

HASHING

Let's consider an Example

Suppose, a company with 250 employees, assign a 5-digit employee number to each employee which is used as primary key in company's employee file.

- We can use employee number as a address of record in memory.
- The search will require no comparisons at all.
- Unfortunately, this technique will require space for 1,00,000 memory locations, where as fewer locations would actually be used.
- So, this trade off for time is not worth the expense.

What is the SOLUTION to this problem?

HASHING!!

- Idea of **using a key to determine address of a record**, but it must be modified so that large amount of memory space is not wasted.
- This modification takes the form of a function H from the set of keys, K into set of memory address, L .

$H: K \rightarrow L$, Is called a *Hash Function*

But there's a Problem!!

COLLISION

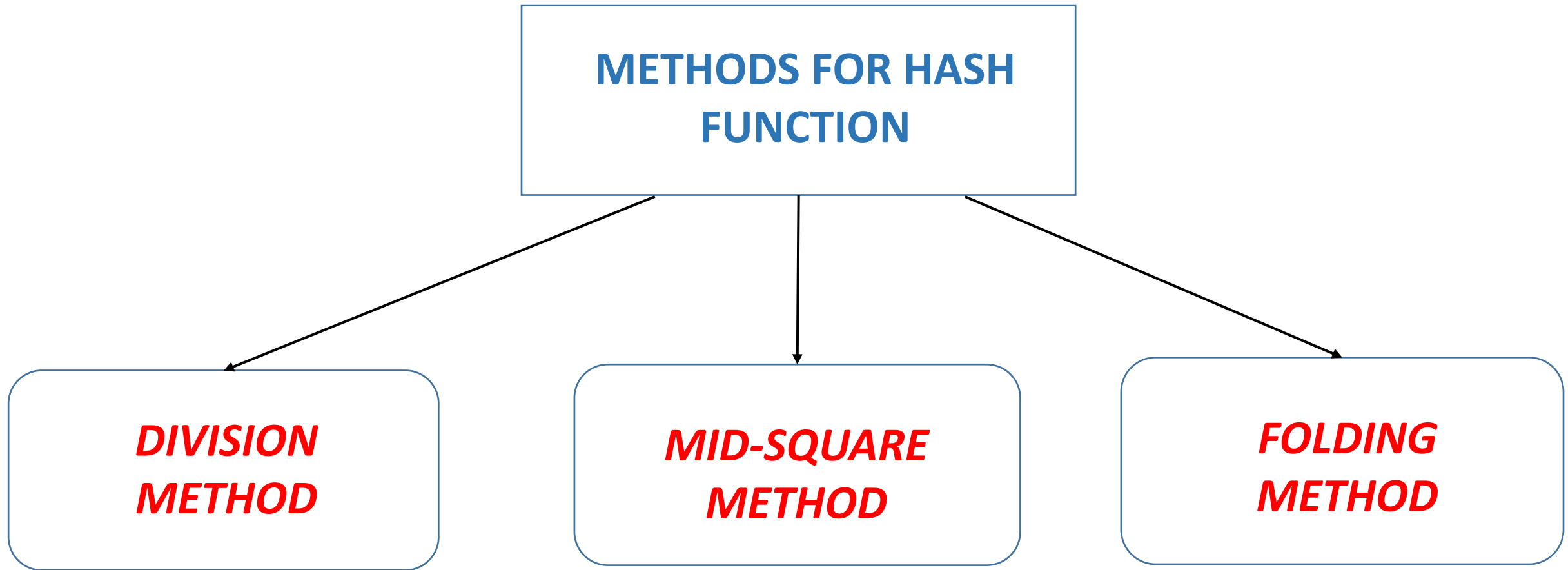
- It may be possible that *function H may not yield distinct values* .
- That means, it is possible that *two different keys k_1 and k_2 may yield same hash address.*
- This situation is called *Collision*.
- Some method must be used to resolve it!

Hash Function

Principal criteria used in selecting a hash function $H: K \rightarrow L$ are :

- 1) Function 'H' should be easy and quick to compute.
- 2) The function 'H' should as far as possible, i.e. uniformly distribute the hash address through out the set L so that there are minimum number of collision.

