CMPS 3120

Algorithm Analysis

Dr. Chengwei Lei
CEECS
California State University, Bakersfield

Hashing

Before We Start


freedom
VISA


Place Your Finger
Lift and rest your finger on the Home button repeatedly.

Block diagram of fingerprint process system.

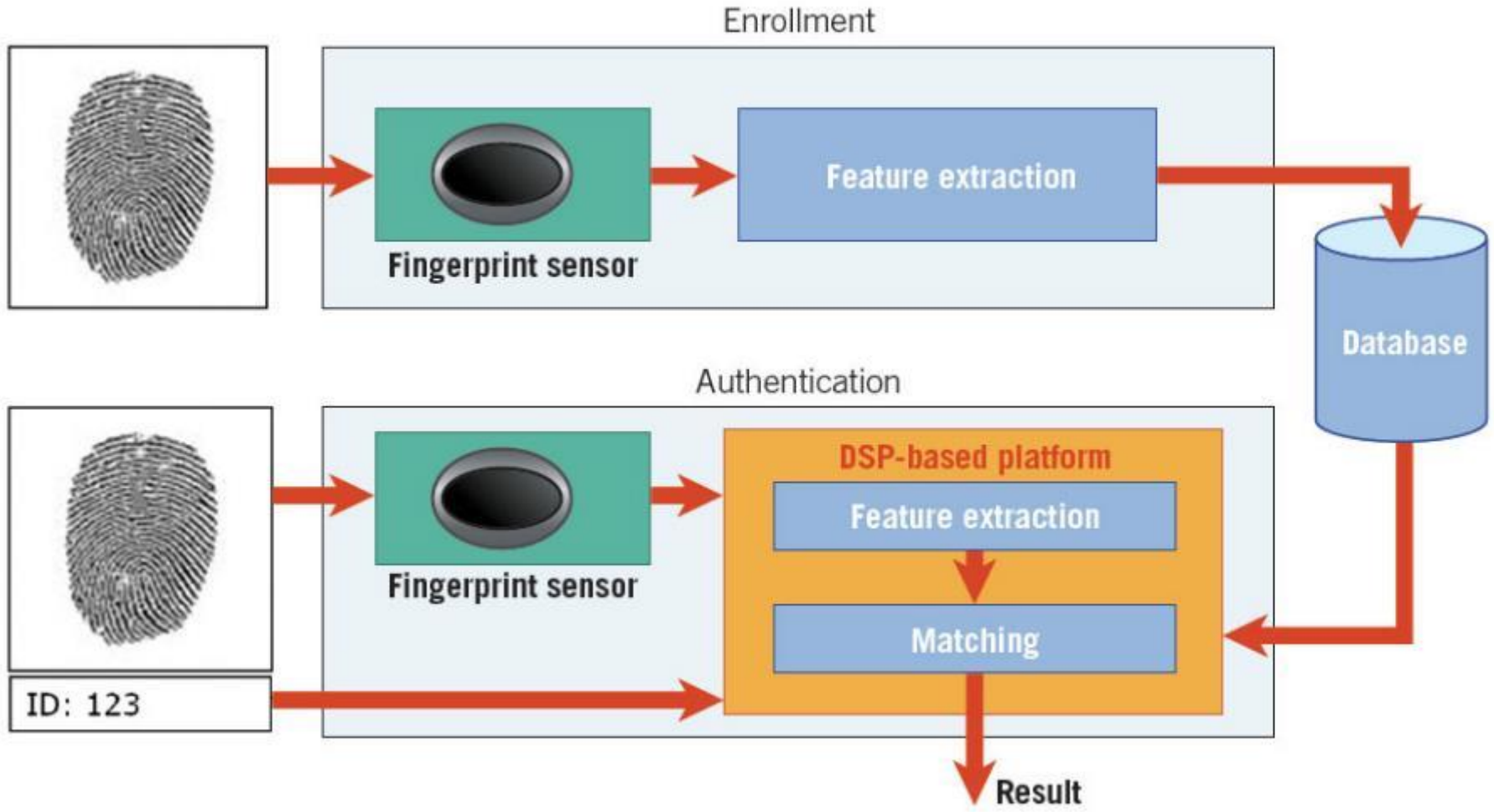


Figure 1

Fingerprint Verification

Fingerprint Verification

Fingerprint Verification

Fingerprint Verification

Back to Class

- How to verify a computer file?
  - An installation file
  - A digital picture
  - A digital book


Files


Files


Files

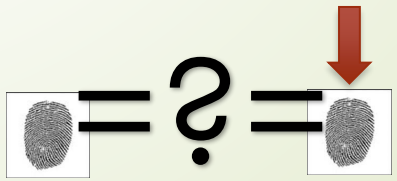
=?=


Files

Digital Fingerprint!

Digital Fingerprint!

Digital Fingerprint!



Hashing

- Hashing is a method of turning some kind of data into a relatively small number that may serve as a digital "fingerprint" of the data.

Hashing

- Hashing is a method of turning some kind of data into a relatively small number that may serve as a digital "fingerprint" of the data.

- The hashing algorithm manipulates the data to create such fingerprints, called hash values.

- A hash function is a function that:
  - When applied to an Object, returns a number
  - When applied to ***equal*** Objects, returns the ***same*** number for each
  - When applied to ***unequal*** Objects, is ***very unlikely*** to return the same number for each

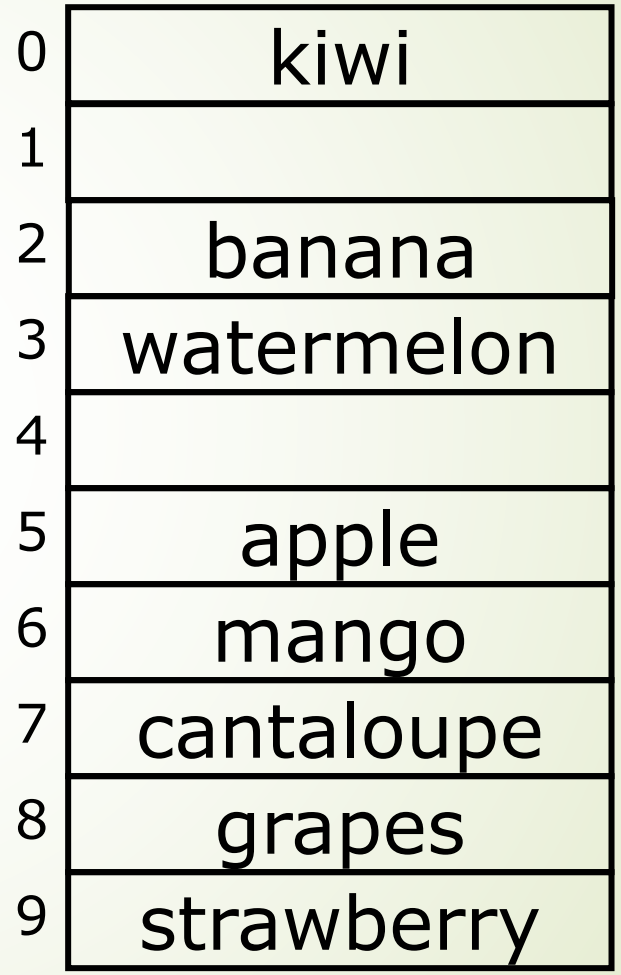
Hashing

- Suppose we were to come up with a "magic function" that, given a value to search for, would tell us exactly where in the array to look
  - If it's in that location, it's in the array
  - If it's not in that location, it's not in the array

Example (ideal) hash function

- Suppose our hash function gave us the following values:

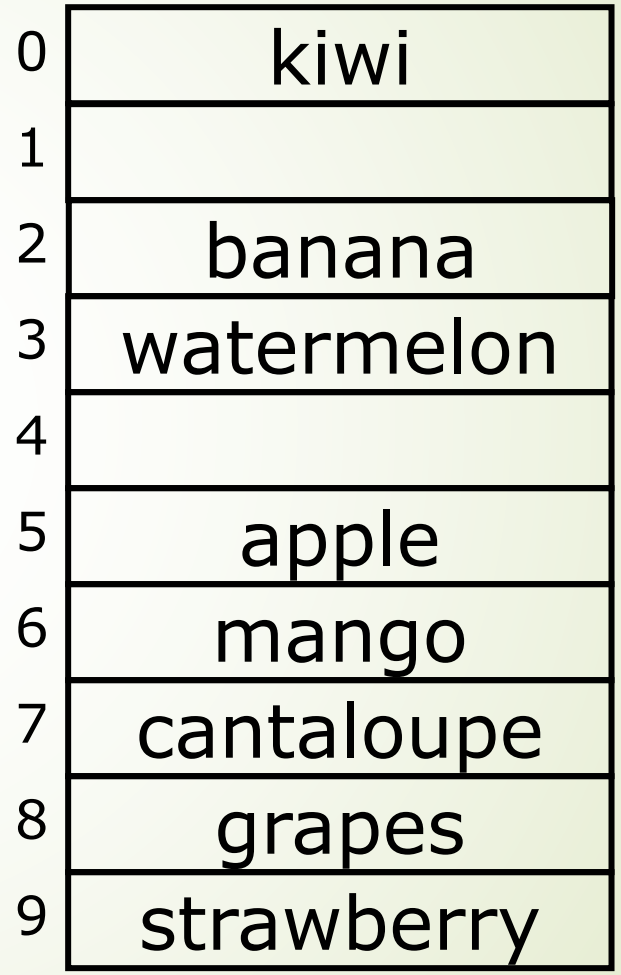
```
hashCode("apple") = 5
hashCode("watermelon") = 3
hashCode("grapes") = 8
hashCode("cantaloupe") = 7
hashCode("kiwi") = 0
hashCode("strawberry") = 9
hashCode("mango") = 6
hashCode("banana") = 2
```



Example (ideal) hash function

- Check if “apple” is on the list

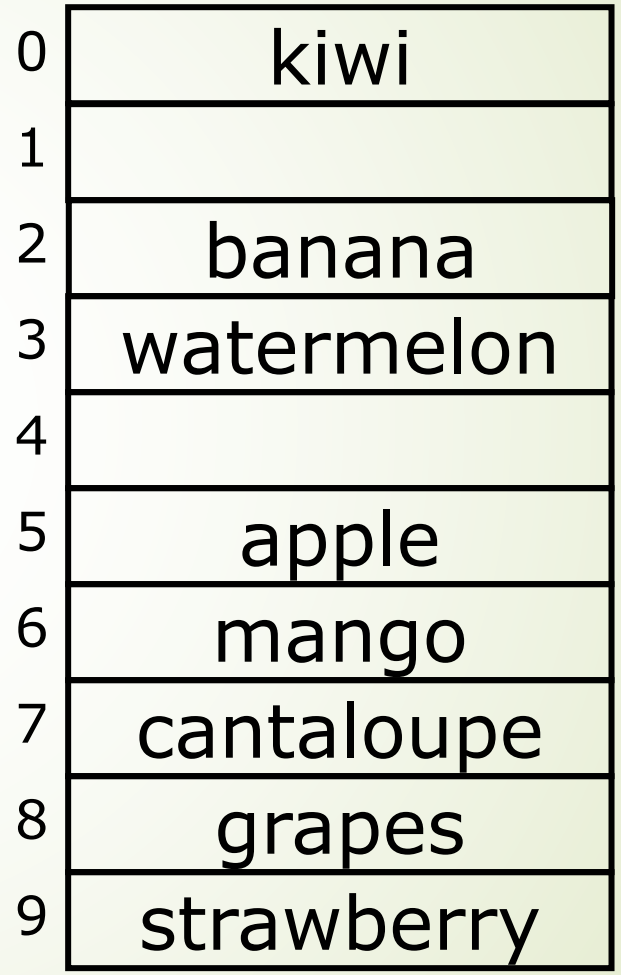
  hashCode(“apple") = ?



Example (ideal) hash function

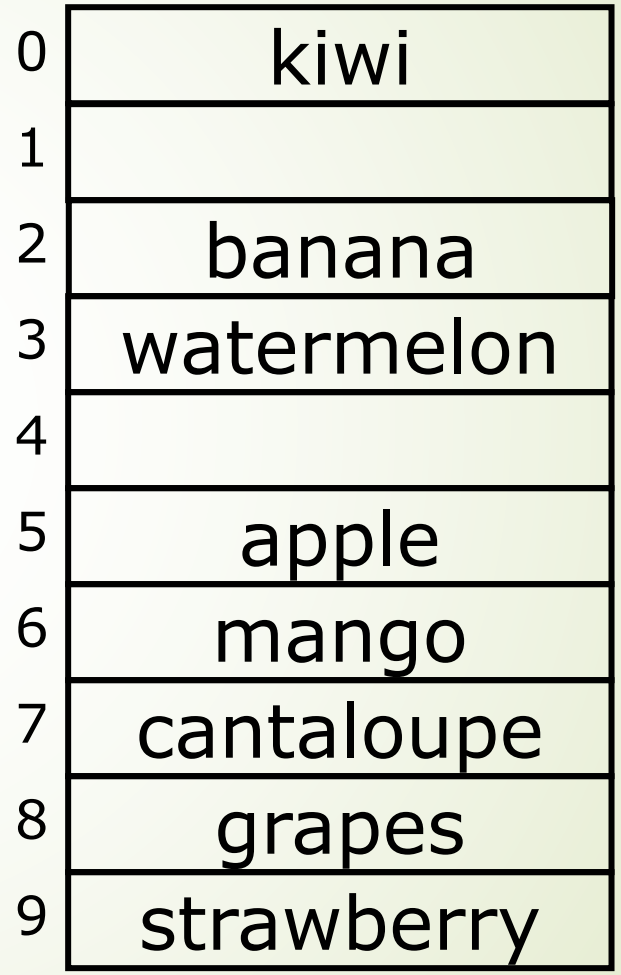
- Check if “apple” is on the list

  hashCode(“apple") = 5



Example (ideal) hash function

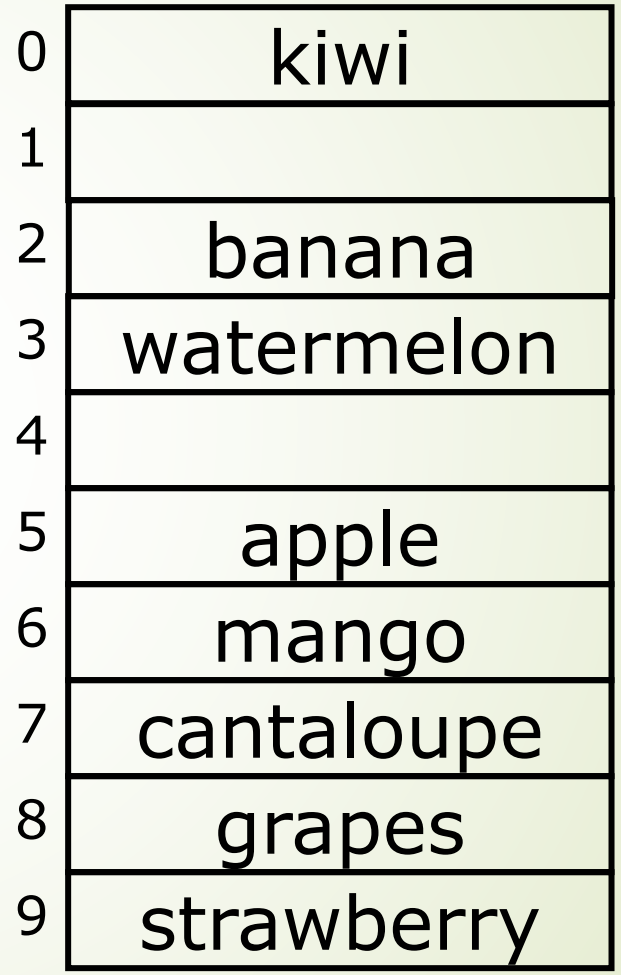
- Check if “pineapple” is on the list



Example (ideal) hash function

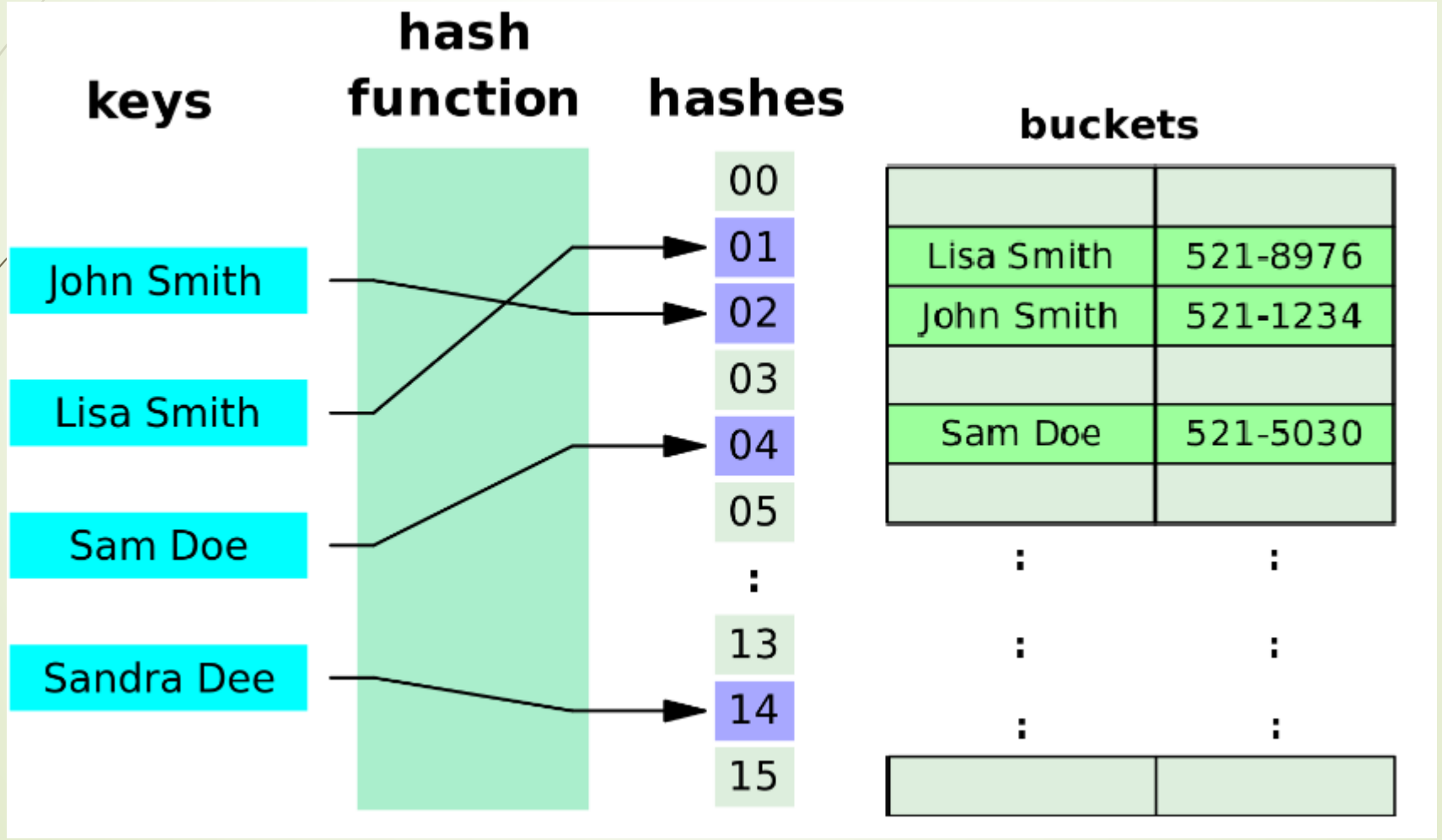
- Check if “pineapple” is on the list

  hashCode(“pineapple") = 4



Componets

- In general, a hash table consists of two major components,
  - a **bucket array**
    - is used to store the data according to their computed indices
  - a **hash function**,
    - maps keys to positions in the hash table


keys
hash function
hashes
buckets
John Smith
Lisa Smith
Sam Doe
Sandra Dee
00
01
02
03
04
05
13
14
15
Lisa Smith
521-8976
John Smith
521-1234
Sam Doe
521-5030

- Hash functions are designed to be
  - fast and
  - yield few hash collisions in expected input domains.

