Storing Data: Disks and Files

Lecture #6

Motivation

- User asks query

  Select eid, ename
  From Employees
  Where salary > 100K

- System scans through table Employees
  - considers each tuple
  - if salary > 100K, then return eid, ename

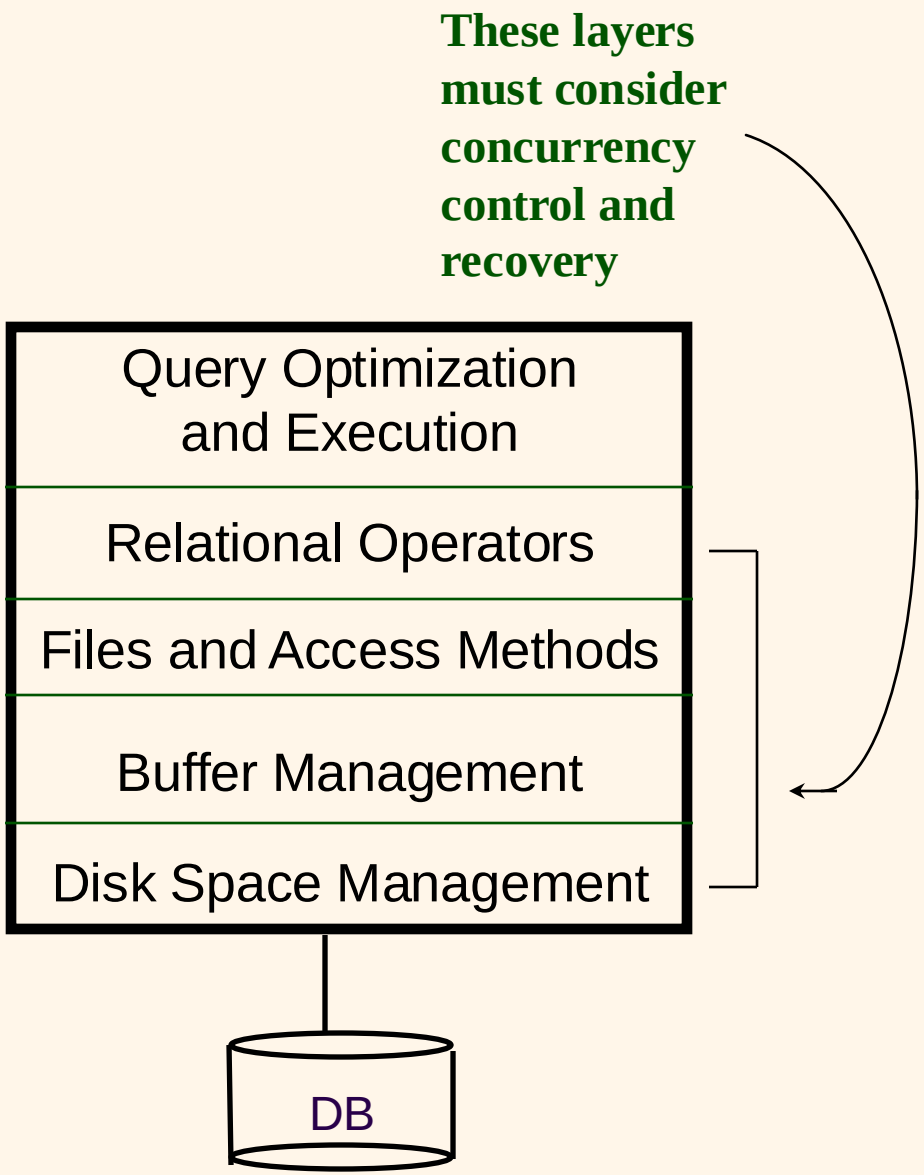
- How does this happen in the backend?

