Hashing

- By

Jyoti Tryambake

Hashing

- In a huge database structure, it is very inefficient to search all the index values and reach the desired data.
  - Hashing technique is used to calculate the direct location of a data record on the disk without using index structure.
- In this technique, data is stored at the data blocks whose address is generated by using the hashing function.
- The memory location where these records are stored is known as **data bucket or data blocks**.

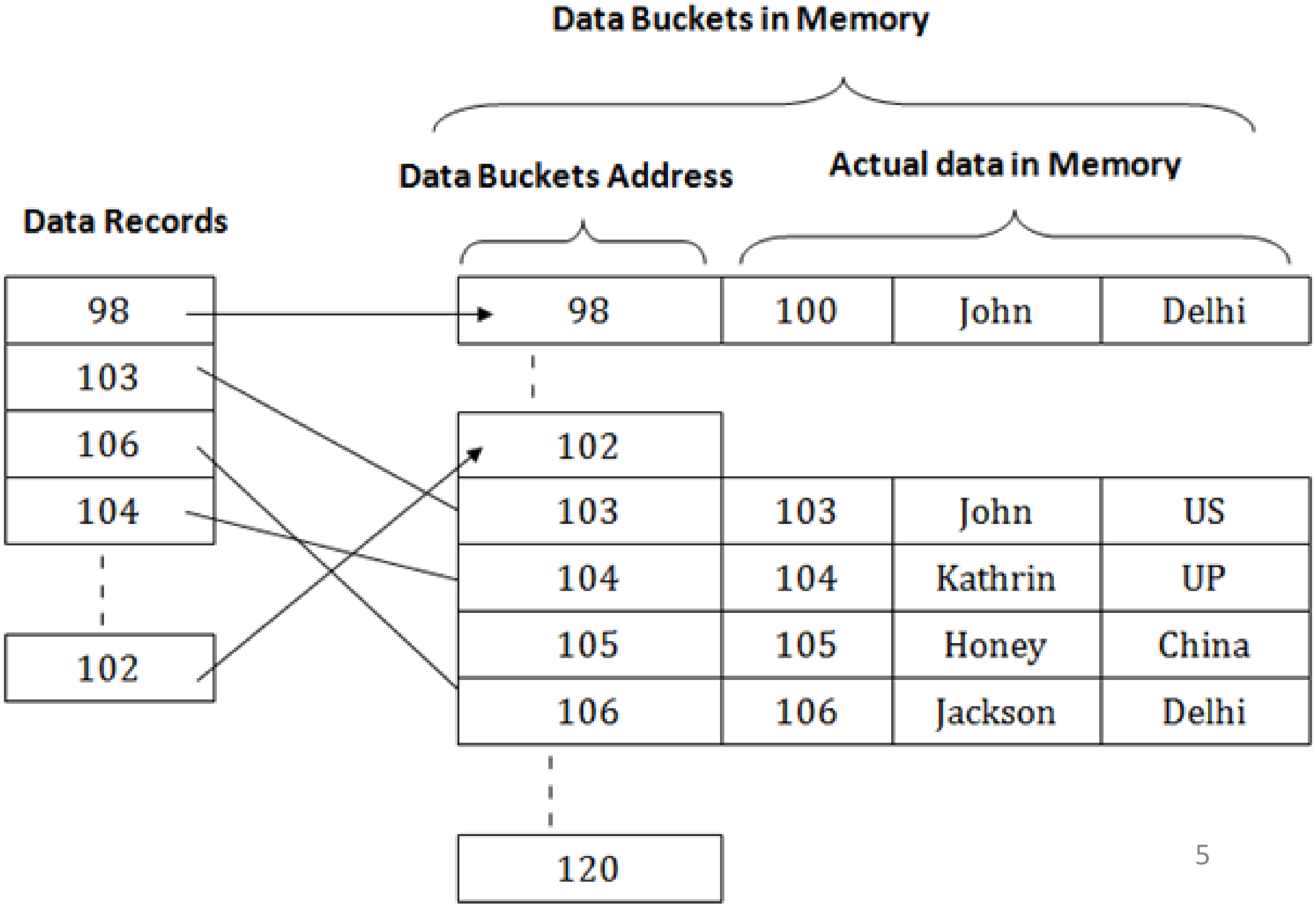
Hashing (cont.)

- In this, 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 block. A hash function is a simple mathematical function to any complex mathematical function.
  - the primary key itself can be considered as the address of the data block. That means each row whose address will be the same as a primary key stored in the data block.

Hash File Organization

- Also known as ‘Direct File Organization’
- Records are stored at known addresses
- To write a record, an address is first calculated by applying mathematical function to the search key of record, record is stored at generated address
- Records stored in buckets = unit of storage that stores one or more records

- One way- The diagram shows data block addresses same as primary key value.



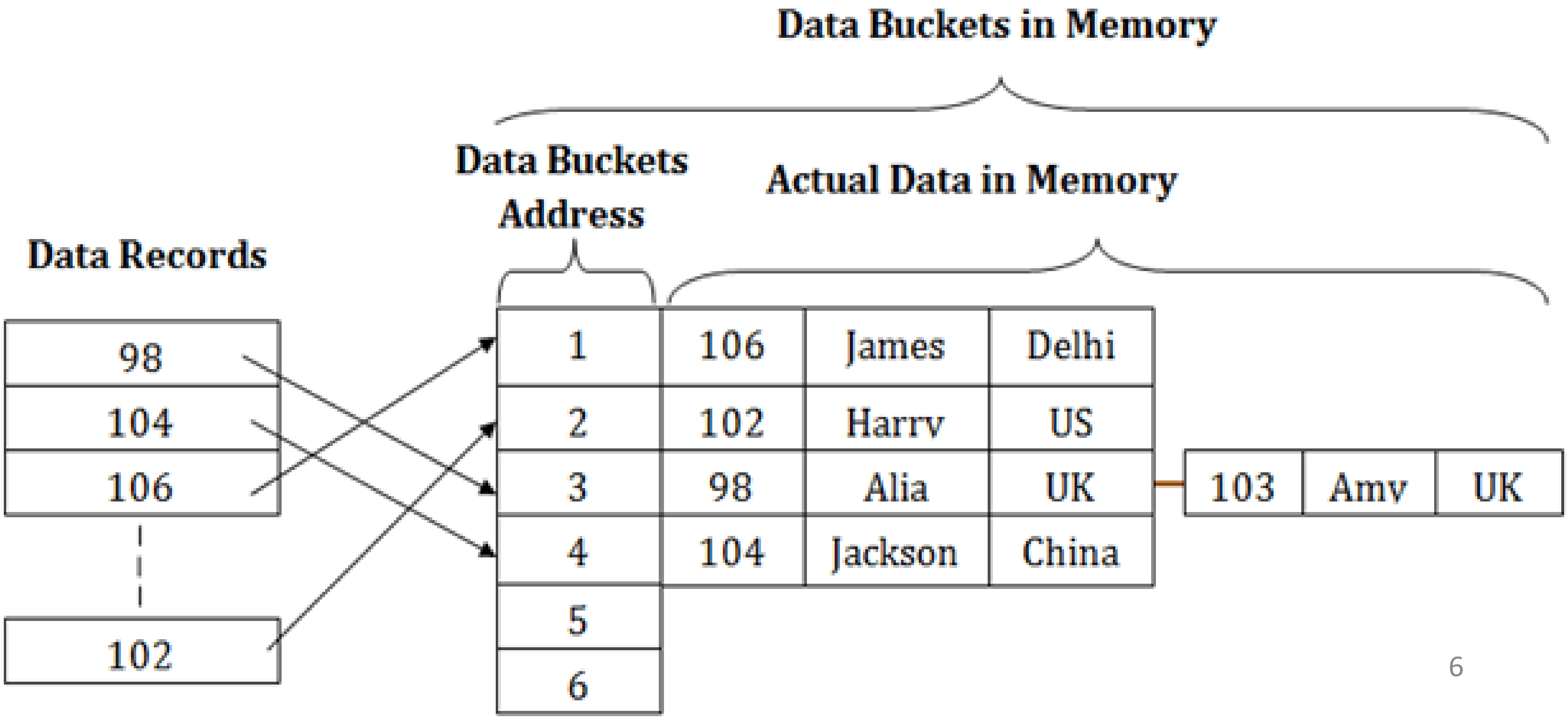
Data Buckets in Memory

Data Records

98
103
106
104
102

Data Buckets Address	Actual data in Memory		
98	100	John	Delhi
102			
103	103	John	US
104	104	Kathrin	UP
105	105	Honey	China
106	106	Jackson	Delhi
120			

- 2nd way- Hash function can also be a simple mathematical function like exponential, mod, cos, sin, etc. Let, 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, 3, 1, 4 and 2 respectively, and records are stored in those data block addresses.



Data Buckets Address	Actual Data in Memory					
1	106	James	Delhi			
2	102	Harry	US			
3	98	Alia	UK	103	Amy	UK
4	104	Jackson	China			
5						
6						

Types of Hashing

- Static Hashing : size of bucket is fixed
- Dynamic Hashing: size of bucket is not fixed

Static Hashing

- The hash function, h, is a function from the set of all search-keys, K, to the set of all bucket addresses, B
- The searching time of Linear and binary searching techniques depends on the number of elements.
- Hashing is a search technique, its searching time does not depend on the number of elements.
- Search time is independent of the position of the record in the file. Insertion, deletion, and lookup are done in constant time

Static Hashing Example

- In static hashing, the resultant data bucket address will always be the same.
  - That means 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. Here, there will be no change in the bucket address.

