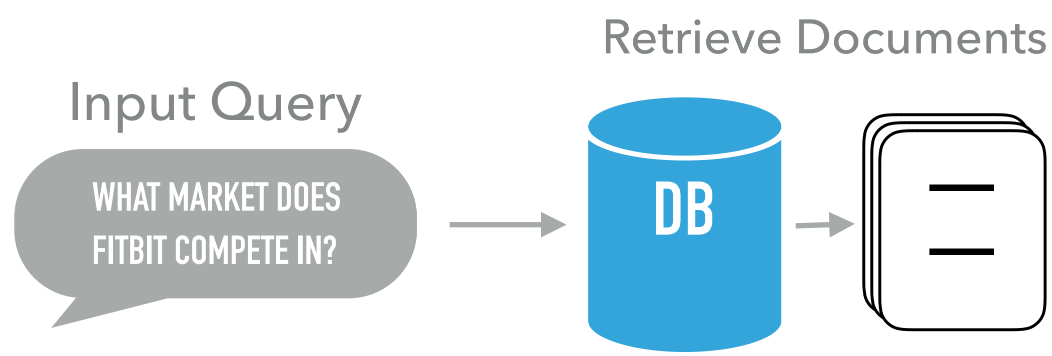


> *Named Entity Recognition (NER) labels sequences of words in a text which are the names of things, such as person and company names…**

What market does FitBit compete in?

*[Stanford Named Entity Recognizer](#)

STEP 1 - NAMED ENTITY RECOGNITION



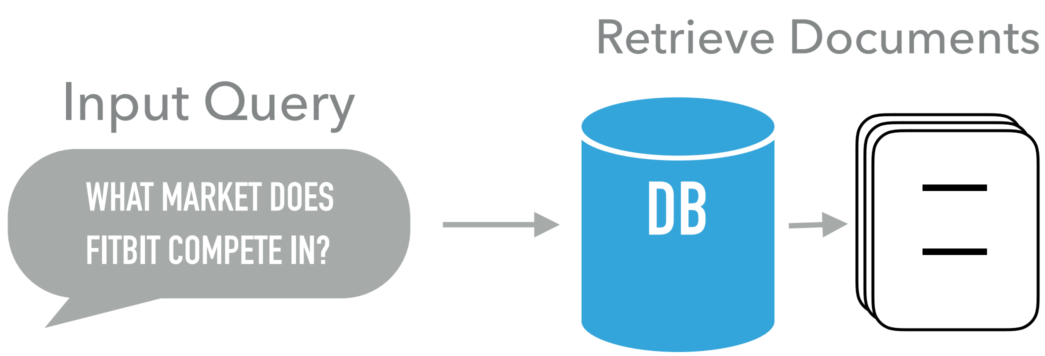
> *Named Entity Recognition (NER) labels sequences of words in a text which are the names of things, such as person and company names…**

Organization

What market does FitBit compete in?

*Stanford Named Entity Recognizer

STEP 1 – PYTHON AND NER

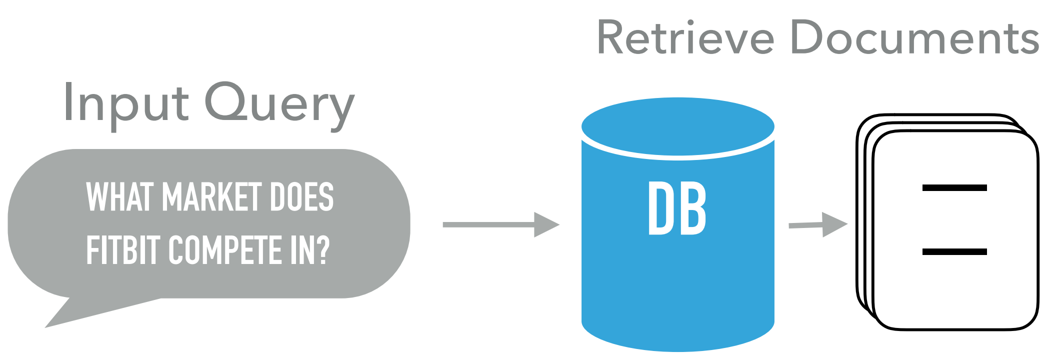


NER is implemented in many different tools, and is still a problem being worked on. A couple of Python implementations can be found in:

- spaCy
- nltk*

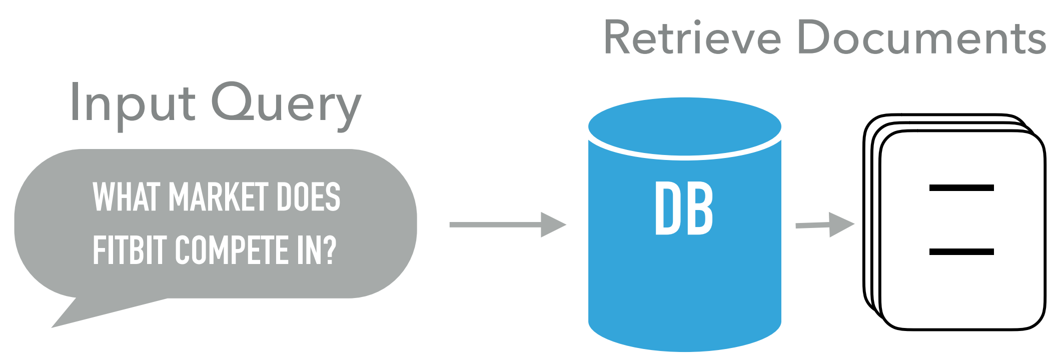
*Will be our choice due to ease of use and already being a dependency in a downstream process we'll see later. spaCy's NER outperforms nltk in general.

STEP 1 – NER IN NLTK




```
QA_NE_TYPES = frozenset(['ORGANIZATION', 'LOCATION', 'FACILITY', 'GPE'])


def ner_extract(text, ne_types=QA_NE_TYPES):
    """Remove non named entities from a string

    :param text: str to remove non named entities from
    :param ne_types: list/set of named entities to keep
    :return: text with non named entities removed
    """
    chunks = nltk.ne_chunk(nltk.pos_tag(nltk.word_tokenize(text)))
    ne_list = []
    for chunk in chunks:
        if hasattr(chunk, 'label'):
            if chunk.label() in ne_types:
                full_ne = ' '.join(c[0] for c in chunk)
                ne_list.append(full_ne)

    return ' '.join(ne_list)
```

