KOÇ UNIVERSITY

Database Management Systems

Hash-Based Indexing

M. Emre Gürsoy

Assistant Professor
Department of Computer Engineering
www.memregursoy.com

Motivation

- When tables are large, it is inefficient to search all values to find matches (aka: linear search or full table scan).

```
SELECT  Lname
FROM    Employee
WHERE   Ssn = 123456789
```

- In databases, an index is a data structure that enables locating data quickly without having to do a full table scan.
- We'll cover two types of indexing:
  - Hash-based indexing
    - Good for equality selections (not so much for range selections)
  - Tree-based indexing

“Create Index”

- Almost all DBMS have a **CREATE INDEX** command that allows you to create indexes on desired attributes.
  - Example from MySQL manual (on the right).
- Indexes are also used in DBMS internals.

```
CREATE [UNIQUE | FULLTEXT | SPATIAL] INDEX index_name
    [index_type]
    ON tbl_name (key_part,...)
    [index_option]
    [algorithm_option | lock_option] ...

key_part: {col_name [(length)] | (expr)} [ASC | DESC]

index_option: {
    KEY_BLOCK_SIZE [=] value
  | index_type
  | WITH PARSER parser_name
  | COMMENT 'string'
  | {VISIBLE | INVISIBLE}
  | ENGINE_ATTRIBUTE [=] 'string'
  | SECONDARY_ENGINE_ATTRIBUTE [=] 'string'
}

index_type:
    USING {BTREE | HASH}

algorithm_option:
    ALGORITHM [=] {DEFAULT | INPLACE | COPY}

lock_option:
    LOCK [=] {DEFAULT | NONE | SHARED | EXCLUSIVE}
```

Table Indexes

- Advantages:
  - Faster search and retrieval **(key reason!)**
- Disadvantages:
  - Indexes must also be updated at each insert/delete/upd.
    - Updating an index itself is work
    - Also, index must be locked before update, potentially affecting other transactions' throughput
  - Indexes cost space
- General advice: Create indexes when they are useful, but don't create redundant indexes!

Outline

- Hashing (basics)
- Static hashing with extensions
  - Overflow chaining
  - Linear probing
- Extendible hashing
- Linear hashing

Hashing

- A hash table is an associative array that maps keys to values.
  - Keys here do not mean keys in the relational model!
- A hash function is used to map data of arbitrary size (e.g., long strings) into fixed-size values (e.g., integers).
- A hash collision occurs when two different keys are mapped to the same hash value.
  - key1 ≠ key2, but h(key1) = h(key2)
- Desired properties of hash functions:
  - Fast (not necessarily cryptographically secure)
  - Low collision rate

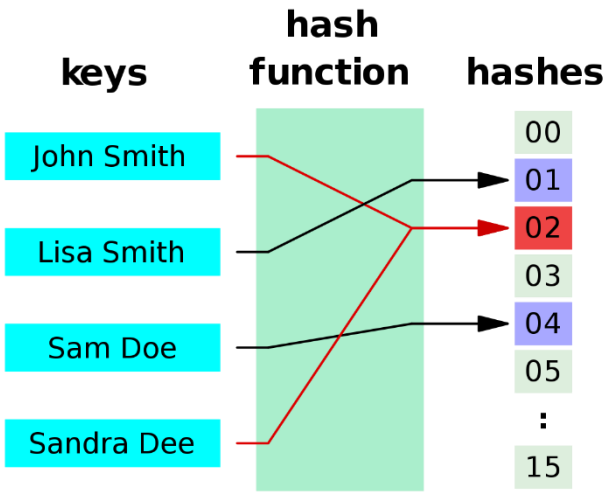
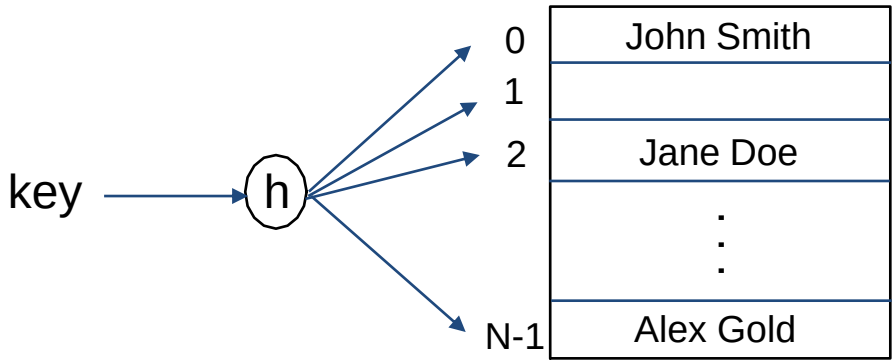


Image source: Wikipedia

Static Hashing

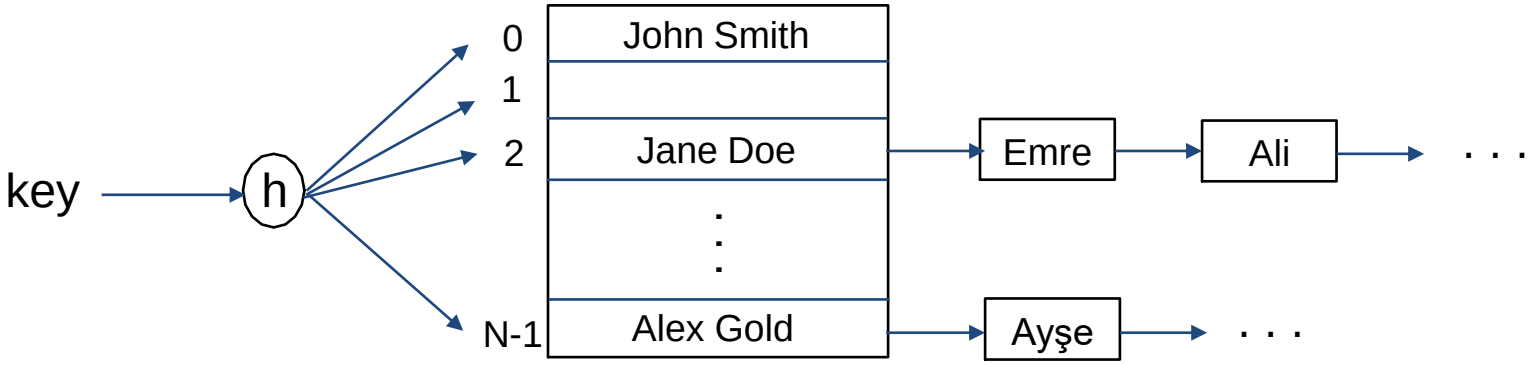
- Allocate a large array that has one slot for each record
  - Potential hash fn: h(k) mod N, where N: # of records
  - Each slot holds one record



- Works when:
  - You know the # of records (N) ahead of time
    - You do not need to grow/shrink the hash table size!
  - Hash function is perfect, i.e., no hash collisions

Overflow Chaining

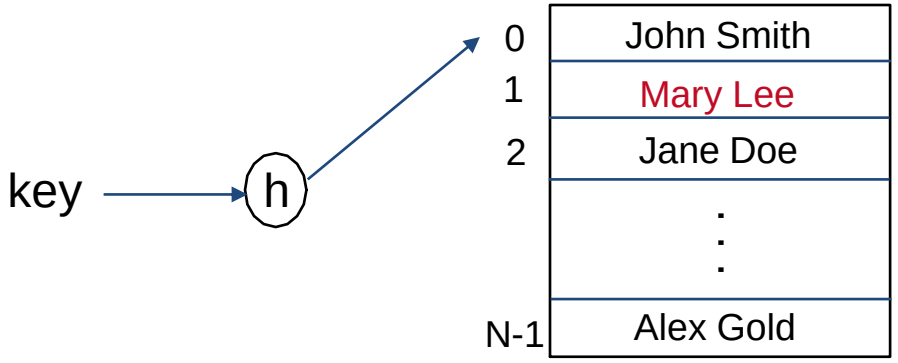
- A straightforward extension of static hashing to handle hash collisions
- If there is a hash collision, create overflow buckets
  - Overflow buckets become a chain (can be treated like a linked list)



