ECE244

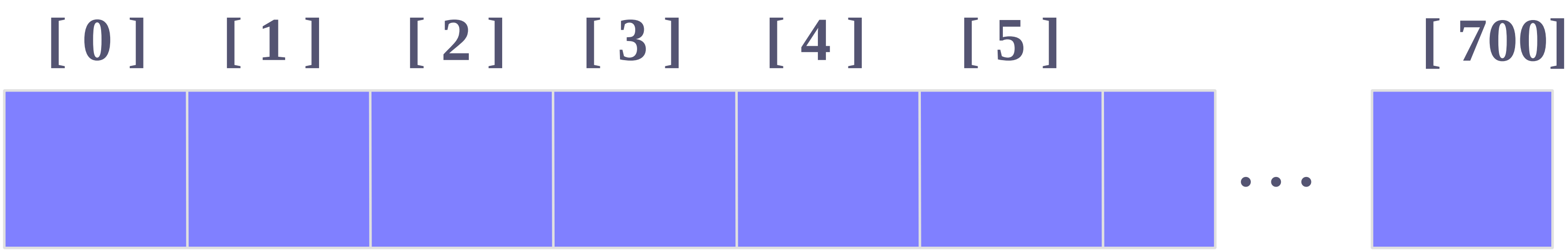
Wael Aboulsaadat

Hashtables

Acknowledgment: these slides are partially based on slides by; Prof. Schmidt from Drexel U., Prof. Shewchuk from UC Berkely, Kruse & Ryba Data Structure and Program Design in C++, Prof. Savitch Problem Solving in C++ and others

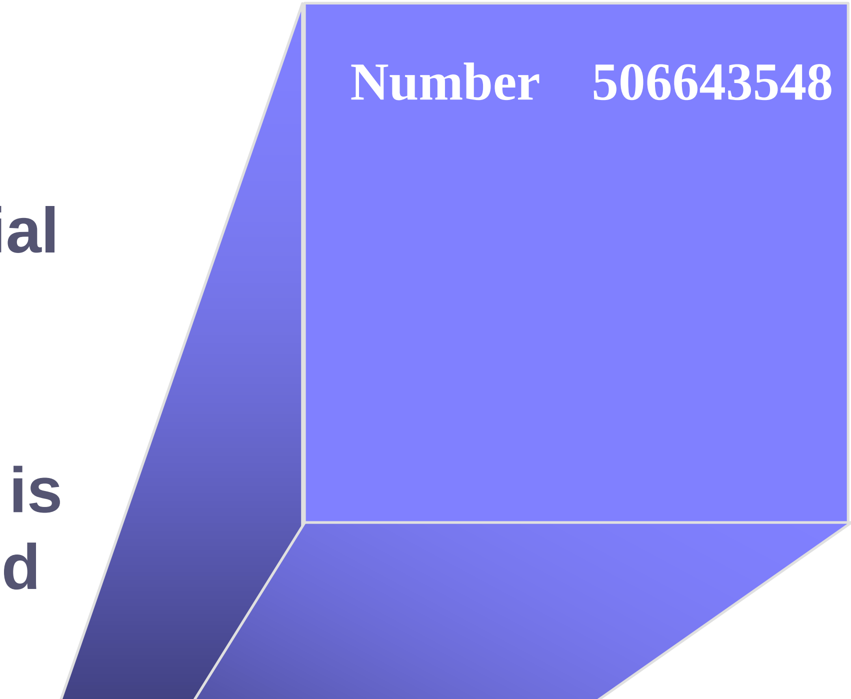
What is a Hash Table ?

- 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 ?



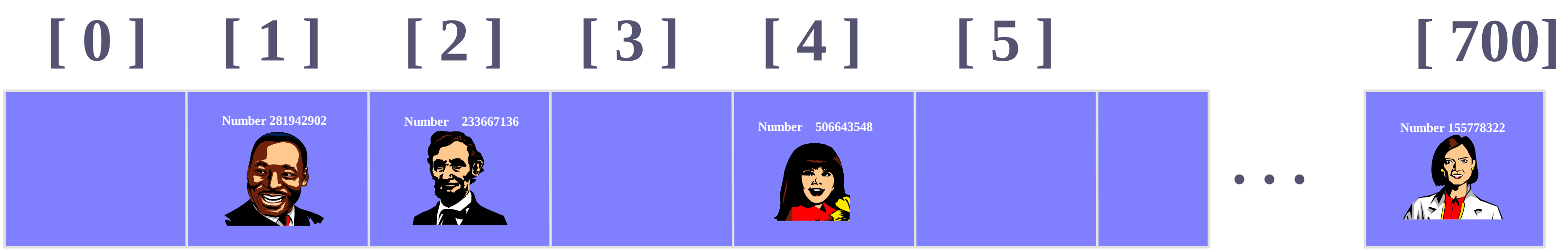
- Each record has a special field, called its **key**.
- In this example, the key is a long integer field called Number.

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.

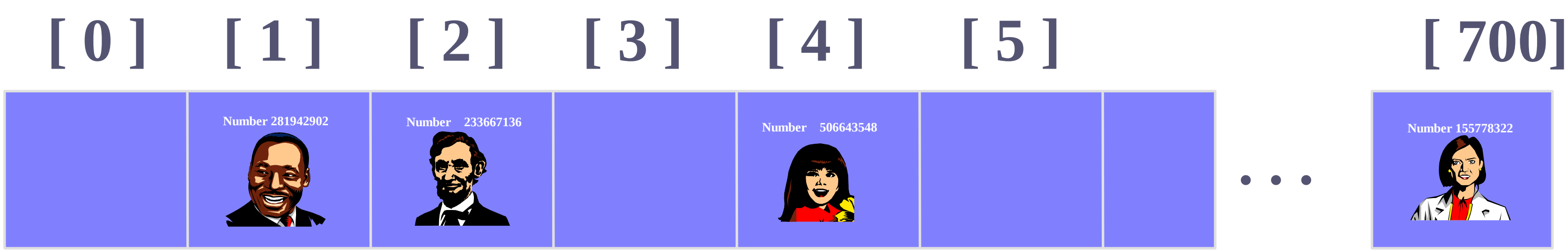
What is a Hash Table ?

- 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.

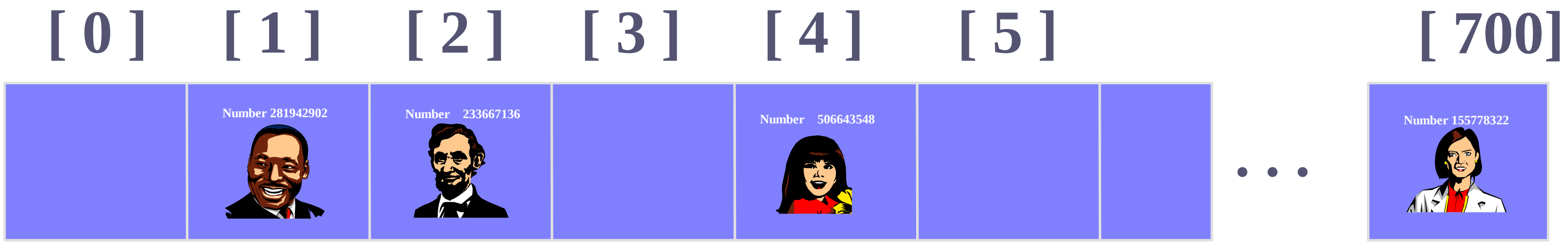


Inserting a New Record

- **Typical way to create a hash value:**

  (Number mod 701)

What is (580625685 mod 701) ?



