CS-2001
DATA STRUCTURE

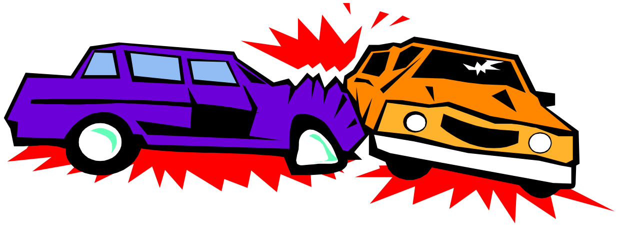
Dr. Hashim Yasin

National University of Computer and Emerging Sciences,

Faisalabad, Pakistan.

HASHING

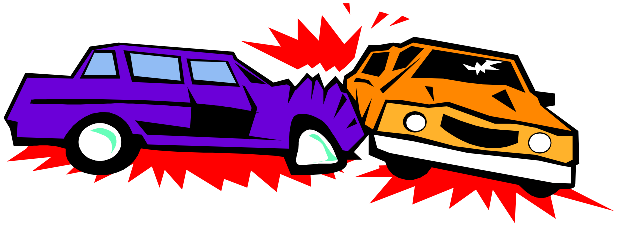
Collision



- The condition resulting when two or more keys produce the same hash location.

- *A good hash function minimizes collisions* by spreading the elements uniformly throughout the array.

Collision



- Collision handling techniques
  - Linear Probing
  - Rehashing
  - Double Hashing
  - Quadratic Probing
  - Random Probing
  - Buckets
  - Chaining

Collision Resolution Techniques

- There are two broad ways of collision resolution:

1. Open Addressing: Array-based implementation.

   (i) Linear probing (linear search)
   (ii) Quadratic probing (nonlinear search)
   (iii) Double hashing (uses two hash functions)

2. Separate Chaining: A linked list implementation

Collision Resolution Techniques

- Collision resolution techniques can be broken into two classes:
  - **open hashing** (also called **separate chaining**)
  - **closed hashing** (also called **open addressing**).
- The **difference** between the two has to do with
  - whether collisions are stored outside the table (open hashing),
  - or whether collisions result in storing one of the records at another slot in the table (closed hashing).

