# Informatics 225
# Computer Science 221

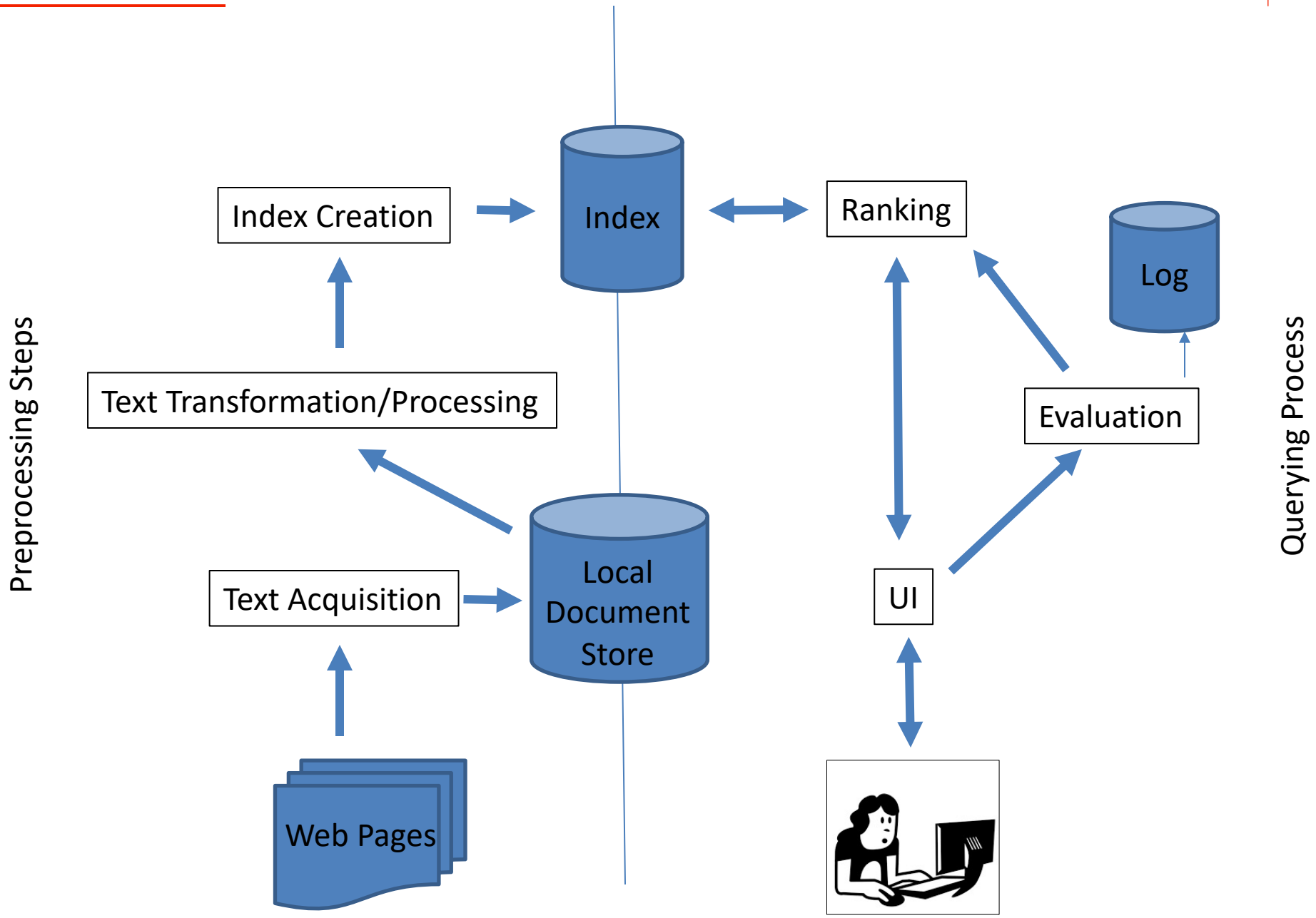
# Information Retrieval

Lecture 23



*These course materials borrow, with permission, from those of Prof. Cristina Videira Lopes, Addison Wesley 2008, Chris Manning, Pandu Nayak, Hinrich Schütze, Heike Adel, Sascha Rothe, Jerome H. Friedman, Robert Tibshirani, and Trevor Hastie. Powerpoint theme by Prof. André van der Hoek.*

