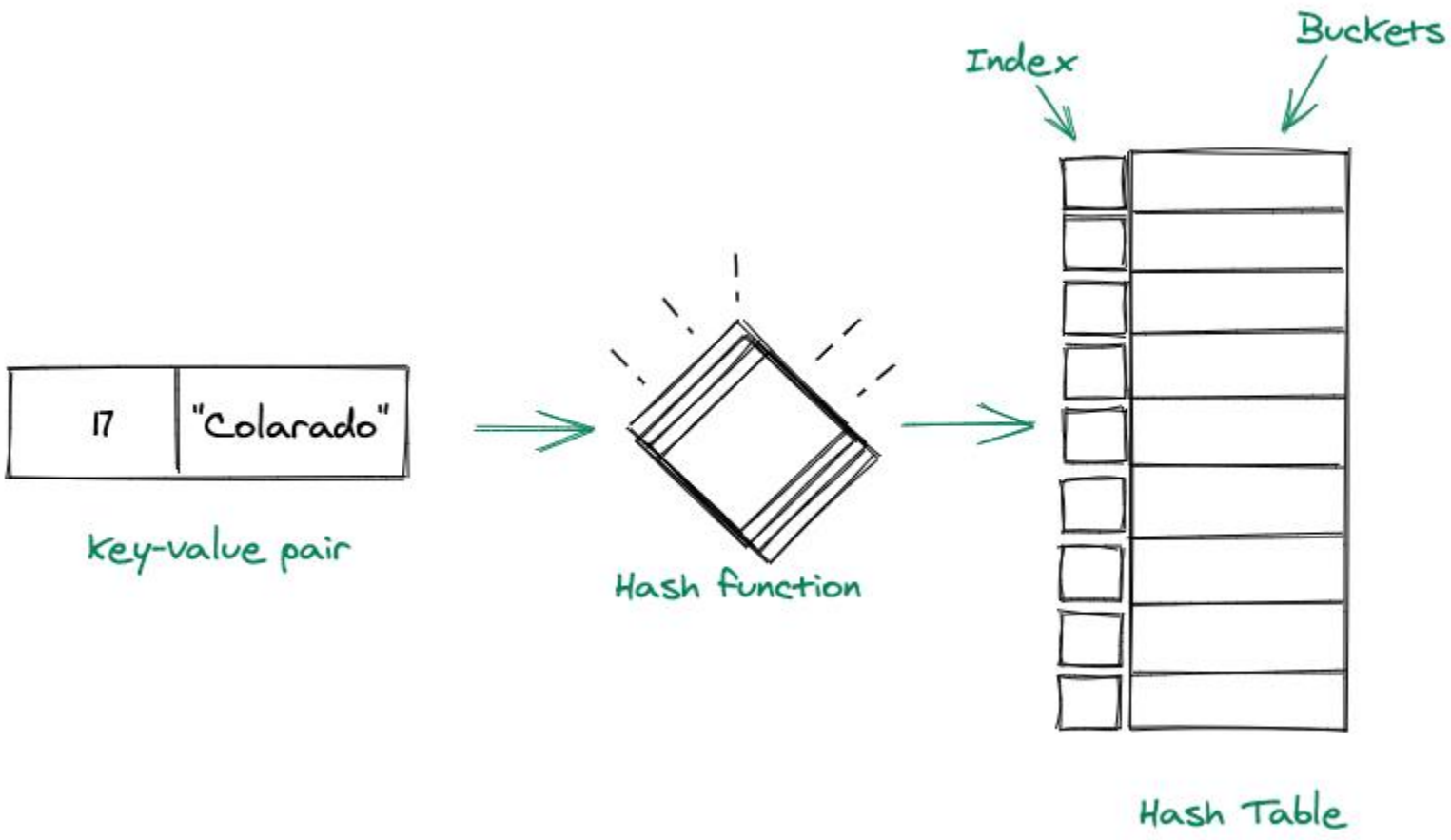
Hash Table Data Structure

A Hash table is a data structure that is used to store the data in key-value pairs.

