PROCESSAMENTO DE CONSULTAS

Processamento de Consultas

- Vocabulário e listas invertidas preferencialmente em memória RAM. Eventualmente em disco;
- Idf dos termos e normas dos documentos são pré-computados;
- Listas hoje em dia são guardadas em memória, mas podem ser guardadas no disco, especialmente com SSDs e novas tecnologias de armazenamento. Memória poderia ser um cache para um índice maior em disco ?? (assunto pra um PIBIC ou um mestrado...)

Processamento de Consultas

- Simplificações que aceleram o processamento:
  - Documentos que não possuem os termos das consultas têm similaridade igual a zero;
  - Score é calculado de maneira que se possa ler as listas invertidas sequencialmente durante o cálculo;

Processamento de Consultas

- Pode ser feito de duas formas:
  - Termo a Termo: term at a time query processing
  - **Documento a Documento**
  - **Atualmente, melhores algoritmos são baseados em documento a documento (document at a time query processing)**
  - Alguns métodos de fato são métodos híbridos)

PROCESSAMENTO TERMO A TERMO

- Processa uma lista invertida de cada vez, calculando similaridades parciais
- Muito bom para realização de podas em consultas durante o cálculo dos scores

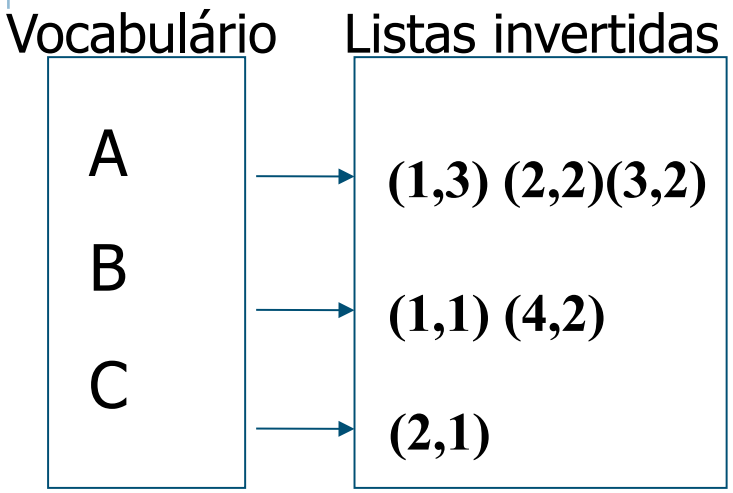
Exemplo com vetorial

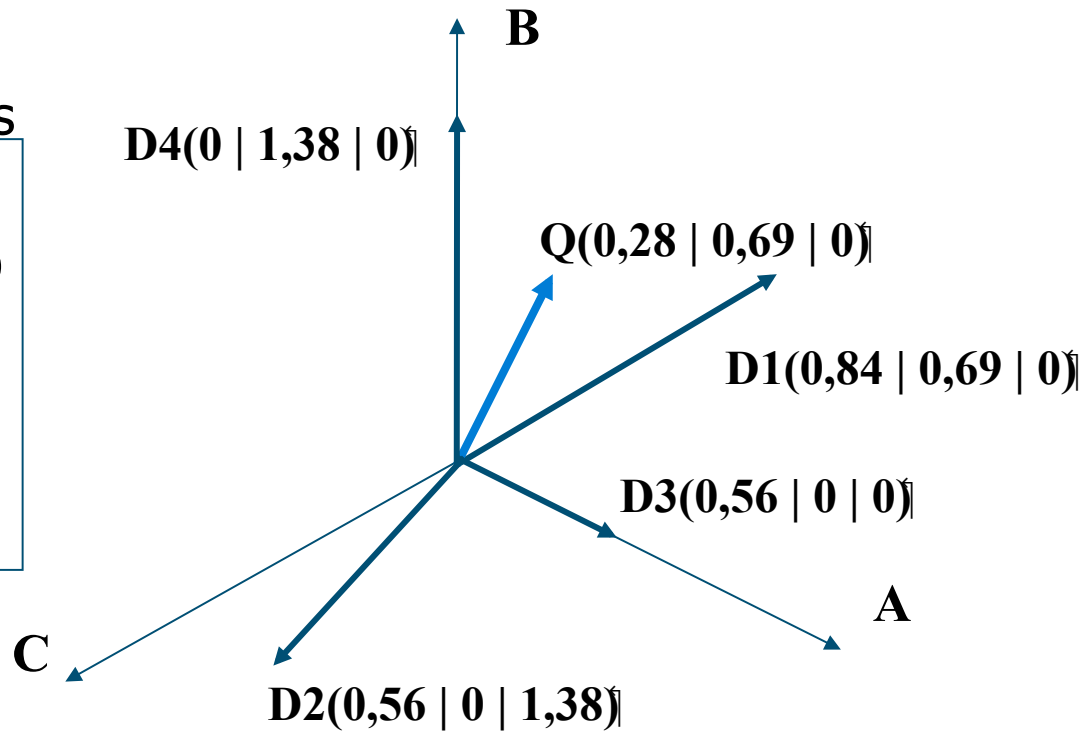
Mesmas ideias podem ser usadas com outros modelos de RI

Estruturas de Dados: Arquivo Invertido

Q = "A B"

D1	A A A B
D2	A A C
D3	A A
D4	B B



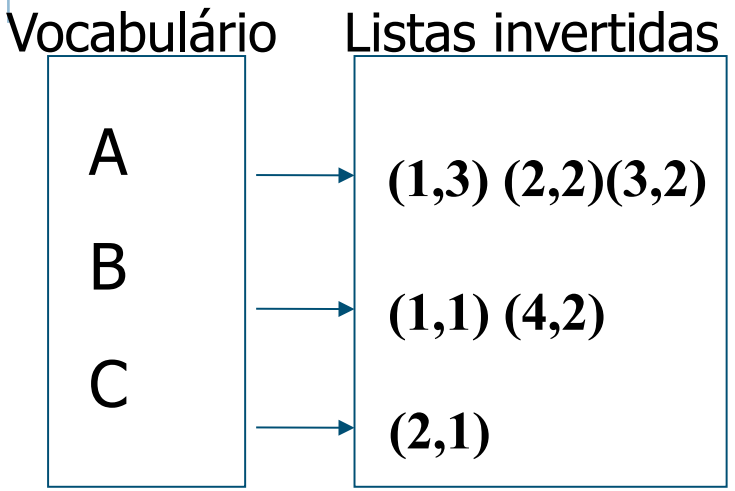


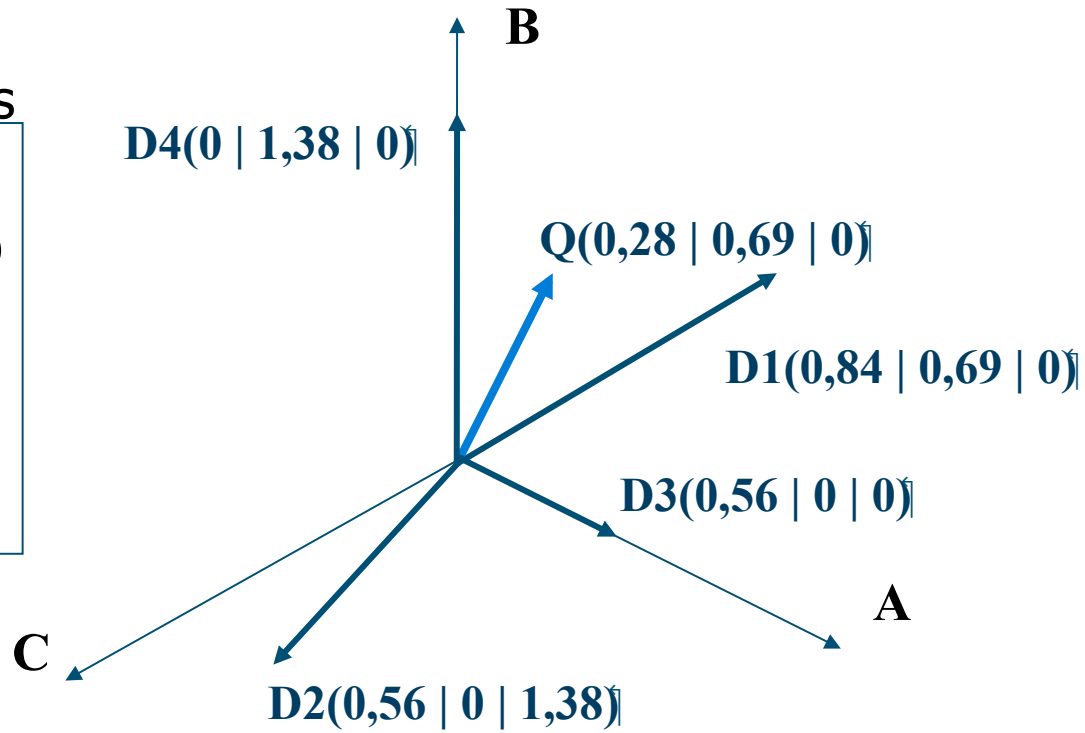
Acumuladores para cálculo da similaridade parcial

Acumuladores:

D1	D2	D3	D4
0	0	0	0

$$sim_{parc}(q,d,t) = w(d,t) \times w(q,t)$$



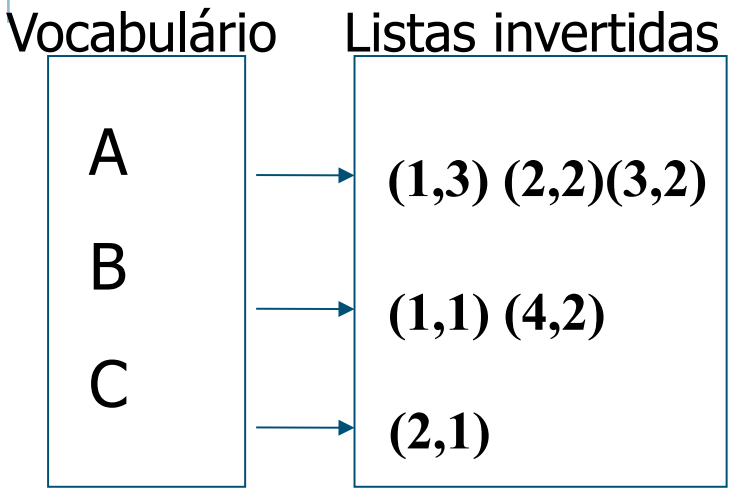


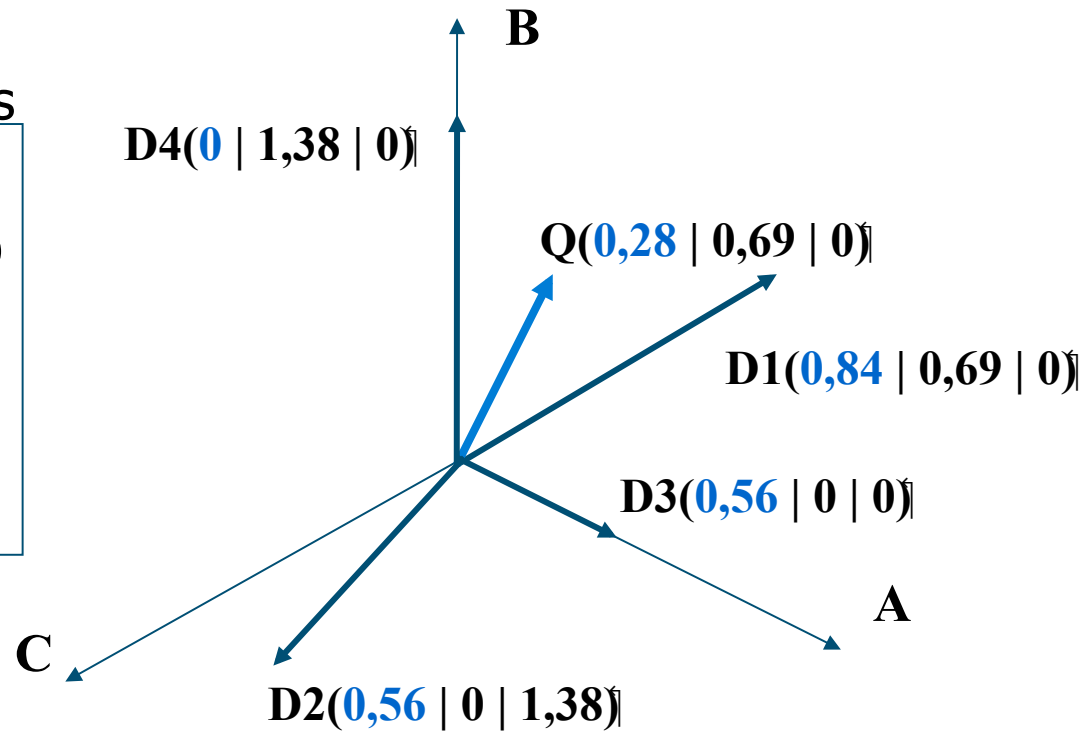
Acumuladores para cálculo da similaridade parcial

