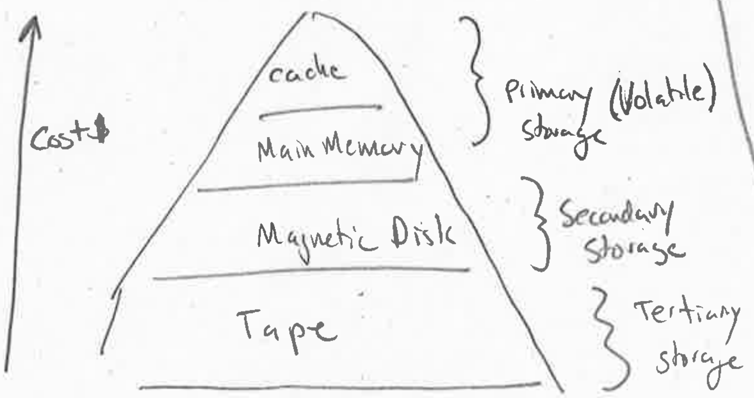
Thursday 26/2/2015 Buffer Management

Lecture Topics

I. Memory Hierarchy
II. RAID
III Disk Space Management
IV Buffer Management
V Record Format
VI Page Format
VII Heap Files (File Format)

I. Memory Hierachy



- 'Why' not have everything in main memory?
- Disk vs. Tape: Random access vs. sequential

II RAID

Goal: Improve performance & reliability

Idea: Use a disk array. Several disks that appear to be one.

Techiques:

(1) Data striping: partition data. size of partition is striping unit. Put partitions on several disks.

(2) Redundancy: more disk could mean more failures. Redundant data helps tolerad failures.

RAID 0: No Redundancy

RAID 1: MIRRORED (2 IDENTICAL COPIES)
Parallel reads
Writes to 2 DISKS
Maximum transfer rate = transfer rate of 1 disk.

RAID 0+1: STRIPING AND MIRRORING
- PARRALLEL READS
- WRITE TO TWO DISKS
- Maximum transfer = aggregate bandwidth (for several blocks)

RAID 2: ERROR CORRECTING CODES
- STRIPING UNIT IS A SINGLE BIT

IDEA:
$$\begin{array}{r} 1\ 0\ 0\ 1 \\ \oplus\ 0\ 1\ 0\ 1 \\ \hline 1\ 1\ 0\ 0 \end{array}$$
← bit check

REALLY: HAMMING CODES
Hamming (7,4) is 3 parity bits for 4 data bits.
→ detect and correct 1 bit error

- 4 DATA DISKS, 3 CHECK DISKS

RAID 3: BIT INTERLEAVED PARITY
- single check disk with parity
- check scheme can't detect which failed, but controller knows.
- STRIPING UNIT IS 1 BIT
- READ, WRITE INVOLVES ALL DISKS.
- CAN PROCESS ONE REQUEST AT A TIME

26/2/2015 Buffer Management Pg 2

RAID 4: BLOCK INTERLEAVED PARITY
- STRIPING UNIT: ONE DISK BLOCK
  ONE CHECK DISK

- LARGE REQUEST CAN UTILIZE FULL AGGREGATE BANDWIDTH

- WRITE TO MODIFIED DISK AND CHECK DISK

RAID 5: BLOCK-INTERLEAVED, DISTRIBUTED PARITY

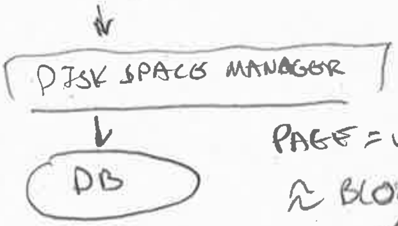
- SIMILAR TO LEVEL 4, BUT PARITY BLOCKS ARE DISTRIBUTED.
- BEST PERFORMANCE OF ALL RAID FOR SMALL/LARGE READ & LARGE WRITE.
- SMALL WRITE REQUIRES READ-MODIFY-WRITE CYCLE

WHICH RAID?

DEPENDS ON CONCERN FOR DATA LOSS AND WORKLOAD.
RAID 5 IS GOOD "GENERAL"
RAID 0 IS GOOD IF NOT WORRIED FOR LOSS
2 < 3, 4 < 5

III DISK SPACE MANAGEMENT

- allocate/de allocate page
- read/write page



PAGE = UNIT OF DATA
≈ BLOCK SIZE
(ASSUMPTION)

- REQUESTS FOR SEQUENCE OF PAGES (E.G. TO EXPLOIT SEQUENTIAL DISK BLOCK ACCESS) MUST BE HANDLED BY MANAGER

TRACK FREE BLOCKS

- MAINTAIN LIST OF "FREE" BLOCKS
- MAINTAIN BIT MAP OF FREE BLOCKS.