BUCKET

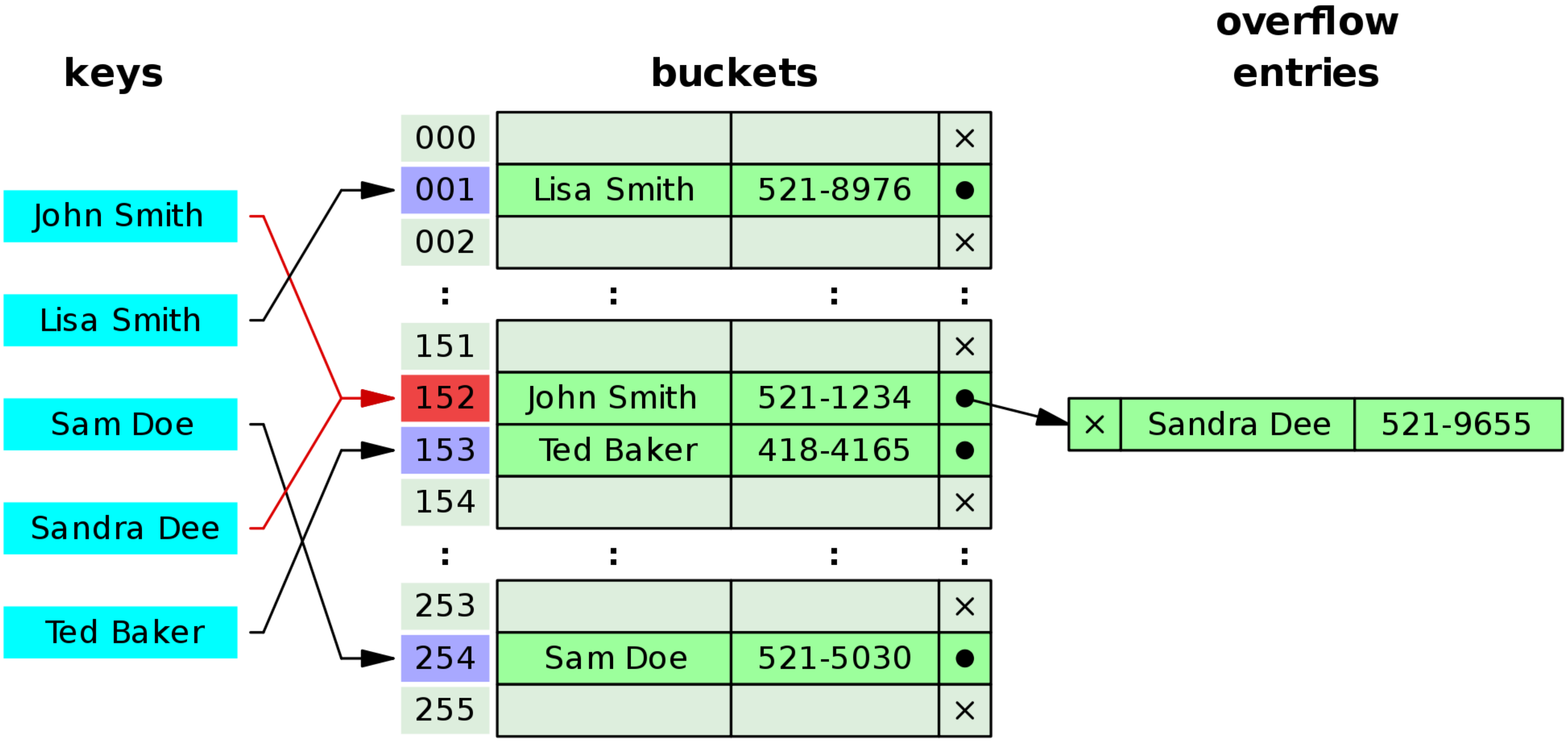
Bucket

- A collection of elements associated with a particular hash location
- Handle collision by allowing multiple-element keys to hash to the same location
- A solution is to let each computed hash location contain slots for multiple elements
- Each of these multi-element locations is called a bucket

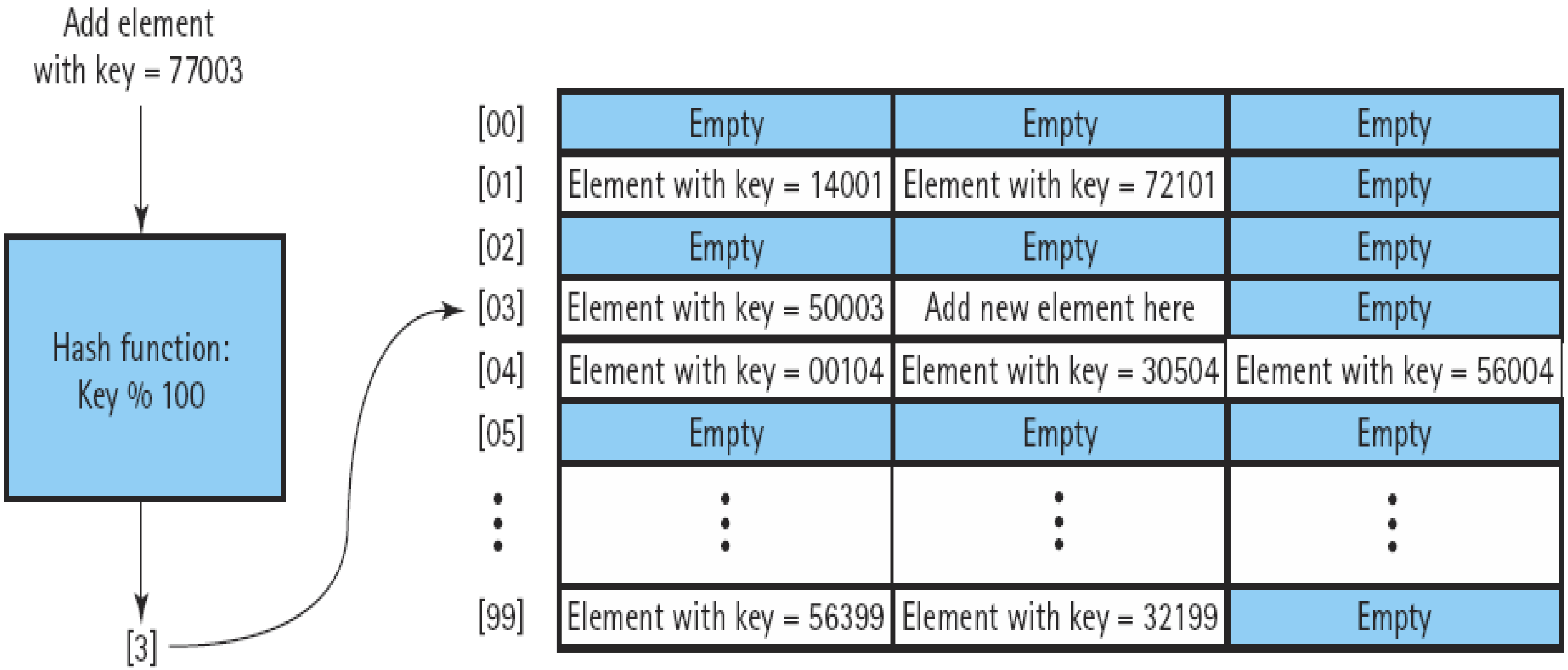
Bucket

- Slots are grouped into buckets
- The hash function transforms the key into a bucket number
- Each bucket contains B slots, and no collision occurs until the bucket is full.
- At that point you need to apply a collision processing strategy to find another bucket

