Storing Data: Disks and Files

Chapter 9

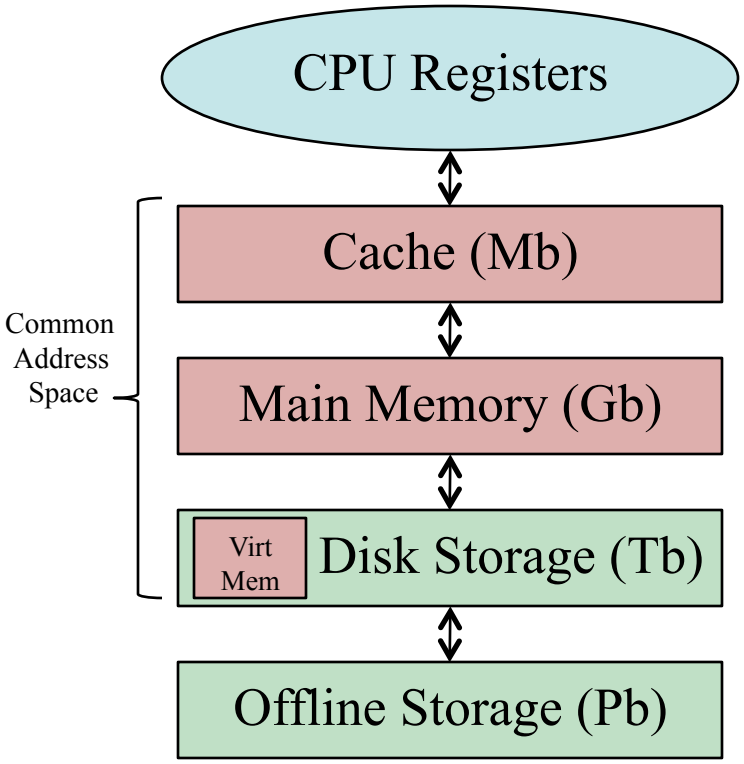
Disks and Files

- 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 Memory?

- *Costs too much.* $100 will buy you either 4GB of RAM or 2TB of disk today (500x).
- *Main memory is volatile.* We want data to be saved between runs. (Obviously!)
- Data Size > Memory Size > Address Space
- Typical storage hierarchy:
  - CPU Registers – temporary variables
  - Cache – Fast copies of frequently accessed memory locations (Cache and memory should indistinguishable)
  - Main memory (RAM) for currently used "addressable" data.
  - Disk for the main "big data" (secondary storage).

