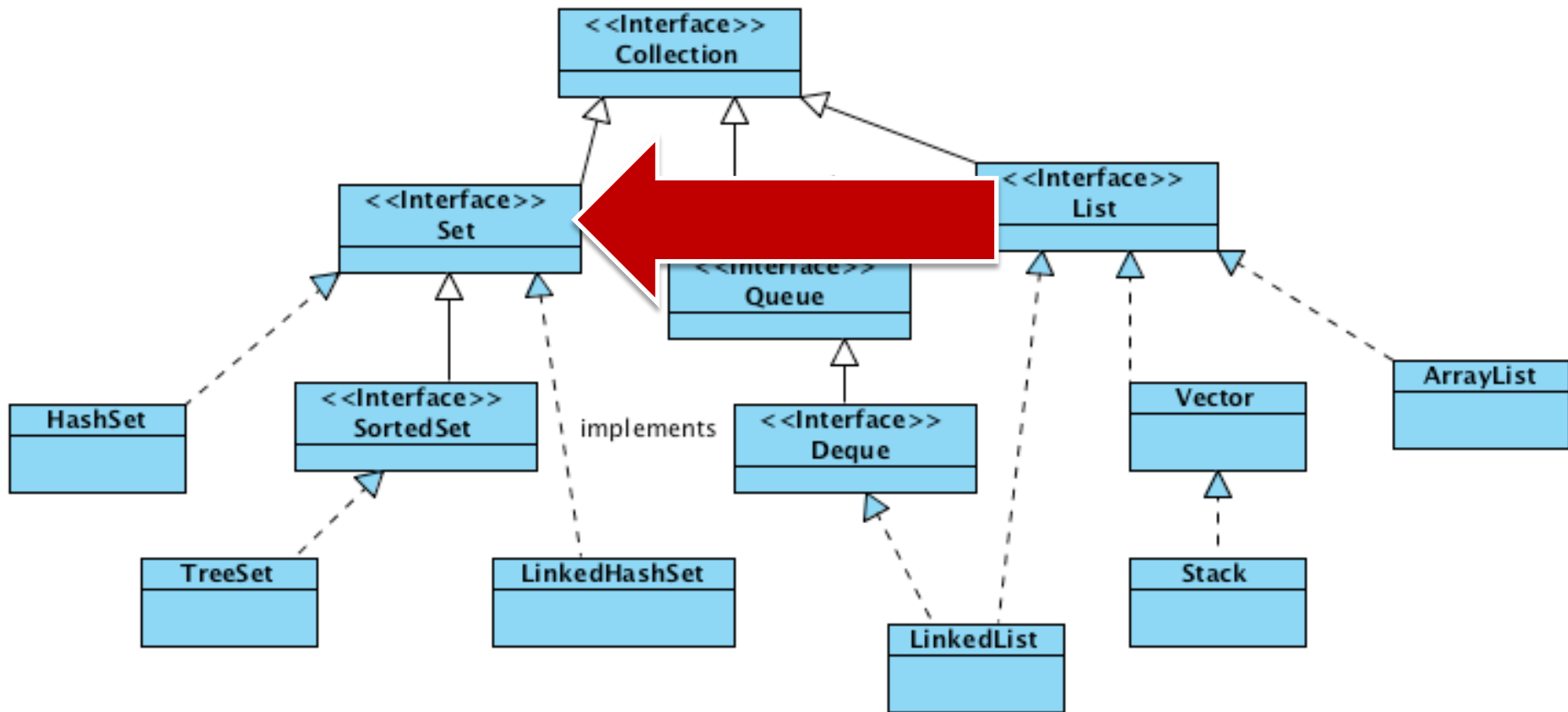




# Collection Family Tree

---



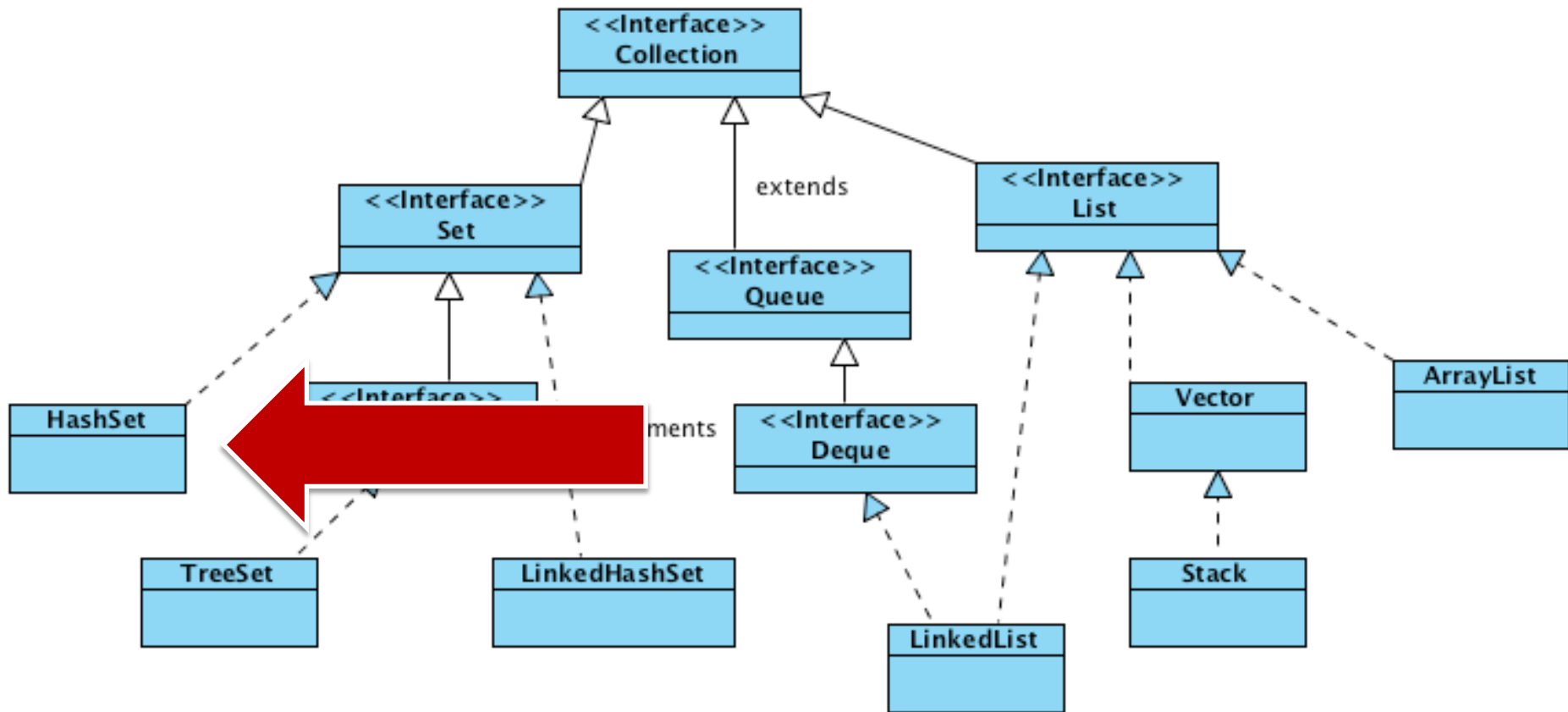


# Set interface

---

- Add/remove elements
  - boolean **add**(element)
  - boolean **remove**(object)
- Search
  - boolean **contains**(object)
- No duplicates
