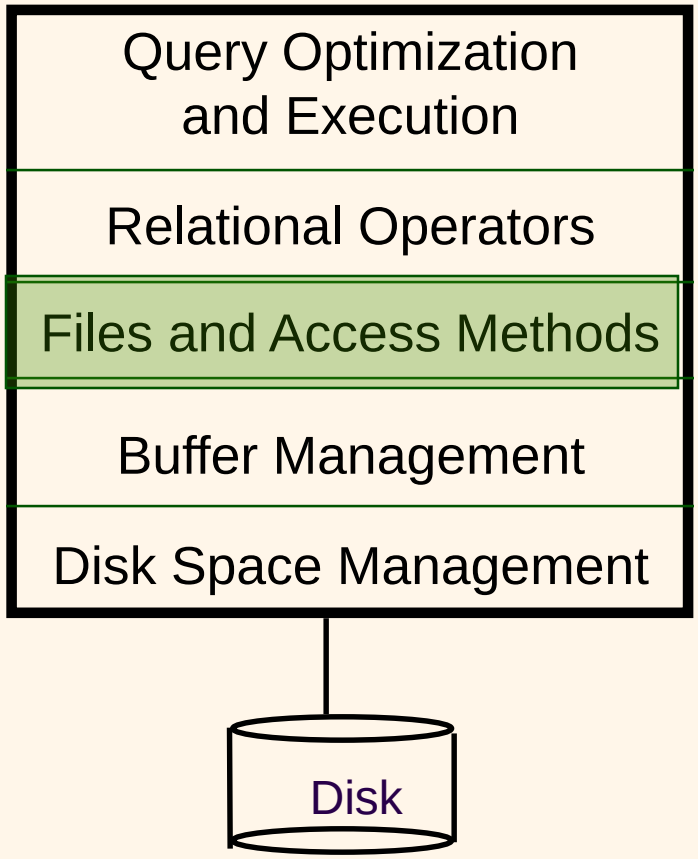
Overview of Storage and Indexing
Storing Data: Disks and Files

Chapters 8-9

Overview of Storage and Indexing

- What would be a good way to store the records of a table of (id,name,age,salary)
- What about if we always want to retrieve records based on the age / salary..??
- Pages: The unit of information read from or written to disk. It is a DBMS parameter, typical values 4KB, 8KB

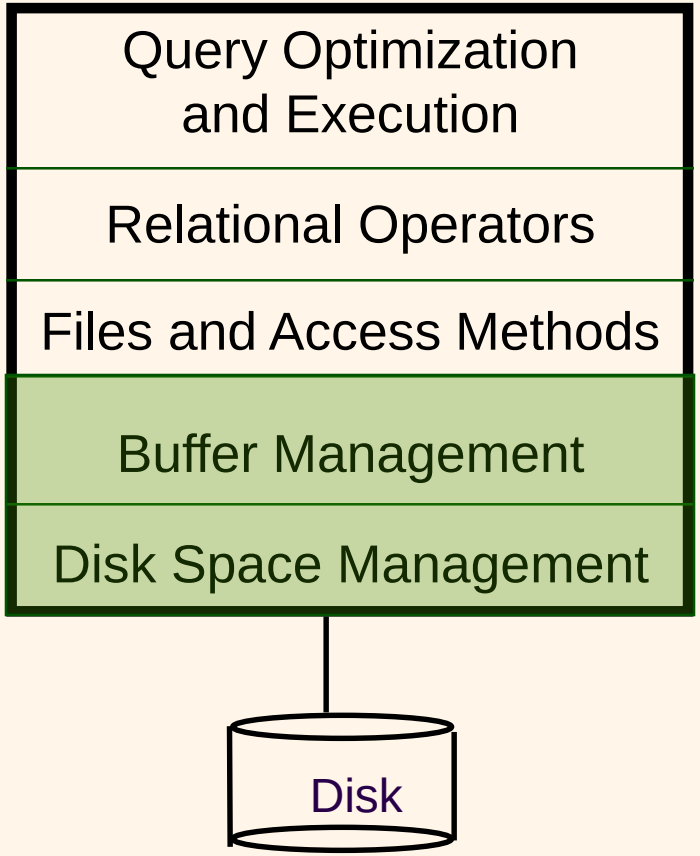


Data on External Storage

- Disks: Can retrieve random page at fixed cost
  - But reading several consecutive pages is much cheaper than reading them in random order
- Tapes: Can only read pages in sequence
  - Cheaper than disks; used for archival storage
- File organization: Method of arranging a file of records on external storage.
  - Record id (rid) is sufficient to physically locate record. An rid can identify the disk address of the page that contains the record.
  - Indexes are data structures that allow us to find the record ids of records with given values in index search key fields

