Storage and File Structure

December 12, 2008

Classifying Physical Storage Media

- Data access speed

Classifying Physical Storage Media

- Data access speed
- Cost (per unit data)

Classifying Physical Storage Media

- Data access speed
- Cost (per unit data)
- Reliability
  - Power loss
  - Physical failure

Classifying Physical Storage Media

- Data access speed
- Cost (per unit data)
- Reliability
  - Power loss
  - Physical failure
- Storage lifetime
  - **volatile**
  - **non-volatile**

Storage Media Types

- **Cache**: fastest, costliest
  - volatile
  - non-accessible

Storage Media Types

- **Cache**: fastest, costliest
  - volatile
  - non-accessible
- **Main Memory**:
  - fast
  - few GBs
    - usually not enough for a DB
  - volatile

Storage Media Types

- **Cache**: fastest, costliest
  - volatile
  - non-accessible
- **Main Memory**:
  - fast
  - few GBs
    - usually not enough for a DB
  - volatile
- **Flash Memory**:
  - non-volatile
  - fast reads (close to main memory), slow writes
  - maximum 10K - 1M write/erase cycle

Storage Media Types

- **Magnetic disk**
  - spinning disks, magnetic write/reads
  - long-term storage; stores complete DB
  - data trasfer to main memory
  - random access (unlike magnetic tape)
  - non-volatile; delicate physical structure

Storage Media Types

- **Optical storage**
  - non-volatile; optical read/write on a spinning disk
  - 640 MB – 4.7/14 GB – 50 GB
  - very slow read/writes

Storage Media Types

- **Tape storage**
  - non-volatile
  - sequential access, removable; extremely slow
  - high capacity (jukeboxes for 1 petabyte or more)
  - backups / archival data

Storage Hierarchy

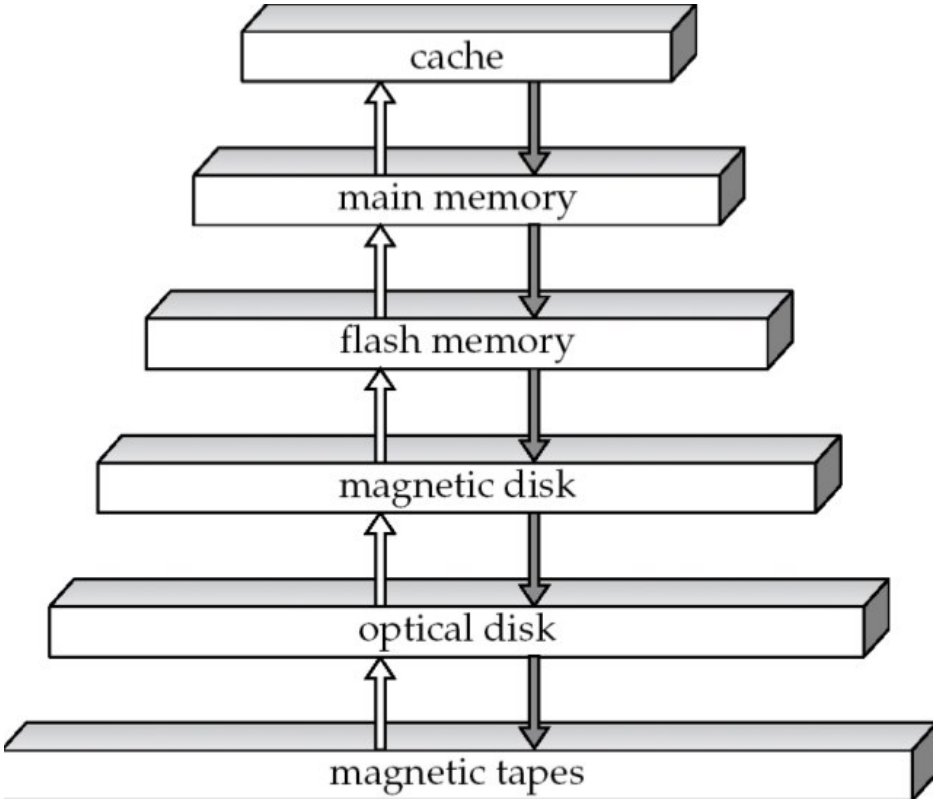


Figure: Storage Hierarchy

Storage Hierarchy

- **Primary storage**: cache, main memory
  - fastest, volatile
- **Secondary storage**: flash, magnetic discs
  - *online* storage
  - non-volatile
  - moderately fast
- **Tertiary storage**: magnetic tape, optical storage
  - slowest, non-volatile
  - backup, archival purposes