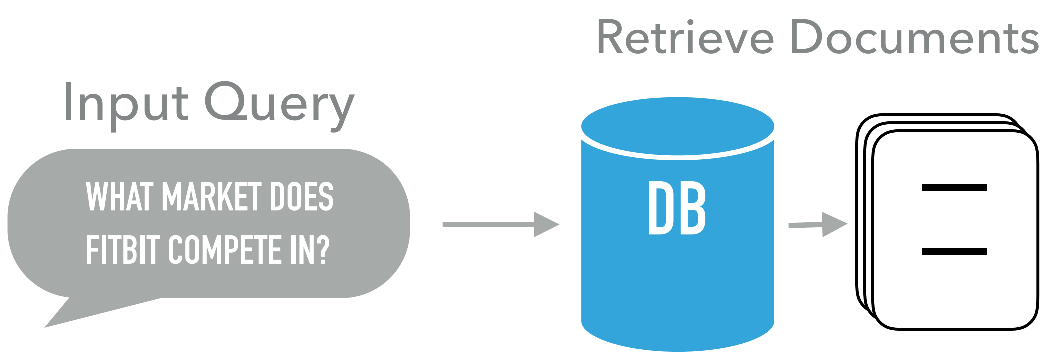
STEP 1 - NER IN NLTK



Ignoring some types like MONEY & DATE

```
QA_NE_TYPES = frozenset(['ORGANIZATION', 'LOCATION', 'FACILITY', 'GPE'])


def ner_extract(text, ne_types=QA_NE_TYPES):
    """Remove non named entities from a string

    :param text: str to remove non named entities from
    :param ne_types: list/set of named entities to keep
    :return: text with non named entities removed
    """
    chunks = nltk.ne_chunk(nltk.pos_tag(nltk.word_tokenize(text)))
    ne_list = []
    for chunk in chunks:
        if hasattr(chunk, 'label'):
            if chunk.label() in ne_types:
                full_ne = ' '.join(c[0] for c in chunk)
                ne_list.append(full_ne)

    return ' '.join(ne_list)
```

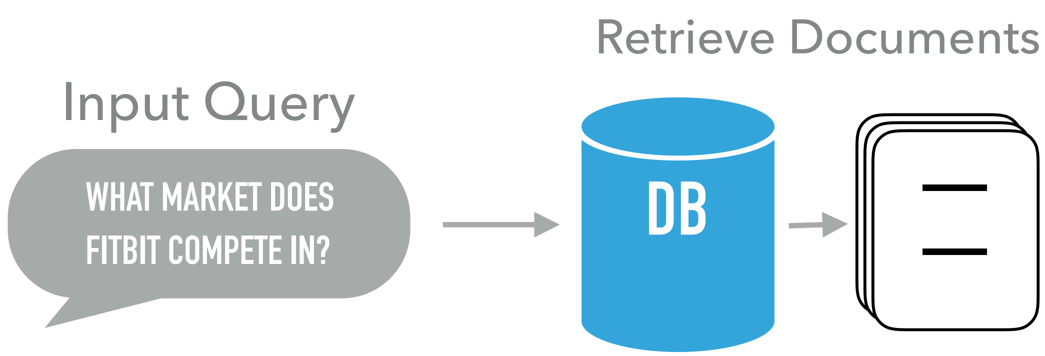
STEP 1 – NER IN NLTK




```
>>> ner_extract('What market does FitBit compete in?')
'FitBit'
```

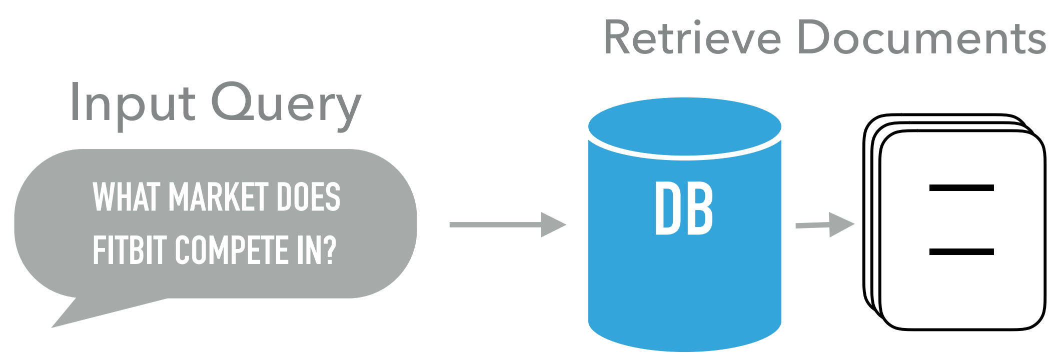
Essentially treats all non Named Entities as stop words.

STEP 1 – EXTENDING WHOOSH WITH NER




```
class NERTokenizer(RegexTokenizer):
    """Named Entity centric version of RegexTokenizer"""
    def __init__(self, ne_types=QA_NE_TYPES, expression=default_pattern, gaps=False):
        self.ne_types = ne_types
        super().__init__(expression=expression, gaps=gaps)


    def __call__(self, text, **kwargs):
        text = ner_extract(text=text, ne_types=self.ne_types)
        return super().__call__(text, **kwargs)
```

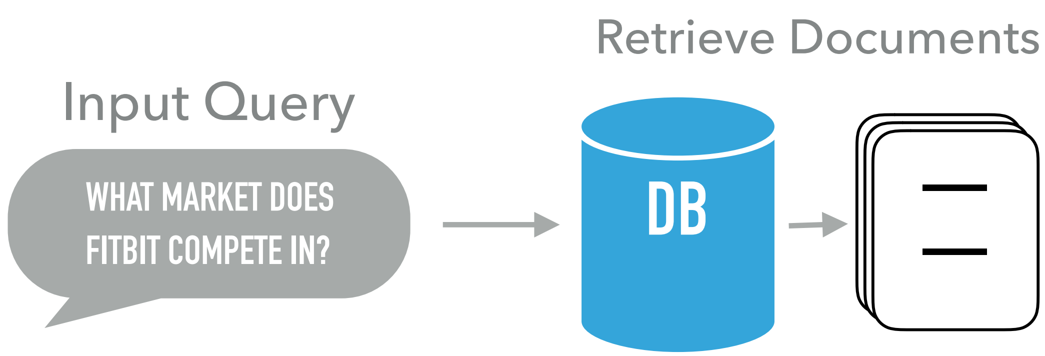
STEP 1 – EXTENDING WHOOSH WITH NER




```python
class NERTokenizer(RegexTokenizer):
    """Named Entity centric version of RegexTokenizer"""
    def __init__(self, ne_types=QA_NE_TYPES, expression=default_pattern, gaps=False):
        self.ne_types = ne_types
        super().__init__(expression=expression, gaps=gaps)


    def __call__(self, text, **kwargs):
        text = ner_extract(text=text, ne_types=self.ne_types)
        return super().__call__(text, **kwargs)
```

Use inheritance + `super()` to build NER into the `RegexTokenizer`'s process with minimal code.

