DATA STRUCTURES

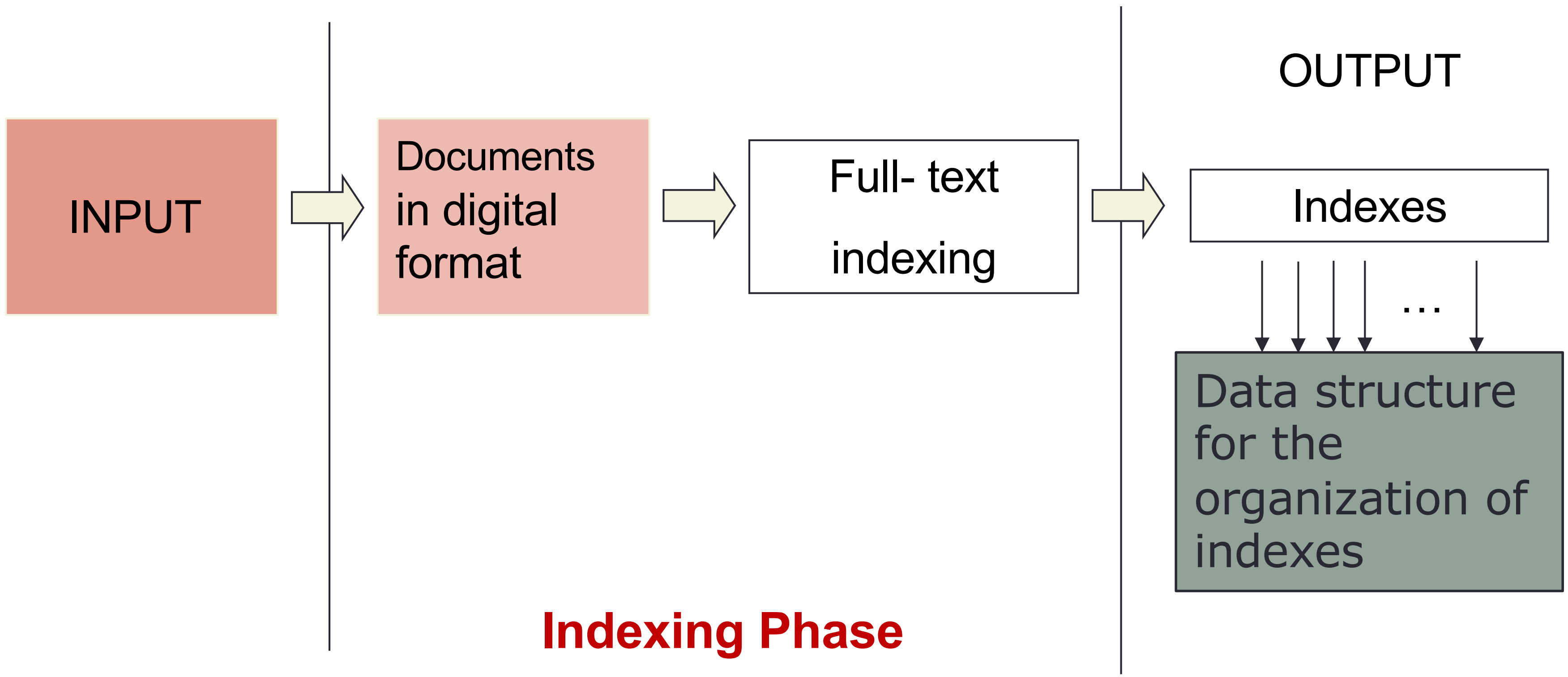
Gabriella Pasi

gabriella.pasi@unimib.it

Automatic indexing of text documents

- The **automatic indexing** of a textual document is the process aimed at the association of **indexes** (index terms) with a text.
- Typically: **full-text indexing**.
- The use of the indexes makes the retrieval **efficient** based on the keywords specified in a query (example: analytical index of a book).
- **DICTIONARY** → All the indexes extracted / associated with all the documents of the collection considered constitute the dictionary of the collection.

