Chapter 10: Storage and File Structure

Database System Concepts, 6th Ed.

Database System Concepts

- Chapter 1: Introduction
- **Part 1: Relational databases**
  - Chapter 2: Introduction to the Relational Model
  - Chapter 3: Introduction to SQL
  - Chapter 4: Intermediate SQL
  - Chapter 5: Advanced SQL
  - Chapter 6: Formal Relational Query Languages
- **Part 2: Database Design**
  - Chapter 7: Database Design: The E-R Approach
  - Chapter 8: Relational Database Design
  - Chapter 9: Application Design
- **Part 3: Data storage and querying**
  - Chapter 10: Storage and File Structure
  - Chapter 11: Indexing and Hashing
  - Chapter 12: Query Processing
  - Chapter 13: Query Optimization
- **Part 4: Transaction management**
  - Chapter 14: Transactions
  - Chapter 15: Concurrency control
  - Chapter 16: Recovery System
- **Part 5: System Architecture**
  - Chapter 17: Database System Architectures
  - Chapter 18: Parallel Databases
  - Chapter 19: Distributed Databases
- **Part 6: Data Warehousing, Mining, and IR**
  - Chapter 20: Data Mining
  - Chapter 21: Information Retrieval
- **Part 7: Specialty Databases**
  - Chapter 22: Object-Based Databases
  - Chapter 23: XML
- **Part 8: Advanced Topics**
  - Chapter 24: Advanced Application Development
  - Chapter 25: Advanced Data Types
  - Chapter 26: Advanced Transaction Processing
- **Part 9: Case studies**
  - Chapter 27: PostgreSQL
  - Chapter 28: Oracle
  - Chapter 29: IBM DB2 Universal Database
  - Chapter 30: Microsoft SQL Server
- **Online Appendices**
  - Appendix A: Detailed University Schema
  - Appendix B: Advanced Relational Database Model
  - Appendix C: Other Relational Query Languages
  - Appendix D: Network Model
  - Appendix E: Hierarchical Model

Chapter 10: Storage and File Structure

- 10.1 Overview of Physical Storage Media
- 10.2 Magnetic Disk and Flash Storage
- 10.3 RAID
- 10.4 Tertiary Storage
- 10.5 File Organization
- 10.6 Organization of Records in Files
- 10.7 Data-Dictionary Storage
- 10.8 Database Buffer

Classification of Physical Storage Media

- Main Issues
  - 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 battery-backed up main-memory

Physical Storage Media [1/5]

- **Cache**
  - fastest and most costly form of storage
    - 25MHz – 5GHz (similar to CPU clock speed)
    - Main memory보다 수십배-수백배 빠른정도
  - Volatile -- SRAM (Static RAM)
  - managed by the computer system hardware
  - Cache size vs Main memory size = 1: 1000
    - eg. Intel i5 uses 3 MegaBytes L2 Cache
- **Main memory:**
  - fast access (10s to 100s of nanoseconds; 1 nanosecond = 10^{-9} second)
  - 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** -- DRAM (Dynamic RAM)

Physical Storage Media [2/5]

- **Flash memory**
  - Nonvolatile - Data survives power failure
  - Can support only a limited number (100K – 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 devices such as digital cameras, phones, and USB keys
    - USB stands for universal serial bus
  - NAND Flash vs NOR Flash
    - Formerly EEPROM (Electrically Erasable Programmable Read-Only Memory)
  - Flash memory for Hard disk ➔ Solid State Disks (SSD)

Physical Storage Media [3/5]

- **Magnetic-disk**
  - Primary medium for the long-term storage of database
  - Data is stored on spinning disk, and read/written magnetically
  - Data must be moved from disk to main memory for access, and written back for storage (100M Bytes/ Sec)
    - Much slower access than main memory (more on this later)
    - 수십 millisecs for Disk access vs 수십 nanosecs in Main memory access
  - **Direct-access** – possible to read data on disk in any order, unlike magnetic tape
  - Hard disks vs Floppy disks
  - 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)
  - Nonvolatile -- survives power failures and system crashes
    - disk failure can destroy data, but is rare

Physical Storage Media [4/5]

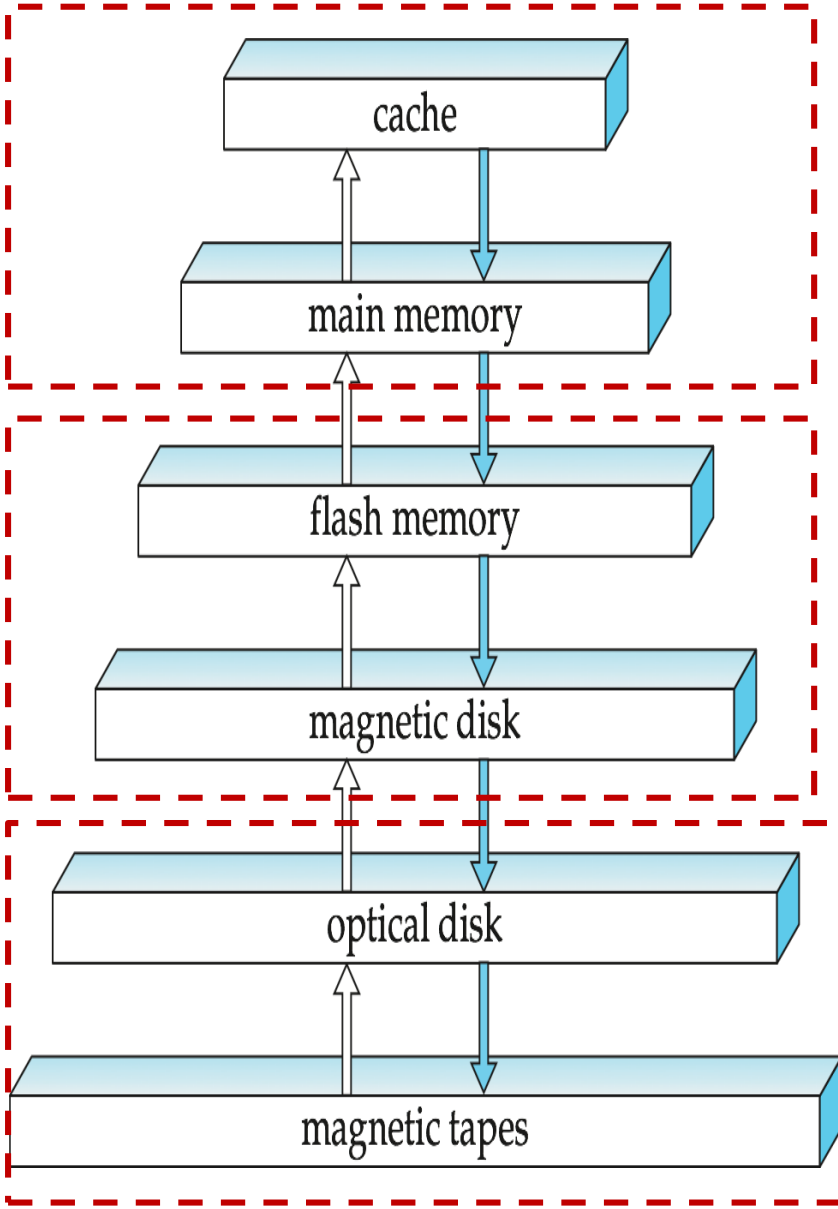
- **Optical storage**
  - Non-volatile, data is read optically from a spinning disk using a laser
  - CD-ROM (640 MB) vs DVD (4.7 to 17 GB) vs Blu-ray disks (27 GB to 54 GB)
  - Write-one, read-many (WORM) optical disks used for archival storage
    - CD-R, DVD-R, DVD+R
  - Multiple W versions available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

Physical Storage Media [5/5]

