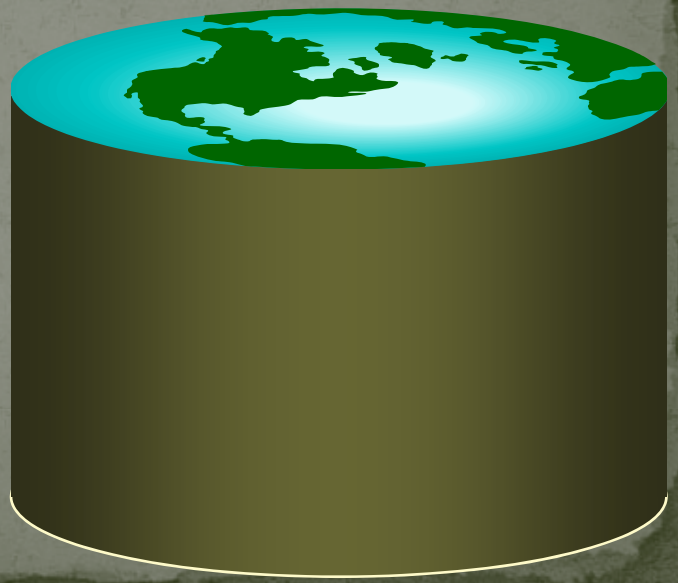
CS 505: Intermediate Topics to Database Systems

