Hashing

By :
Dr. Rinkle Rani
Associate Professor, CSED
TIET, Patiala

What is Hashing in DBMS?

Å In DBMS, hashing is a technique to directly search the location of desired data on the disk without using index structure.

Å Hashing method is used to index and retrieve items in a database as it is faster to search that specific item using the shorter hashed key instead of using its original value.

Å Data is stored in the form of data blocks whose address is generated by applying a hash function in the memory location where these records are stored known as a **data block or data bucket**.

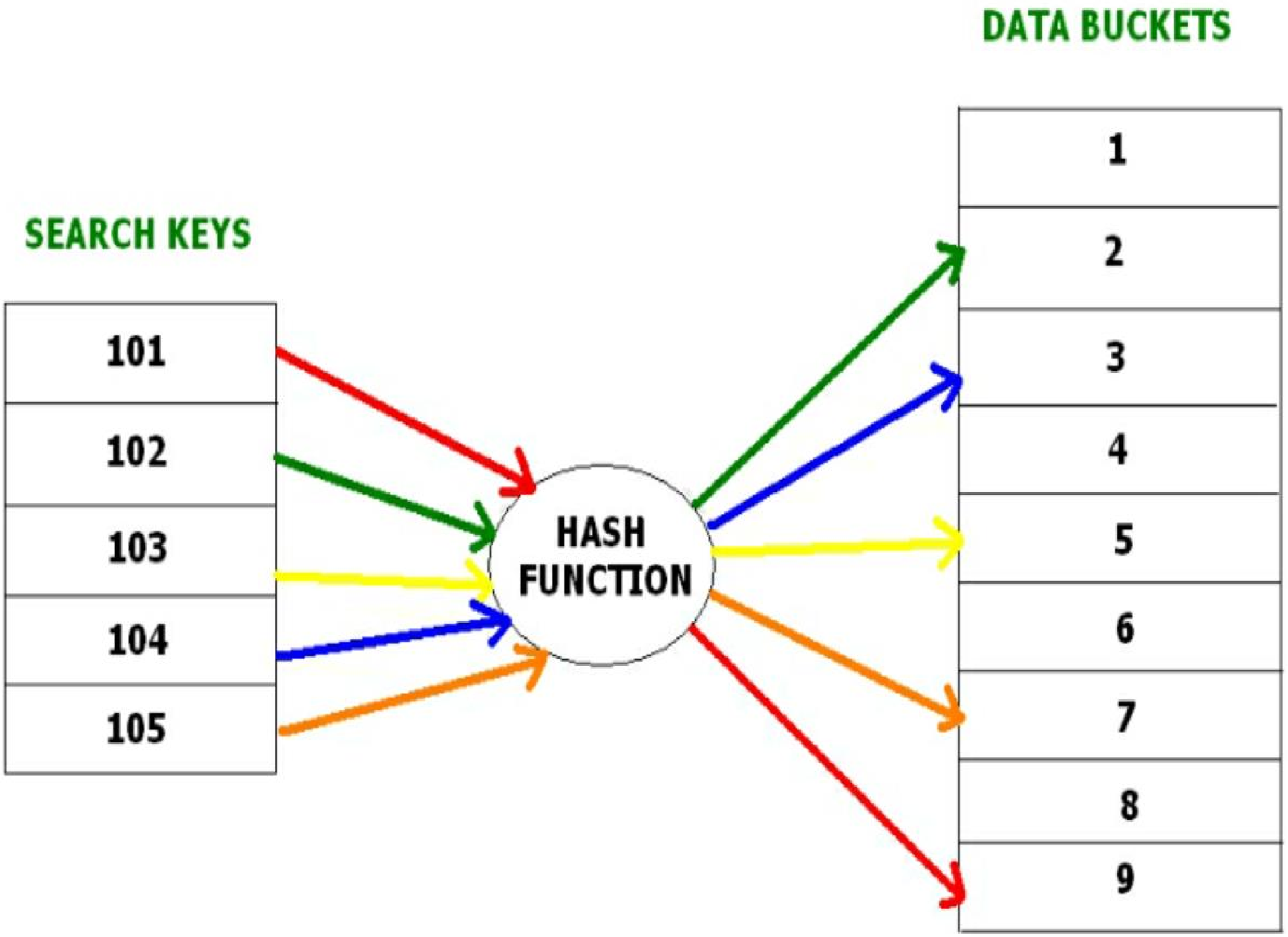
Important Terms using in Hashing

Data bucket – Data buckets are memory locations where the records are stored. It is also known as Unit Of Storage.

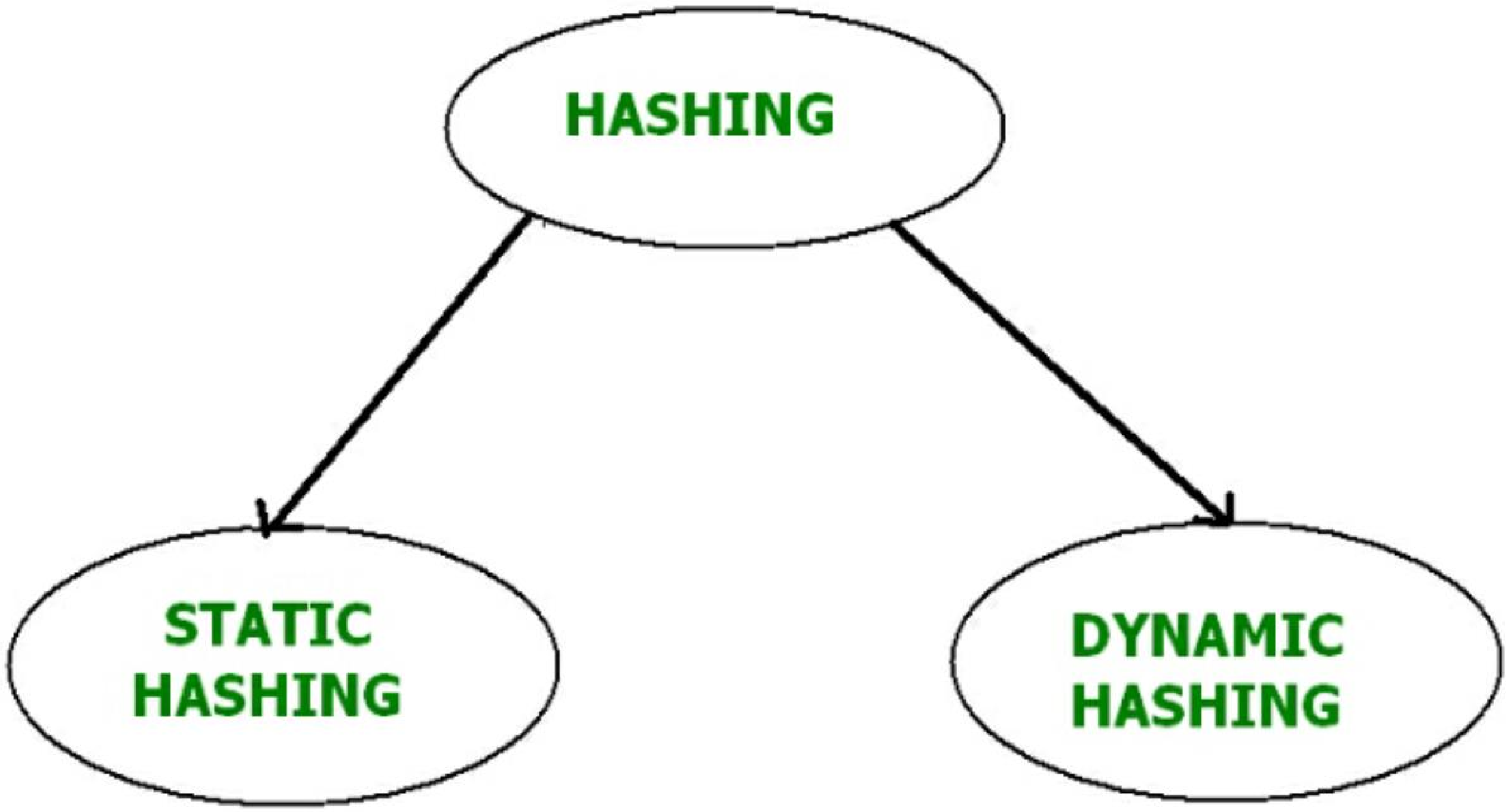
Key: A DBMS key is an attribute or set of an attribute which helps you to identify a row(tuple) in a relation(table). This allows you to find the relationship between two tables.