- **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

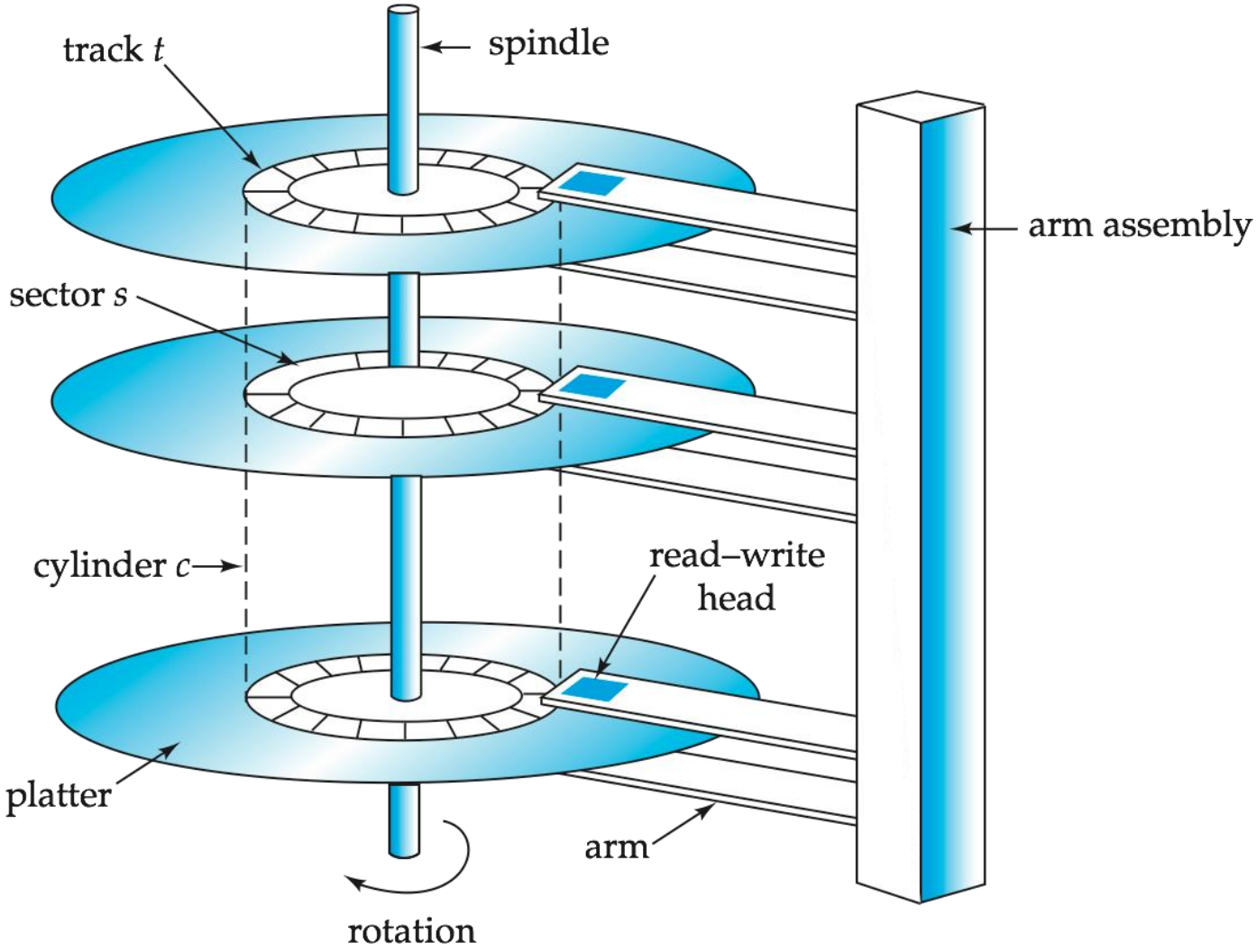
- **primary storage**: Fastest media but volatile
  - E.g. cache, main memory
- **secondary storage**: next level in hierarchy, non-volatile, moderately fast access time
  - also called **on-line storage**
  - E.g. flash memory, magnetic disks
- **tertiary storage**: lowest level in hierarchy, non-volatile, slow access time
  - also called **off-line storage**
  - E.g. magnetic tape, optical storage



Chapter 10: Storage and File Structure

- 10.1 Overview of Physical Storage Media
- 10.2 Magnetic Disk and Flash Storage
- 10.3 RAID
- 10.4 Tertiary Storage
- 10.5 File Organization
- 10.6 Organization of Records in Files
- 10.7 Data-Dictionary Storage
- 10.8 Database Buffer

Magnetic Hard Disk Mechanism



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

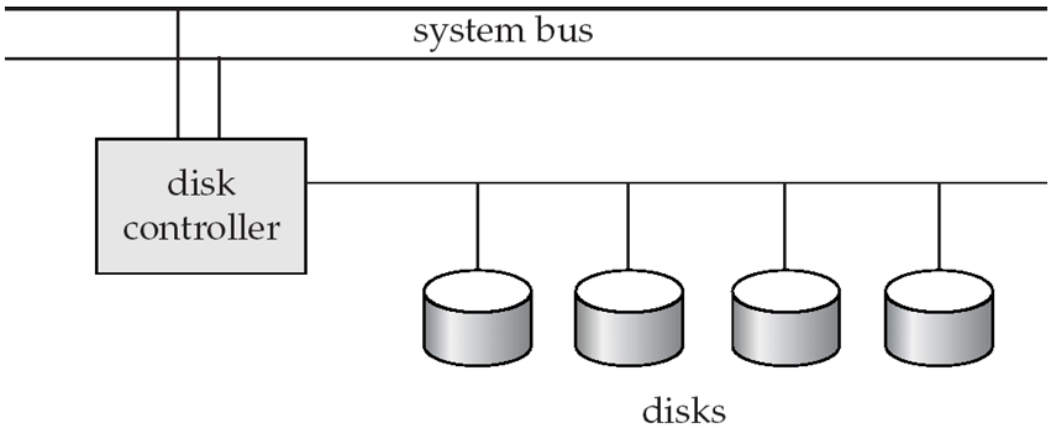
Magnetic Disks [1/2]

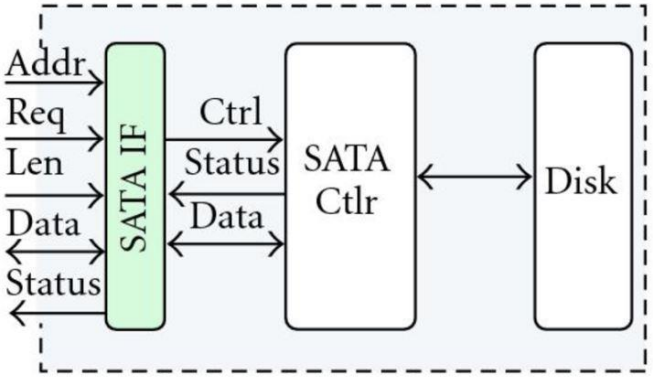
- **Read-write head**
  - Positioned very close to the platter surface (almost touching it)
  - Reads or writes magnetically encoded information.
- Surface of platter divided into circular **tracks**
  - Over 50K-100K tracks per platter on typical hard disks
- Each track is divided into **sectors**.
  - A sector is the smallest unit of data that can be read or written.
  - Sector size typically 512 bytes
  - 500 --1000 sectors on inner tracks vs 1000 -- 2000 sectors 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 [2/2]

- 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**
  - hardware 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
    - Detect bad sector and find a new location for the sector

Disk Subsystem [1/2]





- Multiple disks connected to a computer system through a controller
  - Controllers functionality (checksum, bad sector remapping) often carried out by individual disks in order to reduce load on controller
- Disk interface standards
  - Bit serial interfaces
    - ATA (AT adaptor) range of standards
    - SATA (Serial ATA) / mSATA (micro SATA)
  - Word serial interfaces
    - SCSI (Small Computer System Interconnect) range of standards
    - Parallel ATA

Disk Subsystem [2/2]

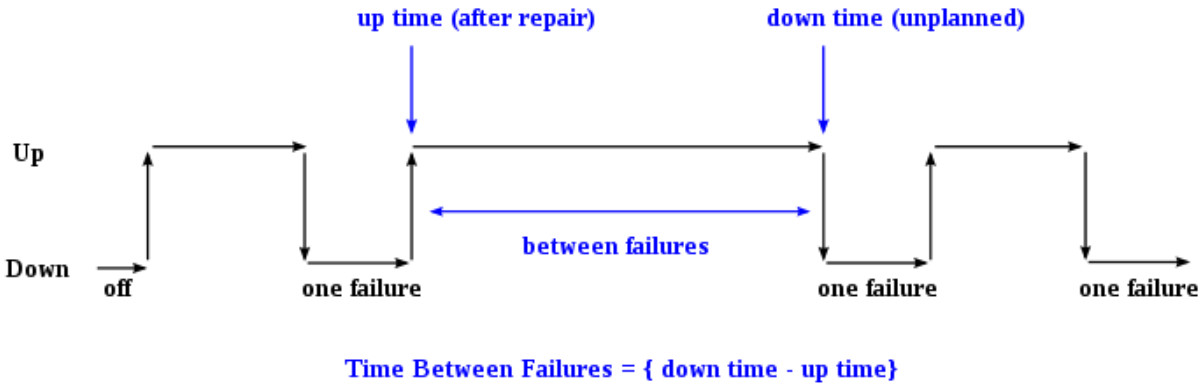
- Disks usually connected directly to disk controller by a high-speed network
- **Storage Area Networks (SAN)**
  - Many disks are connected by a high-speed network to several servers
    - provides disk system interface
      - Read some bits (words) from the given address of a disk number
      - Write some bits (words) to the given address of a disk number
    - A logical view of a very large and very reliable disk
    - Using storage organization technique called RAID (Redundant Array of Independent Disks)
- **Network Attached Storage (NAS)**
    - provides a file system interface
      - Read a word (line) from a file name
      - Write a word (line) to a file name
    - using networked file system protocol (such as NFS, CIFS)

Performance Measures of Disks [1/2]

- **Access time** – the time it takes from when a read or write request is issued to when data transfer begins (보통 5-10 milli secs)
  - **Seek time** – time for repositioning 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 milli secs on typical disks
  - **Rotational latency** – time for rotating the sector under the head
    - Average latency is 1/2 of the worst case latency.
    - 4 to 11 milli secs on typical disks (5400 to 15000 r.p.m.)
- **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
  - Data-transfer rate of disk controller interface
    - SATA: 150 MB/sec, SATA-II 3Gb: 300 MB/sec
    - Ultra 320 SCSI: 320 MB/s, SAS (serial attached SCSI): 3 to 6 Gb/sec
    - Fiber Channel (FC2Gb or 4Gb) for SAN devices: 256 to 512 MB/s

