CompSci 516
Database Systems

Lecture 9 and 10
Storage
and
Index

Instructor: Sudeepa Roy

Announcements (09/24)

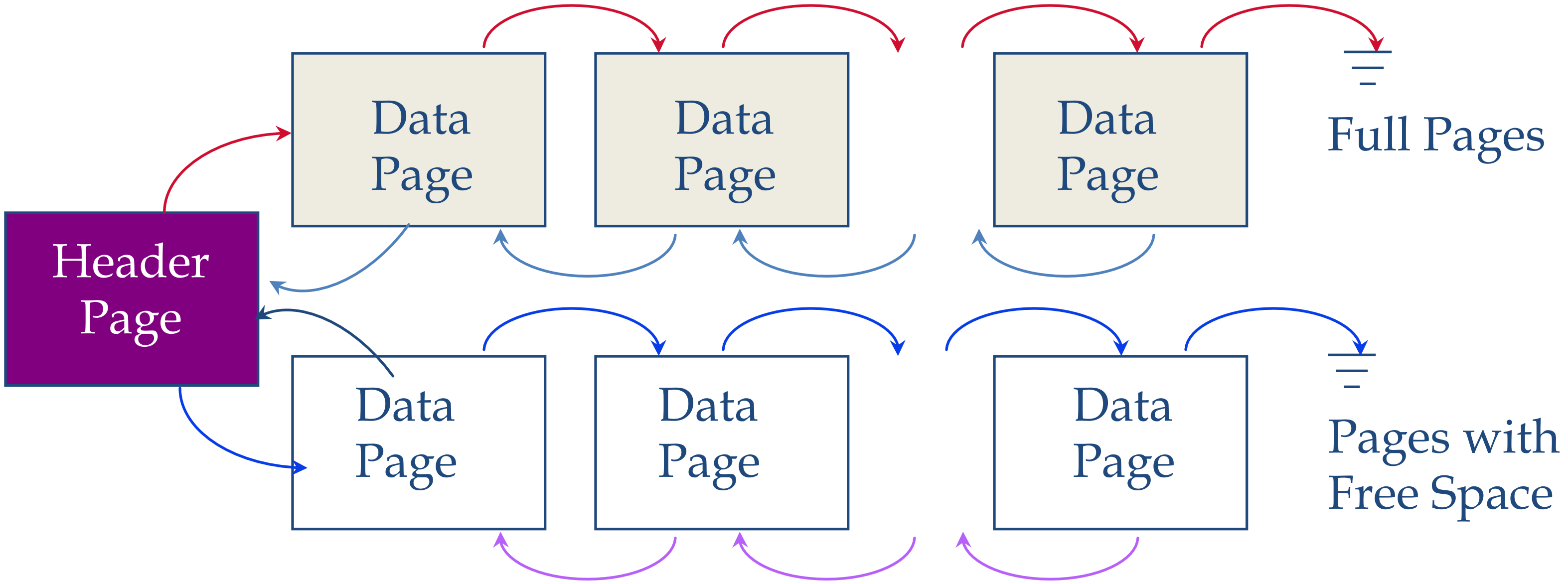
- HW1 Deadlines!
  - Today: Q4
  - Q5: next to next Tuesday 10/01
- Project details and ideas will be posted after class
  - Informal proposal due in a week 10/3 (which problem you want to work on and the group members)
  - 3-4 students in each group
  - But contact me early if you want to discuss your project ideas
  - Work on the projects more when a HW is not due!

Storage

- How are pages stored in a file?
  - Heap file (no particular order of records)
  - Sorted file (records sorted on any given field)
- How are records stored in a page?
  - Fixed length records
  - Variable length records
- How are fields stored in a record?
  - Fixed length fields/records
  - Variable length fields/records

