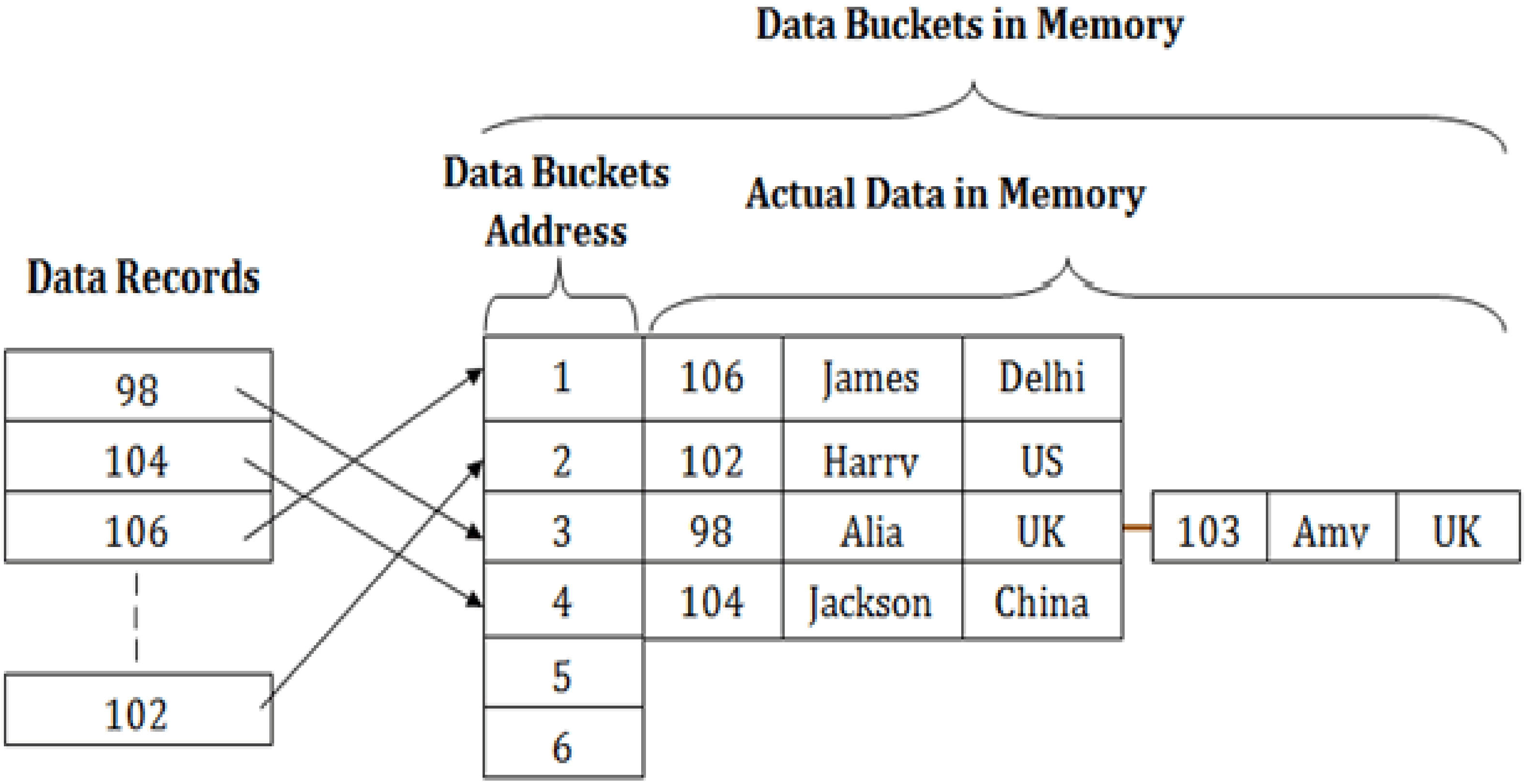
Hashing

Hashing

- Databases in organizations are usually very big / huge.
- So, to find the desired records/data you need to search all the index values which is very inefficient.
- This is where hashing comes into picture.
- Hashing technique allows us to calculate the direct location of a data record on the disk without using index structure.
- Data is stored at the data buckets whose address is generated by using the hashing function.

