Review 1-- Storing Data: Disks and Files

Chapter 9 [Sections 9.1-9.7: Ramakrishnan & Gehrke (Text)]

AND

(Chapter 11 [Sections 11.1, 11.3, 11.6, 11.7: Garcia-Molina et al. (R2)]

OR

Chapter 2 [Sections 2.1, 2.3, 2.6, 2.7: Garcia-Molina et al. (R1)])

What you will learn from Review 1

- Disk storage model & parameters
- Buffer management (in a DBMS)
- Record storage & files of records
- Indexing (intro.)
- System catalogs

What you will learn from this lecture

- Disk storage model & parameters
- Buffer management (in a DBMS)
- Record storage & files of records
- Indexing (intro.)
- System catalogs

Disks and Files

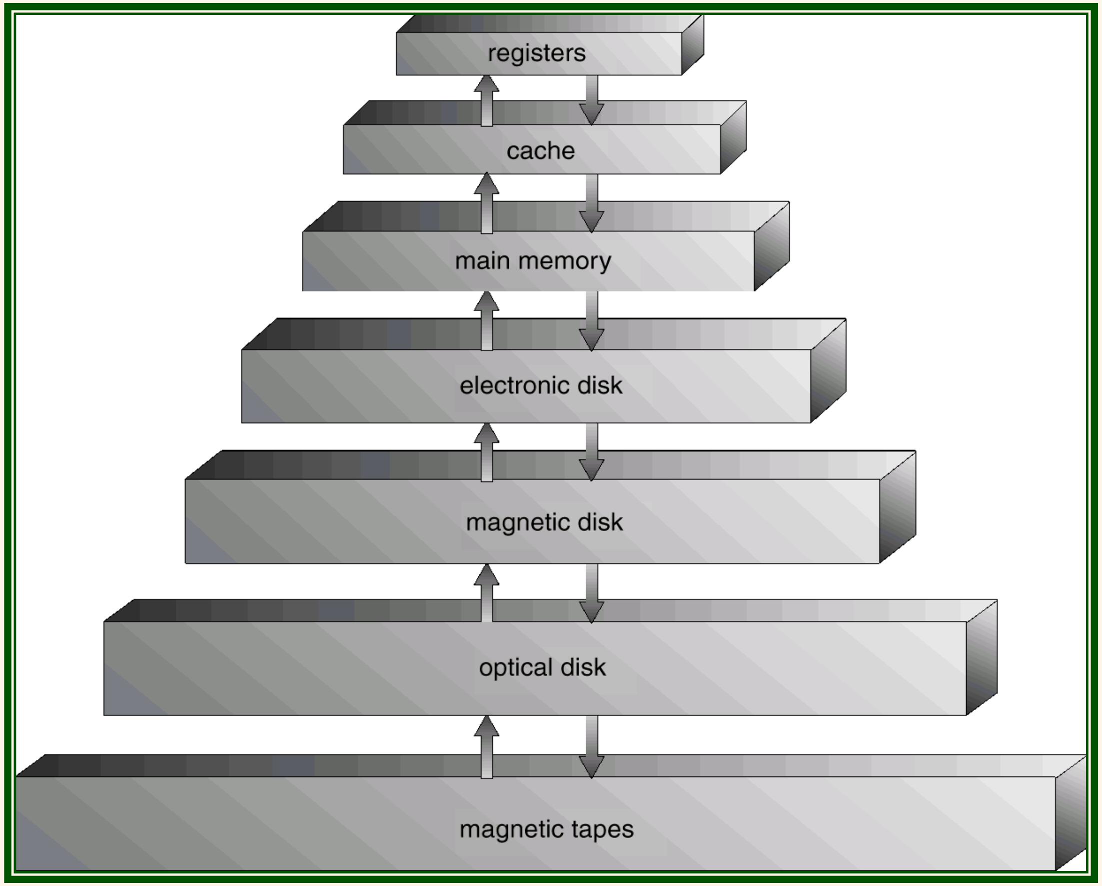
- DBMS stores information on ("hard") disks.
- This has major implications for DBMS design!
  - READ: transfer data from disk to main memory (RAM).
  - WRITE: transfer data from RAM to disk.
  - Both are high-cost operations, relative to in-memory operations, so must be planned carefully!

- Alternative: Flash storage. NAND vs. NOR flash.
- By far hard disks are most widely used for large DBs.
- We will use disks for modeling and costing.

Why Not Store Everything in Main Memory?

- *Main memory is more expensive than disk* for same amount of space.
- *Main memory is volatile.* We want data to be saved between runs. (Obviously!)
- Typical storage hierarchy:
  - Main memory (RAM) for currently used data.
  - Disk for the main database (secondary storage).
  - Tapes for archiving older versions of the data (tertiary storage).

