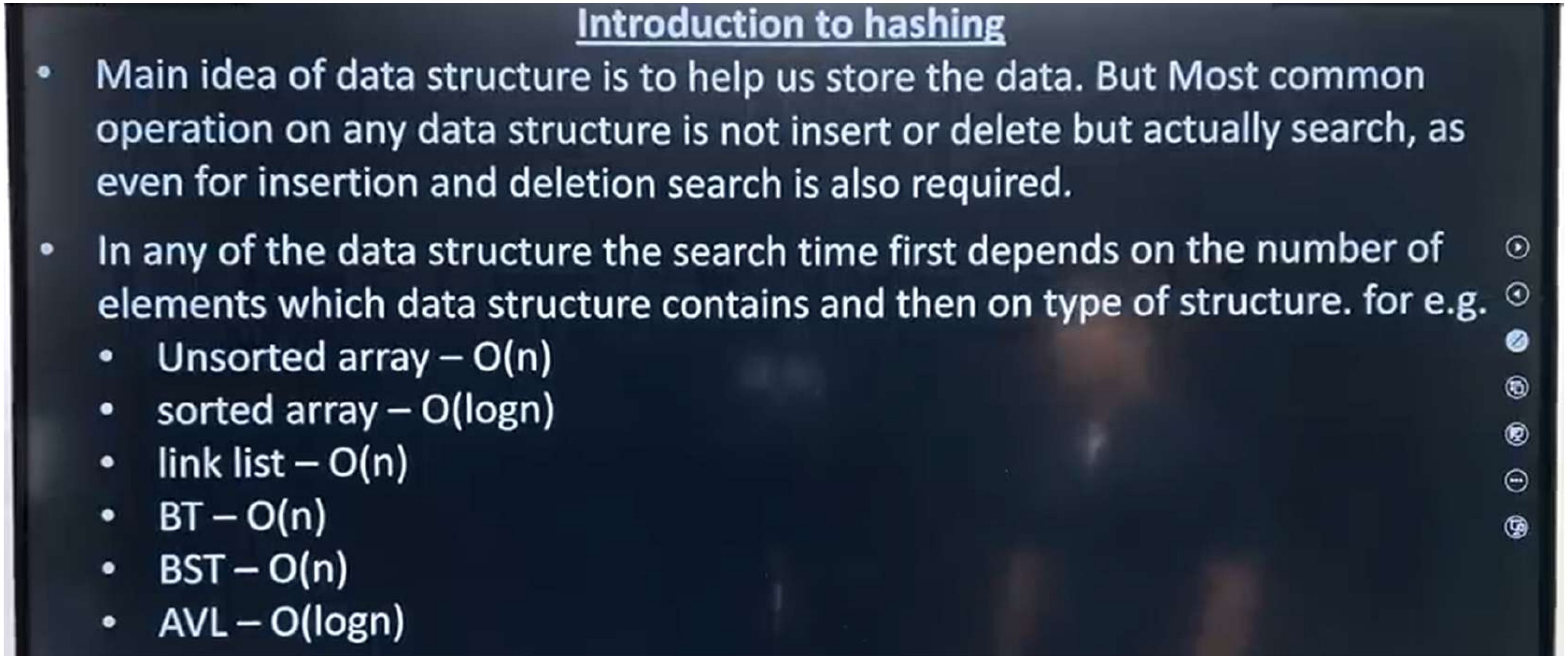
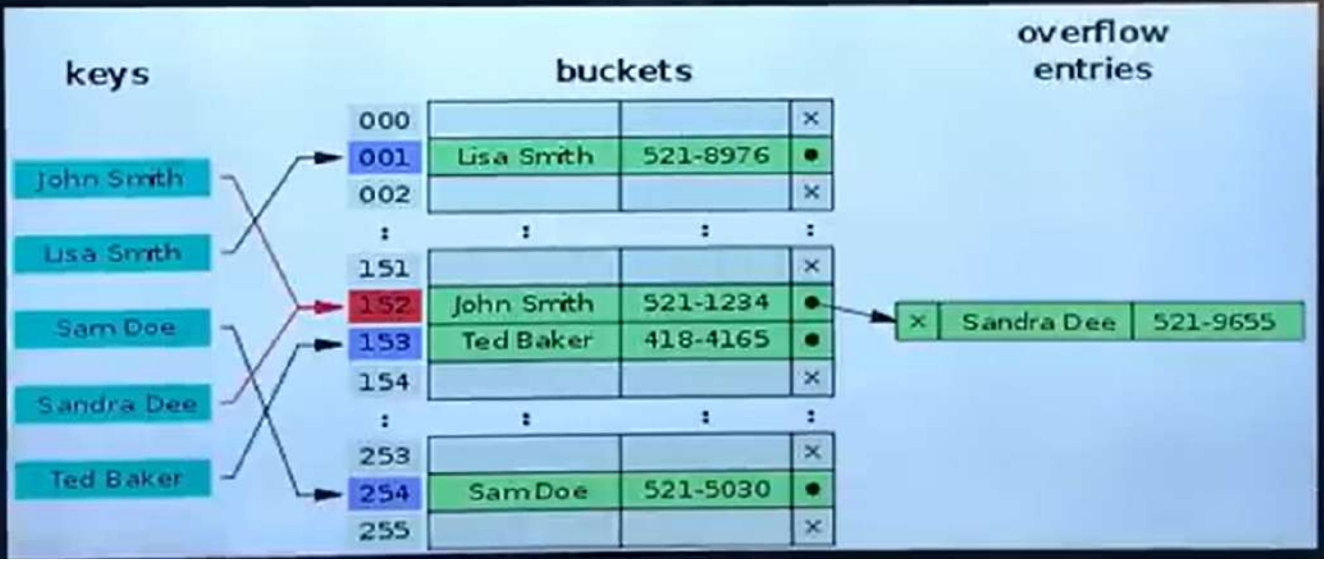
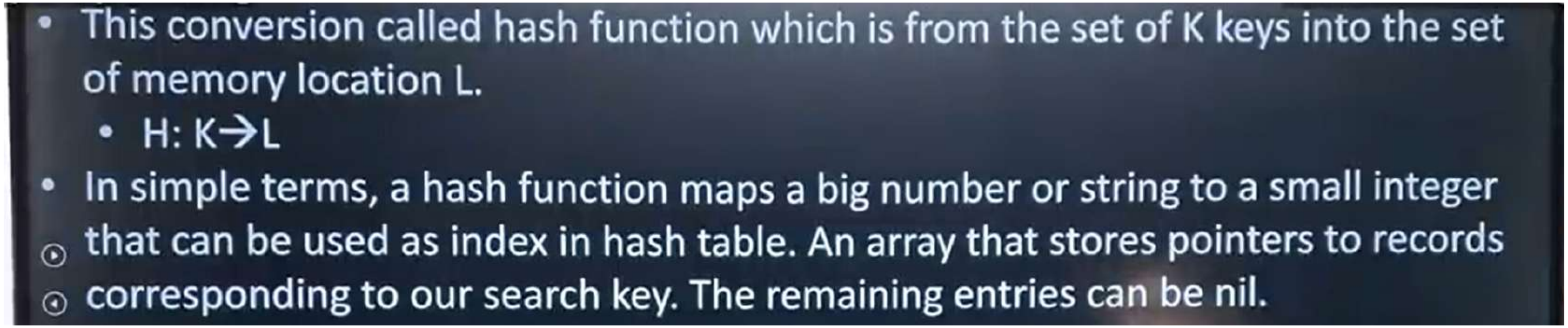


