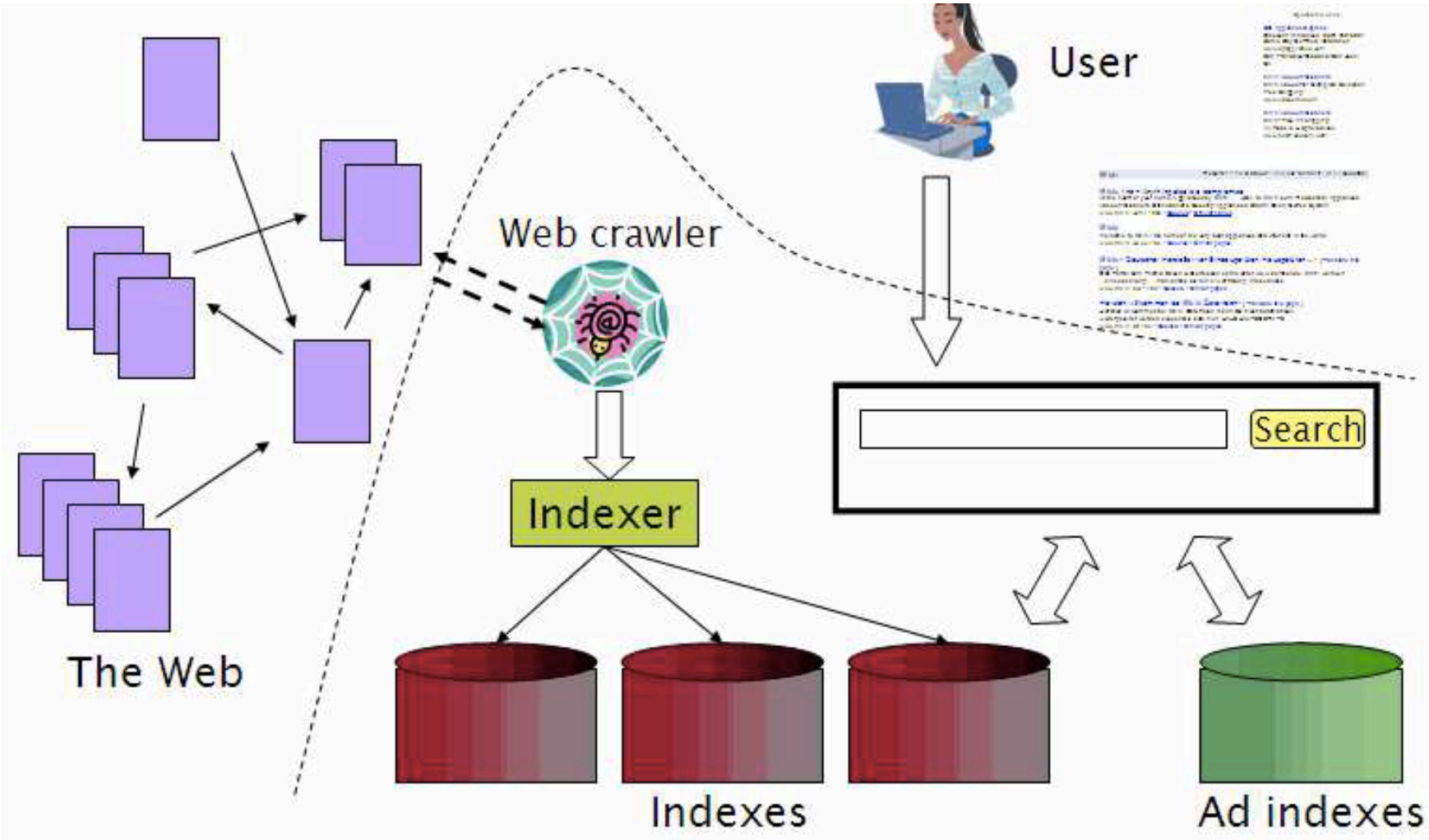
Processamento de Consulta e Ranking

Prof. Luciano Barbosa

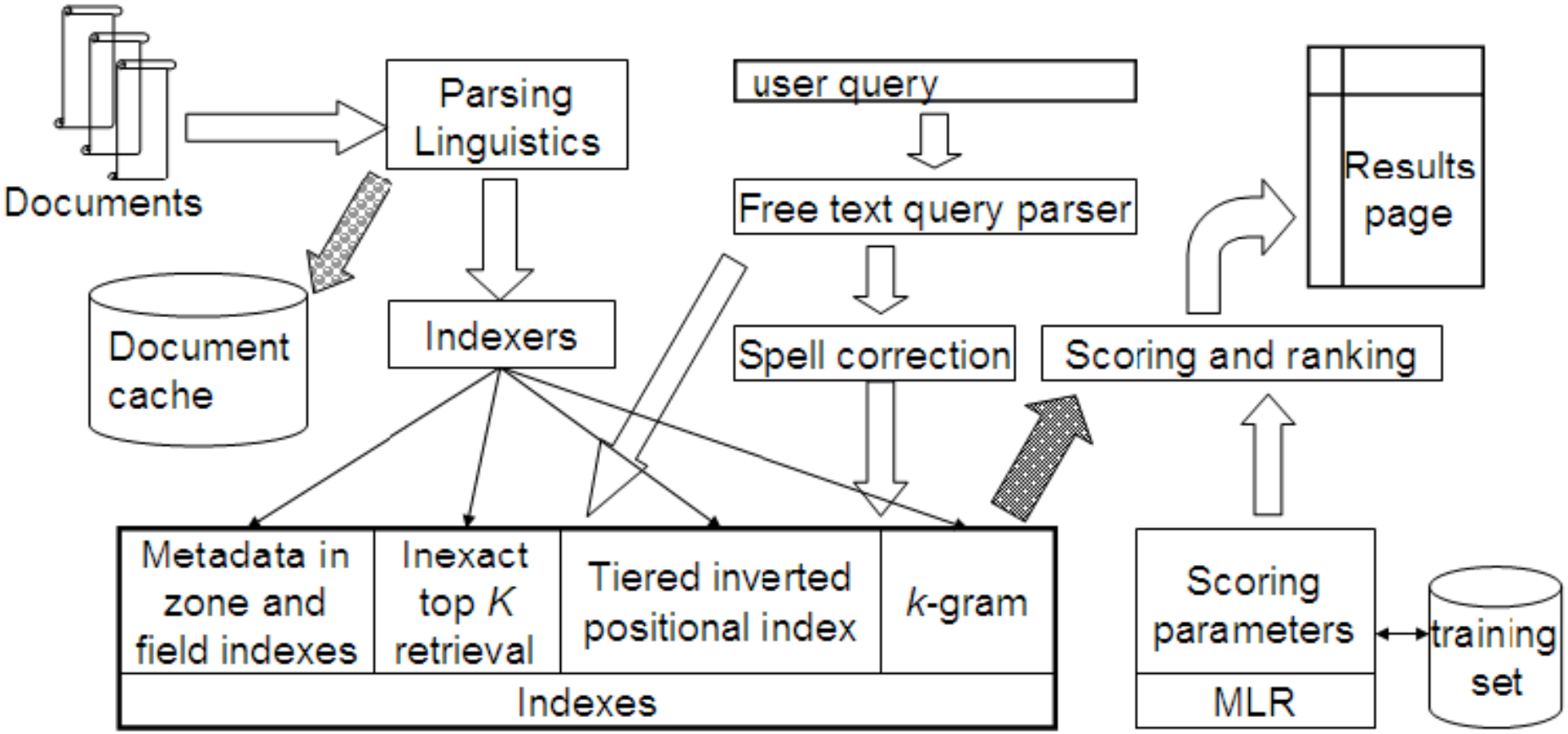
(Parte do material retirado dos slides dos livros adotados)