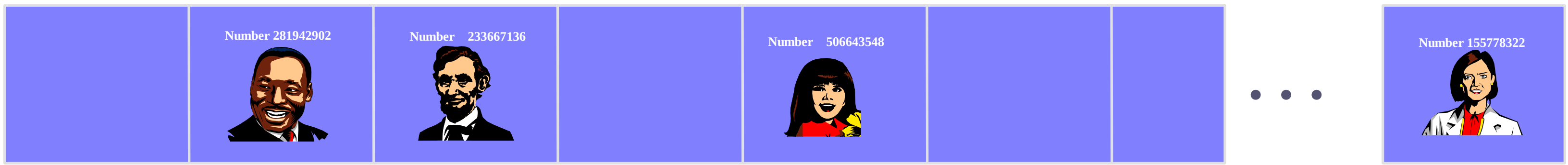
Inserting a New Record

- Typical way to create a hash value:

  (Number mod 701)

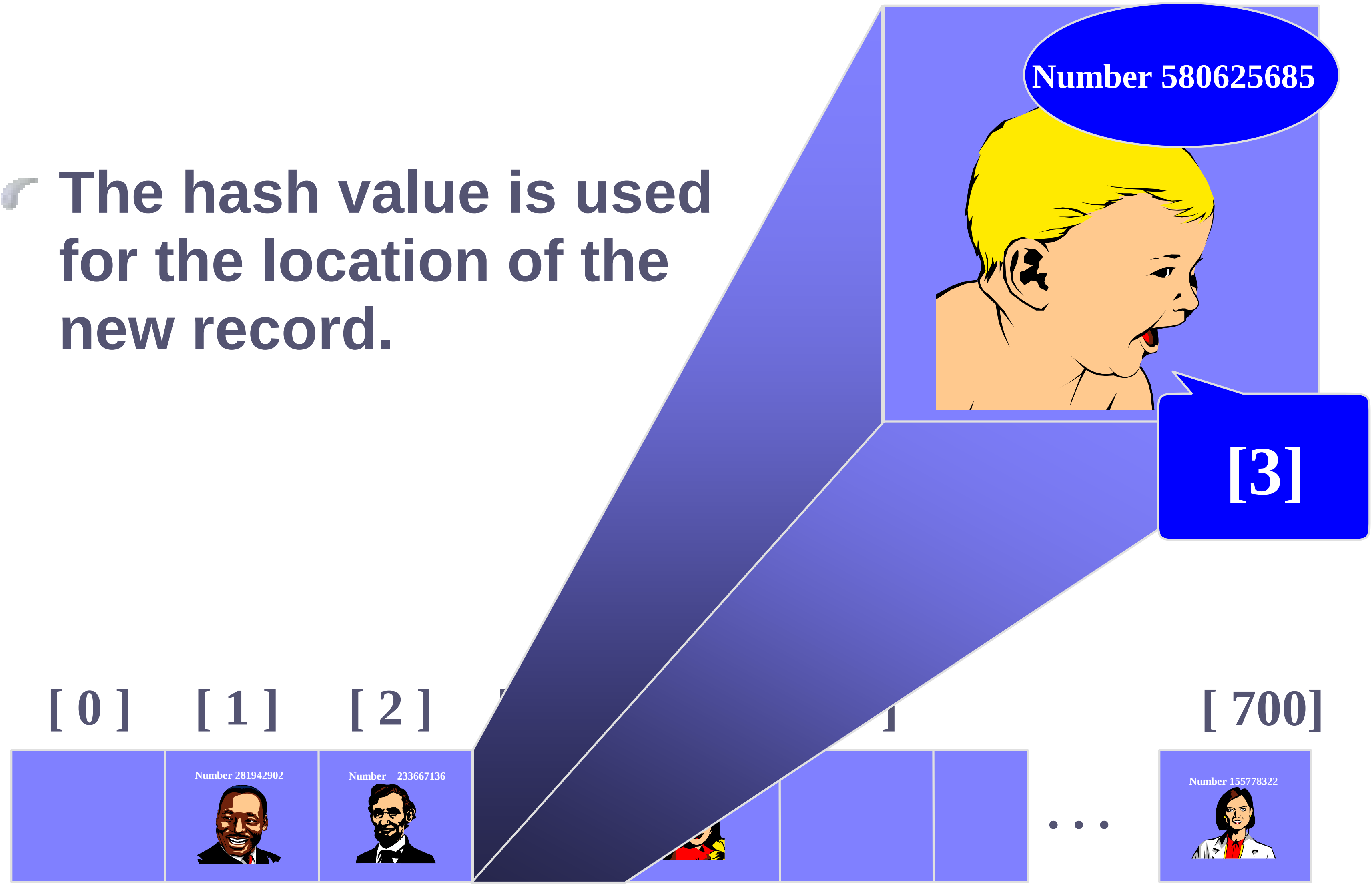
What is (580625685 mod 701) ?

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



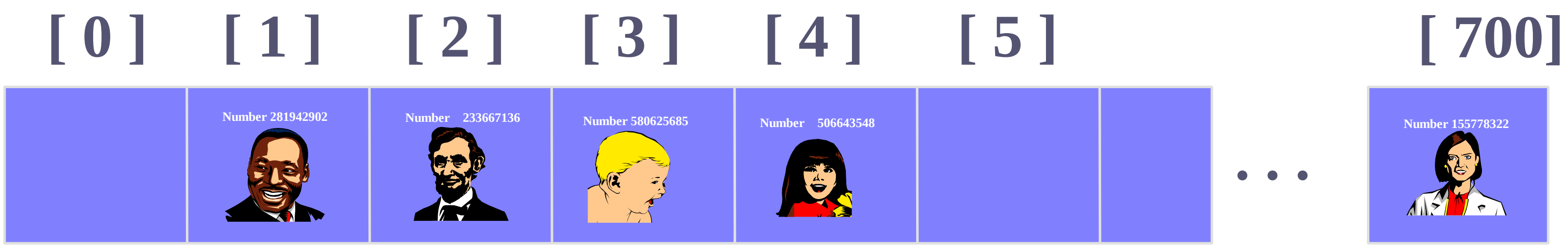
Inserting a New Record

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

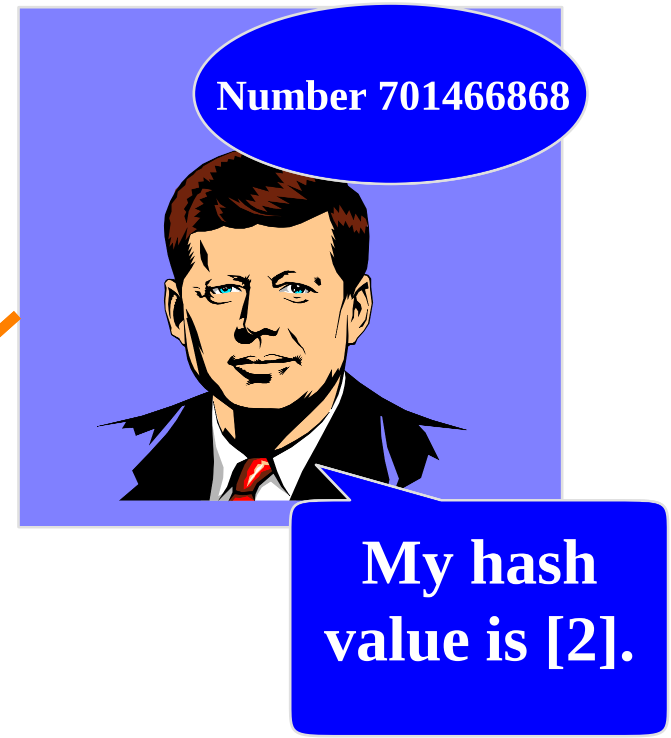


Inserting a New Record

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



Collisions!