Storage Hierarchy



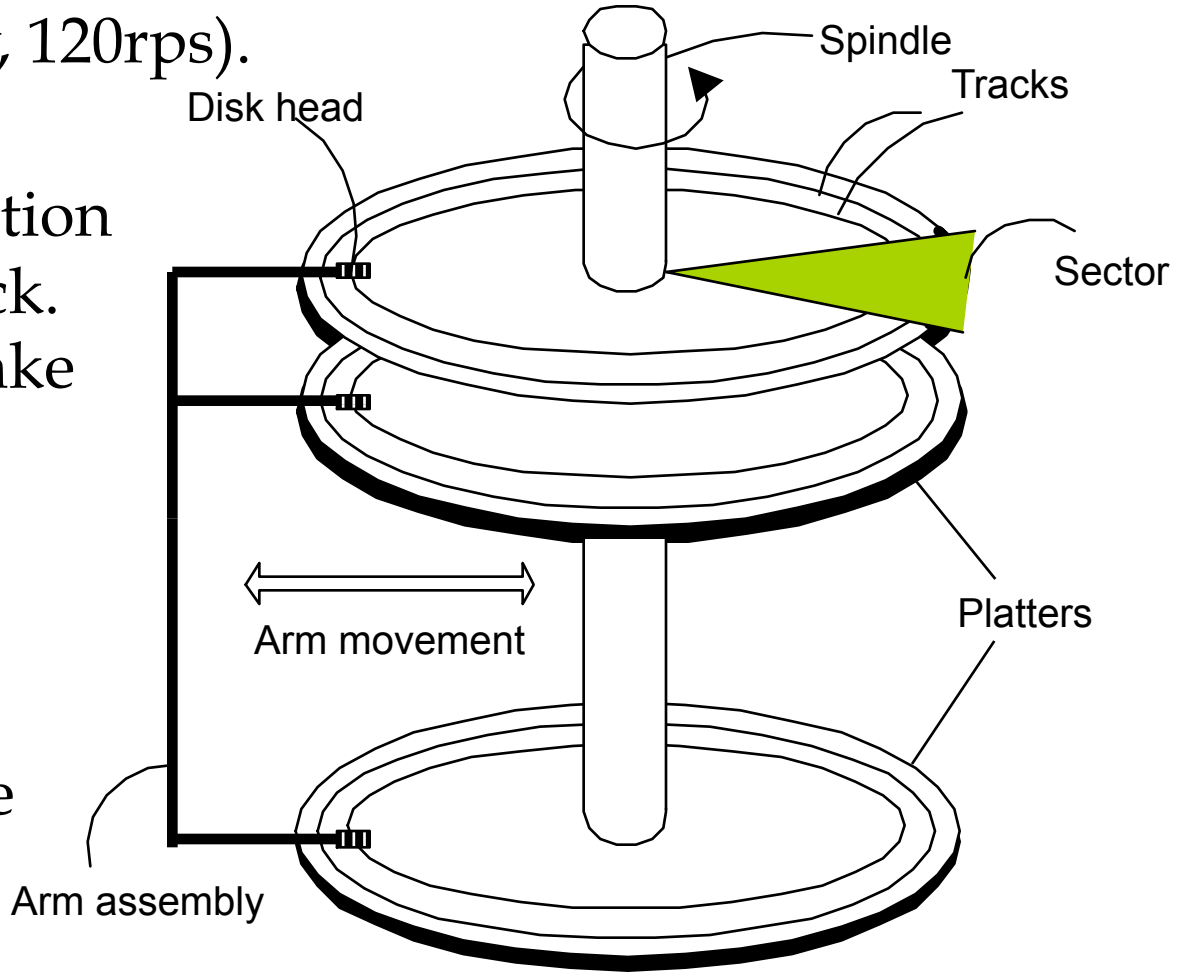
- CPU Registers – temporary program variables
- Cache – Fast copies of frequently accessed memory locations (Cache and memory are indistinguishable)
- Main memory (RAM) for currently used "addressable" data.
- Disk for the main database (*secondary* storage).
- Tapes for archiving older versions of the data (*tertiary* storage).

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!

Components of a Disk

- The platters spin (say, 120rps).
- 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).



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.
  - Seek time varies from about 2 to 15mS
  - Rotational delay from 0 to 8.3mS (ave 4.2mS)
  - Transfer rate is about 3.5mS per 256Kb 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!

RAID

- Redundant Arrays of Independent Disks (RAID)
- Disk Array: Arrangement of several disks that gives abstraction of a single, large disk.
- Goals: Increase performance and reliability.
- Two main techniques:
  - *Data striping*: Data is partitioned; size of a partition is called the striping unit. Partitions are distributed over several disks.
  - *Distributed Redundancy*: More disks → failures. Redundant information allows reconstruction of data if a disk fails.

RAID Levels

- Level 0: No redundancy (striping only)
- Level 1: Mirrored (two identical copies)
  - Each disk has a mirror image (check disk)
  - Only reads one copy, writes involves two disks.
  - Maximum transfer rate = transfer rate of one disk
- Level 0+1: Striping and Mirroring
  - Parallel reads (striping's performance advantage)
  - Writes involves two disks.
  - Maximum transfer rate = aggregate bandwidth

RAID Levels (Contd.)

