មហាវិទ្យាល័យវិស្វកម្ម
FACULTY OF ENGINEERING

Data Structure & Algorithm II

Lecture 4
Hash Table

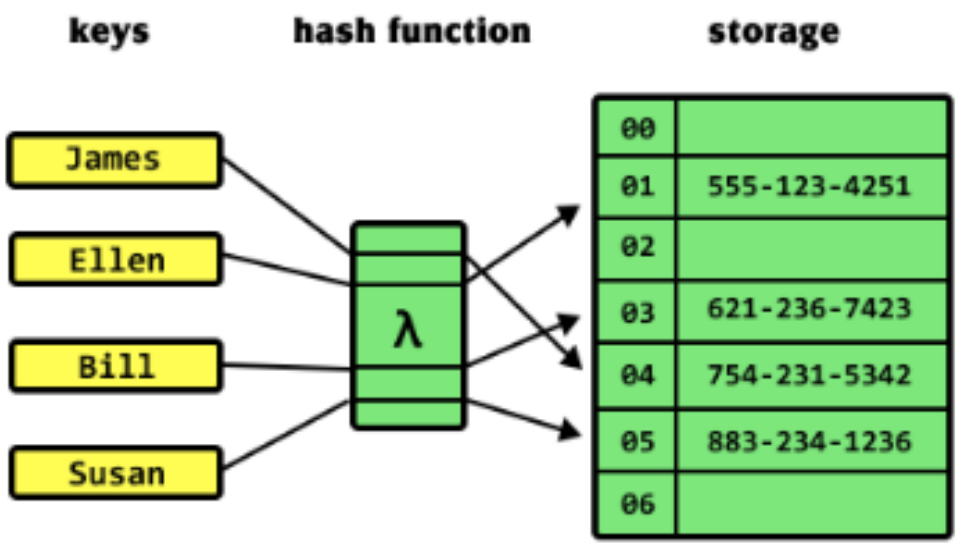
Chhoeum Vantha, Ph.D.
Telecom & Electronic Engineering

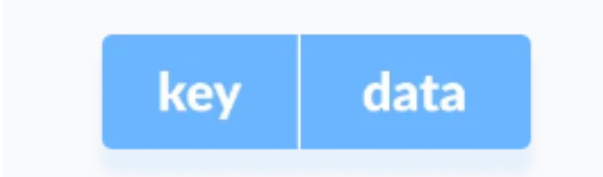
Content

- Hash Tables
- Inserting a New Record
- Collisions
- Searching for a Key
- Deleting a Record

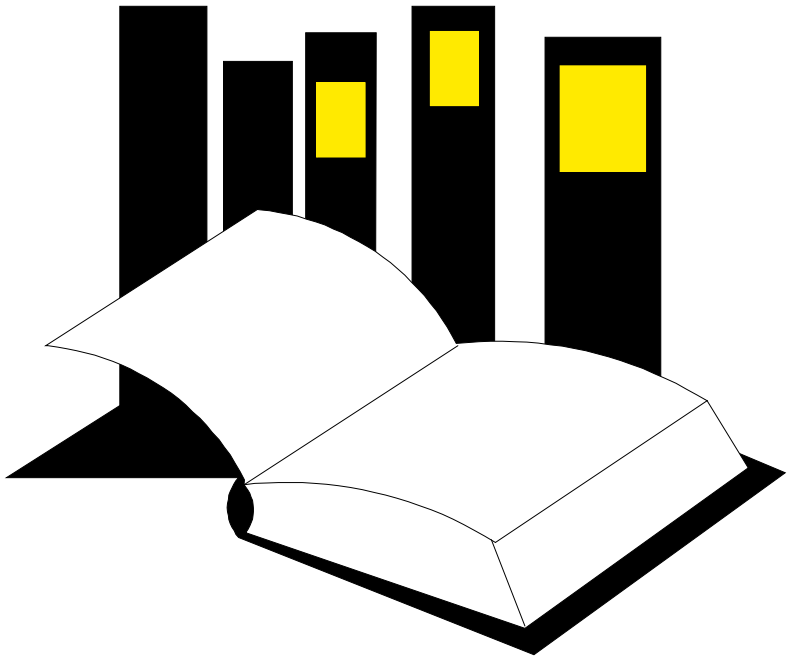
Hash Table

- 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.





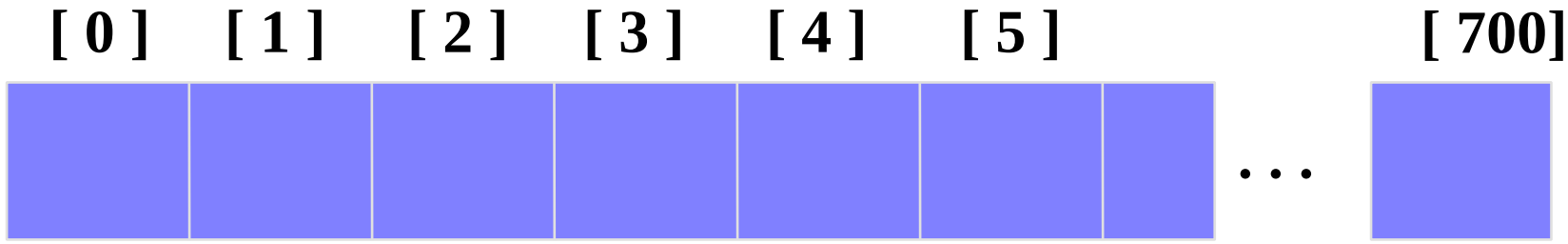
Hash Tables



- There are several ways of storing information in an array, and later searching for the information.
- **Hash tables** are a common approach to the storing/searching problem.

What is a Hash Table ?