Storage Hierarchy

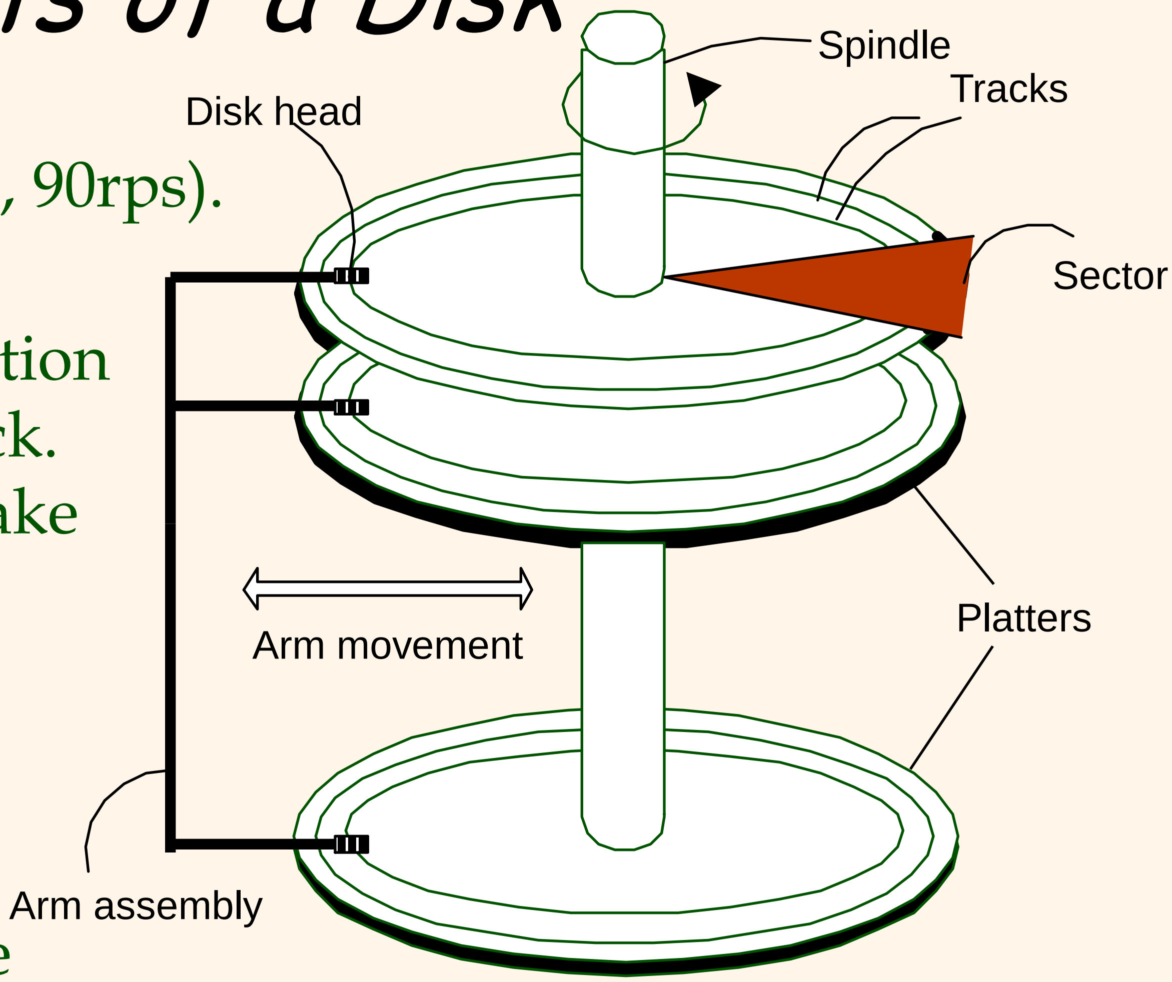


Disks

- Secondary storage device of choice.
- Main advantage over tapes: *random access* vs. *sequential*.
- Data is stored and retrieved in units called *disk blocks* or *pages*.
- Unlike RAM, time to retrieve a disk page varies depending upon location on disk.
  - Therefore, relative placement of pages on disk (often called clustering) has major impact on DBMS performance!

Components of a Disk

- The platters spin (say, 90rps).
- The arm assembly is moved in or out to position a head on a desired track. Tracks under heads make a *cylinder* (imaginary!).
- Only one head reads/writes at any one time.
- *Block size* is a multiple of *sector size* (which is fixed). Block size – design choice.



Accessing a Disk Page

- Time to access (read/write) a disk block:
  - *seek time* (moving arms to position disk head on track)
  - *rotational delay* (waiting for block to rotate under head)
  - *transfer time* (actually moving data between disk surface & RAM)
- Seek time and rotational delay dominate.
  - Seek time varies from about 1 to 20msec
  - Rotational delay varies from 0 to 10msec
  - Transfer rate is about 1msec per 4KB page
- Key to lower I/O cost: reduce seek/rotation delays! Hardware vs. software solutions?