Performance Measures of Disk [2/2]

- **Mean time to failure (MTTF)**
  - The average time the disk is expected to run continuously without any failure
  - Probability of failure of new disks is quite low, due to a “theoretical MTTF” of 500,000 to 1,200,000 hours (57 to 136 years) for a new disk
    - 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
    - Vendor의 표준은 1000개의 new disks들의 Maximum MTTF를 합친것
  - MTTF decreases as disk ages
    - Average Life Span of Disk: Typically 3 to 5 years (1 year = 8760 hrs)



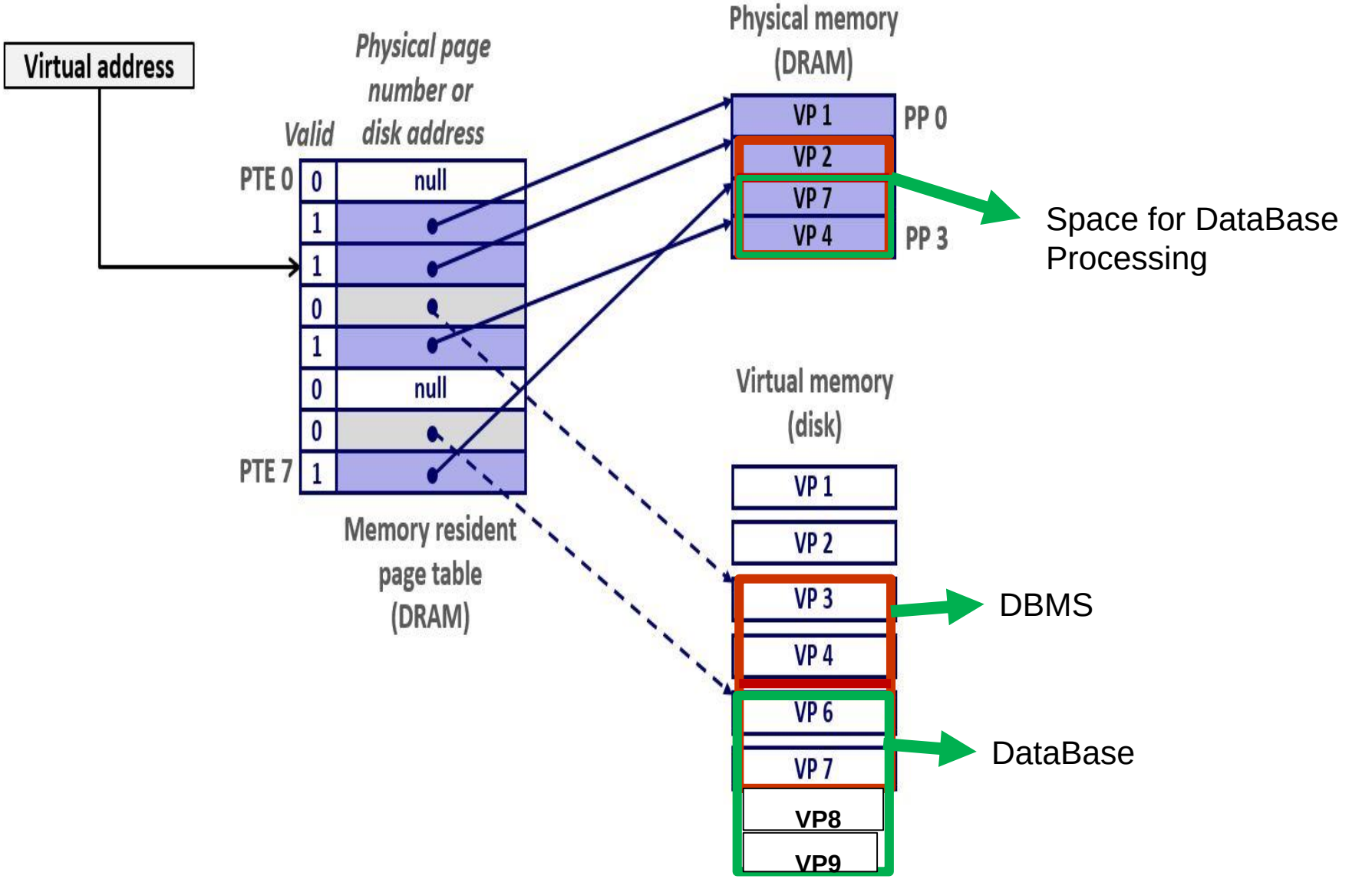
Optimization of Disk-Block Access [1/3]

- **Block** – a contiguous sequence of sectors from a single track
  - Data-transfer unit between disk and main memory
  - sizes range from 512 bytes to several Kbytes
    - 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 Kbytes
      - ("Korth", "Le High", "Professor", "New Jersey", 434-523-6678, 1957-02-02)
        이런 style의 100 character가 있는 one record의 길이는 400 bytes.
        Disk block size가 16Kbyte이라면 1 disk block에 40개 records

- Buffering Technique (utilizing some main-memory area)
  - Buffer Replacement algorithm in OS level
  - Buffer Replacement algorithm in DBMS level

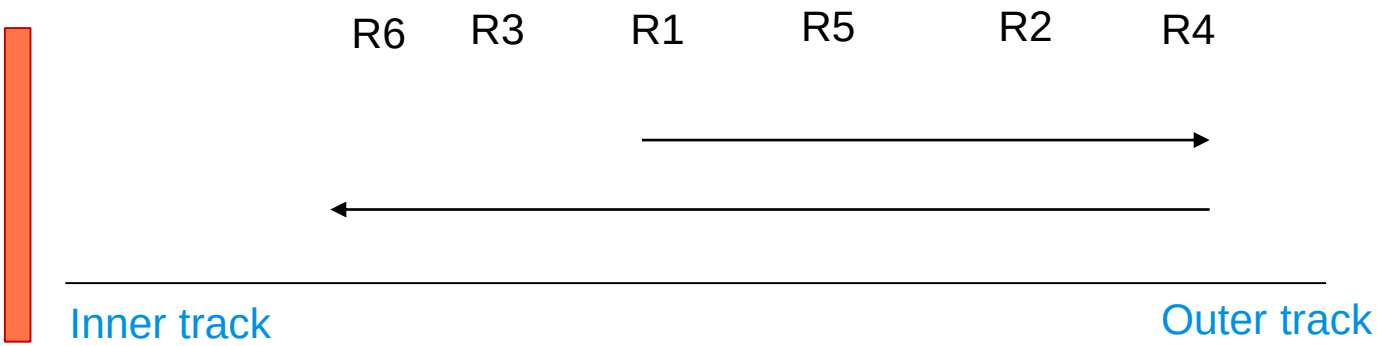
- Read-Ahead Technique
  - Moving consecutive blocks into an in-memory buffer

O/S Page Buffering



Optimization of Disk Block Access [2/3]

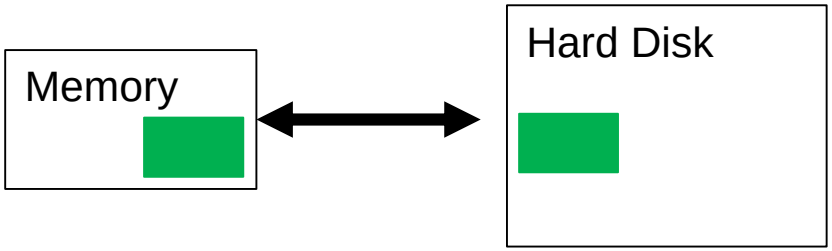
- **Disk-arm-scheduling algorithms** order pending accesses to tracks so that disk arm movement is minimized
  - **elevator algorithm**: move disk arm in one direction (from outer to inner tracks or vice versa), processing next request in that direction, till no more requests in that direction, then reverse direction and repeat



- **File fragmentation control** to optimize block access time
  - Need to organize the blocks to correspond to how data will be accessed
    - E.g. Store related information on the same or nearby cylinders.
  - Sequential access to a fragmented file results in increased disk arm movement
  - Files may get fragmented over time if data is often inserted and deleted
  - New file may have scattered blocks over the disk if free blocks are scattered
  - Some systems have utilities to defragment the file system, in order to speed up file access

Optimization of Disk Block Access [3/3] (Update들을 일단 안전한곳에 모아놓고)

- **Non-volatile write buffers:** battery backed up RAM or flash memory
  - Speed up disk writes by writing blocks to a non-volatile RAM buffer immediately
  - Even if power fails, the data is safe and will be written to disk when power returns
  - Controller then writes to disk whenever the disk has no other requests or request has been pending for some time
  - Writes can be reordered to minimize disk arm movement
  - Supported in High-end disks or RAID
- **Log disk**: a disk (미리정해진 disk area) devoted to writing a sequential log of block updates
  - First all updates are written to a log disk and later to the actual disk
  - Write to log disk is very fast since no seeks are required and
  - Writes to the actual disk can be optimized by reordering writes in log disks
  - No need for special hardware such as Nonvolatile-RAM
  - **Journaling file systems:** file system supporting log disk
    - Supported in most modern file systems



Flash Storage [1/2]

- NOR flash vs NAND flash
  - NAND flash used widely for storage, since it is much cheaper than NOR flash
    - NOR flash has read time comparable to main memory, but very expensive