Outline

Physical Characteristics

Figure: Magnetic Hard Disk

Physical Characteristics

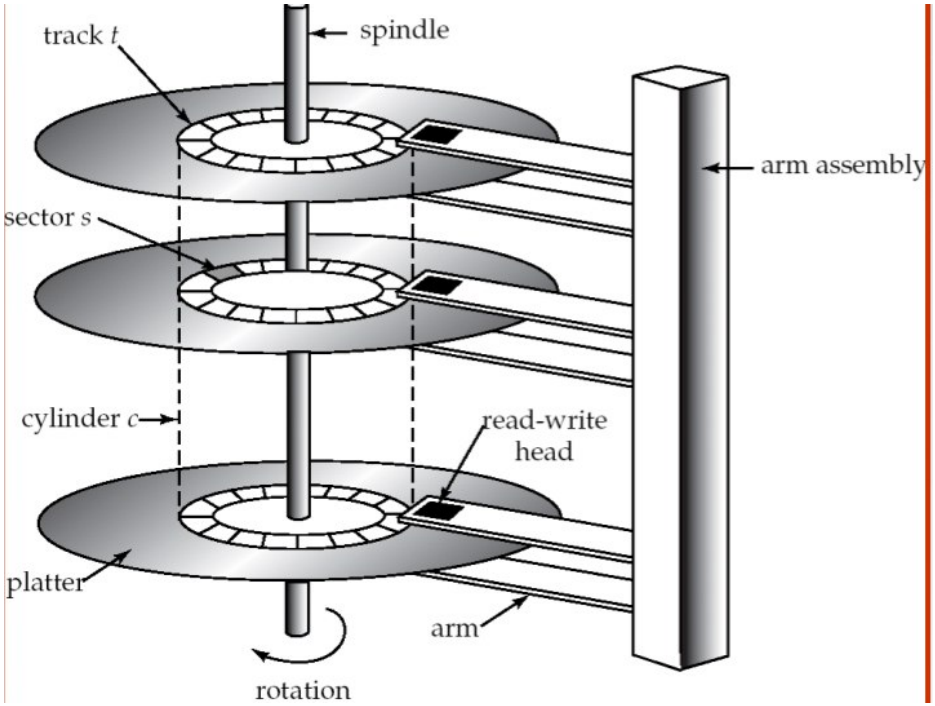


Figure: Magnetic Hard Disk

Magnetic Disk Functioning

- Read-write head: magnetic encoding
- Platters → tracks → sectors
- Corresponding tracks in all platters → a cylinder

Performance Measures

- **Access time**: request (read/write) to data transfer
  - **seek time**: reposition the arm over correct track
  - **rotational latency**: move correct sector over the head
- **Data transfer rate**: read/write speed
  - 25 - 100 MB / sec
- **Mean Time To Failure**
  - average disk life without failure
  - 3 to 5 years

Block Access Optimization

- **Block**: contiguous sectors from a single track
  - 4 - 20 sectors
- Data transfer in blocks
- Disk-arm-scheduling algorithms
  - minimize arm movememnt
  - e.g. *elevator* algorithm

Block Access Optimization

- **File Organization**: minimize arm movement
  - Store related information on the same or nearby blocks/cylinders
    - files, folders

Block Access Optimization

- **File Organization**: minimize arm movement
  - Store related information on the same or nearby blocks/cylinders
    - files, folders
  - Data fragmentation

Block Access Optimization

- **File Organization**: minimize arm movement
  - Store related information on the same or nearby blocks/cylinders
    - files, folders
  - Data fragmentation
    - insertion, deletion
  - Defragmenting utilities

Block Access Optimization

- **Non-volatile write buffers**
  - Non-volatile RAM: flash or battery-backed RAM
- Data is written to NVRAM immediately → fast!
- Controller writes to disk whenever disk is free
- DB operations can continue without waiting for data write to complete
- Writes can be reordered to minimize arm movement

