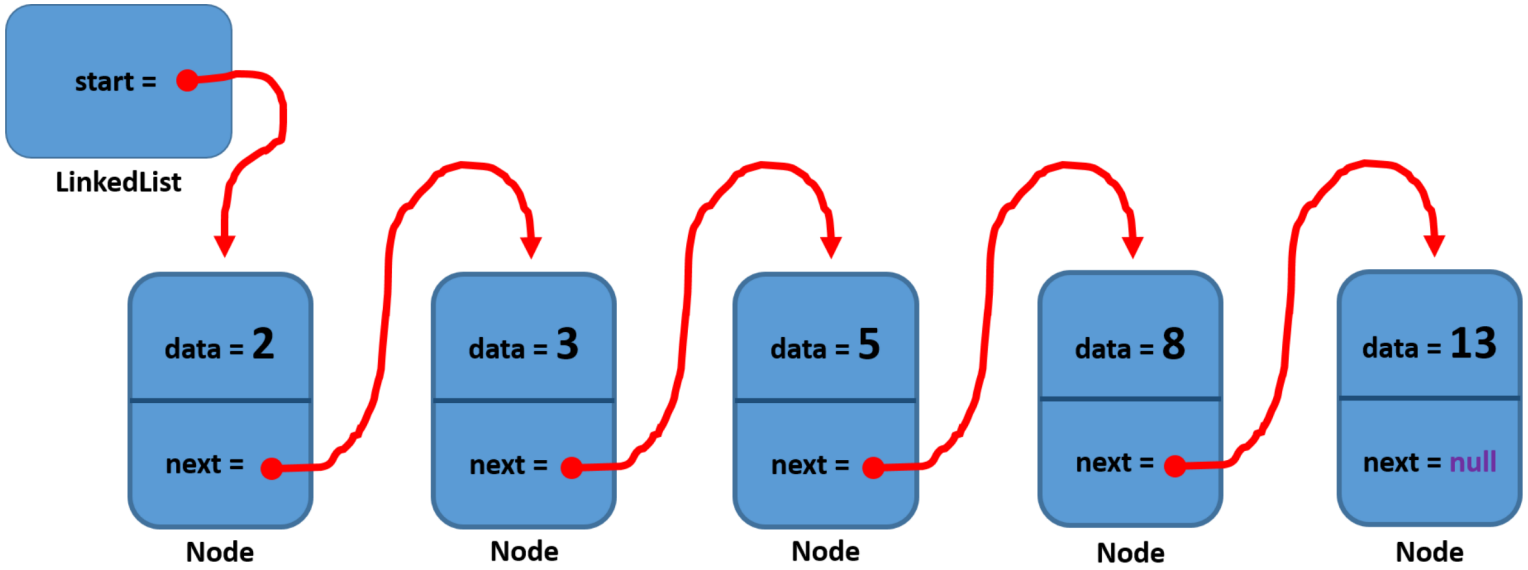
Data Structures: Linked Lists and Hash Tables