Arranging Pages on Disk

- *`Next'* block concept:
  - blocks on same track, followed by
  - blocks on same cylinder, followed by
  - blocks on adjacent cylinder

- Blocks in a file should be arranged sequentially on disk (by `next'), to minimize seek and rotational delay.

- For a sequential scan, *pre-fetching* several pages at a time is a big win!

Typical Example

Megatron 747: disk rpm = 4000.

Block (Page) size = 4096 bytes.

8 platters of 2 surfaces each.

2^13 = 8192 cylinders.

Average # sectors/track = 2^8 = 256.

#bytes/sector = 2^9 = 512.

Moving head assembly between cylinders = 1ms (setup) + 1 ms/500 cylinders.

Typical Example (contd.)

- What is the max/avg seek time?

- What is the max/avg rotational latency time?

- What is the transfer time (i.e., time to read/write a block)?
  -

Remarks

- Disk vs. memory: orders of magnitude difference in speeds (msec vs. nanosec).
- Disk – much cheaper.
- Memory getting faster and more affordable.
- But, data sometimes accumulates much faster!
- What is the diff. b/w random read and sequential read?
  -
  -
  -
  -

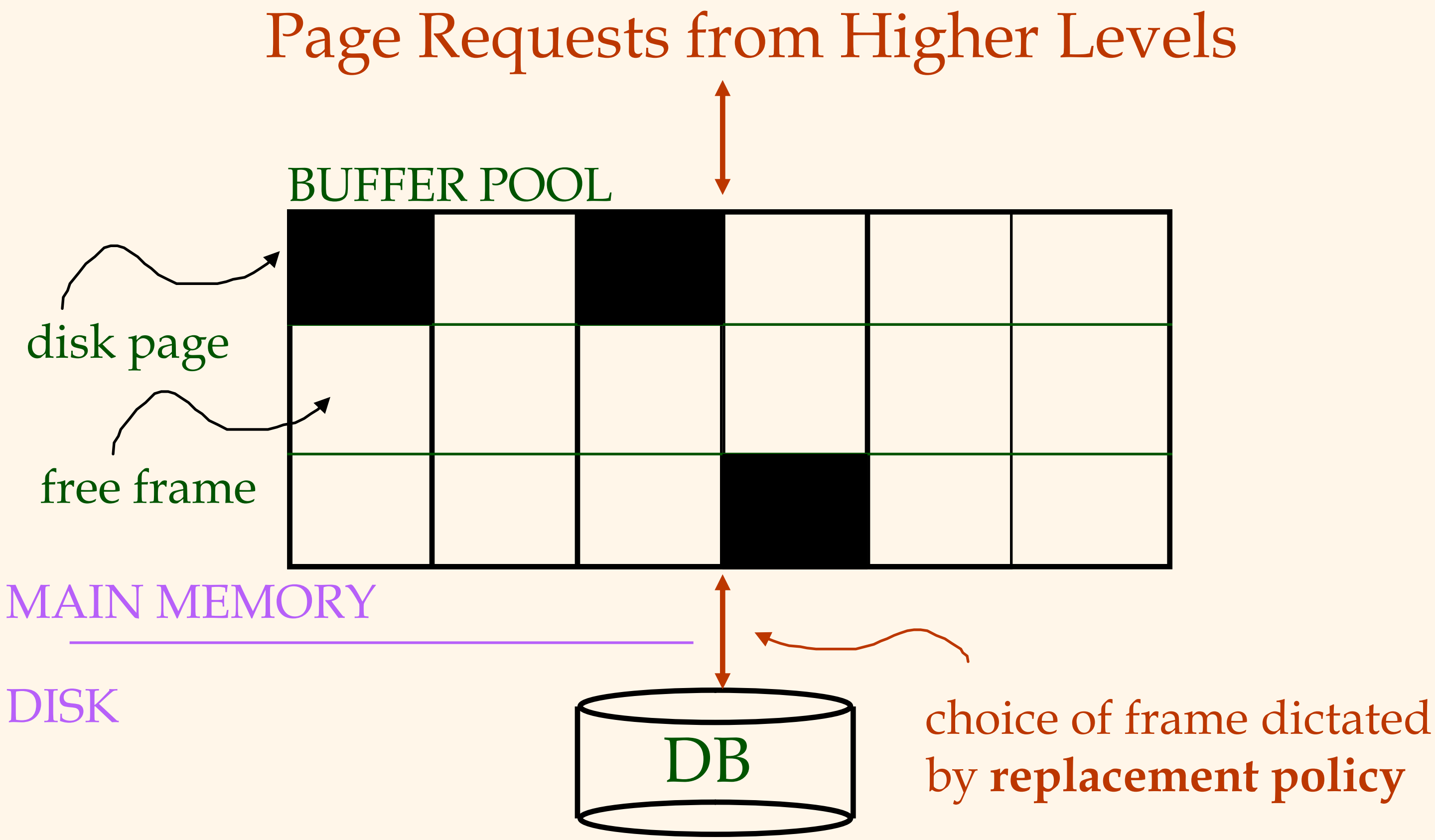
What you will learn from this lecture

- Disk storage model & parameters
- Buffer management (in a DBMS)
- Record storage & files of records
- Indexing (intro.)
- System catalogs

Disk Space Management

- Lowest layer of DBMS software manages space on disk.
- Higher levels call upon this layer to:
  - allocate/de-allocate a page
  - read/write a page
- Request for a *sequence* of pages must be satisfied by allocating the pages sequentially on disk! Higher levels don't need to know how this is done, or how free space is managed.

