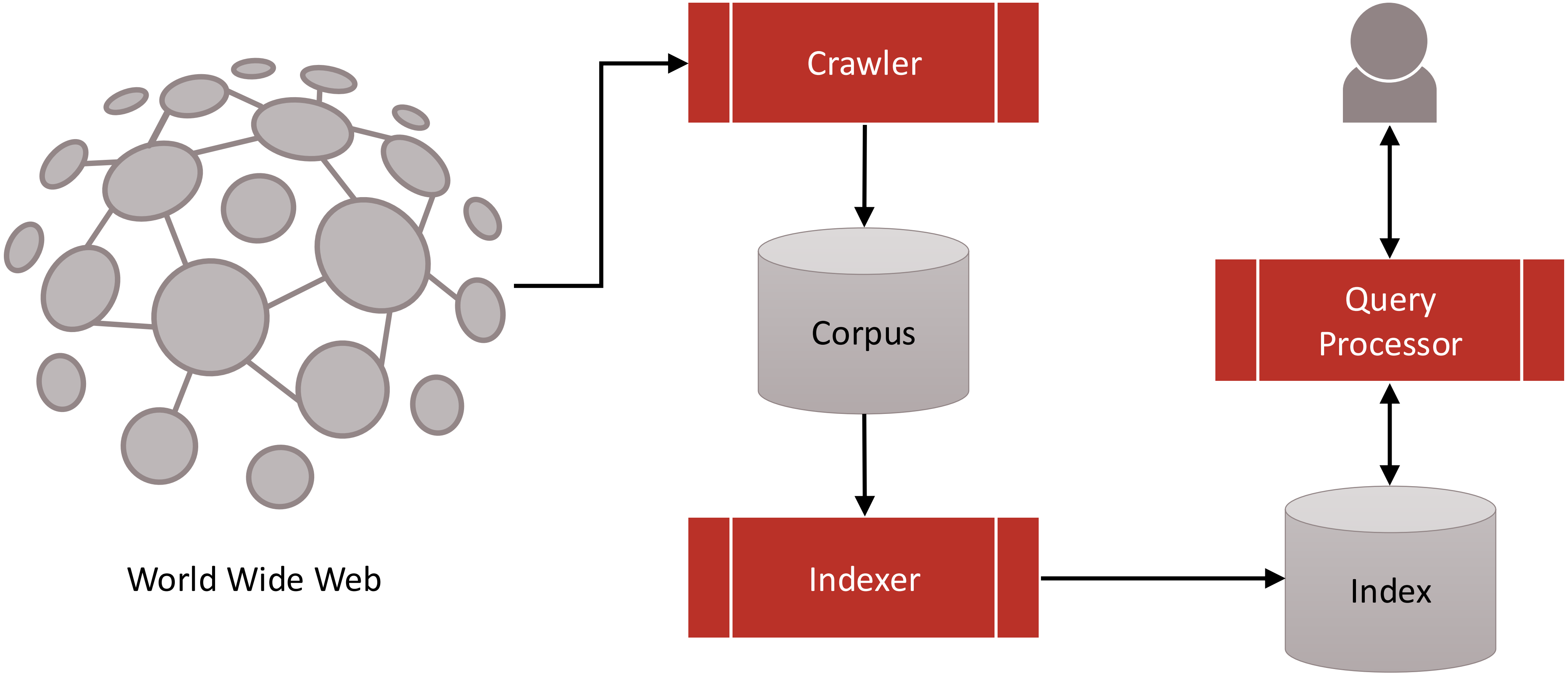
UFMG
UNIVERSIDADE FEDERAL
DE MINAS GERAIS

Information Retrieval

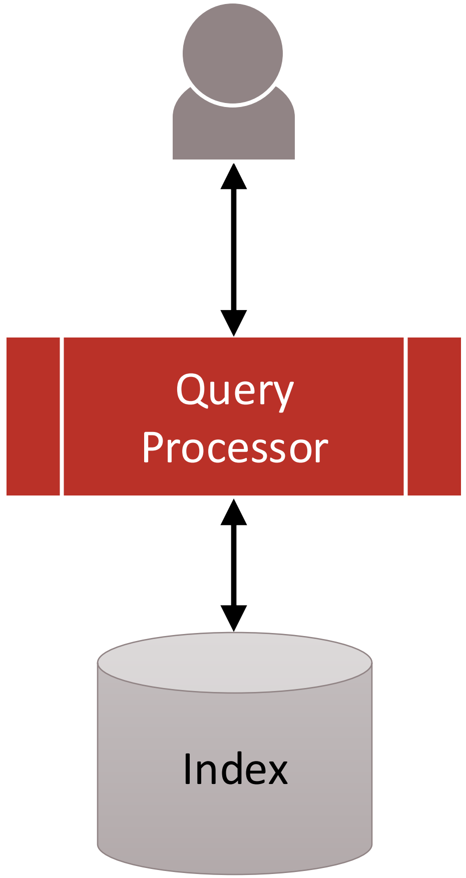
Efficient Matching

Rodrygo L. T. Santos
rodrygo@dcc.ufmg.br

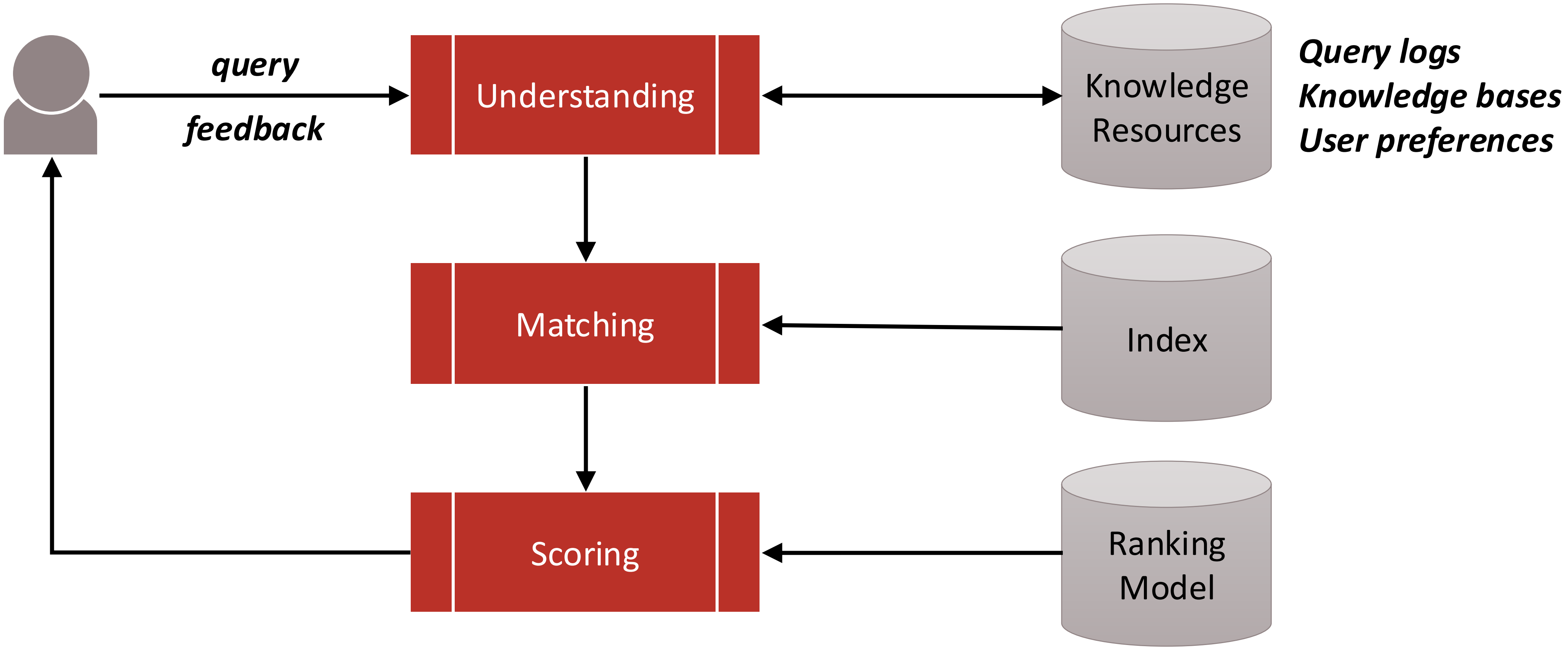
Search components



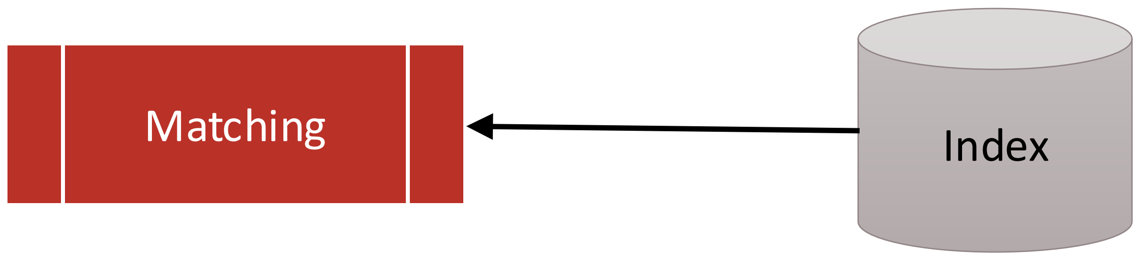
Search components



Query processing overview



Query processing overview



Document matching

Scan postings lists for all query terms

- [aquarium fish]

and	1:1			
aquarium	3:1			
are	3:1	4:1		
…				
environments	1:1			
fish	1:2	2:3	3:2	4:2

Document matching

Scan postings lists for all query terms

- [aquarium fish]

aquarium	3:1			
fish	1:2	2:3	3:2	4:2

index access cost

- memory paging (I/O)
- in-memory processing (CPU)

Score matching documents

- $f(q,d) = \sum_{t \in q} f(t,d)$

scoring cost

- decompression + scoring (CPU)

Index access cost

Inherent cost of matching documents to queries

- Query length (number of posting lists)
- Posting lists length (number of postings per list)

Traversal direction

TAAT: inverted lists processed in sequence

- More memory efficient (sequential access)

DAAT: inverted lists processed in parallel

- Uses less memory (no accumulators)
- Handles complex queries (Boolean, proximity)
- De facto choice for modern search engines

Naïve DAAT

Inverted lists processed in parallel

- One document scored at a time

A	1:3	2:3	3:4	8:4		
B	1:4	5:2	7:2	8:5	9:2	11:5
C	1:6	2:5	5:3	6:7	10:1	11:7

Naïve DAAT

Inverted lists processed in parallel

- One document scored at a time

A	1:3	2:3	3:4	8:4		
B	1:4	5:2	7:2	8:5	9:2	11:5
C	1:6	2:5	5:3	6:7	10:1	11:7
scores	1:13					

Naïve DAAT

Inverted lists processed in parallel

- One document scored at a time

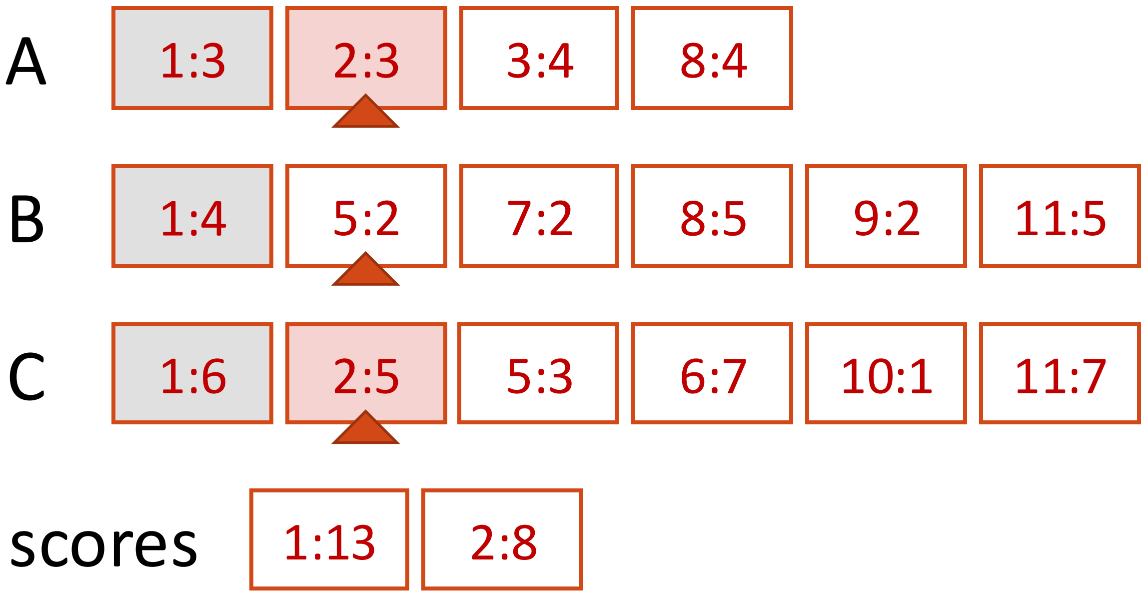
A	1:3	2:3	3:4	8:4		
B	1:4	5:2	7:2	8:5	9:2	11:5
C	1:6	2:5	5:3	6:7	10:1	11:7

scores	1:13

Naïve DAAT

Inverted lists processed in parallel

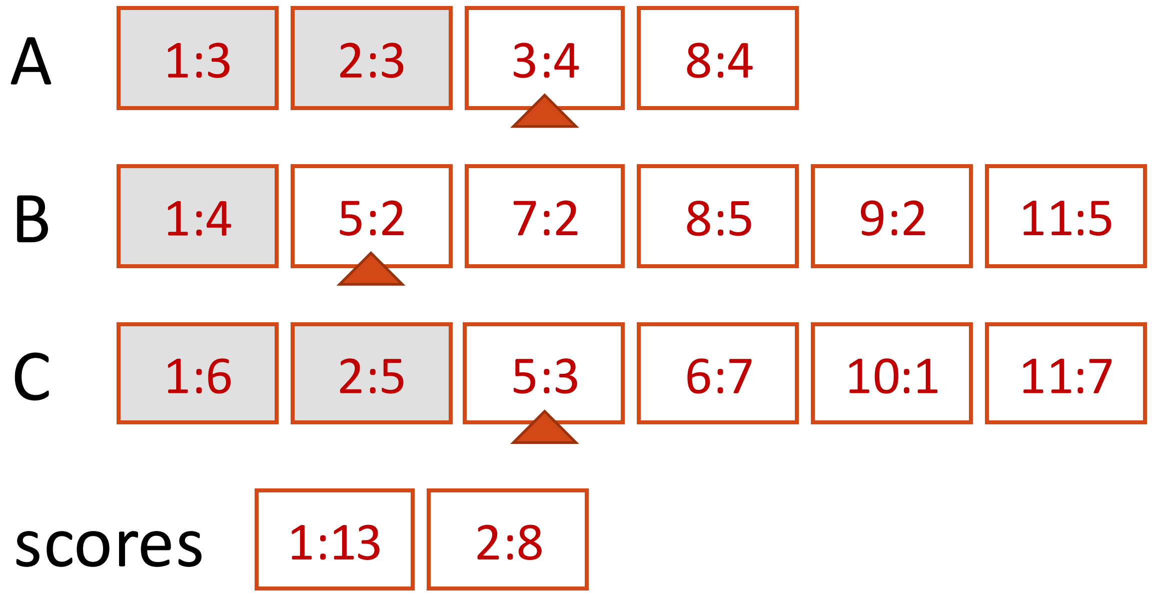
- One document scored at a time



Naïve DAAT

Inverted lists processed in parallel

- One document scored at a time



Naïve DAAT

Inverted lists processed in parallel

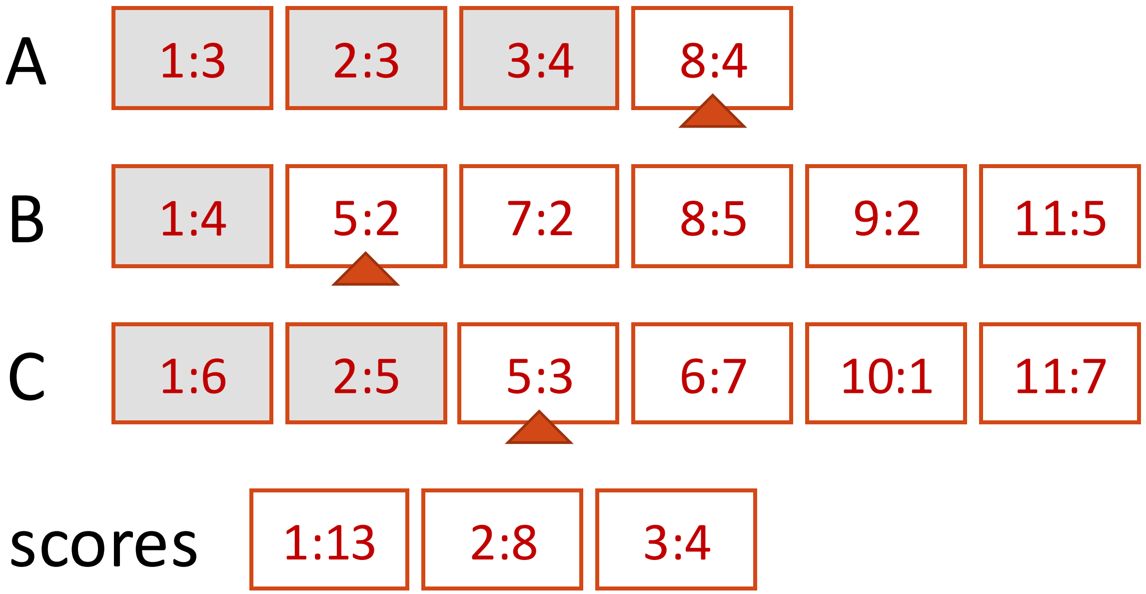
- One document scored at a time

A	1:3	2:3	3:4	8:4		
B	1:4	5:2	7:2	8:5	9:2	11:5
C	1:6	2:5	5:3	6:7	10:1	11:7
scores	1:13	2:8	3:4			

Naïve DAAT

Inverted lists processed in parallel

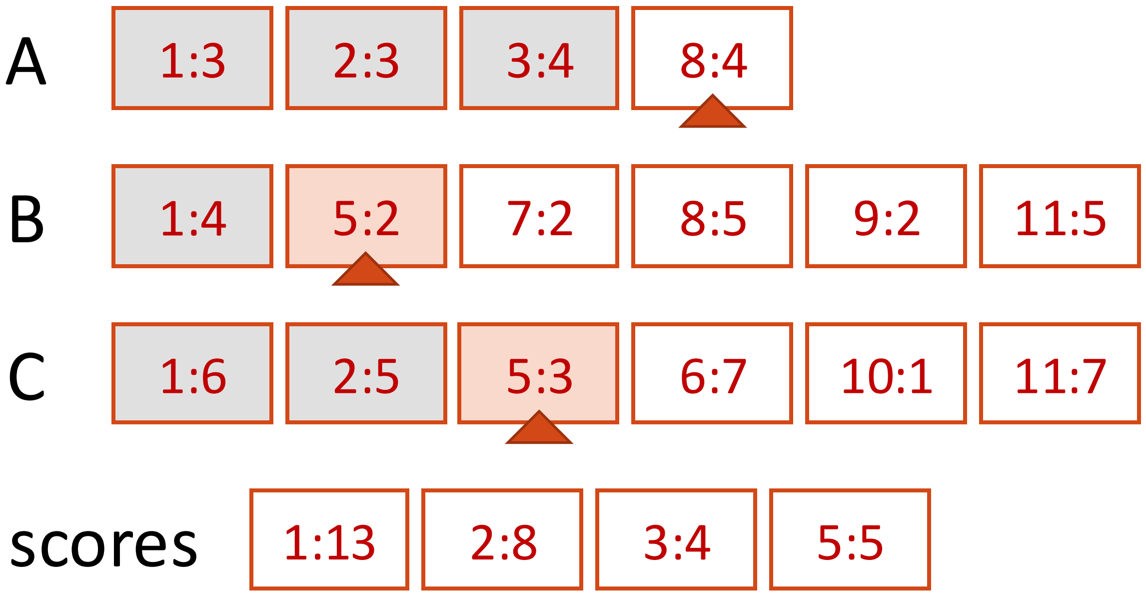
- One document scored at a time



Naïve DAAT

Inverted lists processed in parallel

- One document scored at a time



Naïve DAAT

Inverted lists processed in parallel

- One document scored at a time

A	1:3	2:3	3:4	8:4		
B	1:4	5:2	7:2	8:5	9:2	11:5
C	1:6	2:5	5:3	6:7	10:1	11:7
scores	1:13	2:8	3:4	5:5		

Naïve DAAT

Inverted lists processed in parallel

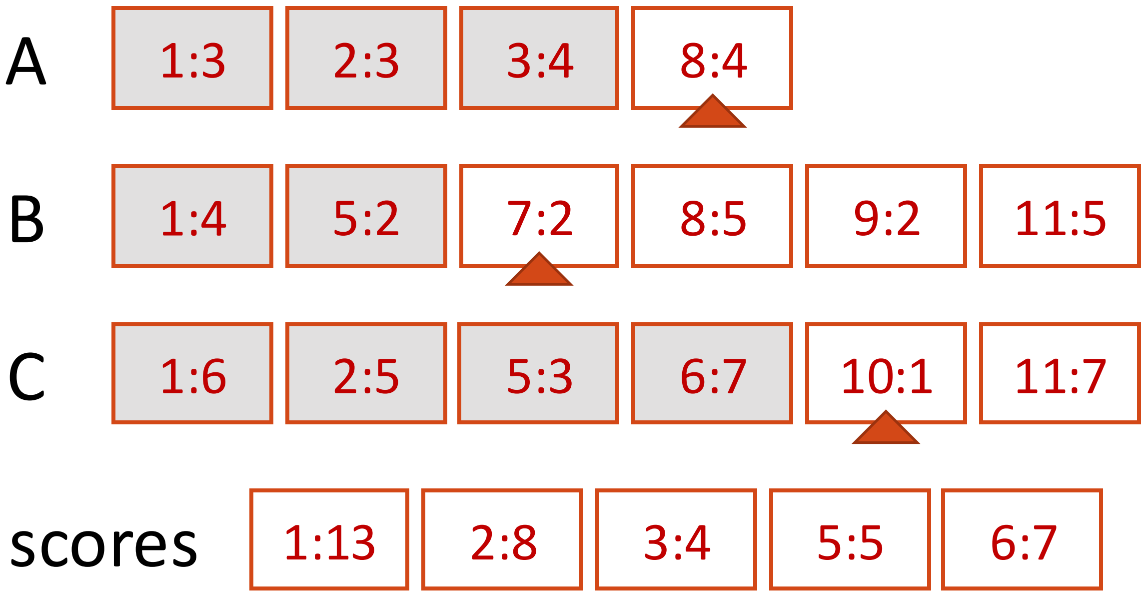
- One document scored at a time

A	1:3	2:3	3:4	8:4		
B	1:4	5:2	7:2	8:5	9:2	11:5
C	1:6	2:5	5:3	6:7	10:1	11:7
scores	1:13	2:8	3:4	5:5	6:7	

