**Hash Table**

The Hash table data structure stores elements in key-value pairs where

* **Key**- unique integer that is used for indexing the values
* **Value** - data that are associated with keys.

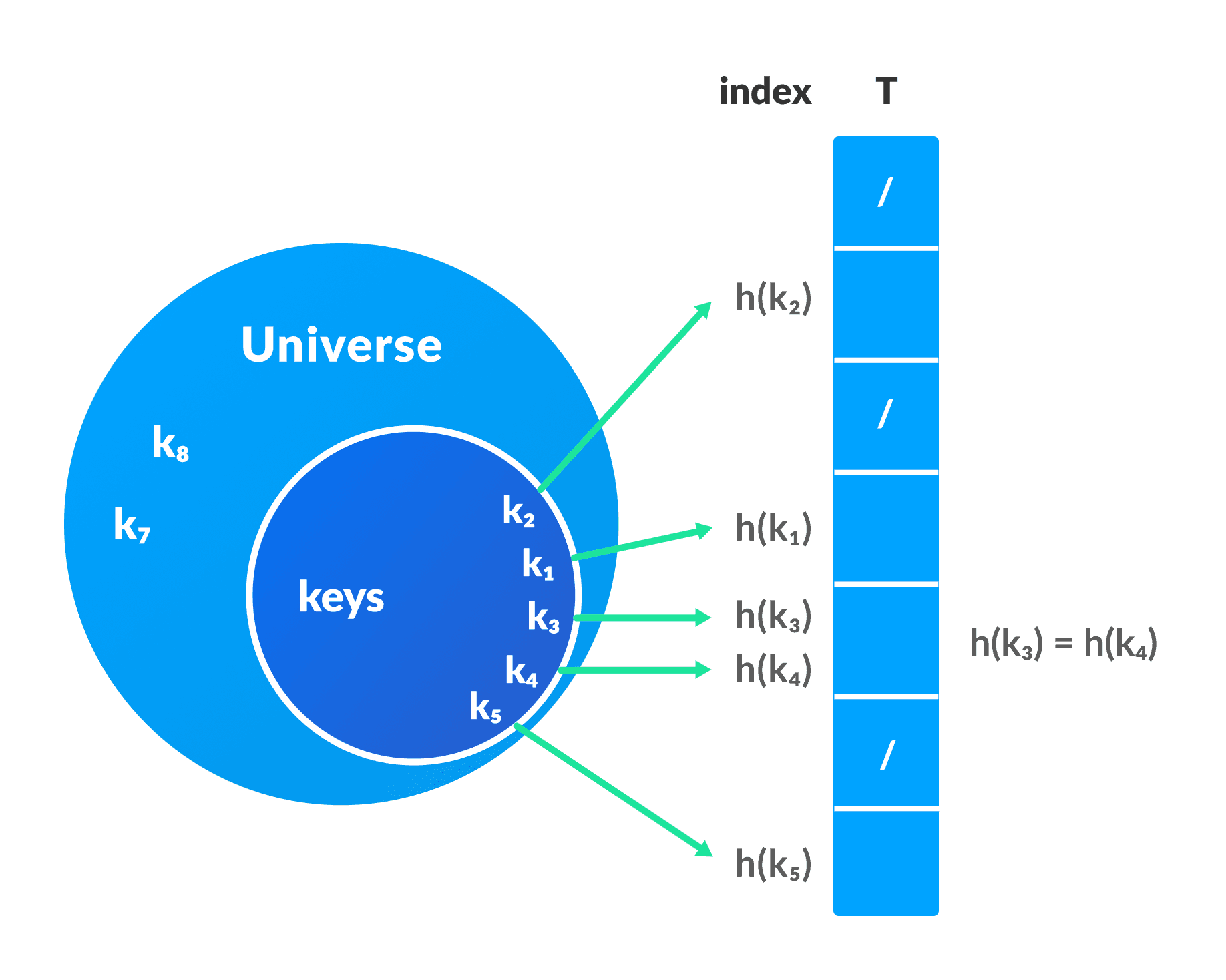


## Hashing (Hash Function)

In a hash table, a new index is processed using the keys. And, the element corresponding to that key is stored in the index. This process is called **hashing**.

Let k be a key and h(x) be a hash function.

Here, h(k) will give us a new index to store the element linked with k.

