Chapter 10: Storage and File Structure

Self Reading Slides: 1 to 18, 29-49

Database System Concepts, 6th Ed.

Chapter 10: Storage and File Structure

- Overview of Physical Storage Media
- Magnetic Disks
- RAID
- Tertiary Storage
- Storage Access
- File Organization
- Organization of Records in Files
- Data-Dictionary Storage

Classification of Physical Storage Media

- Speed with which data can be accessed
- Cost per unit of data
- Reliability
  - Data loss on power failure or system crash
  - Physical failure of the storage device
- Can differentiate storage into
  - **Volatile storage**
    - Loses contents when power is switched off
  - **Non-volatile storage**
    - Contents persist even when power is switched off
    - Includes secondary and tertiary storage, as well as batter-backed up main-memory

Physical Storage Media

- **Cache**
  - Fastest and most costly form of storage; volatile; managed by the computer system hardware
- **Main memory**
  - Fast access (10s to 100s of nanoseconds; 1 nanosecond = 10^{-9} seconds)
  - Generally too small (or too expensive) to store the entire database
    - Capacities of up to a few Gigabytes widely used currently
    - Capacities have gone up and per-byte costs have decreased steadily and rapidly (roughly factor of 2 every 2 to 3 years)
  - **Volatile** — Contents of main memory are usually lost if a power failure or system crash occurs

Physical Storage Media (Cont.)

- **Flash memory**
  - Data survives power failure
  - Data can be written at a location only once, but location can be erased and written to again
    - Can support only a limited number (10K – 1M) of write/erase cycles.
    - Erasing of memory has to be done to an entire bank of memory
  - Reads are roughly as fast as main memory
  - But writes are slow (few microseconds), erase is slower
  - Widely used in embedded devices such as digital cameras, phones, and USB keys

Physical Storage Media (Cont.)

- **Magnetic-disk**
  - Data is stored on spinning disk, and read/written magnetically
  - Primary medium for the long-term storage of data; typically stores entire database.
  - Data must be moved from disk to main memory for access, and written back for storage
    - Much slower access than main memory (more on this later)
  - **Direct-access** – possible to read data on disk in any order, unlike magnetic tape
  - Capacities range up to roughly 1.5 TB as of 2009
    - Much larger capacity and cost/byte than main memory/flash memory
    - Growing constantly and rapidly with technology improvements (factor of 2 to 3 every 2 years)
  - Survives power failures and system crashes
    - disk failure can destroy data, but is rare

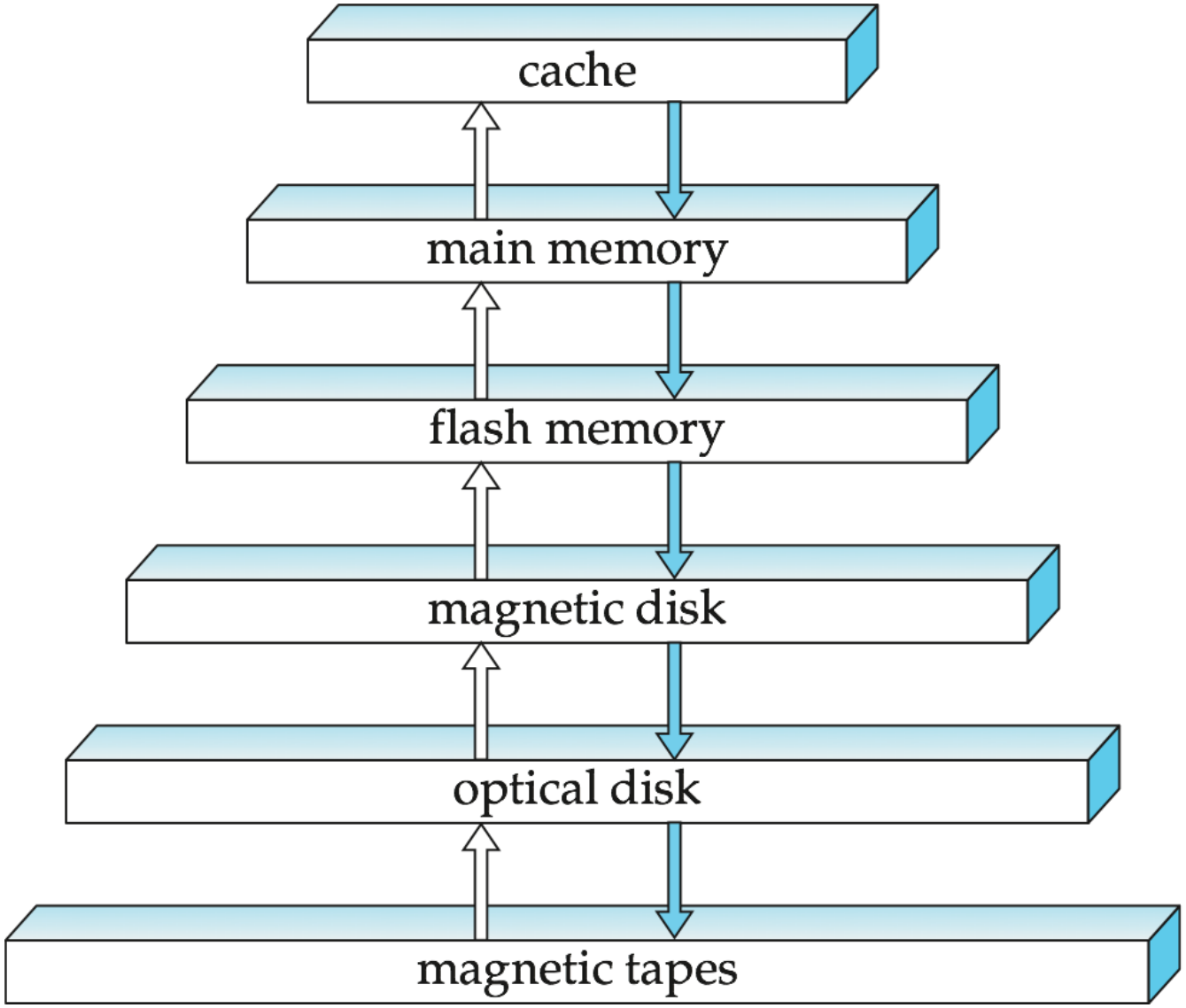
Physical Storage Media (Cont.)

- **Optical storage**
  - Non-volatile, data is read optically from a spinning disk using a laser
  - CD-ROM (640 MB) and DVD (4.7 to 17 GB) most popular forms
  - Blu-ray disks: 27 GB to 54 GB
  - Write-one, read-many (WORM) optical disks used for archival storage (CD-R, DVD-R, DVD+R)
  - Multiple write versions also available (CD-RW, DVD-RW, DVD+RW, and DVD-RAM)
  - Reads and writes are slower than with magnetic disk
  - **Juke-box** systems, with large numbers of removable disks, a few drives, and a mechanism for automatic loading/unloading of disks available for storing large volumes of data

Physical Storage Media (Cont.)

- **Tape storage**
  - Non-volatile, used primarily for backup (to recover from disk failure), and for archival data
  - **Sequential-access** – much slower than disk
  - Very high capacity (40 to 300 GB tapes available)
  - Tape can be removed from drive ⇒ storage costs much cheaper than disk, but drives are expensive
  - Tape jukeboxes available for storing massive amounts of data
    - Hundreds of terabytes (1 terabyte = 10^9 bytes) to even multiple **petabytes** (1 petabyte = 10^{12} bytes)

