Hash Table Data Structure

Aniket Pingley, Ph.D.

Associative Data Structures

Key-Value pair containers

- Two parts to the data: i) the actual data a.k.a value & ii) and identifier a.k.a key
- Storage and Lookup (find operation) is based on the value
- They are Abstract Data Types
- Examples:
  - Students' records are maintained using enrollment number as key
  - Language dictionary stores meaning and usage (value) of words (key)

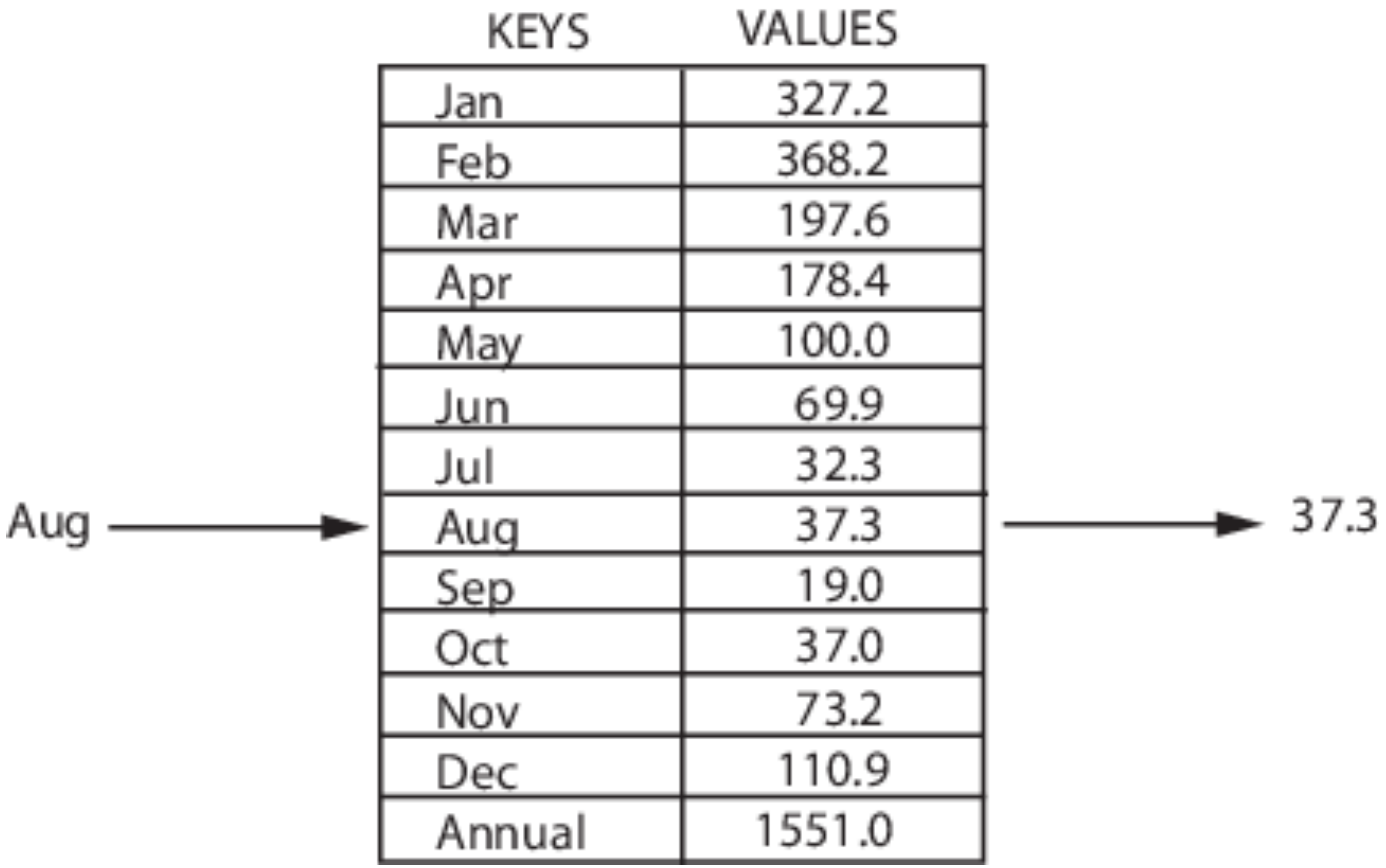
Associative Data Structures

Map container

- In this example, the data structure is of type <string, double>

- Time complexity of Lookup operations can be improved to O(log N) if the keys are sorted