Properties of the Hash Function

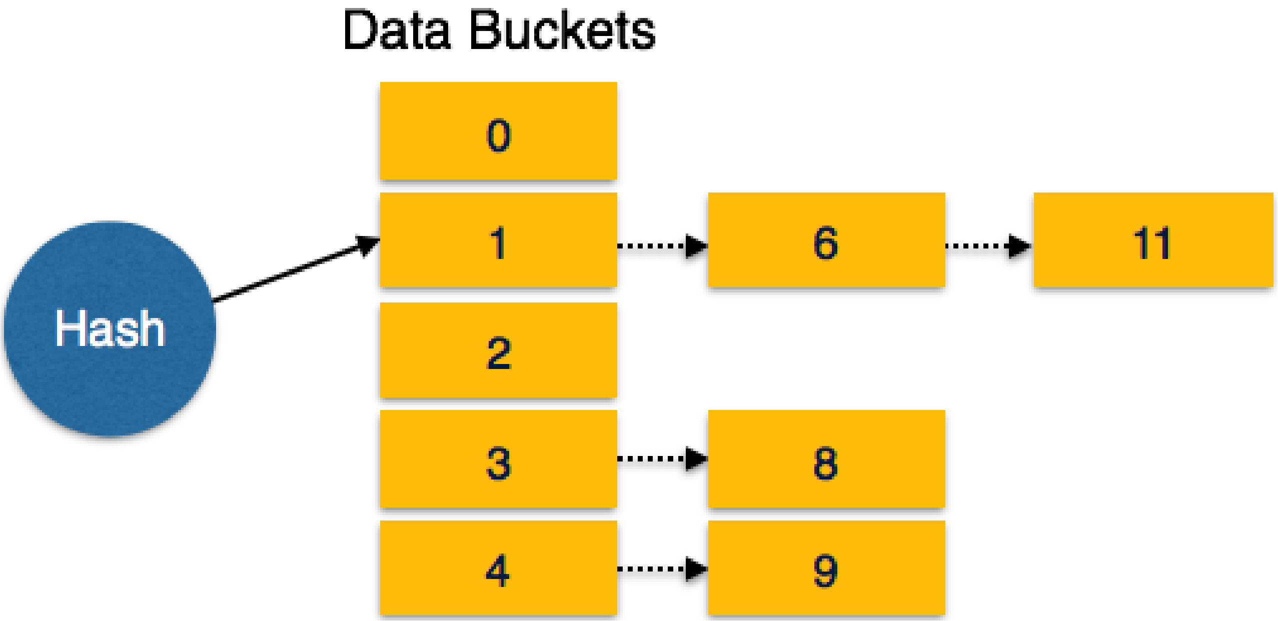
- The distribution should be uniform.
  - An ideal hash function should assign the same number of records in each bucket.
- The distribution should be random.
  - Regardless of the actual search-keys, the each bucket has the same number of records on average
  - Hash values should not depend on any ordering or the search-keys

Bucket Overflow

- How does bucket overflow occur?
  - Insufficient buckets
  - Few buckets have considerably more records than others. This is referred to as skew.
    - Multiple records have the same hash value
    - Non-uniform hash function distribution.

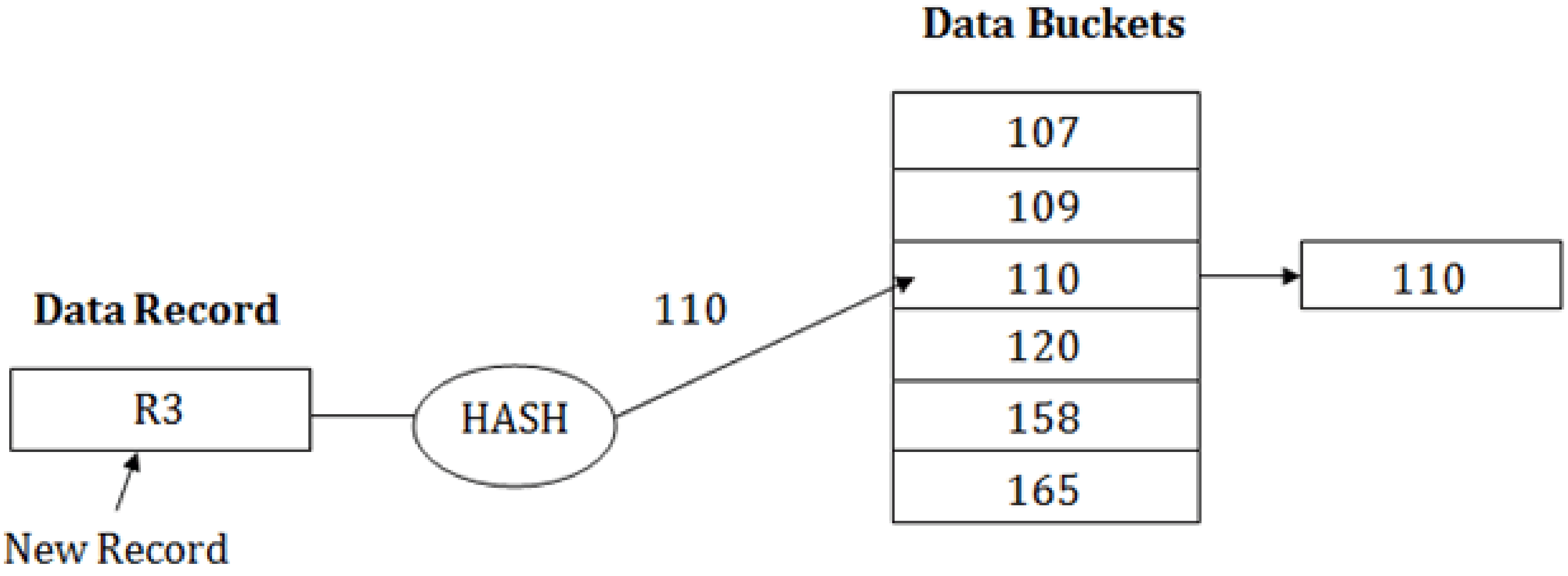
Bucket Overflow

- **Overflow chaining** –
  - When buckets are full, a new bucket is allocated for the same hash result and is linked after the previous one. the overflow buckets of a given bucket are chained together in a linked list.
  - Above scheme is called **closed hashing**



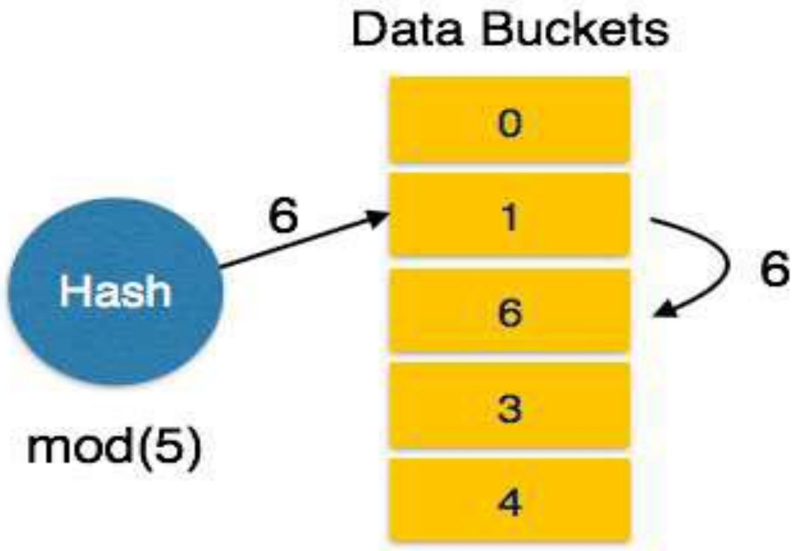
Bucket Overflow – closed chaining example

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.



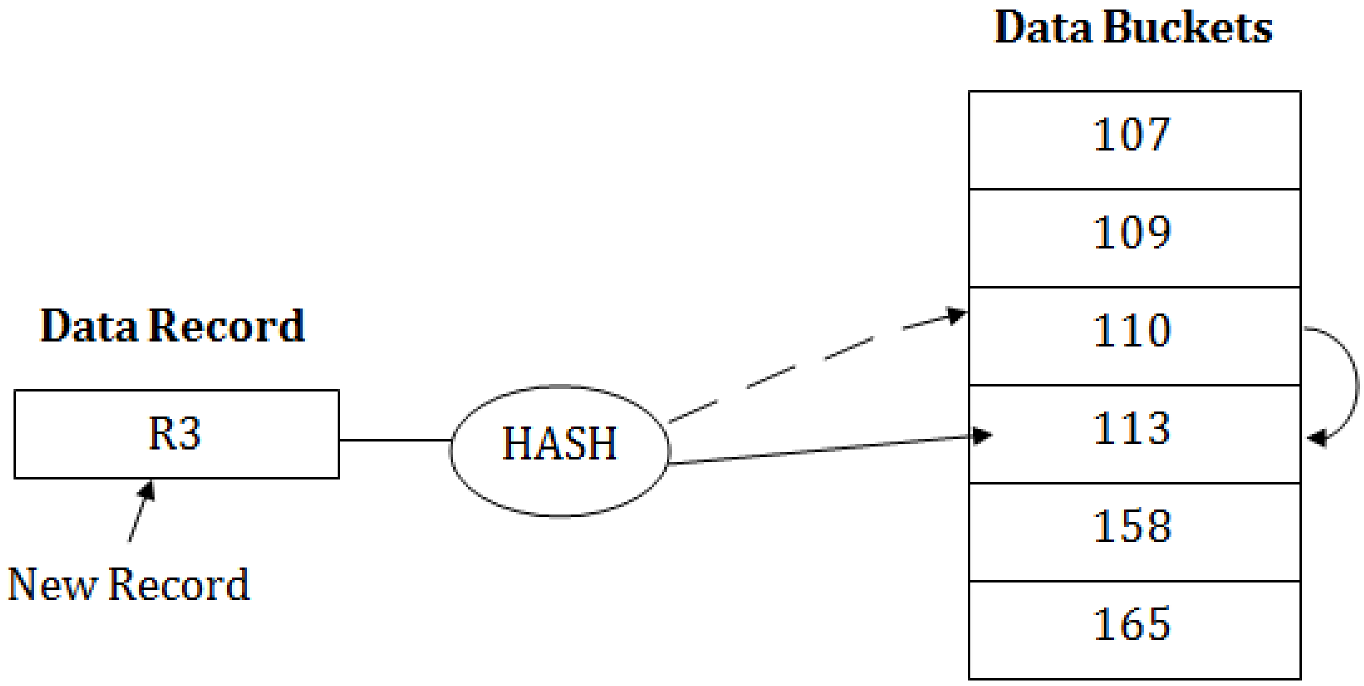
Bucket Overflow

- Linear Probing:
  - When hash function generates an address at which data is already stored, the next free bucket is allocated to it.
  - This mechanism is called Open Hashing.



Bucket Overflow – open hashing example

- Suppose R3 is a new address 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.



