PVG's College of Engineering & S. S. Dhamankar
Institute of Management, Nashik

206, Behind Reliance Petrol Pump, Dindorircad, Mhasrul, Nashik-422004
Phone: 0253-6480000 / 36 /44 Toll Free Number: 18002665330
Website: https://pvgcoenashik.org Email: admission@pvgcoenashik.org

Approved by AICTE, New Delhi, DTE, Mumbai and
Affiliated to Savitribai Phule Pune University, Pune.

DTE Code: EN5330

"HASHING"

Prepared By

Prof. Anand N. Gharu

(Assistant Professor)

PVGCOE Computer Dept.

CLASS : SE COMPUTER 2019

SUBJECT : DSA (SEM-II)

11 JAN 2022

UNIT : I

SYLLABUS

Savitribai Phule Pune University

Second Year of Engineering (2019 Course)

210252: Data Structures and Algorithms

Teaching Scheme	Credit Scheme	Examination Scheme and Marks
Lecture: 03 Hours/Week	03	**Mid_Semester(TH):** 30 Marks **End_Semester(TH):** 70 Marks

Prerequisite Courses:	110005: Programming and Problem Solving 210242: Fundamentals of Data Structures
Companion Course:	210257: Data Structures and Algorithms Laboratory

SYLLABUS

Hash Table- Concepts-hash table, hash function, basic operations, bucket, collision, probe, synonym, overflow, open hashing, closed hashing, perfect hash function, load density, full table, load factor, rehashing, issues in hashing, **hash functions-** properties of good hash function, division, multiplication, extraction, mid-square, folding and universal, **Collision resolution strategies-** open addressing and chaining, Hash table overflow- open addressing and chaining, extendible hashing, closed addressing and separate chaining.

Skip List- representation, searching and operations- insertion, removal

INTRODUCTION

CONTENTS :-

Sample videos :

1. Preprocessor :

https://youtu.be/JZkPEl8JjZo?list=PLhb7SOmGNUc6Fg7zmBOOS3yN2AG0JiREp

2. Compiler execution stages :

https://youtu.be/cJDRShqtTbk?list=PLhb7SOmGNUc6Fg7zmBOOS3yN2AG0JiREp

Introduction

Computer : A programmable device that can store, retrieve, and process data.(Combination of H/w & S/w)

Hardware : things which we can touch.

Software : things which we cannt touch.(Can only see)

Programming: A programming language is a set of commands, instructions, and other syntax use to create a software program.

Data : Information in a form a computer can use

Information : Any knowledge that can be communicated

Introduction

Data type : The specification of how information is represented in the computer as data and the set of operations that can be applied to it Computer

Computer program : Data type specifications and instructions for carrying out operations that are used by a computer to solve a problem

Introduction

Machine language : The language, made up of binarycoded instructions, that is used directly by the computer

Assembly language : A low-level programming language in which a mnemonic is used to represent each of the machine language instructions for a particular computer

Assembler : A program that translates an assembly language program into machine code

Introduction

Compiler : A program that translates a program written in a high-level language into machine code

Source code : Data type specifications and instructions written in a high-level programming language

Object code : A machine language version of source code.

UNIT-I

HASHING

INTRODUCTION

1. **Hashing** is finding an address where the data is to be stored as well as located using a key with the help of the algorithmic function.

2. **Hashing** is a method of directly computing the address of the record with the help of a key by using a suitable mathematical function called the hash function

3. **A hash table** is an array-based structure used to store <key, information> pairs

INTRODUCTION

3. Hash Table: A hash table is a data structure that stores records in an array, called a hash table. A Hash table can be used for quick insertion and searching.

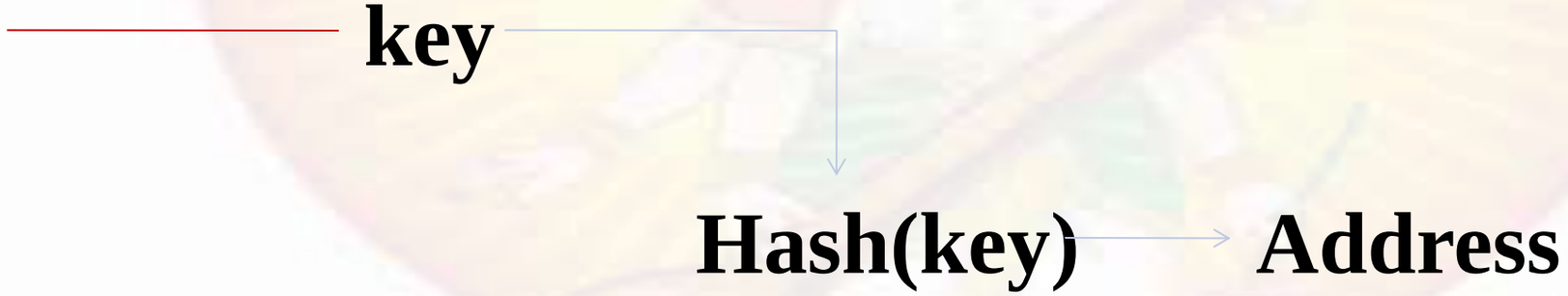


Fig 11.1:Hashing Concept

INTRODUCTION

- The resulting address is used as the basis for storing and retrieving records and this address is called as home address of the record

- For array to store a record in a hash table, hash function is applied to the key of the record being stored, returning an index within the range of the hash table

- The item is then stored in the table of that index position

HASH TABLE

- **Hash table** is one of the most important data structures that uses a special function known as a hash function that maps a given value with a key to access the elements faster.

- **A Hash table** is a data structure that stores some information, and the information has basically two main components, **i.e., key and value**. The hash table can be implemented with the help of an associative array. The efficiency of mapping depends upon the efficiency of the hash function used for mapping.

ADVANTAGES OF HASH TABLE

Here, are pros/benefits of using hash tables:

1. Hash tables have high performance when looking up data, inserting, and deleting existing values.
2. The time complexity for hash tables is constant regardless of the number of items in the table.
3. They perform very well even when working with large datasets.

DISAVANTAGES OF HASH TABLE

Here, are cons of using hash tables:

1. You cannot use a null value as a key.
2. Collisions cannot be avoided when generating keys using. hash functions. Collisions occur when a key that is already in use is generated.
3. If the hashing function has many collisions, this can lead to performance decrease.

OPERATIONS OF HASH TABLE

Here, are the Operations supported by Hash tables:

1. **Insertion –** this Operation is used to add an element to the hash table
2. **Searching –** this Operation is used to search for elements in the hash table using the key
3. **Deleting –** this Operation is used to delete elements from the hash table

APPLICATIONS OF HASH TABLE

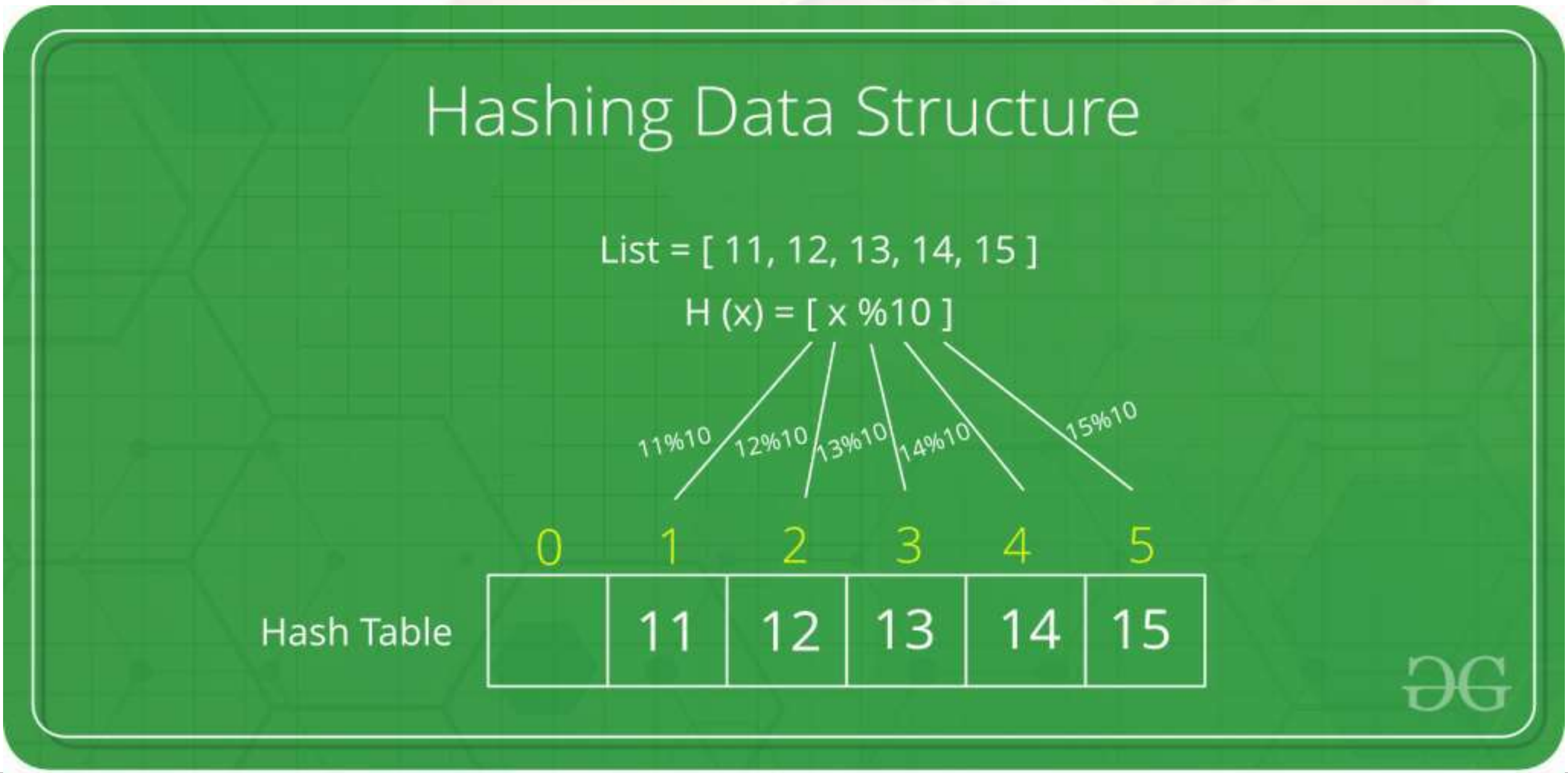
Real-world Applications

In the real-world, hash tables are used to store data for

1. Databases
2. Associative arrays
3. Sets
4. Memory cache

HASH TABLE

- **A hash table** is a data structure that stores records in an array, called a hash table. A Hash table can be used for quick insertion and searching.



PROPERTIES OF HASH FUNCTION

1) Hash function should be simple to computer.

2) Number of collision should be less

2) The hash function uses all the input data.

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

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

HASH FUNCTION

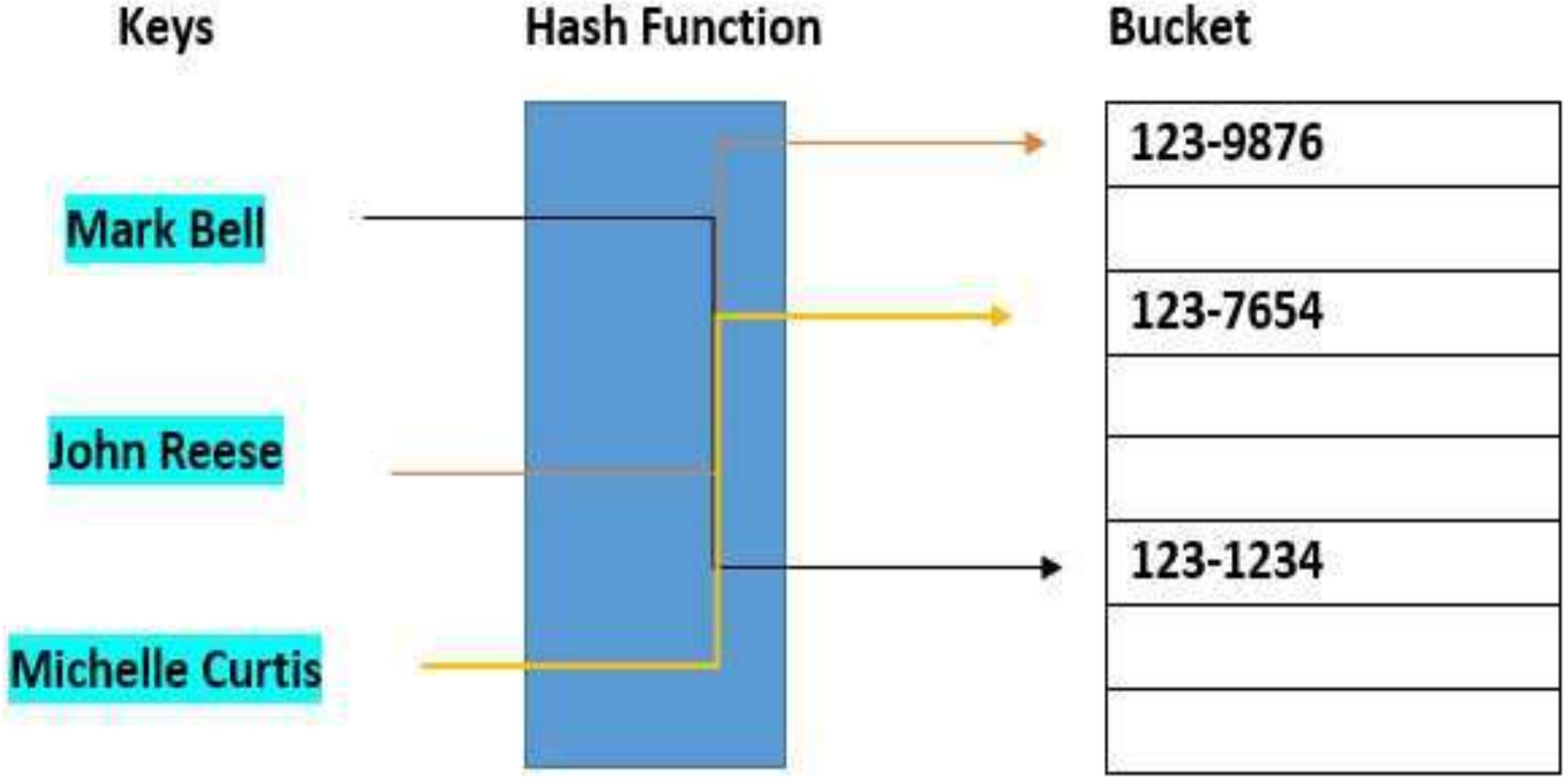
- **A function** that maps a key into the range [0 to Max − 1], the result of which is used as an index (or address) to hash table for storing and retrieving record

- The address generated by hashing function is called as **home address**

- All home addresses address to particular area of memory and that area is called **as prime area**

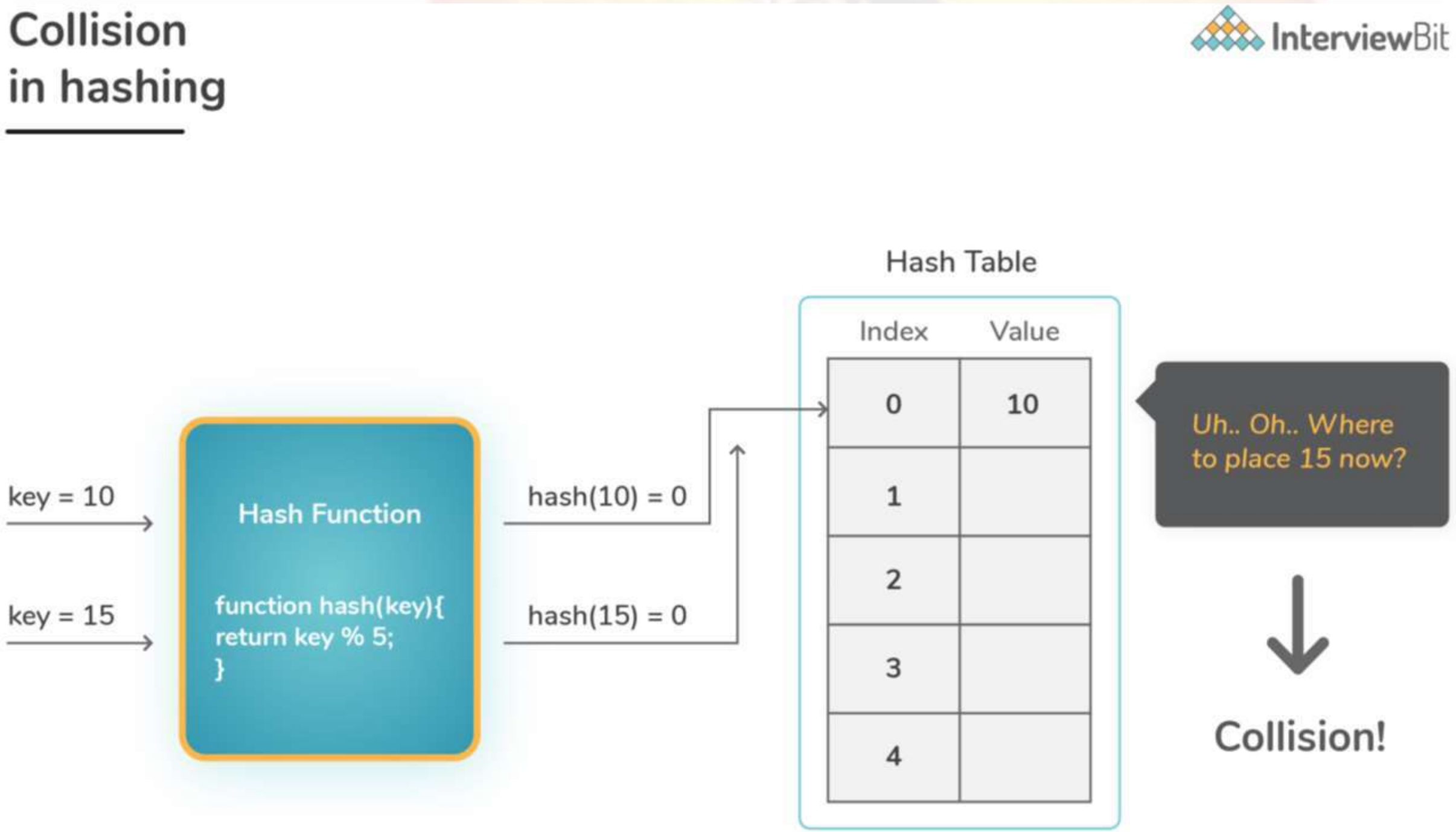
BUCKET

- **Bucket** is an index position in hash table that can store more than one record
- When the same index is mapped with two keys, then both the records are stored in the same bucket