- Insert operation has the same time complexity as the lookup operation



KEYS	VALUES
Jan	327.2
Feb	368.2
Mar	197.6
Apr	178.4
May	100.0
Jun	69.9
Jul	32.3
Aug	37.3
Sep	19.0
Oct	37.0
Nov	73.2
Dec	110.9
Annual	1551.0

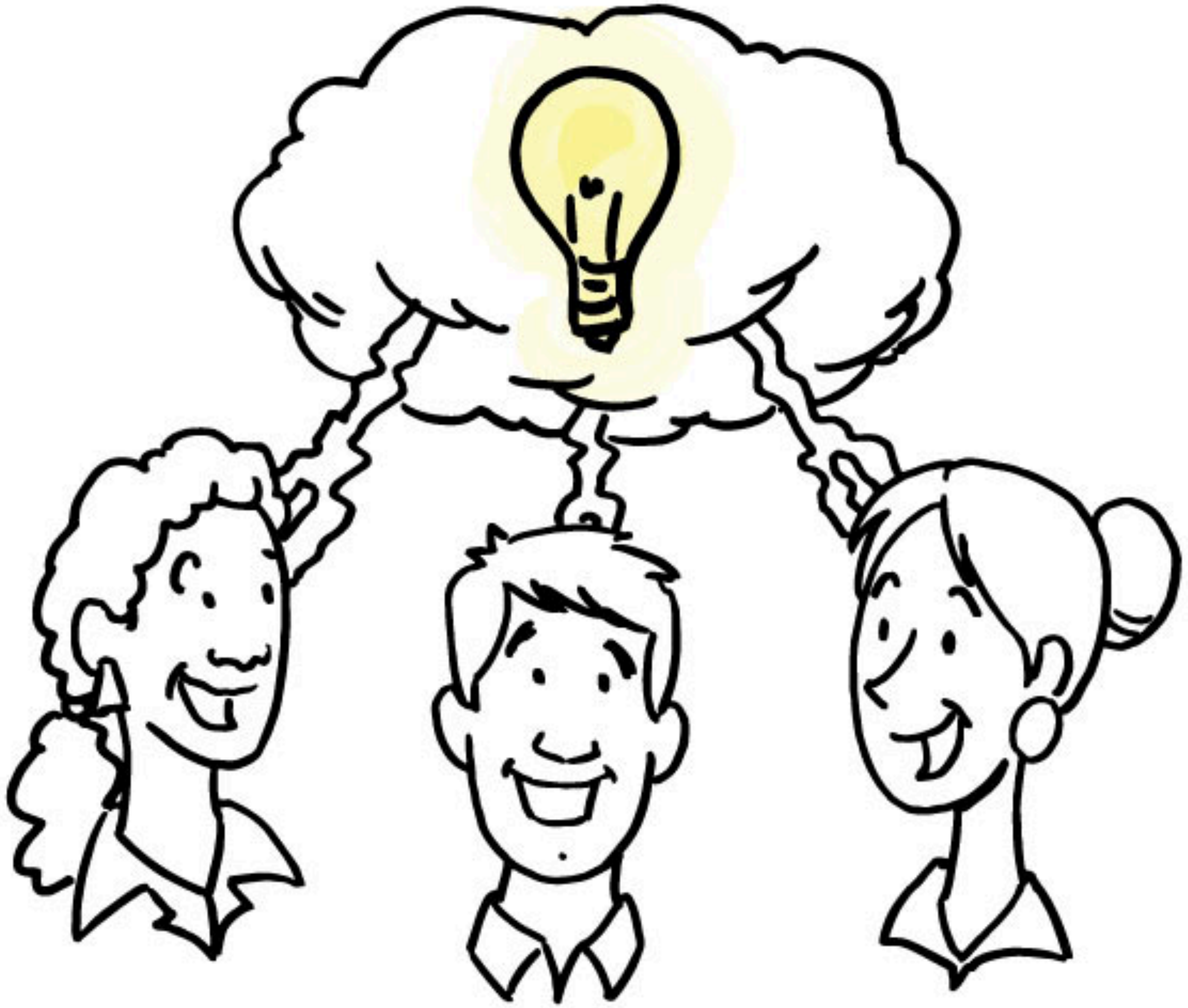
Associative Data Structures

Brainstorming a common scenario

- Brainstorm the logic for reading from the user the name of a text file, counts the word frequencies of all words in the file, and outputs a list of words and their frequency.

