HASH INDEXES

CS 564- Fall 2016

ACKs: *Dan Suciu, Jignesh Patel, AnHai Doan*

HASH INDEXES

- efficient for equality search
- not appropriate for range search

- Types of hash indexes:
  - static hashing
  - extendible (dynamic) hashing

STATIC HASHING

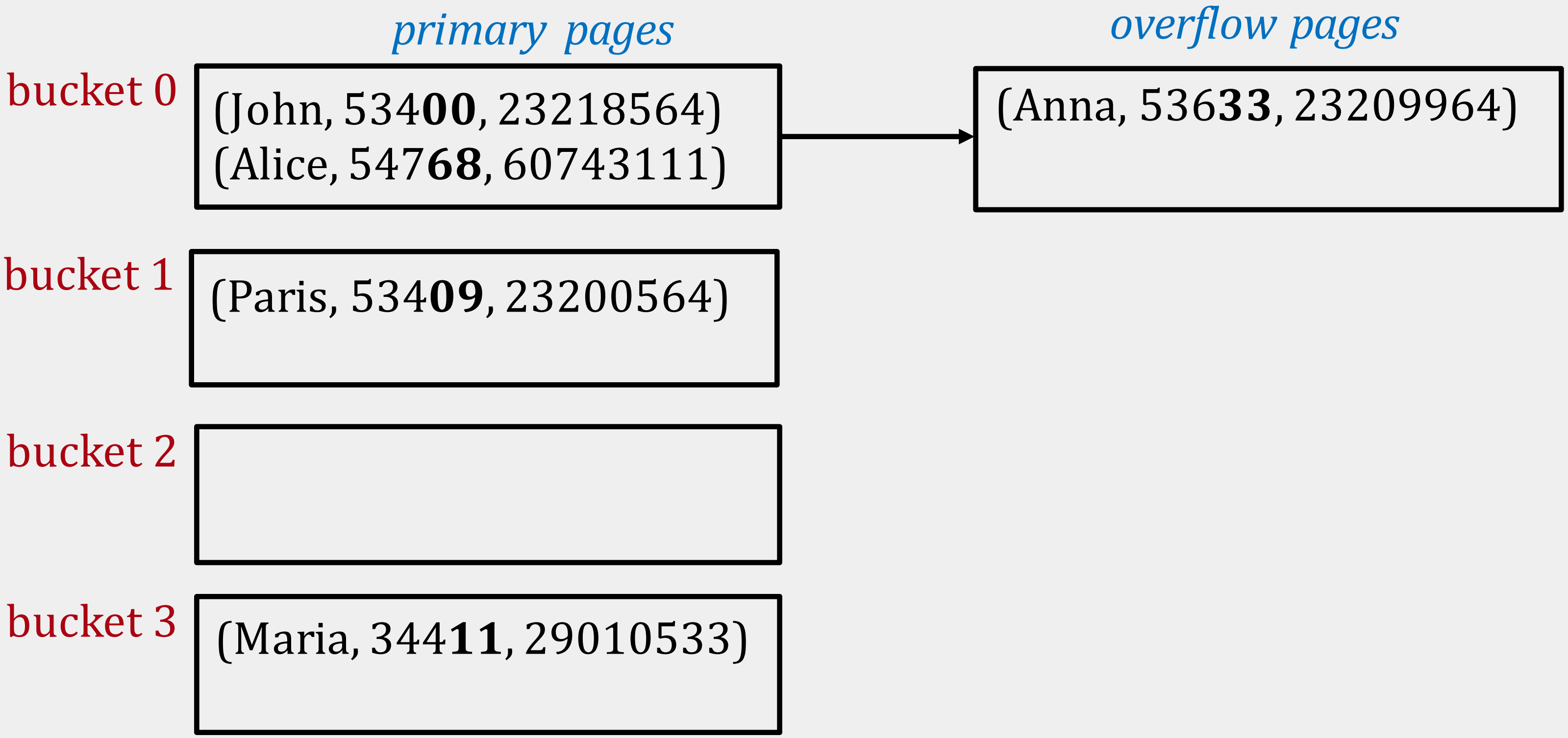
- A **hash index** is a collection of *buckets*
  - bucket = primary page + overflow pages
  - each bucket contains one or more data entries
- To find the bucket for each record, we use a hash function ***h*** applied on the search key *k*
  - ***N*** = number of buckets
  - ***h****(k) mod* ***N*** = bucket in which the data entry belongs
- Records with different search key may belong in the same bucket

Static Hashing: Example

`Person(name,zipcode,phone)`

- *search key*: zipcode
- *hash function* ***h***: last 2 digits

- 4 buckets
- each bucket has 2 data entries (full record)



primary pages

overflow pages

bucket 0

(John, 534**00**, 23218564)
(Alice, 547**68**, 60743111)