COLLISION

- The result of two keys hashing into the same address is called **collision**



PROBE

- Each calculation of an address and test for success is known as **Probe.**

SYNONYMS

- Keys those hash to the same address are called **synonyms**

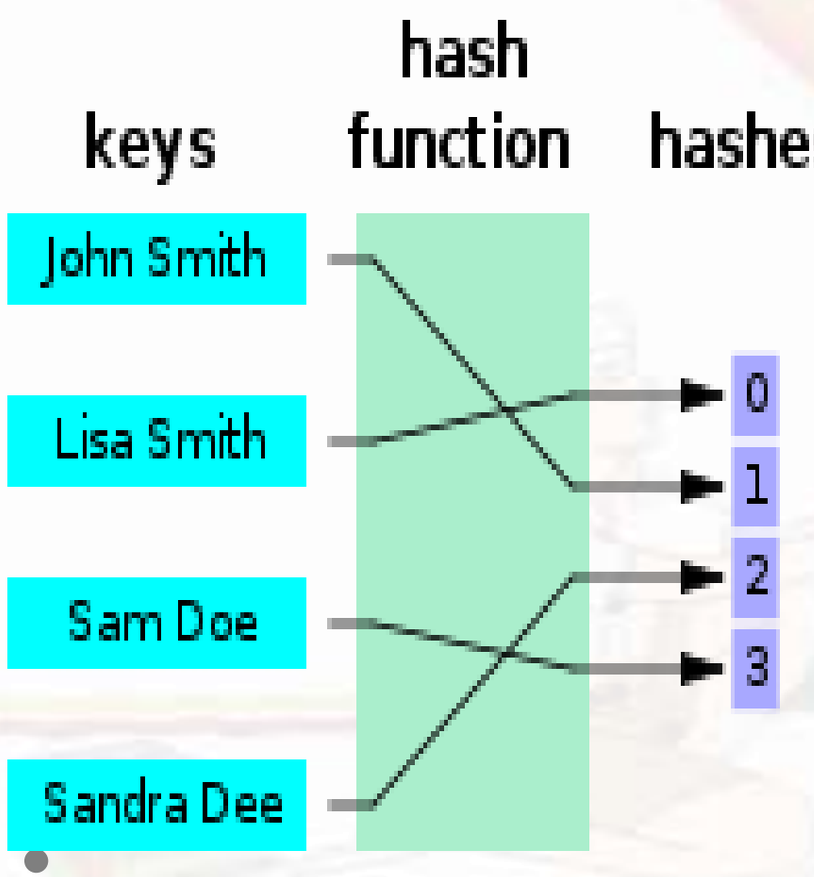
OVERFLOW

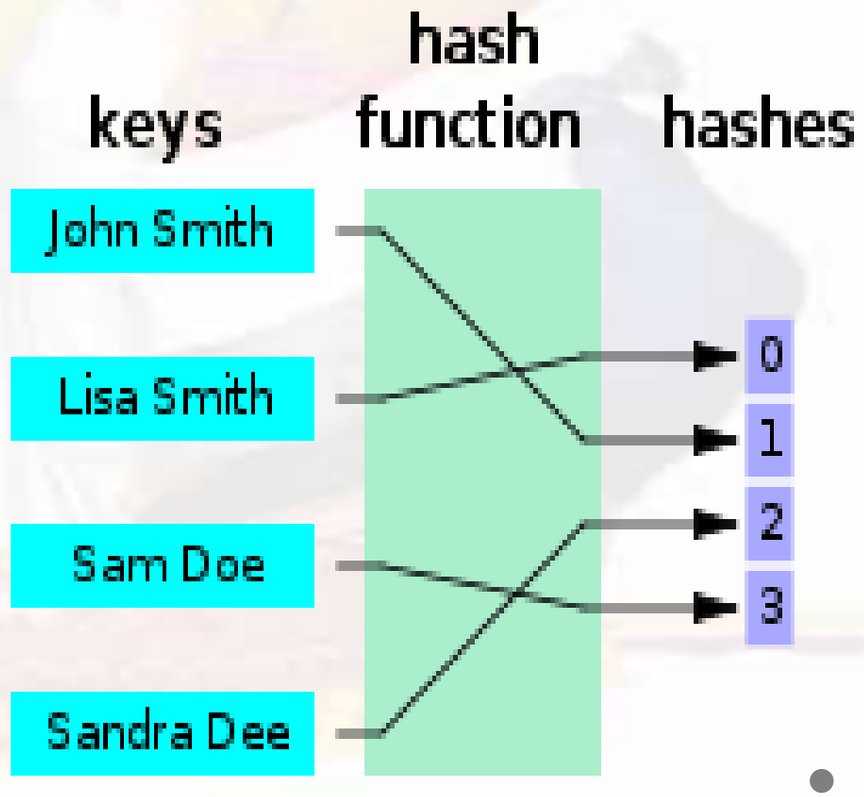
- The result of more keys hashing to the same address and if there is no room in the bucket, then it is said that overflow has occurred

- Collision and overflow are synonymous when the bucket is of size 1

Perfect Hash Function

- A perfect hash function h for a set S is a hash function that maps distinct elements in S to a set of m integers, with no collisions.
- A perfect hash function with values in a limited range can be used for efficient lookup operations, by placing keys from S (or other associated values) in a table indexed by the output of the function.





Perfect Hash Function

- **Advantages :**

1. A perfect hash function with values in a limited range can be used for efficient lookup operations.
2. No need to apply collision resolution techniques.

LOAD FACTOR

Load factor is defined as (m/n) where n is the total size of the hash table and m is the preferred number of entries which can be inserted before a increment in size of the underlying data structure is required.

$$\text{Load Factor } (\alpha) = \frac{\text{Number of elements present in the hash table}}{\text{Total size of the hash table}}$$

If Load factor (α) = constant, then time complexity of Insert, Search, Delete = Θ(1)

Load factor and Load density

- **Load Factor:** The ratio of the number of items in a table to the table's size is called the load factor.

- **Load Density :** The identifier density of a hash table is the ratio n/T,
- where **n is the number of identifiers in the table**. The loading density or loading factor of a hash table is a=n /(sb).
- T is total number of possible element.

Load factor and Load density

- **Load Density :** The identifier density of a hash table is the ratio n/T,
- **Example :**
- Consider the hash table with b = 26 buckets and s = 2.

  We have n = 10 distinct identifiers, each representing a C library function.
- This table has a loading factor, a, of **10/52 = 0.19**

HASHING

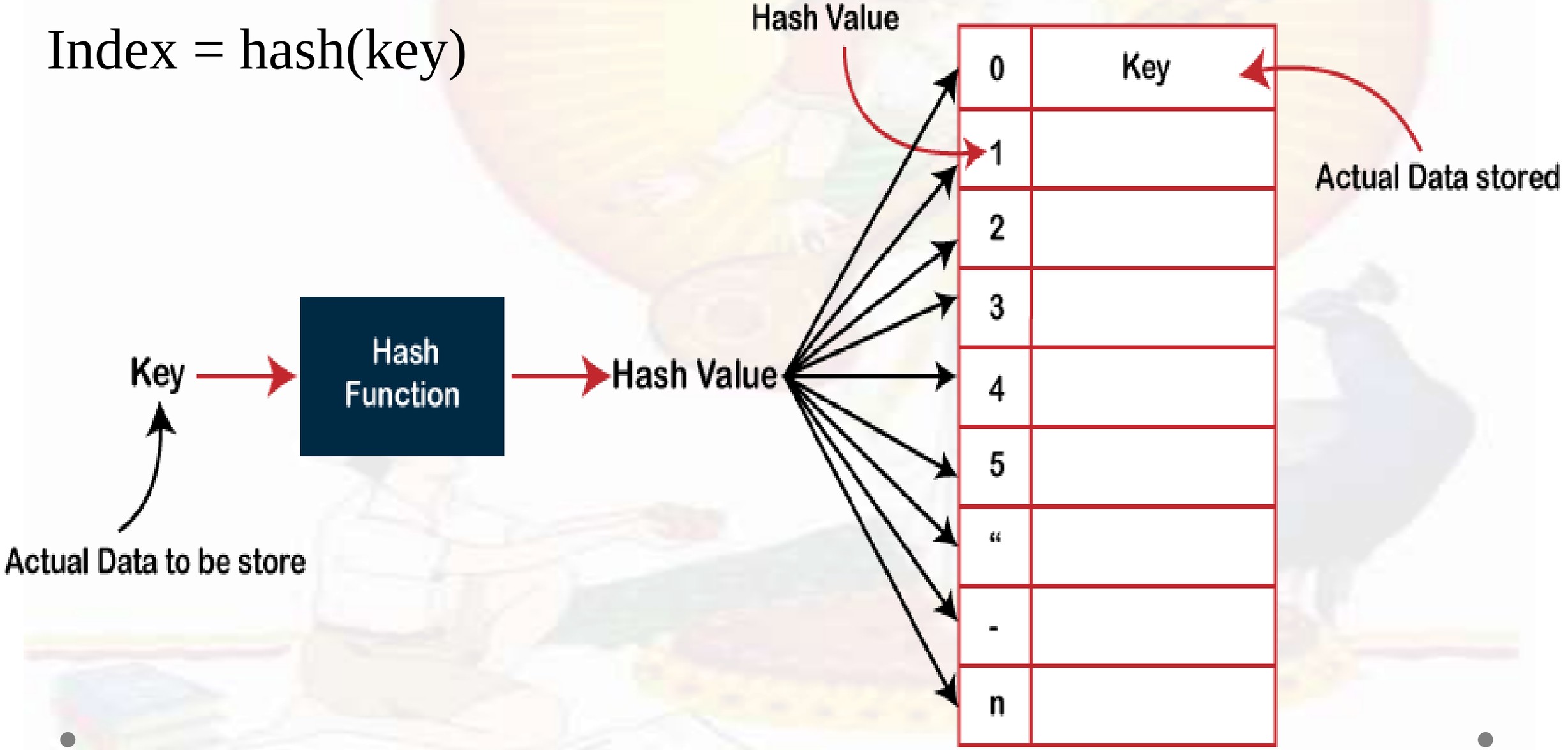
- **Hashing** is one of the searching techniques that uses a constant time. The time complexity in hashing is O(1). Till now, we read the two techniques for searching, i.e., linear search and binary search

- The worst time complexity in linear search is O(n), and O(logn) in binary search. In both the searching techniques, the searching depends upon the number of elements but we want the technique that takes a constant time. So, hashing technique came that provides a constant time.

- **In Hashing technique**, the hash table and hash function are used. Using the hash function, we can calculate the address at which the value can be stored.

HASHING/HASH FUNCTION

- The main idea behind the hashing is to create the (key/value) pairs. If the key is given, then the algorithm computes the index at which the value would be stored. It can be written as:
- Index = hash(key)



TYPES OF HASH FUNCTION

There are three ways of calculating the hash function:

1. Division method
2. Folding method
3. Mid square method

In the division method, the hash function can be defined as:

h(ki) = ki % m;

where m is the size of the hash table.

For example, if the key value is 6 and the size of the hash table is 10. When we apply the hash function to key 6 then the index would be:

h(6) = 6%10 = 6

The index is 6 at which the value is stored.

TYPES OF HASH FUNCTION

1. Division Method:

This is the most simple and easiest method to generate a hash value. The hash function divides the value k by M and then uses the remainder obtained.

Formula:

h(K) = k mod M

Here,

k is the key value, and

M is the size of the hash table.

Example:

k = 12345

M = 95

h(12345) = 12345 mod 95

= 90

k = 1276

M = 11

h(1276) = 1276 mod 11

= 0

TYPES OF HASH FUNCTION

2. The mid square method is a very good hashing method. It involves two steps to compute the hash value-

Square the value of the key k i.e. k^2

Extract the middle r digits as the hash value**.**

Formula:

h(K) = h(k x k)

Here,

k is the key value.

Example:

Suppose the hash table has 100 memory locations.
So r = 2 because two digits are required to map the key to the memory location.

k = 60
k x k = 60 x 60
= 3600
h(60) = 60
The hash value obtained is 60

TYPES OF HASH FUNCTION

3. Digit Folding Method : This method involves two steps:

Divide the key-value k into a number of parts i.e. k1, k2, k3,….,kn, where each part has the same number of digits except for the last part that can have lesser digits than the other parts.

Add the individual parts. The hash value is obtained by ignoring the last carry if any.

Formula:

k = k1, k2, k3, k4, ….., kn

s = k1+ k2 + k3 + k4 +….+ kn

h(K)= s

Here,

s is obtained by adding the parts of the key k

TYPES OF HASH FUNCTION

3. Digit Folding Method :

Example:

k = 12345

k1 = 12, k2 = 34, k3 = 5

s = k1 + k2 + k3

= 12 + 34 + 5

= 51

h(K) = 51

TYPES OF HASH FUNCTION

4. Multiplication method :

This method involves the following steps:

- Choose a constant value A such that $0 < A < 1$.
- Multiply the key value with A.
- Extract the fractional part of kA.
- Multiply the result of the above step by the size of the hash table i.e. M.
- The resulting hash value is obtained by taking the floor of the result obtained in step 4.

TYPES OF HASH FUNCTION

4. Multiplication method :

Formula:

$$h(K) = \text{floor}(M (kA \bmod 1))$$

Here,

M is the size of the hash table.

k is the key value.

A is a constant value.

Where "k A mod 1" means the **fractional part of k A,** that is, k A **-**⌊k A⌋.

TYPES OF HASH FUNCTION

4. Multiplication method :

Example:

k = 12345

A = 0.357840

M = 100

h(12345) = floor[100 (12345*0.357840 mod 1)]

= floor[100 (4417.5348 mod 1)]

= floor[100 (0.5348)]

= floor[53.48]

= 53

COLLISION

When the two different values have the same value, then the problem occurs between the two values, known as a collision. In the above example, the value is stored at index 6. If the key value is 26, then the index would be:

h(26) = 26%10 = 6

Therefore, two values are stored at the same index, i.e., 6, and this leads to the collision problem. To resolve these collisions, we have some techniques known as collision techniques.

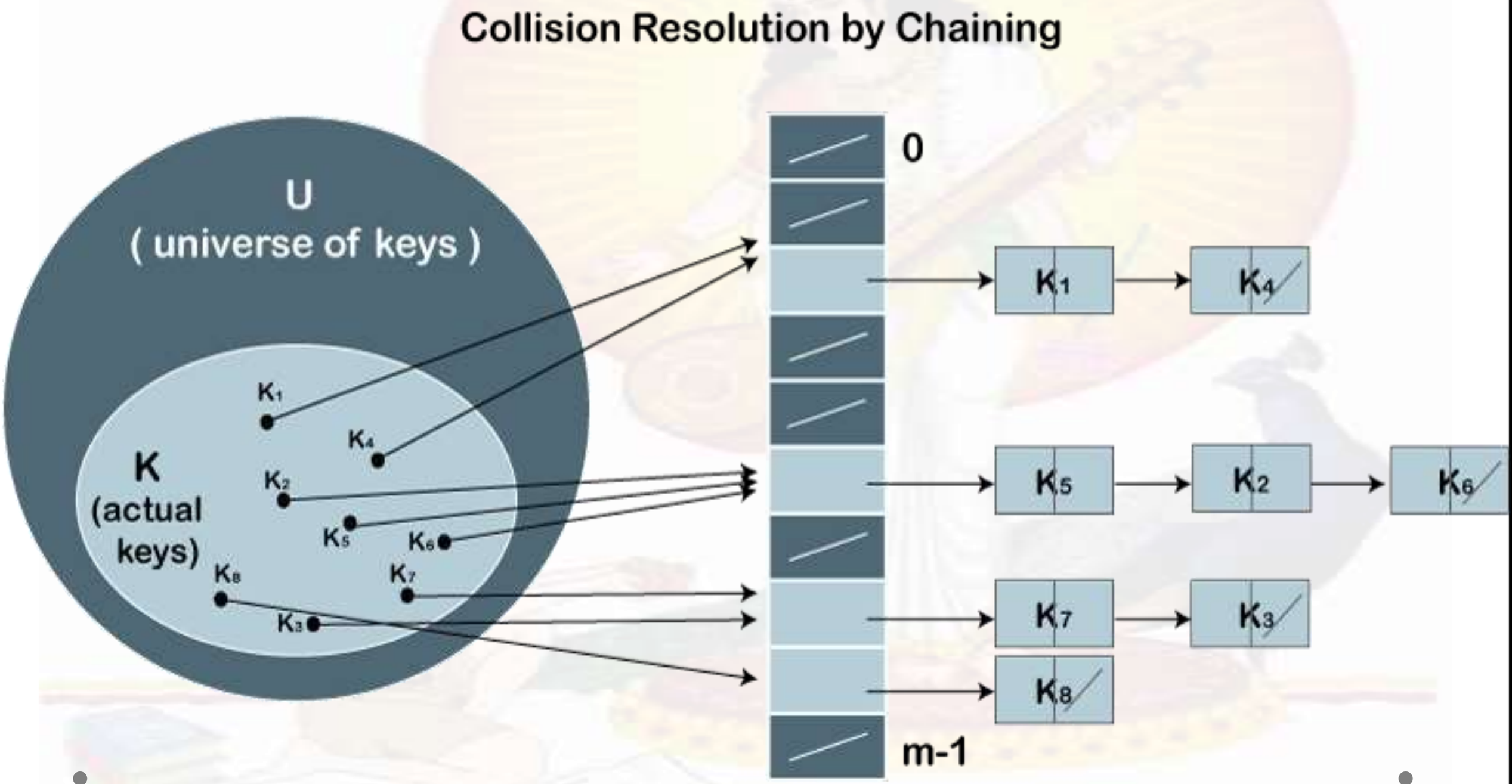
The following are the collision techniques:

Open Hashing: It is also known as closed addressing.

Closed Hashing: It is also known as open addressing.

OPEN HASHING

n Open Hashing, one of the methods used to resolve the collision is known as a chaining method.



OPEN HASHING

Let's first understand the chaining to resolve the collision.