UNIVERSIDADE
FEDERAL
DE PERNAMBUCO

Visão Geral de um Engenho de Busca



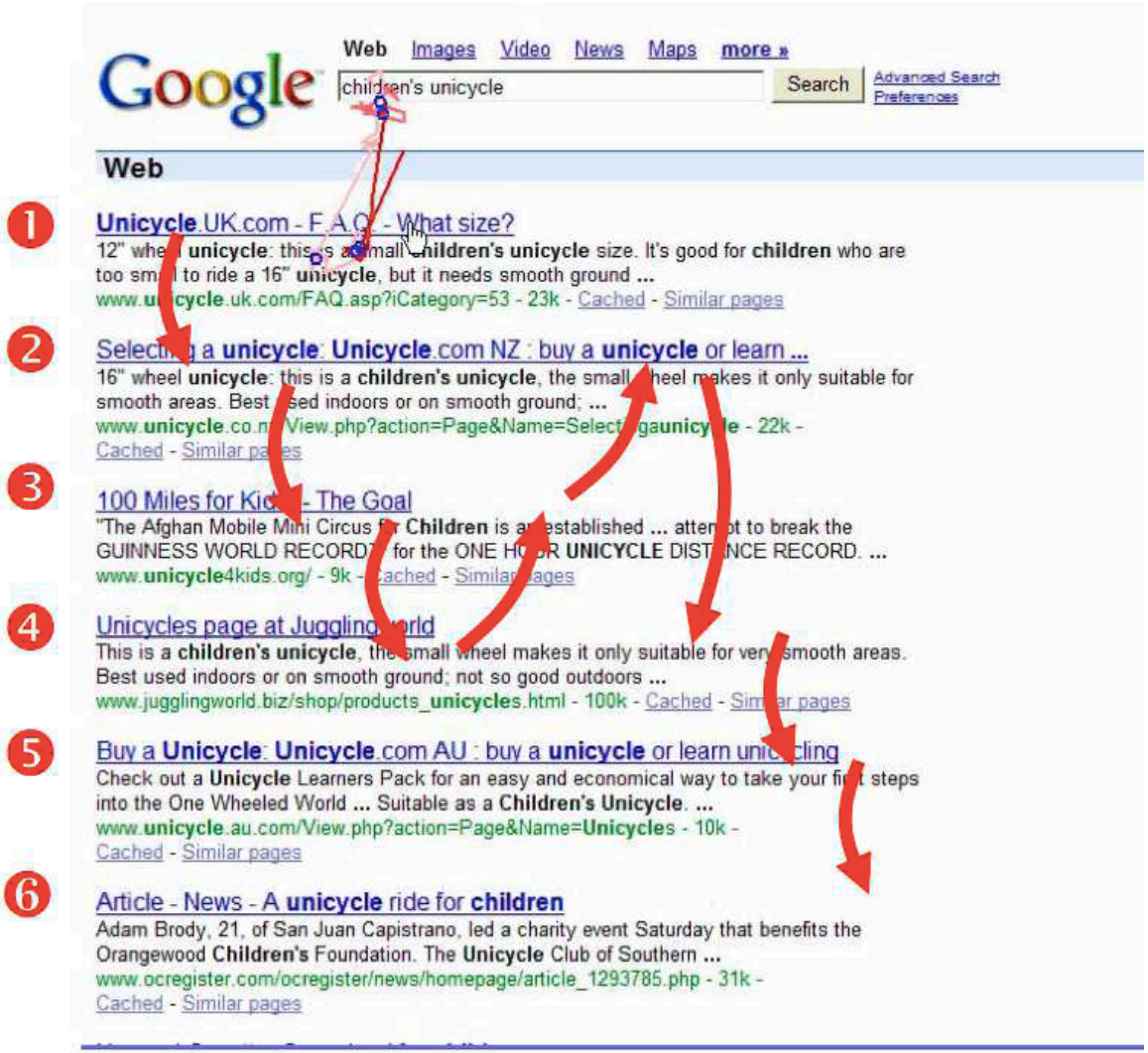
Módulo de Busca



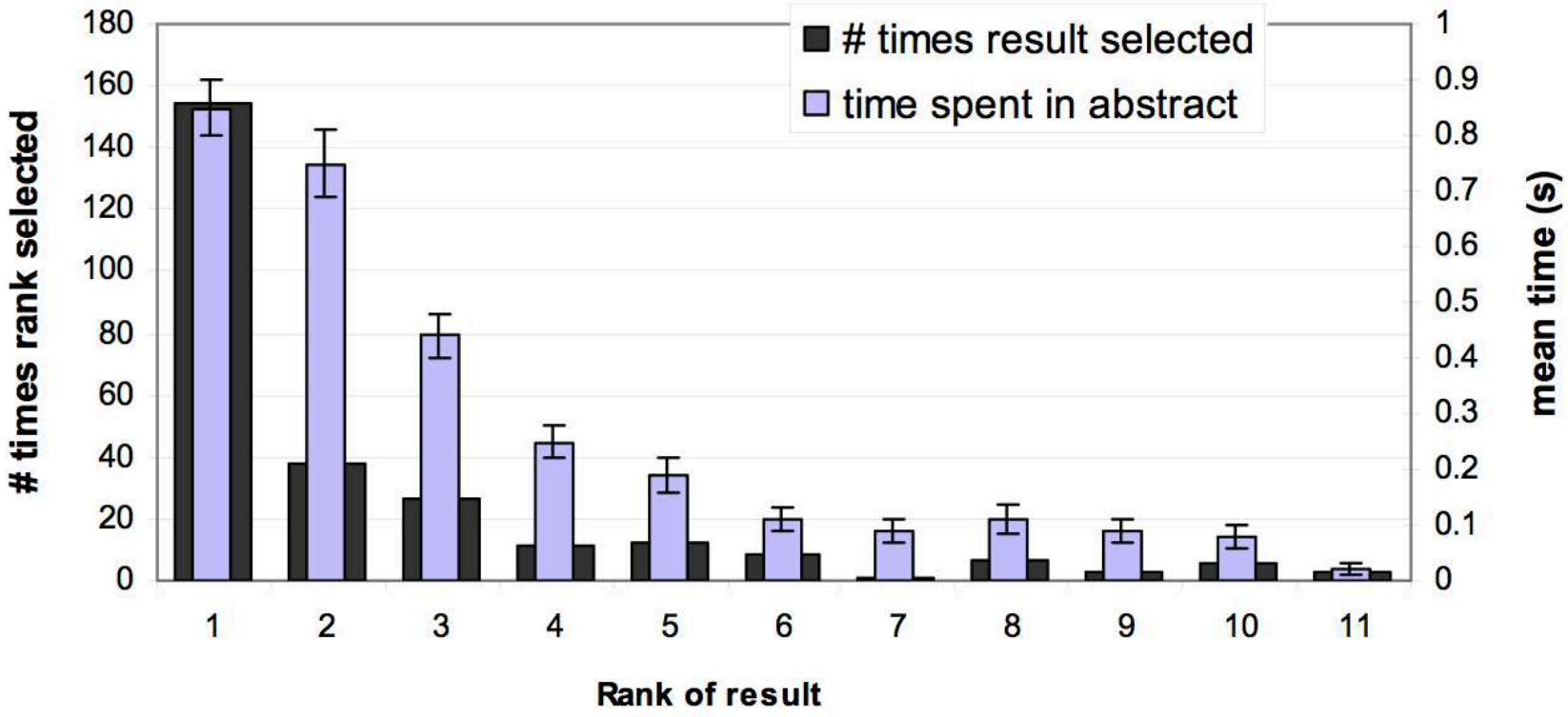
Investigação Empírica do Efeito do Ranking

