Hash Tables and Maps

CSCI 2100 Data Structures

Spring 2018

Tae-Hyuk (Ted) Ahn

Department of Computer Science
Program of Bioinformatics and Computational Biology
Saint Louis University

SAINT LOUIS UNIVERSITY™

EST. 1818

STL Containers

- Sequence Containers – store sequences of values
  - vector, deque, list
- Associative Containers – use "keys" to access data rather than position (Account #, ID, SSN, …)
  - set
  - multiset
  - map
  - multimap
- Container Adapters – specialized interfaces to general containers
  - stack, queue, priority_queue

Associative Containers

- Stores elements based on a key
- Key can consist of one or more attributes to uniquely identify each element (we will assume only one attribute).
- Example: Department of Motor Vehicles (DMV) uses license-plate # to identify a vehicle.
- Similar to vector & list – it is another storage structure with operations to access & modify elements.
- Main difference is that associative-container uses the key rather than an index (vector) or linear search (list) to retrieve an element.

Associative-Container : set

- Stores a set of values (i.e., “keys”)
- Values are unique (stored only once)
- Implemented as a binary search tree
  - `#include <set>`
  - `set<string> s;`
- Fast insert and delete
  - `insert, erase`
- Fast search
  - `find`
- Other operations
  - `size, empty, clear, . .`

StdSet.cpp

2. vim

```cpp
#include <iostream>
#include <set>
#include <string>

using namespace std;

int main() {
    set<string> setOfNumbers;

    // Lets insert four elements
    setOfNumbers.insert("first");
    setOfNumbers.insert("second");
    setOfNumbers.insert("third");
    setOfNumbers.insert("first");

    // Only 3 elements will be inserted
    cout<<"Set Size = "<<setOfNumbers.size()<<endl;

    // Iterate through all the elements in a set and display the value.
    for (set<string>::iterator it=setOfNumbers.begin(); it!=setOfNumbers.end(); ++it) {
        cout << ' ' << *it;
    }
    cout<<"\n";

    // Search for element in set using find member function
    set<string>::iterator it = setOfNumbers.find("second");
    if (it != setOfNumbers.end()) {cout<<"'first'  found"<<endl;}
    else {cout<<"'first' not found"<<endl;}

    // Search for element in set using find member function
    it = setOfNumbers.find("fourth");
    if(it != setOfNumbers.end()) {cout<<"'fourth'  found"<<endl;}
    else {cout<<"'fourth' not found"<<endl;}

    return 0;
}
```

Associative Containers: multiset

- Stores a set of values (i.e., "keys")
- Like set, but values need not be unique
- Implemented as a balanced binary search tree (red-black tree)
  - `#include <set>`
  - `multiset<string> ms;`
- Fast insert and delete
  - `insert, erase`
- Fast search
  - `find`
- Other operations
  - `size, empty, clear, . . .`

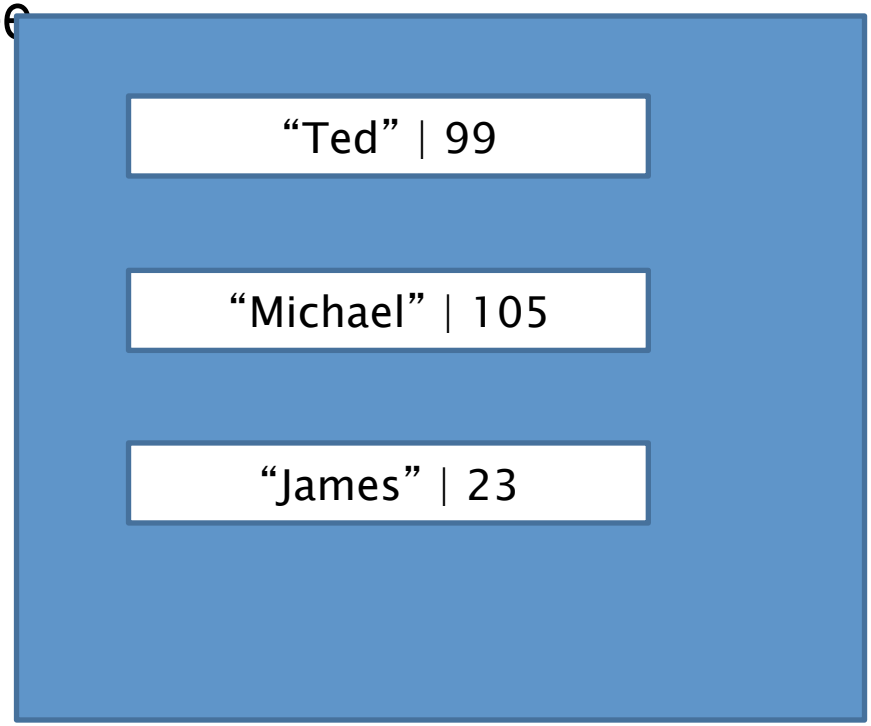
Associative Containers: map

- Stores a set of (key, value) pairs
- Each key has one value
- Implemented as a binary search tree

```
#include <map>
//define a map with
//keys of type string
//and values of int
map<string, int> m;
```

- Fast insert and delete

```
m["Ted"] = 99;
insert, erase
```



Associative Containers: map

- Fast search
  - `int x = m[“Ted”];`
  - `find`
- Other operations
  - `size, empty, clear, . . .`

STL Maps: Constructors

- Copy constructor:

```
map<char,int> m;
map<char,int> m2(m);
```

STL Maps: Data Storage

- An STL map is implemented as a tree-structure, where each node holds a "pair"

- Most important to know when retrieving data from the table
  - Some functions return the pair, not just the value

- A pair has two fields, *first* (holding the key) and *second* (holding the value)

STL Map: Data Storage

- If you have a ***pair object***, you can use the following code to print the key and value:

```
cout << myPairObject.first << " " <<
   myPairObject.second;
```

- If you have a ***pointer to the pair object***, use the arrow operator instead

```
cout << myPairObject->first << " " <<
   myPairObject->second;
```

STL Map: Data Storage

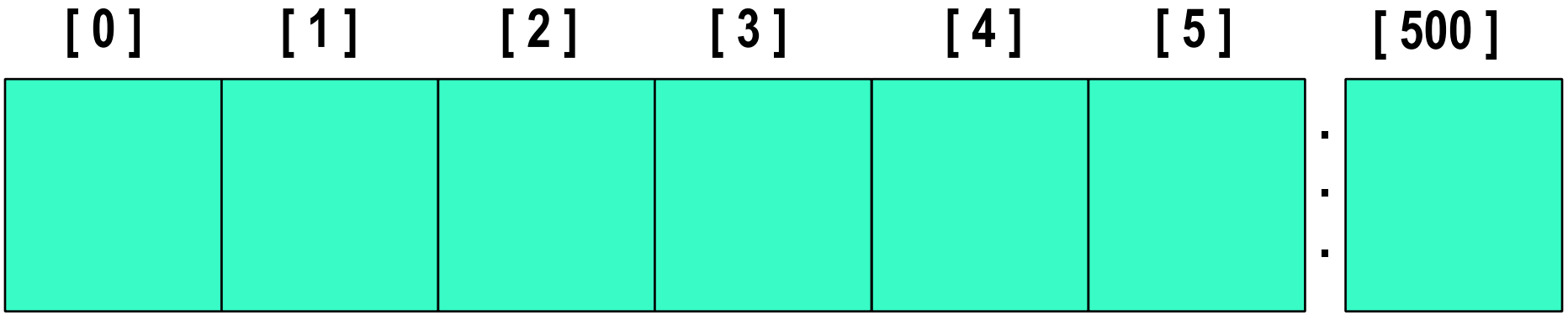
- Access element `at`
  - Returns a reference to the mapped value of the element identified with key k.
  - If k does not match the key of any element in the container, the function throws an out_of_range exception.
- Access element `[]`
  - If k matches the key of an element in the container, the function returns a reference to its mapped value.
  - If k does not match the key of any element in the container, the function inserts a new element with that key and returns a reference to its mapped value.

StdMap.cpp

```cpp
#include <iostream>
#include <string>
#include <map>

using namespace std;

int main () {
  map<int,string> mymap;

  mymap[1]="Banana";
  mymap[2]="Peach";
  mymap[3]=mymap[2];

  cout << "mymap[1] is " << mymap[1] << '\n';
  cout << "mymap[2] is " << mymap[2] << '\n';
  cout << "mymap[3] is " << mymap[3] << '\n';

  mymap.at(1) = "Melon";
  mymap.at(2) = "Strawberry";
  mymap.at(3) = "Kiwi";

  map<int,string>::iterator it;
  for (it=mymap.begin(); it!=mymap.end(); ++it)
    cout << it->first << " => " << it->second << endl;

  it = mymap.find(2);
  if (it != mymap.end())
    cout << "The value of the key [2] is " << it->second << endl;

  return 0;
}
```

What is a Hash Table?