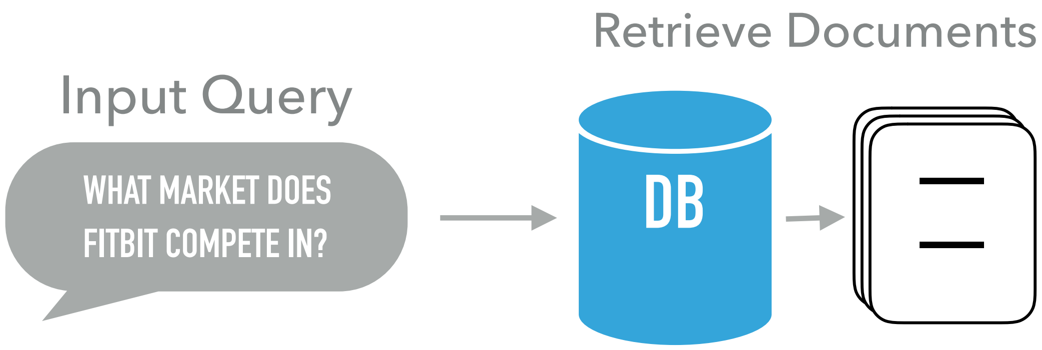
STEP 1 – EXTENDING WHOOSH WITH NER




```
class NERTokenizer(RegexTokenizer):
    """Named Entity centric version of RegexTokenizer"""
    def __init__(self, ne_types=QA_NE_TYPES, expression=default_pattern, gaps=False):
        self.ne_types = ne_types
        super().__init__(expression=expression, gaps=gaps)


    def __call__(self, text, **kwargs):
        text = ner_extract(text=text, ne_types=self.ne_types)
        return super().__call__(text, **kwargs)
```

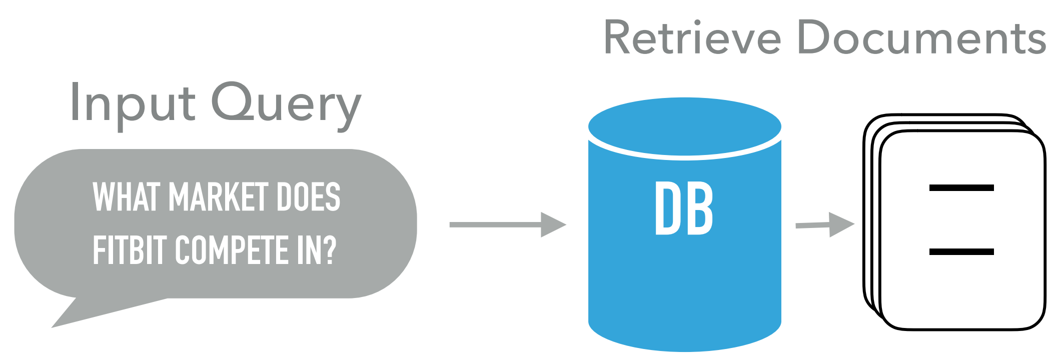
NER relies on the structure of the pre-tokenized text, so it needs to go before tokenization.

STEP 1 – TESTING THE NER TOKENIZER



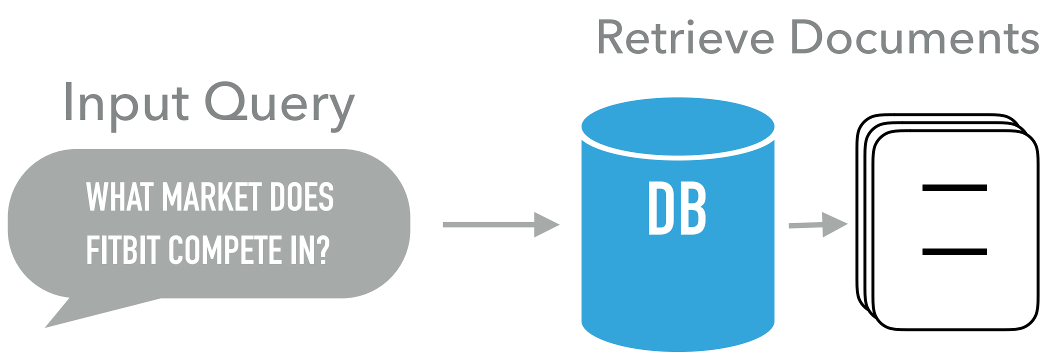

```
>>> [token.text for token in RegexTokenizer()("What market does FitBit compete in?")]
['What', 'market', 'does', 'FitBit', 'compete', 'in']
>>> [token.text for token in NERTokenizer()("What market does FitBit compete in?")]
['FitBit']
```

STEP 1 – TESTING THE NER TOKENIZER



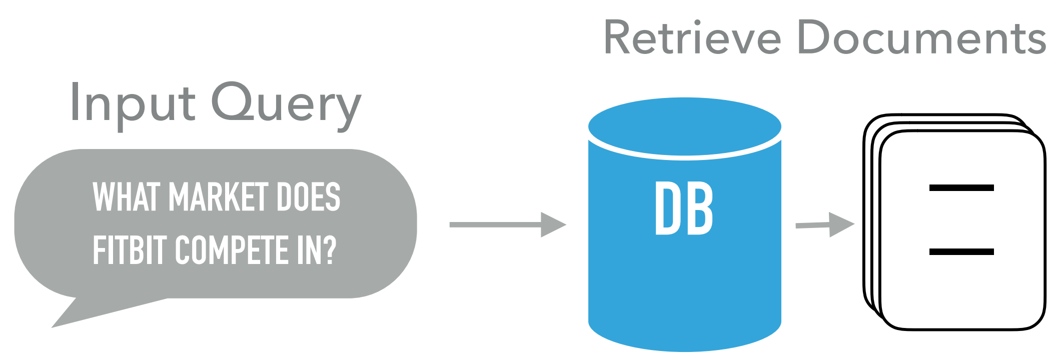

```
>>> [token.text for token in RegexTokenizer()("What market does FitBit compete in?")]
['What', 'market', 'does', 'FitBit', 'compete', 'in']
>>> [token.text for token in NERTokenizer()("What market does FitBit compete in?")]
['FitBit']
```

STEP 1 – Q&A ANALYZER ONE STOP SHOP




```python
# Conforming to camelCase convention used for analyzer functions in whoosh
def NERAnalyzer(ne_types=QA_NE_TYPES, expression=default_pattern, stoplist=QA_STOPWORDS,
                minsize=2, maxsize=None, gaps=False):
    """Named Entity centric version of StandardAnalyzer"""
    chain = NERTokenizer(ne_types, expression, gaps)
    chain |= LowercaseFilter()
    chain |= StopFilter(stoplist=stoplist, minsize=minsize, maxsize=maxsize)

    return chain


def QAAnalyzer(ner_tokenize=False, ne_types=QA_NE_TYPES, expression=default_pattern,
               stoplist=QA_STOPWORDS, minsize=2, maxsize=None, gaps=False):
    """Custom analyzer for Nasdaq Q&A task”""
    if not ner_tokenize:
        return StemmingAnalyzer(expression, stoplist, minsize, maxsize, gaps)
    else:
        return NERAnalyzer(ne_types, expression, stoplist, minsize, maxsize, gaps)
```

