Hashing

CMSC 132

Department of Computer Science

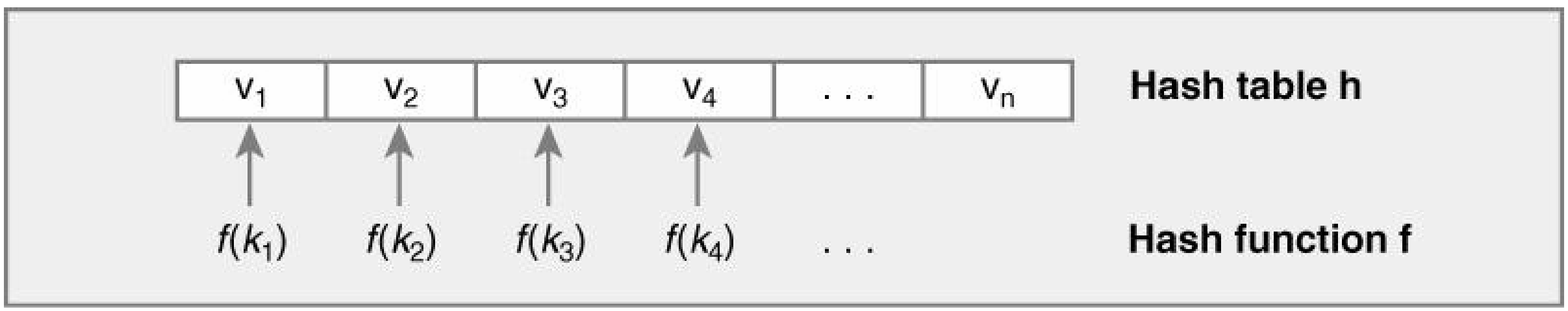
University of Maryland, College Park

Overview

- Hashing
- Scattering Hash Values
- Hash Function
- Open Addressing
- Chaining
- Hashing in Java

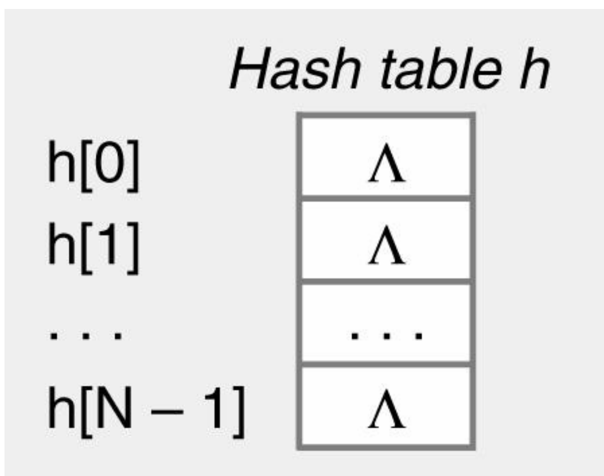
Hashing

- **Approach**
  - **Transform key into number (hash value)**
  - **Use hash value to index object in hash table**
  - **Use hash function to convert key to number**



Hashing

- **Hash Table**
  - **Array indexed using hash values**
  - **Hash Table A with size N**
  - **Indices of A range from 0 to N-1**
  - **Store in A[hashValue % N]**



Location	*Key*
0	Λ
1	10
2	15
3	20
4	Λ
. . .	

Scattering Hash Values

- hashCode is a 32-bit signed int
  - Have to reduce it to 0..N-1
- Could use Math.abs(key.hashCode() % N)
  - might not distribute values well, particularly if N is a power of 2
- Multiplicative congruency method
  - Produces good hash values
  - Hash value = Math.abs((a * key.hashCode()) % N)
  - Where
    - N is table size
    - a, N are large primes

Art and Magic of hashCode

- There is no “right” hashCode function
- some art and magic to finding a good hashCode function, and to finding a hashCode to hashBucket function
- From java.util.HashMap:

```
static int hashBucket(Object x, int N) {
    int h = x.hashCode();
    h += ~(h << 9);
    h ^=  (h >>> 14);
    h +=  (h << 4);
    h ^=  (h >>> 10);
    return Math.abs(h % N);
```

Hash Function

- **Example**

  hashCode("apple") = 5
  hashCode("watermelon") = 3
  hashCode("grapes") = 8
  hashCode("kiwi") = 0
  hashCode("strawberry") = 9
  hashCode("mango") = 6
  hashCode("banana") = 2