Motivation

- Table is stored as a file on disk
- File has multiple pages
- Each page has multiple tuples
- The system (RDBMS) works on a page at a time
  - in memory (the buffer)

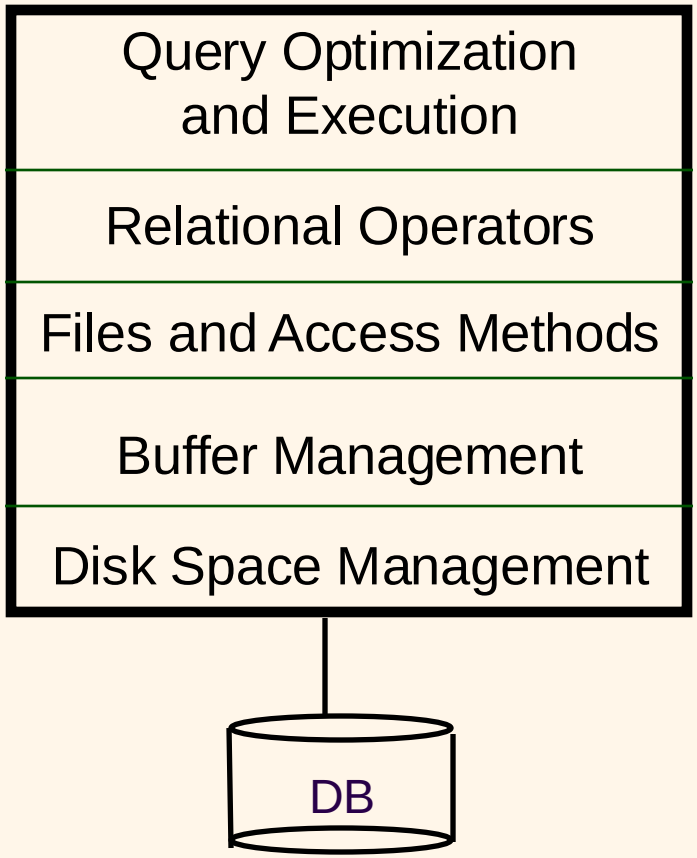
Motivation

- A typical DBMS has a layered architecture.
- The figure does not show the concurrency control and recovery components.
- This is one of several possible architectures; each system has its own variations.



