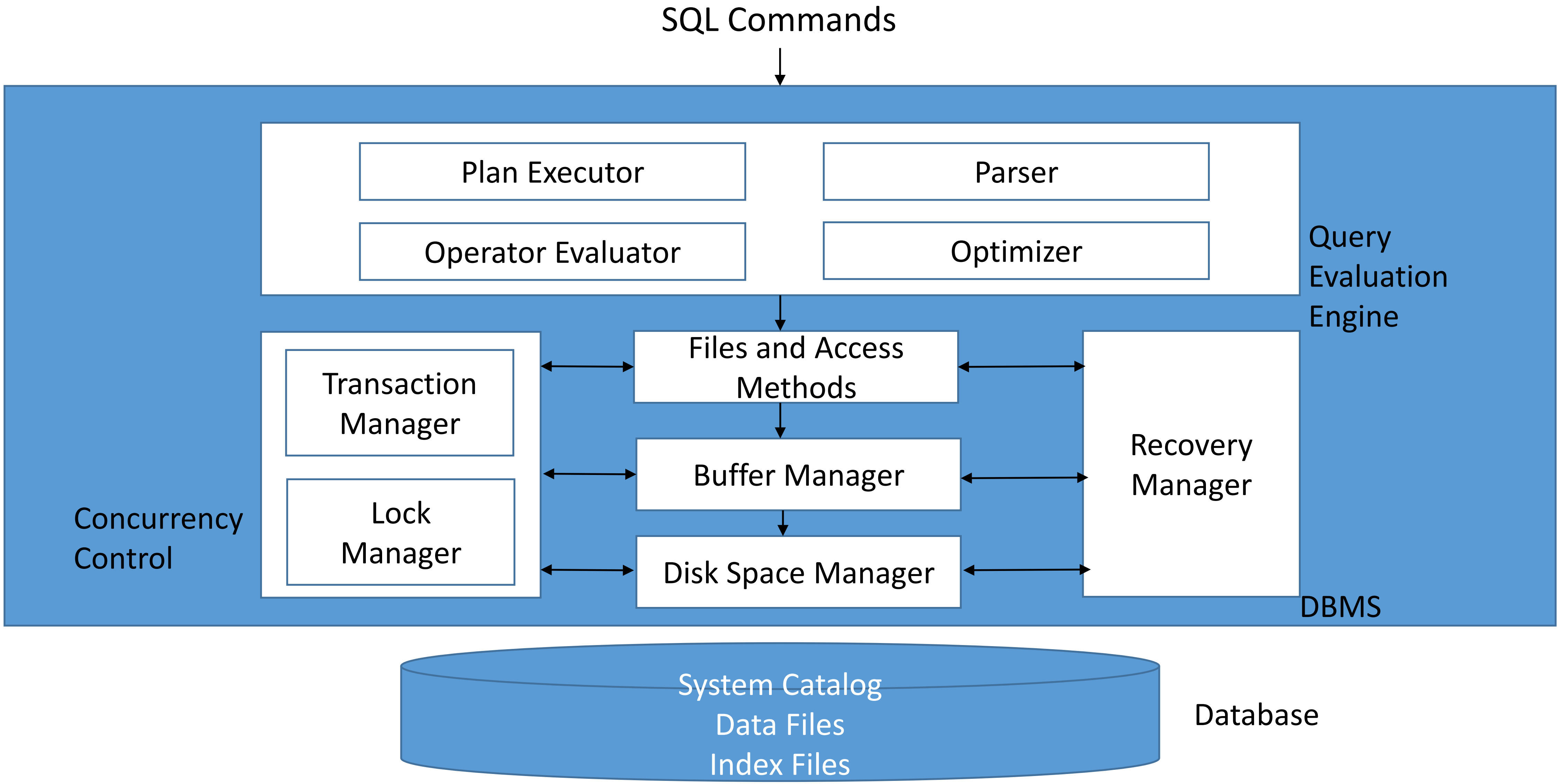
# Databases

Lecture 8

The Physical Structure of Databases

Sabina S. CS

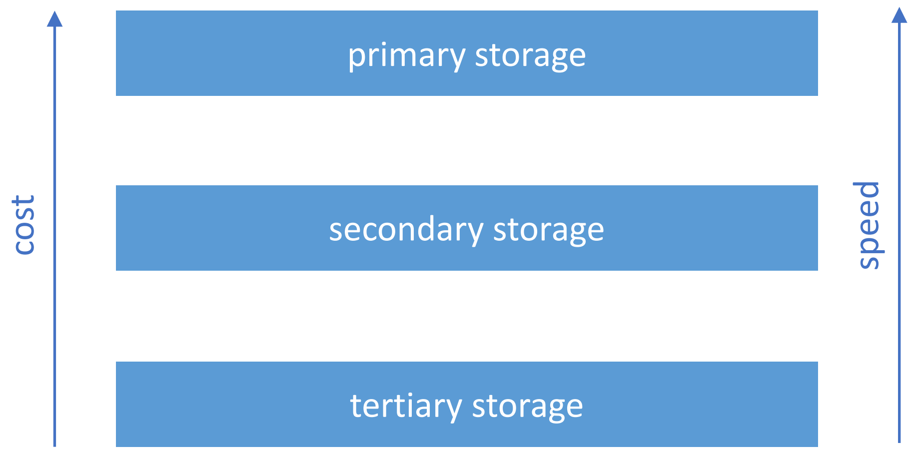
# DBMS Architecture



## The Memory Hierarchy

- primary storage
  - cache, main memory
  - very fast access to data
  - volatile
  - currently used data
- secondary storage
  - e.g., magnetic disks
  - slower storage devices
  - nonvolatile
  - disks - sequential, direct access
  - main database
- tertiary storage
  - e.g., optical disks, tapes
  - slowest storage devices
  - nonvolatile
  - tapes
    - only sequential access
    - good for archives, backups
    - unsuitable for data that is frequently accessed

