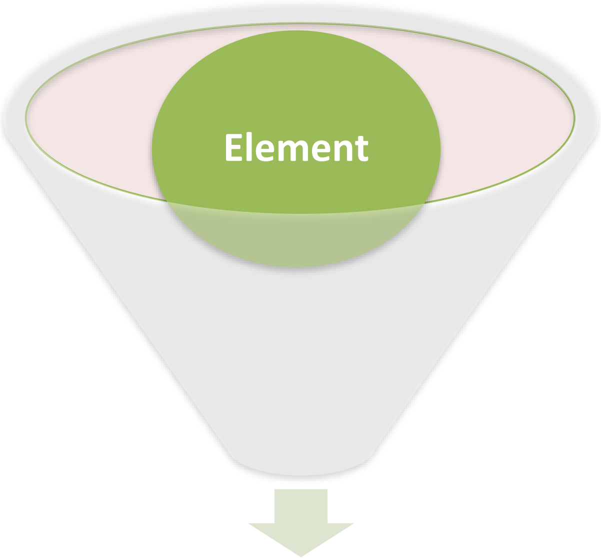
Hashing



Hash value

Hashing allows user to insert/delete/find records in constant average time.

Linear Search in **arrays and linked lists**

Search : 70

20	**40**	**50**	**70**	**80**	**90**	**10**
0	1	2	3	4	5	6

Linear Search

- We need to search in a **linear fashion**, which is costly in practice.

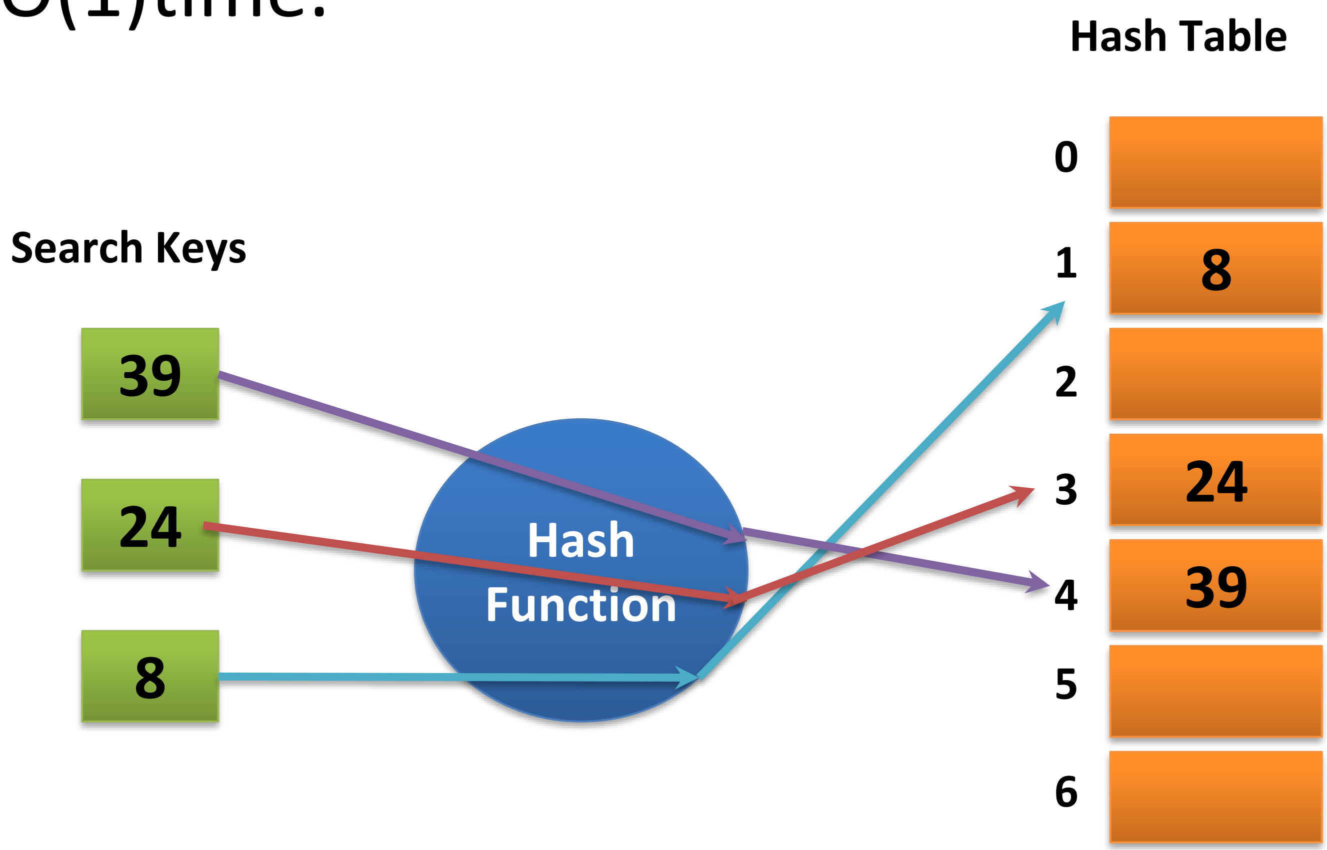
Time complexity of linear search is **O(N)**

Binary Search

If we keep the data sorted, then it can be searched in **O(Logn)** time using Binary Search, but insert and delete operations become costly as we have to maintain sorted order.

Hashing

- Efficient data structure that can be searched in O(1)time.




EFFICIENT
EFFICIENT
EFFICIENT


INEFFICIENT


MEDICAL
MANUALS
PETS
HOUSE
PERSONAL
AUTO
A BOWL FULL OF LEMONS
Life. Organized.

Hashing

- **Hashing** is the process of mapping the data item to a **hash table** with the help of a **hashing function**.
- A **hash table** is a collection of items which are stored in such a way as to make it easy to find them later.
- **hash function** to compute an index so that a data can be stored at a specific location in a table such that it can easily be found.
- With hashing we get **O(1) search time** on average (under reasonable assumptions) and O(n) in worst case.

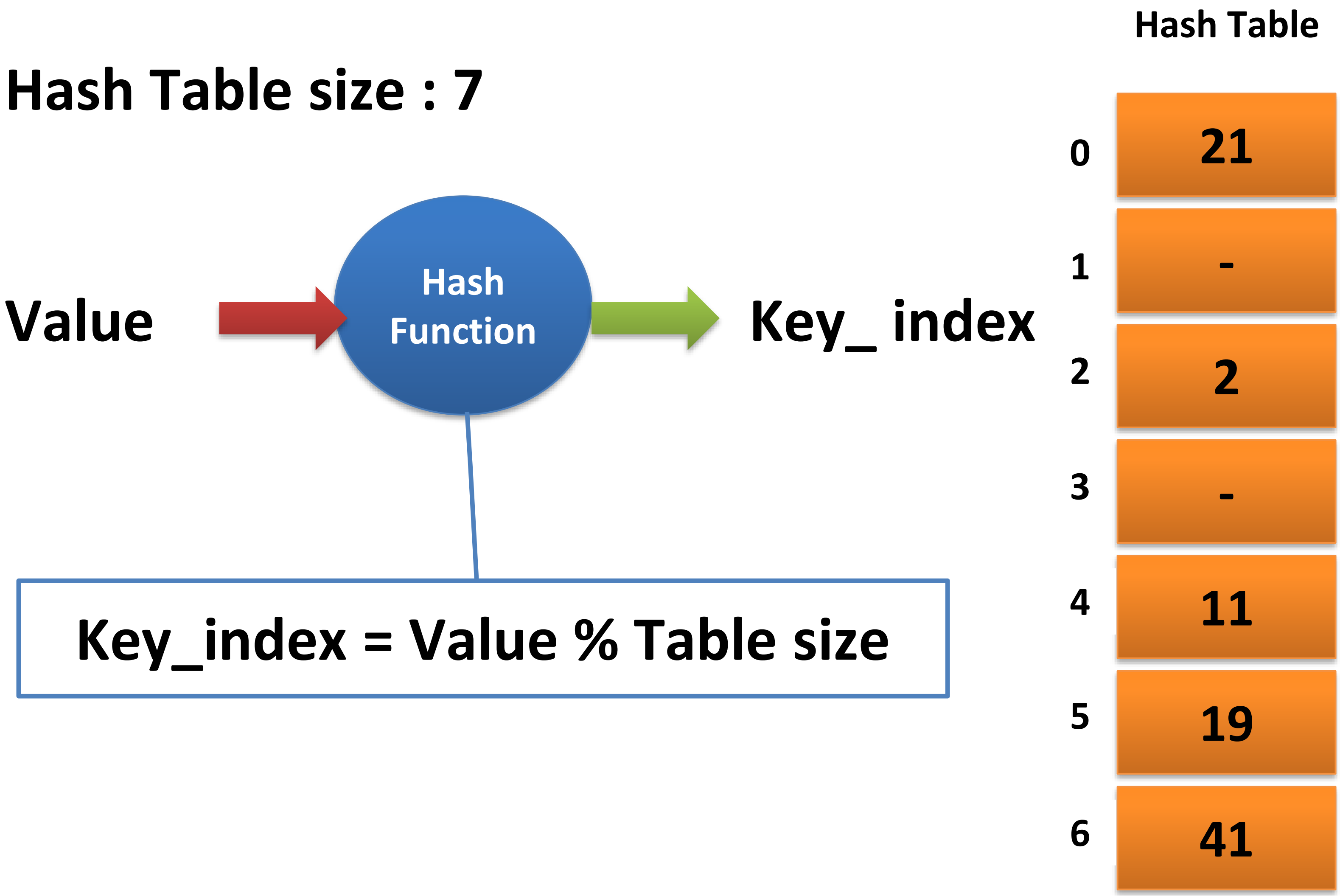
Hashing - Operations

O(1)