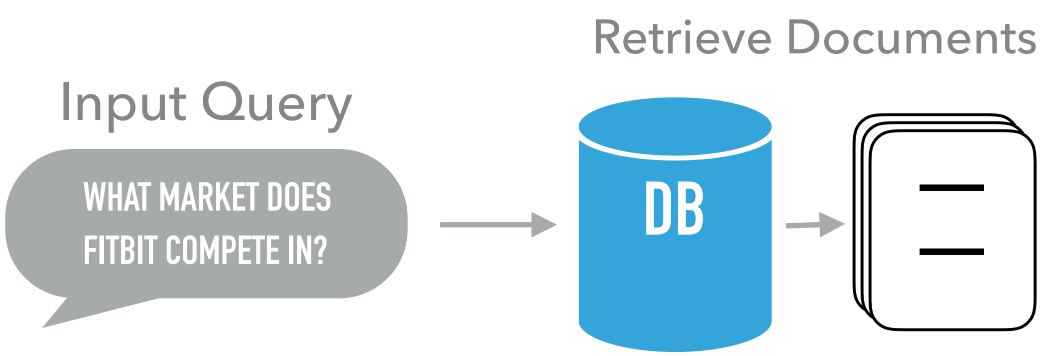
STEP 1 – Q&A ANALYZER ONE STOP SHOP



Analyzers and Filters can be chained with the | operator

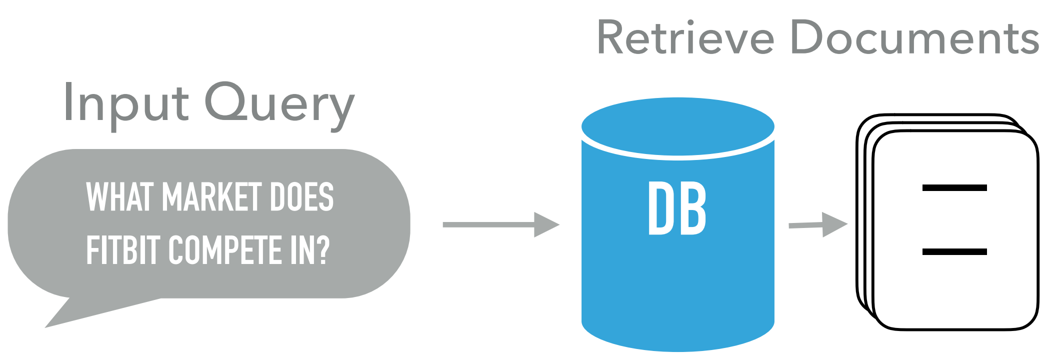
```python
# Conforming to camelCase convention used for analyzer functions in whoosh
def NERAnalyzer(ne_types=QA_NE_TYPES, expression=default_pattern, stoplist=QA_STOPWORDS,
                minsize=2, maxsize=None, gaps=False):
    """Named Entity centric version of StandardAnalyzer"""
    chain = NERTokenizer(ne_types, expression, gaps)
    chain |= LowercaseFilter()
    chain |= StopFilter(stoplist=stoplist, minsize=minsize, maxsize=maxsize)

    return chain


def QAAnalyzer(ner_tokenize=False, ne_types=QA_NE_TYPES, expression=default_pattern,
               stoplist=QA_STOPWORDS, minsize=2, maxsize=None, gaps=False):
    """Custom analyzer for Nasdaq Q&A task"""
    if not ner_tokenize:
        return StemmingAnalyzer(expression, stoplist, minsize, maxsize, gaps)
    else:
        return NERAnalyzer(ne_types, expression, stoplist, minsize, maxsize, gaps)
```

STEP 1 – SCHEMA REDESIGN




```
schema = Schema(
        url=ID(stored=True, unique=True),
        published_datetime=ID(stored=True),
        title=TEXT(stored=True, analyzer=QAAnalyzer()),
        article=TEXT(stored=True, analyzer=QAAnalyzer()),
        title_named_entities=TEXT(analyzer=QAAnalyzer(ner_tokenize=True)),
        article_named_entities=TEXT(analyzer=QAAnalyzer(ner_tokenize=True)),
    )
```

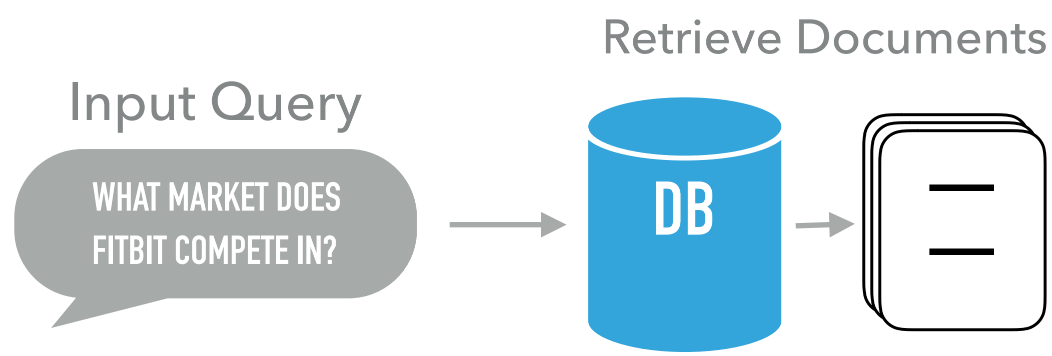
STEP 1 – SCHEMA REDESIGN




```
schema = Schema(
        url=ID(stored=True, unique=True),
        published_datetime=ID(stored=True),
        title=TEXT(stored=True, analyzer=QAAnalyzer()),
        article=TEXT(stored=True, analyzer=QAAnalyzer()),
        title_named_entities=TEXT(analyzer=QAAnalyzer(ner_tokenize=True)),
        article_named_entities=TEXT(analyzer=QAAnalyzer(ner_tokenize=True)),
    )
```

NER is a speed bottle neck in the indexing process. Rebuilding index with NER takes significantly longer

STEP 1 – SEARCHING NEW SCHEMA



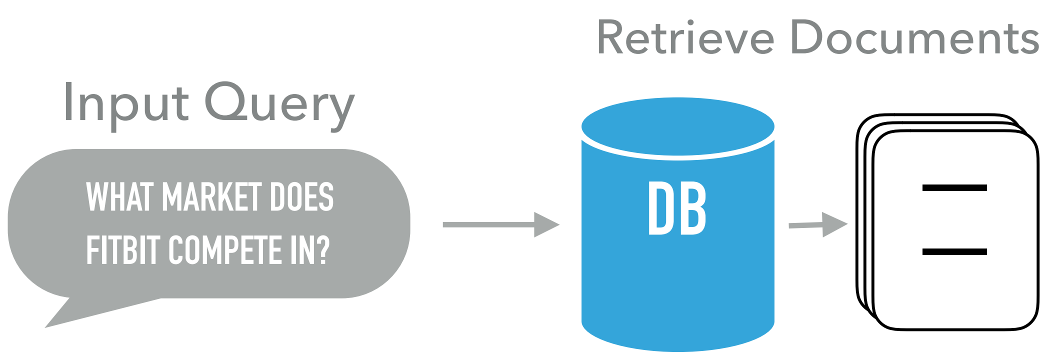

```
>>> import whoosh.index
... from whoosh.qparser import MultifieldParser, OrGroup, WildcardPlugin
...
... whoosh_idx = whoosh.index.open_dir('whoosh_idx', indexname='nasdaq')
... query_parser = MultifieldParser(['title', 'article',
...                                  ‘title_named_entities', ‘article_named_entities'],
                                    schema=whoosh_idx.schema,
                                    group=OrGroup)
... query_parser.remove_plugin_class(WildcardPlugin)
...
... parsed_query = query_parser.parse('What market does FitBit compete in?’)
...
... with whoosh_idx.searcher() as searcher:
...     search_results = searcher.search(parsed_query, limit=1)
...     [print(sr['title']) for sr in search_results]

Apple Watch Sleep Tracking Could Come as Soon as Next Week – Nasdaq.com
```