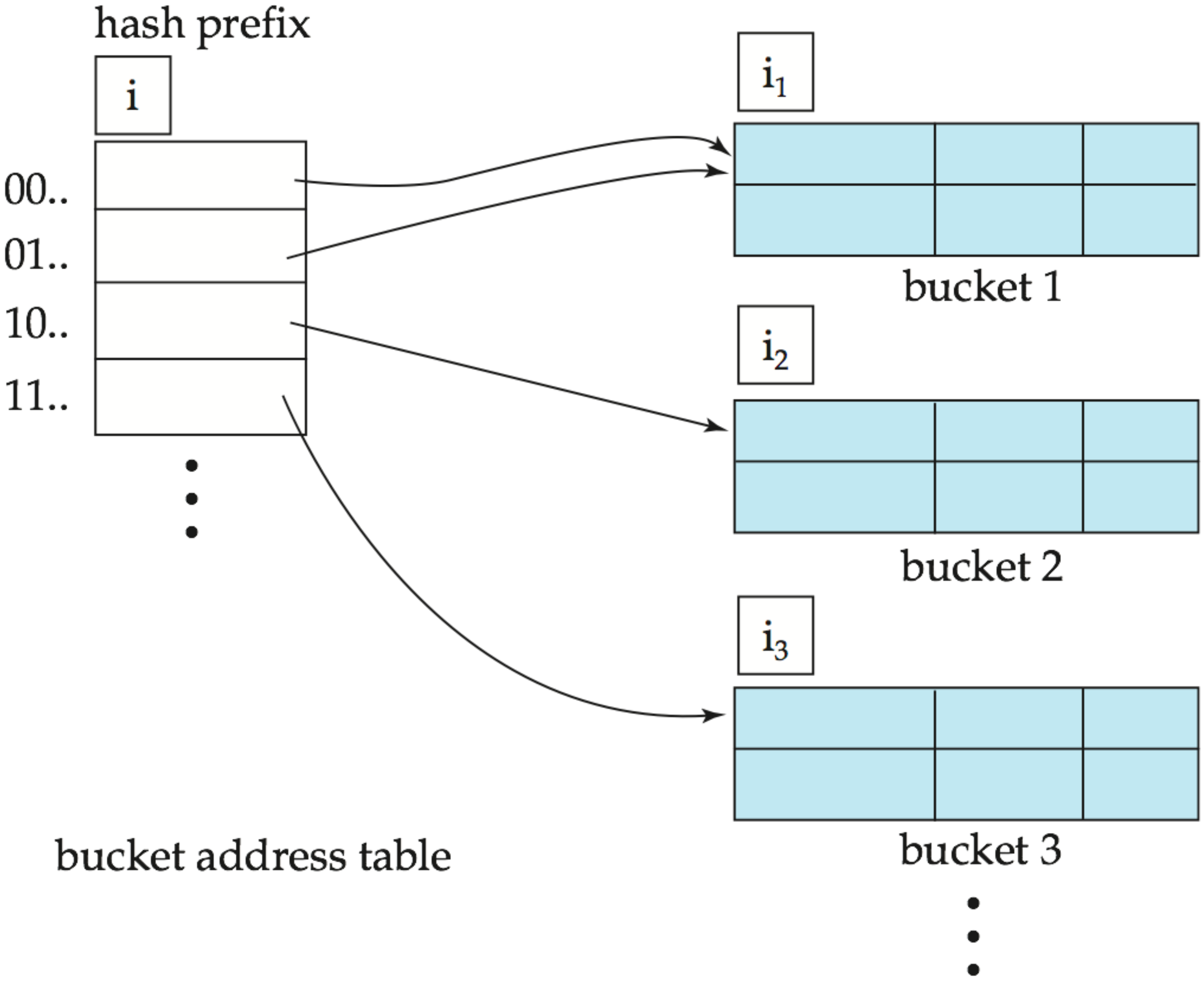
Problem with Static Hashing

- It doesn’t expand or shrink dynamically as the size of database grows or shrinks..

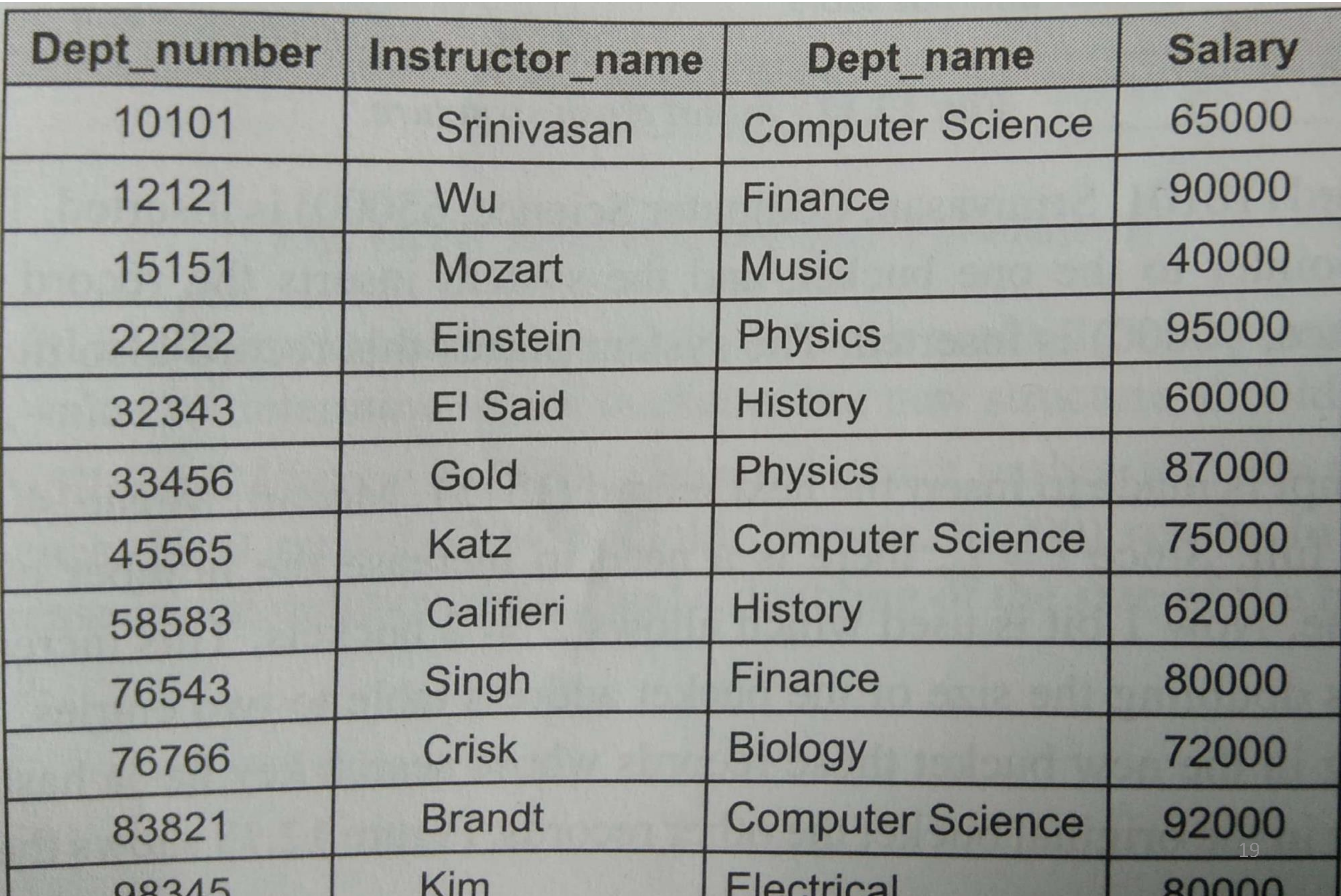
Dynamic Hashing- Extendable Hashing

- Good for database that grows and shrinks in size – overcomes the problem of bucket overflow.
- Allows the hash function to be modified dynamically
- **Extendable hashing** – one form of dynamic hashing
  - Hash function generates values over a large range — typically *b*-bit integers, with *b* = 32.
  - At any time use only a prefix of the hash function to index into a table of bucket addresses.
  - Let the length of the prefix be *i* bits, $0 \leq i \leq 32$.
    - Bucket address table size = $2^{i.}$ Initially *i* = 0
    - Value of *i* grows and shrinks as the size of the database grows and shrinks.

General Structure-Extensible Hashing

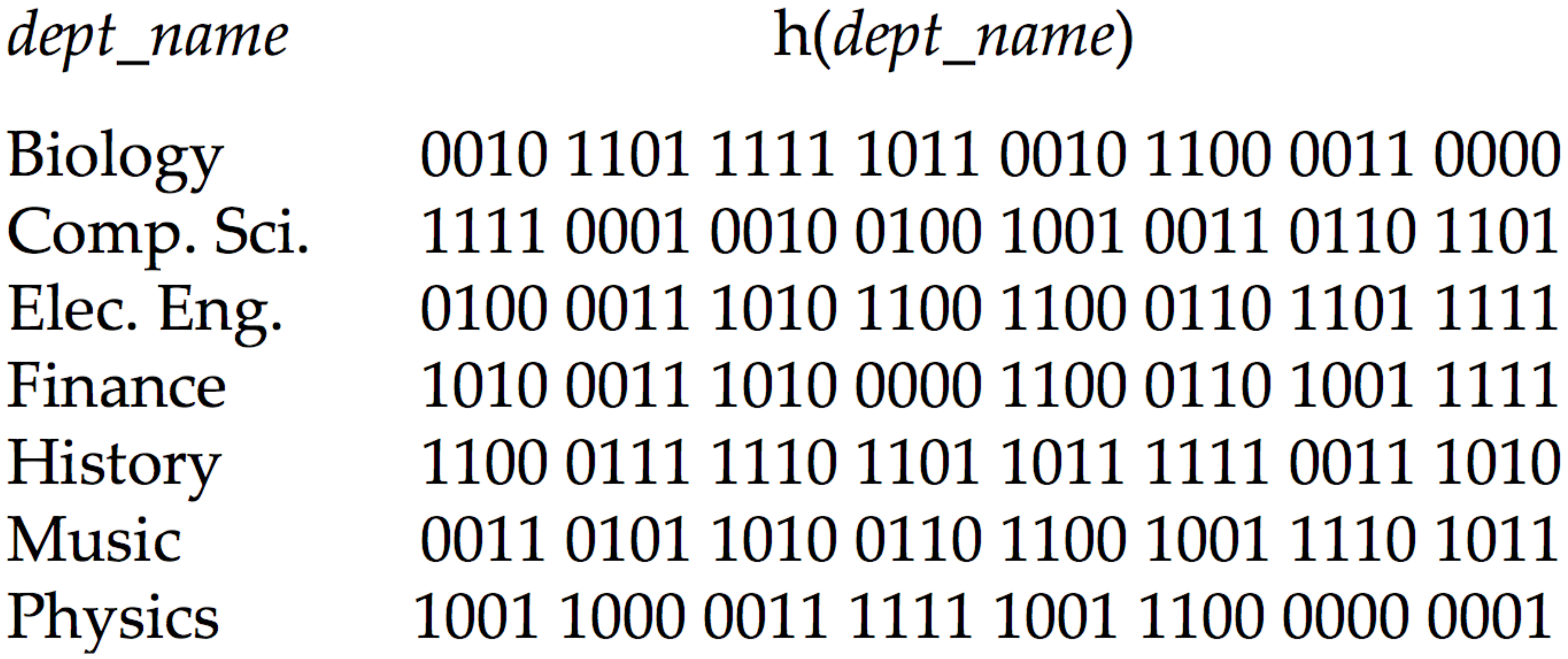


Use of Extendable Hash Structure: Example



Dept_number	Instructor_name	Dept_name	Salary
10101	Srinivasan	Computer Science	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Computer Science	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crisk	Biology	72000
83821	Brandt	Computer Science	92000
98345	Kim	Electrical	80000