on average case

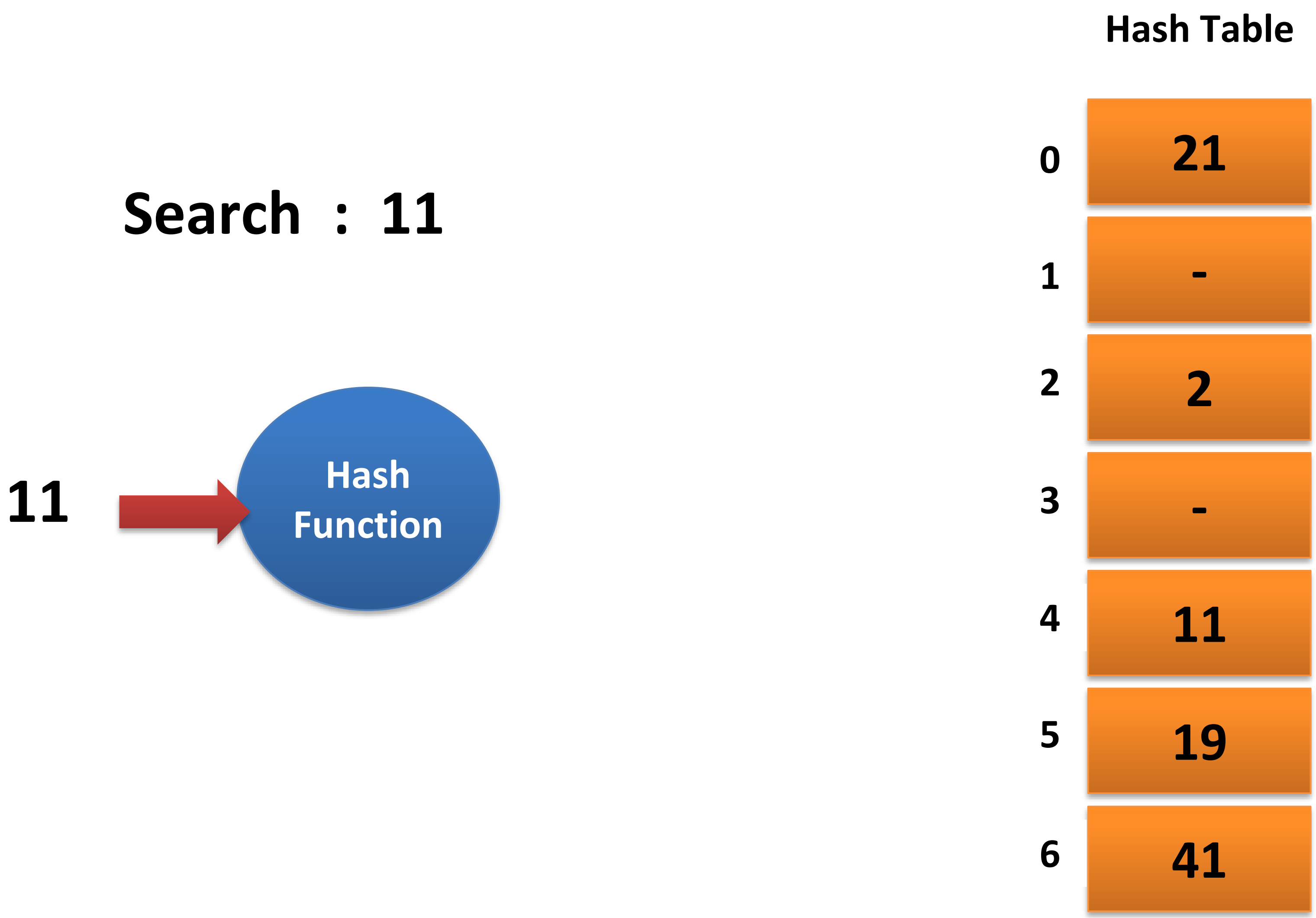
- Search
- Insert
- Delete

Hash Function



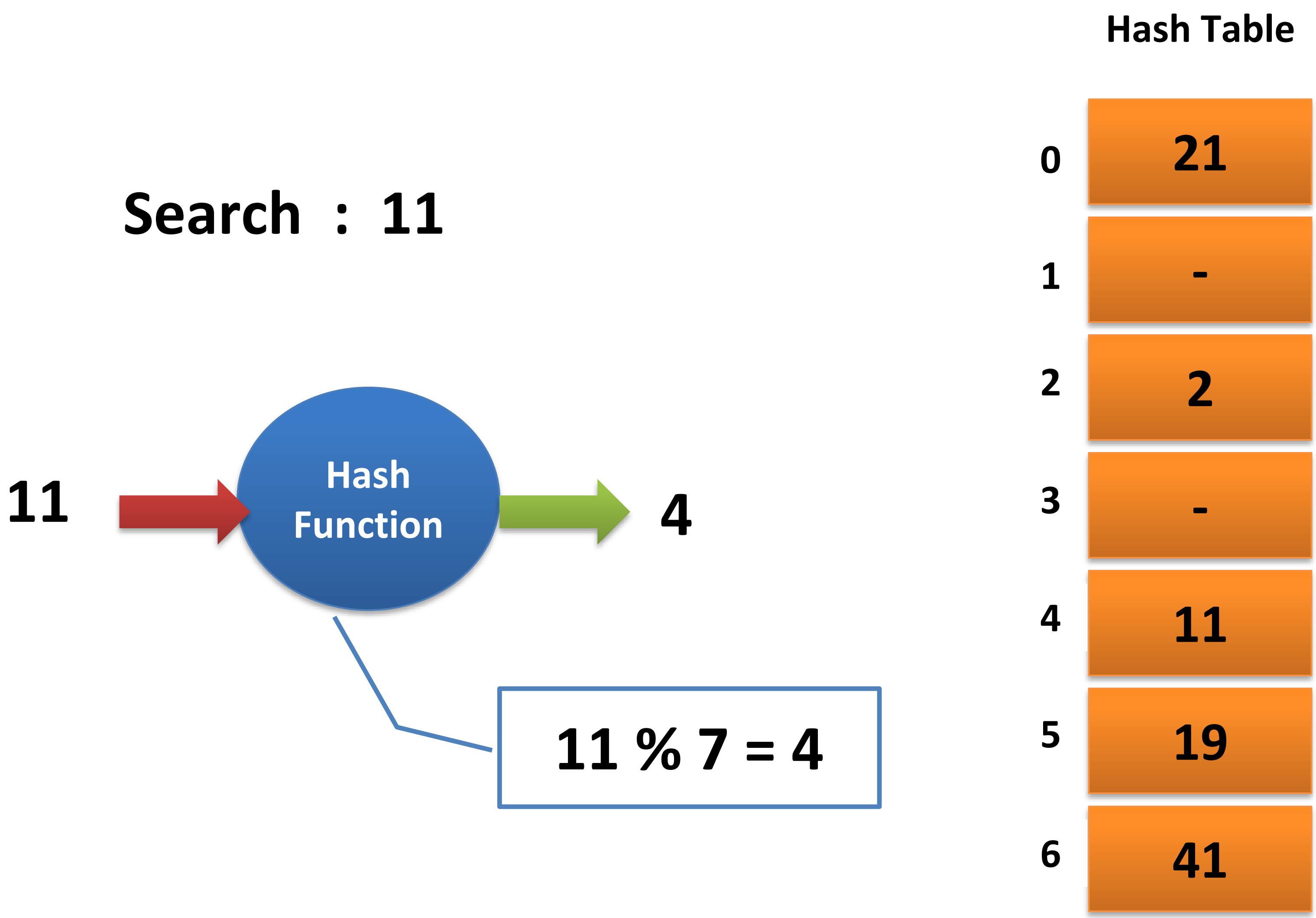
Hash Table

Search : 11

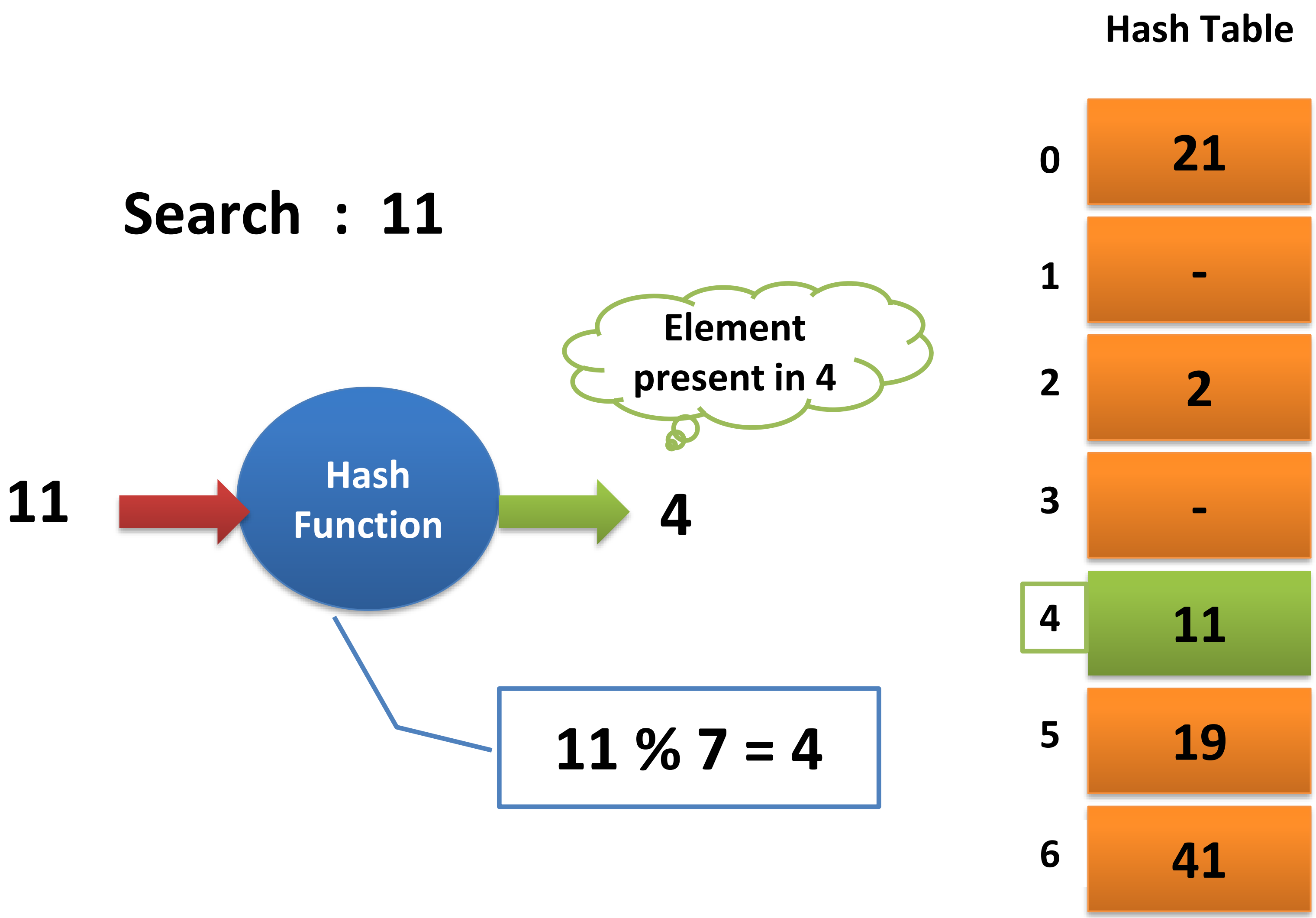


Hash Table

Search : 11

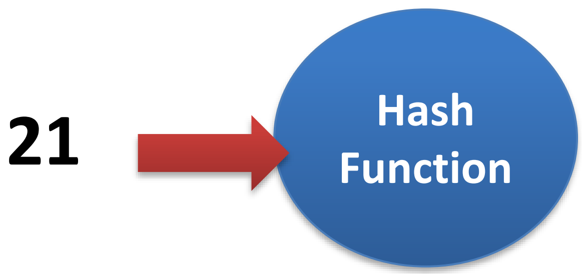


Hash Table


Hash Table
Search : 11
0
21
1
-
2
2
3
-
4
11
5
19
6
41
Element present in 4
11
Hash Function
4
11 % 7 = 4