- What will be your <key, value> pairs in this context?

- How can you make Associative DS better?



Associative Data Structures

What if ...

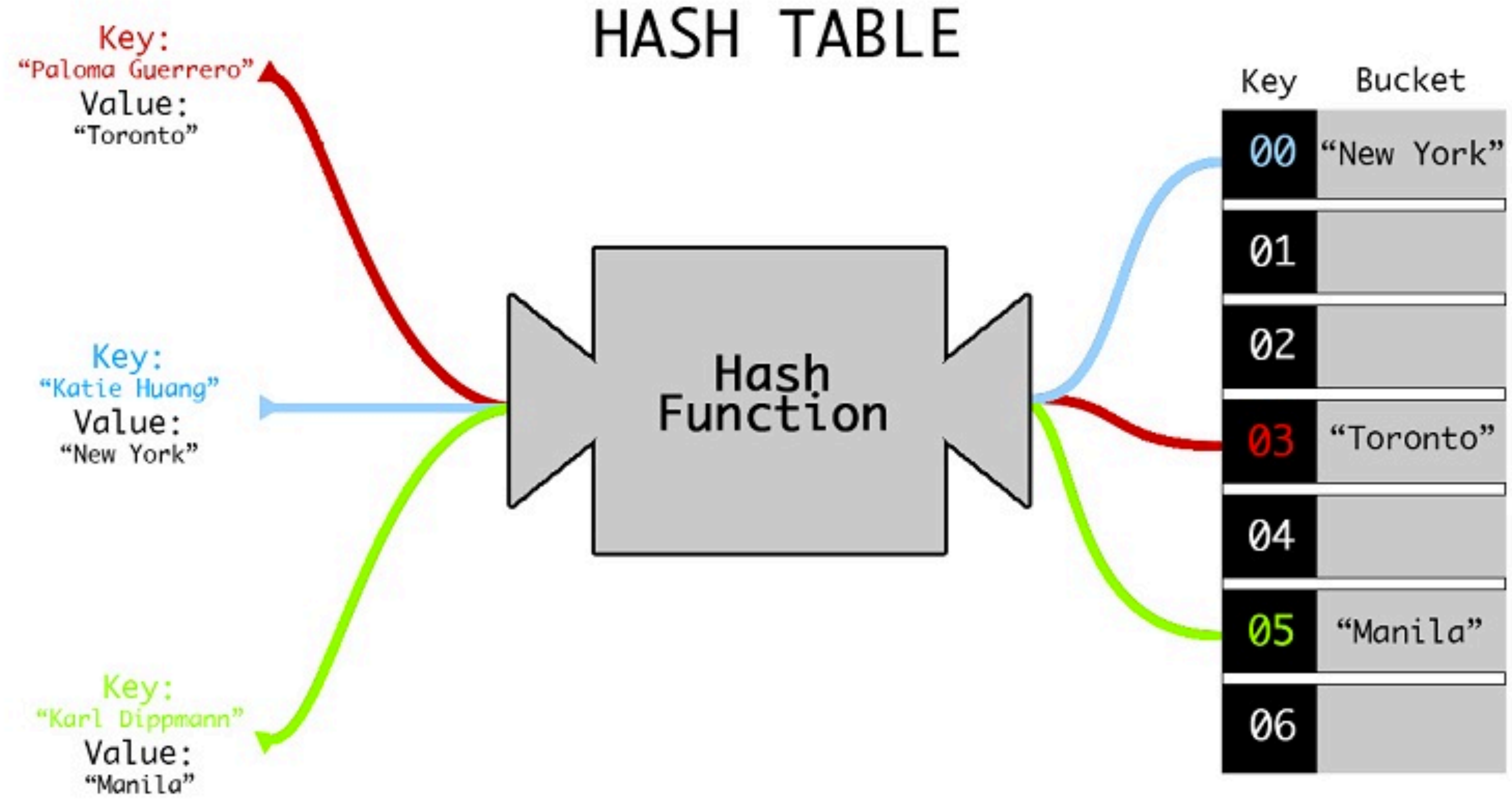
Data Structures ko bhaad me daalne ki Time Complexity kya hogi Bro?