Introduction to hashing

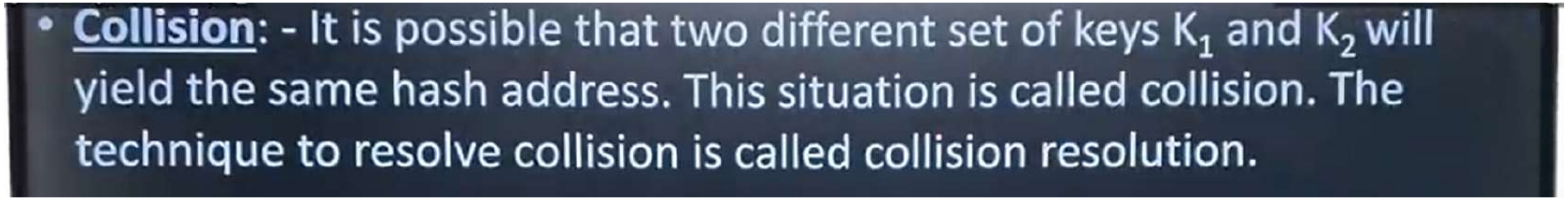
- Main idea of data structure is to help us store the data. But Most common operation on any data structure is not insert or delete but actually search, as even for insertion and deletion search is also required.
- In any of the data structure the search time first depends on the number of elements which data structure contains and then on type of structure. for e.g.
    - Unsorted array – O(n)
    - sorted array – O(logn)
    - link list – O(n)
    - BT – O(n)
    - BST – O(n)
    - AVL – O(logn)

- So hashing is a technique where search time is independent of the number of items in which we are searching a data value.
- The basic idea is to use the key itself to find the address in the memory to make searching easy. For e.g. *to use phone number, roll no, Aadhar card, voter id or any other key and convert it into a smaller* practical *number (*but it must be modified so a great deal of space is not wasted*) and uses the small number as index in a table called hash table.*
- The values are then stored in **hash table**, By using that key you can access the element in **O(1)** time.

