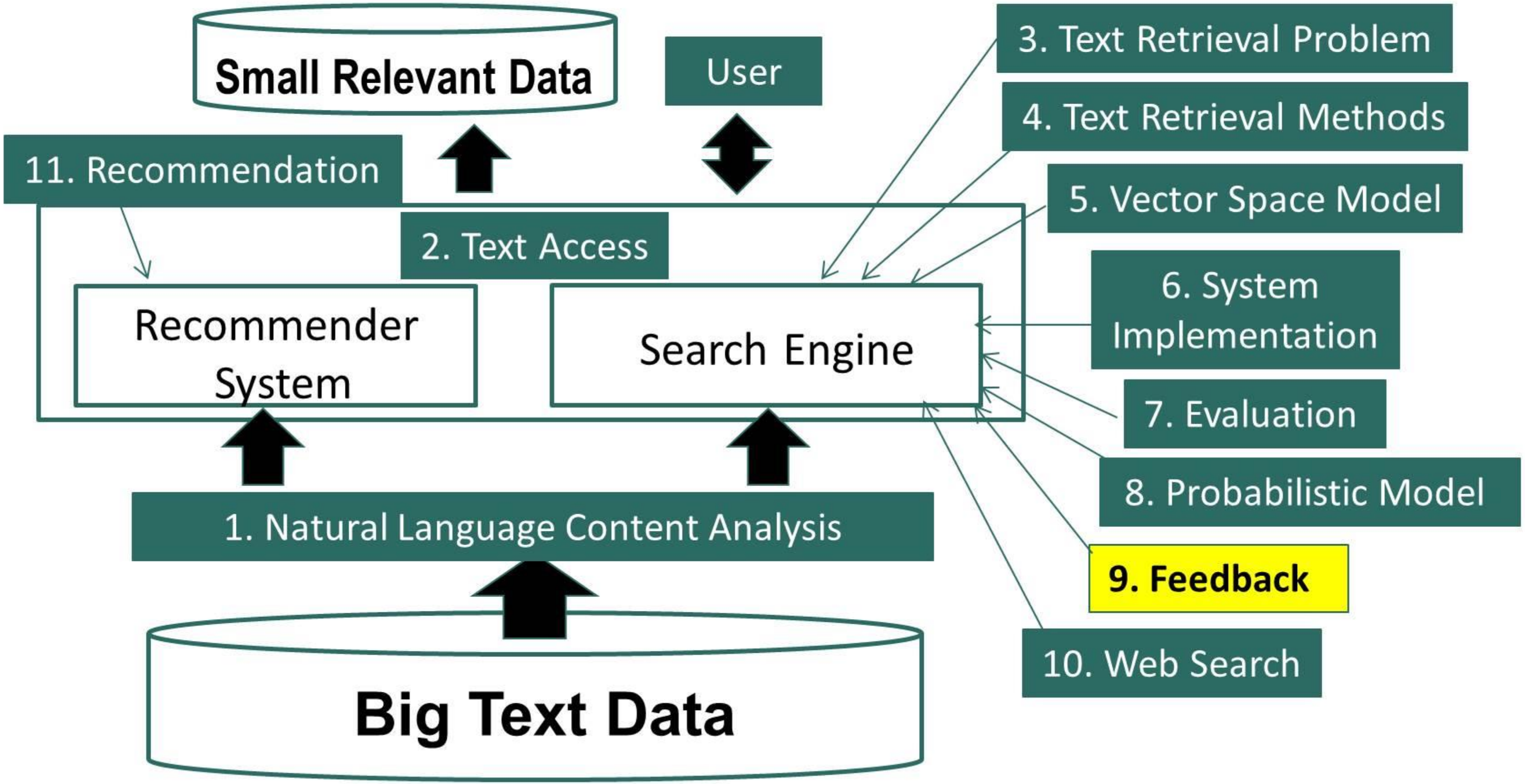
Retrieval Methods: Feedback in Text Retrieval

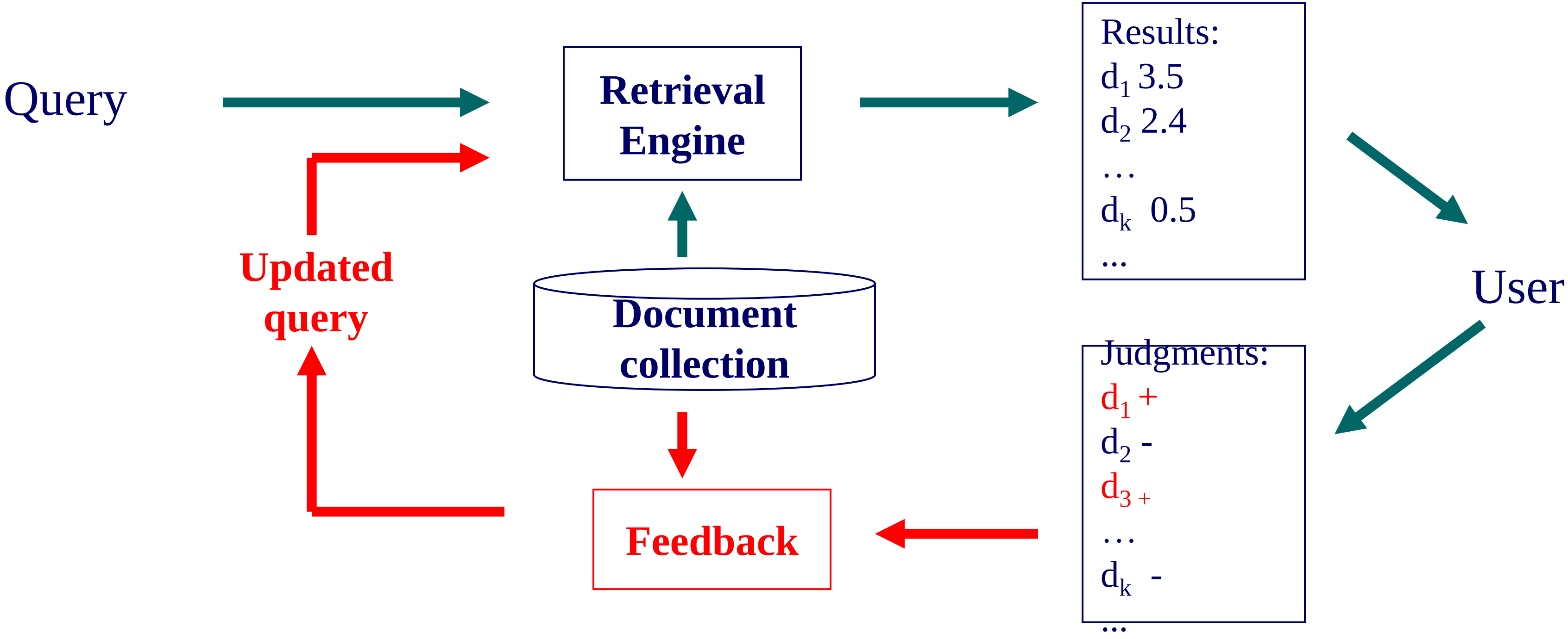
Text Retrieval Methods: Feedback in TR



Slide adapted from ChengXiang Zhai's presentation

Relevance Feedback

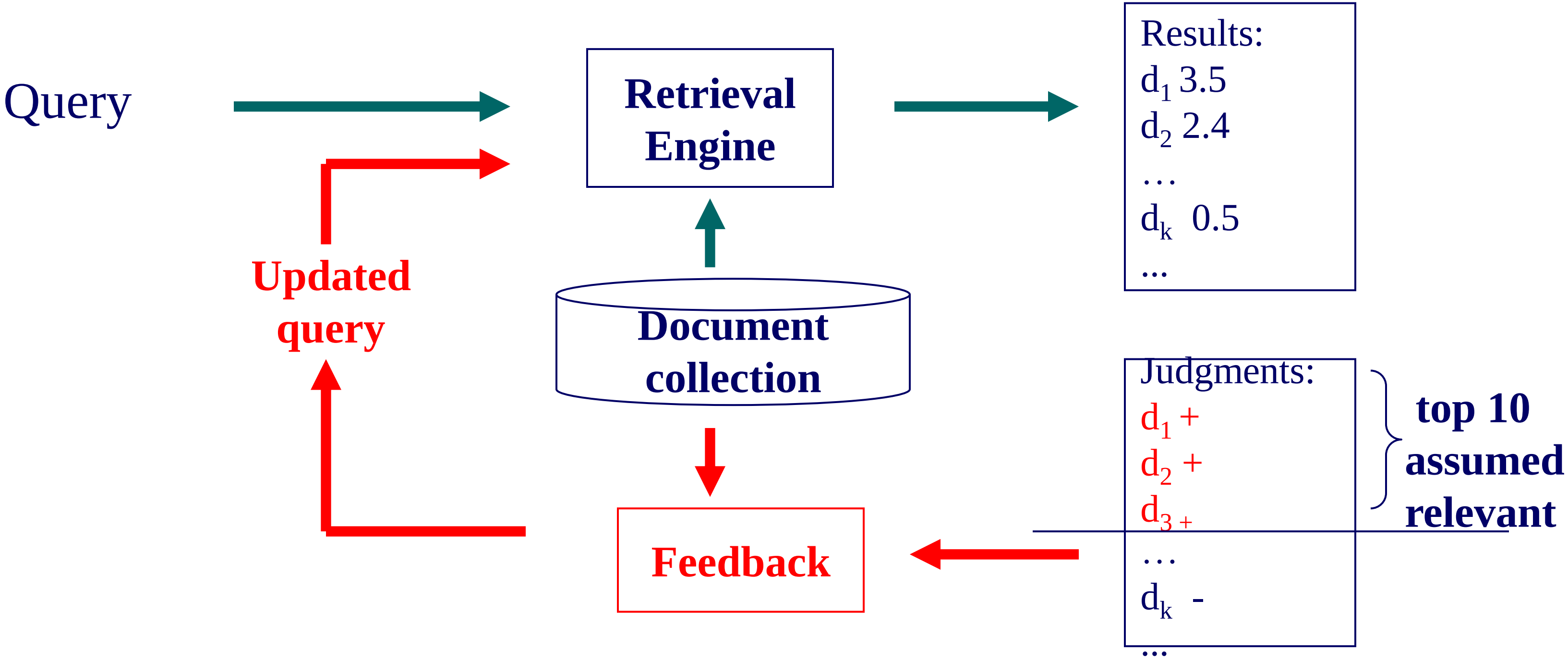
Users make explicit relevance judgments on the initial results
(judgments are reliable, but users don't want to make extra effort)