Data on External Storage (Architecture)

- Buffer manager:
  - Reads data into memory for processing and write to disk for persistent storage
  - When the "file and access methods" layer needs to process a page, it passes the page's rid to the buffer manager that will fetch it in memory if it is not there



- Disk space manager
  - Allocates disk pages upon request from the file layer
  - If a page is freed by the file layer, the space manager tracks it and reuse it later if the file later requests a new page

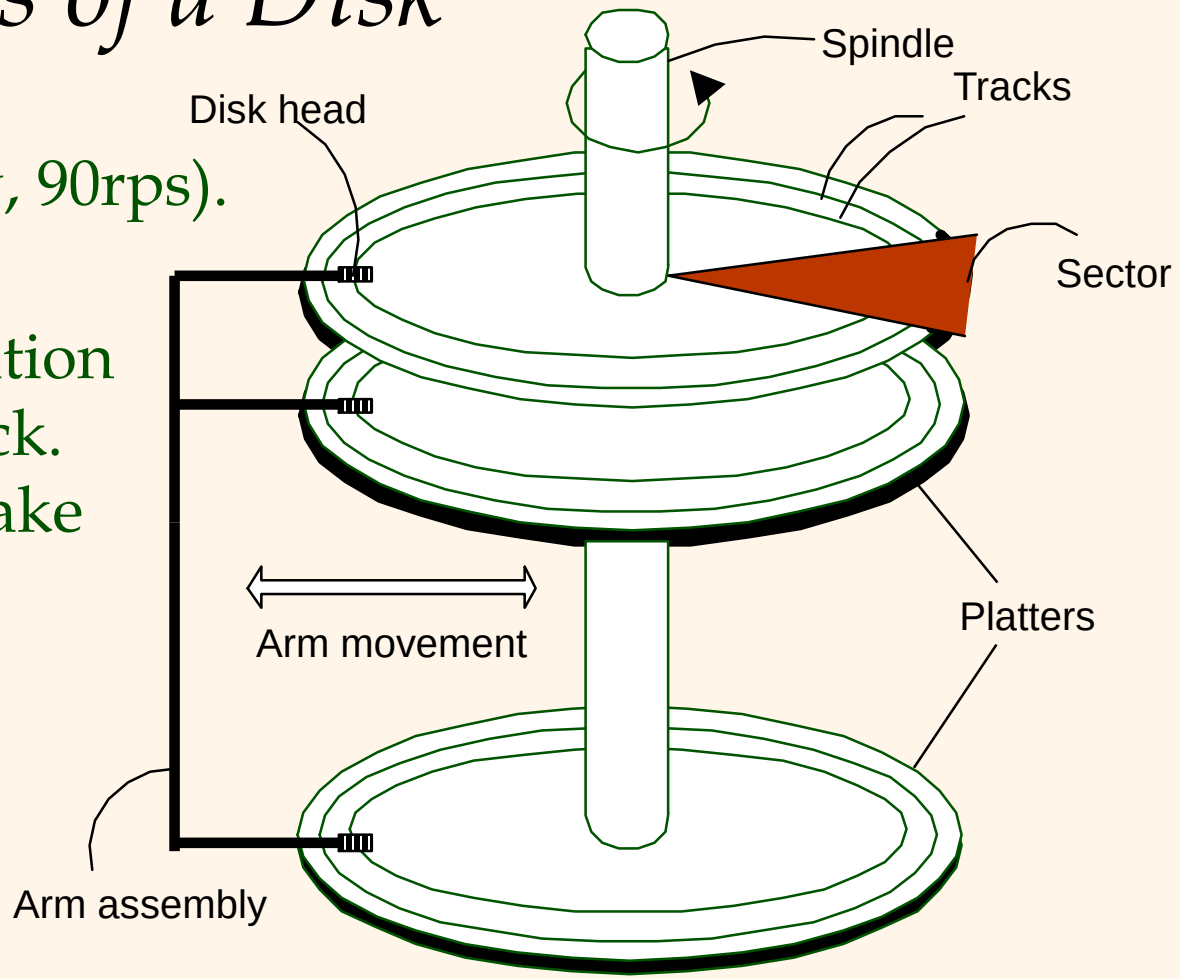
Disks and Files

- 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!

- Why Not Store Everything in Main Memory?
  - *Costs too much.* Disks are much cheaper than memory.
  - *Main memory is volatile.* We want data to be saved between runs. (Obviously!)

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.



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 to/from disk surface)

- Seek time and rotational delay dominate.

