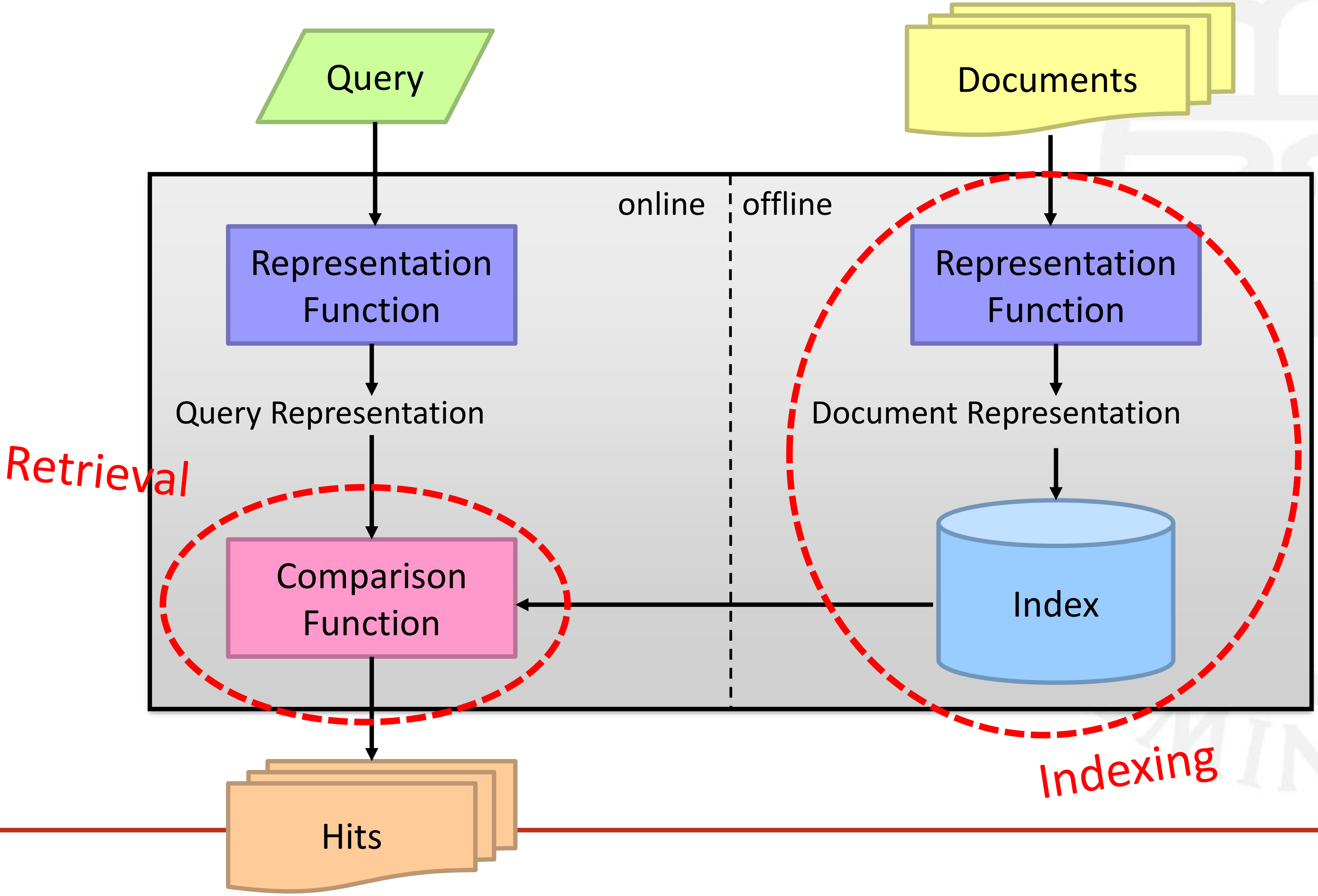
NWI-I00041
Information Retrieval – Lecture 4