Slide adapted from ChengXiang Zhai's presentation

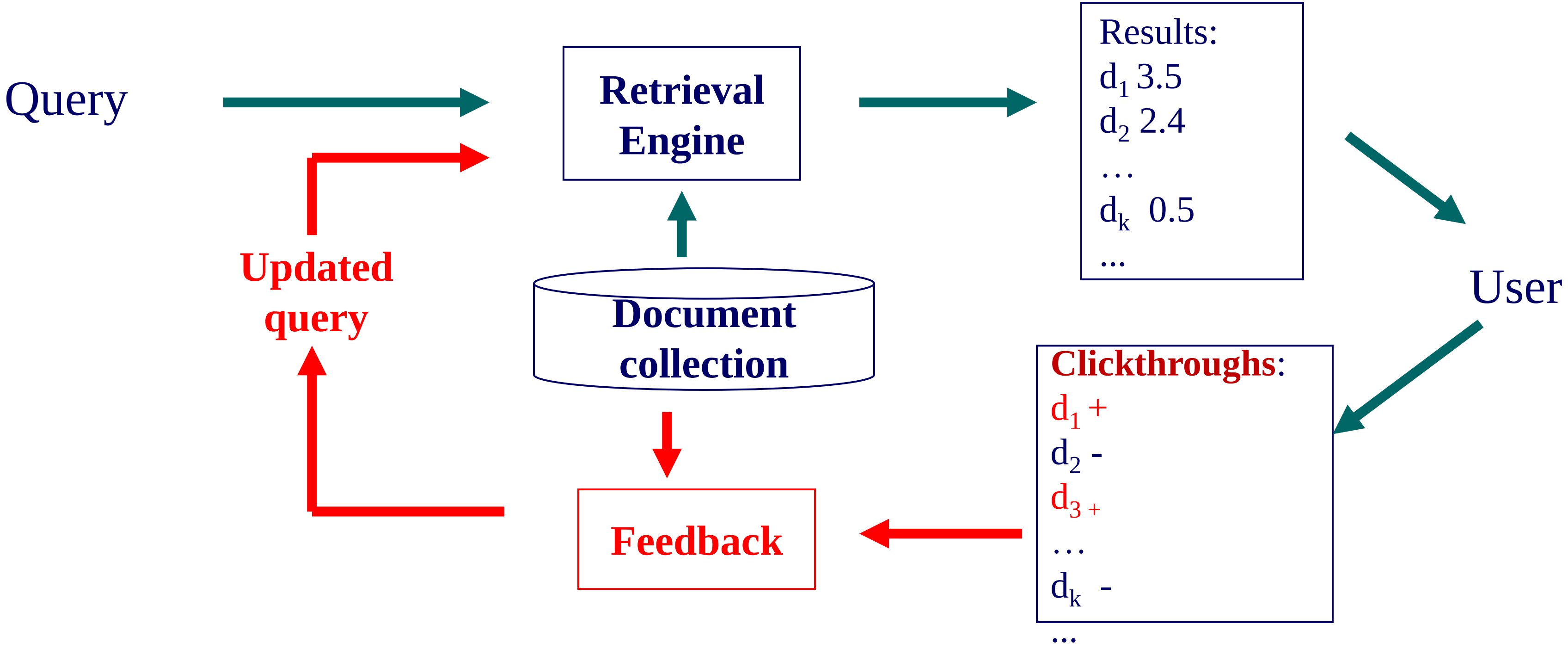
Pseudo/Blind/Automatic Feedback

Top-k initial results are simply assumed to be relevant
(judgments aren't reliable, but no user activity is required)



Implicit Feedback

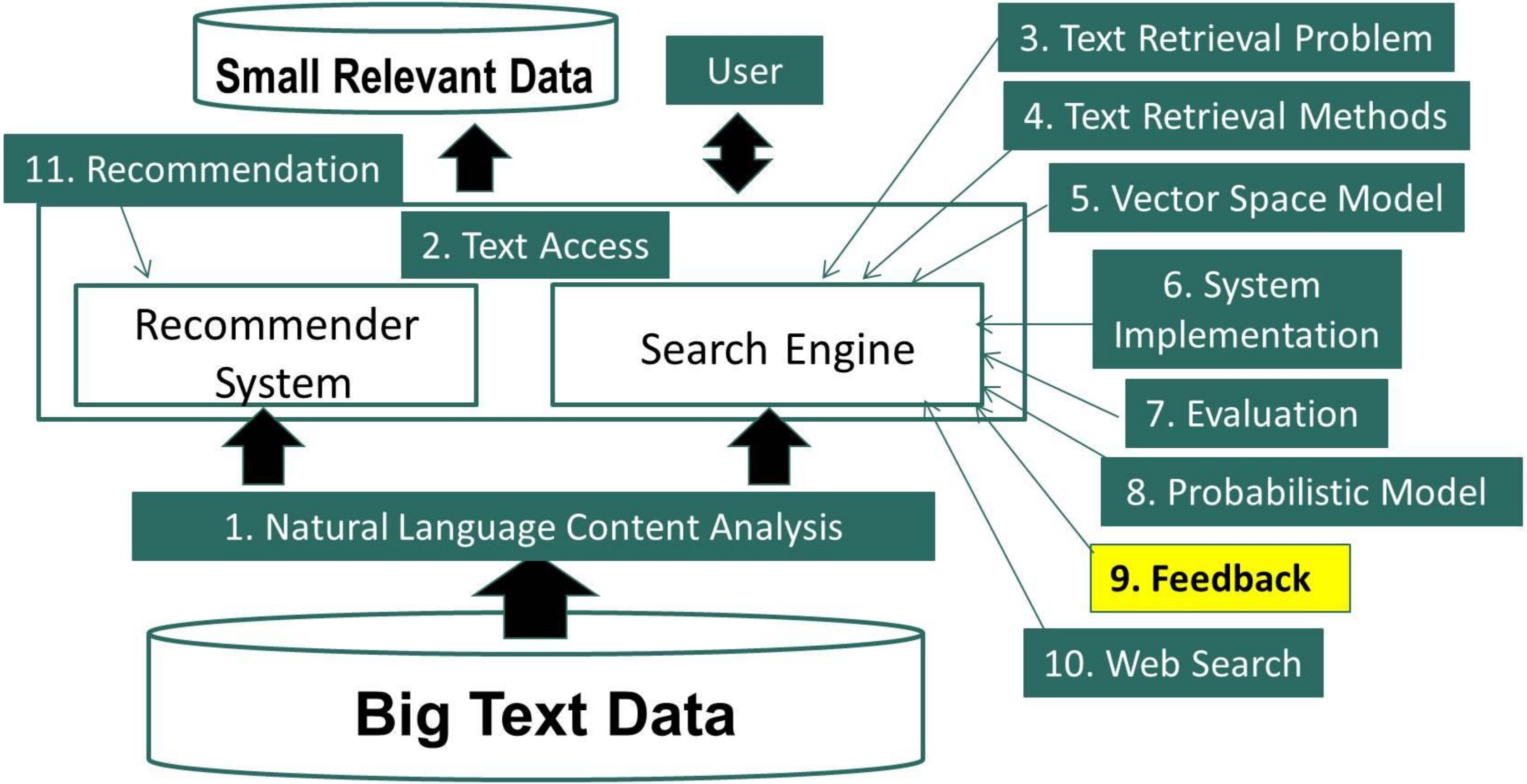
User-clicked docs are assumed to be relevant; skipped ones non-relevant (judgments aren't completely reliable, but no extra effort from users)