An Example

- Select eid, ename
  From Employees
  Where salary > 100K

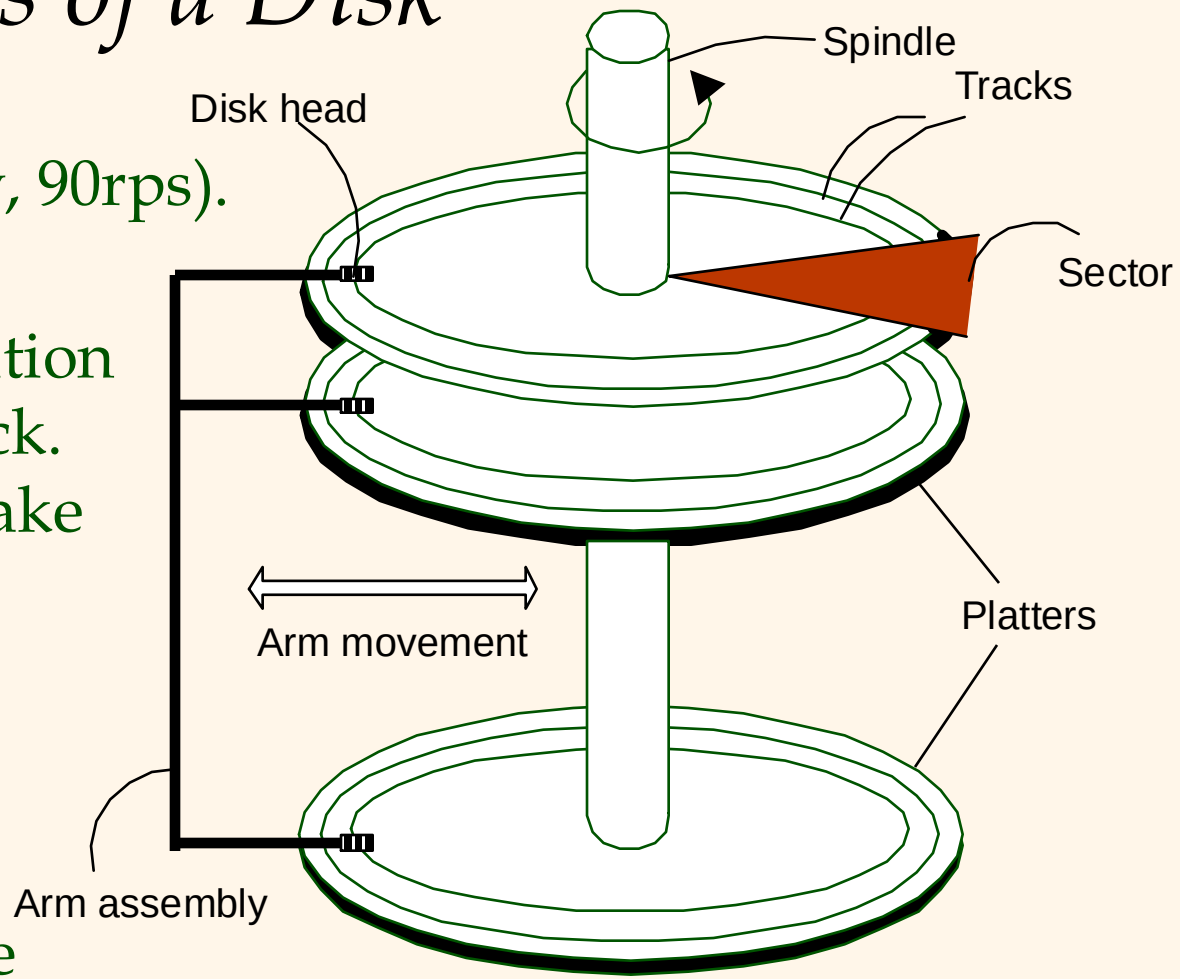


Disks

- Secondary storage device of choice.
- *random access* vs. *sequential*.
- Data is stored and retrieved in units called *disk blocks* or *pages*.

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).



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 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!