Scheme of automatic indexing process



The indexing process output **representation**

	d_1	$\dots$	$\dots$	d_i	$\dots$	$\dots$	d_m
t_1	w_{11}	$\dots$	$\dots$	w_{1i}	$\dots$	$\dots$	w_{1m}
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$
t_k	w_{k1}	$\dots$	$\dots$	w_{ki}	$\dots$	$\dots$	w_{km}
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$
t_n	w_{n1}	$\dots$	$\dots$	w_{ni}	$\dots$	$\dots$	w_{nm}

- ATTENTION: this is a **representation**!!!
- **Sparse matrix**! (presence of many 0 in each column).
- Weights w_{ij} can be binary values, real values or positive integers: they are calculated during the indexing phase.

Creating a data structure to organize indexes

Purpose (efficiency):

- Give "immediate" (**very fast**) answers to user requests.
- **Optimize access** to the dictionary (queries are specified by keywords).
- "**Off-line**" construction before search.

Inverted file structure (motivations)

DocID	Term1	Term2	Term3	Term4	Term5	Term6	Term7
Doc1	3	4	-	-	-	-	-
Doc2	-	-	1	2	6	5	8
Doc3	-	2	-	1	3	-	-
Doc4	-	-	-	-	4	7	6

- The organization of the indexes in a "**static**" data structure, for example a matrix of document/term occurrences would be *very* inefficient.
  - Since the matrix is **sparse**, space would be wasted to store also the "non-occurrence" of the terms (e.g., Term3 in Doc1).
  - **Getting** the list of documents related to a specific term would be burdensome.

«Inverted file» structure (motivations)

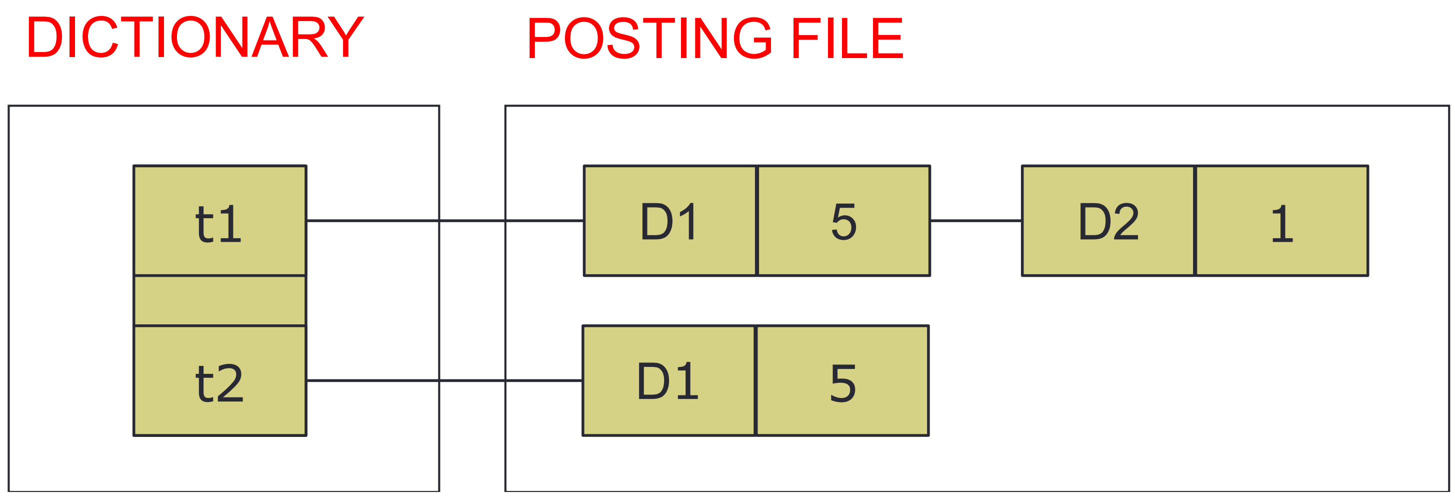
Term	Doc1	Doc2	Doc3	Doc4
Term1	3	-	-	-
Term2	4	-	2	-
Term3	-	1	-	-
Term4	-	2	1	-
Term5	-	6	3	4
Term6	-	5	-	7
Term7	-	8	-	6

- Idea to solve the above problems: the table of occurrences is **inverted** (term/document) and organized in a **dynamic** data structure.

«Inverted file» data structure

- The **index terms** constituting the dictionary of the document repository are organized and stored in an ad hoc file. Each term "points" to a list (store in another file) containing the references to documents of which the term is an index.