Linked Lists

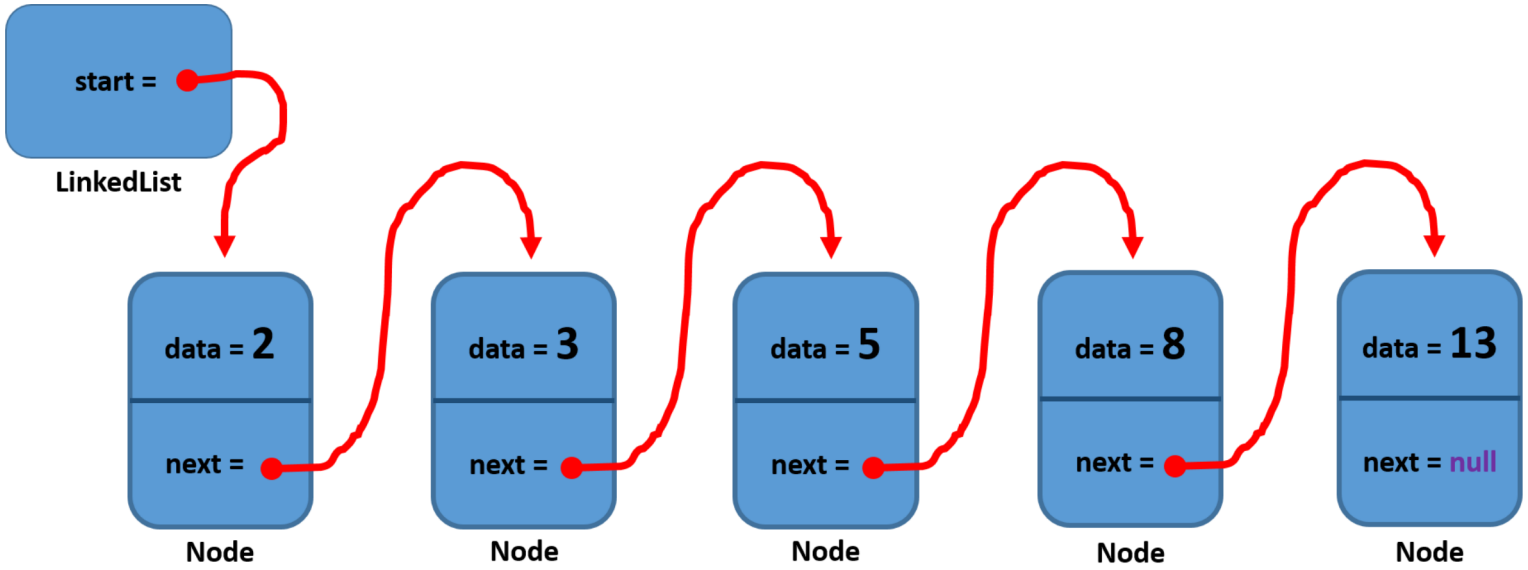
Linked Lists

This is a visual representation of a linked list using `int` values

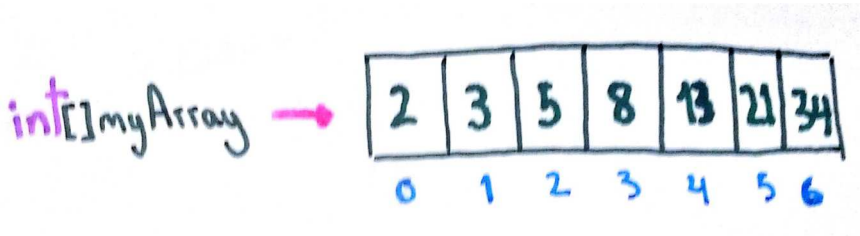


Linked Lists vs Arrays

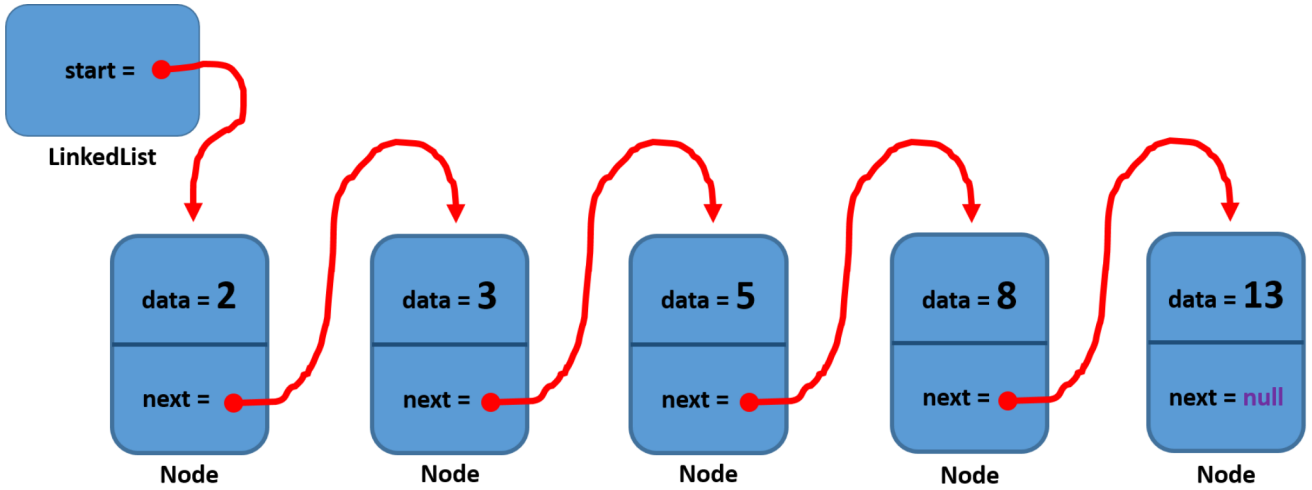
This is a visual representation of a linked list using `int` values



Why would we want to use it instead of an Array?



Operations with LinkedLists

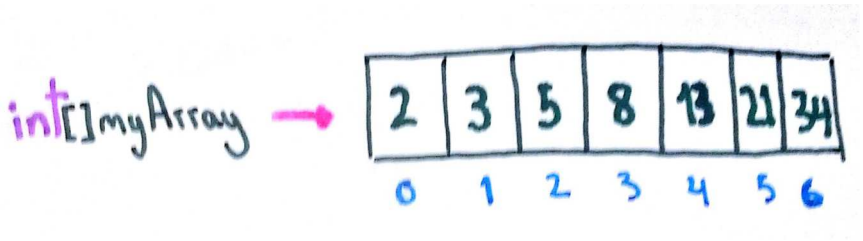


- A `LinkedList` allows us to add or remove elements without having to copy the whole data structure
- In a `LinkedList` with `n` elements, all operations (search, insertion, deletion) take at most `n` comparisons
    - In the worst case you'll want to reach the end of the list. This is only possible by following the `next` pointer of every node.

In big O notation

Operations in a LinkedList have a running time of ==> O(n)

Operations with Arrays



- In a array with `n` elements, you **cannot change the size** of the array without creating a new one and copying its contents (recall the `addVertex` of the `Polygon` class, or the `addFriend` method of the `Person` class)
- But if you know what you are looking for, accessing an element from the array requires only **1** operation
  - To get the `n`-th element of a `LinkedList` we need to follow the `next` pointer `n` times
  - To get the `n`-th element of an array we just need to type: `myArray[n]`