Suppose we have a list of key values

A = 3, 2, 9, 6, 11, 13, 7, 12 where m = 10, and h(k) = 2k+3

In this case, we cannot directly use h(k) = ki/m as h(k) = 2k+3

The index of key value 3 is:

index = h(3) = (2(3)+3)%10 = 9

The value 3 would be stored at the index 9.

The index of key value 2 is:

index = h(2) = (2(2)+3)%10 = 7

The value 2 would be stored at the index 7.

OPEN HASHING

The index of key value 9 is:

index = h(9) = (2(9)+3)%10 = 1

The value 9 would be stored at the index 1.

The index of key value 6 is:

index = h(6) = (2(6)+3)%10 = 5

The value 6 would be stored at the index 5.

The index of key value 11 is:

index = h(11) = (2(11)+3)%10 = 5

The value 11 would be stored at the index 5.

OPEN HASHING

The index of key value 13 is:

index = h(13) = (2(13)+3)%10 = 9

The value 13 would be stored at index 9.

The index of key value 7 is:

index = h(7) = (2(7)+3)%10 = 7

The value 7 would be stored at index 7.

The index of key value 12 is:

index = h(12) = (2(12)+3)%10 = 7

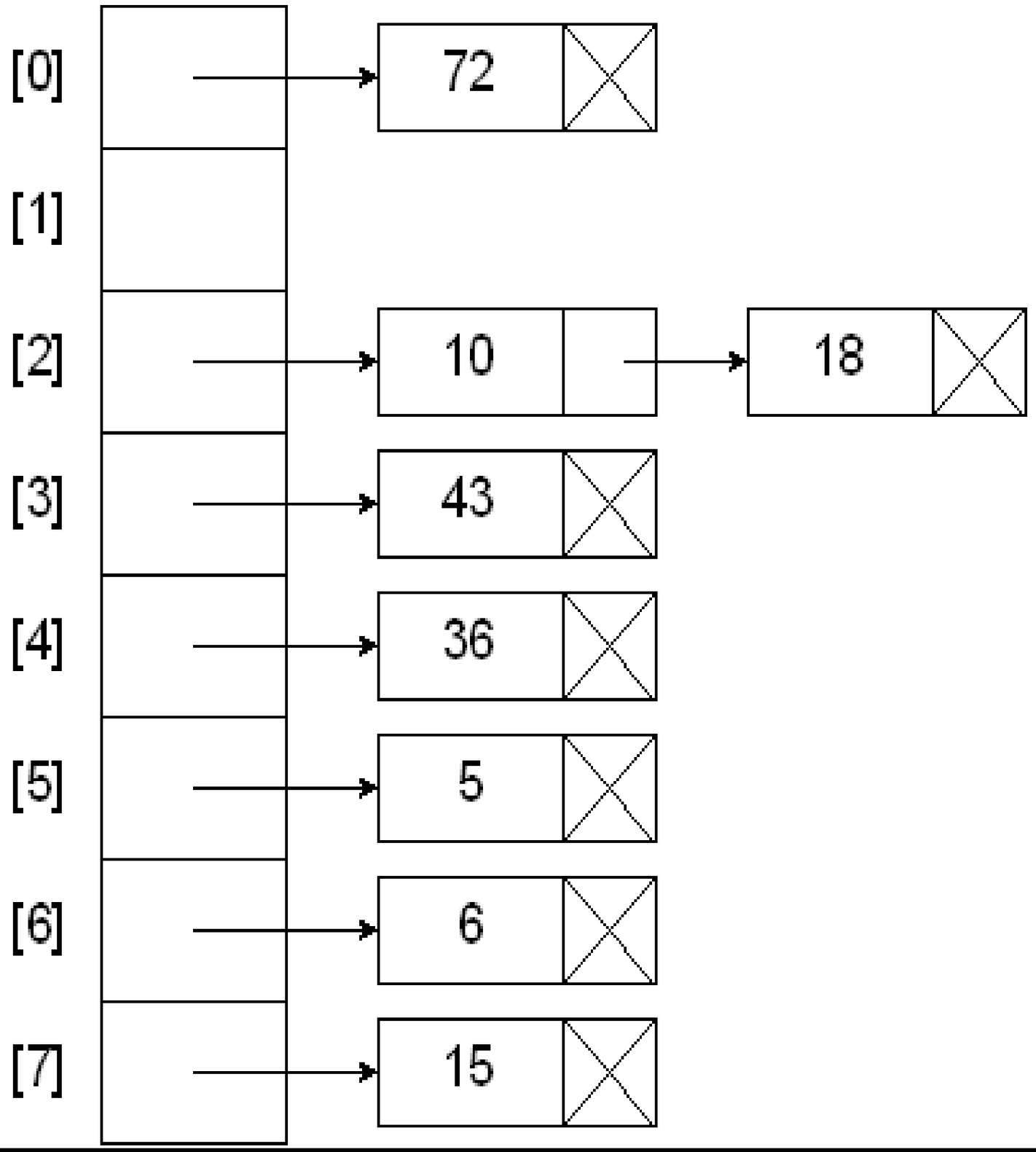
The value 7 would be stored at index 7.

OPEN HASHING

The hash table slots will no longer hold a table element. They will now hold the address of a table element.

Hash key = key % table size

4 = 36 % 8
2 = 18 % 8
0 = 72 % 8
3 = 43 % 8
6 = 6 % 8
2 = 10 % 8
5 = 5 % 8
7 = 15 % 8



OPEN HASHING

$$36 \% 8 = 4$$

$$18 \% 8 = 2$$

$$72 \% 8 = 0$$

$$43 \% 8 = 3$$

$$6 \% 8 = 6$$

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
72		18	43	36		6	

CLOSED HASHING

In Closed hashing, three techniques are used to resolve the collision:

1. Linear probing
2. Quadratic probing
3. Double Hashing technique

PROBING

Method	Description
Linear probing	Just like the name suggests, this method searches for empty slots linearly starting from the position where the collision occurred and moving forward. If the end of the list is reached and no empty slot is found. The probing starts at the beginning of the list.
Quadratic probing	This method uses quadratic polynomial expressions to find the next available free slot.
Double Hashing	This technique uses a secondary hash function algorithm to find the next free available slot.

CLOSED HASHING

Linear Probing

Linear probing is one of the forms of open addressing. As we know that each cell in the hash table contains a key-value pair, **so when the collision occurs by mapping a new key to the cell already occupied by another key, then linear probing technique searches for the closest free locations and adds a new key to that empty cell**. In this case, searching is performed sequentially, starting from the position where the collision occurs till the empty cell is not found.

CLOSED HASHING

Let's understand the linear probing through an example.

Consider the above example for the linear probing:

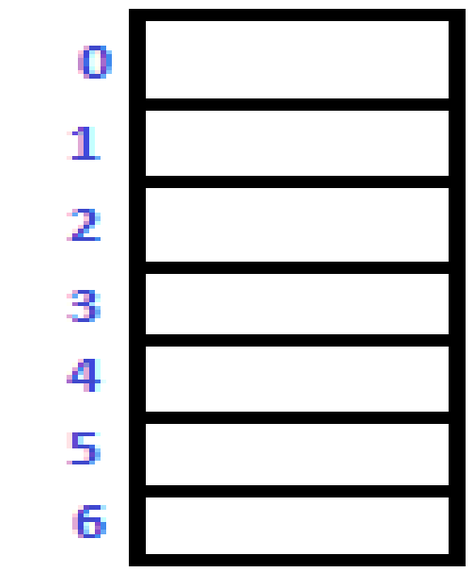
A = 3, 2, 9, 6, 11, 13, 7, 12 where m = 10, and h(k) = 2k+3

The key values 3, 2, 9, 6 are stored at the indexes 9, 7, 1, 5 respectively. The calculated index value of 11 is 5 which is already occupied by another key value, i.e., 6. When linear probing is applied, the nearest empty cell to the index 5 is 6; therefore, the value 11 will be added at the index 6.

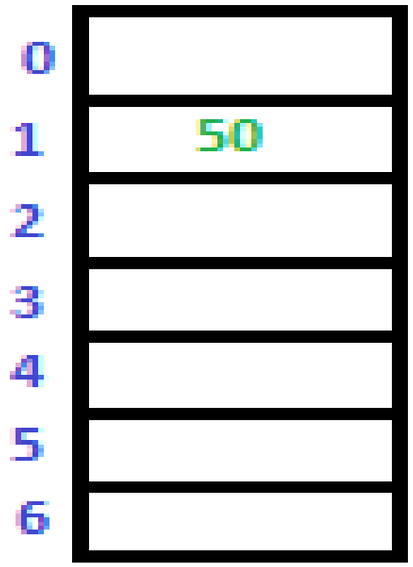
The next key value is 13. The index value associated with this key value is 9 when hash function is applied. The cell is already filled at index 9. When linear probing is applied, the nearest empty cell to the index 9 is 0; therefore, the value 13 will be added at the index 0

CLOSED HASHING

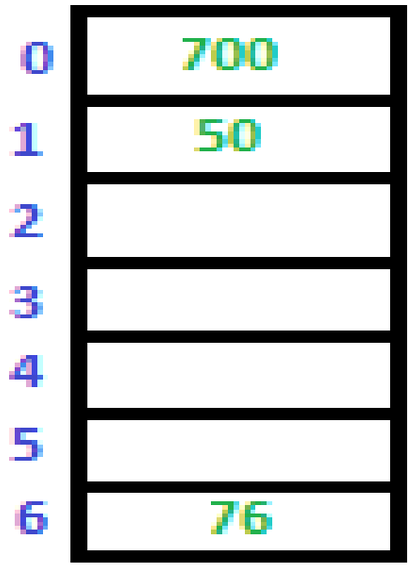
Let us consider a simple hash function as "key mod 7" and a sequence of keys as 50, 700, 76, 85, 92, 73, 101.



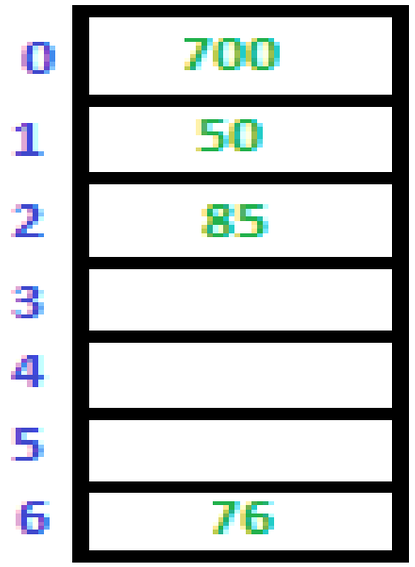
Initial Empty Table



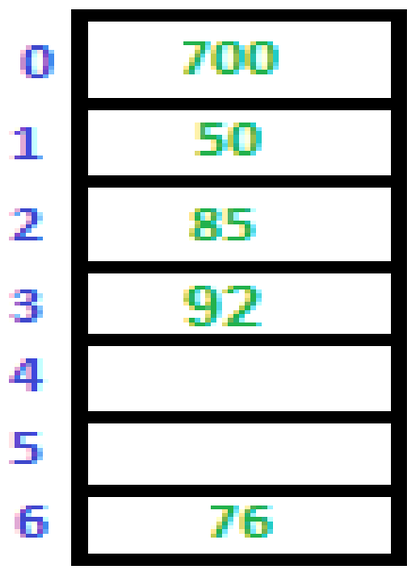
Insert 50



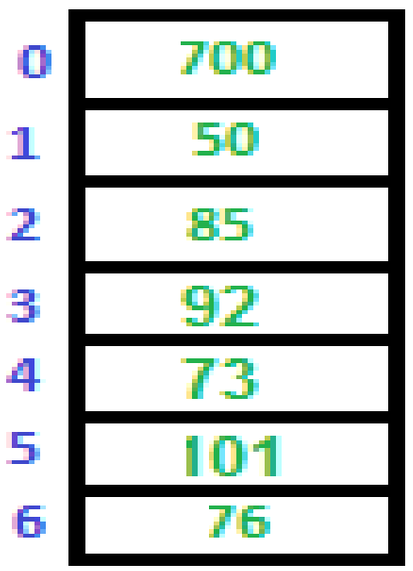
Insert 700 and 76



Insert 85: Collision Occurs, insert 85 at next free slot.



Insert 92, collision occurs as 50 is there at index 1. Insert at next free slot



Insert 73 and 101

CLOSED HASHING

Quadratic Probing

- In case of linear probing, searching is performed linearly. In contrast, quadratic probing is an open addressing technique that uses quadratic polynomial for searching until a empty slot is found.
- It can also be defined as that it allows the insertion ki at first free location from (u+i2)%m where i=0 to m-1.

$\mathbf{h'} = (x) = x \bmod m$

$h(x, i) = (h'(x) + i2) \bmod m$

We can put some other quadratic equations also using some constants

The value of i = 0, 1, . . ., m – 1. So we start from i = 0, and increase this until we get one free space. So initially when i = 0, then the h(x, i) is same as h′(x).

CLOSED HASHING

Quadratic Probing -- Example

- Example:
  - Table Size is 11 (0..10)
  - Hash Function: **h(x) = x mod 11**
  - Insert keys: 20, 30, 2, 13, 25, 24, 10, 9
    - 20 mod 11 = 9
    - 30 mod 11 = 8
    - 2 mod 11 = 2
    - 13 mod 11 = 2 → $2+1^2=3$
    - 25 mod 11 = 3 → $3+1^2=4$
    - 24 mod 11 = 2 → $2+1^2$, $2+2^2=6$
    - 10 mod 11 = 10
    - 9 mod 11 = 9 → $9+1^2$, $9+2^2$ mod 11, $9+3^2$ mod 11 =7

0	
1	
2	2
3	13
4	25
5	
6	**24**
7	**9**
8	30
9	20
10	10

CLOSED HASHING

Double hashing : It is a collision resolving technique in Open Addressed Hash tables. Double hashing uses the idea of applying a second hash function to key when a collision occurs.

Advantages of Double hashing

1. The advantage of Double hashing is that it is one of the best form of probing, producing a uniform distribution of records throughout a hash table.
2. This technique does not yield any clusters.
3. It is one of effective method for resolving collisions

CLOSED HASHING

Double hashing :

Double hashing can be done using :

(hash1(key) + i * hash2(key)) % TABLE_SIZE

Here hash1() and hash2() are hash functions and TABLE_SIZE is size of hash table.

(We repeat by increasing i when collision occurs)

First hash function is typically hash1(key) = key % TABLE_SIZE

A popular second hash function is : hash2(key) = PRIME – (key % PRIME) where PRIME is a prime smaller than the TABLE_SIZE.

CLOSED HASHING

Double hashing : A good second Hash function is:

1. It must never evaluate to zero
2. Must make sure that all cells can be probed

Lets say, Hash1 (key) = key % 13

Hash2 (key) = 7 – (key % 7)

Hash1(19) = 19 % 13 = 6

Hash1(27) = 27 % 13 = 1

Hash1(36) = 36 % 13 = 10

Hash1(10) = 10 % 13 = 10

Hash2(10) = 7 – (10%7) = 4

(Hash1(10) + 1*Hash2(10))%13= 1

(Hash1(10) + 2*Hash2(10))%13= 5

Collision

Hashing

- **Hashing** is a well-known technique to search any particular element among several elements.
- It minimizes the number of comparisons while performing the search.

Advantage-

1. Hashing is extremely efficient.
2. The time taken by it to perform the search does not depend upon the total number of elements.
3. It completes the search with constant time complexity O(1).

Hashing

- **Hashing Mechanism -**

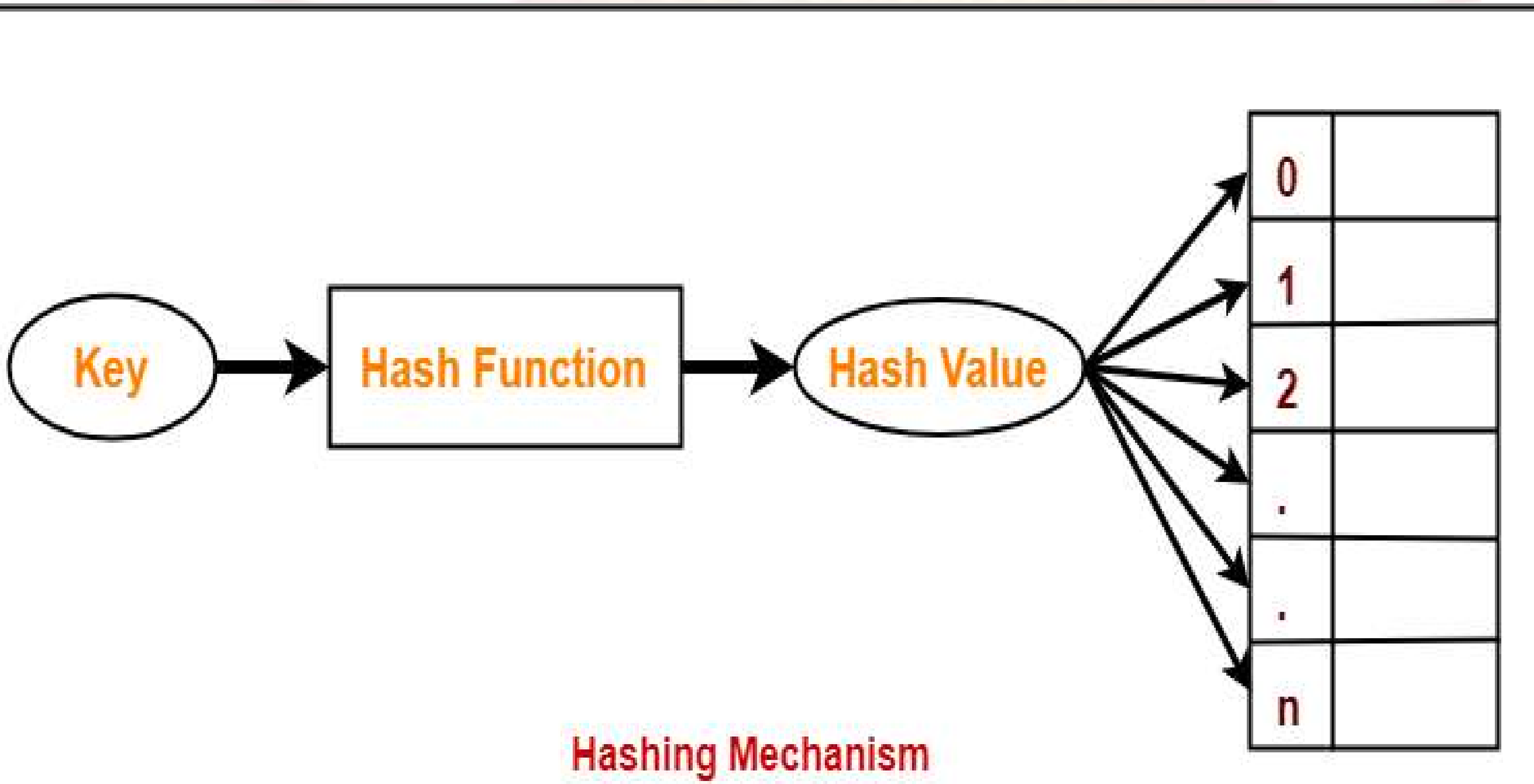
1. An array data structure called as Hash table is used to store the data items.
2. Based on the hash key value, data items are inserted into the hash table.

