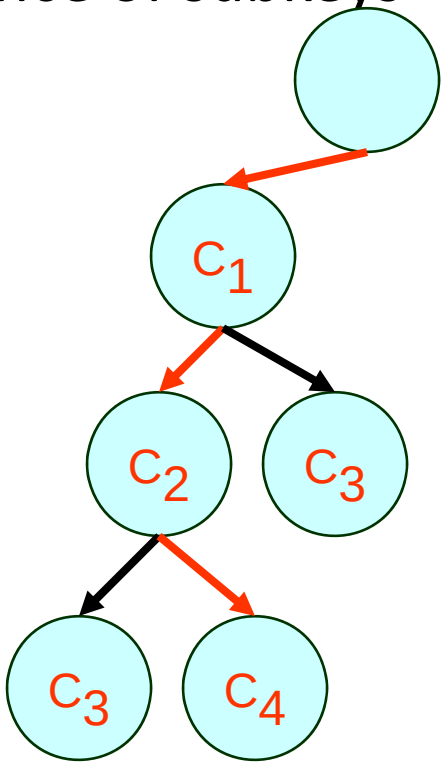
Trie Tree

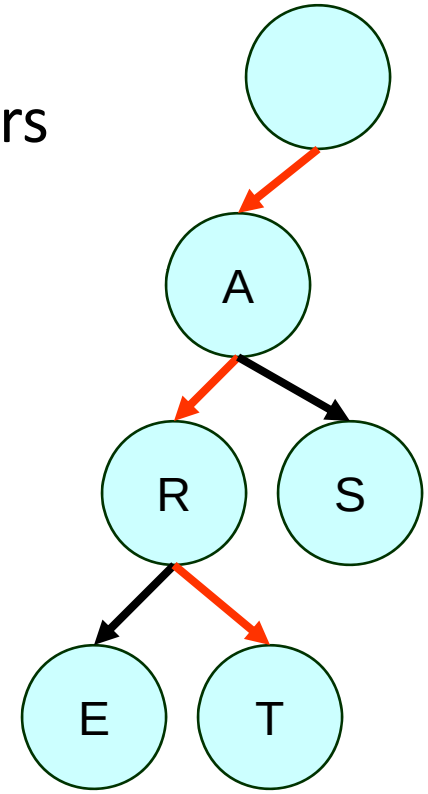
Indexed Search Tree (Trie)

- Special case of tree
- Applicable when
  - Key C can be decomposed into a sequence of subkeys $C_1, C_2, \ldots C_n$
  - Redundancy exists between subkeys
- Approach
  - Store subkey at each node
  - Path through trie yields full key
- Example
  - Huffman tree



Tries

- Useful for searching strings
  - String decomposes into sequence of letters
  - Example
    - "ART" $\Rightarrow$ "A" "R" "T"
- Can be very fast
  - Less overhead than hashing
- May reduce memory
  - Exploiting redundancy
- May require more memory
  - Explicitly storing substrings



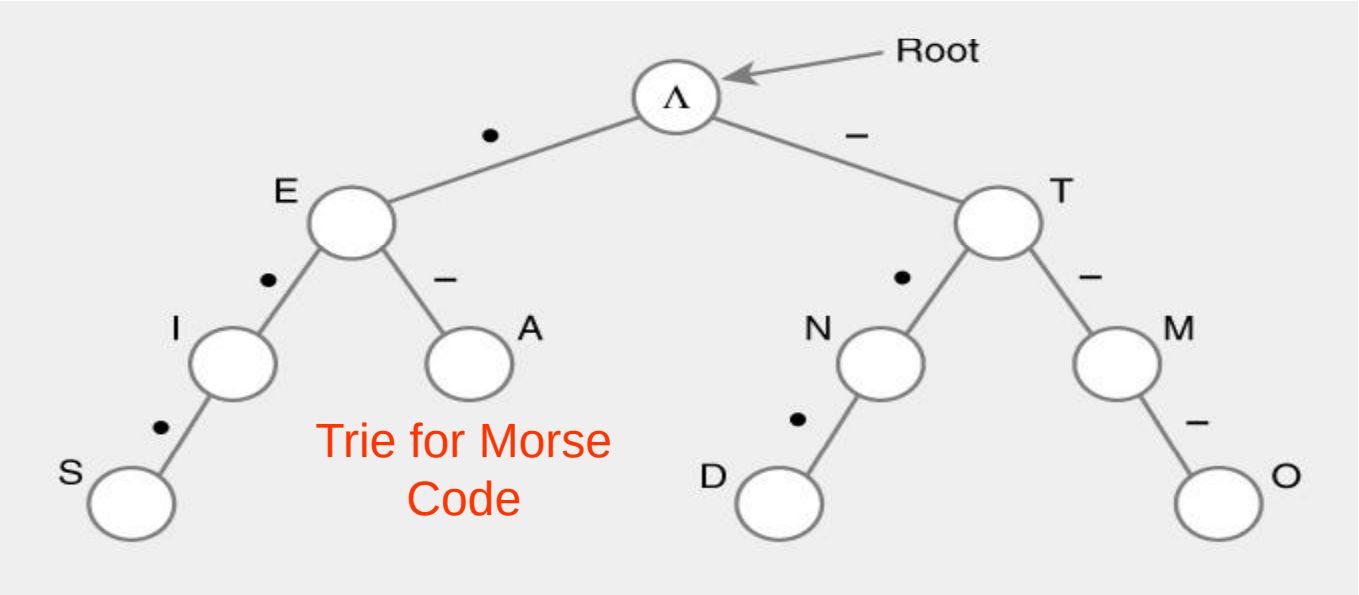
"ART"