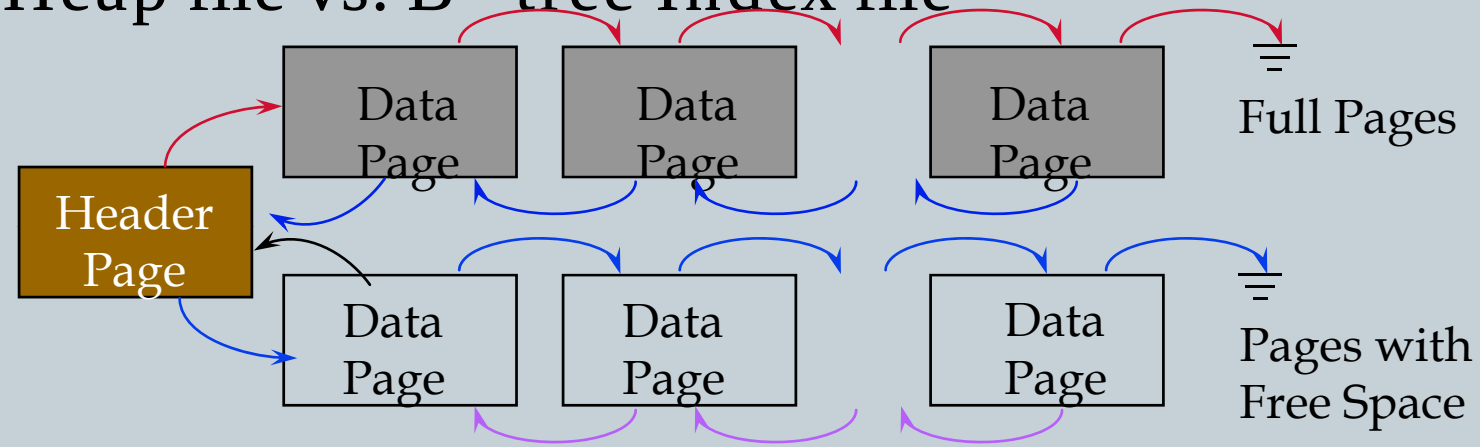
Instructor: Jinze Liu

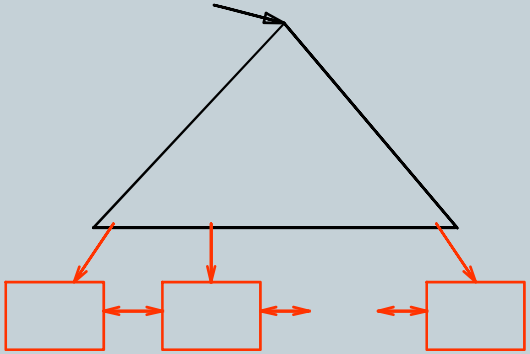
Fall 2008



Index and Data File Organization

- Heap file vs. B$^+$-tree Index file



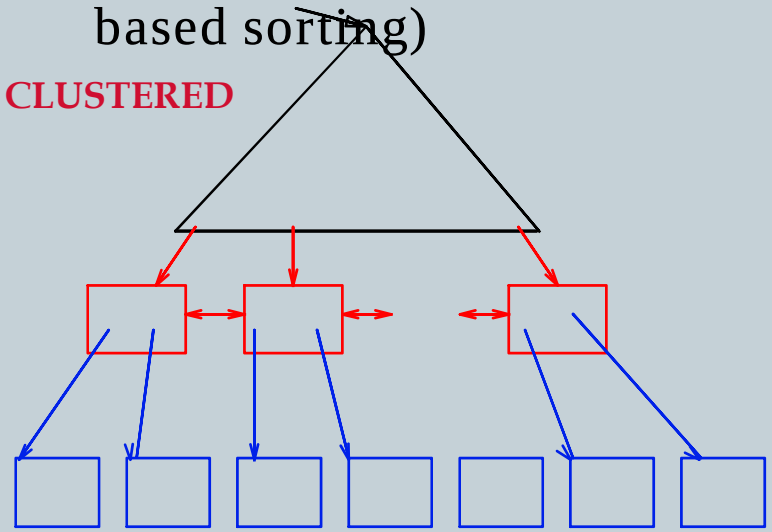


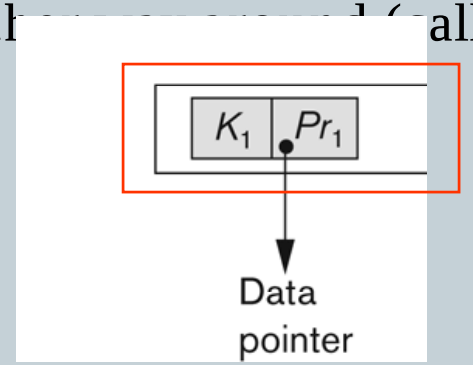
Alternatives for Data Entry in Index

- Three alternatives:
  - Primary indexes
  - Secondary indexes
  - Clustering indexes
- Can have multiple (different) indexes per file.
  - E.g. Employee = (*EID*, *name*, *age*, *salary*)
  - Primary key is *EID*, *name* is unique, *age* and *salary* are non-key attributes
  - We may (or not) sort the file by *EID*, with a B^+-tree index on *EID (primary)*, *name (secondary)* and *age (secondary)*, and a hash index on *salary*.

Primary indexes

- Index for primary key: <**k**, bid of sorted data records>
  - File is sorted according to the primary key
  - The first record in each block is called the anchor record and is used to build the index
    - Sometime the thing goes the oth[illegible]alled B$^+$-tree based sorting)