- **Hash Key Value -**

1. Hash key value is a special value that serves as an index for a data item.
2. It indicates where the data item should be be stored in the hash table.
3. Hash key value is generated using a hash function.

Hashing

- **Hashing Mechanism :**



Hashing Mechanism

Hashing

- **Hash Fuction :**

1. Hash function takes the data item as an input and returns a small integer value as an output.
2. The small integer value is called as a hash value.
3. Hash value of the data item is then used as an index for storing it into the hash table.

Types of Hash Functions-

There are various types of hash functions available such as-

1. Mid Square Hash Function
2. Division Hash Function
3. Folding Hash Function etc

It depends on the user which hash function he wants to use.

Collision in Hashing

- **Collision in Hashing-**

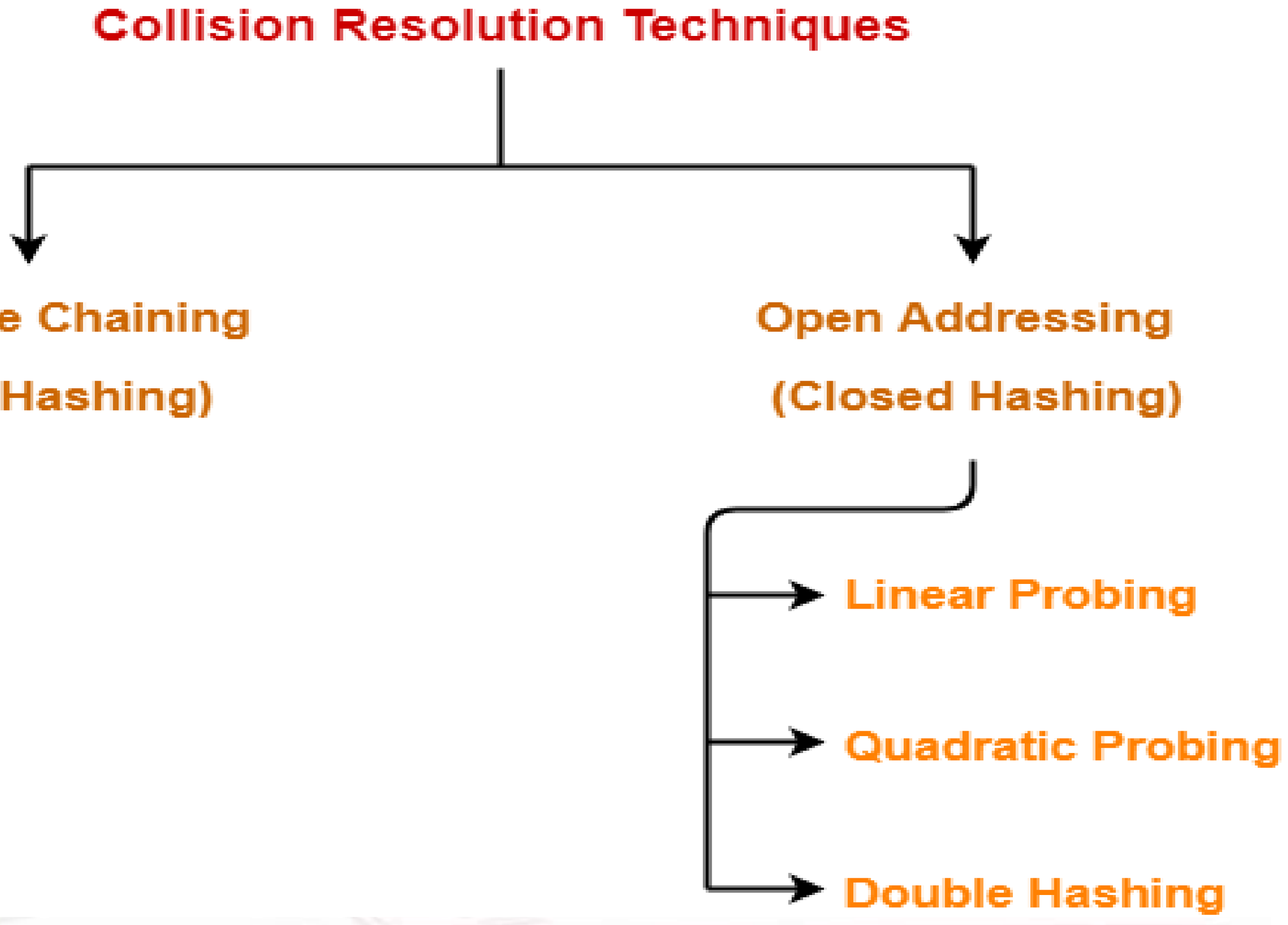
In hashing,

1. Hash function is used to compute the hash value for a key.
2. Hash value is then used as an index to store the key in the hash table.
3. Hash function may return the same hash value for two or more keys.

“When the hash value of a key maps to an already occupied bucket of the hash table, it is called as a Collision”

Collision Resolution Techniques

In Hashing, collision resolution techniques are classified as-



Collision Resolution Techniques

1. Separate Chaining (hashing with chaining/open hashing)

To handle the collision,

1. This technique creates a linked list to the slot for which collision occurs.
2. The new key is then inserted in the linked list.
3. These linked lists to the slots appear like chains.
4. That is why, this technique is called as separate chaining

Separate Chaining

(Open hashing/External hashing)

Collision Resolution Techniques

For Searching-

- In worst case, all the keys might map to the same bucket of the hash table.
- In such a case, all the keys will be present in a single linked list.
- Sequential search will have to be performed on the linked list to perform the search.
- So, time taken for searching in worst case is O(n).

For Deletion-

- In worst case, the key might have to be searched first and then deleted.
- In worst case, time taken for searching is O(n).
- So, time taken for deletion in worst case is O(n).

Collision Resolution Techniques

Advantages:

1) Simple to implement.

2) Hash table never fills up, we can always add more elements to the chain.

3) Less sensitive to the hash function or load factors.

4) It is mostly used when it is unknown how many and how frequently keys may be inserted or deleted. .

Collision Resolution Techniques

Disadvantages:

1) Cache performance of chaining is not good as keys are stored using a linked list. Open addressing provides better cache performance as everything is stored in the same table.

2) Wastage of Space (Some Parts of hash table are never used)

3) If the chain becomes long, then search time can become O(n) in the worst case.

4) Uses extra space for links

Collision Resolution Techniques

Using the hash function 'key mod 7', insert the following sequence of keys in the hash table-

50, 700, 76, 85, 92, 73 and 101.

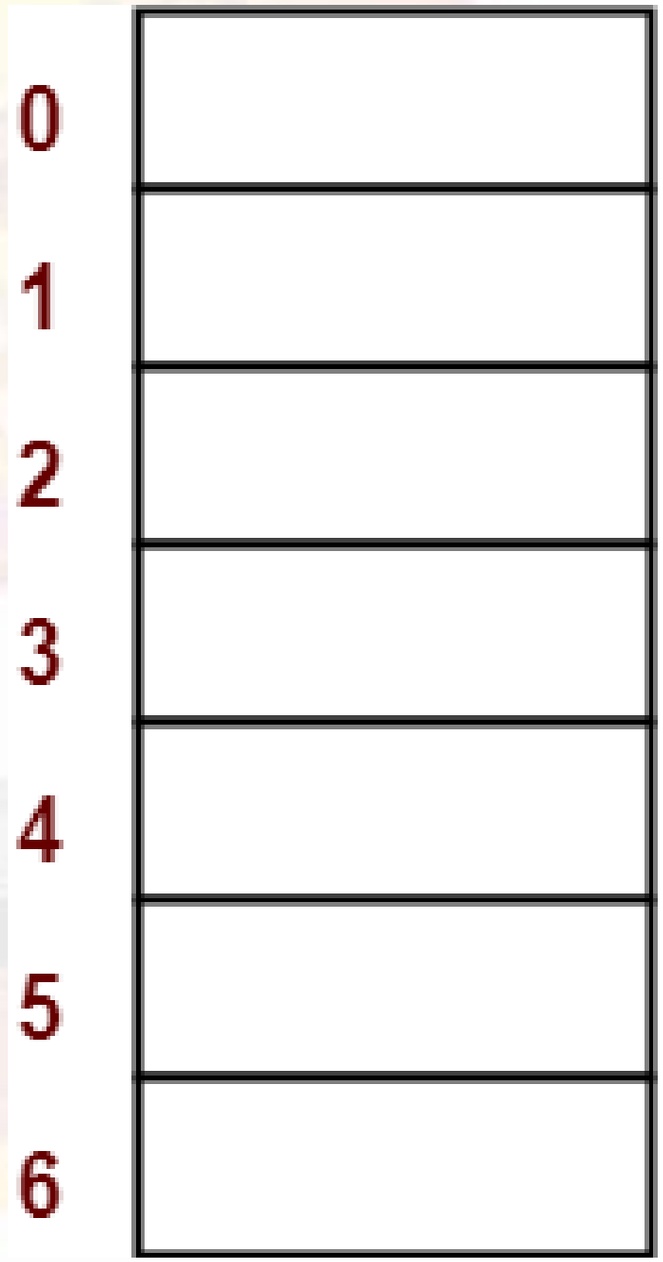
Use separate chaining technique for collision resolution.

Step-01:

Draw an empty hash table.

For the given hash function, the possible range of hash values is [0, 6].

So, draw an empty hash table consisting of 7 buckets as-



Collision Resolution Techniques

Using the hash function 'key mod 7', insert the following sequence of keys in the hash table-

50, 700, 76, 85, 92, 73 and 101.

Use separate chaining technique for collision resolution.

Step-02:

Insert the given keys in the hash table one by one.

The first key to be inserted in the hash table = 50.

Bucket of the hash table to which key 50 maps = 50 mod 7 = 1.

So, key 50 will be inserted in bucket-1 of the hash table as-

0	
1	50
2	
3	
4	
5	
6	

Collision Resolution Techniques

Using the hash function 'key mod 7', insert the following sequence of keys in the hash table-

50, 700, 76, 85, 92, 73 and 101.

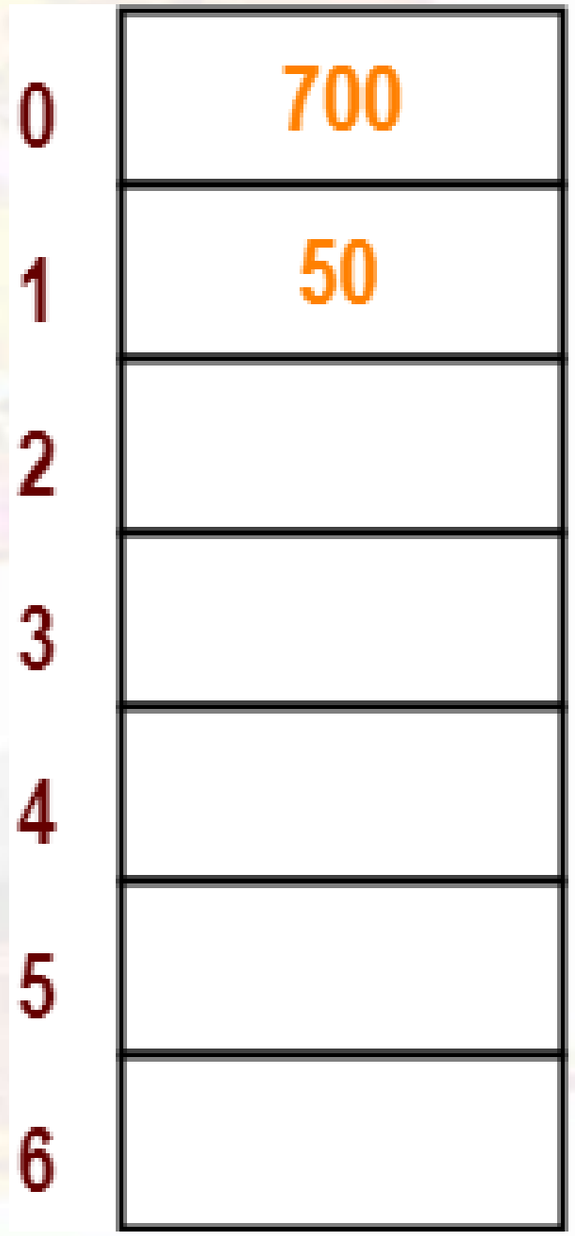
Use separate chaining technique for collision resolution.

Step-03:

The next key to be inserted in the hash table = 700.

Bucket of the hash table to which key 700 maps = 700 mod 7 = 0.

So, key 700 will be inserted in bucket-0 of the hash table as-



Collision Resolution Techniques

Using the hash function 'key mod 7', insert the following sequence of keys in the hash table-

50, 700, 76, 85, 92, 73 and 101.

Use separate chaining technique for collision resolution.

Step-04:

The next key to be inserted in the hash table = 76.

Bucket of the hash table to which key 76 maps = 76 mod 7 = 6.

So, key 76 will be inserted in bucket-6 of the hash table as-

0	700
1	50
2	
3	
4	
5	
6	76

Collision Resolution Techniques

Using the hash function 'key mod 7', insert the following sequence of keys in the hash table-

50, 700, 76, 85, 92, 73 and 101.

Use separate chaining technique for collision resolution.

Step-05:

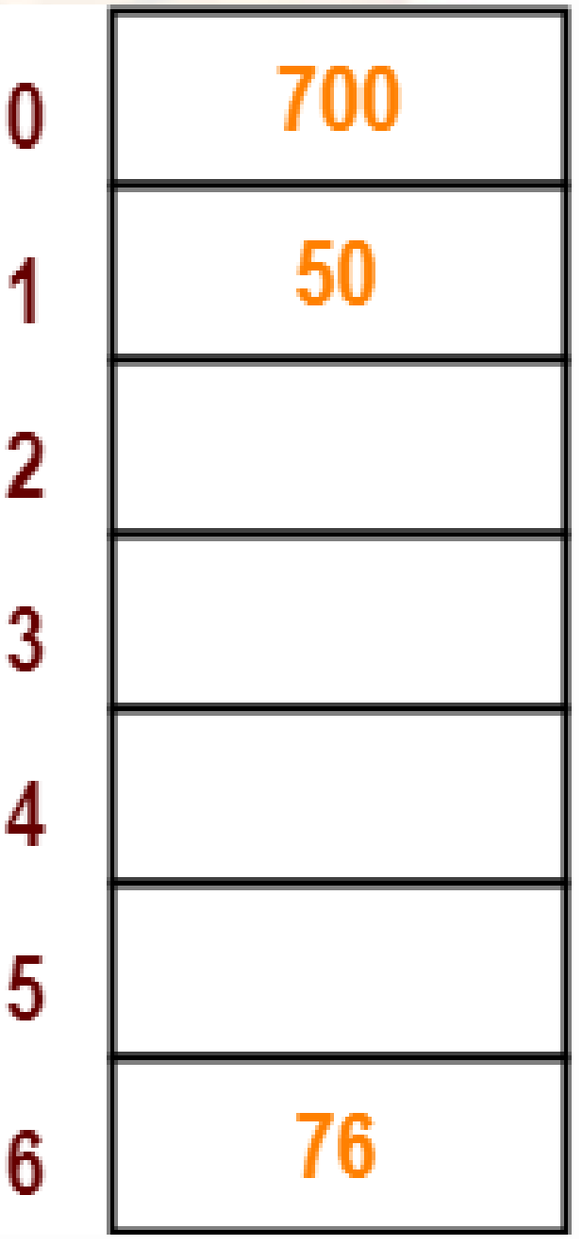
- The next key to be inserted in the hash table = 85.

Bucket of the hash table to which key 85 maps = 85 mod 7 = 1.

Since bucket-1 is already occupied, so collision occurs.

Separate chaining handles the collision by creating a linked list to bucket-1.

So, key 85 will be inserted in bucket-1 of the hash table as-



Collision Resolution Techniques

Using the hash function 'key mod 7', insert the following sequence of keys in the hash table-

50, 700, 76, 85, 92, 73 and 101.

Use separate chaining technique for collision resolution.