Types of Tries

- Standard
  - Single character per node
- Compressed
  - Eliminating chains of nodes
- Compact
  - Stores indices into original string(s)
- Suffix
  - Stores all suffixes of string

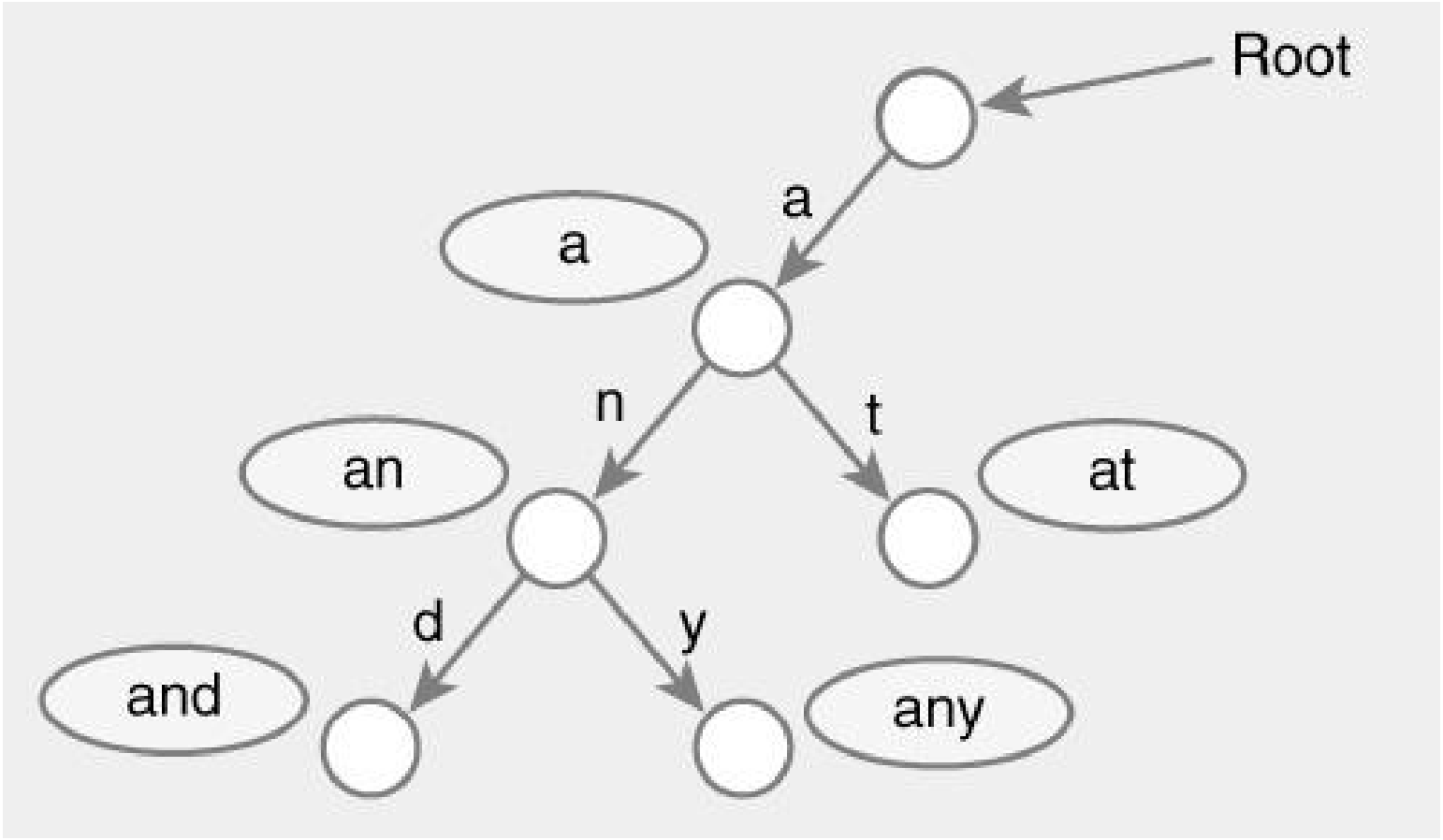
Standard Tries

- Approach
  - Each node (except root) is labeled with a character
  - Children of node are ordered (alphabetically)
  - Paths from root to leaves yield all input strings



