Winter semester 2022/23

344.175 VL: Natural Language Processing
Information Retrieval – Principles & Recent Approaches

Navid Rekab-saz

Email: navid.rekabsaz@jku.at
Office hours: https://navid-officehours.youcanbook.me

JOHANNES KEPLER UNIVERSITY LINZ

Institute of Computational Perception

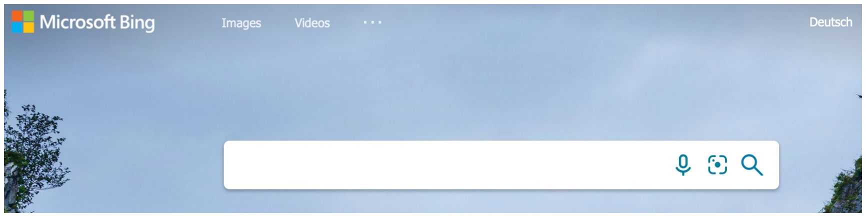
Agenda

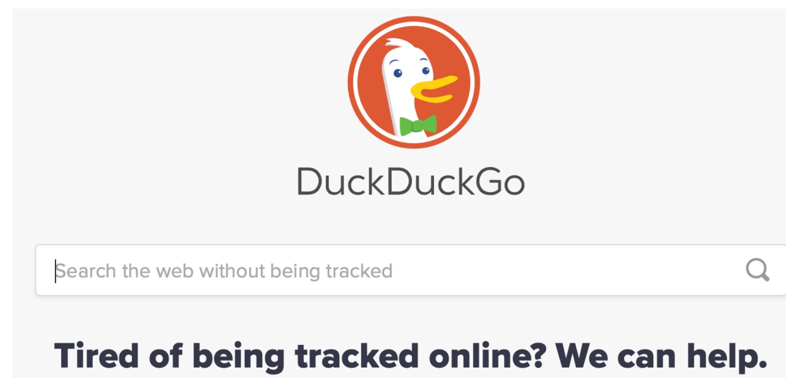
- Principles of Information Retrieval
- Neural IR models
- Learning to Rank

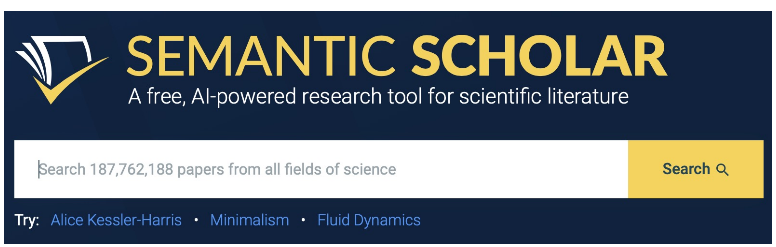
Agenda

- **Principles of Information Retrieval**
- Neural IR models
- Learning to Rank

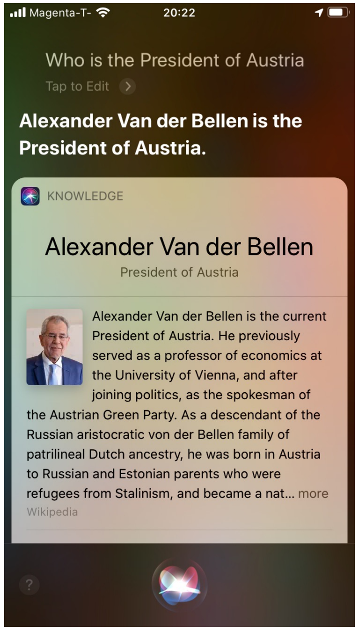
Information Retrieval everywhere!

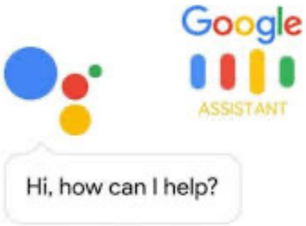






Information Retrieval everywhere!





IBM Watson and Jeopardy

Information Retrieval

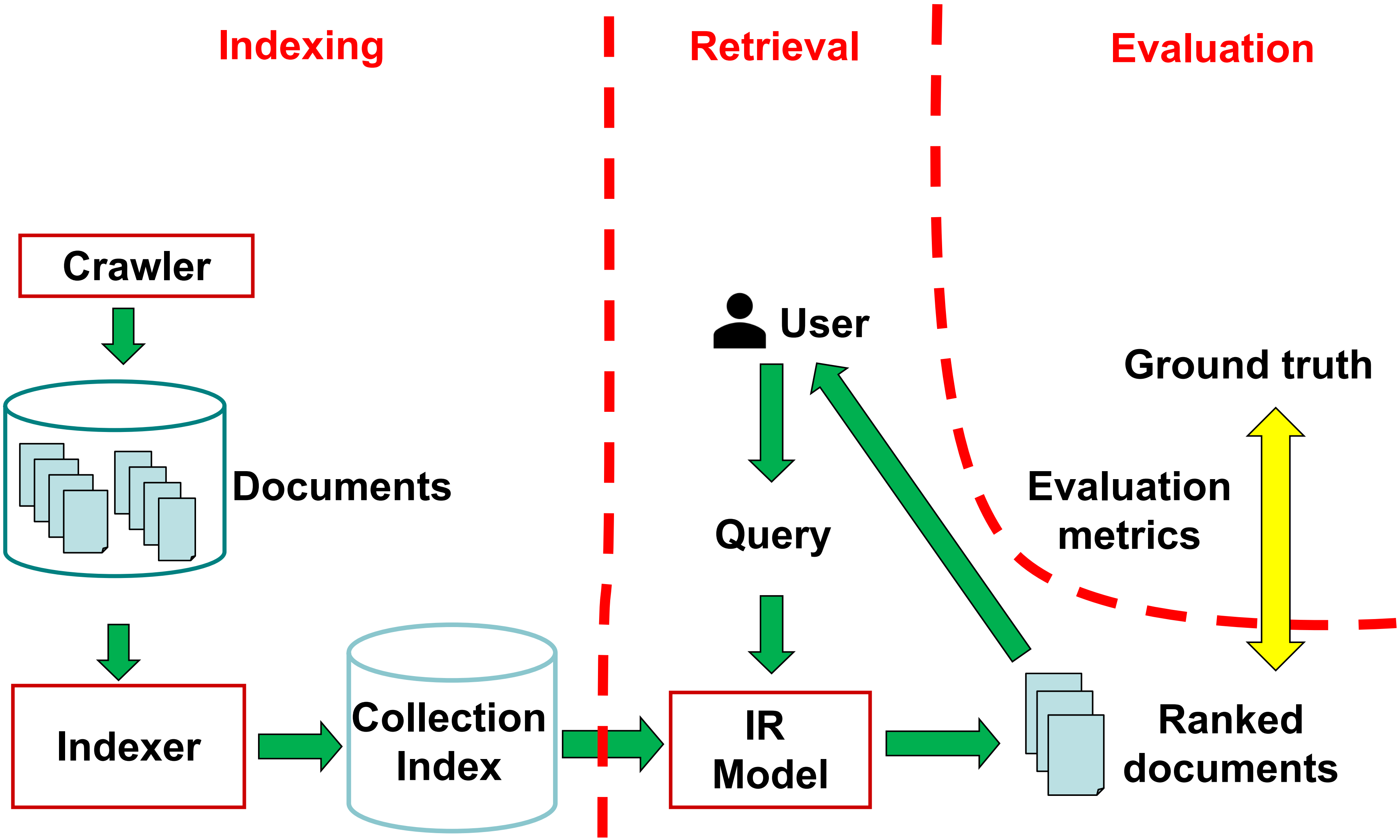
- Information Retrieval (IR) is finding material (usually in the form of documents) of an unstructured nature that satisfies an information need from within large collections

- When talking about IR, we often only think of web search

- The goal of IR is however to retrieve relevant contents to the user's information need

- IR covers a wide set of tasks such as …
  - Ranking, question/answering, information summarization
  - But also … user behavior/experience study, personalization, etc.

