Databases

Lecture 8

The Physical Structure of Databases

Files and Indexes

The Physical Structure of Databases

- 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
  - 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
  - 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

- the time required for DB operations is usually significantly lower than 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

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)

- …

Gordon Moore

- "the improvement of integrated circuits is following an exponential curve that doubles every 18 months"
    - speed of processors, e.g., number of instructions executed / sec
    - no. of bits / chip
    - disks capacities
- parameters that do not follow Moore's law
    - access speed of 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 Drives
- NAND flash components
- extremely low latency
- very fast read / write speeds
- no moving parts, completely silent
- cost per GB - higher than HDDs
- limited write cycles

Managing Space on the Disk

- 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 operate with data as a collection of pages; DSM hides details of the hardware

- 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 Space on the Disk

- free blocks can be identified:
  - by maintaining them as a linked list (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 data set
- *buffer manager* (BM)
  - brings new data pages from disk into 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)
    - *frames*
      - pages in the BP
      - slots 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 releases P
    - the BM is notified, so F can be reused
    - if P has been modified, the BM is notified and propagates the changes in the BP to the disk

Buffer Manager

page requests from higher level code

page from the disk

buffer pool

main memory

free frame

DB

disk
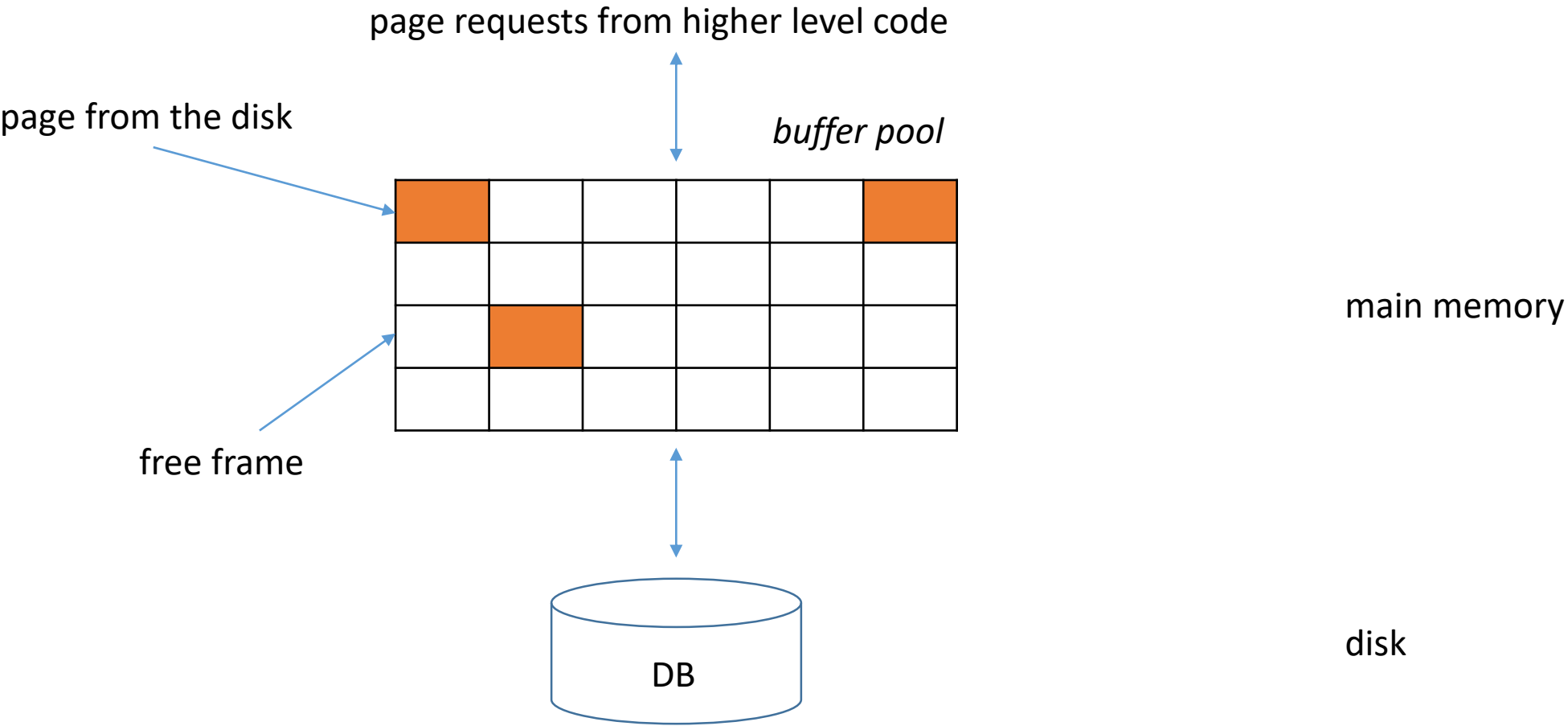
Buffer Manager

- BM maintains 2 variables for every 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, $\forall$ F $\in$ BP
- L asks for a page P; the BM:

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

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 L the address of the BP frame that contains P

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

Buffer Manager

- replacement policies
  - *Least Recently Used (LRU)*
    - queue of pointers to frames with pin_count = 0
    - a frame is added to 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
- e.g.:
  - 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 (but not in all cases)

Disk Space Manager & Buffer Manager

- DSM
  - portability - different OSs
  - specialized disk management, i.e., Disk Space Manager
- BM
  - DBMS can anticipate the next several page requests (operations with a known page access pattern, e.g., 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
  - specialized main memory management, i.e., Buffer Manager

Files and Indexes

Files

- 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, i.e., the rid
- given the rid of a record, one can obtain the page that contains the record

- 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

Heap Files

- appropriate when the expected pattern of use includes frequent 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, i.e., table holding pairs of the form <*heap_file_name, page1_address*>

- 2 lists – pages with free space and full pages
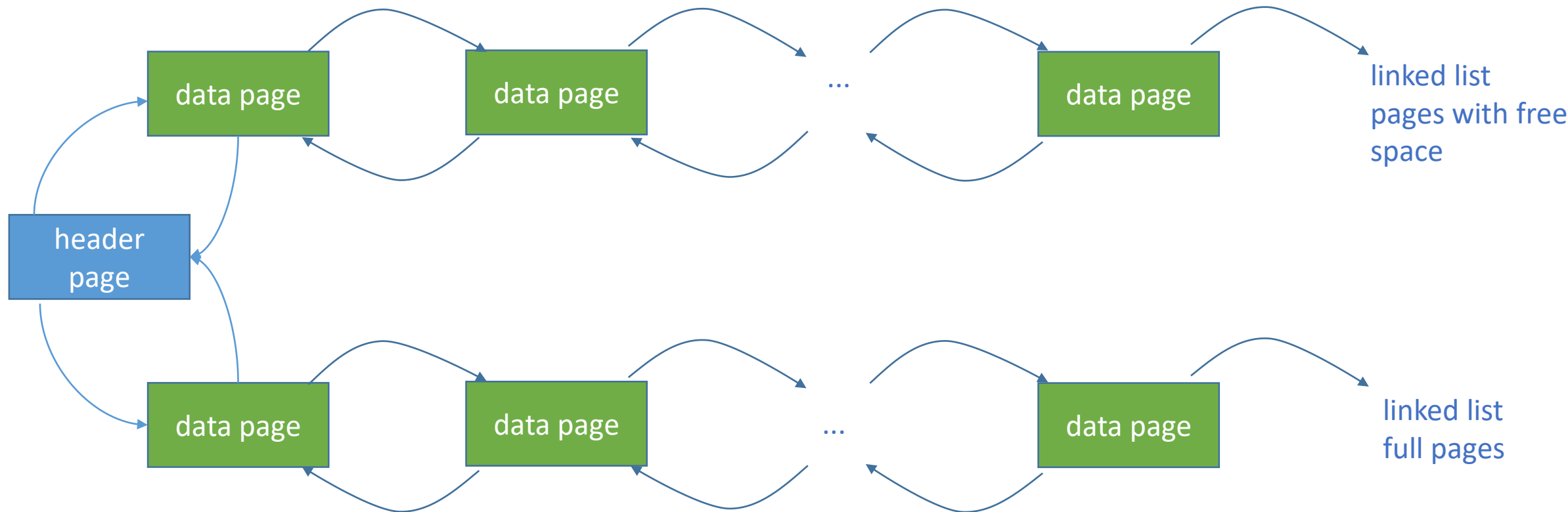
Heap Files - Linked List