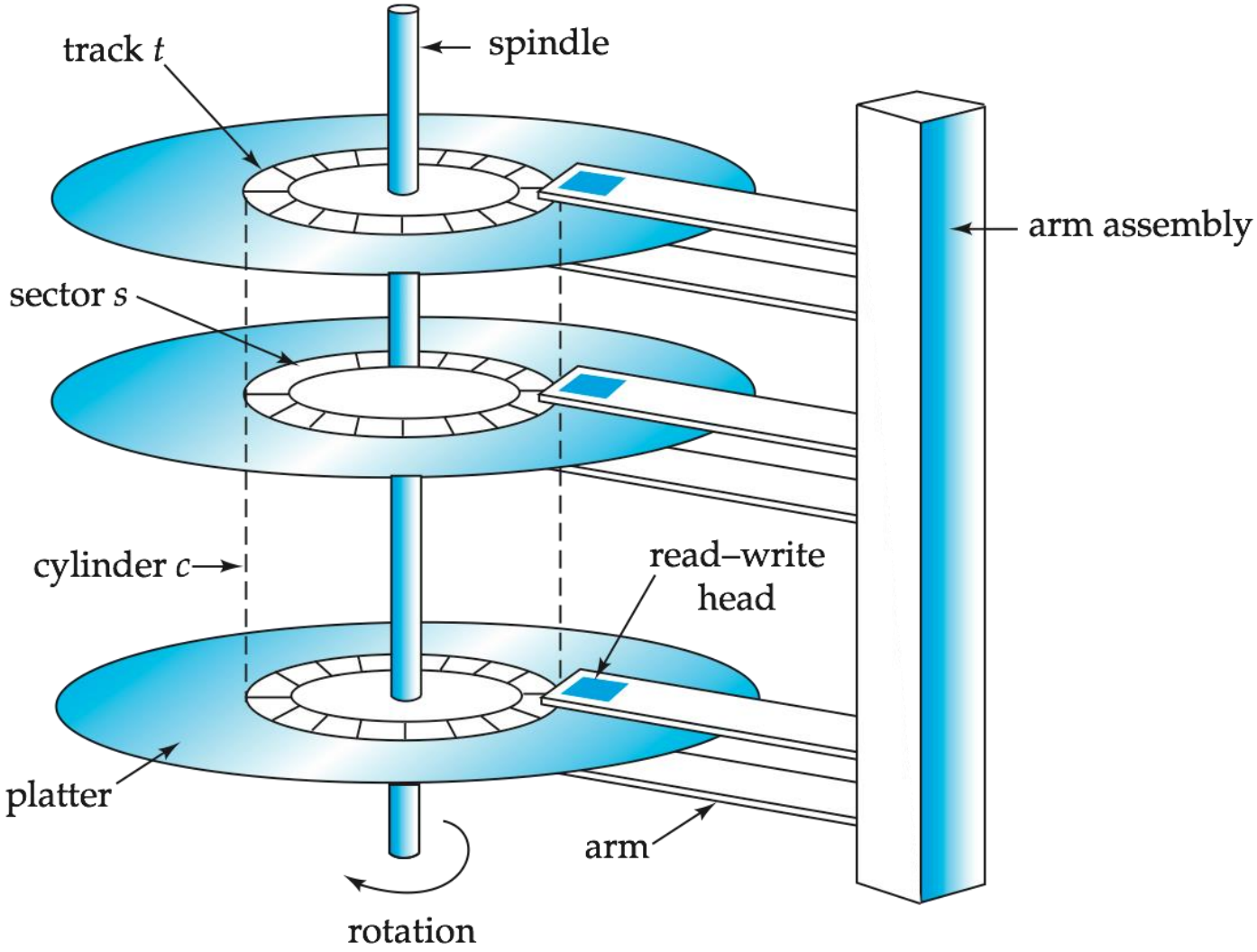
Storage Hierarchy



Storage Hierarchy (Cont.)

- **Primary storage:** Fastest media but volatile (cache, main memory)
- **Secondary storage:** Next level in hierarchy, non-volatile, moderately fast access time
  - Also called **on-line storage**
  - E.g. flash memory, magnetic disks
- **Tertiary storage:** Lowest level in hierarchy, non-volatile, slow access time
  - Also called **off-line storage**
  - E.g. magnetic tape, optical storage

Magnetic Hard Disk Mechanism



NOTE: Diagram is schematic, and simplifies the structure of actual disk drives

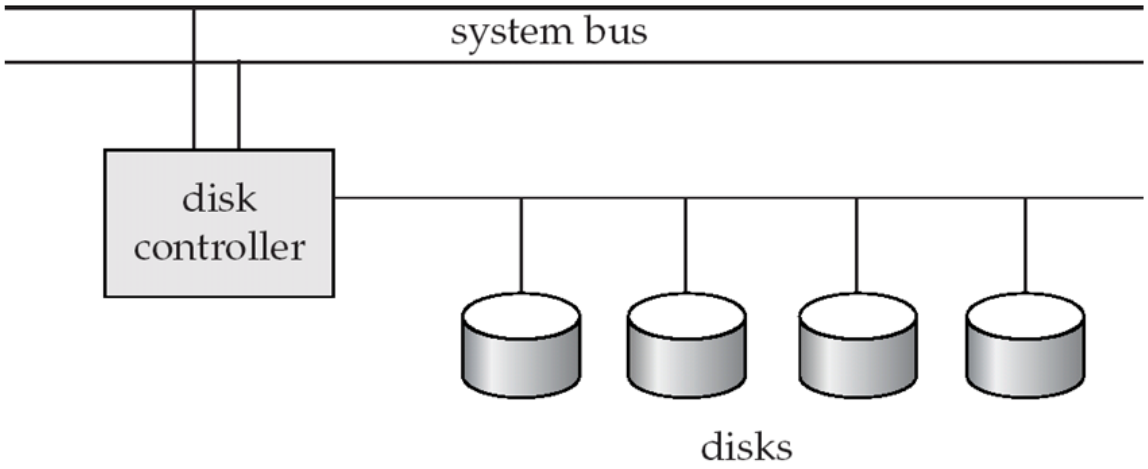
Magnetic Disks

- **Read-write head**
  - Positioned very close to the platter surface (almost touching it)
  - Reads or writes magnetically encoded information
- Surface of platter divided into circular **tracks**
  - Over 50K-100K tracks per platter on typical hard disks
- Each track is divided into **sectors**
  - A sector is the smallest unit of data that can be read or written
  - Sector size typically 512 bytes
  - Typical sectors per track: 500 to 1000 (on inner tracks) to 1000 to 2000 (on outer tracks)
- To read/write a sector
  - Disk arm swings to position head on right track
  - Platter spins continually; data is read/written as sector passes under head
- Head-disk assemblies
  - Multiple disk platters on a single spindle (1 to 5 usually)
  - One head per platter, mounted on a common arm
- **Cylinder** *i* consists of i^{th} track of all the platters

Magnetic Disks (Cont.)

- Earlier generation disks were susceptible to head-crashes
  - Surface of earlier generation disks had metal-oxide coatings which would disintegrate on head crash and damage all data on disk
  - Current generation disks are less susceptible to such disastrous failures, although individual sectors may get corrupted
- **Disk controller** – interfaces between the computer system and the disk drive hardware
  - Accepts high-level commands to read or write a sector
  - Initiates actions such as moving the disk arm to the right track and actually reading or writing the data
  - Computes and attaches **checksums** to each sector to verify that data is read back correctly
    - If data is corrupted, with very high probability stored checksum won't match recomputed checksum
  - Ensures successful writing by reading back sector after writing it
  - Performs remapping of bad sectors

Disk Subsystem



- Multiple disks connected to a computer system through a controller
  - Controllers functionality (checksum, bad sector remapping) often carried out by individual disks; reduces load on controller
- Disk interface standards families
  - ATA (AT adaptor) range of standards
  - SATA (Serial ATA)
  - SCSI (Small Computer System Interconnect) range of standards
  - SAS (Serial Attached SCSI)
  - Several variants of each standard (different speeds and capabilities)

Disk Subsystem

- Disks usually connected directly to computer system
- **Storage Area Networks (SAN)**
  - A large number of disks are connected by a high-speed network to a number of servers
- **Network Attached Storage (NAS)**
  - Networked storage provides a file system interface using networked file system protocol, instead of providing a disk system interface

Performance Measures of Disks

- **Access time**
  - The time it takes from when a read or write request is issued to when data transfer begins
  - Consists of:
    - **Seek time** – time it takes to reposition the arm over the correct track
      - Average seek time is 1/2 the worst case seek time
        - » Would be 1/3 if all tracks had the same number of sectors, and we ignore the time to start and stop arm movement
      - 4 to 10 milliseconds on typical disks
    - **Rotational latency** – time it takes for the sector to be accessed to appear under the head
      - Average latency is 1/2 of the worst case latency
      - 4 to 11 milliseconds on typical disks (5400 to 15000 r.p.m.)

Performance Measures of Disks

- **Data-transfer rate**
  - The rate at which data can be retrieved from or stored to the disk
  - 25 to 100 MB per second max rate, lower for inner tracks
  - Multiple disks may share a controller, so rate that controller can handle is also important
    - E.g. SATA: 150 MB/sec, SATA-II 3Gb (300 MB/sec)
    - Ultra 320 SCSI: 320 MB/s, SAS (3 to 6 Gb/sec)
    - Fiber Channel (FC2Gb or 4Gb): 256 to 512 MB/s

Performance Measures (Cont.)