- No positional Access!

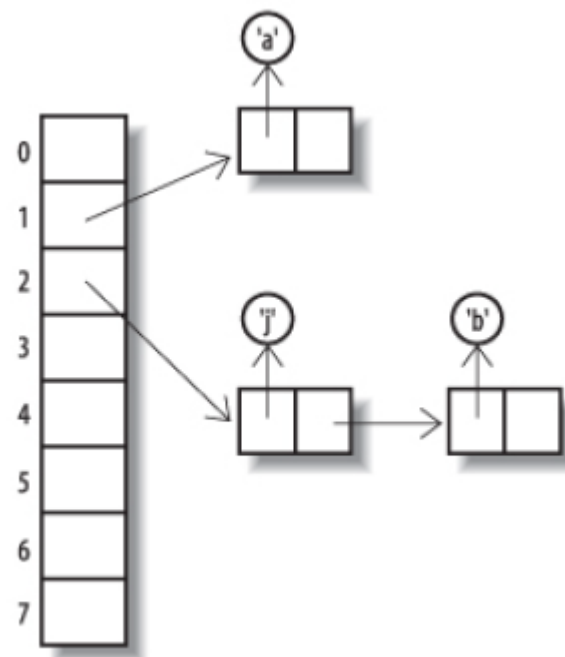
# HashSet





# HashSet

- Add/remove elements
  - boolean **add**(element)
  - boolean **remove**(object)
- Search
  - boolean **contains**(object)
- No duplicates
- No positional Access
- Unpredictable iteration order!

