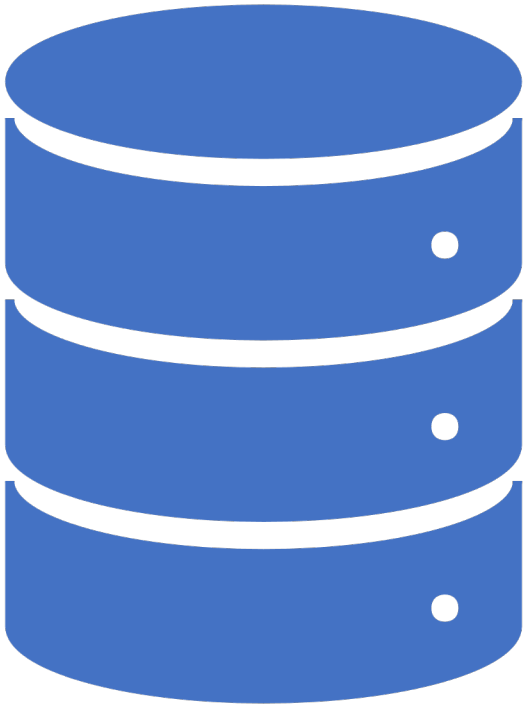
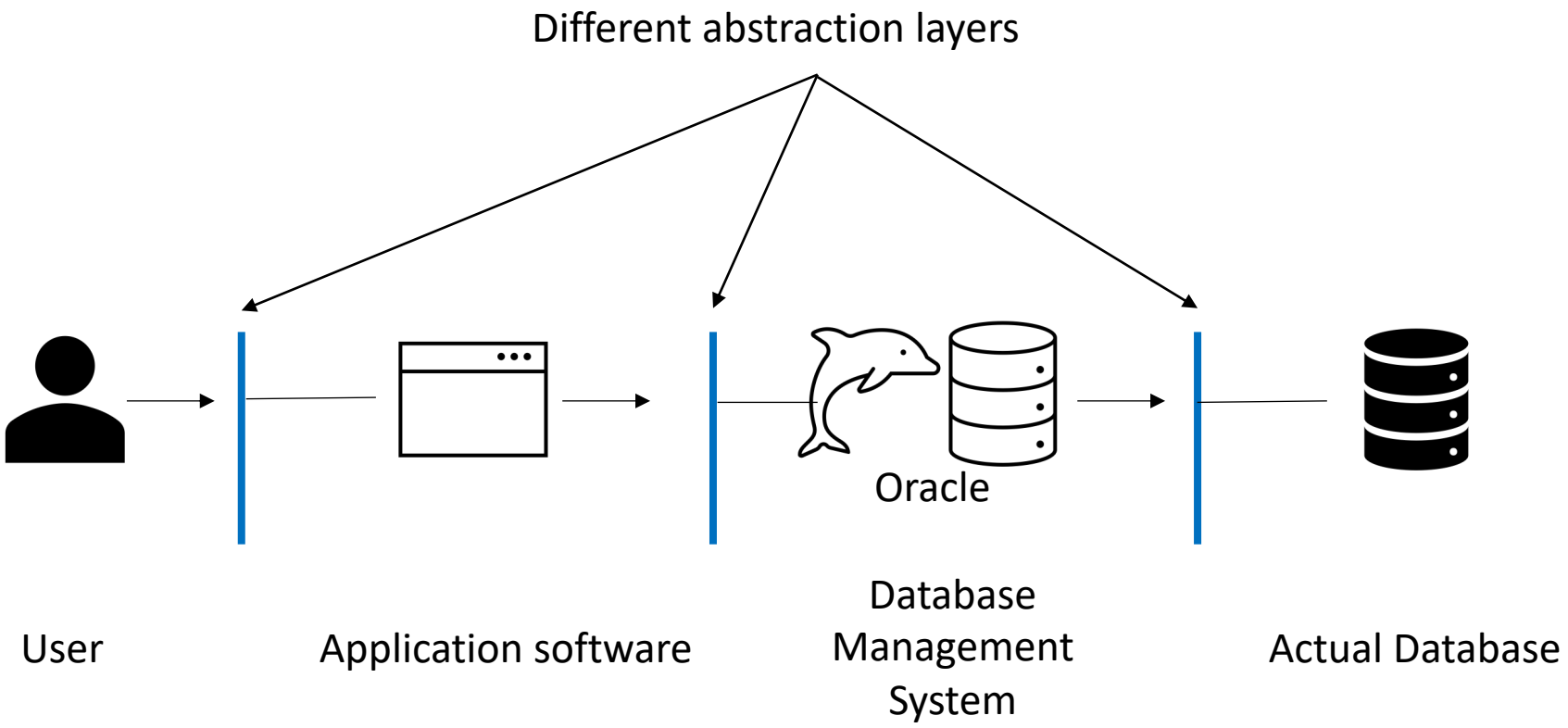


Lecture 8

The Physical Structure of Databases

The Physical Structure of Databases

Abstraction Layer

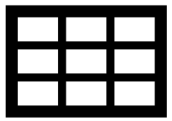


The Physical Structure of Databases

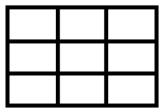
STUDENT
StudentId
StudentName
StudentSurname
DateOfBirth
StudentGroup