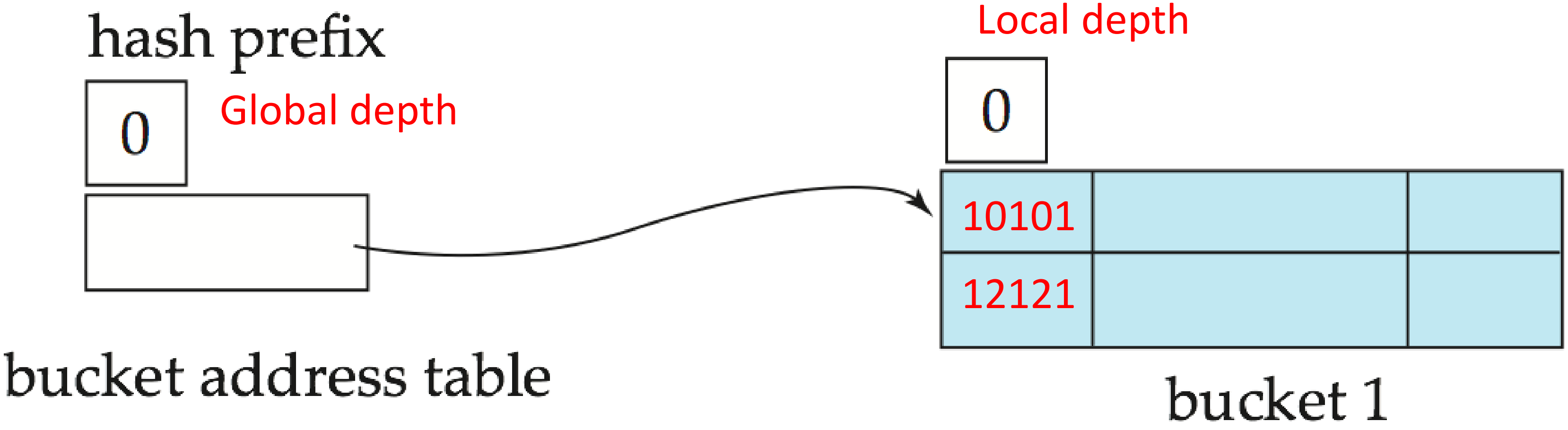
Use of Extendable Hash Structure: Example



dept_name	h(*dept_name*)
Biology	0010 1101 1111 1011 0010 1100 0011 0000
Comp. Sci.	1111 0001 0010 0100 1001 0011 0110 1101
Elec. Eng.	0100 0011 1010 1100 1100 0110 1101 1111
Finance	1010 0011 1010 0000 1100 0110 1001 1111
History	1100 0111 1110 1101 1011 1111 0011 1010
Music	0011 0101 1010 0110 1100 1001 1110 1011
Physics	1001 1000 0011 1111 1001 1100 0000 0001

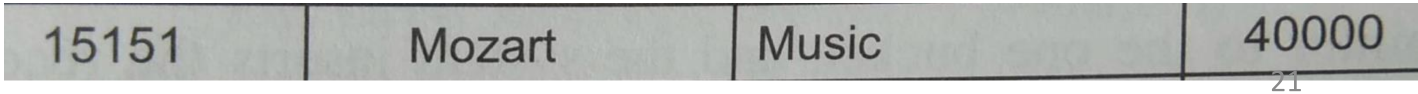
Example (Cont.)

- **Initial Hash structure; bucket size = 2**



- It holds 2records, 10101 , dept- comp sci. and 12121 dept. Finance

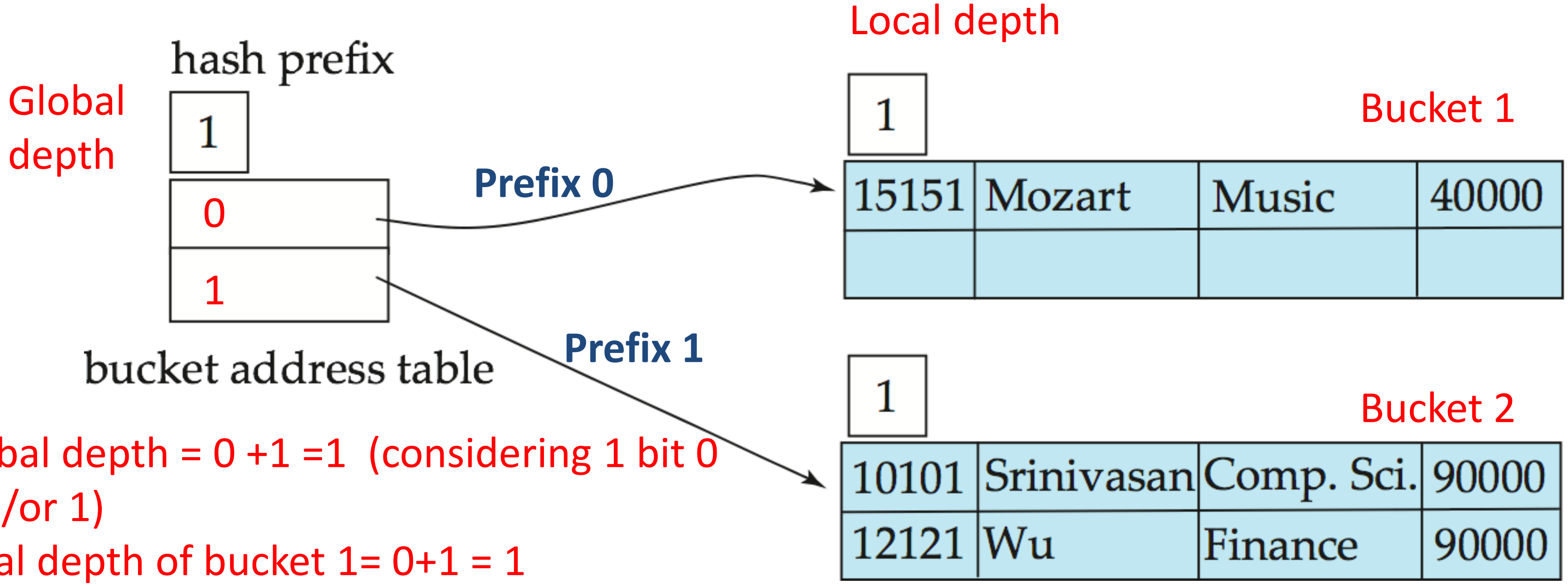
- Insert :



15151	Mozart	Music	40000

Example (Cont.)

- Hash structure after insertion of “Mozart”, “Srinivasan”, and “Wu” records



Bucket 1

15151	Mozart	Music	40000

Bucket 2

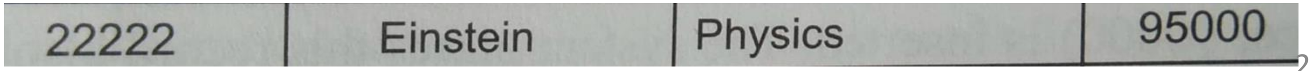
10101	Srinivasan	Comp. Sci.	90000
12121	Wu	Finance	90000

Global depth = 0 +1 =1 (considering 1 bit 0 and/or 1)

Local depth of bucket 1= 0+1 = 1 (considering 0 and/or 1 bit)

Local depth of bucket 2 = 1 (considering 0 and/or 1 bit)