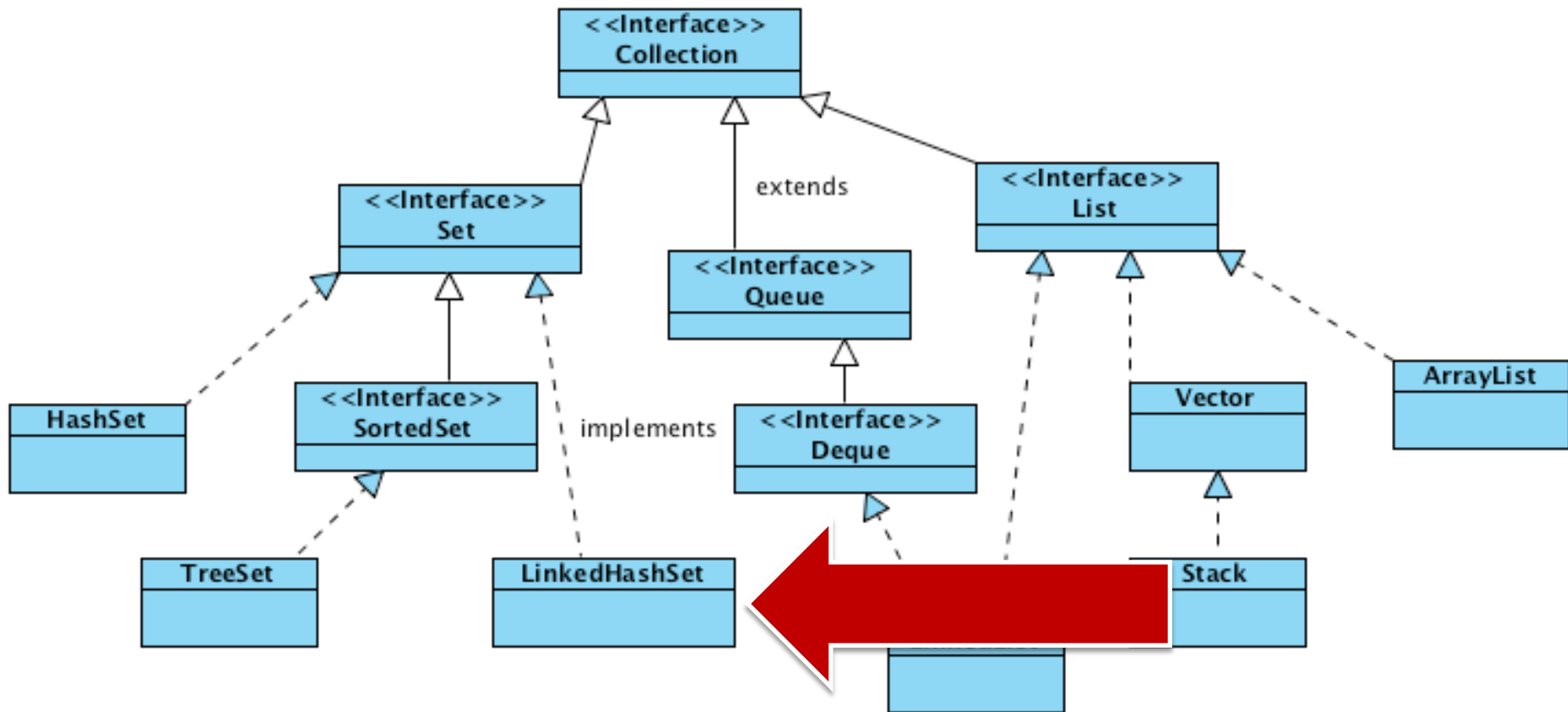


# Costructors

---

- `public HashSet()`
- `public HashSet(Collection<? extends E> c)`

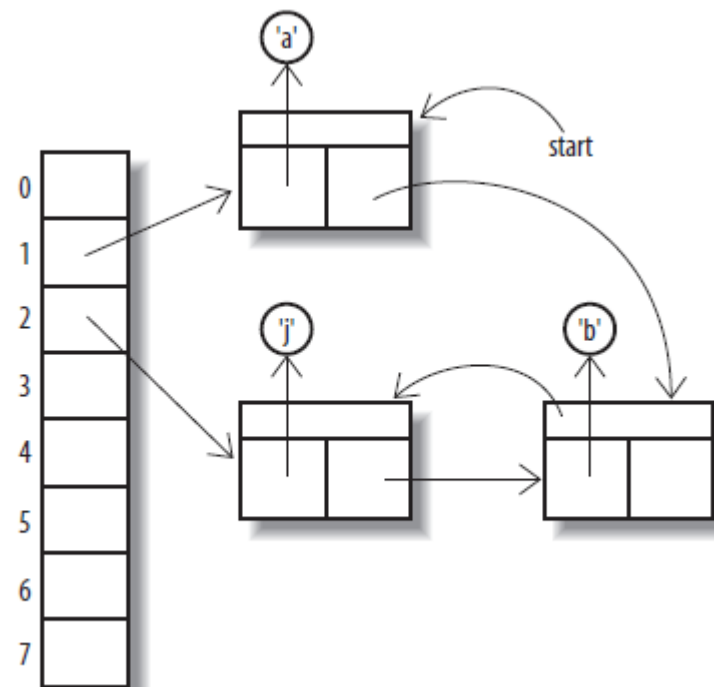
# LinkedHashSet





# LinkedHashSet

- Add/remove elements
  - boolean **add**(element)
  - boolean **remove**(object)
- Search
  - boolean **contains**(object)
- No duplicates
- No positional Access
- **Predictable** iteration order





# Sets vs. Lists

---

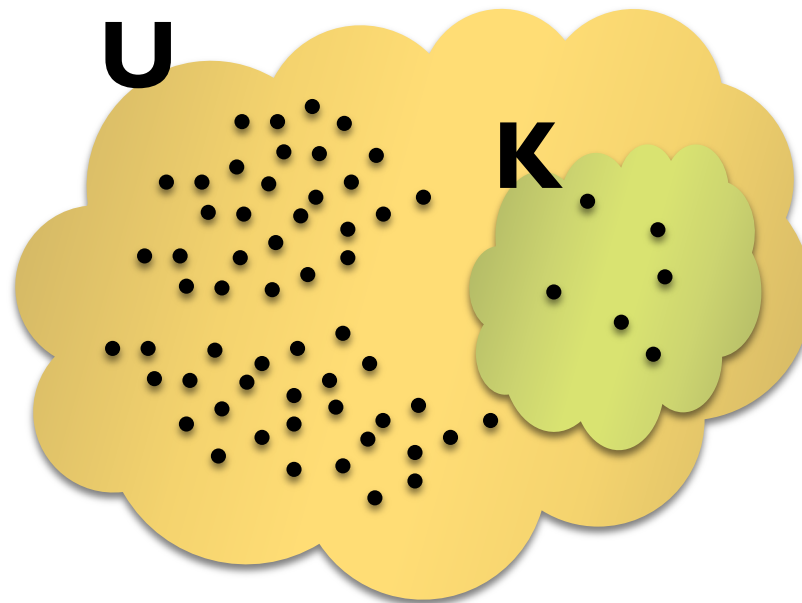
	ArrayList	LinkedList	Set
add(element)	IMMEDIATE	IMMEDIATE	IMMEDIATE