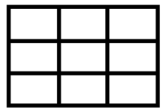
Physical Structure

Faculty database

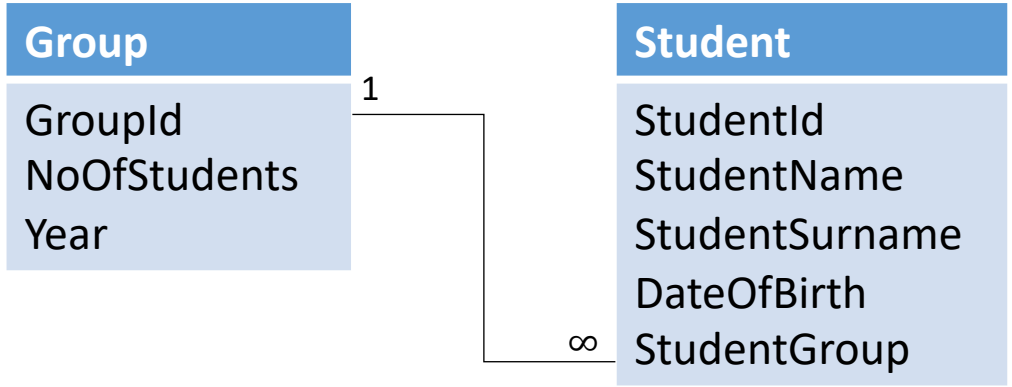


Group table Student table

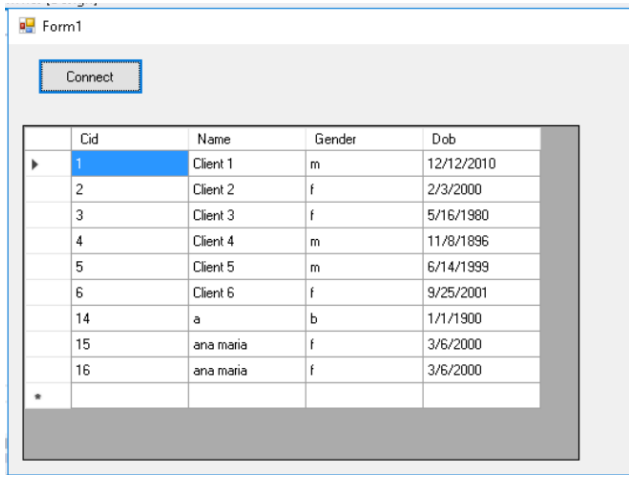




Conceptual Structure



User View



The Physical Structure of Databases

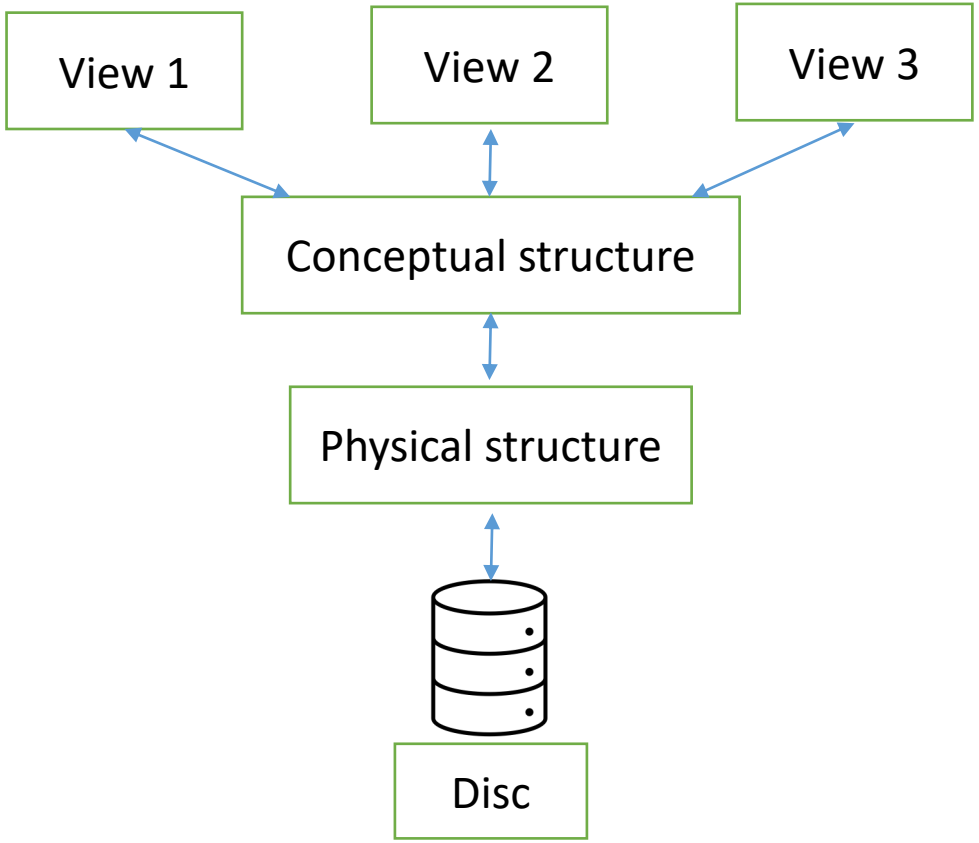
Abstraction Layer

- There are many ***external structures (views)***, only one ***conceptual structure (logical)*** and a ***physical structure (internal)***

