Data structures and Algorithms

Hash Tables

CONTENTS

Review of Searching Techniques

- Recall the efficiency of searching techniques covered earlier.
- The sequential search algorithm takes time proportional to the data size, i.e, **O(n)**.
- Binary search improves on liner search reducing the search time to **O(log n)**.
- With a BST, an **O(log n)** search efficiency can be obtained; but the worst-case complexity is **O(n)**.
- The efficiency of these search strategies depends on the number of items in the container being searched.
- Search methods with efficiency independent on data size would be better.

Review of Searching Techniques (Cont'd)

- Suppose that we want to store 10,000 students records (each with a 5-digit ID) in a given container.
  - A linked list implementation would take **O(n)** time.
  - A height balanced tree would give **O(log n)** access time.
  - Using an array of size 100,000 would give **O(1)**access time but will lead to a lot of space wastage.
- Is there some way that we could get **O(1)** access without wasting a lot of space?
- The answer is hashing.

Introduction to Hashing

- **Hashing** is the technique used for performing almost constant time search in case of insertion, deletion and find operation.
- The essence of hashing is to facilitate the next level searching method when compared with the linear or binary search.
- It is the process of mapping large amount of data item to a smaller table with the help of a **hashing function**.

Introduction to Hashing

- The advantage of this searching method is its efficiency to handle vast amount of data items in a given collection (i.e. collection size).

- Due to this hashing process, the result is a **Hash data structure** that can store or retrieve data items in an average time disregard to the collection size.

Hash Tables

- Let us assume a function f and apply this function to K, then it returns 'i' that is f(k)=i. The ith entry of the access table gives the location of record with key value K.
- One type of such access table is known as Hash table.
- The **hash table** is an array which contains key values with pointers to the corresponding records.
- The basic idea here is to place a key inside the hash table, and the location/index of that key will be calculated from the given key value itself.
- The one to one correspondence between a key value and index in the hash table is known as **hashing**.

What is a Hash Table ?

- A **hash table**, or a **hash map**, is a data structure that associates keys (names) with values (attributes)
- The hash table contains key values with pointers to the corresponding records which helps in retrieval of information in an efficient manner.
- The basic idea is to place key value into location in the hash table and this location will be calculated from the key value itself using the **hash function**.

Hash tables

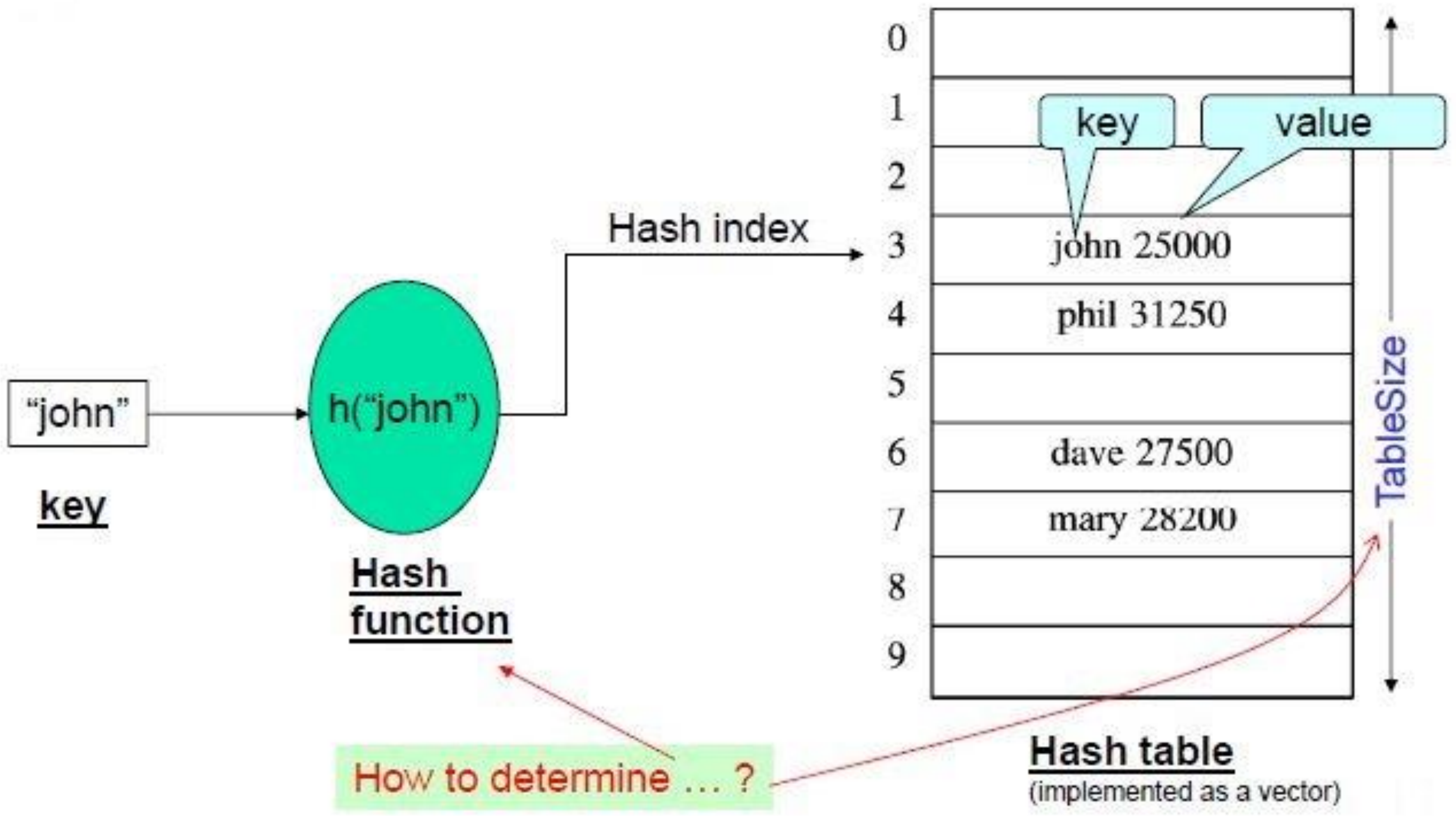
- A hash table amounts to a combination of two things with which we are quite familiar.

  - First, a **hash function,** which returns an nonnegative integer value called a hash code.

  - Second, an **array** capable of storing data of the type we wish to place into the data structure.

- The idea is that we run our data through the hash function, and then store the data in the element of the array represented by the returned hash code

What is a Hash Table ? Main Component



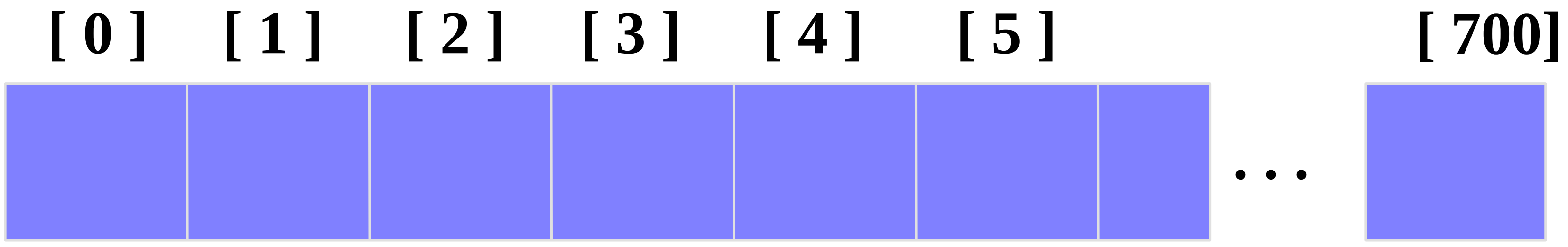
Hash tables

- To get this performance upgrade, we create a new structure whereby when we insert data into the structure, the data itself gives us a clue about where we will find the data, should we need to later look it up.

- The trade off is that hash tables are not great at ordering or sorting data, but if we don't care about that, then we are good to go!

What is a Hash Table ?

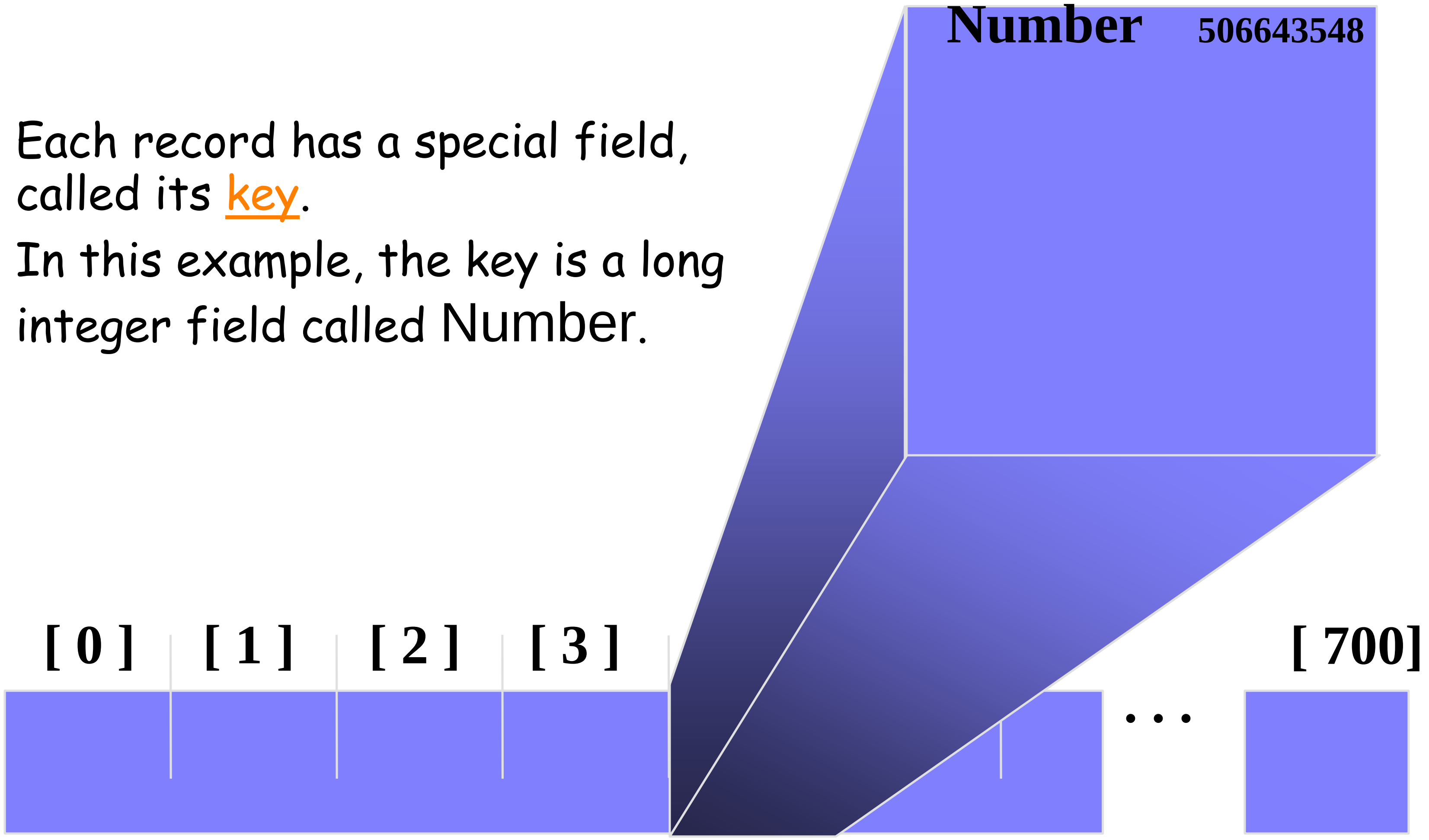
- The simplest kind of hash table is an array of records.
- This example has 701 records.



