CAS CS 460/660
Introduction to Database Systems

Indexing: Hashing

Introduction

- *Hash-based* indexes are best for *equality selections*. *Cannot* support range searches.
- Static and dynamic hashing techniques exist; trade-offs similar to ISAM vs. B+ trees.

- *Recall, 3 alternatives for data entries* k*:
  1. Data record with key value **k**
  2. <**k**, rid of data record with search key value **k**>
  3. <**k**, list of rids of data records w/search key **k**>

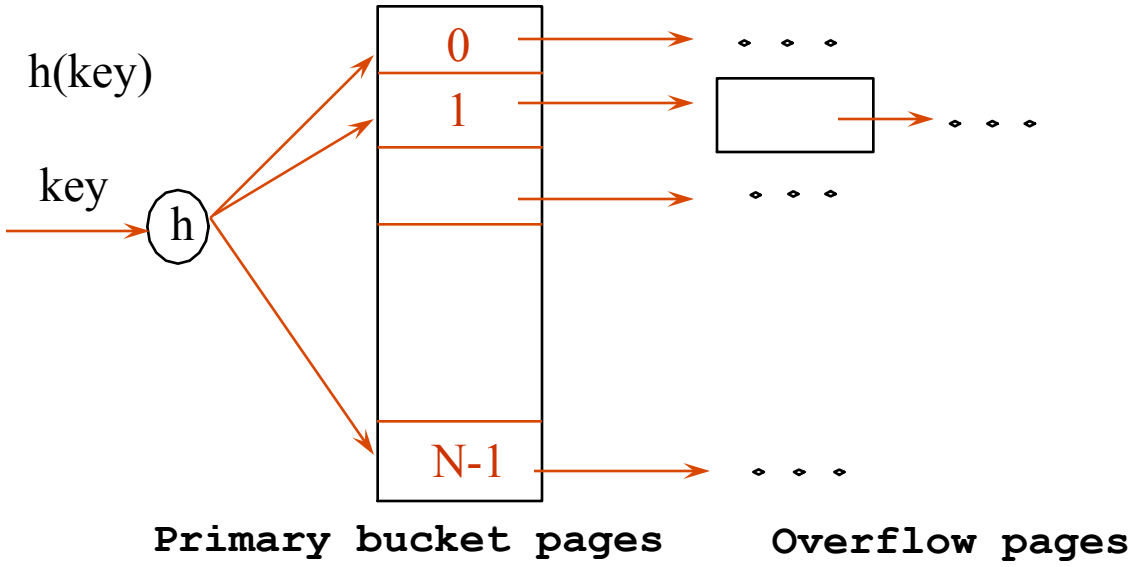
  Choice is orthogonal to the *indexing technique*

Static Hashing

- # primary pages fixed, allocated sequentially, never de-allocated; overflow pages if needed.
- A simple hash function (for N buckets):

  h(*k*) = k MOD N

  is bucket # where data entry with key *k* belongs.



Static Hashing (Contd.)

- Buckets contain *data entries*.
- Hash fn works on *search key* field of record *r.* Use MOD N to distribute values over range 0 ... N-1.
  - **h**(*key*) = key MOD N works well for uniformly distributed data.
    - better: h(key) = (A*key MOD P) mod N, where P is a prime number
  - various ways to tune **h** for non-uniform (checksums, crypto, etc.).
- As with any static structure: Long overflow chains can develop and degrade performance.
  - *Extendible* and *Linear Hashing*: Dynamic techniques to fix this problem.

Extendible Hashing

- Situation: Bucket (primary page) becomes full.
  - Want to avoid overflow pages
- Add more buckets (i.e., increase “N”)?
  - Okay, but need a new hash function!
- D*oubling #* of buckets makes this easier
  - Say N values are powers of 2: how to do “mod N”?
  - What happens to hash function when double “N”?
- Problems with Doubling
  - Don’t want to have to double the size of the file.
  - Don’t want to have to move all the data.

Extendible Hashing (cont)

