ICS 321 Data Storage & Retrieval

Overview of Storage & Indexing (ii)

Prof. Lipyeow Lim

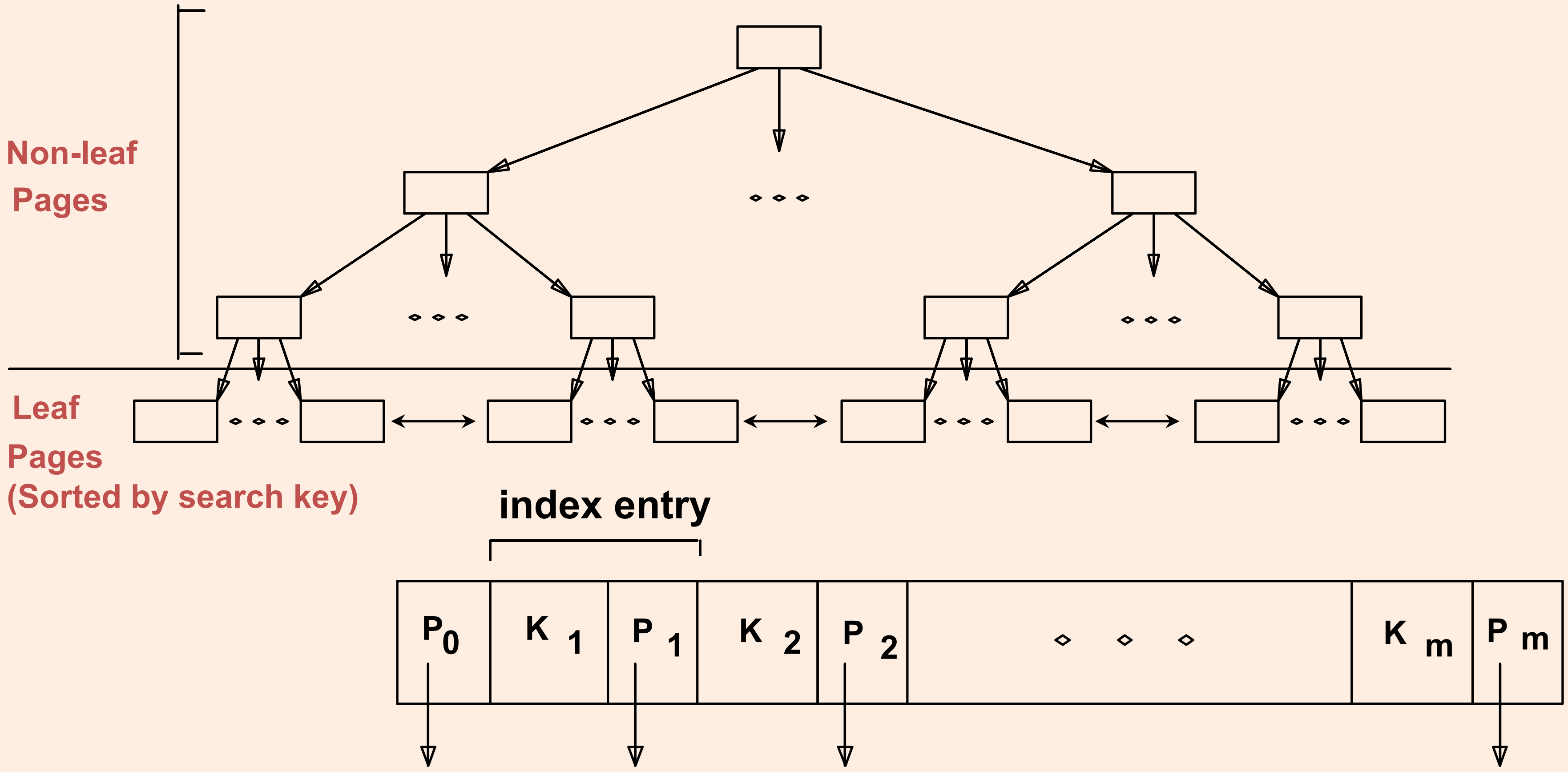
Information & Computer Science Department

