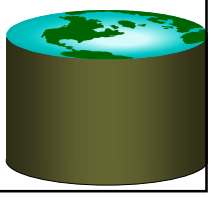
Storing Data: Disks and Files

CS 186 Fall 2002, Lecture 15
(R&G Chapter 7)

"Yea, from the table of my memory
I'll wipe away all trivial fond records."
– Shakespeare, *Hamlet*



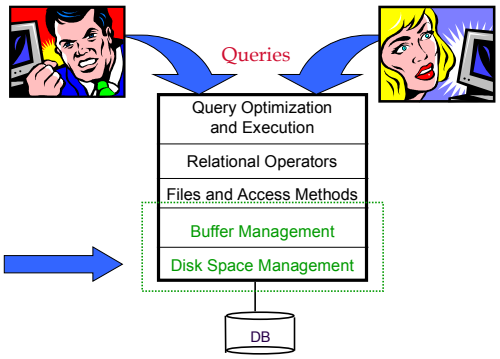
Stuff

- **Rest of this week**
  - My office hours cancelled this Thursday (10/17)
    - Send mail if this is a problem
  - Sirish Chandrasekaran will cover class on Tuesday
- **Project Phase 2**
  - Due Wednesday at 5pm
  - Example testing script posted --- make sure it works!
- **Project Phase 3 – Extended Data Types**
  - Out next wee
- **Midterm**
  - See next slide

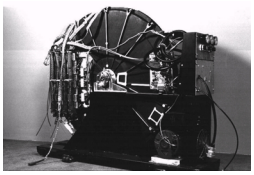
Review and Context

- **First half of course – how to use a database**
  - Aren't Databases Great?
  - Data Modeling with ER
  - Relational Model and Query Languages
- **Rest of the course – A peek under the hood.**
  - Where are the bits?
  - How to find them and keep track of them?
  - How to process those pesky SQL queries
  - How to ensure Transactional Semantics

The BIG Picture



Disks and Files

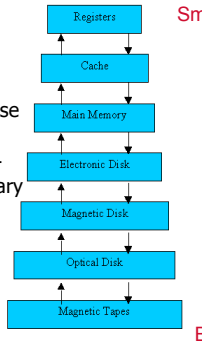


- **DBMS stores information on disks.**
  - In an electronic world, disks are a mechanical anachronism!
- **This has major implications for DBMS design!**
  - READ: transfer data from disk to main memory (RAM).
  - WRITE: transfer data from RAM to disk.
  - Both are high-cost operations, relative to in-memory operations, so must be planned carefully!

Why Not Store It All in Main Memory?

- ***Costs too much.*** **$100 will buy you either 0.5 GB of RAM or 100 GB of disk (EIDI/ATA) or 20GB (SCSI) or today.**
  - High-end Databases today in the 10-100 TB range.
  - Approx 60% of the cost of a production system is in the disks.
- ***Main memory is volatile.*** **We want data to be saved between runs. (Obviously!)**
- **Note, some specialized systems do store entire database in main memory.**
  - Vendors claim 10x speed up vs. traditional DBMS running in main memory.

The Storage Hierarchy



Smaller, Faster

–Main memory (RAM) for currently used data.

–Disk for the main database (secondary storage).

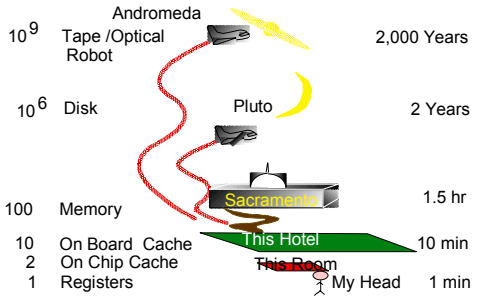
–Tapes for archiving older versions of the data (tertiary storage).

QUESTION: Why does it have to be a hierarchy?

Bigger, Slower

Source: Operating Systems Concepts 5th Edition

Jim Gray's Storage Latency Analogy: How Far Away is the Data?



Disks

- **Secondary storage device of choice.**
- **Main advantage over tapes: *random access* vs. *sequential*.**
  - Also, they work. (Tapes deteriorate over time)
- **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!

