Index Construction

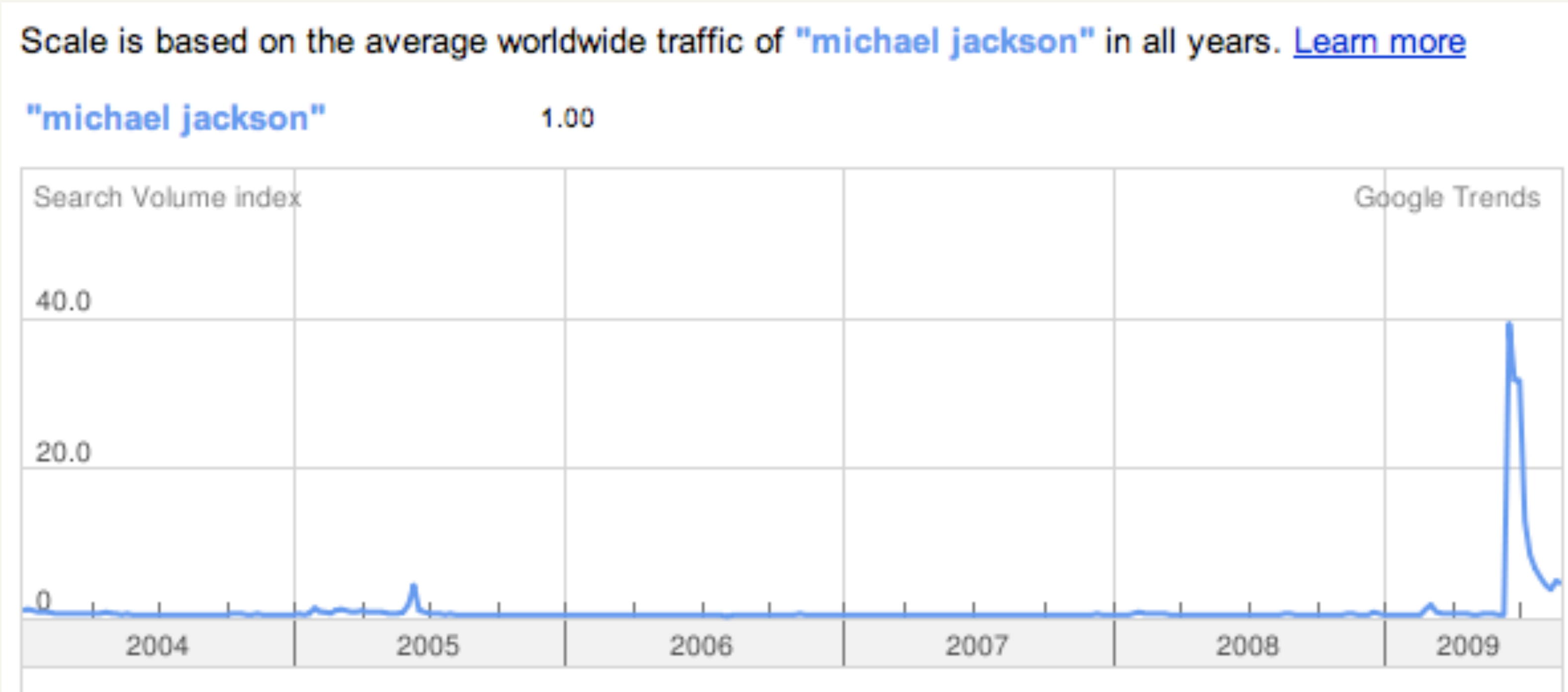
David Kauchak
cs160
Fall 2009

adapted from:
http://www.stanford.edu/class/cs276/handouts/lecture4-indexconstruction.ppt

Administrative

- Homework 2
- Issues with assignment 1?
- Assignment handin procedure on course web page

Google trends: “Michael Jackson”

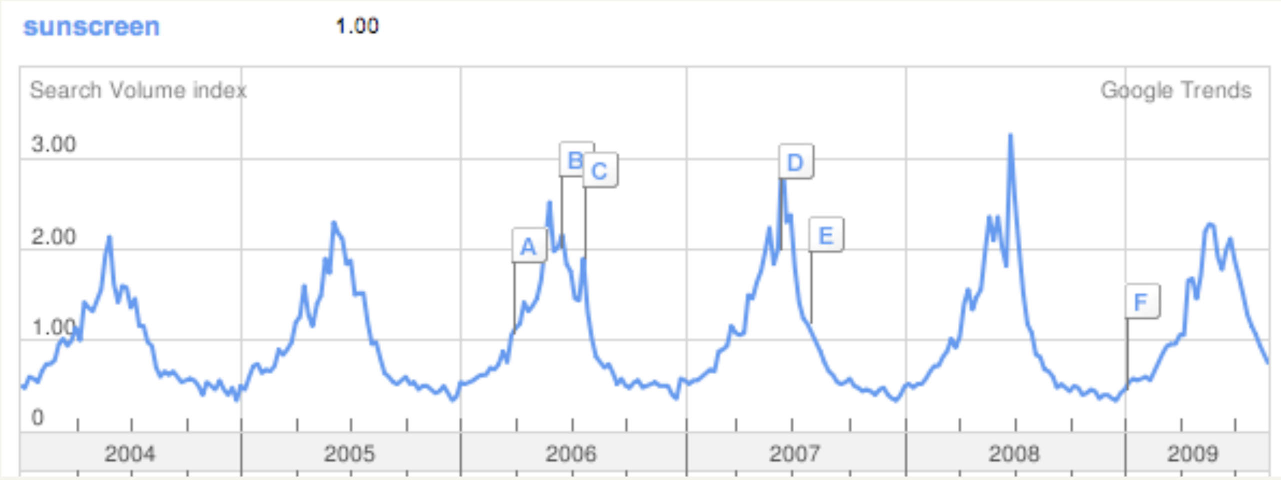


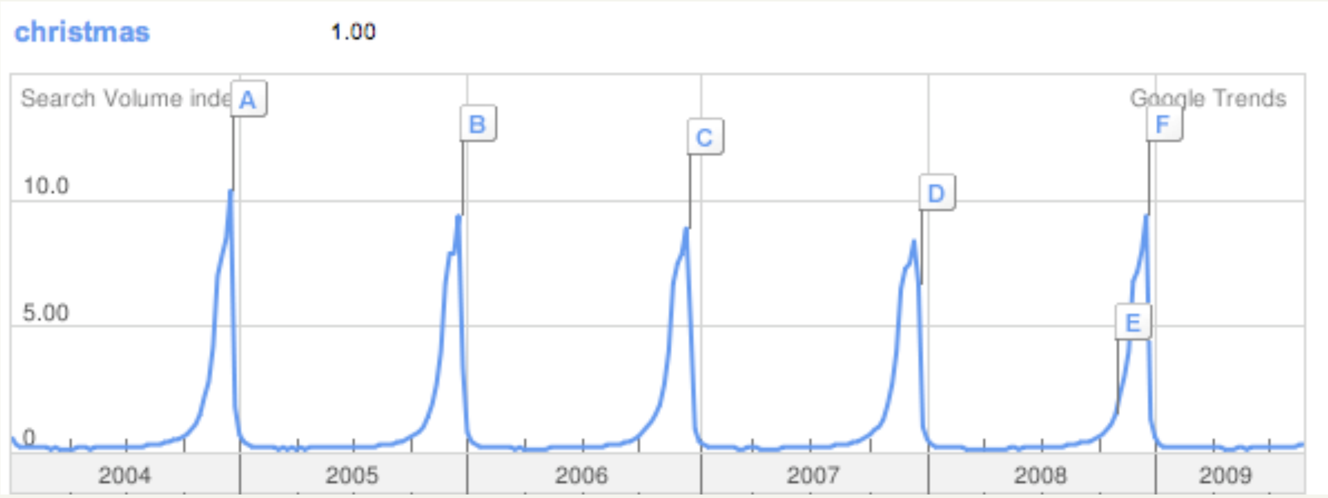
~16.6 Million queries for “Michael Jackson” in Aug.

https://adwords.google.com/select/KeywordToolExternal

Google trends: "fires"

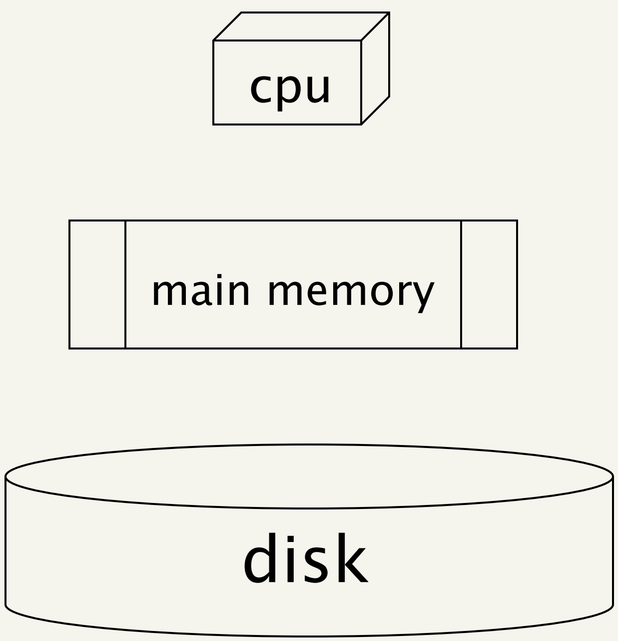
Google trends: cyclical queries



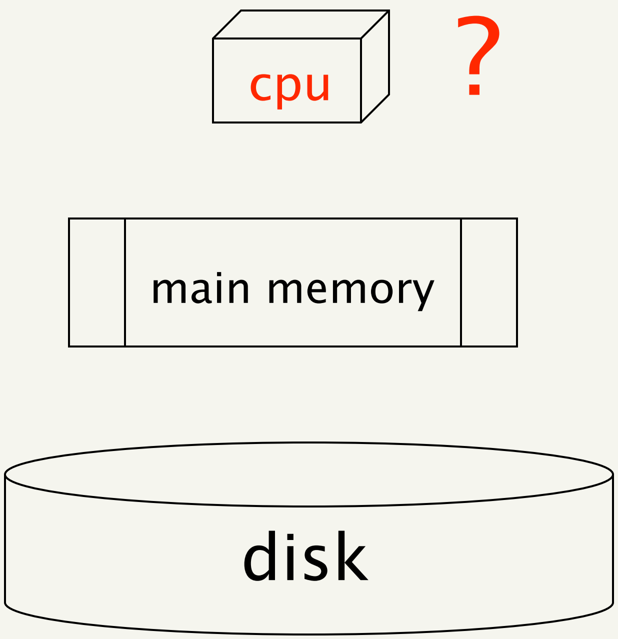


Hardware basics

- Many design decisions in information retrieval are based on the characteristics of hardware