Types of Hash Function



1. Division Method

- Choose a number m larger than number of keys in k , n
- Number m is either a prime number or a number without small divisor, since it minimizes number of collisions.
- Hash function H is defined by :

$$H(k) = k(\bmod m) \quad \text{or}$$

$$H(k) = k(\bmod m) + 1$$

where $k(\bmod m)$ denotes the remainder when k is divided by m .

The second formula is used when we want a hash address to range from 1 to m rather than 0 to $m-1$.

Example

- Consider a company, with 68 employees, assigns a 4-digit employee number to each employee. Suppose L consists of 100 two-digit address: 00, 01, 02 ,99.
- Consider Employee numbers: **3205, 7148, 2345**
- Choose a prime number ***m*** close to 99, let's take $m = 97$
- Apply **$H(k) = k(\bmod m)$** to all the above numbers.
- **$H(3205)=4$** (dividing 3205 by 97 gives remainder of 4)
- So, for 3205 address will be $4+1=5$
- Similarly, **$H(7148)=67+1$** and **$H(2345)=17+1$** [taking addresses from range 1 to m]

2. Mid-Square Method

- The key k is squared.
- Then the hash function H is defined by $H(k) = \ell$, where ℓ is obtained by deleting digits from both end of k^2 .
- **Example:**

$k =$	3205	7148	2345
$k^2 =$	10272025	51093904	5499025
$H(k) =$	72	93	99

3. Folding Method

- The key k is partitioned into number of parts $k_1, k_2 \dots k_r$, where each part, except possibly the last, has same number of digits as required address.
- Then parts are added together, ignoring the last carry.
- Hash function, **$H(k) = k_1 + k_2 + k_3 + \dots + k_r$**

Example

- Chopping the key into two parts and yields following hash addresses:
 - i. $H(3205)=32+05=37$
 - ii. $H(7148)=71+48=19$
 - iii. $H(2345)=23+45=68$

Another variation

- i. $H(3205)=32+50=82$
- ii. $H(7148)=71+84=55$
- iii. $H(2345)=23+54=77$

What is Collision?

- A collision occurs when two values in the set hash to the same value.
- It means we want to add new record *r* with key *k* to our file *F*, but memory location address $h(k)$ is already occupied.
- **Collision**: That event when 2 hash table elements map into the same slot in the array.
- Example: add 41, 34, 7, 18, then 21
 - 21 hashes into the same slot as 41!
 - 21 should not replace 41 in the hash table; they should both be there

Collision resolution: means for fixing collisions in a hash table.

0	
1	14 21
2	
3	
4	34
5	
6	
7	7
8	18
9	

Solution for Collision (Collision Resolution)

- There are two general ways to resolve collisions :
 - **Open addressing** (array method)
 - ❑ *Quadratic Probing*
 - ❑ *Double hashing(Re-hashing)*
 - ❑ *Linear Probing*
 - **Separate Chaining** (linked list method)

Open Addressing: Linear probing and Modifications

Linear Probing: resolving collisions in **slot i** by putting the colliding element into the next available slot ($i+1, i+2, \dots$).

- Example: add 41, 34, 7, 18, then 21, then 57
 - 21 collides (41 is already there), so we search ahead until we find empty slot 2
 - 57 collides (7 is already there), so we search ahead twice until we find empty slot 9
- Lookup algorithm becomes slightly modified; we have to loop now until we find the element or an empty slot.
- What happens when the table gets mostly full?

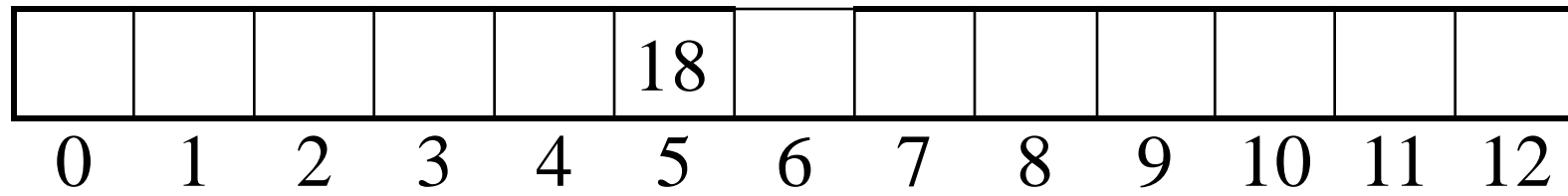
0	
1	41
2	21
3	
4	34
5	
6	
7	7
8	18
9	57

