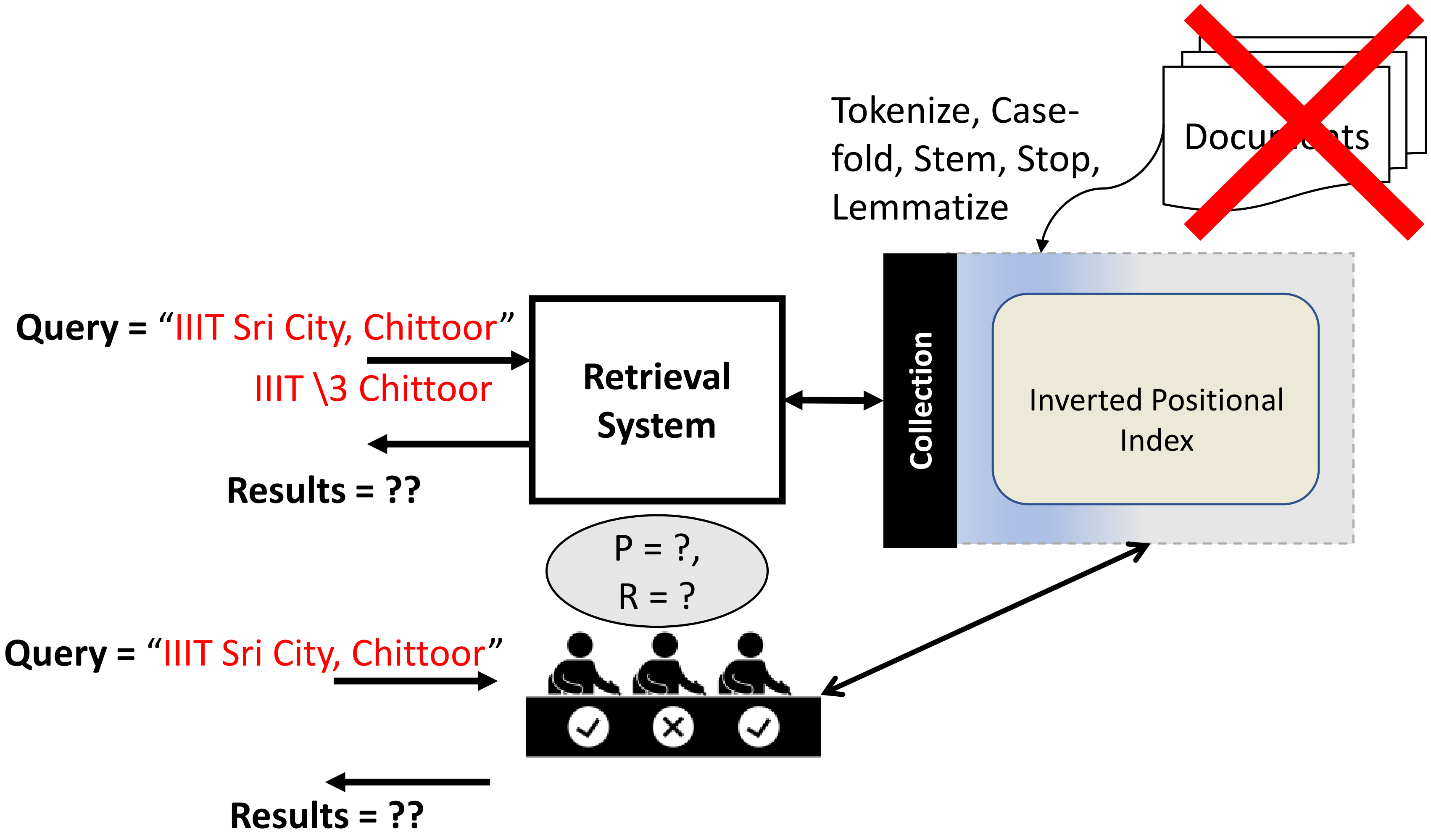
Dynamic Indexing

When the collection gets updated, how to update our index?


Tokenize, Case-fold, Stem, Stop, Lemmatize
Documents
Query = "IIIT Sri City, Chittoor"
IIIT \3 Chittoor
Retrieval System
Results = ??
Collection
Inverted Positional Index
P = ?, R = ?
Query = "IIIT Sri City, Chittoor"
Results = ??

How to deal with changes in the collection?

- Periodically re-index from scratch!

Can we do better?

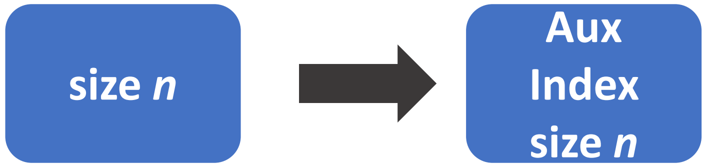
What if new documents must be indexed quickly?

Auxiliary Index

- Keep two indexes
  - Main Index
    - Keep invalidation bit to indicate deletion.
  - Auxiliary Index (for new files)
    - Periodically (or when Aux. Index becomes too large) merge auxiliary index with main index.
- Search in both. Merge the results.

Can we do better?