- *Views* – how the users see data

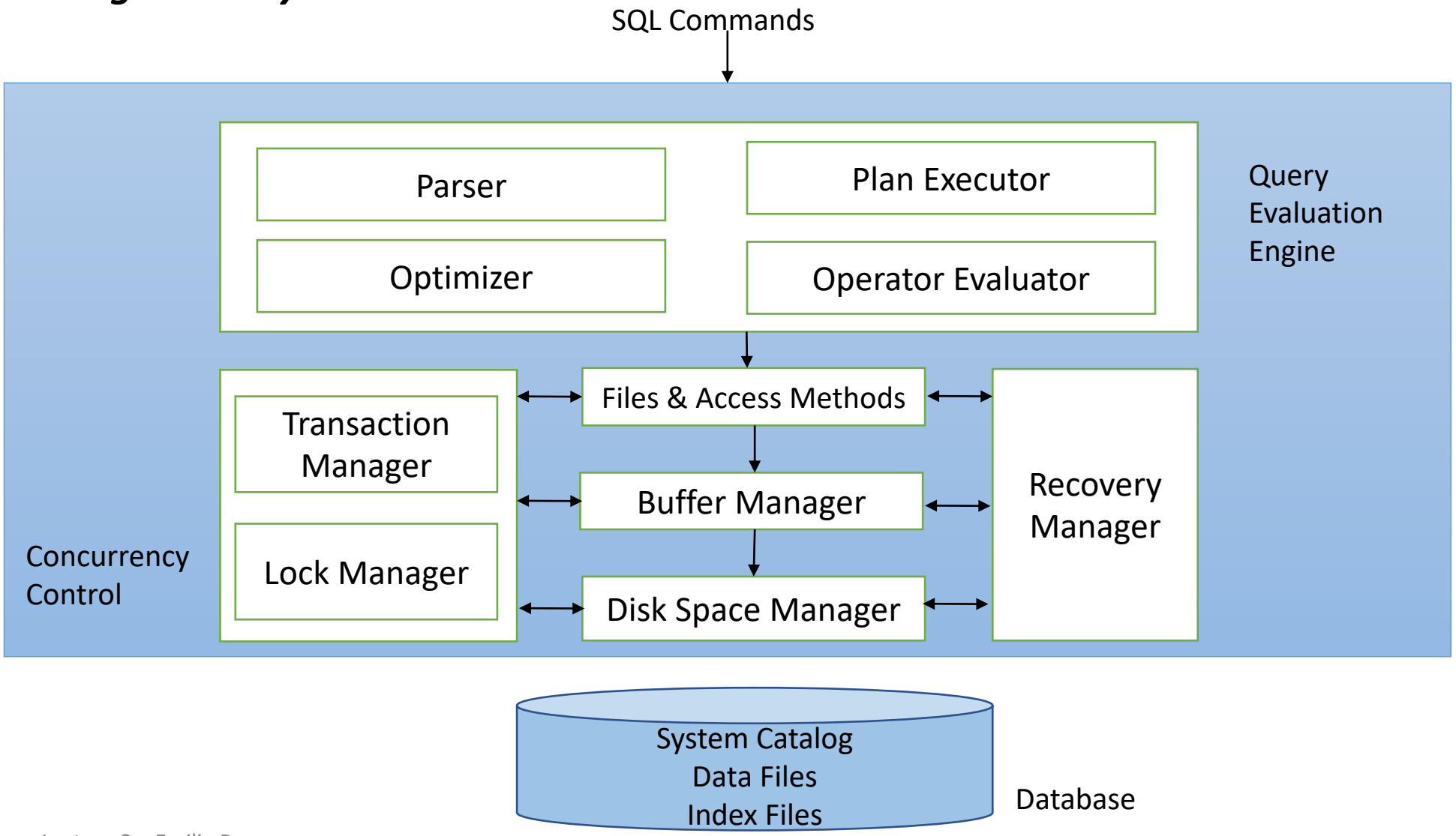
- *Conceptual structure* – logical model composed with attributes, relationships, …

- *Physical structure – data files and indexes*



The Physical Structure of Databases

Database Management Systems Structure



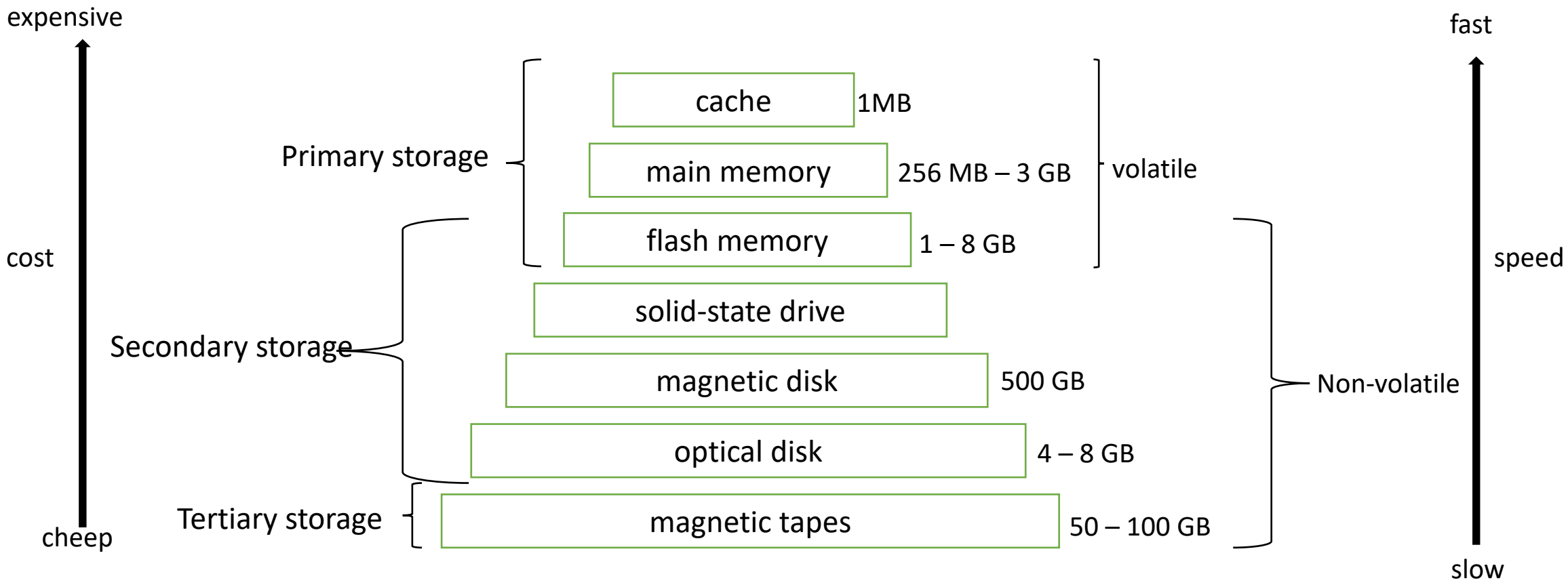
The Physical Structure of Databases

The Physical Structure of the database files

- DBMS store the information on magnetic disks (may cause major implications)
- ***READ*** – data transfer from disk to the internal memory
- ***WRITE*** – data transfer from internal memory to disk
- Both operations are expensive related to *in-memory* operations; should be planned correspondingly
- ***Internal Memory***: cost too much; is volatile (data should be persistent)
- ***Primary storage***: cache, main memory, very fast access to data, volatile, currently used data
- ***Secondary storage***: slower storage devices, non-volatile, disks-sequential, direct access, main database (e.g. magnetic disks)
- ***Tertiary storage***: slower storage devices, non-volatile, tapes (have only sequential access, good for archives, backups, unsuitable for data that is frequently accessed) (e.g. optical disk, tapes) – used to archive the older version of data

The Physical Structure of Databases

Memory Hierarchy



The Physical Structure of Databases

Memory Hierarchy

- disk + tapes – cheaper than the main memory
- a big quantity of data that should be kept when the system is restarted → DBMS should bring the data from disk to the main memory for processing

Gordon Moore Law: *"Integrated circuits are improving in many ways, following an exponential curve that doubles every 18 months"*