An array of records

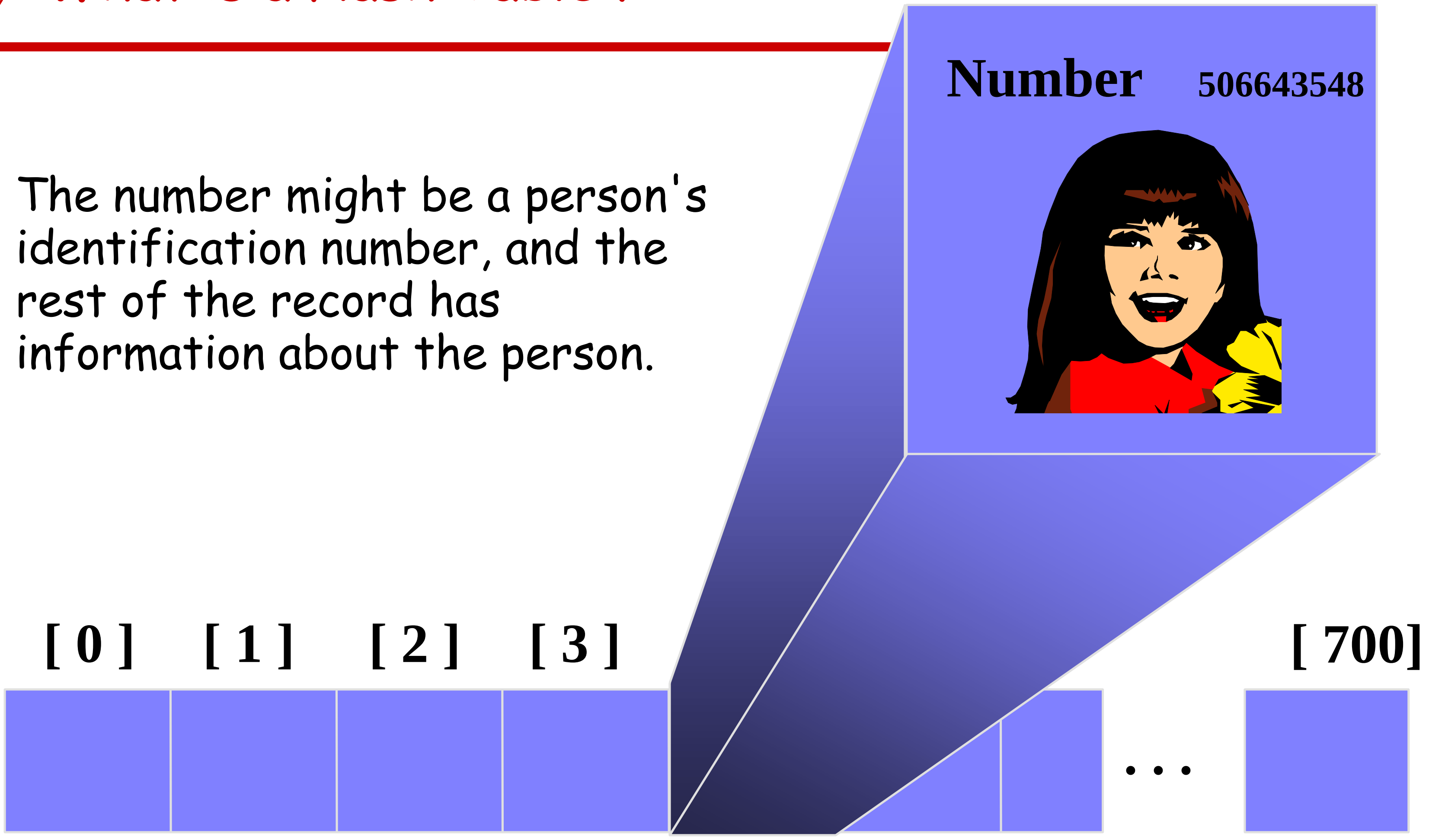
What is a Hash Table ?

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



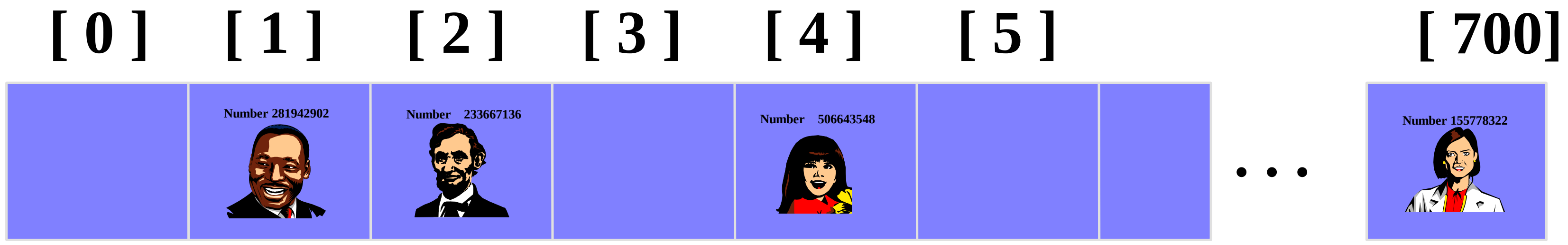
What is a Hash Table ?

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



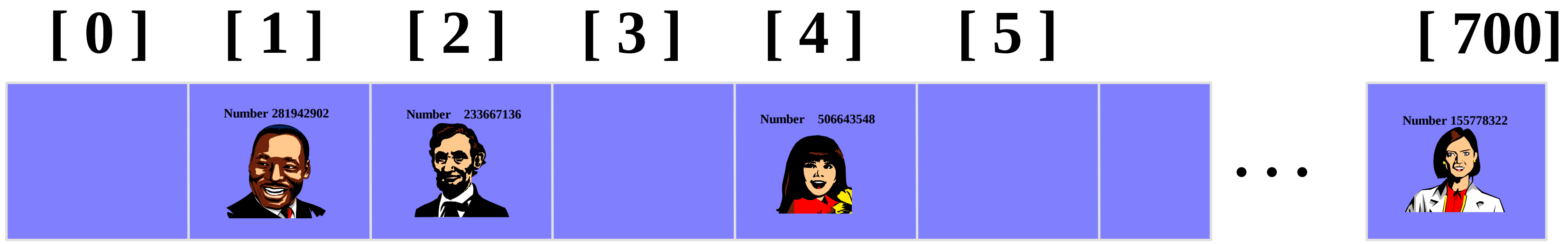
What is a Hash Table ?

- 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.

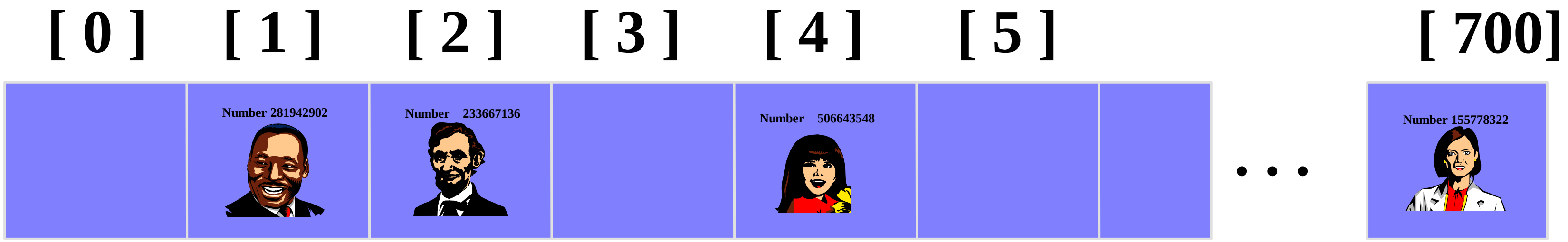


Inserting a New Record

- Typical way create a hash value:

(Number mod 701)

What is (580625685 mod 701) ?