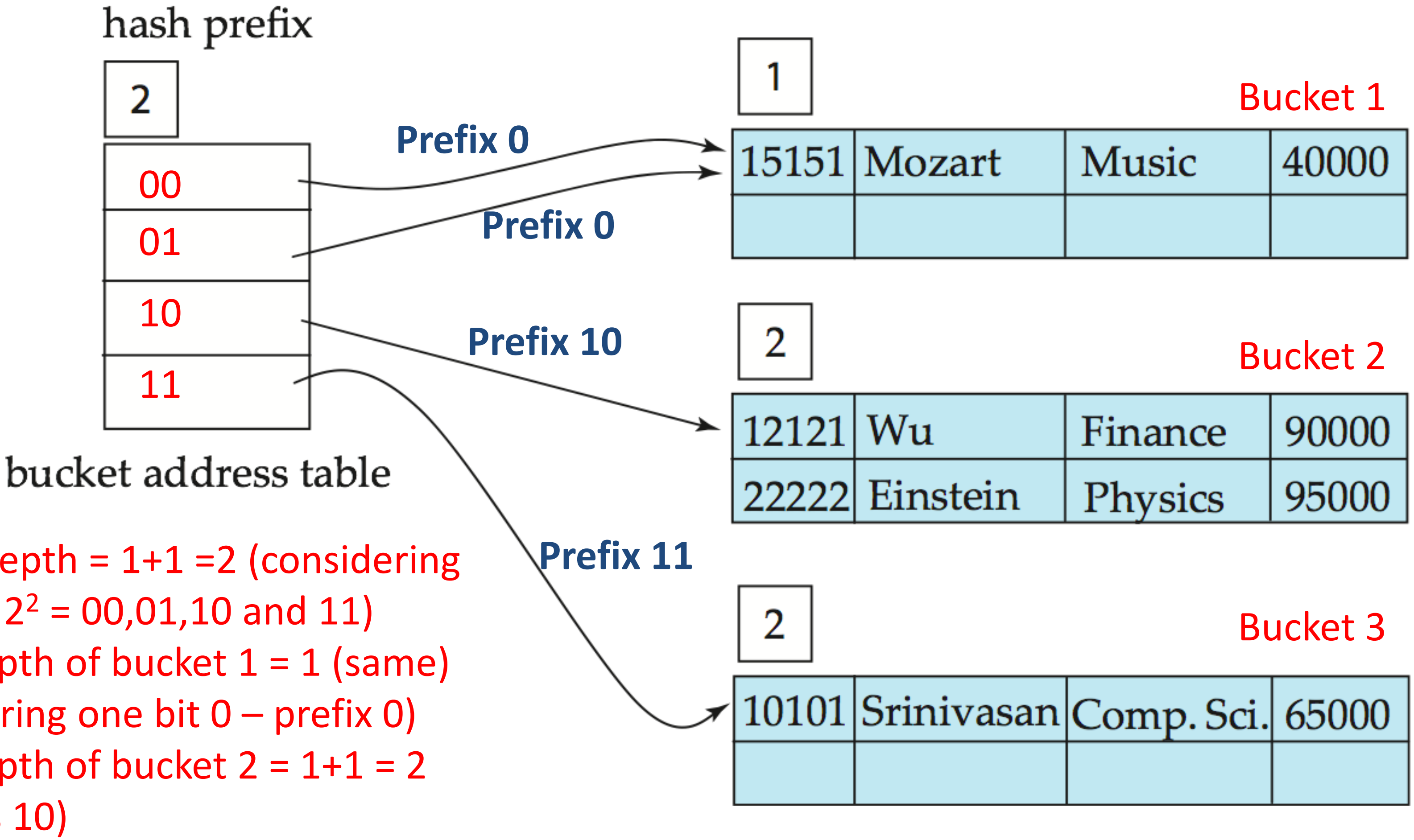
- Insert:



22222	Einstein	Physics	95000

Example (Cont.)

- □ Compare 4 records, records with ’10’ goes in same bucket and with ‘11’ goes in new bucket.
- □ Below is ;Hash structure after insertion of Einstein record



15151	Mozart	Music	40000

12121	Wu	Finance	90000
22222	Einstein	Physics	95000

10101	Srinivasan	Comp. Sci.	65000

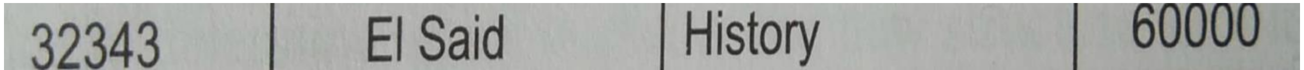
Global depth = 1+1 =2 (considering two bits 2^2 = 00,01,10 and 11)

Local depth of bucket 1 = 1 (same) (considering one bit 0 – prefix 0)

Local depth of bucket 2 = 1+1 = 2 (prefix is 10)

Local depth of bucket 3 = 2

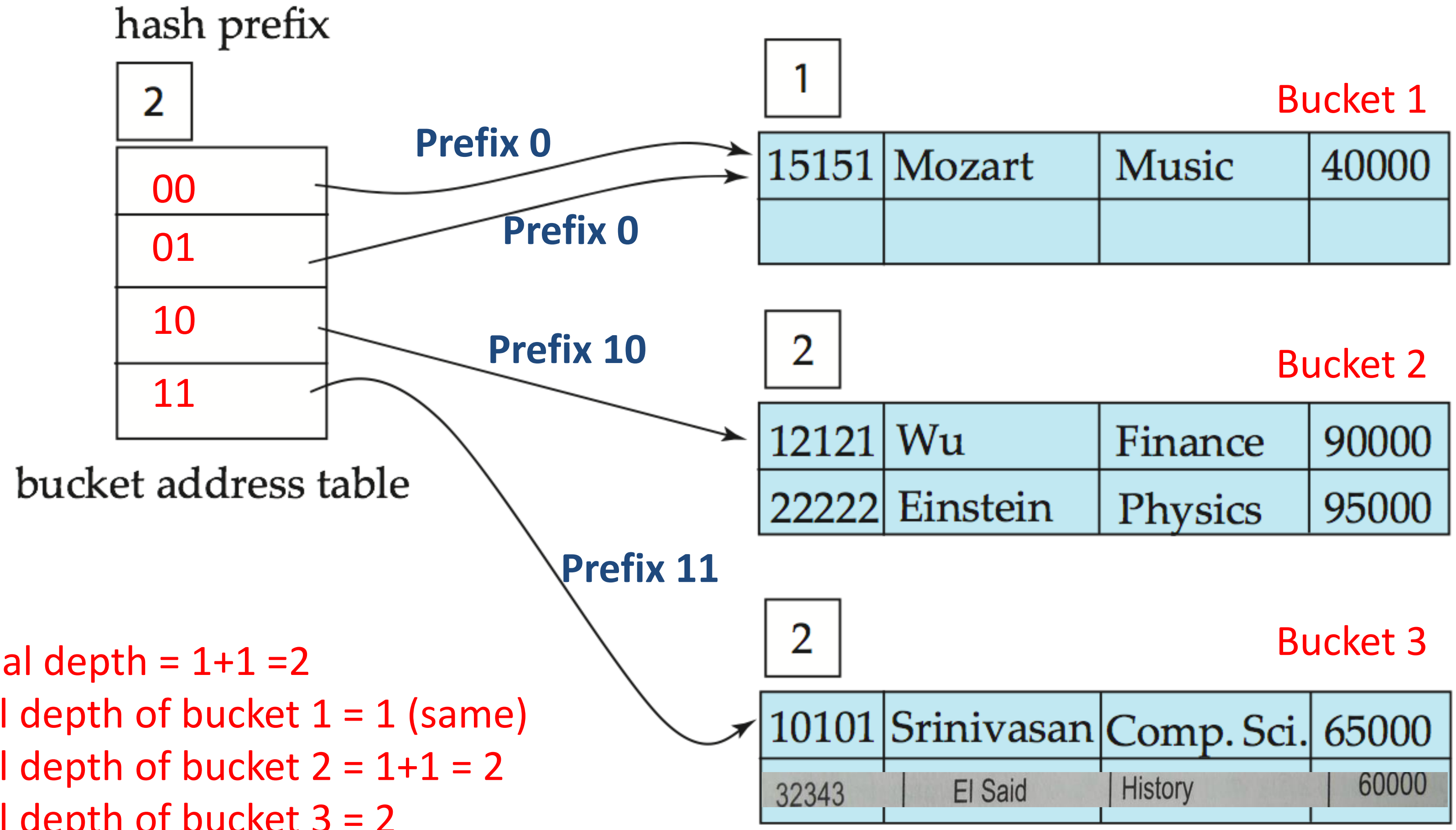
□Insert:



32343	El Said	History	60000

Example (Cont.)

- Compare records, records with ’10’ goes in same bucket and with ‘11’ goes in new bucket.
- Below is ;Hash structure after insertion of Einstein record



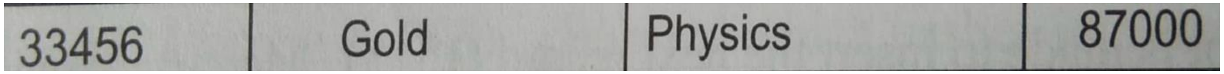
15151	Mozart	Music	40000

12121	Wu	Finance	90000
22222	Einstein	Physics	95000

10101	Srinivasan	Comp. Sci.	65000
32343	El Said	History	60000

Global depth = 1+1 =2
Local depth of bucket 1 = 1 (same)
Local depth of bucket 2 = 1+1 = 2
Local depth of bucket 3 = 2

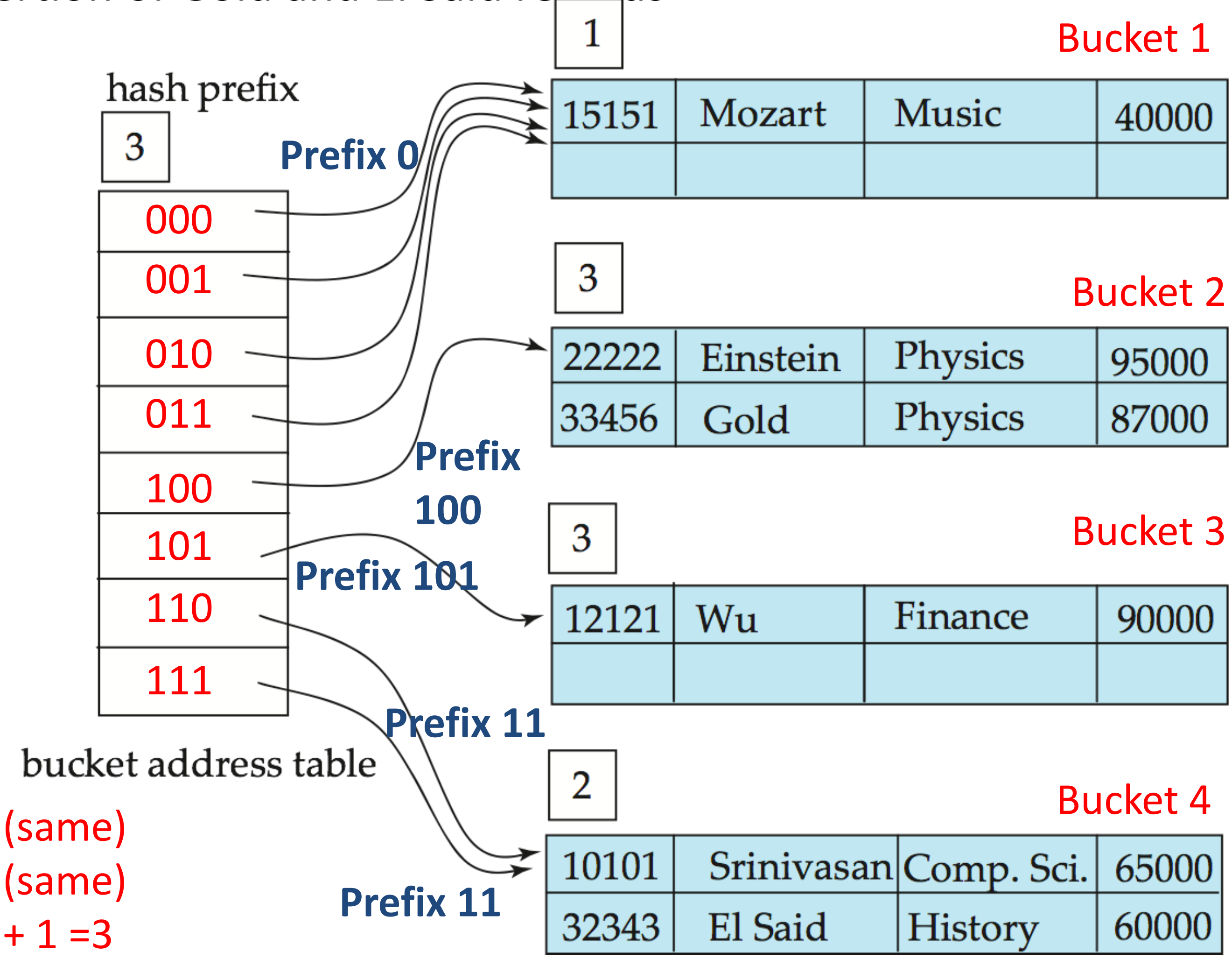
- Insert:



33456	Gold	Physics	87000

Example (Cont.)

