Higher level processes ask buffer manager for pages. BM's job to figure out how to get & whether or not to keep a particular page in its pool.

BM keeps track of two additional pieces of information for each frame:

(1) pin_count - number of requests w/o release on a page

(2) dirty - whether page has been modified

On a page request BM

(1) chks if in pool. If yes update pin_count++ (pinning)

If no chooses a frame for replacement if has dirty bits write to disk

Read in page to frame pin_count = 1

(2) Return address of frame to requestor

When page released pin_count-- (unpinning)

Pin count must be 0 to replace.

Even if all pin-counts are 0 still need a policy to decide w/c frame to replace. This is b/c some pages may be requested more often.

Example Replacement Policies

LRU – least recently used
frame replaced
(use a queue of pointers
to frames with pin count
(bad case sequential flooding | Request pages 1, 2, –N then 1, 2, ...)

Clock – replace unpinned frames in sequential order 1, 2, ... N then 1, etc.

FIFO – first in first out

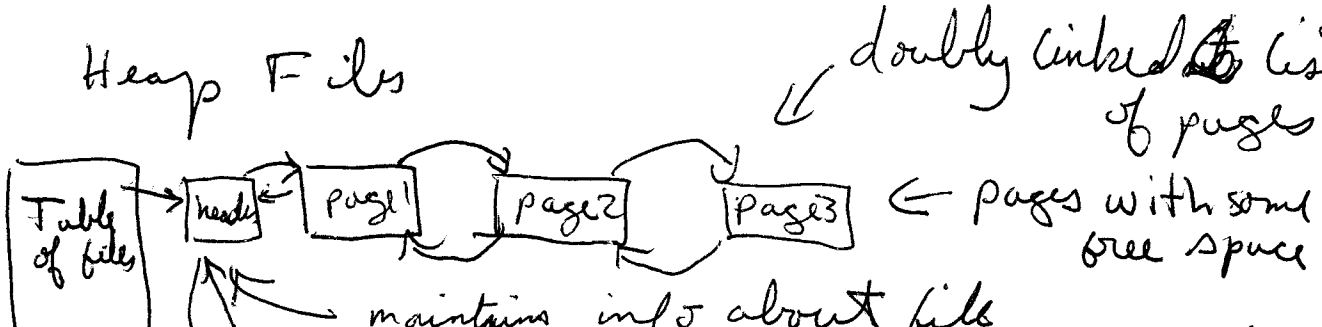
MRU – most recently used

Random

Buffer Management more powerful in DBMS's than OS's as can guess more often a/c page will be needed next. So can prefetch pages. (For instance in sequential scans)

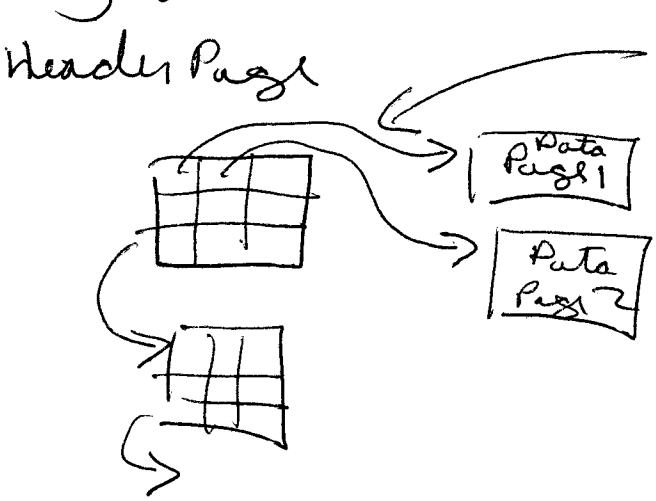
Files : How implemented

Heap Files



Above works well if all records same size. But if have variable length records then almost no page on full list so inserts slow.

Directory of Pages



in addition to the pointers directory has count of free slots per page.

Page Formats

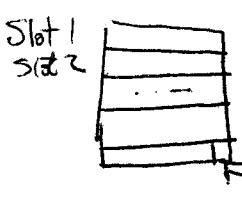
Pages divided into slots that can hold one record (assuming record fixed size)

A record in a file can be identified as <page ID, slot number> (This is record ID)

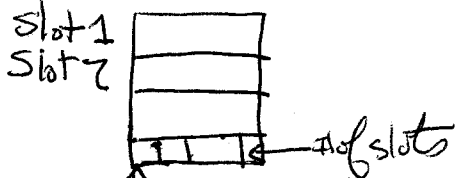
How to manage slots:

Fixed Length

Packed



Unpacked



Variable Length

use a directory of slots. Entries consist of record offset record length