- 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

- 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

header
page

data page 1

data page 2

directory

data page n
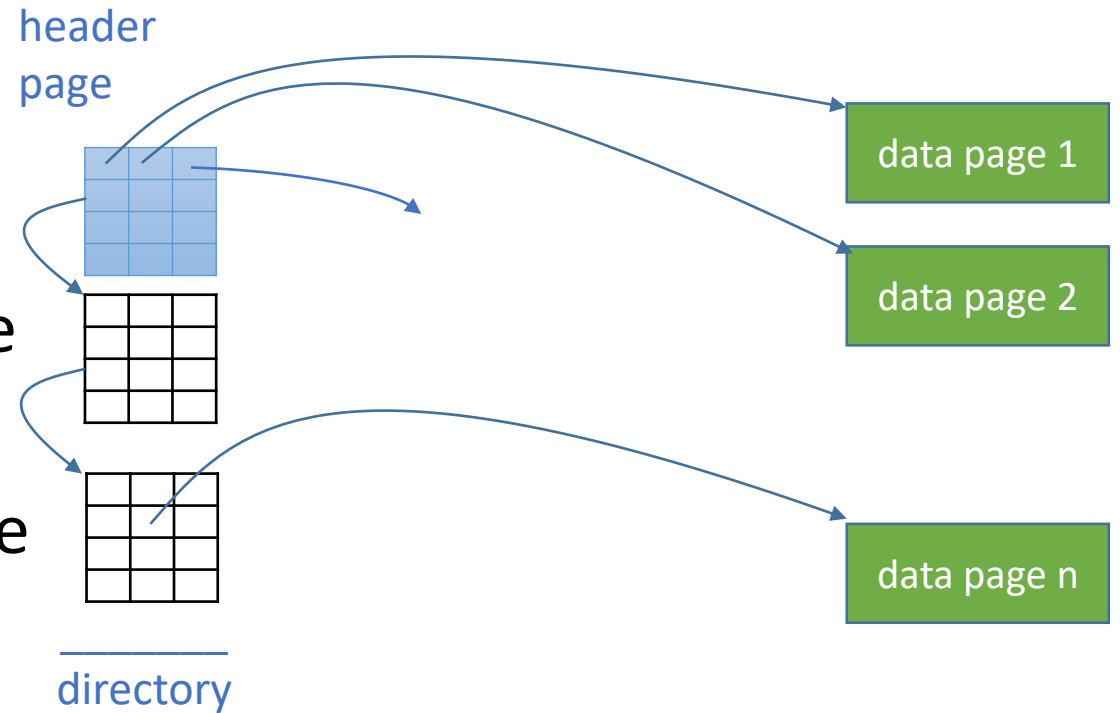
Heap Files - Directory of Pages