Dukhi-Aatma Vaanar

- What if the key is more complex than a string or a number?
- What if the key is not sortable?
- What if the key itself is too long?
- Is there a need for a separate storage for keys?
- Can the storage and lookup time complexity be improved?

Hash Table

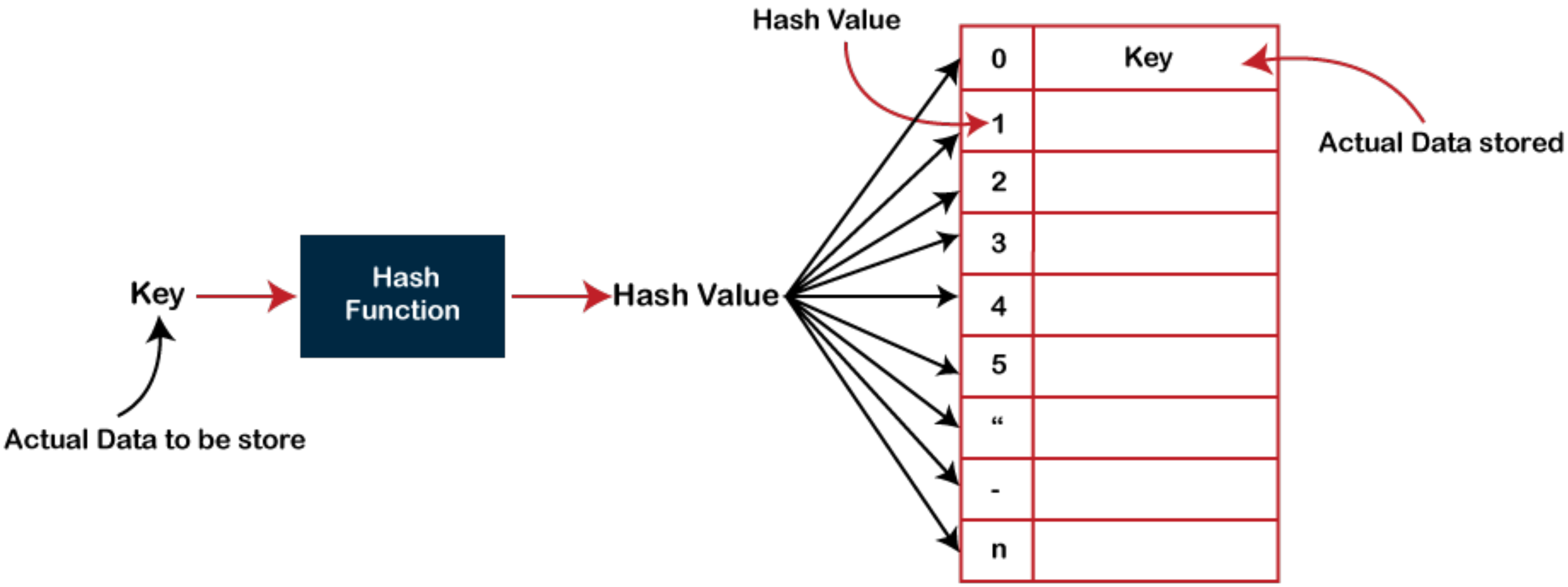
Associative Array ADT



Key	Bucket
00	“New York”
01	
02	
03	“Toronto”
04	
05	“Manila”
06	

Hash Table

Associative Array ADT; Arbitrary blob of data



Hash Function

What is it?

- Map an arbitrary data of an arbitrary size to a data of a fixed size, which can interpreted in numeric value

- Must be easy to implement, i.e. time complexity must be of higher orders

- Output should have nearly uniform distribution

- (Ideally) no two arbitrary inputs should result in the same output, i.e. no collisions

- Finding the index: i) hashVal = hashFn(key); and ii) index = hashVal % arraySz;

Hash Functions

Some well known ones …

- Division Hashing: hash = key % size; the simplest one
- Multiplication Hashing: hash = floor(size ((key * Z) % 1)), where Z is a fractional value between 0 and 1.
- Mid-square Hashing: Some seed value is selected as key. That seed a squared and values in the middle are extracted. This process is repeated for pre-decided number of steps. The output is the hash value.
- Folding Hash Function: key is divided into equal-sized pieces; value of each piece is added and the modulo operation is performed. For example, key = 369475, then hash = (36+94+75) % size

