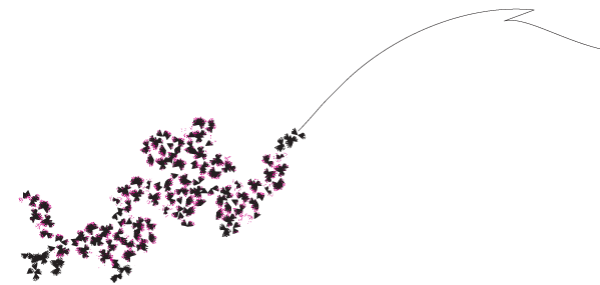


How to build Google in an afternoon *(or any other large web search engine)*

Djoerd Hiemstra

<http://www.cs.utwente.nl/~hiemstra>



Ingredients of this talk:

1. A bit of high school mathematics
 2. Zipf's law
 3. Indexing, query processing
- Shake well...



Course objectives

- Understand the scale of “things”
- Estimate index size and query time
- Index compression
- Top-k optimizations

New web scale search engine

- How much money do we need for our startup?



Dear bank,

- We budget one desktop PC
- We put the entire web index on a desktop PC and search it in reasonable time:
 - a) probably
 - b) maybe
 - c) no
 - d) no, are you crazy?



“Google” Circa 1997 (google.stanford.edu)