WHY NOT OS?

- FOR O.S., FILE IS SEQUENCE OF BYTES
- DB MAY WANT DIFFERENT FEATURES
- O.S. FILE SIZE MAY BE TOO SMALL
  E.G. 32 bit system, largest file is 4GB
- O.S. FILES CAN'T SPAN MULTIPLE DISK DEVICES.

IV BUFFER MANAGER

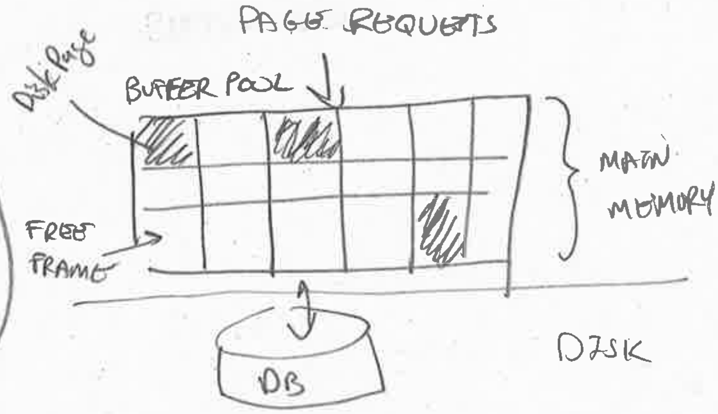


TABLE OF <FRAME #, PAGE ID#> pairs is maintained.

When a page is requested:

- If page is not in the pool:
  - Choose a frame for replacement
  - If frame is dirty, write to disk
  - Read requested page into frame

- Pin the page and return address

pin count = how many times the page has been requested, not released

dirty - if the page has been modified.

- Pages can be pre-fetched if you can predict work load

26/2/2015 Buffer Management Pg 3

- requester must "unpin"
- concurrency control and recovery adds additional steps (write ahead log)

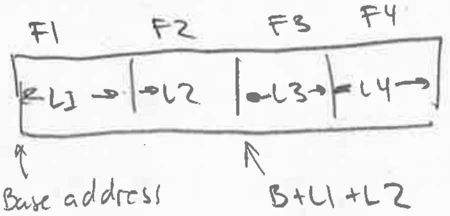
- Replacement policies
  - LRU - Least Recently Used
  - FIFO - first in, first out
  - MRU - most recently used
  - Clock - variation of LRU, see homework

Big impact:
example LRU and sequential scans

Why not O.S?
- portability
- limitations
- access to workload
- need to pin and force page to disk for CC & recovery

V Record Formats

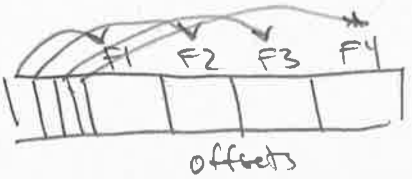
Fixed length:



finding ith field = no scan

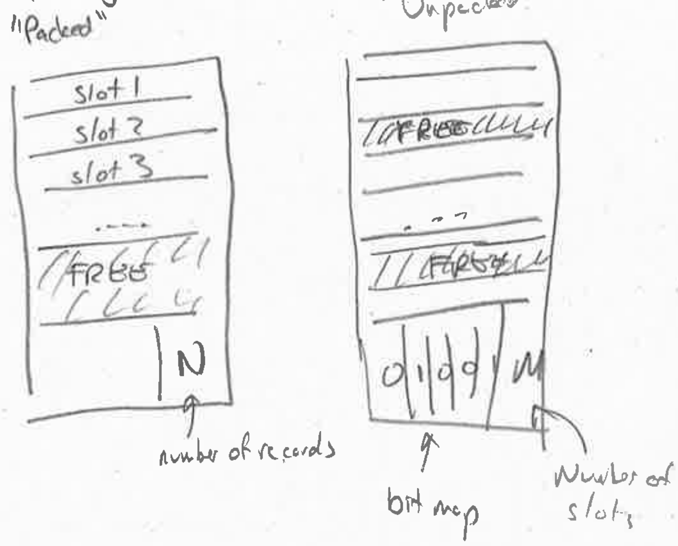
Variable length: (fixed # of fields)