- Level 3: Bit-Interleaved Parity
  - Striping Unit: One bit. One check disk.
  - Each read and write request involves all disks; disk array can process one request at a time.
- Level 4: Block-Interleaved Parity
  - Striping Unit: One disk block. One check disk.
  - Parallel reads possible for small requests, large requests can utilize full bandwidth
  - Writes involve modified block and check disk
- Level 5: Block-Interleaved Distributed Parity
  - Similar to RAID Level 4, but parity blocks are distributed over all disks

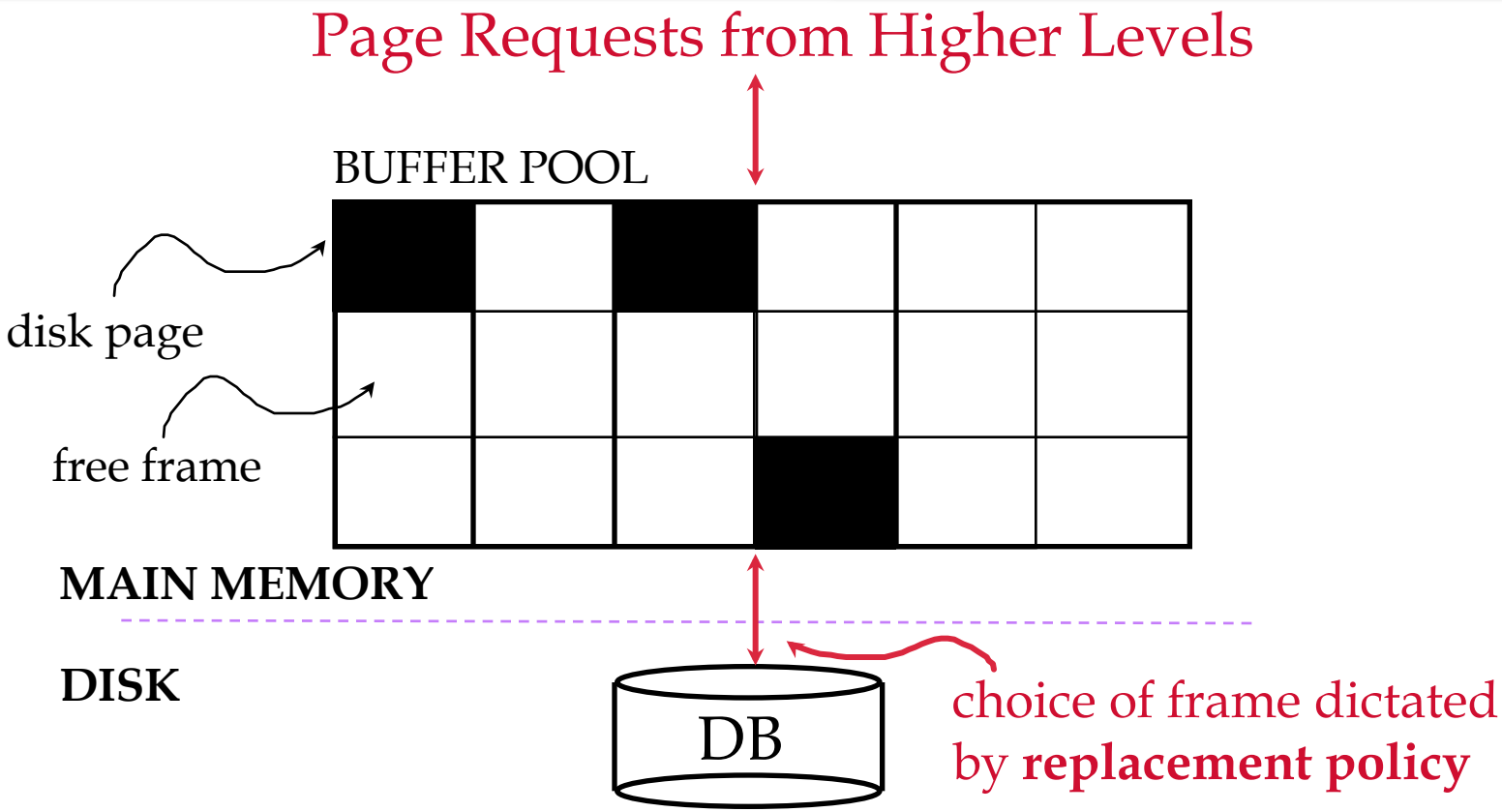
RAID outcome

- Level 0 low-cost, fast, no-protection
- Level 3 always dominates Level 2
- Level 5 always dominates Level 4
- Level 0+1 is commonly used in small systems or when percentage of writes is high
- Level 3 used when optimizing for large high-transfer rates
- Level 5 is a good general-purpose solution

