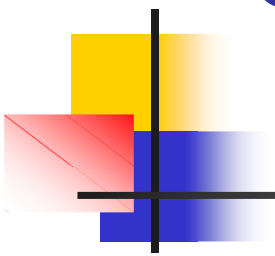
Google Star – Project Presentation



CS8621: Advanced Computer Architecture

Course Instructor : Dr.Ted Pedersen

Team : Fluminense

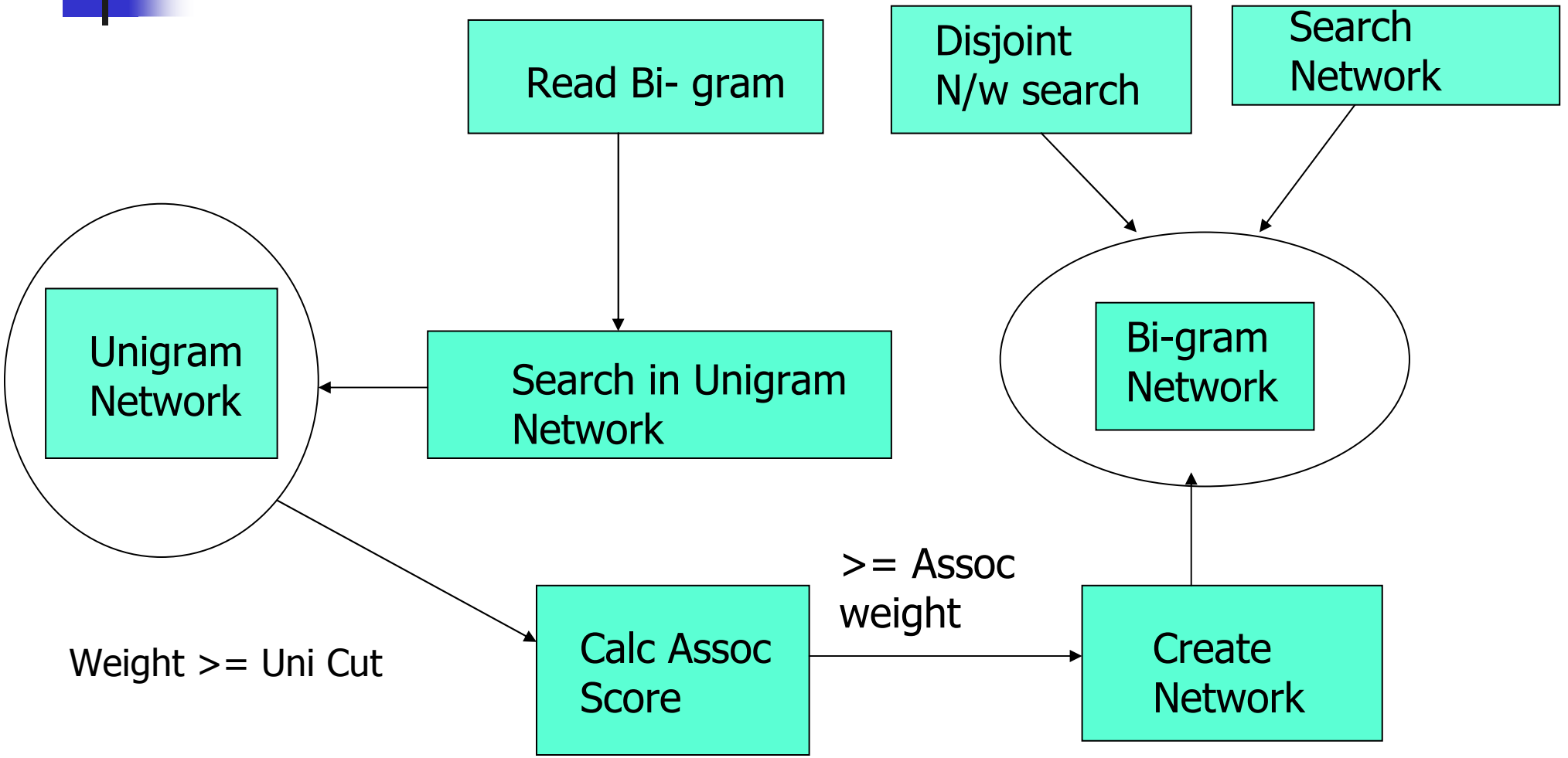
Aneerudh Naik

Ankur Nepalia

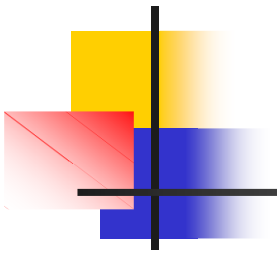
Prafulla Bhalekar

Sarika Mehta

Architecture

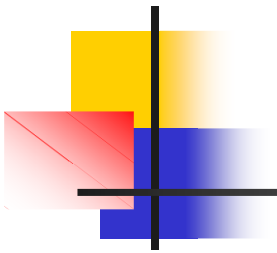


Unigram and Associativity Cut



- Overview :
  - Alpha Stage – to store the network in files.
  - Beta Stage – to handle the unigram cut-off
  - Final Stage – to handle the associativity cut-off and modifications in the unigram network.

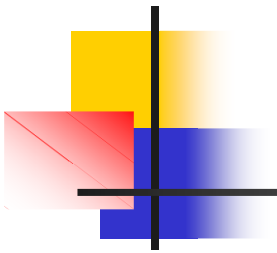
Alpha Stage



- Storing the network in the files
  - Locating and Creating Directories.
  - Creating Files.
  - Traversing the network.
  - Writing to the files.

Note : This has been dropped from the current implementation considering the file I/O cost.

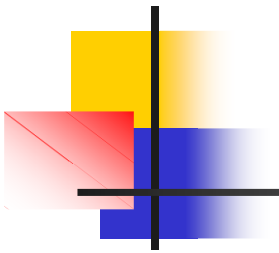
Beta Stage



- Handling unigram cut

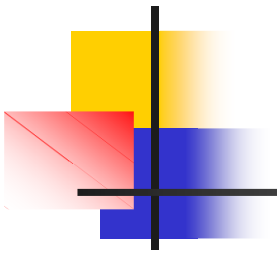
- Creating unigram network on processes 0 and 1(due to memory limitations)
  - Split vocab in two files
  - Create Hash tables on 0 and 1
  - Read and Hash each unigram
  - Attach unigram node to the binary search tree if the weight is >= unigram cut
  - Remove temporary files.

Beta Stage contd...



- When the bigram file is being read, accept each word and pass it on to search function
- Search function : (Tree traversals)
  - Proc 0 :
    - Searches it's own.
    - Sends string to proc1
    - Searches for all other processors.
  - Proc 1:
    - Same as 0
  - Other Processors
    - Just send their unigrams to 0 and 1.
    - Accept the return value and send to the create network.