(Anna, 536**33**, 23209964)

bucket 1

(Paris, 534**09**, 23200564)

bucket 2

bucket 3

(Maria, 344**11**, 29010533)

Operations on Hash Indexes

- **Equality search**
  - apply the hash function on the search key to locate the appropriate bucket
  - search through the primary page (plus overflow pages) to find the record(s)
- **Deletion**
  - find the appropriate bucket, delete the record
- **Insertion**
  - find the appropriate bucket, insert the record
  - if there is no space, create a new overflow page

Hash Functions

- An *ideal* hash function must be uniform: each bucket is assigned the same number of key values
- A *bad* hash function maps all search key values to the same bucket
- Examples of good hash functions:
  - $\boldsymbol{h}(k) = a * k + b$, where a and b are constants
  - a random function

BUCKET OVERFLOW

- Bucket *overflow* can occur because of
  - insufficient number of buckets
  - *skew* in distribution of records
    - many records have the same search-key value
    - the hash function results in a non-uniform distribution of key values

- Bucket overflow is handled using *overflow buckets*

Problems of Static Hashing

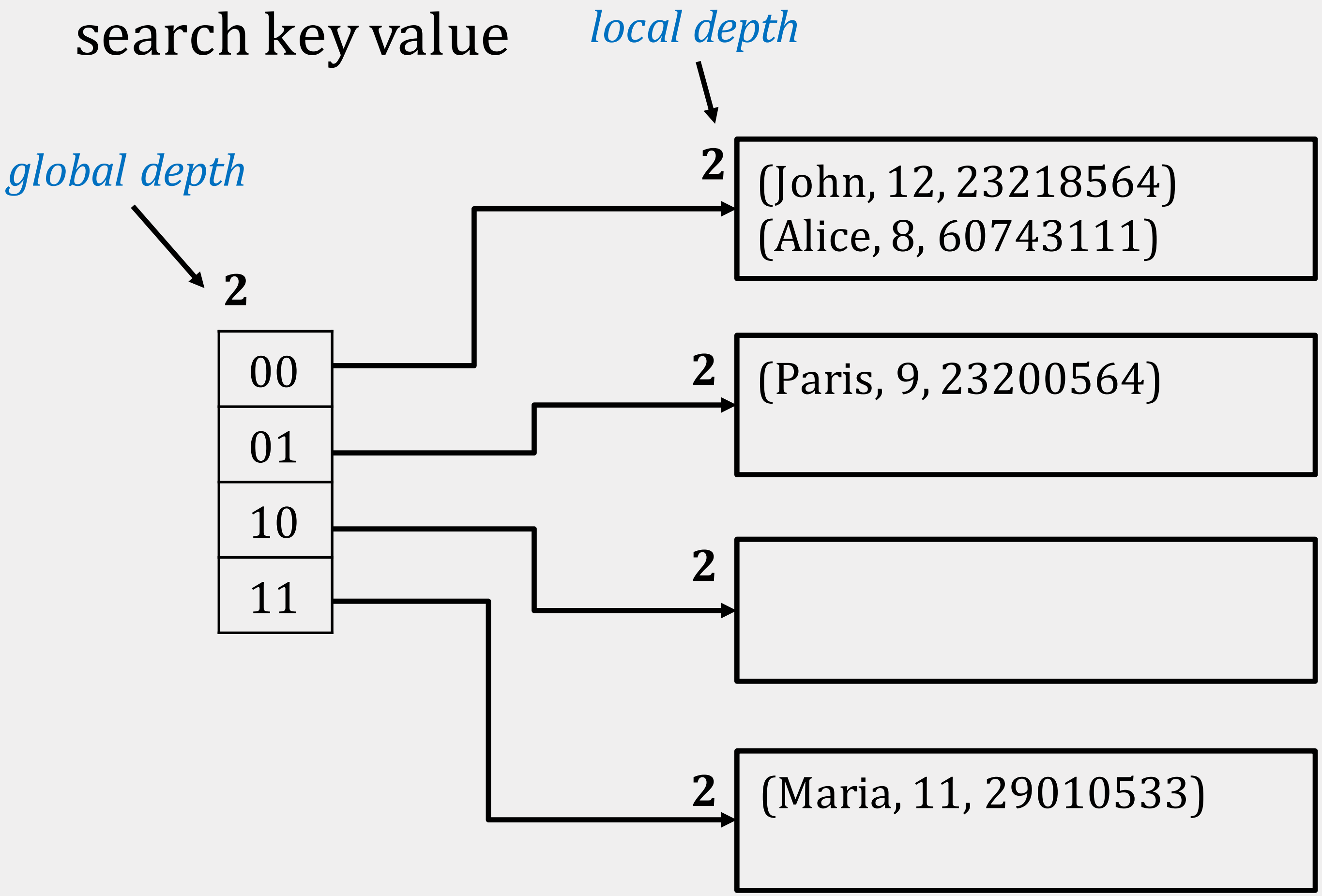
- In static hashing, there is a **fixed** number of buckets in the index
- Issues with this:
  - if the database grows, the number of buckets will be too small: long overflow chains degrade performance
  - if the database shrinks, space is wasted
  - reorganizing the index is expensive and can block query execution