Disk Space Management

- Lowest layer of DBMS software manages how is used space on disk. Abstraction unit is a "*page*"
- 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.
- O/S Disk management vs. DBMS

Buffer Management in a DBMS



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

When a Page is Requested ...

- If requested page is not in pool:
  - Choose a frame for *replacement*
  - If frame is *dirty* *(its contents modified)*, write it to disk
  - Read requested page into chosen frame
- *Pin* the page 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

- Requestor of page must *unpin* it when done, and indicate whether page has been modified:
  - *dirty* bit is used for this.
- Page in pool may be requested many times,
  - a *pin count* is used. A page is a candidate for replacement iff *pin count* = 0.
- Crash recovery protocols may entail additional I/O when a frame is replaced. (*Write-Ahead Log* protocol; more later.)

Buffer Replacement Policy

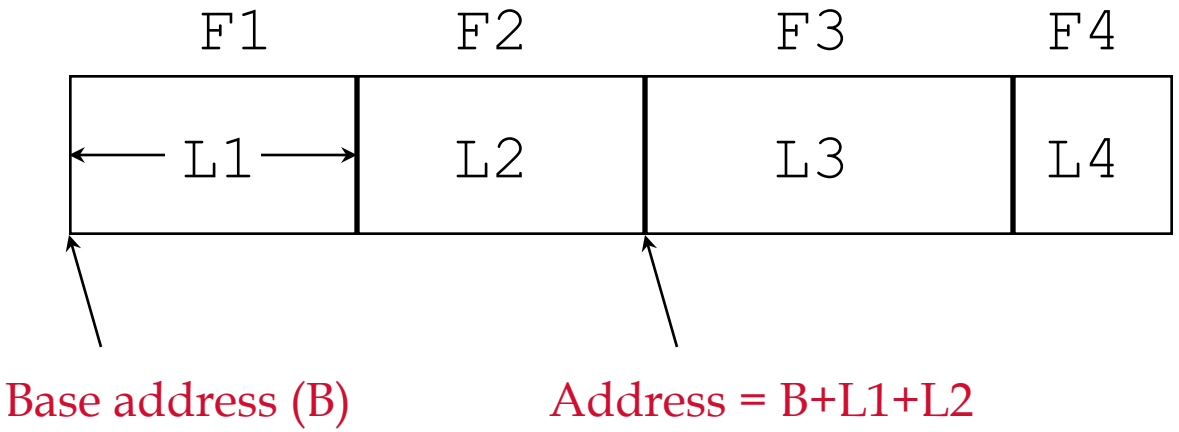
- Frame is chosen for replacement by a *replacement policy:*
  - Least-recently-used (LRU), Clock, MRU etc.
- Policy can have big impact on # of I/O's; depends on the *access pattern*.
- *Sequential flooding*: Nasty collision situation caused by LRU + repeated sequential scans.
  - # buffer frames < # 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 don'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.