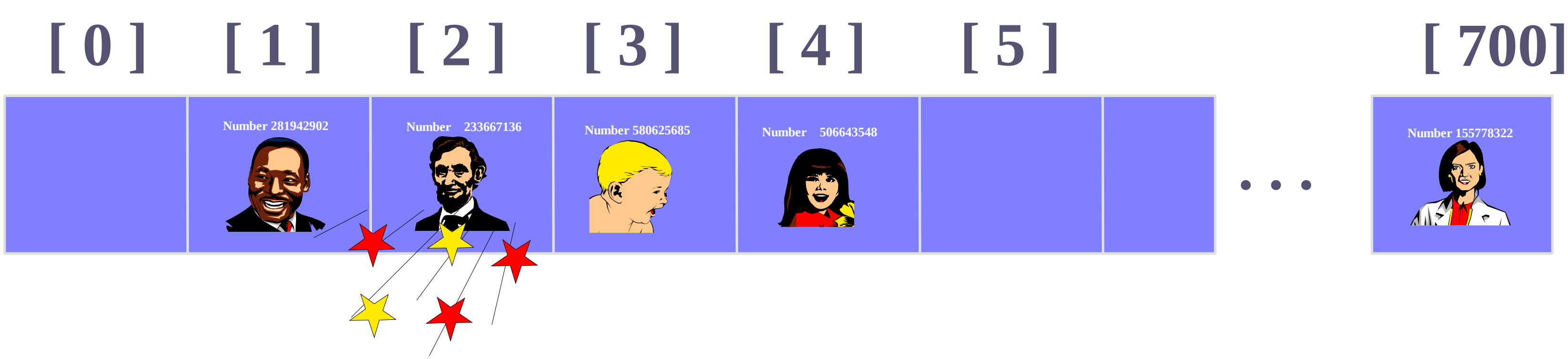


- Here is another new record to insert, with a hash value of 2.

- Collision Handling:

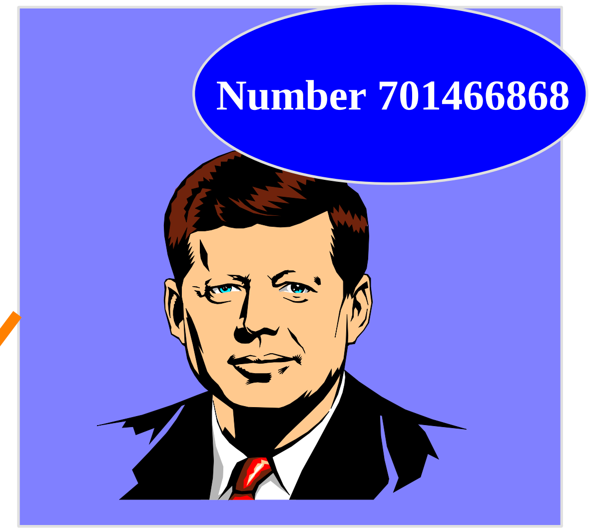
Open Addressing

Separate Chaining

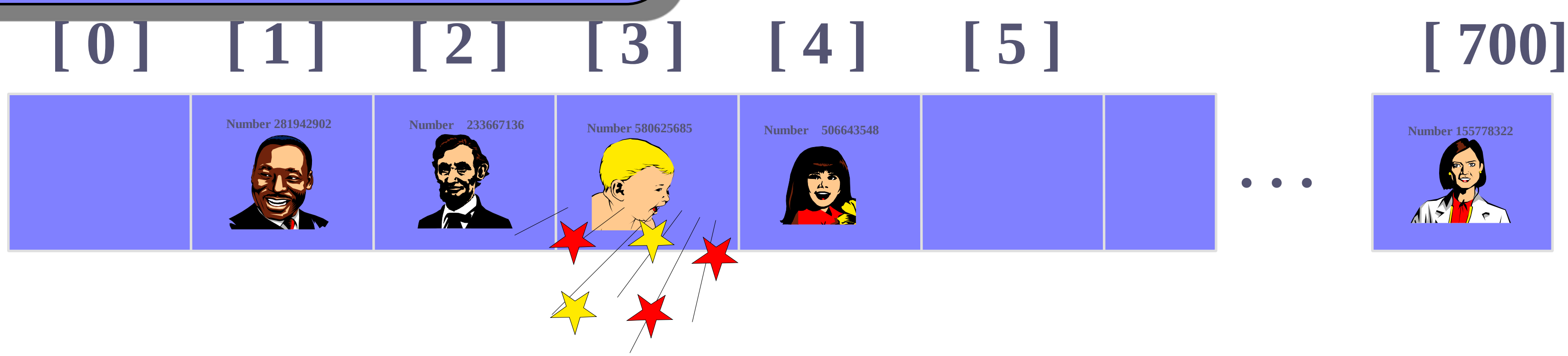


Open Addressing: Linear Probing

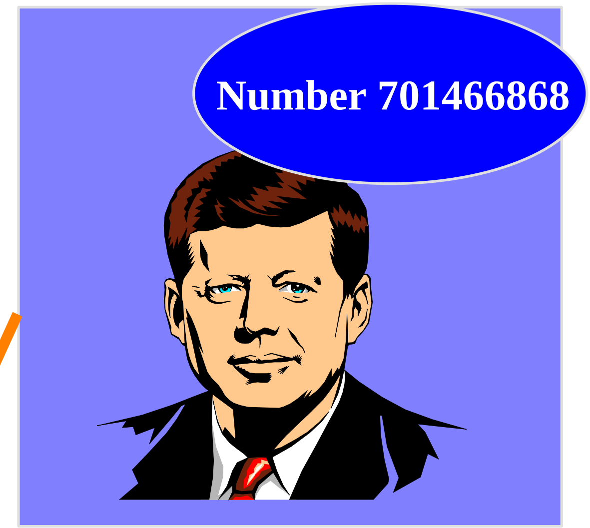
- This is called a collision, because there is already another valid record at [2].



