UNIT 4 : HASHING

Indu Seethanatha
Computer Science & Design
K.K.Wagh Institute of Engineering Education & Research
Nashik

K. K. WAGH EDUCATION SOCIETY

K K Wagh

Syllabus Content

- Hash table Concepts-Hash function, bucket, Collision, Probe, Synonym, Overflow, Open hashing, Closed hashing, Perfect hash function, Load density, Full table, Load factor, Rehashing, Basic operations, 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, Closed addressing and Separate chaining.
- Files-Concept, Need, Primitive operations. Sequential file organization, Direct access file, Indexed sequential file organization-Concept and Primitive operations Self Study-

  SkipList- Representation, Searching

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

- So, in simpler terms, we can say that hashing is a method to store, retrieve, update data to/ from a database

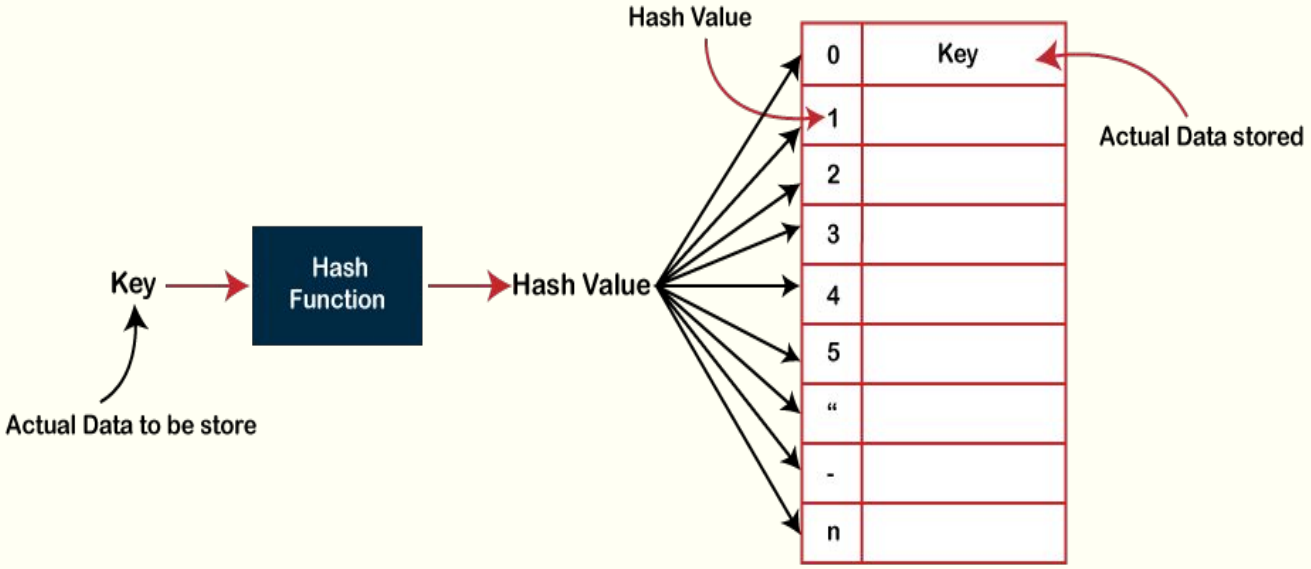
Search key- Searching a database is done using a “key”, ie, to search a student record from student database, it needs to be searched using name or roll number. This name or roll number using which we search is called the “search key”

Introduction

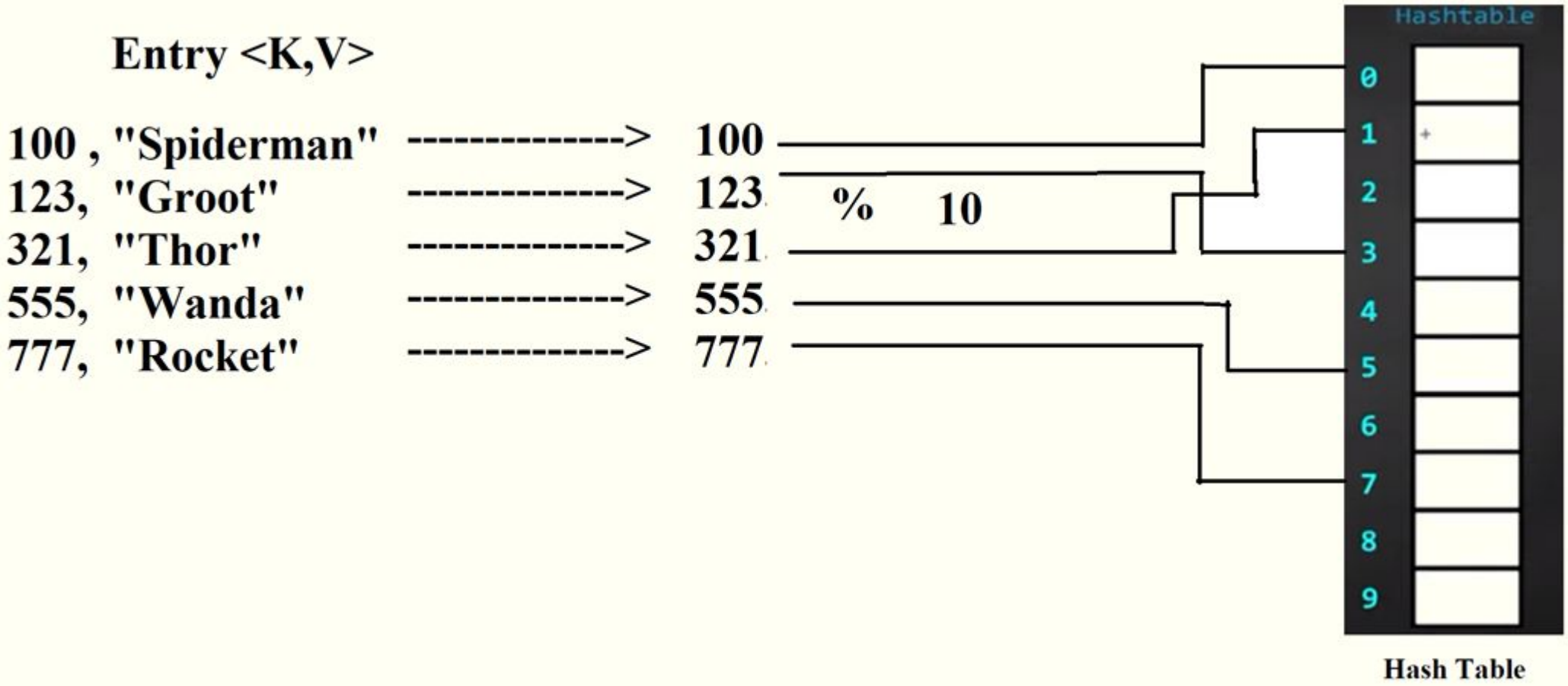
- Hashing technique was developed in order to overcome the shortcomings of linear search & binary search
- Now these search keys need to be put in a **hash table**
- Hash table is a data structure that provides a methodology to save data in a proper manner
- In a way, hash table is like an array.
- Hash table has index just like an array
- However, there in no need to scan the entire table while searching or inserting data
- For that, we use an computational algorithm called as the hash function

Hash Table

- Hash table is a data structure that provide methodology to store data in proper way
- Data will be stored & retrieved
- A hash table is an array of some fixed size, usually a prime number
- Main purpose to store data items in hash table is to find them later very easily

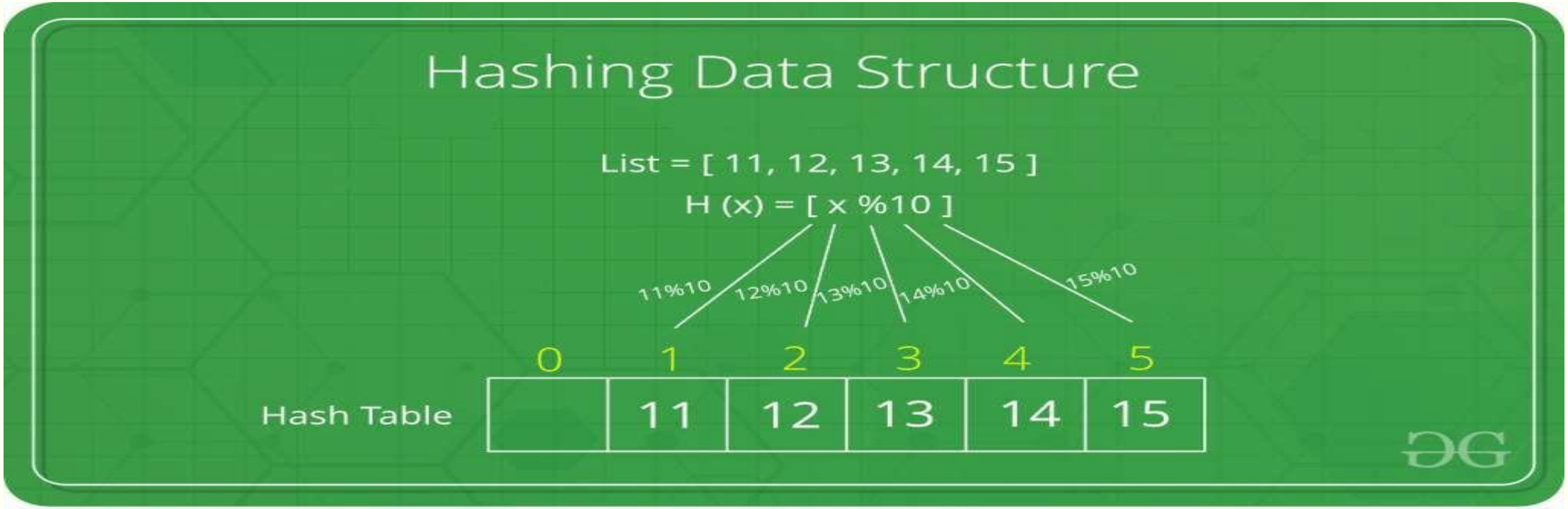


Hash Table


Entry <K,V>
100 , "Spiderman" ----------> 100
123, "Groot" ----------> 123
321, "Thor" ----------> 321
555, "Wanda" ----------> 555
777, "Rocket" ----------> 777
% 10
Hashtable
0
1
2
3
4
5
6
7
8
9
Hash Table

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.



Hashing

- Hashing is a method for storing & retrieving data in O(1) time from the database
- Mapping technique
- Hashing is a well-known technique to search any particular element among several elements.
- Hashing is like indexing : associating a key with a relative record address
- Hashing differs from indexing in two important ways:
  1. Randomizing : With hashing, the addresses generated appear to be random
  2. Collision : With hashing, two different keys may be transformed to the same address so two records may be sent to the same place in the file

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, i.e., It completes the search with constant time complexity O(1).
3. Hashing is extremely efficient, they perform very well even when working with large datasets.
4. The time taken by it to perform the search does not depend upon the total number of elements.

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.