- Each key in the hash table is mapped to a value that is present in the hash table.
- The key present in the hash table are used to index the values, as we take this key and pass it to a hash function which in turn performs some arithmetic operation on it. The resulting value we get after the operation is the index of the hash table where we store the key-value pairs in the table.
- Internally a hash table contains buckets, and the location of which bucket to use for a particular key is determined by the key's hash function.

Consider the visual representation below:



Components of a Hash Table

A hash table comprises two components in total, these are:

Hash function:

- A hash function is used to determine the index of the key-value pair.
- It is always recommended that we should choose a good hash function for creating a good hash table.
- Besides a good hash function, it should be a one-way function, i.e. we should be able to get the hash value from the key and not vice versa.
- Also, it should avoid producing the same hash for different keys.

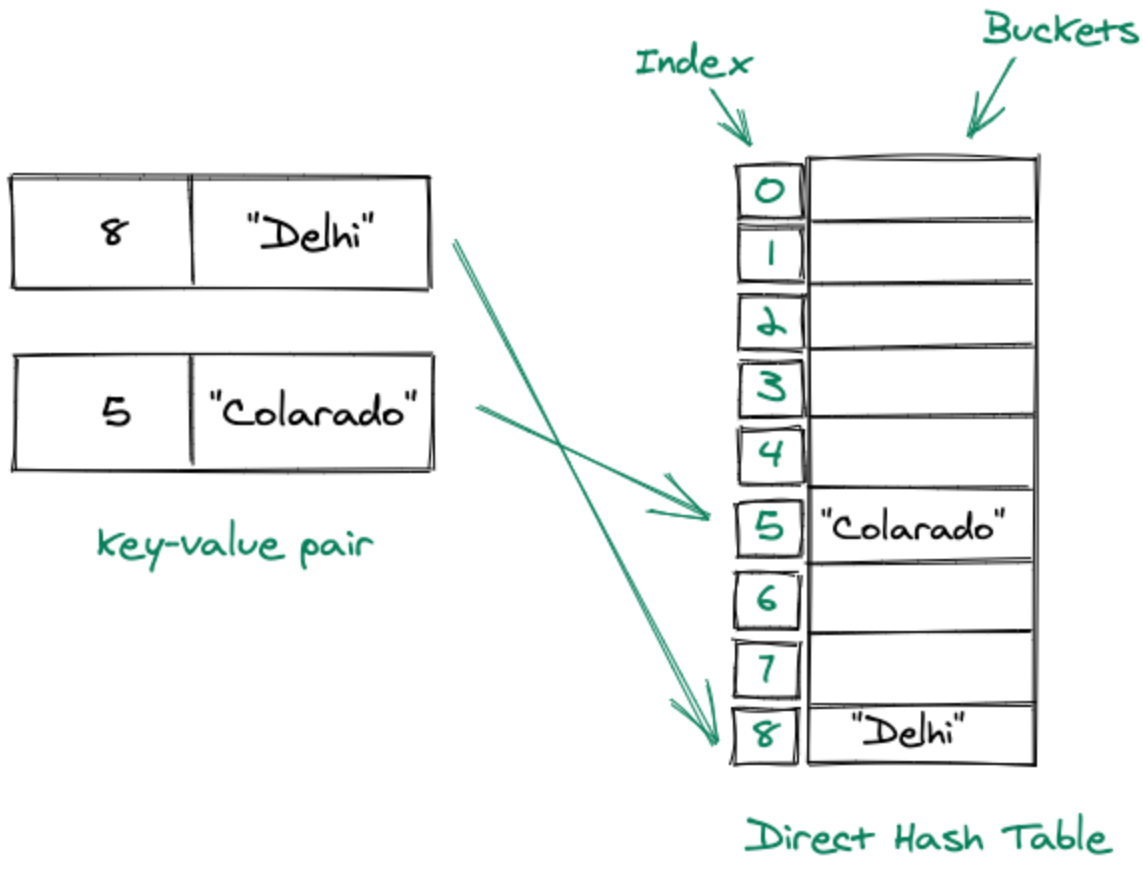
Arrays:

- The array(buckets) are used to hold the key-value pairs.
- The size of the array should be set accordingly to the number of key-value pairs we will have.