Slide adapted from ChengXiang Zhai's presentation

Feedback in Text Retrieval: Feedback in VSM

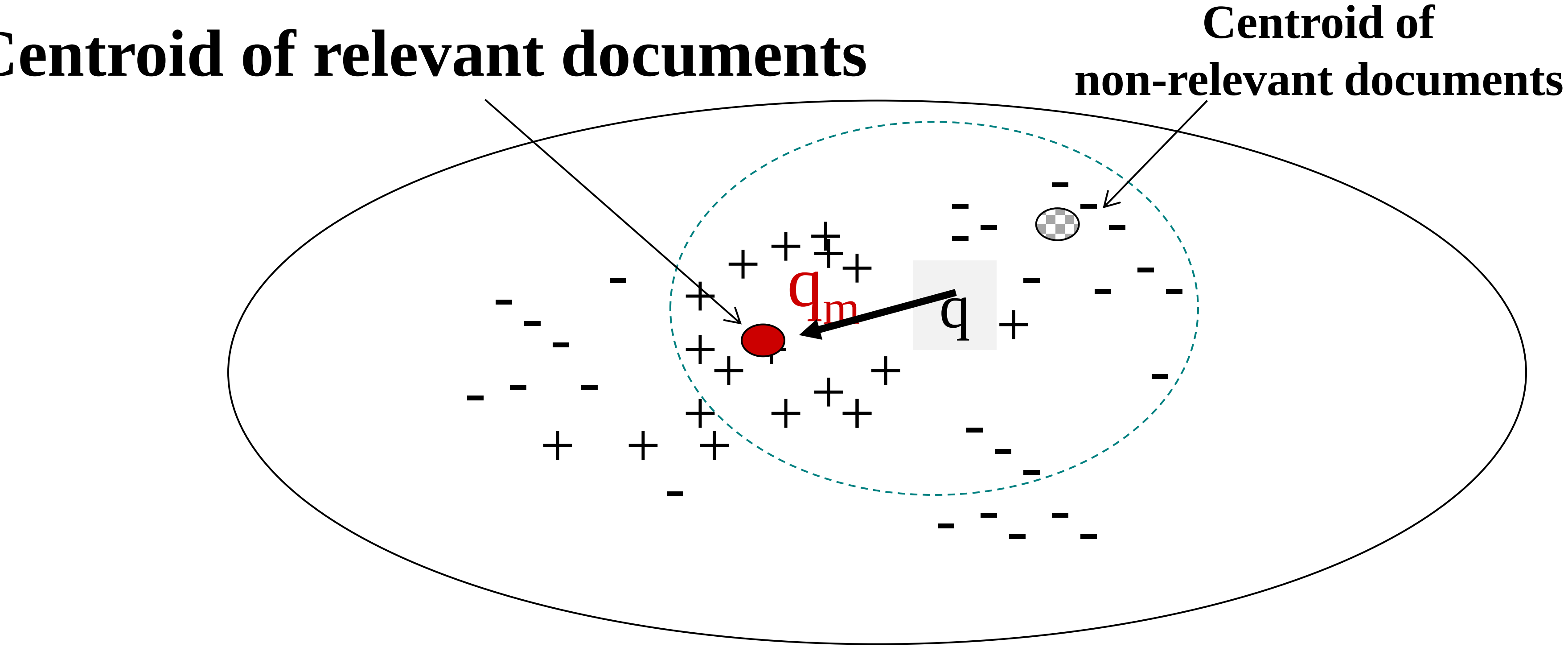
Feedback in Text Retrieval: Feedback in VSM



Feedback in Vector Space Model

- How can a TR system learn from examples to improve retrieval accuracy?
  - Positive examples: docs known to be relevant
  - Negative examples: docs known to be non-relevant
- General method: query modification
  - Adding new (weighted) terms (query expansion)
  - Adjusting weights of old terms

Rocchio Feedback: Illustration



Rocchio Feedback: Formula

Parameters

New query

$$\vec{q}_m = \alpha\,\vec{q} + \frac{\beta}{|D_r|}\sum_{\forall \vec{d}_j \in D_r} \vec{d}_j - \frac{\gamma}{|D_n|}\sum_{\forall \vec{d}_j \in D_n} \vec{d}_j$$

Origial query

Rel docs

Non-rel docs

Slide adapted from ChengXiang Zhai's presentation

Example of Rocchio Feedback

V= {news about presidential camp. food …. }

Query = "news about presidential campaign"

Q= (1, 1, 1, 1, 0, 0, …)

New Query Q'= (α*1+β*1.5-γ*1.5, α*1-γ*0.067, α*1+β*3.5, α*1+β*2.0-γ*2.6, -γ*1.3, 0, 0, …)

- D1= (1.5, 0.1, 0, 0, 0, 0, …)

D2

… **news about** organic food **campaign**…

- D2= (1.5, 0.1, 0, 2.0, 2.0, 0, …)

D3

… **news** of **presidential campaign** …

+ D3= (1.5, 0, 3.0, 2.0, 0, 0, …)

D4

+ Centroid Vector= ((1.5+1.5)/2, 0, (3.0+4.0)/2, (2.0+2.0)/2, 0, 0, …)
=(1.5 , 0, 3.5, 2.0, 0, 0,…)

+ D4= (1.5, 0, 4.0, 2.0, 0, 0, …)

- Centroid Vector= ((1.5+1.5+1.5)/3, (0.1+0.1+0)/3, 0, (0+2.0+6.0)/3, (0+2.0+2.0)/3, 0, …)
=(1.5 , 0.067, 0, 2.6, 1.3, 0,…)

- D5= (1.5, 0, 0, 6.0, 2.0, 0, …)

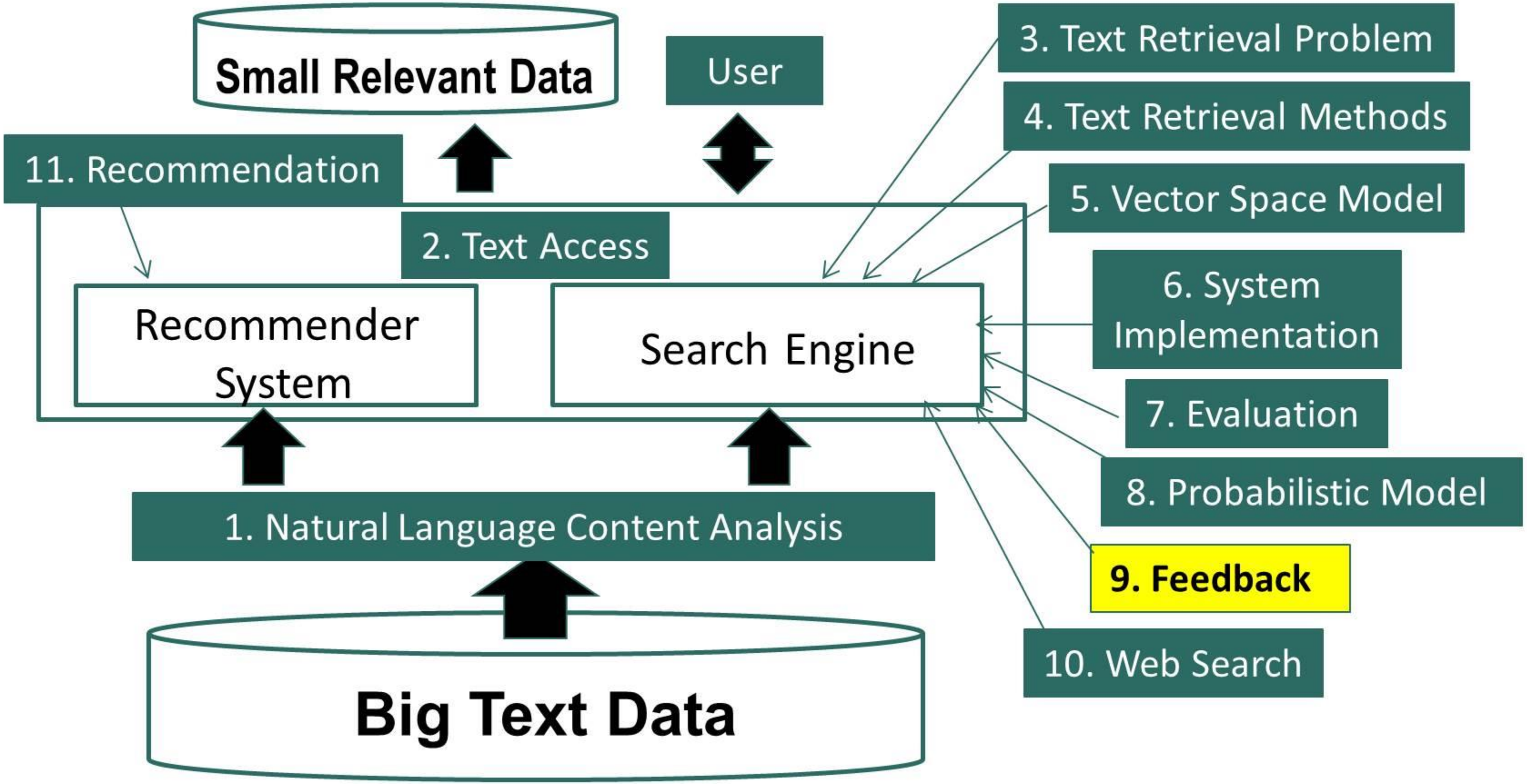
Rocchio in Practice

- Negative (non-relevant) examples are not very important (why?)
- Often truncate the vector (i.e., consider only a small number of words that have highest weights in the centroid vector) (efficiency concern)
- Avoid “over-fitting” (keep relatively high weight on the original query weights) (why?)
- Can be used for relevance feedback and pseudo feedback (β should be set to a larger value for relevance feedback than for pseudo feedback)
- Usually robust and effective

Slide adapted from ChengXiang Zhai's presentation

Feedback in Text Retrieval: Feedback in LM

Feedback in Text Retrieval: Feedback in LM



Feedback with Language Models

- Query likelihood method can’t naturally support relevance feedback
- Solution:
  – Kullback-Leibler (KL) divergence retrieval model as a generalization of query likelihood
  – Feedback is achieved through query model estimation/updating

Kullback-Leibler (KL) Divergence Retrieval Model

Query Likelihood

$$f(q,d) = \sum_{\substack{w_i \in d \\ w_i \in q}} \boxed{c(w,q)}[\log \frac{p_{Seen}(w_i \mid d)}{\alpha_d \, p(w_i \mid C)}] + n \log \alpha_d$$

KL-divergence (cross entropy)

