CS350: Data Structures

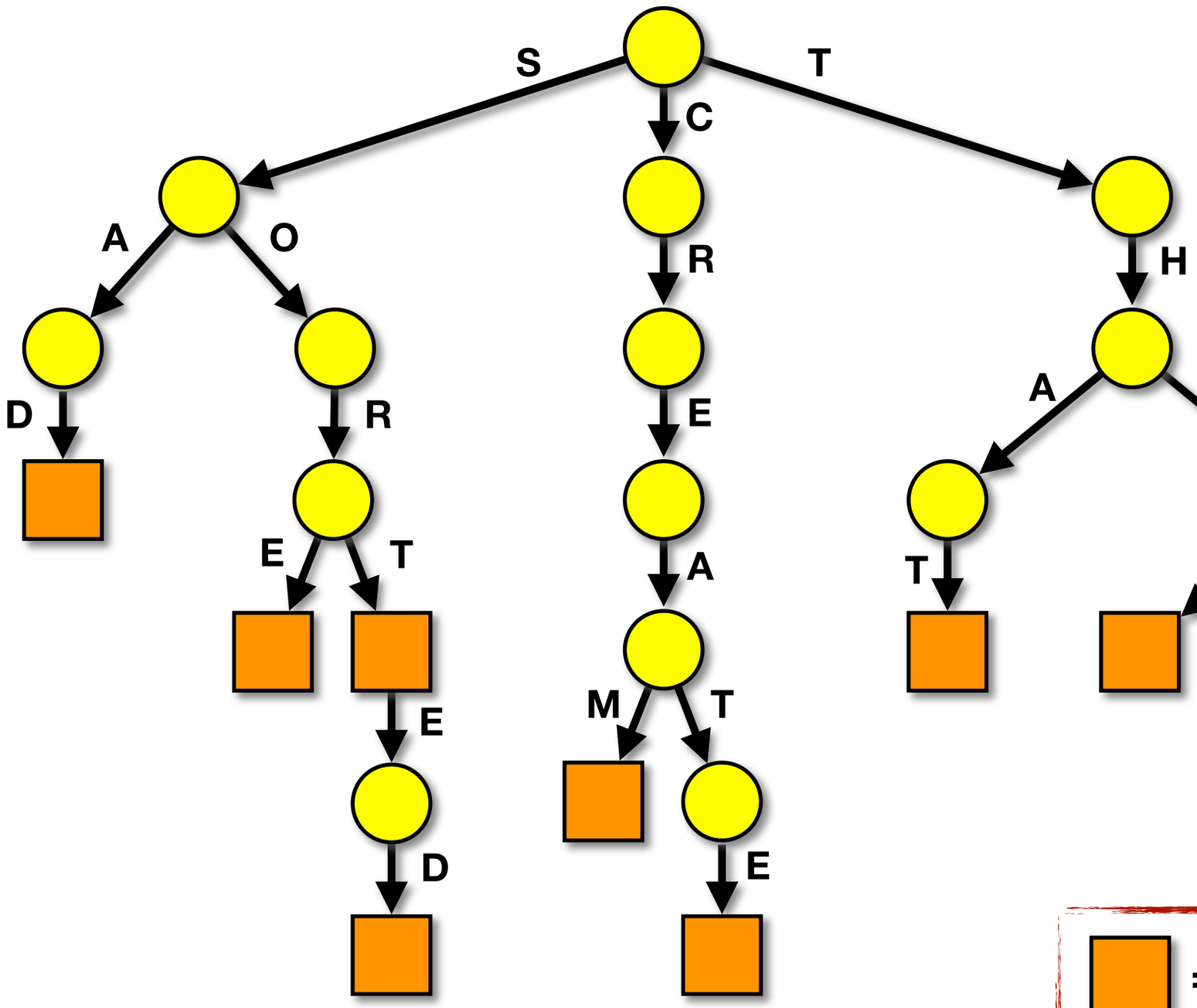
Tries

James Moscola
Department of Engineering & Computer Science
York College of Pennsylvania

Tries

- **A type of tree used to store associative arrays (e.g. (key, value) pairs)**
  - Number of internal nodes from root node to leaf node is equal to the length of the key
- **Often used to store a set of character strings, such as a dictionary**
  - Facilitates longest-prefix matching algorithms
- **The term ‘trie’ is comes from the word ‘retrieval’**
- **You may occasionally hear them called PATRICIA Tries**
  - **P**ractical **A**lgorithm **T**o **R**etrieve **I**nformation **C**oded **I**n **A**lphanumeric