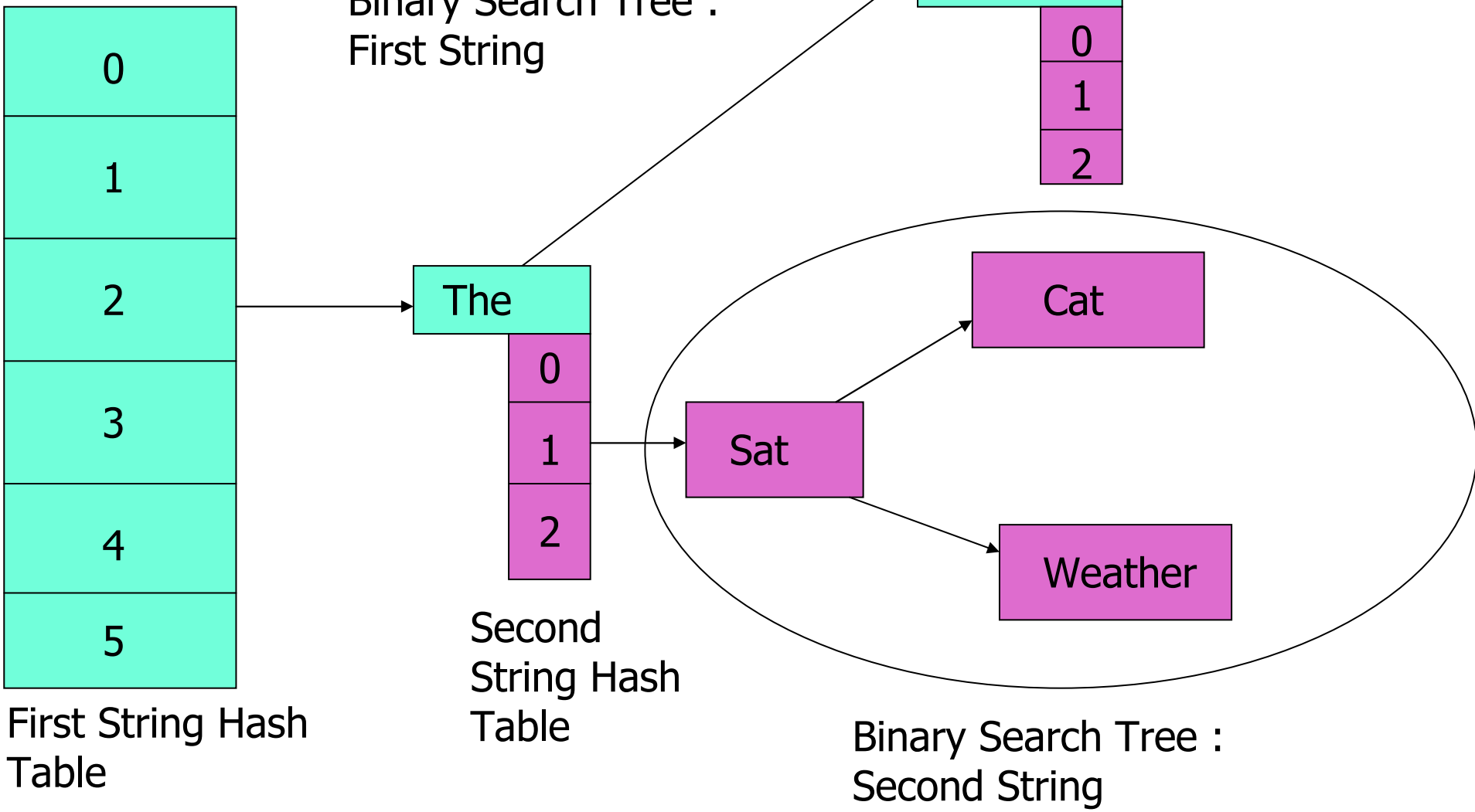
Final Stage



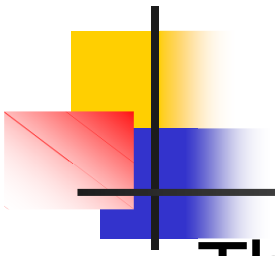
- Unigram network
  - Was modified to hold and return unigram weights
- Handling Associativity cut
  - Accept bigram weight
  - Accept weights of searched unigram
  - Calculate Associativity cut-off