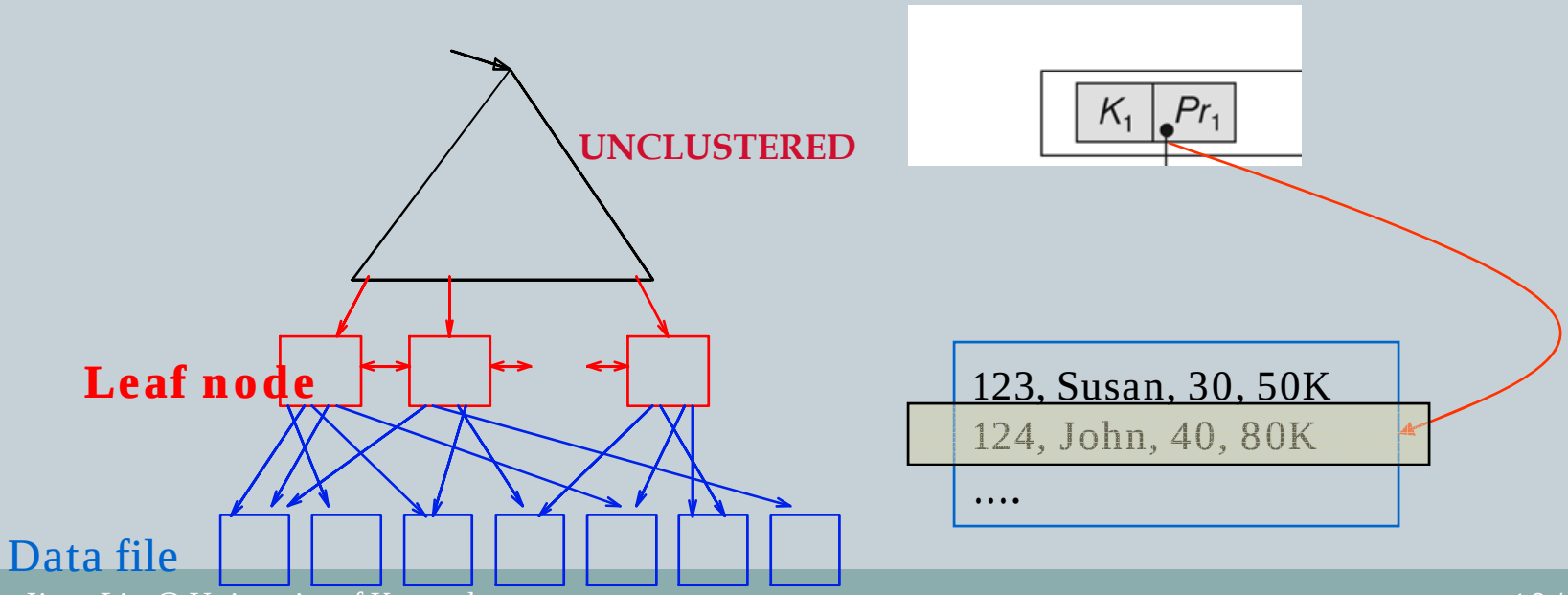




123, Susan, 30, 50K
124, John, 40, 80K
....