Naïve DAAT

Inverted lists processed in parallel

- One document scored at a time



Naïve DAAT

Inverted lists processed in parallel

- One document scored at a time

Naïve DAAT

Inverted lists processed in parallel

- One document scored at a time

A	1:3	2:3	3:4	8:4		
B	1:4	5:2	7:2	8:5	9:2	11:5
C	1:6	2:5	5:3	6:7	10:1	11:7
scores	1:13	2:8	3:4	5:5	6:7	7:2

Naïve DAAT

Inverted lists processed in parallel

- One document scored at a time

A	1:3	2:3	3:4	8:4			
B	1:4	5:2	7:2	8:5	9:2	11:5	
C	1:6	2:5	5:3	6:7	10:1	11:7	
scores	1:13	2:8	3:4	5:5	6:7	7:2	8:9

Naïve DAAT

Inverted lists processed in parallel

- One document scored at a time

A	1:3	2:3	3:4	8:4			
B	1:4	5:2	7:2	8:5	9:2	11:5	
C	1:6	2:5	5:3	6:7	10:1	11:7	
scores	1:13	2:8	3:4	5:5	6:7	7:2	8:9

Naïve DAAT

Inverted lists processed in parallel

- One document scored at a time

A	1:3	2:3	3:4	8:4				
B	1:4	5:2	7:2	8:5	9:2	11:5		
C	1:6	2:5	5:3	6:7	10:1	11:7		
scores	1:13	2:8	3:4	5:5	6:7	7:2	8:9	9:2

Naïve DAAT

Inverted lists processed in parallel

- One document scored at a time

A	1:3	2:3	3:4	8:4				
B	1:4	5:2	7:2	8:5	9:2	11:5		
C	1:6	2:5	5:3	6:7	10:1	11:7		
scores	1:13	2:8	3:4	5:5	6:7	7:2	8:9	9:2

Naïve DAAT

Inverted lists processed in parallel

- One document scored at a time

A	1:3	2:3	3:4	8:4					
B	1:4	5:2	7:2	8:5	9:2	11:5			
C	1:6	2:5	5:3	6:7	10:1	11:7			
scores	1:13	2:8	3:4	5:5	6:7	7:2	8:9	9:2	10:1

Naïve DAAT

Inverted lists processed in parallel

- One document scored at a time

A	1:3	2:3	3:4	8:4					
B	1:4	5:2	7:2	8:5	9:2	11:5			
C	1:6	2:5	5:3	6:7	10:1	11:7			
scores	1:13	2:8	3:4	5:5	6:7	7:2	8:9	9:2	10:1

Naïve DAAT

Inverted lists processed in parallel

- One document scored at a time

A	1:3	2:3	3:4	8:4						
B	1:4	5:2	7:2	8:5	9:2	11:5				
C	1:6	2:5	5:3	6:7	10:1	11:7				
scores	1:13	2:8	3:4	5:5	6:7	7:2	8:9	9:2	10:1	11:12

Naïve DAAT

Inverted lists processed in parallel

- One document scored at a time

A	1:3	2:3	3:4	8:4						
B	1:4	5:2	7:2	8:5	9:2	11:5				
C	1:6	2:5	5:3	6:7	10:1	11:7				
scores	1:13	2:8	3:4	5:5	6:7	7:2	8:9	9:2	10:1	11:12

- 0 / 16 postings skipped
  ≈ 0% savings

What if we only want the top k results?

Dynamic pruning

Dynamic pruning strategies aim to make scoring faster by only scoring a subset of the documents

- Assume user is only interested in the top k results
- Check if a document can make it to the top k
- Early terminate (or even skip) unviable documents

Effectiveness guarantees

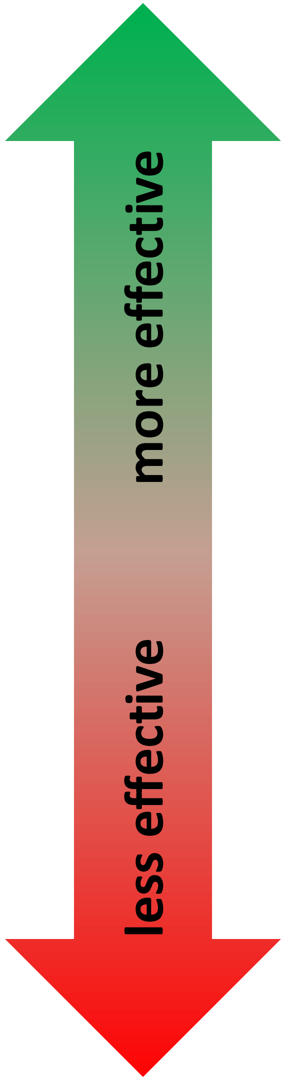
Safe: exhaustive (i.e. no pruning) matching

<u>**Score safe:** top k with correct scores</u>

<u>**Rank safe:** top k with correct order</u>

Set safe: top k with correct documents

Unsafe: no correctness guarantees whatsoever



MaxScore [Turtle and Flood, IPM 1995]

In a multi-term query, not all terms are worth the same

- Some will be “essential” for scoring documents
- Others will be “non-essential” terms

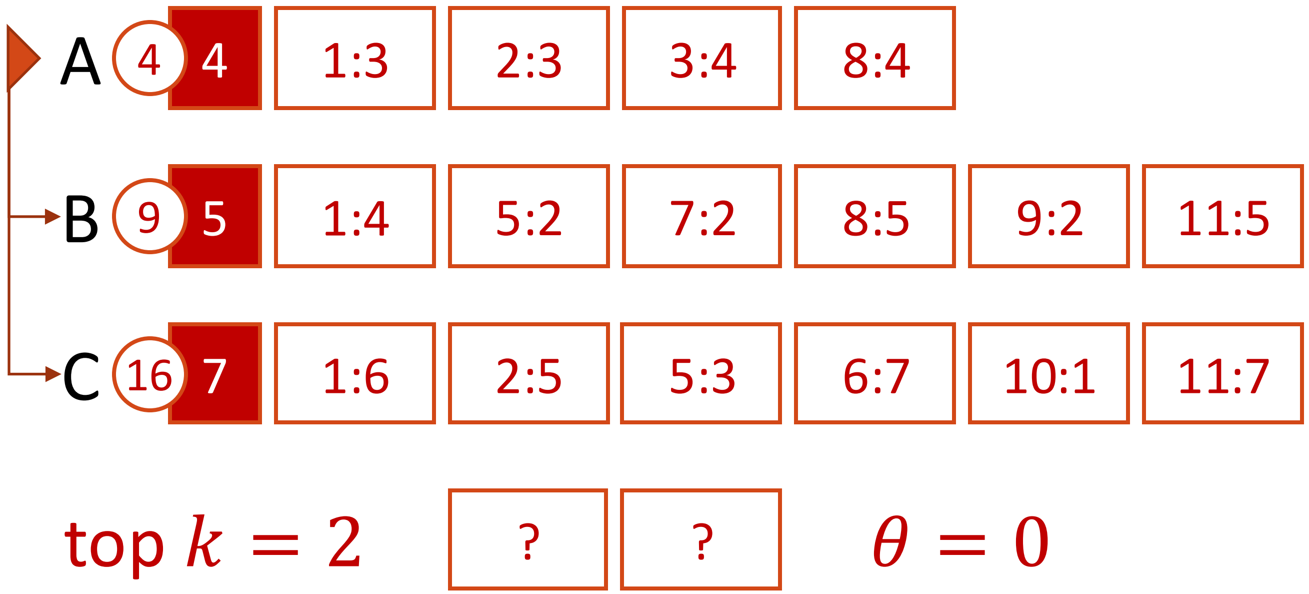
Key idea

- Traverse “essential” terms first (in DAAT mode)
- Check “non-essential” terms only if promising

MaxScore ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

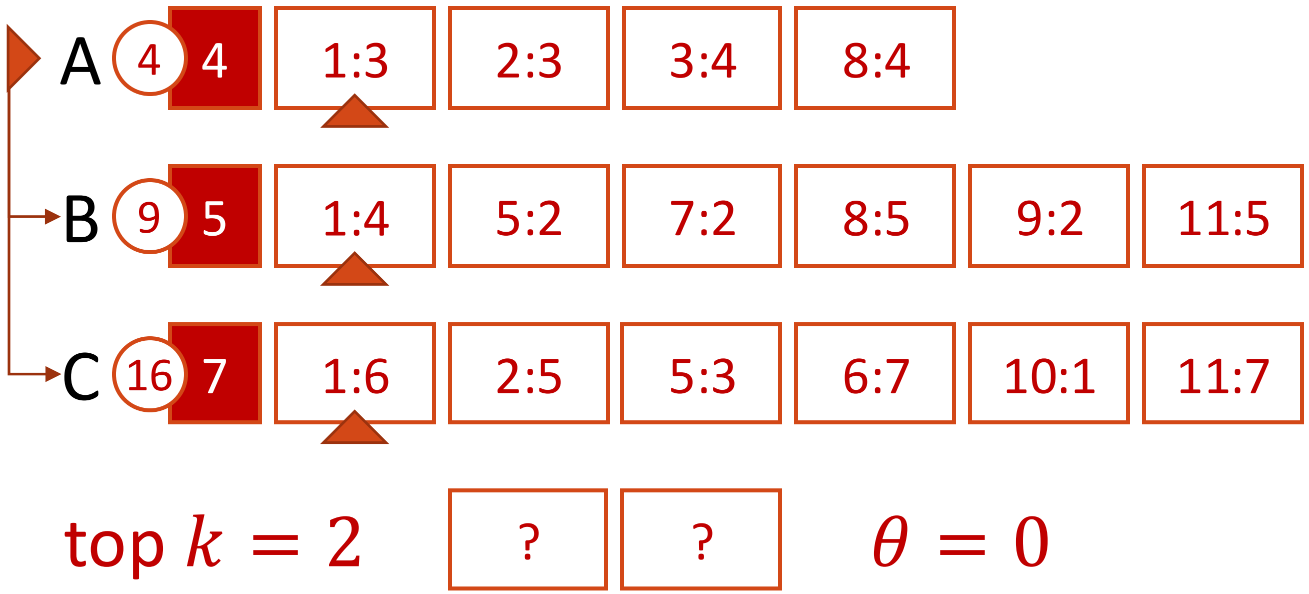


- terms sorted by inc. max-score
- pivot chosen as least term that cumulatively beats threshold θ
- terms at least as promising as the pivot deemed “essential”
  - others are “non-essential”

MaxScore ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

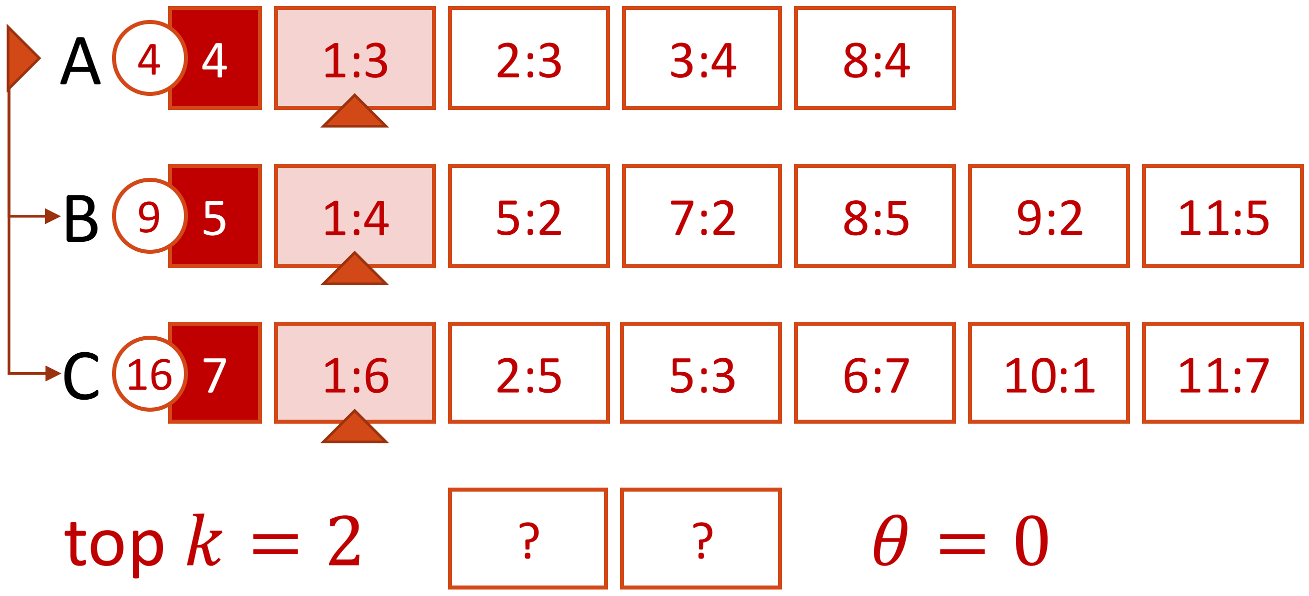


- process “essential” lists first

MaxScore ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

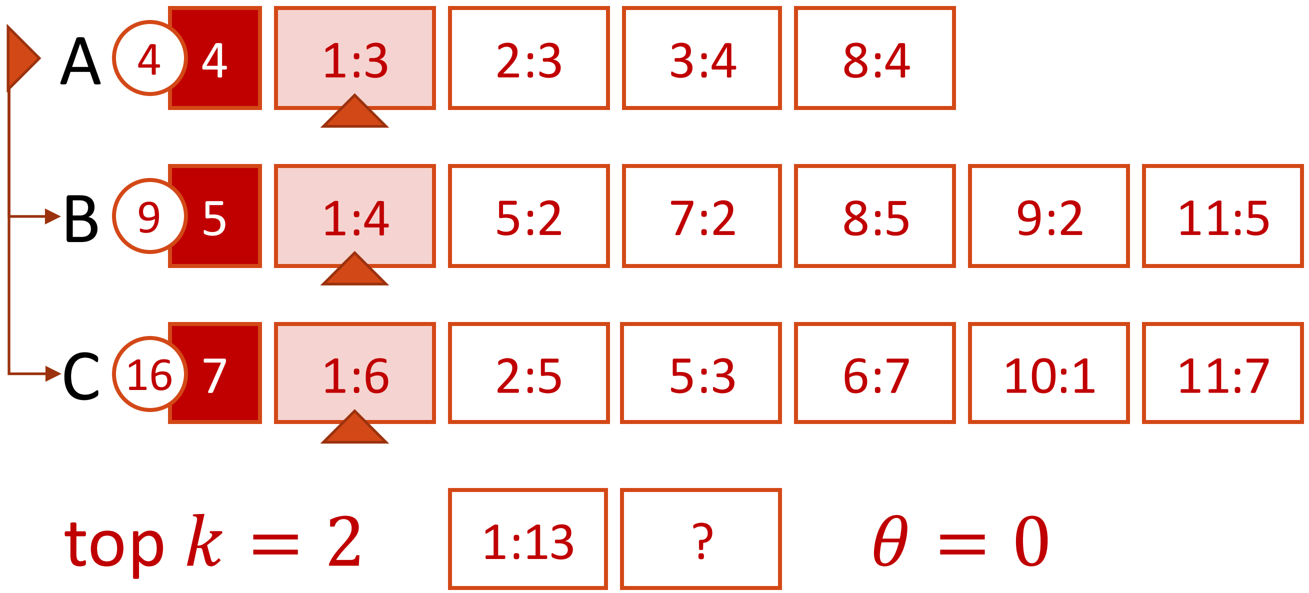


- process "essential" lists first
- process "non-essential" lists only if they are promising
- update top k results and θ

MaxScore ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

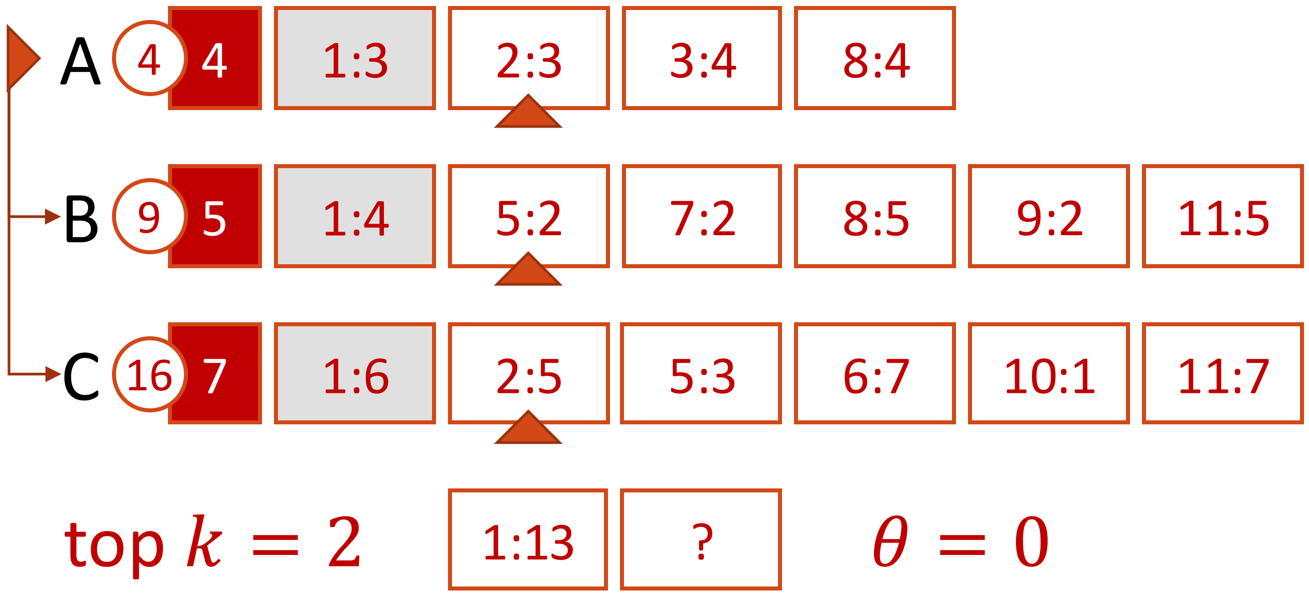


- process "essential" lists first
- process "non-essential" lists only if they are promising
- update top k results and θ
- update pivot on θ changes

MaxScore ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

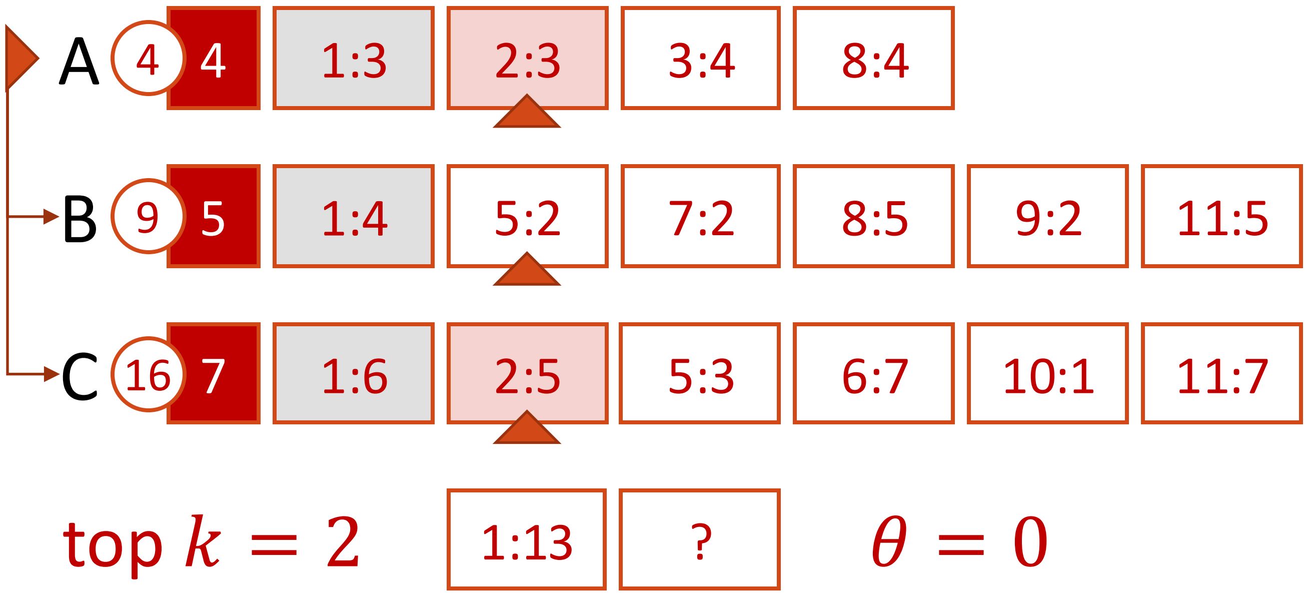


- process “essential” lists first

MaxScore ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

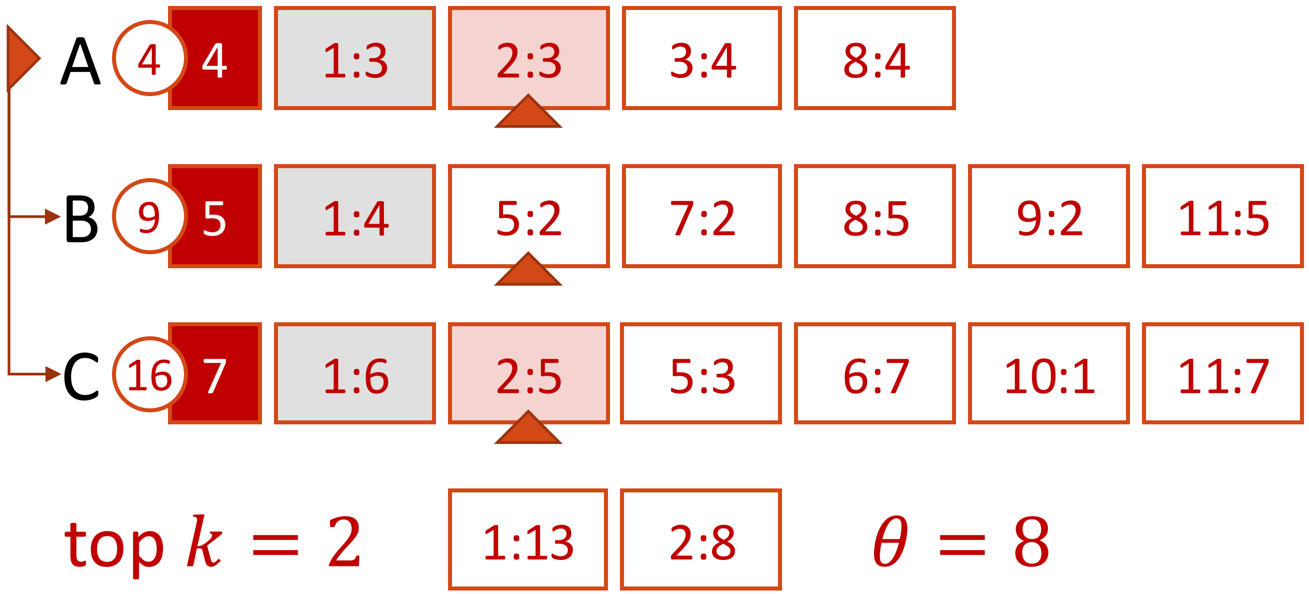


- process "essential" lists first
- process "non-essential" lists only if they are promising
- update top k results and θ