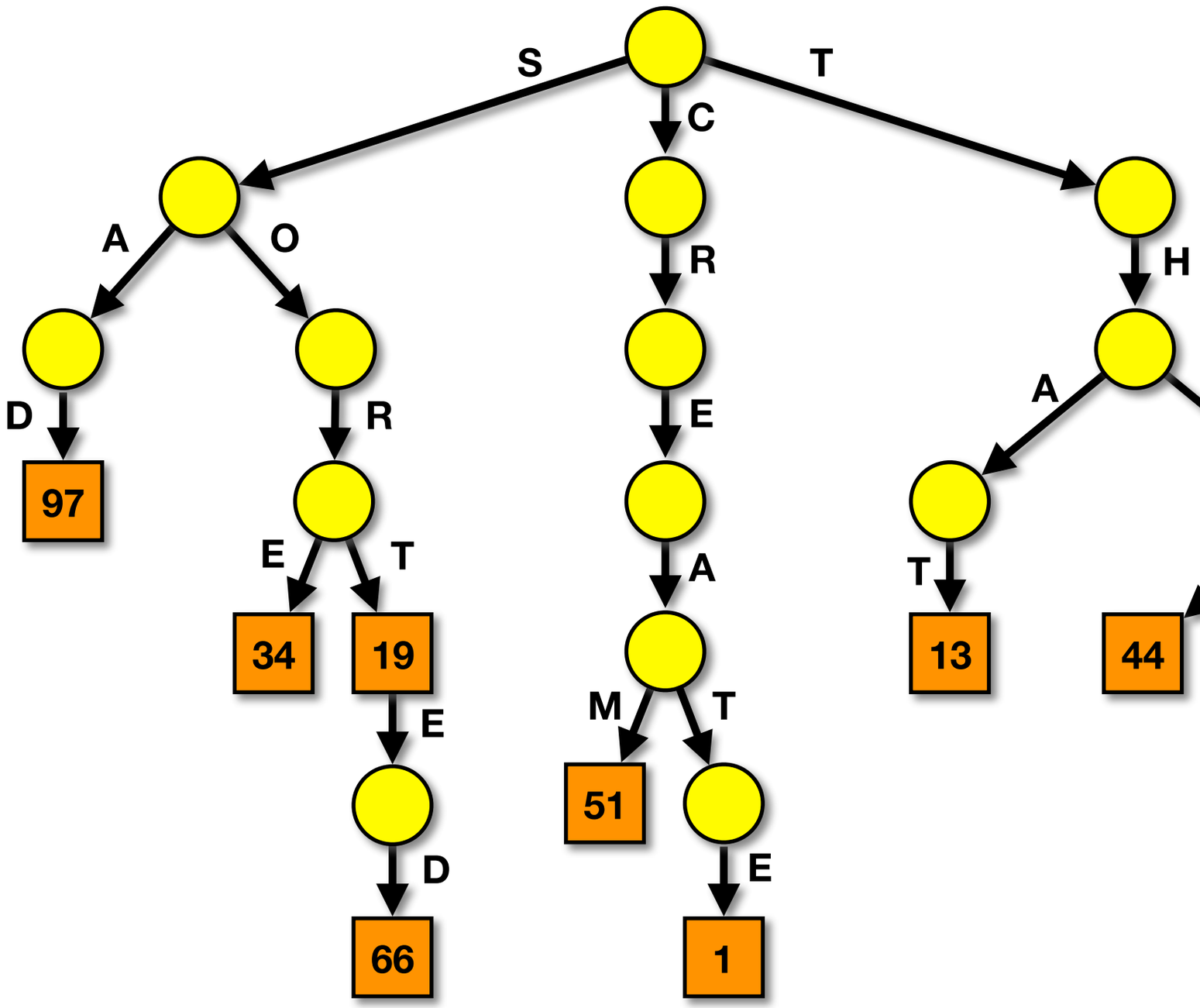
Example of a Trie



Contains the following words:
SAD
SORE
SORT
SORTED
CREAM
CREATE
THAT
THIS
THIN
THICK

= an endpoint of a key entry

Example of a Trie



Contains the following key value pairs:
[SAD, 97]
[SORE, 34]
[SORT, 19]
[SORTED, 66]
[CREAM, 51]
[CREATE, 1]
[THAT, 13]
[THIS, 44]
[THIN, 14]
[THICK, 12]

Introduction to Tries

- **Useful when many entries begin with the same sequence of characters/values**
  - Perform autocomplete as you type
  - Longest prefix matching for IP address lookup
- **Storing information in a Trie**
  - All leaf nodes are endpoints of key entries
    - Each path from the root to a leaf node corresponds to a word in the Trie data structure
  - Internal nodes can also be endpoints
  - All descendants of a node have the same prefix
  - Nodes can be marked as endpoints with a simple boolean flag
  - If storing non-case-sensitive character strings, each node has at most 26 children (most nodes will have far fewer children)

Insertion Into a Trie

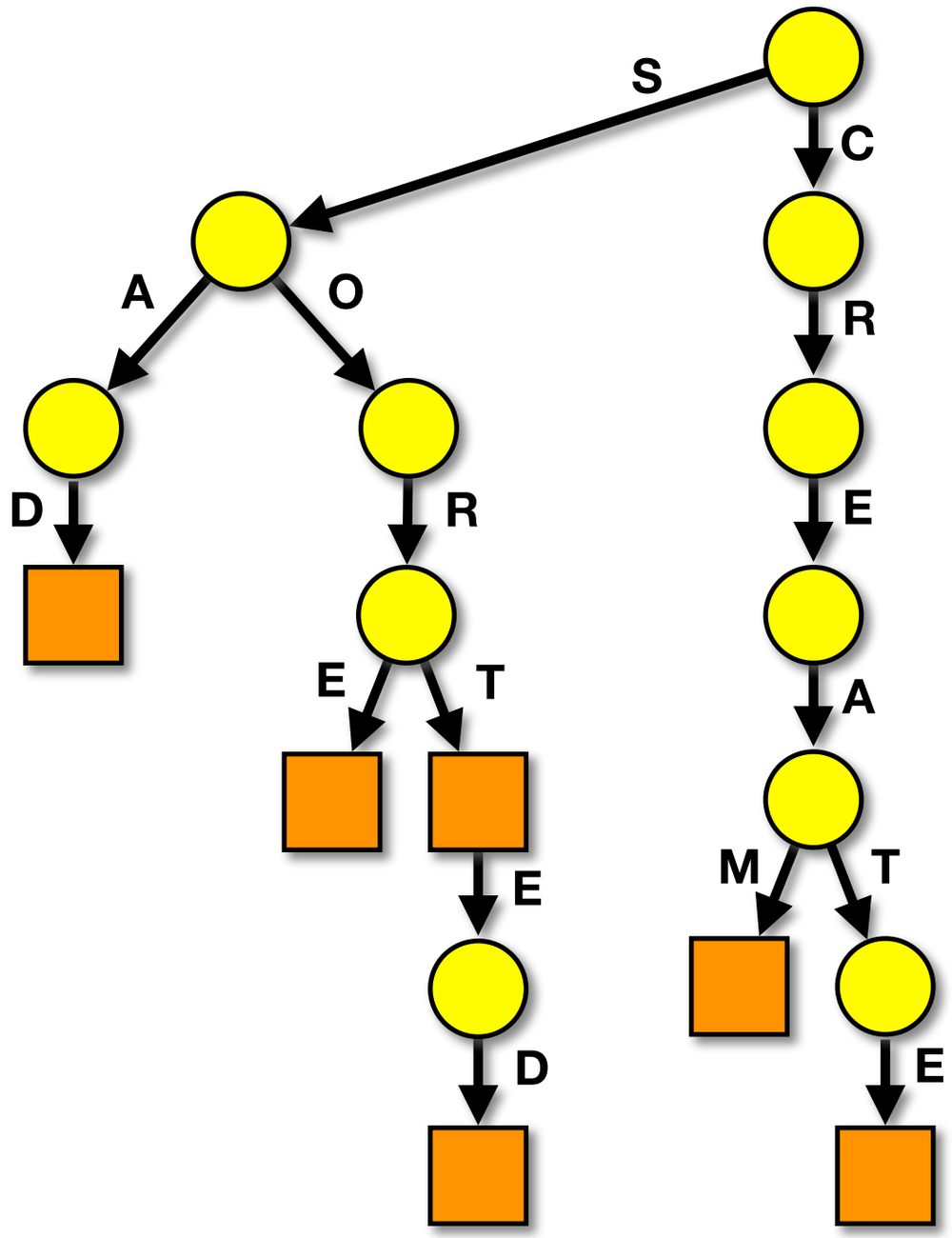
- **Given a key value (of characters):**
  - Start at the root node of the Trie (i.e. currentNode = rootNode)
  - For each character in the key:
    - Check to see if a child link exists for the current character at the currentNode
    - If a child link does exist for current character value v:
      (1) Follow the link to the childNode (i.e. set currentNode = $childNode_v$)
      (2) Go to next character in the key
    - If a link does not exist for the current character value v:
      (3) Create a new childNode and attach it to the currentNode with a link that corresponds to the current character value
      (4) Set the currentNode to the newly created childNode (i.e. currentNode = $childNode_v$)
      (5) Go to the next character in the key
    - If no more characters left in the key, then set currentNode as an endPoint