- 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

header page

directory
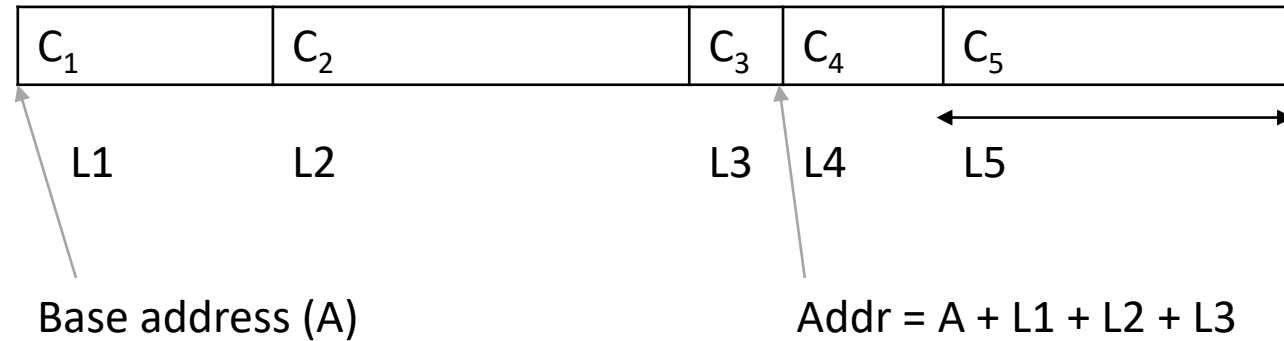
data page 1

data page 2

data page n
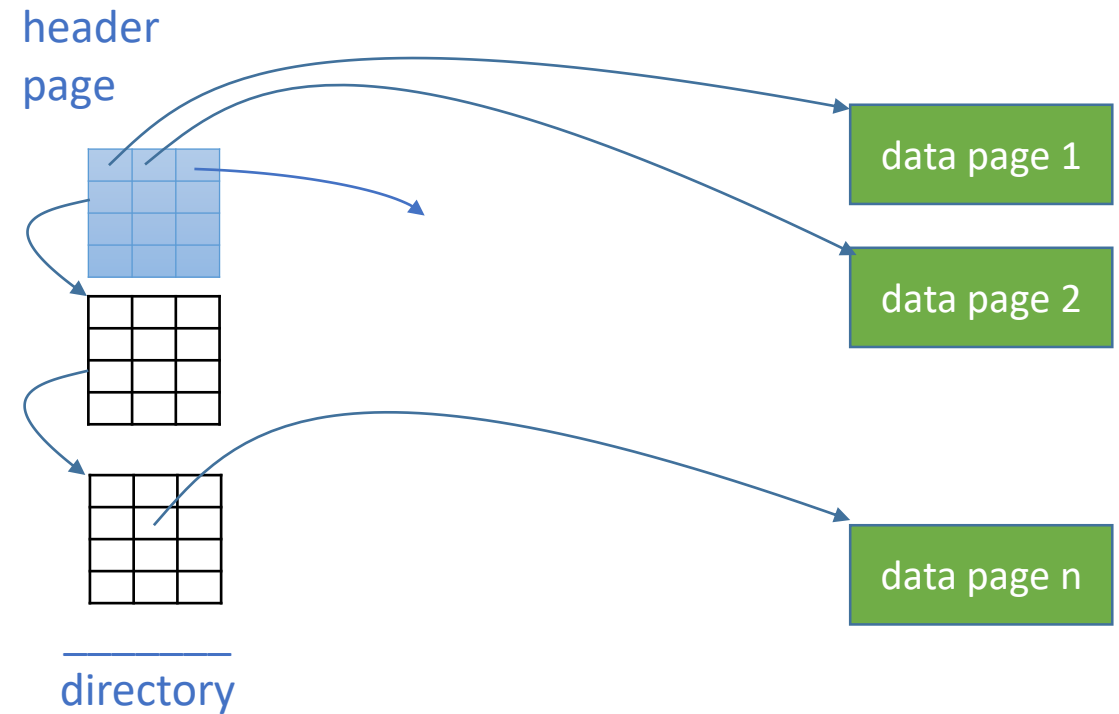
Files - Other File Organizations

- ordered files
  - suitable when data must be sorted, when doing range selections
- files hashed on some fields
  - records are stored according to a hash function

Record Formats

- fixed-length records

| C_1 | C_2 | | C_3 | C_4 | C_5 | |

L1   L2   L3   L4   L5

Base address (A)   Addr = A + L1 + L2 + L3

- 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
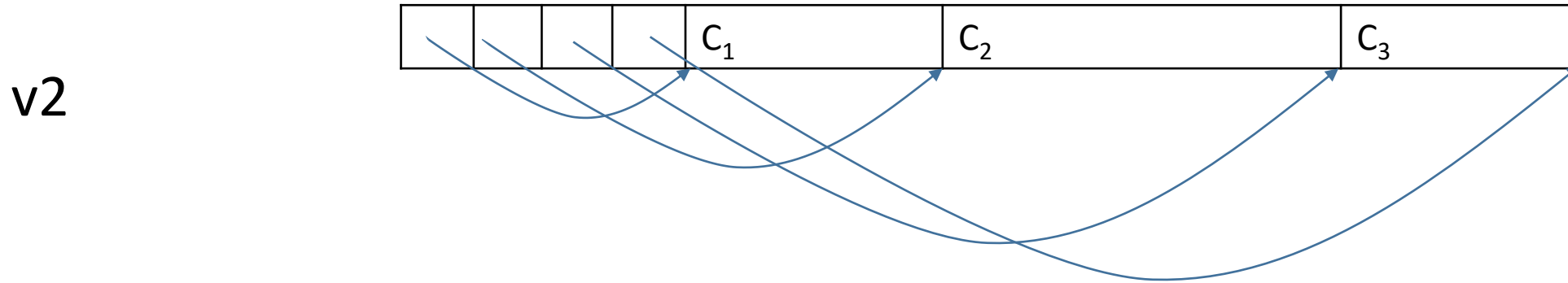 - variable-length fields