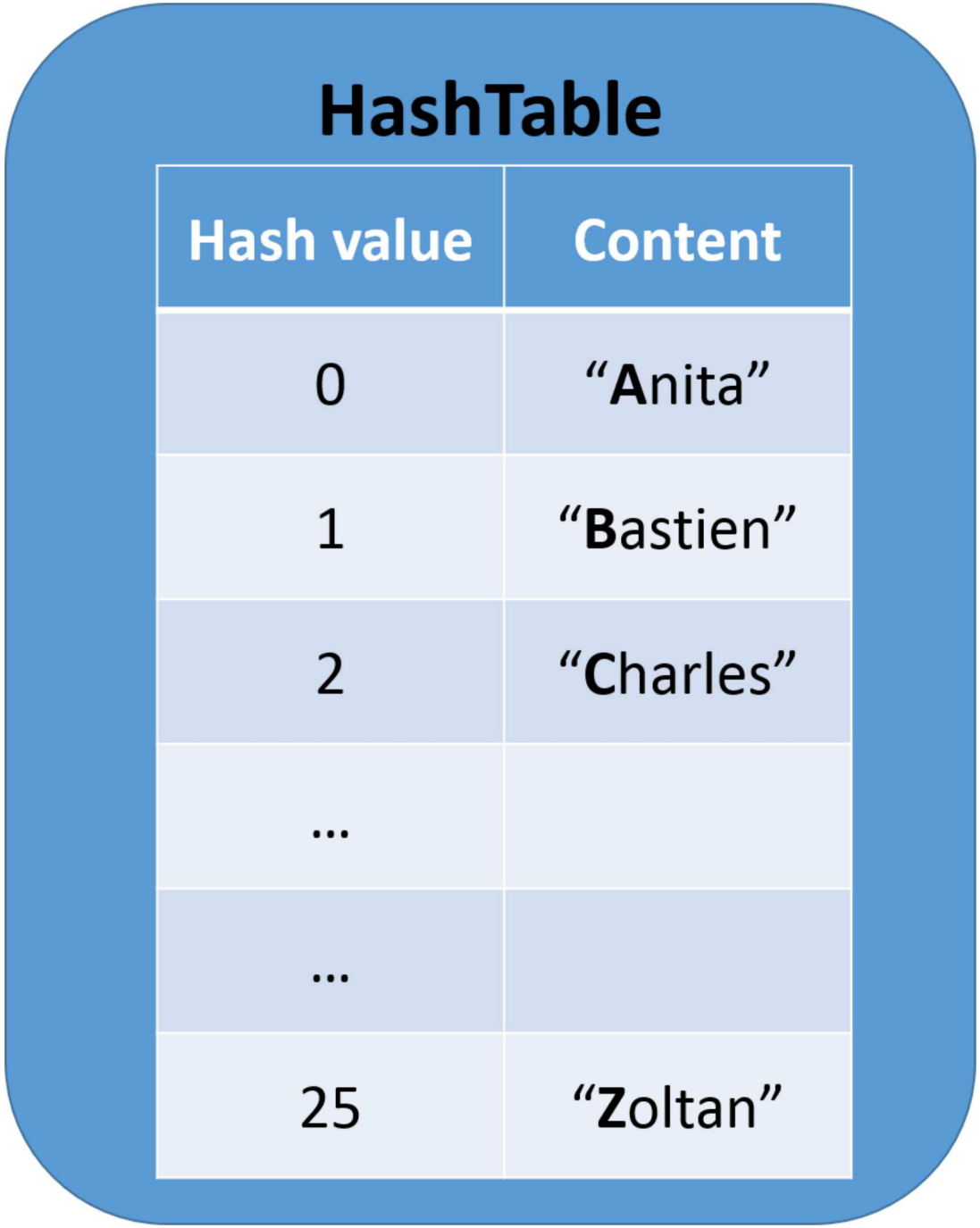
How can we get both fast access and variable size?

If access is fast on an array, can we do something similar for search, insertion and deletion?

Hash tables

Hash tables

You can think of hash tables as doing the inverse operation that you do with an array.