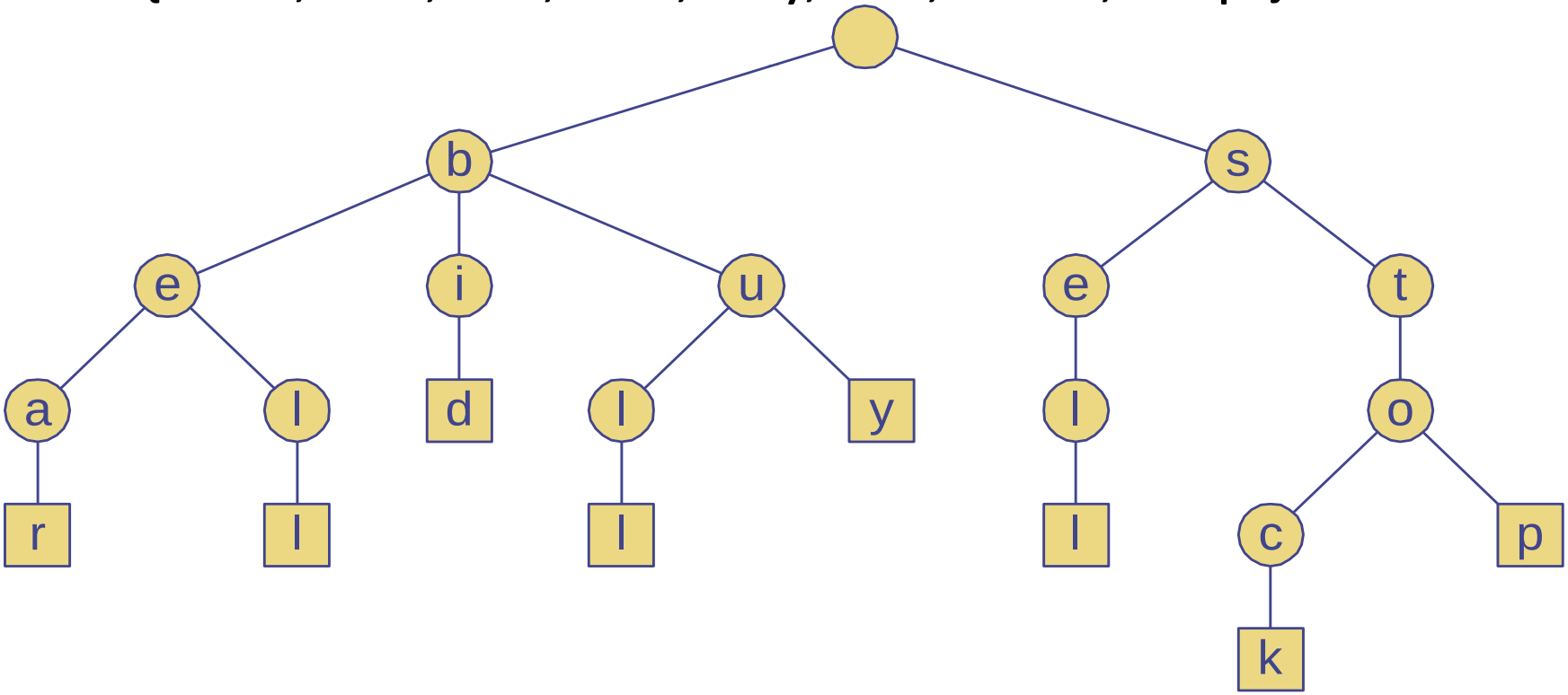
Standard Trie Example

- For strings



Standard Trie Example

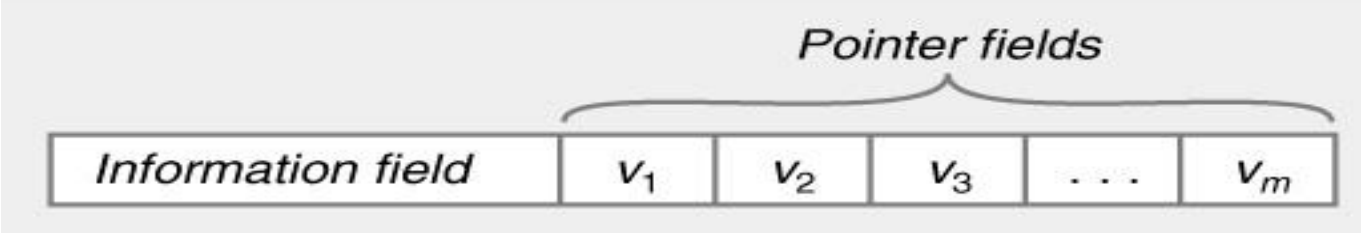
- For strings
  - { bear, bell, bid, bull, buy, sell, stock, stop }



Standard Tries

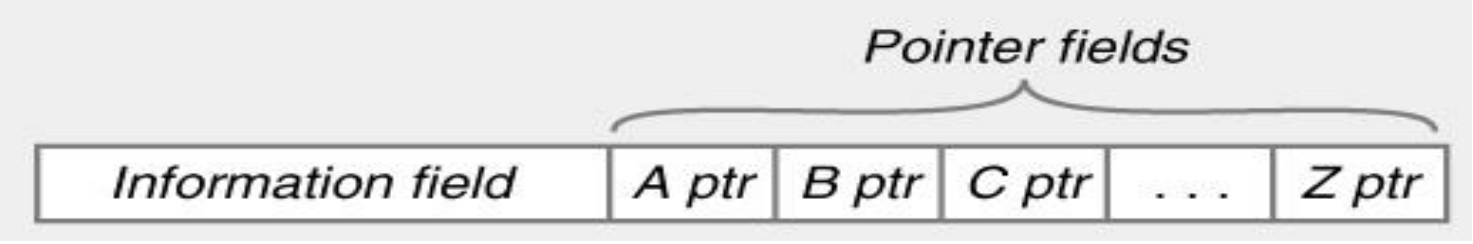
- Node structure
  - Value between 1…m
  - Reference to m children
    - Array or linked list
- Example

```
Class Node {
    Letter value;    // Letter V = { V1, V2, … Vm }
    Node child[ m ];
}
```



