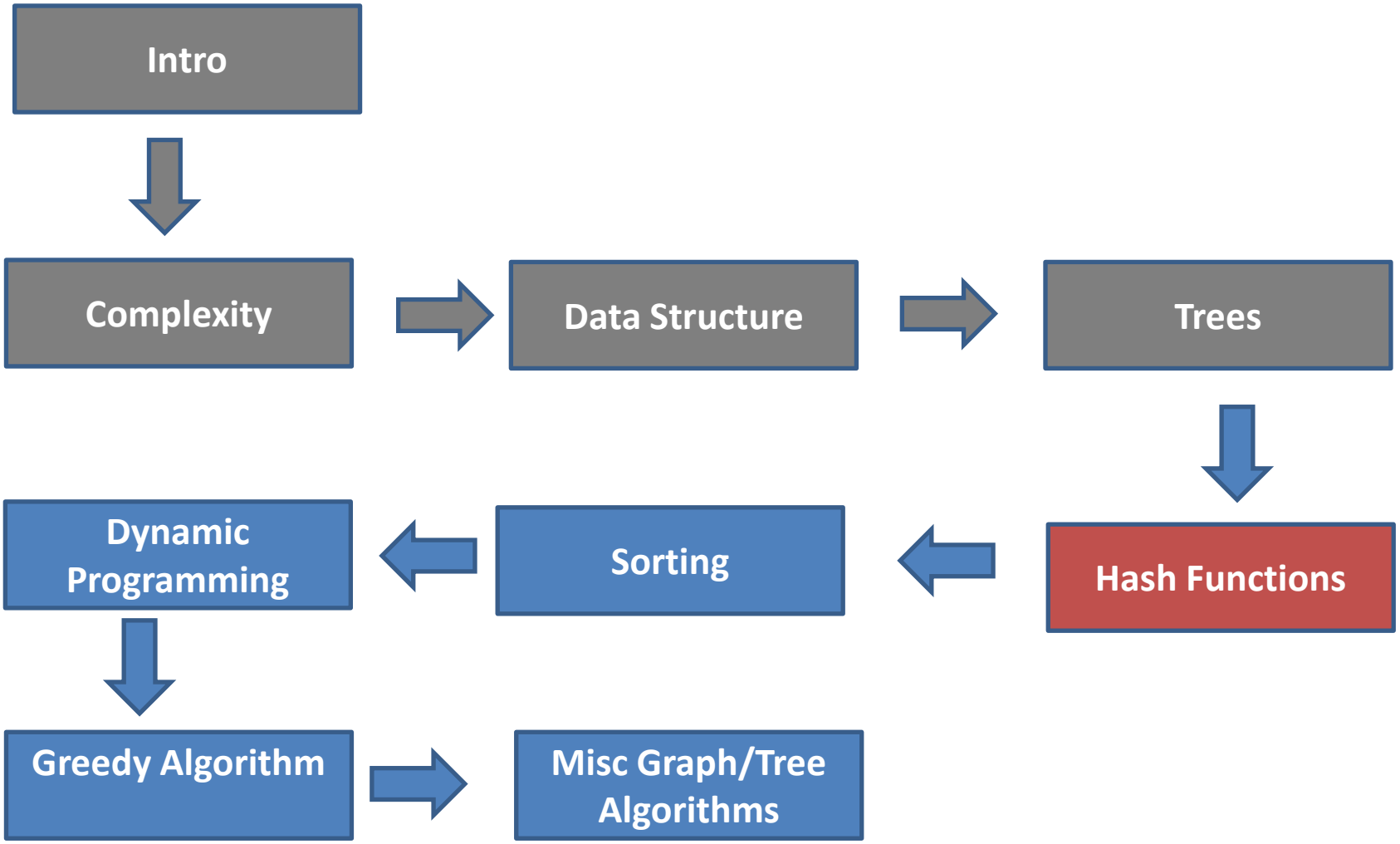
An Introduction to Algorithms

By

Hossein Rahmani

h_rahmani@iust.ac.ir
http://webpages.iust.ac.ir/h_rahmani/


Intro
Complexity
Data Structure
Trees
Hash Functions
Sorting
Dynamic Programming
Greedy Algorithm
Misc Graph/Tree Algorithms

Motivation

- Arrays provide an indirect way to access a **set**.
- Many times we need an association between two sets, or a set of **keys** and associated data.
- Ideally we would like to access this data directly with the keys.
- We would like a data structure that supports fast search, insertion, and deletion.
  - Do not usually care about sorting.
- The abstract data type is usually called a **Dictionary, Map** or **Partial Map**
  - **float googleStockPrice = stocks[“Goog”].CurrentPrice;**

Dictionaries

- What is the best way to implement this?
  - Linked Lists?
  - Double Linked Lists?
  - Queues?
  - Stacks?
  - Multiple indexed arrays (e.g., data[key[i]])?
- To answer this, ask what the complexity of the operations are:
  - Insertion
  - Deletion
  - Search