- Como se mede a importância de um ranking?
- Observe usuários quando eles fazem busca num ambiente controlado
  - Grava vídeo
  - Pede para eles pensarem alto
  - Entrevista
  - Rastreia movimento dos olhos
  - Mede o tempo
  - Grava os links clicados

Observando os Resultados



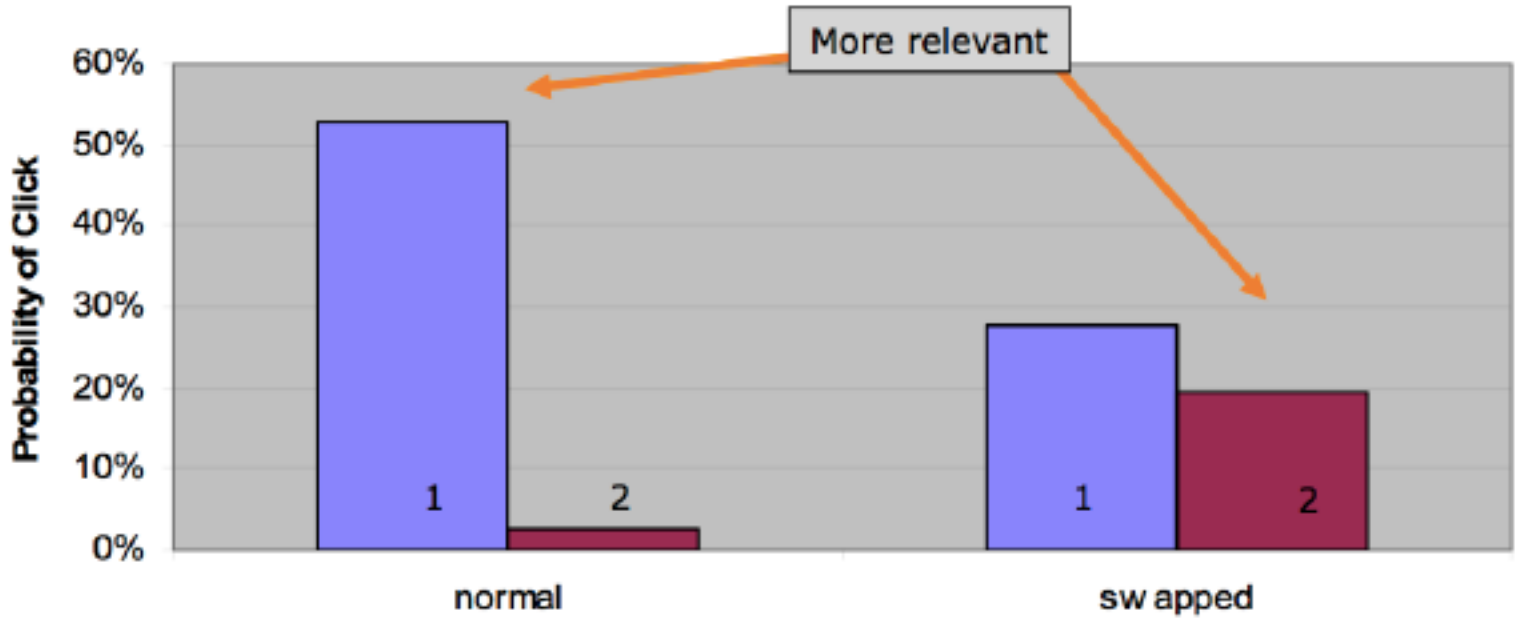
Olhar vs. Clicar



- Observam mais primeiro e segundo resultados
- Usuários clicam mais no primeiro

Ranking Influencia Comportamento do Usuários

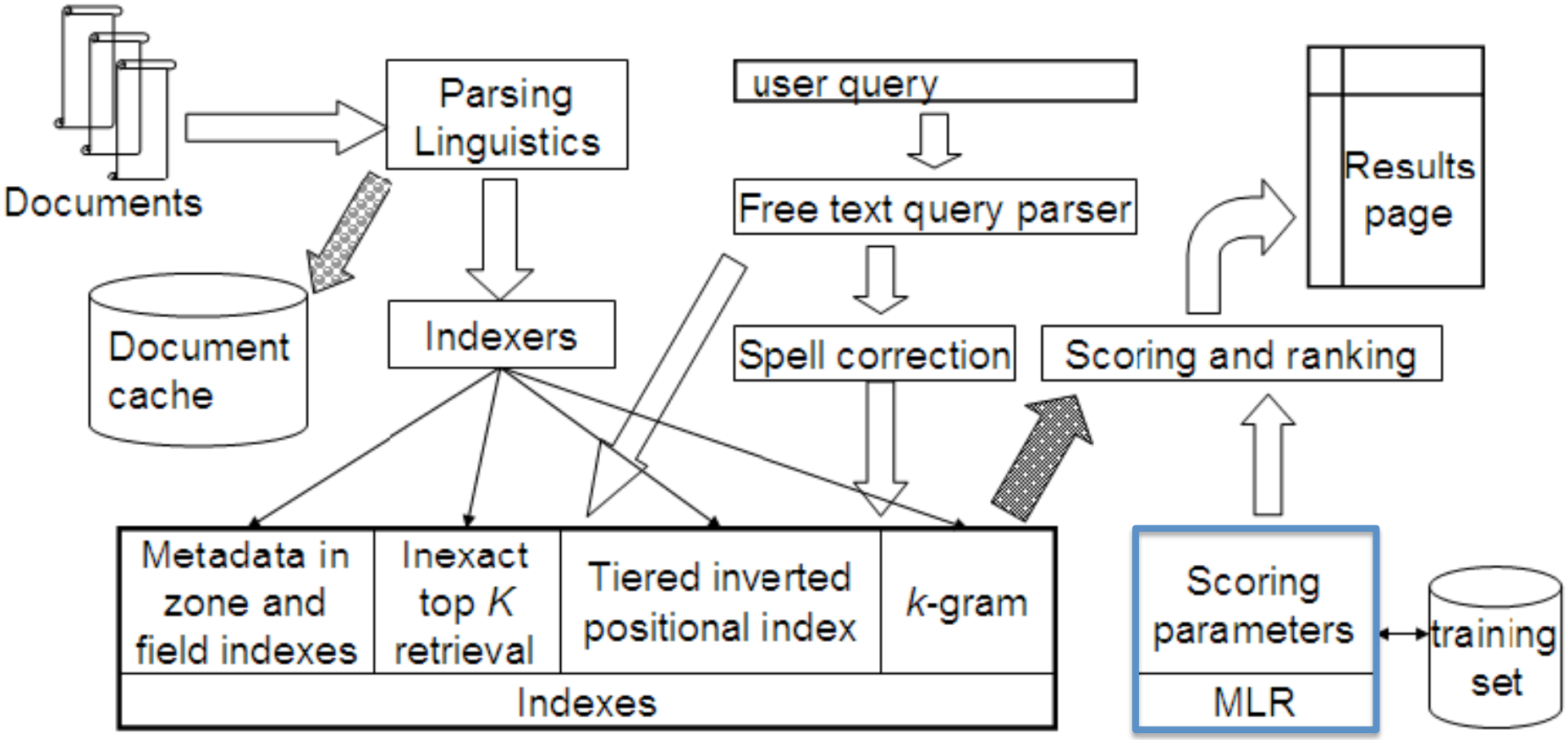
- Ordem dos resultados influencia onde usuários olham e onde clicam



Importância do Ranking: Sumário

- Usuários tendem a observar mais resultados no topo do ranking
- Usuários clicam 50% das vezes no primeiro resultado
- Mesmo que o primeiro resultado não seja relevante, o usuário clica 30% nele
- Construir um bom ranking é importante
- Colocar uma página relevante no topo é ainda mais importante