Linear Probing

$$h(K) = K \bmod 13$$

Insert: 18, 41, 22, 59, 32, 31, 73

$h(K)$: 5

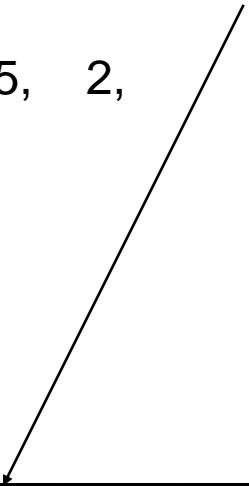


Linear Probing

$$h(K) = K \% 13$$

Insert: 18, 41, 22, 59, 32, 31, 73

$h(K)$: 5, 2,



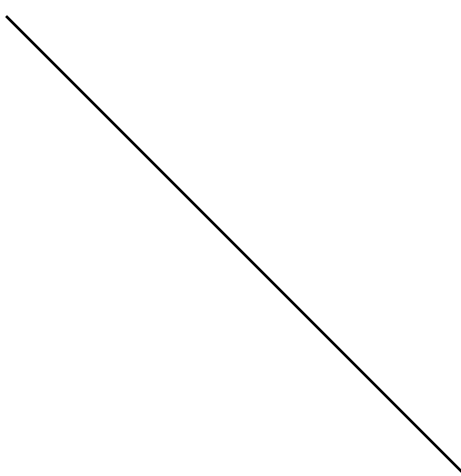
		41			18							
0	1	2	3	4	5	6	7	8	9	10	11	12

Linear Probing

$$h(K) = K \% 13$$

Insert: 18, 41, 22, 59, 32, 31, 73

$h(K)$: 5, 2, 9,




		41			18				22			
0	1	2	3	4	5	6	7	8	9	10	11	12

Linear Probing

$$h(K) = K \% 13$$

Insert: 18, 41, 22, 59, 32, 31, 73

$h(K)$: 5, 2, 9, 7,



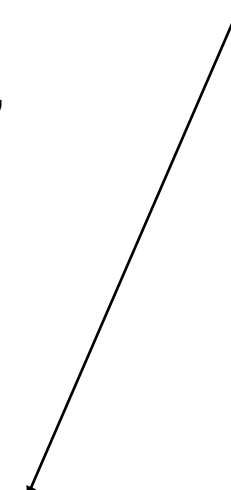
		41			18		59		22			
0	1	2	3	4	5	6	7	8	9	10	11	12

Linear Probing

$$h(K) = K \% 13$$

Insert: 18, 41, 22, 59, 32, 31, 73

$h(K)$: 5, 2, 9, 7, 6,



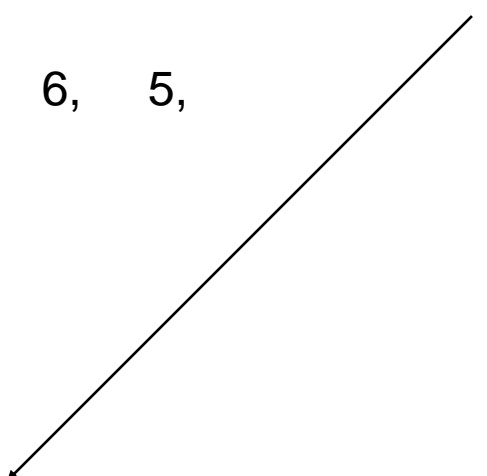
		41			18	32	59		22			
0	1	2	3	4	5	6	7	8	9	10	11	12