Buffer Management in a DBMS



- *Data must be in RAM for DBMS to operate on it!*
- *Table of <frame#, pageid> pairs is maintained.*

More on Buffer Management

- Page in pool may be requested many times,
  - a *pin count* is used. Every request increments it. A page is a candidate for replacement iff *pin count* = 0.
- Requestor of page must unpin it (i.e., ↓its pin_count) once it's done with that page, and indicate whether page has been modified:
  - *dirty* bit is used for this.
- CC & recovery may entail additional I/O when a frame is chosen for replacement. (*Write-Ahead Log* protocol.)

When a Page is Requested ...

- If requested page in pool, return frame no.
- If requested page not in pool:
  - Choose a frame for *replacement* (first time, it's not a replacement: why?)
  - *Replacement policy may be more sophisticated than LRU*
  - If frame is dirty, write it to disk
  - Read requested page into chosen frame
- *Pin* the page (i.e., ↑its pin_count) and return its address (i.e., frame no.).

☒ *If requests can be predicted (e.g., sequential scans) pages can be <u>pre-fetched</u> several pages at a time!*

Buffer Replacement Policy

- Frame is chosen for replacement by a *replacement policy:*
  - Least-recently-used (LRU), Clock, MRU etc.
- Policy can have big impact on # of I/O's; depends on the *access pattern.*
- *Sequential flooding*: Nasty situation caused by LRU + repeated sequential scans.
  - # buffer frames < # pages in file means each page request causes an I/O. MRU much better in this situation (but not in all situations, of course).

DBMS vs. OS File System

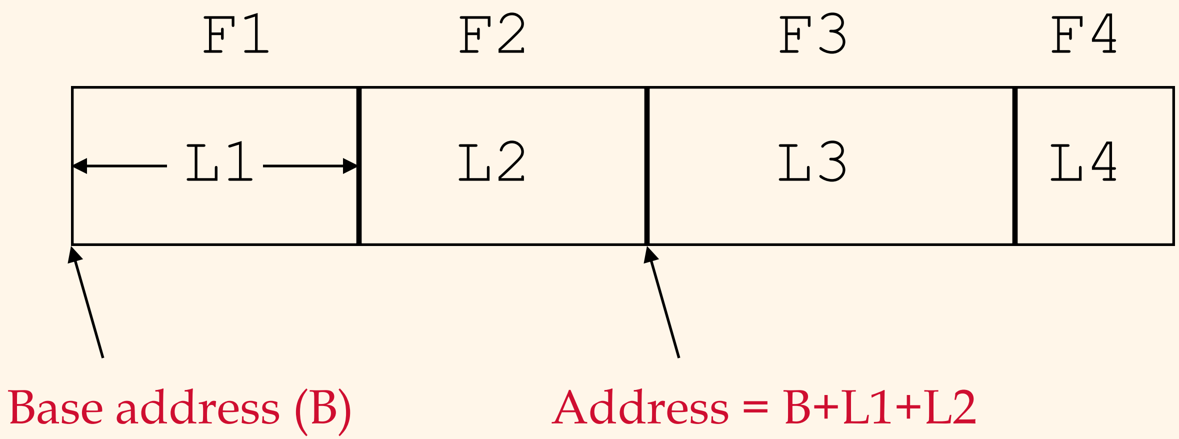
OS does disk space & buffer mgmt: why not let OS manage these tasks?

- Differences in OS support: portability issues
- Some limitations, e.g., files can't span disks.
- Buffer management in DBMS requires ability to:
  - pin a page in buffer pool, force a page to disk (important for implementing CC & recovery),
  - adjust *replacement policy,* and pre-fetch pages based on access patterns in typical DB operations.
- The replacement policy may be *semantic*: frame chosen for replacement may depend on what is known about access pattern.