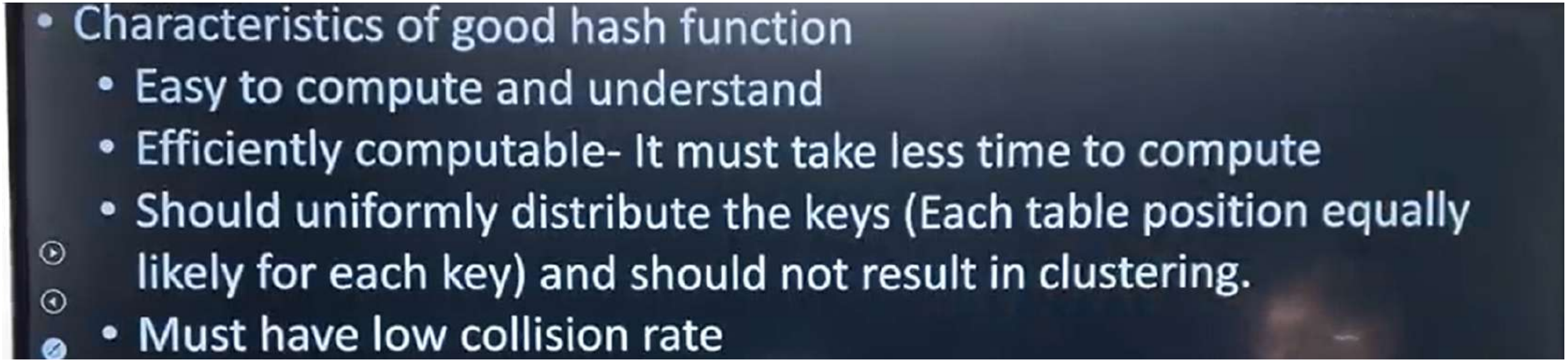


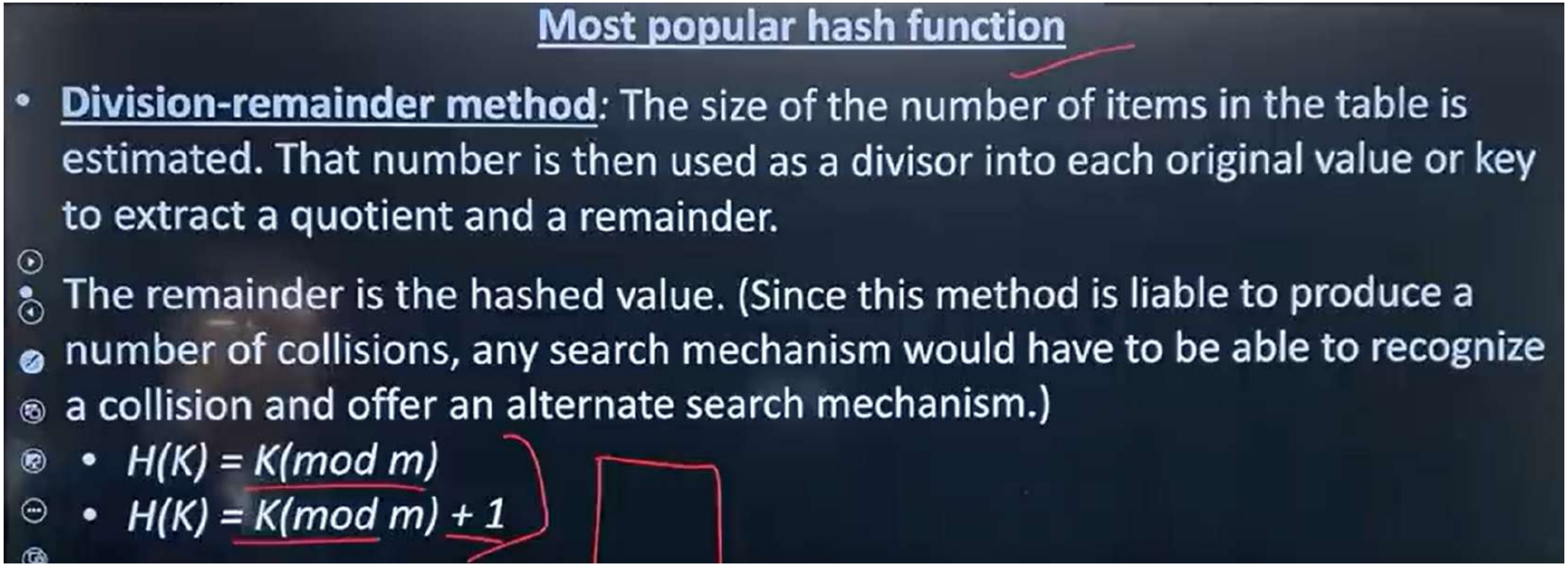


- This conversion called hash function which is from the set of K keys into the set of memory location L.
    - H: K→L
- In simple terms, a hash function maps a big number or string to a small integer that can be used as index in hash table. An array that stores pointers to records corresponding to our search key. The remaining entries can be nil.



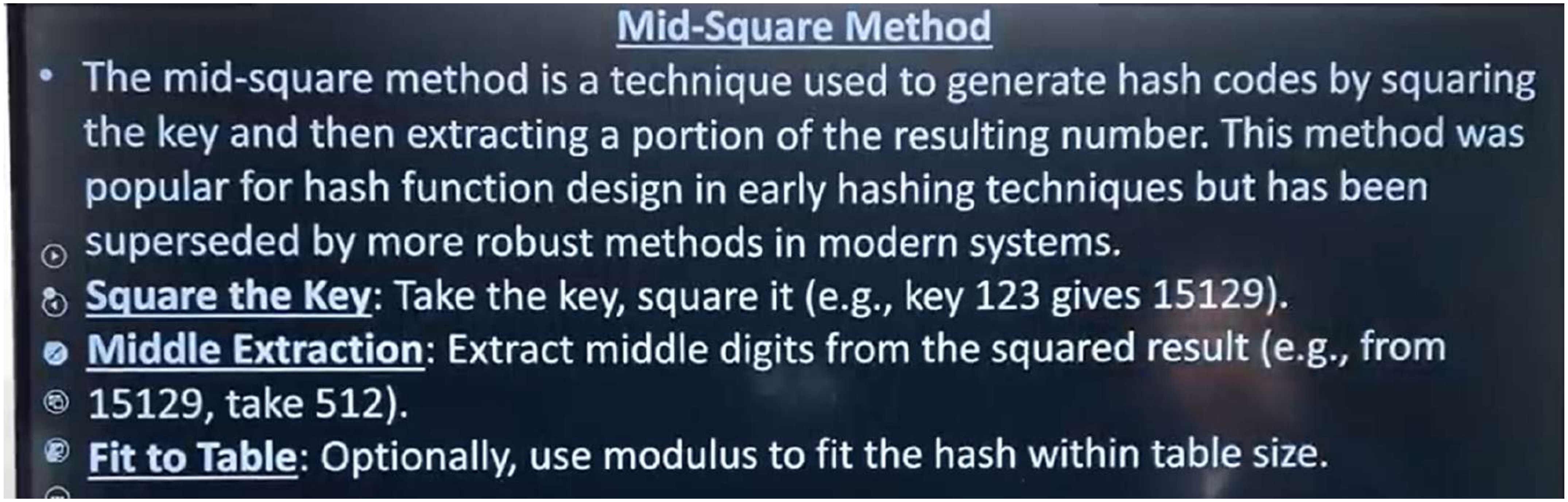
- **Collision**: - It is possible that two different set of keys K_1 and K_2 will yield the same hash address. This situation is called collision. The technique to resolve collision is called collision resolution.


- Characteristics of good hash function
  - Easy to compute and understand
  - Efficiently computable- It must take less time to compute
  - Should uniformly distribute the keys (Each table position equally likely for each key) and should not result in clustering.
  - Must have low collision rate