Direct Addressing

- Let's look at an easy case, suppose:
  - The range of keys is 0..*m*-1
  - Keys are distinct
- Possible solution
  - Set up an array T[0..m-1] in which
    - T[*i*] = *x* if $x \in T$ and key[*x*] = *i*
    - T[*i*] = NULL otherwise
  - This is called a *direct-address table*
    - Operations take O(1) time!
    - *So what's the problem?*

Direct Addressing

- Direct addressing works well when the range *m* of keys is relatively small
- But what if the keys are 32-bit integers?
  - Problem 1: direct-address table will have 2^{32} entries, more than 4 billion
  - Problem 2: even if memory is not an issue, the time to initialize the elements to NULL may be
- Solution: map keys to smaller range 0..*p*-1
  - Desire $p = \boldsymbol{O}(m)$.

Hash Table

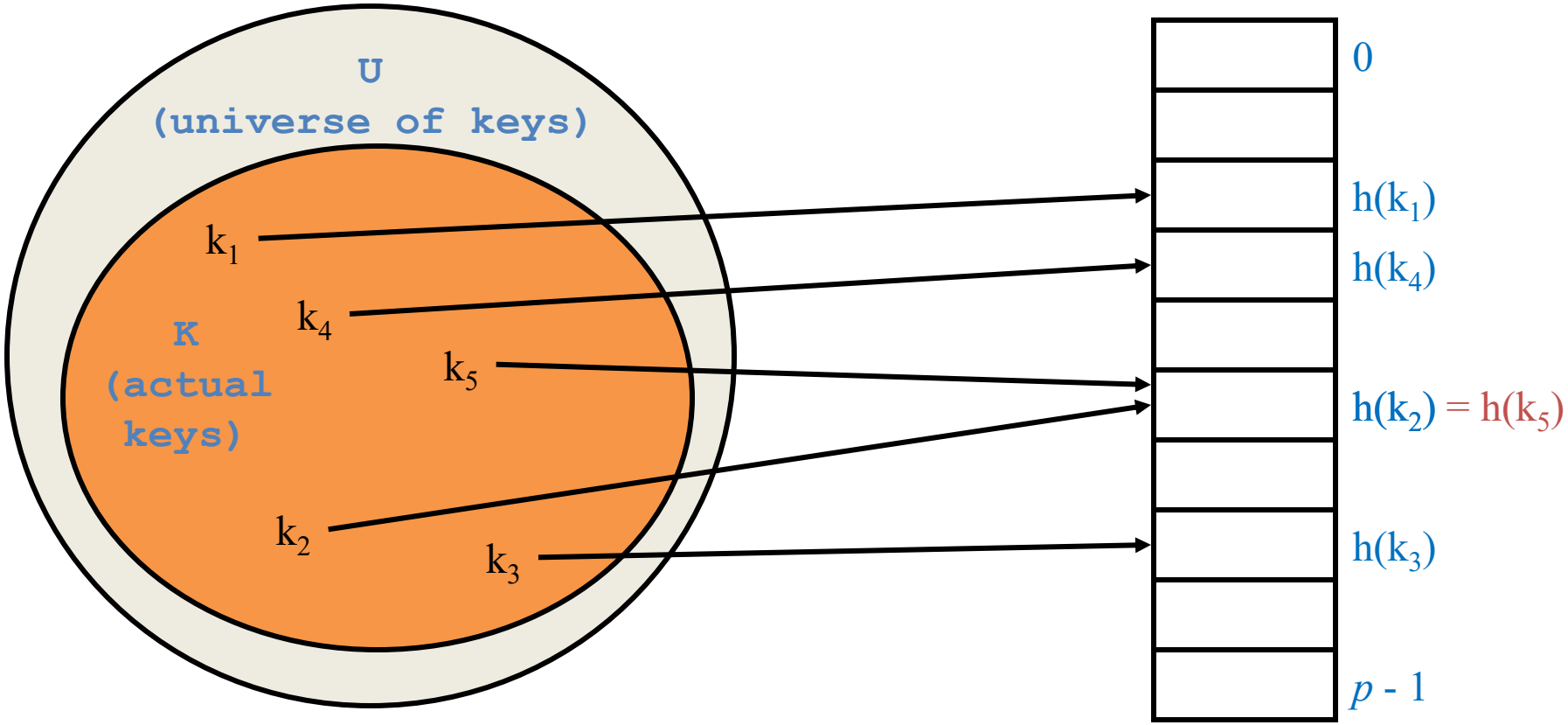
- Hash Tables provide ***O***(1) support for all of these operations!
- The key is rather than index an array directly, index it through some function, ***h***(*x*), called a *hash function*.
  - myArray[*h*(index)]
- Key questions:
  - What is the set that the *x* comes from?
  - What is *h()* and what is its range?

Hash Table

- Consider this problem:
  - If I know a priori the *p* keys from some finite set **U**, is it possible to develop a function *h(x)* that will uniquely map the *p* keys onto the set of numbers 0..*p*-1?