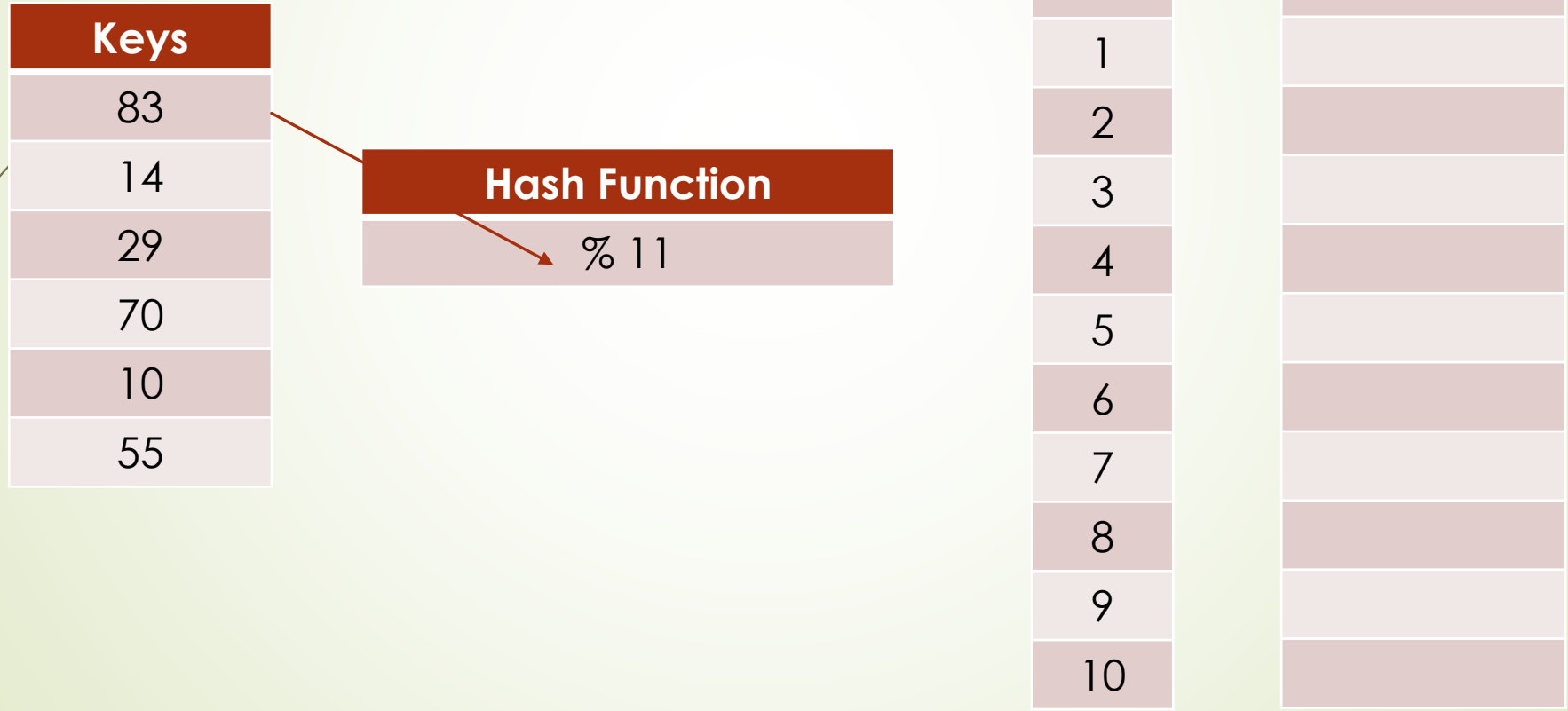
Example

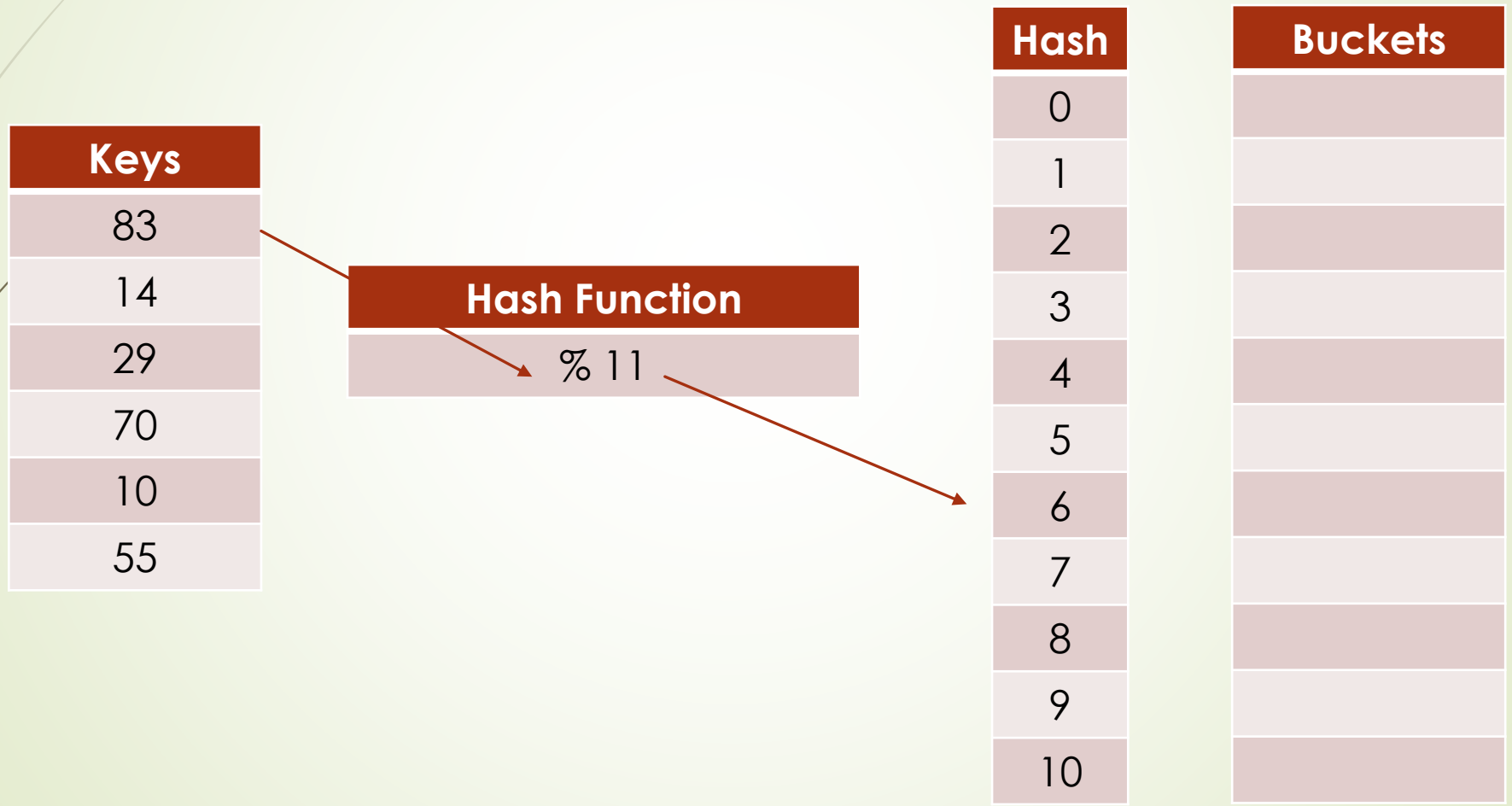
- Let the hash table be an 11-element array.
- If k is the key of a data record, let H(k) represent the hash function,
  - where H(k) = k mod 11.
- Insert the keys 83, 14, 29, 70, 10, 55:

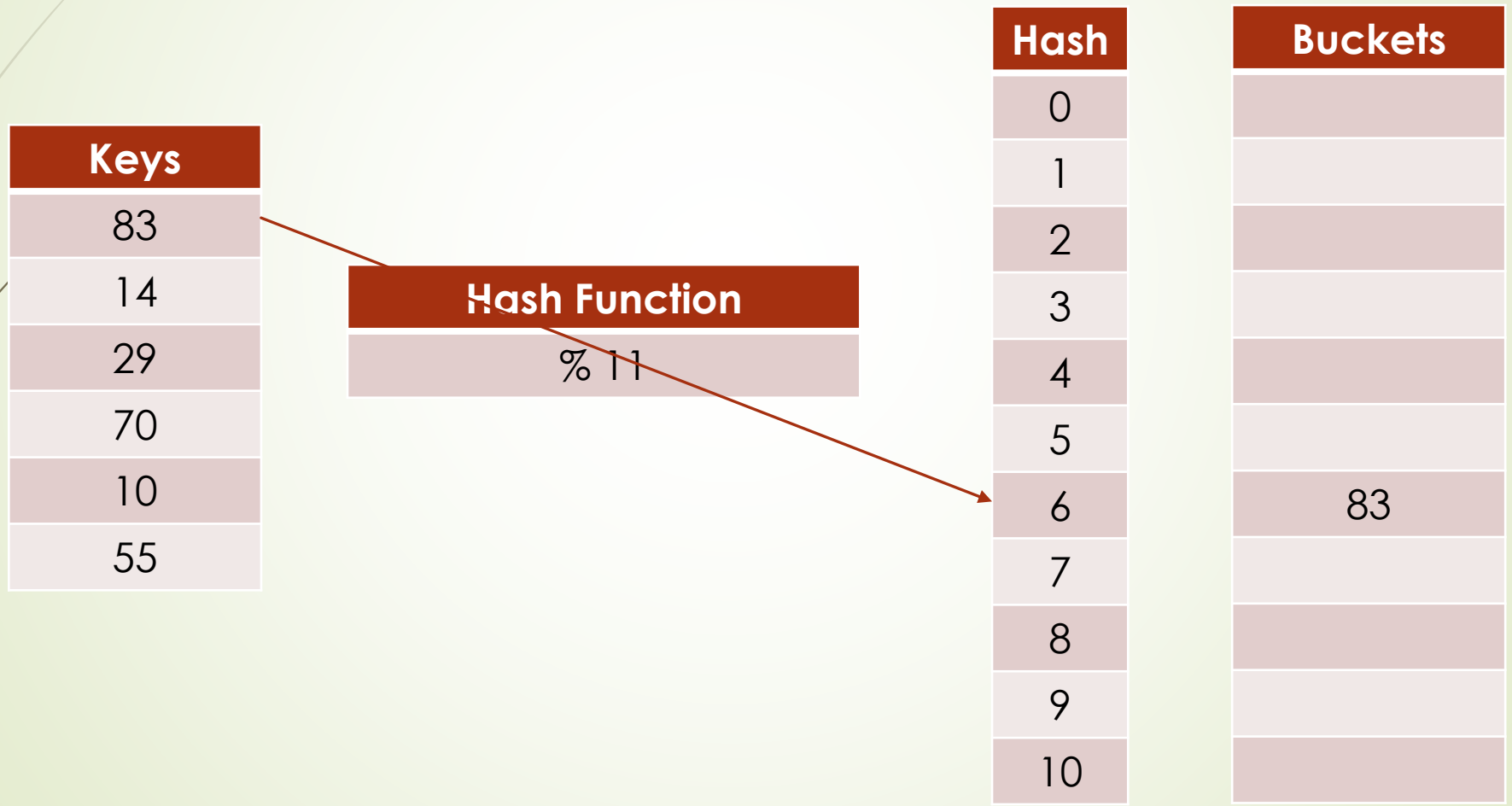
Keys
83
14
29
70
10
55

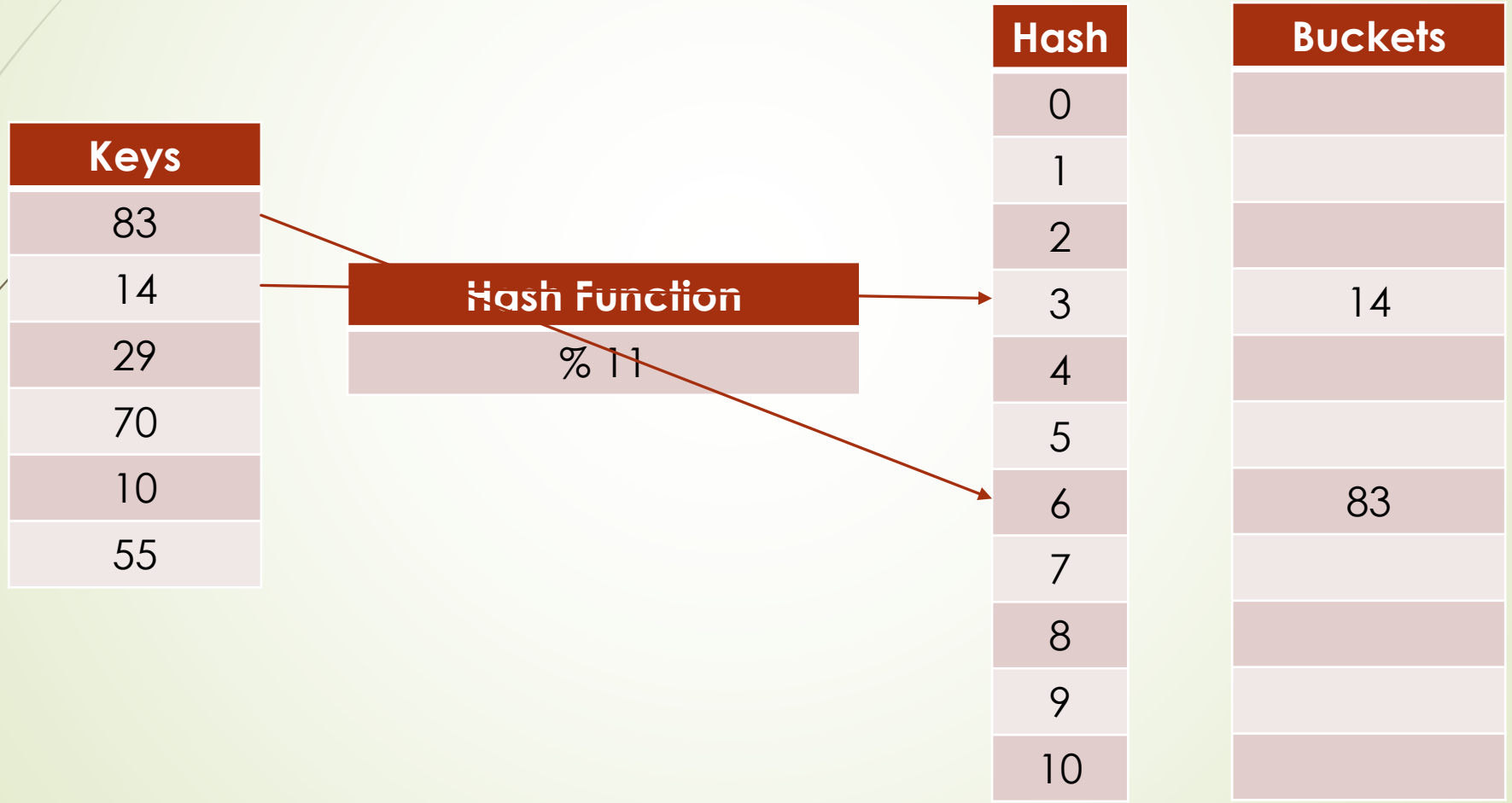
Hash Function
% 11

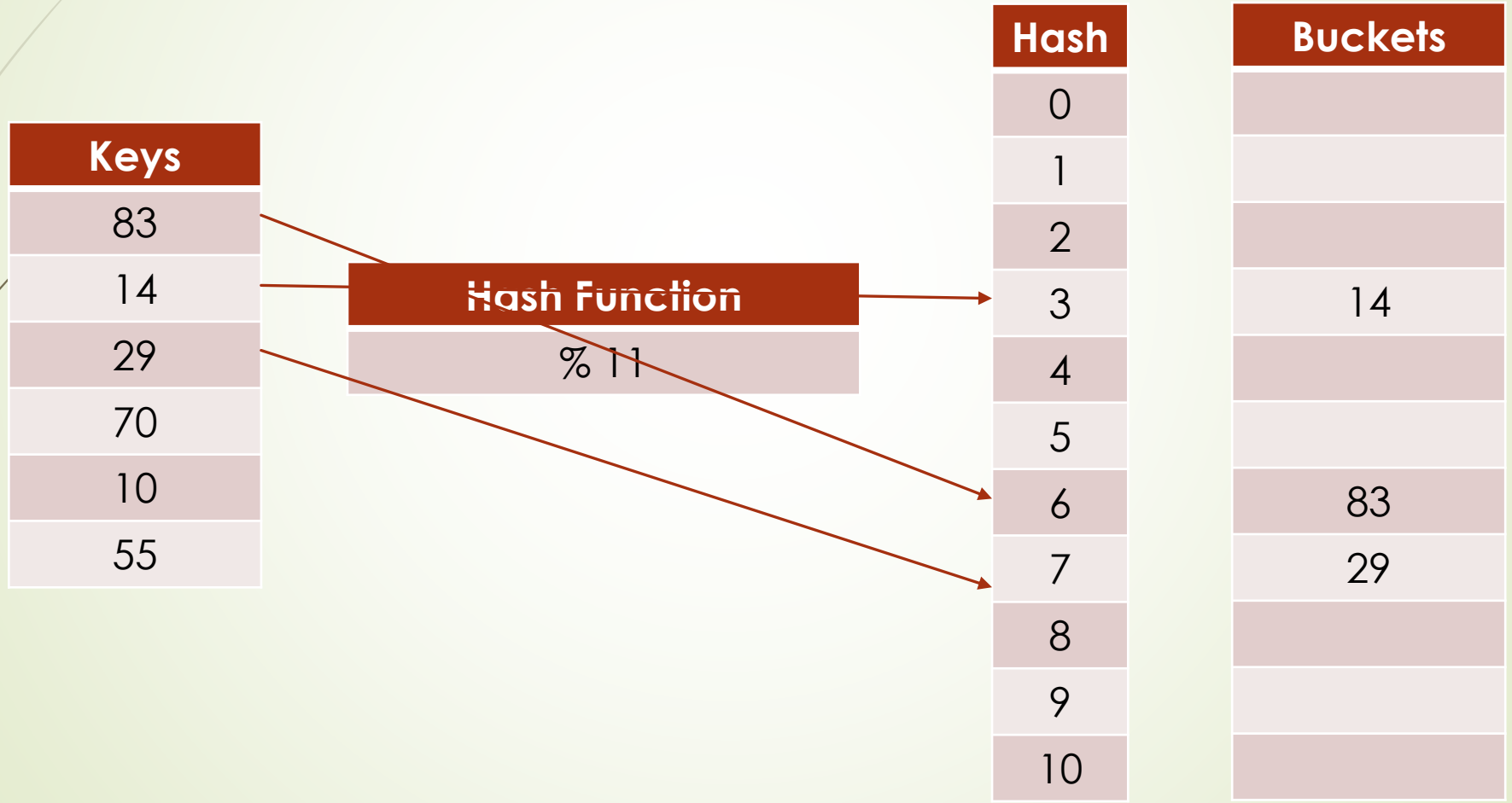
Hash	Buckets
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