- **Perfect hash function**
  - **Unique values for each key**

0	kiwi
1	
2	banana
3	watermelon
4	
5	apple
6	mango
7	
8	grapes
9	strawberry

Hash Function

- **Suppose now**

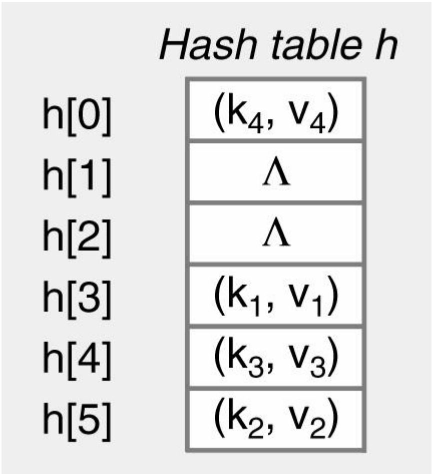
  hashCode("apple") = 5
  hashCode("watermelon") = 3
  hashCode("grapes") = 8
  hashCode("kiwi") = 0
  hashCode("strawberry") = 9
  hashCode("mango") = 6
  hashCode("banana") = 2
  hashCode(“orange") = 3

0	kiwi
1	
2	banana
3	watermelon
4	
5	apple
6	mango
7	
8	grapes
9	strawberry

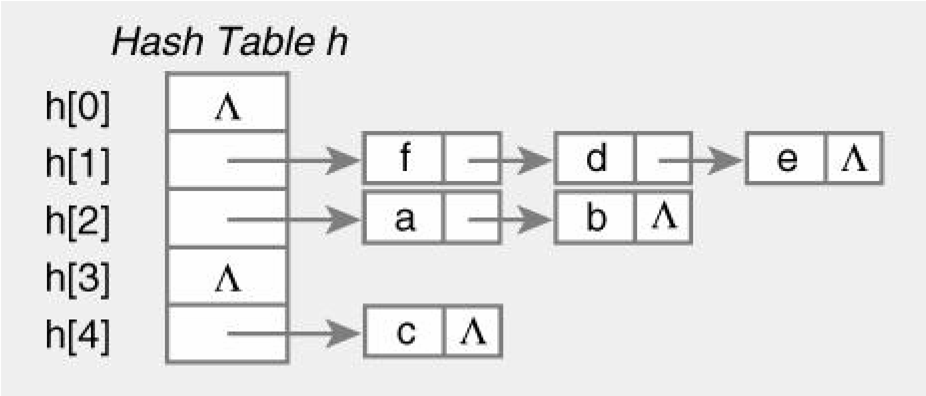
- **Collision**
  - **Same hash value for multiple keys**

Types of Hash Tables

- **Open addressing**
  - **Store objects in each table entry**



- **Chaining (bucket hashing)**
  - **Store lists of objects in each table entry**



Open Addressing Hashing

- Approach
  - Hash table contains objects
  - Probe $\Rightarrow$ examine table entry
  - Collision
    - Move K entries past current location
    - Wrap around table if necessary
  - Find location for X
    - Examine entry at A[bucket(X)]
    - If entry = X, found
    - If entry = empty, X not in hash table
    - Else increment location by K, repeat

Open Addressing Hashing

- Approach
  - Linear probing
    - K = 1
    - May form clusters of contiguous entries
  - Deletions
    - Find location for X
    - If X inside cluster, leave non-empty marker
  - Insertion
    - Find location for X
    - Insert if X not in hash table
    - Can insert X at first unoccupied location