Simplified architecture of an IR system



Terminology

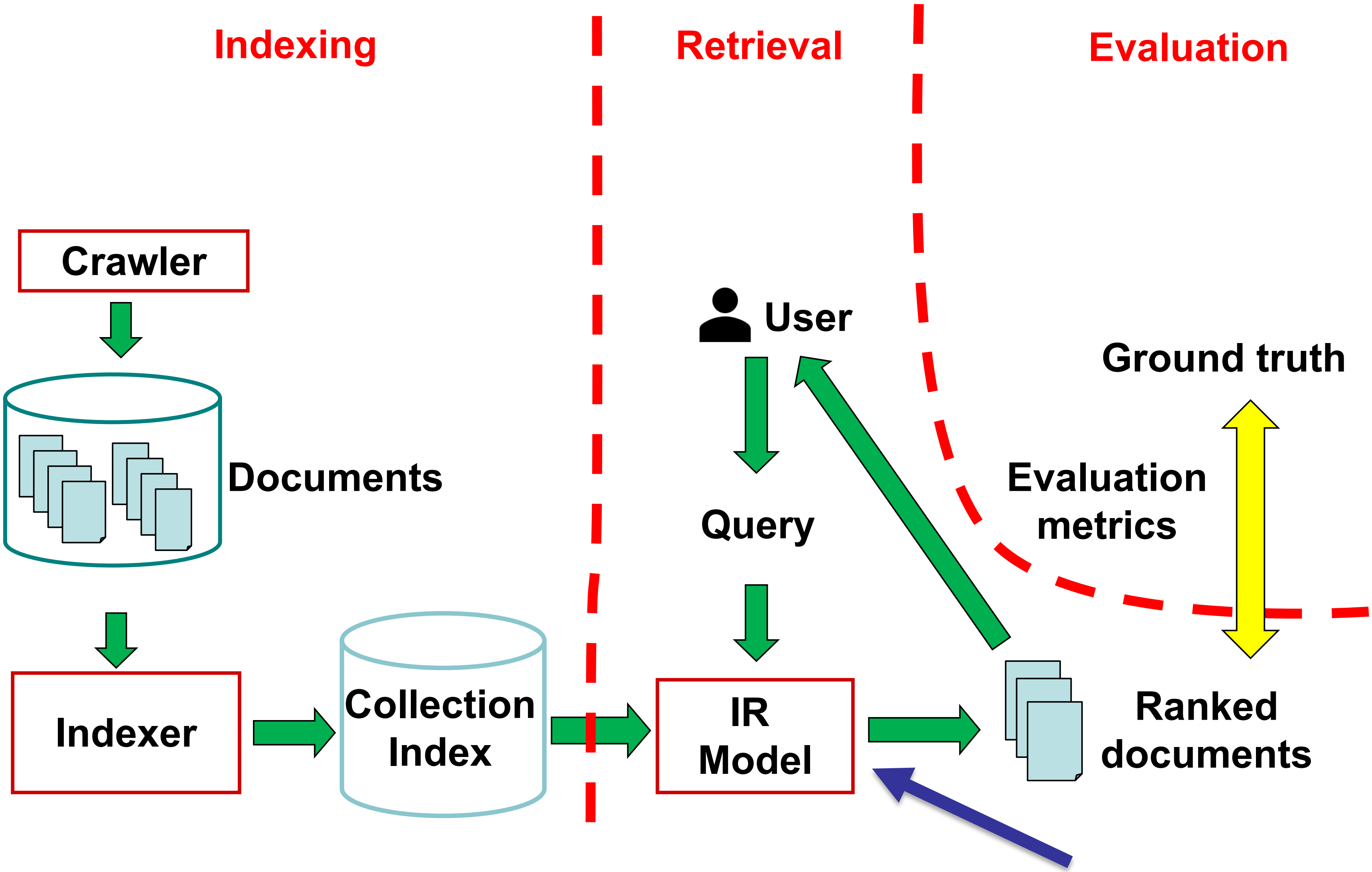
- Information need
  - E.g. *My swimming pool bottom is becoming black and needs to be cleaned*
- Query
  - A designed representation of users’ information need
  - E.g. *pool cleaner*
- Document
  - A unit of data in text, image, video, audio, etc.
- Relevance
  - Whether a document satisfies user’s information need
  - Relevance has multiple aspects: topical, semantic, temporal, spatial, etc.

Ad-hoc IR (all we discuss in this lecture)

- Studying the methods to estimate relevance, solely based on the contents (texts) of queries and documents
  - In ad-hoc IR, *meta-knowledge* such as temporal, spatial, user-related information are normally taken out
  - The focus of ad-hoc IR is on methods to exploit contents

- Ad-hoc IR is a part of the ranking mechanism of search engines, but there are several other aspects…
  - Diversity of information
  - Personalization
  - Information need understanding
  - Search engine log files analysis
  - …

Simplified architecture of an IR system



Relevance scoring & IR models

query: wisdom of mountains
(q)

$d20$

$d1402$

$d5$

$d100$

- Collection contains M documents. Each document d and each query q consists of a set of terms
- An IR model calculates/predicts a relevance score between the query and document:

$$s(q, d)$$

- Documents are ranked according to their predicted relevance scores to the query, from highest to lowest

Exact-matching IR models

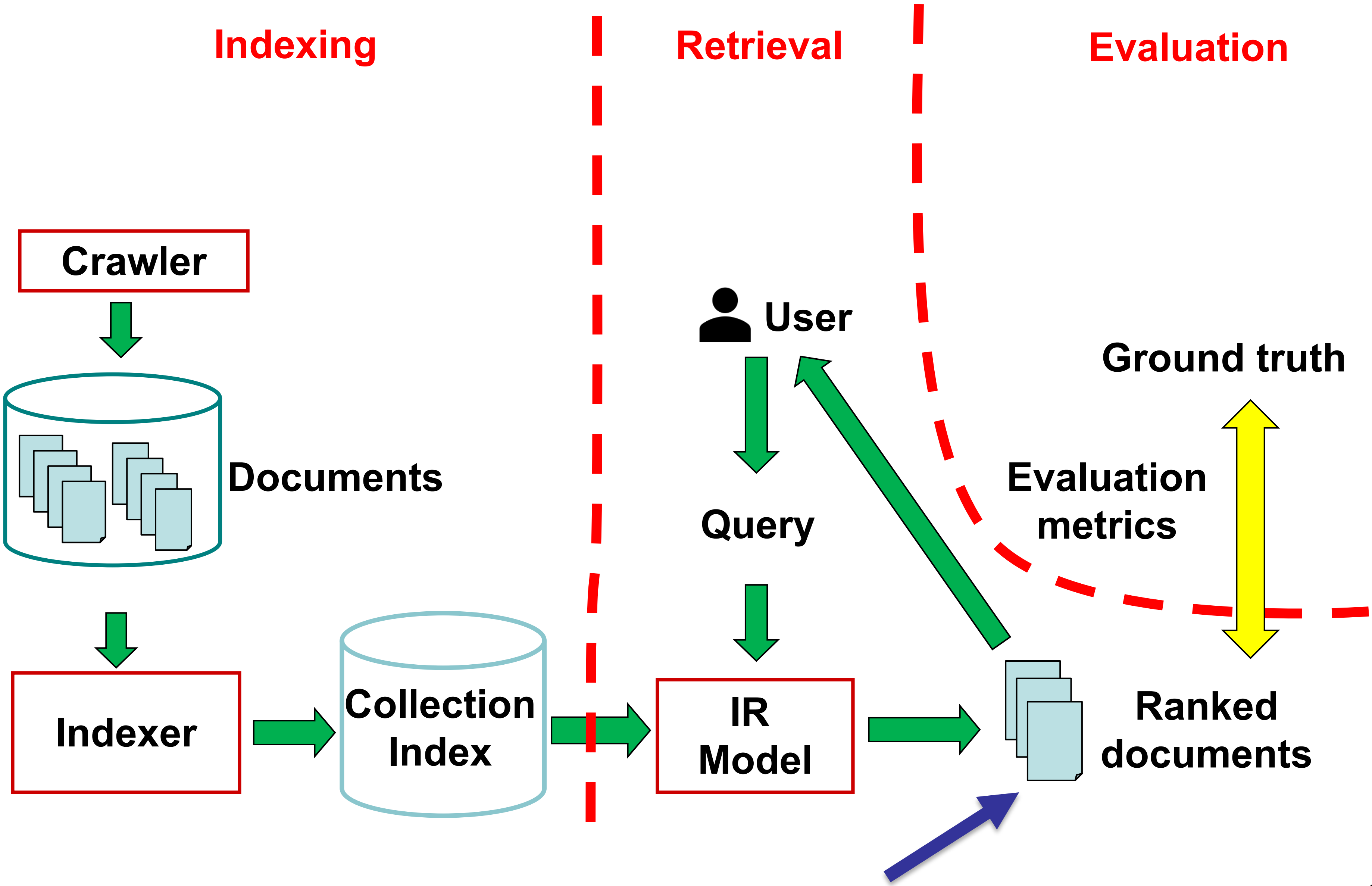
- Exact-matching IR models – in their basic forms – assign importance weights to each query term that appears in a document