Bucket-Example



Bucket



[00]	Empty	Empty	Empty
[01]	Element with key = 14001	Element with key = 72101	Empty
[02]	Empty	Empty	Empty
[03]	Element with key = 50003	Add new element here	Empty
[04]	Element with key = 00104	Element with key = 30504	Element with key = 56004
[05]	Empty	Empty	Empty
⋮	⋮	⋮	⋮
[99]	Element with key = 56399	Element with key = 32199	Empty

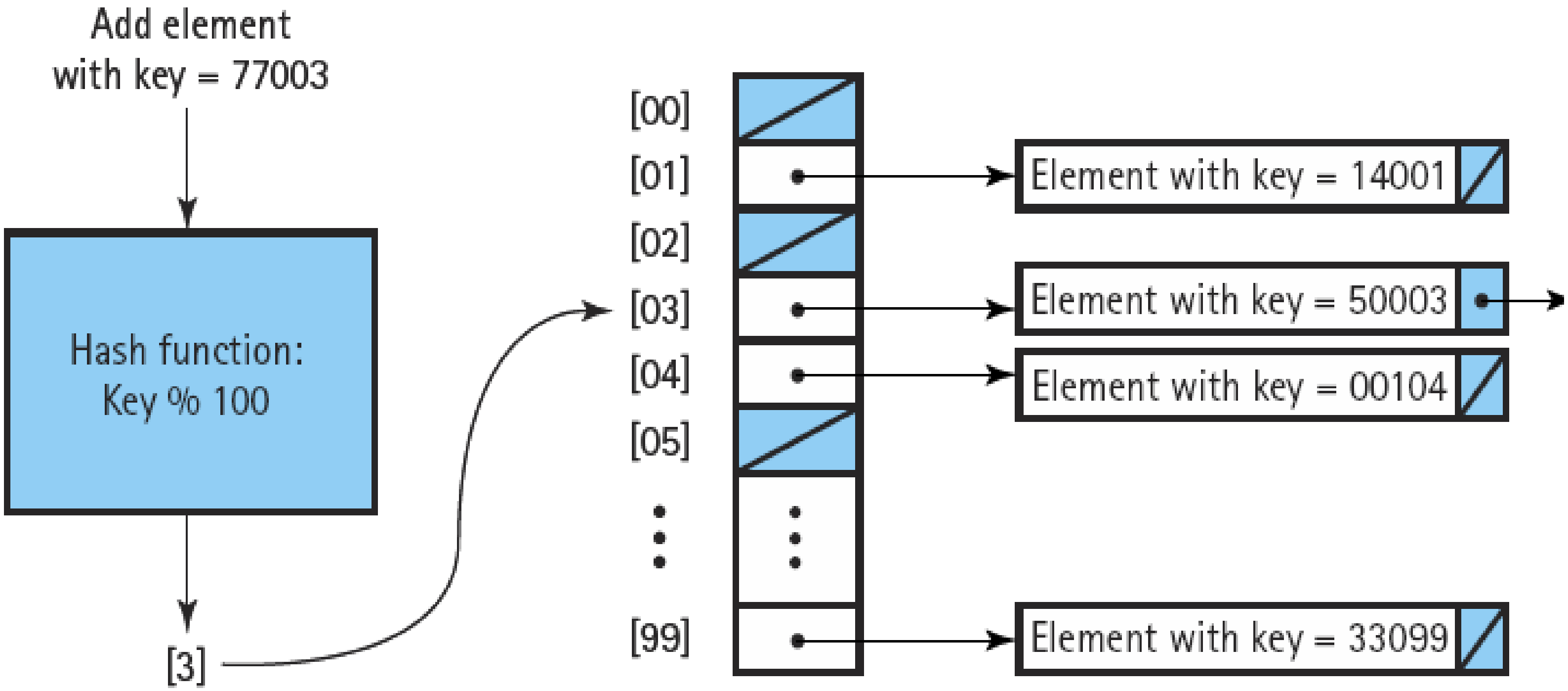
- When the bucket becomes full, we must again deal with the problem of handling collision