Searching a Trie

- **Given a key value (of characters):**
  - Start at the root node of the Trie (i.e. currentNode = rootNode)
  - For each character in the key:
    - Check to see if a child link exists for the current character at the currentNode
    - If a link does not exist for the current character value, then the key does not exist in the Trie
    - If a link does exist for the current character value v, then follow the link to the childNode (i.e. set currentNode = $childNode_v$)
  - When no more characters in the key, check to see if currentNode is an endPoint. If TRUE, then key exists in Trie, otherwise key does not exist in the Trie

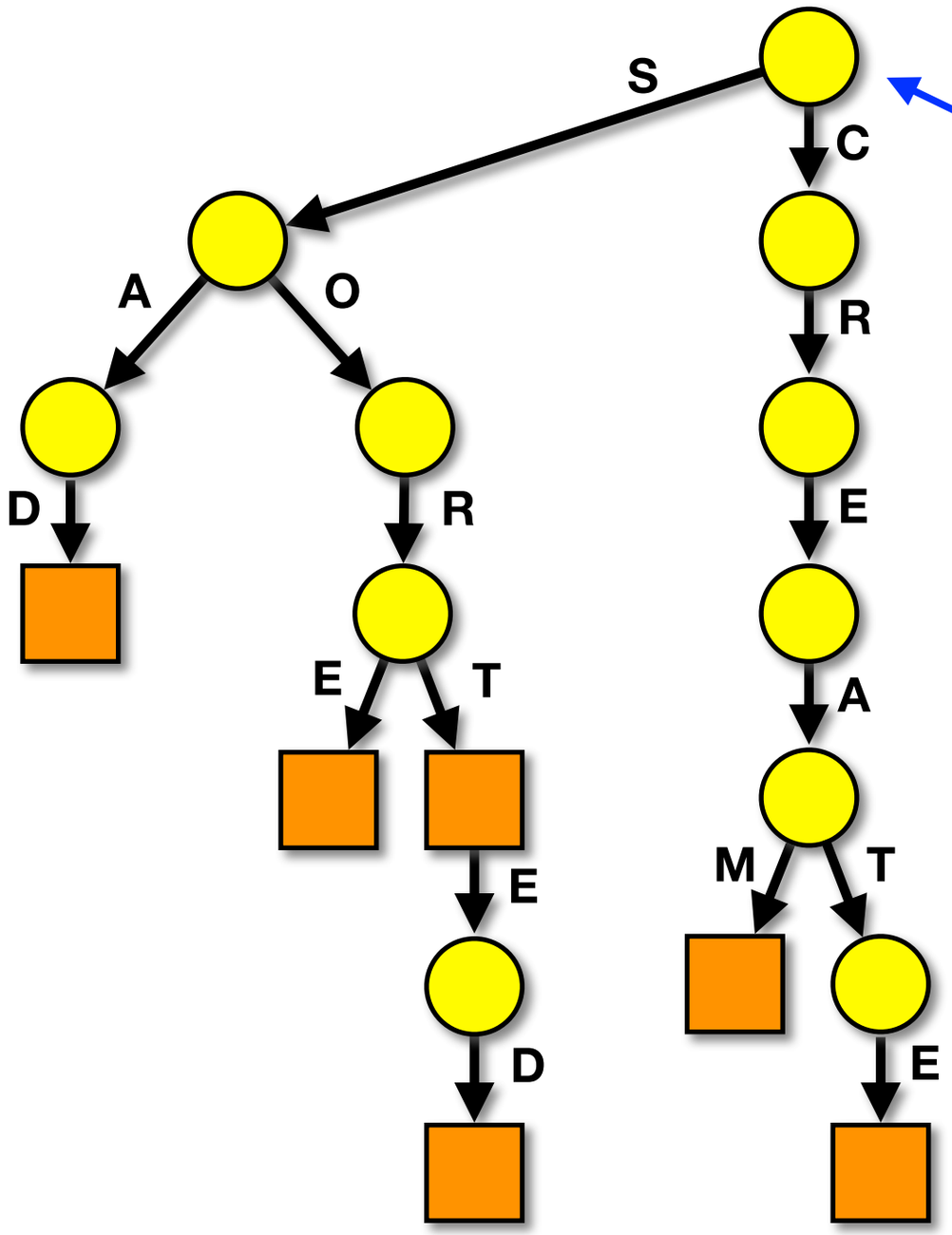
Example of Inserting Into a Trie

Inserting the key ‘DOG’