- **Mean time to failure (MTTF)**
  - The average time the disk is expected to run continuously without any failure
  - Typically 3 to 5 years
  - Probability of failure of new disks is quite low, corresponding to a "theoretical MTTF" of 500,000 to 1,200,000 hours for a new disk
    - E.g., an MTTF of 1,200,000 hours for a new disk means that given 1000 relatively new disks, on an average one will fail every 1200 hours
  - MTTF decreases as disk ages

Disk Access

- On read from / write to the disk
  - Request for disk I/O generated from OS by
    - File system and by the Virtual Memory manager
    - Request specifies the Address on the disk to be referenced, as
      - Block number
- **Block** – Logical unit of a fixed number of contiguous sequence of sectors from a single track
  - Data is transferred between disk and main memory in blocks
  - Sizes range from 512 bytes to several kilobytes
    - Smaller blocks: more transfers from disk
    - Larger blocks : more space wasted due to partially filled blocks
    - Typical block sizes today range from 4 to 16 kilobytes

Disk-Block Access

- Sequence of requests of blocks from disk, classified as
  - **Sequential Access**
    - Successive block numbers requests are for blocks on the SAME or on ADJACENT TRACK
    - Disk seek required for the first block, but successive requests either not require a seek, or require a seek to an adjacent track
      - Faster than a seek to a track that is farther away
  - **Random Access**
    - Transfer rate is significantly lower than the sequential access
      - Successive requests are for blocks that are randomly located on the disk
      - Each such request require a seek
      - The number of random block accesses that can be satisfied by a single disk in a second depends on the seek time
      - Typically around 100 to 200 accesses per second
      - As only a small amount of data is read per seek

Optimization of Disk-Block Access

- Techniques to improve the speed of access to blocks
  - **1. Buffering**
    - Blocks from disk are read in main memory buffer, to satisfy future requests
    - Done by both the Operating system and DBMS
    - Discuss detail later on
  - **2. Read-Ahead**
    - On disk block access, consecutive blocks from the same track are read into an in-memory buffer even if there is no pending request for the blocks
      - If sequential access, then many blocks are ready in memory when they are requested and minimizes the disk seeks and rotational latency per block read
      - If random access, then not very useful

Optimization of Disk-Block Access

3. Scheduling

- If several blocks from a cylinder to be read from disk to main memory, then it save access time, if requesting blocks in the order in which they will pass under the heads
- If the desired blocks are on different cylinders, it is advantageous to request the pending blocks in an order that minimizes the disk-arm movement
  - **Disk-Arm Scheduling Algorithm**
    - Order the access to tracks to increase the number of accesses that can be processed
    - **Elevator algorithm**

Optimization of Disk-Block Access

- **Disk-Arm Scheduling Algorithm**
  - Order the access of tracks to increase the number of accesses that can be processed
  - **Elevator algorithm**
    - The arm is moving from innermost track toward the outside of the disk
    - During this, serve the access request
    - And continue till the farther out
    - Then change the direction and continue
    - Disk controllers reorders the read requests
      - Based on the organization of block on disk, of the rotational position of the disk platters and of the position of the disk-arm

Optimization of Disk Block Access (Cont.)

4. File organization

- Optimize block access time by organizing the blocks correspond to how data will be accessed
- i.e. Store related information on the same or nearby cylinders
  - Sequentially accessed file
- Files may get **fragmented** over time
  - E.g. if data is inserted to/deleted from the file
  - Or free blocks on disk are scattered, and newly created file has its blocks scattered over the disk
  - Sequential access to a fragmented file results in increased disk arm movement
- Some systems have utilities to defragment the file system, in order to speed up file access

Optimization of Disk Block Access (Cont.)

5. Nonvolatile Write Buffers

- Main memory content lost on power failure
- Database updates has to be recorded on disk to survive possible system crashes
- Non-Volatile RAM (NVRAM)
  - Speeds up disk writes by writing blocks to a NVRAM buffer immediately
  - Battery backed up RAM or flash memory
  - Even if power fails, the data is safe and will be written to disk when power returns

Optimization of Disk Block Access (Cont.)

- Non-Volatile RAM (NVRAM)
  - Database operations that require data to be safely stored before continuing can continue without waiting for data to be written to disk
  - Controller writes to disk whenever
    - The disk has no other requests / Pending request for some time
    - The NVRAM buffer becomes full
  - Found in RAID Controllers (Later)

Optimization of Disk Block Access (Cont.)

6. Log disk

- Another approach to reduce write latency
- A disk devoted to writing a sequential log of block updates
- Used exactly like NVRAM, But no need of special hardware NVRAM
- Write to log disk is very fast as number of blocks at a time
- Writes can be reordered to minimize disk arm movement
- On system crashes before some writes to the actual disk location have completed, when the system comes back up it reads the log disk to find those writes that had not been completed and carries out them out then

Optimization of Disk Block Access (Cont.)

- **Log Disk**
  - If no log disk, Risk of corruption of file system data
    - Old generation file system
  - **Journaling file systems**
    - In modern file system, Log disk requires file systems that support log disks to reorder writes to disk
    - Write data in safe order log disk
    - Improves performance, if separate data and log disk
    - Can be implemented even without a separate log disk
      - Keep data and the log on the same disk
      - Reduces the monetary cost at the expense of lower performance
    - Ch. 16 Recovery system

Flash Storage

- NOR flash
  - Allows random access to the individual WORDS of memory
- NAND flash
  - Used widely for storage
    - Since it is much cheaper than NOR flash and higher storage
  - Requires page-at-a-time read (page: 512 bytes to 4 KB-Approx. sector of mag. disk) into main memory
  - Transfer rate around 20 MB/sec
  - Can use multiple flash storage devices to provide higher transfer rate of 100 to 200 MB/sec

Flash Storage

- NAND flash
  - **Write**
    - Little more complicated takes few microseconds
    - Cannot be directly overwritten
      - Requires first erase and then allowed to write
      - Erase is very slow (1 to 2 milli secs)
      - Erase block contains multiple pages (erase block)
      - **Remapping** of logical page addresses to physical page addresses avoids waiting for erase
        - **Translation table** tracks mapping
        - Also stored in a label field of flash page
      - Remapping done by **flash translation layer software**
        - Provides Flash storage the identical look as magnetic disk storage to File system and DB Storage
        - Address mapping: logical to physical
        - Garbage collection and power off recovery
        - Wear leveling & bad block management

Flash Storage

- NAND flash
  - **Delete**
    - Blocks containing multiple deleted pages are periodically erased
    - After 100,000 to 1,000,000 erases, erase block becomes unreliable and cannot be used
    - **Wear leveling**
      - **Principle of evenly distributing erase operations across physical blocks**
        - Assigns “Cold data” i.e. data that are rarely updated to physical pages that have been **erased many times**
        - Assigns “Hot data” i.e. data that are updated frequently to physical pages that have not been erased many times
      - Done by flash memory controller
- **Hybrid Disk Drives**
  - Hard disk with combination of Magnetic storage with a smaller amount of flash memory
    - Flash memory used as a cache for frequently accessed data i.e. rarely updated

RAID

- Data storage requirement growing fast so needed a large no. of disks
- RAID: Redundant Arrays of Inexpensive Disks
  - Originally a cost-effective alternative to large, expensive disks as several small cheap disks
    - I in RAID originally stood for "inexpensive"
    - Today RAIDs are used for their higher reliability and bandwidth
      - The "I" is interpreted as independent
- **RAID: Redundant Arrays of Independent Disks**
  - Disk organization techniques that manage a large numbers of disks, providing a view of a single disk of
    - High capacity and high speed by using multiple disks in parallel
      - Improves read/write data rate by allowing independent read/write
    - High reliability by storing data redundantly on multiple disks
      - So that data can be recovered even if a disk fails

RAID

- The chance that some disk out of a set of *N* disks will fail is much higher than the chance that a specific single disk will fail
  - E.g., a system with 100 disks, each with MTTF of 100,000 hours (approx. 11 years), will have a system MTTF of 1000 hours (approx. 41 days)
  - Solved the problem of reliability by Redundancy
    - To avoid data loss are critical with large numbers of disks
  - **Redundancy** – store extra information that can be used to rebuild information lost in a disk failure
  - E.g., **Mirroring** (or **shadowing**)
    - Duplicate every disk
    - Logical disk consists of two physical disks
    - Every write is carried out on both disks
    - Reads can take place from either disk

Improvement of Reliability via Redundancy

- E.g., **Mirroring** (or **shadowing**)
  - If one disk in a pair fails, data still available in the other
    - Data loss would occur only if a disk fails, and its mirror disk also fails before the system is repaired
      - Probability of combined event is very small
        - » Except for dependent failure modes such as fire or building collapse or electrical power surges
- **Mean time to data loss** depends on mean time to failure, and **mean time to repair**
  - E.g. MTTF of 100,000 hours, mean time to repair of 10 hours gives mean time to data loss of $500*10^6$ hours (or 57,000 years) for a mirrored pair of disks (ignoring dependent failure modes)

Improvement in Performance via Parallelism

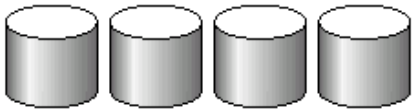
- Benefits
  - The read request rate can be doubled
    - As the request can be send to either disk
  - Improve transfer rate by striping data across multiple disks
    - **Bit-level striping** – splits bits of each byte across multiple disks
      - In an array of eight disks, write bit *i* of each byte to disk *i*
      - Each access can read data at eight times the rate of a single disk
      - Not used much any more
    - **Block-level striping** – with *n* disks, block *i* of a file goes to disk (*i* mod *n*) + 1
      - e.g. 8 disks, logical block 0 is stored in physical block 0 of disk1, logical block 11 is stored in physical block 1 of disk 4
      - Requests for different blocks can run in parallel if the blocks reside on different disks
      - A request for a long sequence of blocks can utilize all disks in parallel