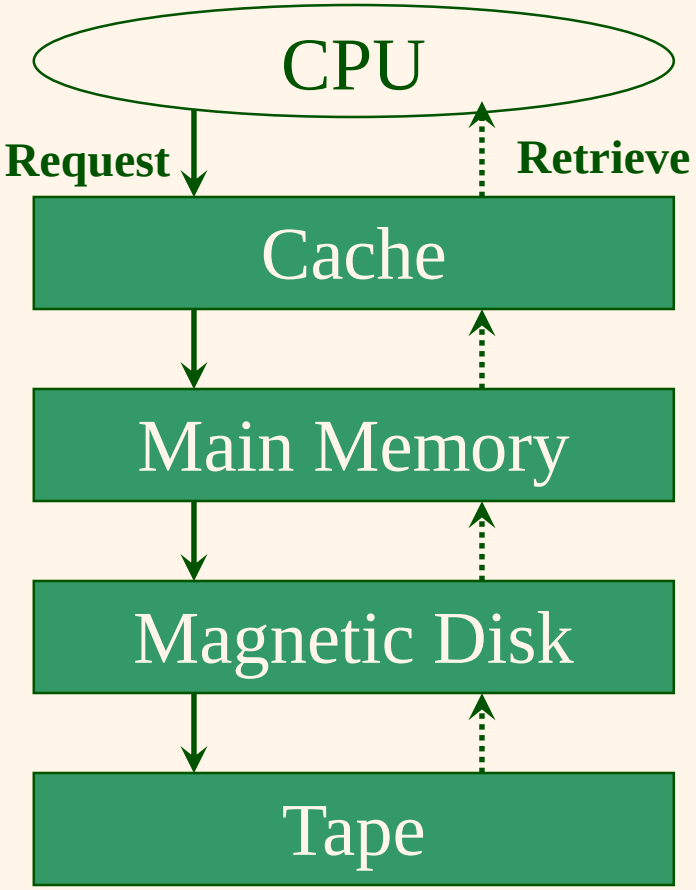
- Key to lower I/O cost: reduce seek/rotation delays! Hardware vs. software solutions?

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 has major impact on DBMS performance!