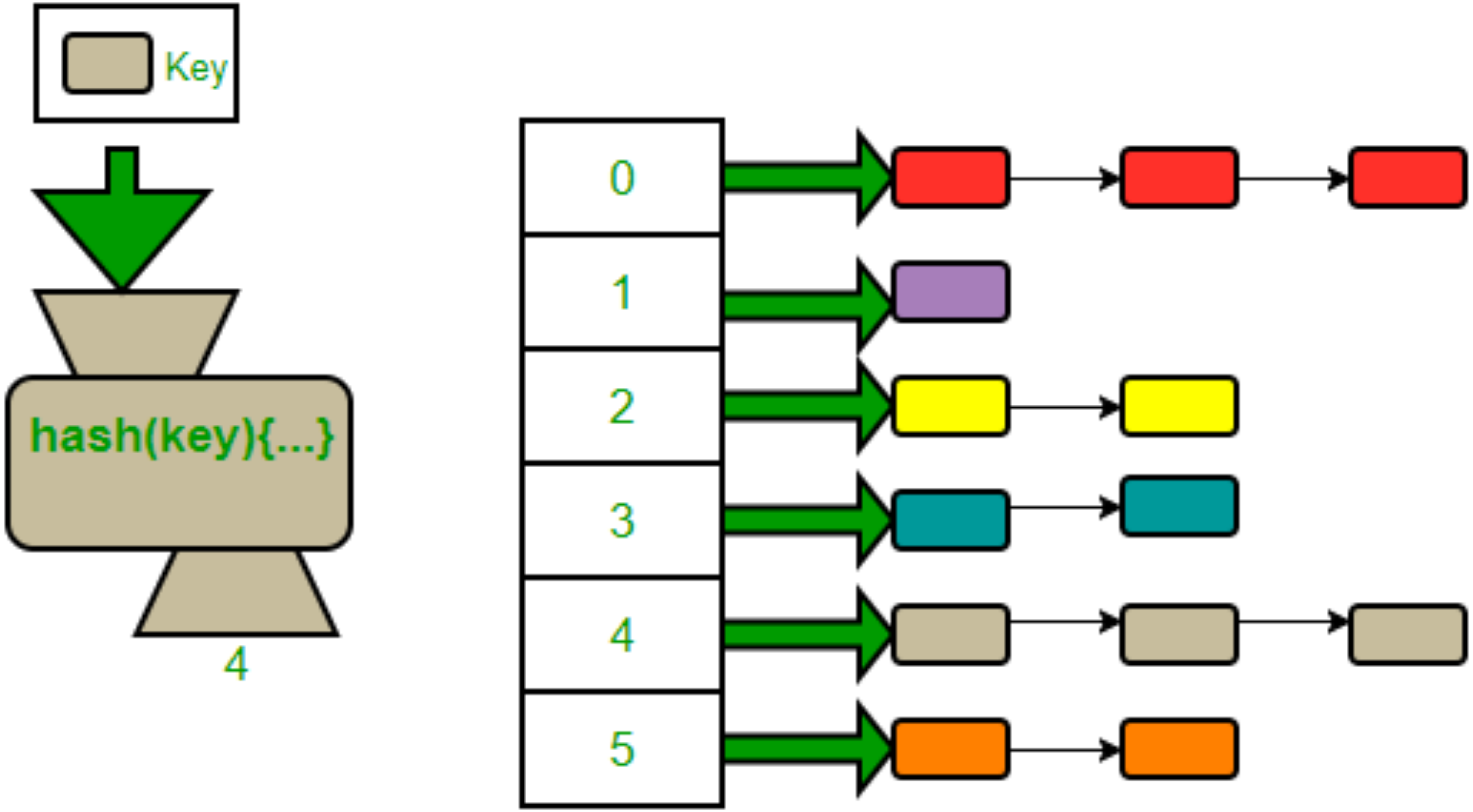
What is collision in hashing

An overview

- Due to limitation of size of buckets, hash function results in much small numerical value for a key which could be a big integer or string.
- Thus, the possibility that two keys result in the same hashed value increases.
- Collision = Mapping of the hashed value of a key already occupied bucket in the hash table.
- It must be handled using some collision handling technique.