- NAND flash
  - requires page-at-a-time read (page: 512 bytes to 4 KB)
    - Random access speed: 1-2 micro sec in NAND flash, 5 -10 milli sec in Disk
  - Flash memory has transfer rate around 20 MB/sec lower than hard disk (~100 MB/sec)
  - **solid state disks**: SSD use multiple flash storage devices in parallel (100 to 200 MB/sec)
  - 결론적으로 Flash memory의 access time (data read)은 hard disk에 비해서 대충 (roughly) USB key인경우는 200배, SSD인 경우는 1000배가 빠르다!
  - Flash memory의 read는 빠르다!
  - But, Flash memory의 write는 매우 느리다
    - 한번 값을 write한후에 rewrite는 erase를 먼저하고 다시 write을 한다

Flash Storage [2/2]

- Flash Memory Write
  - "erase block" contains multiple pages, so erase is very slow
    - erase is very slow (1 to 2 milli secs: read보다 roughly 1000배 느리다)
  - After 100,000 to 1,000,000 erases, erase block becomes unreliable and cannot be used
  - Solving slow erase and limited erases
    - **flash-memory controller (HW)**
    - **flash translation layer (SW)**
      - Maintaining translation table (logical page numbers to physical page numbers)
      - **wear leveling: evenly distributing erase operations across physical blocks**
        - 이미 여러 번 rewriting 이루어진 page에는 향후에 rarely updated 될것 같은 cold data를 배치, 비교적 rewriting이 적은 page에는 frequently updated 될것 같은 hot data를 배치하여 flash memory가 골고루 wear out될수 있도록 control하는 기법

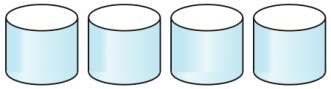
Chapter 10: Storage and File Structure

- 10.1 Overview of Physical Storage Media
- 10.2 Magnetic Disk and Flash Storage
- 10.3 RAID
- 10.4 Tertiary Storage
- 10.5 File Organization
- 10.6 Organization of Records in Files
- 10.7 Data-Dictionary Storage
- 10.8 Database Buffer

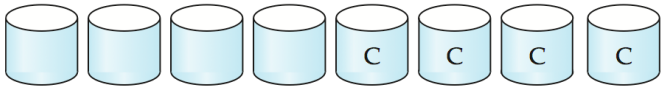
RAID

- **RAID: Redundant Arrays of Independent Disks**
  - disk organization technique in SAN (storage area network)
  - providing a logical view of a single disk of
    - high capacity and high speed by using multiple disks in parallel
    - high reliability by storing data redundantly, so that data can be recovered even if a disk fails
- 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)
  - Therefore, techniques for using redundancy to avoid data loss are critical with large numbers of disks
- Originally a cost-effective alternative to large, expensive disks
  - I in RAID originally stood for ``inexpensive''
  - Today RAIDs are used for their higher reliability and bandwidth
    - The "I" is interpreted as independent

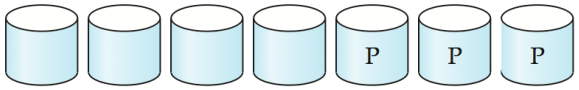
Figure 10.03: RAID levels



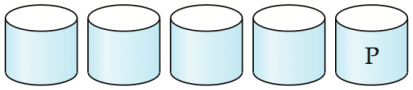
(a) RAID 0: nonredundant striping



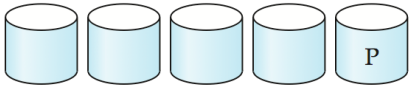
(b) RAID 1: mirrored disks



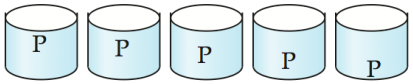
(c) RAID 2: memory-style error-correcting codes



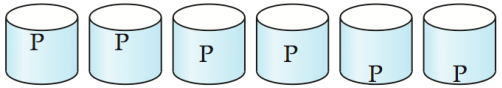
(d) RAID 3: bit-interleaved parity



(e) RAID 4: block-interleaved parity



(f) RAID 5: block-interleaved distributed parity



(g) RAID 6: P + Q redundancy

Improvement of Reliability via Redundancy

- **Redundancy** – Redundant information can be used to rebuild information lost in a disk failure
- **Mirroring** (or **shadowing or copying**)
  - Duplicate every disk (So, Logical one disk consists of two physical disks)
  - Every write is carried out on both disks
    - Reads can take place from either disk
  - 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 this case is very small except for dependent failure modes such as fire or building collapse or electrical power surges
- Mean time to data loss (MTDL) depends on mean time to failure (MTTF) and mean time to repair (MTTR)
  - MTDL = (MTTH) ** 2 / 2 * MTTR [Chen et al, ACM Computing Survey 1994]
  - E.g. MTTF of 100,000 hours, MTTR of 10 hours gives mean time to data loss of 500*106 hours (or 57,000 years) for a mirrored pair of disks (ignoring dependent failure modes)

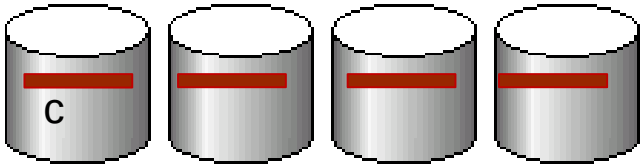
Improvement in Performance via Parallelism

- 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.
- Improve transfer rate by striping data across multiple disks.
- **Bit-level striping** – split the 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.
  - But seek/access time worse than for a single disk
    - Bit level striping is not used much any more
- **Block-level striping** – with *n* disks, block *i* of a file goes to disk (i mod n) + 1
  - 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

RAID Levels [1/6]

- Different RAID organizations provide redundancy at lower cost by using disk striping combined with parity bits
  - RAID levels have different cost, performance and reliability characteristics

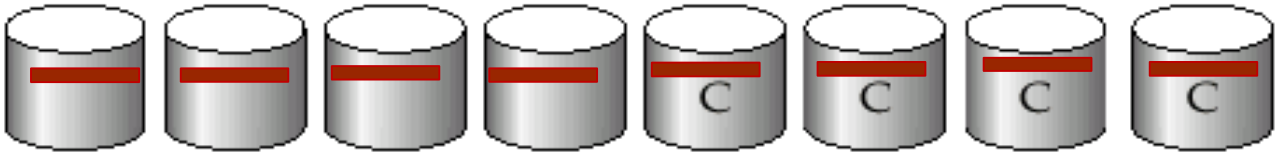
- **RAID Level 0**: Block striping; non-redundant.
  - Used in high-performance applications where data loss is not critical



(a) RAID 0: nonredundant striping

What if
one disk is defected?

- **RAID Level 1**: Mirrored disks with block striping (popular!)
  - Offers best write performance.
  - Popular for applications such as storing log files in a database system



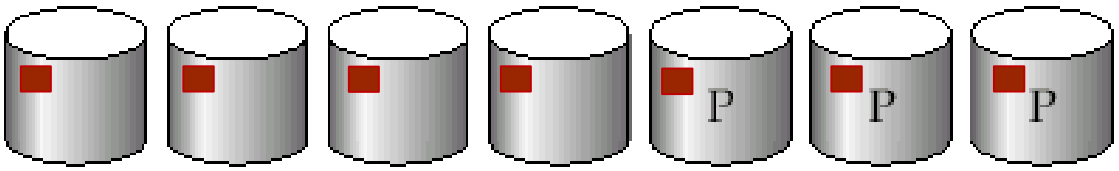
(b) RAID 1: mirrored disks

RAID Levels [2/6]

- **RAID Level 2**: Memory-Style Error-Correcting-Codes (ECC) with bit striping
  - 1 parity bit for 8 bit data for error detection
  - 2 or more extra bits for error correction
  - Storage efficient than RAID Level 1
    - But not used by any of the commercially available systems.

    (Always worse than RAID Level 3)

  - Suppose we have 4 bits records



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

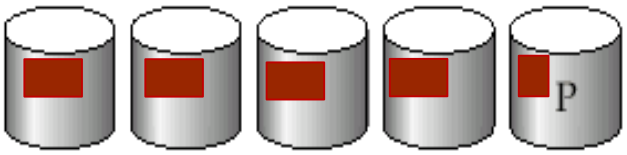
What if
one disk is defected?

RAID Levels [3/6]

RAID Level 3: Bit-Interleaved Parity

- A single parity bit is enough for error correction, not just detection
  - The disk controller knows which sector it is damaged (unlike memory system)
    - For each bit B in the sector, compute XOR of corresponding bits from other disks (including parity bit disk)
    - If the parity of the remaining bits is equal to B, the missing bit is 0, otherwise it is 1
- Faster data transfer than with a single disk, but fewer I/Os per second since every disk has to participate in every I/O
- Subsumes Level 2 : same performance with RAID Level 2 but needs less disks
- But not commonly used in practice because parity disk becomes a bottleneck

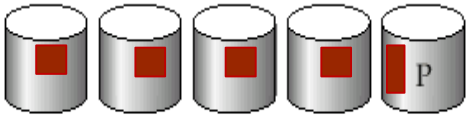
  (RAID Level 4 is not commonly used for the same reason)



(d) RAID 3: bit-interleaved parity

What if
one disk is defected?