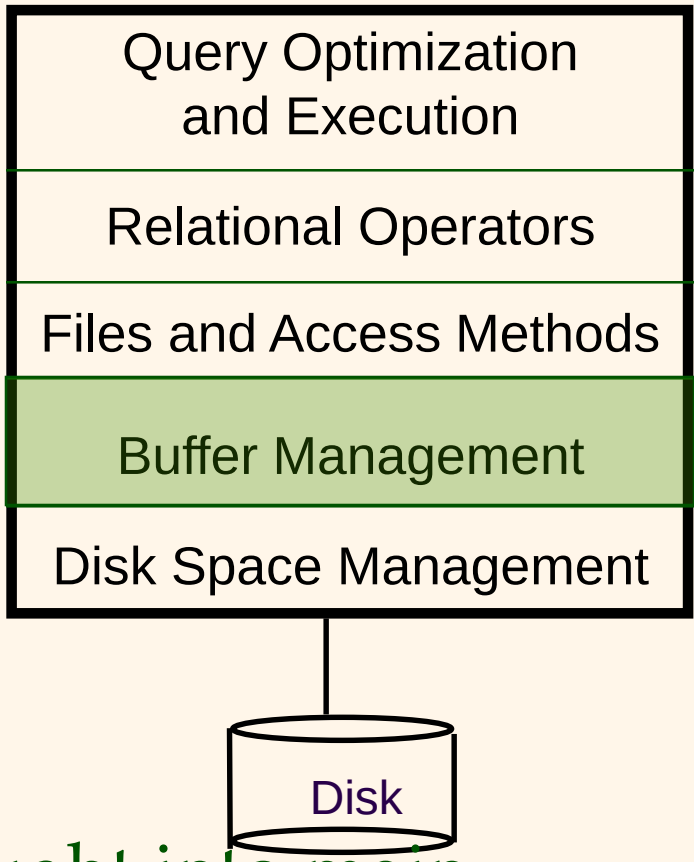
Storage Hierarchy



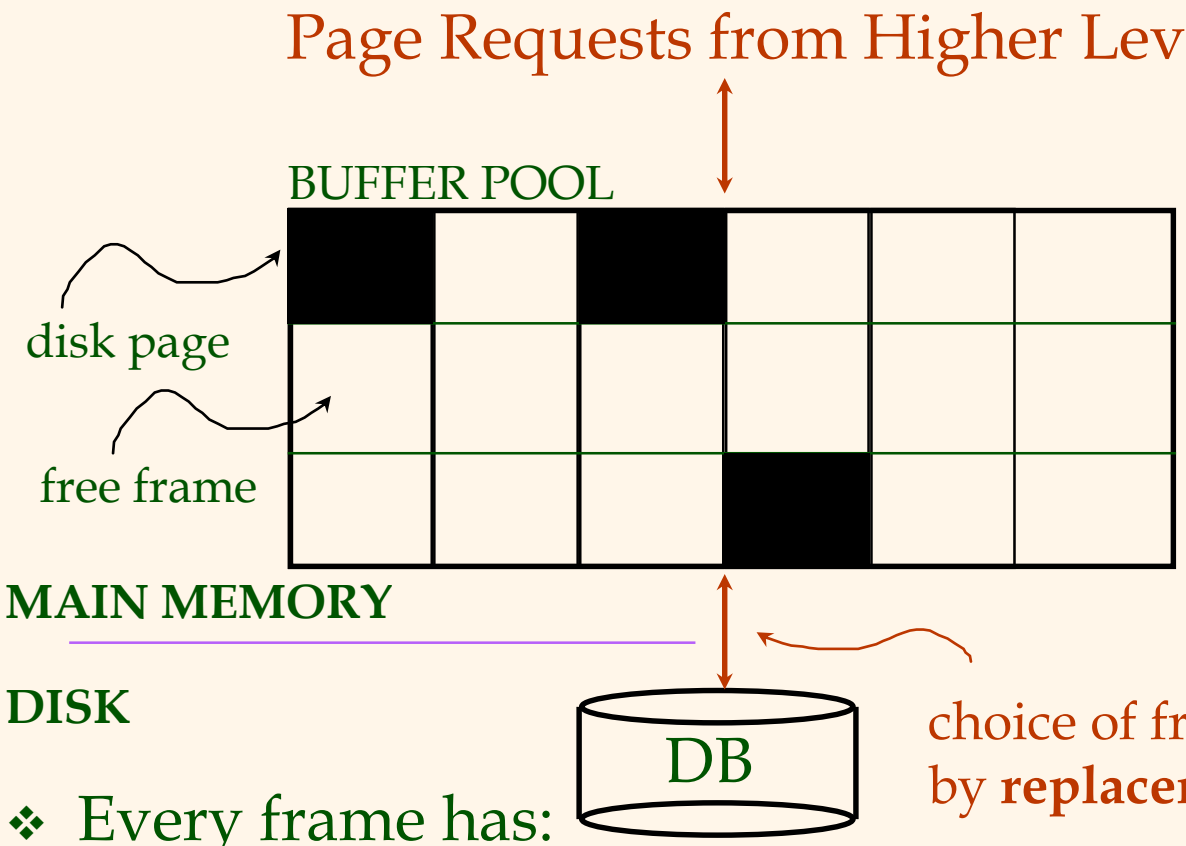
- Typical storage hierarchy:
  - Main memory (RAM) for currently used data.
  - Disk for the main database (secondary storage).
  - Tapes for archiving older versions of the data (tertiary storage).

Buffer Manager

- The buffer manager is a software layer responsible for bringing pages from disk to main memory as needed
- The buffer manager manages the available main memory (*buffer pool*) by partitioning it into a collection of pages (*frames*)
- Because all the data cannot be brought into main memory at one time, the buffer manager brings pages into memory only as they are needed, and decide what existing pages in main memory need to be replaced (*replacement policy*)



Buffer Pool



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

- Every frame has:
  - *Pin_count:* The number of users who are using that frame
  - *Dirty bit:* Indication whether the page has been modified since brought to the buffer

When a Page is Requested ...

- If requested page is not in pool:
  - If there are page frames in the pool with pin_count=0
    - Choose one of these frames for replacement
    - If the chosen frame is dirty, write it to disk
    - Read requested page into the chosen frame
- Pin the page (increment the pin count) and return its address.

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

More on Buffer Management

- The requestor of a page is responsible on releasing it (*unpinning*), and indicate whether it has been modified:
  - *dirty* bit is used for this.
- A page in the pool may be requested many times,
  - A page is a candidate for replacement iff *pin count* = 0.
- If no frame has zero *pin_count* and a new page is requested, then the buffer manger must wait until some page is released or the transaction requesting the page is aborted
- Allowing multiple transactions to pin the same page may result in conflicting changes
  - Concurrency control would take care of this

Buffer Replacement Policy

- A frame is chosen for replacement by a *replacement policy.*

- *Least Recently Used (LRU):*
  - Can be implemented using a queue of pointers for frames with zero pin count
  - A frame is added to end of the queue when it becomes candidate for replacement (i.e., pin count becomes zero)
  - The page chosen for replacement is the one in the frame at the head of the queue.