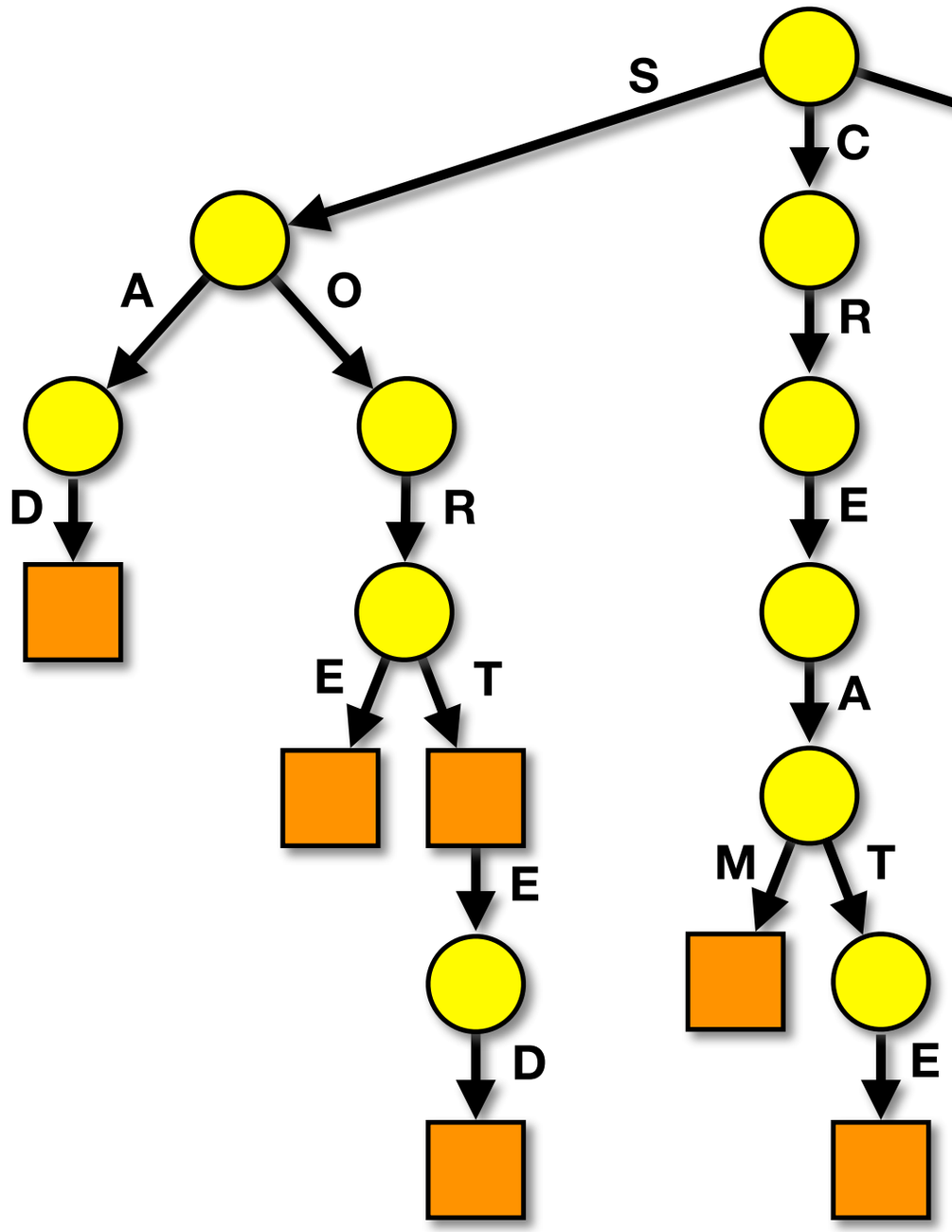
Example of Inserting Into a Trie

Inserting the key ‘DOG’



Current character is ‘D’, and there is no ‘D’ link coming out of the root, so make a new node for ‘D’

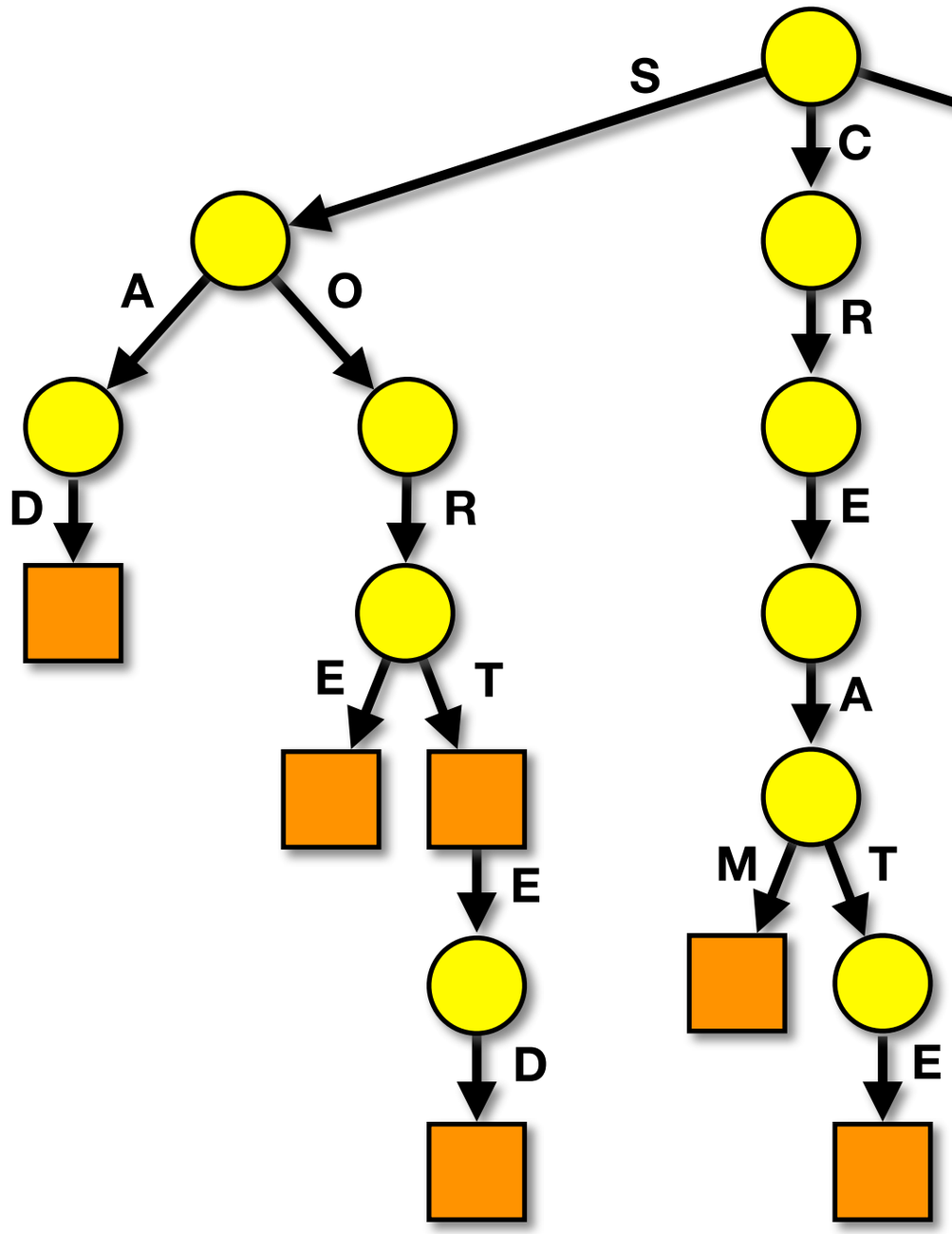
Example of Inserting Into a Trie



Inserting the key ‘DOG’

Current character is ‘O’, and there is no ‘O’ link coming out of the current node, so make a new node for ‘O’