Anatomy of a Disk

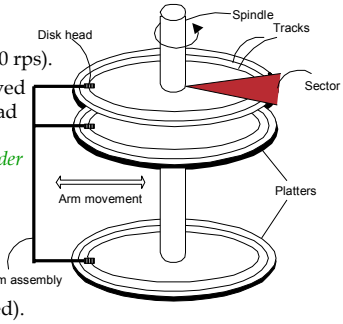
The platters spin (say, 150 rps).

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.

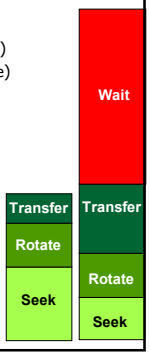
❖ *Block size* is a multiple of *sector size* (which is fixed).

❖Newer disks have several "zones", with more data on outer tracks.



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 < 1msec per 4KB page
- **Key to lower I/O cost:**
  reduce seek/rotation delays!
  Hardware vs. software solutions?
- **Also note: For shared disks most time spent waiting in queue for access to arm/controller**



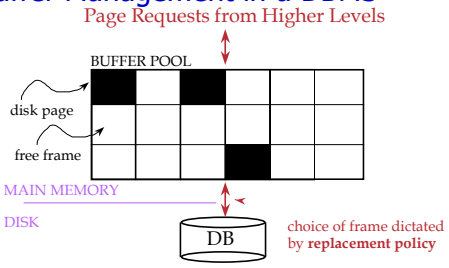
Arranging Pages on Disk

- **`*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!**
- **Also, modern controllers do their own caching.**

Disk Space Management

- **Lowest layer of DBMS software manages space on disk (using OS file system or not?).**
- **Higher levels call upon this layer to:**
  - allocate/de-allocate a page
  - read/write a page
- **Best if a request for a *sequence* of pages is satisfied by pages stored sequentially on disk! Higher levels don't need to know if/how this is done, or how free space is managed.**

Buffer Management in a DBMS



- ***Data must be in RAM for DBMS to operate on it!***
- ***Buffer Mgr hides the fact that not all data is in RAM***

When a Page is Requested ...

- **Buffer pool information table contains:** ***<frame#, pageid, pin_count, dirty>***
- **If requested page is not in pool:**
  - Choose a frame for *replacement* *(only un-pinned pages are candidates)*
  - If frame is "dirty", 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, 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 (*"unpinned"*)
- **CC & recovery may entail additional I/O when a frame is chosen for replacement. (*Write-Ahead Log* protocol; more later.)**

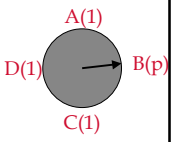
Buffer Replacement Policy

- **Frame is chosen for replacement by a *replacement policy:***
  - Least-recently-used (LRU), MRU, Clock, etc.
- **Policy can have big impact on # of I/O's; depends on the *access pattern*.**

LRU Replacement Policy

- ***Least Recently Used (LRU)***
  - for each page in buffer pool, keep track of time last *unpinned*
  - replace the frame which has the oldest (earliest) time
  - very common policy: intuitive and simple
- **Problems?**
- ***Problem: Sequential flooding***
  - 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).

"Clock" Replacement Policy



- **An approximation of LRU.**
- **Arrange frames into a cycle, store one "reference bit" per frame**
- **When pin count goes to 0, reference bit set on.**
- **When replacement necessary:**

```
do {
    if (pincount == 0 && ref bit is off)
        choose current page for replacement;
    else if (pincount == 0 && ref bit is on)
        turn off ref bit;
    advance current frame;
} until a page is chosen for replacement;
```

Questions:
How like LRU?
Problems?

DBMS vs. OS File System

OS does disk space & buffer mgmt: why not let OS manage these tasks?

- **Some limitations, e.g., files can't span disks.**
  - Note, this is changing --- OS File systems are getting smarter (i.e., more like databases!)
- **Buffer management in DBMS requires ability to:**
  - pin a page in buffer pool, force a page to disk & order writes (important for implementing CC & recovery)
  - adjust *replacement policy,* and pre-fetch pages based on access patterns in typical DB operations.

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.