CSCI 232:
Data Structures and Algorithms

Hashing (Part 3)

Reese Pearsall

Spring 2024

Announcements

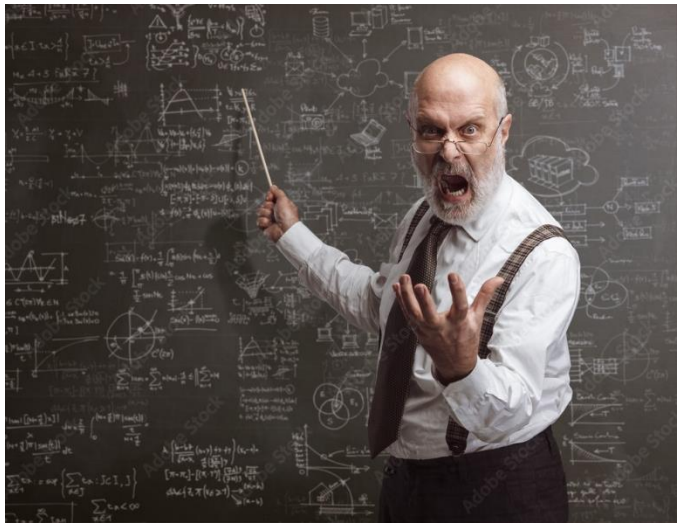
Lab 6 due **Friday** at 11:59 PM

Program 1 due **tonight** at 11:59 PM

No class on Thursday

Program 2 posted, due **Sunday** March 10th

Professor: “THIS CODE DOESN’T EVEN WORK. WHO WROTE THIS AWFUL CODE??”



Me, a first year CS student:

java.lang.StackOverflowError

java.lang.NullPointerException:

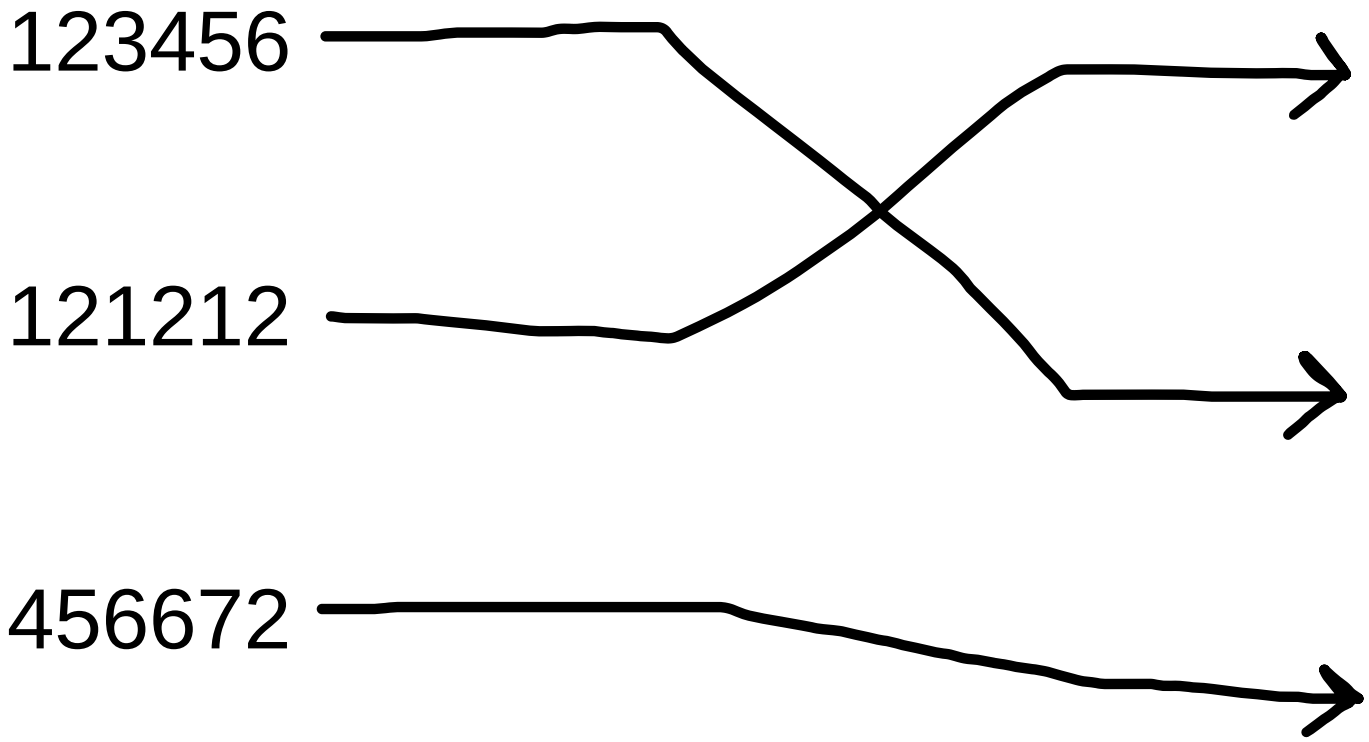
Syntax error, insert "}" to complete MethodBody

Hash Tables

Student ID

Hash Function

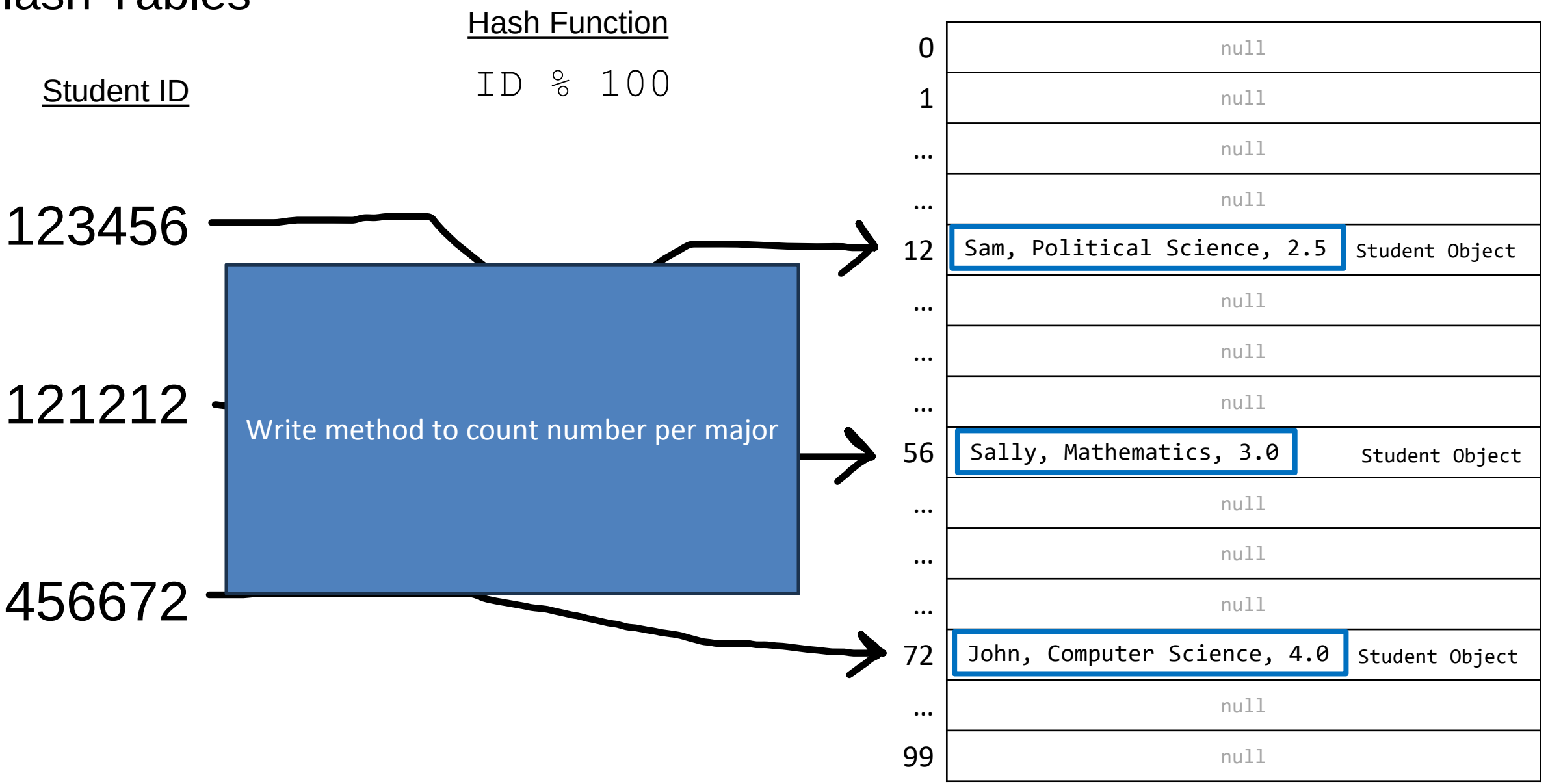
`ID % 100`



Student[] Array

0	null
1	null
…	null
…	null
12	Sam, Political Science, 2.5 — Student Object
…	null
…	null
…	null
56	Sally, Mathematics, 3.0 — Student Object
…	null
…	null
…	null
72	John, Computer Science, 4.0 — Student Object
…	null
99	null

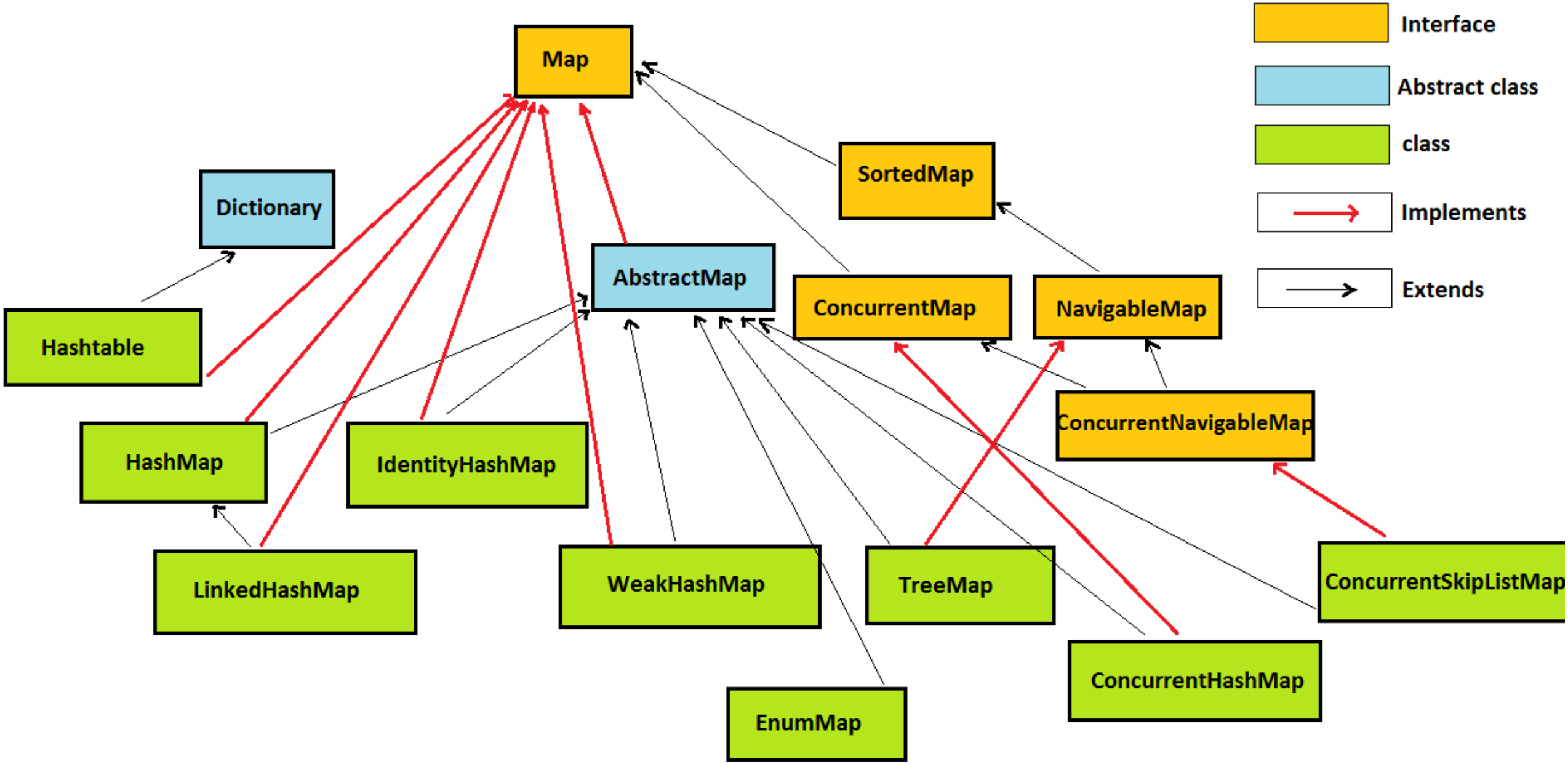
Hash Tables

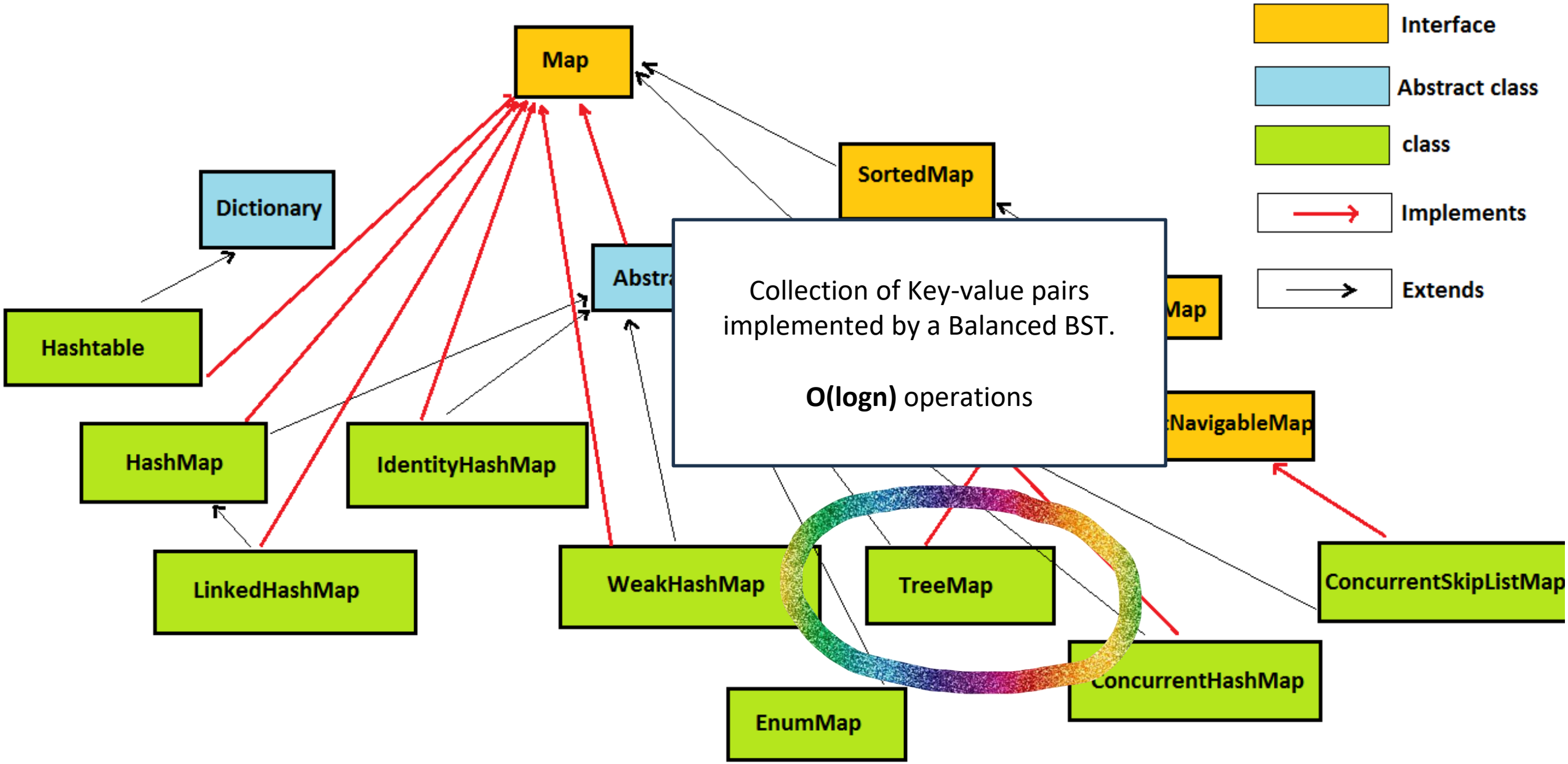

Hash Function
ID % 100
Student ID
123456
121212
456672
Write method to count number per major
Student[] Array
0
1
...
...
12
...
...
...
56
...
...
...
72
...
99
null
Sam, Political Science, 2.5
Student Object
Sally, Mathematics, 3.0
Student Object
John, Computer Science, 4.0
Student Object