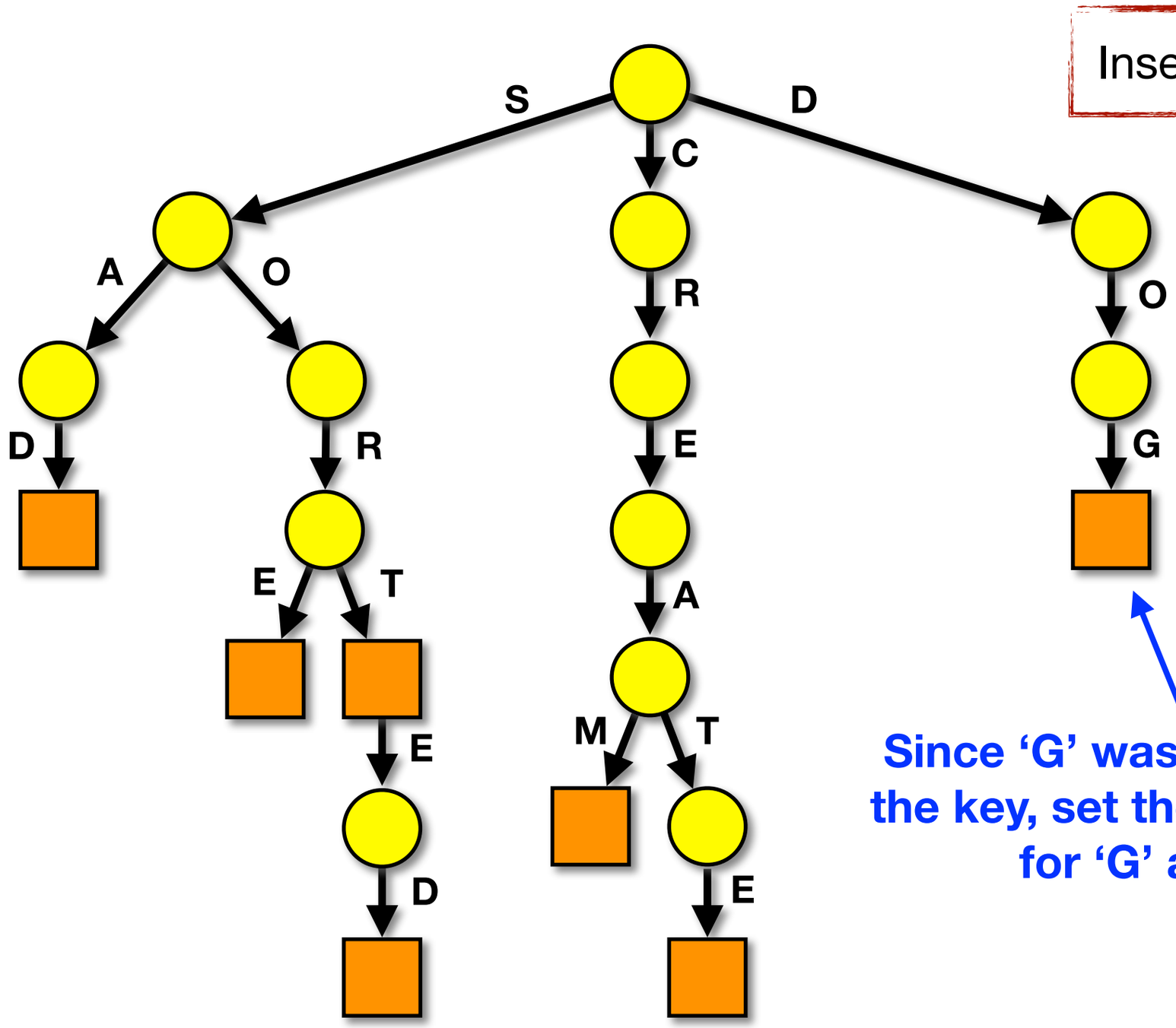
Example of Inserting Into a Trie



Inserting the key ‘DOG’

Current character is ‘G’, and there is no ‘G’ link coming out of the current node, so make a new node for ‘G’

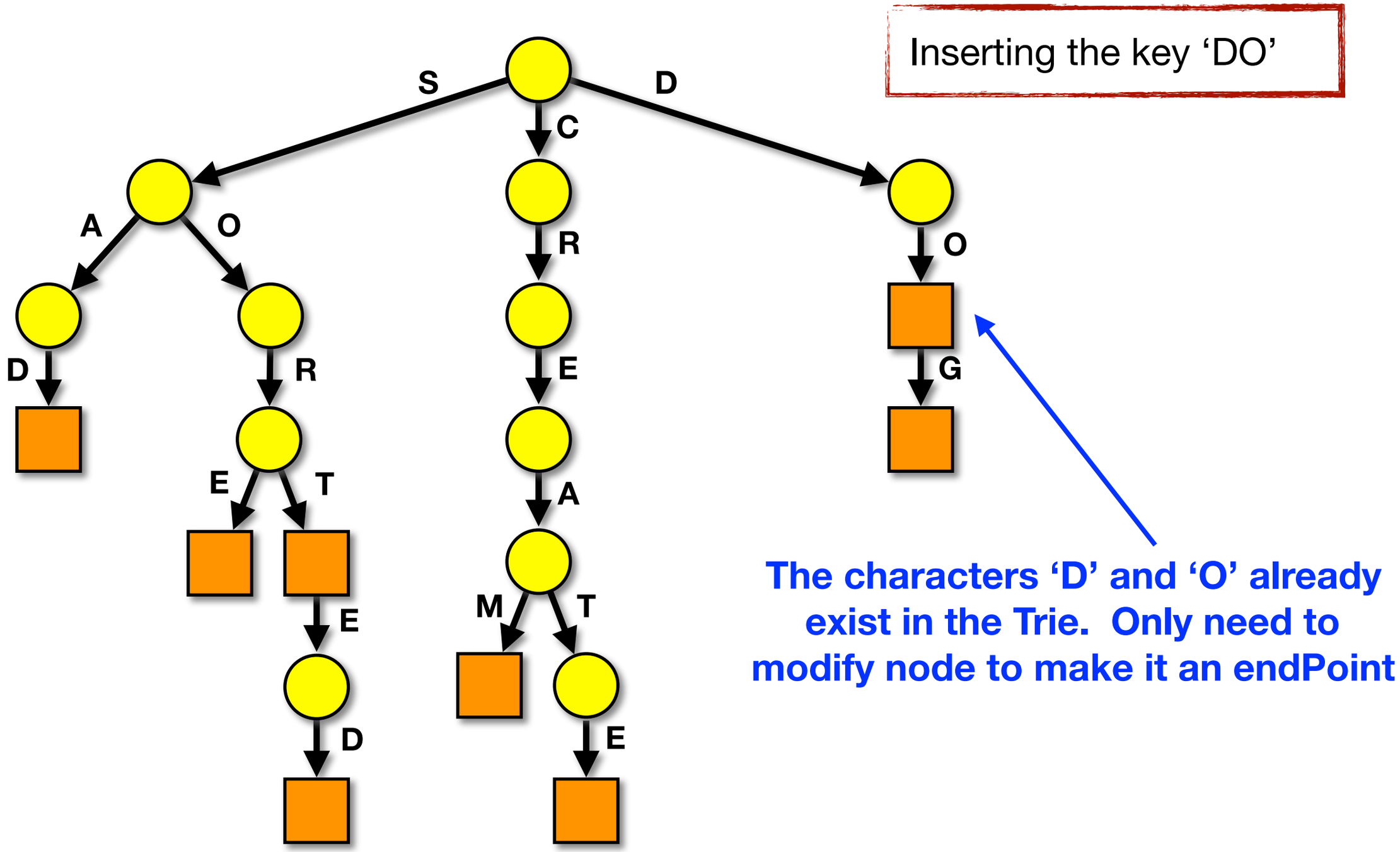
Example of Inserting Into a Trie



Inserting the key ‘DOG’