Step-06 :

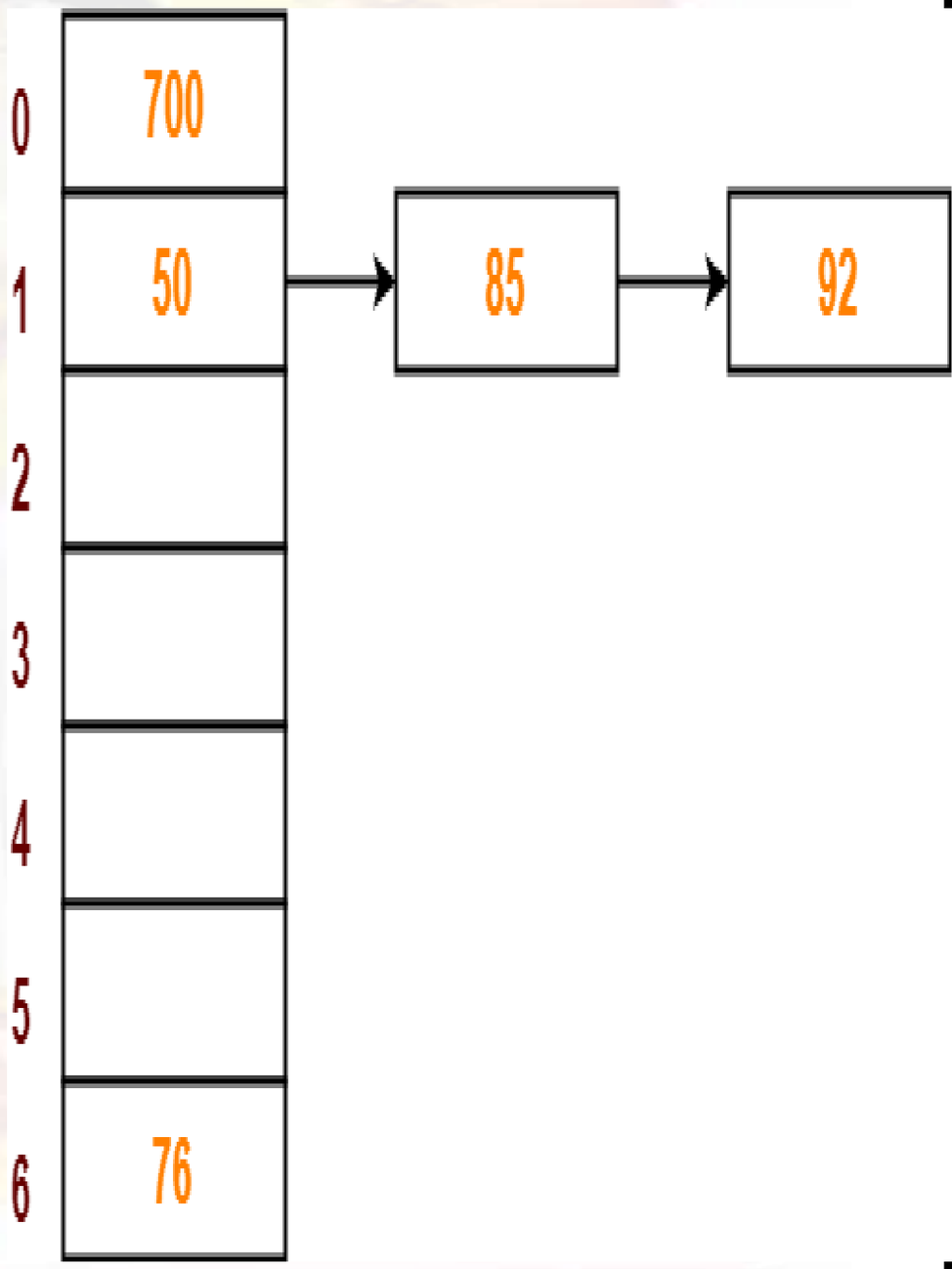
The next key to be inserted in the hash table = 92.

Bucket of the hash table to which key 92 maps = 92 mod 7 = 1.

Since bucket-1 is already occupied, so collision occurs.

Separate chaining handles the collision by creating a linked list to bucket-1.

So, key 92 will be inserted in bucket-1 of the hash table as-



Collision Resolution Techniques

Using the hash function 'key mod 7', insert the following sequence of keys in the hash table-

50, 700, 76, 85, 92, 73 and 101.

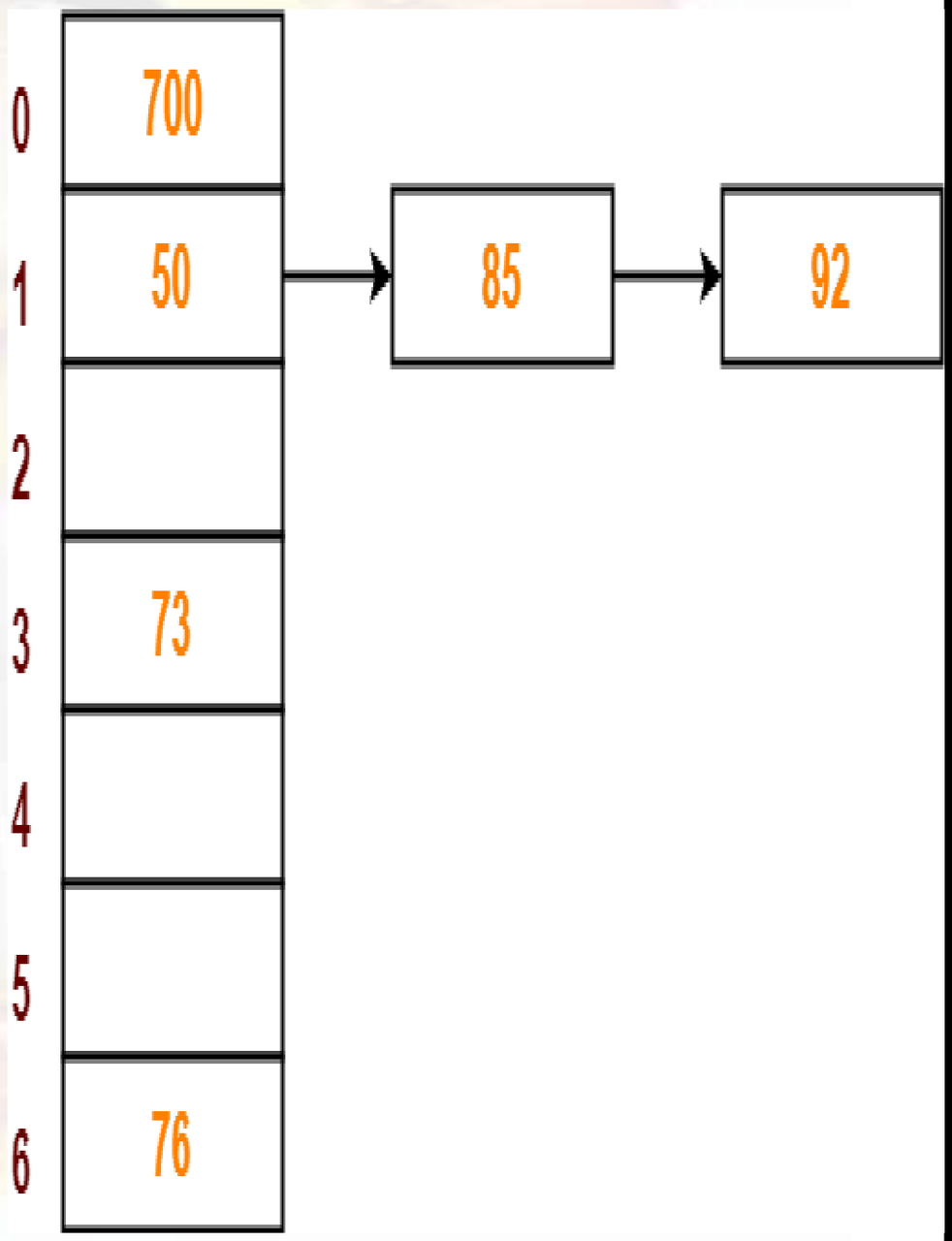
Use separate chaining technique for collision resolution.

Step-07 :

The next key to be inserted in the hash table = 73.

Bucket of the hash table to which key 73 maps = 73 mod 7 = 3.

So, key 73 will be inserted in bucket-3 of the hash table as-



Collision Resolution Techniques

Using the hash function 'key mod 7', insert the following sequence of keys in the hash table-

50, 700, 76, 85, 92, 73 and 101.

Use separate chaining technique for collision resolution.

Step-08 :

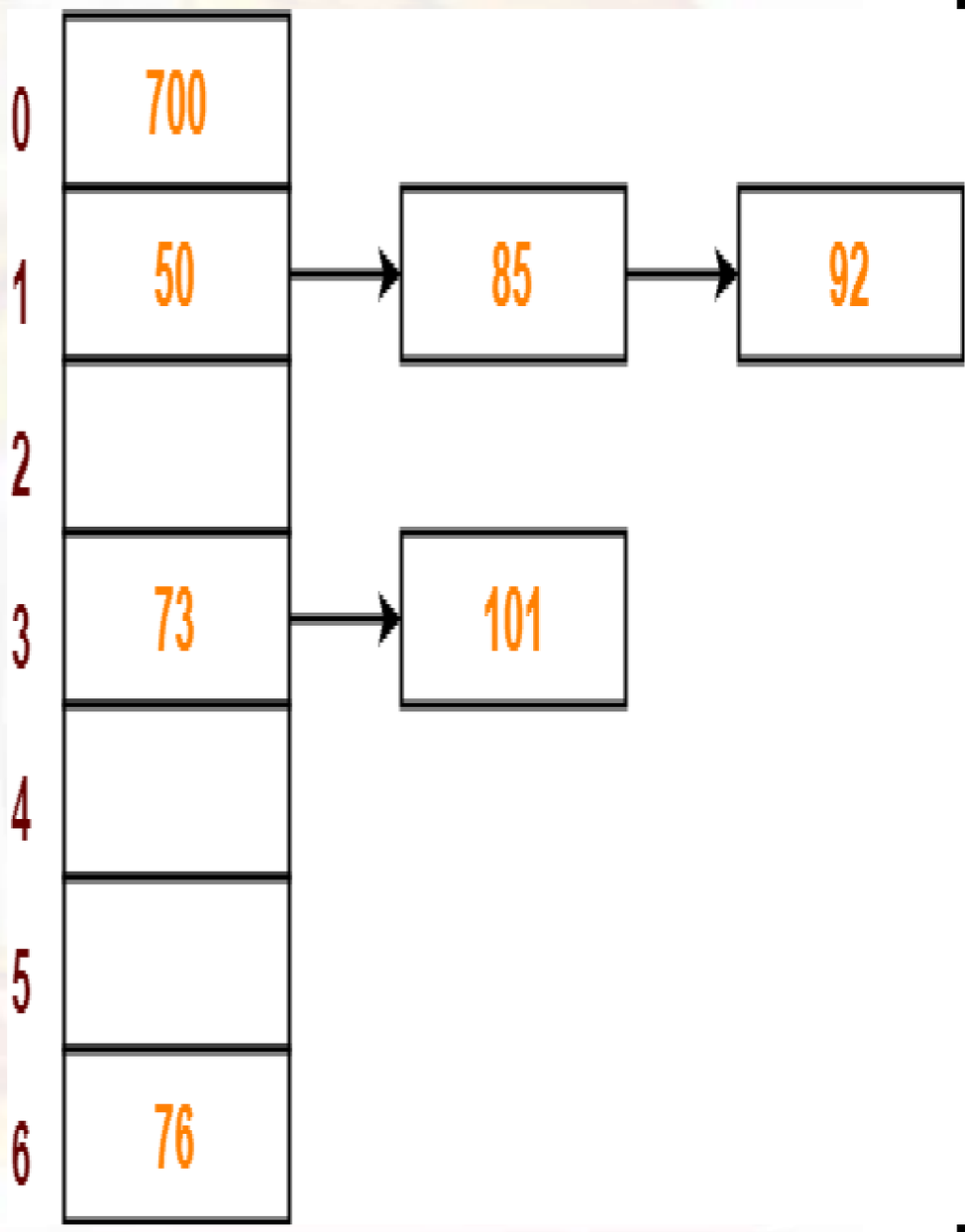
The next key to be inserted in the hash table = 101.

Bucket of the hash table to which key 101 maps = 101 mod 7 = 3.

Since bucket-3 is already occupied, so collision occurs.

Separate chaining handles the collision by creating a linked list to bucket-3.

So, key 101 will be inserted in bucket-3 of the hash table as-

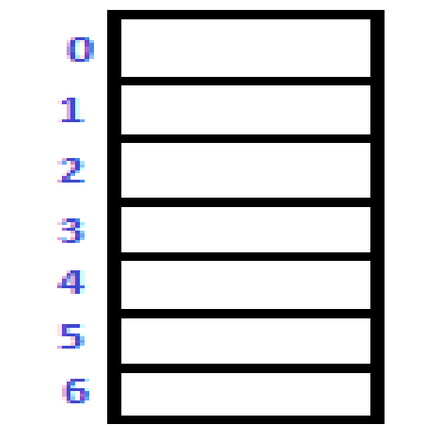


Collision Resolution Techniques

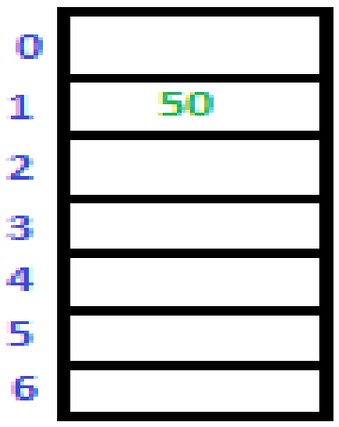
Using the hash function 'key mod 7', insert the following sequence of keys in the hash table-

50, 700, 76, 85, 92, 73 and 101.

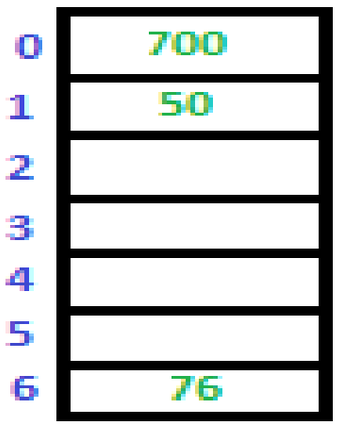
Use separate chaining technique for collision resolution.



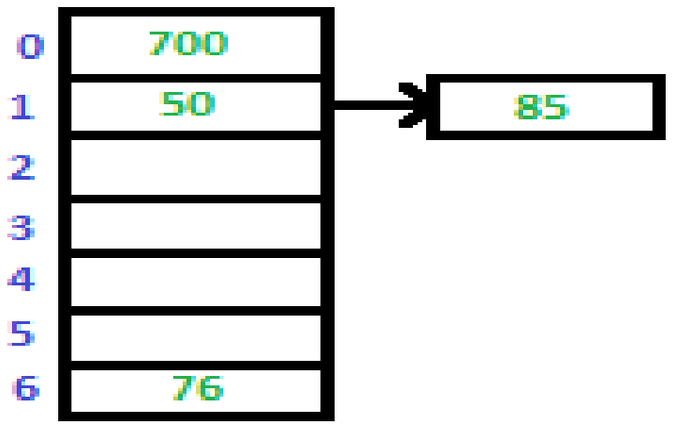
Initial Empty Table



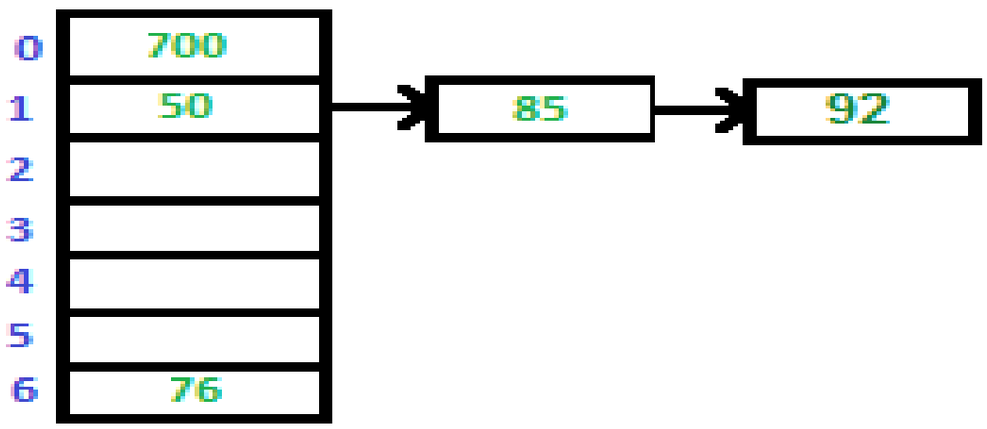
Insert 50



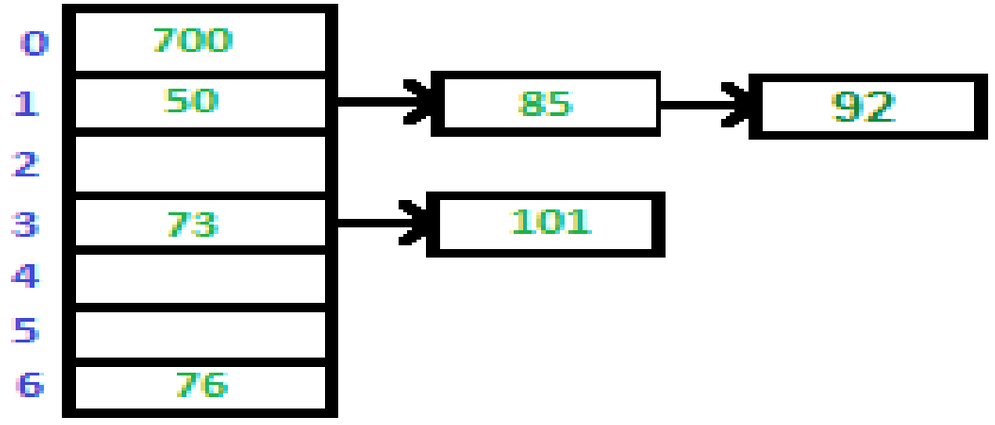
Insert 700 and 76



Insert 85: Collision
Occurs, add to chain



Inser 92 Collision
Occurs, add to chain



Insert 73 and 101

Open addressing

(Close hashing/Internal hashing)

Open Addressing

In open addressing,

1. Unlike separate chaining, all the keys are stored inside the hash table.
2. No key is stored outside the hash table.

Techniques used for open addressing are-

1. Linear Probing
2. Quadratic Probing
3. Double Hashing

Open Addressing

Operations In open addressing,

1. Insert Operation-

- Hash function is used to compute the hash value for a key to be inserted.
- Hash value is then used as an index to store the key in the hash table.

In case of collision,

- Probing is performed until an empty bucket is found.
- Once an empty bucket is found, the key is inserted.
- Probing is performed in accordance with the technique used for open addressing.

Open Addressing

Operations In open addressing,

Search Operation-

To search any particular key,

- Its hash value is obtained using the hash function used.
- Using the hash value, that bucket of the hash table is checked.
- If the required key is found, the key is searched.
- Otherwise, the subsequent buckets are checked until the required key or an empty bucket is found.
- The empty bucket indicates that the key is not present in the hash table.

Open Addressing

Operations In open addressing,

Search Operation-

- The key is first searched and then deleted.
- After deleting the key, that particular bucket is marked as "deleted".

Open Addressing Techniques

1. Linear Probing-

1. When collision occurs, we linearly probe for the next bucket.
2. We keep probing until an empty bucket is found.

Advantage-

1. It is easy to compute.

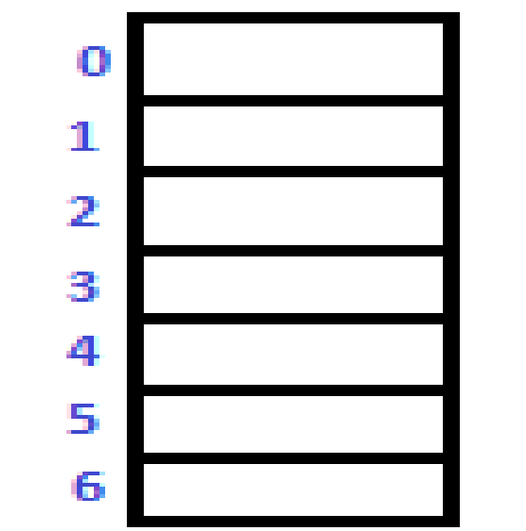
Disadvantage-

1. The main problem with linear probing is clustering.
2. Many consecutive elements form groups.
3. Then, it takes time to search an element or to find an empty bucket.

