The Art of Data Structures

Hashing

Alan Beadle

CSC 162: The Art of Data Structures

UNIVERSITY of ROCHESTER

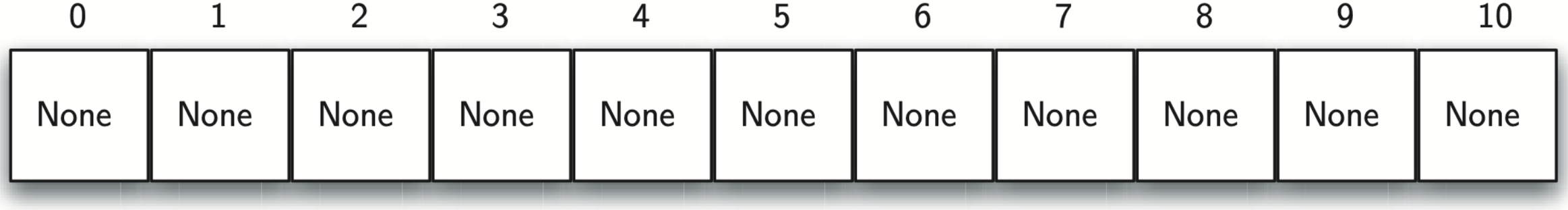
Agenda

- To understand the idea of hashing as a search technique
- To introduce the map ADT
- To implement the map ADT using hashing