Acumuladores:

D1	D2	D3	D4
0,24	0,16	0,16	0

$$sim_{parc}(Q, D1, A) = w(D1, A) \times w(Q, A) = 0{,}84 \times 0{,}28 = 0{,}24$$

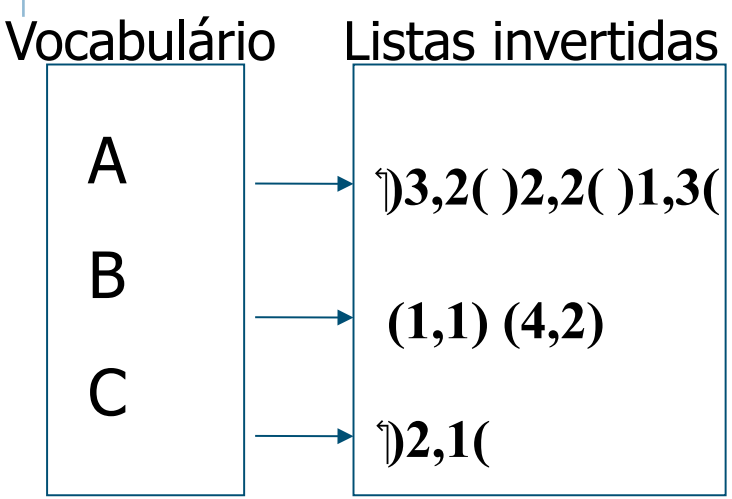


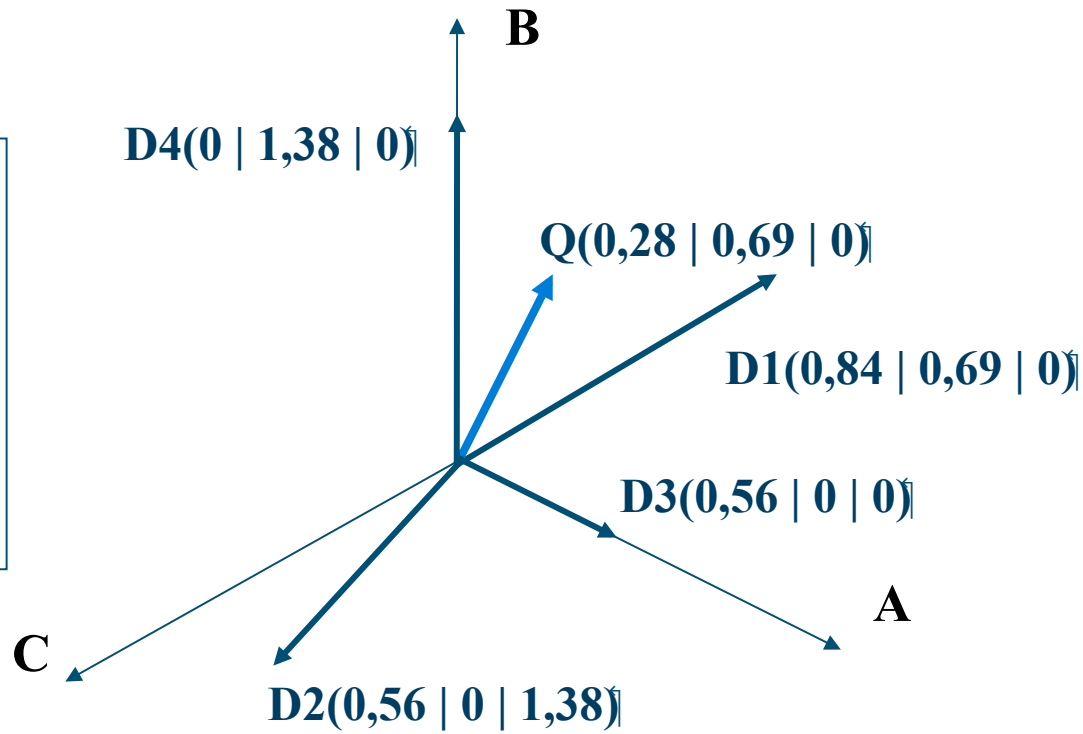


Acumuladores para cálculo da similaridade parcial

Acumuladores:

D1	D2	D3	D4
0,71	0,16	0,16	0,95





Acumuladores para cálculo da similaridade parcial

Acumuladores:

D1	D2	D3	D4
0,71	0,16	0,16	0,95

$$sim(d,q) = \cos\theta = \frac{\sum_{i=1}^{t} w(i,d) \times w(i,q)}{\sqrt{\sum (w(i,d))^2} \times \sqrt{\sum (w(i,q))^2}}$$

Norma do documento

Norma da consulta

Acumuladores para cálculo da similaridade parcial

$$norma(d1) = \sqrt{\sum_{i \in V} (w(i,d1))^2} = \sqrt{(0,84)^2 + (0,69)^2} = 1,08$$

$$norma(d2) = 1,49$$

$$norma(d3) = 0,56$$

$$norma(d4) = 1,38$$

Acumuladores para cálculo da similaridade parcial

Acumuladores:

D1	D2	D3	D4
0,71	0,16	0,16	0,95

$$sim(d1,q) = \frac{Acum(d1)}{\|\vec{d}1\| \times \|\vec{q}\|} = \frac{0,71}{1,08 \times \|\vec{q}\|} = \frac{0,66}{\|\vec{q}\|}$$

$norma(d1) = 1,08$

$$sim(d2,q) = \frac{0,16}{1,49 \times \|\vec{q}\|} = \frac{0,107}{\|\vec{q}\|}$$

$norma(d2) = 1,49$

$$sim(d3,q) = \frac{0,16}{0,56 \times \|\vec{q}\|} = \frac{0,28}{\|\vec{q}\|}$$

$norma(d3) = 0,56$

$$sim(d4,q) = \frac{0,95}{1,38 \times \|\vec{q}\|} = \frac{0,69}{\|\vec{q}\|}$$

$norma(d4) = 1,38$

Algoritmo

1. Para cada documento d na coleção, criar $A\{d\} = 0$.
2. Para cada termo t na consulta,
   (a) Recuperar a lista invertida para o termo t do disco.
   (b) Para cada entrada $\langle d, f(d,t) \rangle$ na lista invertida,

$$A\{d\} = A\{d\} + sim(q,d,t)\,\$.$$

3. Dividir cada acumulador $A\{d\} \neq 0$ pela norma do documento $|\vec{d}|$.
4. Identificar os k valores mais altos de acumuladores, onde k é o número de documentos retornados para o usuário.

Figura 3: Algoritmo para o cálculo da similaridade no modelo vetorial.

Dúvidas?

Processamento documento a documento

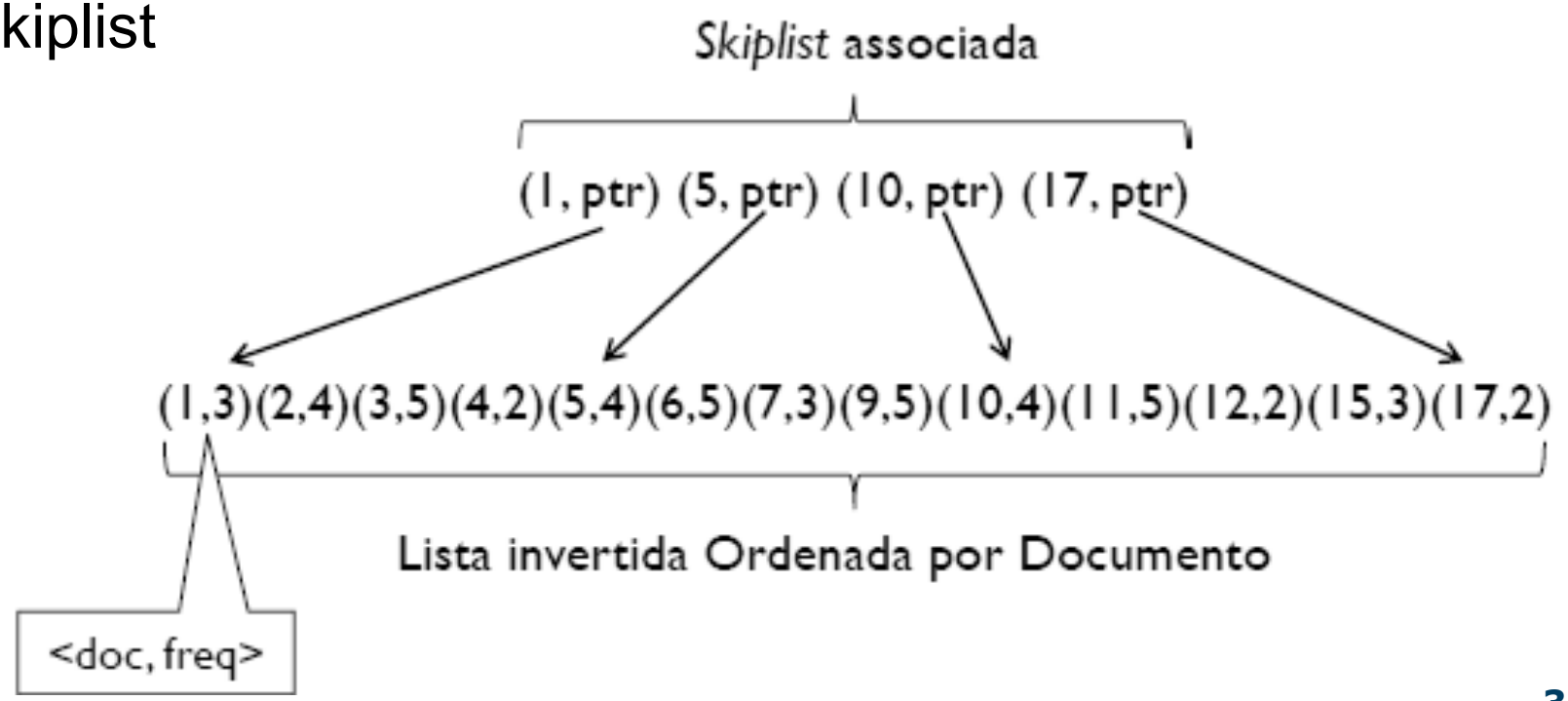
- Listas invertidas são lidas simultaneamente
- Buffers em memória são usados para acelerar leitura das listas
- As listas são ordenadas necessariamente por documento
- Um heap é usado para processar as entradas das listas, como em um processo de merge
- Reduz custo com acumuladores