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

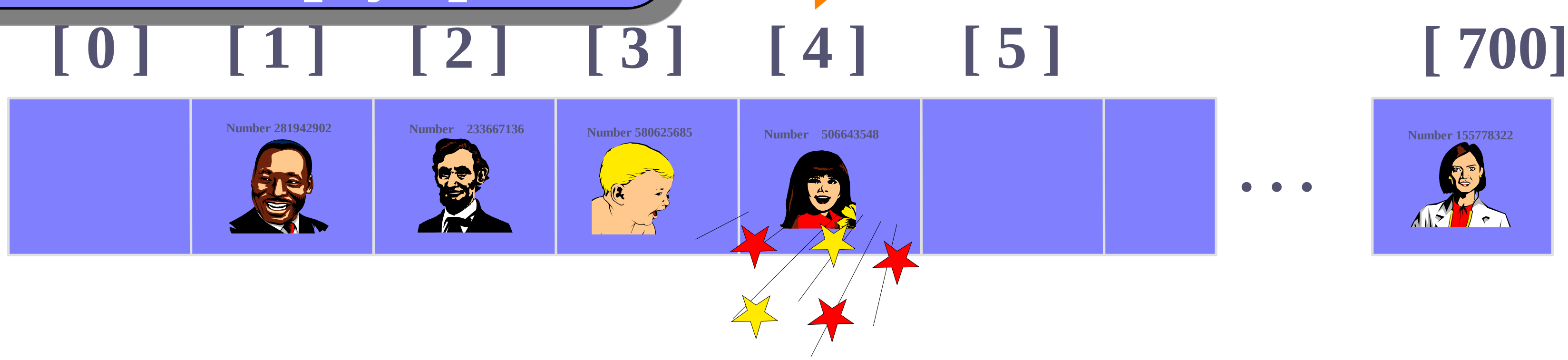


Open Addressing: Linear Probing



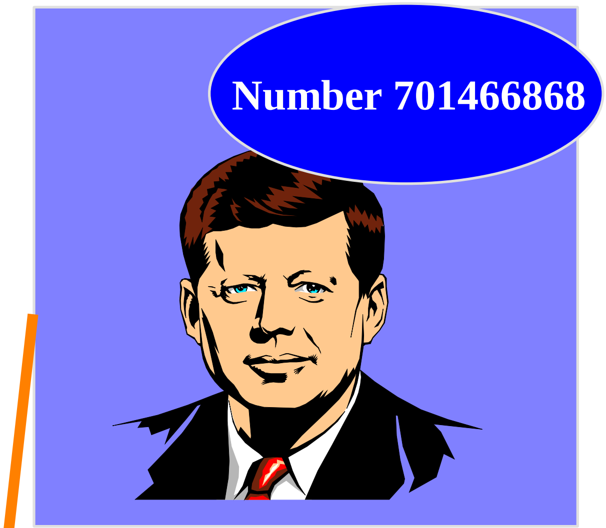
- This is called a collision, because there is already another valid record at [2].

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

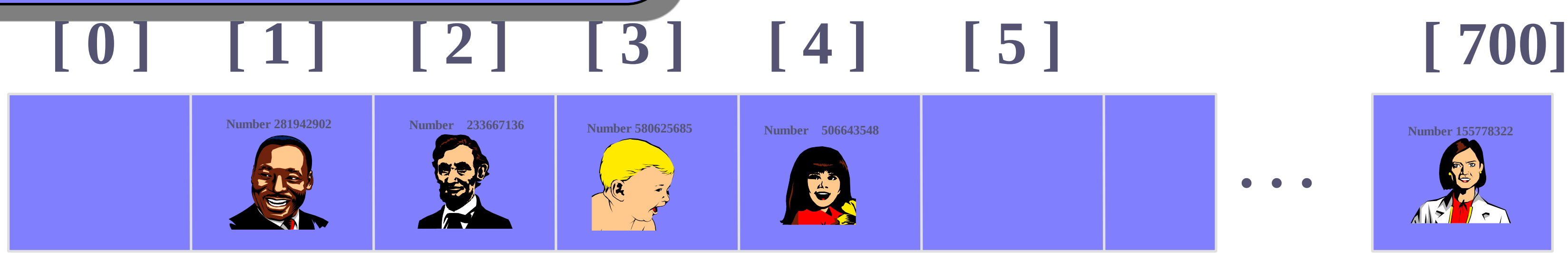


Open Addressing: Linear Probing

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



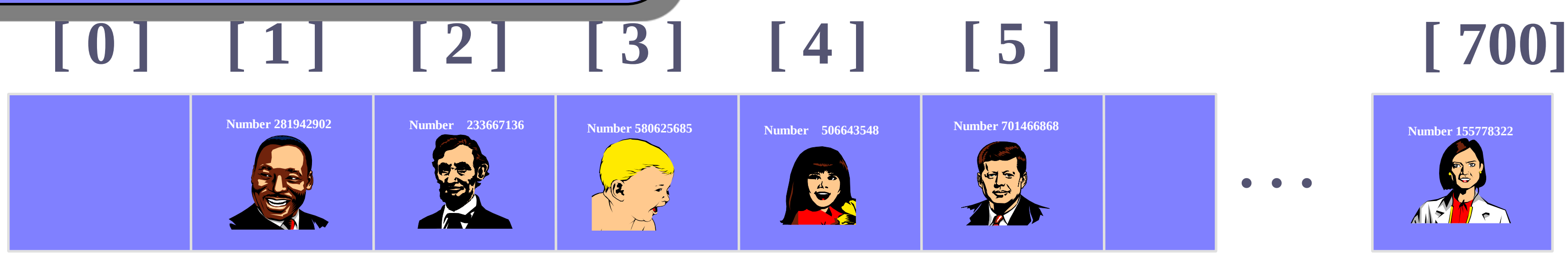
When a collision occurs,
move forward until you find an empty spot.



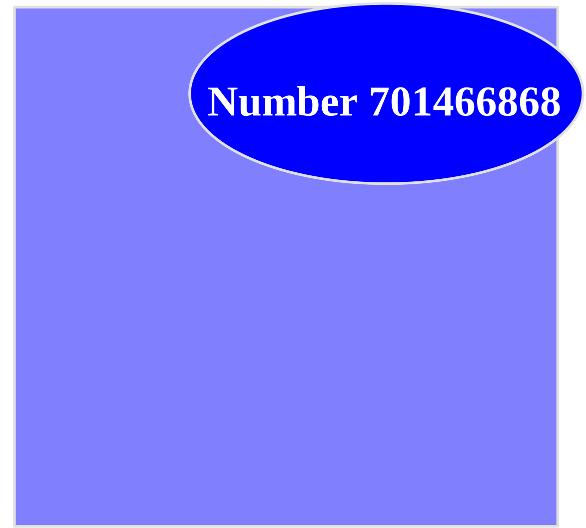
Open Addressing: Linear Probing

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

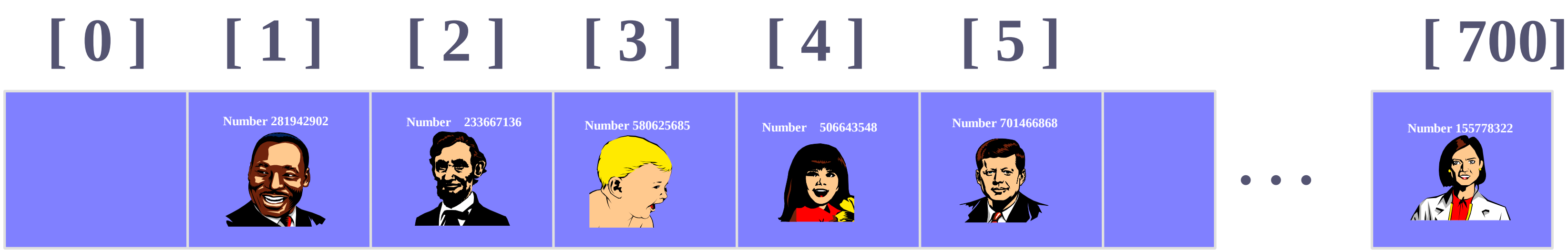
The new record goes in the empty spot.