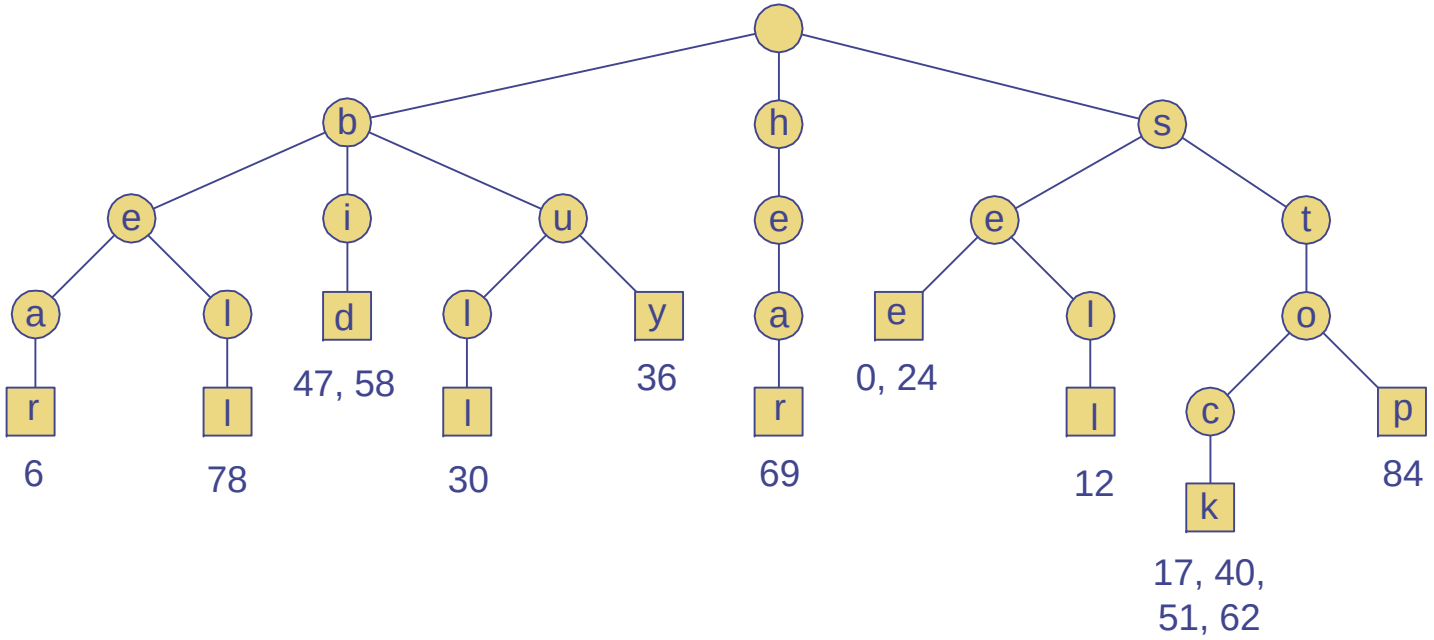
Standard Tries

- Efficiency
  - Uses O(n) space
  - Supports search / insert / delete in O(d×m) time
  - For
    - n total size of strings indexed by trie
    - d length of the parameter string
    - m size of the alphabet