Hash Table

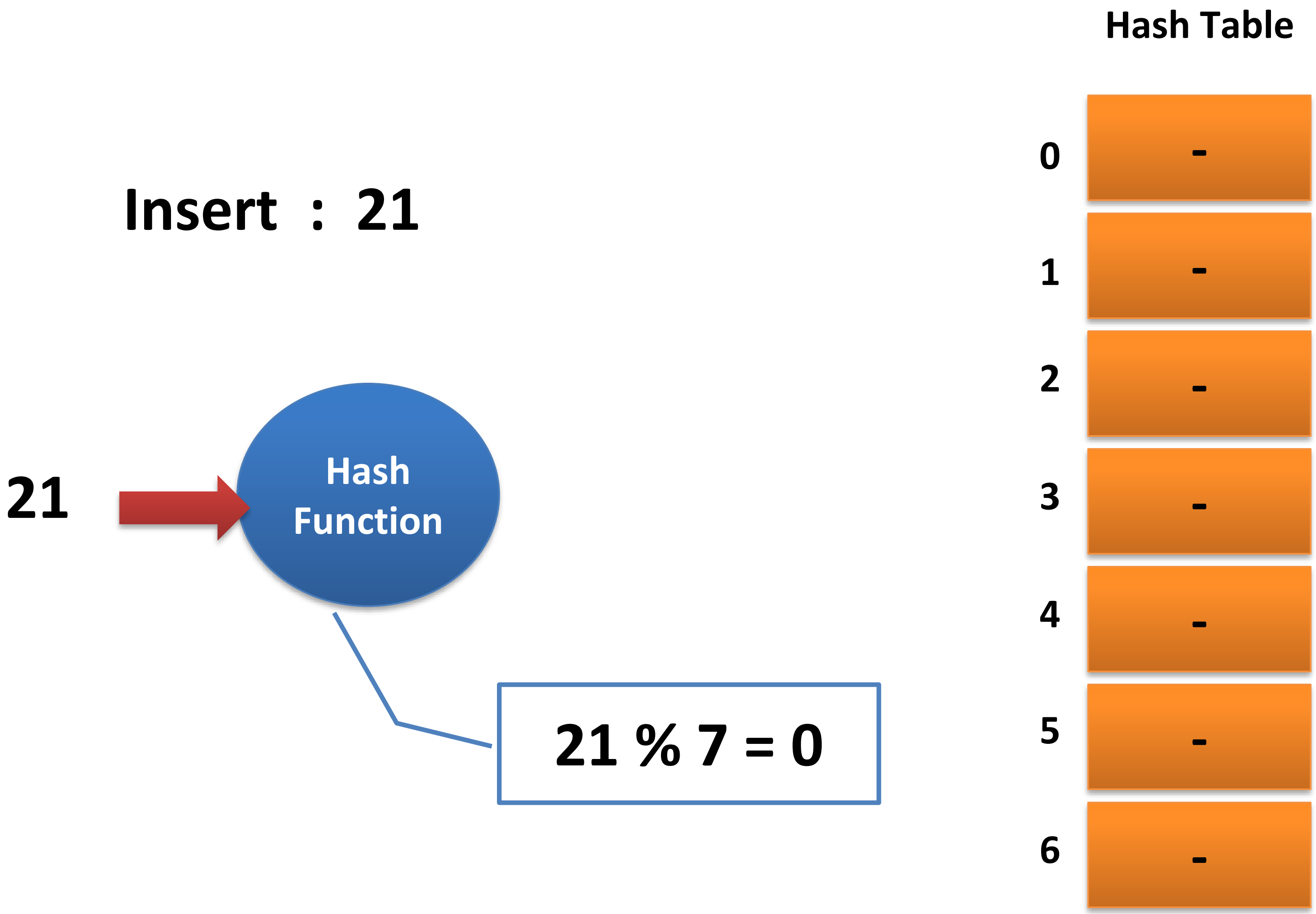
Insert : 21



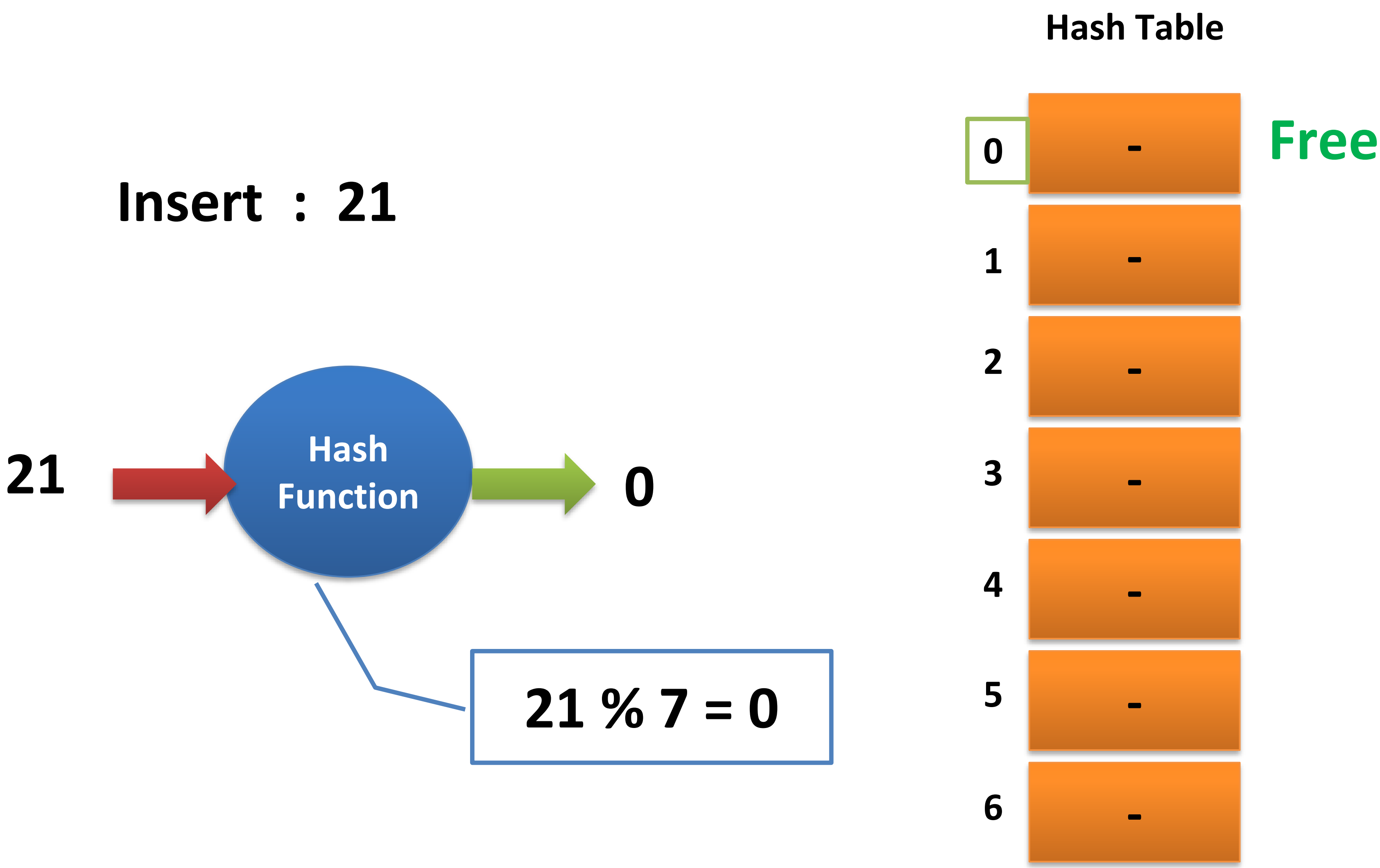
Hash Table

0	-
1	-
2	-
3	-
4	-
5	-
6	-

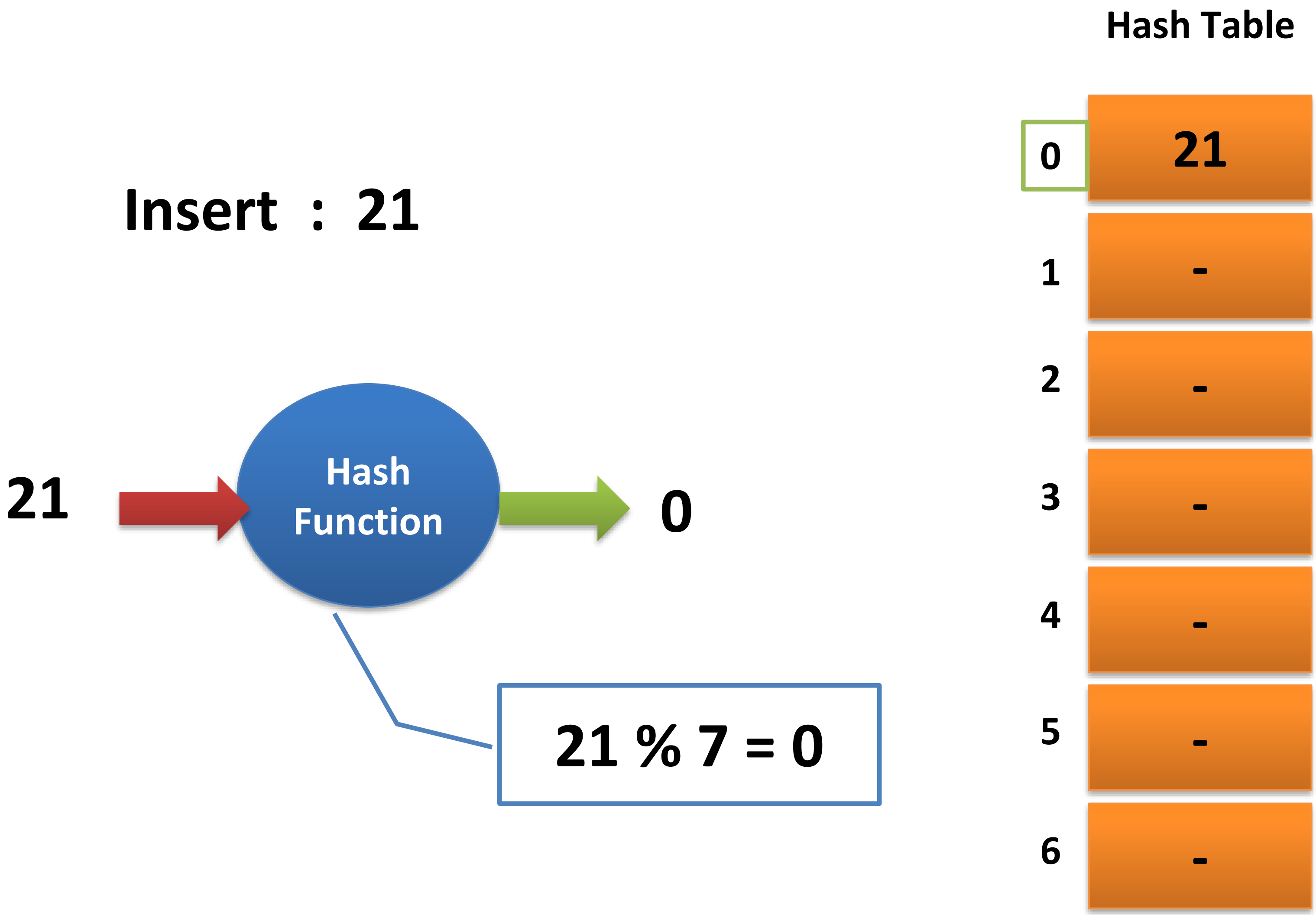
Hash Table



Hash Table

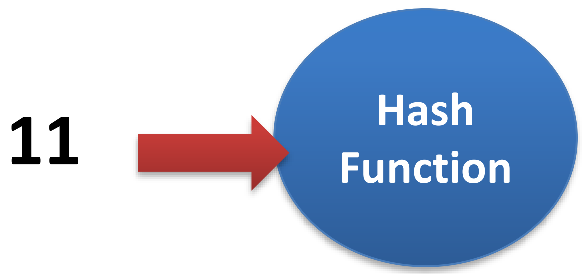


Hash Table



Hash Table

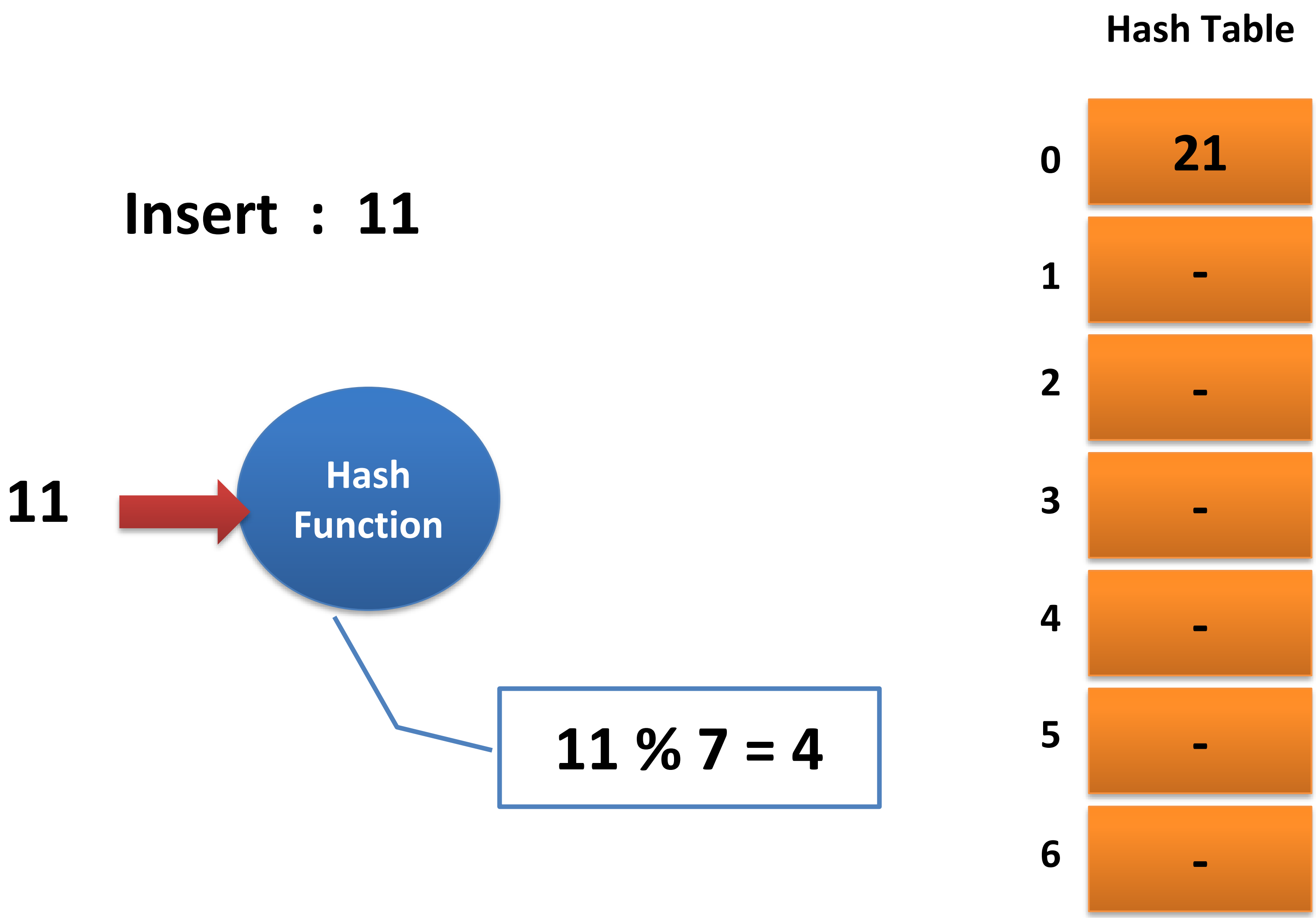
Insert : 11



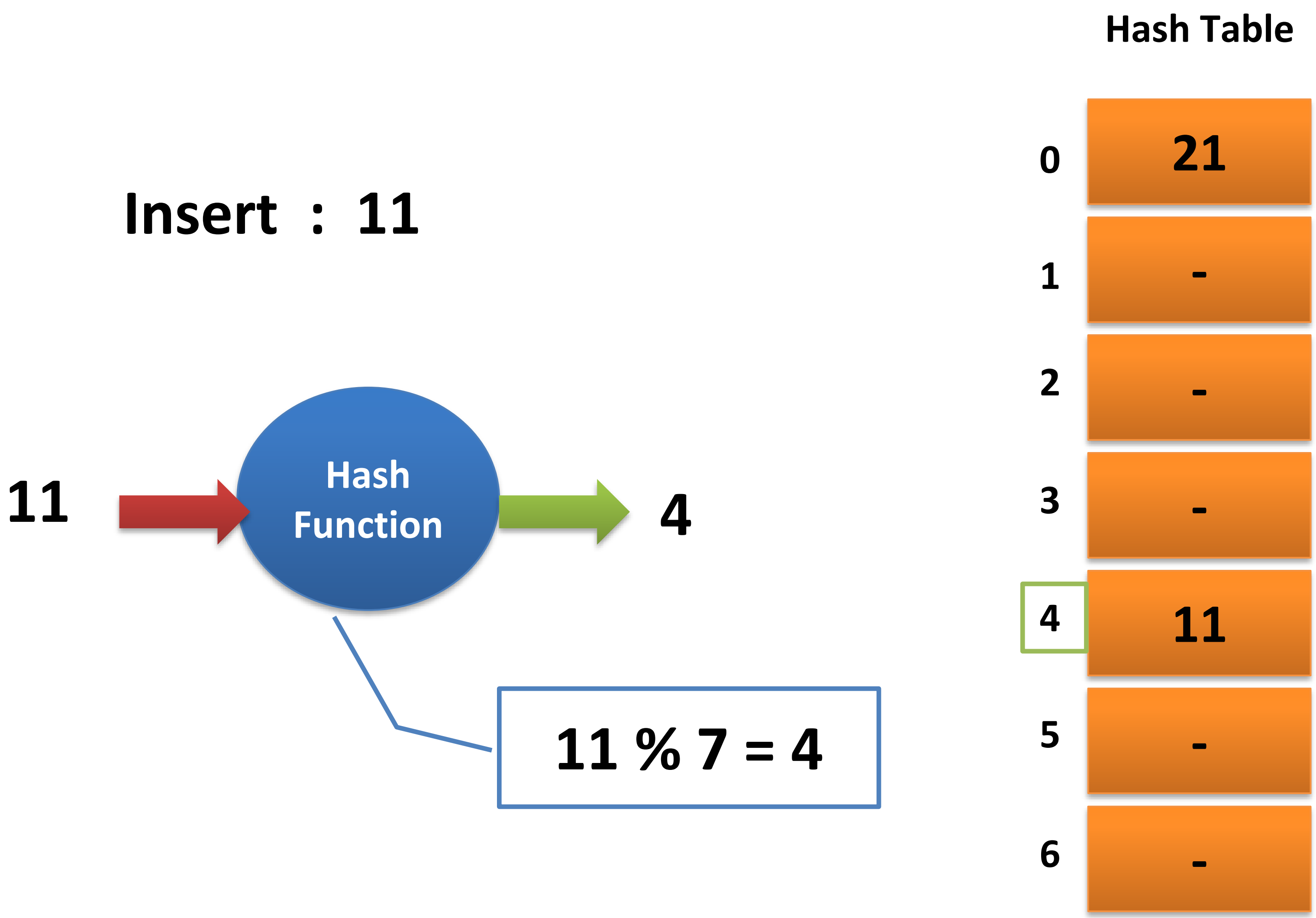
Hash Table

0	21
1	-
2	-
3	-
4	-
5	-
6	-

Hash Table



Hash Table



Hash Table

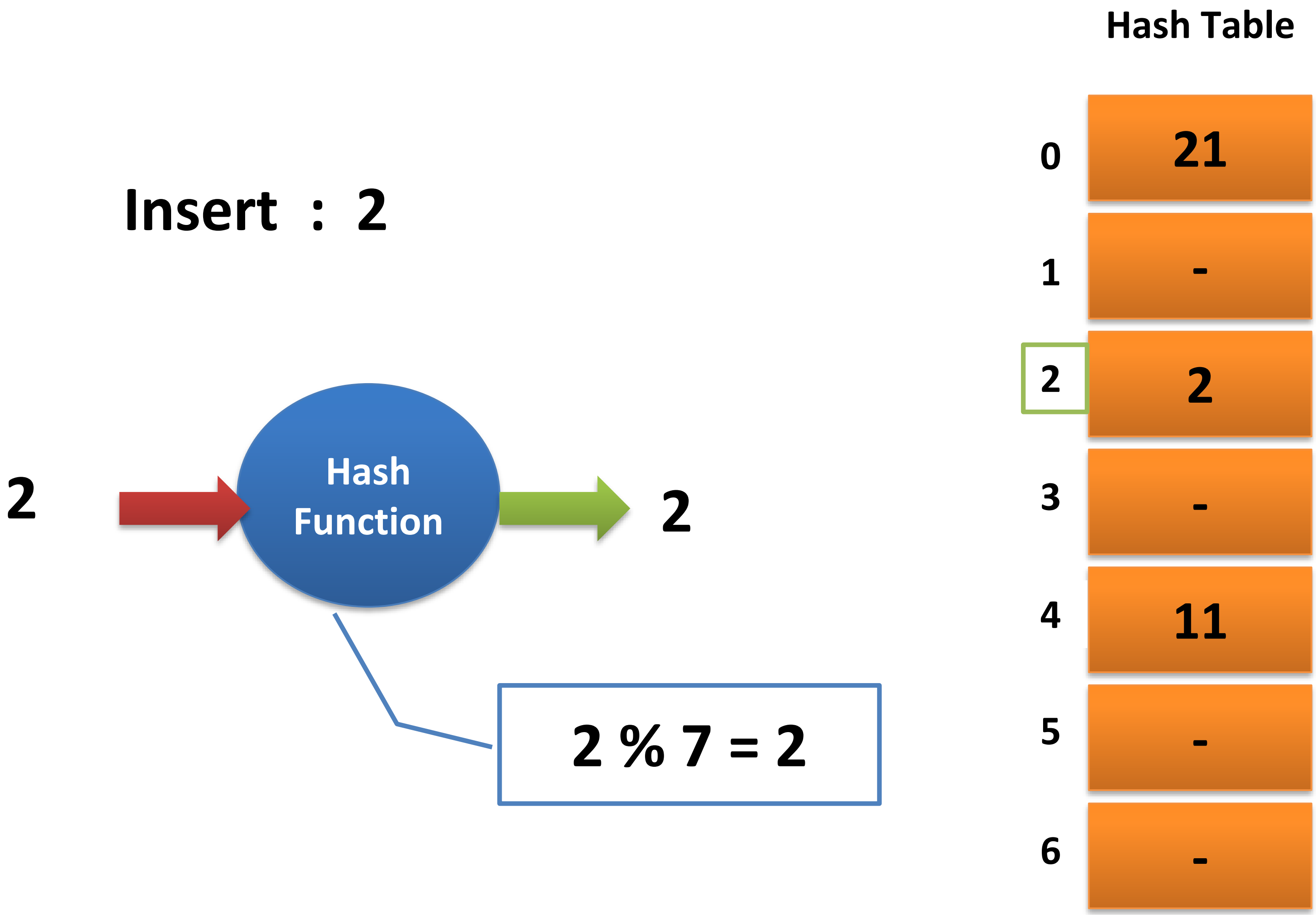
Insert : 2

Hash Table

0	21
1	-
2	-
3	-
4	11
5	-
6	-

Hash Table

Insert : 2



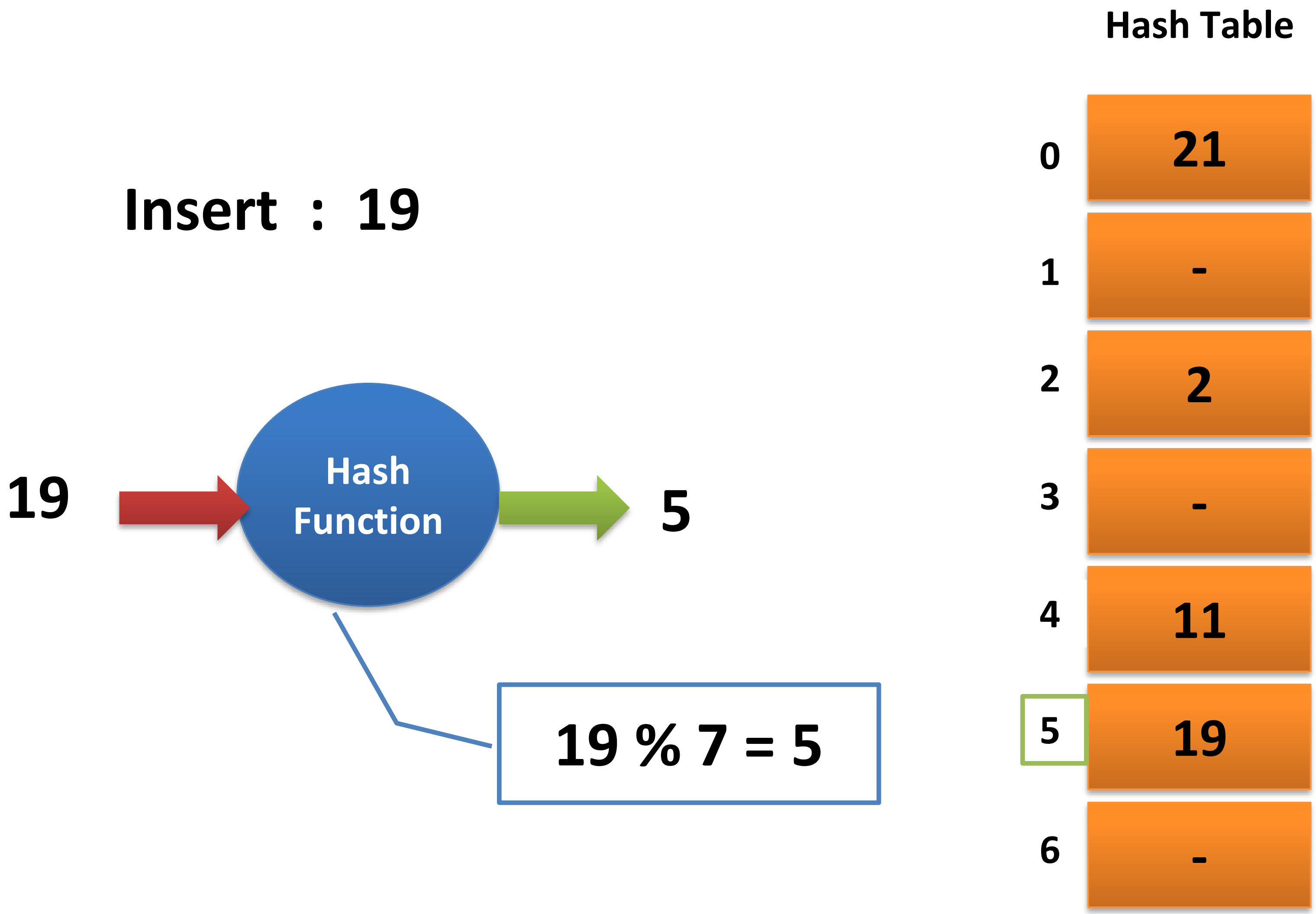
Hash Table

Insert : 19

Hash Table

0	21
1	-
2	2
3	-
4	11
5	-
6	-

Hash Table

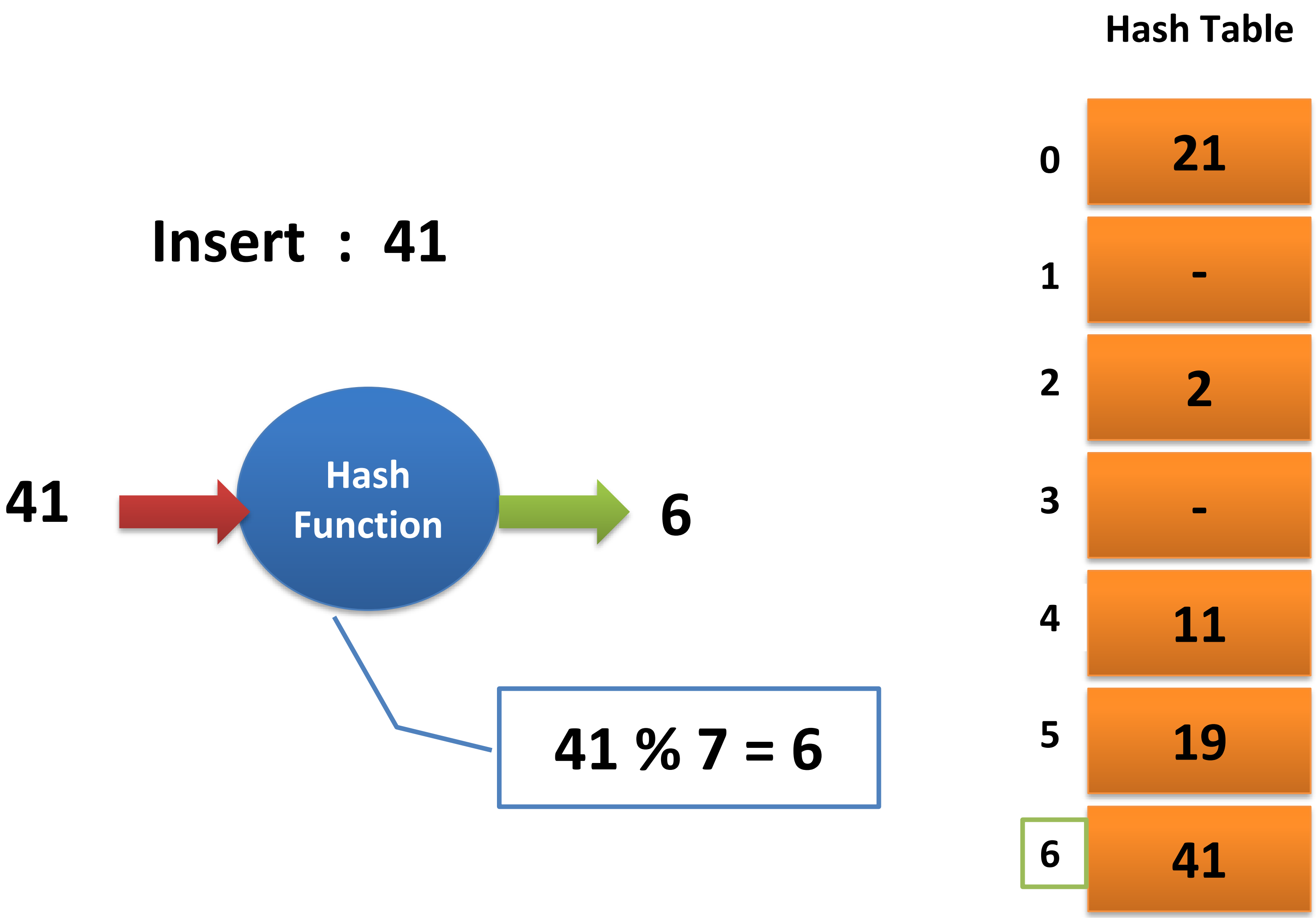

Hash Table
Insert : 19
19
Hash Function
5
19 % 7 = 5
0
21
1
-
2
2
3
-
4
11
5
19
6
-