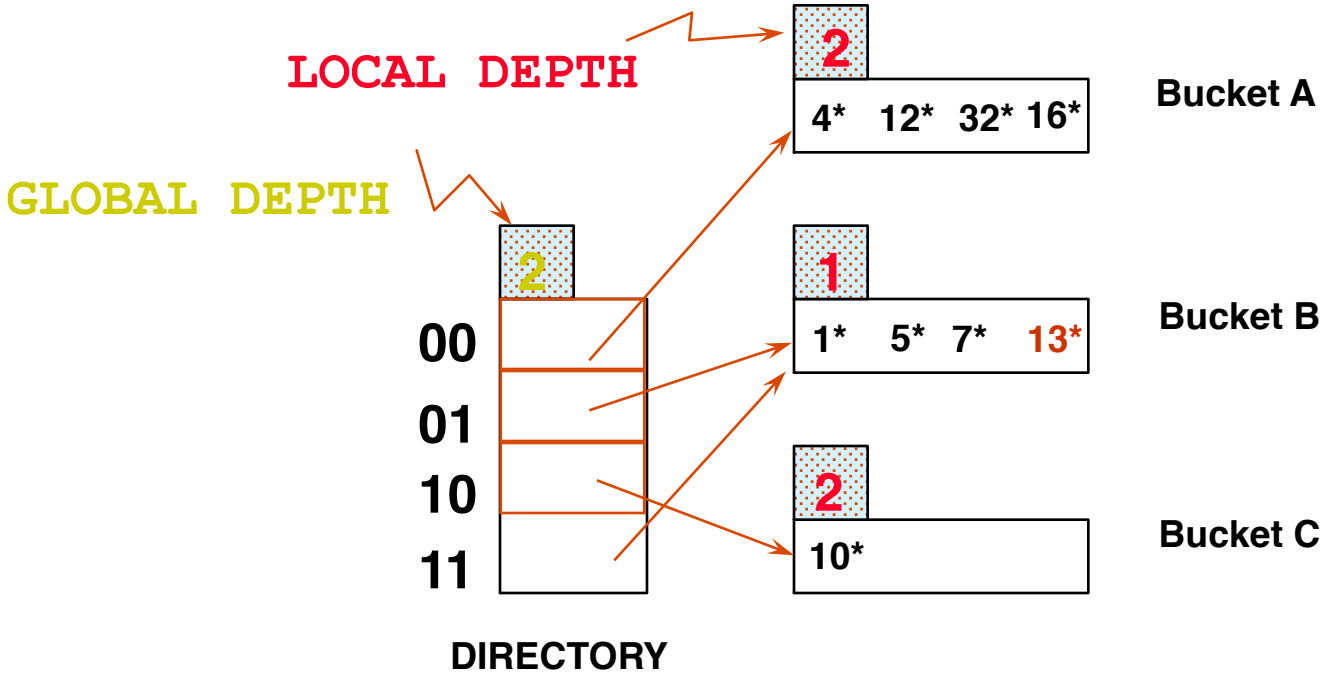
- *Idea*: Add a level of indirection!

- Use *directory of pointers to buckets*,
- Double # of buckets by *doubling the directory*
  - Directory much smaller than file, so doubling it is much cheaper.

- Split only the bucket that just overflowed!
  - *No overflow pages*!
  - Trick lies in how hash function is adjusted!

How it Works

- **Directory is array of size 4, so 2 bits needed.**
- **Bucket for record *r* has entry with index = `*global depth*' least significant bits of** h(***r***);
  - If **h**(*r*) = 5 = binary 101, it is in bucket pointed to by 01.
  - If **h**(*r*) = 7 = binary 111, it is in bucket pointed to by 11.