Program 2

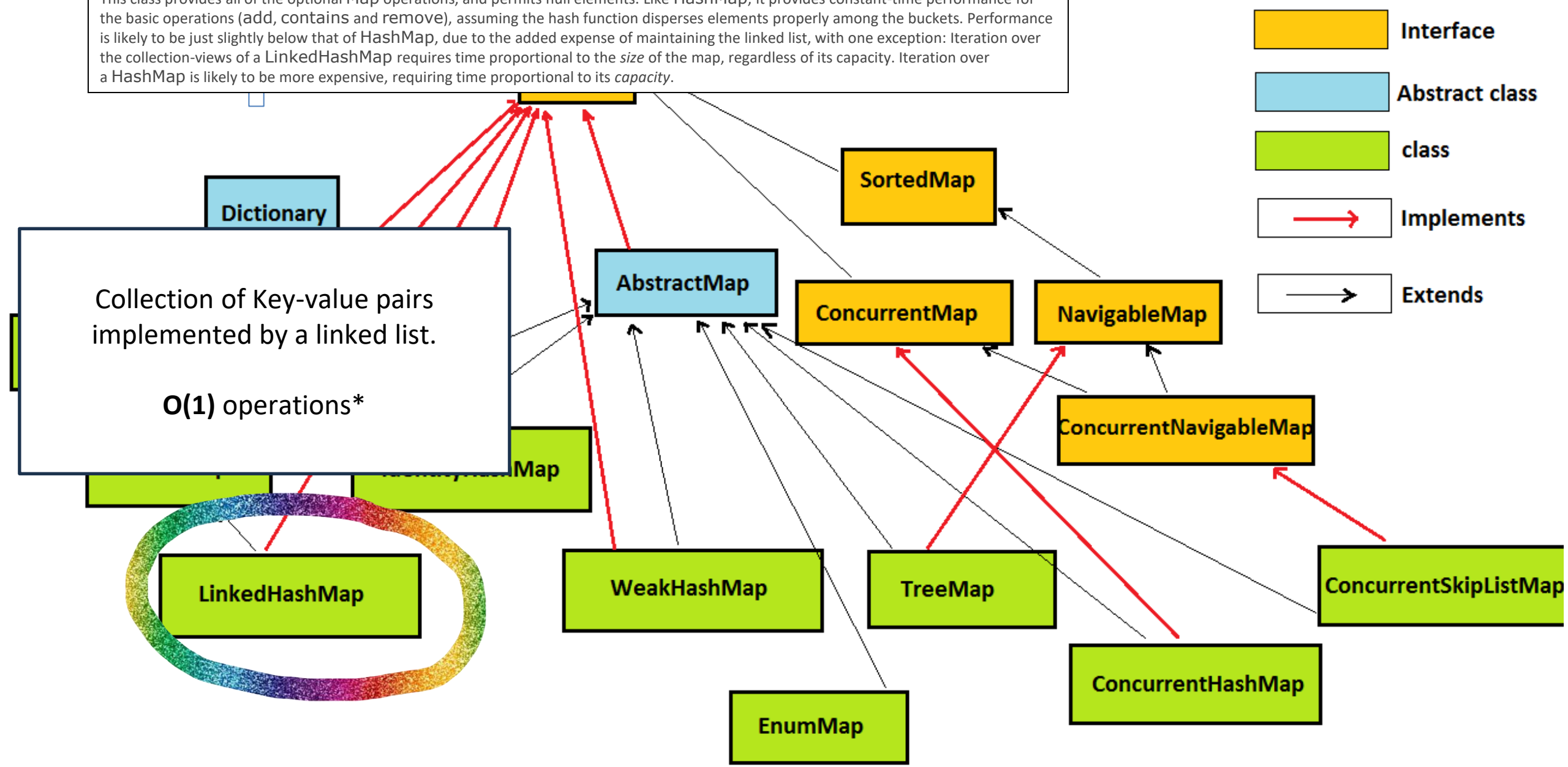
Hash Tables

Hash Set	Linked List	Array list
O(1) Insertion	O(1) Insertion*	O(n) Insertion
O(1) Delete by name	O(n) Delete by name	O(n) Delete by name
O(1) Contains	O(n) Contains	O(n) Contains
Not ordered	Ordered	Ordered
No duplicate values	Allow for Duplicate Values	Allow for Duplicate Values


Map
Dictionary
SortedMap
AbstractMap
ConcurrentMap
NavigableMap
Hashtable
HashMap
IdentityHashMap
ConcurrentNavigableMap
LinkedHashMap
WeakHashMap
TreeMap
ConcurrentSkipListMap
ConcurrentHashMap
EnumMap
Interface
Abstract class
class
Implements
Extends

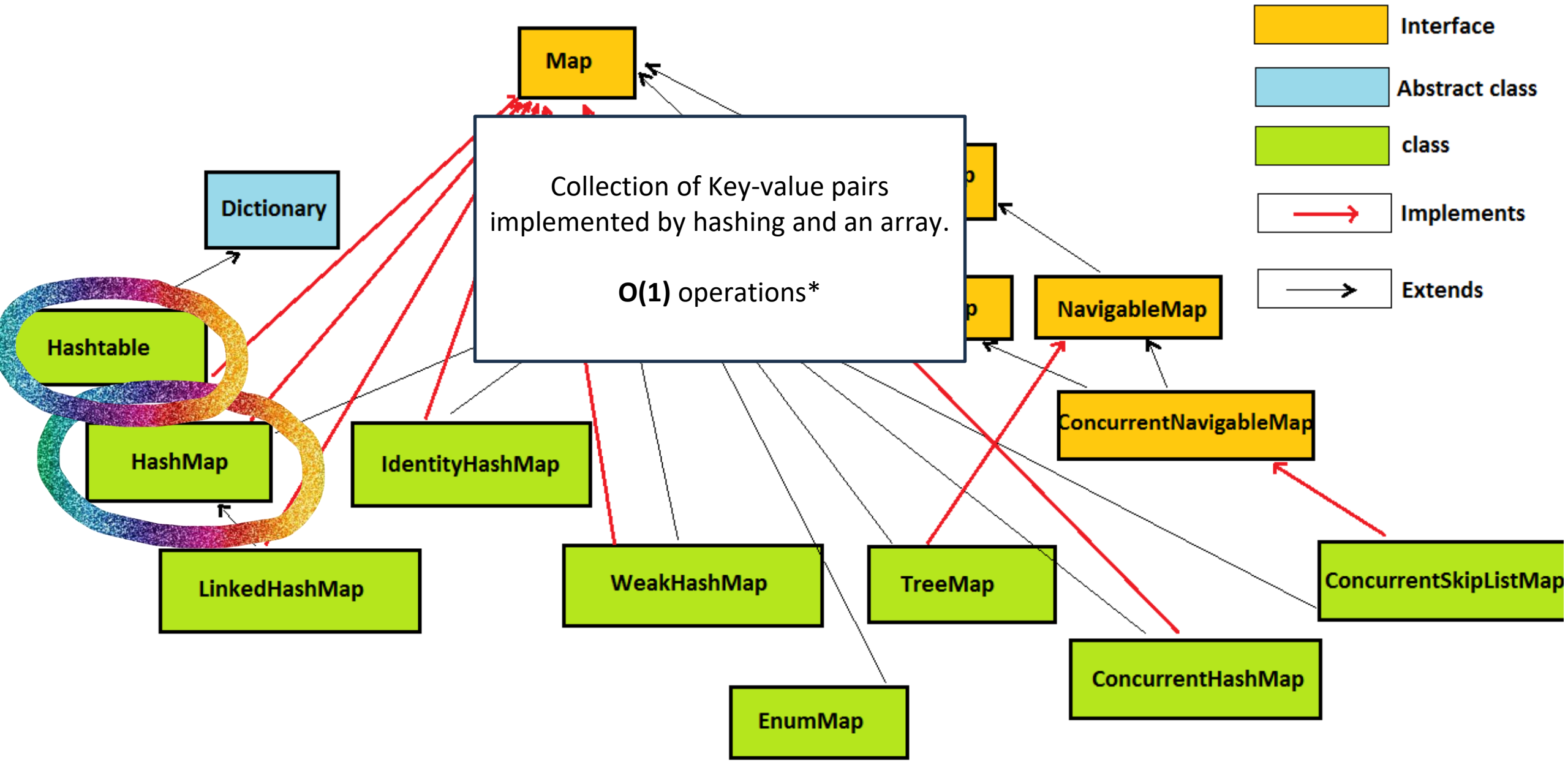

Map
Dictionary
SortedMap
Abstra
Map
Hashtable
HashMap
IdentityHashMap
NavigableMap
LinkedHashMap
WeakHashMap
TreeMap
ConcurrentSkipListMap
ConcurrentHashMap
EnumMap
Collection of Key-value pairs implemented by a Balanced BST.
O(logn) operations
Interface
Abstract class
class
Implements
Extends

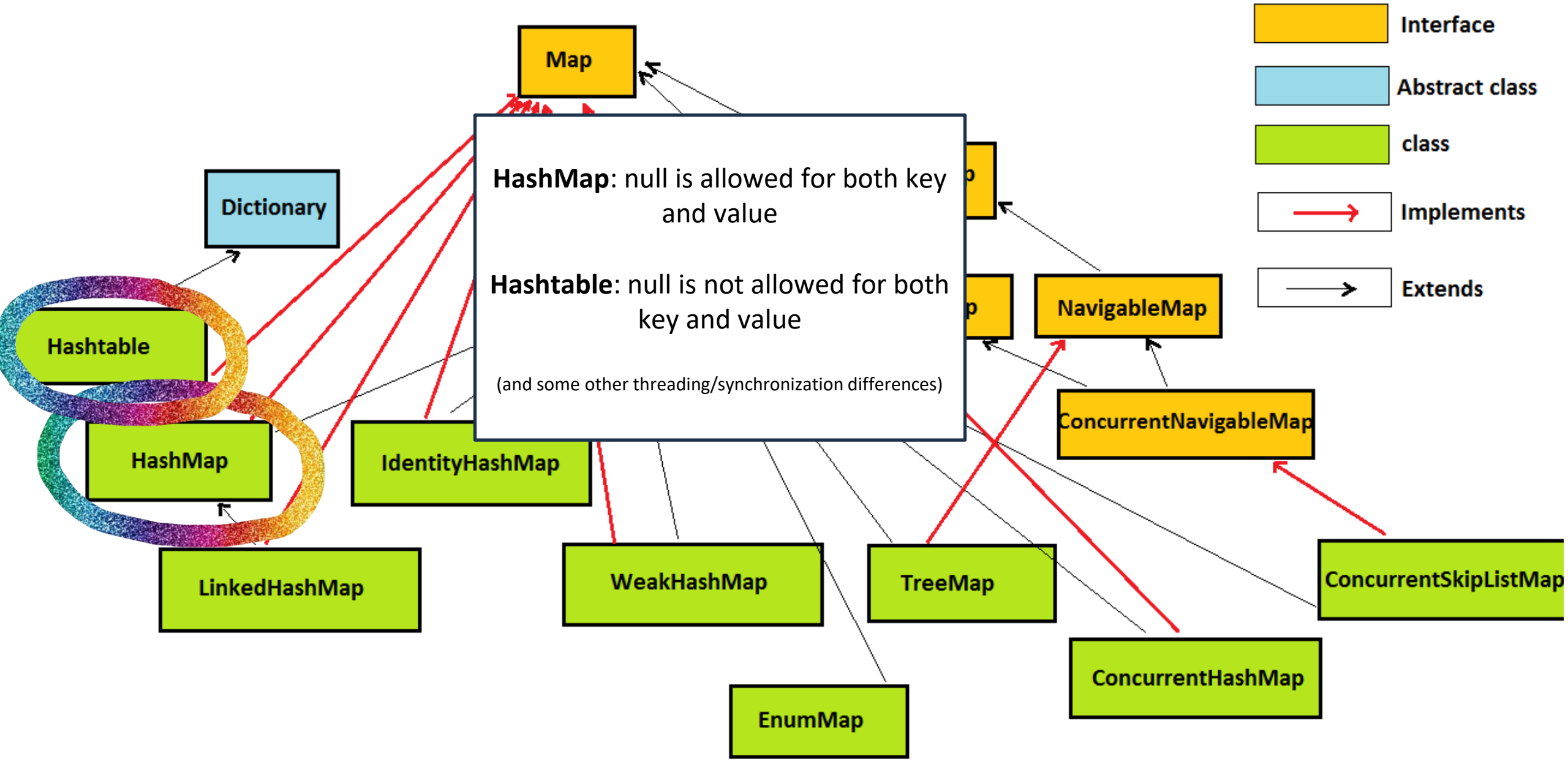
This class provides all of the optional Map operations, and permits null elements. Like HashMap, it provides constant-time performance for the basic operations (add, contains and remove), assuming the hash function disperses elements properly among the buckets. Performance is likely to be just slightly below that of HashMap, due to the added expense of maintaining the linked list, with one exception: Iteration over the collection-views of a LinkedHashMap requires time proportional to the *size* of the map, regardless of its capacity. Iteration over a HashMap is likely to be more expensive, requiring time proportional to its *capacity*.



Collection of Key-value pairs implemented by a linked list.

O(1) operations*


Map
Dictionary
Hashtable
HashMap
LinkedHashMap
IdentityHashMap
WeakHashMap
EnumMap
TreeMap
NavigableMap
ConcurrentNavigableMap
ConcurrentHashMap
ConcurrentSkipListMap
Collection of Key-value pairs implemented by hashing and an array.
O(1) operations*
Interface
Abstract class
class
Implements
Extends

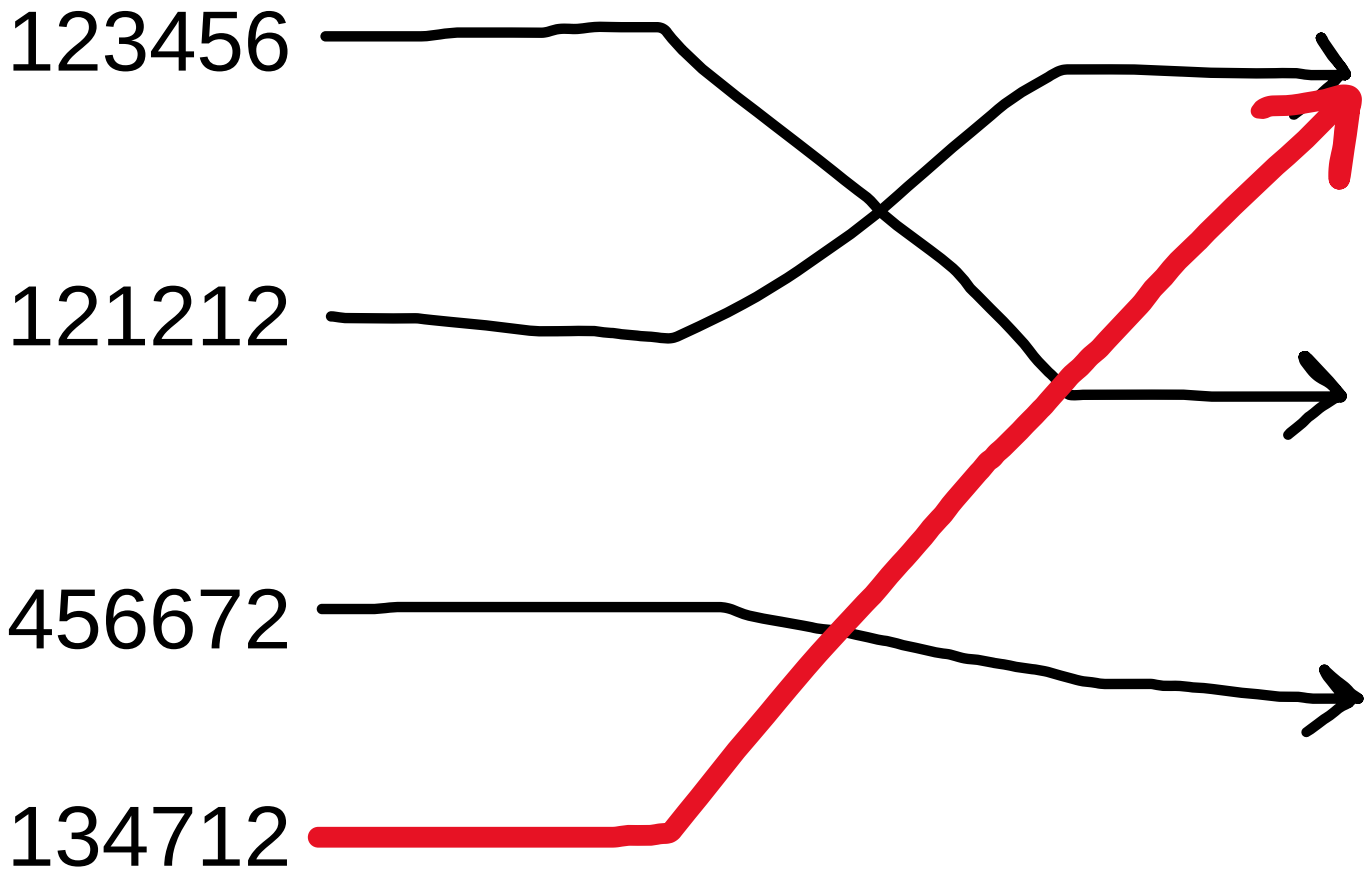

Map
Dictionary
Hashtable
HashMap
IdentityHashMap
LinkedHashMap
WeakHashMap
EnumMap
TreeMap
NavigableMap
ConcurrentNavigableMap
ConcurrentHashMap
ConcurrentSkipListMap
HashMap: null is allowed for both key and value
Hashtable: null is not allowed for both key and value
(and some other threading/synchronization differences)
Interface
Abstract class
class
Implements
Extends