Search the web using Google

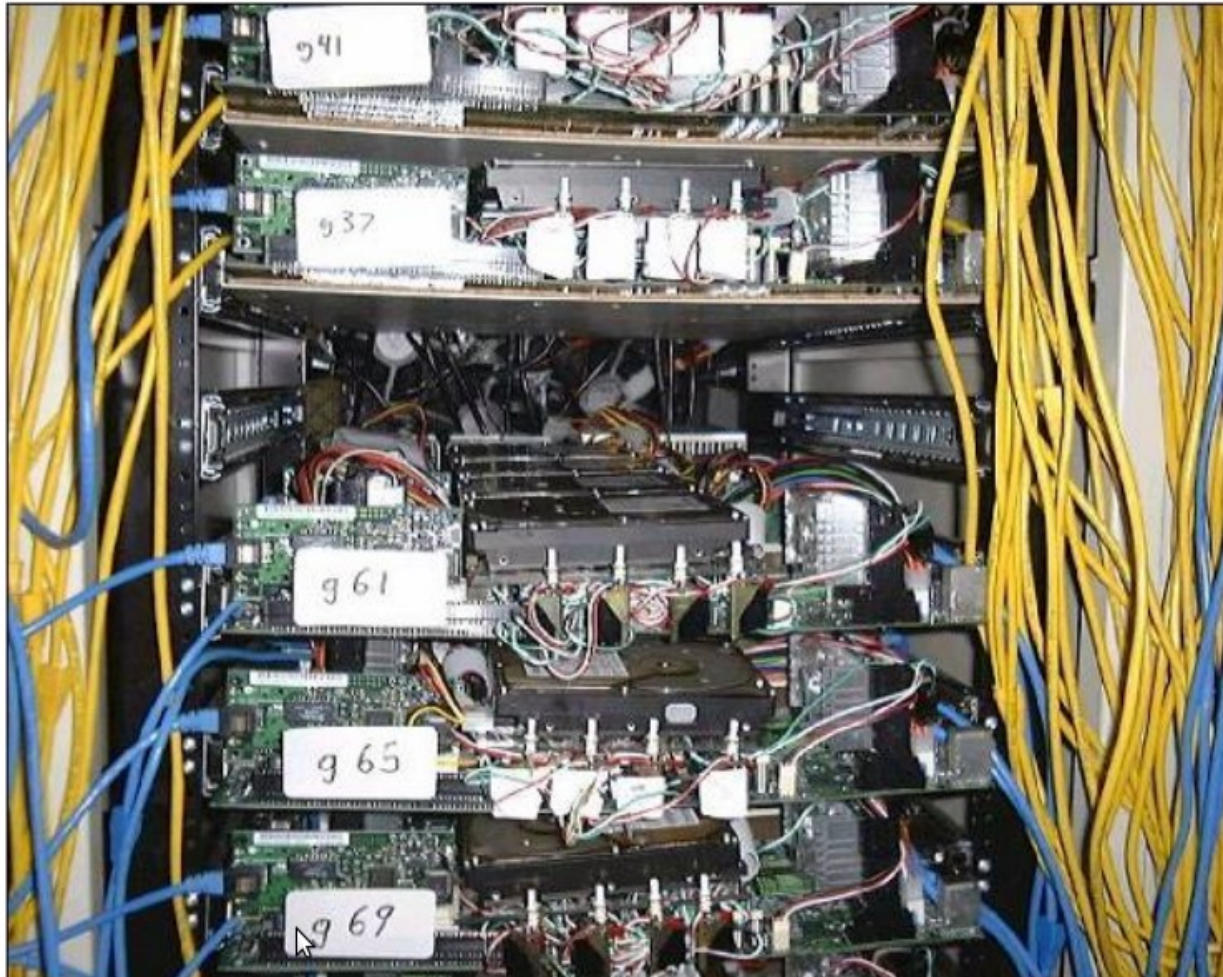
Google Search

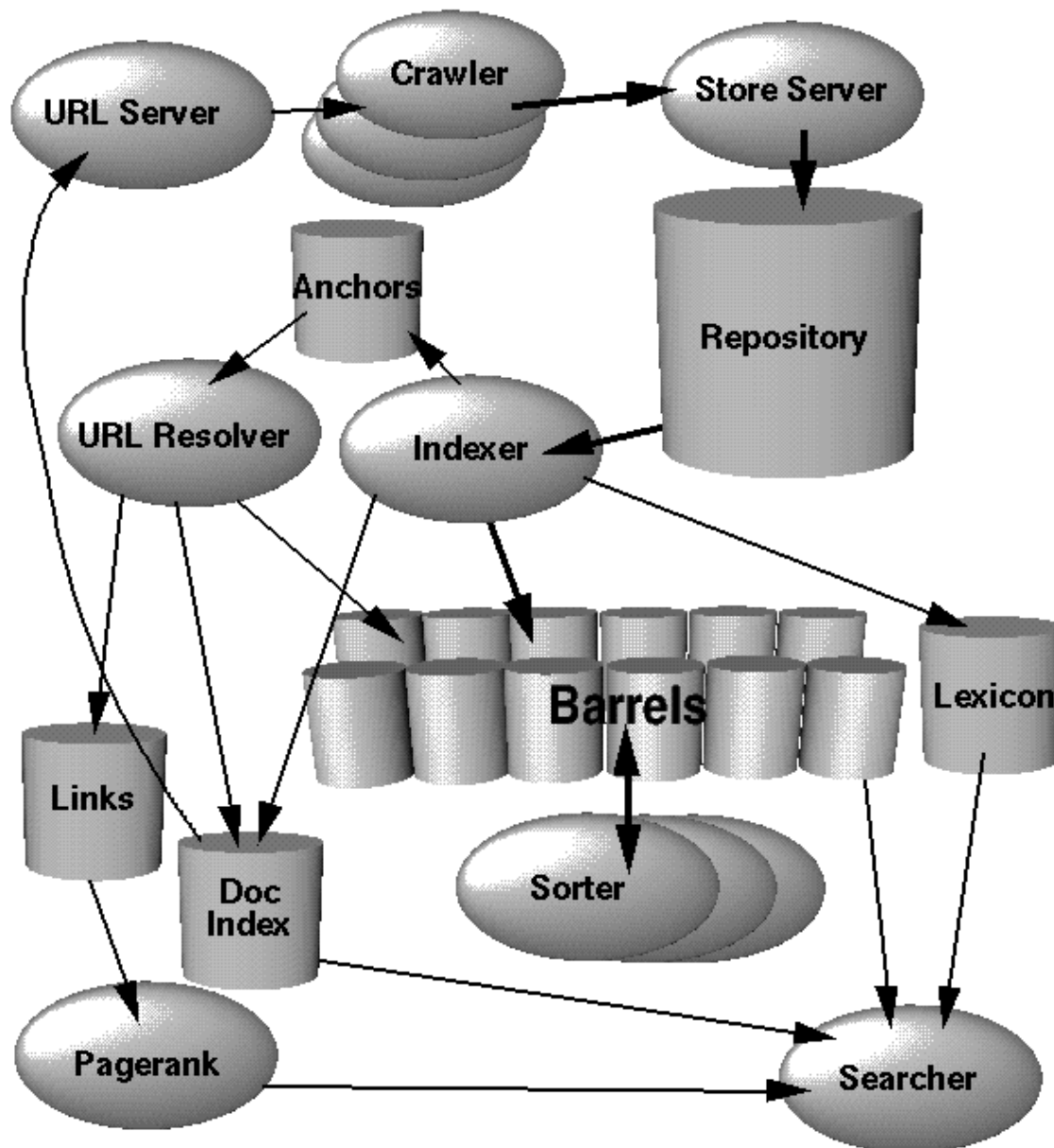
I'm feeling lucky

[More Google!](#)

Copyright ©1999 Google Inc.

“Corkboards” (1999)





(Brin & Page 1998)

Google '98: Forward & Inverted Index

Hit: 2 bytes

plain:	cap:1	imp:3	position: 12		
fancy:	cap:1	imp = 7	type: 4	position: 8	
anchor:	cap:1	imp = 7	type: 4	hash:4	pos: 4

Forward Barrels: total 43 GB

docid	wordid: 24	nhits: 8	hit hit hit hit
	wordid: 24	nhits: 8	hit hit hit hit
	null wordid		
docid	wordid: 24	nhits: 8	hit hit hit hit
	wordid: 24	nhits: 8	hit hit hit hit
	wordid: 24	nhits: 8	hit hit hit hit
	null wordid		

...

Lexicon: 293MB

Inverted Barrels: 41 GB

wordid	ndocs	—	→	docid: 27	nhits:5	hit hit hit hit
wordid	ndocs	—	→	docid: 27	nhits:5	hit hit hit
wordid	ndocs	—	→	docid: 27	nhits:5	hit hit hit hit
			→	docid: 27	nhits:5	hit hit
...						

Google'98: Storage numbers

Total Size of Fetched Pages	147.8 GB
Compressed Repository	53.5 GB
Short Inverted Index	4.1 GB
Full Inverted Index	37.2 GB
Lexicon	293 MB
Temporary Anchor Data (not in total)	6.6 GB
Document Index Incl. Variable Width Data	9.7 GB
Links Database	3.9 GB
Total Without Repository	55.2 GB
Total With Repository	108.7 GB

Google'98: Page search

Web Page Statistics	
Number of Web Pages Fetched	24 million
Number of URLs Seen	76.5 million
Number of Email Addresses	1.7 million
Number of 404's	1.6 million

Google'98: Search speed

	Initial Query		Same Query Repeated (IO mostly cached)	
Query	CPUTime(s)	Total Time(s)	CPU Time(s)	Total Time(s)
al gore	0.09	2.13	0.06	0.06
vice president	1.77	3.84	1.66	1.80
hard disks	0.25	4.86	0.20	0.24
search engines	1.31	9.63	1.16	1.16

