# Information Retrieval

Kevin Duh
Johns Hopkins University

# Acknowledgments

These slides draw heavily from these excellent sources:

- Paul McNamee's JSALT2018 tutorial:
  - https://www.clsp.jhu.edu/wp-content/uploads/sites/75/2018/06/2018-06-19-McNamee-JSALT-IR-Soup-to-Nuts.pdf
- Doug Oard's Information Retrieval Systems course at UMD
  - http://users.umiacs.umd.edu/~oard/teaching/734/spring18/
- Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze, Introduction to Information Retrieval, Cambridge U. Press. 2008.
  - https://nlp.stanford.edu/IR-book/information-retrieval-book.html
- W. Bruce Croft, Donald Metzler, Trevor Strohman, Search Engines: Information Retrieval in Practice, Pearson, 2009
  - http://ciir.cs.umass.edu/irbook/

***I never waste memory on things that can easily be stored and retrieved from elsewhere.***

***-- Albert Einstein***

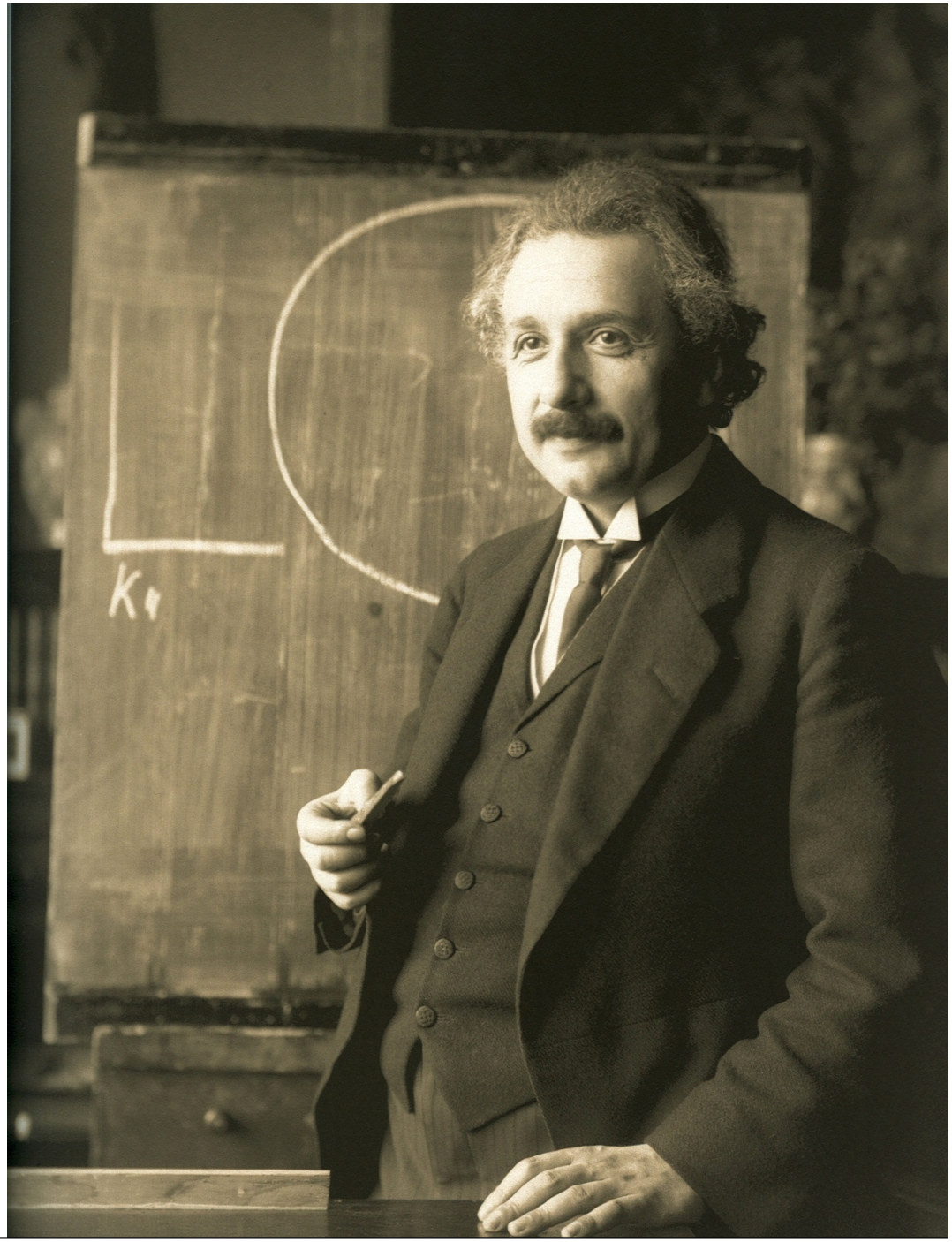




Image source: Einstein 1921 by F Schmutzer
https://en.wikipedia.org/wiki/Albert_Einstein#/media/File:Einstein_1921_by_F_Schmutzer_-_restoration.jpg

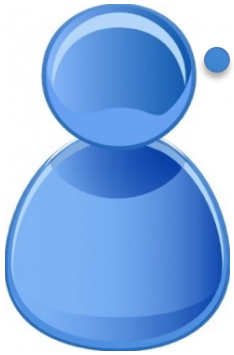
# What is Information Retrieval (IR)?

1. Information retrieval is a field concerned with the **structure, analysis, organization, storage, searching, & retrieval of information**.

   (Gerard Salton, IR pioneer, 1968)

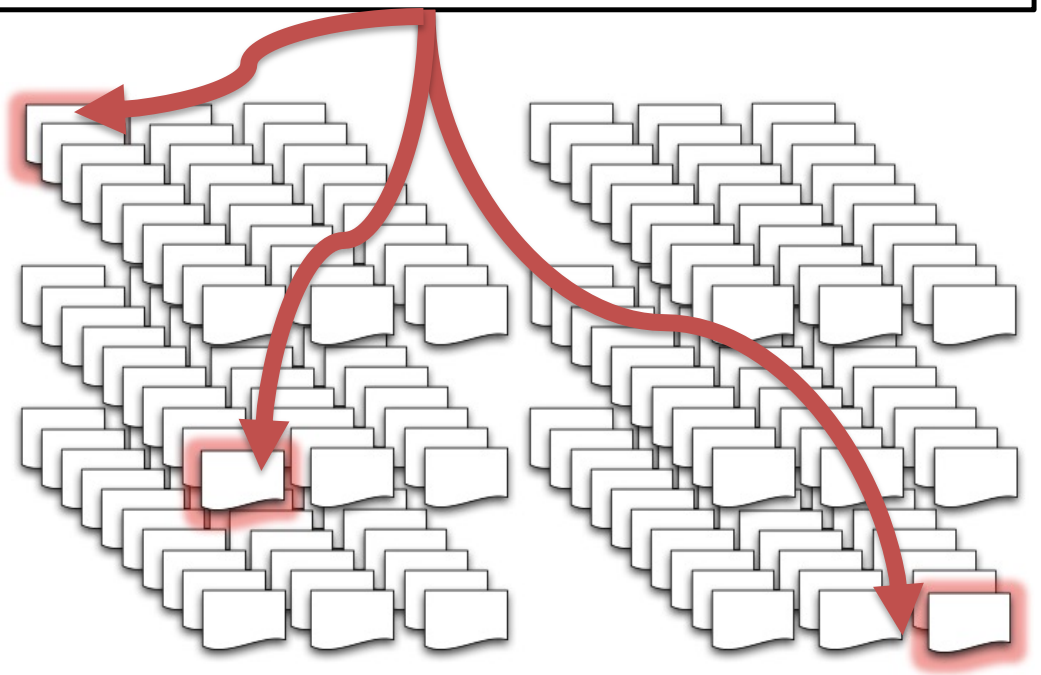
2. Information retrieval focuses on the efficient recall of information that **satisfies a user's information need**.

INFO NEED: I need to understand why I'm getting a NullPointer Exception when calling randomize() in the FastMath library