- The followed parameters: processors speed, number of the bits from a chip, number of bytes on an hard disk
- The un-followed parameters: the speed to access the data from internal memory, the rotation speed of the disk
- The latency becomes higher: the transfer time between the hierarchical levels of the store devices becomes higher comparing with the calculus time

The Physical Structure of Databases

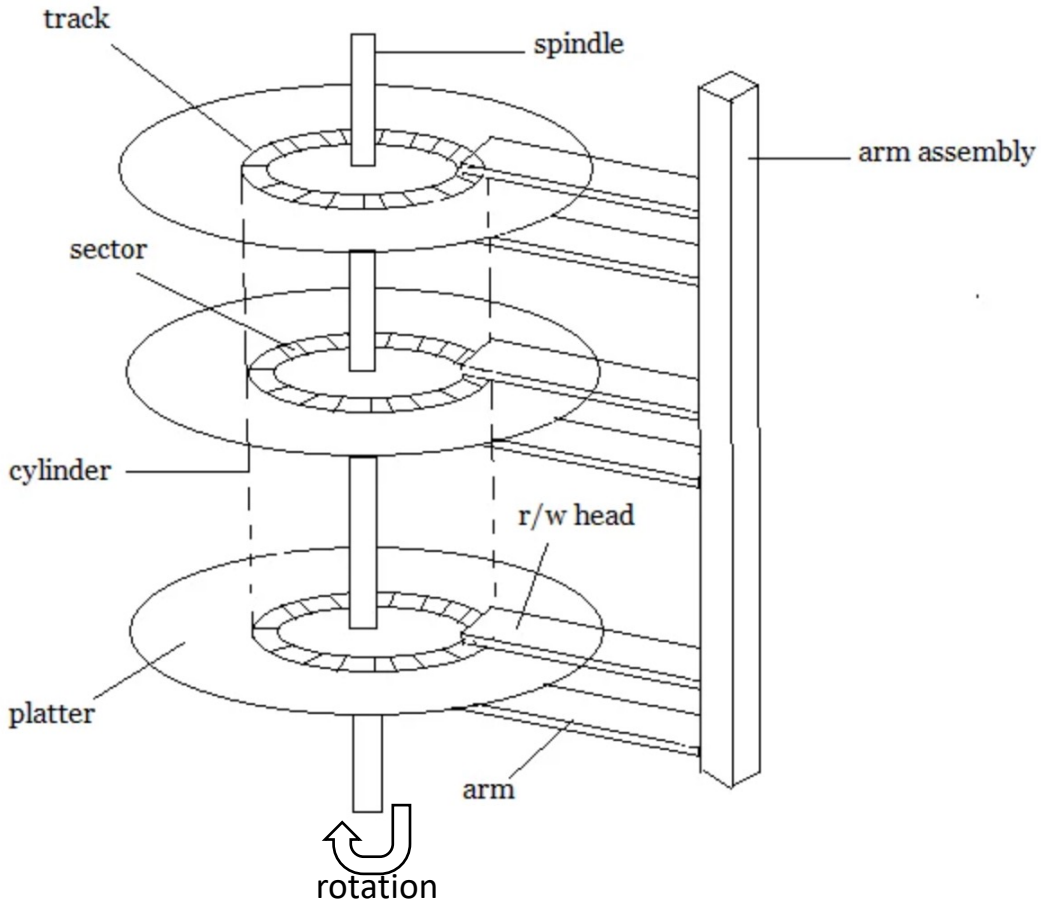
Secondary Storage – Magnetic Disks

- direct access
- data is stored and read in units called *disk blocks (block)* or *pages*
- the transfer time for blocks / pages vary due to their position on the disk
- the relative position of the blocks / pages on the disk has an important impact on the performance of a DBMS
- used in database applications
- in DBMSs applications do not need to know if the data is on the disk or in main memory
- *disk block* – sequence of contiguous bytes
  - unit for data storage and data transfer (read/write data on the disk – input/output operation)
- *tracks* – concentric rings that contain blocks; are recorded on one or more platters
- *sectors* – are arcs on tracks
- *platters* – can be single/double sided (data recorded on one or both surfaces)
- *cylinder* – set all the tracks with the same diameter
- *disk heads* – one per recorded surface
  - systems have one active head; all disk heads are moved as a unit
  - a head has to be on the top of the block to perform the read and write of a block

The Physical Structure of Databases

Secondary Storage – Magnetic Disks

- the platter rotation is 90rps
- *arm assembly* - moves to obtain the position of the *read-write head* on the desired platter
- the tracks that have the same distance to the center of the platters from an imaginary cylinder
- only one read-write head read/write at a specified moment
- a block contains more sectors (are fixed)
- the sector size cannot be modified
- the block size is multiple of the sector size
- DBMS operates on data when it is in memory
- the block is the unit used for data transfer between the disk and the main memory



[https://www.studytonight.com/operating-system/secondary-storage]

The Physical Structure of Databases

Secondary Storage – Magnetic Disks

- The time to access a location can be in the main memory or on the disk
- Access time (read/write) for a disk block:
  - **seek time** – time to move the disk head to the desired track (smaller platted size goes to a decreased seek time)
  - **rotational delay** – time for the block to get under the head
  - **transfer time** – time to read/write the block, once the disk is positioned over it
- s*eek time* (vary between 1 and 20 msec*)* and *rotational delay* (vary between 0 and 10msec) are dominant – the *transfer rate* is approximate 1msec on 4KB per page
- The required time for the operations performed on the databases is dominated by the time taken to transfer blocks between disk and main memory
- reducing the input/output cost, also reduce the *seek time* and the *rotational delay*
- *Goal*: to minimize the access time (data should be carefully placed on disk)
- The records that are usually used together should be closed one to another: same block, track, cylinder, adjacent cylinder
- The sequential manner to access data reduces the *seek time* and the *rotational delay*

The Physical Structure of Databases

Secondary Storage – Magnetic Disks

- e.g. characteristics: storage capacity (GB), platters (single-side, double-side, number), average / max seek time (ms), average rotational delay (ms), number of rotations/minute, data transfer rate (MB/sec)

Arrangement of the pages (that have records) on the disk: *next block* – blocks from the same platter, followed by, blocks from the same cylinder, followed by blocks from adjacent cylinders

- the blocks from a file should be sequential disposed on the disk (*next*) to minimize the *seek delay* and the *rotational delay*
- in case of a *sequential scan*, the page reading in advance (*pre-fetching*) is essential.