- or -



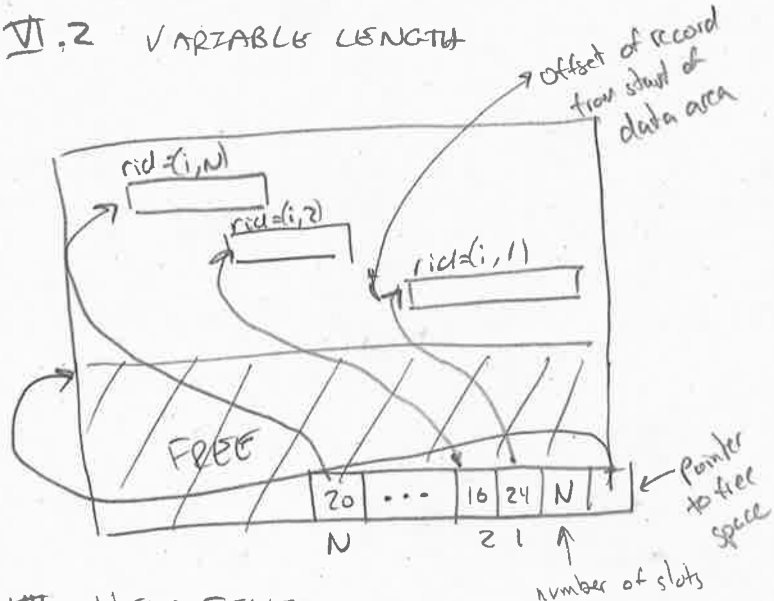
VI Page Formats

VI.1 Fixed length records



Record id = <page id, slot #>
first approach can change RID.

VI.2 VARIABLE LENGTH



VII HEAP FILES

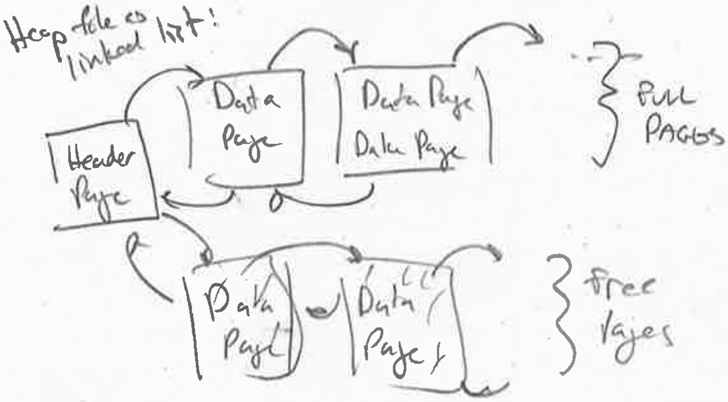
- FILE == COLLECTION OF PAGES == COLLECTION OF RECORDS
- DBMS operates on files and records
- Need to:
  - insert/delete/modify record
  - read a particular record
  - scan all records

26/2/2015 Buffer Management Pg 4

Unordered (Heap) Files

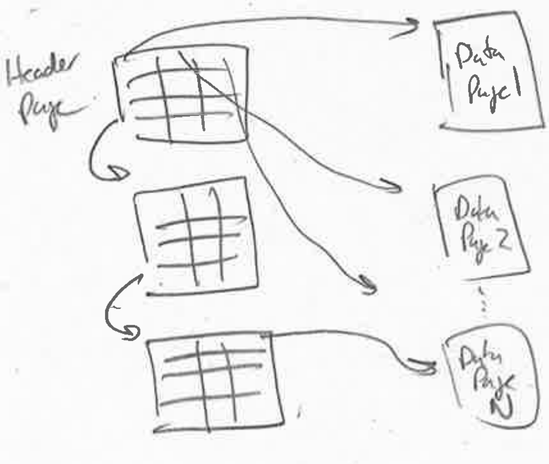
- simplest file structure, no order
- as file grows, shrinks, disk pages are allocated, de-allocated
- Need to:
  - track pages in a file
  - track free space on pages
  - track records on a page
- Many approaches/alternatives

Heap file as linked list:



- store header page id & heap file name somewhere
- Each page has 2 pointers
- Problem: for variable length records, every page will have free space

Heap File as Page Directory



Each directory entry == sequence of pages

- Use bit ~~for~~ 1 free space ~~and~~ count of available space
- can quickly search for pages with enough free space