RAID Levels [4/6]



(e) RAID 4: block-interleaved parity

- **RAID Level 4**: Block-Interleaved Parity
  - Before writing a block, corresponding block of parity data must be computed and written to a parity disk
    - 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
  - To find value of a damaged block, compute XOR of bits from corresponding blocks (including parity block) from other disks
  - Provides higher I/O rates for independent block reads than Level 3
    - A 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
  - Parity block may become a bottleneck for independent block writes since every block write also writes to parity disk

RAID Levels [5/6]

- **RAID Level 5**: Block-Interleaved Distributed Parity (Popular!)
  - partitions data and parity among all N + 1 disks, rather than storing data in N disks and parity in 1 disk
  - E.g., with 5 disks, parity block for nth set of blocks is stored on one disk (*n mod* 5) + 1, with the data blocks stored on the other 4 disks
  - 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

(f) RAID 5: block-interleaved distributed parity

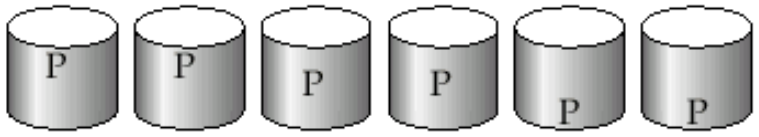
P0	0	1	2	3
4	P1	5	6	7
8	9	P2	10	11
12	13	14	P3	15
16	17	18	19	P4

4 bit data vs 1bit parity

What if one disk is defected?

RAID Levels [6/6]

- **RAID Level 6**: P+Q Redundancy scheme
  - Similar to Level 5 (Block-Interleaved Distributed Parity)
  - But stores extra redundant information to guard against multiple disk failures
    - Heavy error correcting code
      - Reed-Solomon code (2 parity bits for 4 bit data)
    - Duplicate Parity Block
  - Better reliability than Level 5 at a higher cost (too much!, not used as widely)



(g) RAID 6: P + Q redundancy

What if
two disks are defected?

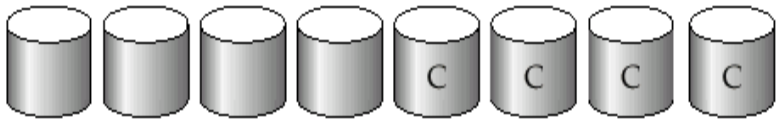
Disk1	Disk2	Disk3	Disk4	Disk5	Disk6
p0	p0	b0	b1	b2	b4
b5	b6	p1	p1	b7	b8
b9	b10	b11	b12	p2	p2
p3	p3	b13	b14	b15	b16
b17	b18	p4	p4	b19	b20

Choice of RAID Level [1/2]

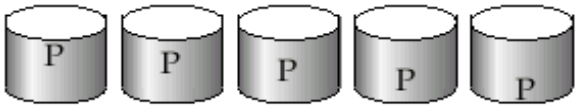
- Factors in choosing RAID level
  - Monetary cost
  - Performance: Num of I/Os per second, and Bandwidth during normal operation
  - Performance during failure
  - Performance during rebuild of failed disk
    - Including time taken to rebuild failed disk
- RAID 0 is used only when data safety is not 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 cases
- So in most cases, competition is between 1 and 5 only

Choice of RAID Level: Level 1 vs Level 5 [2/2]

- Disk and I/O trend
  - Disk drive capacities increasing rapidly (50% / year)
  - Disk access times have decreased much less (x 3 times in 10 years)
  - I/O requirements have increased greatly, e.g. for Web servers
- Level 1 provides much better write performance than level 5
  - Level 5 requires 2 block writes and additional block reads for making a new parity block to write a single block, whereas Level 1 only requires 2 block writes
- Level 1 had higher storage cost than level 5
  - 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!
- Therefore, Level 1 is preferred for all other applications
  - Level 1 preferred for high update environments such as log disks
- Level 5 is preferred for applications with low update rate and large data



(b) RAID 1: mirrored disks



(f) RAID 5: block-interleaved distributed parity

Upgrade RAID with Additional Hardware [1/2]

- **Software RAID**: RAID implementations done entirely in software, with no special hardware support
- **Hardware RAID**: RAID implementations done with special hardware support

- **Utilizing NV(non-volatile)-RAM**
  - Keep writes in NV-RAM until writes are executed successfully in RAID
  - Suppose failure happens 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

Upgrade RAID with Additional Hardware [2/2]

- Failures during IO: re-send or parity checking
- Latent failures (or Bit rot): (Latent: 숨어있는, 잠복의)
  - Data successfully written earlier gets damaged
  - Can result in data loss even if only one disk fails
- **Data scrubbing**:
  - Whenever time permits, disk controller continually scan for latent failures, and recover from copy or with parity
  - **Utilizing other disk**
- **Hot swapping**:
  - Replacement of disk with **online spare disks** without power down
  - Can reduce time to recovery, and improves availability greatly
- **Redundant power supplies with battery backup** to cope with power failure
- **Multiple controllers and multiple interconnections** to guard against controller/interconnection failures

Chapter 10: Storage and File Structure

- 10.1 Overview of Physical Storage Media
- 10.2 Magnetic Disk and Flash Storage
- 10.3 RAID
- 10.4 Tertiary Storage
- 10.5 File Organization
- 10.6 Organization of Records in Files
- 10.7 Data-Dictionary Storage
- 10.8 Database Buffer

Optical Disks

- Compact disk-read only memory (CD-ROM)
  - Removable disks, 640 MB per disk
  - Seek time about 100 micro secs (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 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

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
  - 330 GB with Ampex helical scan format
  - Transfer rates from few to 10s of MB/s
- Currently the cheapest storage medium
  - 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)

Chapter 10: Storage and File Structure

- 10.1 Overview of Physical Storage Media
- 10.2 Magnetic Disk and Flash Storage
- 10.3 RAID
- 10.4 Tertiary Storage
- 10.5 File Organization
- 10.6 Organization of Records in Files
- 10.7 Data-Dictionary Storage
- 10.8 Database Buffer

File Organization

- The database is stored as a collection of files of the underlying OS
  - Each file is a sequence of records.
  - A record is a sequence of fields
- Each file is logically partitioned into fixed-length storage units called blocks
  - Most DB uses block sizes of 4 to 8 KB
- We assume
  - No record is larger than a block
  - Each record is entirely contained in a single block
  - Different files are used for different relations
  - Each file has records of one particular type only
- Physical File Organization for a Logical “Relation”
  - Fixed-length record representation
  - Variable-length record representation

C - File I/O

Opening Files

You can use the **fopen( )** function to create a new file or to open an existing file. This call will initialize an object of the type **FILE**, which contains all the information necessary to control the stream. The prototype of this function call is as follows –

```
FILE *fopen( const char * filename, const char * mode );
```

Closing a File

To close a file, use the fclose( ) function. The prototype of this function is –

```
int fclose( FILE *fp );
```

Writing a File

Following is the simplest function to write individual characters to a stream –

```
int fputc( int c, FILE *fp );
```

Reading a File

Given below is the simplest function to read a single character from a file –

```
int fgetc( FILE * fp );
```

But!
We want to manipulate records and attributes one by one and in a random sequence!

Fixed-Length Record Representation

instructor

Relation == Logical Representation

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000

- Simple Fixed-Length Record Representation:
  - Store record *i* starting from byte $n * (i - 1)$, where *n* is the size of each record
  - Record access is simple but records may cross blocks
    - Modification: do not allow records to cross block boundaries
- Deletion of record *i:* alternatives*:*
  - move records $i + 1, \ldots, n$ to $i, \ldots, n - 1$
  - move record *n* to *i*
  - 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

Simple Fixed-Length Record Representation

Deleting record 3 and compacting

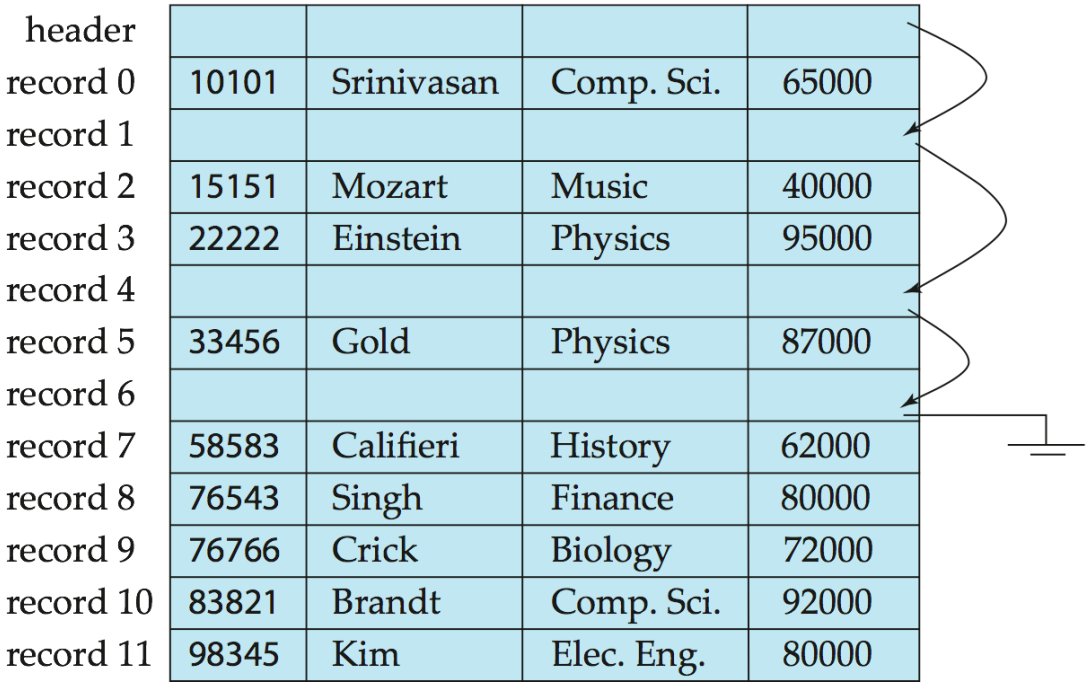
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766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

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

Utilizing Free Lists

- 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
- Can think of these stored addresses as pointers since they "point" to the location of a record.
- More space efficient representation: reuse space for normal attributes of free records to store pointers. (No pointers stored in in-use records.)



