

COL106: Data Structures and Algorithms

Lecture 9

Keerti Choudhary
Department of Computer Science, IIT Delhi

Today's Lecture

1. Dictionaries
2. Hashing, Collision Resolution Techniques in Hashing

Dictionary Data structure

- A **dictionary** store elements so that they can be located very quickly using keys.
- **Example:** A dictionary may store bank accounts:
 - Each account is an object that can be identified by unique account number.
 - Each account may store additional information like:
Current balance, Name, Address, Aadhar number, Transaction history, etc.
- **Some operations that needs to be supported:**
 - **Size()**, **IsEmpty()**, **ListElements()**
 - Query:
SearchElement(key k)
 - Update:
InsertItem(key k , element e), **RemoveElement**(key k)
- **Note:** There need not be a notion of predecessor/successor on keys.
Only comparison operation on keys is needed.

Dictionary Data structure

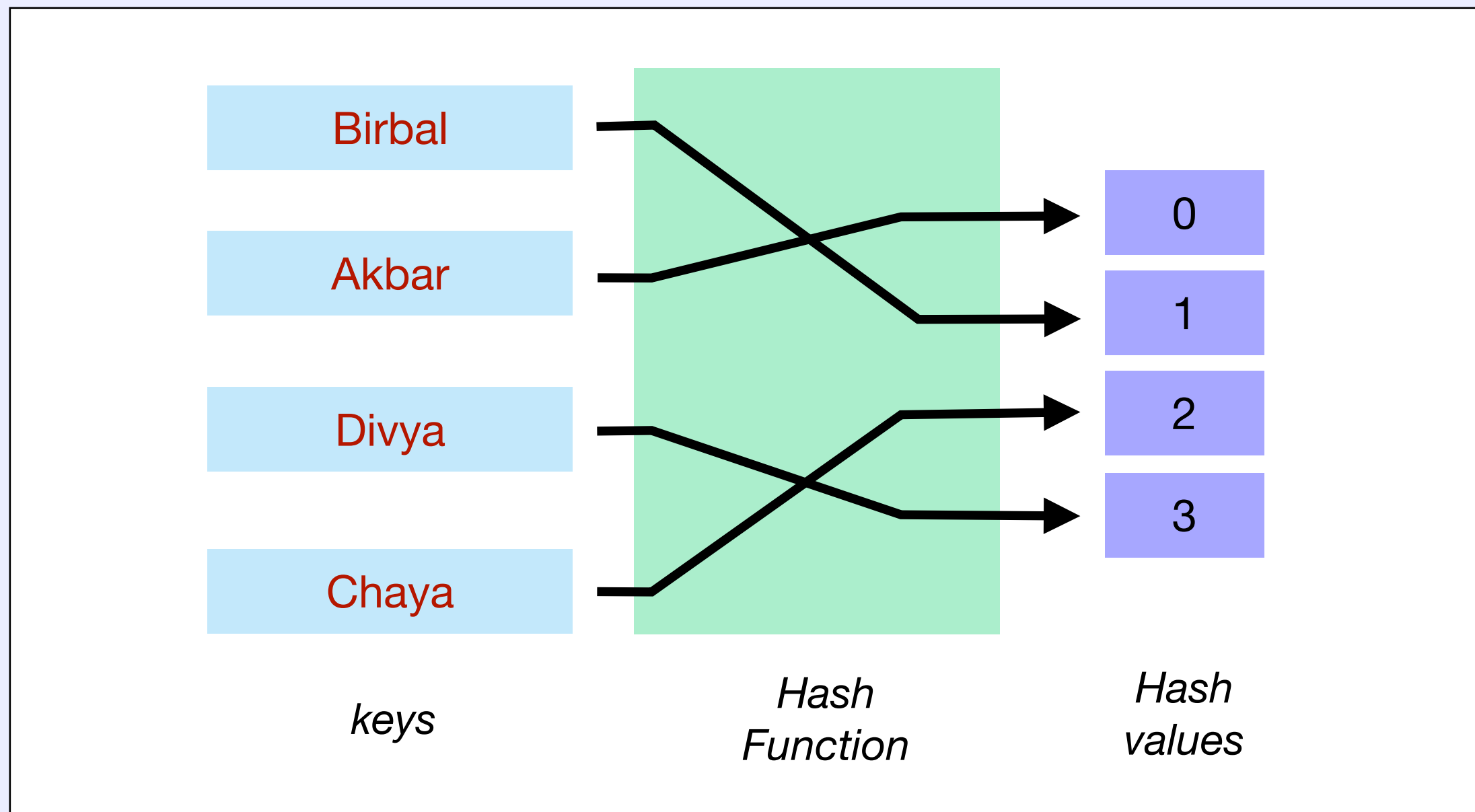
- Different data-structures to realise dictionaries:
 - Trivial (inefficient): Linked-lists, Arrays, Queues
 - **Hash Table**
 - Skip-lists
 - Binary trees
 - Red-Black Trees
 - AVL Trees
 - B-trees

