

Databases

Lectures 9-10*

The Physical Structure of Databases. Indexes

*01.12.2022 - National Day

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, $\text{pin_count} = 0$, $\text{dirty} = \text{off}$, $\forall F \in \text{BP}$
- L asks for a page P; the BM:
 1. checks whether page P is in the BP; if so, $\text{pin_count}(F)++$, where F is the frame containing Potherwise:
 - a. BM chooses a frame FR for replacement
 - if the BP contains multiple frames with $\text{pin_count} = 0$, one frame is chosen according to the BM's replacement policy
 - $\text{pin_count}(\text{FR})++$;
 - b. if $\text{dirty}(\text{FR}) = \text{on}$, 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 of Records

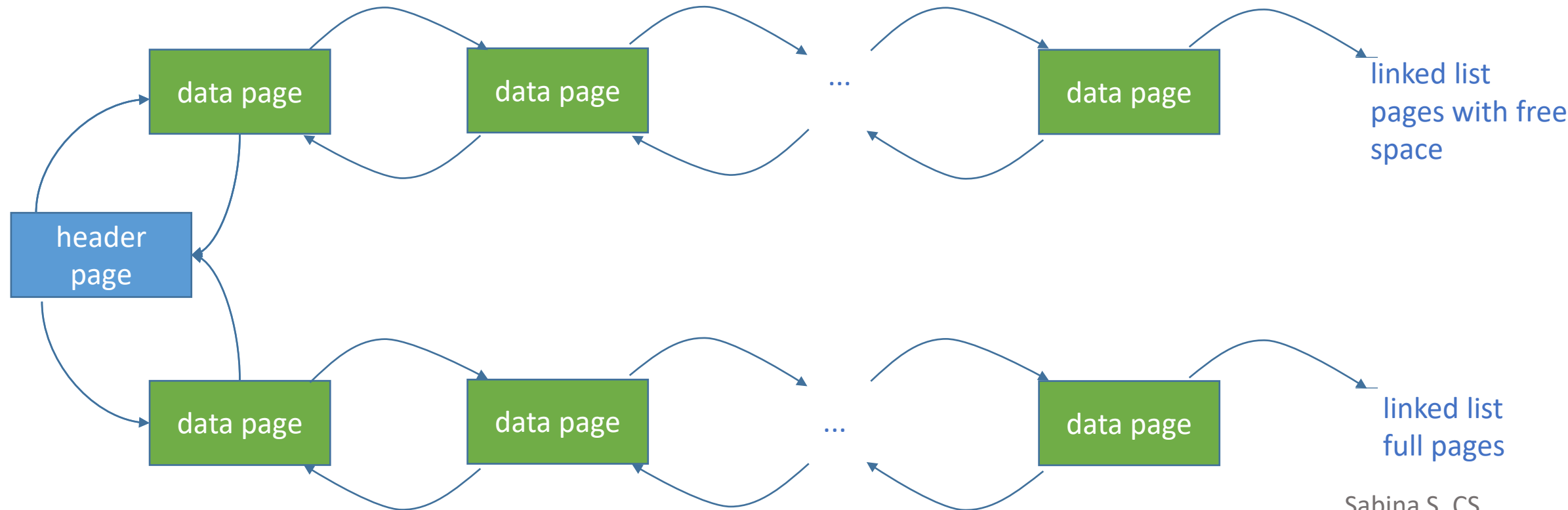
- higher level layers in the DBMS treat pages as collections of records
- file of records
 - collection of records; one or more pages
- different ways to organize a file's collection of pages
- every record has an identifier: the rid
- given the rid of a record, one can identify the page that contains 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 $\langle \text{heap_file_name}, \text{page1_address} \rangle$)
- 2 lists – pages with free space and full pages