Improvement in Performance via Parallelism

- Two main goals of parallelism in a disk system:
  1. Load balance multiple small accesses to increase throughput
  2. Parallelize large accesses to reduce response time
- Mirroring provides high reliability, but it is expensive
- Striping provides high data rates, but does not improve reliability
- Solution is **PARITY**
  - Memory system uses parity bits for error detection and correction
  - Each byte in a memory system have a parity bit
    - To record the no. of bits in the byte that are set to 1 is even parity=0) or odd (parity=1)
    - If one of the byte gets damaged, the parity of the byte changes and thus will not match the computed parity
    - Detects only 1-bit errors

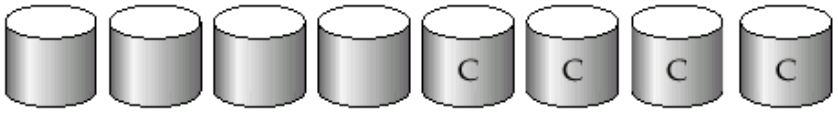
RAID Levels

- Schemes to provide redundancy at lower cost by using disk striping combined with parity bits
  - Different RAID organizations, or RAID levels, have differing cost, performance and reliability characteristics
- **RAID Level 0**: Block striping; non-redundant
  - Used in high-performance applications where data loss is not critical



(a) RAID 0: nonredundant striping

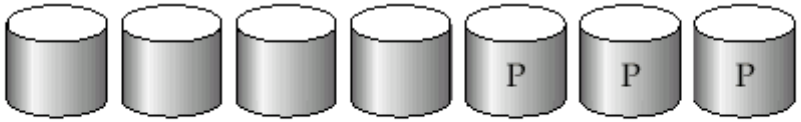
- **RAID Level 1**: Mirrored disks with block striping
  - Offers best write performance
  - Popular for applications such as storing log files in a database system
  - For Four disk requires 4 mirror disks (C indicates a 2nd copy of data)



(b) RAID 1: mirrored disks

RAID Levels (Cont.)

- **RAID Level 2**: Memory-Style Error-Correcting-Codes (ECC) with bit striping

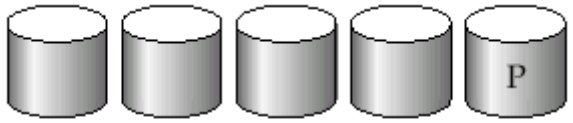


(c) RAID 2: memory-style error-correcting codes

- Parity bits
  - Memory system uses parity bits for error detection and correction (Hamming Code)
- P disk stores error-correcting bits
  - If one of the disks fails, the remaining bits of the byte and the associated error-correction bits can be read from other disks and can be used to reconstruct the damaged data
- For 4 disks, RAID Level 2 requires more 3 disks' overhead for four disks of data

RAID Levels (Cont.)

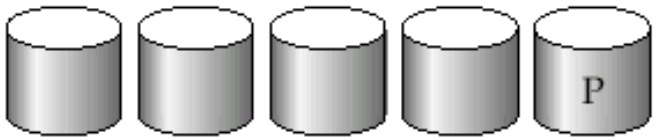
- **RAID Level 3**: Bit-Interleaved Parity
  - Uses byte level stripping along with parity
  - A single parity bit is enough for error correction, not just detection, since we know which disk has failed
    - When writing data, corresponding parity bits must also be computed and written to a parity bit disk
    - To recover data in a damaged disk, compute XOR of bits from other disks (including parity bit disk)
  - Advantages
    - Needs only one parity disk for several regular disks
      - Reduces storage overhead
    - Reads and writes of a byte are spread out over multiple disks, with N-way striping of data, the transfer rate for reading or writing a single block is N times faster than a RAID level 1
    - Supports a lower of I/O operations per second
      - As every disk has to participate in every I/O request
    - Subsumes Level 2 (provides all its benefits, at lower cost)



(d) RAID 3: bit-interleaved parity

RAID Levels (Cont.)

- **RAID Level 4:** Block-Interleaved Parity; uses block-level striping, and keeps a parity block on a separate disk for corresponding blocks from *N* other disks
  - When writing data block, corresponding block of parity bits must also be computed and written to parity disk
  - To find value of a damaged block, compute XOR of bits from corresponding blocks (including parity block) from other disks



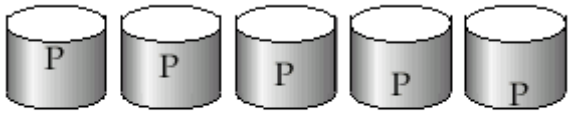
(e) RAID 4: block-interleaved parity

RAID Levels (Cont.)

- **RAID Level 4**
  - Provides higher I/O rates for independent block reads than Level 3
    - Block read goes to a single disk, so blocks stored on different disks can be read in parallel
  - Provides high transfer rates for reads of multiple blocks than no-striping
  - Before writing a block, parity data must be computed
    - Can be done by using old parity block, old value of current block and new value of current block (2 block reads + 2 block writes)
    - Or by recomputing the parity value using the new values of blocks corresponding to the parity block
      - More efficient for writing large amounts of data sequentially
  - Parity block becomes a bottleneck for independent block writes since every block write also writes to parity disk

RAID Levels (Cont.)

- **RAID Level 5:** Block-Interleaved Distributed Parity; partitions data and parity among all *N* + 1 disks, rather than storing data in *N* disks and parity in 1 disk



(f) RAID 5: block-interleaved distributed parity

  - E.g., With array of 5 disks, 20 blocks on 5 disks➔ 0 to 4 sets the parity block Pk for logical blocks 4k, 4k+1, 4k+2, 4k+3 is stored in disk k mod 5, i.e.
    - Parity bit for Block set 0 of Blocks 0, 1, 2, 3 is stored in 0th Disk
    - Parity bit for Block set 1 of Blocks 4, 5, 6, 7 is stored in 1st Disk

P0	0	1	2	3
4	P1	5	6	7
8	9	P2	10	11
12	13	14	P3	15
16	17	18	19	P4

RAID Levels (Cont.)

- **RAID Level 5**
  - Higher I/O rates than Level 4
    - Block writes occur in parallel if the blocks and their parity blocks are on different disks
  - Subsumes Level 4: Provides same benefits, but avoids bottleneck of parity disk
- **RAID Level 6**: P+Q Redundancy scheme; similar to Level 5, but stores double redundant information to guard against multiple disk failures
  - Better reliability than Level 5 at a higher cost; not used as widely

(g) RAID 6: P + Q redundancy

Choice of RAID Level

- Factors in choosing RAID level
  - Monetary cost
  - Performance: Number of I/O operations per second, and bandwidth during normal operation
  - Performance during failure
  - Performance during rebuild of failed disk
    - Including time taken to rebuild failed disk
- RAID 0 is used only when data safety is not important
  - E.g. data can be recovered quickly from other sources
- Level 2 and 4 never used since they are subsumed by 3 and 5
- Level 3 is not used anymore since bit-striping forces single block reads to access all disks, wasting disk arm movement, which block striping (level 5) avoids
- Level 6 is rarely used since levels 1 and 5 offer adequate safety for most applications

Choice of RAID Level (Cont.)

- Level 1 provides much better write performance than level 5
  - Level 5 requires at least 2 block reads (Data + Parity) and 2 block writes (Data + Parity) to write a single block, whereas Level 1 only requires 2 block writes (Data + Copy)
  - Level 1 preferred for high update environments such as log disks
- Level 1 had higher storage cost than level 5
  - Disk drive capacities increasing rapidly (50%/year) whereas disk access times have decreased much less (x 3 in 10 years)
  - I/O requirements have increased greatly, e.g. for Web servers
  - When enough disks have been bought to satisfy required rate of I/O, they often have spare storage capacity
    - So there is often no extra monetary cost for Level 1!
- Level 5 is preferred for applications with low update rate, and large amounts of data
- Level 1 is preferred for all other applications

RAID Issues

- **Software RAID**: RAID implementations done entirely in software, with no special hardware support
- **Hardware RAID**: RAID implementations with special hardware
  - Uses non-volatile RAM to record writes that are being executed
  - Beware: power failure during write can result in corrupted disk
    - E.g. failure after writing one block but before writing the second in a mirrored system
    - Such corrupted data must be detected when power is restored
      - Recovery from corruption is similar to recovery from failed disk
      - NV-RAM helps to efficiently detected potentially corrupted blocks
        - Otherwise all blocks of disk must be read and compared with mirror/parity block

Hardware RAID Issues (Cont.)