University of Hawaii at Manoa

Indexes

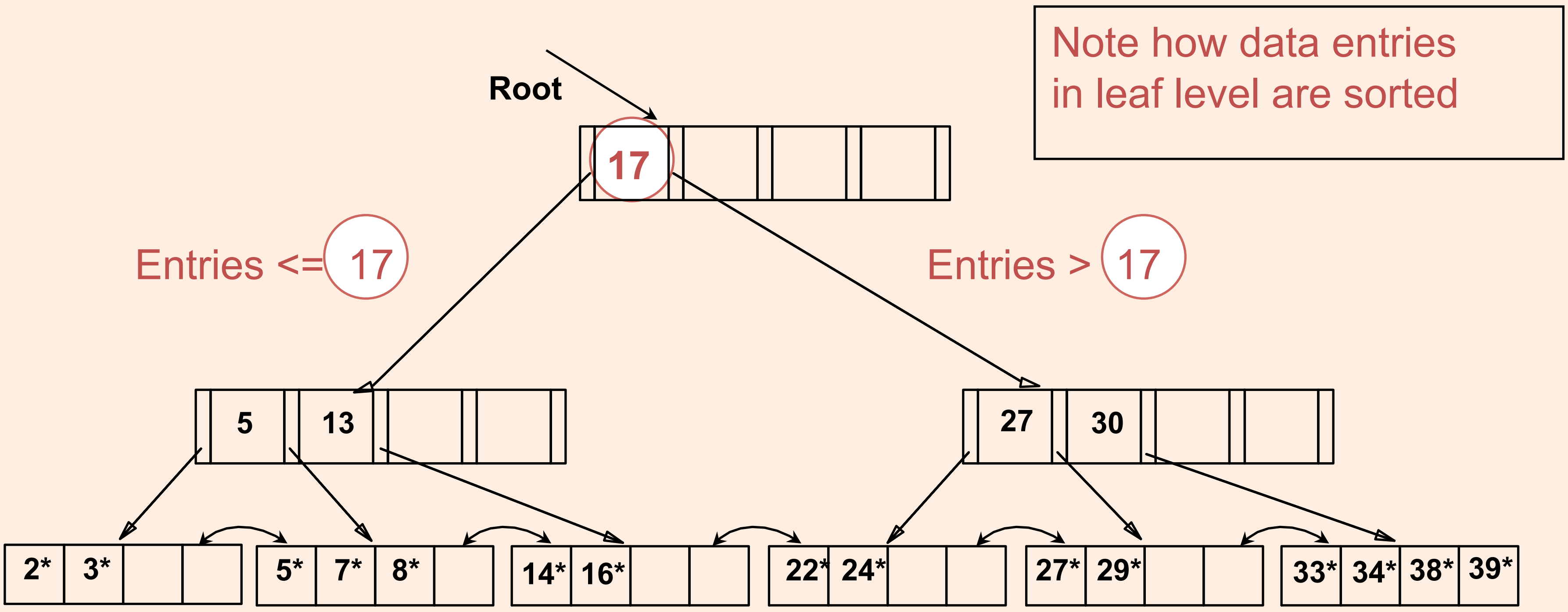
- An *index* on a file speeds up selections on the *search key fields* for the index.
  - Any subset of the fields of a relation can be the search key for an index on the relation.
  - *Search key* is not the same as *key* (minimal set of fields that uniquely identify a record in a relation).
- An index 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 <key, rid>
  - 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



