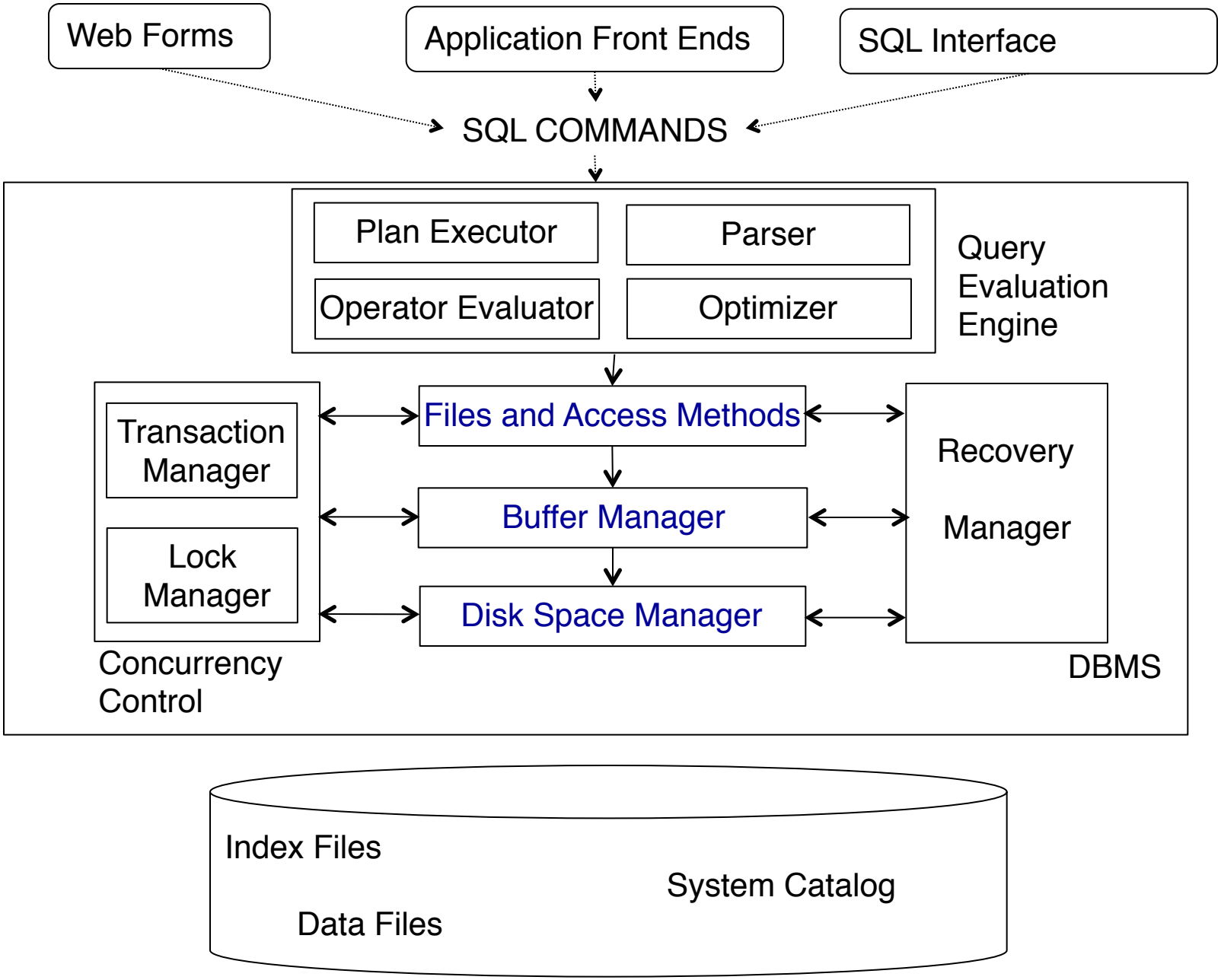
Storage and File Structure

Chapter 9 in Ramakrishnan&Gehrke book

Questions

- What are the different types of memory in a computer system?
- What are the physical characteristics of disks and tapes, and how do they affect the design of database systems?
- How does a DBMS keep track of space on disks? How does a DBMS access and modify data on disks? What is the significance of pages as a unit of storage and transfer?
- How does a DBMS create and maintain files of records? How are records arranged on pages, and how are pages organized within a file?