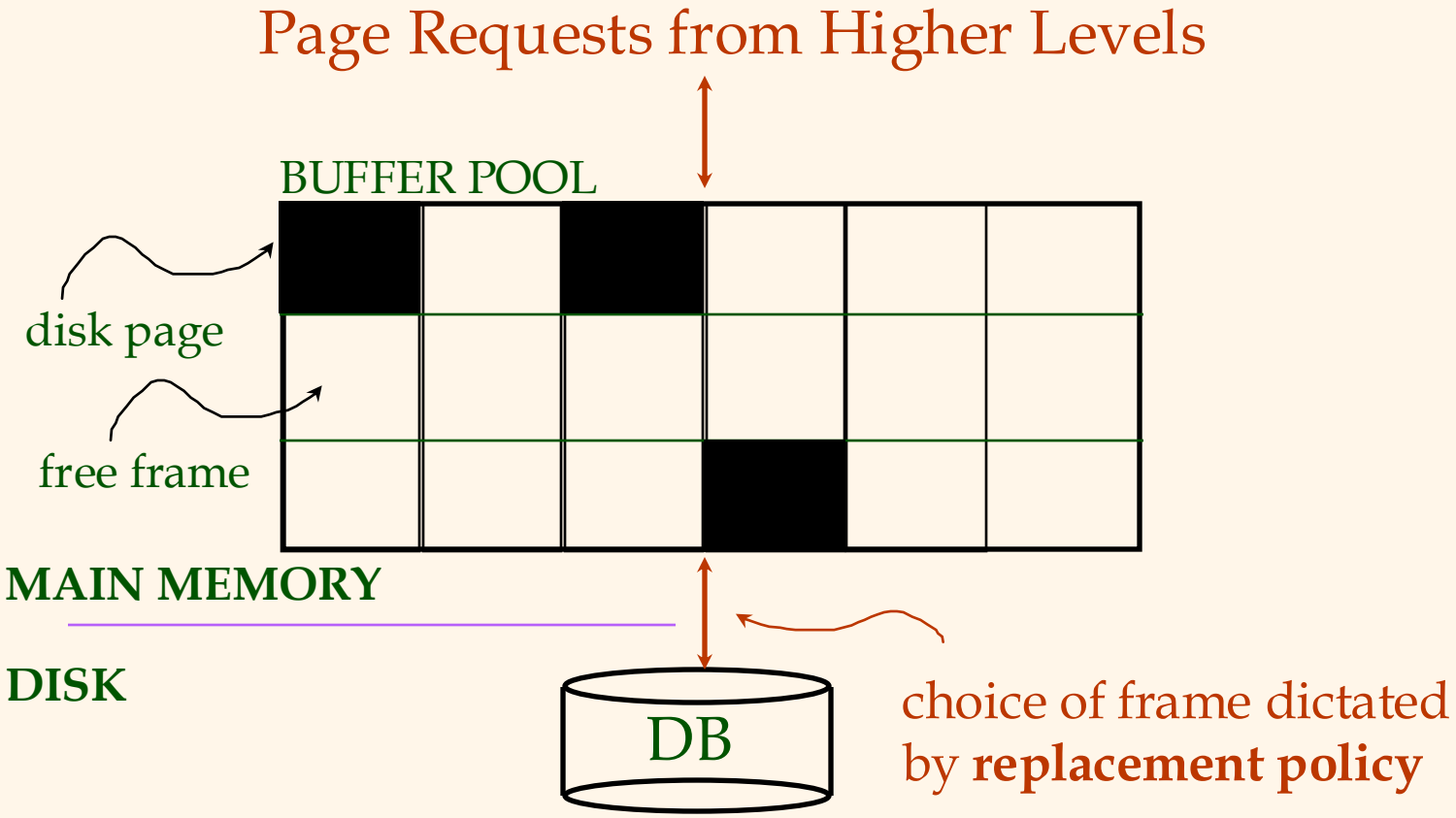
Key Things to Remember

- Notion of pages on disk
- Reading pages from disk, wring to pages on disk
  - very expensive
  - try to minimize this if we can

Disk Space Management

- Lowest layer of DBMS software manages space on disk.
- Higher levels call upon this layer to:
  - allocate/delete a page
  - read/write a page

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, write it to disk
  - Read requested page into chosen frame
- *Pin* the page and return its address.

More on Buffer Management

- When done, requestor of page releases the page: must unpin it, 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.

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).