Antes de falar do processamento documento a documento…

- Vamos estudar alguns conceitos básicos e entender como as listas invertidas são efetivamente armazenadas (em memória, ou em disco)

Processamento Doc a Doc

- **Índice invertido**
- Vocabulário
- Lista invertida
- Skiplist



SkipLists

- Listas invertidas são divididas em blocos, tipicamente de tamanho fixo, 128 documentos
- Skiplist indica o 1o e o último documento de cada bloco e onde o mesmo começa e onde termina no índice
- Também contém informação sobre valores de scores de elementos dentro do bloco que podem ser usadas para descartar todo o bloco durante o processamento

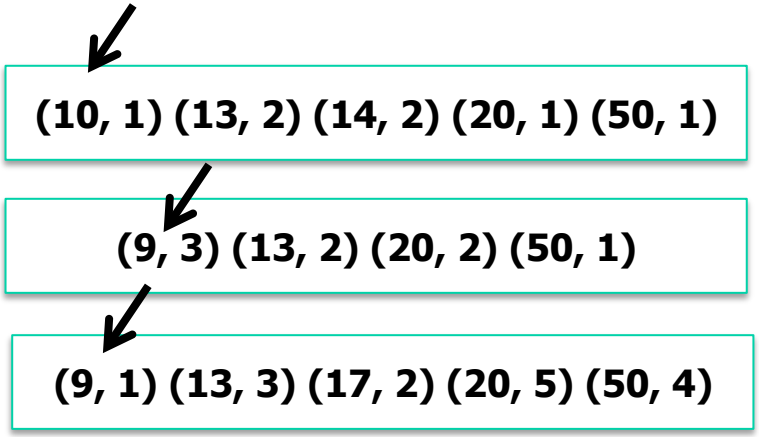
Algoritmos de processamento Documento a Documento

Alguns trabalhos importantes

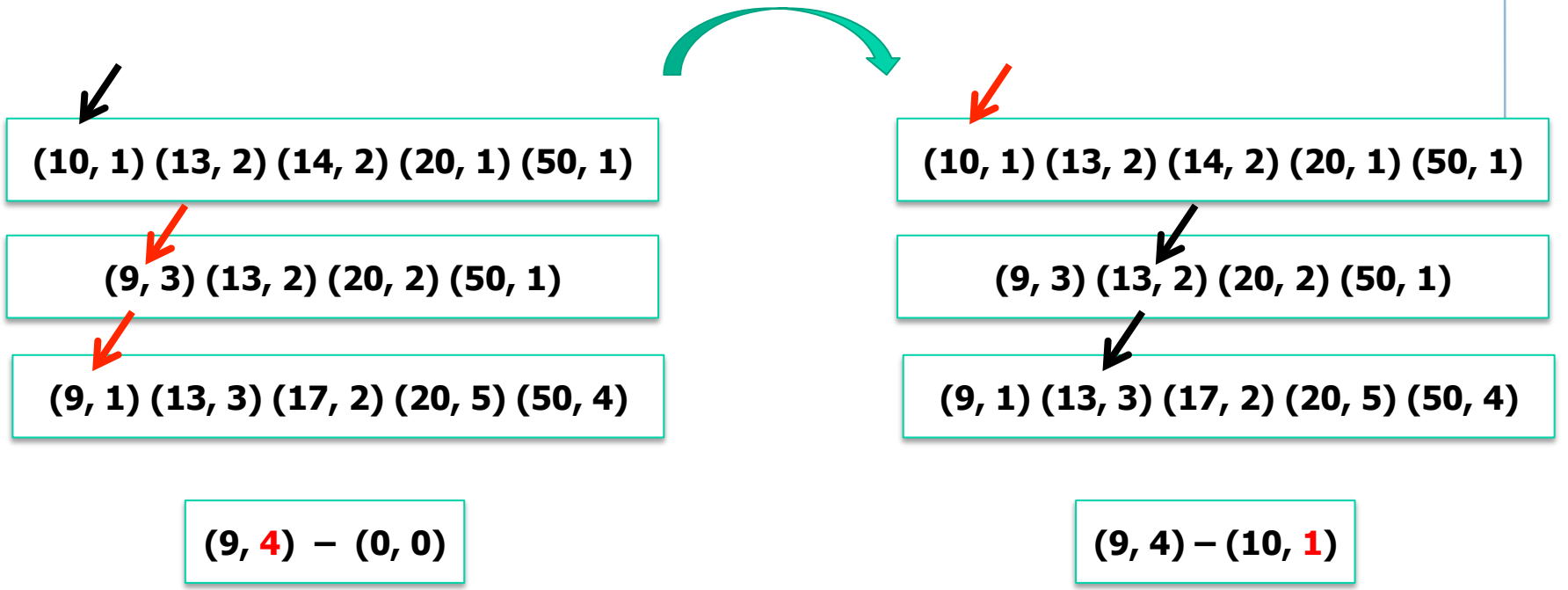
- DaD - Documento a Documento
- WAND [Broder, CIKM, 2003]
- BMW [Ding e Suel, SIGIR, 2011]
- MBMW [Ding e Suel, SIGIR, 2011]
- BMW-CS [Rossi el al, SIGIR, 2013]
- BMW-CSP [Daoud et al, IP&M 2016]
- Waves [Daoud et al, IRJ 2017]

DaD – Documento a Documento

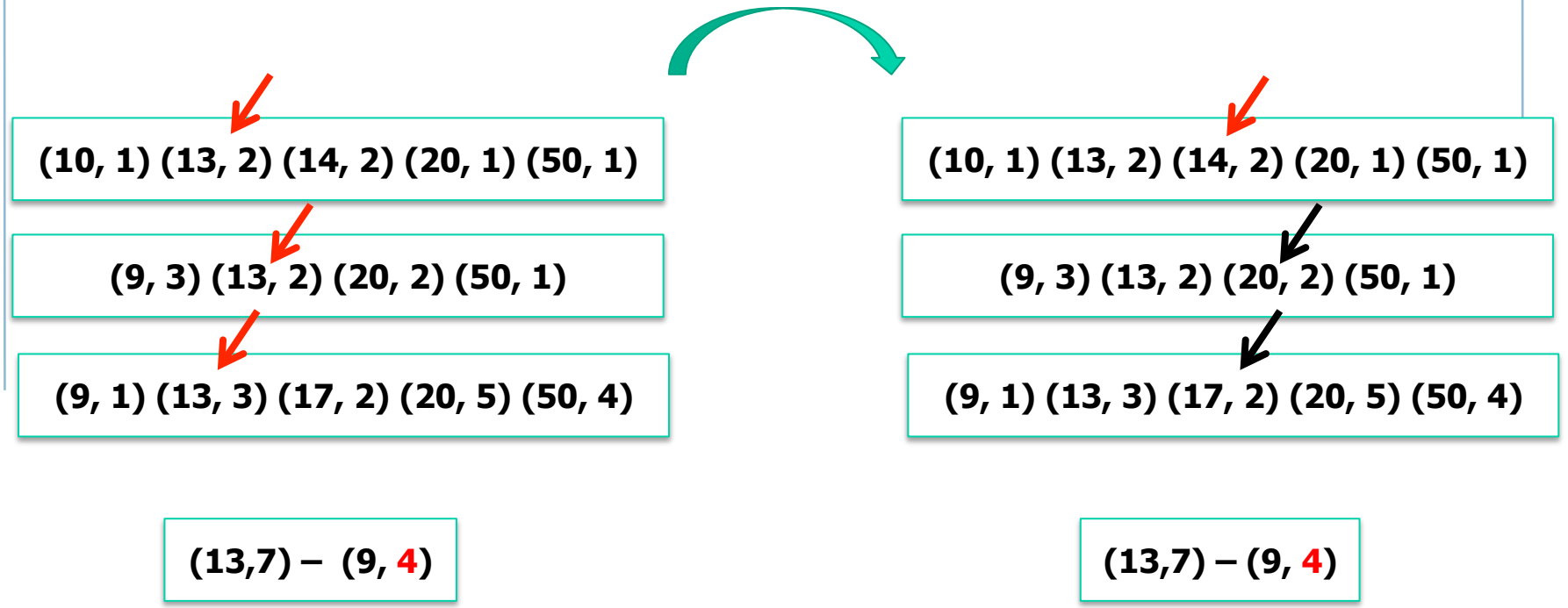
- **Heap** armazena os top-k documentos de maior *score* já avaliados.
- **Limiar de Poda/Corte** (*threshold*): Menor *score* do *heap*



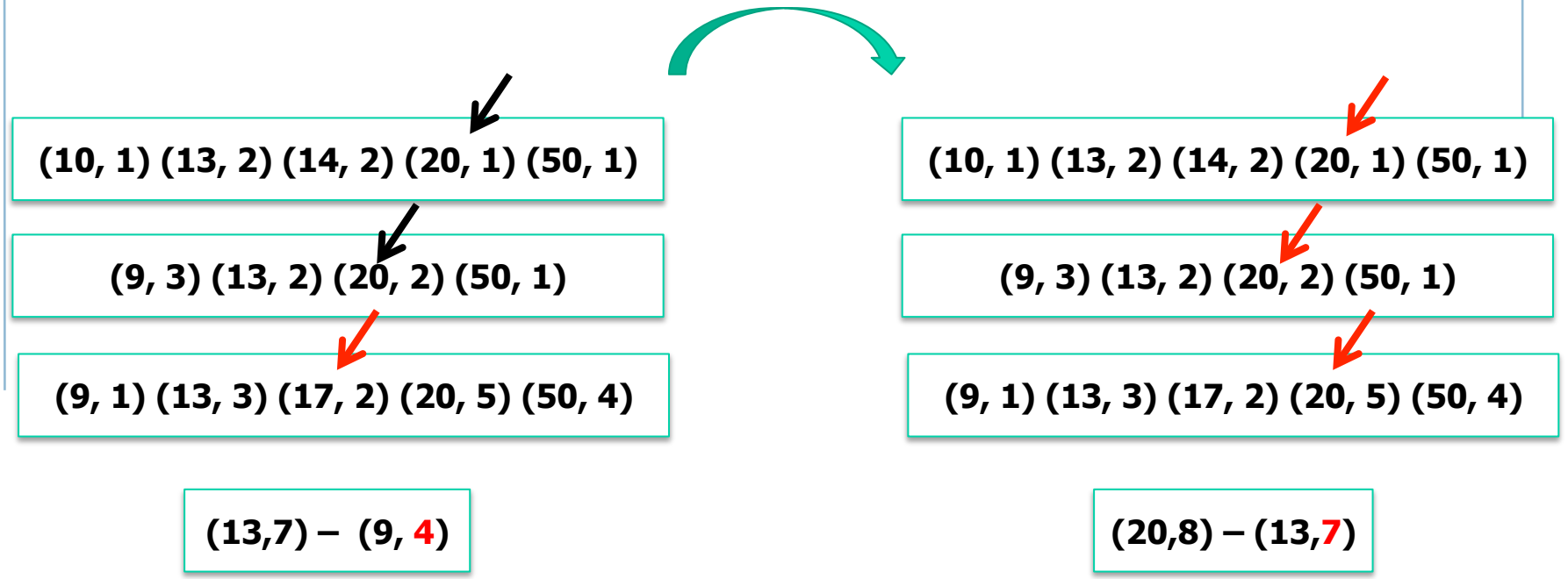
DaD – Documento a Documento



DaD – Documento a Documento



DaD – Documento a Documento



WAND

Consulta

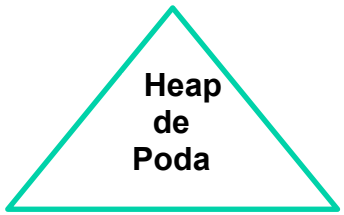
Termos **Listas Invertidas**

max score = 3

(Doc1, F1) (Doc2, F2) ... (Dock, Fk)