- Compare WU, Einstein and gold record for three prefix bits
- Hash structure after insertion of Gold and El Said records



hash prefix

3

000
001
010
011
100
101
110
111

bucket address table

Prefix 0

Prefix 100

Prefix 101

Prefix 11

Prefix 11

1

Bucket 1

15151	Mozart	Music	40000

3

Bucket 2

22222	Einstein	Physics	95000
33456	Gold	Physics	87000

3

Bucket 3

12121	Wu	Finance	90000

2

Bucket 4

10101	Srinivasan	Comp. Sci.	65000
32343	El Said	History	60000

Global depth = 3

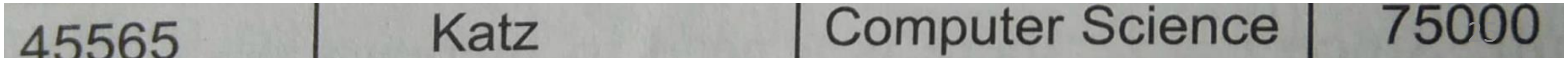
Local depth of bucket 1 = 1 (same)

Local depth of bucket 4 = 2 (same)

Local depth of bucket 2 = 2 + 1 =3

Local depth of bucket 3 = 3

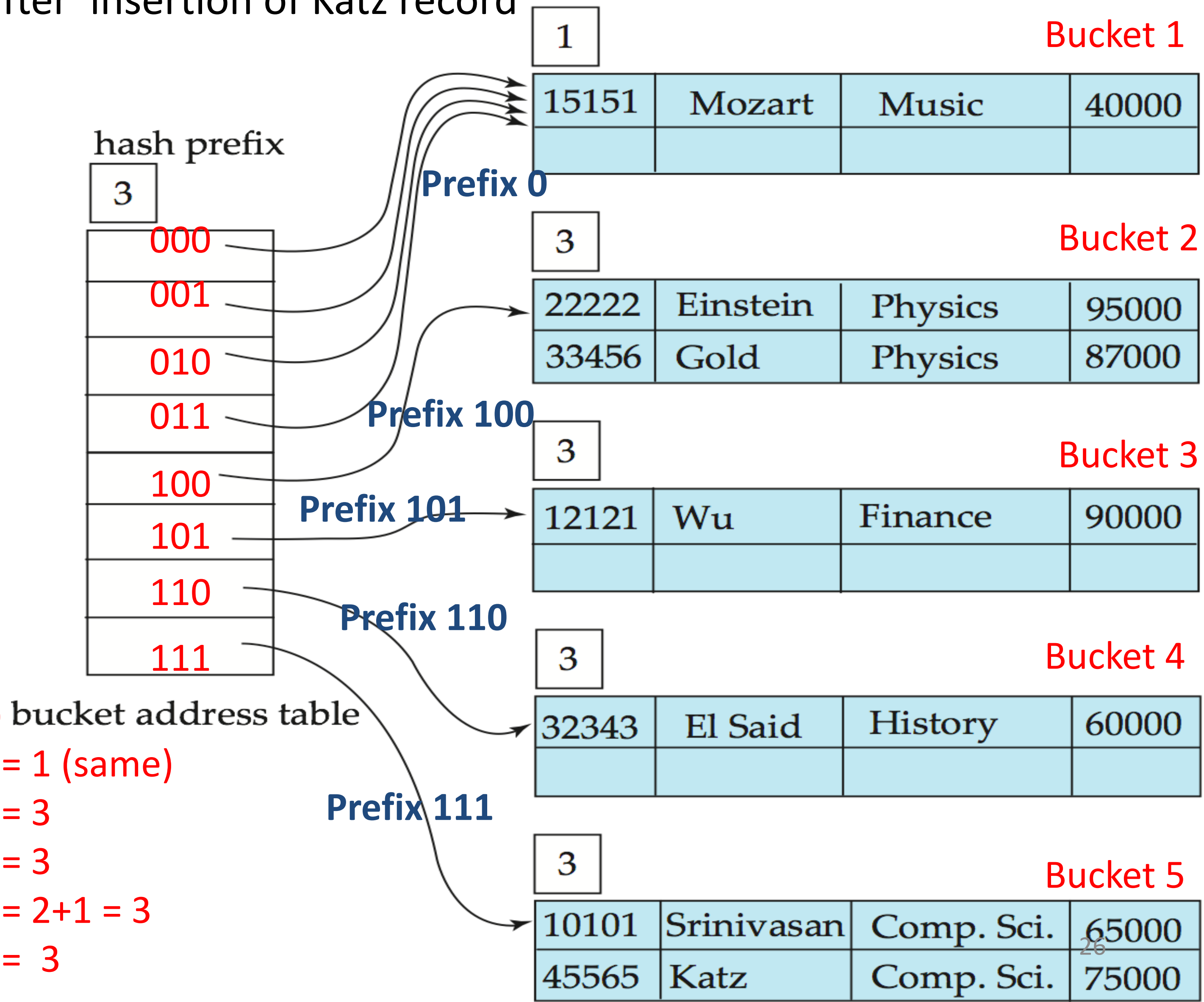
- Insert:



45565	Katz	Computer Science	75000

Example (Cont.)

- Compare Shrinivasan, El Said and katz
- Hash structure after insertion of Katz record



Bucket 1

15151	Mozart	Music	40000

Bucket 2

22222	Einstein	Physics	95000
33456	Gold	Physics	87000

Bucket 3

12121	Wu	Finance	90000

Bucket 4

32343	El Said	History	60000

Bucket 5

10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000

Global depth = 3 (same)

Local depth of bucket 1 = 1 (same)

Local depth of bucket 2 = 3

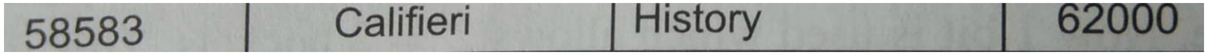
Local depth of bucket 3 = 3

Local depth of bucket 4 = 2+1 = 3

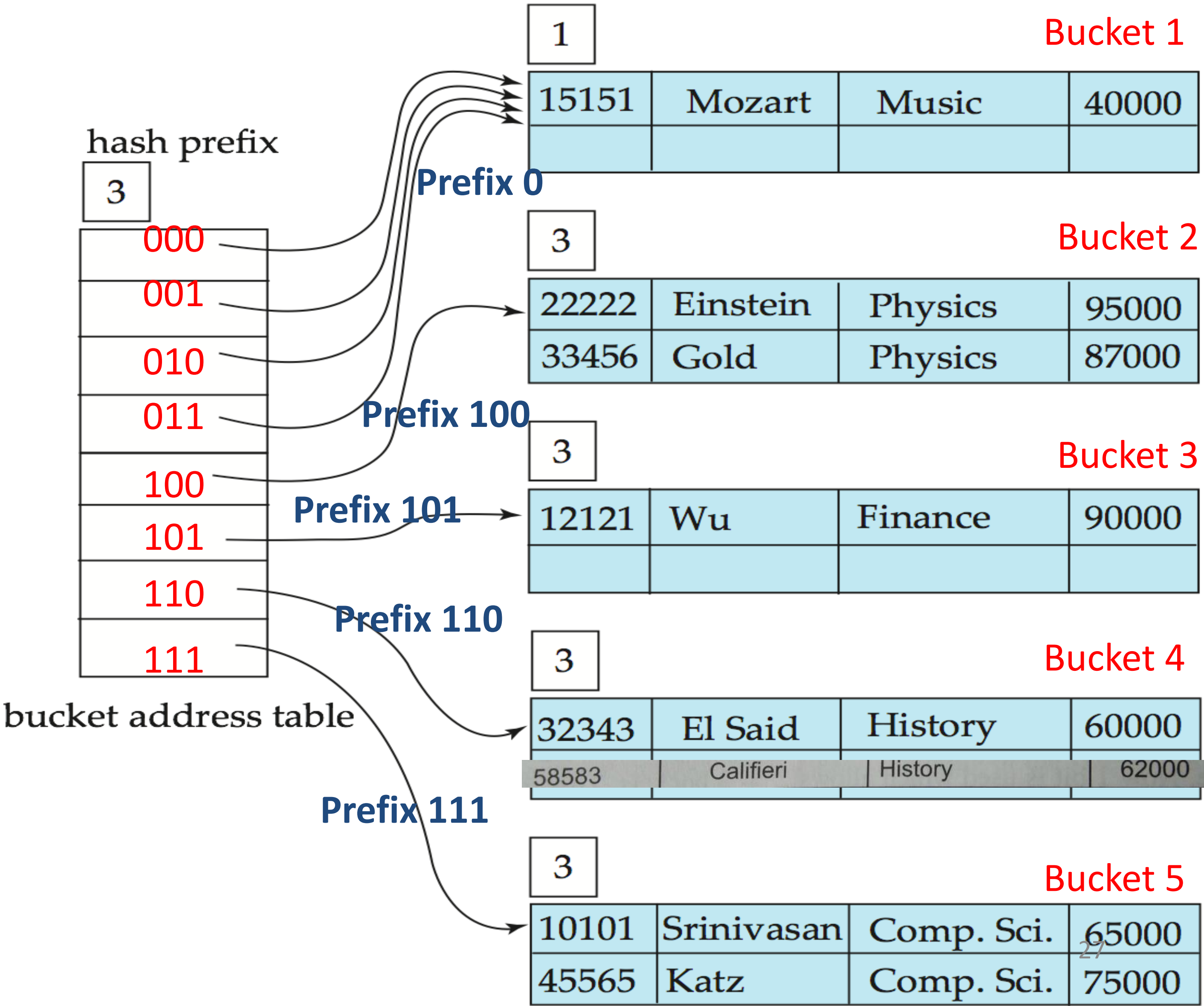
Local depth of bucket 5 = 3

Example (Cont.)