Hashing

Hashing

Hash Table with 11 Empty Slots



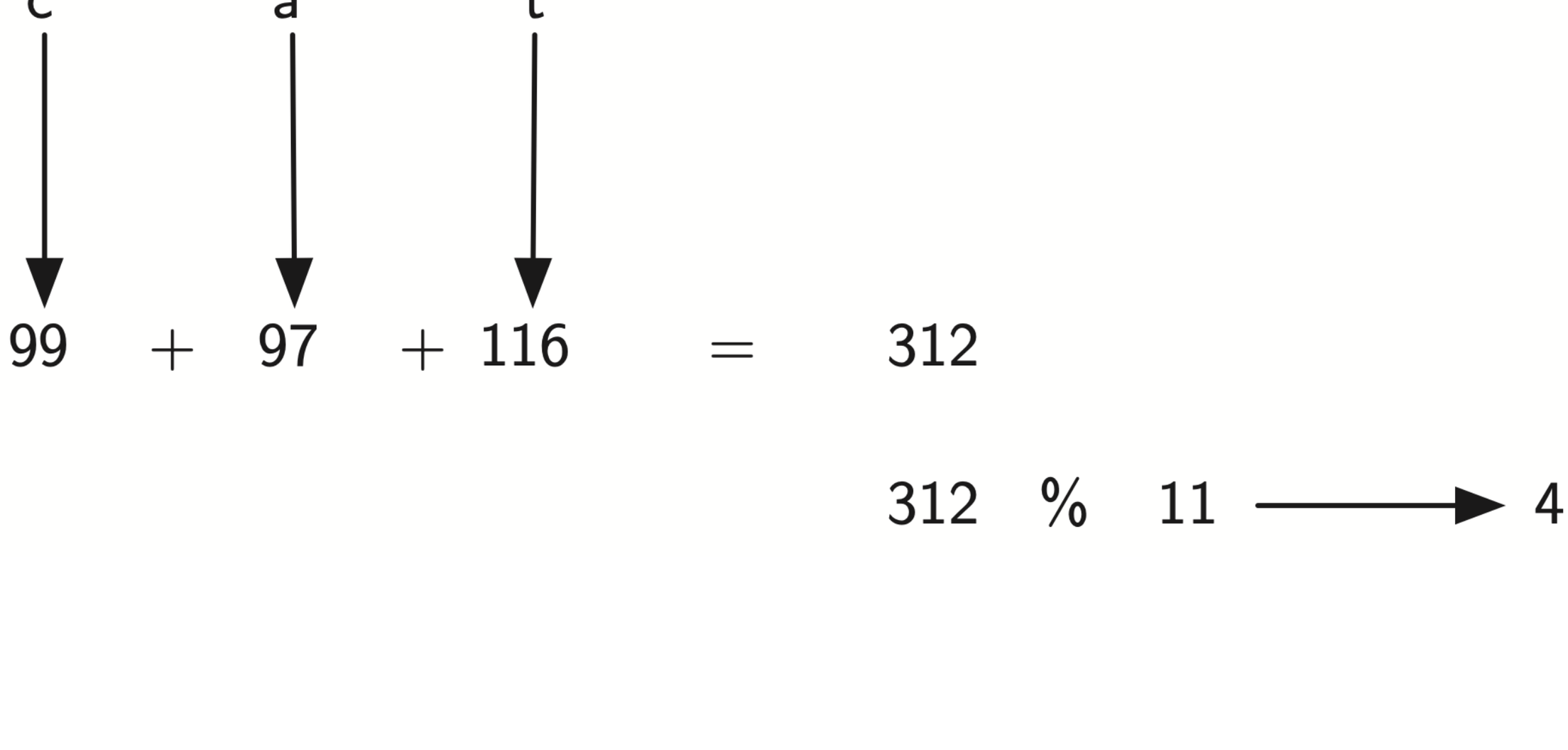
Hashing

Hash Table with Six Items

0	1	2	3	4	5	6	7	8	9	10
77	None	None	None	26	93	17	None	None	31	54

Hashing

A String Using Ordinal Values



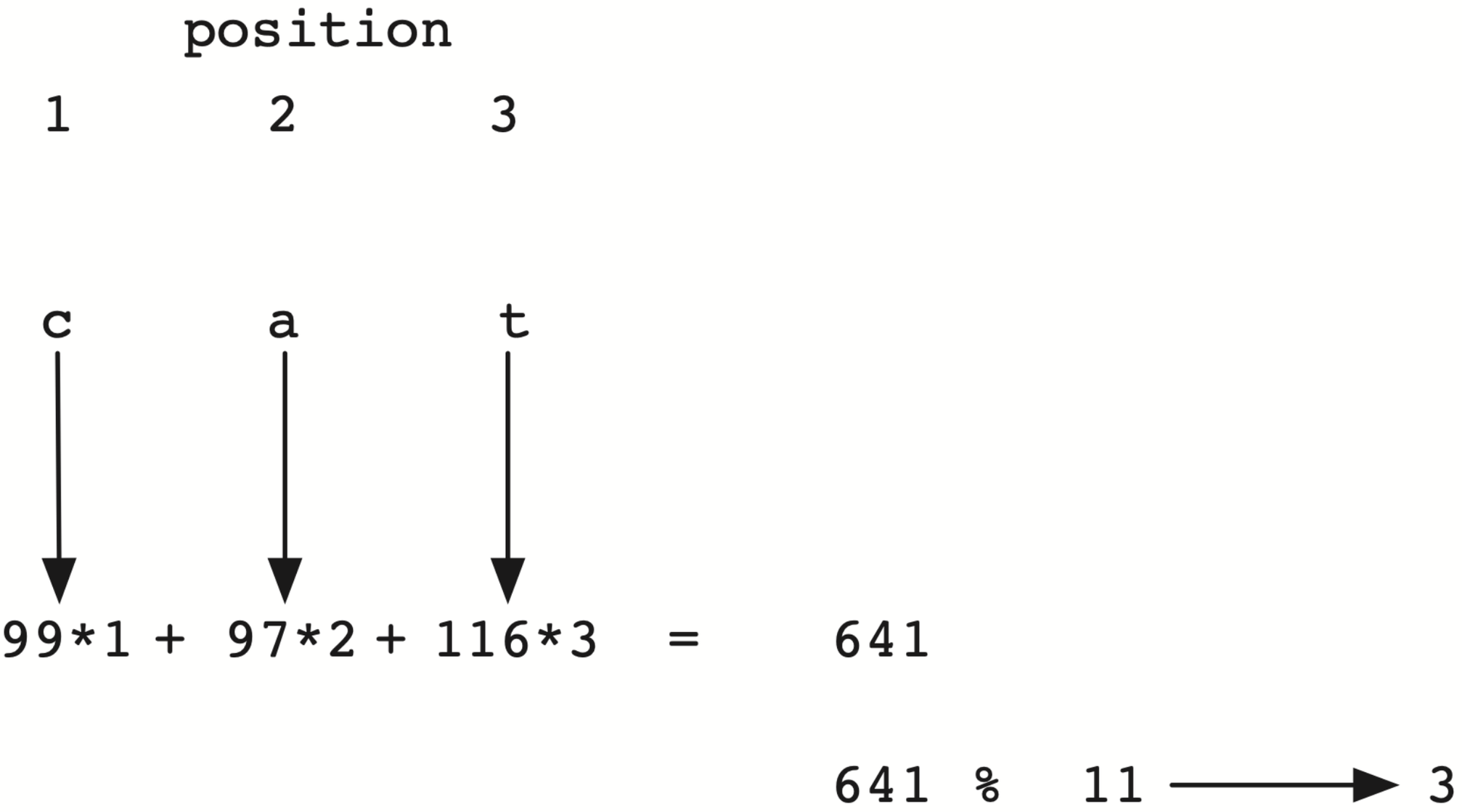
Hashing

Simple Hash Function for Strings

```
def hash(astring, tablesize):
    sum = 0
    for pos in range(len(astring)):
        sum = sum + ord(astring[pos])
    return sum%tablesize
```