Hash function: A hash function, is a mapping function which maps all the set of search keys to the address where actual records are placed.

Hash index – It is an address of the data block. A hash function could be a simple mathematical function to even a complex mathematical function.


DATA BUCKETS
SEARCH KEYS
101
102
103
104
105
HASH FUNCTION
1
2
3
4
5
6
7
8
9

Hashing is further divided into two sub categories :



Static Hashing –

In static hashing, when a search-key value is provided, the hash function always computes the same address.

For example, if we want to generate address for STUDENT_ID = 76 using mod (5) hash function, it always result in the same bucket address 4.

There will not be any changes to the bucket address here. Hence number of data buckets in the memory for this static hashing remains constant throughout.

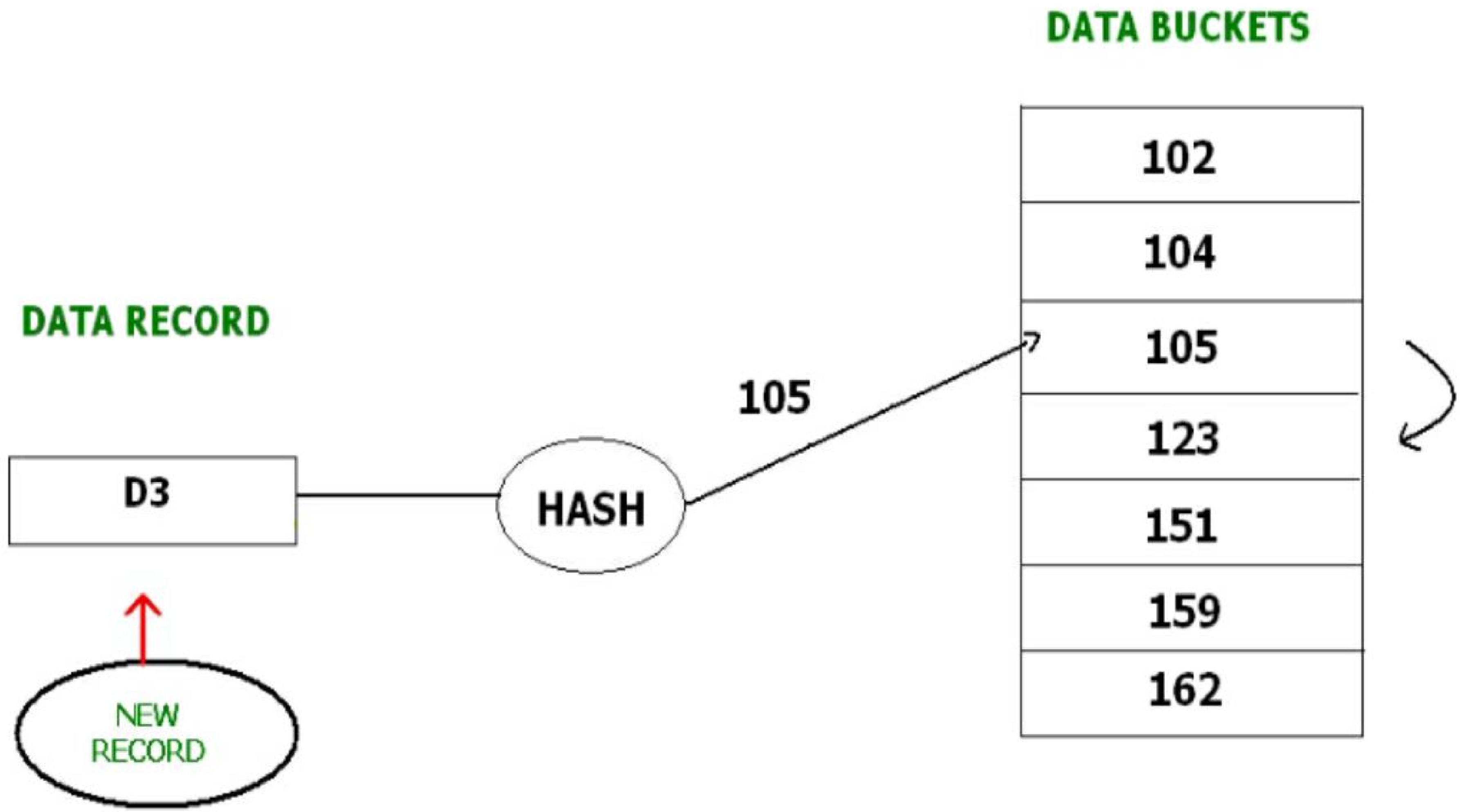
Now, If we want to insert some new records into the file But the data bucket address generated by the hash function is not empty or the data already exists in that address. This becomes a critical situation to handle. This situation in the static hashing is called **bucket overflow**.

