Hash-Based Indexes

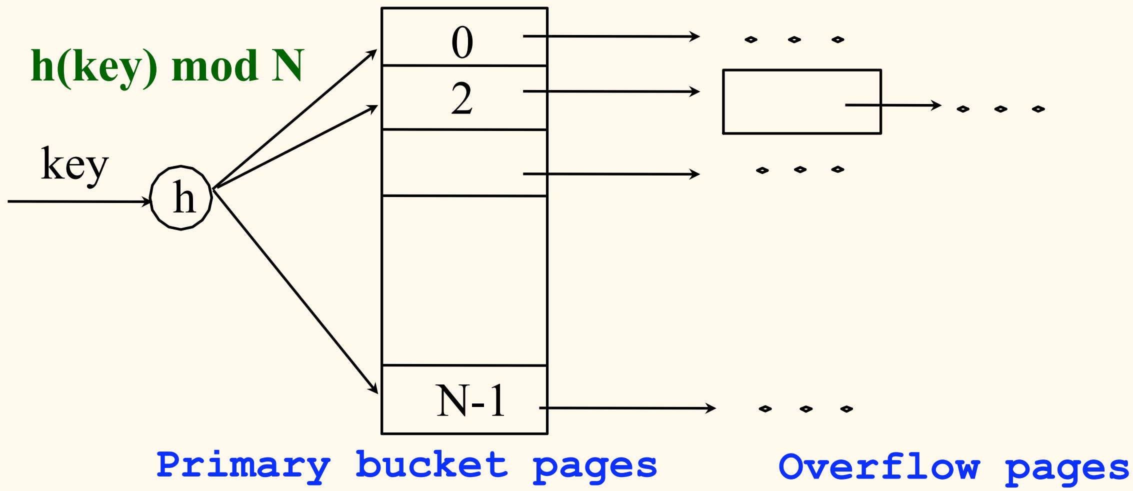
UMass Amherst

Fall 2008

Introduction

- *As for any index, 3 alternatives for data entries* **k***:
  - Data record with key value **k**
  - <**k**, rid of data record with search key value **k**>
  - <**k**, list of rids of data records with search key **k**>
  - Choice orthogonal to the *indexing technique*
- *Hash-based* indexes are best for *equality selections*. ***Cannot*** support range searches.
- Static and dynamic hashing techniques exist; trade-offs for dynamic data

Static Hashing



- **h**(*k*) mod N = bucket to which data entry with key *k* belongs. k1≠k2 can lead to the same bucket.
- Static: # buckets (N) fixed
  - main pages allocated sequentially, never de-allocated;
  - overflow pages if needed.

Static Hashing (Contd.)

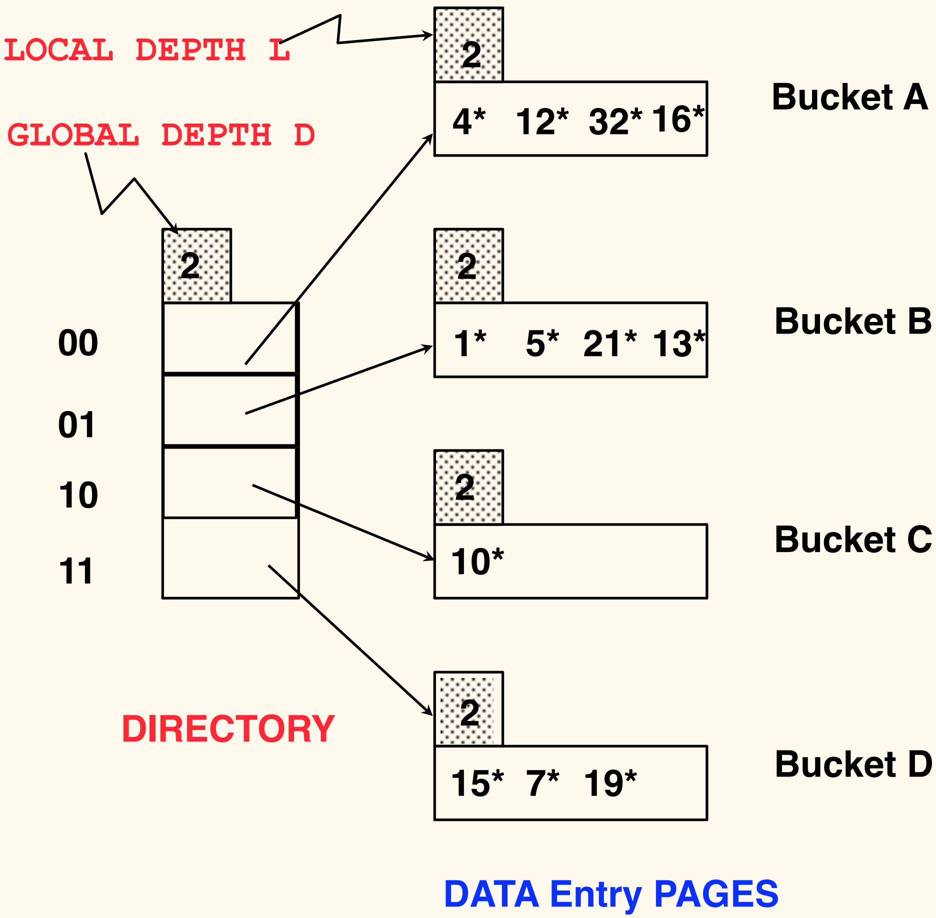
- Hash fn works on *search key* field of record *r.* Must distribute values over range 0 ... N-1.
  - **h**(*key*) mod N = (a * *key* + b) mod N usually works well.
  - a and b are constants; lots known about how to tune **h**.
- Buckets contain *data entries*.
- Long overflow chains can develop and degrade performance.
  - *Extendible* and *Linear Hashing*: Dynamic techniques to fix this problem.