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
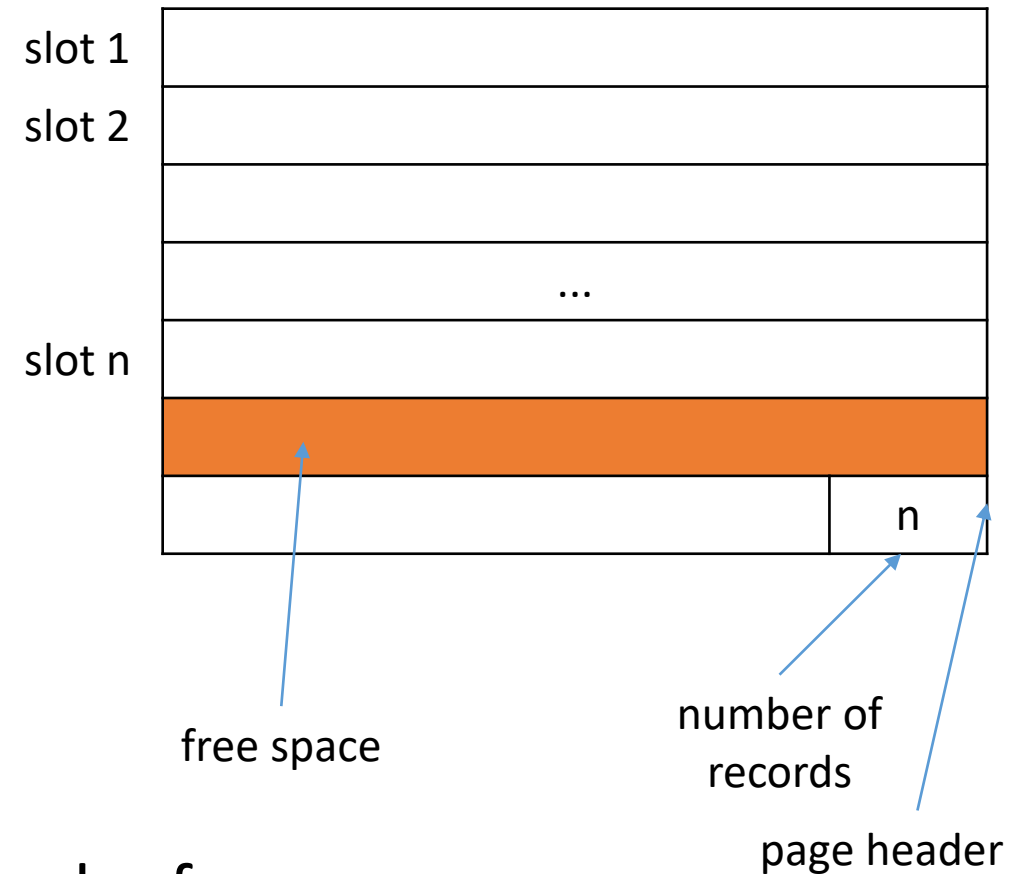
Page Formats

