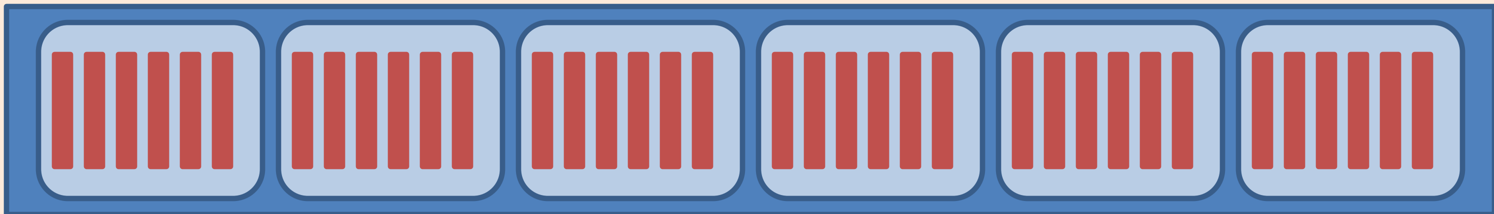
ICS 421 Spring 2010
Indexing (1)

Asst. Prof. Lipyeow Lim
Information & Computer Science Department
University of Hawaii at Manoa

How to speed up queries?

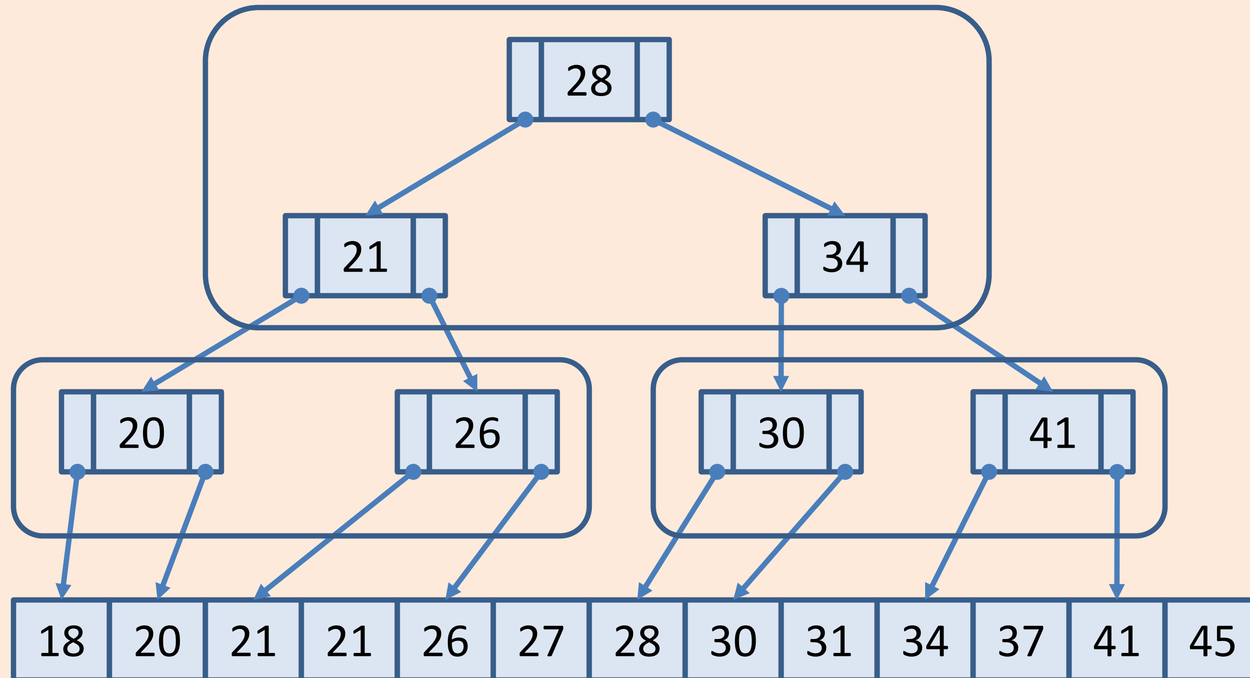
```
SELECT *
FROM Sailors
WHERE age>40
```