# Architecture



# Generic approach

- Find a set *A* of *contenders*, with $K < |A| << N$
  - *A* does not necessarily contain the top *K,*
  - but has many docs from among the top *K*
  - Return the top *K* docs in *A*

- Think of *A* as pruning non-contenders

- The same approach is also used for other (non-cosine) scoring functions

- Will look at a few schemes following this approach

# Index elimination

- Basic algorithm for cosine computation only considers docs containing at least one query term

- Take this heuristic further:
  - Only consider high-idf query terms
  - Only consider docs containing many query terms

# Champion lists

- Precompute for each dictionary term *t,* the *r* docs of highest weight in *t*'s postings

# Champion lists

- Precompute for each dictionary term *t,* the *r* docs of highest weight in *t*'s postings
  - Call this the champion list for *t*
  - (aka fancy list or top docs for *t*)

# Champion lists

- Precompute for each dictionary term *t,* the *r* docs of highest weight in *t*'s postings
  - Call this the champion list for *t*
  - (aka fancy list or top docs for *t*)

- Note that *r* has to be chosen at index build time
  - Thus, it's possible that $r < K$

# Champion lists

- Precompute for each dictionary term *t,* the *r* docs of highest weight in *t*'s postings
  - Call this the champion list for *t*
  - (aka fancy list or top docs for *t*)
- Note that *r* has to be chosen at index build time
  - Thus, it's possible that $r < K$
- At query time, only compute the complete scores for docs in the champion list of some query term

# Champion lists

- Precompute for each dictionary term *t,* the *r* docs of highest weight in *t*'s postings
  - Call this the champion list for *t*
  - (aka fancy list or top docs for *t*)

- Note that *r* has to be chosen at index build time
  - Thus, it's possible that $r < K$

- At query time, only compute the complete scores for docs in the champion list of some query term
  - Pick the *K* top-scoring docs from amongst these

# Static quality scores

- We want top-ranking documents to be both *relevant* and *authoritative*

# Static quality scores

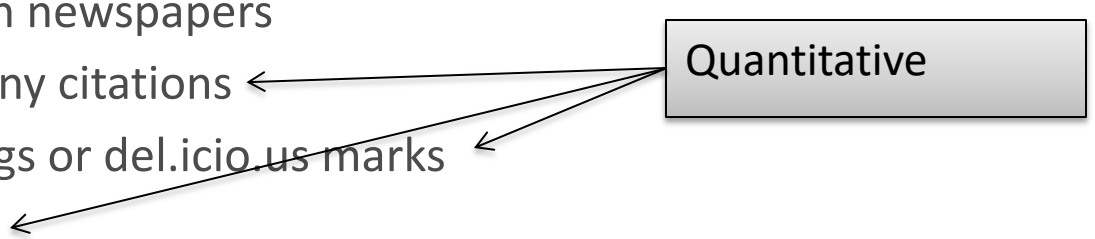
- We want top-ranking documents to be both *relevant* and *authoritative*
- *Relevance* is being modeled by cosine scores

# Static quality scores

- We want top-ranking documents to be both *relevant* and *authoritative*
- *Relevance* is being modeled by cosine scores
- *Authority* is typically a query-independent property of a document

# Static quality scores

- We want top-ranking documents to be both *relevant* and *authoritative*
- *Relevance* is being modeled by cosine scores
- *Authority* is typically a query-independent property of a document
- Examples of authority signals
  - Wikipedia among websites
  - Articles in certain newspapers
  - A paper with many citations
  - Many bitly's, diggs or del.icio.us marks
  - (Pagerank, HITS)



# Modeling authority

- Assign to each document a *query-independent* quality score in [0,1] to each document *d*
  - Denote this by *g(d)*

# Modeling authority

- Assign to each document a *query-independent* quality score in [0,1] to each document *d*
  - Denote this by *g(d)*

- Thus, a quantity like the number of citations is scaled into [0,1]
  - Informal exercise: suggest a formula for this.