MaxScore ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ



- process “essential” lists first
- process “non-essential” lists only if they are promising
- update top k results and θ
- update pivot on θ changes

MaxScore ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

A	4	4	1:3	2:3	3:4	8:4			
B	9	5	1:4	5:2	7:2	8:5	9:2	11:5
C	16	7	1:6	2:5	5:3	6:7	10:1	11:7

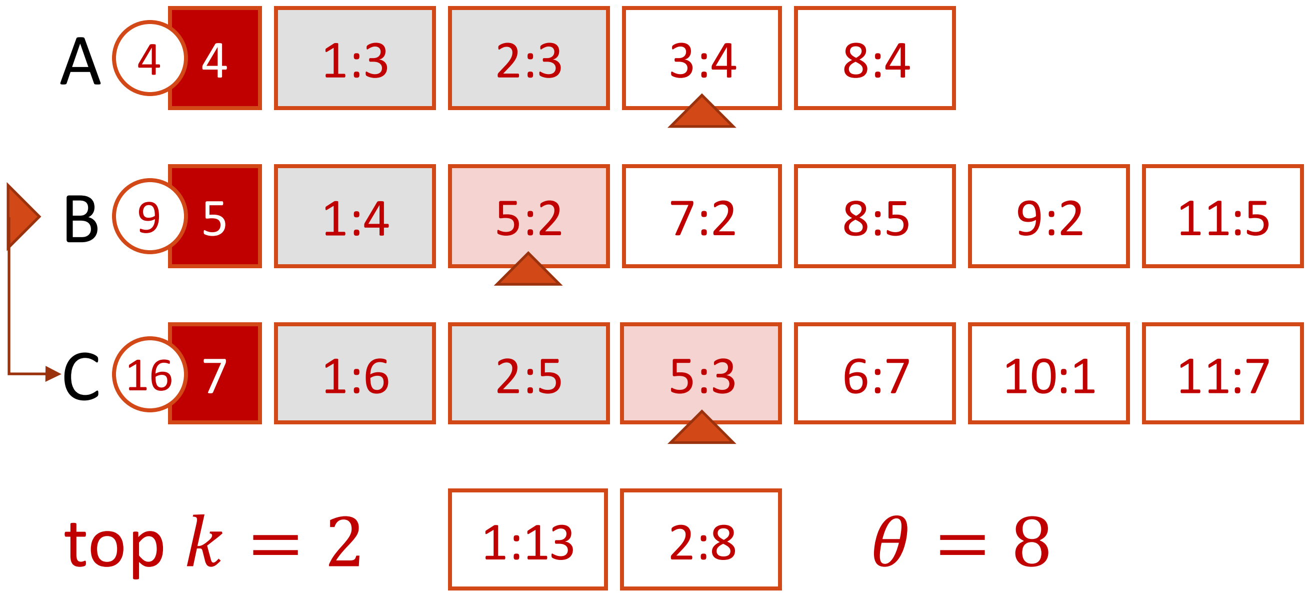
top $k = 2$ | 1:13 | 2:8 | $\theta = 8$

- process "essential" lists first

MaxScore ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

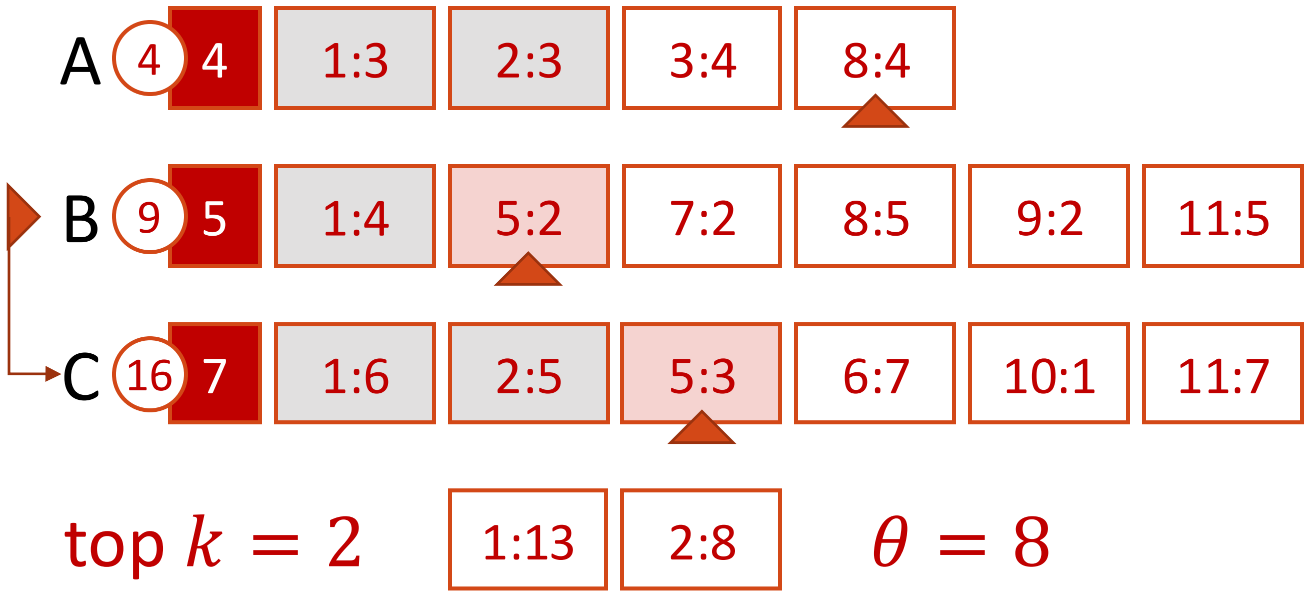


- process "essential" lists first
- process "non-essential" lists only if they are promising
  - A: $(2 + 3 + 4 = 9 > \theta)$ ✓

MaxScore ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

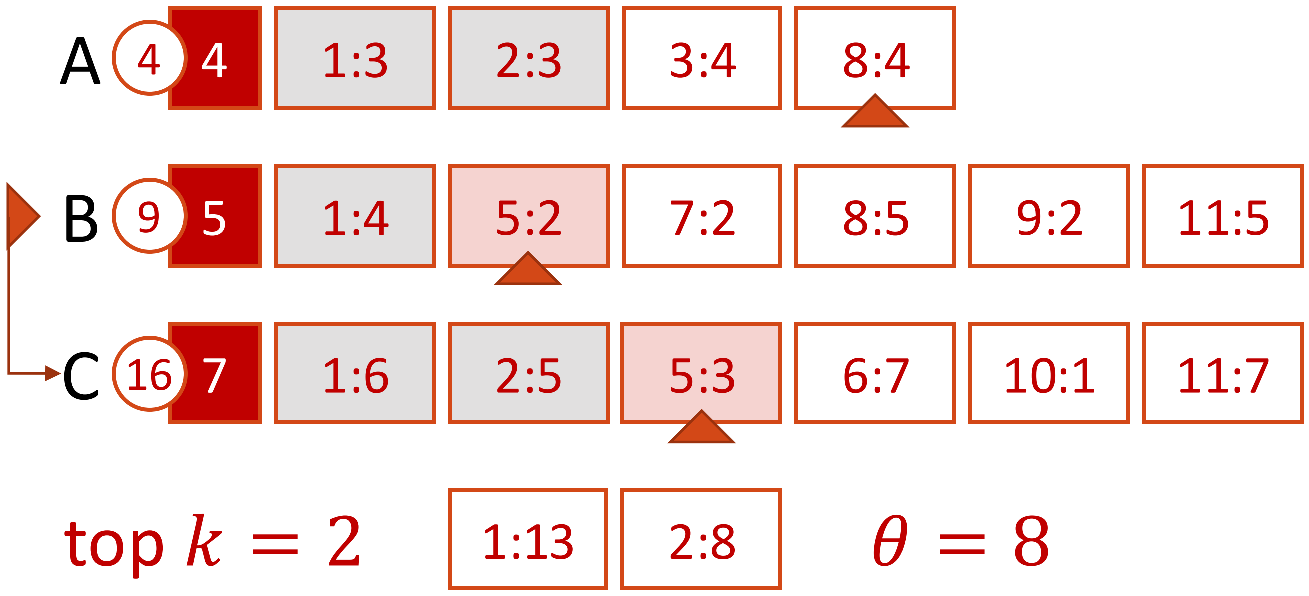


- process "essential" lists first
- process "non-essential" lists only if they are promising
  - A: $(2 + 3 + 4 = 9 > \theta)$ ✓
  - list miss on docid 5!

MaxScore ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

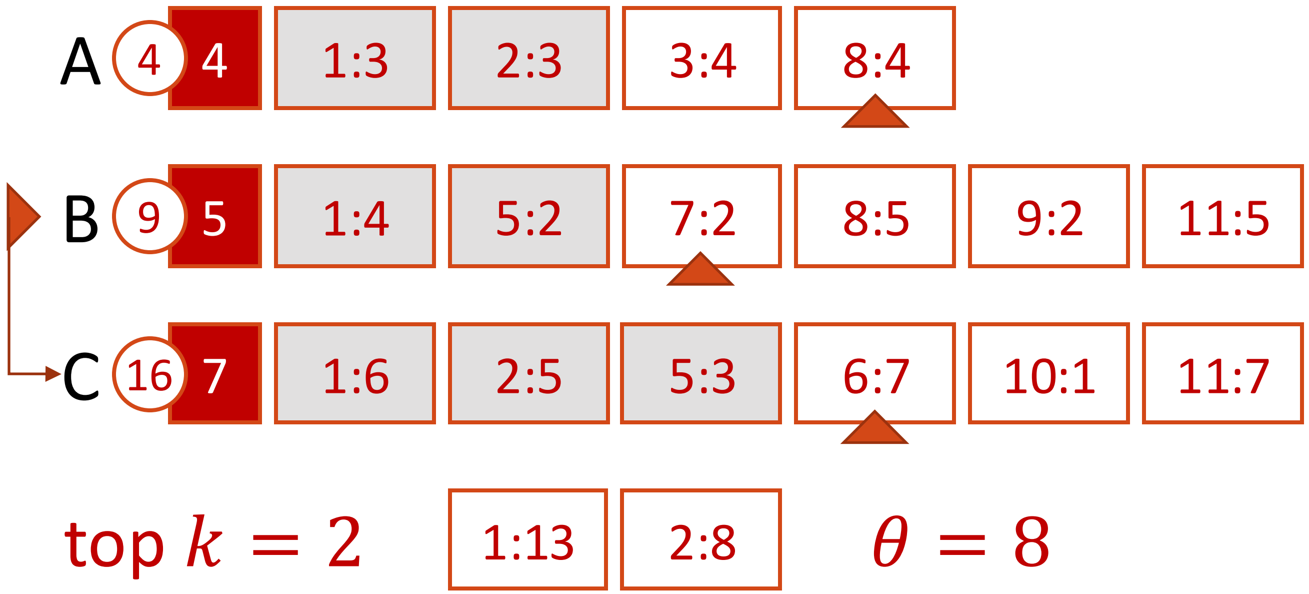


- process “essential” lists first
- process “non-essential” lists only if they are promising
- update top k results and θ
- update pivot on θ changes

MaxScore ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

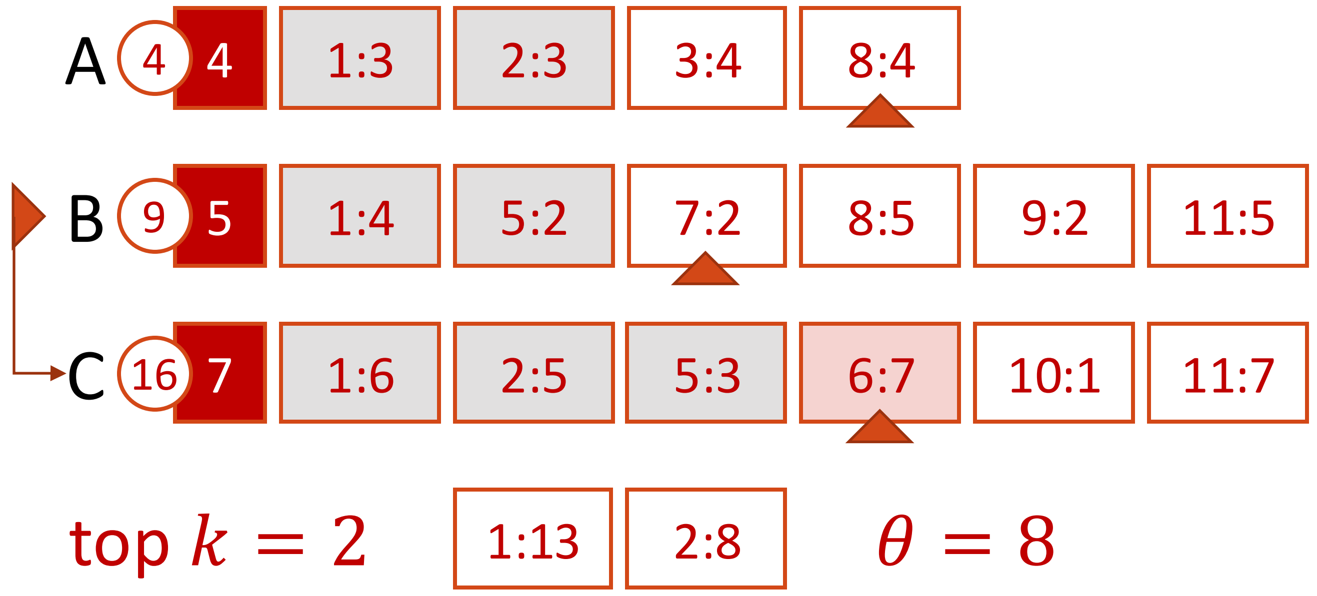


- process “essential” lists first

MaxScore ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

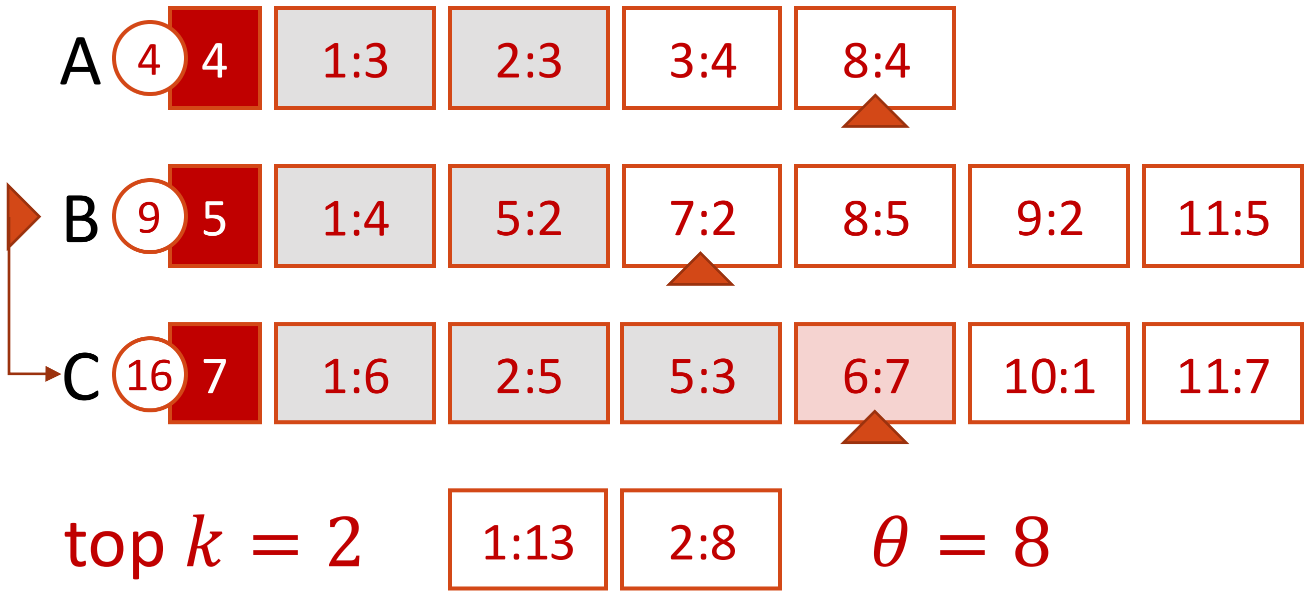


- process “essential” lists first
- process “non-essential” lists only if they are promising
  - A: $(7 + 4 = 11 > \theta)$ ✓
  - list miss on docid 6!

MaxScore ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

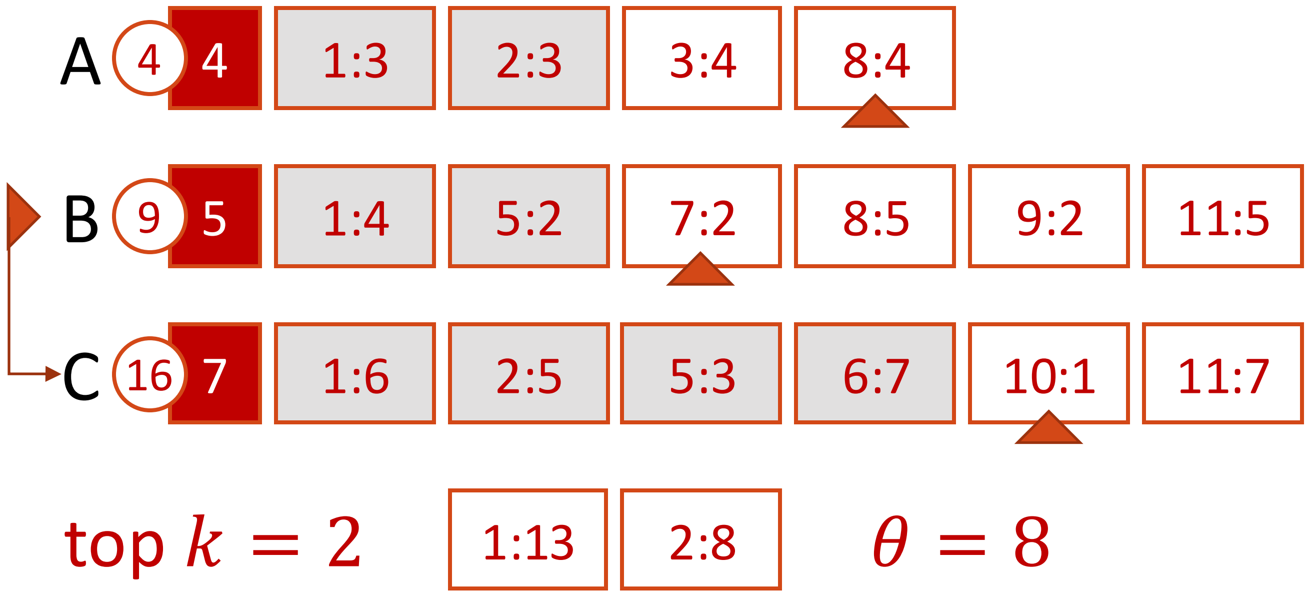


- process “essential” lists first
- process “non-essential” lists only if they are promising
- update top k results and θ
- update pivot on θ changes

MaxScore ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

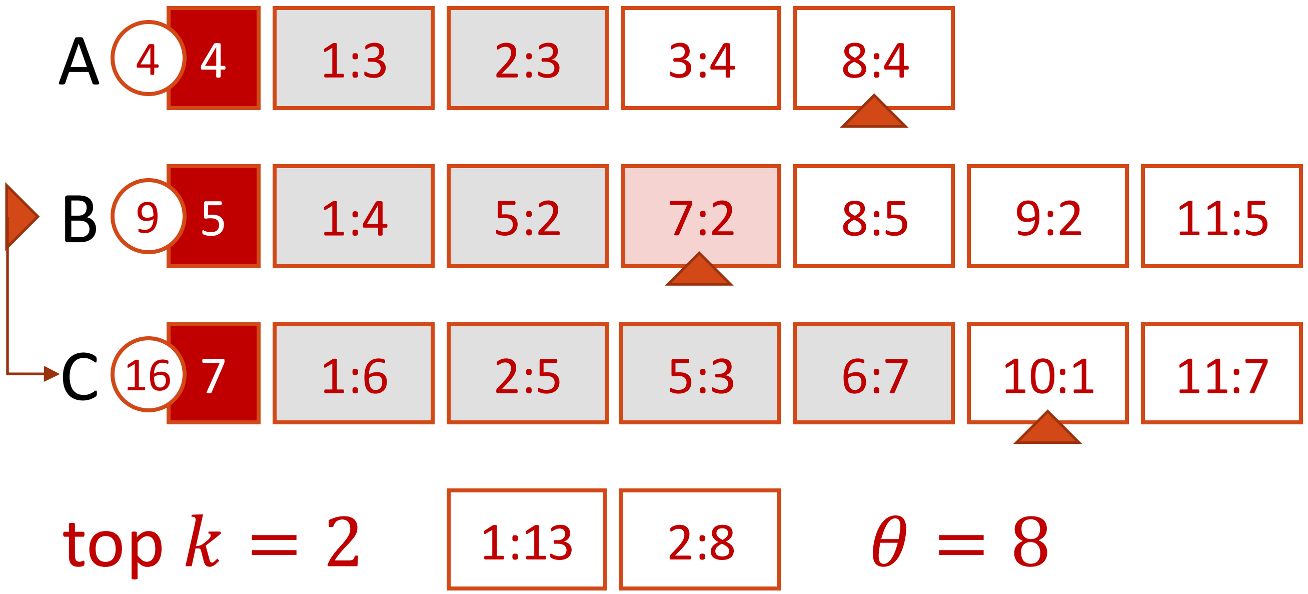


- process "essential" lists first

MaxScore ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

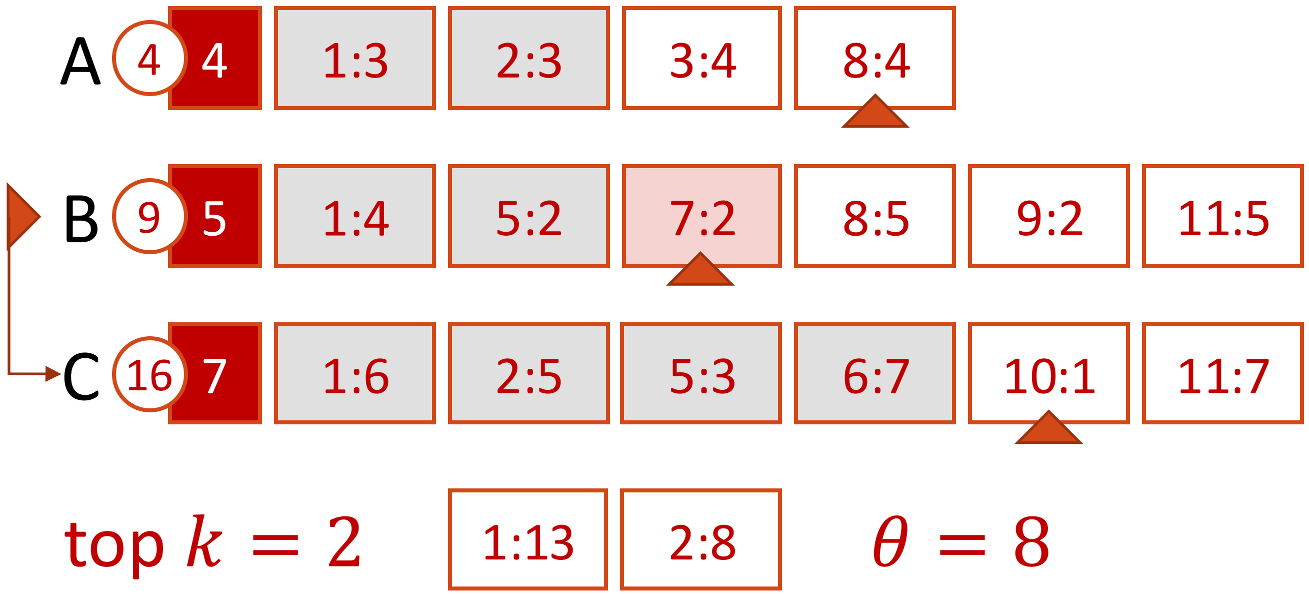


- process "essential" lists first
- process "non-essential" lists only if they are promising
  - A: $(2 + 4 = 6 \leq \theta)$ ✗