Array of Sailor Tuples/Records



File of Record for Sailors

Binary Search Trees

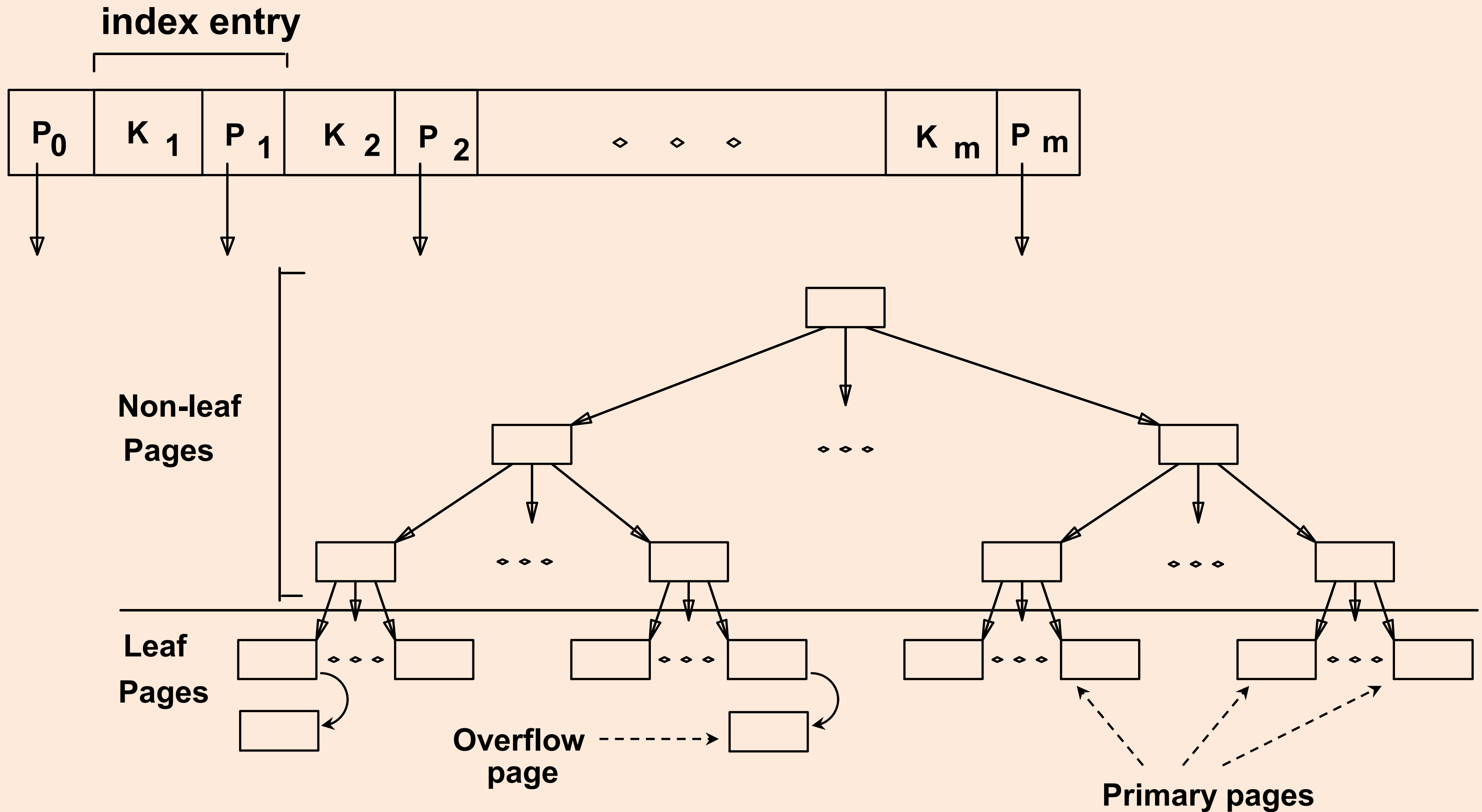


- Given search value
  - if value < node.value, then follow left pointer
  - Else follow right pointer
- How do generalize each index node to an index page ?
- How do we generalize this to search pages of records ?

