


Physical Data Organization

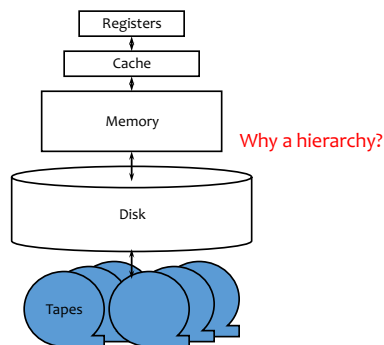
Introduction to Databases
CompSci 316 Fall 2016



Outline

- It's all about disks!
 - That's why we always draw databases as 
 - And why the single most important metric in database processing is (oftentimes) the number of disk I/O's performed
- Storing data on a disk
 - Record layout
 - Block layout

Storage hierarchy



How far away is data?

| Location | Cycles | Location | Time |
|----------------|--------|-----------------|----------|
| Registers | 1 | My head | 1 min. |
| On-chip cache | 2 | This room | 2 min. |
| On-board cache | 10 | Duke campus | 10 min. |
| Memory | 100 | Washington D.C. | 1.5 hr. |
| Disk | 10^6 | Pluto | 2 yr. |
| Tape | 10^9 | Andromeda | 2000 yr. |

Computer time
clock ticks – 5ns

Human time

(Source: AlphaSort paper, 1995)
The gap has been widening!

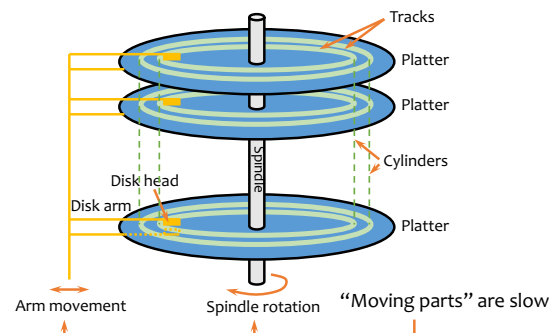
☞ I/O dominates—design your algorithms to reduce I/O!

A typical hard drive



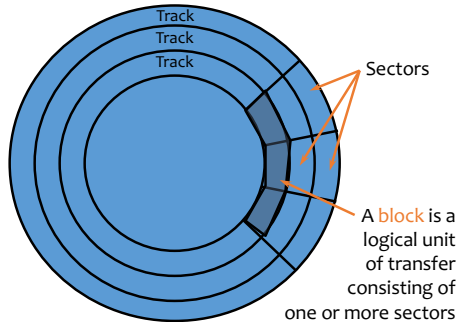
<http://upload.wikimedia.org/wikipedia/commons/0/08/Laptop-hard-drive-exposed.jpg>

A typical hard drive



Top view

“Zoning”: more sectors/data on outer tracks



Disk access time

Sum of:

- **Seek time:** time for disk heads to move to the correct cylinder
- **Rotational delay:** time for the desired block to rotate under the disk head
- **Transfer time:** time to read/write data in the block (= time for disk to rotate over the block)

Random disk access

Seek time + rotational delay + transfer time

- Average seek time
 - “Typical” value: 5 ms
- Average rotational delay
 - Time for a half rotation (a function of RPM)
 - “Typical” value: 4.2 ms (7200 RPM)

Sequential disk access

Seek time + rotational delay + transfer time

- Seek time
 - 0 (assuming data is on the same track)
- Rotational delay
 - 0 (assuming data is in the next block on the track)
- Easily an order of magnitude faster than random disk access!

What about SSD (solid-state drives)?



<http://www.techgoonda.com/wp-content/uploads/2012/12/SSD-6-25-121.jpg>

What about SSD (solid-state drives)?

- No mechanical parts
- Mostly flash-based nowadays
- 1-2 orders of magnitude faster random access than hard drives (under 0.1ms vs. several ms)
 - But still much slower than memory ($\sim 0.1\mu\text{s}$)
- Little difference between random vs. sequential read performance
- Random writes still hurt
 - In-place update would require erasing the whole “erasure block” and rewriting it!

Important consequences

- It's all about reducing I/O's!
- Cache blocks from stable storage in memory
 - DBMS maintains a memory **buffer pool** of blocks
 - Reads/writes operate on these memory blocks
 - Dirty (updated) memory blocks are "flushed" back to stable storage
- Sequential I/O is much faster than random I/O

Performance tricks

- Disk layout strategy
 - Keep related things (what are they?) close together: same sector/block → same track → same cylinder → adjacent cylinder
- Prefetching
 - While processing the current block in memory, fetch the next block from disk (overlap I/O with processing)
- Parallel I/O
 - More disk heads working at the same time
- Disk scheduling algorithm
 - Example: "elevator" algorithm
- Track buffer
 - Read/write one entire track at a time