# Net relevance score

- Consider a simple total score combining cosine relevance and authority

# Net relevance score

- Consider a simple total score combining cosine relevance and authority

- net-score(*q,d*) = *g(d) +* cosine(*q,d*)
  - Can use some other linear combination: g(d) + α*cosine(q,d)
  - Indeed, any function of the two "signals" of user happiness

# Net relevance score

- Consider a simple total score combining cosine relevance and authority

- net-score(*q,d*) = *g(d) +* cosine(*q,d*)
  - Can use some other linear combination: g(d) + α*cosine(q,d)
  - Indeed, any function of the two "signals" of user happiness

- Now we seek the top *K* docs by net score

# Top *K* by net score – fast methods

- First idea: Order all postings by *g(d)*

# Top *K* by net score – fast methods

- First idea: Order all postings by *g(d)*
- Key: this is a common ordering for all postings

# Top *K* by net score – fast methods

- First idea: Order all postings by *g(d)*
- Key: this is a common ordering for all postings
- Thus, can concurrently traverse query terms' postings for
  - Postings intersection
  - Cosine score computation

# Why order postings by *g(d)*?

- Under *g(d)*-ordering, top-scoring docs likely to appear early in postings traversal (remember : g(d) + α*cosine(q,d))

# Why order postings by *g(d)*?

- Under *g(d)*-ordering, top-scoring docs likely to appear early in postings traversal

- In time-bound applications (say, we have to return whatever search results we can in ~50 ms), this allows us to stop postings traversal early
  - Short of computing complete scores for all docs in postings

# Champion lists in *g(d)*-ordering

- Can combine champion lists with *g(d)*-ordering

# Champion lists in *g(d)*-ordering

- Can combine champion lists with *g(d)*-ordering
- Maintain for each term a champion list of the *r* docs with highest *g(d) +* tf-idf$_{td}$

# Champion lists in *g(d)*-ordering

- Can combine champion lists with *g(d)*-ordering
- Maintain for each term a champion list of the *r* docs with highest *g(d) +* tf-idf$_{td}$
- Seek top-*K* results from only the docs in these champion lists

# Another heuristic: High and low lists

- For each term, we maintain two disjoint sets: *high* and *low lists*
  - Think of *high* as the champion list

# Another heuristic: High and low lists

- For each term, we maintain two disjoint sets: *high* and *low lists*
  - Think of *high* as the champion list

- When traversing postings on a query, traverse *high* lists first

# Another heuristic: High and low lists

- For each term, we maintain two disjoint sets: *high* and *low lists*
  - Think of *high* as the champion list

- When traversing postings on a query, traverse *high* lists first
  - If we get more than *K* docs, select the top *K* and stop
  - Else proceed to get docs from the *low* lists

# Another heuristic: High and low lists

- For each term, we maintain two disjoint sets: *high* and *low lists*
  - Think of *high* as the champion list

- When traversing postings on a query, traverse *high* lists first
  - If we get more than *K* docs, select the top *K* and stop
  - Else proceed to get docs from the *low* lists

- Can be used even for simple cosine scores, without global quality *g(d)*

# Another heuristic: High and low lists

- For each term, we maintain two disjoint sets: *high* and *low lists*
  - Think of *high* as the champion list

- When traversing postings on a query, traverse *high* lists first
  - If we get more than *K* docs, select the top *K* and stop
  - Else proceed to get docs from the *low* lists

- Can be used even for simple cosine scores, without global quality *g(d)*

- A means for segmenting index into two tiers

# Impact-ordered postings

- We only want to compute scores for docs for which $wf_{t,d}$ is high enough

# Impact-ordered postings

- We only want to compute scores for docs for which $wf_{t,d}$ is high enough
- We sort each postings list by $wf_{t,d}$

# Impact-ordered postings

- We only want to compute scores for docs for which $wf_{t,d}$ is high enough
- We sort each postings list by $wf_{t,d}$
- Now: not all postings in a common order!