- Use of two main files: **dictionary** and **posting file** (it contains the posting lists of all index terms).



«Inverted file» data structure

- Independent from the Retrieval model that the IR system adopts.
  - We will see IR models in the next lessons.

- Information associated with each **index**:
  - **The list of documents containing it** plus document related information (in the posting file).
  - **The term occurrences in the collection** (in the dictionary) for the computation of the IDF, and to optimize the evaluation of Boolean queries).

Simple representation of the inverted file

- **D1**: The GDP incrased 2 percent this quarter.

- **D2**: The spring economic slowdown continued to spring downwards this quarter.

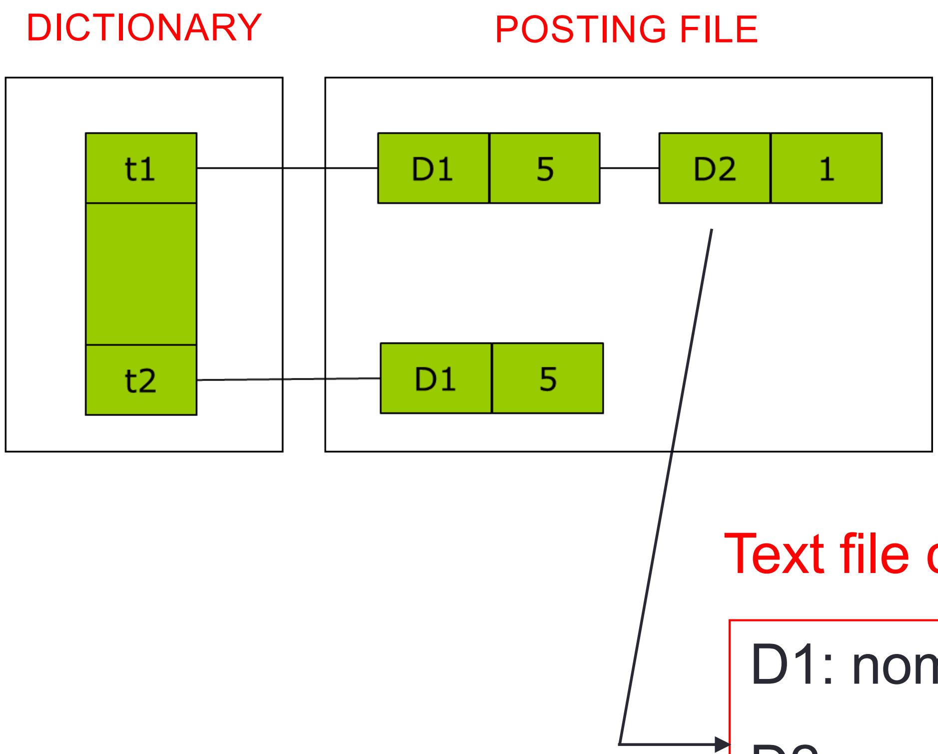
Dictionary File		Posting File
2	1	→ D1
continued	1	→ D2
downwards	1	→ D2
economic	1	→ D2
GDP	1	→ D1
increased	1	→ D1
percent	1	→ D1
quarter	2	→ D1, D2
slowdown	1	→ D2
spring	1	→ D2
this	2	→ D1, D2
the	2	→ D1, D2
to	1	→ D2

«Inverted file» data structure

- The **dictionary** contains for each index: term, global frequency in the dictionary, pointer to the posting list in the posting file.

- The **posting list** (may) contain for each element:
  - The unique document identifier (DocID) that is associated with a file name or URL location (mandatory).
  - The frequency of the term in the document (tf).
  - The position in the document of each occurrence of the term (optional, only for query with context). The position can be expressed as:
    - word number from the beginning of the document,
    - number of bytes from the beginning of the document,
    - section number, paragraph, sentence, word number in the sentence.

Posting file – DocID



Text file containing the ordered list of DocIDs

D1: nome_file1.doc,	ms_word
D2: nome_file2.txt,	wordpad
D3: nome_file3,	acrobat_reader
…	

Posting file – Location of occurrences

- **Location of occurrences**:
  - The posting file can maintain information about the location of each occurrence of a term in the documents
  - Use:
    - evaluation of *adjacency* and *proximity* operators.
    - design of user interfaces for highlighting occurrences of the terms searched within the retrieved documents.