Searching for a Key



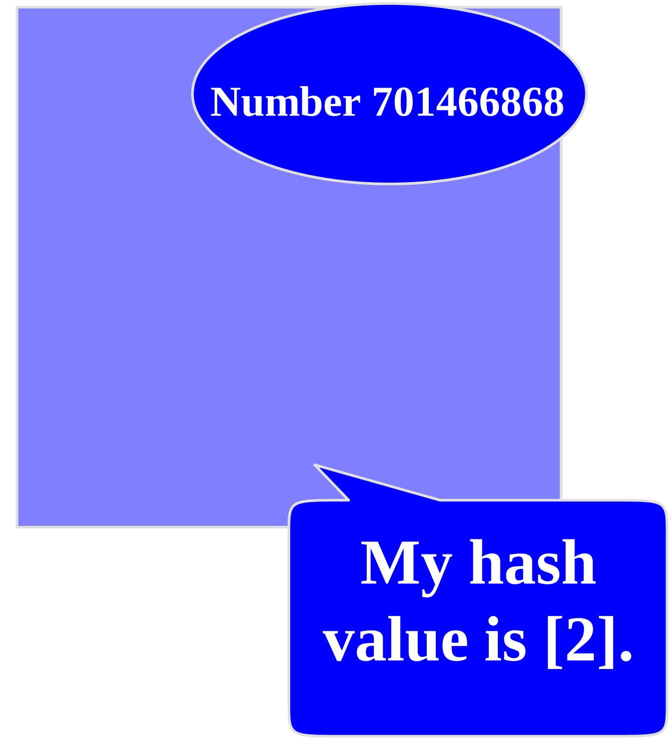
- The data that's attached to a key can be found fairly quickly.
- Note: assuming open addressing/ linear probing

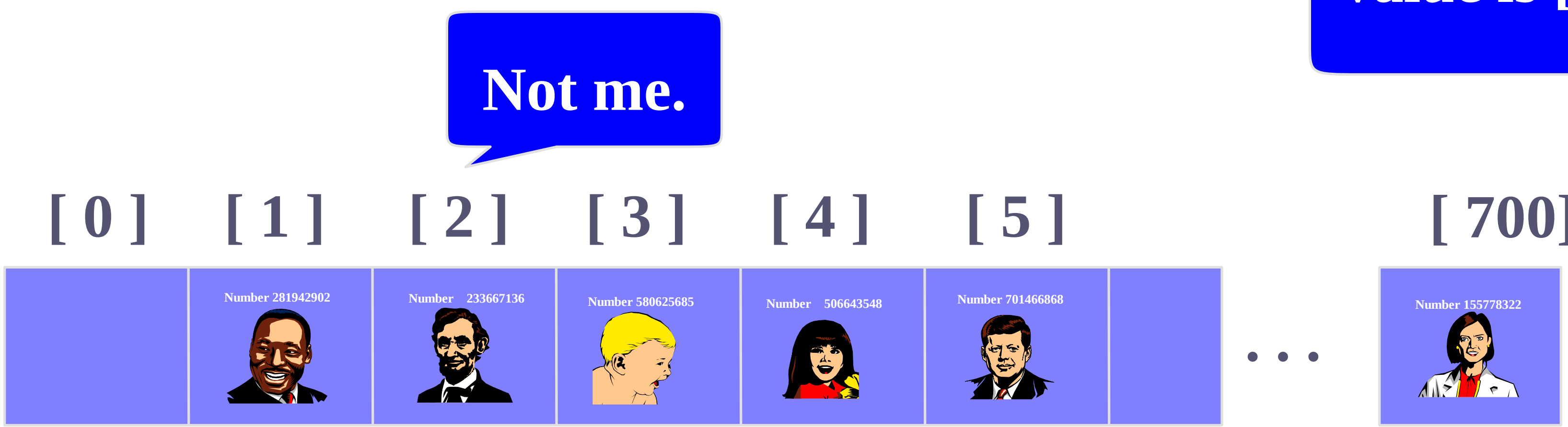


Searching for a Key

1) Calculate the hash value.

2) Check that location of the array for the key.





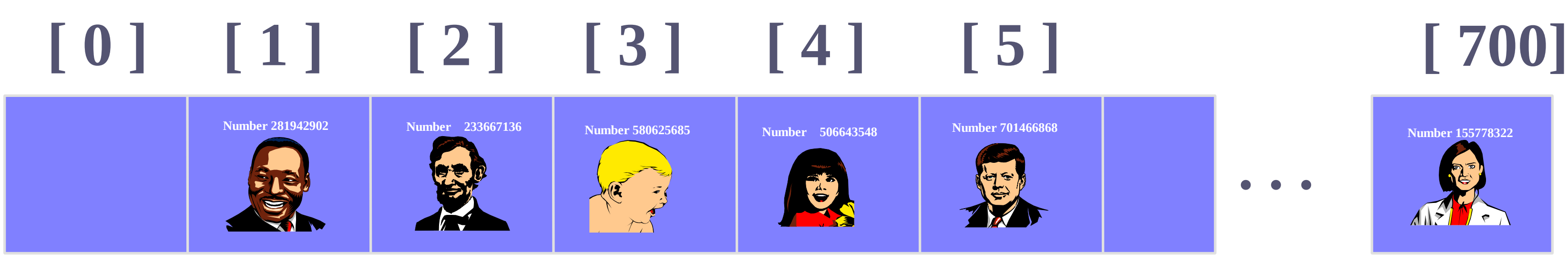
Searching for a Key

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

Number 701466868

My hash value is [2].

Not me.



Searching for a Key

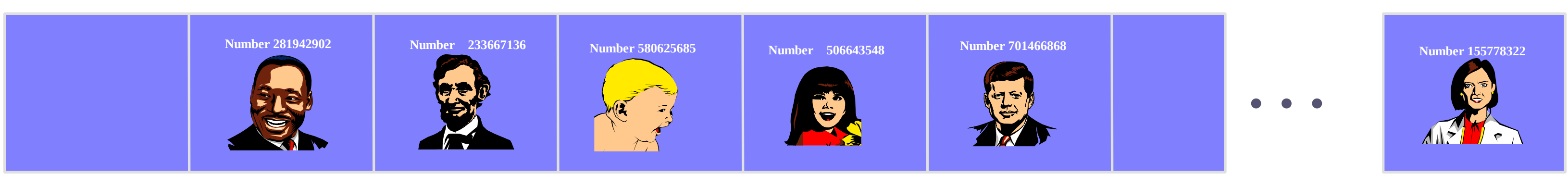
Number 701466868

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

My hash value is [2].

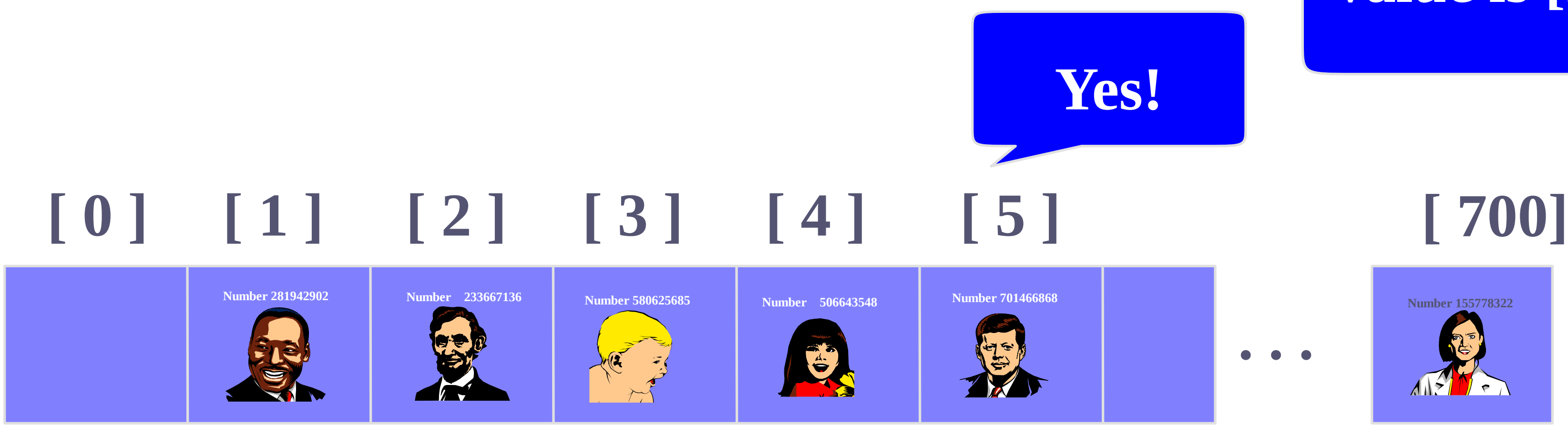
Not me.