Hash Tables

Student ID

Hash Function

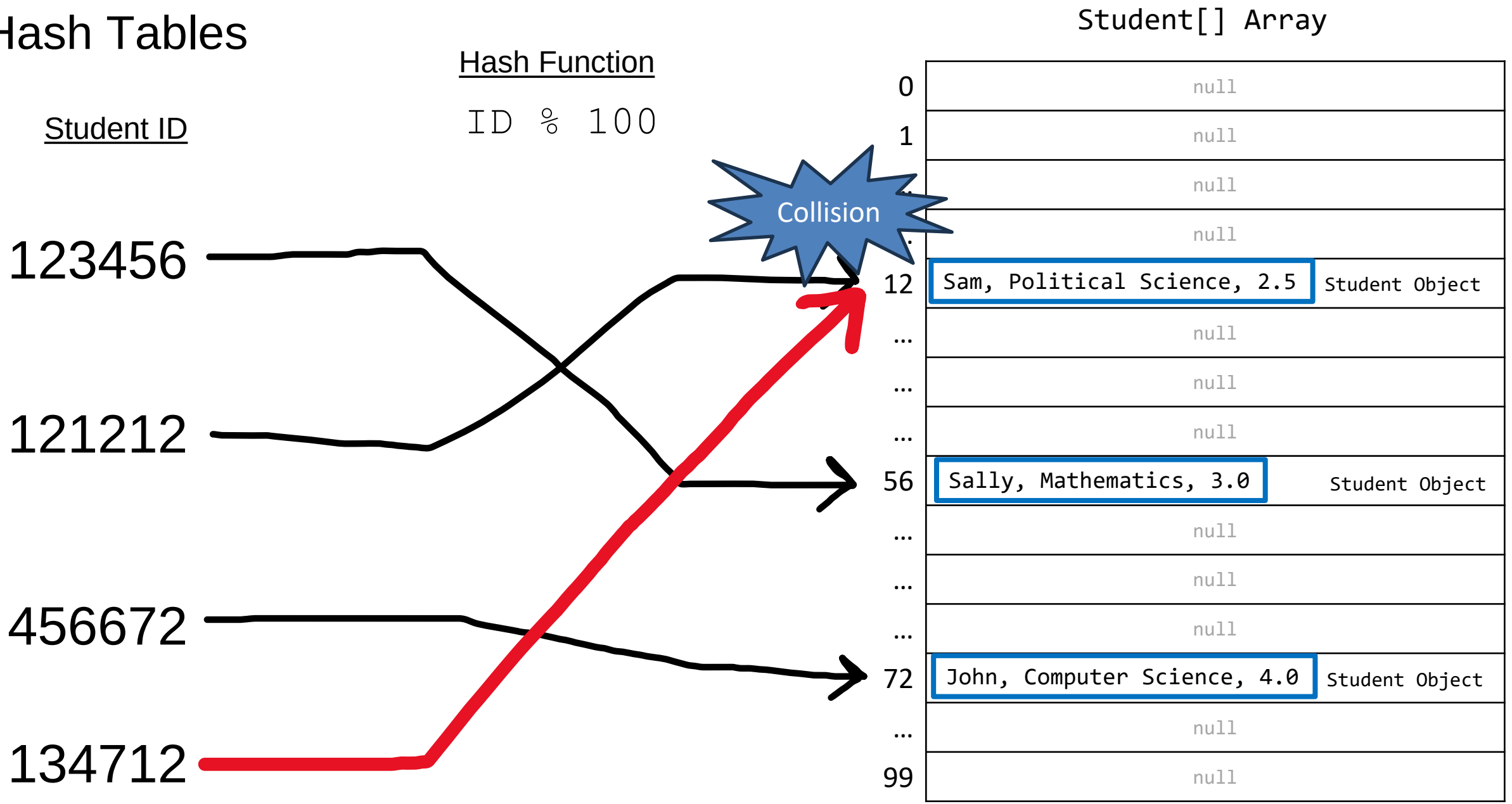
`ID % 100`



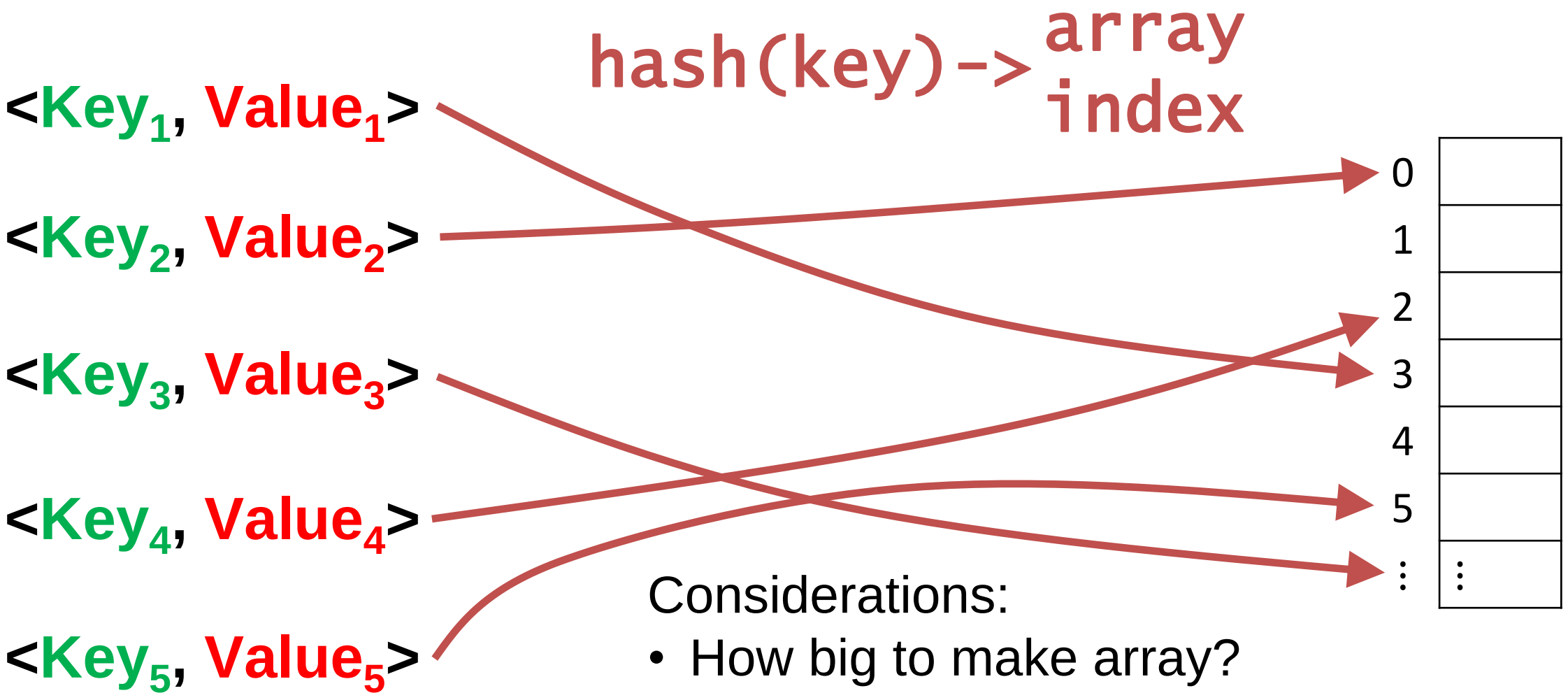
Student[] Array

0	null
1	null
...	null
...	null
12	Sam, Political Science, 2.5 — Student Object
...	null
...	null
...	null
56	Sally, Mathematics, 3.0 — Student Object
...	null
...	null
...	null
72	John, Computer Science, 4.0 — Student Object
...	null
99	null

Hash Tables


Hash Function
ID % 100
Student ID
123456
121212
456672
134712
Collision
Student[] Array
0
1
12
56
72
99
null
Sam, Political Science, 2.5
Student Object
Sally, Mathematics, 3.0
Student Object
John, Computer Science, 4.0
Student Object

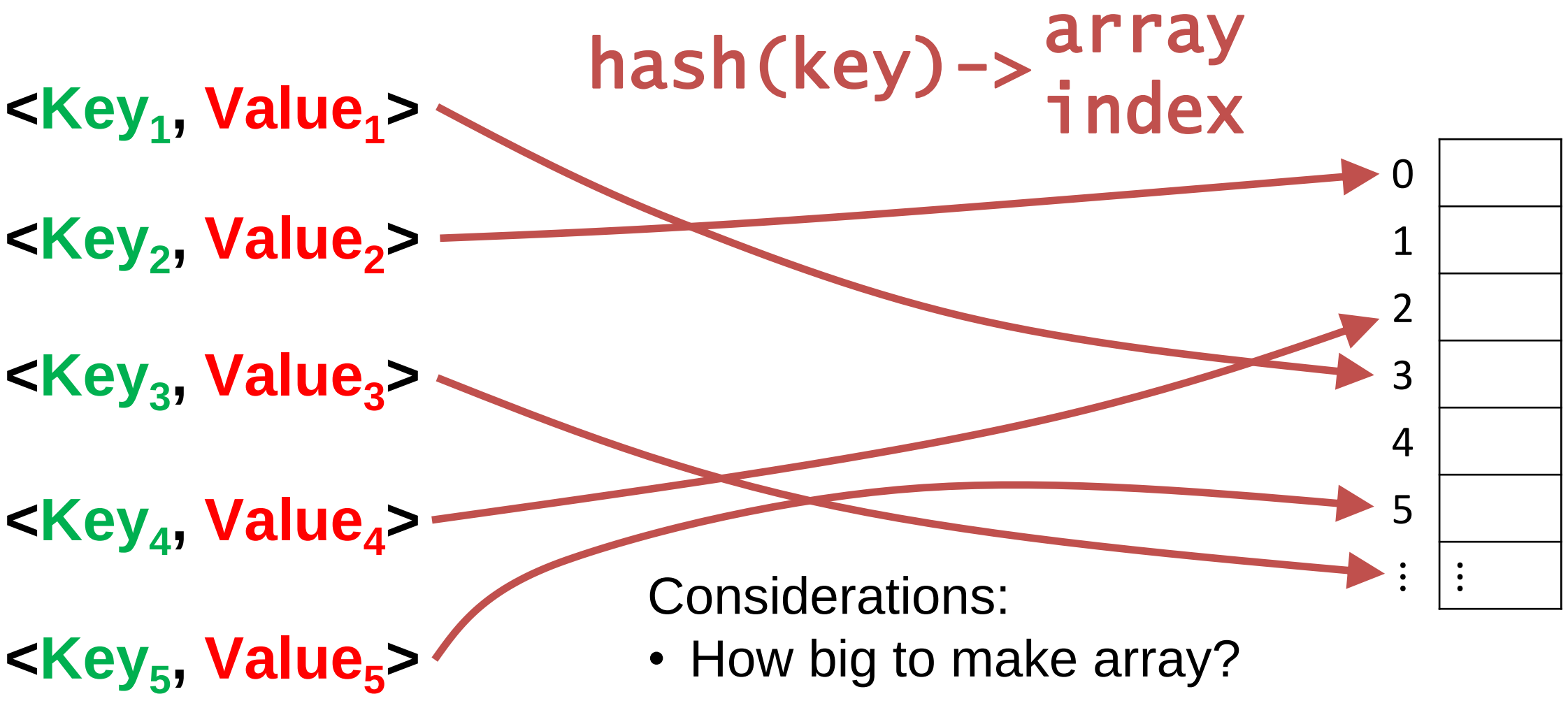
Hash Tables



Considerations:

- How big to make array?
- How to avoid collisions?
- How to handle collisions?

Hash Tables



Considerations:

- How big to make array?
- **How to avoid collisions?**
- How to handle collisions?

Hash Functions

Fact: Every hash function is susceptible to collisions

Hash Function: Function that translates key values to array indices

Hash Functions

Fact: Every hash function is susceptible to collisions

However, if we write a **good** hash function, we can reduce the number of collisions in our hash table

Hash Functions

Hash Functions

Keys	Hash Value
516-07-0854	
516-66-6218	
531-01-7352	
336-82-2121	
517-90-7152	
516-98-8002	
517-45-0907	
531-81-7489	
517-07-7312	
516-24-6185	
669-44-6499	
516-16-4236	
530-92-1795	
611-52-4556	
516-34-3352	
607-86-0812	

Hash Functions

Hash Function

Lessons:

Keys	Hash Value
516-07-0854	
516-66-6218	
531-01-7352	
336-82-2121	
517-90-7152	
516-98-8002	
517-45-0907	
531-81-7489	
517-07-7312	
516-24-6185	
669-44-6499	
516-16-4236	
530-92-1795	
611-52-4556	
516-34-3352	
607-86-0812	

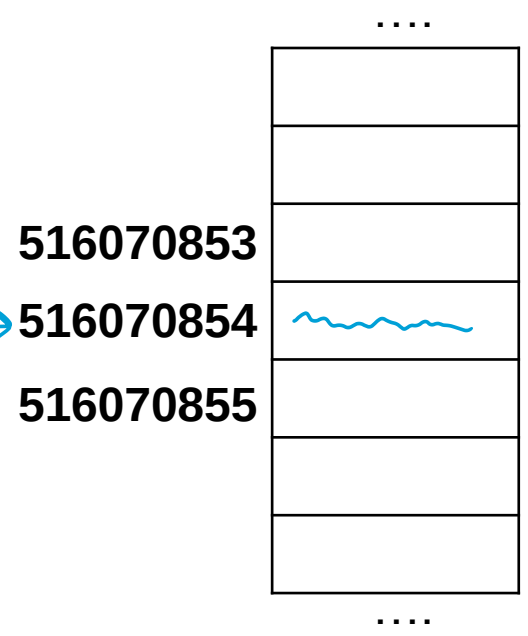
Hash Functions

Hash Function

$F(x) = x$

Lessons:

Keys	Hash Value
516-07-0854	516070853
516-66-6218	516070854
531-01-7352	516070855
336-82-2121	
517-90-7152	
516-98-8002	
517-45-0907	
531-81-7489	
517-07-7312	
516-24-6185	
669-44-6499	
516-16-4236	
530-92-1795	
611-52-4556	
516-34-3352	
607-86-0812	



Hash Functions

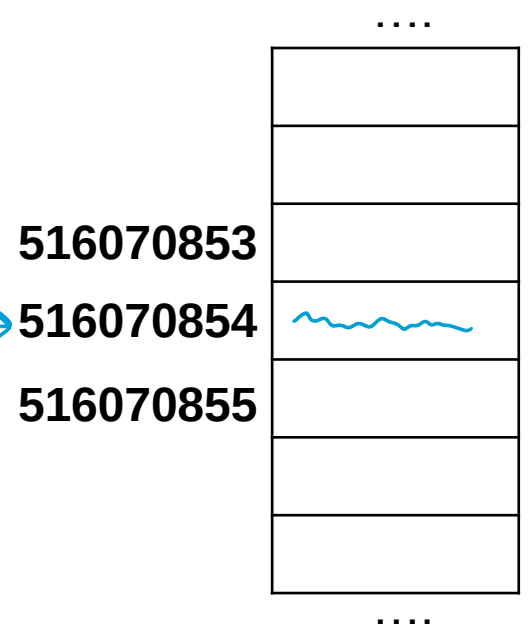
Hash Function

$$F(x) = x$$

Not good…
Very big array
Lots of wasted space

Lessons:

Keys	Hash Value
516-07-0854	516070854
516-66-6218	
531-01-7352	
336-82-2121	
517-90-7152	
516-98-8002	
517-45-0907	
531-81-7489	
517-07-7312	
516-24-6185	
669-44-6499	
516-16-4236	
530-92-1795	
611-52-4556	
516-34-3352	
607-86-0812	



Hash Functions

Hash Function

(First three digits) % 100

Size of Array

(**Modular hashing**)

Lessons:

Keys	Hash Value
516-07-0854	
516-66-6218	
531-01-7352	
336-82-2121	
517-90-7152	
516-98-8002	
517-45-0907	
531-81-7489	
517-07-7312	
516-24-6185	
669-44-6499	
516-16-4236	
530-92-1795	
611-52-4556	
516-34-3352	
607-86-0812	

Hash Functions

Hash Function

$$(\text{First three digits}) \% 100$$

Lessons:

Keys	Hash Value
516-07-0854	16
516-66-6218	16
531-01-7352	31
336-82-2121	36
517-90-7152	17
516-98-8002	16
517-45-0907	17
531-81-7489	31
517-07-7312	17
516-24-6185	16
669-44-6499	69
516-16-4236	16
530-92-1795	30
611-52-4556	11
516-34-3352	16
607-86-0812	07

Hash Functions

Hash Function

$$(\text{First three digits}) \% 100$$

Lessons:

What is the problem?

Keys	Hash Value
516-07-0854	16
516-66-6218	16
531-01-7352	31
336-82-2121	36
517-90-7152	17
516-98-8002	16
517-45-0907	17
531-81-7489	31
517-07-7312	17
516-24-6185	16
669-44-6499	69
516-16-4236	16
530-92-1795	30
611-52-4556	11
516-34-3352	16
607-86-0812	07

Hash Functions

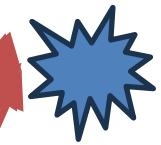
Hash Function

$$(\text{First three digits}) \% 100$$

Lessons:

What is the problem?

Keys	Hash Value
516-07-0854	16
516-66-6218	16
531-01-7352	31
336-82-2121	36
517-90-7152	17
516-98-8002	16
517-45-0907	17
531-81-7489	31
517-07-7312	17
516-24-6185	16
669-44-6499	69
516-16-4236	16
530-92-1795	30
611-52-4556	11
516-34-3352	16
607-86-0812	07



Hash Functions

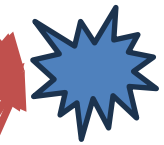
Hash Function

$$(\text{First three digits}) \% 100$$

Lessons:

What is the problem?
Why is the problem occurring?

Keys	Hash Value
516-07-0854	16
516-66-6218	16
531-01-7352	31
336-82-2121	36
517-90-7152	17
516-98-8002	16
517-45-0907	17
531-81-7489	31
517-07-7312	17
516-24-6185	16
669-44-6499	69
516-16-4236	16
530-92-1795	30
611-52-4556	11
516-34-3352	16
607-86-0812	07



Hash Functions

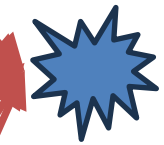
Hash Function

$$(\text{First three digits}) \% 100$$

Lessons:

What is the problem?
Why is the problem occurring?
How can we fix it?