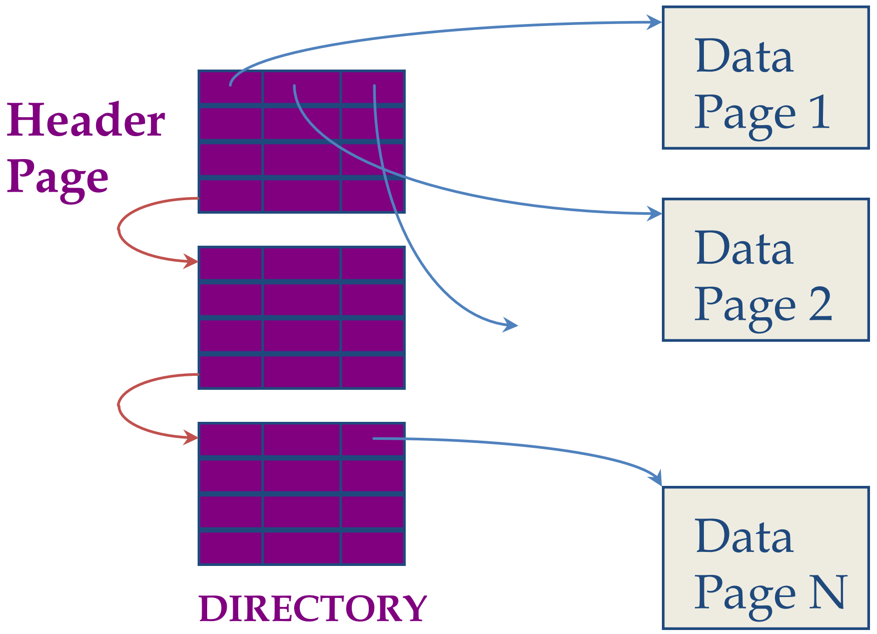
The following slides
give you the basic ideas,
exact implementation may vary

Heap File Implemented as a List



- The header page id and Heap file name must be stored someplace
- Each page contains 2 `pointers’ plus data
- But to insert a new record, we may need to scan several pages on the free list to find one with sufficient space

Heap File Using a Page Directory



- The entry for a page can include the number of free bytes on the page.
- The directory is a collection of pages
  - linked list implementation of directory is just one alternative
  - Much smaller than linked list of all heap file pages!

Storage

- How are pages stored in a file?
- How are records stored in a page?
  - Fixed length records
  - Variable length records
- How are fields stored in a record?
  - Fixed length fields/records
  - Variable length fields/records

How do we arrange a collection of records on a page?

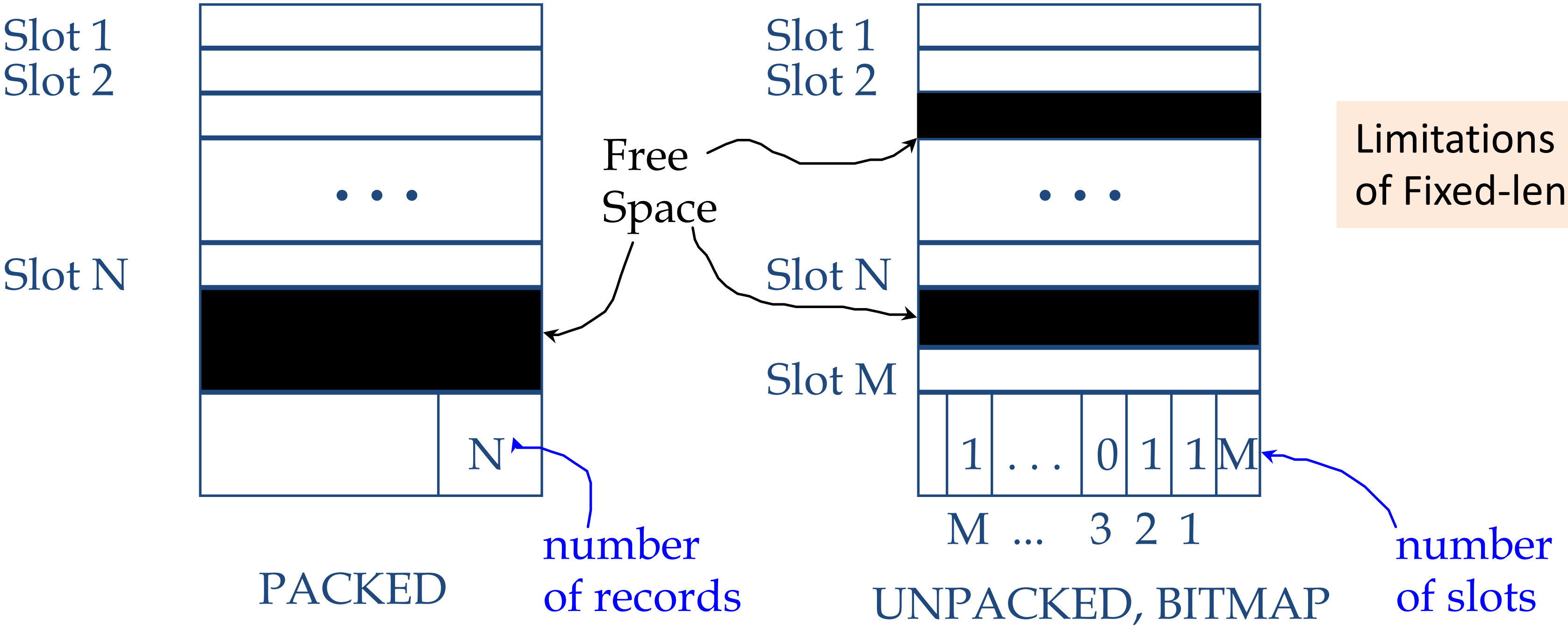
- Each page contains several slots
  - one for each record