$$s(q,d) = \sum_{q_i \in q} \mathrm{term\_weighting}_{q_i,d}$$

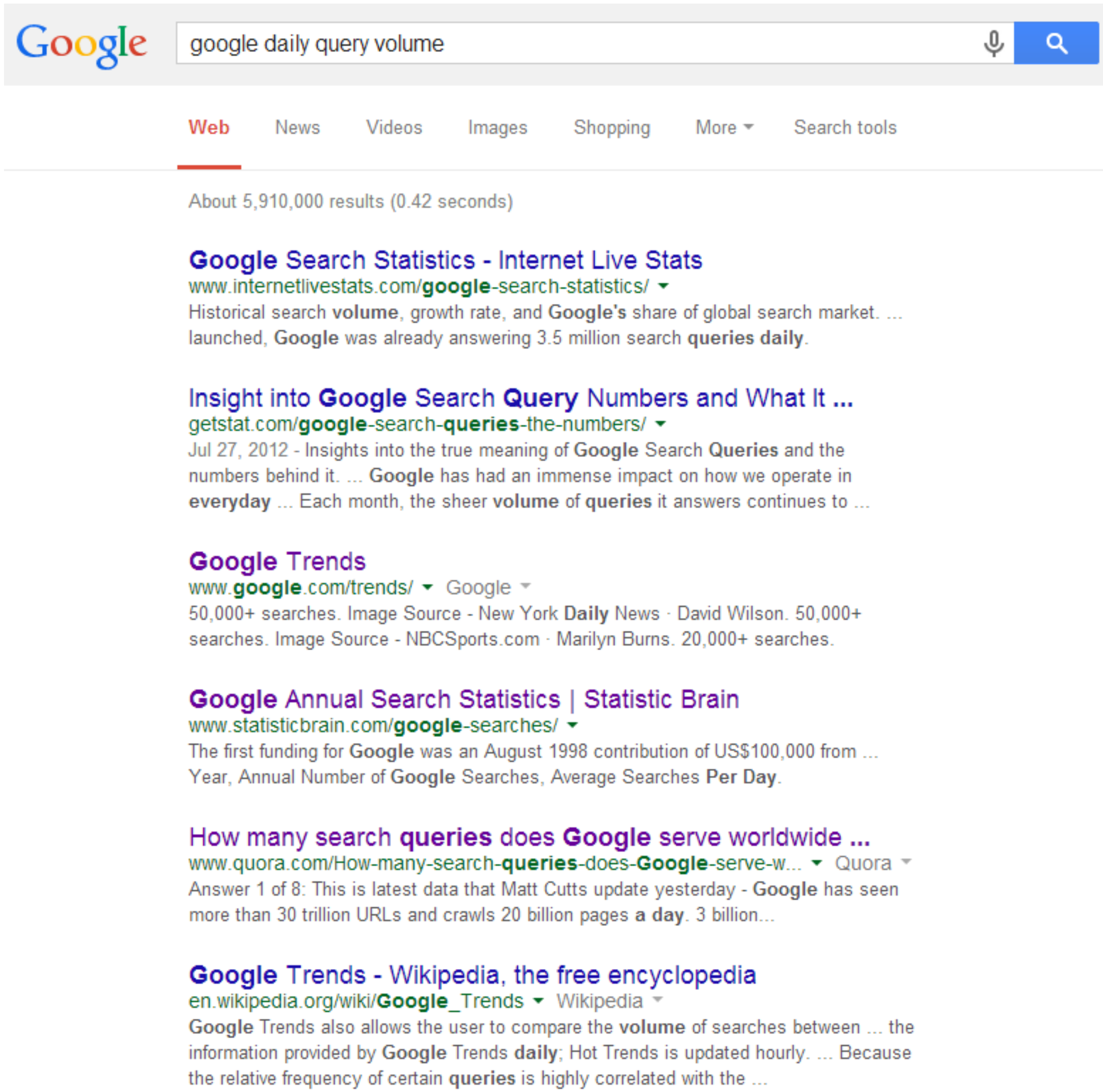
- Possible exact-matching term weighting models:
  - tc
  - tf-idf
  - PL
  - BM25

- Neural IR models (next topic in the lecture)

Simplified architecture of an IR system



Ranking results as we know!

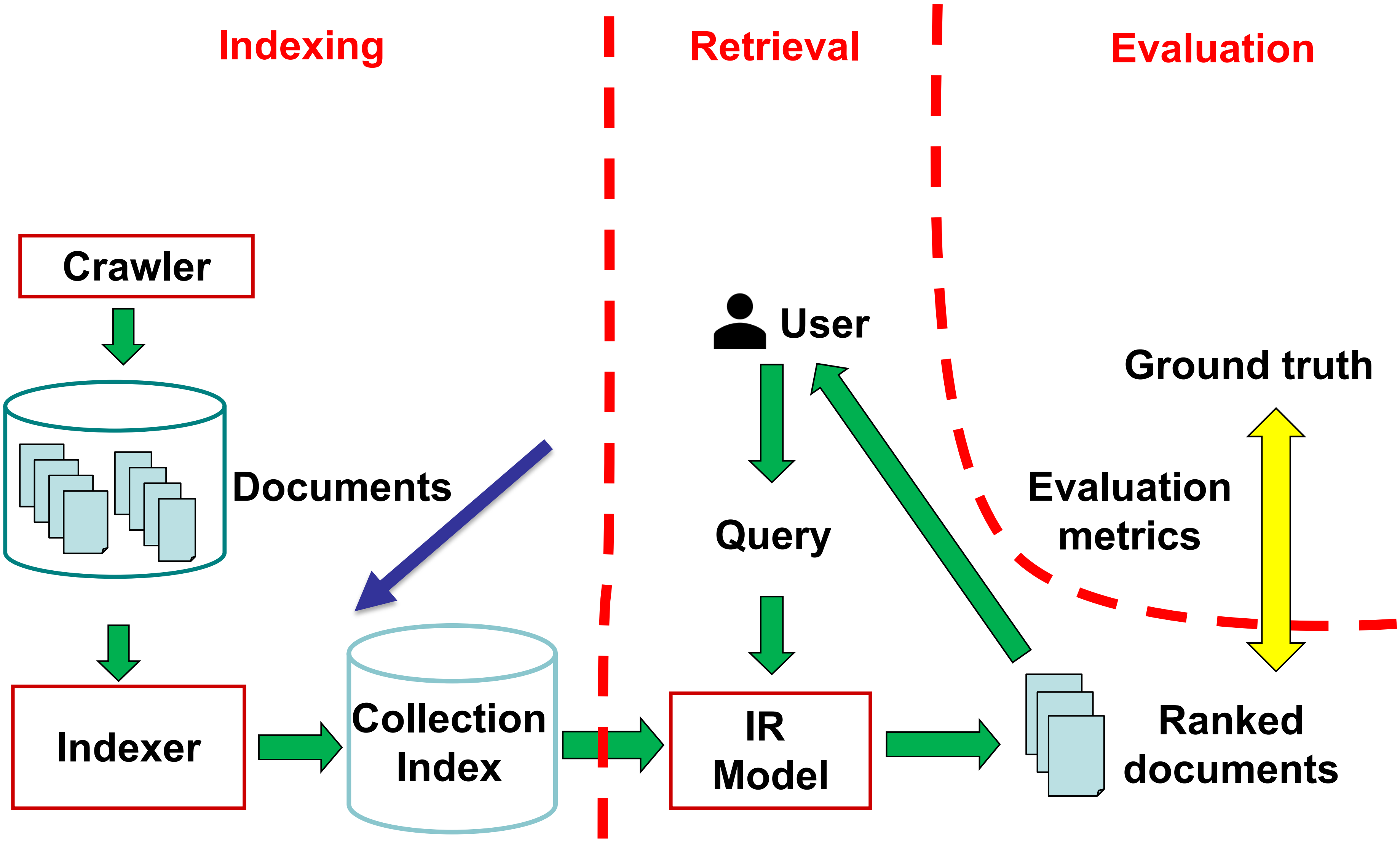


Sample ranking results – format in research!