score1 **score2** **scorek**

max score = 2



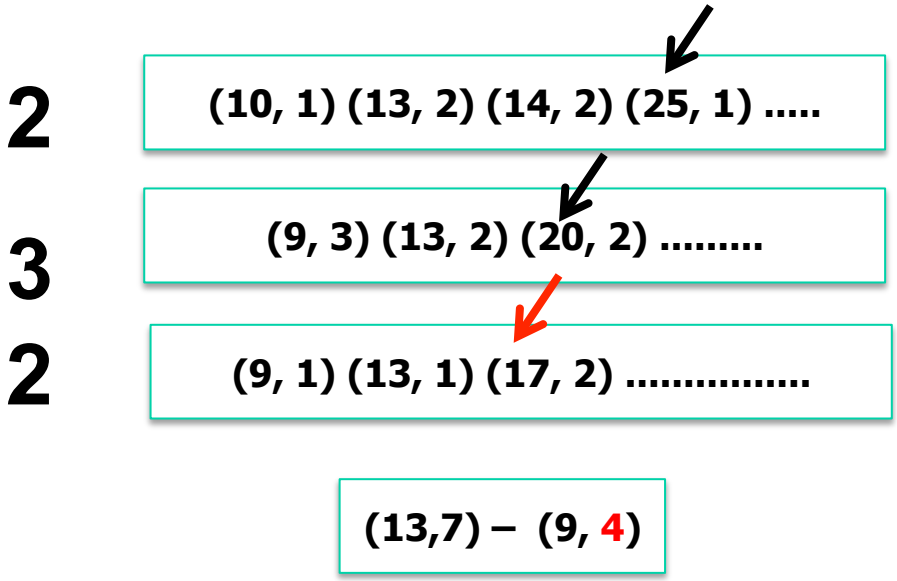
Limiar de descarte

- ✓ **Max_score**
- ✓ **Score**

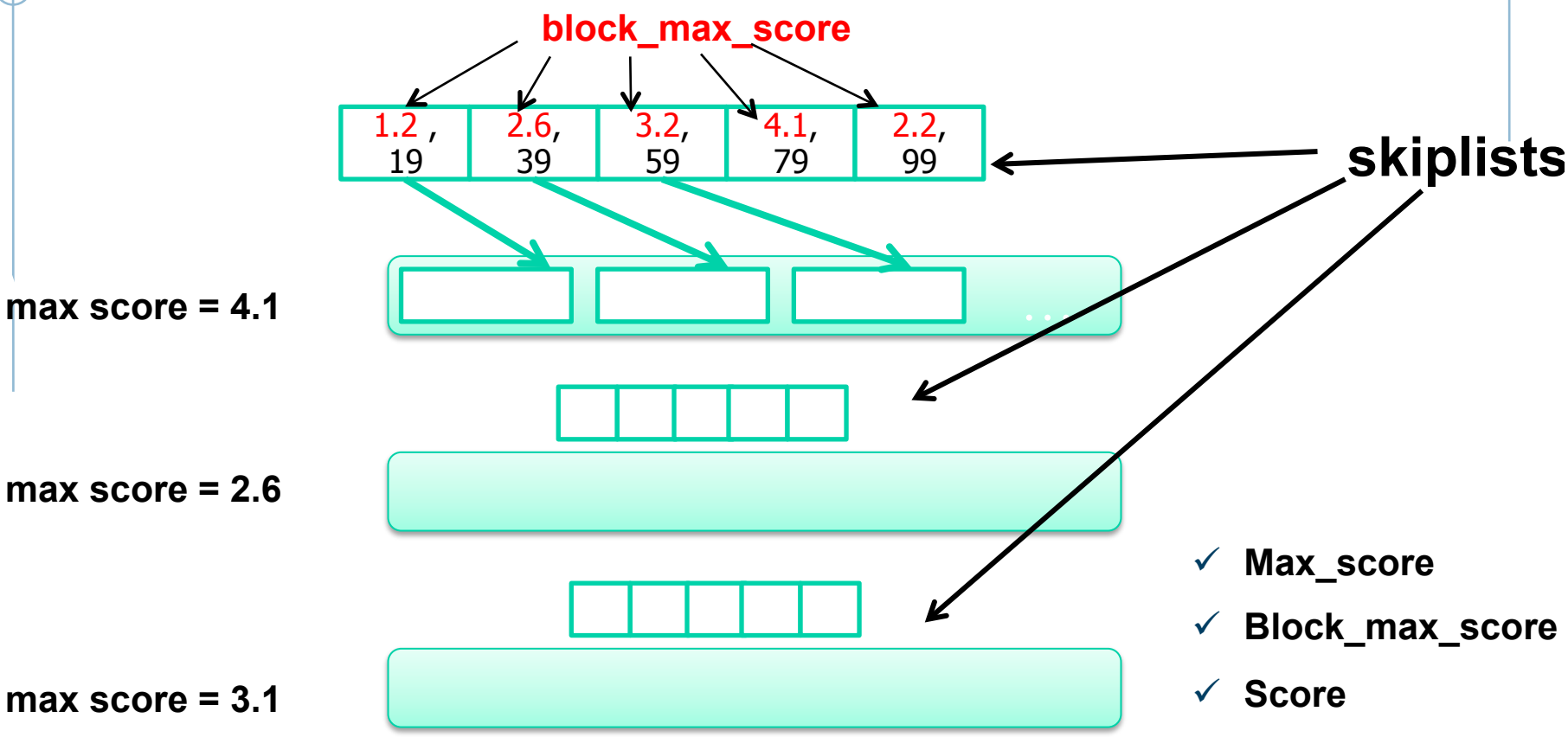
WAND

- Max Score: Maior score dentre os encontrados em uma lista de um dado termo.
- Usado para estimar o ganho caso um documento considerado possa entrar na resposta.

Ex. WAND

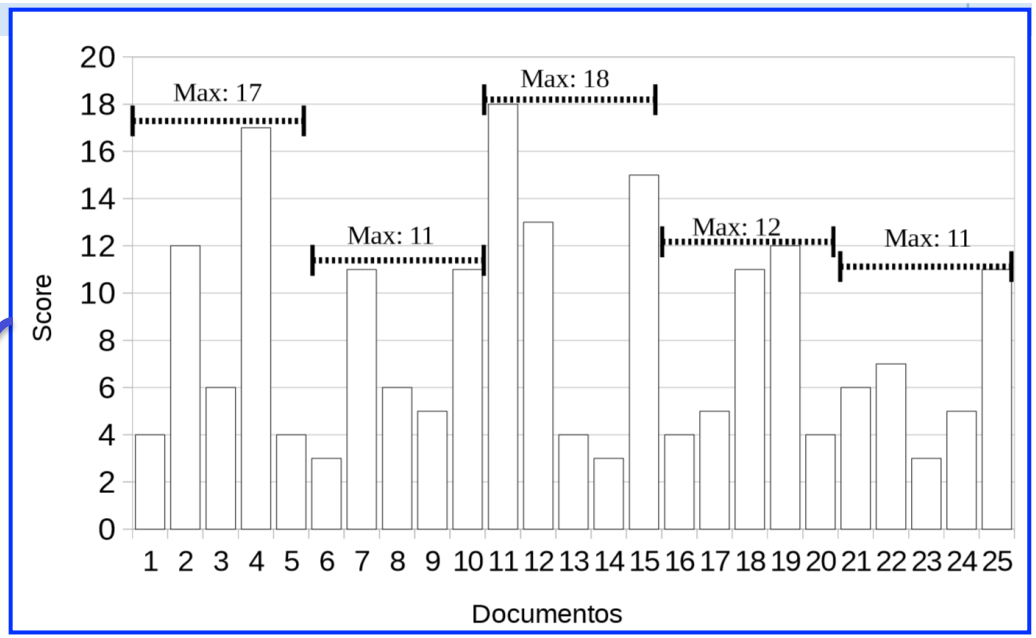


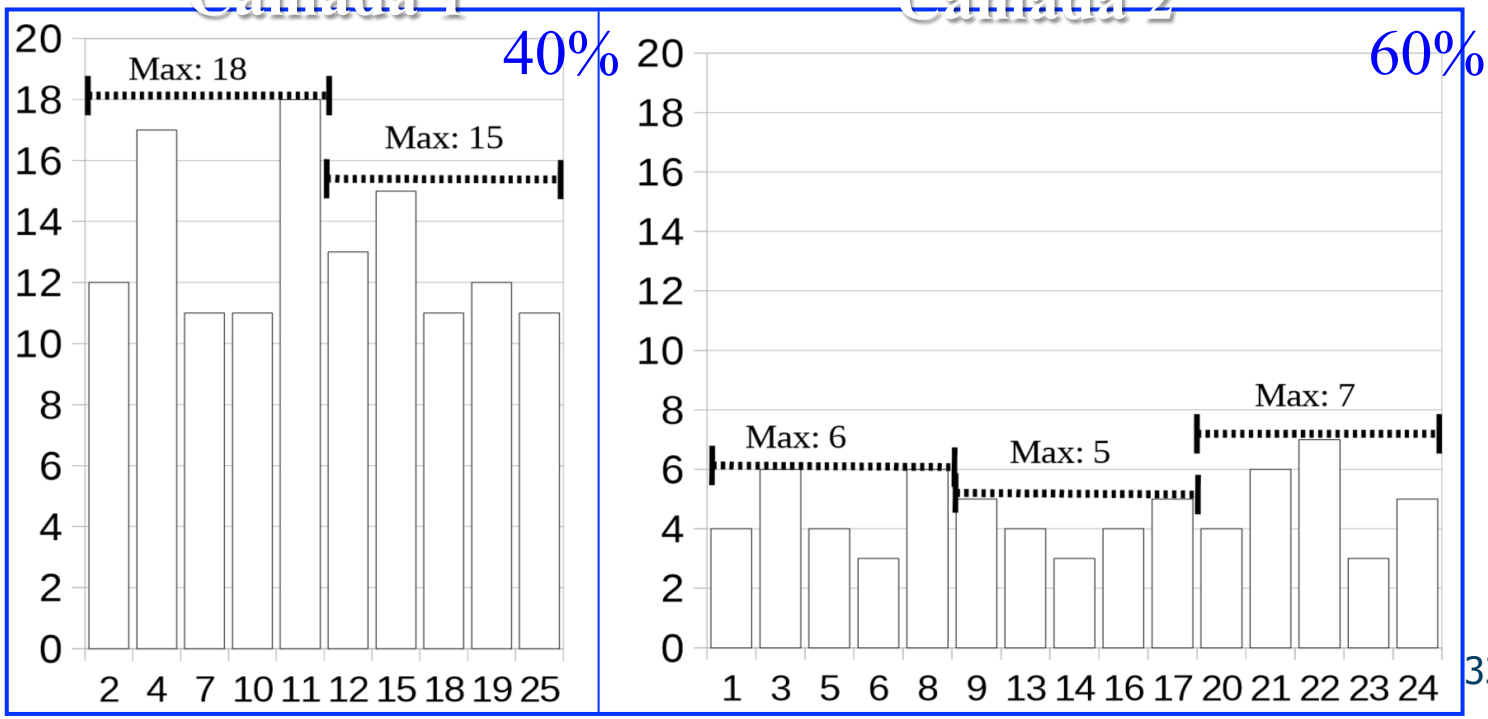
BMW (Block Max Wand)