Word Matching Trie

- Insert words into trie
- Each leaf stores occurrences of word in the text

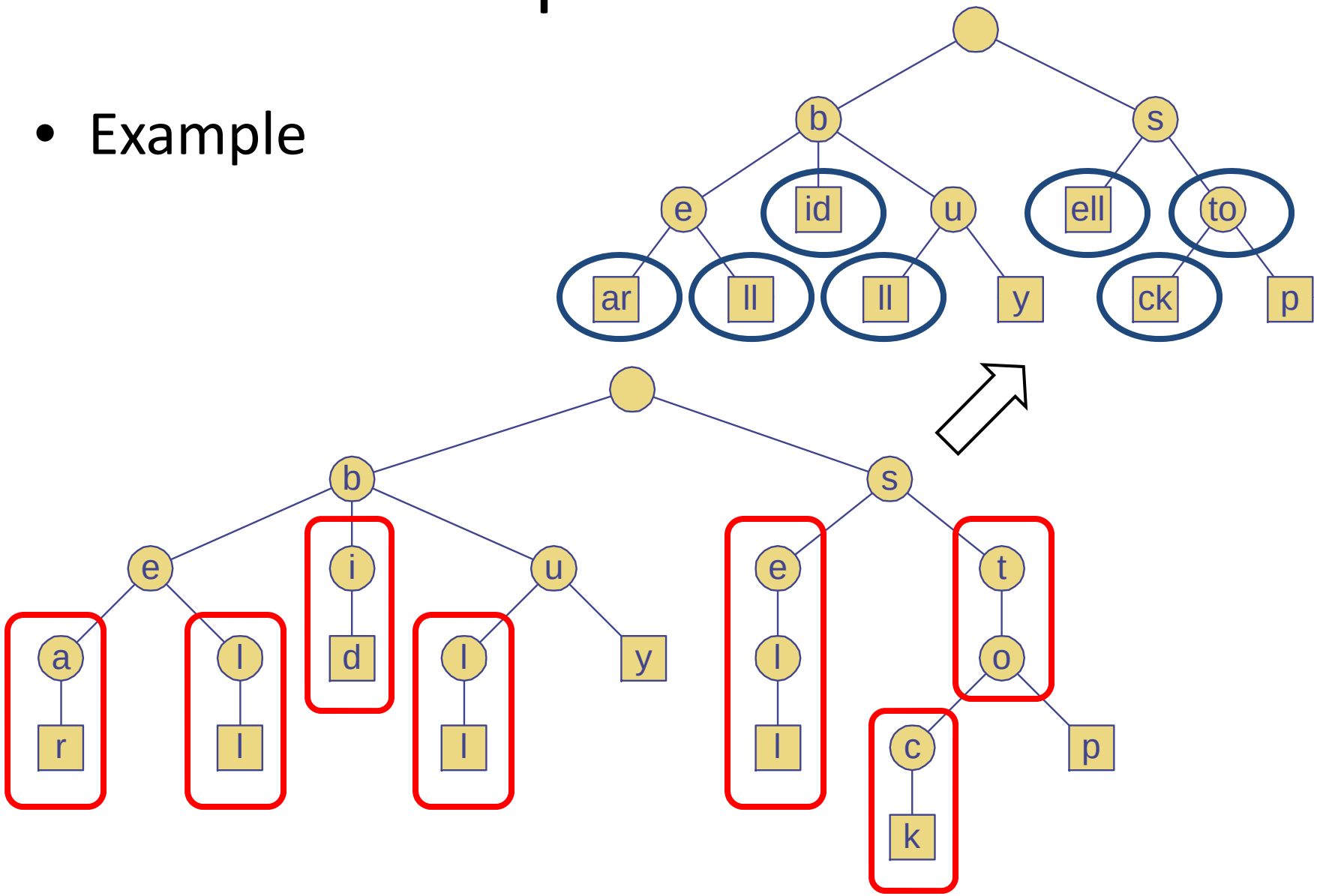


Compressed Trie

- Observation
  - Internal node v of T is redundant if v has one child and is not the root
- Approach
  - A chain of redundant nodes can be compressed
    - Replace chain with single node
    - Include concatenation of labels from chain
- Result
  - Internal nodes have at least 2 children
  - Some nodes have multiple characters