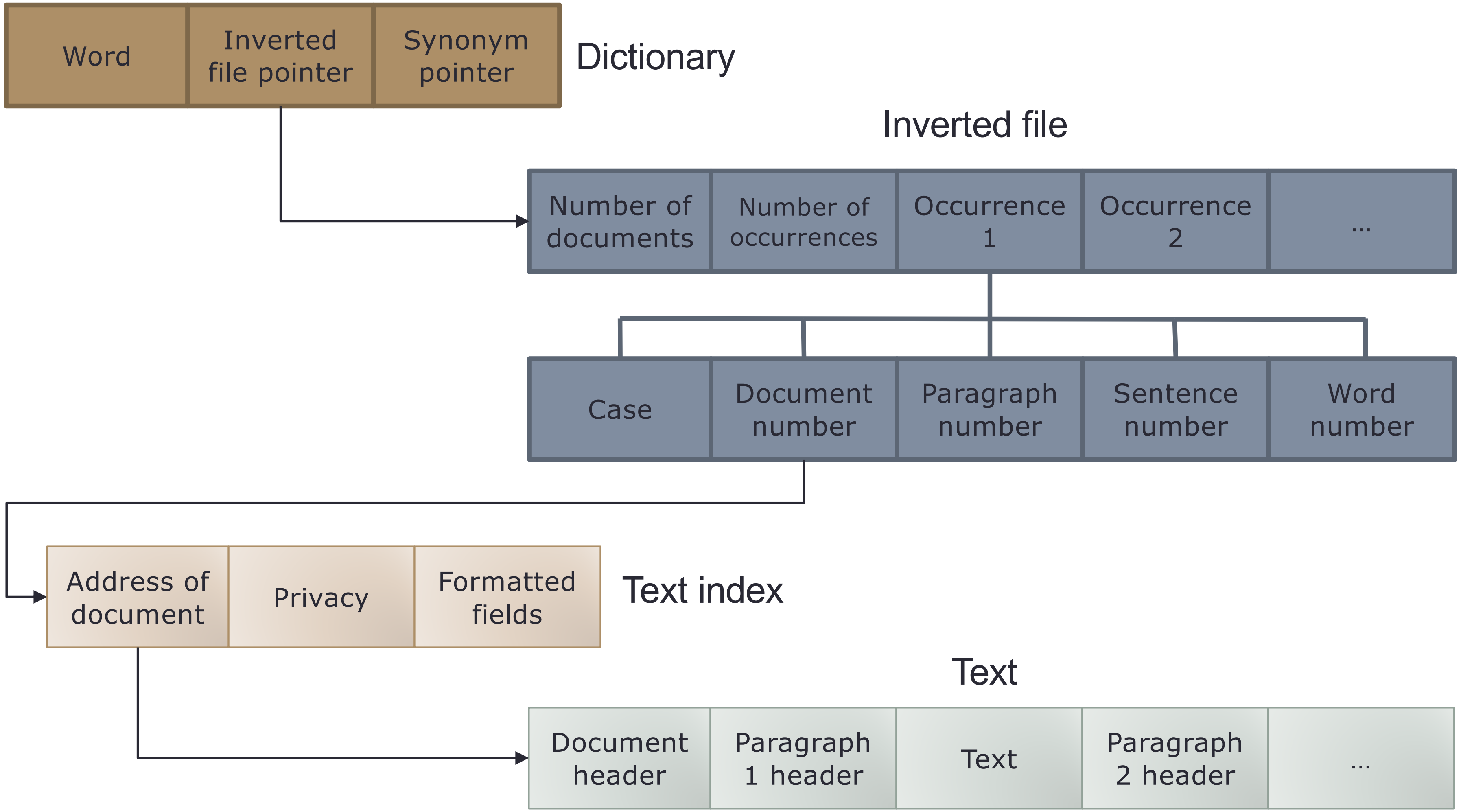
Improved representation of the posting file

- **D1**: The GDP incrased 2 percent this economy quarter.

- **D2**: The spring economic slowdown continued to spring downwards this economy quarter.