# The Memory Hierarchy




- disks and tapes - significantly cheaper than main memory
- large amounts of data that shouldn't be discarded when the system is restarted

=> the need for DBMSs that bring data from disks into main memory for processing

# Secondary Storage – Magnetic Disks

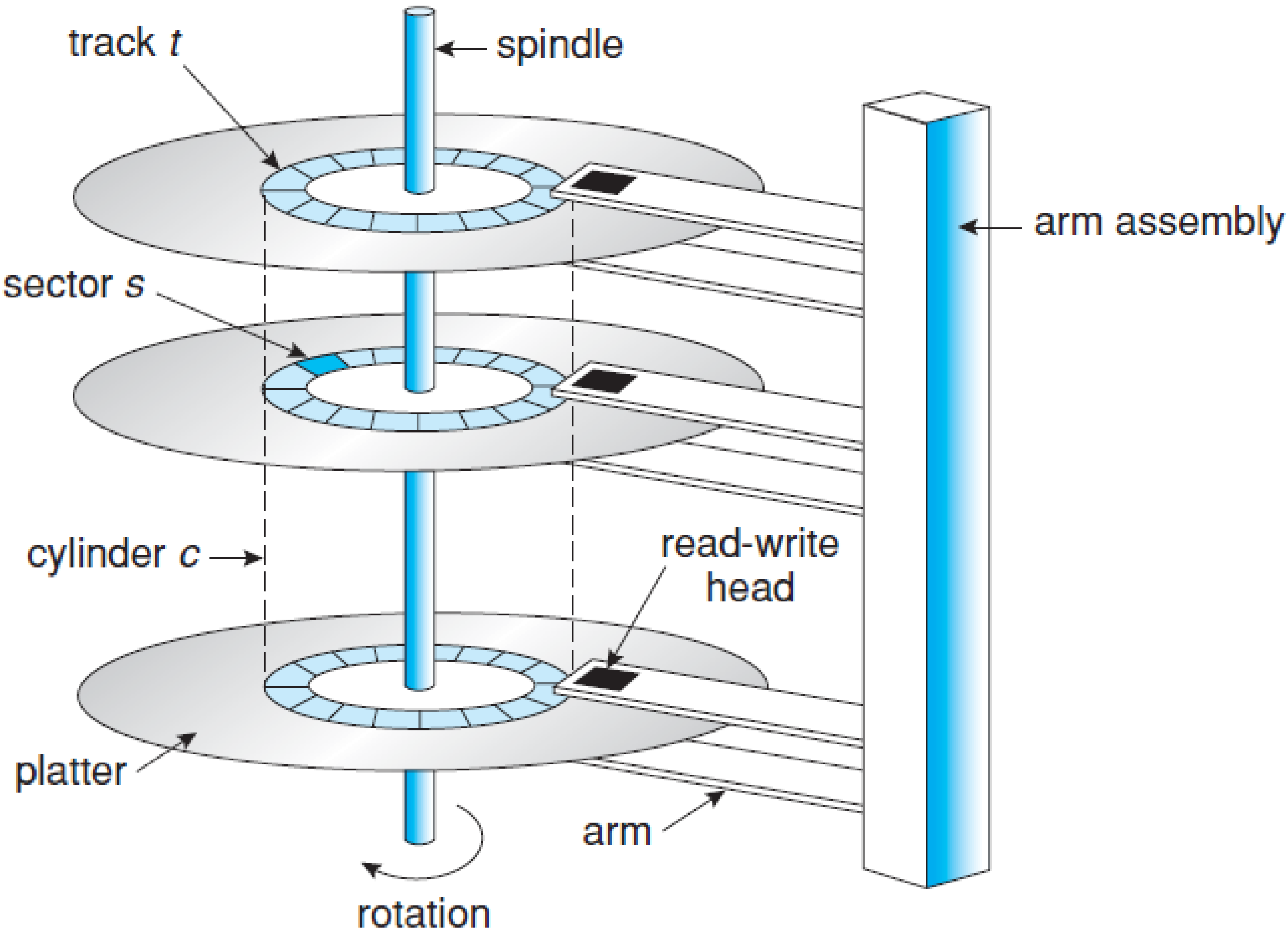
- direct access
- extremely used in database applications
- DBMSs - applications don't need to know whether the data is on disk or in main memory
- *disk block*
  - sequence of contiguous bytes
  - unit for data storage
  - unit for data transfer (reading data from disk / writing data to disk)
  - reading / writing a block - an input / output (I/O) operation
- *tracks*
  - concentric rings containing blocks, recorded on one or more platters

# Secondary Storage – Magnetic Disks

- *sectors*
    - arcs on tracks
- *platters*
    - single-sided, double-sided (data recorded on one / both surfaces)
- *cylinder*
    - set of all tracks with the same diameter
- *disk heads*
    - one per recorded surface
    - to read / write a block, a head must be on top of the block
    - all disk heads are moved as a unit
    - systems with one active head

# Secondary Storage – Magnetic Disks

- sector size
  - characteristic of the disk, cannot be modified
- block size
  - multiple of the sector size




[Si08]

Secondary Storage – Magnetic Disks

- DBMSs operate on data when it is in memory
- block - unit for data transfer between disk and main memory
- time to access a desired location:
  - main memory - approximately the same for any location
  - disk - depends on where the data is stored
- disk access time:
  - seek time + rotational delay + transfer time
  - seek time
    - time to move the disk head to the desired track (smaller platter size => 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 head is positioned over it

## Secondary Storage – Magnetic Disks

- time required for DB operations - dominated by the time taken to transfer blocks between disk and main memory
- goal
    - minimize access time
    - for this purpose, data should be carefully placed on disk
- records that are often used together should be close to each other:
    - same block
    - same track
    - same cylinder
    - adjacent cylinder
- accessing data in a sequential fashion reduces seek time and rotational delay

Secondary Storage – Magnetic Disks

* characteristics, e.g.:

- storage capacity (e.g., GB)
- platters
  - number, *single-sided* or *double-sided*
- average / max seek time (ms)
- average rotational delay (ms)
- number of rotations / min
- data transfer rate (MB/s)
- ...