Bi-Gram Network Creation

Data structure

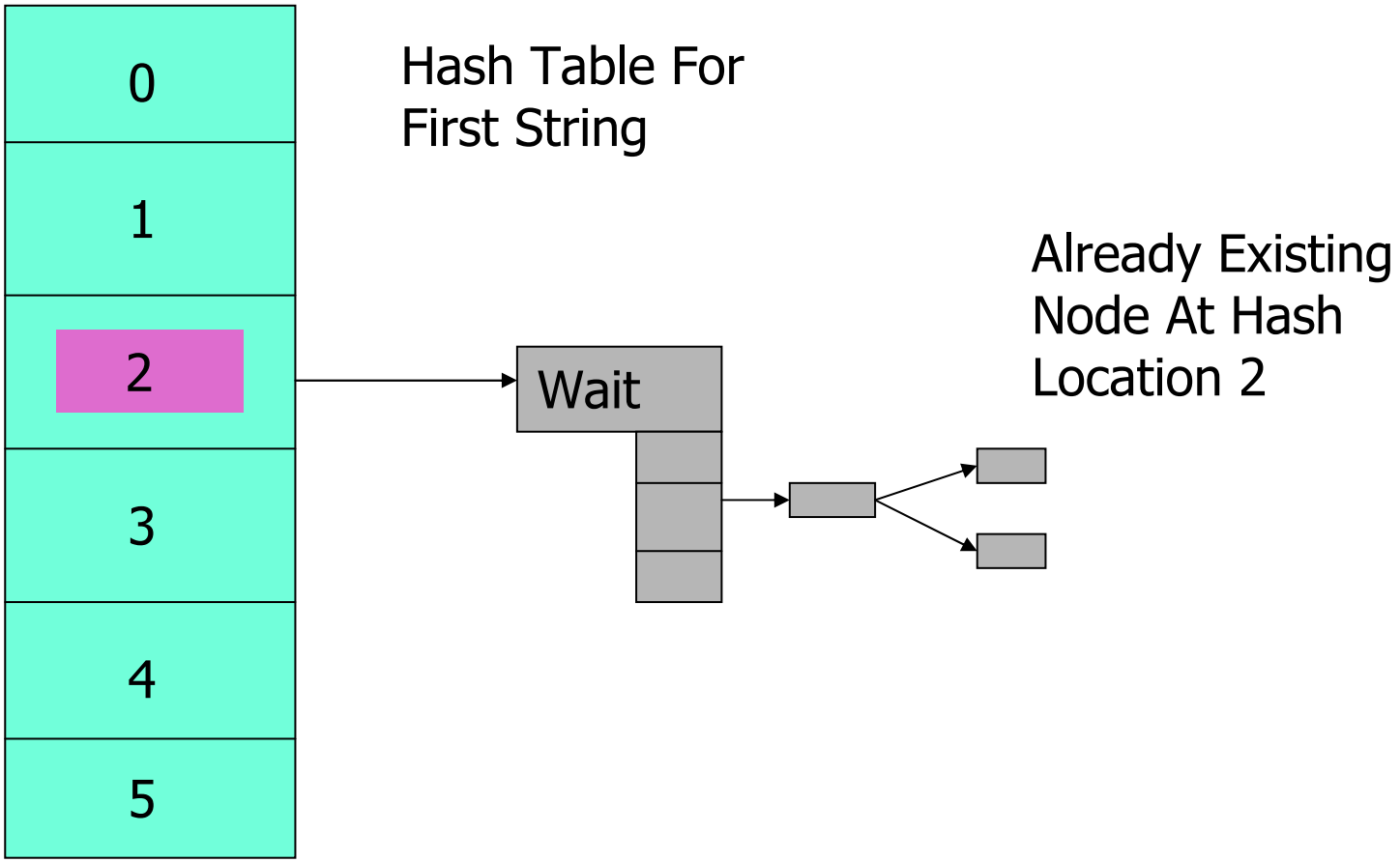


Insertion Method

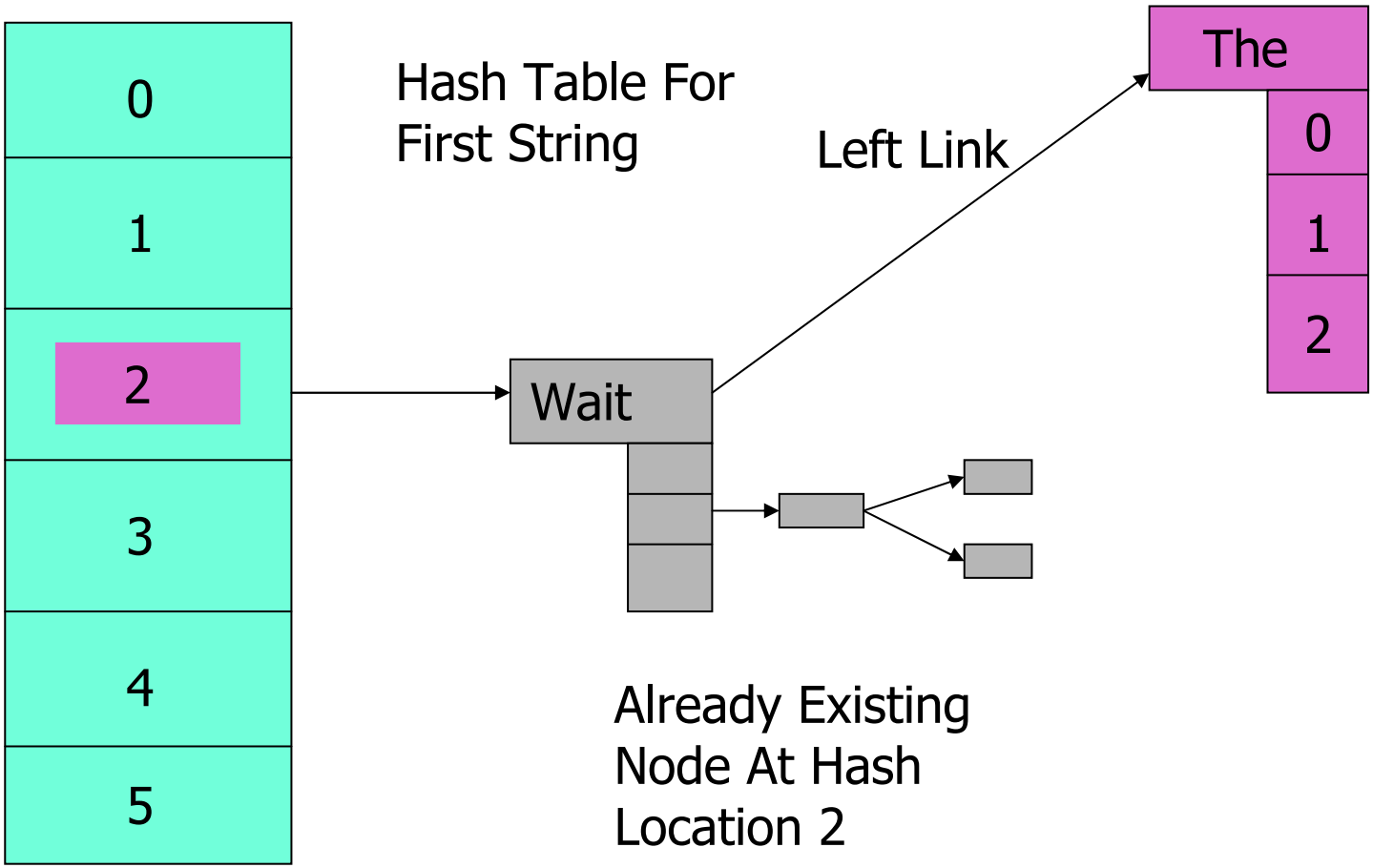


- The Apple 100 (Bi-gram)
  - ‘The -> Apple’ linked as front entry
    - Front Flag : 1
  - ‘Apple -> The’ linked as back entry
    - Back Flag : 1
    - Helps get back connected strings during search
- Short Algorithm :
  - Hash the String
  - Access hash table location
  - Insert into binary search tree

