AVL Trees

Splay Trees

Red-Black Trees

B+ Trees

1 | Google

1 - eBizMBA Rank | **1,800,000,000** - Estimated Unique Monthly Visitors | 1 - Quantcast Rank | 1 - Alexa Rank | 1 - SimilarWeb Rank | *Last Updated:* February 1, 2020. The Best Search Engines | eBizMBA

2 | Bing

33 - eBizMBA Rank | **500,000,000** - Estimated Unique Monthly Visitors | 8 - Quantcast Rank | 40 - Alexa Rank | 43 - SimilarWeb Rank | *Last Updated:* February 1, 2020. The Best Search Engines | eBizMBA

3 | Yahoo! Search

43 - eBizMBA Rank | **490,000,000** - Estimated Unique Monthly Visitors | 8 - Quantcast Rank | *56* - Alexa Rank | *67* - SimilarWeb Rank | *Last Updated:* February 1, 2020. The Best Search Engines | eBizMBA

4 | Baidu

54 - eBizMBA Rank | **480,000,000** - Estimated Unique Monthly Visitors | *150* - Quantcast Rank | 4 - Alexa Rank | 9 - SimilarWeb Rank | *Last Updated:* February 1, 2020. The Best Search Engines | eBizMBA

5 | Ask

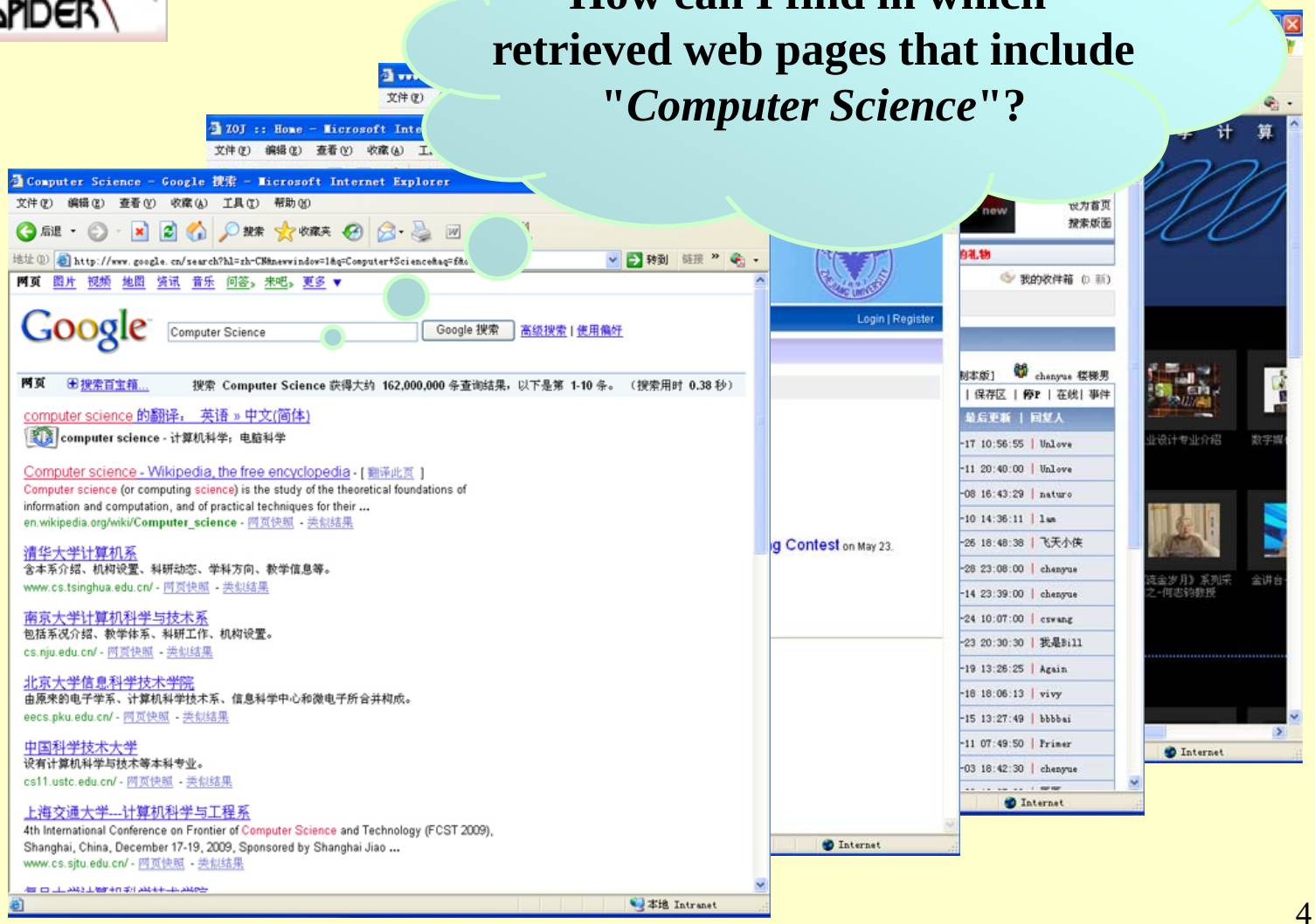
205 - eBizMBA Rank | **300,000,000** - Estimated Unique Monthly Visitors | 329 - Quantcast Rank | 110 - Alexa Rank | 177 - SimilarWeb Rank | *Last Updated:* February 1, 2020. The Best Search Engines | eBizMBA