- **Latent failures**: Data successfully written earlier gets damaged
  - Can result in data loss even if only one disk fails
- **Data scrubbing:**
  - Continually scan for latent failures, and recover from copy/parity
- **Hot swapping**: Replacement of disk while system is running, without power down
  - Supported by some hardware RAID systems,
  - Reduces time to recovery, and improves availability greatly
- Many systems maintain spare disks which are kept online, and used as replacements for failed disks immediately on detection of failure
  - Reduces time to recovery greatly
- Many hardware RAID systems ensure that a single point of failure will not stop the functioning of the system by using
  - Redundant power supplies with battery backup
  - Multiple controllers and multiple interconnections to guard against controller/interconnection failures

Tertiary Storage-Optical Disks

- Compact disk-read only memory (CD-ROM)
  - Removable disks, 640 MB per disk
  - Seek time about 100 msec (optical read head is heavier and slower)
  - Higher latency (3000 RPM) and lower data-transfer rates (3-6 MB/s) compared to magnetic disks
- Digital Video Disk (DVD)
  - DVD-5 holds 4.7 GB , and DVD-9 holds 8.5 GB
  - DVD-10 and DVD-18 are double sided formats with capacities of 9.4 GB and 17 GB
  - Blu-ray DVD: 27 GB (54 GB for double sided disk)
  - Slow seek time, for same reasons as CD-ROM
- Record once versions (CD-R and DVD-R) are popular
  - data can only be written once, and cannot be erased.
  - high capacity and long lifetime; used for archival storage
  - Multi-write versions (CD-RW, DVD-RW, DVD+RW and DVD-RAM) also available

Tertiary Storage-Magnetic Tapes

- Hold large volumes of data and provide high transfer rates
  - Few GB for DAT (Digital Audio Tape) format, 10-40 GB with DLT (Digital Linear Tape) format, 100 GB+ with Ultrium format, and 330 GB with Ampex helical scan format
  - Transfer rates from few to 10s of MB/s
- Tapes are cheap, but cost of drives is very high
- Very slow access time in comparison to magnetic and optical disks
  - limited to sequential access.
  - Some formats (Accelis) provide faster seek (10s of seconds) at cost of lower capacity
- Used mainly for backup, for storage of infrequently used information, and as an off-line medium for transferring information from one system to another.
- Tape jukeboxes used for very large capacity storage
  - Multiple petabyes (10^{15} bytes)

File Organization, Record Organization and Storage Access

DBMS manages the data at physical level

How to store entities and their relations

Separate Entity & Relations or Together Entity and Relations

File Organization

- Files
  - The database is stored as a collection of different *files*
    - Maintained by OS
    - Permanently stored on disks
  - Each file is a logical sequence of *records*
    - A record is a sequence of field's values
    - Records are mapped onto disk blocks
  - Need to understand how logical data model is stored as files
  - Must support
    - Insertion/deletion/modification of record
    - Read a particular record (given with record id)
    - Scan all records (possibly with some conditions on the records)

File Organization

- File: Logically partitioned into Fixed Length Block
- Block
  - Compatible to the storage allocation an data transfer
  - 4 to 8 KB or DBMS dependent
  - May contain several records
    - Set of record determined by the physical data organization form

File Organization

- Block
  - Assumption 1
    - No record is larger than a block
    - Images which are larger than block size (Discuss later on)
      - » Stores the large data items separately
      - » And also stores a pointer to the data item in the record
  - Assumption 2 (To speed up access)
    - Each record is entirely contained in a single block
      - »One record per block, No record is contained partly in one block and partly in another

File Organization

- RDBMS
  - Tuples of distinct relations are of different size
    - Approach to map database to files
      - Use **several files and store records of only one fixed length** in any given file
      - Use **file with multiple length for records**
  - Different types of file according to the record size
    1. Fixed Length Record
    2. Variable Length Record

File Organization

- File contains different record size
  1. Fixed Length Record
     - **Each file has records of one particular type only**
     - **Different files are used for different relations**
     - Easiest to implement
       - Store information in catalog
  2. Variable Length Record
     - **Can have different types of records**
     - Difficult to implement compare to fixed length
       - Encodes the length of each field, utilizes record length or uses a field delimiter

Fixed-Length Records

- Simple approach
  - File of Instructor records stores
    - (ID varchar(5), name varchar(20), dept_name varchar(20), salary numeric (8,2))
    - Character occupies 1 byte and numeric(8,2) 8 bytes
    - Total 1 record size is 53 bytes

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

Fixed-Length Records

- Simple approach
  - Two problems
    1. Block size happens to be a multiple of record size (example 53)
       - It may cross the block boundaries
       - Solution
         - Allocate as many records to a block as would fit entirely in the block
         - **Divide the block size by the record size, discard fraction part**
         - Store record based on arithmetic calculation
         - Store record *i* starting from byte $n * (i - 1)$, where n is the size of each record.
         - Record access is simple
         - Do not allow records to cross block boundaries

Fixed-Length Records

2. Difficult to delete a record
   - Solution
     - The space occupied by the record to be deleted must be filled with some other record of the file
       1. **Move records *i* + 1, . . ., *n* to *i*, . . . , *n* – 1**
       2. **Move record *n* to i**
     - Have a way of marking deleted records to ignore
       3. **Do not move records, but link all free records on a *free list***

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

Deleting record 3 and compacting

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

- Difficult as requires number of records to move, desirable to move the final record at the deleted record space

Deleting record 3 and moving last record

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 11	98345	Kim	Elec. Eng.	80000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000

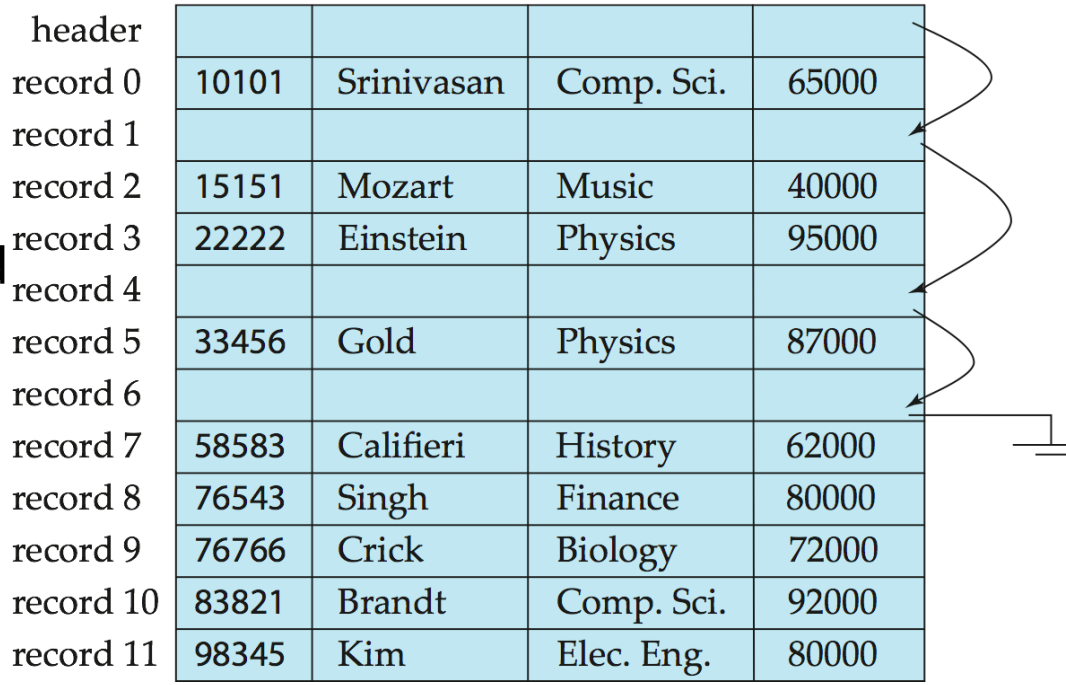
- Also requires number of block access
- Insertions tend to be more frequent than deletions, leave space open by the deleted record and to wait for a subsequent insertion

Free Lists

- **File Header**, Additional structure is required to do so,
- Contains variety of information about the file
- Here, Store the address of the first deleted record in the file header
- Use this first record to store the address of the second deleted record, and so on
- Consider, stored addresses

as pointers since they

"point" to the location of a record



header				
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1				
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4				
record 5	33456	Gold	Physics	87000
record 6				
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

Free Lists

- On insertion of a new record
  - Use the record pointed by the header
    - Change the header pointer to point to the next available record
  - If no space is available, add the new record to the end of file
- More space efficient representation and simple
  - As inserted and deleted record size is same

Variable-Length Records

- Need of Variable-length records in database systems in several ways
  1. **Storage of multiple record types in a file**
  2. Record types that allow variable lengths for one or more fields such as strings (**varchar**)
  3. Record types that allow **repeating fields** (used in some older data models)
- Two problems to solve to use this technique
  1. The way of representations of single record to extract individual attributes of the record
  2. Storage of record within a block, such that records in a block can be extracted easily