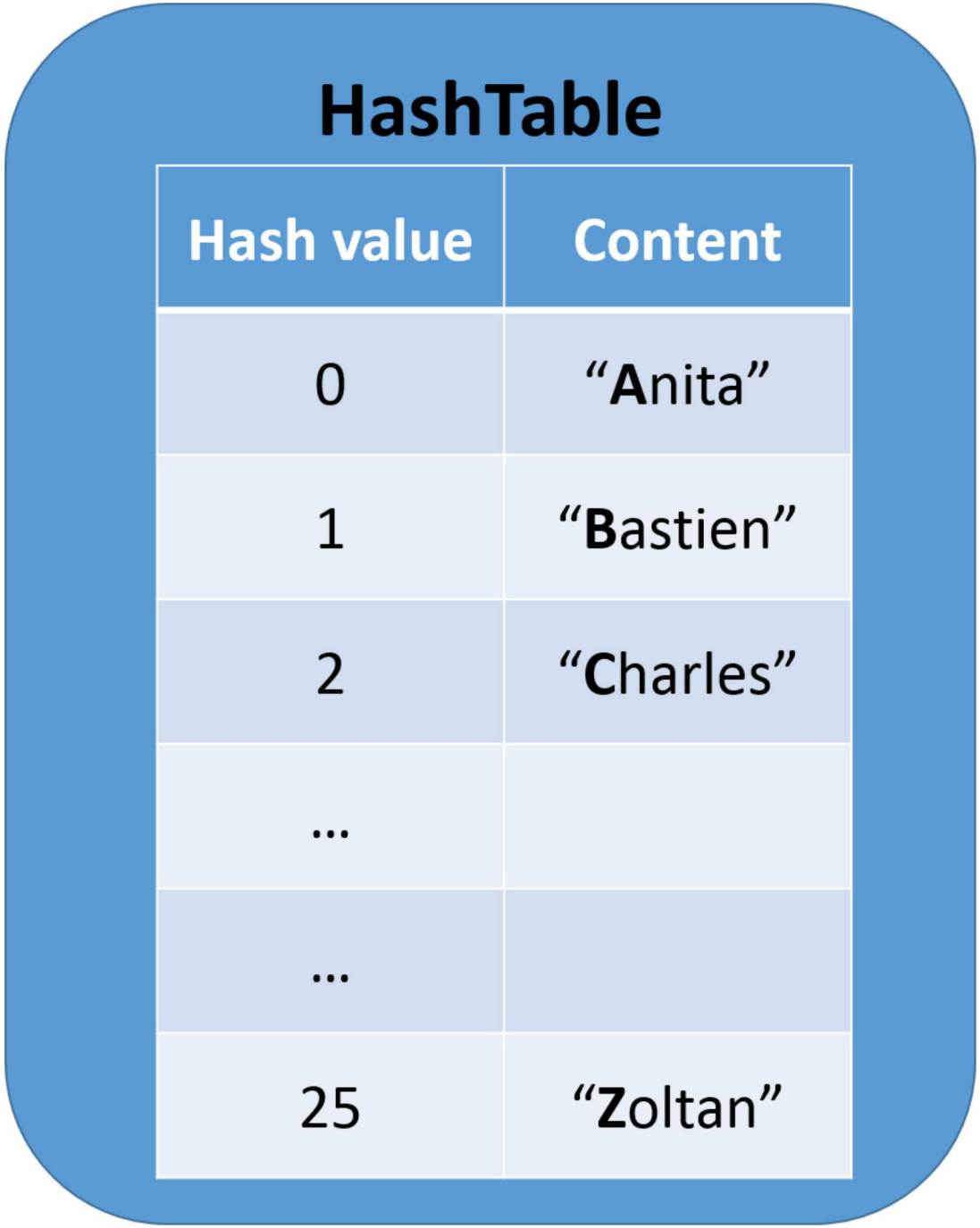


In an array you use an value (e.g. an `int` number) and you get a key (e.g. a `String`)

In a HashTable you use a key (e.g. a `String`) to get an value (e.g. an `int` number)

Hash tables

Hash tables will allow us to combine some good things about arrays, with some good things about LinkedLists

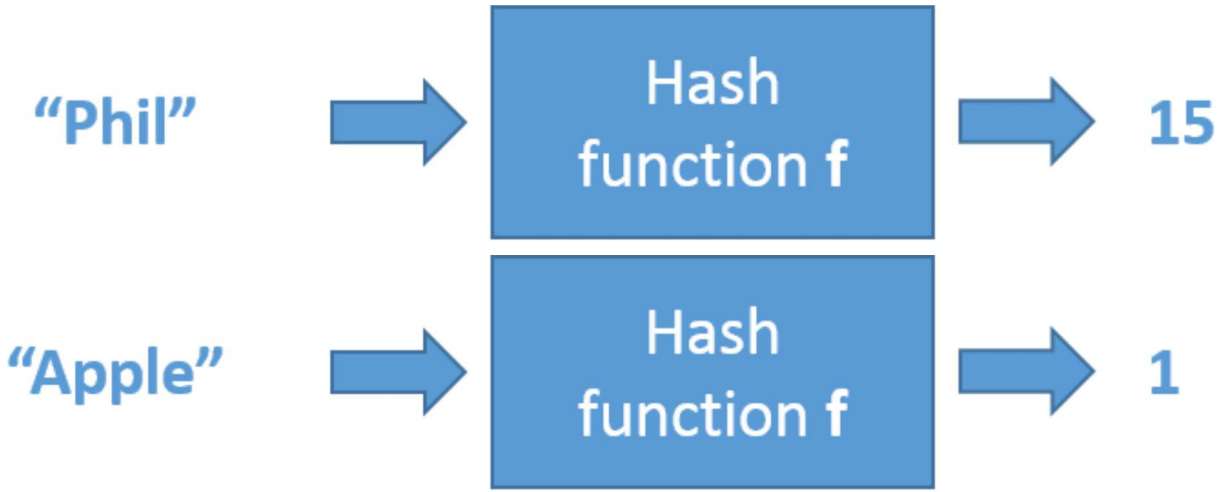


In a HashTable you use a key (e.g. a `String`) to get an value (e.g. an `int` number)

To do this we will use a hashing function

Hashing function

A hashing function maps input data of arbitrary size to output data of fixed size