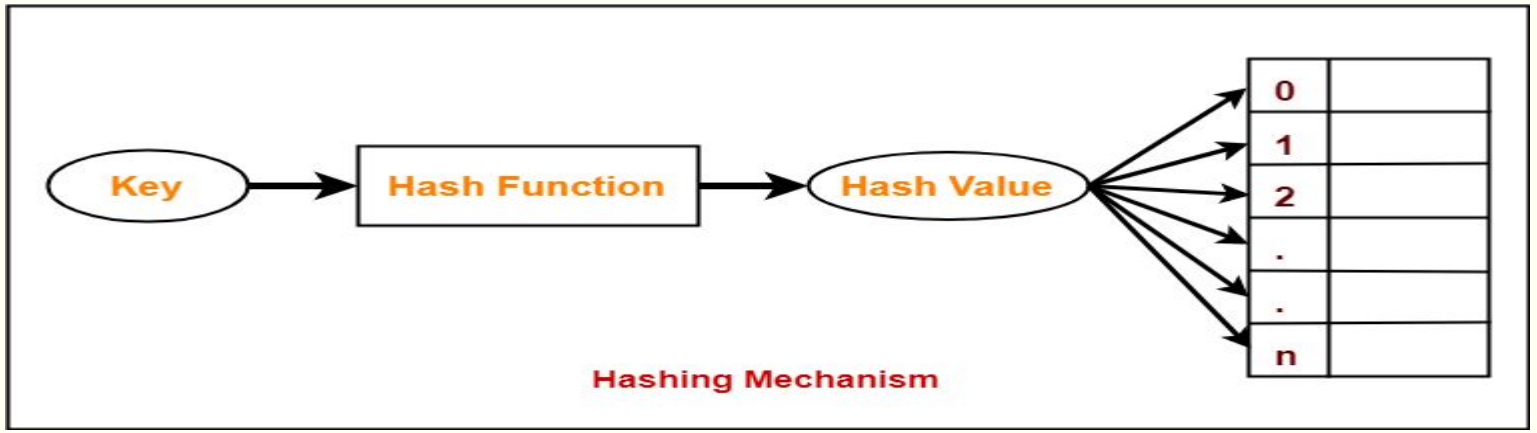
BASIC OPERATIONS OF HASH TABLE

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

Hashing Mechanism-

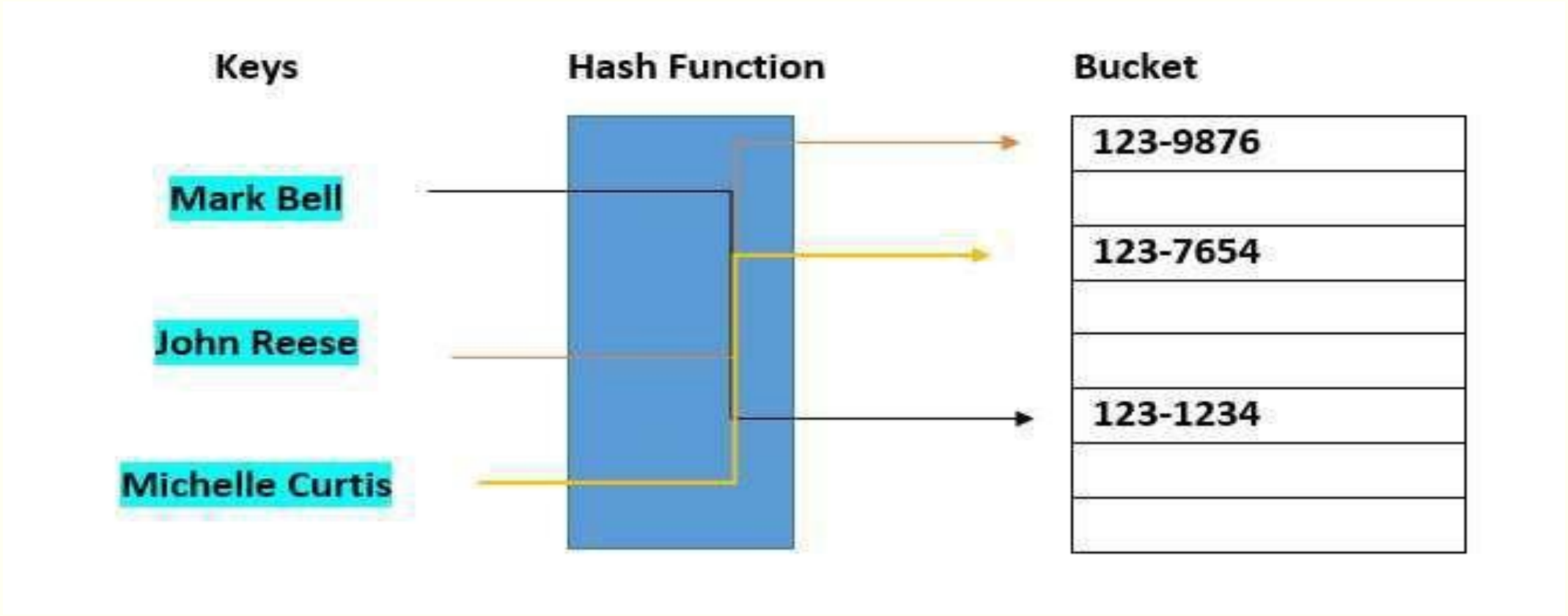
- In hashing, An array data structure called as **Hash table** is used to store the data items.
- Based on the hash key value, data items are inserted into the hash table.
- **Hash Key Value-**
- Hash key value is a special value that serves as an index for a data item.
- It indicates where the data item should be stored in the hash table.
- Hash key value is generated using a hash function.



Hashing Mechanism

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



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

Synonym

- 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

Collision

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

Open or external hashing

- When we allow records to be stored in potentially unlimited space, it is called as open or external hashing

Closed or internal hashing

- When we use fixed space for storage eventually limiting the number of records to be stored, it is called as closed or internal hashing

Home Address

- 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

Hash function

- Hash function is an arithmetic function that transforms a key into an address and the address is used for storing and retrieving a record

Perfect hash function

- The hash function that transforms different keys into different addresses is called perfect hash function
- The worth of hash function depends on how well it avoids collision

Full Table

- Full table is the one in which all locations are occupied
- Owing to the characteristics of hash functions, there are always empty locations

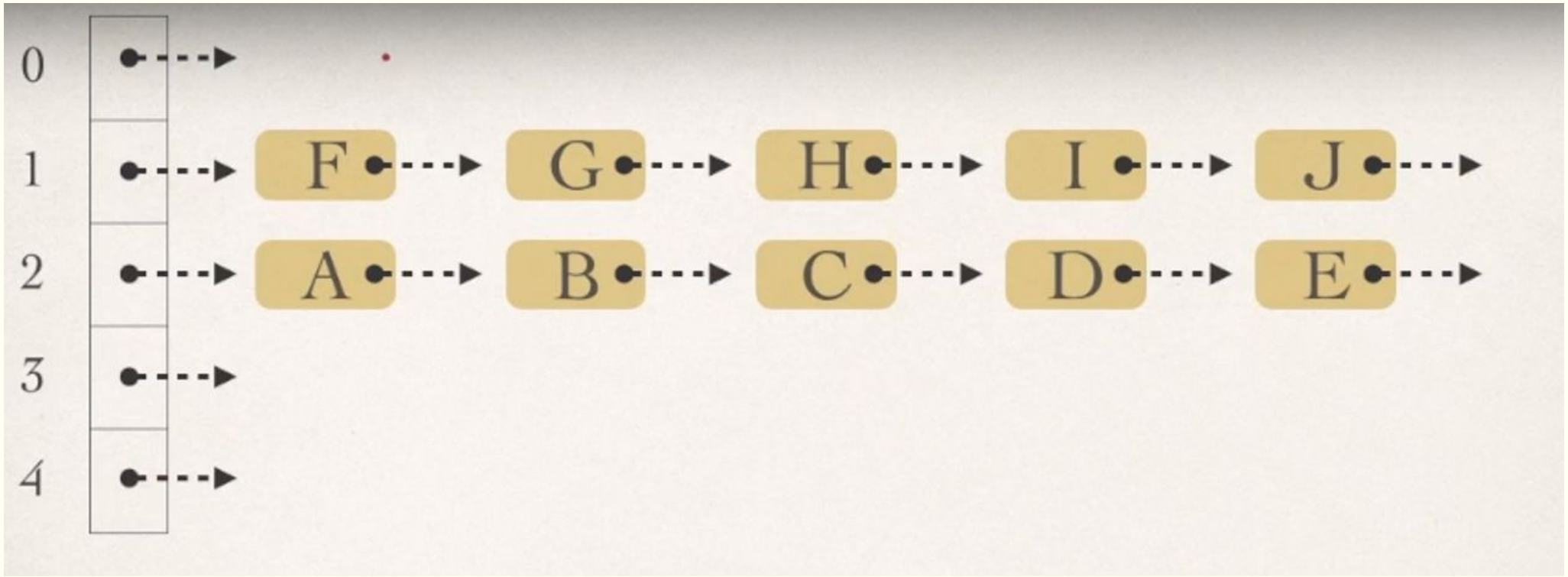
Load factor

- Load factor is the number of records stored in table divided by maximum capacity of table, expressed in terms of percentage

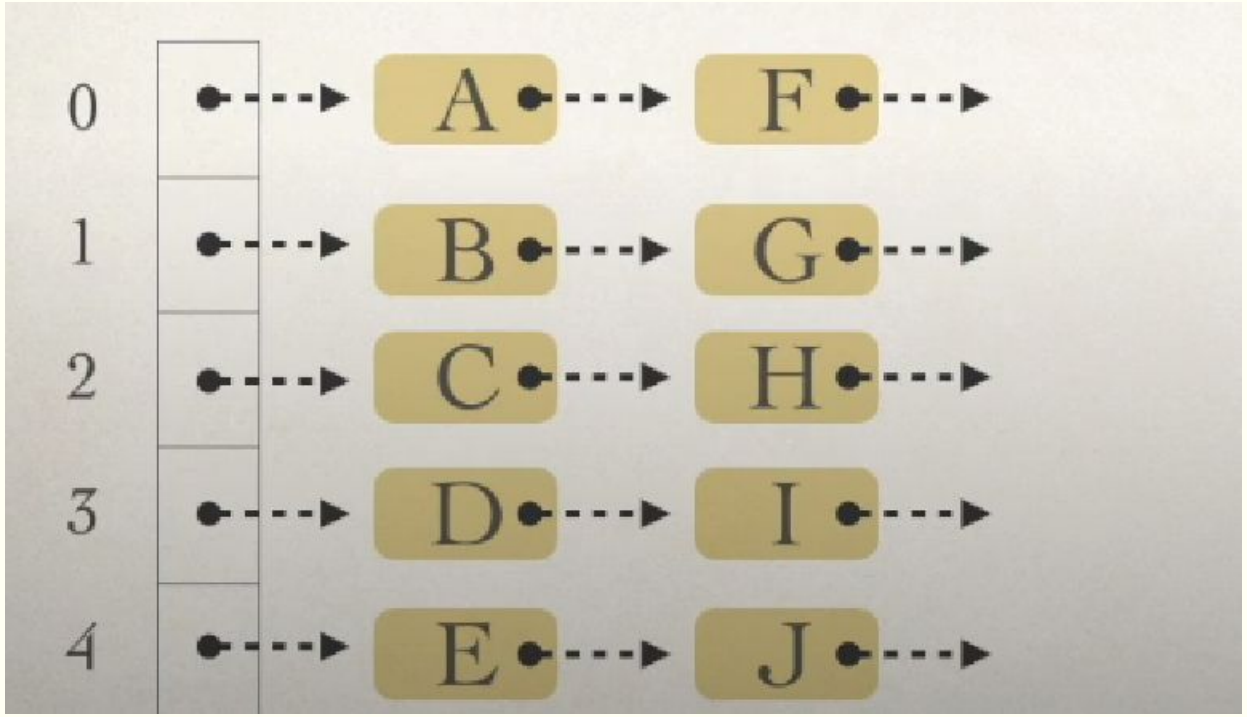
Load factor

- Load factor is the number of records stored in table divided by maximum capacity of table
- The higher the load factor, slower the retrieval
- Load factor is denoted by alpha(α)

Load factor- is basically the average of load value of the buckets



Load factor- is basically the average of load value of the buckets



Load factor is defined as average number of elements in a slot.

$$\frac{10}{5} = \frac{\text{Number of elements}}{\text{Number of slots}}$$

Given a hash table T with 25 slots that stores 2000 elements, the load factor α for T is

_____

=

Features of a Good Hash Function

- The average performance of hashing depends on how the hash function distributes the set of keys among the slots
- Assumption is that any given record is equally likely to hash into any of the slots, independently of whether any other record has been already hashed to it or not
- This assumption is called as **simple uniform hashing**
- A good hash function is the one which satisfies the assumption of simple uniform hashing

Features of a good hashing function

- Addresses generated from the key are uniformly and randomly distributed
- Small variations in the value of key will cause large variations in the record addresses to distribute records (with similar keys) evenly
- The hashing function must minimize the collision

Hash Function-

- Hash function is a function that maps any big number or string to a small integer value.
- Hash function takes the data item as an input and returns a small integer value as an output.
- The small integer value is called as a **hash value.**
- Hash value of the data item is then used as an **index** for storing it into the hash table.
- The address generated by hashing function is called as **home address**
- 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
- All home addresses address to particular area of memory and that area is called as **prime area**

PROPERTIES OF HASH FUNCTION

1) Hash function should be simple to computer.
2) Number of collision should be less
3) The hash function uses all the input data
4) The hash function "uniformly" distributes the data across the entire set of possible hash values.
5) The hash function generates very different hash values for similar strings.

The properties of a good hash function are-

It is efficiently computable.

It minimizes the number of collisions.

It distributes the keys uniformly over the table.

PROPERTIES OF A GOOD HASH FUNCTION

The properties of a good hash function are-

1) It is efficiently computable.
2) It minimizes the number of collisions.
3) It distributes the keys uniformly over the table.

Types of Hash Functions-

- Division method
- Folding method
- Mid square method

Hash Functions

:

- Division Method
- **Multiplication Method**
- **Extraction Method**
- Mid-Square Hashing
- Folding Technique
- **Rotation**
- **Universal Hashing**

Division Method

- One of the required features of the hash function is that the resultant index must be within the table index range
- One simple choice for a hash function is to use the modulus division indicated as MOD (the operator % in C/C++)
- The function returns an integer
- If any parameter is NULL, the result is NULL
- Hash(Key) = Key % M

Mid-Square Hashing

- The mid-square hashing suggests to take square of the key and extract the middle digits of the squared key as address
- The difficulty is when the key is large. As the entire key participates in the address calculation, if the key is large, then it is very difficult to store the square of it as the square of key should not exceed the storage limit
- So mid-square is used when the key size is less than or equal to 4 digits
- The difficulty of storing larger numbers square can be overcome if for squaring we use few of digits of key instead of the whole key

Mid-Square Hashing

key=22,11,13,14

$H(key)=key^2$

H(22)= 22 x 22=484

H(11)= 11 x 11=121

H(13)= 13 x 13=169

H(14)= 14 x 14=196

0	
1	
2	11
3	
4	
5	
6	13
7	
8	22
9	14

Consider there are 68 employees in a company. So here, we consider 2 digit elements from the middle of k^2

H(1234)=01522756
H(1437)=02064969
H(3438)=11819844
H(4678)=21883684
H(5941)=35295481

00	
..	
19	3438
22	1234
..	
64	1437
..	
83	4678
..	
95	5941
..	
99	

- Suppose k = 1234. Then let's find the hash value for a hash table of size 100.
- Since the index of hash table varies between 0 and 99, we can choose r = 2 .