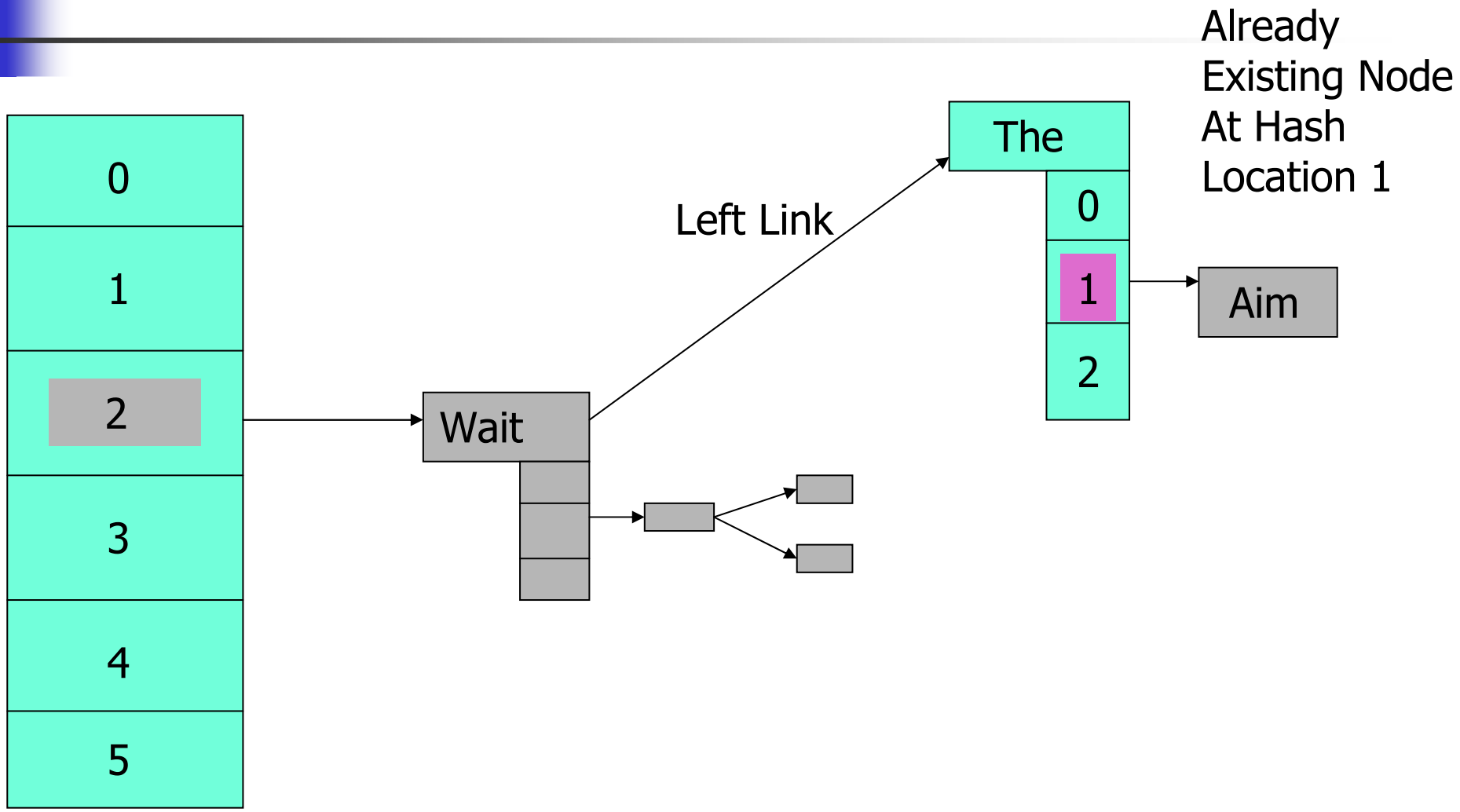
Hash (“The”) = 2


0
1
2
3
4
5
Hash Table For
First String
Wait
Already Existing
Node At Hash
Location 2