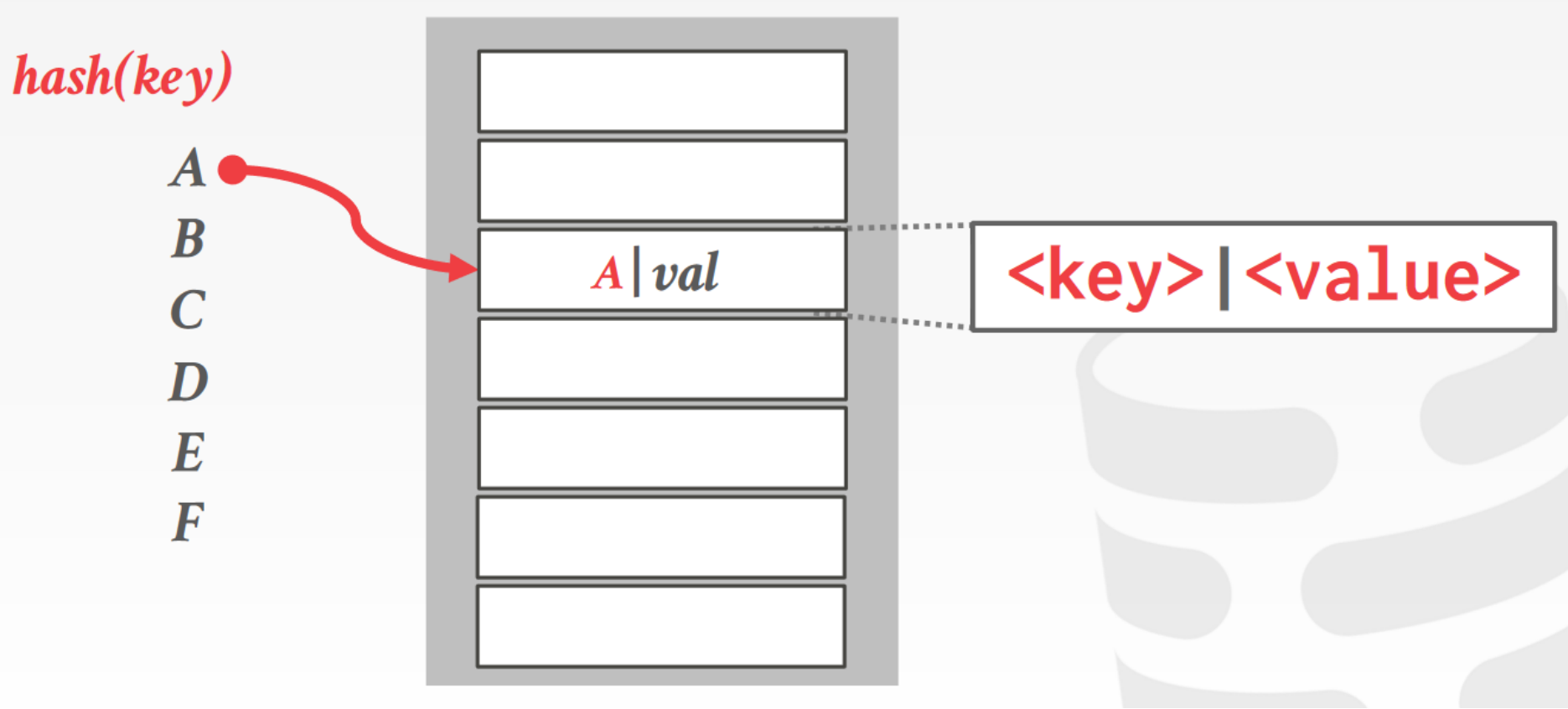
- How do you Search? Insert? Delete?
  - Best case complexity? Worst case complexity?

Linear Probing

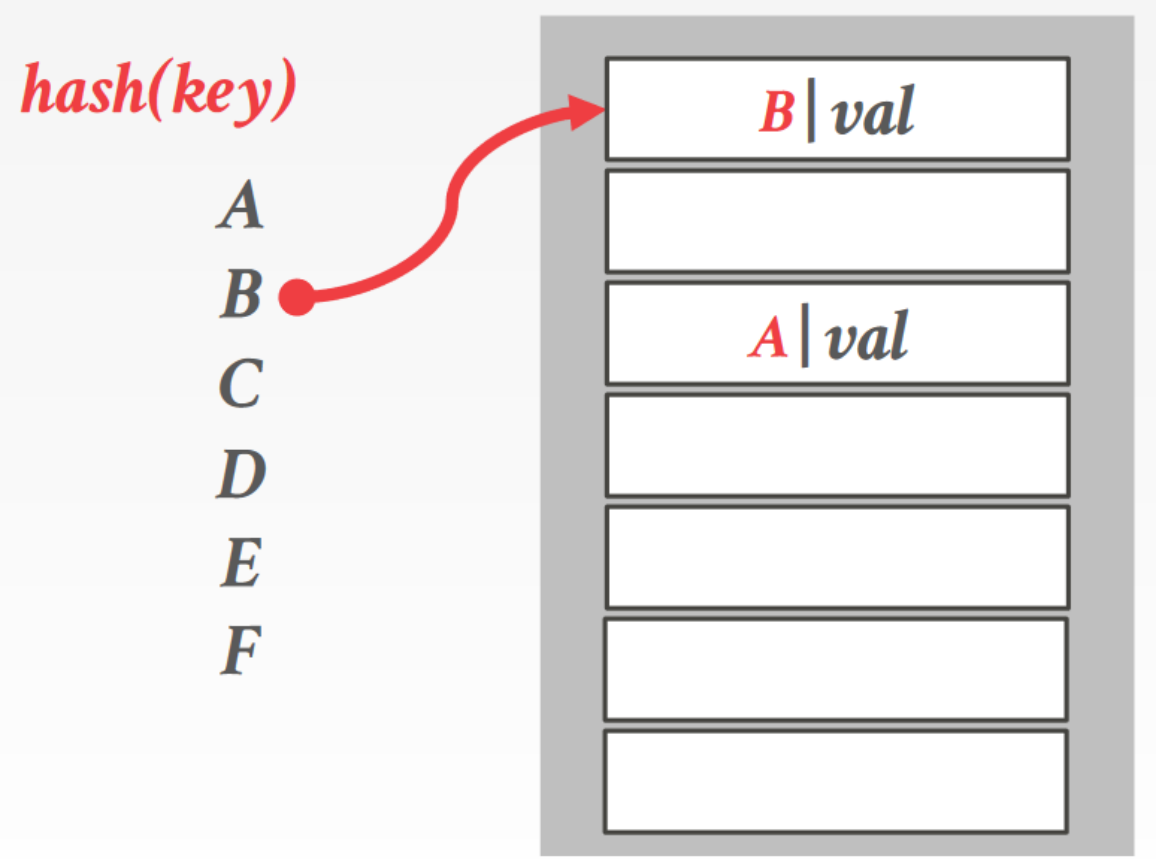
- Another extension of static hashing to handle hash collisions
- Resolve collisions by linearly searching (probing) for the next free slot in the hash table
  - Hash to the key's location and start scanning
  - When you find an empty slot, insert the record there



