INFO20003 Database Systems

Dr Renata Borovica-Gajic

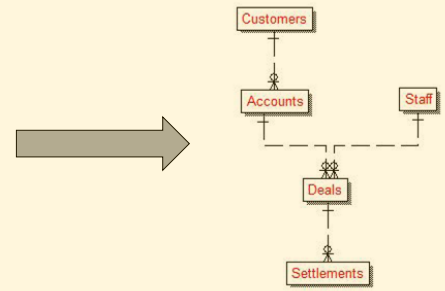
Lecture 10
Storage and Indexing

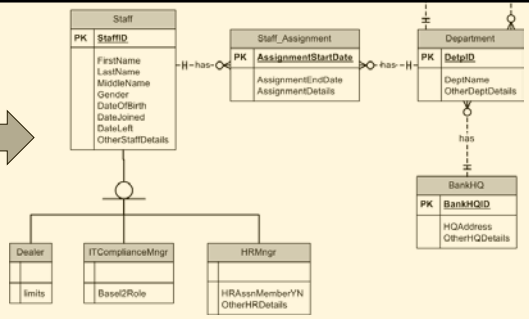
Week 5

What this subject is all about. Remember this?

Organisational Description and Problem Area

- An investment bank wants to have a database to provide it with the ability to store information about its trading operations. The bank essentially works with customers by providing the capability for trading stocks, shares and other commodities. The bank has three branches in which exist a number of departments. Departments have a department manager who supervises a number of staff within the department. A set of accounts are used to store information about the currency of the organisations operations. Accounts can be customer accounts or internal "house" accounts, each of which allow trades to be made upon them. There are a number of account types. There are many customers and customers may have one or more contacts. Customers have a facility for lending money to pay for their purchases of stocks and commodities. Staff make deals on the behalf of their customers using a funding source and keeping track of settlements on the deals being made. There are many types of deal to be made. Settlements are full or partial payments of the deals and are recorded whenever a payment is made.
- Please note that this section is purely made up and by all means is a very short description of a real investment bank (although many details have been left out and wide ranging assumptions have been made.

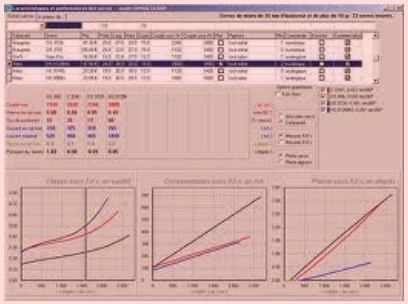




MODELLING

SQL

ARCHITECTURE / INTERNAL WORKINGS



SQL Queries

```
select val from sales
where id = max;
```

Results

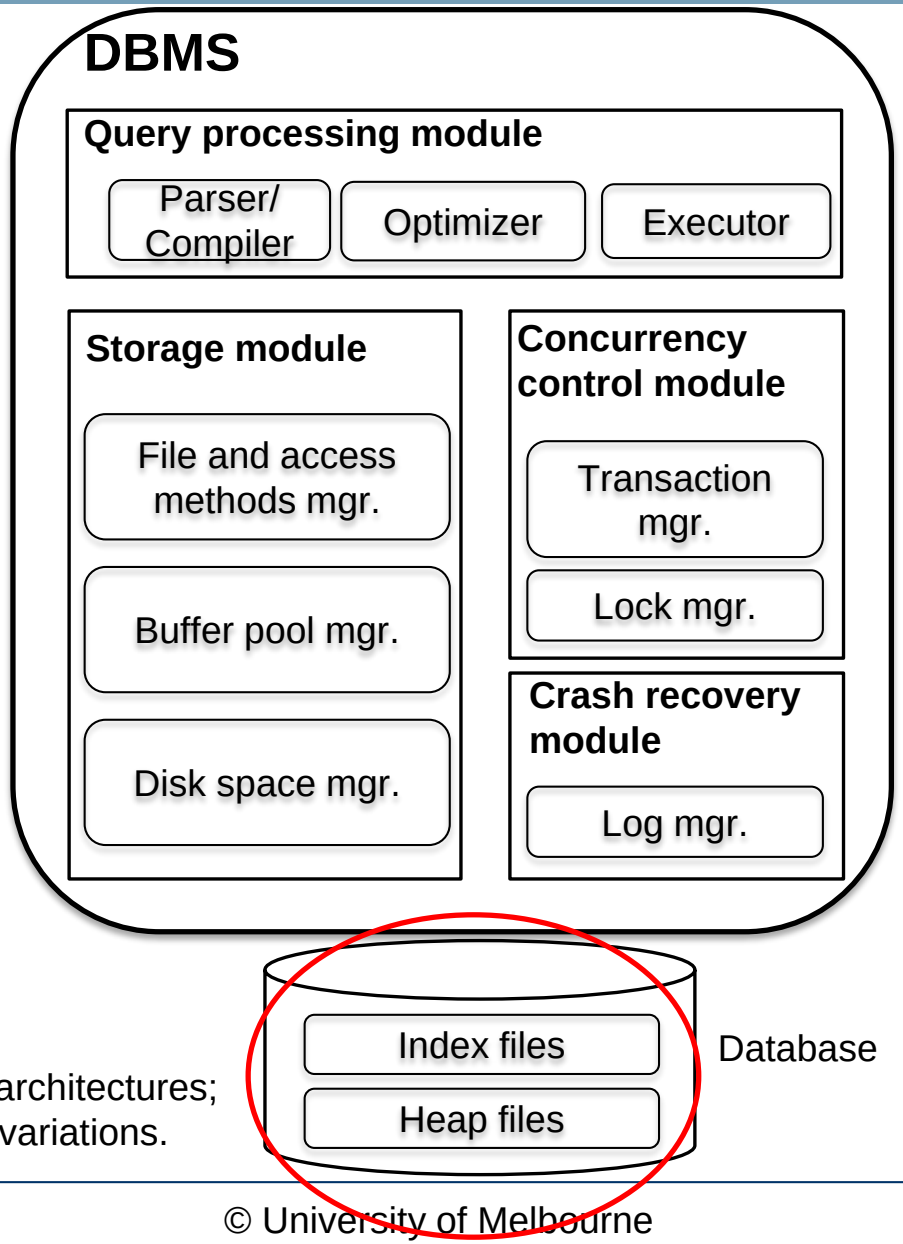
Process

Access

Store

Database System

Components of a DBMS



TODAY

This is one of several possible architectures; each system has its own slight variations.