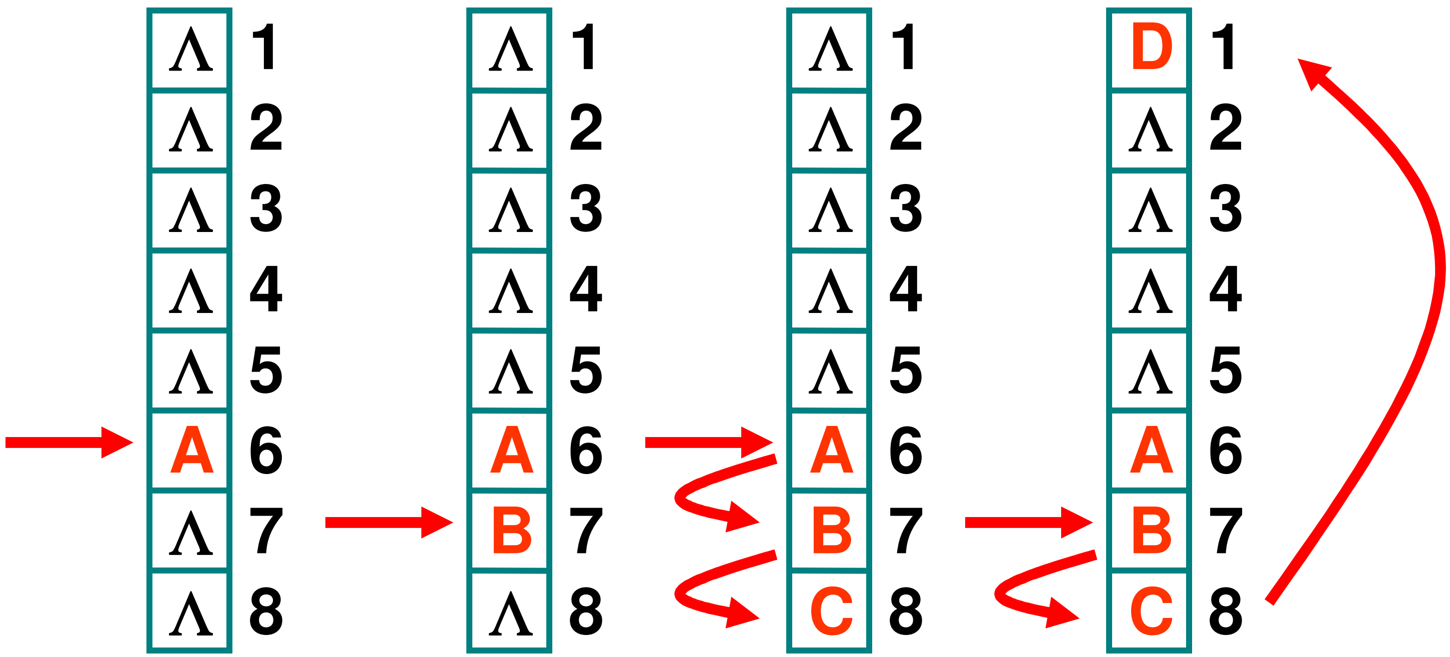
Open Addressing Example

- **Hash codes**
  - H(A) = 6　　H(C) = 6
  - H(B) = 7　　H(D) = 7
- **Hash table**
  - Size = 8 elements
  - Λ = empty entry
  - * = non-empty marker
- **Linear probing**
  - Collision ⇒ move 1 entry past current location