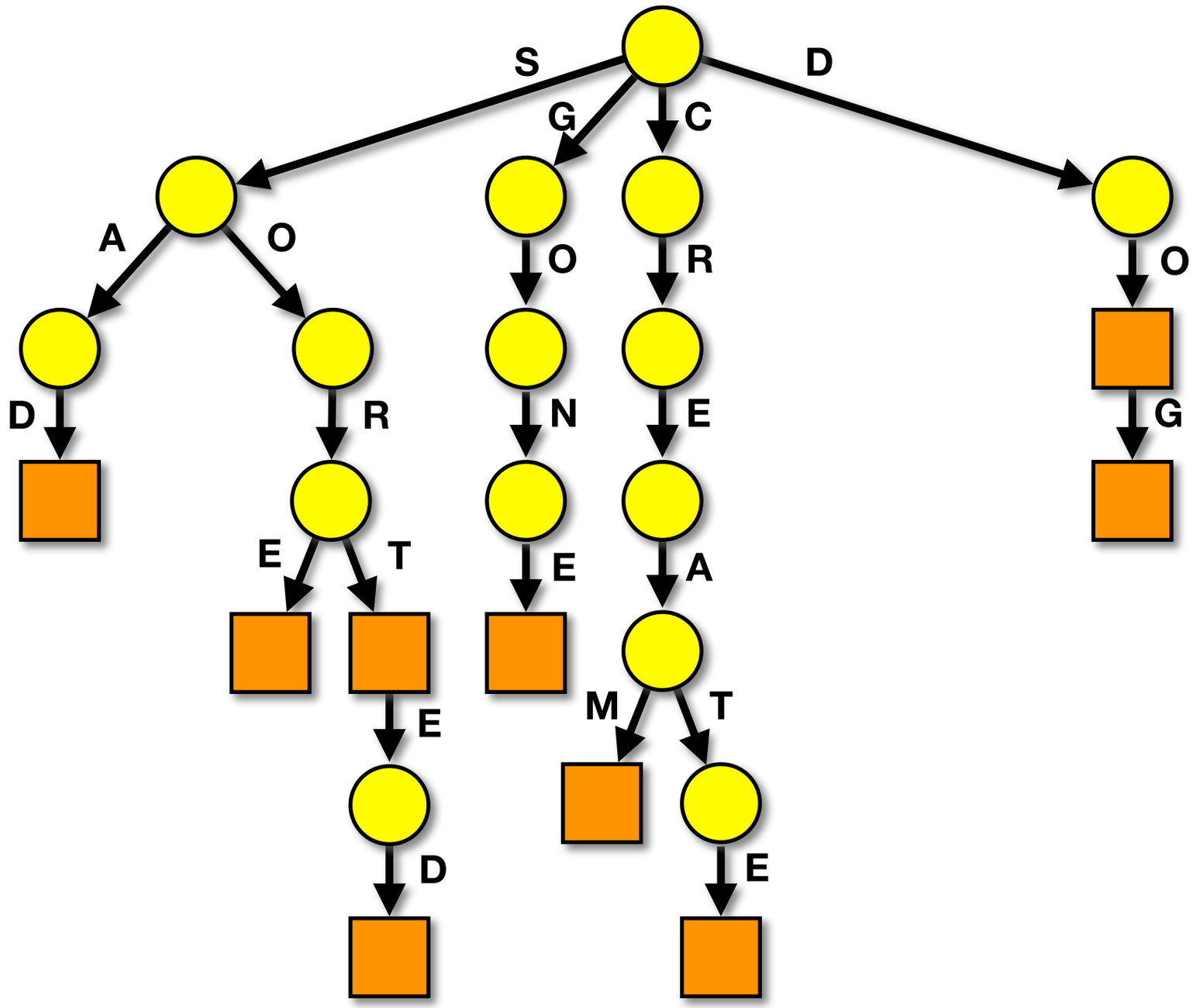
Since ‘G’ was the last character in the key, set the newly created node for ‘G’ as an endPoint

Example of Inserting Into a Trie



Deleting From a Trie

- **What must be done to delete a key from a Trie?**
  - Remove GONE?
  - Remove SAD?
  - Remove DO?
  - Remove SORTED?



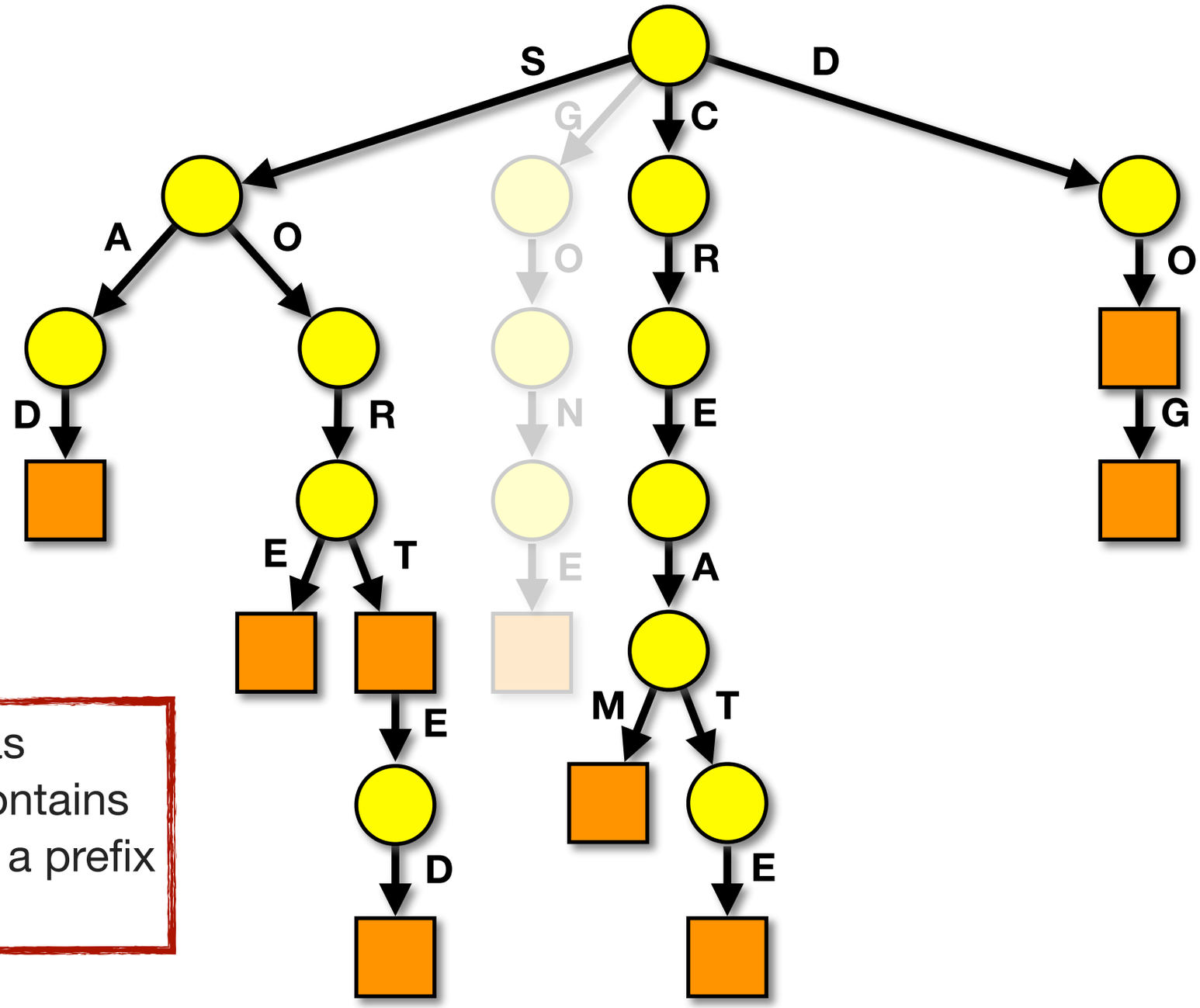
Deleting From a Trie

- **Delete keys in bottom up manner using recursion**
- **Must consider the following five cases:**