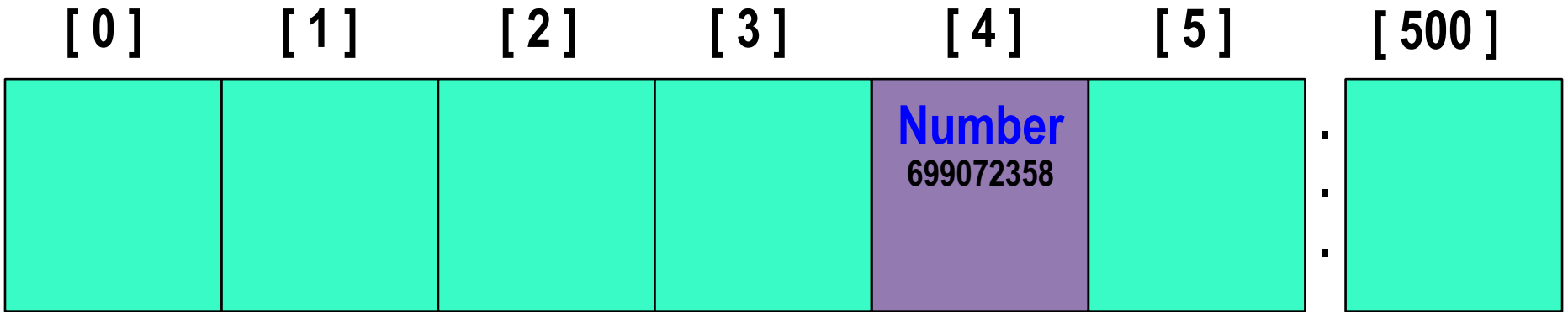
- The simplest kind of hash table is an array of records.
- This example has 501 records.



An array of records

What is a Hash Table?

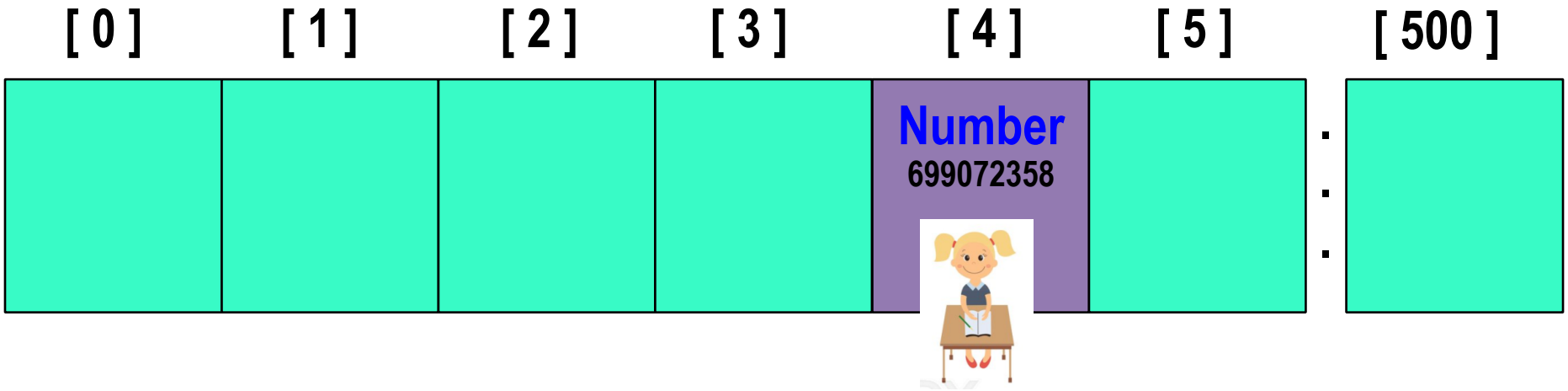
- Each record has a special field, called its key.
- In this example, the key is a long integer field called Number.



An array of records

What is a Hash Table?

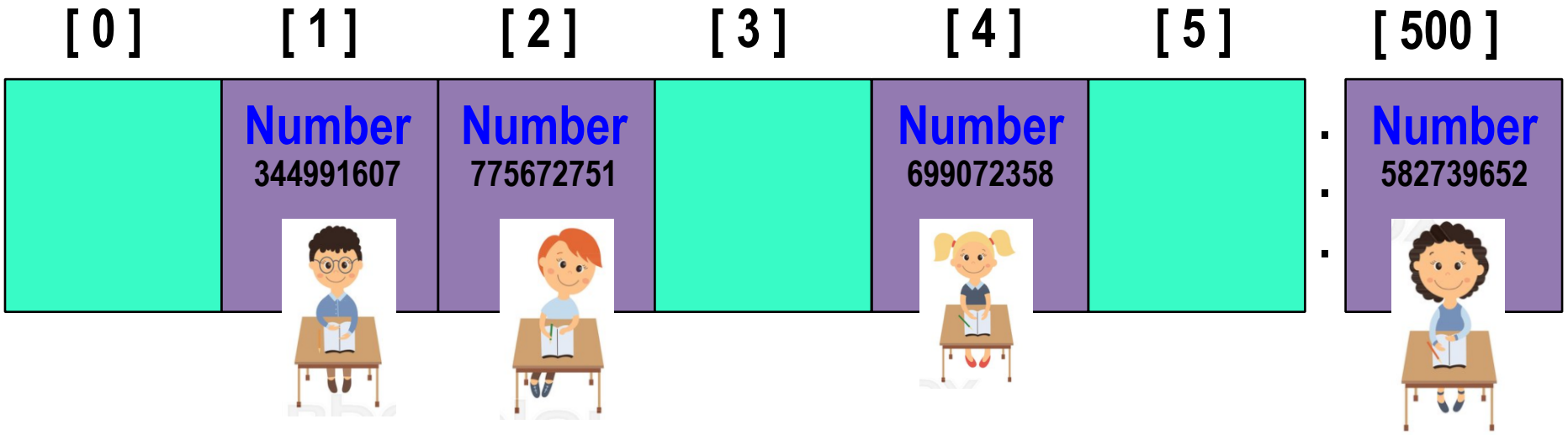
- The Number might be a person's identification number (e.g., student ID, SSN), and the rest of the record has information about the person.



An array of records

What is a Hash Table?

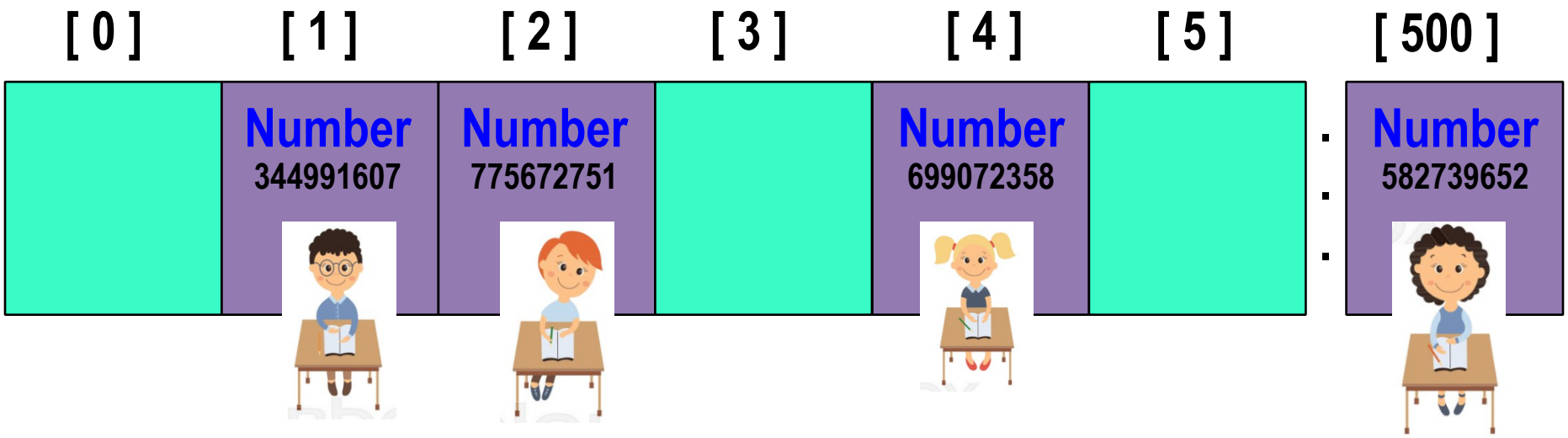
- When a hash table is in use, some spots contain valid records, and other spots are "empty".



An array of records

Inserting a New Record

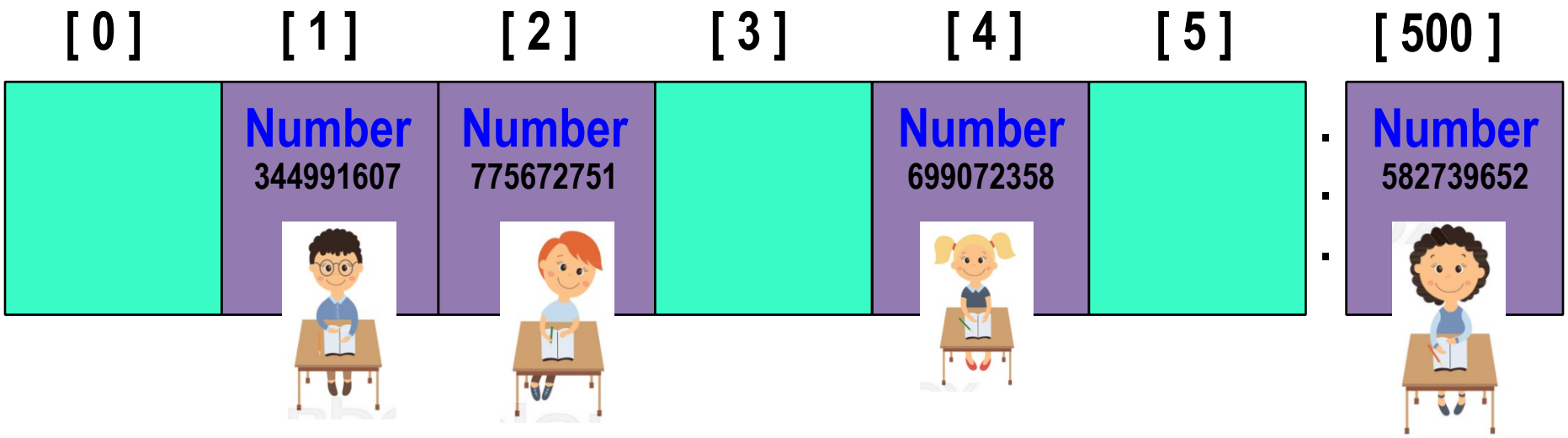
- In order to insert a new record, the key must somehow be converted to an array index.
- The index is called the hash value of the key.