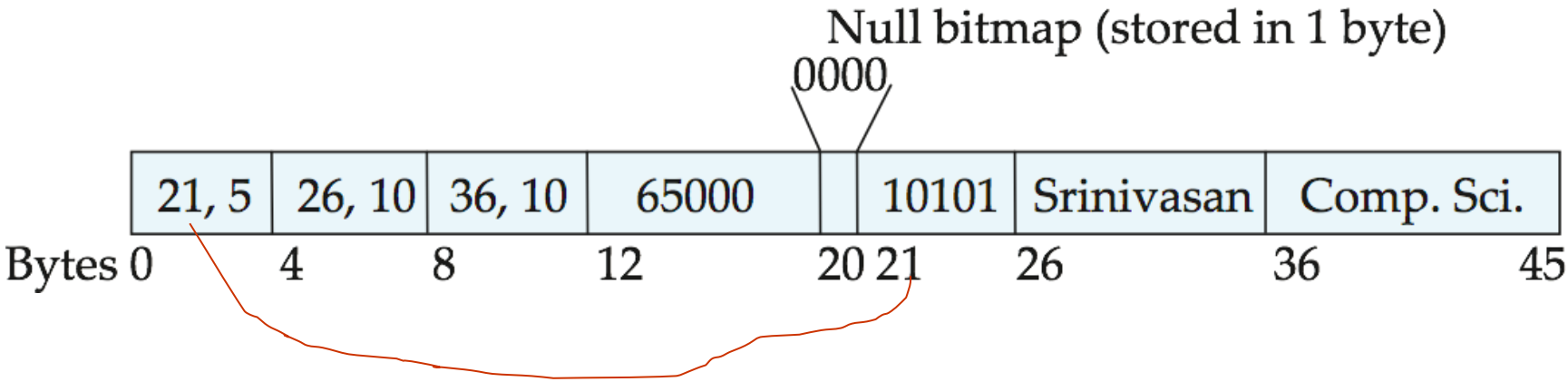
Variable-Length Records

- Variable length record has two parts
  - **Initial part with fixed length attributes**
    - E.g. Numeric values, dates or fixed length character string
  - **Followed by data for variable length attributes**
    - E.g. Varchar types
- Solution
  - Attributes are stored in order
    - Variable length attributes represented by fixed size (offset, length), with actual data stored after all fixed length attributes
  - Utilizes Null Bitmap

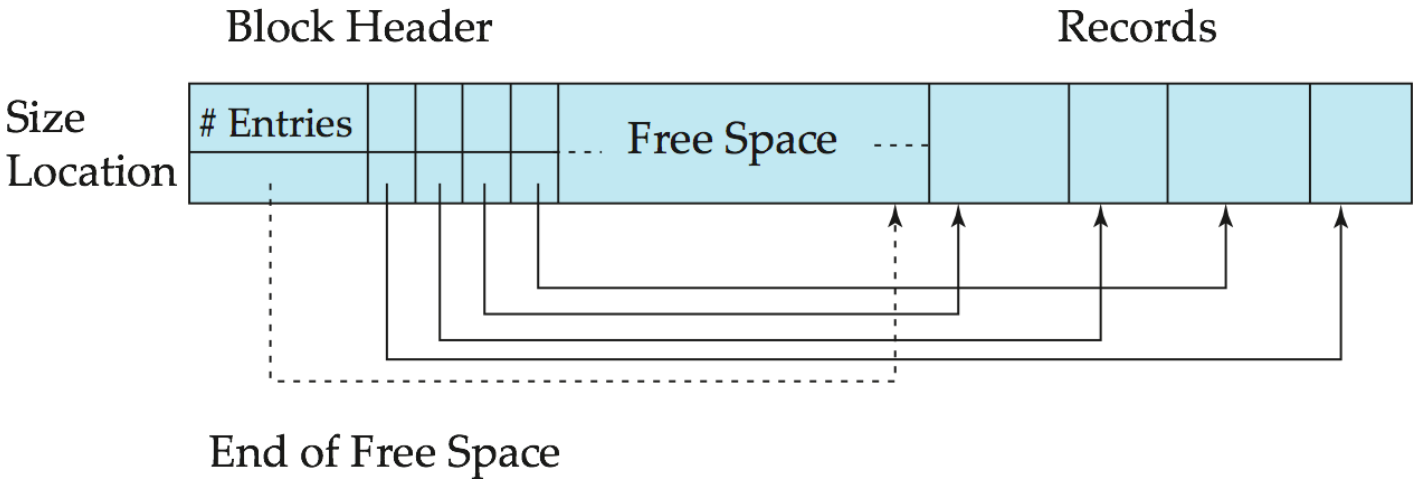
Variable-Length Records

- Utilizes Null Bitmap
  - Indicates which attributes of the record has a Null value
  - Use 1 to indicate null value
    - Ignore the bytes stored at the offset
  - Sometimes stored at the beginning of the record
    - Not storing data (value, offset/length) at all
    - Save storage space, at the cost of extra work to extract

(ID varchar(5), name varchar(20), dept_name varchar(20), salary numeric (8,2))

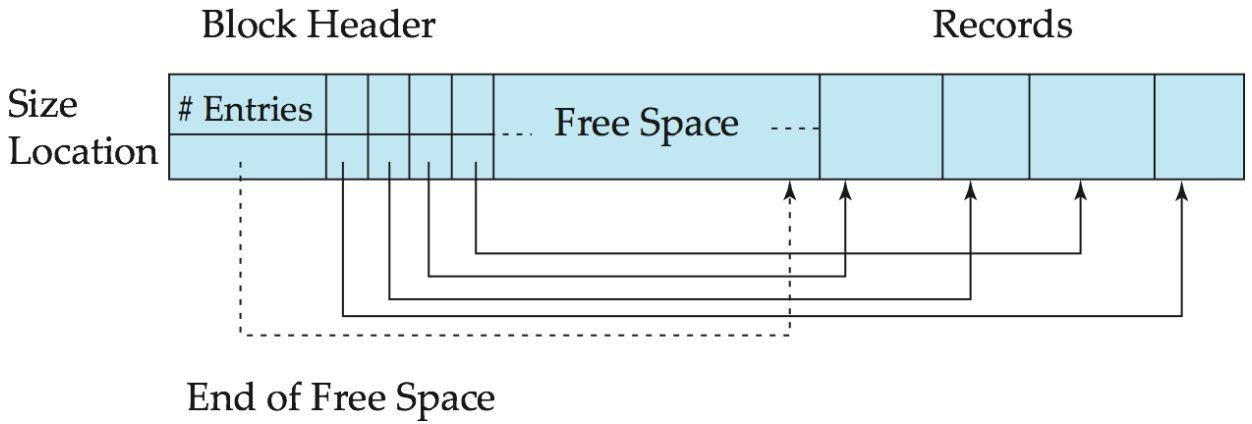


Variable-Length Records: Storage



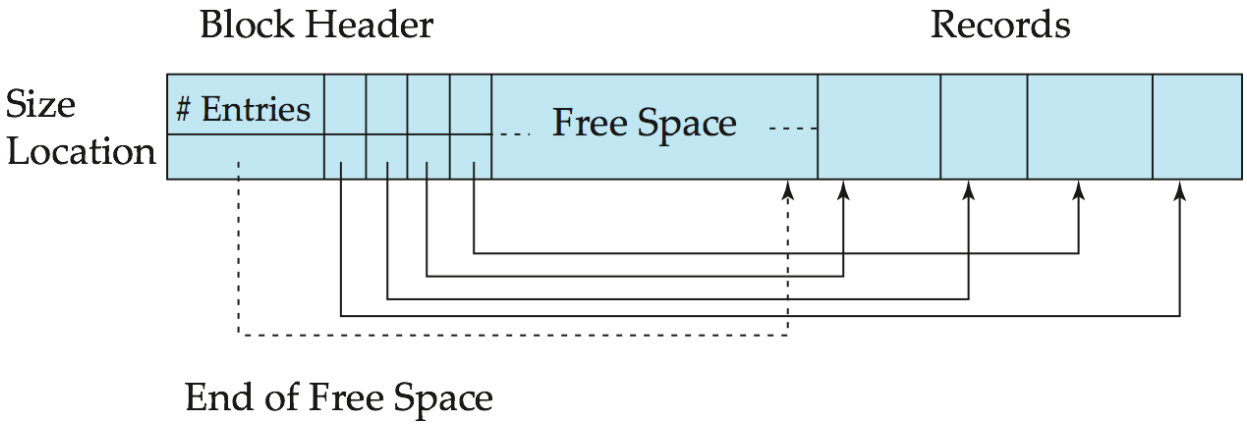
- Utilizes slotted page structure
- **Slotted page** header contains
  1. Number of record entries
  2. End of free space in the block
  3. Location and size of each record
- The actual records are allocated contiguously in the block, starting from the end of the block
- The free space in the block is contiguous, between the final entry in the header array and the first record