What you will learn from this lecture

- Disk storage model & parameters
- Buffer management (in a DBMS)
- Record storage & files of records
- Indexing (intro.)
- System catalogs

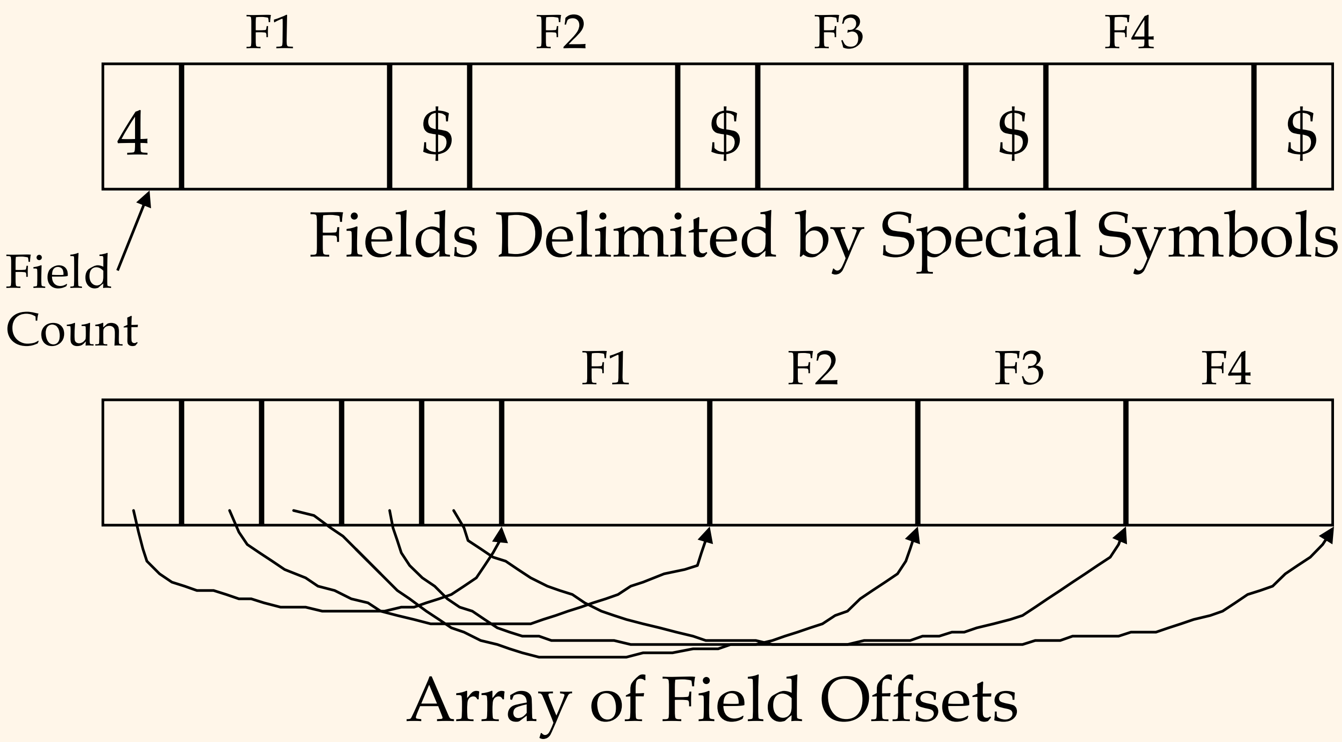
Record Formats: Fixed Length



- Information about field types same for all records in a file; stored in *system catalogs.*
- Naïve: Finding *i'th* field requires scan of record.