An array of records

Inserting a New Record

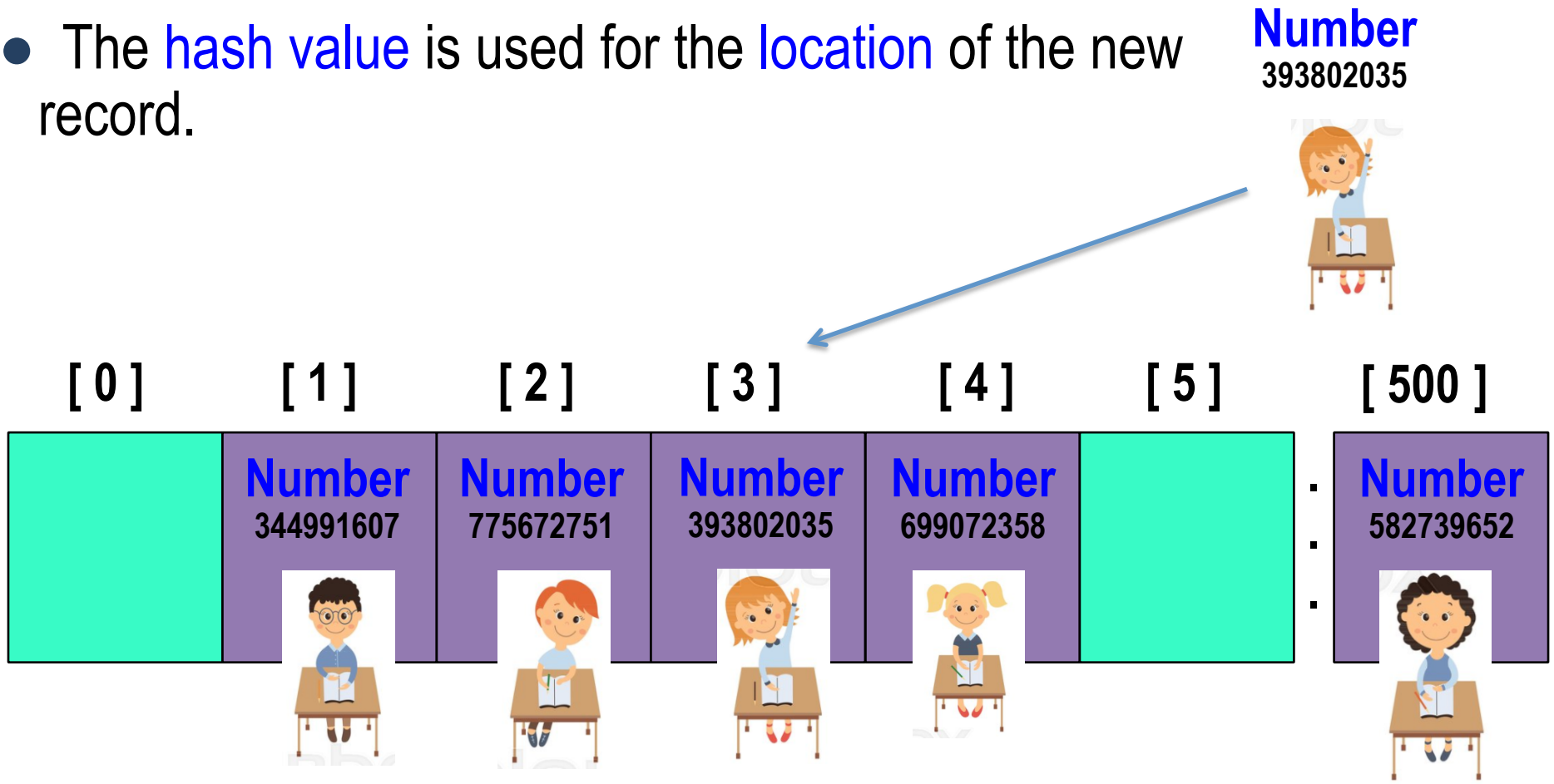
- Typical way create a hash value:
  - Number mod ArraySize
  - 393802035 mod 501 = 3



An array of records

Inserting a New

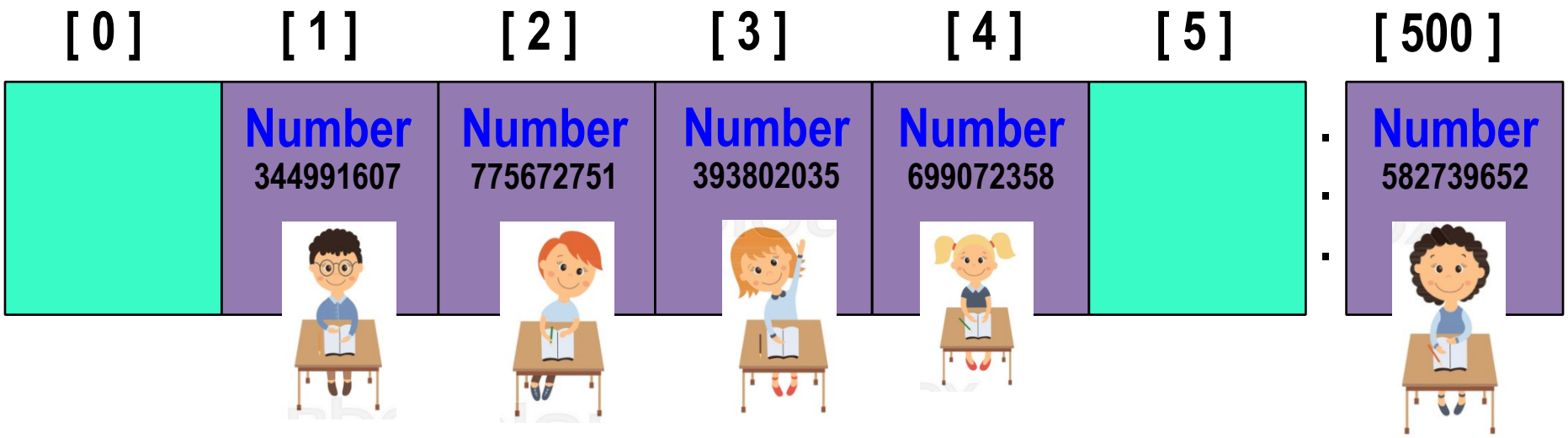
- The hash value is used for the location of the new record.



An array of records

Hash Collisions

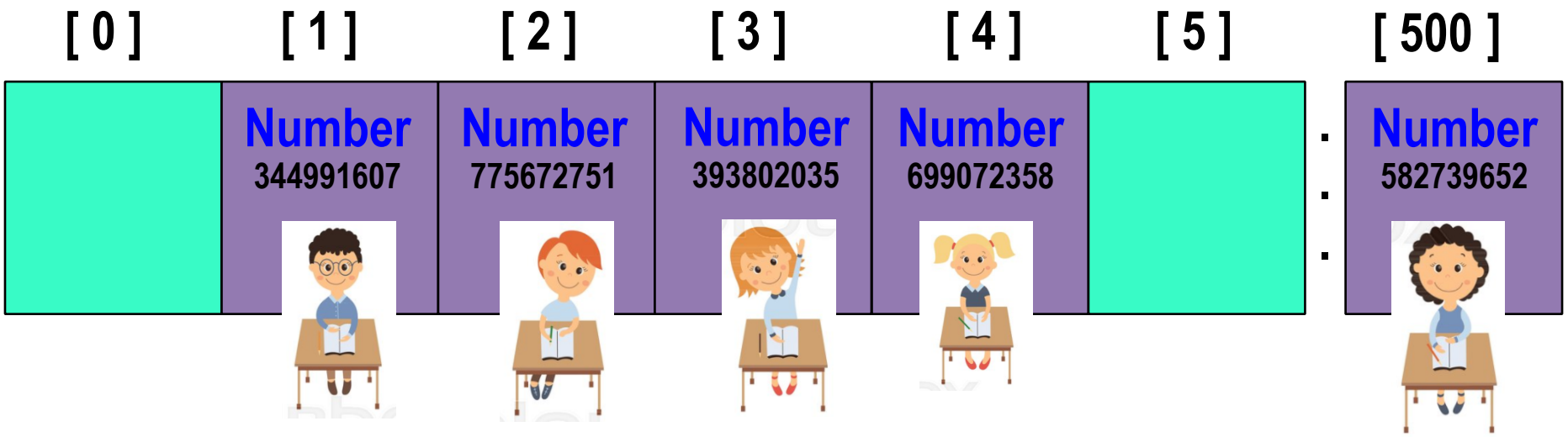
- Here is another new record to insert.