Hash Table

Insert : 41

	Hash Table
0	21
1	-
2	2
3	-
4	11
5	19
6	-

Hash Table



Hash Table
Insert : 41
41
Hash Function
6
41 % 7 = 6
0
21
1
-
2
2
3
-
4
11
5
19
6
41

Hash Table

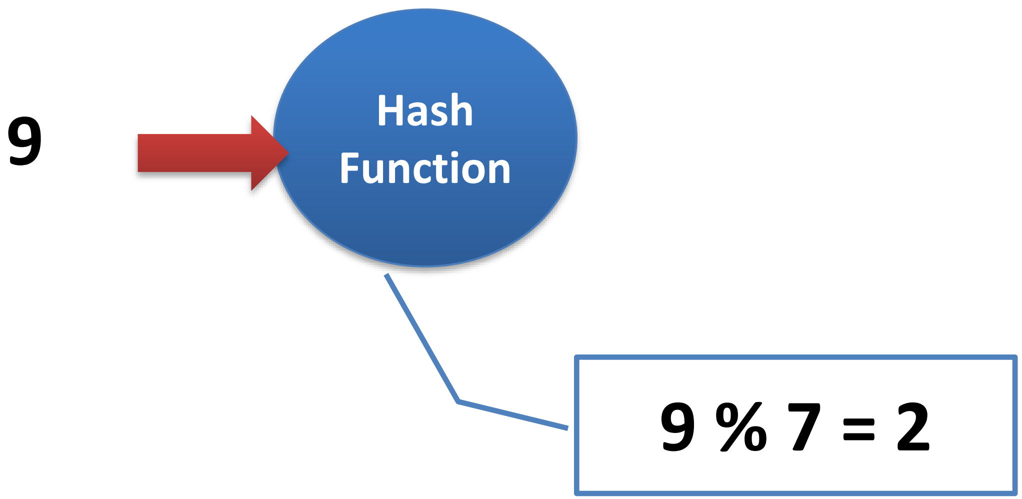
Load factor = $\dfrac{\textbf{Number of slots are now occupied}}{\textbf{Table size}}$

= 5 / 7

	Hash Table
0	21
1	-
2	2
3	-
4	11
5	19
6	41

linear probing

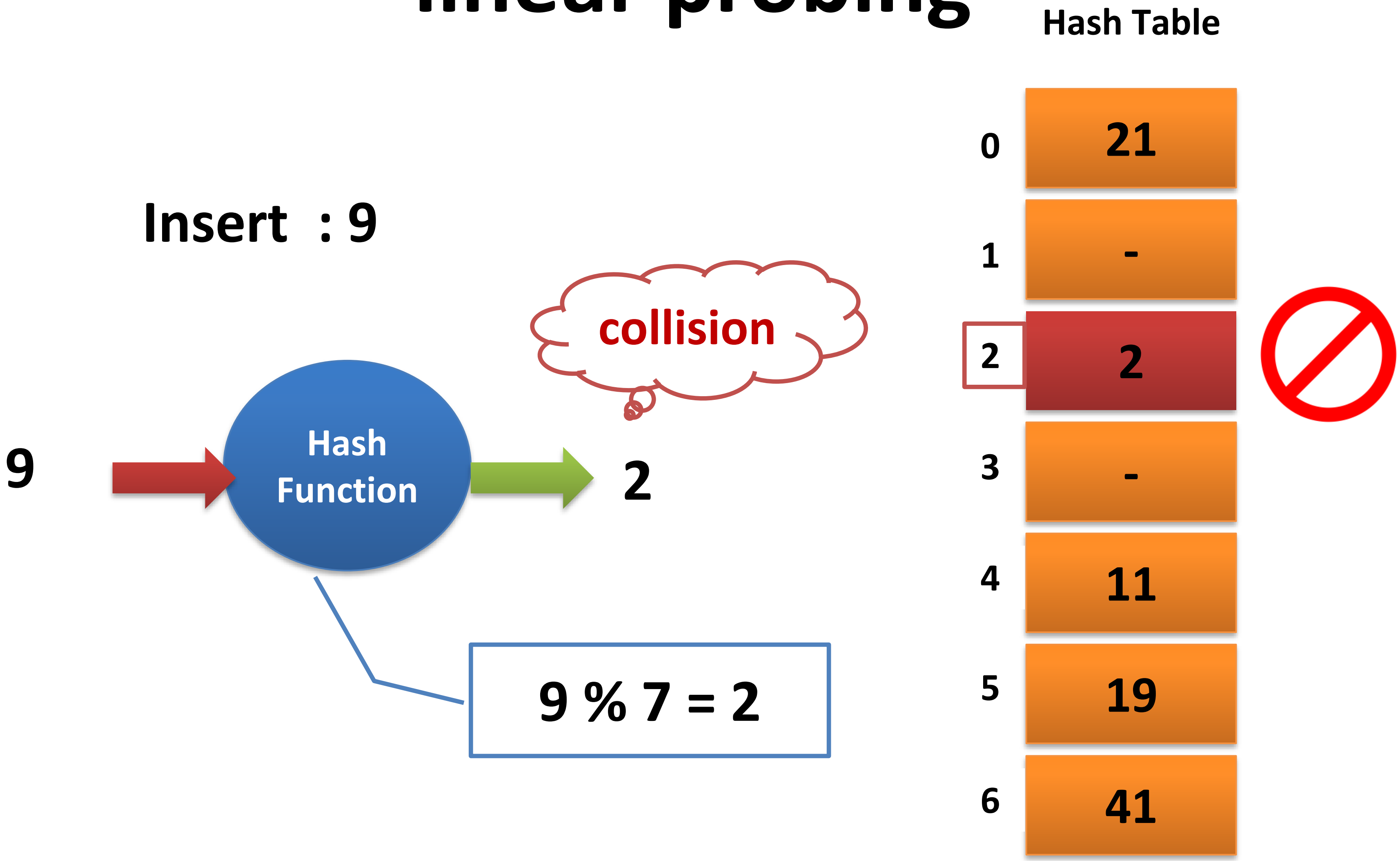
Insert : 9



Hash Table

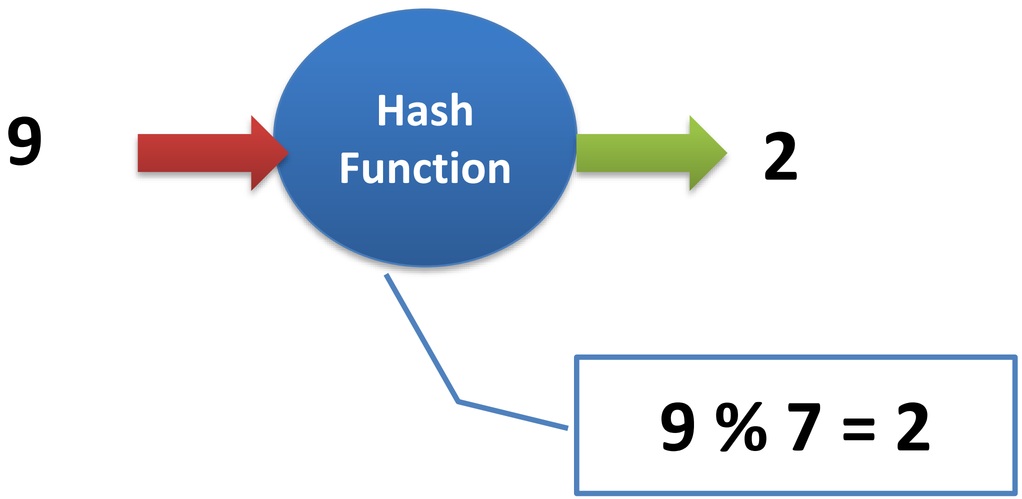
0	21
1	-
2	2
3	-
4	11
5	19
6	41

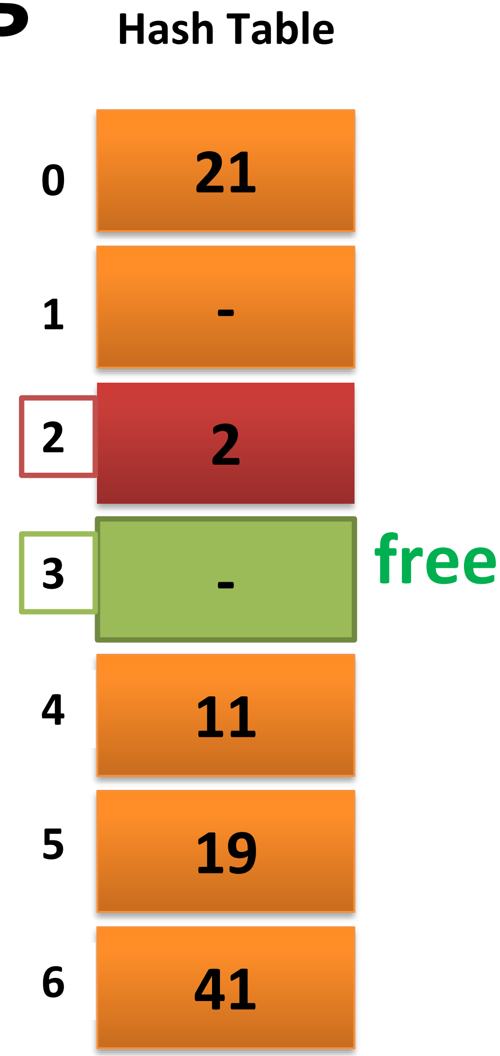
linear probing



linear probing

Insert : 9

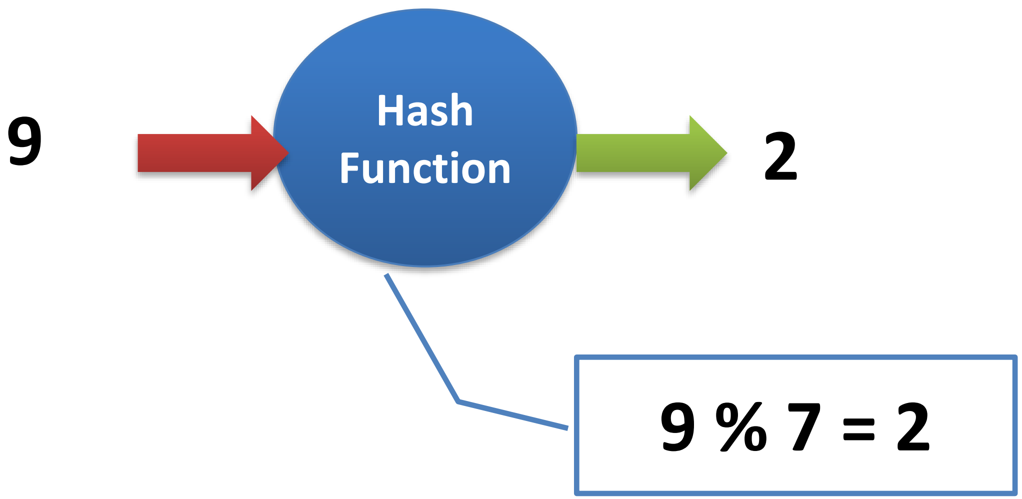




looking for the next available location i + 1 = 2+1

linear probing

Insert : 9



Hash Table

0	21
1	-
2	2
3	9
4	11
5	19
6	41

next location 2+1 =3 is free. Insert 9 at location 3.

Collision Resolution Techniques

- Collision: hash(x) = hash(y) for two different keys x and y.

- Collision Resolution techniques:
  - Closed Hashing or Open Addressing
    - Linear Probing
    - Quadratic Probing
    - Double Hashing
  - Open Hashing:
    - Separate Chaining

Open Addressing

- Linear Probing:
  - (hash(key) + 1) % hashTableSize, (hash(key) + 2) % hashTableSize, etc
- Quadratic Probing:
  - (hash(key) + 1^2) % hashTableSize, (hash(key) + 2^2) % hashTableSize, etc
- Double Hashing:
  - (hash1(key) + i * hash2(key)) % hashTableSize

Search - Procedure

1. **Find the hash value of the element to be found.**

   position = **Element** % table_size

2. **If** the **hash_table[position] == Element** then,

   element is found in the position.

   Else

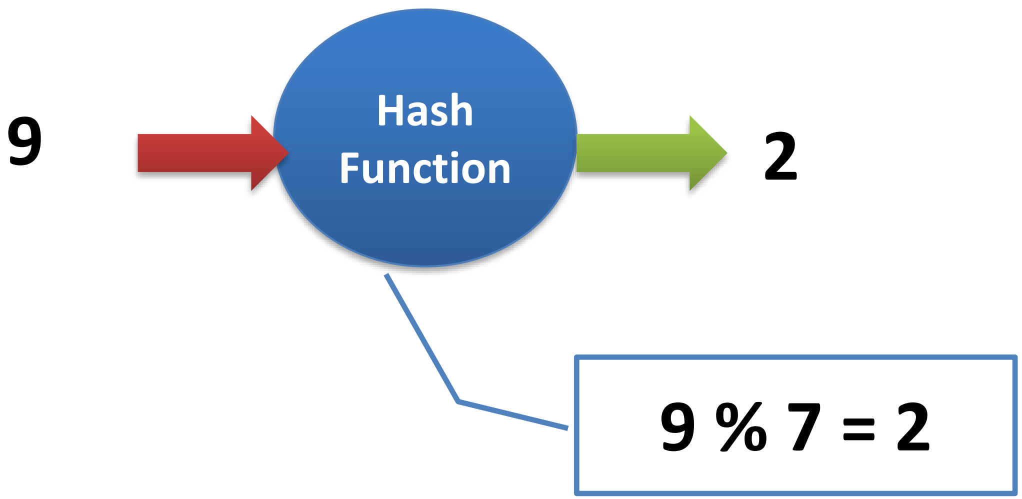
   Step through an array one item at a time looking for a desired item.

   The search **stops** when the item **is found or** when it **find any null value or** when the search has **examined each item without success.**

Hashing

search : 9

1. Find the hash value of the element to be found.



Hash Table

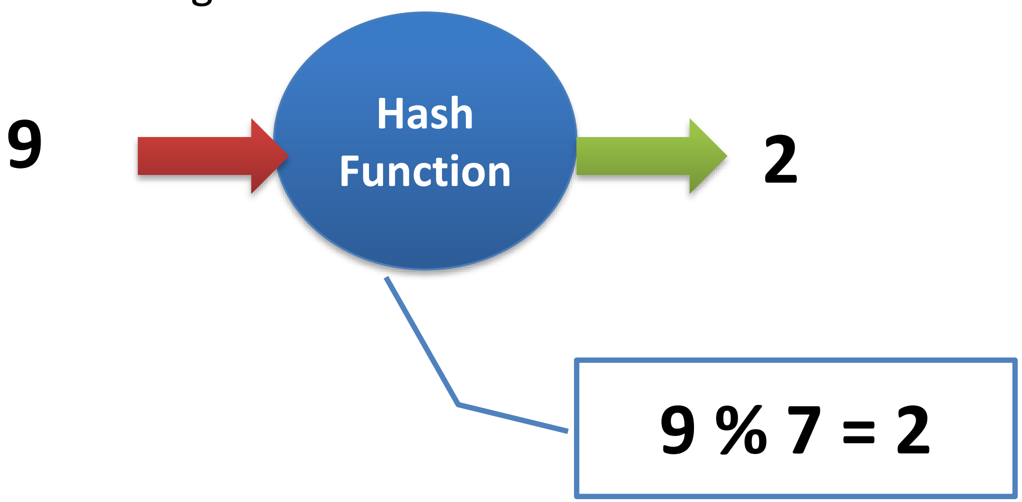
0	21
1	-
2	2
3	9
4	11
5	19
6	41

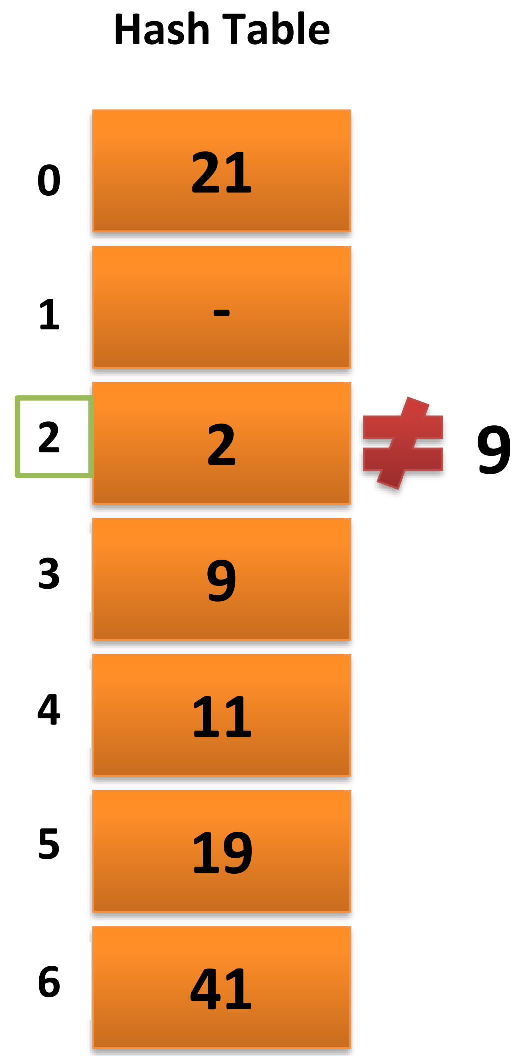
Hashing

search : 9

If (**hash_table[position] != Element**)then

Step through an array one item at a time looking for a desired item.