Módulo de Busca



Learning to Rank

- Usa machine learning para aprender o ranking
- BIM e relevance feedback são específicos para consultas
  - Aprender um ranqueamento para cada consulta
- Learning to rank é independente de consulta
- É aprendido um único classificador
- Pode-se ranquear documentos para consultas sem julgamento de relevância para elas

Learning to Rank

- Cada par documento-consulta é um ponto de dados
- Duas classes:
  - Relevante: documentos relevantes para as consultas
  - Não-relevante: documentos não relevantes para as consultas

Example	DocID	Query	s_T	s_B	Judgment
Φ_1	37	linux	1	1	Relevant
Φ_2	37	penguin	0	1	Nonrelevant
Φ_3	238	system	0	1	Relevant
Φ_4	238	penguin	0	0	Nonrelevant
Φ_5	1741	kernel	1	1	Relevant
Φ_6	2094	driver	0	1	Relevant
Φ_7	3194	driver	1	0	Nonrelevant

- Problema clássico de classificação
- Documentos ranqueados de acordo com probabilidade de relevância

Forma Simples de L2R: Zone Scoring

- Dada uma coleção dividida em campos
  - Ex: autor, título e corpo
- Um peso para o conteúdo de cada campo
- Os campos não são igualmente importantes
  - Ex: autor < título < corpo -> g_1=0.2; g_2=0.3; g_3=0.5
  - Se termos da consulta aparecem no título e corpo de um documento: score = (0.3 * 1) + (0.5 * 1) = 0.8

Funcionamento do Zone Scoring

- Entrada: consulta q e documento d
- Saída: score de “similaridade” entre 0 e 1
- Combinação linear dos campos (cada campo possui um peso)
  - Consider a set of documents, which have l zones
  - Let $g_1, \ldots, g_l \in [0, 1]$, such that $\sum_{i=1}^{l} g_i = 1$
  - For $1 \leq i \leq l$, let s_i be the Boolean score denoting a match (or non-match) between q and the i^{th} zone
    - $s_i = 1$ if a query term occurs in zone i, 0 otherwise

Weighted zone score a.k.a ranked Boolean retrieval

Rank documents according to $\sum_{i=1}^{l} g_i s_i$

Aprendendo os Pesos

- No free lunch: rotulagem de julgamentos de relevância de usuários
- Grandes engenhos de busca colocam bastante recursos para criar conjuntos de treinamento para L2R
- Uma vez com um conjunto grande o suficiente, o problema se reduz a um simples problema de otimização

Zone Scoring: Exemplo

- Considere documentos com dois campos: título e corpo
- Soma dos pesos dos campos:

$$score(d,q) = g \cdot s_T(d,q) + (1-g) \cdot s_B(d,q)$$

Onde: s_T(d,q): termo da consulta q aparece no título

s_B(d,q): termo da consulta q aparece no corpo

g: peso entre 0 e 1 e soma igual a 1

Determinando g

- Conjunto de treinamento: triplas $\Phi_j = (d_j, q_j, r(d_j, q_j))$

Example	DocID	Query	s_T	s_B	Judgment
Φ_1	37	linux	1	1	Relevant
Φ_2	37	penguin	0	1	Nonrelevant
Φ_3	238	system	0	1	Relevant
Φ_4	238	penguin	0	0	Nonrelevant
Φ_5	1741	kernel	1	1	Relevant
Φ_6	2094	driver	0	1	Relevant
Φ_7	3194	driver	1	0	Nonrelevant

$$score(d, q) = g \cdot s_T(d, q) + (1 - g) \cdot s_B(d, q)$$

Aprendendo os Pesos

- Comparar o resultado de score(d_j, q_j) com julgamento de relevância para o mesmo par (d_j, q_j)
- Erro:

$$\epsilon(g, \Phi_j) = (r(d_j, q_j) - score(d_j, q_j))^2$$

- Total de erro em um conjunto de treinamento

$$\sum_j \epsilon(g, \Phi_j)$$

- Problema de otimização: encontrar g que minimiza o erro total

Minimizando Erro Total

Conjunto de Treinamento

Example	DocID	Query	s_T	s_B	Judgment
Φ_1	37	linux	1	1	Relevant
Φ_2	37	penguin	0	1	Nonrelevant
Φ_3	238	system	0	1	Relevant
Φ_4	238	penguin	0	0	Nonrelevant
Φ_5	1741	kernel	1	1	Relevant
Φ_6	2094	driver	0	1	Relevant
Φ_7	3194	driver	1	0	Nonrelevant

- Computar score para cada instância do treinamento

$$score(d,q) = g \cdot s_T(d,q) + (1-g) \cdot s_B(d,q)$$

- Computar erro total: $\sum_j \epsilon(g, \Phi_j)$

Onde: $\epsilon(g, \Phi_j) = (r(d_j, q_j) - score(d_j, q_j))^2$

- Selecionar o valor de g que minimiza o erro

Minimizando Erro Total

- Computar score para cada instância do treinamento

$$score(d_1, q_1) = g \cdot 1 + (1-g) \cdot 1 = g + 1 - g = 1$$
$$score(d_2, q_2) = g \cdot 0 + (1-g) \cdot 1 = 0 + 1 - g = 1 - g$$
$$score(d_3, q_3) = g \cdot 0 + (1-g) \cdot 1 = 0 + 1 - g = 1 - g$$
$$score(d_4, q_4) = g \cdot 0 + (1-g) \cdot 0 = 0 + 0 = 0$$
$$score(d_5, q_5) = g \cdot 1 + (1-g) \cdot 1 = g + 1 - g = 1$$
$$score(d_6, q_6) = g \cdot 0 + (1-g) \cdot 1 = 0 + 1 - g = 1 - g$$
$$score(d_7, q_7) = g \cdot 1 + (1-g) \cdot 0 = g + 0 = g$$

- Computar erro total $\sum_j \epsilon(g, \Phi_j)$

$$(1-1)^2 + (0-1+g)^2 + (1-1+g)^2 + (0-0)^2 + (1-1)^2 + (1-1+g)^2 + (0-g)^2 = 0 + (-1+g)^2 + g^2 + 0 + 0 + g^2 + g^2 = 1 - 2g + 4g^2$$

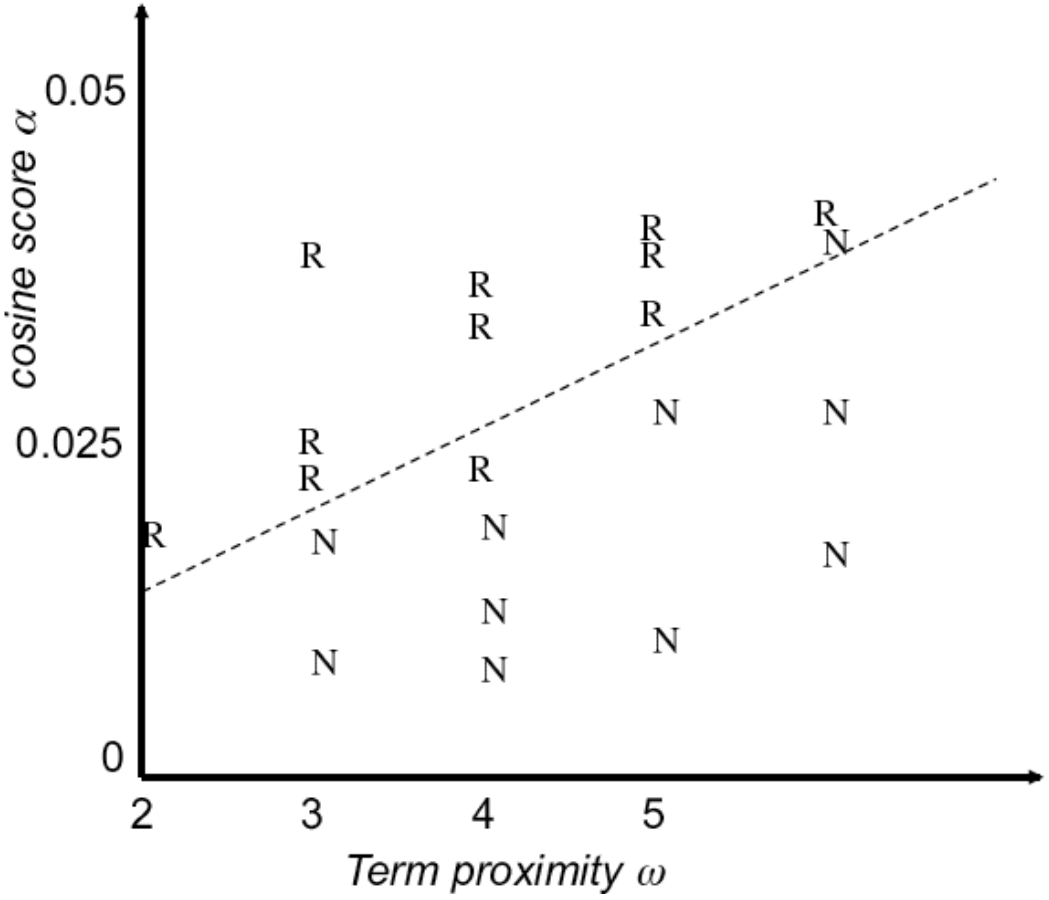
- Selecionar o valor de g que minimiza o erro
  - Setar a derivada para 0 -> g = 0.25