remove(object)	SLUGGISH	LESS SLUGGISH	IMMEDIATE
get(index)	IMMEDIATE	SLUGGISH	n.a.
set(index, elem)	IMMEDIATE	SLUGGISH	n.a.
add(index, elem)	SLUGGISH	SLUGGISH	n.a.
remove(index)	SLUGGISH	SLUGGISH	n.a.
contains(object)	SLUGGISH	SLUGGISH	IMMEDIATE
indexOf(object)	SLUGGISH	SLUGGISH	n.a.



# Notation

---

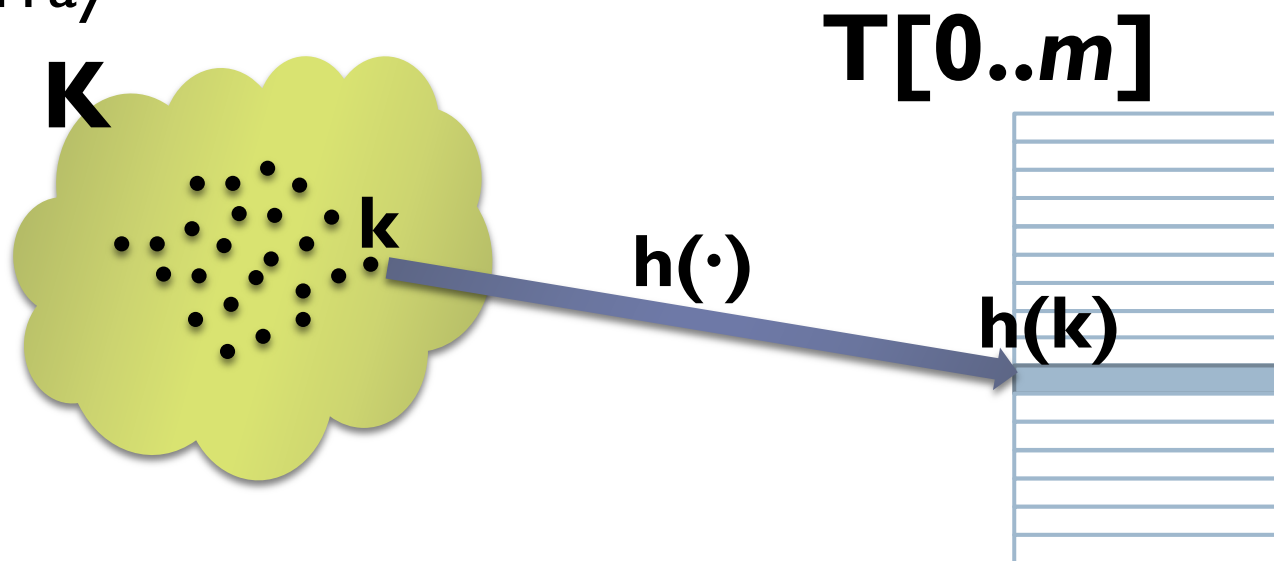
- A set stores *keys*
- $U$  – Universe of all possible keys, e.g., all the possible words
- $K$  – Set of keys actually stored, e.g., all the names of the students of the Politecnico