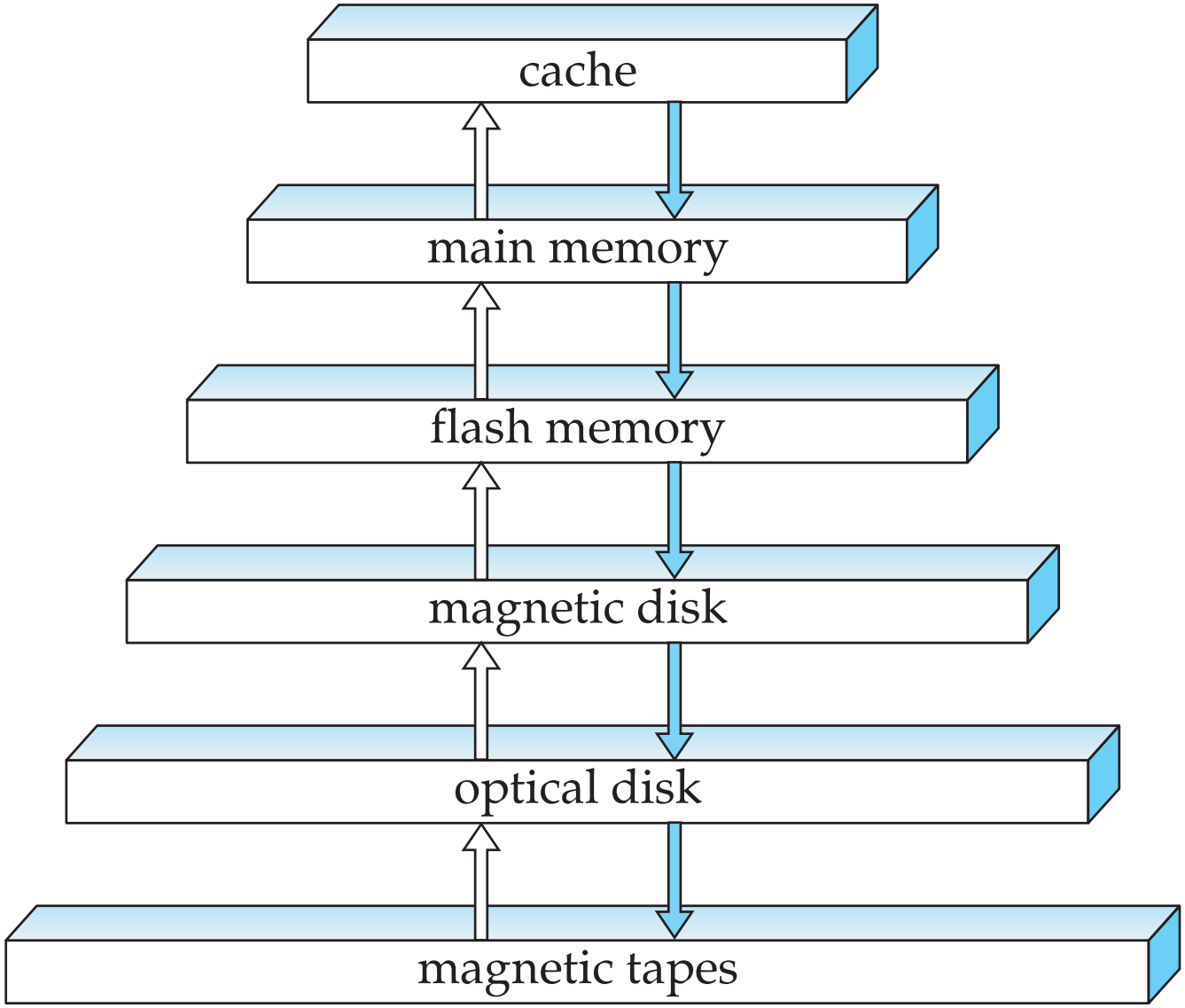


Architecture of a DBMS

Storage and File Structure

- **Physical Storage Media**
  - **Magnetic Disks**
- Disk Space Manager
- Buffer Manager
- File Organization
- Data-Dictionary Storage

Storage Hierarchy



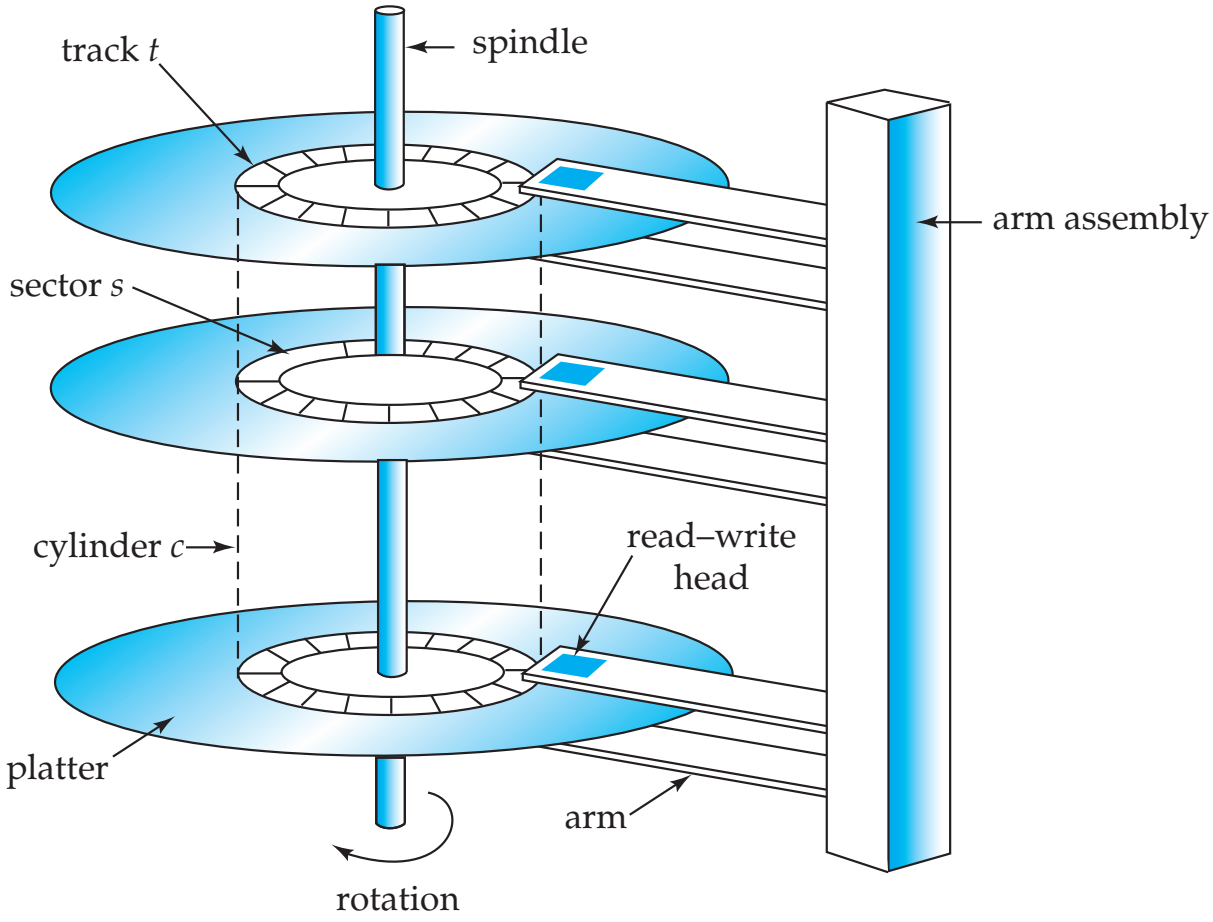
Storage Hierarchy (Cont.)