Multiplication Method

- The following steps are involved in multiplication method.

1.Choose a constant A such that $0 < A < 1$

2.Multiply k by A

3.Extract the fractional part

4.Multiply the result by the size of hash table (m). **And take floor of the result**

- The hash function can be given as:
- $h(k) = \lfloor m (kA \bmod 1) \rfloor$

Multiplication Method

- Works practically with any value of A, Knuth recommends

$$A \simeq \frac{\sqrt{5}-1}{2} = 0.6180339887\ldots \ (\textit{Golden Ratio})$$

1. Start with the equation $x^2 = x + 1$.
2. Rearrange the equation to $x^2 - x - 1 = 0$.
3. Use the quadratic formula to solve for x:
   $x = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$
   where $a = 1, b = -1$, and $c = -1$.
4. Substituting the values of a, b, and c, we get:
   $x = \frac{-(-1) \pm \sqrt{(-1)^2-4 \cdot 1 \cdot (-1)}}{2 \cdot 1}$
   which simplifies to:
   $x = \frac{1 \pm \sqrt{1+4}}{2}$
   $x = \frac{1 \pm \sqrt{5}}{2}$

The positive solution, $(1 + \sqrt{5})/2$, is known as the golden ratio, denoted by the Greek letter φ (phi). It's an irrational number that appears frequently in mathematics, art, architecture, and nature due to its unique and aesthetically pleasing properties.

Multiplication Method

- h(k) = ⌊ m (kA mod 1) ⌋
- Given a hash table of size 1000, map the key 12345 to an appropriate location in the hash table.
- Use A = 0.618033, m = 1000, and k = 12345

h(12345) = ⌊ 1000 (12345 x 0.618033 mod 1) ⌋

h(12345) = ⌊ 1000 (7629.617385 mod 1) ⌋

h(12345) = ⌊ 1000 (0.617385) ⌋

h(12345) = ⌊ 617.385 ⌋

h(12345) = 617

Extraction Method

- When a portion of the key is used for the address calculation, the technique is called as the extraction method

- In digit extraction, few digits are selected and extracted from the key which are used as the address

Keys and addresses using digit extraction

Key	Hashed Address
345678	357
234137	243
952671	927

Extraction Method

DIGIT EXTRACTION METHOD??

- Worst Method out of All.
- Only few digits participates not All.
- You take or extract required digits and remaining digits you leave as it is.

Folding Technique

- In folding technique, the key is subdivided into subparts that are combined or folded and then combined to form the address. There are 2 types of folding 1) Fold Shift 2) Fold Boundary

- The key k is divided into a number of parts of same length k1, k2, … , kr.
- And they are added together ignoring last carry if any.
- The hash function can be given by:
- $h(k) = k1 + k2 + \ldots + kr$
- If the hash table has a size of 1000, then there are 1000 locations in the hash table.
- To address these 1000 locations, we need at least three digits; therefore, each part of the key must have three digits except the last part which may have lesser digits.
- Given a hash table of 100 locations, calculate the hash value using folding method for keys 5678, 321, and 34567

Folding Technique

- Given a hash table of 100 locations, calculate the hash value using folding method for keys 5678, 321, and 34567

key	5678	321	34567
Parts	56 and 78	32 and 1	34, 56 and 7
Sum	134	33	97
Hash value	34 (ignore the last carry)	33	97

(5) Folding Method :

Two folding methods are used

1. Fold shift

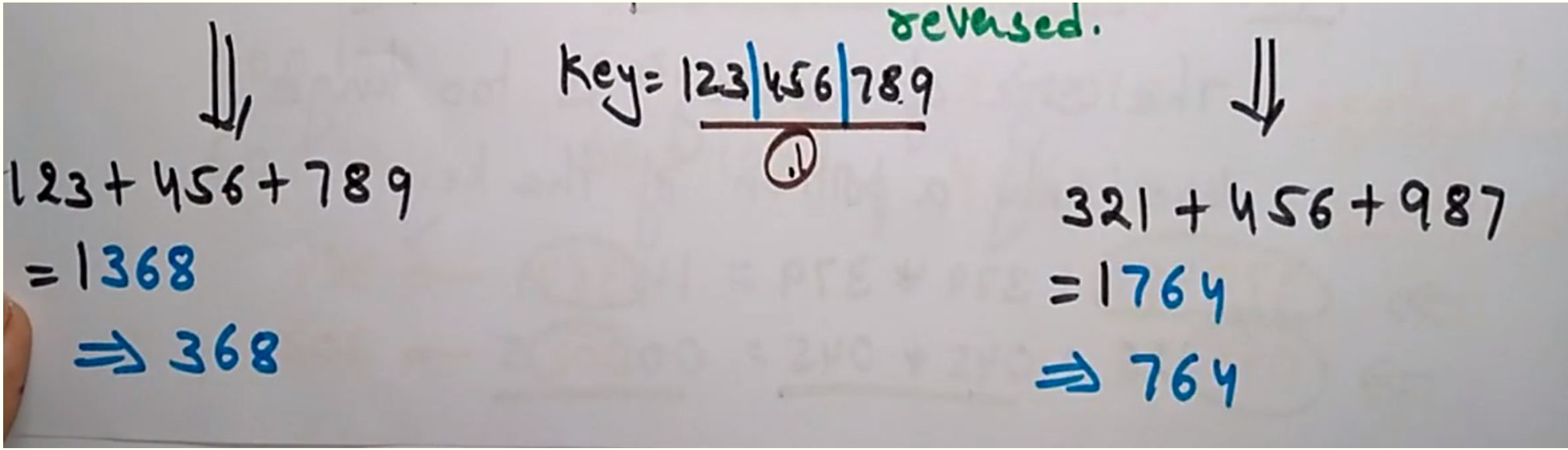
The key values is divided into parts whose size matches the size of required add and added

2. Fold boundary

The left and right numbers are folded on a fixed bounda between them and the cent The two outside values are reversed.

Key = 123 | 456 | 789

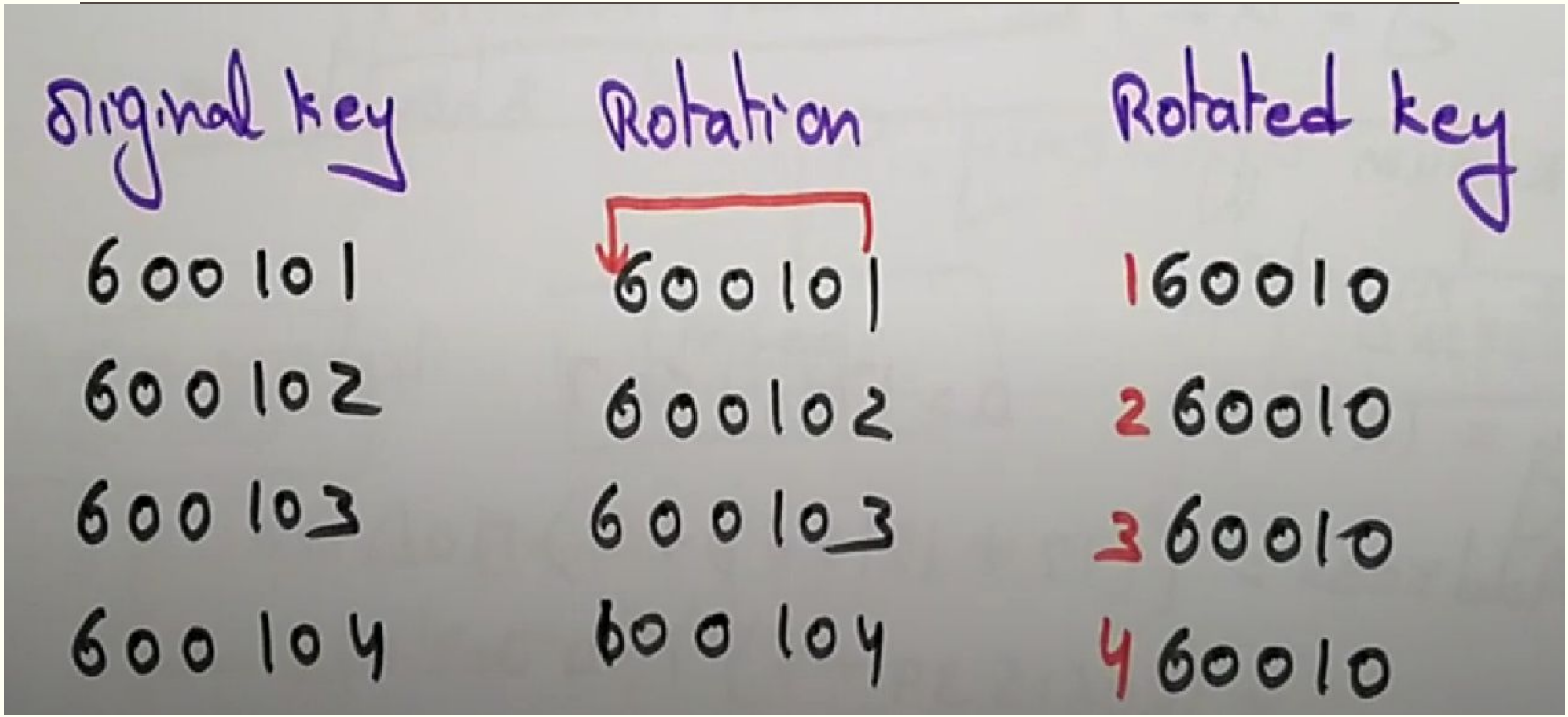
Folding Technique



Rotation

- Rotation hashing is not used by itself, its used with other hashing methods
- When keys are serial, they vary in only last digit and this leads to the creation of synonyms
- Rotating key would minimize this problem. This method is used along with other methods
- Here, the key is rotated right by one digit and then use of folding would avoid synonym
- For example,
  let the key be 120605, when it is rotated we get 512060
- Then further the address is calculated using any other hash function

Rotation


Original key
Rotation
Rotated key
600101
600102
600103
600104
600101
600102
600103
600104
160010
260010
360010
460010

Universal Hashing

- The main idea behind universal hashing is to select the hash function at random at run time from a carefully designed set of functions
- Because of randomization, the algorithm can behave differently on each execution; even for the same input
- This approach guarantees good average case performance, no matter what keys are provided as input

Example

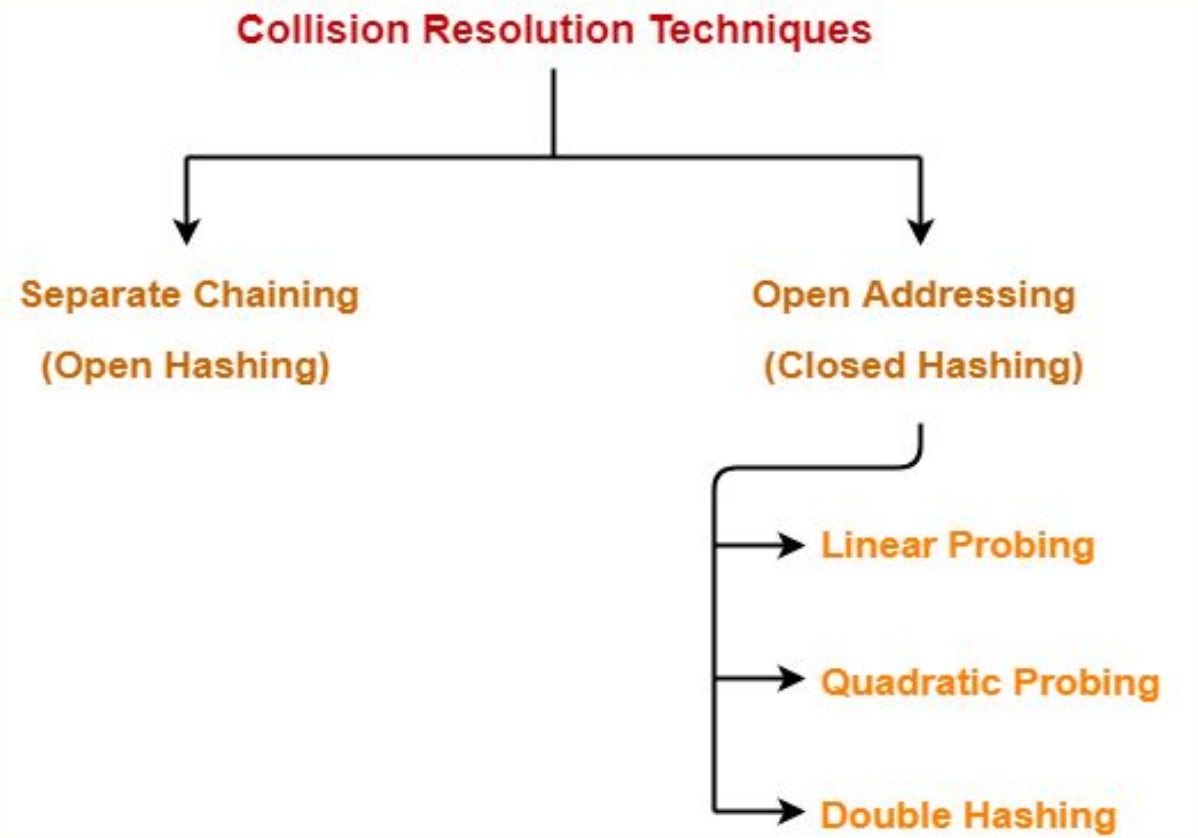
- Using the hash function ‘key mod 10’, insert the following sequence of keys in the hash table- Search Key (24, 52, 91, 67, 48, 83)