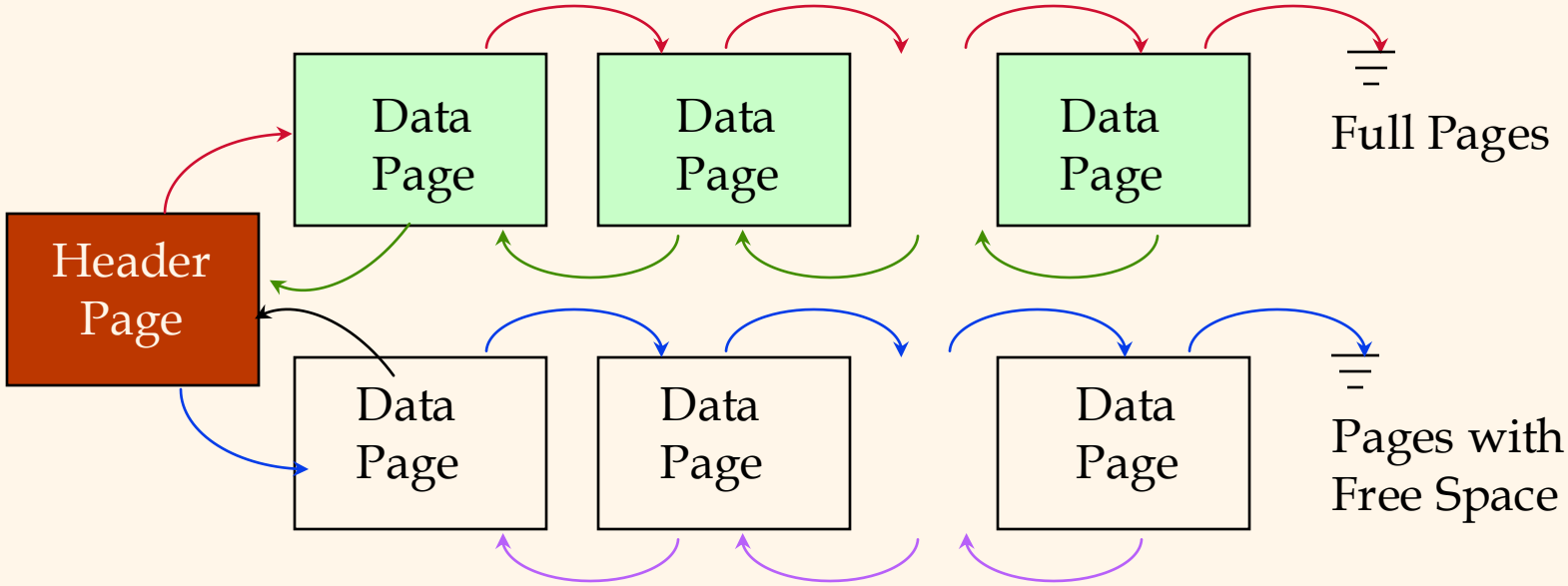
Files of Records

- 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

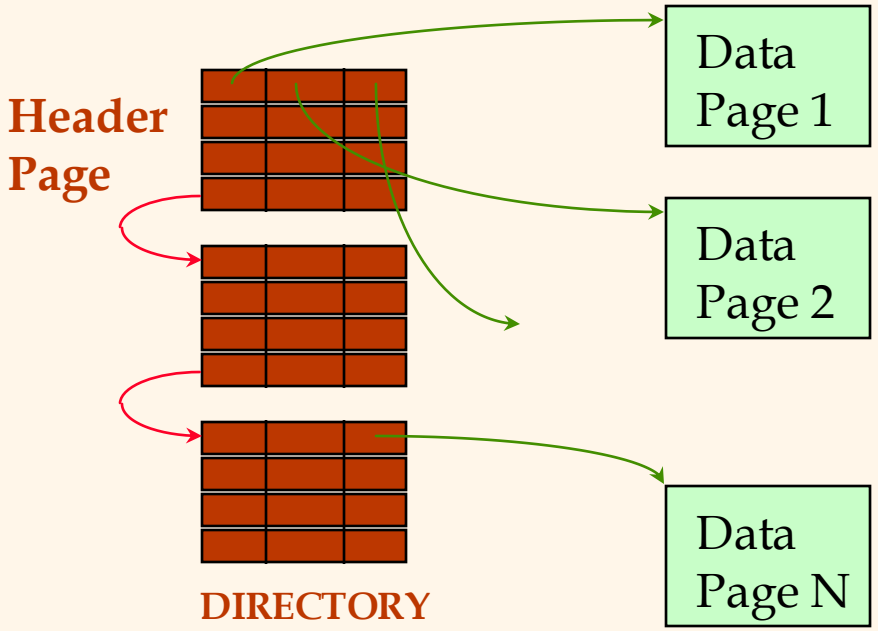
- Simplest file structure contains records in no particular order.
- As file grows and shrinks, disk pages are allocated and de-allocated.
- To support record level operations, we must:
  - keep track of the *pages* in a file
  - keep track of *free space* on pages
  - keep track of the *records* on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