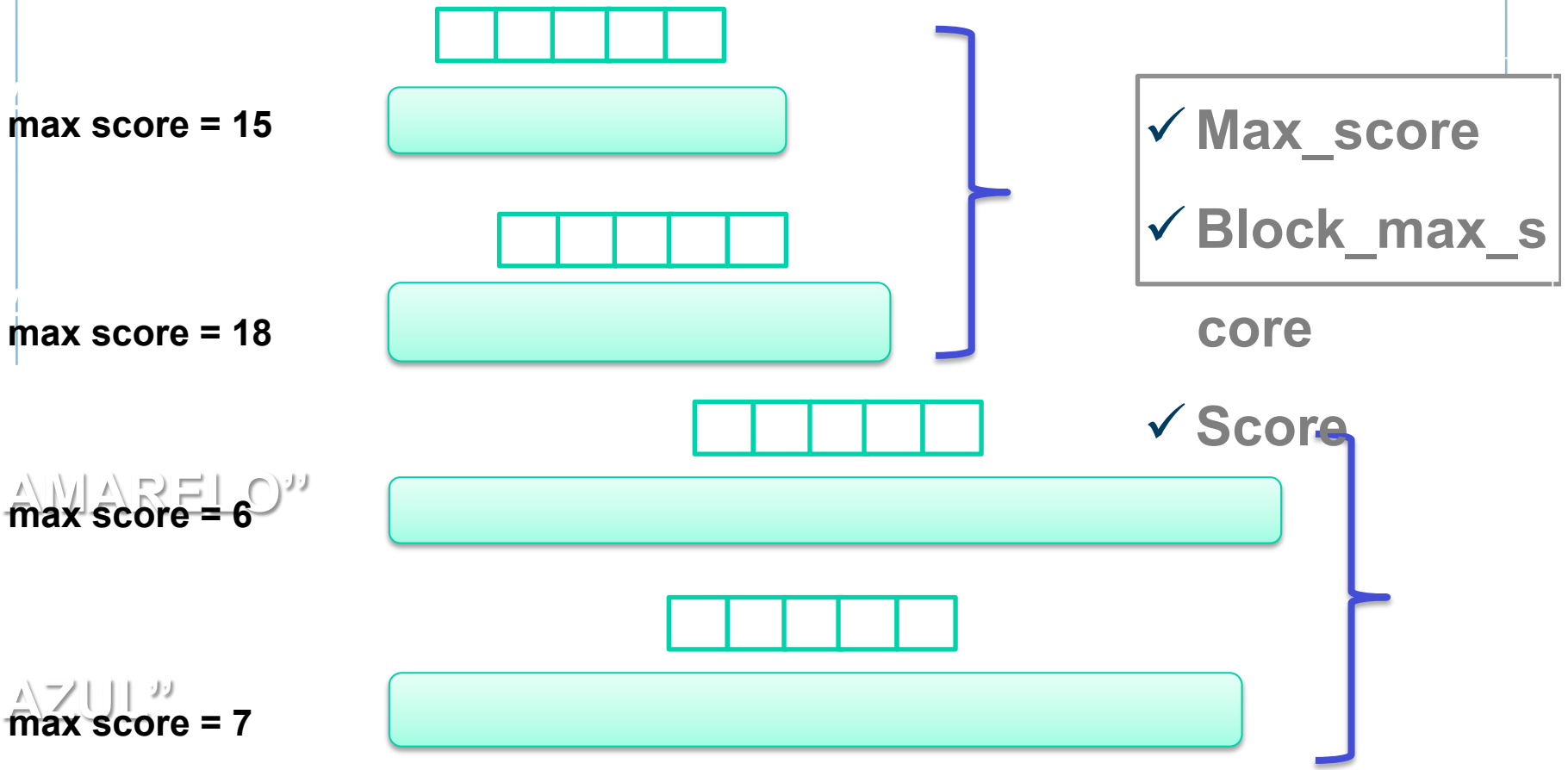
- ✓ Max_score
- ✓ Block_max_score
- ✓ Score

MBMW


Max: 17
Max: 18
Max: 11
Max: 12
Max: 11
Score
Documentos


Camada 1
40%
Max: 18
Max: 15
Camada 2
60%
Max: 6
Max: 5
Max: 7

MBMW


max score = 15
max score = 18
AMARELO”
max score = 6
AZUL”
max score = 7
✓ Max_score
✓ Block_max_s
core
✓ Score

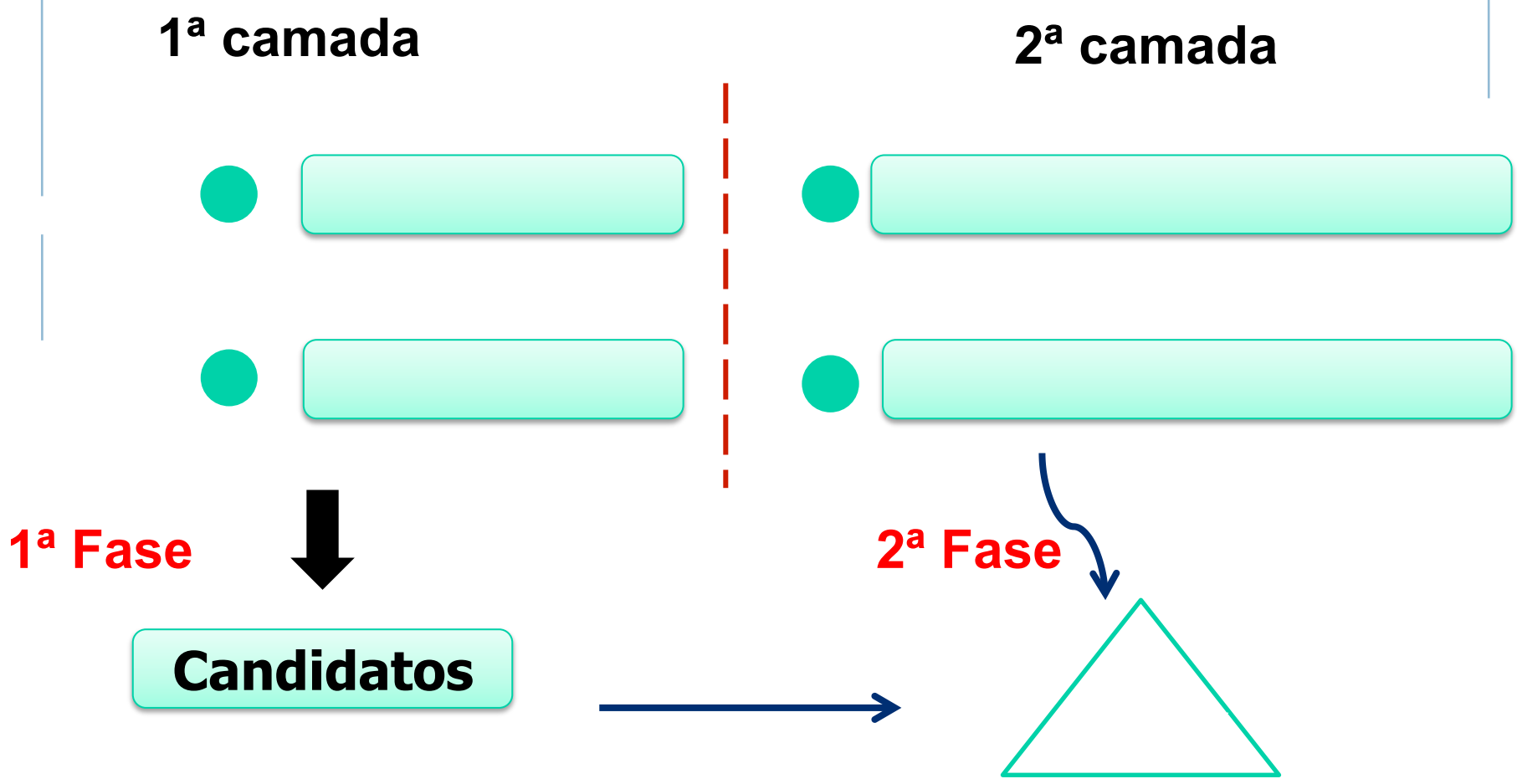
Comparações de tempo

TREC 2006						
	avg	2	3	4	5	> 5
exhaustive OR	225.7	60	159.2	261.4	376	646.4
WAND	77.6	23.0	42.5	89.9	141.2	251.6
SC	64.3	12.2	36.7	75.6	117.2	226.3
BMW	27.9	4.07	11.52	33.6	54.5	114.2
exhaustive AND	11.4	10.3	10.8	14.0	15.4	15.2
TREC 2005						
	avg	2	3	4	5	> 5
exhaustive OR	369.3	62.1	238.9	515.2	778.3	1501.4
WAND	64.4	23.5	43.7	73.4	98.9	265.9
SC	63.5	14.2	37.5	119.7	172.9	316.9
BMW	21.2	3.5	12.7	25.2	39	104
exhaustive AND	6.86	6.4	7.3	9.2	4.7	5.9

*SC: Strohman and Croft (Sigir, 2007)

Table 1: Average query processing time in ms for different numbers of query terms, using different algorithms on the TREC 2006 and 2005 query logs. Exhaustive OR, WAND, SC, and BMW are for disjunctive queries, while Exhaustive AND is for conjunctive queries.

BMW-CS (with Candidates Selection**)**



BMW-CSP

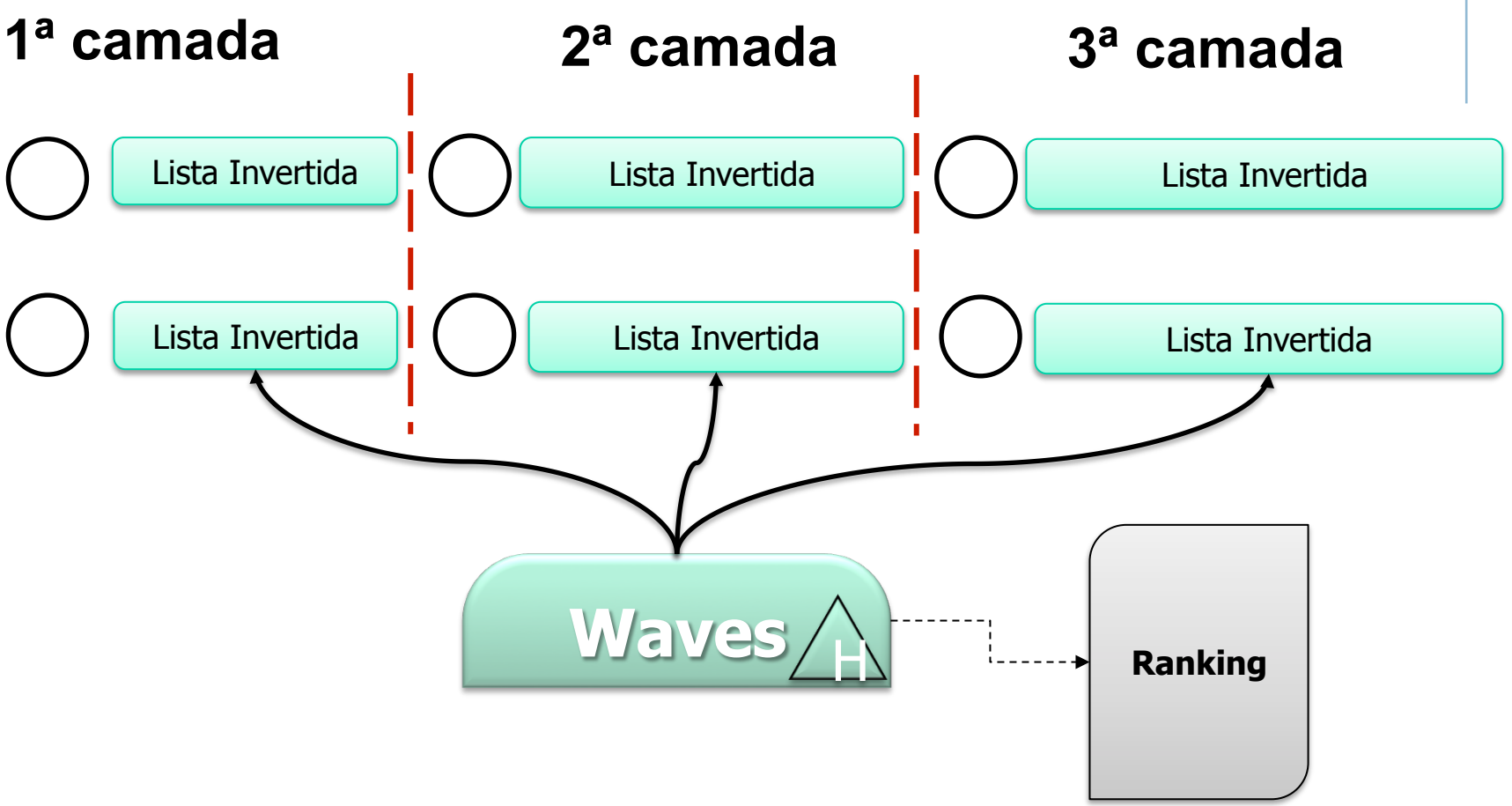
1ª camada

- Lista Invertida
- Lista Invertida

2ª camada

- Lista Invertida
- Lista Invertida

Waves


1ª camada
2ª camada
3ª camada
Lista Invertida
Lista Invertida
Lista Invertida
Lista Invertida
Lista Invertida
Lista Invertida
Waves
H
Ranking