Hashing



Data Buckets in Memory

Data Buckets Address

Actual Data in Memory

Data Records

98

104

106

102

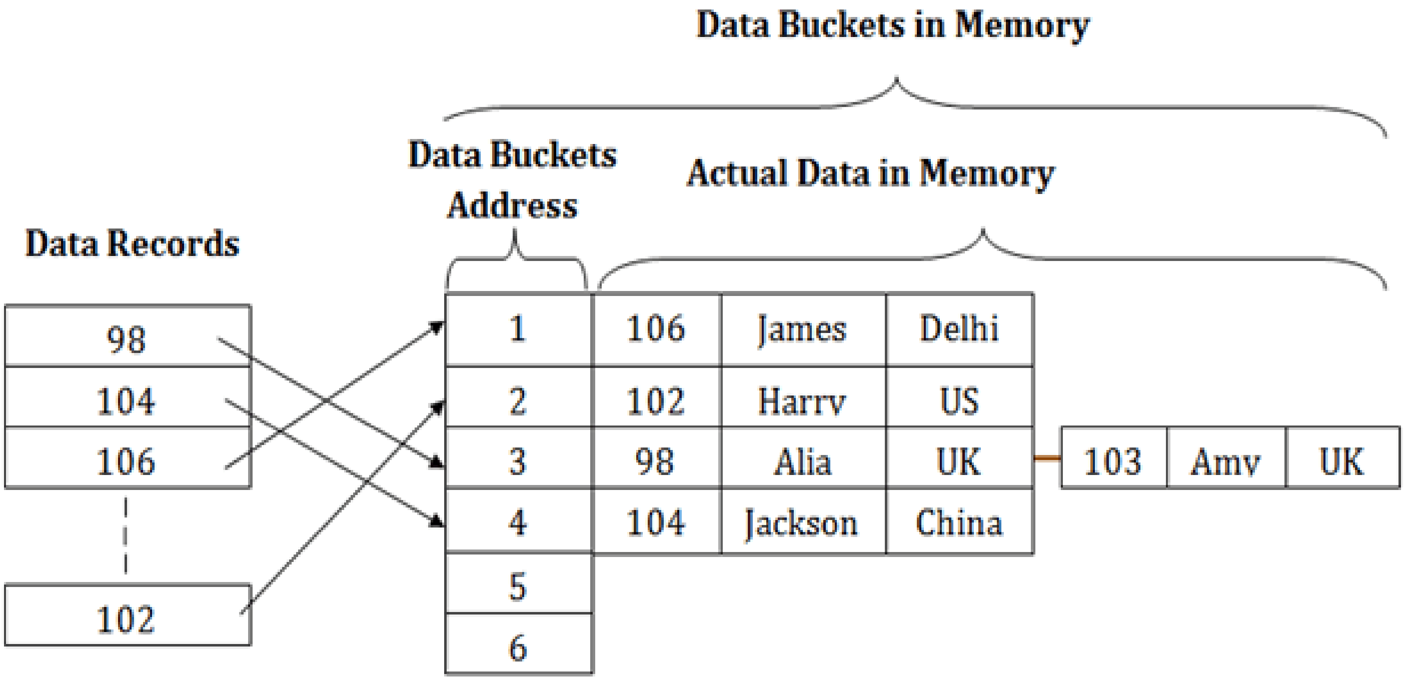
1	106	James	Delhi			
2	102	Harry	US			
3	98	Alia	UK	103	Amy	UK
4	104	Jackson	China			
5						
6						

Hash Function

- A hash function can choose any of the column value to generate the address.
- Most of the time, the hash function uses the primary key to generate the address of the data bucket.
- A hash function is a simple mathematical function (mod , cos, sin, exponential, etc) to any complex mathematical function.
- The key value (v) used to generate the hash function value (h(v)).