CHAIN

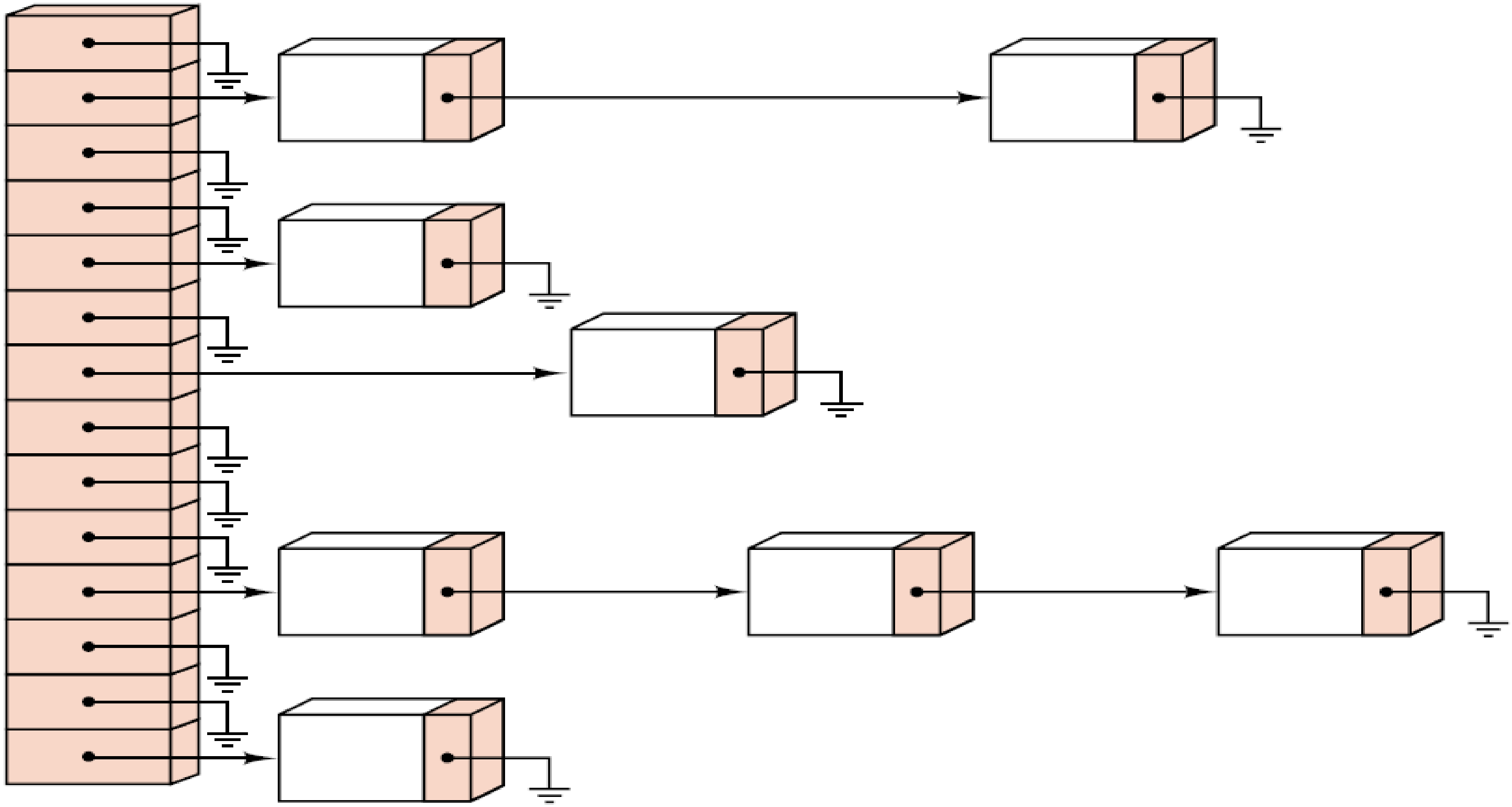
Chain

- A linked list of elements that share the same hash location
- Use the hash value not as the actual location of the element, but rather as the *index into an array of pointer*
- *Each pointer accesses a chain of elements that share the same hash location*
- Good when deleting an element from the linked list