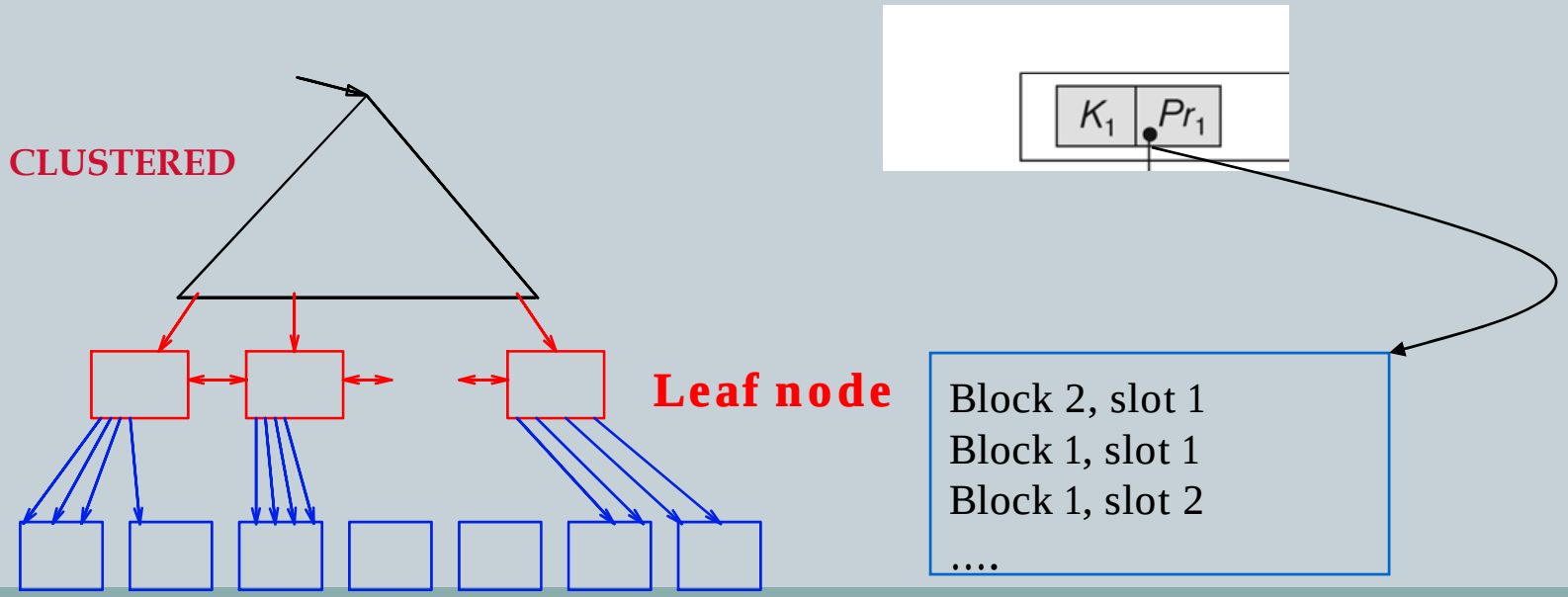
Secondary Index

- Alternative 2: <**k**, rid of matching data record>
  - Easier to maintain (do not need to sort the data file)
  - k must be a candidate key of the relation, e.g. unique attribute
  - Data pointer is the RID of the record indexed by k
    - E.g. (Block#, slot#)



Clustering Index

- Alternative 3: <**k**, list of rids of matching data records>
  - Also need to sort the data file
  - k must be a non-key attribute of the relation
  - Data pointer is the BID of the first RID that is indexed by k