Hashing

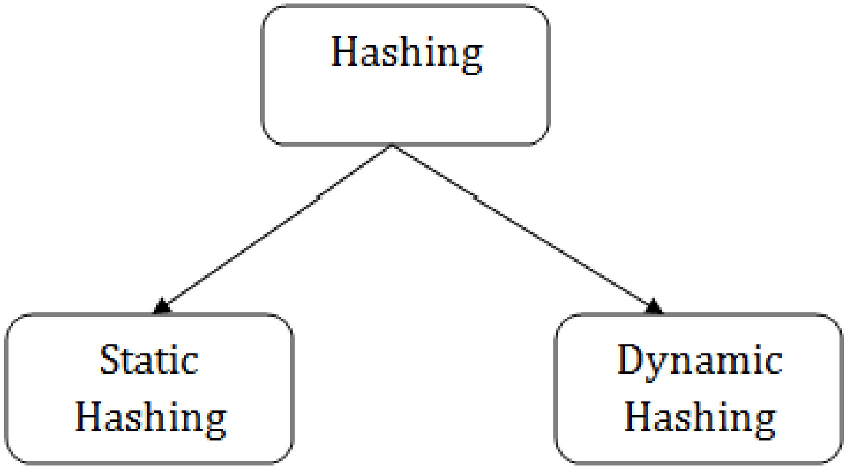
- Suppose we have mod (5) hash function to determine the address of the data block.
- In this case, it applies mod (5) hash function on the primary keys and generates 3, 4, 1, … and 2 respectively, and records are stored in those data block addresses.



1	106	James	Delhi			
2	102	Harry	US			
3	98	Alia	UK	103	Amy	UK
4	104	Jackson	China			
5						
6						

Types of hashing

- There are mainly 2 types of hashing.



Static Hashing

- In this type of hashing, the resultant data bucket address will always be the same.
- For Example, if we generate an address for EMP_ID =103 using the hash function mod (5) then it will always result in same bucket address 3.
- The number of data buckets in memory remains constant throughout.
- In this our example, we will have five data buckets in the memory used to store the data.

Operations of Static Hashing

- Searching a record
- Insert a record
- Delete a record
- Update a record

Collision Problem

- If we want to insert some new record into the file but the address of a data bucket generated by the hash function is not empty, or data already exists in that address (i.e. bucket is full).
- This situation in the static hashing is known as **bucket overflow**.
- What to do now????
- To overcome this situation, there are various methods:
  - Open hashing or Open Addressing
  - Closed Hashing or Separate Chaining