Indexes

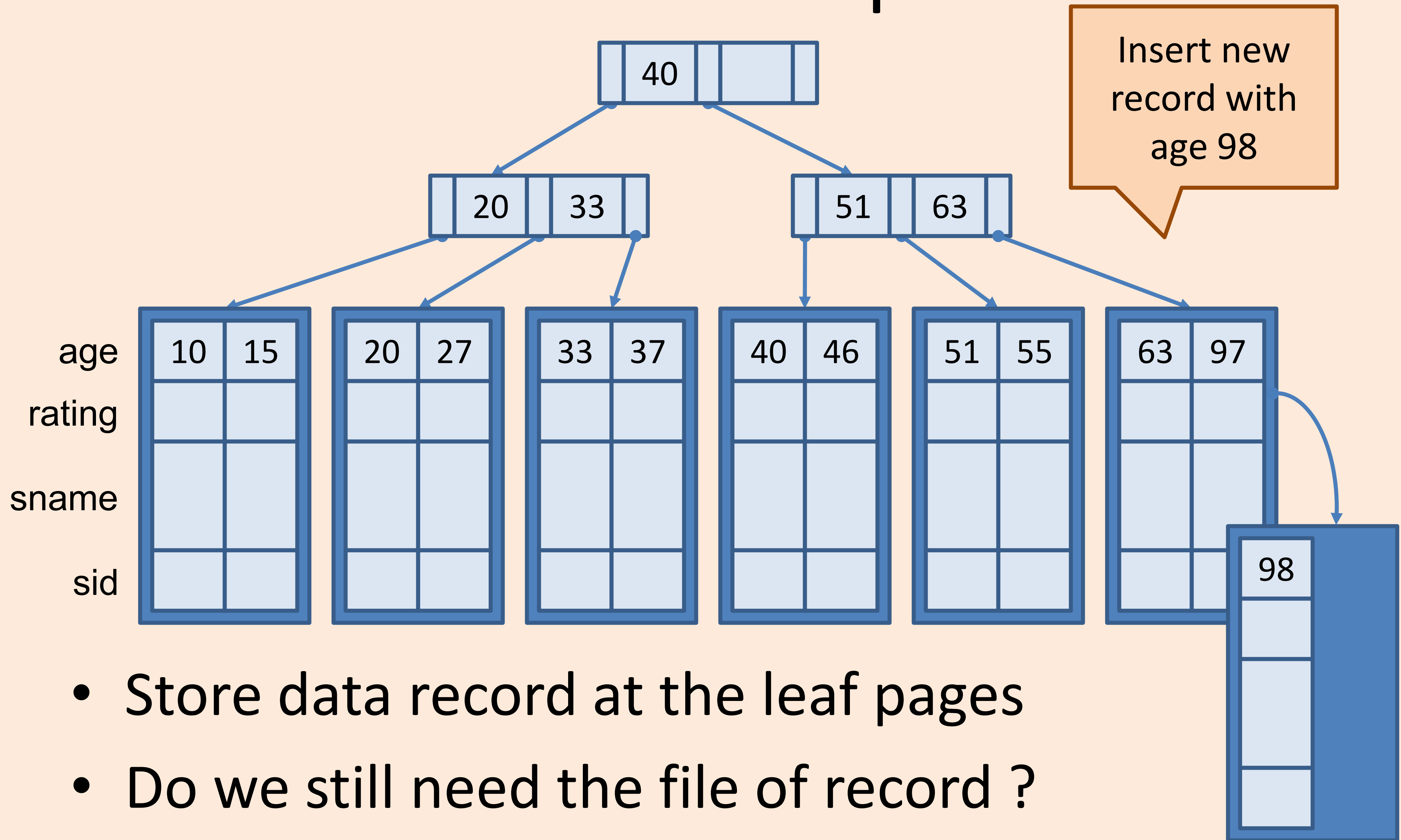
- What do we store in the index nodes ? Let k be the key value for an index entry:
  1. Data record with key value k
  2. <k, rid of data record with key value k>
  3. <k, list of rids of data records with key value k>
- What kind of queries does the index support?
  - Range
  - Point (or equality)