# Hash Table

---

- Devise a function to transform each *key* into an index (a number)
- Use an array, where the calculated index is the position in the array



# Hash Table

---

- When inserting an object into a hash table you use a key. The hash code of this key is calculated, and used to determine where to **store** the object internally.
- When you need to lookup an object in a hash table you also use a key. The hash code of this key is calculated and used to determine where to **search** for the object.

# Hash Function

---

- Is a function that maps data of arbitrary size to data of fixed size
- Mapping from  $\mathbf{U}$  to the slots of a hash table  $T[0\dots m-1]$

$$h : \mathbf{U} \rightarrow \{0, 1, \dots, m-1\}$$

- $h(k)$  is the “hash value” of key  $k$
- Since the range of the hash values is fixed:
  1. Convert the key into a number
  2. Compression/Expansion:

$$h_N : \mathbf{U} \rightarrow \mathbf{N}^+$$

$$h(k) = h_N(k) \bmod m$$

$$h_R : \mathbf{U} \rightarrow [0, 1[ \in \mathbf{R}$$

$$h(k) = \lfloor h_R(k) \cdot m \rfloor$$



# Hash Function

---

- Main application:
  - Hash table
  - Cryptographic hash function
    - Authentication
    - Ensure file integrity (to avoid tampering)
    - Calculate digest for digital signature
    - *Used by Git too.*



# Hash Function

---

## □ Main properties:

### □ Hash table

- Deterministic: same key, same hash value
- Uniform: “Any key should be equally likely to hash into any of the  $m$  slots, independent of where any other key hashes to”
- Defined range

### □ Cryptographic hash function

- Collision resistance (large hash value) e.g. SHA-1 160 bit
- Non invertible: it is not possible to reconstruct  $k$  from  $h(k)$





# A simple hash function

---

- Split the key into its “component”, then sum their integer representation

- $h_N(k) = h_N(x_0x_1x_2 \dots x_n) = \sum_{i=0}^n x_i$

- $h(k) = h_N(k) \% m$



# A simple hash (problems)

---

## □ Problems

- $h_N(\text{"NOTE"}) = 78+79+84+69 = 310$
- $h_N(\text{"TONE"}) = 310$
- $h_N(\text{"STOP"}) = 83+84+79+80 = 326$
- $h_N(\text{"SPOT"}) = 326$