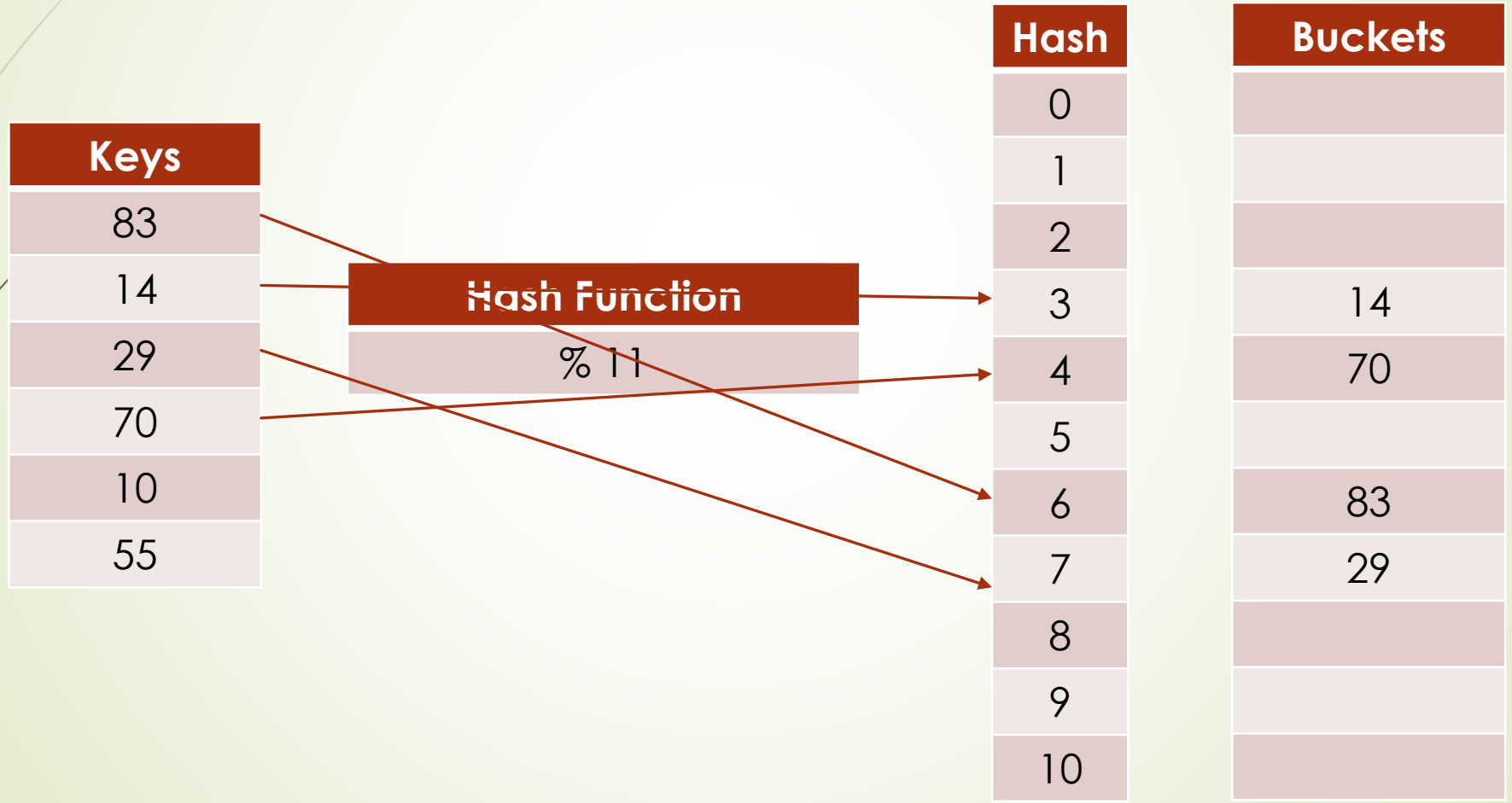

Keys
83
14
29
70
10
55
Hash Function
% 11
Hash
0
1
2
3
4
5
6
7
8
9
10
Buckets

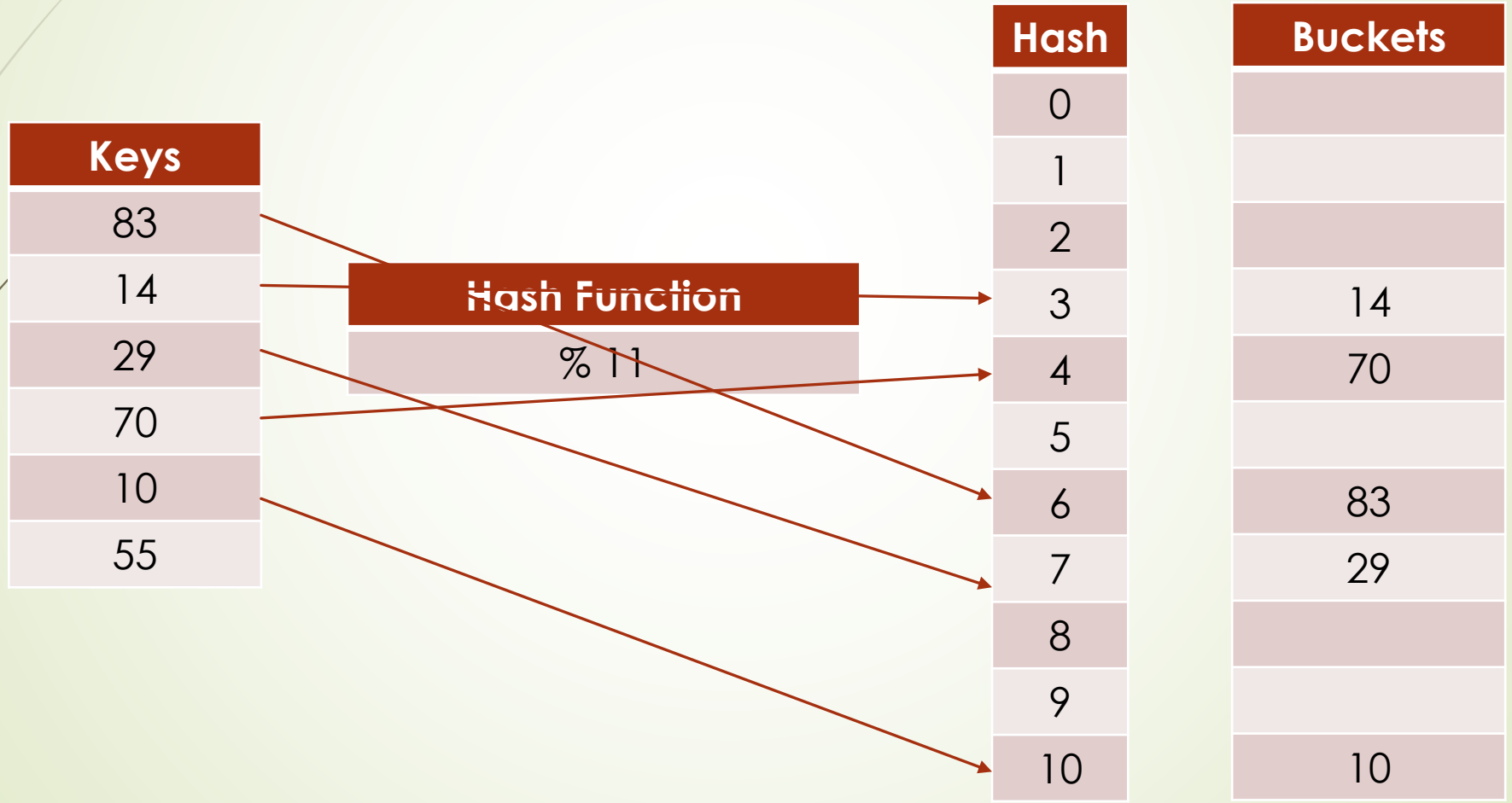

Keys
83
14
29
70
10
55
Hash Function
% 11
Hash
0
1
2
3
4
5
6
7
8
9
10
Buckets

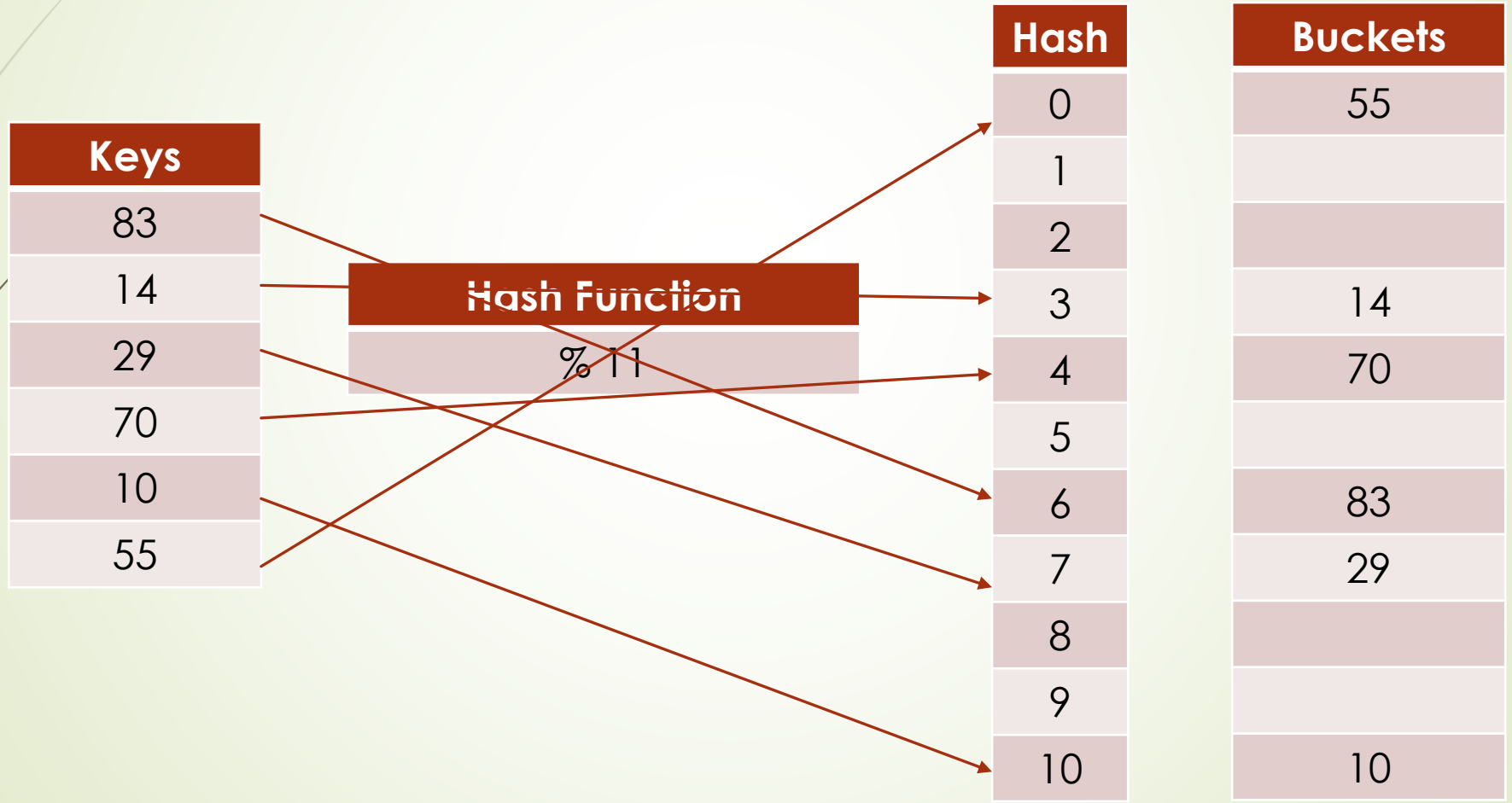

Keys
83
14
29
70
10
55
Hash Function
% 11
Hash
0
1
2
3
4
5
6
7
8
9
10
Buckets
83

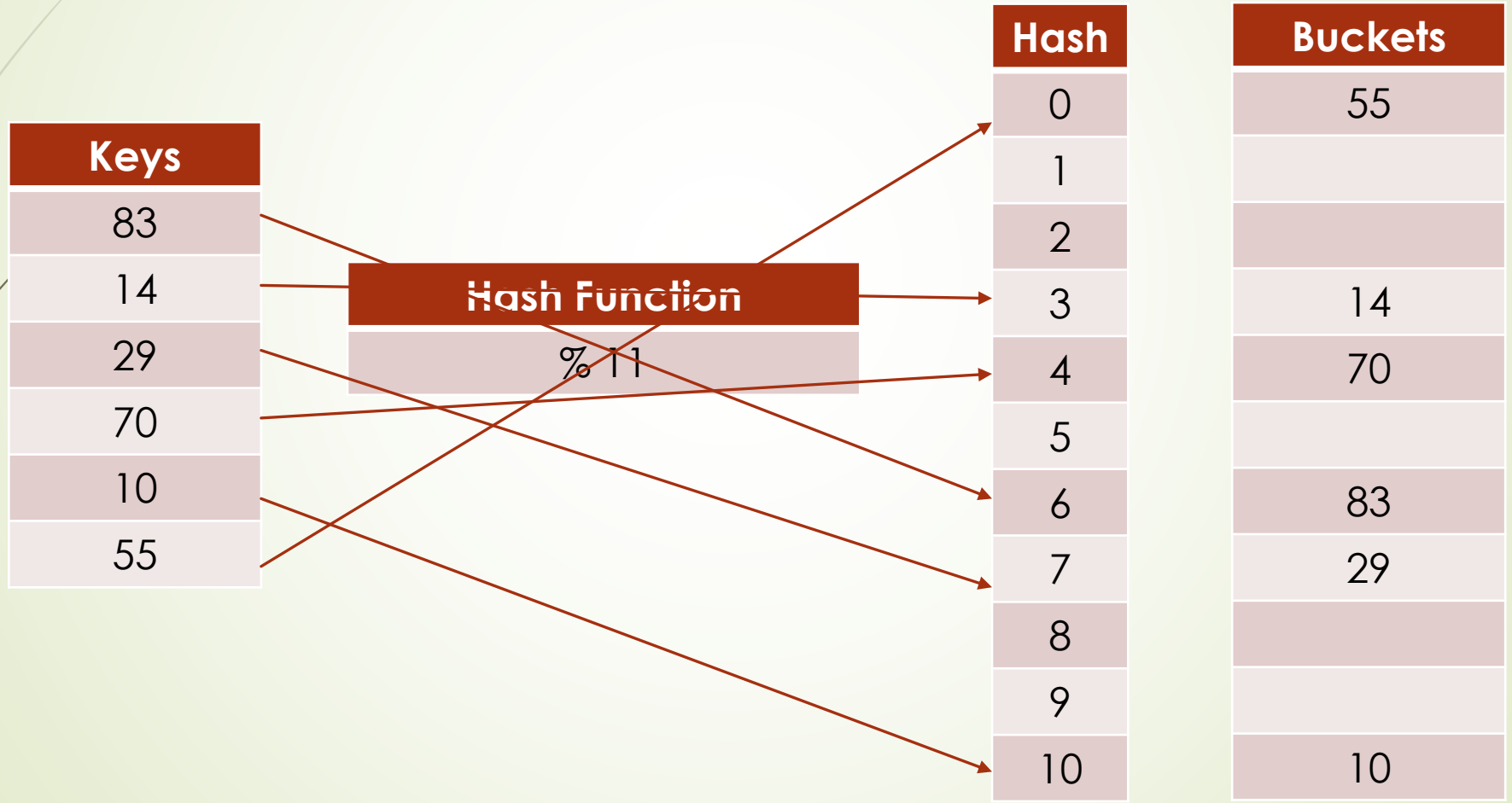

Keys
83
14
29
70
10
55
Hash Function
% 11
Hash
0
1
2
3
4
5
6
7
8
9
10
Buckets
14
83


Keys
83
14
29
70
10
55
Hash Function
% 11
Hash
0
1
2
3
4
5
6
7
8
9
10
Buckets
14
83
29