Variable-Length Records: Slotted Page Structure



- Record insertion
  - Space is allocated for it at the end of the free space
  - An entry containing its size and location is added to the header
- Record deletion
  - The occupied space is freed and its entry is set to deleted (e.g. set size -1)
  - Records can be moved around within a page to keep them contiguous with no empty space between them; entry in the header must be updated

Variable-Length Records: Slotted Page Structure



- Pointers should not point directly to record — instead they should point to the entry for the record in header
  - Prevents fragmentation of space inside the block

Organization of Records in Files

- Discussed how records are represented in a file structure
- But, relation is a set of records
  - How to ORGANIZE RECORDS IN A FILE
    - Records of each relation stored in a separate file
      1. **Heap File Organization**
      2. **Sequential File Organization**
      3. **Hashing File Organization**
    - Records of several relations stored in one file
      4. **Multitable clustering file organization**

Organization of Records in Files

1. **Heap File Organization**
   - Any record can be placed anywhere in the file where there is space
   - No ordering of records is required
   - Single file for each relation
2. **Sequential File Organization**
   - Store records in sequential order, based on the value of the search key of each record
3. **Hashing File Organization**
4. **Multitable Clustering File Organization**

Organization of Records in Files

1. **Heap File Organization**
2. **Sequential File Organization**
3. **Hashing File Organization**
   - A hash function computed on some attribute of each record; the result specifies in which block of the file the record should be placed
4. **Multitable Clustering File Organization**
   - Records of several different relations can be stored in the same file
   - Motivation
     - Store related records on the same block to minimize I/O
     - To reduce I/O operations

Sequential File Organization

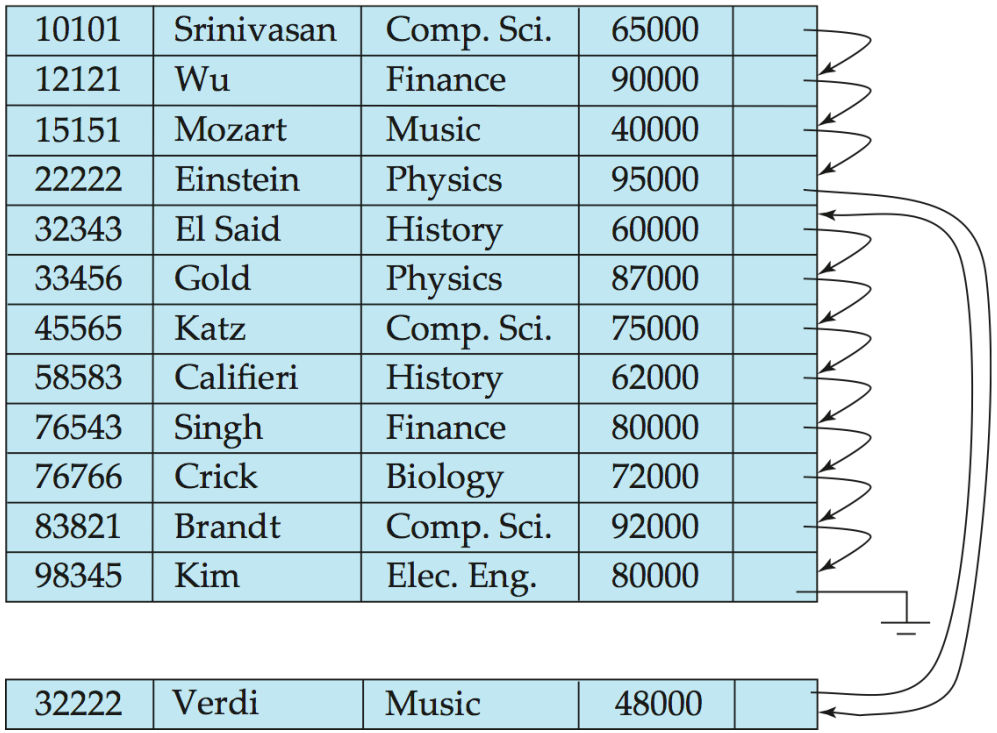
- Suitable for applications that require sequential processing of the entire file
  - Mainly display purposes as well as for certain query processing algorithm
- Search-key
  - Sorted the records in the file, ordered by this key which is Any attribute or set of attributes and not the primary key, or even a super-key
- Chain together records by pointer for fast retrieval of records

Sequential file with ID as search key

10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	

Sequential File Organization (Cont.)

- Deletion – Use pointer chains
- Insertion –Locate the position where the record is to be inserted
  - If there is free space insert there
  - If no free space, insert the record in an overflow block
  - In either case, pointer chain must be updated
- Works well
  - If relatively few records stored in overflow block
- Need to reorganize the file from time to time to restore sequential order



Multitable Clustering File Organization

- Beneficiary, If database size is large
- Instead of be dependent upon OS, DBMS manages this file
- **Store several relations in one file**
  - But, records of one relation stores in a given block
  - Simplifies data management
  - Sometimes, beneficiary if storing records of multiple relations in one block

Multitable Clustering File Organization

- Join on Department and Instructor table

department

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Physics	Watson	70000

instructor

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000

- System must locate the instructor tuples with the same value for dept_name
  - Utilization of Indices to locate the record (See later)
  - Records have to be transferred from disk into main memory
    - Worst Case: each record reside on a different block, forcing one block read for each record

Multitable Clustering File Organization

- Join on Department and Instructor table
  - Mixes together tuples of two relations for efficient processing
  - When department tuple is read, the entire block containing that tuple is copied from disk to main memory

department

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Physics	Watson	70000

instructor

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000

Multitable clustering of *department* and *Instructor*
-No dept_name for instructors
-Followed by department tuple

Comp. Sci.	Taylor	100000
45564	Katz	75000
10101	Srinivasan	65000
83821	Brandt	92000
Physics	Watson	70000
33456	Gold	87000

Multitable Clustering File Organization

- Join on Department and Instructor table
  - **Efficient query processing**
  - Utilizes nearby blocks if records do not fit in one block

department

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Physics	Watson	70000

instructor

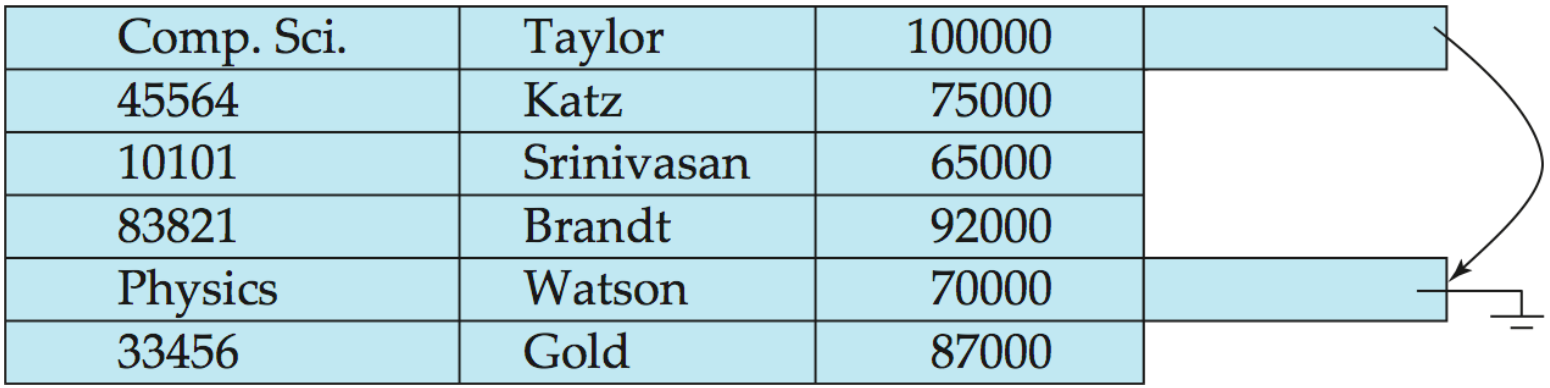
ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000

Multitable clustering of *department* and *instructor*

Comp. Sci.	Taylor	100000
45564	Katz	75000
10101	Srinivasan	65000
83821	Brandt	92000
Physics	Watson	70000
33456	Gold	87000

Multitable Clustering File Organization (cont.)