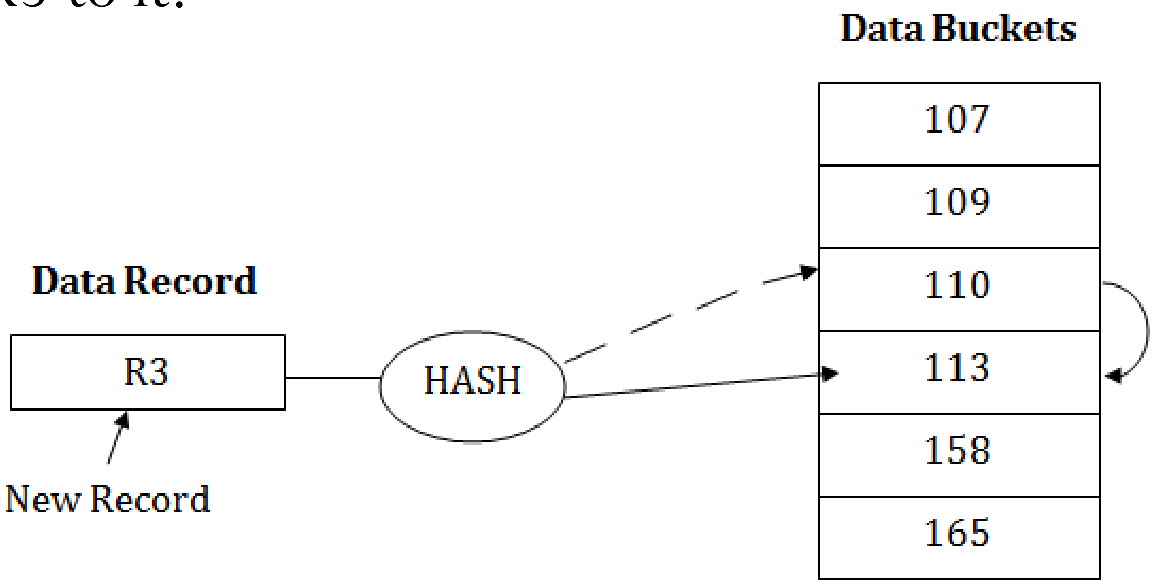
Open Hashing

- In this method, when a hash function generates an address at which data is already stored, then the next bucket will be allocated to it.
- This mechanism is called as **Linear Probing**.
- There are other mechanisms as well like quadratic probing and double hashing.

More details read : https://www.geeksforgeeks.org/hashing-set-3-open-addressing/

Open Hashing

- Suppose R3 is a new record which needs to be inserted, the hash function generates address as 110 for R3.
- But the generated address is already full.
- So the system searches next available data bucket, 113 and assigns R3 to it.



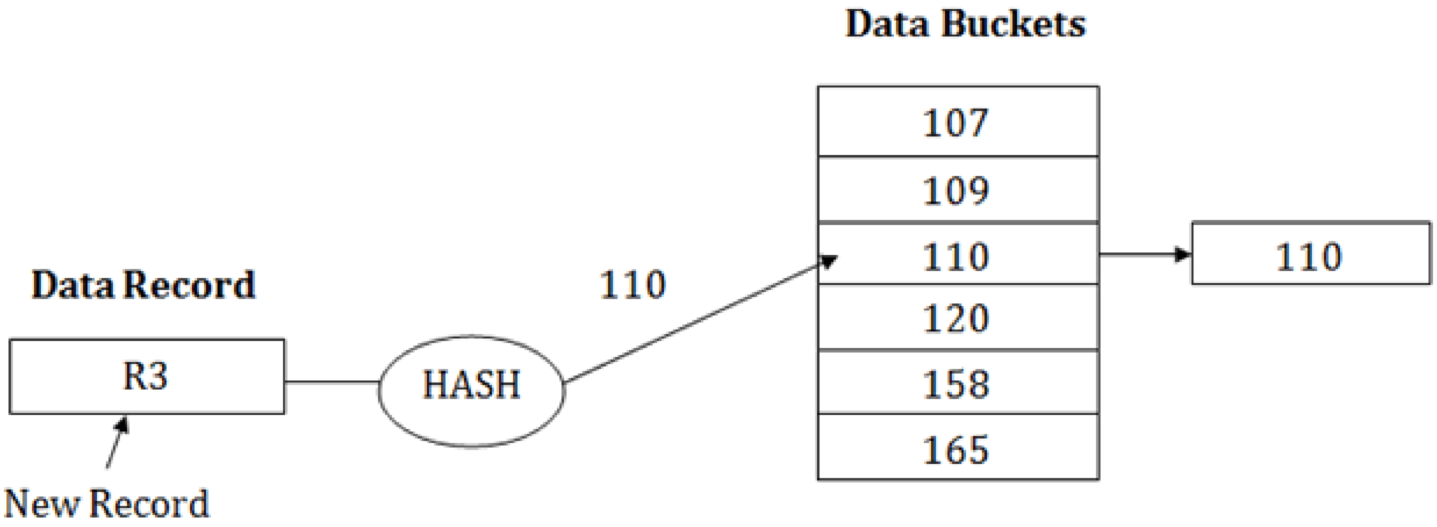
Closed Hashing/Separate Chaining

- In this method, when buckets are full, then a new data bucket is allocated for the same hash result and is linked after the previous one.
- The idea is to make each cell of hash table point to a linked list of records that have same hash function value.
- This mechanism is known as **Overflow chaining**.

