

ICS 321 Spring 2011

# Overview of Storage & Indexing (ii)

Asst. Prof. Lipyeow Lim

Information & Computer Science Department

University of Hawaii at Manoa

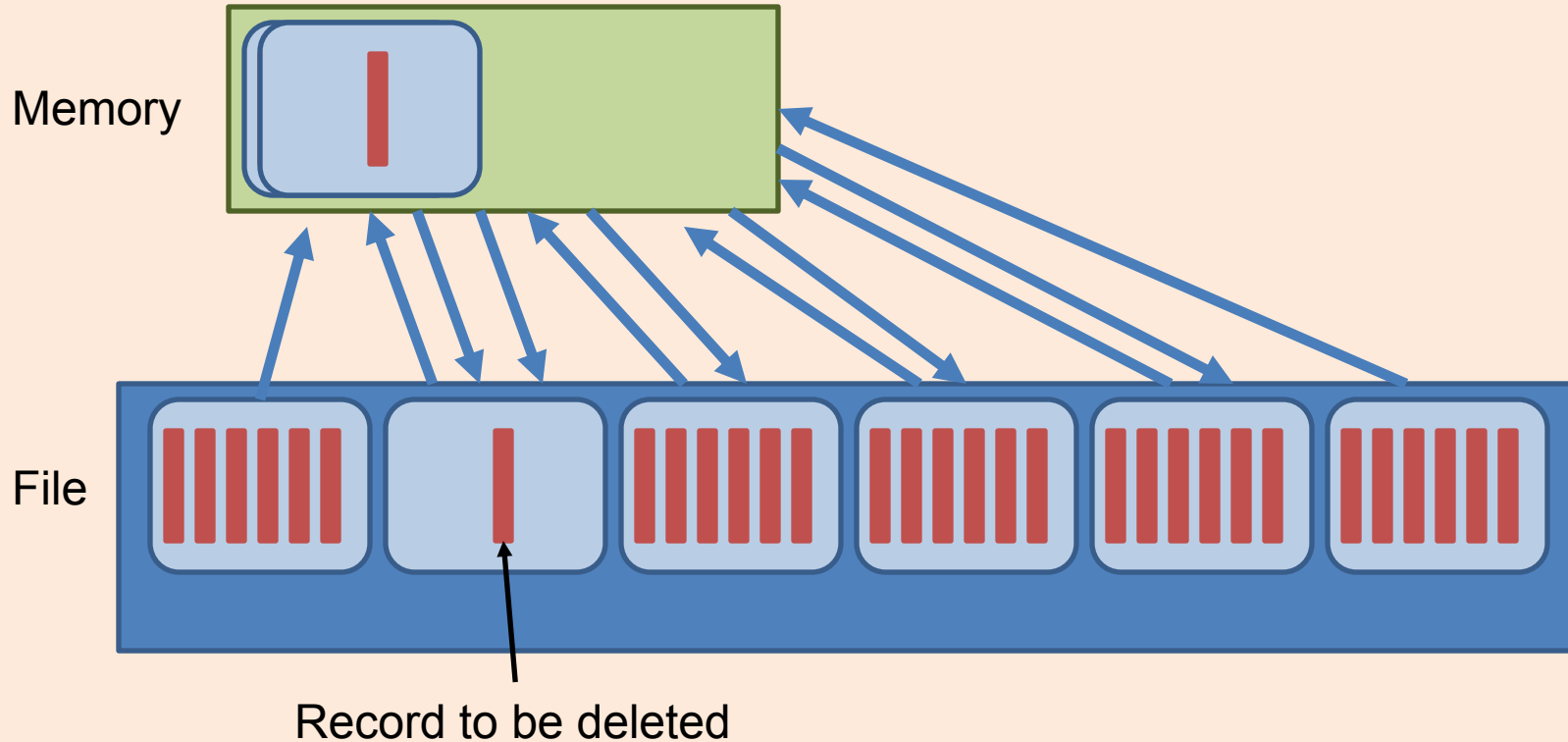
# Analysis of Heap File Storage

Operation	Worst Case Analysis	
Scans	$B*(D + R*C)$	<ul style="list-style-type: none"><li>• Fetch all B pages from disk into memory</li><li>• Process each record on each page</li></ul>
Point Query	$B*(D + R*C)$	<ul style="list-style-type: none"><li>• In the worst case, the desired record is the last record on the last page</li></ul>
Range Query	$B*(D + R*C)$	<ul style="list-style-type: none"><li>• Since file is unsorted, the desired records can be anywhere in the file, so we have to scan the entire file.</li></ul>
Insert	$2*D + C$	<ul style="list-style-type: none"><li>• Insert at the end of the file.</li><li>• Read in the last page</li><li>• Add record</li><li>• Write the page back</li></ul>
Delete	$2* B * (D + R*C)$	<ul style="list-style-type: none"><li>• Search for the record to be deleted</li><li>• Delete the record</li><li>• Move all subsequent records &amp; pages forward.</li></ul>