Block Access Optimization

- **Log disk**: disk devoted to writing a sequential log of block updates
  - Used like NVRAM
  - Writes are fast since no seek is required
  - No need for special hardware
- File systems reorder writes to disk to improve performance
  - **Journaling file systems** write data in safe order to NVRAM or log disk

Outline

RAID

- **R**edundant **A**rrays of **I**ndependent **D**isks
  - disk organization managing several disks with a single disk view

RAID

- **R**edundant **A**rrays of **I**ndependent **D**isks
  - disk organization managing several disks with a single disk view
- parallel operation ⇒ **high capacity, high speed**
- redundant storage ⇒ **high reliability**

RAID

- **R**edundant **A**rrays of **I**ndependent **D**isks
  - disk organization managing several disks with a single disk view
- parallel operation ⇒ **high capacity**, **high speed**
- redundant storage ⇒ **high reliability**
- Probability of one disk failure out of N > that for single disk failure
  - System of 100 disks, each with MTTF of 100,000 hrs (11 years), will have system MTTF of 1000 hours (41 days)
- Better: store data redundantly

Improving Reliability with Redundancy

- **Redundancy**: store extra information that can be used to rebuild data after loss
- Extreme e.g.: **Mirroring** (or shadowing)
  - duplicate all data (disks)
  - 1 logical disk = 2 physical disks
    - write twice, read from anywhere
    - loss if combined failure (very low probability, except for fire, etc.)

Improving Performance with Parallelism

- Goals:
  - load balance multiple small accesses to increase throughput
  - parallelize large access to reduce response time
- Improve transfer speed by striping data across disks

Improving Performance with Parallelism

- Goals:
  - load balance multiple small accesses to increase throughput
  - parallelize large access to reduce response time
- Improve transfer speed by striping data across disks
- **Bit-level Striping**: split bits of a byte across available disks
  - with parallel access, 8 times the speed

Improving Performance with Parallelism

- Goals:
  - load balance multiple small accesses to increase throughput
  - parallelize large access to reduce response time
- Improve transfer speed by striping data across disks
- **Bit-level Striping**: split bits of a byte across available disks
  - with parallel access, 8 times the speed
- **Block-level Striping**: split blocks of data across disks
  - block i of a file goes to disk (i `mod` n) + 1
  - for large reads: read n blocks in parallel from n disks
  - for single block read: 1 disk used, rest perform other operations

RAID Levels

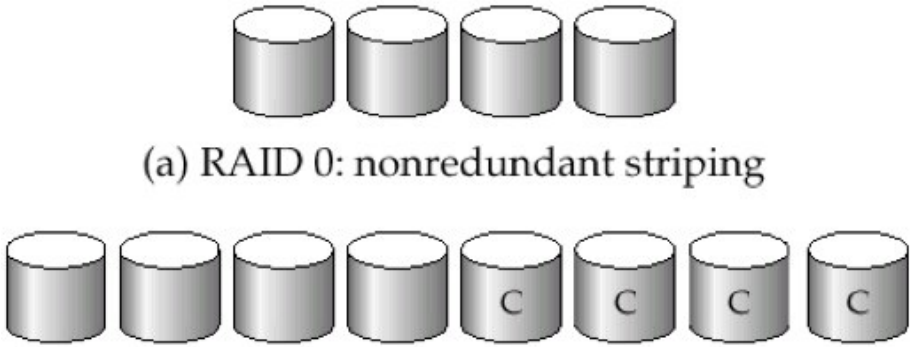
- Issues:
  - mirroring improves reliability, but expensive
  - striping improves data transfer rates, but not reliability

RAID Levels

- Issues:
  - mirroring improves reliability, but expensive
  - striping improves data transfer rates, but not reliability
- RAID Levels or organizations
  - different cost, performance, reliability

RAID Levels

- Issues:
  - mirroring improves reliability, but expensive
  - striping improves data transfer rates, but not reliability
- RAID Levels or organizations
  - different cost, performance, reliability