Dr. ir. Faegheh Hasibi
f.hasibi@cs.ru.nl

Prof.dr.ir. Arjen P. de Vries
arjen@acm.org

Nijmegen, September 21st, 2020

Abstract IR Architecture



Inverted Index

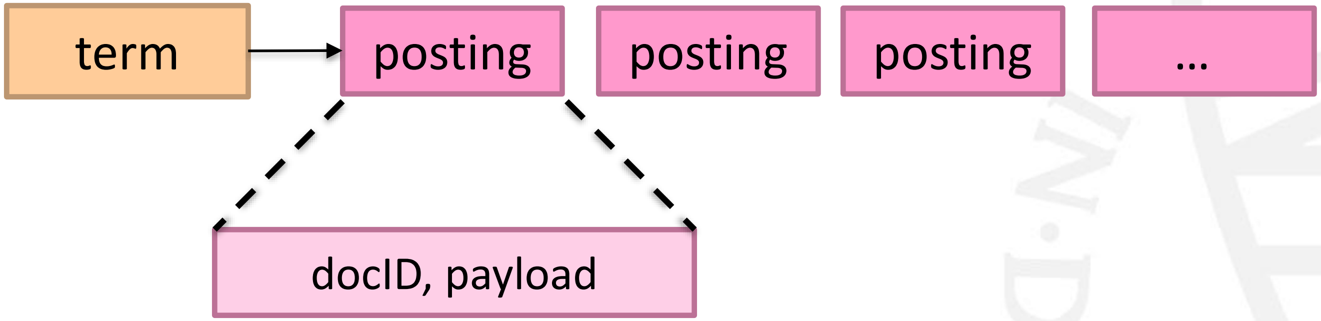
- Indexes are data structures designed to make search faster

- The inverted index is an efficient & flexible data structure

- Default technique to rank documents efficiently on their similarity to the query

- Find all exact matches efficiently
  - Assumes the query is “correct”

Inverted Index



Payload (optional): other associated information such as count and position

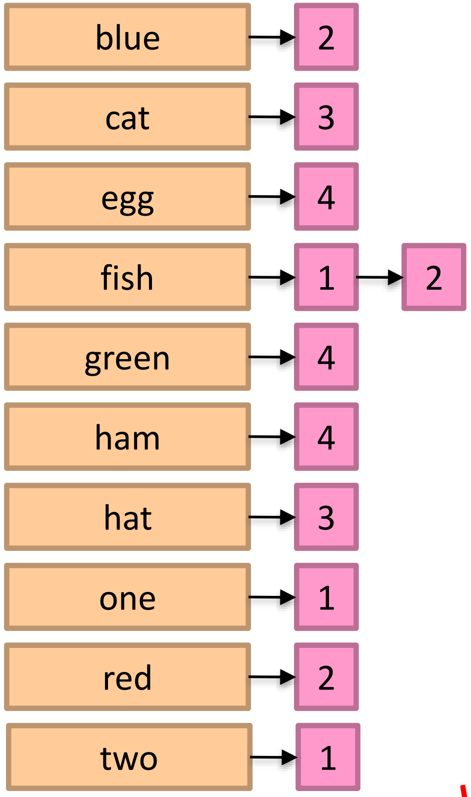
Doc 1
one fish, two fish

Doc 2
red fish, blue fish

Doc 3
cat in the hat

Doc 4
green eggs and ham

	1	2	3	4
blue		1		
cat			1	
egg				1
fish	1	1		
green				1
ham				1
hat			1	
one	1			
red		1		
two	1			



blue → 2
cat → 3
egg → 4
fish → 1 → 2
green → 4
ham → 4
hat → 3
one → 1
red → 2
two → 1

postings lists
(often in sorted order)