How will we insert data in this case? There are several methods provided to overcome this situation. Some commonly used methods are discussed below:

Å Open hashing
Å Close hashing.

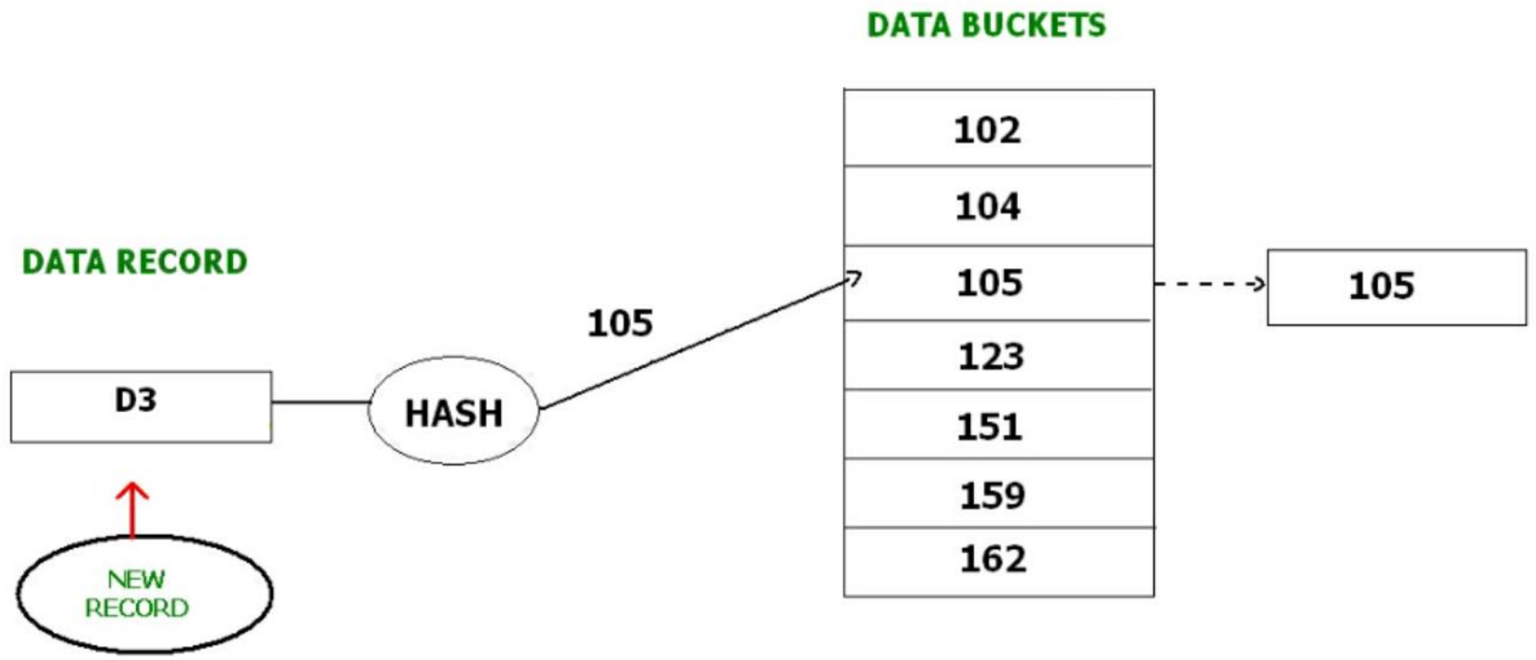
Open Hashing –

In Open hashing method, next available data block is used to enter the new record, instead of overwriting older one.
This method is also called linear probing.
For example, D3 is a new record which needs to be inserted , the hash function generates address as 105. But it is already full. So the system searches next available data bucket, 123 and assigns D3 to it


DATA BUCKETS
102
104
105
123
151
159
162
DATA RECORD
D3
HASH
105
NEW RECORD

Close Hashing