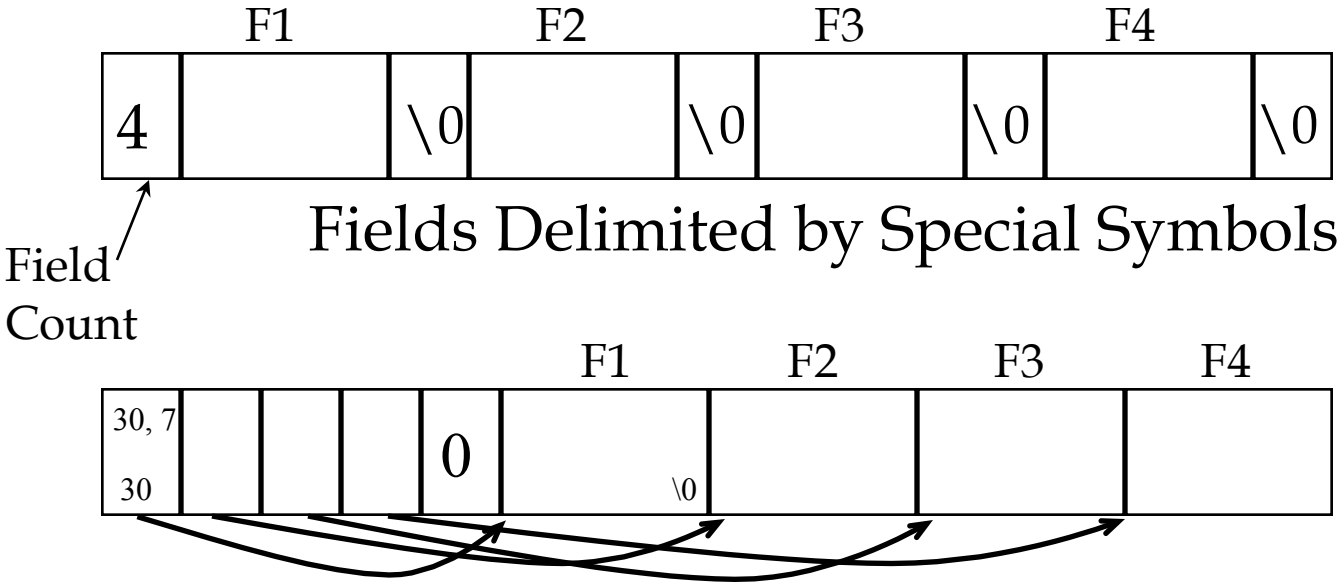
Record Formats: Fixed Length



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

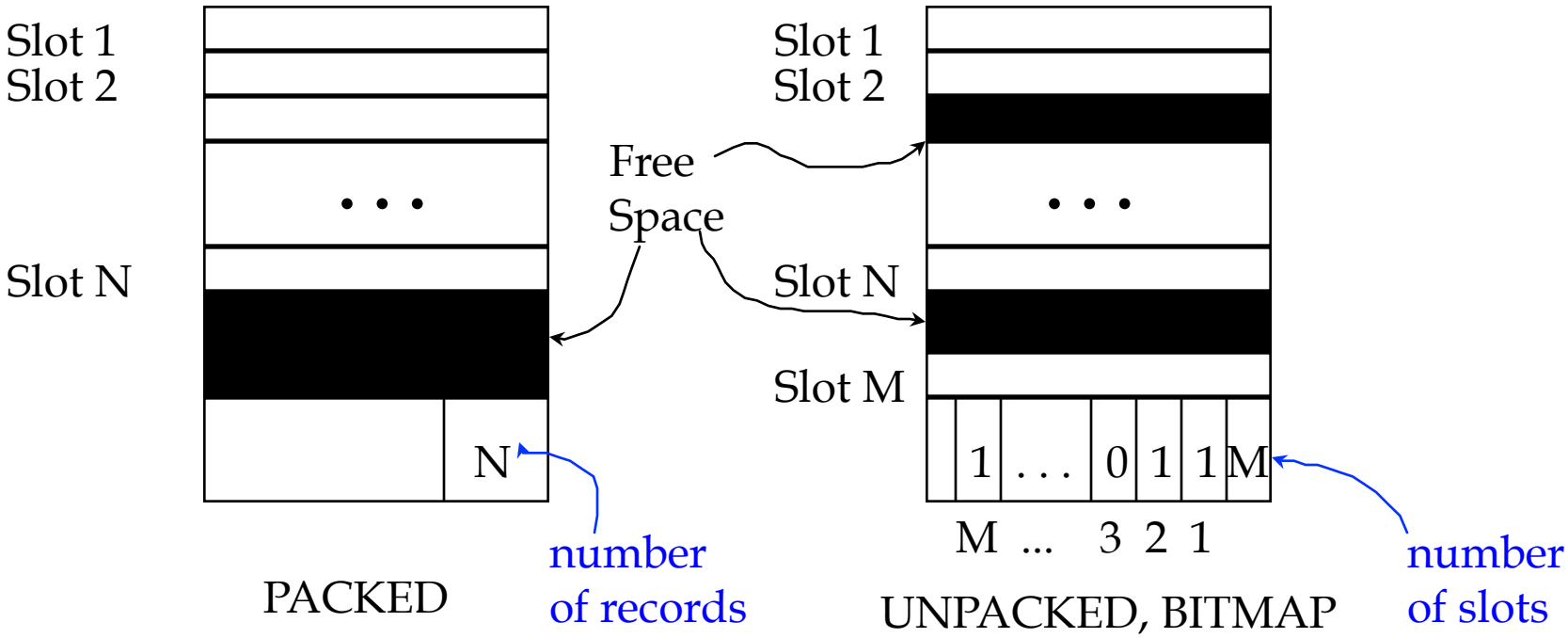
Record Formats: Variable Length

- Two alternative formats (# fields is fixed):



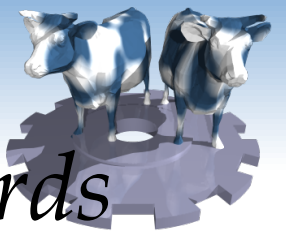
☛ Second offers direct access to i'th field, efficient storage of *nulls* (special *don't know* value); small directory overhead.