Keys	Hash Value
516-07-0854	16
516-66-6218	16
531-01-7352	31
336-82-2121	36
517-90-7152	17
516-98-8002	16
517-45-0907	17
531-81-7489	31
517-07-7312	17
516-24-6185	16
669-44-6499	69
516-16-4236	16
530-92-1795	30
611-52-4556	11
516-34-3352	16
607-86-0812	07



Hash Functions

Hash Function

$$(\text{First three digits}) \% 100$$

Lessons:

1. Use as much of the key as possible

Keys	Hash Value
516-07-0854	
516-66-6218	
531-01-7352	
336-82-2121	
517-90-7152	
516-98-8002	
517-45-0907	
531-81-7489	
517-07-7312	
516-24-6185	
669-44-6499	
516-16-4236	
530-92-1795	
611-52-4556	
516-34-3352	
607-86-0812	

Hash Functions

Hash Function

$$(\text{Nine-Digit Number}) \% 5$$

Lessons:

1. Use as much of the key as possible

Keys	Hash Value
516-07-0854	
516-66-6218	
531-01-7352	
336-82-2121	
517-90-7152	
516-98-8002	
517-45-0907	
531-81-7489	
517-07-7312	
516-24-6185	
669-44-6499	
516-16-4236	
530-92-1795	
611-52-4556	
516-34-3352	
607-86-0812	

Hash Functions

Hash Function

$$(\text{Nine-Digit Number}) \% 5$$

Lessons:

1. Use as much of the key as possible

Keys	Hash Value
516-07-0854	4
516-66-6218	3
531-01-7352	2
336-82-2121	1
517-90-7152	2
516-98-8002	2
517-45-0907	2
531-81-7489	4
517-07-7312	2
516-24-6185	0
669-44-6499	4
516-16-4236	1
530-92-1795	0
611-52-4556	1
516-34-3352	2
607-86-0812	2

Hash Functions

Hash Function

(Nine-Digit Number) % 5

Lessons:

1. Use as much of the key as possible

Keys	Hash Value
516-07-0854	4
516-66-6218	3
531-01-7352	2
336-82-2121	1
517-90-7152	2
516-98-8002	2
517-45-0907	2
531-81-7489	4
517-07-7312	2
516-24-6185	0
669-44-6499	4
516-16-4236	1
530-92-1795	0
611-52-4556	1
516-34-3352	2
607-86-0812	2

Hash Functions

Hash Function

$$(\text{Nine-Digit Number}) \% 5$$

Lessons:

1. Use as much of the key as possible
2. Make sure your array is big enough

Keys	Hash Value
516-07-0854	4
516-66-6218	3
531-01-7352	2
336-82-2121	1
517-90-7152	2
516-98-8002	2
517-45-0907	2
531-81-7489	4
517-07-7312	2
516-24-6185	0
669-44-6499	4
516-16-4236	1
530-92-1795	0
611-52-4556	1
516-34-3352	2
607-86-0812	2

Hash Functions

Hash Function

$$(\text{Nine-Digit Number}) \% 500000000000$$

Lessons:

1. Use as much of the key as possible
2. Make sure your array is big enough

Keys	Hash Value
516-07-0854	
516-66-6218	
531-01-7352	
336-82-2121	
517-90-7152	
516-98-8002	
517-45-0907	
531-81-7489	
517-07-7312	
516-24-6185	
669-44-6499	
516-16-4236	
530-92-1795	
611-52-4556	
516-34-3352	
607-86-0812	

Hash Functions

Hash Function

$$(\text{Nine-Digit Number}) \% 500000000000$$

Lessons:

1. Use as much of the key as possible
2. Make sure your array is big enough (but not too big)

Keys	Hash Value
516-07-0854	
516-66-6218	
531-01-7352	
336-82-2121	
517-90-7152	
516-98-8002	
517-45-0907	
531-81-7489	
517-07-7312	
516-24-6185	
669-44-6499	
516-16-4236	
530-92-1795	
611-52-4556	
516-34-3352	
607-86-0812	

Hash Functions

Hash Function

$$(\text{Nine-Digit Number}) \% 100$$

Lessons:

1. Use as much of the key as possible
2. Make sure your array is big enough (but not too big)

Keys	Hash Value
516-07-0854	
516-66-6218	
531-01-7352	
336-82-2121	
517-90-7152	
516-98-8002	
517-45-0907	
531-81-7489	
517-07-7312	
516-24-6185	
669-44-6499	
516-16-4236	
530-92-1795	
611-52-4556	
516-34-3352	
607-86-0812	

Hash Functions

Hash Function

$$(\text{Nine-Digit Number}) \% 100$$

Lessons:

1. Use as much of the key as possible
2. Make sure your array is big enough (but not too big)

Keys	Hash Value
516-07-0854	54
516-66-6218	18
531-01-7352	52
336-82-2121	21
517-90-7152	52
516-98-8002	02
517-45-0907	07
531-81-7489	89
517-07-7312	12
516-24-6185	85
669-44-6499	99
516-16-4236	36
530-92-1795	95
611-52-4556	56
516-34-3352	52
607-86-0812	12

Hash Functions

Hash Function

$$(\text{Nine-Digit Number}) \% 100$$

Lessons:

1. Use as much of the key as possible
2. Make sure your array is big enough (but not too big)

Keys	Hash Value
516-07-0854	54
516-66-6218	18
531-01-7352	52
336-82-2121	21
517-90-7152	52
516-98-8002	02
517-45-0907	07
531-81-7489	89
517-07-7312	12
516-24-6185	85
669-44-6499	99
516-16-4236	36
530-92-1795	95
611-52-4556	56
516-34-3352	52
607-86-0812	12

Hash Functions

Hash Function

$$(\text{Nine-Digit Number}) \% 100$$

Lessons:

1. Use as much of the key as possible
2. Make sure your array is big enough (but not too big)
3. Use a **prime number** array size

Keys	Hash Value
516-07-0854	54
516-66-6218	18
531-01-7352	52
336-82-2121	21
517-90-7152	52
516-98-8002	02
517-45-0907	07
531-81-7489	89
517-07-7312	12
516-24-6185	85
669-44-6499	99
516-16-4236	36
530-92-1795	95
611-52-4556	56
516-34-3352	52
607-86-0812	12

Hash Functions

Hash Function

$$(\text{Nine-Digit Number}) \ \% \ 97$$

Lessons:

1. Use as much of the key as possible
2. Make sure your array is big enough (but not too big)
3. Use a **prime number** array size

Keys	Hash Value
516-07-0854	
516-66-6218	
531-01-7352	
336-82-2121	
517-90-7152	
516-98-8002	
517-45-0907	
531-81-7489	
517-07-7312	
516-24-6185	
669-44-6499	
516-16-4236	
530-92-1795	
611-52-4556	
516-34-3352	
607-86-0812	

Hash Functions

Hash Function

$$(\text{Nine-Digit Number}) \ \% \ 97$$

Lessons:

1. Use as much of the key as possible
2. Make sure your array is big enough (but not too big)
3. Use a **prime number** array size

Keys	Hash Value
516-07-0854	8
516-66-6218	83
531-01-7352	67
336-82-2121	0
517-90-7152	96
516-98-8002	21
517-45-0907	42
531-81-7489	51
517-07-7312	91
516-24-6185	60
669-44-6499	29
516-16-4236	76
530-92-1795	55
611-52-4556	84
516-34-3352	33
607-86-0812	30

Hash Functions

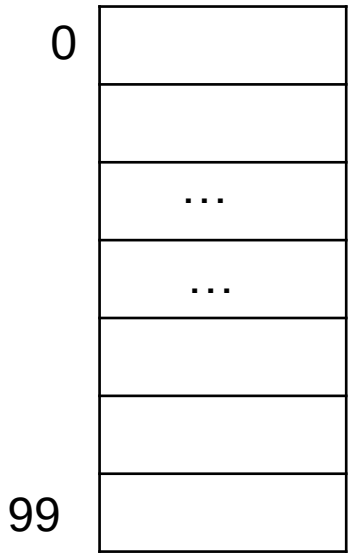
Hash Function

$$(\text{Nine-Digit Number}) \% 7753$$

Lessons:

1. Use as much of the key as possible
2. Make sure your array is big enough (but not too big)
3. Use a **prime number** array size

Keys	Hash Value
516-07-0854	
516-66-6218	
531-01-7352	
336-82-2121	
517-90-7152	
516-98-8002	
517-45-0907	
531-81-7489	
517-07-7312	
516-24-6185	
669-44-6499	
516-16-4236	
530-92-1795	
611-52-4556	
516-34-3352	
607-86-0812	



???

Hash Functions

Function that translates key values into array indices.

Requirements:

- **Well defined for all input.**
- **Identical keys map to identical indices.**
- **Maps to a specified range.**

Goals:

- **Easy to compute.**
- **Uniformly distributes keys across range (collision avoidance).**

Function that translates key values into array indices.

Requirements:

- **Well defined for all input.**
- **Identical keys map to identical indices.**
- **Maps to a specified range.**

Goals:

- **Easy to compute.**
- **Uniformly distributes keys across range (collision avoidance).**

Impossible:

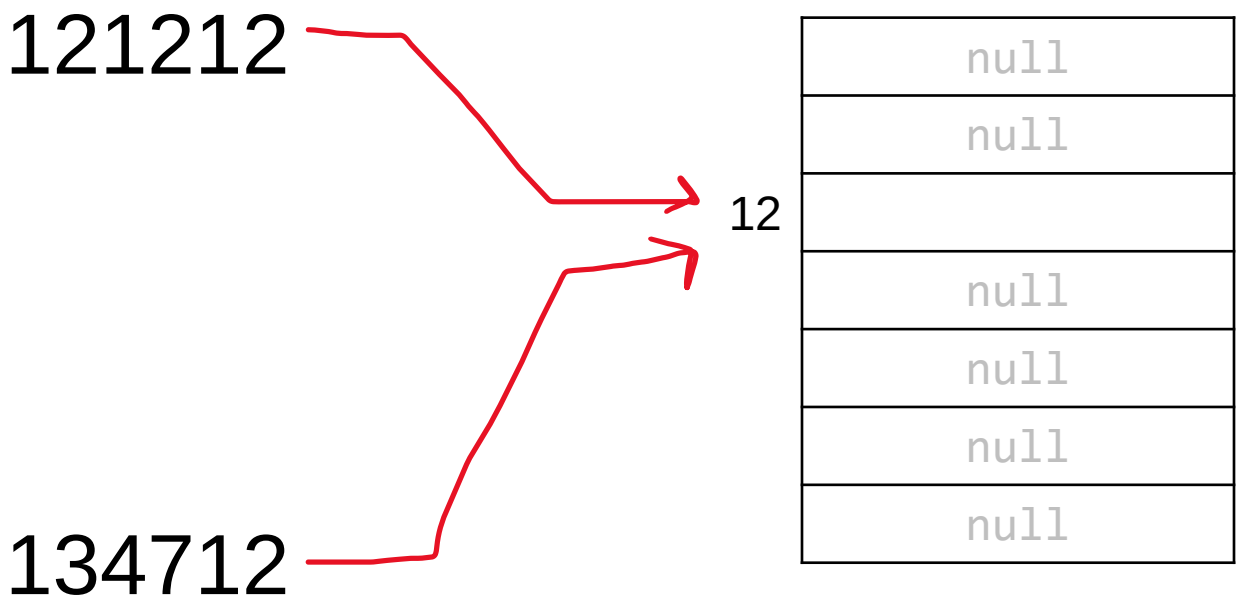
- 0% collision chance

Possible:

- 0.001% collision chance

Collision Resolution

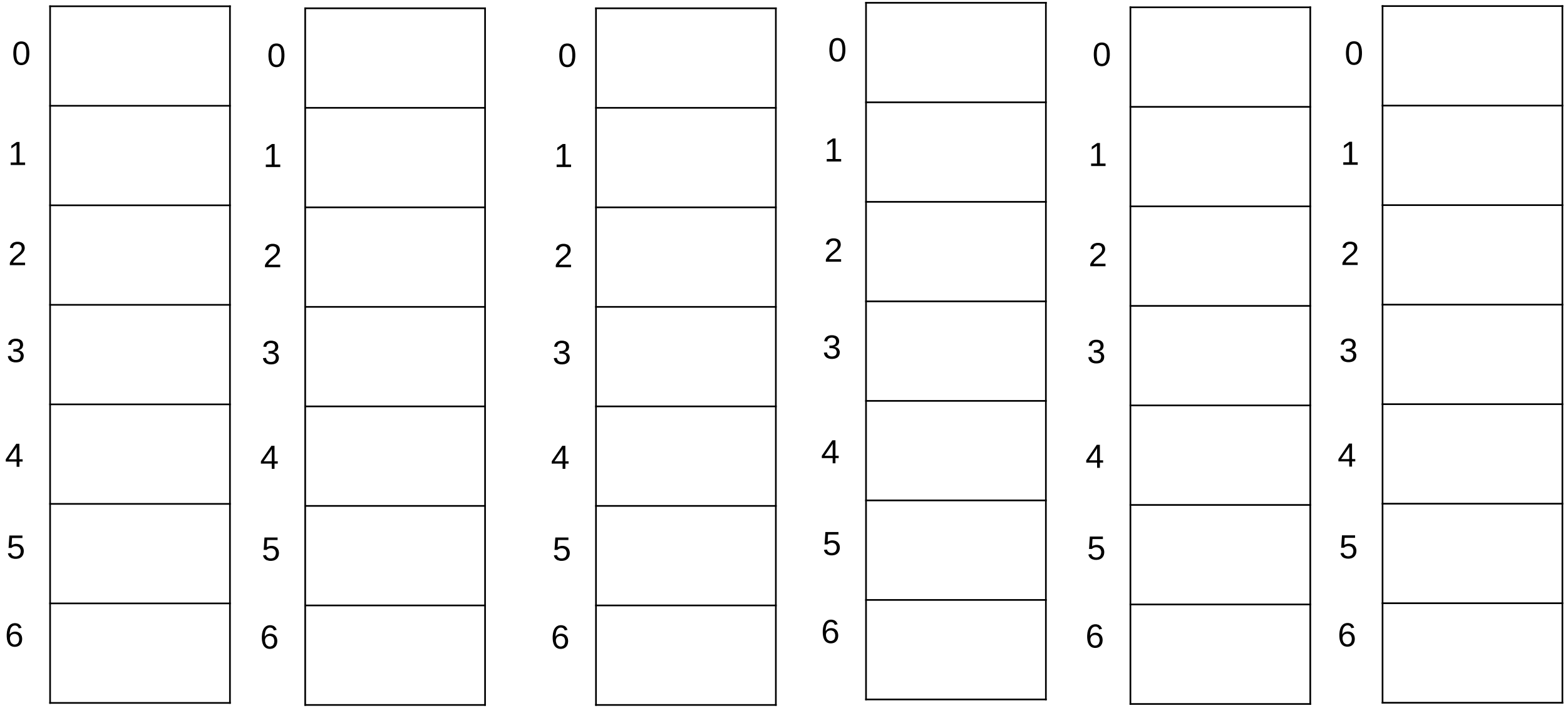
We know how to avoid collisions (strong hashing function), but collisions are bound to happen with every hash function



How to deal with collisions?