# Analysis of Heap File Storage (Disk Only)

Operation	Worst Case Analysis	
Scans	$B * D$	<ul style="list-style-type: none"><li>• Fetch all B pages from disk into memory</li><li>• Process each record on each page</li></ul>
Point Query	$B * D$	<ul style="list-style-type: none"><li>• In the worst case, the desired record is the last record on the last page</li></ul>
Range Query	$B * D$	<ul style="list-style-type: none"><li>• Since file is unsorted, the desired records can be anywhere in the file, so we have to scan the entire file.</li></ul>
Insert	$2 * D$	<ul style="list-style-type: none"><li>• Insert at the end of the file.</li><li>• Read in the last page</li><li>• Add record</li><li>• Write the page back</li></ul>
Delete	$2 * B * D$	<ul style="list-style-type: none"><li>• Search for the record to be deleted</li><li>• Delete the record</li><li>• Move all subsequent records &amp; pages forward.</li></ul>

# Deleting a Record



# Analysis of Sorted File Storage

Op	Worst Case Analysis	
Scans	$B*(D + R*\epsilon)$	<ul style="list-style-type: none"> <li>Fetch all B pages from disk into memory</li> <li>Process each record on each page</li> </ul>
Point Query	$D \log B + C \log R$	<ul style="list-style-type: none"> <li>Binary search for the desired page</li> <li>Binary search for the desired record within the page</li> </ul>
Range Query	$D \log B + C \log R + \lfloor S/R \rfloor * D + S*\epsilon$	<ul style="list-style-type: none"> <li>Let S be the number of records in the result</li> <li>Binary search for the desired page and record</li> <li>Fetch the next S records</li> </ul>
Insert	$D \log B + C \log R + 2*B*(D + R*\epsilon)$	<ul style="list-style-type: none"> <li>Binary search to insertion point</li> <li>In worst case, page has no extra space, so page is split</li> <li>Move all subsequent pages back</li> </ul>
Delete	$D \log B + C \log R + 2*B*(D + R*\epsilon)$	<ul style="list-style-type: none"> <li>Search for the record to be deleted</li> <li>Delete the record</li> <li>Move all subsequent pages forward</li> </ul>