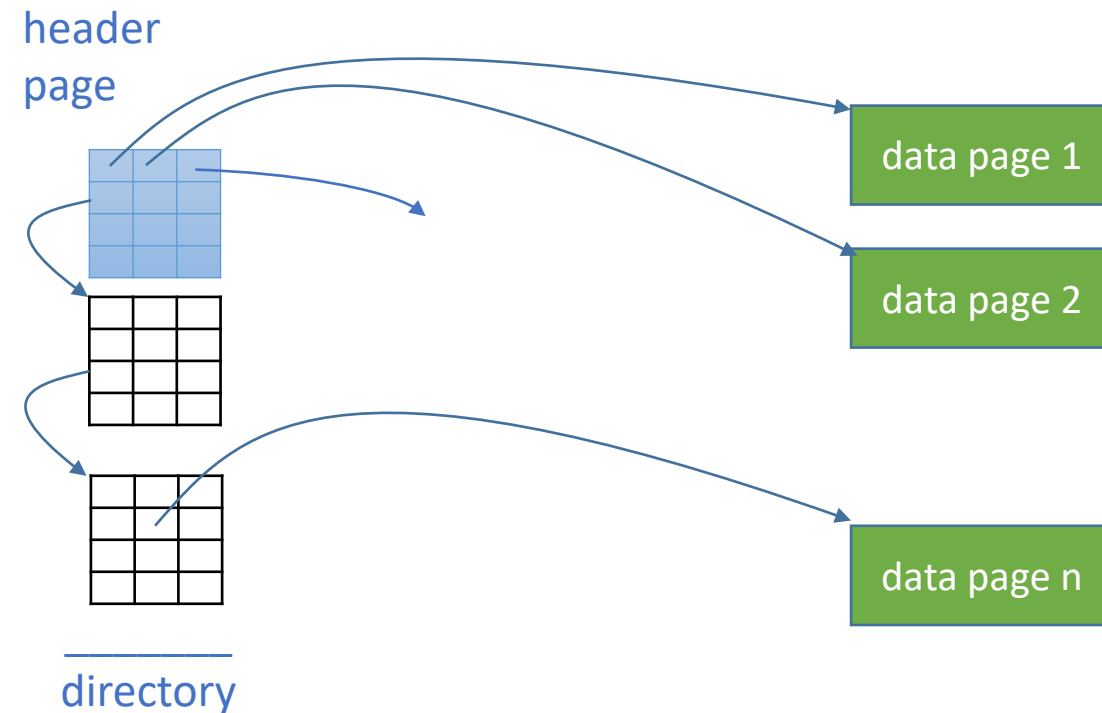


Heap Files - Linked List

- 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

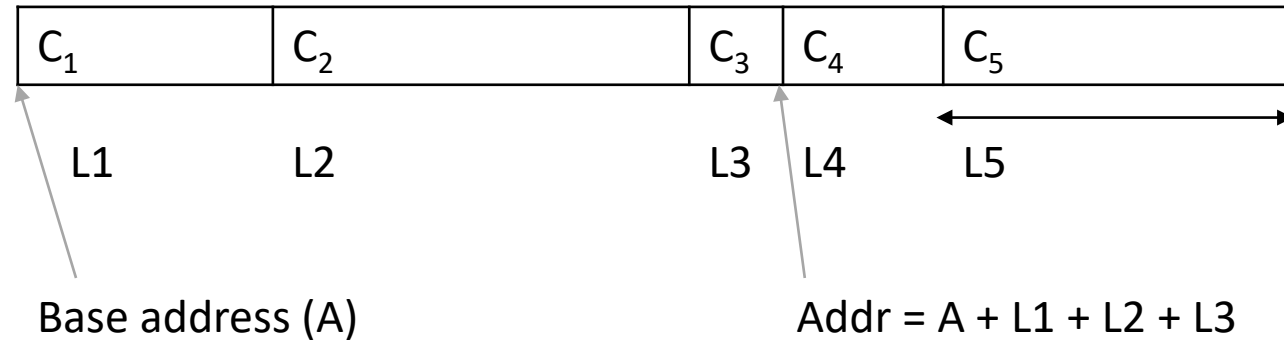


Other File Organizations

- 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

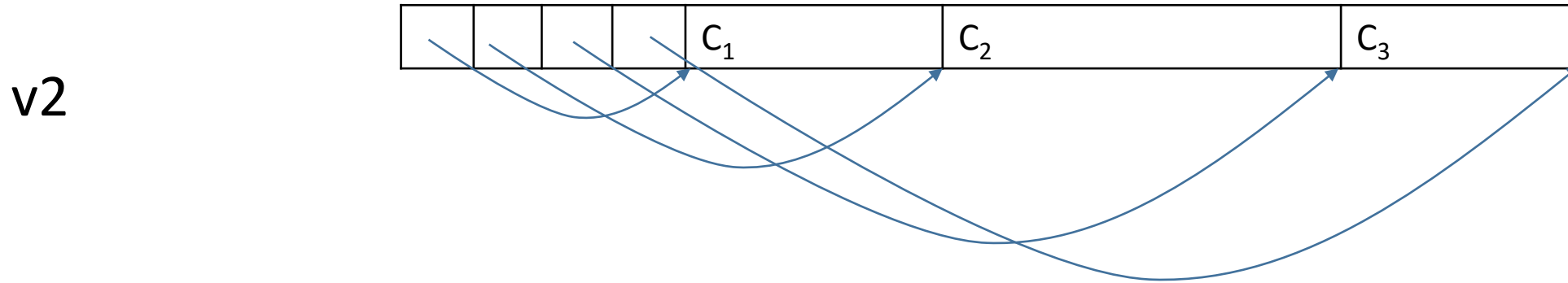
v1

C ₁	\$	C ₂	\$	C ₃	\$
----------------	----	----------------	----	----------------	----