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

Saving Repeated Data Space using Repeating Field Representation

- A bank branch has a number of accounts

account_num	branch_name	balance
A-101	Downtown	500
A-110	Downtown	600
A-102	Perryridge	400
A-201	Perryridge	900
A-218	Perryridge	700
A-305	Round Hill	350
A-215	Mianus	350

- Record types that allow repeating fields
  - Perryridge (A-102,400) (A-201, 900) (A-218, 700)
  - RoundHill (A-305, 350)
  - Mianus (A-215,700)

- Reserved space method – can use fixed-length records of a known maximum length
  - unused space in shorter records filled with a null or end-of-record symbol.

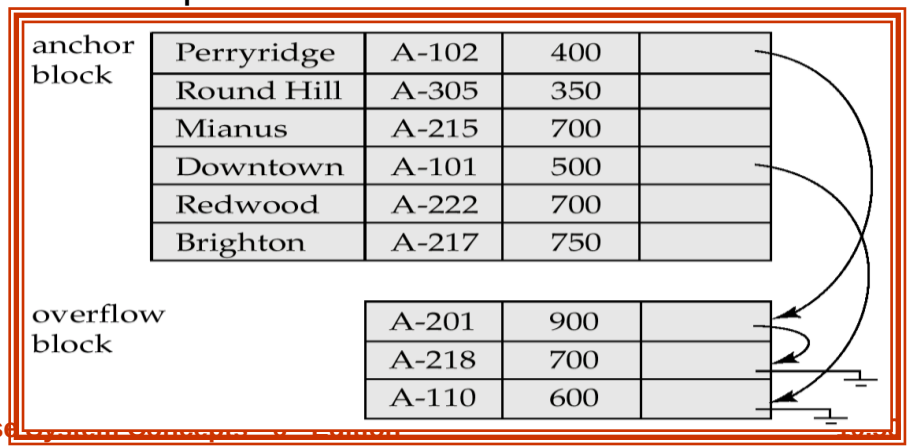
0	Perryridge	A-102	400	A-201	900	A-218	700
1	Round Hill	A-305	350	⊥	⊥	⊥	⊥
2	Mianus	A-215	700	⊥	⊥	⊥	⊥
3	Downtown	A-101	500	A-110	600	⊥	⊥
4	Redwood	A-222	700	⊥	⊥	⊥	⊥
5	Brighton	A-217	750	⊥	⊥	⊥	⊥

Repeating Field Representations

- A variable-length record is represented by a list of fixed-length records, chained via pointers.
  - Can be used even if the maximum record length is not known
  - space is wasted in all records except the first in a chain

account_num	branch_name	balance
A-101	Downtown	500
A-110	Downtown	600
A-102	Perryridge	400
A-201	Perryridge	900
A-218	Perryridge	700
A-305	Round Hill	350
A-215	Mianus	350

- Better pointer method with overflow blocks



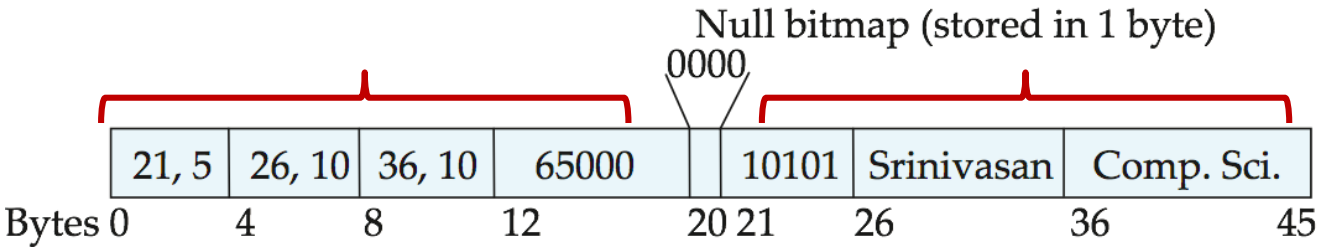
Saving Empty Space inside Record Field using Variable-Length Record Representation

instructor

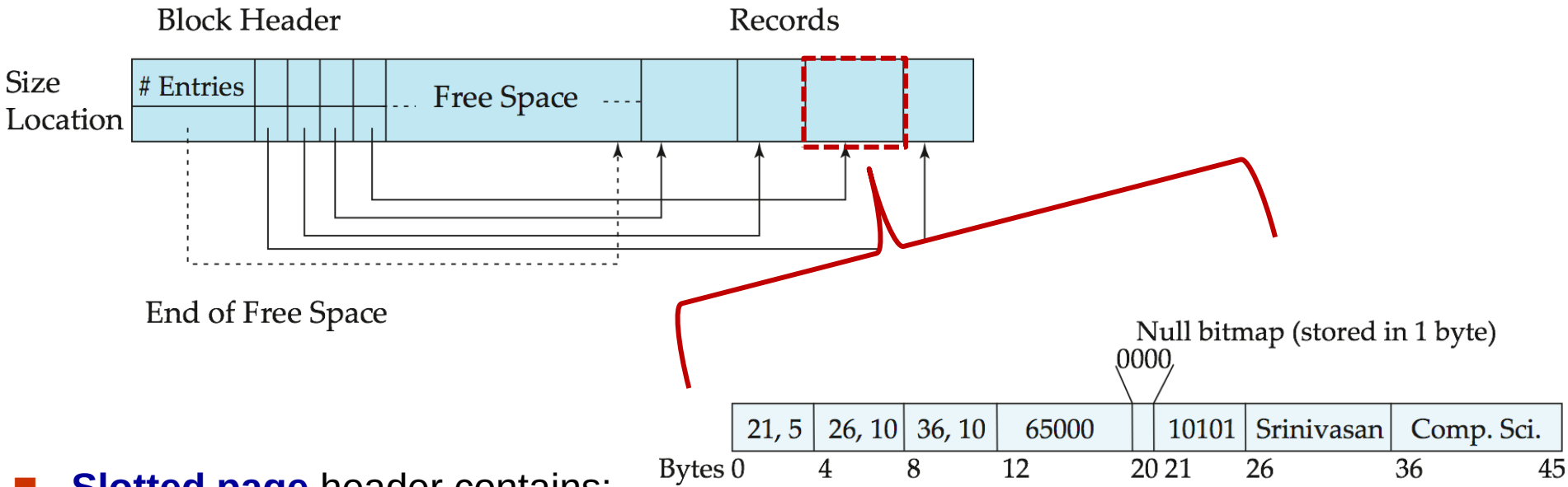
Relation == Logical Representation

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000

- Attributes are stored in order
  - Initial part with fixed length attributes
  - Data for variable length attributes
- Variable length attributes represented by fixed size (offset, length), with actual data stored after all fixed length attributes
- Null values represented by null-value bitmap



Slotted Page Structure for Variable-Length Record Representation



- **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 around within a page to keep them contiguous with no empty space between them; entry in the header must be updated.
- Pointers should not point directly to record — instead they should point to the entry for the record in header.