Λ	1
Λ	2
Λ	3
Λ	4
Λ	5
Λ	6
Λ	7
Λ	8

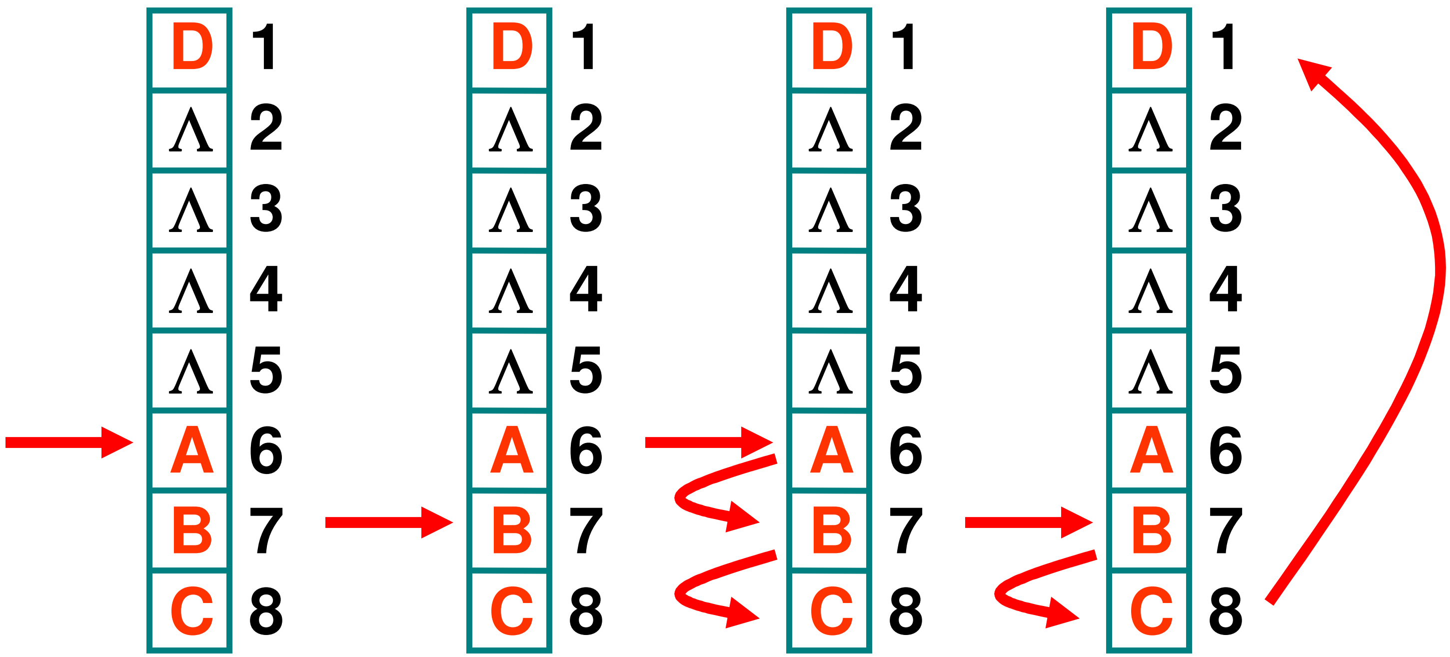
Open Addressing Example

- **Operations**
  - Insert A, Insert B, Insert C, Insert D



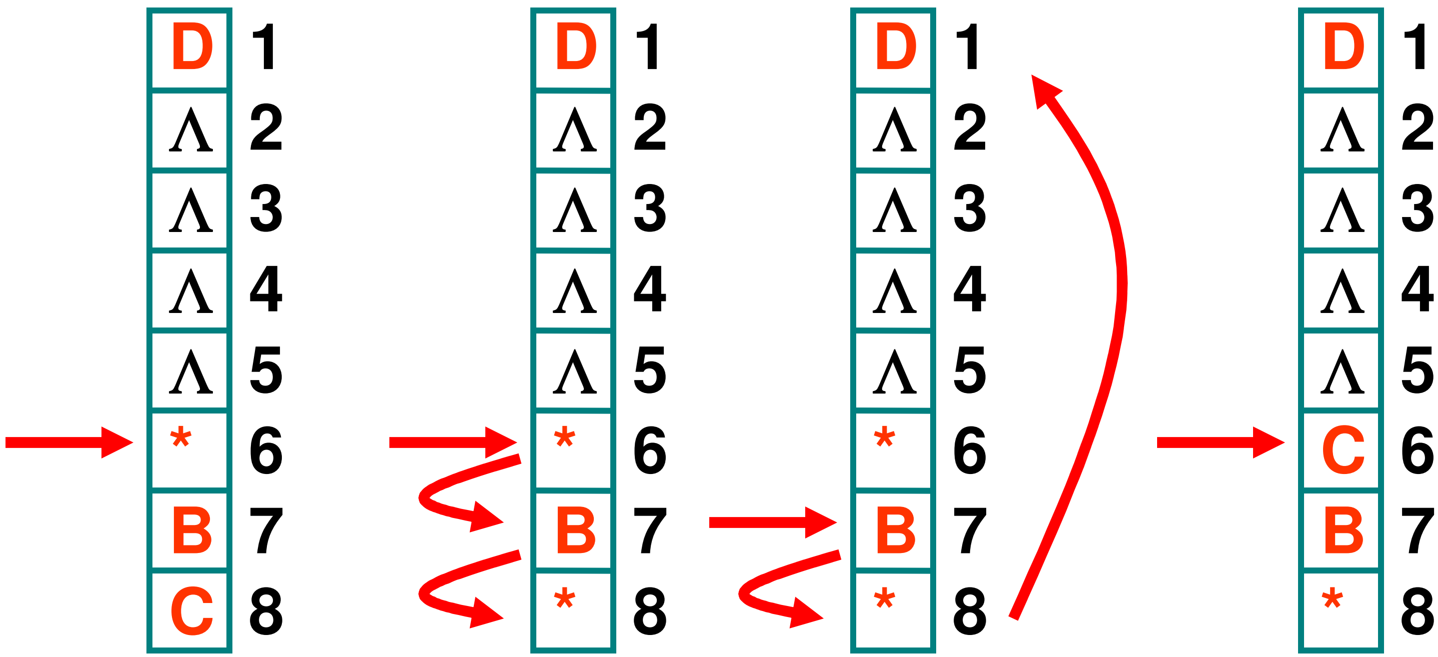
Open Addressing Example

- **Operations**
  - Find A,   Find B,   Find C,   Find D



Open Addressing Example

- **Operations**
  - Delete A, Delete C, Find D, Insert C



Efficiency of Open Hashing

- **Load factor = entries / table size**
- **Hashing is efficient for load factor < 90%**