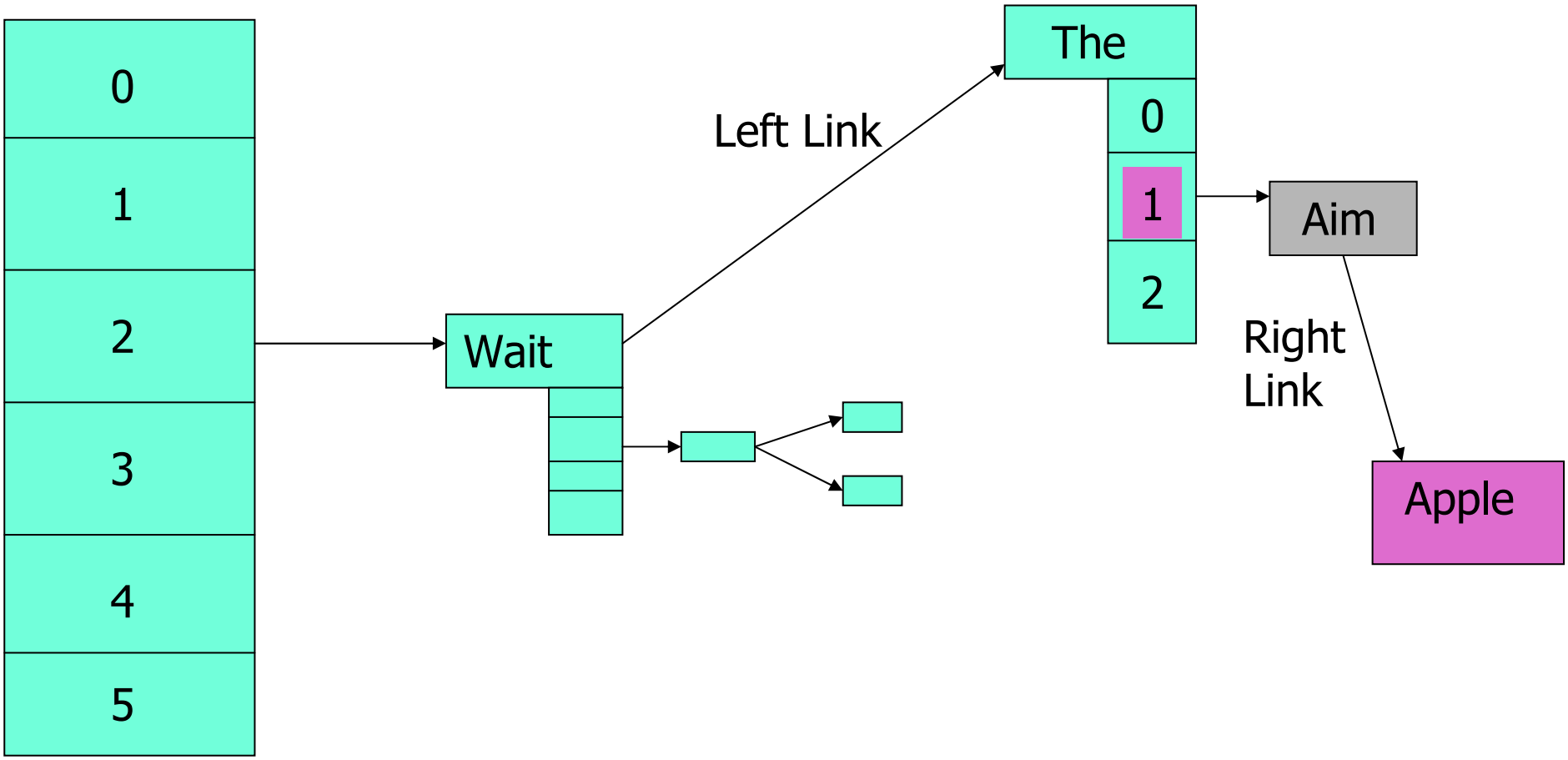
The : Inserted



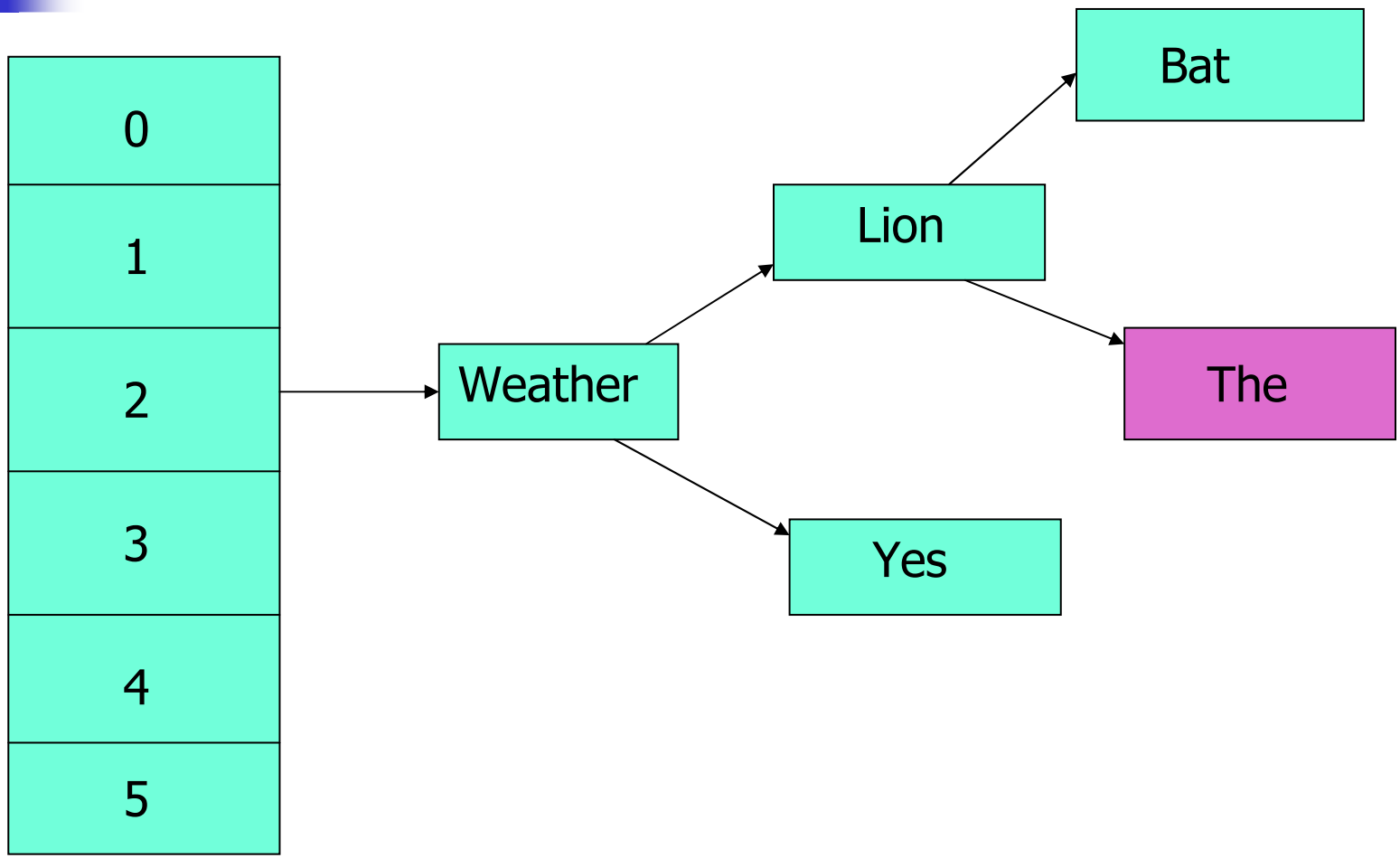
Apple : Hash = 1


Already
Existing Node
At Hash
Location 1
0
1
2
3
4
5
Left Link
The
0
1
2
Aim
Wait