Keys
83
14
29
70
10
55
Hash Function
% 11
Hash
0
1
2
3
4
5
6
7
8
9
10
Buckets
14
70
83
29

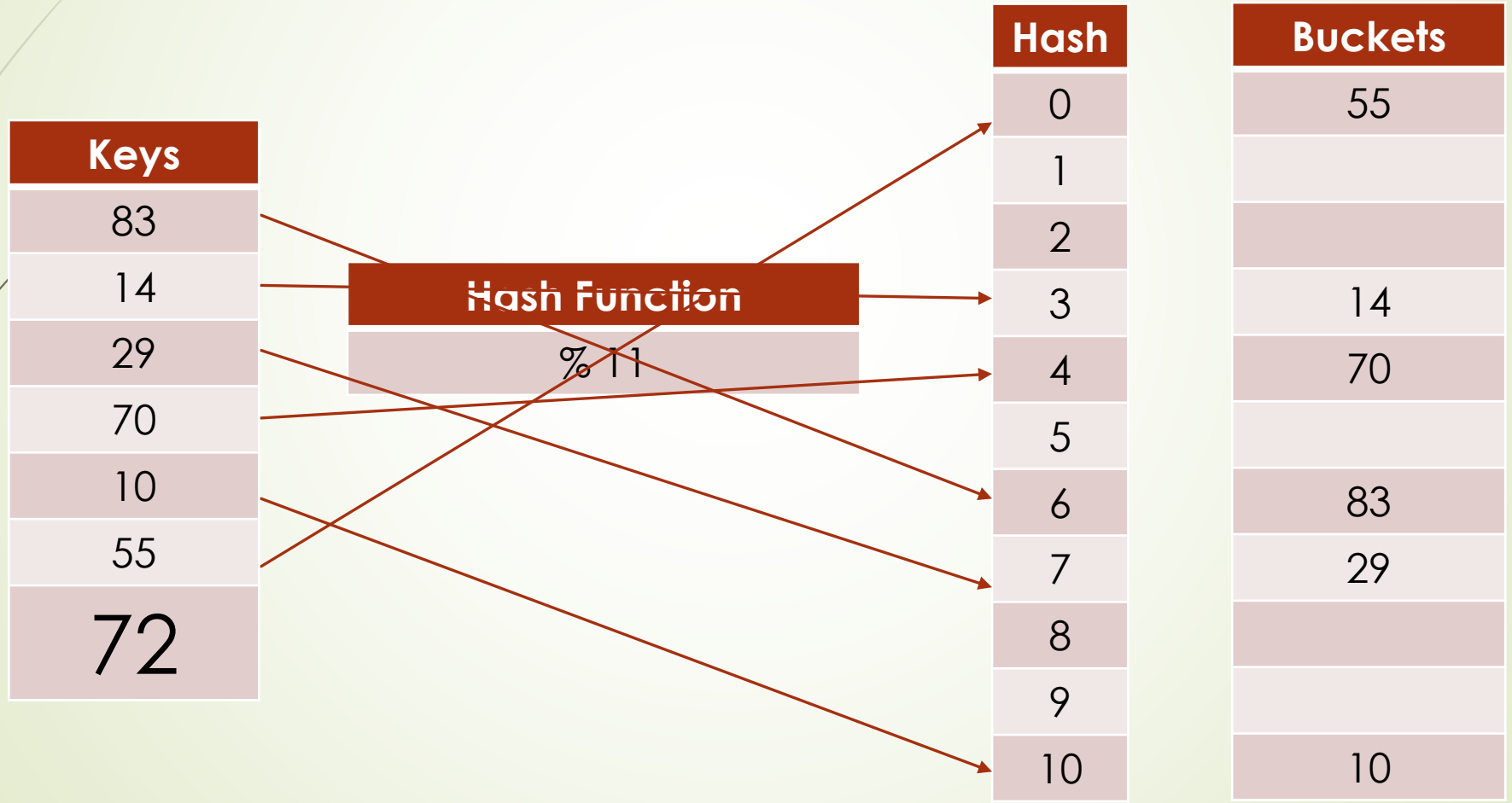

Keys
83
14
29
70
10
55
Hash Function
% 11
Hash
0
1
2
3
4
5
6
7
8
9
10
Buckets
14
70
83
29
10

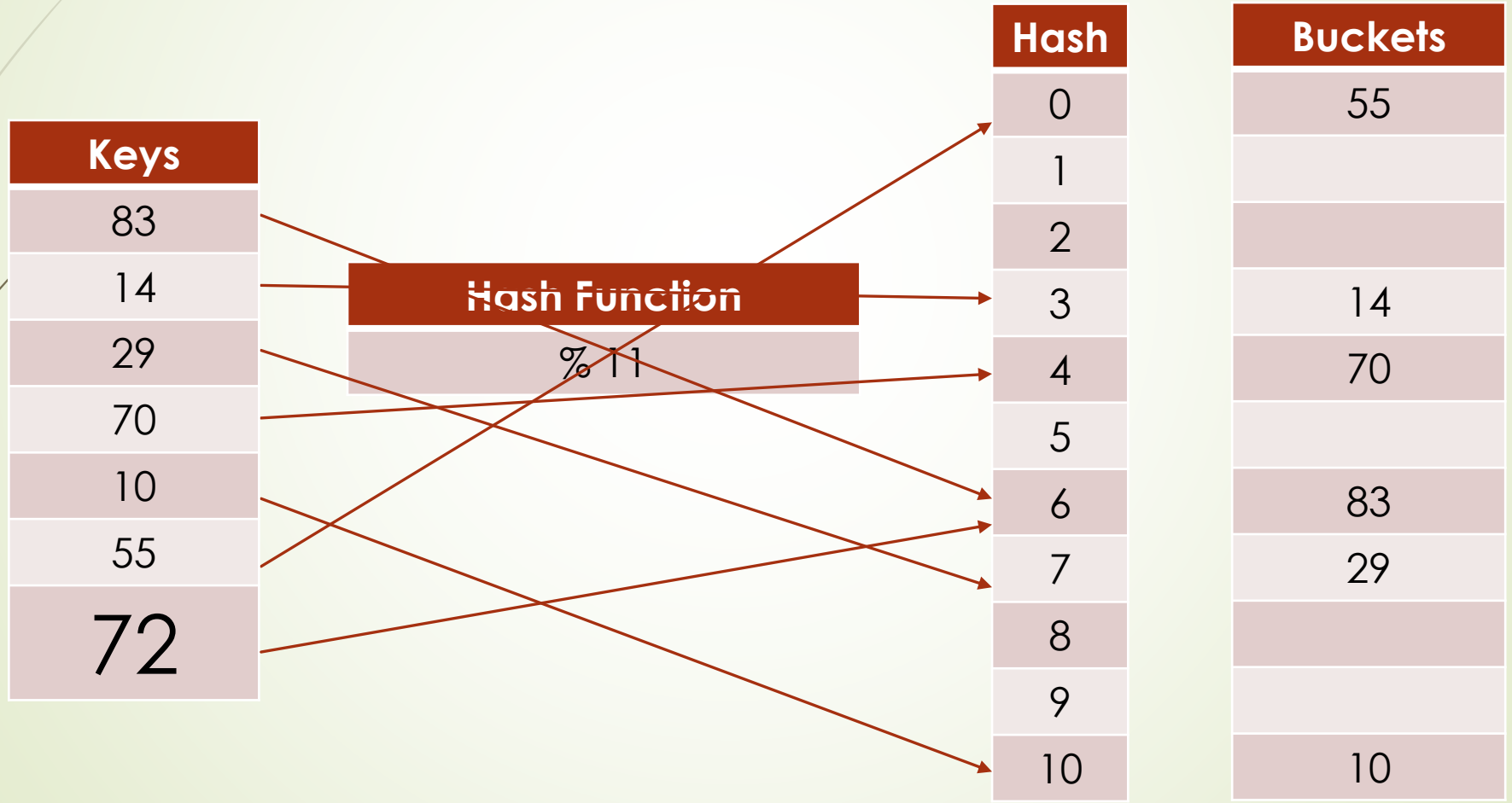

Keys
83
14
29
70
10
55
Hash Function
% 11
Hash
0
1
2
3
4
5
6
7
8
9
10
Buckets
55
14
70
83
29
10

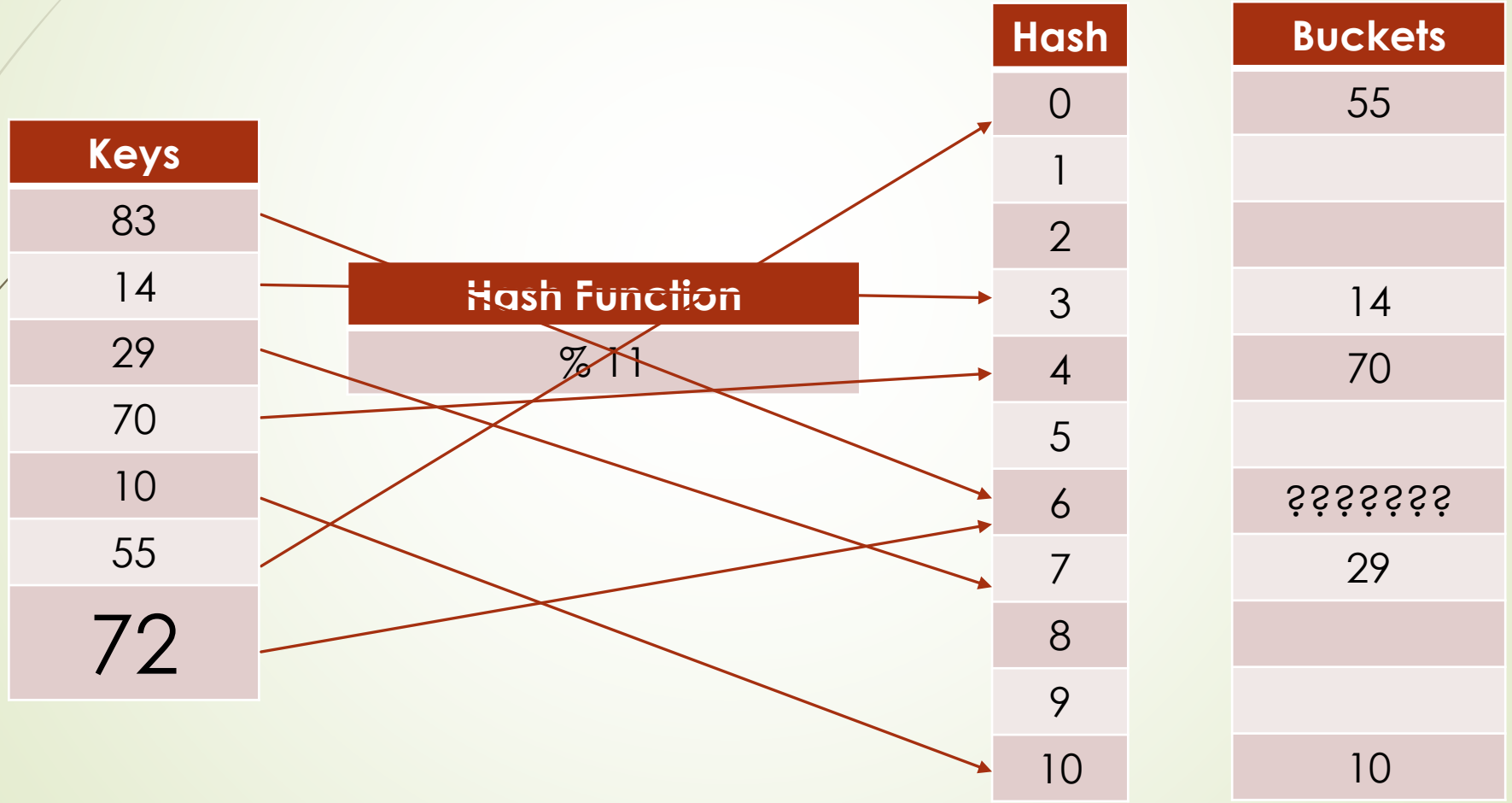

Keys
83
14
29
70
10
55
Hash Function
% 11
Hash
0
1
2
3
4
5
6
7
8
9
10
Buckets
55
14
70
83
29
10

Collision

- If a hash function maps two keys to the same position in the hash table, then a collision occurs.


Keys
83
14
29
70
10
55
72
Hash Function
% 11
Hash
0
1
2
3
4
5
6
7
8
9
10
Buckets
55
14
70
83
29
10


Keys
83
14
29
70
10
55
72
Hash Function
% 11
Hash
0
1
2
3
4
5
6
7
8
9
10
Buckets
55
14
70
83
29
10


Keys
83
14
29
70
10
55
72
Hash Function
% 11
Hash
0
1
2
3
4
5
6
7
8
9
10
Buckets
55
14
70
???????
29
10

Collision

- For any hashing problem of reasonable size, we are almost certain to have collisions.

Example

- Let k be a birthday.
  - Hash each birthday into a table of size 365 (one cell for each day of the year).
- 36 students in this class
  - What is the chance of having at least one collision.

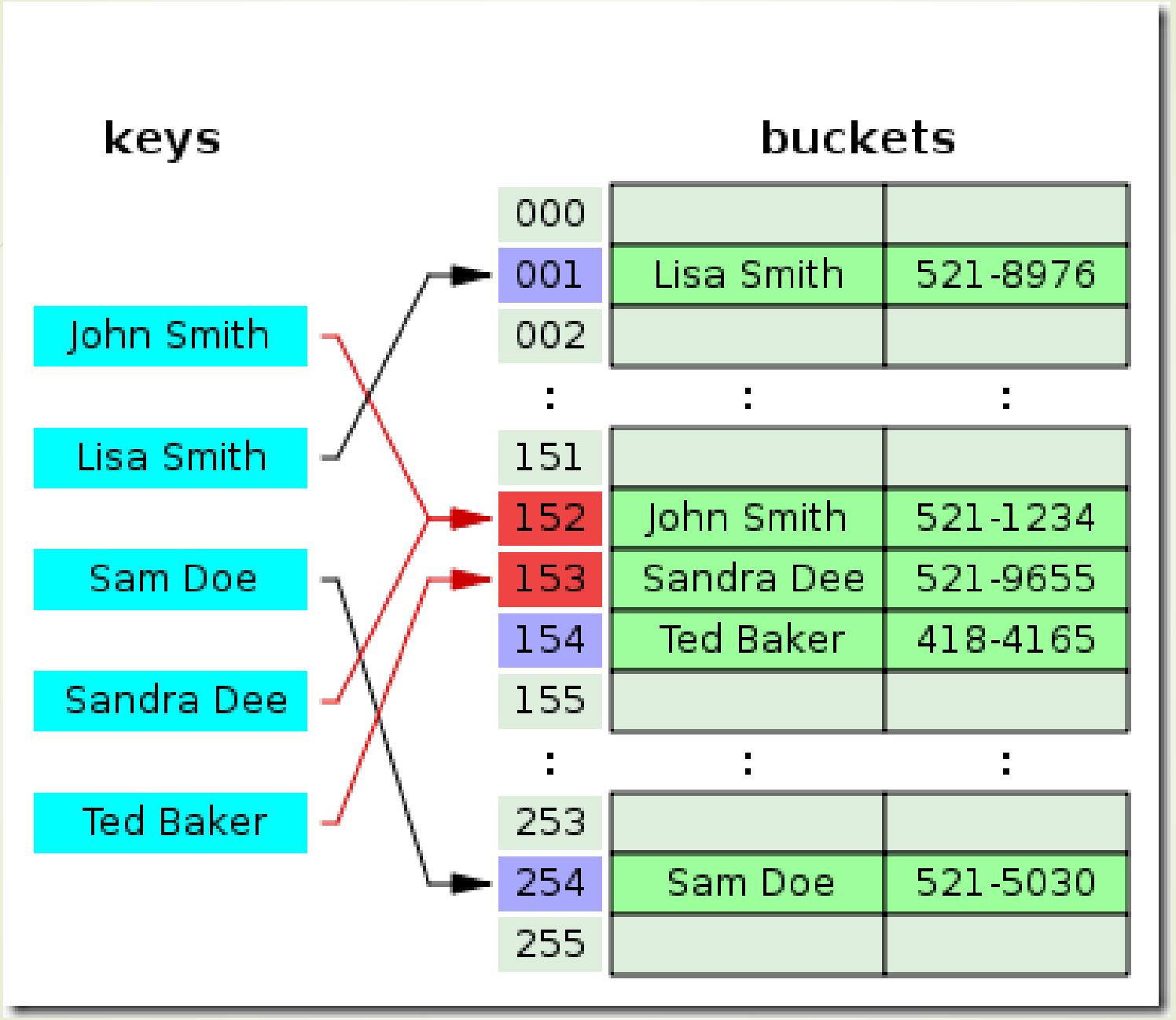
Example

- Let k be a birthday.
  - Hash each birthday into a table of size 365 (one cell for each day of the year).