STEP 1 - SEARCHING NEW SCHEMA

```
>>> import whoos
... from whoosh.
...
... whoosh_idx =
... query_parser
...                d_entities'],


... query_parser
...
... parsed_query                ')
...
... with whoosh_
...     search_r
...     [print(s

Apple Watch Slee
```

*Image from Strange Planet by Nathan W. Pyle

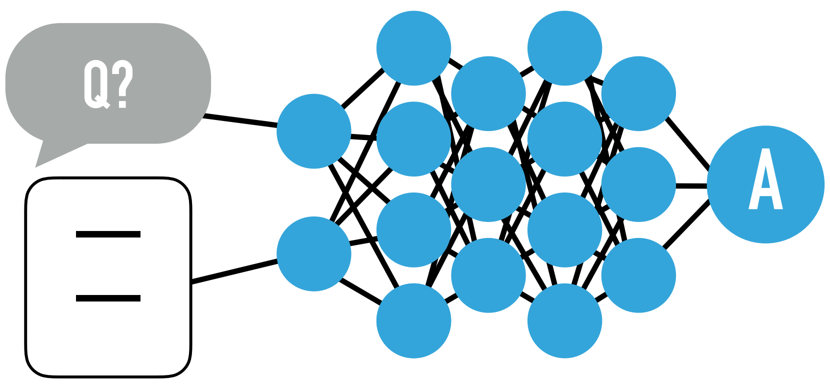
STEP 1 – SEARCHING NEW SCHEMA



- One test case doesn’t prove much
- Adhoc analysis shows the original answer that seems right…
    - …mentions FitBit less than our new answer. So our attempt at boosting Named Entities in search results was successful, but possibly misguided.
    - …mentions “what” a lot, which would boost its rank. So we might have arrived at the right answer the wrong way.

STEP 2 - SQuAD

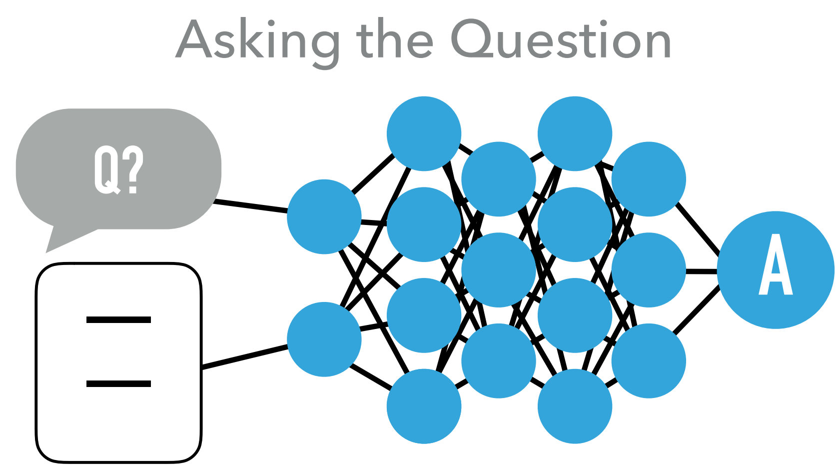
Asking the Question



> ***S**tanford **Qu**estion **A**nswering **D**ataset (SQuAD) is a reading comprehension dataset, consisting of questions posed by crowdworkers on a set of Wikipedia articles, where the answer to every question is a segment of text, or span, from the corresponding reading passage, or the question might be unanswerable.**

*SQuAD Explorer

STEP 2 – SQuAD LEADERBOARD

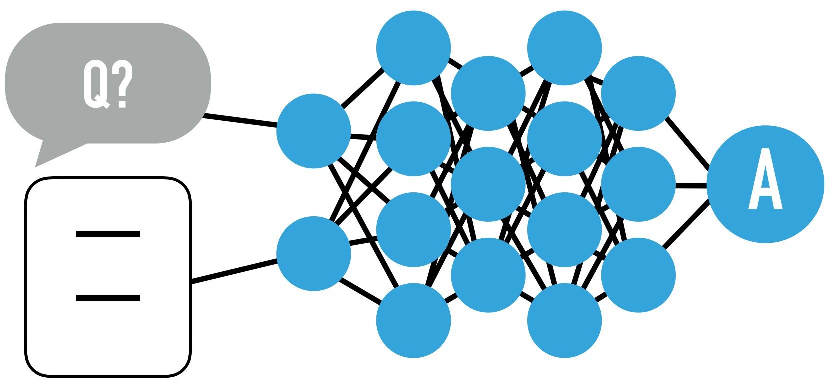


Rank	Model	EM	F1
	Human Performance *Stanford University* (Rajpurkar & Jia et al. '18)	86.831	89.452
1 Sep 18, 2019	ALBERT (ensemble model) *Google Language ALBERT Team*	**89.731**	**92.215**
2 Jul 22, 2019	XLNet + DAAF + Verifier (ensemble) *PINGAN Omni-Sinitic*	88.592	90.859
2 Sep 16, 2019	ALBERT (single model) *Google Language ALBERT Team*	88.107	90.902

*SQuAD Explorer

STEP 2 – SQuAD LEADERBOARD

Asking the Question



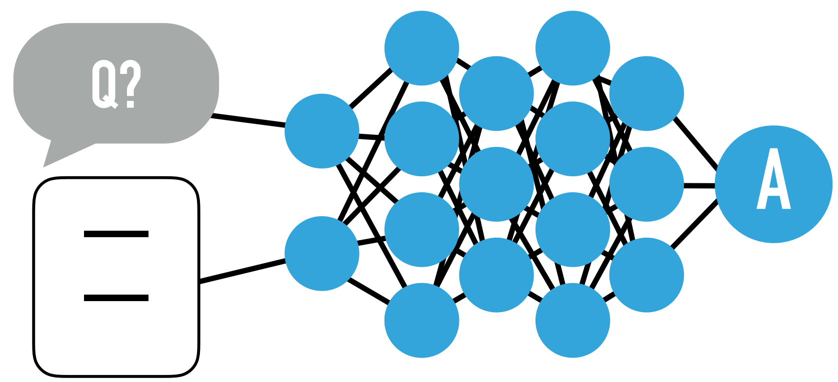
Problem still very actively being worked on

Rank	Model	EM	F1
	Human Performance *Stanford University* (Rajpurkar & Jia et al. '18)	86.831	89.452
1 Sep 18, 2019	ALBERT (ensemble model) *Google Language ALBERT Team*	**89.731**	**92.215**
2 Jul 22, 2019	XLNet + DAAF + Verifier (ensemble) *PINGAN Omni-Sinitic*	88.592	90.859
2 Sep 16, 2019	ALBERT (single model) *Google Language ALBERT Team*	88.107	90.902