Note how data entries in leaf level are sorted

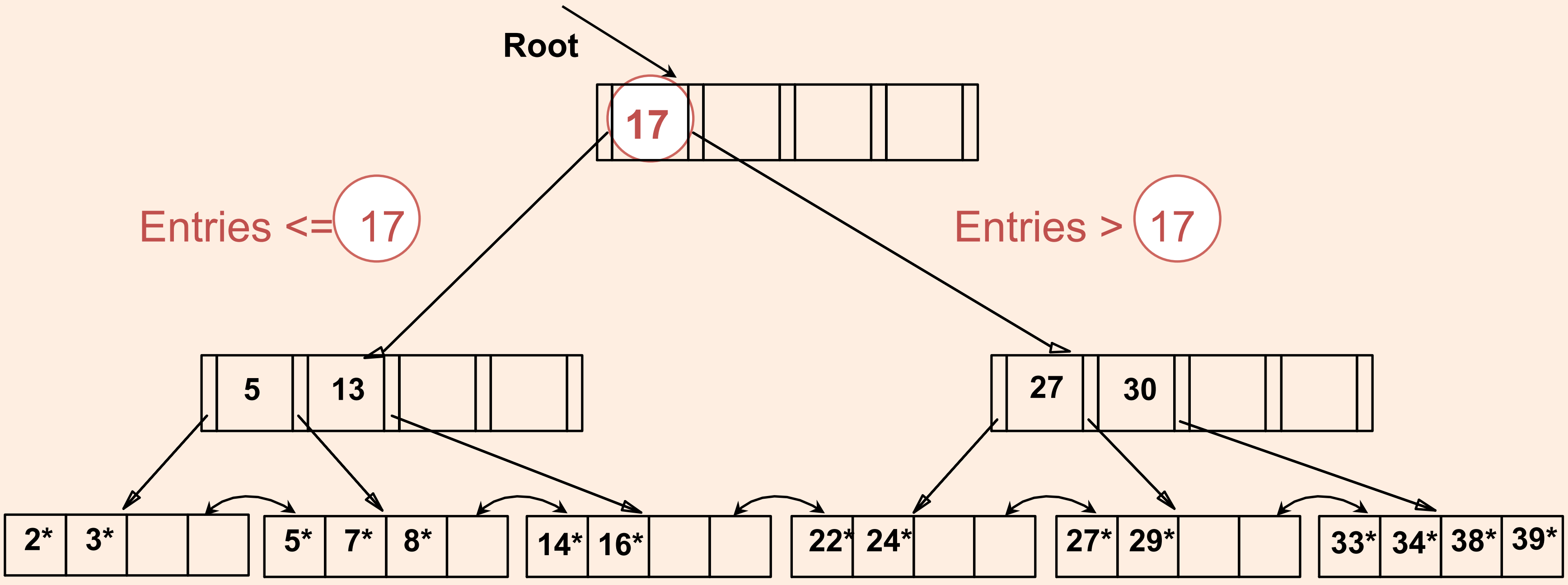
- Find 28*? 29*? All > 15* and < 30*
- Insert/delete: Find data entry in leaf, then change it. Need to adjust parent sometimes.
  - And change sometimes bubbles up the tree