Managing Disk Space

- to manage the space on the disk is used *Disk Space Manager* (DSM)
- a page is a unit of data that has the size of a disk block with one input/output operation (as a read/write operation per page)
- the upper layers in the DBMS could treat the data as a collection of pages

The Physical Structure of Databases

Managing Disk Space

- Disk Space Manager knows the correspondence between the pages and the disk blocks
- Disk Space Manager has commands to allocate/deallocate and read/write pages
- Disk Space Manager monitors the disk usage by keeping track of available disk blocks
- The free block can be identified by having:
  - a linked list of the free blocks (on deallocation, the block is added to the list)
  - a bitmap with one bit / block (that indicate if the specified block is used or not) – useful to identify the contiguous available areas on the disk

RAID (Redundant Array of Inexpensive/Independent Disks) - ***Disk Array***

- A combination of magnetic disks that abstract a single disk
- Less expensive because are used multiple small and cheep disks instead of a disk with a higher capacity
- Goal: to increase the performance and the reliability
- Techniques: *data striping* (distribute the data on multiple disks) and *mirroring* (automatically store a copy of the data in other disks – *redundancy*; useful to recover data when are problems with the disks)

The Physical Structure of Databases

RAID

- Level 0: no redundancy
- Level 1: mirrored disks
  - Each disk has a check disk (mirror)
  - Parallel reads; a writing involves 2 disks
  - Maximum transfer rate = transfer rate of a disk
- Level 0+1: interlaced and mirrored
- Level 3: Parity bit interleaved
  - Striping unit: one bit, one disk to check
  - each read and write involves all the disks
- Level 4: Parity block
  - Striping unit: one block, one disk to check
  - Parallel readings to small dimension requests
  - The writings involves the modified block and the check disk
- Level 5: Distributed parity block
  - Equivalent with Level 4, except the fact that the parity blocks are distributed on all the disks

The Physical Structure of Databases

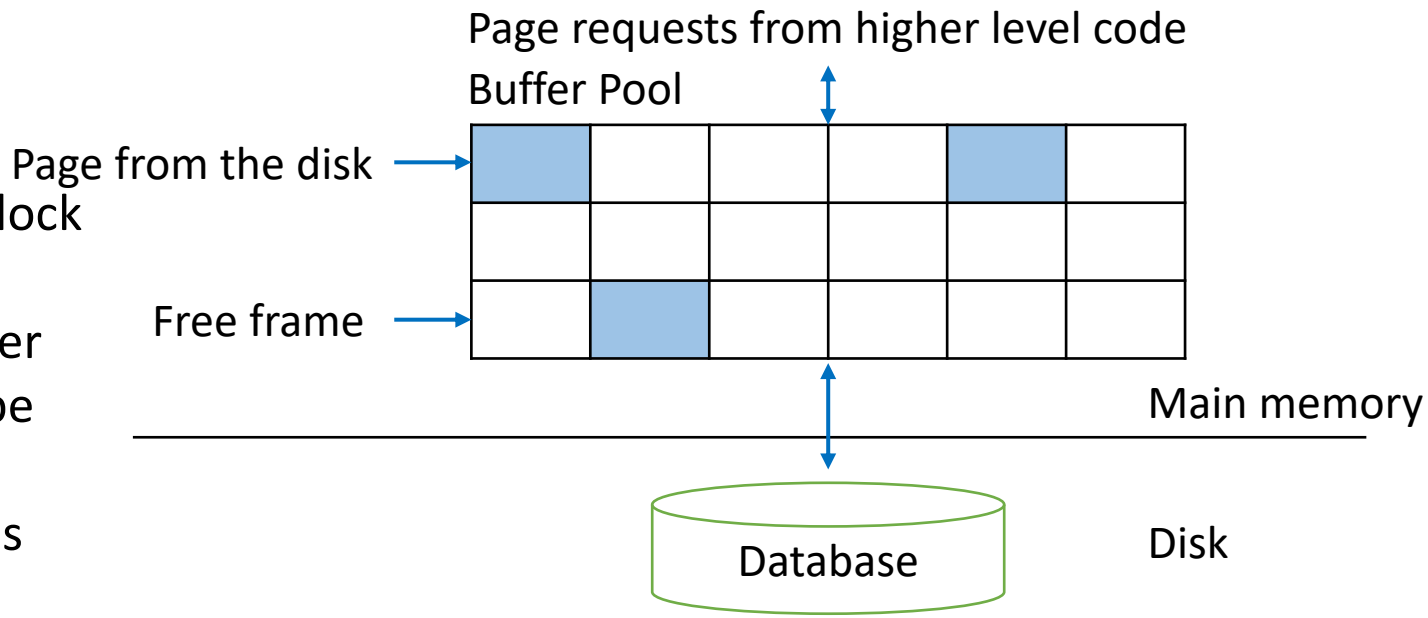
Buffer Manager

- *Buffer* = a partition of the internal memory used to store copies of the data blocks
- *Buffer Manager* = the DBMS part that is responsible with the buffer space management in the internal memory
- Buffer Manager is used when is need the accessing of a block from the disk (DBMS operate on the data from the internal memory)
- e.g. database has 300000 pages and in main memory 2000 available pages
- Buffer Manager brings new data pages from the disk to the main memory when are requires and decides what part of main memory pages can be replaced
- Buffer Manager manage the available main memory: collection of pages (*buffer pool*), pages in Buffer Manager with a slot that hold a page (*frame*)
- *Replacement* policy = The policy used to perform the choices of replacement frames in the Buffer Manager

The Physical Structure of Databases

Buffer Manager

- A table with the form <number_block_buffer, id_block_buffer> is modified
- If the block/page is not in the Buffer Manager:
  - Buffer Manager choose an available block for replacement (free frame)
  - When the block is no need it, the Buffer Manager release it and the block can be reused
  - If the block contains modifications, it is transferred to the disk
  - The wished block is read instead of the old block
- The block is fixed and return its address
- If the requirements are predictable (e.g. sequential scans), can be read in advance by multiple blocks at one moment