Collision Resolution

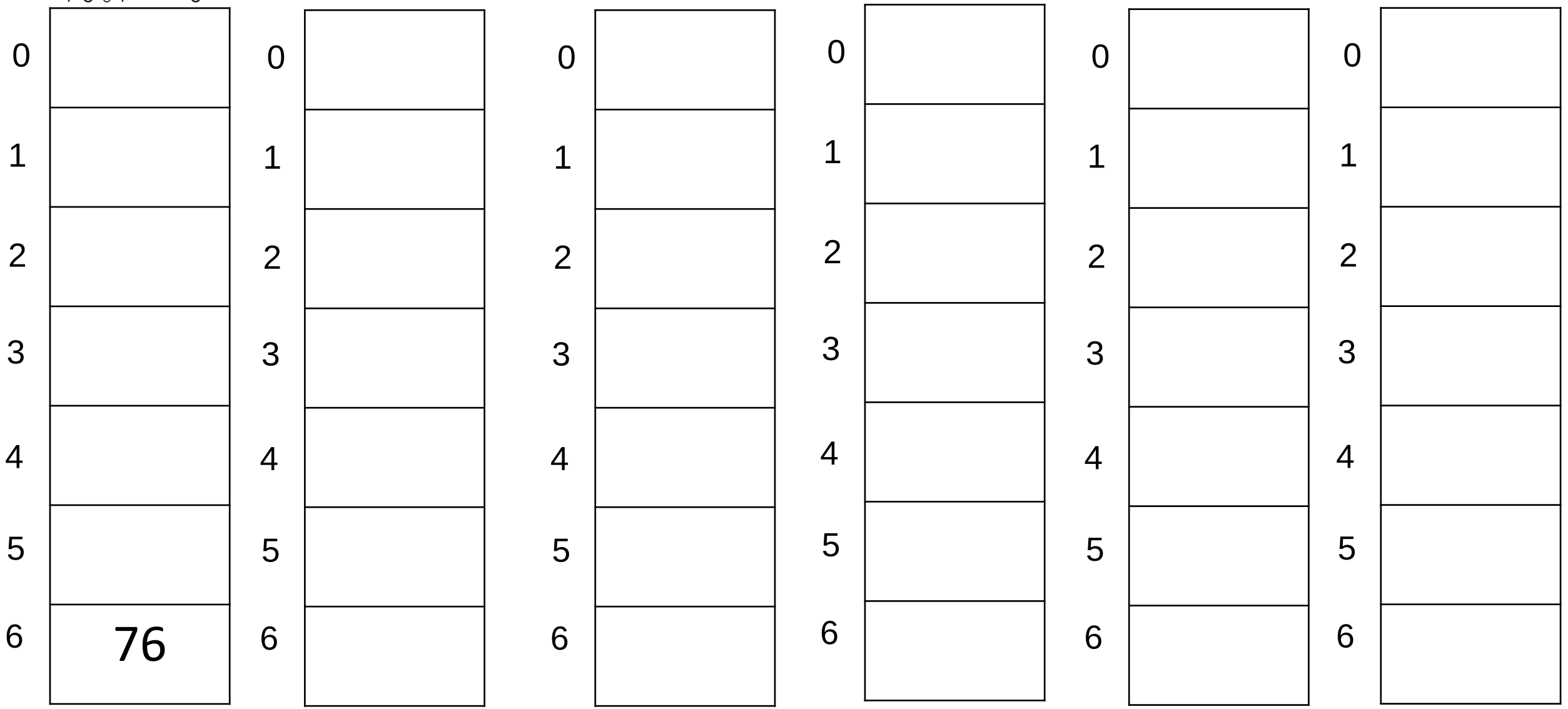
Linear Probing – Place value at next, sequential open place



Collision Resolution

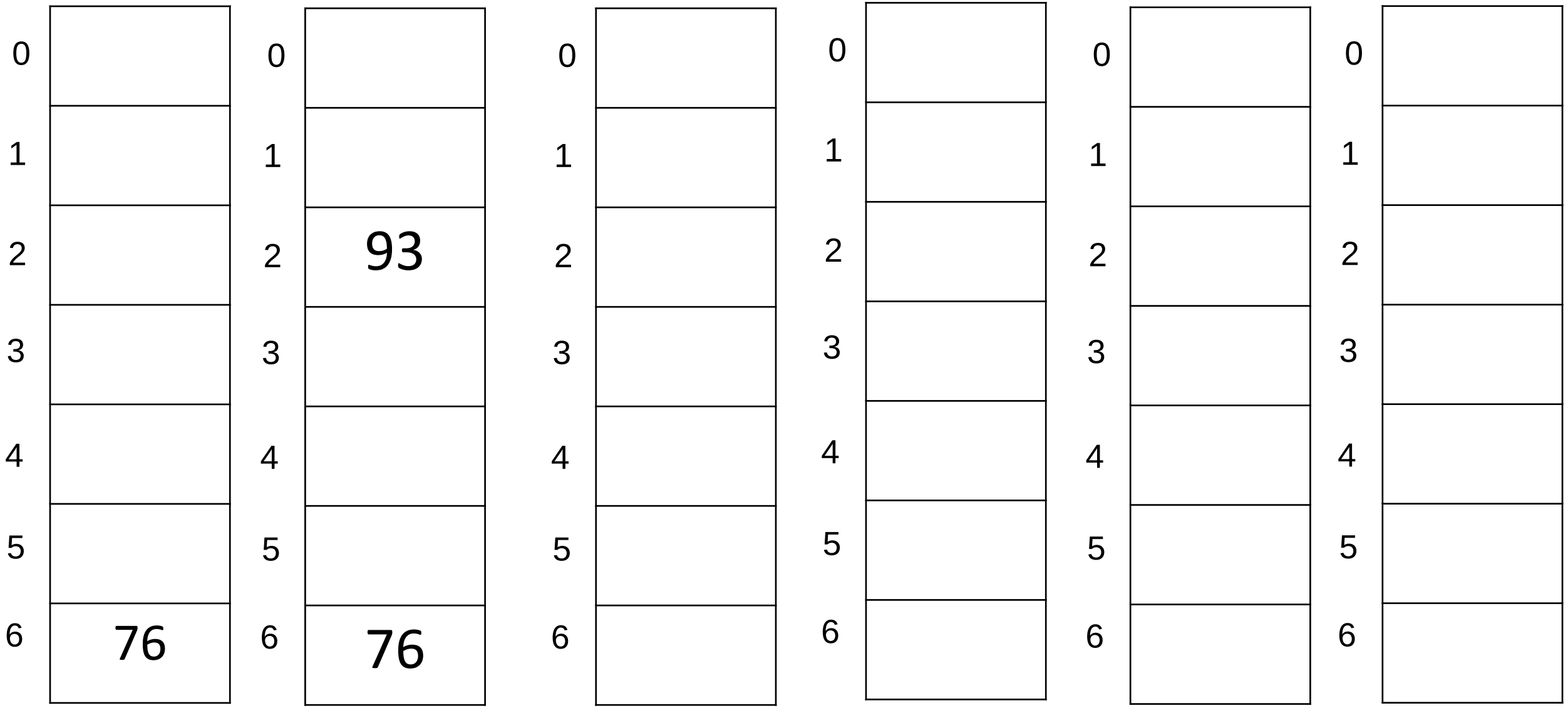
Linear Probing – Place value at next, sequential open place

```
insert(76)
76%7 = 6
```



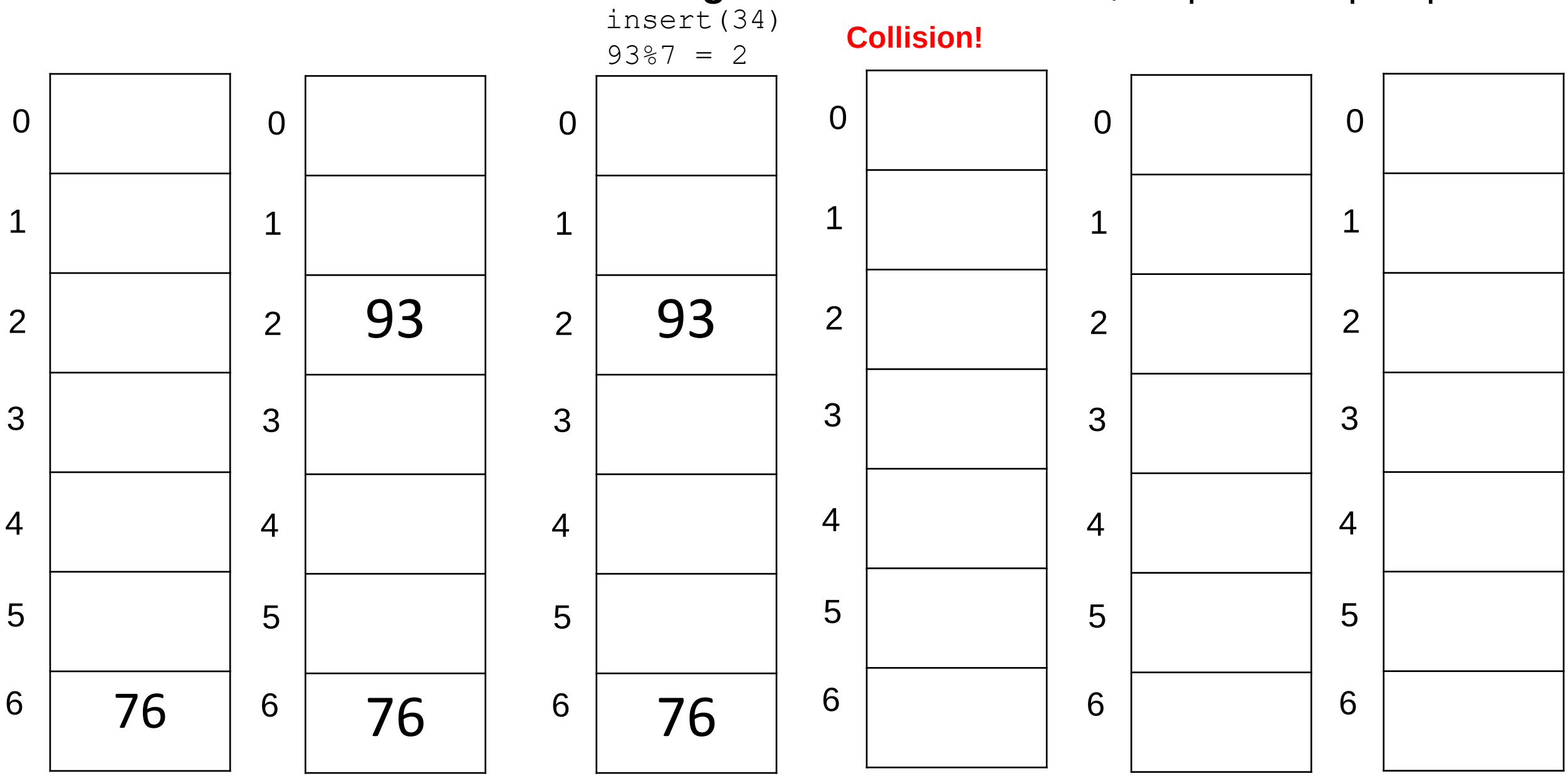
Collision Resolution **Linear Probing** – Place value at next, sequential open place

```
insert(93)
93%7 = 2
```