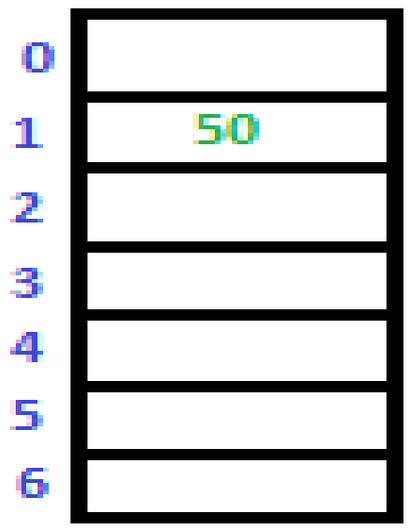
Open Addressing Techniques

1. Linear Probing -

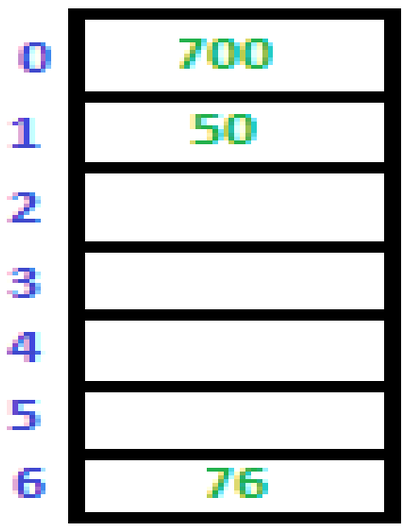
Let us consider a simple hash function as “key mod 7” and a sequence of keys as 50, 700, 76, 85, 92, 73, 101.



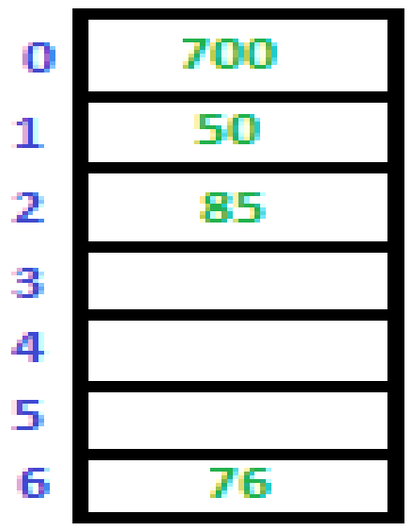
Initial Empty Table



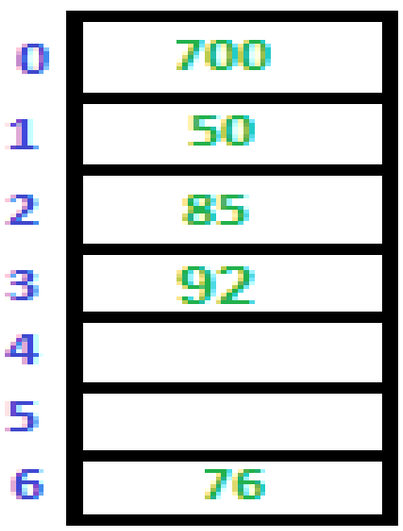
Insert 50



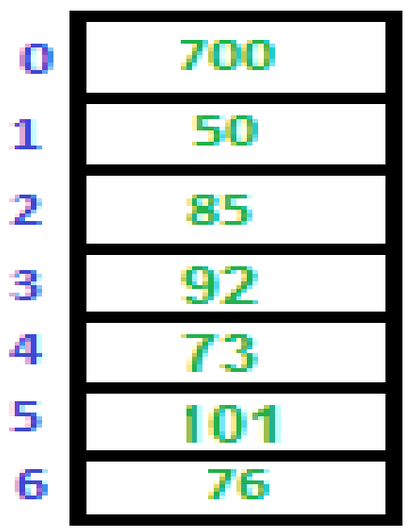
Insert 700 and 76



Insert 85: Collision Occurs, insert 85 at next free slot.



Insert 92, collision occurs as 50 is there at index 1. Insert at next free slot



Insert 73 and 101

Open Addressing Techniques

2. Quadratic Probing-

- When collision occurs, we probe for i2‘th bucket in ith iteration.
- We keep probing until an empty bucket is found.

Open Addressing Techniques

3. Double Hashing-

- We use another hash function hash2(x) and look for i * hash2(x) bucket in ith iteration.
- It requires more computation time as two hash functions need to be computed.
- Double hashing is a collision resolving technique in Open Addressed Hash tables. Double hashing uses the idea of applying a second hash function to key when a collision occurs.

Open Addressing Techniques

3. Double Hashing-

Advantages of Double hashing

1. The advantage of Double hashing is that it is one of the best form of probing, producing a uniform distribution of records throughout a hash table.
2. This technique does not yield any clusters.
3. It is one of effective method for resolving collisions.

Open Addressing Techniques

3. Double Hashing-

Lets say, Hash1 (key) = key % 13

Hash2 (key) = 7 – (key % 7)

Hash1(19) = 19 % 13 = 6

Hash1(27) = 27 % 13 = 1

Hash1(36) = 36 % 13 = 10

Hash1(10) = 10 % 13 = 10

Hash2(10) = 7 – (10%7) = 4

(Hash1(10) + 1*Hash2(10))%13= 1

Collision

(Hash1(10) + 2*Hash2(10))%13= 5

Open Addressing Techniques

Separate Chaining	Open Addressing
Keys are stored inside the hash table as well as outside the hash table.	All the keys are stored only inside the hash table.
The number of keys to be stored in the hash table can even exceed the size of the hash table.	The number of keys to be stored in the hash table can never exceed the size of the hash table.
Deletion is easier.	Deletion is difficult.
Extra space is required for the pointers to store the keys outside the hash table.	No extra space is required.
Cache performance is poor. This is because of linked lists which store the keys outside the hash table.	Cache performance is better. This is because here no linked lists are used.
Some buckets of the hash table are never used which leads to wastage of space.	Buckets may be used even if no key maps to those particular buckets.

Rehashing

Rehashing is a collision resolution technique.

Rehashing is a technique in which the table is resized, i.e., the size of table is doubled by creating a new table. It is preferable is the total size of table is a prime number. There are situations in which the rehashing is required.

- When table is completely full
- With quadratic probing when the table is filled half.
- When insertions fail due to overflow.

In such situations, we have to transfer entries from old table to the new table by re computing their positions using hash functions

Rehashing

Hash Table with input 7, 16, 33, 42

h(x) = x mod 7

0	7
1	
2	16
3	
4	
5	33
6	42

Insert 22

0	7
1	22
2	16
3	
4	
5	33
6	42

Rehashing

h(x) = x mod 17

0	
1	
2	
3	
4	
5	22
6	
7	7
8	42
9	
10	
11	
12	
13	
14	
15	33
16	16

Rehashing

As the name suggests, rehashing means hashing again. Basically, when the load factor increases to more than its pre-defined value (default value of load factor is 0.75), the complexity increases. So to overcome this, the size of the array is increased (doubled) and all the values are hashed again and stored in the new double sized array to maintain a low load factor and low complexity.

Rehashing

Why rehashing?

Rehashing is done because whenever key value pairs are inserted into the map, the load factor increases, which implies that the time complexity also increases as explained above. This might not give the required time complexity of O(1).

Hence, rehash must be done, increasing the size of the bucketArray so as to reduce the load factor and the time complexity

Rehashing

How Rehashing is done?

Rehashing can be done as follows:

- For each addition of a new entry to the map, check the load factor.
- If it's greater than its pre-defined value (or default value of 0.75 if not given), then Rehash.
- For Rehash, make a new array of double the previous size and make it the new bucketarray.
- Then traverse to each element in the old bucketArray and call the insert() for each so as to insert it into the new larger bucket array.

Extensible/Extendible Hashing

1. The dynamic hashing method is used to overcome the problems of static hashing like bucket overflow.
2. In this method, data buckets grow or shrink as the records increases or decreases. This method is also known as **Extendable hashing method**.
3. This method makes hashing dynamic, i.e., it allows insertion or deletion without resulting in poor performance.

Extensible/Extendible Hashing

- **How to search a key**

1. First, calculate the hash address of the key.
2. Check how many bits are used in the directory, and these bits are called as i.
3. Take the least significant i bits of the hash address. This gives an index of the directory.
4. Now using the index, go to the directory and find bucket address where the record might be.

Extensible/Extendible Hashing

- **How to insert a new record**

1. Firstly, you have to follow the same procedure for retrieval, ending up in some bucket.
2. If there is still space in that bucket, then place the record in it.
3. If the bucket is full, then we will split the bucket and redistribute the records.

Extensible/Extendible Hashing

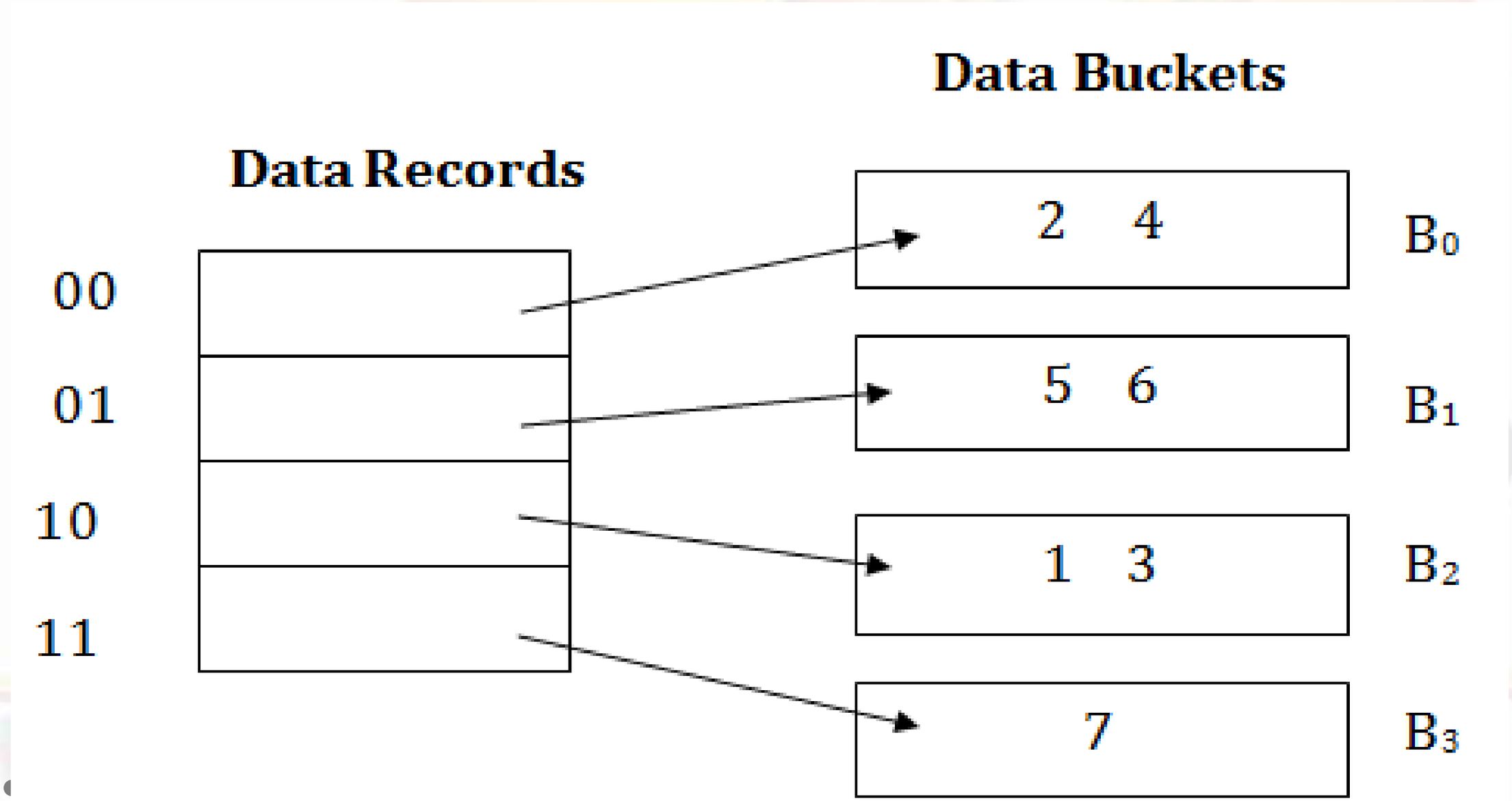
- **For example :**

Consider the following grouping of keys into buckets, depending on the prefix of their hash address:

Key	Hash address
1	11010
2	00000
3	11110
4	00000
5	01001
6	10101
7	10111

Extensible/Extendible Hashing

The last two bits of 2 and 4 are 00. So it will go into bucket B0. The last two bits of 5 and 6 are 01, so it will go into bucket B1. The last two bits of 1 and 3 are 10, so it will go into bucket B2. The last two bits of 7 are 11, so it will go into B3.

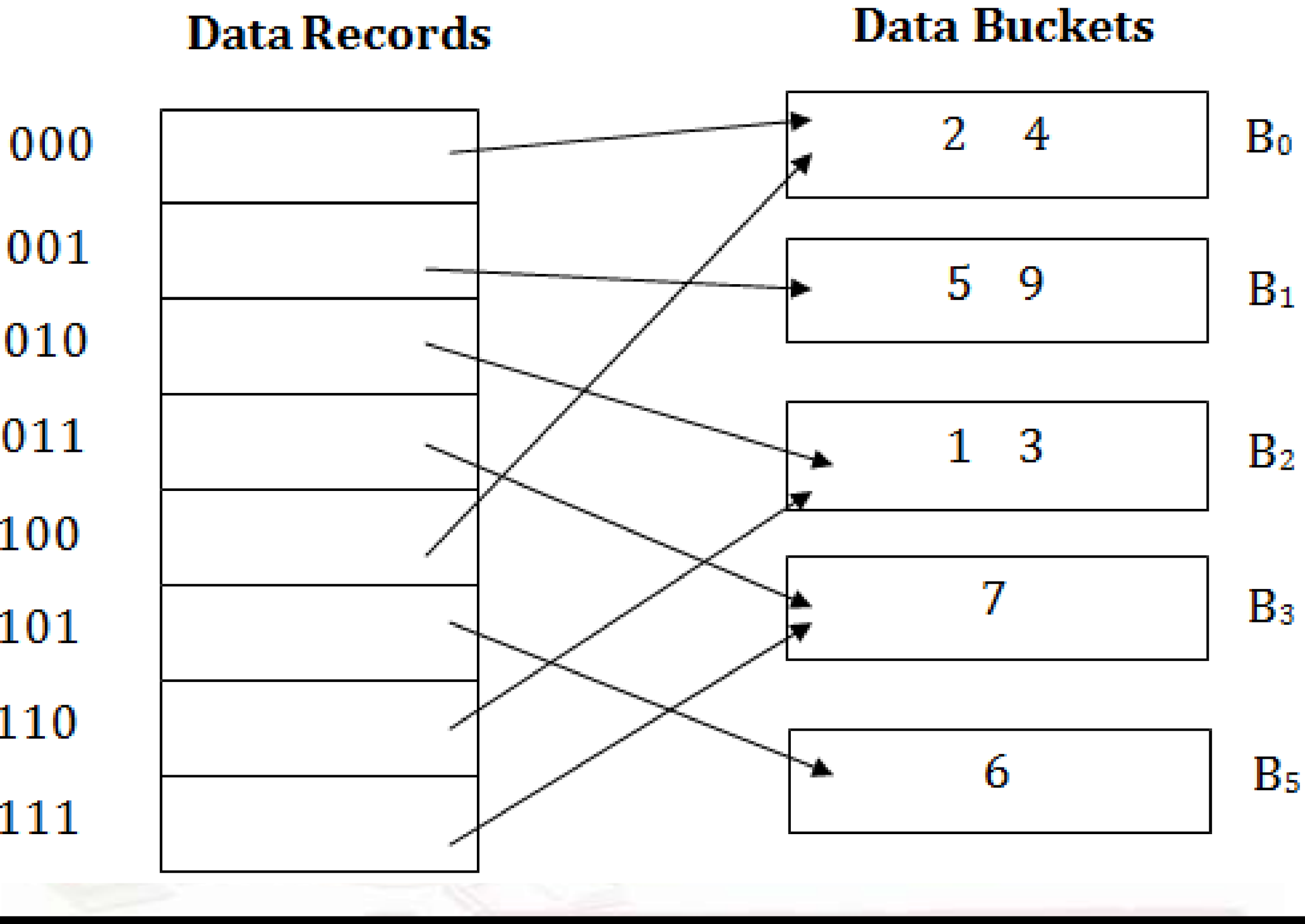


Extensible/Extendible Hashing

Insert key 9 with hash address 10001 into the above structure:

1. Since key 9 has hash address 10001, it must go into the first bucket. But bucket B1 is full, so it will get split.
2. The splitting will separate 5, 9 from 6 since last three bits of 5, 9 are 001, so it will go into bucket B1, and the last three bits of 6 are 101, so it will go into bucket B5.
3. Keys 2 and 4 are still in B0. The record in B0 pointed by the 000 and 100 entry because last two bits of both the entry are 00.
4. Keys 1 and 3 are still in B2. The record in B2 pointed by the 010 and 110 entry because last two bits of both the entry are 10.
5. Key 7 are still in B3. The record in B3 pointed by the 111 and 011 entry because last two bits of both the entry are 11.

Extensible/Extendible Hashing



Advantages of Extensible Hashing

1. In this method, the performance does not decrease as the data grows in the system. It simply increases the size of memory to accommodate the data.

2. In this method, memory is well utilized as it grows and shrinks with the data. There will not be any unused memory lying.

3. This method is good for the dynamic database where data grows and shrinks frequently

Disadvantages of Extensible Hashing

1. In this method, if the data size increases then the bucket size is also increased.
2. In this case, the bucket overflow situation will also occur. But it might take little time to reach this situation than static hashing.