- Record is identified by
  record id or rid = <page-id, slot-number>

- Fixed-Length Records
- Variable-Length Records

- For both, there are options for
  - Record formats (how to organize the fields within a record)
  - Page formats (how to organize the records within a page)

Page Formats: Fixed Length Records



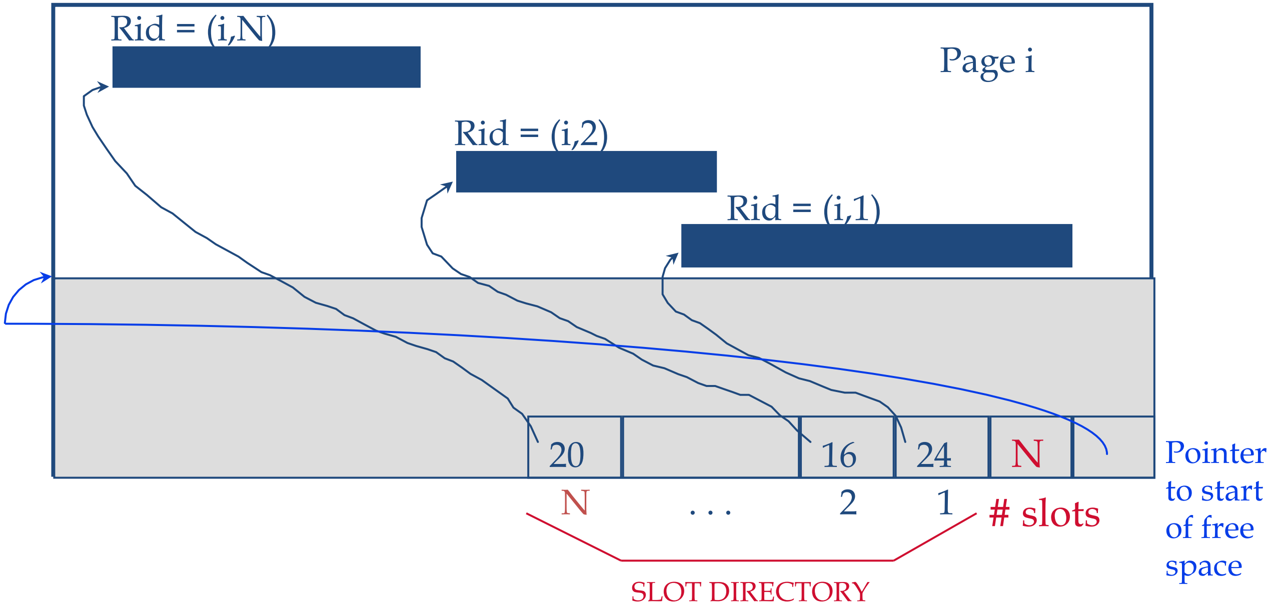
Limitations of Fixed-length?