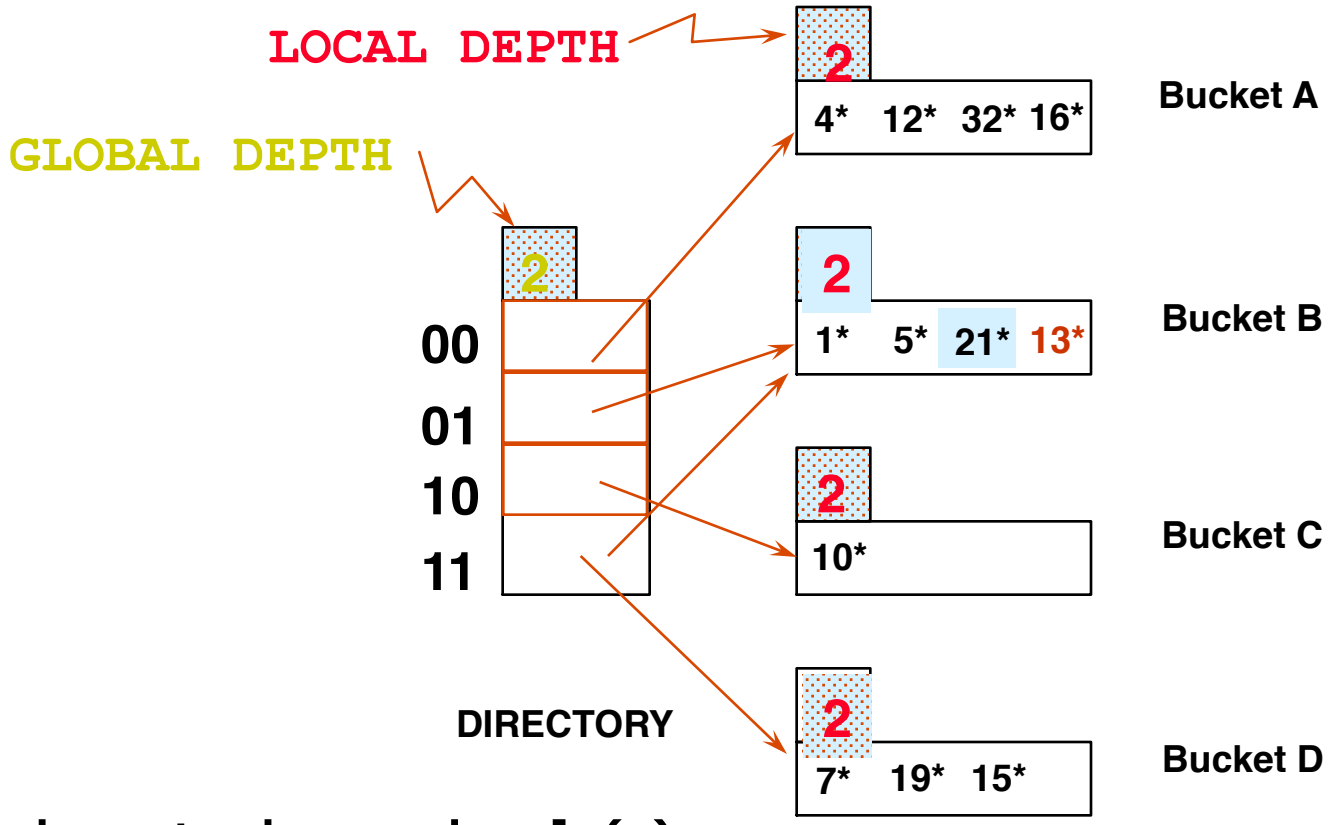


Handling Inserts

- Find bucket where record belongs.
- If there's room, put it there.
- Else, if bucket is full, *split* it:
  - increment local depth of original page
  - allocate new page with new local depth
  - re-distribute records from original page.
  - add entry for the new page to the directory