# Impact-ordered postings

- We only want to compute scores for docs for which $wf_{t,d}$ is high enough
- We sort each postings list by $wf_{t,d}$
- Now: not all postings in a common order!
- How do we compute scores in order to pick off top *K?*
  - Two ideas follow

# Idea 1. Early termination

- When traversing *t*'s postings, stop early after either
  - a fixed number of *r* docs
  - $wf_{t,d}$ drops below some threshold

# Idea 1. Early termination

- When traversing *t*'s postings, stop early after either
  - a fixed number of *r* docs
  - $wf_{t,d}$ drops below some threshold

- Take the union of the resulting sets of docs
  - One from the postings of each query term

# Idea 1. Early termination

- When traversing *t*'s postings, stop early after either
  - a fixed number of *r* docs
  - $wf_{t,d}$ drops below some threshold
- Take the union of the resulting sets of docs
  - One from the postings of each query term
- Compute only the scores for docs in this union

# Idea 2. idf-ordered terms

- When considering the postings of query terms

# Idea 2. idf-ordered terms

- When considering the postings of query terms
- Look at them in order of decreasing idf
  - High idf terms likely to contribute most to score

# Idea 2. idf-ordered terms

- When considering the postings of query terms
- Look at them in order of decreasing idf
  - High idf terms likely to contribute most to score
- As we update score contribution from each query term
  - Stop if doc scores relatively unchanged

# Idea 2. idf-ordered terms

- When considering the postings of query terms
- Look at them in order of decreasing idf
  - High idf terms likely to contribute most to score
- As we update score contribution from each query term
  - Stop if doc scores relatively unchanged
- Can apply to cosine or some other net scores

# Parametric and zone indexes

- Thus far, a doc has been a sequence of terms

# Parametric and zone indexes

- Thus far, a doc has been a sequence of terms
- In fact documents have multiple parts, some with special semantics:
  - Author
  - Title
  - Date of publication
  - Language
  - Format
  - etc.

# Parametric and zone indexes

- Thus far, a doc has been a sequence of terms
- In fact documents have multiple parts, some with special semantics:
  - Author
  - Title
  - Date of publication
  - Language
  - Format
  - etc.
- These constitute the metadata about a document

# Fields

- We sometimes wish to search by these metadata
  - E.g., find docs authored by William Shakespeare in the year 1601, containing *alas poor Yorick*

# Fields

- We sometimes wish to search by these metadata
  - E.g., find docs authored by William Shakespeare in the year 1601, containing *alas poor Yorick*

- Year = 1601 is an example of a field

- Also, author last name = shakespeare, etc.

# Fields

- We sometimes wish to search by these metadata
  - E.g., find docs authored by William Shakespeare in the year 1601, containing *alas poor Yorick*
- Year = 1601 is an example of a field
- Also, author last name = shakespeare, etc.
- Field or parametric index: postings for each field value
  - Sometimes build range trees (e.g., for dates)

# Fields

- We sometimes wish to search by these metadata
  - E.g., find docs authored by William Shakespeare in the year 1601, containing *alas poor Yorick*
- Year = 1601 is an example of a field
- Also, author last name = shakespeare, etc.
- Field or parametric index: postings for each field value
  - Sometimes build range trees (e.g., for dates)
- Field query typically treated as conjunction
  - (doc *must* be authored by shakespeare)