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

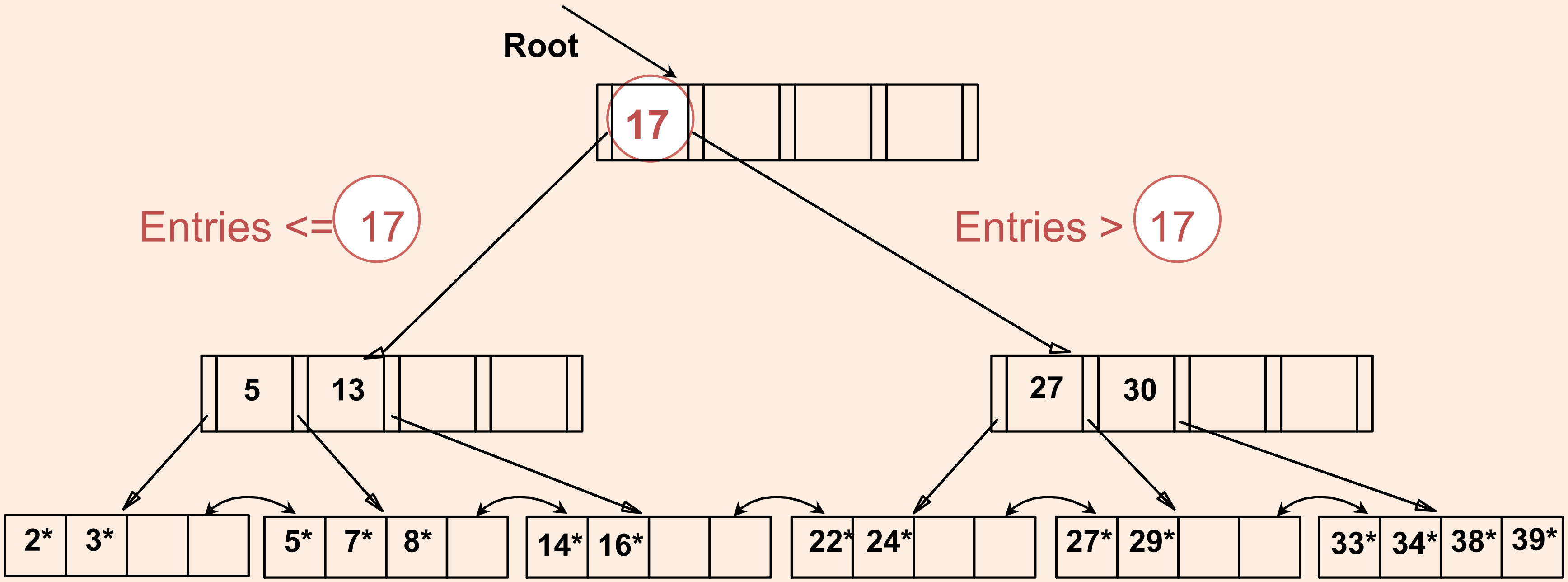


Range Queries using B+ Trees

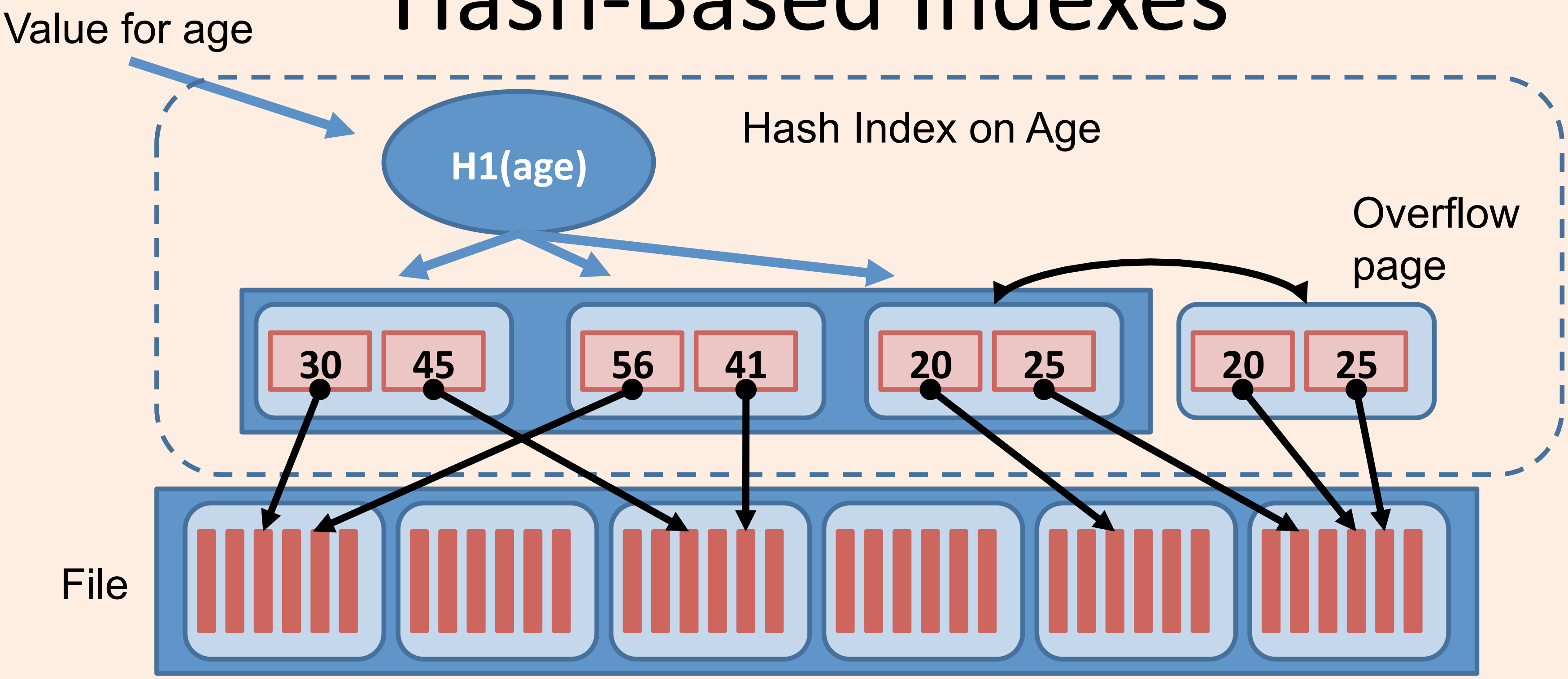
```
SELECT *
FROM Employees
WHERE age>30
```