Heap File Implemented as a List



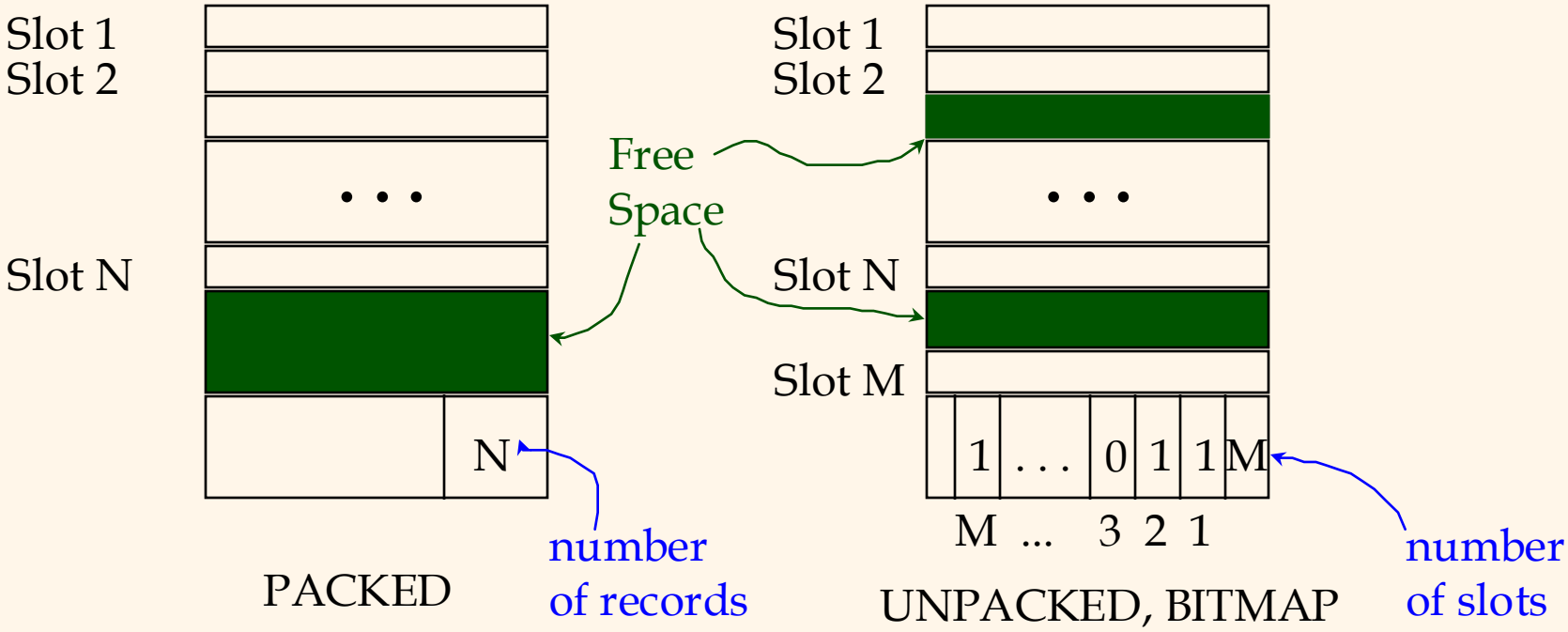
- The header page id and Heap file name must be stored someplace.
- Each page contains 2 `pointers' plus data.