Buffer Replacement Policy

- LRU + Clock:
  - A variable, named *current*, is set from 1 to N (no. buffers)
  - The *current* frame is considered for replacement, if it does not qualify, the *current* is incremented
  - Each frame has a reference bit, initially set to 0, turned to 1 once the *pin count* becomes 0
  - If the *current* frame has reference bit 1, the clock algorithm turns the bit to 0 and increments *current*
    - This way , a recently referenced page is less likely to be replaced
  - If the *current* frame has pin count 0 and reference count 0, it is chosen for replacement

Buffer Replacement Policy

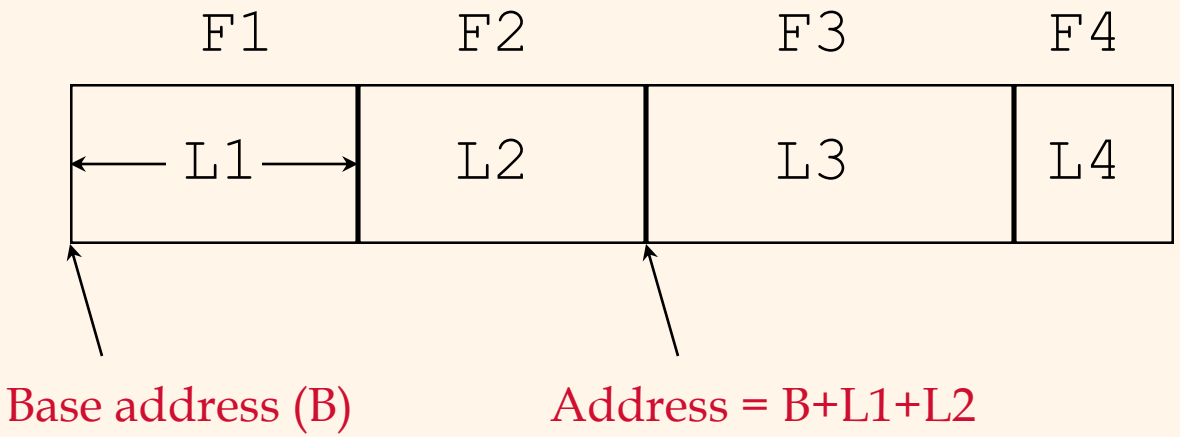
- Policy can have big impact on # of I/O's; depends on the *access pattern*.
- *Sequential flooding*: Nasty situation caused by LRU + repeated sequential scans.
  - # buffer frames < # pages in file means each page request causes an I/O.
- Other replacement techniques can be used, e.g., Random, FIFO, and MRU (Most recently used)

DBMS vs. OS File System

OS does disk space & buffer management:
Why not let OS manage these tasks?

- Differences in OS support: portability issues
- 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.

Record Formats: Fixed Length



- Information about field types same for all records in a file; stored in *system catalogs.*
- Finding *i'th* field does not require scan of record.

System Catalogs

- 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.

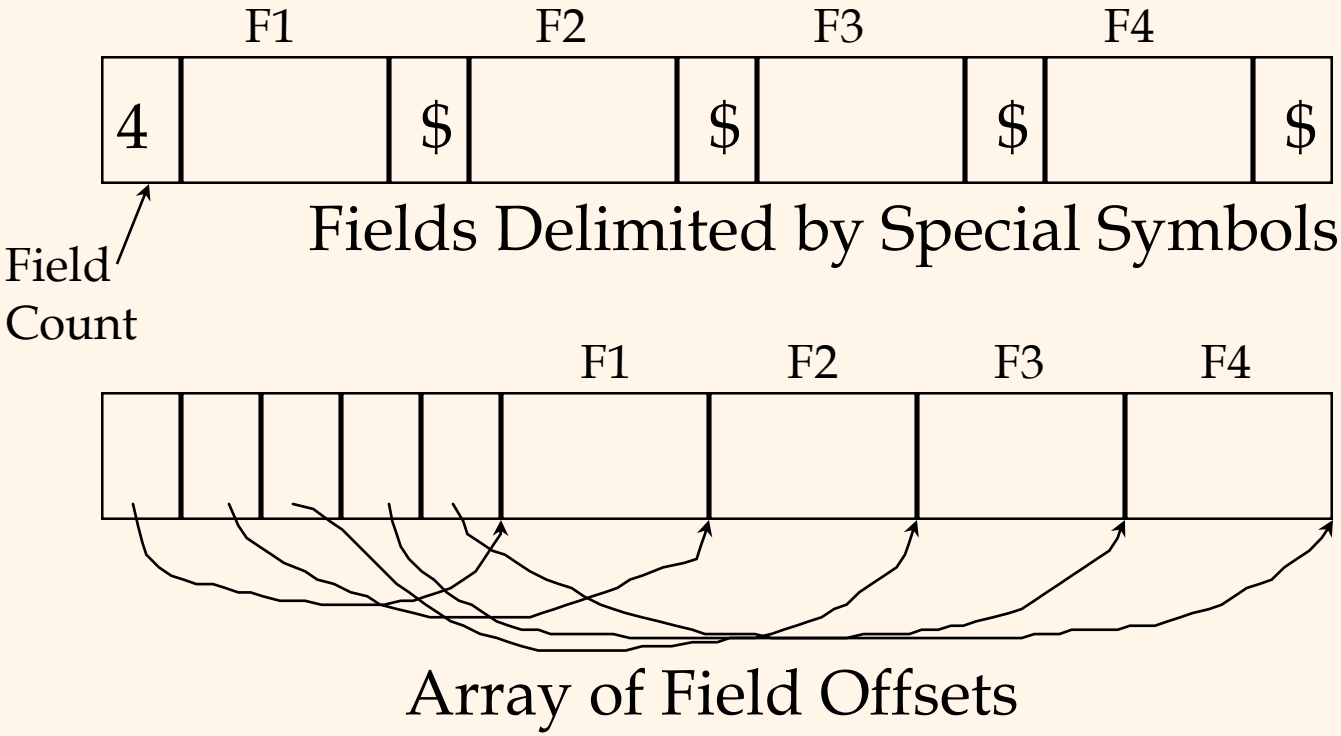
* *Catalogs are themselves stored as relations*!