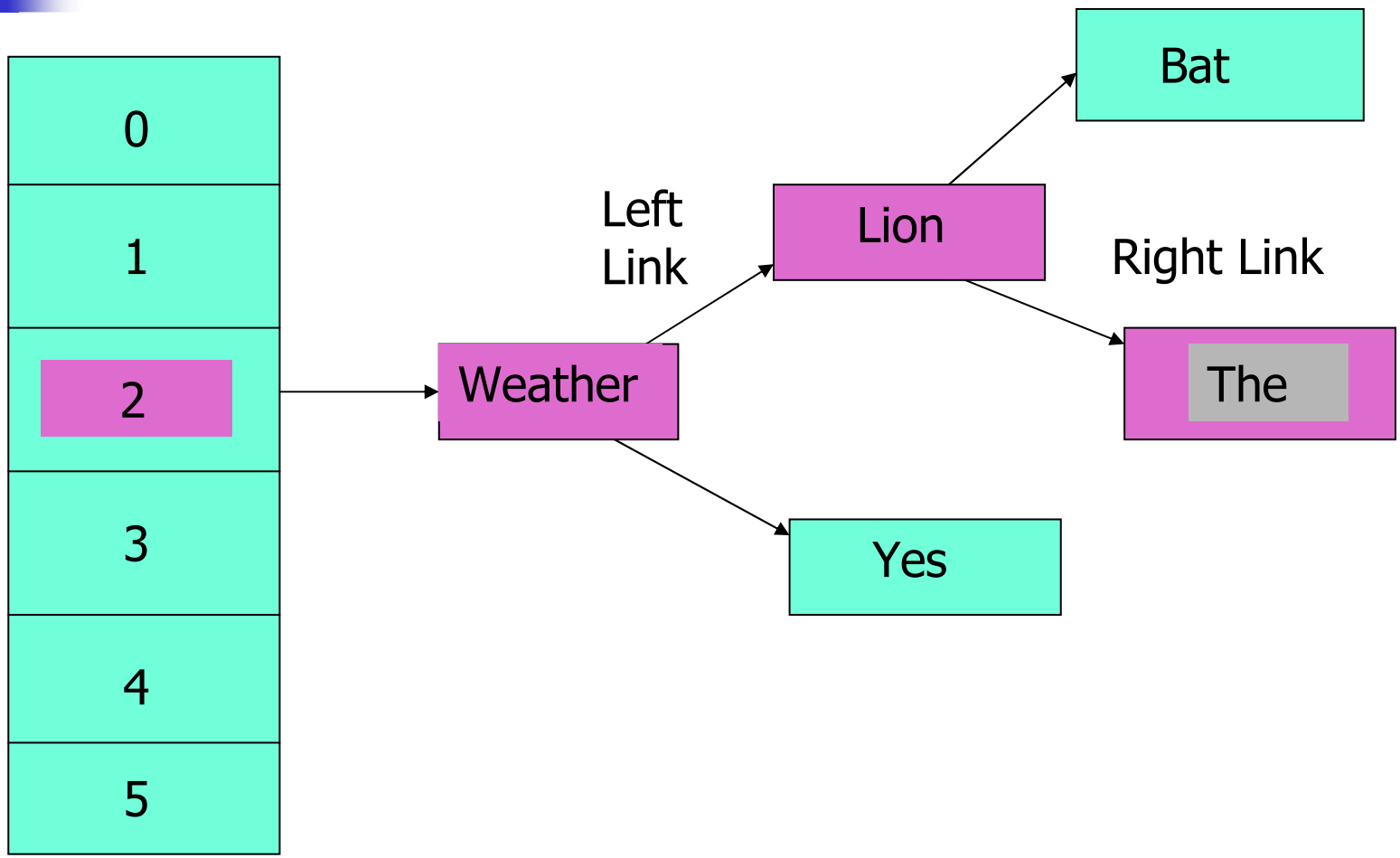
Apple : Inserted



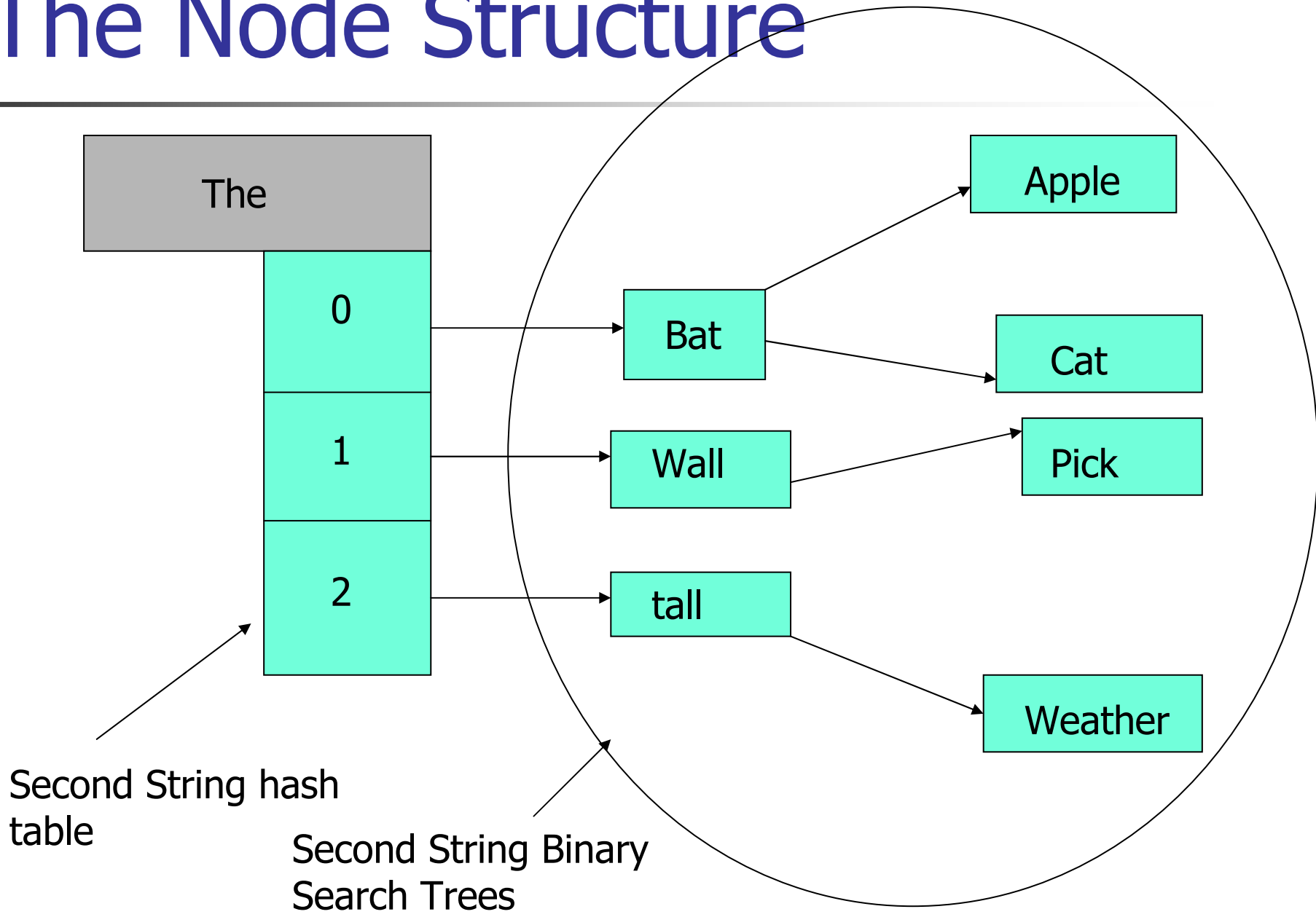
Search Method : 'The'



Search Method : 'The'

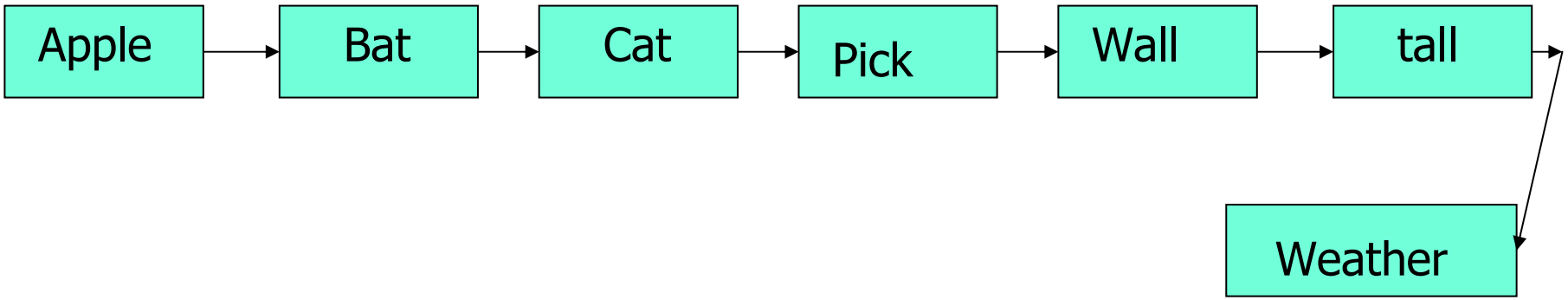


The Node Structure



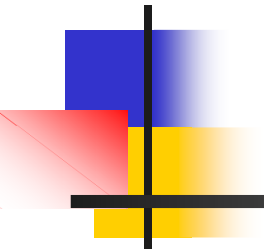
Linked List : Connected Strings

Linked List Created On Process 0



All Connected String Sent to Process 0 and Linked list of all connected strings created

Front Connection : Front Flag = 1 Back Connections : Back Flag = 1



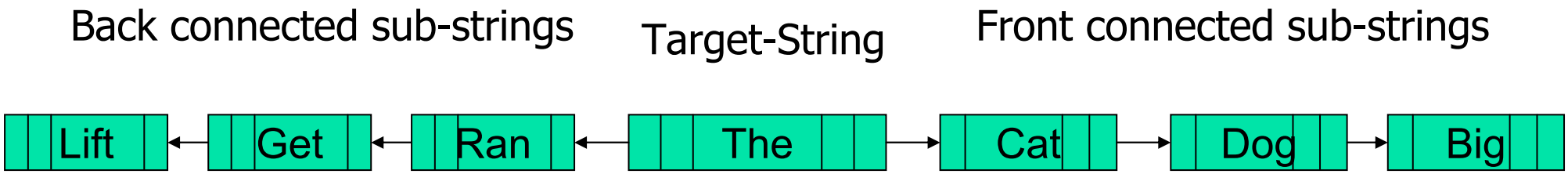
Searching Strings

Creating Target-string network

- Read the target-list file one line at a time.
- Suppose the target string is ‘The’.
- Create a data structure for storing all the connected strings to the target string ‘The’, both in front and back.