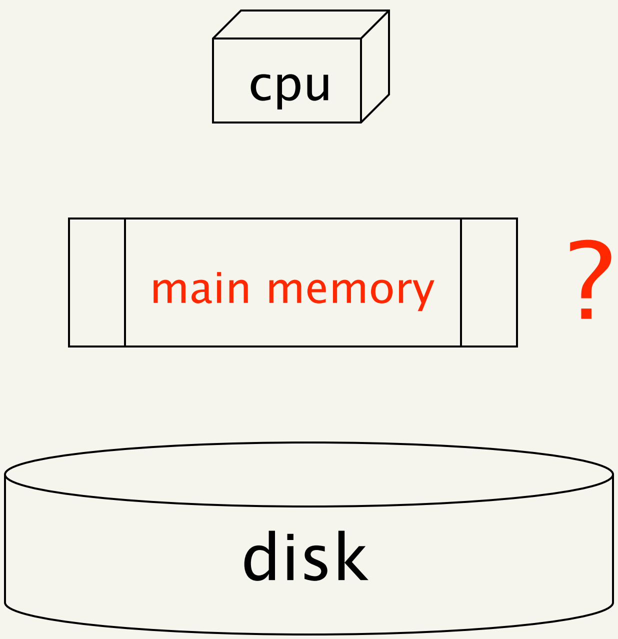


Hardware basics



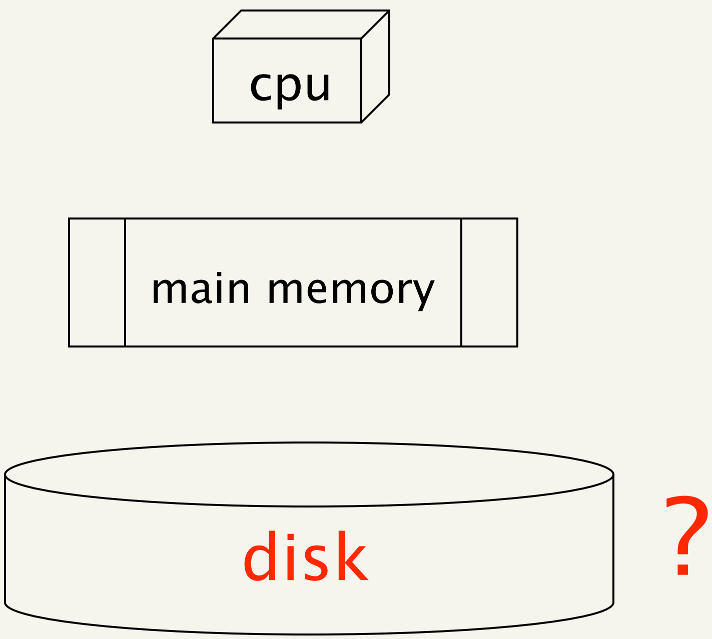
- fast, particularly relative to hard-drive access times
- gigahertz processors
- multi-core
- 64-bit for larger workable address space

Hardware basics



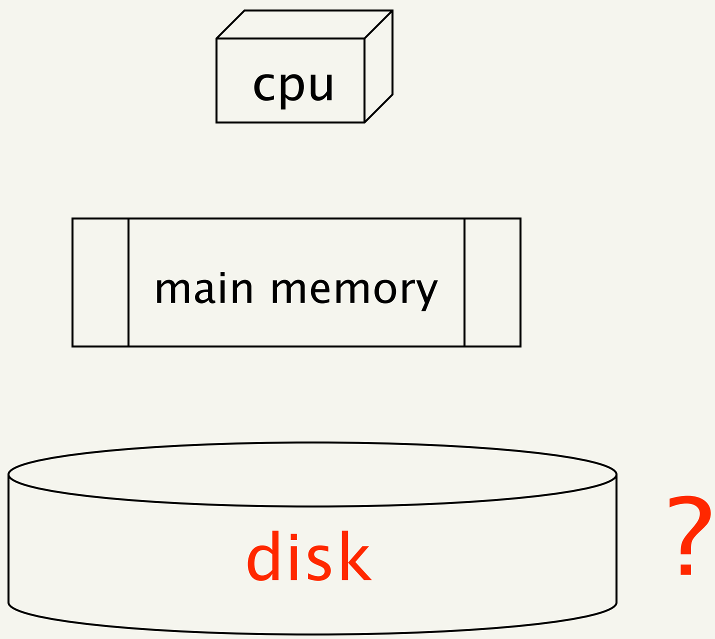
- GBs to 10s of GBs for servers
- main memory buses run at hundreds of megahertz
- ~random access

Hardware basics



- No data is transferred from disk while the disk head is being positioned
- Transferring one large chunk of data from disk to memory is faster than transferring many small chunks
- Disk I/O is block-based: Reading and writing of entire blocks (as opposed to smaller chunks).
- Block sizes: 8KB to 256 KB.

Hardware basics



- 100s of GBs to TBs
- average seek time 5 ms
- transfer time per byte 0.02 μs

RCV1: Our corpus for this lecture

- As an example for applying scalable index construction algorithms, we will use the Reuters RCV1 collection
- This is one year of Reuters newswire (part of 1995 and 1996)
- Still only a moderately sized data set

Extreme conditions create rare Antarctic clouds

Tue Aug 1, 2006 3:20am ET

Email This Article | Print This Article | Reprints

[-] Text [+]

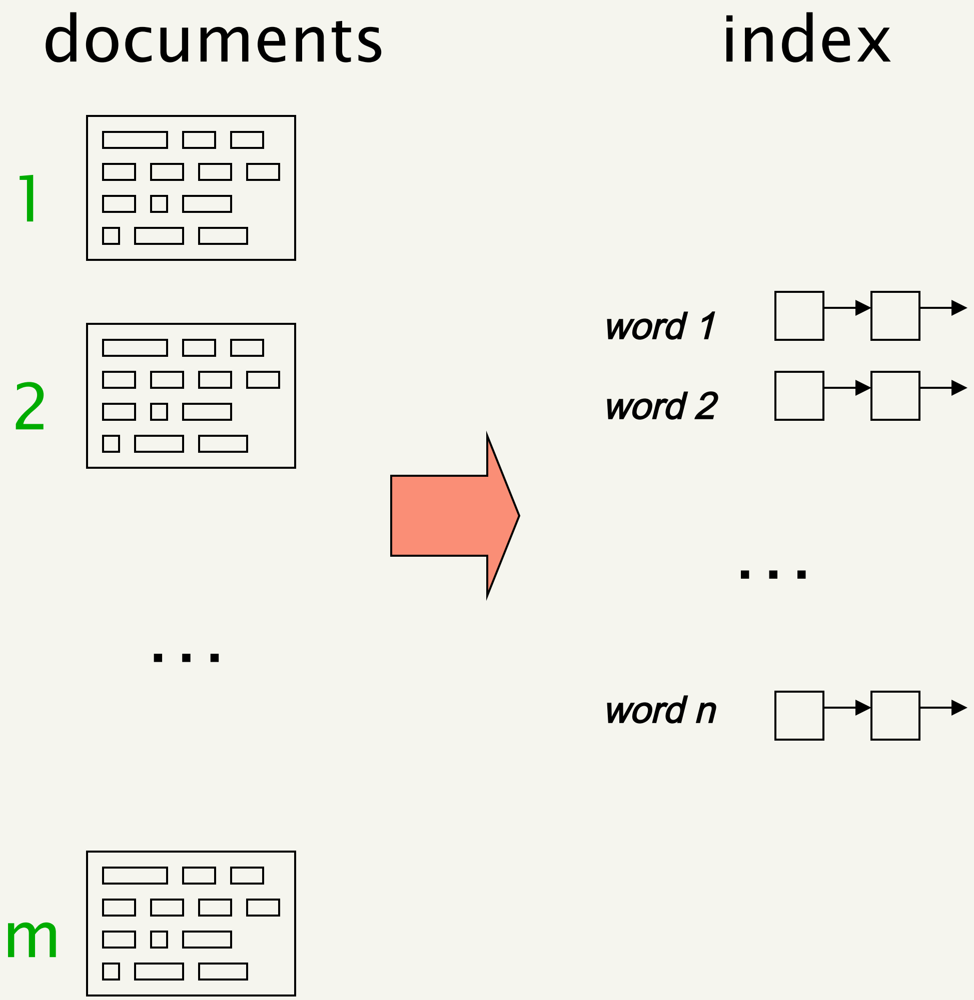
SYDNEY (Reuters) - Rare, mother-of-pearl colored clouds caused by extreme weather conditions above Antarctica are a possible indication of global warming, Australian scientists said on Tuesday.

Known as nacreous clouds, the spectacular formations showing delicate wisps of colors were photographed in the sky over an Australian meteorological base at Mawson Station on July 25.

Reuters RCV1 statistics

statistic	value
documents	800K
avg. # tokens per doc	200
terms	400K
non-positional postings	100M

Index construction



- Documents are tokenized/ normalized
- Postings lists are sorted by docID

How can we do this?