- 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
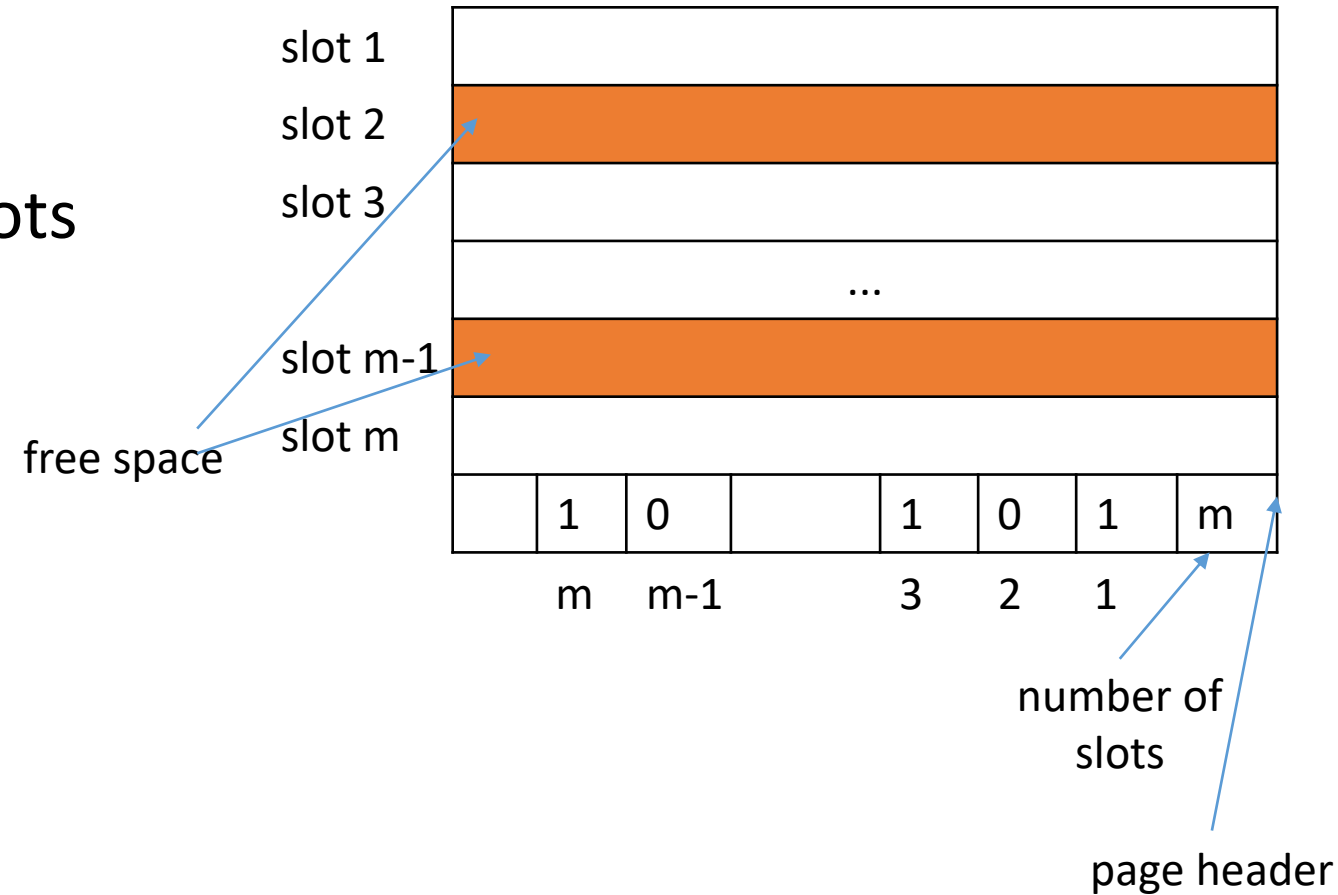   - locating records

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!

slot 1
slot 2
…
slot n

free space

n

number of records

page header

Page Formats

- fixed-length records - v2

- array of bits to monitor available slots

- 1 bit / slot

- deleting a record - turning off the corresponding bit

slot 1
slot 2
slot 3
...
slot m-1
slot m
free space

	1	0		1	0	1	m
m	m-1			3	2	1	

number of slots

page header

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 (e.g., when records need to be kept sorted)