Search String: ‘The’ Path Length: 1

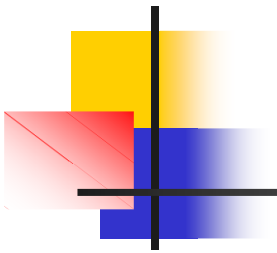
- Passing the target string to the search function.
- Returns back a connected link list of all the sub-strings to the target string



Creating the sub-string network

- Passing the substrings connected to the target string to the search function.
- Front Connected Sub-strings to ‘The’

- Back Connected Sub-strings to ‘The’

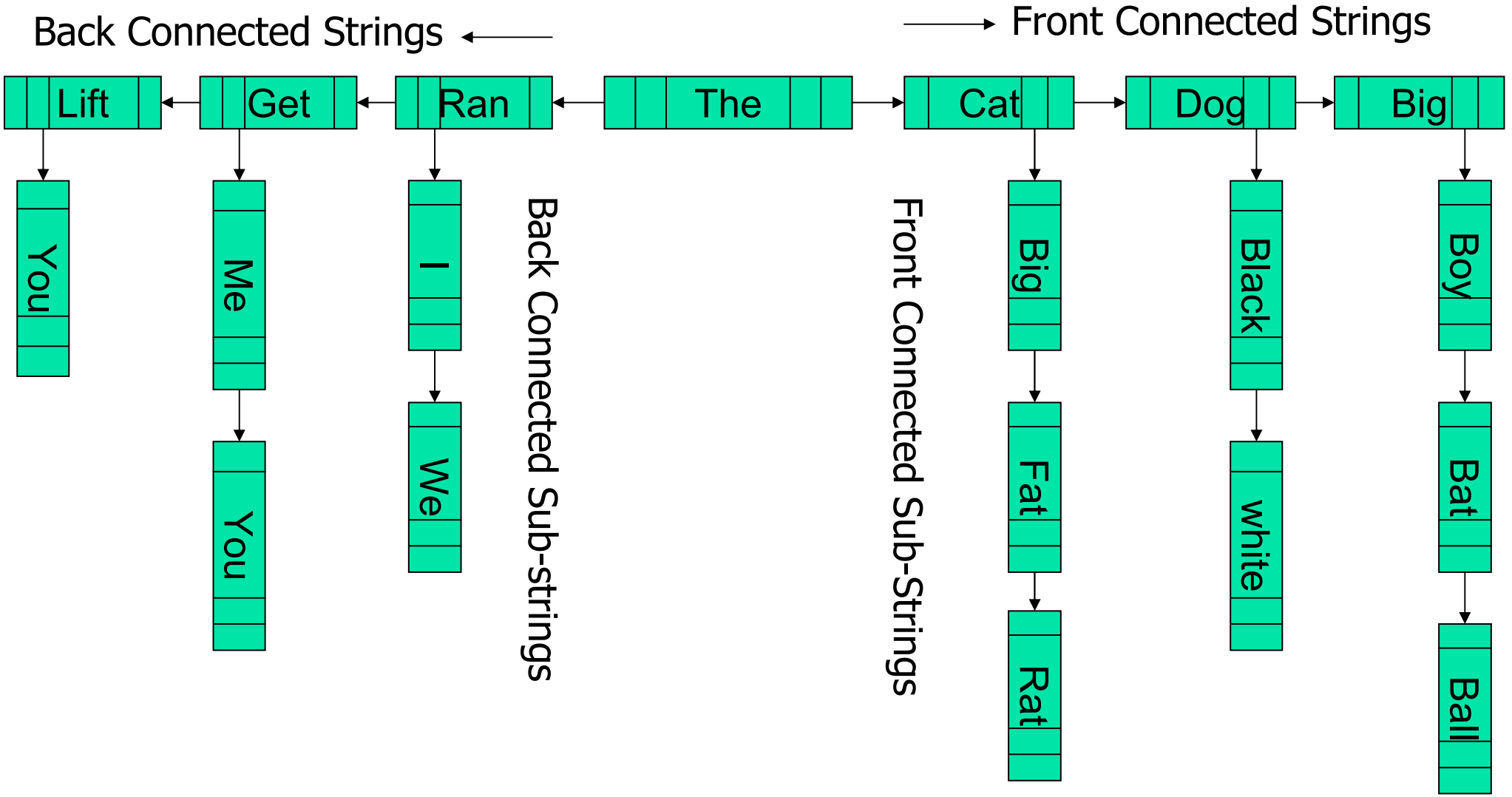


Creating the sub-string network…

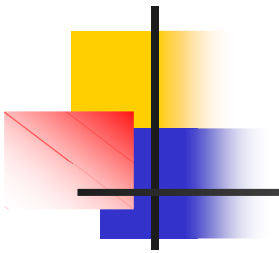
- This again gives back all the connected strings to the substring passed to the search function in the form of a connected link-list.

- This process is done for both the front and back connected substrings to the target string ‘The’.

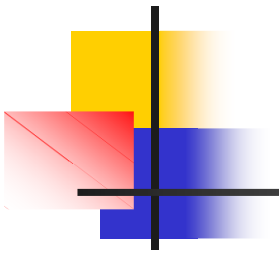
Search String: 'The' Path Length: 2



Display Search String Network



- The target-string network creation is limited by the path-length.
- Recursively displaying the target-string network both in front and back directions.



Hashing Function

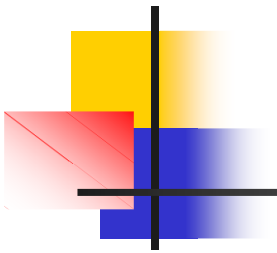
4 Ideal Characteristics:

1) Hash value is fully determined by the data being hashed.

2) Hash function uses all the input data.