6 | Aol Search

273 - eBizMBA Rank | **200,000,000** - Estimated Unique Monthly Visitors | *350* - Quantcast Rank | 276 - Alexa Rank | *194* - SimilarWeb Rank | *Last Updated:* February 1, 2020. The Best Search Engines | eBizMBA

Inverted File Index

How can I find in which retrieved web pages that include "*Computer Science*"?



☞ **Solution 1:** **Scan each page for the string "Computer Science".**

Wait till your next life !

☞ Solution 2: Term-Document Incidence Matrix

〖Example〗 Document sets

Doc	Text
1	Gold silver truck
2	Shipment of gold damaged in a fire
3	Delivery of silver arrived in a silver truck
4	Shipment of gold arrived in a truck

	1	2	3	4
a	0	1	1	1
arrived	0	0	1	1
damaged	0	1	0	0
delivery	0	0	1	0
fire	0	1	0	0
gold	1	1	0	1
of	0	1	1	1
in	0	1	1	1
shipment	0	1	0	1
silver	1	0	1	0
truck	1	0	1	1

silver & truck

☞ Solution 3: Compact Version - Inverted File Index

【Definition】 Index is a mechanism for locating a given term in a text.

【Definition】 Inverted file contains a list of pointers (e.g. the number of a page) to all occurrences of that term in the text.

Doc	Text
1	Gold silver truck
2	Shipment of gold damaged in a fire
3	Delivery of silver arrived in a silver truck
4	Shipment of gold arrived in a truck

Inverted File Index →

No.	Term	Times; Documents
1	a	<3; 2,3,4>
2	arrived	<2; 3,4>
3	damaged	<1; 2>
4	delivery	<1; 3>
5	fire	<1; 2>
6	gold	<3; 1,2,4>
7	of	<3; 2,3,4>
8	in	<3; 2,3,4>
9	shipment	<2; 2,4>
10	silver	<2; 1,3>
11	truck	<3; 1,3,4>

Doc	Text
1	Gold silver truck
2	Shipment of gold damaged in a fire
3	Delivery of silver arrived in a silver truck
4	Shipment of gold arrived in a truck

No.	Term	Times; Documents Words
1	a	<3; (2;6),(3;6),(4;6)>
2	arrived	<2; (3;4),(4;4)>
3	damaged	<1; (2;4)>
4	delivery	<1; (3;1)>
5	fire	<1; (2;7)>
6	gold	<3; (1;1),(2;3),(4;3)>
7	of	<3; (2;2),(3;2),(4;2)>
8	in	<3; (2;5),(3;5),(4;5)>
9	shipment	<2; (2;1),(4;1)>
10	silver	<2; (1;2),(3;3,7)>
11	truck	<3; (1;3),(3;8),(4;7)>

Term Dictionary Posting List

How to easily print the sentences which contain the words and highlight the words?

Why do we keep “times” (frequency)?

Index Generator

```
while ( read a document D ) {
    while ( read a term T in D ) {
        if ( Find( Dictionary, T ) == false )
            Insert( Dictionary, T );
        Get T's posting list;
        Insert a node to T's posting list;
    }
}
Write the inverted index to disk;
```

Token Analyzer 中文分词
Stop Filter

Vocabulary Scanner 词义查询

Vocabulary Insertor

Memory management

While reading a term ……

✂ ***Word Stemming***

Process a word so that only its stem or root form is left.