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

Record Formats

- variable-length records



- 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

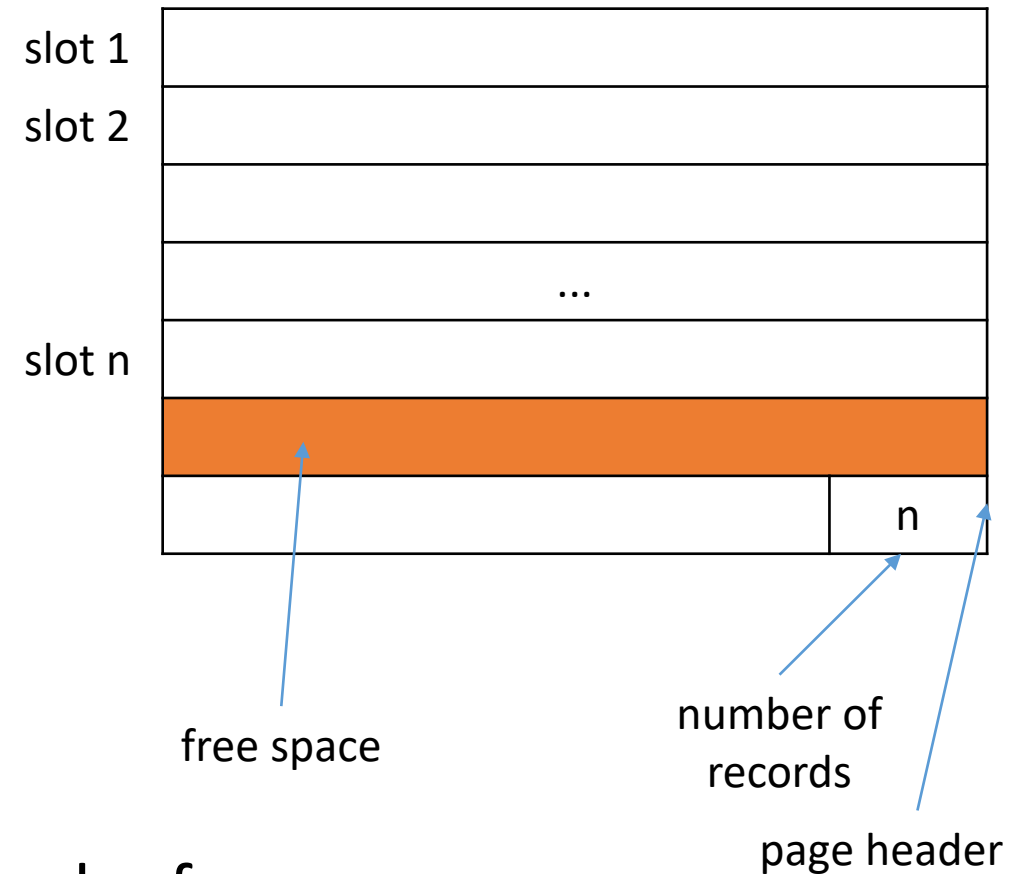
- page
 - collection of slots
 - 1 record / slot
- identifying a record
 - record id (rid): <page id, slot number>
- how to arrange records on pages
- how to manage slots