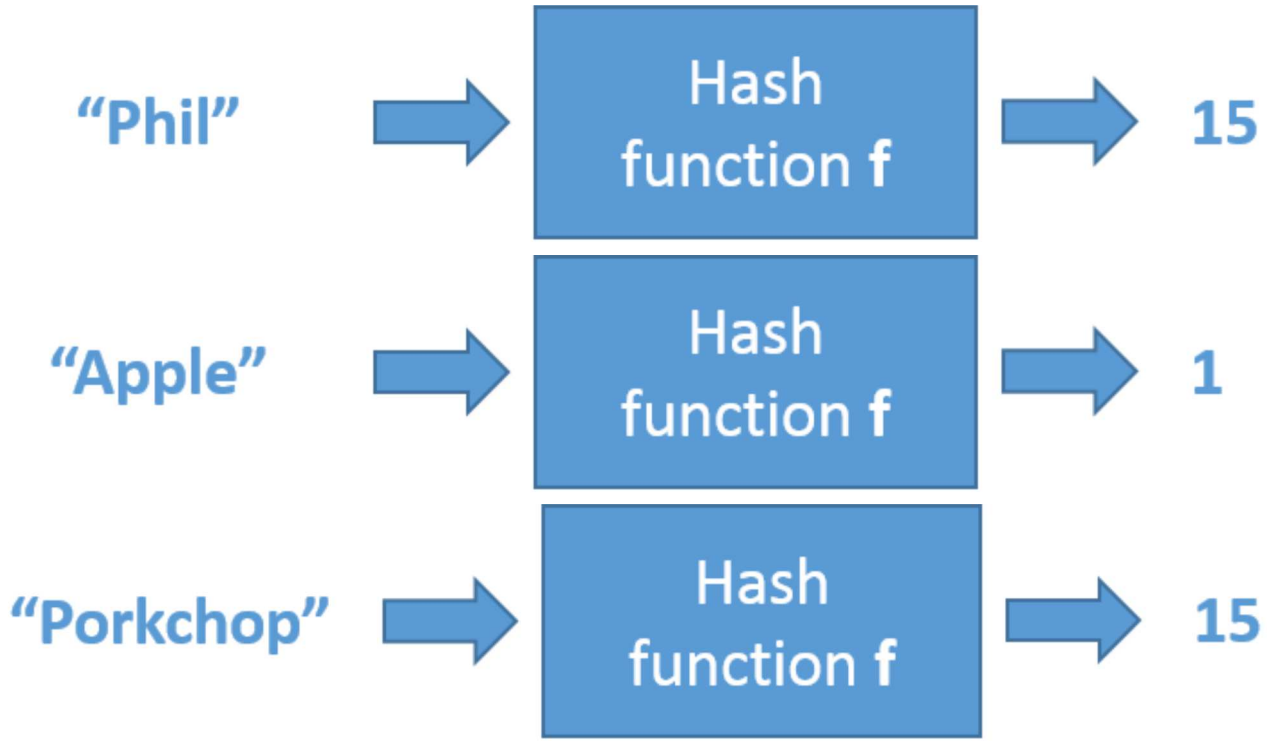


We call the input a key. We call the output a Hash Value

We can use the has value as an index to find data, e.g., in an array

Hashing function

A hashing function maps input data of arbitrary size to output data of fixed size

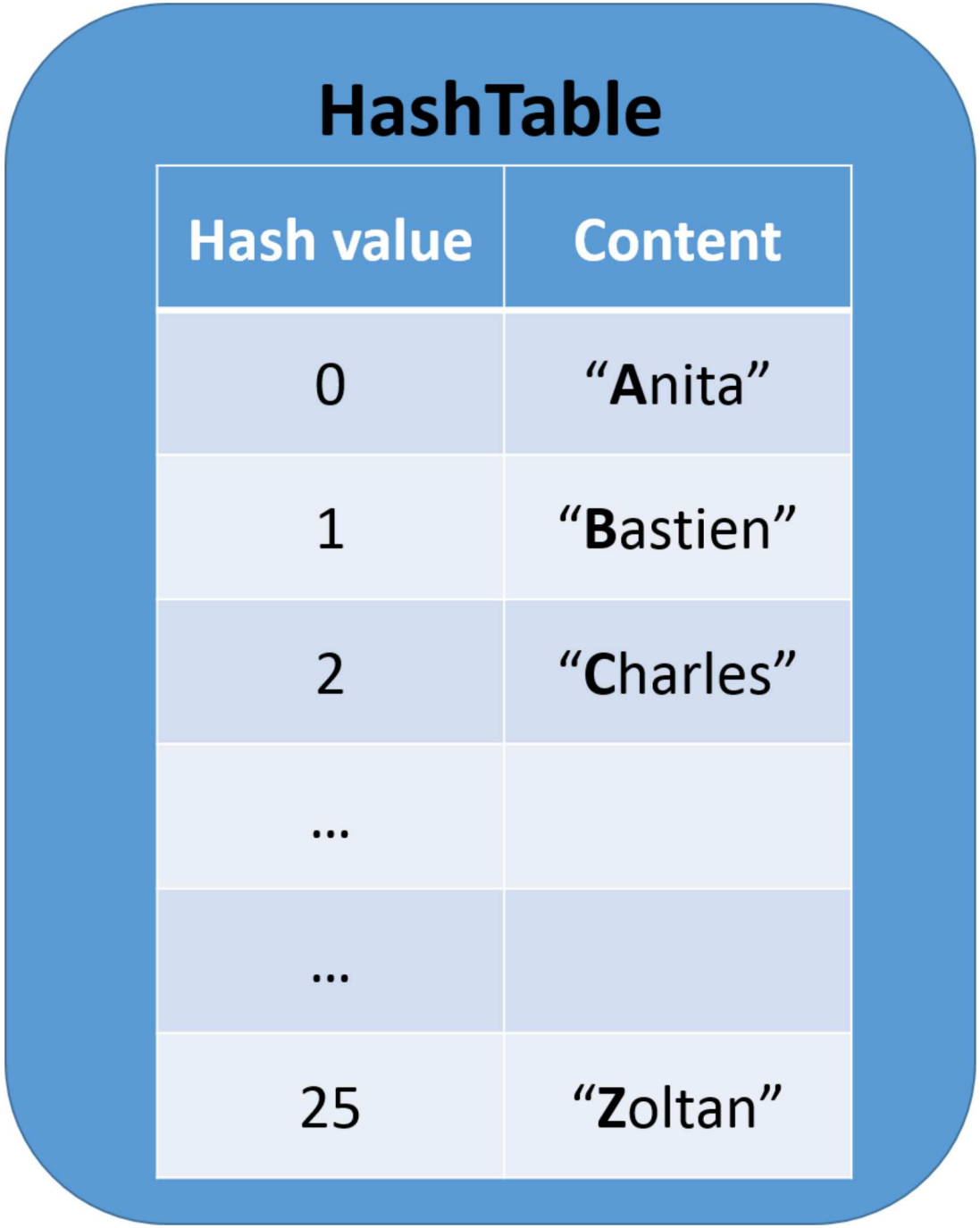


Arbitrary size: Person names in English, Fixed Size: Groups of names by their first letter

- In our example, we only have 26 possible values. But there are more than 26 possible person names
- Inevitably, multiple elements will have the same same hash value.