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



Hash-Based Indexes



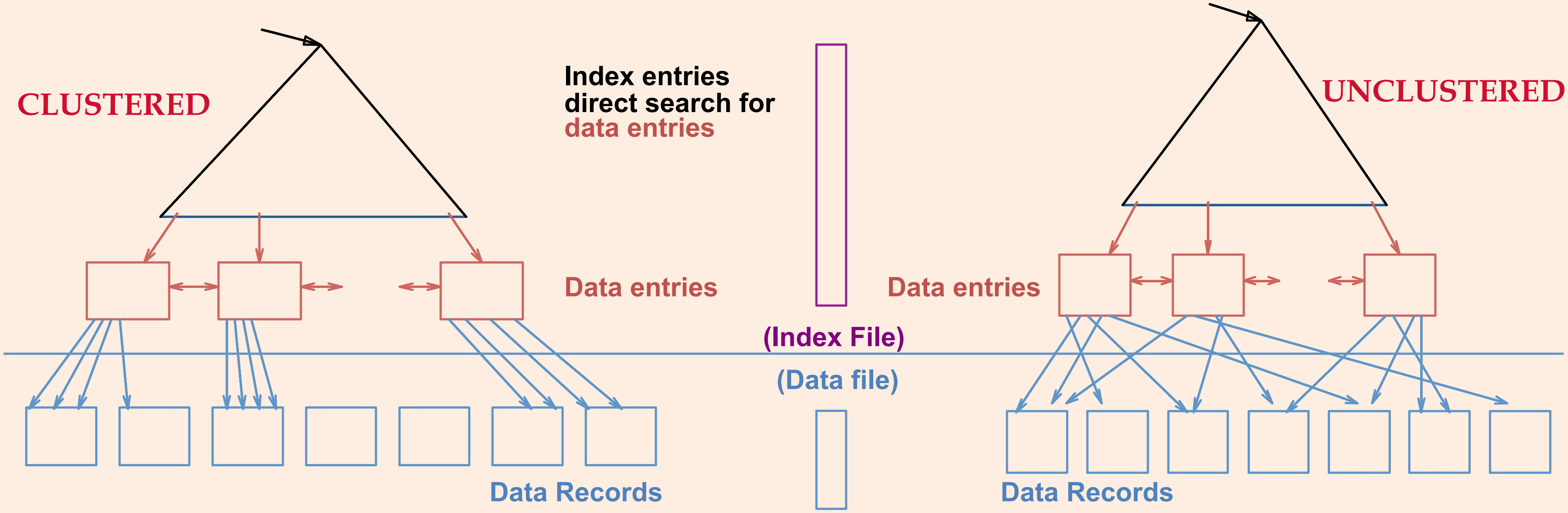
- Index is a collection of ***buckets*** that contain data entries
  - Bucket = *primary* page plus zero or more *overflow* pages.
- *Hashing function* **h**: **h**(*r*) = bucket in which (data entry for) record *r* belongs. **h** looks at the *search key* fields of *r.*
- *No "index entries" in this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