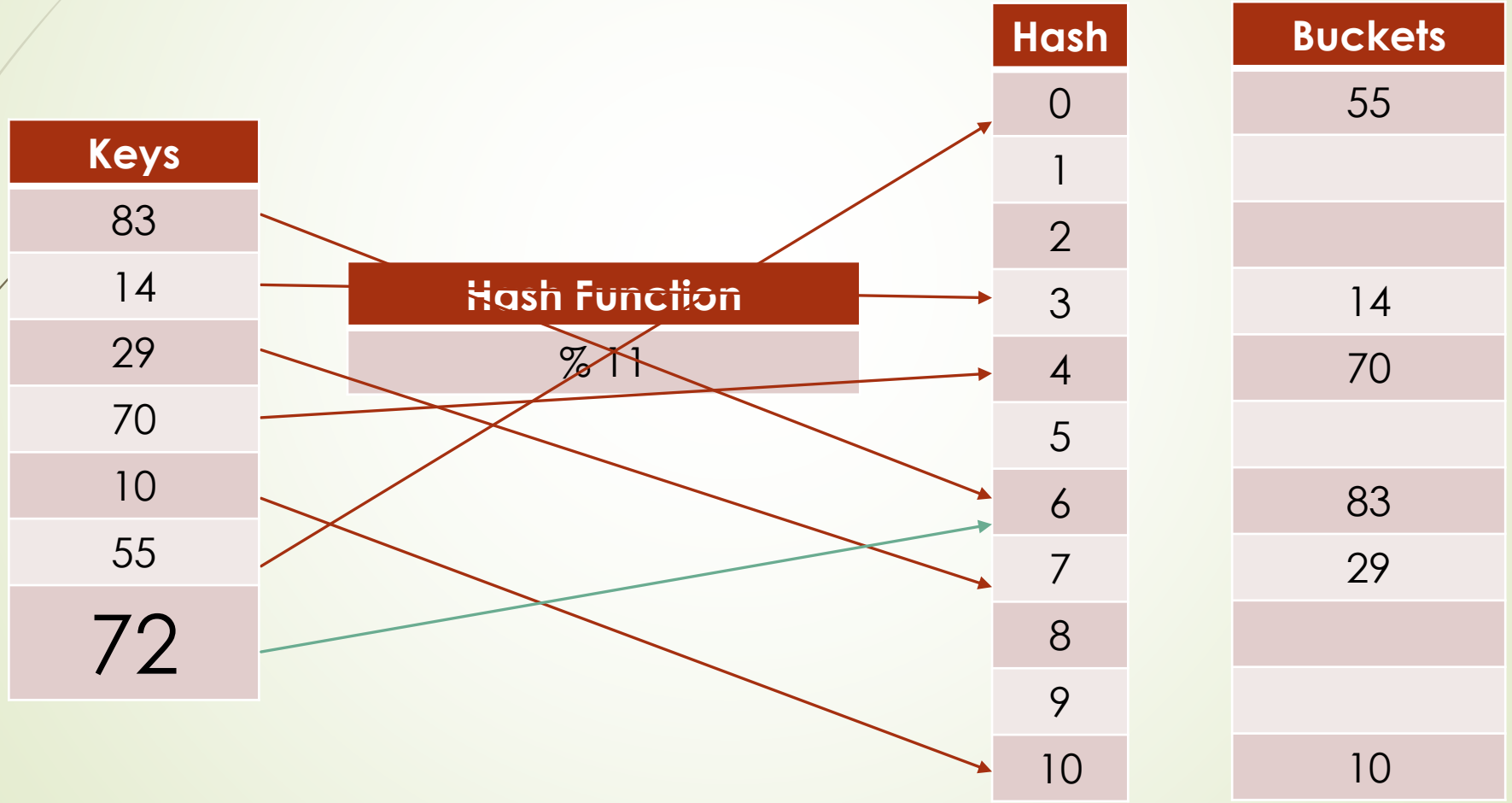
- Probability that n people don't have the same
  - birthday: p = (364/365)*(363/365)*...*((365-n+1)/365)
    - When n > 36, p < 0.167.
    - This means when n > 36, we have 83.3% chances that at least two people share the same birthday!

Solutions for Collision

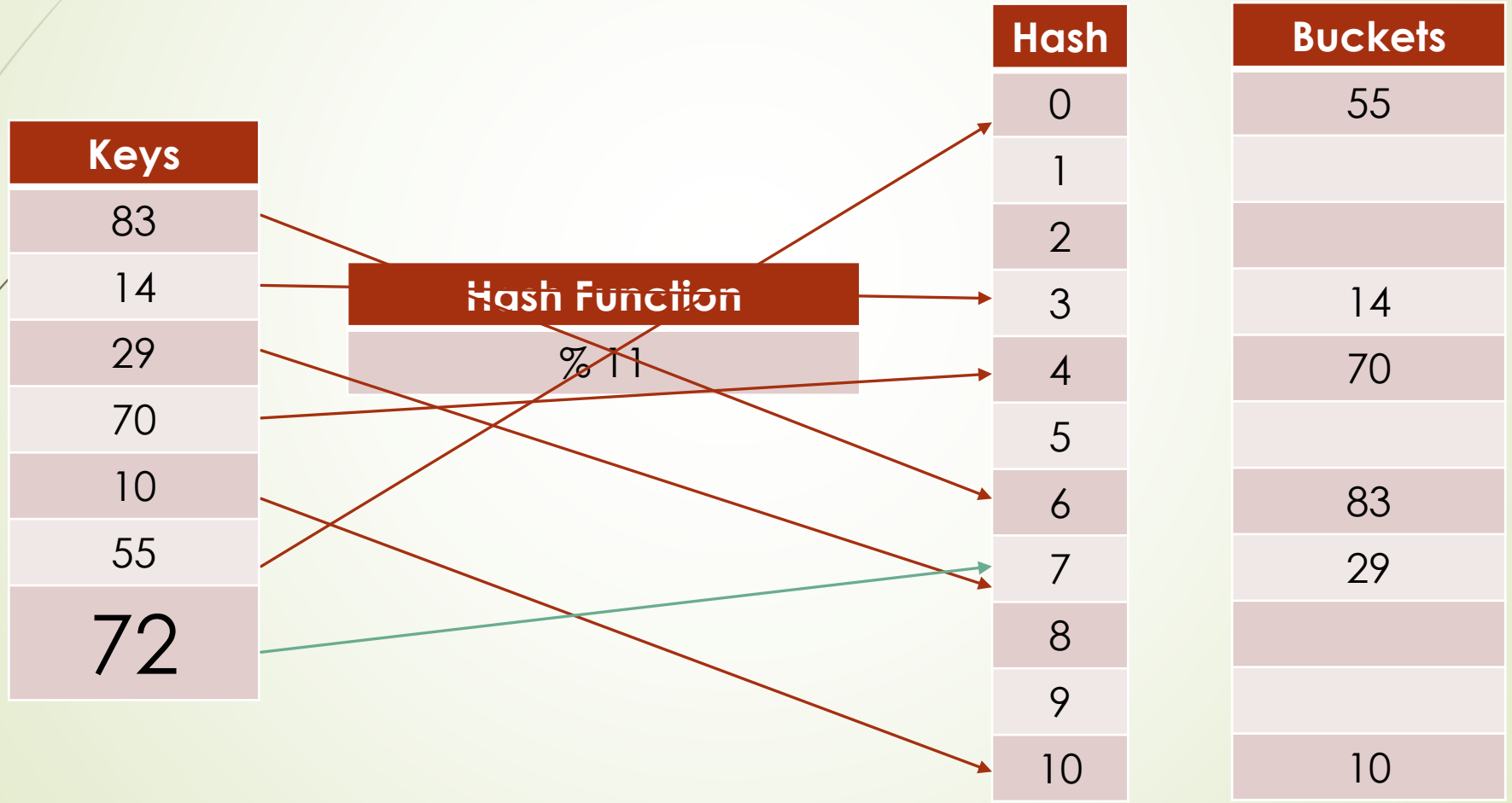
- Linear Probing
  - During insert of key k to position p:
    - If position p contains a different key, then examine positions p+1, p+2, etc.* until an empty position is found and insert k there.
  - During a search for key k at position p:
    - If position p contains a different key, then examine positions p+1, p+2, etc.* until either the key is found or an unused position is encountered.


keys
buckets
John Smith
Lisa Smith
Sam Doe
Sandra Dee
Ted Baker
000
001
002
151
152
153
154
155
253
254
255
Lisa Smith
521-8976
John Smith
521-1234
Sandra Dee
521-9655
Ted Baker
418-4165
Sam Doe
521-5030