Realizing Dictionary using Arrays

- If keys are integers then dictionary can store keys in an array in sorted order.
- Searching a key:
 - Takes $O(\log n)$ time
 - We can use Binary Search
- Insertion/Deletion:
 - Takes $O(n)$ time
 - We need to rebuild array from Scratch
- **Note:** This is useful when insertions/deletions are rare, but we have a lot of look-up queries.

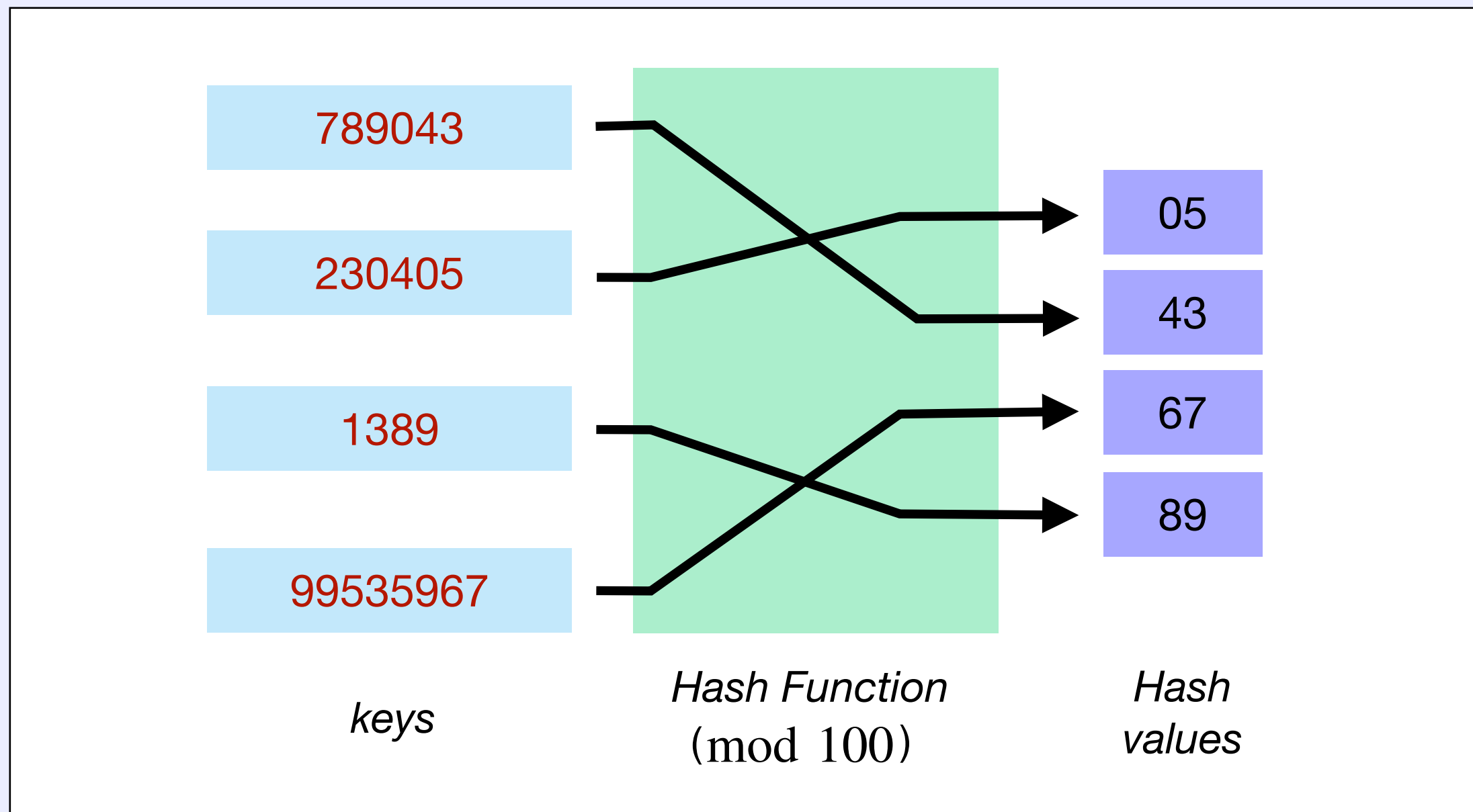
Hashing

Hashing: A process of mapping keys or strings of arbitrary size to shorter, fixed-length outputs.



Hashing

Hashing: A process of mapping keys or strings of arbitrary size to shorter, fixed-length outputs.



Hash Table

- A data-structure that uses a hash-function H to store keys and their associated elements.
- Techniques for collision resolution: chaining, secondary hash tables, etc.