〖Example〗

Process **processing** **processes** **processed**	**process**	**says** **said** **saying**	**say**

✂ ***Stop Words***

Some words are so common that almost every document contains them, such as "a" "the" "it". It is useless to index them. They are called *stop words*. We can eliminate them from the original documents.

While accessing a term ……

- **Solution 1: Search trees (*B- trees, B+ trees, Tries, ...*)**

- **Solution 2: Hashing**

Discussion 3:
What are the pros and cons of using hashing, comparing to using search trees?

- 👍 **faster for one word**
- 👎 **scanning in sequential order is not possible (e.g. range searches are expensive)**

While not having enough memory ……

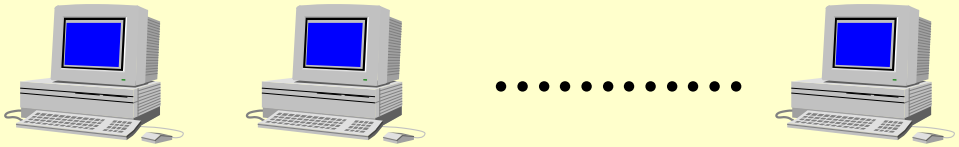
```
BlockCnt = 0;
while ( read a document D ) {
  while ( read a term T in D ) {
    if ( out of memory ) {
      Write BlockIndex[BlockCnt] to disk;
      BlockCnt ++;
      FreeMemory;
    }
    if ( Find( Dictionary, T ) == false )
      Insert( Dictionary, T );
    Get T's posting list;
    Insert a node to T's posting list;
  }
}
for ( i=0; i<BlockCnt; i++ )
  Merge( InvertedIndex, BlockIndex[i] );
```

Sorted

Distributed indexing (for web-scale indexing — don't try this at home!)