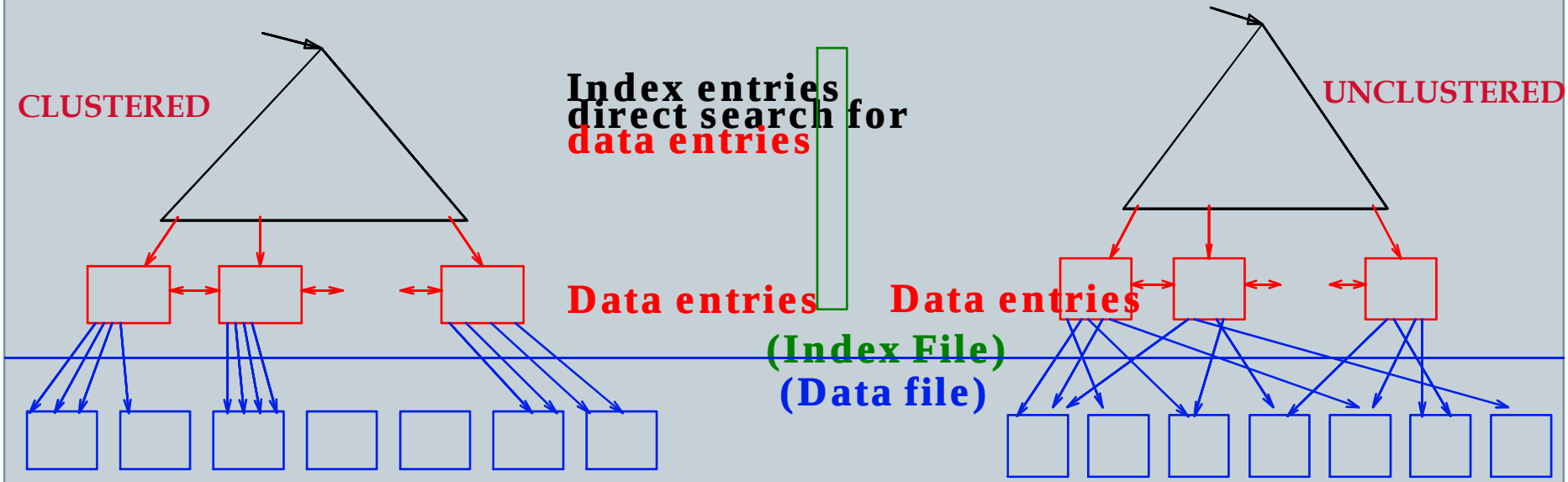


Index Classification

- *Clustered* vs. *unclustered*: If order of data records is the same as, or `close to', order of index data entries, then called *clustered index*.
  - A file can be clustered on at most one search key.
  - Cost of retrieving data records through index varies *greatly* based on whether index is clustered or not!

Clustered vs. Unclustered Index

- Suppose that clustering index is used for data entries, and that the data records are stored in a Heap file.
  - To build clustering index, first sort the Heap file (with some free space on each block for future inserts).
  - Overflow blocks may be needed for inserts. (Thus, order of data recds is `close to', but not identical to, the sort order.)



Comparison

Index	Dense (D)/Sparse (S)	Clustered (C) or not (N)	Sorting required
Primary	S	C	Yes
Secondary	D	N	No
Clustering	S	C	Yes

Cost of Operations

B: The number of data pages
R: Number of records per page
D: (Average) time to read or write disk page

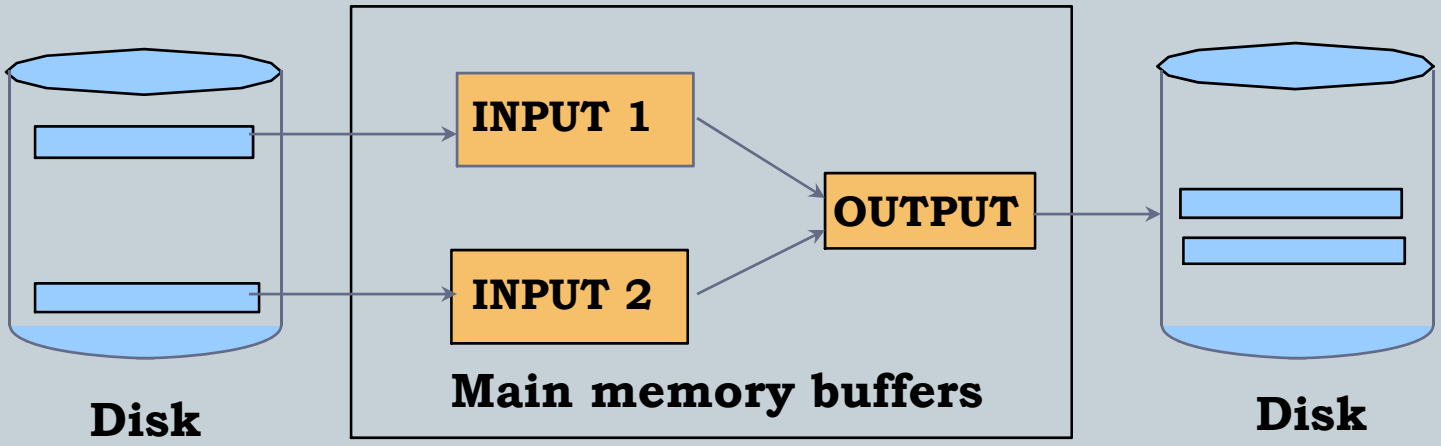
	Heap File
Scan all records	BD
Equality Search	0.5 BD
Range Search	BD
Insert	2D
Delete	search + D

Why Sort?