Attr_Cat(attr_name, rel_name, type, position)

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

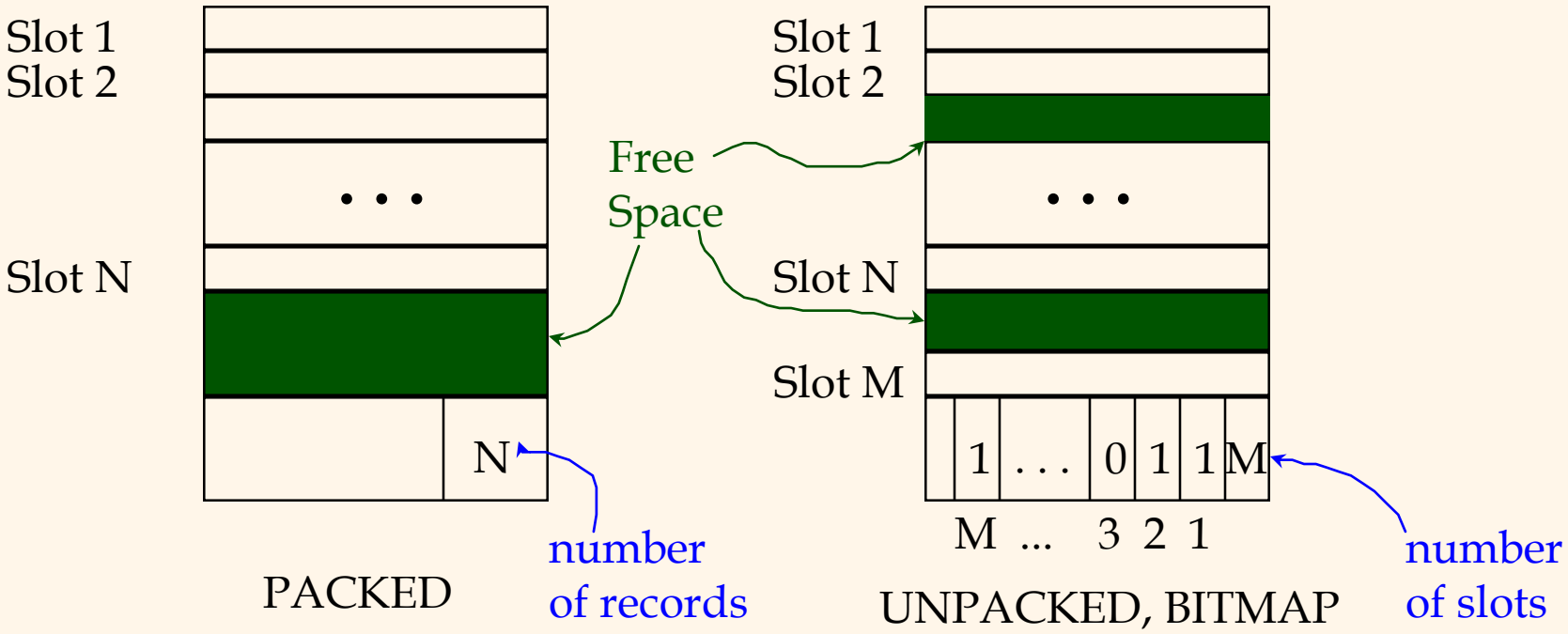
Record Formats: Variable Length

- Two alternative formats (# fields is fixed):