Chain

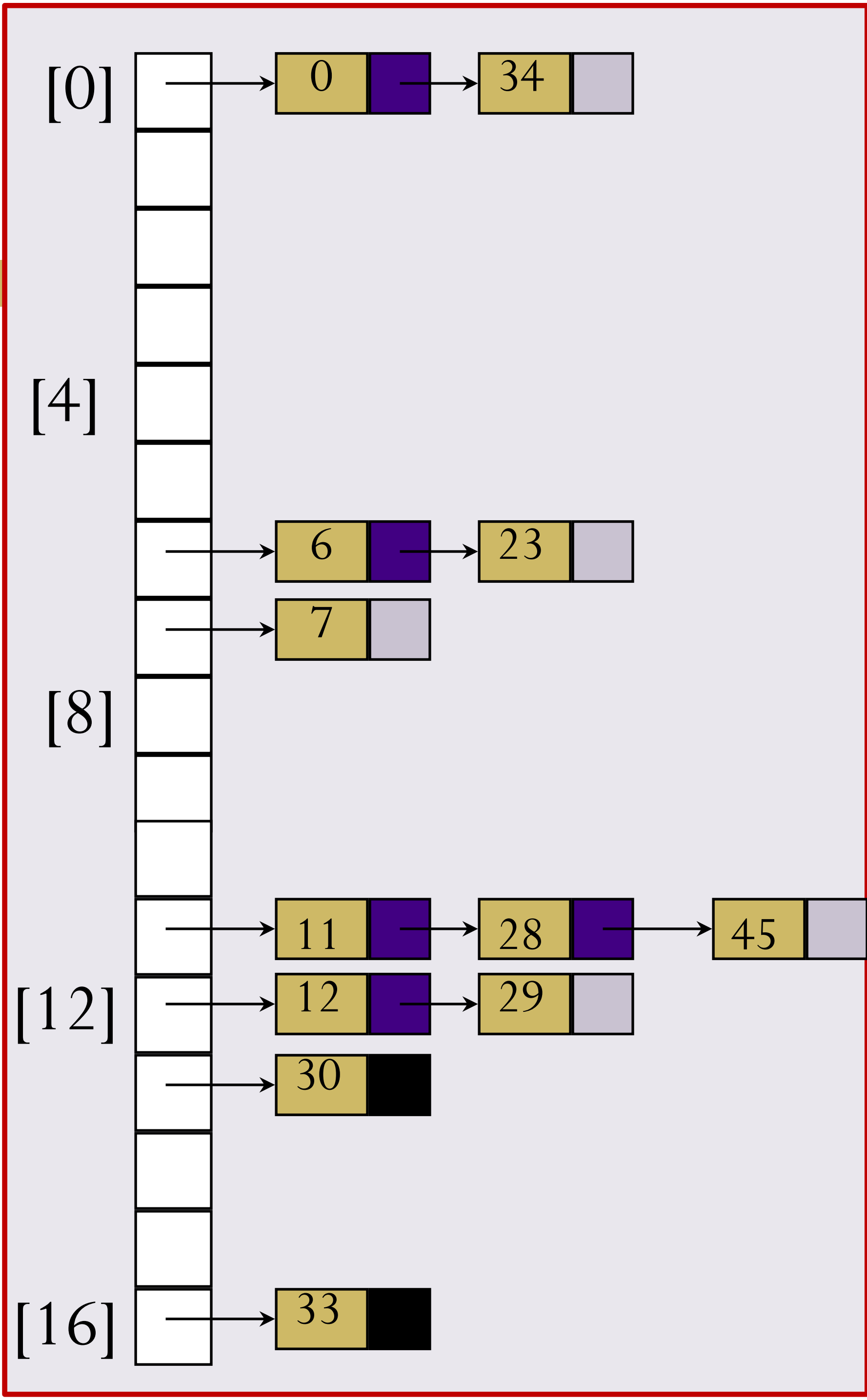

Add element
with key = 77003
Hash function:
Key % 100
[3]
[00]
[01]
[02]
[03]
[04]
[05]
[99]
Element with key = 14001
Element with key = 50003
Element with key = 00104
Element with key = 33099