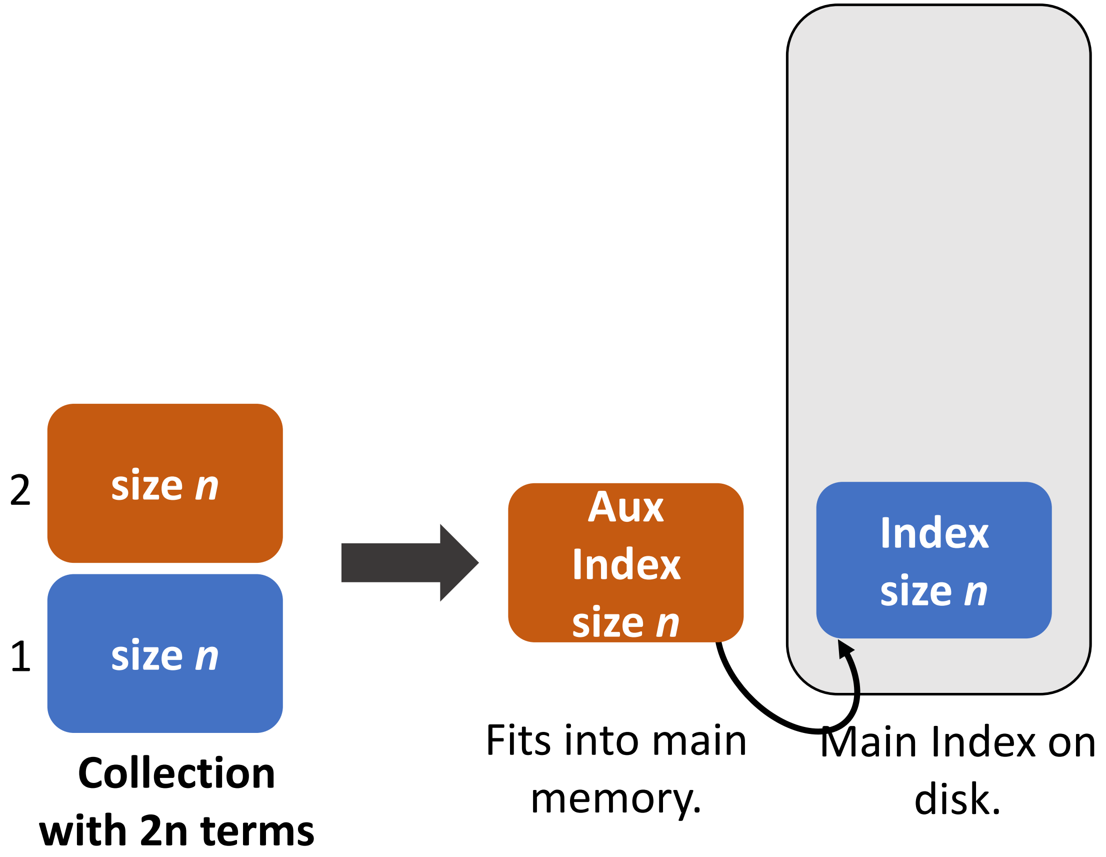
Main Index and Auxiliary Index



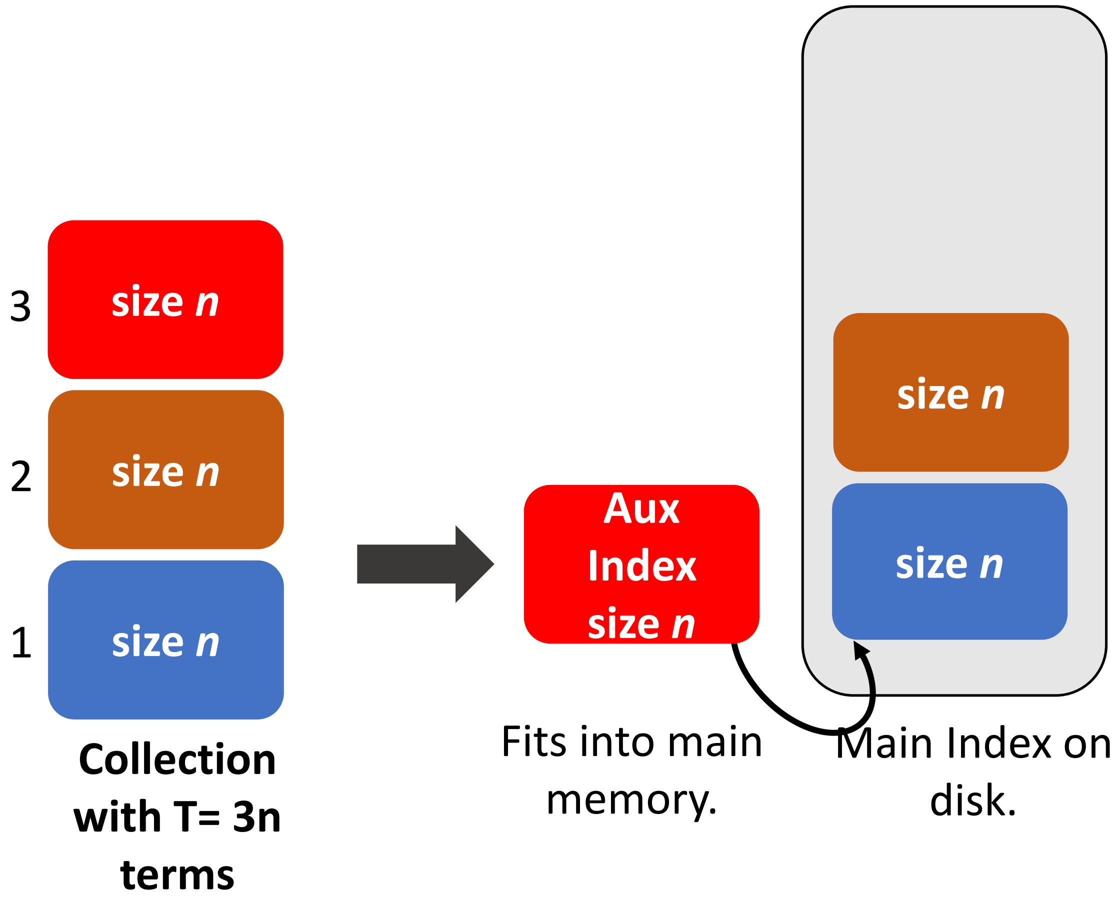
Collection with n terms

Start with a small index that fits into main memory.

Merge

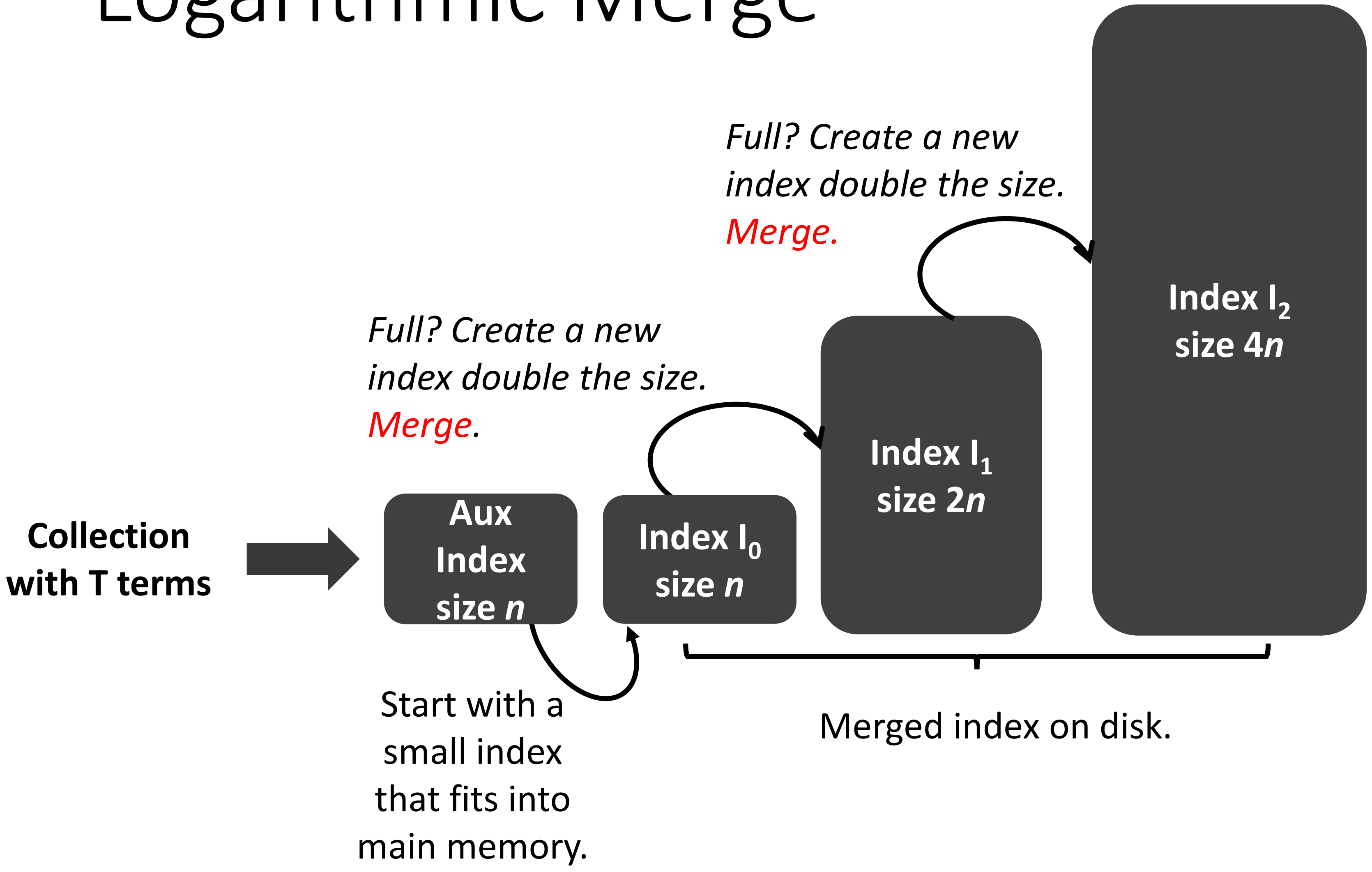

2
size *n*
1
size *n*
Collection with 2n terms
Aux Index size *n*
Fits into main memory.
Index size *n*
Main Index on disk.

Merge



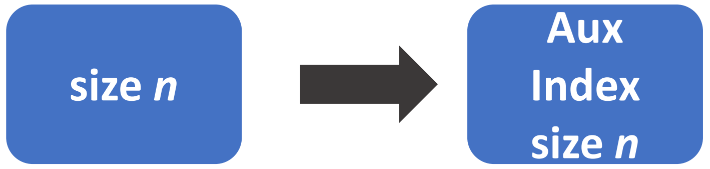
Merge T/n times!