Extendible Hashing

- Situation: Bucket (primary page) becomes full. Why not re-organize file by *doubling* # of buckets?
  - Reading and writing all pages is expensive!
  - *Idea*: Use *directory of pointers to buckets*, double # of buckets by (1) *doubling the directory,* (2) splitting just the bucket that overflowed!
  - Directory much smaller than file, so doubling it is much cheaper. Only one page of data entries is split. *No overflow page*!
  - Trick lies in how hash function is adjusted!

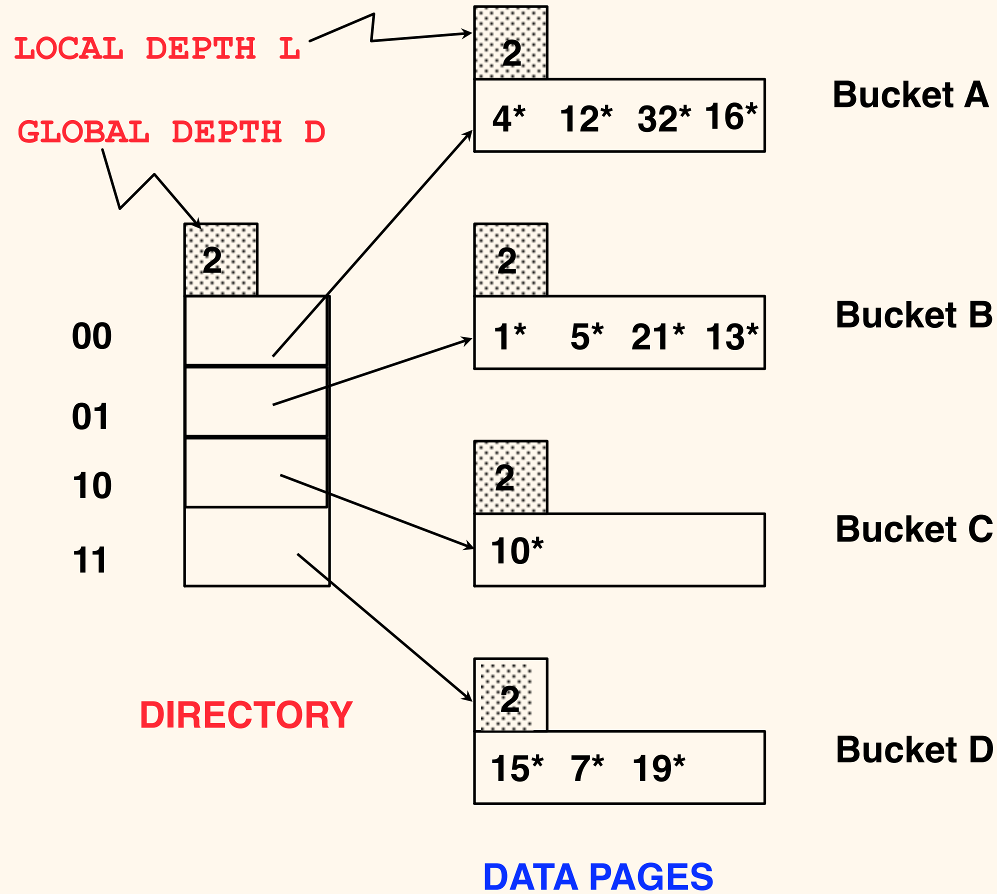
Example

- Directory is array of size 4, *global depth* D = 2.
- Each bucket has *local depth* L ($L \leq D$)
- To find bucket for *r*, (1) get **h**(*r*), (2) take last `*global depth*' # bits of **h**(*r*).
  - If **h**(*r*) = 5 = binary 101,
  - Take last 2 bits, go to bucket pointed to by 01.