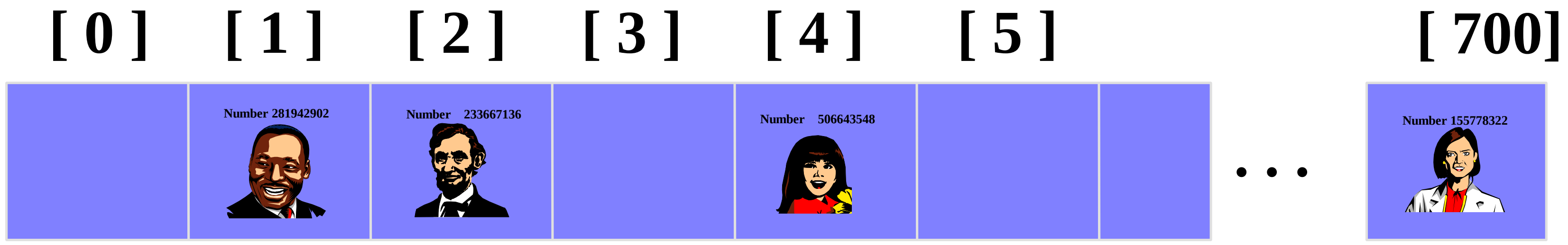
Inserting a New Record

- Typical way to create a hash value:

(Number mod 701)

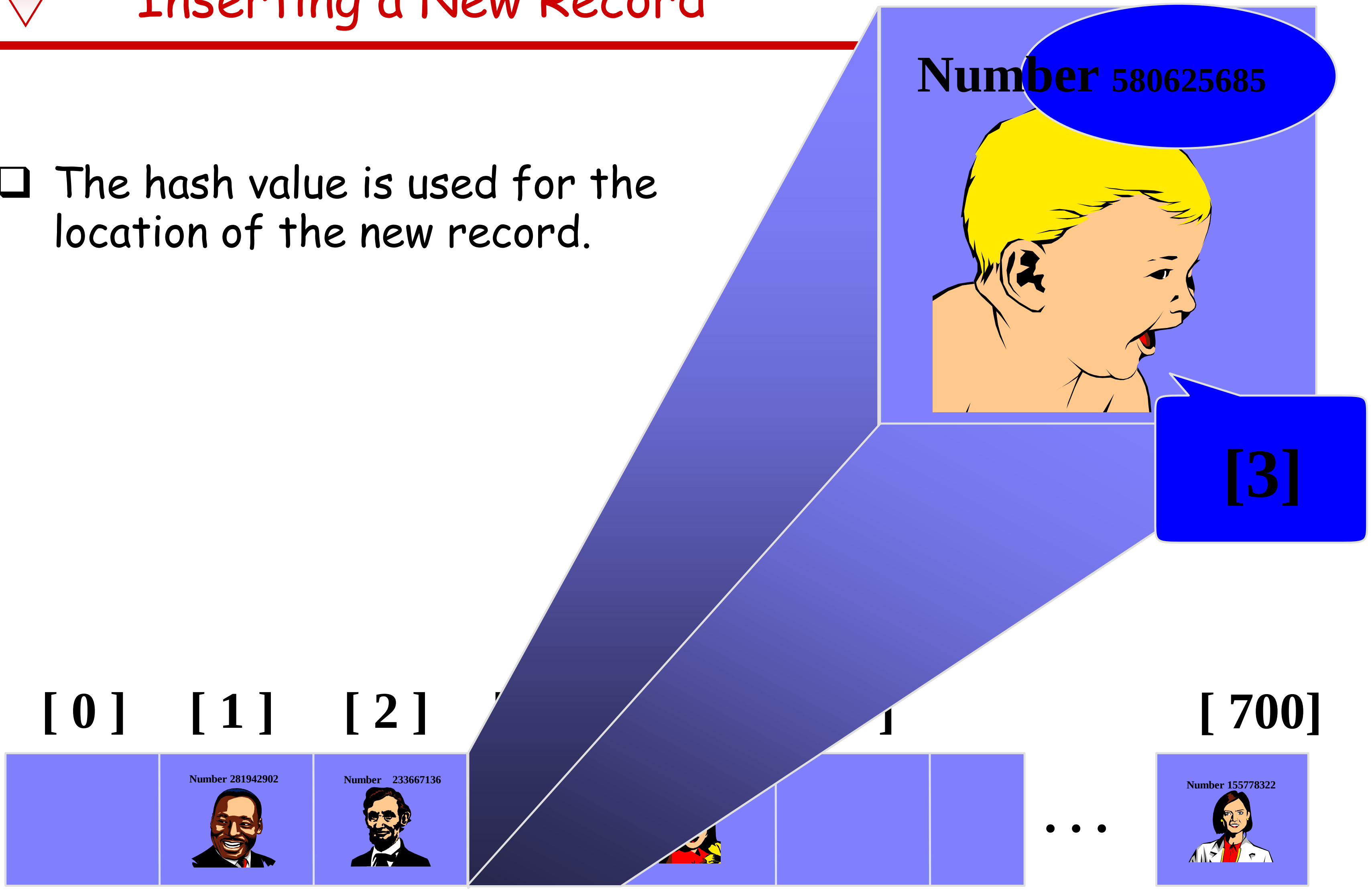
What is (580625685 mod 701) ?

Number 580625685
3



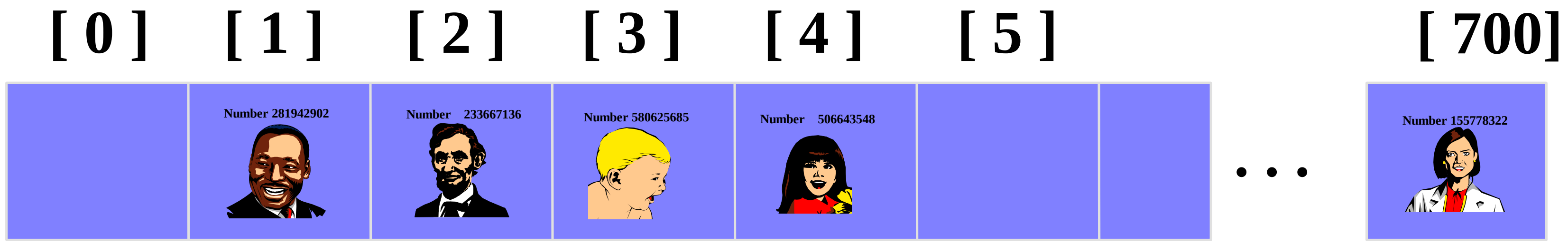
Inserting a New Record

- 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.



Applications of Hash Tables

- ❑ Keeping track of customer account information at a bank
  - ◆ Search through records to check balances and perform transactions
- ❑ Keep track of reservations on flights
  - ◆ Search to find empty seats, cancel/modify reservations
- ❑ Search engine
  - ◆ Looks for all documents containing a given word

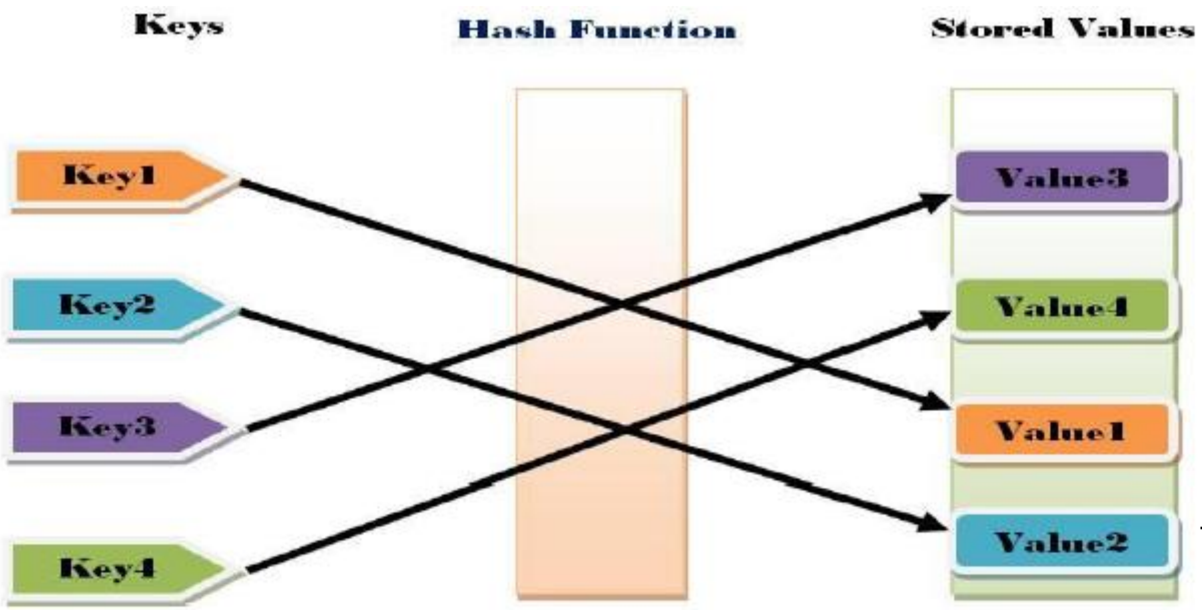
Hash Tables

- How to define a hash function? Really no limit to the number of possible hash functions.

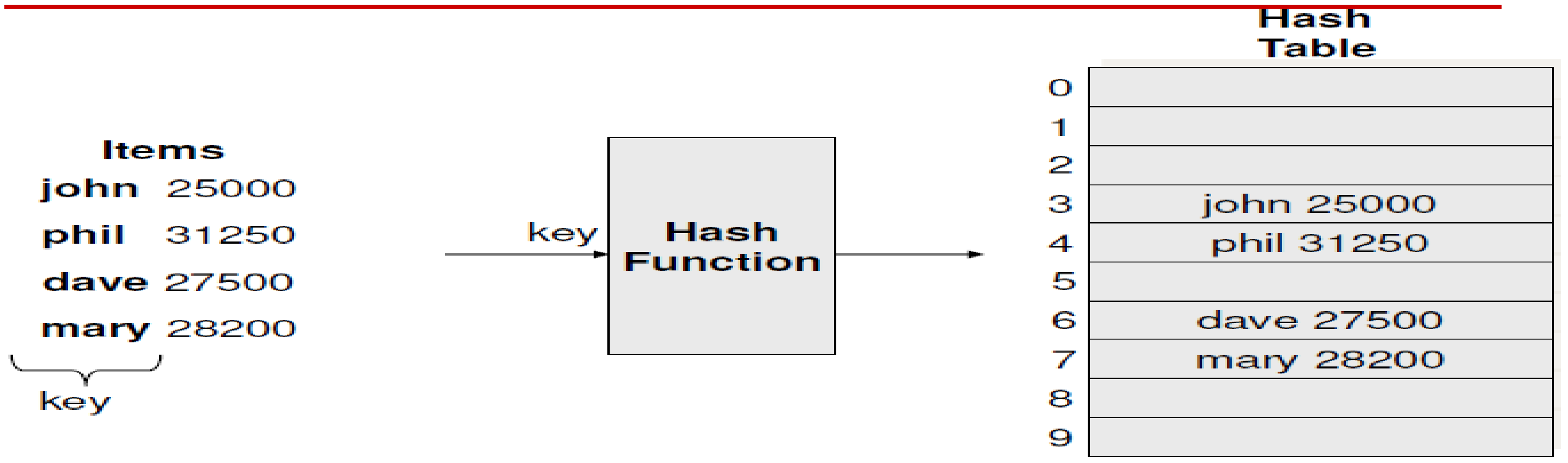
- A good hash function should;
  - Use only the data being hashed
  - Use all of the data being hashed
  - Be deterministic
  - Uniformly distribute data
  - Generate very different hash codes for very similar data

Choosing a Hash Function

- A hash function maps keys to small integers (buckets). An ideal hash function maps the keys to the integers in a random-like manner, so that bucket values are evenly distributed even if there are regularities in the input data.
- This process can be divided into two steps:
  - Map the key to an integer.
  - Map the integer to a bucket.