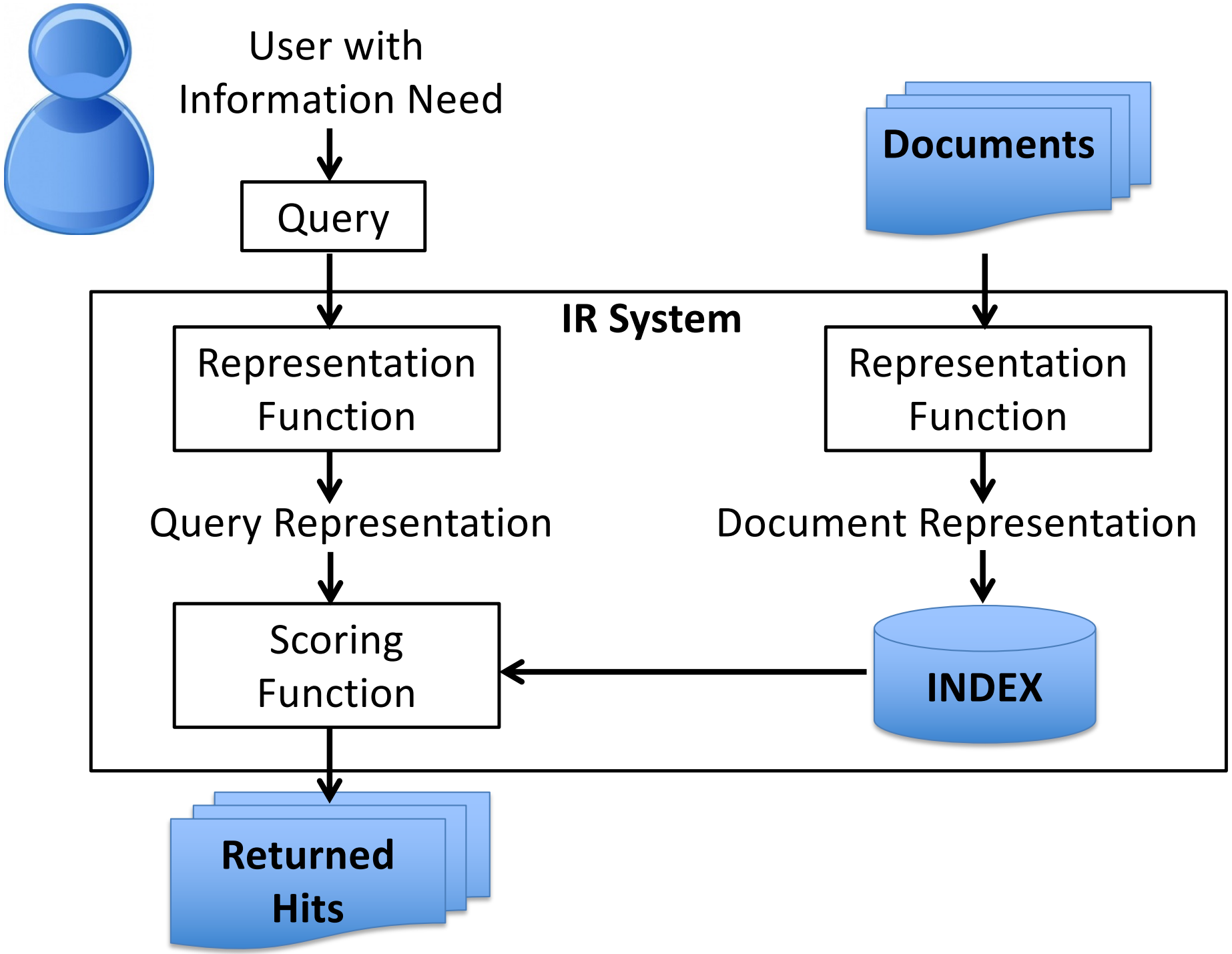


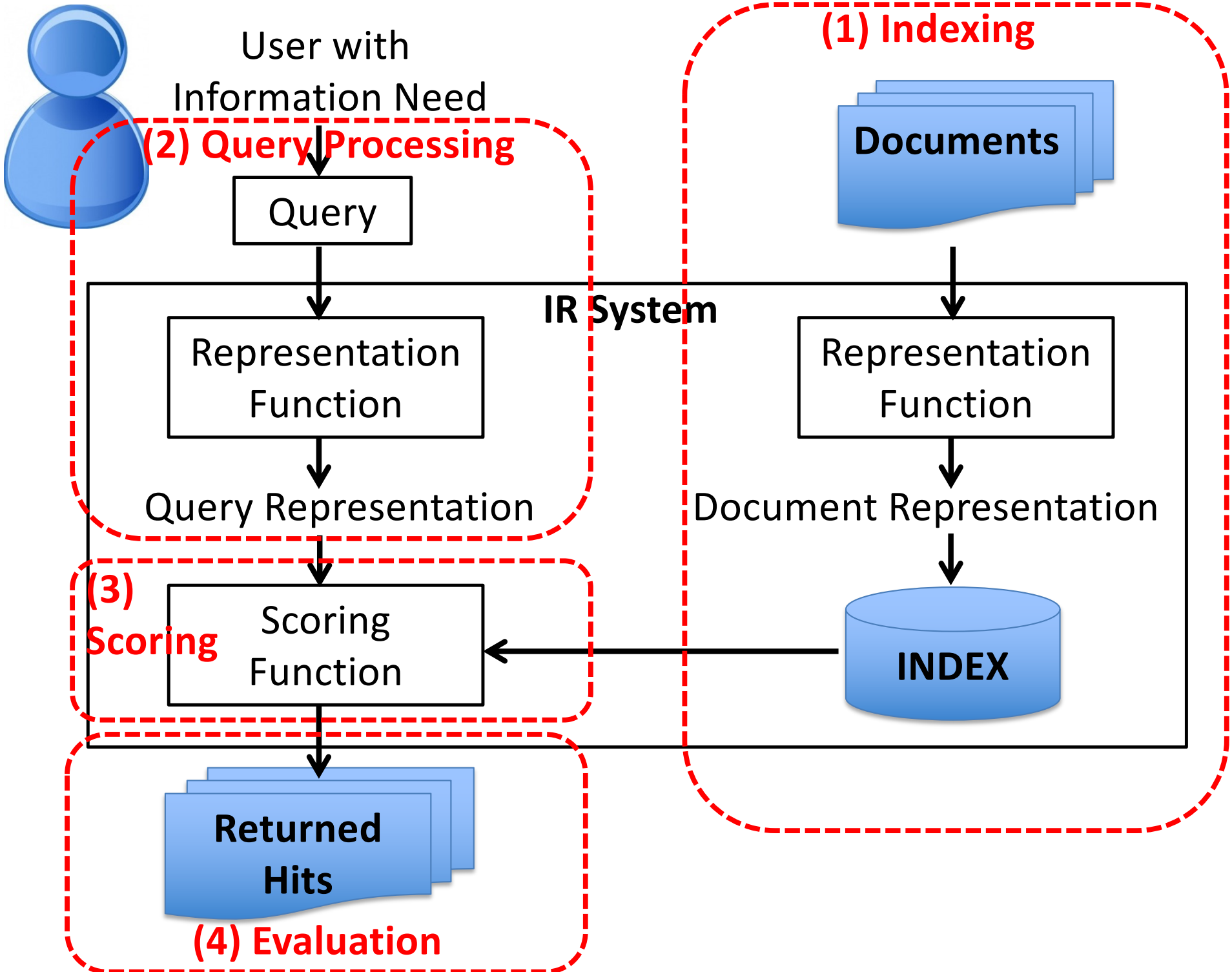
QUERY:
NullPointer Exception randomize() FastMath

Web documents
that may be relevant



# Structure of IR System


User with
Information Need
Query
Documents
IR System
Representation
Function
Representation
Function
Query Representation
Document Representation
Scoring
Function
INDEX
Returned
Hits


User with
Information Need
(1) Indexing
Documents
(2) Query Processing
Query
IR System
Representation
Function
Representation
Function
Query Representation
Document Representation
(3)
Scoring
Scoring
Function
INDEX
Returned
Hits
(4) Evaluation

# Index vs Grep

- Say we have collection of Shakespeare plays
- We want to find all plays that contain:

QUERY:
Brutus AND Caesar AND NOT Calpurnia

- Grep: Start at 1st play, read everything and filter if criteria doesn't match (linear scan, 1M words)
- Index (a.k.a. Inverted Index): build index data structure off-line. Quick lookup at query-time.

These examples/figures are from: Manning, Raghavan, Schütze, Intro to Information Retrieval, CUP, 2008

# The Shakespeare collection as Term-Document Incidence Matrix

| | Antony 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 | |
| … | | | | | | | |

Matrix element (t,d) is:

1 if term t occurs in document d,

0 otherwise

These examples/figures are from: Manning, Raghavan, Schütze, Intro to Information Retrieval, CUP, 2008

# The Shakespeare collection as Term-Document Incidence Matrix

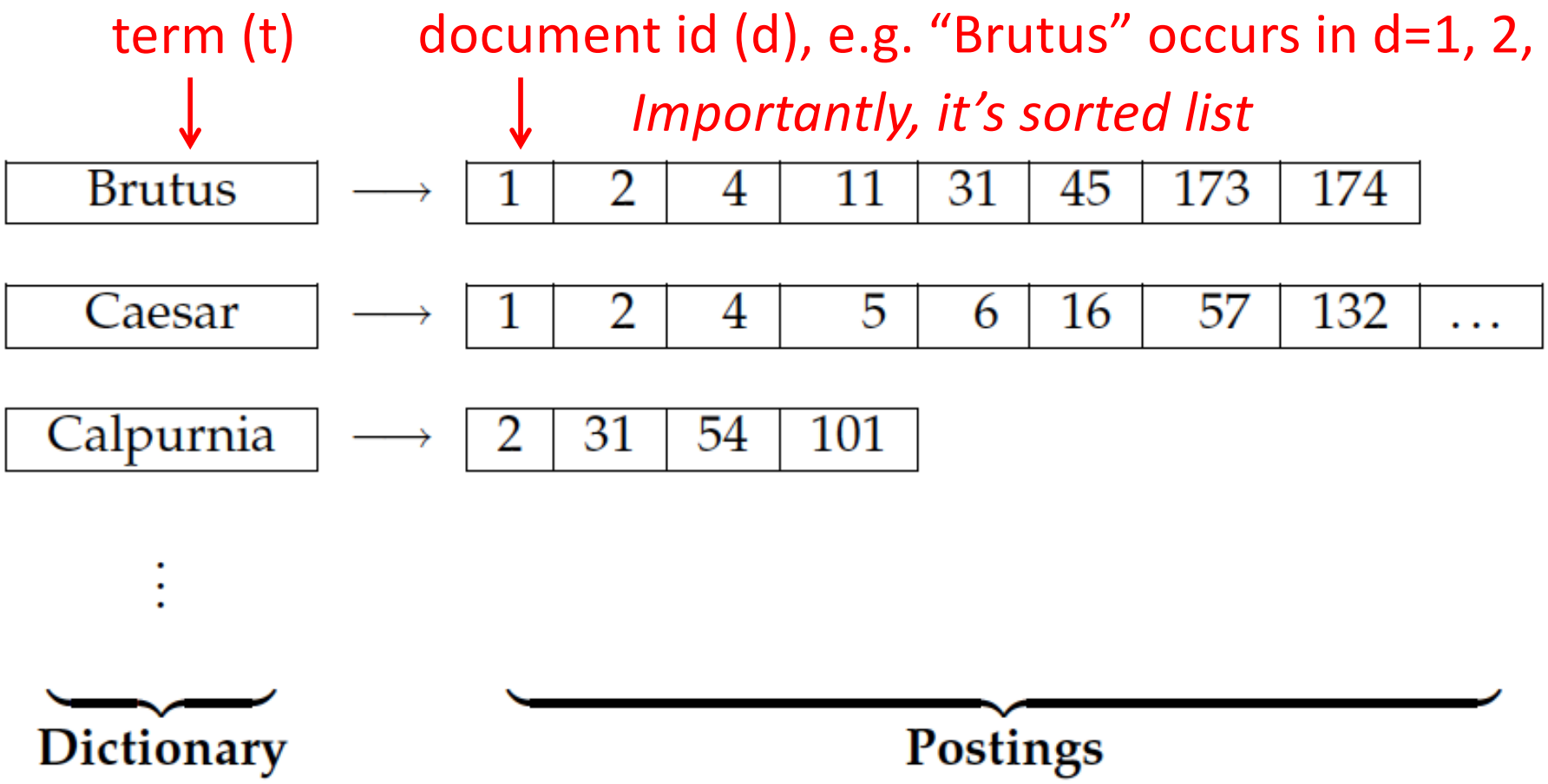
| | Antony and Cleopatra | Julius Caesar | The Tempest | Hamlet | Othello | Macbeth | … |
|---|---|---|---|---|---|---|---|
| Antony | 1 | 1 | 0 | 0 | 0 | 1 | |
| Brutus | 1 | 1 | 0 | 1 | 0 | 0 | |
| Caesar | 1 | 1 | 0 | 1 | 1 | 1 | |
| Calpurnia | 0 | 1 | 0 | 0 | 0 | 0 | |
| Cleopatra | 1 | 0 | 0 | 0 | 0 | 0 | |
| mercy | 1 | 0 | 1 | 1 | 1 | 1 | |
| worser | 1 | 0 | 1 | 1 | 1 | 0 | |
| … | | | | | | | |

QUERY:
Brutus AND Caesar AND NOT Calpurnia

*Answer: "Antony and Cleopatra"(d=1), "Hamlet"(d=4)*

These examples/figures are from: Manning, Raghavan, Schütze, Intro to Information Retrieval, CUP, 2008