## Moore's Law

- Gordon Moore: "the improvement of integrated circuits is following an exponential curve that doubles every 18 months"
- parameters that follow Moore’s law
  - speed of processors (number of instructions executed / sec)
  - no. of bits / chip
  - capacity of largest disks
- parameters that do not follow Moore’s law
  - speed of accessing data in main memory
  - disk rotation speed

=> "latency" keeps increasing

- time to move data between memory hierarchy levels appears to take longer compared with computation time

# Solid-State Disks

- NAND flash components
- faster random access
- higher data transfer rates
- no moving parts
- higher cost per GB
- limited write cycles

# Managing Disk Space

- the *disk space manager* (DSM) manages space on disk
- page
  - unit of data
  - size of a page = size of a disk block
  - R/W a page - one I/O operation
- upper layers in the DBMS can treat the data as a collection of pages
- DSM
  - commands to allocate / deallocate / read / write a page
  - knows which pages are on which disk blocks
  - monitors disk usage, keeping track of available disk blocks

## Managing Disk Space

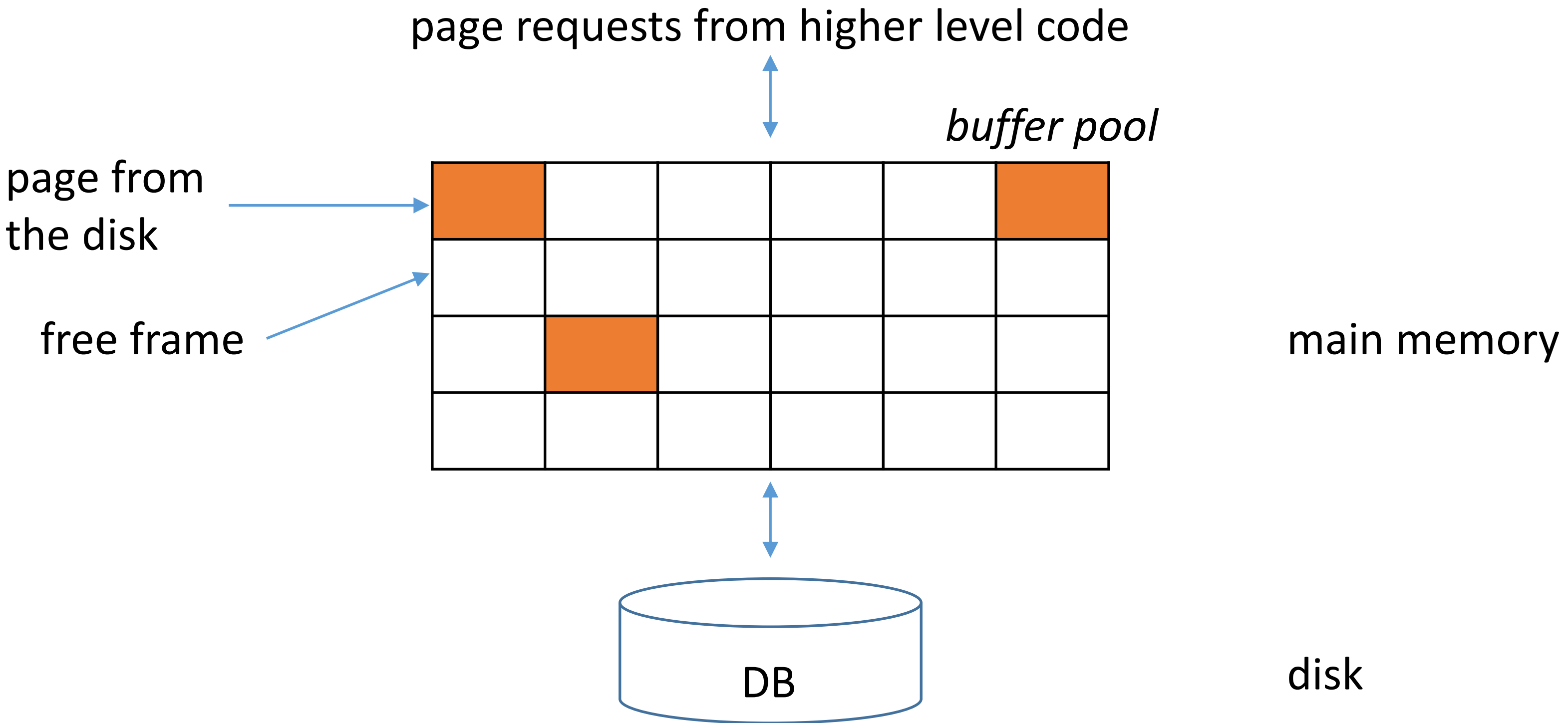
- free blocks can be identified:
  - by maintaining a linked list of free blocks (on deallocation, a block is added to the list)
  - by maintaining a bitmap with one bit / block, indicating whether the corresponding block is used or not
    - allows for fast identification of contiguous available areas on disk

## Buffer Manager

- e.g., DB = 500.000 pages, main memory - 1000 available pages, query that scans the entire file
- *buffer manager* (BM)
  - brings new data pages from disk to main memory as they are required
  - decides what main memory pages can be replaced
  - manages the available main memory
    - collection of pages called the *buffer pool* (BP)
    - *frame*
      - page in the BP
      - slot that can hold a page
- *replacement policy*
  - policy that dictates the choice of replacement frames in the BP