- TREC run file: standard format to report the ranking results of top-1000 documents for some queries, retrieved by a model

qry_id	(iter)	doc_id	rank	score	run_id
2	Q0	1782337	1	21.656799	cool_model
2	Q0	1001873	2	21.086500	cool_model
			…		
2	Q0	6285819	999	3.43252	cool_model
2	Q0	6285819	1000	1.6435	cool_model
8	Q0	2022782	1	33.352300	cool_model
8	Q0	7496506	2	32.223400	cool_model
8	Q0	2022782	3	30.234030	cool_model
			…		
312	Q0	2022782	1	14.62234	cool_model
312	Q0	7496506	2	14.52234	cool_model
			…		

Simplified architecture of an IR system



Efficient retrieval with pre-computed Collection Index

query: wisdom of mountains
(q)

$d20$

$d1402$

$d5$

$d100$

How can we efficiently calculate relevance scores for documents? → Collection Index

☞ Since the IR models so far are based on <u>exact matching</u>, an we can focus on calculating relevance scores only for the <u>documents that contain query terms</u> → done by Inverted Index

Inverted index

- Inverted index is a data structure for efficient retrieval
  - Inverted index is created once at index time for all documents in the collection, and used for each query during query time
- Inverted index creates a posting list for each unique term in collection
  - A posting list of a term contains the list of the IDs of the documents, in which the term appears

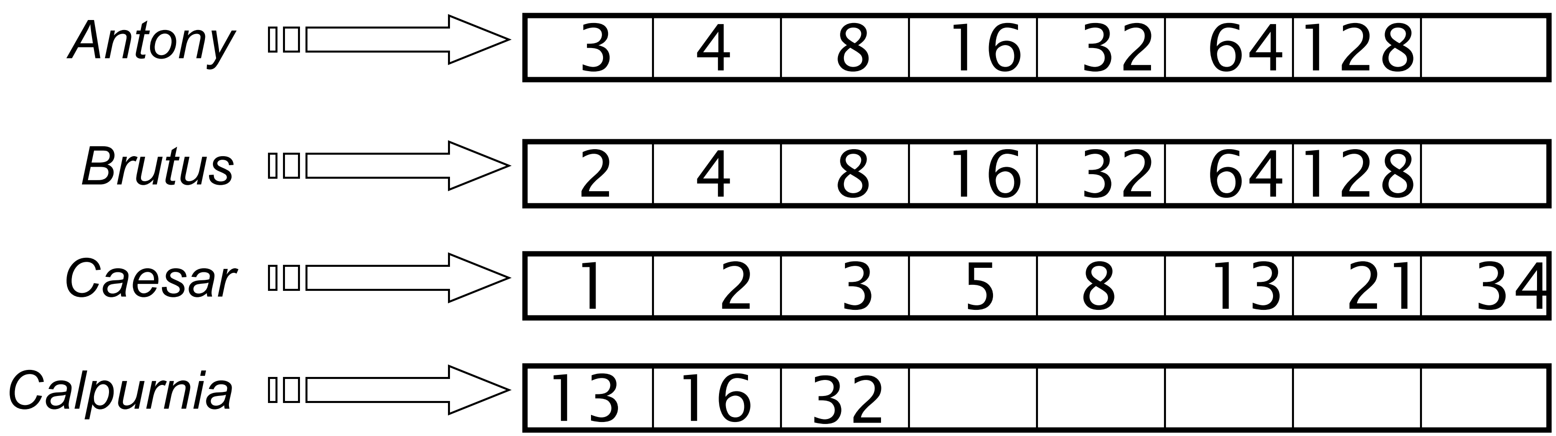
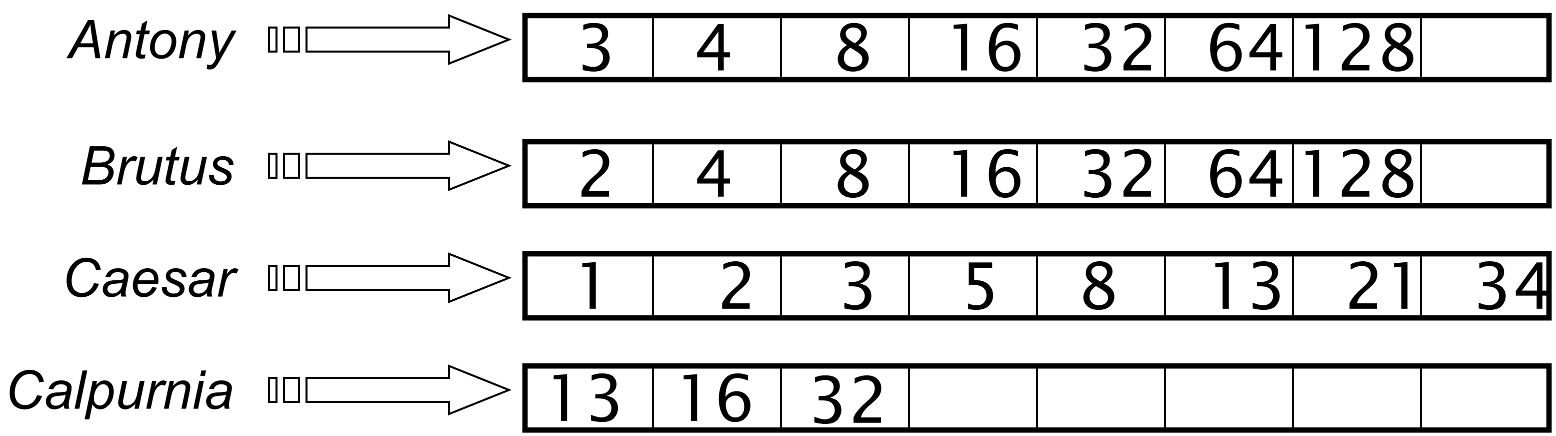


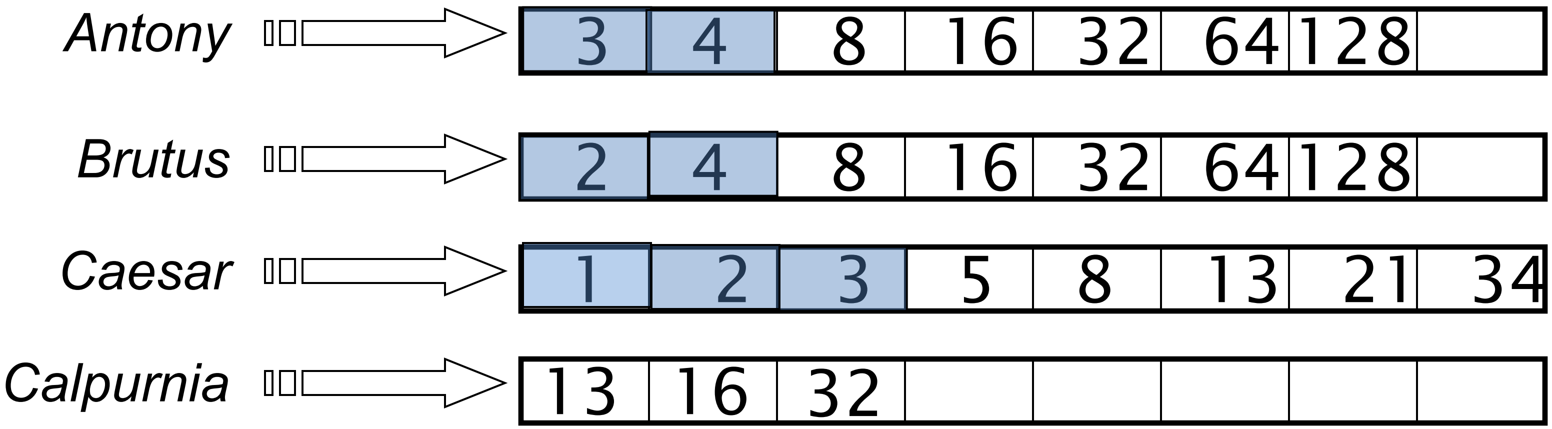
Image source: https://web.stanford.edu/class/cs276/19handouts/lecture2-intro-boolean-1per.pdf