EXTENDIBLE HASHING

- **Extendible hashing** is a type of *dynamic* hashing
- It keeps a directory of pointers to buckets
- On overflow, it reorganizes the index by doubling the directory (and not the number of buckets)

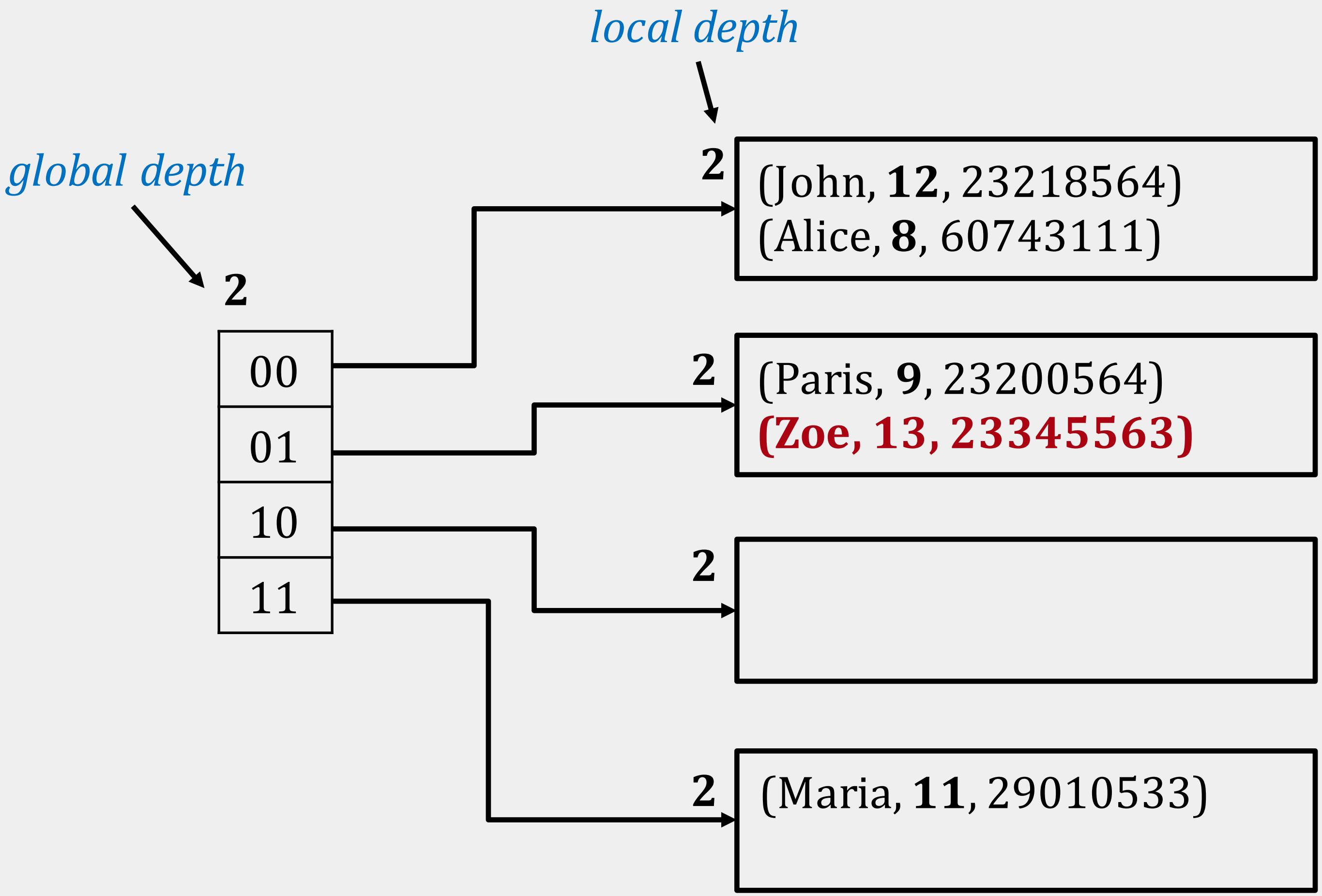
Extendible Hashing

To search, use the last **2** digits of the **binary** form of the search key value