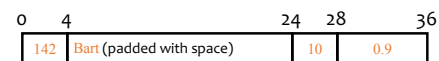
Record layout

Record = row or tuple in a table

- Variable-format records
 - Rare in DBMS—table schema dictates the format
 - Relevant for semi-structured data such as XML
- Focus on fixed-format records
 - With fixed-length fields only, or
 - With possible variable-length fields

Fixed-length fields

- All field lengths and offsets are constant
 - Computed from schema, stored in the system catalog
- Example: CREATE TABLE User(uid INT, name CHAR(20), age INT, pop FLOAT);



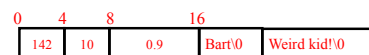
- Watch out for alignment
 - May need to pad; reorder columns if that helps
- What about NULL?
 - Add a bitmap at the beginning of the record

Variable Length Records - motivation

- In practice,
 1. Data size may vary
 - address, name
 2. Repeating fields
 - e.g. pointers for a many-many relationship
 3. Variable format records
 - do not know at the beginning
 4. Enormous fields
 - like videos

Variable-length records

- Example: CREATE TABLE User(uid INT, name VARCHAR(20), age INT, pop FLOAT, comment VARCHAR(100));
- Approach 1: use field delimiters ('0' okay?)



- Approach 2: use an offset array



- Put all variable-length fields at the end (why?)
- Update is messy if it changes the length of a field

BLOB fields

- Binary Large Objects
- Must be stored on a sequence of blocks
- Example: `CREATE TABLE User(uid INT, name CHAR(20), age INT, pop FLOAT, picture BLOB(32000));`
- Student records get “de-clustered”
 - Bad because most queries do not involve picture
- Decomposition (automatically and internally done by DBMS without affecting the user)
 - (`uid`, `name`, `age`, `pop`)
 - (`uid`, `picture`)

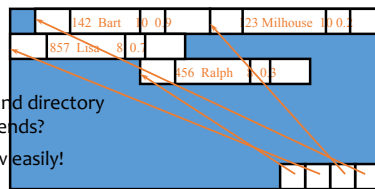
Block layout

How do you organize records in a block?

- **NSM** (N-ary Storage Model)
 - Most commercial DBMS
- **PAX** (Partition Attributes Across)
 - Ailamaki et al., VLDB 2001

NSM

- Store records from the beginning of each block
- Use a directory at the end of each block
 - To locate records and manage free space
 - Necessary for variable-length records



Why store data and directory at two different ends?

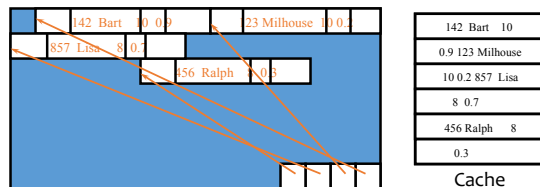
So both can grow easily!

Options

- Reorganize after every update/delete to avoid fragmentation (gaps between records)
 - Need to rewrite half of the block on average
- A special case: What if records are fixed-length?
 - Option 1: reorganize after delete
 - Only need to move one record
 - Need a pointer to the beginning of free space
 - Option 2: do not reorganize after update
 - Need a bitmap indicating which slots are in use

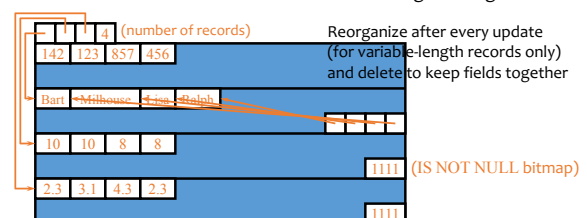
Cache behavior of NSM

- Query: `SELECT uid FROM User WHERE pop > 0.8;`
- Assumptions: no index, and cache line size < record size
- Lots of cache misses
 - `uid` and `pop` are not close enough by memory standards



PAX

- Most queries only access a few columns
- Cluster values of the same columns in each block
 - When a particular column of a row is brought into the cache, the same column of the next row is brought in together



Beyond block layout: column stores 25

- The other extreme: store tables by columns instead of rows
- Advantages (and disadvantages) of PAX are magnified
 - Not only better cache performance, but also fewer I/O's for queries involving many rows but few columns
 - Aggressive compression to further reduce I/O's
- More disruptive changes to the DBMS architecture are required than PAX
 - Not only storage, but also query execution and optimization

Summary 26

- Storage hierarchy
 - Why I/O's dominate the cost of database operations
- Disk
 - Steps in completing a disk access
 - Sequential versus random accesses
- Record layout
 - Handling variable-length fields
 - Handling NULL
 - Handling modifications
- Block layout
 - NSM: the traditional layout
 - PAX: a layout that tries to improve cache performance
- Column store: NSM transposed, beyond blocks