Query processing: phrase queries and positional indexes

Paolo Ferragina
Dipartimento di Informatica
Università di Pisa

Phrase queries

- Want to be able to answer queries such as "***stanford university***" – as a phrase
- Thus the sentence *"I went at Stanford my university"* is not a match.

Solution #1: 2-word indexes

- For example the text “Friends, Romans, Countrymen” would generate the biwords
  - ***friends romans***
  - ***romans countrymen***

- Each of these **2-words** is now an entry in the dictionary

- Two-word phrase query-processing is immediate.

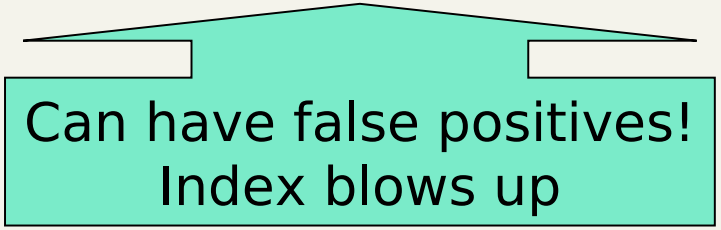
Longer phrase queries

- Longer phrases are processed by reducing them to bi-word queries in AND

stanford university palo alto can be broken into the Boolean query on biwords, such as

stanford university *AND* ***university palo*** *AND* ***palo alto***

Need the docs to verify
+
They are combined with other solutions



Solution #2: Positional indexes

- In the postings, store for each ***term*** and ***document*** the position(s) in which that term occurs:

  <***term***, number of docs containing ***term***;
  doc1: position1, position2 … ;
  doc2: position1, position2 … ;
  etc.>

Processing a phrase query

- "***to be or not to be***".
  - ***to***:
    - *2*:1,17,74,222,551; *4*:8,16,190,429,433; *7*:13,23,191; ...
  - ***be***:
    - *1*:17,19; *4*:17,191,291,430,434; *5*:14,19,101; ...

- Same general method for proximity searches

Query term proximity

- Free text queries: just a set of terms typed into the query box – common on the web
- Users prefer docs in which query terms occur within close proximity of each other
- Would like scoring function to take this into account – how?

Positional index size

- You can compress position values/offsets
- Nevertheless, a positional index expands postings storage *by a factor 2-4 in English*

- Nevertheless, a positional index is now commonly used because of the power and usefulness of phrase and proximity queries … whether used explicitly or implicitly in a ranking retrieval system.

Combination schemes

- 2-Word + Positional index is a profitable combination
  - 2-word is particularly useful for particular phrases (***“Michael Jackson”, “Britney Spears”***)
- More complicated mixing strategies do exist!

Soft-AND

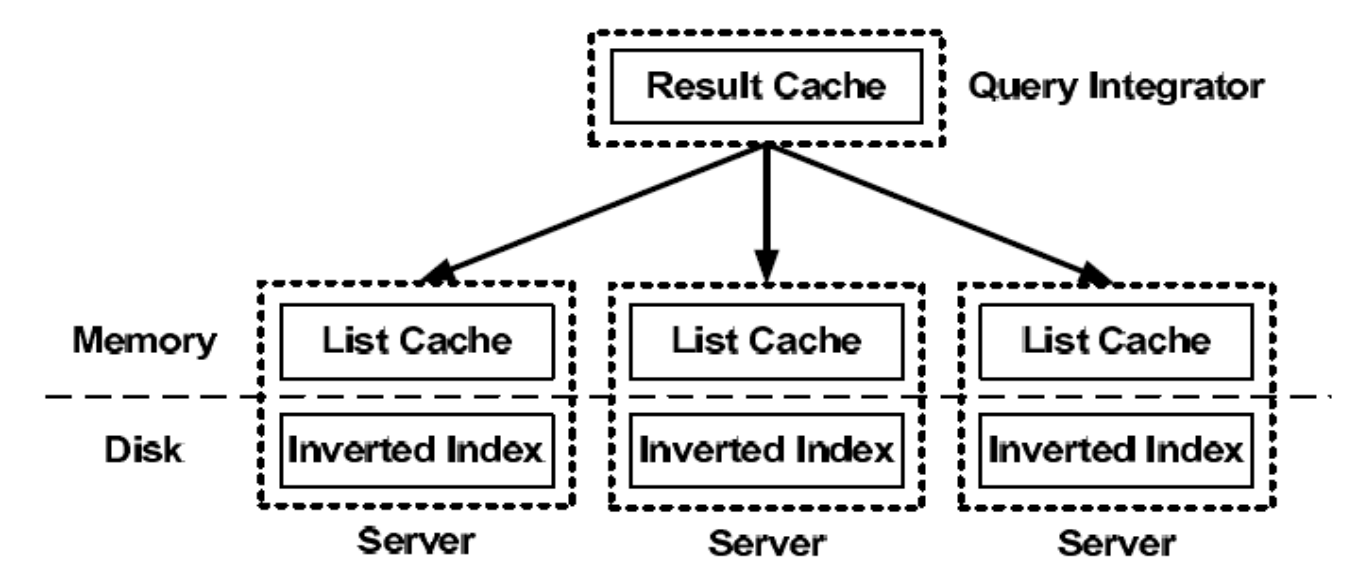
- E.g. query *rising interest rates*
  - Run the query as a phrase query
  - If <*K* docs contain the phrase *rising interest rates*, run the two phrase queries *rising interest* and *interest rates*
  - If we still have <*K* docs, run the "vector space query" *rising interest rates* *(…see next…)*
  - "Rank" the matching docs *(…see next…)*