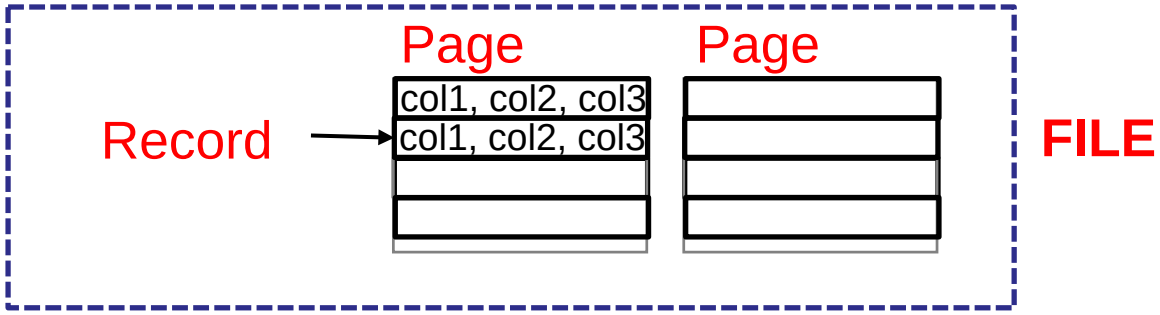
Coverage

- File organization (Heap & sorted files)
- Index files & indexes
- Index classification

Readings: Chapter 8, Ramakrishnan & Gehrke, Database Systems

Files (in a DBMS)

- **FILE**: A collection of **pages**, each containing a collection of **records**.



- DBMS must support:
  - insert/delete/modify record
  - read a particular record (specified using *record id*)
  - scan all records (possibly with some conditions on the records to be retrieved)

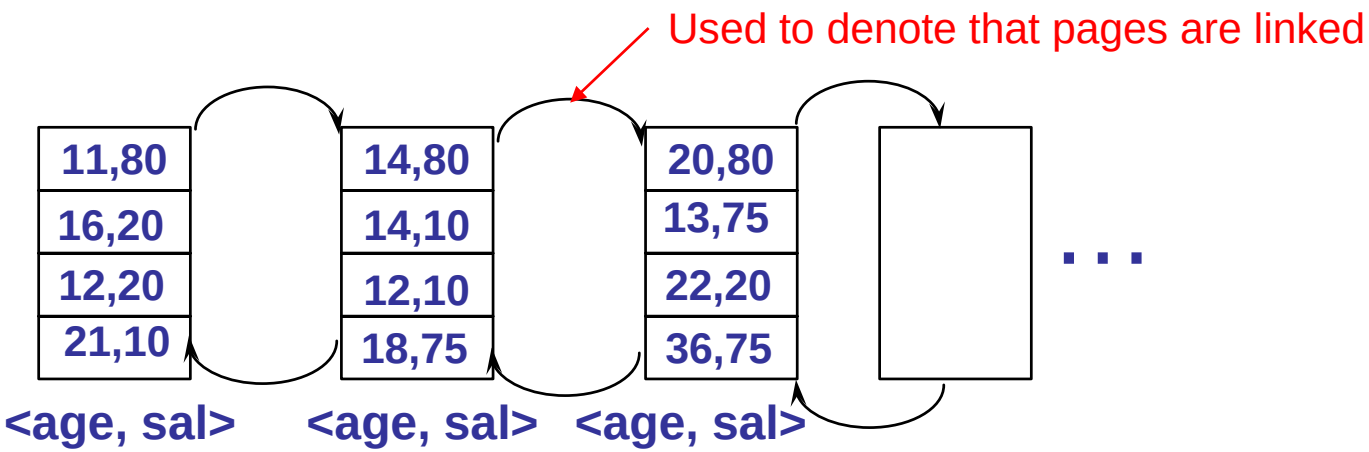
Alternative File Organizations

- Many alternatives exist, *each good for some situations, and not so good in others:*

1. **Heap files**: no particular order among records
   - Suitable when typical access is a file scan retrieving **all** records
2. **Sorted Files:** pages and records within pages are ordered by some condition
   - Best for retrieval (of a range of records) in some order
3. **Index File Organizations:**
   - Special data structure that has the fastest retrieval in some order
   - Will cover shortly..