Indexed Sequential Access Method (ISAM)



- Static (m+1)-way Search Tree

ISAM: Example



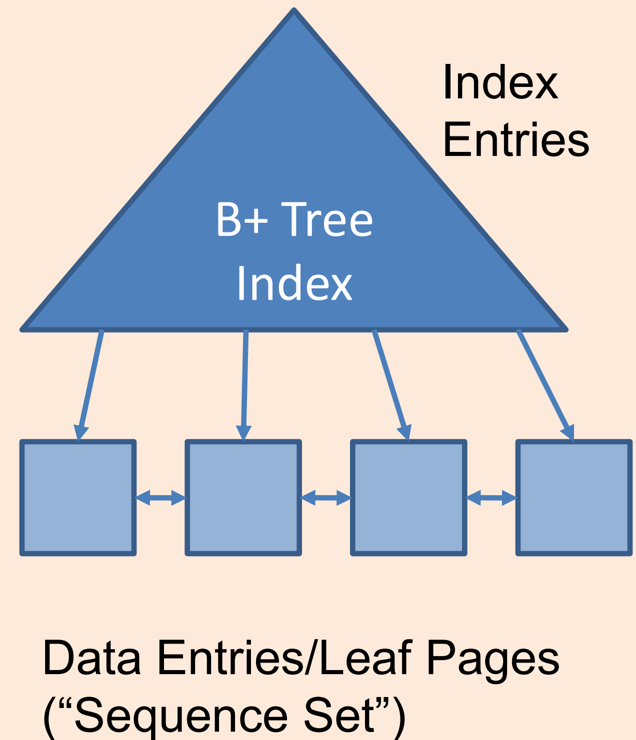
- Store data record at the leaf pages
- Do we still need the file of record ?

ISAM Facts

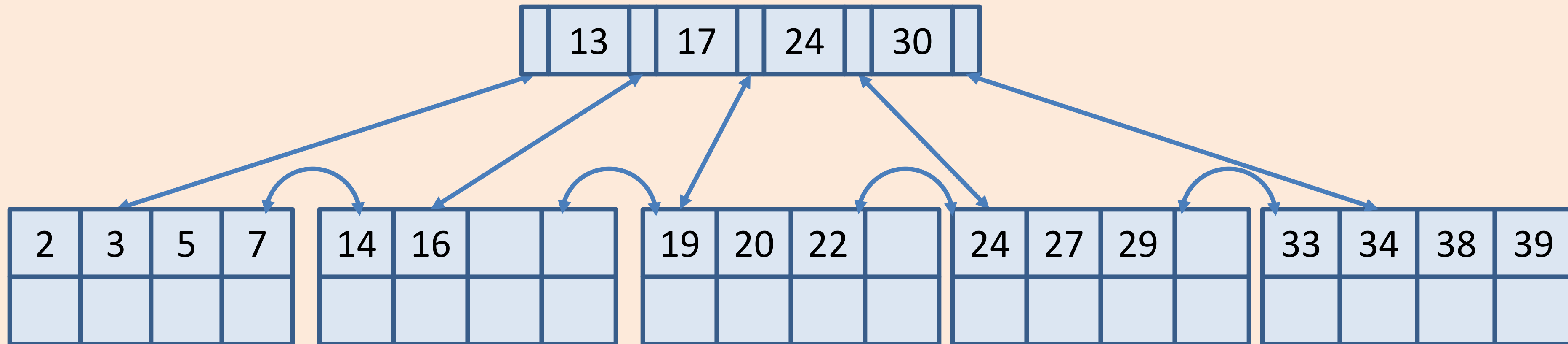
- *File creation*: Leaf (data) pages allocated sequentially, sorted by search key; then index pages allocated, then space for overflow pages.
- *Index entries*: <search key value, page id>; they `direct' search for *data entries*, which are in leaf pages.
- *Search*: Start at root; use key comparisons to go to leaf. Cost=O($\log_F N$) ; F = # entries/index pg, N = # leaf pgs
- *Insert*: Find leaf data entry belongs to, and put it there. If full, allocate and put in overflow page
- *Delete*: Find and remove from leaf; if empty overflow page, de-allocate.
- Static tree structure: inserts/deletes affect only leaf pages.