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



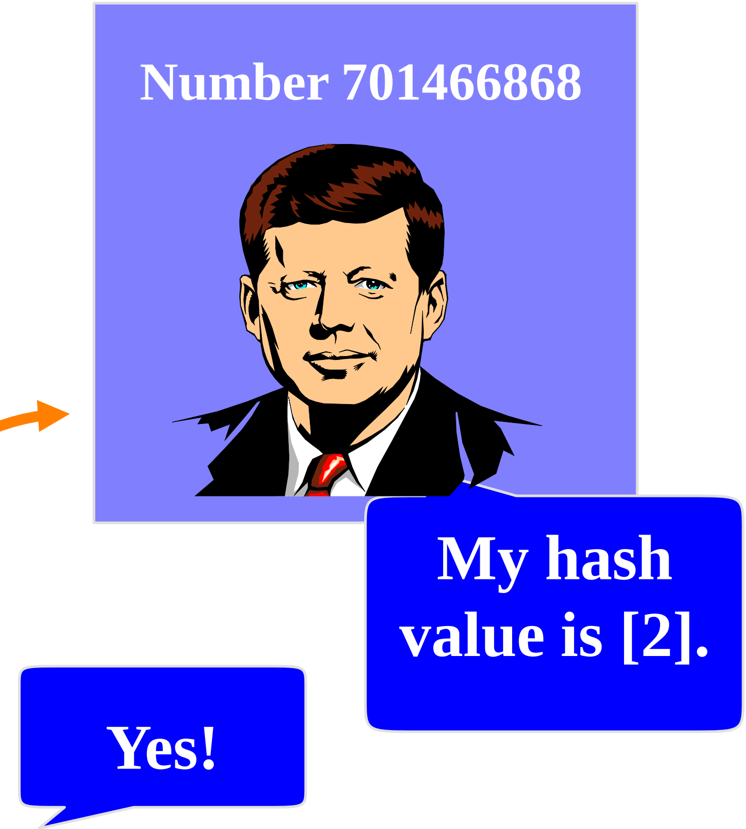
Searching for a Key

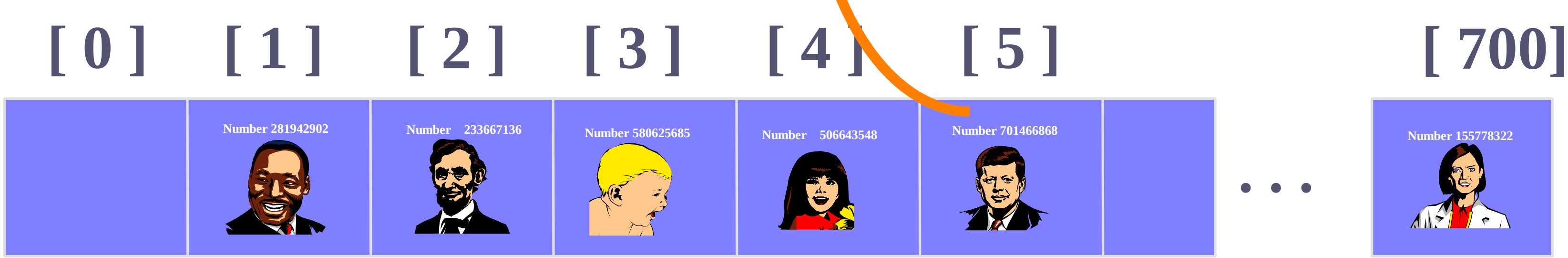
- Keep moving forward until you find the key, or you reach an empty spot.



Searching for a Key

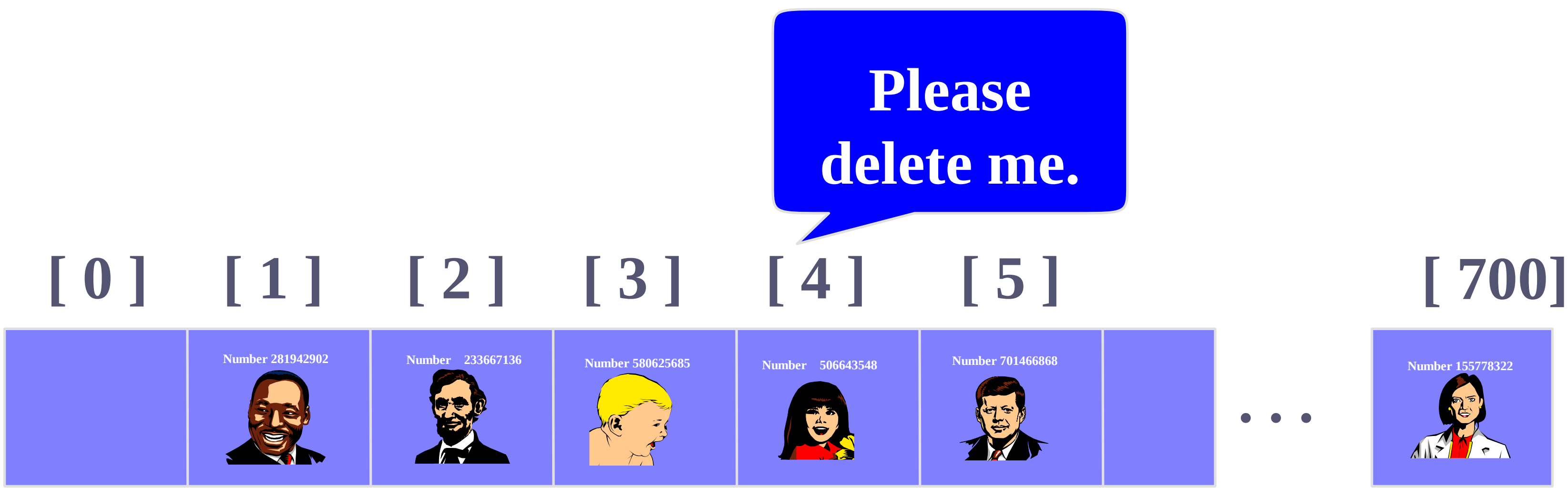
- When the item is found, the information can be copied to the necessary location.





Deleting a Record

- Records may also be deleted from a hash table.

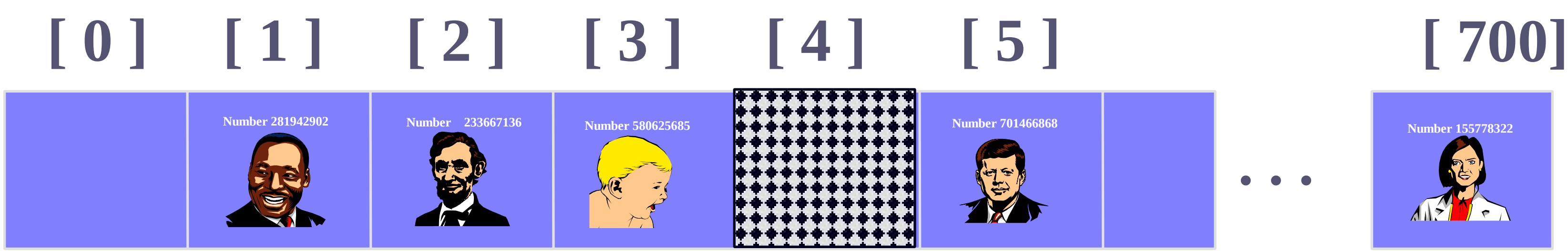


Deleting a Record

- 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 (Note: assuming open addressing/ linear probing).