Heap File Using a Page Directory



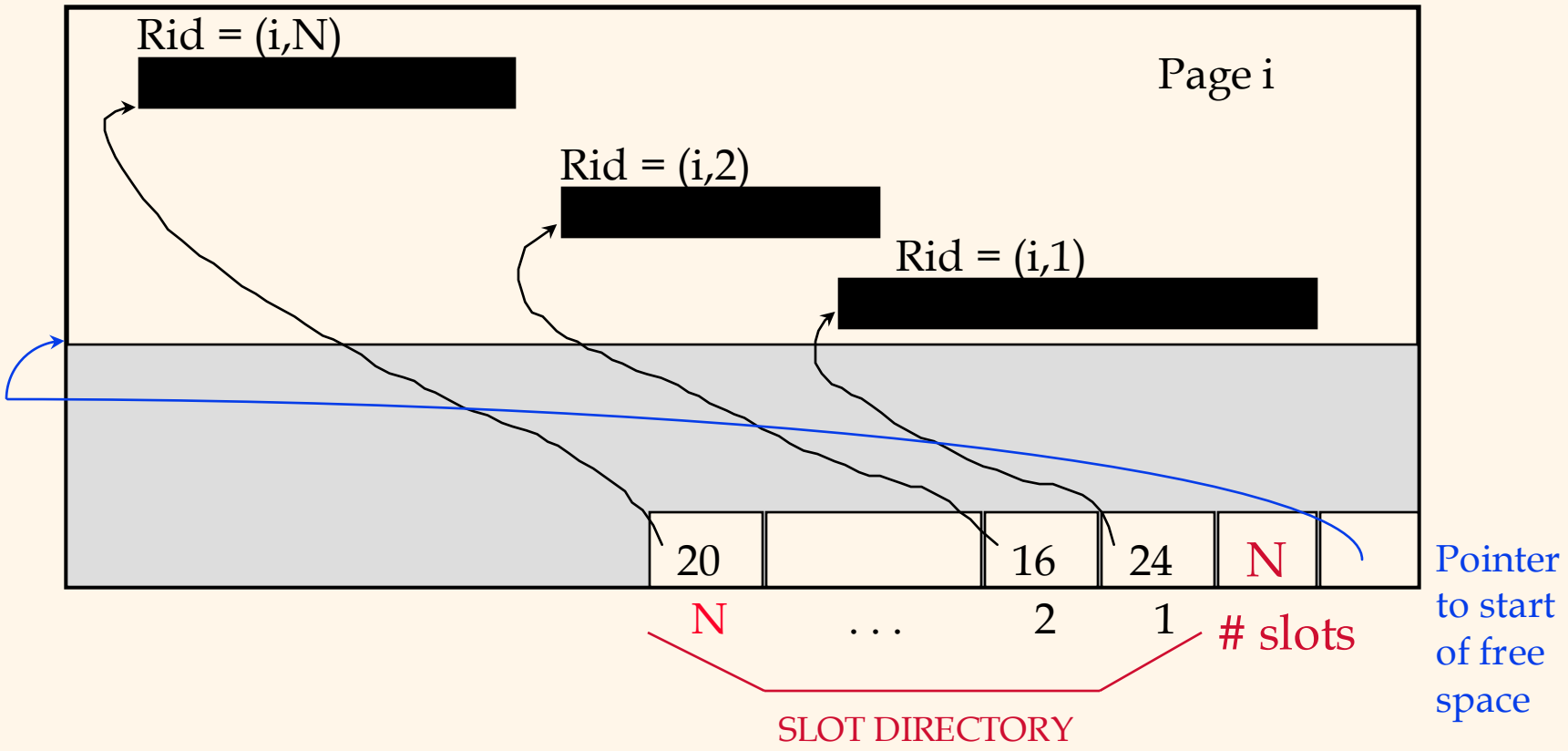
- The entry for a page can include the number of free bytes on the page.
- The directory is a collection of pages; linked list implementation is just one alternative.
  - *Much smaller than linked list of all HF pages*!

Page Formats: Fixed Length Records



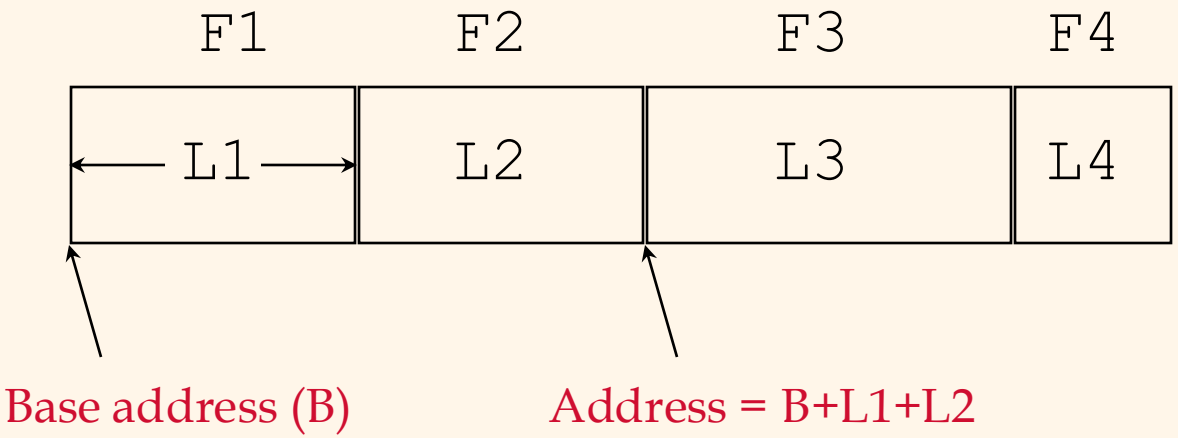
- *Record id = <page id, slot #>. In first alternative, moving records for free space management changes rid; may not be acceptable.*