(a) RAID 0: nonredundant striping

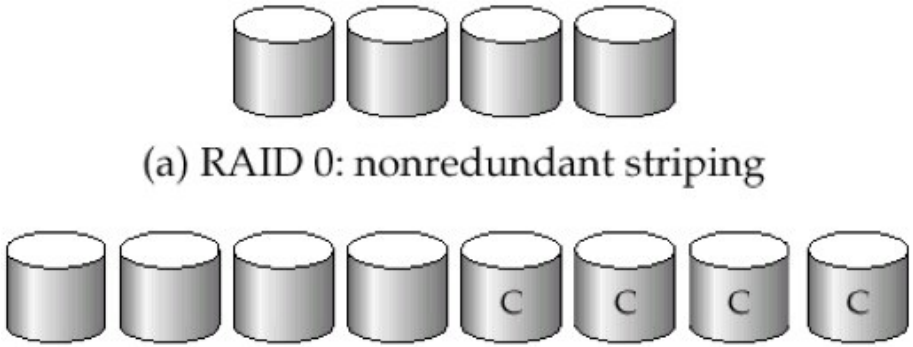
(b) RAID 1: mirrored disks

Figure: Levels 0 and 1

- **RAID Level 0**: block-striping, non-redundant
  - for high-performance
  - data-loss is not critical

RAID Levels

- Issues:
  - mirroring improves reliability, but expensive
  - striping improves data transfer rates, but not reliability
- RAID Levels or organizations
  - different cost, performance, reliability



(a) RAID 0: nonredundant striping

(b) RAID 1: mirrored disks

Figure: Levels 0 and 1

- **RAID Level 0**: block-striping, non-redundant
  - for high-performance
  - data-loss is not critical
- **RAID Level 1**: mirrored disks with block striping
  - good write performance
  - e.g., for storing log files in a DB system

RAID Levels

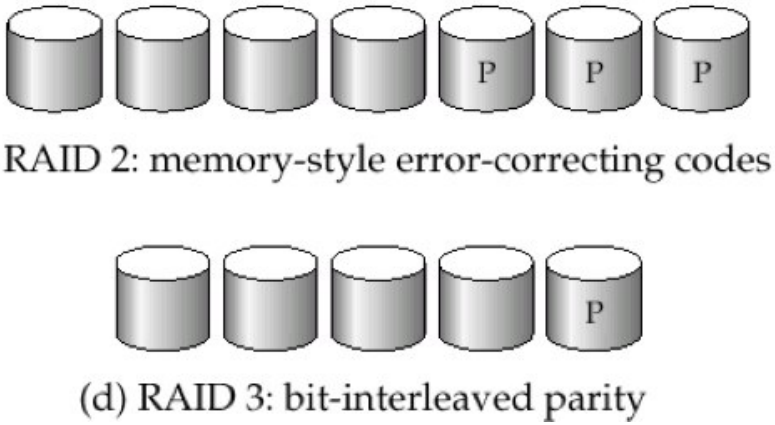
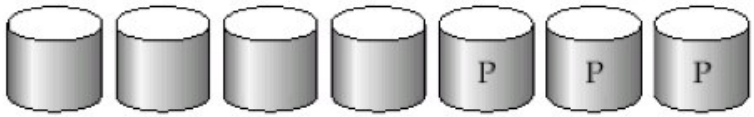


Figure: Raid Levels 2 and 3

- **RAID Level 2**: Memory-style Error-Correcting Codes (ECC) with bit striping
  - use 1-bit parity code: *detect* 1-bit errors
  - use 3-bit code: *correct* 1-bit errors

RAID Levels



(c) RAID 2: memory-style error-correcting codes

(d) RAID 3: bit-interleaved parity

Figure: Raid Levels 2 and 3

- **RAID Level 2**: Memory-style Error-Correcting Codes (ECC) with bit striping
  - use 1-bit parity code: *detect* 1-bit errors
  - use 3-bit code: *correct* 1-bit errors
- **RAID Level 3**: Bit-Interleaved Parity
  - 1 parity bit is enough for error correction, not just detection
    - Corresponding parity bits for data are computed and written to a parity bit disk
    - To recover data: compute XOR of bits from other disks (including parity bit disk)

RAID Levels

- RAID Level 3 (continued)
  - Faster than single disk
  - Fewer I/O /sec since every disk involved in I/O

RAID Levels

- RAID Level 3 (continued)
  - Faster than single disk
  - Fewer I/O /sec since every disk involved in I/O
- **RAID Level 4**: block inter-leaved parity