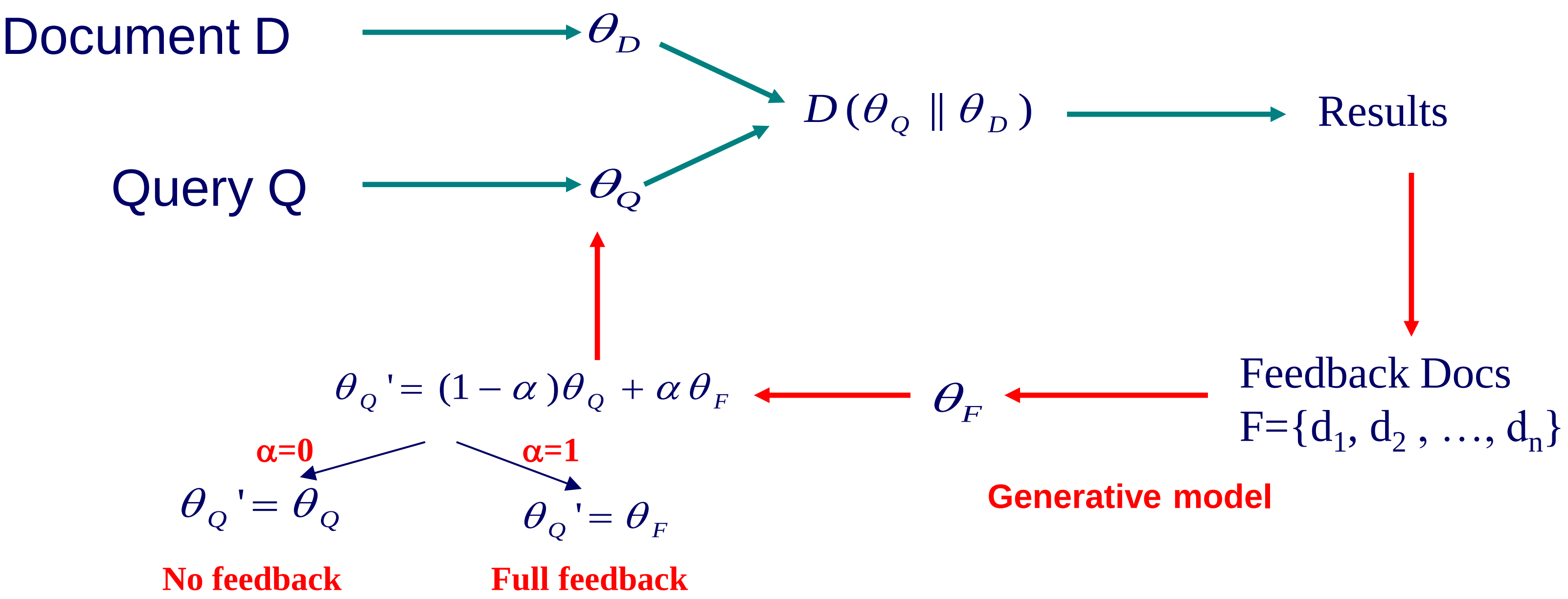
$$f(q,d) = \sum_{w \in d, p(w|\theta_Q) > 0} \boxed{[p(w \mid \hat{\theta}_Q)}\log \frac{p_{seen}(w \mid d)}{\alpha_d p(w \mid C)}] + \log \alpha_d$$

Query LM

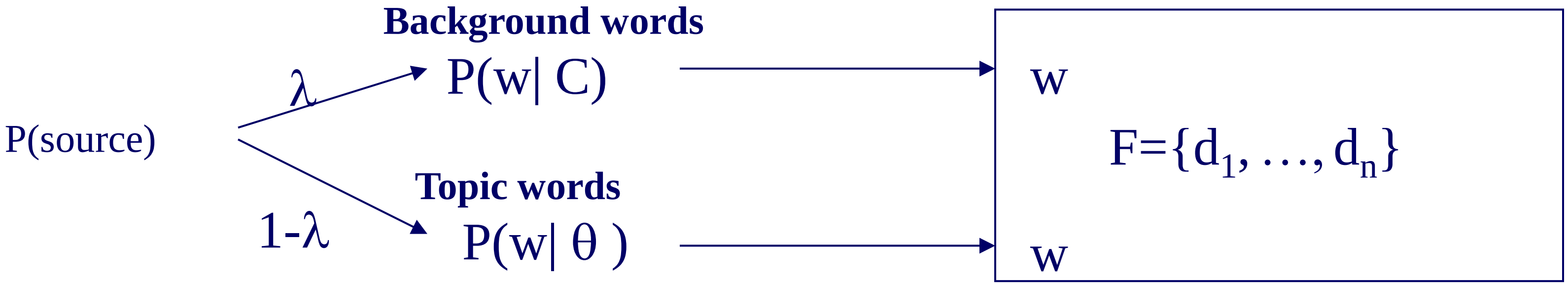
$$p(w \mid \hat{\theta}_Q) = \frac{c(w,Q)}{\mid Q \mid}$$

Slide adapted from ChengXiang Zhai's presentation

Feedback as Model Interpolation



Generative Mixture Model



$$\log p(F|\theta) = \sum_i \sum_w c(w;d_i) \log[(1-\lambda)p(w|\theta) + \lambda p(w|C)]$$

Maximum Likelihood $\theta_F = \arg\max_{\theta} \log p(F|\theta)$

λ = Noise in feedback documents

Example of Pseudo-Feedback Query Model

Query: "airport security"

Mixture model approach

Web database

Top 10 docs

$\lambda=0.9$

W	$p(W\|\theta_F)$
security	0.0558
airport	0.0546
beverage	0.0488
alcohol	0.0474
bomb	0.0236
terrorist	0.0217
author	0.0206
license	0.0188
bond	0.0186
counter-terror	0.0173
terror	0.0142
newsnet	0.0129
attack	0.0124
operation	0.0121
headline	0.0121

$\lambda=0.7$

W	$p(W\|\theta_F)$
the	0.0405
security	0.0377
airport	0.0342
beverage	0.0305
alcohol	0.0304
to	0.0268
of	0.0241
and	0.0214
author	0.0156
bomb	0.0150
terrorist	0.0137
in	0.0135
license	0.0127
state	0.0127
by	0.0125

Slide adapted from ChengXiang Zhai's presentation

Summary of Feedback in Text Retrieval

- Feedback = learn from examples
- Three major feedback scenarios
  - Relevance, pseudo, and implicit feedback
- Rocchio for VSM
- Query model estimation for LM
  - Mixture model
  - Many other methods (e.g., relevance model) have been proposed [Zhai 08]

Slide adapted from ChengXiang Zhai's presentation

Additional Readings

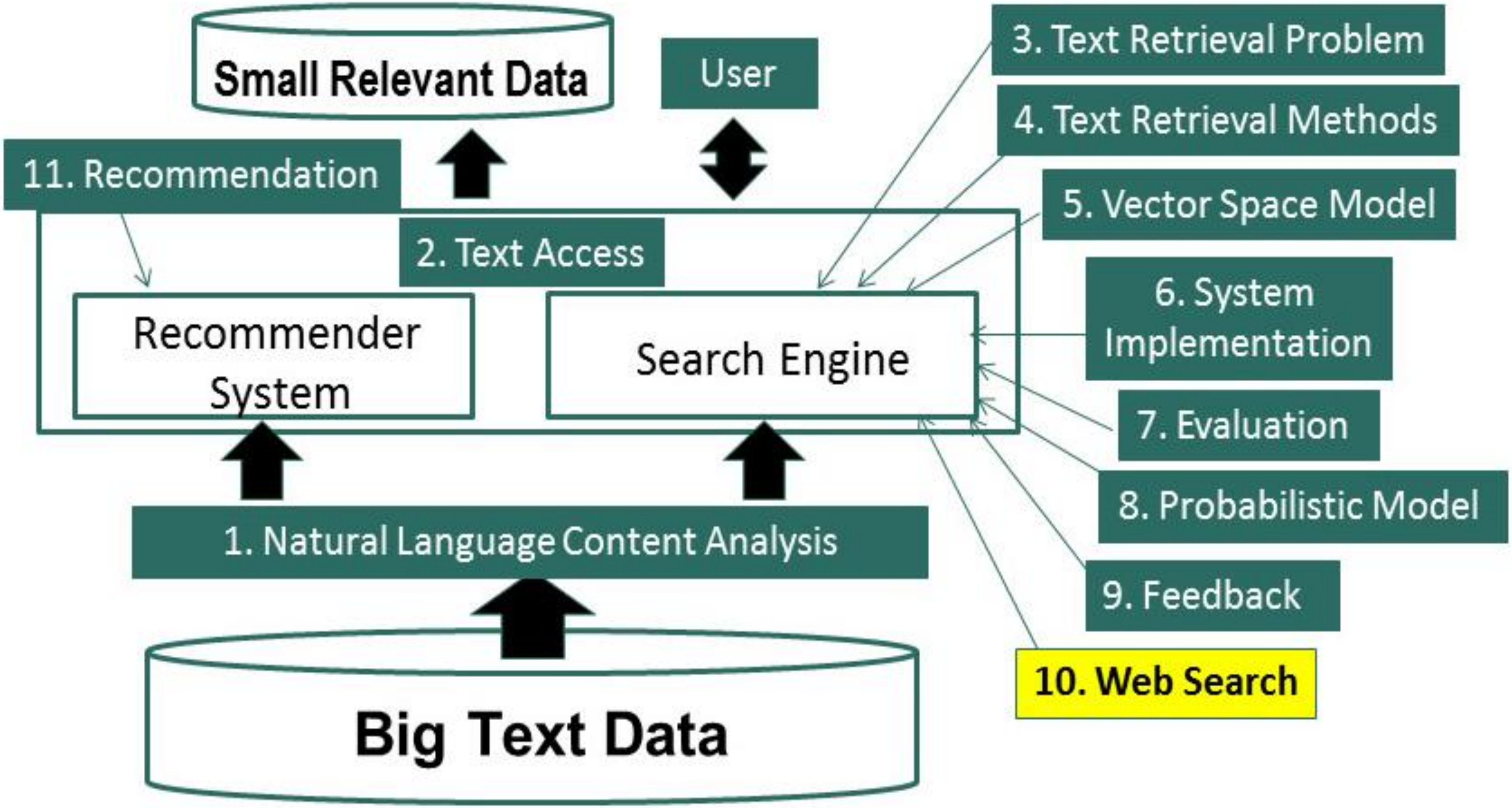
- ChengXiang Zhai, *Statistical Language Models for Information Retrieval* (Synthesis Lectures Series on Human Language Technologies), Morgan & Claypool Publishers, 2008.