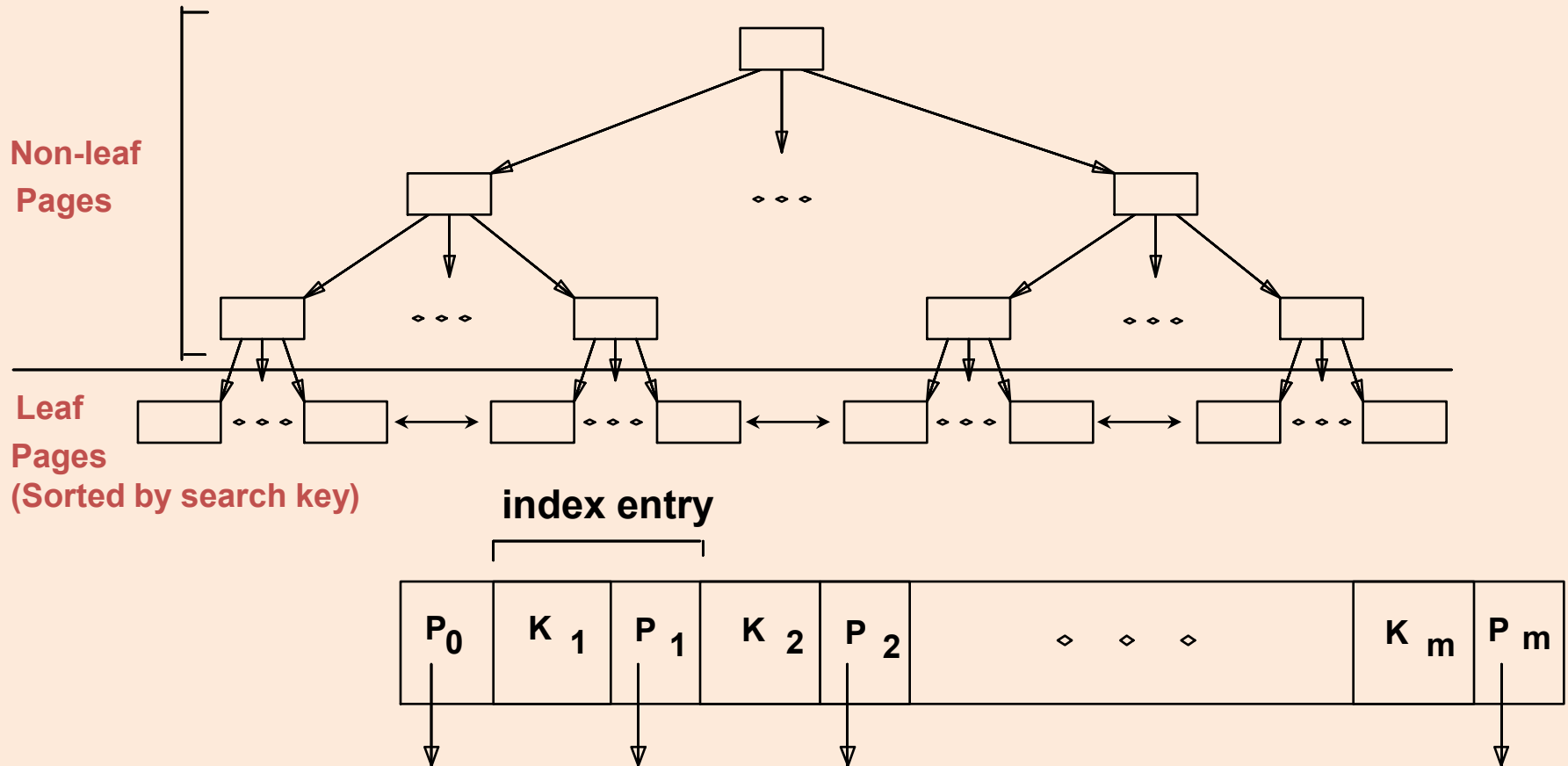
# Heap vs Sorted File

Op	Heap	Sorted
Scans	$B * D$	$B * D$
Point Query	$B * D$	$D \log B$
Range Query	$B * D$	$D \log B + \lfloor S/R \rfloor * D$
Insert	$2 * D$	$D \log B + 2 * B * D$
Delete	$2 * B * D$	$D \log B + 2 * B * D$

# Indexes

- An *index* on a file speeds up selections on the *search key fields* for the index.
  - Any subset of the fields of a relation can be the search key for an index on the relation.
  - *Search key* is *not* the same as *key* (minimal set of fields that uniquely identify a record in a relation).
- An index contains a collection of *data entries*, and supports efficient retrieval of all data entries  $k^*$  with a given key value  $k$ .
  - A data entry is usually in the form  $\langle \text{key}, \text{rid} \rangle$
  - Given data entry  $k^*$ , we can find record with key  $k$  in at most one disk I/O. (Details soon ...)