Logarithmic Merge



Full? Create a new index double the size. Merge.
Full? Create a new index double the size. Merge.
Collection with T terms
Aux Index size *n*
Index I_0 size *n*
Index I_1 size 2*n*
Index I_2 size 4*n*
Start with a small index that fits into main memory.
Merged index on disk.

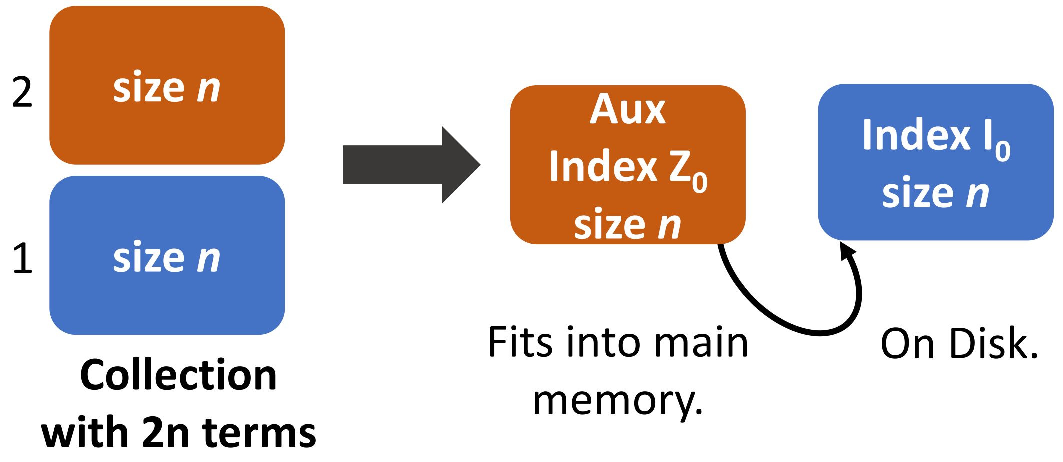
Logarithmic Merge



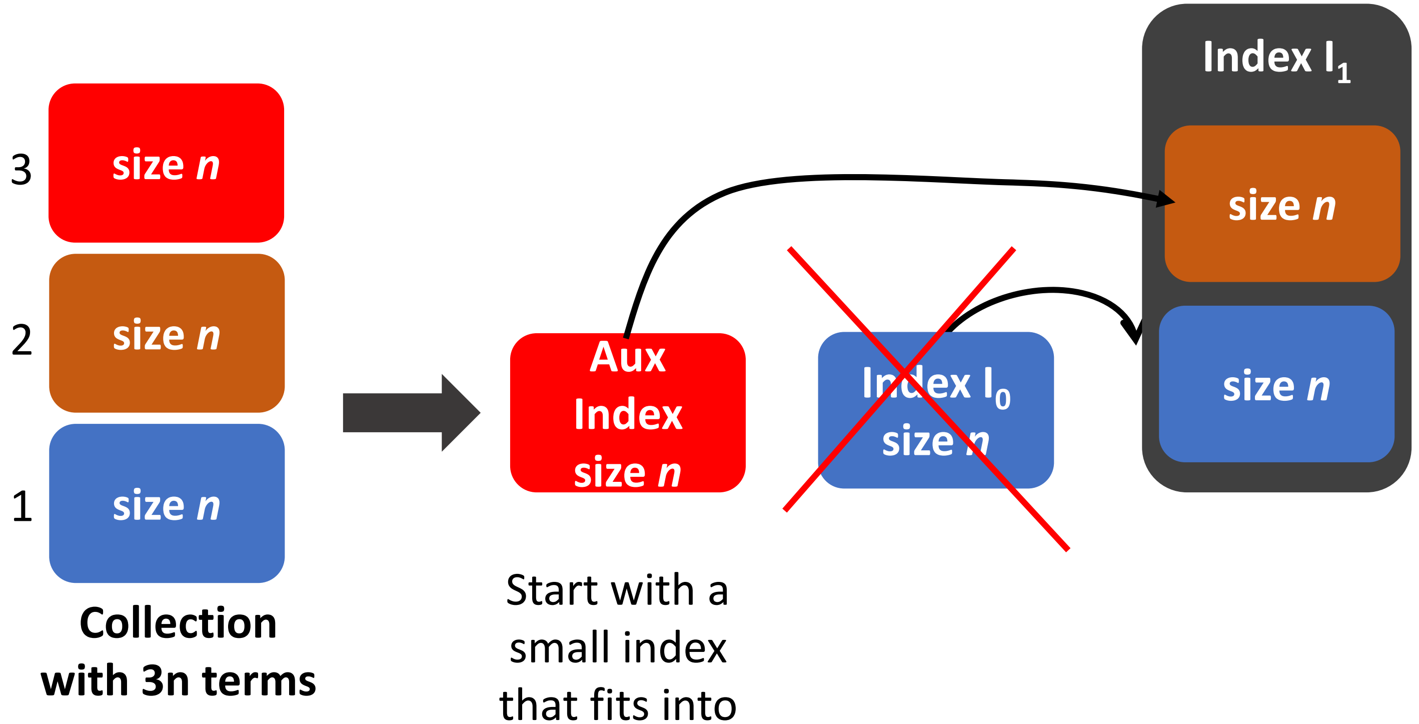
Collection with n terms

Start with a small index that fits into main memory.

Logarithmic Merge



Logarithmic Merge



Start with a small index that fits into main memory.

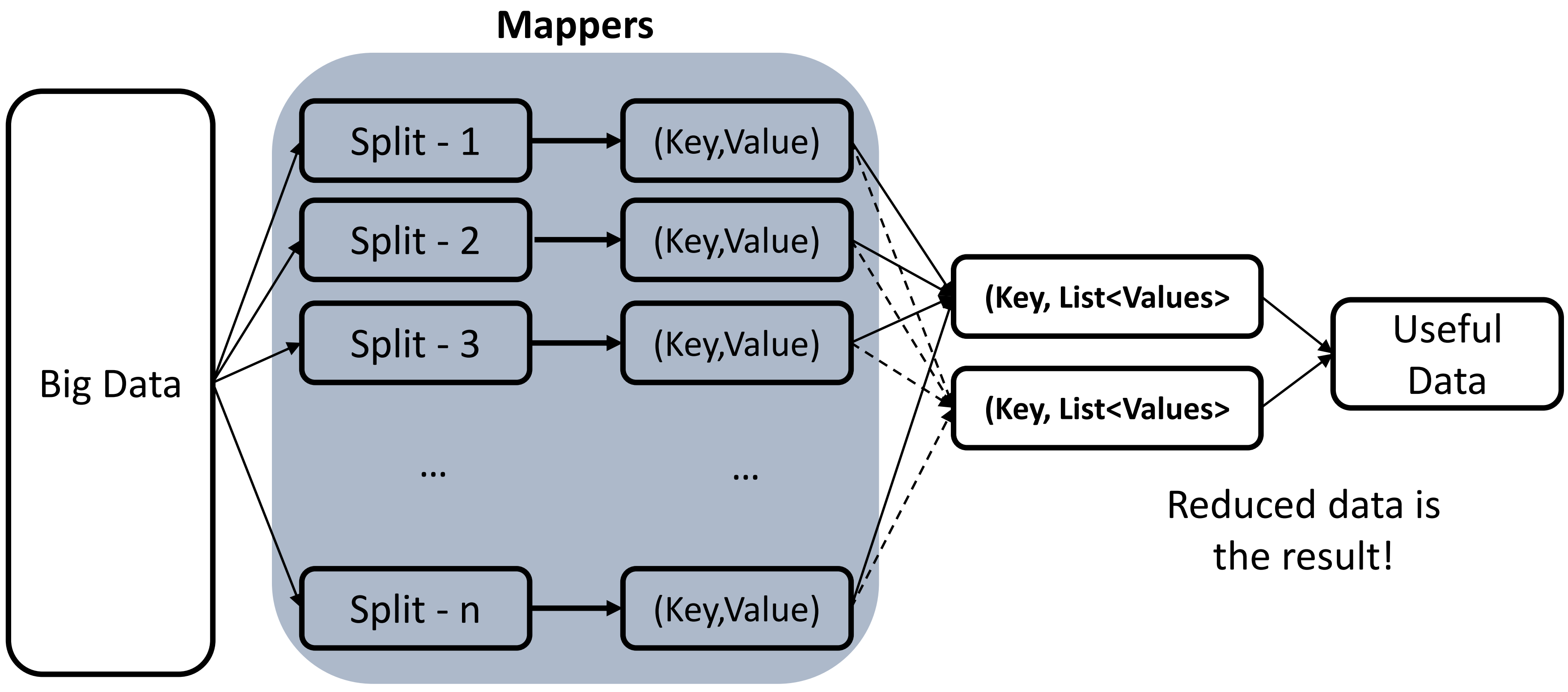
Quiz

- What is the overall index construction time? (choose the best answer)
  - $\theta(1)$
  - $\theta(T/n)$
  - $\theta(T^{2/n})$
  - $\theta(\log(T/n))$
  - $\theta(T \log(T/n))$

Distributed Indexing

How to leverage cloud computing infrastructure and distributed computing frameworks for indexing?