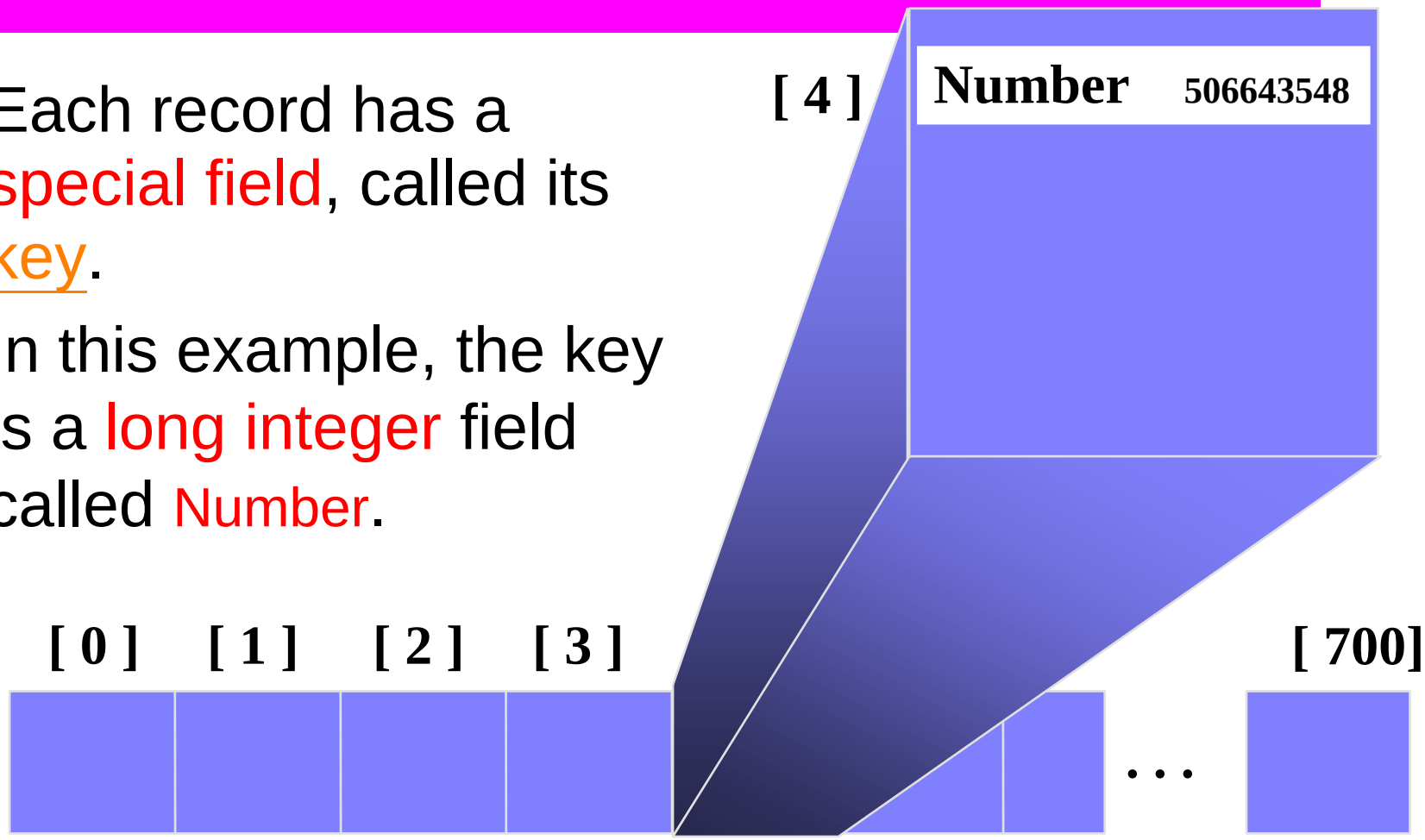
- The simplest kind of hash table is an array of records.
- This example has 701 records.



An array of records

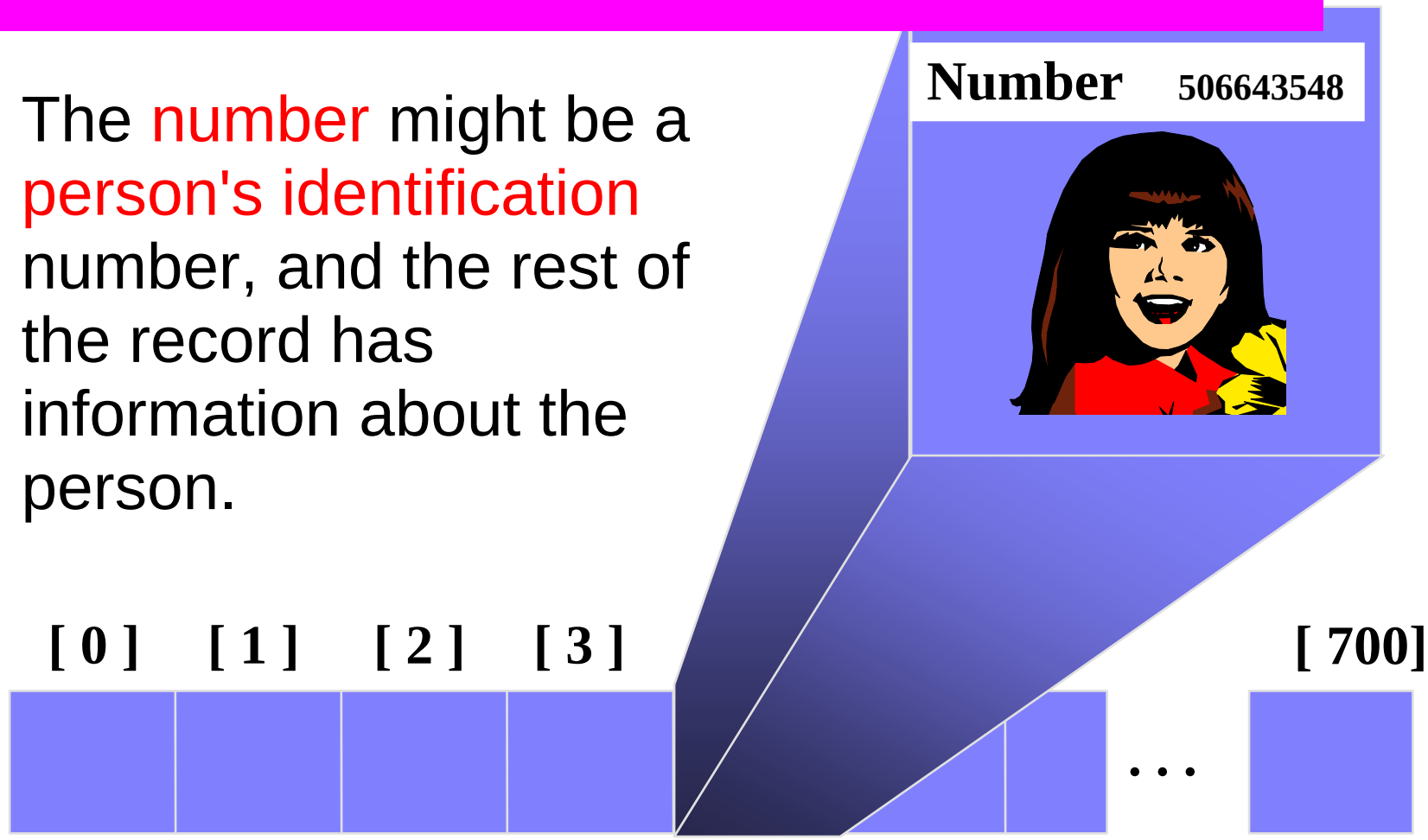
What is a Hash Table ?

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



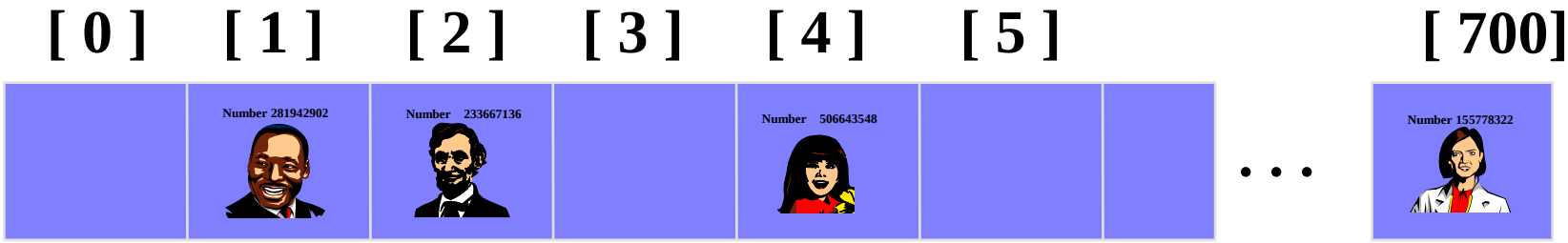
What is a Hash Table ?

- The number might be a person's identification number, and the rest of the record has information about the person.