B+ Tree Index

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



B+ Tree: Search Example

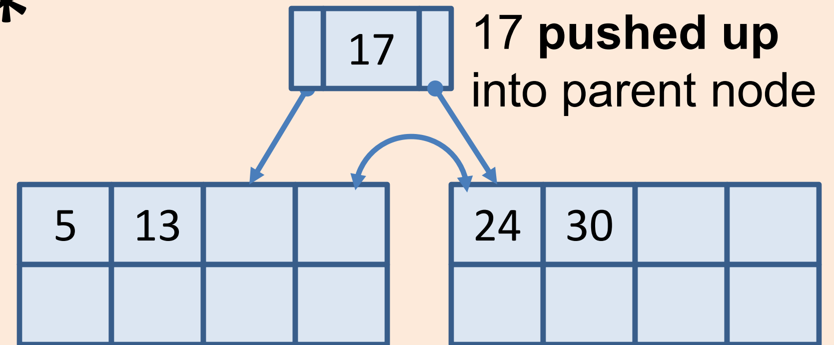


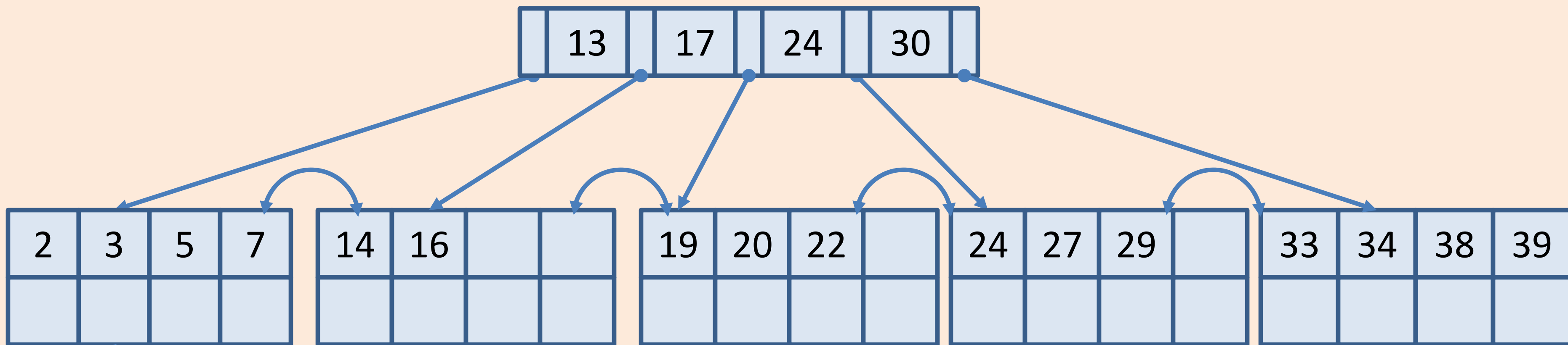
- Leaf entries store <key,rid> pairs
- What is the order ?
- Search for: age=5, age=15, age>=24