Dictionary File		Posting File
2	1	→ D1:4
continued	1	→ D2:5
downwards	1	→ D2:8
economic	1	→ D2:3
economy	2	→ D1:7, D2:10
GDP	1	→ D1:2
increased	1	→ D1:3
percent	2	→ D1:5
quarter	2	→ D1:8, D2:11
slowdown	1	→ D2:4
spring	2	→ D2:2

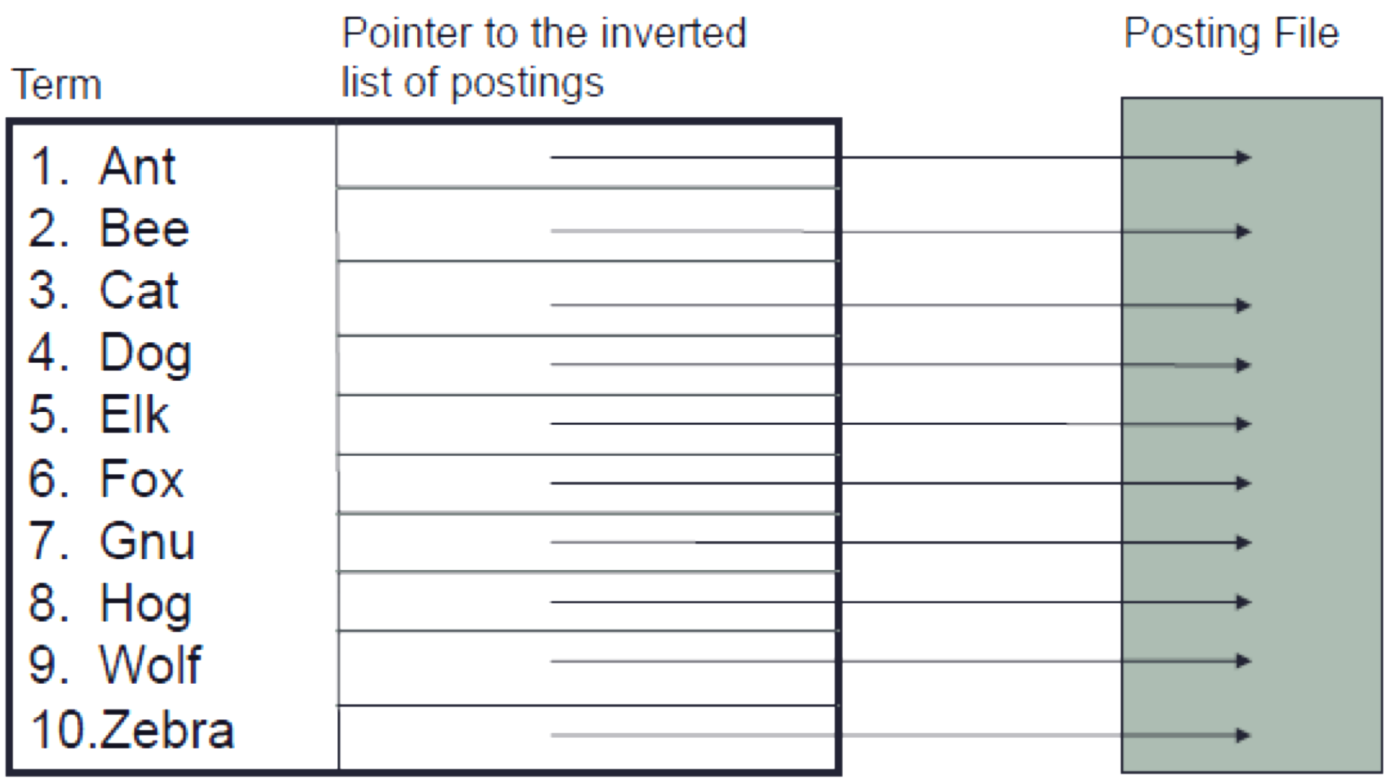
Inverted file of the STAIRS system (IBM)



Dictionary organization

1) Dictionary – Linear structure

- Index terms are stored in the dictionary in alphabetical order.