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

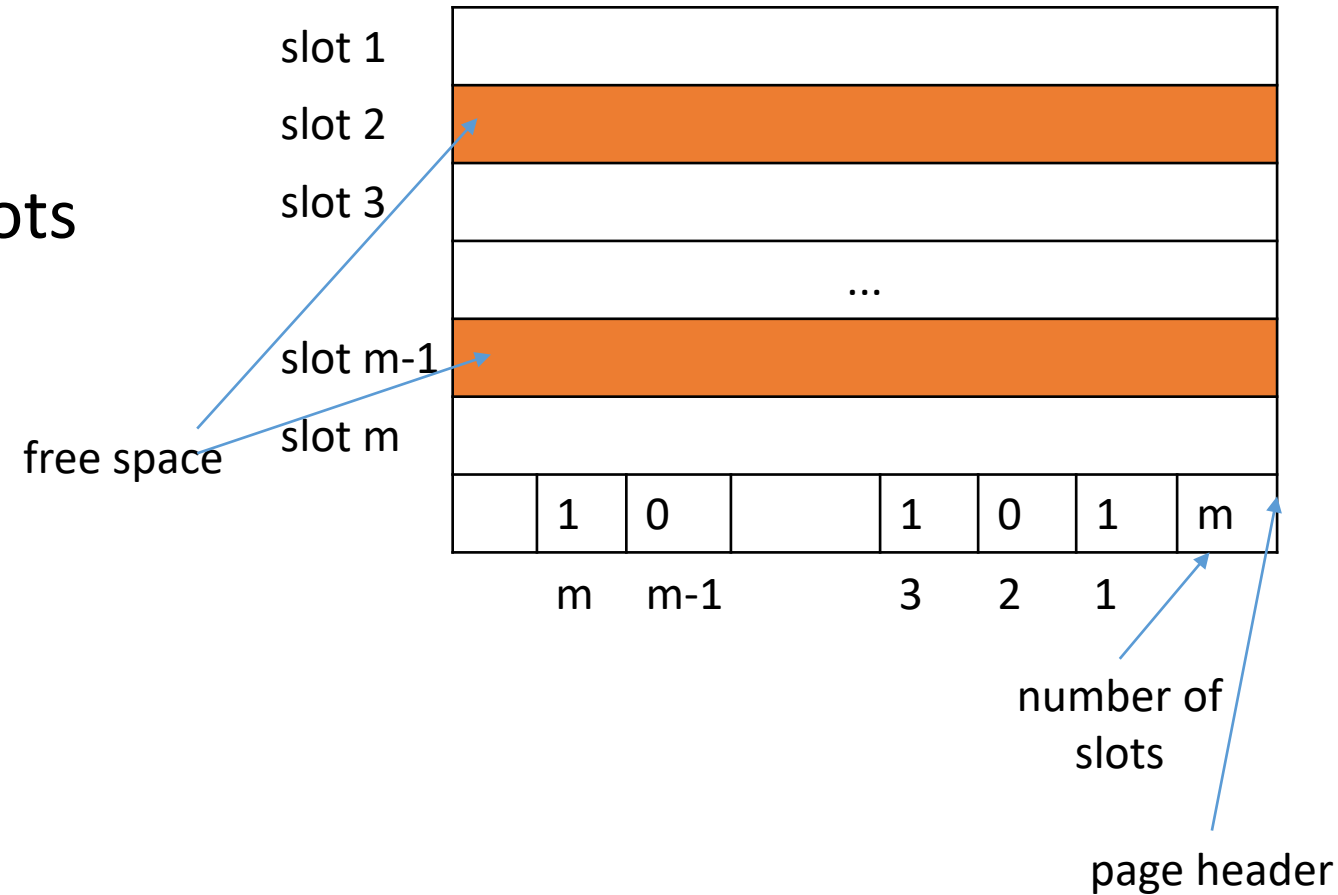
Page Formats

- 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

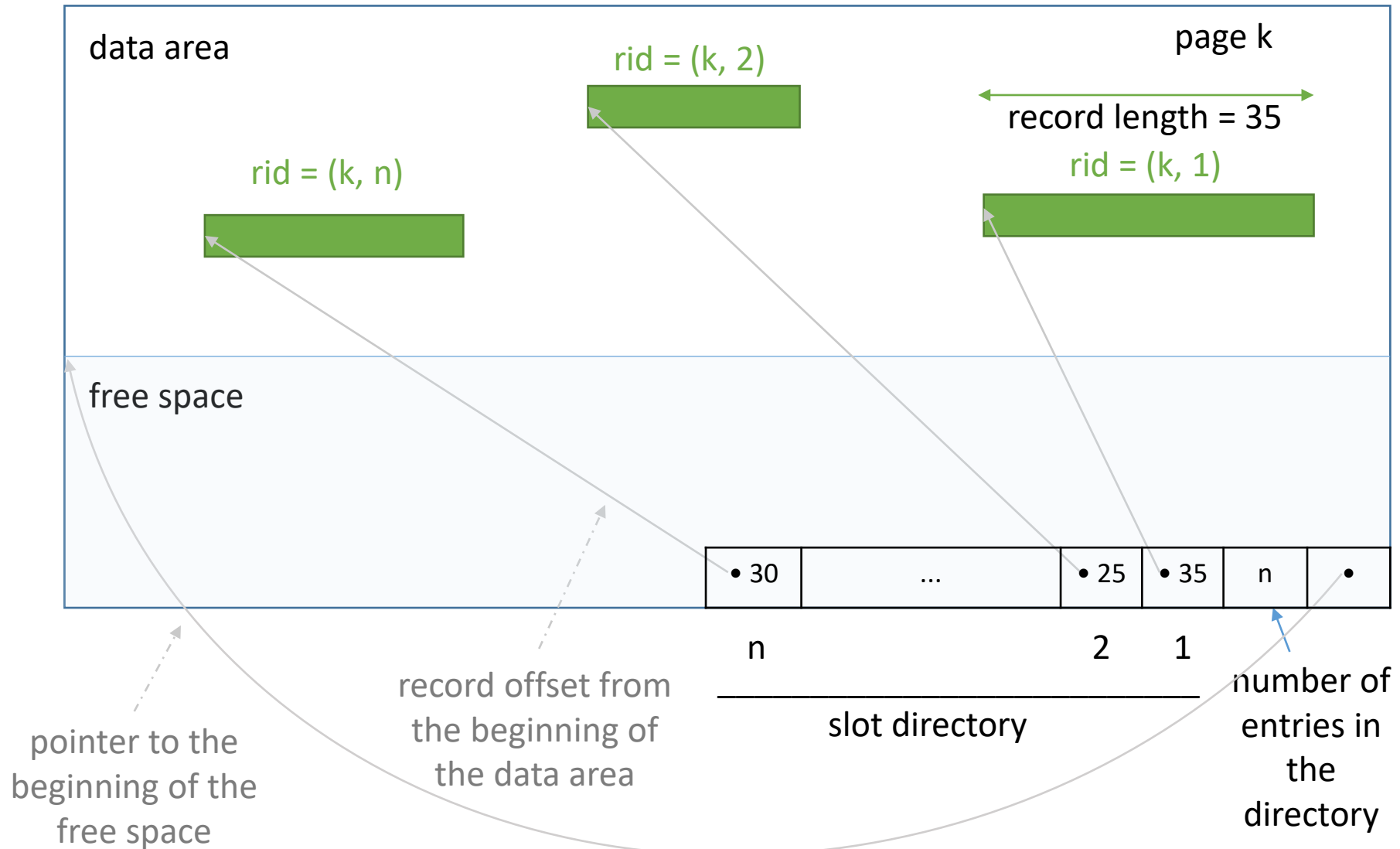


Page Formats

- 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 number 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: $\langle \text{city}, \text{rid} \rangle$, 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 $\text{city} = \text{'Timișoara'}$
 - 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

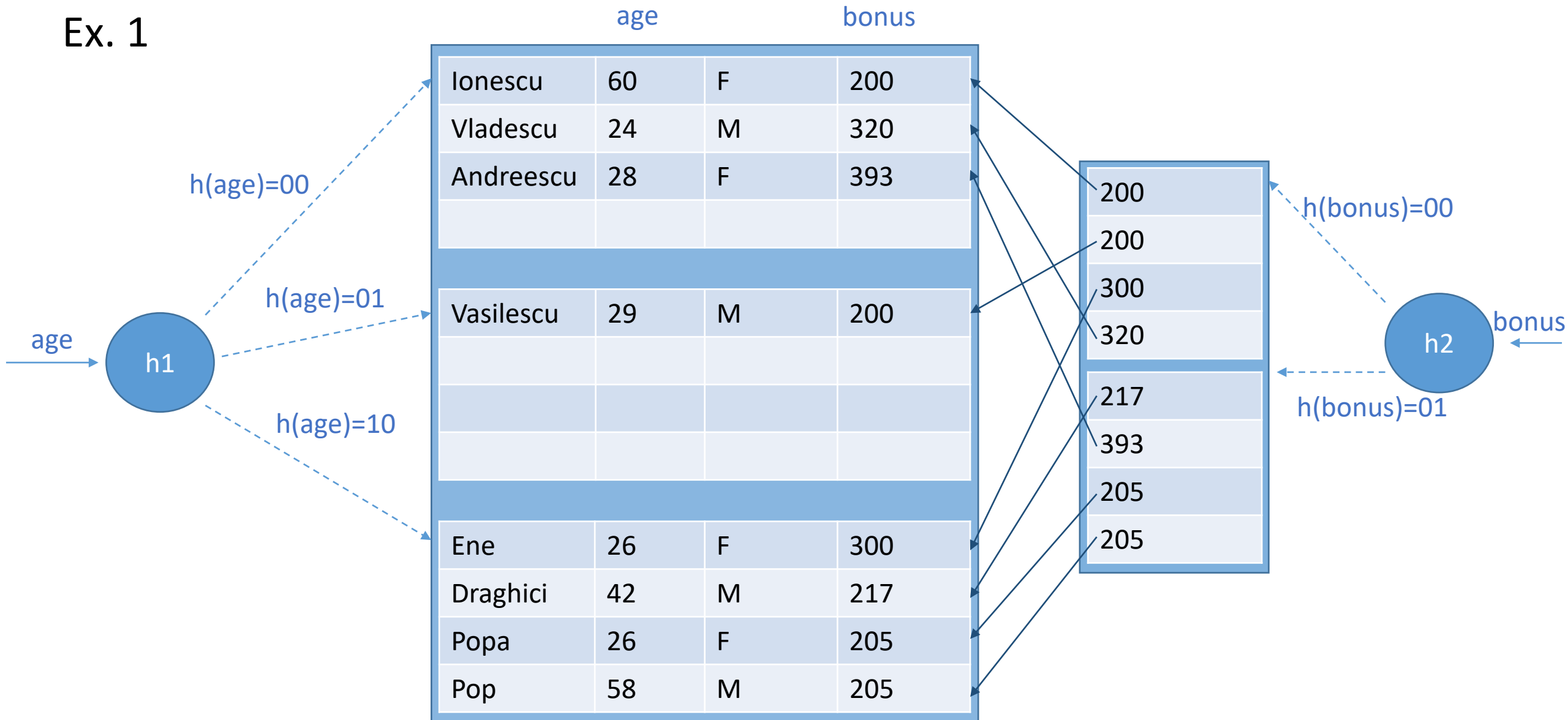
- problems
 - what does a data entry contain?
 - how are the entries of an index organized?
- let k^* be a data entry in an index; the data entry:
 - alternative 1
 - is an actual data record with search key value = k
 - alternative 2
 - is a pair $\langle k, \text{rid} \rangle$ (rid – id of a data record with search key value = k)
 - alternative 3
 - is a pair $\langle k, \text{rid_list} \rangle$ (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