- Record id = <page id, slot #>
- Packed: moving records for free space management changes rid; may not be acceptable or may be slow to reorganize
- Unpacked: use a bitmap – scan the bit array to find an empty slot
- Each page also may contain additional info like the id of the next page (not shown)

Page Formats: Variable Length Records

- Need to find a page with the right amount of space
  - Too small – cannot insert
  - Too large – waste of space
- if a record is deleted, need to move the records so that all free space is contiguous
  - need ability to move records within a page
  - Changes record id
- Can maintain a directory of slots (next slide)

Page Formats: Variable Length Records Directory of Slots

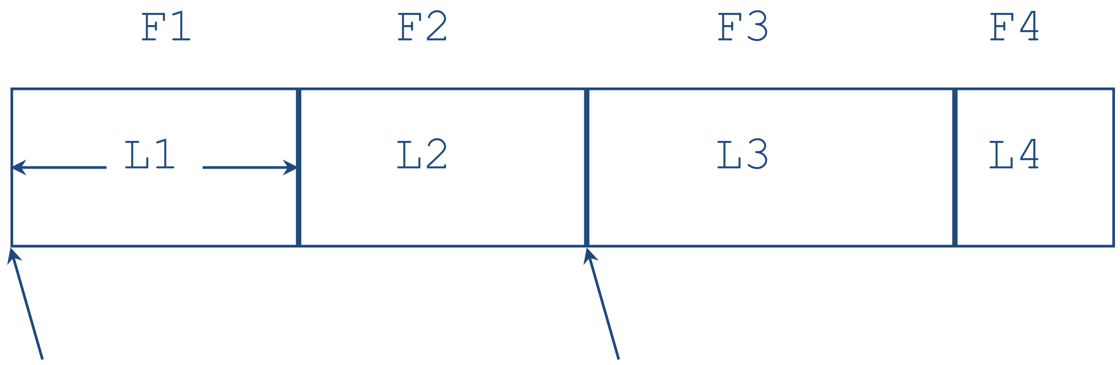


- Each slot contains <record-offset, record-length>
  - deletion = set record-offset to -1
- Record-id rid = <page, slot-in-directory> remains unchanged
  - Can move records on page without changing rid
  - so, attractive for fixed-length records too

Storage

- How are pages stored in a file?
- How are records stored in a page?
  - Fixed length records
  - Variable length records
- How are fields stored in a record?
  - Fixed length fields/records
  - Variable length fields/records

Record Formats: Fixed Length



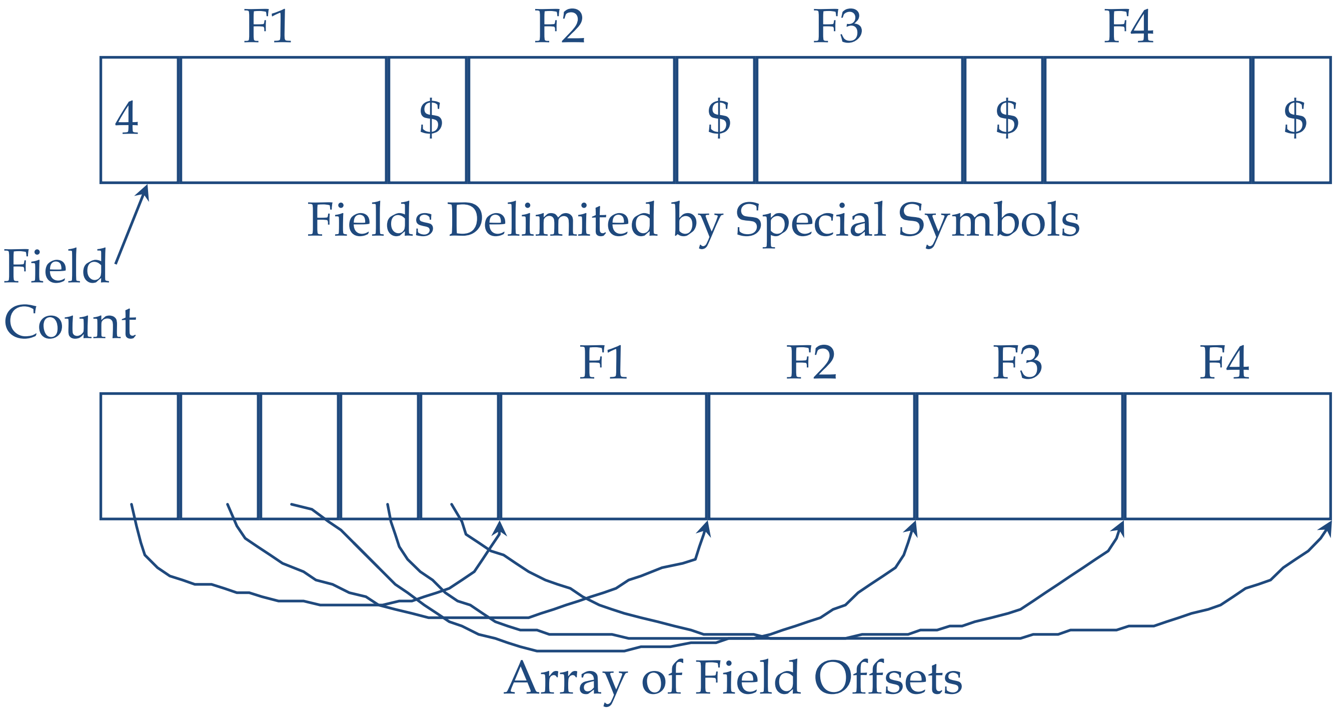
Base address (B)