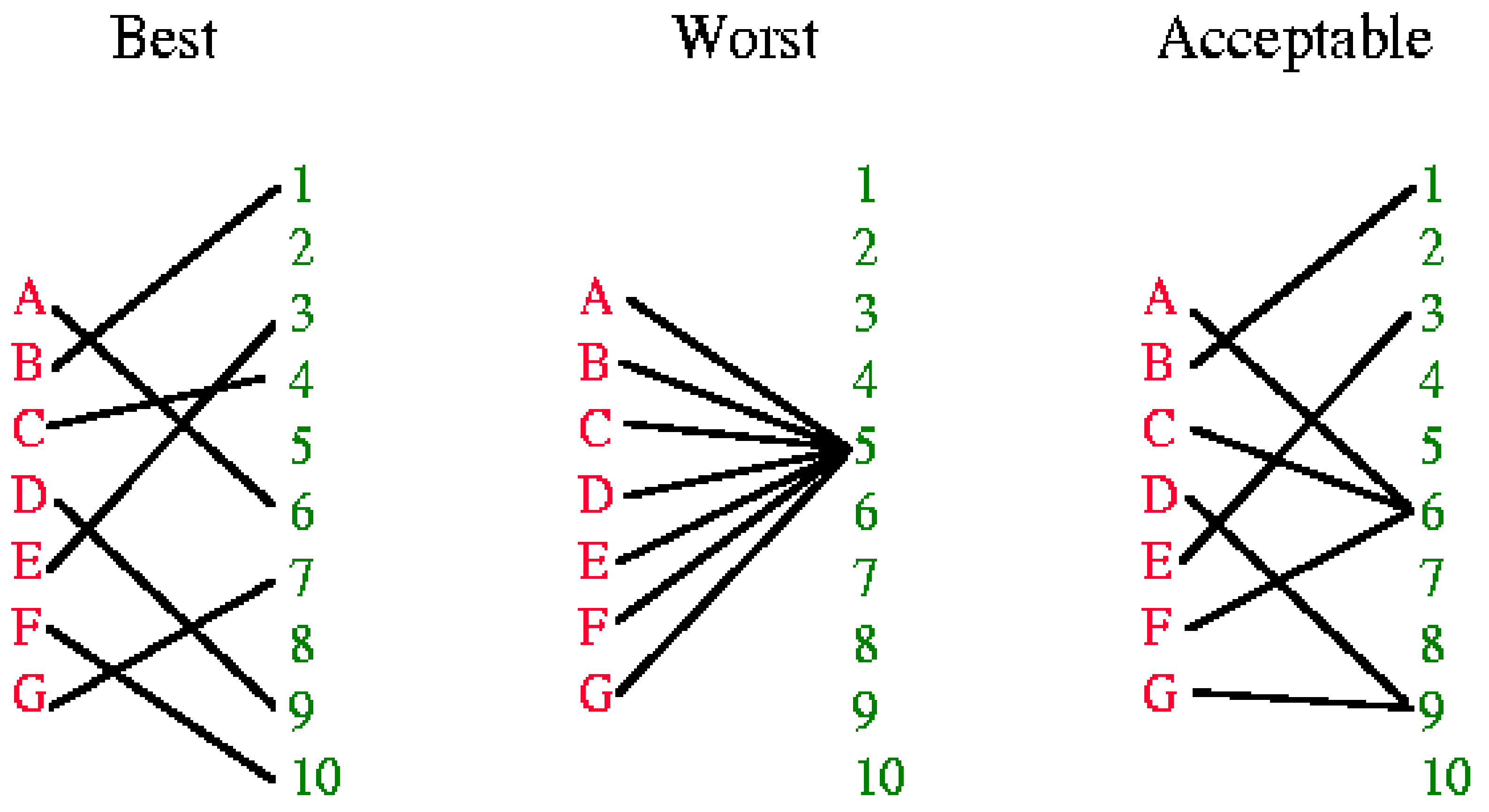


Example



- The hash function:
  - must be simple to compute.
  - must distribute the keys evenly among the cells.
  - Minimize collision.
  - Resolve any collision
  - Generate very different hash codes for very similar data

Good and Bad Functions



Different Approaches to generate hash function

- ❑Truncation Method
- ❑Mid Square Method
- ❑Folding Method
- ❑Modular Method

Truncation Method

- Ignore a part of the key and use the remaining part directly as the index.
- **Ex:1** If a hash table contains 999 entries at the most or 999 different key indexes may be kept, then a hash function may be defined such that from an eight digit integer 12345678, first, second and fifth digits from the right may be used to define a key index i.e. 478, which is the key position in the hash lookup table where this element will be inserted. Any other key combination may be used.
- **Ex:2** If students have an 9-digit identification number, take the last 3 digits as the table position
  - e.g. **925371622**
  - becomes
  - **622**

Mid Square Method

- In Mid-Square method, the key element is multiplied by itself to yield the square of the given key.
- If the hash table contains maximum 999 entries, then three digits from the result of square may be chosen as a hash key for mapping the key element in the lookup table.
- It generates random sequences of hash keys, which are generally key dependent.
- Mid Square method is a flexible method of selecting the hash key value.

- **Example**: If the input is the number 4567, squaring yields an 8-digit number, 20857489. The middle two digits of this result are 57. All digits of the original key value (equivalently, all bits when the number is viewed in binary) contribute to the middle two digits of the squared value. Thus, the result is not dominated by the distribution of the bottom digit or the top digit of the original key value.

Folding Method

- Partition the key into several parts and combine the parts in a convenient way (often addition or multiplication) to obtain the index.
- Ex-1:

  An eight digit integer can be divided into groups of three, three and two digits (or any other combination) the groups added together and truncated if necessary to be in the proper range of indices.

  Hence 12345678 maps to 123+456+78 = 657, since any digit can affect the index, it offers a wide distribution of key values.
- Ex-2:

  Split a 9-digit number into three 3-digit numbers, and add them e.g. 925371622 becomes 925 + 376 + 622 = 1923

Modular Method

- For mapping a given key element in the hash table, mod operation of individual key is calculated. The remainder denotes particular address position of each element.
- The result so obtained is divided by an integer, usually taken to be the size of the hash table to obtain the remainder as the hash key to place that element in the lookup table.

Modular Method

- **Idea:**
  - Map a key k into one of the m slots by taking the remainder of k divided by m

    h(k) = k mod m
- **Advantage:**
  - fast, requires only one operation
- **Disadvantage:**
  - Certain values of m are bad, e.g.,
    - power of 2
    - non-prime numbers

Collisions in Hash Tables

- A **collision** occurs when two pieces of data, when run through the hash function, yield the same hash code.
- Presumably we want to store both pieces of data in the hash table, so we shouldn't simply overwrite the data that happened to be placed in there first.
- We need to find a way to get both elements into the hash table while trying to preserve quick insertion and lookup.

Collisions in Hash Tables

- Two or more keys hash to the same slot!!
- For a given set K of keys
  - If $|K| \leq m$, collisions may or may not happen, depending on the hash function
  - If $|K| > m$, collisions will definitely happen (i.e., there must be at least two keys that have the same hash value)
- Avoiding collisions completely is hard, even with a good hash function

Collision Resolution(Open Hashing)

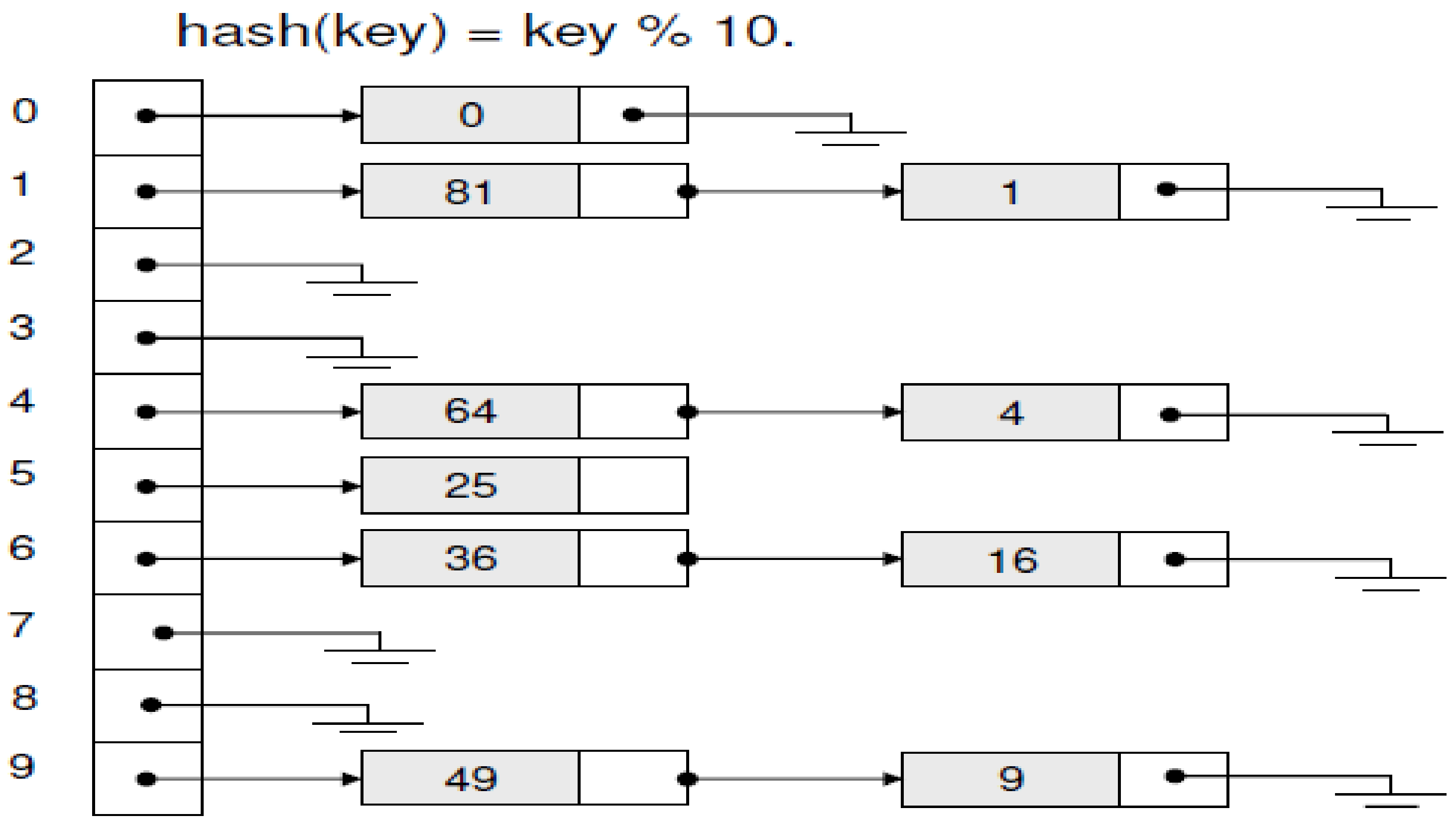
- If, when an element is inserted, it hashes to the same value as an already inserted element, then we have a collision and need to resolve it.
- There are several methods for dealing with this:
  - **Separate chaining**
  - **Open addressing**
    - Linear Probing
    - Quadratic Probing
    - Double Hashing

Separate Chaining

- The idea is to keep a list of all elements that hash to the same value.
  - The array elements are pointers to the first nodes of the lists.
  - A new item is inserted to the front of the list.
- Advantages:
  - Better space utilization for large items.
  - Simple collision handling: searching linked list.
  - Overflow: we can store more items than the hash table size.
  - Deletion is quick and easy: deletion from the linked list.
- Disadvantages
  - Linked lists could get long
    - Especially when N approaches M
    - Longer linked lists could negatively impact performance
  - More memory because of pointers

Example

- Keys: 0, 1, 4, 9, 16, 25, 36, 49, 64, 81



Operations

- **Initialization**: all entries are set to NULL
- **Find**:
  - locate the cell using hash function.
  - sequential search on the linked list in that cell.
- **Insertion**:
  - Locate the cell using hash function.
  - (If the item does not exist) insert it as the first item in the list.
- **Deletion**:
  - Locate the cell using hash function.
  - Delete the item from the linked list.