Linear Probing

$$h(K) = K \% 13$$

Insert: 18, 41, 22, 59, 32, 31, 73

$h(K)$: 5, 2, 9, 7, 6, 5,



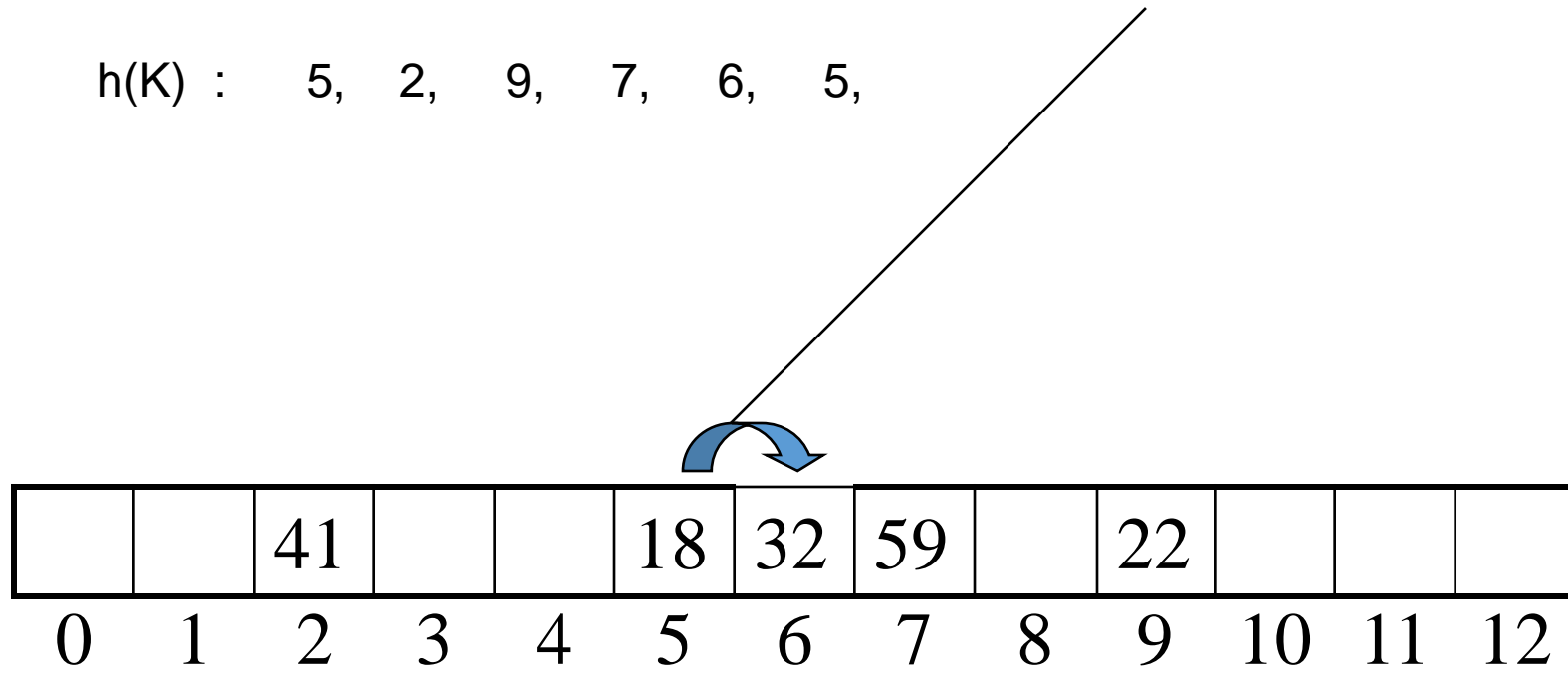
		41			18	32	59		22			
0	1	2	3	4	5	6	7	8	9	10	11	12