Extendible Hashing: Insert

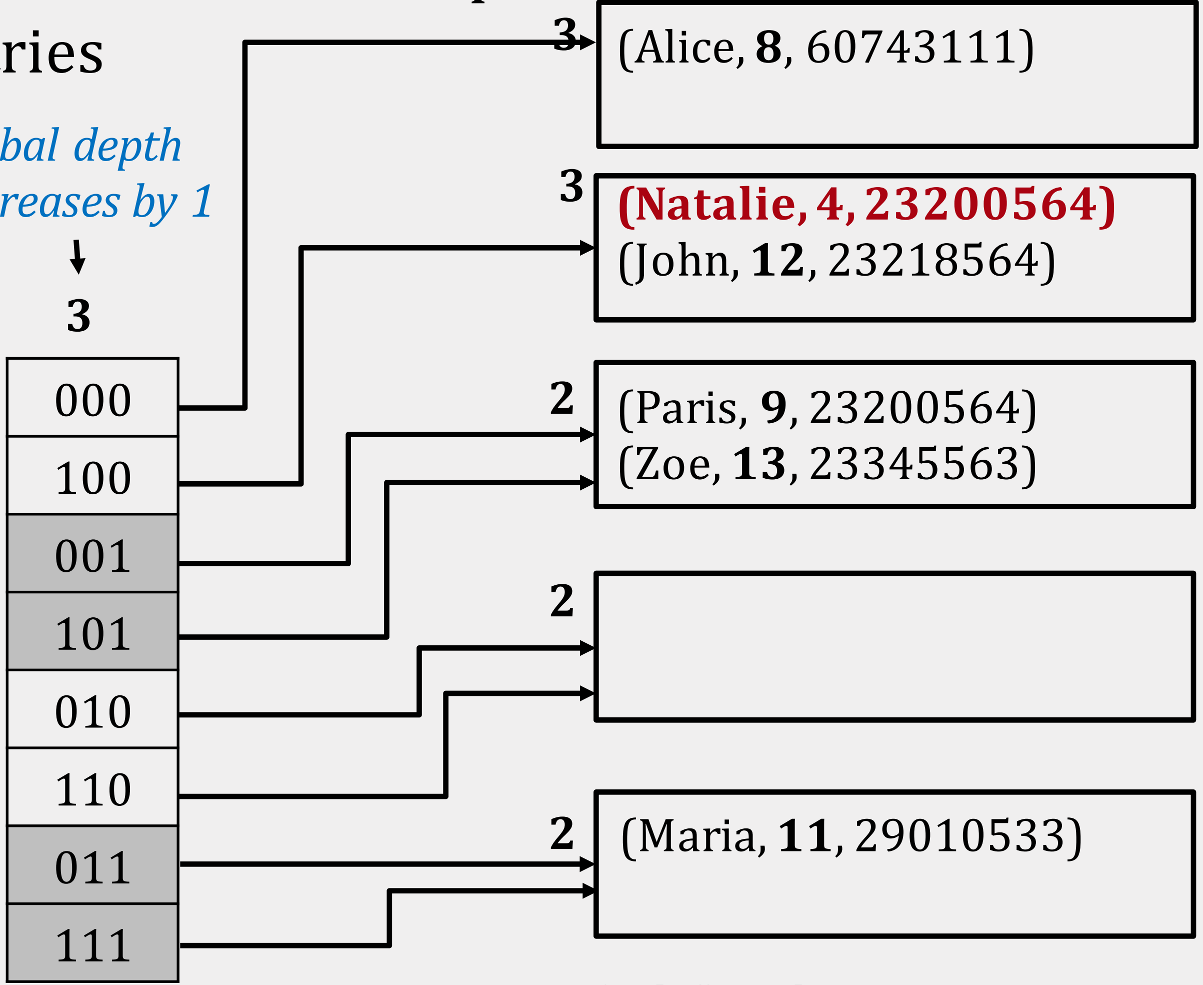
If there is space in the bucket, simply add the record



EXTENDIBLE HASHING: INSERT

If the bucket is full, split the bucket and redistribute the entries

global depth increases by 1



3

000

100

001

101

010

110

011

111

3 (Alice, **8**, 60743111)

3 **(Natalie, 4, 23200564)** (John, **12**, 23218564)

2 (Paris, **9**, 23200564) (Zoe, **13**, 23345563)

2

2 (Maria, **11**, 29010533)

local depth increases for the split bucket!

local depth remains the same for the other buckets

Extendible Hashing: Delete

- Locate the bucket of the record and remove it
- If the bucket becomes empty, it can be removed (and update the directory)
- Two buckets can also be coalesced together if the sum of the entries fit in a single bucket
- Decreasing the size of the directory can also be done, but it is expensive

More On Extendible Hashing

- How many disk accesses for equality search?
  - One if directory fits in memory, else two
- Directory grows in spurts, and, if the distribution of hash values is skewed, the directory can grow very large
- We may need overflow pages when multiple entries have the same hash