- **primary storage:** Fastest media but volatile (cache, main memory).
- **secondary storage:** next level in hierarchy, non-volatile, moderately fast access time
  - also called online storage
  - E.g., flash memory, magnetic disks
- **tertiary storage:** lowest level in hierarchy, non-volatile, slow access time
  - also called offline storage
  - E.g., magnetic tape, optical storage

Classification of Physical Storage Media

- Cost per unit of data storage
- Speed with which data can be accessed
- Reliability
  - data loss on power failure or system crash
    - volatile storage: loses contents when power is switched off
    - nonvolatile storage: contents persist even when power is switched off
  - physical failure of the storage device

Magnetic Hard Disk Mechanism



NOTE: Diagram is schematic, and simplifies the structure of actual disk drives

Magnetic Disks

- Magnetic disks support **direct access** to a desired location.
- Data is stored on disk in units called **disk blocks**, which is the unit of reading or writing.
- Surface of platter divided into circular **tracks**
  - Over 50K-100K tracks per platter on typical hard disks
- Each track is divided into **sectors**
  - A sector is the smallest unit of data that can be read or written.
  - Sector size typically 512 bytes
  - Typical **sectors per track**: 500 (on inner tracks) to 1000 (on outer tracks)
- **Platter** may have one or two surfaces
- **Cylinder**: set of all tracks with the same diameter. Cylinder *i* consists of i^{th} track of all the platters

Magnetic Disks

- An array of read-write **disk heads**
  - One head per platter, mounted on a common arm.
  - Current systems typically allow at most one disk head to read or write at any one time
- A disk **controller**: implements commands to read or write a sector by (a) moving the arm assembly and transferring data to ad from the disk surfaces
  - Calculate a **checksum**: for error checking
- **Disk-arm-scheduling** algorithms order pending accesses to tracks so that disk arm movement is minimized
  - **elevator algorithm**: move disk arm in one direction (from outer to inner tracks or vice versa), processing next request in that direction, till no more requests in that direction, then reverse direction and repeat

Magnetic disks (cont.)

- **Mechanical characteristics**
  - Rotation speed (5400-7200RPM) (as of 2013/12)
  - Number of platters (1-30)
  - Number of tracks (<=10000)
  - Number of bytes/track (10^5)
- **Disk block**: the size of a disk block can be set when the disk is initialized as a multiple of the sector size. typically: 4K, or 8K, or 16K

Performance Measures of Disks

- **Access time** = seek time + rotational latency + transfer time
  - Time between when a command is issued and when data is in memory
- **Seek time** – time for move the disk heads to reach the desired track
- **Rotational latency** – time it takes for the sector to be accessed to appear under the head.
- **Data-transfer time**: time to actually read or write the data in the block once the head is positions
  - Data-transfer rate – the rate at which data can be retrieved from or stored to the disk.

Implications of Disk Structure to DBMS

- Data must be **in memory** for the DBMS to operate on it
- The **unit** for data transfer between disk and main memory is a **block**; reading or writing a disk block is called an I/O operation
- If two records are frequently used together, we should place them **close together**

Arranging Pages on Disk

- **A disk is organized into blocks** (a.k.a. pages)
  - blocks on same track, followed by
  - blocks on same cylinder followed by
  - blocks on adjacent cylinder
- A file should (ideally) consist of **sequential blocks** on disk, to minimize seek and rotational delay.
- For a sequential scan, pre-fetching several pages at a time is a big win!

Optimization of Disk Block Access

- **Nonvolatile write buffers** speed up disk writes by writing blocks to a non-volatile RAM buffer immediately
  - Non-volatile RAM: battery backed up RAM or flash memory
- **Log disk** – a disk devoted to writing a sequential log of block updates
  - Used exactly like nonvolatile RAM
    - Write to log disk is very fast since no seeks are required
    - No need for special hardware (NV-RAM)
- File systems typically reorder writes to disk to improve performance
  - **Journaling file systems** write data in safe order to NV-RAM or log disk

- RAID (S.S.)

Storage and File Structure

- Physical Storage Media
  - Magnetic Disks
- **Disk Space Manager**
- Buffer Manager
- File Organization
- Data-Dictionary Storage

Managing Free Blocks

- **Disk space manager**, manages space on disk
- Disk space manager supports the concept of a **page** as a unit of data
- The size of a page is chosen to be the size of a disk block
- Useful to allocate a sequence of pages as a contiguous sequence of blocks to hold data frequently accessed.

Managing Free Blocks

- Disk space manager **keeps track** of
  - Which blocks are in use
  - Which blocks are on which disk blocks
- **Linked list of free blocks**
  - When a block is de-allocated, it is added to the free list
  - A pointer to the first block on the free block list is stored in a known location on disk.
- **Bitmap**
  - One bit for each disk block
  - Allow very fast identification and allocation of contiguous areas on disk.

Disk space manager

- Use OS files
  - The entire DB resides in one or more OS files for which a number of blocks are allocated (by the OS) and initialized
- Do not rely on OS file system
  - Self-contained code to support several OS platforms
  - A file may be too small (e.g., 32 bit system, the largest file size is 4GB)
  - OS files cannot span disk devices

Storage and File Structure

- Physical Storage Media
  - Magnetic Disks
- Disk Space Manager
- **Buffer Manager**
- File Organization
- Data-Dictionary Storage

Buffer manager

- **Motivation**
  - DB contains 1M pages
  - Main memory can only hold 1000 pages
- **Buffer manager**: bring pages from disk to main memory as needed.
- Buffer manager manages available main memory by partitioning it to a collection of pages, **buffer pool**.
- The main memory pages are called **frames**.

Buffer Manager

- Maintains book-keeping information and two variables for each frame in the pool
  - Variable ***pin_count***: the number of times that a page currently in a given frame has been requested but not released
  - Variable ***dirty***: indicates whether the page has been modified since it was brought into the buffer pool from the disk
- **Pinning**: incrementing *pin_count*
- **Unpinning**: the requestor releases a page and its *pin_count* is decremented

Buffer Manager

- **When a page is requested**, the bugger manager does:
  - If the requested page is **in the buffer pool**, increments the *pin_count* of that frame.
  - If the request page is **not in the buffer pool**, the buffer manager
    - Chooses a frame for replacement using the replacement policy and increments its *pin_count*
    - If the *dirty* bit for the replacement frame is on, write the page it contains to disk
    - Reads the requested page into the replacement frame
  - Returns the **main memory address** of the frame containing the requested page to the requestor.