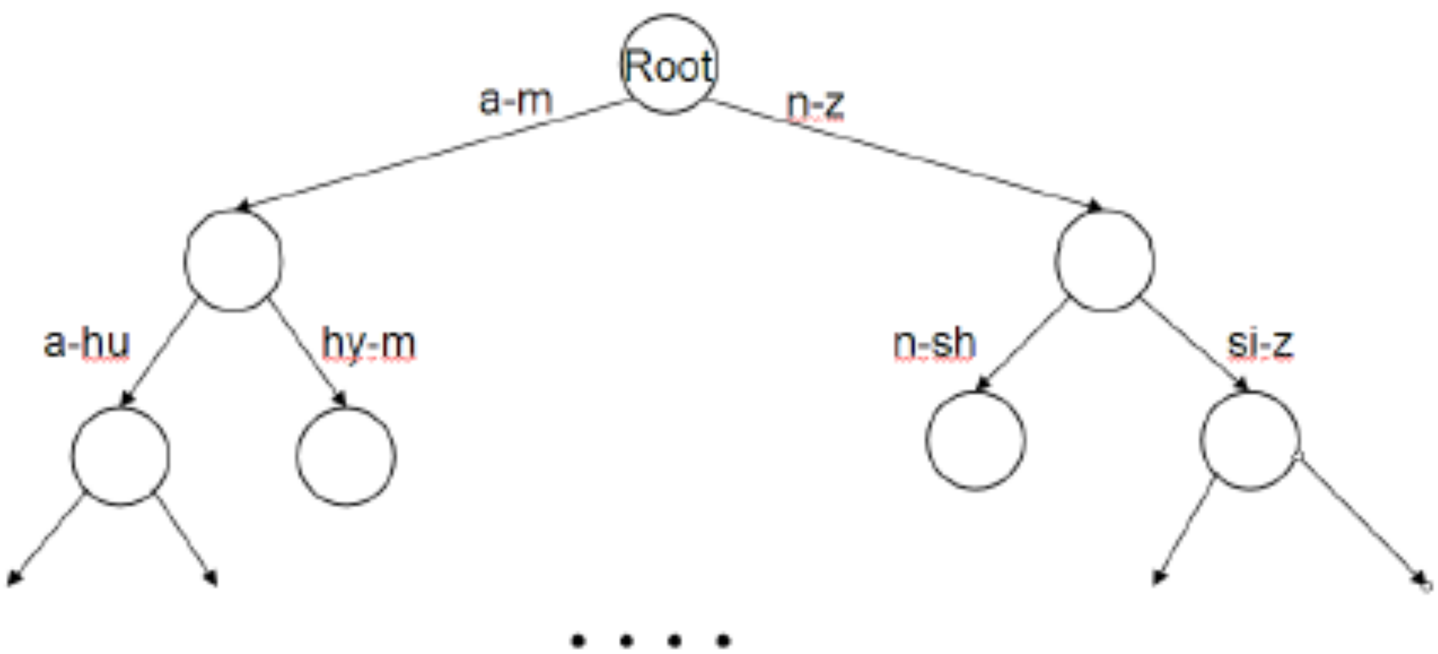
1) Dictionary – Linear structure

- **Advantages**
  - It can be accessed quickly, e.g., with a binary search $O(\log n)$.
  - It is suitable for sequential evaluation, e.g., comp*
  - Efficient use of memory space.

- **Disadvantages**
  - Indexes must be rebuilt if new terms are inserted.

2) Dictionary – Binary tree structure

- Every node of the tree **has two children**.
  - The search for the term begins at the root of the tree
  - Each node (including the root) allows a binary test (lexigographic order), on the basis of which the search proceeds in one of the two sub-nodes below the considered node.