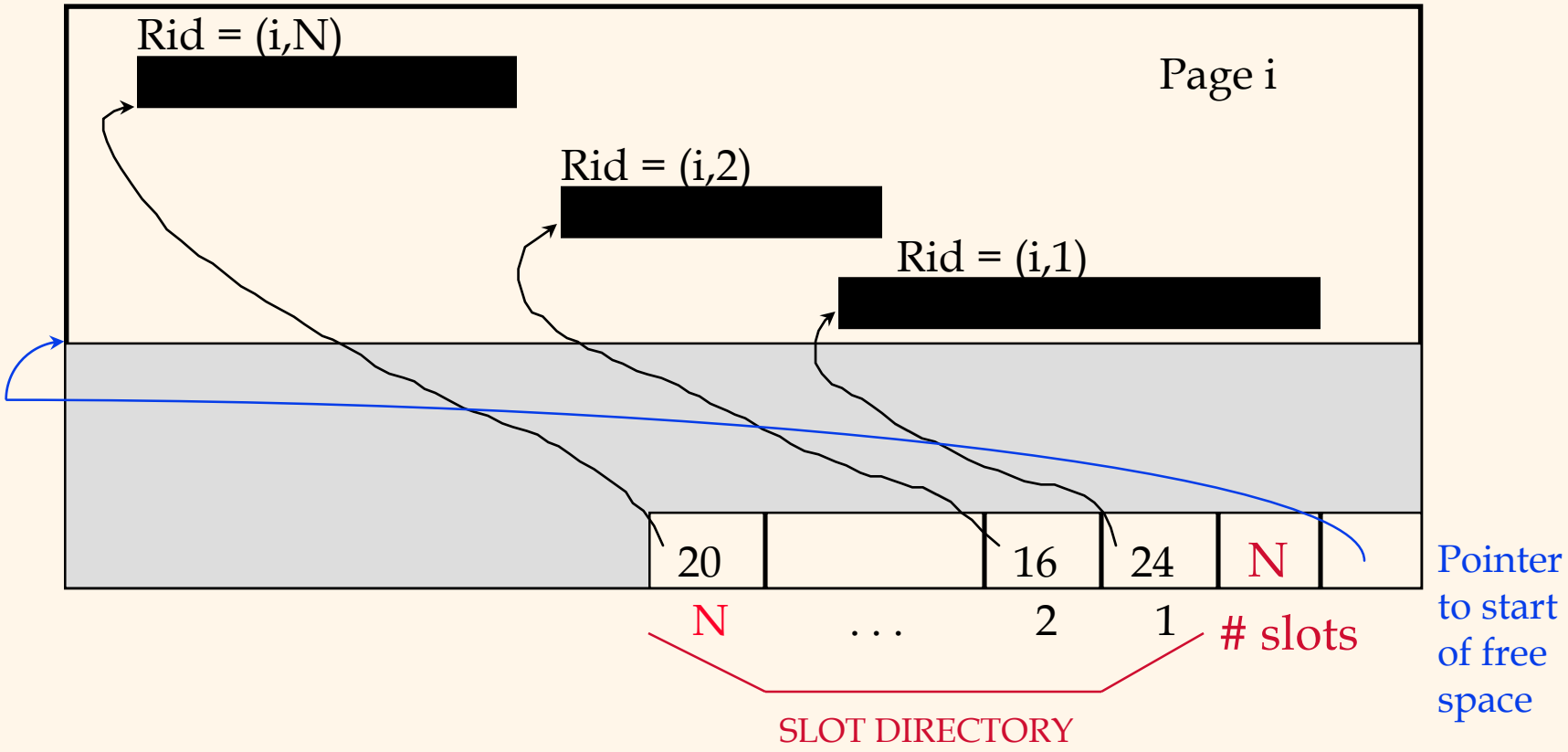
* Second offers direct access to i'th field, efficient storage of *nulls* (special *don't know* value); small directory overhead.