α	*Number of Comparisons*	*Approximate Behavior*	*(Table Size N = 100)*
0.1	1.06		
0.2	1.13		
0.3	1.21		
0.4	1.33	O(1)	
0.5	1.50		
0.6	1.75		
0.7	2.17		
0.8	3.00		
0.9	5.50	O(log N)	
0.95	10.5		
0.98	26.5	O(N)	
0.99	50.5		

Chaining (Bucket Hashing)

- Approach
  - Hash table contains lists of objects
  - Find location for X
    - Find hash code key for X
    - Examine list at table entry A[key]
  - Collision
    - Multiple entries in list for entry

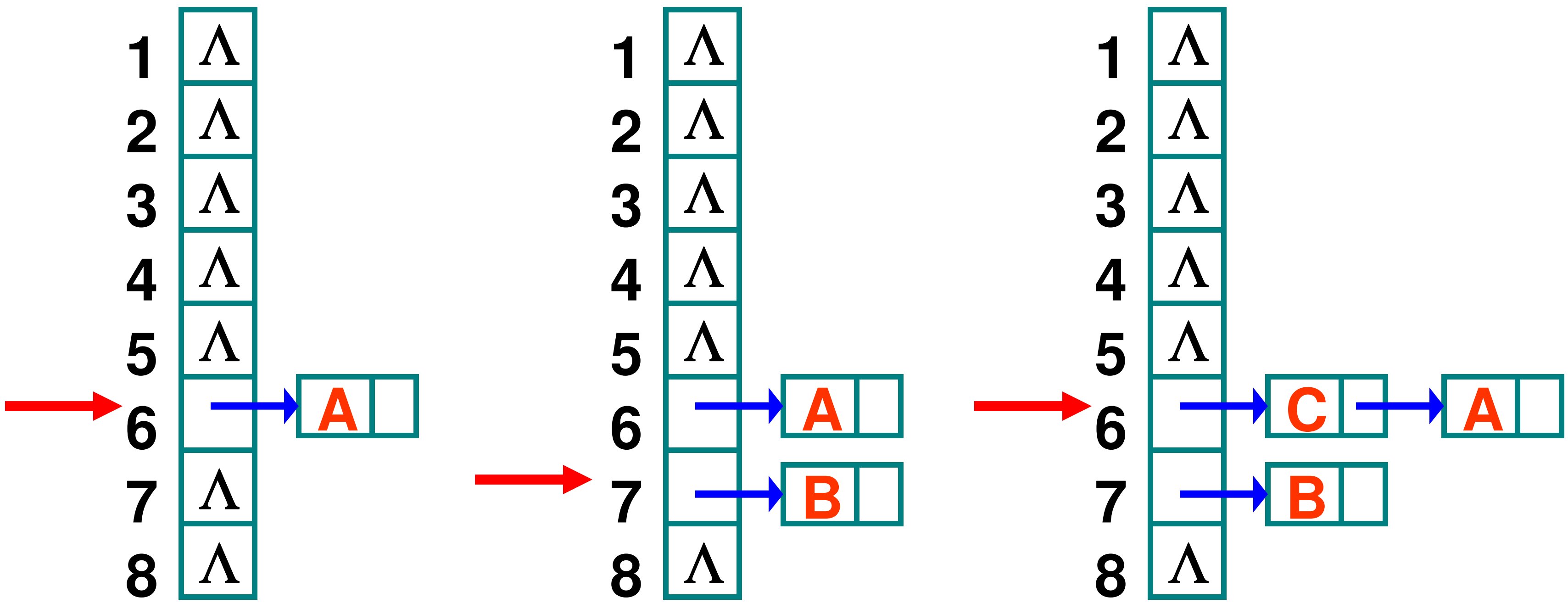
Chaining Example

- Hash codes
  - H(A) = 6 H(C) = 6
  - H(B) = 7 H(D) = 7
- Hash table
  - Size = 8 elements
  - Λ = empty entry