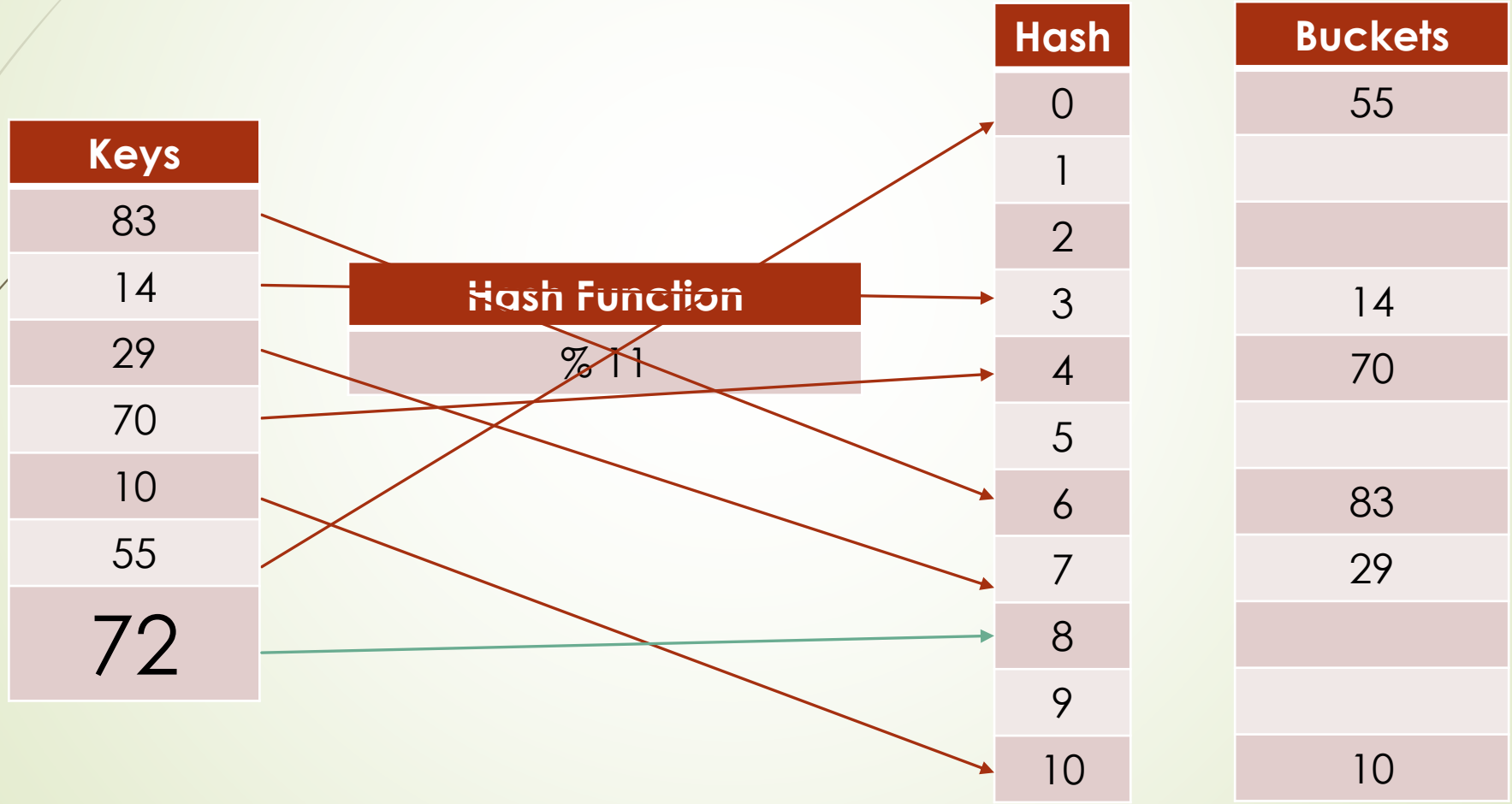
Linear Probing



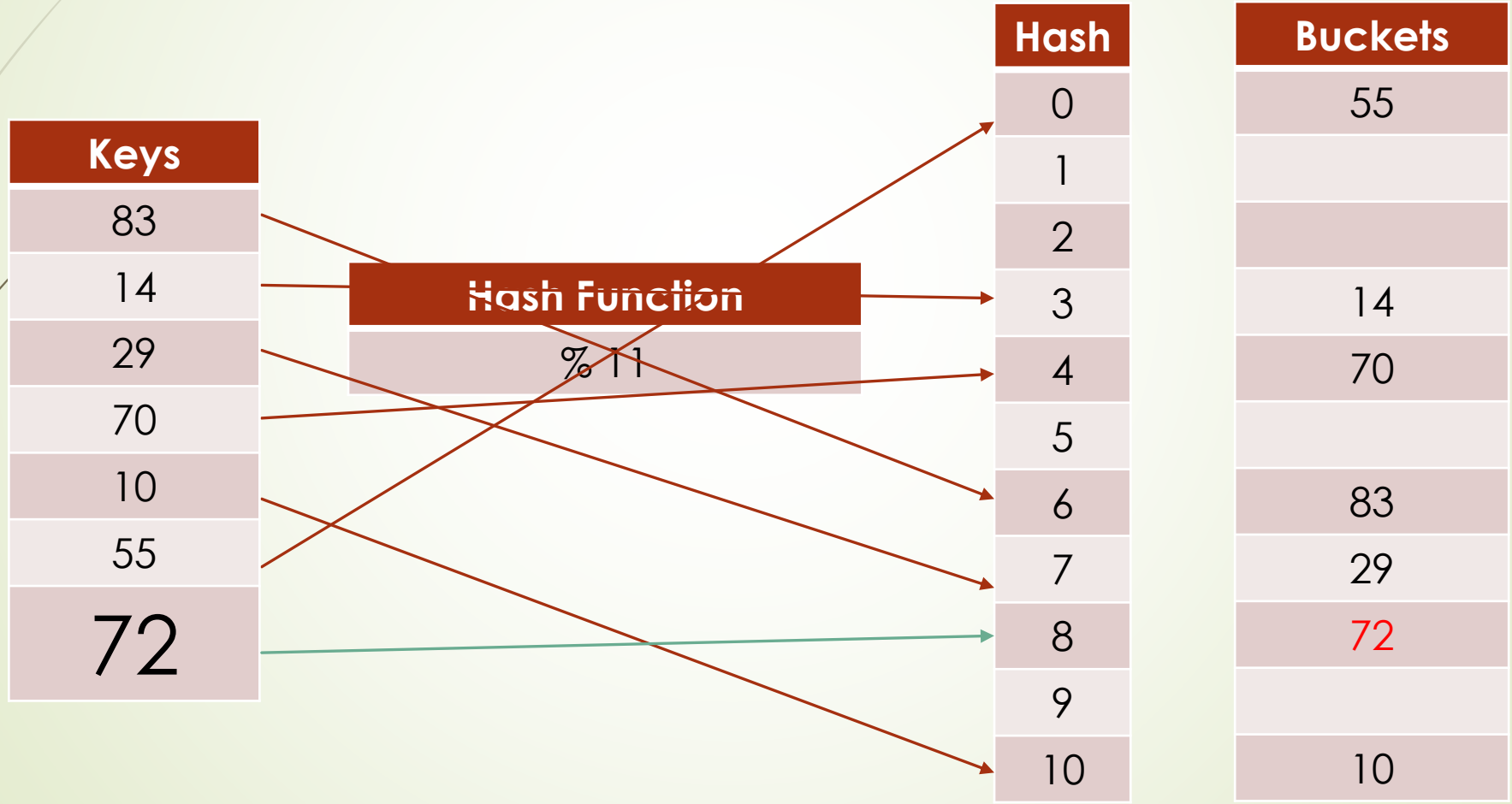
Linear Probing



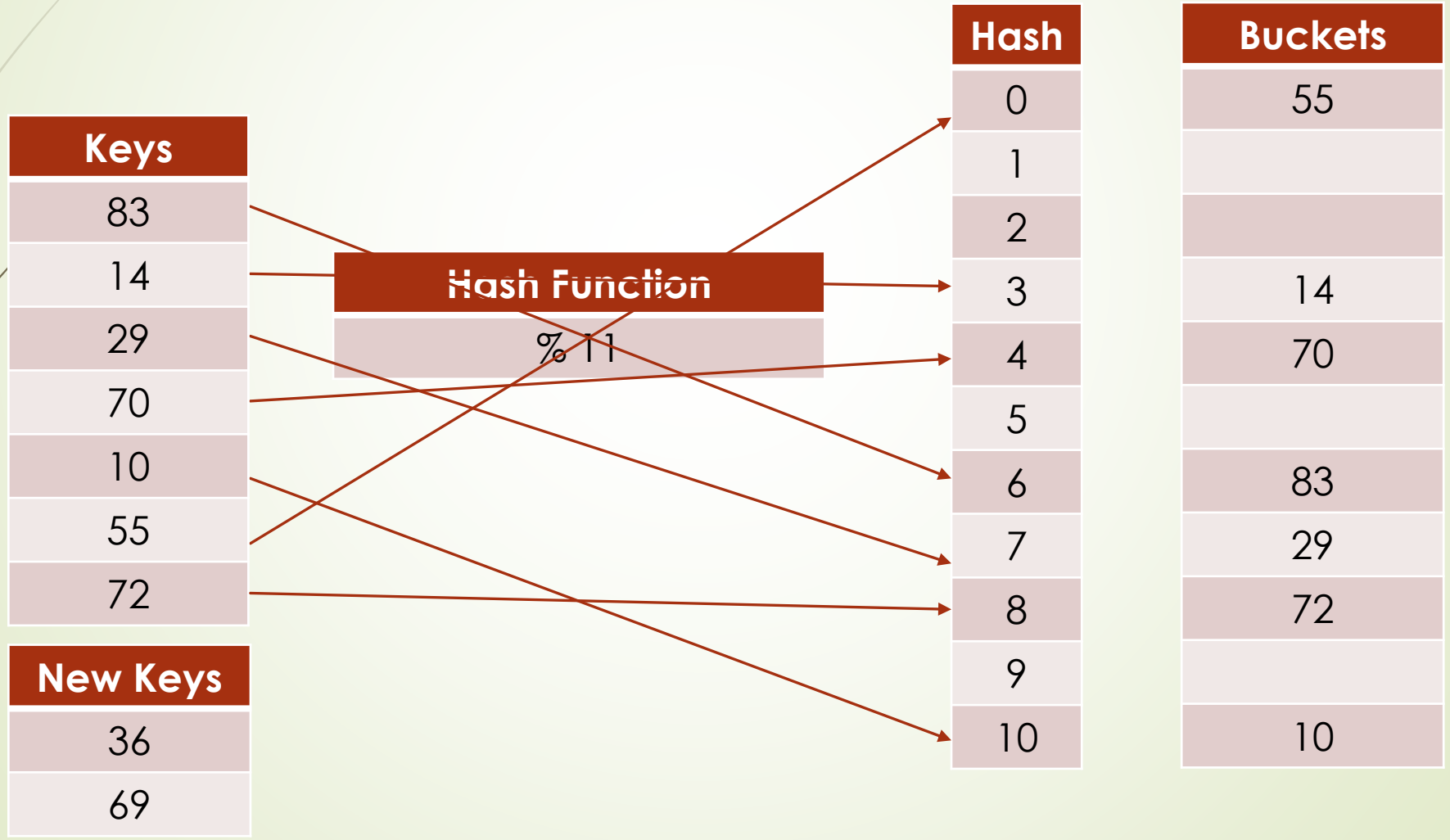
Linear Probing



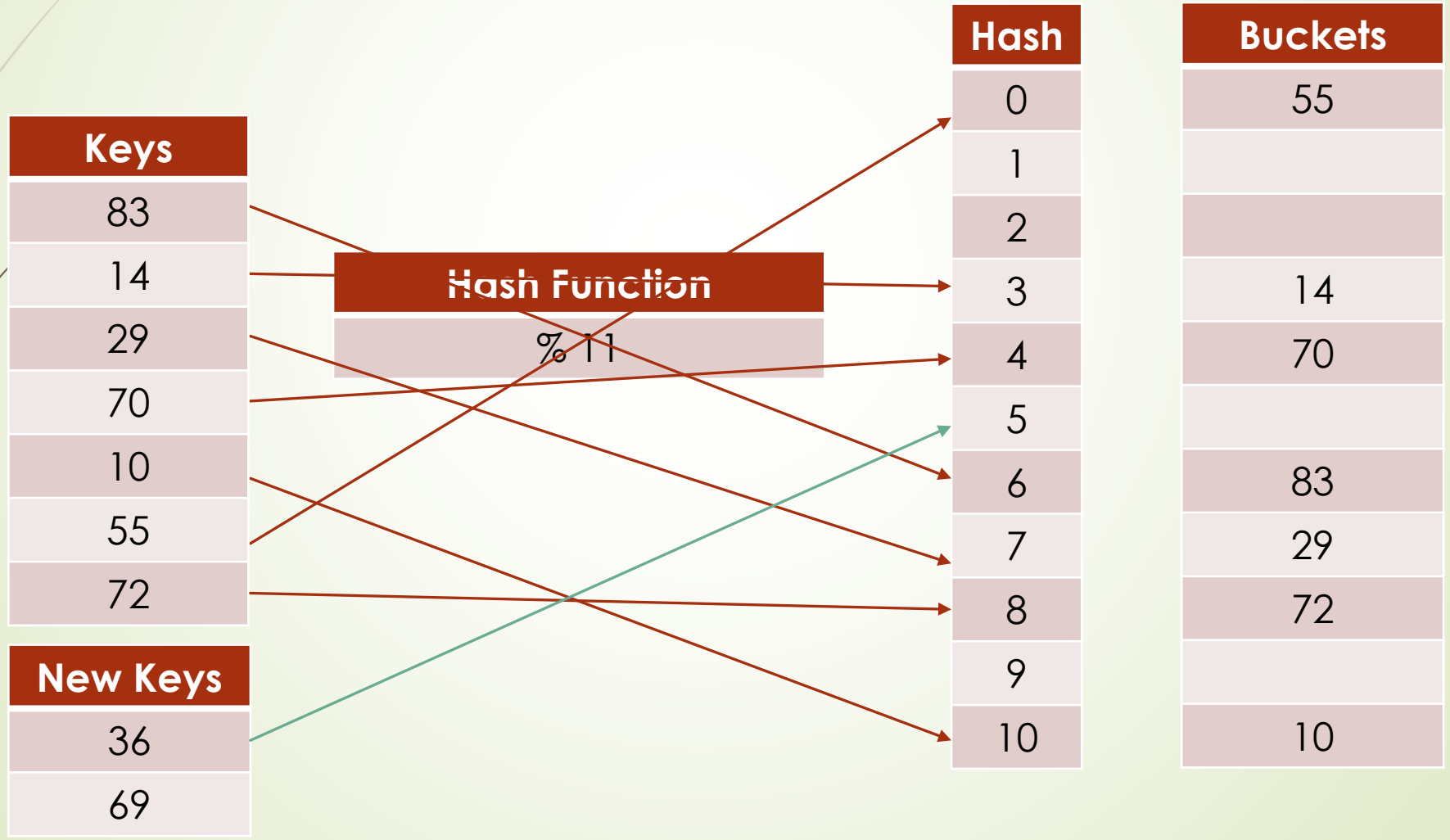
Linear Probing



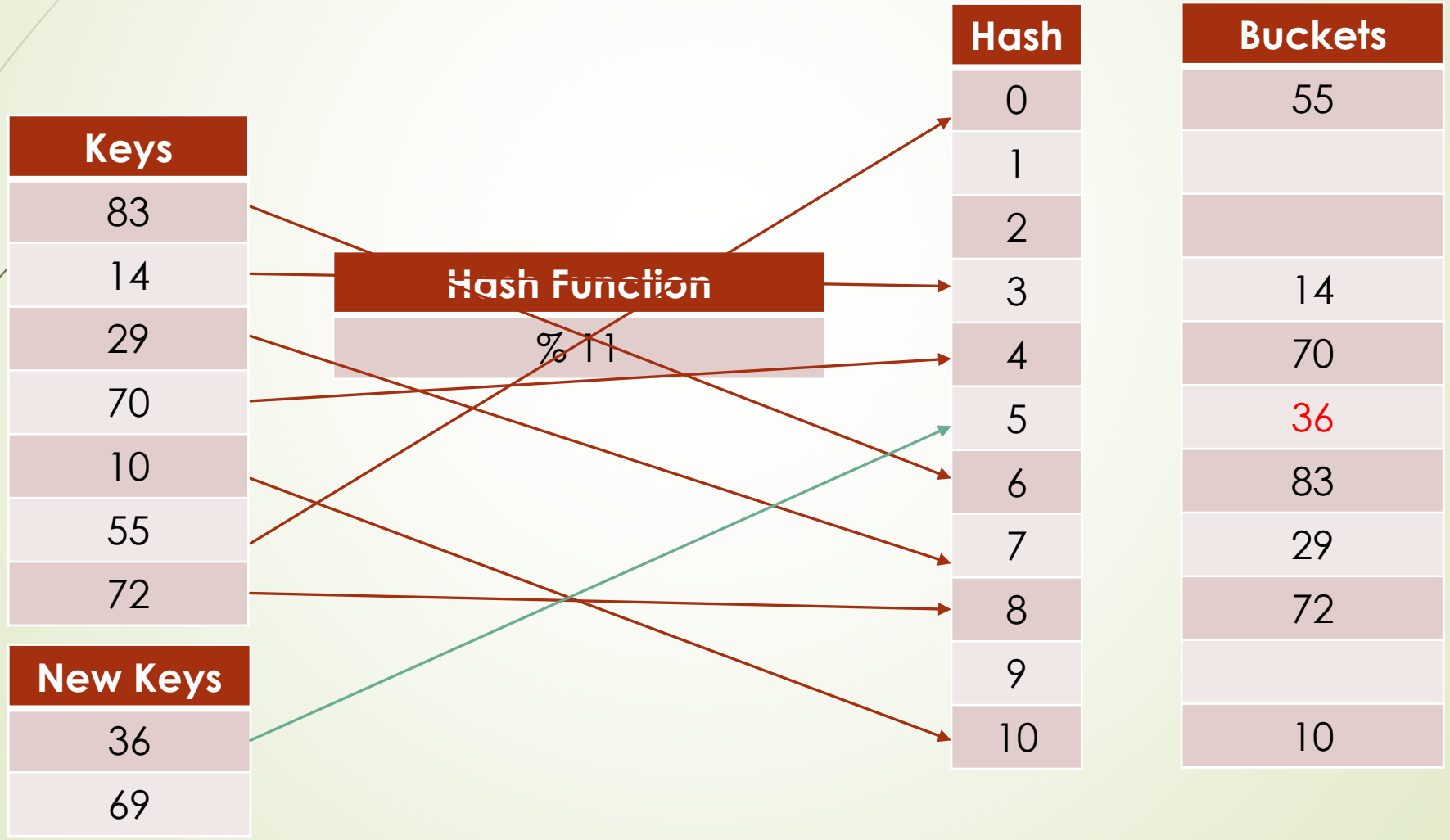
Linear Probing



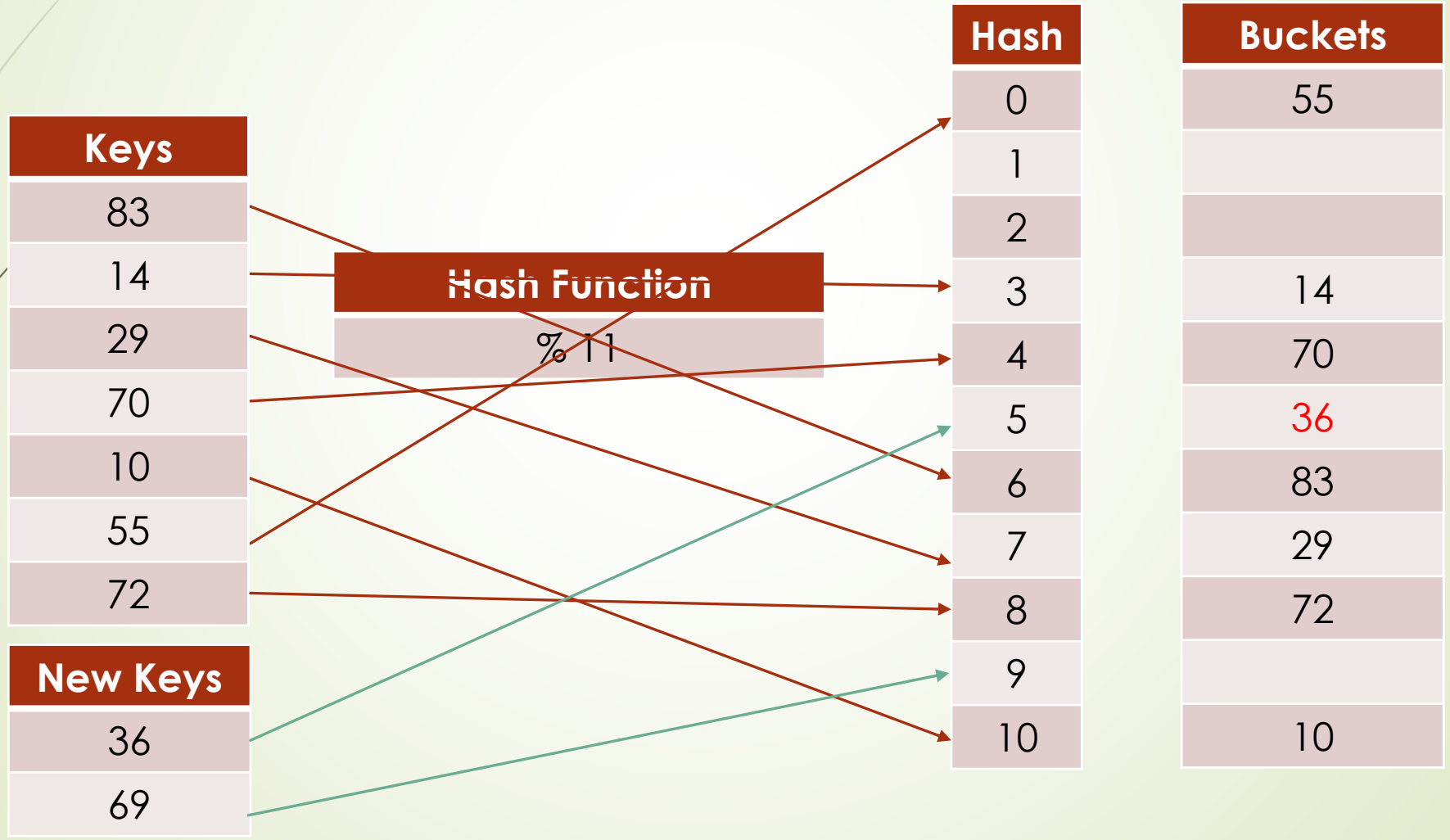
Linear Probing



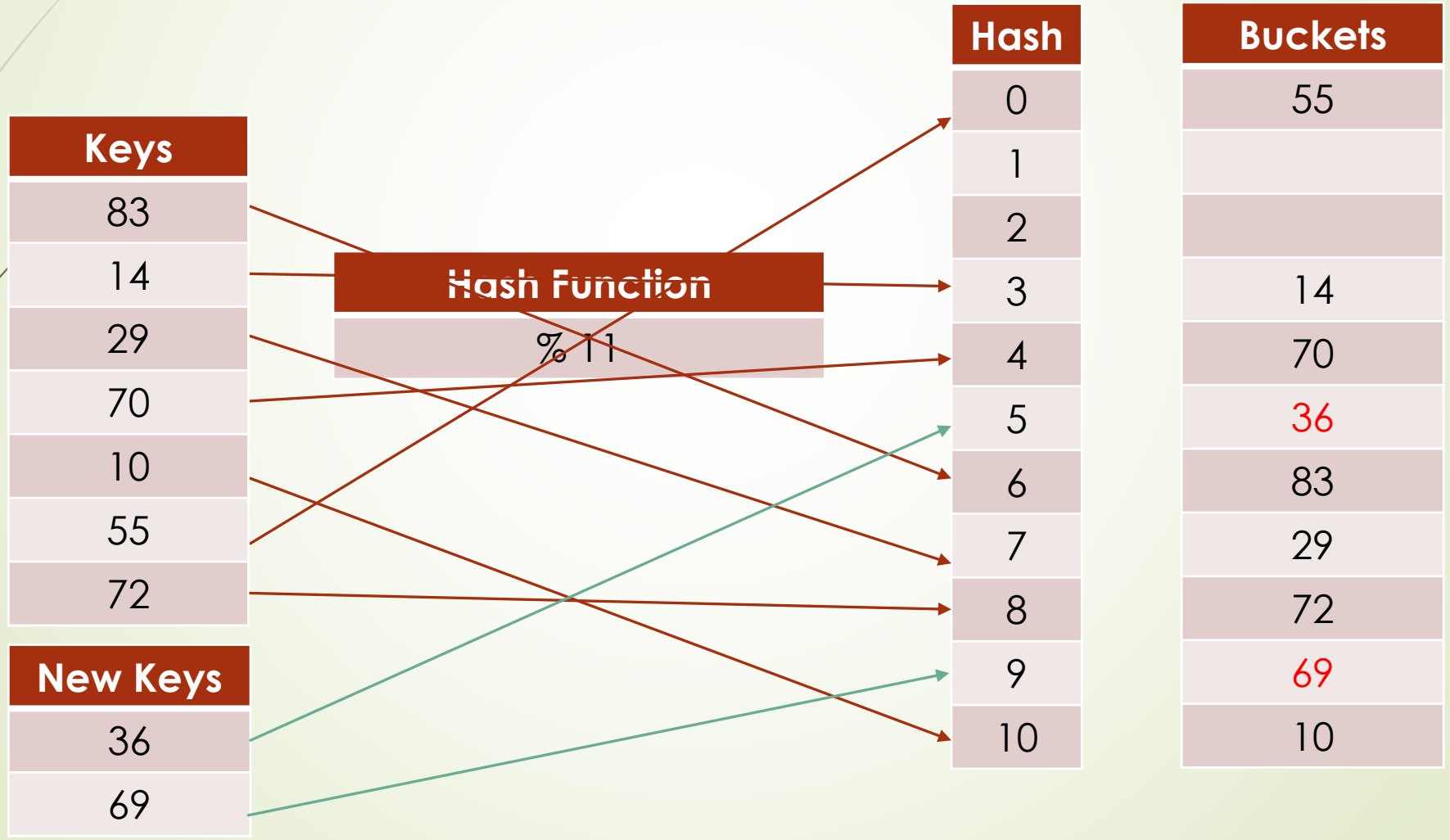
Linear Probing



Linear Probing



Linear Probing



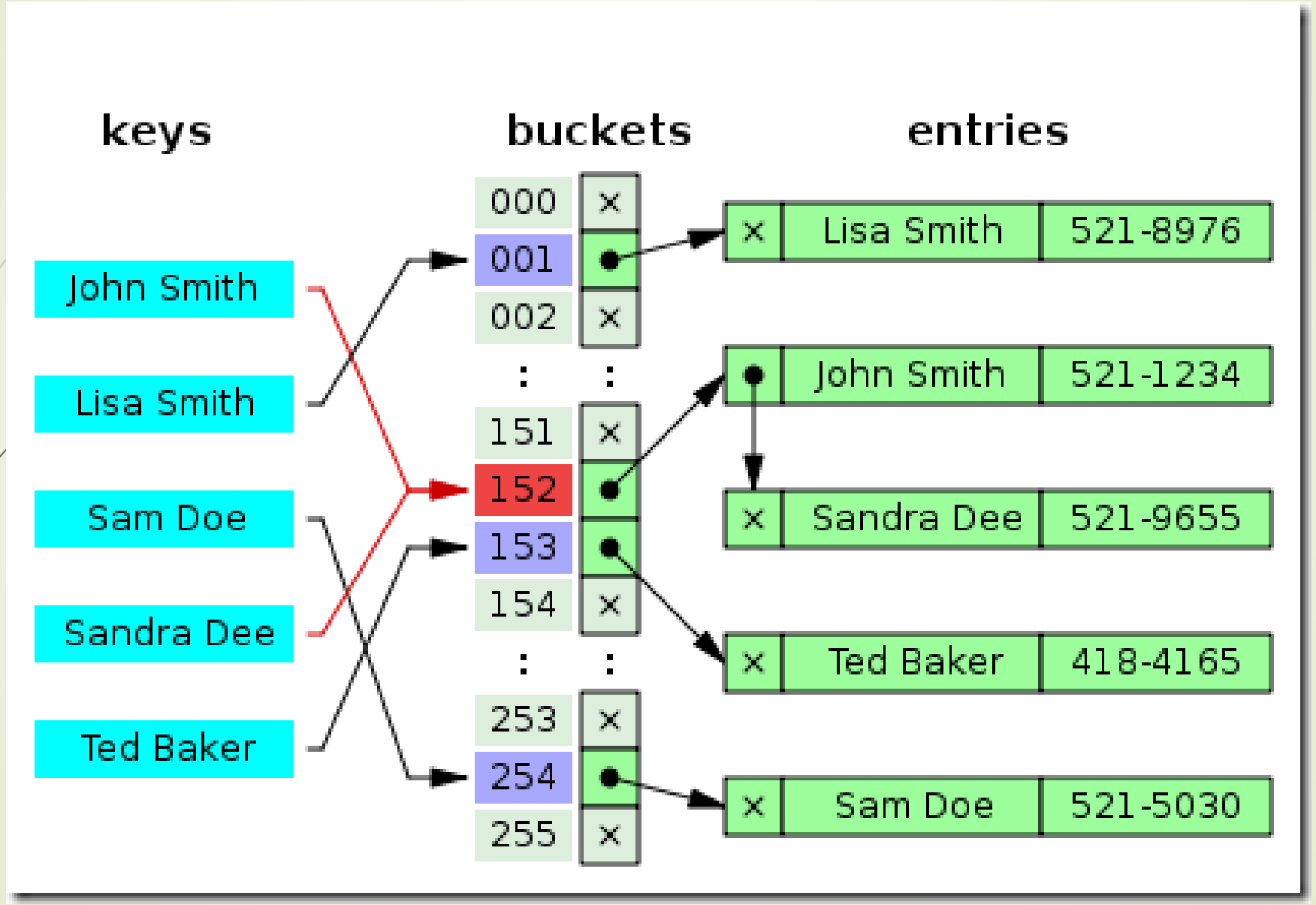
Linear Probing

- A potential problem with linear probing is **clustering**, where collisions that are resolved with linear probing cause groups of consecutive locations in the hash table to be occupied.