*SQuAD Explorer

STEP 2 – SQuAD LEADERBOARD

Asking the Question



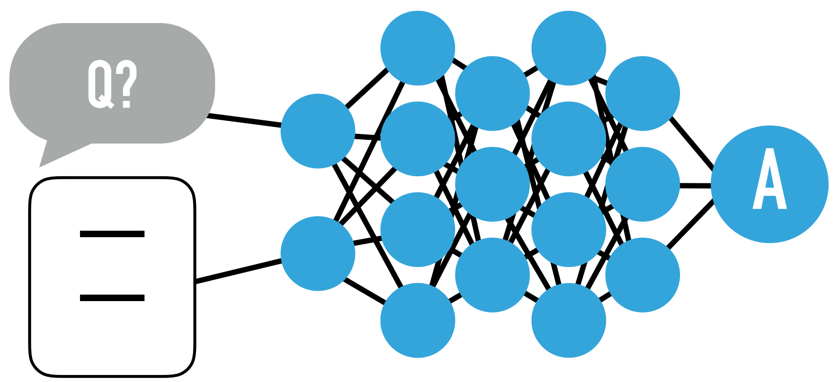
This ALBERT guy seems to do pretty well

Rank	Model	EM	F1
	Human Performance *Stanford University* (Rajpurkar & Jia et al. '18)	86.831	89.452
1 Sep 18, 2019	ALBERT (ensemble model) *Google Language ALBERT Team*	**89.731**	**92.215**
2 Jul 22, 2019	XLNet + DAAF + Verifier (ensemble) *PINGAN Omni-Sinitic*	88.592	90.859
2 Sep 16, 2019	ALBERT (single model) *Google Language ALBERT Team*	88.107	90.902

*SQuAD Explorer

STEP 2 - ALBERT'S OLDER BROTHER, BERT

Asking the Question

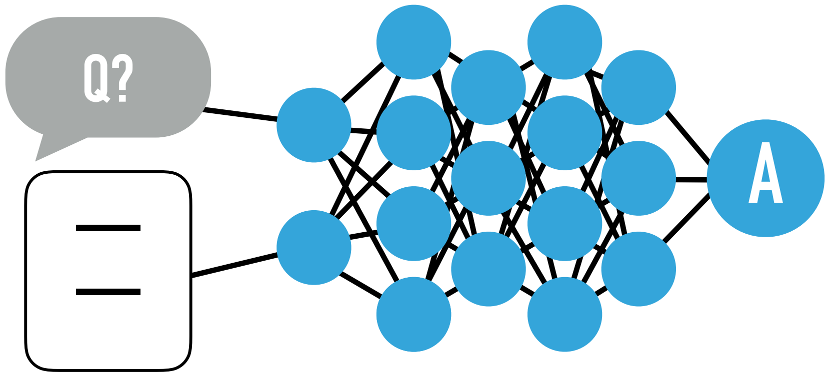


BERT (**B**idirectional **E**ncoder **R**epresentations for **T**ransformers) from Google (2018) is the predecessor to the ALBERT (**A** **L**ite **BERT**).

BERT came on to the scene and immediately started topping the leaderboard of a lot of NLP tasks like SQuAD.

STEP 2 – BERT

Asking the Question



Predecessor language models, like GPT-2, are trained with the intention of predicting the next word given the previous words. 

The direction can be reversed and combined with the other, like in ELMo, but these embeddings are trained uni-directionally.

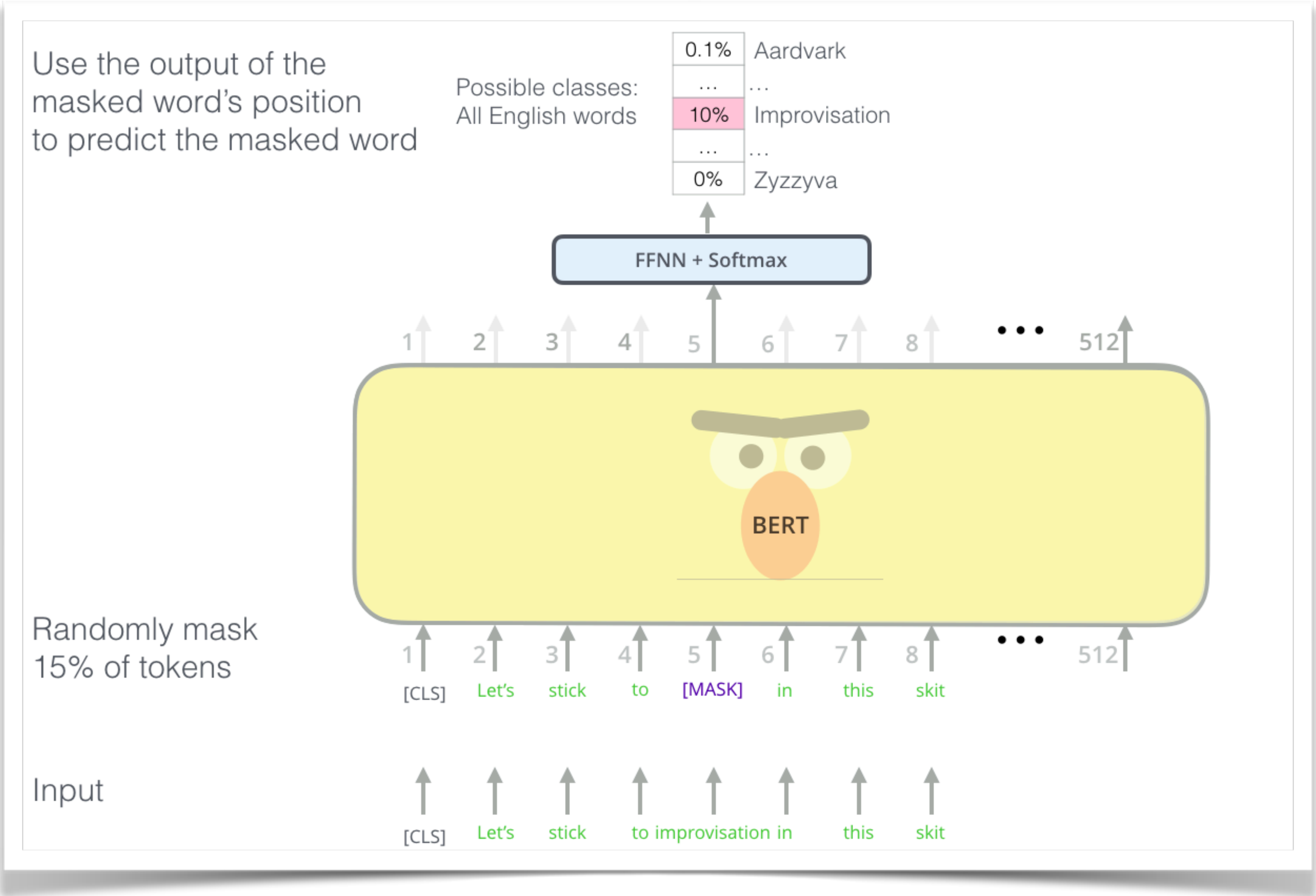


*Image from [Illustrated BERT by Jay Alammar](#)
(highly recommended for a deeper dive)