An array of records

Hash Collisions

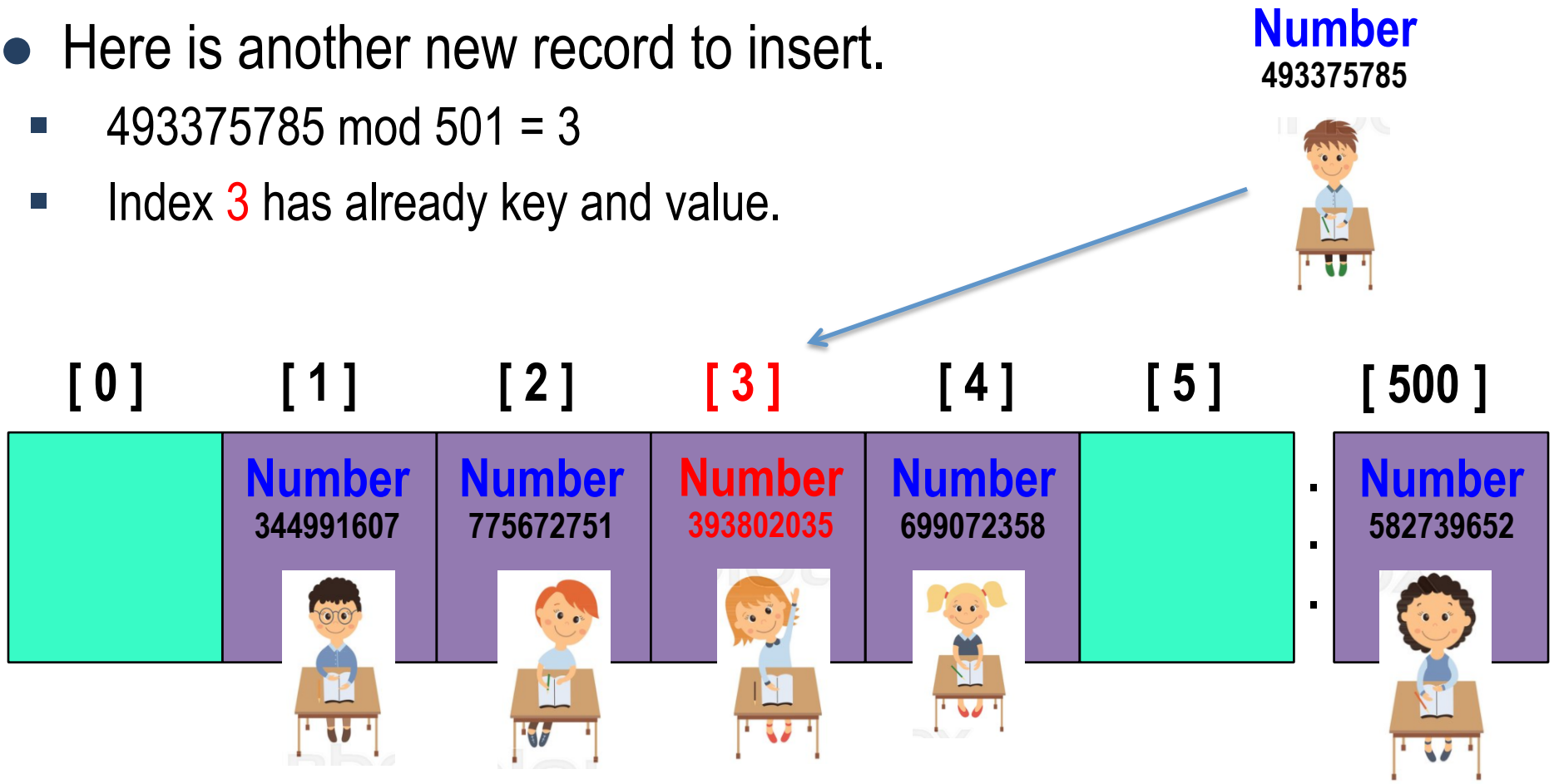
- Here is another new record to insert.
  - 493375785 mod 501 = 3



An array of records

Hash Collisions

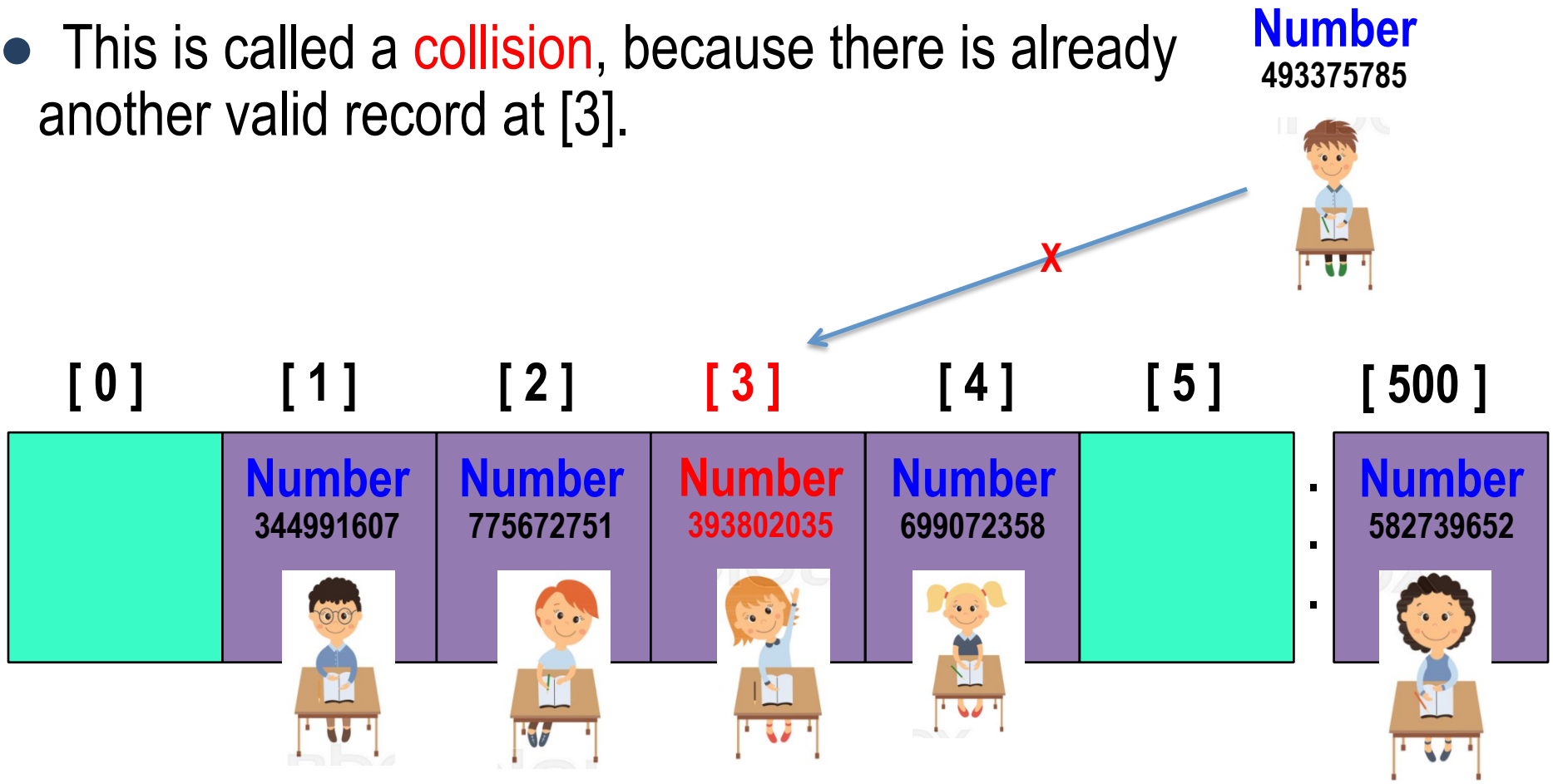
- Here is another new record to insert.
  - 493375785 mod 501 = 3
  - Index 3 has already key and value.