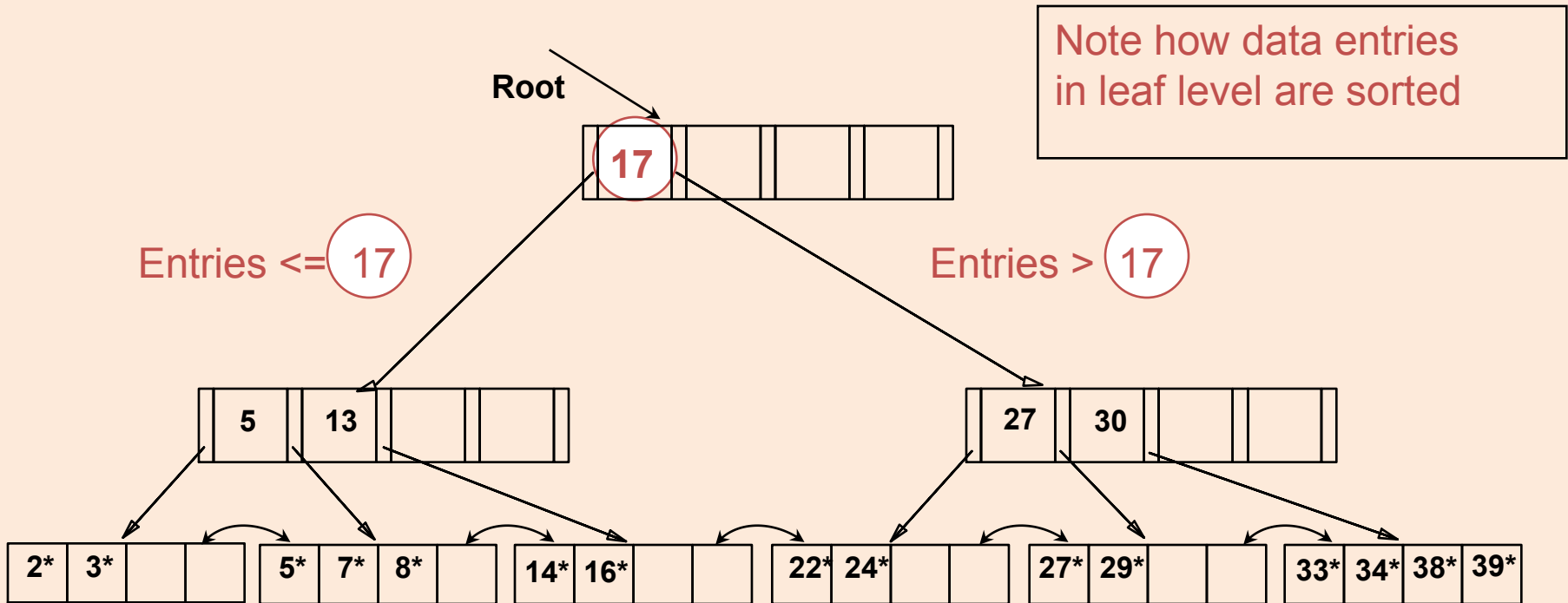
# B+ Tree Indexes



- Leaf pages contain **data entries**, and are chained (prev & next)
- A data entry typically contain a key value and a rid.
- Non-leaf pages have **index entries**; only used to direct searches:



# Example B+ Tree



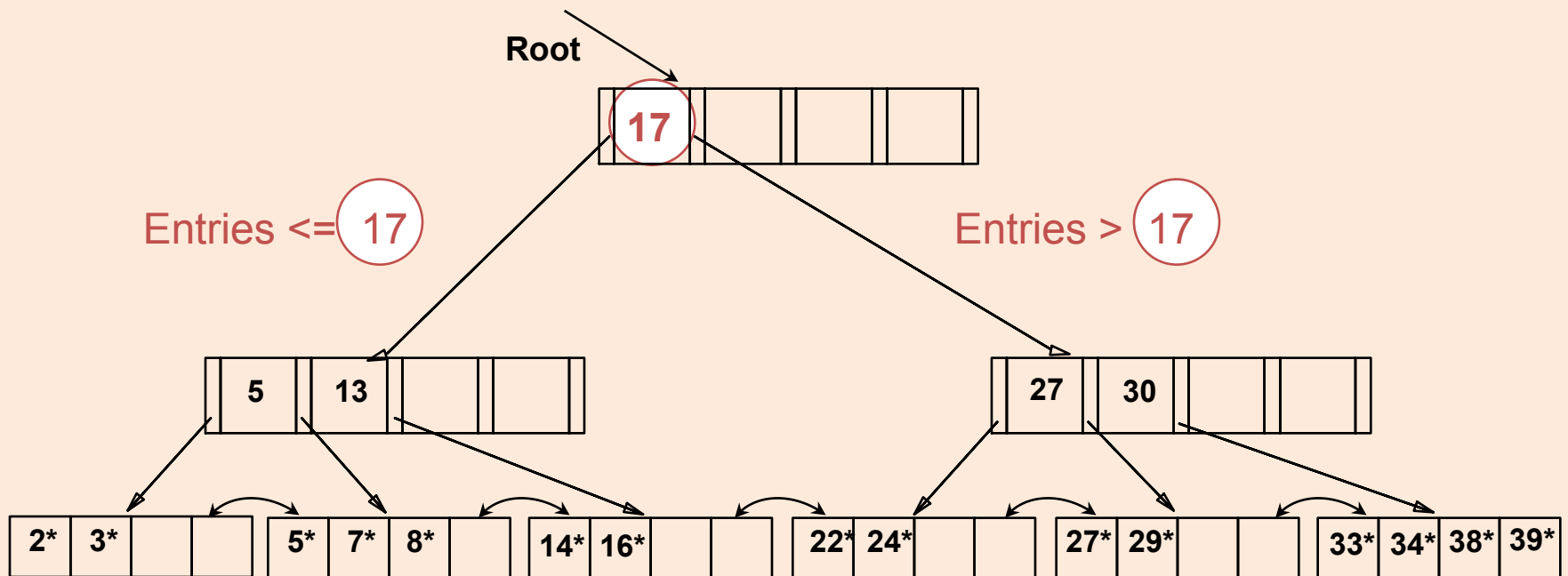
- Find 28\*? 29\*? All  $> 15^*$  and  $< 30^*$
- Insert/delete: Find data entry in leaf, then change it. Need to adjust parent sometimes.
  - And change sometimes bubbles up the tree