Collision in Hashing-

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

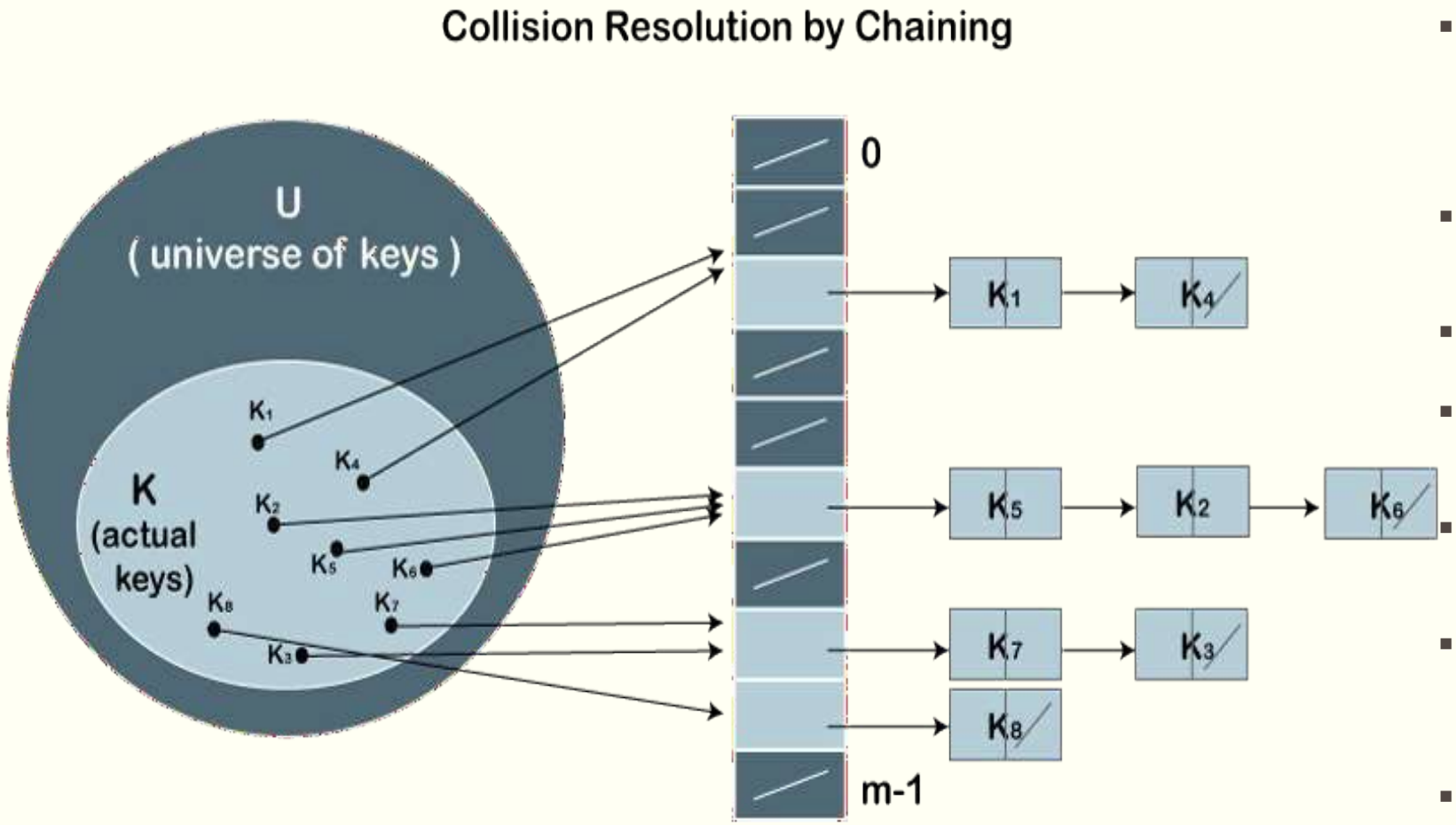
- Collision Resolution Techniques are the techniques used for resolving or handling the collision.



Collision resolution techniques

- Collision resolution techniques can be broken into two classes:
- **open hashing** (also called **separate chaining**) and
- **closed hashing** (also called **open addressing**).
- The difference between the two has to do with whether collisions are stored outside the table (open hashing), or whether collisions result in storing one of the records at another slot in the table (closed hashing).

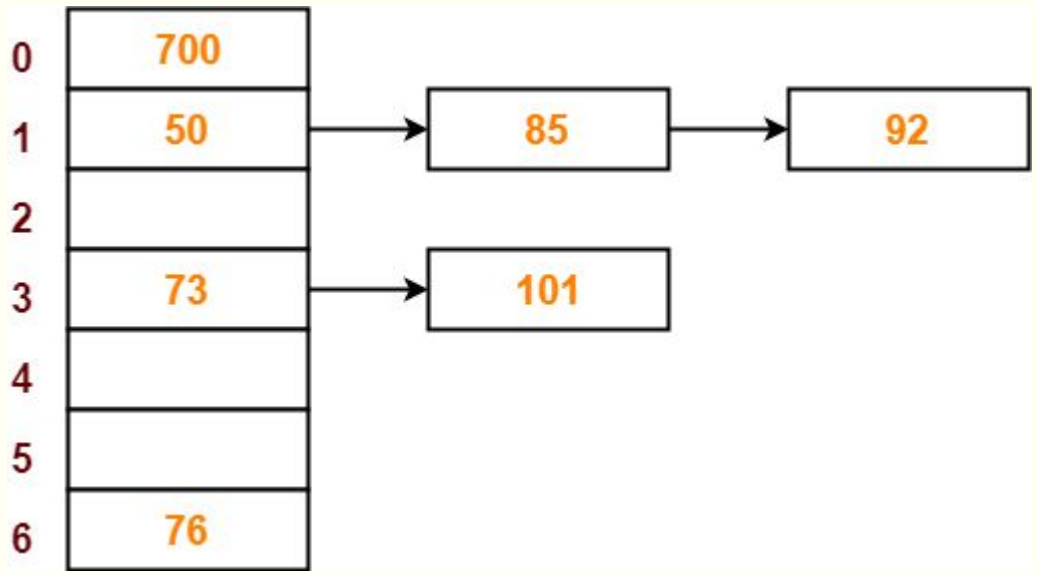
Open Hashing/ Separate Chaining



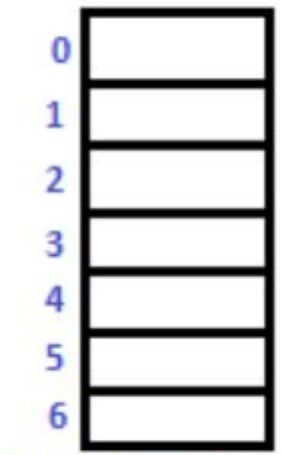
- Here, keys are stored in linked list attached to cells of hash table, i.e., a linked list is created to the slot for which collision occurs.
- The new key is then inserted in the linked list.
- External hashing
- Allows records to be stored in unlimited space
- No limitation on the size of the tables
- Each bucket in the hash table is the head of the linked list of records mapped to that bucket
- These linked lists to the slots appear like chains, hence the name **separate chaining.**

- 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.

- 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.

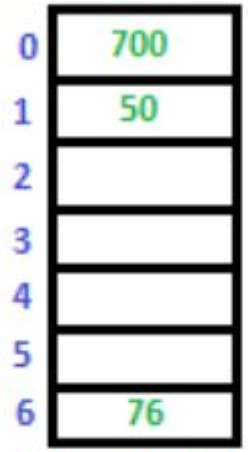


Let us consider a simple hash function as “**key mod 7**” and a sequence of keys as 50, 700, 76, 85, 92, 73, 101

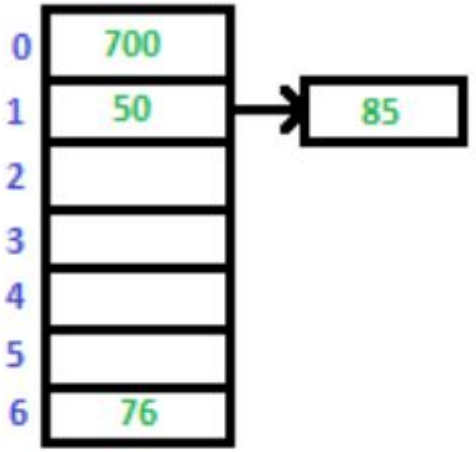


Initial Empty Table

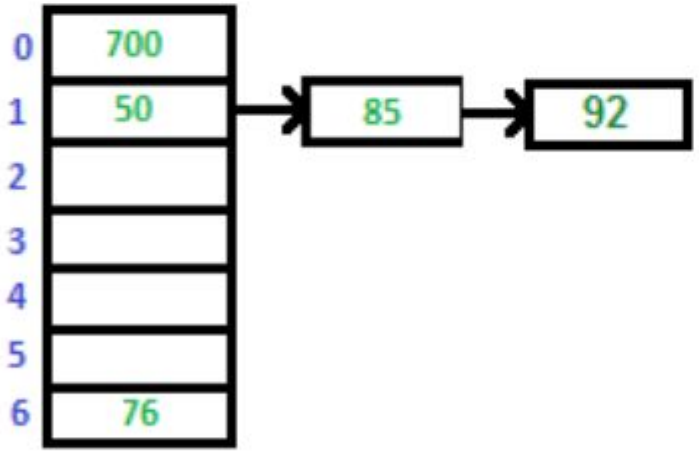
Insert 50



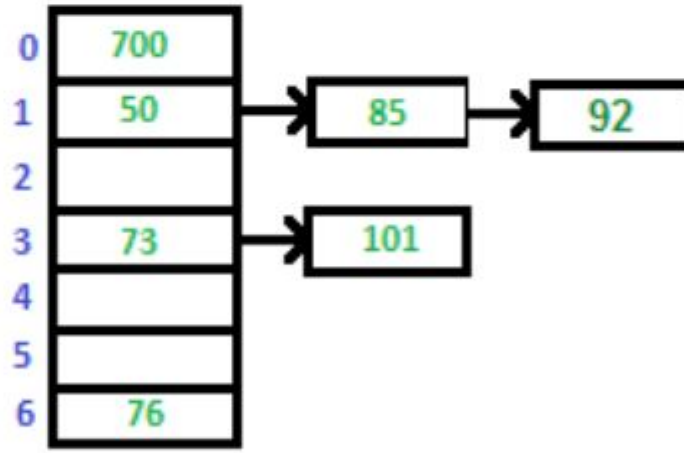
Insert 700 and 76



Insert 85: Collision Occurs, add to chain



Inser 92 Collision Occurs, add to chain



Insert 73 and 101

Time Complexity-

- **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).