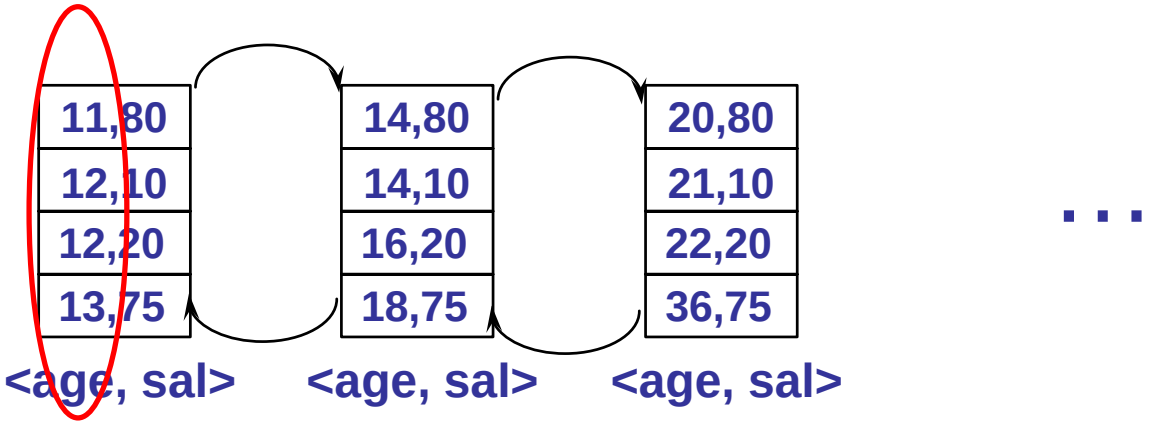
1. Heap (Unordered) Files

- Simplest file structure, contains records in **no** particular order
- As file grows and shrinks, disk pages are allocated and de-allocated
  - Fastest for inserts compared to other alternatives



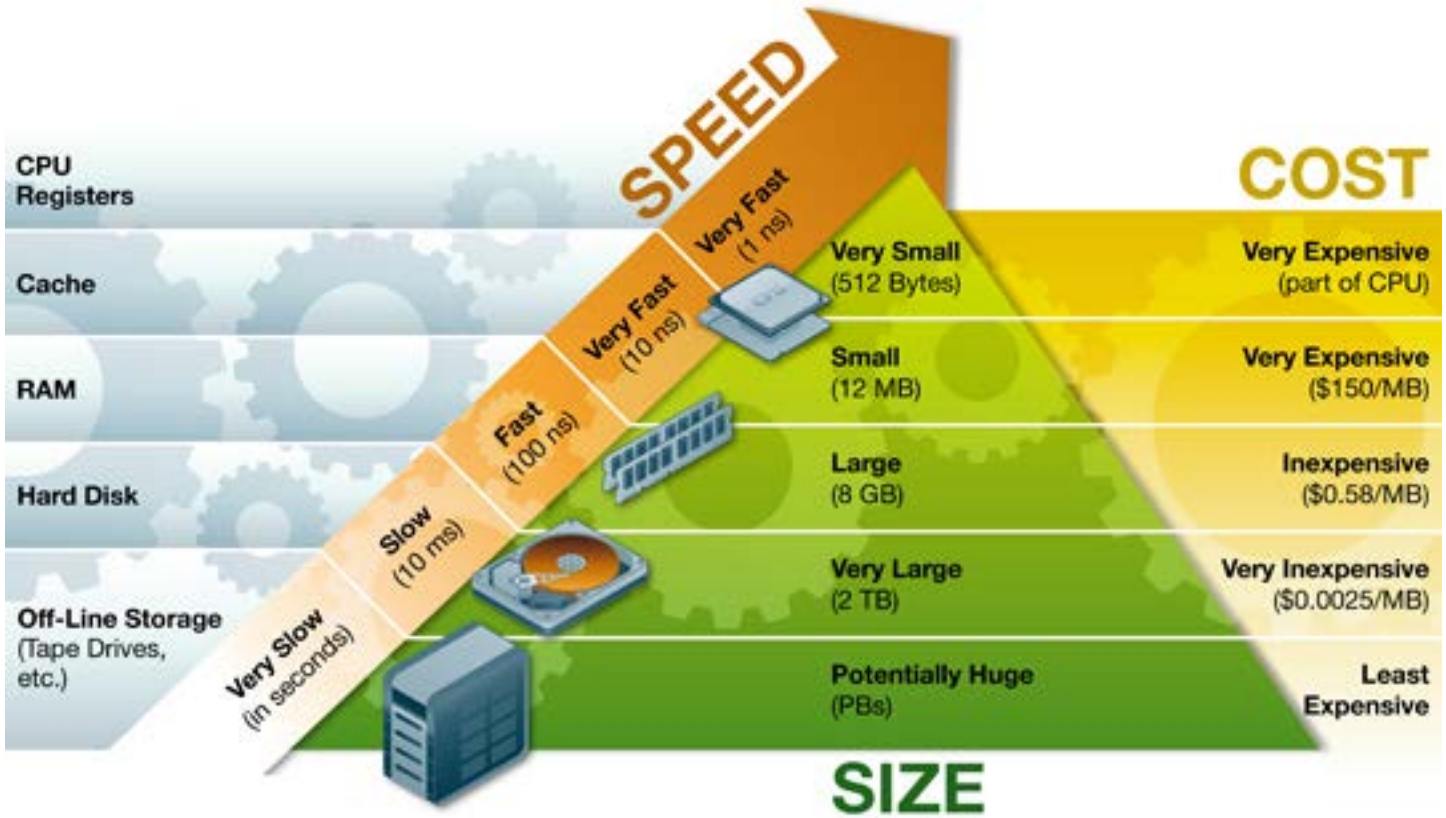
Sorted files

- Similar structure like heap files (pages and records), but pages and records are **ordered**
- Fast for range queries, but hard for maintenance (each insert potentially reshuffles records)
- **Example**: A sorted file ordered by *age*



Storage hierarchy

- Data is typically stored in pages on Hard Disks (HDD).
- *To be able to process and analyze it – data needs to be brought to Memory (RAM).*



How does a DBMS decide which option is better?