Indexing with the Map-Reduce Framework



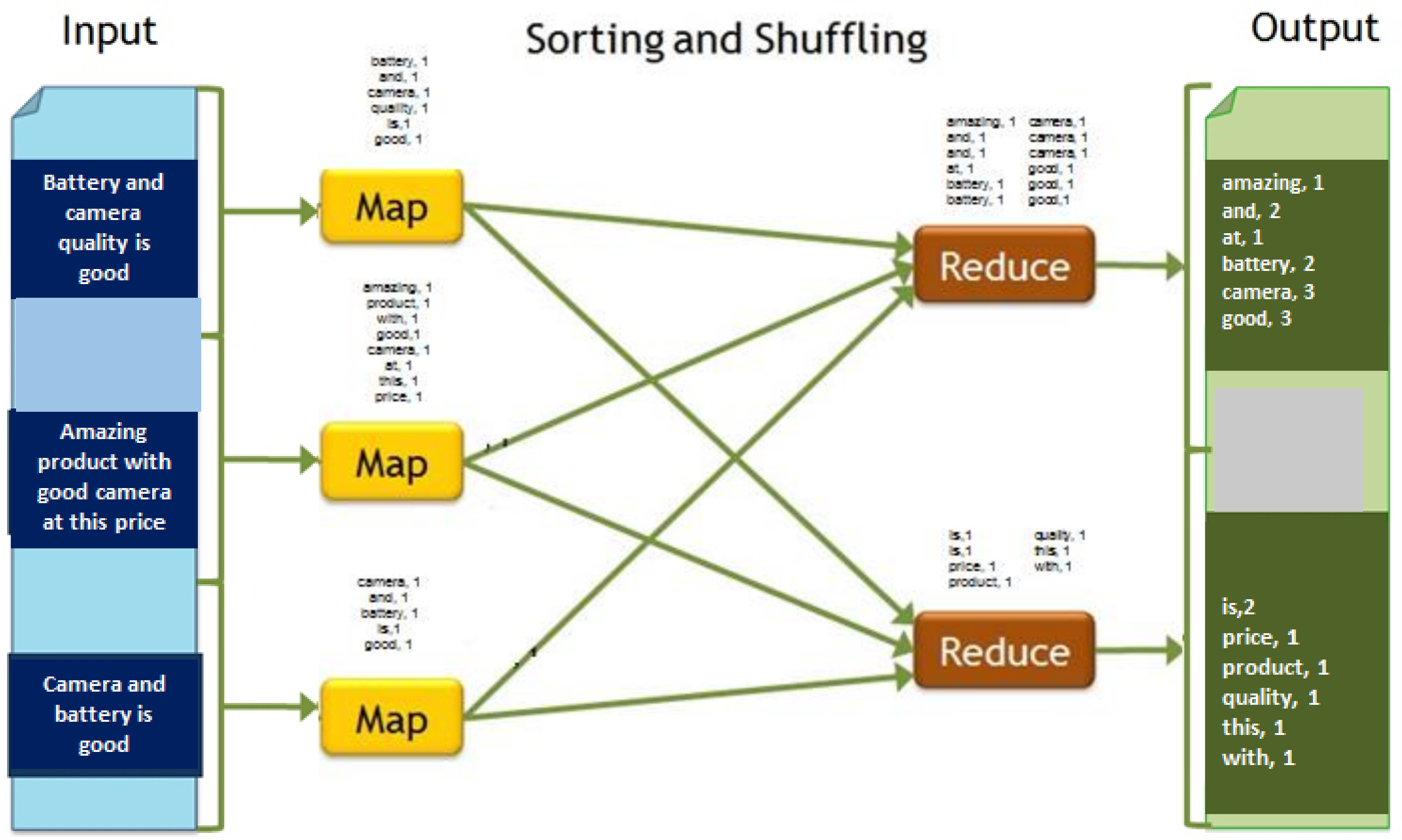
Reduced data is the result!

Input splits sent to Mappers

Mappers process the input

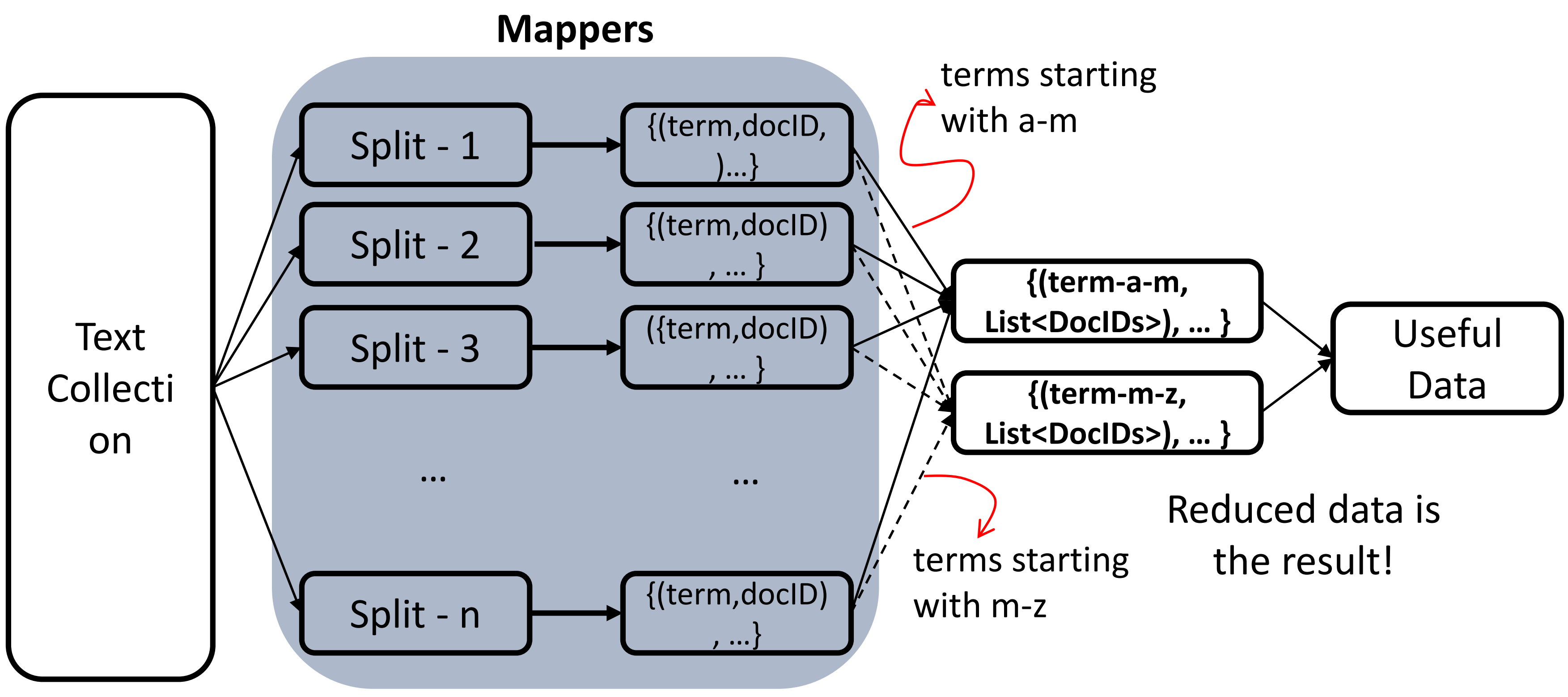
Aggregate the results in reducers

Map-Reduce Framework



Picture Source: Internet.