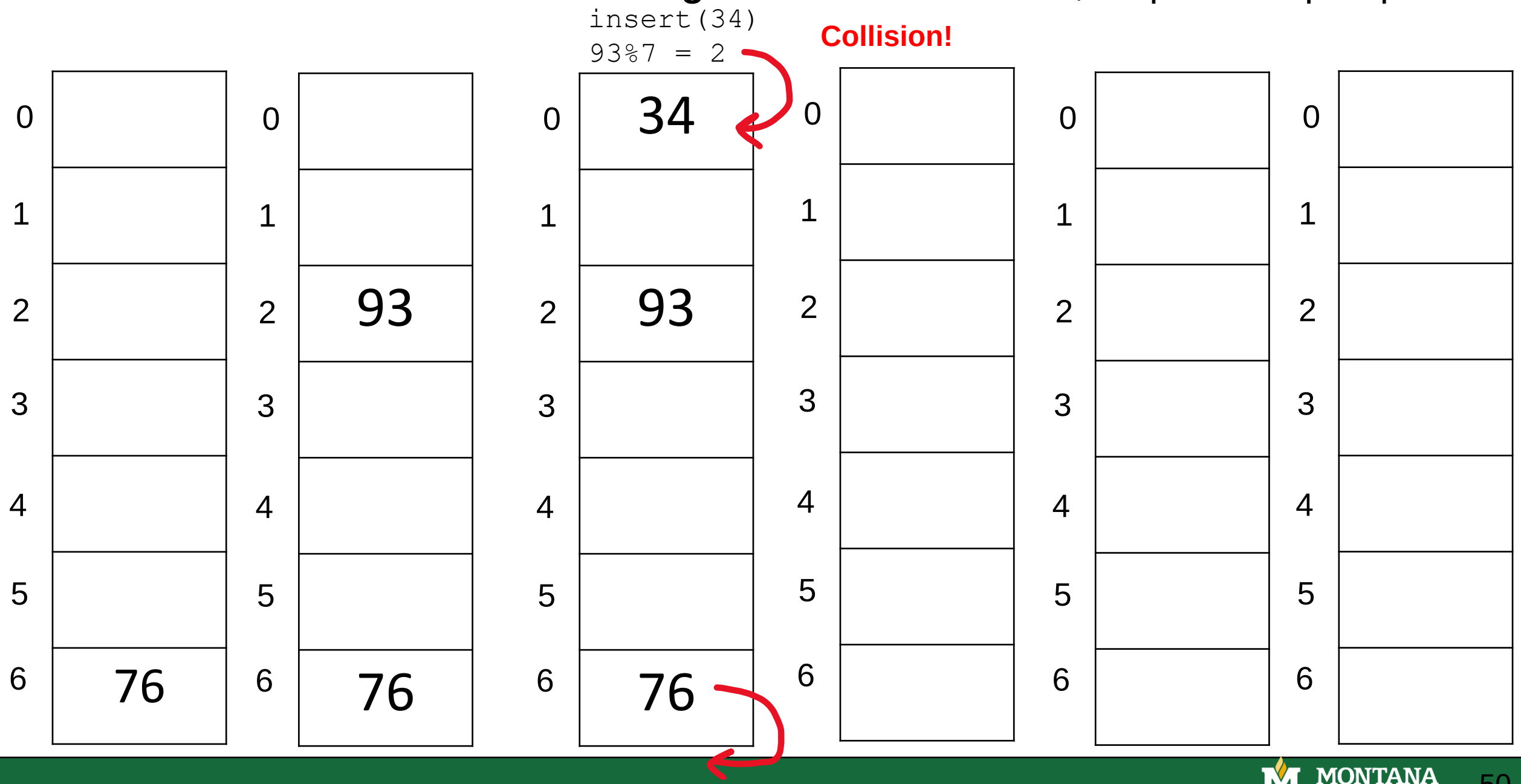
Collision Resolution

Linear Probing – Place value at next, sequential open place



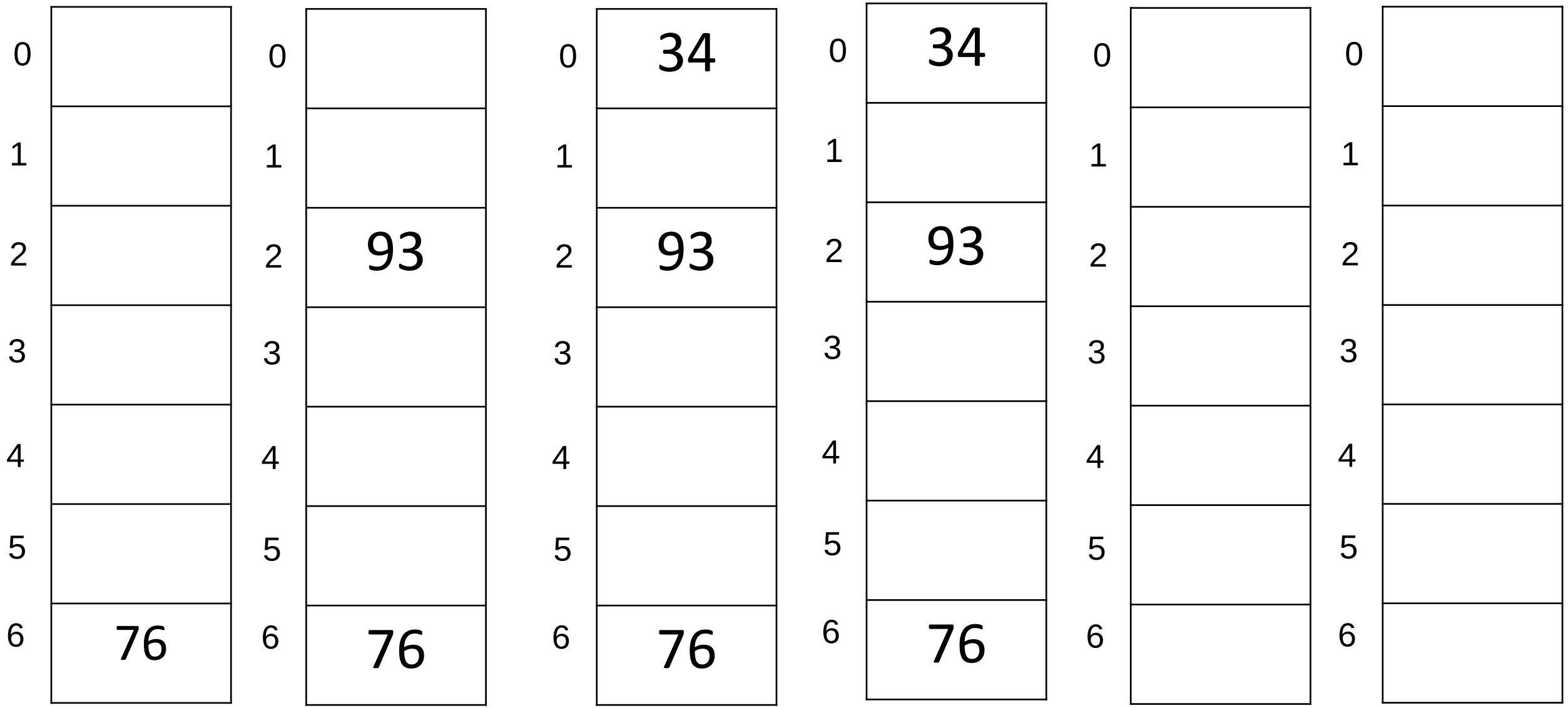
Collision Resolution

Linear Probing – Place value at next, sequential open place



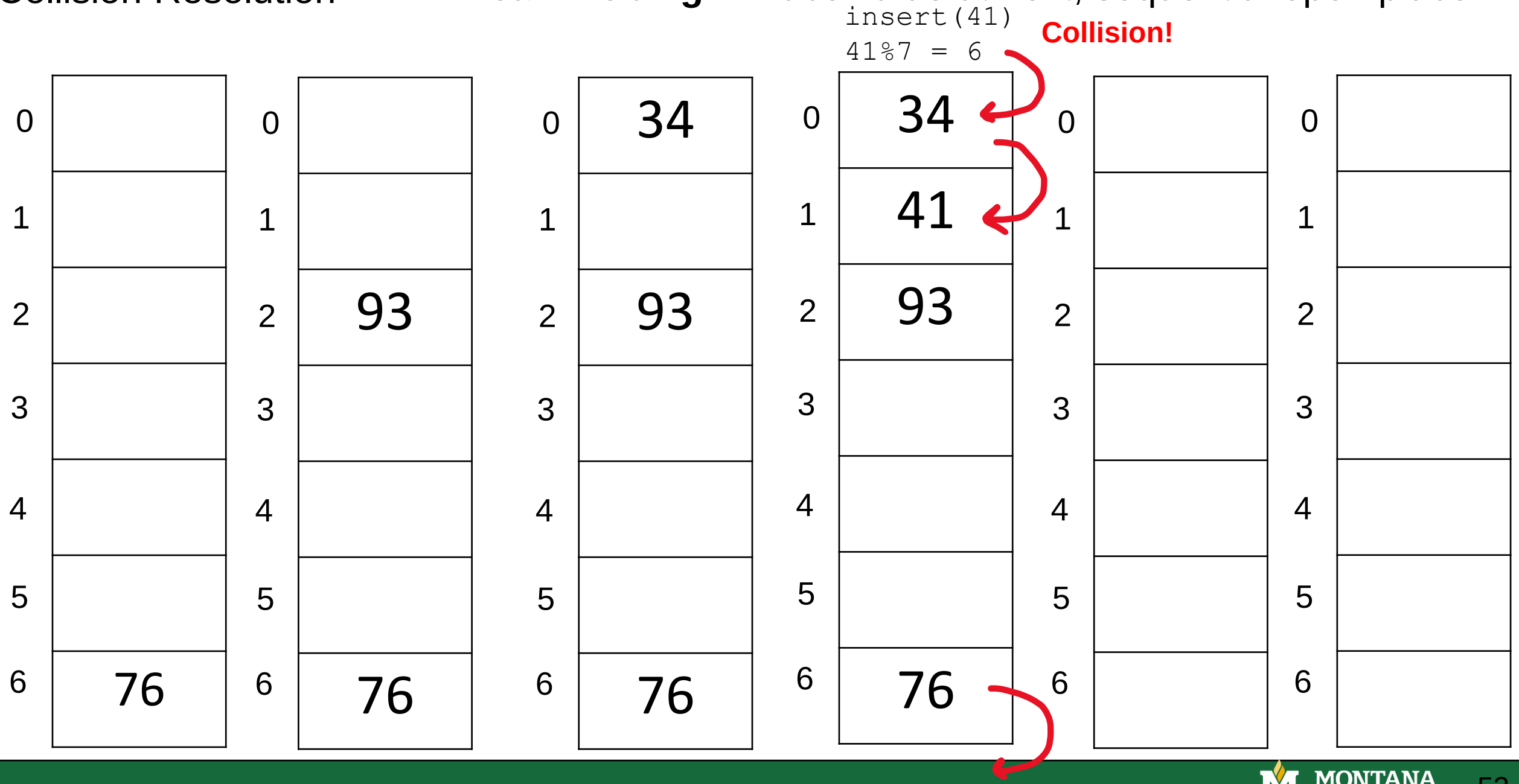
Collision Resolution

Linear Probing – Place value at next, sequential open place



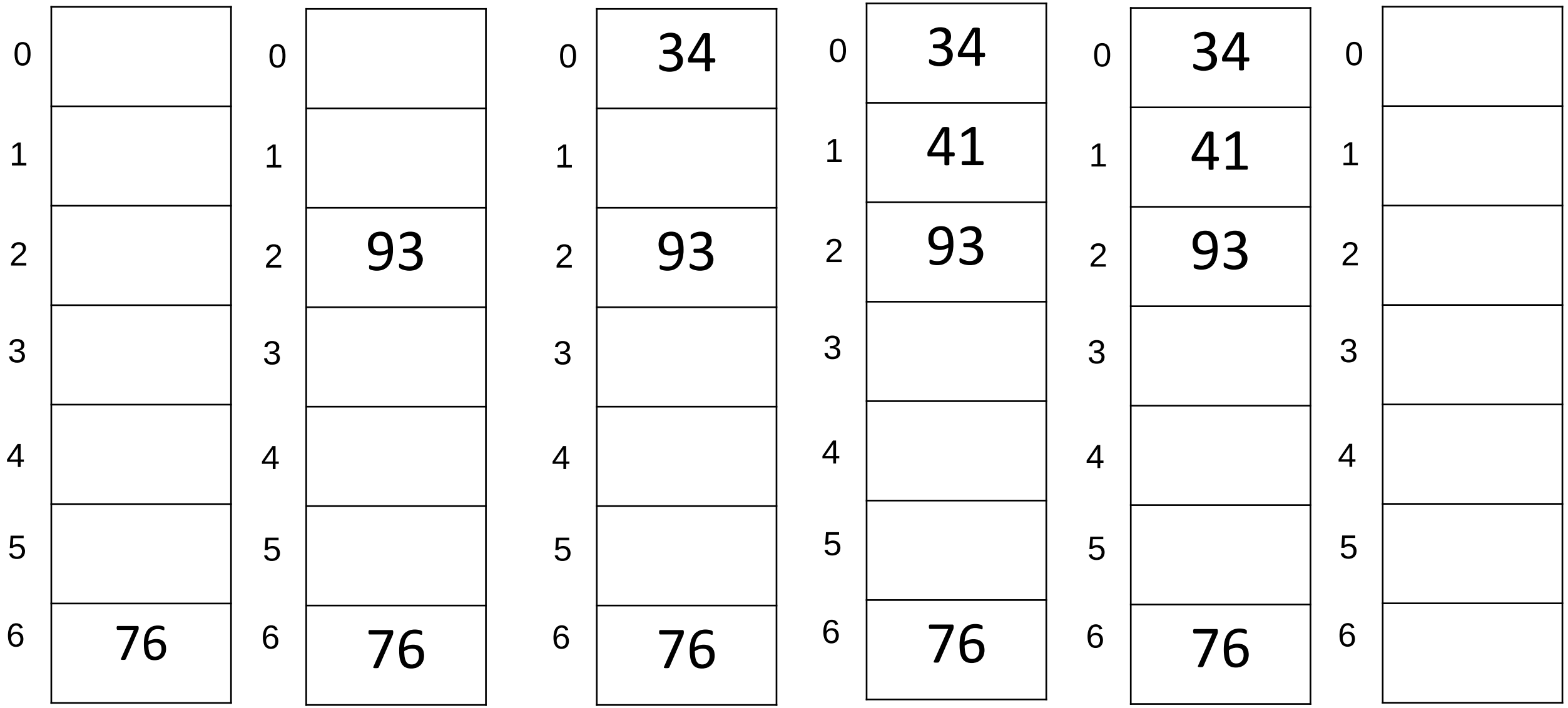
Collision Resolution

Linear Probing – Place value at next, sequential open place



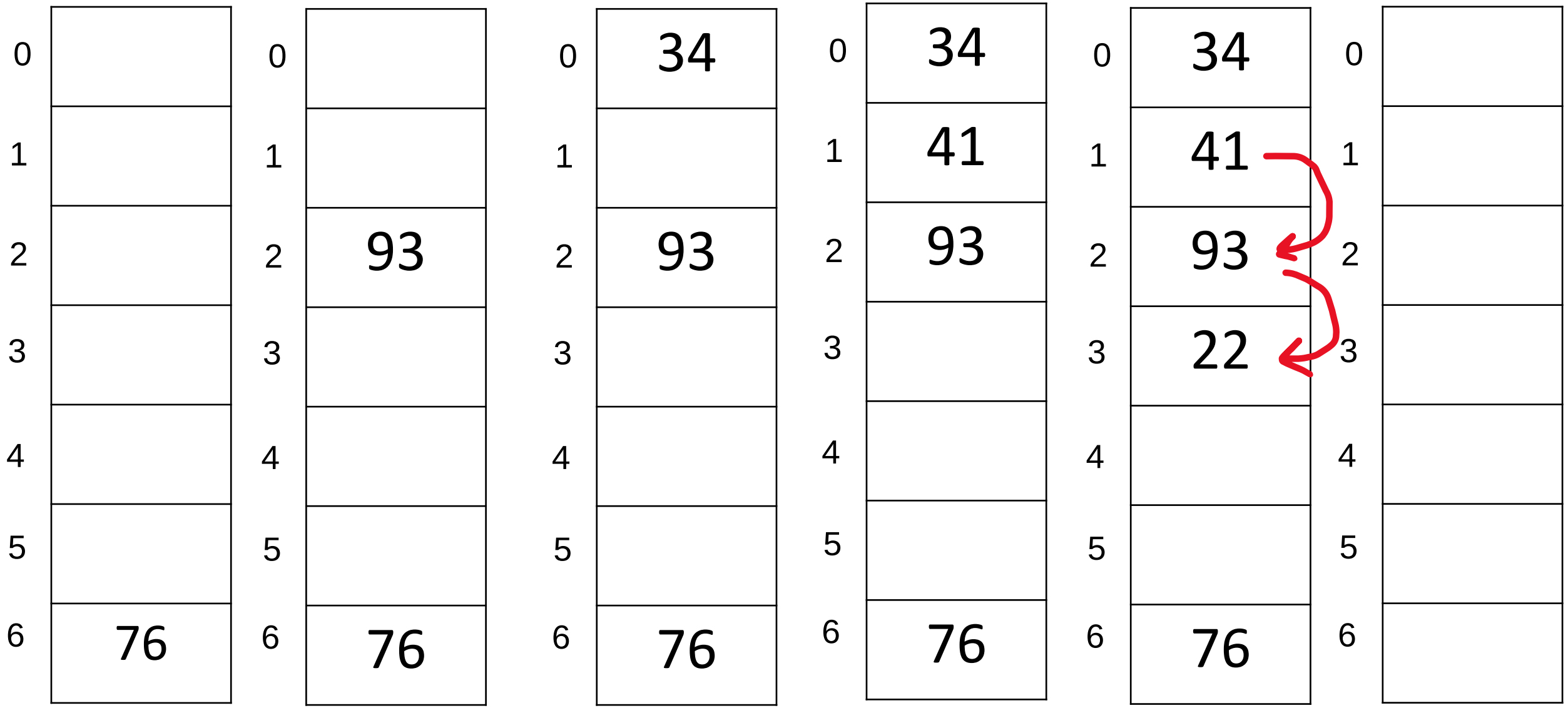
Collision Resolution

Linear Probing – Place value at next, sequential open place



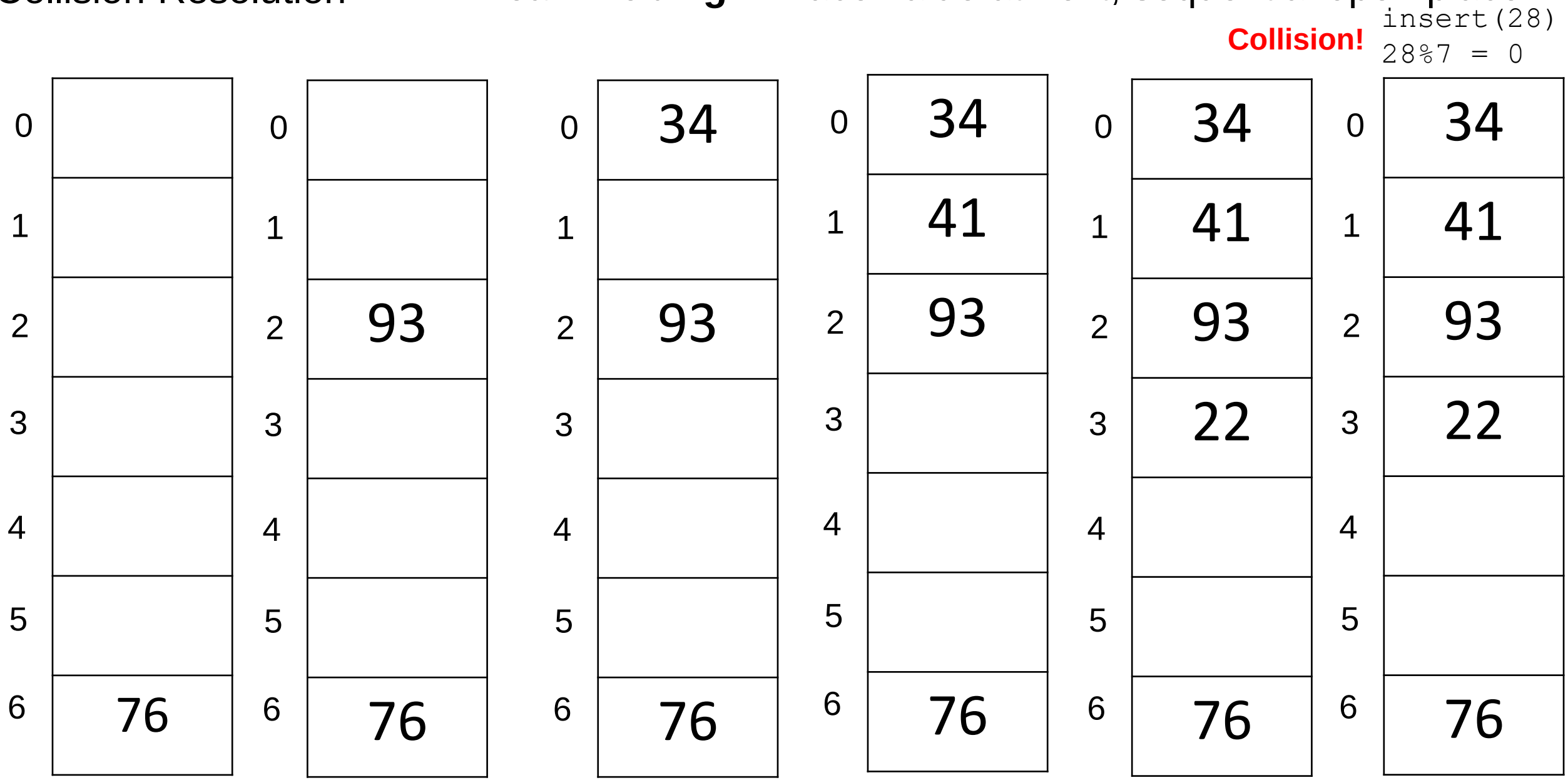
Collision Resolution

Linear Probing – Place value at next, sequential open place



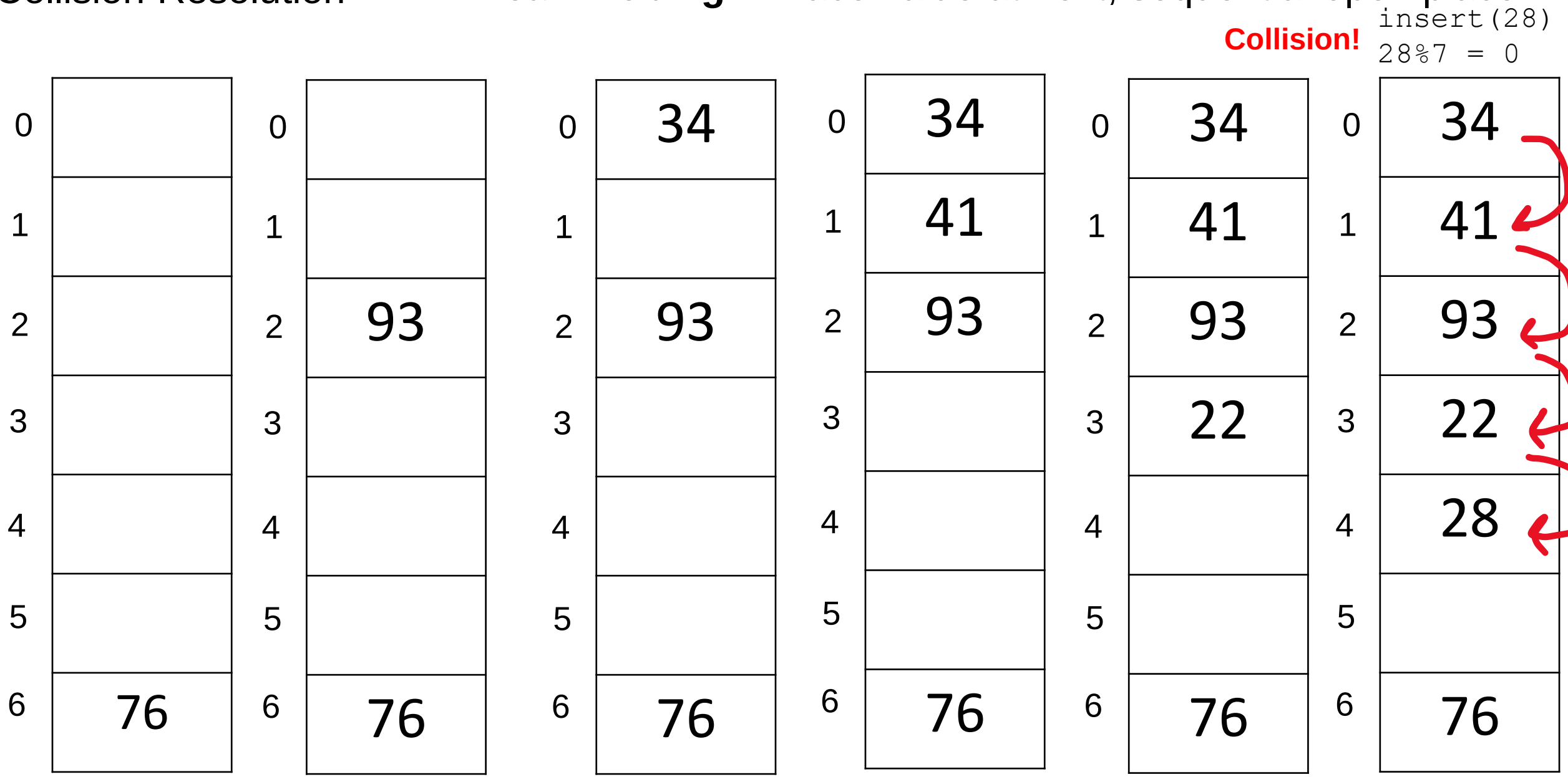
Collision Resolution

Linear Probing – Place value at next, sequential open place



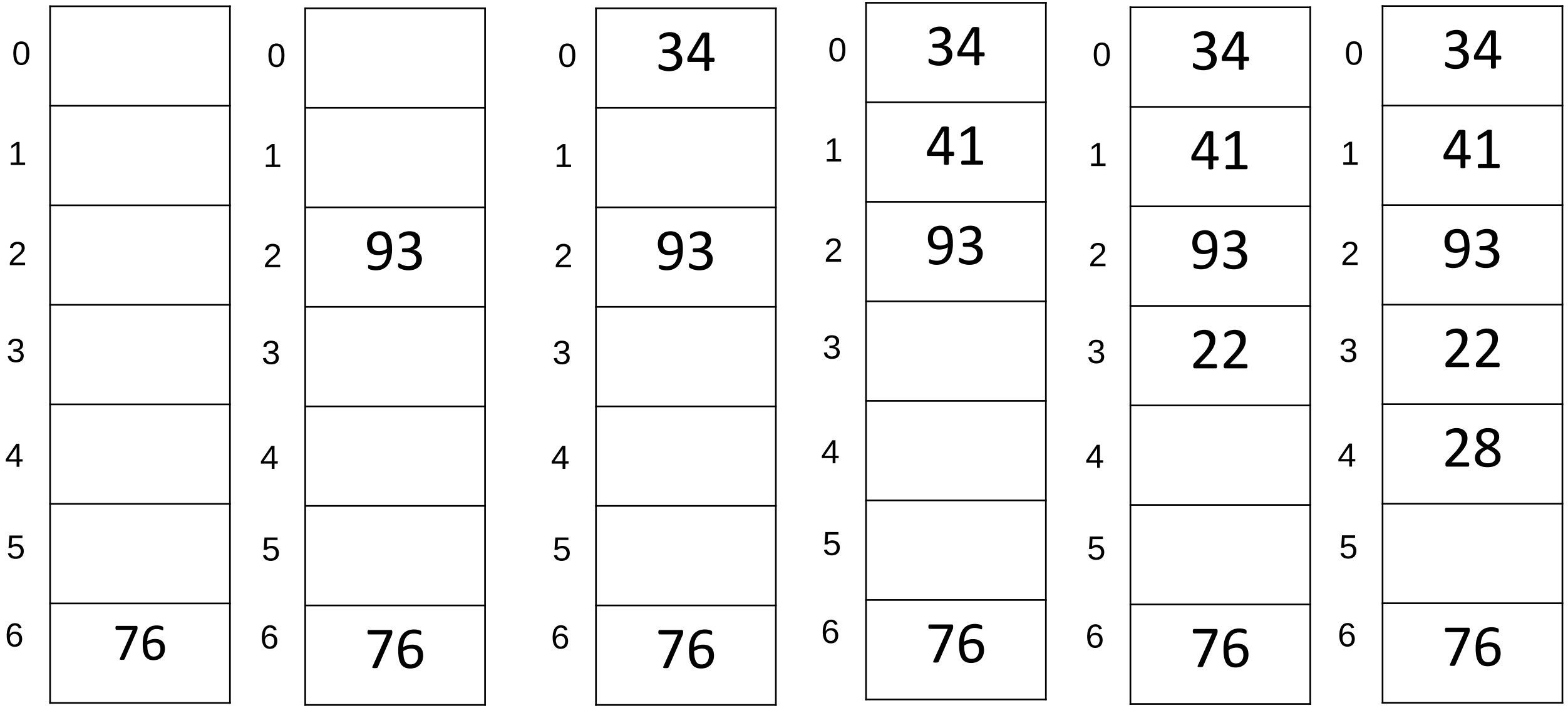
Collision Resolution

Linear Probing – Place value at next, sequential open place



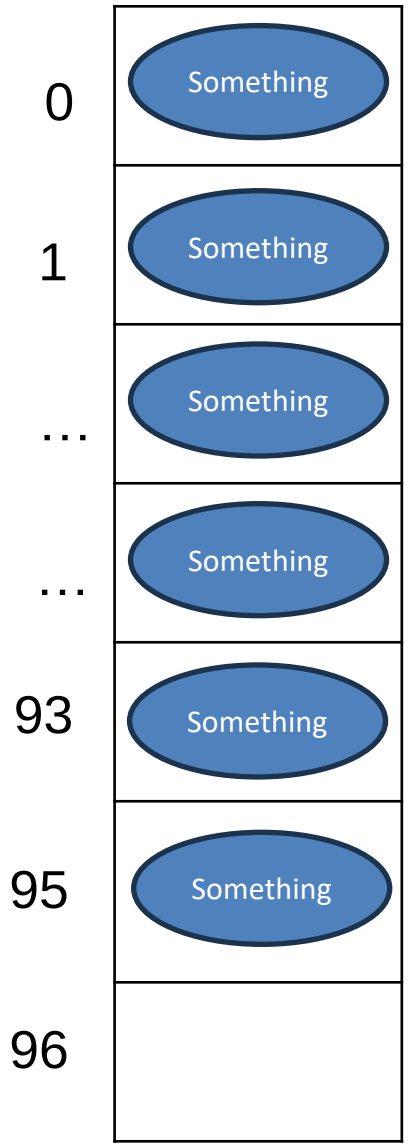
Collision Resolution

Linear Probing – Place value at next, sequential open place



Collision Resolution

Linear Probing – Place value at next, sequential open place

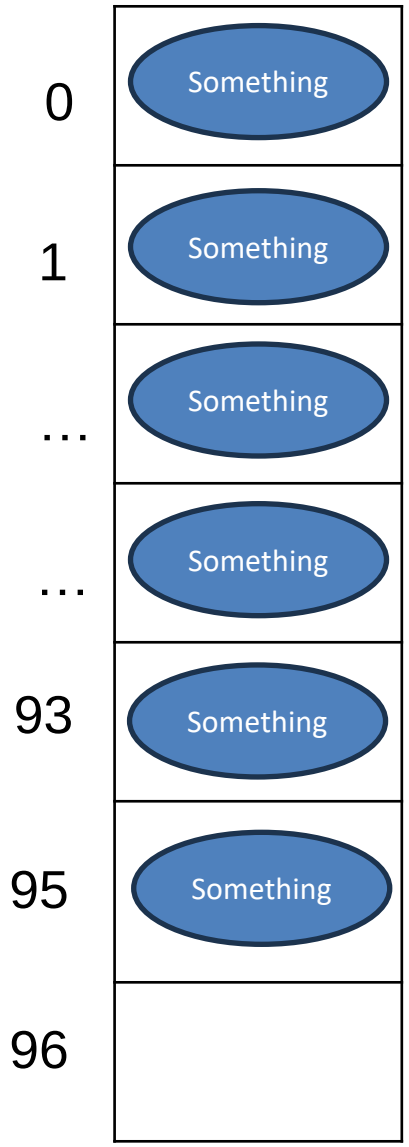


Hash Function:

F(x) = x % 97

Collision Resolution

Linear Probing – Place value at next, sequential open place



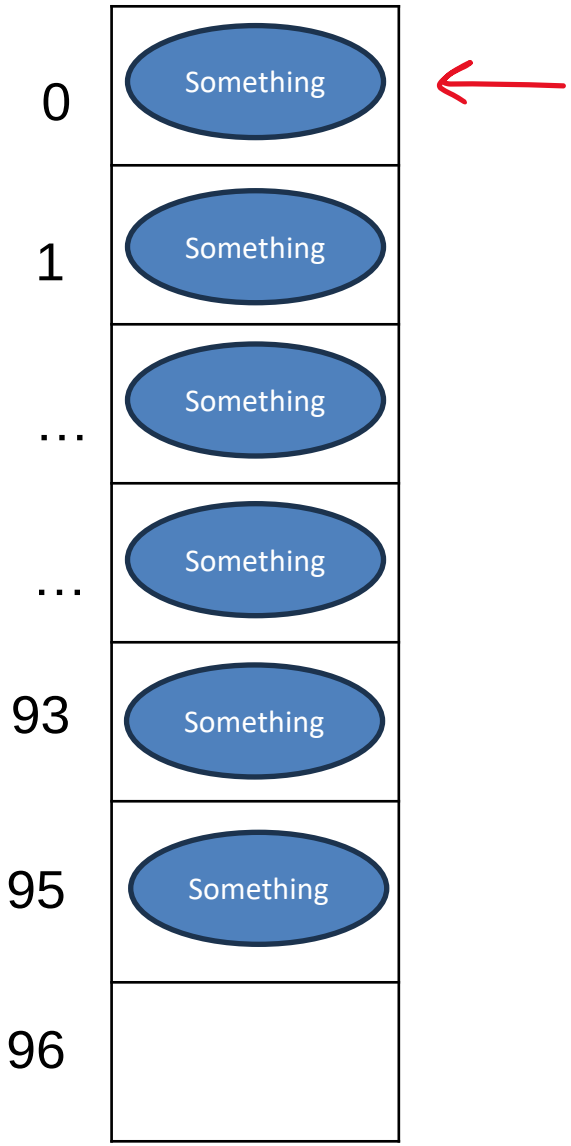
Hash Function:

F(x) = x % 97

```
insert(194)
```

Collision Resolution

Linear Probing – Place value at next, sequential open place



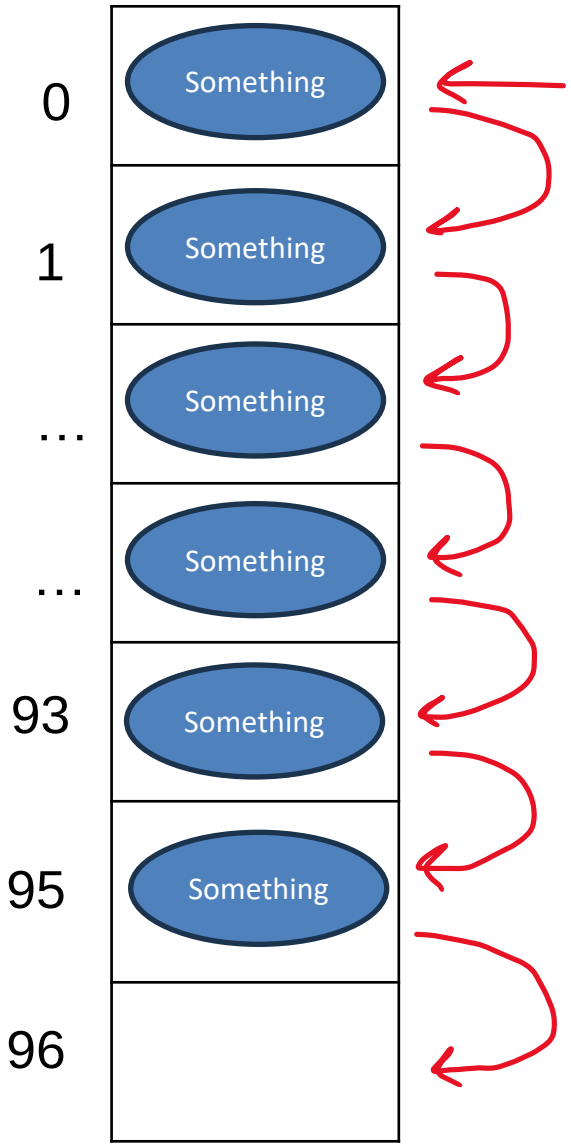
Hash Function:

F(x) = x % 97

```
insert(194)
```

Collision Resolution

Linear Probing – Place value at next, sequential open place



Hash Function:

F(x) = x % 97

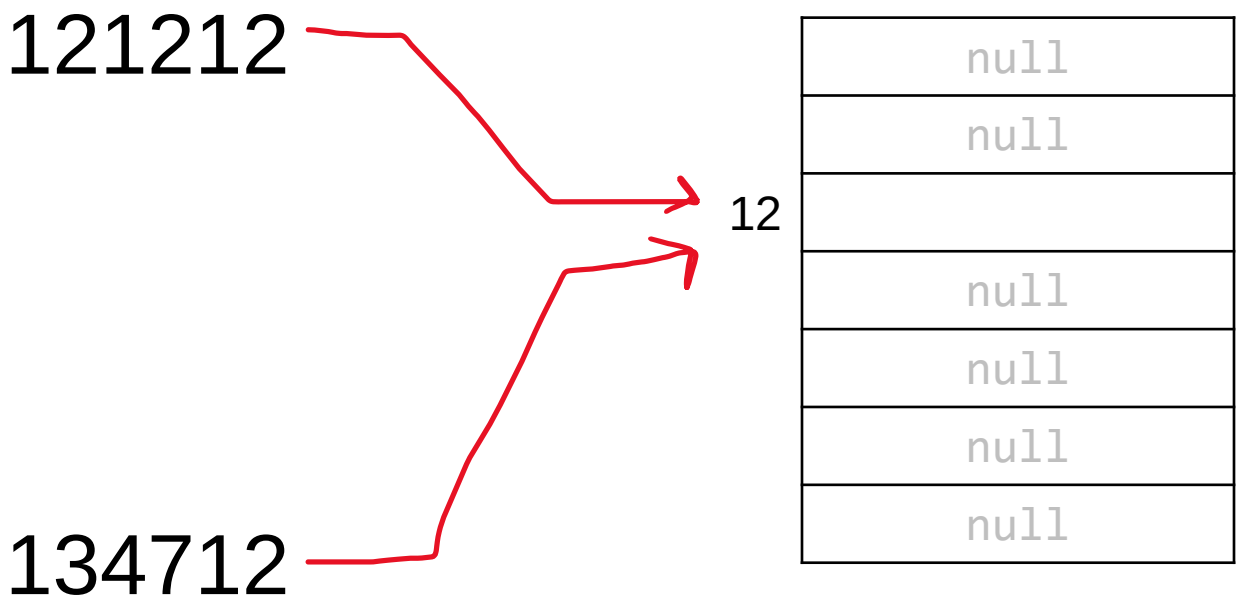
```
insert(194)
```

Worst case scenario, we might have to traverse the entire array before we find an open spot

That is, insertion becomes **O(n)**

Collision Resolution

We know how to avoid collisions (strong hashing function), but collisions are bound to happen with every hash function



How to deal with collisions?

Collision Resolution

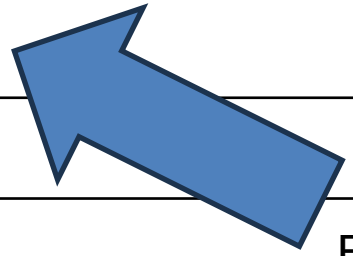
Separate Chaining- Array spots have containers that can hold multiple values

Array Index	Database
0	
1	
…	
…	
11	
…	
30	
…	
45	
…	
99	

Collision Resolution

Separate Chaining- Array spots have containers that can hold multiple values

Array Index	Database
0	< >
1	< >
…	< >
…	< >
11	< >
…	< >
30	< >
…	< >
45	< >
…	< >
99	< >

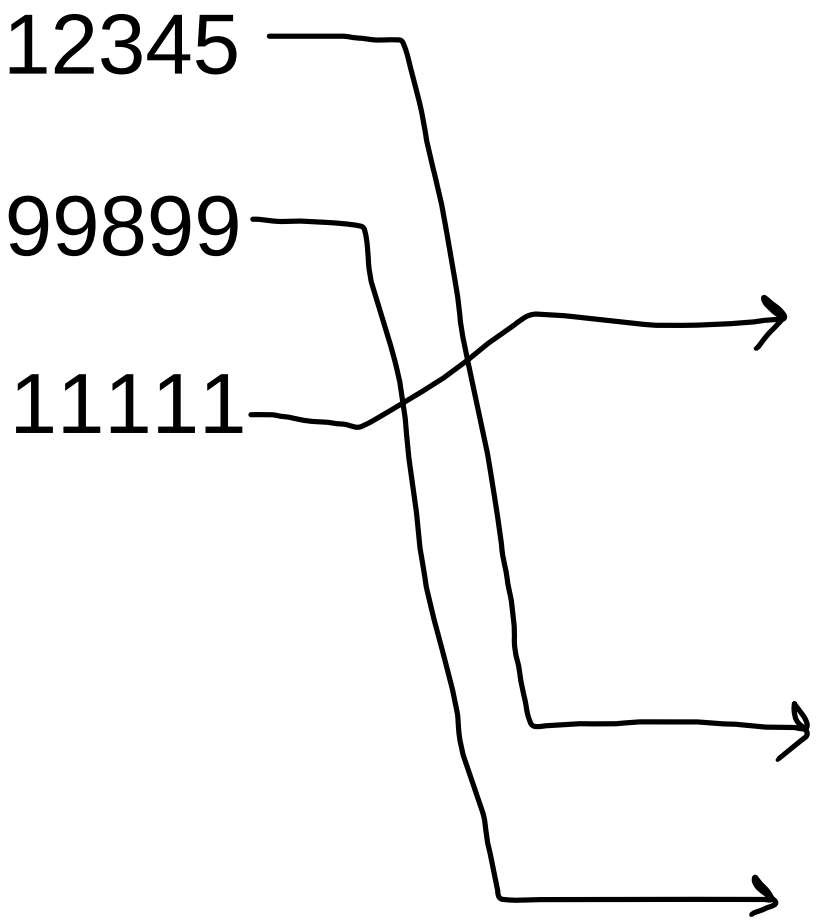


Empty Linked List, Array List, Hash Set

Collision Resolution

Separate Chaining- Array spots have containers that can hold multiple values

`x % 100`

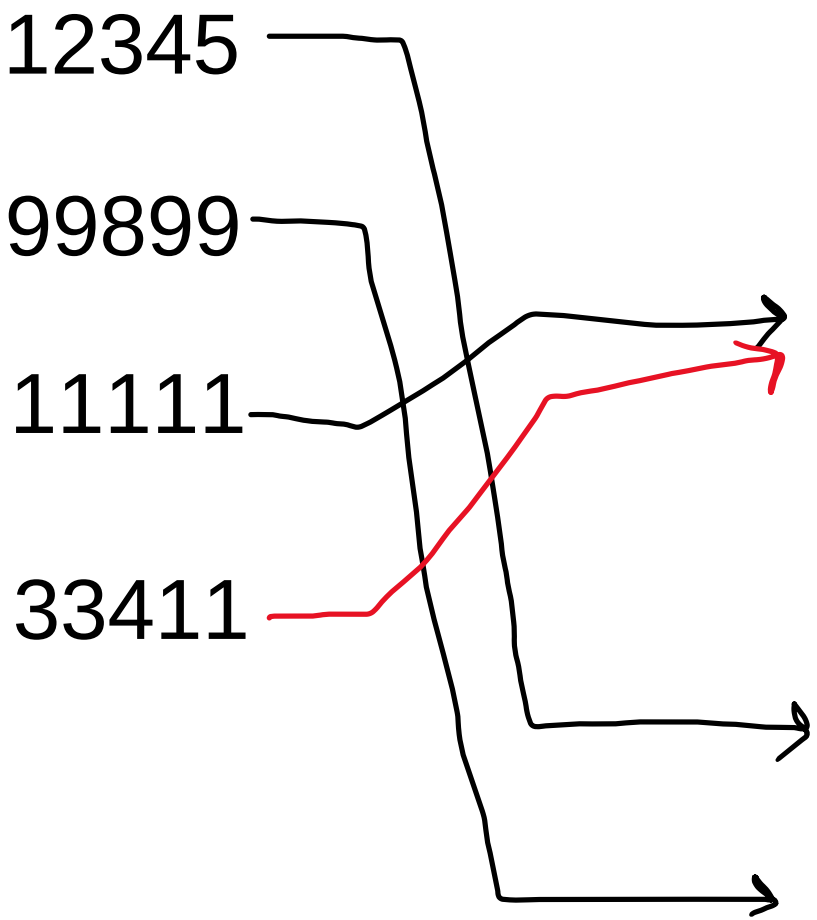


Array Index	Database
0	< >
1	< >
…	< >
…	< >
11	< Student >
…	< >
30	< >
…	< >
45	< Student >
…	< >
99	< Student >

