

Linear Hashing



- ❖ This is another dynamic hashing scheme, an alternative to Extendible Hashing.
- ❖ LH handles the problem of long overflow chains without using a directory.
- ❖ Idea: Use a family of hash functions h_0, h_1, h_2, \dots
 - $h_i(\text{key}) = h(\text{key}) \bmod(2^i N)$; N = initial # buckets
 - h is some hash function
 - If $N = 2^{d0}$, for some $d0$, h_i consists of applying h and looking at the last d_i bits, where $d_i = d0 + i$.
 - h_{i+1} doubles the range of h_i (similar to directory doubling)

Linear Hashing (Contd.)

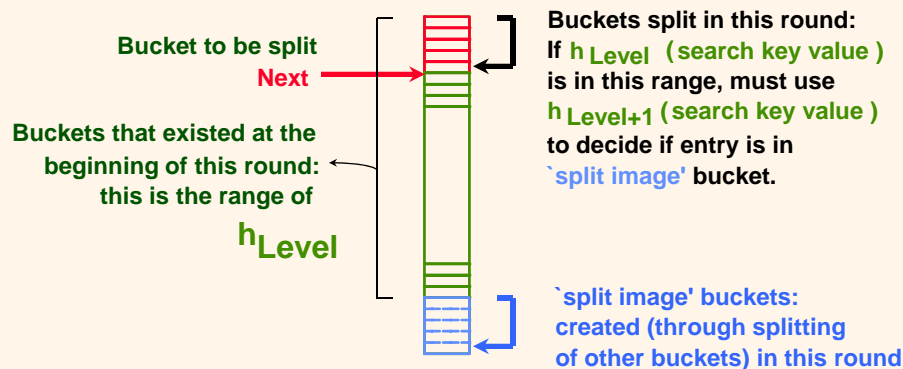


- ❖ Directory avoided in LH by using overflow pages, and choosing bucket to split round-robin.
 - **Splitting proceeds in `rounds`**. Round ends when all N_R initial (for round R) buckets are split. Buckets 0 to $Next-1$ have been split; $Next$ to N_R yet to be split.
 - **Current round number is Level**.
 - **Search**: To find bucket for data entry r , find $h_{Level}(r)$:
 - If $h_{Level}(r)$ in range $Next$ to N_R , r belongs here.
 - Else, r could belong to bucket $h_{Level}(r)$ or bucket $h_{Level}(r) + N_R$; must apply $h_{Level+1}(r)$ to find out.

Overview of LH File



- ❖ In the middle of a round.



Linear Hashing (Contd.)

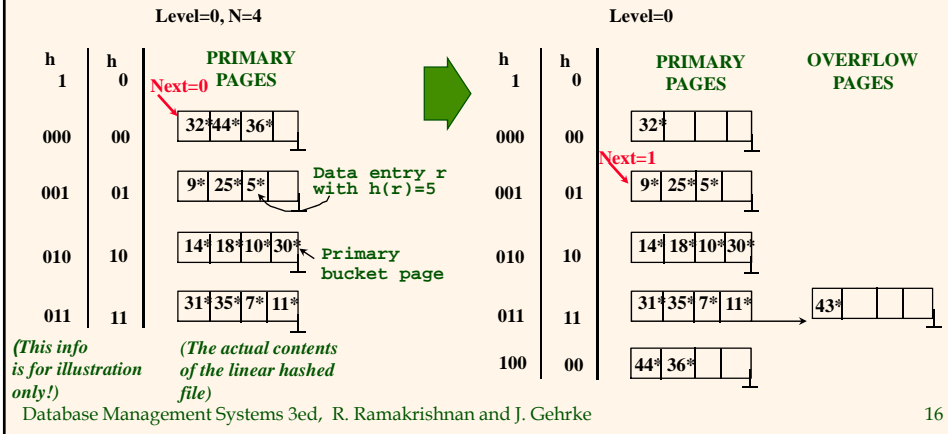


- ❖ **Insert:** Find bucket by applying $h_{Level} / h_{Level+1}$:
 - If bucket to insert into is full:
 - Add overflow page and insert data entry.
 - Split *Next* bucket and increment *Next*.
- ❖ Can choose any *criterion* to 'trigger' split.
- ❖ Since buckets are split round-robin, long overflow chains don't develop!
- ❖ Doubling of directory in Extendible Hashing is similar; switching of hash functions is *implicit* in how the # of bits examined is increased.

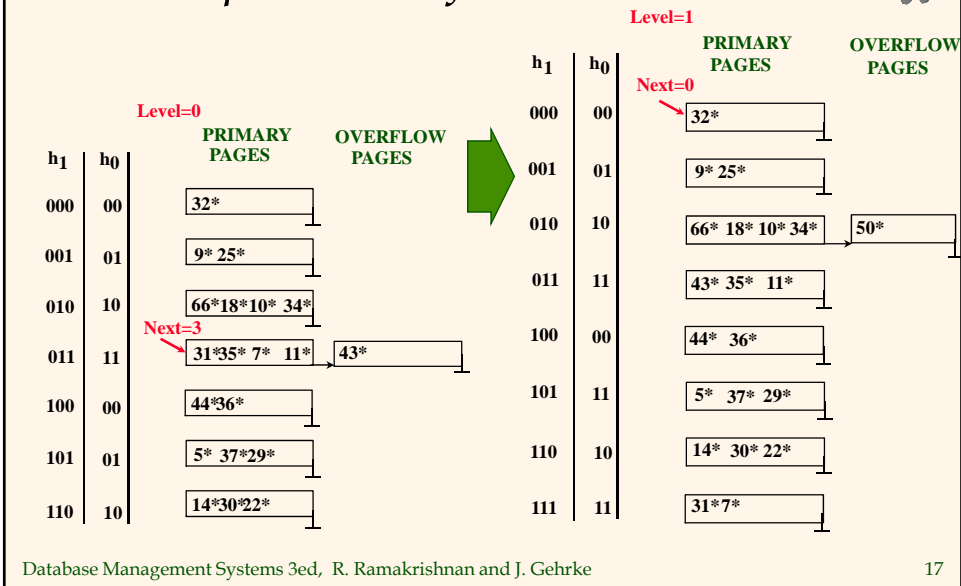
Example of Linear Hashing



❖ On split, $h_{Level+1}$ is used to re-distribute entries.



Example: End of a Round



LH Described as a Variant of EH



- ❖ The two schemes are actually quite similar:
 - Begin with an index where directory has N elements.
 - Use overflow pages, split buckets round-robin.
 - First split is at bucket 0. (Imagine directory being doubled at this point.) We only need to add directory element N now. Elements $\langle 1, N+1 \rangle$, $\langle 2, N+2 \rangle$, ... are the same.
 - When bucket 1 splits, create directory element $N+1$, etc.
- ❖ So, directory can double gradually. Also, primary bucket pages are created in order. If they are *allocated* in sequence too (so that finding i 'th is easy), we actually don't need a 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. (*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.

Summary (Contd.)



- ❖ Linear Hashing avoids directory by splitting buckets round-robin, and using overflow pages.
 - Overflow pages not likely to be long.
 - Duplicates handled easily.
 - Space utilization could be lower than Extendible Hashing, since splits not concentrated on `dense' data areas.
 - Can tune criterion for triggering splits to trade-off slightly longer chains for better space utilization.
- ❖ For hash-based indexes, a *skewed* data distribution is one in which the *hash values* of data entries are not uniformly distributed!