Resolving Collisions in Hash Tables

- **Chaining**
  - We have eliminated clustering.
  - We know from experience with linked lists that insertion (and creation, if necessary) into a linked list is an O(1) operation.
  - For lookup, we only need to search through what is hopefully a small list, since we are distributing what would otherwise be one huge list across n lists.

Closed Hashing(Open Addressing)

- If we have enough contiguous memory to store all the keys ($m > N$) ➡ store the keys in the table itself
- No need to use linked lists anymore
- Basic idea:
  - Insertion: if a slot is full, try another one, until you find an empty one
  - Search: follow the same sequence of probes
  - Deletion: more difficult ... (we'll see why)
- Search time depends on the length of the probe sequence!

Closed Hashing(Open Addressing)

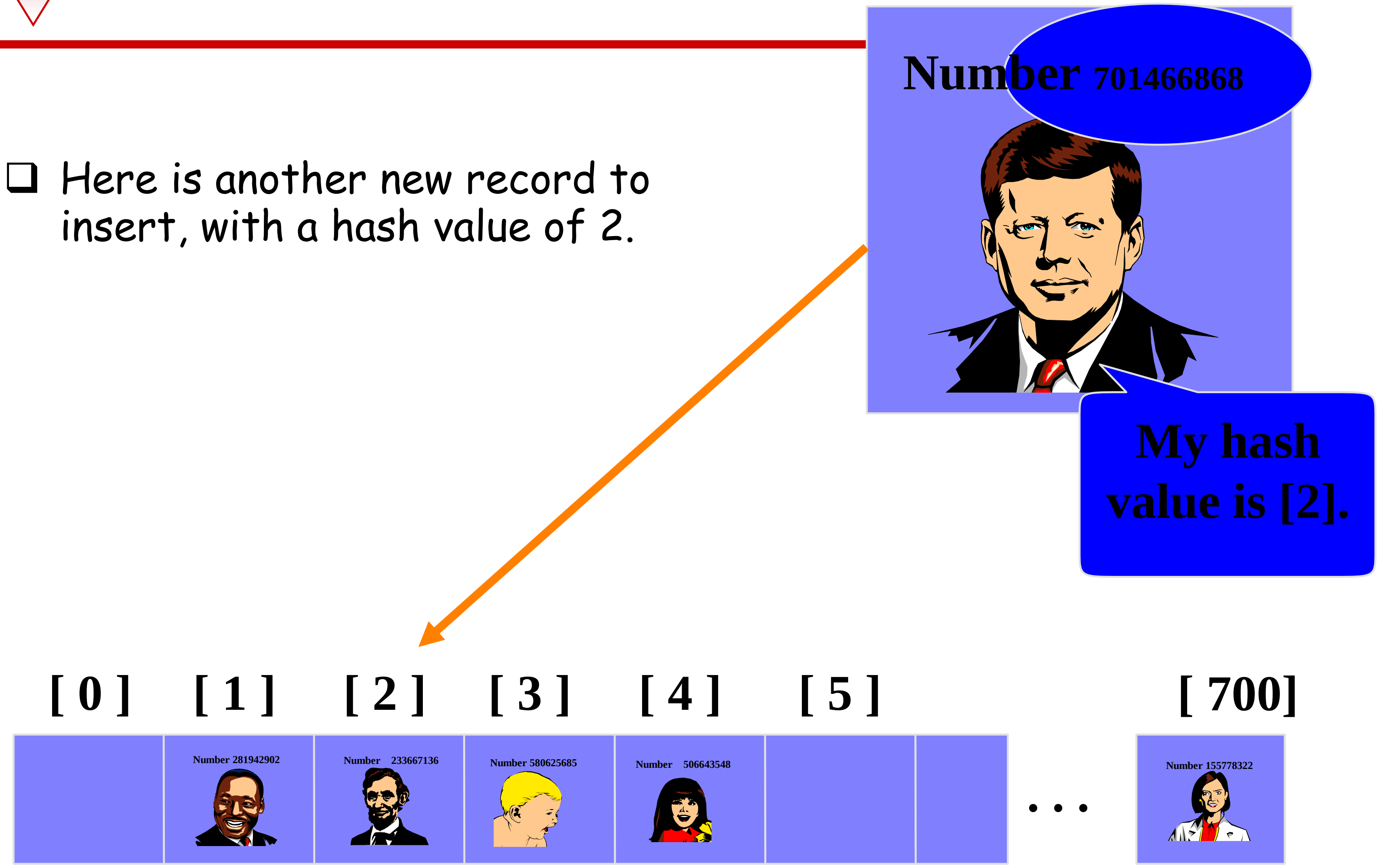
- **Linear Probing**
  - In this method, if we have a collision, we try to place the data in the next consecutive element in the array (wrapping around to the beginning if necessary) until we find a vacancy.
  - That way, if we don't find what we are looking for in the first location, at least hopefully the element is somewhere nearly.
  - Linear probing is subject to a problem called clustering. Once there's a miss, two adjacent cells will contains data, making it more likely in the future that the cluster will grow.
  - Even if we switch to another probing technique, we are still limited. We can only store as much data as we have locations in our array.

Closed Hashing(Open Addressing)

- **When a collision occurs, look elsewhere in the table for an empty slot**
- Advantages over chaining
  - No need for list structures
  - No need to allocate/deallocate memory during insertion/deletion (slow)
- Disadvantages
  - Slower insertion – May need several attempts to find an empty slot
  - Table needs to be bigger (than chaining-based table) to achieve average-case constant-time performance

Collisions (Closed Hashing Example

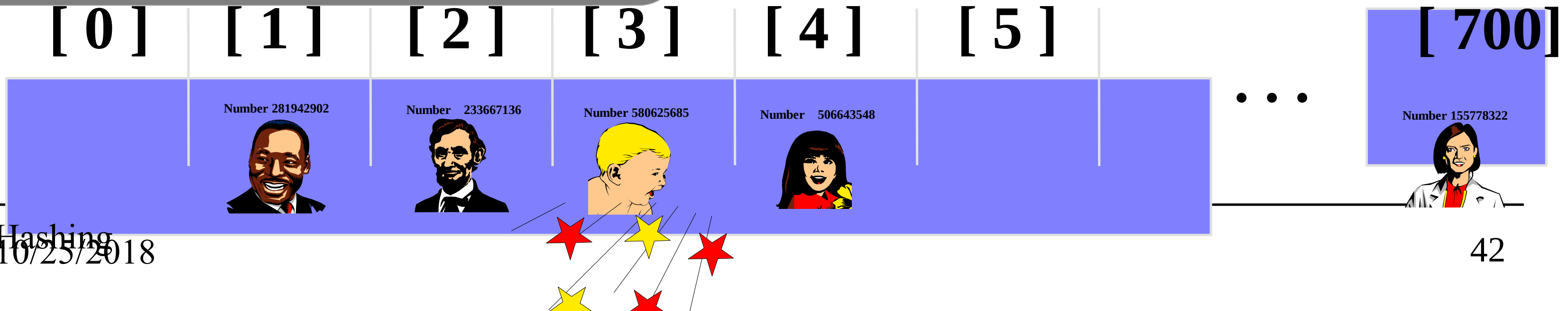
- Here is another new record to insert, with a hash value of 2.



Collisions (Closed Hashing Example

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

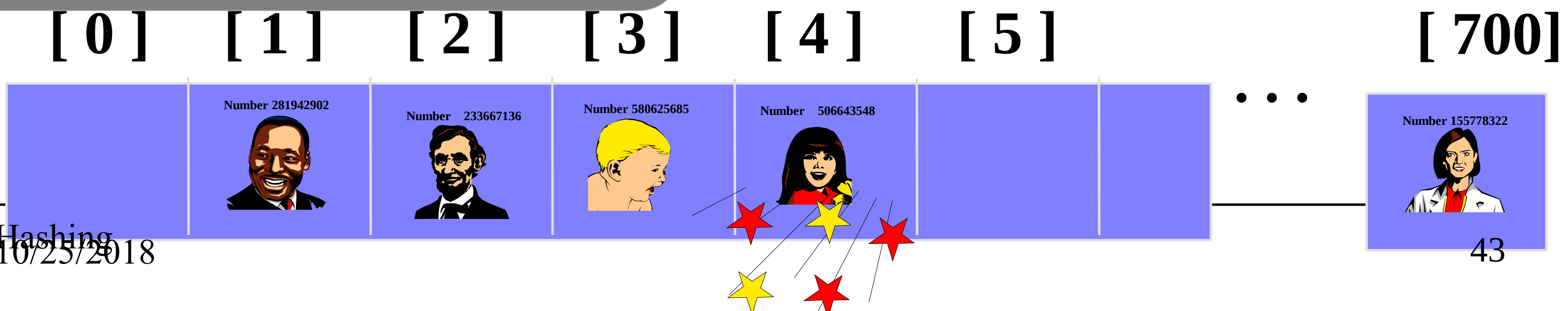
When a collision occurs,
move forward until you find an empty spot.