Buffer Manager

- The buffer manager will not read another page into a frame until its *pin_count* becomes 0
  - When a frame is needed, a frame with *pin_count* 0 is chosen for replacement; If there are many such frames, buffer *replacement policies* are applied.
  - **Frame whose *dirty* bit is not set**, newly requested page directly replace such frames
  - **Frames whose *dirty* bit is set**, propagate modification to the disk (recovery manager)
- If **no page in the buffer pool has *pin_count* 0 and a page that is not in the pool is requested**, the buffer manager must wait until some page is released before responding to the page request
- A buffer manager assumes that an appropriate **lock has been obtained** before a page is requested (Transaction manager)

Buffer Replacement Policies

- Choose **an unpinned page for replacement**
- **LRU (least recently used)**: use a queue of pointers to unpinned pages (enter to queue tail and chosen from queue head)
- LRU **can be a bad strategy** for certain access patterns involving repeated scans of data. E.g.,
  - buffer pool has 10 pages
  - *r* has 1 page, *s* has 11 pages
  - Join *r* and *s* by a nested loops
    for each tuple *tr* of *r* do
      for each tuple *ts* of *s* do
        if the tuples *tr* and *ts* match …
  - for each *tr*, read 11 pages

Storage and File Structure

- Physical Storage Media
  - Magnetic Disks
- Disk Space Manager
- Buffer Manager
- **File Organization**
- Data-Dictionary Storage

Issues

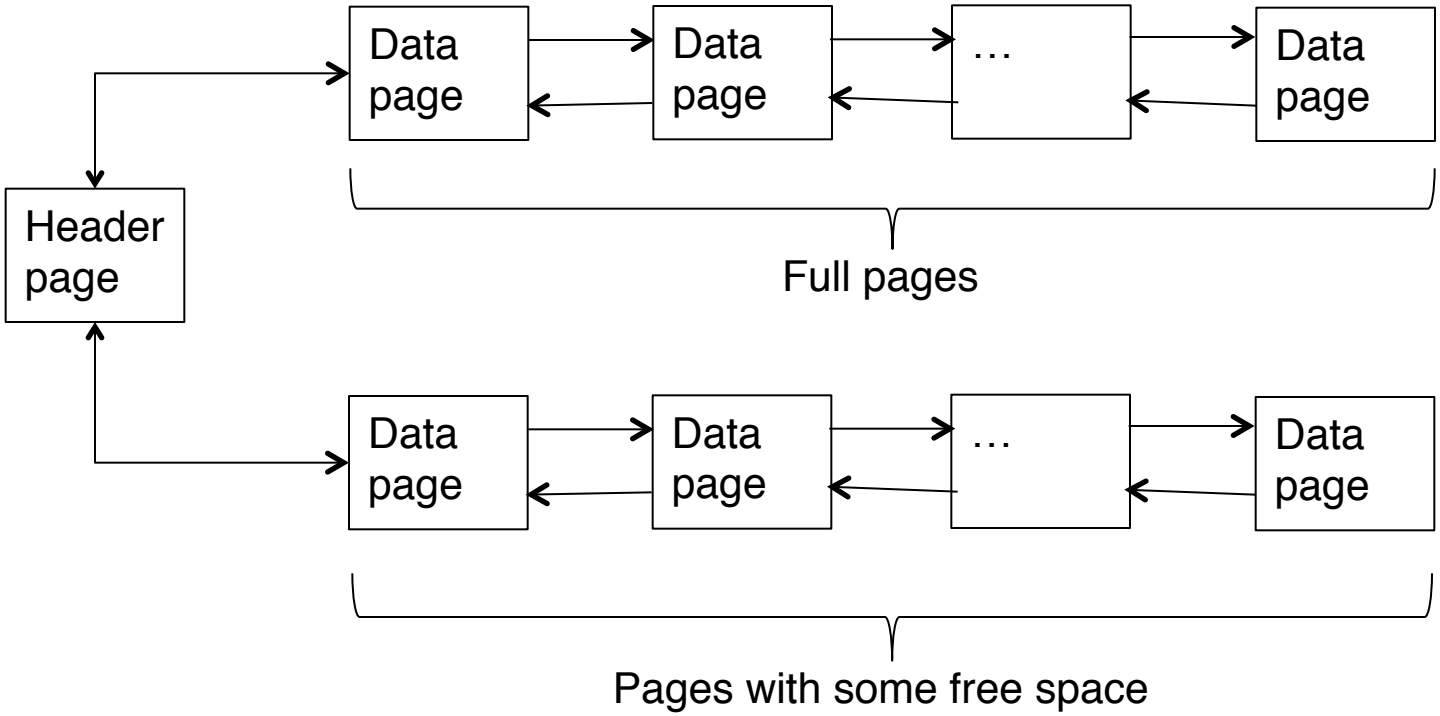
- **File** format: how are the pages organized in a file
- **Page** formats: how are records organized in a page
- **Record** formats: how are attributes organized in a record

File Format - Heap Files

- The data in the pages of a heap file is **not ordered**.
- The database is stored as a collection of **files**. Each file consists of one or more **pages**.
- **Every page** in a file is of the same size.
- **Every record** in the file has a unique rid.
- Supported operations
  - File creation and destroy
  - Record insertion, deletion, and retrieval
- Information to keep
  - Keep track of pages in each heap file
  - Keep track of pages that contain free space

