Outline

In implementing DBMS we need to answer

- How should the system store and manage very large amounts of data?
- What data structures should be used in order to support efficient manipulation of these data?

Focus on 1st question

- Consider memory hierarchy
- Algorithms to move data between main and secondary/tertiary memory
- Techniques to reduce read/write times
- Methods to improve reliability of disks
- Introduce a block model of secondary storage
**Storage Devices**

We can classify storage devices on:

- Speed to access the data
- Cost per byte
- Reliability
- Persistence
  - volatile storage
    - loses contents after power is removed
  - non-volatile storage
    - retains contents even after power is removed
Memory Hierarchy

- Cache
- Main memory
- Virtual memory
- Secondary storage
  - Floppies
  - Hard-disks
- Tertiary storage
  - Ad-hoc tape storage (heap of tapes)
  - Optical-disk juke boxes
  - Tape silos
Disks

Physical characteristics

- Rotation speed (e.g., 7200rpm)
- #platters per unit
- #tracks per surface
- #sectors per track
- #bytes per sector
- #bits per inch (radial, peripheral)
  - Inner tracks will have higher density if capacity of all tracks is fixed
Disks

Figure from DSC
Disk Access Characteristics

- **Latency to move data between memory and disk**
  - Read data from disk
    - Seek time
      - Max is linear in #cylinders crossed
      - Avg #cylinders = 1/3 of total #cylinders
    - Rotational latency
    - Transfer time
    - Other delays
      - CPU time to issue I/O
      - Contention for controller
      - Contention for bus, memory
  - Write data to disk
    - Same as read
    - Verifying data written doubles latency
Disk Access

- Random access to disk blocks is expensive, while sequential access is much cheaper
  - ~20ms vs ~1ms for 1KB blocks
- Block pointers
  - Dealing with bad blocks is messy (requires some kind of map)
Using disks efficiently

- Often, the primary cost for operations on large amounts of data is disk I/O

- Merge-Sort for Secondary storage
  - Two-phase Multi-way Merge-sort
    - partition data into segments that fit in memory – sort each segment
    - (multi-way) merge segments to get progressively larger segments
  - For example, with memory of size M, blocks of size B, and records of size R
    - we can sort a relation with up to \( N = \frac{M}{R} \left( \frac{M}{B} - 1 \right) \) records by doing \( 4NR/B \) disk I/Os
Accelerating access to disks

- Organize data by cylinders since the seek time for adjacent cylinders is much smaller than for random cylinders
  - Good when access is predictable and one process is using the disk
- Use multiple smaller disks to get parallelism as long the other delays are kept small
  - Watch for contention for disk controller, bus, and main memory
  - Placement of data onto disks critical to get parallelism from using many disks
  - Cost of multiple disks is larger than the cost of single larger disk
Accelerating access to disks

- Use mirroring to disks
  - Decreases read latency for batches of blocks due to parallel reads, but does not affect write latency due to sequential writes
  - Improves fault tolerance
  - Pay for more capacity than is effectively used

- Use an appropriate disk-scheduling algorithm that decides in what order outstanding requests for disk blocks will be serviced
  - Elevator algorithm
  - It is most effective when there are many outstanding requests
Accelerating access to disks

- **Prefetching**
  - when reading a block, prefetch a few more neighboring blocks, while
  - Good when spatial locality in block accesses exists, but timing of requests if unpredictable

- **Double buffering**
  - overlap the processing of one block with the reading of another block

- Both requires extra main memory
**Disk failures**

- **Intermittent failures**
  - Attempt to access a block fails, but repeated tries succeed

- **Checksums**
  - Detect failure to read/write correctly using checksum bits in each sector

- **Media decay**
  - Some bits are permanently corrupted

- **Write failure**
  - Can not rewrite or read a sector

- **Stable storage**
  - use two actual sectors to store the contents of one and following a write-all-read-one strategy
  - Handles media decay and write failures
Disk failures

- **Disk crashes**
  - Suddenly entire disk becomes permanently inaccessible
  - Mirroring can help with crashes but may use too many extra disks
  - Redundancy together with error-correcting coding techniques can handle crashes without substantial increase in the number of extra disks (e.g., RAID)