Index construction: collecting documentIDs

Doc 1

I did enact Julius Caesar I was killed i' the Capitol; Brutus killed me.

Doc 2

So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious

Term	Doc #
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

running time?

Θ(tokens)

memory?

O(1)

now what?

Index construction: sort dictionary

Term	Doc #
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

sort based on terms

Term	Doc #
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

running time?

$\Theta(T \log T)$

memory?

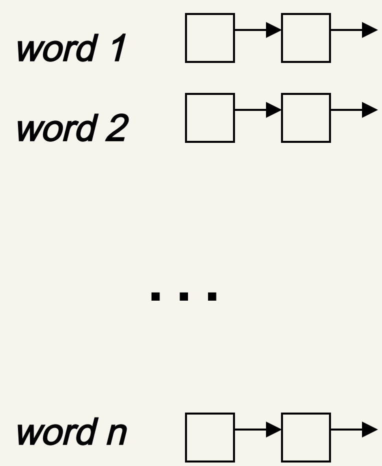
$\Theta(T)$

and then?

Index construction: create postings list

Term	Doc #
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

create postings lists from identical entries



running time?

Θ(tokens)

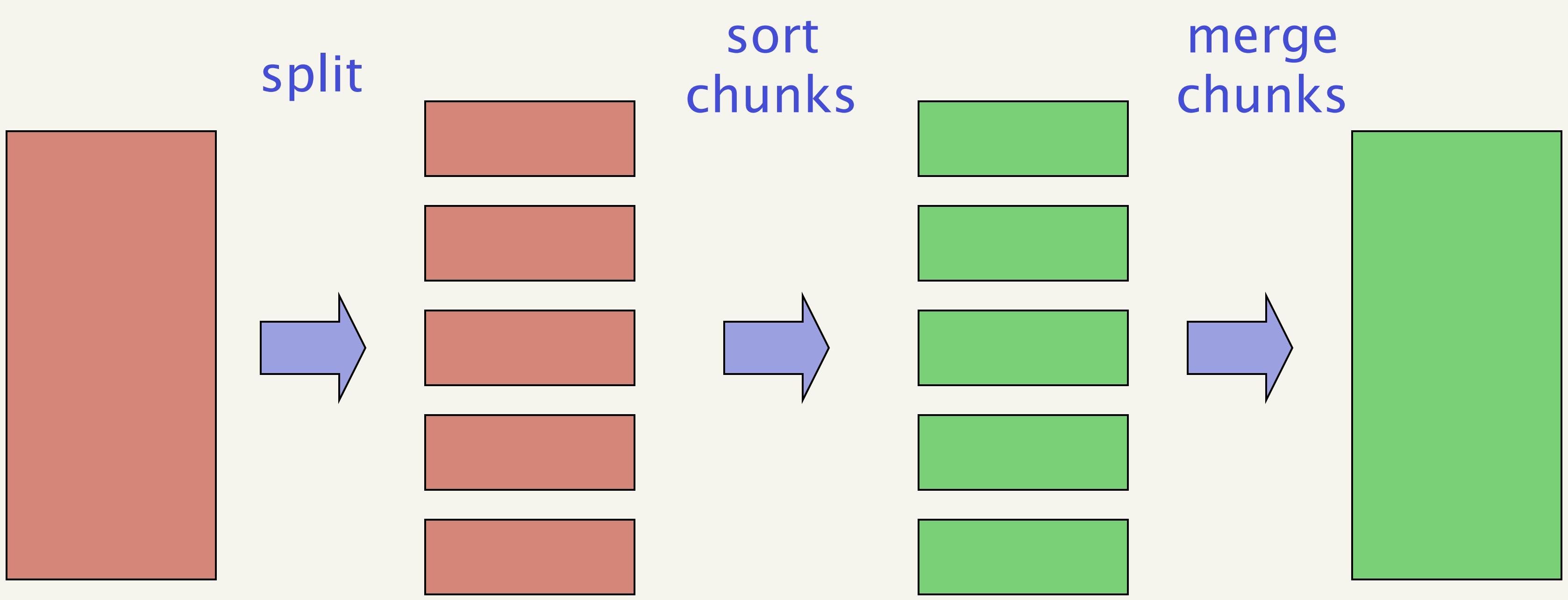
What does this imply about the sorting algorithm?

Scaling index construction

- In-memory index construction does not scale
- What is the major limiting step?
  - both the collecting document IDs and creating posting lists require little memory since it's just a linear traversal of the data
  - sorting is memory intensive! Even in-place sorting algorithms still require O(n) memory
- For RCV1, we could do still do it in memory
- What about larger data sets?
  - On-disk sorting

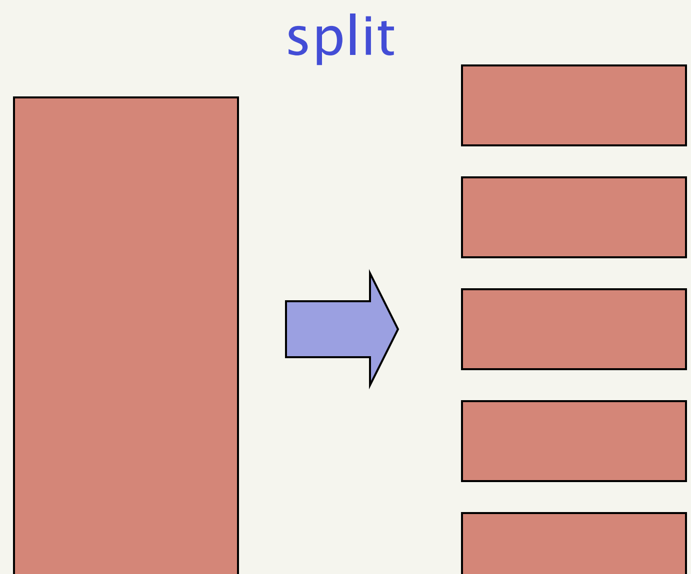
On-disk sorting

- What are our options?
  - Literally, sort on-disk: keep all data on disk. When we need to access entries, access entries
    - Random access on disk is slow……
  - Break up list into chunks. Sort chunks, then merge chunks (e.g. unix "merge" function)



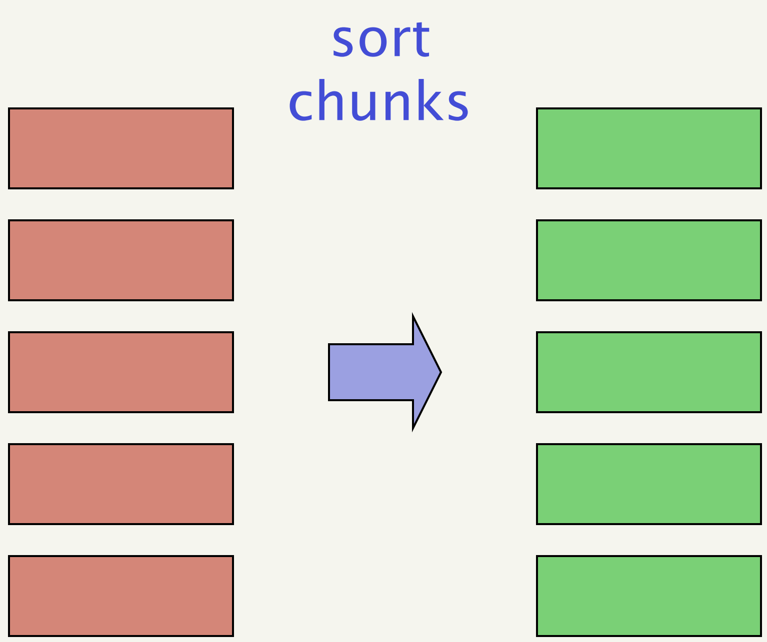
On-disk sorting

- Can do this while processing
- When we reach a particular size, start the sorting process