http://www.morganclaypool.com/doi/abs/10.2200/S00158ED1V01Y200811HLT001

- Victor Lavrenko and W. Bruce Croft. 2001. Relevance based language models. In *Proceedings of ACM SIGIR 2011.*

Web Search

Course Schedule



Slide adapted from ChengXiang Zhai's presentation

Web Search: Challenges & Opportunities

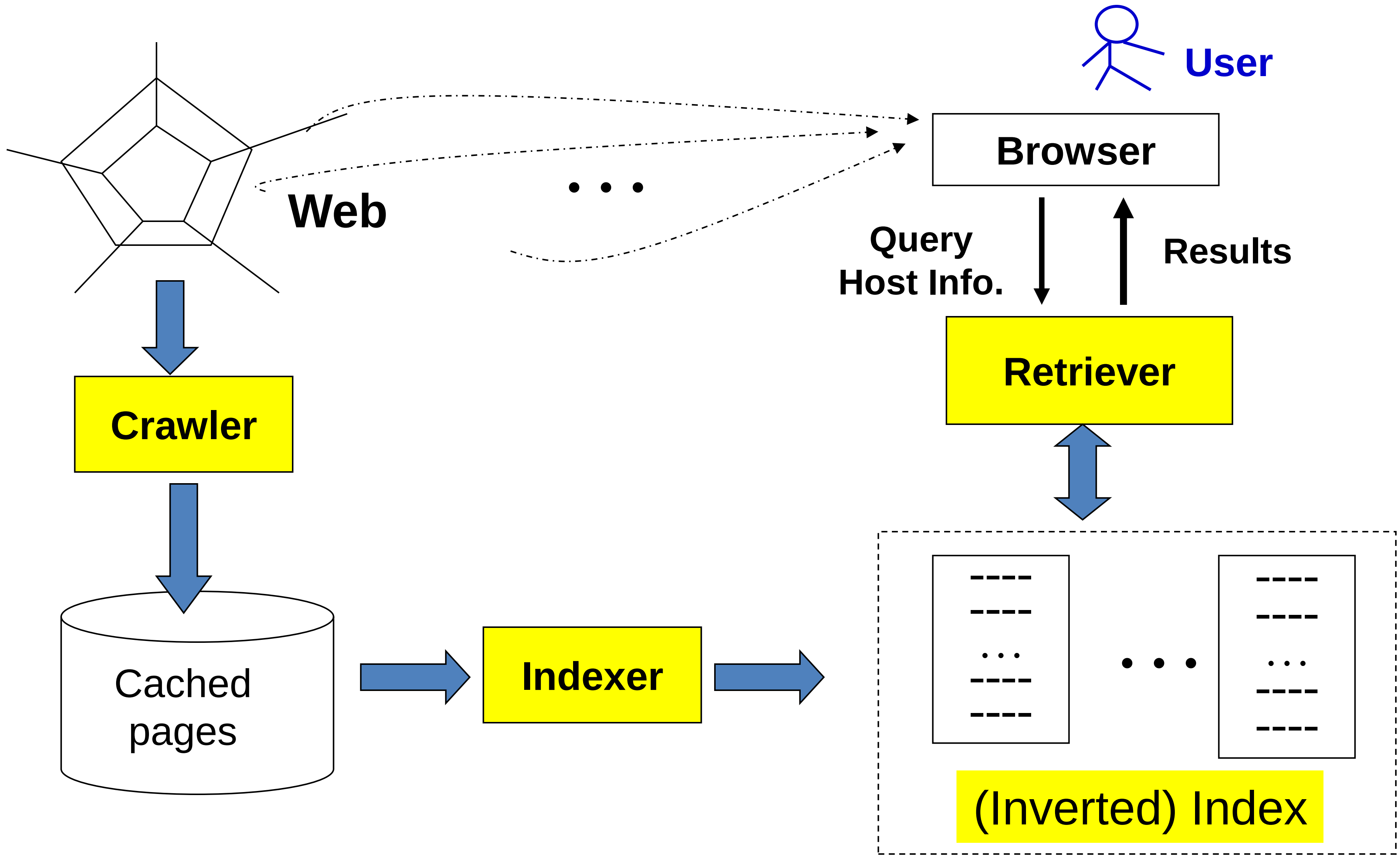
- Challenges
  - Scalability
    - How to handle the size of the Web and ensure completeness of coverage?
    - How to serve many user queries quickly?
  - Low quality information and spams
  - Dynamics of the Web
    - New pages are constantly created and some pages may be updated very quickly
- Opportunities
  - many additional heuristics (e.g., links) can be leveraged to improve search accuracy

➔ Parallel indexing & searching (MapReduce)

➔ Spam detection & Robust ranking

➔ Link analysis & multi-feature ranking

Basic Search Engine Technologies



Slide adapted from ChengXiang Zhai's presentation

Component I: Crawler/Spider/Robot

- Building a “toy crawler” is easy
  - Start with a set of “seed pages” in a priority queue
  - Fetch pages from the web
  - Parse fetched pages for hyperlinks; add them to the queue
  - Follow the hyperlinks in the queue
- A real crawler is much more complicated…
  - Robustness (server failure, trap, etc.)
  - Crawling courtesy (server load balance, robot exclusion, etc.)
  - Handling file types (images, PDF files, etc.)
  - URL extensions (cgi script, internal references, etc.)
  - Recognize redundant pages (identical and duplicates)
  - Discover “hidden” URLs (e.g., truncating a long URL)

Major Crawling Strategies

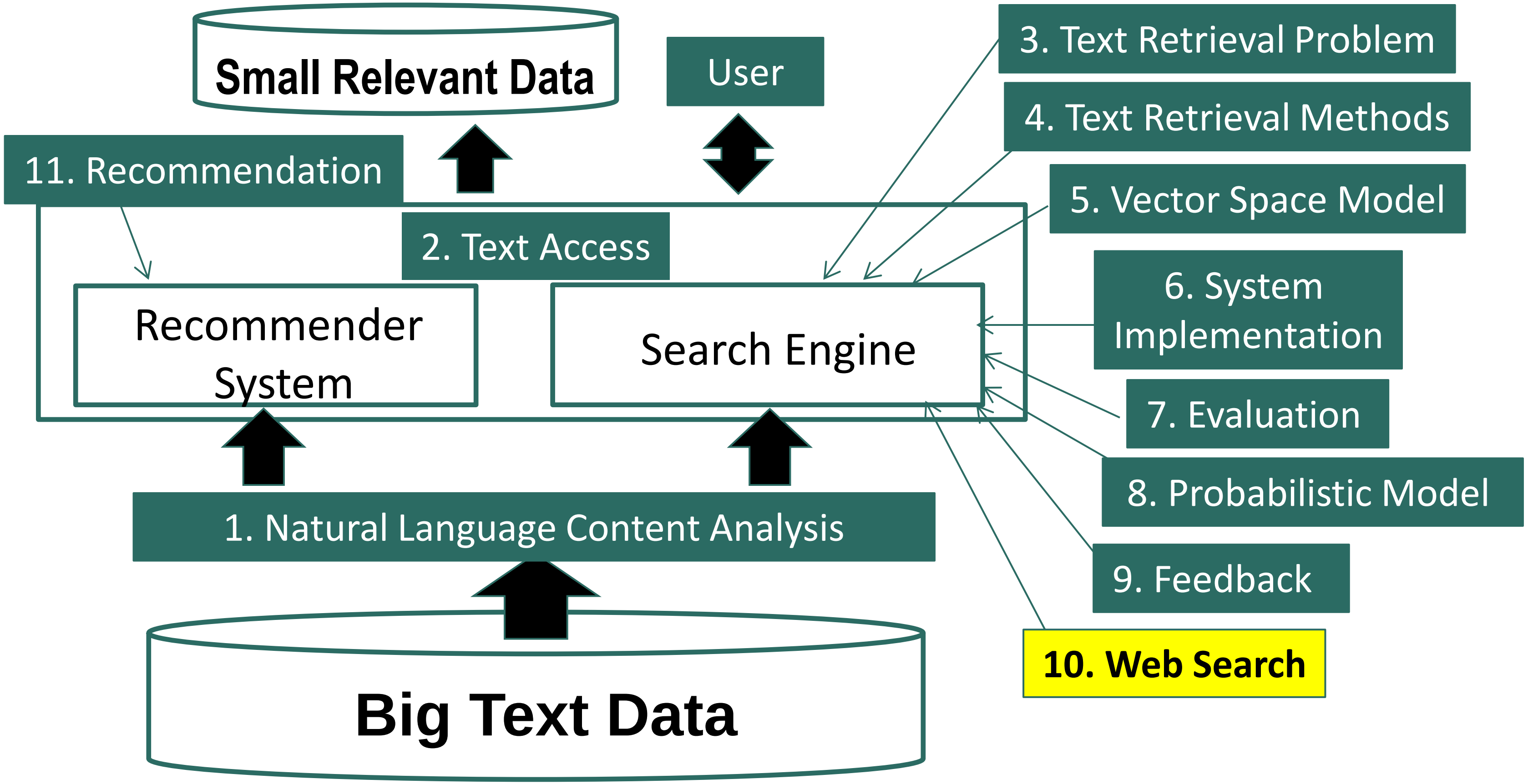
- Breadth-First is common (balance server load)
- Parallel crawling is natural
- Variation: focused crawling
  - Targeting at a subset of pages (e.g., all pages about “automobiles”)
  - Typically given a query
- How to find new pages (they may not linked to an old page!)
- Incremental/repeated crawling
  - Need to minimize resource overhead
  - Can learn from the past experience (updated daily vs. monthly)
  - Target at : 1) frequently updated pages; 2) frequently accessed pages