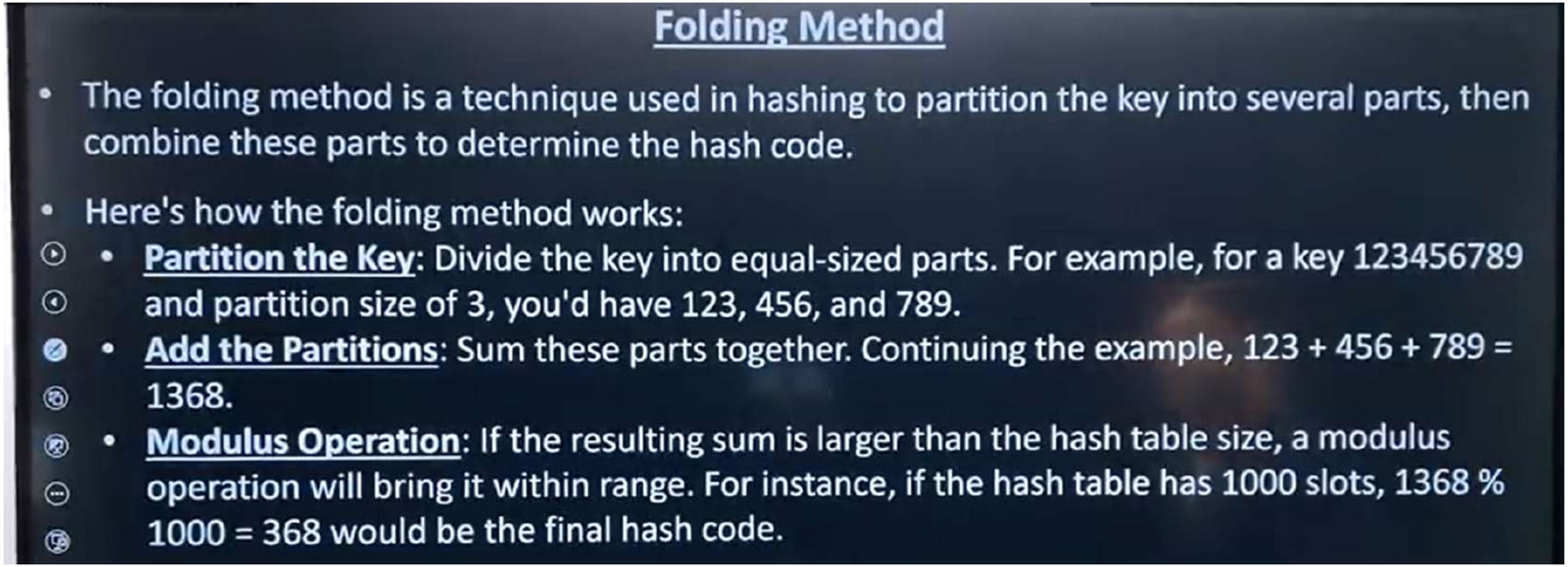
Most popular hash function

- **Division-remainder method**: The size of the number of items in the table is estimated. That number is then used as a divisor into each original value or key to extract a quotient and a remainder.
- The remainder is the hashed value. (Since this method is liable to produce a number of collisions, any search mechanism would have to be able to recognize a collision and offer an alternate search mechanism.)
  - $H(K) = K(\mathrm{mod}\ m)$
  - $H(K) = K(\mathrm{mod}\ m) + 1$



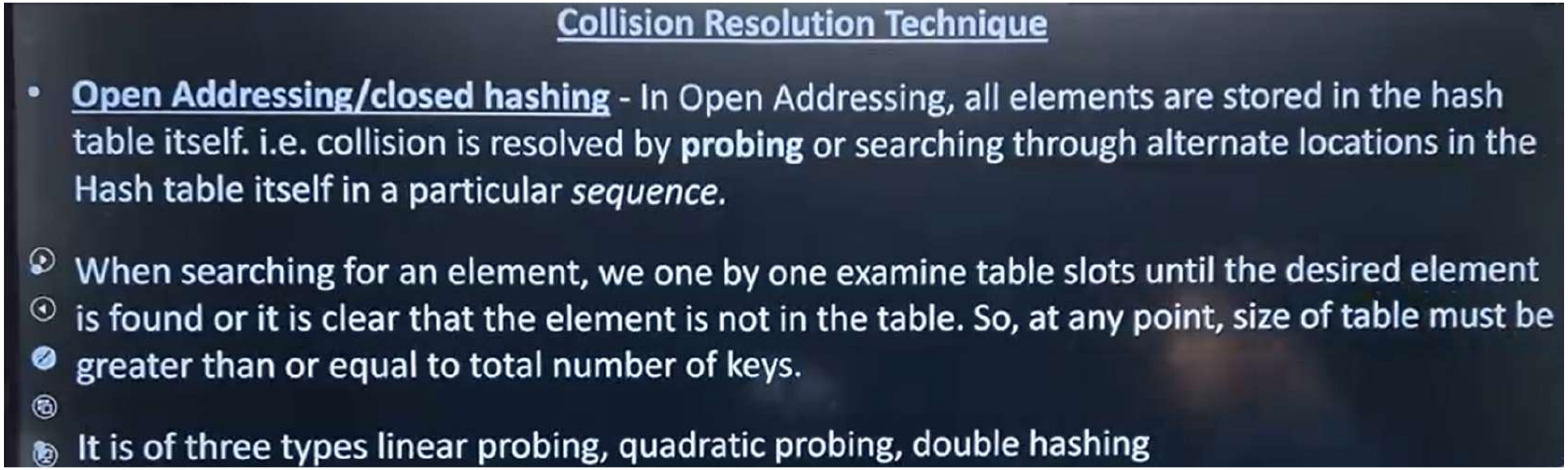
Mid-Square Method

- The mid-square method is a technique used to generate hash codes by squaring the key and then extracting a portion of the resulting number. This method was popular for hash function design in early hashing techniques but has been superseded by more robust methods in modern systems.
- **Square the Key**: Take the key, square it (e.g., key 123 gives 15129).
- **Middle Extraction**: Extract middle digits from the squared result (e.g., from 15129, take 512).
- **Fit to Table**: Optionally, use modulus to fit the hash within table size.



Folding Method

- The folding method is a technique used in hashing to partition the key into several parts, then combine these parts to determine the hash code.
- Here's how the folding method works:
  - **Partition the Key**: Divide the key into equal-sized parts. For example, for a key 123456789 and partition size of 3, you'd have 123, 456, and 789.
  - **Add the Partitions**: Sum these parts together. Continuing the example, 123 + 456 + 789 = 1368.
  - **Modulus Operation**: If the resulting sum is larger than the hash table size, a modulus operation will bring it within range. For instance, if the hash table has 1000 slots, 1368 % 1000 = 368 would be the final hash code.



Collision Resolution Technique

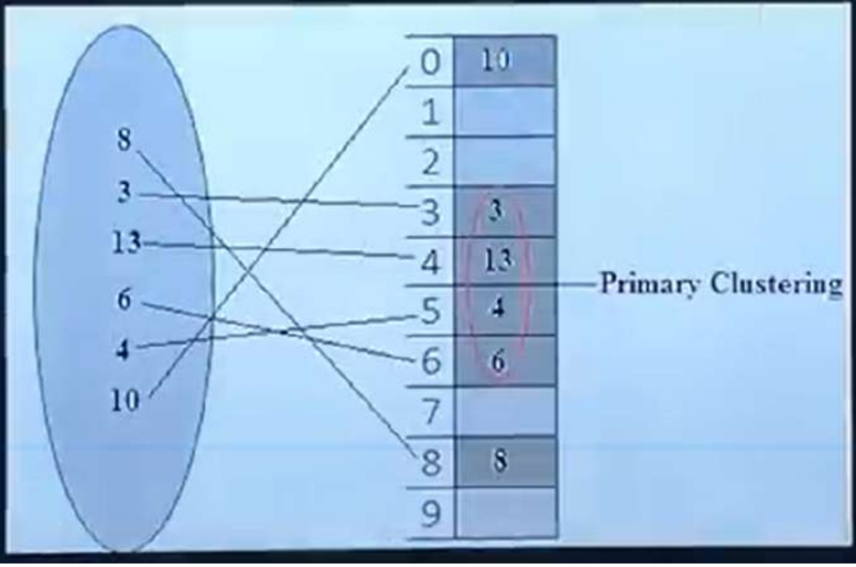
- **Open Addressing/closed hashing** - In Open Addressing, all elements are stored in the hash table itself. i.e. collision is resolved by **probing** or searching through alternate locations in the Hash table itself in a particular *sequence*.