Google's 20th birthday



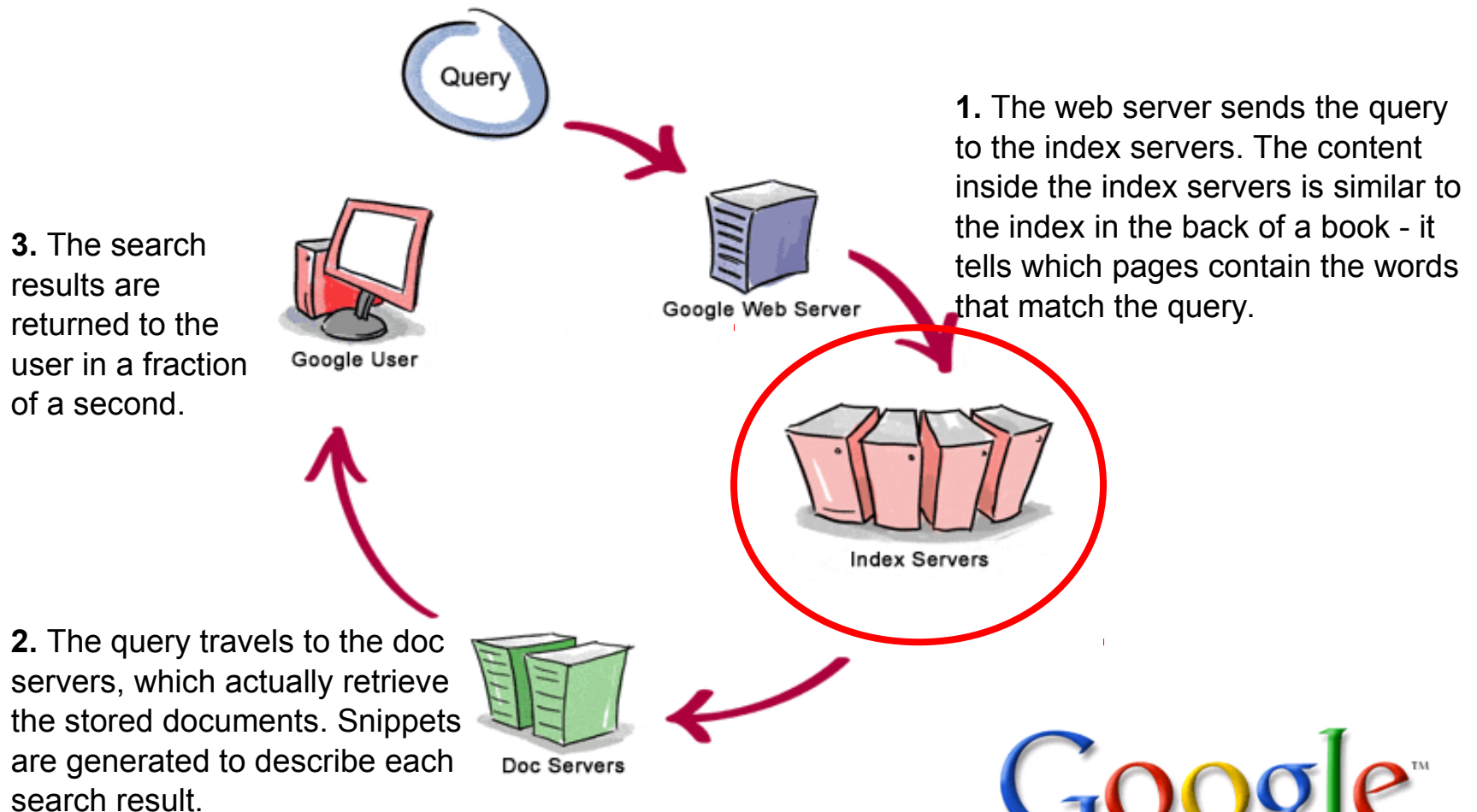
Larger-Scale Computing



Google's 20th birthday

- World's largest cluster of commodity hardware (over 1,000,000 servers)
- Over XX(?) billion web pages are indexed
 - 1 trillion pages reportedly found, but not everything in index

Architecture



More info:

Jeff Dean's WSDM 2009 keynote:

Challenges in Building Large-Scale Information Retrieval Systems

<http://research.google.com/people/jeff/WSDM09-keynote.pdf>

http://videolectures.net/wsdm09_dean_cblirs/



Q1: How many bytes is 10 billion pages?

- Only the text

?

How many bytes?

- About 10 billion pages
- Assume a page contains 500 terms on average (ClueWeb09: about 900)
- Each term consists of 5 characters on average
- To store the web you need:
 - $10^{10} \times 500 \times 6 \sim 30 \text{ TB}$

Ian H. Witten, Alistair Moffat, Timothy C. Bell. Managing Gigabytes: Compressing and Indexing Documents and Images, 2nd edition, Morgan Kaufmann, 1999

► **Table 4.2** Collection statistics for Reuters-RCV1. Values are rounded for the computations in this book. The unrounded values are: 806,791 documents, 222 tokens per document, 391,523 (distinct) terms, 6.04 bytes per token with spaces and punctuation, 4.5 bytes per token without spaces and punctuation, 7.5 bytes per term, and 96,969,056 tokens. The numbers in this table correspond to the third line (“case folding”) in Table 5.1 (page 87).

Symbol	Statistic	Value
N	documents	800,000
L_{ave}	avg. # tokens per document	200
M	terms	400,000
	avg. # bytes per token (incl. spaces/punct.)	6
	avg. # bytes per token (without spaces/punct.)	4.5
	avg. # bytes per term	7.5
T	tokens	100,000,000

What did we ignore?

- Text statistics:
 - Term frequency
 - Collection frequency
 - Inverse document frequency ...
- Hypertext statistics:
 - Ingoing and outgoing links
 - Anchor text
 - Term positions, proximities, sizes, and characteristics ...

Q2: How fast can we scan 30 TB?

- How would you estimate this?

?

How fast can we search 30 TB?

- We need to find a very **large** hard disk
 - Size: 30 TB ??
 - Hard disk transfer time 100 MB/s
- Time needed to sequentially scan the data:
 - 300,000 seconds ...
 - ... so, we have to wait for 3.5 days to get the answer to one (1) query
- We can definitely do better than that!

From the book

► **Table 4.1** Typical system parameters in 2007. The seek time is the time needed to position the disk head in a new position. The transfer time per byte is the rate of transfer from disk to memory when the head is in the right position.