An array of records

Hash Collisions

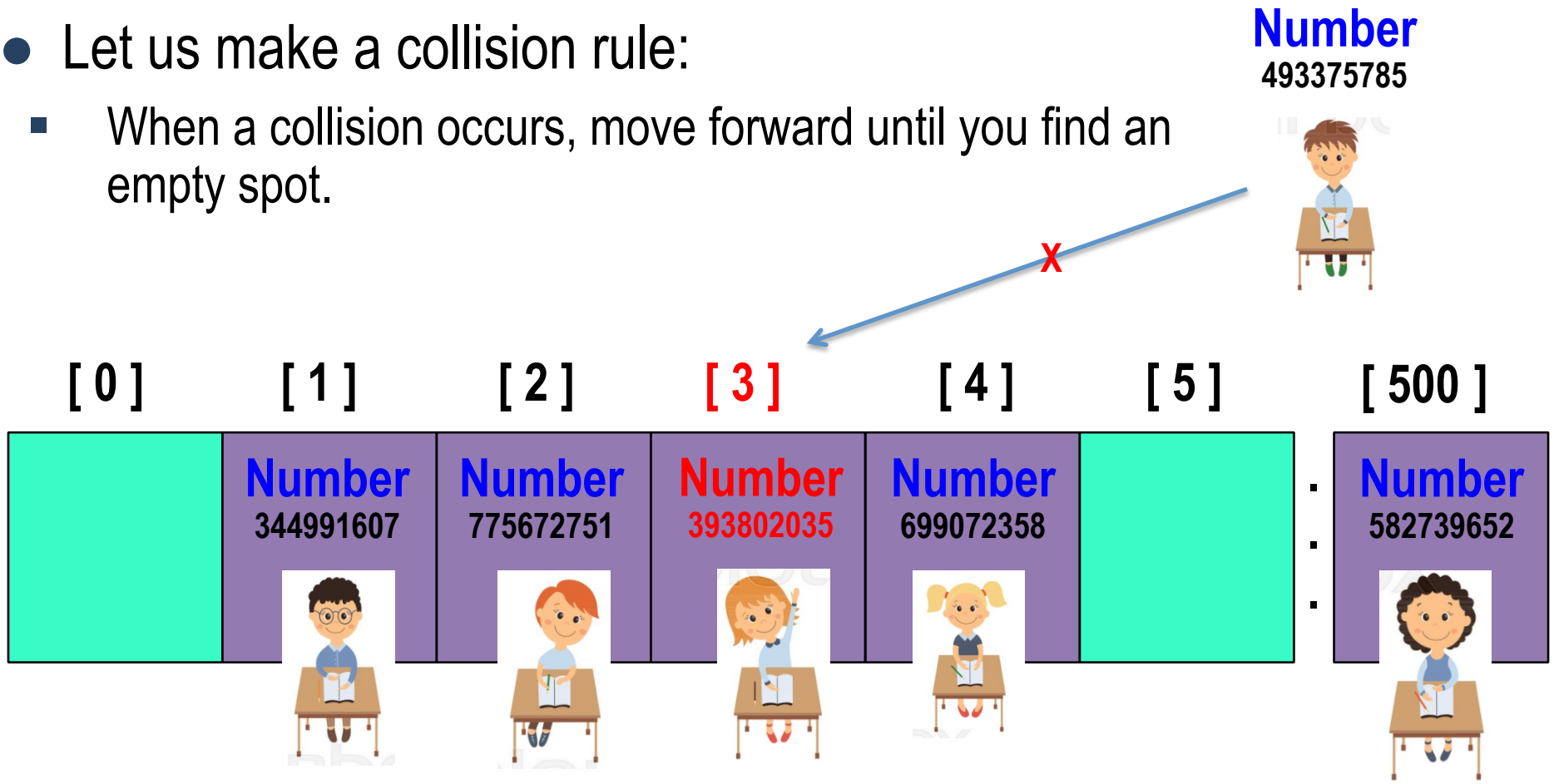
- This is called a collision, because there is already another valid record at [3].



An array of records

Hash Collisions

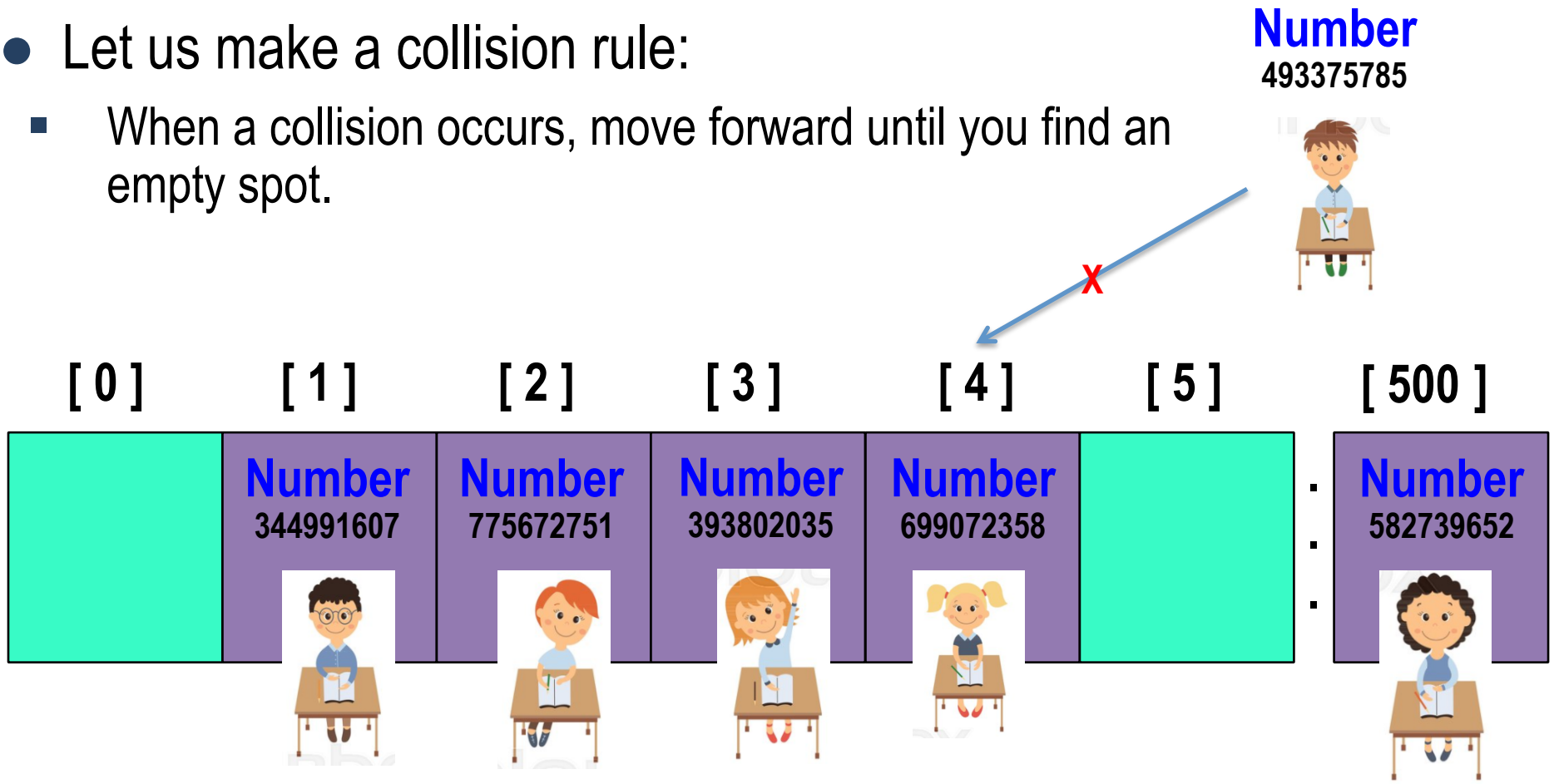
- Let us make a collision rule:
  - When a collision occurs, move forward until you find an empty spot.



An array of records

Hash Collisions

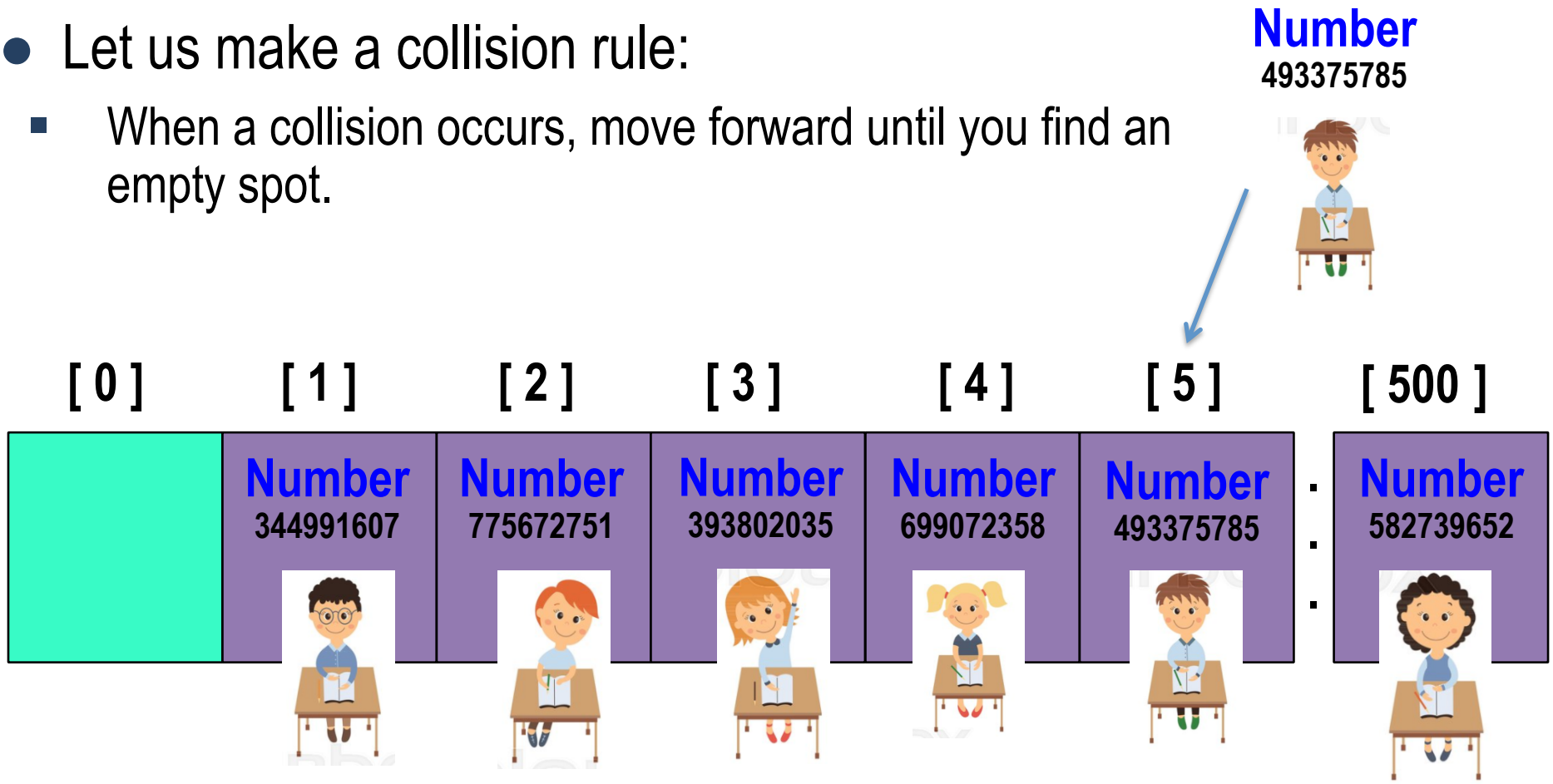
- Let us make a collision rule:
  - When a collision occurs, move forward until you find an empty spot.