Collisions (Closed Hashing Example

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

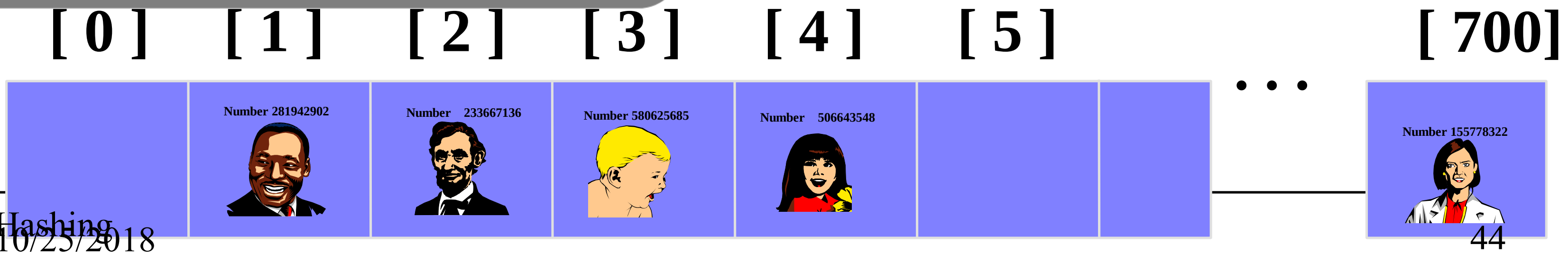
**When a collision occurs,
move forward until you find an empty spot.**



Collisions (Closed Hashing Example

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

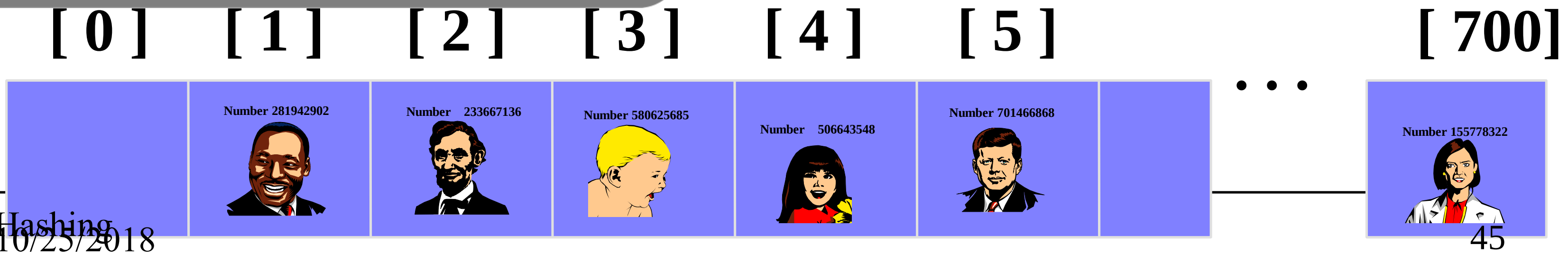
**When a collision occurs,
move forward until you find an empty spot.**



Collisions (Closed Hashing Example

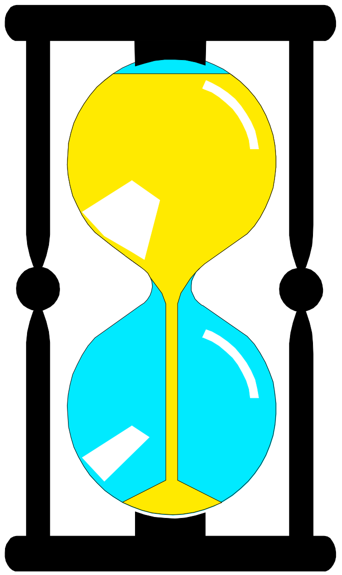
- This is called a **collision**, because there is already another valid record at [2].

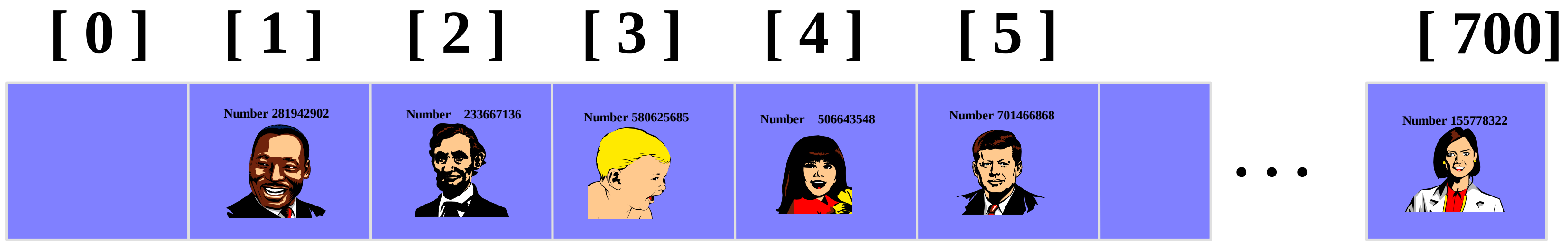
The new record goes in the empty spot.



A Quiz

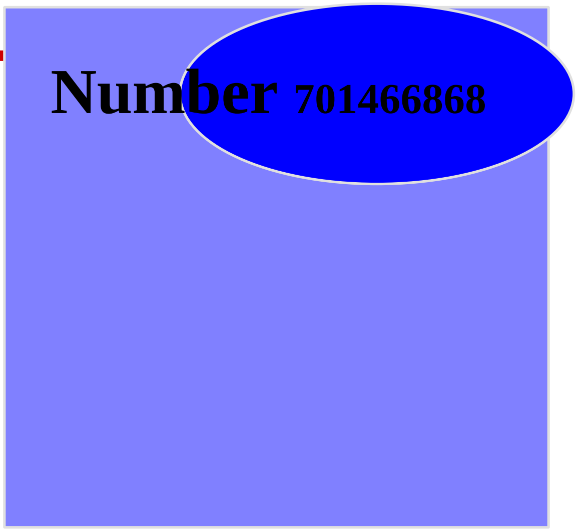
Where would you be placed in this table, if there is no collision? Use your social security number or some other favorite number.

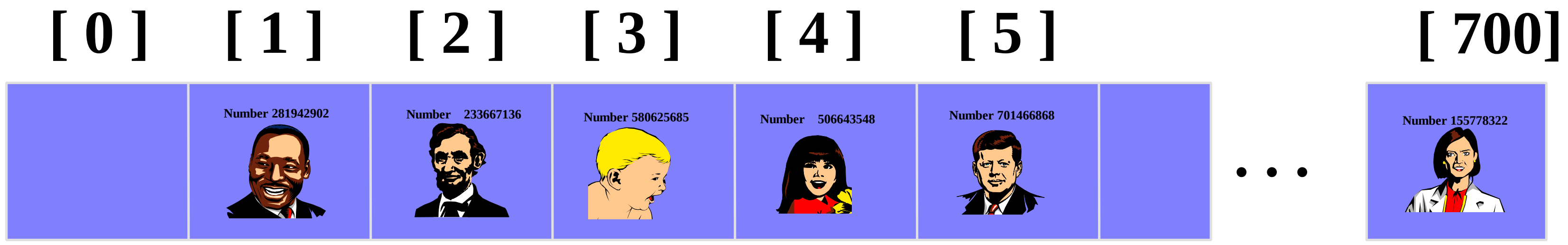




Searching for a Key

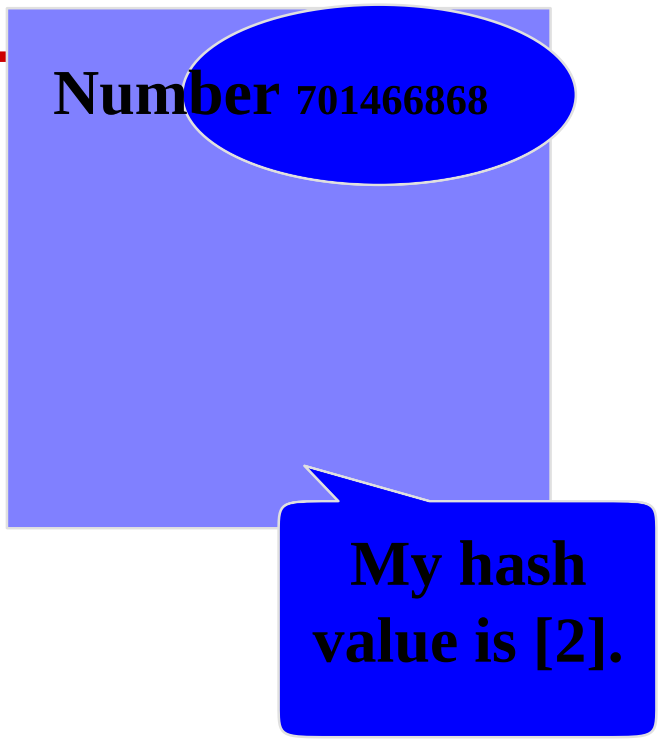
- The data that's attached to a key can be found fairly quickly.