Deleting a Record

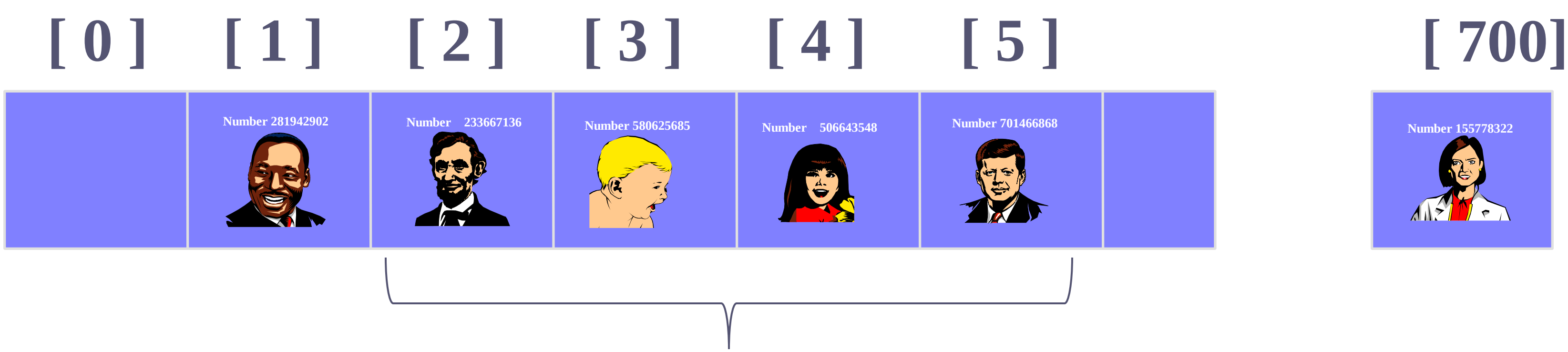
- Records may also be deleted from a hash table.
- But the location must not be left as an ordinary "empty spot" since that could interfere with searches.
- The location must be marked in some special way so that a search can tell that the spot used to have something in it.



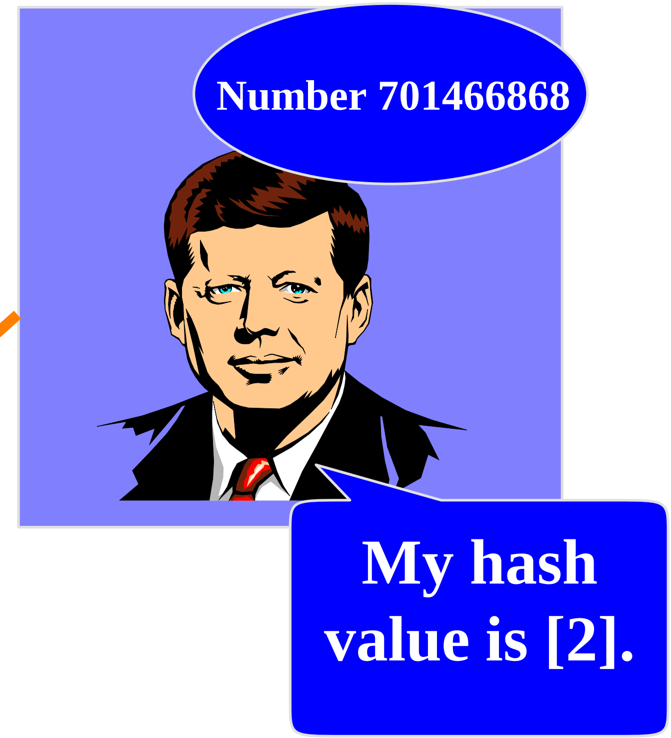
Problem with Linear Probing

- Clustering

My hash value is [2].



Open Addressing: Quadratic Probing



- **If collision occurs** ←

check H(x) + 1 else

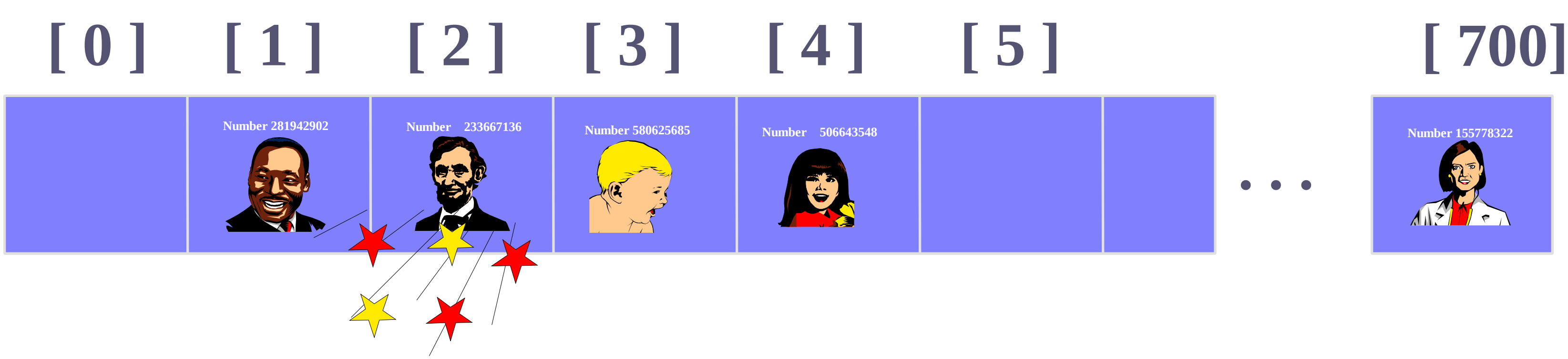
H(x) + 4 else

H(x) + 9 else

H(x) + 16

...