MaxScore ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

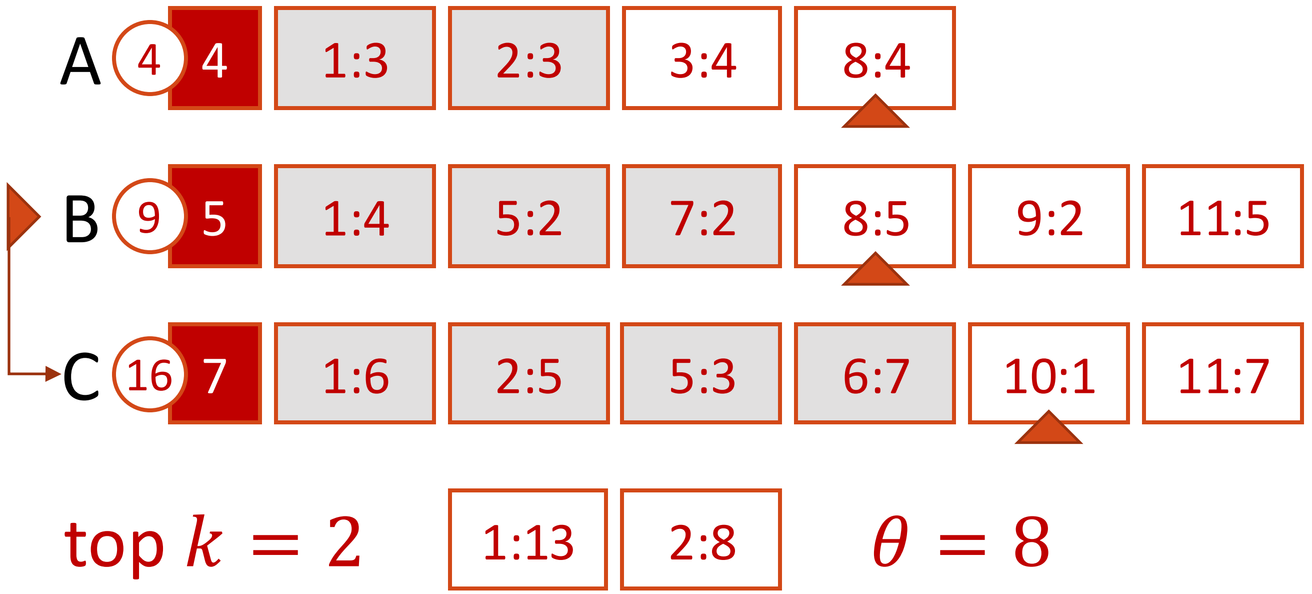


- process “essential” lists first
- process “non-essential” lists only if they are promising
- update top k results and θ
- update pivot on θ changes

MaxScore ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

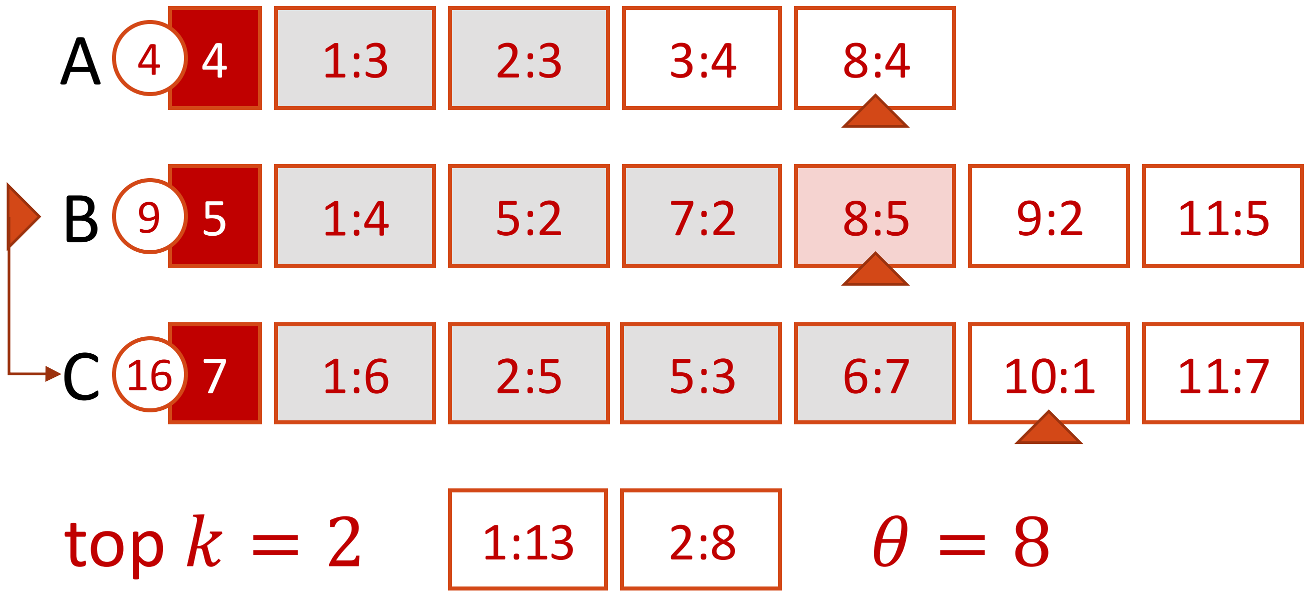


- process "essential" lists first

MaxScore ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

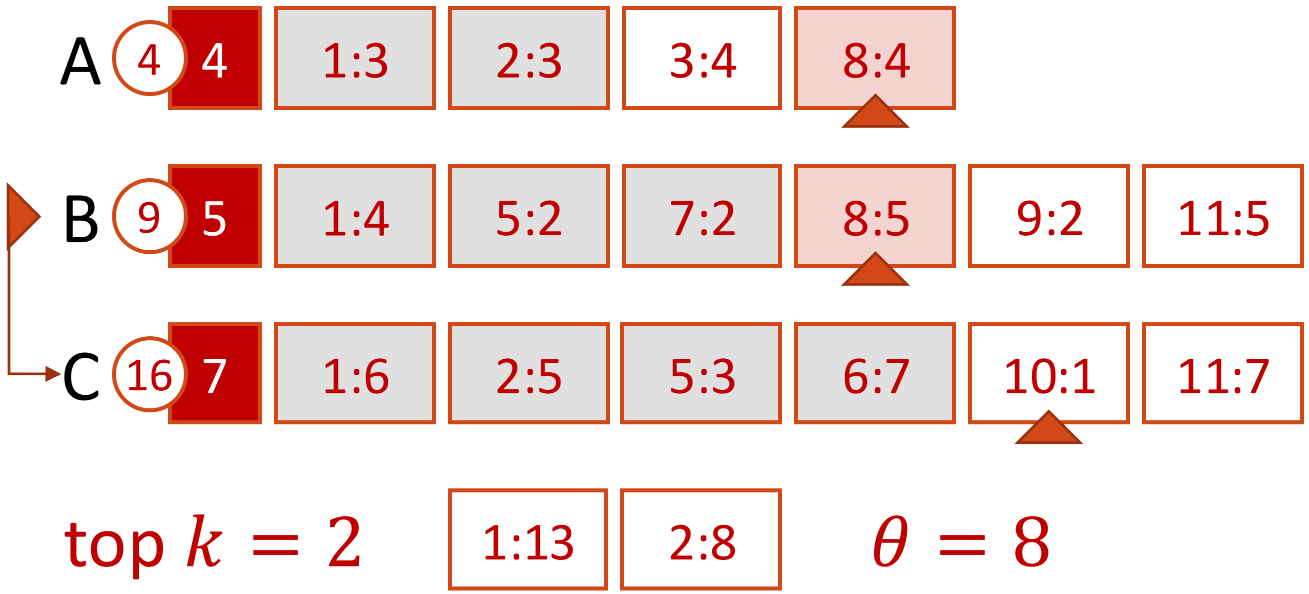


- process "essential" lists first
- process "non-essential" lists only if they are promising
  - A: $(5 + 4 = 9 > \theta)$ ✓

MaxScore ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

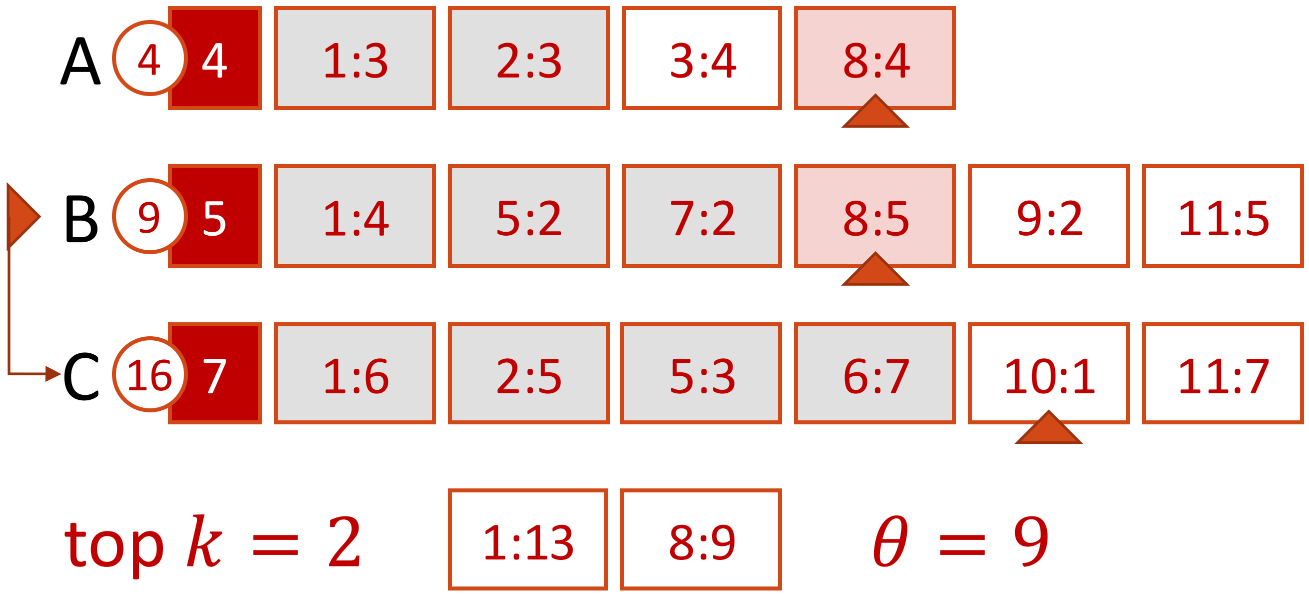


- process "essential" lists first
- process "non-essential" lists only if they are promising
- update top k results and θ

MaxScore ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

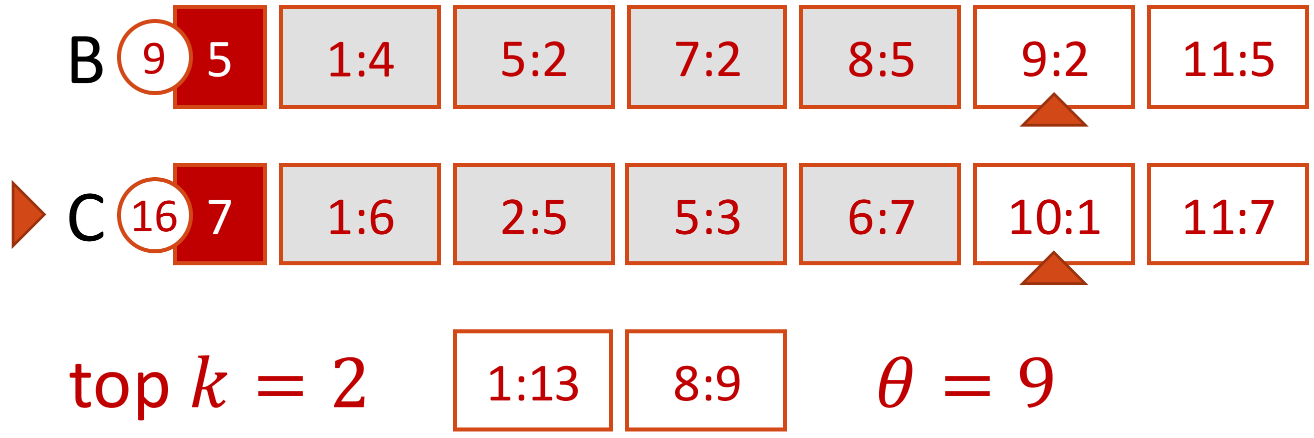


- process “essential” lists first
- process “non-essential” lists only if they are promising
- update top k results and θ
- update pivot on θ changes

MaxScore ($k = 2$)

Each list has an upper bound (aka max-score)

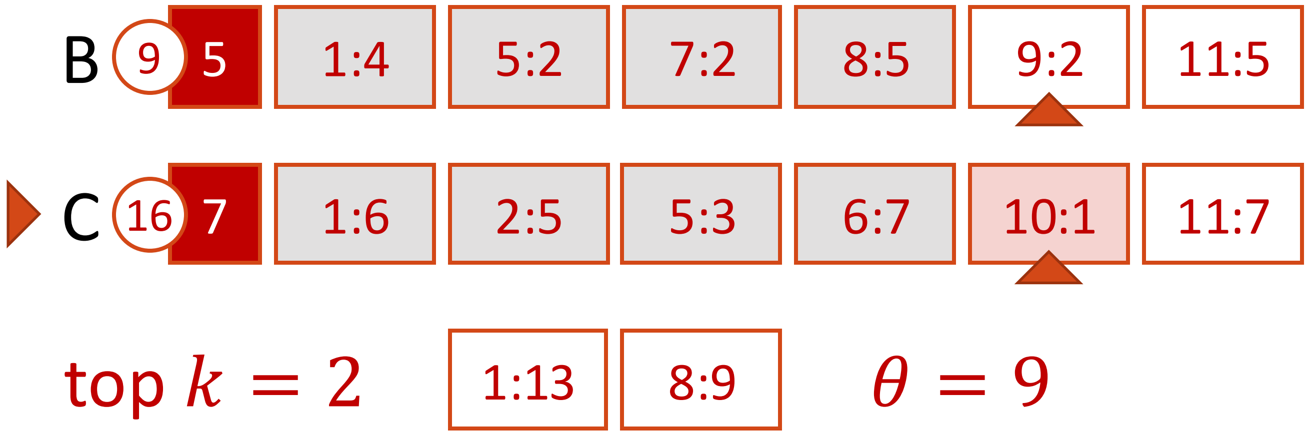
- Top k results have acceptance threshold θ
   - process "essential" lists first



MaxScore ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

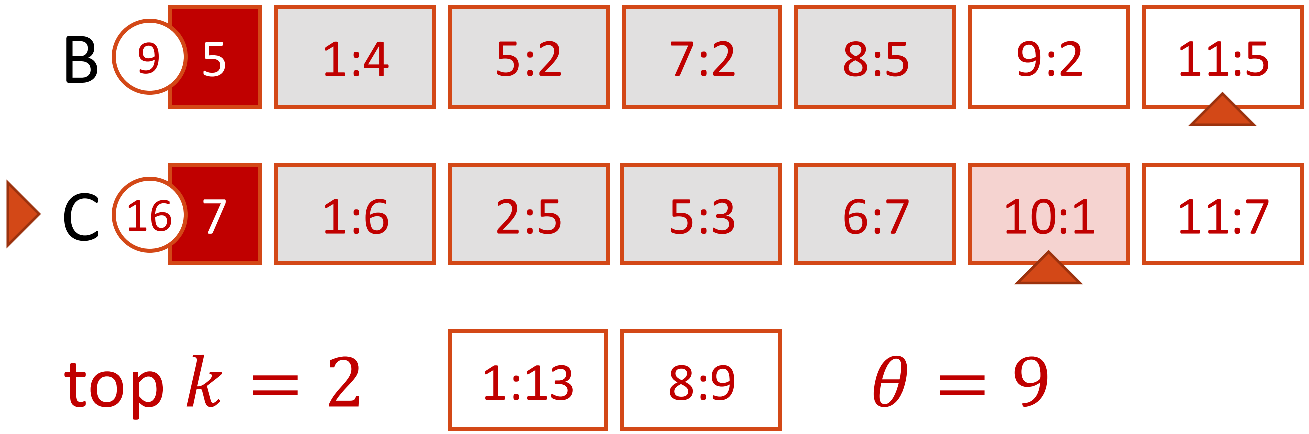


- process "essential" lists first
- process "non-essential" lists only if they are promising
  - B: $(1 + 9 = 10 > \theta)$ ✓

MaxScore ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

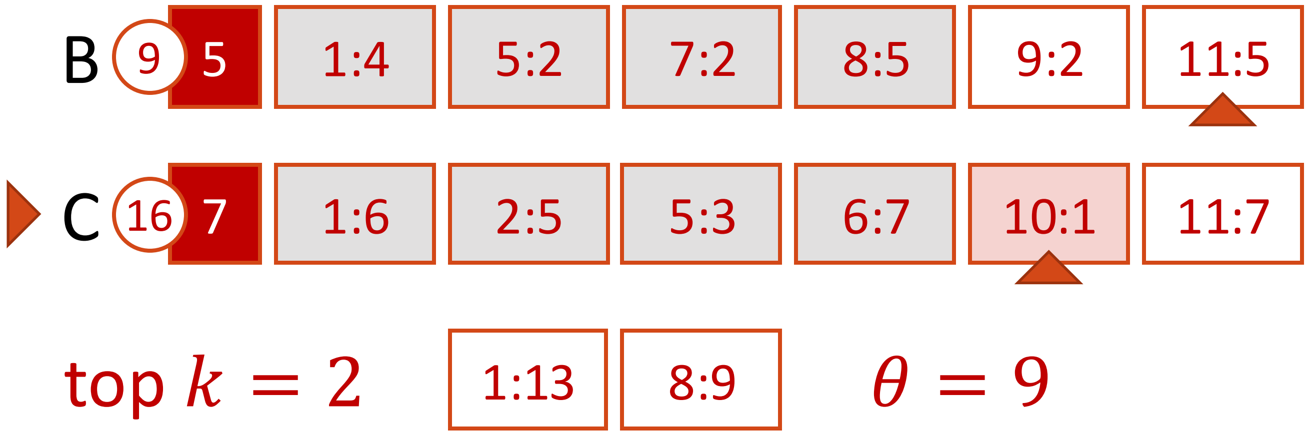


- process “essential” lists first
- process “non-essential” lists only if they are promising
  - B: $(1 + 9 = 10 > \theta)$ ✓
  - list miss on docid 10!