1) Key k_D not in the tree
   - Do nothing
2) Key k_D is present as unique key -- no part of k_D contains another shorter key, nor is k_D a prefix of another longer key in trie
   - Delete all the nodes of k_D
3) Key k_D is prefix of another longer key in trie
   - Simply unmark the endPoint of k_D
4) Key k_D has at least one other key as a prefix
   - Delete nodes from end of k_D until reaching endPoint of the longest prefix key of k_D (Yay recursion!)
5) Key k_D shares a common prefix with another key, but neither endPoint is a descendant/ ancestor of the other
   - Delete nodes from end of k_D until reaching a node that contains multiple children

Deleting From a Trie

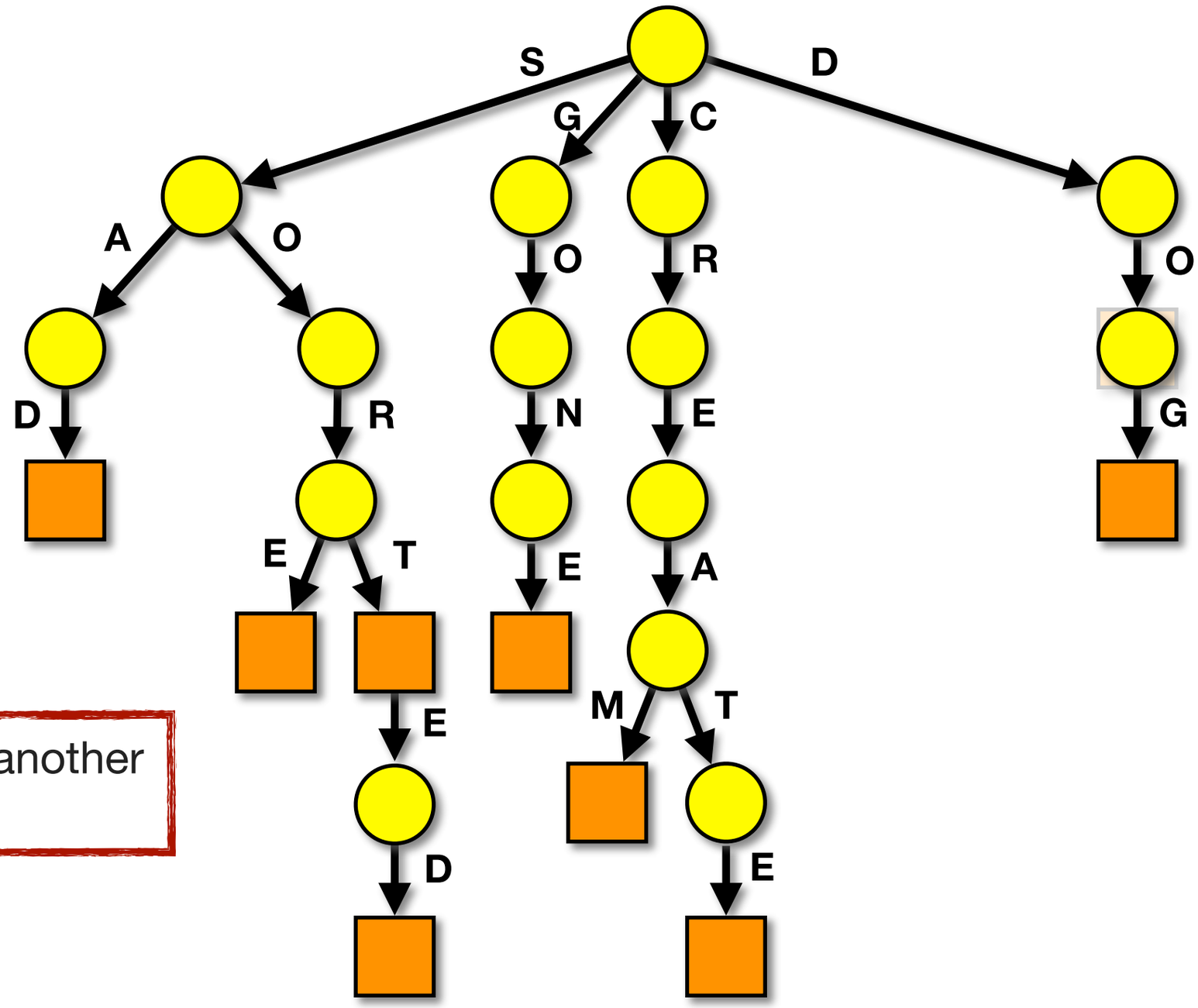
- **What must be done to delete a key from a Trie?**
  - Remove GONE?



Case #2 - Key k_D is present as unique key -- no part of k_D contains another shorter key, nor is k_D a prefix of another longer key in trie