Usando Outras Features

- Similaridade de cosseno entre a consulta e o documento (α)
- Janela mínima que os termos da consulta aparecem no documento (w)

Example	DocID	Query	Cosine score	ω	Judgment
Φ_1	37	linux operating system	0.032	3	*relevant*
Φ_2	37	penguin logo	0.02	4	*nonrelevant*
Φ_3	238	operating system	0.043	2	*relevant*
Φ_4	238	runtime environment	0.004	2	*nonrelevant*
Φ_5	1741	kernel layer	0.022	3	*relevant*
Φ_6	2094	device driver	0.03	2	*relevant*
Φ_7	3191	device driver	0.027	5	*nonrelevant*
...	...	...	...	...	...

Representação Gráfica do Conjunto de Treinamento



$$Score(d, q) = Score(\alpha, \omega) = a\alpha + b\omega + c$$

L2R Features usadas por MSR

Zones: body, anchor, title, url, whole document

Features derived from standard IR models: query term number, query term ratio, length, idf, sum of term frequency, min of term frequency, max of term frequency, mean of term frequency, variance of term frequency, sum of length normalized term frequency, min of length normalized term frequency, max of length normalized term frequency, mean of length normalized term frequency, variance of length normalized term frequency, sum of tf-idf, min of tf-idf, max of tf-idf, mean of tf-idf, variance of tf-idf, boolean model, BM25

L2R Features usadas por MSR

- Language models: LMIR.DIR, LMIR.JM
- Específicas da Web: número de barras na URL, tamanho da URL, número de inlinks, número de outlinks, PageRank, SiteRank
- Features de uso: contagem de cliques da URL

Ranqueamento

- Vimos como dar pesos aos documentos
- Vamos ver como fazer ranqueamento de forma eficiente
- Objetivo: encontrar os k documentos mais próximos da consulta q

Ranking com Cosseno

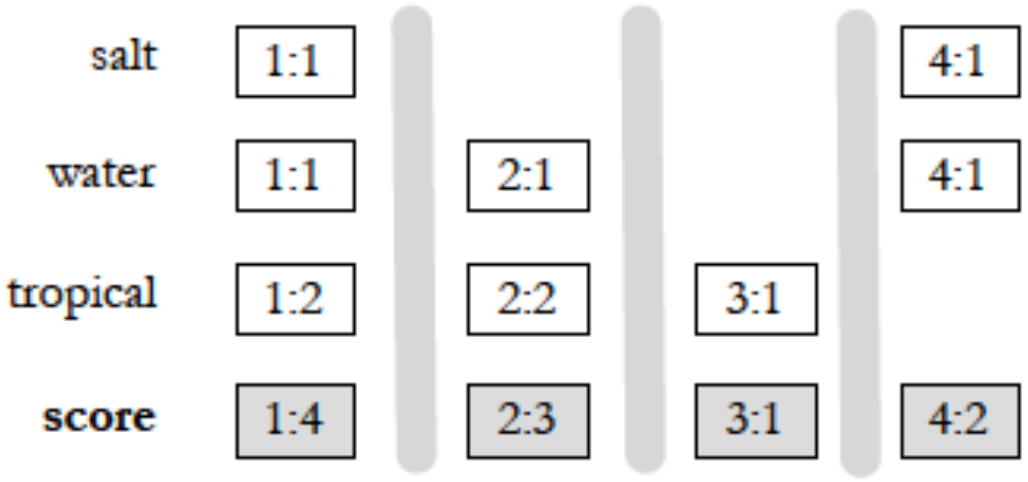
```
CosineScore(q)
 1   float Scores[N] = 0
 2   Initialize Length[N]
 3   for each query term t
 4   do calculate w_{t,q} and fetch postings list for t
 5       for each pair(d, tf_{t,d}) in postings list
 6       do Scores[d] += wf_{t,d} × w_{t,q}
 7   Read the array Length[d]
 8   for each d
 9   do Scores[d] = Scores[d]/Length[d]
10   return Top K components of Scores[]
```

Processamento de Consulta

- Como postings são lidos para a consulta
- Dois tipos
  - Document-at-a-time
  - Term-at-a-time

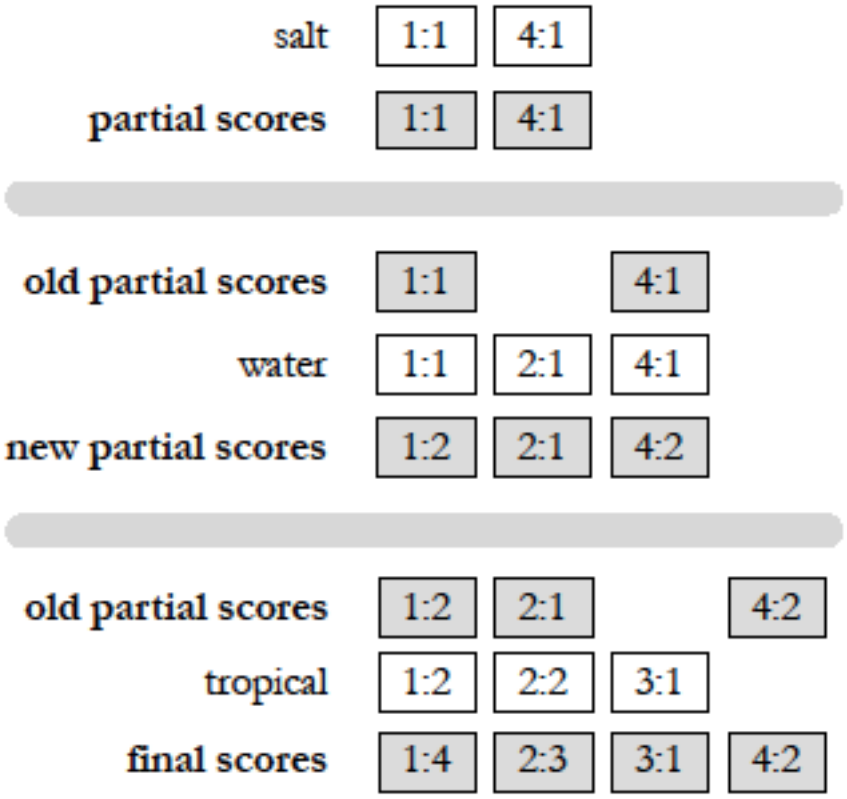
Document-at-a-Time

- Ex: consulta “salt water tropical”