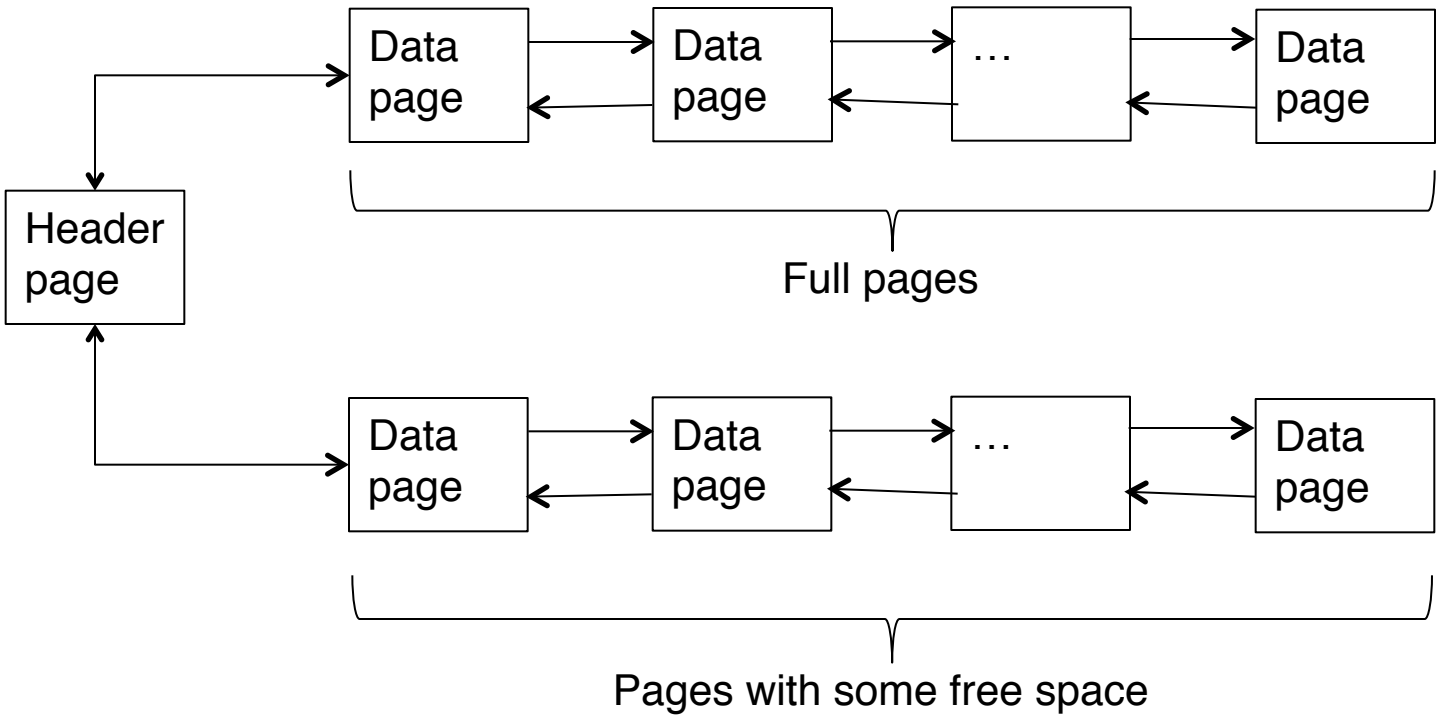
Heap files – linked list of pages

- Heap file: **doubly linked list of pages**
- DBMS remembers the first page (**header page**) in a table containing <heap_file_name, header_page_address>
- Track **pages that are not full** (i.e., have some free space)
- Track free space within a page (later)

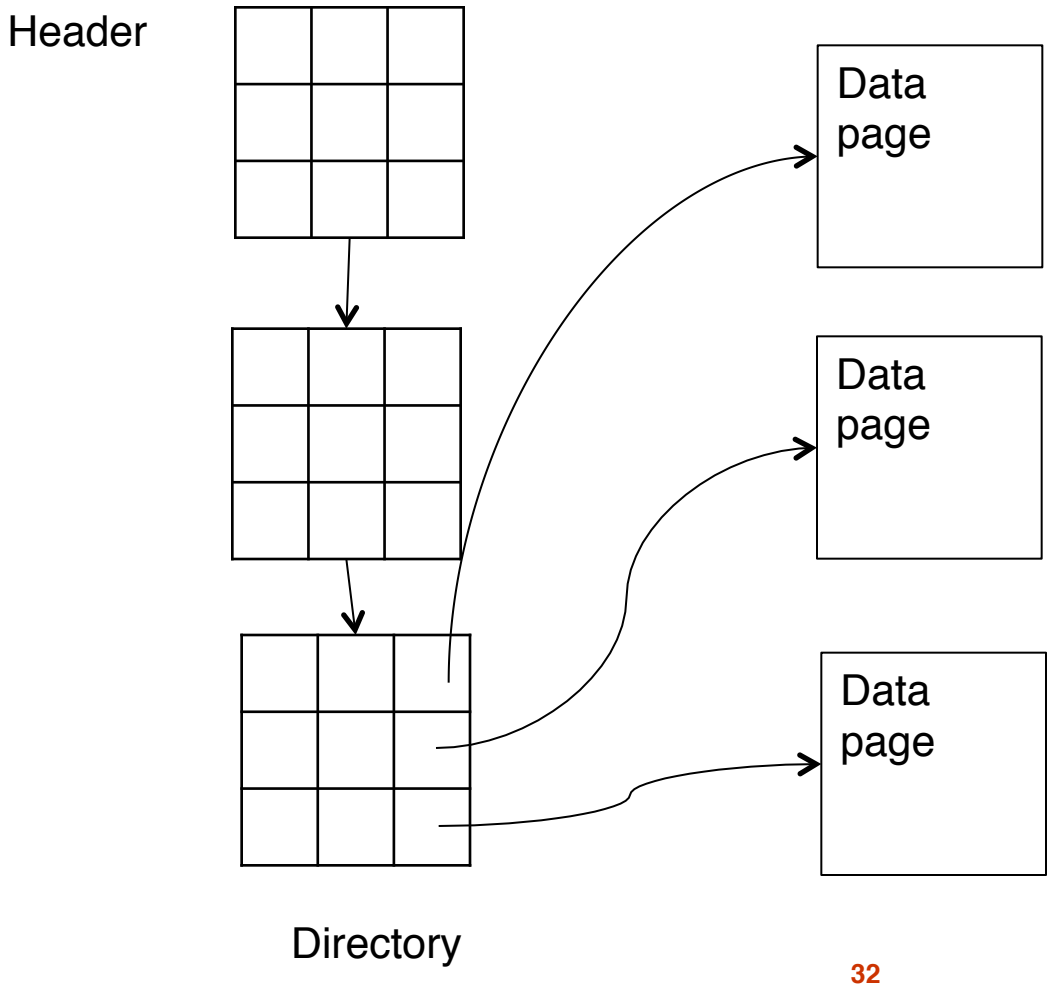


Heap files – linked list of pages

- **Disadvantage**
  - For records of variable length, virtually every page has some space