Buffer Manager

- higher level layer L in the DBMS asks the BM for page P
- if P is not in the BP, the BM brings it into a frame F in the BP
- when P is no longer needed, L notifies the BM (it releases P), so F can be reused
- if P has been modified, L notifies the BM, which propagates the changes in F to the disk



# Buffer Manager

- BM maintains 2 variables for each frame F
  - *pin_count*
    - number of current users (requested the page in F but haven't released it yet)
    - only frames with pin_count = 0 can be chosen as replacement frames
  - *dirty*
    - boolean value indicating whether the page in F has been changed since being brought into F
- incrementing pin_count
  - pinning a page P in a frame F
- decrementing pin_count
  - unpinning a page

## Buffer Manager

- initially, pin_count = 0, dirty = off, ∀ F ∈ BP
- L asks for a page P; the BM:

1. checks whether page P is in the BP; if so, pin_count(F)++, where F is the frame containing P

otherwise:

a. BM chooses a frame FR for replacement

- if the BP contains multiple frames with pin_count = 0, one frame is chosen according to the BM's replacement policy
- pin_count(FR)++;

b. if dirty(FR) = on, BM writes the page in FR to disk

c. BM reads page P in frame FR

2. the BM returns the address of the BP frame that contains P to L

## Buffer Manager

- obs. if no BP frame has pin_count = 0 and page P is not in BP, BM has to wait / the transaction may be aborted
- page requested by several transactions; no conflicting updates
- crash recovery, Write-Ahead Log (WAL) protocol - additional restrictions when a frame is chosen for replacement

- replacement policies
  - *Least Recently Used (LRU)*
    - queue of pointers to frames 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
  - *Most Recently Used (MRU)*
  - *random*
  - *...*

Buffer Manager

- replacement policies
  - *clock replacement*
    - LRU variant
    - *n* – number of frames in BP
    - frame - *referenced* bit; set to *on* when *pin_count* becomes 0
    - *crt* variable - frames 1 through *n*, circular order
    - if the current frame is not chosen, then *crt*++, examine next frame

    - if *pin_count* > 0
      - current frame not a candidate, *crt*++
    - if *referenced = on*
      - *referenced* := *off*, *crt*++
    - if *pin_count* = 0 AND *referenced = off*
      - choose current frame for replacement

# Buffer Manager

- replacement policies
    - can have a significant impact on performance
- example:
    - BM uses LRU
    - repeated scans of file *f*
    - BP: 5 frames, *f*: <= 5 pages
        - first scan of *f* brings all the pages in the BP
        - subsequent scans find all the pages in the BP
    - BP: 5 frames, *f*: 6 pages
        - *sequential flooding*: every scan of *f* reads all the pages
        - MRU – better in this case

## Disk Space Manager & Buffer Manager

- DSM
  - portability - different OSs
- BM
  - DBMS can anticipate the next several page requests (operations with a known page access pattern, like sequential scans)
  - *prefetching* - BM brings pages in the BP before they are requested
  - prefetched pages
    - contiguous: faster reading (than reading the same pages at different times)
    - not contiguous: determine an access order that minimizes seek times / rotational delays

## Disk Space Manager & Buffer Manager

- BM
  - DBMS needs
    - ability to explicitly force a page to disk
    - ability to write some pages to disk before other pages are written
      - WAL protocol - first write log records describing page changes, then write modified page

# Files and Indexes

## Files of Records

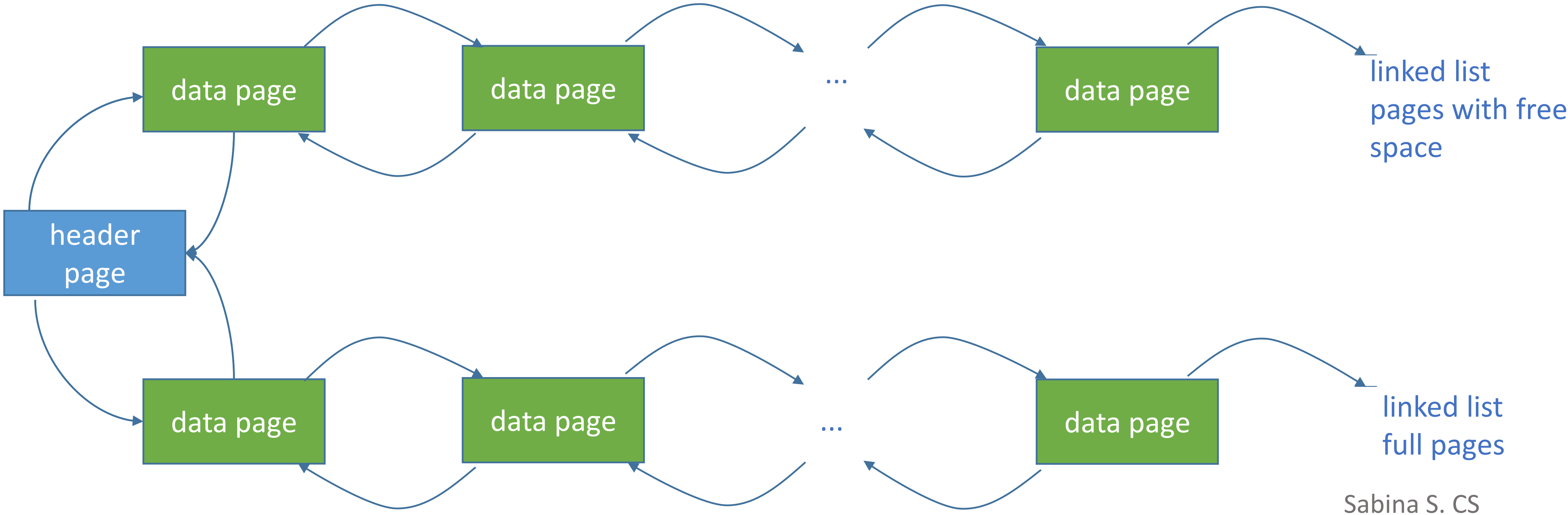
- higher level layers in the DBMS treat pages as collections of records
- file of records
  - collection of records; one or more pages
- different ways to organize a file's collection of pages
- every record has an identifier: the rid
- given the rid of a record, one can identify the page that contains the record