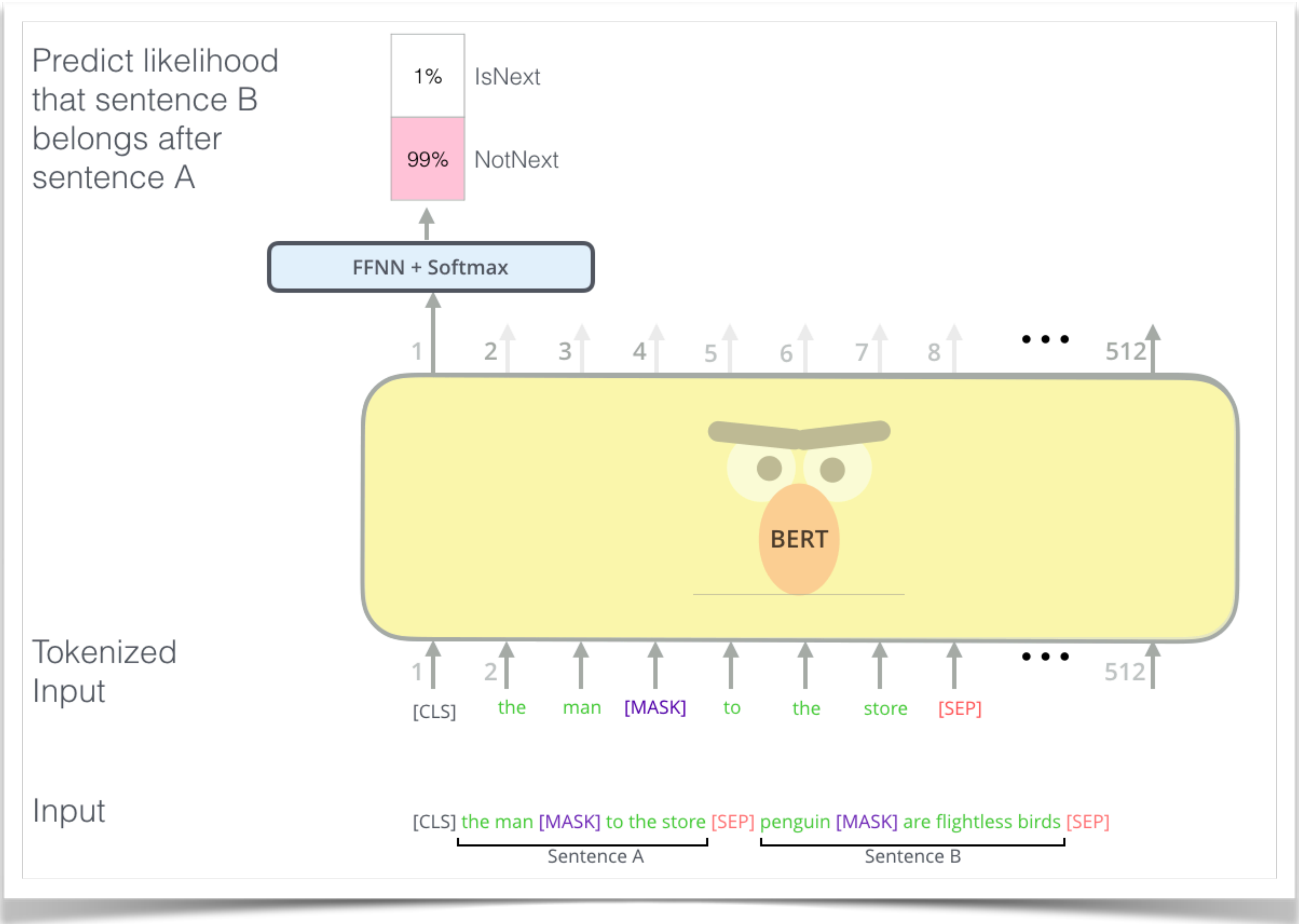
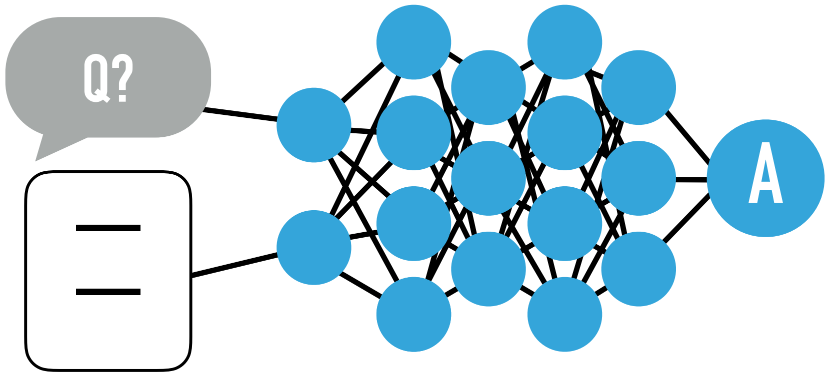


*Image from Illustrated BERT by Jay Alammar
(highly recommended for a deeper dive)

STEP 2 - BERT PRACTICAL APPLICATION KNOWLEDGE

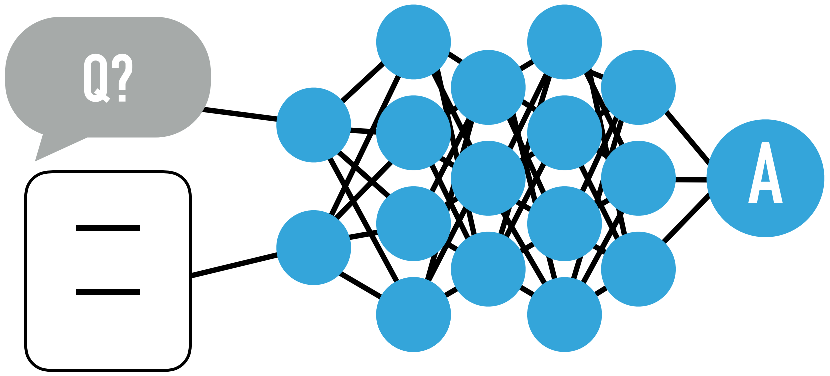
Asking the Question



- It works really well on a lot a of NLP/NLU tasks (including SQuAD)
- Some SQuAD specific BERT models are already pertained, open-sourced, and packaged. Like the one provided by [DeepPavlov]

STEP 2 - DEEPPAVLOV + BERT + SQuAD

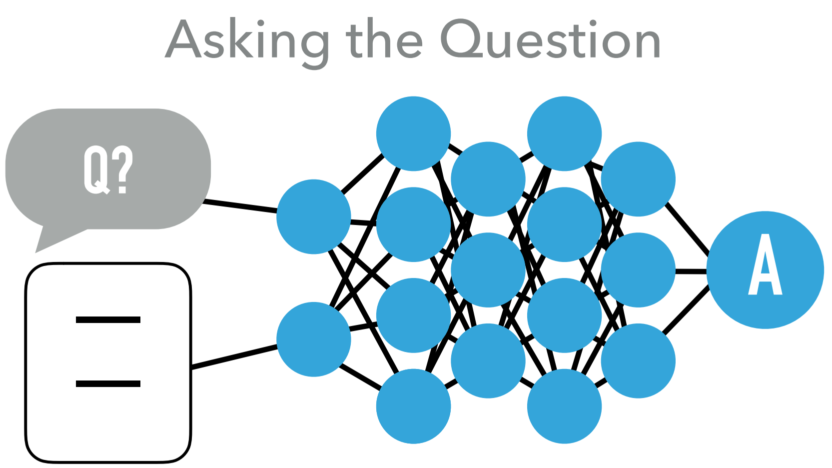
Asking the Question



The pre-trained model we'll be using was trained on SQuAD 1.1. In SQuAD 2.0 the label of "No Answer" was introduced. Models trained with 1.1 will always provide what they think is the best answer (even if the answer is terrible).