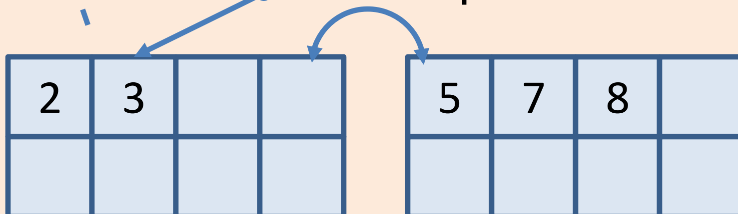
Inserting a new data entry

- Find correct leaf *L.*
- Put data entry onto *L*.
  - If *L* has enough space, *done*!
  - Else, must *split* *L (into L and a new node L2)*
    - Redistribute entries evenly, **copy up** middle key.
    - Insert index entry pointing to *L2* into parent of *L*.
- This can happen recursively
  - To split index node, redistribute entries evenly, but **push up** middle key. (Contrast with leaf splits.)
- Splits "grow" tree; root split increases height.
  - Tree growth: gets *wider* or *one level taller at top.*

Example: Insert 8*





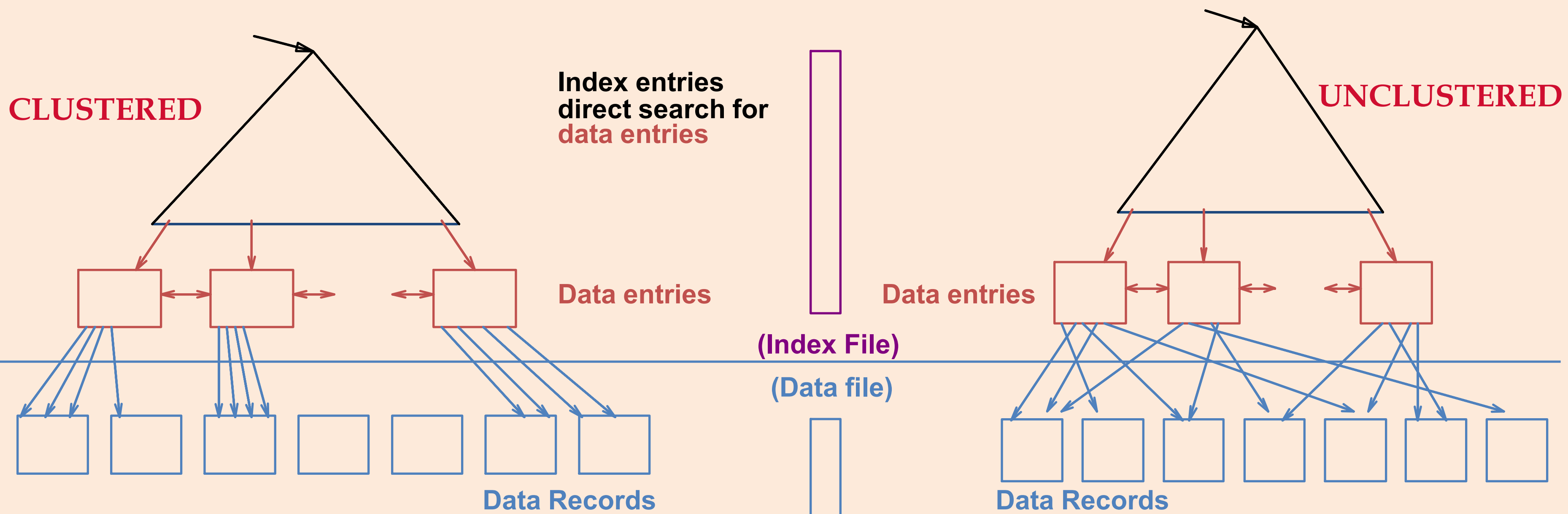


Deleting a data entry

- Start at root, find leaf *L* where entry belongs.
- Remove the entry.
  - If L is at least half-full, *done!*
  - If L has only **d-1** entries,
    - Try to re-distribute, borrowing from *sibling* *(adjacent node with same parent as L)*.
    - If re-distribution fails, *merge* *L* and sibling.
- If merge occurred, must delete entry (pointing to *L* or sibling) from parent of *L*.
- Merge could propagate to root, decreasing height.