Caching for faster query

Two opposite approaches:

I. Cache the query results (exploits query locality)

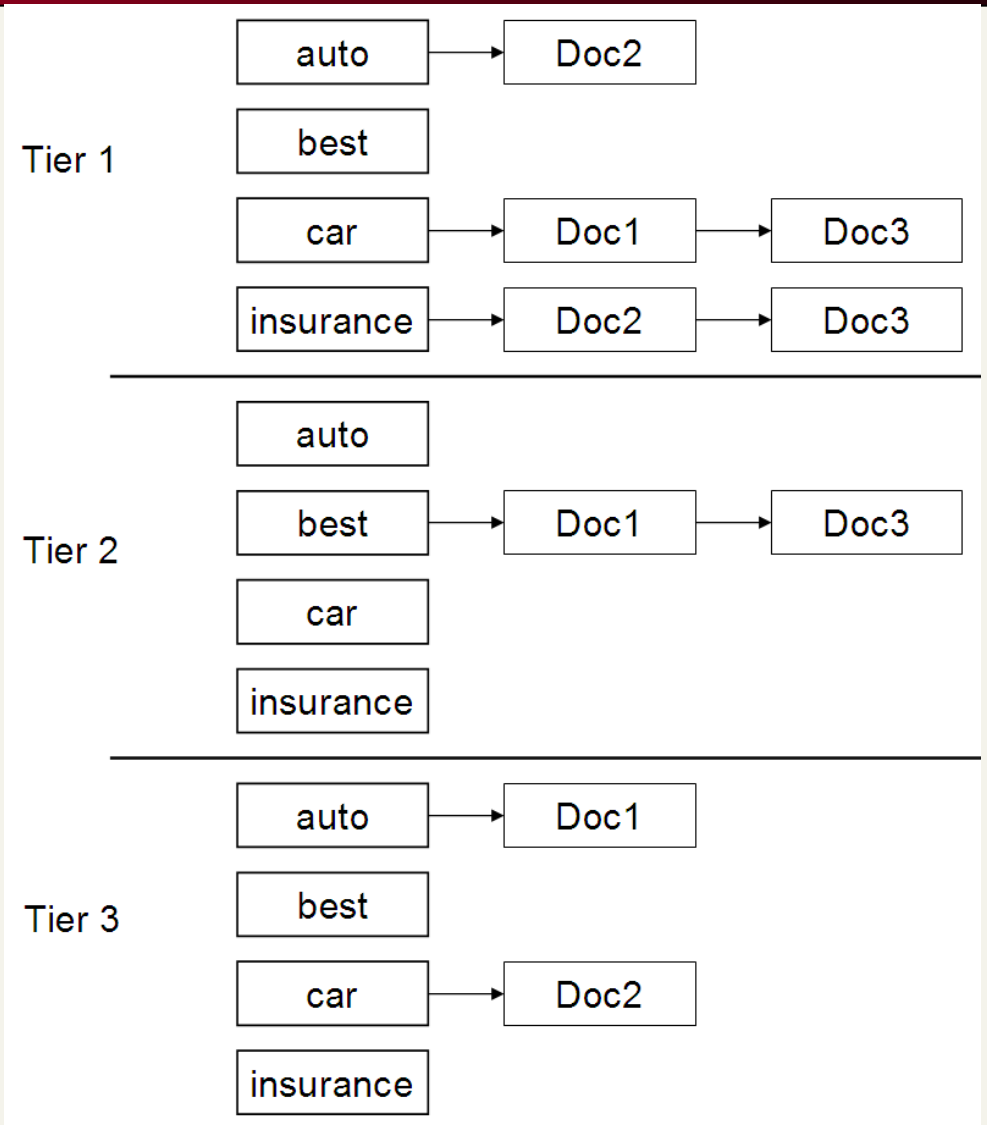
II. Cache pages of posting lists (exploits term locality)



Tiered indexes for faster query

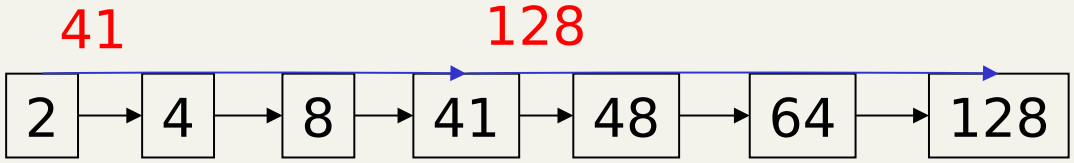
- Break postings up into a hierarchy of lists
  - Most important
  - …
  - Least important
- Inverted index thus broken up into tiers of decreasing importance
- At query time use top tier unless it fails to yield *K* docs
  - If so drop to lower tiers