Linear Probing

$$h(K) = K \% 13$$

Insert: 18, 41, 22, 59, 32, 31, 73

$h(K)$: 5, 2, 9, 7, 6, 5,

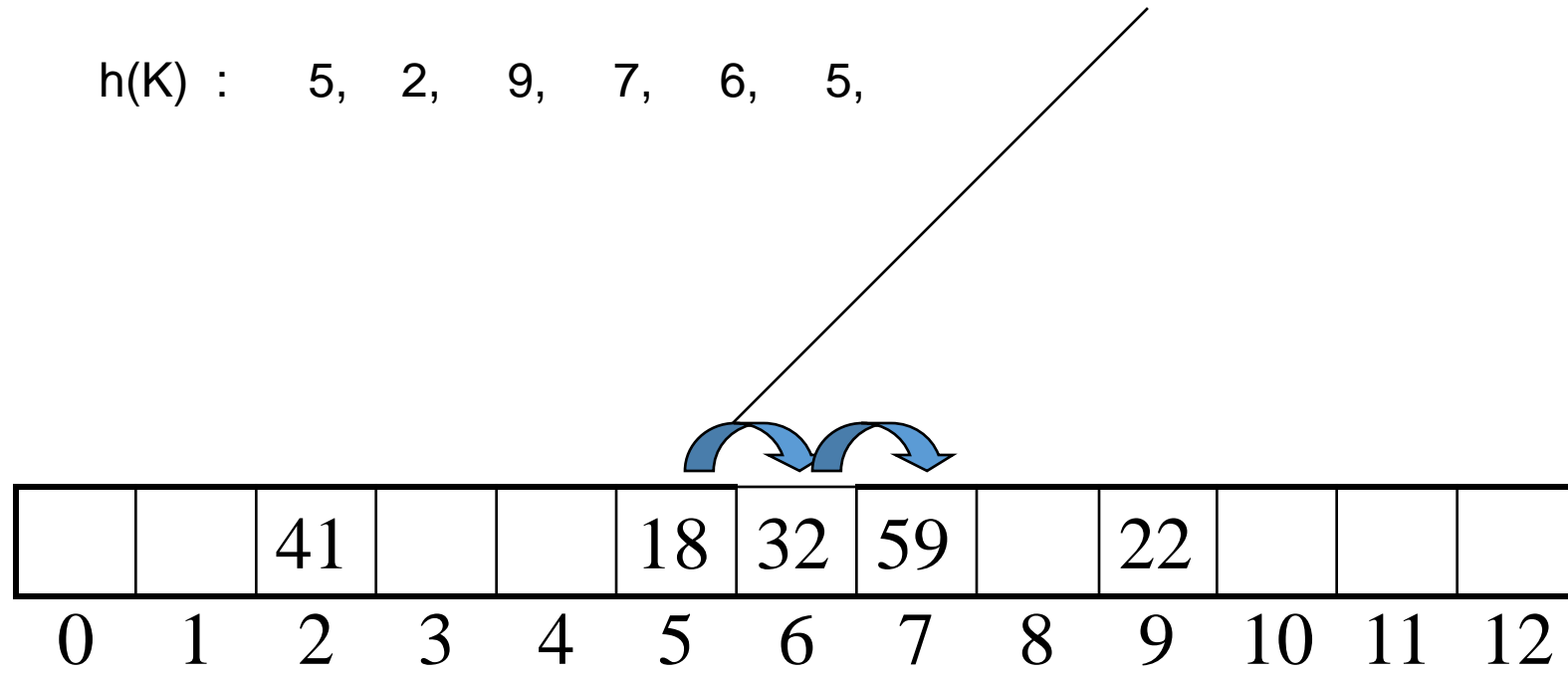


Linear Probing

$$h(K) = K \% 13$$

Insert: 18, 41, 22, 59, 32, 31, 73

$h(K)$: 5, 2, 9, 7, 6, 5,

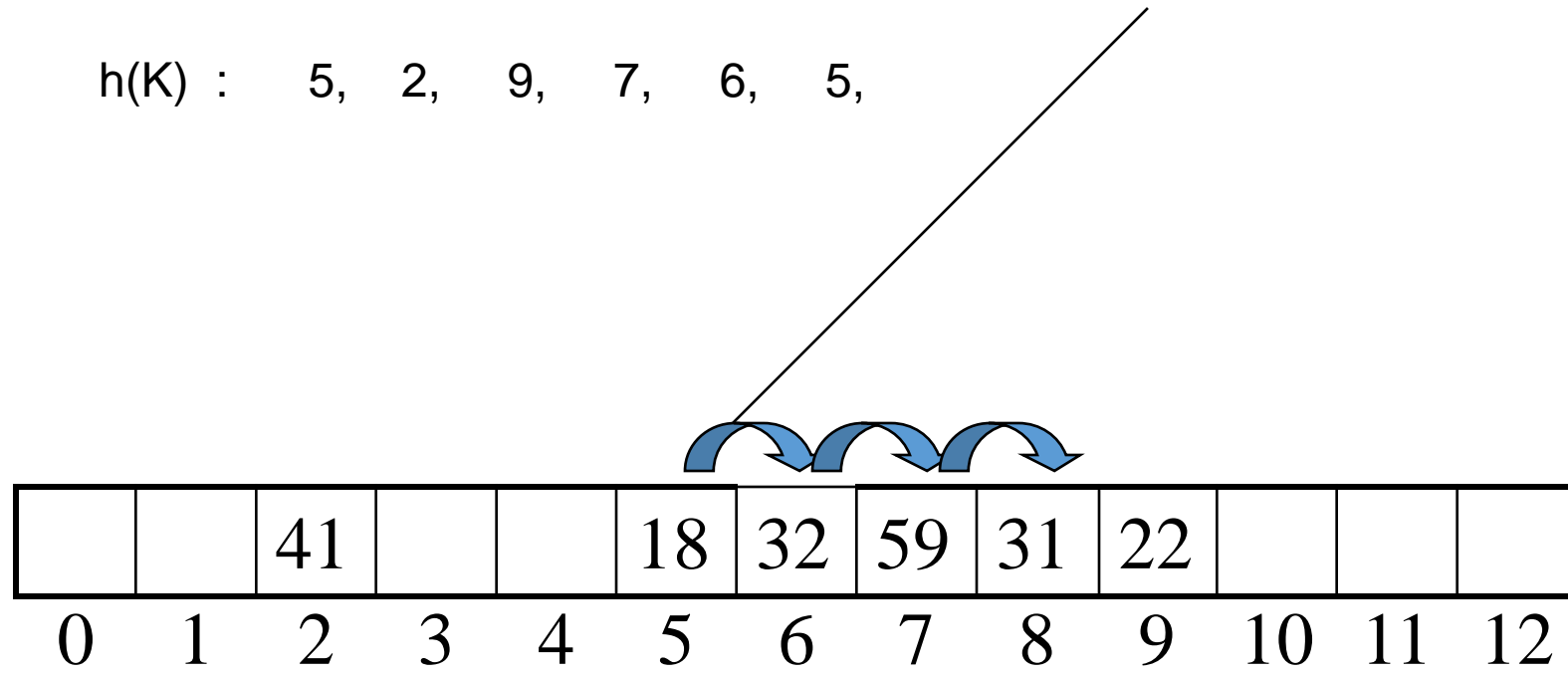


Linear Probing

$$h(K) = K \% 13$$

Insert: 18, 41, 22, 59, 32, 31, 73

$h(K)$: 5, 2, 9, 7, 6, 5,

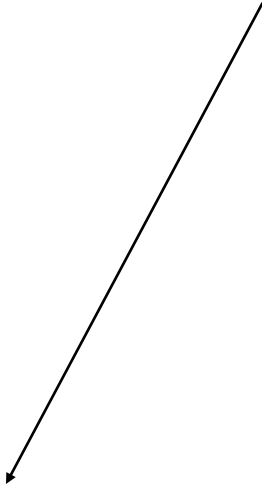


Linear Probing

$$h(K) = K \% 13$$