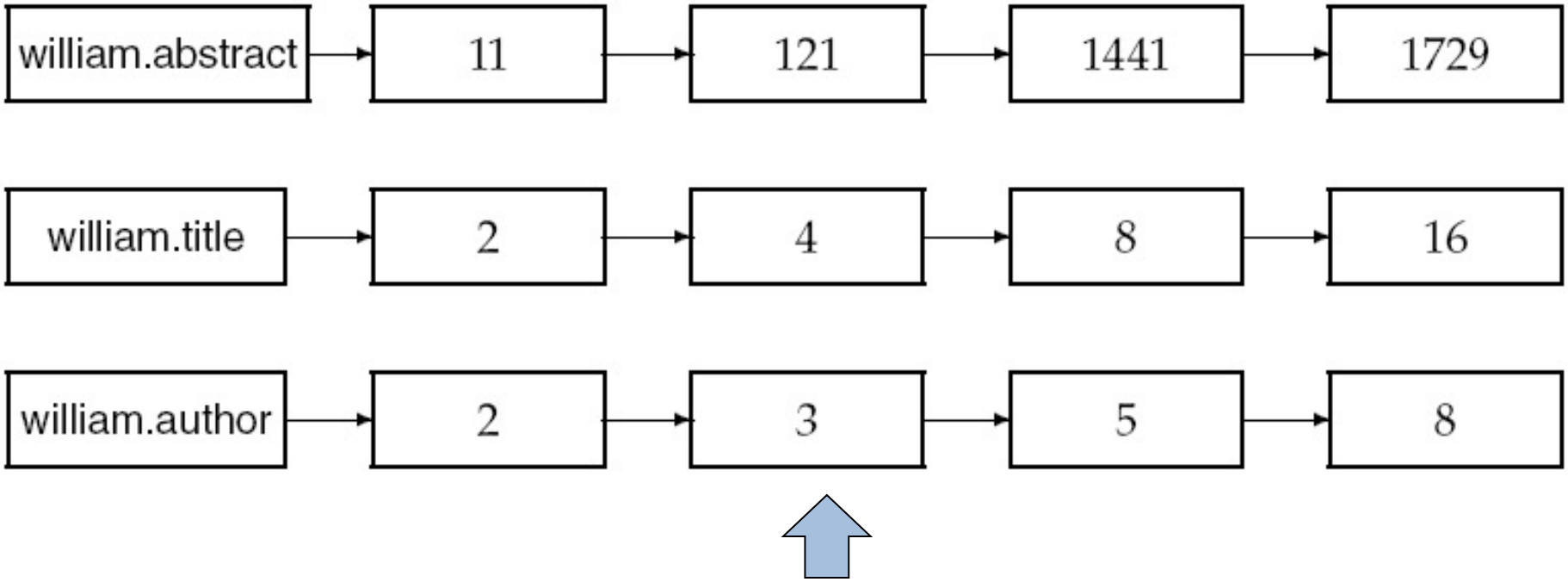
# Zone

- A zone is a region of the doc that can contain an arbitrary amount of text, e.g.,
  - Title
  - Abstract
  - References …

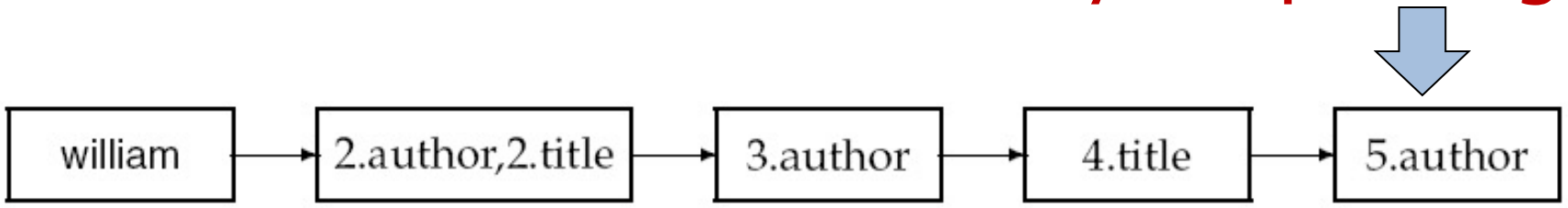
# Zone

- A zone is a region of the doc that can contain an arbitrary amount of text, e.g.,
  - Title
  - Abstract
  - References …
- Build inverted indexes on zones as well to permit querying
- E.g., "find docs with *merchant* in the title zone and matching the query *gentle rain*"

# Example zone indexes




Encode zones in dictionary vs. postings.



# Tiered indexes

- Break postings up into a hierarchy of lists
  - Most important
  - …
  - Least important

# Tiered indexes

- Break postings up into a hierarchy of lists
  - Most important
  - …
  - Least important

- Can be done by *g(d)* or another measure (zone weight, tf-idf, etc…)

# Tiered indexes

- Break postings up into a hierarchy of lists
  - Most important
  - …
  - Least important

- Can be done by *g(d)* or another measure (zone weight, tf-idf, etc…)