Compressed Trie

- Example

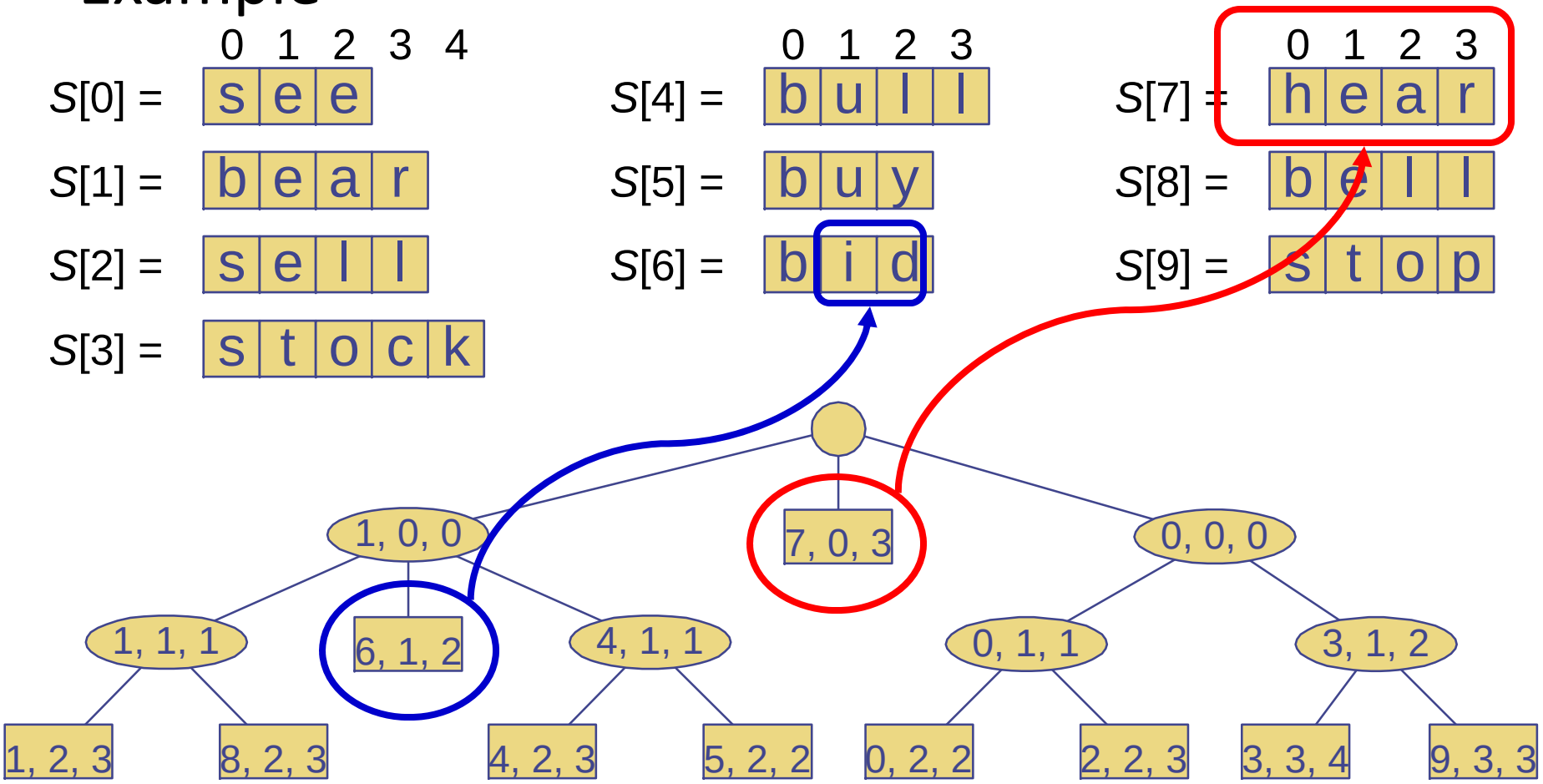


Compact Tries

- Compact representation of a compressed trie
- Approach
  - For an array of strings S = S[0], … S[s-1]
  - Store ranges of indices at each node
    - Instead of substring
  - Represent as a triplet of integers (i, j, k)
    - Such that X = s[i][j..k]