MaxScore ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ



- process "essential" lists first
- process "non-essential" lists only if they are promising
- update top k results and θ
- update pivot on θ changes

MaxScore ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

- process "essential" lists first

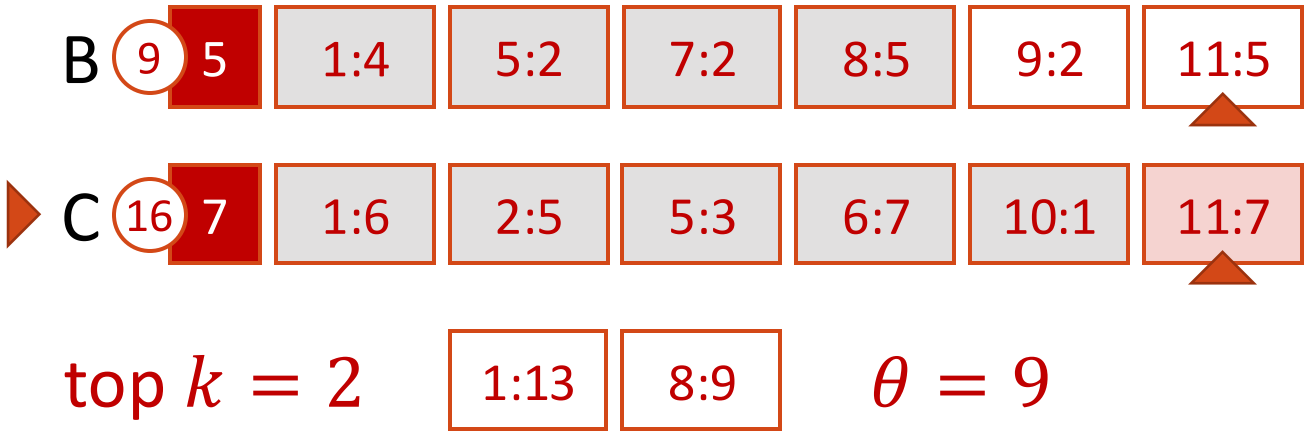
B	9	5	1:4	5:2	7:2	8:5	9:2	11:5
C	16	7	1:6	2:5	5:3	6:7	10:1	11:7

top $k = 2$ | 1:13 | 8:9 | $\theta = 9$

MaxScore ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ



- process “essential” lists first
- process “non-essential” lists only if they are promising
  - B: $(7 + 9 = 16 > \theta)$ ✓

MaxScore ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

B	9	5	1:4	5:2	7:2	8:5	9:2	11:5
C	16	7	1:6	2:5	5:3	6:7	10:1	11:7

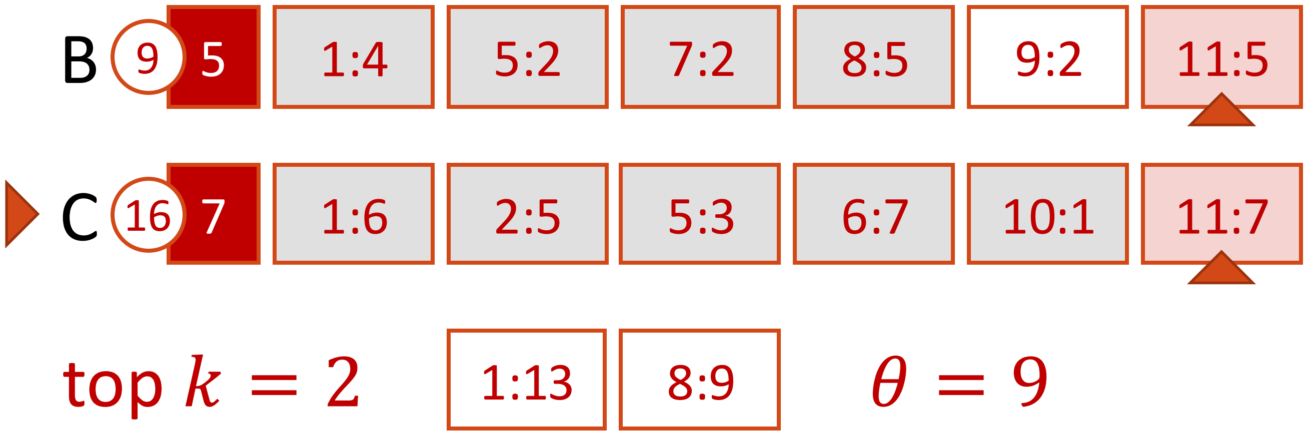
top $k = 2$ 1:13   8:9 $\theta = 9$

- process "essential" lists first
- process "non-essential" lists only if they are promising
  - B: $(7 + 9 = 16 > \theta)$ ✓

MaxScore ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

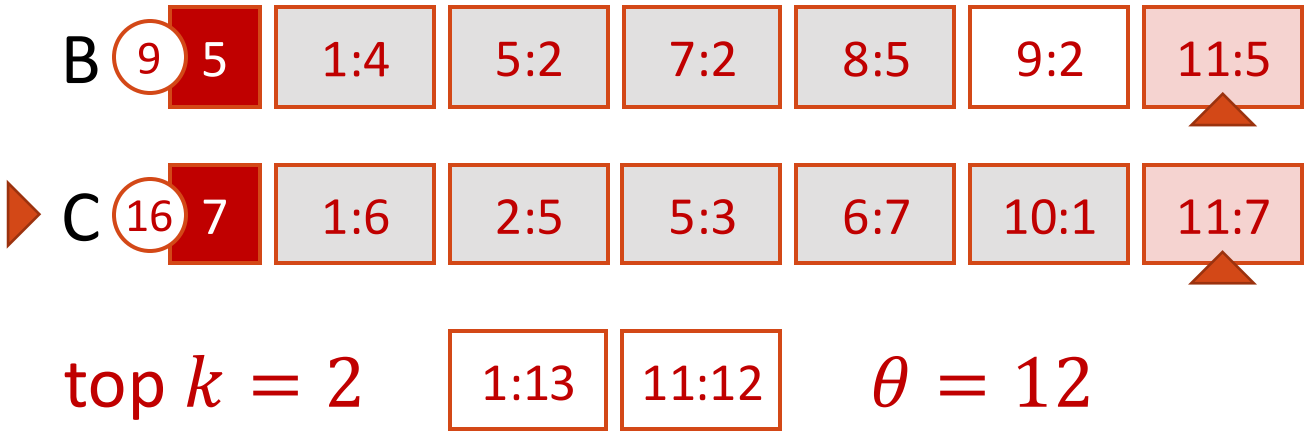


- process “essential” lists first
- process “non-essential” lists only if they are promising
- update top k results and θ

MaxScore ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ



- process “essential” lists first
- process “non-essential” lists only if they are promising
- update top k results and θ
- update pivot on θ changes

MaxScore ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

A	4	1:3	2:3	3:4	8:4		
B	5	1:4	5:2	7:2	8:5	9:2	11:5
C	7	1:6	2:5	5:3	6:7	10:1	11:7

top $k = 2$ | 1:13 | 11:12 | $\theta = 12$

- 2 / 16 postings skipped
  ≈ 12% savings

MaxScore limitations

MaxScore relies on “non-essential” terms for skipping

- Naïve DAAT performed on “essential” terms

It may take long for a term to become “non-essential”

- It may be a poor term (low max-score)
- It may get hard to make the heap (high threshold)

Hindered efficiency, particularly for long queries

WAND [Broder et al., CIKM 2003]

MaxScore fully evaluates “essential” lists

- Not all documents in “essential” lists are promising

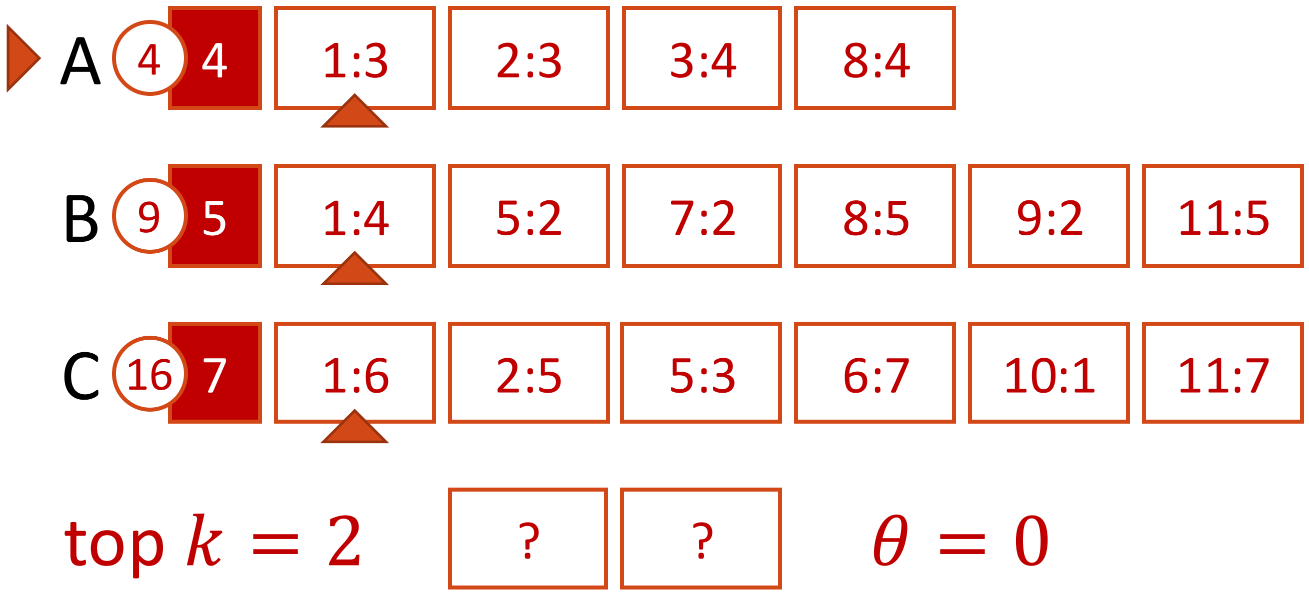
Key idea

- Evaluate *documents* (not lists) if they are promising (i.e. have a promising cumulative upper bound)

WAND ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

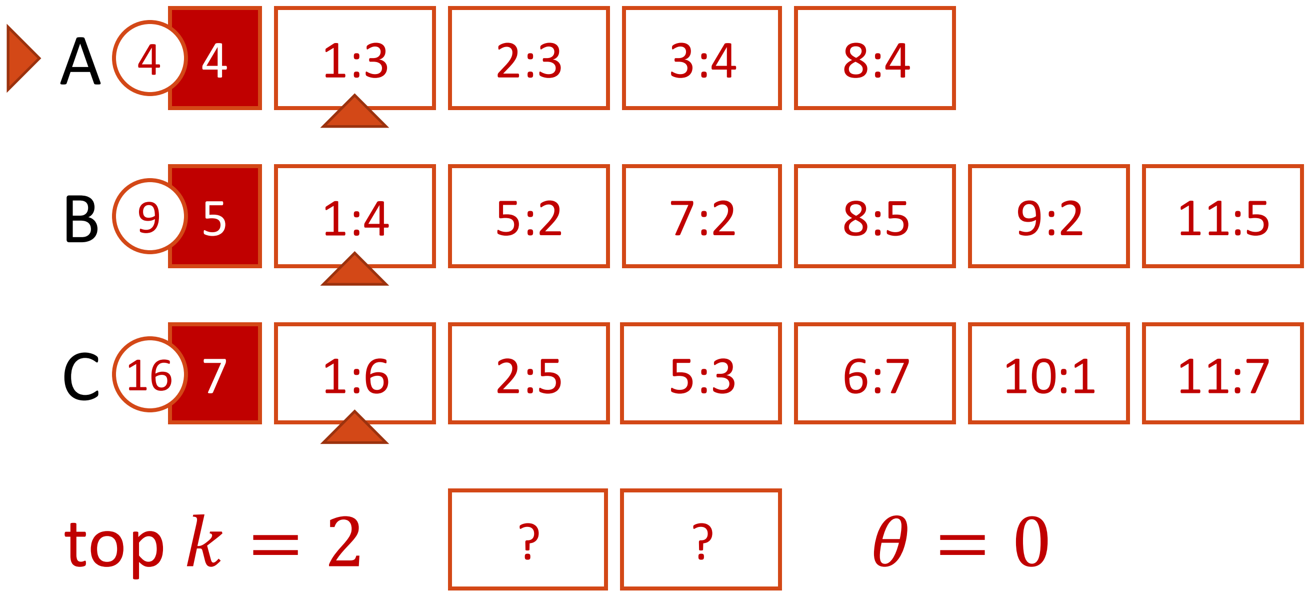


- terms sorted by inc. docid
- pivot chosen as least term that cumulatively beats threshold θ
- terms managed dynamically
  - lists synced to pivot document
  - terms re-sorted on every move

WAND ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ



- locate pivot term p
- evaluate pivot document if present in all lists up to p; else align lists up to p

WAND ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

A	4	4	1:3	2:3	3:4	8:4		
B	9	5	1:4	5:2	7:2	8:5	9:2	11:5
C	16	7	1:6	2:5	5:3	6:7	10:1	11:7

top $k = 2$ | ? | ? | $\theta = 0$

- locate pivot term p
- evaluate pivot document if present in all lists up to p; else align lists up to p

WAND ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

A	4	4	1:3	2:3	3:4	8:4			
B	9	5	1:4	5:2	7:2	8:5	9:2	11:5	
C	16	7	1:6	2:5	5:3	6:7	10:1	11:7	

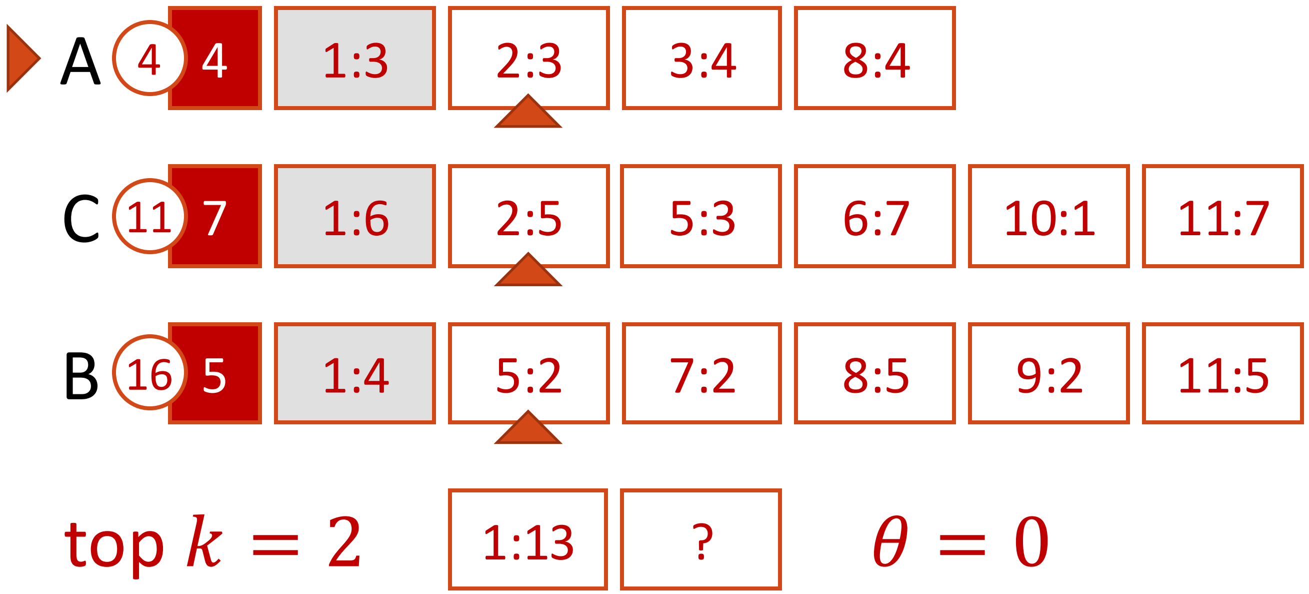
top $k = 2$ | 1:13 | ? | $\theta = 0$

- locate pivot term p
- evaluate pivot document if present in all lists up to p; else align lists up to p
- re-sort terms by docid

WAND ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

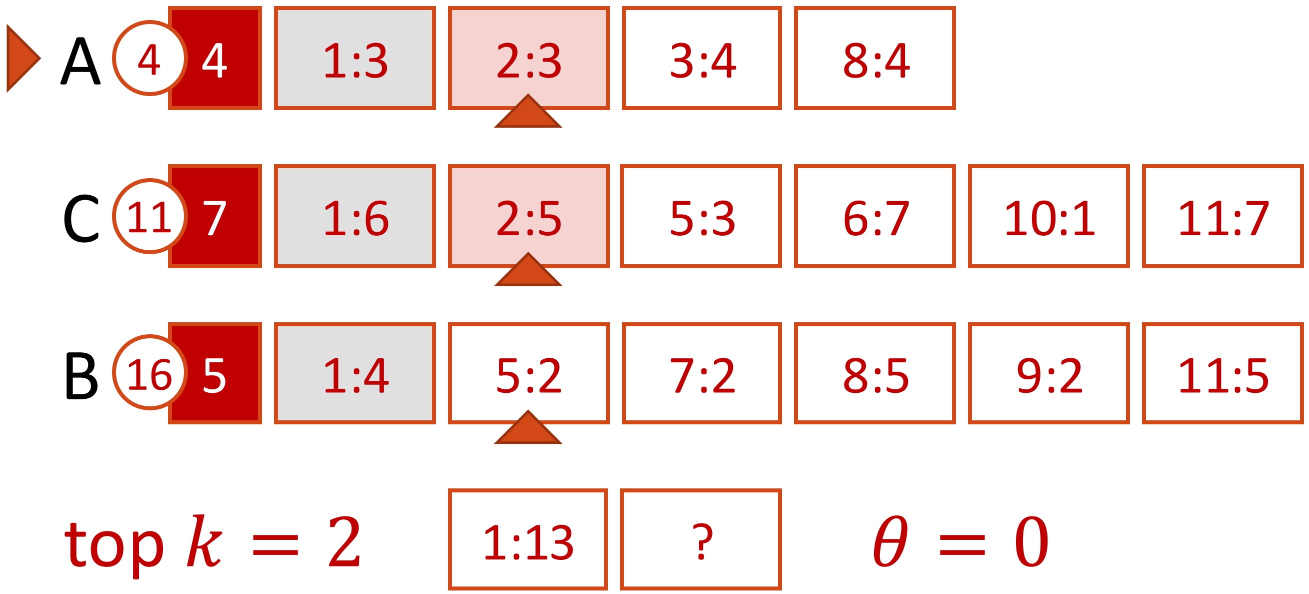


- locate pivot term p
- evaluate pivot document if present in all lists up to p; else align lists up to p

WAND ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

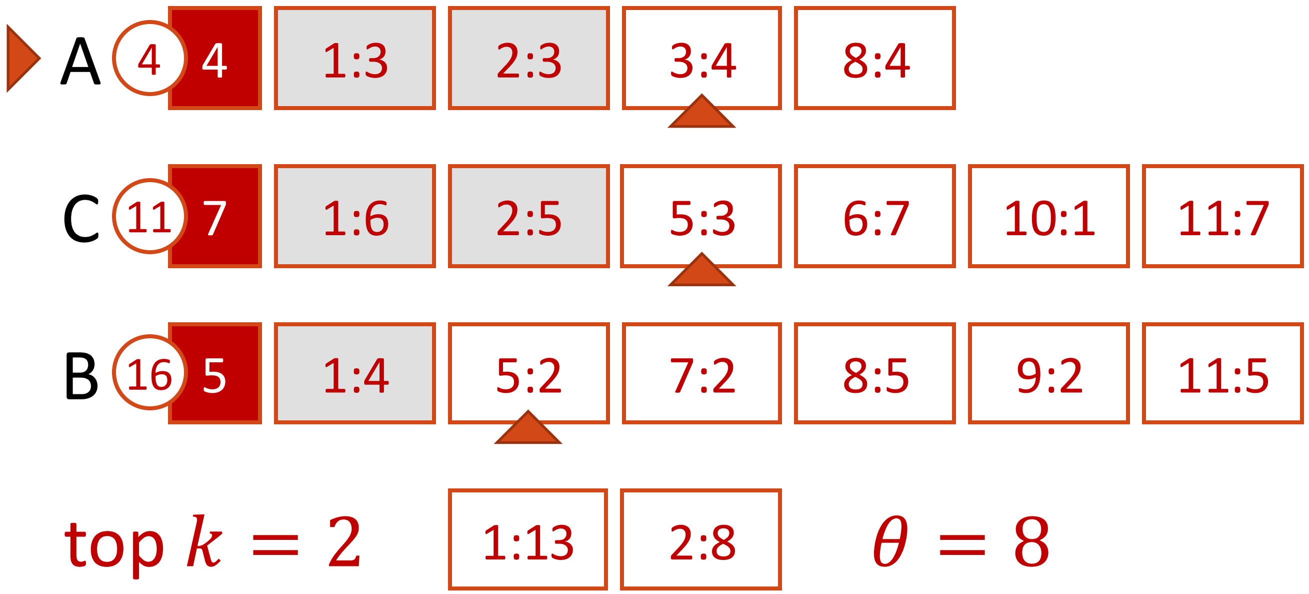


- locate pivot term p
- evaluate pivot document if present in all lists up to p; else align lists up to p

WAND ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

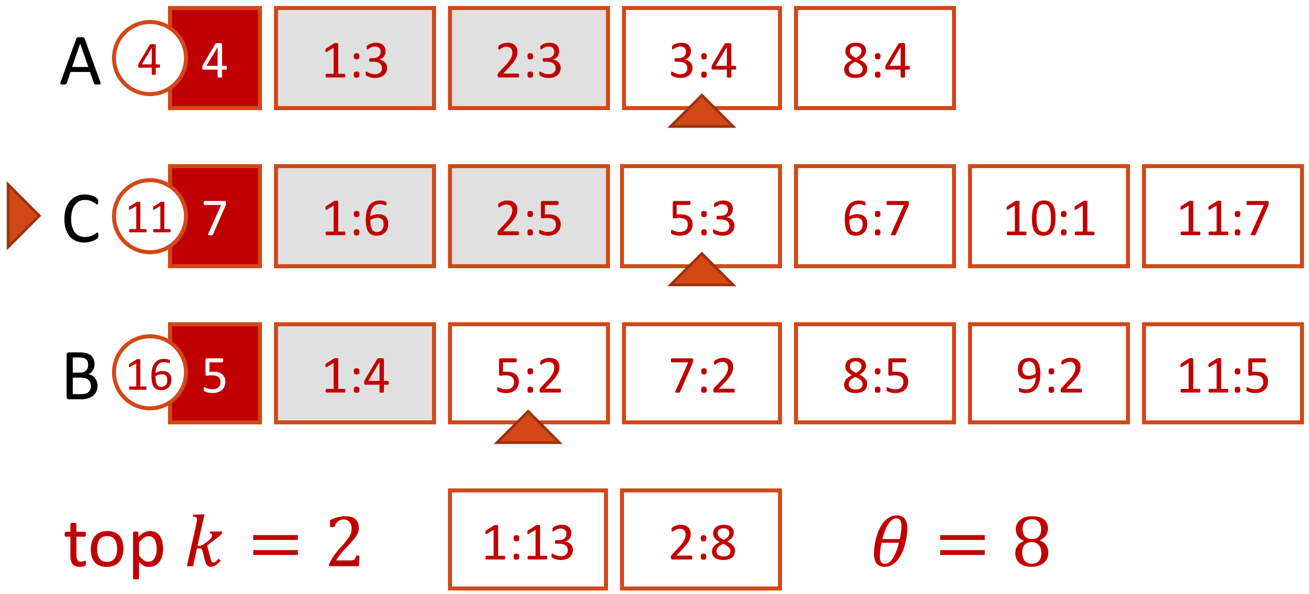


- locate pivot term p
- evaluate pivot document if present in all lists up to p; else align lists up to p
- re-sort terms by docid

WAND ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

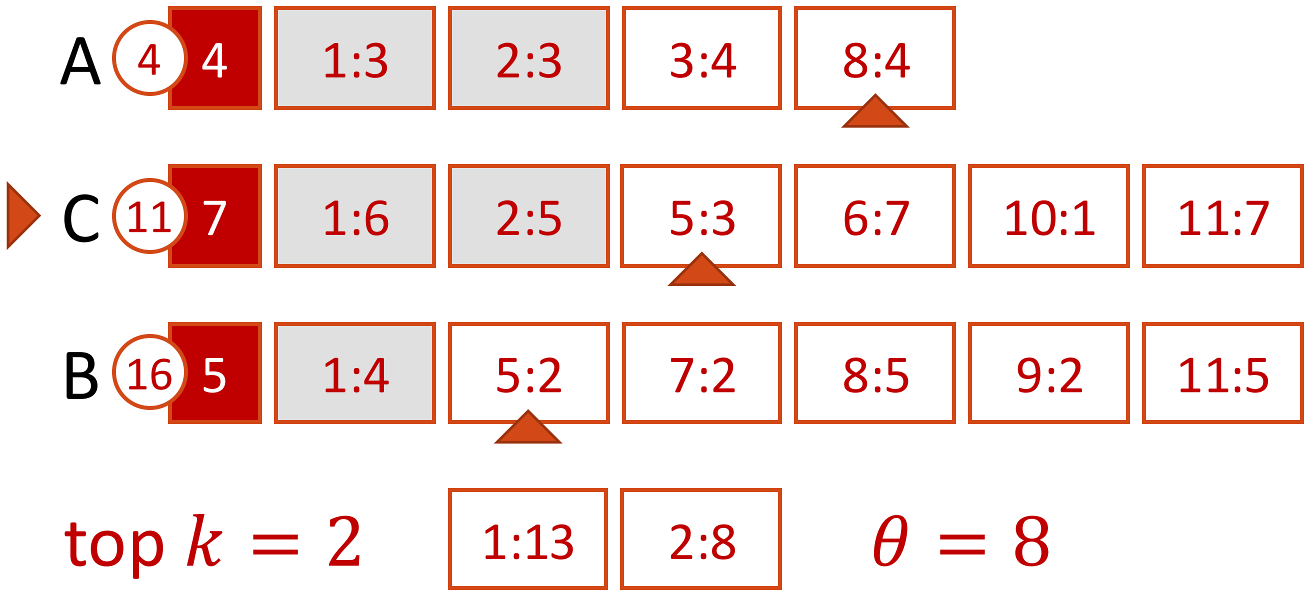


- locate pivot term p
- evaluate pivot document if present in all lists up to p; else align lists up to p

WAND ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

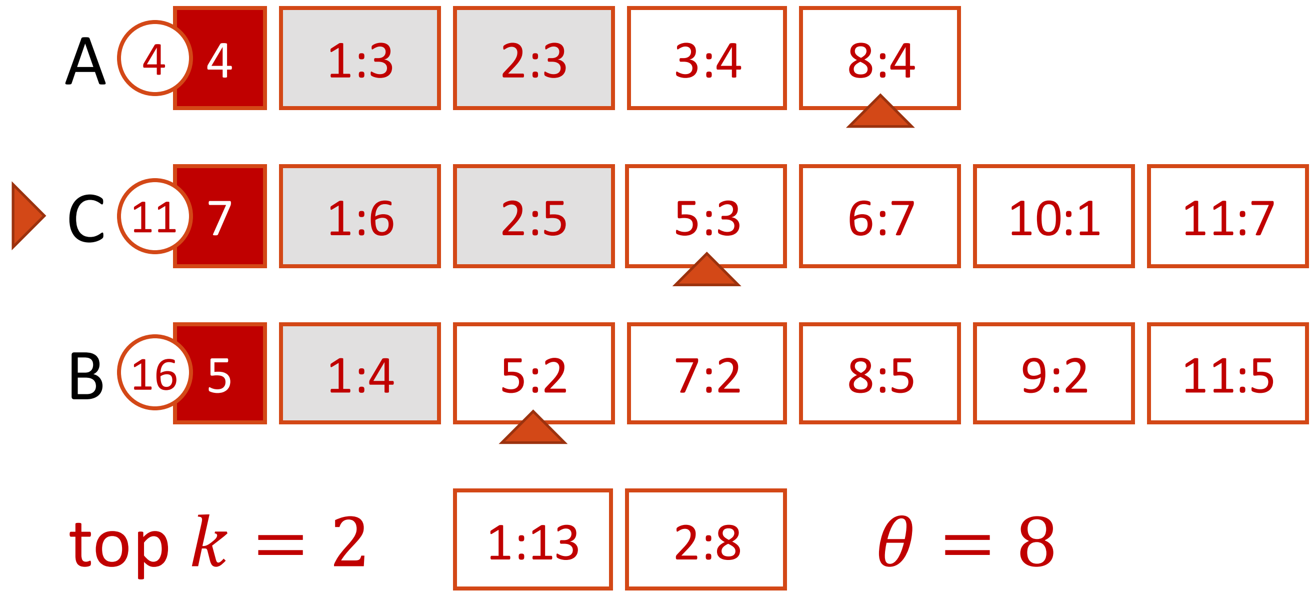


- locate pivot term p
- evaluate pivot document if present in all lists up to p; else align lists up to p
  - list miss on docid 5!