Heap Files

- the simplest file structure
- records are not ordered
- supported operations
    - create file
    - destroy file
    - insert a record
        - need to monitor pages with free space
    - retrieve a record given its rid
    - delete a record given its rid
    - scan all records
        - need to keep track of all the pages in the file
- appropriate when the expected pattern of use includes scans to obtain all the records

# Heap Files - Linked List

- doubly linked list of pages
- DBMS stores the address of the first page (*header page*) of each file (a table holding pairs of the form <*heap_file_name*, *page1_address*>)
- 2 lists – pages with free space and full pages

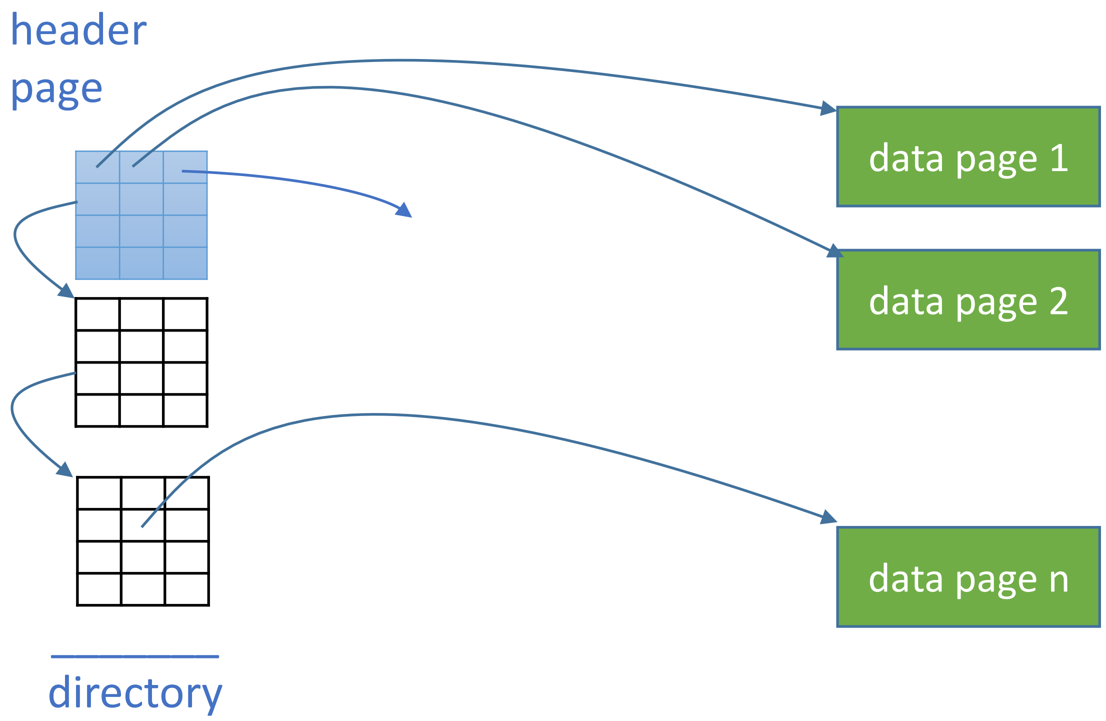


# Heap Files - Linked List

- drawback
  - variable-length records => most of the pages will be in the list of pages with free space
  - when adding a record, multiple pages have to be checked until one is found that has enough free space

# Heap Files - Directory of Pages

- DBMS stores the location of the header page for each heap file
- directory - collection of pages (e.g., linked list)
- directory entry - identifies a page in the file
- directory entry size - much smaller than the size of a page
- directory size - much smaller than the size of the file
- free space management
    - 1 bit / directory entry - corresponding page has / doesn't have free space
    - count / entry - available space on the corresponding page => efficient search of pages with enough free space when adding a variable-length record

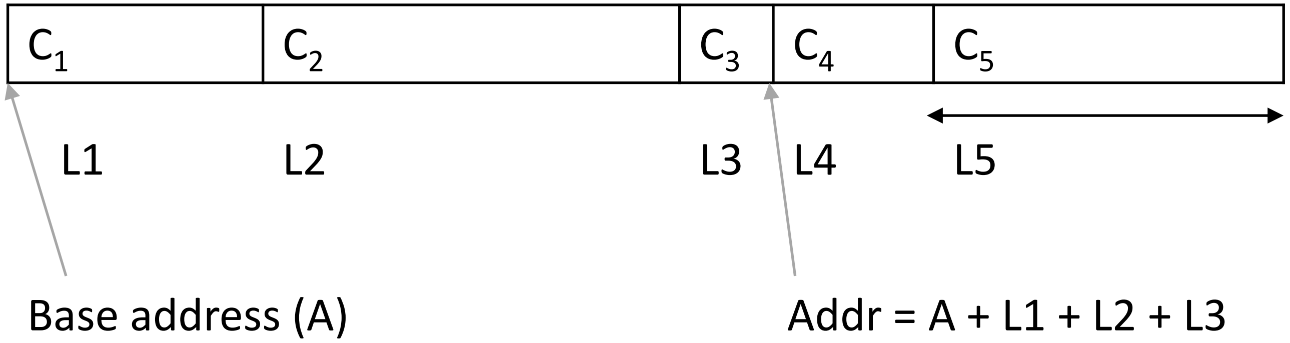


# Other File Organizations

- sorted files
  - suitable when data must be sorted, when doing range selections
- hashed files
  - files that are hashed on some fields (records are stored according to a hash function); good for equality selections