- Inverted index thus broken up into tiers of decreasing importance

# Tiered indexes

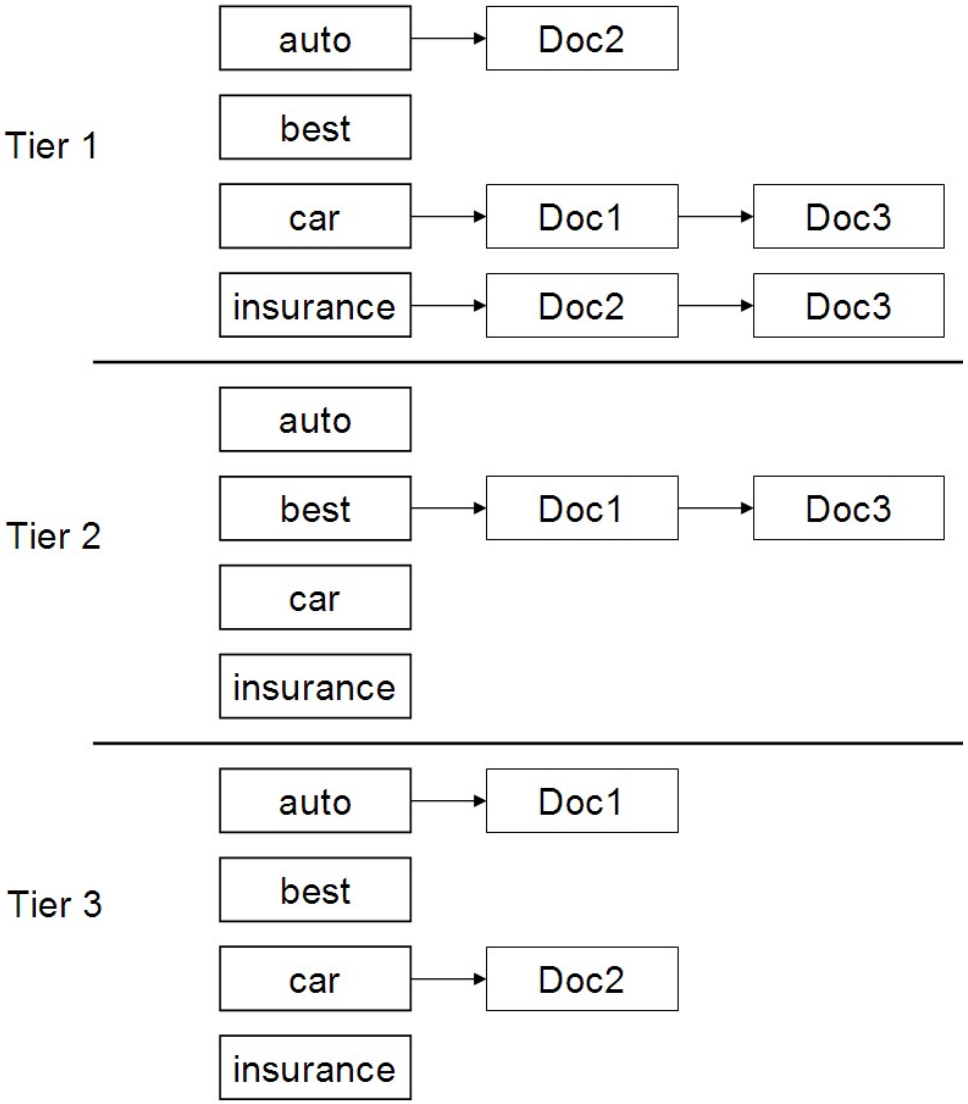
- Break postings up into a hierarchy of lists
  - Most important
  - …
  - Least important

- Can be done by *g(d)* or another measure (zone weight, tf-idf, etc…)

- 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 term proximity

- Free text queries: just a set of terms typed into the query box – common on the web

# 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

# 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

- Let *w* be the smallest window in a doc containing all query terms, e.g.,

# 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

- Let *w* be the smallest window in a doc containing all query terms, e.g.,
  - For the query *strained mercy* the smallest window in the doc *The quality of mercy is not strained* is 4 (words)

# 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

- Let *w* be the smallest window in a doc containing all query terms, e.g.,
  - For the query *strained mercy* the smallest window in the doc *The quality of mercy is not strained* is 4 (words)

  - Intuition : the smaller *w* is, the better the document matches the query.

# Query term proximity

- Free text queries: just a set of terms typed into the query box – common on the web