WAND ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

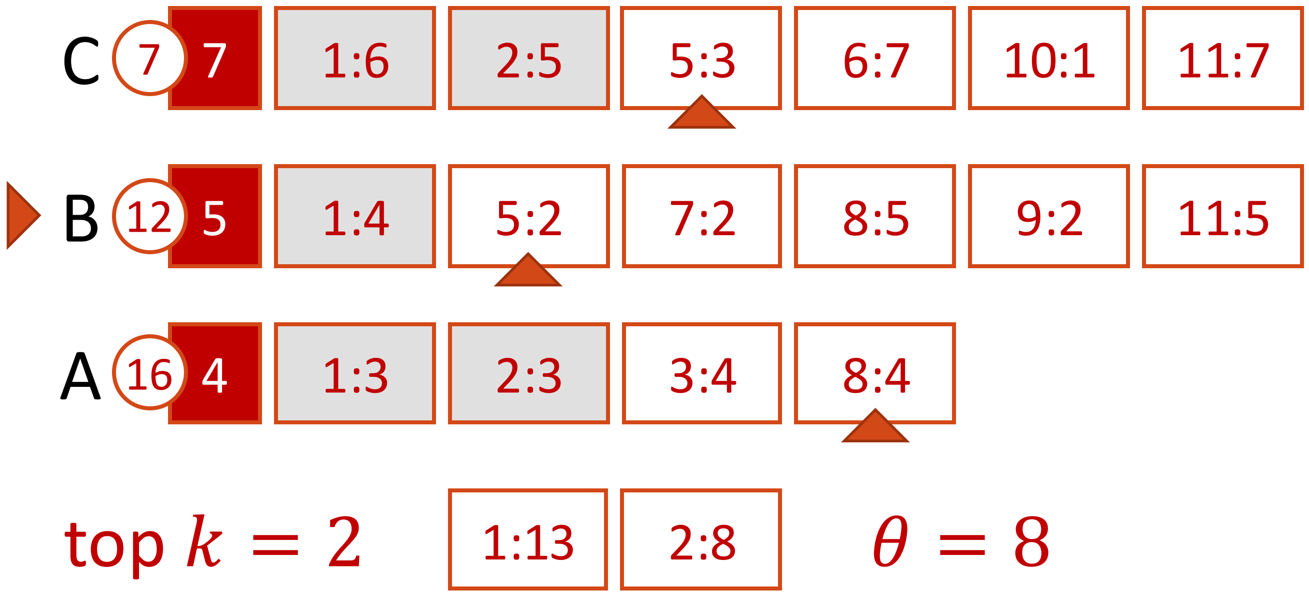


- locate pivot term p
- evaluate pivot document if present in all lists up to p; else align lists up to p
- re-sort terms by docid

WAND ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ



- locate pivot term p
- evaluate pivot document if present in all lists up to p; else align lists up to p

WAND ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

C	7	7	1:6	2:5	5:3	6:7	10:1	11:7
B	12	5	1:4	5:2	7:2	8:5	9:2	11:5
A	16	4	1:3	2:3	3:4	8:4		

top $k = 2$ | 1:13 | 2:8 | $\theta = 8$

- locate pivot term p
- evaluate pivot document if present in all lists up to p; else align lists up to p

WAND ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

C	7	7	1:6	2:5	5:3	6:7	10:1	11:7
B	12	5	1:4	5:2	7:2	8:5	9:2	11:5
A	16	4	1:3	2:3	3:4	8:4		

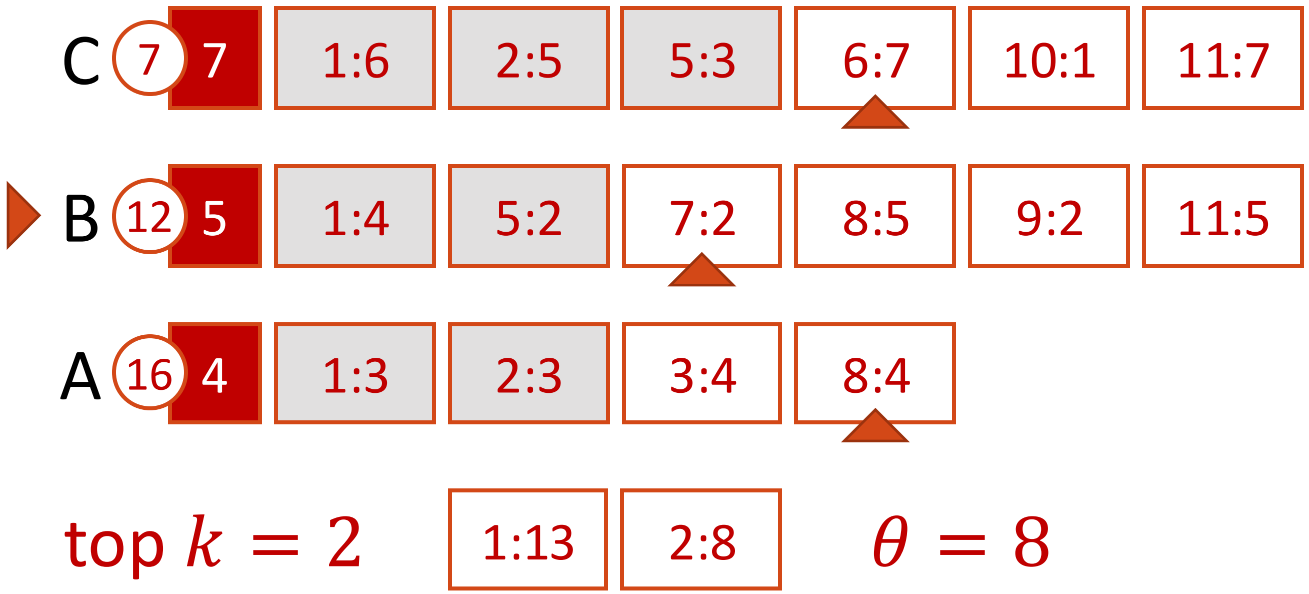
top $k = 2$ | 1:13 | 2:8 | $\theta = 8$

- locate pivot term p
- evaluate pivot document if present in all lists up to p; else align lists up to p
- re-sort terms by docid

WAND ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

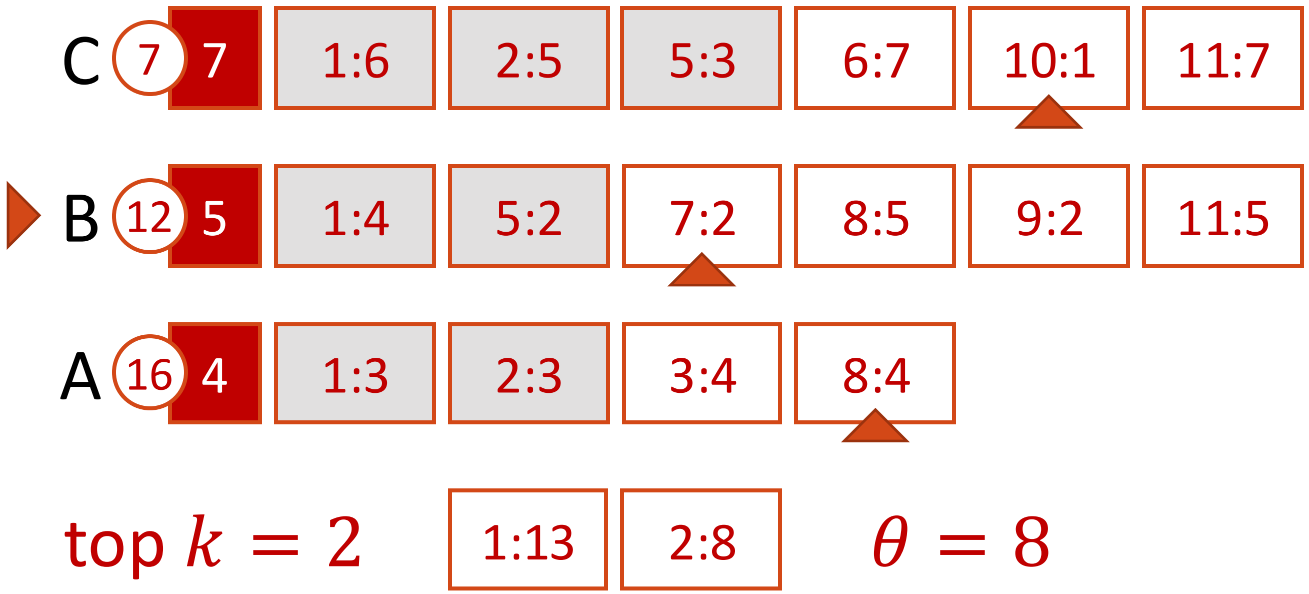


- locate pivot term p
- evaluate pivot document if present in all lists up to p; else align lists up to p

WAND ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

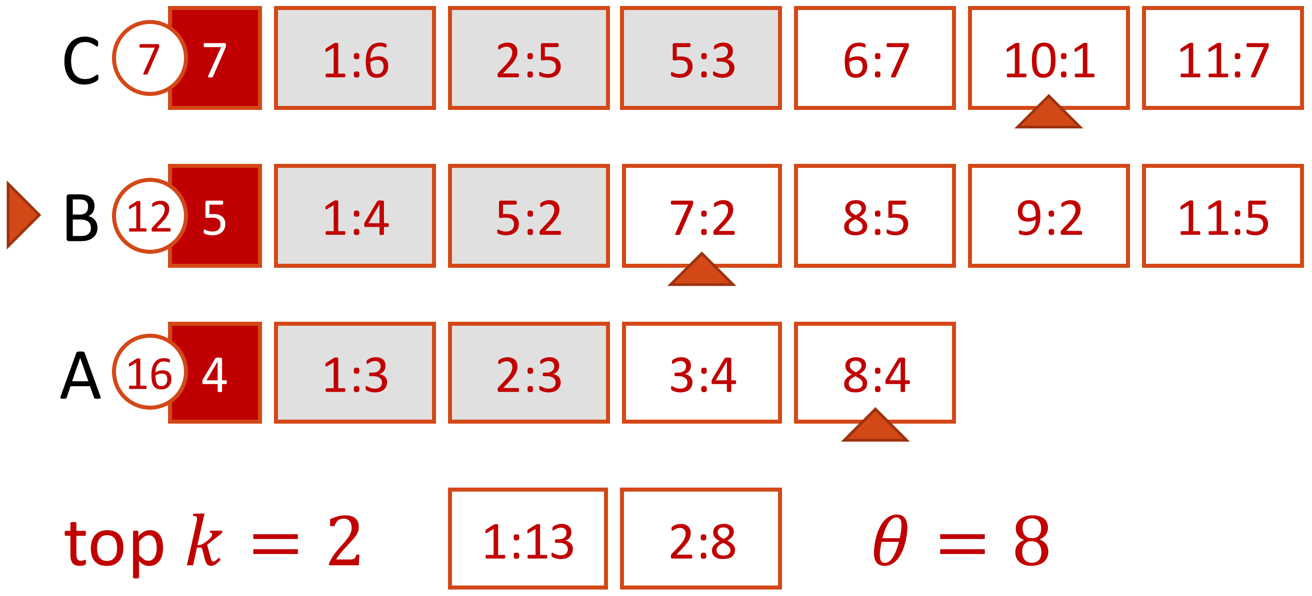


- locate pivot term p
- evaluate pivot document if present in all lists up to p; else align lists up to p
  - list miss on docid 7!

WAND ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

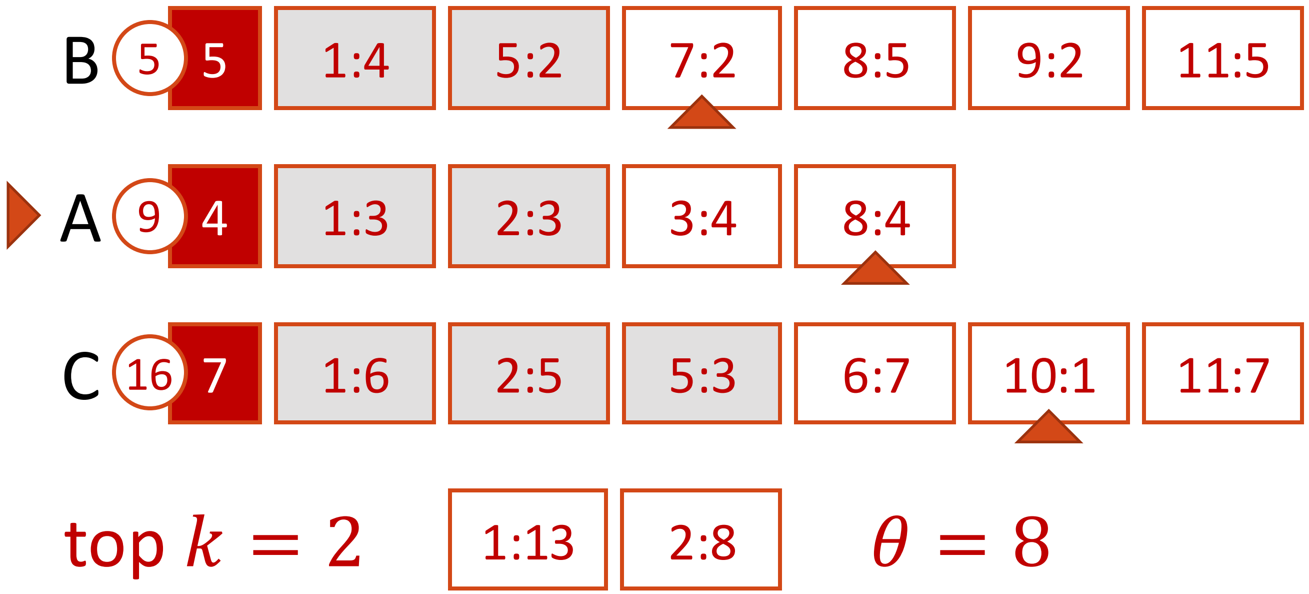


- locate pivot term p
- evaluate pivot document if present in all lists up to p; else align lists up to p
- re-sort terms by docid

WAND ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

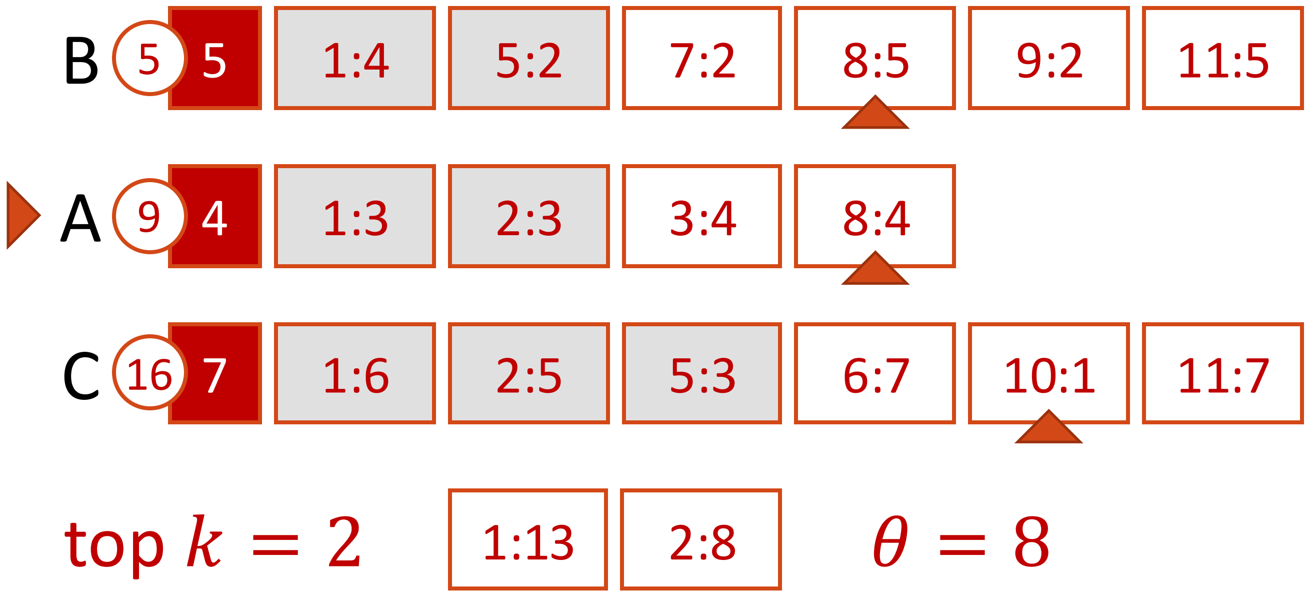


- locate pivot term p
- evaluate pivot document if present in all lists up to p; else align lists up to p

WAND ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

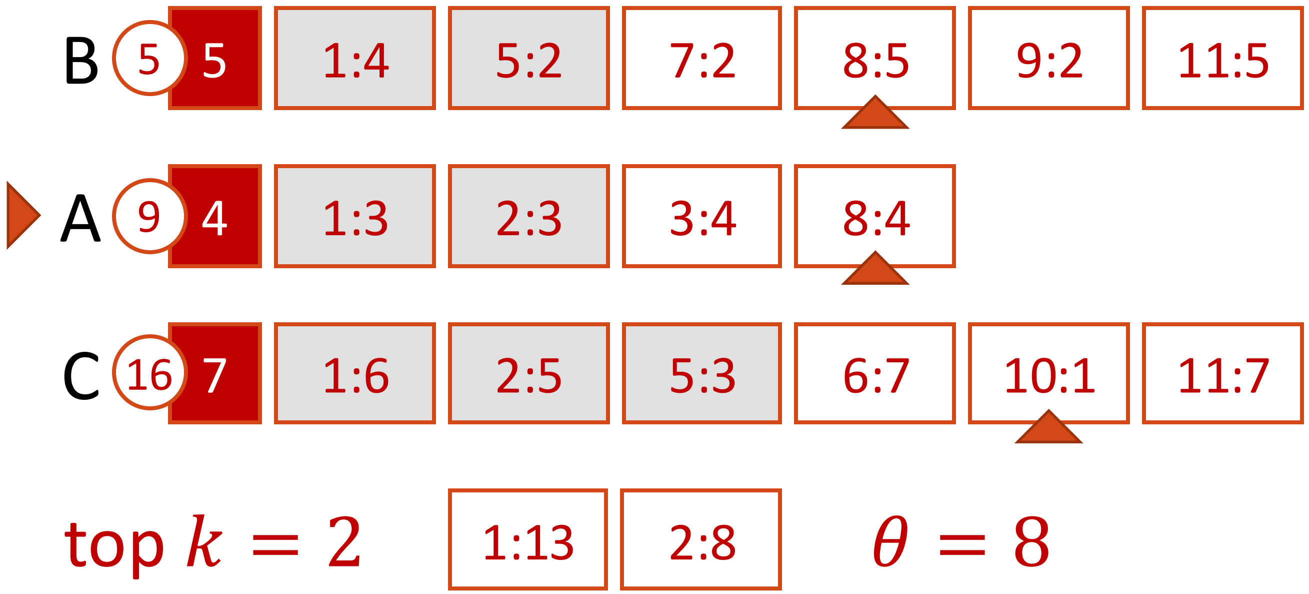


- locate pivot term p
- evaluate pivot document if present in all lists up to p; else align lists up to p
- re-sort terms by docid