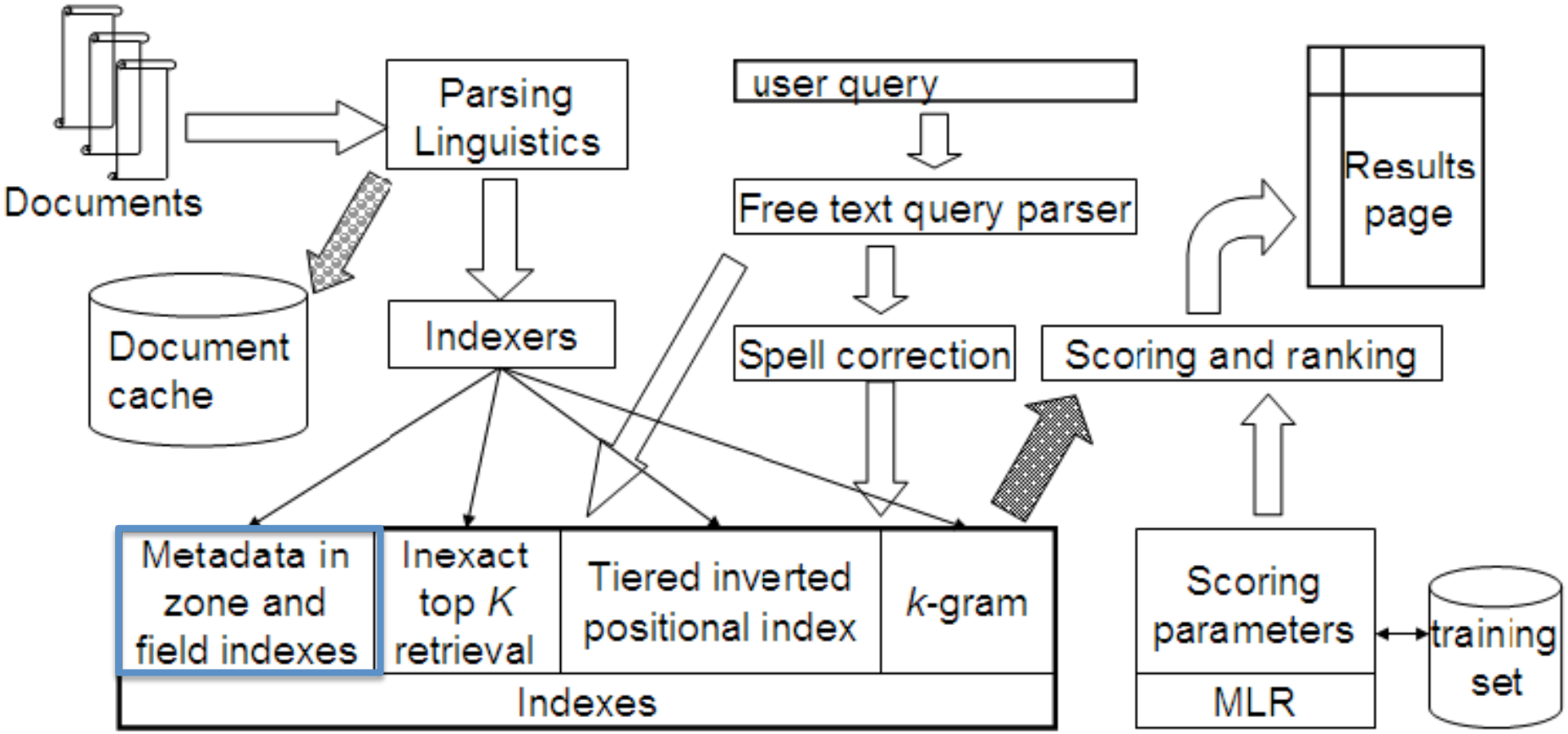
Term-at-a-Time

- Consulta: “salt water tropical”