More Details Read: https://www.geeksforgeeks.org/hashing-set-2-separate-chaining/

Closed Hashing/Separate Chaining

- Suppose R3 is a new address which needs to be inserted into the table, the hash function generates address as 110 for it.
- But this bucket is full to store the new data.
- In this case, a new bucket is inserted at the end of 110 buckets and is linked to it.



Dynamic Hashing

- It is used to overcome the problems of static hashing like bucket overflow.
- In this method, data buckets grow or shrink as the records increases or decreases.
- This method is also known as **Extendable hashing method**.
- This method makes hashing dynamic, i.e., it allows insertion or deletion without resulting in poor performance.

Searching a key

- First, calculate the hash address of the key.
- Check how many bits are used in the directory (level of indirection), and these bits are called as ***i***.
- Take the least significant ***i*** bits of the hash address. This gives an index of the directory.
- Now using the index, go to the directory and find bucket address where the record might be.

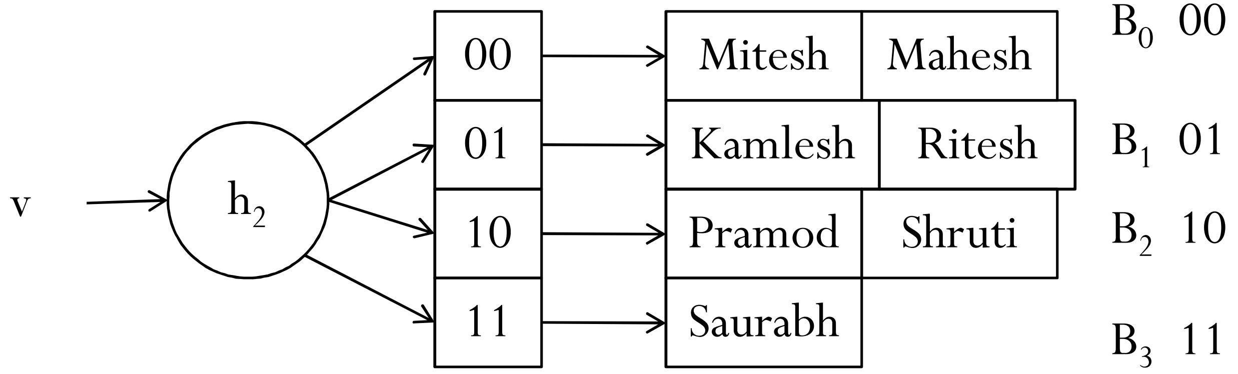
Inserting a New Record

- Firstly, you have to follow the same procedure for retrieval, ending up in some bucket.
- If there is still space in that bucket, then place the record in it.
- If the bucket is full, then we will split the bucket and redistribute the records.

Working of Dynamic Hashing

v	**h(v)**
Pramod	110**10**
Mitesh	000**00**
Shruti	111**10**
Mahesh	000**00**
Kamlesh	010**01**
Ritesh	101**01**
Saurabh	101**11**