# Record Formats

- fixed-length records




  - each field has a fixed length
  - fixed number of fields
  - fields - stored consecutively
  - computing a field's address
    - record address, length of preceding fields (from the system catalog)

# Record Formats

- variable-length records
  - variable-length fields

v1

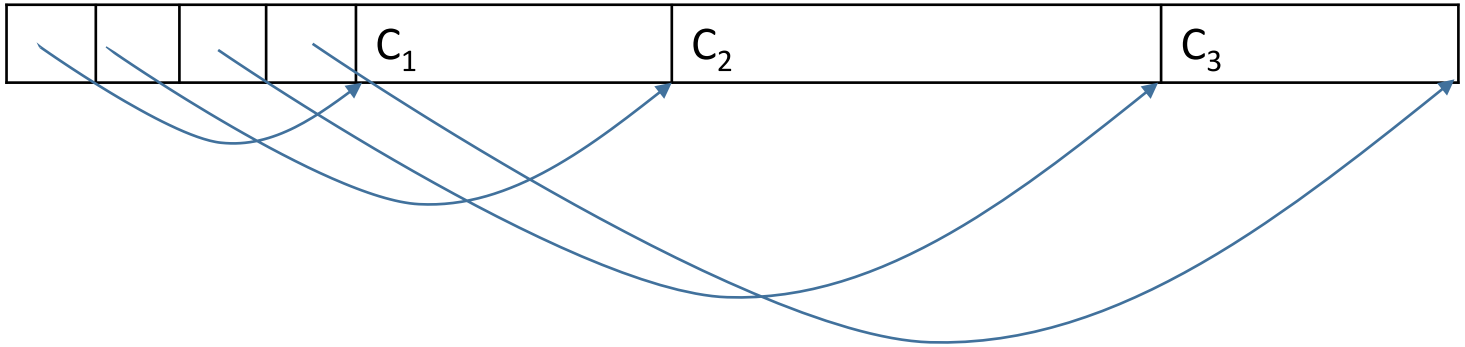
| | | | | | |
|---|---|---|---|---|---|
| $C_1$ | $ | $C_2$ | $ | $C_3$ | $ |

  - fields
    - stored consecutively, separated by delimiters
  - finding a field
    - a record scan

## Record Formats

- variable-length records

v2




- reserve space at the beginning of the record
  - array of fields offsets, offset to the end of the record
- array overhead, but direct access to every field

# Page Formats

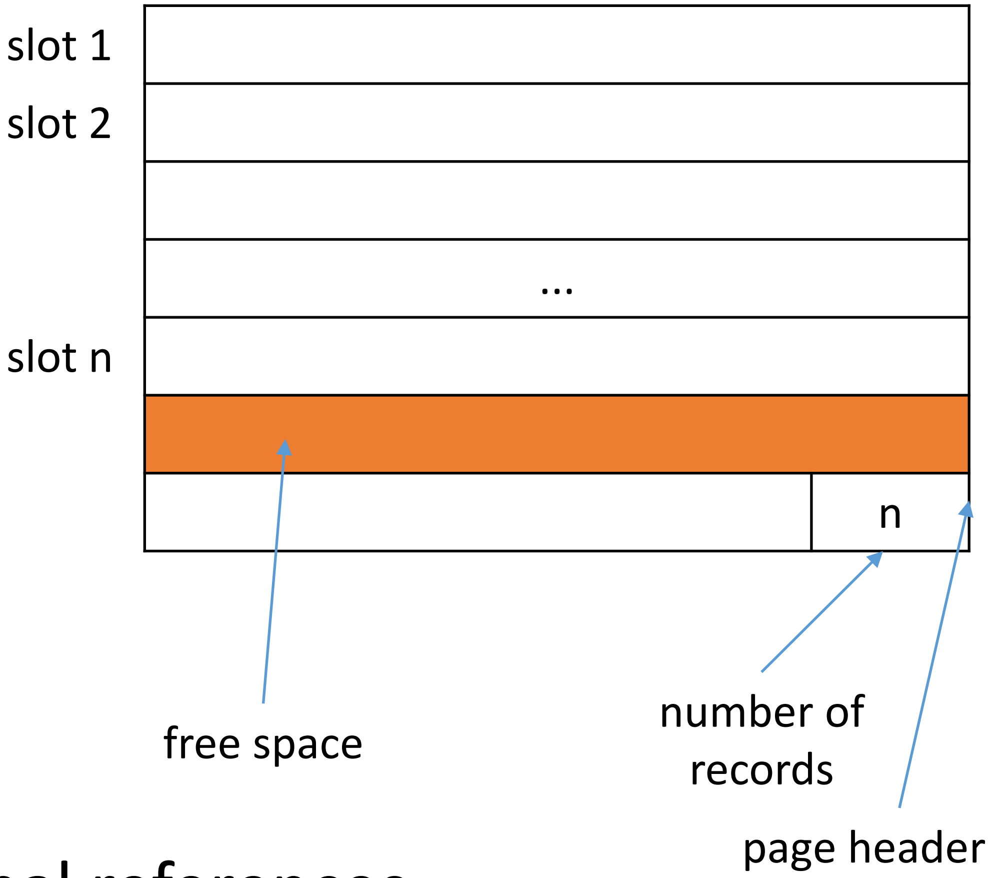
- page
  - collection of slots
  - 1 record / slot
- identifying a record
  - record id (rid): <page id, slot number>
- how to arrange records on pages
- how to manage slots

## Page Formats

- fixed-length records
  - records have the same size
  - uniform, consecutive slots
  - adding a record
    - finding an available slot
  - problems
    - keeping track of available slots
    - locating records