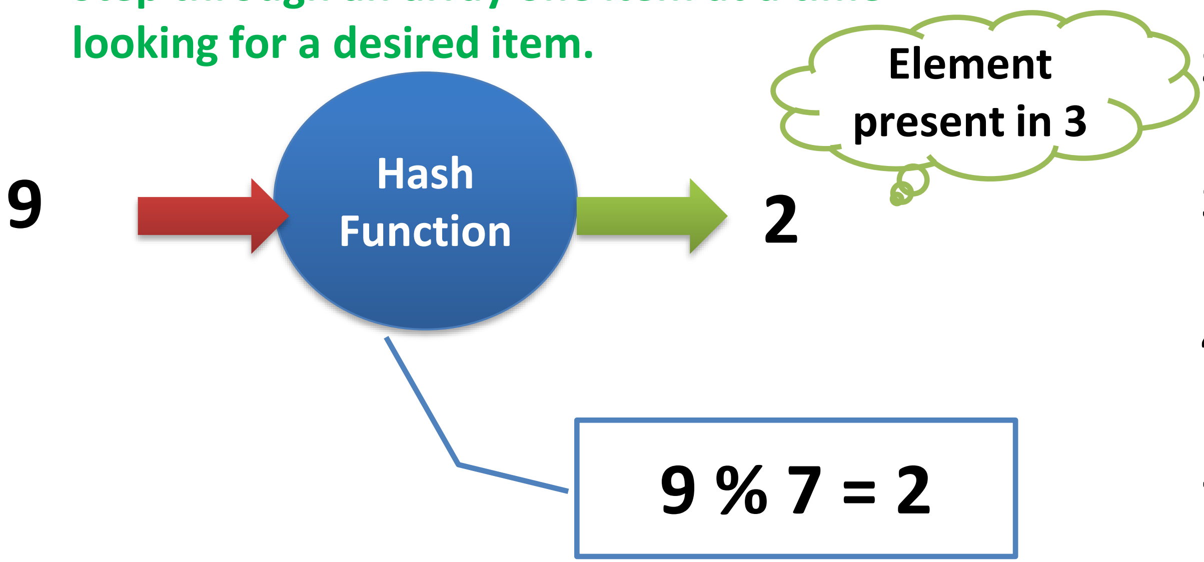


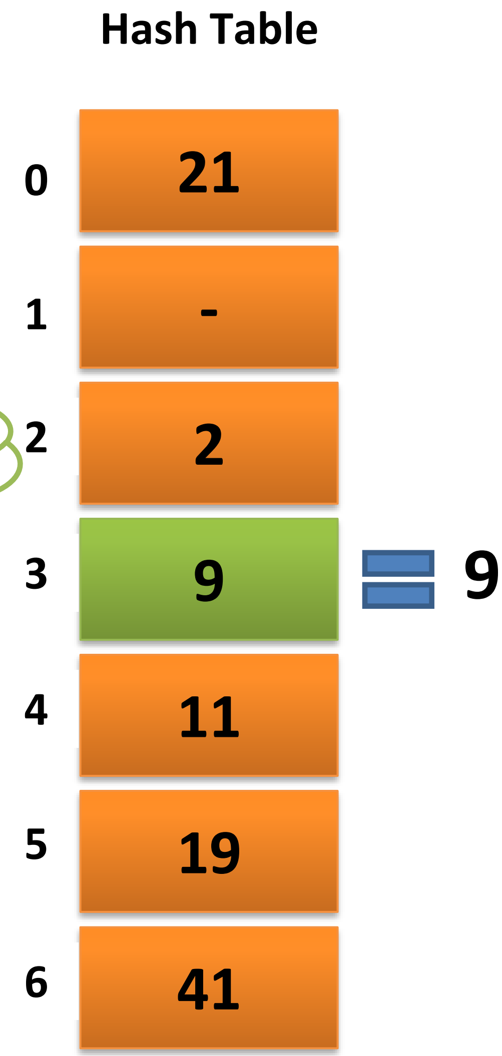
Hashing

search : 9

If (hash_table[position] != Element) then

Step through an array one item at a time looking for a desired item.

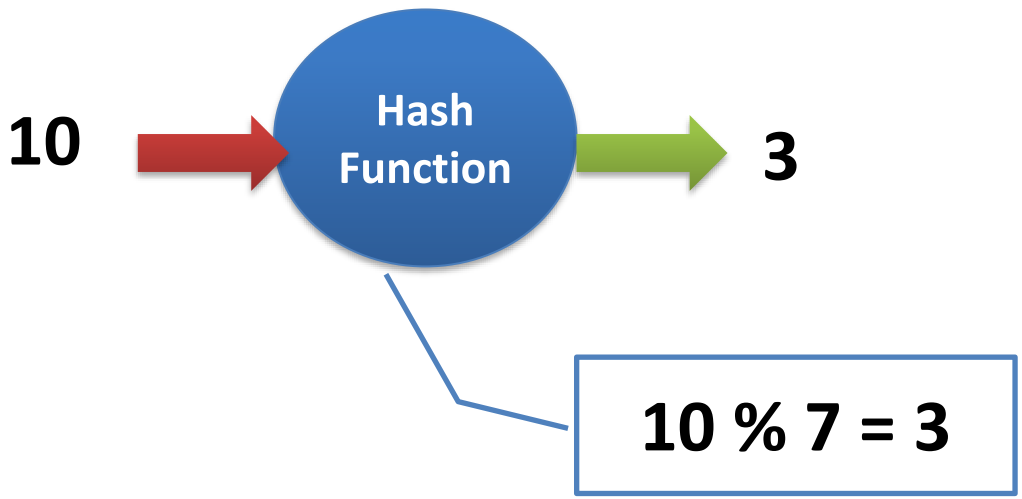


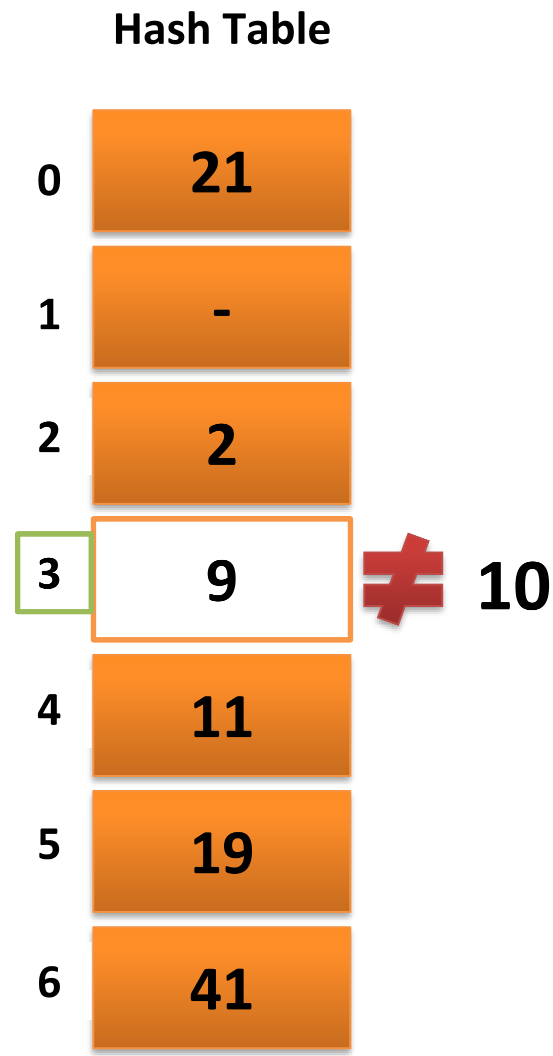


Hashing

search : 10

1. Find the hash value of the element to be found.



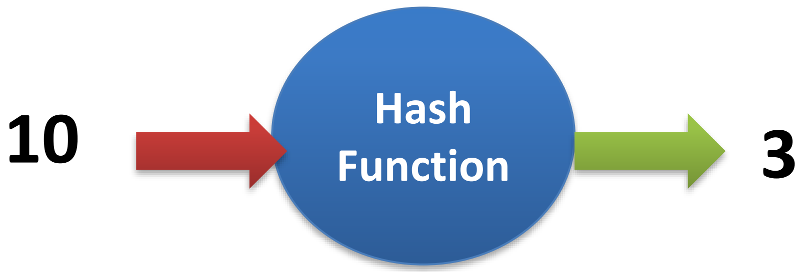


2. If (hash_table[position] != Element)then

   Step through an array one item at a time looking for a desired item.

Hashing

search : 10



Hash Table

0	21	
1	-	
2	2	
3	9	
4	11	≠ 10
5	19	
6	41	

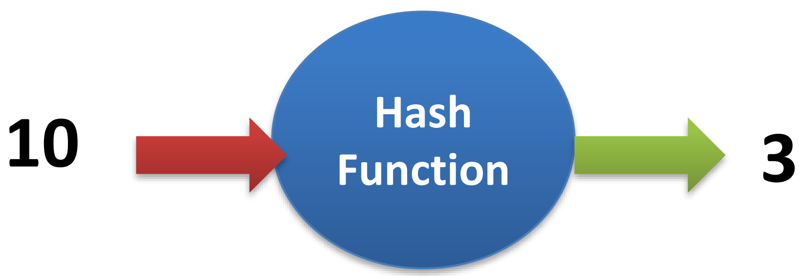
Step through an array one item at a time looking for a desired item.

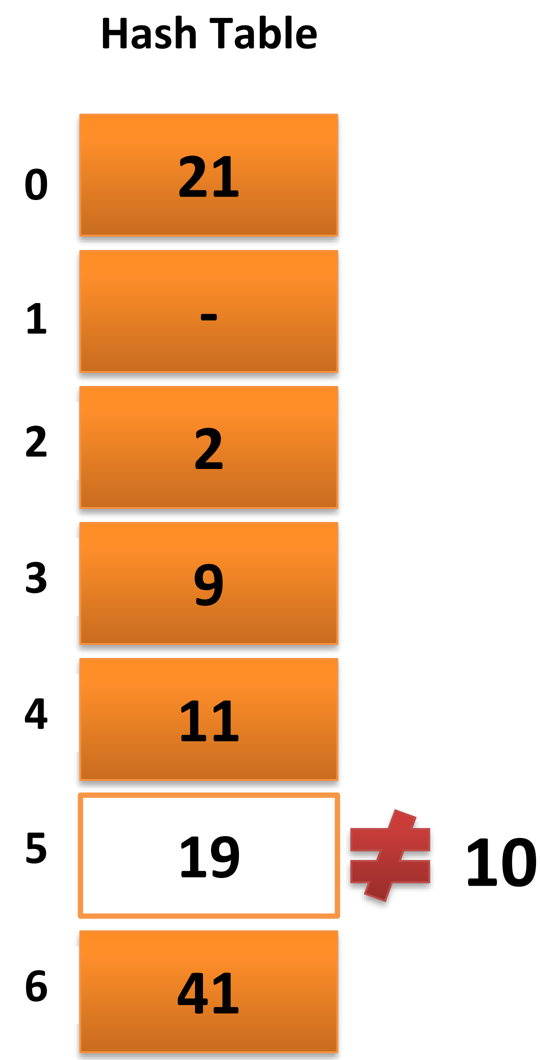
looking for the next slot 3 + 1 = 4

The search **stops** when the item **is found or** when it **find any null value or** when the search has **examined each item without success.**

Hashing

search : 10





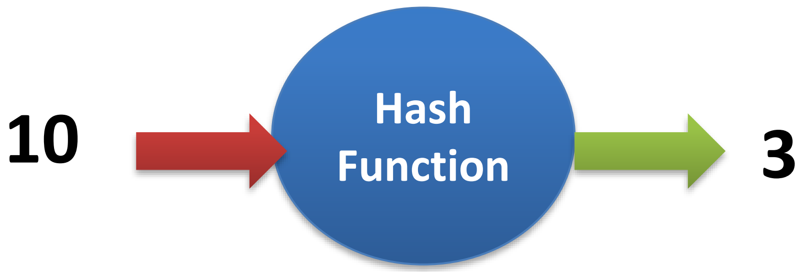
Step through an array one item at a time looking for a desired item.

looking for the next slot 4 + 1 = 5

The search **stops** when the item **is found or** when it **find any null value or** when the search has **examined each item without success.**

Hashing

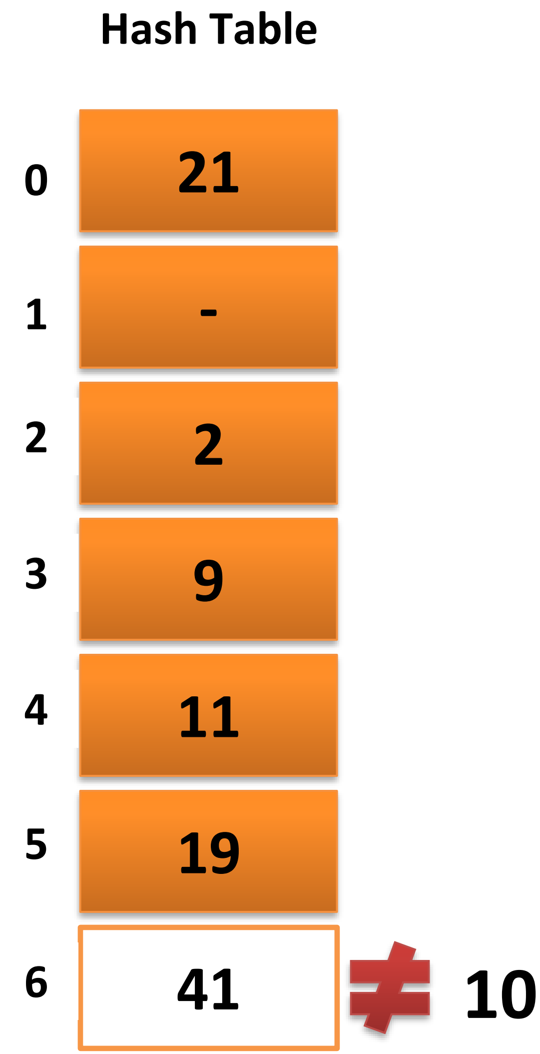
search : 10



Step through an array one item at a time looking for a desired item.

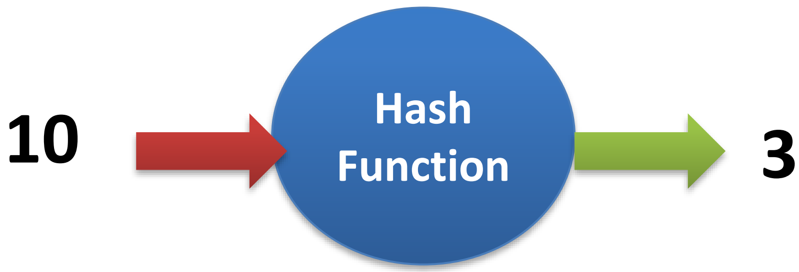
looking for the next slot 5 + 1 = 6

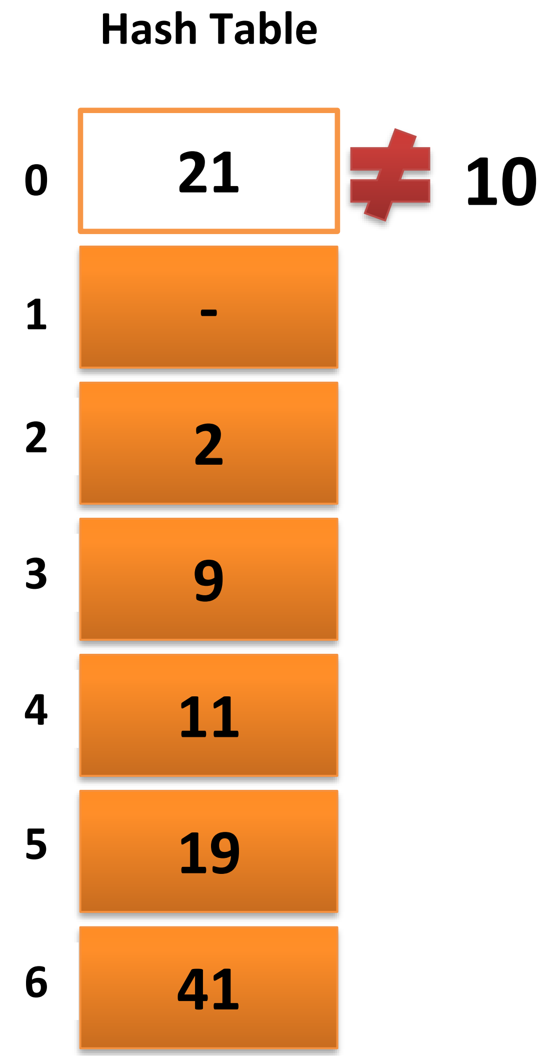
The search **stops** when the item **is found** **or** when it **find any null value** **or** when the search has **examined each item without success.**



Hashing

search : 10





Step through an array one item at a time looking for a desired item.