# Point Queries using B+ Trees

```
SELECT *  
FROM Employees  
WHERE age=30
```

Assume heap file  
data storage

- Use index to find 30\*
- Request tuple from buffer manager
- If not in bufferpool, fetch page from disk

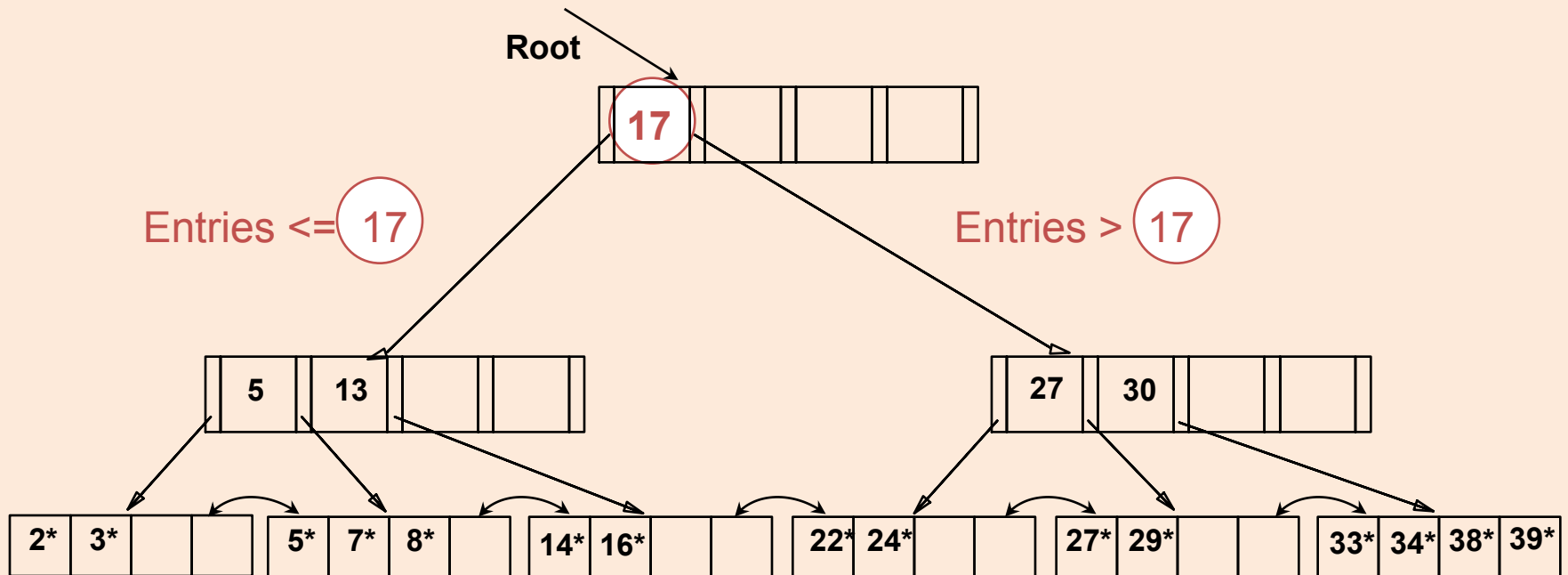


# Range Queries using B+ Trees

```
SELECT *  
FROM Employees  
WHERE age > 30
```

Assume heap file  
data storage

- Use index to find 30\*
- For each data entry to the right of 30\*
- Request tuples from buffer manager
- If not in bufferpool, fetch page from disk



# Analysis of Heap File with B+Tree Index

Op	Worst Case Analysis
Scans	$B * D$
Point Query	$D \log_F B + D$
Range Query	$D \log_F B + \lfloor S/R \rfloor * D + S * D$
Insert	$2 * D + 3 * D * \log_F B$
Delete	$D \log_F B + 2 * B * D$

- B+ tree search for the desired index page
- Binary search for the desired record within the index page
- Fetch the data page

- Let S be the number of records in the result
- B+ tree search for the desired index page
- Fetch the next S/R index leaf pages
- Fetch the data pages for the S records

- Insert record to end of heap file
- B+ tree search to find index page for the inserted record
- create a data entry for the inserted record in the index page. In worst case, index page has no extra space and page split cascades up. Write index pages

