

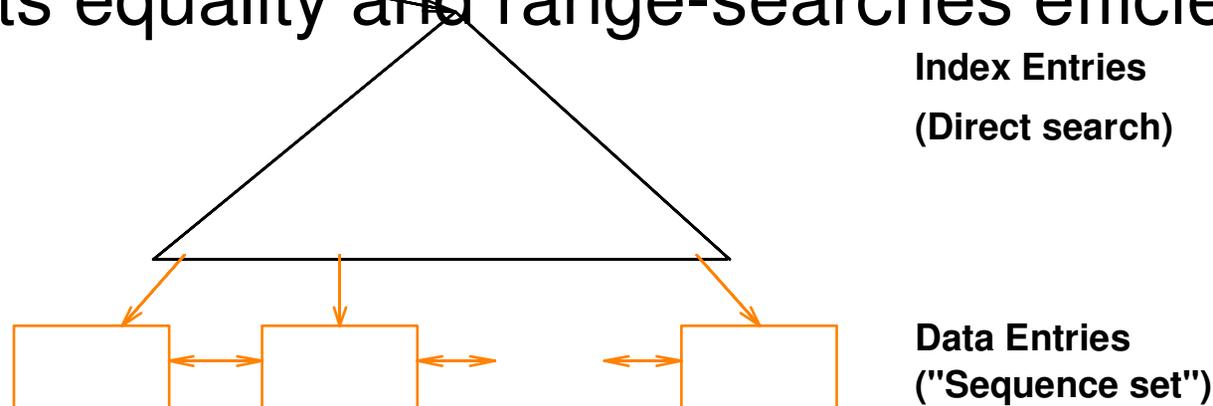
# Indexes: Tree Based and Hash Based

B+-tree

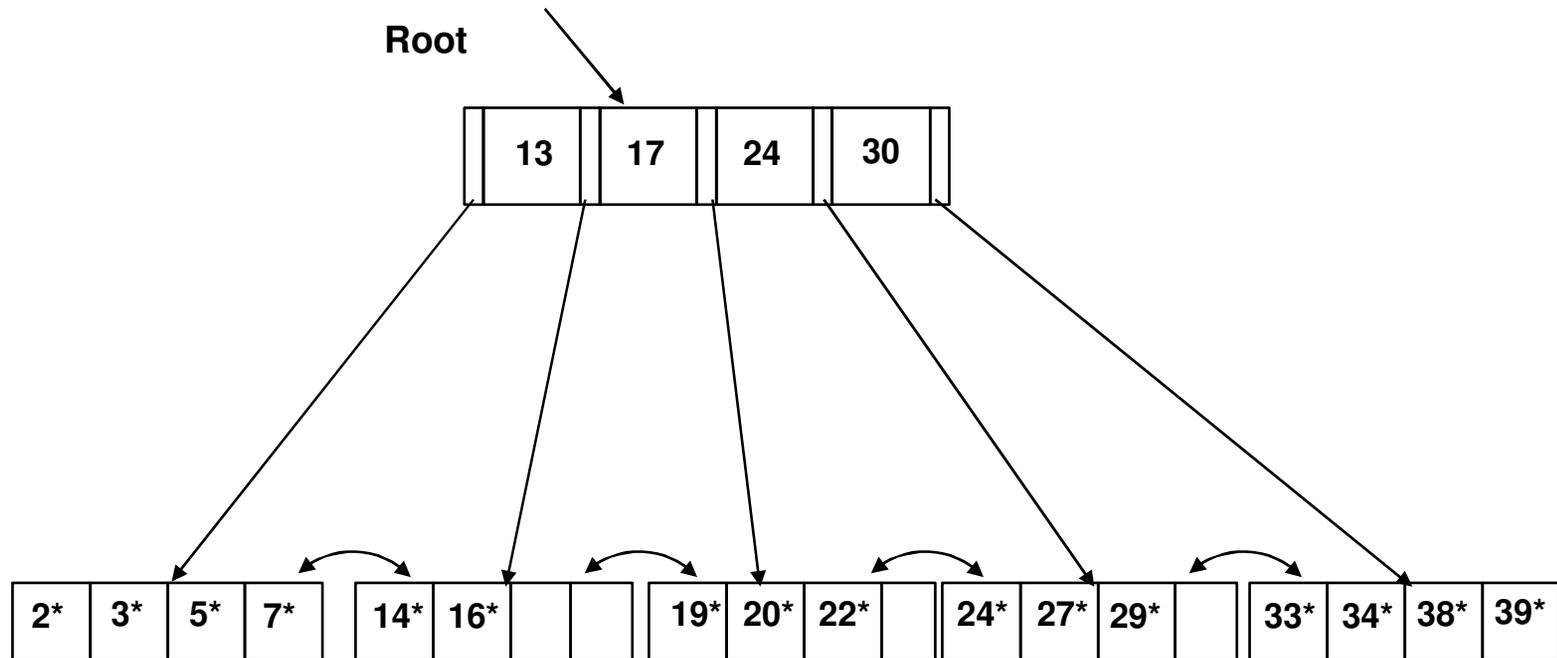
Linear Hashing

# B+ Tree: The Most Widely Used Index

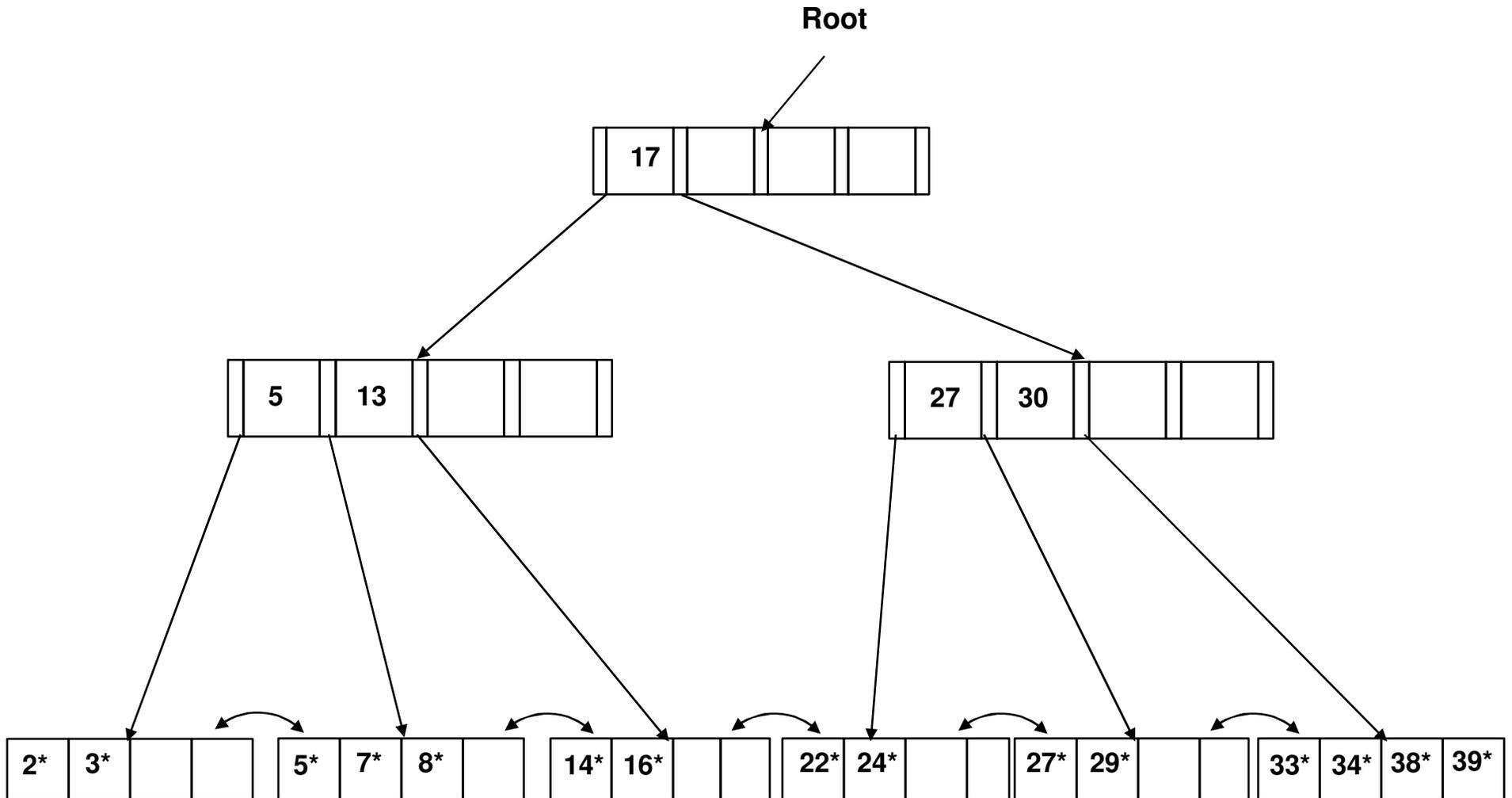
- Insert/delete at \_\_\_\_\_ cost
  - keep tree *height-balanced*. (F = fanout, N = # leaf pages)
- Minimum 50% occupancy (except for root).
  - Each node contains  $\mathbf{d} \leq \underline{m} \leq 2\mathbf{d}$  entries.
  - The parameter  $\mathbf{d}$  is called the *order* of the tree.
- Supports equality and range-searches efficiently.