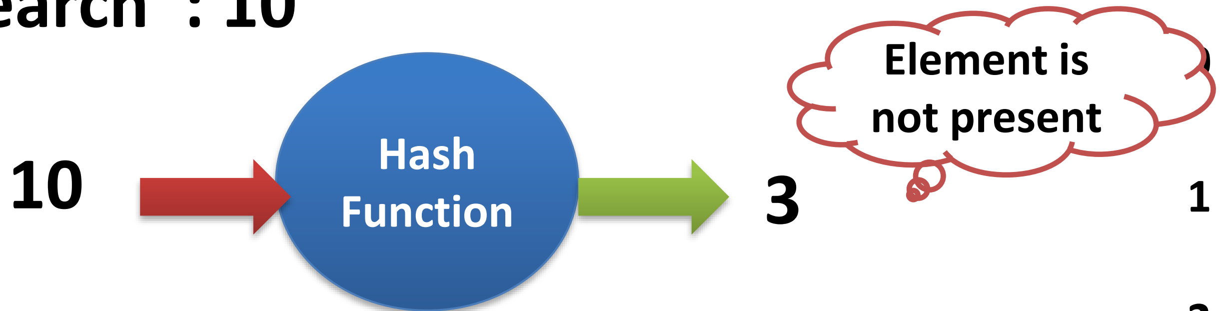
looking for the next slot 6 + 1 = 7

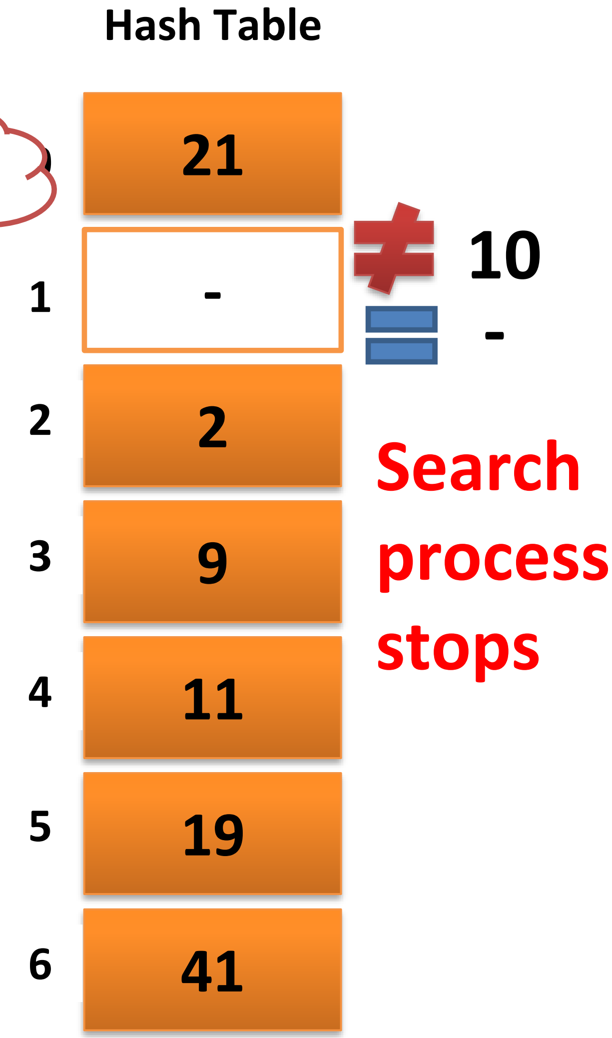
(6+1) mod 7 = 0

The search **stops** when the item **is found** **or** when it **find any null value** **or** when the search has **examined each item without success.**

Hashing – unsuccessful search

search : 10





Step through an array one item at a time looking for a desired item.

The search **stops** when the item **is found** **or** when it **find any null value** **or** when the search has **examined each item without success.**

Quadratic Probing

Let hash(x) be the slot index computed using the hash function.

- If the slot hash(x) % *S* is full, then we try (hash(x) + 1*1) % S.
- If (hash(x) + 1*1) % S is also full, then we try (hash(x) + 2*2) % S.
- If (hash(x) + 2*2) % S is also full, then we try (hash(x) + 3*3) % S.
- This process is repeated for all the values of i until an empty slot is found.

Separate chaining

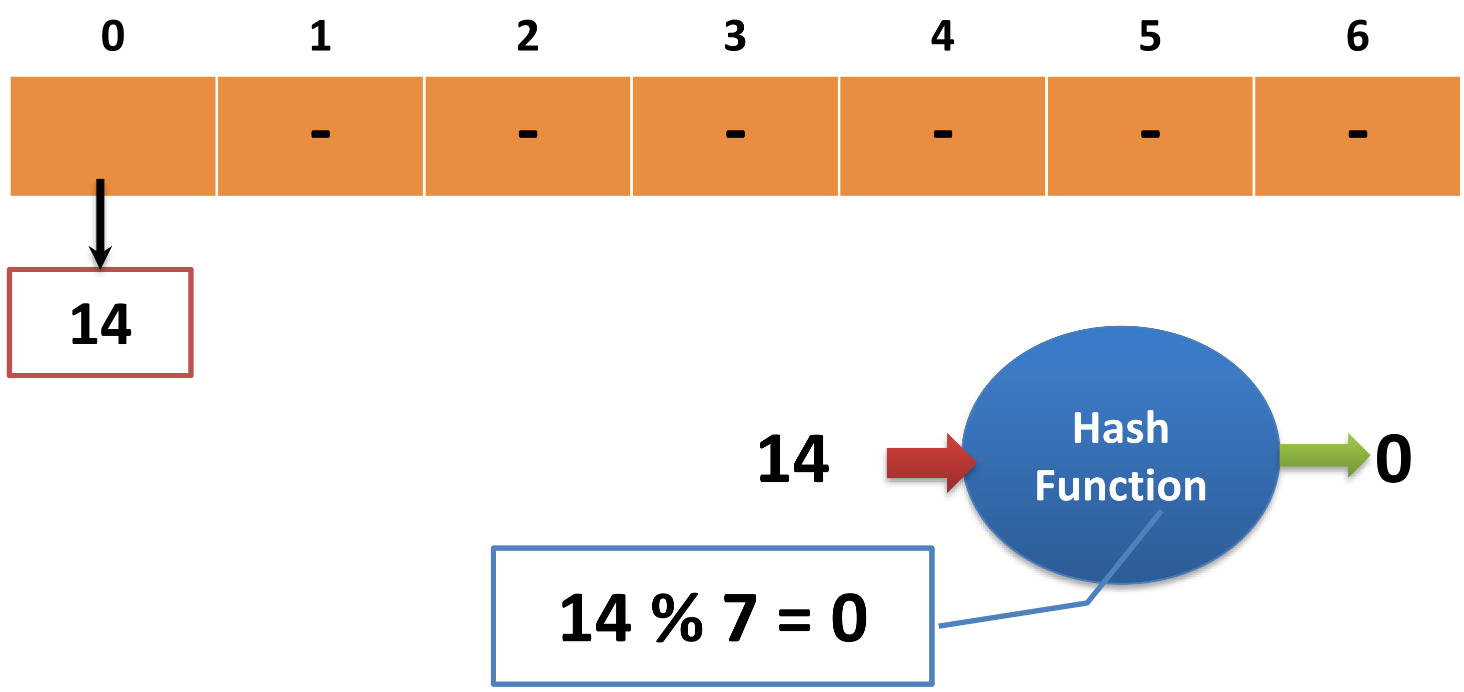
The process of creating a **linked list of values** if they hashed into the same location.

In **open addressing**, each array element can **hold just one entry**. When the array is full, no more records can be added to the table.

But in **Chaining** each component of the hash table's array **can hold more than one entry**.