Skip List

1. A skip list is a probabilistic data structure.

2. The skip list is used to store a sorted list of elements or data with a linked list.

3. It allows the process of the elements or data to view efficiently. In one single step, it skips several elements of the entire list, which is why it is known as a skip list.

Skip List

4. The skip list is an extended version of the linked list.

5. It allows the user to search, remove, and insert the element very quickly.

6. It consists of a base list that includes a set of elements which maintains the link hierarchy of the subsequent elements.

Skip List

Skip list structure

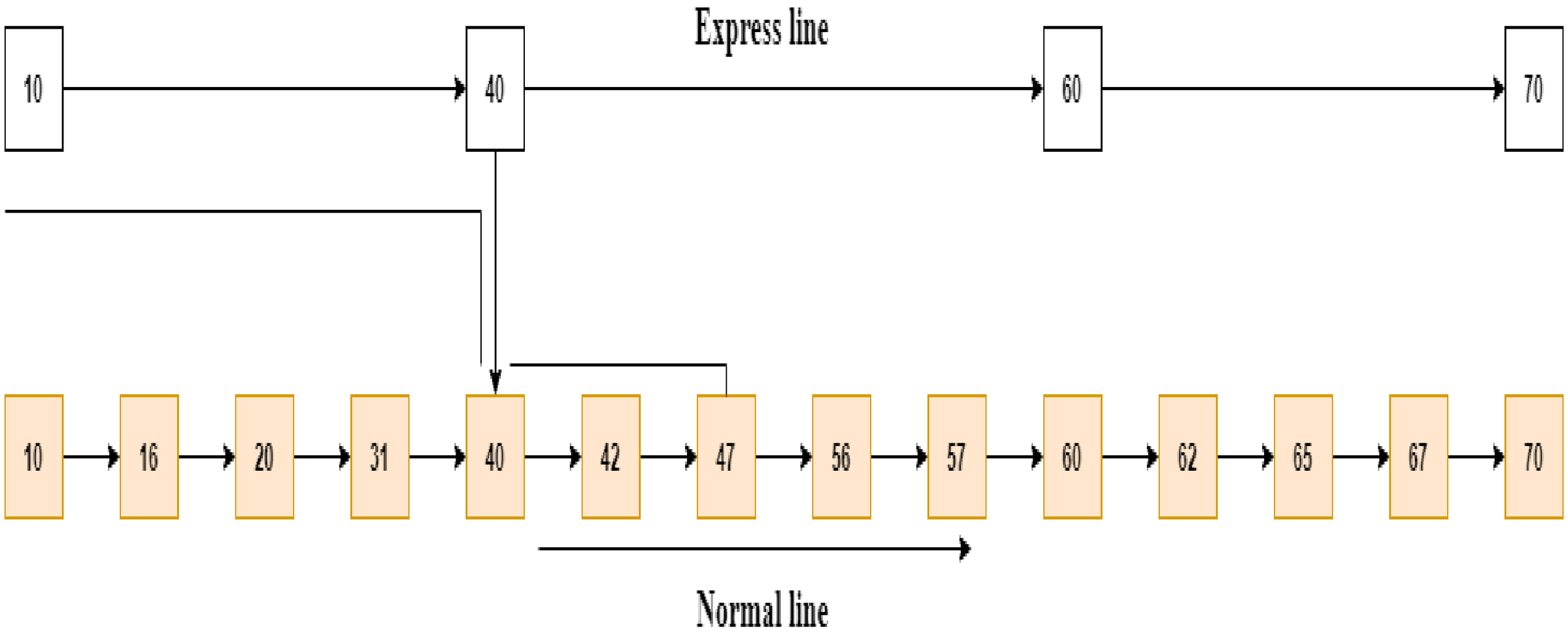
It is built in two layers: The lowest layer and Top layer.

The lowest layer of the skip list is a common sorted linked list, and the top layers of the skip list are like an "express line" where the elements are skipped.

Skip List

- Let's take an example to understand the working of the skip list. In this example, we have 14 nodes, such that these nodes are divided into two layers, as shown in the diagram.
- The lower layer is a common line that links all nodes, and the top layer is an express line that links only the main nodes, as you can see in the diagram.
- Suppose you want to find 47 in this example. You will start the search from the first node of the express line and continue running on the express line until you find a node that is equal a 47 or more than 47.
- You can see in the example that 47 does not exist in the express line, so you search for a node of less than 47, which is 40. Now, you go to the normal line with the help of 40, and search the 47, as shown in the diagram.

Skip List


Express line
10
40
60
70
10
16
20
31
40
42
47
56
57
60
62
65
67
70
Normal line

Skip List

Skip List Basic Operations

There are the following types of operations in the skip list.

1. **Insertion operation:** It is used to add a new node to a particular location in a specific situation.

2. **Deletion operation:** It is used to delete a node in a specific situation.

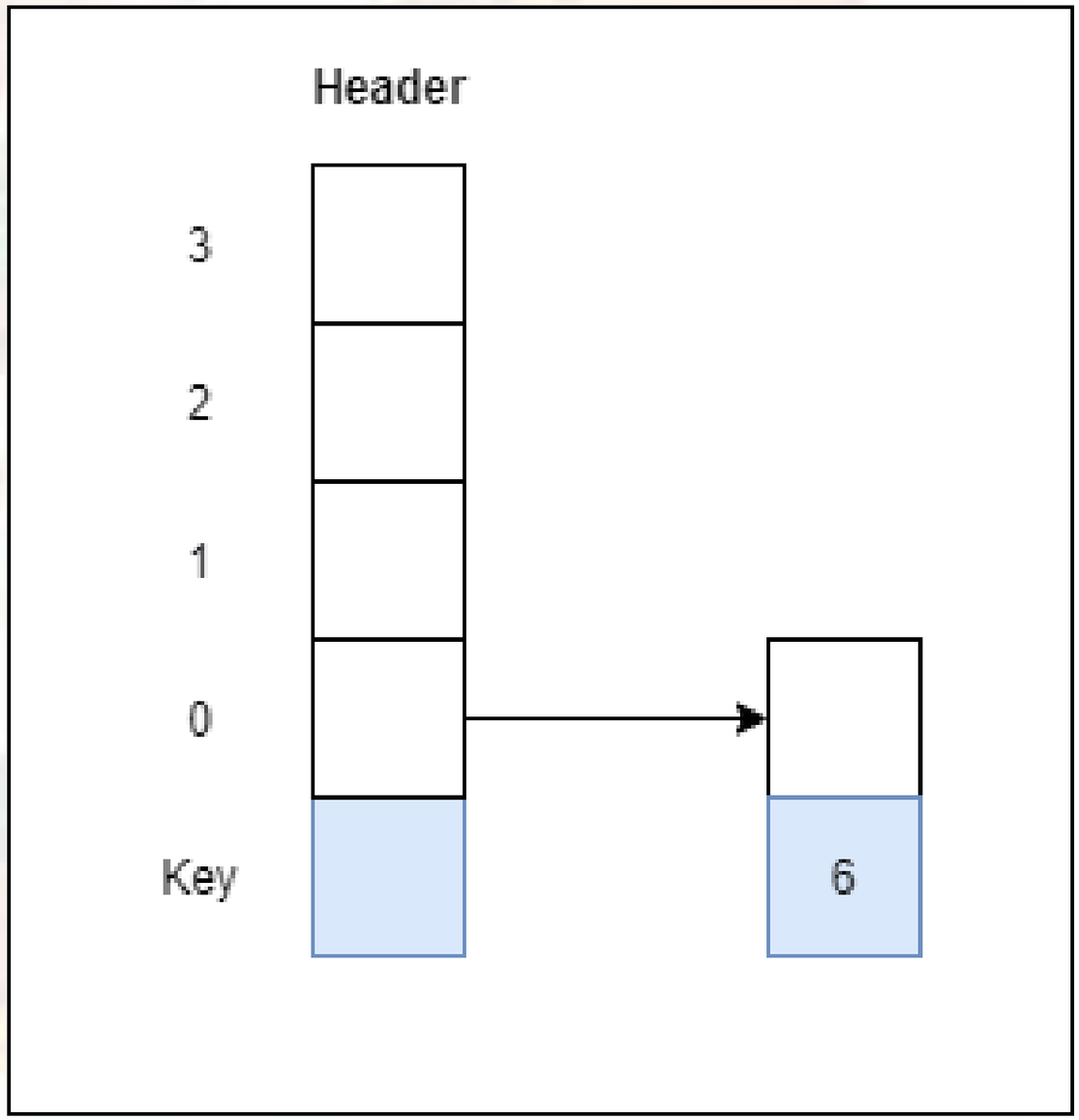
3. Se**arch Operation**: The search operation is used to search a particular node in a skip list.

Skip List

Example 1: Create a skip list, we want to insert these following keys in the empty skip list.

1. **6 with level 1.**
2. 29 with level 1.
3. 22 with level 4.
4. 9 with level 3.
5. 17 with level 1.
6. 4 with level 2.

Step 1: Insert 6 with level 1

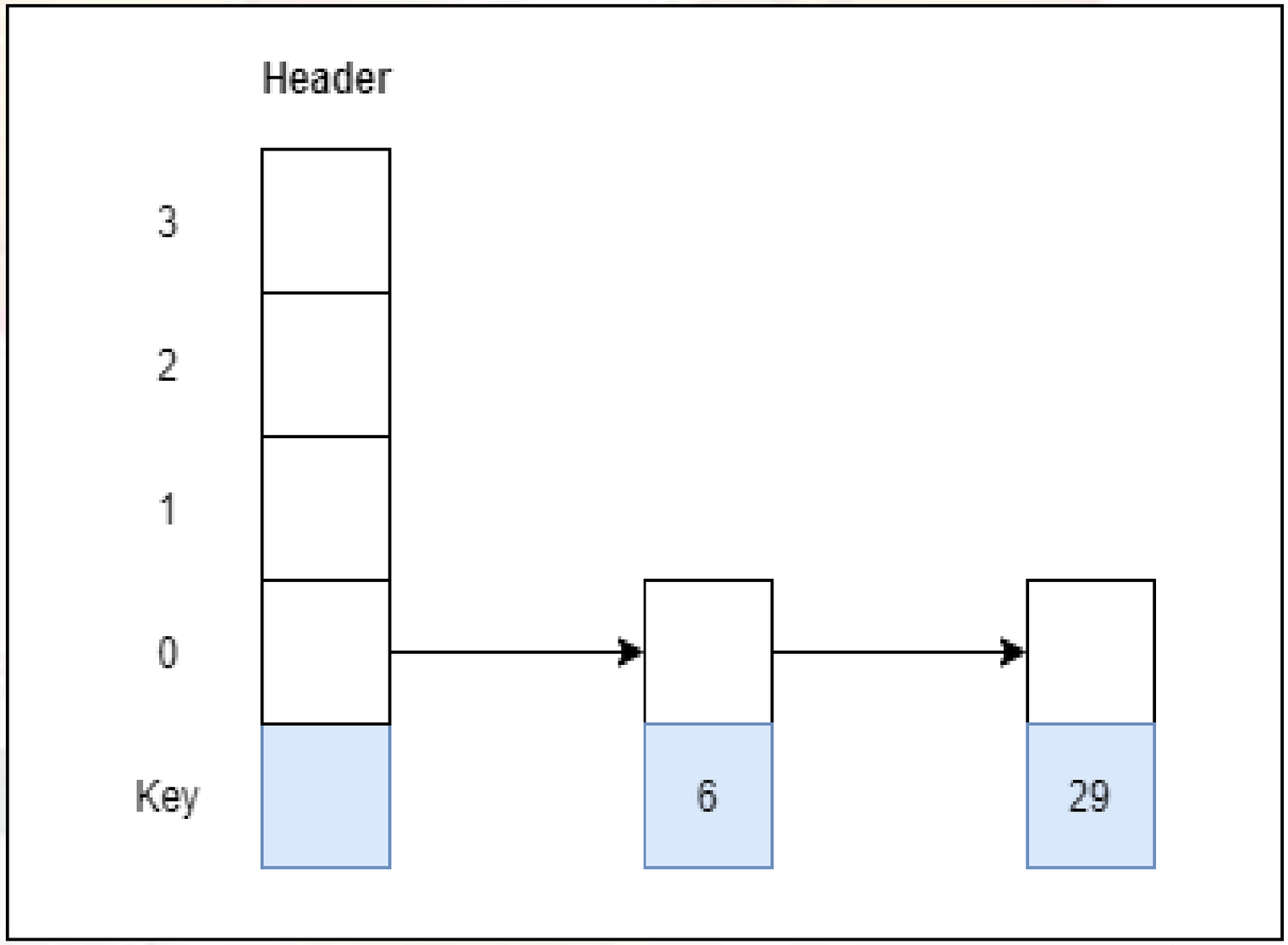


Skip List

Example 1: Create a skip list, we want to insert these following keys in the empty skip list.

1. 6 with level 1.
2. **29 with level 1.**
3. 22 with level 4.
4. 9 with level 3.
5. 17 with level 1.
6. 4 with level 2.

Step 2: Insert 29 with level 1

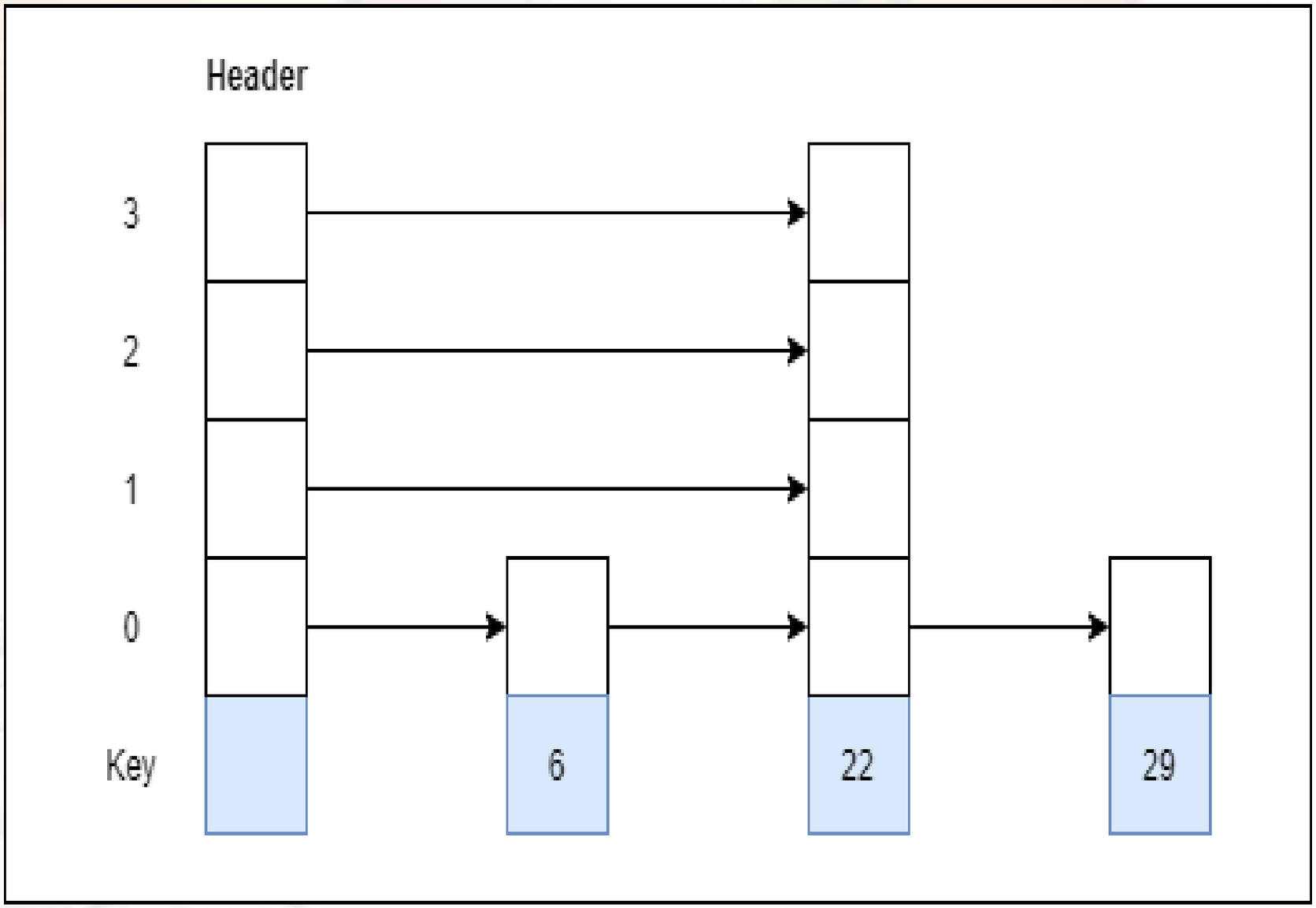


Skip List

Example 1: Create a skip list, we want to insert these following keys in the empty skip list.

1. 6 with level 1.
2. 29 with level 1.
3. **22 with level 4**.
4. 9 with level 3.
5. 17 with level 1.
6. 4 with level 2.