Chain



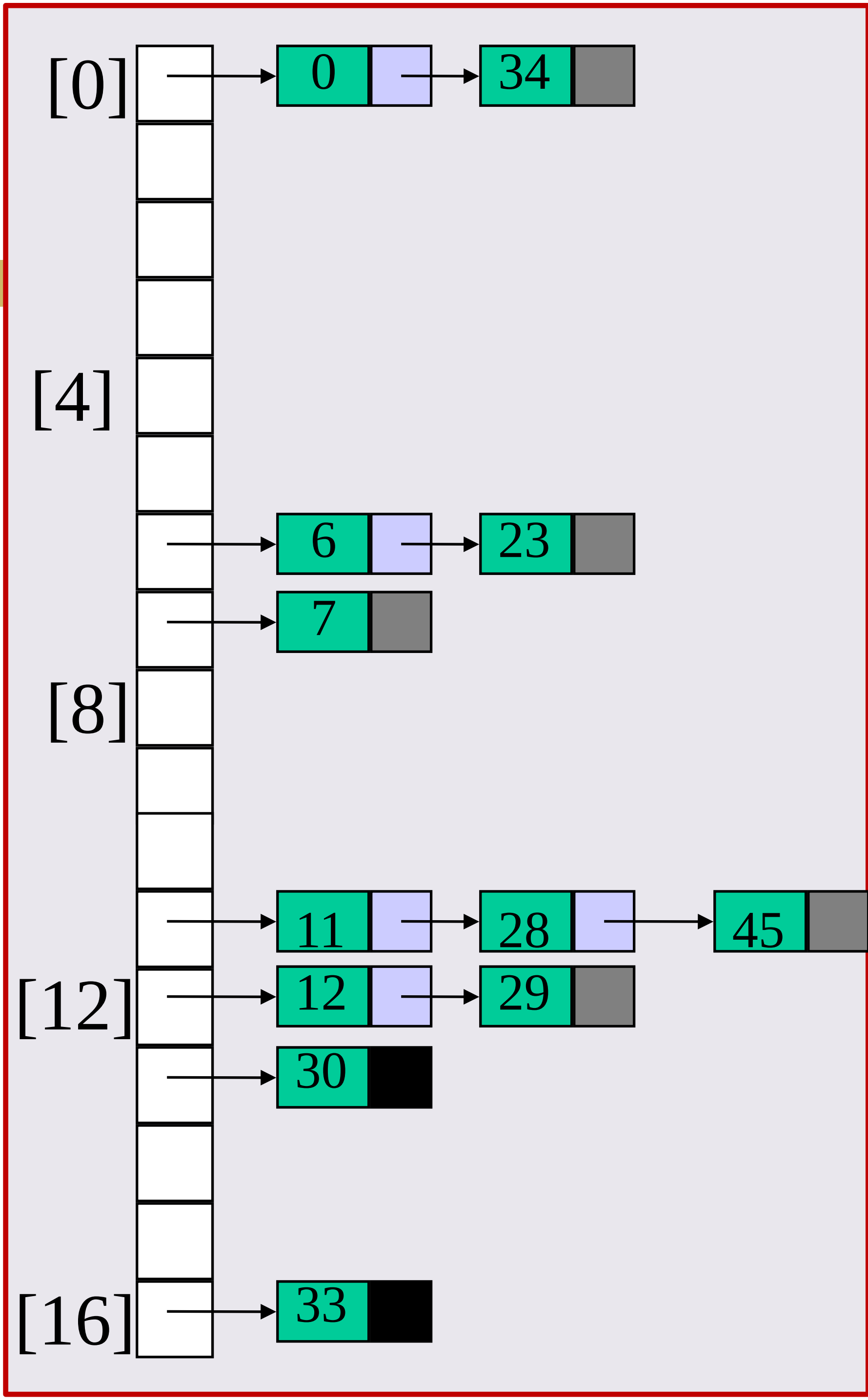
Chain

- Use linked list
  - to connect the identifiers with the same hash value and
  - to increase the capacity of a bucket.