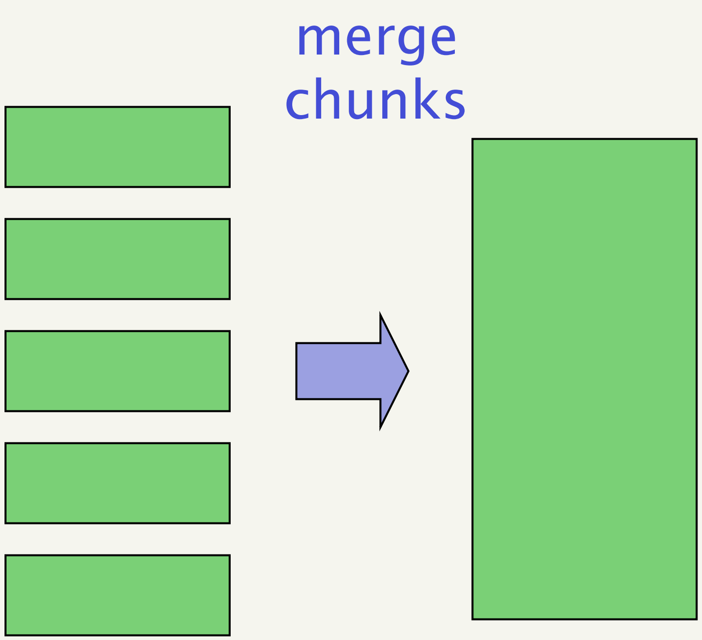
On-disk sorting

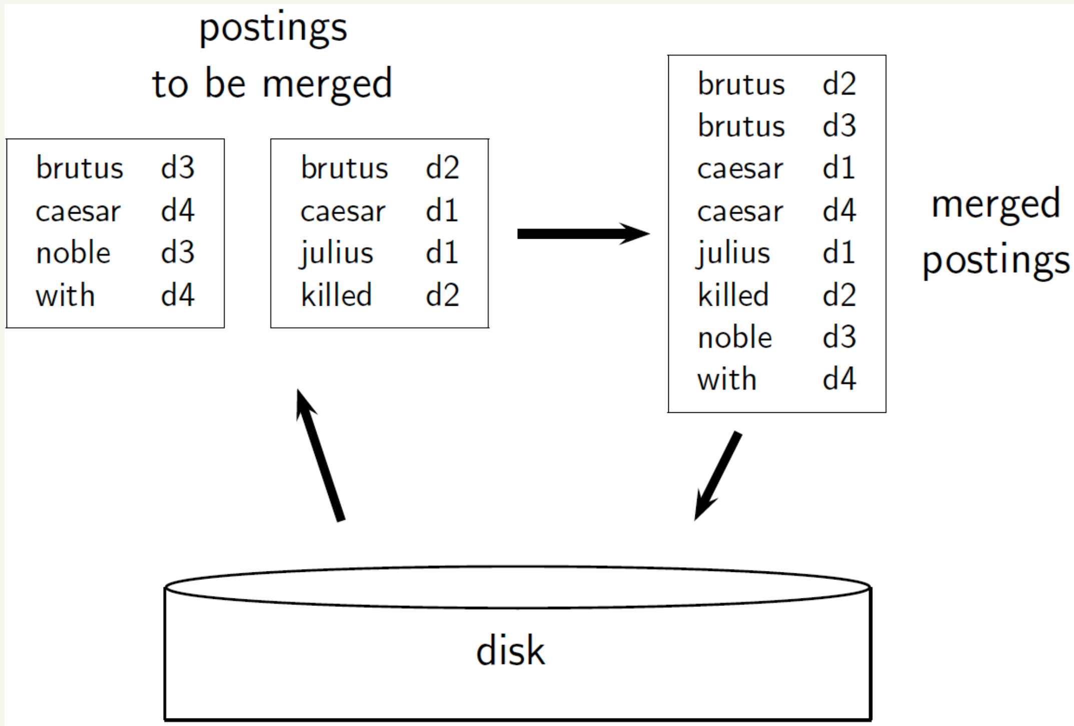
- We can pick the chunk size so that we can sort the chunk in memory
- Generally, pick as large a chunk as possible while still being able to sort in memory



On-disk sorting

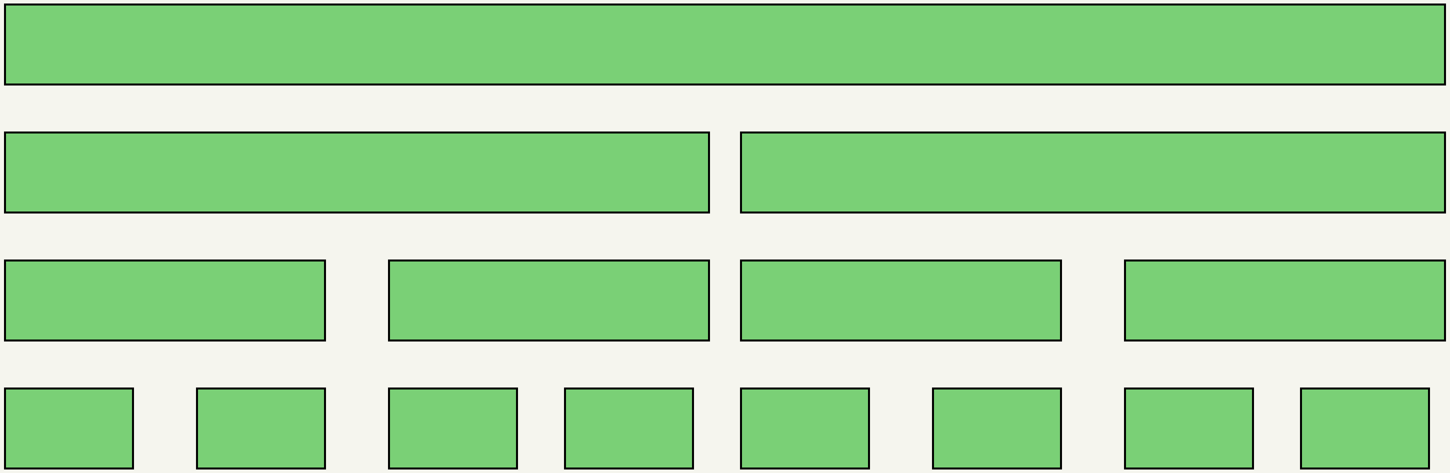
- How can we do this?





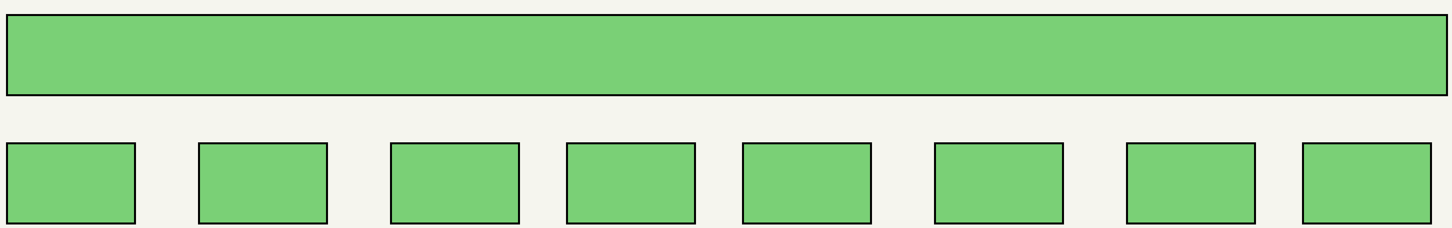
Binary merges

- Can do binary merges, with a merge tree
- For *n* chunks, how many levels will there be?
  - log(n)



n-way merge

- More efficient to do an *n*-way merge, where you are reading from all blocks simultaneously
- Providing you read decent-sized chunks of each block into memory, you’re not killed by disk seeks
- Only one level of merges!
- Is it linear?



Another approach: SPIMI

- Sorting can still be expensive
- Is there any way to do the indexing without sorting?
- Accumulate posting lists as they occur
- When size gets too big, start a new chunk
- Merge chunks at the end

Another approach

Doc 1

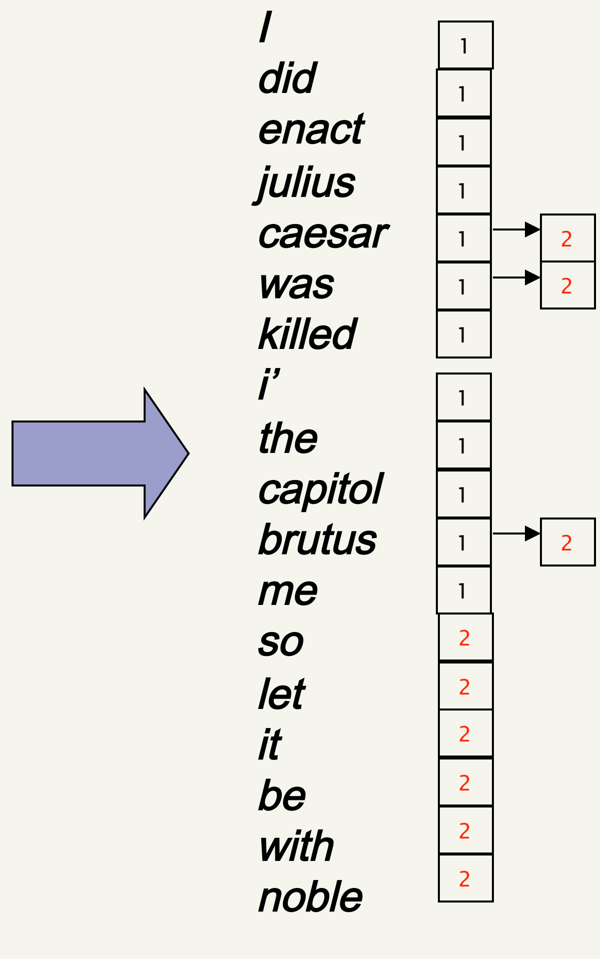
I did enact Julius Caesar I was killed i' the Capitol; Brutus killed me.

I	1
did	1
enact	1
julius	1
caesar	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
me	1