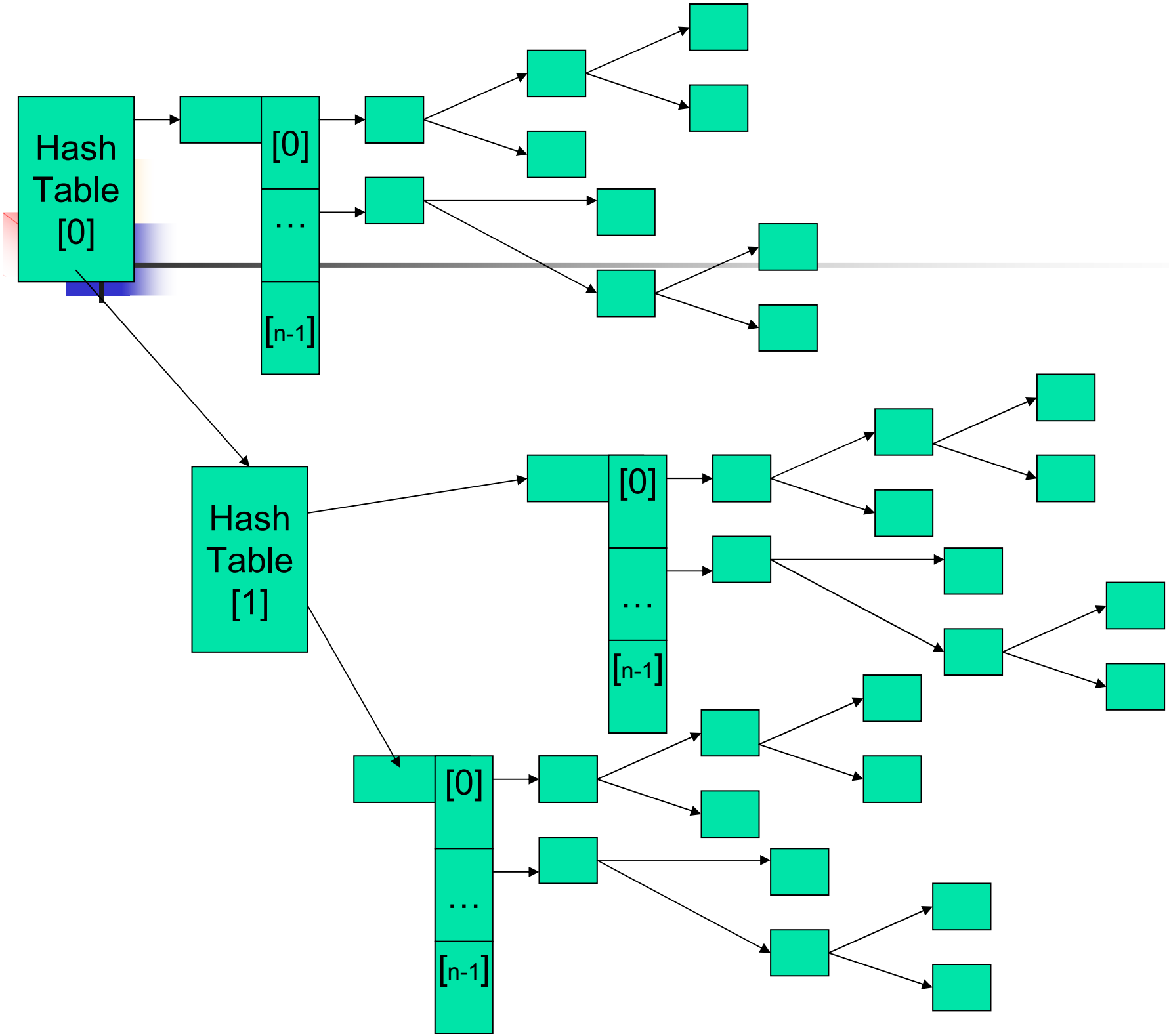
3) Hash function "uniformly" distributes data across entire set of possible hash values.

4) Hash function generates very different hash values for similar strings.

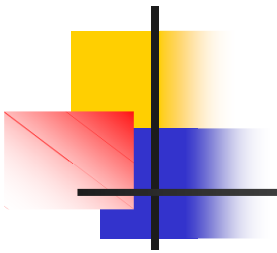


Hashing Function…

- djb2 hash function has been implemented
- Ideally suited to string hashing
- Supports any character.
- Fast.
- Robust.
- Uses XOR function.


Hash Table [0]
[0]
…
[n-1]
Hash Table [1]
[0]
…
[n-1]
[0]
…
[n-1]

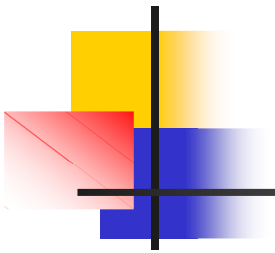
Disjoint Networks



Creation of disjoint networks:

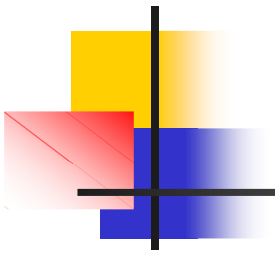
- Each processor creates its own disjoint network files.
- Files created are network and stat files.
- Network files hold actual data (words).
- Stat files hold number of nodes and number of edges.

Disjoint Networks…



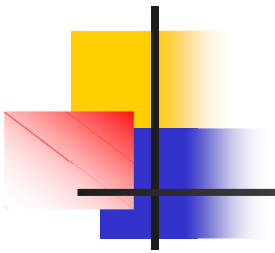
- Search for creating networks starts at the root.
- Root is traversed to find all connected nodes.
- Once, all strings connected to the root are traversed, that entire network is separated or deducted from the unigram cut.
- Process is repeated for each root, in the remaining networks.
- What results is the set of disjoint networks.

Disjoint Networks…



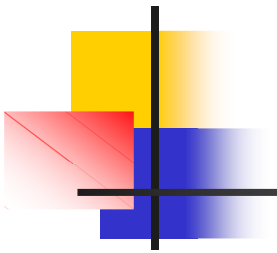
- Each network file holding nodes, from each processor, is compared to every other processor's network file.
- Nodes are inserted into arrays and compared.
- If any string is found common among them, then total network count is reduced.

Disjoint Networks…



- This means those 2 networks are, in fact, joint.
- Then, nodes in the files with a common word are added.
- Count of common words is subtracted.
- Edges are added, as the joint network contains edges from both the networks.

Implementation Example



- Network files named according to following convention:

  process 0: 000netw000, 000netw001,…
  process 1: 001netw000, 001netw001,…

  …

  process *i*: 00 *i* netw000, 00 *i* netw001,…

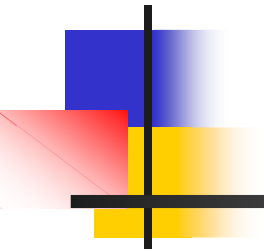
Implementation Example…

- Stat files named similarly,

process 0: 000stat000, 000stat001,…
process 1: 001stat000, 001stat001,…

…

process *i*: 00 *i* stat000, 00 *i* stat001,…



Acknowledgments

Dr. Ted Pedersen

Thank You…