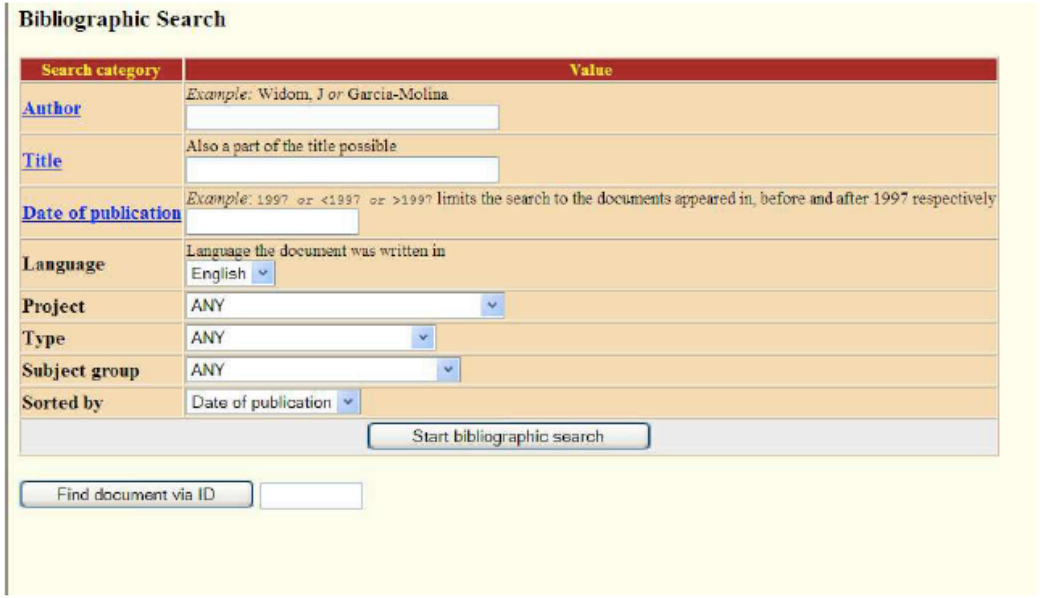


Chamada

Módulo de Busca



Field Index

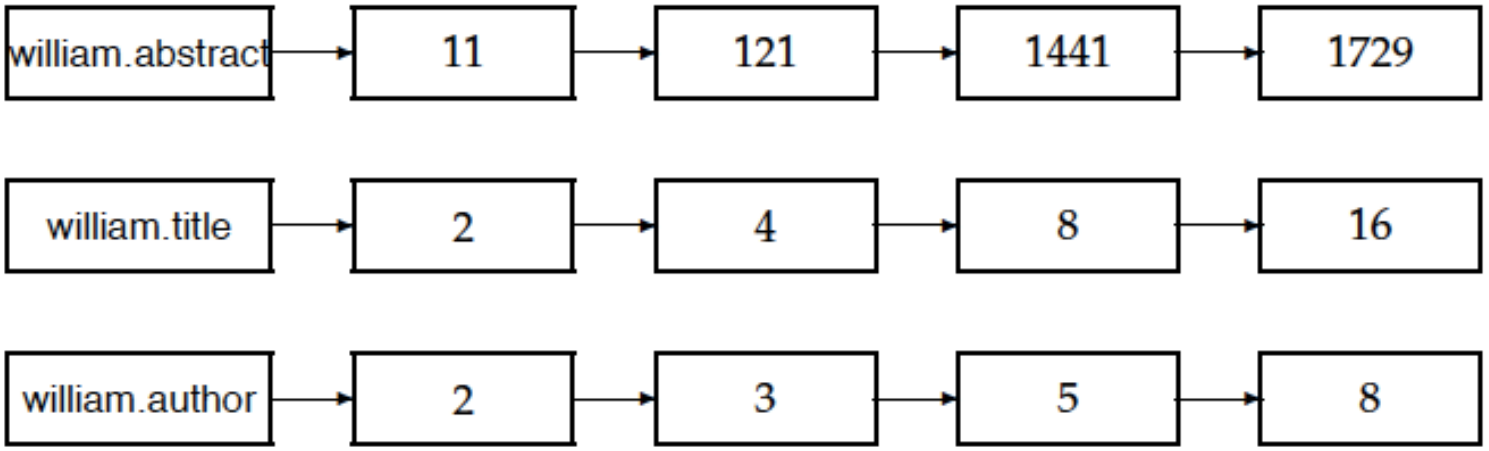


Bibliographic Search

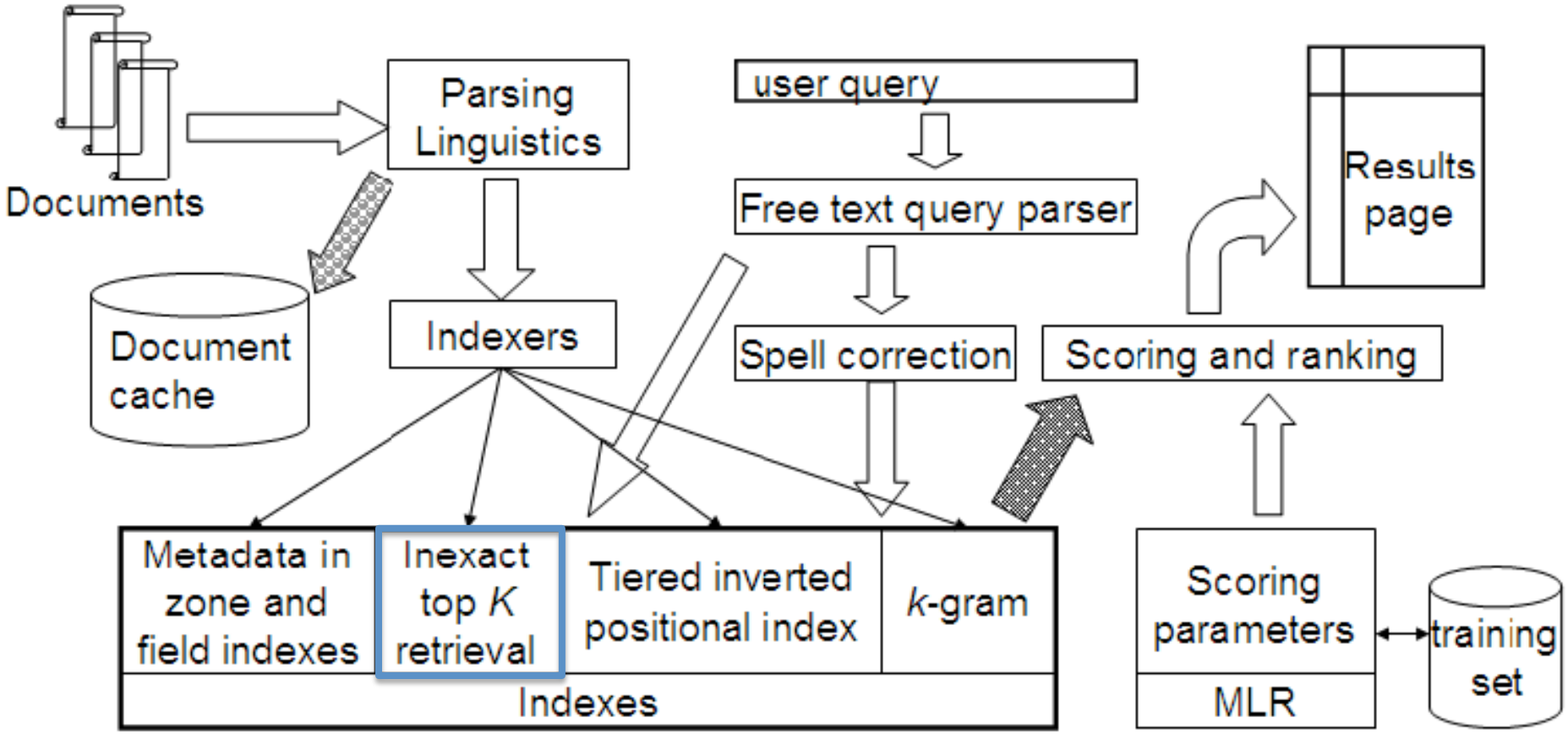
Search category	Value
Author	*Example:* Widom, J *or* Garcia-Molina
Title	Also a part of the title possible
Date of publication	*Example:* `1997` `or <1997` `or >1997` limits the search to the documents appeared in, before and after 1997 respectively
Language	Language the document was written in: English
Project	ANY
Type	ANY
Subject group	ANY
Sorted by	Date of publication

Start bibliographic search

Find document via ID



Módulo de Busca



Top-K Aproximado

- Até agora: como calcular exatamente os top-k documentos para uma consulta
- Custo: varrer todo o conjunto de documentos da base (ex. bilhões de documentos)
- Objetivo: diminuir o custo de computar os top-k sem impactar qualidade

Champion List

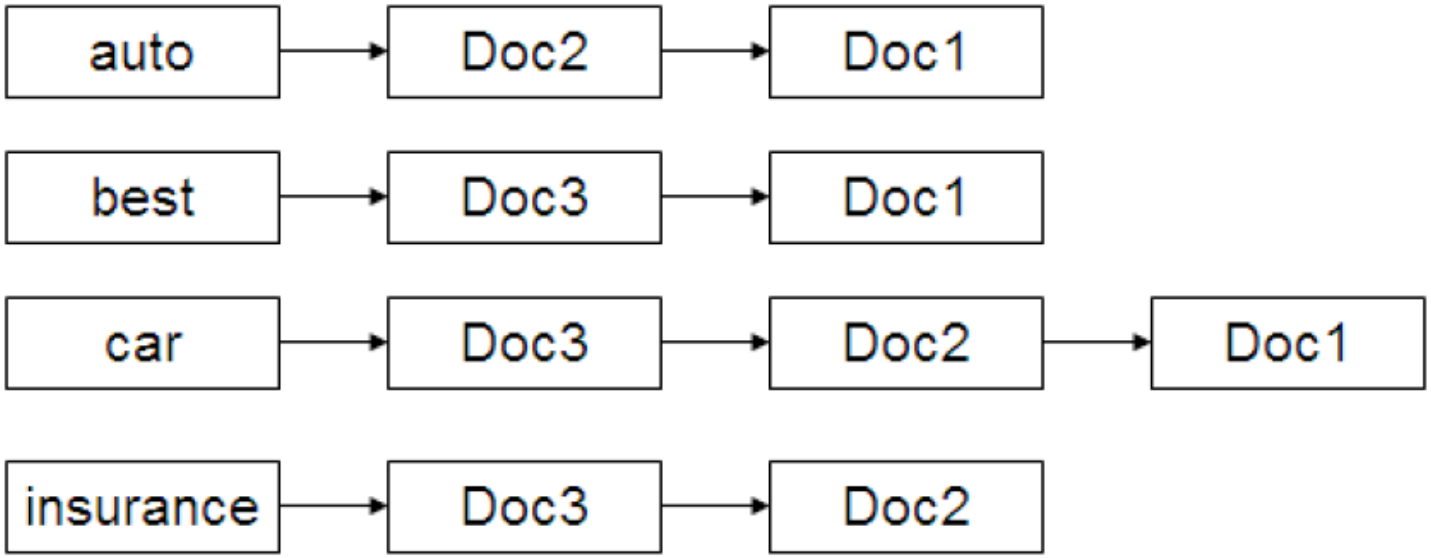
1. Encontar um conjunto A de documentos candidatos
   - A não necessariamente contém os top-k mas é provável de conter muitos deles
   - $K < |A| << N$
2. Retornar os top-k documentos em A