WAND ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

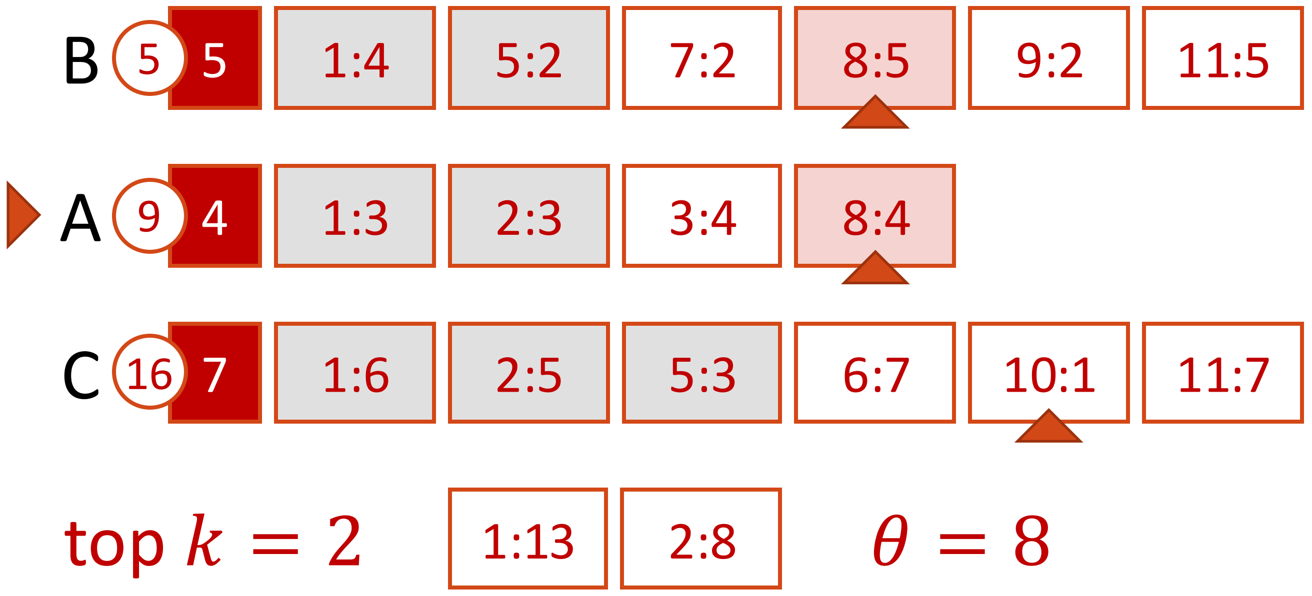


- locate pivot term p
- evaluate pivot document if present in all lists up to p; else align lists up to p

WAND ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ



- locate pivot term p
- evaluate pivot document if present in all lists up to p; else align lists up to p

WAND ($k = 2$)

Each list has an upper bound (aka max-score)

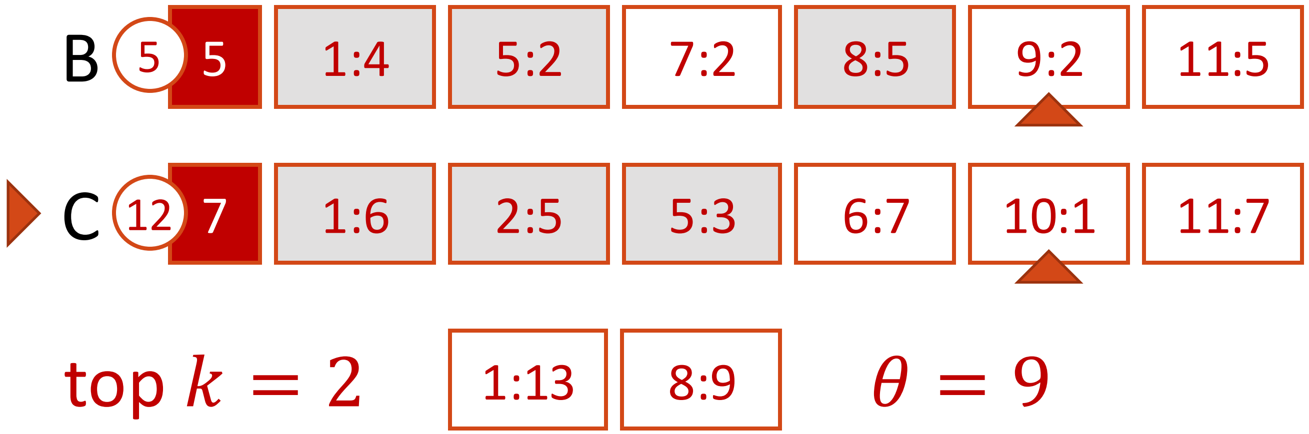
- Top k results have acceptance threshold θ

- locate pivot term p
- evaluate pivot document if present in all lists up to p; else align lists up to p
- re-sort terms by docid

WAND ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

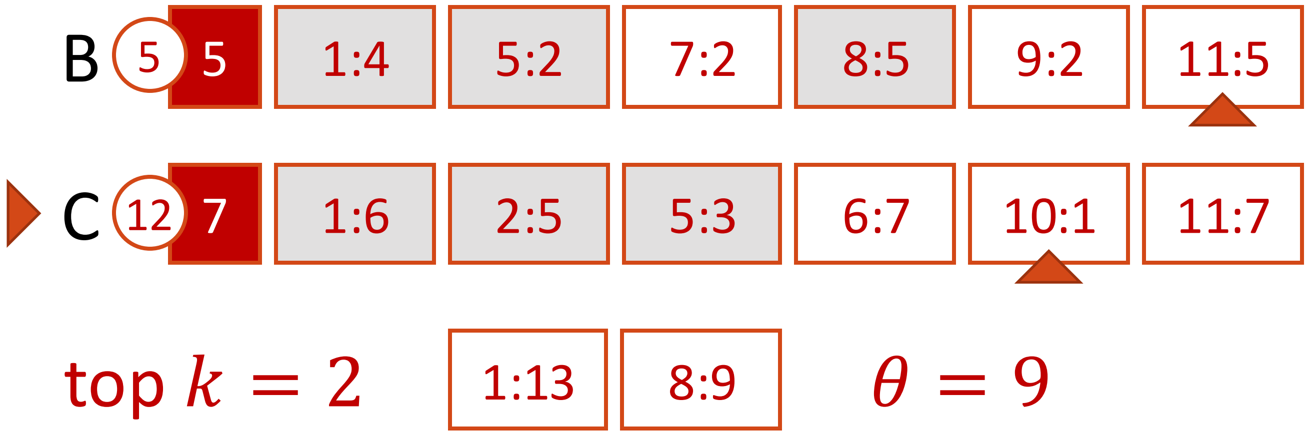


- locate pivot term p
- evaluate pivot document if present in all lists up to p; else align lists up to p

WAND ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

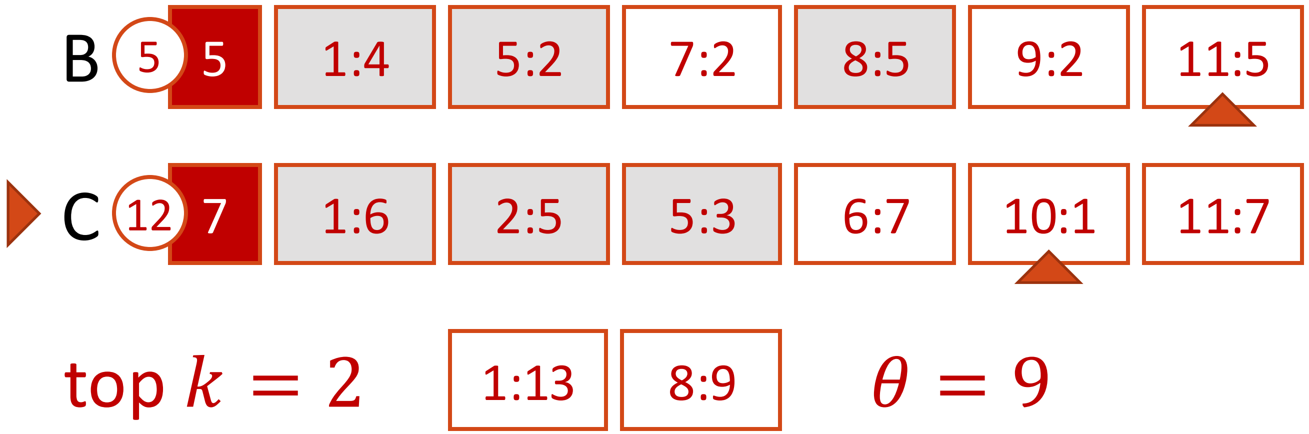


- locate pivot term p
- evaluate pivot document if present in all lists up to p; else align lists up to p
  - list miss on docid 10!

WAND ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

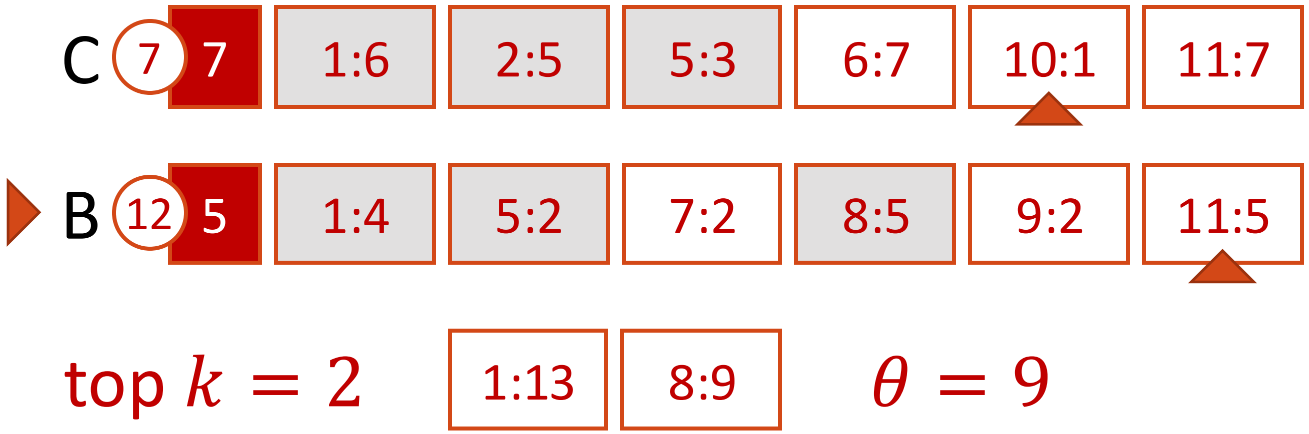


- locate pivot term p
- evaluate pivot document if present in all lists up to p; else align lists up to p
- re-sort terms by docid

WAND ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

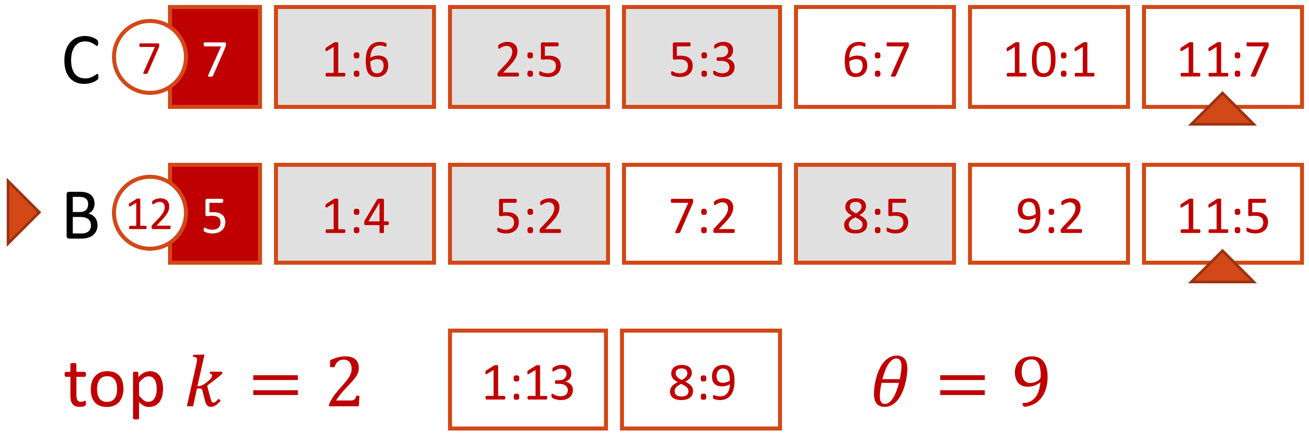


- locate pivot term p
- evaluate pivot document if present in all lists up to p; else align lists up to p

WAND ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

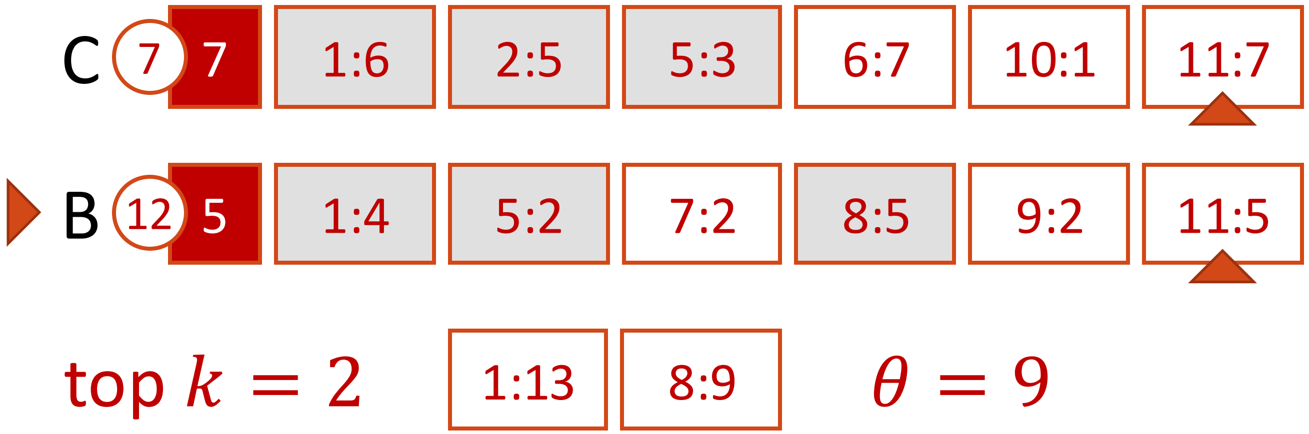


- locate pivot term p
- evaluate pivot document if present in all lists up to p; else align lists up to p
- re-sort terms by docid

WAND ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

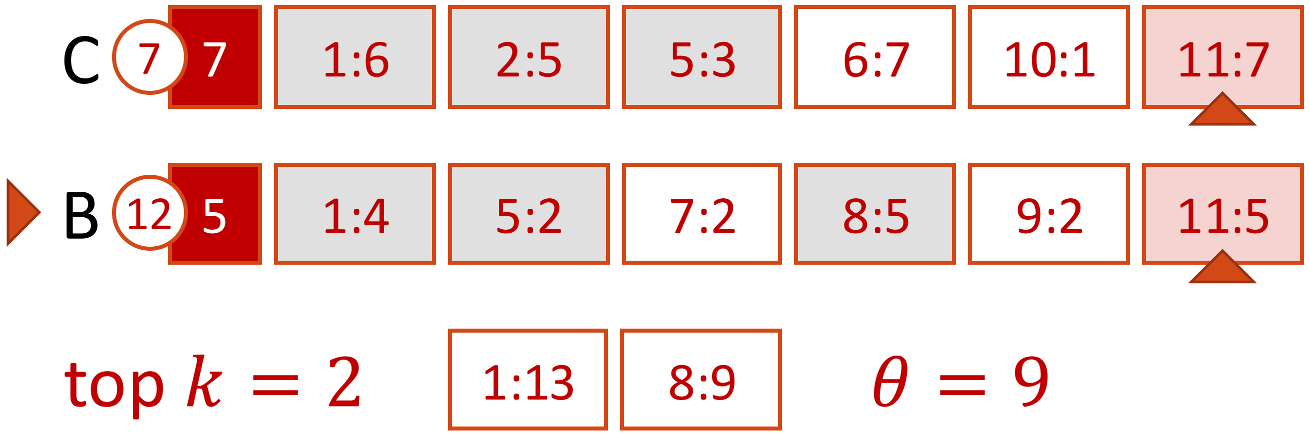


- locate pivot term p
- evaluate pivot document if present in all lists up to p; else align lists up to p

WAND ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ



- locate pivot term p
- evaluate pivot document if present in all lists up to p; else align lists up to p

WAND ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

C	7	7	1:6	2:5	5:3	6:7	10:1	11:7
B	12	5	1:4	5:2	7:2	8:5	9:2	11:5

top $k = 2$ 1:13 11:12 $\theta = 12$

- locate pivot term p
- evaluate pivot document if present in all lists up to p; else align lists up to p
- re-sort terms by docid

WAND ($k = 2$)

Each list has an upper bound (aka max-score)

- Top k results have acceptance threshold θ

A	4	1:3	2:3	3:4	8:4		
C	7	1:6	2:5	5:3	6:7	10:1	11:7
B	5	1:4	5:2	7:2	8:5	9:2	11:5

top $k = 2$ | 1:13 | 11:12 | $\theta = 12$

- 5 / 16 postings skipped
  ≈ 31% savings

In-memory WAND [Fontoura et al., VLDB 2011]

Substantial reduction in processed postings

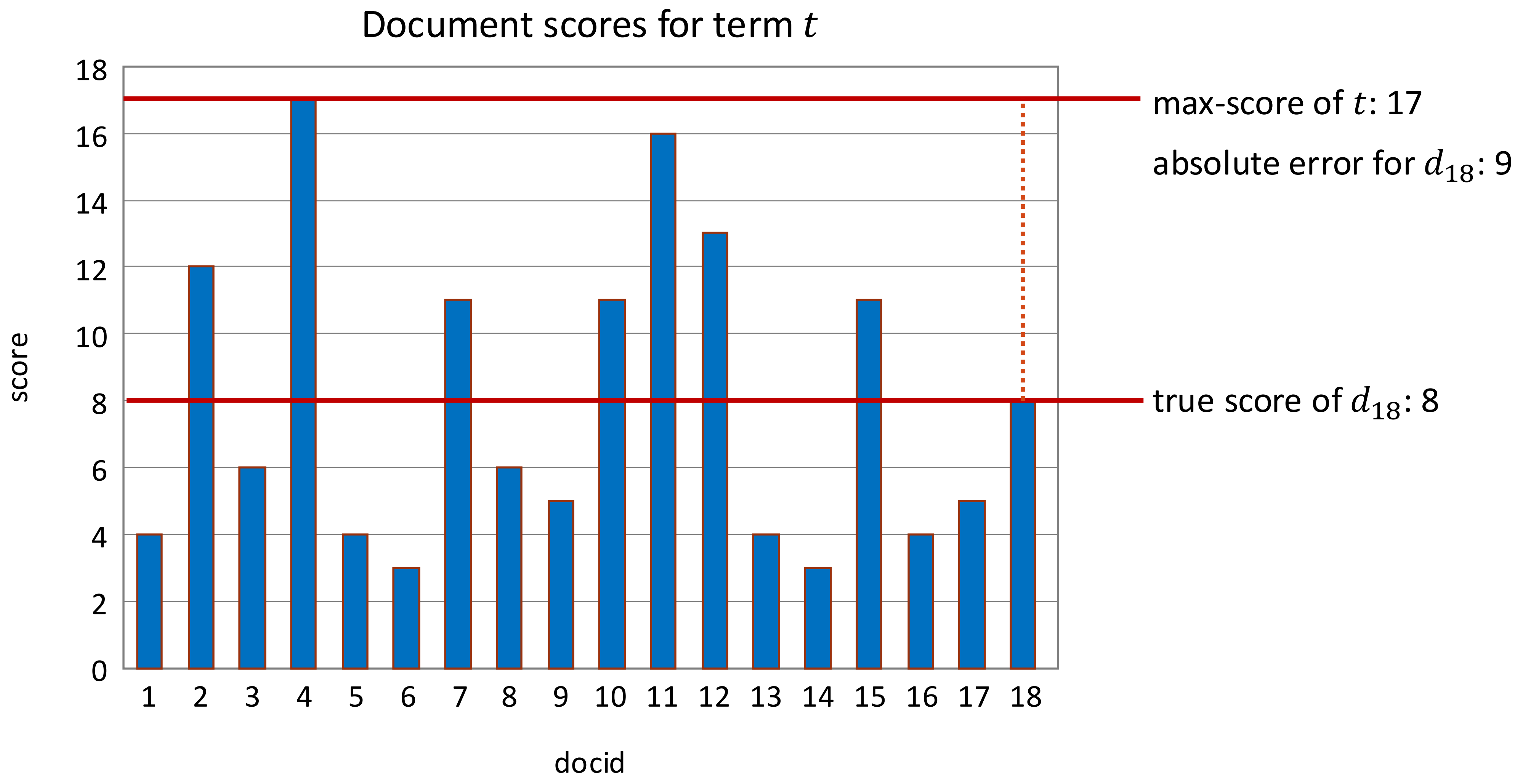
- Particularly efficient for long queries

But latency only improved for disk-based indexes

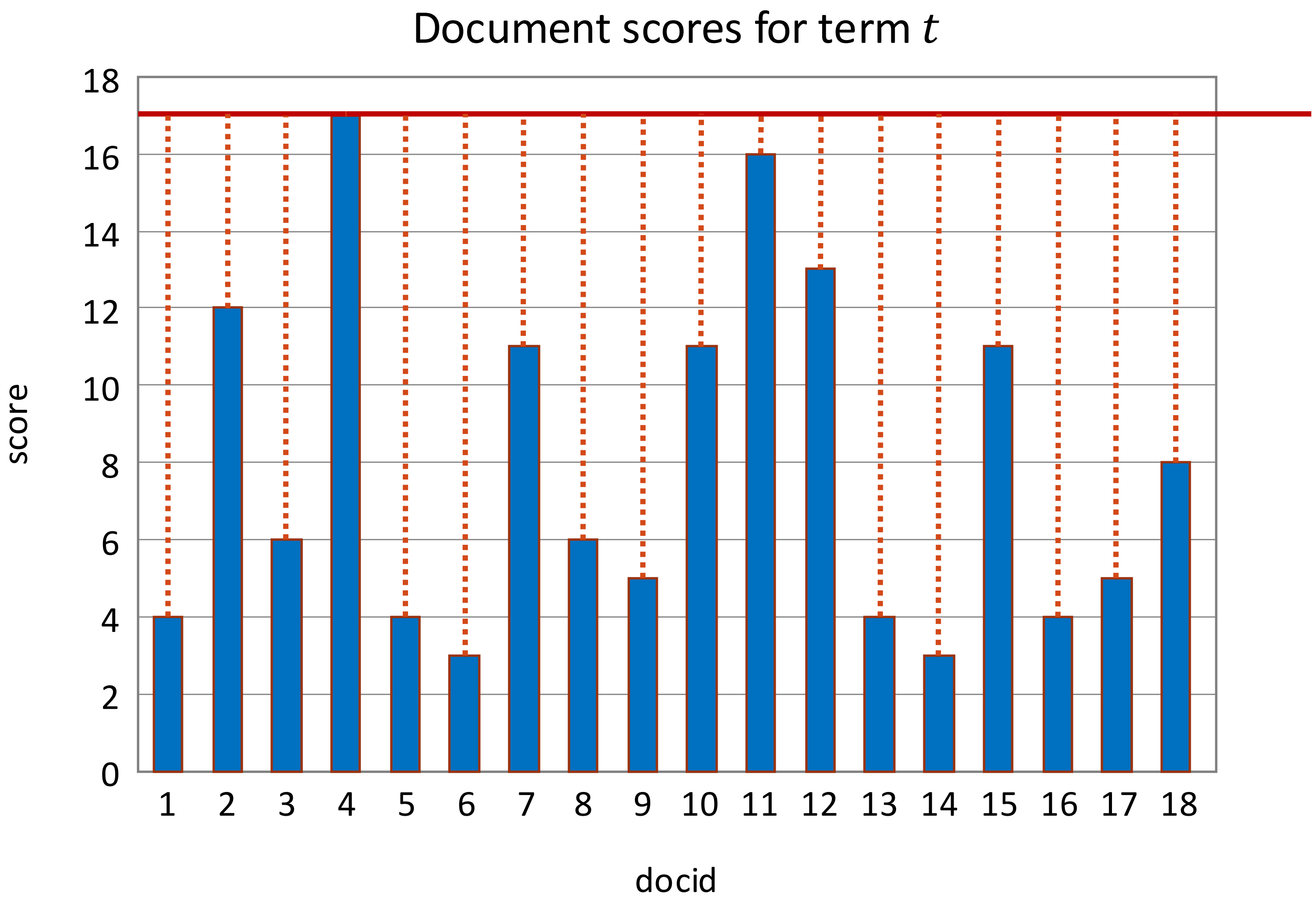
- No memory paging for in-memory indexes!
- Pivot management offsets gains in skipping