Implementing a HashTable

A hash table will consist of a collection of entries, a `hashing function`, and methods to insert and remove data from it



We will use an array to store the entries in our data structure

Implementing a HashTable

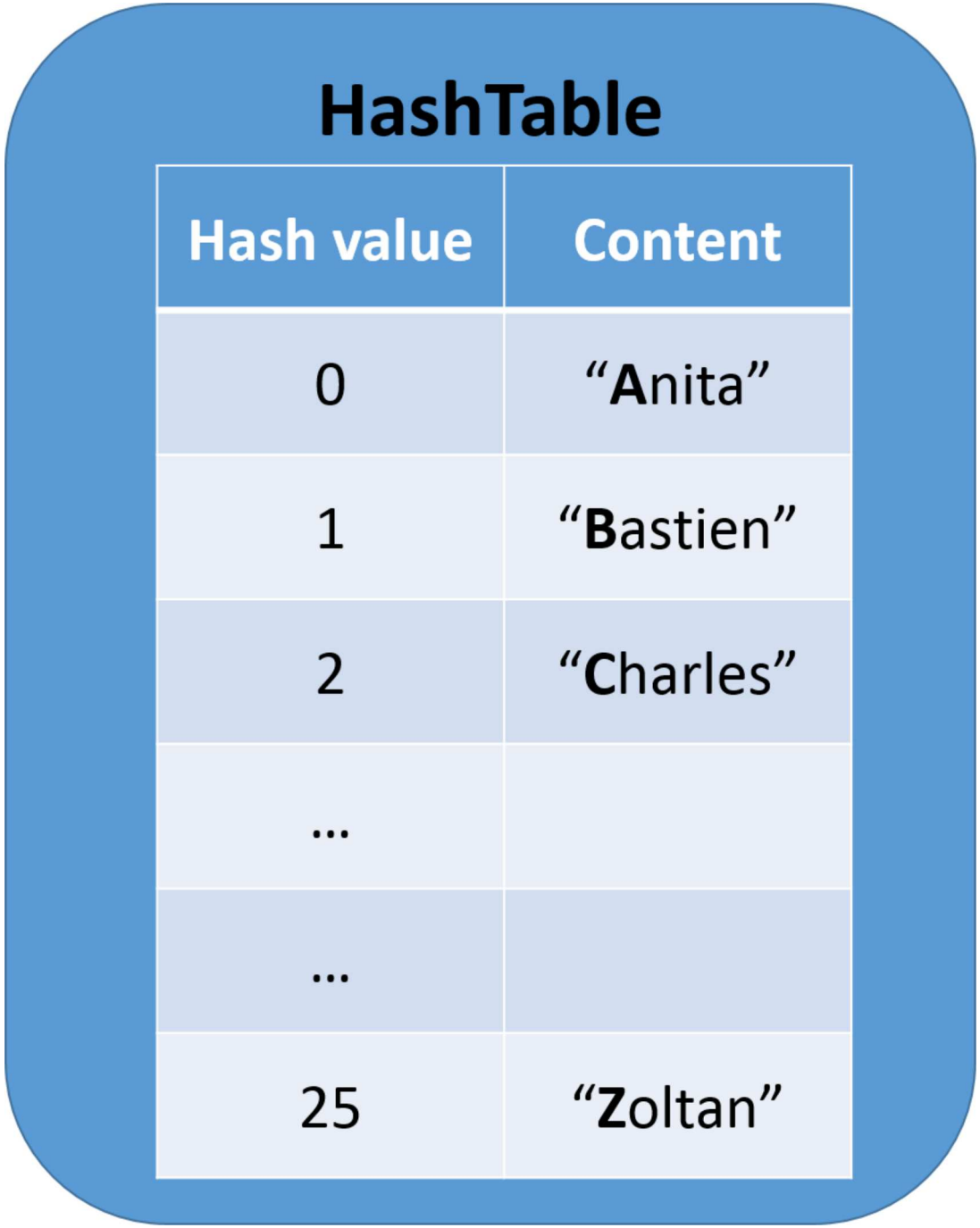
A hash table will consist of an collection of entries, a `hashing function`, and methods to insert and remove data from it

We use an array to store the entries in our data structure

```
1     public class Hashtable{
2         /*
3          * the entries could be of any type
4          * here, we use the String entries, for example.
5          */
6         private String[] entries;
7
8         // add a constructor here
9
10        // add Insertion, search and deletion methods here
11
12        /*
13         * The hashing function an index  in the entries
14         * array for the given element
15         */
16        public int hashFunction( String name ){
17            // calculate an int ( the hash values)
18            // for the String name
19        }
20    }
21
```

Collision in Hash Tables

When two elements have the same has value, we will have to put them in the same entry