An array of records

Hash Collisions

- Let us make a collision rule:
  - When a collision occurs, move forward until you find an empty spot.

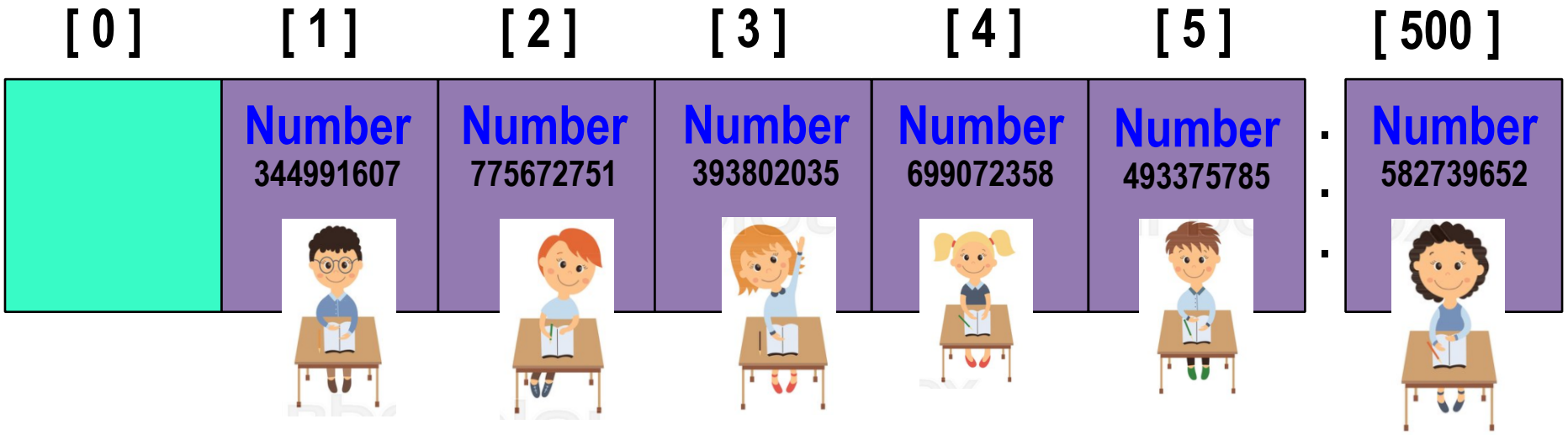


An array of records

Searching for a Key

- The data (or value) that is attached to a key can be found fairly quickly.

Number
493375785

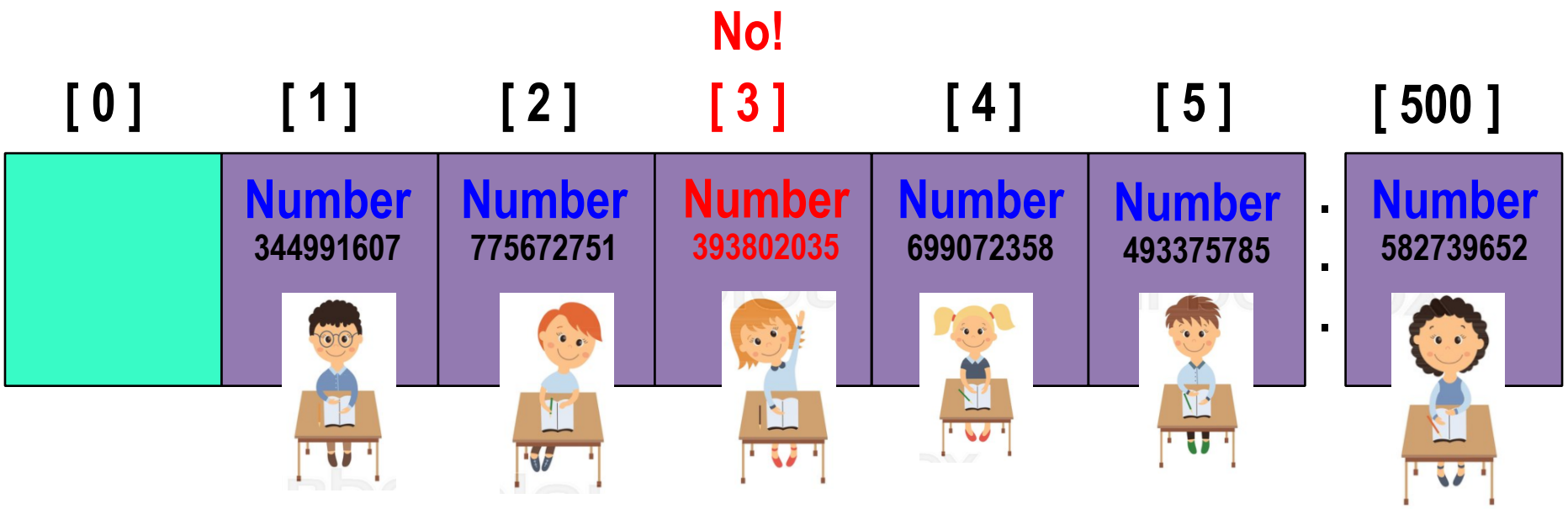


An array of records

Searching for a Key

- The data (or value) that is attached to a key can be found fairly quickly.
  - 493375785 mod 501 = 3

Number
493375785

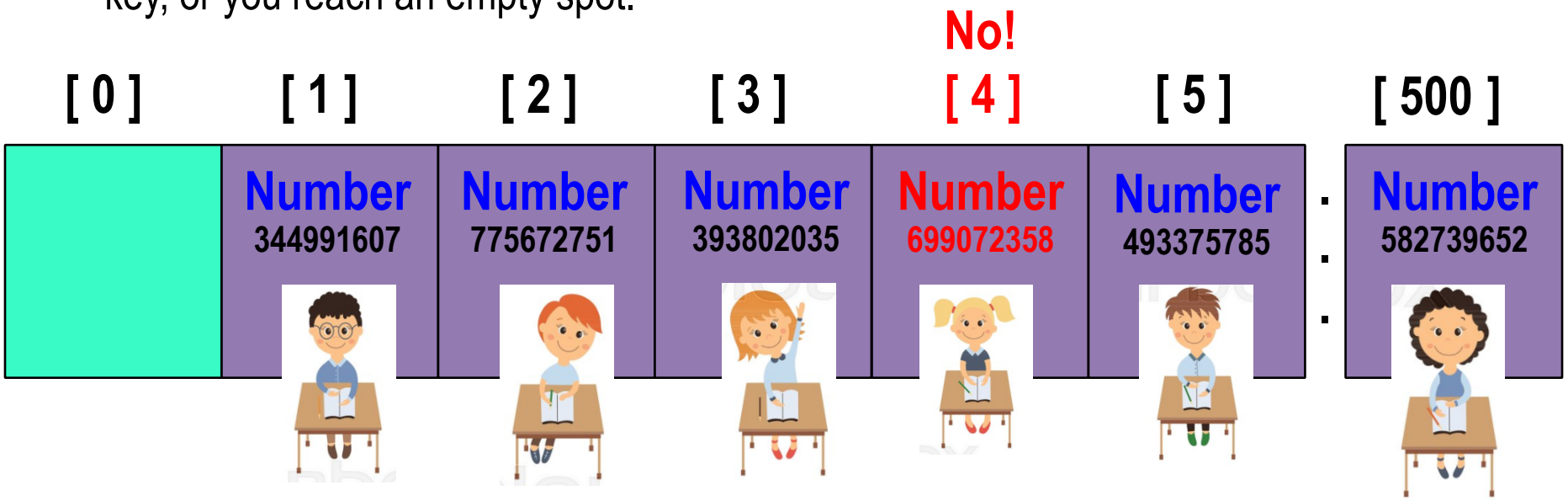


An array of records

Searching for a Key

- The data (or value) that is attached to a key can be found fairly quickly.
  - 493375785 mod 501 = 3
  - Follow the collision rule (keep moving forward) until you find the key, or you reach an empty spot.