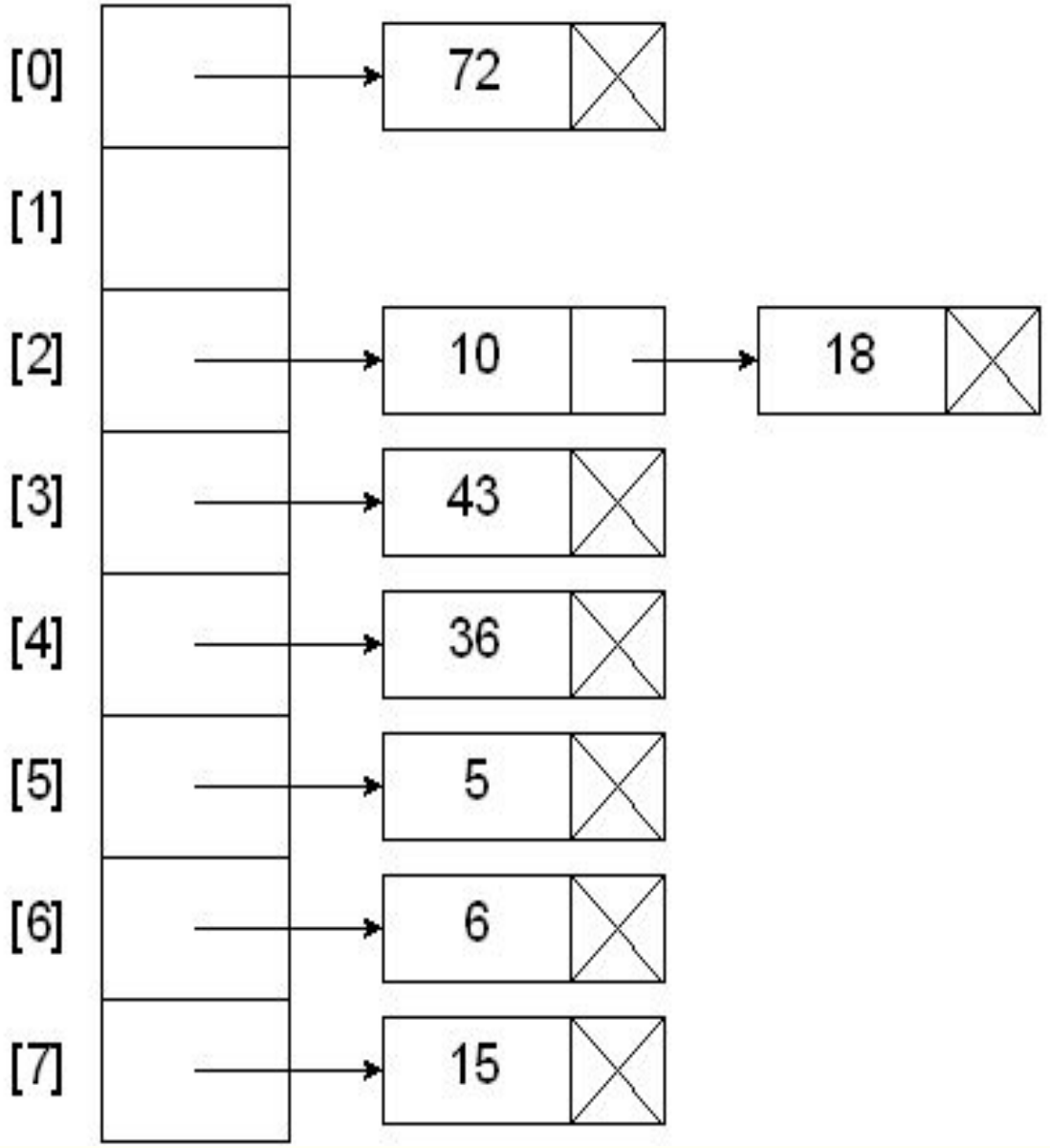
- **For Deletion-**
- In worst case, the key might have to be searched first and then deleted.
- In worst case, time taken for searching is O(n).
- So, time taken for deletion in worst case is O(n).

OPEN HASHING [2nd method]