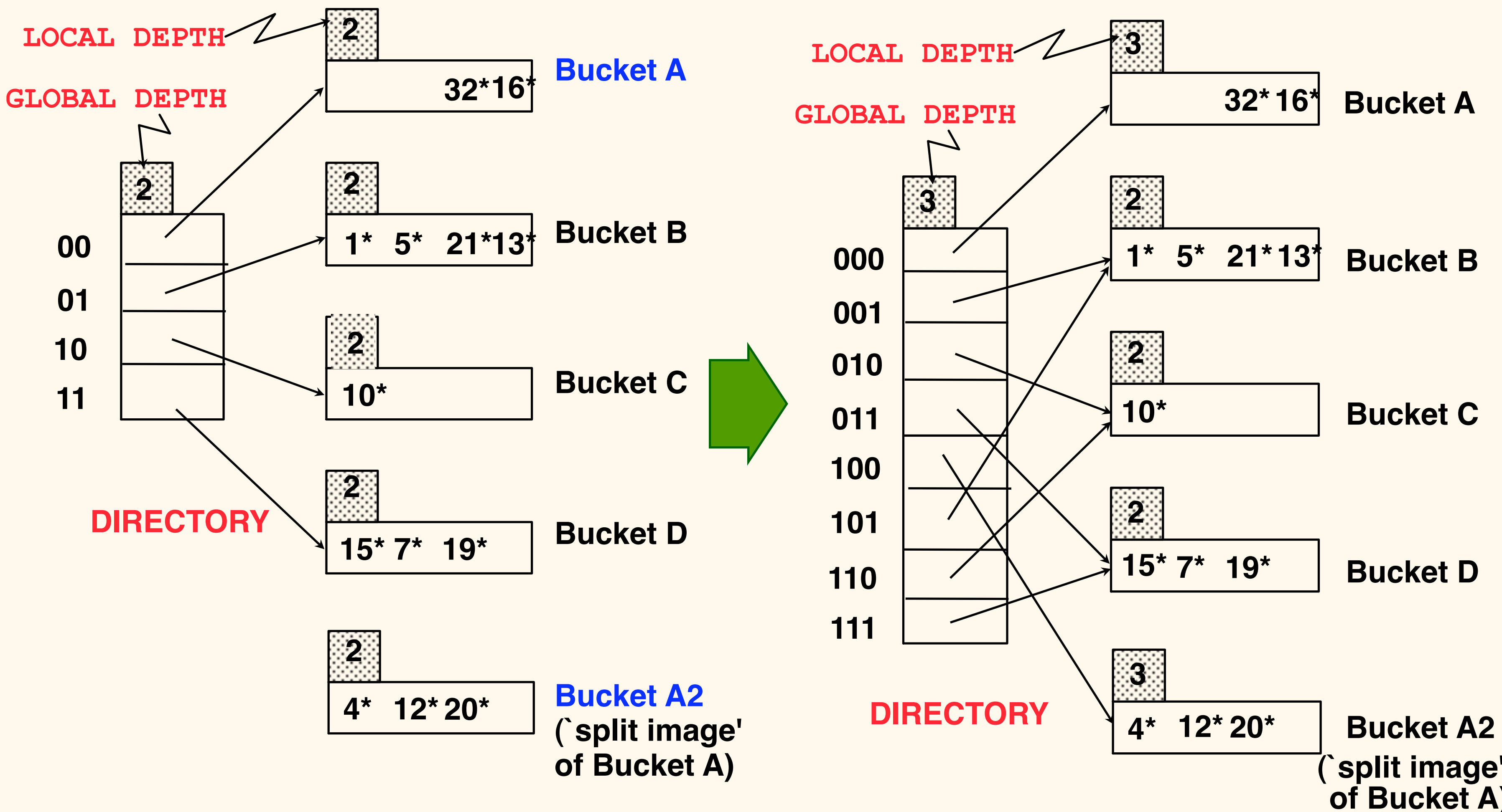
Inserts

- If bucket is full, *split* it (*allocate new page, re-distribute*).
- *If necessary*, double the directory. Splitting or not can be decided by comparing *global depth* and *local depth* for the split bucket.
  - Split if global depth = local depth.
  - Don't otherwise.