Symbol	Statistic	Value
s	average seek time	$5 \text{ ms} = 5 \times 10^{-3} \text{ s}$
b	transfer time per byte	$0.02 \mu\text{s} = 2 \times 10^{-8} \text{ s}$
	processor's clock rate	10^9 s^{-1}
p	lowlevel operation (e.g., compare & swap a word)	$0.01 \mu\text{s} = 10^{-8} \text{ s}$
	size of main memory	several GB
	size of disk space	1 TB or more

Issues that we do not address

- Web crawling
 - politeness, freshness, duplicates, missing links, loops, server problems, virtual hosts, etc.
- Maintain large cluster of servers
 - Page servers: store and deliver the results of the queries
 - Index servers: resolve the queries
- Answer 100 million of user queries per day
 - Caching, replicating, parallel processing, etc.
 - **Indexing, compression, coding**, fast access, etc.

Ingredients of this talk:

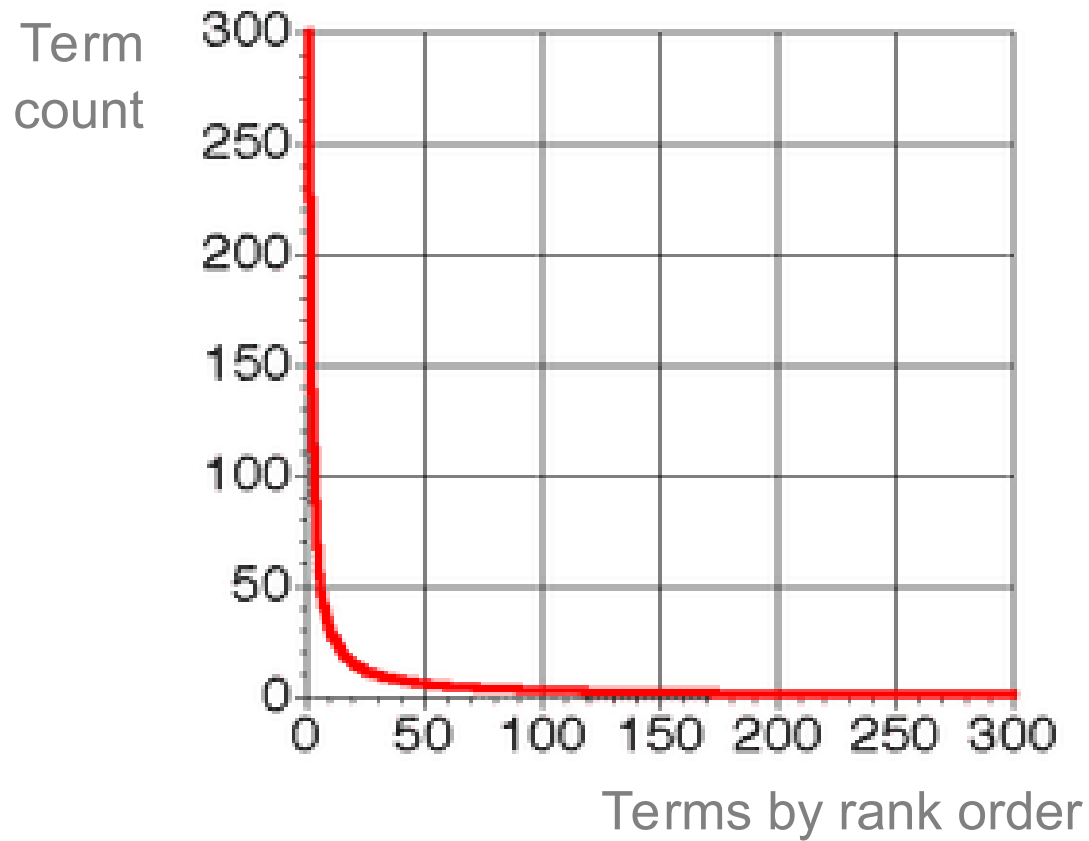
1. A bit of high school mathematics
 2. Zipf's law
 3. Indexing, query processing
- Shake well...

George Zipf's law



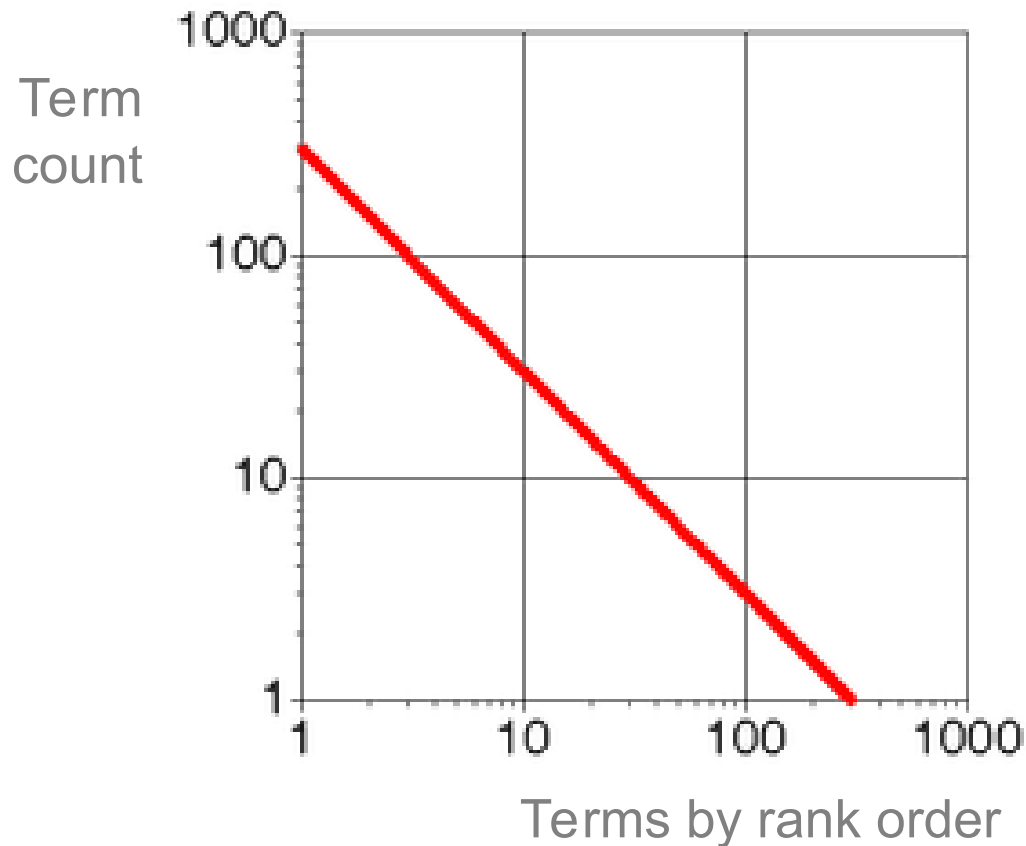
- Count how many times a term occurs in the collection
 - call this f
- Order them in descending order
 - call the rank r
- Zipf's claim:
 - For each word, the product of frequency and rank is approximately constant: $f \times r = c$