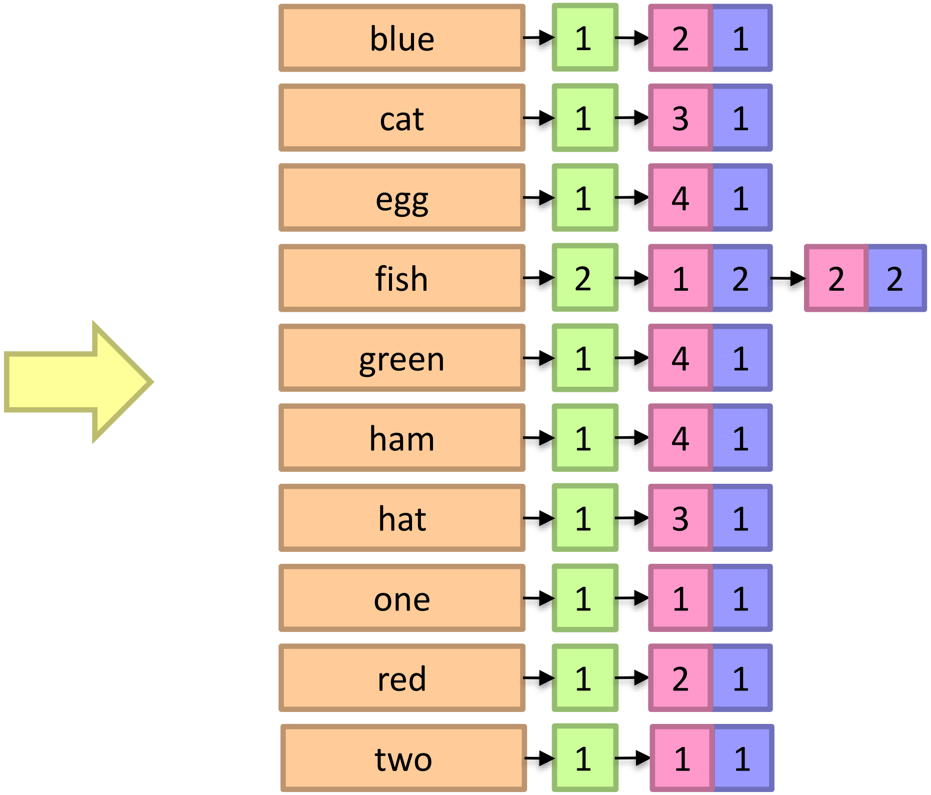
Doc 1
one fish, two fish

Doc 2
red fish, blue fish

Doc 3
cat in the hat

Doc 4
green eggs and ham

	tf				
	1	2	3	4	*df*
blue		1			1
cat			1		1
egg				1	1
fish	2	2			2
green				1	1
ham				1	1
hat			1		1
one	1				1
red		1			1
two	1				1



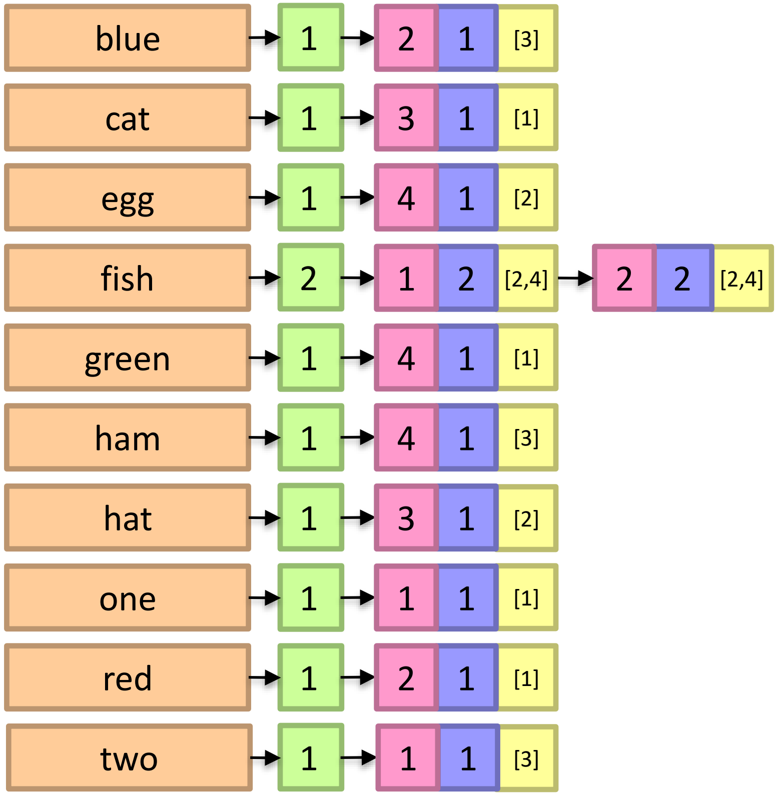
Doc 1
one fish, two fish

Doc 2
red fish, blue fish

Doc 3
cat in the hat

Doc 4
green eggs and ham

	tf				
	1	2	3	4	*df*
blue		1			1
cat			1		1
egg				1	1
fish	2	2			2
green				1	1
ham				1	1
hat			1		1
one	1				1
red		1			1
two	1				1



blue	1	2	1	[3]			
cat	1	3	1	[1]			
egg	1	4	1	[2]			
fish	2	1	2	[2,4]	2	2	[2,4]
green	1	4	1	[1]			
ham	1	4	1	[3]			
hat	1	3	1	[2]			
one	1	1	1	[1]			
red	1	2	1	[1]			
two	1	1	1	[3]			