## □ Problems (m = 173)

- $h(74,778) = 42$
- $h(16,823) = 42$
- $h(1,611,883) = 42$



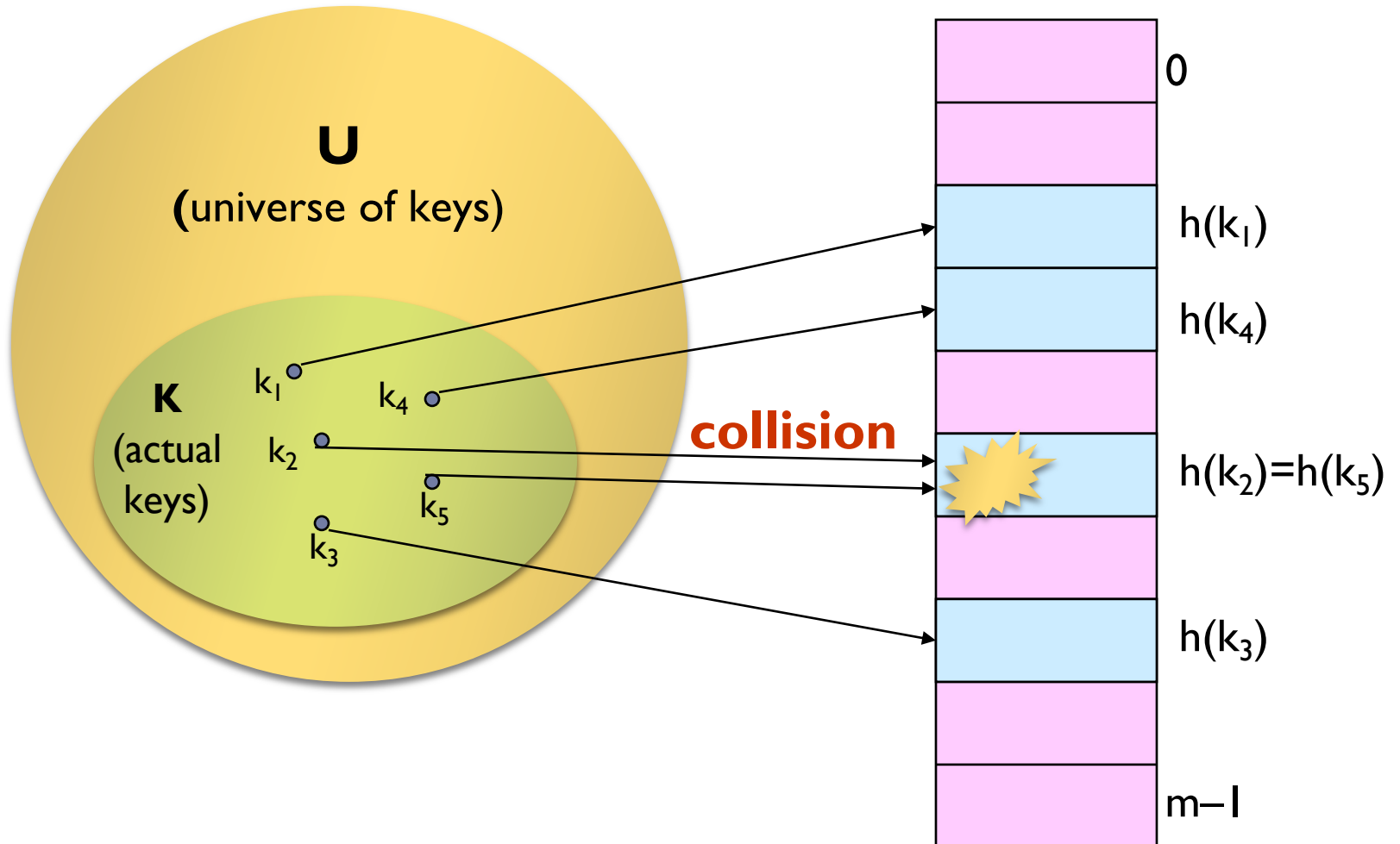
# A perfect hash function

---

- If the data to be hashed is fixed (and small enough), one can use the data itself (reinterpreted as an integer) as the hashed value. The cost of computing this "trivial" (identity) hash function is effectively zero.
- This hash function is **perfect**, as it maps each input to a distinct hash value.



# Collisions



# Collisions

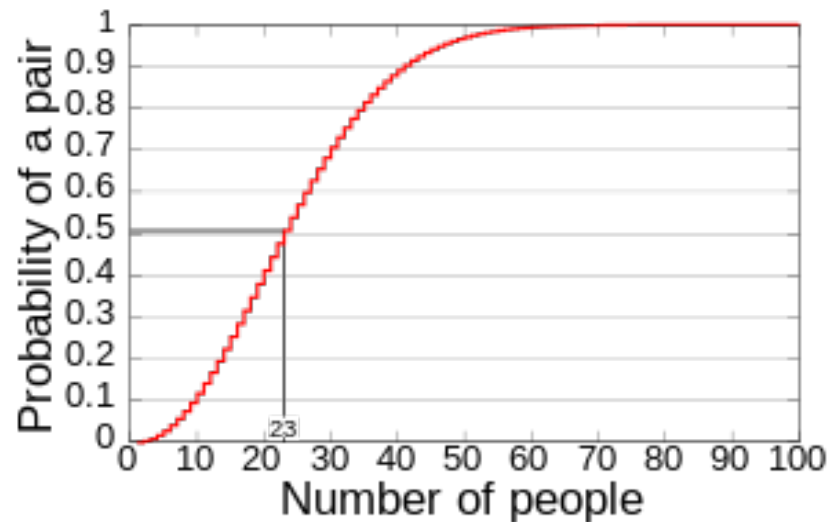
---

- ❑ Collisions are possible!
- ❑ Multiple keys can hash to the same slot
  - ❑ Design hash functions such that collisions are minimized
- ❑ But avoiding collisions is impossible.
  - ❑ Birthday paradox
  - ❑ Design collision-resolution techniques