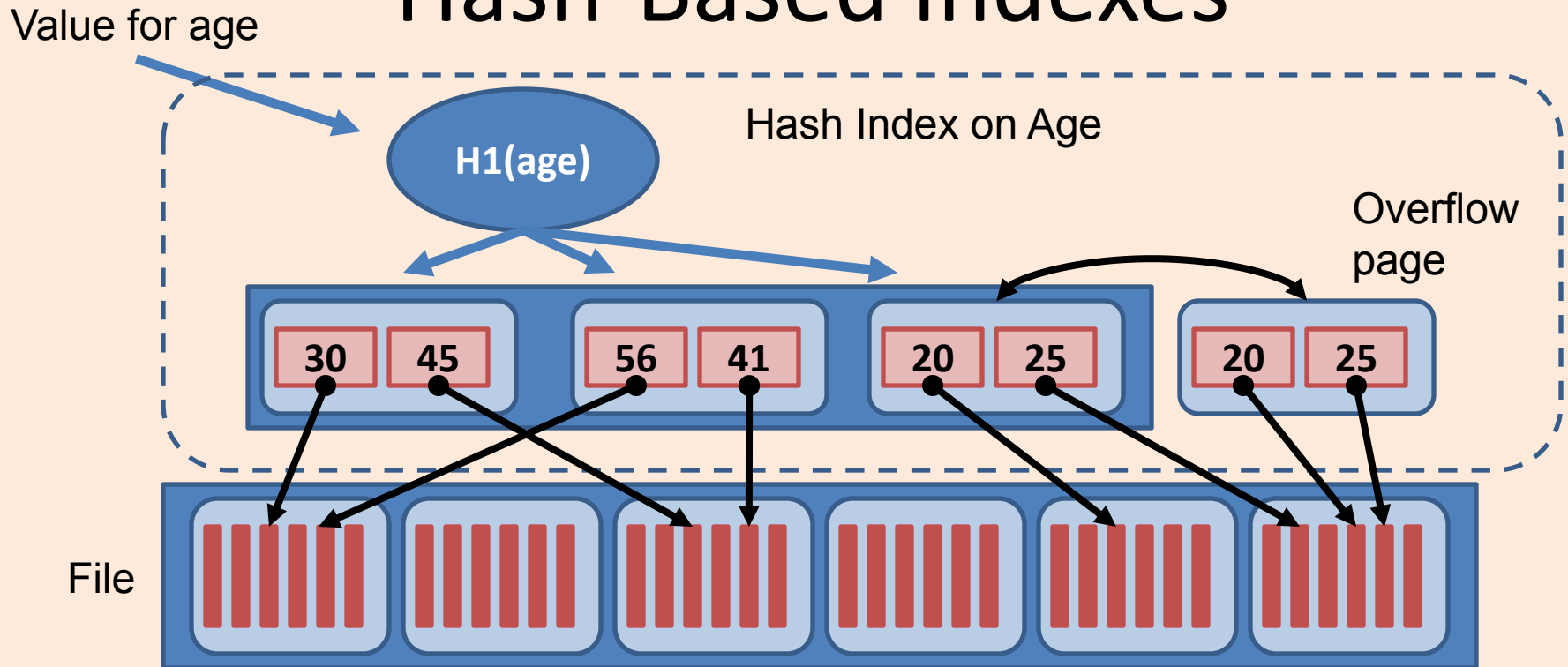
- B+ tree search for the desired index page and record
- Fetch the data page and delete the record
- In the worst case, data page is empty after deletion and needs to be removed from heap file

Assume index page density = data page density

# Running Comparison

Op	Heap	Sorted	Heap+Tree
Scans	$B * D$	$B * D$	$B * D$
Point Query	$B * D$	$D \log B$	$D \log_F B + D$
Range Query	$B * D$	$D \log B + \lfloor S/R \rfloor * D$	$D \log_F B + \lfloor S/R \rfloor * D + S * D$
Insert	$2 * D$	$D \log B + 2 * B * D$	$2 * D + 3 * D * \log_F B$
Delete	$2 * B * D$	$D \log B + 2 * B * D$	$D \log_F B + 2 * B * D$

# Hash-Based Indexes



- Index is a collection of buckets that contain data entries
  - Bucket = *primary page* plus zero or more *overflow pages*.
- *Hashing function h*:  $h(r)$  = bucket in which (data entry for) record  $r$  belongs.  $h$  looks at the *search key* fields of  $r$ .
- *No "index entries" in this scheme.*

# Analysis of Heap File with Hash Index

Op	Worst Case Analysis
Scans	$B * D$
Point Query	$2 * D$
Range Query	$B * D$
Insert	$4 * D$
Delete	$3 * D + 2 * B * D$

- Hash search for the desired index page
- Linear search for the desired record within the index page
- Fetch the data page

- Hash index does not support range queries
- Fall back on scanning the heap file