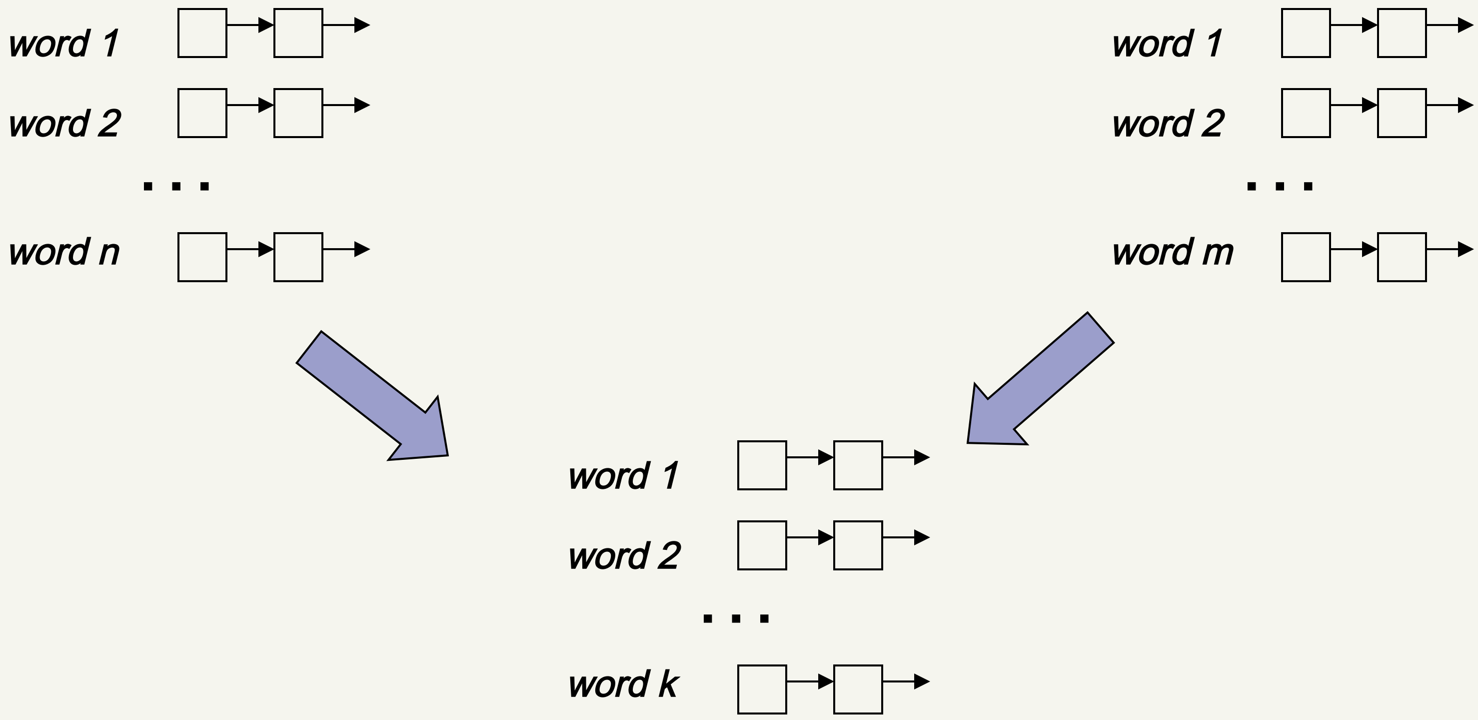
Another approach

Doc 2

So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious



The merge



- Running time?
  - linear in the sizes of the postings list being merged
- As with merging sorted dictionary entries we can either do pairwise binary tree type merging or do an *n*-way merge

Distributed indexing

- For web-scale indexing we must use a distributed computing cluster
- Individual machines are fault-prone
  - Can unpredictably slow down or fail
- How do we exploit such a pool of machines?

Google data centers

- Google data centers mainly contain commodity machines
- Data centers are distributed around the world
- Estimate: a total of 1 million servers, 3 million processors/cores (Gartner 2007)
- Estimate: Google installs 100,000 servers each quarter
  - Based on expenditures of 200–250 million dollars per year
- This would be 10% of the computing capacity of the world!?!

Fault tolerance

- Hardware fails

>30% chance of failure
within 5 years

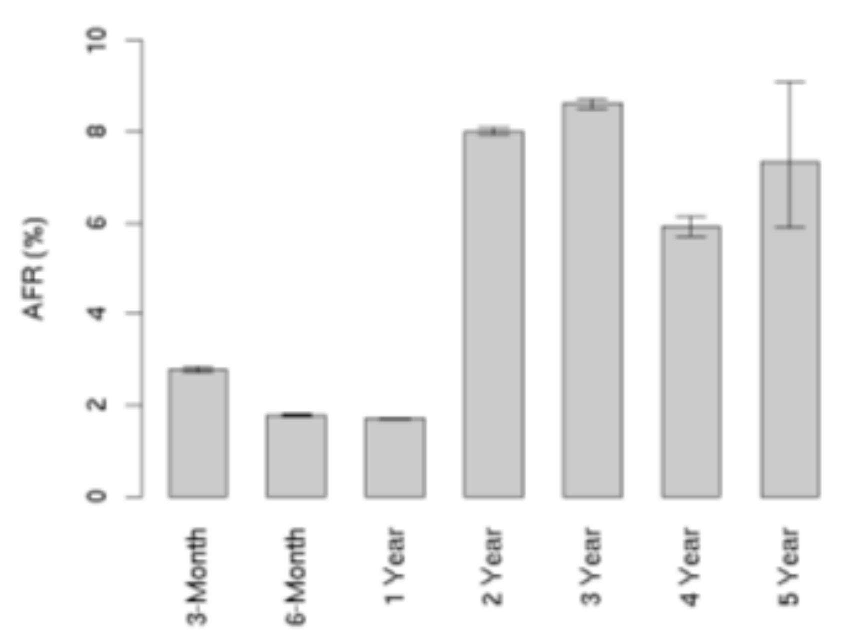


Figure 2: Annualized failure rates broken down by age groups

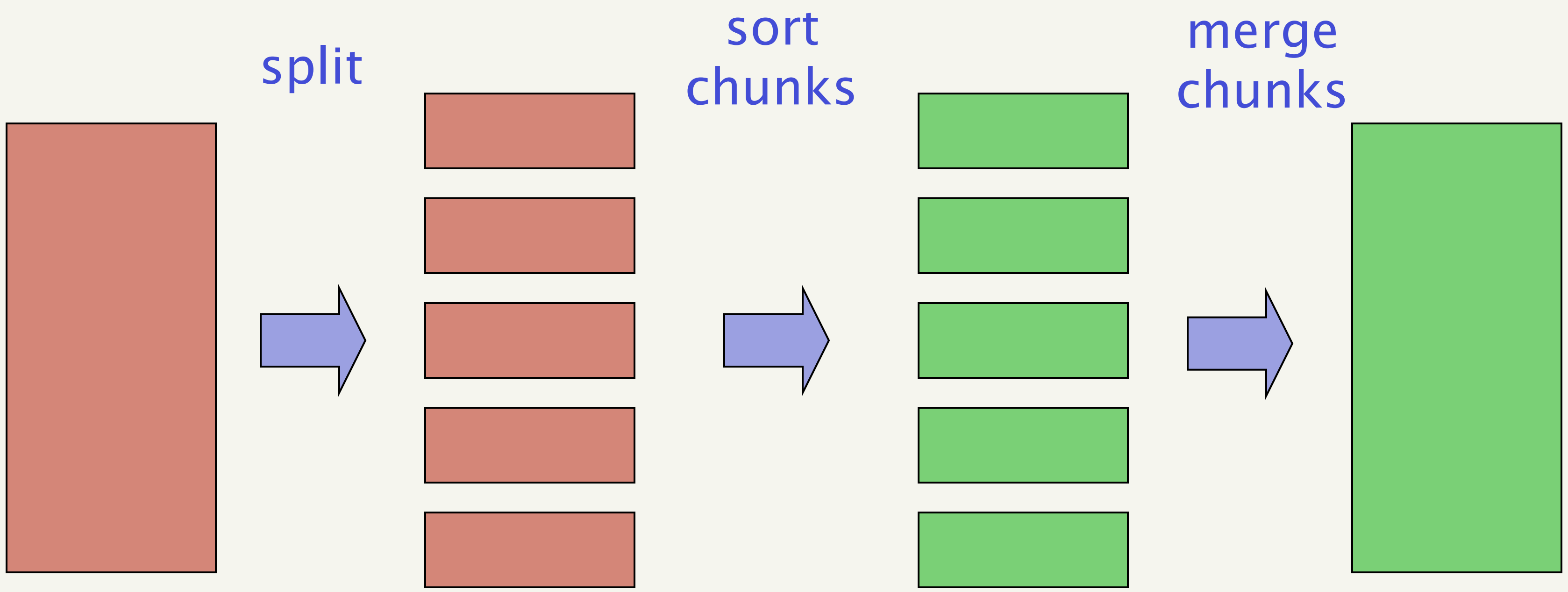
http://labs.google.com/papers/disk_failures.pdf

- What happens when you have 1 million servers?
- Hardware is always failing!

Distributed indexing

- Maintain a *master* machine directing the indexing job – considered “safe”
- Break up indexing into sets of (parallel) tasks
- Master machine assigns each task to an idle machine from a pool
- Besides speed, one advantage of a distributed scheme is fault tolerance

Distributed indexing

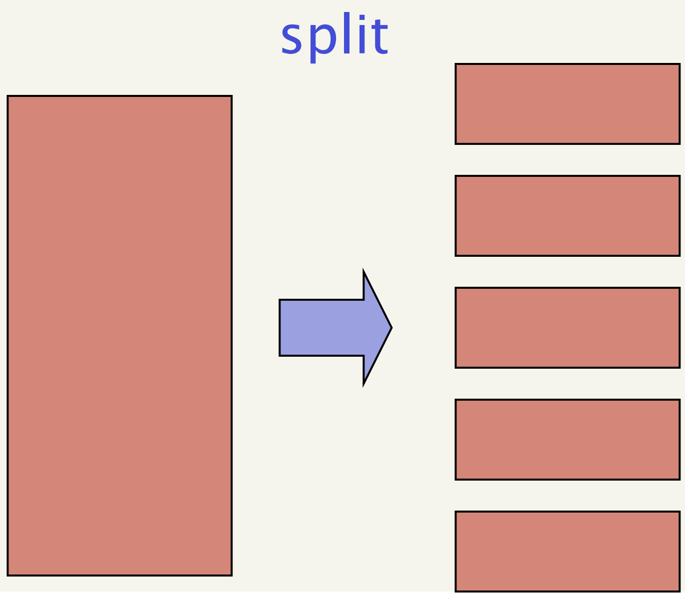


Can we break these steps into parallelizable activities?