In the close hashing method, when buckets are full, a new bucket is allocated for the same hash and result are linked after the previous one.



Quadratic probing :

Quadratic probing is very much similar to open hashing or linear probing. Here, The only difference between old and new bucket is linear. Quadratic function is used to determine the new bucket address.

Double Hashing :

Double Hashing is another method similar to linear probing. Here the difference is fixed as in linear probing, but this fixed difference is calculated by using another hash function. That’s why the name is double hashing.

Dynamic Hashing –

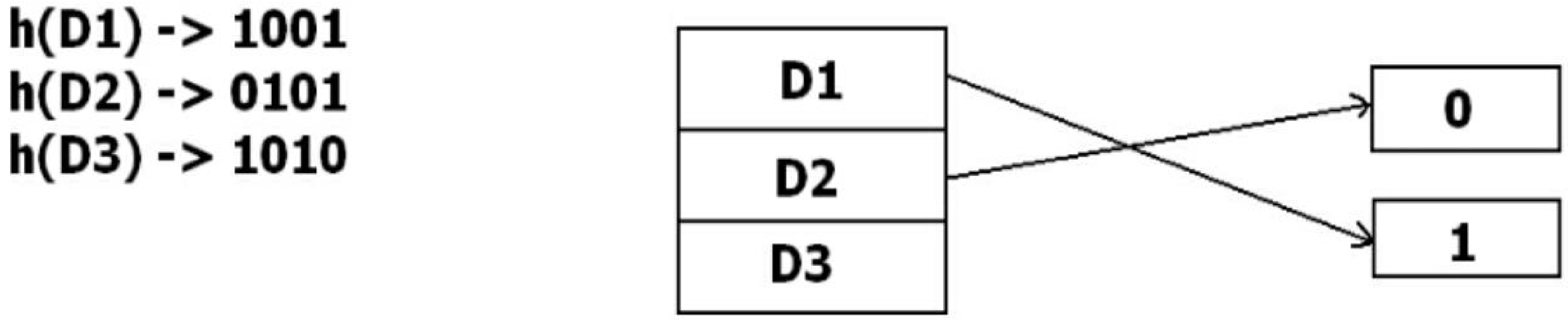
Å The drawback of static hashing is that that it does not expand or shrink dynamically as the size of the database grows or shrinks.