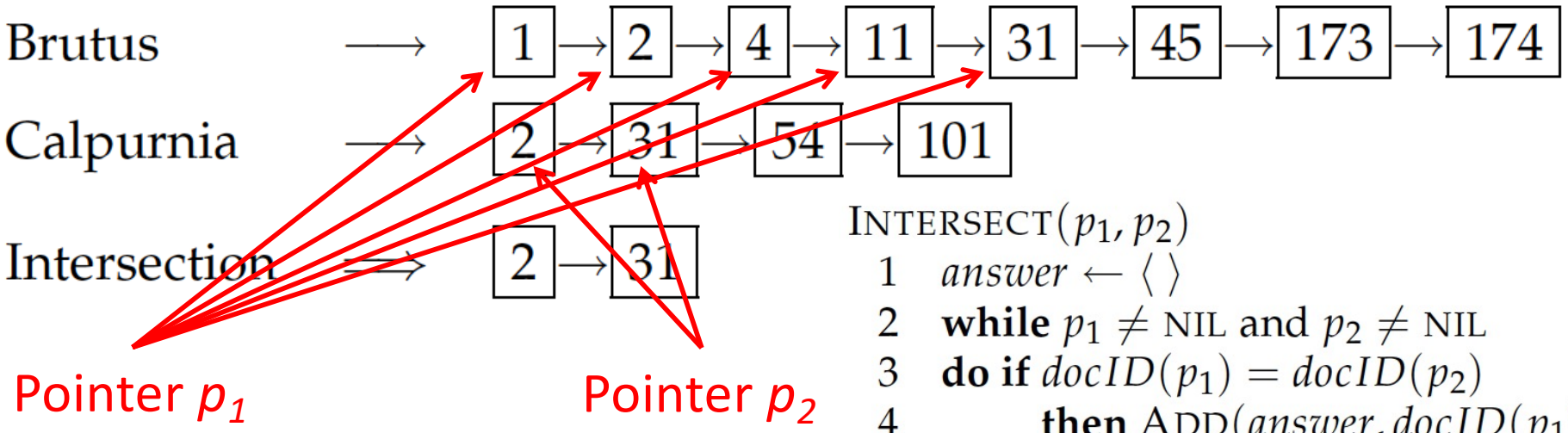
# Inverted Index Data Structure




These examples/figures are from: Manning, Raghavan, Schütze, Intro to Information Retrieval, CUP, 2008

# Efficient algorithm for List Intersection (for Boolean conjunctive "AND" operators)

QUERY:
Brutus AND Calpurnia




```
INTERSECT(p1, p2)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then ADD(answer, docID(p1))
 5             p1 ← next(p1)
 6             p2 ← next(p2)
 7        else if docID(p1) < docID(p2)
 8                then p1 ← next(p1)
 9                else p2 ← next(p2)
10  return answer
```

These examples/figures are from: Manning, Raghavan, Schütze, Intro to Information Retrieval, CUP, 2008

# Time and Space Tradeoffs

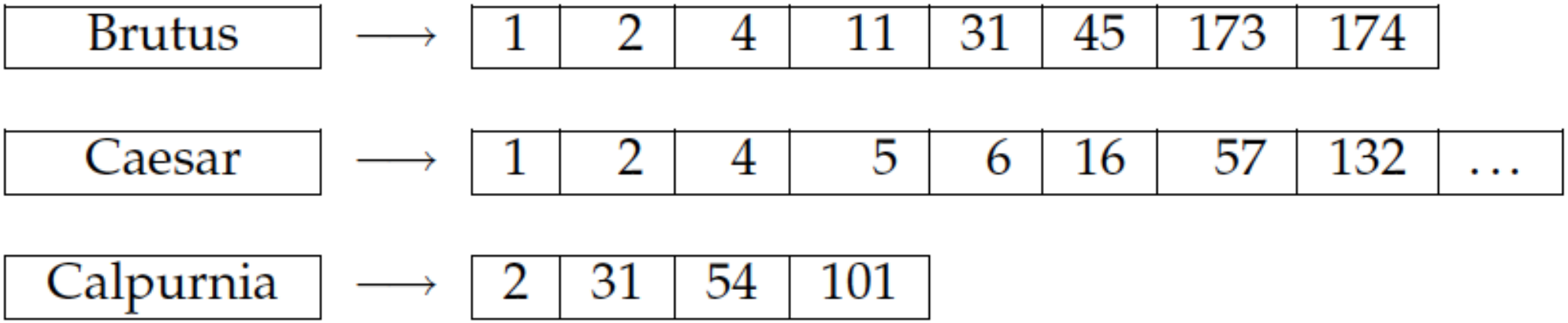
- Time complexity at query-time:
  - Linear scan over postings
  - $O(L_1 + L_2)$ where $L_t$ is length of posting for term t
  - vs. grep through all documents O(N), L << N
- Time complexity at index-time:
  - O(N) for one pass through collection
  - Additional issue: efficient adding/deleting documents
- Space complexity (example setup):
  - Dictionary: Hash/Trie in RAM
  - Postings: Array on disk

# *Quiz: How would you process these queries?*

QUERY:
Brutus AND Caesar AND Calpurnia

QUERY:
Brutus AND (Caesar OR Calpurnia)

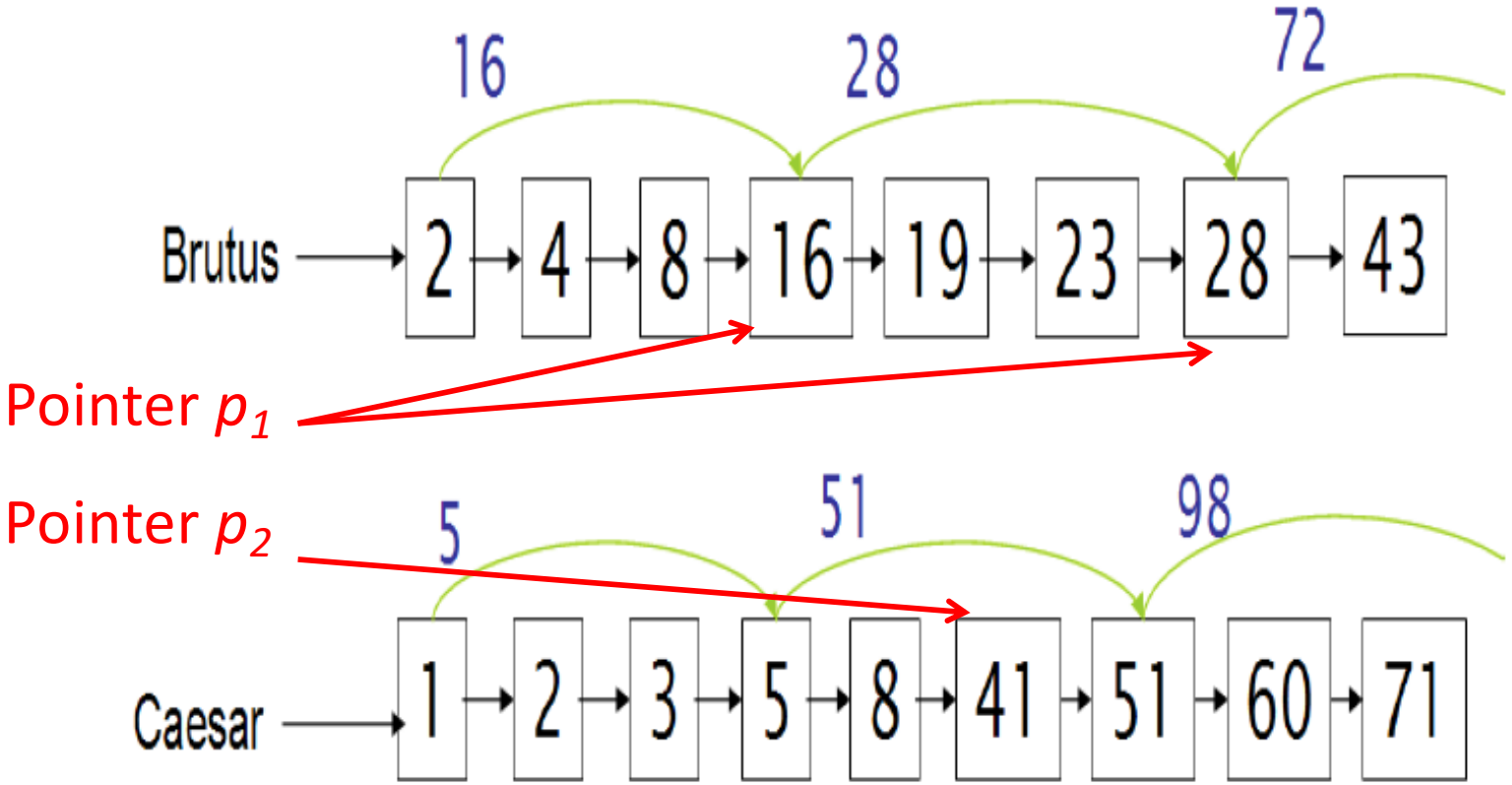
QUERY:
Brutus AND Caesar AND NOT Calpurnia




*Think: What terms to process first? How to handle OR, NOT?*

# Optional meta-data in inverted index

- Skip pointers: For faster intersection, but extra space




These examples/figures are from: Manning, Raghavan, Schütze, Intro to Information Retrieval, CUP, 2008

# Optional meta-data in inverted index

- Position of term in document: Enables phrasal queries

QUERY:
"to be or not to be"

to, 993427:
⟨ 1, 6: ⟨7, 18, 33, 72, 86, 231⟩;
2, 5: ⟨1, 17, 74, 222, 255⟩;
4, 5: ⟨8, 16, 190, 429, 433⟩;
5, 2: ⟨363, 367⟩;
7, 3: ⟨13, 23, 191⟩; . . . ⟩

term (t)

document frequency

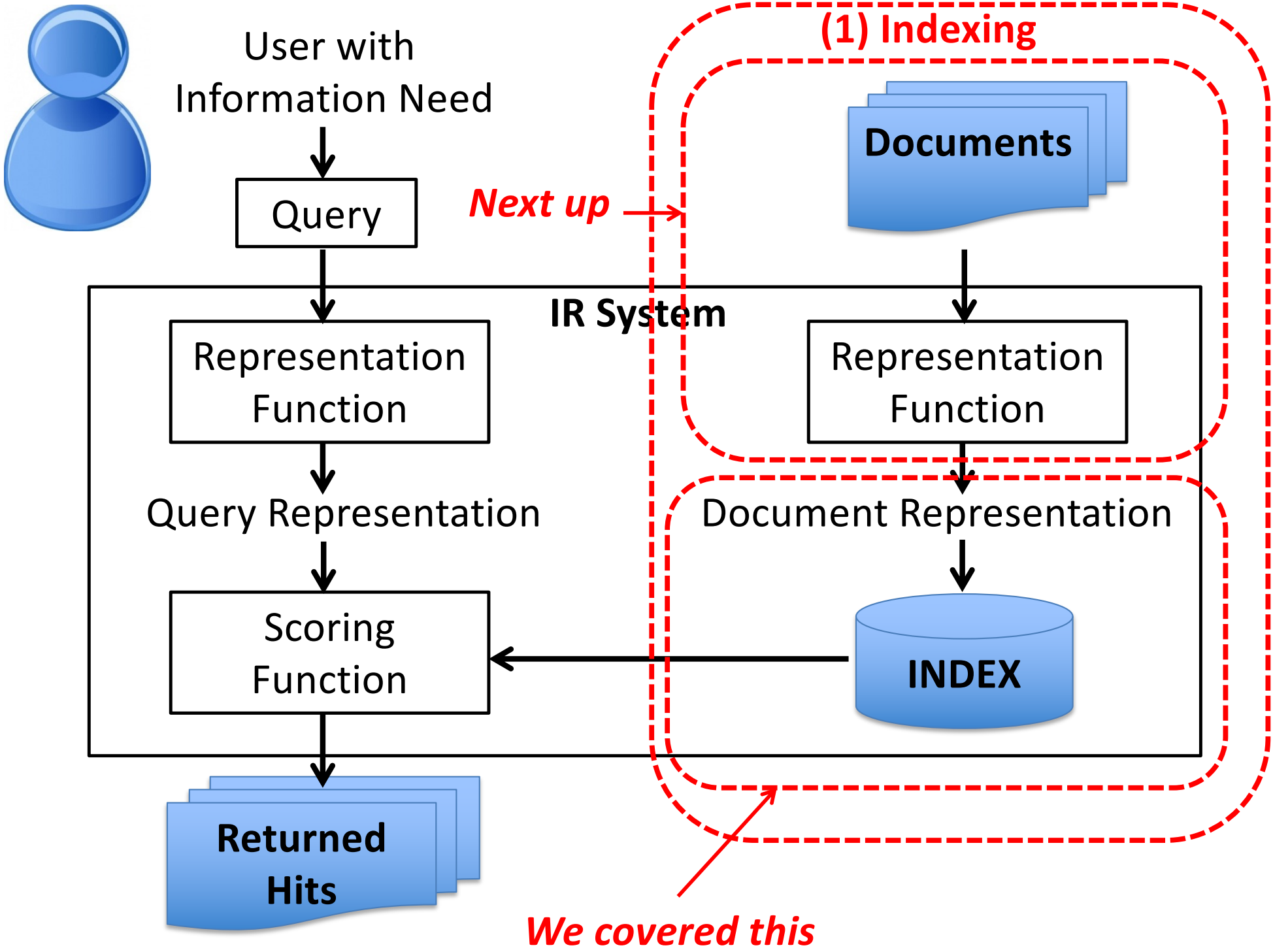
be, 178239:
⟨ 1, 2: ⟨17, 25⟩;
4, 5: ⟨17, 191, 291, 430, 434⟩;
5, 3: ⟨14, 19, 101⟩; . . . ⟩