- Requires careful use
  - Good for queries involving *department* ⋈ *instructor*, and for queries involving one single department and its instructors
  - Bad for queries involving only *department, slows down execution*
    - Results in more block access
    - Can add pointer chains to link records of a particular



Comp. Sci.	Taylor	100000	
45564	Katz	75000	
10101	Srinivasan	65000	
83821	Brandt	92000	
Physics	Watson	70000	
33456	Gold	87000	

Data Dictionary Storage

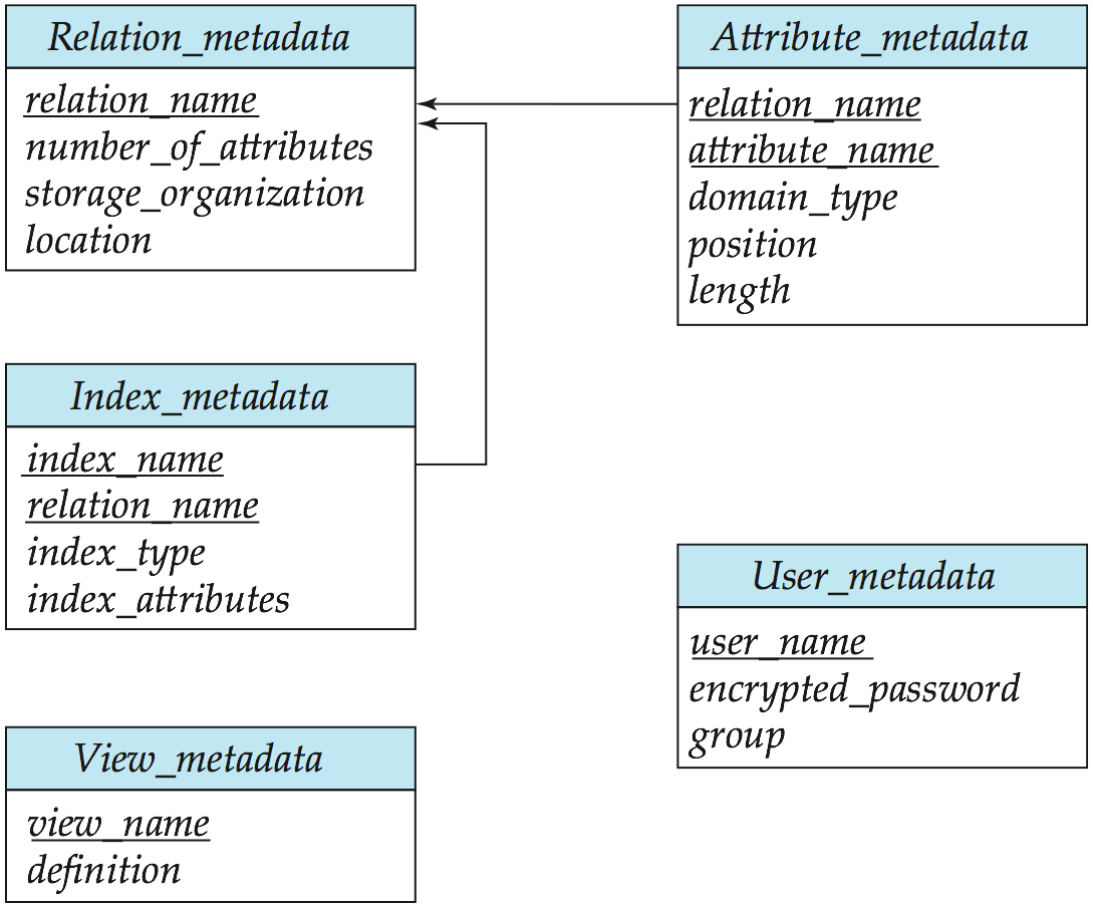
- Discussed only representation of the relations
- Requires to maintain data about the relations-data about data (METADATA)
- **Data dictionary** (**System Catalog**)
  - Stores **metadata**; that is, data about data, such as
    1. User and accounting information, including passwords
    2. Information about relations
       - Names of relations
       - Names, types and lengths of attributes of each relation
       - Names and definitions of views
       - Integrity constraints
    3. Statistical and descriptive data
       - Number of tuples in each relation
    4. Physical file organization information
       - How relation is stored (sequential/hash/…)
       - Physical location of relation
    5. Information about indices (Chapter 11)

Data Dictionary Storage

- **Metadata**
  - Constitutes miniature database
  - Stored using special purpose data structure and code
    - As relations in the database itself
    - Made by the system designers
  - **Whenever the database system needs to retrieve records from a relation**
    - First consult the *Relation_metadata* relation to find the location and storage organization of the relation
    - Then fetch records using this information

Relational Representation of System Metadata

- Relational representation on disk
- Specialized data structures designed for efficient access, in memory



- Approach

  Stored in a nonnormalized form to achieve fast access

  Attribute index_attributes of the relation Index_metadata assumed to contain a list of one or more attributes

Storage Access

- **Blocks**
  - A database file is partitioned into fixed-length storage units
  - Units of both storage allocation and data transfer
- **Database system seeks to minimize the number of block transfers between the disk and memory**
- Can be reduced the number of disk accesses by keeping as many blocks as possible in main memory
  - **Buffer** – Portion of main memory available to store copies of disk blocks
  - **Buffer manager** – Subsystem responsible for allocating buffer space in main memory

Buffer Manager

- Programs call on the buffer manager when they need a block from disk
  1. If the **block is already in the buffer**, buffer manager **returns the address of the block in main memory**
  2. If the **block is not in the buffer**, the buffer manager
     1. Allocates space in the buffer for the block
        1. **Replacing** (throwing out) some other block, if required, **to make space for the new block**
        2. Replaced block **written back to disk only if it was modified** since the most recent time that it was written to/fetched from the disk
     2. **Reads the block from the disk to the buffer**, and returns the address of the block in main memory to requester

Buffer Manager

- Just like Virtual memory manager found in OS
  - Difference is that the size of the database might be larger than the hardware address space of a machine, so memory addresses are not sufficient to address all disk blocks
  - Utilizes techniques more sophisticated than typical VM management schemes
    1. Buffer Replacement Policies
       1. Least Recently Used (LRU)
       2. Toss-Immediate
       3. Most Recently Used
    2. Pinned blocks
    3. Forced output of blocks

Buffer-Replacement Policies

- To minimize access to the disk
- In general-purpose program, not possible to predict accurately which blocks will be referenced
- Queries have well-defined access patterns (such as sequential scans), and a database system can use the information in a user's query to predict future references

Buffer-Replacement Policies

1.1 LRU strategy

- Idea behind LRU – OS uses past pattern of block references as a predictor of future references
- **The blocks that have been referenced recently are likely to be referenced again**
- Most OSs replace the block **least recently used** to disk
- When there is no room left in buffer, a block must be removed from the buffer before a new one can be read in

Buffer-Replacement Policies (Cont.)

2. **Pinned block**
   - Memory block that is not allowed to be written back to disk
     - As block should not be written to disk while an update on the block is in progress
   - Many **OS does not support pinned blocks** as durable to crash
3. **Forced output of blocks**
   - Write situations, in which it is necessary to write back the block to disk, even though the buffer space that it occupies is not needed, it is for the purpose of recovery (See later in Ch. 16 Recovery Algorithm)

Buffer-Replacement Policies

1.2 Toss-immediate Strategy

- Frees the space occupied by a block as soon as the final tuple of that block has been processed
- Example Query: Select * from instructor natural join department
- Strategy:
  - for each tuple *tr* of Instructor do
    for each tuple *ts* of department do
    if the tuples *tr* and *ts* match …
- If these two relations stored in separate files, once a tuple of the instructor has been processed, that tuple is not needed in main memory, even though it has been used recently

Buffer-Replacement Policies

1.3 Most Recently Used Strategy

- Example Query: Select * from instructor natural join department
- Strategy:
  - for each tuple *tr* of Instructor do
    for each tuple *ts* of department do
    if the tuples *tr* and *ts* match …
- Need to examine every block of department tuples once for each tuple of the instructor relation
- Processing of department block is completed, we know that block will not be accessed again until all other department blocks have been processed
  - Most recently used department block will be the final block to be re-referenced
  - Least recently used department block is the block that will be referenced next
- Optimal strategy for this query

Buffer-Replacement Policies

1.3 Most recently used (MRU) strategy (Cont.)

- System must pin the block currently being processed
- After the final tuple of that block has been processed, the block is unpinned, and it becomes the most recently used block
- Buffer manager can use statistical information regarding the probability that a request will reference a particular relation
  - E.g., The data dictionary is frequently accessed
  - Heuristic: keep data-dictionary blocks in main memory buffer

Buffer-Replacement Policies

- Policies Summary
  - No single strategy is known that handles all the possible scenarios well
  - **Large no. of DBMSs use LRU, even though it fails**
  - Mixed strategy influenced by factors
    - If the system is processing requests by several users concurrently, the Concurrency control subsystem (Ch. 15) need to delay certain requests to preserve database consistency
      - Alter the block replacement strategy, blocks needed by active (non-delayed) requests can be retained in the buffer at the expense of blocks needed by the delayed requests

Buffer-Replacement Policies

- Policies Summary
  - Mixed strategy influenced by factors
    - The crash-recovery subsystem imposes that,
      - If the block has been modified, the buffer manager is not allowed to write back the new version of the block in the buffer to disk, since that would destroy the old version
        - » Permission to set from crash-recovery to buffer manager before writing out a block
        - » Certain blocks be forced-output

End of Chapter 10