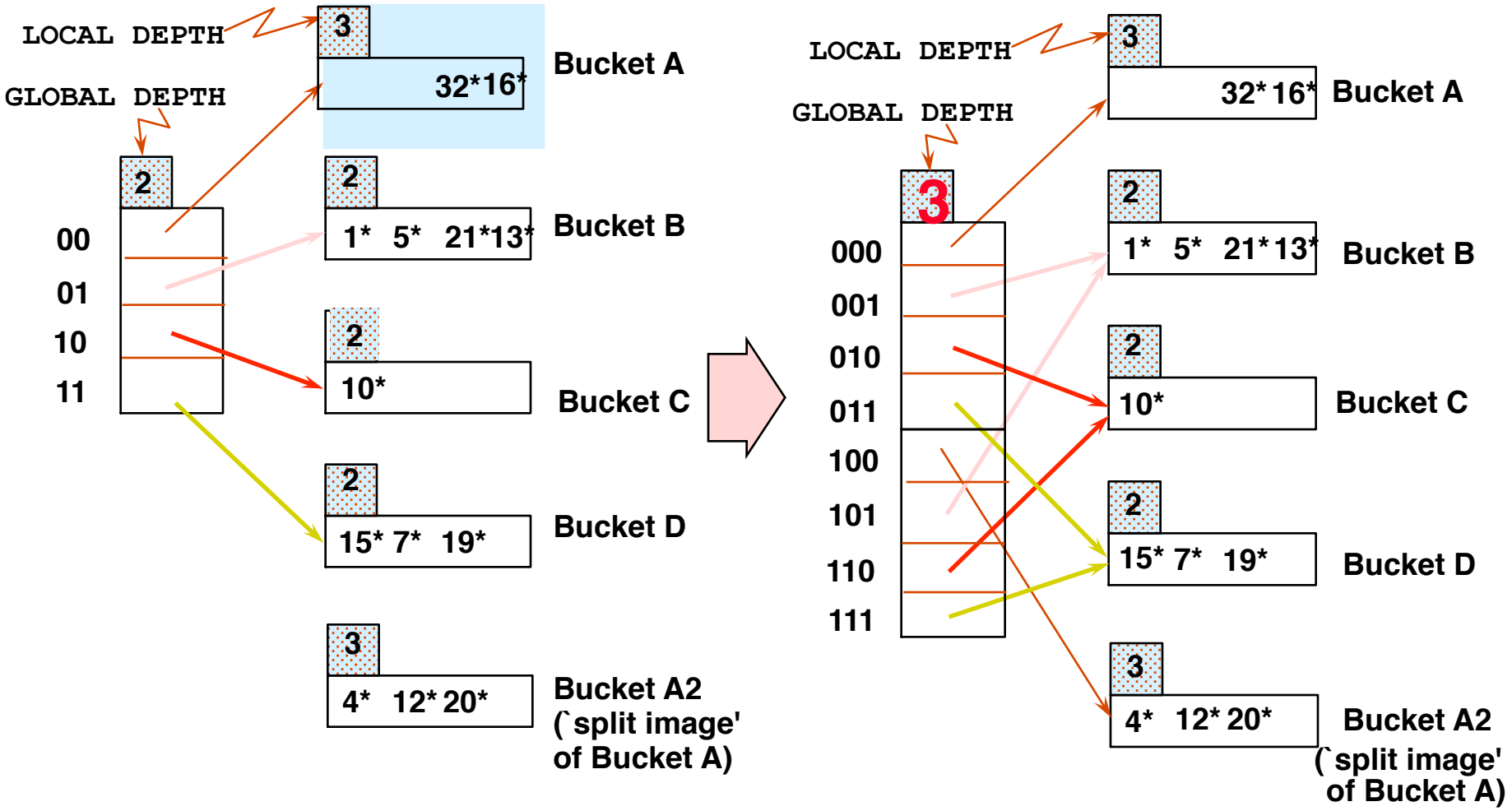
Example: Insert 21,19, 15

- 21 = 10101
- 19 = 10011
- 15 = 01111



we denote key *r* by **h**(*r*).

Insert h(r)=20 (Causes Doubling)



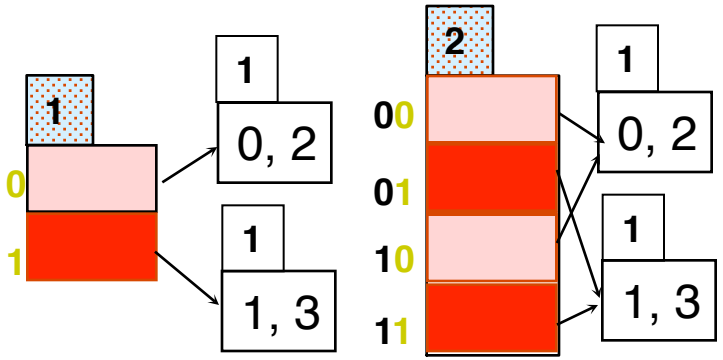
Points to Note

- 20 = binary 10100. Last 2 bits (00) tell us *r* in either A or A2. Last 3 bits needed to tell which.
  - *Global depth of directory*: Max # of bits needed to tell which bucket an entry belongs to.
  - *Local depth of a bucket*: # of bits used to determine if an entry belongs to this bucket.
- When does split cause directory doubling?
  - Before insert, *local depth* of bucket = *global depth*. Insert causes *local depth* to become > *global depth*; directory is doubled by *copying it over* and `fixing' pointer to split image page.

Directory Doubling

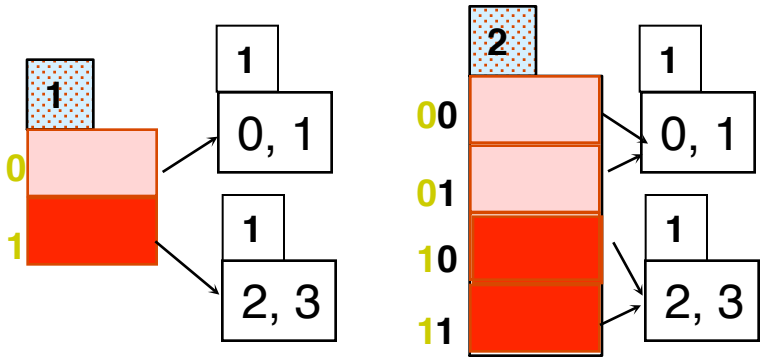
Why use least significant bits in directory (instead of the *most* significant ones)?

Allows for doubling by copying the directory and appending the new copy to the original.



Least Significant

vs.



Most Significant

Comments on Extendible Hashing

- If directory fits in memory, equality search answered with one disk access; else two.
  - 100MB file, 100 bytes/rec, 4K pages contains 1,000,000 records (as data entries) and 25,000 directory elements; chances are high that directory will fit in memory.
  - Directory grows in spurts, and, if the distribution *of hash values* is skewed, directory can grow large.
  - Multiple entries with same hash value cause problems!

Comments on Extendible Hashing

Delete:

- If removal of data entry makes bucket empty, can be merged with `split image'

- If each directory element points to same bucket as its split image, can halve directory.

Summary

- Hash-based indexes: best for equality searches, cannot support range searches.
- Static Hashing can have long overflow chains.
- Extendible Hashing avoids overflow pages by splitting a full bucket when a new data entry is to be added to it. (*Duplicates may require overflow pages.*)
  - Directory to keep track of buckets, doubles periodically.
  - Can get large with skewed data; additional I/O if this does not fit in main memory.