# Birthday paradox

---

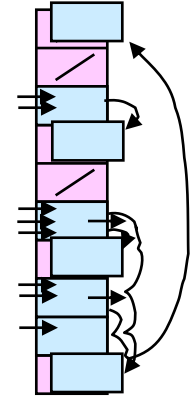
- Let's use birthday as hash function.
  - 365 slot in the array
  - Let's consider the probability of a collision



# Resolution of collisions

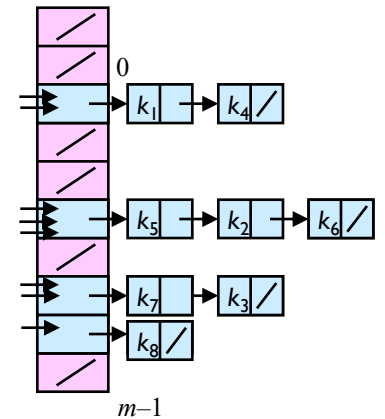
## □ Open Addressing

- When collisions occur, use a systematic (consistent) procedure to store elements in free slots of the table
- “Double hashing”, “linear probing”, ...

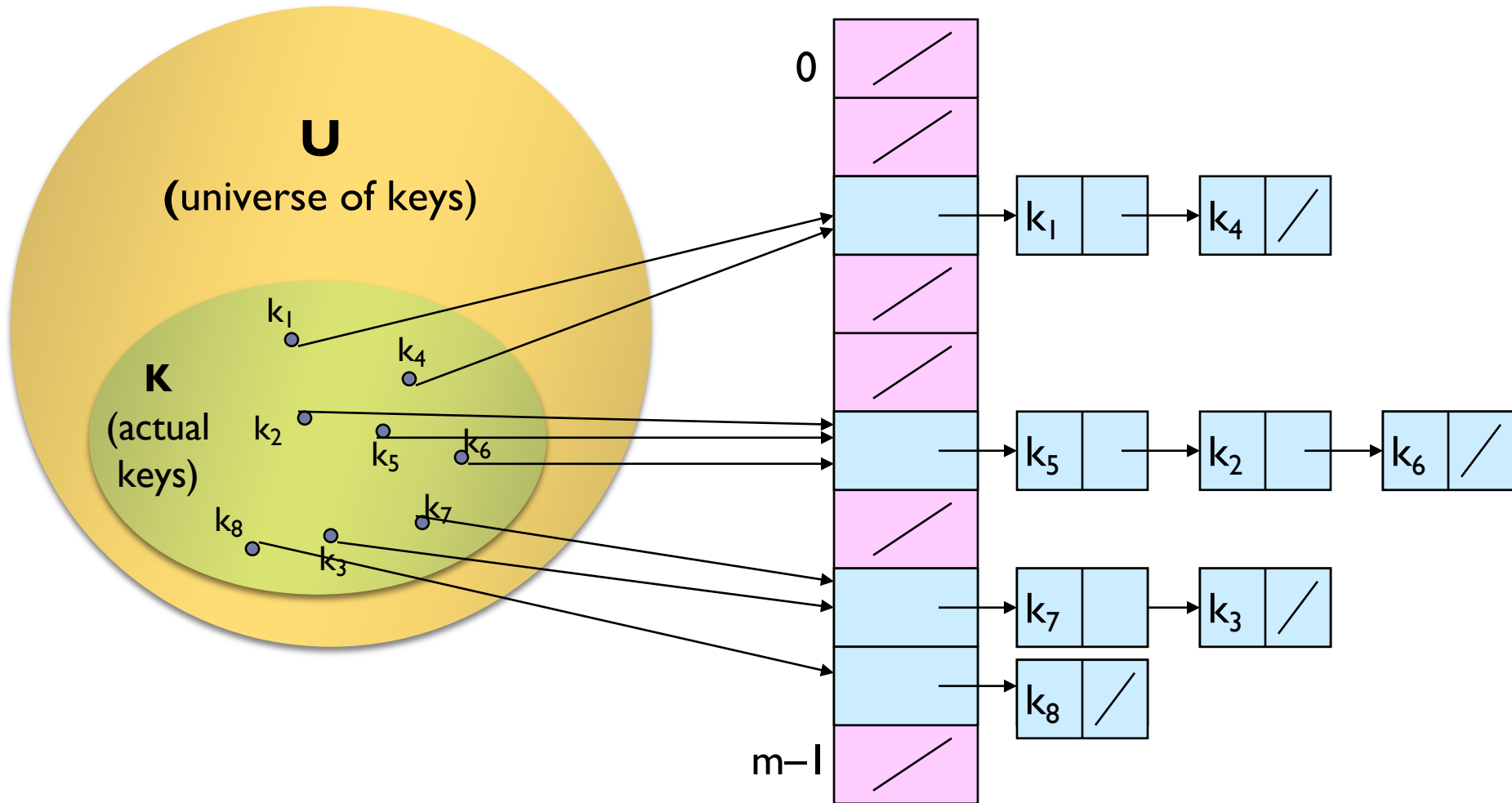


## □ Chaining

- Store all elements that hash to the same slot in a linked list



# Chaining





# JCF's HashSet

---

- Built-in hash function
- Dynamic hash table resize
- Smoothly handles collisions (chaining)
- Very fast operations (well, usually)
- Take it easy!

# Default hash function in Java

---



- ▶ In Java every class must provide a **hashCode()** method which digests the data stored in an instance of the class into a single 32-bit value
- ▶ Some basic types, e.g., String, already have an implemented hashCode()
- ▶ But the basic Object's hashCode() is implemented by **converting the internal address of the object into an integer!**

# The hashCode and Equals Contract



```
public boolean equals(Object obj);  
public int hashCode();
```

- If two objects **are equal** according to the equals() method, then hashCode() must produce the same result
- If two objects **are not equal** according to the equals() method, performances are better whether the hashCode() produces different results



# The hashCode and Equals Contract



```
public boolean equals(Object obj);  
public int hashCode();
```

**hashCode()** and **equals()** should always be defined together

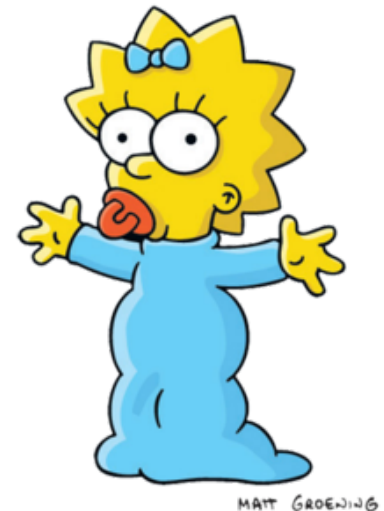


MATT GROENING

# The hashCode and Equals Contract



- `public int hashCode()`
  - returns a 32-bit signed integer
    - 32-bit Float or 32-bit Integer could be used directly
    - Perfect hash function: map each input to a different hash value
  - Eclipse provides a convenient method to automatically generate `equals()` and `hashCode()` implementation