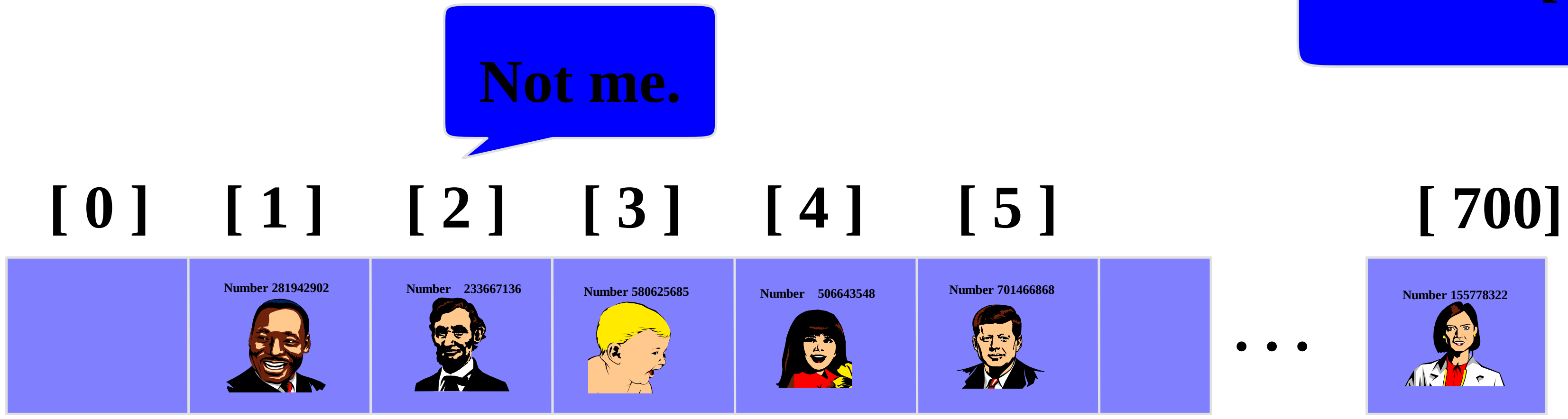




Searching for a Key

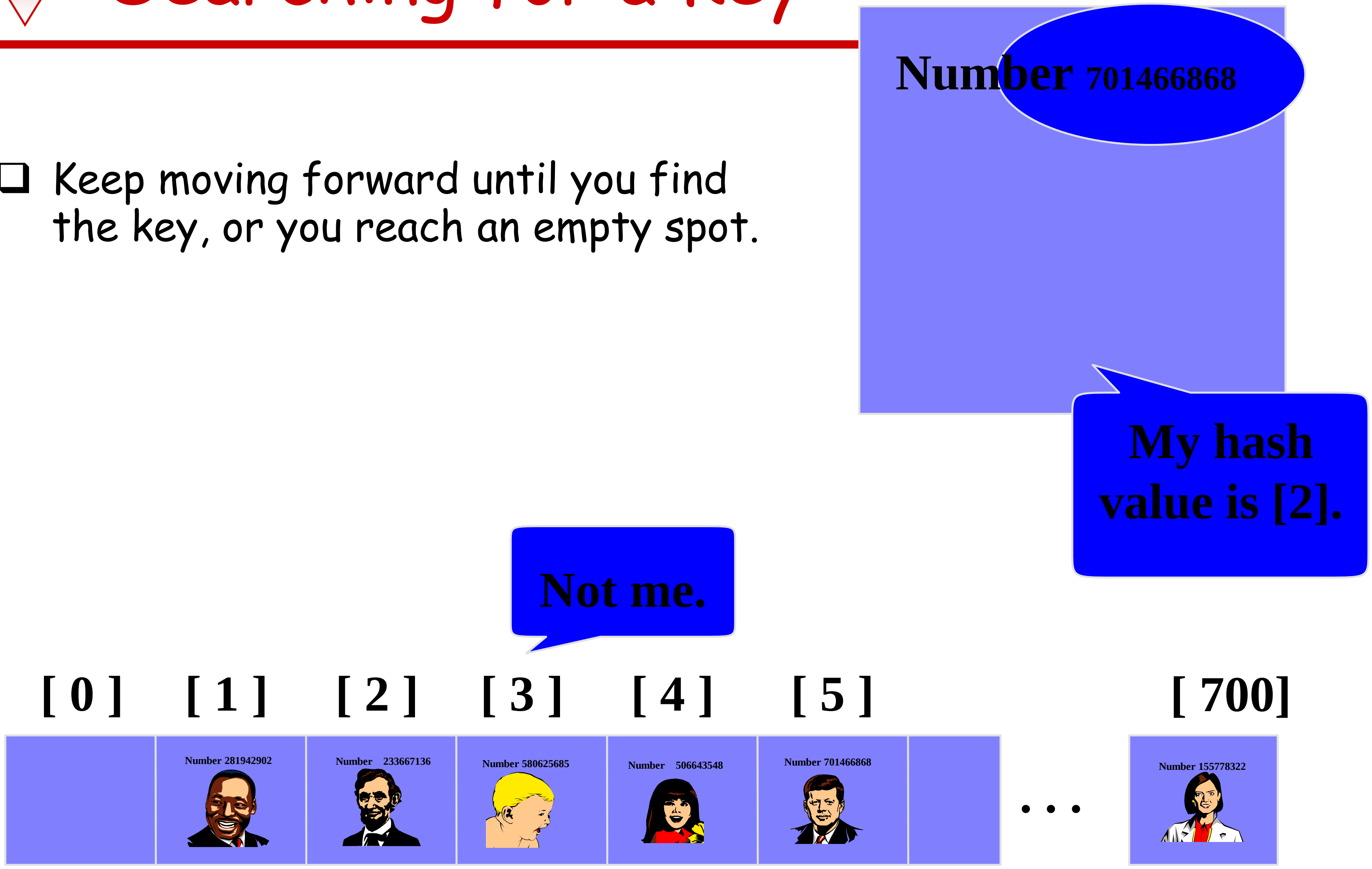
- Calculate the hash value.
- Check that location of the array for the key.





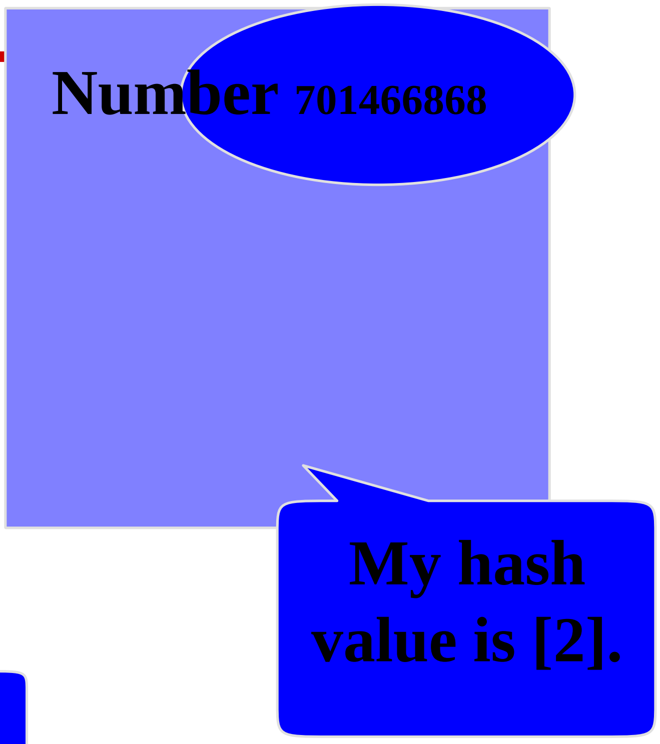
Searching for a Key

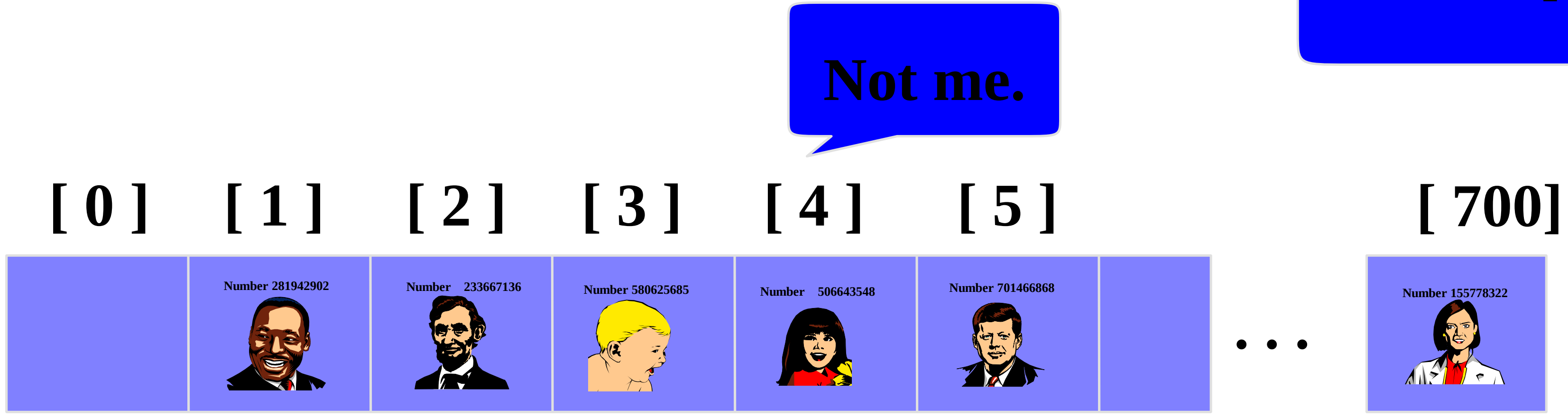
- Keep moving forward until you find the key, or you reach an empty spot.



Searching for a Key

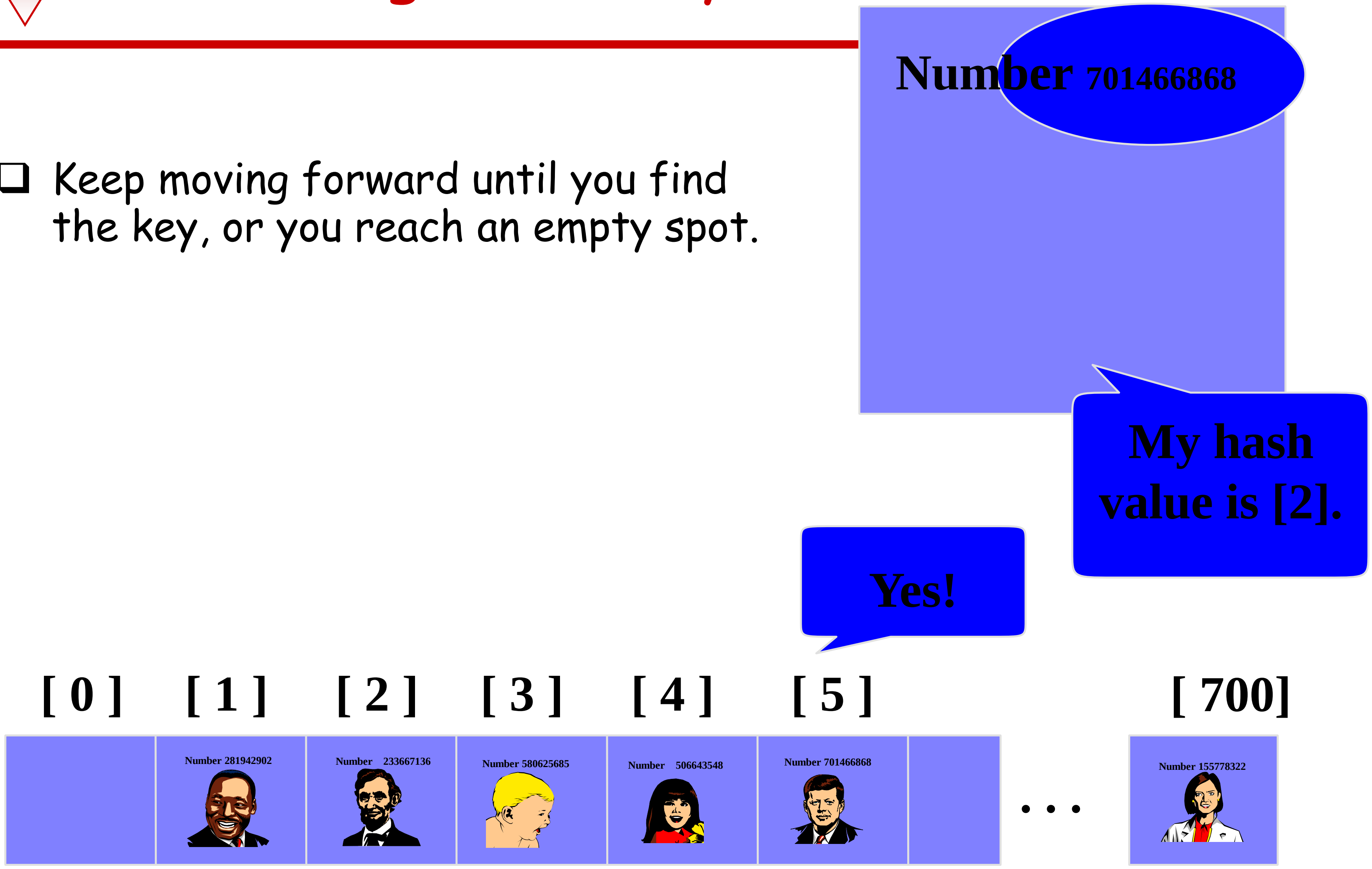
- Keep moving forward until you find the key, or you reach an empty spot.