Waves

Camada 1

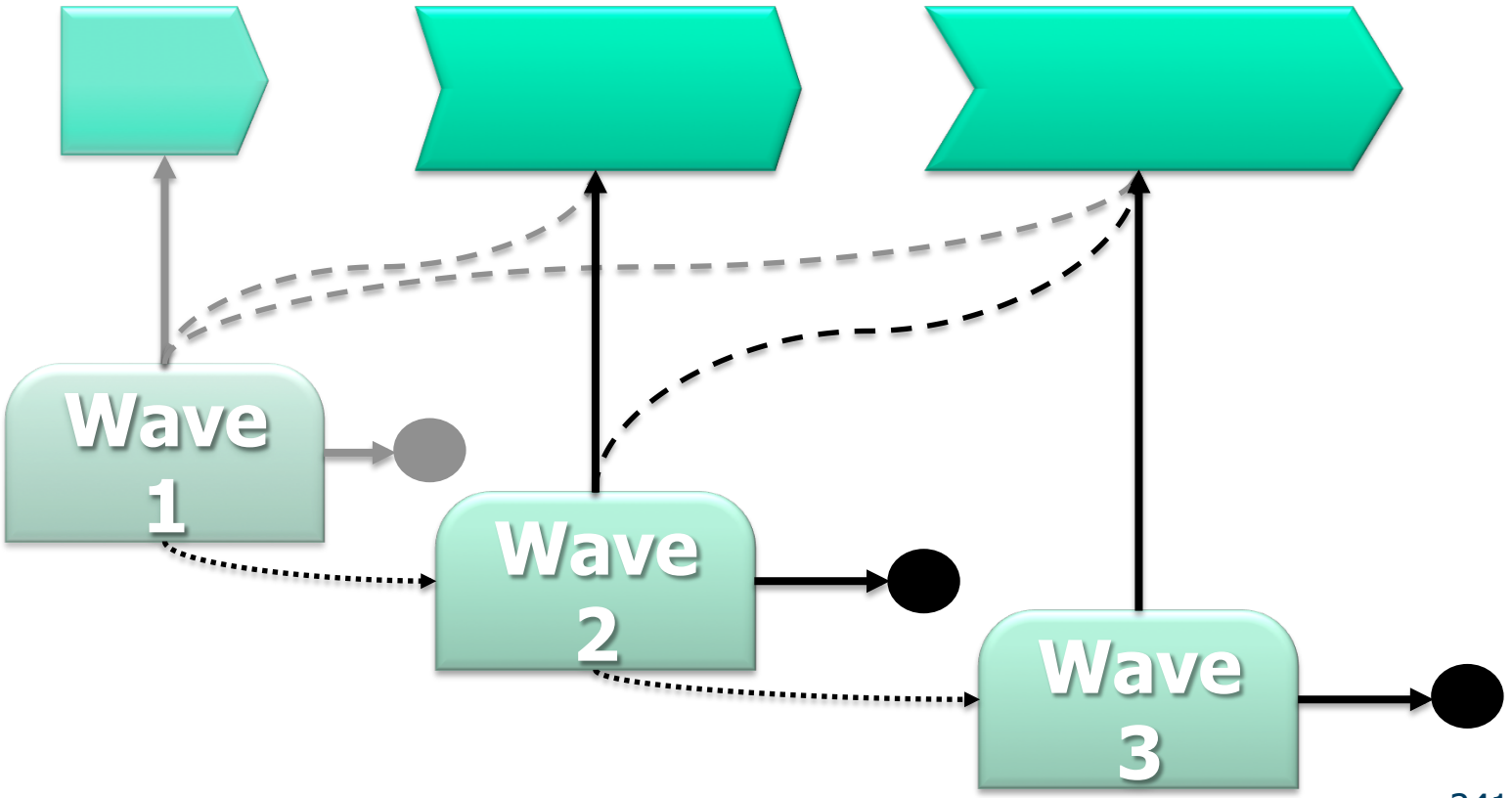


Table 7 Average time(ms) while computing top-10 and top-1000 results for subsets of GOV2

#Terms	# Gov100k 1,539,109	Gov1m 6,113,818	Gov10m 26,329,422	GovFull 45,664,906
Top-10				
BMWT	0.88 ± 0.03	3.83 ± 0.14	22.30 ± 0.91	46.21 ± 2.09
MBMWT	0.77 ± 0.02	3.06 ± 0.12	16.68 ± 0.83	35.27 ± 1.91
BMW-CSP	0.30 ± 0.01	1.63 ± 0.09	9.34 ± 0.62	19.64 ± 1.35
Waves	0.29 ± 0.01	1.51 ± 0.09	8.31 ± 0.55	16.08 ± 1.03
Top-1000				
BMWT	1.87 ± 0.04	8.58 ± 0.24	52.14 ± 1.65	108.28 ± 3.69
MBMWT	1.71 ± 0.04	7.18 ± 0.21	39.82 ± 1.42	82.66 ± 3.14
BMW-CSP	1.23 ± 0.04	6.06 ± 0.20	32.42 ± 1.25	66.35 ± 2.92
Waves	1.21 ± 0.04	5.85 ± 0.20	31.31 ± 1.20	57.40 ± 2.55

Confidence interval for times were computed at confidence level 95%

ıble 8 Average number of aluated pivots (entries where e actual frequency was acces-d) and average time(ms) while mputing top-10 and top-1000 sults and processing queries of LUEWEB09

	#Blocks	#Pivots	Time (r
Top-10			
BMWT	15,482	181,891	13.06 ±
MBMWT	13,719	122,501	10.97 ±
BMW-CSP	5715	95,688	7.85 ±
Waves	3779	76,468	4.27 ±
Top-1000			
BMWT	42,994	683,119	44.62 ±
MBMWT	32,472	418,113	33.36 ±
BMW-CSP	14,883	480,486	30.27 ±
Waves	14,580	188,081	18.22 ±

onfidence interval for times ere computed at confidence vel 95%

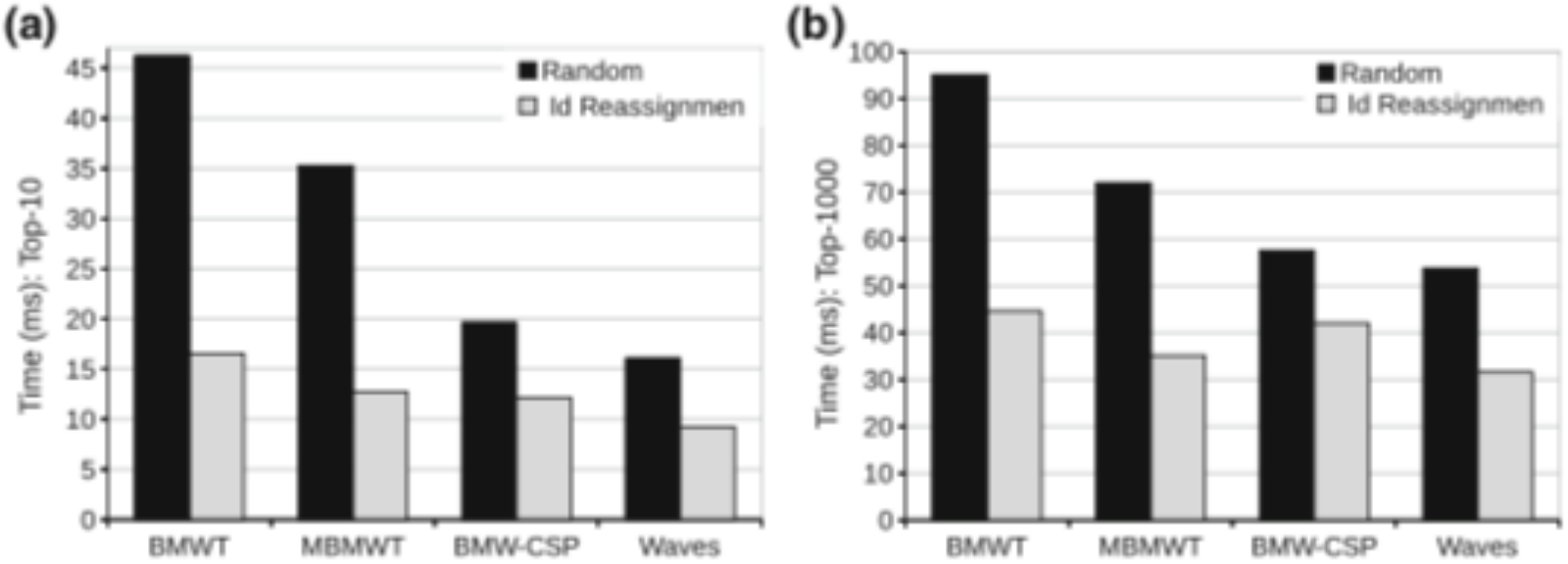


Fig. 8 Average times achieved by BMWT, MBMWT, BMW-CSP and Waves while using a document id reassignment technique to index the GOV2 collection (Id Reassignment), compared to the times achieved while using a random document id assignment (Random Assignment)

Estratégias para melhorar desempenho sistemas de busca

Exemplos

- Métodos de Poda (falamos sobre poda durante o processamento, podemos ir mais longe....)
- Compressão de Dados (já falamos de índice, agora vamos falar em textos...)
- Cache
- Paralelismo

Métodos de Poda

- Visam utilizar a menor porção possível dos índices durante o processamento de consultas
- Podem ser dinâmicos ou estáticos
- Métodos estáticos podem ser vistos como compressão com perdas

Método de Persin

- Algumas entradas das listas invertidas não causam impacto na ordenação final das respostas
- Pode-se reduzir significativamente o custo de processamento se identificarmos estes elementos antes de utiliza-los
- Persin et al propuseram um método para identificar estes elementos
- Método economiza memória e reduz tempo de processamento

Método de Persin

- As listas invertidas devem ter seus elementos ordenados por freqüência
- Os termos devem ser processados em ordem decrescente de Idf
- Dois limiares são usados durante a poda:
  - Limiar de inserção S_ins
  - Limiar de Adição S_add