2.0 was chosen based on accuracy given experimentation with the DeepPavlov provided models.

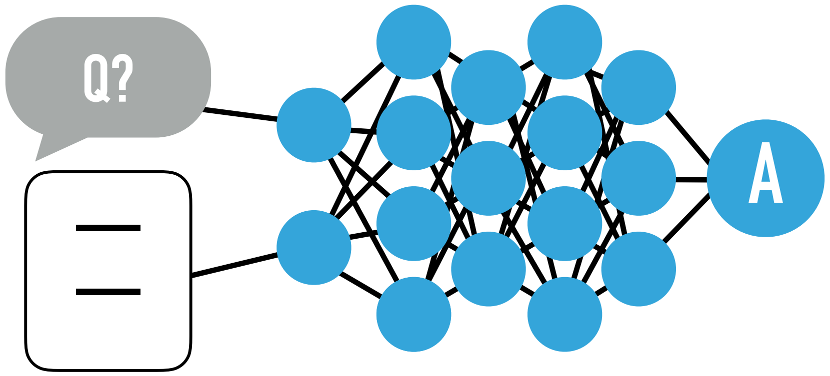
STEP 2 – USING THE MODEL




```
>>> from deeppavlov import build_model, configs
...
... document = ['The rain in Spain falls mainly on the plain.']
... question = ['Where does the rain in Spain fall?']
...
... model = build_model(configs.squad.squad, download=False)
... model(document, question)
[['the plain'], [34], [132021.109375]]
```

STEP 2 – USING THE MODEL

Asking the Question

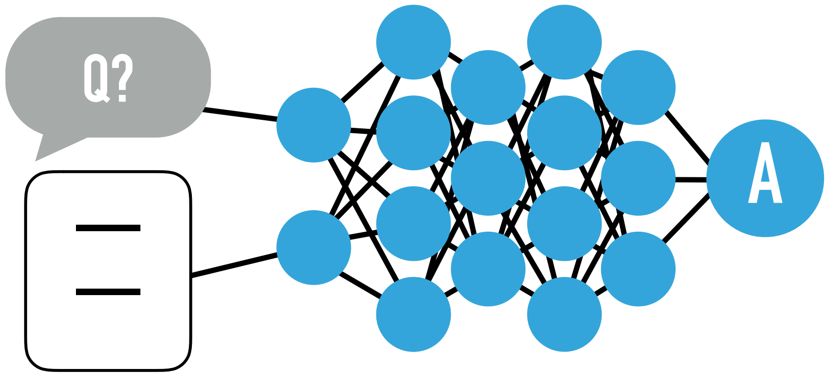



```
>>> from deeppavlov import build_model, configs
...
... document = ['The rain in Spain falls mainly on the plain.']
... question = ['Where does the rain in Spain fall?']
...
... model = build_model(configs.squad.squad, download=False)
... model(document, question)
[['the plain'], [34], [132021.109375]]
```

The answer itself

STEP 2 - USING THE MODEL

Asking the Question

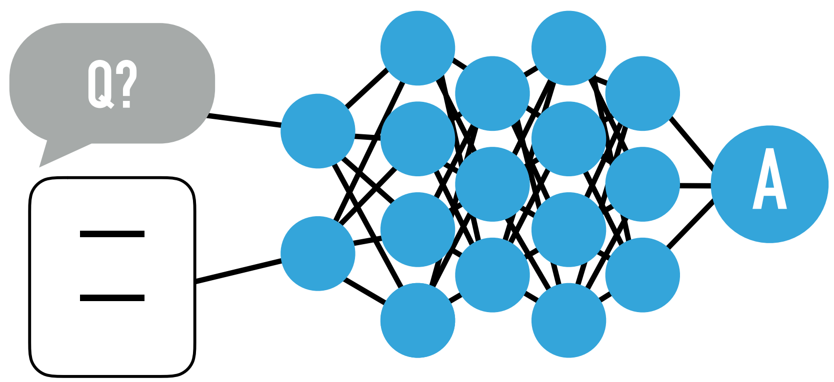



```
>>> from deeppavlov import build_model, configs
...
... document = ['The rain in Spain falls mainly on the plain.']
... question = ['Where does the rain in Spain fall?']
...
... model = build_model(configs.squad.squad, download=False)
... model(document, question)
[['the plain'], [34], [132021.109375]]
```

The answer's first character's index in the document

STEP 2 - USING THE MODEL

Asking the Question

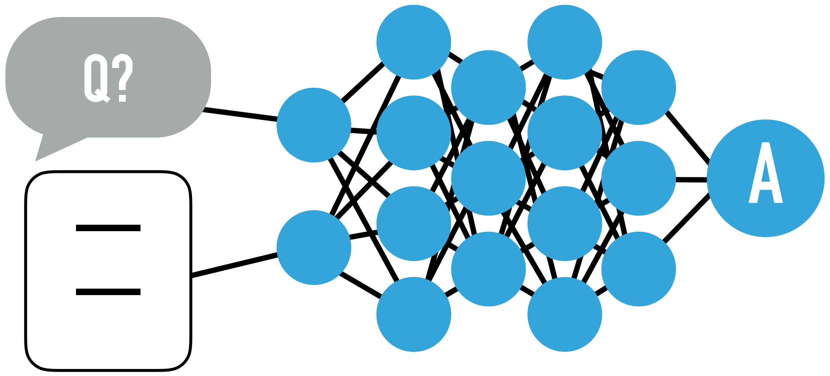



```
>>> from deeppavlov import build_model, configs
...
... document = ['The rain in Spain falls mainly on the plain.']
... question = ['Where does the rain in Spain fall?']
...
... model = build_model(configs.squad.squad, download=False)
... model(document, question)
[['the plain'], [34], [132021.109375]]
```

A measure of answer confidence

STEP 2 - DOCUMENT Q&A

Asking the Question



Combining search and SQuAD

STEP 3 – RANKING ANSWERS

Ranking Answers

#	As
1	-----
2	-----

ranks from search and from SQuAD… why not both?

STEP 3 – RANKING ANSWERS

Ranking Answers

#	As
1	-----
2	-----

String together full process