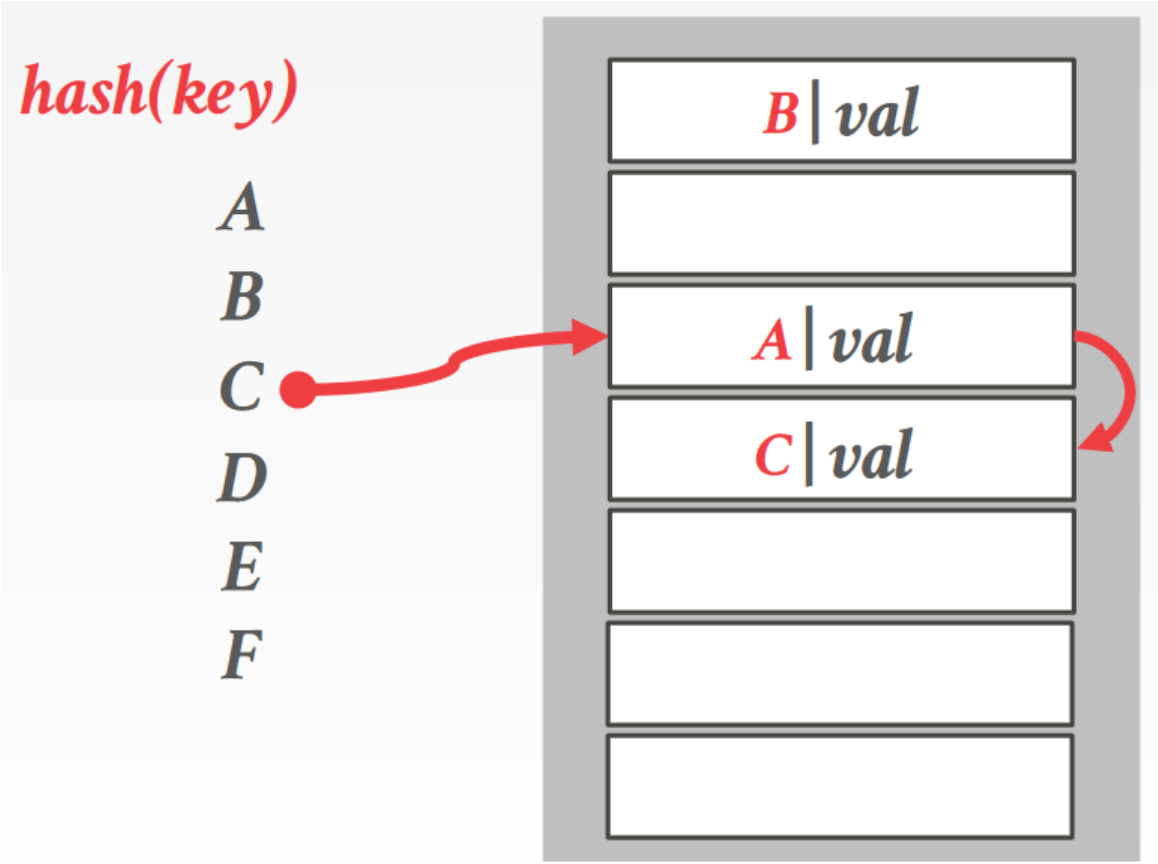
Linear Probing



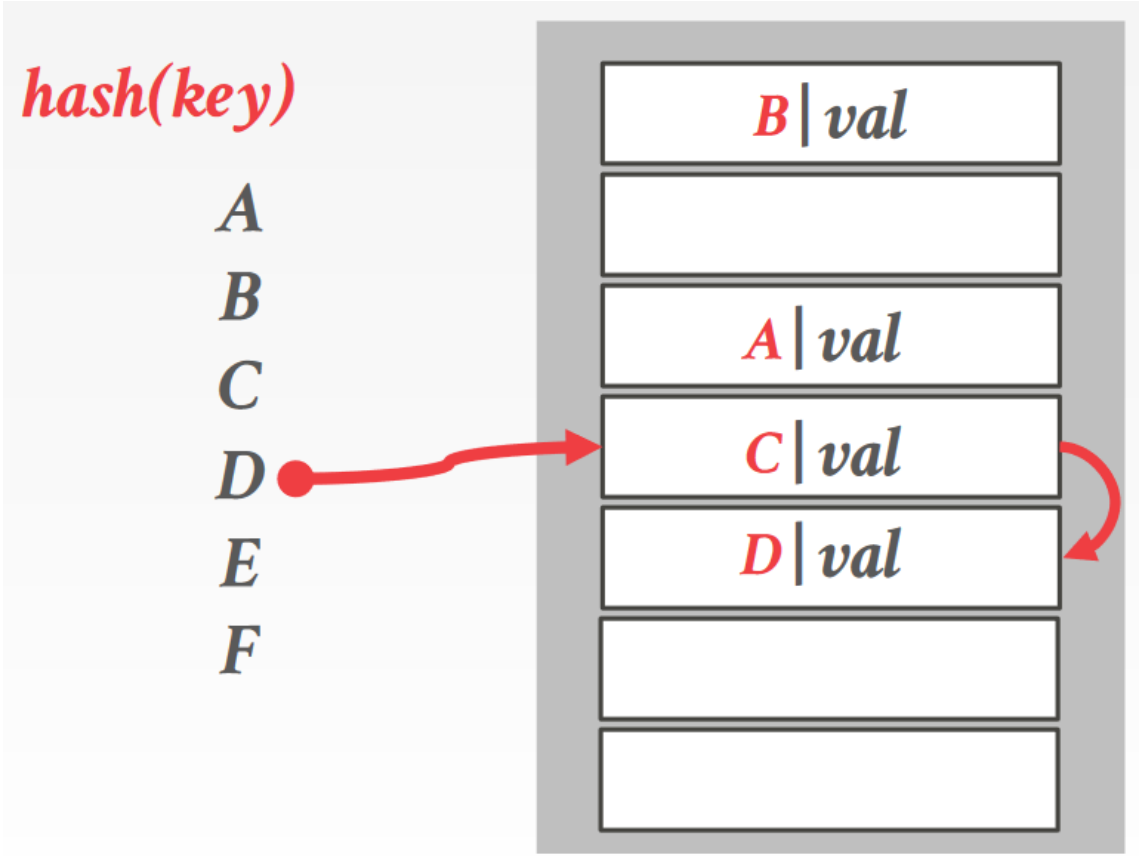
Linear Probing



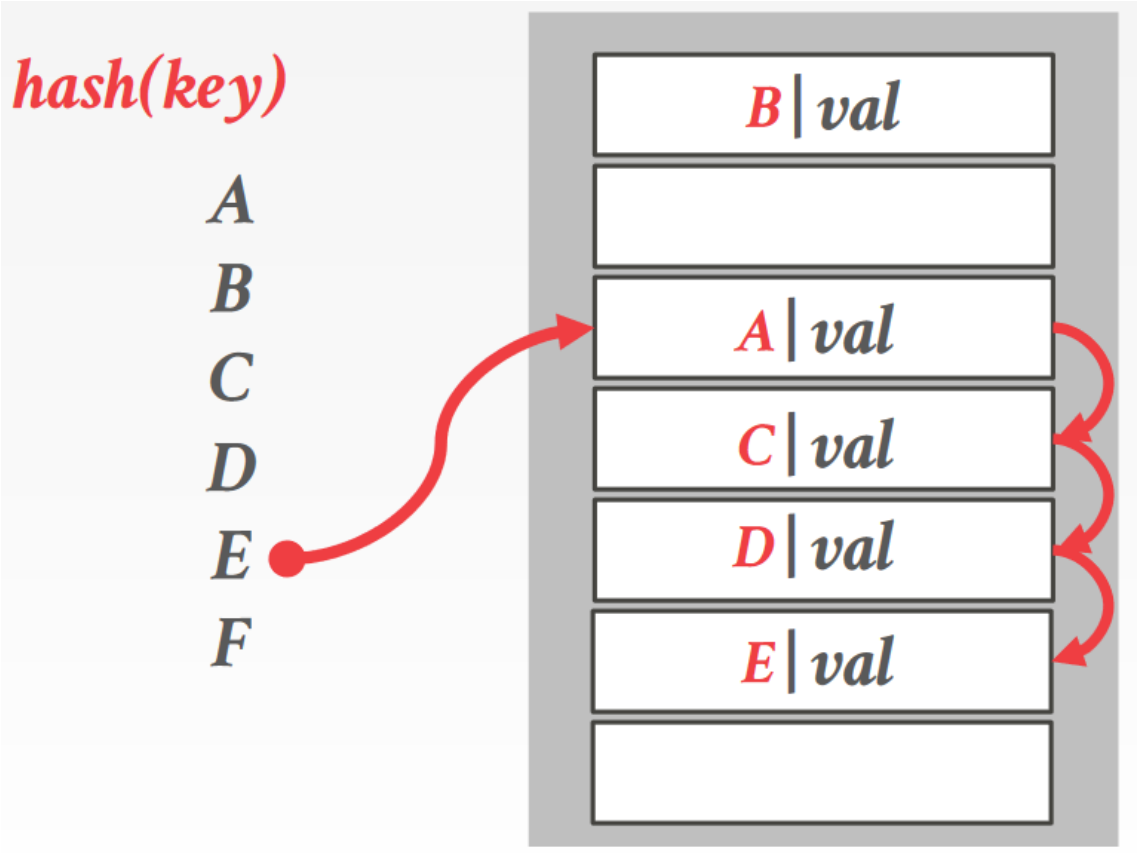
Linear Probing