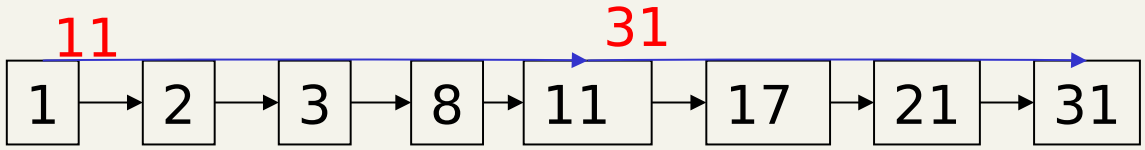
Example tiered index



Query processing: optimizations

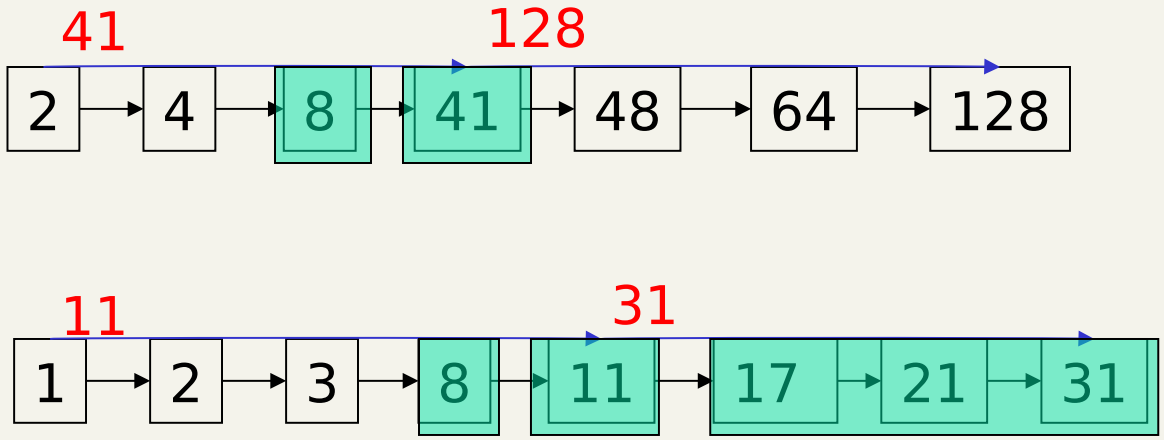
Skip pointers (at indexing time)





- How do we deploy them ?
- Where do we place them ?

Using skips



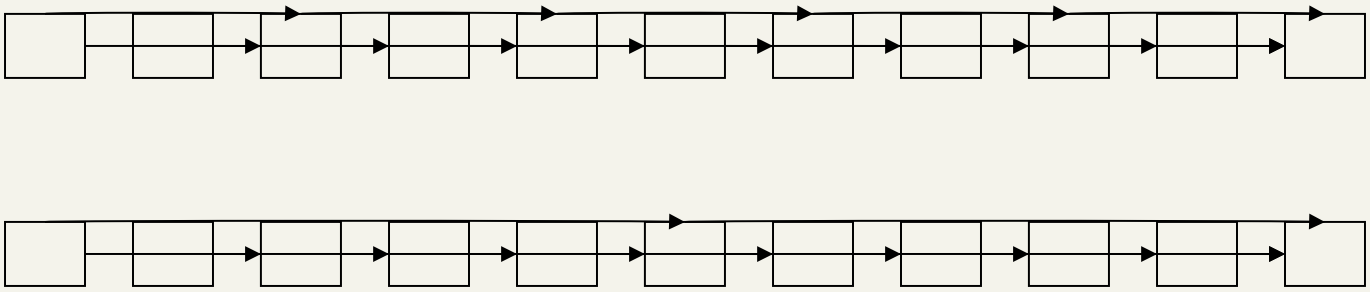
Suppose we've stepped through the lists until we process **8** on each list. We match it and advance.

We then have **41** and **11** on the lower. **11** is smaller.

But the skip successor of **11** on the lower list is **31**, so we can skip ahead past the intervening postings.

Placing skips

- Tradeoff:
  - More skips → shorter spans ⇒ more likely to skip. But lots of comparisons to skip pointers.
  - Fewer skips → longer spans ⇒ few successful skips. Less pointer comparisons.



Placing skips

- Simple heuristic for postings of length L
  - use $\sqrt{L}$ evenly-spaced skip pointers.
  - This ignores the distribution of query terms.
  - Easy if the index is relatively static.

- This definitely useful for in-memory index
  - The I/O cost of loading a bigger list can outweigh the gains!