Å In Dynamic hashing, data buckets grows or shrinks (added or removed dynamically) as the records increases or decreases.

Å Dynamic hashing is also known as extended hashing.

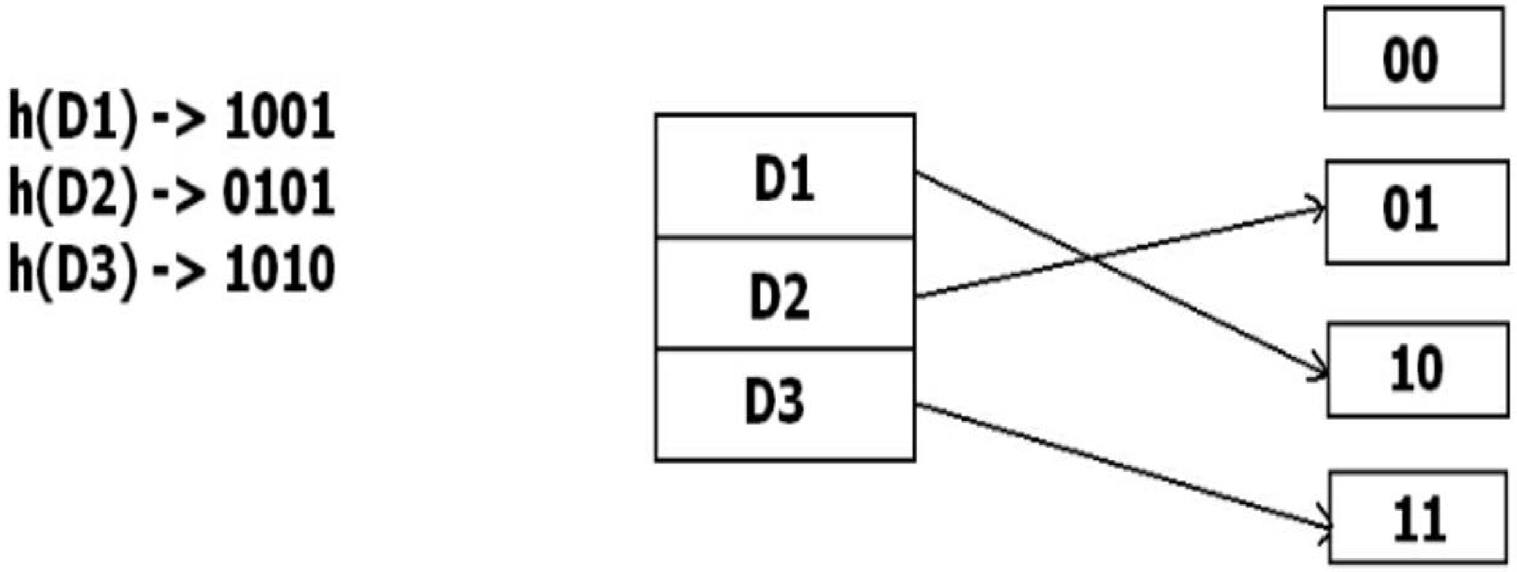
Å In dynamic hashing, the hash function is made to produce a large number of values.

For Example, there are three data records D1, D2 and D3 . The hash function generates three addresses 1001, 0101 and 1010 respectively. This method of storing considers only part of this address – especially only first one bit to store the data. So it tries to load three of them at address 0 and 1.



h(D1) -> 1001
h(D2) -> 0101
h(D3) -> 1010

But the problem is that No bucket address is remaining for D3. The bucket has to grow dynamically to accommodate D3. So it changes the address have 2 bits rather than 1 bit, and then it updates the existing data to have 2 bit address. Then it tries to accommodate D3.


h(D1) -> 1001
h(D2) -> 0101
h(D3) -> 1010
D1
D2
D3
00
01
10
11