$H(Key)=(Key \bmod 701) + i^2$



Open Addressing: Quadratic Probing

- If collision occurs

check H(x) + 1 else ←

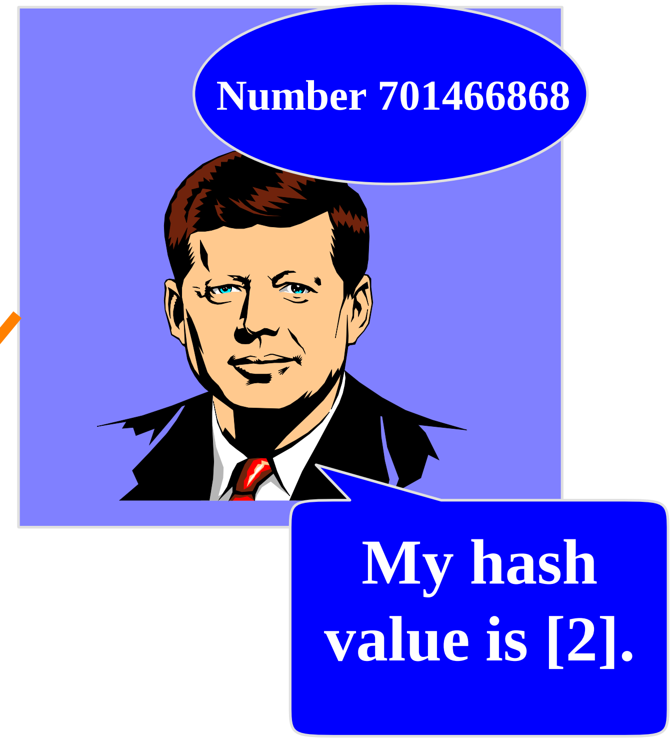
H(x) + 4 else

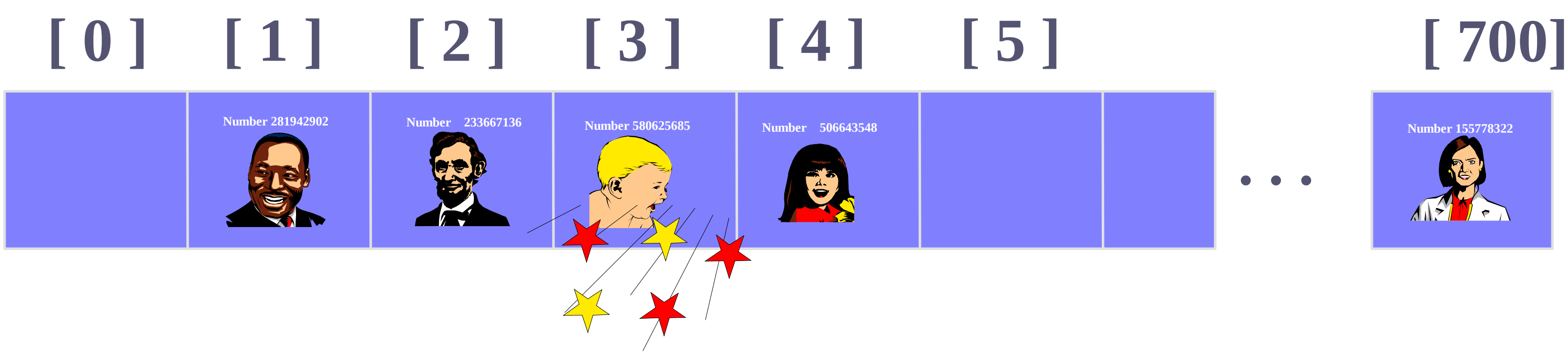
H(x) + 9 else

H(x) + 16

...

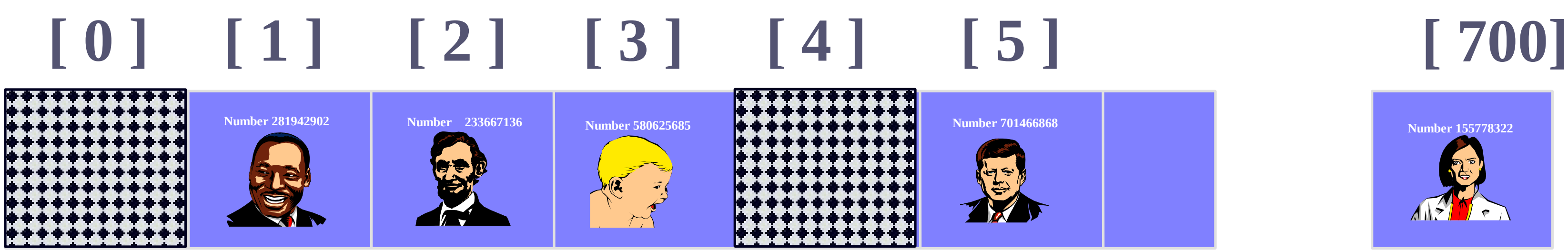
$H(Key)=(Key \bmod 701) + i^2$





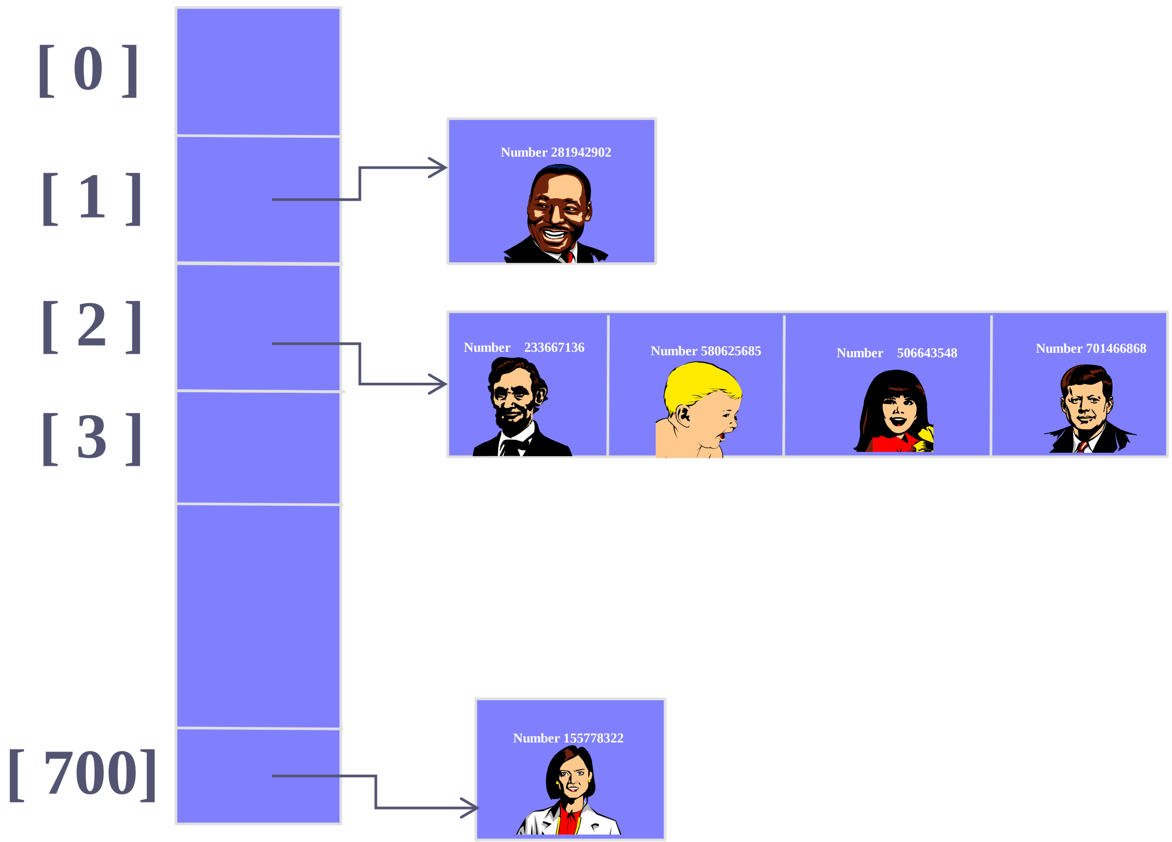
Problem with Open Addressing

- Table can become full with dead items



Separate Chaining

- A List of elements (bucket) that has same key



Analysis of Hashing with Chaining

- **Worst case**
  - All keys hash into the same bucket
  - a single linked list.
  - insert, delete, find take O(n) time.

- **Average case**
  - Keys are uniformly distributed into buckets
  - O(N/B): N is the number of elements in a hash table, B is the number of buckets.

Re-Hashing

- **If table gets too full, operations will take too long.**
- **Build another table, twice as big**
- **Insert every element again to this table**
- **Rehash after a percentage of the table becomes full (70%)**

Issues with Hashing

- ***What do we lose?***
  - **Operations that require ordering are inefficient**
  - **FindMax**: O(n) O(log n) Balanced binary tree
  - **FindMin**: O(n) O(log n) Balanced binary tree

- ***What do we gain?***
  - **Insert**: O(1) O(log n) Balanced binary tree
  - **Delete**: O(1) O(log n) Balanced binary tree
  - **Find**: O(1) O(log n) Balanced binary tree