- Insert record to end of heap file
- Hash search to find index page for the inserted record
- Create a data entry for the inserted record in the index page.
- Write index page back to disk

- Hash search for the desired index page and record
- Fetch the data page, delete the record
- In the worst case, pages need to be moved forward
- update index page and write back to disk

# Running Comparison

Op	Heap	Sorted	Heap+Tree	Heap+Hash
Scans	$B * D$	$B * D$	$B * D$	$B * D$
Point Query	$B * D$	$D \log B$	$D \log_F B + D$	$2 * D$
Range Query	$B * D$	$D \log B + \lfloor S/R \rfloor * D$	$D \log_F B + \lfloor S/R \rfloor * D + S * D$	$B * D$
Insert	$2 * D$	$D \log B + 2 * B * D$	$2 * D + 3 * D \log_F B$	$4 * D$
Delete	$2 * B * D$	$D \log B + 2 * B * D$	$D \log_F B + 2 * B * D$	$3 * D + 2 * B * D$

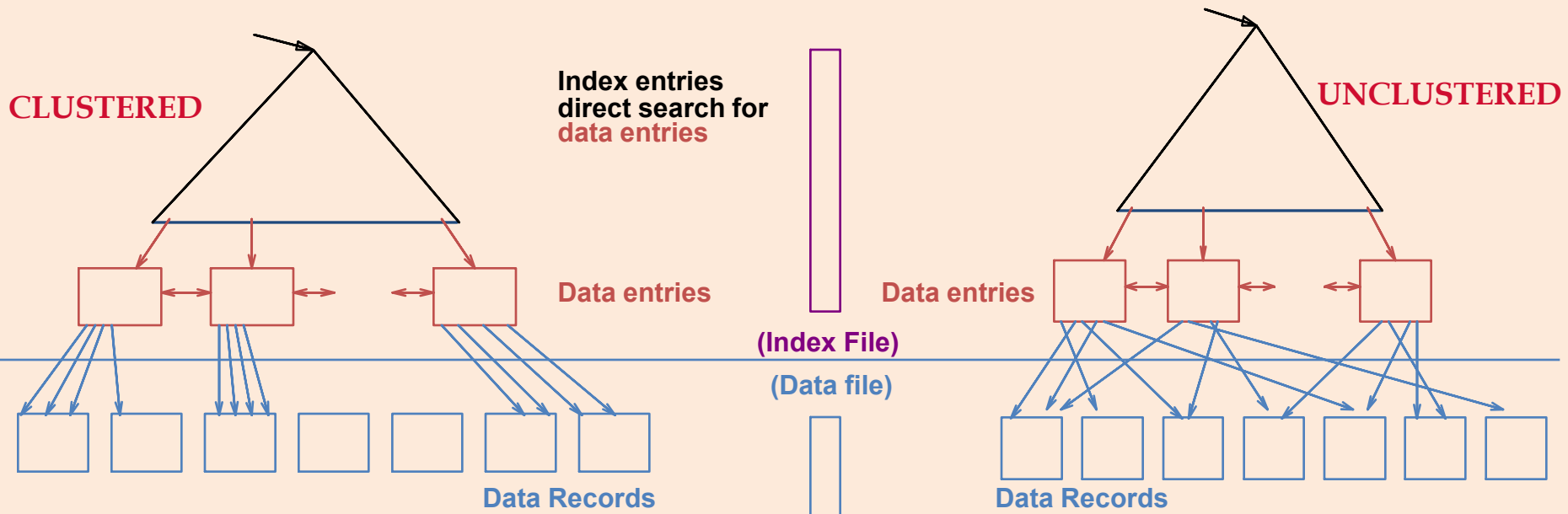


# Index Classifications

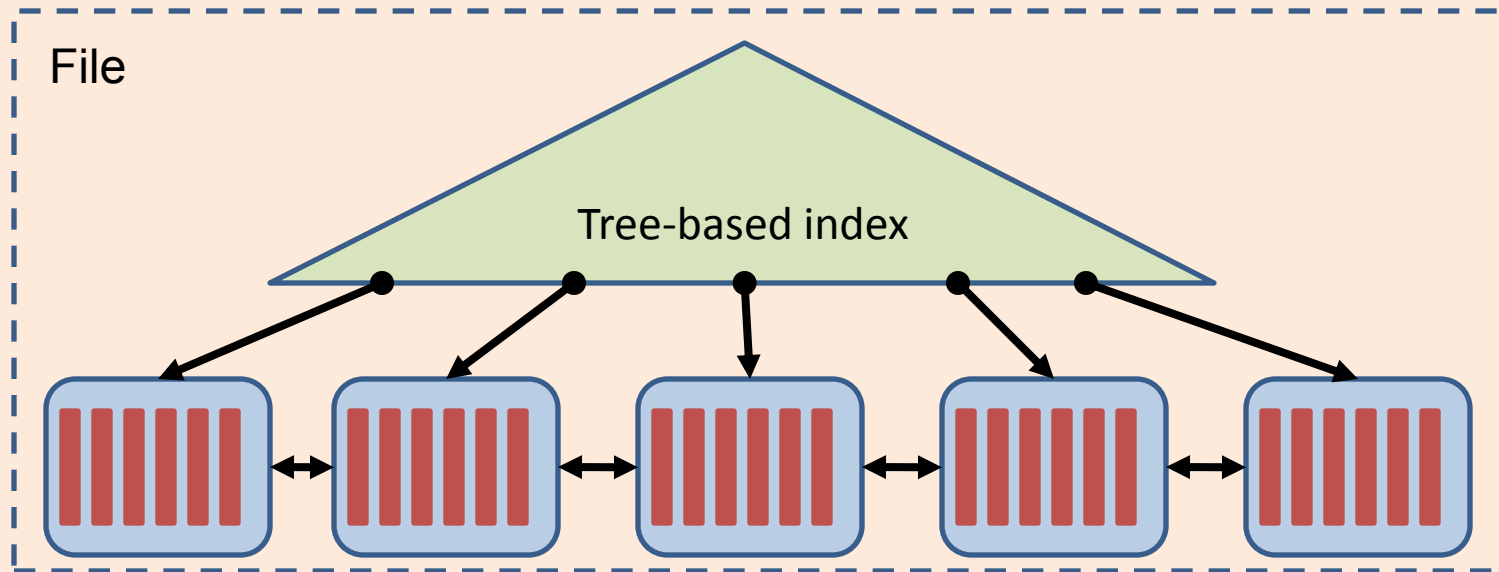
- What should be in a Data Entry  $k^*$  ?
  - Possibilities:
    - The data record itself with key value  $k$
    - $\langle k, \text{rid of data record with key value } k \rangle$
    - $\langle k, \text{list of rids of data records with key value } k \rangle$ 
      - Variable size data entries
  - Applies to any indexing technique
- Primary vs Secondary
  - **Primary index** : search key contains primary key
  - **Unique Index** : search key contains candidate key
- Clustered vs unclustered
  - **Clustered index**: order of data records same or close to order of data entries

# Clustered vs Unclustered Index

- Suppose data records are stored in a Heap file.
  - To build clustered index, first sort the Heap file (with some free space on each page for future inserts).