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.*

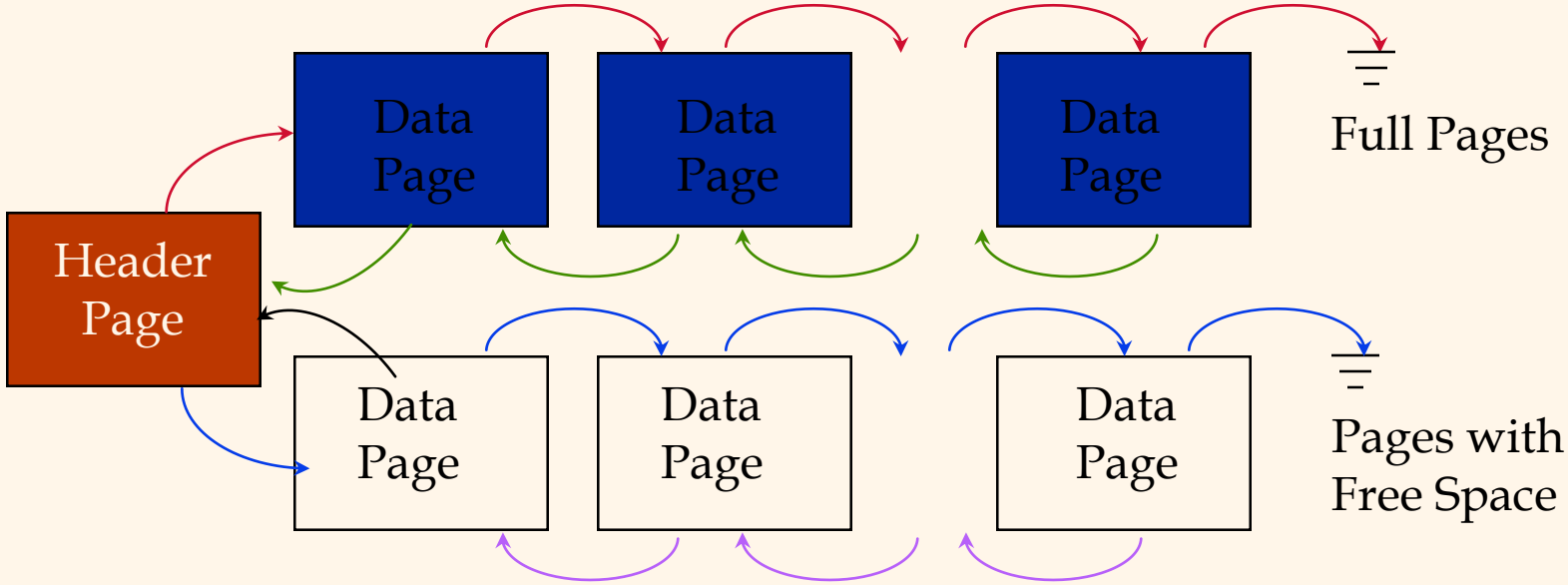
Files of Records

- 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)

Unordered (Heap) Files

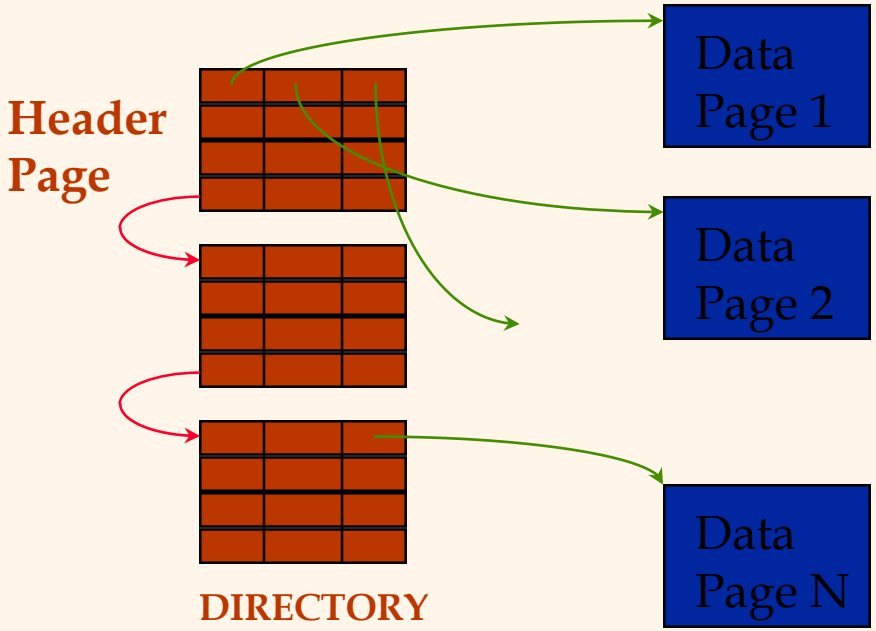
- Simplest file structure contains records in no particular order.
- As file grows and shrinks, disk pages are allocated and de-allocated.
- To support record level operations, we must:
  - keep track of the *pages* in a file
  - keep track of *free space* on pages
  - keep track of the *records* on 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 a collection of pages; linked list implementation is just one alternative.
  - *Much smaller than linked list of all HF pages*!