Retrieval process using inverted index

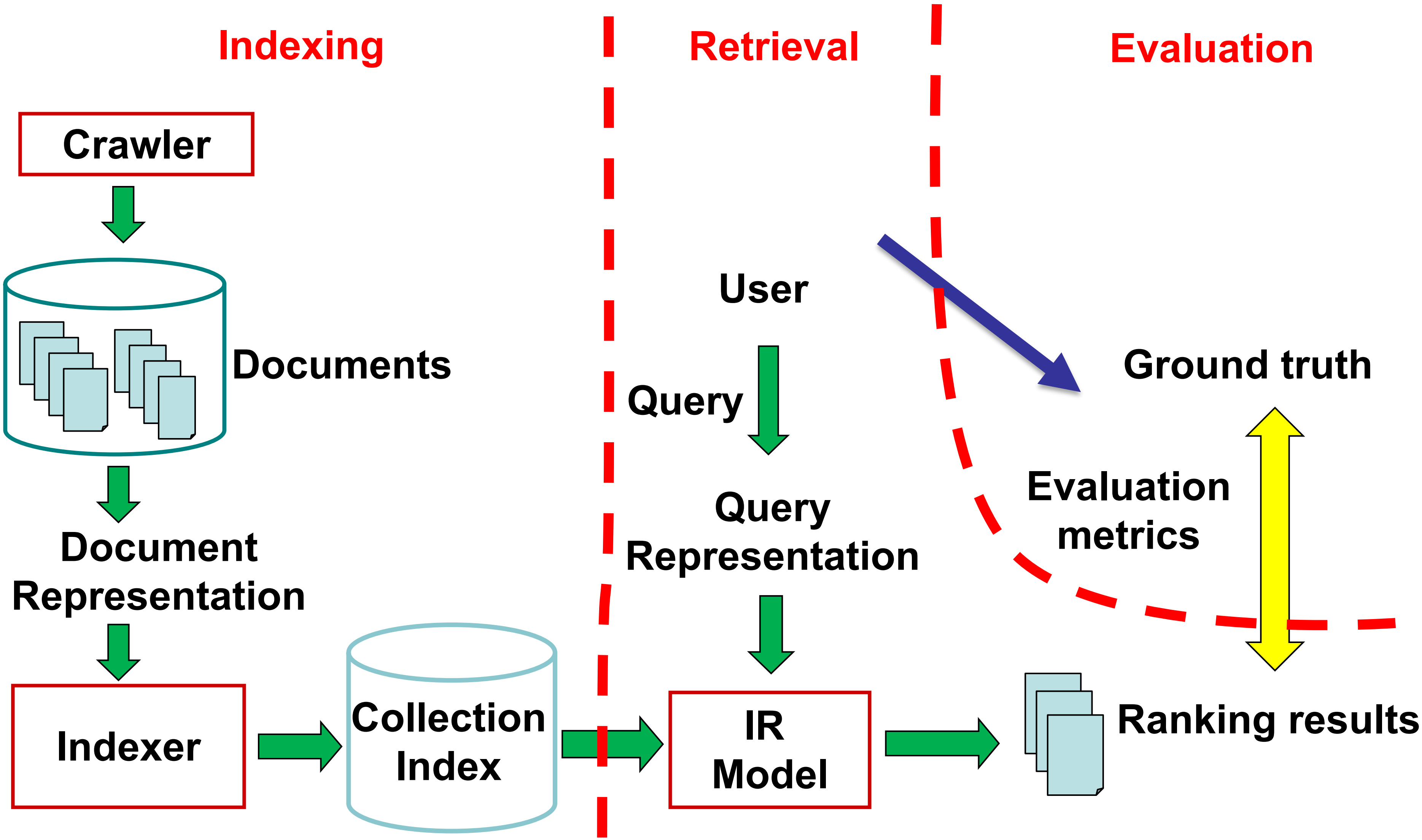
1. Fetch the posting lists of query terms
2. Traverse through posting lists, and calculate the relevance score for each document in the posting lists
3. Retrieve top *n* documents with the highest relevance scores



Search with concurrent traversal



Components of an IR System (simplified)



IR evaluation

- Evaluation of an IR system requires three elements:
  - A benchmark document collection
  - A benchmark suite of queries
  - Relevance judgements for pairs of query–document
    - Judgements specifies whether the document addresses the underlying information need of the query
    - Ideally done by human, but also through user interactions
    - Relevance judgements appear in forms of …
      - Binary: 0 (non-relevant) vs. 1 (relevant), or …
      - Multi-grade: more nuanced relevance levels, e.g. 0 (non-relevant), 1 (fairly relevant), 2 (relevant), 3 (highly relevant)

Evaluation Campaigns

- Text REtrieval Conference (TREC)

https://trec.nist.gov

- Conference and Labs of the Evaluation Forum (CLEF)

http://www.clef-initiative.eu

- MediaEval Benchmarking Initiative for Multimedia Evaluation

http://www.multimediaeval.org

Sample relevance judgement – format in research!

- TREC QRel (QueryRelevance) file: standard format to provide relevance judgements of some queries regarding to some documents

```
qry_id (iter)    doc_id   relevance_grade

 101    0         183294       0
 101    0         123522       2
 101    0         421322       1
 101    0         12312        0
                 …
 102    0         375678       2
 102    0         123121       0
                 …
 135    0         124235       0
 135    0         425591       1
                 …
```

Common IR Evaluation Metrics

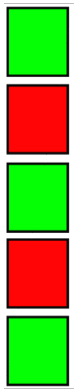
- Binary relevance
  - Precision@*n* (P@*n*)
  - Recall@*n* (P@*n*)
  - Mean Reciprocal Rank (MRR)
  - Mean Average Precision (MAP)

- Multi-grade relevance
  - Normalized Discounted Cumulative Gain (nDCG)

Precision@*n*

- **Precision@*n***: fraction of retrieved docs at top-*n* results that are relevant

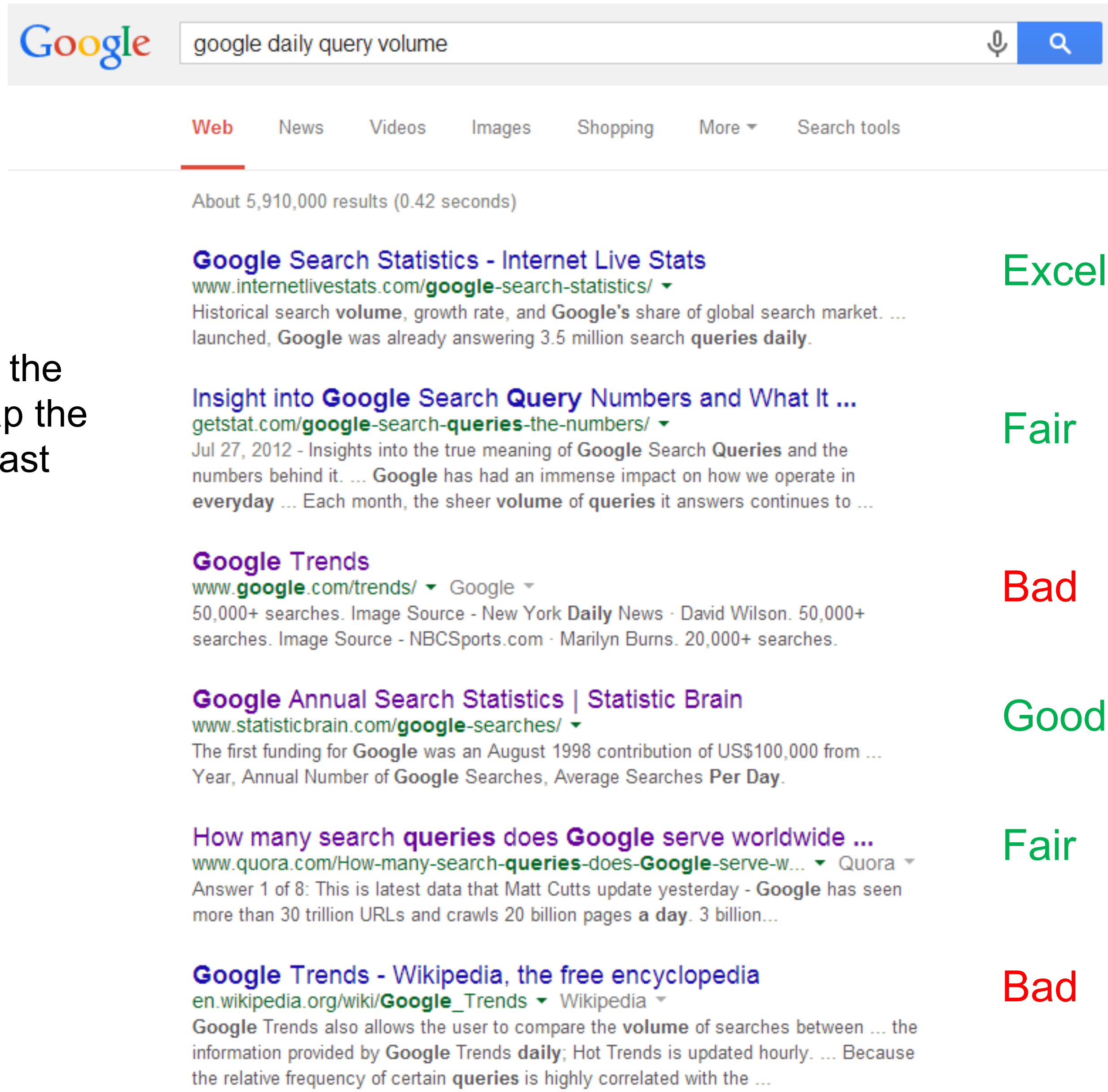
- Example:
  - P@3 = 2/3
  - P@4 = 2/4
  - P@5 = 3/5