Chapter 10: Storage and File Structure

- 10.1 Overview of Physical Storage Media
- 10.2 Magnetic Disk and Flash Storage
- 10.3 RAID
- 10.4 Tertiary Storage
- 10.5 File Organization
- 10.6 Organization of Records in Files
- 10.7 Data-Dictionary Storage
- 10.8 Database Buffer

Organization of Records in Files (Locating Records inside a File)

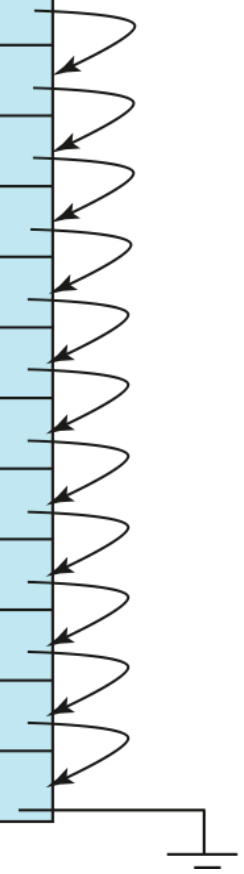
- **File내부에서 record들의 위치에 관한 문제**

- **Heap**
  - a record can be placed anywhere in the file where there is space
  - Sometimes called, Pile
- **Sequential**
  - store records in sequential order, based on the value of the search key of each record
- **Hashing** (chapter 11)
  - a hash function computed on some attribute of each record
  - the result specifies in which block of the file the record should be placed
- **Multi-table 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

Sequential File [1/2]

- Suitable for applications that require sequential processing of the entire file
- The records in the file are ordered by a 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 [2/2]

- 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
- Need to reorganize the file from time to time to restore sequential order

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	

32222	Verdi	Music	48000	

Example of Hash File Organization (ch 11)

Hash file organization of *instructor* file, using *dept_name* as key

- There are 10 buckets,
- The binary representation of the *i*th character is assumed to be the integer *i*.
- The hash function returns the sum of the binary representations of the characters modulo 10
  - E.g. h(Music) = 1 h(History) = 2 h(Physics) = 3 h(Elec. Eng.) = 3

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

bucket 0

bucket 1

15151	Mozart	Music	40000

bucket 2

32343	El Said	History	80000
58583	Califieri	History	60000

bucket 3

22222	Einstein	Physics	95000
33456	Gold	Physics	87000
98345	Kim	Elec. Eng.	80000

bucket 4

12121	Wu	Finance	90000
76543	Singh	Finance	80000

bucket 5

76766	Crick	Biology	72000

bucket 6

10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000

bucket 7

Multi-Table Clustering File [1/2]

Store several relations in one file using a **multitable clustering** file organization

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

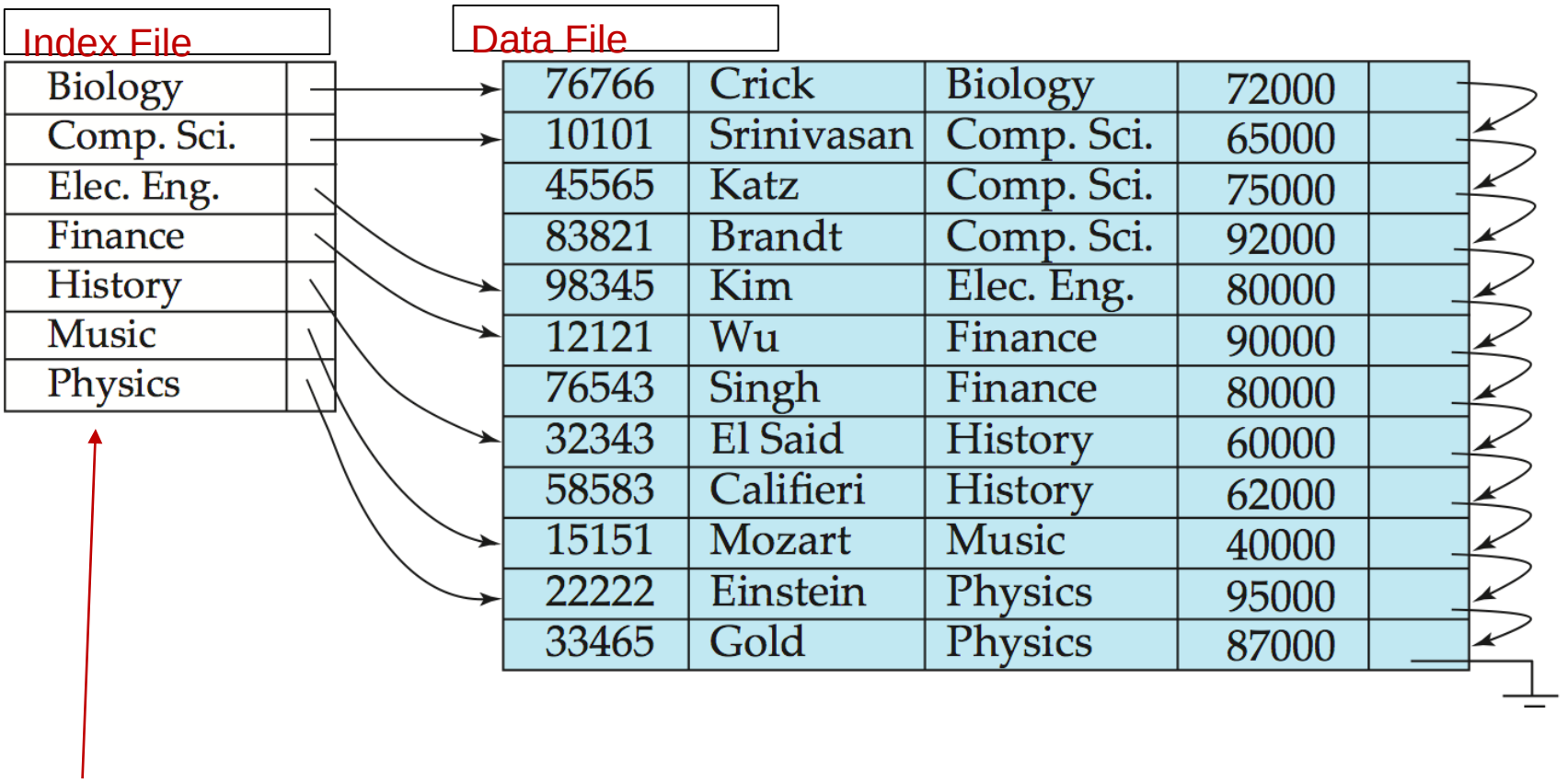
Multi-table Clustering File [2/2]

- good for queries involving *department* ⋈ *instructor*, and for queries involving one single department and its instructors
- bad for queries involving only *department*
- results in variable size records (with repeating fields)
- Can add pointer chains to link records of a particular relation

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

Data File 외부에 추가적인 Index File (ch 11)

- So far we concentrated on file structures for data records
- We will study various types of index method (extra access methods) in chapter 11



Chapter 10: Storage and File Structure

- 10.1 Overview of Physical Storage Media
- 10.2 Magnetic Disk and Flash Storage
- 10.3 RAID
- 10.4 Tertiary Storage
- 10.5 File Organization
- 10.6 Organization of Records in Files
- 10.7 Data-Dictionary Storage
- 10.8 Database Buffer

Data Dictionary Storage [1/2]

- **Data dictionary** (also called **system catalog**) stores **metadata** (data about data)
- Information about relations
  - names of relations
  - names, types and lengths of attributes of each relation
  - names and definitions of views
  - integrity constraints
- User and accounting information, including passwords
- Statistical and descriptive data
  - number of tuples in each relation
- Physical file organization information
  - How relation is stored (sequential / hash / …)
  - Physical location of relation
    - operating system file name or
    - disk addresses of blocks containing records of the relation
- Information about indices (Chapter 11)

```sql
create table classroom
    (building       varchar(15),
     room_number        varchar(7),
     capacity       numeric(4,0),
     primary key (building, room_number)
    );

create table department
    (dept_name      varchar(20),
     building       varchar(15),
     budget             numeric(12,2) check (budget > 0)
     primary key (dept_name)
    );

create table course
    (course_id      varchar(8),
     title          varchar(50),
     dept_name      varchar(20),
     credits        numeric(2,0) check (credits > 0),
     primary key (course_id),
     foreign key (dept_name) references department
        on delete set null
    );

create table instructor
    (ID         varchar(5),
     name           varchar(20) not null,
     dept_name      varchar(20),
     salary         numeric(8,2) check (salary > 29000),
     primary key (ID),
     foreign key (dept_name) references department
        on delete set null
    );
```

```sql
create table section
    (course_id      varchar(8),
         sec_id         varchar(8),
     semester       varchar(6)
        check (semester in ('Fall', 'Winter', 'Spring', 'Summer')),
     year           numeric(4,0) check (year > 1701 and year < 2100),
     building       varchar(15),
     room_number        varchar(7),
     time_slot_id       varchar(4),
     primary key (course_id, sec_id, semester, year),
     foreign key (course_id) references course
        on delete cascade,
     foreign key (building, room_number) references classroom
        on delete set null
    );

create table teaches
    (ID         varchar(5),
     course_id      varchar(8),
     sec_id         varchar(8),
     semester       varchar(6),
     year           numeric(4,0),
     primary key (ID, course_id, sec_id, semester, year),
     foreign key (course_id,sec_id, semester, year) references section
        on delete cascade,
     foreign key (ID) references instructor
        on delete cascade
    );

create table student
    (ID         varchar(5),
     name           varchar(20) not null,
     dept_name      varchar(20),
     tot_cred       numeric(3,0) check (tot_cred >= 0),
     primary key (ID),
     foreign key (dept_name) references department
        on delete set null
    );
```

Data Dictionary Storage [2/2]

- Catalog structure
  - a set of relations, with existing system features used to ensure efficient access
  - A possible catalog representation:

Relation-metadata = (relation-name, number-of-attributes, storage-organization, location)

Attribute-metadata = (attribute-name, relation-name, domain-type, position, length)

User-metadata = (user-name, encrypted-password, group)