Heap files – directory of pages



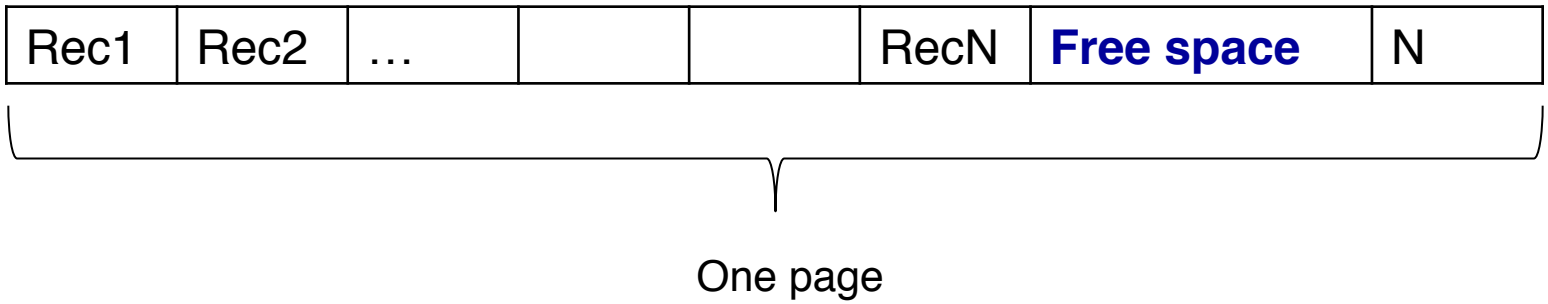
- DBMS remembers the **first directory page** of each heap file
- Each **directory entry**
  - a **bit** (indicating whether the corresponding page has any free space)
  - OR **a count** (indicating the amount of free space on the page)

Page Formats

- How a **collection of records are arranged on a page**?
- A page is a collection of slots, which are for records
- Each record is identified by
  - **RID = <page_id, slot_number>**
  - There are alternative ways to maintain record ids
- Issues to consider
  - 1 page = fixed size (e.g. 8KB)
  - Records: Fixed length, Variable length

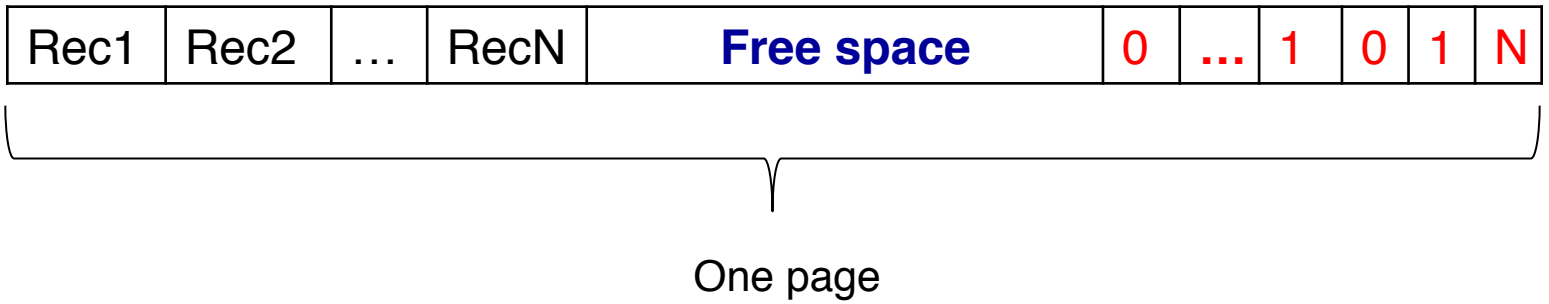
Page formats – fixed-length records

- Fixed-length records: **packed** representation
- **Delete a record**: move the last record on the page to the vacated slot
- **Locate the *i*th record** by a simple offset calculation
- **Disadvantage**: there are external references to the record that is moved