Address = B+L1+L2

- Each field has a fixed length
  - for all records
  - the number of fields is also fixed
  - fields can be stored consecutively
- Information about field types same for all records in a file
  - stored in system catalogs
- Finding i-th field does not require scan of record
  - given the address of the record, address of a field can be obtained easily

Record Formats: Variable Length

- Cannot use fixed-length slots for records
- Two alternative formats (note: # fields is fixed for relational data)



1. use delimiters

2. use offsets at the start of each record

- Second offers direct access to i-th field, efficient storage of nulls (special don't know value); small directory overhead

Main takeaways: storage

- Disk is slow but large and persistent
- Main memory or buffer is fast but small and not persistent
- If a page is edited in memory, needs to be written back to disk
- Unit of cost = page I/O (read and write)
- A record (= tuple) is accessed by rid (record id): gives the address of the page and the slot

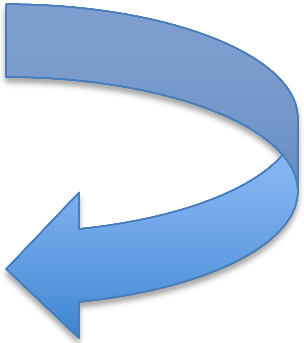
Indexes

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”!
- An index contains a collection of data entries, and supports efficient retrieval of all data entries k* with a given key value k
  - Why multiple entries for a given k?

Remember: Terminology

- Index search key (key): k
  - Used to search a record



INDEX
does this

- Data entry : k*
  - Pointed to by k
  - Contains record id(s) or record itself
- Records or data
  - Actual tuples
  - Pointed to by record ids

Alternatives for Data Entry **k*** in Index **k**

Advantages/
Disadvantages?

- In a data entry k* we can store:
  1. (Alternative 1) The actual data record with key value **k,** or
  2. (Alternative 2) <**k**, rid>
     - rid = record of data record with search key value **k**, or
  3. (Alternative 3) <**k**, rid-list>
     - list of record ids of data records with search key **k**>

- Choice of alternative for data entries is orthogonal to the indexing technique used to locate data entries with a given key value **k**

Alternatives for Data Entries: Alternative 1

- In a data entry k* we can store:
  1. The actual data record with key value **k**
  2. <**k**, rid>
     - rid = record of data record with search key value **k**
  3. <**k**, rid-list>
     - list of record ids of data records with search key **k**>

- Index structure is a file organization for data records
  - instead of a Heap file or sorted file
- At most one index can use Alternative 1
  - Otherwise, data records are duplicated, leading to redundant storage and potential inconsistency
- Problem with Alt-1: If data records are very large, #pages with data entries is high
  - Implies size of auxiliary information in the index is also large

Alternatives for Data Entries: Alternative 2, 3

- In a data entry k* we can store:
  1. The actual data record with key value **k**
  2. <**k**, rid>
     - rid = record of data record with search key value **k**
  3. <**k**, rid-list>
     - list of record ids of data records with search key **k**>

- Data entries typically much smaller than data records
  - So, better than Alternative 1 with large data records
  - Especially if search keys are small.
- Alternative 3 more compact than Alternative 2
  - but leads to variable-size data entries even if search keys have fixed length.