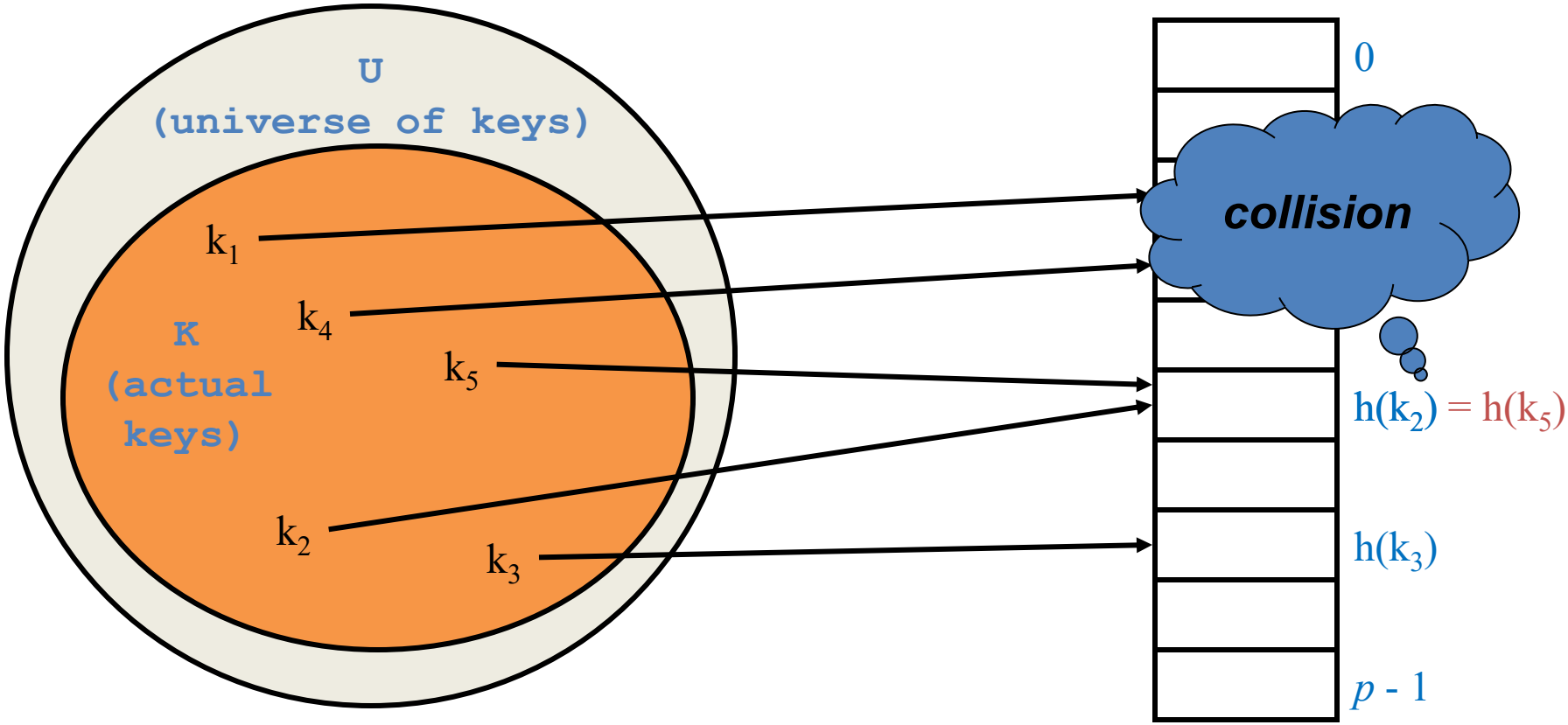
Hash Functions

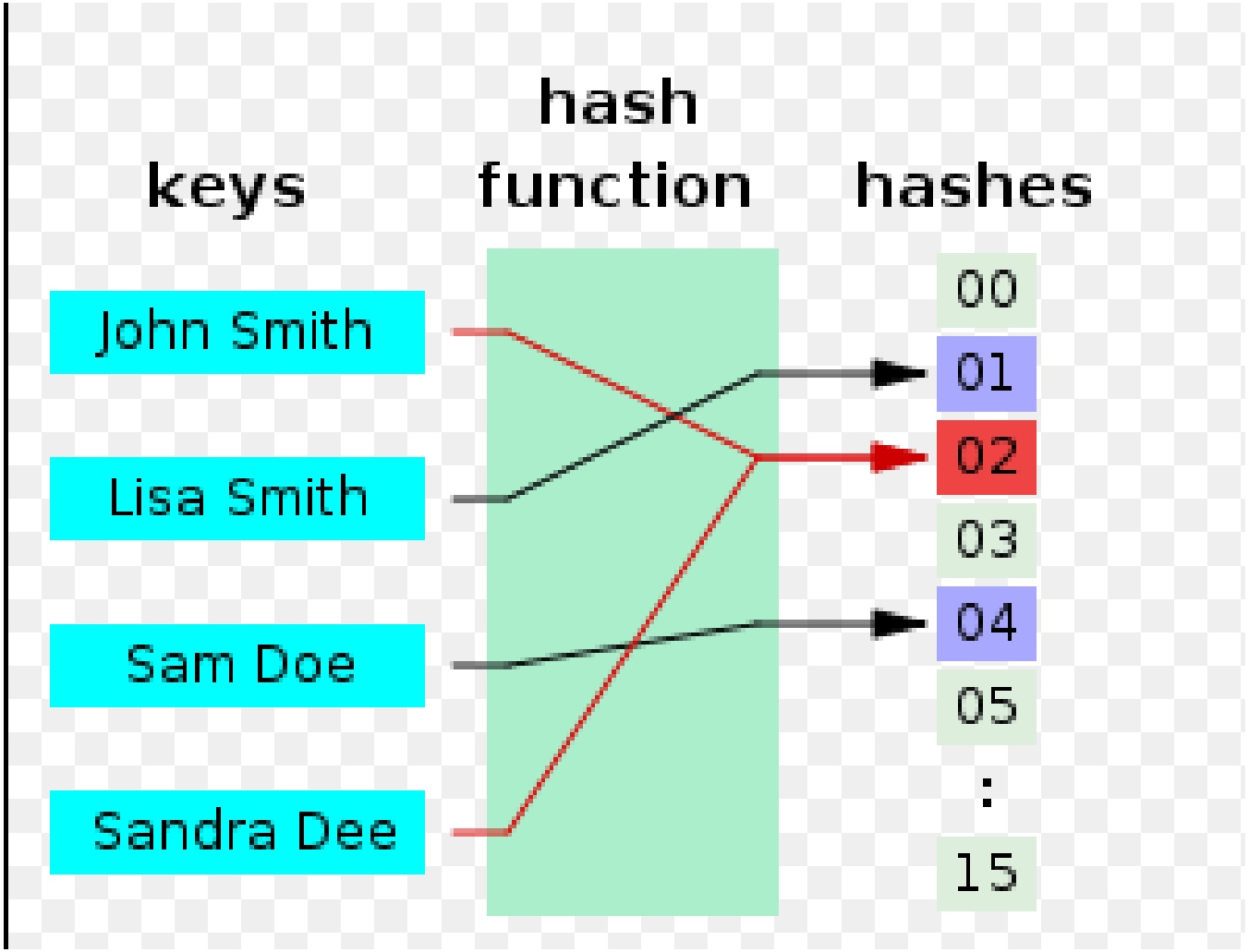
- In general a difficult problem. Try something simpler.



Hash Functions

- A **collision** occurs when *h(x)* maps two keys to the same location.




keys
hash function
hashes
John Smith
Lisa Smith
Sam Doe
Sandra Dee
00
01
02
03
04
05
:
15

Hash Functions

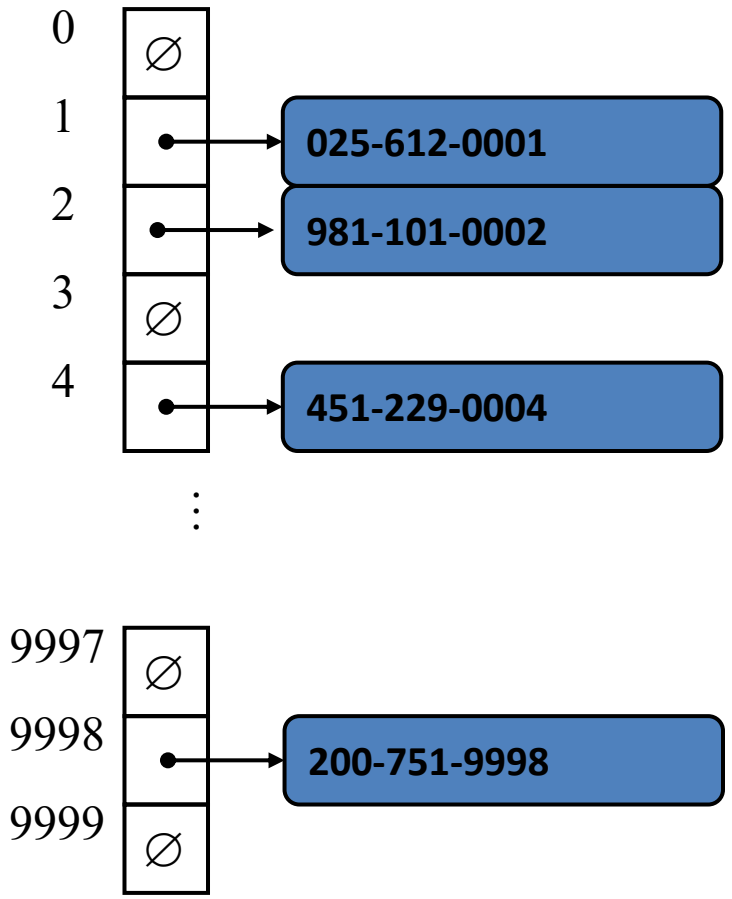
- A hash function, h, maps keys of a given type to integers in a fixed interval $[0, N-1]$
- Example:

  $$h(x) = x \bmod N$$

  is a hash function for integer keys
- The integer $h(x)$ is called the hash value of x.
- A hash table for a given key type consists of
  - Hash function h
  - Array (called table) of size N
- **The goal is to store item (k, o) at index $i = h(k)$**