The Physical Structure of Databases

Buffer Manager

- The program that asked for the data block has to free it and indicate if the block was modified (it is used a *dirty bit*)
- The same data block from buffer can be used in multiple programs, by using a *pin count*
- For each frame, the Buffer Manager has 2 variables:
  - *pin_count* – number of current users (only the frames with *pin_count*=0 can be chosen as replacement frames)
  - *dirty* – boolean value that indicate if the page in the frame has been changed since being brought into the frame
- incrementing pin_count = pinning a page into a frame
- decrementing pin_count = unpinning a page

The Physical Structure of Databases

Buffer Manager

- initially, pin_count=0, dirty=off, for any frame from the Buffer Pool
- for a page, the Buffer Manager
  - checks if the page is in the Buffer Manager
    - if the page is in the Buffer Manager then pin_count(frame_that_contain_the_page)++
    - else, Buffer Manager chooses a frame_for_replacement
      - if Buffer Manager contains multiple frames with pin_count=0 then one frame is chosen accordingly to the Buffer Manager replacement policy
      - pin_count(frame_for_replacement)++
    - if dirty(frame_for_replacement)=on then Buffer Manager writes the page in frame_for_replacement to disk
    - Buffer Manager reads page in frame_for_replacement
  - Buffer Manager returns to the address of the Buffer Pool frame that contains the page

- Remark: if no Buffer Pool has pin_count=0 and page is not in Buffer Pool, then Buffer Manager has to wait / the transaction may be aborted

The Physical Structure of Databases

Buffer Manager

- The *Concurrency Control & Crash Recovery* may involve additional I/O operations on replacement of a block-buffer
- ***Least Recently Used*** (LRU) – uses the block usage template to predict what follows; queries have well access defined templates (e.g. sequential scans) and DBMS may use the information from the query to predict the following accesses to the blocks
  - queue of pointers to frame with pin_count=0
  - a frame is added to the end of the queue when its pin_count becomes 0
  - the frame at the head of the queue is chosen for replacement
- ***Toss-immediate*** – free the occupied space of a block, when was processed the last record from that block
- ***Most recently used*** (MRU) – after processing the last record from a block, the block is released (pin_count is decremented) and becomes the latest most used block
  - it is random

The Physical Structure of Databases

Replacement policies

- can have a significant impact on performance and also on the number of input/output – dependent on the *access template*
- Buffer Manager can use statistic information with respect to the probability that a specified request to refer to a specified block or to a relation
- *Sequential flooding* – problem generated by the LRU and the repeated sequential scans
  - number_of_buffer_blocks < number_of_table_blocks – each page request determine an I/O. Preferable MRU.
- Example:
  - Buffer Manager uses LRU
  - repeated scans of file f
  - Buffer Pool: 5 frames, f: <=5 pages
    - first scan of f brings all the pages in the Buffer Pool
    - subsequent scans find all the pages in the Buffer Pool
  - Buffer Pool: 5 frames, f: 6 pages
    - *sequential flooding*: every scan of f reads all the pages
    - MRU is better

The Physical Structure of Databases

DBMS versus File System of the Operating System

- Operating System manage the space on the disk and also the Buffer (also done by the DBMS)
- Operating System – support offers refers to portability problems
- Operating System – has limits (e.g. files that cannot be saved on multiple disks)
- Managing the buffer with DBMS brings also the abilities:
  - fixing / releasing blocks, forcing the saving of a block on the disk (important for *concurrency control & crash recovery*)
  - adjust the replacement policy and reading in advance of the blocks – based on the access template of the typical operations of the databases

The Physical Structure of Databases

Files of Records

- Are collections of records organized in one or more pages
- In DBMS on the higher level layers, the pages are treat as collections of records
- A file collection of pages can be organized in different manners
- Each record has an identifier = ***rid*** – helps in identifying the page that contain the record

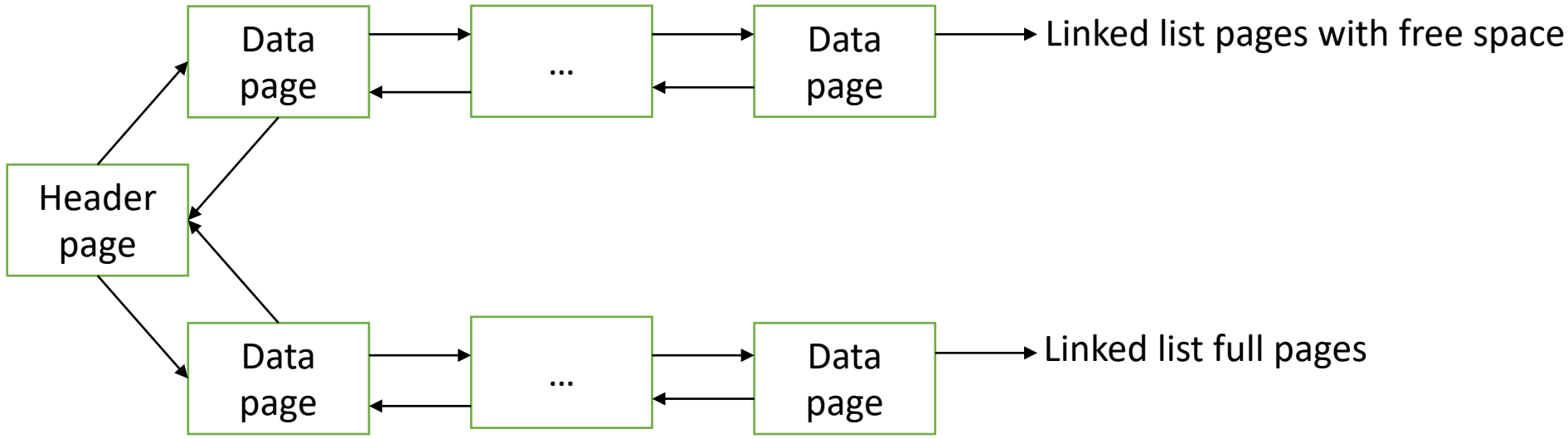
Heap Files

- is the simplest file structure in which the records are un-ordered
- operations: create / destroy file, insert a record (the pages are checked related to the free space), retrieve / delete a record (given by the rid), scan the records (a track with all the pages in the file should be kept)
- useful when the expected pattern of use includes scans to obtain all the records

The Physical Structure of Databases

Heap Files – Linked List

- doubly linked list of pages
- DBMS store the address of the first page (header page) of each file
- 2 lists: pages with free space, full pages
- drawback: variable-length records cause the fact that the pages will be in the list of pages with free space; by adding a record, multiple pages have to be checked until one is found with enough free space