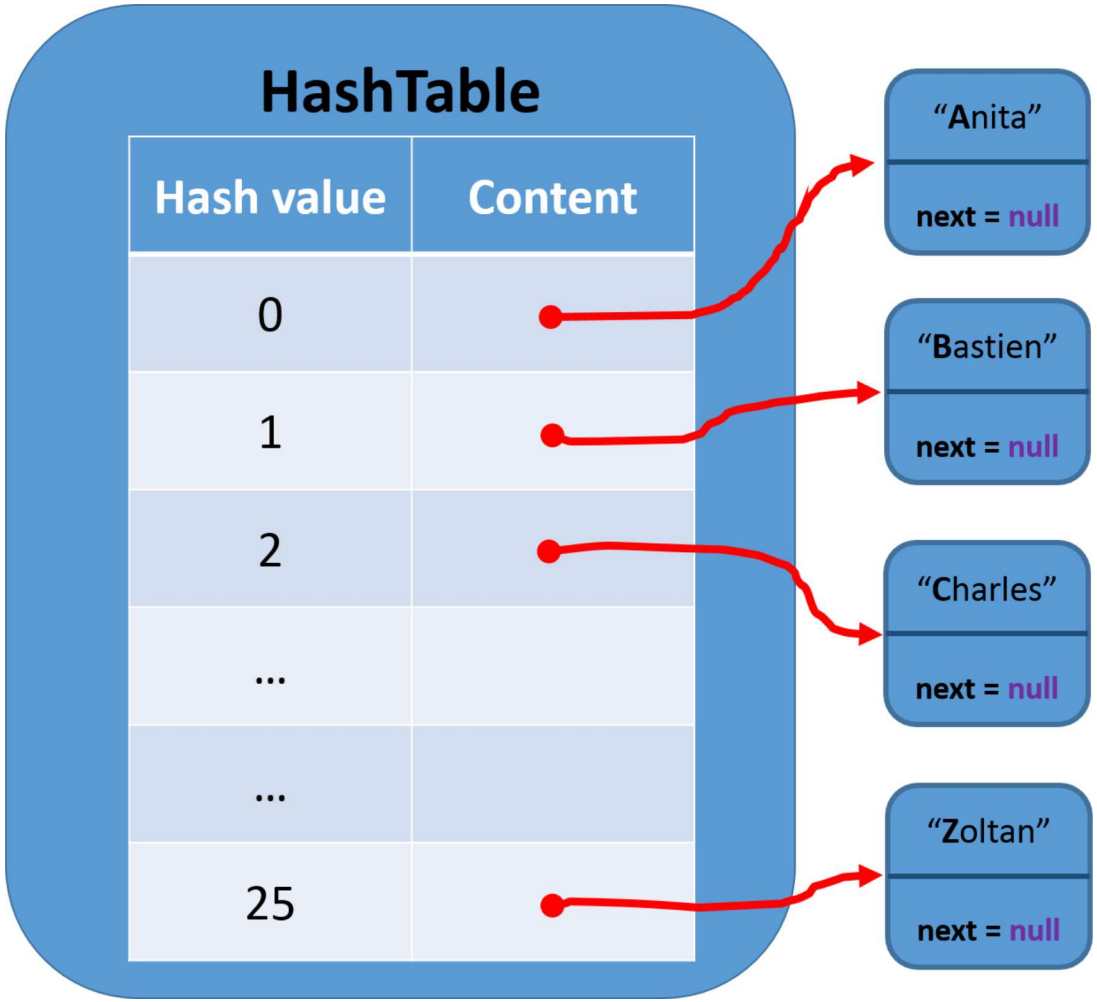
We call this situation a collision

How do we deal with collisions

- **Option 1:** Change the hash function so that we have more possible hash values? (E.g. Use the first two letters of a name to compute its hash value.)
  - If the input data is larger than the output data, we will **always** have collisions
- **Option 2:** Replace each entry in the hash table with a `LinkedList`

Hash tables with multiple elements per entry

Replace each entry in the hash table with a `LinkedList`



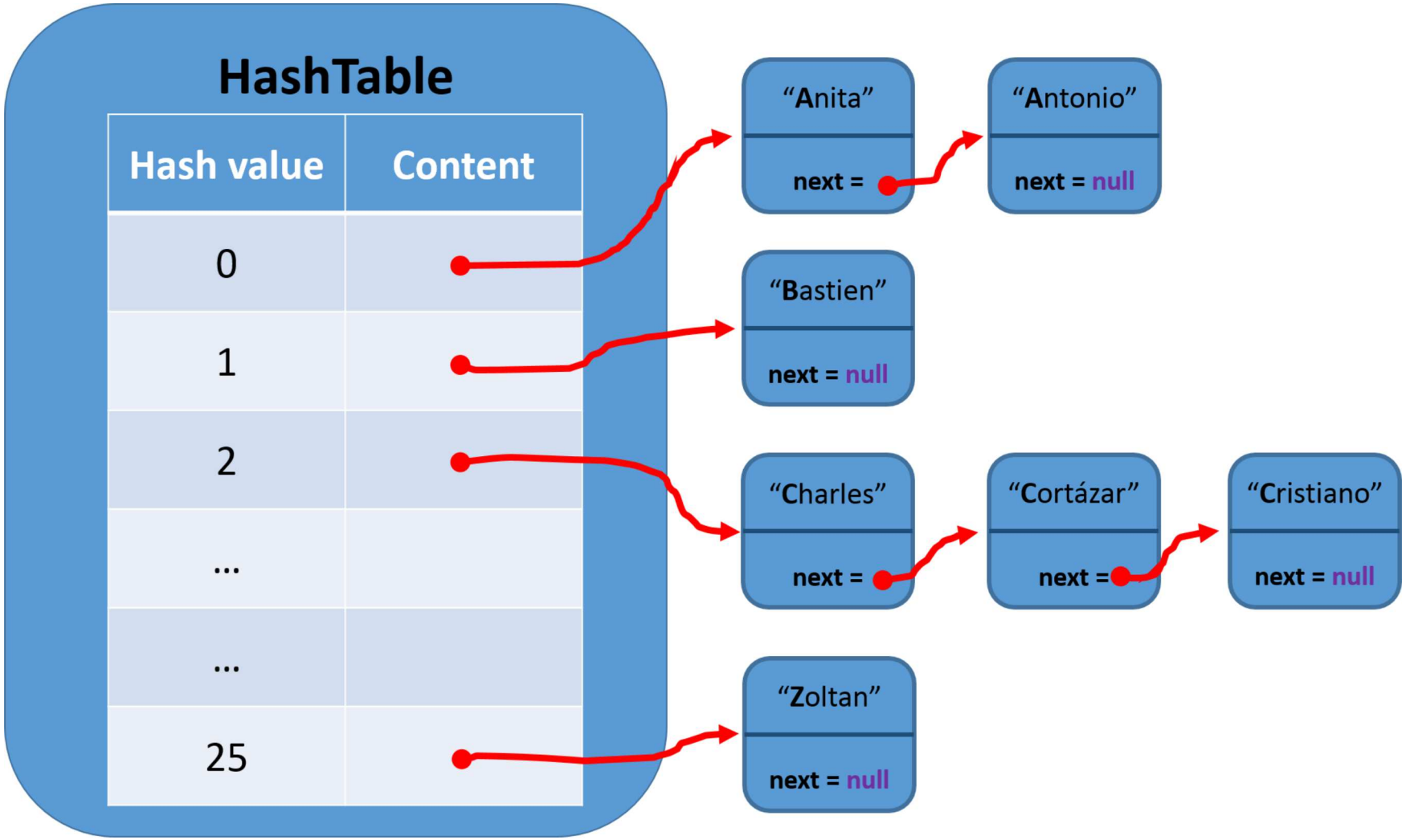
Hash tables with multiple elements per entry