Deleting From a Trie

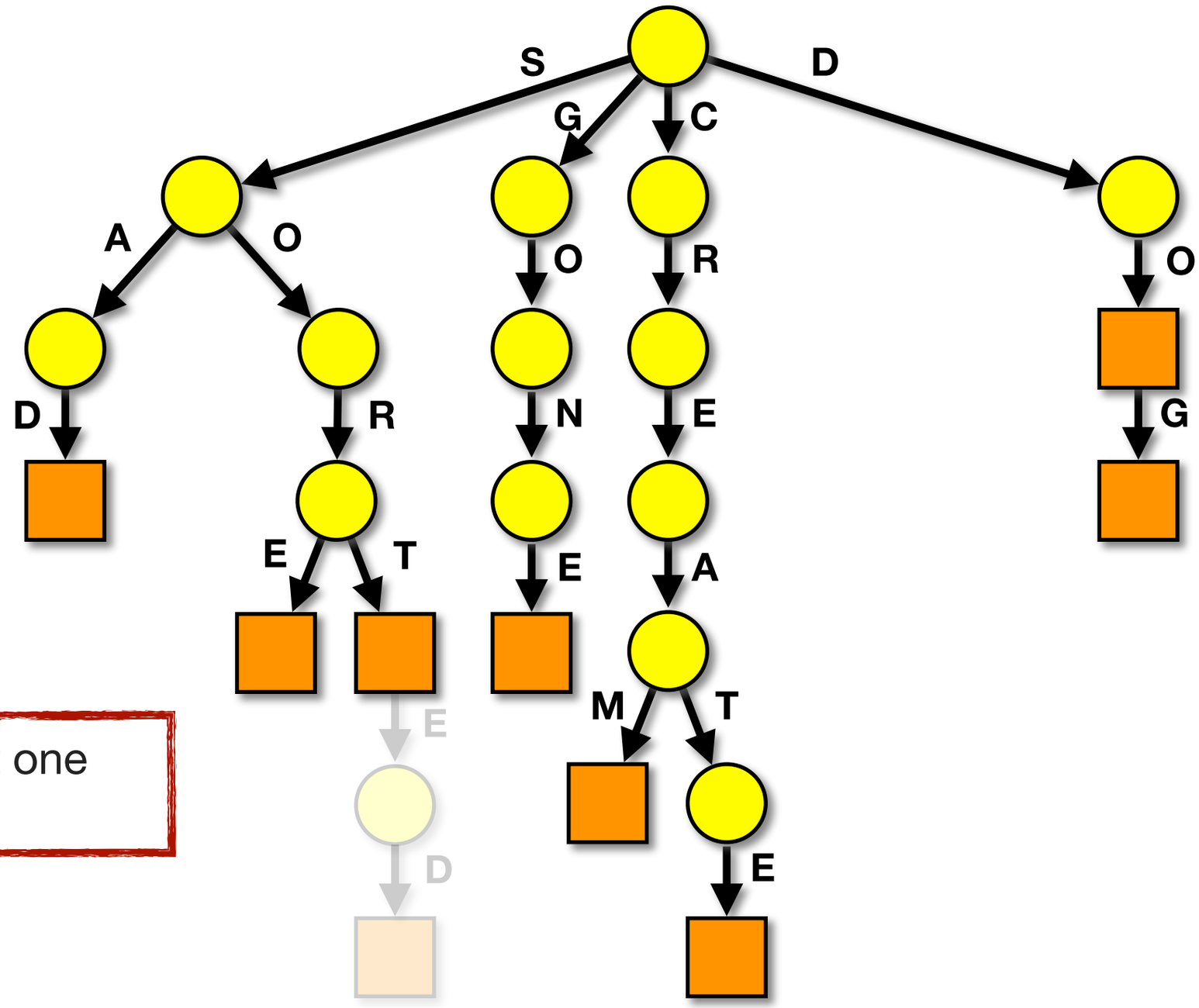
- **What must be done to delete a key from a Trie?**
  - Remove DO?



Case #3 - Key k_D is prefix of another longer key in trie

Deleting From a Trie

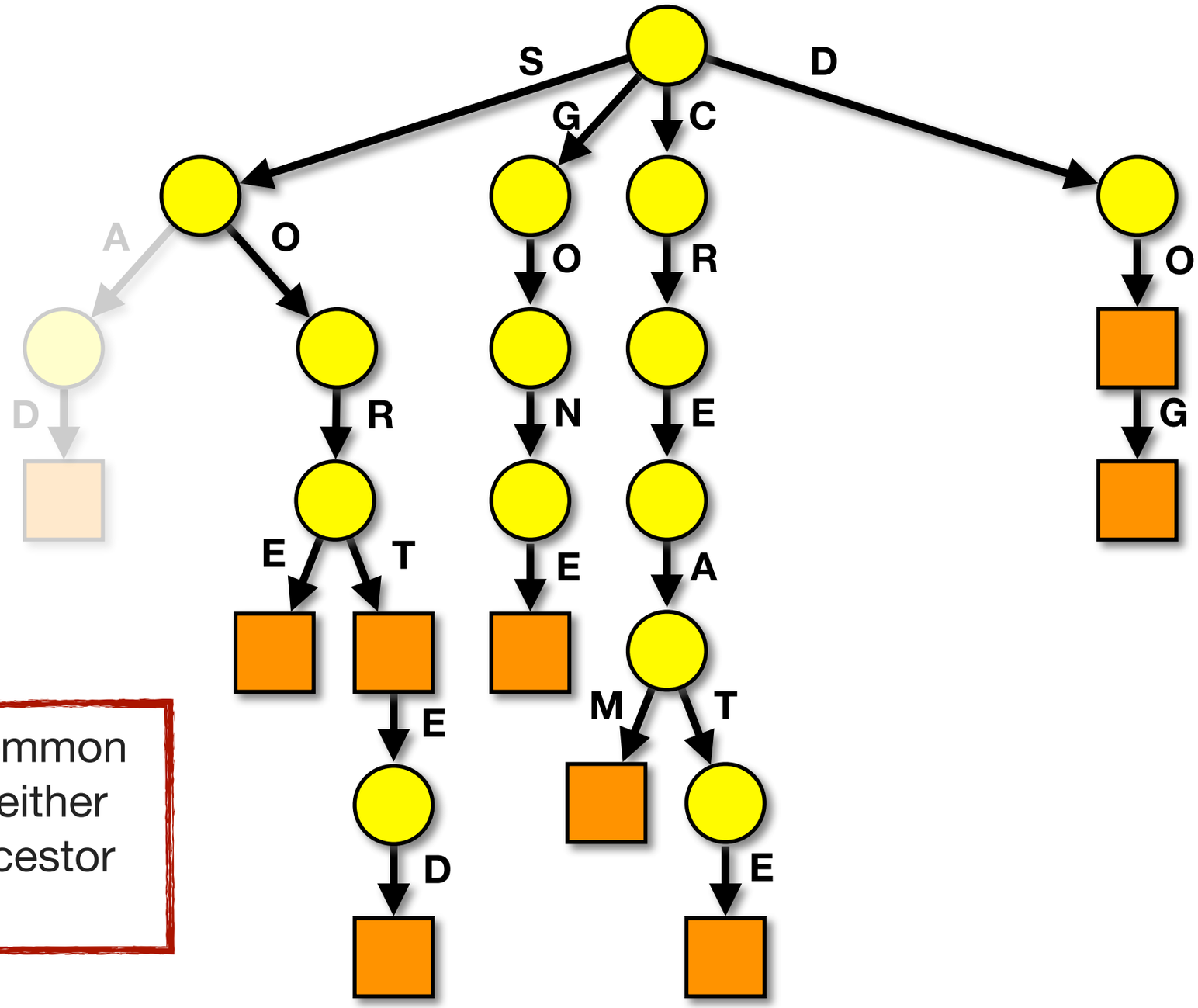
- **What must be done to delete a key from a Trie?**
  - Remove SORTED?



Case #4 - Key k_D has at least one other key as a prefix

Deleting From a Trie

- **What must be done to delete a key from a Trie?**
  - Remove SAD?



Case #5 - Key k_D shares a common prefix with another key, but neither endPoint is a descendant/ancestor of the other