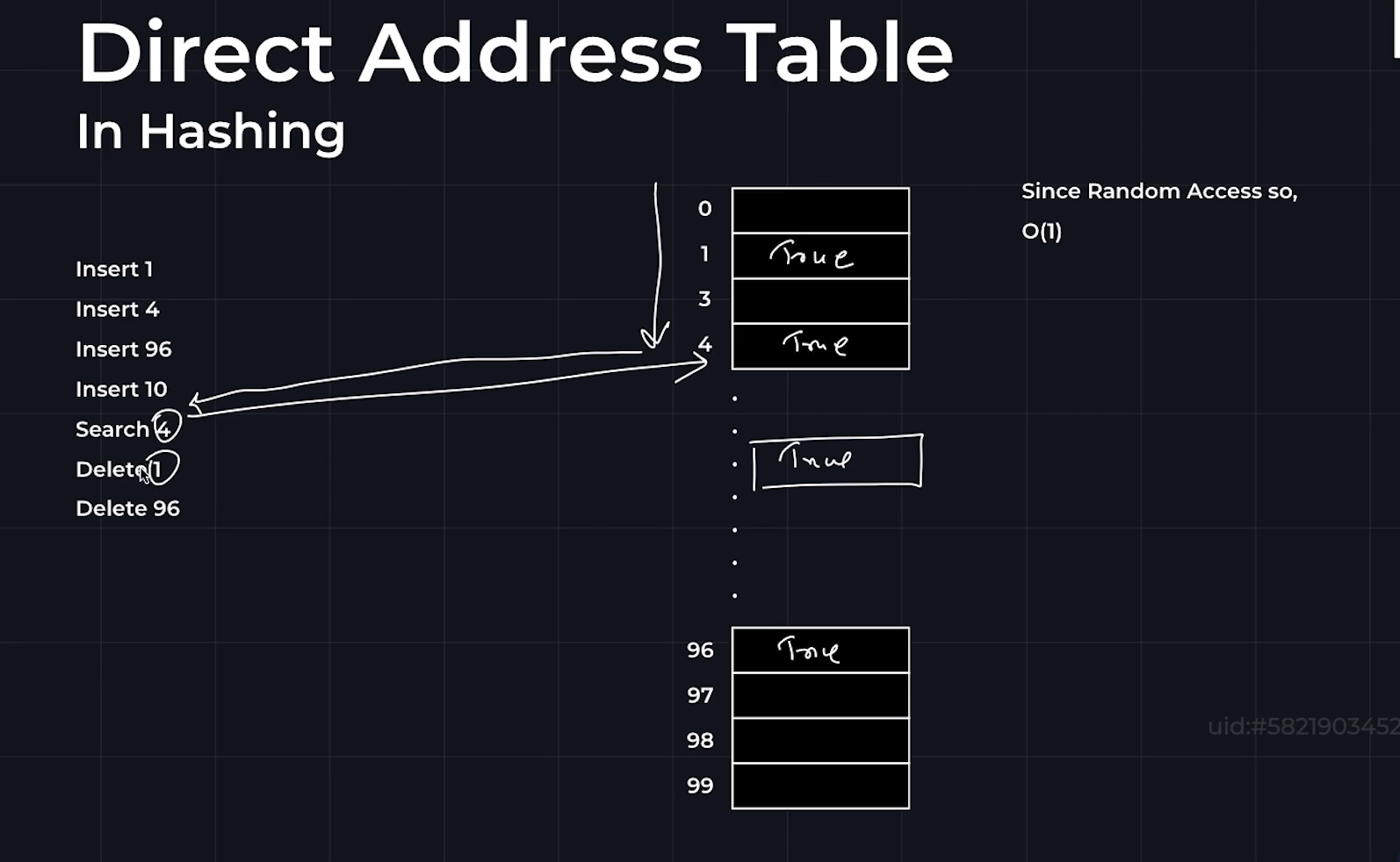
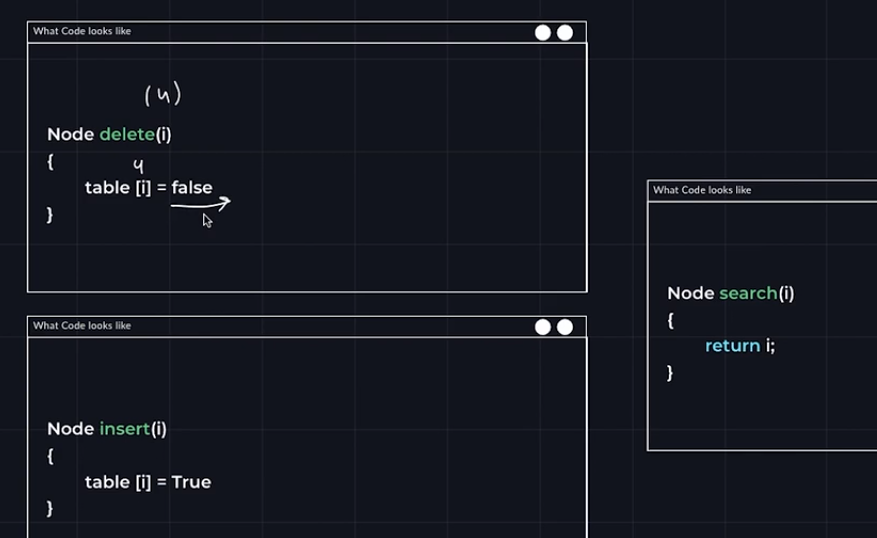
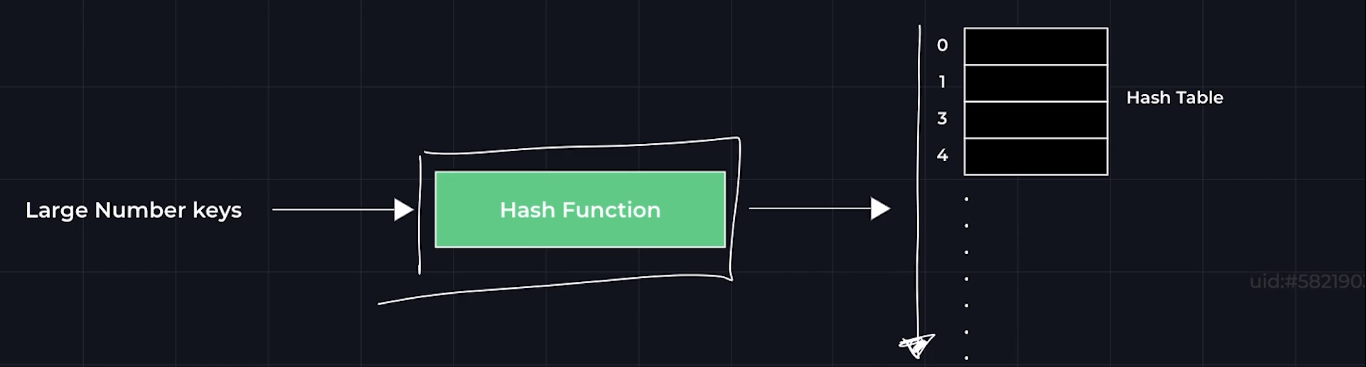
**HASHING CHEAT SHEET**