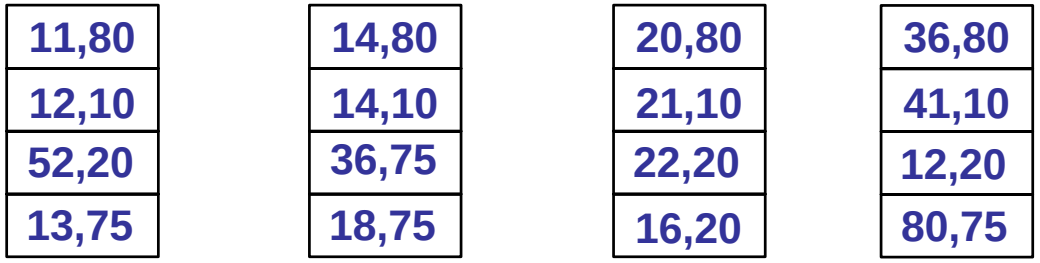
- DBMS model the **cost** of all operations
- The cost is typically expressed in **the number of page accesses** (or disk **I/O** operations – to bring data from disk to memory)
  - 1 page access (on disk) == 1 I/O (used interchangeably)

- **Example**: If we have a table of 100 records, and each page can store 10 records, what would be the cost of accessing the entire file

- **Answer**: For 100 records we have 10 pages in total (100/10), thus the cost to access the entire file is 10 I/O (or 10 pages)

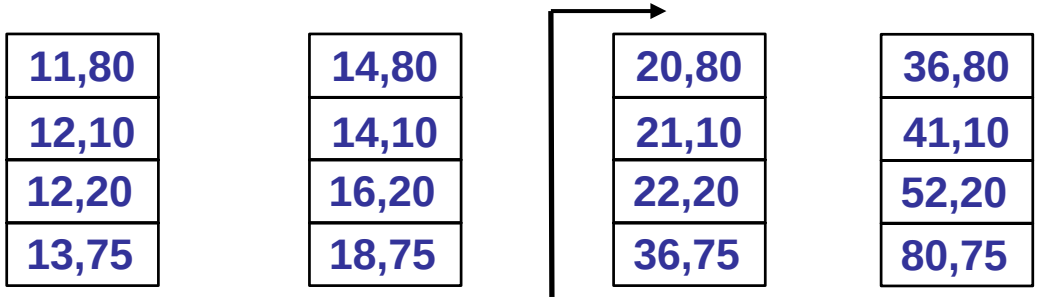
Which alternative is better?

- **Example**: *Find all records with ages between 20 and 30, for the file that has B pages. Consider both alternative: having an unsorted and sorted file. What would be the cheapest cost?*
- **20 < age <30**, num pages = B
- Heap file (no order) = B;



Heap file

- Sorted file (exploit order) = log2 B



Sorted file

File Organization & Indexing

- File organization (Heap & sorted files)
- Index files & indexes
- Index classification