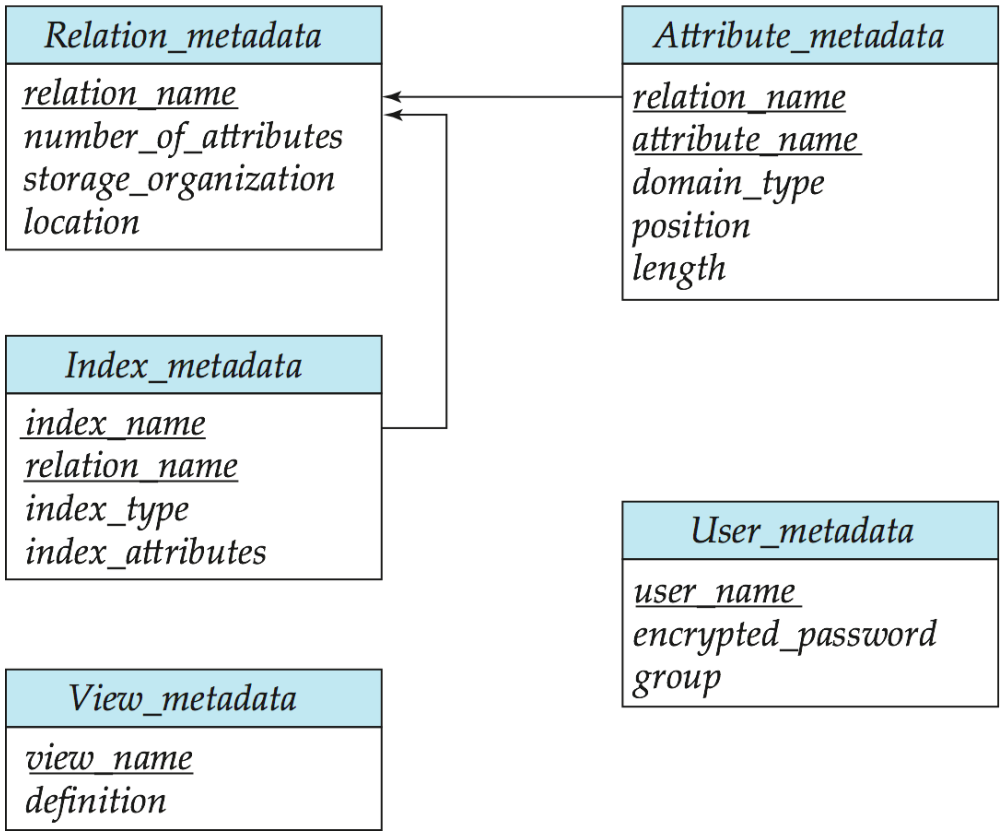
Index-metadata = (index-name, relation-name, index-type, index-attributes)

View-metadata = (view-name, definition)

- For example: Whenever DBMS needs to perform queries on a relation, DBMS must consult the Relation_metadata relation to find the location and storage organization of the relation, and then fetch records using the information

Relational Representation of System Metadata

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



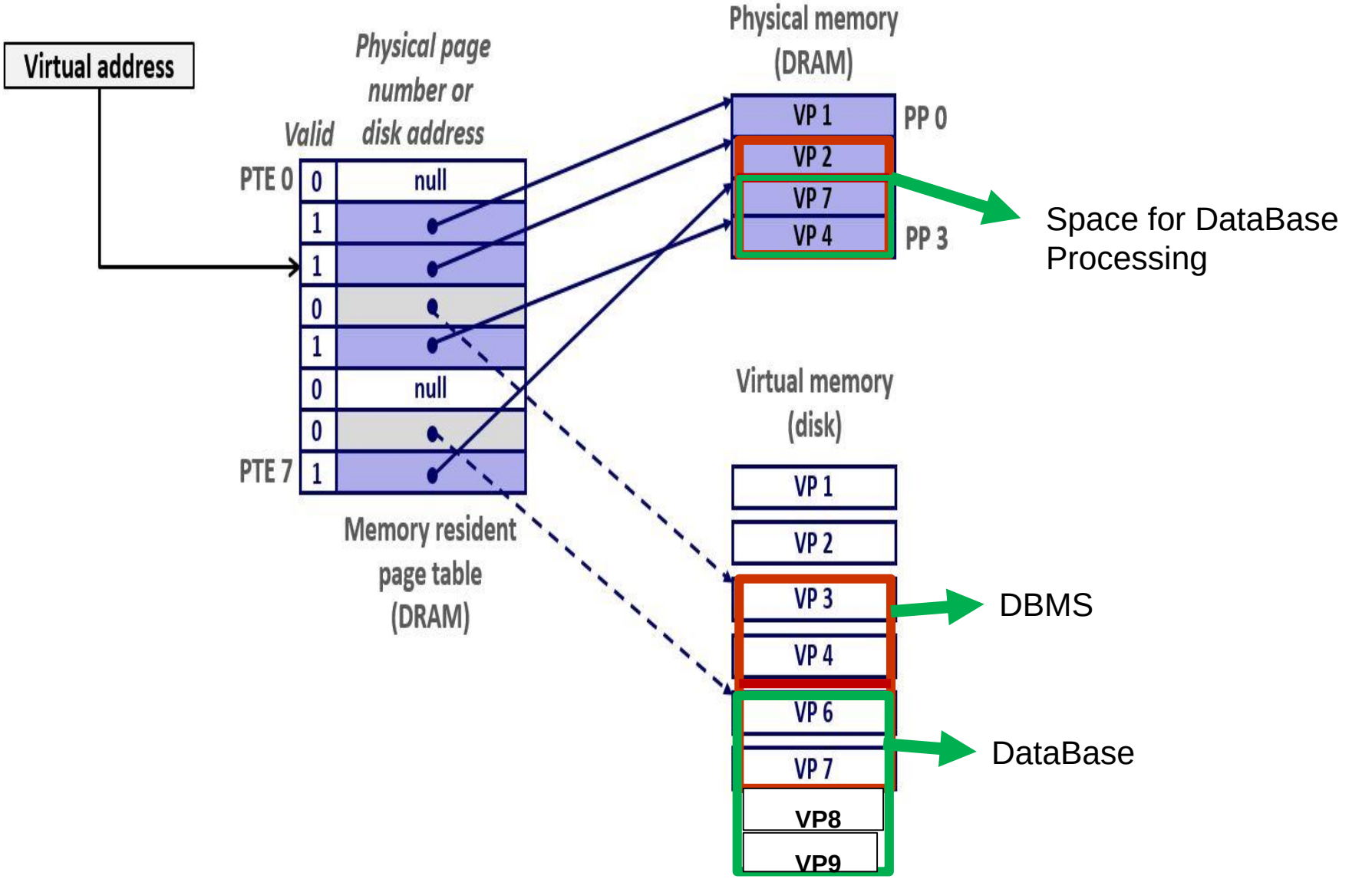
Chapter 10: Storage and File Structure

- 10.1 Overview of Physical Storage Media
- 10.2 Magnetic Disk and Flash Storage
- 10.3 RAID
- 10.4 Tertiary Storage
- 10.5 File Organization
- 10.6 Organization of Records in Files
- 10.7 Data-Dictionary Storage
- 10.8 Database Buffer

Storage Access

- A database file is partitioned into fixed-length storage units called **blocks**
- Blocks are units of both storage allocation and data transfer
- Database system seeks to minimize the number of block transfers between the disk and memory
  - We can reduce 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

O/S Page Buffering



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. The buffer manager allocates space in the buffer for the block, replacing (throwing out) some other block, if required, to make space for the new block.
     2. The block that is thrown out is written back to disk only if it was modified since the most recent time that it was written to/fetched from the disk.
     3. Once space is allocated in the buffer, the buffer manager reads the block from the disk to the buffer, and passes the address of the block in main memory to requester.

Buffer-Replacement Policies [1/2]

- Most operating systems replace the block **least recently used** (LRU strategy)
  - use past pattern of block references as a predictor of future references

- 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
  - LRU can be a bad strategy for certain access patterns involving repeated scans of data
    - For example: when computing the join of 2 relations r and s by a nested loops

      for each tuple *tr* of *r* do
      for each tuple *ts* of *s* do
      if the tuples *tr* and *ts* match …

  - Mixed strategy with hints on replacement strategy provided by the query optimizer is preferable

Nested-loop join

보조자료

Loan relation : N = 12, B = 3

Loan number	Branch name	amount
L- 170	Downtown	300
L – 42	Redwood	400
L – 48	Redwood	1500
L – 112	Perryridge	2300
L – 321	Redwood	3100
L – 90	Downtown	800
L – 112	Perryridge	2300
L – 31	Redwood	200
L - 70	Perryridge	600
L - 221	Downtown	1000
L – 155	Redwood	800
L - 320	Downtown	2500

Borrower relation: N = 16, B = 4

Customer name	Loan number
Jones	L - 170
Bahn	L – 82
Kim	L – 42
Lee	L – 48
Jane	L – 112
Smith	L – 34
Hwang	L – 321
Choi	L – 109
Pedro	L – 90
Sammy	L – 112
Jun	L – 31
Jung	L – 62
Shin	L – 99
Koh	L – 70
Mark	L – 221
Harry	L - 116

Buffer-Replacement Policies [2/2]

- Buffer manager can use **Alternatives to LRU**
  - **Pinned block** – memory block that is not allowed to be written back to disk.
  - **Toss-immediate** strategy – frees the space occupied by a block as soon as the final tuple of that block has been processed
  - **Most recently used (MRU) strategy**
    - 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 managers also support **forced output** of blocks for the purpose of recovery (more in Chapter 16)