Direct Addressing

It is a technique in which we make use of direct address tables to map the values with their keys. It uses the keys as indexes of the buckets and then store the values at those bucket locations. Though direct addressing does facilitate fast searching, fast inserting and fast deletion operations, but at a cost.

Consider the pictorial representation below:



Advantages of Direct Address Table:

- **Insertion in O(1) time:** Inserting an element in a direct address table is the same as inserting an element in an array, hence we can do that in **O(1) time** as we already know the index(via key).
- **Deletion in O(1) time:** Deleting an element from a direct address table is the same as deleting from an array, hence the **O(1) time**.
- **Searching in O(1) time:** Searching an element takes linear time**(O(1))** as we can easily access an element in an array in linear time if we already know the index of that element.

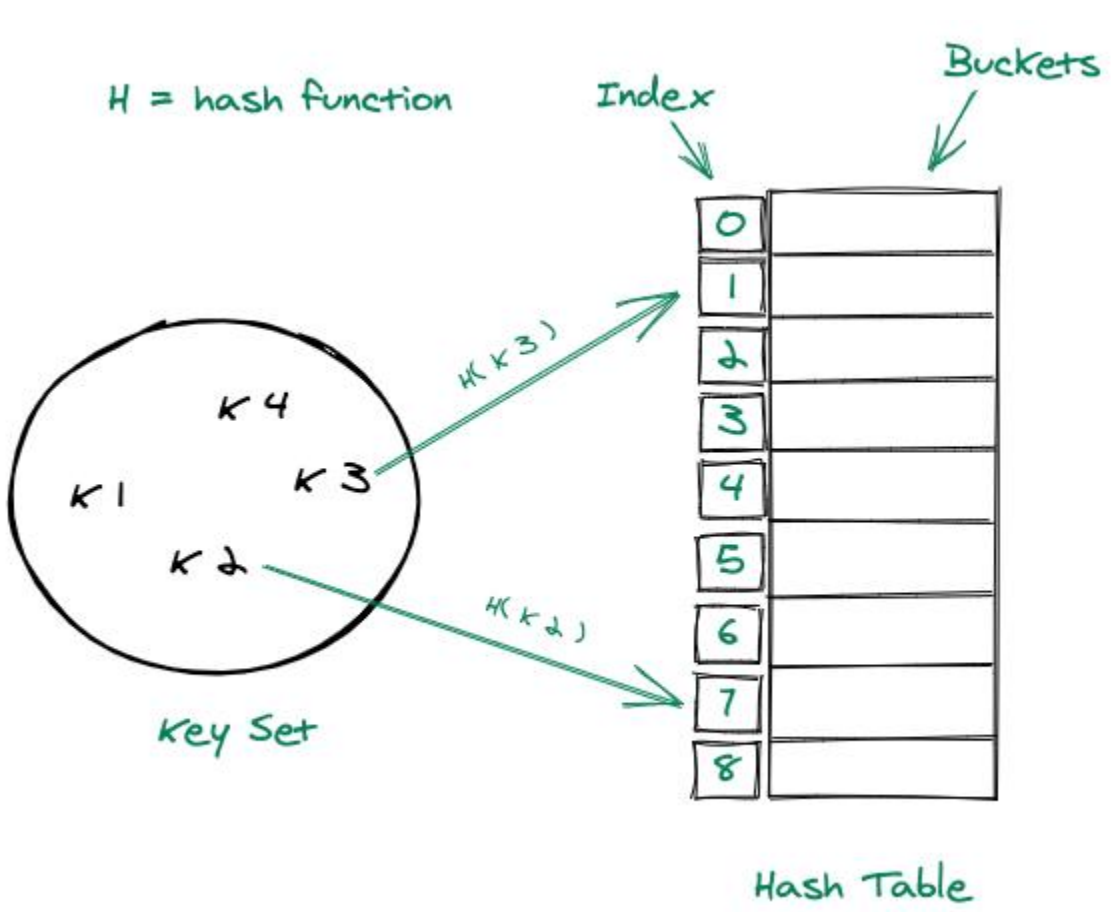
Disadvantages of Direct Address Table:

- It is not recommended using the direct address table if the key values are **very large**.
- It cannot handle the case where two keys are equal and contain different data.

Hash Table

Direct addressing has serious disadvantages, making it not suitable for the practical usage of current world scenarios, which is why we make use of Hash Tables. Unlike a direct address table where we take the key as indices of the address table, we use keys in different way. We process these keys via a hash function and the result we get from that is basically the location of the bucket where we store our data. A Hash Table has different implementations in different languages, like in Java we have **Hash Table**, **HashMap** and many more, in python we have **dictionaries**, but no matter how they are implemented the core functionality remains the same.

Consider the pictorial representation below:



Advantages of Hash Table:

- While the advantages of hash table is same when we talk about **insertion**, **deletion** or **searching** of an element, there's a huge advantage that hash table has over address table, which is that it maintains the size constraint. Let us consider a **key = 7898789**, which in turn is a large number, if we insert this in a direct address table, then we are **wasting too much space** as we will have to find this location(key) and then insert the value at that location, but in case of a hash table we can process this **key via a hash function**, say it yields us = 17, now we are only left with inserting at position(17) of the hash table.

Disadvantages of Hash Table:

- A situation might arise when we get the same bucket location for different keys via our hash function, this situation is known as collision. Though we can improve the hash quality, but we can't guarantee that collisions won't take place.

Handling Collisions in Hash Table:

There are multiple ways with which we can handle collisions. Some of them are:

A good hash function:

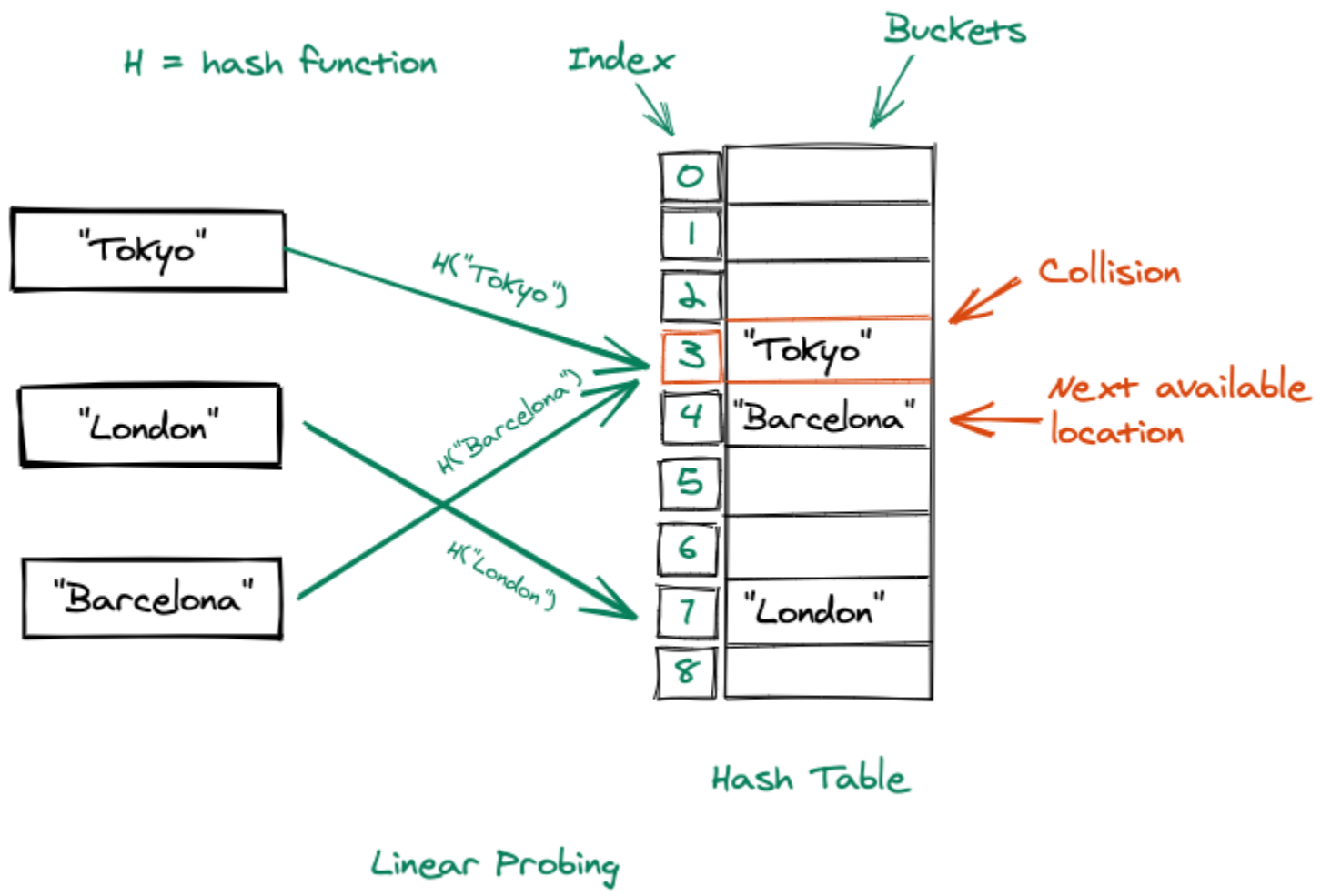
A good hash function can mean many things at the same time, but we can conclude that a hash function is considered as a good hash function, if:

- It minimizes the amount of collisions as much as possible.
- It should not generate the bucket locations that are larger than the hash table, in that case we will be wasting too much space.
- The bucket locations generated should neither be too far apart and too much closer.

Linear Probing:

It is a technique that makes sure that if we have a collision at a particular bucket, then we check for next available bucket location(just below it), and insert our data in that bucket.