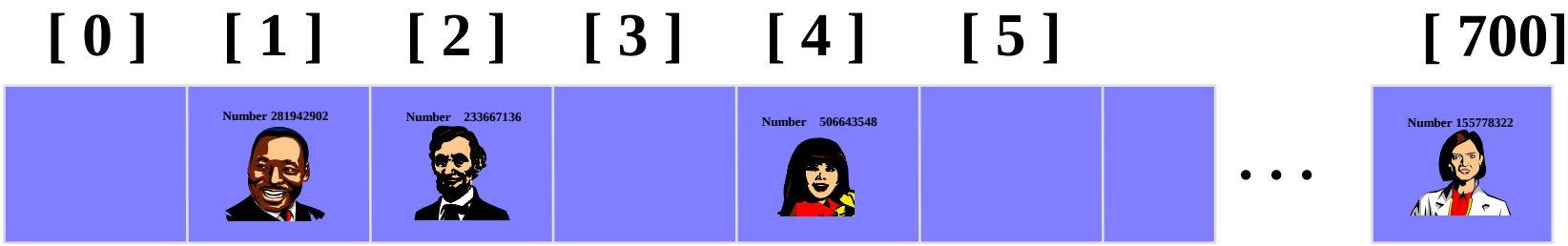
What is a Hash Table ?

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



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.



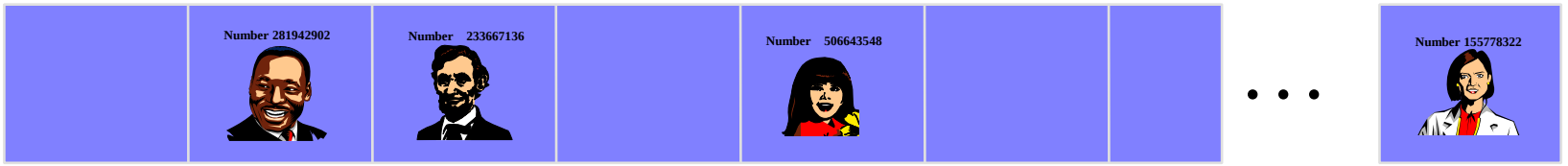
Inserting a New Record

- Typical way create a hash value:

  (Number mod 701)

What is (580625685 mod 701) ?

[0] **[1]** **[2]** **[3]** **[4]** **[5]** **[700]**

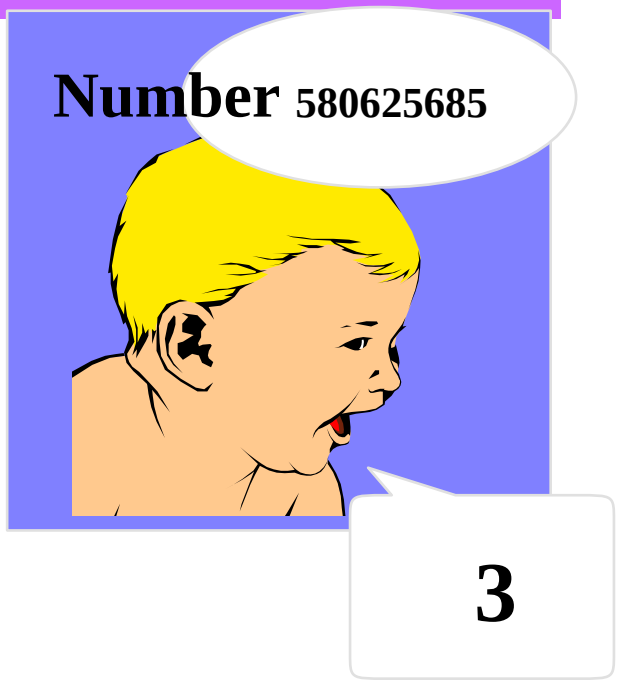


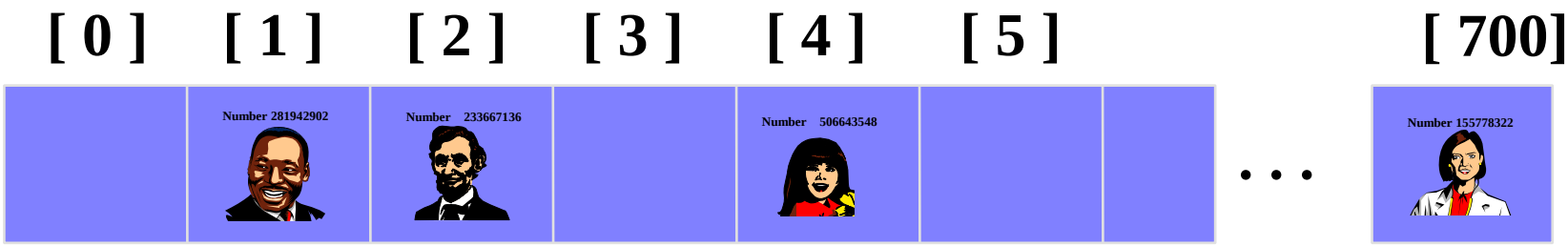
Inserting a New Record

- Typical way to create a hash value:

(Number mod 701)

What is (580625685 mod 701)?





Inserting a New Record

Modulo Calculator

a mod *b* = ?

a = 580625685

b = 701

Clear | Calculate

Answer:

580625685 mod 701 = 3

Proof

Divide *a* by *b* to find the remainder.