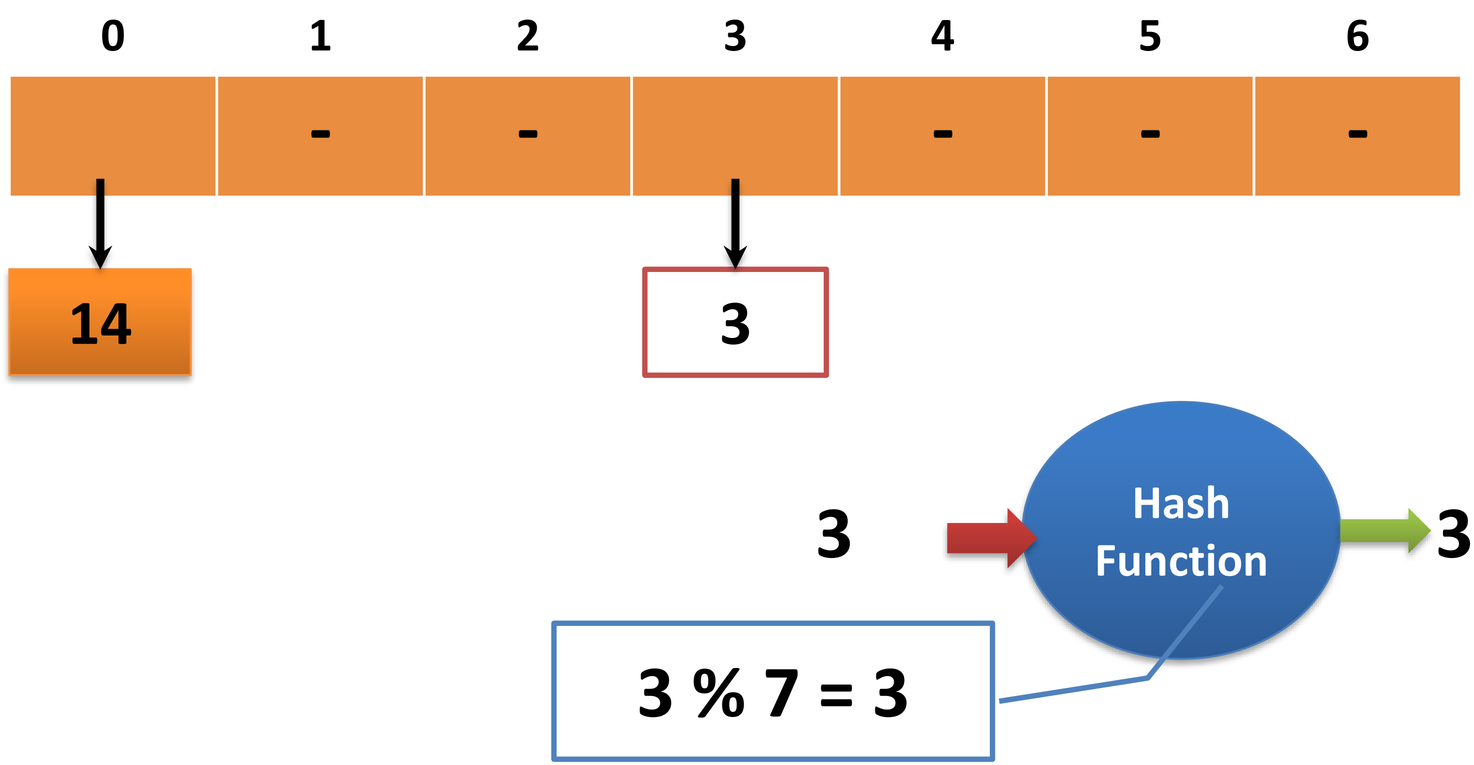
Separate chaining

Insert Elements in this order 14, 3, 9, 7, 23, 19, 35



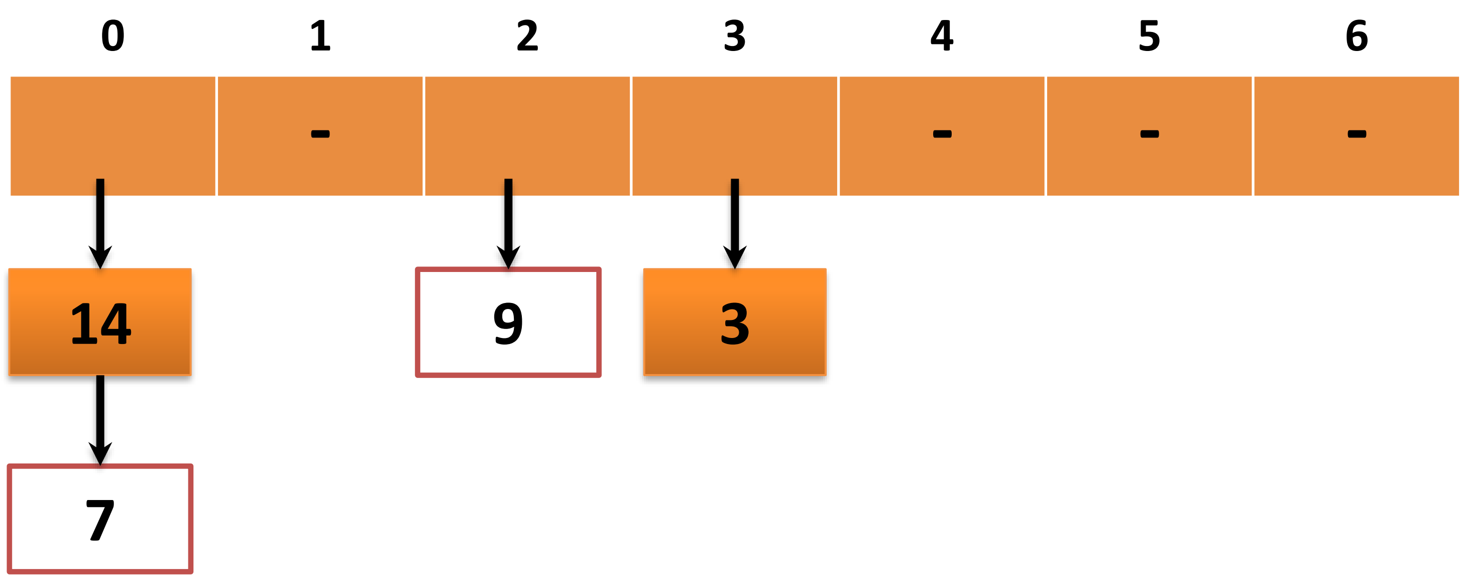
Separate chaining

Insert Elements in this order 14, 3, 9, 7, 23, 19, 35



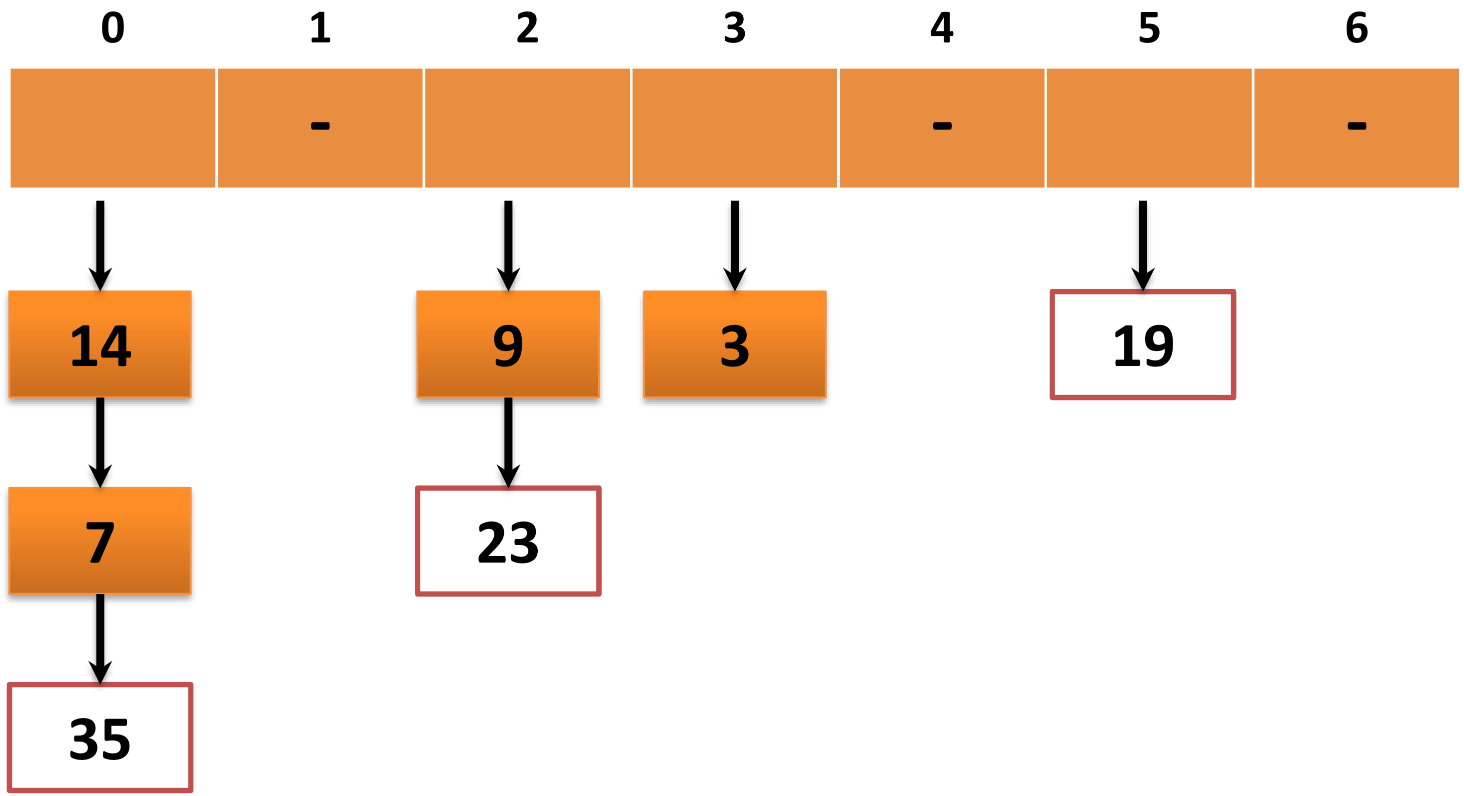
Separate chaining

Insert Elements in this order 14, 3, 9, 7, 23, 19, 35

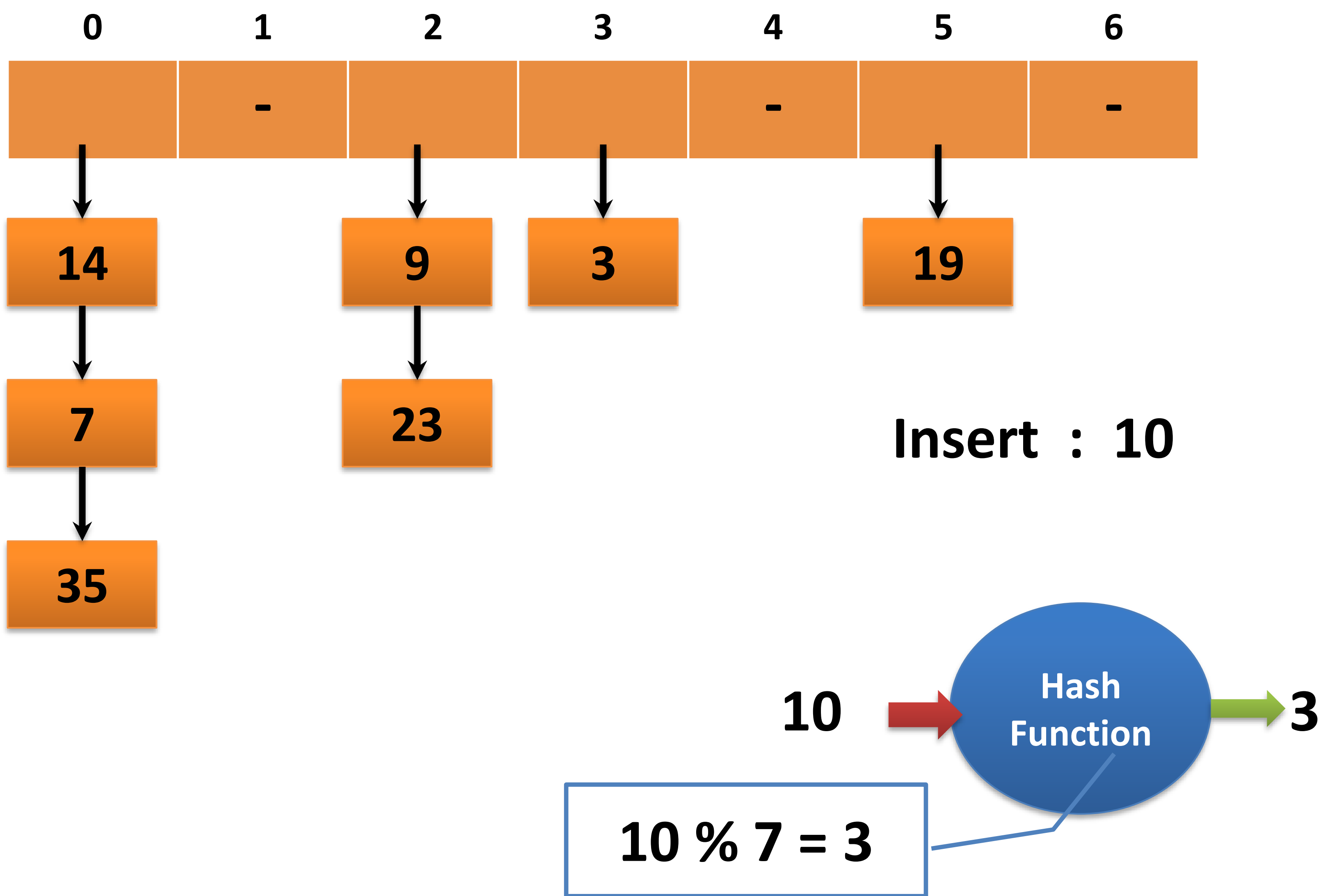


Separate chaining

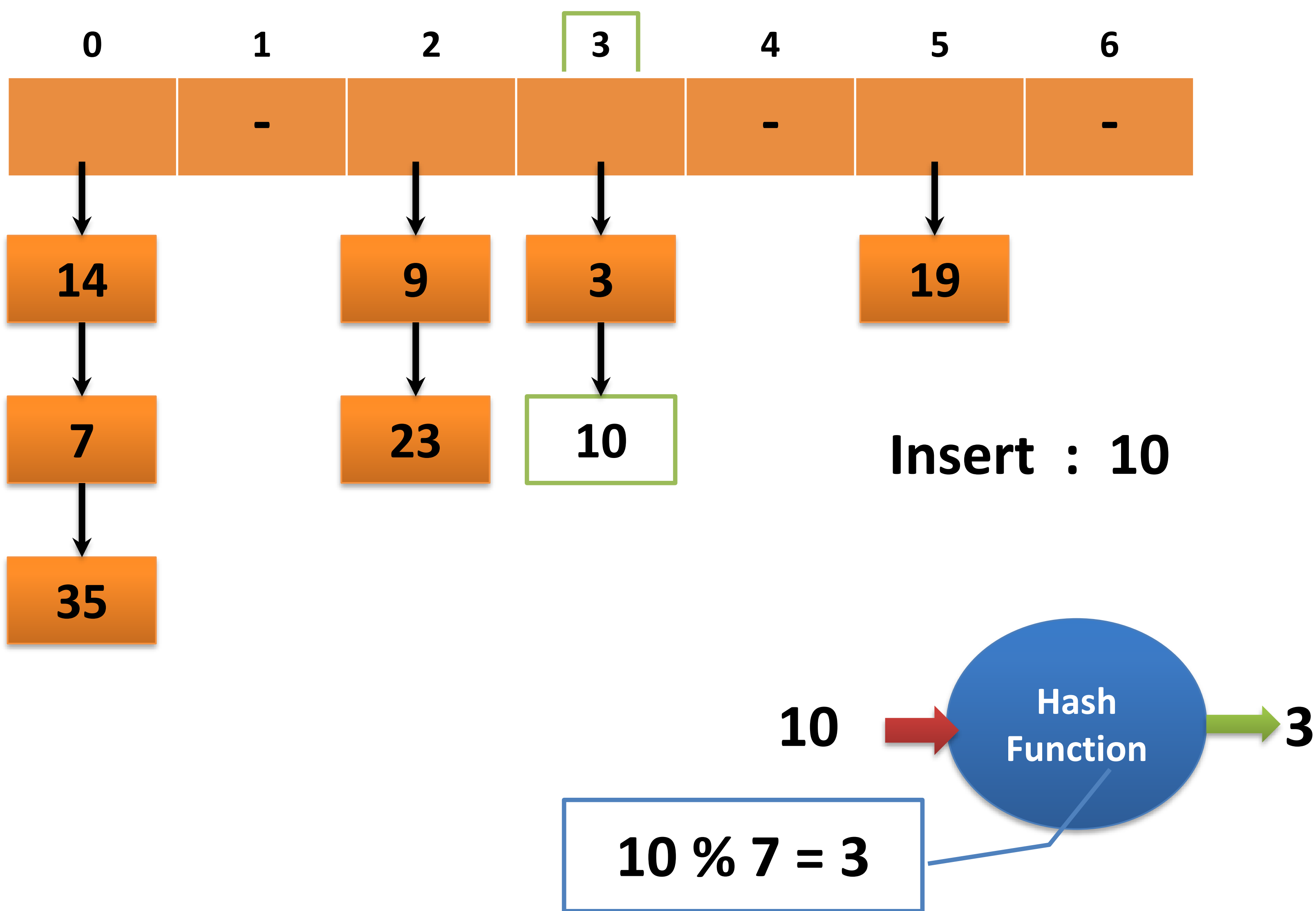
Insert Elements in this order 14, 3, 9, 7, 23, 19, 35



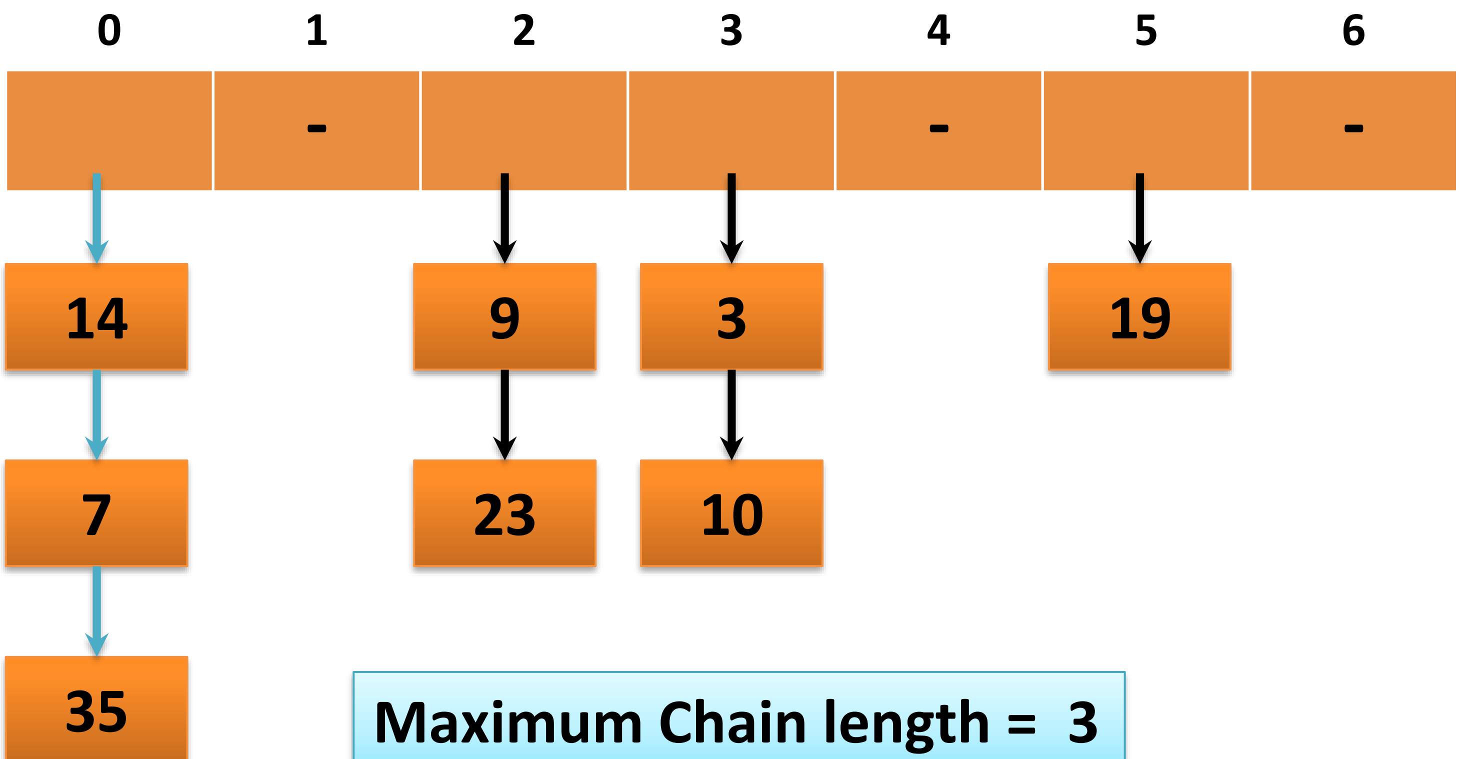
Separate chaining



Separate chaining



Separate chaining



Maximum Chain length = 3

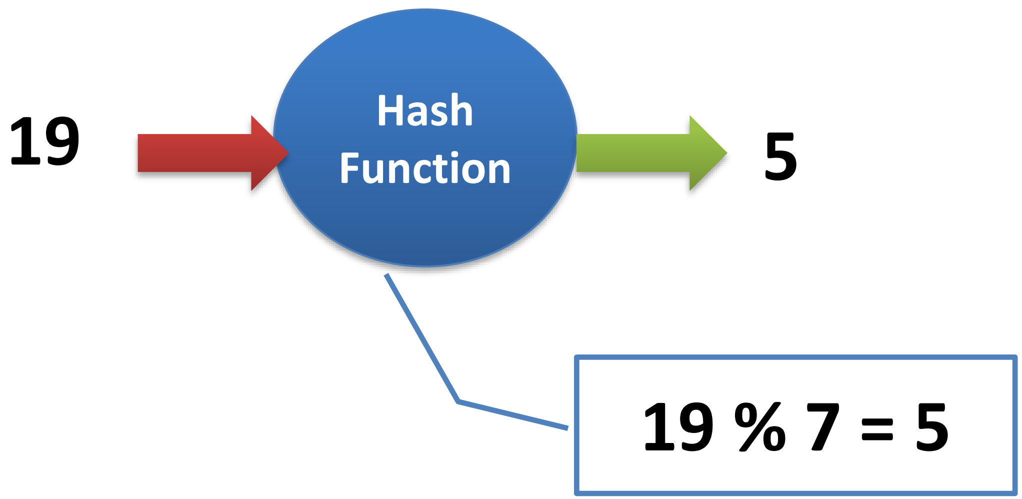
Minimum Chain length = 0

Average Chain length = (3 + 0 + 2 + 2+ 0 + 1 + 0) / 7 = 8 /7

- **separate chaining:** if the number of records is not known in advance
- **open addressing:** if the number of the records can be predicted and there is enough memory available