- Final evaluation result is the mean of P@*n* across all queries in test set

Rank positions matter!

P@6 remains the same if we swap the first and the last result!



Excellent

Fair

Bad

Good

Fair

Bad

Discounted Cumulative Gain (DCG)

- A popular measure for evaluating web search and other related tasks

- Assumptions:
  - Highly relevant documents are more useful than marginally relevant documents (multi-grade relevance)
  - The lower the ranked position of a relevant document, the less useful it is for the user, since it is less likely to be examined
    - This common behavior of users when interacting with ranked lists is known as *position bias*

Discounted Cumulative Gain (DCG)

- Gain: define gain as graded relevance, provided by relevance judgements

- Discounted Gain: gain is reduced as going down the ranking list. A common discount function: ${}^{1}\!/_{\log_2(\text{rank position})}$
  - With base 2, the discount at rank 4 is 1/2, and at rank 8 it is 1/3

- Discounted Cumulative Gain: the discounted gains are accumulated starting at the top of the ranking to the lower ranks till rank n

Discounted Cumulative Gain (DCG)

- Given the ranking results of a query, DCG at the position n of the ranking list is:

$$\text{DCG@}n = rel_1 + \sum_{i=2}^{n} \frac{rel_i}{\log_2 i}$$

where rel_i is the graded relevance (in relevance judgements) of the document at position i of the ranking results

- Alternative formulation (commonly used):

$$\text{DCG@}n = \sum_{i=1}^{n} \frac{2^{rel_i} - 1}{\log_2(i + 1)}$$

DCG Example

Rank	Retrieved document ID	Gain (relevance)	Discounted gain	DCG
1	$d20$	3	3	3
2	$d243$	2	2/1=2	5
3	$d5$	3	3/1.59=1.89	6.89
4	$d310$	0	0	6.89
5	$d120$	0	0	6.89
6	$d960$	1	1/2.59=0.39	7.28
7	$d234$	2	2/2.81=0.71	7.99
8	$d9$	2	2/3=0.67	8.66
9	$d35$	3	3/3.17=0.95	9.61
10	$d1235$	0	0	9.61

$$\mathrm{DCG@10} = 9.61$$

Normalized DCG (nDCG)

- Depending on the relevance judgements, the range of good/bad DCG results might be different across queries, and hence DCG results of different queries would not be comparable
  - Calculate the mean of DCG values across all test queries is therefore not reasonable

- To normalize DCG at ranking position n:
  - For each query, estimate Ideal DCG (IDCG) which is the DCG for the ranking list sorted by relevance judgements (best possible ranking)
  - Calculate nDCG by dividing DCG by IDCG

- Final nDCG@n is the mean across all test queries

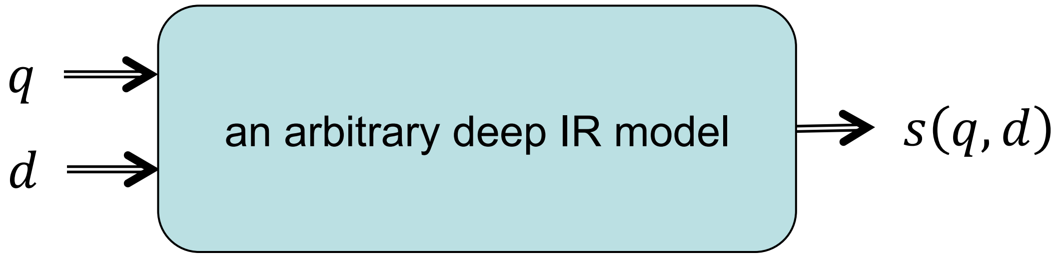
Agenda

- Principles of Information Retrieval
- **Neural IR models**
- Learning to Rank

Learning to predict relevance scores

- Instead of defining a formula as in classical IR models, we can learn to predict relevance scores $s(q,d)$ by training a neural network model

- Such neural/deep IR models can benefit from semantic relations in the embedding space, …
  - Hence do soft-matching between terms, in contrast to exact-matching in classical IR models



Neural IR – task formulation

Training time

- The model receives a given query q to a document d, and learns to calculate the relevance score between them:

$$s(q, d)$$

  - Training is done using LTR (next topic of the lecture)

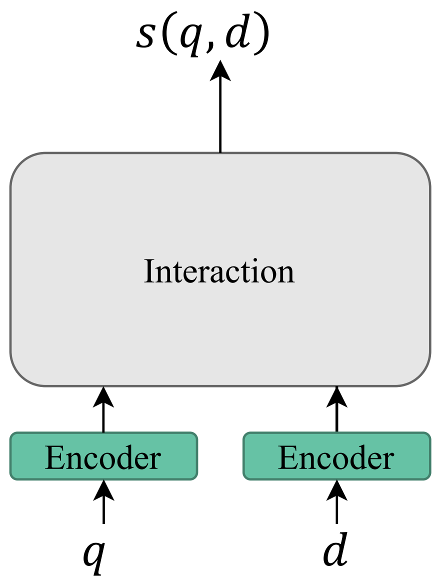
Inference/Retrieval time

- For a given query q and the set of (candidate) documents $[d1, d2, d3, \dots, dM]$, the model calculates the relevance scores:

$$[s(q, d1), s(q, d2), s(q, d3), \dots, s(q, dM)].$$

- This list is sorted from the highest predicted relevance score to the lowest, and the corresponding top documents are retrieved