  - Overflow pages may be needed for inserts. (Thus, order of data recs is `close to`, but not identical to, the sort order.)



# Clustered File



- An index where the data entry contains the data record itself (cf. just the key value, RID pair).
- No heap/sorted file is used, the index IS the file of record
- Steps to build a clustered file:
  - Sort data records
  - Partition into pages
  - Build the tree on the pages

# Analysis of Clustered Files

Op	Worst Case Analysis
Scans	$B * D$
Point Query	$D \log_F B$
Range Query	$D \log_F B + \lfloor S/R \rfloor * D$
Insert	$3 * D \log_F B$
Delete	$2 * D \log_F B$

- B+ tree search for the desired index page
- Binary search for the desired record within the index page

- Let  $S$  be the number of records in the result
- B+ tree search for the desired index page
- Fetch the next  $S/R$  index leaf pages which contains the data records as well

- B+ tree search to find index page for the insertion point
- create a data entry for the inserted record in the index page. In worst case, index page has no extra space and page split cascades up. Write index pages

- B+ tree search for the desired index page and record
- Delete the record
- In the worst case, the index page is underfilled after deletion and needs to be rebalanced

# Running Comparison

Op	Heap	Sorted	Heap+Tree	Heap+Hash	Clustered File
Scans	$B * D$	$B * D$	$B * D$	$B * D$	$B * D$
Point Query	$B * D$	$D \log B$	$D \log_F B + D$	$2 * D$	$D \log_F B$
Range Query	$B * D$	$D \log B + \lfloor S/R \rfloor * D$	$D \log_F B + \lfloor S/R \rfloor * D + S * D$	$B * D$	$D \log_F B + \lfloor S/R \rfloor * D$
Insert	$2 * D$	$D \log B + 2 * B * D$	$2 * D + 3 * D \log_F B$	$4 * D$	$3 * D \log_F B$
Delete	$2 * B * D$	$D \log B + 2 * B * D$	$D \log_F B + 2 * B * D$	$3 * D + 2 * B * D$	$2 * D \log_F B$