580625685 ÷ 701 = 828282 R3

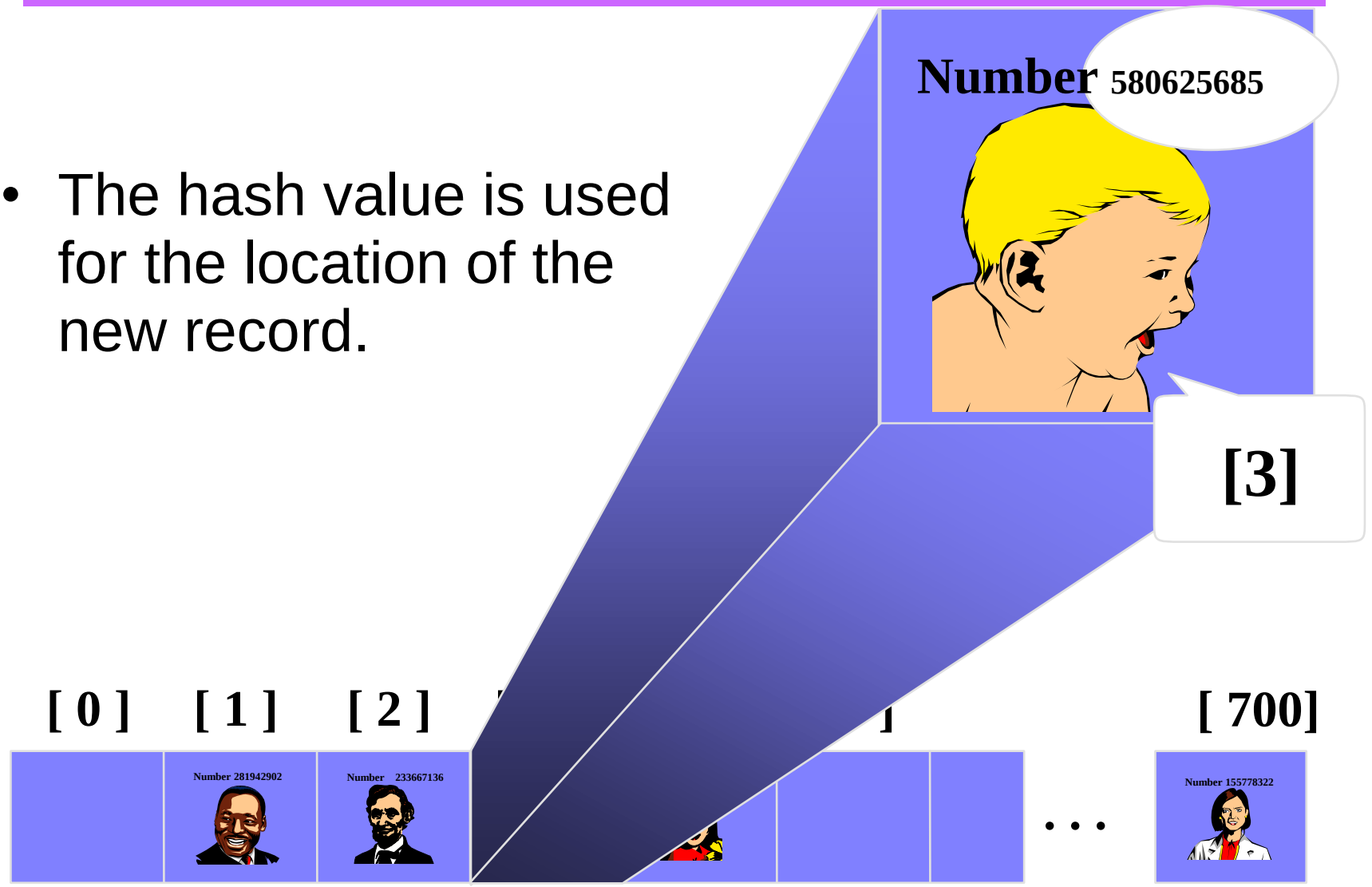
Confirm the answer satisfies the equation:

Quotient × Divisor + Remainder = Dividend

828282 × 701 + 3 = 580625685

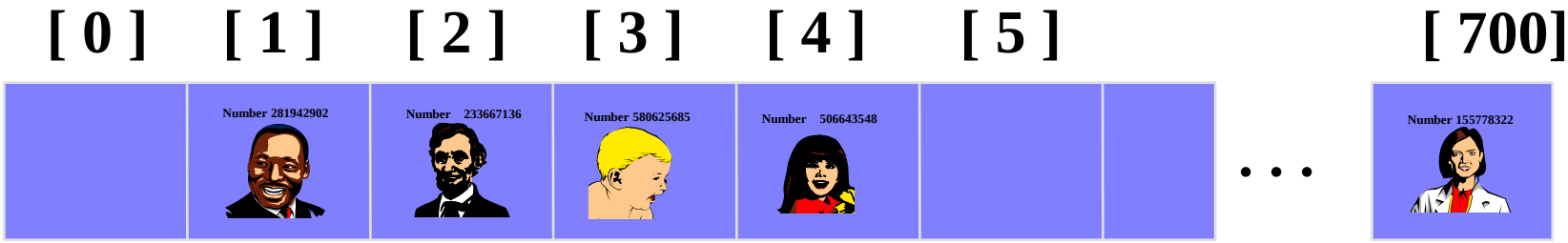
Inserting a New Record

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



Inserting a New Record

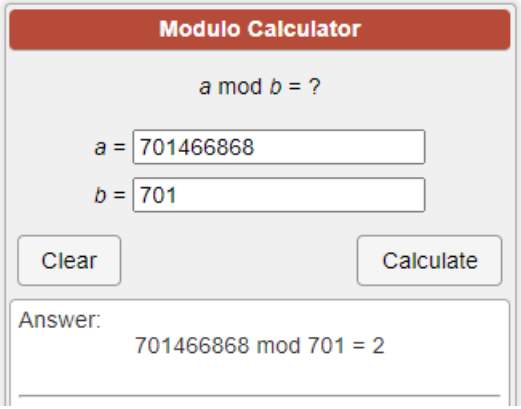
- The hash value is used for the location of the new record.

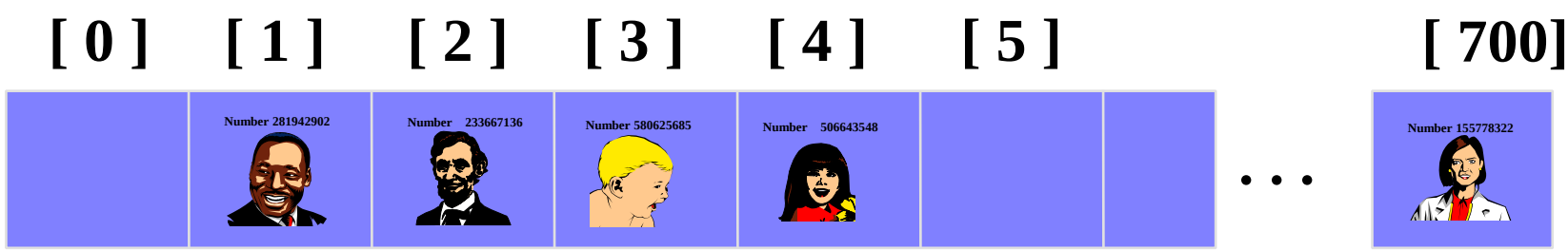


Collisions

- Here is another new record to insert, with a hash value of 2.

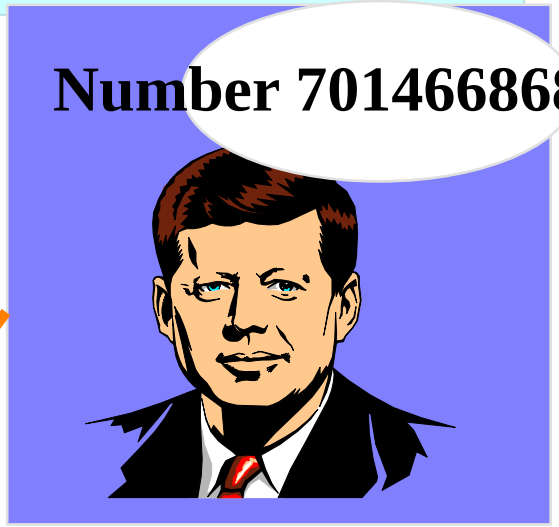
My hash value is [2].



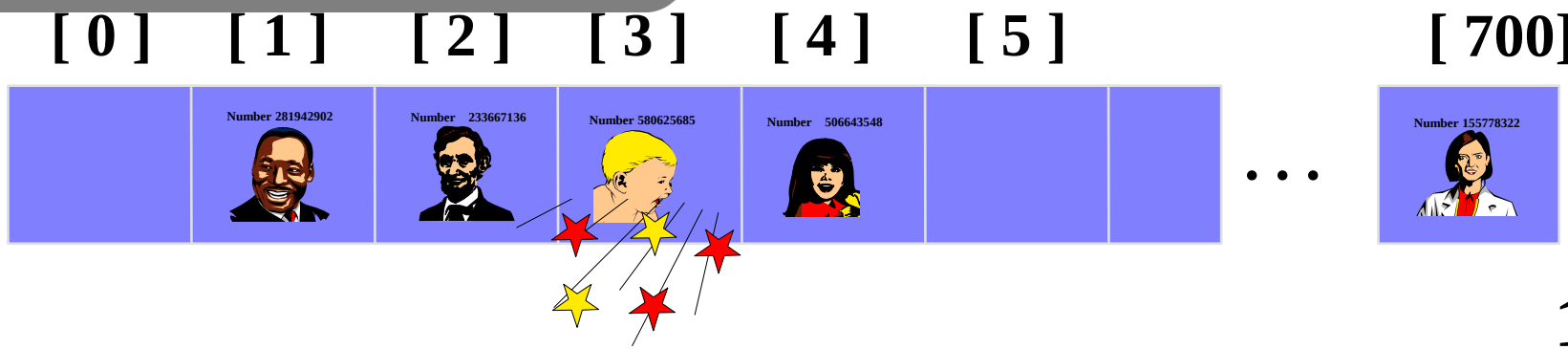


Collisions

- This is called a **collision**, because there is already another valid record at [2].