Specify exactly how we split the data

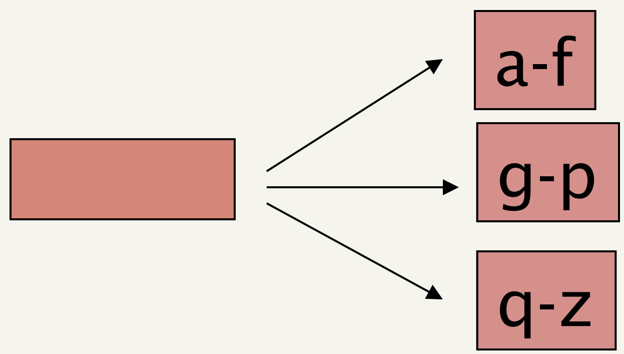
Parallel tasks

- We will use two sets of parallel tasks
  - Parsers
  - Inverters
- Break the input document corpus into *splits*
- Each split is a subset of documents



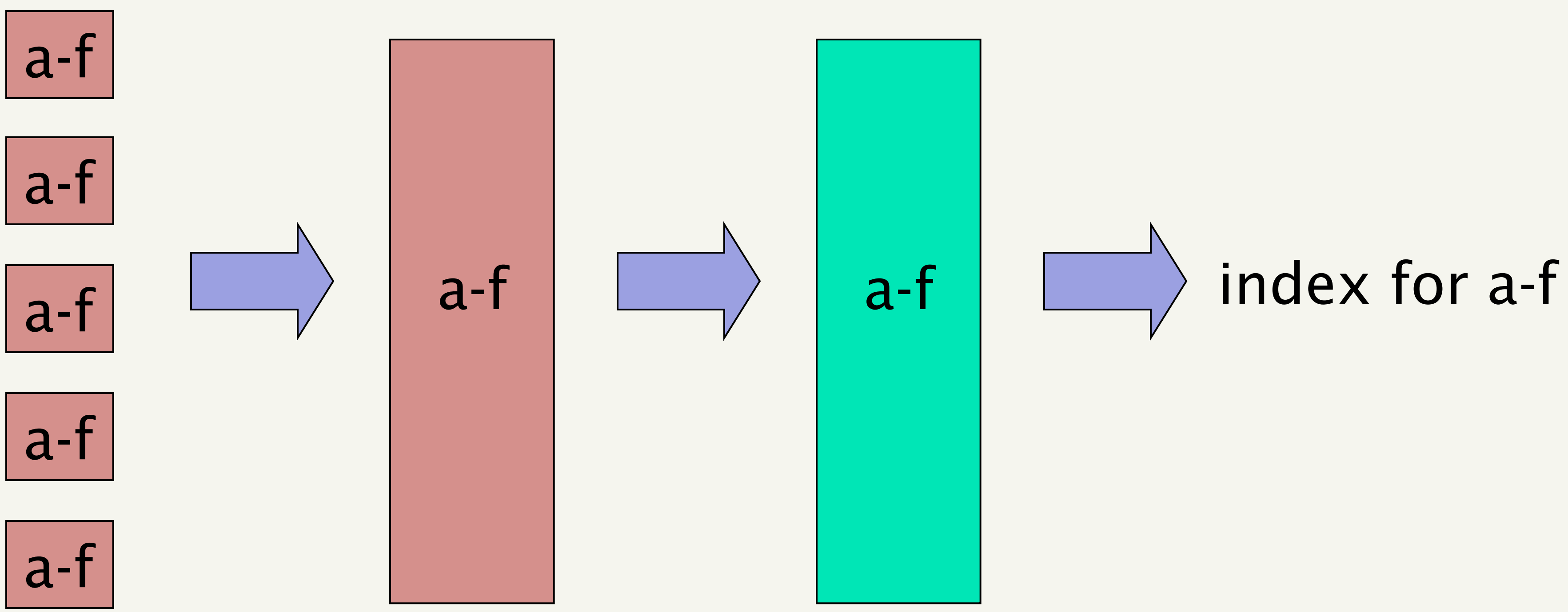
Parsers

- Master assigns a split to an idle parser machine
- Parser reads a document at a time and emits (term, doc) pairs
- Parser writes pairs into *j* partitions
- Each partition is for a range of terms' first letters
  - (e.g., ***a-f, g-p, q-z***) – here *j*=3.

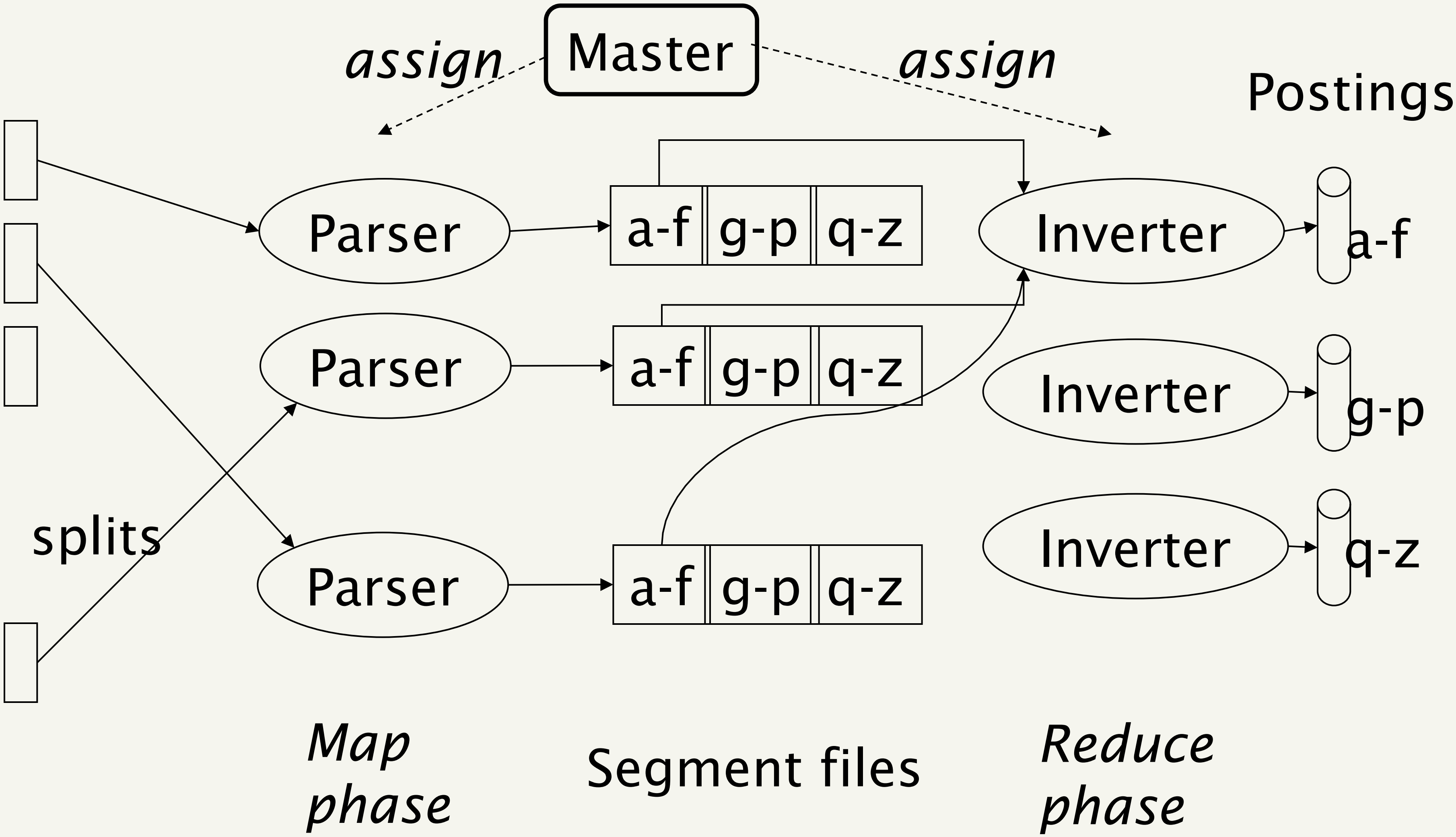


Inverters

- An inverter collects all (term, doc) pairs for one term-partition
- Sorts and writes to postings lists



Data flow


Master
assign
assign
Postings
Parser
Parser
Parser
a-f
g-p
q-z
a-f
g-p
q-z
a-f
g-p
q-z
Inverter
Inverter
Inverter
a-f
g-p
q-z
splits
Map phase
Segment files
Reduce phase

MapReduce

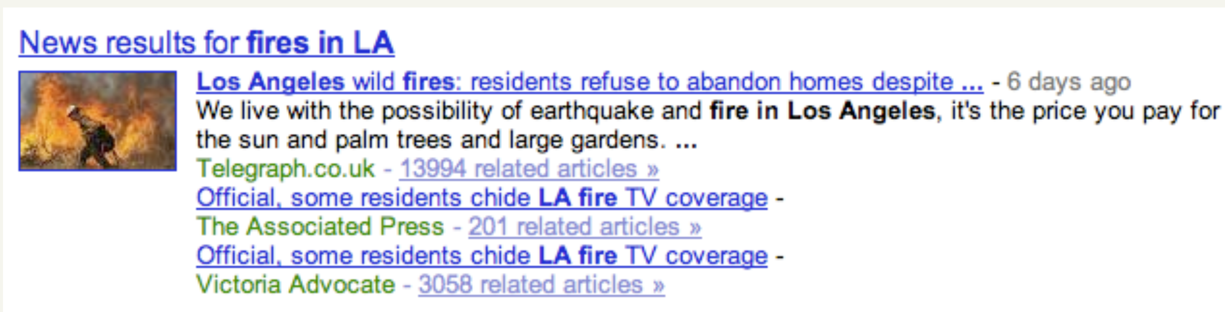
- The index construction algorithm we just described is an instance of MapReduce
- MapReduce (Dean and Ghemawat 2004) is a robust and conceptually simple framework for
- distributed computing without having to write code for the distribution part
- The Google indexing system (ca. 2002) consists of a number of phases, each implemented in MapReduce

MapReduce

- Index construction is just one phase
- After indexing, we need to be ready to answer queries
- There are two ways to we can partition the index
  - *Term-partitioned:* one machine handles a subrange of terms
  - *Document-partitioned:* one machine handles a subrange of documents
- Which do you think search engines use? Why?