Collision Resolution

Separate Chaining- Array spots have containers that can hold multiple values

`x % 100`

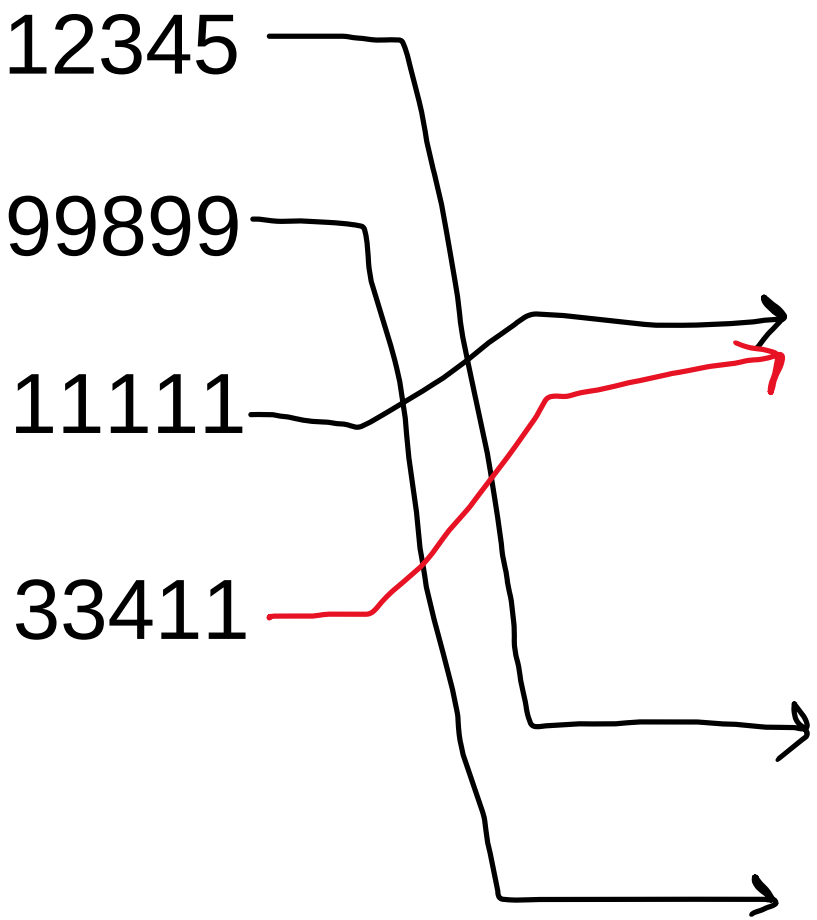


Array Index	Database
0	< >
1	< >
…	< >
…	< >
11	< Student >
…	< >
30	< >
…	< >
45	< Student >
…	< >
99	< Student >

Collision Resolution

Separate Chaining- Array spots have containers that can hold multiple values

`x % 100`

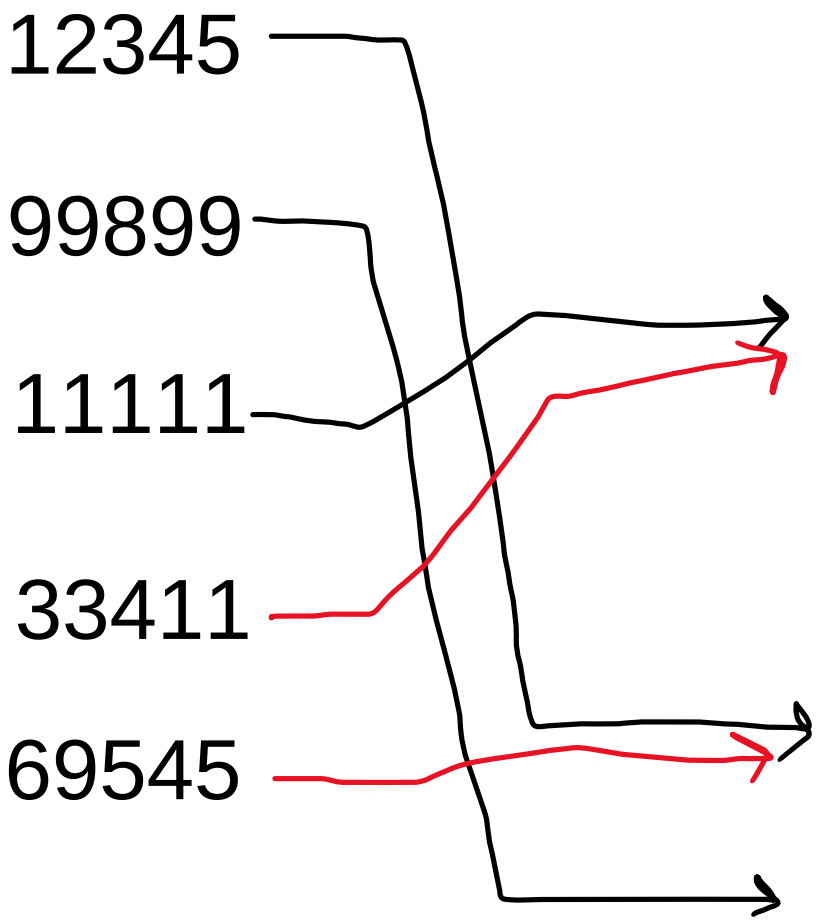


Array Index	Database
0	< >
1	< >
…	< >
…	< >
11	< Student → Student >
…	< >
30	< >
…	< >
45	< Student >
…	< >
99	< Student >

Collision Resolution

Separate Chaining- Array spots have containers that can hold multiple values

`x % 100`

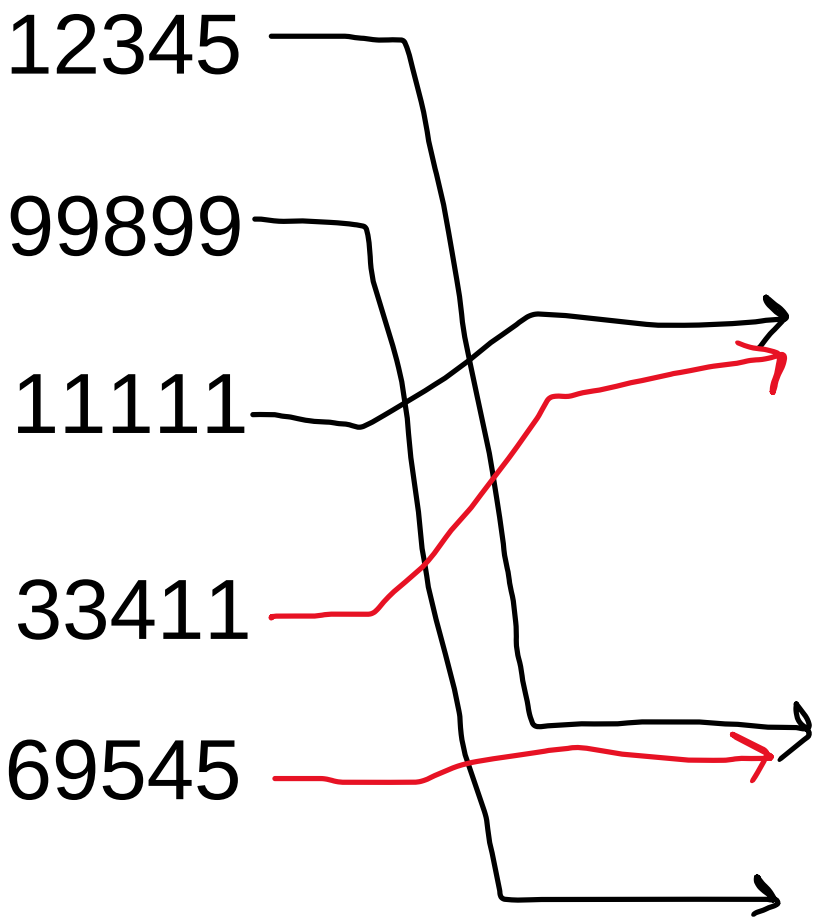


Array Index	Database
0	< >
1	< >
…	< >
…	< >
11	< Student → Student >
…	< >
30	< >
…	< >
45	< Student >
…	< >
99	< Student >

Collision Resolution

Separate Chaining- Array spots have containers that can hold multiple values

```
x % 100
```

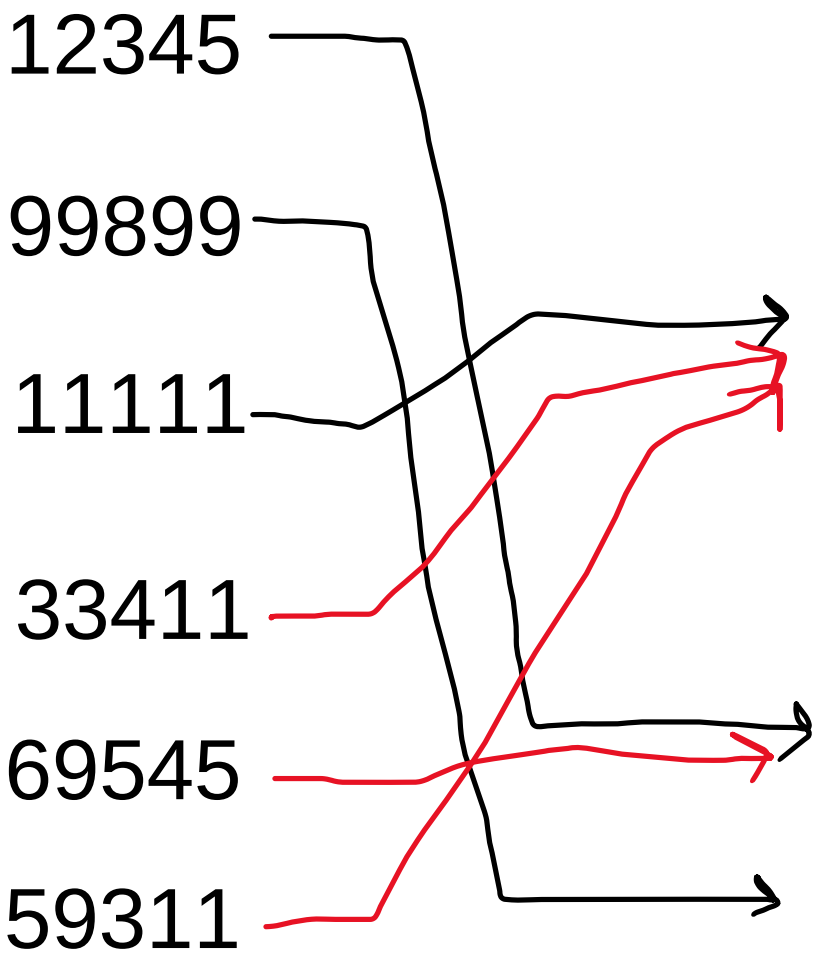


Array Index	Database
0	< >
1	< >
…	< >
…	< >
11	< Student → Student >
…	< >
30	< >
…	< >
45	< Student → Student >
…	< >
99	< Student >

Collision Resolution

Separate Chaining- Array spots have containers that can hold multiple values

`x % 100`



Array Index	Database
0	< >
1	< >
…	< >
…	< >
11	< Student → Student → Student >
…	< >
30	< >
…	< >
45	< Student → Student >
…	< >
99	< Student >

Collision Resolution

Separate Chaining- Array spots have containers that can hold multiple values

```
x % 100
```

```
insert(87611) ?
```

Array Index	Database
0	< >
1	< >
…	< >
…	< >
11	<●→●→●→●→●→●→●→●→●→●>
…	< >
30	< >
…	< >
45	< >
…	< >
99	< >

Collision Resolution

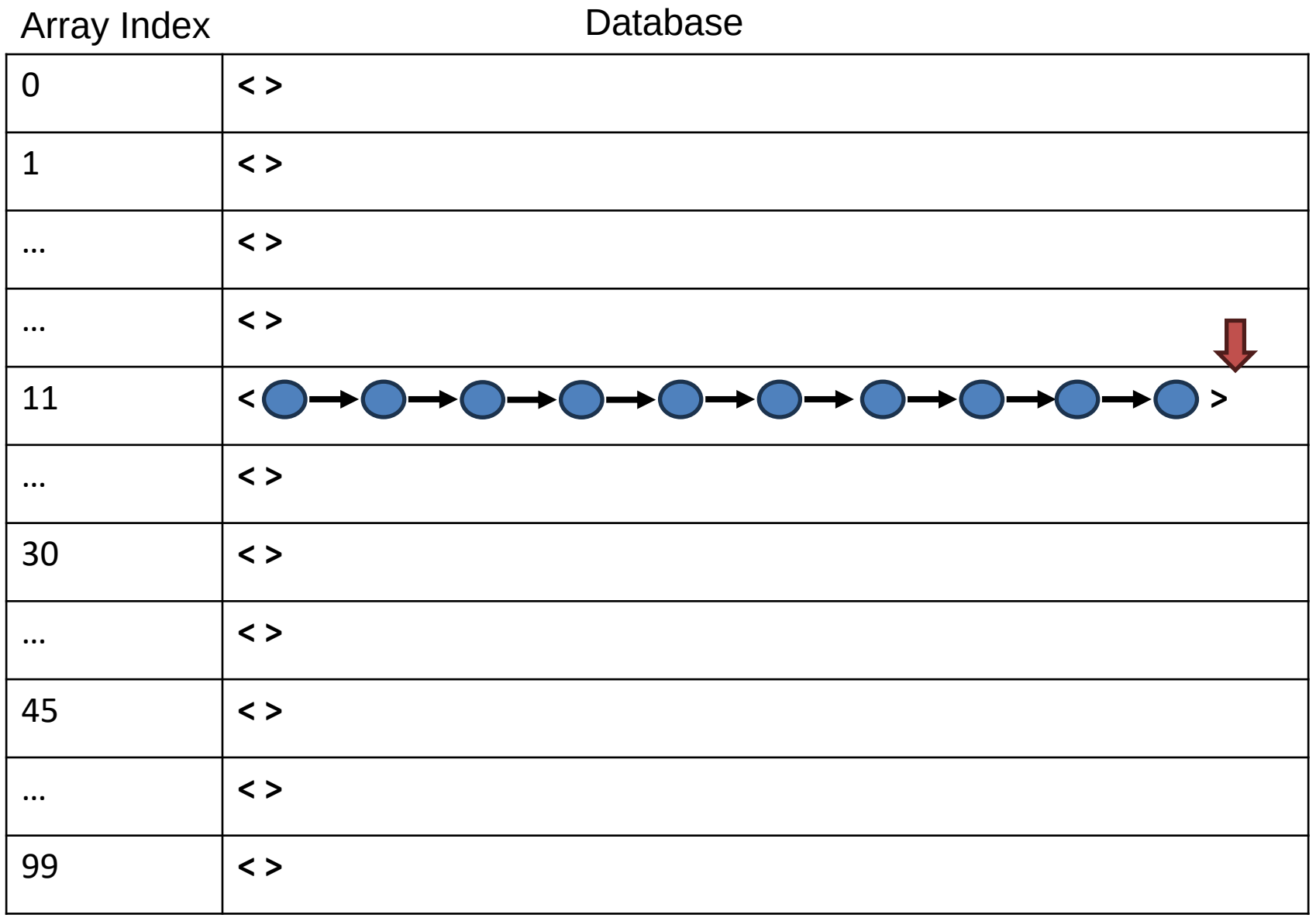
Separate Chaining- Array spots have containers that can hold multiple values

```
x % 100
```

```
insert(87611) ?
```

We might have to travel the entire linked list to find our insertion spot

O (| linked list |)



Collision Resolution

Consequences of collisions?

Our **O(1)** running time for insertions, deletion, and contains turn into **O(n)** 

Thankfully, the Java libraries use smart, complex hashing mechanisms to ensure the collisions are minimized, so we can get **O(1)** with HashMaps, HashSets, etc

Lab 6