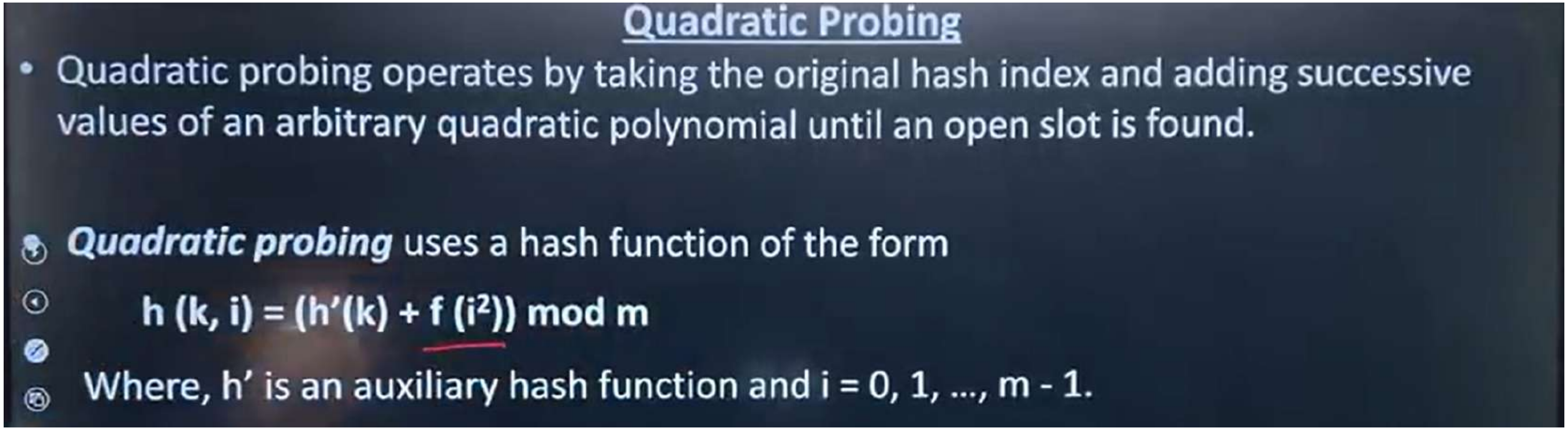
When searching for an element, we one by one examine table slots until the desired element is found or it is clear that the element is not in the table. So, at any point, size of table must be greater than or equal to total number of keys.

It is of three types linear probing, quadratic probing, double hashing

Example: Let us take the previous example, where the key value 13 was causing the collision at location 3.

- h (13) = (h(13) + 0) mod 10 = 3, since it is causing collision we consider the next value of i, i.e. i =1.
- h (13) = (h(13) + 1) mod 10 = 4, now at this location there is no collision so we place the value 13 at location 4.





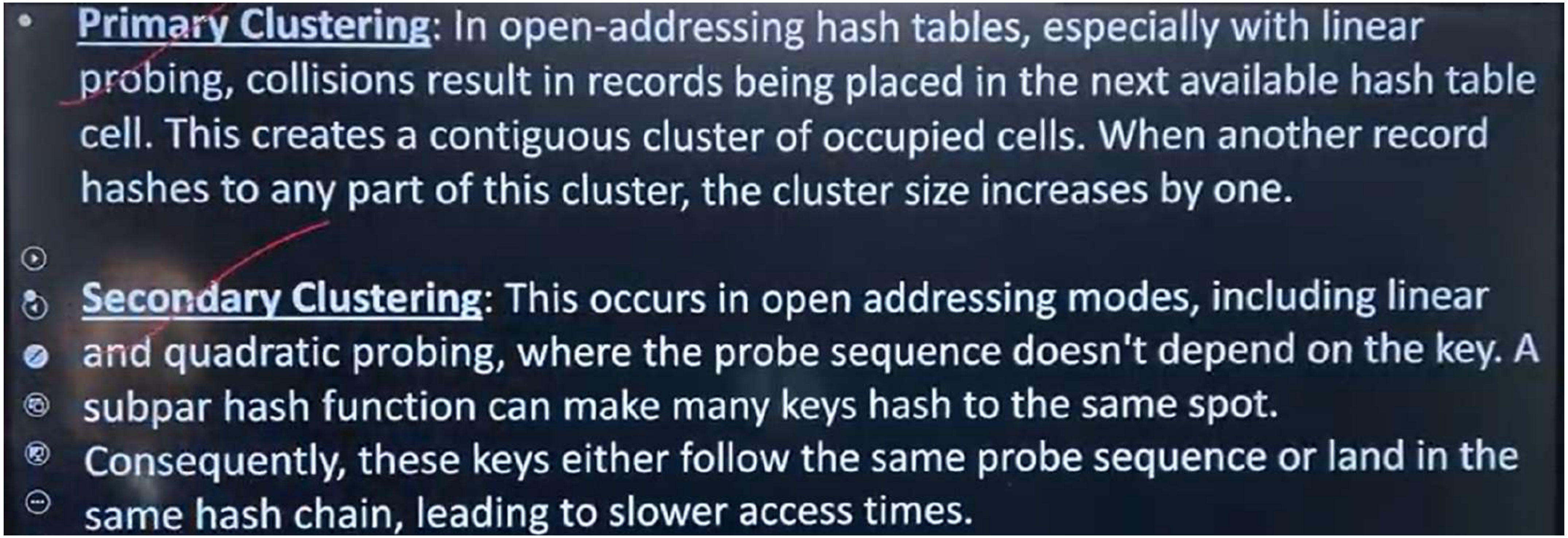
Quadratic Probing

- Quadratic probing operates by taking the original hash index and adding successive values of an arbitrary quadratic polynomial until an open slot is found.

- ***Quadratic probing*** uses a hash function of the form

$$h\ (k, i) = (h'(k) + \underline{f\ (i^2)})\ \mathrm{mod}\ m$$

Where, h' is an auxiliary hash function and i = 0, 1, ..., m - 1.