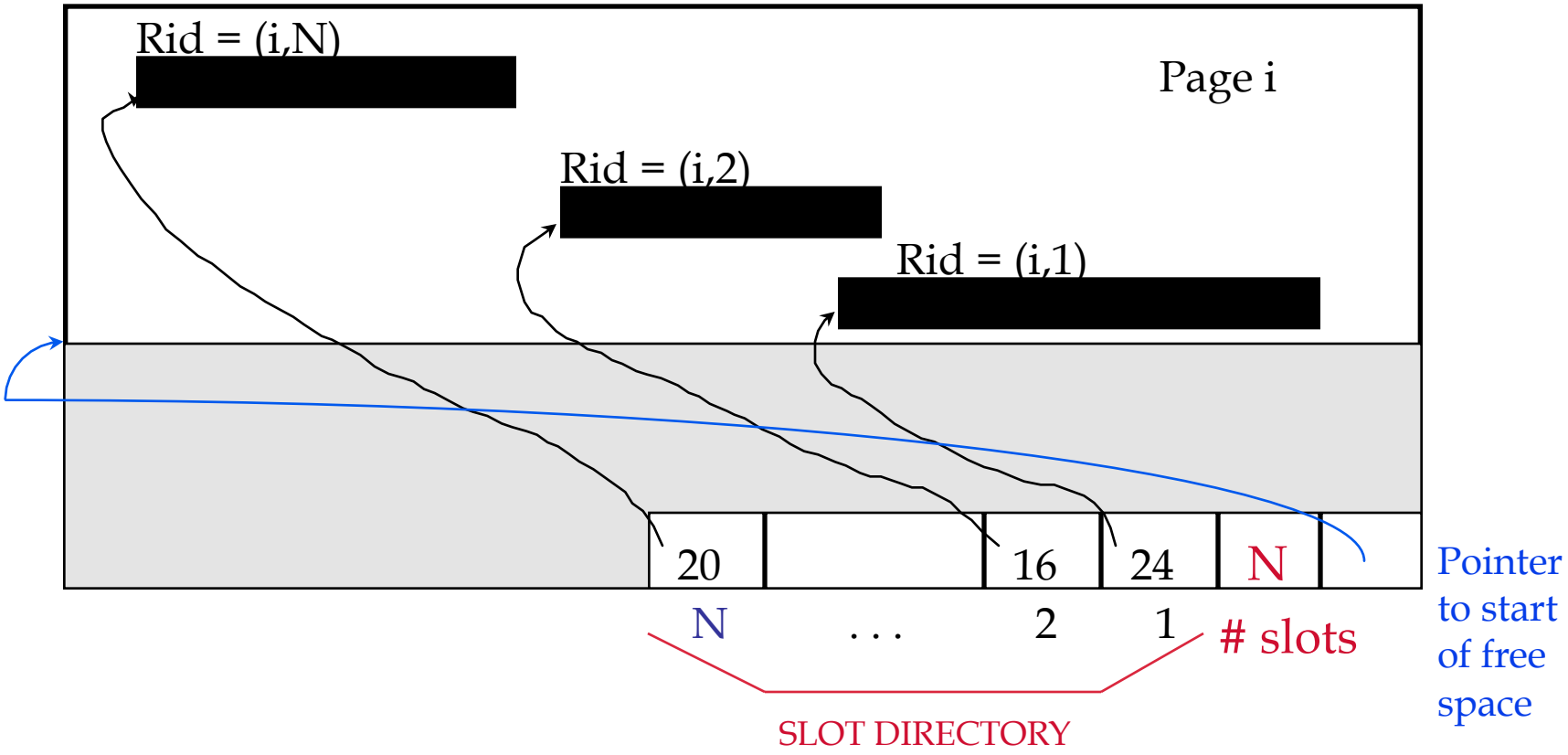
Page Formats: Fixed Length Records



- *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.*