Insert: 18, 41, 22, 59, 32, 31, 73

$h(K)$: 5, 2, 9, 7, 6, 5, 8



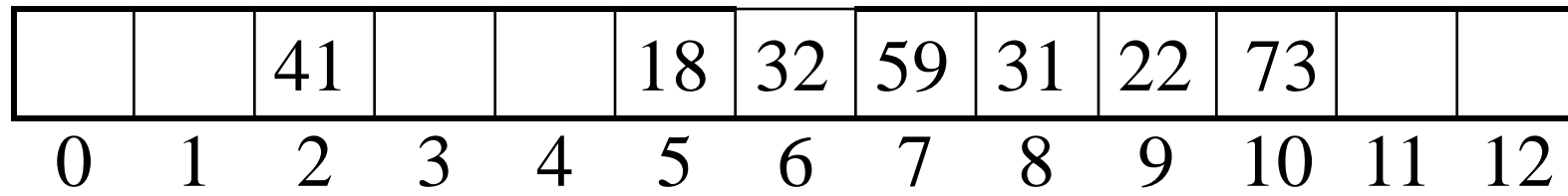
		41			18	32	59	31	22			
0	1	2	3	4	5	6	7	8	9	10	11	12

Linear Probing

$$h(K) = K \% 13$$

Insert: 18, 41, 22, 59, 32, 31, 73

$h(K)$: 5, 2, 9, 7, 6, 5, 8



Open Addressing: Quadratic probing & Double Hashing

- Disadvantage of linear probing: Records tend to cluster, i.e. , appear next to one another, when the load factor $> 50\%$.