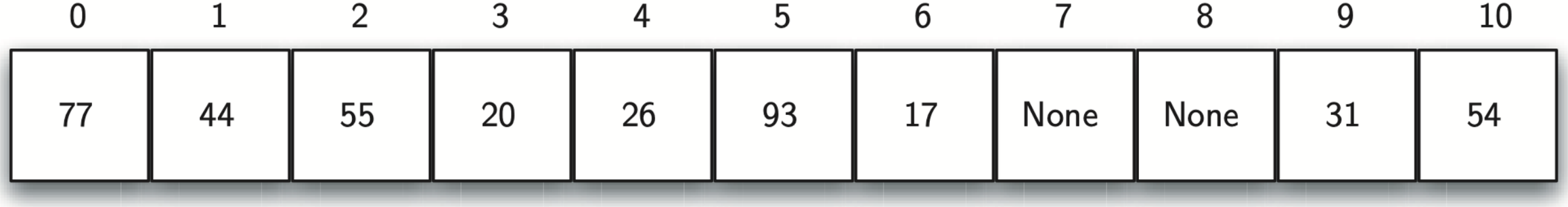
Hashing

A String Using Weighted Ordinal Values



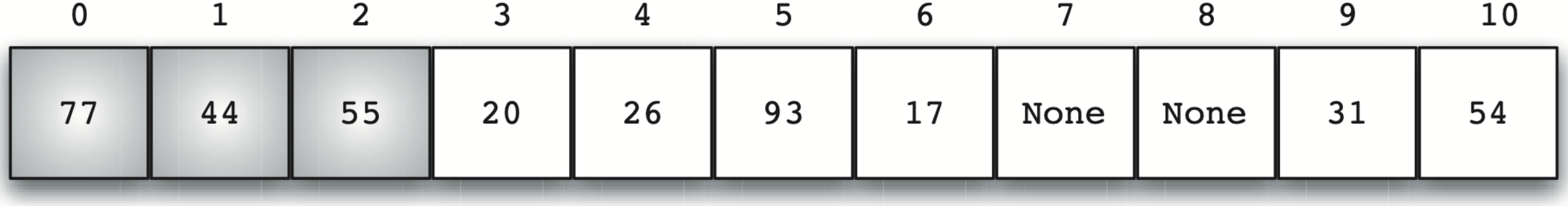
Hashing

Collision Resolution with Linear Probing



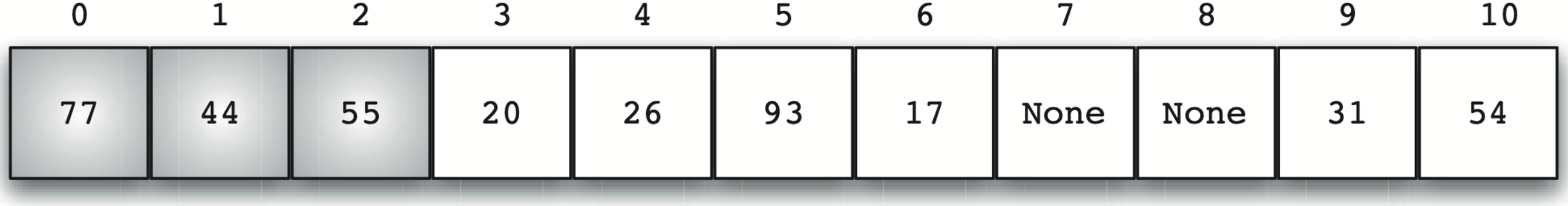
Hashing

A Cluster of Items for Slot 0



Hashing

A Cluster of Items for Slot 0



Hashing

Collision Resolution Using "Plus 3"