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
```



- **Load Factor (α)-**

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

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

2. Open Addressing-

- Unlike separate chaining, all the keys are stored inside the hash table.
- No key is stored outside the hash table.
- Techniques used for open addressing are-
  - Linear Probing
  - Quadratic Probing
  - Double Hashing

PROBE

- Each calculation of an address and testfor success is known as

  Probe.

  "*Probing involves finding another location in the hash table for the data element when a collision occurs. The simplest form of probing is linear probing, where the hash table is searched sequentially to find an empty location*."

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.

1. Linear Probing-

- In linear probing, When collision occurs, we linearly probe for the next bucket.
- We keep probing until an empty bucket is 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, using division method in hash function

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.

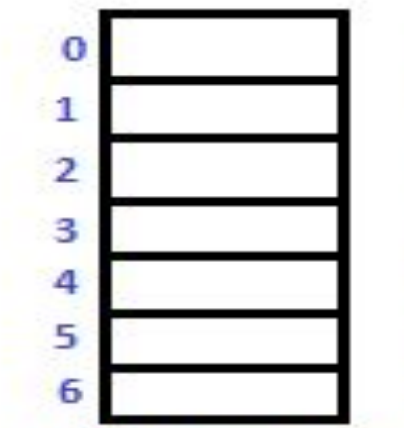
The next key value is 13. The index value associated with this key value is 9 when hash function is applied. The cell is already filled at index 9. When linear probing is applied, the nearest empty cell to the index 9 is 0; therefore, the value 13 will be added at the index 0

CLOSED HASHING

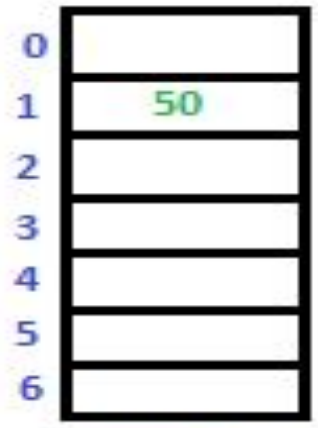
Let us consider a simple hash function as “key mod 7” and a sequence of keys as 50, 700, 76, 85, 92, 73, 101.

CLOSED HASHING

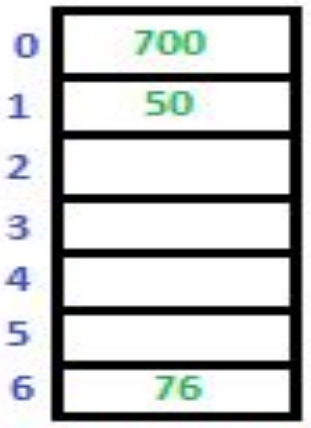
Let us consider a simplehash function as “key mod 7” and a sequence of keys as 50, 700, 76, 85, 92, 73, 101.



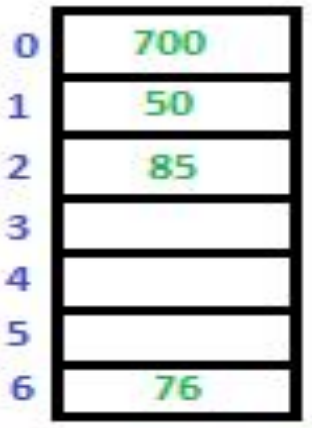
Initial Empty Table



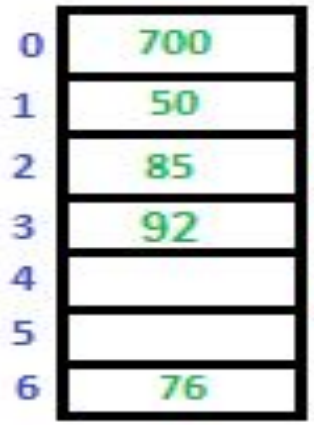
Insert 50



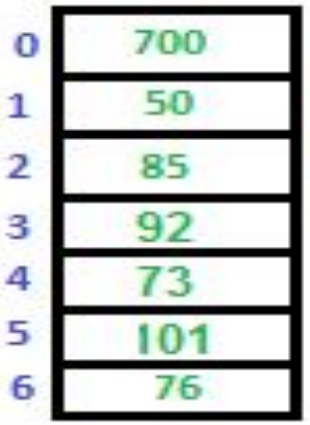
Insert 700 and 76



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

- **Advantage-**
- It is easy to compute.
- **Disadvantage-**
- The main problem with linear probing is clustering.
- Many consecutive elements form groups.
- Then, it takes time to search an element or to find an empty bucket.
- **Time Complexity-**
- Worst time to search an element in linear probing is O (table size)

2. Quadratic Probing-

- In quadratic probing, when collision occurs, we probe for i^2'th bucket in i^{th} iteration.
- We keep probing until an empty bucket is found.
- Rehash(key)=$(n+k^2)$ %table size where k=0,1,2,3,……

R'(K,i)=$(R(k)+ i^2)$mod10

R(k)----□ hash value

i--------□ probe number

- R(k)=k mod 10
- R'(k,i) = $(R(k)+i^2)$mod 10
- Keys : 42,16,91,33,18,27,36

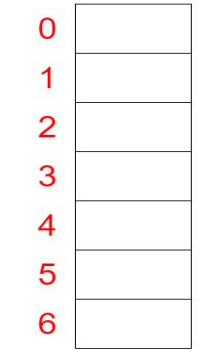
Practise Problem

- Using the hash function ‘key mod 10’, insert the following sequence of keys in the hash table-31,19,2,13,25,24,21,9. Use quadratic probing technique for collision resolution.

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

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

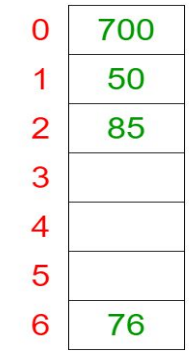
Quadratic Probing Example



Initial Empty Table

Insert 50

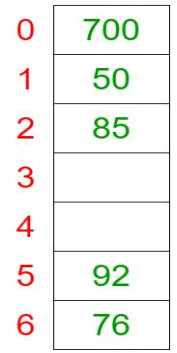
Insert 700 and 76



Insert 85:

Collision occurs.

Insert at 1 + 1*1 position



Insert 92:

Collision occurs at 1.