The two technique that minimize the clustering are :

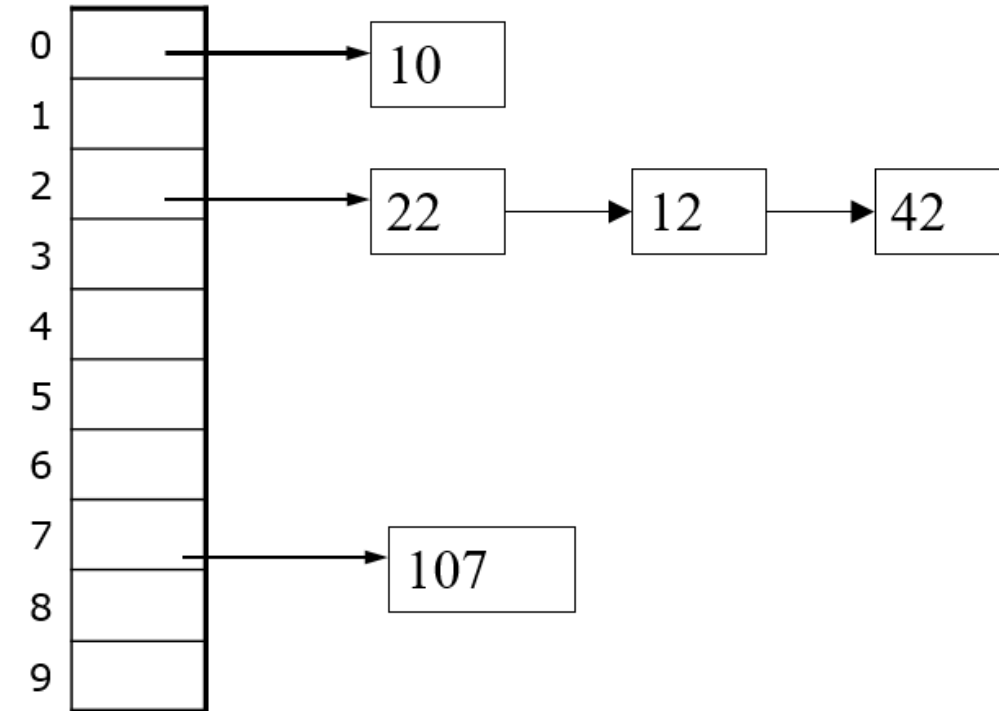
1. **Quadratic probing**: Suppose the record R with key K has the hash address $H(k) = h$. Then instead of searching the locations with address $h, h+1, h+2, \dots$, we search the location with address
$$h, h+1, h+4, h+9, \dots, h+i^2.$$
2. **Double hashing**: Here the second hash function H' is used for resolving a collision. Suppose a record R with key k has a hash address $H(k) = h$ and $H'(k) = h' \neq m$. Then, we linearly search the locations with address

$$h, h+h', h+2h', h+3h', \dots, h+nh'$$

0	49
1	
2	58
3	79
4	
5	
6	
7	
8	18
9	89

Separate Chaining

- It involves maintaining 2 tables in memory.
- First of all, as before, there is a table T in memory which contains the records in F, except that T now has an additional field **LINK** which is used so that all record in T with same hash address **h** may be linked together to form a linked list. Second, there is a hash address table LIST which contain pointers to linked lists in T.
- Suppose a new record **R** with key **k** is added to the file **F**. **We place R in the first available location in the table T and then add R to the linked list with pointer LIST[H(k)].**
- All keys that map to the same hash value are kept in a linked list.



Difference between Open Addressing and Chaining

SEPARATE CHAINING

1. Chaining is Simpler to implement.
2. In chaining, Hash table never fills up, we can always add more elements to chain.
3. Chaining is Less sensitive to the hash function or load factors.
4. Chaining is mostly used when it is unknown how many and how frequently keys may be inserted or deleted.
5. Cache performance of chaining is not good as keys are stored using linked list.
6. Wastage of Space (Some Parts of hash table in chaining are never used).
7. Chaining uses extra space for links.

OPEN ADDRESSING

- Open Addressing requires more computation.
- In open addressing, table may become full.
- Open addressing requires extra care for to avoid clustering and load factor.
- Open addressing is used when the frequency and number of keys is known.
- Open addressing provides better cache performance as everything is stored in the same table.
- In Open addressing, a slot can be used even if an input doesn't map to it.
- No links in Open addressing