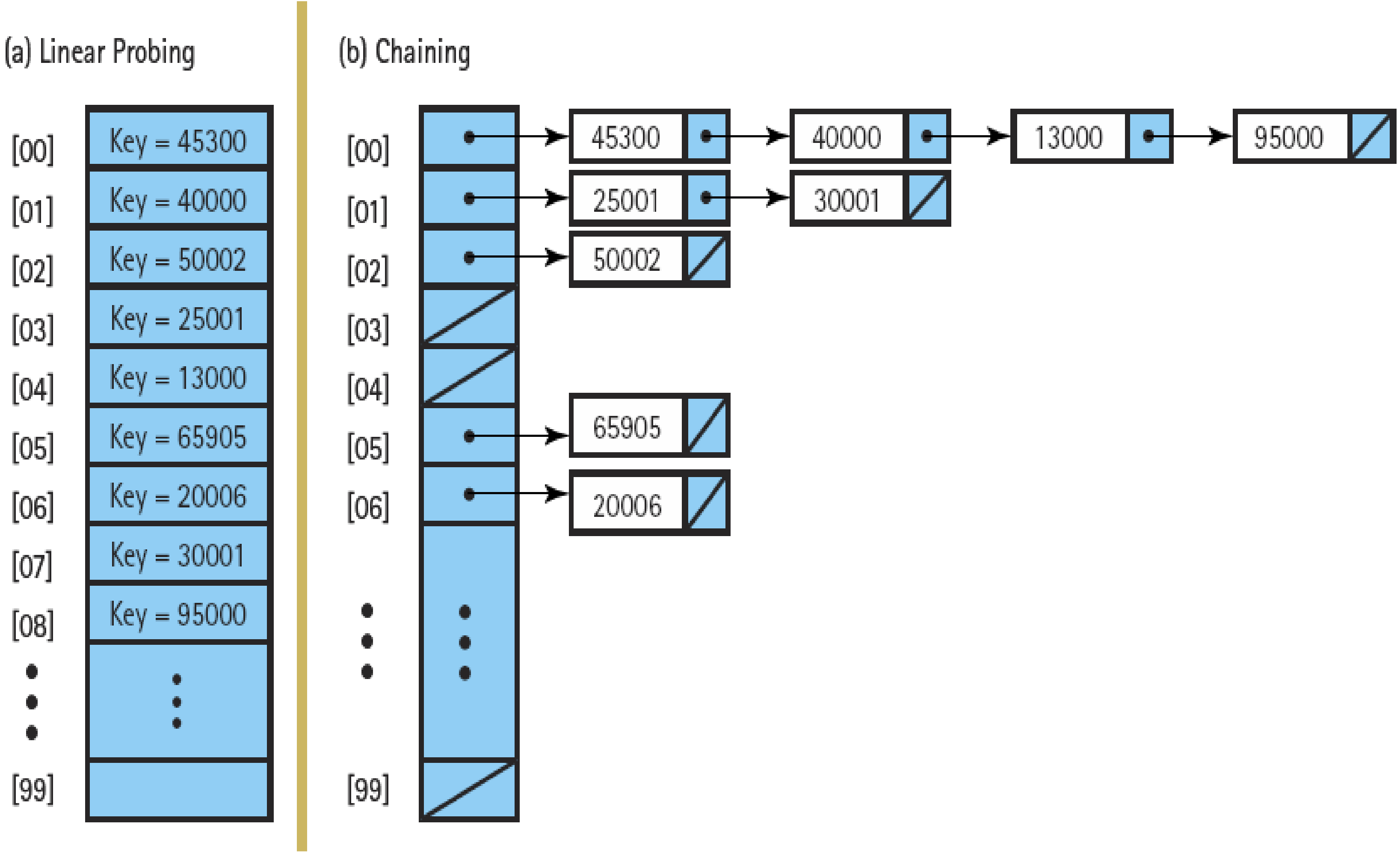


Sorted Chains

- Put in pairs whose keys are 6, 12, 34, 29, 28, 11, 23, 7, 0, 33, 30, 45
- Bucket = key % 17.



Comparison – Linear Probing & Chaining



(a) Linear Probing

[00]	Key = 45300
[01]	Key = 40000
[02]	Key = 50002
[03]	Key = 25001
[04]	Key = 13000
[05]	Key = 65905
[06]	Key = 20006
[07]	Key = 30001
[08]	Key = 95000
• • •	⋮
[99]	

(b) Chaining

[00] → 45300 → 40000 → 13000 → 95000

[01] → 25001 → 30001

[02] → 50002

[03]

[04]

[05] → 65905

[06] → 20006

• • •

[99]

SEPARATE CHAINING

Separate Chaining

- Retrieval of an item, **`r`**, with hash address, **`i`**, is simply retrieval from the linked list at position **`i`**.

- Deletion of an item, **`r`**, with hash address, **`i`**, is simply deleting **`r`** from the linked list at position **`i`**.

Separate Chaining

Example: Load the keys **23, 13, 21, 14, 7, 8, and 15** , in this order, in a hash table of size **7** using separate chaining with the hash function: **h(key) = key % 7**