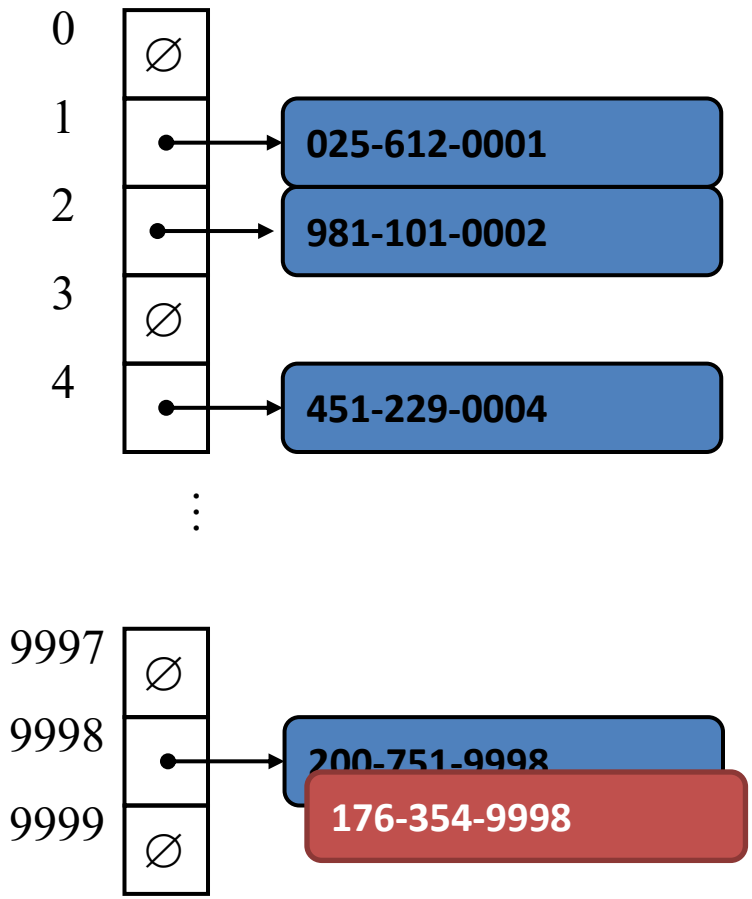
Example

- We design a hash table storing employees records using their social security number, SSN as the key.
  - SSN is a nine-digit positive integer
- Our hash table uses an array of size $N = 10{,}000$ and the hash function
  $h(x) = \text{last four digits of } x$



Example

- Our hash table uses an *array* of size **N = 100.**
- We have ***n* = 49** employees.
  - Need a method to handle ***collisions***.
- As long as the chance for collision is low, we can achieve this goal.
- Setting **N = 1000** and looking at the last four digits will *reduce* the chance of collision.