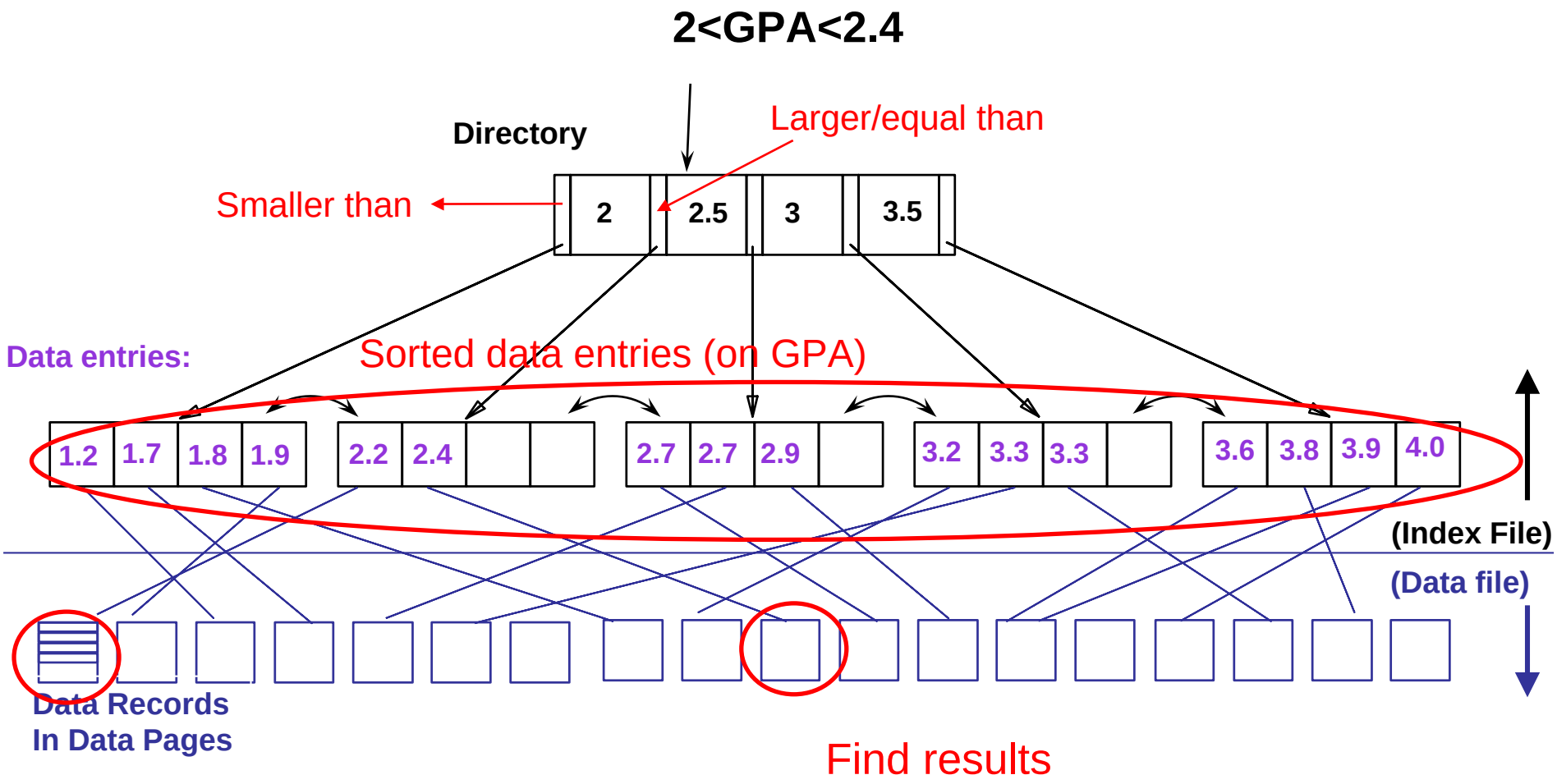
Indexes

- Sometimes, we want to retrieve records by specifying the *values in one or more fields*, e.g.,
  - Find all students in the **"CIS" department**
  - Find all students with a **gpa > 3**

- An **index** is a data structure built on top of data pages used for efficient search. The index is built over specific fields called ***search key fields.*** E.g. we can build an index on *GPA*, or department name.
  - The index speeds up selections on the *search key fields*
  - **Any** subset of the fields of a relation can be the search key for an index on the relation
  - **Note***: Search key* is not the same as *key* (e.g., doesn't have to be unique)

Example: Simple Index on GPA

An index contains a collection of **data entries**, and supports efficient retrieval of **data records** matching a given **search condition**



File Organization & Indexing

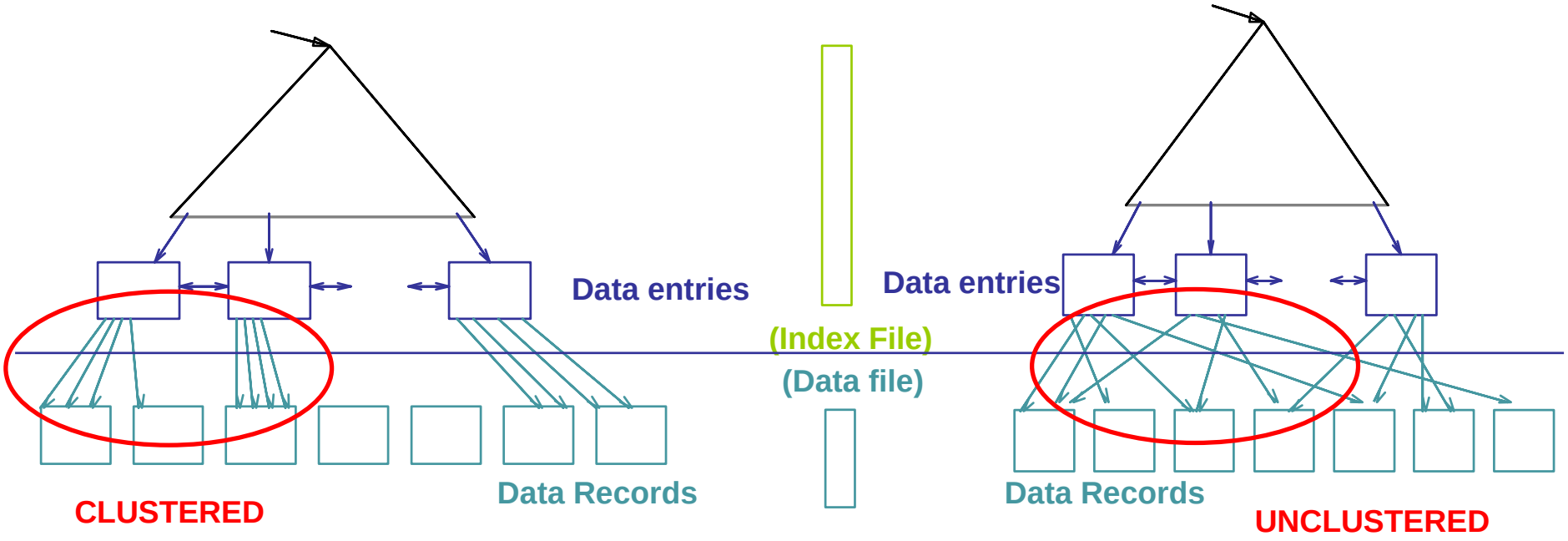
- File organization (Heap & sorted files)
- Index files & indexes
- Index classification