- A classic problem in computer science!
- Data requested in sorted order
  - e.g., find students in increasing *gpa* order
- Sorting is first step in *bulk loading* B+ tree index.
- Sorting useful for eliminating *duplicate copies* in a collection of records (Why?)
- Sorting is useful for summarizing related groups of tuples
- *Sort-merge* join algorithm involves sorting.
- Problem: sort 100Gb of data with 1Gb of RAM.
  - why not virtual memory?

2-Way Sort: Requires 3 Buffers

- Pass 0: Read a page, sort it, write it.
  - only one buffer page is used (as in previous slide)
- Pass 1, 2, 3, …, etc.:
  - requires 3 buffer pages
  - merge pairs of runs into runs twice as long
  - three buffer pages used.



Two-Way External Merge Sort

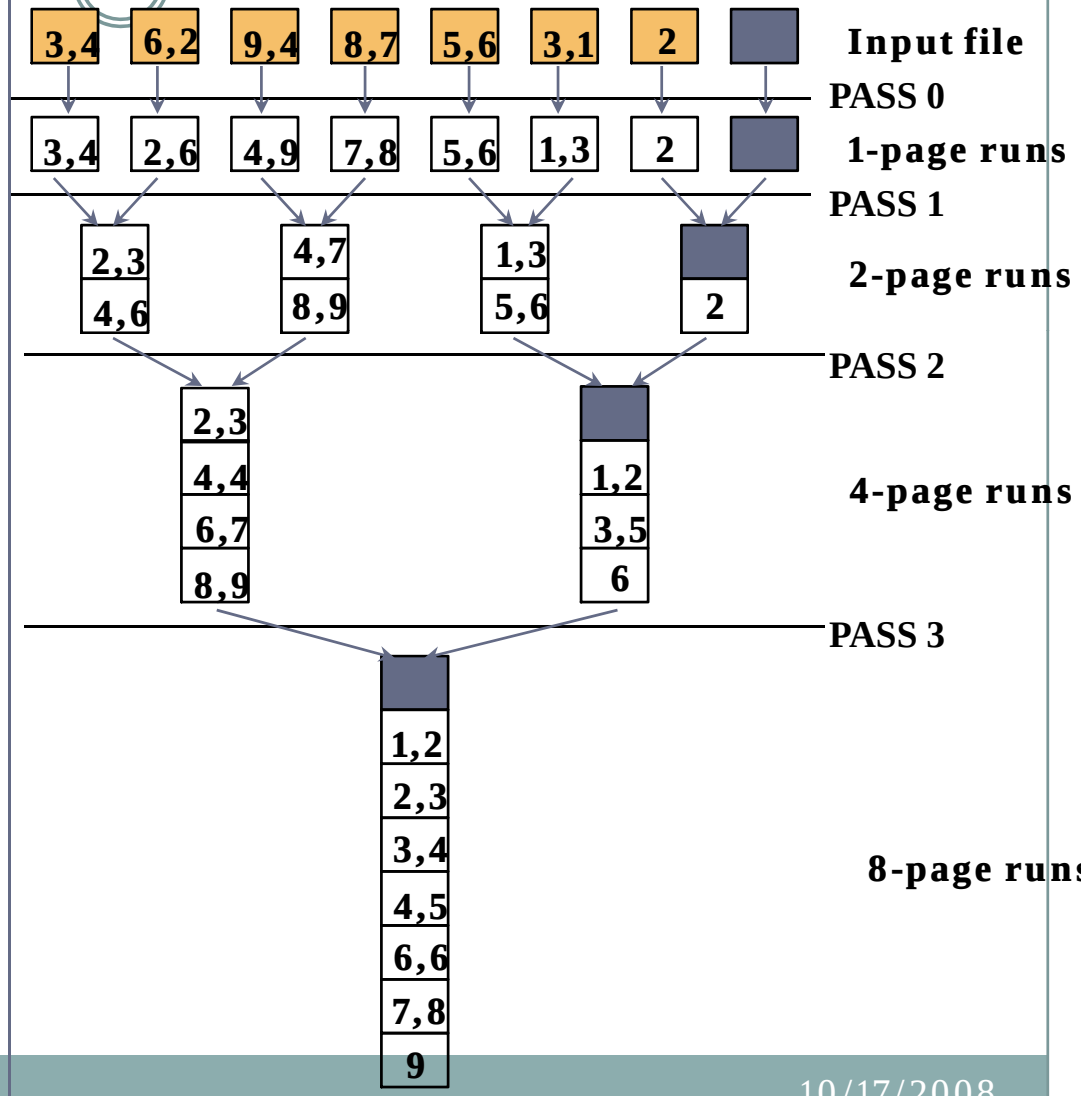
- Each pass we read + write each page in file.
- N pages in the file => the number of passes