Collision occurs at 1 + 1*1 position

Insert at 1 + 2*2 position.

Insert 73 and 101

- **Advantage-**
- No extra space
- Primary clustering resolved
- **Disadvantage-**
- Search time O(n)
- Secondary clustering
- No guarantee of finding slot

3. Double Hashing-

- In double hashing, we use another hash function hash2(x) and look for i * hash2(x) bucket in i^{th} iteration.
- It requires more computation time as two hash functions need to be computed.
- Double hashing uses two hash functions, one for accessing the home address of a Key and the other for resolving the conflict.
- H1(key)= key%m (for accessing home address of the key, where m is the size of the table)
- H2(key)= m-(key % m) (for resolving collision, where m is size of the table)

$$(H1(key)+i*H2(key))\bmod m$$

Operations in Open Addressing-

- **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.**

- **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.
- **Delete Operation-**
- The key is first searched and then deleted.
- After deleting the key, that particular bucket is marked as “deleted”.

- **Conclusions-**

- Linear Probing has the best cache performance but suffers from clustering.
- Quadratic probing lies between the two in terms of cache performance and clustering.
- Double caching has poor cache performance but no clustering.

Load Factor (α)-

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

In open addressing, the value of load factor always lie between 0 and 1.

This is because-

- In open addressing, all the keys are stored inside the hash table.
- So, size of the table is always greater or at least equal to the number of keys stored in the table.

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.
- Load Factor, $\alpha = \frac{n}{N}$, where n= Entries and N= Buckets

- Best case scenario: $\alpha<1$, which means N > n, ie , we should always have enough buckets to accommodate all the entries into the hash table
- When $\alpha >= 1$, then we have to increase the number of buckets
- **This increase in the number of buckets is known as Rehashing**
- Basically, when the load factor increases to more than its pre- defined value (default value of load factor is 0.75), the complexity increases.

Rehashing

Steps in Rehashing

1. **Increase the number of buckets from N to N'**
2. **Modify the hash function , ie if hash function initially was x mod N, it will now become x mod N'**

   h(x) = x mod N will become

   h'(x)= x mod N'

3. **Apply the modified hash function, h'(x), to existing elements**

Now we can say that we have an enlarged hash table & α will be < 1

Rehashing

How to choose N'

1. **N' is the closest prime number to 2N**

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.***

Eg. h(x) = x mod 3

key=(6,7,8)

n=3

N=3

Rehashing

Eg. h(x) = x mod 3

key=(6,7,8)

n=3

N=3

α=1, so now we have to rehash

0	6
1	7
2	8

N'= 2N

=2*3

=6 , in this case we have 2 choices of prime number 5 or 7, we will choose the bigger number because we are trying to increase size of the array

So N' =7

h'(x)= x mod 7

Rehashing

h’(x)= x mod 7

keys=(6,7,8)

n=3

N’=7

α=n/N’

=3/7 < 1

0	7
1	8
2	
3	
4	
5	
6	6

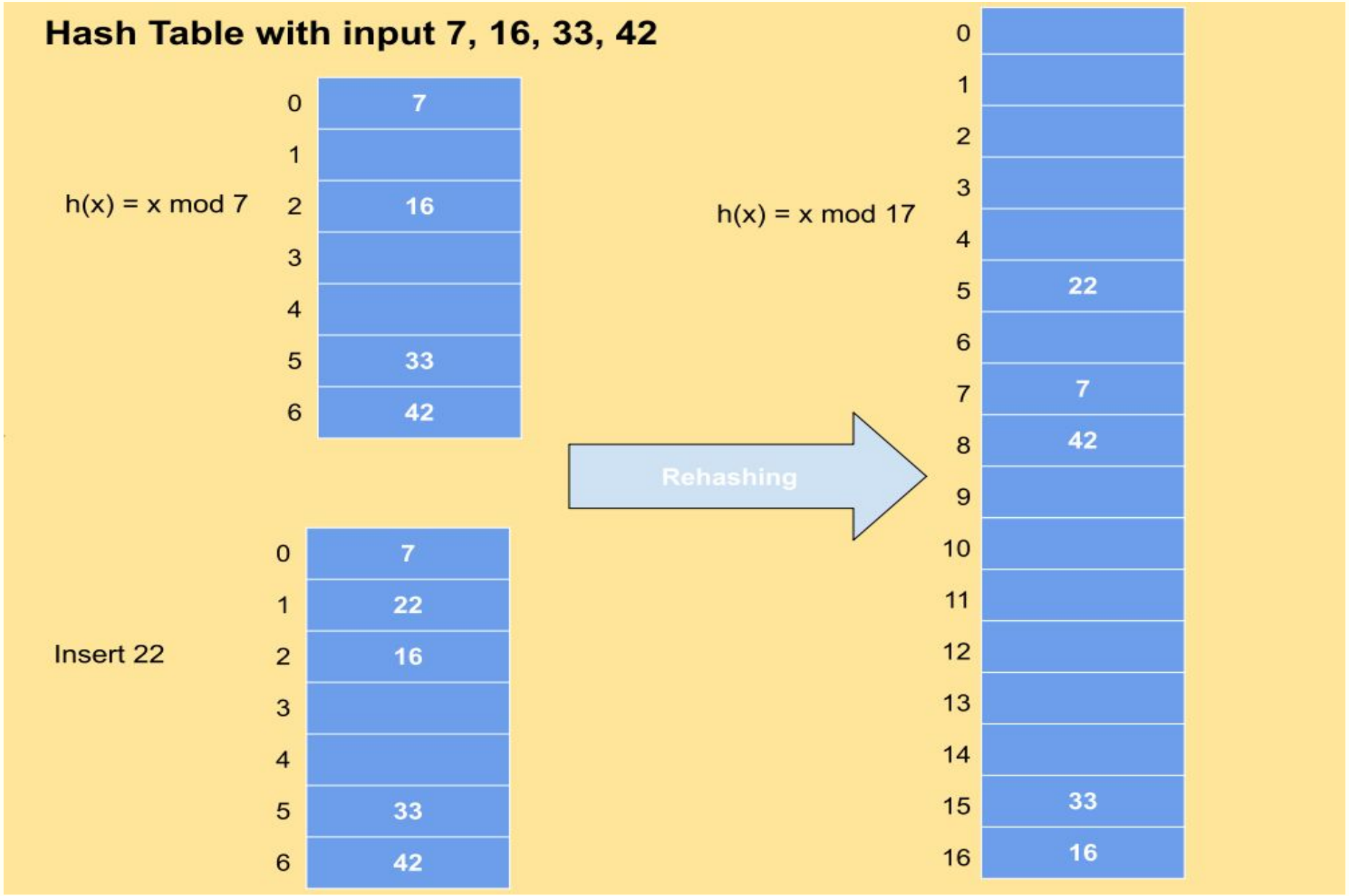
Rehashing

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 if 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