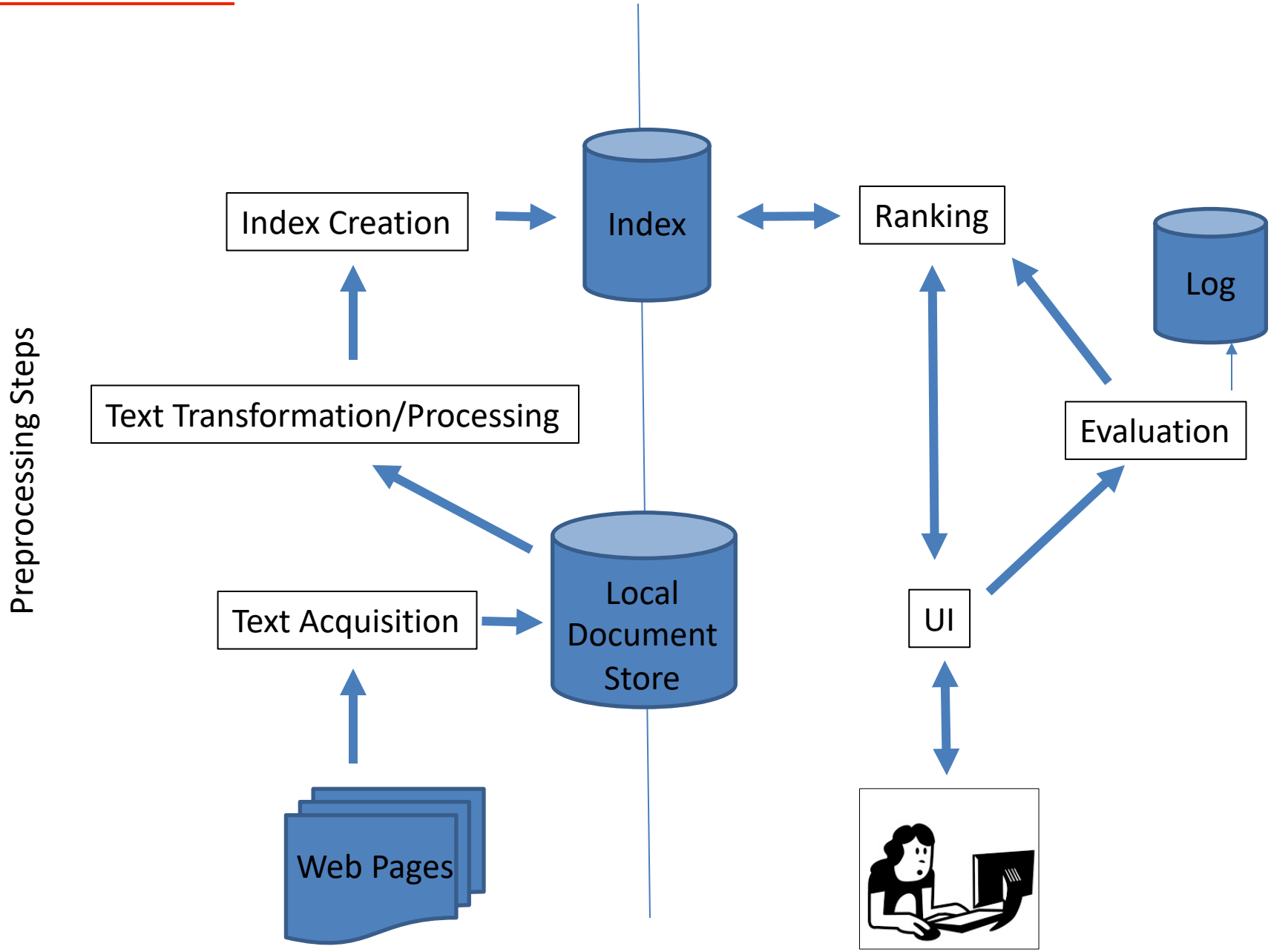
- Users prefer docs in which query terms occur within close proximity of each other

- Let *w* be the smallest window in a doc containing all query terms, e.g.,
  - For the query *strained mercy* the smallest window in the doc *The quality of mercy is not strained* is 4 (words)

  - Intuition : the smaller *w* is, the better the document matches the query.

  - **Would like scoring function to take this into account : add weights!**

# Architecture



# Query parsers

- Free text query from user may in fact spawn one or more queries to the indexes, e.g., query *rising interest rates*

# Query parsers

- Free text query from user may in fact spawn one or more queries to the indexes, 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*
  - Rank matching docs by vector space scoring

# Query parsers

- Free text query from user may in fact spawn one or more queries to the indexes, 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*
  - Rank matching docs by vector space scoring

- This sequence is issued by a **query parser**
  - Some method that will transform a user query into a stream of multiple queries.