The Physical Structure of Databases

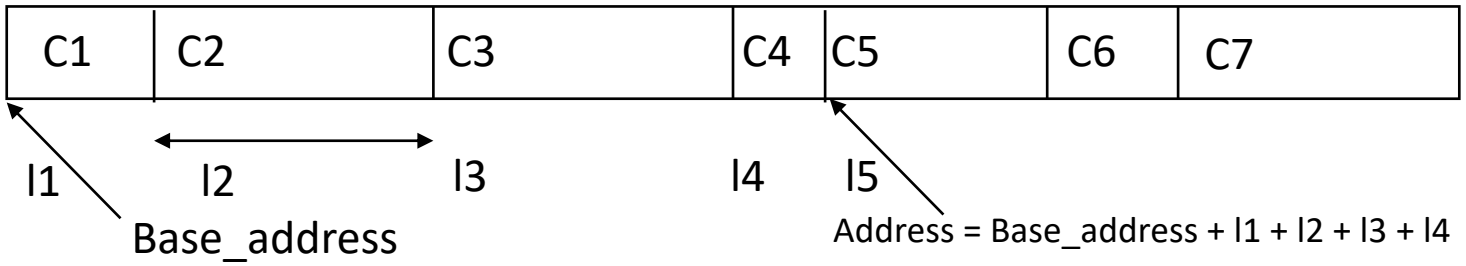
Heap Files – Directory of Pages

- For each heap file it is stored the location of the header page
- directory = collection of pages (e.g. linked list)
- directory entry – a page in the file
- directory entry size – smaller than the size of a page
- directory size – smaller than the size of the file
- free space management
  - 1 bit / directory entry – the corresponding page can have and or not free space
  - entry / count – available space on the corresponding page – efficient search of pages with enough free space on adding a variable-length record
- files that are sorted – useful for data that has to be sorted, on the range selections
- hashed files – useful for equality selections – the hashed is on the fields and the records are stored due to a hash function

The Physical Structure of Databases

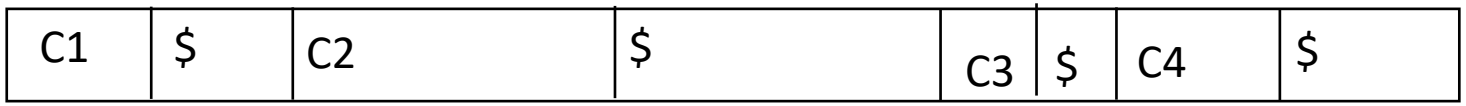
Formats for the records

- Fixed-length records
  - fixed number of fields
  - each field has a fixed length
  - the fields are stored consequently
  - the field address can be computed by the record address, length of the preceding fields (that are in the system catalog)

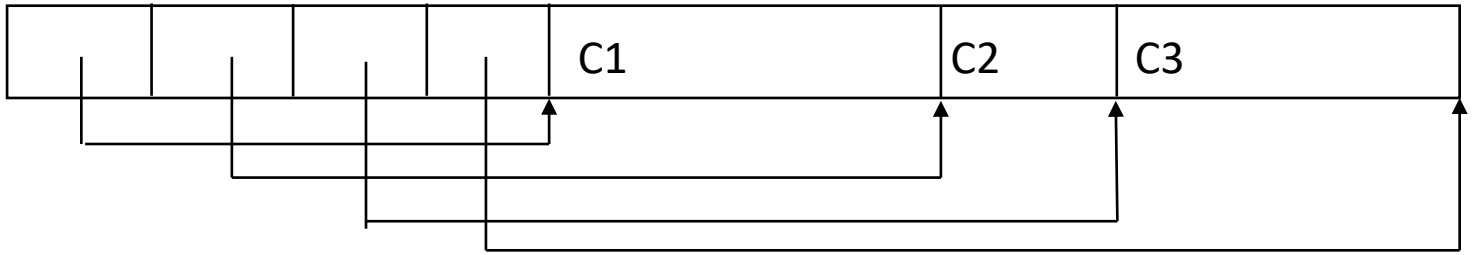


- Variable-length records

  Version 1:
  - variable-length fields
  - the fields are stored consecutively and are separated by delimiters
  - to find a field is performed a record scan



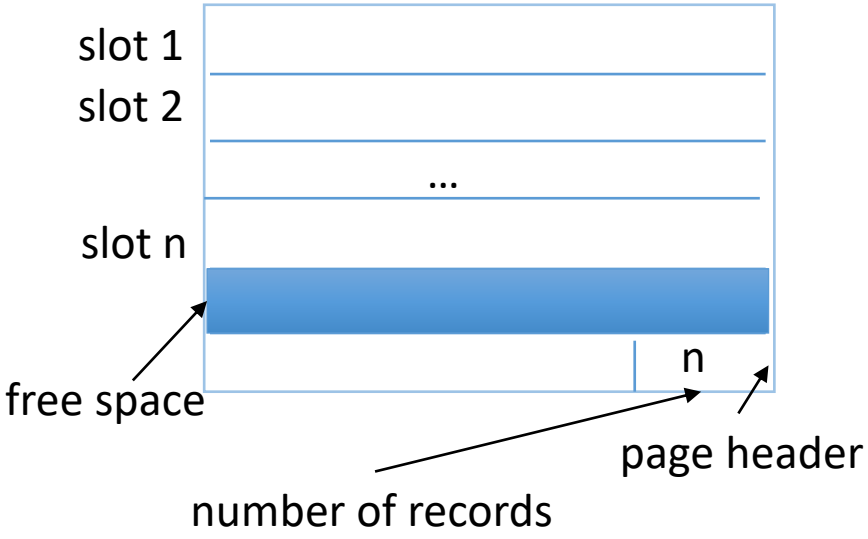
  Version 2: the space is reserved in the beginning of the record (array of fields offset, offset to the end of the record) – the array is overhead but provide direct access to every field