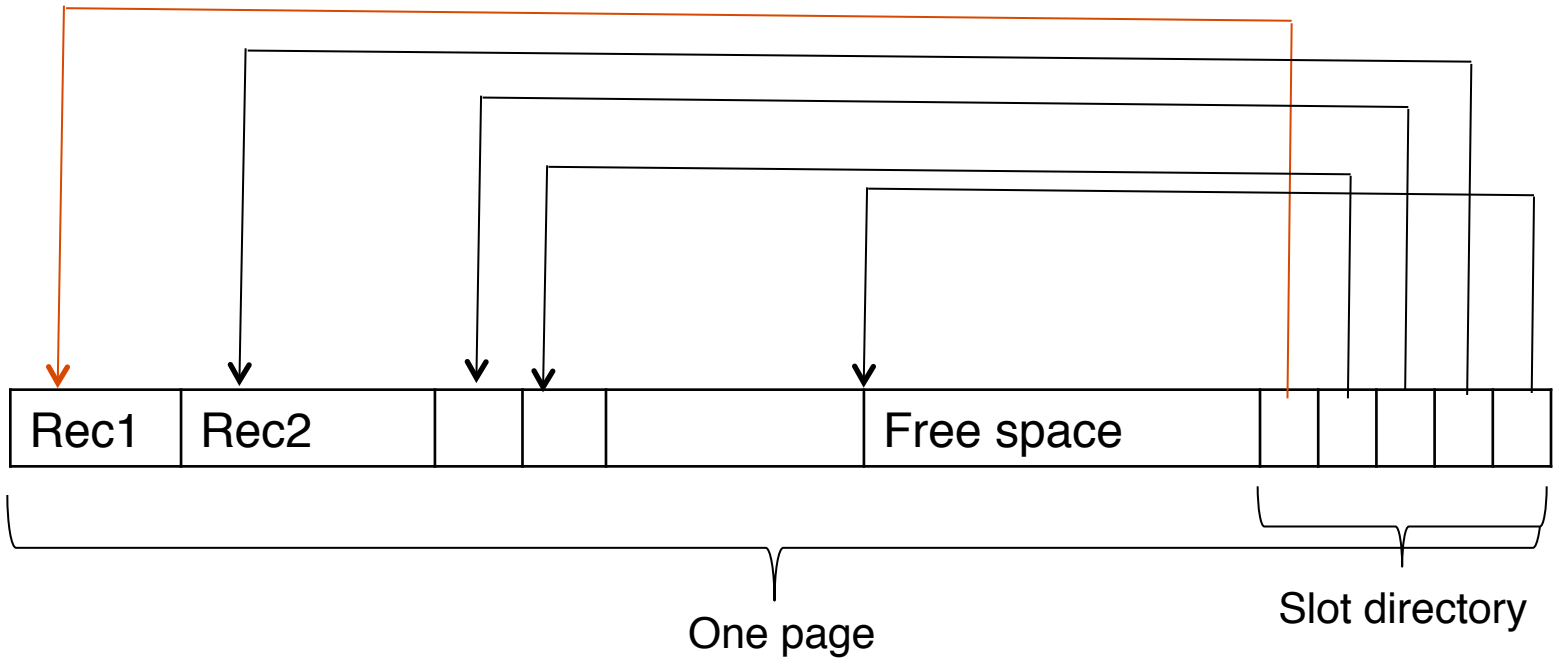
Page formats – fixed-length records

- Fixed-length records: **unpacked** representation
- Handle deletions using **an array of bitsM**
- **Delete a record**: set its bit off
- **Locating a record**: scanning the bit array
- **N**: number of slots



Page formats – variable-length records

- Maintain a **directory of slots** for each page
  - **Each slot** <record offset, record length>
  - **Delete a record**: set its record offset to be -1, and the record can be actually moved around
  - **Insert a record**: search for an empty slot



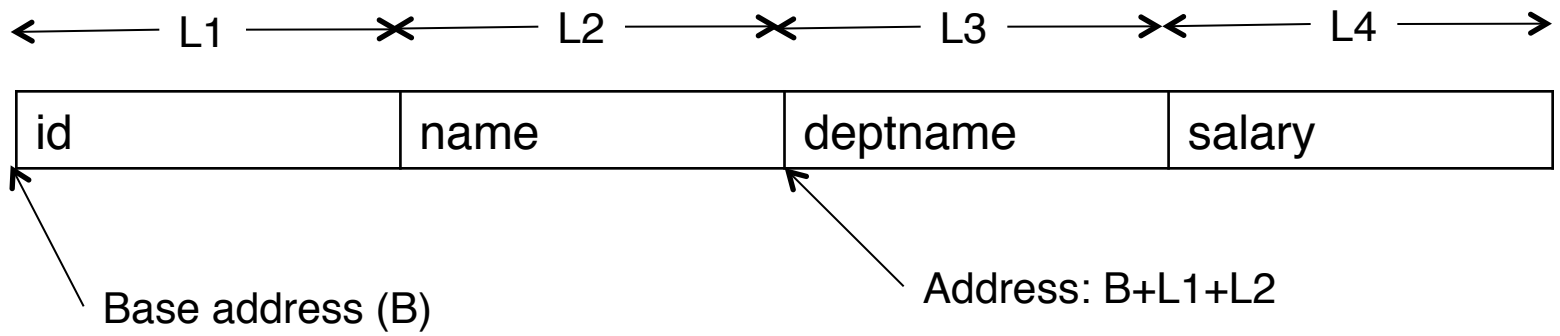
Page formats – variable-length records

- **Variations**
  - Can be used for fixed-length records if the fixed-length records need to be moved frequently
  - Slots only maintain the record offsets (without the length) and the length is stored with the record (This works for both fixed-length and variable-length records)

Record Formats: fixed-length

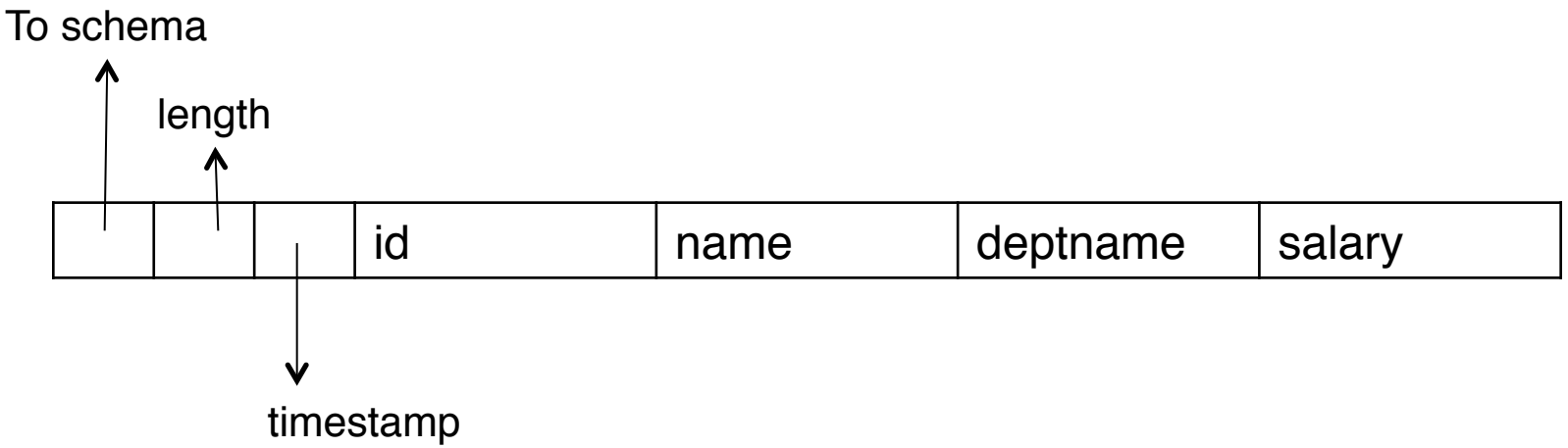
- **How to organize fields within a record?**
- Information about field types same for all records in a file; stored in **system catalogs**.