—— Each node contains index of a subset of collection

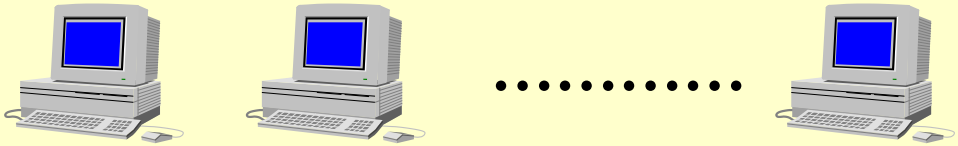
☞ Solution 1: Term-partitioned index



A~C　　D~F　　X~Z

困难：不同字母频数不一，如何切分

☞ Solution 2: Document-partitioned index

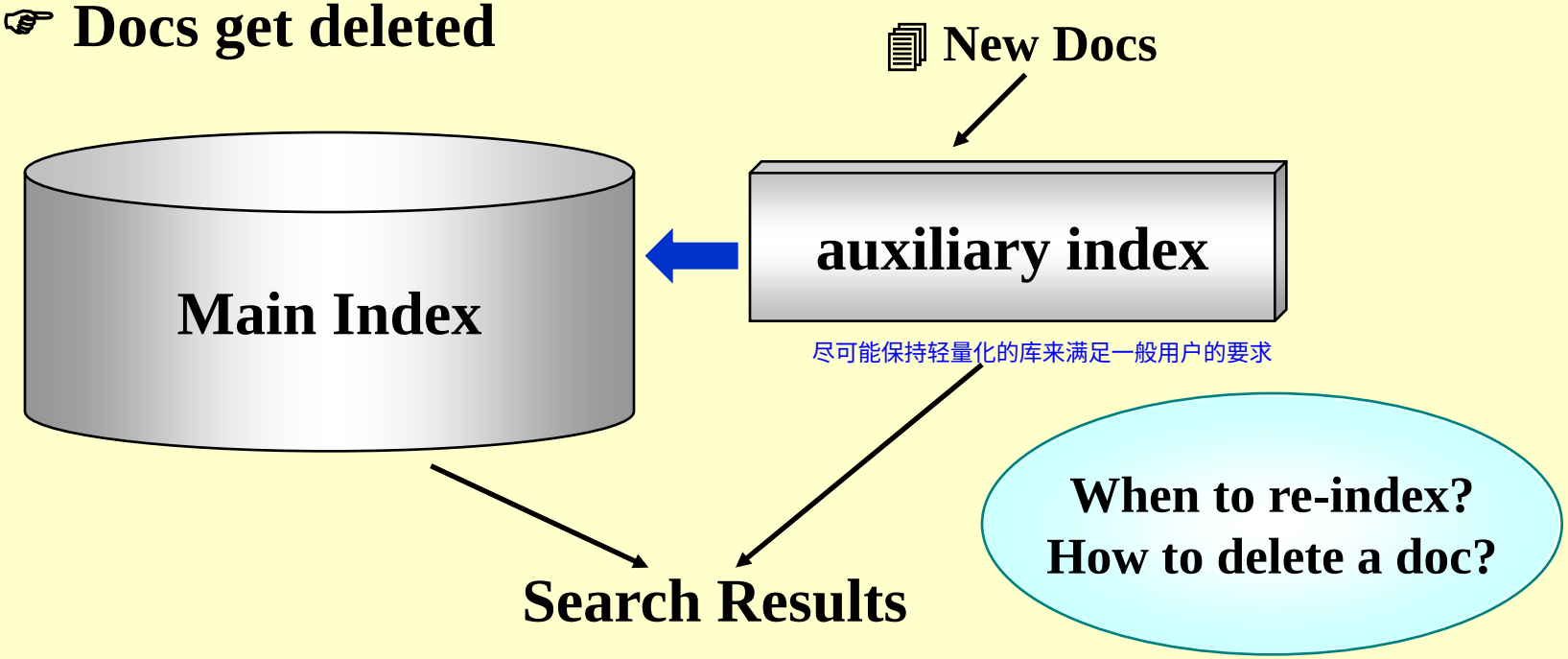


1~ 10000　　10001~ 20000　　90001~ 100000

需要Merge

Dynamic indexing

- ☞ Docs come in over time
  - postings updates for terms already in dictionary
  - new terms added to dictionary
- ☞ Docs get deleted



Compression

Term
~~a~~
arrive
damage
deliver
fire
gold
~~of~~
~~in~~
shipment
silver
truck

arrivedamagedeliverfiregoldshipmentsilvertruck

Posting List:

computer → 2, 15, 47, …, 58879, 58890, …

2, 13, 32, … …, 11, …

☝ Most gaps can be encoded with far fewer than 20 bits

Thresholding 阈值 != 阀值

☞ **Document:** only retrieve the top *x* documents where the documents are ranked by weight

- 👎 Not feasible for Boolean queries
- 👎 Can miss some relevant documents due to truncation

☞ **Query:** Sort the query terms by their frequency in ascending order; search according to only some percentage of the original query terms

T1	T2	T3	T4	T5	T6	T7	T8	T9	T10

20% 40% 80%

Measures for a search engine

- ☞ **How fast does it index**
  - **- Number of documents/hour**
- ☞ **How fast does it search**
  - **- Latency as a function of index size**
- ☞ **Expressiveness of query language**
  - **- Ability to express complex information needs**
  - **- Speed on complex queries**

☞ **User happiness ?**

- Data Retrieval Performance Evaluation (after establishing correctness)

> **Response time**

> **Index space**

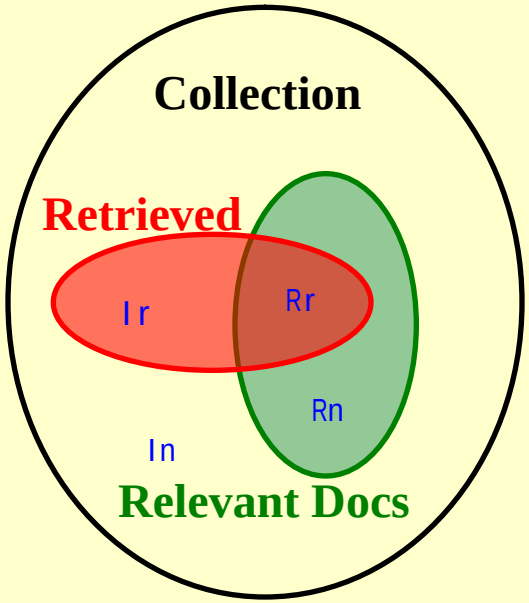
- Information Retrieval Performance Evaluation