  - Example: S[0] = “abcd”, (0,1,2) = “bc”
- Properties
  - Uses O(s) space, where s = # of strings in the array
  - Serves as an auxiliary index structure

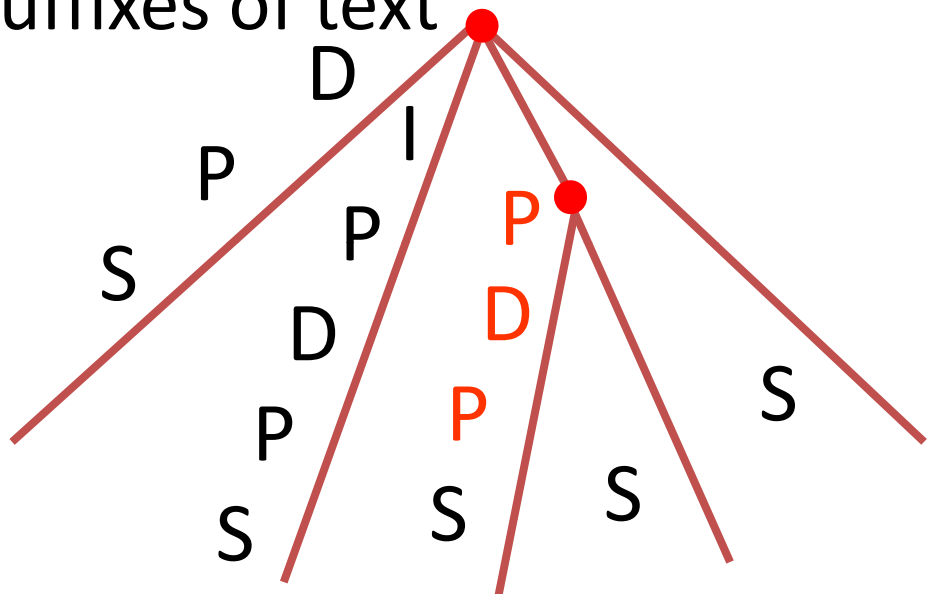
Compact Representation

- Example



Suffix Trie

- Compressed trie of all suffixes of text
- Example: “IPDPS”
  - Suffixes
    - IPDPS
    - PDPS
    - DPS
    - PS
    - S