Summary

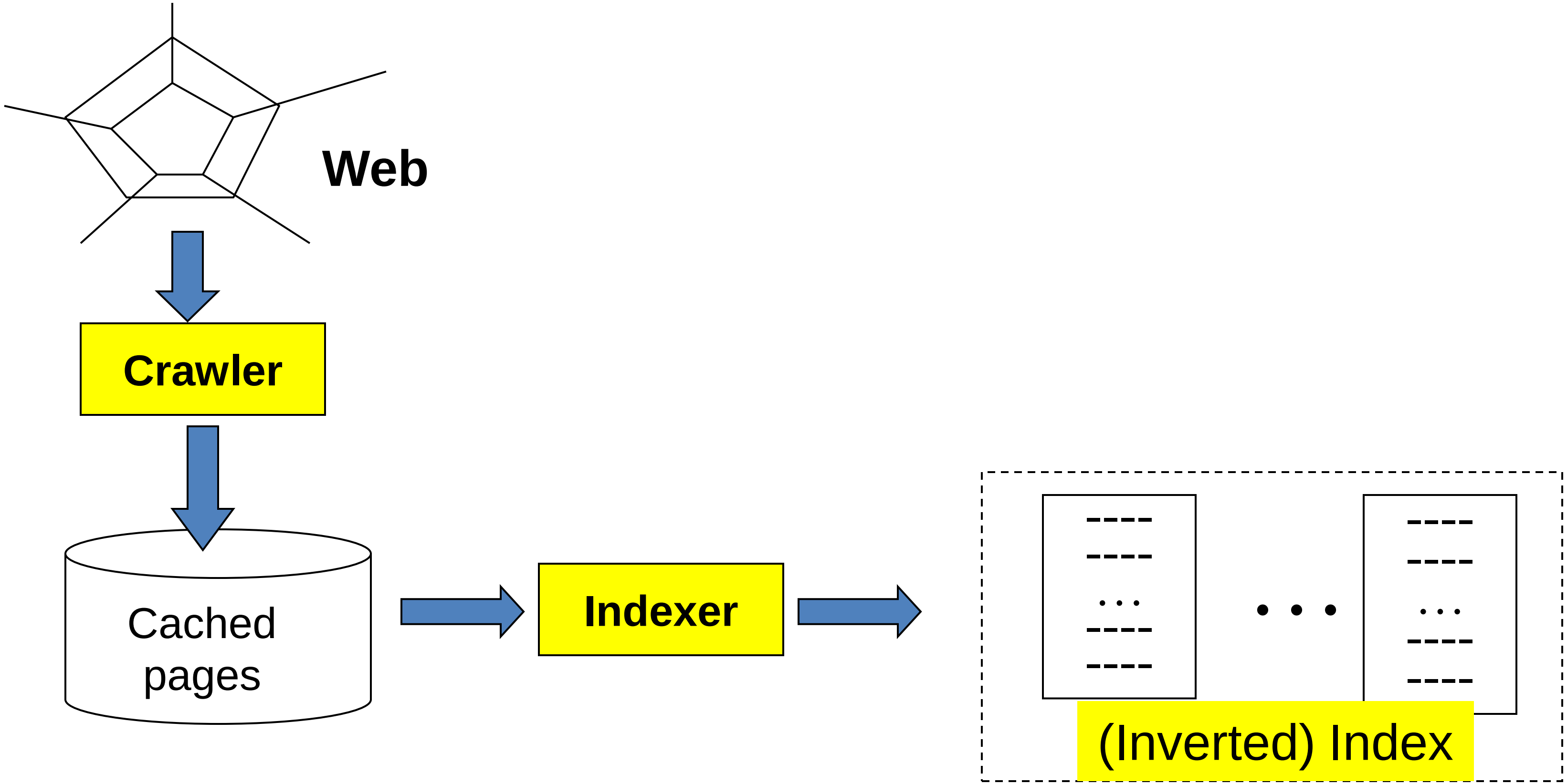
- Web search is one of the most important applications of text retrieval
  - New challenges: scalability, efficiency, quality of information
  - New opportunities: rich link information, layout, etc
- Crawler is an essential component of Web search applications
  - Initial crawling: complete vs. focused
  - Incremental crawling: resource optimization

Web Search: Web Indexing

Web Search: Web Indexing



Basic Search Engine Technologies



Slide adapted from ChengXiang Zhai's presentation

Overview of Web Indexing

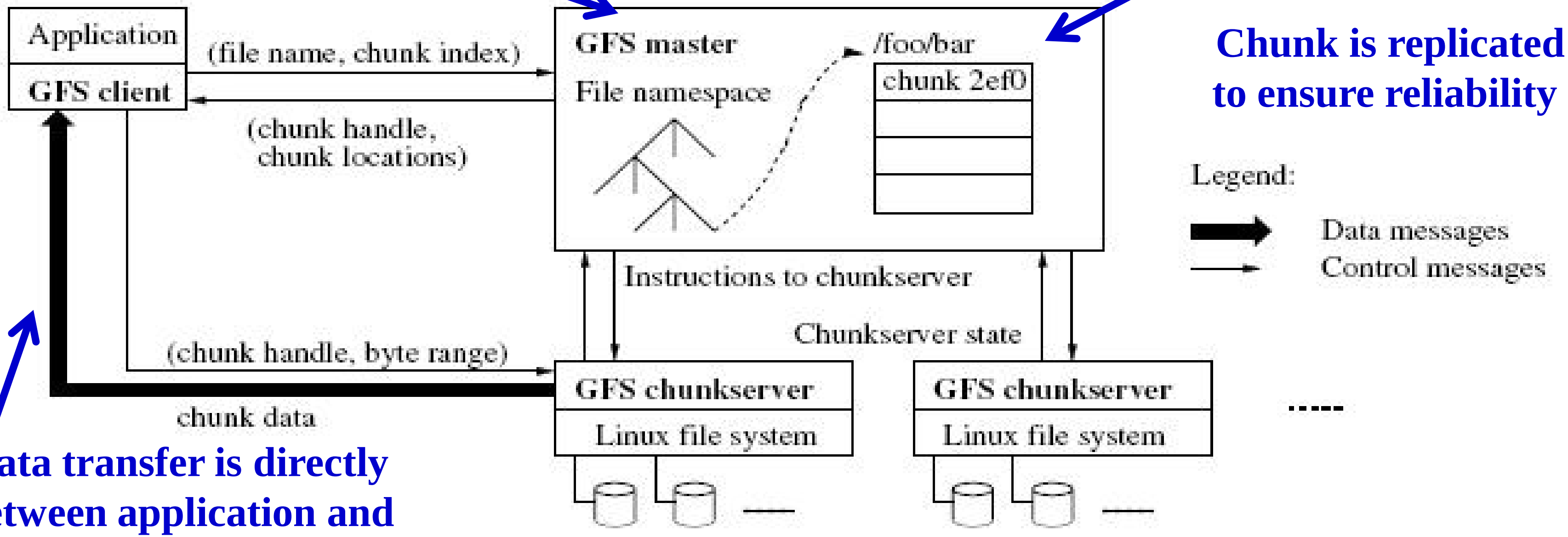
- Standard IR techniques are the basis, but insufficient
  - Scalability
  - Efficiency
- Google's contributions:
  - Google File System (GFS): distributed file system
  - MapReduce: Software framework for parallel computation
  - Hadoop: Open source implementation of MapReduce

GFS Architecture

Simple centralized management

Fixed chunk size (64 MB)