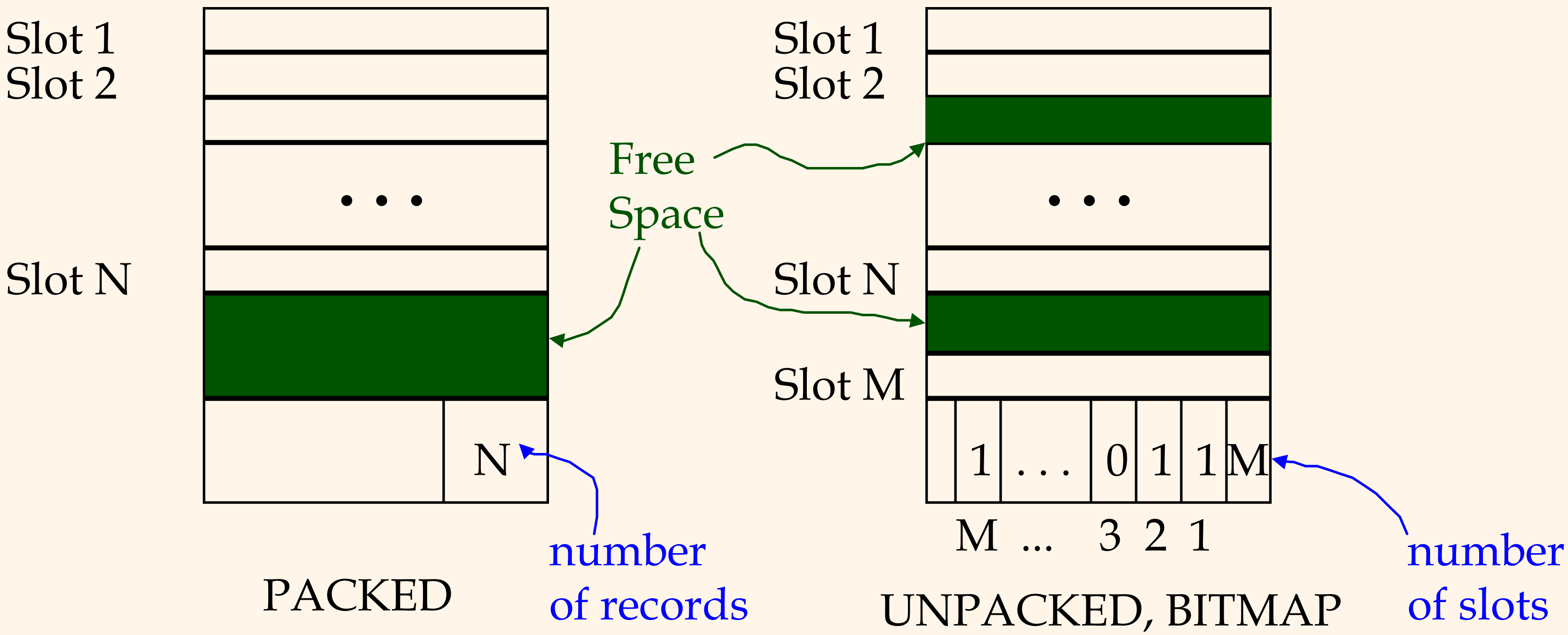
Record Formats: Variable Length

- Two alternative formats (# fields is fixed):



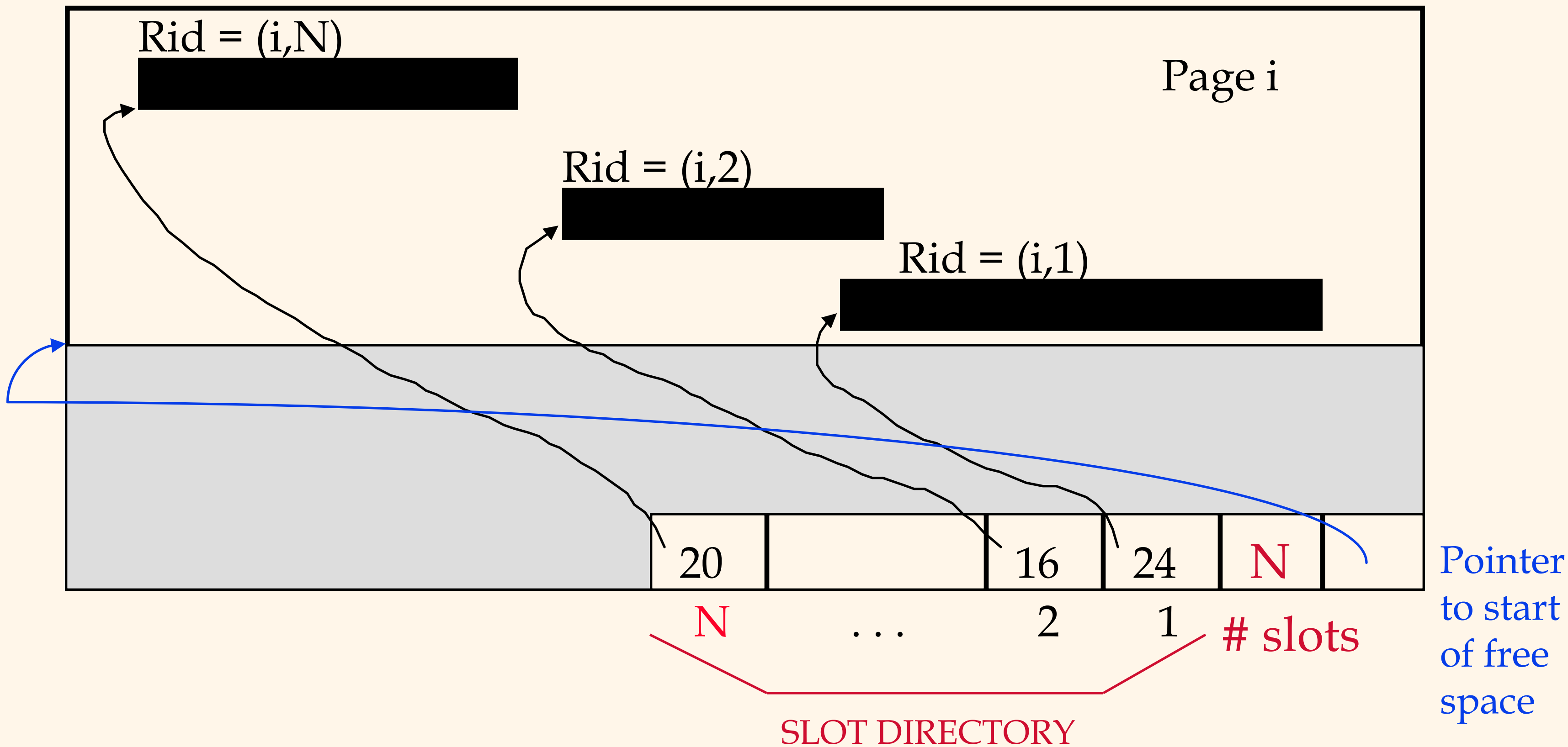
☒ Second offers direct access to i'th field, efficient storage of *nulls* (special *don't know* value); small directory overhead.