Page Formats: Variable Length Records



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

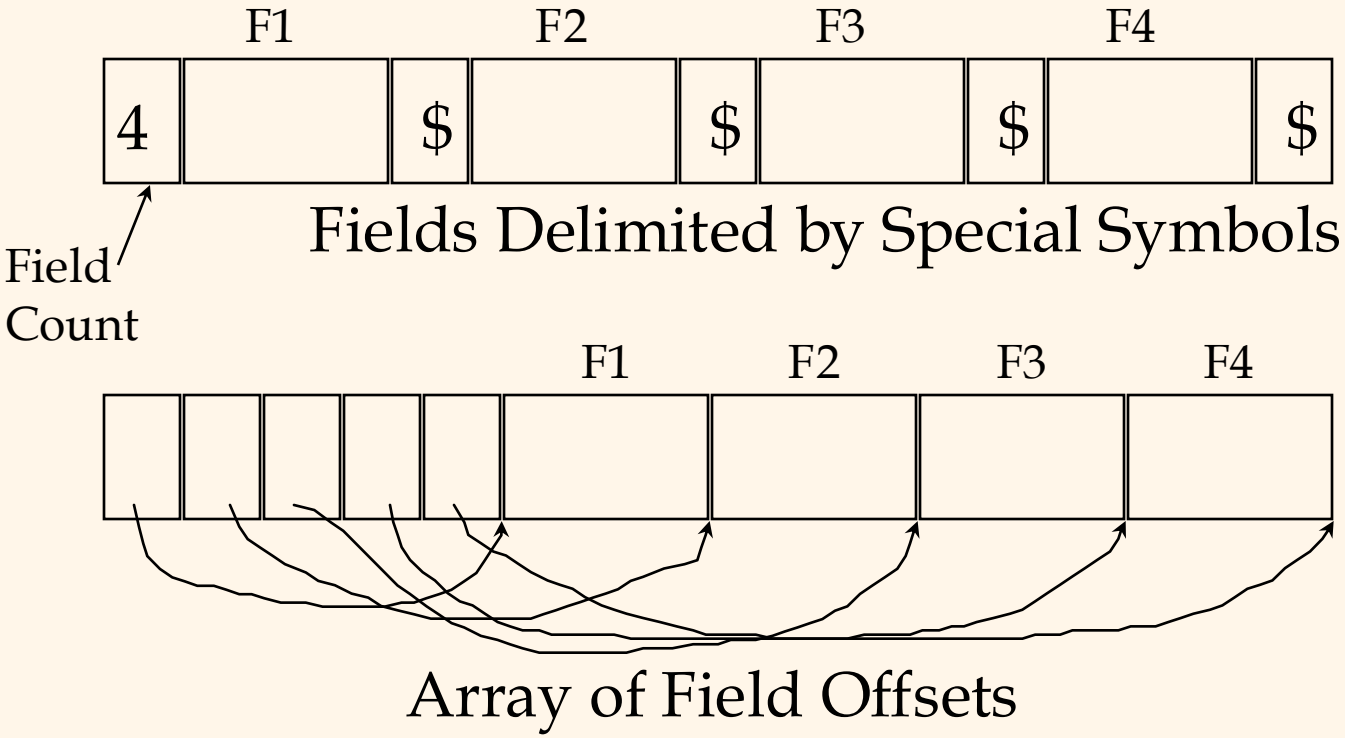
Record Formats: Fixed Length



- 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.

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.

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