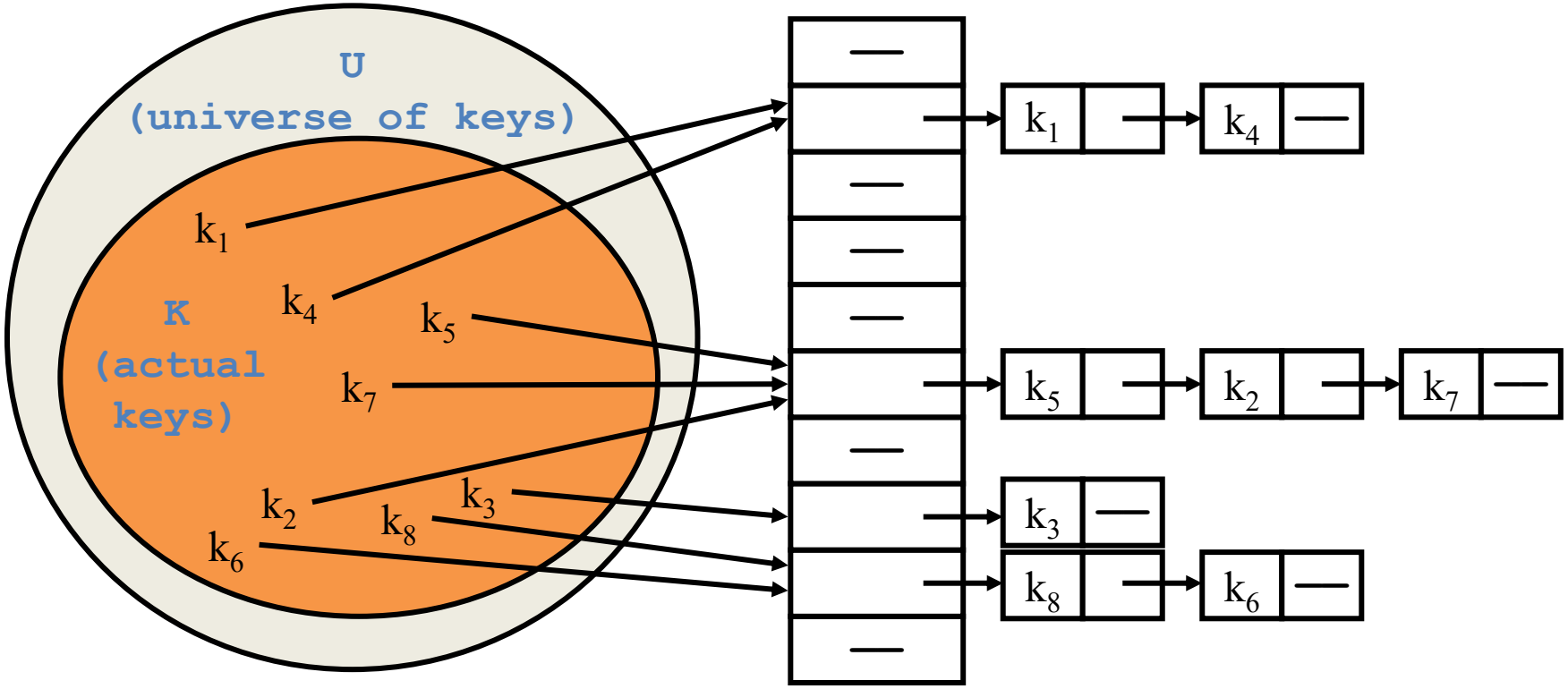


Collisions

- Can collisions be avoided?
  - If my data is immutable, yes
    - See *perfect hashing* for the case were the set of keys is static (not covered).
  - In general, no.
- Two primary techniques for resolving collisions:
  - **Chaining** – keep a collection at each key slot.
  - **Open addressing** – if the current slot is full use the ***next open*** one.