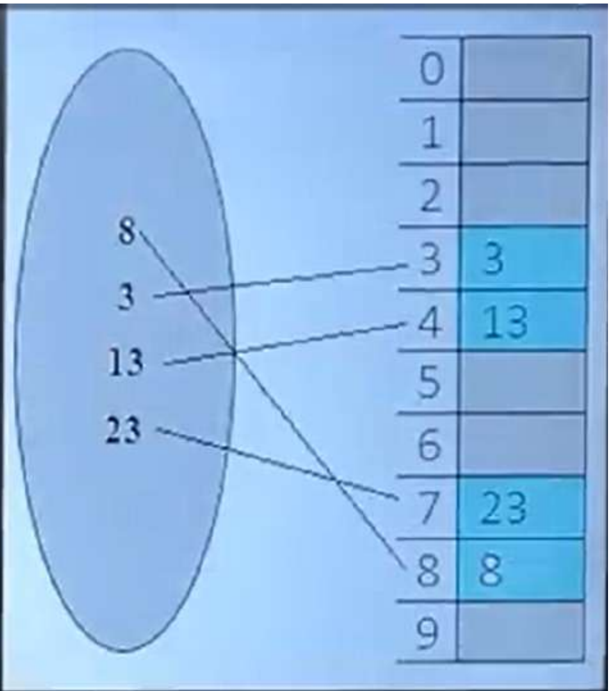


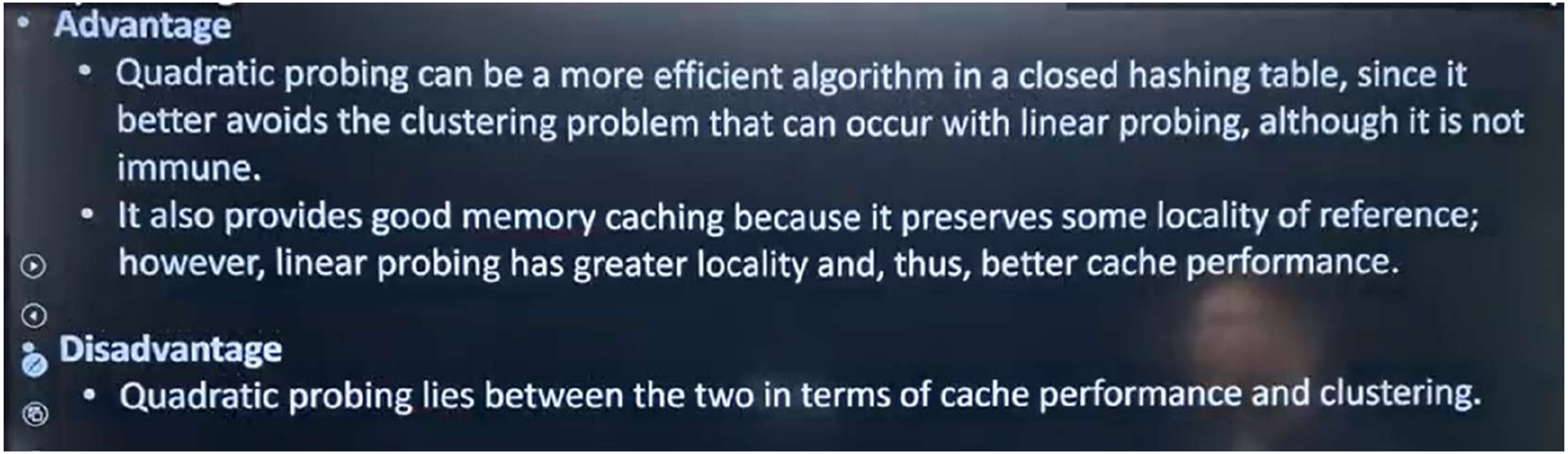
- **Primary Clustering**: In open-addressing hash tables, especially with linear probing, collisions result in records being placed in the next available hash table cell. This creates a contiguous cluster of occupied cells. When another record hashes to any part of this cluster, the cluster size increases by one.

- **Secondary Clustering**: This occurs in open addressing modes, including linear and quadratic probing, where the probe sequence doesn't depend on the key. A subpar hash function can make many keys hash to the same spot. Consequently, these keys either follow the same probe sequence or land in the same hash chain, leading to slower access times.

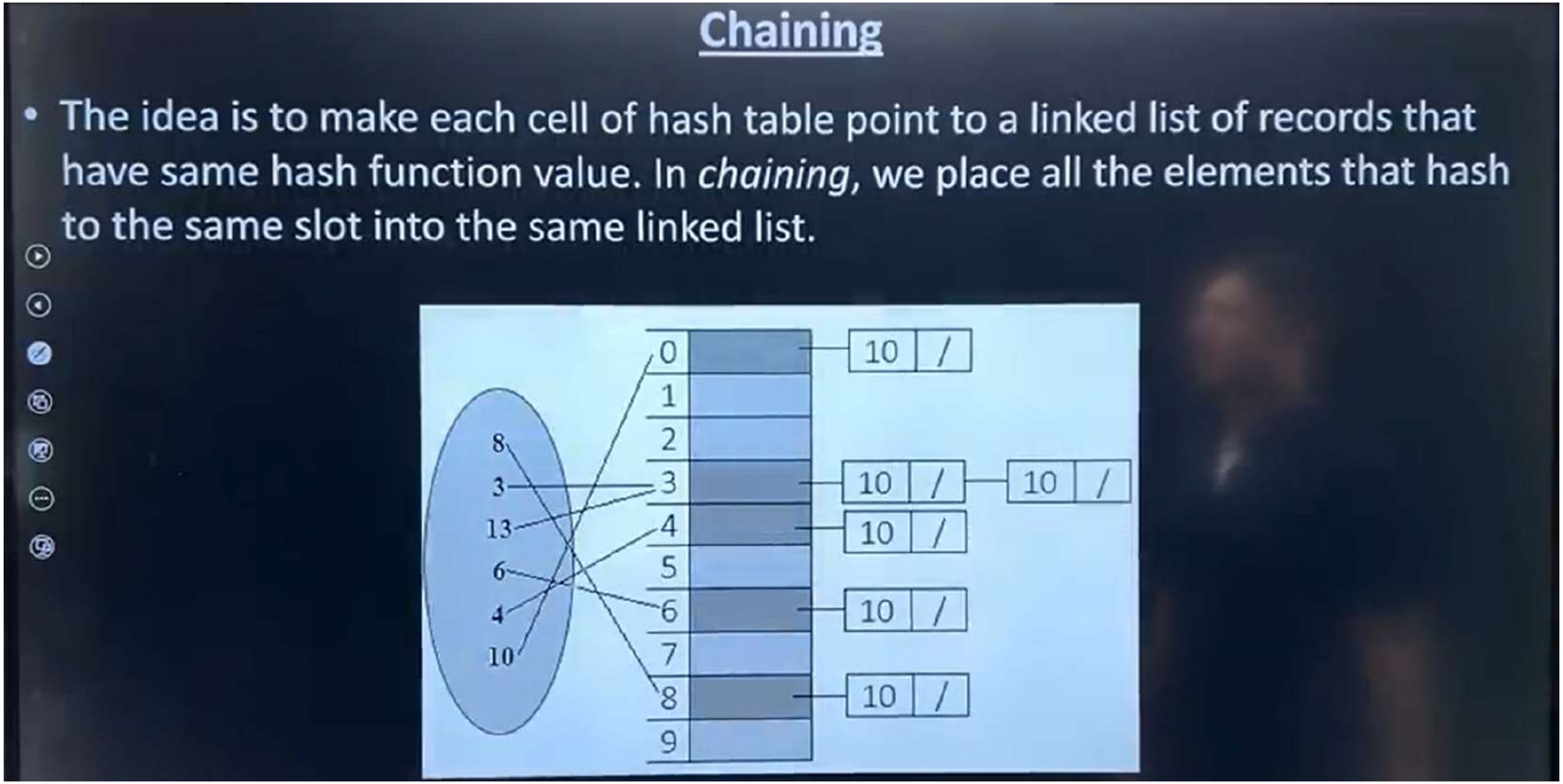
Example: Consider the key values 8, 3, 13, 23 and the hash table size is 10.