Number
493375785

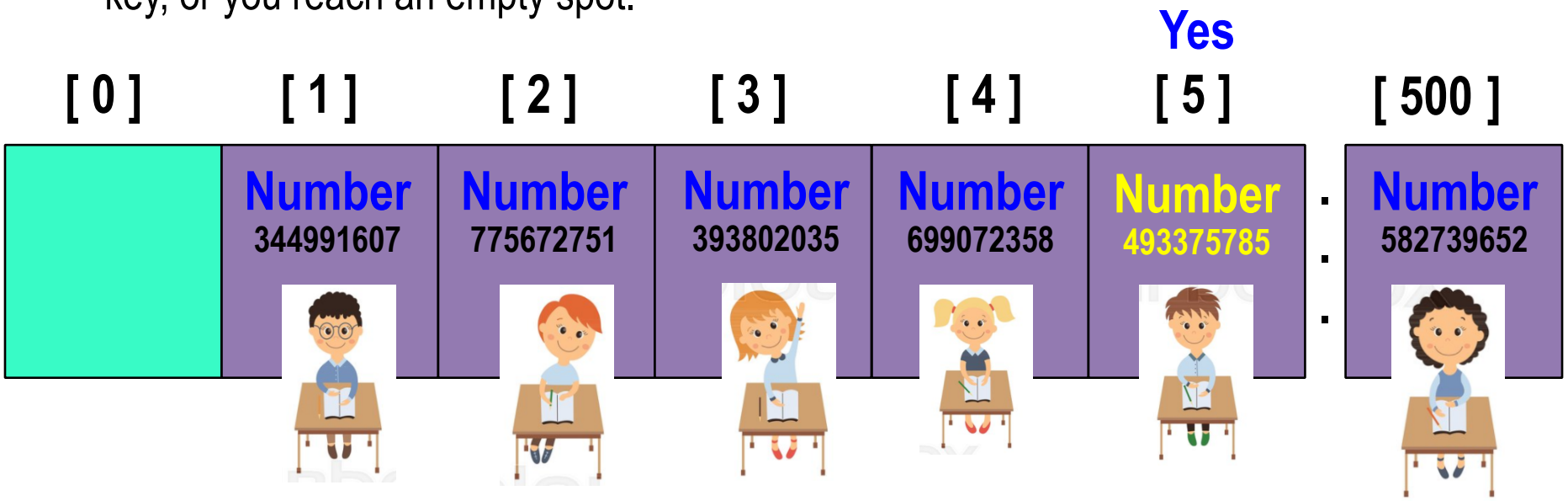


An array of records

Searching for a Key

- The data (or value) that is attached to a key can be found fairly quickly.
  - 493375785 mod 501 = 3
  - Follow the collision rule (keep moving forward) until you find the key, or you reach an empty spot.

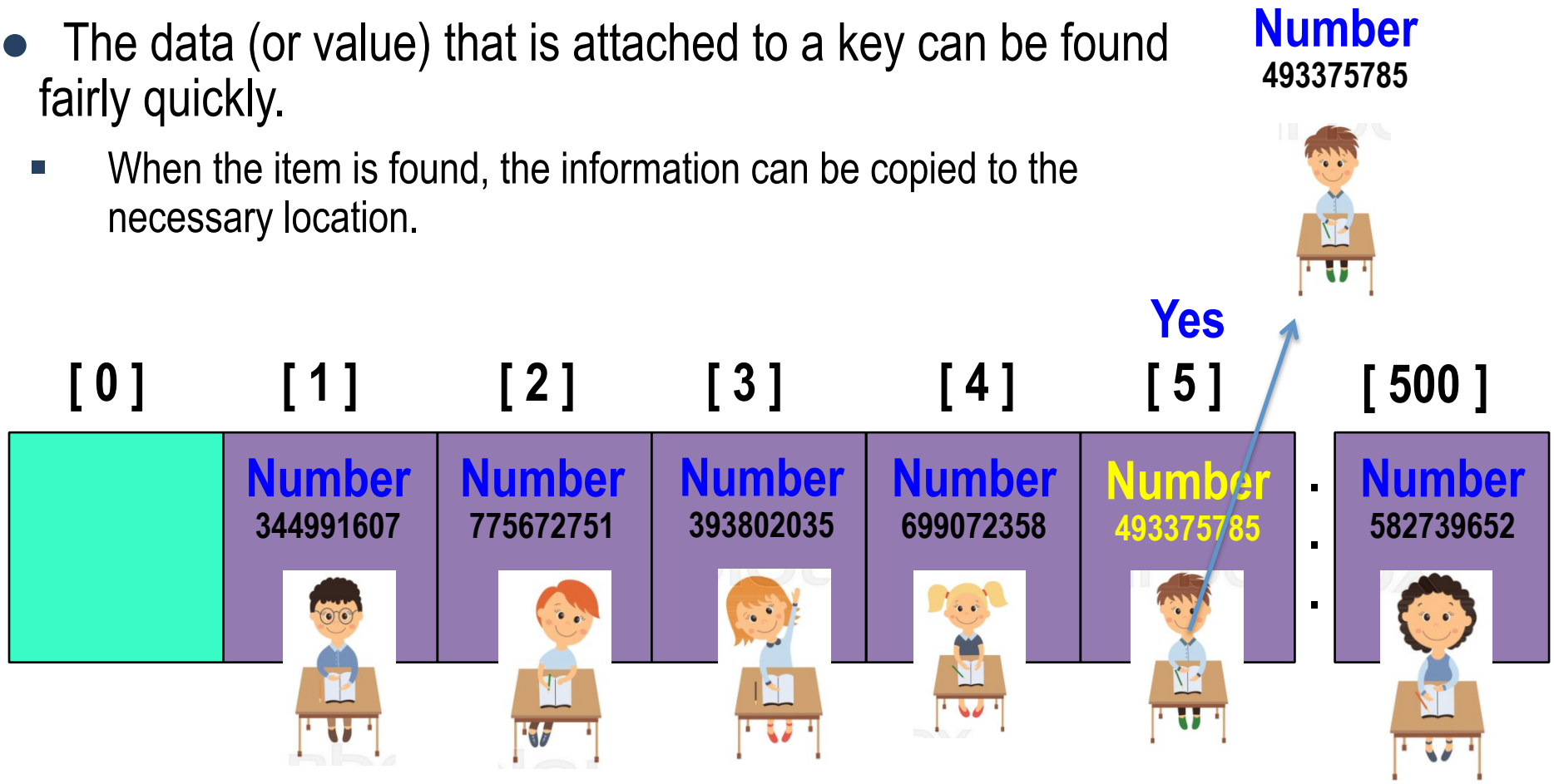
Number
493375785



An array of records

Searching for a Key

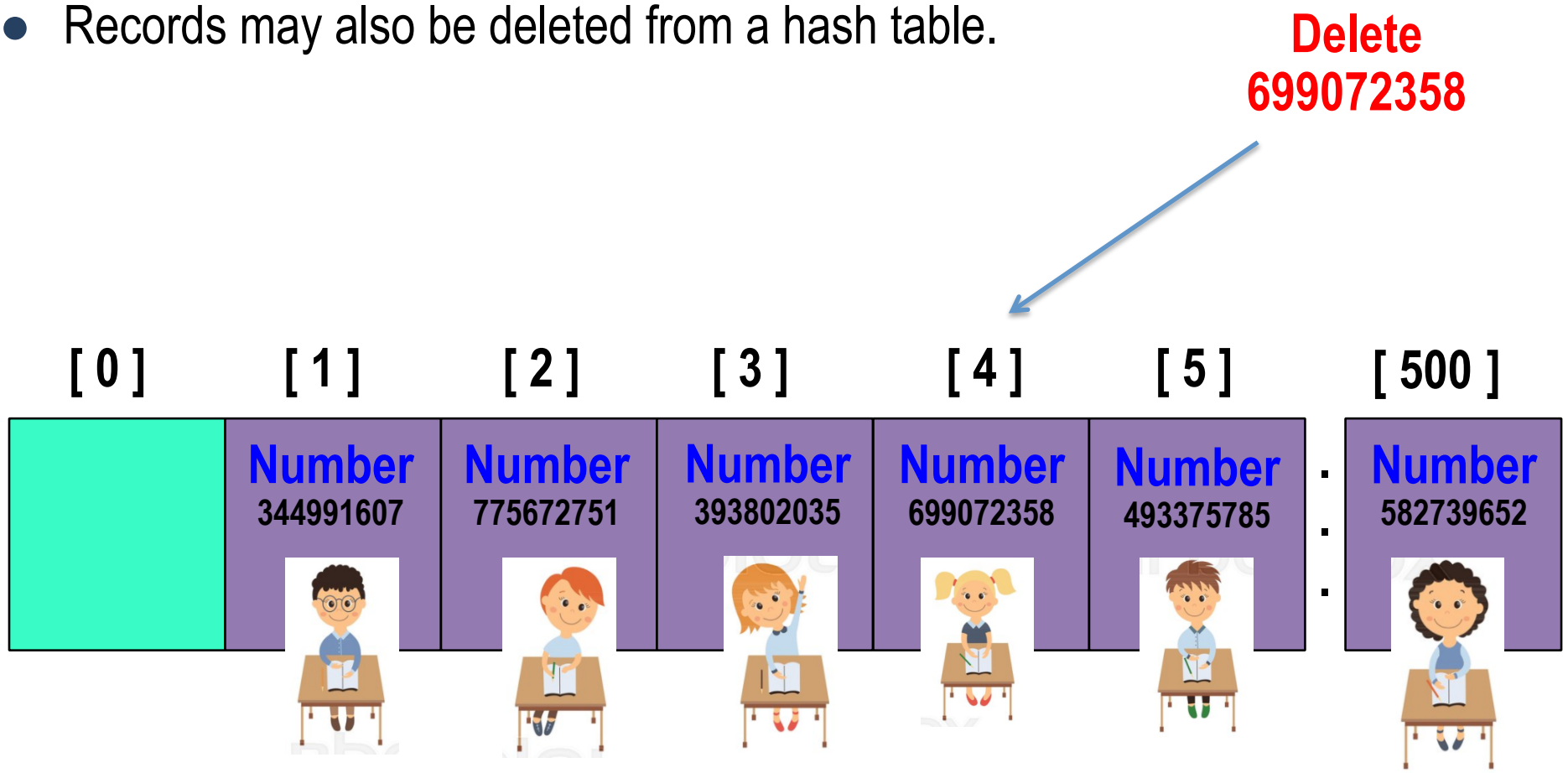
- The data (or value) that is attached to a key can be found fairly quickly.
  - When the item is found, the information can be copied to the necessary location.