# ICE: Inserting 35\* ...



# ICE: Deleting 29\*



# ICE: Composite Search Keys

- B+-tree index on (Age, Salary)
- Which can you answer efficiently using a B+-tree?
  - Age = 20
  - Age > 20
  - Age = 20, Salary = 100000
  - Age > 20, Salary = 100000
  - Age = 20, Salary > 100000
  - Age > 20, Salary > 100000
- Assume B+-tree index on (Age, Salary, Bonus); which can you answer efficiently?
  - Age = 20, Salary = 100000, Bonus > 5000
  - Age = 20, Salary > 100000, Bonus > 5000

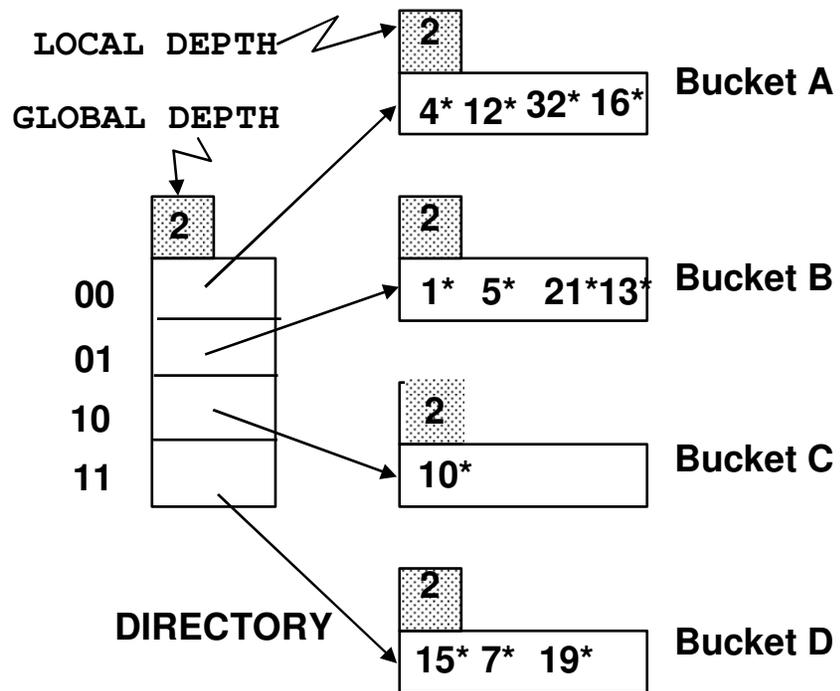
# Extendible Hashing

- Main idea: If bucket (primary page) becomes full, why not re-organize file by *doubling* # of buckets?