Page Formats: Fixed Length Records



- Record id = <page id, slot #>. In first alternative, moving records for free space management changes rid; may not be acceptable.

Page Formats: Variable Length Records



- *Can move records on page without changing rid; so, attractive for fixed-length records too.*

Files of Records

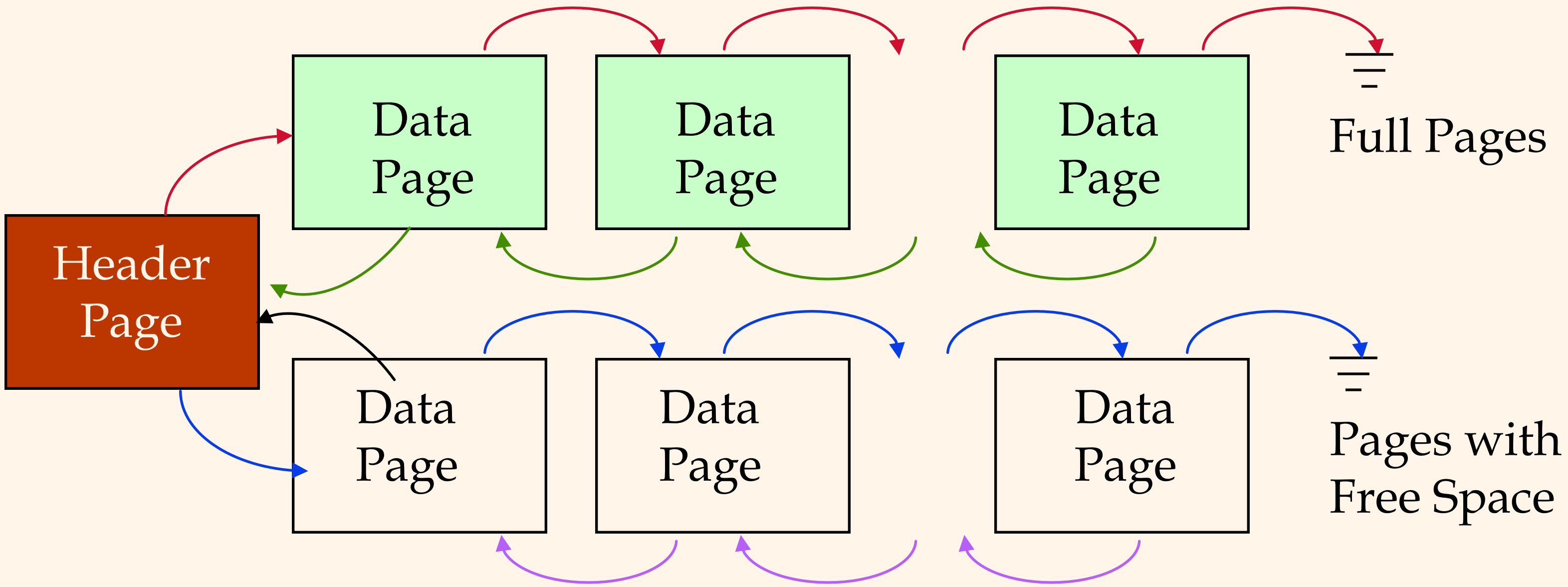
- Page or block is OK when doing I/O, but higher levels of DBMS operate on *records*, and *files of records*.

- FILE: A collection of pages, each containing a collection of records. Must support:
  - insert/delete/modify record
  - read a particular record (specified using *record id*)
  - scan all records (possibly with some conditions on the records to be retrieved): e.g., find all movies with rating >= 4.