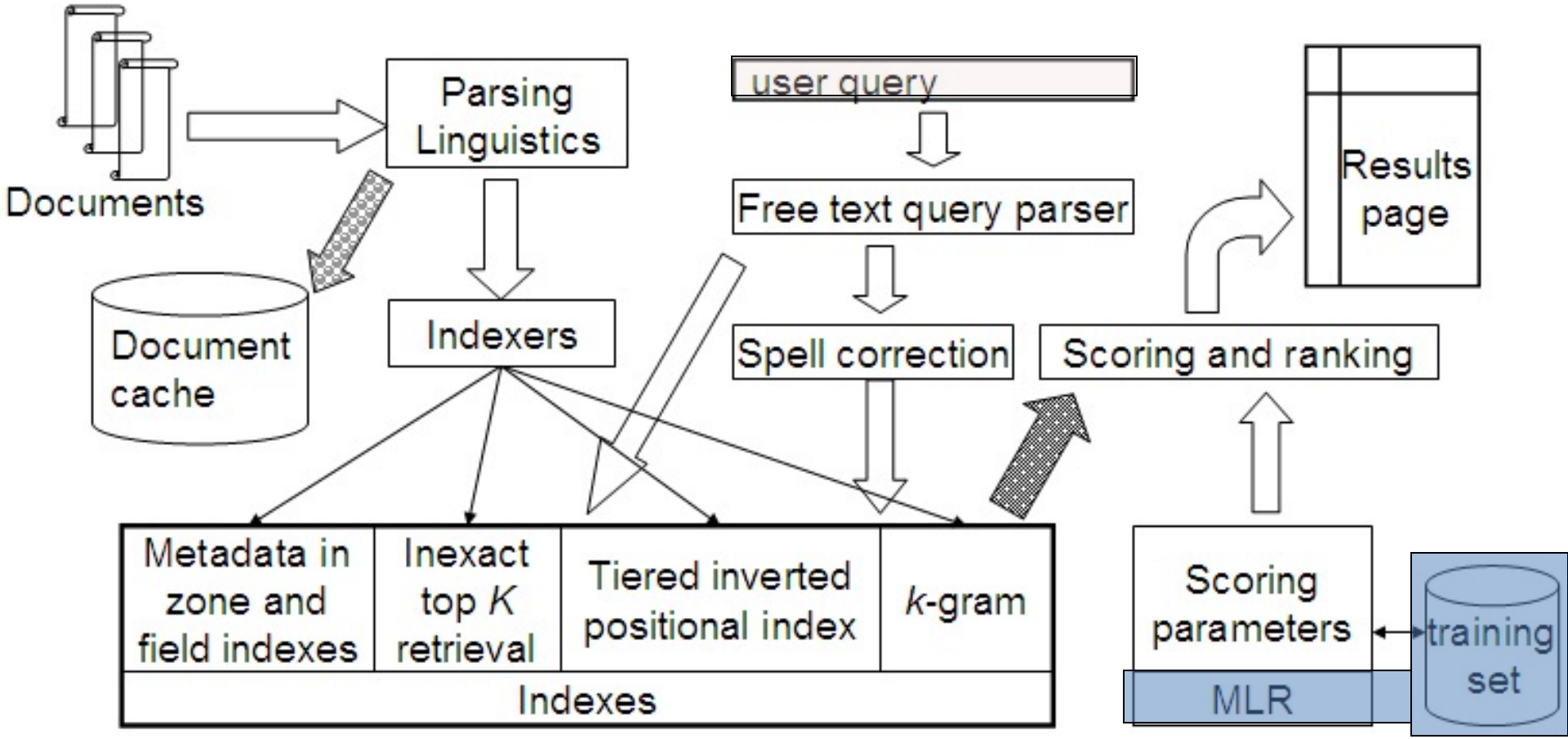
# Aggregate scores

- We've seen that score functions can combine cosine, static quality, proximity, etc.

# Aggregate scores

- We've seen that score functions can combine cosine, static quality, proximity, etc.
- How do we know the best combination?
  - *User surveys : ask real people to use your search engine (think MS3!)*
  - Some applications – expert-tuned
  - Hand-tune real-life systems is hard (e.g. tens or hundreds of possible threshold choices, what to consider or not, etc.)
    - Increasingly common technique: adaptive-learning based optimization

# Putting it all together



# Architecture