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.

- For example, if table is of size 7 (Table 11.13) and hash function is key % 7 then,

Insert 7,15,13,74,73

0	**7**
1	**15**
2	
3	**73**
4	**74**
5	
6	

Rehashing

- Rehashing is with respect to closed hashing. When we try to store the record with Key1 at bucket
- Hash(Key1) position and find that it already holds a record, it is collision situation
- To handle collision, we use strategy to choose a sequence of alternative locations Hash1(Key1), Hash2(Key1), … within the bucket table so as to place the record with Key1
- This is called as rehashing

Chaining

- This technique used to handle synonym is chaining that chains together all the records that hash to the same address. Instead of relocating synonyms, a linked list of synonyms is created whose head is home address of synonyms
- However, we need to handle pointers to form a chain of synonyms
- The extra memory is needed for storing pointers

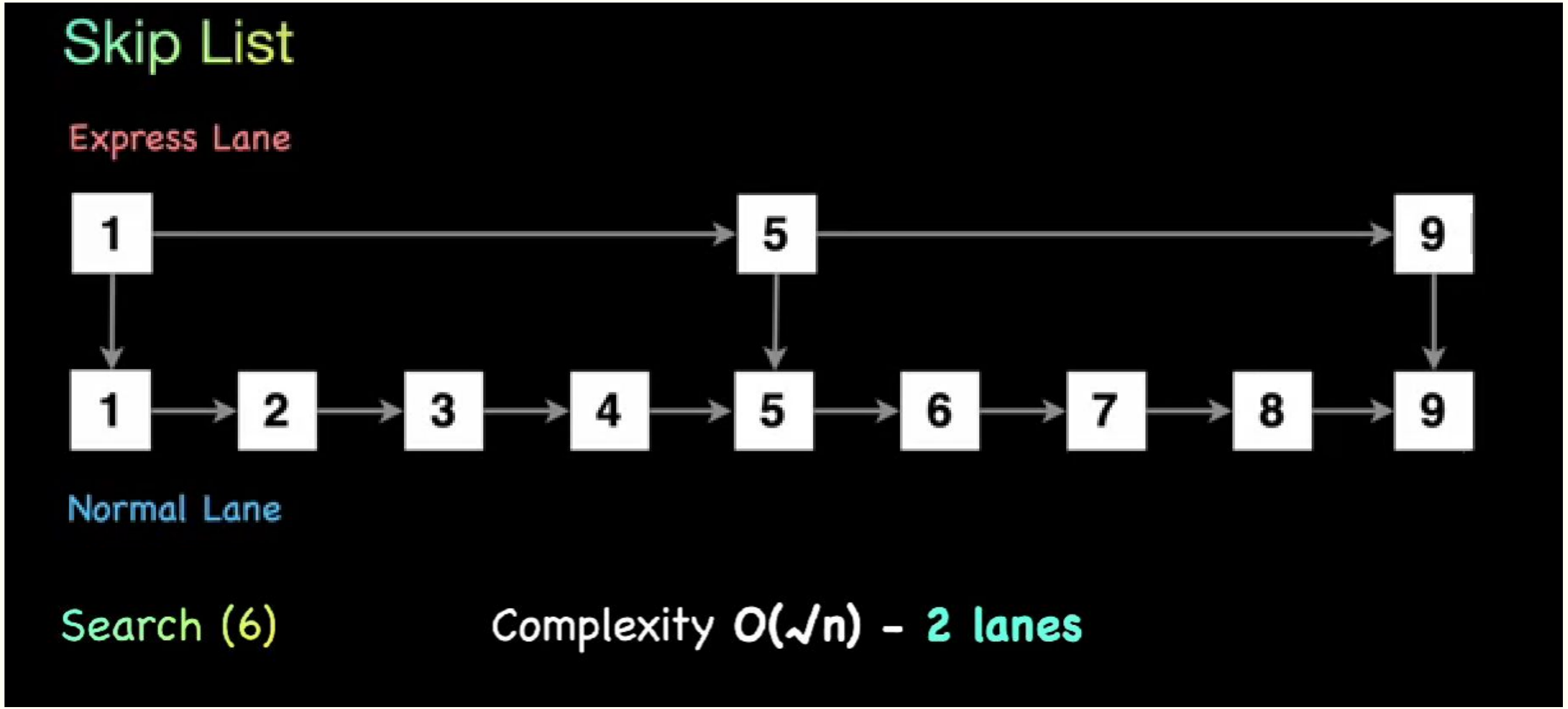
Hash Table Overflow

- An overflow is said to occur when a new identifier is mapped or hashed into a full bucket

- When the bucket size is one, collision and overflow occur simultaneously

Skip List

Skip List


Skip List
Express Lane
1
5
9
1
2
3
4
5
6
7
8
9
Normal Lane
Search (6)
Complexity O(√n) - 2 lanes

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

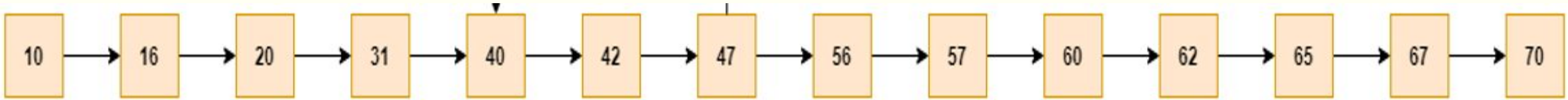
It is built in two layers: The lowest layer and Top layer.
The lowest layer of the skip list is a common sorted linked

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
- 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.

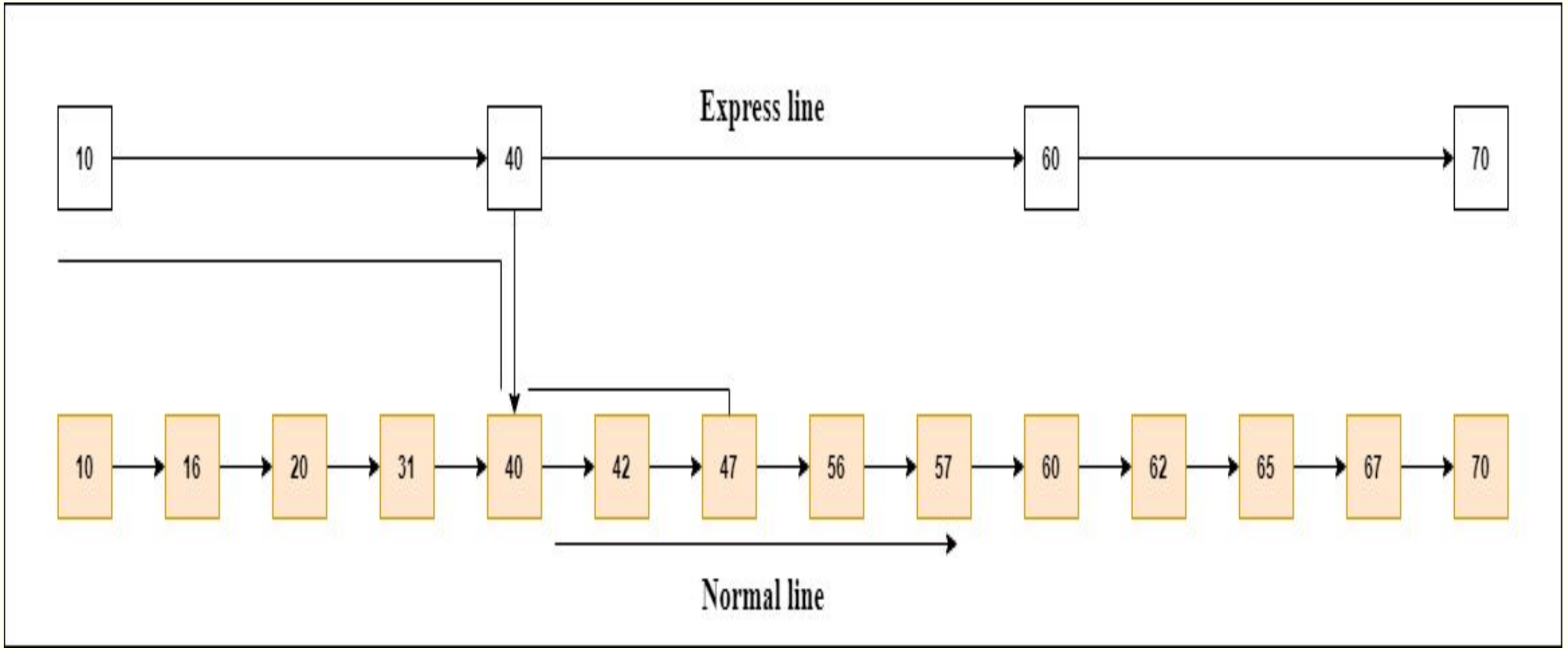


- Suppose you want to find 47 in this example

Skip List

- 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 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.

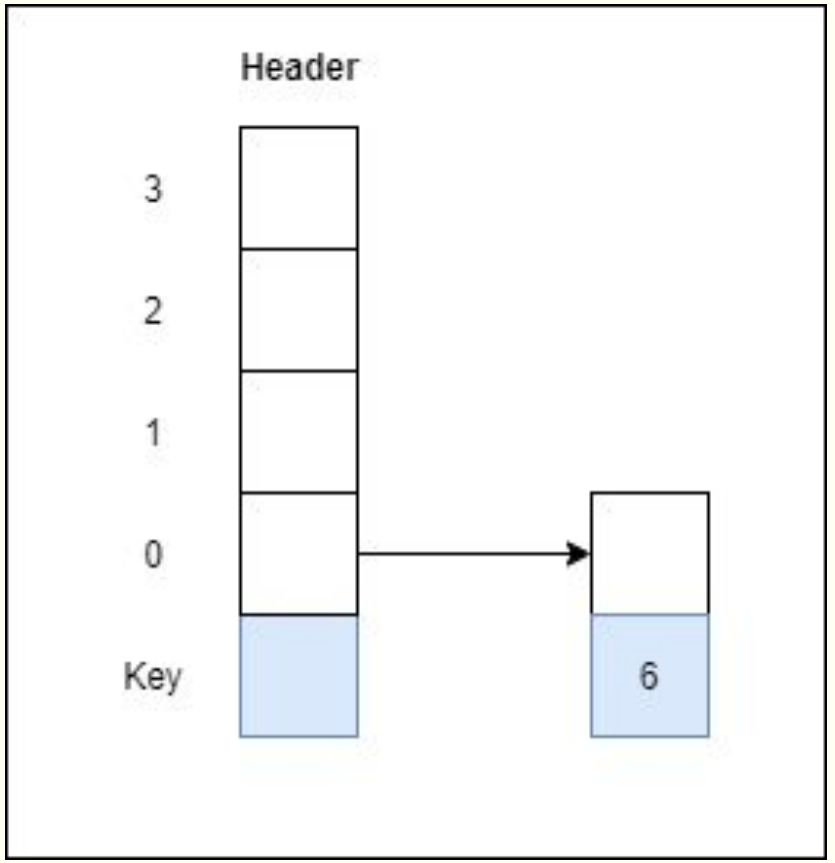
3. Se**arch Operation**: The search operation is used to search a particular node in a skip list.

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

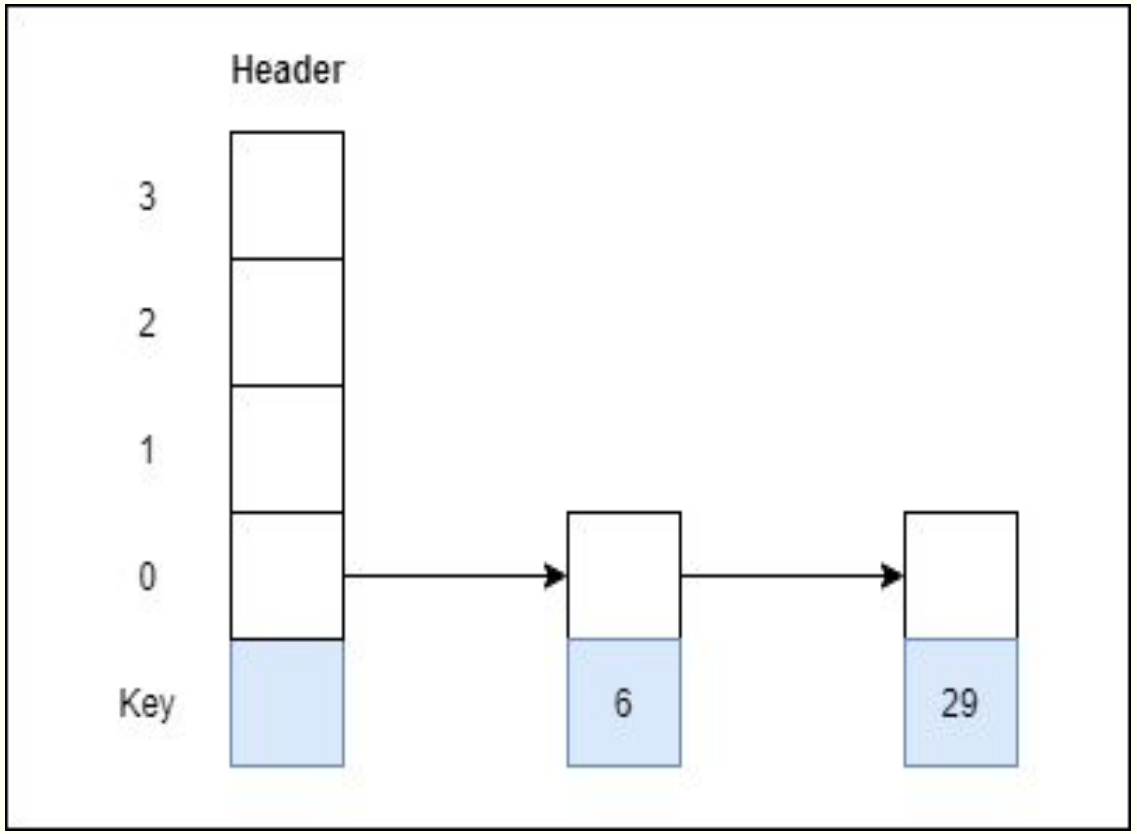


Skip List

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

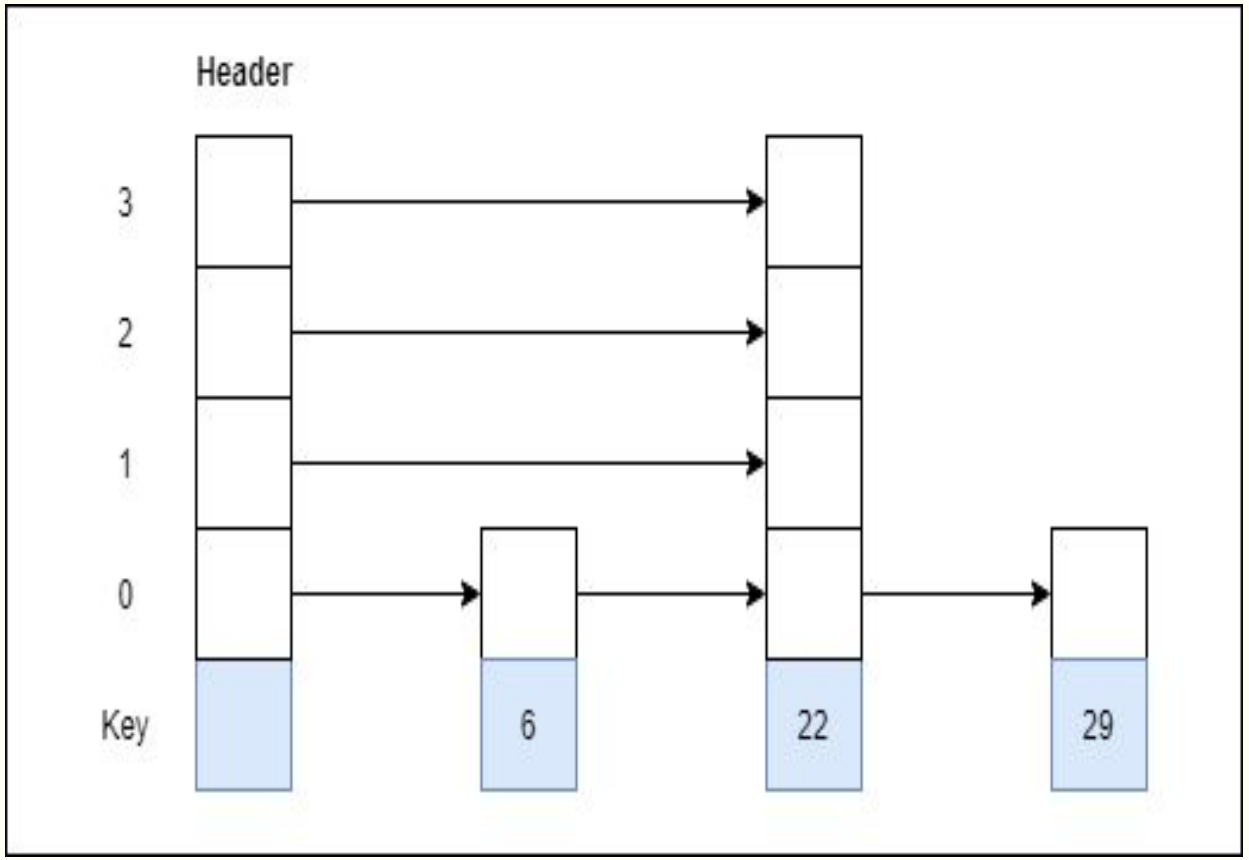


Skip List

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

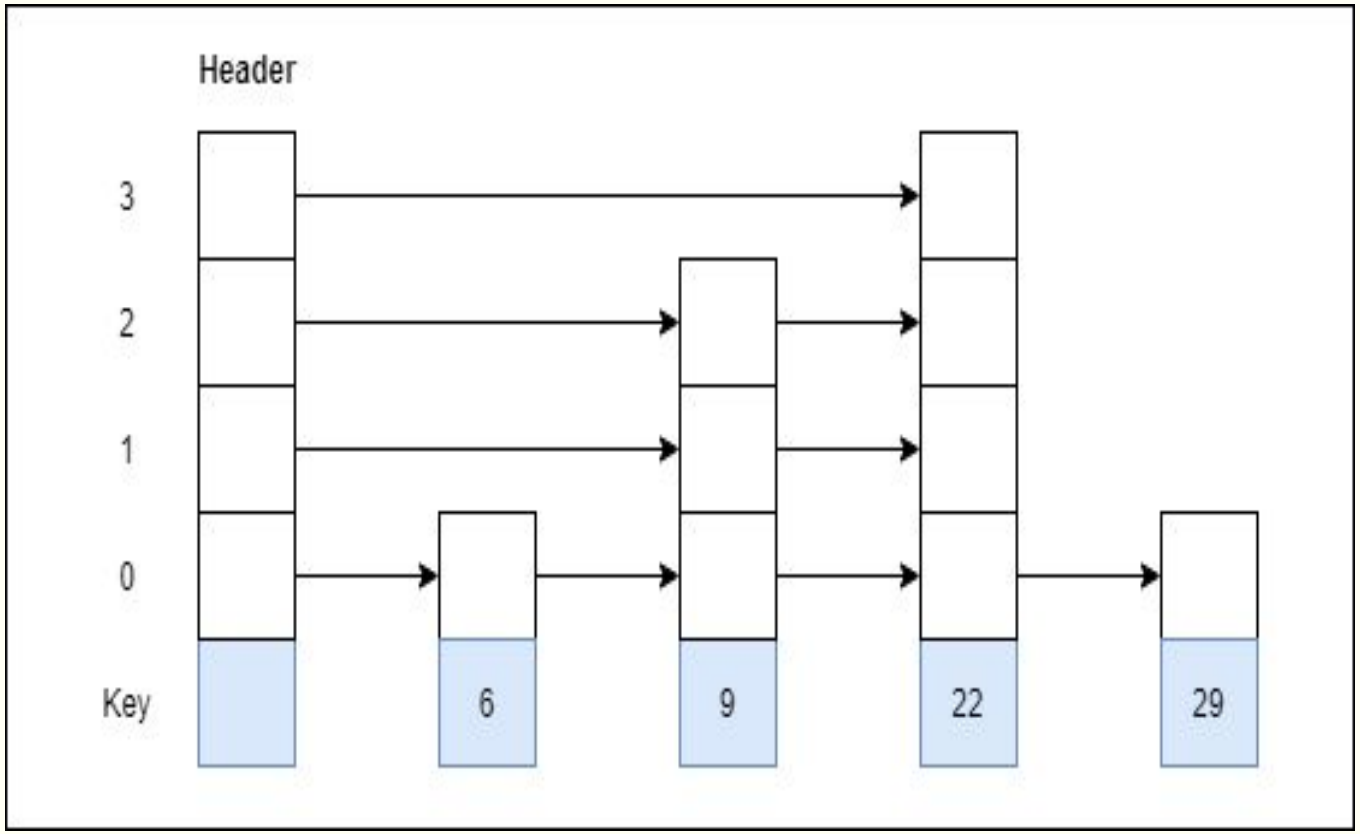


Skip List

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

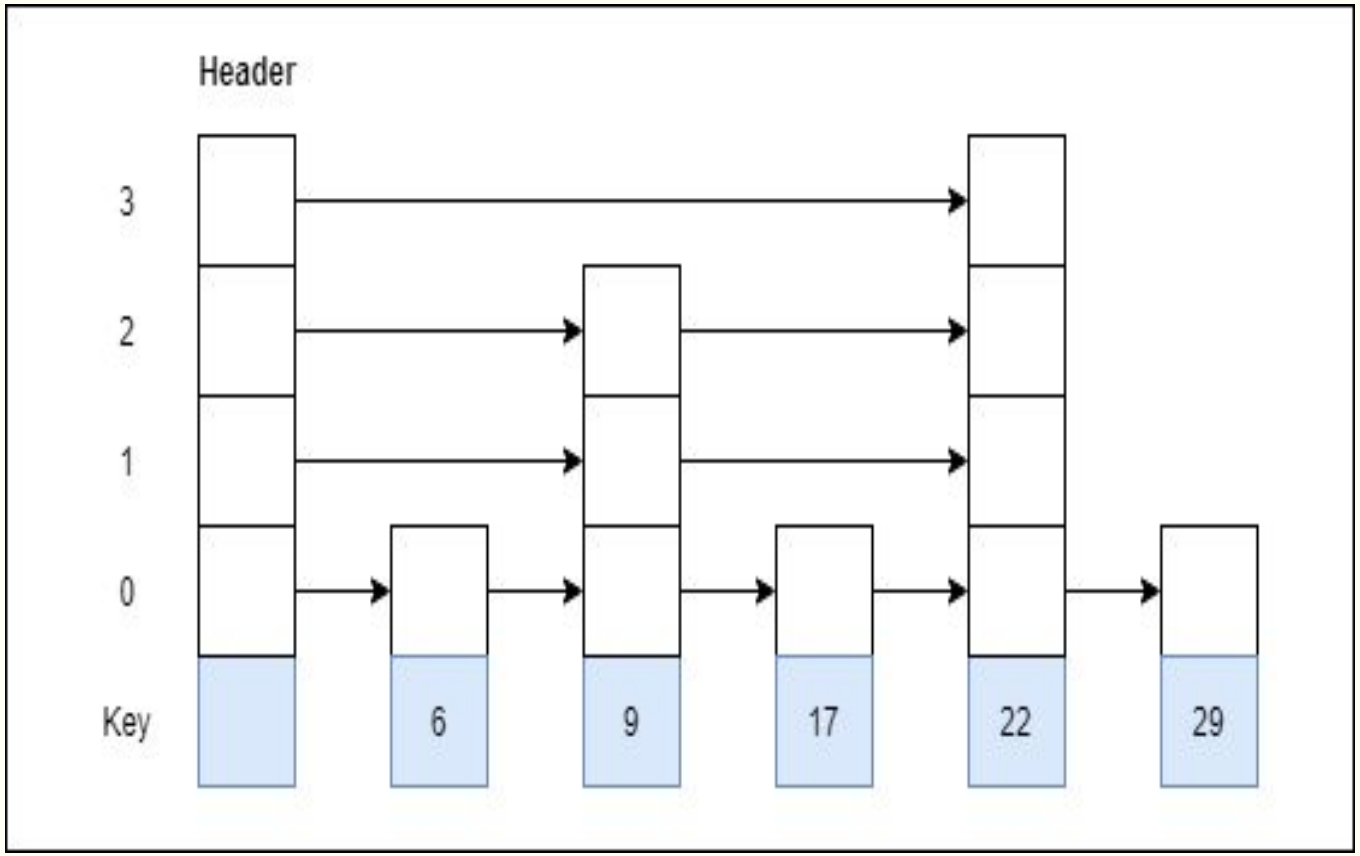


Skip List

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

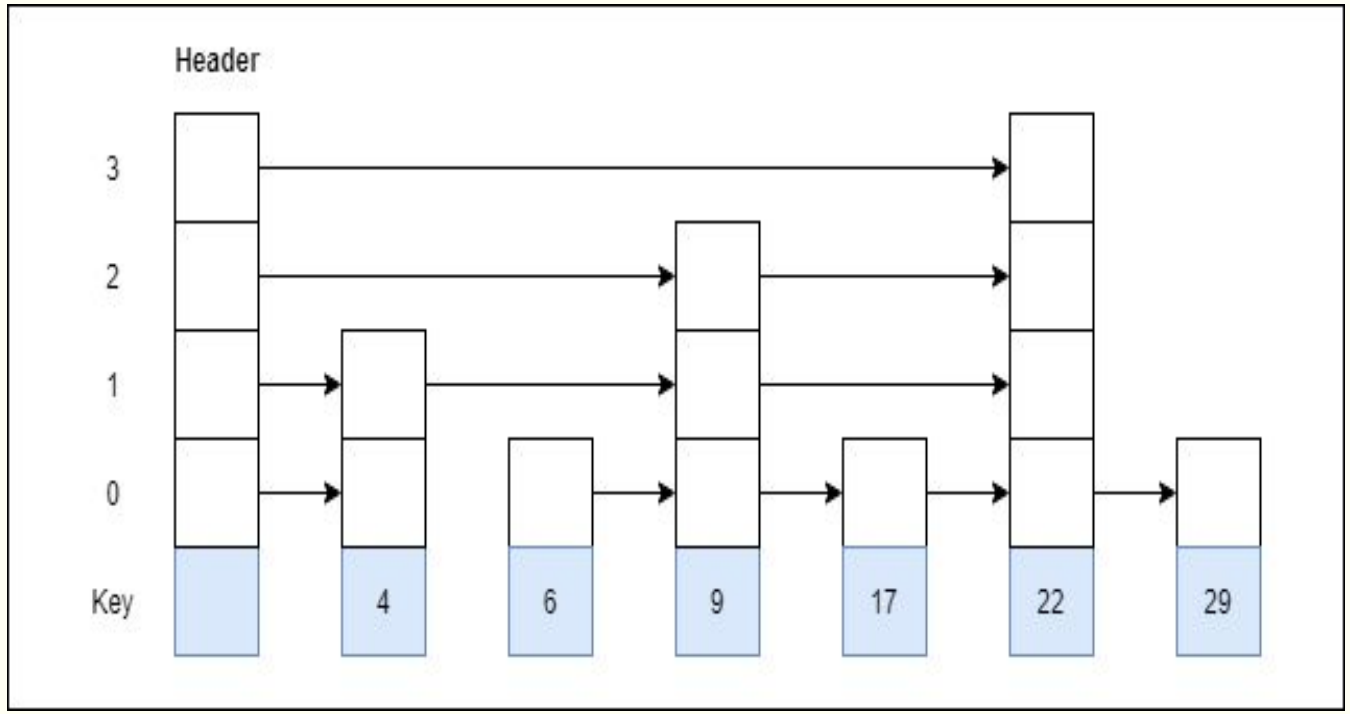


Skip List

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

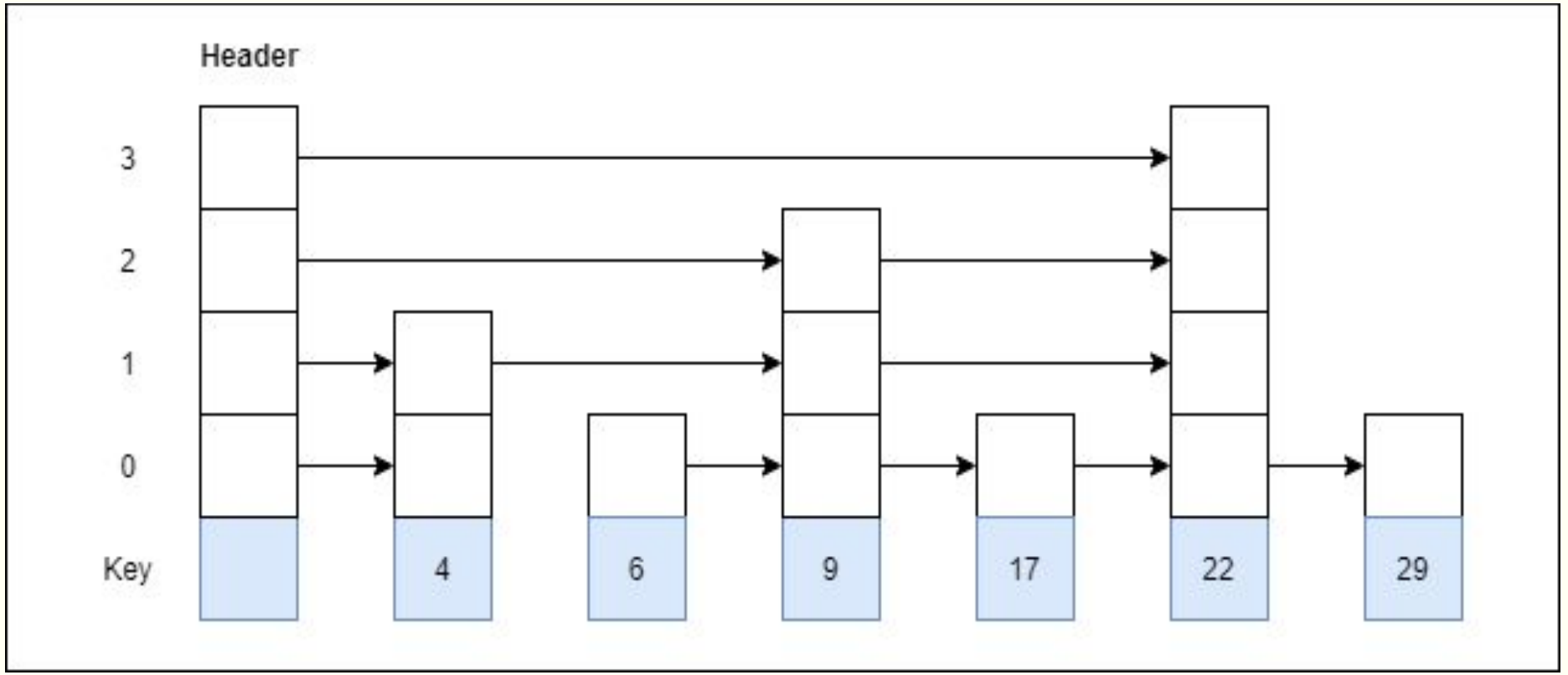
1. 6 with level 1.
2. 29 with level 1.
3. 22 with level 4.
4. 9 with level 3.
5. 17 with level 1.
6. **4 with level 2.**

Step 6: Insert 4 with level 2



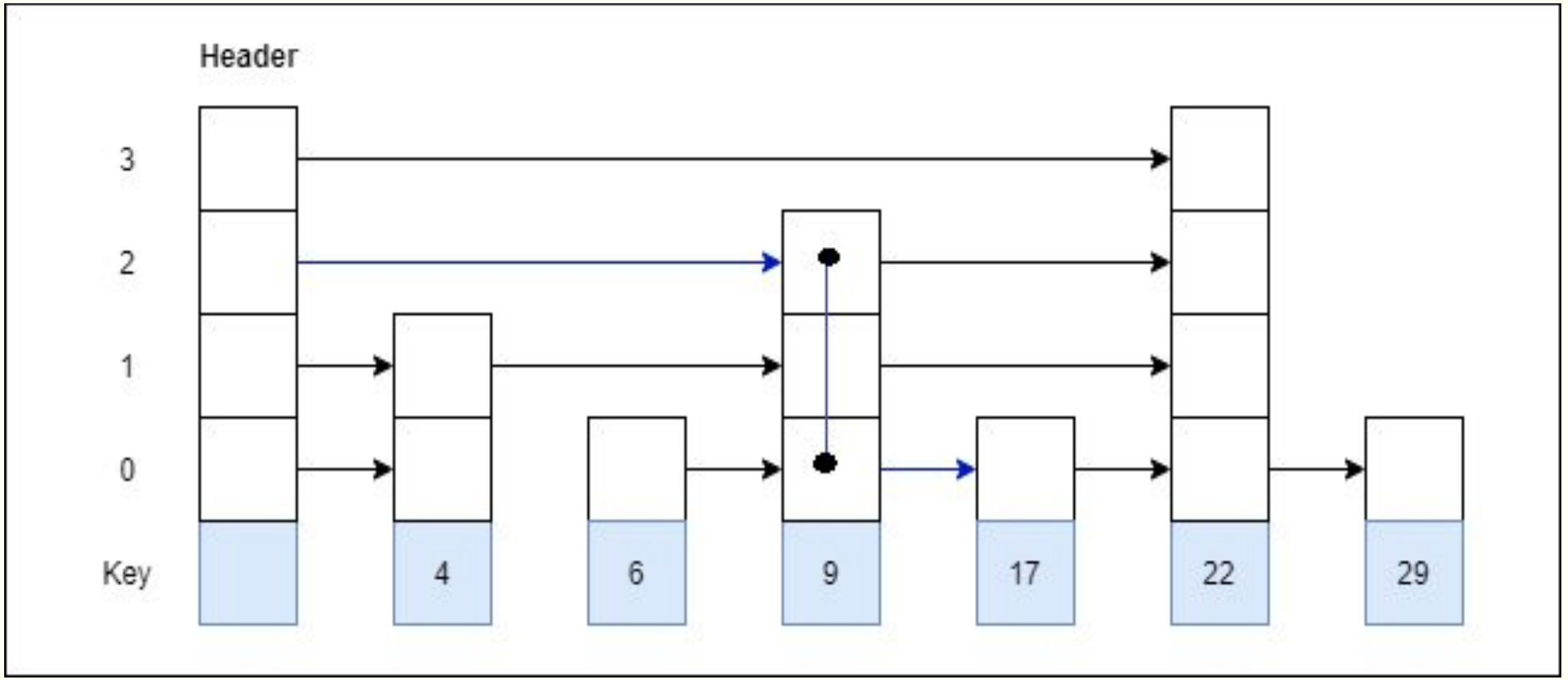
Skip List

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.

https://www.youtube.com/watch?v=UGaOXaXAM5M&list=PLzIiTkKvVk1IDqoaif7SRXFcGGAklJC47&index=4

Please watch this video for in-depth and proper understanding of insertion, search and deletion in skip list. This video alone is sufficient for this topic

THANK YOU !!

Indu Seethanathan

inseethanathan@kkwagh.edu.in

K.K.Wagh Institute of Engineering Education & Research, Nashik