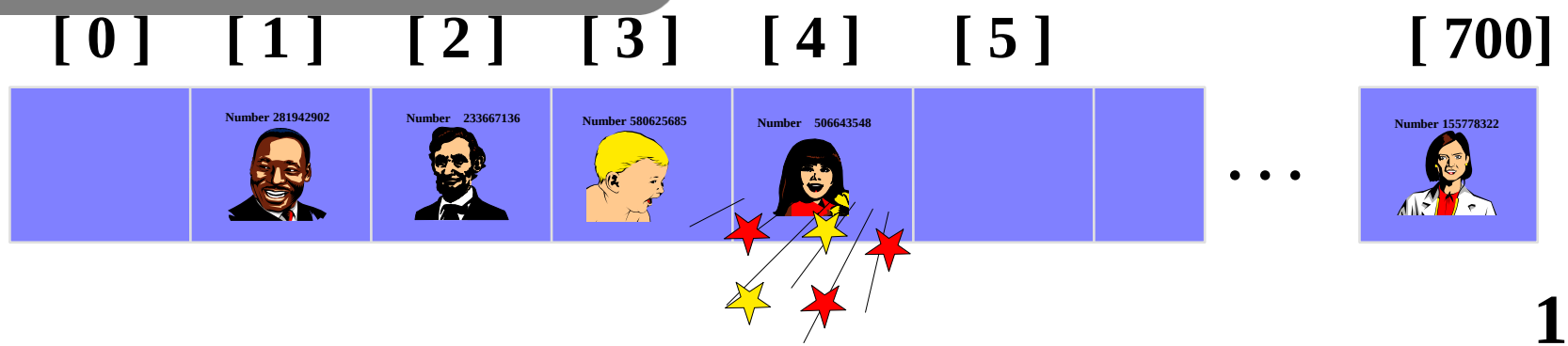
When a collision occurs, move forward until you find an empty spot.



Collisions

- This is called a **collision**, because there is already another valid record at [2].

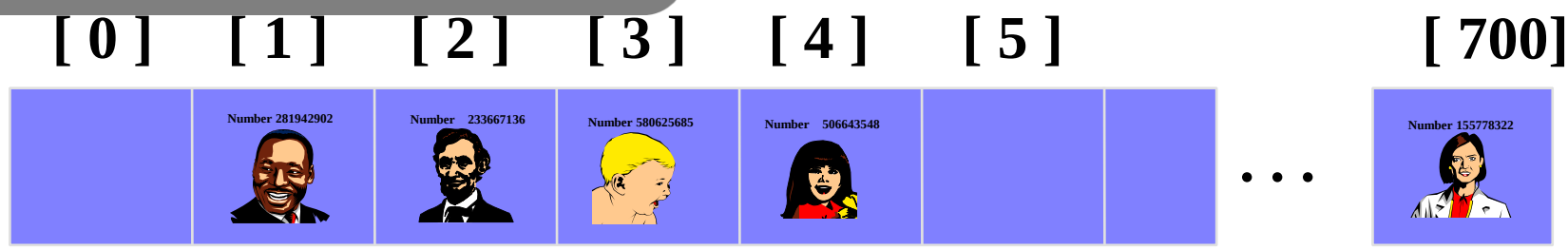
When a collision occurs, move forward until you find an empty spot.



Collisions

- This is called a **collision**, because there is already another valid record at [2].

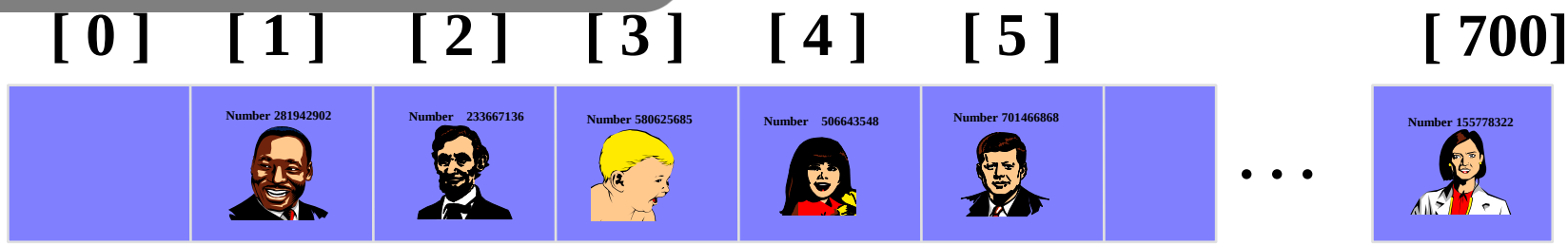
When a collision occurs, move forward until you find an empty spot.



Collisions

- This is called a **collision**, because there is already another valid record at [2].

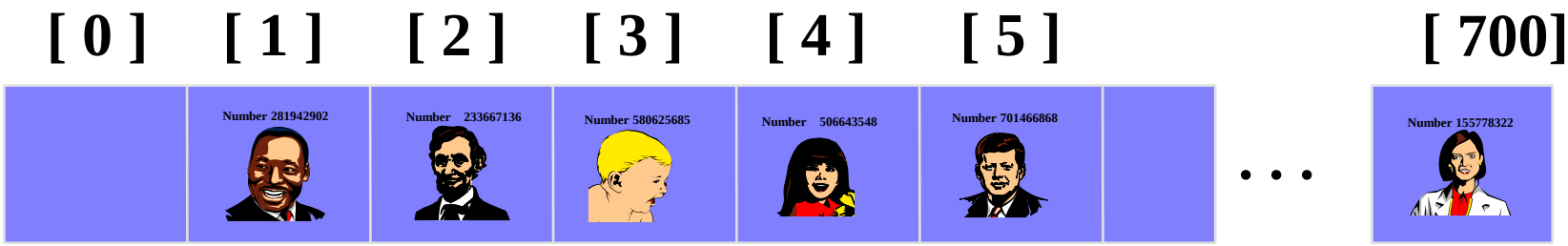
The new record goes in the empty spot.