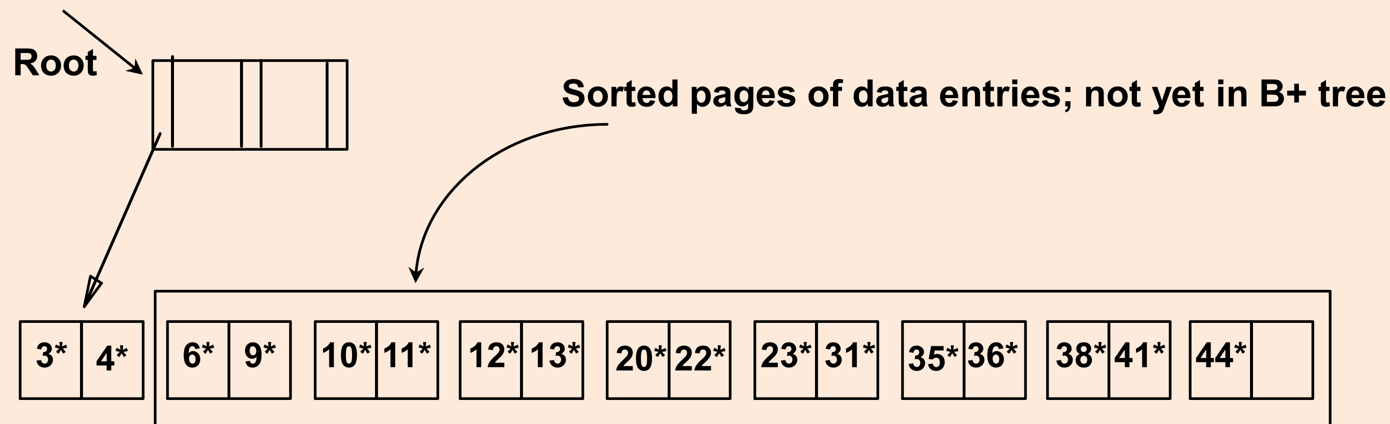
Miscellaneous

- How do we handle data with duplicates ?
  - Overflow buckets
  - Make rid part of the key
  - Each data entry stores <key, list of rids>
- Clustered vs Unclustered indexes