Método de Persin

- A cada nova entrada da lista, a similaridade parcial sim(q,t,d) é calculada
- Esta similaridade representa o impacto da entrada na consulta
- Possibilidades:
  - sim(q,t,d) >= S_ins (inserir/acumular)
  - S_ins > sim(q,t,d) >= S_add (acumular)
  - S_add > sim(q,t,d) (dercartar)

Cálculo dos limiares

- Os limiares são calculados a partir da maior similaridade parcial encontrada nos acumuladores (S_max):
  - S_add = c_add x S_max (c_add >0)
  - S_ins = c_ins x S_max (c_ins > c_add)

$$s \leq sim_{(q,d,t)}$$

$$s \leq f_{d,t} \cdot idf_t \cdot f_{q,t} \cdot idf_t$$

$$\frac{s}{f_{q,t} \cdot idf_{t^2}} \leq f_{d,t}$$

Cálculo dos limiares

$$s \leq sim_{(q,d,t)}$$

$$s \leq f_{d,t} \cdot idf_t \cdot f_{q,t} \cdot idf_t$$

$$\frac{s}{f_{q,t} \cdot idf_{t^2}} \leq f_{d,t}$$

Cálculo dos limiares

$$f_{ins} = \frac{s_{ins}}{f_{q,t} \cdot idf_t^2} = \frac{c_{ins} \cdot S_{\max}}{f_{q,t} \cdot idf_t^2}$$

$$f_{add} = \frac{s_{add}}{f_{q,t} \cdot idf_t^2} = \frac{c_{add} \cdot S_{\max}}{f_{q,t} \cdot idf_t^2}$$

Algoritmo de Persin

1. Criar uma estrutura vazia de acumuladores.
2. Ordenar os termos da consulta por *idf* decrescente.
3. $S_{max} \leftarrow 0$.
4. Para cada termo t na consulta,
   (a) Calcular os valores dos limiares f_{ins} e f_{add}.
   (b) Se $f_t^{max} < f_{add}$ ento retornar ao passo 4.
   (c) Para o primeiro bloco na lista invertida de t e para cada documento d na seqncia do bloco,
      i. Se $f_{d,t} \geq f_{ins}$, criar um acumulador A_d se necessário, e $A_d \leftarrow A_d + sim_{q,d,t}$.
      ii. Caso contrário, se $f_{d,t} \geq f_{add}$ e A_d existe ento $A_d \leftarrow A_d + sim_{q,d,t}$.
      iii. $S_{max} \leftarrow max(S_{max}, A_d)$.

Algoritmo de Persin

(d) Para cada bloco restante na lista invertida de t com $f_{d,t} \geq f_{add}$
e para cada documento d no bloco,

i. Se $f_{d,t} \geq f_{ins}$, criar um acumulador A_d
se necessário, e $A_d \leftarrow A_d + sim_{q,d,t}$.

ii. Caso contrário, se $f_{d,t} \geq f_{add}$ e A_d existe
ento $A_d \leftarrow A_d + sim_{q,d,t}$.

iii. $S_{max} \leftarrow max(S_{max}, A_d)$.

5. Dividir cada acumulador $A_d \neq 0$ pela norma do documento $|\vec{d}|$.
6. Identificar os k valores mais altos de acumuladores, onde k
é o número de documentos retornados.

Resultados

- Quantidade de memória usada cai para ¼ da memória usada no vetorial
- Tempo de processamento é reduzido a uma taxa próxima a 5 vezes

Métodos de poda estática

- Retiram entradas dos índices durante a indexação.
- Índice é reduzido, mas pode perder resultados de busca.

Poda estática

- Métodos baseados em localidade
- Intuição de que termos de uma consulta tendem a ocorrer próximos nos texto
- Diversos métodos foram propostos
- Redução de até 50% no tamanho do índice sem perda de qualidade nas respostas
- Perda aceitável até 60%

Combinando métodos

- Pode-se combinar métodos estáticos com dinâmicos ?
- Poda por tipo de consulta ?

Compressão de Dados

- Métodos de compressão podem ser usados para reduzir custos
- Já vimos que compressão de índices reduz demanda por espaço em disco e tempo de processamento

Compressão de Textos

- Para grandes bases de dados pode-se usar o método de Huffman
- Símbolos a serem codificados são palavras ao invés de caracteres

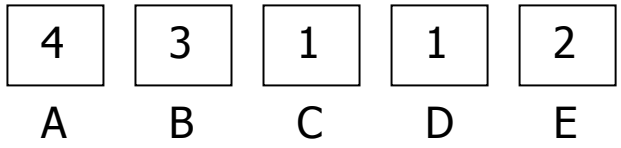
Algoritmo de Huffman

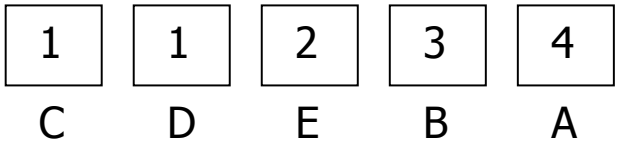
- Dado um conjunto de símbolos com suas respectivas probabilidades de ocorrências constrói-se uma árvore de codificação das folhas para a raiz.

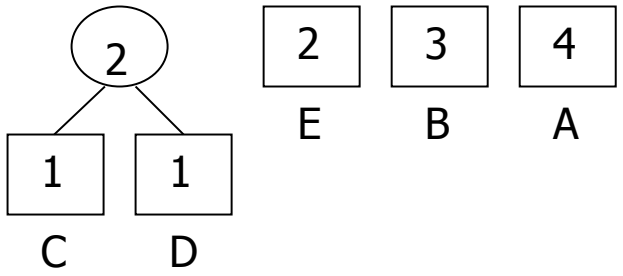
Algoritmo de Huffman

- inicialmente cada símbolo é representado por um nó que é tratado como uma árvore individual
- cada nó recebe um peso equivalente a probabilidade de ocorrência do símbolo correspondente
- As duas árvores de menor peso são então unidas por um nó interno que recebe peso igual a soma das probabilidades das árvores unidas
- O processo é repetido até que reste somente uma árvore, a qual é conhecida como árvore de Huffman

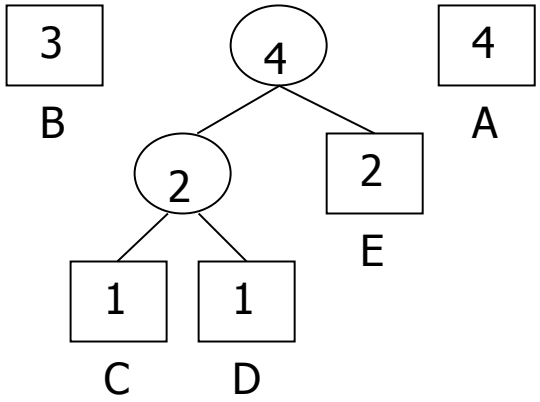
Exemplo de árvore de Huffman



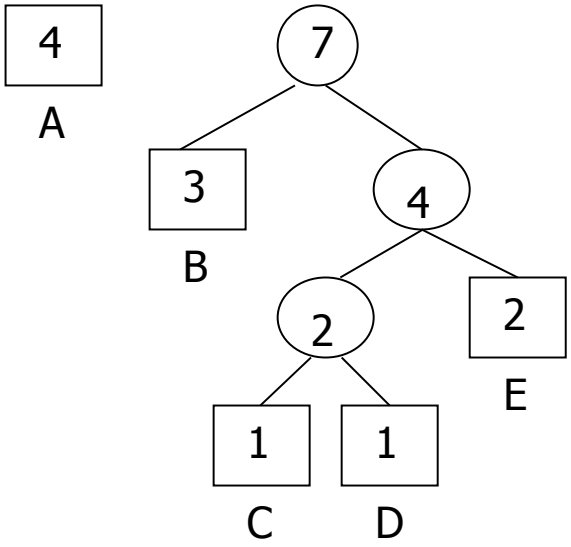




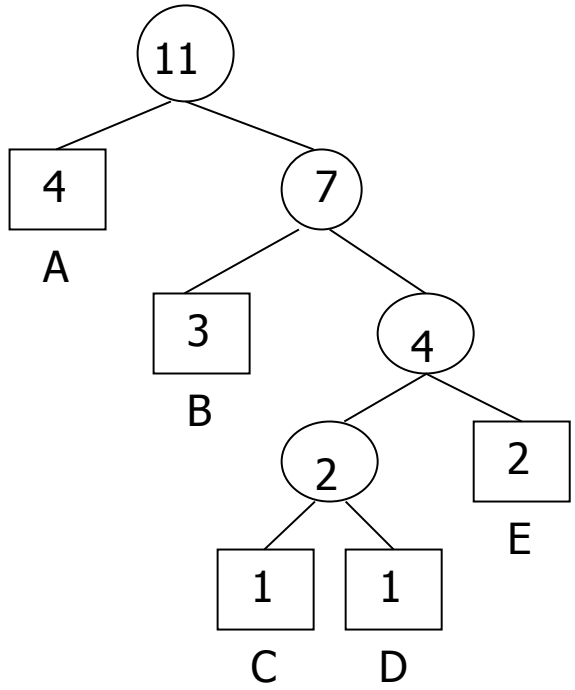
Exemplo de árvore de Huffman



Exemplo de árvore de Huffman



Exemplo de árvore de Huffman



Construa o Código de Huffman

A 1/2

B 1/4

C 1/8

D 1/16

E 1/32

A 1

B 5

C 2

D 9

E 13

F 1

Código Canônico

- Um código de Huffman é considerado canônico se ele atende às seguintes propriedades
  - os códigos são dispostos em ordem crescente de tamanho
  - todos os códigos de um dado tamanho são números consecutivos
  - Se c for o primeiro código de l dígitos (o de menor valor numérico) e d for o último código de l-1 dígitos, então c = 2 x(d+1)

Código Canônico

- Um código canônico pode ser montado utilizando-se pouca informação
  - menor nível, maior nível e número de elementos em cada nível

Exemplo

- Dado que:
  - Min = 2, Max = 3
  - NumElem[2] = 3
  - NumElem[3]= 2
  - Símbolos ordenados são A,B,E,C,D
- Qual a árvore de codificação canônica correspondente ?

Resultado

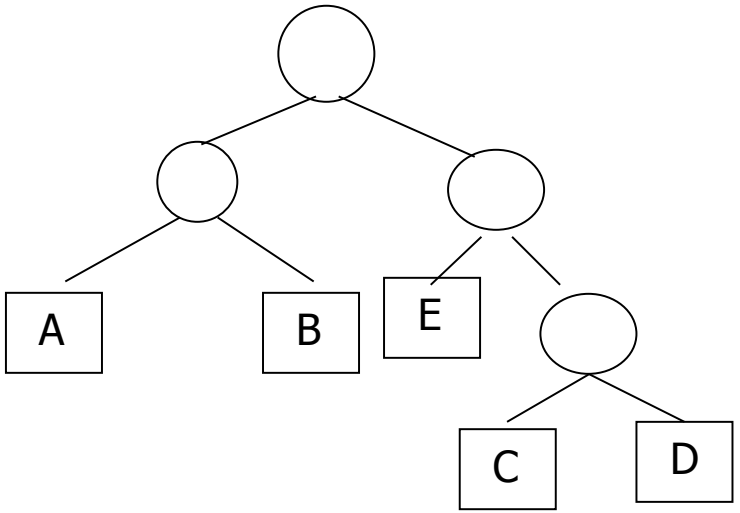
- Códigos começam pelos símbolos menores (três elem. nível 2):

  00 – A; 01 – B; 10 – C

- Primeiro elemento do próximo nível igual a 2x(2+1) = 6

  110 – C; 111 - D

Resultado



Decodificação do código canônico

- Decodificação pode ser realizada utilizando-se propriedades do código canônico utilizando-se mais duas tabelas:
  - Limite – contém o maior número existente em cada nível
  - Base – contém a posição do último elemento de cada nível na tabela de símbolos do decodificador
  - Posição(c) = Base[l(c)] + c - Limite[l(c)]

Decodificação do código canônico

- Limite serve para identificar que um código completo foi obtido

Codificação e Decodificação

- Min = 2, Max = 3
- NumElem[2] = 3
- NumElem[3]= 2
- Base[2]=2
- Base[3]=4
- Limite[2] =2
- Limite[3]=7

Tabela de Símbolos Ordenada:

Posição	Símbolo
0	A
1	B
2	E
3	C
4	D

Exemplo

- Decodifique 11100 utilizando as tabelas fornecidas

Utilização em Sistemas de Busca

- Compressão da base de texto
- Huffman para palavras
- Byte Huffman (árvores com 256 filhos)
- Boa velocidade para comprimir e descomprimir
- Descompressão de fragmentos
- Utilização do vocabulário do sistema como vocabulário de Huffman
- Código canônico com poucos bytes para representar árvore

Questões importantes

- Compressão de HTMLs ?
- Compressão do texto para mostrar trechos e impacto nas listas invertidas posicionais (marcação de termos da consulta pode ser facilitada se base está comprimida!)
- Quando comprimir (durante a coleta ?)
- Como atualizar o vocabulário nesses casos ?