> **+ How *relevant* is the answer set?**

Relevance **measurement requires 3 elements:**

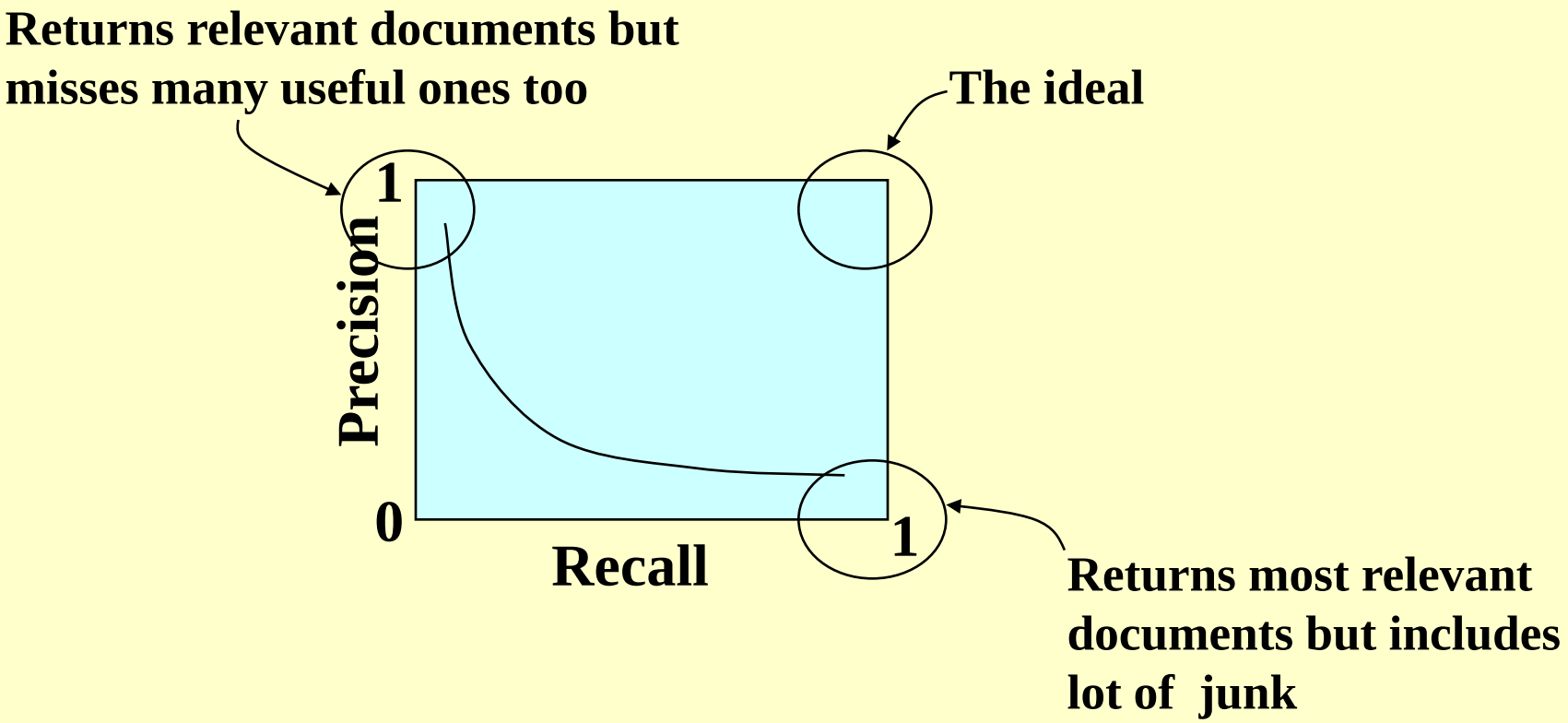
1. **A benchmark document collection**
2. **A benchmark suite of queries**
3. **A binary assessment of either <u>Relevant</u> or <u>Irrelevant</u> for each query-doc pair**

	Relevant	Irrelevant
Retrieved	R_R	I_R
Not Retrieved	R_N	I_N



Precision 准确率 $P = R_R / (R_R + I_R)$

Recall 召回率 $R = R_R / (R_R + R_N)$


Returns relevant documents but misses many useful ones too
The ideal
1
Precision
0
1
Recall
Returns most relevant documents but includes lot of junk

Discussion 4:

How to improve the *relevancy* of search results?

- **Page Rank**
- **Semantic Web**
- **……**

Reference:

Download "InvertedFileIndex.zip".

- **The Google File System.pdf**
- **Building an Inverted Index.pdf**
- **Inverted Index Construction(ppt).pdf**
- **Compression.pdf**