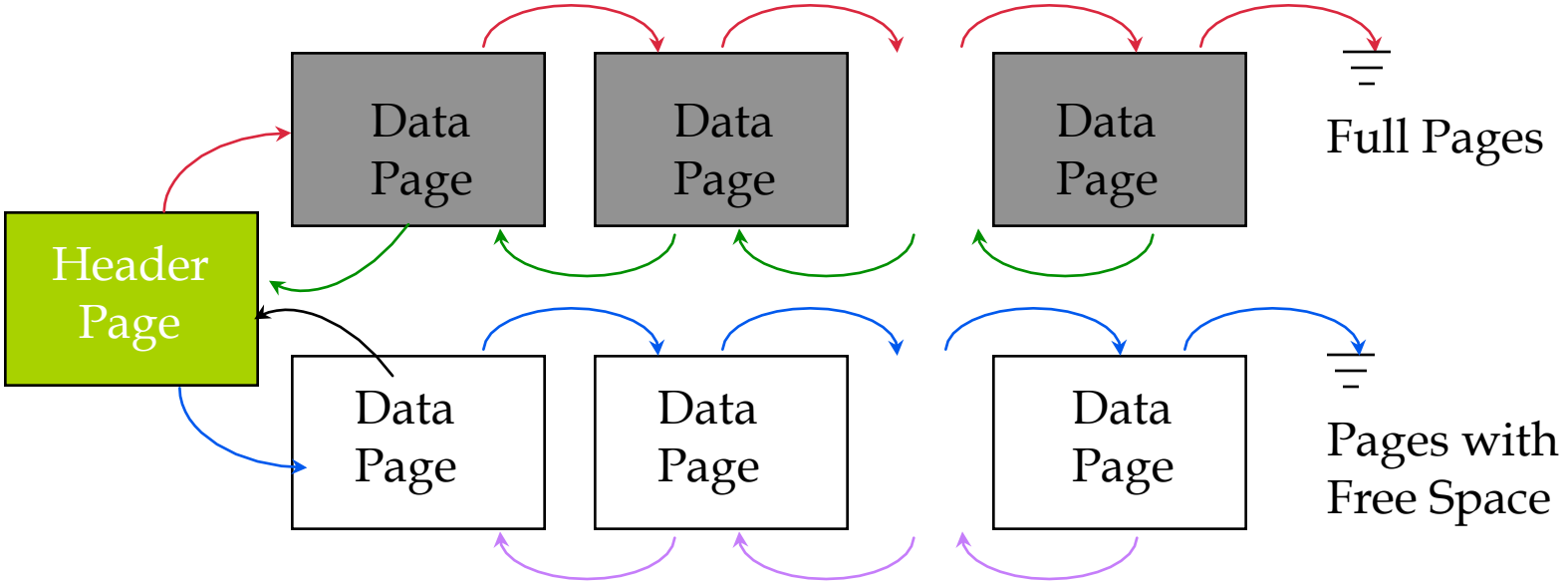
Files of Records

- 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)