Searching for a Key

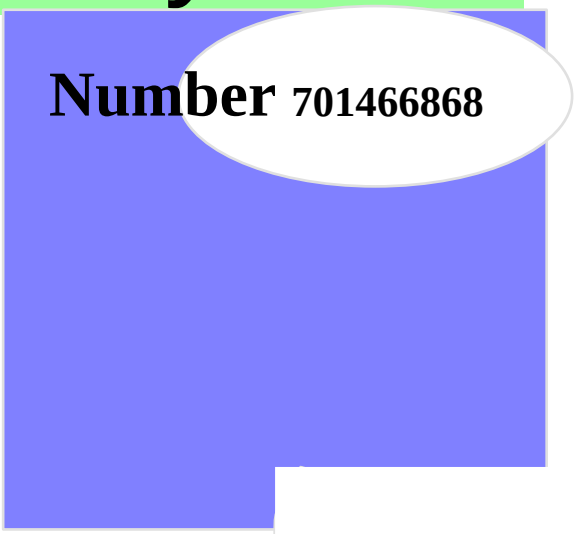
Number 701466868

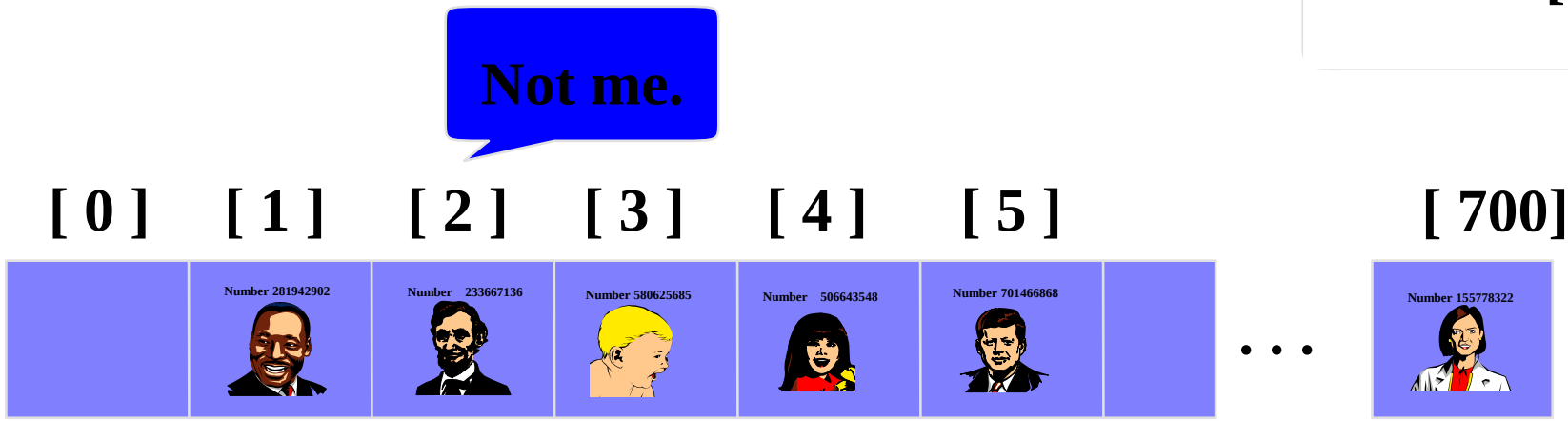
- The data that's attached to a key can be found fairly quickly.



Searching for a Key

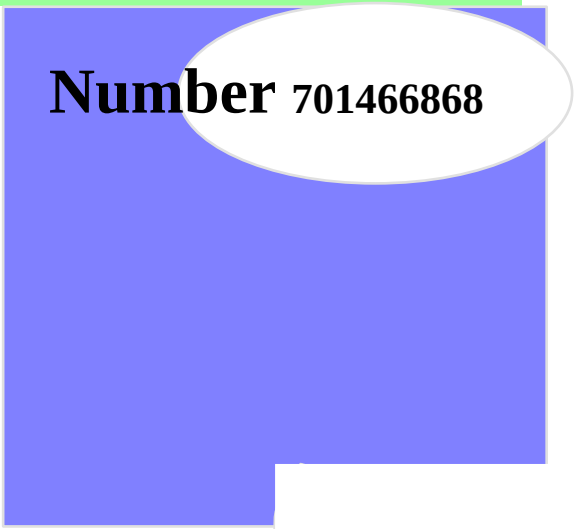
- Calculate the hash value.
- Check that location of the array for the key.





Searching for a Key

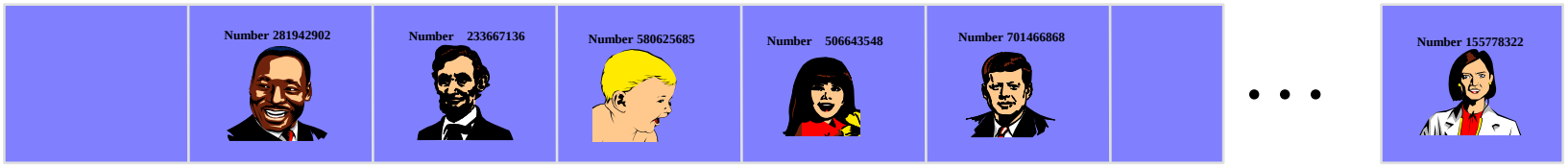
- Keep moving forward until you find the key, or you reach an empty spot.



My hash value is [2].

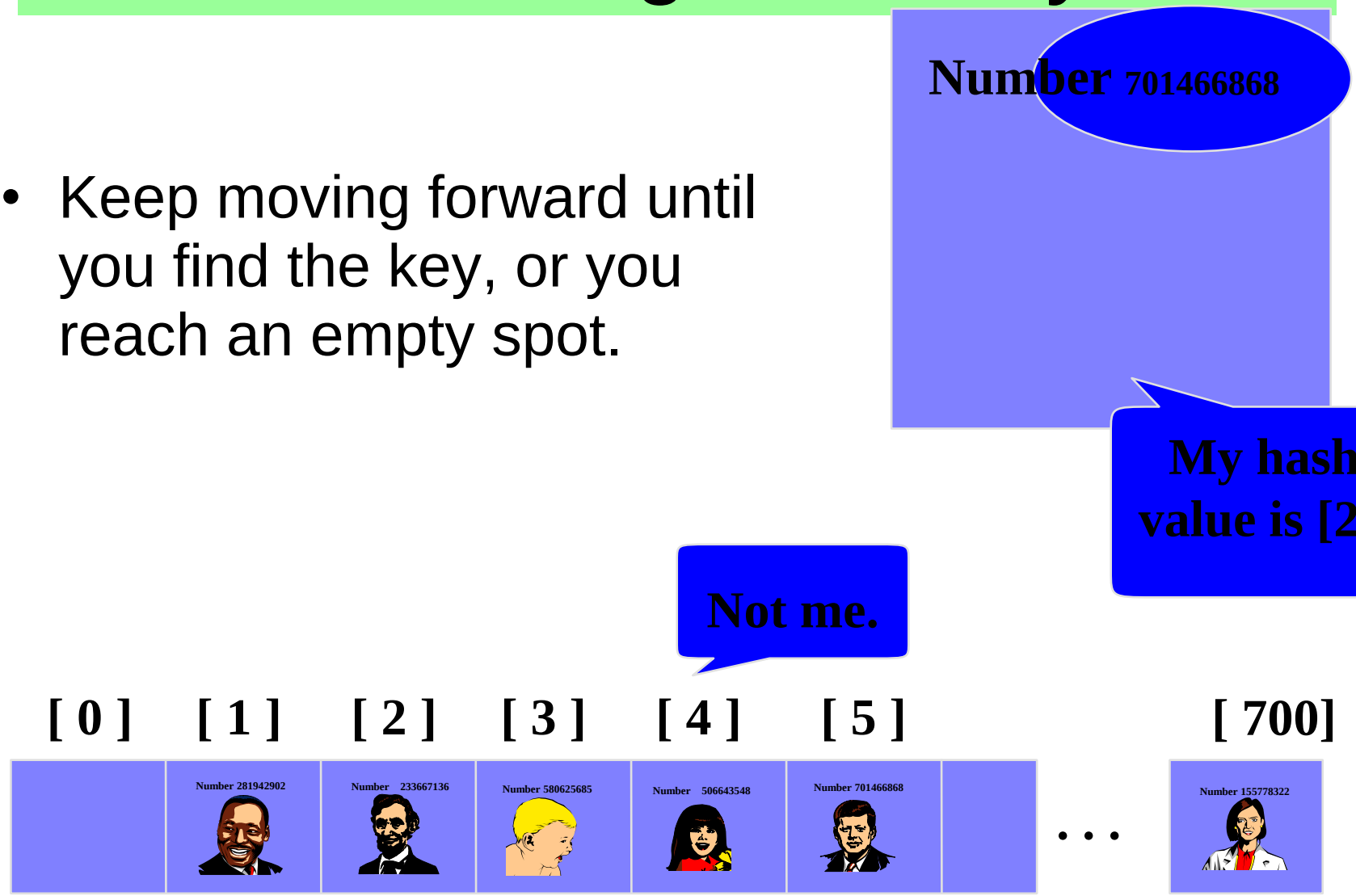
Not me.