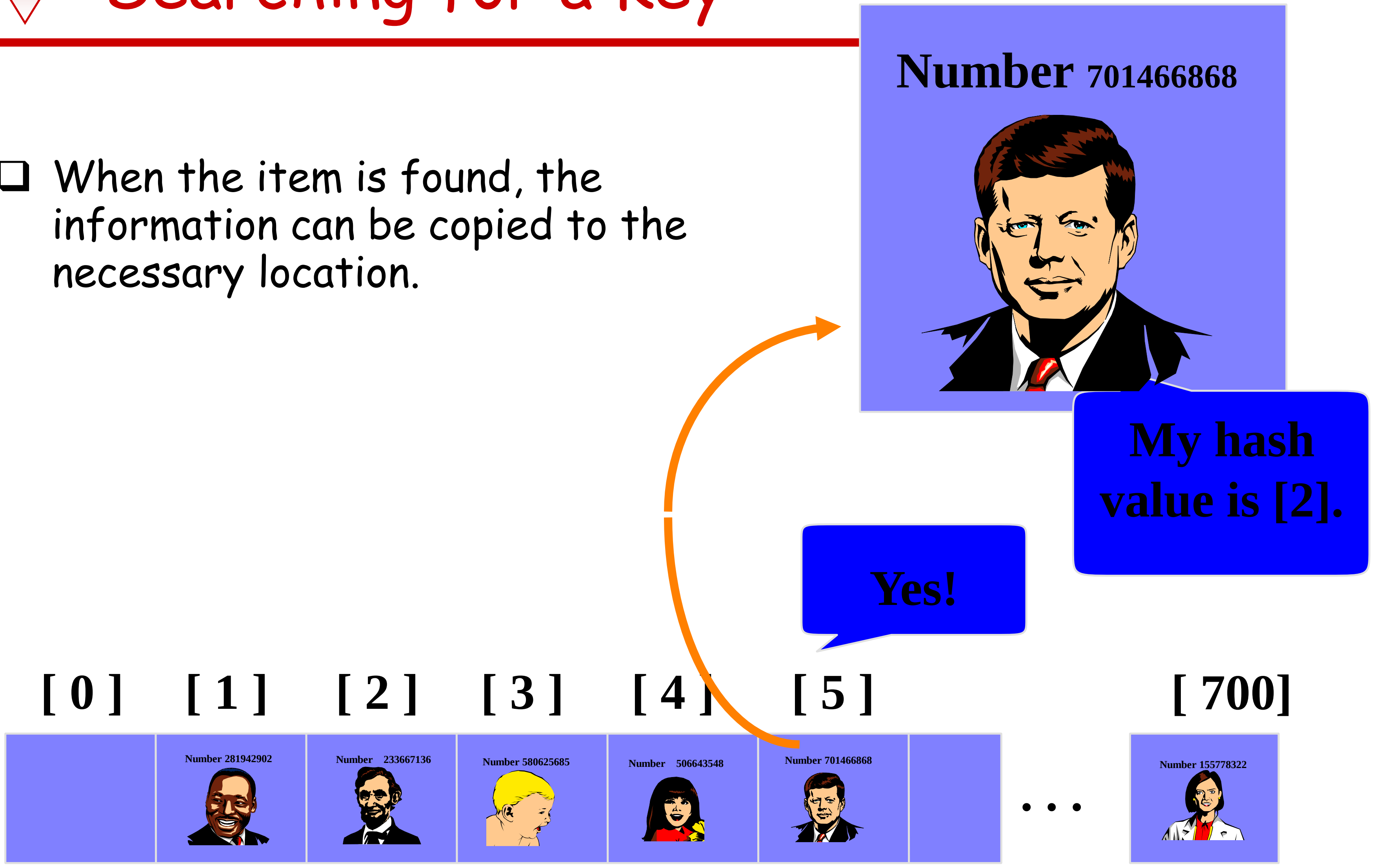
Searching for a Key

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



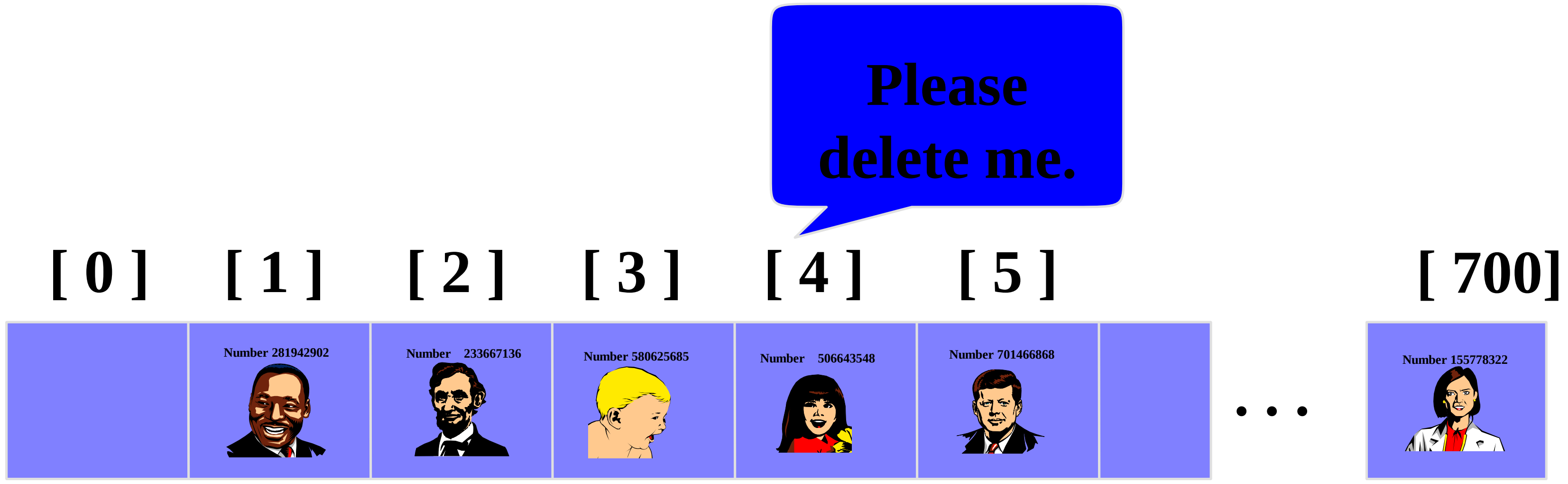
Searching for a Key

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



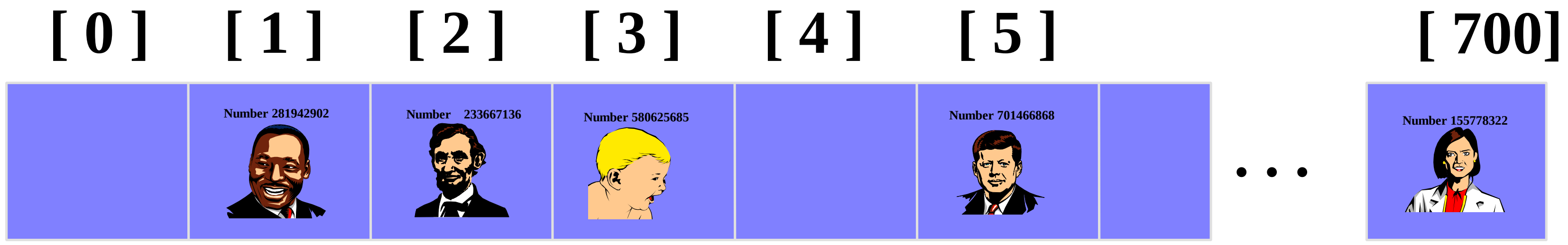
Deleting a Record

- Records may also be deleted from a hash table.



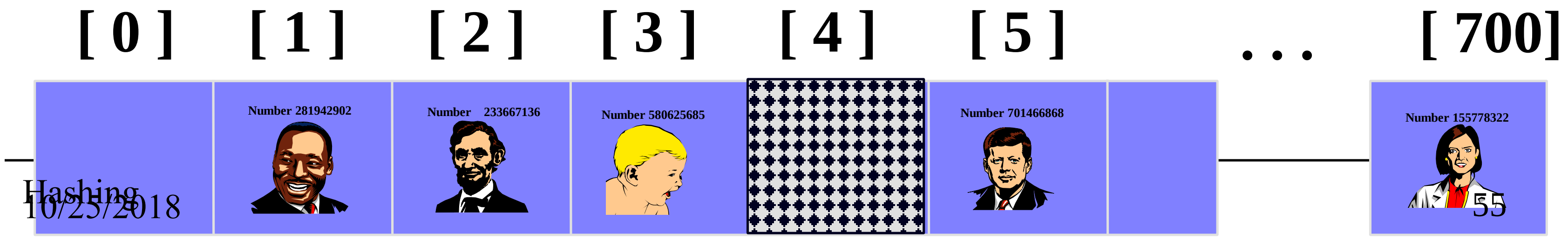
Deleting a Record

- 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.



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.



Schemes to reduce collision

- Spread out the records(uniform distribution)
  - A hash function is uniformly random if for any key in the file, every address is equally likely to be chosen.
- Use extra memory
  - Increase the ratio of address space to key space
- Buckets
  - A single address contains more than one record.

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.

The End