Unordered (Heap) Files

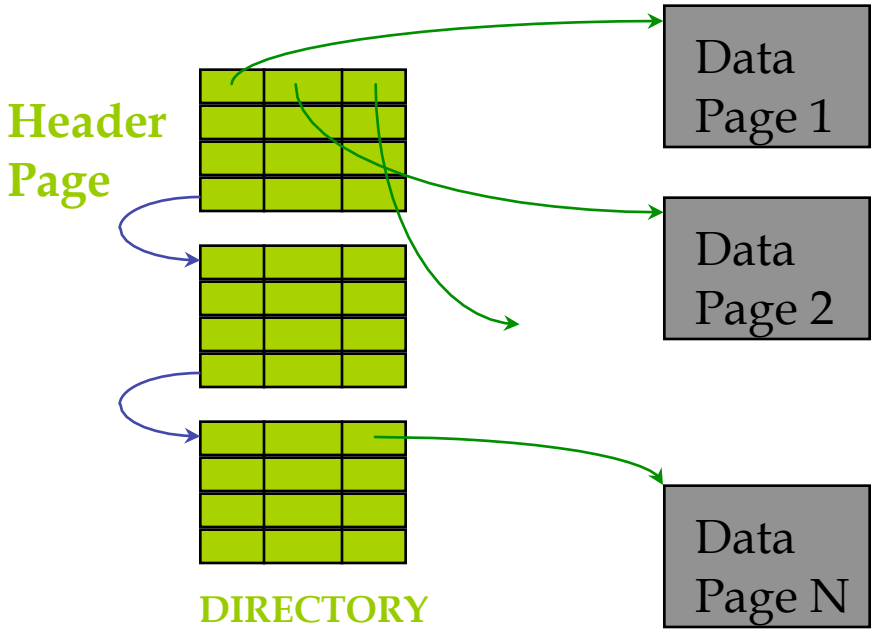
- Simplest file structure contains records in no particular order.
- As file grows and shrinks, disk pages are allocated and de-allocated.
- To support record level operations, we must:
  - keep track of the *pages* in a file
  - keep track of *free space* on pages
  - keep track of the *records* on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

System Catalogs

- 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 index:
  - structure (e.g., B+ tree) and search key fields
- 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

Sqlite_master

```
>>> import sqlite3
>>> db = sqlite3.connect("dogs.db")
>>> cursor = db.cursor()
>>> cursor.execute("SELECT * FROM sqlite_master")
<sqlite3.Cursor object at 0x1004388c8>
>>> for row in cursor:
...     print row
...
```

Sqlite_master

```
(u'table', u'Customers', u'Customers', 2,
    u'CREATE TABLE Customers (cardNo  integer primary key,
        first text, last text, sex char, dob date)')
(u'table', u'Movies', u'Movies', 19173,
    u'CREATE TABLE Movies (movieId integer primary key,
        title text, year integer)')
(u'table', u'Rentals', u'Rentals', 19753,
   u'CREATE TABLE Rentals (cardNo  integer,
       movieId integer, date date, rating  integer,
       PRIMARY KEY(cardNo, movieID, date),
       FOREIGN KEY (cardNo) REFERENCES Customers,
       FOREIGN KEY (movieId) REFERENCES Movies)')
(u'index', u'sqlite_autoindex_Rentals_1', u'Rentals', 19754,
   None)
```

Summary

- Disks provide cheap, non-volatile storage.
  - Random access, but cost 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 when frame chosen for replacement (which is sometime 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.

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*.)