Chunk is replicated to ensure reliability



Data transfer is directly between application and chunk servers

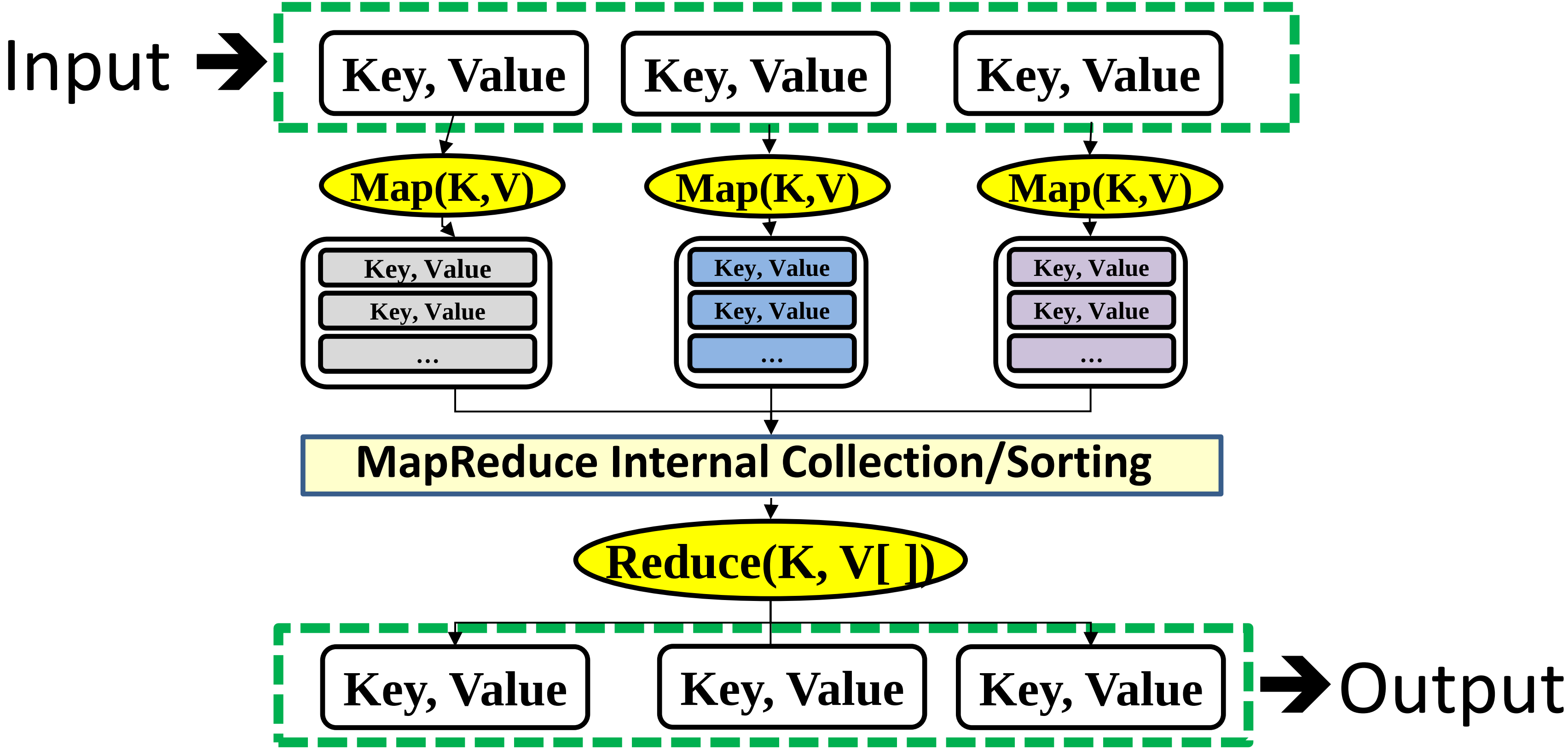
Figure 1: GFS Architecture

GHEMAWAT, S., GOBIOFF, H., AND LEUNG, S.-T. The google file system. In SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles (New York, NY, USA, 2003), ACM, pp. 29–43.
http://static.googleusercontent.com/media/research.google.com/en/us/archive/gfs-sosp2003.pdf

MapReduce: A Framework for Parallel Programming

- Minimize effort of programmer for simple parallel processing tasks
- Features
  - Hide many low-level details (network, storage)
  - Built-in fault tolerance
  - Automatic load balancing

MapReduce: Computation Pipeline



Slide adapted from Alexander Behm & Ajey Shah’s presentation (http://www.slideshare.net/gothicane/behm-shah-pagerank)

Slide adapted from ChengXiang Zhai's presentation

Word Counting

Input: Text Data

Hello World Bye World
Hello Hadoop Bye Hadoop
Bye Hadoop Hello Hadoop
… …

Output:
Count of each word

Bye 3
Hadoop 4
Hello 3
World 2
…

How can we do this within the MapReduce framework?

Slide adapted from Alexander Behm & Ajey Shah's presentation (http://www.slideshare.net/gothicane/behm-shah-pagerank)

Word Counting: Map Function

Input			Output
1, “Hello World Bye World”	Map(K,V)		<Hello,1> <World,1> <Bye,1> <World,1>
2, “Hello Hadoop Bye Hadoop”	Map(K,V)		<Hello,1> <Hadoop,1> <Bye,1> <Hadoop,1>

….

Map(K, V)
{ For each word w in V, Collect(w, 1);}

Slide adapted from Alexander Behm & Ajey Shah’s presentation (http://www.slideshare.net/gothicane/behm-shah-pagerank)

Word Counting: Reduce Function

Map Output

<Hello,1>
<World,1>
<Bye,1>
<World,1>

<Hello,1>
<Hadoop,1>
<Bye,1>
<Hadoop,1>
…

After internal grouping

<Bye → 1, 1, 1> → **Reduce(K, V[ ])** → <Bye, 3>

<Hadoop → 1, 1, 1, 1> → **Reduce(K, V[ ])** → <Hadoop, 4>

<Hello → 1, 1, 1> → **Reduce(K, V[ ])** → <Hello, 3>

Output

Reduce(K, V[ ])
{ Int count = 0; For each v in V, count += v; Collect(K, count); }

Slide adapted from Alexander Behm & Ajey Shah's presentation (http://www.slideshare.net/gothicane/behm-shah-pagerank)

Slide adapted from ChengXiang Zhai's presentation

Inverted Indexing with MapReduce

D1: java resource java class

Key	Value
java	(D1, 2)
resource	(D1, 1)
class	(D1,1)

D2: java travel resource

Key	Value
java	(D2, 1)
travel	(D2,1)
resource	(D2,1)

D3: …

Map

Built-In Shuffle and Sort: aggregate values by keys

Reduce

Key	Value
java	{(D1,2), (D2, 1)}
resource	{(D1, 1), (D2,1)}
class	{(D1,1)}
travel	{(D2,1)}
…	

Slide adapted from Jimmy Lin's presentation

Slide adapted from ChengXiang Zhai's presentation

Inverted Indexing: Pseudo-Code

class MAPPER
    procedure MAP(docid n, doc d)
        $H \leftarrow$ new ASSOCIATIVEARRAY
        for all term $t \in$ doc d **do**
            $H\{t\} \leftarrow H\{t\} + 1$
        for all term $t \in H$ **do**
            EMIT(term t, posting $\langle n, H\{t\}\rangle$)

class REDUCER
    procedure REDUCE(term t, postings $[\langle a_1, f_1\rangle, \langle a_2, f_2\rangle \ldots]$)
        $P \leftarrow$ new LIST
        for all posting $\langle a, f\rangle \in$ postings $[\langle a_1, f_1\rangle, \langle a_2, f_2\rangle \ldots]$ **do**
            APPEND($P, \langle a, f\rangle$)
        SORT(P)
        EMIT(term t, postings P)

Slide adapted from Jimmy Lin’s presentation

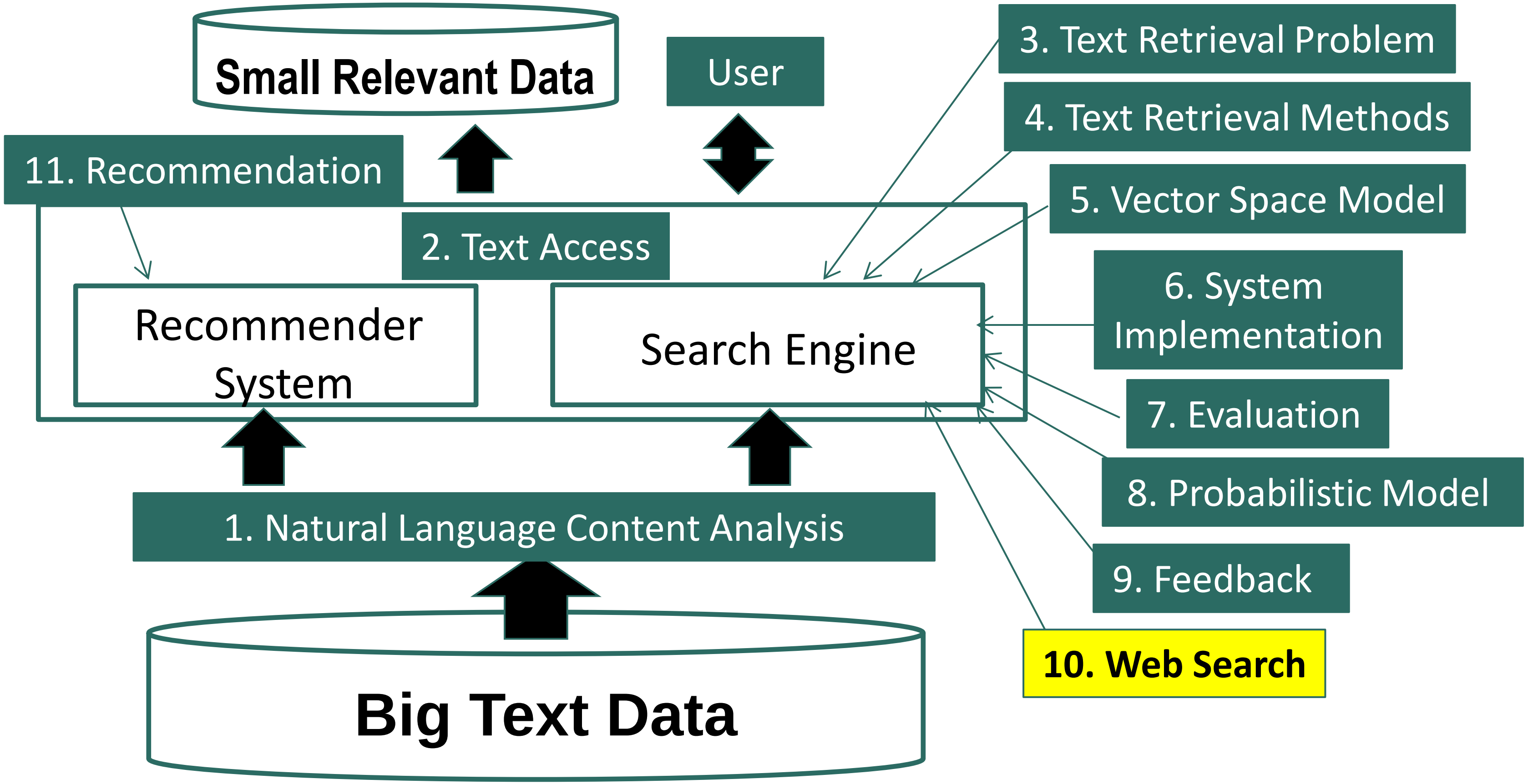
Slide adapted from ChengXiang Zhai's presentation

Summary

- Web scale indexing requires
  - Storing the index on multiple machines (GFS)
  - Creating the index in parallel (MapReduce)
- Both GFS and MapReduce are general infrastructures