- The calculated hash code only points to a certain "area" internally. Since different key objects could potentially have the same hash code, the hash code itself is no guarantee that the right key is found. Java then iterates this area (all keys with the same hash code) and uses the key's `equals()` method to find the right key



# The hashCode and Equals Contract

---



□ == or !=

□ Used to compare the references of two objects

```
MyData foo = new MyData();
MyData bar = new MyData();

if(foo != bar) {
    System.out.println("References are different");
}

if(foo == bar){
    System.out.println("References are equal");
}
```

# The hashCode and Equals Contract

---



## □ equals()

- Used to give **equality** information about the objects

```
MyData foo = new MyData();
MyData bar = new MyData();

if(foo.equals(bar)) {
    System.out.println("Objects have the same values");
} else {
    System.out.println("Objects have different values");
}
```

# The hashCode and Equals Contract

---



## □ hashCode()

- Return the hash value of an object
- Must behave in a way consistent with the same object equals() method






```
MyData foo = new MyData();
MyData bar = new MyData();

if(foo.equals(bar)) {
    if(foo.hashCode() == bar.hashCode()) {
        System.out.println("Hash code must be equal")
    }
}
```



# Licenza d'uso



- Queste diapositive sono distribuite con licenza Creative Commons “Attribuzione - Non commerciale - Condividi allo stesso modo (CC BY-NC-SA)”
- Sei libero:
  - di riprodurre, distribuire, comunicare al pubblico, esporre in pubblico, rappresentare, eseguire e recitare quest'opera 
  - di modificare quest'opera 
- Alle seguenti condizioni:
  - Attribuzione** — Devi attribuire la paternità dell'opera agli autori originali in modo tale da non suggerire che essi avallino te o il modo in cui tu usi l'opera. 
  - Non commerciale** — Non puoi usare quest'opera per fini commerciali. 
  - Condividi allo stesso modo** — Se alteri o trasformi quest'opera, o se la usi per crearne un'altra, puoi distribuire l'opera risultante solo con una licenza identica o equivalente a questa. 
- <http://creativecommons.org/licenses/by-nc-sa/3.0/>