$$= \lceil \log_2 N \rceil + 1$$

- So total cost is:

$$2N\left(\lceil \log_2 N \rceil + 1\right)$$

- *Idea:* ***Divide and conquer:*** sort subfiles and merge

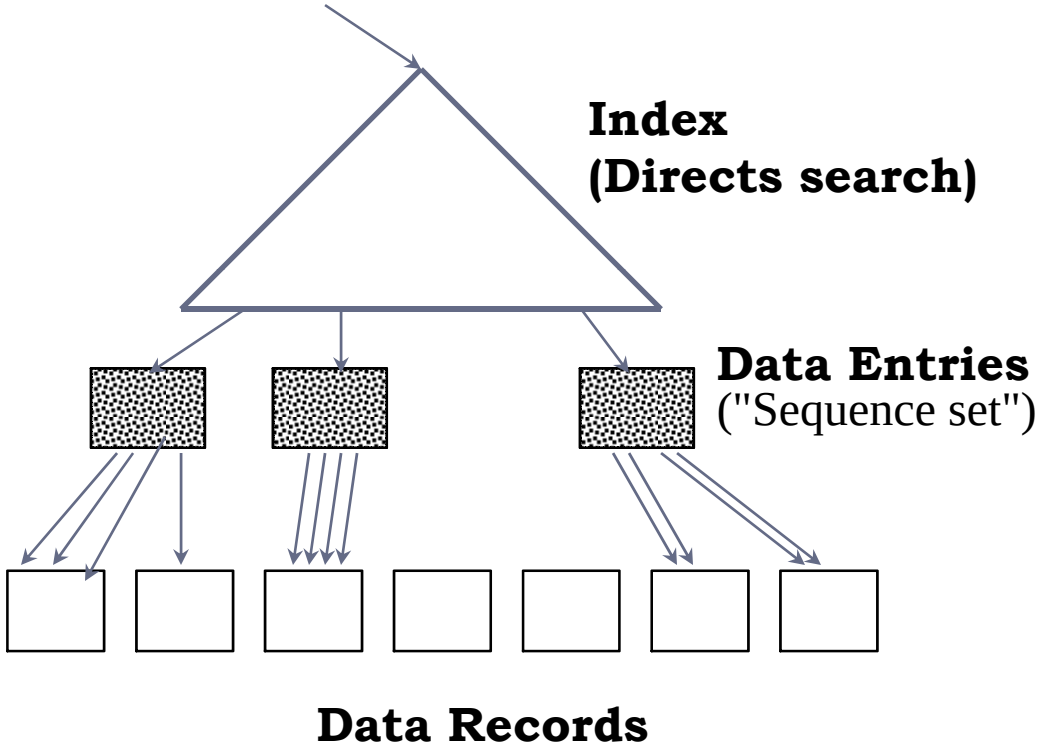


Using B+ Trees for Sorting

- Scenario: Table to be sorted has B+ tree index on sorting column(s).
- Idea: Can retrieve records in order by traversing leaf pages.
- ***Is this a good idea?***
- Cases to consider:
  - B+ tree is clustered ***Good idea!***
  - B+ tree is not clustered ***Could be a very bad idea!***