Essentially “splitting” buckets

- But reading and writing all buckets is expensive!
  - Idea: Use *directory of pointers to buckets*,
  - Double # of buckets by *doubling the directory*, splitting just the bucket that overflowed!
  - Directory much smaller than file, so doubling it is much cheaper.
  - *No overflow pages!*

# ICE: Insert $h(r)=20$



# Comments on Extendible Hashing

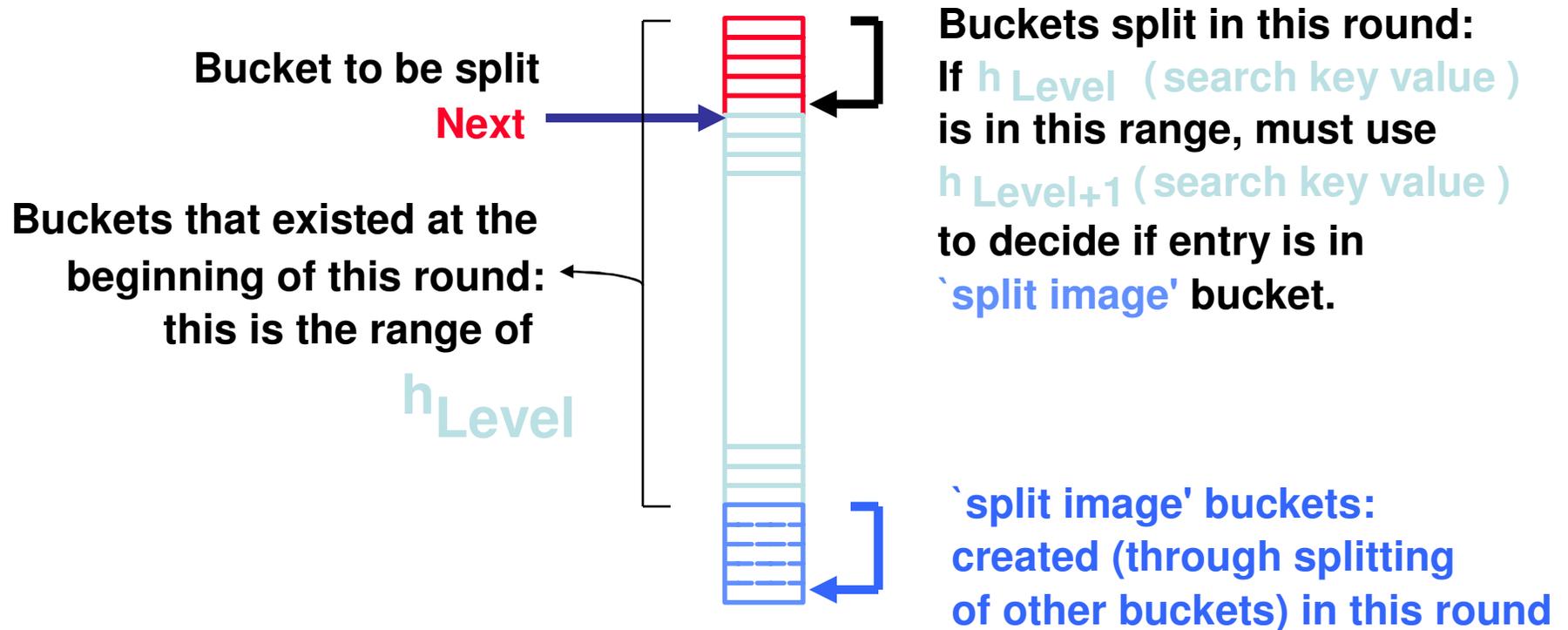
- If directory fits in memory, equality search answered with \_\_\_\_\_ I/O; else \_\_\_\_\_
  - 100MB file, 100 bytes/rec, 4K pages contain 1,000,000 records (as data entries) and 25,000 directory elements; chances are high that directory will fit in memory.
- Directory grows in spurts, and, if the distribution *of hash values* is \_\_\_\_\_, directory can grow large