# Page Formats

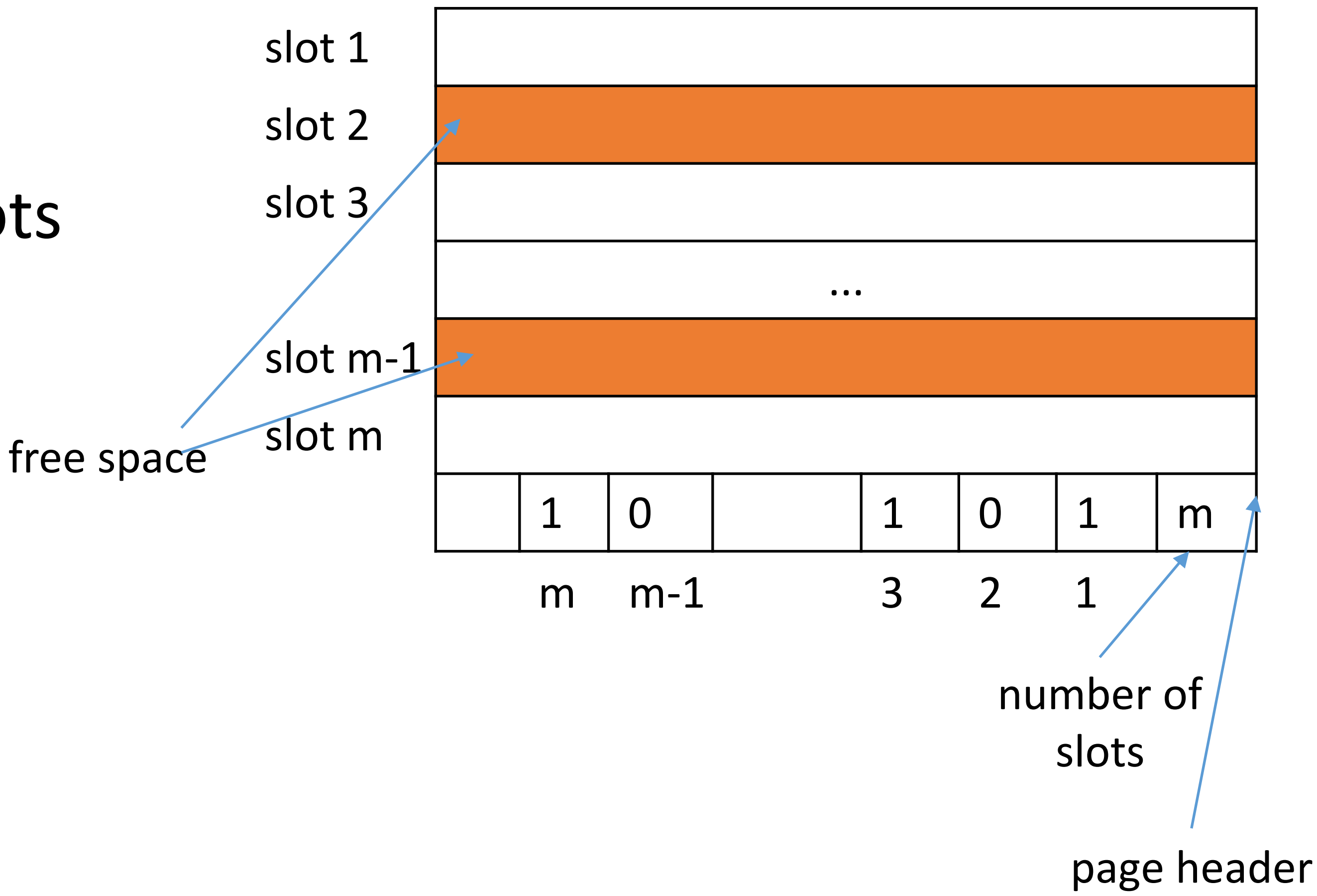
- fixed-length records - v1
  - *n* – number of records on the page
  - records are stored in the first *n* slots
  - locating record *i* - compute corresponding offset
  - deleting a record - the last record on the page is moved into the empty slot
  - empty slots - at the end of the page




  - problems when a moved record has external references
    - the record's slot number would change, but the rid contains the slot number!

# Page Formats

- fixed-length records - v2
- array of bits to monitor available slots
- 1 bit / slot
- deleting a record - turning off the corresponding bit