term occurs in document d=4
with term frequency of 5,
at positions 17, 191, 291, 430, 434

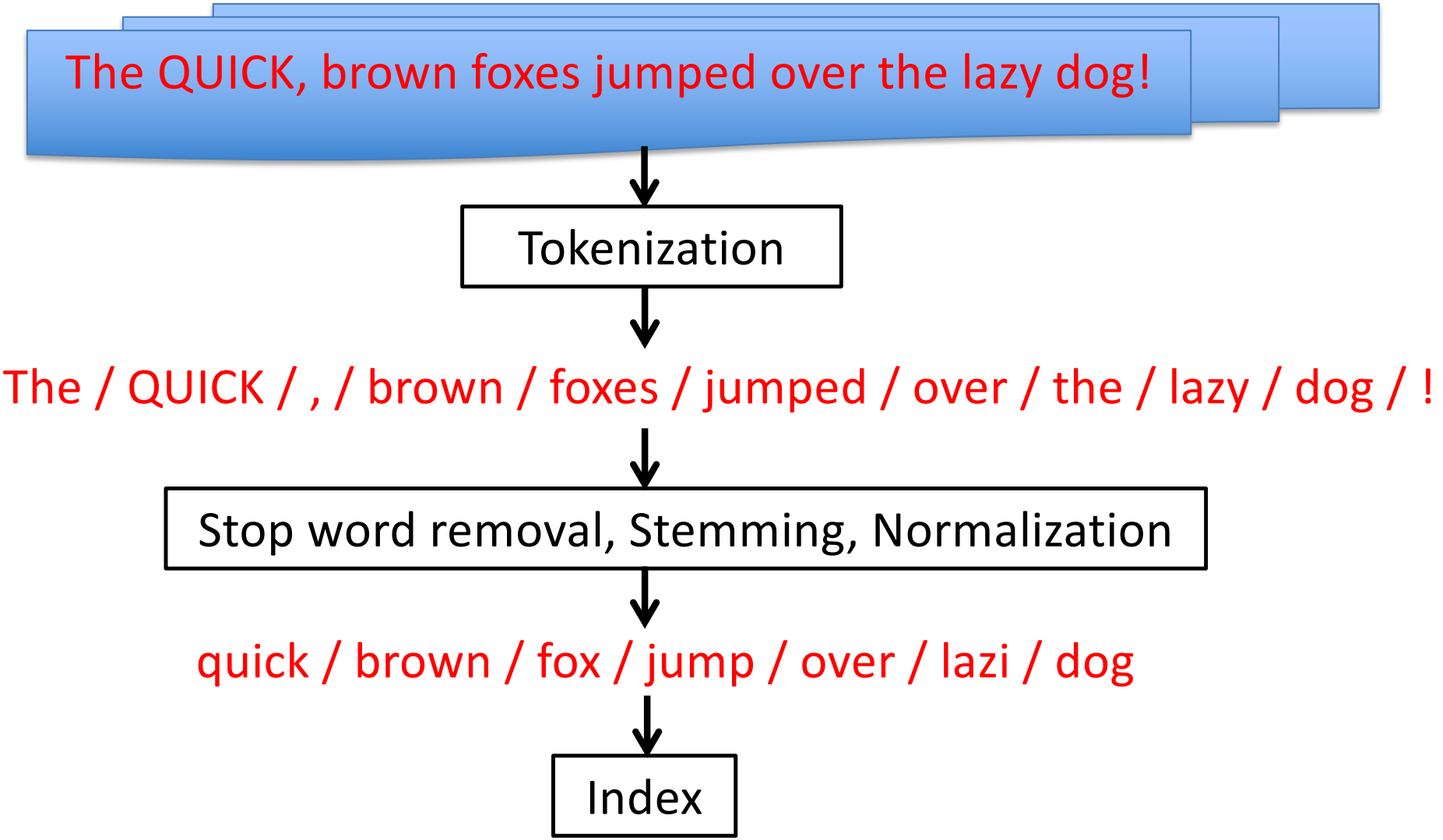
# Index construction and management

- Dynamic index
  - Searching Twitter vs. static document collection
- Distributed solutions
  - MapReduce, Hadoop, etc.
  - Fault tolerance
- Pre-computing components for score function

→ Many interesting technical challenges!


User with
Information Need
Query
Next up
(1) Indexing
Documents
IR System
Representation
Function
Representation
Function
Query Representation
Document Representation
Scoring
Function
INDEX
Returned
Hits
We covered this

# Representing a Document as a Bag-of-words (but what words?)



# Issues in Document Representation

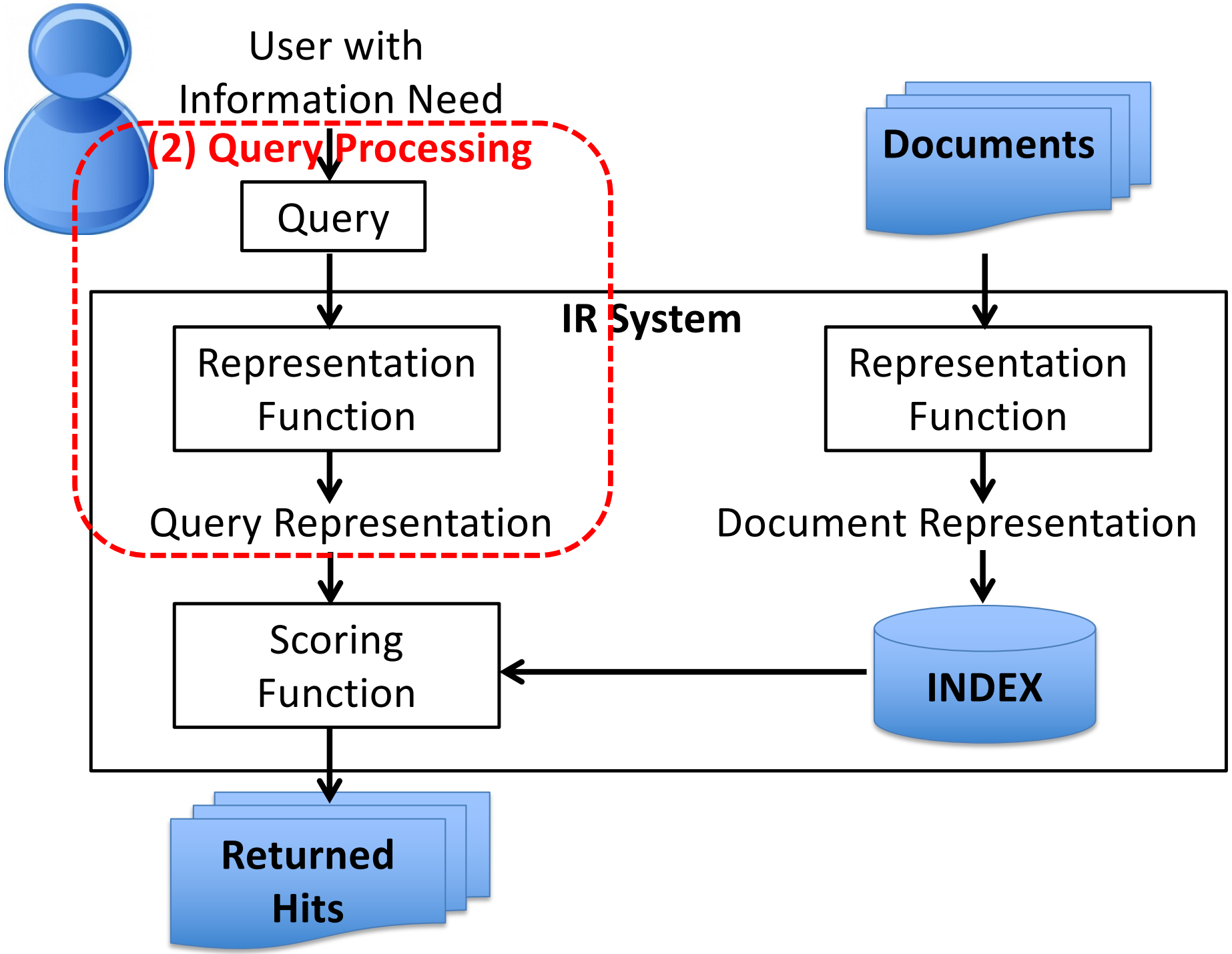
- Language-specific challenges
- Polysemy & Synonyms:
  - "bank" in multiple senses, represented the same?
  - "jet" and "airplane" should be same?
- Acronyms, Numbers, Document structure
- Morphology

aghnaaguq

| | | | |
|---|---|---|---|
| aghnagh- | -∼:(ng)u- | -∼$_f$(g/t)u- | -q |
| woman- | -to.be.N- | -INTR.IND- | -3SG |

'*She is a woman*'

Central Siberian Yupik morphology example from E. Chen & L. Schartz, LREC 2018: http://dowobeha.github.io/papers/lrec18.pdf


User with
Information Need
(2) Query Processing
Query
Documents
IR System
Representation
Function
Representation
Function
Query Representation
Document Representation
Scoring
Function
INDEX
Returned
Hits

# Query Representation

- Of course, the query string must go through the same tokenization, stop word removal and normalization process like the documents

- But we can do more, esp. for free-text queries
  - to guess user's intent & information need

# Keyword search vs. Conceptual search

- Keyword search / Boolean retrieval:

BOOLEAN QUERY:
Brutus AND Caesar AND NOT Calpurnia

  - Answer is exact, must satisfy these terms

- Conceptual search (or just “search” like Google)