- 8 will be placed at: h (8) = [h (8) + f (0^2)] mod 10 = 8, so it gets placed at location 8.
- 3 will be placed at: h (3) = [h (3) + f (0^2)] mod 10 = 3, no collision, so it gets placed at location 3
- 13 will be placed at: h (13) = [h (13) + f (0^2)] mod 10 = 3, collision occurred, so we increase the value of i.
  - h (13) = [h (13) + f (1^2)] mod 10 = 4, no collision, so it gets placed at location 4.
- 23 will be placed at: h (23) = [h (23) + f (0^2)] mod 10 = 3, collision occurred, so we increase the value of i.
  - h (23) = [h (23) + f (1^2)] mod 10 = 4, again collision occurred, so we increase the value of i.
  - h (23) = [h (23) + f (2^2)] mod 10 = 3 + 4 = 7, no collision occurred, so it gets placed at location 7.
- Quadratic probing avoids clustering of elements and thus improves the searching time.



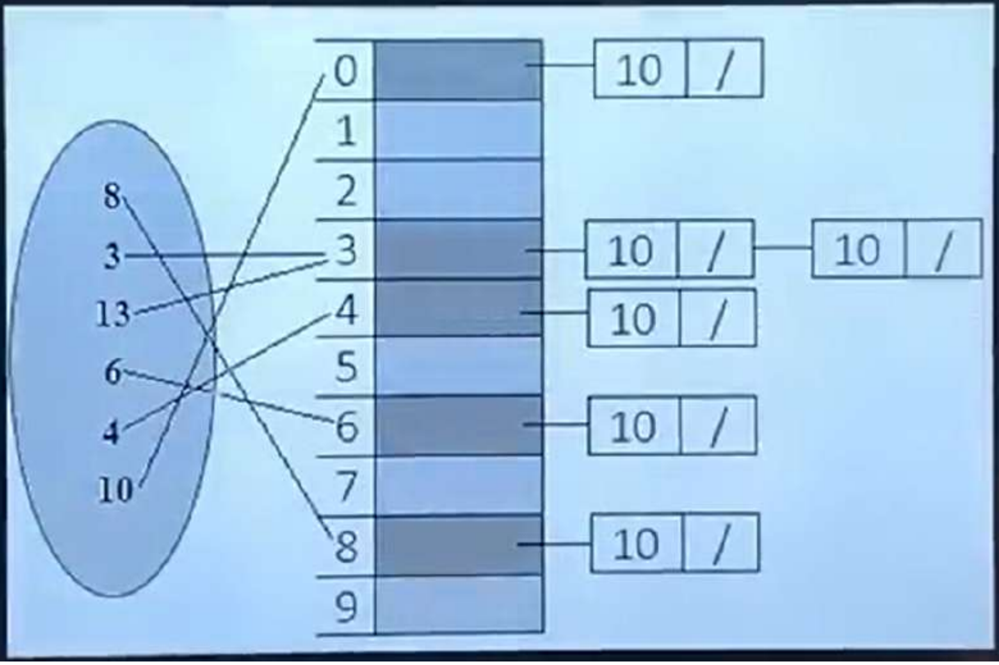


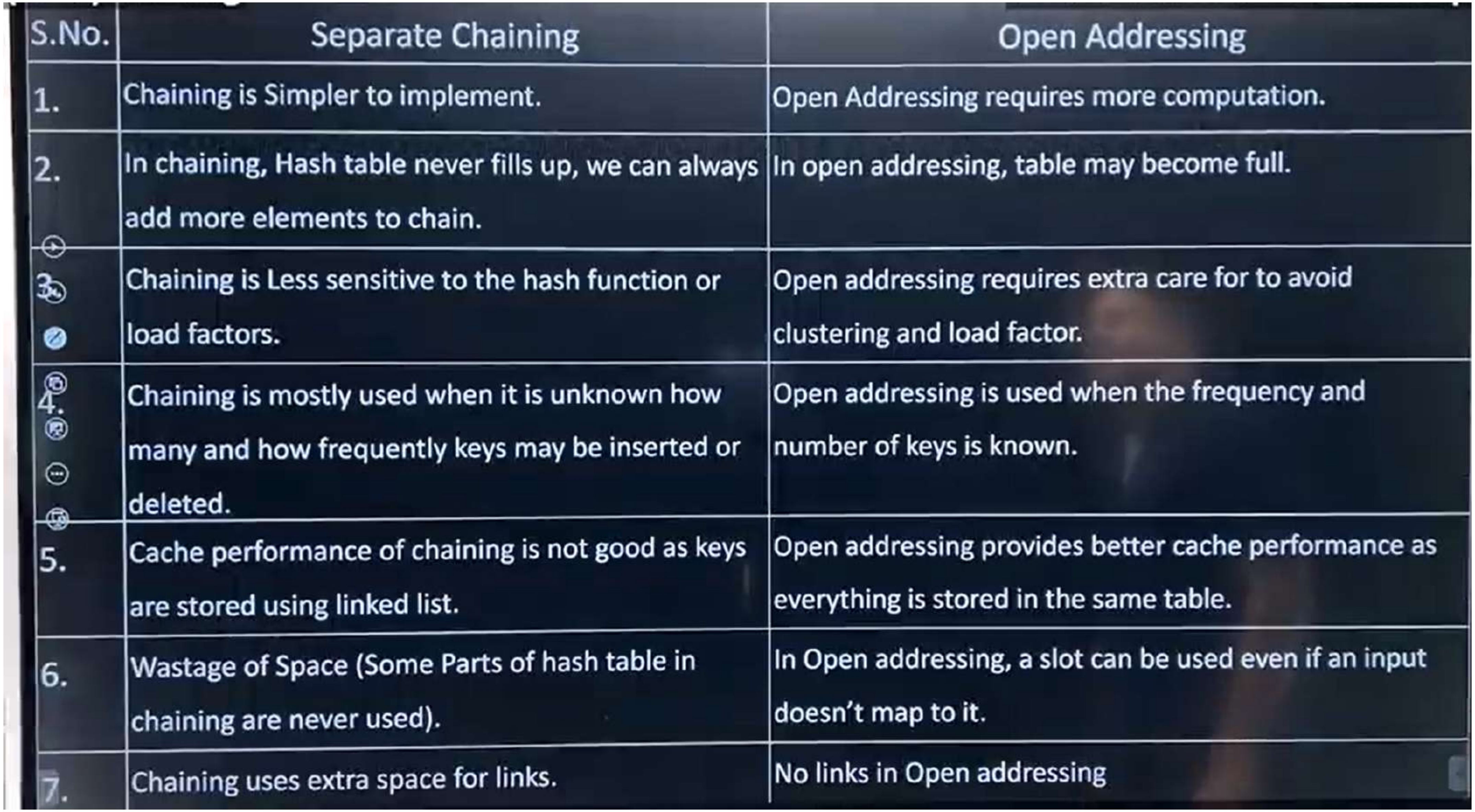
- **Advantage**
  - Quadratic probing can be a more efficient algorithm in a closed hashing table, since it better avoids the clustering problem that can occur with linear probing, although it is not immune.
  - It also provides good memory caching because it preserves some locality of reference; however, linear probing has greater locality and, thus, better cache performance.
- **Disadvantage**
  - Quadratic probing lies between the two in terms of cache performance and clustering.