Web Search: Link Analysis

Web Search: Link Analysis

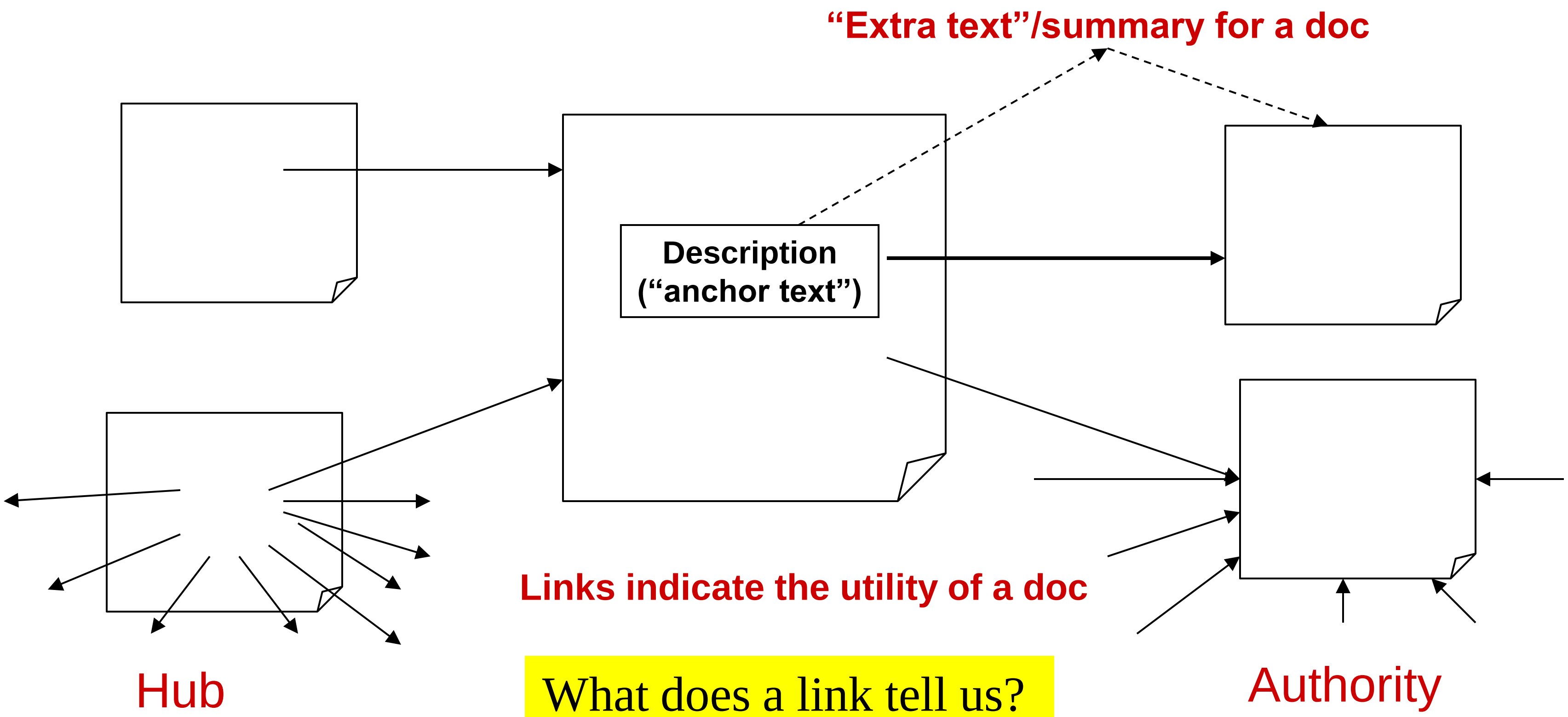


Slide adapted from ChengXiang Zhai's presentation

Ranking Algorithms for Web Search

- Standard IR models apply but aren’t sufficient
  - Different information needs
  - Documents have additional information
  - Information quality varies a lot
- Major extensions
  - Exploiting links to improve scoring
  - Exploiting clickthroughs for massive implicit feedback
  - In general, rely on machine learning to combine all kinds of features

Exploiting Inter-Document Links



What does a link tell us?

PageRank: Capturing Page “Popularity”

- Intuitions
  - Links are like citations in literature
  - A page that is cited often can be expected to be more useful in general
- PageRank is essentially “citation counting”, but improves over simple counting
  - Consider “indirect citations” (being cited by a highly cited paper counts a lot…)
  - Smoothing of citations (every page is assumed to have a non-zero pseudo citation count)
- PageRank can also be interpreted as random surfing (thus capturing popularity)

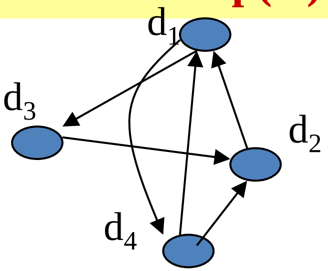
The PageRank Algorithm

Random surfing model: At any page,

With prob. α, randomly jumping to another page

With prob. $(1-\alpha)$, randomly picking a link to follow.

$p(d_i)$: PageRank score of d_i = average probability of visiting page d_i



Transition matrix

$$M = \begin{bmatrix} 0 & 0 & 1/2 & 1/2 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 \end{bmatrix}$$

M_{ij} = probability of going from d_i to d_j

$$\sum_{j=1}^{N} M_{ij} = 1$$

probability of visiting page dj at time t+1

probability of at page di at time t

"Equilibrium Equation":

$$p_{t+1}(d_j) = \underbrace{(1-\alpha)\sum_{i=1}^{N} M_{ij}\, p_t(d_i)}_{\text{Reach dj via following a link}} + \underbrace{\alpha \sum_{i=1}^{N} \tfrac{1}{N}\, p_t(d_i)}_{\text{Reach dj via random jumping}}$$

N= # pages

Reach dj via following a link

Reach dj via random jumping

dropping the time index

$$p(d_j) = \sum_{i=1}^{N} [\tfrac{1}{N}\alpha + (1-\alpha)M_{ij}]\, p(d_i)$$

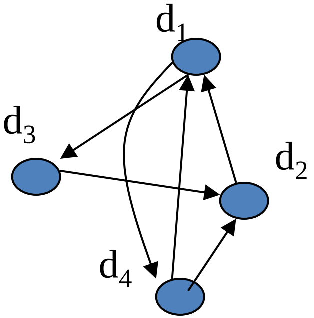
$$\vec{p} = (\alpha I + (1-\alpha)M)^T \vec{p}$$

I_{ij} = 1/N

We can solve the equation with an iterative algorithm

Slide adapted from ChengXiang Zhai's presentation

PageRank: Example



$$p(d_j) = \sum_{i=1}^{N} [\tfrac{1}{N}\alpha + (1-\alpha) M_{ij}]\, p(d_i)$$

$$\vec{p} = (\alpha I + (1-\alpha) M)^T \,\vec{p}$$

$$A = (1-0.2)M + 0.2I = 0.8 \times \begin{bmatrix} 0 & 0 & 1/2 & 1/2 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 \end{bmatrix} + 0.2 \times \begin{bmatrix} 1/4 & 1/4 & 1/4 & 1/4 \\ 1/4 & 1/4 & 1/4 & 1/4 \\ 1/4 & 1/4 & 1/4 & 1/4 \\ 1/4 & 1/4 & 1/4 & 1/4 \end{bmatrix}$$

$$\begin{bmatrix} p^{n+1}(d_1) \\ p^{n+1}(d_2) \\ p^{n+1}(d_3) \\ p^{n+1}(d_4) \end{bmatrix} = A^T \begin{bmatrix} p^{n}(d_1) \\ p^{n}(d_2) \\ p^{n}(d_3) \\ p^{n}(d_4) \end{bmatrix} = \begin{bmatrix} 0.05 & 0.85 & 0.05 & 0.45 \\ 0.05 & 0.05 & 0.85 & 0.45 \\ 0.45 & 0.05 & 0.05 & 0.05 \\ 0.45 & 0.05 & 0.05 & 0.05 \end{bmatrix} \times \begin{bmatrix} p^{n}(d_1) \\ p^{n}(d_2) \\ p^{n}(d_3) \\ p^{n}(d_4) \end{bmatrix}$$

$$p^{n+1}(d_1) = 0.05 * p^n(d_1) + 0.85 * p^n(d_2) + 0.05 * p^n(d_3) + 0.45 * p^n(d_4)$$

Initial value p(d)=1/N, iterate until converge

Do you see how scores are propagated over the graph?

Slide adapted from ChengXiang Zhai's presentation

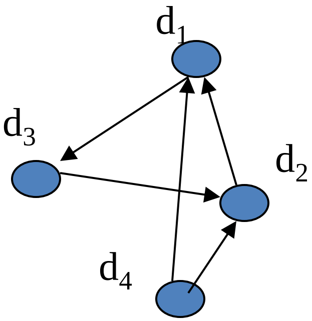
PageRank in Practice

- Computation can be quite efficient since M is usually sparse
- Normalization doesn't affect ranking, leading to some variants of the formula
- The zero-outlink problem: p(di)'s don't sum to 1
  - One possible solution = page-specific damping factor (α=1.0 for a page with no outlink)
- Many extensions (e.g., topic-specific PageRank)
- Many other applications (e.g., social network analysis)

HITS: Capturing Authorities & Hubs

- Intuitions
  - Pages that are widely cited are good authorities
  - Pages that cite many other pages are good hubs
- The key idea of HITS (Hypertext-Induced Topic Search)
  - Good authorities are cited by good hubs
  - Good hubs point to good authorities
  - Iterative reinforcement…
- Many applications in graph/network analysis

The HITS Algorithm



$$A = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

"Adjacency matrix"

Initial values: $a(d_i)=h(d_i)=1$

$$h(d_i) = \sum_{d_j \in OUT(d_i)} a(d_j)$$

$$a(d_i) = \sum_{d_j \in IN(d_i)} h(d_j)$$

Iterate

Normalize:

$$\sum_i a(d_i)^2 = \sum_i h(d_i)^2 = 1$$

$$\vec{h} = A\vec{a}; \qquad \vec{a} = A^T\vec{h}$$

$$\vec{h} = AA^T\vec{h}; \quad \vec{a} = A^TA\vec{a}$$

Summary

- Link information is very useful
  - Anchor text
  - PageRank
  - HITS
- Both PageRank and HITS have many applications in analyzing other graphs or networks