- The number of fields is fixed and each field has a fixed length
- **Finding *i*'th field:** requires scan of record.



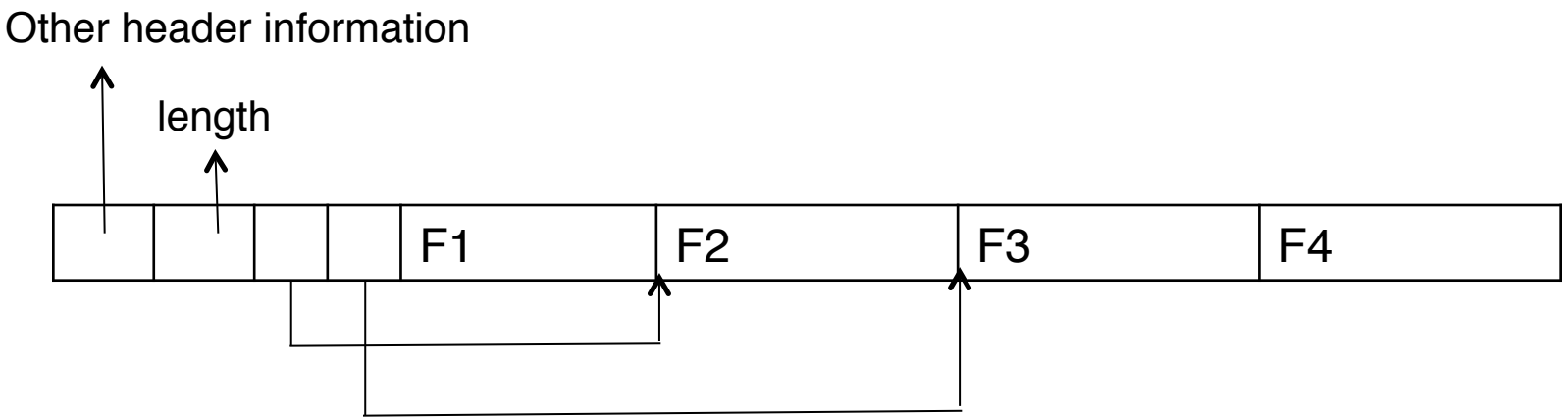
Record Header

- Need the **header** because:
  - The schema may change for a while new+old may coexist
  - Records from different relations may coexist



Variable Length Record

- **Array of integer offsets** at the beginning of a record
  - The *i*th integer is the starting address of the *i*th field
  - ending offset?
  - Place the **fixed fields** first: F1
  - Then the variable length fields: F2, F3, F4.
- **Null values:** pointer to the end of the field = pointer to the beginning of the field



BLOB

- Binary large objects
- Supported by modern database systems
  - E.g. images, sounds, etc.
  - Storage: attempt to cluster blocks together
- CLOB = character large object
  - Supports only restricted operations

Page format (used by Postgres)

- Used by Postgres tables and indexes
- Every table/index is stored as an array of pages of a fixed size (8KB)
  - Index: the first page special use
- Each page
  - Header (PageHeaderData, 24 bytes: WAL entry, … free space pointer…)
  - ItemIdData: pairs of (offset, length), each pair requires 4 bytes
  - Free Space
  - Items (table rows or index entries)

- http://www.postgresql.org/docs/9.2/static/storage-page-layout.html

Organization of Records in Files

- **Heap** – (random order) files: Suitable when typical access is a file scan retrieving all records.
- **Sorted File (Sequential)** – Best if records must be retrieved in some order, or only a `range' of records is needed.
- **Hashing** – a hash function computed on some attribute of each record; the result specifies in which block of the file the record should be placed
- Records of each relation may be stored in a separate file. In a **multi-table clustering file organization** records of several different relations can be stored in the same file
  - Motivation: store related records on the same block to minimize I/O

Multitable Clustering File Organization

Store several relations in one file using a **multitable clustering** file organization.

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Physics	Watson	70000

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000

Multitable Clustering File Organization (cont.)

Multitable clustering organization of *department* and *instructor:*

Comp. Sci.	Taylor	100000
45564	Katz	75000
10101	Srinivasan	65000
83821	Brandt	92000
Physics	Watson	70000
33456	Gold	87000

- good for queries involving *department* $\bowtie$ *instructor*
- bad for queries involving only instructor
- results in variable size records
- Can add pointer chains to link records of a particular relation

Storage and File Structure

- Physical Storage Media
  - Magnetic Disks
- Disk Space Manager
- Buffer Manager
- File Organization
- **Data-Dictionary Storage**

Data Dictionary Storage

Data dictionary (also called **system catalog**) stores **metadata**; that is, data about data, such as

- Information about relations
  - names of relations
  - names and types of attributes of each relation
  - names and definitions of views
  - integrity constraints
- User and accounting information, including passwords
- Statistical and descriptive data
  - number of tuples in each relation
- Physical file organization information
  - How relation is stored (sequential/hash/…)
  - Physical location of relation
- Information about indices

Data Dictionary Storage (Cont.)

- **Catalog structure**
  - Relational representation on disk
  - specialized data structures designed for efficient access, in memory
- A possible catalog representation:

  Relation_metadata = (*relation_name*, *number_of_attributes*, *storage_organization*, *location*)

  Attribute_metadata = (*attribute_name*, *relation_name*, *domain_type*, *position*, *length*)

  User_metadata = (*user_name*, *encrypted_password*, *group*)

  Index_metadata = (*index_name*, *relation_name*, *index_type*, *index_attributes*)

  View_metadata = (*view_name*, *definition*)

Data Dictionary Storage (Cont.)

- Mysql: INFORMATION_SCHEMA database
  - http://www.java2s.com/Tutorial/MySQL/0500__Data-Dictionary/TheINFORMATIONSCHEMADatabase.htm
- Postgres: System catalogs
  - http://www.postgresql.org/docs/8.3/static/catalogs.html