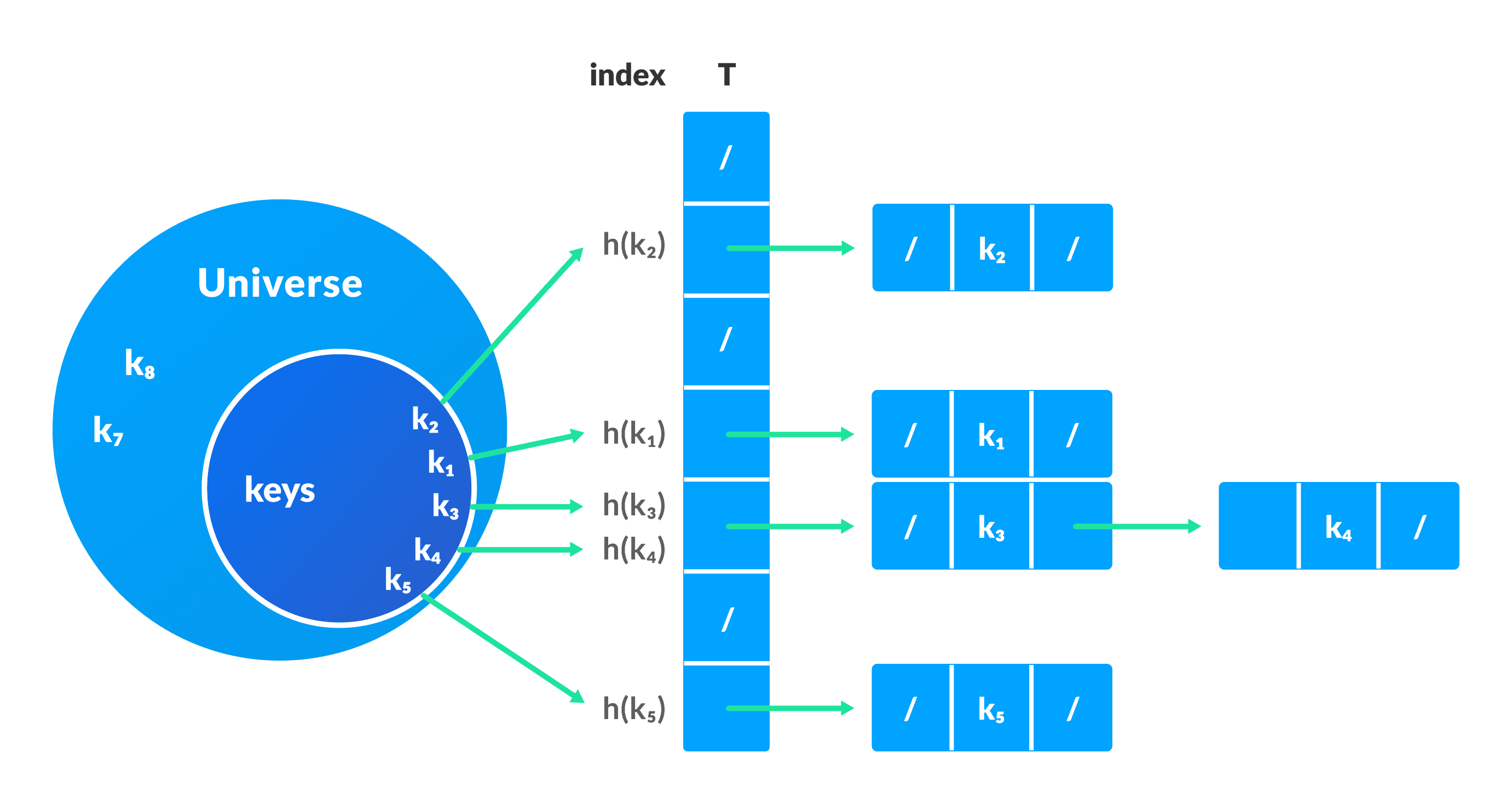


## Hash Collision

When the hash function generates the same index for multiple keys, there will be a conflict (what value to be stored in that index). This is called a **hash collision.**

We can resolve the hash collision using one of the following techniques.

* Collision resolution by chaining
* Open Addressing: Linear/Quadratic Probing and Double Hashing



## 2. Open Addressing

Unlike chaining, open addressing doesn't store multiple elements into the same slot. Here, each slot is either filled with a single key or left NIL.

Different techniques used in open addressing are:

### i. Linear Probing

In linear probing, collision is resolved by checking the next slot.

h(k, i) = (h′(k) + i) mod m

where

* i = {0, 1, ….}
* h'(k) is a new hash function

If a collision occurs at h(k, 0), then h(k, 1) is checked. In this way, the value of i is incremented linearly.

The problem with linear probing is that a cluster of adjacent slots is filled. When inserting a new element, the entire cluster must be traversed. This adds to the time required to perform operations on the hash table.

### ii. Quadratic Probing

It works similar to linear probing but the spacing between the slots is increased (greater than one) by using the following relation.

h(k, i) = (h′(k) + c1i + c2i2) mod m

where,

* c1 and c2 are positive auxiliary constants,
* i = {0, 1, ….}

### iii. Double hashing

If a collision occurs after applying a hash function h(k), then another hash function is calculated for finding the next slot.

h(k, i) = (h1(k) + ih2(k)) mod m

## Good Hash Functions

A good hash function may not prevent the collisions completely however it can reduce the number of collisions.

Here, we will look into different methods to find a good hash function