Insert *r* with h(*r*)=20?

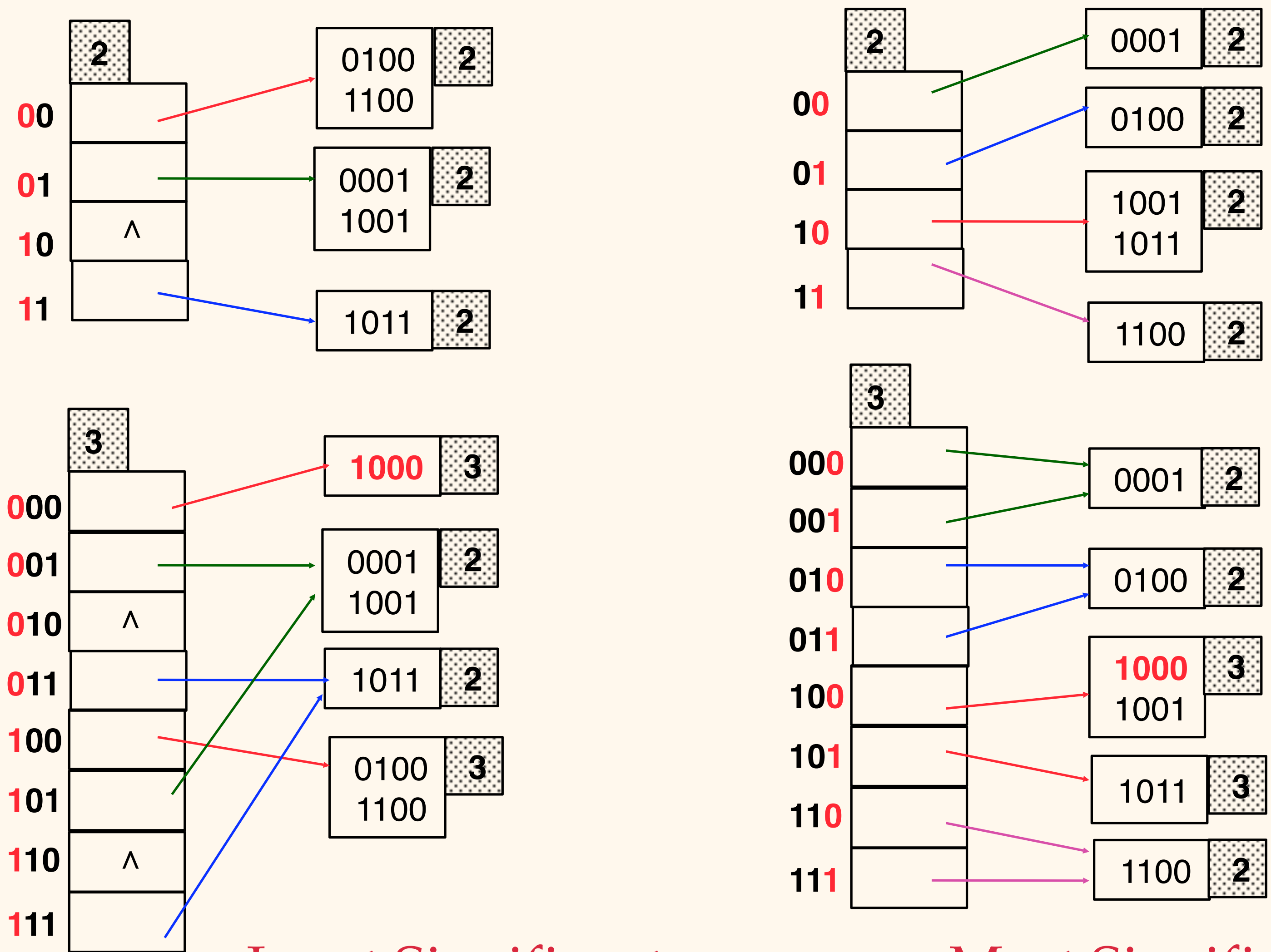
*Insert **h**(r)=20 (Causes Doubling)*



Points to Note

- 20 = binary 10100. Last **2** bits (00) tell us *r* belongs in 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 bucket 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. (Use of least significant bits enables efficient doubling via copying of directory!)

Directory Doubling (inserting 8)*



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
  - 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.
  - Entries with *same key value* (duplicates) need overflow pages!
- **Delete**: removal of data entry from bucket
  - If bucket is 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 lead to long overflow chains.
- Extendible Hashing avoids overflow pages by splitting a full bucket when a new data entry is to be added to it. (*But 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.