- Hash structure after insertion of



58583	Califieri	History	62000



hash prefix

3

000
001
010
011
100
101
110
111

bucket address table

Prefix 0

Prefix 100

Prefix 101

Prefix 110

Prefix 111

Bucket 1

1

15151	Mozart	Music	40000

Bucket 2

3

22222	Einstein	Physics	95000
33456	Gold	Physics	87000

Bucket 3

3

12121	Wu	Finance	90000

Bucket 4

3

32343	El Said	History	60000
58583	Califieri	History	62000

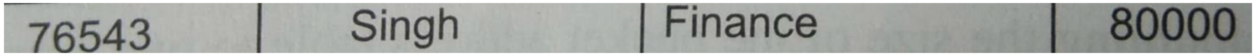
Bucket 5

3

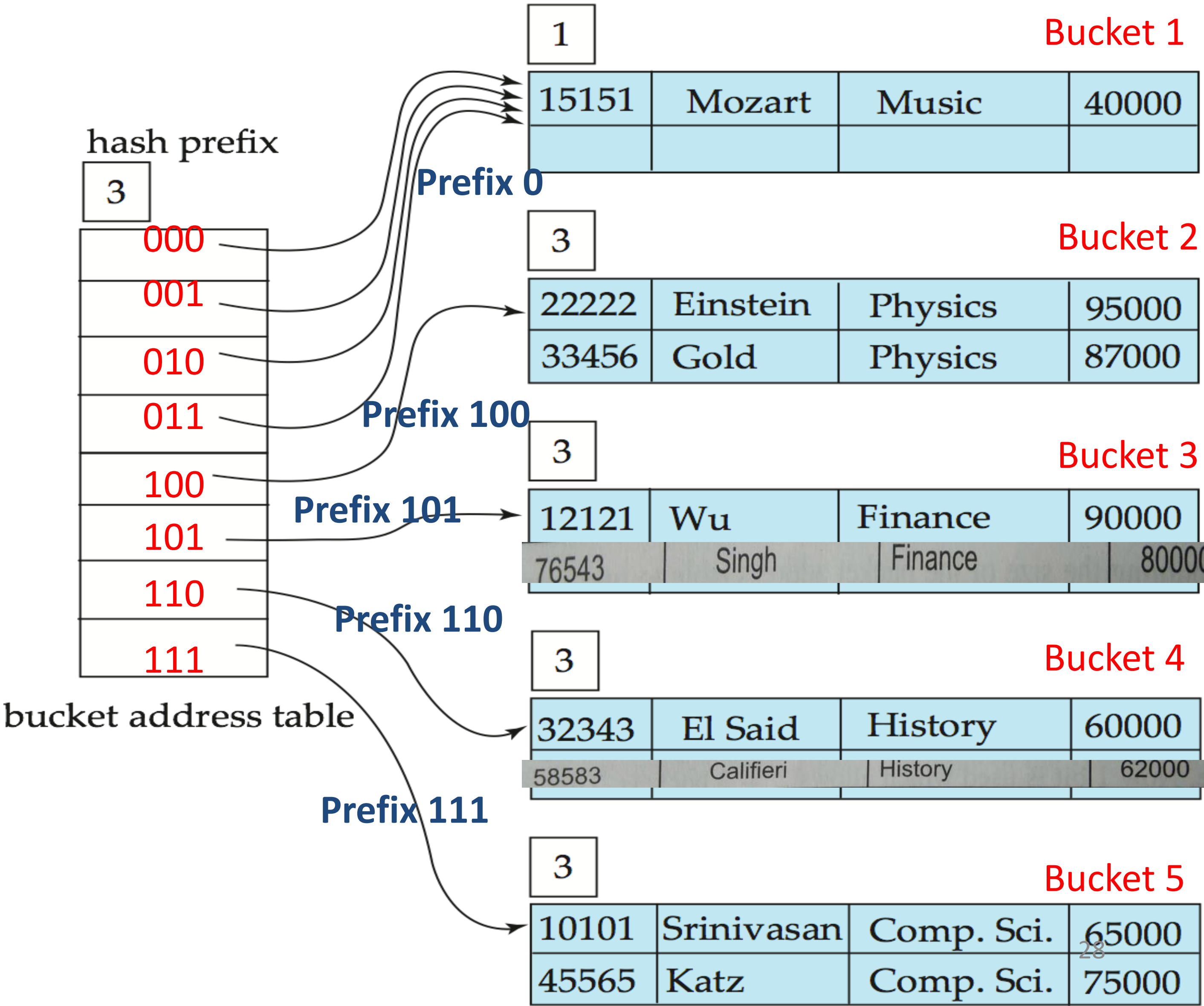
10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000

Example (Cont.)

- Hash structure after insertion of



76543	Singh	Finance	80000



hash prefix

3

bucket address table

000
001
010
011
100
101
110
111

Prefix 0

Prefix 100

Prefix 101

Prefix 110

Prefix 111

Bucket 1

1

15151	Mozart	Music	40000

Bucket 2

3

22222	Einstein	Physics	95000
33456	Gold	Physics	87000

Bucket 3

3

12121	Wu	Finance	90000
76543	Singh	Finance	8000

Bucket 4

3

32343	El Said	History	60000
58583	Califieri	History	62000

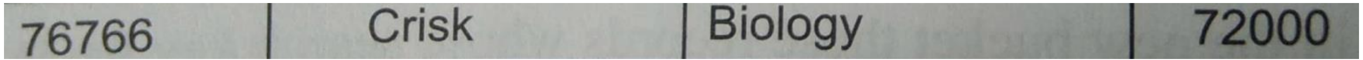
Bucket 5

3

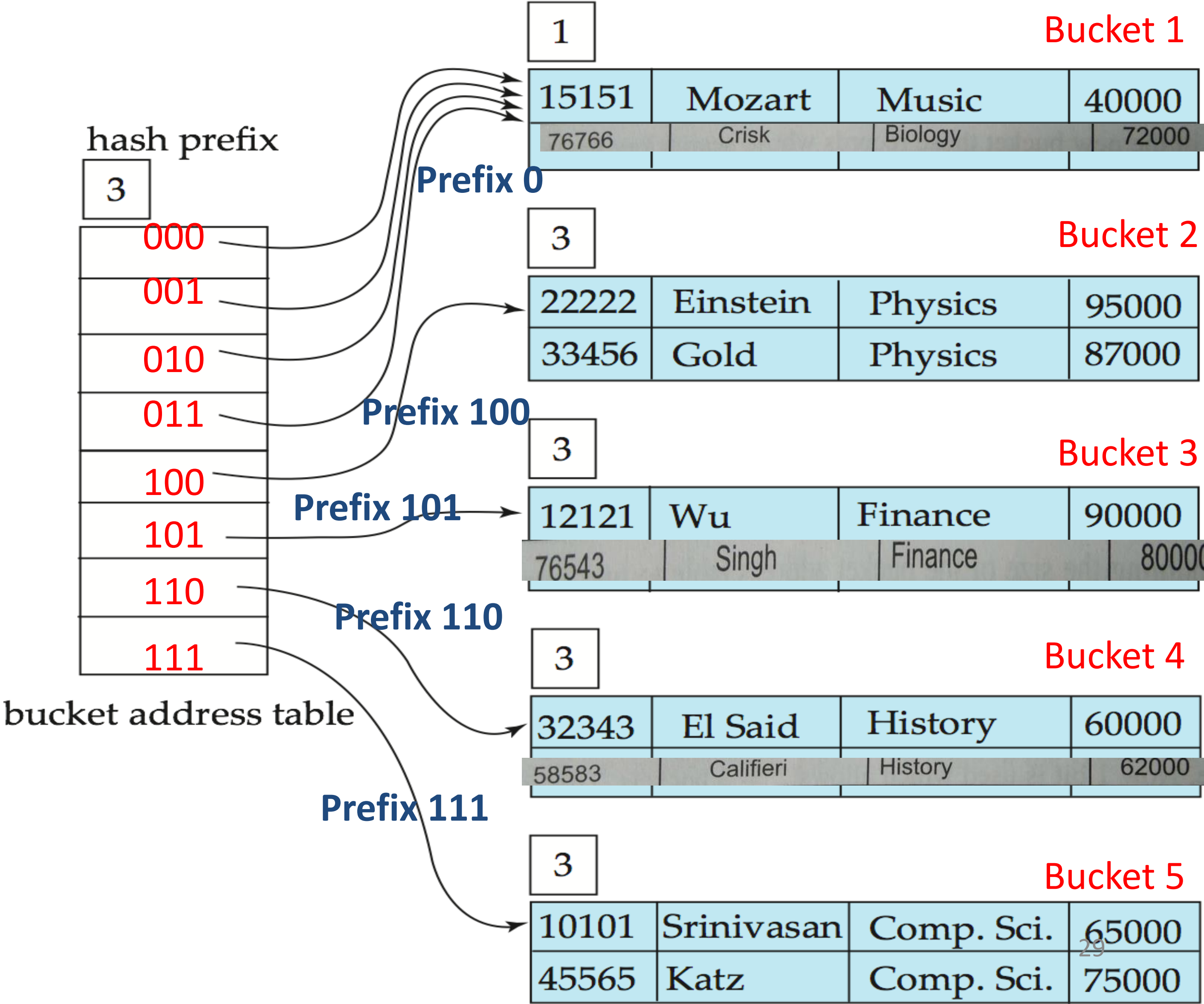
10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000

Example (Cont.)