23405 Akbar	21389 Birbal
78043 Chaya	95743 Divya
89943 Esha	99589 Ramesh

$n = 6$ records

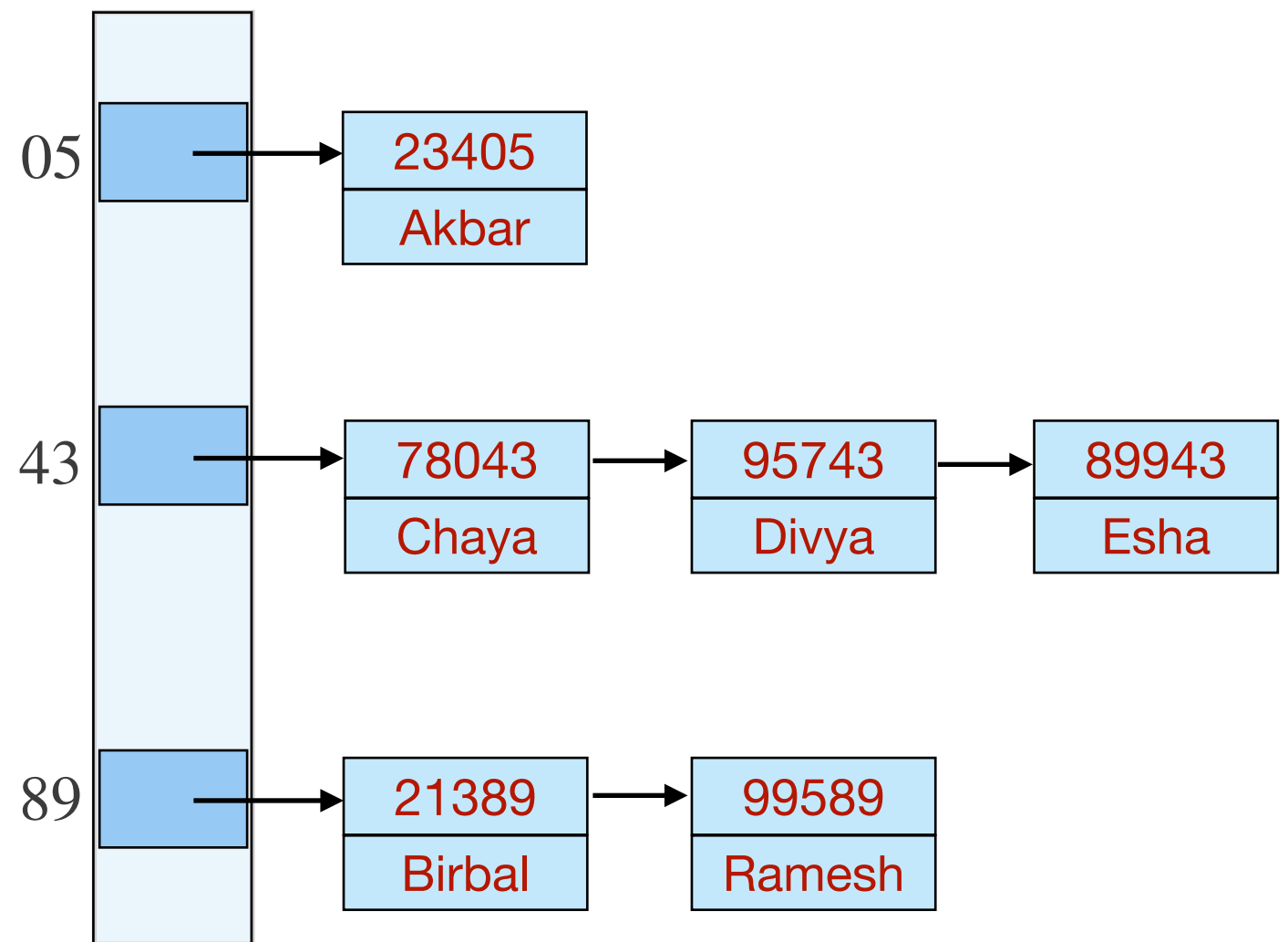


Table T

Hash Table

- A data-structure that uses a hash-function H to store keys and their associated elements.
- Techniques for collision resolution: chaining, secondary hash tables, etc.

Hash Function:

$$H(z) = (z \bmod 100)$$

List at $T[i]$ stores:

$$\{(k, e) \in U \mid H(k) = i\}$$

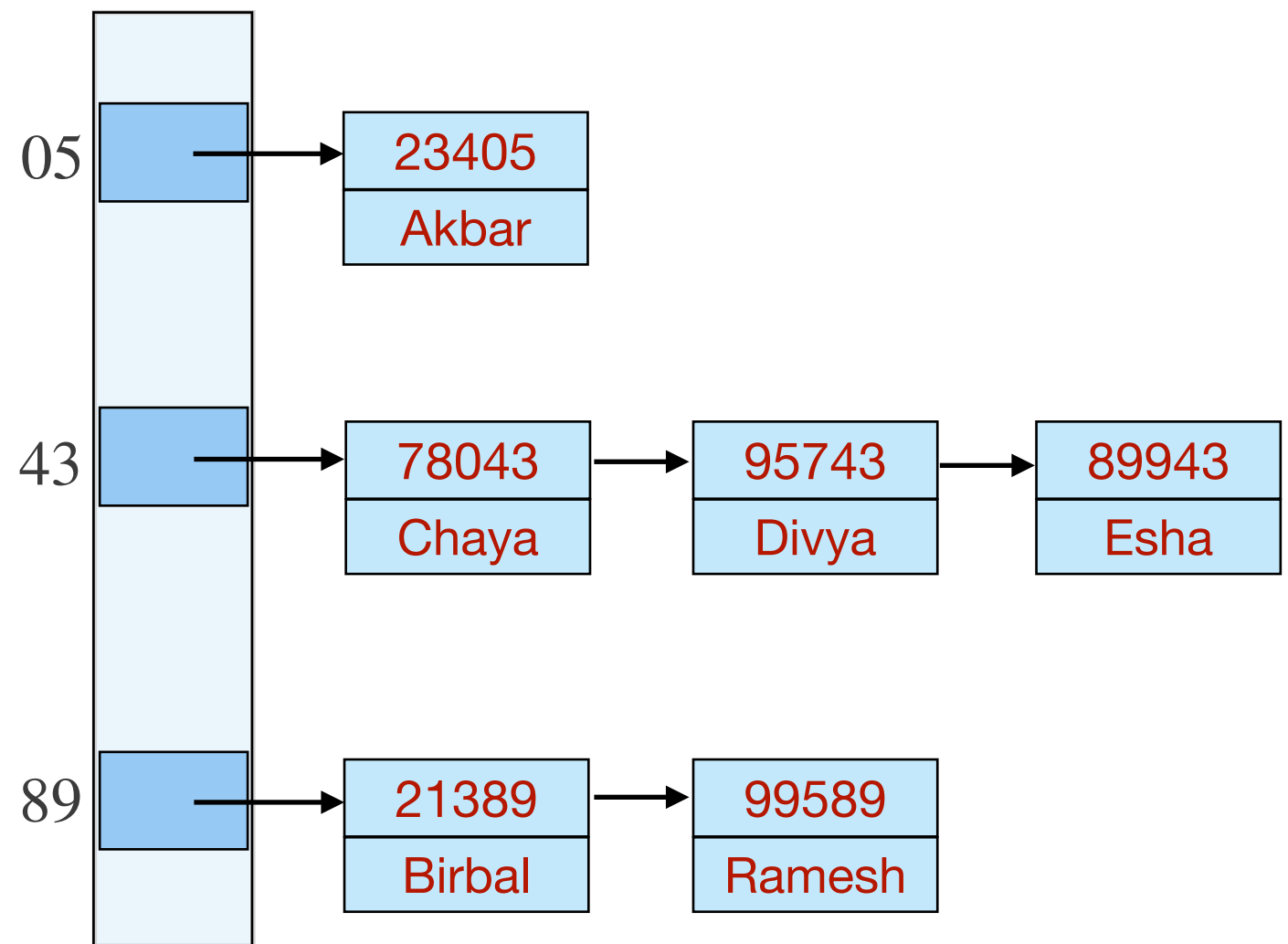


Table T

Hash Table

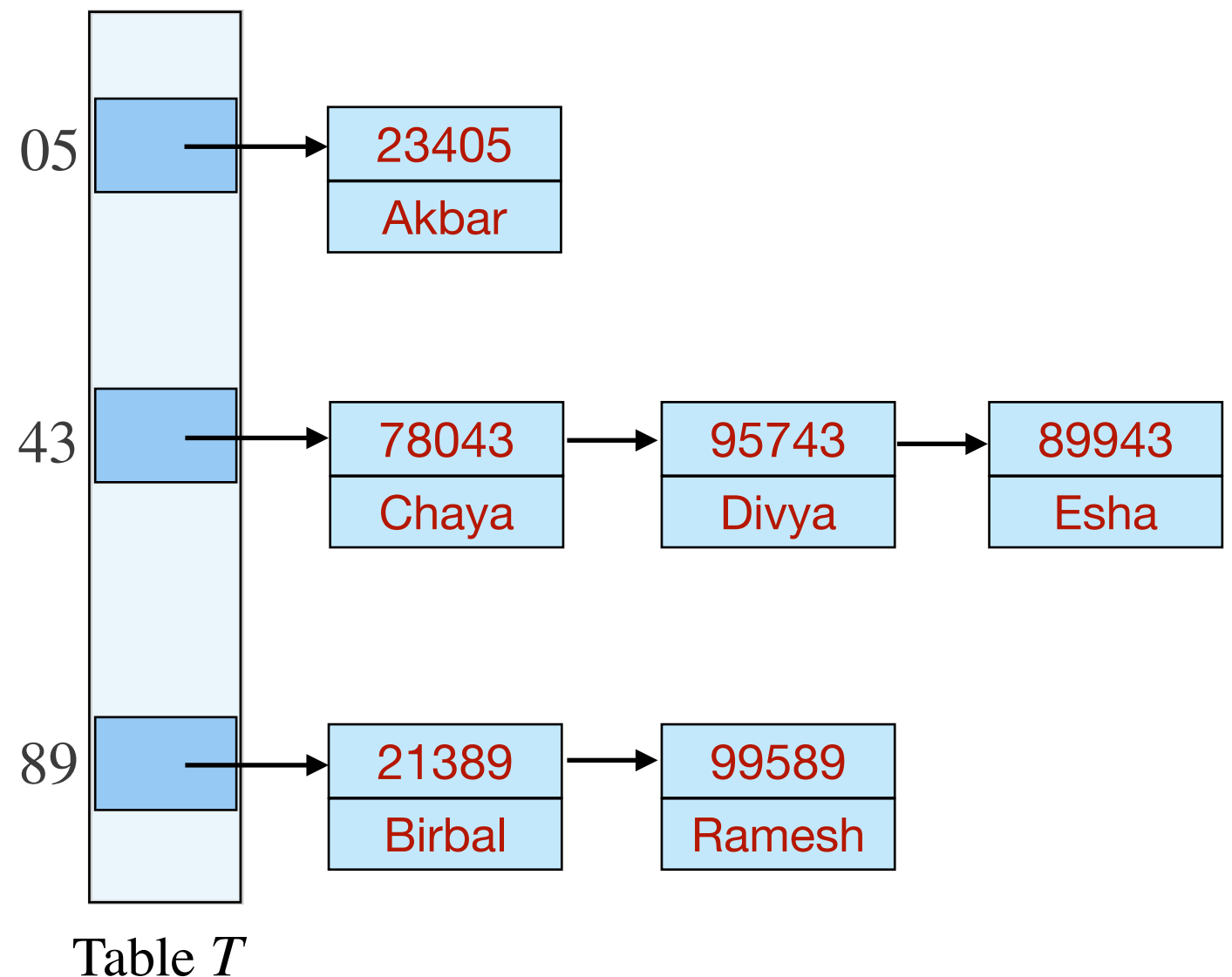
- A data-structure that uses a hash-function H to store keys and their associated elements.
- Techniques for collision resolution: chaining, secondary hash tables, etc.