Index Classification

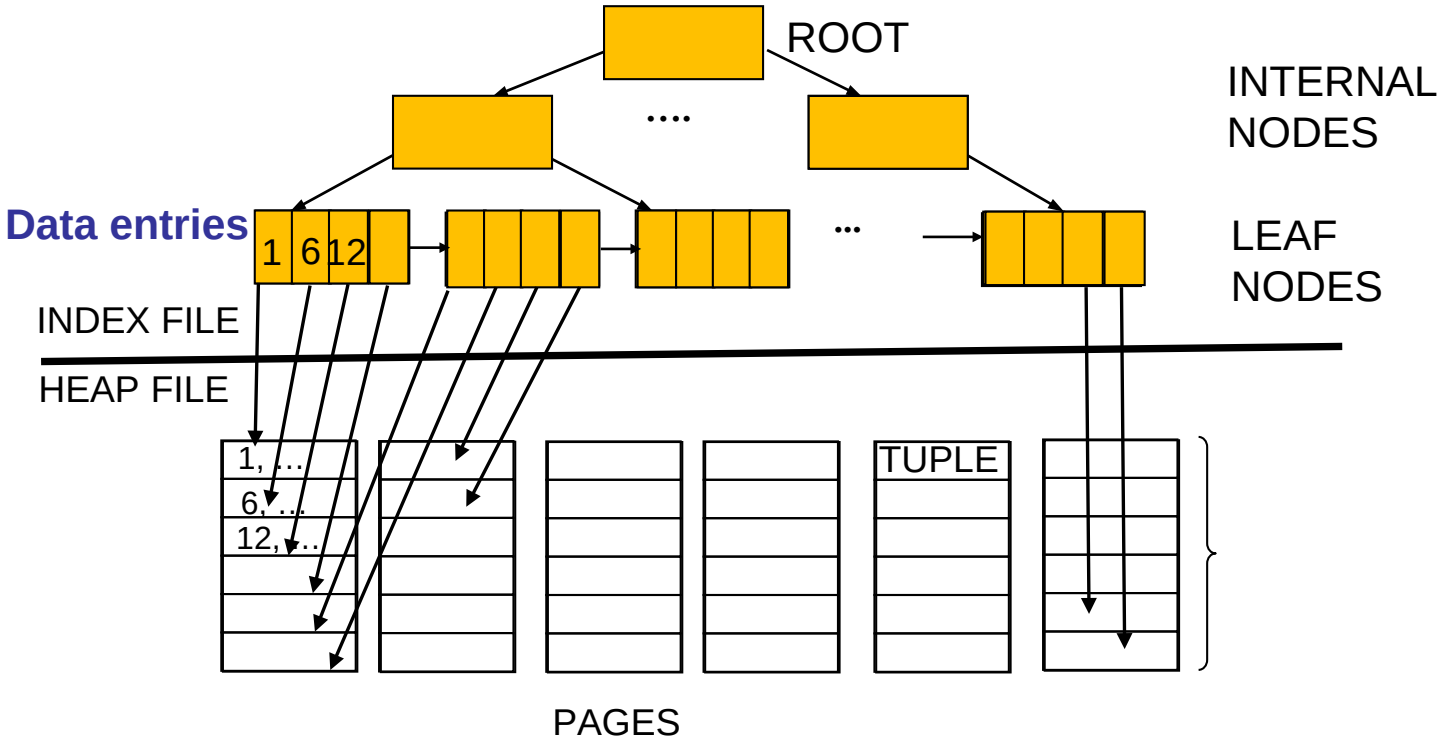
- Classification based on various factors:
  - Clustered vs. Unclustered
  - Primary vs. Secondary
  - Single Key vs. Composite
  - Indexing technique:
    - Tree-based, hash-based, other

Index Classification: Clustering

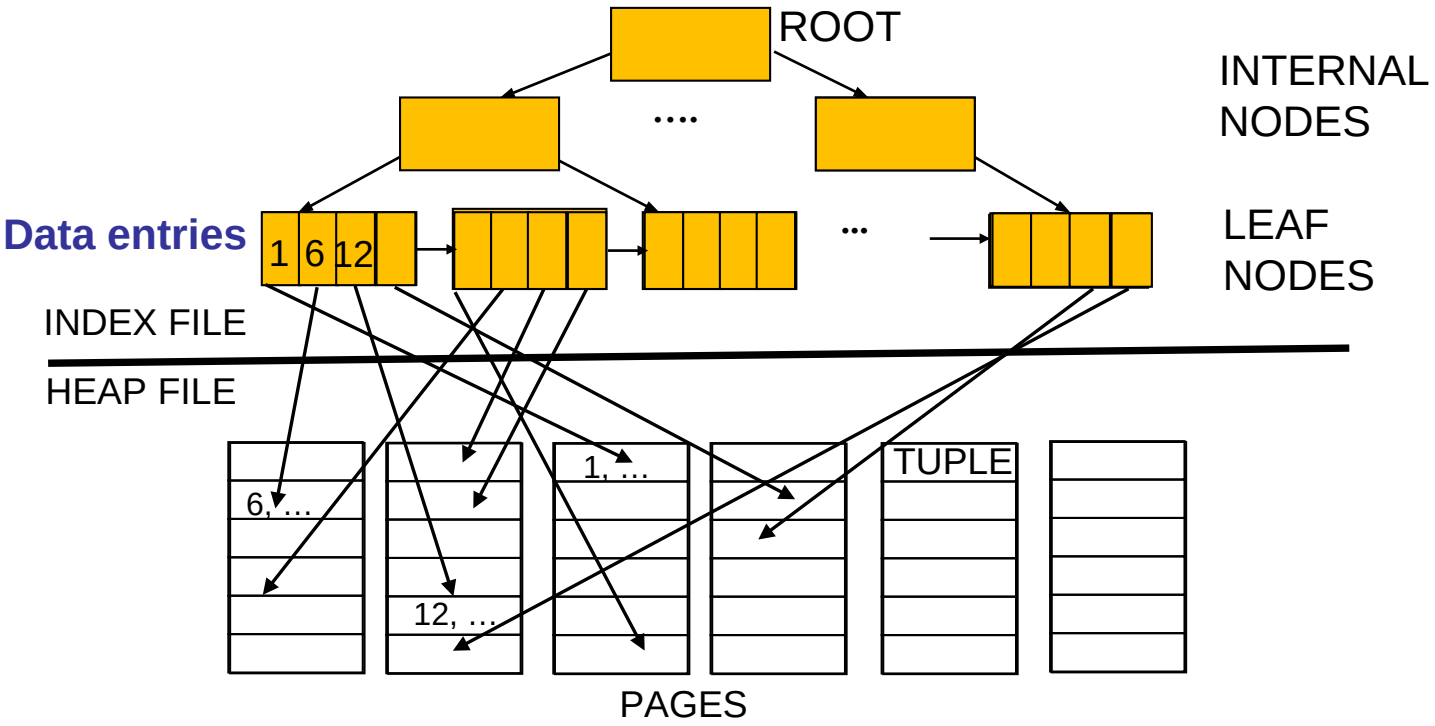
- **Clustered vs. unclustered**: If order of data records is the same as the order of index data entries, then the index is called **clustered index.** Otherwise is **unclustered**.