# 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, and handles duplicates
- Main idea:

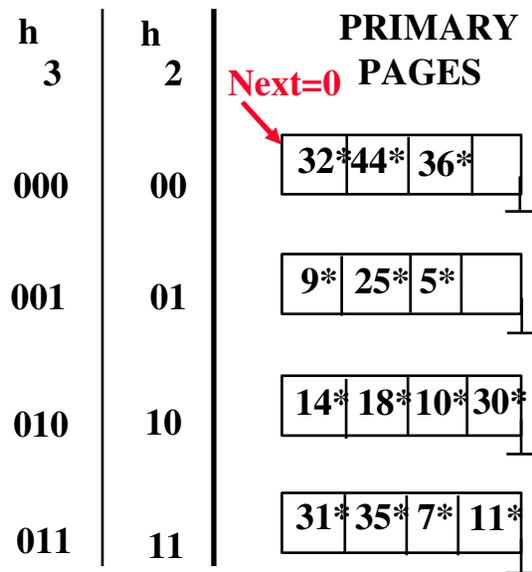
# Overview of LH File

- In the middle of a round.



# ICE: Inserting $h(r) = 43$

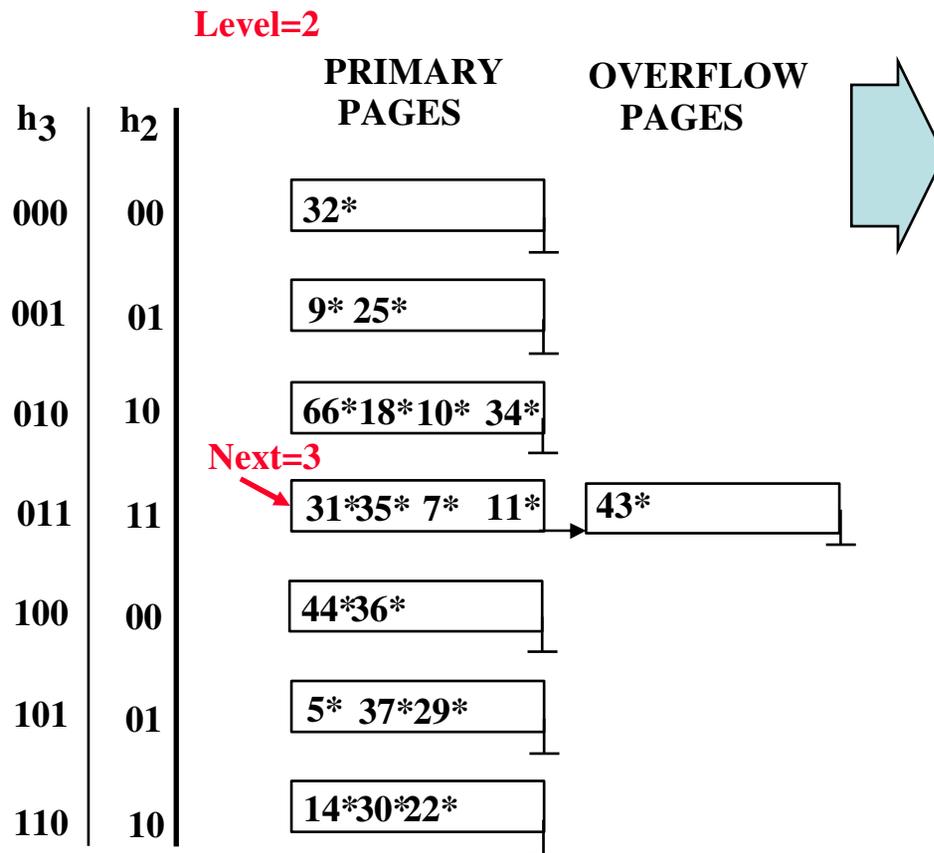
Level=2, N=4



*(This info is for illustration only!)*

*(The actual contents of the linear hashed file)*

# Inserting $h(r) = 50$ (End of a Round)



# Summary - Hashing

- Hash-based indexes: best for \_\_\_\_\_ searches, cannot support \_\_\_\_\_ searches.
- Static Hashing can lead to \_\_\_\_\_.
- Extendible Hashing uses directory doubling to avoid \_\_\_\_\_
  - Duplicates may require \_\_\_\_\_
- Linear hashing avoids directory by splitting in rounds
  - Naturally handles \_\_\_\_\_
  - Uses overflow buckets (but not very long in practice)