- Hash structure after insertion of



76766	Crisk	Biology	72000



hash prefix

3

bucket address table

000
001
010
011
100
101
110
111

Prefix 0

Prefix 100

Prefix 101

Prefix 110

Prefix 111

Bucket 1

1

15151	Mozart	Music	40000
76766	Crisk	Biology	72000

Bucket 2

3

22222	Einstein	Physics	95000
33456	Gold	Physics	87000

Bucket 3

3

12121	Wu	Finance	90000
76543	Singh	Finance	8000

Bucket 4

3

32343	El Said	History	60000
58583	Califieri	History	62000

Bucket 5

3

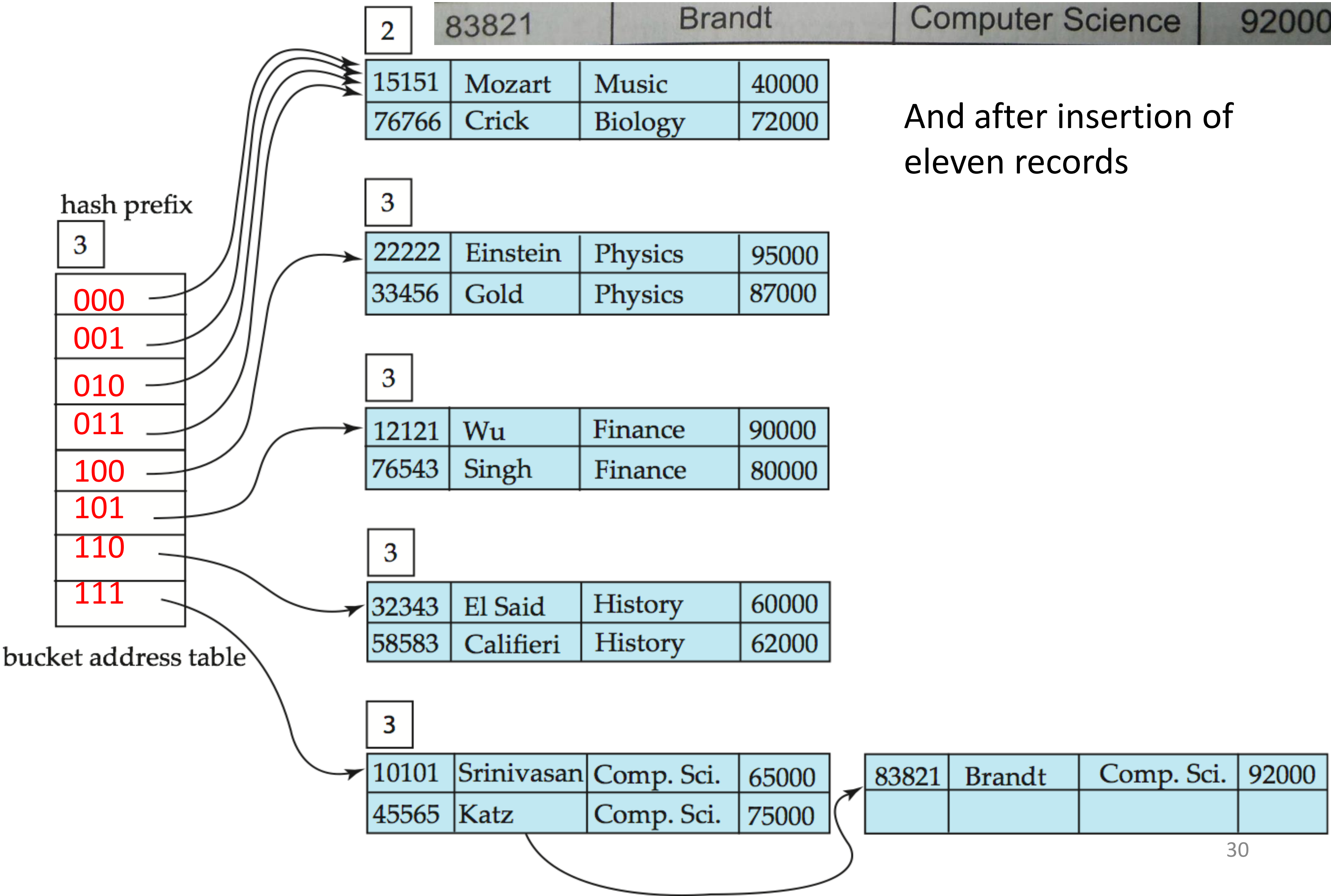
10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000

Example (Cont.)

- Hash structure after insertion of

83821	Brandt	Computer Science	92000

And after insertion of eleven records



15151	Mozart	Music	40000
76766	Crick	Biology	72000

22222	Einstein	Physics	95000
33456	Gold	Physics	87000

12121	Wu	Finance	90000
76543	Singh	Finance	80000

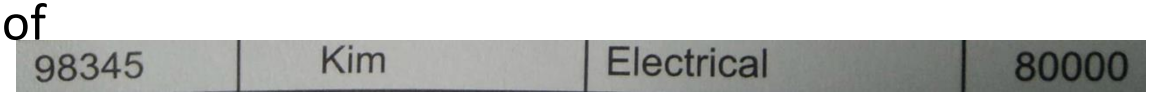
32343	El Said	History	60000
58583	Califieri	History	62000

10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000

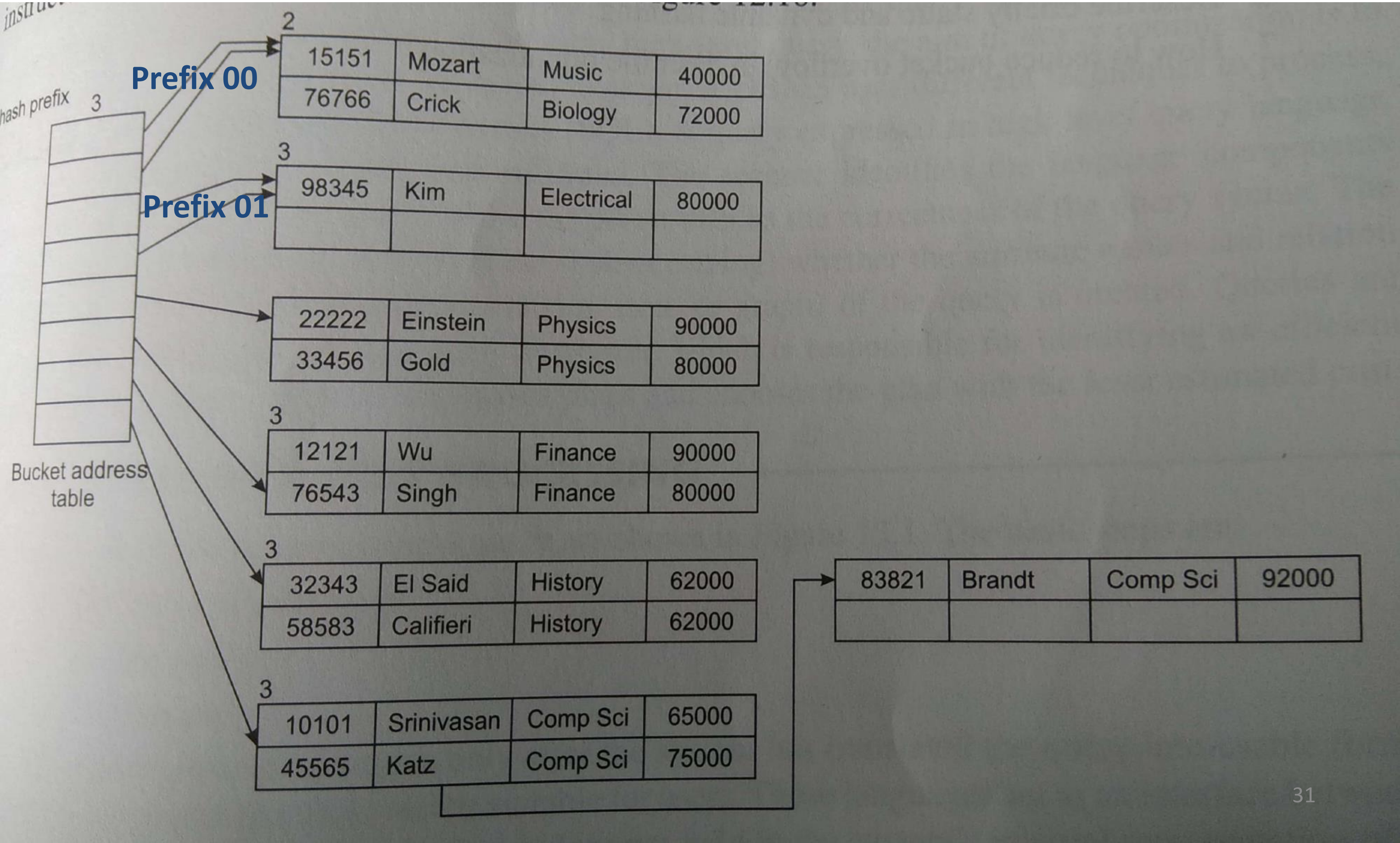
83821	Brandt	Comp. Sci.	92000

Example (Cont.)

- Hash structure after insertion of



98345	Kim	Electrical	80000



15151	Mozart	Music	40000
76766	Crick	Biology	72000

98345	Kim	Electrical	80000

22222	Einstein	Physics	90000
33456	Gold	Physics	80000

12121	Wu	Finance	90000
76543	Singh	Finance	80000

32343	El Said	History	62000
58583	Califieri	History	62000

10101	Srinivasan	Comp Sci	65000
45565	Katz	Comp Sci	75000

83821	Brandt	Comp Sci	92000

Note:

- When Depth is 1 = it considers 1 bit only either 0 or 1 from prefix (MSB) of given hash value
- When Depth is 2 = considers two bits 2^2 = 00,01,10,11 as prefix from given hash value
- When Depth is 3 = considers two bits 2^3 = 000 to 111 as prefix from given hash value

Ordered Indexing Vs Hashing

Indexing	Hashing
It is a technique that allows to quickly retrieve records from database file.	It is a technique that allows to quickly retrieve records from database file.
It is generally used to optimize or increase performance of database simply by minimizing number of disk accesses that are required when a query is processed.	It is generally used to index and retrieve items in database as it is faster to search that specific item using shorter hashed key rather than using its original value.
It is not considered best for large databases and its good for small databases.	It is considered best for large databases.
It uses data reference to hold address of disk block.	It uses mathematical functions known as hash function to calculate direct location of records on disk.

References

Korth