Indexing with the Map-Reduce Framework

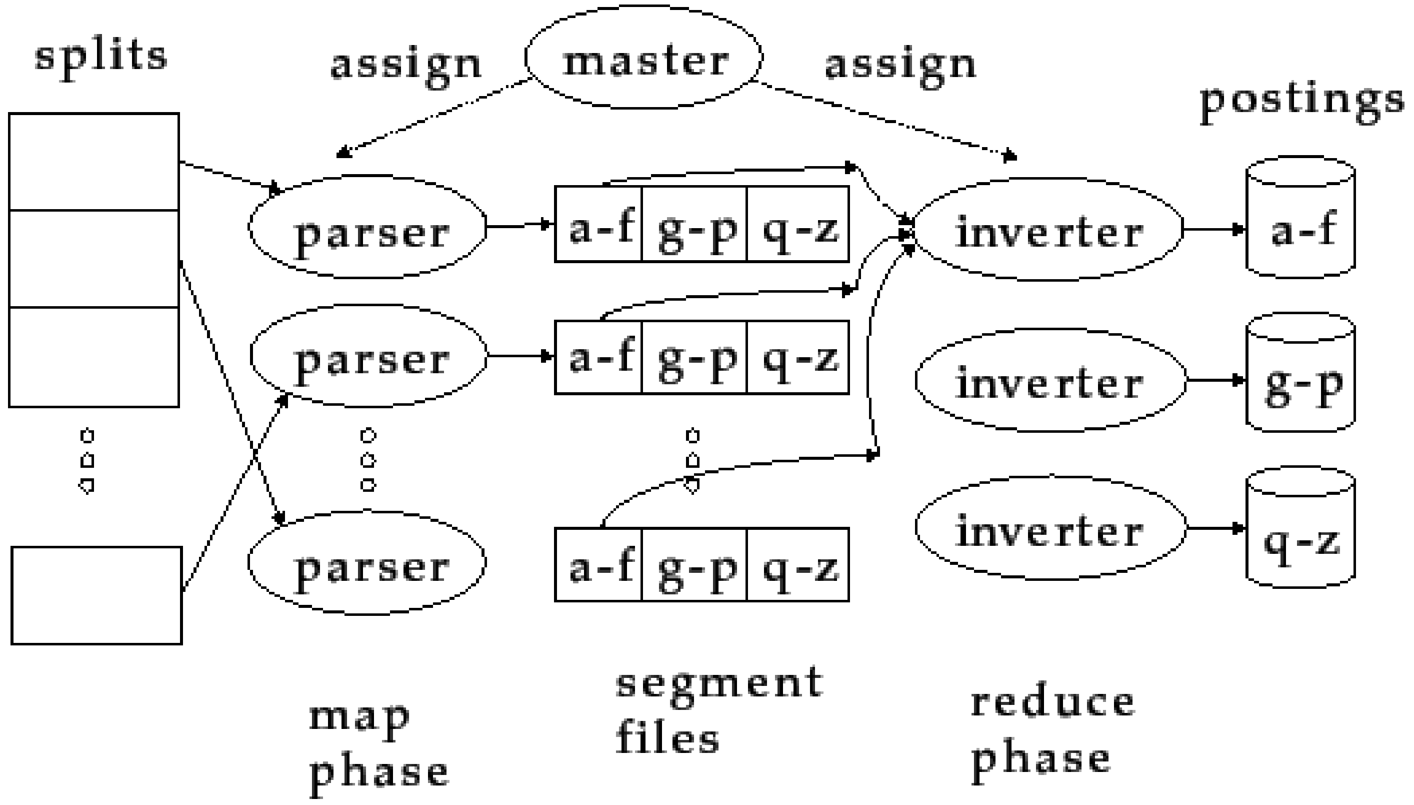


Input splits sent to Mappers

Mappers process the input

Aggregate the results in reducers

Indexing


splits
assign
master
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
map
phase
segment
files
reduce
phase

Summary

- We discussed
  - Blocked Sort-based Indexing
  - Single-pass In-memory Indexing
  - Distributed Indexing
  - Dynamic Indexing
- Advances in hardware, networking and software capabilities have led to wide variety of techniques for efficient indexing.

Information Retrieval

Venkatesh Vinayakarao

Term: Aug – Sep, 2019
Chennai Mathematical Institute

For me, data compression is more than a manipulation of numbers; it is the process of discovering structures that exist in the data.
– Khalid Sayood, University of Nebraska.

Agenda

- Statistic properties of terms
    - The rule of 30
    - Heap’s Law
    - Zipf’s Law
- Index Compression
    - Compressing Dictionaries
    - Compressing Postings

Index Compression

The Rule of 30

The 30 most common words account for 30% of the tokens in written text.

Add a, an, the, … to the stop words list.

Quiz

Given a collection (of documents), how to estimate the number of terms?

One (bad) Approach

- Use Oxford Dictionary
  - Oxford English Dictionary defines 600K+ words.
- The Problem
  - Does not contain people, places, products which users may query.

Heap’s Law

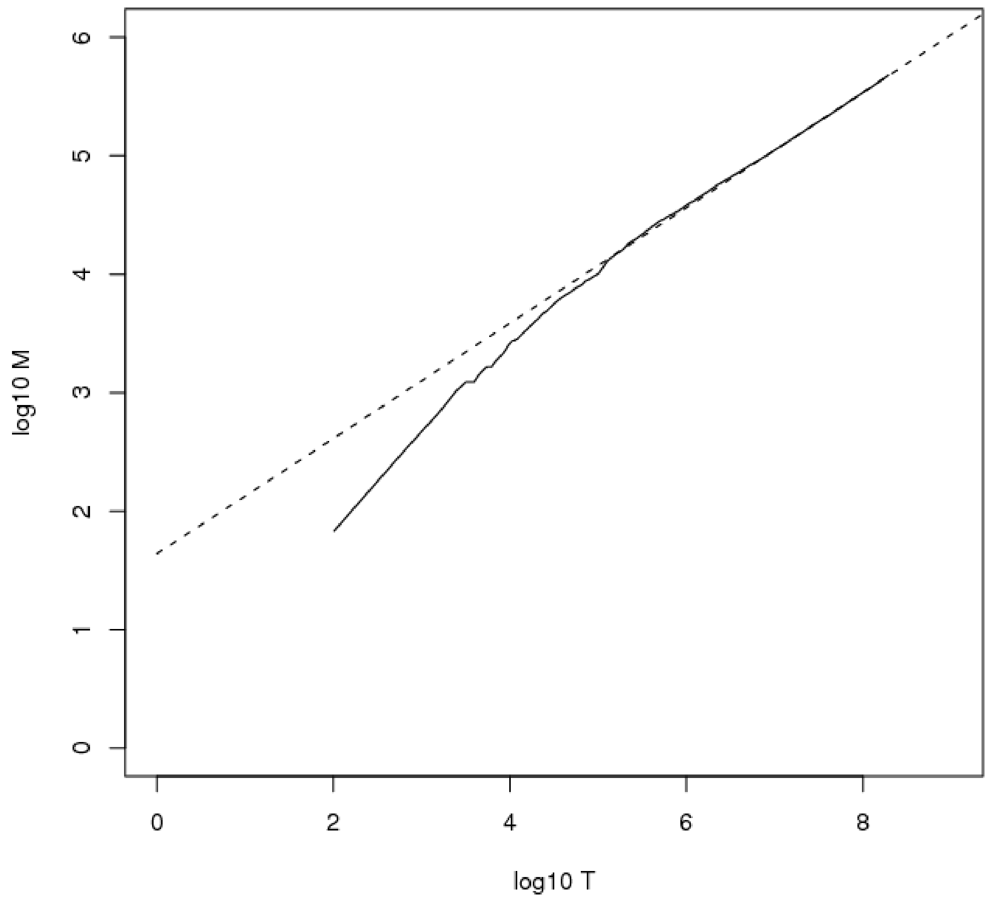
- An Empirical Finding (from experiments on several datasets)

$$M = kT^b$$

- *M* is the size of the vocabulary; *T* is the number of tokens in the collection

Heap’s Law

- $\log_{10} M = 0.49 \log_{10} T + 1.64$ is the best least squares fit for RCV1.
- So $k = 10^{1.64} \approx 44$ and $b = 0.49$.
- For first 1,000,020 tokens, law predicts 38,323 terms.
- Actually, we have 38,365 terms.