Compression

- Inverted lists are very large, stored on disk

- Compression of the postings lists reduces the bandwidth needed to read from disk

- Compression algorithms optimized for high decompression speed (~3 GB/s and up; PFORDelta invented at CWI Amsterdam is widely used)

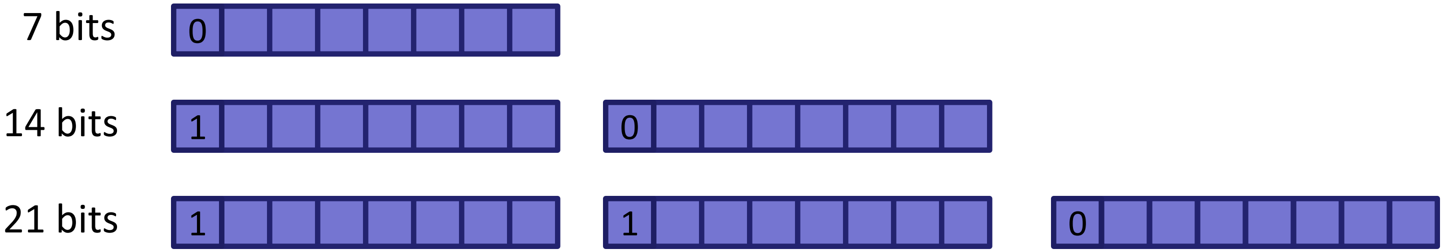
Compression Methods

- Bit-aligned
  - Unary codes
  - γ/δ codes
  - Golomb codes (local Bernoulli model)

- Byte-aligned technique
  - Vbyte

VByte

- Simple idea: use only as many bytes as needed
  - Need to reserve one bit per byte as the “continuation bit”
  - Use remaining bits for encoding value



7 bits: 0

14 bits: 1, 0

21 bits: 1, 1, 0

What to compress?!

- Gap-encoding: compress the numbers that result from the difference between consecutive document IDs
  - Frequent terms: small numbers – better compression as a result
  - Infrequent terms: still large numbers, likely unique; but very few relatively, given Zipf’s law

Doc 1
one fish, two fish

Doc 2
red fish, blue fish

Doc 3
cat in the hat

Doc 4
green eggs and ham

	1	2	3	4
blue		1		
cat			1	
egg				1
fish	1	1		
green				1
ham				1
hat			1	
one	1			
red		1		
two	1			

Indexing: building this structure

Retrieval: manipulating this structure

Retrieval in a Nutshell

- Look up postings lists corresponding to query terms
- Traverse postings for each query term
- Store partial query-document scores in accumulators
- Select top *k* results to return

Retrieval and Query Processing

- Strategies for processing the data in the index for producing query results

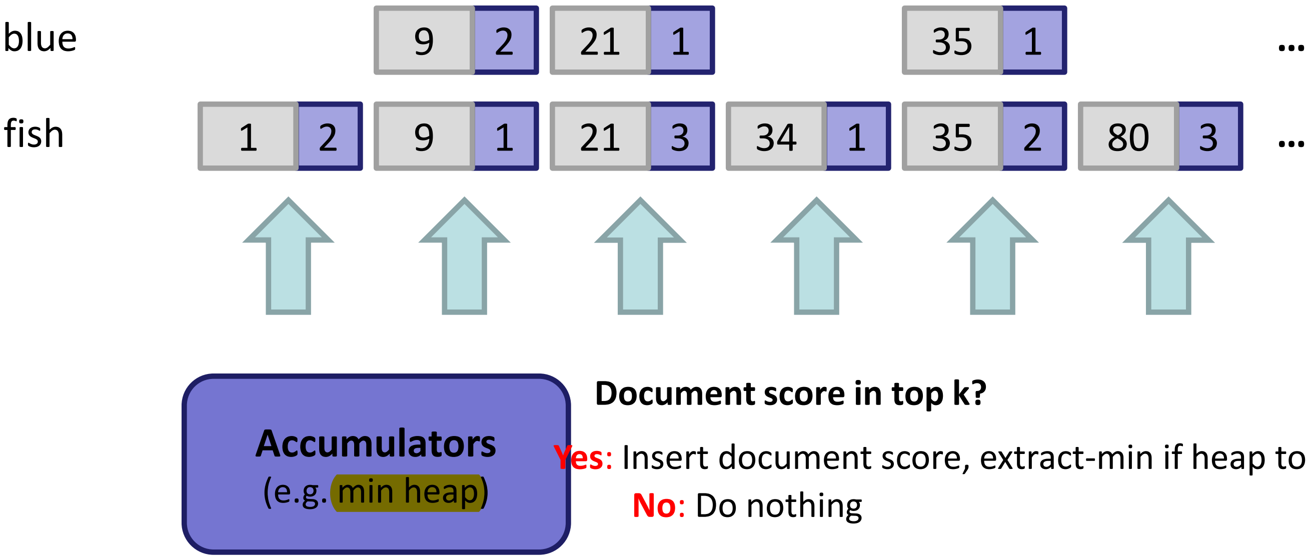
- Document-at-a-time
  - Calculates complete scores for documents by processing all term lists, one document at a time