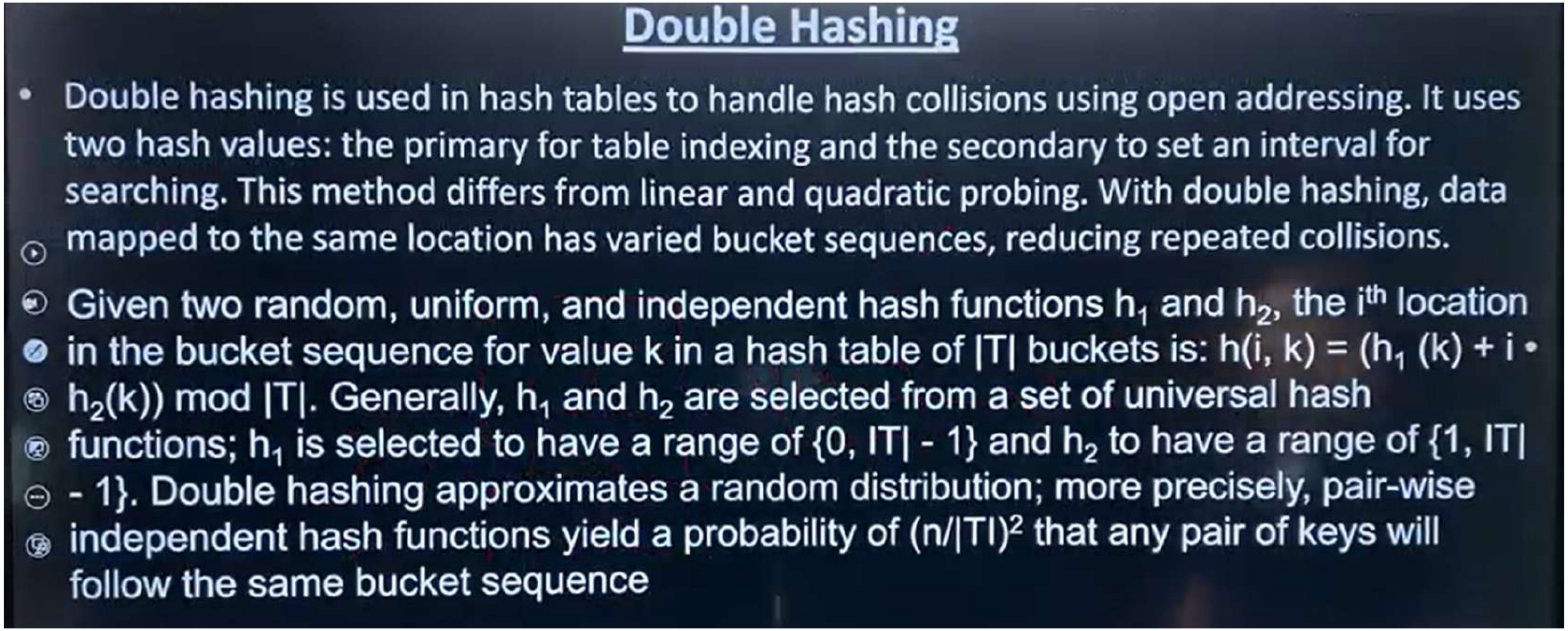
Chaining

- The idea is to make each cell of hash table point to a linked list of records that have same hash function value. In *chaining*, we place all the elements that hash to the same slot into the same linked list.





S.No.	Separate Chaining	Open Addressing
1.	Chaining is Simpler to implement.	Open Addressing requires more computation.
2.	In chaining, Hash table never fills up, we can always add more elements to chain.	In open addressing, table may become full.
3.	Chaining is Less sensitive to the hash function or load factors.	Open addressing requires extra care for to avoid clustering and load factor.
4.	Chaining is mostly used when it is unknown how many and how frequently keys may be inserted or deleted.	Open addressing is used when the frequency and number of keys is known.
5.	Cache performance of chaining is not good as keys are stored using linked list.	Open addressing provides better cache performance as everything is stored in the same table.
6.	Wastage of Space (Some Parts of hash table in chaining are never used).	In Open addressing, a slot can be used even if an input doesn't map to it.
7.	Chaining uses extra space for links.	No links in Open addressing



Double Hashing

- Double hashing is used in hash tables to handle hash collisions using open addressing. It uses two hash values: the primary for table indexing and the secondary to set an interval for searching. This method differs from linear and quadratic probing. With double hashing, data mapped to the same location has varied bucket sequences, reducing repeated collisions.

Given two random, uniform, and independent hash functions h_1 and h_2, the ith location in the bucket sequence for value k in a hash table of |T| buckets is: $h(i, k) = (h_1(k) + i \cdot h_2(k)) \bmod |T|$. Generally, h_1 and h_2 are selected from a set of universal hash functions; h_1 is selected to have a range of $\{0, IT| - 1\}$ and h_2 to have a range of $\{1, IT| - 1\}$. Double hashing approximates a random distribution; more precisely, pair-wise independent hash functions yield a probability of $(n/|TI)^2$ that any pair of keys will follow the same bucket sequence