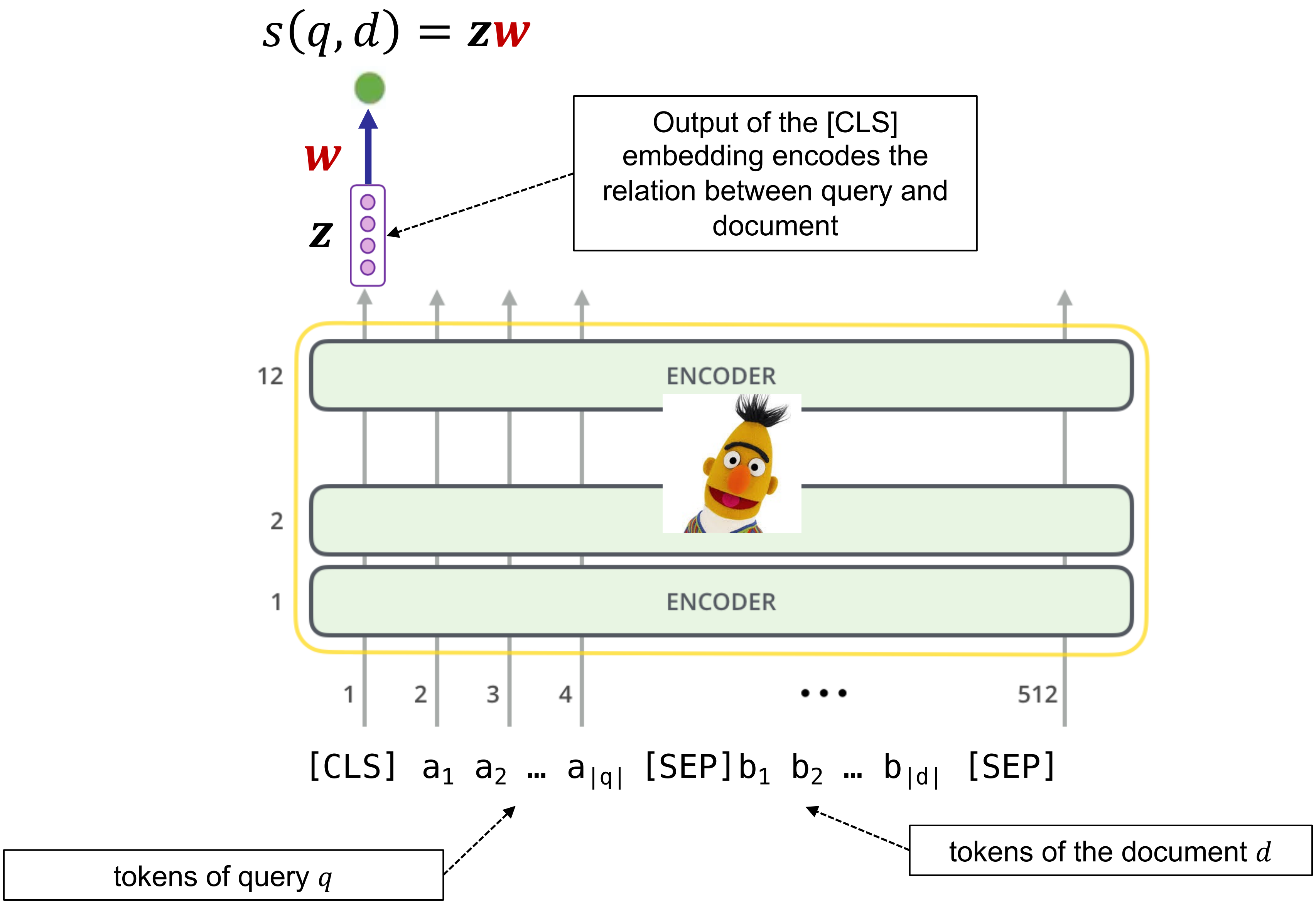
Neural IR paradigms



Interaction-based Retrieval models

- calculate the *interactions* between the input embeddings of the document and query
- output a *feature vector*, representing the relation between query and document
- $s(q,d)$ is calculated from the feature vector

Interaction-based Retrieval models
using an encoder LM like BERT



Interaction-based Retrieval – inference/retrieval time

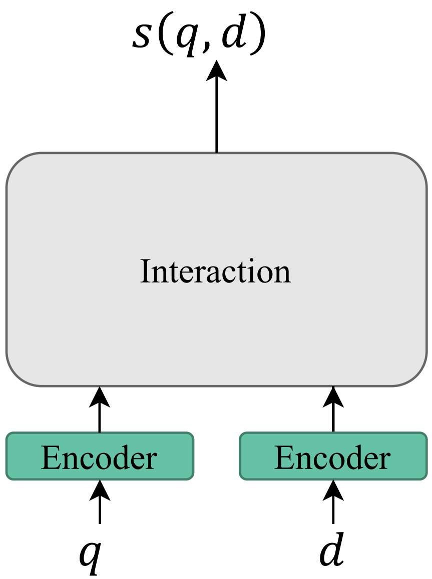
- Neural/deep IR models can't readily use an inverted index for retrieval

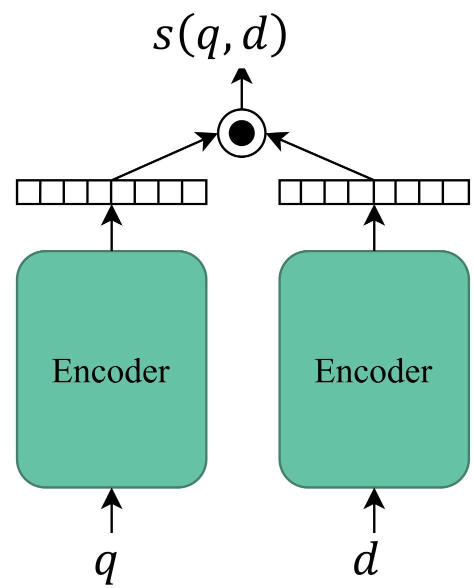
Two (non-optimal) approaches:

- Full-ranking: given a query, calculate relevance scores for all documents, sort the results, and retrieve the documents with highest relevance scores
  - Pros: including all documents, cons: very very expensive!

- Re-ranking: re-rank top-t results of another IR model called first ranker
  - Pass the query to the first ranker and retrieve its top-t documents, called candidate documents
    - First ranker is usually an efficient but weaker IR model like BM25
    - t is usually a number between 100 to 1000
  - Calculate relevance scores for the candidate documents using the (stronger) neural IR model
  - Update the original ranking results by re-ordering (re-ranking) the candidate documents using the new relevance scores
  - Pros: efficiency, cons: there might be relevant documents that do not appear in the candidate set

Neural IR paradigms





Interaction-based Retrieval models

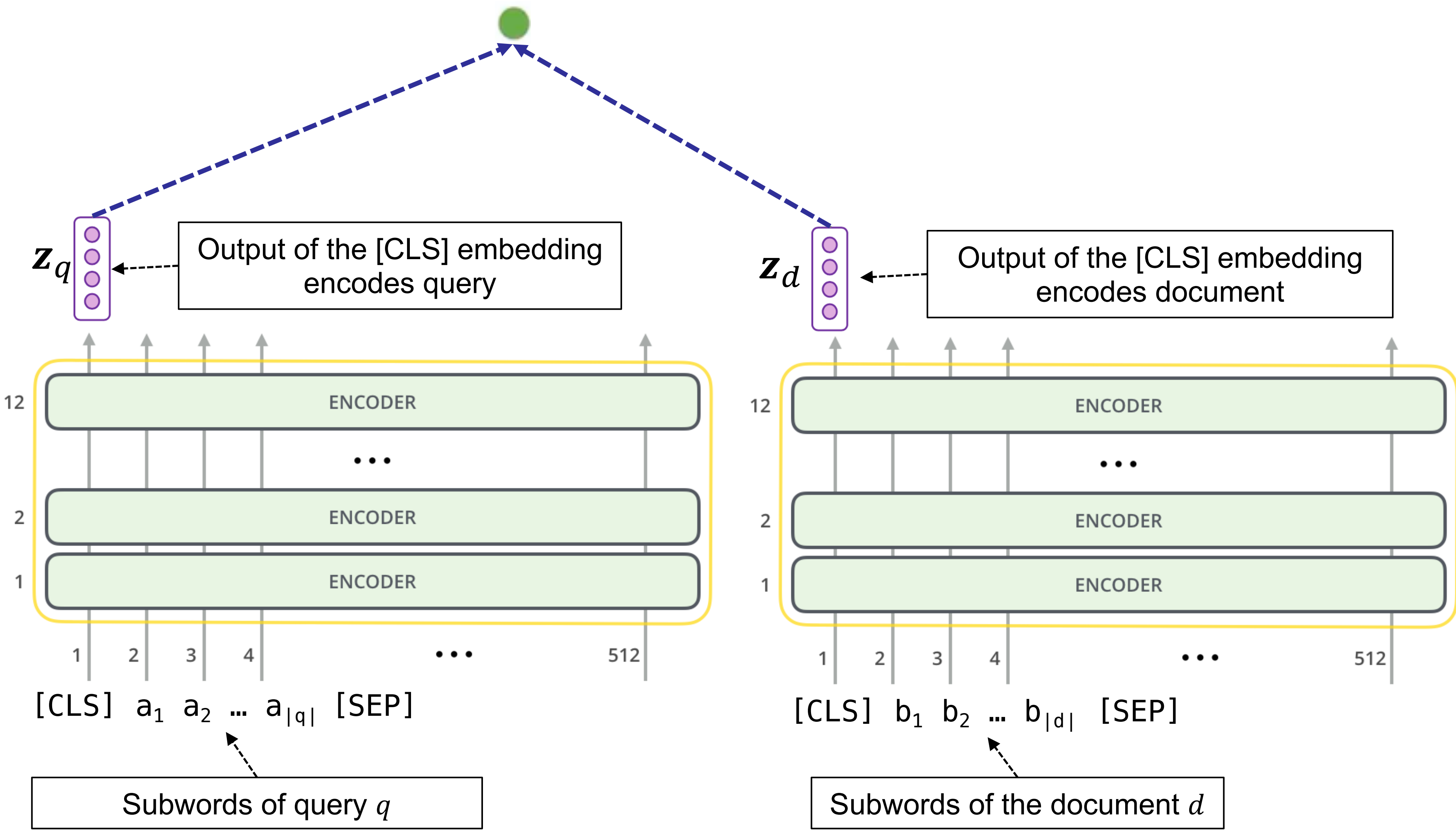
- calculate the *interactions* between the input embeddings of the document and query
- output a *feature vector*, representing the relation between query and document
- $s(q, d)$ is calculated from the feature vector