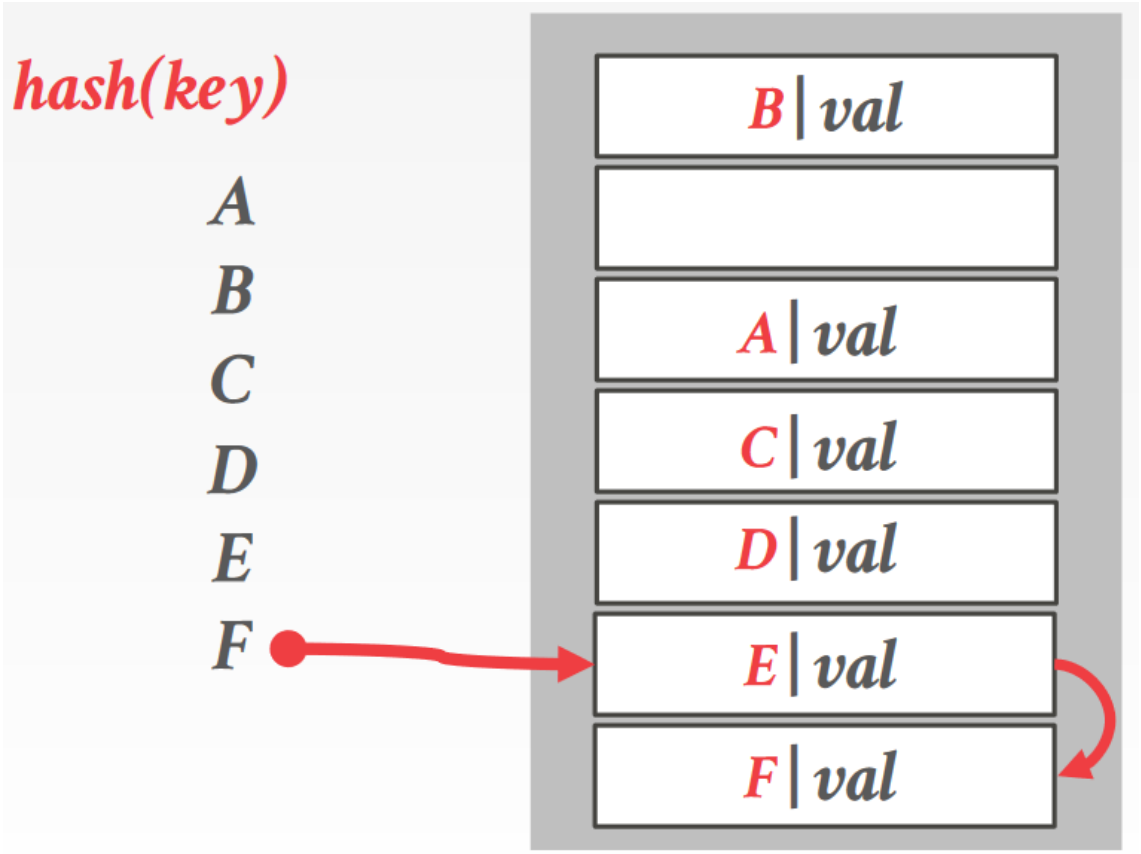
Linear Probing



Linear Probing



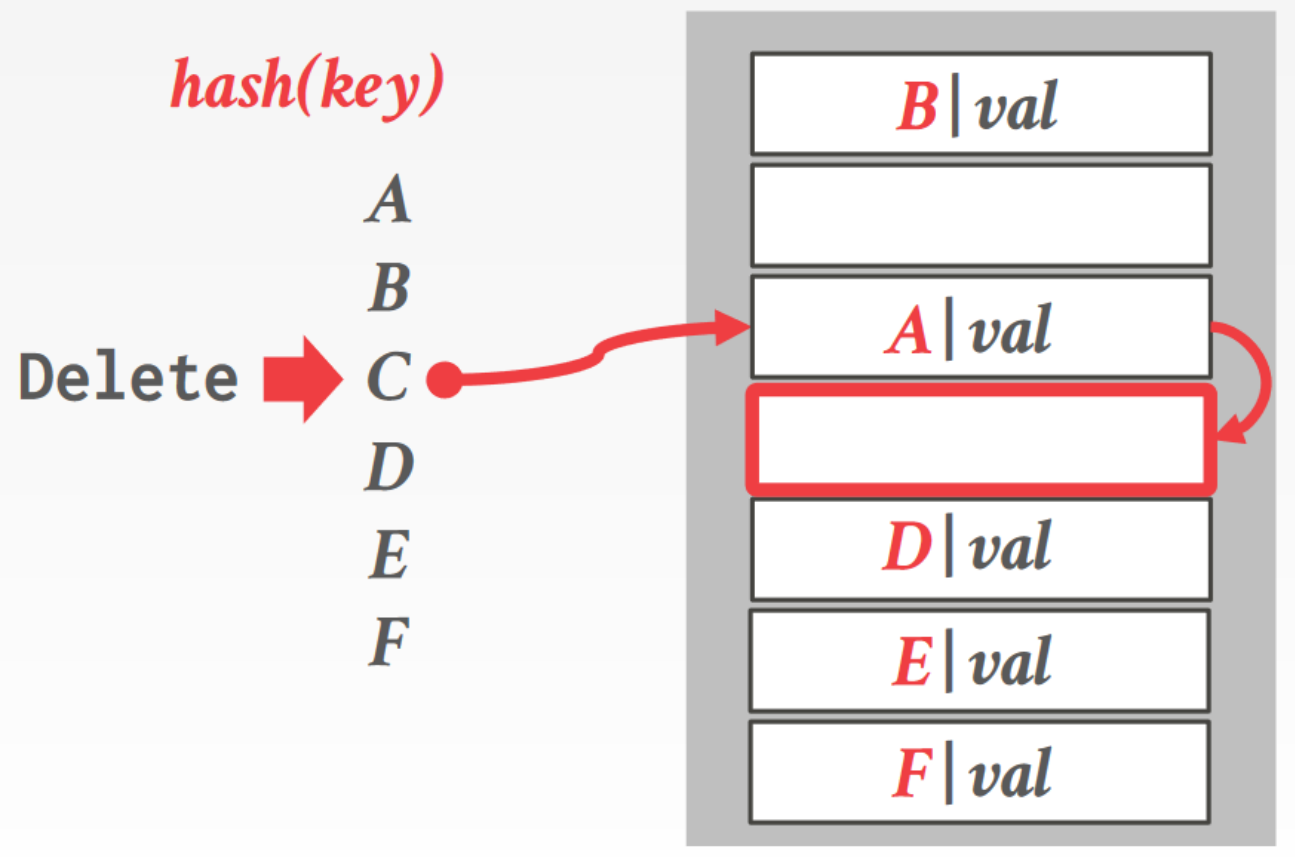
Linear Probing



How to perform Search, e.g., search for G?