Chained Hashing

- A simple and efficient way for dealing with collisions is to have each bucket $A[i]$ store a list of (k, v) pairs with $h(k) = i$.

- We can store more elements than the table size by using chained hashing.
  - Each array position in the hash table is a head reference to a linked list of keys (a "bucket").
  - All colliding keys that hash to an array position are inserted to that bucket.


keys
buckets
entries
John Smith
Lisa Smith
Sam Doe
Sandra Dee
Ted Baker
000
001
002
151
152
153
154
253
254
255
Lisa Smith
521-8976
John Smith
521-1234
Sandra Dee
521-9655
Ted Baker
418-4165
Sam Doe
521-5030

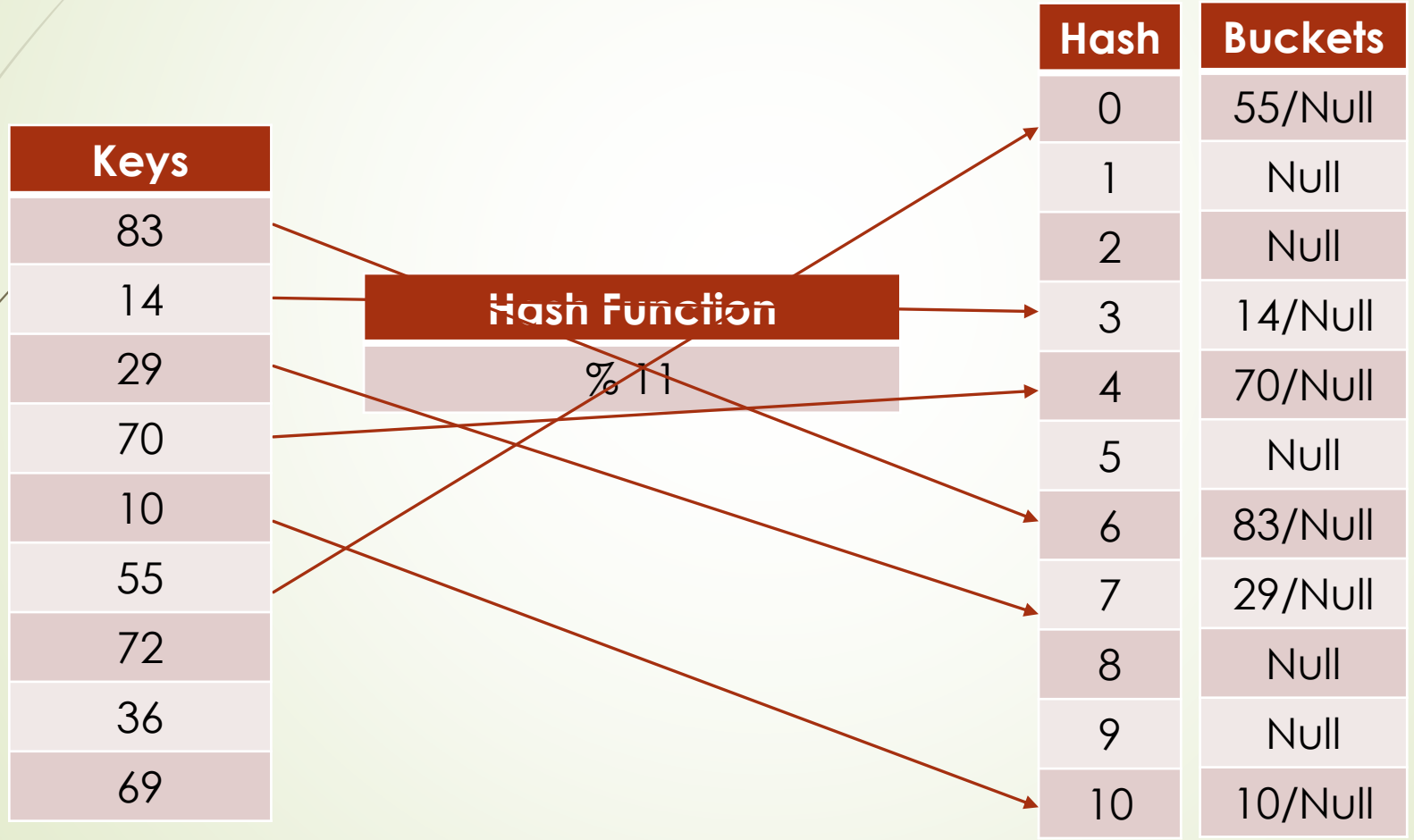
Chaining

Keys
83
14
29
70
10
55
72
36
69

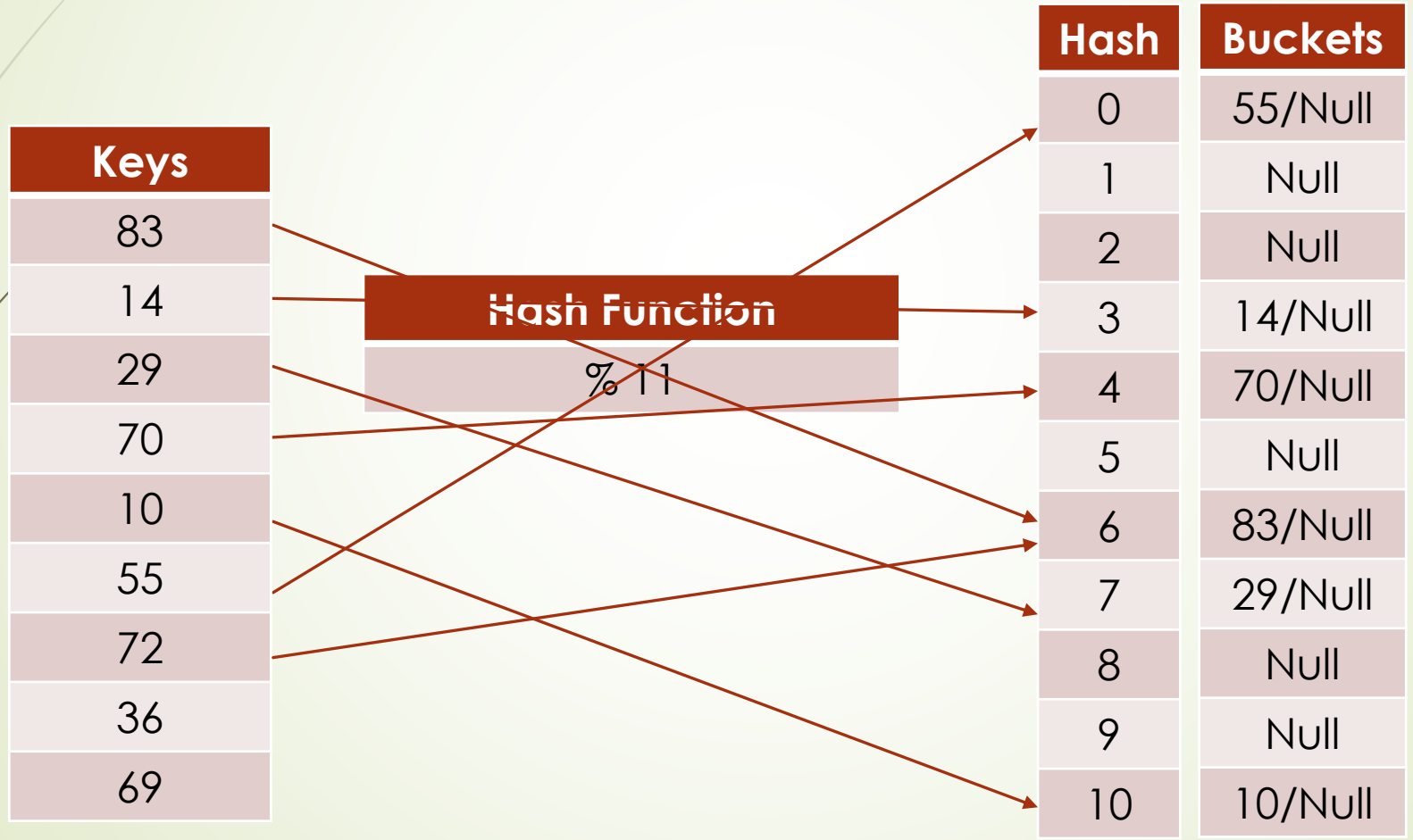
Hash Function
% 11

Hash	Buckets
0	Null
1	Null
2	Null
3	Null
4	Null
5	Null
6	Null
7	Null
8	Null
9	Null
10	Null

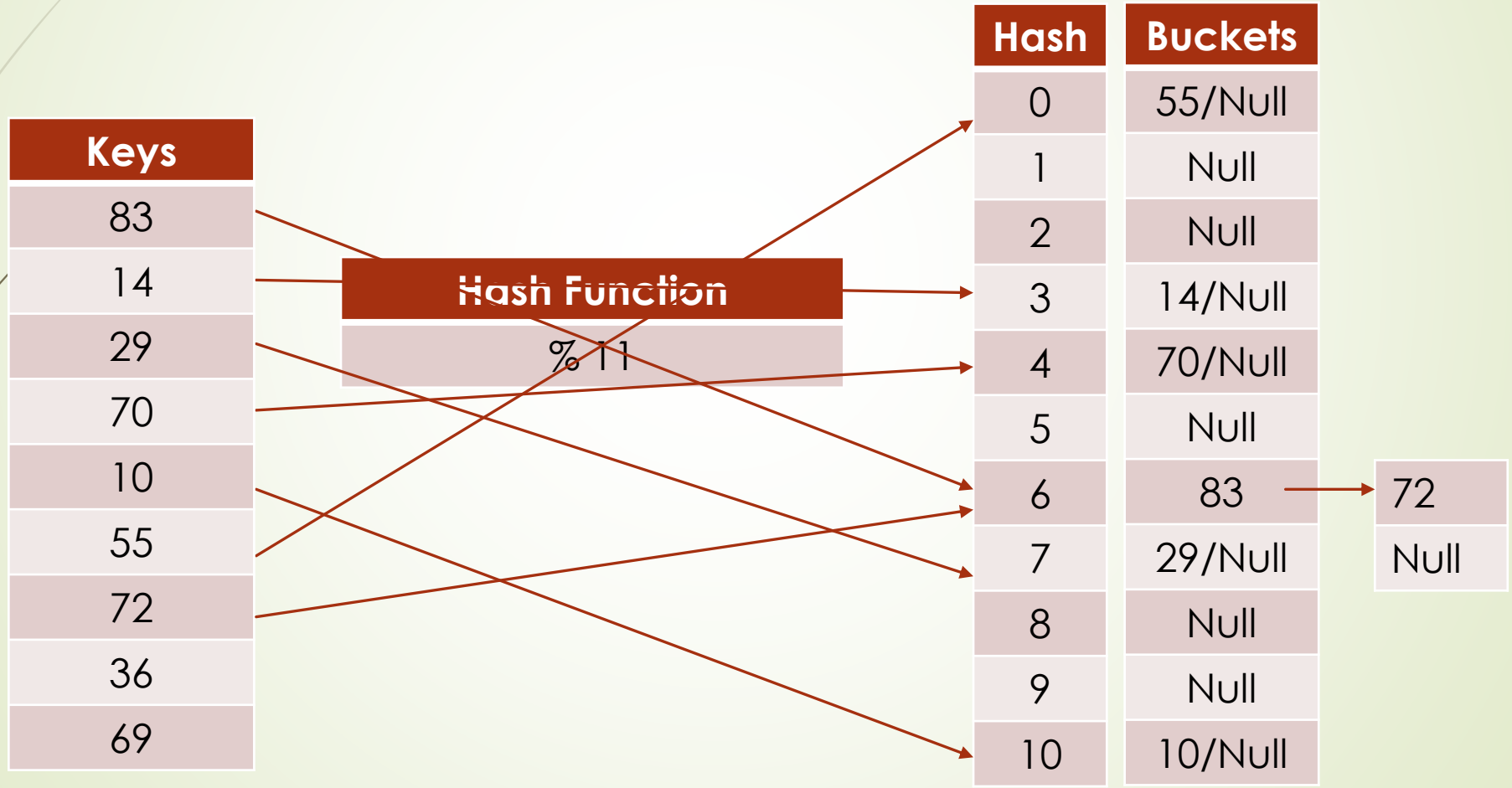
Chaining


Keys
83
14
29
70
10
55
72
36
69
Hash Function
% 11
Hash
Buckets
0
55/Null
1
Null
2
Null
3
14/Null
4
70/Null
5
Null
6
83/Null
7
29/Null
8
Null
9
Null
10
10/Null

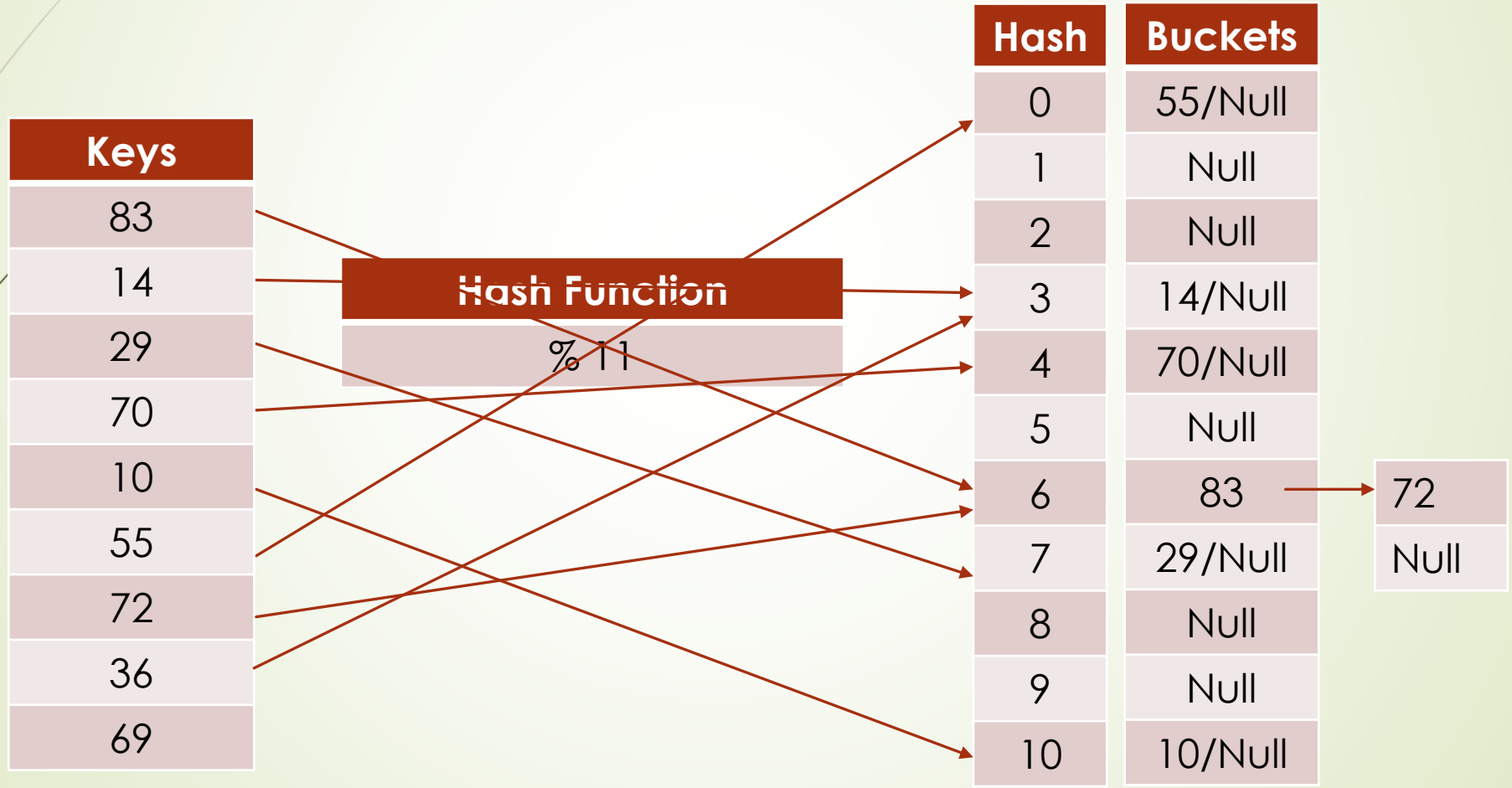
Chaining


Keys
83
14
29
70
10
55
72
36
69
Hash Function
% 11
Hash
Buckets
0
55/Null
1
Null
2
Null
3
14/Null
4
70/Null
5
Null
6
83/Null
7
29/Null
8
Null
9
Null
10
10/Null