Dynamic indexing

- Up to now, we have assumed that collections are static
- What happens when we need to
  - add a document
  - remove a document
  - modify the contents of a document



- This means that the dictionary and postings lists have to be modified:
  - Postings updates for terms already in dictionary
  - New terms added to dictionary

Dynamic indexing

- What are our options?
  - Rebuild the index from scratch
  - Update the index each time
  - Keep an auxiliary index with all of the new changes

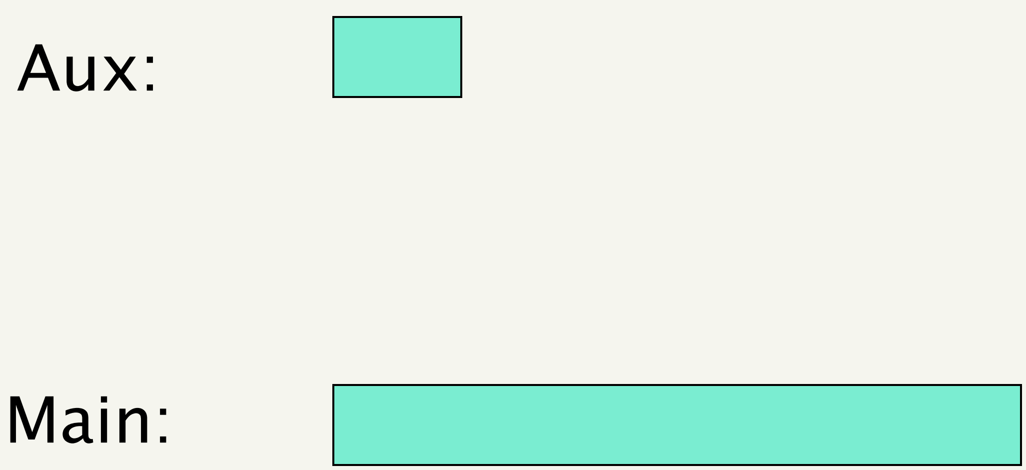
Common approach auxiliary index

- Maintain “big” main index
- New docs go into “small” auxiliary index
- Deletions
  - Invalidation bit-vector for deleted docs
  - Filter docs output on a search result by this invalidation bit-vector
- What is the cost of a search now?
  - still basically the same
  - search across both, merge results

Auxiliary index

- To make changes efficient for the auxiliary index, it should be small enough to fit in main memory
  - Otherwise, we're back to where we started with updating an on-disk index
- What happens when this index gets to big to fit in to memory?
- We need to merge it in to the main index

Merging

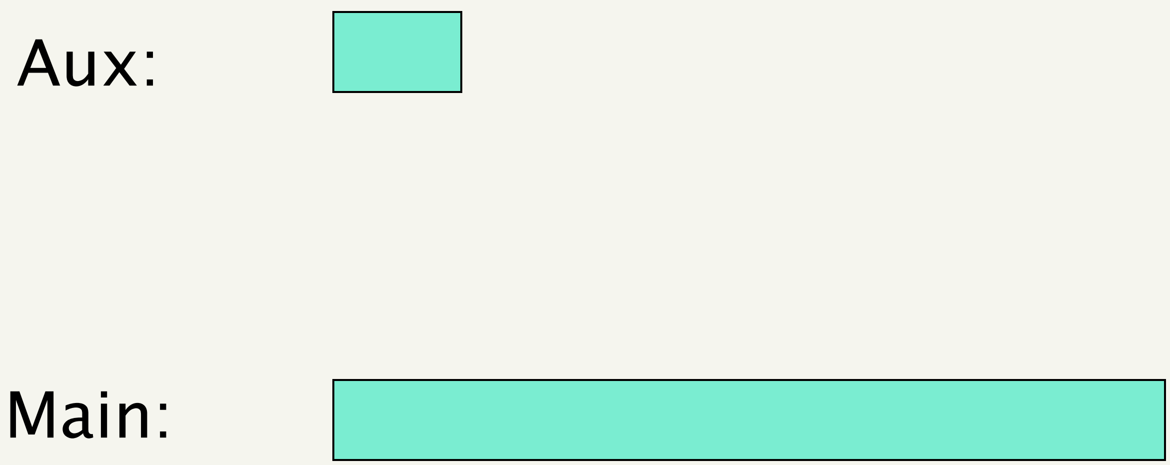


Merging

Aux:

Main:

Merging



Merging

Aux:

Main:

Every time we merge we merge with the entire index

Can we do better?

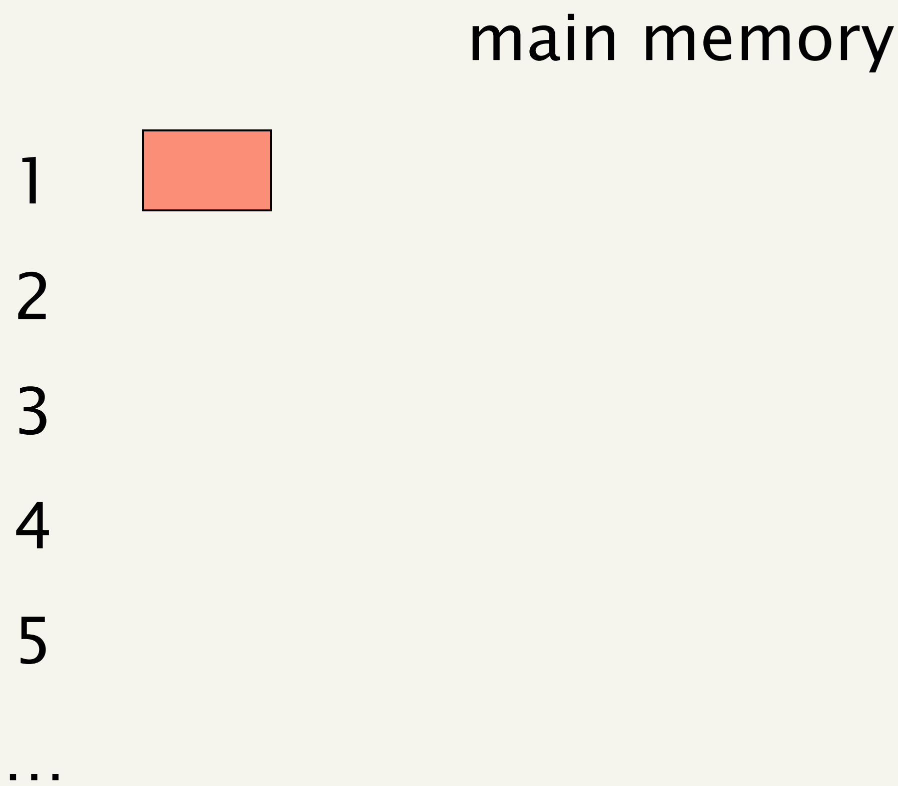
Logarithmic merge

- Maintain a series of indexes, each twice as large as the previous one.
- Keep smallest (Z_0) in memory
- Larger ones (I_0, I_1, …) on disk
- If Z_0 gets too big (> *n*), write to disk as I_0
- or merge with I_0 (if I_0 already exists) as Z_1
- Either write merge Z_1 to disk as I_1 (if no I_1)
- Or merge with I_1 to form Z_2
- etc.