Champion List Baseada na Importância dos Termos

- Pré-computar para cada termo t no dicionário, o conjunto dos r documentos com maiores pesos para t
  - Tf-idf
- Conjuntos de tamanho r: champion lists
- Dada uma consulta q, faz-se a união dos conjuntos r com os termos em q
- Parâmetro r dependente de aplicação

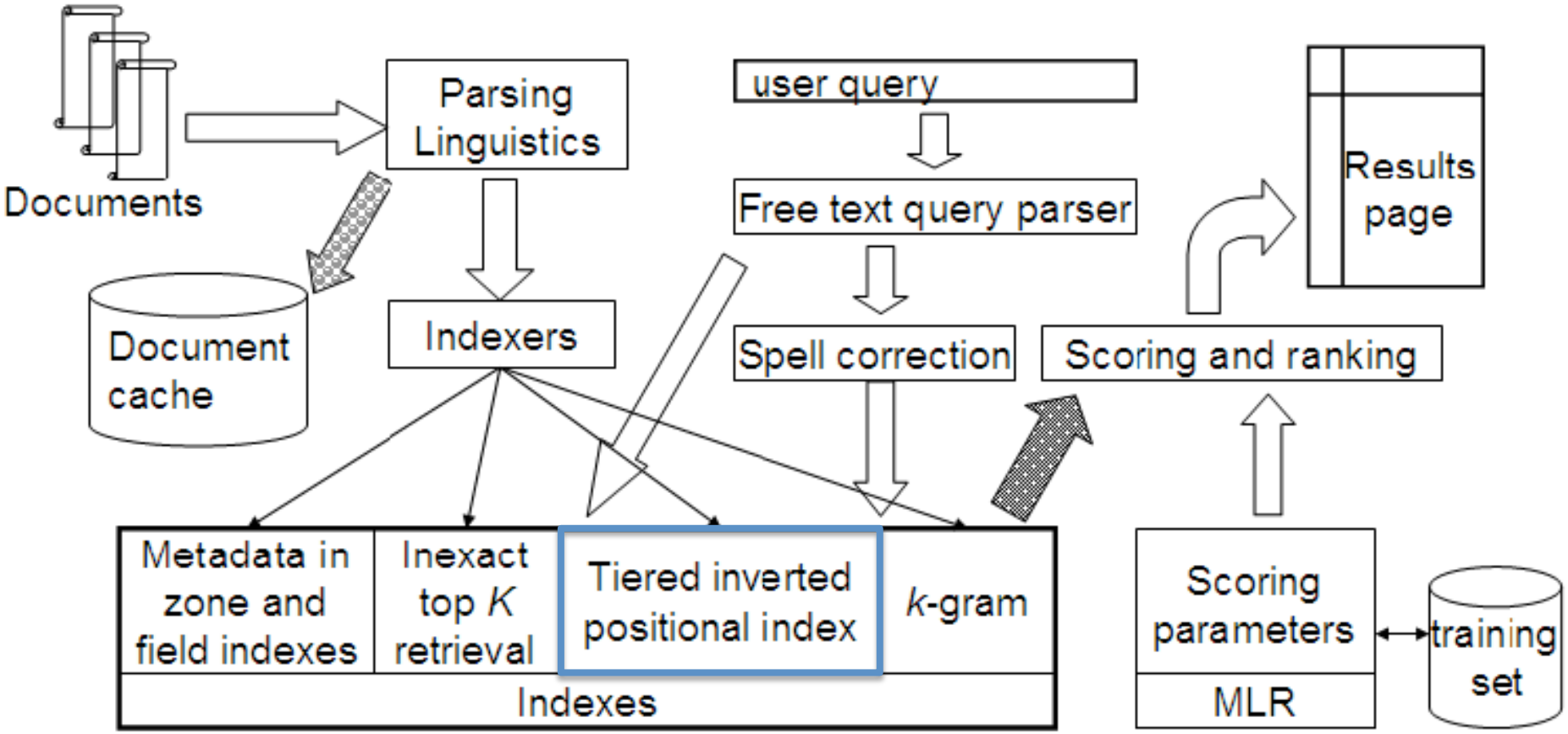
Champion List Baseada na Importância dos Documentos

- Medida de qualidade para cada documento: g(d)
- Independente de consulta -> estática
- Ex: número de avaliações positivas de um artigo na Web



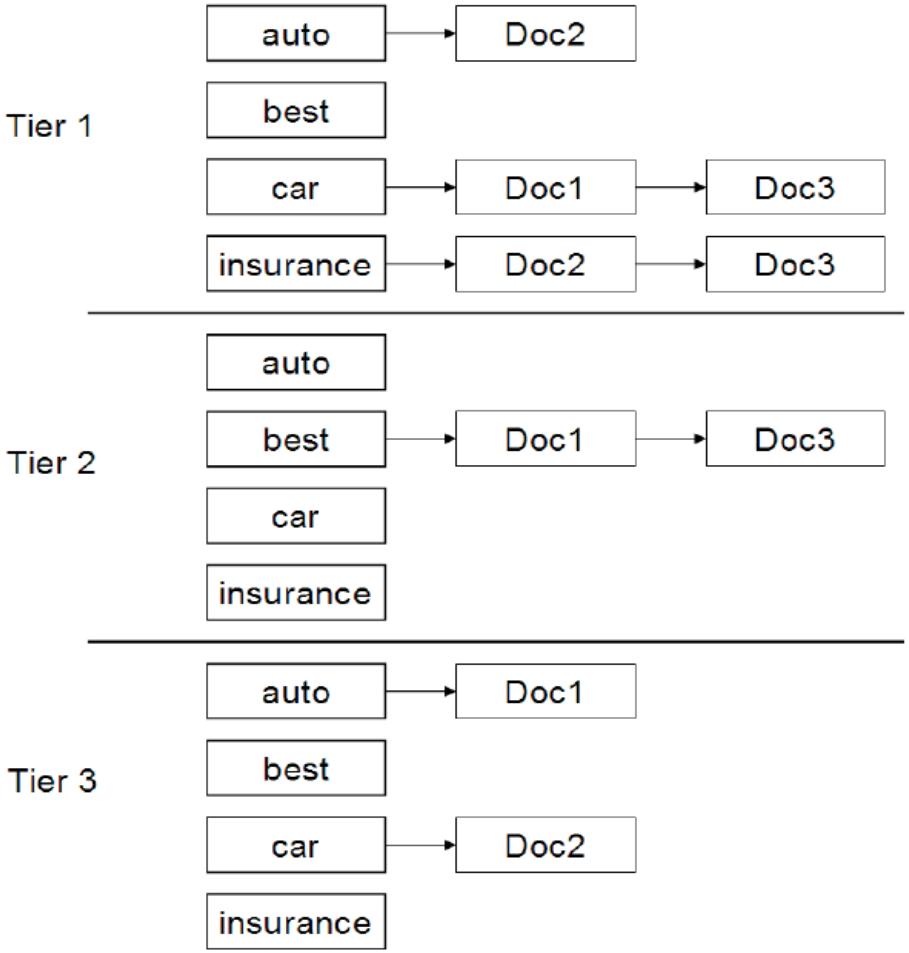
$g(3) > g(2) > g(1)$

Módulo de Busca



Tiered Indexes

- Ideia básica
  - Quanto mais alto o nível, maior a importância
  - Começa pelo nível mais alto
  - Se obtiver o número mínimo de desejado de resultados, retorna
  - Senão, desce pro próximo nível
- Exemplo 1:
  - Nível 1: índice com todos os títulos
  - Nível 2: Índice com o resto dos documentos
- Uma das causas da qualidade de busca do Google, junto com outro fatores (ex. âncora, proximidade)



Tiered Indexes - Radix