Index Classification

- Primary vs. secondary
- Clustered vs. unclustered
- Tree-based vs. Hash-based

Primary vs. Secondary Index

- If search key contains primary key, then called primary index, otherwise secondary
  - Unique index: Search key contains a candidate key

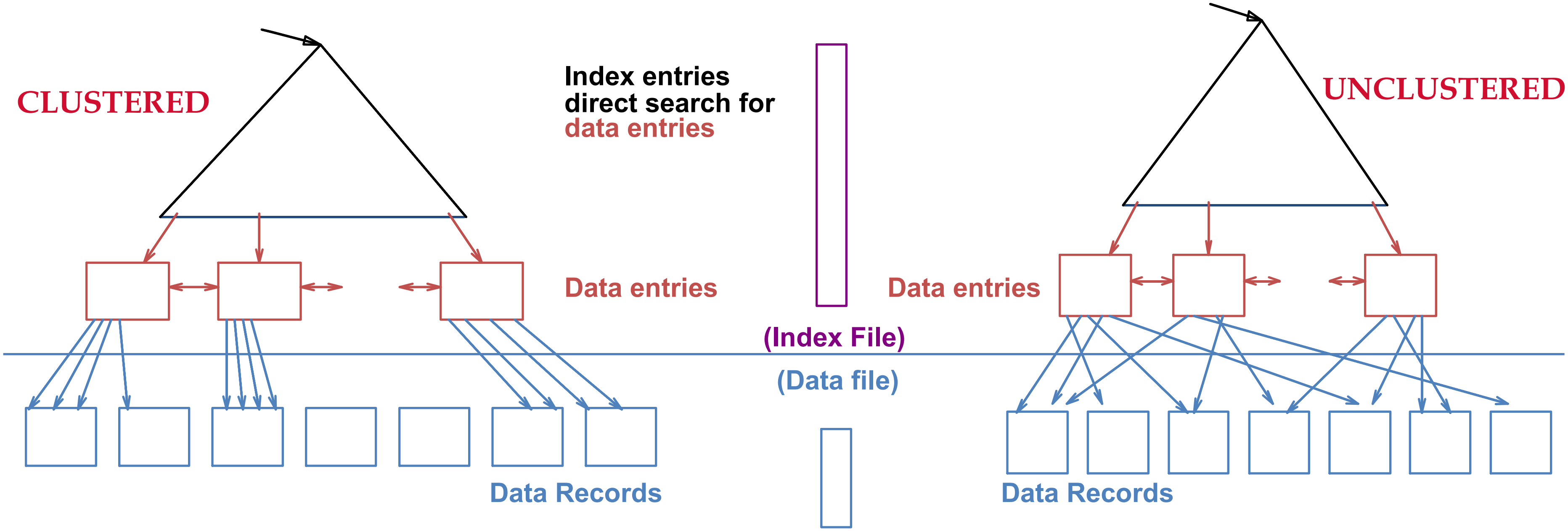
- Duplicate data entries:
  - if they have the same value of search key field k
  - Primary/unique index never has a duplicate
  - Other secondary index can have duplicates

Clustered vs. Unclustered Index

- If order of data records in a file is the same as, or `close to’, order of data entries in an index, then clustered, otherwise unclustered
- A file can be clustered on at most one search key
- Cost of retrieving data records (range queries) through index varies greatly based on whether index is clustered or not

Clustered vs. Unclustered Index

- Suppose that Alternative (2) is used for data entries, and that the 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, data records are `close to', but not identical to, sorted


CLUSTERED
Index entries
direct search for
data entries
UNCLUSTERED
Data entries
Data entries
(Index File)
(Data file)
Data Records
Data Records