Takeaways from Heap's Law

- Dictionary Size grows with collection size.
- Size of dictionary can get "really" large!

Term Frequency

- What are top-3 frequent terms in the text given below? Give the frequency of those terms.

Being an excellent student has more benefits than just getting good grades. In the short term, it will make you a more appealing college candidate and, in many cases, can earn you some fairly hefty scholarships. Big picture, the skills you learn at school will stick with you for the rest of your life, helping you tackle any problem that comes your way.

Repeating Terms

- Give me the term frequency for each repeating term.

Being an excellent student has more benefits than just getting good grades. In the short term, it will make you a more appealing college candidate and, in many cases, can earn you some fairly hefty scholarships. Big picture, the skills you learn at school will stick with you for the rest of your life, helping you tackle any problem that comes your way.

Repeating Tokens are Highlighted

- you = 5, the = 3, in = 2, your = 2 (case-folded)

You, being an excellent student have more benefits than just getting good grades. In the short term, it will make you a more appealing college candidate and, in many cases, can earn you some fairly hefty scholarships. Big picture, the skills you learn at school will stick with you for the rest of your life, helping you tackle any problem that comes your way.

Zipf's Law

The *i*th most frequent term has frequency proportional to

$$\frac{1}{i}$$

Do not apply on small examples…

Being an excellent student has more benefits than just getting good grades. In the short term, it will make you a more appealing college candidate and, in many cases, can earn you some fairly hefty scholarships. Big picture, the skills you learn at school will stick with you for the rest of your life, helping you tackle any problem that comes your way.

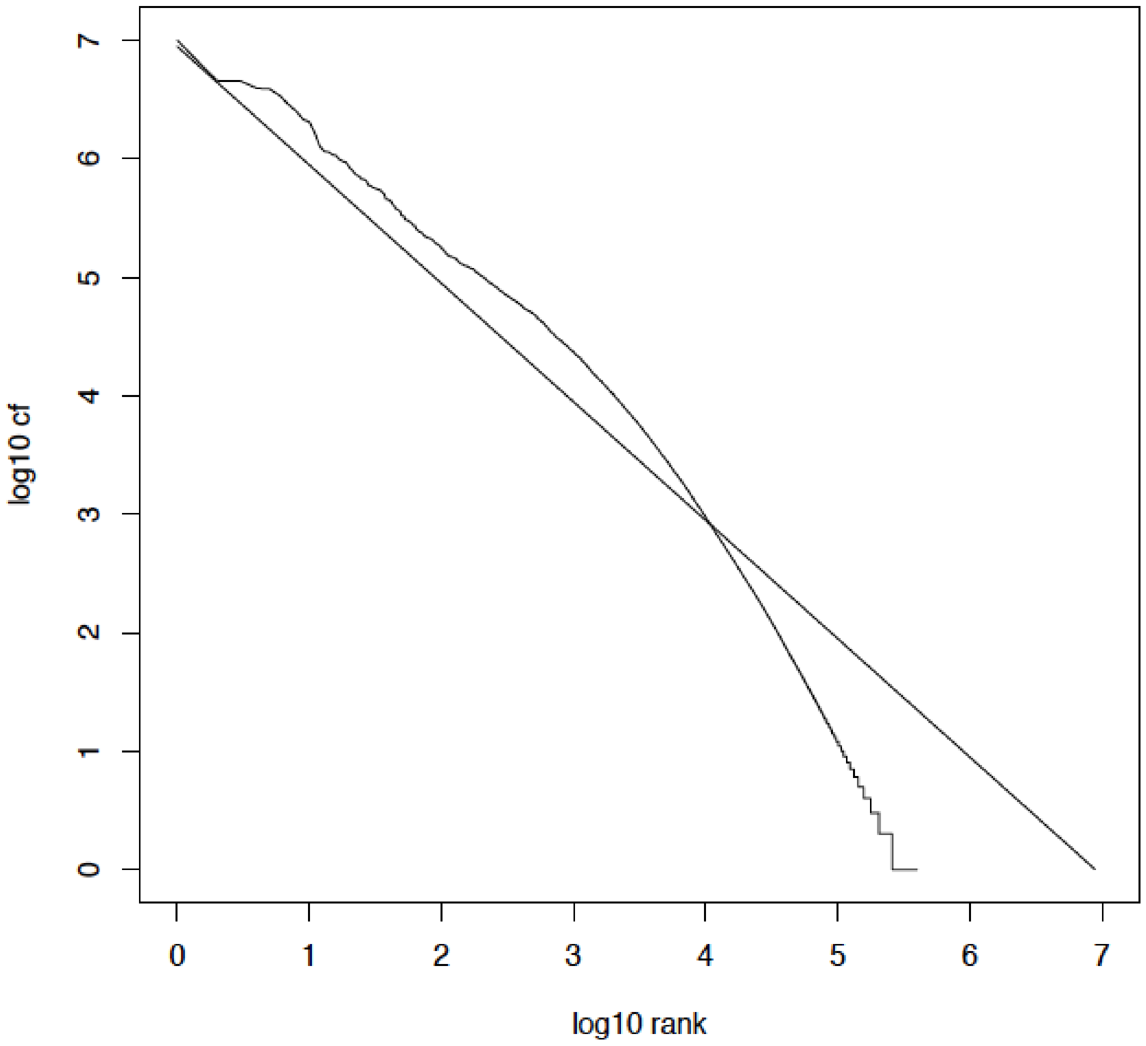
Only for illustration.