FREE-TEXT QUERY:
Brutus assassinate Caesar reasons

  - Answer may not need to exactly match these terms
  - *Note this naming may not be standard*

# Query Expansion for “conceptual” search

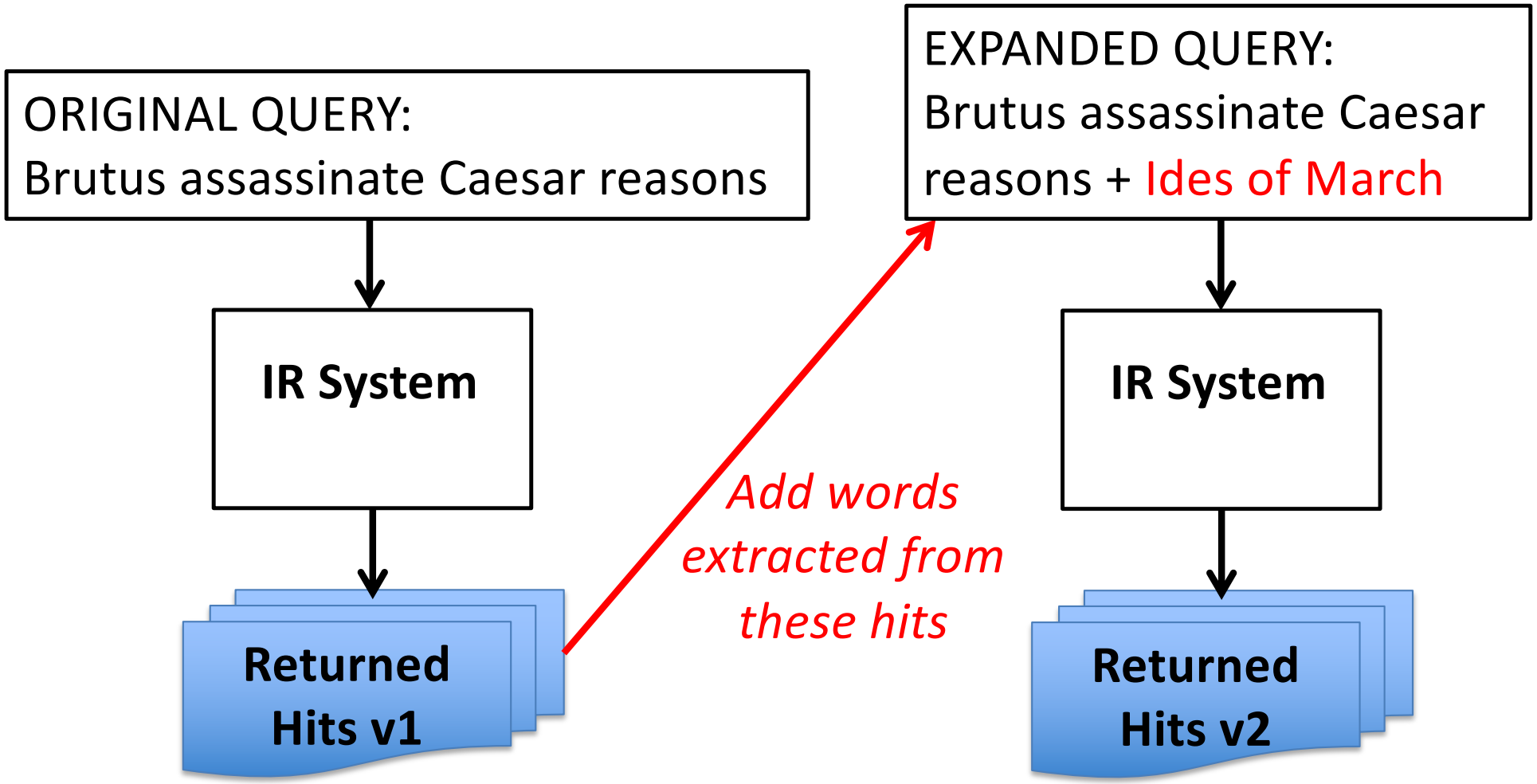
- Add terms to the query representation
  - Exploit knowledge base, WordNet, user query logs

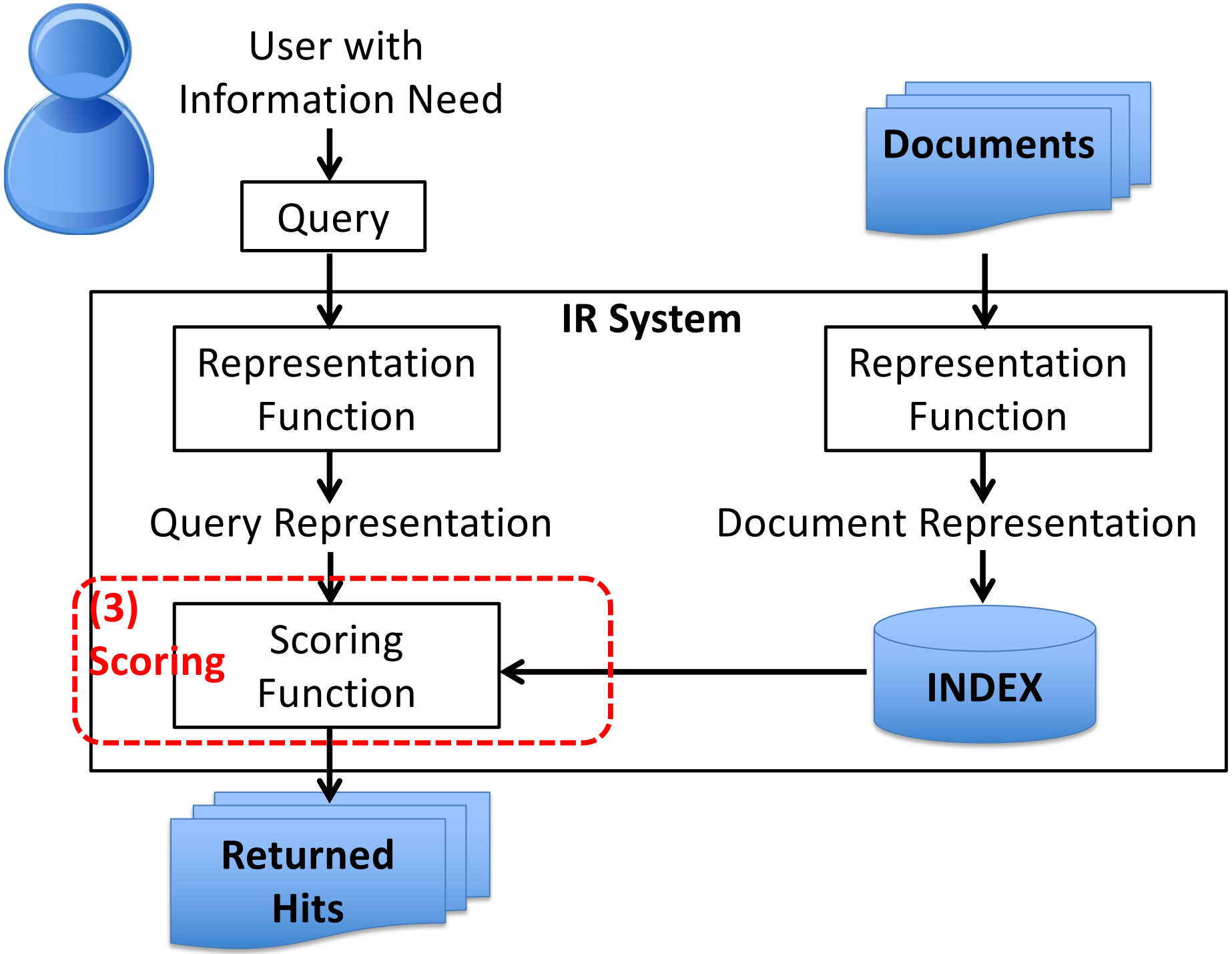
ORIGINAL FREE-TEXT QUERY:
Brutus assassinate Caesar reasons

EXPANDED QUERY:
Brutus assassinate kill Caesar reasons why

# Pseudo-Relevance Feedback

- Query expansion by iterative search




User with
Information Need
Query
Documents
IR System
Representation
Function
Representation
Function
Query Representation
Document Representation
(3)
Scoring
Scoring
Function
INDEX
Returned
Hits

# Motivation for scoring documents

- For keyword search, all documents returned should satisfy query, and are equally relevant
- For conceptual search:
  - May have too many returned documents
  - Relevance is a gradation

  → *Score* documents and return a *ranked list*

# TF-IDF Scoring Function

- Given query *q* and document *d*

$$\text{TF-IDF}(q,d) = \sum_{t \in q} \text{tf}_{t,d} \times \text{idf}_t$$

terms t in q

Term frequency (raw count) of t in d

Inverse document frequency

$$\text{idf}_t = \log \frac{N}{\text{df}_t}$$

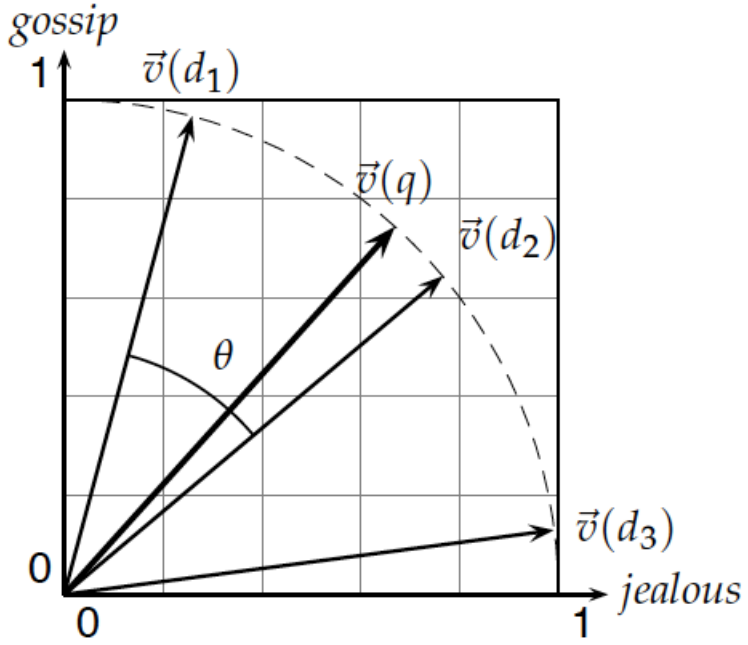
$N$ ← Total number of documents

$\text{df}_t$ ← Number of documents with >=1 occurrence of t

# Vector-Space Model View

- View documents (d) & queries (q) each as vectors,
  - Each vector element represents a term
  - whose value is the TF-IDF of that term in d or q
- Score function can be viewed as e.g. Cosine Similarity between vectors

$$\text{score}(q,d) = \frac{\vec{V}(q) \cdot \vec{V}(d)}{|\vec{V}(q)||\vec{V}(d)|}$$




These examples/figures are from: Manning, Raghavan, Schütze, Intro to Information Retrieval, CUP, 2008

# Alternative Scoring Functions: BM25

$$score(q,d) = \sum_{t \in q} \mathrm{idf_t} \times \frac{\mathrm{tf_{t,d}} \cdot (k_1 + 1)}{\mathrm{tf_{t,d}} + k_1 \cdot (1 - b + b \cdot \frac{|D|}{avgdl})}$$