Zipf distribution



Linear scale

Zipf distribution



Logarithmic scale

Consequences

Few terms occur very frequently: a, an, the, ... => non-informative (stop) words

- Many terms occur very infrequently: spelling mistakes, foreign names, ...
- Medium number of terms occur with medium frequency

Word resolving power

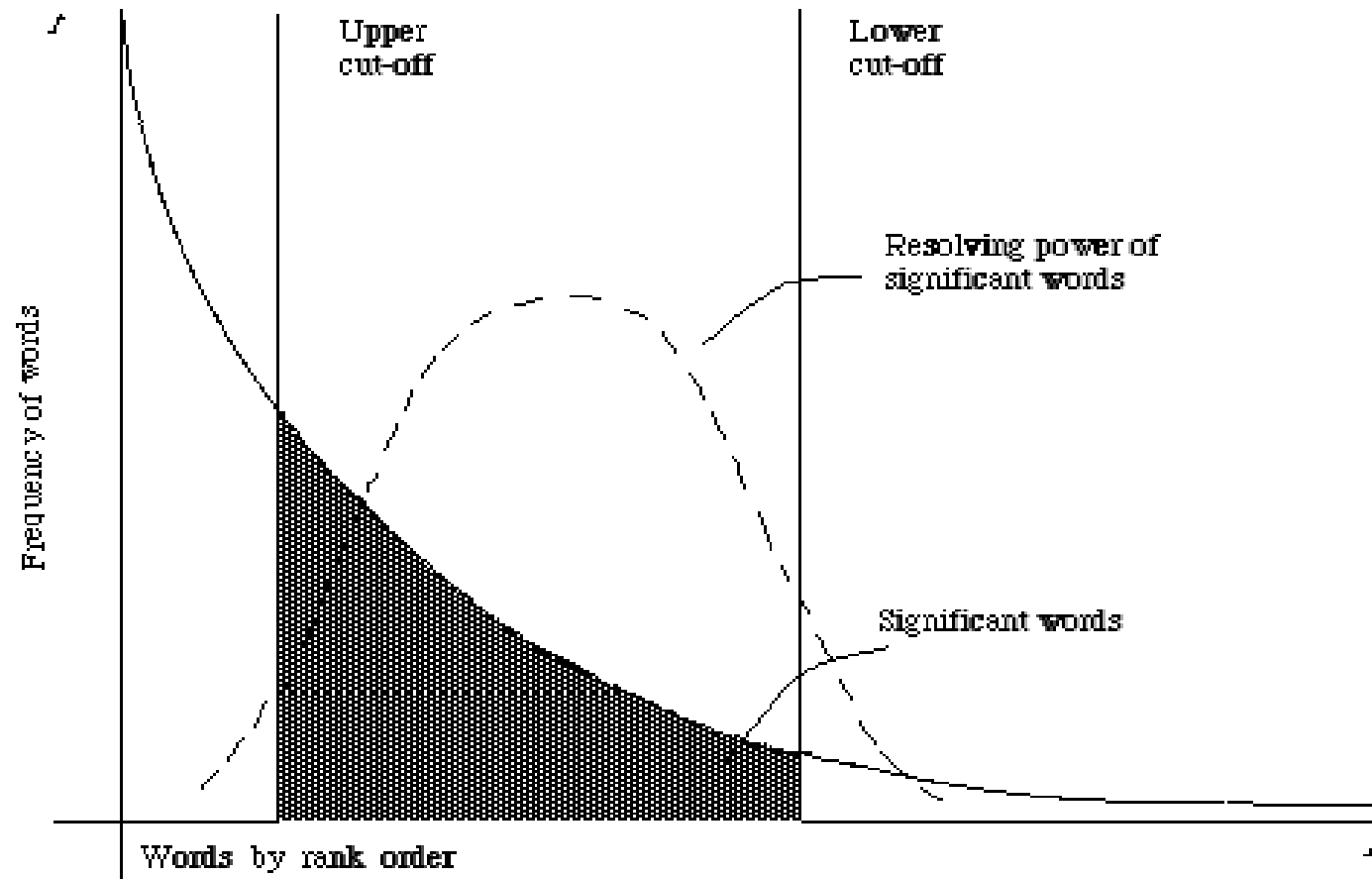
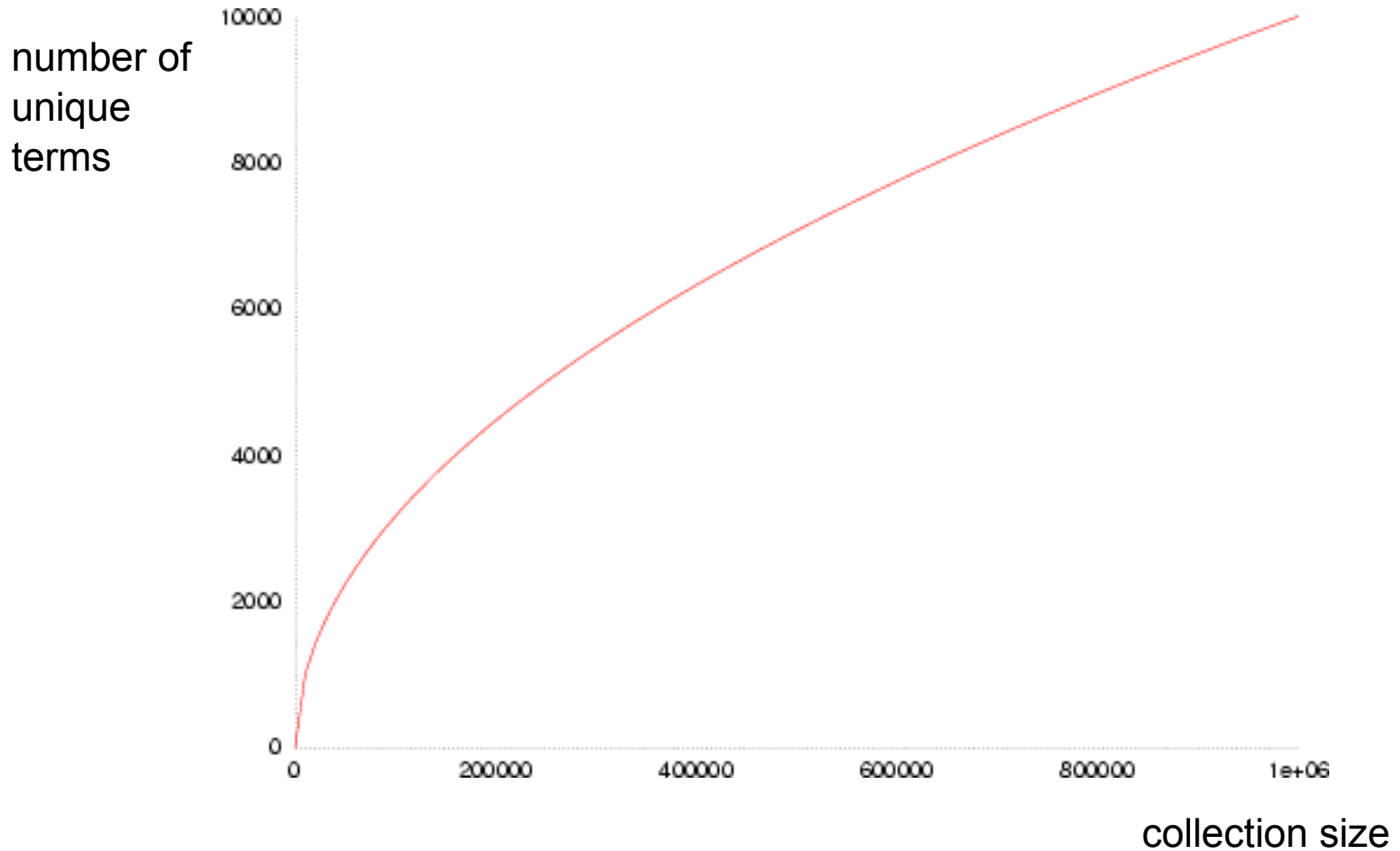