An array of records

Deleting a Record

- Records may also be deleted from a hash table.

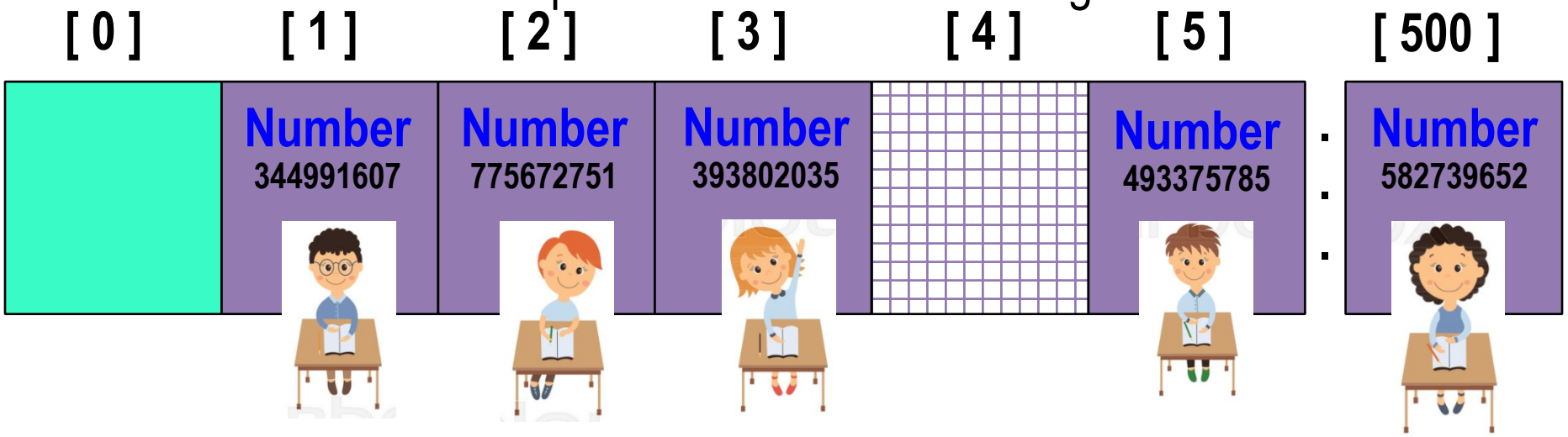


An array of records

Deleting a Record

- Records may also be deleted from a hash table.
- But the location must not be left as an ordinary "empty spot" since that could interfere with searches.
- The location must be marked in some special way so that a search can tell that the spot used to have something in it.

Delete 699072358



An array of records

Map Methods

- Delegate operations to a list-based map at each cell:

Algorithm search(k):
Output: The value associated with the key k in the map, or **null** if there is no entry with key equal to k in the map
return $A[h(k)]$.search(k) {delegate the search to the list-based map at $A[h(k)]$}

Algorithm insert(k,v):
Output: If there is an existing entry in our map with key equal to k, then we return its value (replacing it with v); otherwise, we return **null**
$t = A[h(k)]$.insert(k,v) {delegate the put to the list-based map at $A[h(k)]$}
if t = **null then** {k is a new key}
    $n = n + 1$
return t

Algorithm remove(k):
Output: The (removed) value associated with key k in the map, or **null** if there is no entry with key equal to k in the map
$t = A[h(k)]$.remove(k) {delegate the remove to the list-based map at $A[h(k)]$}
if $t \neq$ **null then** {k was found}
    $n = n - 1$
return t

Interview Question

- Implement Hash Map
  - Hash function just returns the remainder when the key is divided by the hash table size.
  - Hash entry (node) has key and value structure.
  - In addition, the class contains search(key) function to access mapped value by key, insert(key,value) function to put key-value pair in table and remove(key) function to remove hash node by key.
  - For collision resolution, separate chaining strategy could be used.

Implement Simple Hash Map: testHashMap.cpp

```cpp
#include<iostream>

using namespace std;

const int TABLE_SIZE = 128;

// HashEntry Class Declaration
class HashEntry
{
    public:
        int key;
        int value;
        HashEntry(int key, int value)
        {
            this->key = key;
            this->value = value;
        }
};

//HashMap Class Declaration
class HashMap
{
    private:
        HashEntry **table;
    public:
        HashMap()
        {
            table = new HashEntry * [TABLE_SIZE];
            for (int i = 0; i< TABLE_SIZE; i++)
            {
                table[i] = NULL;
            }
        }

        // Hash Function
        int HashFunc(int key)
        {
            return key % TABLE_SIZE;
        }

        void Insert(int key, int value)
        {
            int hash = HashFunc(key);
            while (table[hash] != NULL && table[hash]->key != key)
            {
                hash = HashFunc(hash + 1);
            }
            if (table[hash] != NULL)
                delete table[hash];
            table[hash] = new HashEntry(key, value);
        }

        // Search Element at a key
        int Search(int key)
        {
            int  hash = HashFunc(key);
            while (table[hash] != NULL && table[hash]->key != key)
            {
                hash = HashFunc(hash + 1);
            }
            if (table[hash] == NULL)
                return -1;
            else
                return table[hash]->value;
        }

        // Remove Element at a key
        void Remove(int key)
        {
            int hash = HashFunc(key);
            while (table[hash] != NULL)
            {
                if (table[hash]->key == key)
                    break;
                hash = HashFunc(hash + 1);
            }
            if (table[hash] == NULL)
            {
                cout<<"No Element found at key "<<key<<endl;
                return;
            }
            else
            {
                delete table[hash];
            }
            cout<<"Element Deleted"<<endl;
        }
```

Implement Simple Hash Map: testHashMap.cpp

```
89          ~HashMap()
90          {
91              for (int i = 0; i < TABLE_SIZE; i++)
92              {
93                  if (table[i] != NULL)
94                      delete table[i];
95                  delete[] table;
96              }
97          }
98  };
99
100 int main()
101 {
102     HashMap hash;
103     int key, value;
104     int choice;
105     while (1)
106     {
107         cout<<"\n---------------------"<<endl;
108         cout<<"Operations on Hash Table"<<endl;
109         cout<<"\n---------------------"<<endl;
110         cout<<"1.Insert element into the table"<<endl;
111         cout<<"2.Search element from the key"<<endl;
112         cout<<"3.Delete element at a key"<<endl;
113         cout<<"4.Exit"<<endl;
114         cout<<"Enter your choice: ";
115         cin>>choice;
117         switch(choice)
118         {
119         case 1:
120             cout<<"Enter element to be inserted: ";
121             cin>>value;
122             cout<<"Enter key at which element to be inserted: ";
123             cin>>key;
124             hash.Insert(key, value);
125             break;
126         case 2:
127             cout<<"Enter key of the element to be searched: ";
128             cin>>key;
129             if (hash.Search(key) == -1)
130             {
131             cout<<"No element found at key "<<key<<endl;
132             continue;
133         }
134         else
135         {
136             cout<<"Element at key "<<key<<" : ";
137             cout<<hash.Search(key)<<endl;
138         }
139             break;
140         case 3:
141             cout<<"Enter key of the element to be deleted: ";
142             cin>>key;
143             hash.Remove(key);
144             break;
145         case 4:
146             exit(1);
147         default:
148            cout<<"\nEnter correct option\n";
149        }
150     }
151     return 0;
152 }
```

Reading C type declarations

- http://unixwiz.net/techtips/reading-cdecl.html

Pointer to the pointer


```
1 #include <iostream>
2
3 using namespace std;
4
5 int main () {
6    int  var;
7    int  *ptr;
8    int  **pptr;
9
10   var = 3000;
11
12   // take the address of var
13   ptr = &var;
14
15   // take the address of ptr using address of operator &
16   pptr = &ptr;
17
18   // take the value using pptr
19   cout << "Value of var :" << var << endl;
20   cout << "Value available at *ptr :" << *ptr << endl;
21   cout << "Value available at **pptr :" << **pptr << endl;
22
23   return 0;
24 }
```

Why do we use double pointers?

277

If you want to have a list of characters (a word), you can use `char *word`

If you want a list of words (a sentence), you can use `char **sentence`

If you want a list of sentences (a monologue), you can use `char ***monologue`

If you want a list of monologues (a biography), you can use `char ****biography`

If you want a list of biographies (a bio-library), you can use `char *****biolibrary`

If you want a list of bio-libraries (a ??lol), you can use `char ******lol`

... ...

yes, I know these might not be the best data structures

https://stackoverflow.com/questions/5580761/why-use-double-pointer-or-why-use-pointers-to-pointers