Indexes - Data Entries

Ex. 1



Indexes - Data Entries

Ex. 1

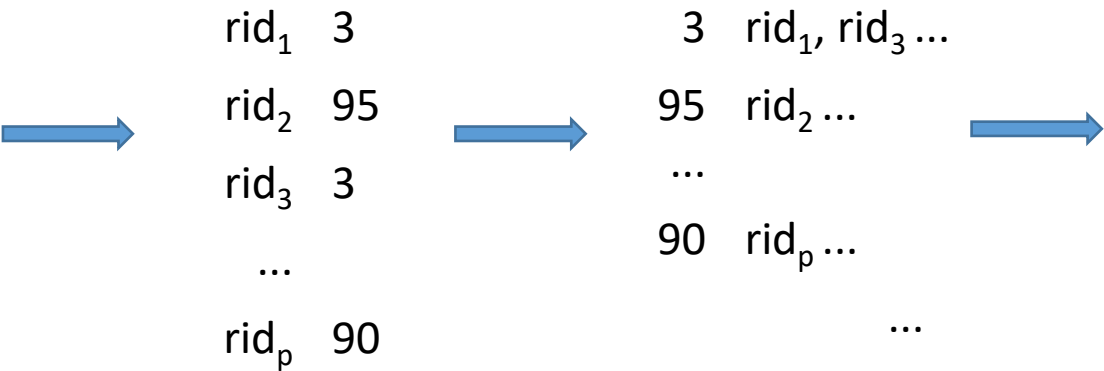
- file with Employee records hashed on *age*
 - record <Ionescu, 60, F, 200>:
 - apply hash function to *age*: convert 60 to its binary representation, take the 2 least significant bits as the bucket identifier for the record
- index file that uses alternative 1 (data entries are the actual data records), search key *age*
- index that uses alternative 2 (data entries have the form <search key, rid>), search key *bonus*
- both indexes use hashing to locate data entries

Indexes - Data Entries

Ex. 2

	Name	Score	Age
rid ₁	Popescu	3	44
rid ₂	Ionescu	95	80
rid ₃	Vladescu	3	45
...		...	
rid _p	Xulescu	90	14