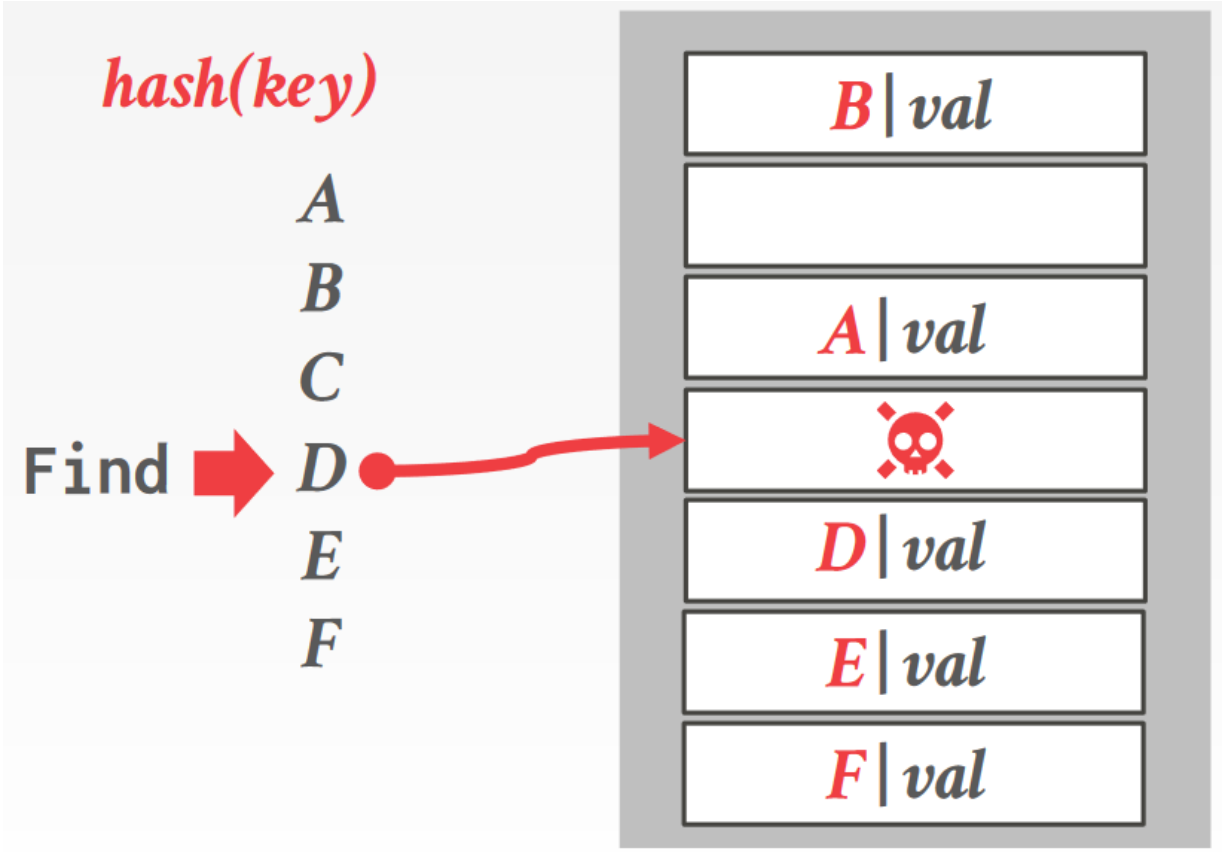
Linear Probing

- Potential problem with Delete:



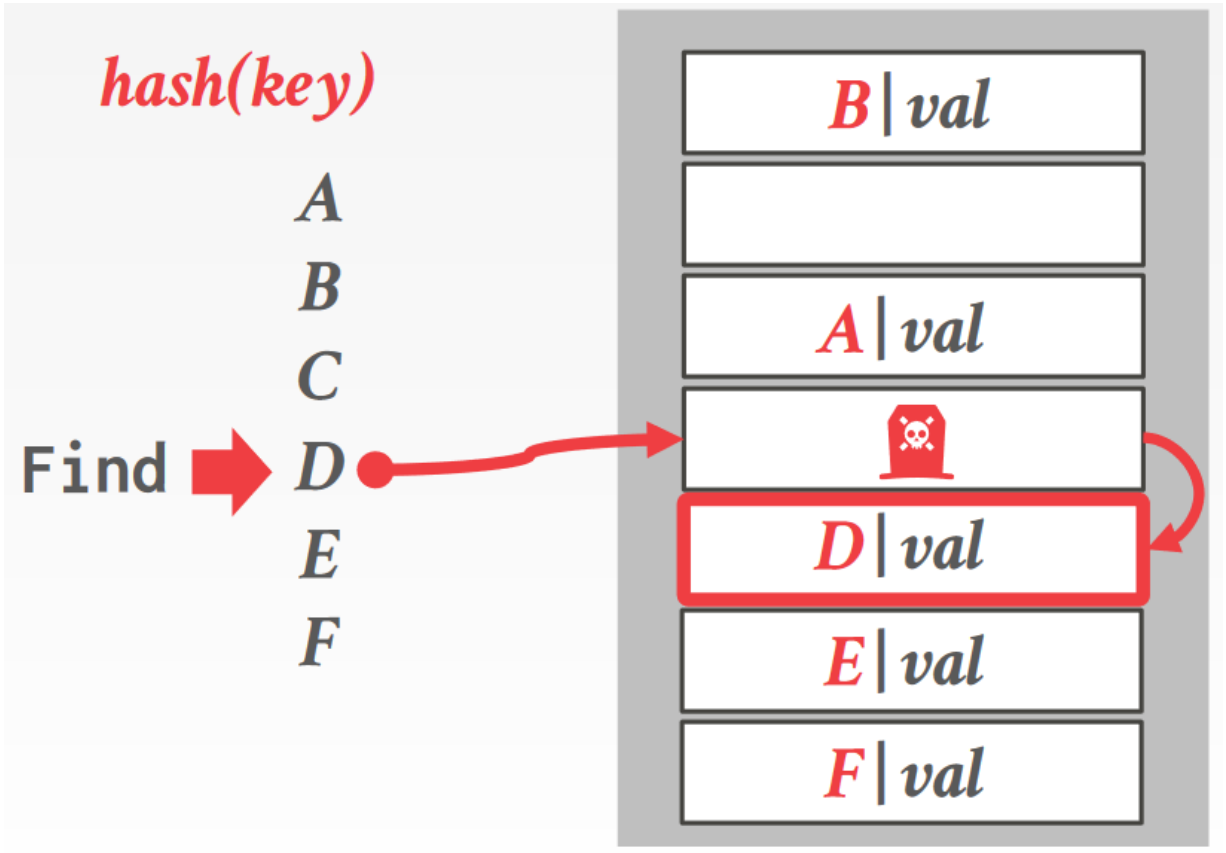
Linear Probing

- Potential problem with Delete:



Linear Probing

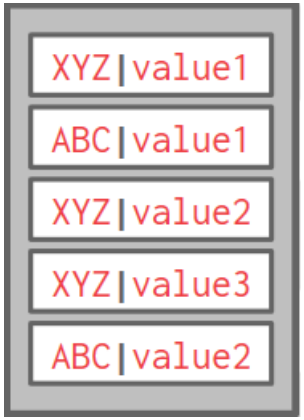
- The “Tombstone” approach to solving this problem:



Non-Unique Hash Keys

- For simplicity, we'll make the assumption that our hash keys are unique.
  - Students(Sid, Fname, Lname, DoB, Address)
  - If hash key = Sid, then our assumption holds
  - If hash key = Lname, then we have non-unique keys

- **Handling non-unique keys:** Allow for redundancy
  - Store duplicate keys' entries separately in the hash table
  - Same idea can be used in other hashing schemes as well (not just linear probing)

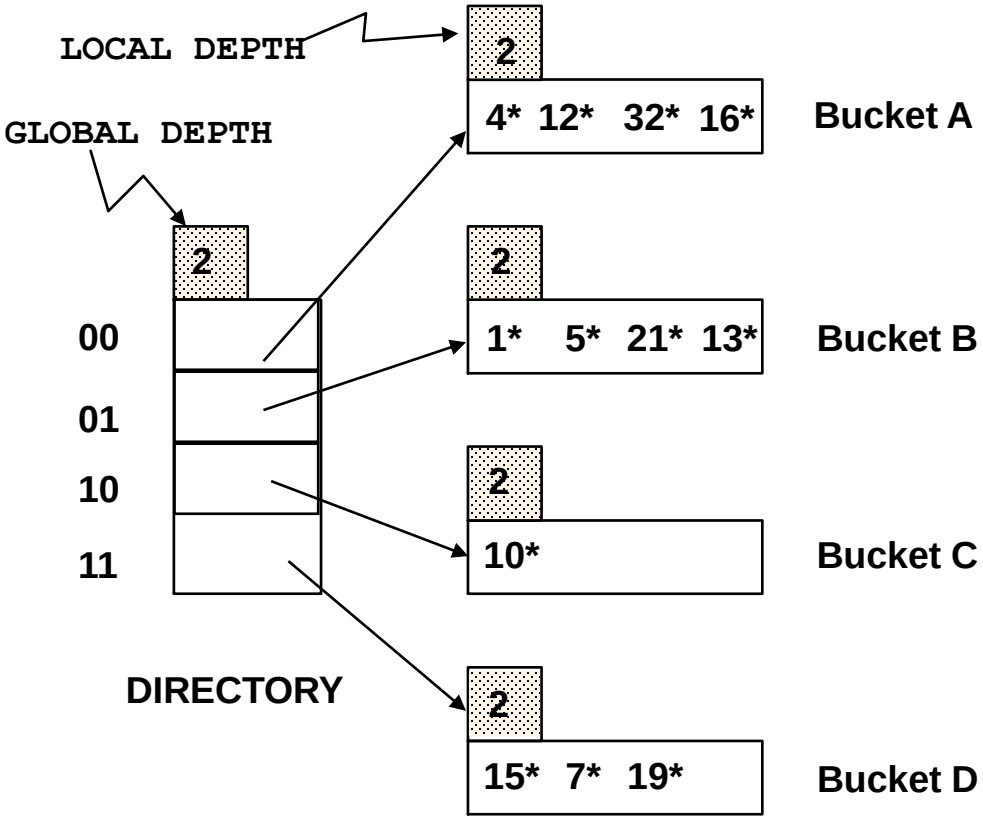


Outline

- Hashing (basics)
- Static hashing with extensions
  - Overflow chaining
  - Linear probing
- Extendible hashing
- Linear hashing

Extendible Hashing

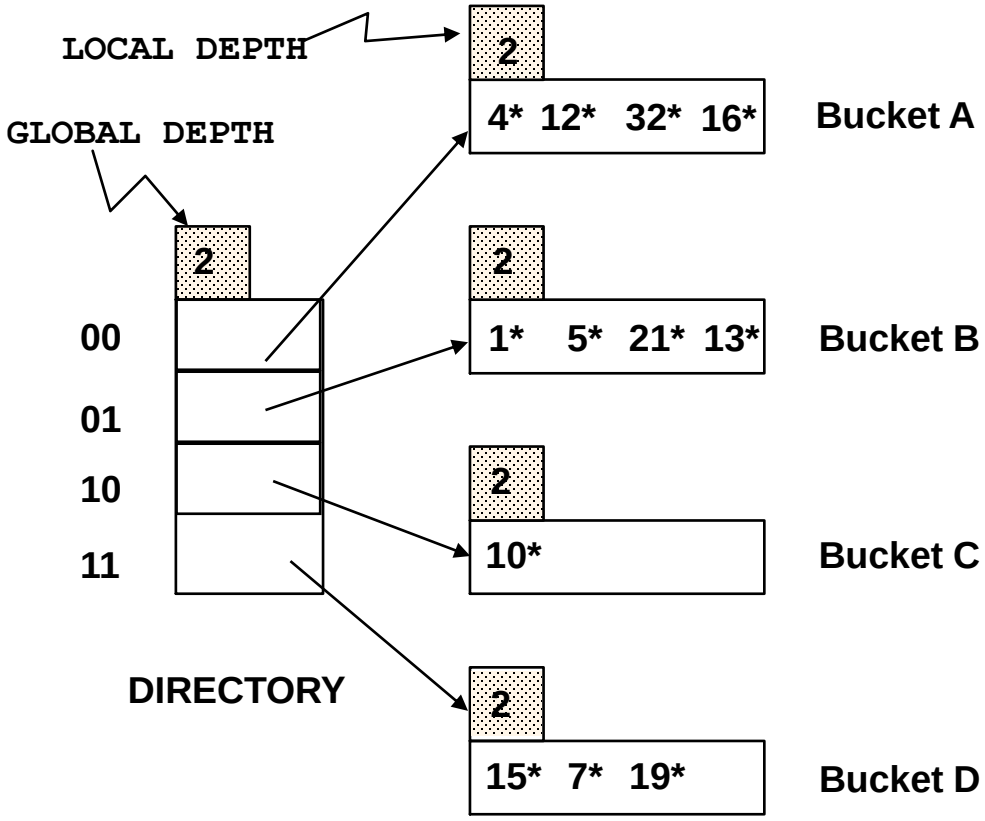
- Dynamic hashing approach
- Directory contains pointers to buckets
- To find bucket for r, take last global depth # of bits of h(r)
  - h(r) = 5 = binary 101, so it is placed in bucket pointed to by 01



- We represent r by h(r) in the figures for clarity.