Search (worst-case):

1. Time to compute Hash
2. Plus, $\max_i |T[i]|$

Space:

1. Size of table T
2. Plus, $\sum_i |T[i]|$



Properties of Hash Function

- A **good** hash function will be one which
 - Distributes keys uniformly
 - Is easy to compute
 - Takes less space
- **Question:** Is a hash function that maps each key to uniformly random value desirable?
 - No, it cannot be stored compactly.
 - Computing the hash of a key is also costly.
- **An ideal hash function:**
 - Should yield list length of $\alpha = n/m$. Here, n denotes the total number of keys, and m is the size of Table T .

A Simple Hash Function

- $H(k) = k \bmod m$
- Bad if all keys are from set $\{m, 2m, 3m, \dots\}$.
- Good if the keys are chosen **randomly**.
(Question: How to prove this formally?)

A Simple Hash Function

Hash function: $H(k) = k \bmod m$

Lemma 1: If two keys k_x, k_y are chosen uniformly randomly from universe $U = [1, L]$, then the *collision probability* (prob that $H(k_x) = H(k_y)$) is $1/m$.

Suppose $H(k_x) = i$, for some i in range $[0, m - 1]$.

Then, $H(k_y) = i$ if and only if $k_y \in \{i, m + i, 2m + i, \dots\}$.

$$\text{So, Prob}(H(k_y) = i) = \frac{|\{i, m + i, 2m + i, \dots\}|}{L} \approx \frac{1}{m}$$

Note: Two keys are said to collide if they are mapped to same value by hash function.

A Simple Hash Function

Hash function: $H(k) = k \bmod m$

Lemma 2: If keys $k_1, k_2, \dots, k_n$ are chosen uniformly randomly from universe $U = [1, L]$, then the expected size of $T[j]$, for any given $j \in [0, m - 1]$ is n/m .

Suppose $j = 5$. We will calculate $E(|T[5]|)$.

For a key k_i , define random variable $Z_i = \begin{cases} 1 & \text{if } H(k_i) = 5, \text{ and} \\ 0 & \text{otherwise.} \end{cases}$

The random variable Z_i captures contribution of key k_i in list $T[5]$.

Observe $|T[5]| = Z_1 + \dots + Z_n$.

So, $E(|T[5]|) = E(Z_1) + \dots + E(Z_n) = \text{Prob}(Z_1 = 1) + \dots + \text{Prob}(Z_n = 1) = \frac{n}{m}$

A Simple Hash Function

Hash function: $H(k) = k \bmod m$

Lemma 3: Let T be a hash table build over keys $k_1, k_2, \dots, k_n$ that are chosen uniformly randomly from universe $U = [1, L]$.

Then, the expected time to **search** any key $k \in U$ in hash table T is $O(1 + n/m)$.

- Consider a key $k \in U$.
- We first compute $H(k)$. This takes $O(1)$ time.
- Now suppose $H(k) = i$. Then we will search k in linked-list located at $T[i]$ in brute-force way.
- Time to perform this search is $O(|T[i]|)$.
- As expected value of $|T[i]|$ is $O(n/m)$, the expected time to perform search is $O(1 + n/m)$.