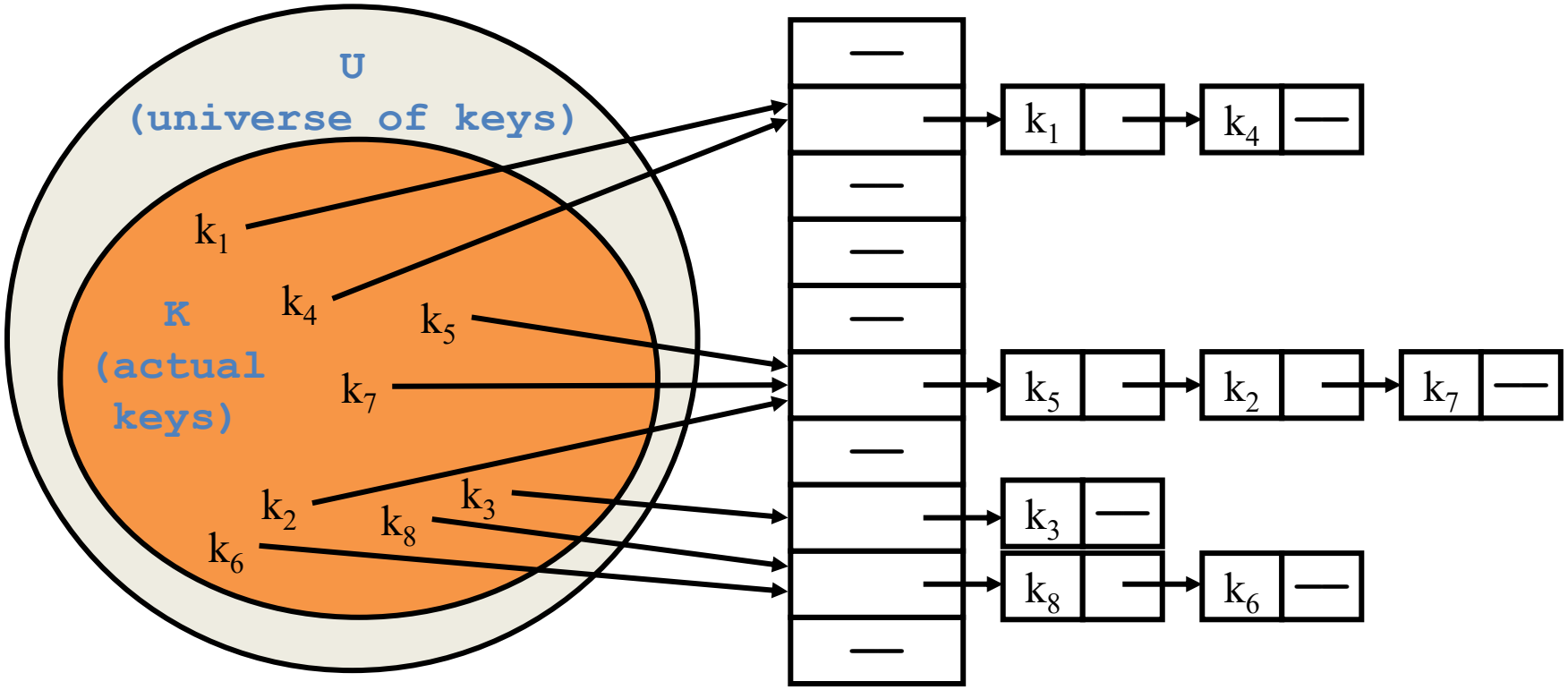
Chaining

- Chaining puts elements that hash to the same slot in a linked list:



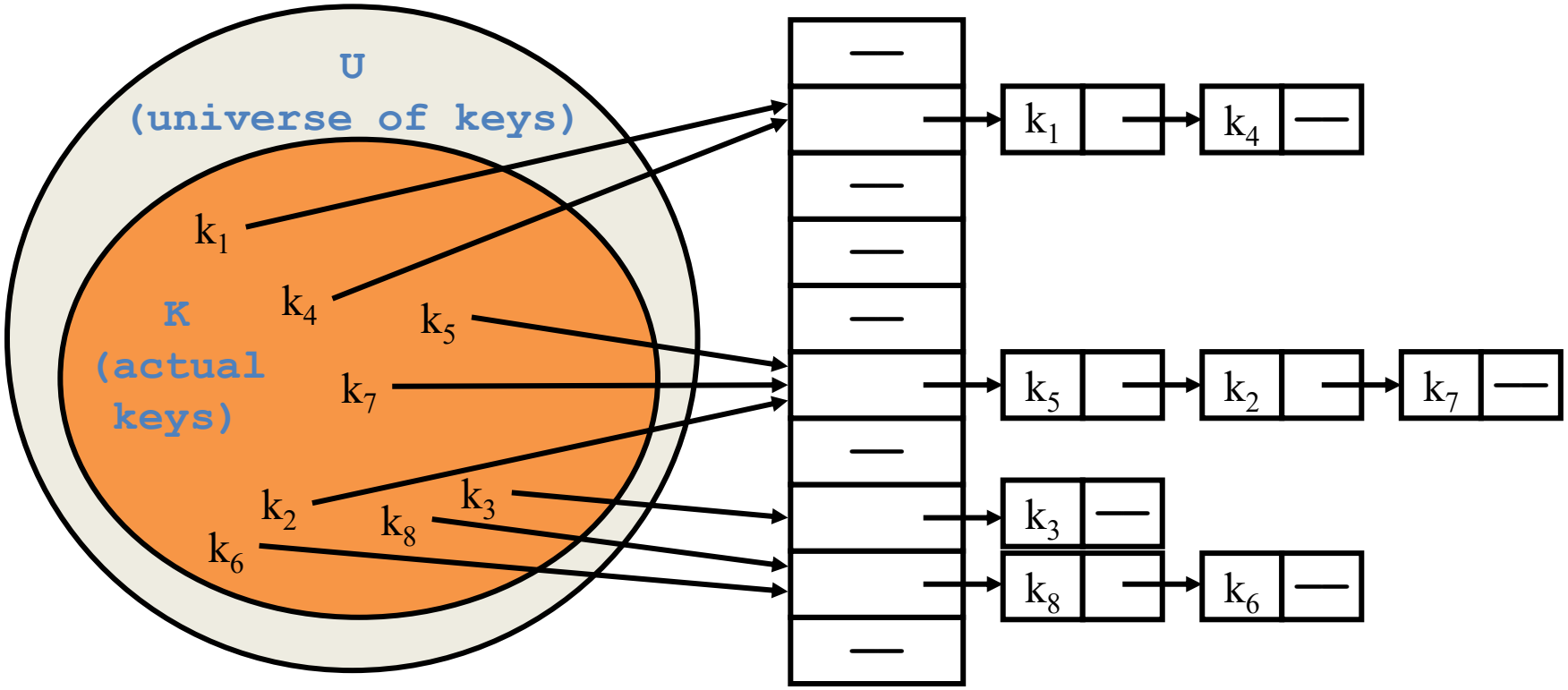
Chaining

- *How do we <u>insert</u> an element?*



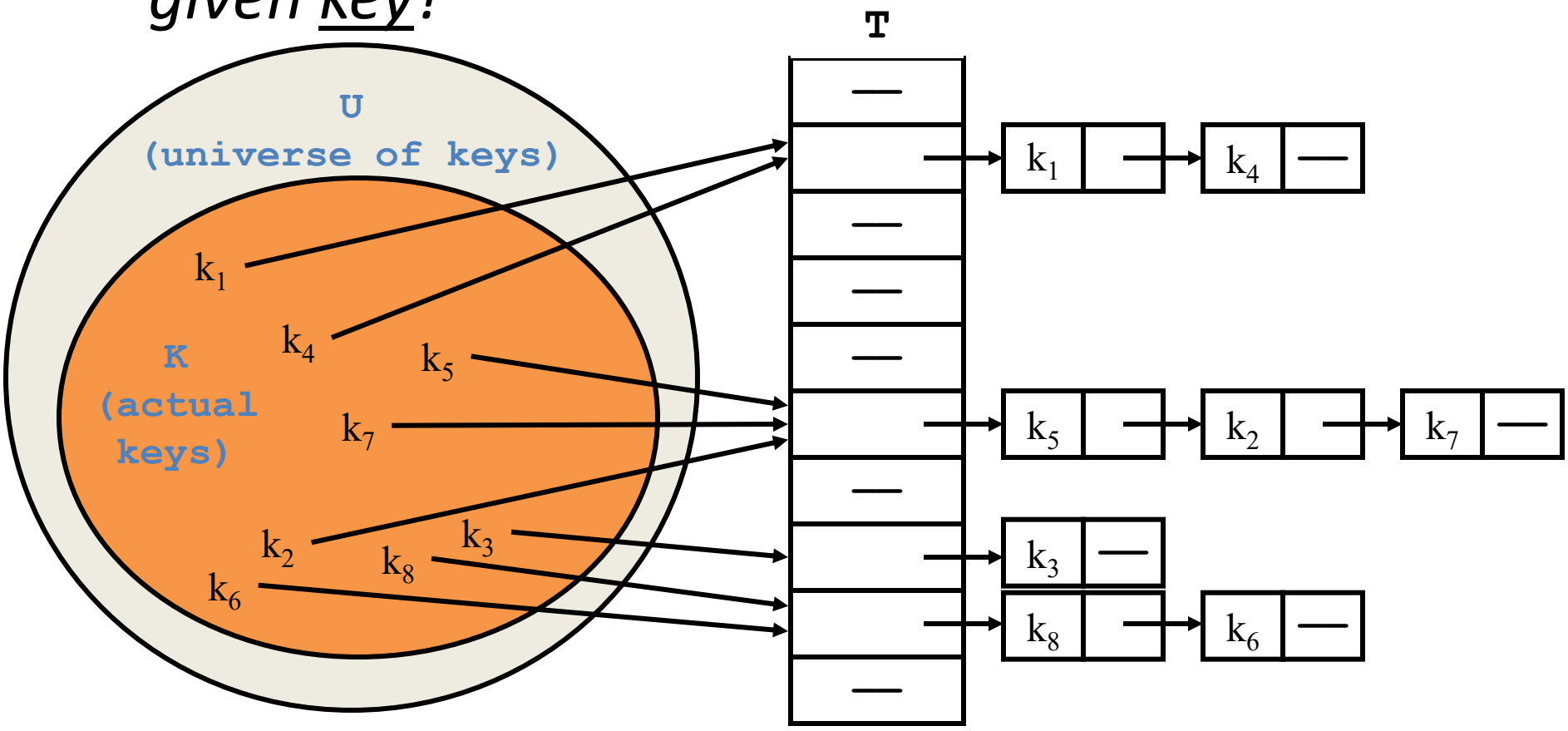
Chaining

- *How do we <u>delete</u> an element?*



Chaining

- *How do we search for a element with a given key?*



Quiz 1

- Given the following input (4322, 1334, 1471, 9679, 1989, 6171, 6173, 4199) and the hash function x mod 10, which of the following statements are true? (i.) 9679, 1989, 4199 hash to the same value (ii.) 1471, 6171 has to the same value (iii.) All elements hash to the same value (iv.) Each element hashes to a different value

A	i only
B	ii only
C	i and ii only
D	iii or iv

Quiz 2

- Consider a hash table with 100 slots. Collisions are resolved using chaining. Assuming simple uniform hashing, what is the probability that the first 3 slots are unfilled after the first 3 insertions?

A	$(97 \times 97 \times 97)/100^3$
B	$(99 \times 98 \times 97)/100^3$
C	$(97 \times 96 \times 95)/100^3$
D	$(97 \times 96 \times 95)/(3! \times 100^3)$

Quiz 3

- Which one of the following hash functions on integers will distribute keys most uniformly over 10 buckets numbered 0 to 9 for i ranging from 0 to 2020?

A	$h(i) = i^2 \bmod 10$
B	$h(i) = i^3 \bmod 10$
C	$h(i) = (11 * i^2) \bmod 10$
D	$h(i) = (12 * i) \bmod 10$

Quiz 4

- Consider a hash function that distributes keys uniformly. The hash table size is 20. After hashing of how many keys will the probability that any new key hashed collides with an existing one exceed 0.5?

Quiz 5

- Given an initially empty hash table with capacity 13 and hash function H (x) = x % 13, insert these values in the order given: 5, 17, 4, 22, 31, 43, 44