Step 3: Insert 22 with level 4

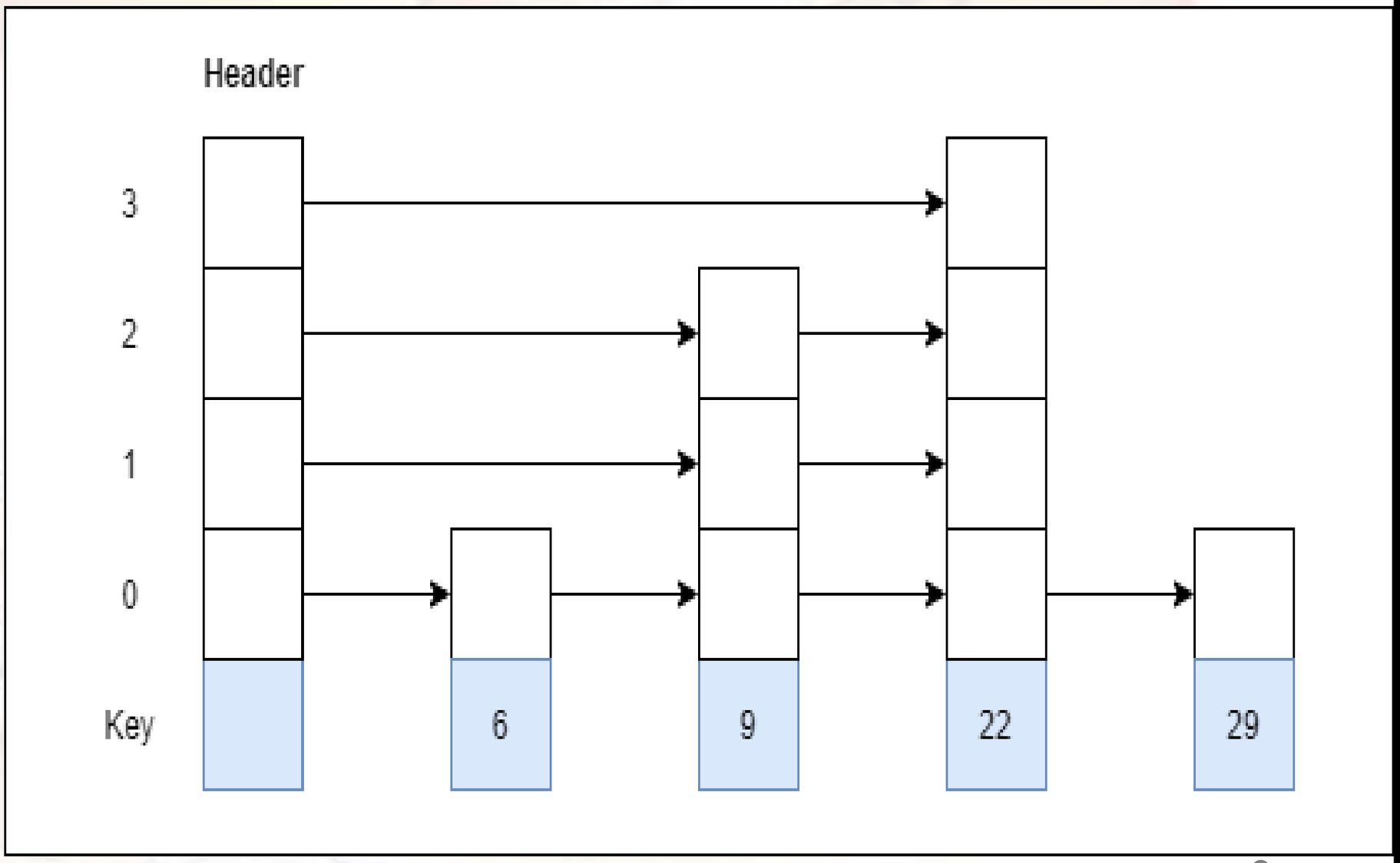


Skip List

Example 1: Create a skip list, we want to insert these following keys in the empty skip list.

1. 6 with level 1.
2. 29 with level 1.
3. 22 with level 4.
4. **9 with level 3.**
5. 17 with level 1.
6. 4 with level 2.

Step 4: Insert 9 with level 3

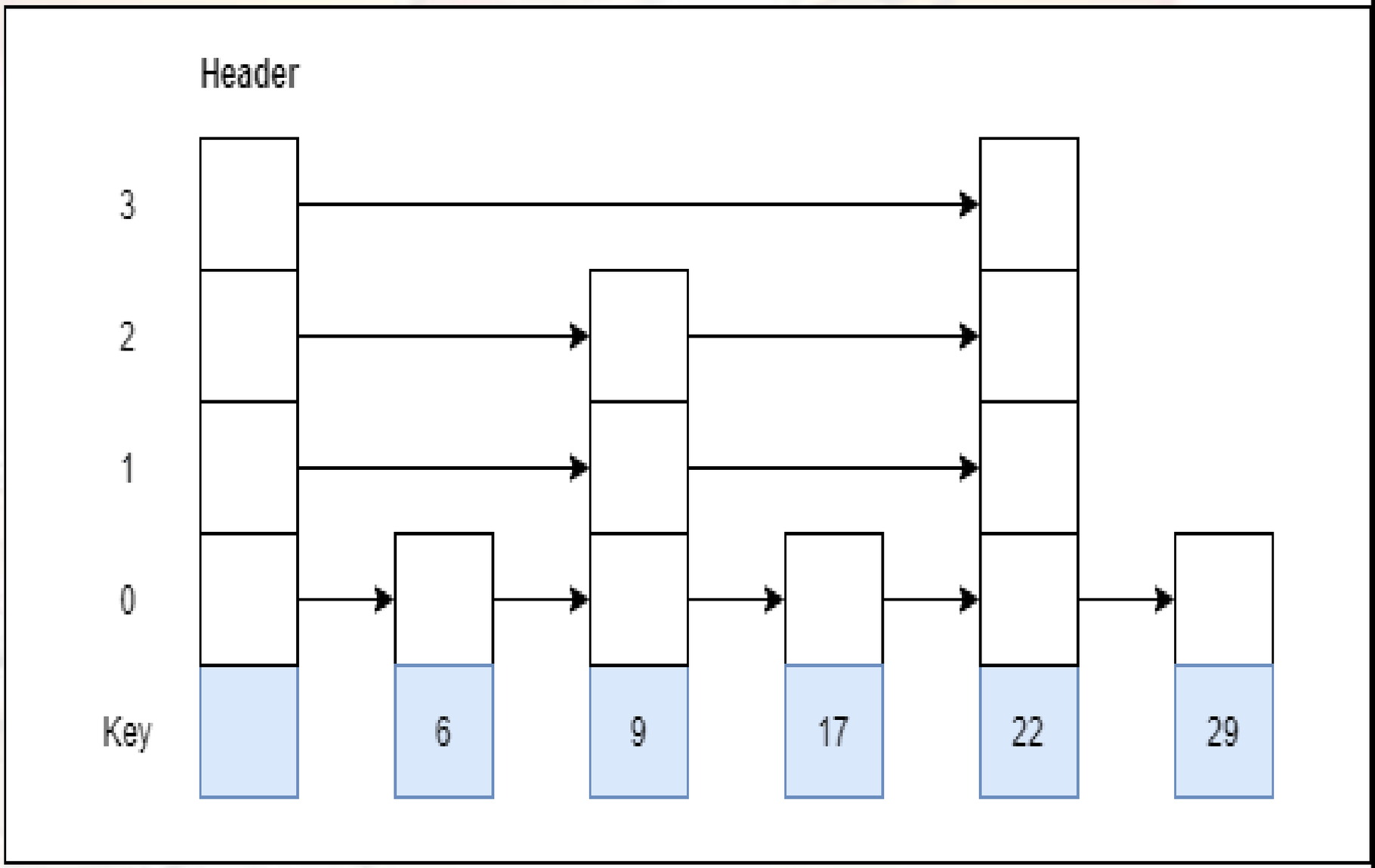


Skip List

Example 1: Create a skip list, we want to insert these following keys in the empty skip list.

1. 6 with level 1.
2. 29 with level 1.
3. 22 with level 4.
4. 9 with level 3.
5. **17 with level 1.**
6. 4 with level 2.

Step 5: Insert 17 with level 1

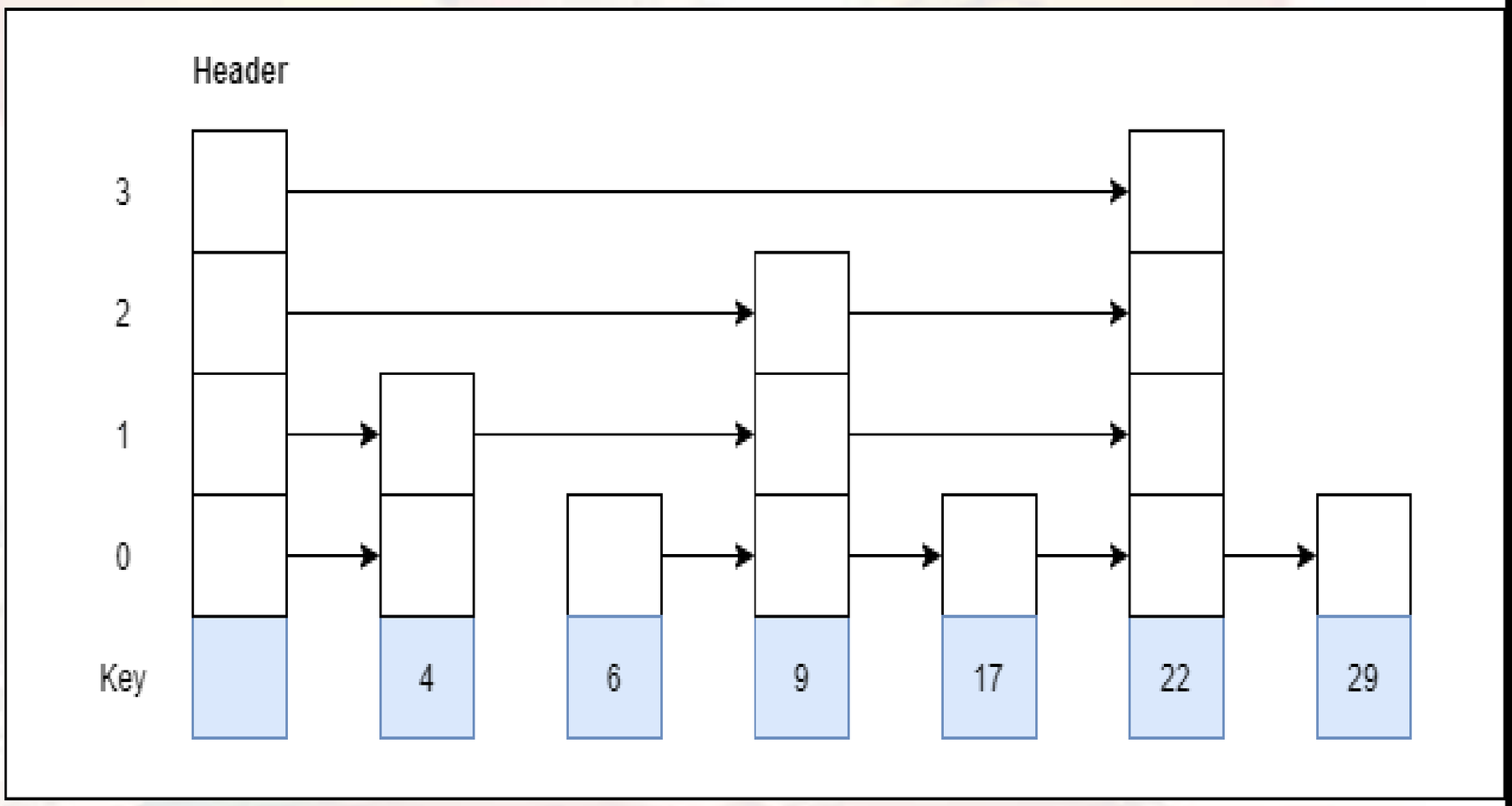


Skip List

Example 1: Create a skip list, we want to insert these following keys in the empty skip list.

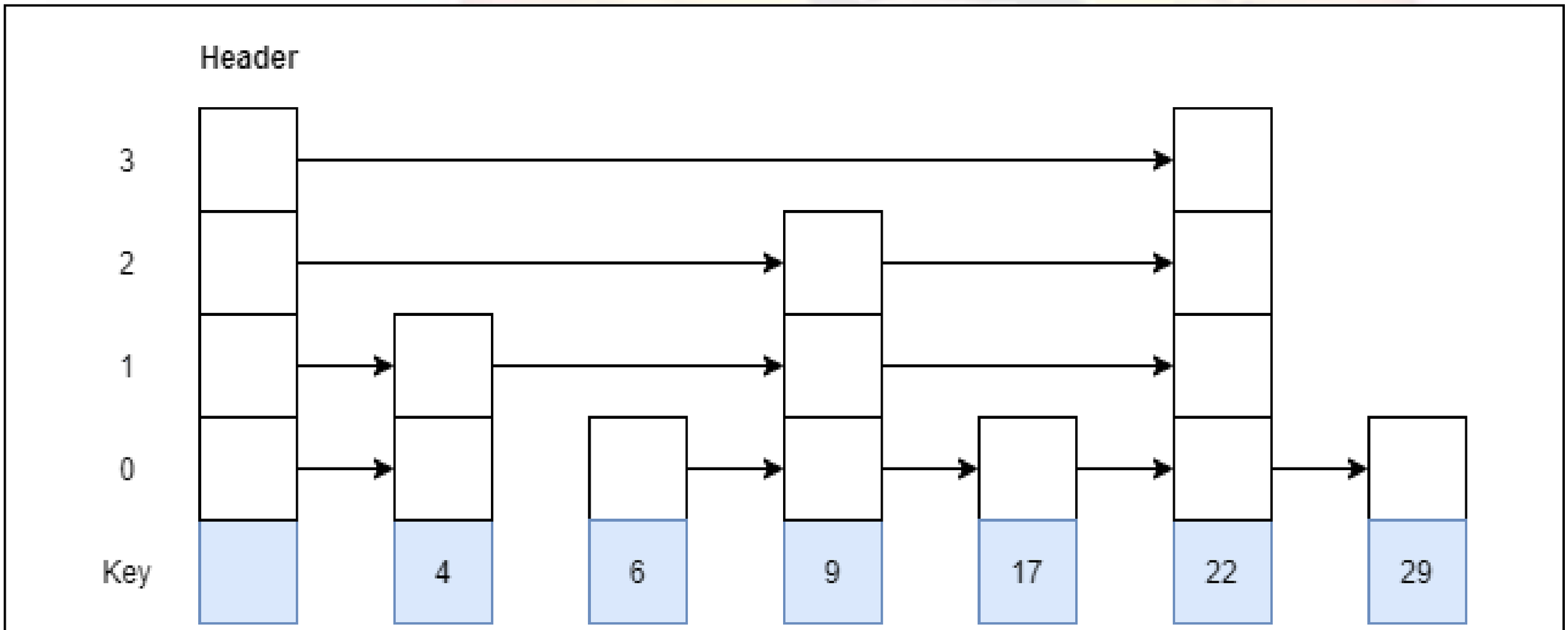
1. 6 with level 1.
2. 29 with level 1.
3. 22 with level 4.
4. 9 with level 3.
5. 17 with level 1.
6. **4 with level 2.**

Step 6: Insert 4 with level 2



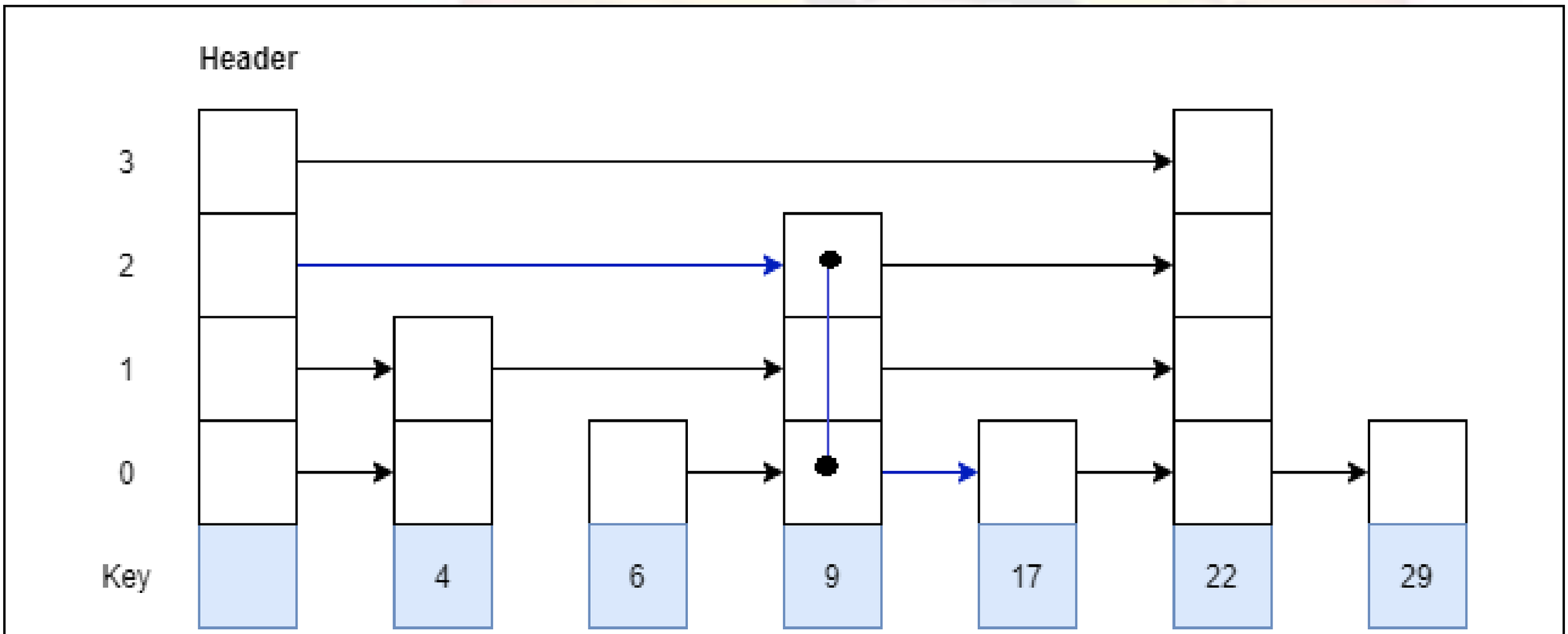
Skip List

Example 2: Consider this example where we want to search for key 17.



Skip List

Example 2: Consider this example where we want to search for key 17.



Advantages of Skip List

1. If you want to insert a new node in the skip list, then it will insert the node very fast because there are no rotations in the skip list.
2. The skip list is simple to implement as compared to the hash table and the binary search tree.
3. It is very simple to find a node in the list because it stores the nodes in sorted form.
4. The skip list algorithm can be modified very easily in a more specific structure, such as indexable skip lists, trees, or priority queues.
5. The skip list is a robust and reliable list.

Disadvantages of Skip List

1. It requires more memory than the balanced tree.
2. Reverse searching is not allowed.
3. The skip list searches the node much slower than the linked list.

Applications of Skip List

1. Skip list are used in distributed applications. In distributed systems, the nodes of skip list represents the computer systems and pointers represent network connection.
2. Skip list are used for implementing highly scalable concurrent priority queues with less lock contention (struggle for having a lock on a data item)

   https://www.geeksforgeeks.org/implementation-of-locking-in-dbms/.

Applications of Skip List

3. It is also used with **the QMap template class**. (Value-based template class that provides a dictionary)
4. The indexing of the skip list is used in running **median problems**.
5. skipdb is an open-source database format using ordered key/value pairs.

THANK YOU!!!

My Blog : https://anandgharu.wordpress.com/

Email : gharu.anand@gmail.com