Zoom in Clustered Index



Zoom in Unclustered Index

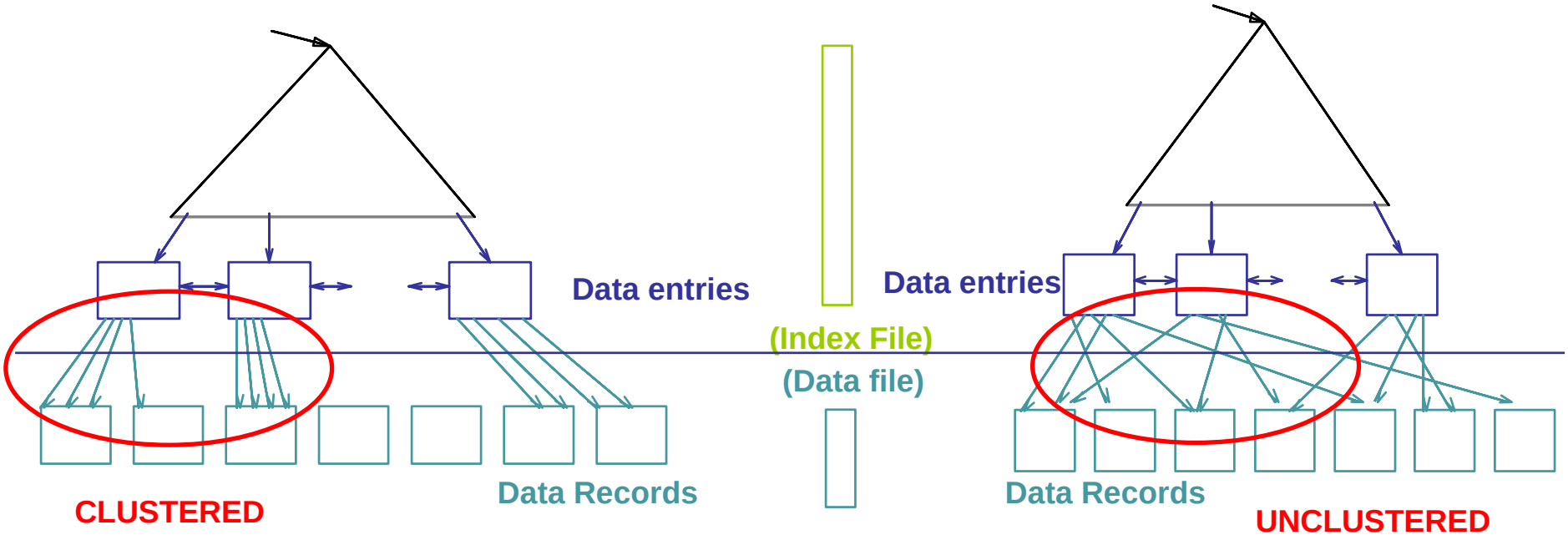


Clustering properties

- A data file can have a clustered index on at most **one search key combination** (i.e. we cannot have multiple clustered indexes over a single table)
- Cost of retrieving data records through index varies *greatly* based on whether index is clustered (cheaper for clustered)
- Clustered indexes are more **expensive to maintain** (require file reorganization), but are really efficient for **range search**

Clustered vs. Unclustered Index: Cost

- (Approximated) cost of retrieving records found in range scan:
  1. Clustered: cost ≈ # **pages** in data file with matching records
  2. Unclustered: cost ≈ # of matching index **data entries (data records)**



Primary vs. Secondary Index

- **Primary** index includes the table's **primary key**
- **Secondary** is any other index

- Properties:
  - Primary index **never** contains duplicates
  - Secondary index **may** contain duplicates

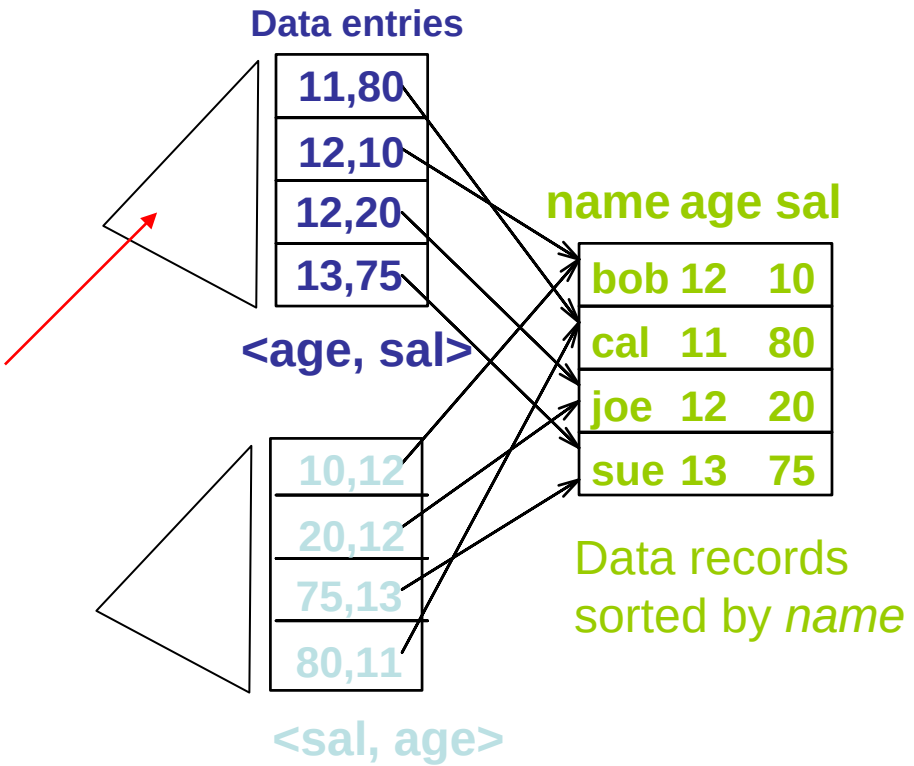
Composite Search Keys

- An index can be built over a combination of search keys
- Data entries in index sorted by search keys