0	1	2	3	4	5	6	7	8	9	10
77	55	None	44	26	93	17	20	None	31	54

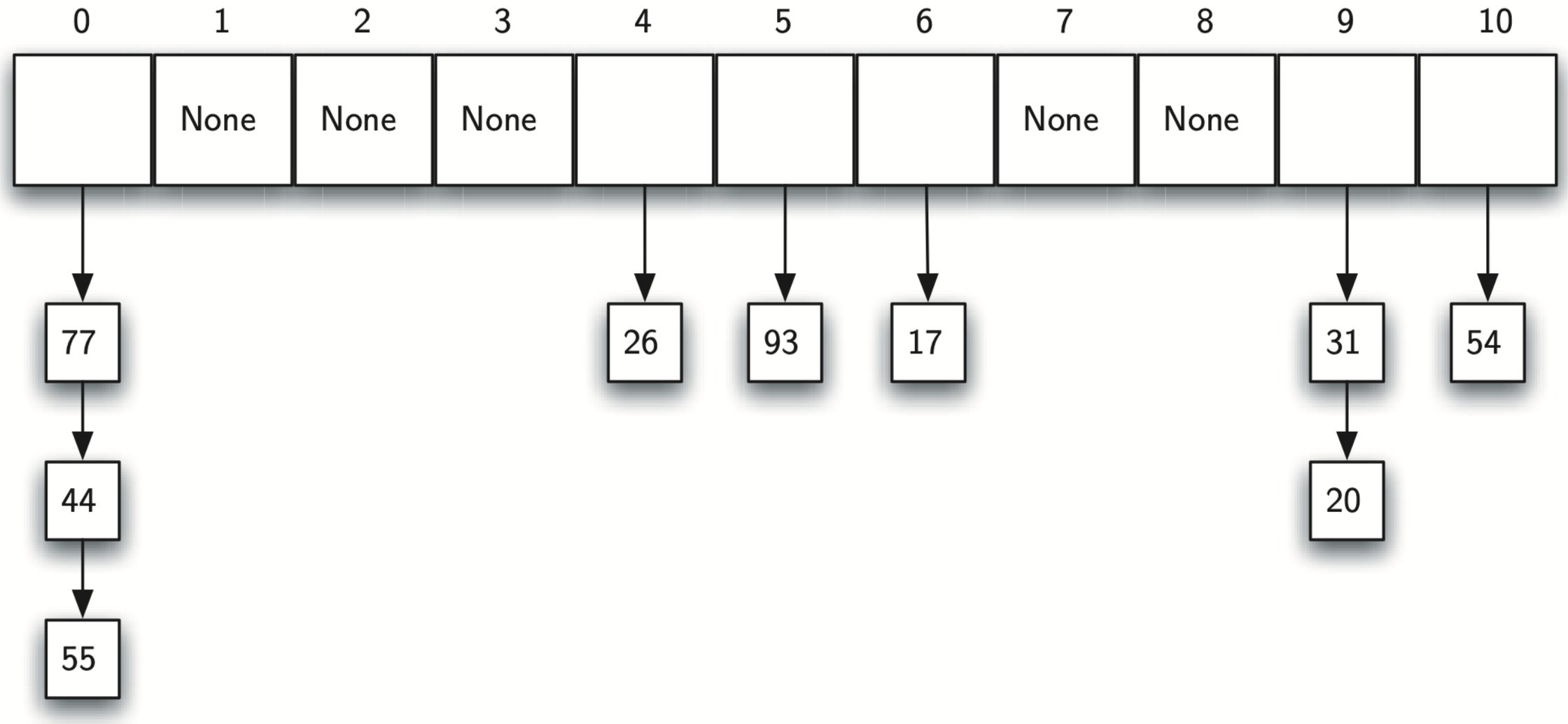
Hashing

Collision Resolution with Quadratic Probing

0	1	2	3	4	5	6	7	8	9	10
77	44	20	55	26	93	17	None	None	31	54

Hashing

Collision Resolution with Chaining



Hashing

Specification

- `HashTable(size)` creates a new hash table
  - It needs the size and returns a hash table with `size` empty slots named `0` through `size-1`
- `put(key, data)` stores a new piece of data in the hash table using the item as the key location
  - It needs the item and the associated data; it returns nothing

Hashing
Specification (cont.)

- `get(key)` returns the data value associated with the key item
  - It returns `None` if the key is not in the hash table
- `hashfunction(key, size)` returns
- `rehash(key, size)` returns

Hashing

`HashTable` *Implementation*

```
class HashTable:
    def __init__(self, size):
        self.size = size
        self.slots = [None] * self.size
        self.data = [None] * self.size
```

Hashing

`HashTable` *Implementation*

```
def put(self, key, data):
    hashvalue = self.hashfunction(key, len(self.slots))

    if self.slots[hashvalue] == None:
      self.slots[hashvalue] = key
      self.data[hashvalue] = data
    else:
      if self.slots[hashvalue] == key:
        self.data[hashvalue] = data  #replace
      else:
        nextslot = self.rehash(hashvalue, len(self.slots))
        while self.slots[nextslot] != None and \
                      self.slots[nextslot] != key:
          nextslot = self.rehash(nextslot, len(self.slots))

        if self.slots[nextslot] == None:
            self.slots[nextslot]=key
            self.data[nextslot]=data
        else:
            self.data[nextslot] = data #replace
```

Hashing

`HashTable` *Implementation*

```
def get(self, key):
    startslot = self.hashfunction(key, len(self.slots))
    data = None
    stop = False
found = False
position = startslot
while self.slots[position] != None and  \
                    not found and not stop:
   if self.slots[position] == key:
     found = True
     data = self.data[position]
   else:
     position=self.rehash(position, len(self.slots))
     if position == startslot:
         stop = True
return data
```

Hashing

`HashTable` *Implementation*

```
def hashfunction(self, key, size):
    return key%size

def rehash(self, oldhash, size):
    return (oldhash+1)%size
```

Questions?