search key



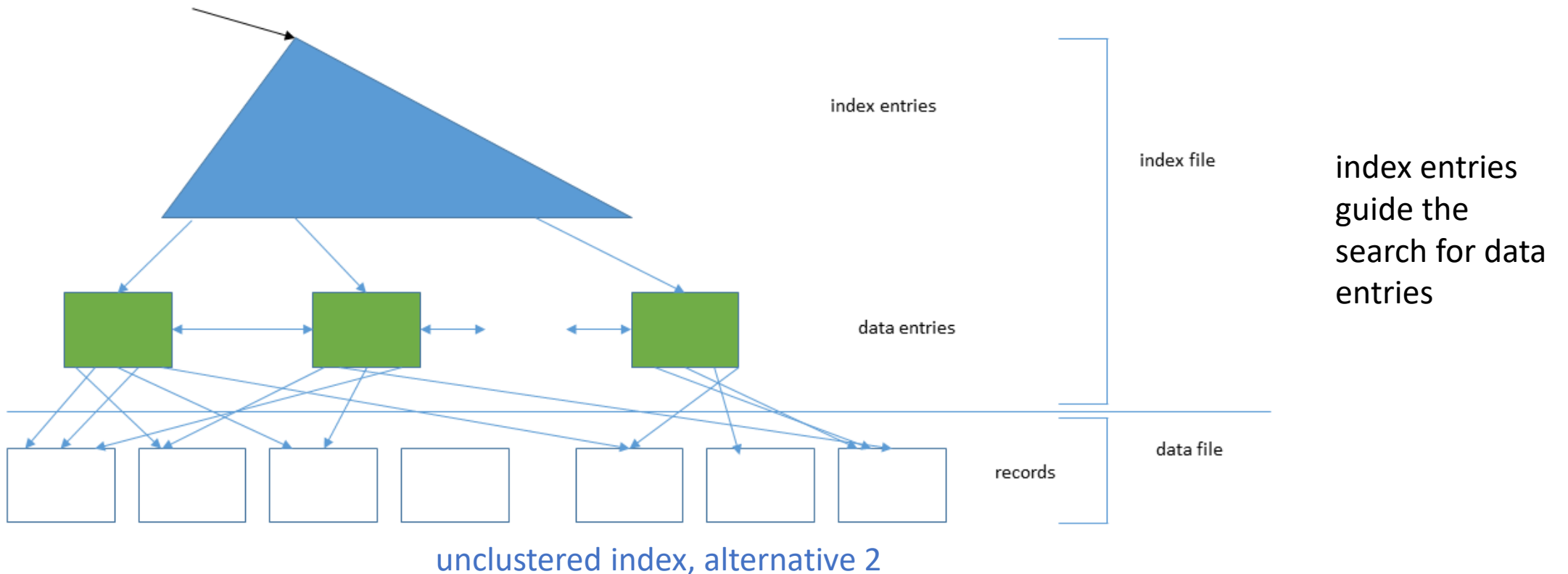
data entry

...	
3	rid ₁ , rid ₃ ...
...	
90	rid _p ...
...	
95	rid ₂ ...
...	

index file

Clustered / Unclustered Indexes

- clustered index: the order of the data records is close to / the same as the order of the data entries
- unclustered index: index that is not clustered



Clustered / Unclustered Indexes

- index that uses alternative 1 - clustered (by definition, since the data entries are the actual data records)
- indexes using alternatives 2 / 3 are clustered only if the data records are ordered on the search key
- in practice:
 - expensive to maintain the sort order for files, so they are rarely kept sorted
 - a clustered index is an index that uses alternative 1 for data entries
 - an index that uses alternative 2 or 3 for data entries is unclustered
- on a collection of records:
 - there can be at most 1 clustered index
 - and several unclustered indexes

Clustered / Unclustered Indexes

- range search query (e.g., *where age between 20 and 30*)
 - cost of using an unclustered index
 - each data entry that meets the condition in the query could contain a rid pointing to a distinct page
 - the number of I/O operations could be equal to the number of data entries that satisfy the query's condition

References

- [Ta13] ȚÂMBULEA, L., Curs Baze de date, Facultatea de Matematică și Informatică, UBB, 2013-2014
- [Ra00] RAMAKRISHNAN, R., GEHRKE, J., Database Management Systems (3rd Edition), McGraw-Hill, 2002
- [Ga08] GARCIA-MOLINA, H., ULLMAN, J., WIDOM, J., Database Systems: The Complete Book (2nd Edition), Pearson Education, 2009
- [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, <http://codex.cs.yale.edu/avi/db-book/>