Rank	Frequency
1 (you)	6
2 (the)	3
3 (in)	2

$cf_i \propto 1/i = k/i$

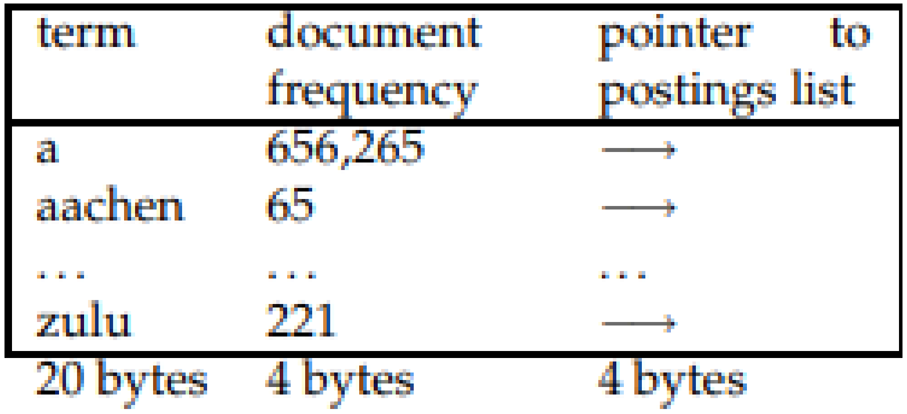
$k = 6$ *in our case!*

Zipf’s law for Reuters RCV1



Compressing Dictionaries

Dictionary as a Sorted Array

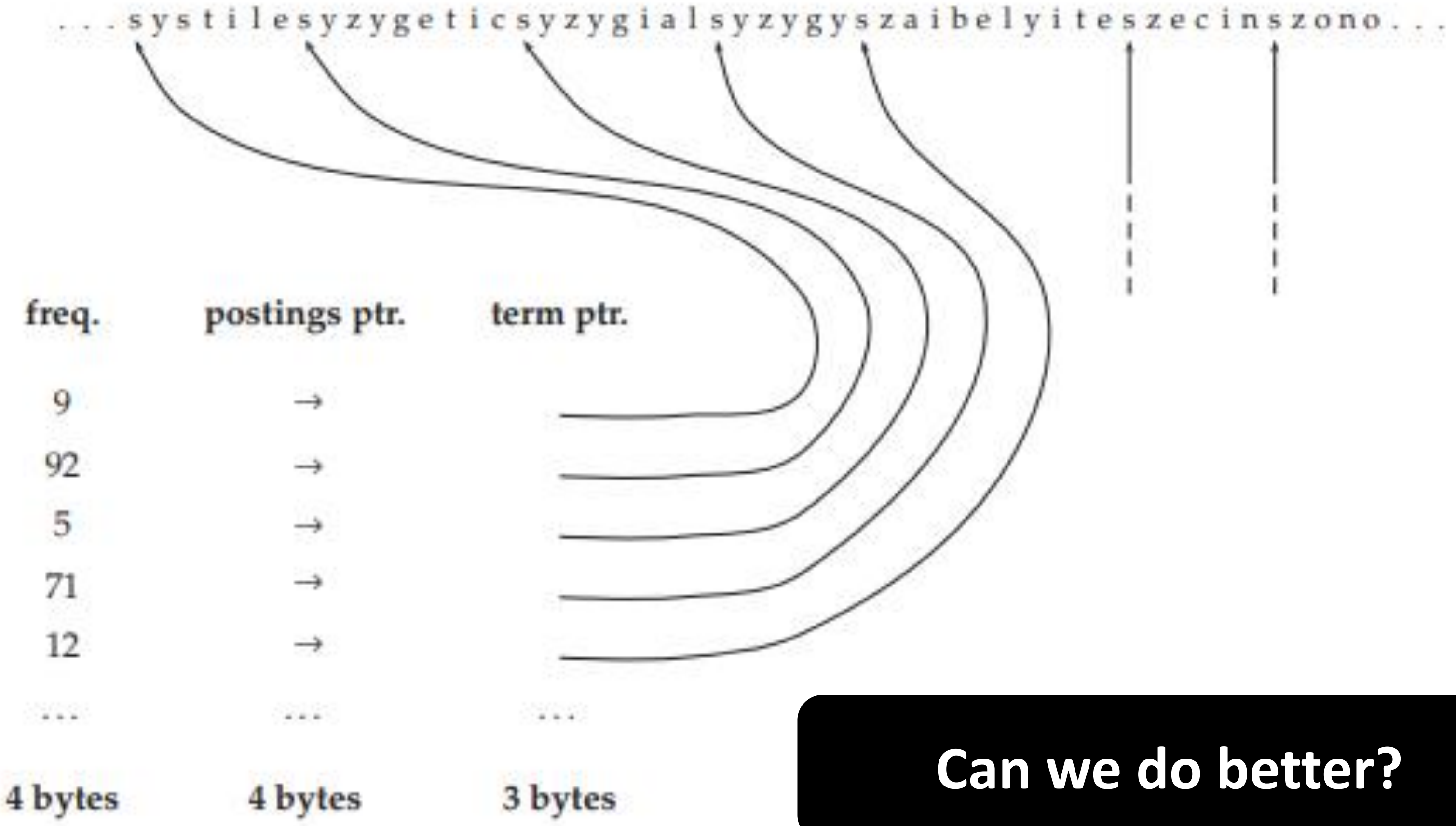


term	document frequency	pointer to postings list
a	656,265	$\longrightarrow$
aachen	65	$\longrightarrow$
...	...	...
zulu	221	$\longrightarrow$
20 bytes	4 bytes	4 bytes

Apply binary search to search the term array!

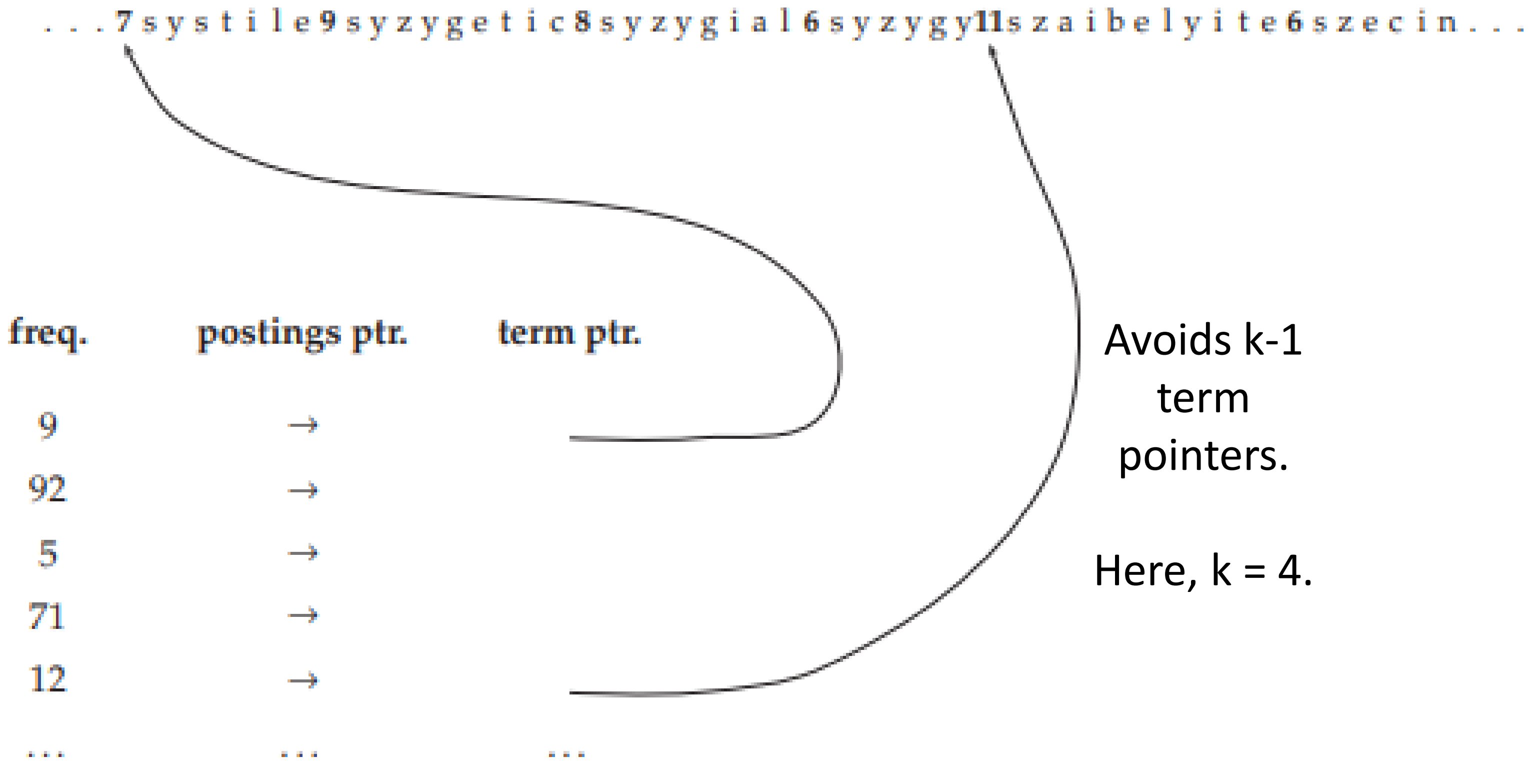
Can we do better?

Dictionary as a String



Can we do better?

Blocked Storage



. . . 7 s y s t i l e 9 s y z y g e t i c 8 s y z y g i a l 6 s y z y g y 11 s z a i b e l y i t e 6 s z e c i n . . .
freq.
postings ptr.
term ptr.
9
92
5
71
12
. . .
. . .
. . .