[0] [1] [2] [3] [4] [5] [700]



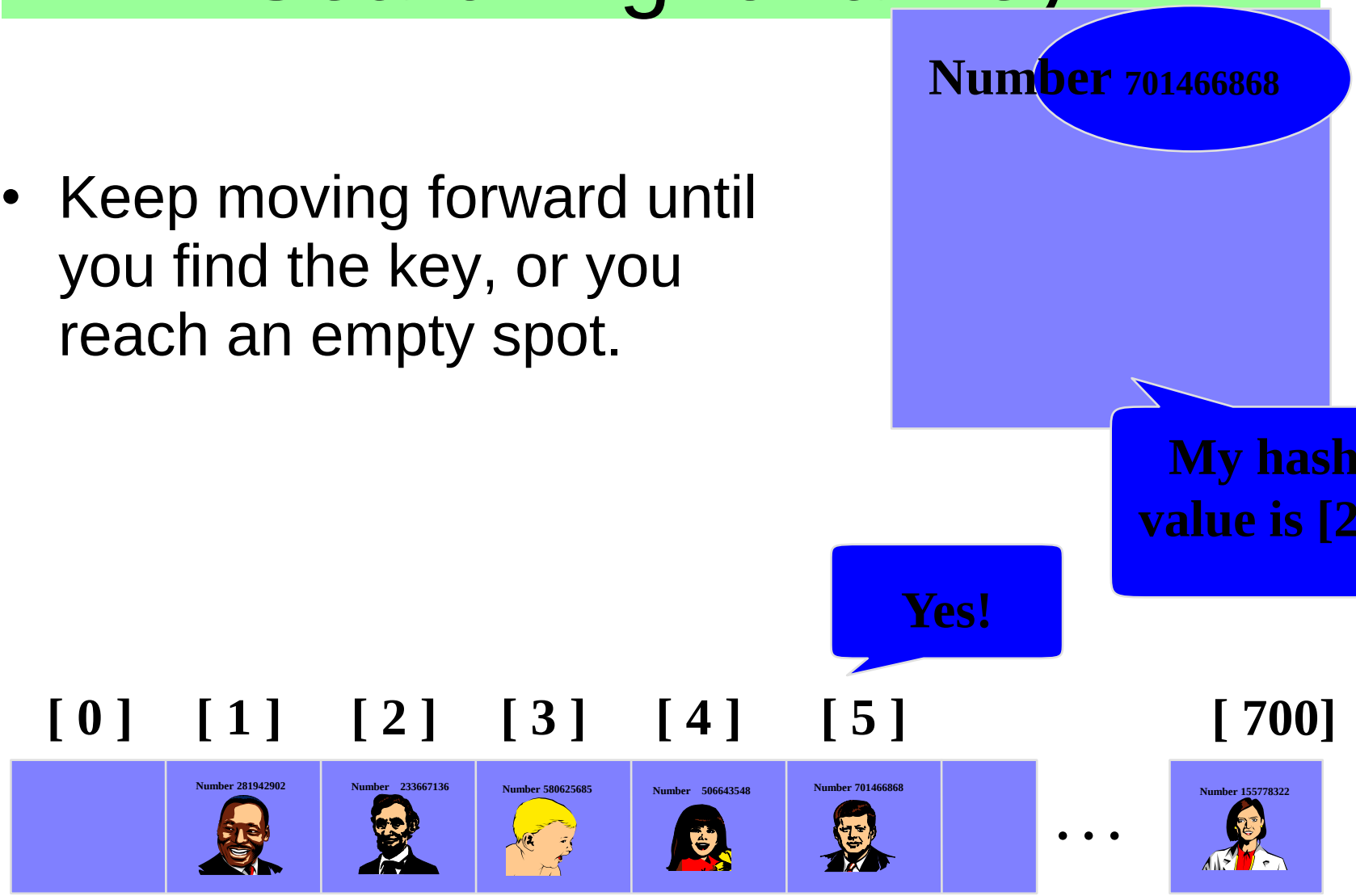
Searching for a Key

- Keep moving forward until you find the key, or you reach an empty spot.



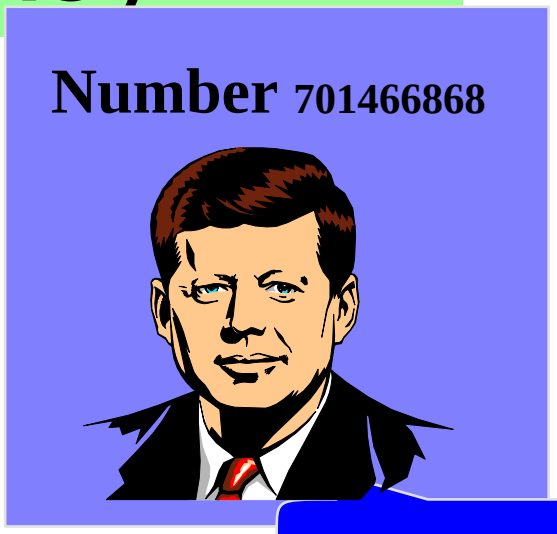
Searching for a Key

- Keep moving forward until you find the key, or you reach an empty spot.



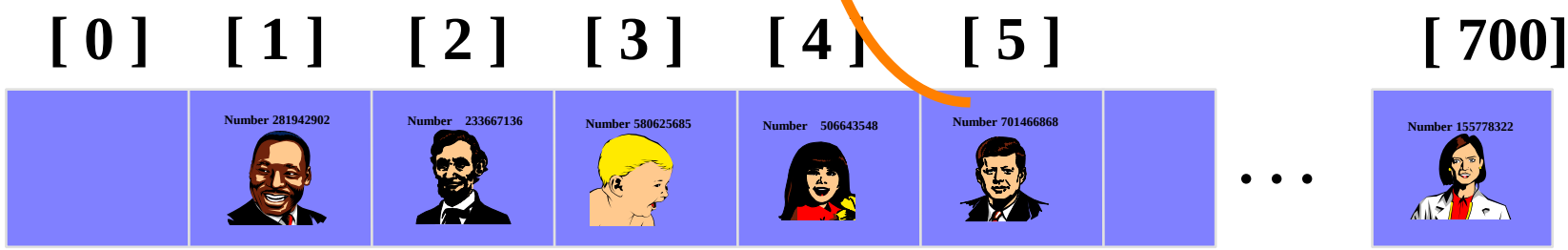
Searching for a Key

- When the item is found, the information can be copied to the necessary location.