Figure 2.1. A plot of the hyperbolic curve relating f , the frequency of occurrence and r , the rank order (Adapted from Schultz⁴⁴ page 120)

Heap's law for dictionary size



Ingredients of this talk:

1. A bit of high school mathematics
 2. Zipf's law
 3. Indexing
- Shake well...

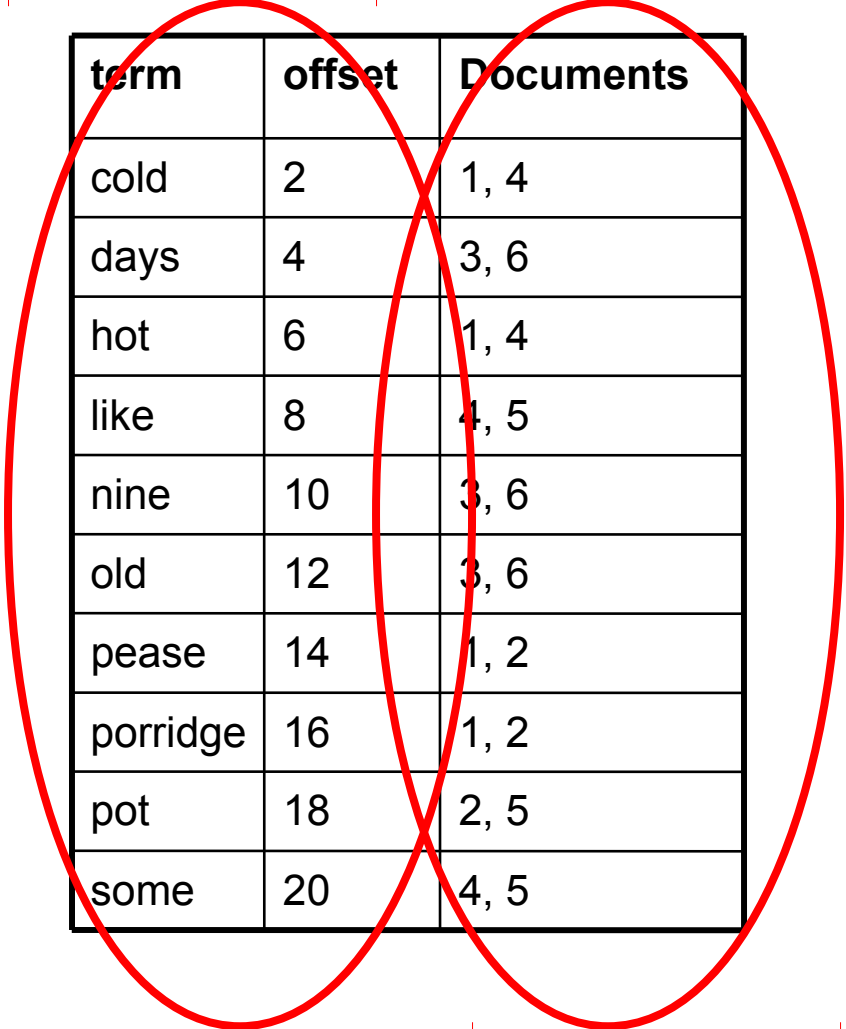
Example

Document number	Text
1	Pease porridge hot, pease porridge cold
2	Pease porridge in the pot
3	Nine days old
4	Some like it hot, some like it cold
5	Some like it in the pot
6	Nine days old

Stop words: in, the, it.

(Witten, Moffat & Bell, 1999)

Inverted index



term	offset	Documents
cold	2	1, 4
days	4	3, 6
hot	6	1, 4
like	8	4, 5
nine	10	3, 6
old	12	3, 6
pease	14	1, 2
porridge	16	1, 2
pot	18	2, 5
some	20	4, 5

dictionary

postings

Q3: Estimate the size of the
inverted index

?

Size of the inverted index

- Number of postings (term-document pairs):
 - Number of documents: $\sim 10^{10}$,
 - Average number of unique terms per document (document size ~ 500): ~ 250
 - 5 bytes for each posting (why?)
 - So, $10^{10} \times 250 \times 5 = 12.5$ TB
 - postings take about half the size of the data

Size of the inverted index

- Number of unique terms is, say, 10^8
 - 6 bytes on average
 - plus off-set in postings, another 8 bytes
 -
 - So, $10^8 \times 14 = 1.4$ GB
 - So, dictionary is tiny compared to postings (0.01 %)

Inverted index encoding

- The inverted file entries are usually stored in order of increasing document number

– [*<retrieval*; 7; [2, 23, 81, 98, 121, 126, 180]]>

(the term “retrieval” occurs in 7 documents with document identifiers 2, 23, 81, 98, etc.)

Query processing (1)

- Each inverted file entry is an ascending ordered sequence of integers
 - allows merging (joining) of two lists in a time linear in the size of the lists

Query processing (2)

- Usually queries are assumed to be *conjunctive* queries
 - query: *information retrieval*
 - is processed as *information AND retrieval*
- [<*retrieval*; 7; [2, 23, 81, 98, 121, 126, 139]>
[<*information*; 9; [1, 14, 23, 45, 46, 84, 98, 111, 120]>
- intersection of posting lists gives:
[23, 98]

Query processing (3)

- Remember the Boolean model?
 - intersection, union and complement is done on posting lists
 - so, *information OR retrieval*

[<*retrieval*; 7; [2, 23, 81, 98, 121, 126, 139]>

[<*information*; 9; [1, 14, 23, 45, 46, 84, 98, 111, 120]>

- union of posting lists gives:

[1, 2, 14, 23, 45, 46, 81, 84, 98, 111, 120, 121, 126, 139]

Q4: Estimate the time needed for the query “information retrieval” using the inverted file

- Assume the selectivity of terms:
 - Suppose *information* occurs on 1 billion pages
 - Suppose *retrieval* occurs on 10 million pages

Is this a reasonable estimate?



Query processing (4)

- Estimate of selectivity of terms:
 - Suppose *information* occurs on 1 billion pages
 - Suppose *retrieval* occurs on 10 million pages
- size of postings (5 bytes per docid):
 - 1 billion * 5B = 5 GB for *information*
 - 10 million * 5B = 50 MB for *retrieval*
- Hard disk transfer time:
 - 50 sec. for *information* + 0.5 sec. for *retrieval*
 - (ignore CPU time and disk latency)

Query processing (5)

- We just brought query processing down from 3 days to just 50.5 seconds (!)
:-)
- Still... way too slow...
:-(

Inverted file compression (1)

- Trick 1, store sequence of doc-ids:
 - [*<retrieval*; 7; [2, 23, 81, 98, 121, 126, 180]]>

as a sequence of gaps

- [*<retrieval*; 7; [2, 21, 58, 17, 23, 5, 54]]>
- No information is lost.
- Always process posting lists from the beginning, so easily decoded into the original sequence

Inverted file compression (2)

- Does it help?
 - maximum gap determined by the number of indexed web pages...
 - infrequent terms coded as a few large gaps
 - frequent terms coded by many small gaps
- Trick 2: use variable byte length encoding.

Variable byte encoding (1)

► **Table 5.5** Some examples of unary and γ codes. Unary codes are only shown for the smaller numbers. Commas in γ codes are for readability only and are not part of the actual codes.

number	unary code	length	offset	γ code
0	0			
1	10	0		0
2	110	10	0	10,0
3	1110	10	1	10,1
4	11110	110	00	110,00
9	1111111110	1110	001	1110,001
13		1110	101	1110,101
24		11110	1000	11110,1000
511		1111111110	11111111	111111110,11111111
1025		11111111110	0000000001	11111111110,0000000001

(Manning et al. 1999)

Q5: Give γ code for $x=5$

- γ code: represent number x as:
 - first bits as the unary code for $1 + \lceil \log_2 x \rceil$
 - remainder bits as binary code for $x - 2^{\lceil \log_2 x \rceil}$
 - unary part (minus 1) specifies how many bits are required to code the remainder part

Variable byte encoding (2)