h(23) = 23 % 7 = 2

h(13) = 13 % 7 = 6

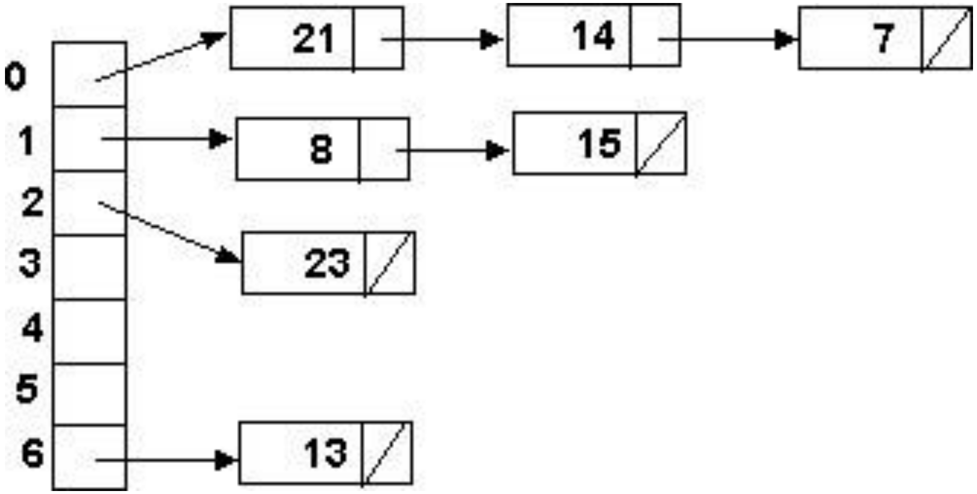
h(21) = 21 % 7 = 0

h(14) = 14 % 7 = 0 collision

h(7) = 7 % 7 = 0 collision

h(8) = 8 % 7 = 1

h(15) = 15 % 7 = 1 collision

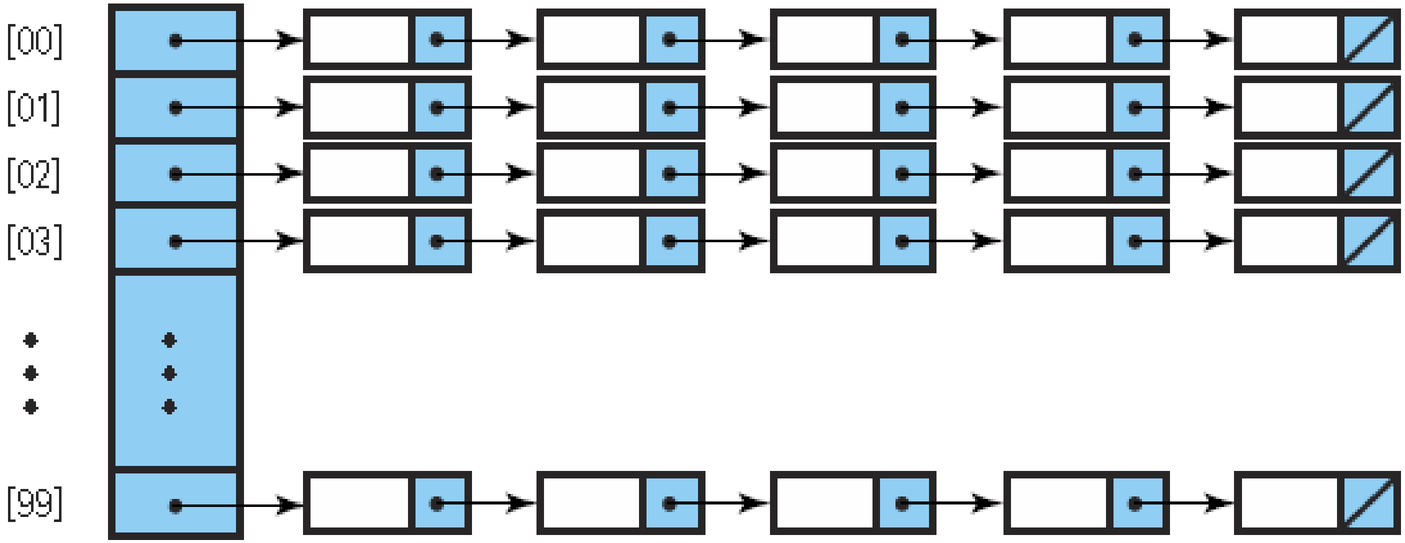


Designing a good Hash Function

- A good hash function minimize the collisions
- One Solution
  - Use a data structure that has more space for keys
- Another Solution
  - Design hash function to minimize the collisions
  - Produce unique keys as much as possible
- To avoid collision causing worst case need to know statistical distribution of keys.

Choosing Good Hash Function

(a) The plan



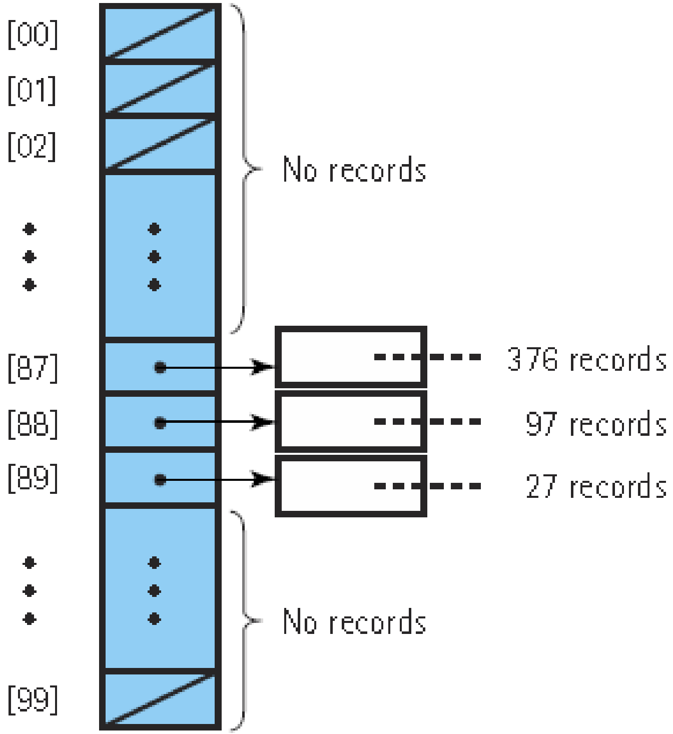
Average 5 records/chain

5 records × 100 chains = 500 employees

Expected search – 0(5)

Contd..

(b) The reality



376 employees hired in 1987
97 employees hired in 1988
27 employees hired in 1989

500 employees
Actual search O(N)

Contd…

- While choosing a good hash function consider the following two things:
- **First,** *consider the efficiency of calculating the function.*
    - Even if a hash function always produces unique values, it is not a good hash function if ***it takes longer to calculate the hash value than to search half the list***.

Contd..

- **Second**, *consider the program time*.
  - A function that somehow produces unique hash values for all of the known key values may fail if the *domain of possible key values changes* in a later modification.
  - The programmer who has to modify the program may then waste a lot of time trying to find another hash function that is equally clever.

Separate Chaining versus Open-addressing

- **Open Addressing**
  - All items are stored in the hash table itself.
  - In addition to the cell data (if any), each cell keeps one of the three states: EMPTY, OCCUPIED, DELETED.
  - While inserting, if a collision occurs, alternative cells are tried until an empty cell is found.

Separate Chaining versus Open-addressing

Disadvantages of Separate Chaining:

- It requires the implementation of a separate data structure for chains, and code to manage it.
- The main cost of chaining is the extra space required for the linked lists.
- For some languages, creating new nodes (for linked lists) is expensive and slows down the system.

References

- Nell Dale – Chapter 10.
- http://www.cplusplus.com/doc/tutorial/templates/
- Robert Lafore, Chapter 14, Page 681