Hash Table – Lazy Deletion

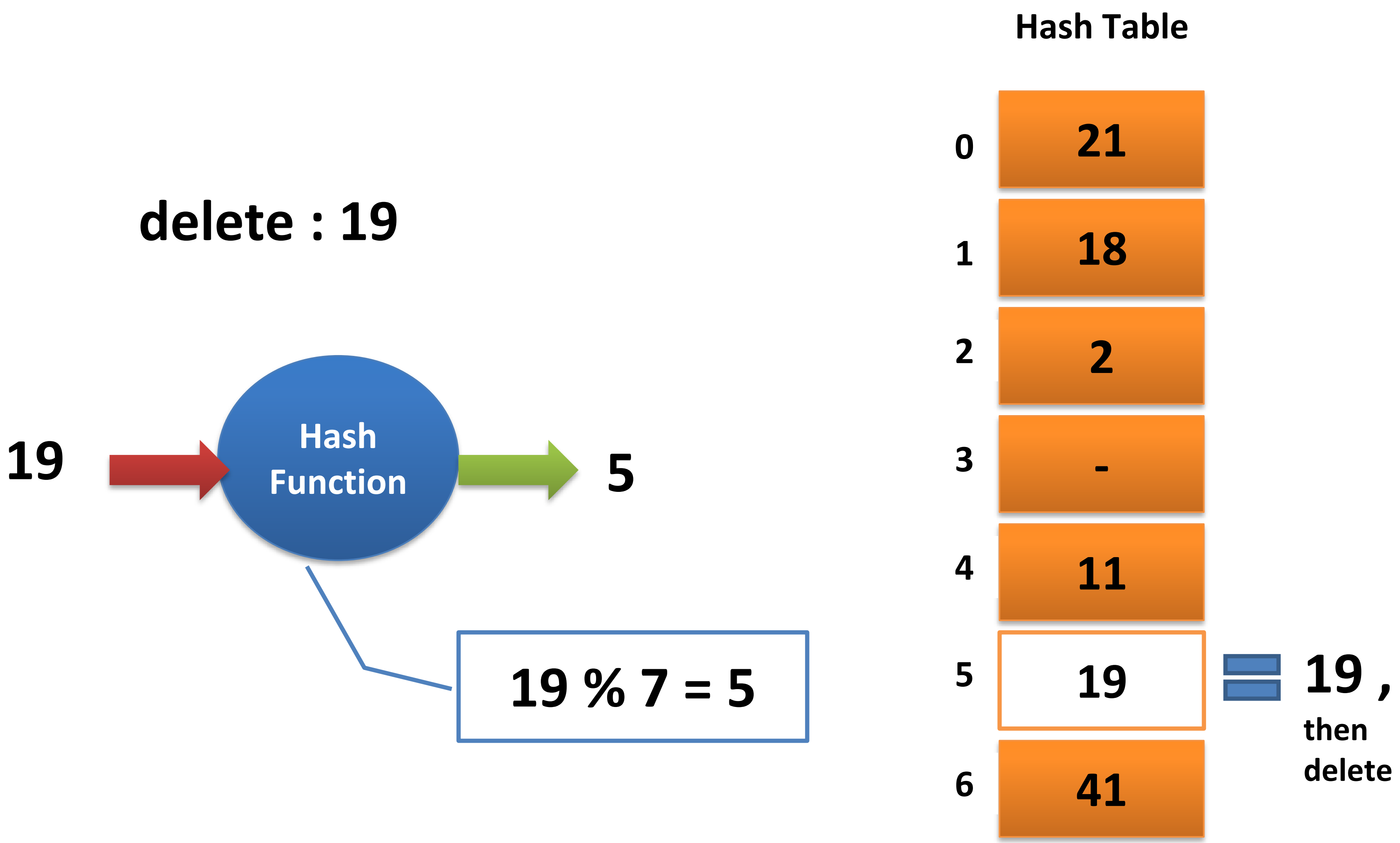
delete : 19



Hash Table

0	21
1	18
2	2
3	-
4	11
5	19
6	41

Hash Table- Lazy Deletion

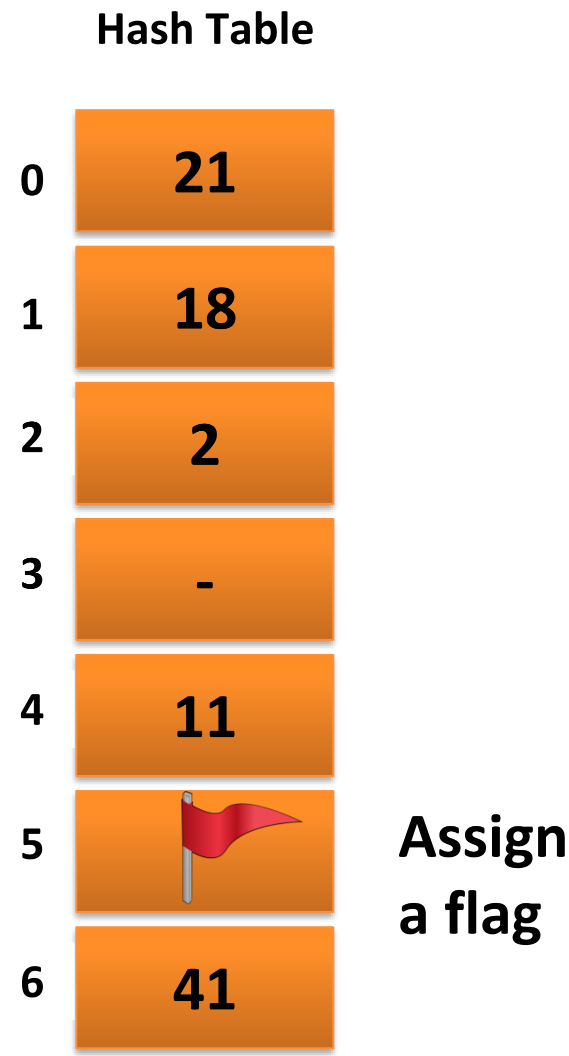


Hash Table - Lazy Deletion

Data 19 is removed and we put a flag to indicate the element is deleted.

The flag indicates that a record once occupied the slot but does so no longer.

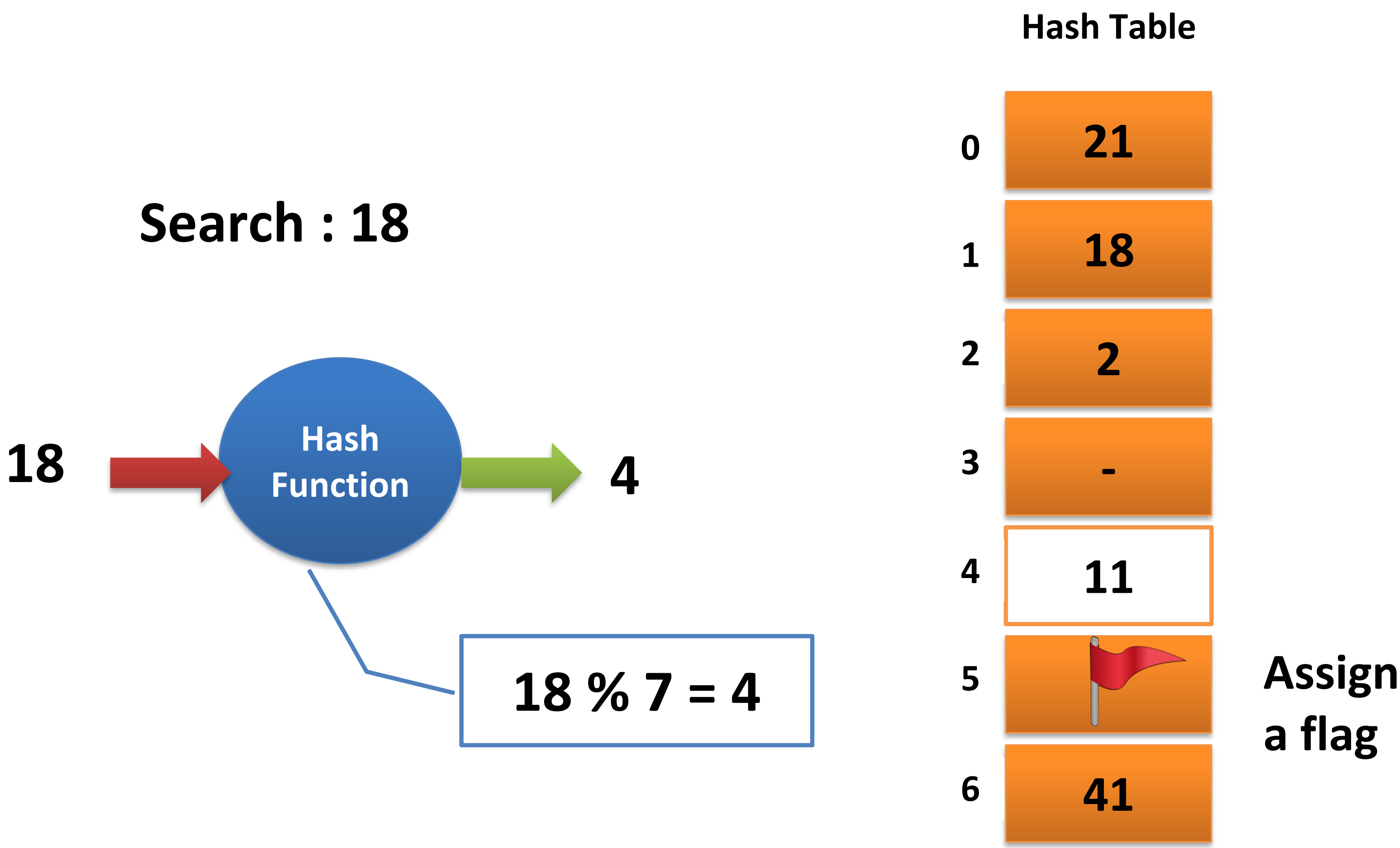
deletions are done by marking an element as deleted, rather than erasing it entirely .



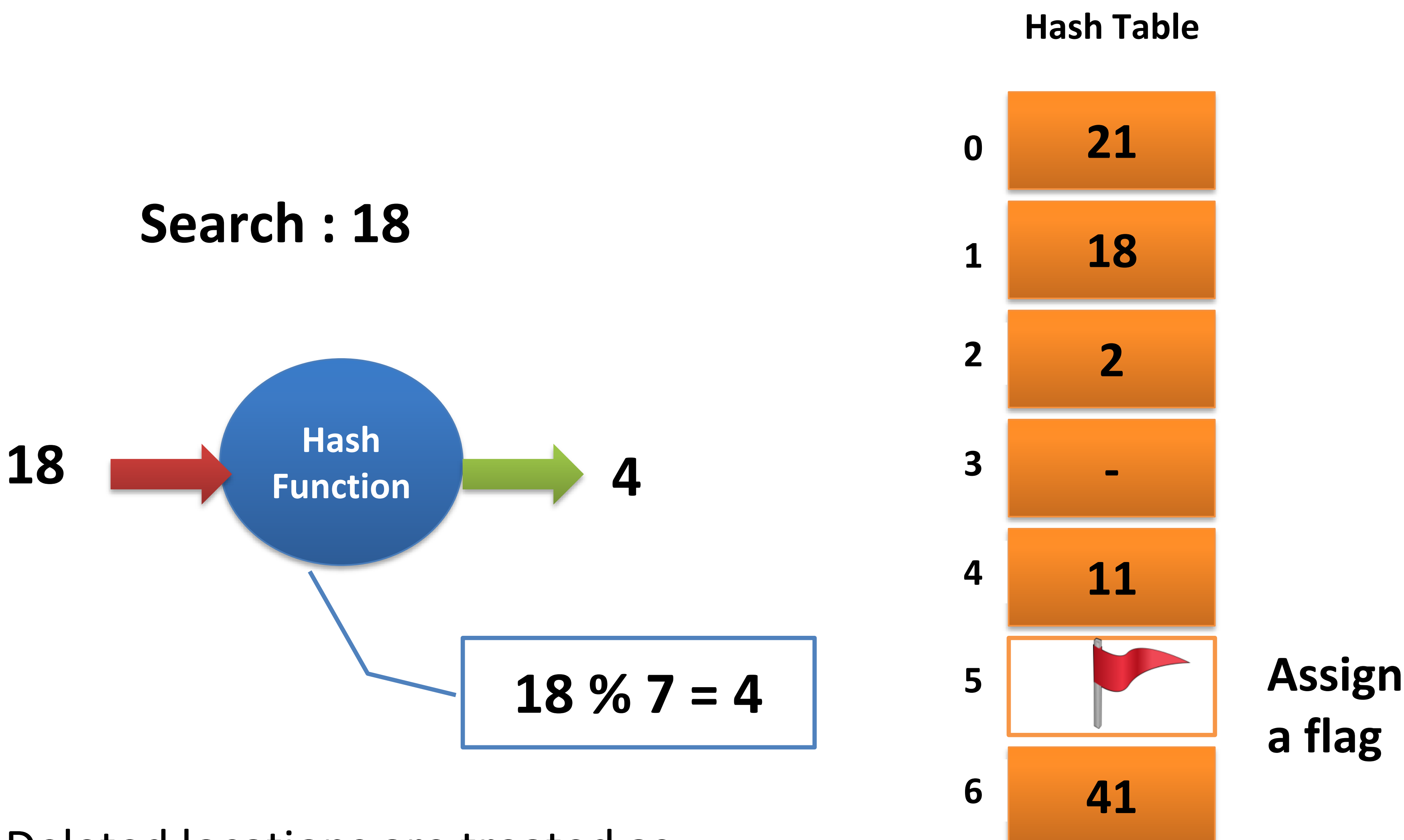
Hash Table - Lazy Deletion

- Deleted locations are treated as **empty** when inserting and as **occupied** during a search.

Hash Table - Lazy Deletion

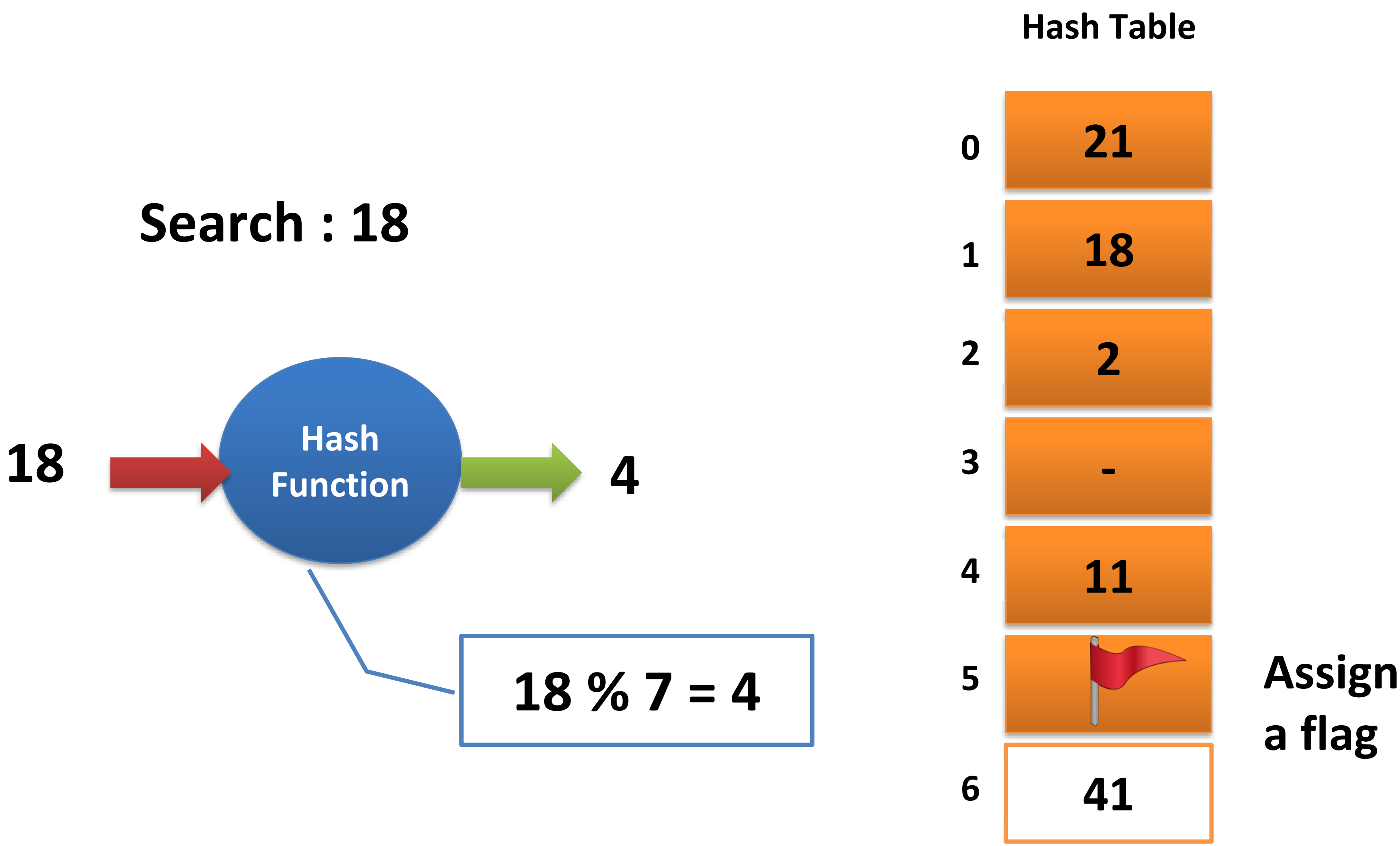


Hash Table - Lazy Deletion



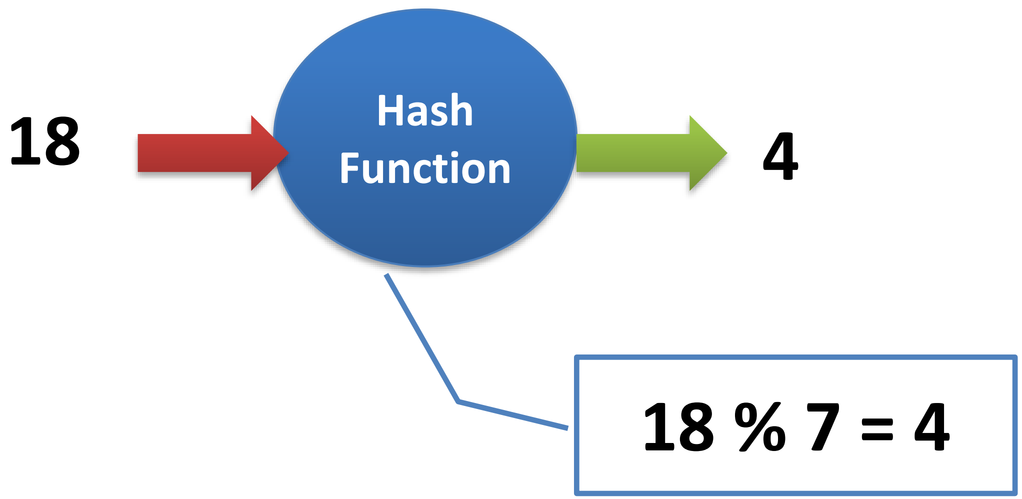
Deleted locations are treated as **occupied** during a search.

Hash Table - Lazy Deletion



Hash Table - Lazy Deletion

Search : 18



Hash Table

0	21
1	18
2	2
3	-
4	11
5	
6	41

Assign a flag

Hash Table - Lazy Deletion