Extendible Hashing

- Buckets have a capacity (here, 4).
- When a bucket is full and a new insertion arrives, the bucket must split.
- The directory may or may not have to double when a split occurs.
  - Compare global depth vs local depth

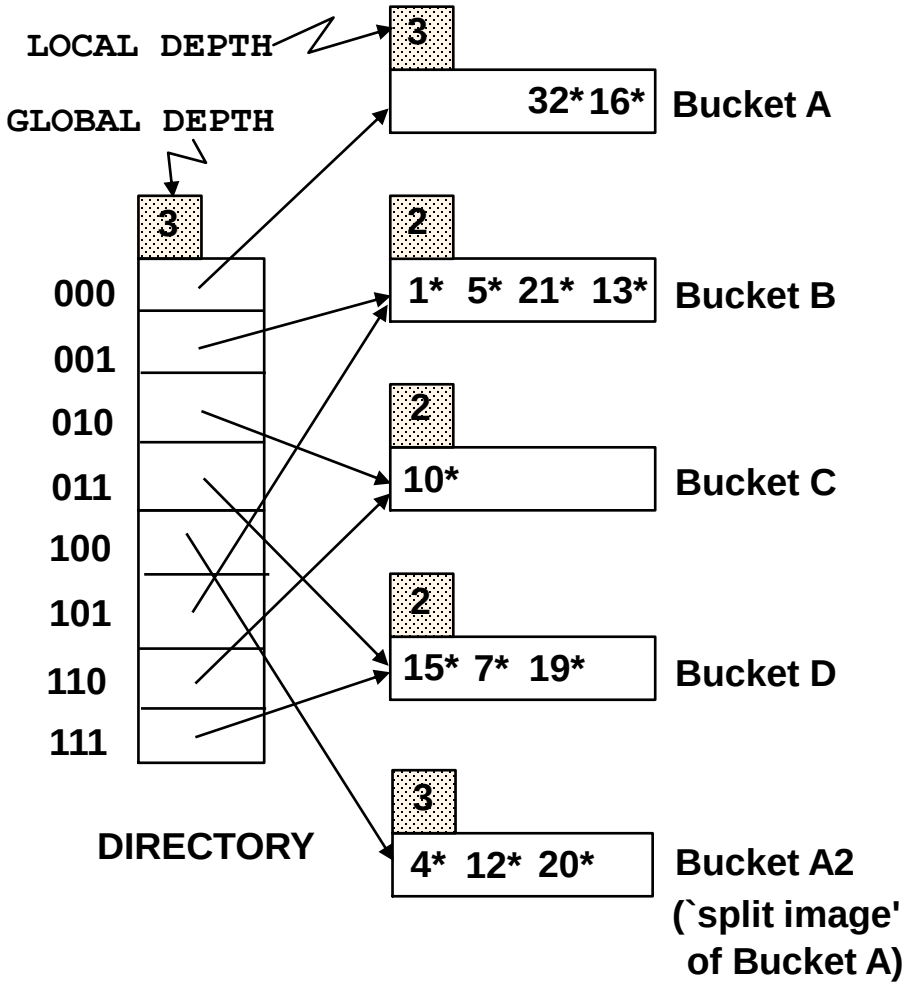


*Insert **h**(r)=20 (Causes Splitting and Doubling)*

20 is 10100 in binary format

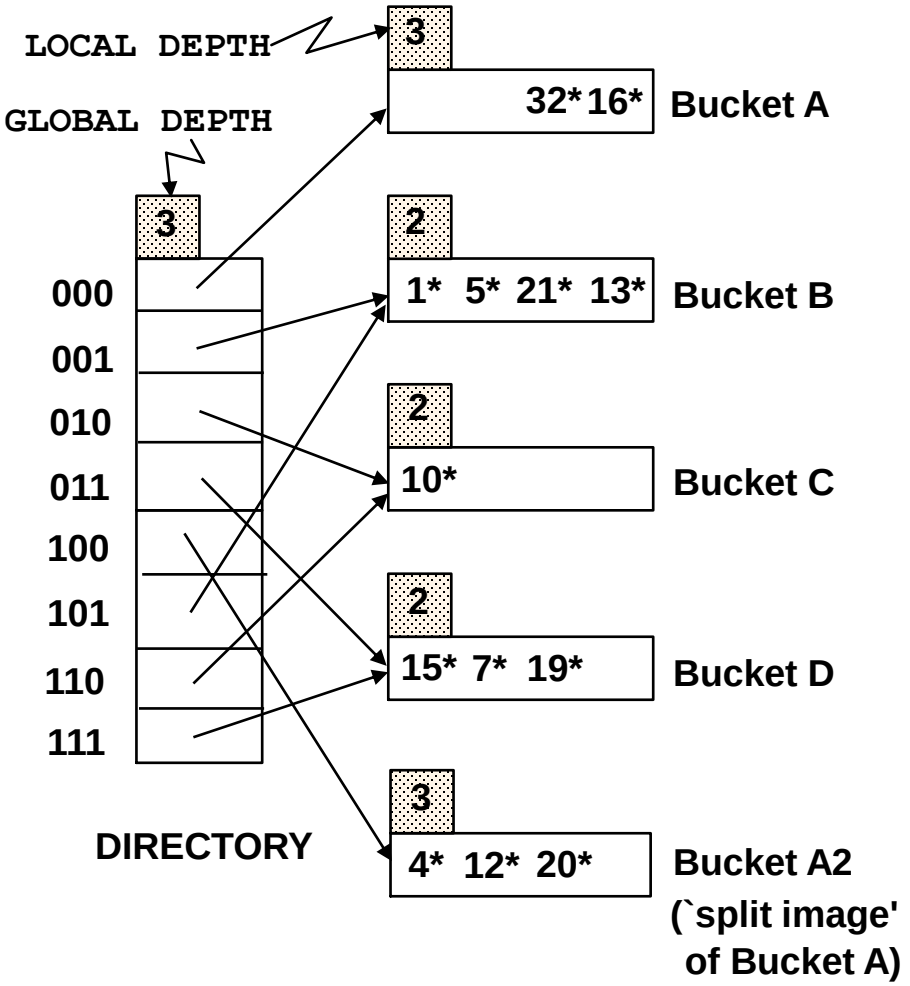
Extendible Hashing

- When does splitting cause directory doubling?
  - Local depth > global depth
- Notice that buckets B, C, D are not modified by insertion of 20
- Will inserting to bucket C cause splitting? Or directory doubling?
  - How about bucket B?