Query

Document

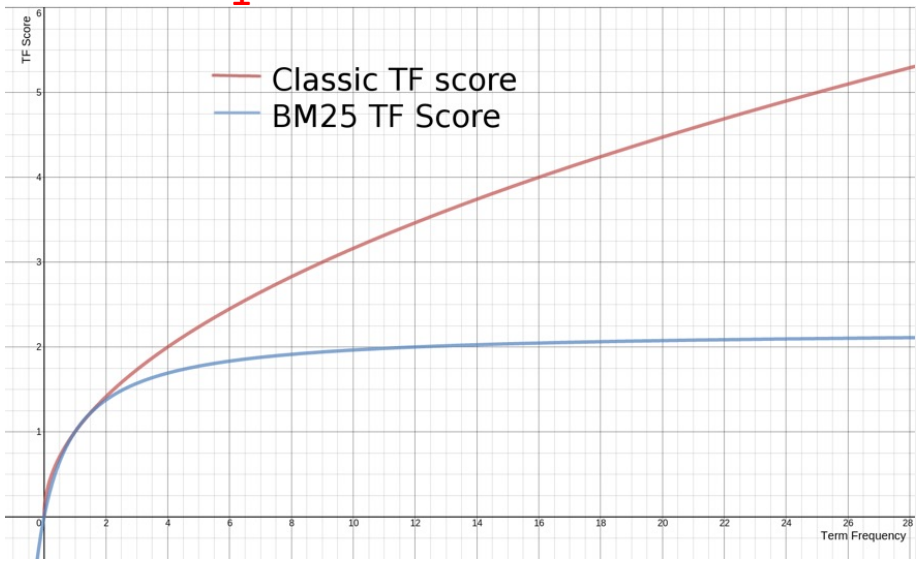
Inverse Document Frequency of query term

Frequency of query term in document

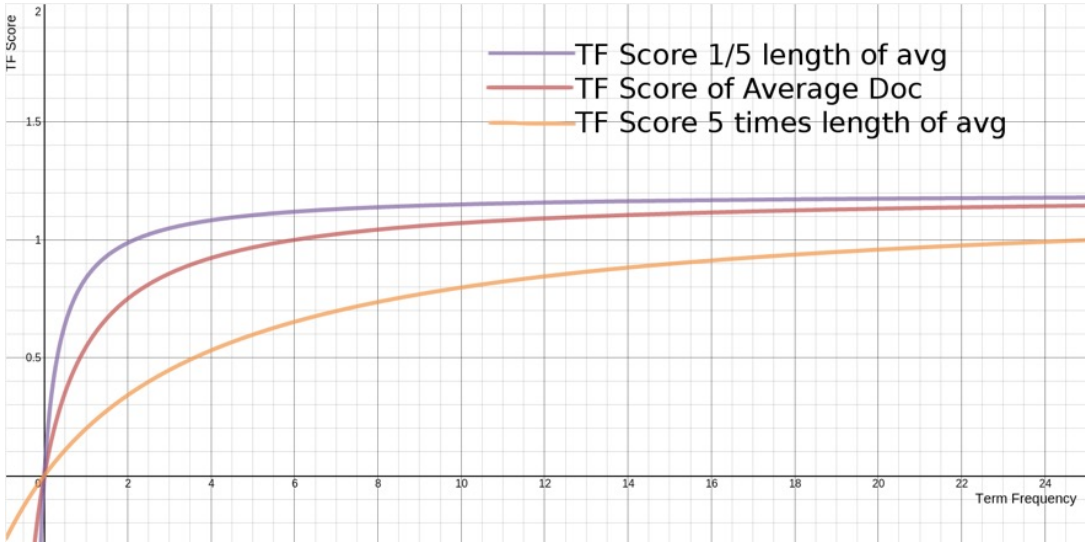
Document length ratio

**Tunable Hyperparameters**

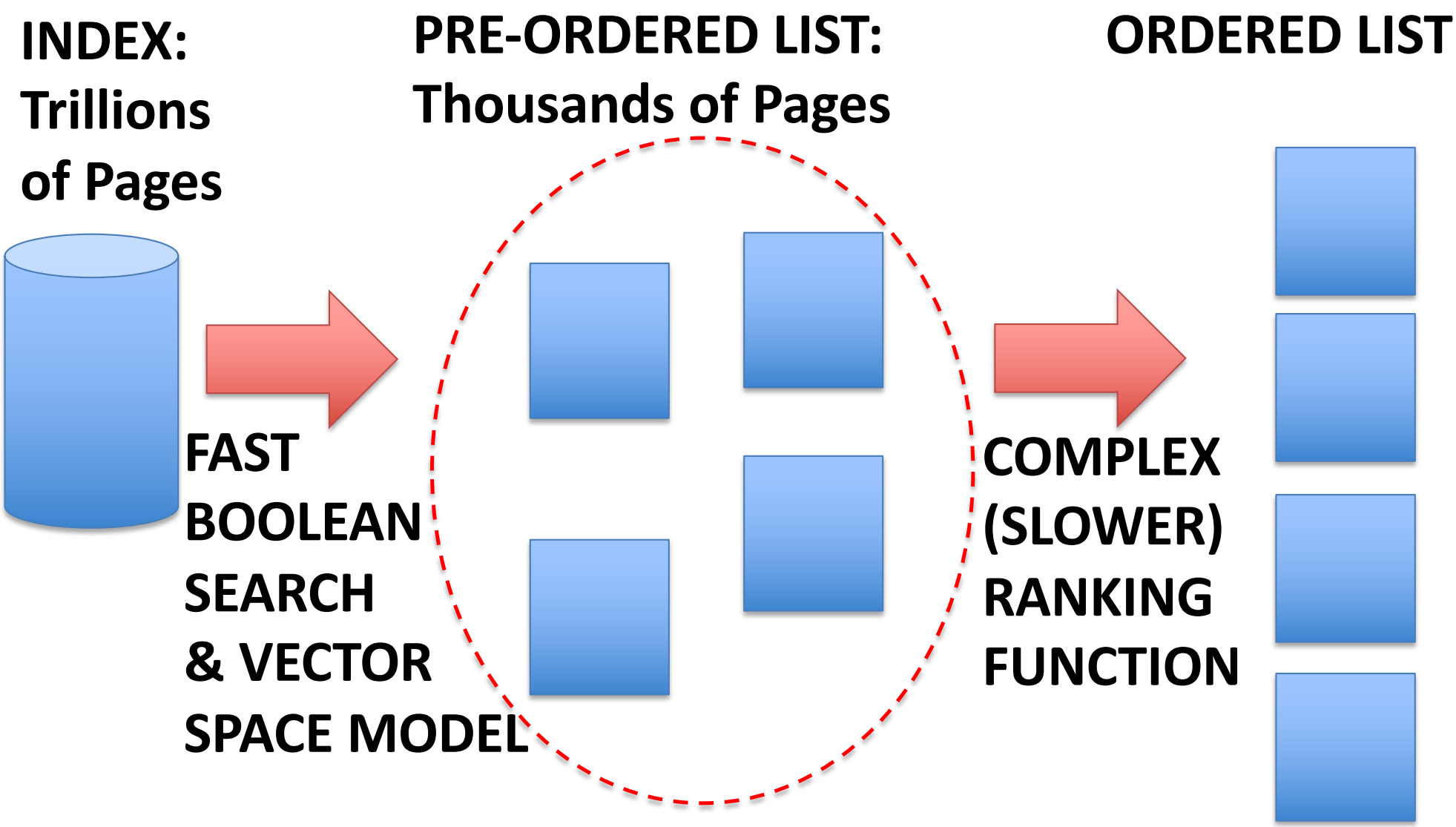
$k_1$: Saturation for tf




b: Document length bias



# Two-Stage Scoring



# Motivation of Two-Stage Scoring with ``Learning-to-Rank" methods

- Machine learning approach:
  - Enables more **features** (signal sources)
  - 1st stage aims for high Recall, 2nd stage aims for high Precision
- Useful features based on Query (q) and Document (d)
  - Various vector space model results, applied to text, URL, title
  - Click-through: e.g. How many times d is clicked given q vs. How many times d is skipped
  - Results after Query-expansion
- Useful features based on Document only (static)
  - Popularity of page: #Likes, #inlinks, Pagerank
  - Domain structure: main page or subpage

# Example features


Word match?

URL match?

Popular page?

Recently updated?

User intention match?

Not spam?

Clickthrough log?

**Johns Hopkins University Computer Science | Facebook**
https://**www.facebook.com**/compsci**jhu**
**Johns Hopkins University Computer Science**, Baltimore. 407 likes · 5 talking about this. **Computer Science** at **Johns Hopkins University** (**CS@JHU**) is a...

600.318/418: Operating Systems - **Johns Hopkins University**
srl.**cs.jhu**.edu/**courses**/600.418/index.html
**Computer science** majors and graduate students will be admitted regardless of enrollment limits. ... This **course** provides an introduction to operating systems.

Department of **Computer Science | Course Information**
www.**cs.jhu**.edu/**course**-info
**CS Course** Catalog – complete list of departmental **courses** with descriptions. Note that not all **courses** are offered every year. **Course** Area Designators – a chart ...

**Johns Hopkins University • Free Online Courses** and ...
www.class-central.com › Universities › **Johns Hopkins University**
Discover free online **courses** taught by **Johns Hopkins University**. Watch videos, do assignments, earn a certificate while learning from some of the best Professors.

# Problem Formulation


1. Feature extraction

<query, $doc_1$> → vector $x_1$

<query, $doc_2$> → vector $x_2$

<query, $doc_3$> → vector $x_3$

2. Apply ranking function & sort

$F(x_1) = 3$ → Rank 2

$F(x_2) = 1$ → Rank 3 (worst)

$F(x_3) = 4$ → Rank 1 (best)

What function class for F()?

Assume linear weights: $F(x_i) = w^T x_i$

Learn weights w that replicate ranking on training set

# Training Set

| Query 1 | Labels for each query-doc pair |
|---|---|
| <query, $doc_1$> → vector $x_1$ | label($x_1$) = 3 |
| <query, $doc_2$> → vector $x_2$ | label($x_2$) = 1 |
| <query, $doc_3$> → vector $x_3$ | label($x_3$) = 4 |

| Query 2 | Labels for each query-doc pair | |
|---|---|---|
| <query, $doc_1$> → vector $x_1$ | label($x_1$) = 3 | (Very relevant) |
| <query, $doc_2$> → vector $x_2$ | label($x_2$) = 2 | (Relevant) |
| <query, $doc_3$> → vector $x_3$ | label($x_3$) = 1 | (Slightly relevant) |
| <query, $doc_4$> → vector $x_4$ | label($x_4$) = 0 | (Irrelevant) |

# Where does the label come from?

- Human annotation
  - High quality, but expensive
- Click-through logs
  - Noisy, but cheap/abundant

# Notation

query: $q^{(n)} \quad n = 1, \ldots, N$

document for query n: $d_i^{(n)} \quad i = 1, \ldots, I_n$

vector of $D$ features per query-doc: $x_i^{(n)} \in \mathcal{R}^D$

label for each query-doc pair: $l_i^{(n)} \in \mathcal{Z}$

Training set: $\{q^{(n)}, \{x_i^{(n)}, l_i^{(n)}\}\}$

Ranking Function: $F(x_i^{(n)}) = w^T x_i^{(n)}$

# Different training approaches

- How to optimize something on a set with a sort operation? Reduce to traditional regression/classification problems

| Training Approach | Reduction |
|---|---|
| Point-wise | Document |
| Pair-wise | Two Documents |
| List-wise | All Documents per query |

# Point-wise Approach

Training set: $\{q^{(n)}, \{x_i^{(n)}, l_i^{(n)}\}\}$

Ranking Function: $F(x_i^{(n)}) = w^T x_i^{(n)}$

Find w that makes each F(x) equal to its label

Training Objective: $\sum_n \sum_i (F(x_i^{(n)}) - l_i^{(n)})^2$

Training Objective: $\sum_n \sum_i (F(x_i^{(n)}) - l_i^{(n)})^2$

$\rightarrow \sum_z (F(x_z) - l_z)^2$ where z ranges over all i, n

Solve with linear regression!