****

* The block Is created previously, and all values is set to be false by default.
* If an element is inserted, then the values is changed from false to true.
* If an element is deleted, then the values is changed from true to false.
* The values from the address table can be accessed directly without any traversal





Larger keys can be converted /stored in smaller hash table with the help of hash function.

**Rules for hash function:**

* It should point to same index in hash table for the same values always.
* Hash function should generate values from [0,n-1]
* Hash function should be unbiased and should uniformly distribute keys in the hash table.
* Should be fast.

**Time complexity: -**

* Integer: - O (1)
* String: - O (n)

General form of hashing: **hash(key)=key%N**

* N is taken as prime number.
* Do not take power of ten as N because it may lead to collision

**Collision: - Two keys after hash function point to the same index.**

E.g., of collision: - **9901%10=990** same as **9902%10=990**

**Size of hash table depends on number of keys and not range of keys**

**UNIVERSAL HASHING: -**

* You have a large number of built-in hash function.
* Whenever we require hashing a random hash function is picked up.
* Again, if we need a hash a function a random hash function is picked up again.

**COLLISION HANDLING:**

Collision cannot be avoided during hashing, but it can be managed in many ways.

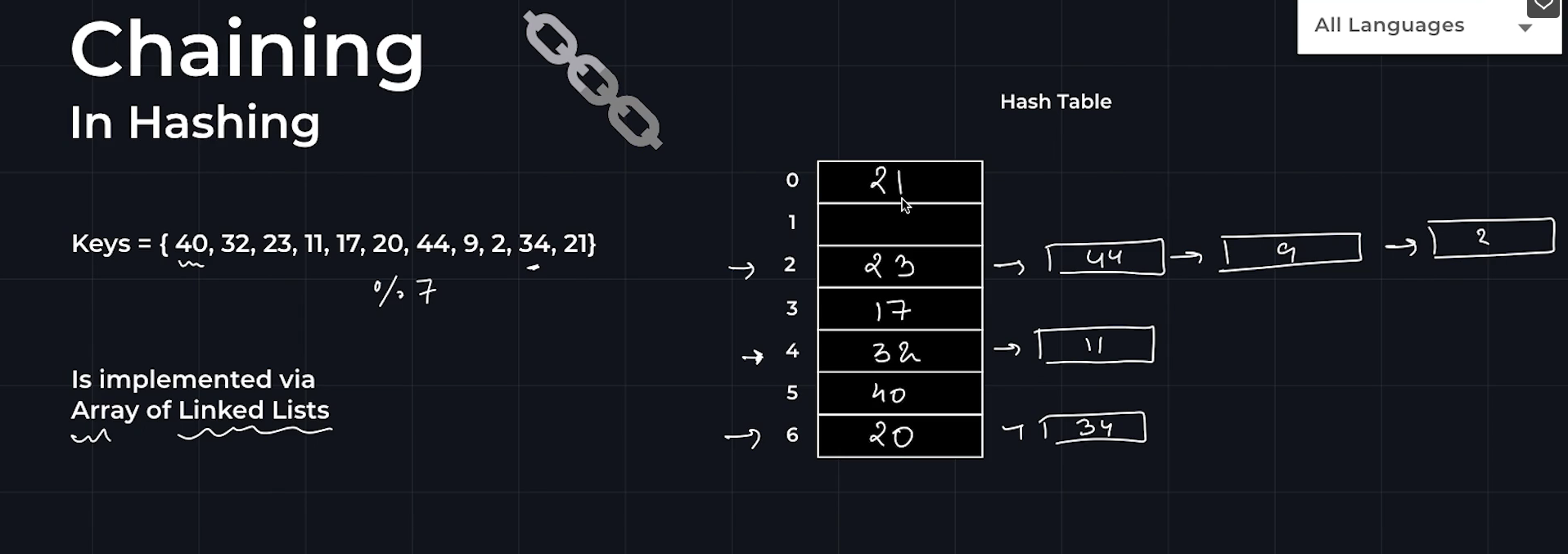
Methods of handling:

1 Chaining

2) Open Addressing

* + Linear Probing
  + Quadratic Probing
  + Double Probing

**CHAINING**



The values at memory (**there may be a high chance that two or more values have the same key no matter how efficient the hash function is**) is chained.

**PERFORMANCE: -**

It is measured based on load factor.

**LOAD FACTOR=N/M : N-> No of keys to be inserted M->No of slots in hash table**

**Load factor is small:** large no of collision

**Load factor is large:** Less probability of collision, but higher unused slots and extra memory.

**Insertion, deletion, search= O(1+Load factor)**

**Text

Description automatically generated**

**Linear probing:-**

A picture containing diagram

Description automatically generated

We assume that the no of keys is equal to the number of memory at least India's map

If the memory block of index is full or occupied then the data is placed in the next empty space

The next empty spaces is searched downwards. If we reach the end of the memory then we go to the starting of the memory that is the memory is listed as a circular list

**searching**

we go to the memory block after dashing function.

If the value is not equal to the required value then we go down line.

similar to insertion if we rea reach the end of a block we go to first of the block.

If you find the value return the value.

If we get empty space return false.

If we come to starting index again return false.

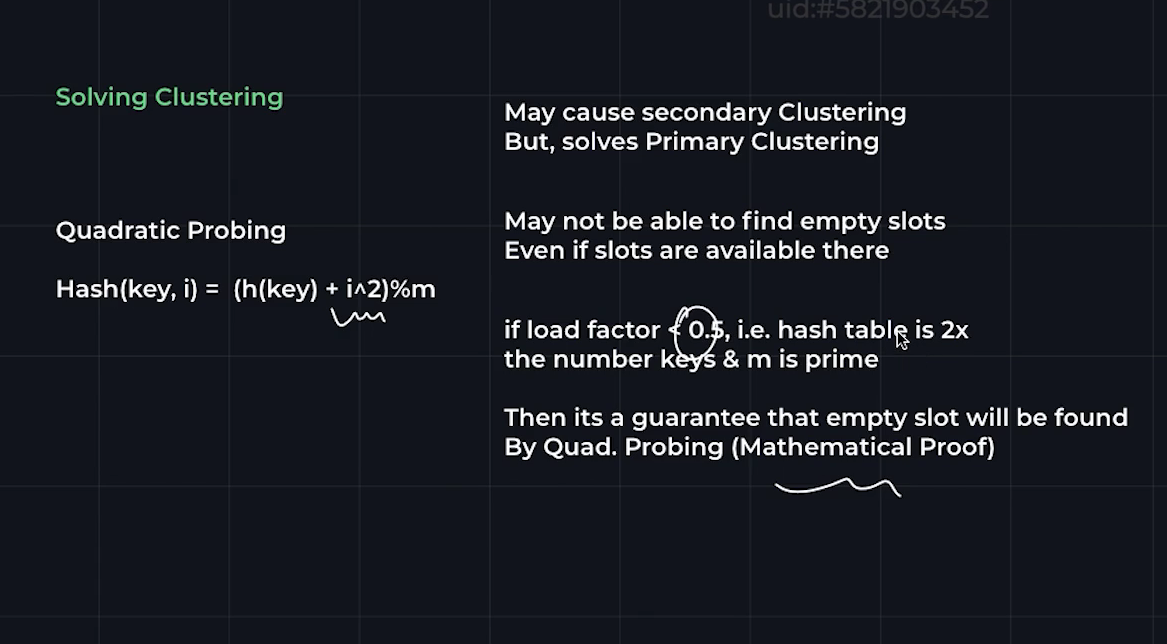
**Deletion.**:

While deleting. We don't set the value of the memory block to false but instead set it to. Deleted. This is done because while searching we don’t need to confuse between deleted node and empty node.

**Disadvantages.**

Linear probing causes clustering of data.

**Quadratic probing:**

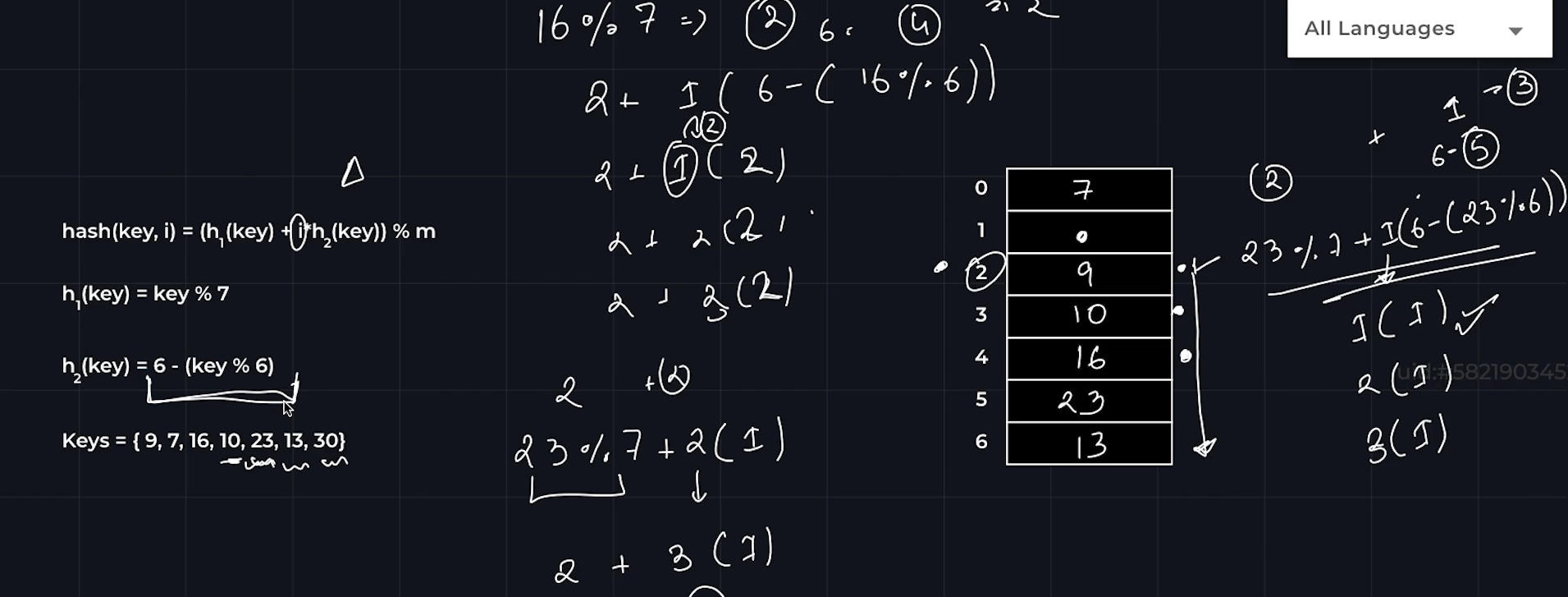
****

In the above image

I -> no of time collision happens for a particular memory block.

M -> Prime number.

**Double Probing:**

****

**Chaining vs open addressing:**