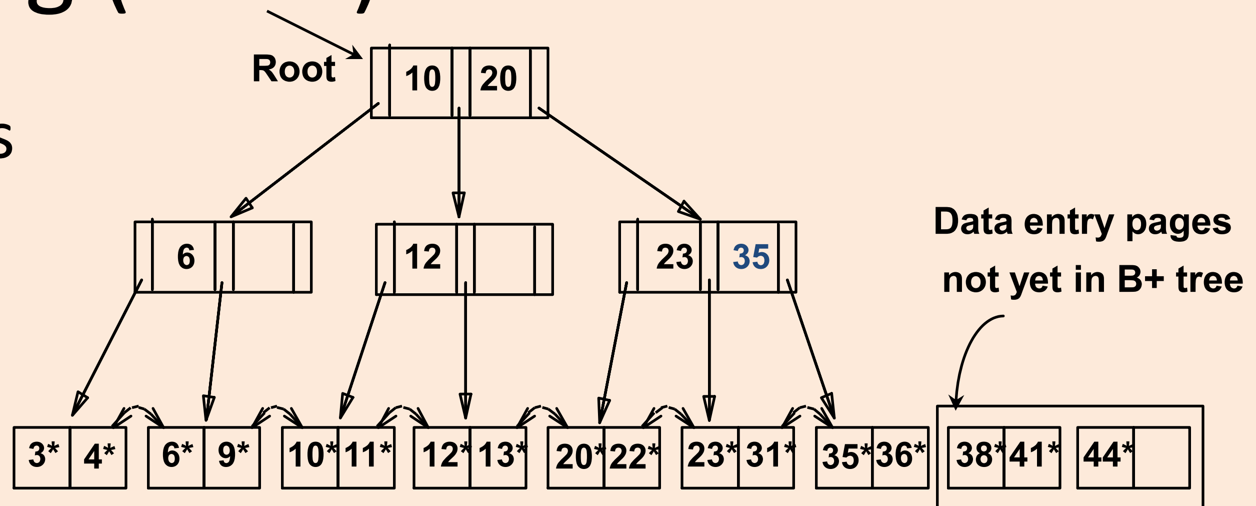
Bulk Loading a B+ Tree

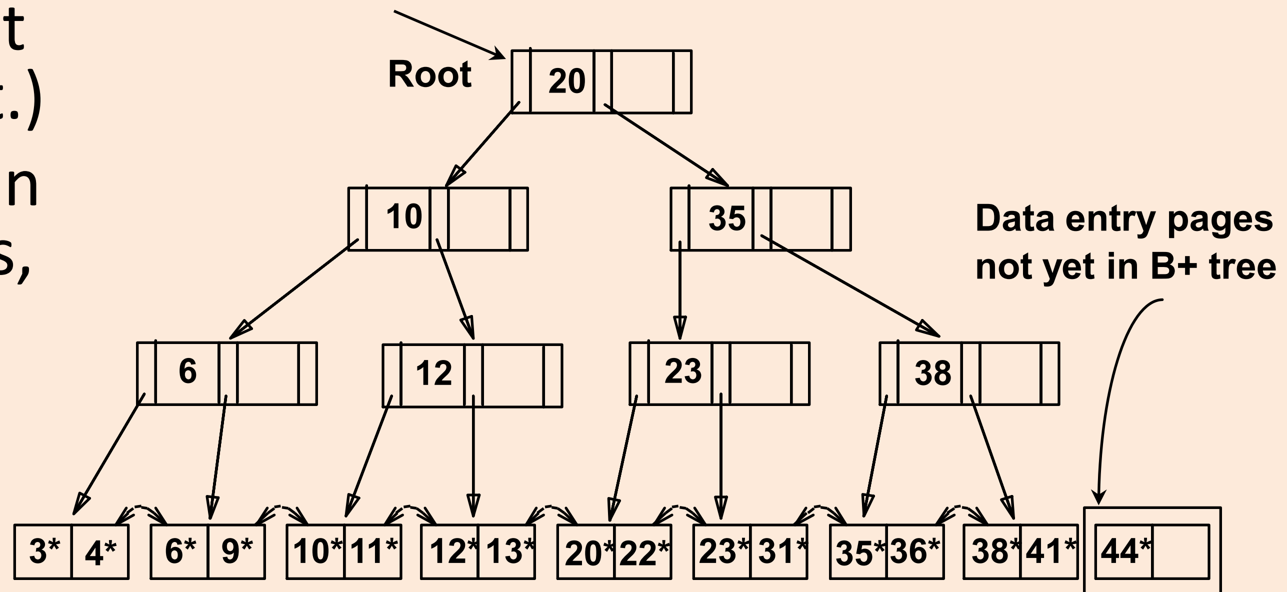
- If we have a large collection of records, and we want to create a B+ tree on some field, doing so by repeatedly inserting records is very slow.
- *Bulk Loading* can be done much more efficiently.
- *Initialization*: Sort all data entries, insert pointer to first (leaf) page in a new (root) page.



Bulk Loading (cont.)

- Index entries for leaf pages always entered into right-most index page just above leaf level. When this fills up, it splits. (Split may go up right-most path to the root.)
- Much faster than repeated inserts, especially when one considers locking!





Creating Indexes

- Most DBMS (eg. DB2) supports only B+ tree indexes:

  CREATE INDEX myIdx **ON** mytable(col1, col3)
  CREATE UNIQUE INDEX myUniqIdx **ON** mytable(col2, col5)
  CREATE INDEX myIdx **ON** mytable(col1, col3) **CLUSTER**

- If a primary key is specified in the CREATE TABLE statement, an (unclustered) index is automatically created for the PK.
- To create a clustered PK index:
  - Create table without PK constraint
  - Create index on PK with cluster option
  - Alter table to add PK constraint
- To get rid of unused indexes: **DROP INDEX** myIdx;