# Pair-wise Approach

Training set: $\{q^{(n)}, \{x_i^{(n)}, l_i^{(n)}\}\}$

Ranking Function: $F(x_i^{(n)}) = w^T x_i^{(n)}$

Find w that gives every pair the correct ranking

Training Objective:

$$F(x_i^{(n)}) > F(x_j^{(n)}) \quad \forall \quad i, j \quad \text{s.t.} \quad l_i^{(n)} > l_j^{(n)}$$

Training Objective:

$$F(x_i^{(n)}) > F(x_j^{(n)}) \quad \forall \quad i,j \quad \text{s.t.} \quad l_i^{(n)} > l_j^{(n)}$$

$$\rightarrow F(x_i^{(n)}) - F(x_j^{(n)}) > 0 \quad \forall \quad i,j \quad \text{s.t.} \quad l_i^{(n)} > l_j^{(n)}$$

$$\rightarrow w^T x_i^{(n)} - w^T x_j^{(n)} > 0 \quad \forall \quad i,j \quad \text{s.t.} \quad l_i^{(n)} > l_j^{(n)}$$

$$\rightarrow w^T (x_i^{(n)} - x_j^{(n)}) > 0 \quad \forall \quad i,j \quad \text{s.t.} \quad l_i^{(n)} > l_j^{(n)}$$

$$\rightarrow w^T (\delta_{ij}^{(n)}) > 0 \quad \forall \quad i,j \quad \text{s.t.} \quad l_i^{(n)} > l_j^{(n)}$$

Solve with binary classification!

Make a new sample out of every pair

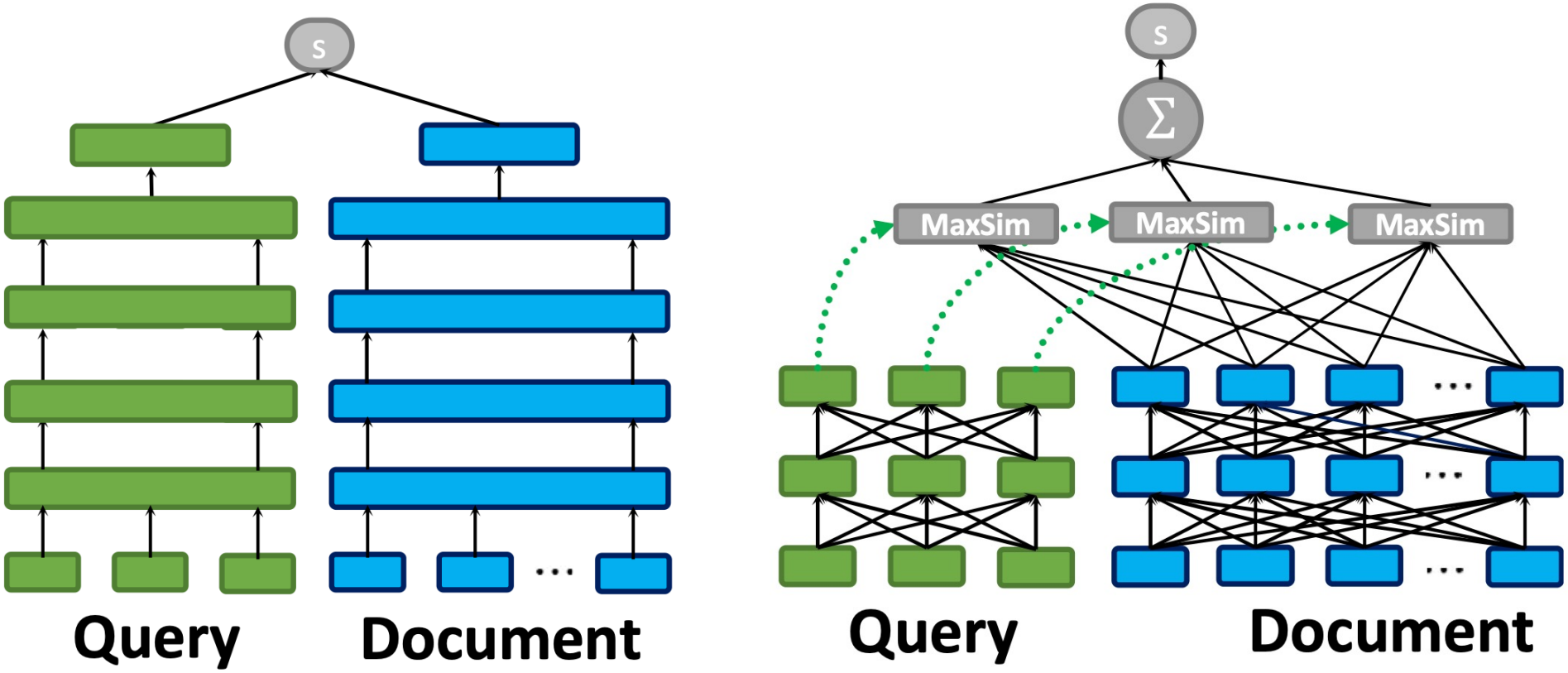
Give new label: Positive for i,j pairs

Negative for j,i pairs

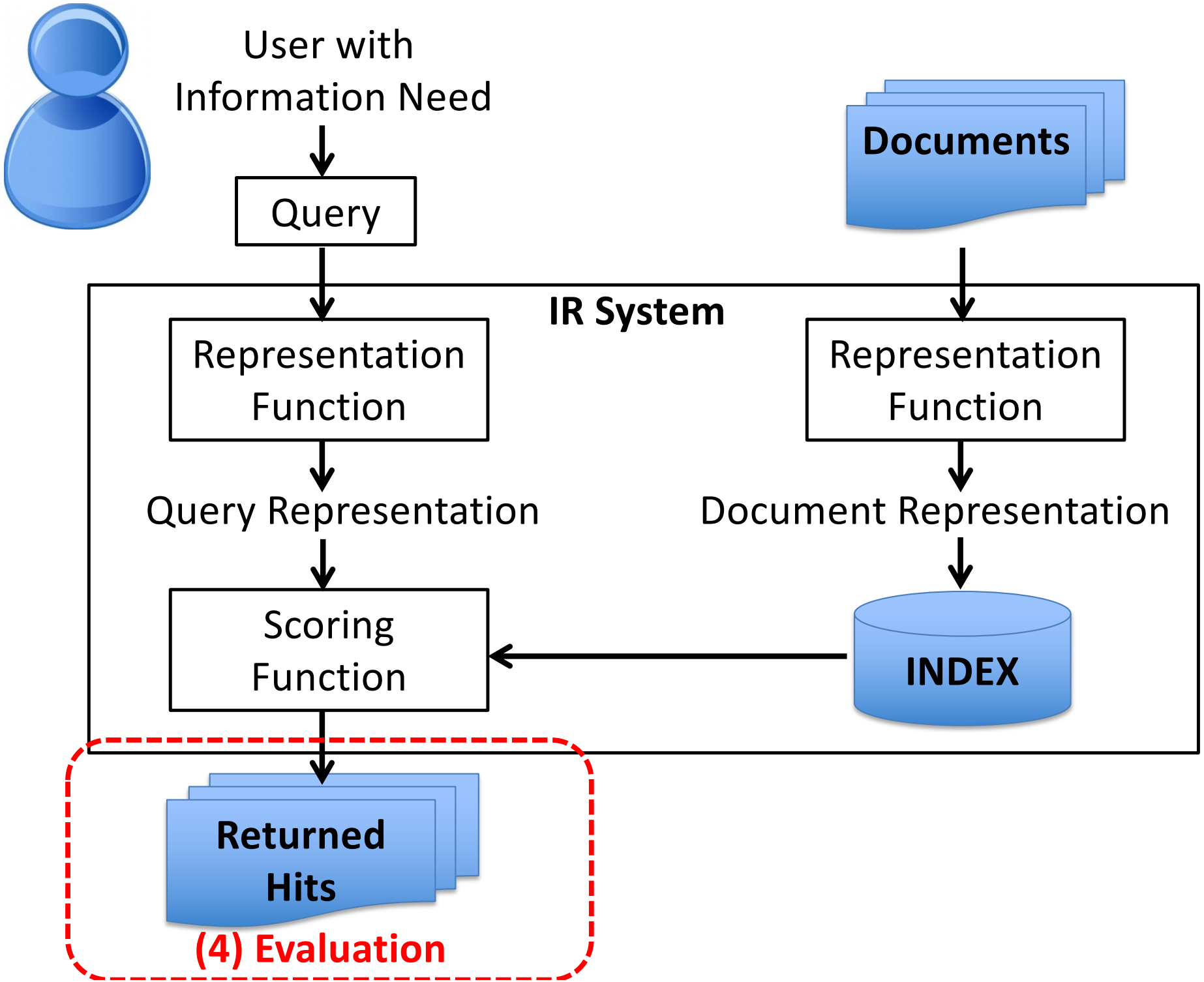
# Disclaimer

- We've focused on very simple ranking functions (linear) for simplicity
- In practice, more complex functions (e.g. decision trees, neural nets) are common
- Some functions use "dense" word embeddings as opposed to "sparse" features described previously
- Recommend further reading:
  - Dawei Yin, et. al. "Ranking Relevance in Yahoo Search", Proceedings of KDD2016
  - Omar Khattab & Matei Zaharia. "ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT", SIGIR 2020

# Embeddings & Neural Nets for Scoring



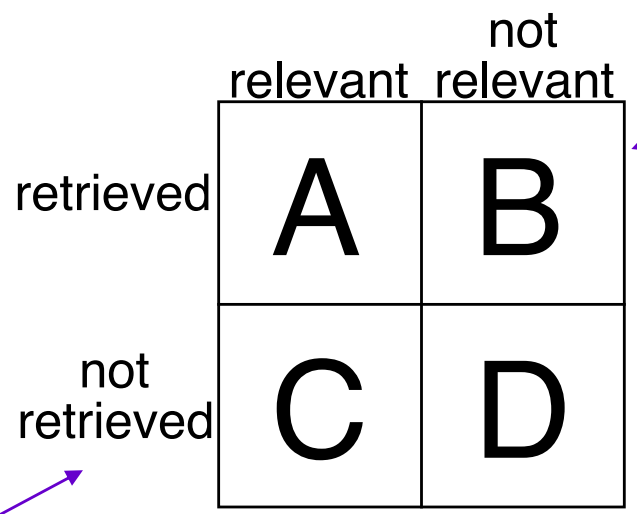

From: Khattab (SIGIR2020) https://arxiv.org/pdf/2004.12832.pdf


User with
Information Need
Query
Documents
IR System
Representation
Function
Representation
Function
Query Representation
Document Representation
Scoring
Function
INDEX
Returned
Hits
(4) Evaluation