- Examples:

1. Index on <age, sal>
2. Index on <sal, age>

3. Efficient to answer:
   age=12 and sal = 10
   age=12 and sal > 15

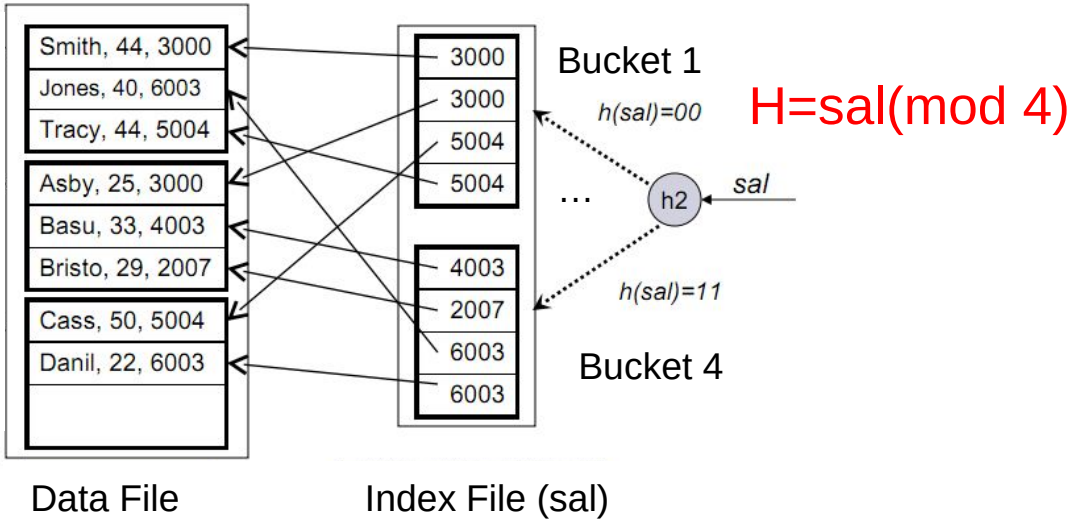


Hash-based index

- Hash-based index:
  - Represents index as a collection of *buckets*. Hash function maps the search key to the corresponding bucket.
    - **h**(*r.search_key*) = bucket in which record *r* belongs
  - Good for **equality** selections

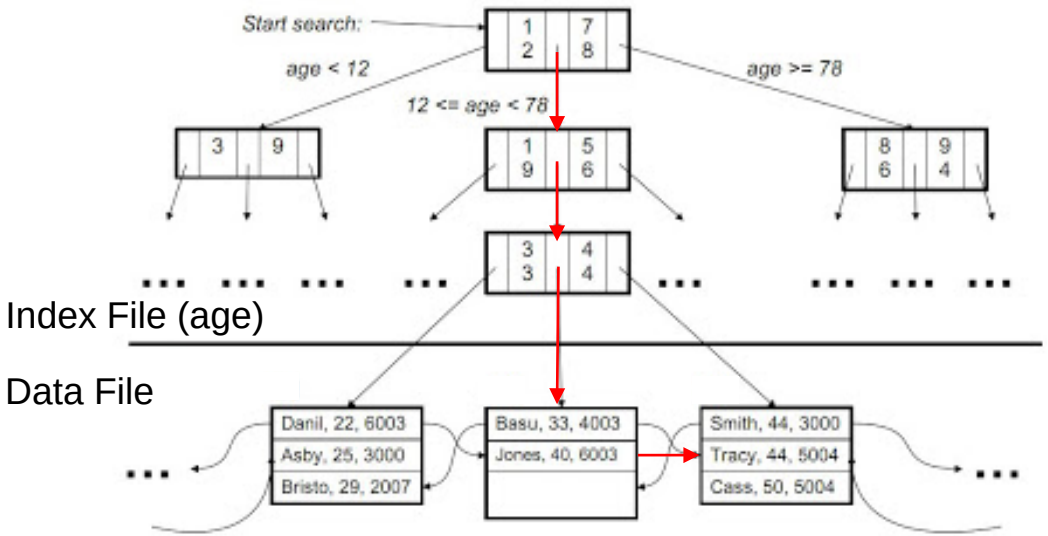
- **Example**: Hash-based index on (sal)

Find Sal = 2007
2007 mod 4 = 3 go to Buck.4



Tree-based index

- Tree-based index:
  - Underlying data structure is a binary (B+) tree. Nodes contain pointers to lower levels (search left for lower, right for higher). Leaves contain data entries sorted by search key values.
  - Good for **range** selections
  - So far we have shown those
- **Example**: Tree-based index on (age)



Find age > 39

Summary

- Many alternative file organizations exist, each appropriate in some situation
- If selection queries are frequent, sorting the file or building an *index* is important

- Index is an additional data structure (i.e. file) introduced to quickly find entries with given key values
  - Hash-based indexes only good for equality search
  - Sorted files and tree-based indexes best for range search; also good for equality search
  - Files rarely kept sorted in practice (because of the cost of maintaining them); B+ tree index is better

What's examinable

- Describe alternative file organizations
- What is an index, when do we use them
- Index classification

Next Lecture

- Query processing part 1
  - Selection and projection (execution, costs)
  - Let’s demystify how DBMS perform work