Figure: RAID Level 4

RAID Levels

- RAID Level 3 (continued)
  - Faster than single disk
  - Fewer I/O /sec since every disk involved in I/O
- **RAID Level 4**: block inter-leaved parity

Figure: RAID Level 4

- **RAID Level 5**: block-interleaved distributed parity

Figure: RAID Level 5

Choosing RAID Levels

- Factors:
  - monetary cost
  - performance
  - failure performance
  - rebuild performance

Choosing RAID Levels

- Factors:
  - monetary cost
  - performance
  - failure performance
  - rebuild performance
- RAID 0: if data safety is outsourced
- RAID 2 and 4: subsumed by 3 and 5 $\Rightarrow$ not used
- RAID 3: bit-striping needs all-disk access for single block write
  - not used
- Choose between 1 and 5

RAID Level 1 or 5

- Level 1: much better write performance
  - Level 5 needs 2 block reads + 2 block writes for single block write
  - Level 1 needs 2 block writes
  - Level 1 for high update environments, e.g. log disks

Hardware Issues

- **Software RAID**: Implementation in software
- **Hardware RAID**: Implementation by special hardware
- **Hot Swapping**: replacing a drive without power down
  - supported by some hardware RAID systems
  - reduces time to recovery, increases availability
- *Spare disks*: kept online, brought in as replacements on disk failure
- *Multiple controllers*
- *Redundant battery backups*

Outline

Database Storage

- DB file split into fixed length **blocks**
- DB system: minimize `disk` ↔ `memory` block transfer
  - keep as many blocks in memory as possible
- **Buffer**: memory portion for storing disk blocks copies
- **Buffer manager**: DB system for managing buffer

Buffer Manager

- DB calls BM when disk block is needed
- BM Tasks:
- *if* block in buffer ⇒ return memory address
- *if* block not in buffer ⇒
  - allocate space in buffer for block:
    - deallocate some old blocks from buffer if not enough space
    - deallocated blocks written to disk if and only if it was modified
  - read from disk to buffer, return address for memory space

Buffer-Replacemenet Strategies

- Most BM replace the block *least-recently used* (LRU strategy)
- *Toss-immediate strategy*: free a block when last tuple in the block has been processed
- BM also uses statistical information
  - e.g.: heuristic: data dictionary often used ⇒ keep DD blocks in memory

File Organization

- DB stored as a collection of files
- A file → records (rows)
- A record → fields (columns)

File Organization

- DB stored as a collection of files
- A file → records (rows)
- A record → fields (columns)
- *Simple approach*: assume each record is fixed length
  - ⇒ *fixed length records*

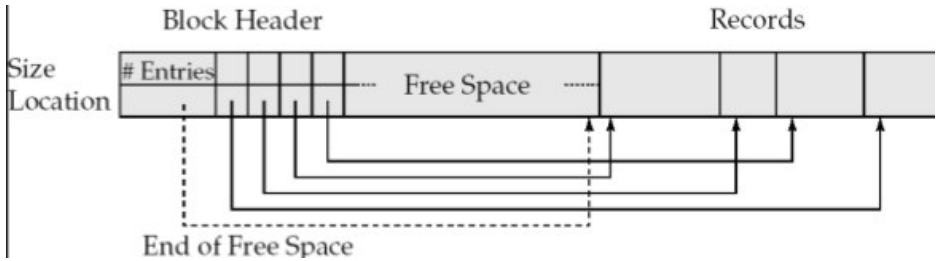
record 0	A-102	Perryridge	400
record 1	A-305	Round Hill	350
record 2	A-215	Mianus	700
record 3	A-101	Downtown	500
record 4	A-222	Redwood	700
record 5	A-201	Perryridge	900
record 6	A-217	Brighton	750
record 7	A-110	Downtown	600
record 8	A-218	Perryridge	700

  - each file stores records of one table
  - ∴, store record i at byte $n * (i - 1)$, where n is record size
  - record deletion requires book-keeping:
    - *record movement* or *list of free records*

Variable-length Records

- Can occur for several reasons
  - storing multiple records types in a file
  - single record type but with variable length fields
  - in older models → record types with repeating fields

Variable-length Records: Slotted Page Structure



- Slotted page header contains
  - number of record entries
  - end of free space in the block
  - location and size of each record
- records can be moved to keep them contiguous; header entry must be updated
- pointer to a record points to its entry in header; not to record itself