Replace each entry in the hash table with a `LinkedList`

```
    public class HashtableL{
        /*
         * the entries could be of any type.
         * here, we use a LinkedList to keep
         * multiple entries in each poistion
         * of the array.
         */
        private LinkedList[] entries;

        // add a constructor here

        // add Insertion, search and deletion methods here

        /*
         * The hashing function an index  in the entries
         * array for the given element
         */
        public int hashFunction( String name ){
            // calculate an int ( the hash values)
            // for the String name
        }
    }

```

Hash tables with multiple elements per entry

Replace each entry in the hash table with a `LinkedList`



Some remarks about Hash tables

- A `Hashtable` allows us to add or remove elements quickly by making use of a **hash function**
- To deal with multiple elements in a single location in the dictionary, we combine the hashing function with a `LinkedList`

In big O notation

Operations in a Hash Table with `n` elements have a running time of ==> O(k)

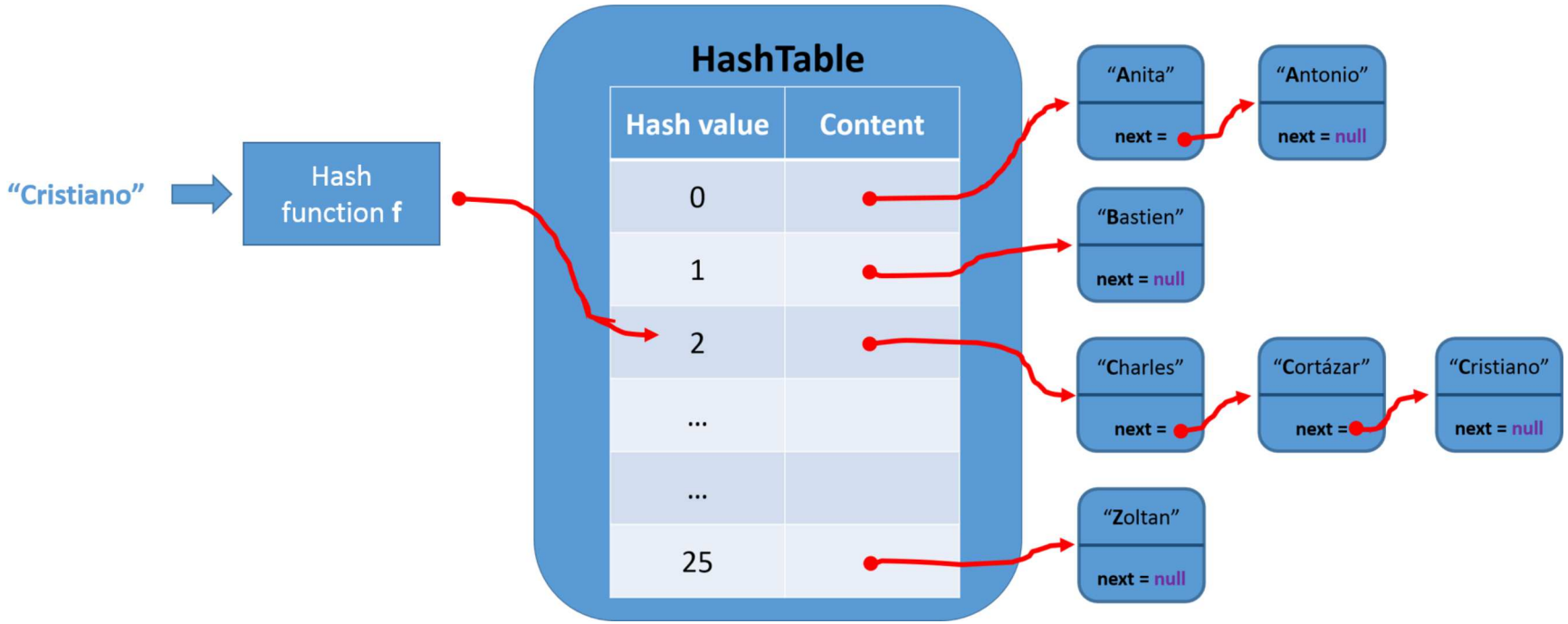
- `k` is size of the longest `LinkedList` in the HashTable.

Operations in a Hash Table with `n` elements have a running time of ==> O(1)

- When `k` is considerably smaller than `n`

You can view a HashTable as a Dictionary

We map names (keys) to Nodes in the HashTable (values)



But you could use this data structure to map arbitrary types of data

- Names to phone numbers
- Phone numbers to names
- English words to their definitions
- English words to words in French
- Musical note letters to their pitch frequency

HashTables in Java

Java has a generic implementation of a HashTable

```
    Hashtable<key, value> dictionary = new Hashtable<key, value>();
    //This creates a hash table that will be
    //indexed by variables of type key and
    //contain values of type value.
    //We can add entries using the .put() method.
    name.put(key, value);
    //We can obtain values using .get() method.
    name.get(key);

```

To try for yourself

Try to implement an English to French dictionary using a HashTable<String,String> object

```
    Hashtable<String, String> englishToFrench = new Hashtable<String, String>();
    //This creates a hash table that will be
    //indexed by variables of type key and
    //contain values of type value.
    //We can add entries using the .put() method.
    englishToFrench.put("Good morning", "Bonjour");
    //We can obtain values using .get() method.
    englishToFrench.get("Good morning");

```

Resources

- http://en.wikipedia.org/wiki/Hash_table
- http://www.tutorialspoint.com/java/java_hashtable_class.htm

Drawing Tools