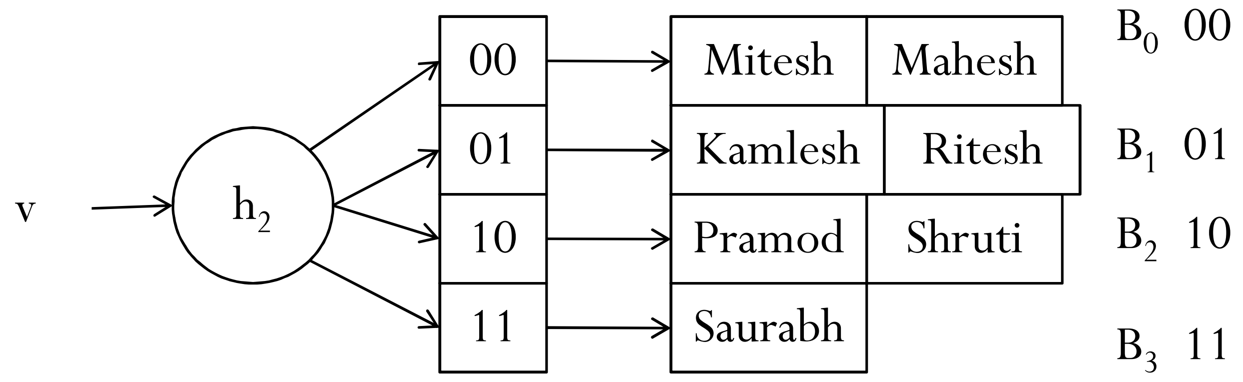
Suppose Bucket size = 2



$h_k(v) = h(v) \bmod 2^k$ → use last k-bits of h(v)

New Insertion

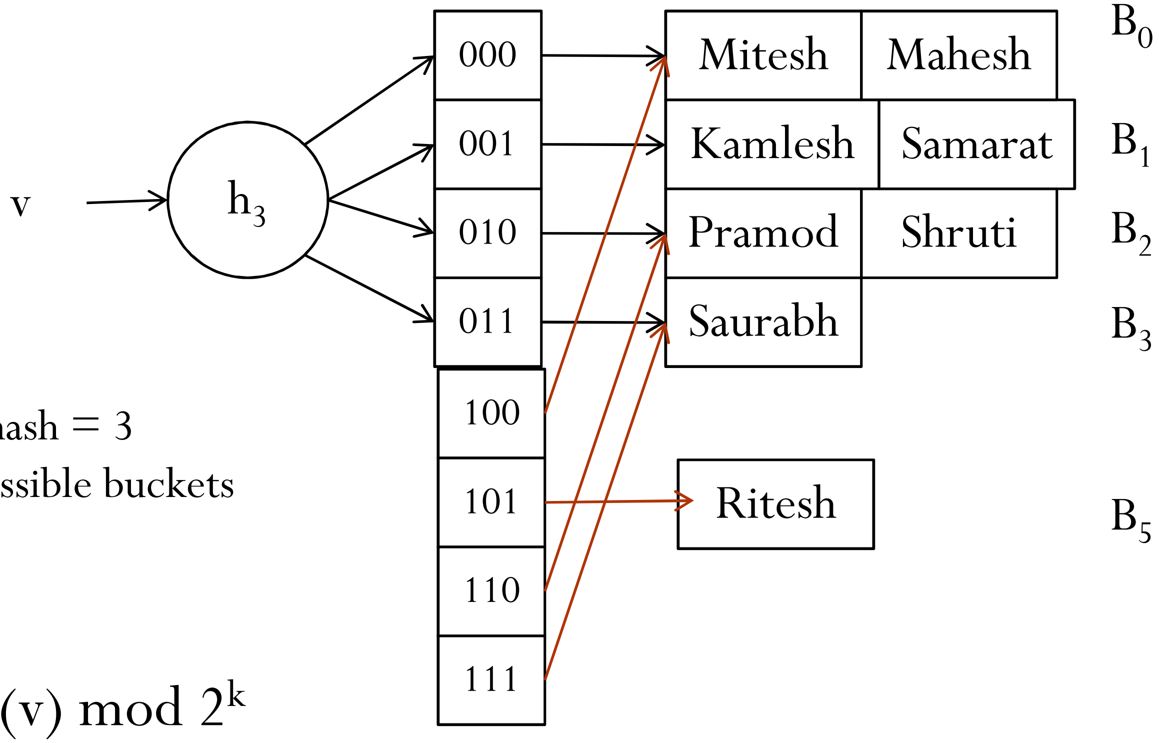
- Now Insert "Smarat" where h(samarat) = 10001
- This causes overflow in B_1.
- So we need to switch to h_3.



$h_k(v) = h(v) \bmod 2^k$ → use last k-bits of h(v)

New Insertion

- Concatenate copy of old directory to new.
- Split overflow bucket B into B & B`, dividing the entries.
- Current hash identifies the current hash function.



Current hash = 3

$2^3 = 8$ possible buckets

k=3

$h_k(v) = h(v) \bmod 2^k$