Clustered B+ Tree Used for Sorting

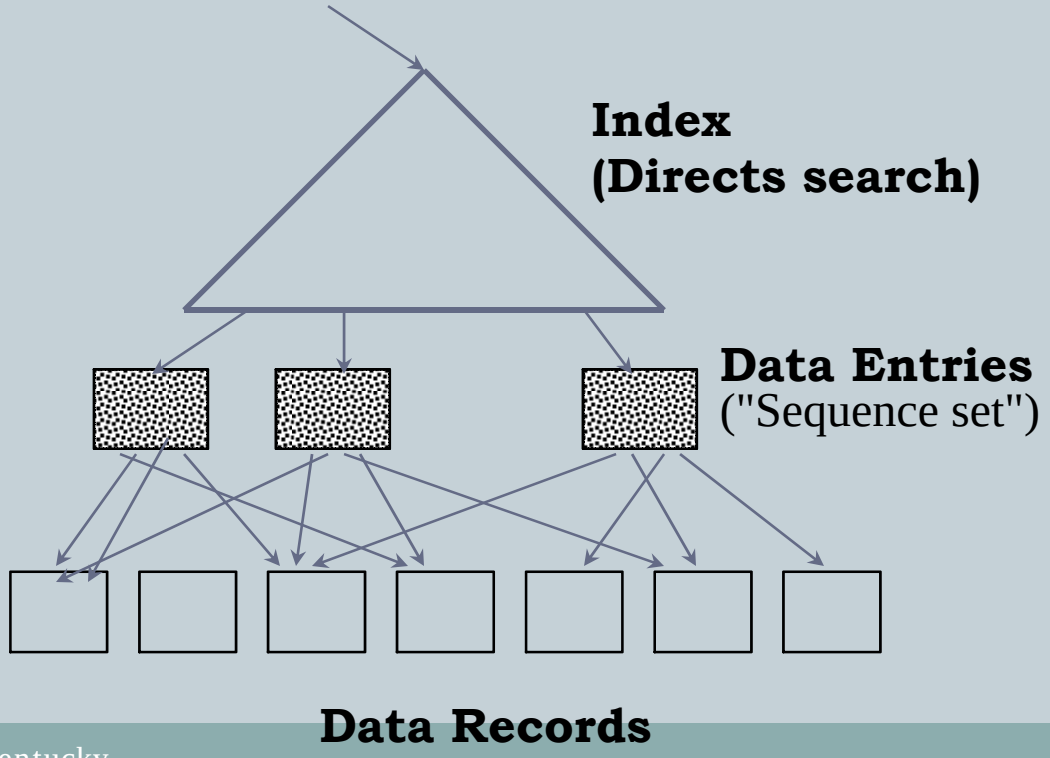
- Cost: root to the left-most leaf, then retrieve all leaf pages (primary index)
- If clustering index is used? Additional cost of retrieving data records: each page fetched just once.



* *Always better than external sorting!*

Unclustered B+ Tree Used for Sorting

- Unclustered index for data entries; each data entry contains *rid* of a data record. In general, one I/O per data record!



Summary

- Index can be used to organize files (clustered file)
- External sorting is important tool for many applications including building index and answering query (next time)
- External merge sort minimizes disk I/O cost:
  - Pass 0: Produces sorted ***runs*** of size ***B*** (# buffer pages). Later passes: ***merge*** runs.
  - # of runs merged at a time depends on ***B***, and ***block size***.
  - Larger block size means less I/O cost per page.
  - Larger block size means smaller # runs merged.
  - In practice, # of runs rarely more than 2 or 3.