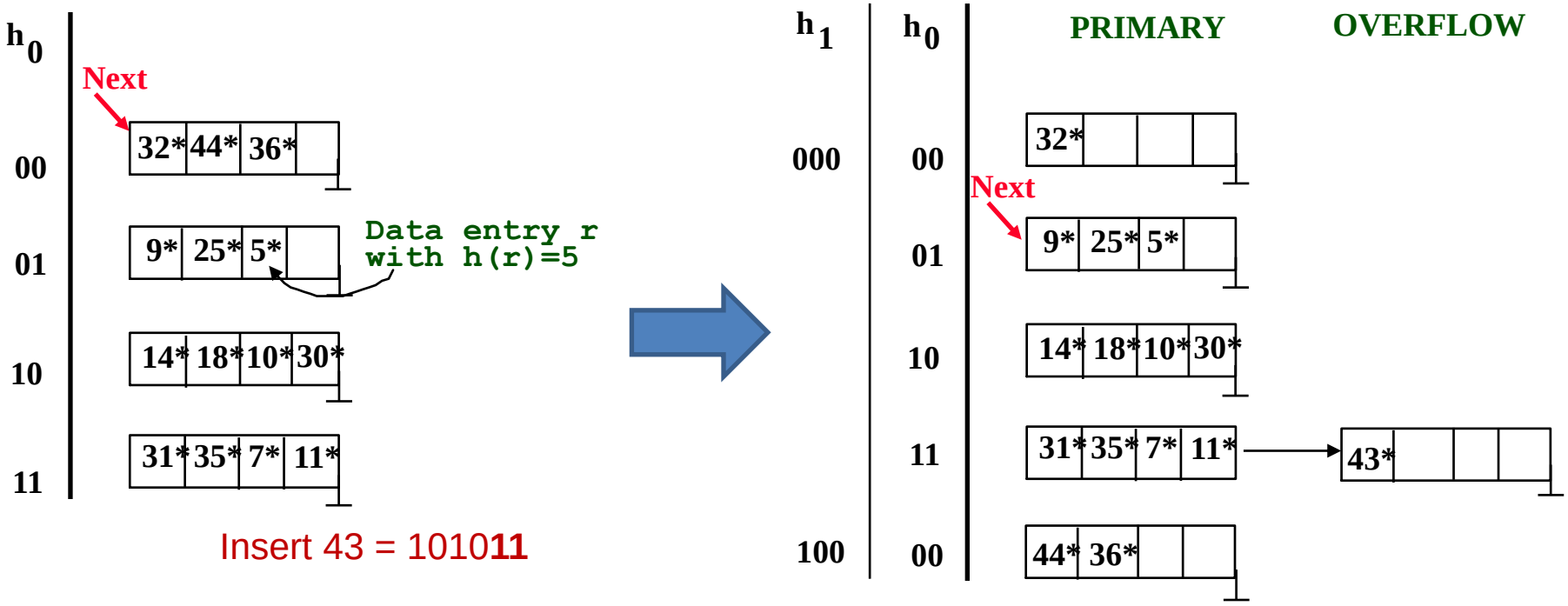
Extendible Hashing

- How about **deletions**?
  - If bucket becomes empty and has a split image, they merge
  - If each directory element points to same bucket as its split image, halve the directory



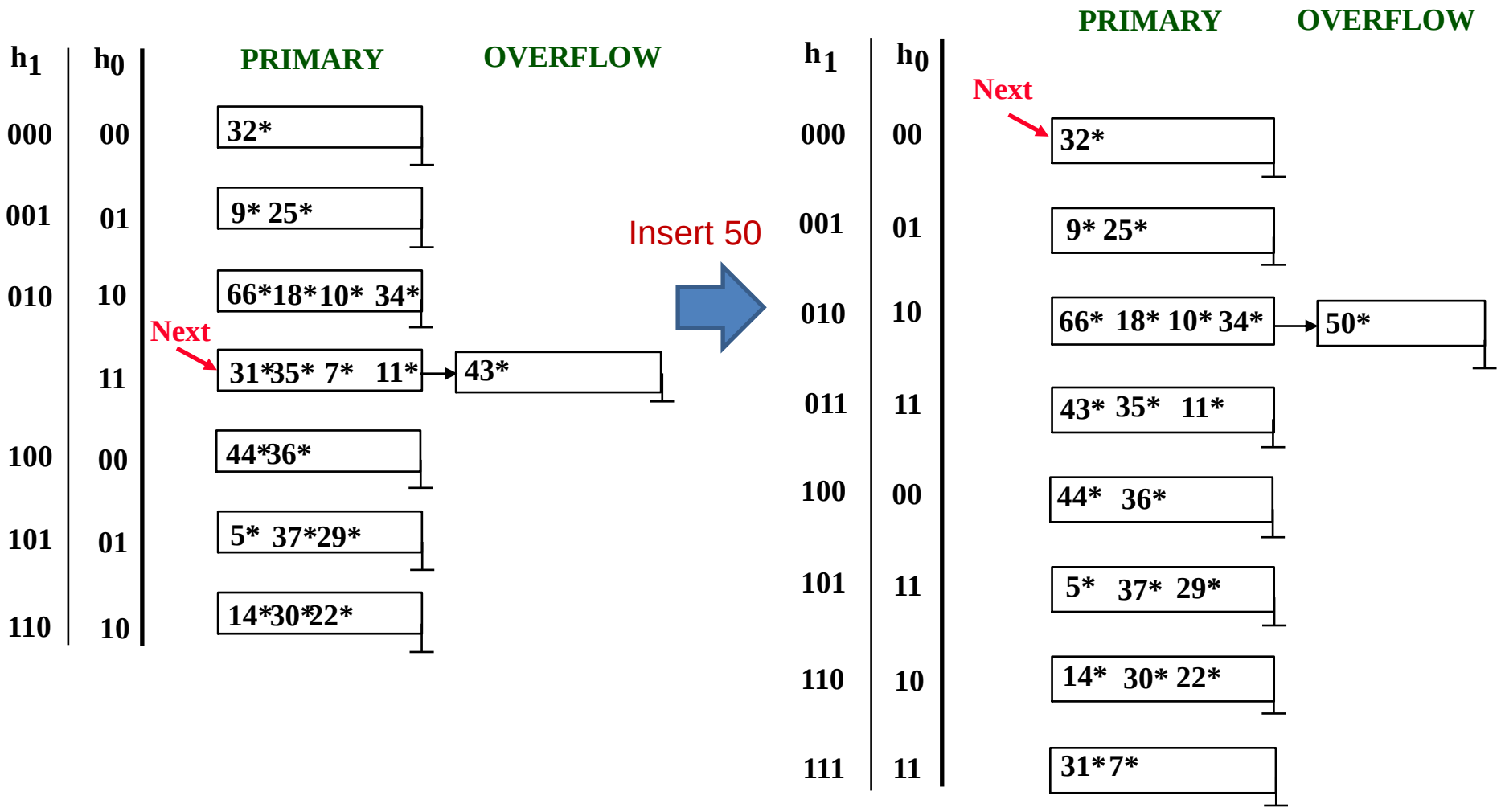
Linear Hashing

- As opposed to extendible hashing, always split the bucket that is pointed to by the "Next" pointer.
  - Splitting is not guaranteed to fix overflows
  - Next pointer is linear – will eventually get to overflows
- If no overflow, no need to split or advance Next pointer



Linear Hashing

- End of a round:



Linear Hashing

- How to handle empty buckets resulting from deletions?
  - Delete 7, then delete 31

- **Strategy #1:** Merge buckets, decrement Next pointer
  - When there's a new overflow eventually, you'll need to re-do the work and the splitting
  - You may again end up with an empty bucket

- **Strategy #2:** Do nothing (let the bucket remain empty)
  - Cannot reclaim previously allocated memory
  - (This is the strategy we will use.)

Summary

- Hash-based indexes are fast data structures that provide O(1) average time search.
  - Hash functions and hash-based indexing are also used in DBMS internals and for optimizing storage.
- Different hash-based indexing methods:
  - Static hashing with extensions: Linear probing, overflow chaining
  - Extendible hashing
  - Linear hashing
- Hash-based indexes are good for equality searches but not for range searches.
  - Next topic: Tree-based indexes, which better support range searches