- Term-at-a-time
  - Accumulates scores for documents by processing term lists one at a time

Both approaches have optimization techniques that significantly reduce time required to generate scores

Document-at-a-Time

- Evaluate documents one at a time (score all query terms)



blue		9 2	21 1		35 1		…
fish	1 2	9 1	21 3	34 1	35 2	80 3	…

Document score in top k?

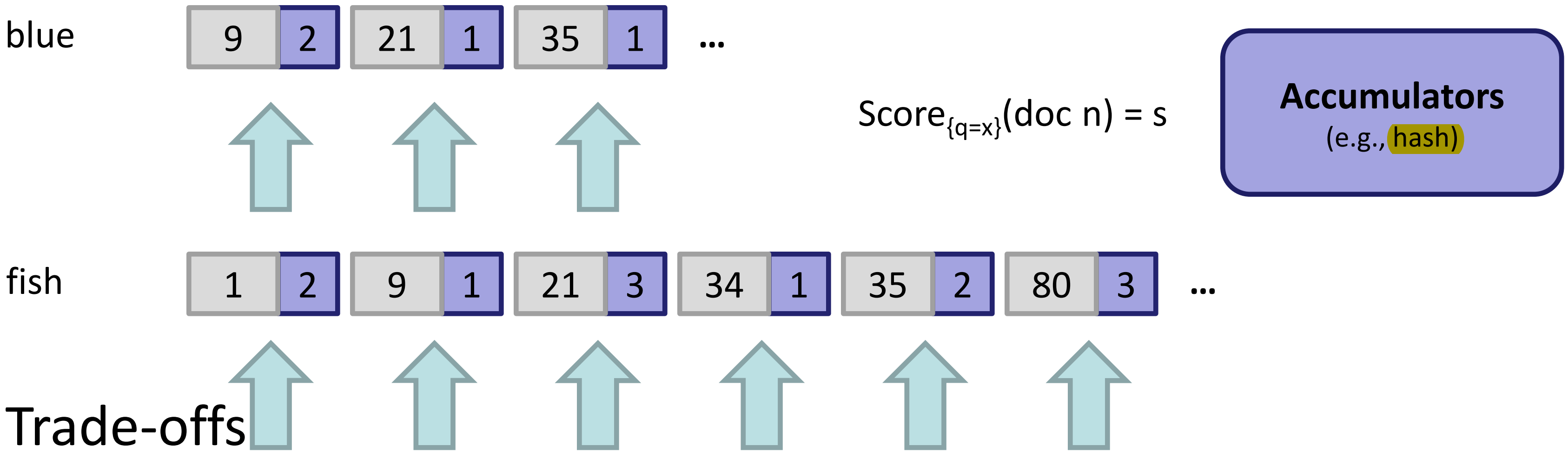
Yes: Insert document score, extract-min if heap too large

No: Do nothing

- Trade-offs
  - Small memory footprint (good)
  - Skipping possible to avoid reading all postings (good)
  -

Term-At-A-Time

- Evaluate documents one query term at a time
  - Usually, starting from most rare



blue 9 2 | 21 1 | 35 1 …

fish 1 2 | 9 1 | 21 3 | 34 1 | 35 2 | 80 3 …

$\text{Score}_{\{q=x\}}(\text{doc } n) = s$

Accumulators
(e.g., hash)

- Trade-offs
  - Early termination heuristics (good)
  - Large memory footprint (bad), but filtering heuristics possible