Index Classifications

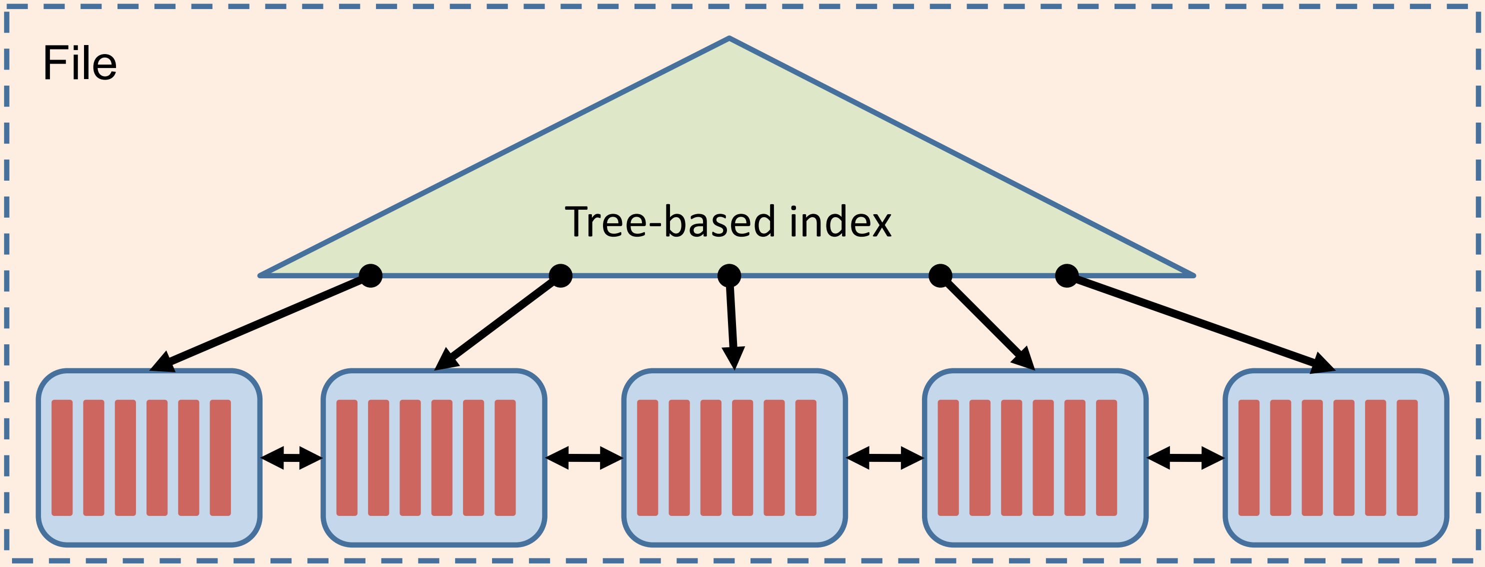
- What should be in a Data Entry k* ?
  - Possibilities:
    - The data record itself with key value k
    - <k, rid of data record with key value k>
    - <k, list of rids of data records with key value k>
      - Variable size data entries
  - Applies to any indexing technique
- Primary vs Secondary
  - 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