- γ code: represent number x as:
 - first bits as the unary code for $1 + \lceil {}^2 \log x \rceil$
 - remainder bits as binary code for $x - 2^{\lceil {}^2 \log x \rceil}$
 - unary part (minus 1) specifies how many bits are required to code the remainder part
- For example $x = 5$:
 - first bits: 110 $\left(1 + \lceil {}^2 \log 5 \rceil = 1 + \lceil 2.32 \rceil = 3 \right)$
 - remainder: 01 $\left(5 - 2^{\lceil {}^2 \log 5 \rceil} = 5 - 2^2 = 1 \right)$

Index sizes

Method	Bits per pointer			
	<i>Bible</i>	<i>GNUbib</i>	<i>Comact</i>	<i>TREC</i>
<i>Global methods</i>				
Unary	264	920	490	1719
Binary	15.00	16.00	18.00	20.00
Bernoulli	9.67	11.65	10.58	12.61
γ	6.55	5.69	4.48	6.43
δ	6.26	5.08	4.36	6.19
Observed frequency	5.92	4.83	4.21	5.83
<i>Local methods</i>				
Bernoulli	6.13	6.17	5.40	5.73
Hyperbolic	5.77	5.17	4.65	5.74
Skewed Bernoulli	5.68	4.71	4.24	5.28
Batched frequency	5.61	4.65	4.03	5.27

Table 3.7 Compression of inverted files, in bits per pointer.

Q6: Estimate the compressed
index size

?

Index size of our search engine

- Number of postings (term-document pairs):
 - 10 billion documents
 - 250 unique terms on average
 - Assume on average 6 bits per doc-id
 - $10^{10} \times 250 \times 6 \text{ bits} \approx 1.9 \text{ TB}$
 - about 15% of the uncompressed inverted file.
- It nicely fits one big hard drive :-)

Q7: Estimate the time needed for the query “information retrieval” using the compressed inverted file

- Assume the selectivity of terms:
 - Suppose *information* occurs on 1 billion pages
 - Suppose *retrieval* occurs on 10 million pages



Query processing on compressed index

- size of postings (6 bits per docid):
 - 1 billion * 6 bits = 750 Mb for "*information*"
 - 10 million * 6 bits = 7.5 Mb for "*retrieval*"
- Hard disk transfer time:
 - 7.5 sec. for information + 0.08 sec. for retrieval
 - (ignore CPU time and disk latency)

Query processing – Continued (1)

- We already brought down query processing from more than 1 day to 50.5 seconds...
- and brought that down to 7.58 seconds
:-)
- but that is still too slow...
:-(

Google PageRank

- Given a document D , the documents page rank at step n is:

$$P_n(D) = (1 - \lambda) P_0(D) + \lambda \left(\sum_{I \text{ linking to } D} P_{n-1}(I) P(D|I) \right)$$

- where

$P(D | I)$: probability that the monkey reaches page D through page I ($= 1 / \text{\#outlinks of } I$)

λ : probability that the follows a link

$1-\lambda$: probability that the monkey types a url

Early termination (1)

- Suppose we re-sort the document ids for each posting such that the best documents come first
 - e.g., sort document identifiers for "*retrieval*" by their tf.idf values.
 - [*<retrieval*; 7; [98, 23, 180, 81, 98, 121, 2, 126,]]>
 - then: top 10 documents for the query "*retrieval*" can be retrieved very quickly: stop after processing the first 10 document ids from the posting list!
 - but compression and merging (multi-word queries) of postings no longer possible...

Early termination (2)

- Trick 3: define a static (or global) ranking of all documents
 - such as Google PageRank (!)
 - re-assign document identifiers by ascending PageRank
 - For every term, documents with a high PageRank are in the initial part of the posting list
 - Estimate the selectivity of the query and only process part of the posting files.

Q8: Estimate the time when
early termination is implemented

?

Early termination (3)

- Probability that a document contains a term:
 - $1 \text{ billion} / 10 \text{ billion} = 0.1$ for *information*
 - $10 \text{ million} / 10 \text{ billion} = 0.001$ for *retrieval*
- Assume independence between terms:
 - $0.1 \times 0.001 = 0.0001$ of the documents contains both terms
 - so, every $1 / 0.0001 = 10,000$ documents on average contains *information AND retrieval*.
 - for top 30, process 3,000,000 documents.
 - $3,000,000 / 10 \text{ billion} = 0.0003$ of the posting files

Query processing on compressed index with early termination

- process about 0.0003 of postings:
 - $0.0003 * 750 \text{ Mb} = 225 \text{ kb}$ for *information*
 - $0.0003 * 7.5 \text{ Mb} = 2.25 \text{ kb}$ for *retrieval*
- Hard disk transfer time:
 - 2 msec. for *information* + 0.02 msec. for *retrieval*
 - (NB now, ignoring CPU time, disk latency and decompressing time is no longer reasonable, so it is likely that it takes some more time)

Query processing on compressed index with early termination

- process about 0.0003 of postings:
 - $0.0003 * 750 \text{ Mb} = 225 \text{ kb}$ for *information*
 - $0.0003 * 7.5 \text{ Mb} = 2.25 \text{ kb}$ for *retrieval*
- Hard disk transfer time:
 - 2 msec. for *information* + 0.02 msec. for *retrieval*
 - (NB now, ignoring CPU time, disk latency and decompressing time is no longer reasonable, so it is likely that it takes some more time)

Query processing – Continued (2)

- We just brought query processing down from more than 3.5 days to about 2 ms. ! :-)

“This engine is incredibly, amazingly, ridiculously fast!”

(from “Top Gear”)

Indexing - Recap

- Inverted files
 - dictionary & postings
 - merging of posting lists
 - delta encoding + variable byte encoding
 - static ranking + early termination
- Put the entire web index on a desktop PC and search it in reasonable time:
 - a) probably*

Ingredients of this talk:

1. A bit of high school mathematics
2. Zipf's law
3. Indexing

Shake well...

Summary

- Search engines
 - Google: first steps and now
- Indexing techniques (inverted files)
 - How to index the web
- Compression, coding, and optimization
 - How to search terabytes