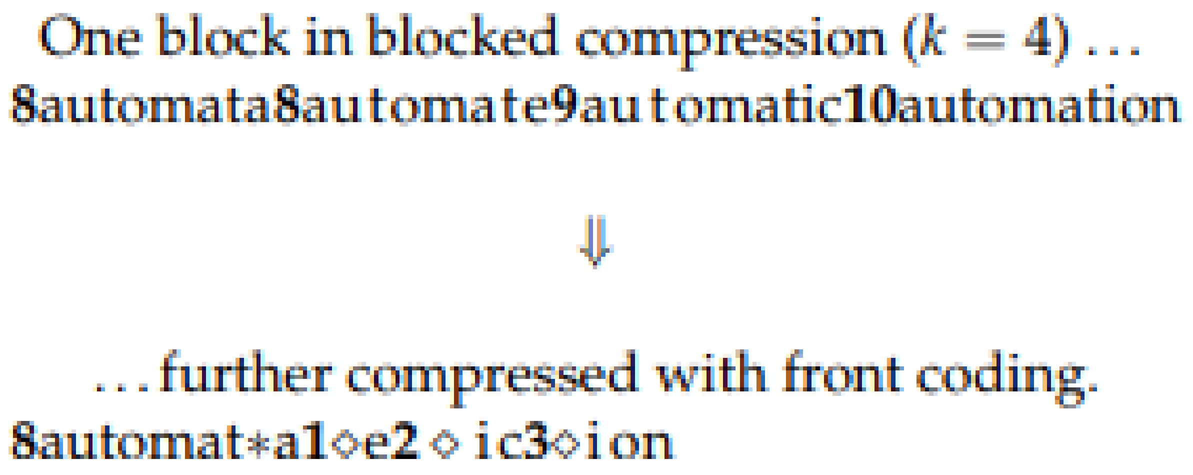
Avoids k-1 term pointers.

Here, k = 4.

Can we do better?

Clue: Consecutive entries in an alphabetically sorted list (Dictionary) share common prefixes!

Front Coding



One block in blocked compression ($k = 4$) ...

8automata8automate9automatic10automation

⇓

...further compressed with front coding.

8automat∗a1◇e2◇ic3◇ion

Can you front-code at k = 3, "interspecies","interstellar","interstate"?

12inter*species7◊stellar5◊state

or

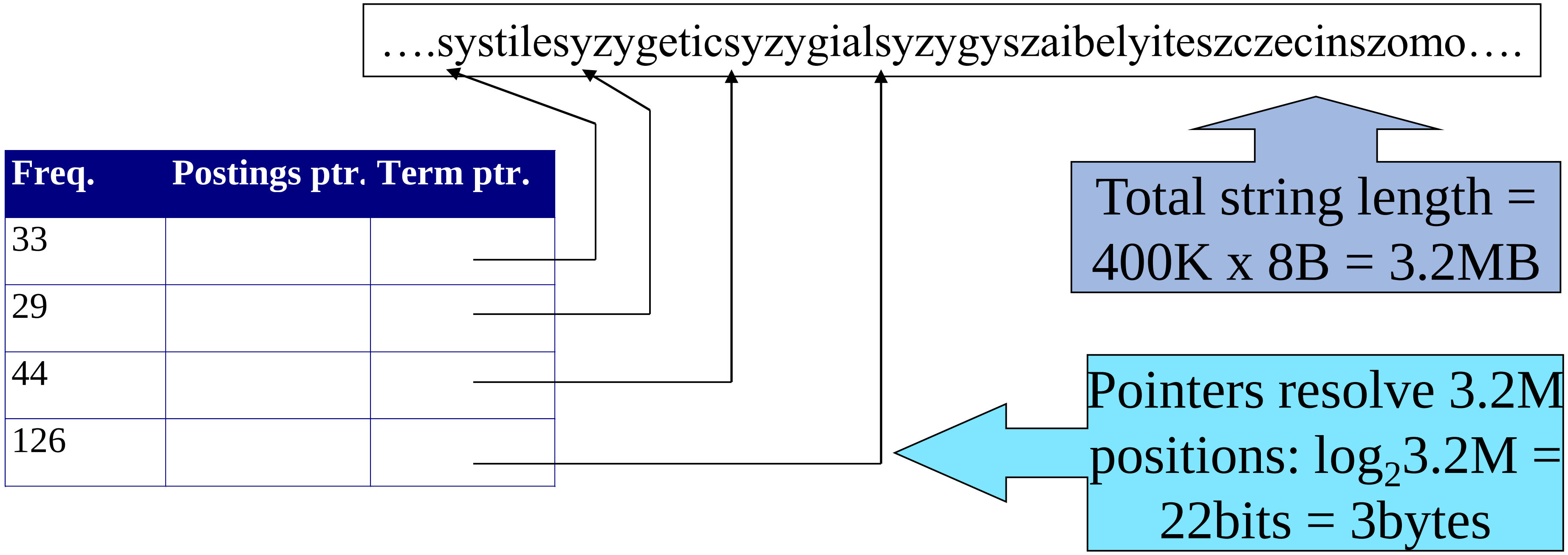
12inters*pecies6◊tellar4◊tate

Right Answer

- **12inters*pecies6◊tellar4◊tate**

Compressing the term list: Dictionary-as-a-String

- Store dictionary as a (long) string of characters:
  - Pointer to next word shows end of current word
  - Hope to save up to 60% of dictionary space.



….systilesyzygeticsyzygialsyzygyszaibelyiteszczecinszomo….

Freq.	Postings ptr.	Term ptr.
33		
29		
44		
126		

Total string length = 400K x 8B = 3.2MB

Pointers resolve 3.2M positions: $\log_2 3.2M$ = 22bits = 3bytes

Compressing Postings

How to store a large number of “numbers” efficiently?

Store Gaps

33	**47**	**154**	**159**	**202**

$$\Big\downarrow$$

33	**14**	**107**	**5**	**43**

Can we do better?

Clue: Smaller numbers can be represented using fewer bits.

Variable Byte Encoding

824	829	215406	…	…	
824	5	214577	…	…	Gaps
<6, 56>	<5>	<13, 12, 49>	…	…	VBEncode

824 = 2^7 * 6 + 56. **So, we use <56, 6> to represent it.**
5 = 2^7 * 0 + 5. **So, we use <5> to represent it.**
214577 = 2^7 * ((2^7 * 13) + 12) + 49. So, <49,12,13>.

00000110 10111000 10000101 0001101 00001100 10110001 …	Encoded Bytestream

How to decode the bytestream?

$\underline{0}$0000110 $\underline{1}$0111000	$\underline{1}$0000101	$\underline{0}$001101 $\underline{0}$0001100 $\underline{1}$0110001	…	…

Continuation bits are underlined.

Another Dig at 214577

- What is the VB encoding for 214577?

n	$\lfloor n/128 \rfloor$	n % 128
214577	1676	49
1676	13	12
13	0	13

- So, 214577 = <13, 12, 49> which means ((13*128)+12)*128 + 49.
- Thus we have, 214577 represented as 0001101 00001100 10110001 in VB encoded bytestream.

Quiz

- Compute the variable byte codes for the postings list (100, 200, 400, 800)

Quiz

- Compute the variable byte codes for the postings list (100, 200, 400, 800)

Postings	100	200	400	800
Gaps	100	100	200	400

Decomposing Gaps

n	$\lfloor n/128 \rfloor$	$n \% 128$
100	0	100

n	$\lfloor n/128 \rfloor$	$n \% 128$	n	$\lfloor n/128 \rfloor$	$n \% 128$
200	1	72	400	3	16
1	0	1	3	0	3

VB Encoding	<100>	<100>	<1, 72>	<3, 16>

Result 11100100 11100100 00000001 11001000 000000011 10010000

Quiz

- Compute the variable byte codes for the postings list (100, 200, 400, 800)
- Answer: 11100100 11100100 00000001 11001000 000000011 10010000

Questions