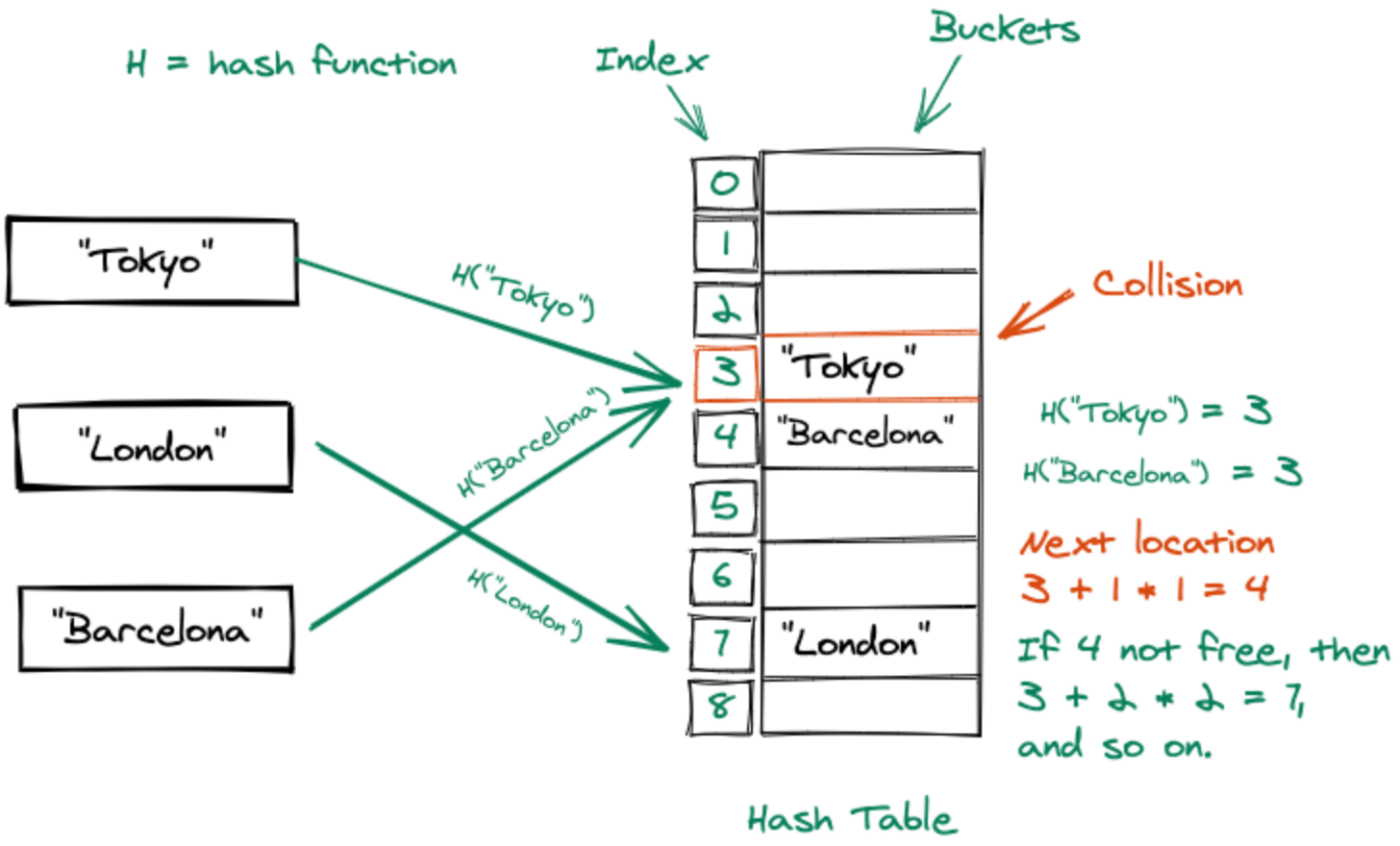
Consider the pictorial representation shown below:



Quadratic Probing:

In quadratic probing the next bucket location is decided using the formula: **h(k, i) = (h?(k) + x*i + y*i^2)** where x and y are constants. Another way to look at this to say that the distance between the next location is increased quadratically.

Consider the pictorial representation shown below:

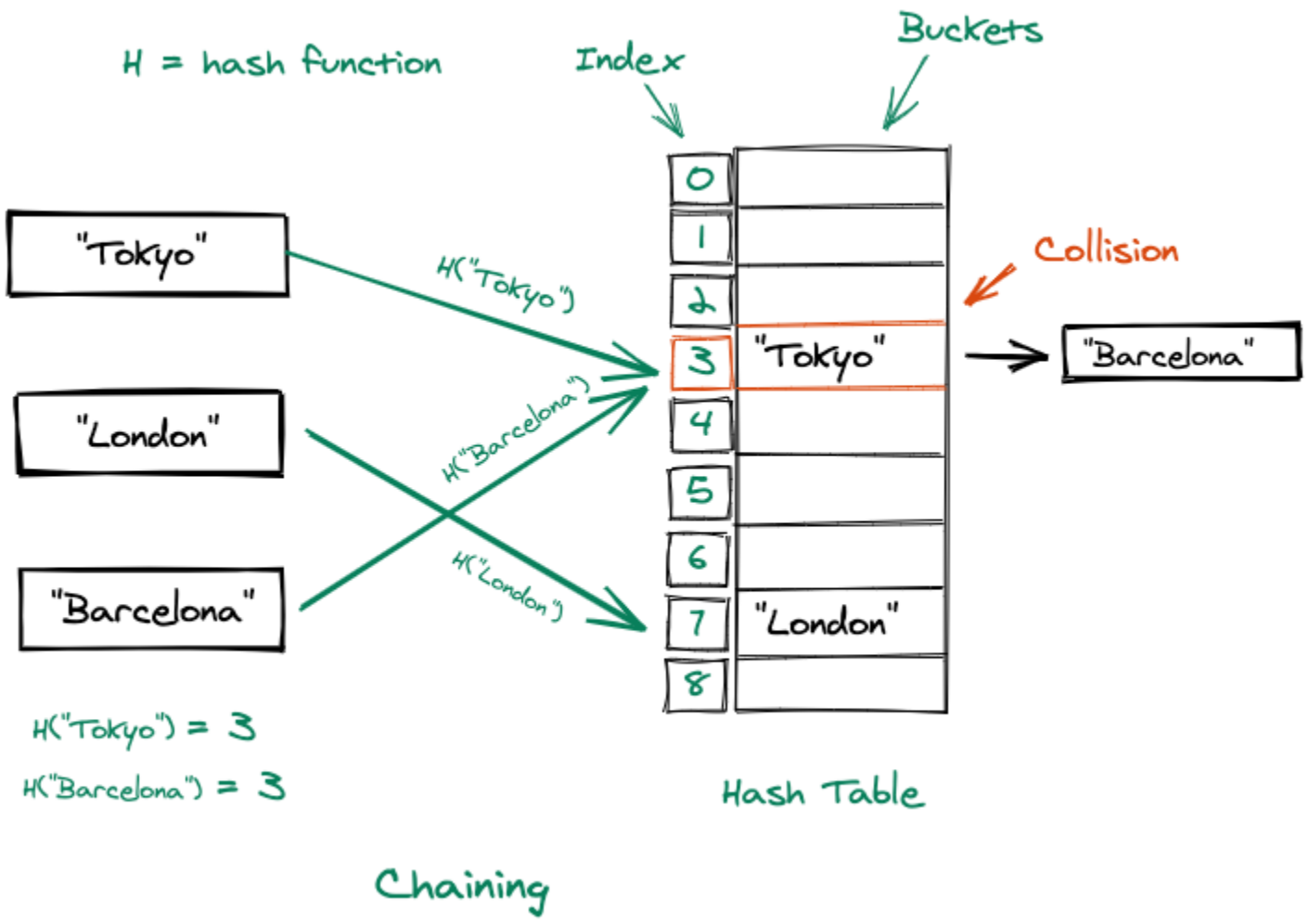


Quadratic Probing

Chaining:

In this technique of collision handling, if at a particular bucket location a collision occurs then at that location we simply create a linked list, and the value will be added as the next node of the list. Hash table becomes an array of linked list.

Consider the pictorial representation shown below:



Chaining

Applications of Hash Tables:

Pattern Matching:

Hash Tables search time complexity makes it a perfect candidate for finding a pattern in a pool of strings.

Compilers:

Compilers make use of hash tables to store the keywords and other identifiers to store the values of a programming language.

File Systems:

The mapping between the name of the file in our file system and the path of that file is stored via a map that intern makes use of a hash table.

Conclusions

- We learned what a Hash table is, what are its main components.
- Then we learned about direct address tables, followed by a detailed explanation of hash tables.
- Then we talked about different techniques of handling collisions in hash tables, followed by the real-world applications of a hash table.