Unordered (Heap) Files

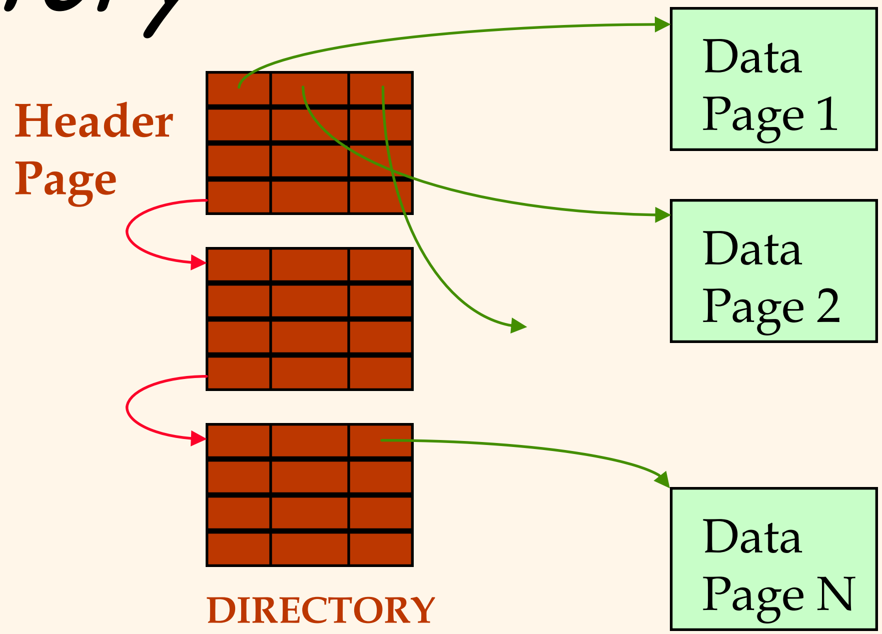
- Simplest file structure contains records in no particular order.
- As file grows and shrinks, disk pages are allocated and de-allocated.
- To support record level operations, we must:
  - keep track of the *pages* in a file
  - keep track of *free space* on pages
  - keep track of the *records* on a page
- There are many alternatives for keeping track of this.

Heap File Implemented as a List



- The header page id and Heap file name must be stored someplace.
- Each page contains 2 `pointers' plus data.

Heap File Using a Page Directory



- The entry for a page can include the number of free bytes on the page.
- The directory is itself a collection of pages; linked list implementation is just one alternative.
  - *Much smaller than linked list of all HF pages!*

What you will learn from this lecture

- Disk storage model & parameters
- Buffer management (in a DBMS)
- Record storage & files of records
- Indexing (intro.)
- System catalogs

Indexes

- A Heap file allows us to retrieve records:
  - by specifying the *rid*, or
  - by scanning all records sequentially
- Sometimes, we want to retrieve records by specifying the *values in one or more fields*, e.g.,
  - Find all users who rated "One Missed Call".
  - Find all users who rated "Sweeny Todd" at 8 or more.
- Indexes are file structures that enable us to answer such value-based queries efficiently.
- Index = primitive value-based query results materialized, i.e., pre-computed, crudely speaking.

What you will learn from this lecture

- Disk storage model & parameters
- Buffer management (in a DBMS)
- Record storage & files of records
- Indexing (intro.)
- System catalogs

System Catalogs

- Store meta-data.
- For each index:
  - structure (e.g., B+ tree) and search key fields
- For each relation:
  - name, file name, file structure (e.g., Heap file)
  - attribute name and type, for each attribute
  - index name, for each index
  - integrity constraints
- For each view:
  - view name and definition
- Plus statistics, authorization, buffer pool size, etc.

✉ *Catalogs are themselves stored as relations!*

Attr_Cat(attr_name, rel_name, type, position)

Students(sid,name,login,age,gpa)

attr_name	rel_name	type	position
attr_name	Attribute_Cat	string	1
rel_name	Attribute_Cat	string	2
type	Attribute_Cat	string	3
position	Attribute_Cat	integer	4
sid	Students	string	1
name	Students	string	2
login	Students	string	3
age	Students	integer	4
gpa	Students	real	5
fid	Faculty	string	1
fname	Faculty	string	2
sal	Faculty	real	3

Faculty(fid,fname,sal)

Summary

- Disks provide cheap, non-volatile storage.
  - Random access, but cost of accessing a page depends on location of page on disk; important to arrange data sequentially to minimize *seek* and *rotation* delays.
- Buffer manager brings pages into RAM.
  - Page stays in RAM until released by requestor.
  - Written to disk if frame chosen for replacement & frame dirty (which is some time after requestor releases the page).
  - Choice of frame to replace based on *replacement policy.*
  - Tries to *pre-fetch* several pages at a time.

Summary (Contd.)

- DBMS vs. OS File Support
  - DBMS needs features not found in many OS's, e.g., forcing a page to disk, controlling the order of page writes to disk, files spanning disks, ability to control pre-fetching and page replacement policy based on predictable access patterns, etc.
- Variable length record format with field offset directory offers support for direct access to i'th field and null values.
- Slotted page format supports variable length records and allows records to move on page, without causing *rid* to change.

Summary (Contd.)

- File layer keeps track of pages in a file, and supports abstraction of a collection of records.
  - Pages with free space identified using linked list or directory structure (similar to how pages in file are kept track of).
- Indexes support efficient retrieval of records based on the values in some fields.
- Catalog relations store information about relations, indexes and views. (*Information that is common to all records in a given collection.*)
- Meta-data plays key role when trying to integrate info. in multiple databases.