Dense Retrieval models

- first *encode* the document and the query in two separate vectors
- $s(q, d)$ is then calculated as the *similarity* of the two vectors
- This method enables direct retrieval of documents, achieved by finding the document embeddings which appear at the nearest proximity of the embedding of a query

Dense Retrieval models
using an encoder LM like BERT

$$s(q,d) = \boldsymbol{z}_q \boldsymbol{z}_d$$



Dense Retrieval – inference/retrieval time

- The architecture of Dense Retrieval models enables direct retrieval instead of full-ranking or re-ranking

To retrieve the set of relevant documents …

- After training, the embeddings of all documents ($\boldsymbol{z}_d$) are calculated
  - E.g., the embeddings are often stored/indexed in the data structure of an Approximate Nearest Neighbor (ANN) algorithm for more efficient retrieval
- At inference time, given the query q, …
  - the embedding of the query ($\boldsymbol{z}_q$) is calculated
  - the most similar document embeddings to $\boldsymbol{z}_q$ are retrieved
    - E.g., by calculating the dot product of $\boldsymbol{z}_q$ to all document embeddings, or instead using a highly efficient ANN data structures

- Dense Retrieval models enable highly efficient retrieval (even comparable with classical IR models), but might show a weaker performance in comparison with interaction-based models

Agenda

- Principles of Information Retrieval
- Neural IR models
- **Learning to Rank**

Learning to Rank (LTR)

- It is insufficient to approach the learning of the models with ranking objectives, in the same way as the regression/classification models
- Consider the list of predicted scores by a model:

$$[s(q,d1), s(q,d2), s(q,d3), \dots, s(q,dM)]$$

- The final position of a document can only be known by comparing its predicted score with the ones of other documents
  - For example, only by looking at $s(q,d3) = 1.423$ we can not know in which position document $d3$ will end up

How should a model learn to predict scores according to a rank?!

- Learning to rank approaches:
  - Pointwise
  - Pairwise
  - Listwise (out of the scope of this lecture)

LTR Pointwise/Pairwise – training data

- For a given query q, training data consists of …
- a (small) set of relevant or positive documents:

$$D_{+}^{q} = [d_{+}^{(1)}, d_{+}^{(2)}, \dots]$$

  - Each d_{+} is a document judged as relevant to q
  - Usually only a few positive documents per each query are available
- as well as a set of non-relevant or negative documents:

$$D_{-}^{q} = [d_{-}^{(1)}, d_{-}^{(2)}, \dots]$$

  - Each d_{-} can be …
    - a document judged as non-relevant to q (usually only a few are available)
    - a randomly sampled document from the collection – random negatives
    - a document sampled from a list of candidate documents, like from the top 1000 retrieved documents for q using a first ranker– hard negatives

Available collections with large training data

- MS MARCO (Microsoft MAchine Reading Comprehension)
  - Queries and retrieved passages of BING, annotated by human

	MS MARCO [28]
# of documents	8,841,822
Average document length	58.8 ± 23.5
Average query length	6.3 ± 2.6
# of training data points	39,780,811
# of validation queries	6,980
# of test queries	48,598

- TripClick (Collection & Log Files of a Health Web Search Engine)
  - Queries and clicked documents of TripDatabase search engine

Number of query-document interactions	4,054,593
Number of documents	1,523,878
Number of queries (all/HEAD/TORSO/TAIL)	692,699 / 5,879 / 108,314 / 578,506
Average query length	4.4 ± 2.4
Average document length	259.0 ± 81.7

https://microsoft.github.io/msmarco/
https://tripdatabase.github.io/tripclick/

Pointwise LTR

- Pointwise LTR models learn the relevance prediction of every positive/negative document <u>independently of the other documents</u>
  - Pointwise models are in fact classification/regression models

- Training data is therefore prepared in the form of:

`[input=(query, document), label(y)=relevance score]`

Example: For the query q

$$\left[\text{input} = \left(q, d_+^{(1)}\right), y = 1\right]$$
$$\left[\text{input} = \left(q, d_+^{(2)}\right), y = 1\right]$$
$$\left[\text{input} = \left(q, d_+^{(3)}\right), y = 1\right]$$
$$\ldots$$
$$\left[\text{input} = \left(q, d_-^{(1)}\right), y = 0\right]$$
$$\left[\text{input} = \left(q, d_-^{(2)}\right), y = 0\right]$$
$$\left[\text{input} = \left(q, d_-^{(3)}\right), y = 0\right]$$
$$\ldots$$

Pointwise LTR – loss

- Similar to classification tasks, Cross Entropy is a commonly used as the loss of pointwise LTR:

$$\mathcal{L} = -\mathbb{E}_{[(q,d),y]\sim\mathcal{T}}[y \log \sigma(s(q,d))]$$

  - $\mathcal{T} \rightarrow$ the set of all training data
  - $\sigma(s(q,d)) \rightarrow$ sigmoid applied to the predicted score to turn the score into a probability

Pairwise LTR

- Pair-wise LTR is applied to pairs of positive-negative documents
- Pair-wise optimization aims to make the predicted score of a query to a relevant document higher than the one to a non-relevant document: $s(q, d_+) > s(q, d_-)$
  - This means that the IR model learns to give a higher relevance score to d_+ and therefore rank d_+ in a higher position than d_-. This (hopefully) leads to a better overall ranking results for the given query.
- The training data is therefore provided in the form of triplets:

```
[query, positive-document, negative-document]
```

Example: For the query q

$$\begin{array}{ccc} \left[q, d_+^{(1)}, d_-^{(1)}\right] & \left[q, d_+^{(2)}, d_-^{(1)}\right] & \dots \\ \left[q, d_+^{(1)}, d_-^{(2)}\right] & \left[q, d_+^{(2)}, d_-^{(2)}\right] & \dots \\ \left[q, d_+^{(1)}, d_-^{(3)}\right] & \left[q, d_+^{(2)}, d_-^{(3)}\right] & \dots \\ \dots & \dots & \dots \end{array}$$

Pairwise LTR – Max Margin loss

- Max-Margin is a widely used loss function for pair-wise training
  - Also called *Hinge loss*, *contrastive loss*, or *margin objective*

- Max-Margin ranking loss "punishes" the network until a given margin hyperparameter C is held between the predicted scores of the relevant and non-relevant documents:

$$\mathcal{L} = \mathbb{E}_{(q,d_+,d_-)\sim\mathcal{T}}[\max(0, C - (s(q, d_+) - s(q, d_-)))]$$

Examples when $C = 1$:

If $s(q, d_+) = 2$ and $s(q, d_-) = 1.8 \rightarrow \mathcal{L} = 0.8$

If $s(q, d_+) = 2$ and $s(q, d_-) = 3.8 \rightarrow \mathcal{L} = 2.8$

If $s(q, d_+) = 2$ and $s(q, d_-) = 0.8 \rightarrow \mathcal{L} = 0.0$