1	Λ
2	Λ
3	Λ
4	Λ
5	Λ
6	Λ
7	Λ
8	Λ

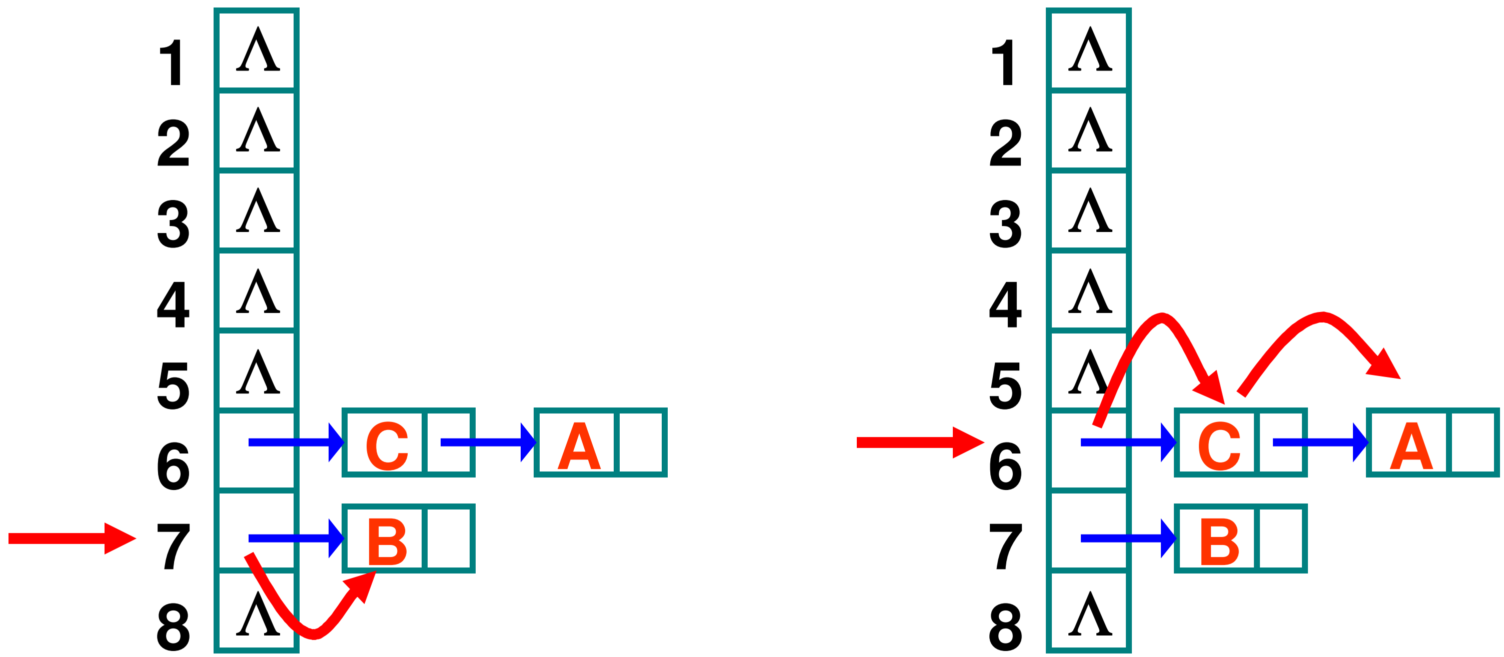
Chaining Example

- **Operations**
  - Insert A, Insert B, Insert C



Chaining Example

- **Operations**
  - **Find B, Find A**



Efficiency of Chaining

- Load factor = entries / table size
- Average case
  - Evenly scattered entries
  - Operations = O(load factor)
- Worse case
  - Entries mostly have same hash value
  - Operations = O(entries)

Hashing in Java

- Collections
  - HashMap & HashSet implement hashing
- Objects
  - Built-in support for hashing
    - boolean equals(object o)
    - int hashCode()
  - Can override with own definitions
  - Must be careful to support Java contract

Java Contract

- hashCode()
  - Must return same value for object in each execution, provided no information used in equals comparisons on the object is modified
- equals()
  - if a.equals(b), then a.hashCode() must be the same as b.hashCode()
  - if a.hashCode() != b.hashCode(), then !a.equals(b)
- a.hashCode() == b.hashCode()
  - Does not imply a.equals(b)
  - Though Java libraries will be more efficient if it is true