# Evaluation: How good/bad is my IR?

- Evaluation is important:
  - Compare two IR systems
  - Decide whether our IR is ready for deployment
  - Identify research challenges
- Two Ingredients for a trustworthy evaluation:
  - Answer Key
  - A Meaningful Metric: given query q, returned ranked list, and answer key, computes a number

# Precision and Recall

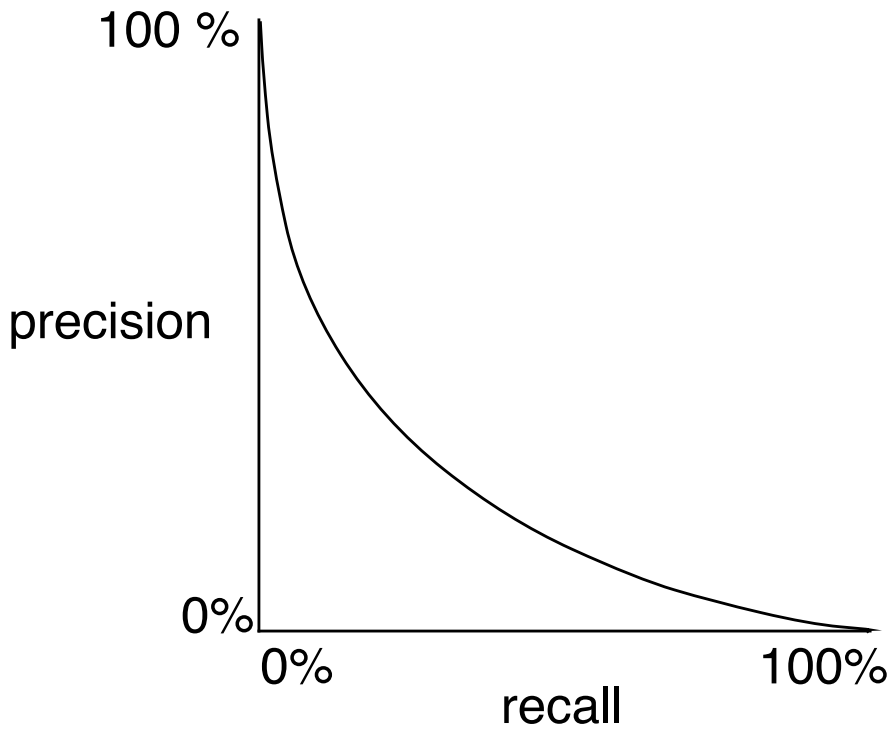



"Type one errors" "Errors of commission" "False positives"

"Type two errors"
"Errors of omission"
"False negatives"

$$\text{precision} = \frac{A}{A + B}$$

$$\text{recall} = \frac{A}{A + C}$$




average precision = area under curve

From Paul McNamee's JSALT 2018 tutorial slides

# Issues with Precision and Recall

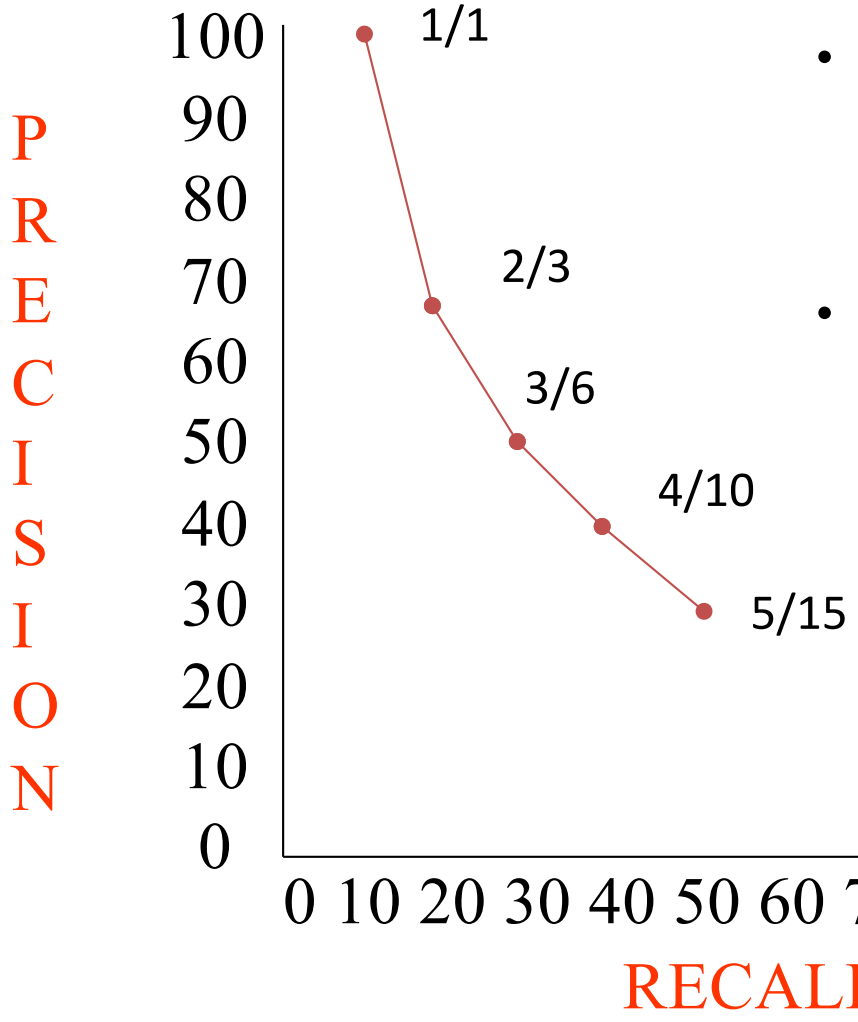
- We often don't know true recall value
  - For large collection, impossible to have annotator read all documents to assess relevance of a query
- Focused on evaluating sets, rather than ranked lists

We'll introduce Mean Average Precision (MAP) here. Note that IR evaluation is a deep field, worth another lecture by itself!

# Example for 1 query: precision & recall at different positions in ranked list

10 relevant: $R_q=\{d_3,d_5,d_9,d_{25},d_{39},d_{44},d_{56},d_{71},d_{89},d_{123}\}$

Ranked List: $d_{123}$, $d_{84}$, $d_{56}$, $d_6$, $d_8$, $d_9$, $d_{511}$, $d_{129}$, $d_{187}$, $d_{25}$, $d_{38m}$, $d_{48}$, $d_{250}$, $d_{113}$, $d_3$

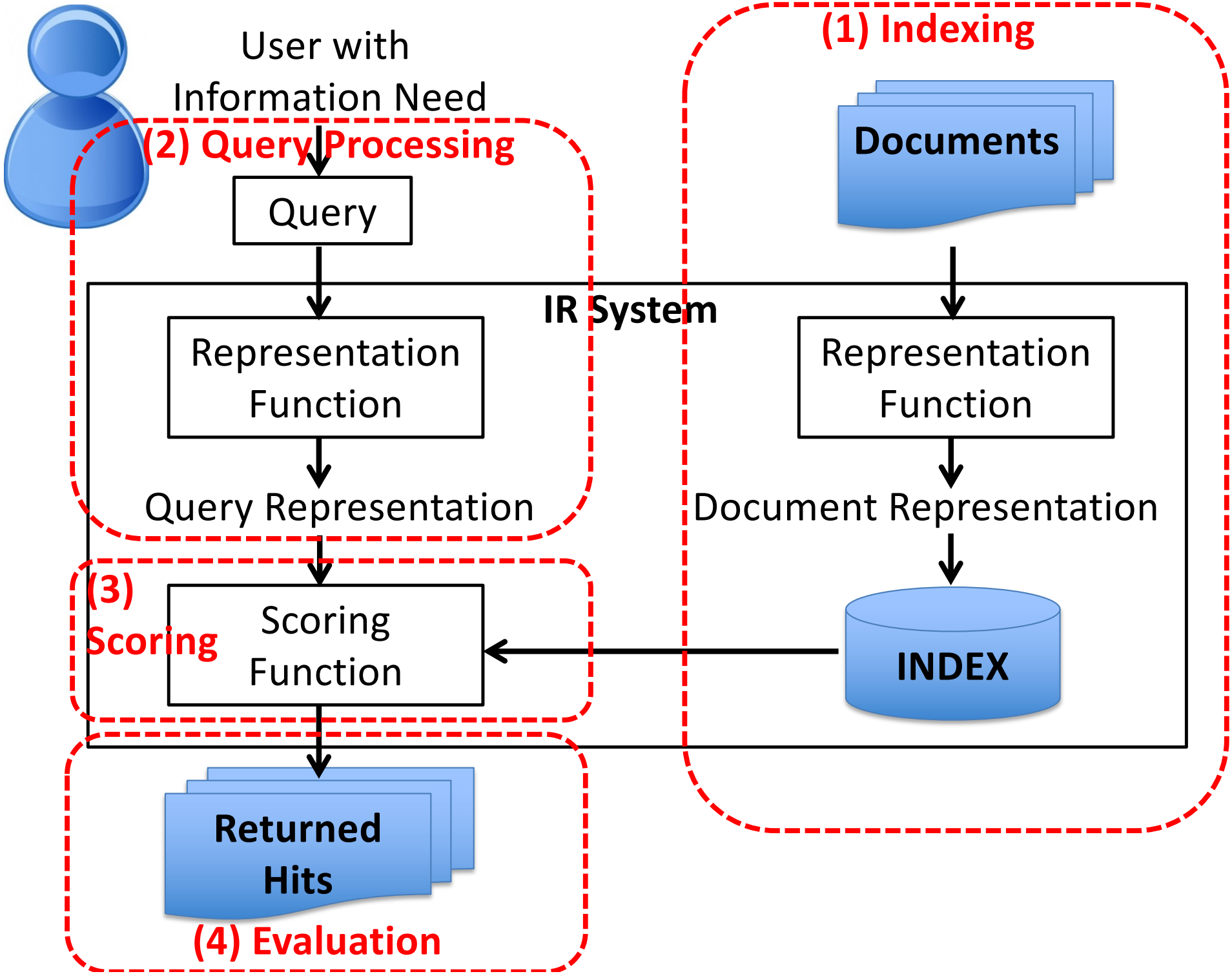



- First ranked doc $d_{123}$ is relevant, which is 10% of the total relevant. Therefore Precision at the 1/10=10% Recall level is 1/1=100%
- Next Relevant $d_{56}$ gives us 2/3=66% Precision at 2/10=20% recall level

**Average Precision (AP):**
(1/1 + 2/3 + 3/6 + 4/10 + 5/15) / 5 = 0.58

**Mean Average Precision (MAP):**
Mean of AP over multiple queries

From Paul McNamee's JSALT 2018 tutorial slides


User with
Information Need
(1) Indexing
Documents
(2) Query Processing
Query
IR System
Representation
Function
Representation
Function
Query Representation
Document Representation
(3)
Scoring
Scoring
Function
INDEX
Returned
Hits
(4) Evaluation