The Physical Structure of Databases

Page formats

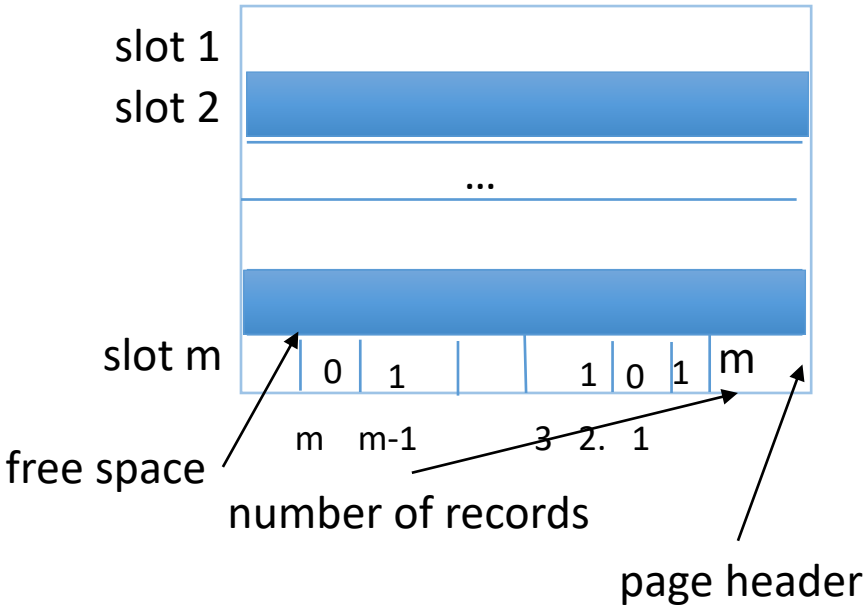
- Page – is a collection of slots with 1 record / slot
- Each record can be identified by using the ***rid***, record id, (page_id, slot_number)
- The records are arranged in pages in slots
- Fixed-length records
  - the records have the same size, are uniform and arranged in consecutive slots
  - adding a record involves the finding of an available slot
  - problems may arise on keeping track of available slots and on locating the records
    - version 1 – let *n* be the number of records on the page; records are stored in the first *n* slots
    - locate record *I* – compute the correspondent offset
    - delete a record – the last record from the page is moved intro a empty slot
    - empty slot – in the end of the page
    - when a moved record has external references – the slot number of the record would be changed, but the rid contains the slot number



The Physical Structure of Databases

Page formats

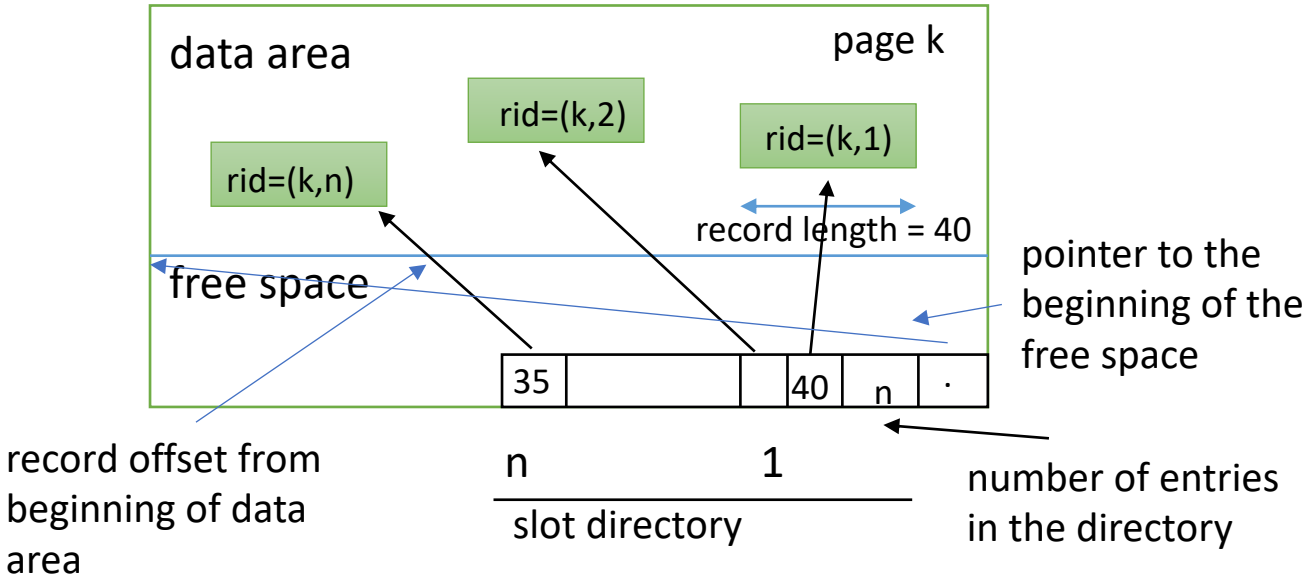
- Fixed-length records
  - version 2 – array of bits to monitor available slots
  - 1 bit / slot
  - delete a record - turn off the corresponding bit



The Physical Structure of Databases

Page formats

- Variable-length records
  - add record – find an empty slot of the right size
  - delete a record – contiguous free space
  - a directory of slots / page
  - a pair (record_offset, record_length) / slot
  - a pointer to the beginning of the free space area on the page
  - move a record on the page – only the changes of the record offset; the slot remains unmodified
  - can be used for fixed-length records (for sorted records)



References:

- C.J. Date, *An Introduction to Database Systems (8th Edition)*, Addison-Wesley, 2003.
- H. Garcia-Molina, J. Ullman, J. Widom, *Database Systems: The Complete Book, Prentice Hall Press*, 2008.
- G. Hansen, J. Hansen, *Database Management And Design (2nd Edition),* Prentice Hall, 1996.
- R. Ramakrishnan, J. Gehrke, *Database Management Systems*, McGraw- Hill, 2007. http://pages.cs.wisc.edu/~dbbook/openAccess/thirdEdition/slides/slides3ed.html
- R. Ramakrishnan, J. Gehrke, *Database Management Systems (2nd Edition),* McGraw-Hill, 2000.
- A. Silberschatz, H. Korth, S. Sudarshan, *Database System Concepts*, McGraw-Hill, 2010. http://codex.cs.yale.edu/avi/db-book/
- L. Țâmbulea, *Curs Baze de date*, Facultatea de Matematică și Informatică, UBB, 2013-2014.
- J. Ullman, J. Widom, *A First Course in Database Systems*, http://infolab.stanford.edu/~ullman/fcdb.html
- Secondary Storage and Disk Scheduling Algorithms, https://www.studytonight.com/operating-system/secondary-storage