Solution: align all cursors before a new re-sort

Block-Max WAND [Ding and Suel, SIGIR 2011]



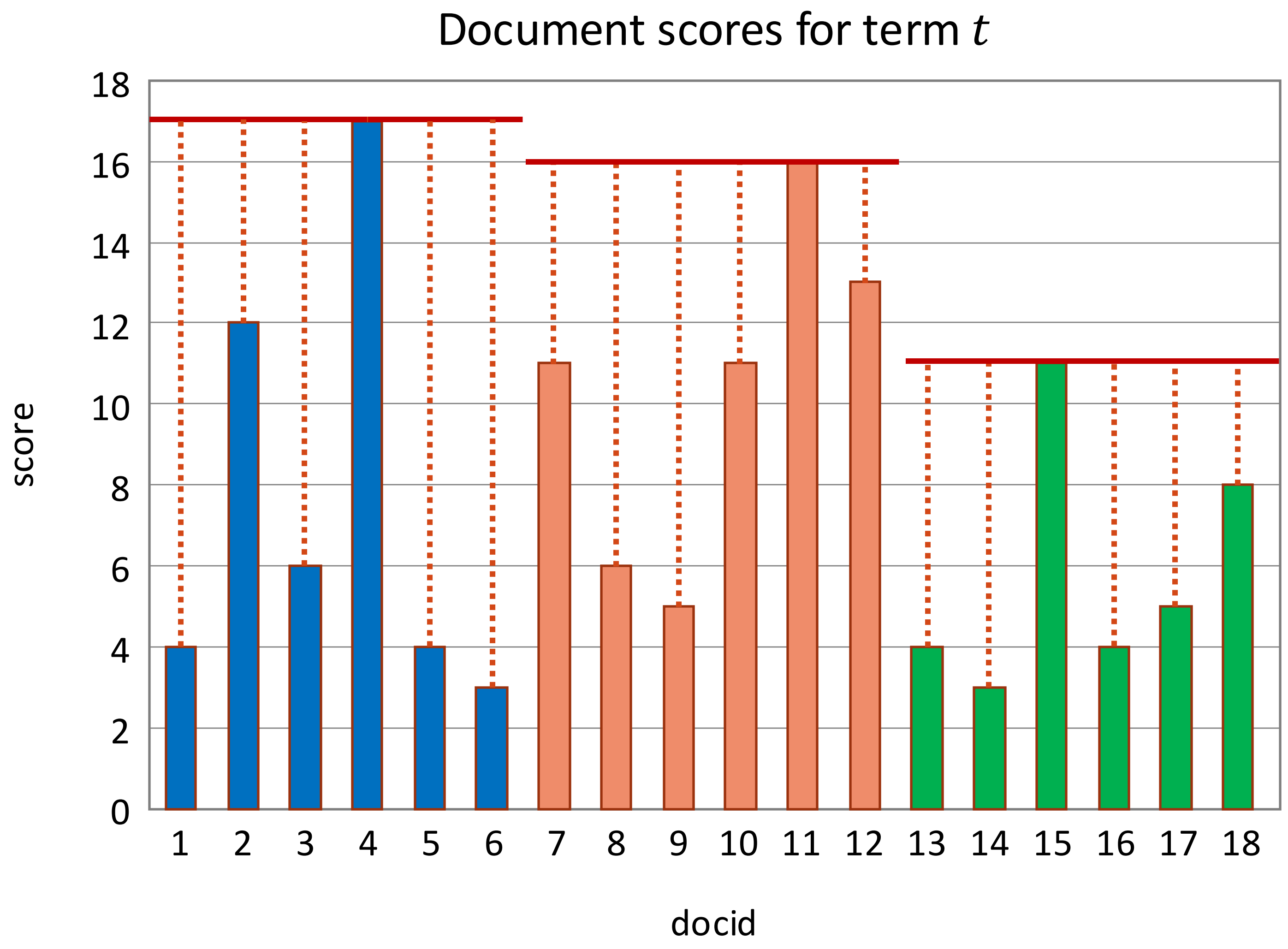
Block-Max WAND [Ding and Suel, SIGIR 2011]



max-score of t: 17

mean absolute error: 9.1

Block-Max WAND [Ding and Suel, SIGIR 2011]

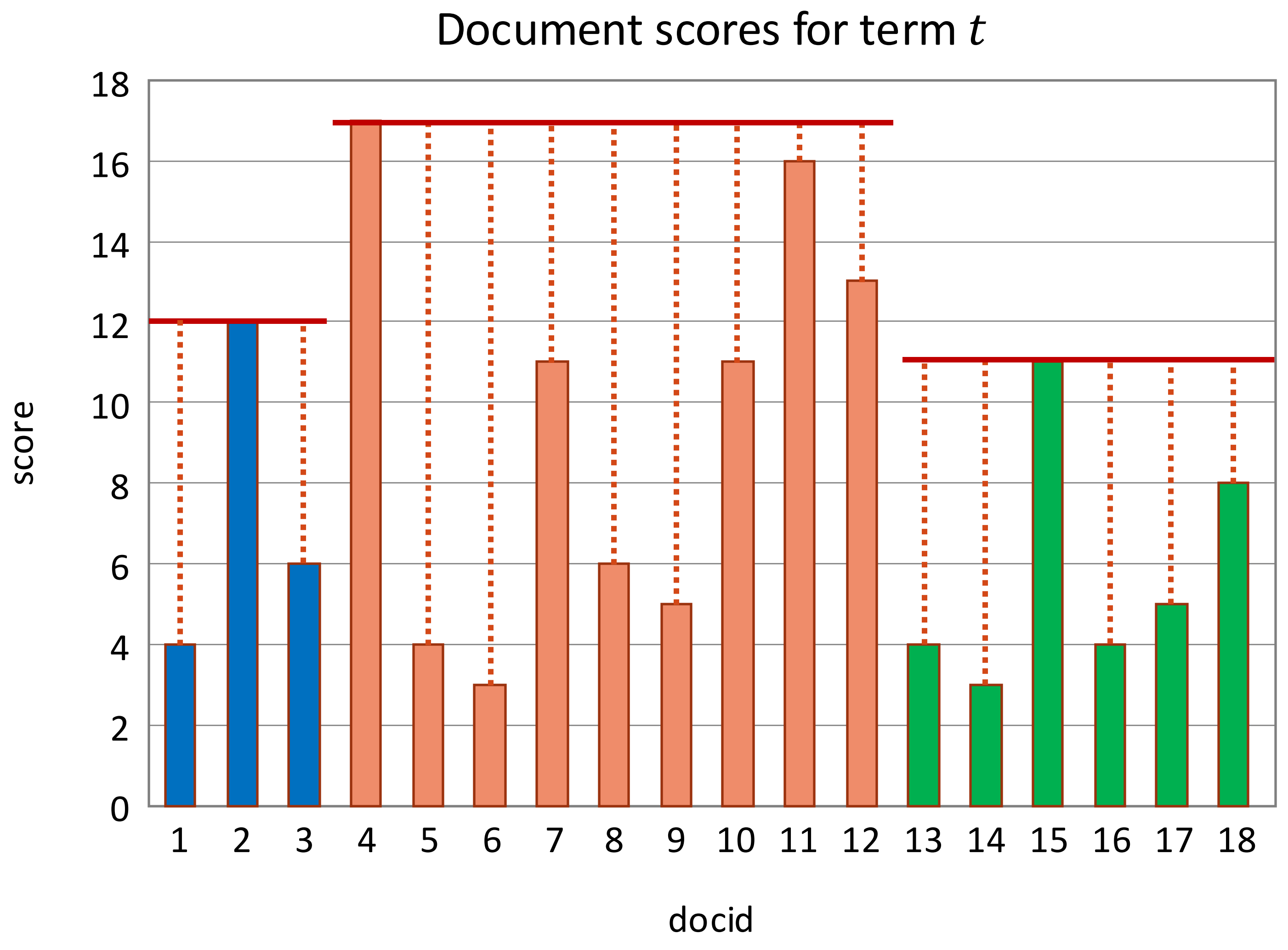


max-score of t: 17

- block max-score of b_1: 17
- block max-score of b_2: 16
- block max-score of b_3: 11

mean absolute error: 6.7

Variable-sized blocks [Mallia et al., SIGIR 2017]

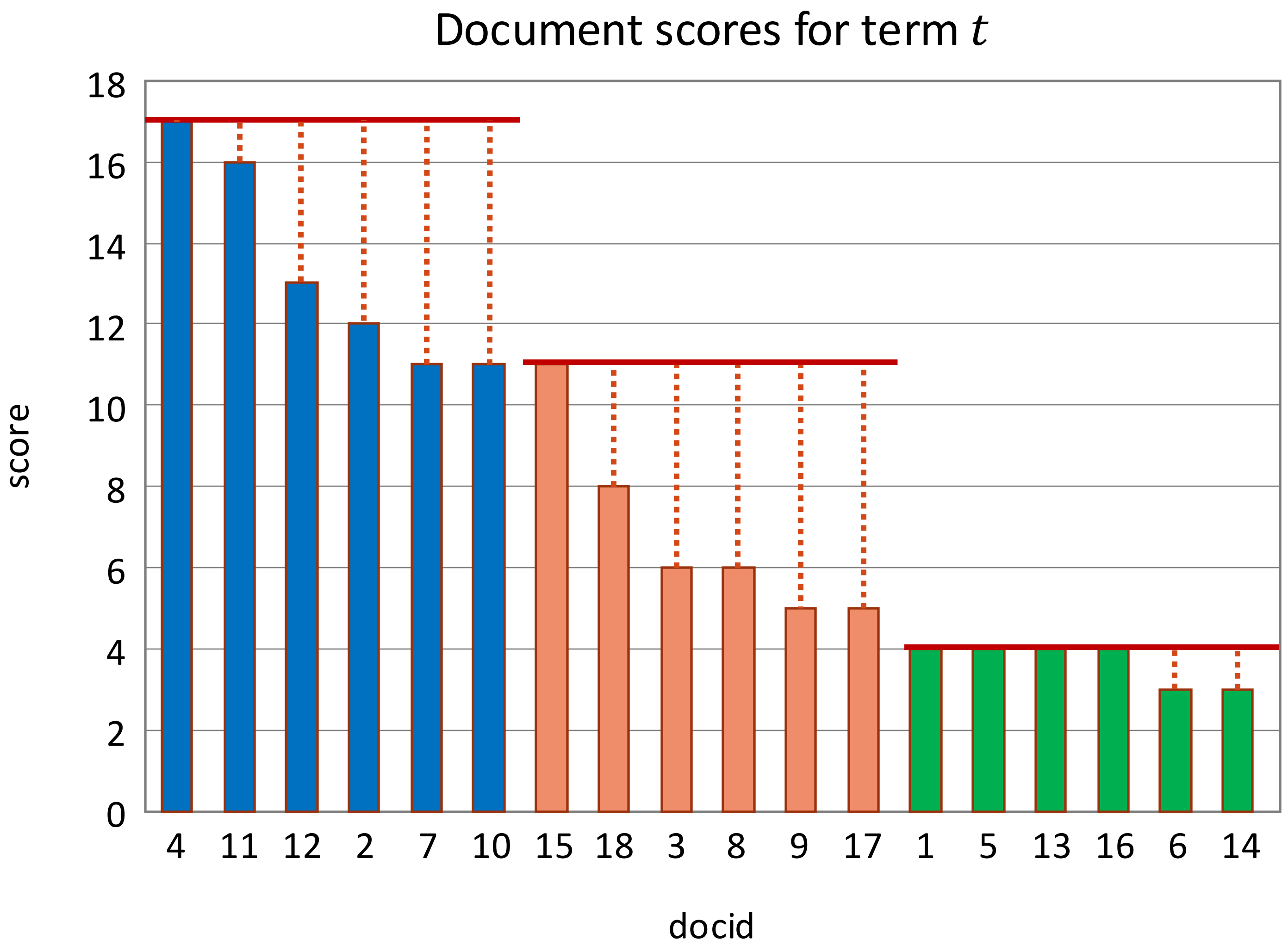


max-score of t: 17

- block max-score of b_1: 12
- block max-score of b_2: 17
- block max-score of b_3: 11

mean absolute error: 6.2

Impact-sorted blocks [Ding and Suel, SIGIR 2011]

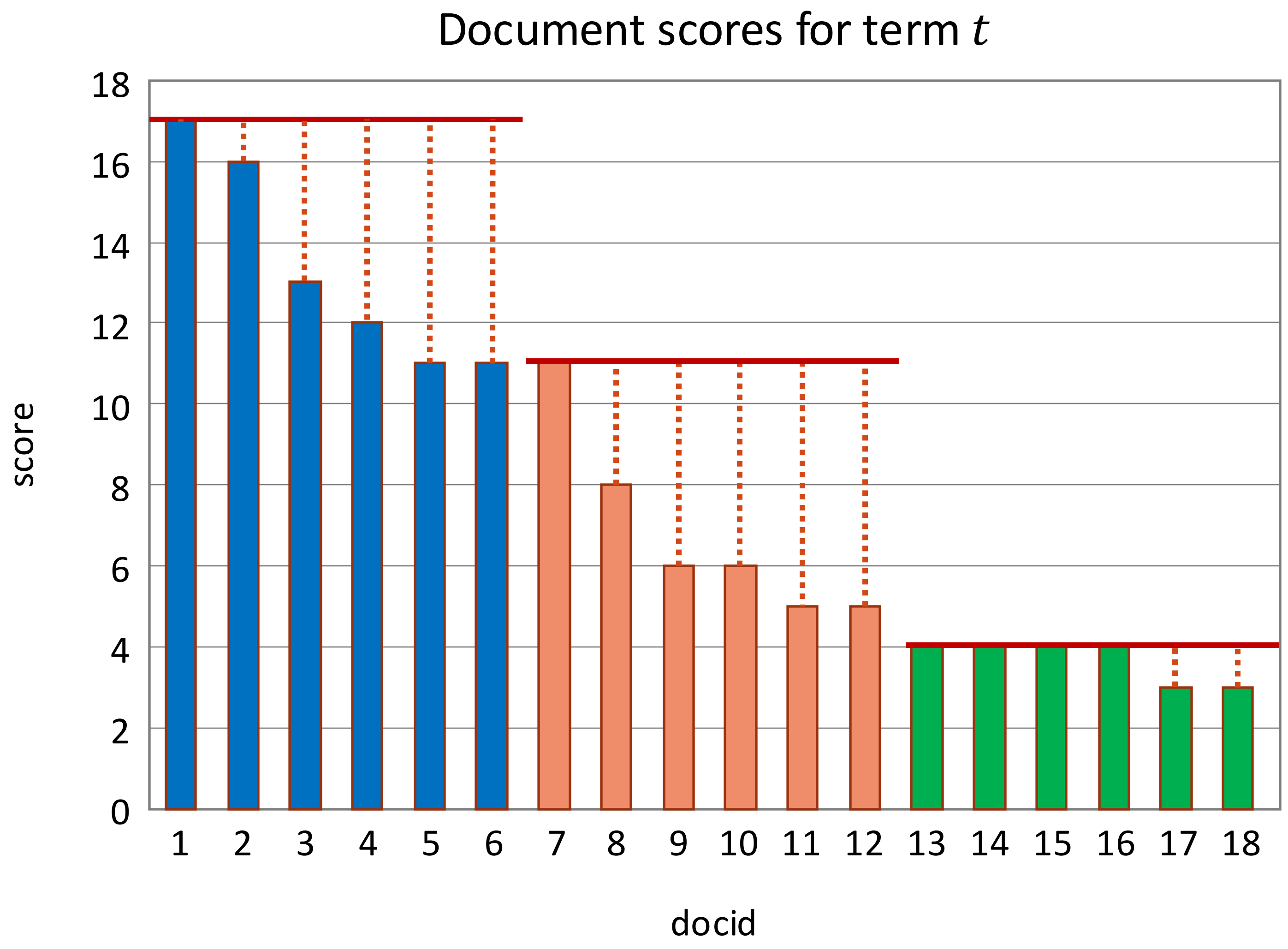


max-score of t: 17

- block max-score of b_1: 17
- block max-score of b_2: 11
- block max-score of b_3: 4

mean absolute error: 2.7

Docid reassignment [Ding and Suel, SIGIR 2011]

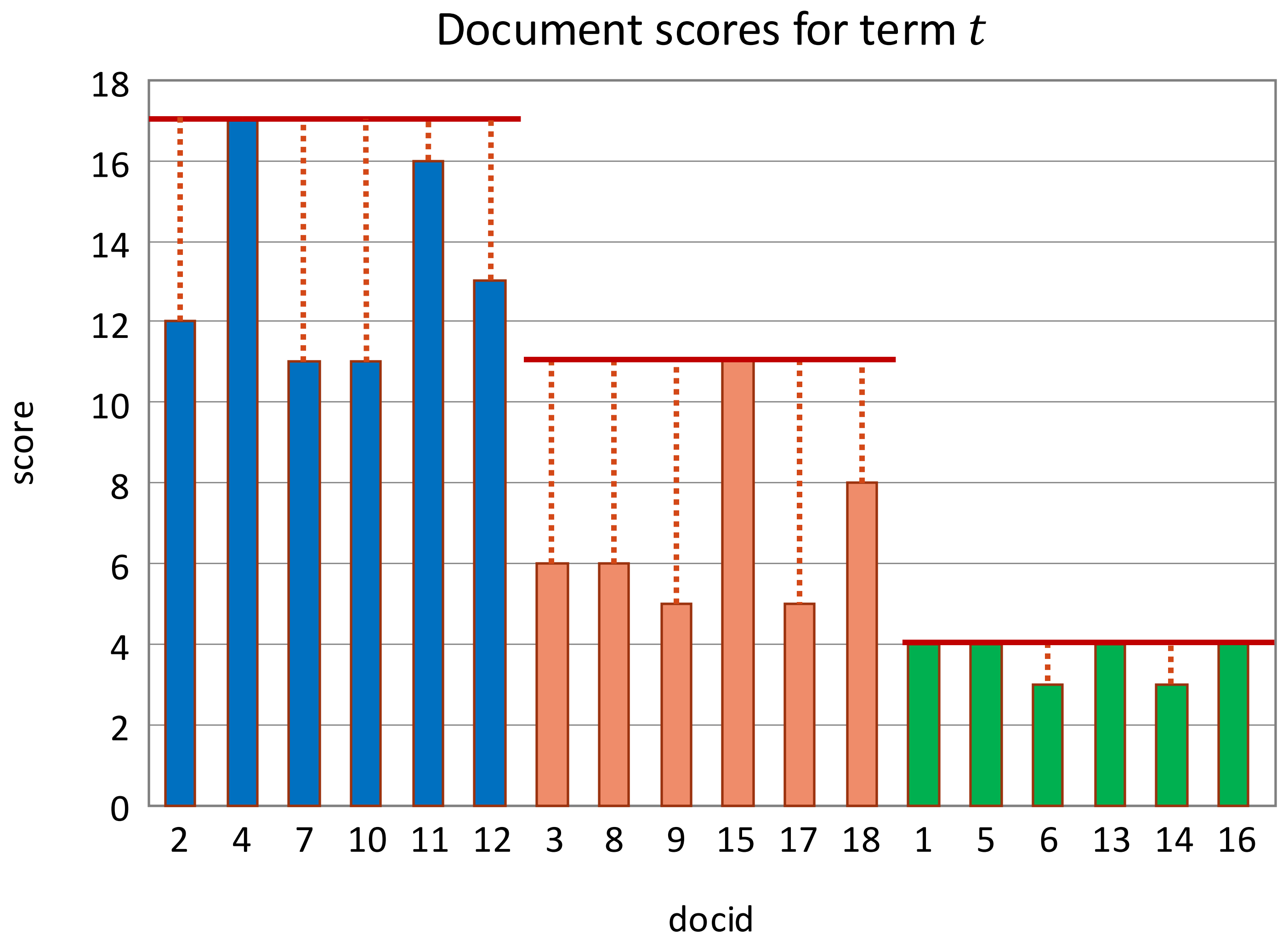


max-score of t: 17

- block max-score of b_1: 17
- block max-score of b_2: 11
- block max-score of b_3: 4

mean absolute error: 2.7

Impact-layered blocks [Ding and Suel, SIGIR 2011]



max-score of t: 17

- block max-score of b_1: 17
- block max-score of b_2: 11
- block max-score of b_3: 4

mean absolute error: 2.7

Summary

Efficient matching for subsecond response times

- Skip postings (or lists) that won't help make the top k

Carefully play with upper bounds and thresholds

- Can be extended with blocks, layers, list orderings

Can always trade-off safety for efficiency

- Anytime ranking for QoS [Lin and Trotman, ICTIR 2015]

References

Scalability Challenges in Web Search Engines, Ch. 4
Cambazoglu and Baeza-Yates, 2015

Efficient Query Processing Infrastructures
Tonellotto and Macdonald, SIGIR 2018

Efficient Query Processing for Scalable Web Search
Tonellotto et al., FnTIR 2018

References

Query evaluation: strategies and optimizations
Turtle and Flood, IP&M 1995

Efficient query eval. using a two-level retrieval process
Broder et al., CIKM 2003

Faster top-k document retr. using block-max indexes
Ding and Suel, SIGIR 2011

UFMG UNIVERSIDADE FEDERAL DE MINAS GERAIS

Coming next...

Vector Space Models

Rodrygo L. T. Santos
rodrygo@dcc.ufmg.br