Page Formats
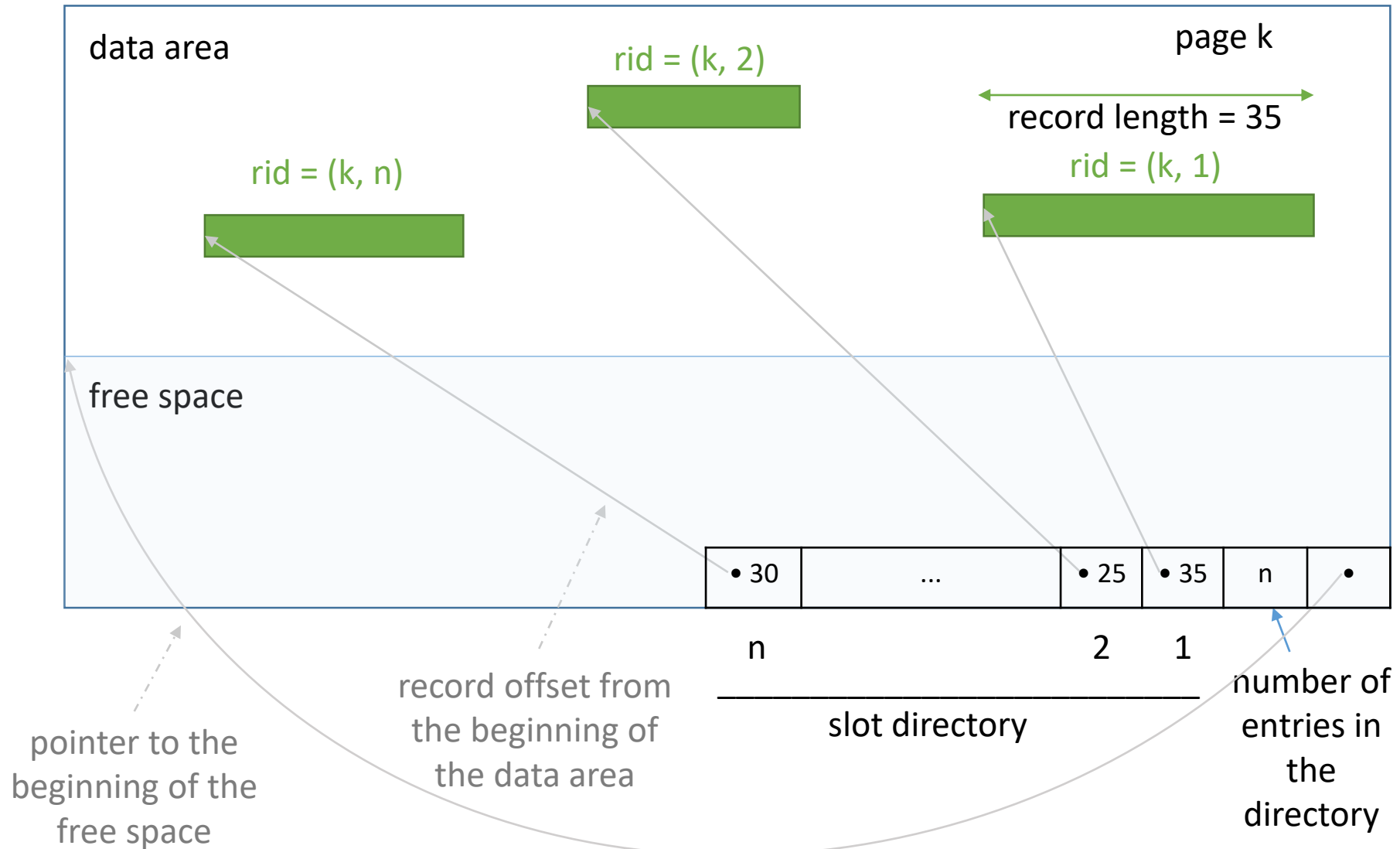
- variable-length records

- motivating example
  - file of students records sorted by name
  - the following operation can be efficiently carried out, given the organization of the file
    - *retrieve all the students in alphabetical order*
  - but the file organization doesn't support the efficient execution of the following operations, both of which would require a scan of the file
    - *retrieve all the students whose age is in a given range*
    - *retrieve all the students who live in Timișoara*

Indexes

- 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
  - enable the retrieval of the records with a given search key value, e.g., <search key, rid>

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 *= 'Timisoara'*
    - follow rids from obtained entries to retrieve records storing data about students who live in Timișoara

Indexes

- an index can improve the efficiency of certain types of queries, not of all queries, e.g., 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
  - structures based on hash functions

Indexes

- 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

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
- [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
- [Si10] SILBERSCHATZ, A., KORTH, H., SUDARSHAN, S., Database System Concepts, McGraw-Hill, 2010