Collisions in Hashing

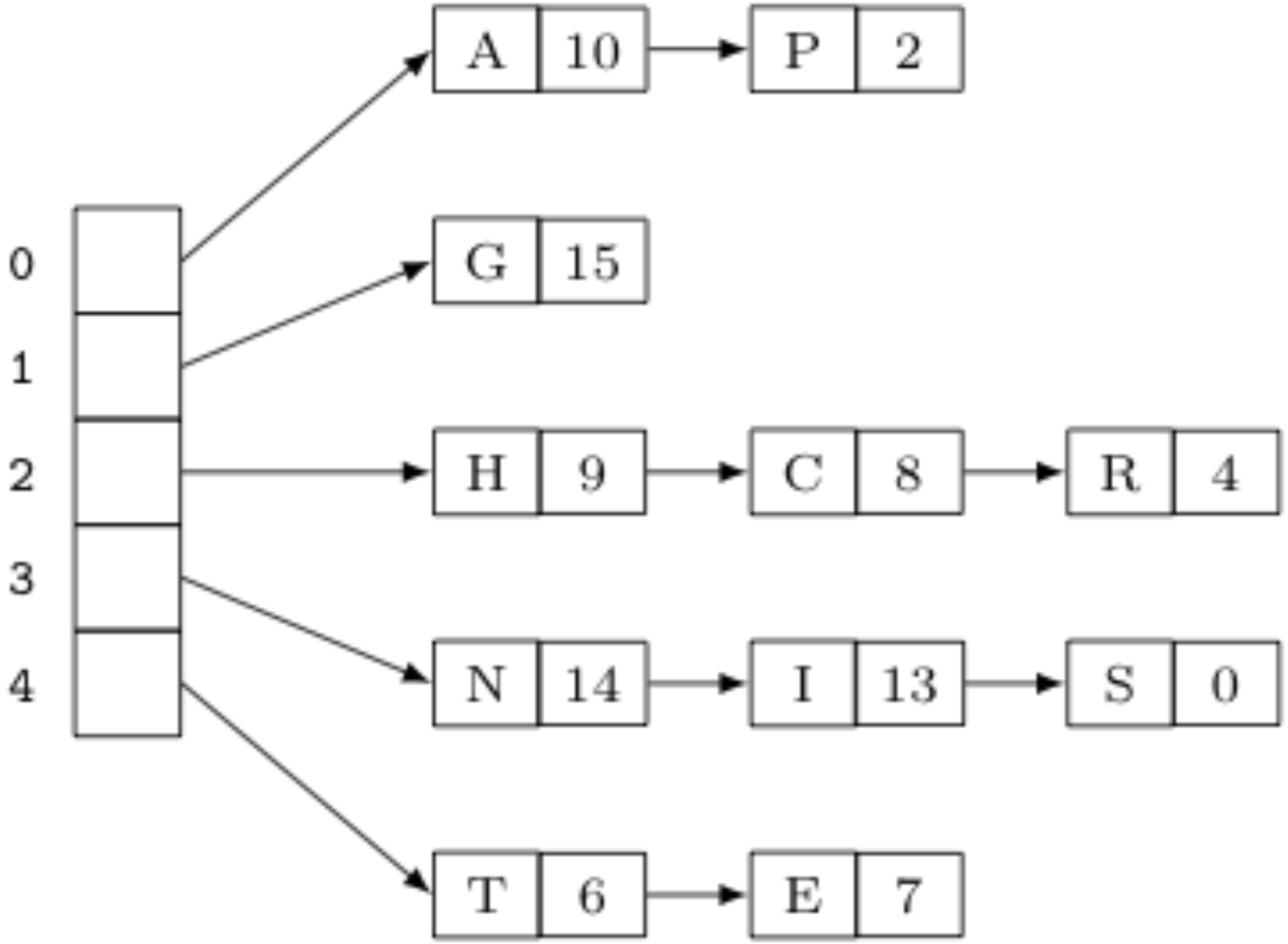
Separate Chaining



Collisions in Hashing

Separate Chaining

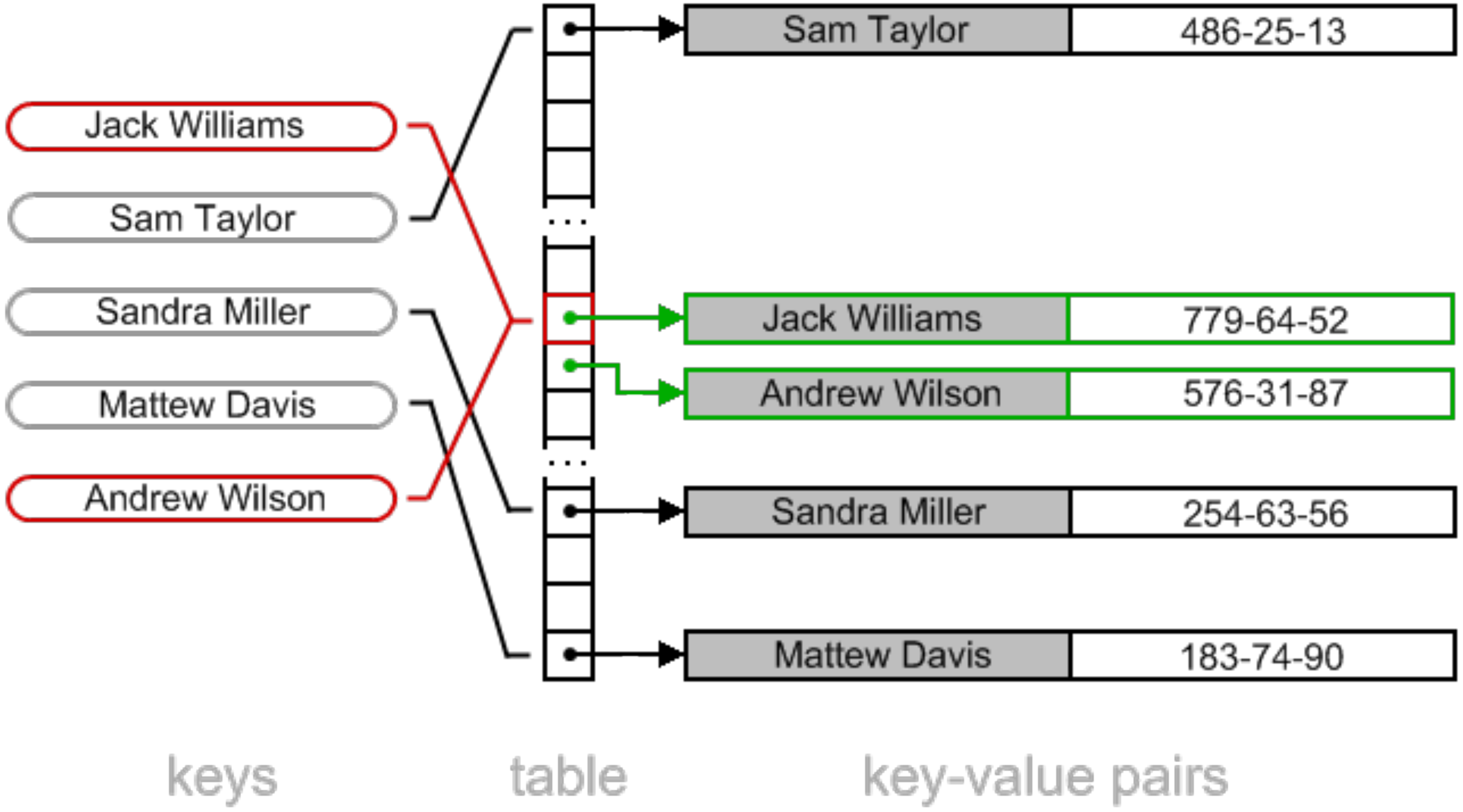
key	slot	value
S	3	0
E	4	1
P	0	2
A	0	3
R	2	4
A	0	5
T	4	6
E	4	7
C	2	8
H	2	9
A	0	10
I	3	11
N	3	12
I	3	13
N	3	14
G	1	15



Collisions in Hashing

Open Addressing

- Linear Probing: look for next empty slot
- Quadratic Probing: look for k^2 slot for k^{th} input
- Operations:
    - Insertion: probe until empty slot is found
    - Lookup: probe until the input key compares or empty slot is found
    - Delete: need a mechanism to demarcate deleted entry versus empty slot



Jack Williams
Sam Taylor
Sandra Miller
Mattew Davis
Andrew Wilson
Sam Taylor
486-25-13
Jack Williams
779-64-52
Andrew Wilson
576-31-87
Sandra Miller
254-63-56
Mattew Davis
183-74-90
keys
table
key-value pairs