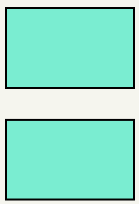
Logarithmic merge

main memory

Logarithmic merge



Logarithmic merge



main memory

1

2

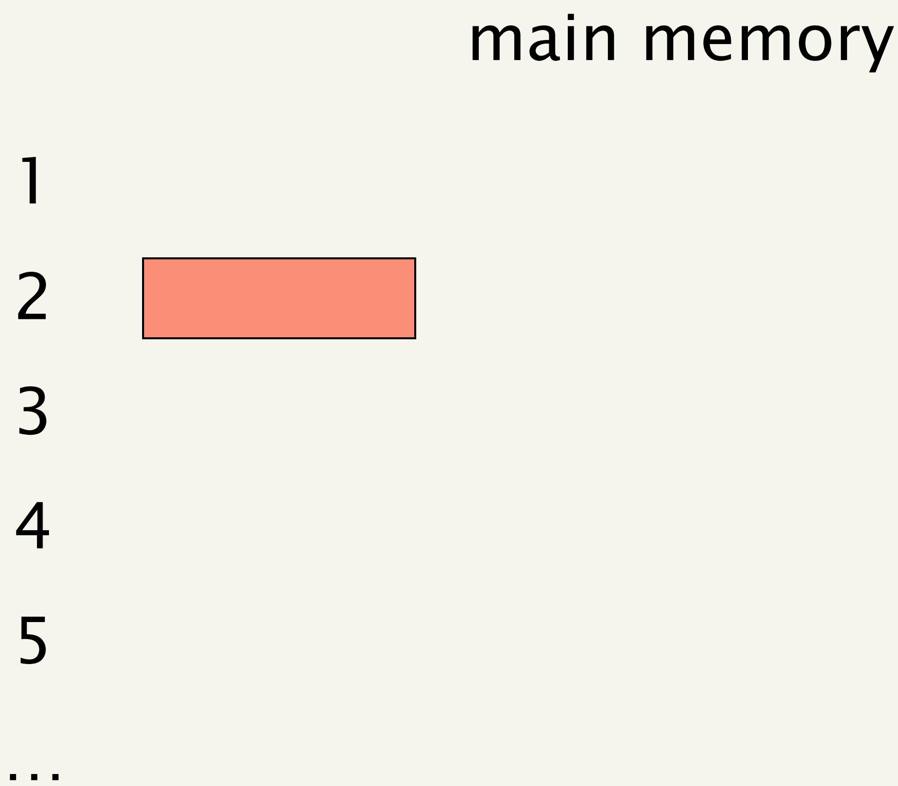
3

4

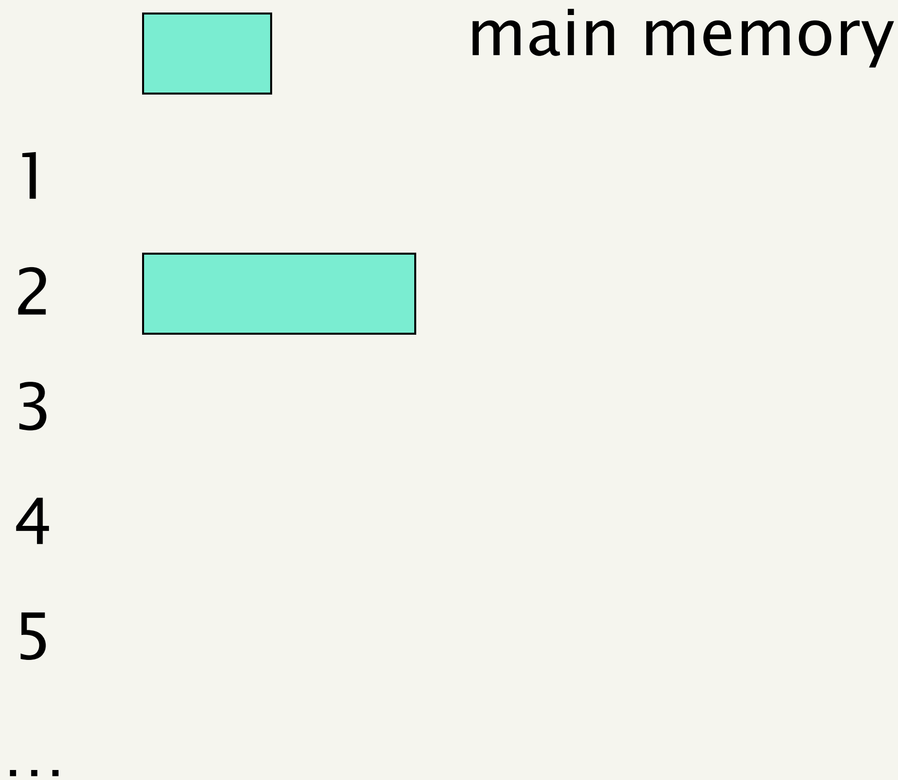
5

…

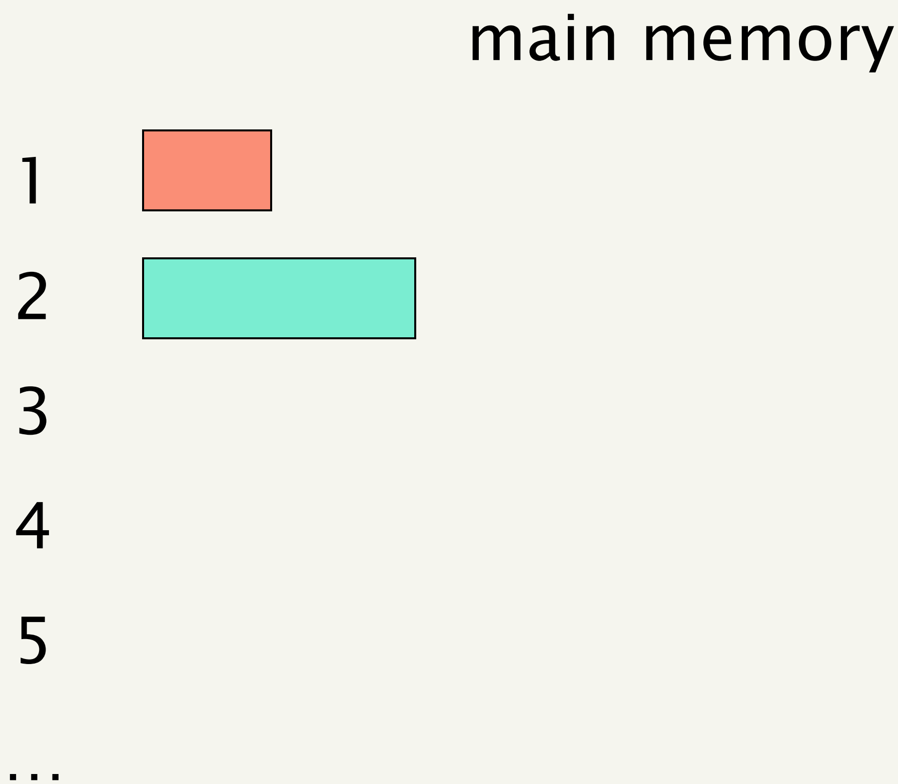
Logarithmic merge



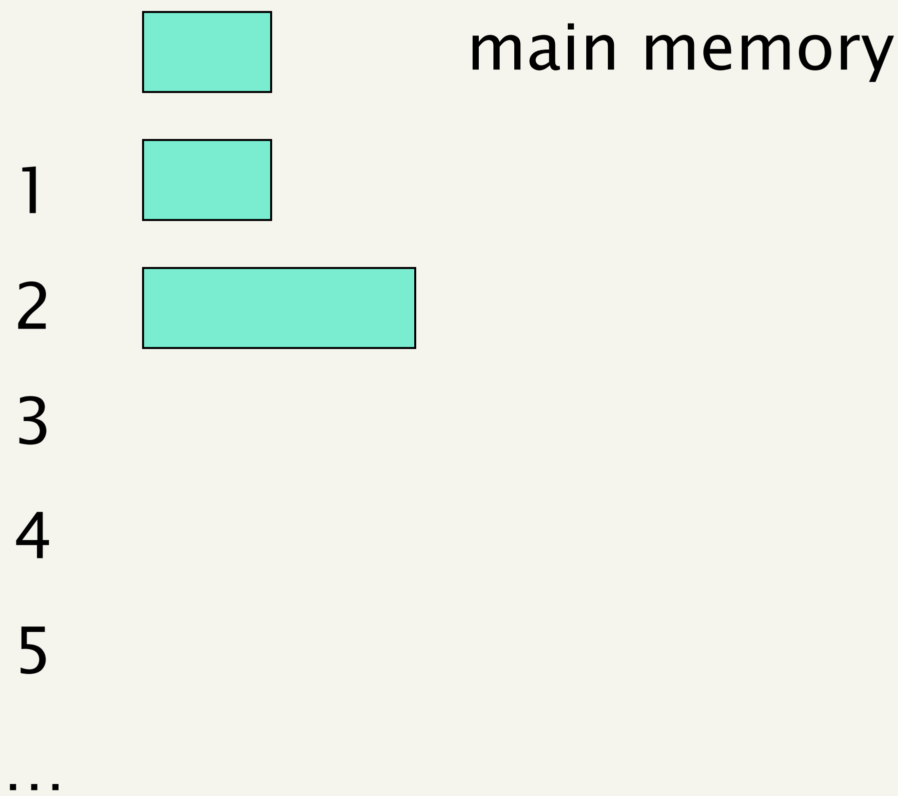
Logarithmic merge



Logarithmic merge

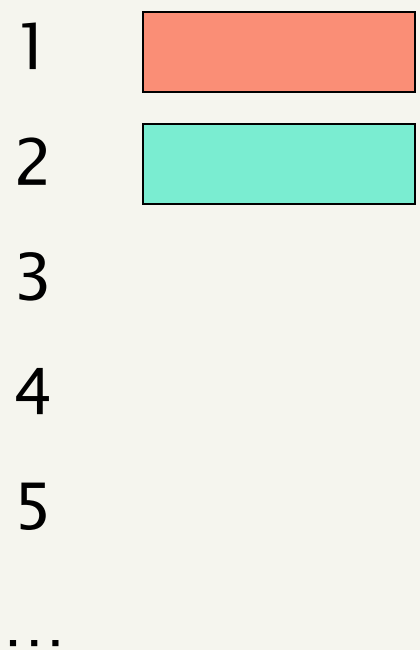


Logarithmic merge



Logarithmic merge

main memory



Logarithmic merge

main memory

1

2

3

4

5

…

Logarithmic merge

- Logarithmic merging is much more efficient for index construction
- On the order of linear vs. logarithmic
- What's the downside?
  - But query processing now requires the merging of O(log T) indexes
  - We can alleviate this some by using parallel resources

Dynamic indexing at search engines

- All the large search engines do dynamic indexing
- Their indices have frequent incremental changes
  - News items, new topical web pages
- But (sometimes/typically) they also periodically reconstruct the index from scratch
  - Query processing is then switched to the new index, and the old index is then deleted

« Local Store And Inventory Data Poised To Transform "Online Shopping" | Main | SEO Company, Fathom Online, Acquired By Geary Interactive »

Mar 31, 2008 at 8:45am Eastern by Barry Schwartz

Google Dance Is Back? Plus Google's First Live Chat Recap & Hyperactive Yahoo Slurp

Is the Google Dance back? Well, not really, but I am noticing Google Dance-like behavior from Google based on reading some of the feedback at a WebmasterWorld thread.

The Google Dance refers to how years ago, a change to Google's ranking algorithm often began showing up slowly across data centers as they reflected different results, a sign of coming changes. These days Google's data centers are typically always showing small changes and differences, but the differences between this data center and this one seem to be more like the extremes of the past Google Dances.

So either Google is preparing for a massive update or just messing around with our heads. As of now, these results have not yet moved over to the main Google.com results.

Resources

- Chapter 4 of IIR
- Original publication on MapReduce: Dean and Ghemawat (2004)
- Original publication on SPIMI: Heinz and Zobel (2003)

Remaining problem…

- This approach is scalable, but we can do better
- What is the memory requirement of a chunk?
- We need to store both the dictionary and the postings list
- What is the size of the postings list dependent on?
  - size of the postings list is dependent on the number and size of the documents and…
  - our posting list representation
- What is the size of the dictionary?
  - depends on the number and length of terms
  - Can we do any better?

Remaining problem with sort-based algorithm

- Storing the actual words in the dictionary is expensive
  - For our small corpus, the average length of an entry is 8 characters
  - This increases the larger the corpus. Why?
- Ideas?
- Instead of storing the words, for each word, we store an index
- We then have to keep a universal mapping from term to index

dictionary

ambitious → 0
be → 1
brutus → 2
… …

index

0
1
2
…