2) Dictionary – Binary tree structure

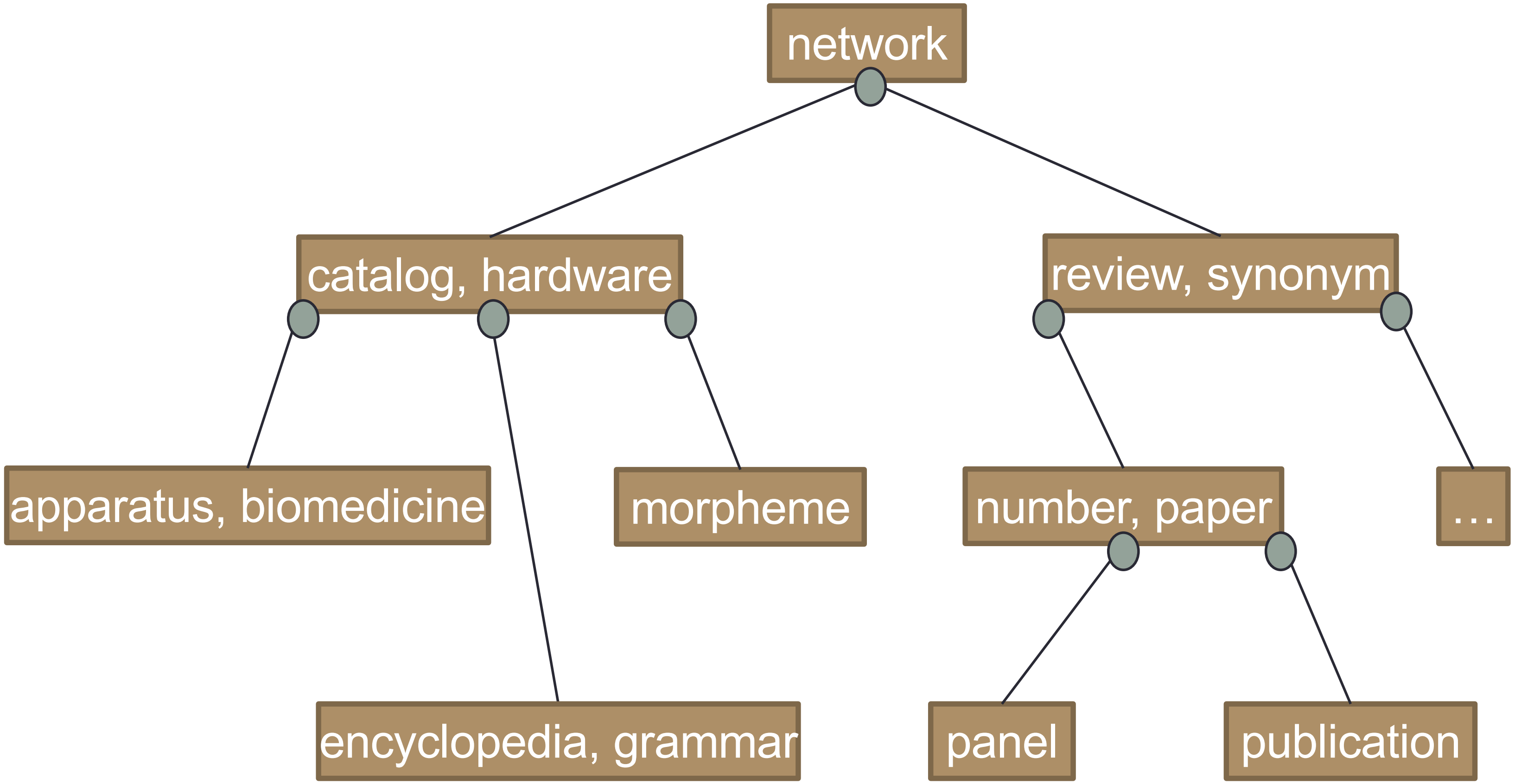
- **Advantages**
  - Efficient search (the number of comparisons is at most $O(\log n)$).

- **Disadvantages**
  - The main problem is to rebalance the data structure: since the terms are inserted and eliminated from the binary search tree, it must be re-updated each time so that the equilibrium property is maintained.

3) Dictionary – B-tree structure

- B-tree of order d:
  - It is a balanced tree.
  - Each node has a variable number of children.
  - Each node of a B-tree contains several terms (max d) and pointers to sub-trees:
    - the root contains a number of terms ranging between 1 and d.
    - all other nodes may contain a number of nodes between $d/2$ and d.
  - If k_i is the i-th term in an intermediate node:
    - all terms contained in the children nodes from the first to the i-th are lexicographically smaller than k_i.
    - all terms contained in the children nodes from the $(i+1)$-th are lexicographically larger than k_i.

3) Dictionary – B-tree structure



Dictionary – B-tree structure

- Advantages
  - It can be accessed quickly: the maximum number of accesses to a B-tree of order d → $O(\log_d n)$, where n is the number of levels (height of the B-tree).
  - New terms can be added quickly.
  - Efficient use of memory space.

- Disadvantages
  - Inefficient for sequential search, comp*
  - Become unbalanced after numerous insertions if n grows.