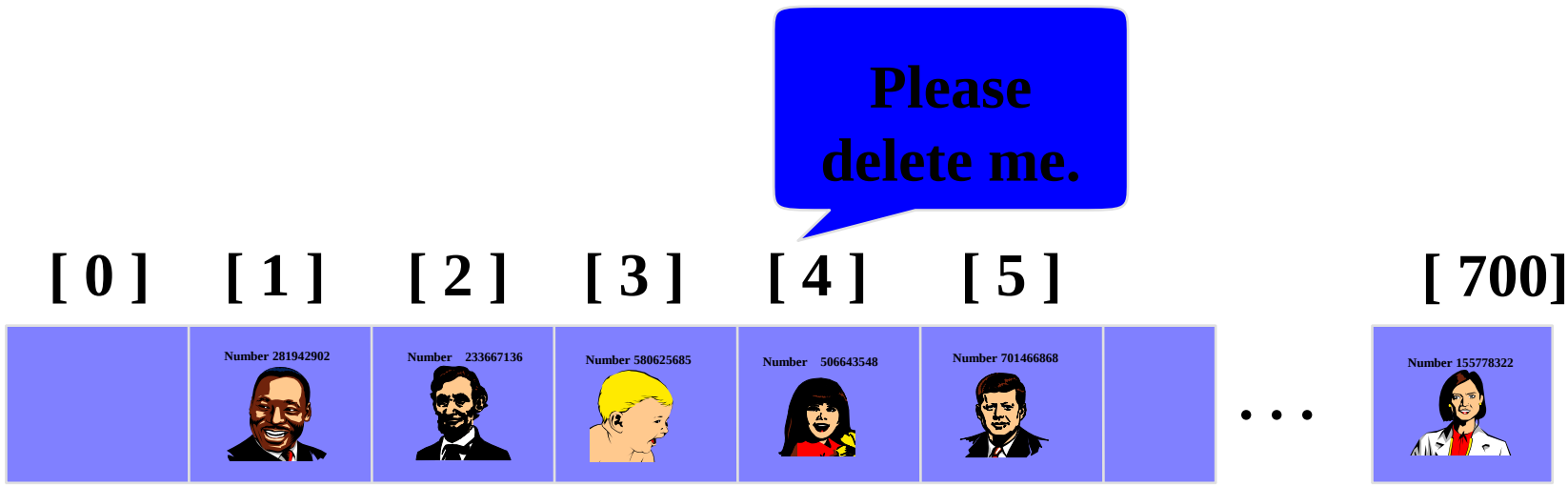
My hash value is [2].

Yes!



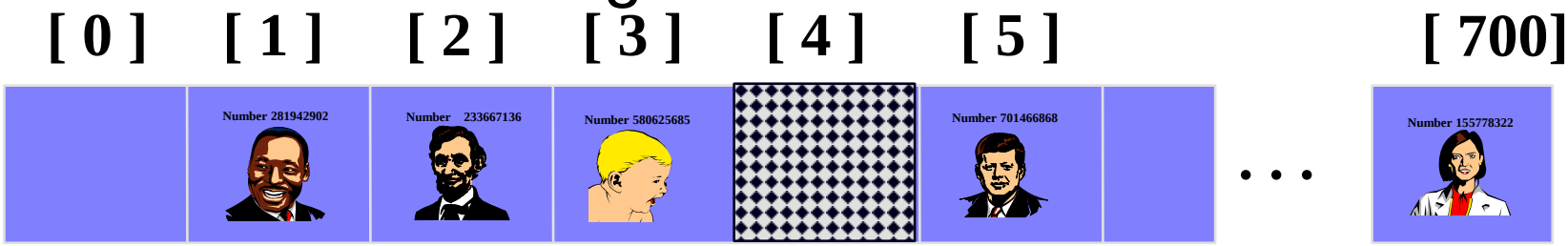
Deleting a Record

- Records may also be deleted from a hash table.



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.



Summary

- Hash tables store a collection of records with keys.
- The location of a record depends on the hash value of the record's key.
- When a collision occurs, the next available location is used.
- Searching for a particular key is generally quick.
- When an item is deleted, the location must be marked in a special way, so that the searches know that the spot used to be used.

W4 – Lab

class Hash

```
1  // CPP program to implement hash table
2  #include<bits/stdc++.h>
3  using namespace std;
4  class Hash
5  {
6      // No. of buckets
7      int BUCKET;
8      // Pointer to an array containing buckets
9      list<int> *table;
```

class Hash

```
10  public:
11      // Constructor
12      Hash(int V);
13      // inserts a key into hash table
14      void insertItem(int x);
15      // deletes a key from hash table
16      void deleteItem(int key);
17      // hash function to map values to key
18      int hashFunction(int x)
19      {
20          return (x % BUCKET);
21      }
22      // function to display the hash table
23      void displayHash();
24  };
```

class Hash

```
25  Hash::Hash(int b)
26  {
27      this->BUCKET = b;
28      table = new list<int>[BUCKET];
29  }
30  void Hash::insertItem(int key)
31  {
32      int index = hashFunction(key);
33      table[index].push_back(key);
34  }
```

class Hash

```
35  void Hash::deleteItem(int key)
36  {
37    // get the hash index of key
38    int index = hashFunction(key);
39    // find the key in (index)th list
40    list <int> :: iterator i;
41    for (i = table[index].begin();
42            i != table[index].end(); i++) {
43      if (*i == key)
44        break;
45    }
46    // if key is found in hash table, remove it
47    if (i != table[index].end())
48      table[index].erase(i);
49  }
```

class Hash

```
50  // function to display hash table
51  void Hash::displayHash() {
52    for (int i = 0; i < BUCKET; i++) {
53      cout << i;
54      for (auto x : table[i])
55        cout << " --> " << x;
56      cout << endl;
57    }
58  }
```

Ex. 1 – Individual work

Execute in Main() with length of hash table with 10

a) Display of the hash table after inserting data

b) Display of the hash table after deleting 12

c) Let inserting data then to look at collision event

(a)

```
0 --> 20
1 --> 11
2 --> 12
3 --> 23
4
5 --> 5
6
7 --> 17
8 --> 8
9 --> 9
```

(b)

```
0 --> 20
1 --> 11
2
3 --> 23
4
5 --> 5
6
7 --> 17
8 --> 8
9 --> 9
```

(c)

```
0 --> 20
1 --> 11
2
3 --> 23 --> 13
4 --> 44
5 --> 5
6 --> 86
7 --> 17 --> 57
8 --> 8
9 --> 9
```

Ex. 2 (Teamwork)

- Create your own class Hash Table to store string data then apply any appropriate operation to store and structure data

Next week homework

1. Explain code Line-by-line
2. What is the **list** mean **(line 9)**? What are the operations in a **list**? Please explain briefly.
3. Add one more function for Searching

Thanks!