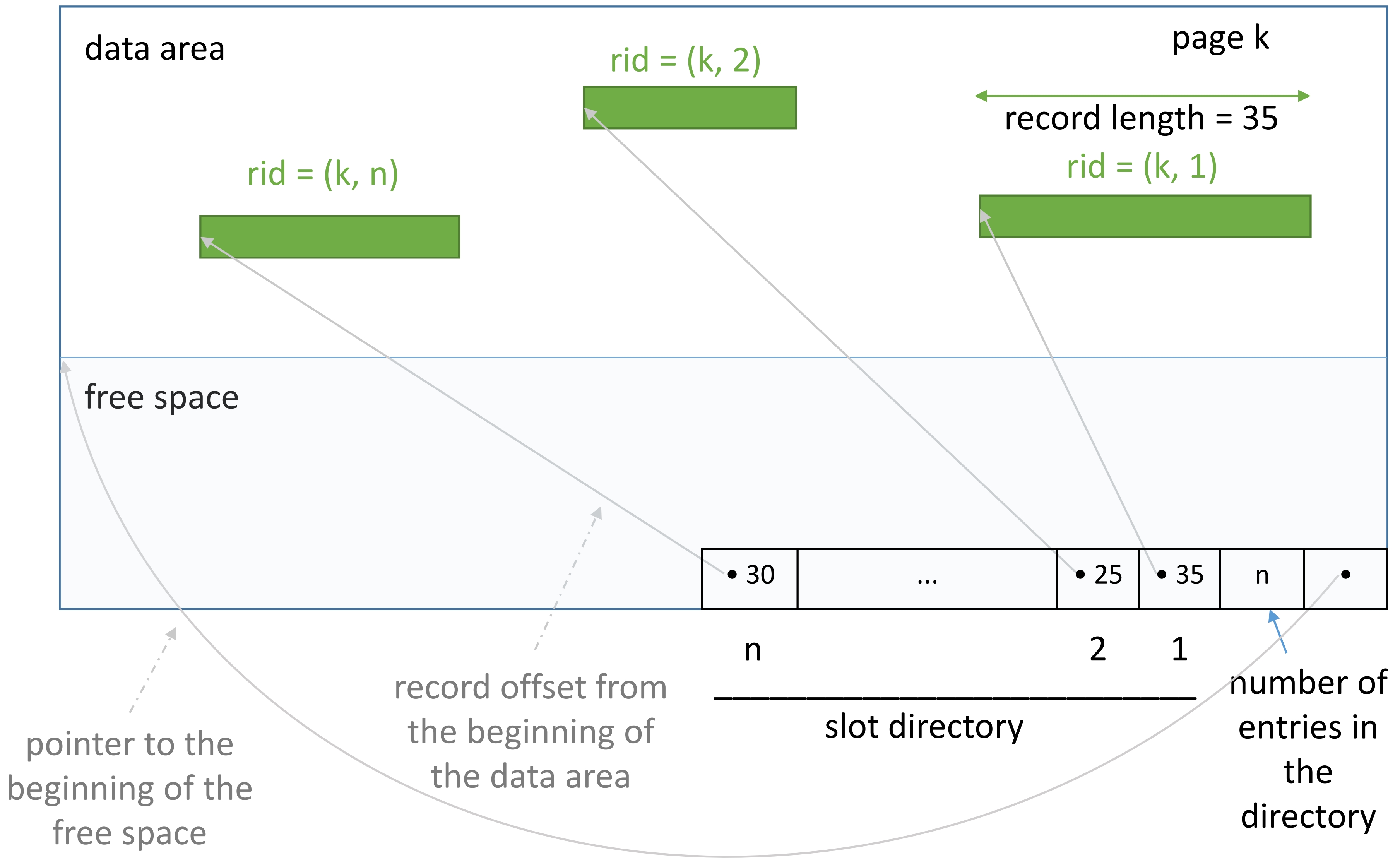


Page Formats

- variable-length records
  - adding a record
    - finding an empty slot of the right size
  - deleting 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
  - moving a record on the page
    - only the record's offset changes
    - its slot remains unmodified
  - can also be used for fixed-length records (when records need to be kept sorted)

# Page Formats

- variable-length records



Indexes

- motivating example
  - file of students records sorted by name
    - good file organization
      - retrieve students in alphabetical order
    - not a good file organization
      - retrieve students whose age is in a given range
      - retrieve students who live in Timișoara
- index
  - auxiliary data structure that speeds up operations which can't be efficiently carried out given the file's organization
  - enables the retrieval of the rids of records that meet a selection condition (e.g., the rids of records describing students who live in Timișoara)

## Indexes

- *search key*
  - set of one or more attributes of the indexed file (different from the *key* that identifies records)
- an index speeds up queries with equality / range selection conditions on the search key
- *entries*
  - records in the index (e.g., <search key, rid>)
  - enable the retrieval of records with a given search key value

## Indexes

- example
  - files with students records
  - index built on attribute *city*
  - entries: <city, rid>, where rid identifies a student record
  - such an index would speed up queries about students living in a given city:
    - find entries in the index with city *= 'Timișoara'*
    - follow rids from obtained entries to retrieve records describing students who live in Timișoara

# Indexes

- an index can improve the efficiency of certain types of queries, not of all queries (analogy - when searching for a book at the library, index cards sorted on author name cannot be used to efficiently locate a book given its title)

- organization techniques (access methods) - examples
  - B+ trees
  - hash-based structures

- changing the data in the file => update the indexes associated with the file (e.g., inserting records, updating search key columns, updating columns that are not part of the key, but are included in the index)
- index size
  - as small as possible, as indexes are brought into main memory for searches

## Indexes - Data Entries

- problems
  - what does a data entry contain?
  - how are the entries of an index organized?

- let k* be a data entry in an index; the data entry:
  - alternative 1

    is an actual data record with search key value = k
  - alternative 2
    - is a pair <k, rid> (rid – id of a data record with search key value = k)
  - alternative 3
    - is a pair <k, rid_list> (rid_list – list of ids of data records with search key value = k)

## Indexes - Data Entries

- a1
  - the file of data records needn't be stored in addition to the index
  - the index is seen as a special file organization
  - at most 1 index / collection of records should use alternative a1 (to avoid redundancy)
- a2, a3
  - data entries point to corresponding data records
  - in general, the size of an entry is much smaller than the size of a data record
  - a3 is more compact than a2, but can contain variable-length records
  - can be used by several indexes on a collection of records
  - independent of the file organization

# References


- [Ta13] ȚÂMBULEA, L., Curs Baze de date, Facultatea de Matematică și Informatică, UBB, 2013-2014
- [Ra00] RAMAKRISHNAN, R., GEHRKE, J., Database Management Systems (2nd Edition), McGraw-Hill, 2000
- [Da03] DATE, C.J., An Introduction to Database Systems (8th Edition), Addison-Wesley, 2003
- [Ga08] GARCIA-MOLINA, H., ULLMAN, J., WIDOM, J., Database Systems: The Complete Book, Prentice Hall Press, 2008
- [Ra07] RAMAKRISHNAN, R., GEHRKE, J., Database Management Systems, McGraw-Hill, 2007, http://pages.cs.wisc.edu/~dbbook/openAccess/thirdEdition/slides/slides3ed.html
- [Si08] SILBERSCHATZ, A., GALVIN, P.B., GAGNE, G., Operating System Concepts (8th Edition), Wiley Publishing, 2008
- [Ul11] ULLMAN, J., WIDOM, J., A First Course in Database Systems, http://infolab.stanford.edu/~ullman/fcdb.html