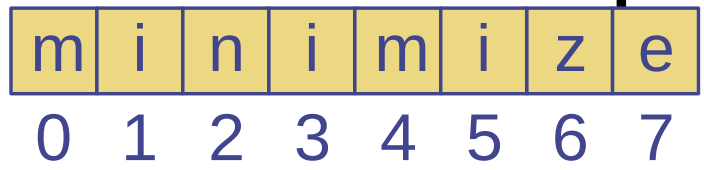


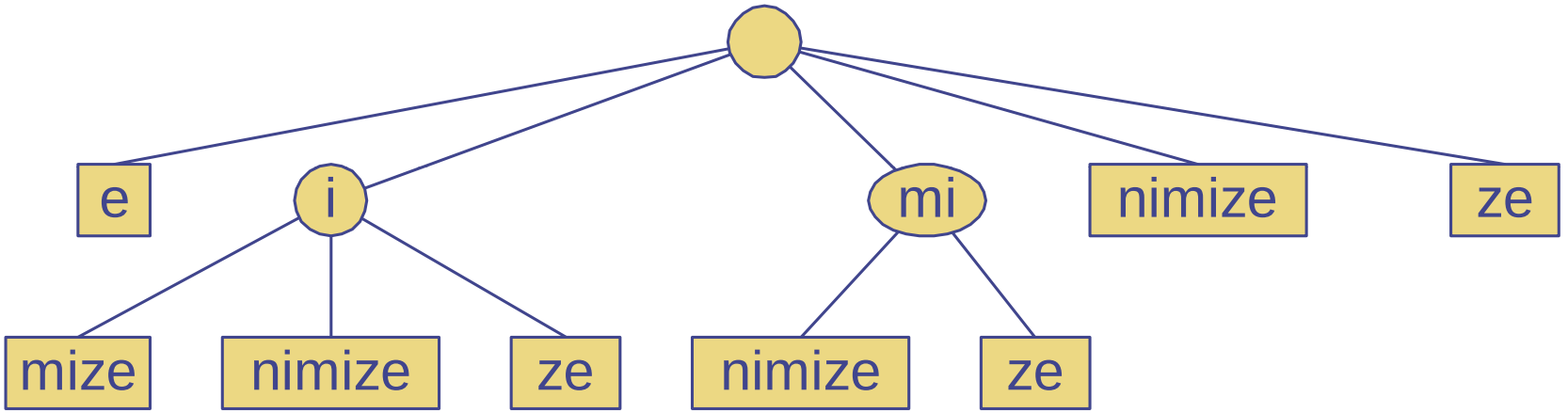
- Useful for finding pattern in any part of text
  - Occurrence $\Rightarrow$ prefix of some suffix
  - Example: find PDP in IPDPS

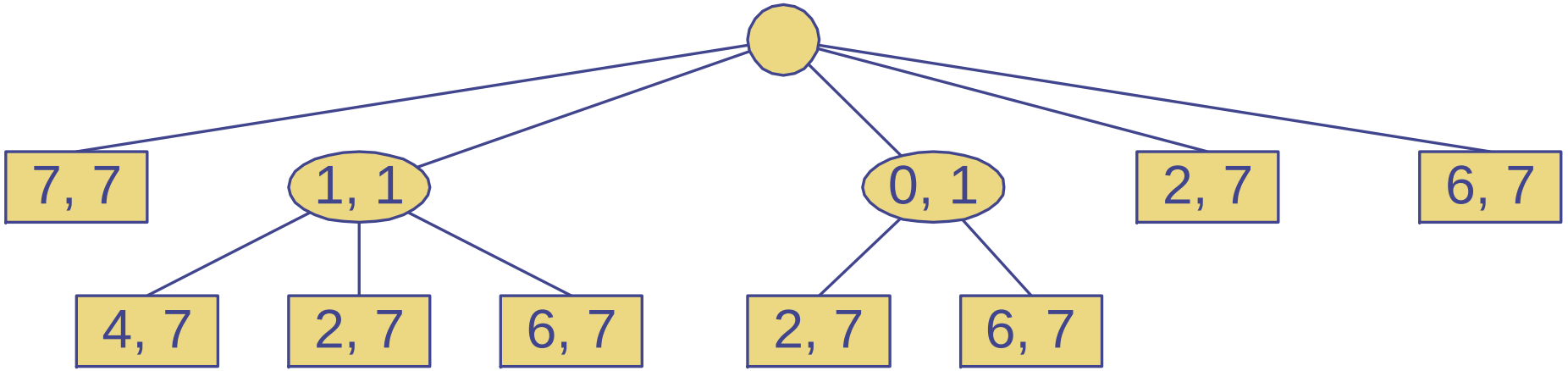
Suffix Trie

- Properties
  - For
    - String X with length n
    - Alphabet of size m
    - Pattern P with length d
  - Uses $O(n)$ space
  - Can be constructed in $O(n)$ time
  - Find pattern P in X in $O(d \times m)$ time
    - Proportional to length of pattern, not text

Suffix Trie Example







Tries and Web Search Engines

- Search engine index
  - Collection of all searchable words
  - Stored in compressed trie
- Each leaf of trie
  - Associated with a word
  - List of pages (URLs) containing that word
    - Called occurrence list
- Trie is kept in memory (fast)
- Occurrence lists kept in external memory
  - Ranked by relevance

Trie Insertion

1. If the input string length is zero, then set the marker for the root node to be true.

2. If the input string length is greater than zero, repeat steps 3 and 4 for each character

3. If the character is present in the child node of the current node, set the current node point to the child node.

4. If the character is not present in the child node, then insert a new node and set the current node to that newly inserted node.

5. Set the marker flag to true when the end character is reached.

Thank You