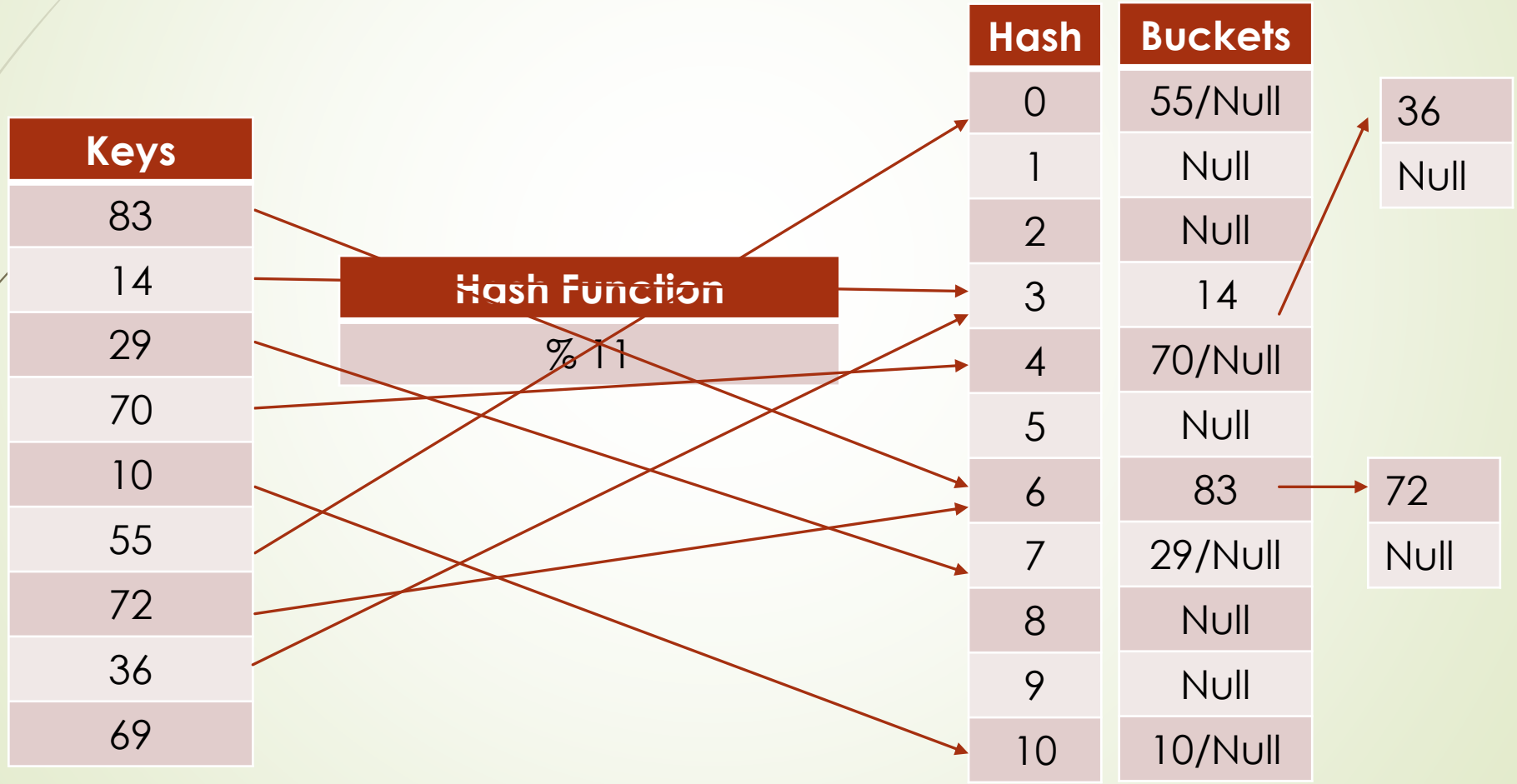
Chaining


Keys
83
14
29
70
10
55
72
36
69
Hash Function
% 11
Hash
Buckets
0
55/Null
1
Null
2
Null
3
14/Null
4
70/Null
5
Null
6
83
72
7
29/Null
Null
8
Null
9
Null
10
10/Null

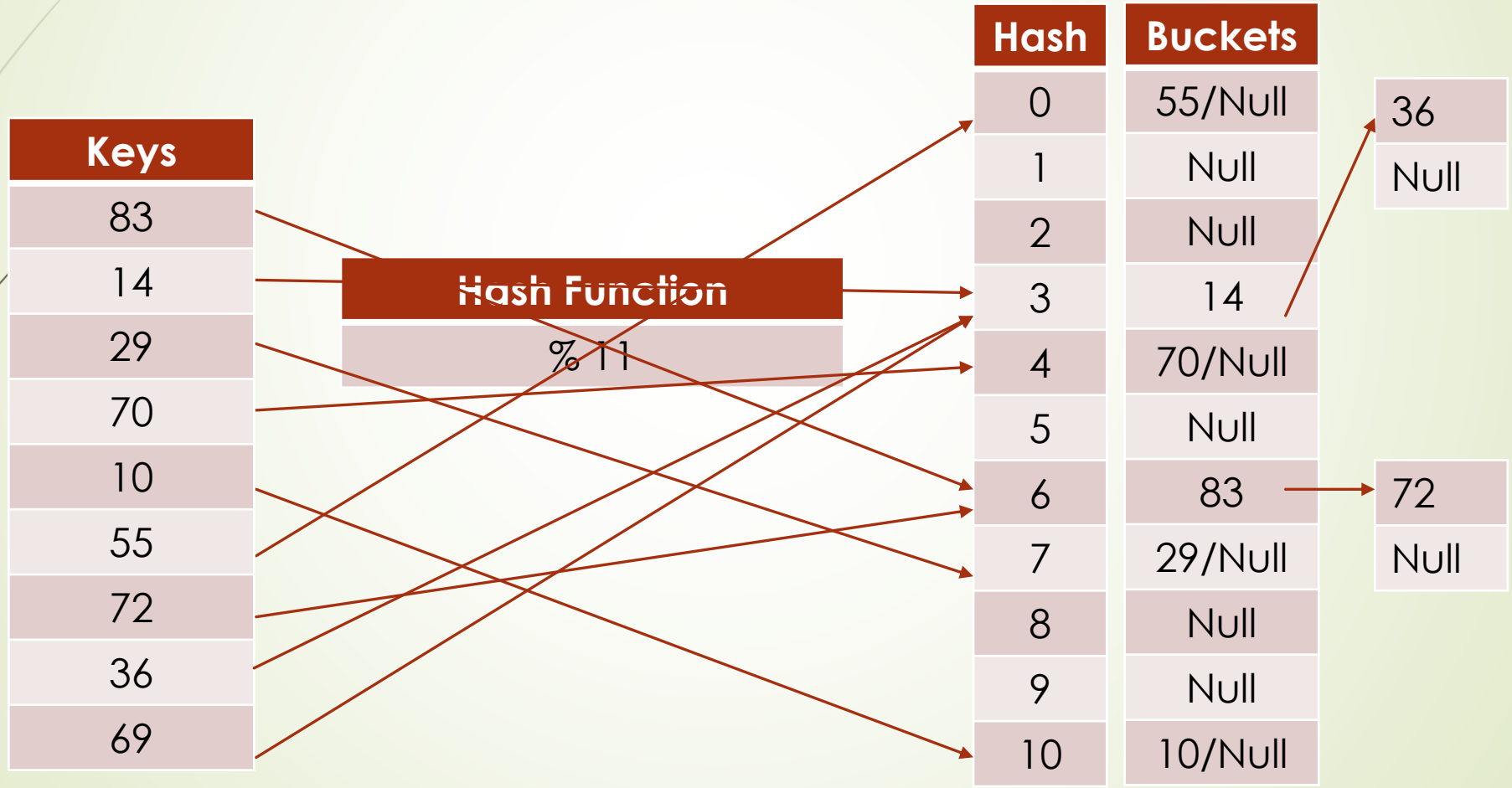
Chaining


Keys
83
14
29
70
10
55
72
36
69
Hash Function
% 11
Hash
Buckets
0
55/Null
1
Null
2
Null
3
14/Null
4
70/Null
5
Null
6
83
72
7
29/Null
Null
8
Null
9
Null
10
10/Null

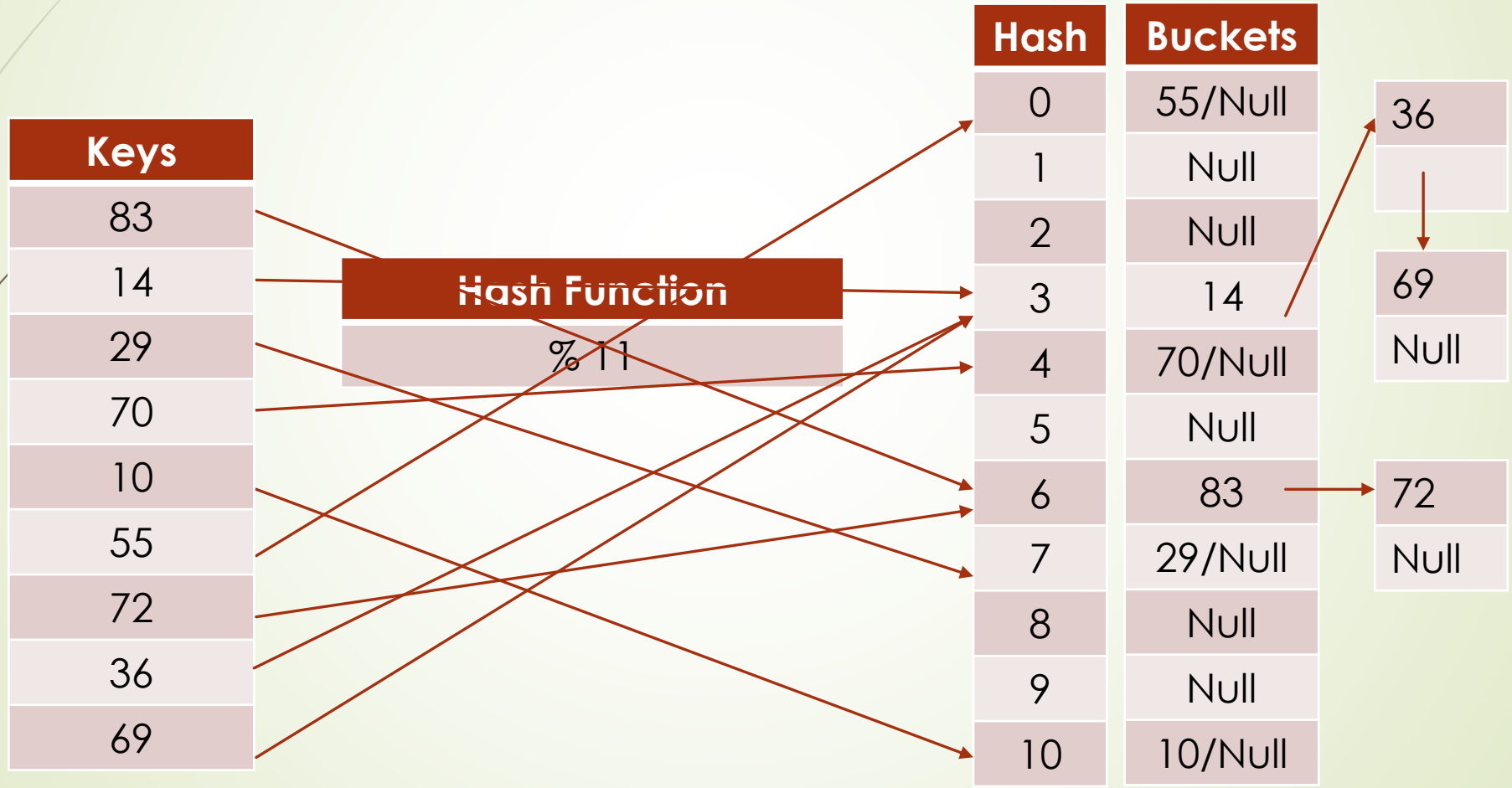
Chaining


Keys
83
14
29
70
10
55
72
36
69
Hash Function
% 11
Hash
0
1
2
3
4
5
6
7
8
9
10
Buckets
55/Null
Null
Null
14
70/Null
Null
83
29/Null
Null
Null
10/Null
36
Null
72
Null

Chaining


Keys
83
14
29
70
10
55
72
36
69
Hash Function
% 11
Hash
0
1
2
3
4
5
6
7
8
9
10
Buckets
55/Null
Null
Null
14
70/Null
Null
83
29/Null
Null
Null
10/Null
36
Null
72
Null

Chaining


Keys
83
14
29
70
10
55
72
36
69
Hash Function
% 11
Hash
0
1
2
3
4
5
6
7
8
9
10
Buckets
55/Null
Null
Null
14
70/Null
Null
83
29/Null
Null
Null
10/Null
36
69
Null
72
Null

Number of Key comparison for a Successful Search

- Largest

- Average

The End

Hash Codes for Strings

Hash Codes for Strings

- Each character in a string has a unicode (int) value.
  - 'A'=65 'B'=66 'C'=67, ..., 'a'=97, 'b'=98, 'c'=99, ...

Hash Codes for Strings

- Summing up
  - Act=?

064 @	080 P	096 `	112 p
065 A	081 Q	097 a	113 q
066 B	082 R	098 b	114 r
067 C	083 S	099 c	115 s
068 D	084 T	100 d	116 t
069 E	085 U	101 e	117 u
070 F	086 V	102 f	118 v
071 G	087 W	103 g	119 w
072 H	088 X	104 h	120 x
073 I	089 Y	105 i	121 y
074 J	090 Z	106 j	122 z
075 K	091 [	107 k	123 {
076 L	092 \	108 l	124 |
077 M	093]	109 m	125 }
078 N	094 ^	110 n	126 ~
079 O	095 _	111 o	127 ⌂

Hash Codes for Strings

- Summing up
  - Act=65+99+116=280

064 @	080 P	096 `	112 p
065 A	081 Q	097 a	113 q
066 B	082 R	098 b	114 r
067 C	083 S	099 c	115 s
068 D	084 T	100 d	116 t
069 E	085 U	101 e	117 u
070 F	086 V	102 f	118 v
071 G	087 W	103 g	119 w
072 H	088 X	104 h	120 x
073 I	089 Y	105 i	121 y
074 J	090 Z	106 j	122 z
075 K	091 [	107 k	123 {
076 L	092 \	108 l	124 |
077 M	093]	109 m	125 }
078 N	094 ^	110 n	126 ~
079 O	095 _	111 o	127 ⌂

Hash Codes for Strings

- Summing up
  - Act=65+99+116=280
  - Boy=66+111+121=298

064 @	080 P	096 `	112 p
065 A	081 Q	097 a	113 q
066 B	082 R	098 b	114 r
067 C	083 S	099 c	115 s
068 D	084 T	100 d	116 t
069 E	085 U	101 e	117 u
070 F	086 V	102 f	118 v
071 G	087 W	103 g	119 w
072 H	088 X	104 h	120 x
073 I	089 Y	105 i	121 y
074 J	090 Z	106 j	122 z
075 K	091 [	107 k	123 {
076 L	092 \	108 l	124 |
077 M	093]	109 m	125 }
078 N	094 ^	110 n	126 ~
079 O	095 _	111 o	127 ⌂

Hash Codes for Strings

- Summing up
  - Act=65+99+116=280
  - Boy=66+111+121=298
  - Cat=?
  - Ads=?

064 @	080 P	096 `	112 p
065 A	081 Q	097 a	113 q
066 B	082 R	098 b	114 r
067 C	083 S	099 c	115 s
068 D	084 T	100 d	116 t
069 E	085 U	101 e	117 u
070 F	086 V	102 f	118 v
071 G	087 W	103 g	119 w
072 H	088 X	104 h	120 x
073 I	089 Y	105 i	121 y
074 J	090 Z	106 j	122 z
075 K	091 [	107 k	123 {
076 L	092 \	108 l	124 |
077 M	093]	109 m	125 }
078 N	094 ^	110 n	126 ~
079 O	095 _	111 o	127 ⌂

Hash Codes for Strings

- Summing up
  - Act=65+99+116=280
  - Boy=66+111+121=298
  - Cat=280
  - Ads=280

064 @	080 P	096 `	112 p
065 A	081 Q	097 a	113 q
066 B	082 R	098 b	114 r
067 C	083 S	099 c	115 s
068 D	084 T	100 d	116 t
069 E	085 U	101 e	117 u
070 F	086 V	102 f	118 v
071 G	087 W	103 g	119 w
072 H	088 X	104 h	120 x
073 I	089 Y	105 i	121 y
074 J	090 Z	106 j	122 z
075 K	091 [	107 k	123 {
076 L	092 \	108 l	124 |
077 M	093]	109 m	125 }
078 N	094 ^	110 n	126 ~
079 O	095 _	111 o	127 ⌂

Hash Codes for Strings

- Summing up the int values of the characters can lead to a lot of collisions.

- Hash functions are designed to be
  - fast and
  - yield few hash collisions in expected input domains.

Hash Codes for Strings

- Polynomial hash codes
  - We choose a nonzero constant, a != 1, and calculate $(x_0a^{k-1}+ x_1a^{k-2}+ ...+ x_{k-2}a+ x_{k-1})$ as the hash code, ignoring overflows
  - By using the Horner's rule, default a = 31

Hash Codes for Strings

- Formular
  - $h = s[0]*31^{k-1} + \ldots + s[k-3]*31^{2} + s[k-2]*31^{1} + s[k-1]*31^{0}$

Hash Codes for Strings

- Formular
  - $h = s[0]*31^{k-1} + \ldots + s[k-3]*31^{2} + s[k-2]*31^{1} + s[k-1]*31^{0}$
- "Cal"

064 @	080 P	096 `	112 p
065 A	081 Q	097 a	113 q
066 B	082 R	098 b	114 r
067 C	083 S	099 c	115 s
068 D	084 T	100 d	116 t
069 E	085 U	101 e	117 u
070 F	086 V	102 f	118 v
071 G	087 W	103 g	119 w
072 H	088 X	104 h	120 x
073 I	089 Y	105 i	121 y
074 J	090 Z	106 j	122 z
075 K	091 [	107 k	123 {
076 L	092 \	108 l	124 |
077 M	093]	109 m	125 }
078 N	094 ^	110 n	126 ~
079 O	095 _	111 o	127 ⌂

Hash Codes for Strings

- Formular
  - $h = s[0]*31^{k-1} + \ldots + s[k-3]*31^{2} + s[k-2]*31^{1} + s[k-1]*31^{0}$
- "Cal"
  - $67 \cdot 31^{2} + 97 \cdot 31^{1} + 108 \cdot 31^{0} = 67502$

064 @	080 P	096 `	112 p
065 A	081 Q	097 a	113 q
066 B	082 R	098 b	114 r
067 C	083 S	099 c	115 s
068 D	084 T	100 d	116 t
069 E	085 U	101 e	117 u
070 F	086 V	102 f	118 v
071 G	087 W	103 g	119 w
072 H	088 X	104 h	120 x
073 I	089 Y	105 i	121 y
074 J	090 Z	106 j	122 z
075 K	091 [	107 k	123 {
076 L	092 \	108 l	124 |
077 M	093]	109 m	125 }
078 N	094 ^	110 n	126 ~
079 O	095 _	111 o	127 ⌂

- “CalStateBakersfield”

```
int GetHashCode(string str)
{
    int hash = 0;
    for (int i = 0; i < str.length(); i++)
    {
        hash = str[i] + (31 * hash);
    }
    return hash;
}
```

- String: "Act" "Cat" "Ads"
- hashcode: 65650 67510 67602

- Hash functions are designed to be
  - fast and
  - yield few hash collisions in expected input domains.