Cache

- Pode-se utilizar também estratégias de Cache para sistemas de busca
- Uma proposta é combinar cache de respostas com cache de listas invertidas

Cache em Máquinas de Busca para a Web

- Total de consultas distintas representa cerca de 30% do total de consultas
- Total de termos distintos representa menos que 5% do total de termos nas consultas
- Estes dados indicam que técnicas de cache podem dar bons resultados

Cache em Máquinas de Busca

- O uso de cache de respostas aumenta a capacidade total de processamento em cerca de 100%
- O uso de cache de listas invertidas aumenta a capacidade de processamento em cerca de 50%
- A combinação dos dois pode dar um resultado melhor ?

Cache em Máquinas de Busca

- Sim:
  - Um cache que armazena respostas e listas invertidas pode aumentar o número de consultas processadas em até 3 vezes.

Questões importantes

- Cache x velocidade de atualização de respostas (mudanças no ranking)
- Cache x Atualização do índice
- Distribuição do índice/ replicação de máquinas (vamos ver em seguida)

- Cache em três níveis (www 2005): cache de pares de palavras (fragmentos da consulta)

Cache, muito mais por ai

- Há vários trabalhos publicados sobre cache nos últimos anos.
- Procure estudar mais sobre o assunto.
- Teremos seminário ligado ao tema.

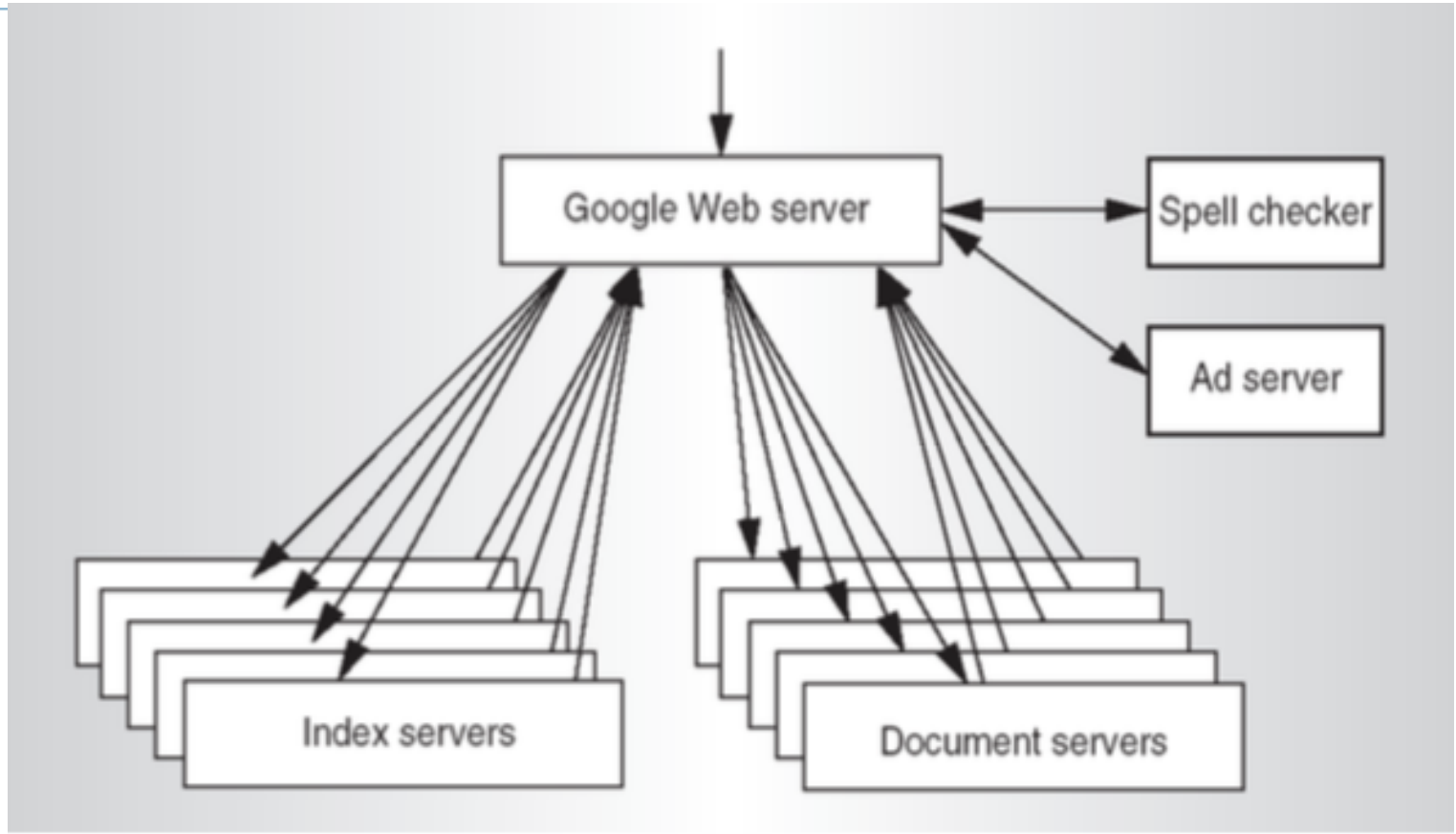
Paralelismo

- Geração distribuída de índices
- Processamento distribuído de consultas
  - Local Index (distribuição baseada em documentos, cada máquina tem um índice local completo)
  - Global Index (distribuição baseada em listas invertidas, cada máquina fica com listas invertidas completas)
  - Desafios…

Local Index

- Opção parece ser a adotada por MBs comerciais
  - Manutenção é simples
  - Facilidade de implementação
  - Desempenho é similar

Arquitetura de índices do Google de um sistema de busca em larga escala



Barroso, L. A., Dean, J., & Hölzle, U. (2003). Web search for a planet: The Google cluster architecture. *IEEE micro*, (2), 22-28.

Arquitetura esperada de um sistema com local index

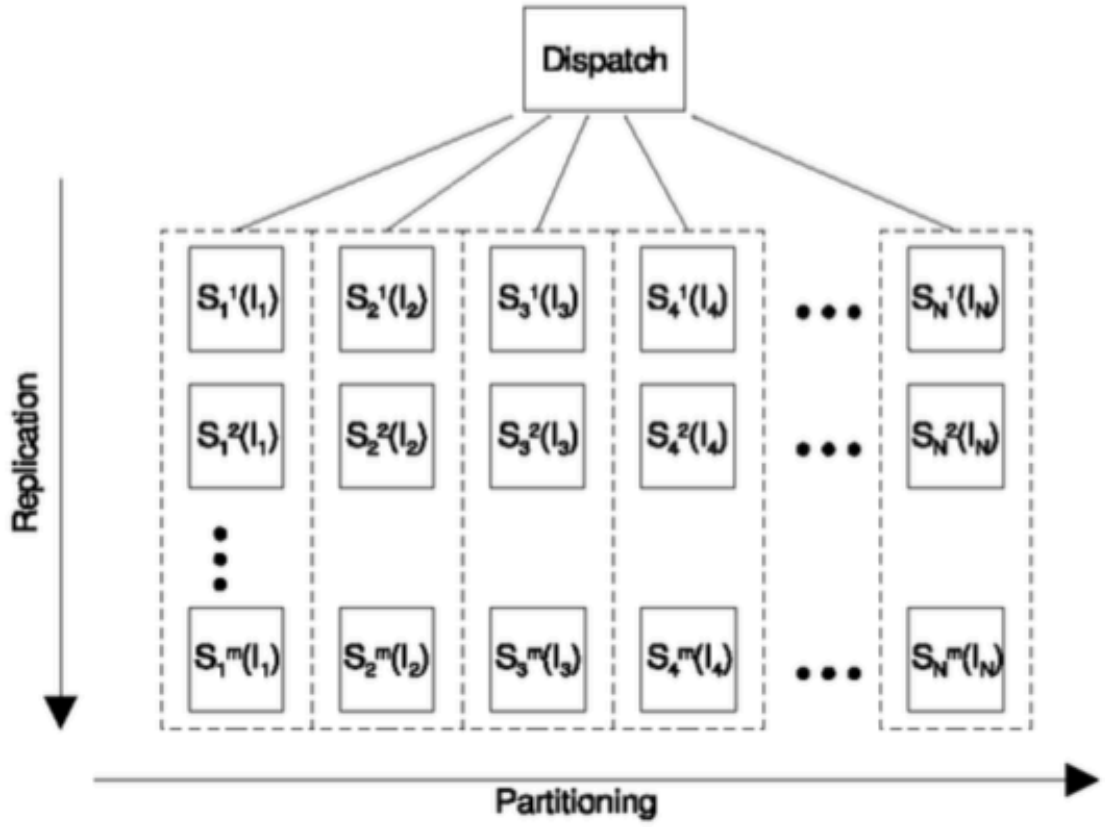


Figure 3.4: Fast search cluster overview.

Questões importantes

- Devo usar brokers separados dos processadores de consulta ou acoplados aos mesmos ?
- Como atualizar os índices ?
- Quantas respostas devo pegar de cada máquina para formar a resposta final (qualidade x custo) ?
- Balanceamento de carga: parece simples, mas não é!

Referências para se aprofundar

- Referência: tese de doutorado de Claudine Badue, UFMG (2007)
- Xu, C., Wang, Y., Lv, P., & Xu, J. Caching-Aware Techniques for Query Workload Partitioning in Parallel Search Engines. In *2017 14th Web Information Systems and Applications Conference (WISA)* (2017).
- Moffat, A., Webber, W., Zobel, J., & Baeza-Yates, R. A pipelined architecture for distributed text query evaluation. *Information Retrieval*, . (2007).
- Jonassen, Simon; Bratsberg, Svein Erik. A Combined Semi-pipelined Query Processing Architecture for Distributed Full-Text Retrieval. WEB INFORMATION SYSTEM ENGINEERING-WISE 2010

Atualização de Índices

- Reconstrução completa
- Online
- Log de updates

Referências adicionais

- Zobel, Justin, and Alistair Moffat. "Inverted files for text search engines." *ACM computing surveys* (2006)
- ALICI, Sadiye et al. Timestamp-based result cache invalidation for web search engines. In: **SIGIR (**2011)
- SARIGIANNIS, Charalampos; PLACHOURAS, Vassilis; BAEZA-YATES, Ricardo. A study of the impact of index updates on distributed query processing for web search. In: **European Conference on Information Retrieval**. (2009)

Questões importantes

- Como comprimir índices que são atualizados ?
- Como distribuir índices que são atualizados?