



# Chapter 10: Storage and File Structure

**Database System Concepts, 6<sup>th</sup> Ed.**

©Silberschatz, Korth and Sudarshan

See [www.db-book.com](http://www.db-book.com) for conditions on re-use



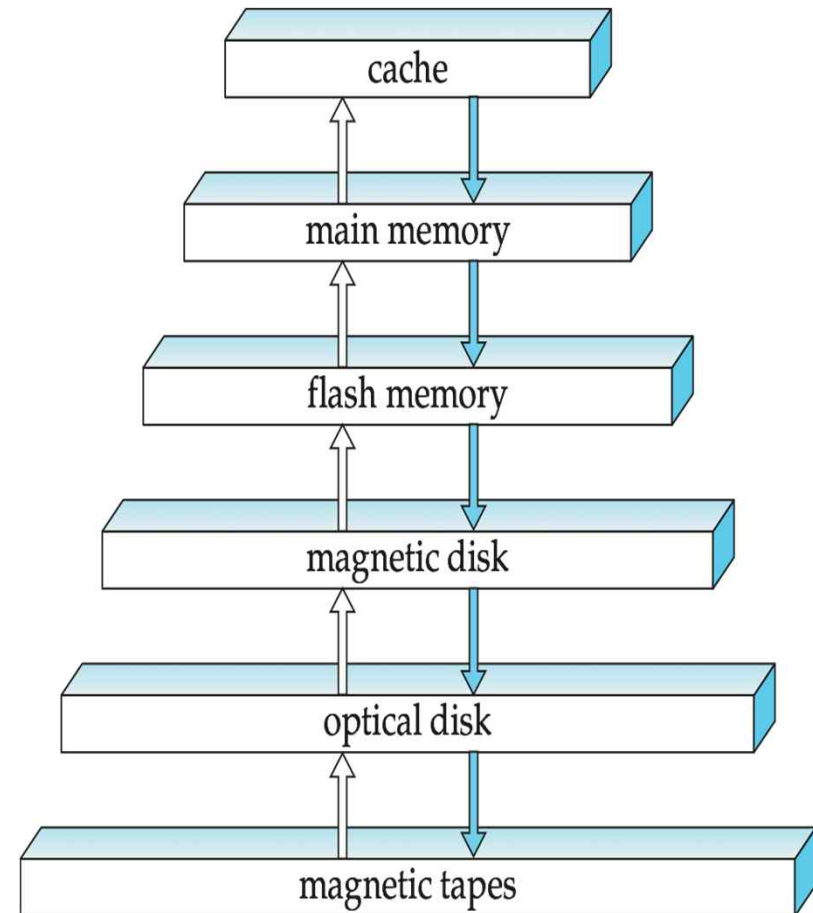
# Classification of Physical Storage Media

- Data access speed
- Cost per unit of data
- Reliability
  - Data loss on power failure or system crash
  - Physical failure of the storage device
- Persistence
  - **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.



# 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





# Primary Storage

## ■ 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



# Flash Memory

## ■ 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



# Magnetic Disk

## ■ 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



# Tertiary Storage

## ■ Optical storage

- Non-volatile, data is read optically from a spinning disk using a laser
- CD-ROM (640 MB), DVD (4.7 to 17 GB), Blu-ray (27 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

## ■ Tape storage

- Non-volatile, used primarily for backup (to recover from disk failure), and for archival data
- Storage costs much cheaper than disk
- **Sequential-access** – much slower than disk
- Very high capacity (40 to 300 GB tapes available)



# Magnetic Disks

## ■ Read-write head

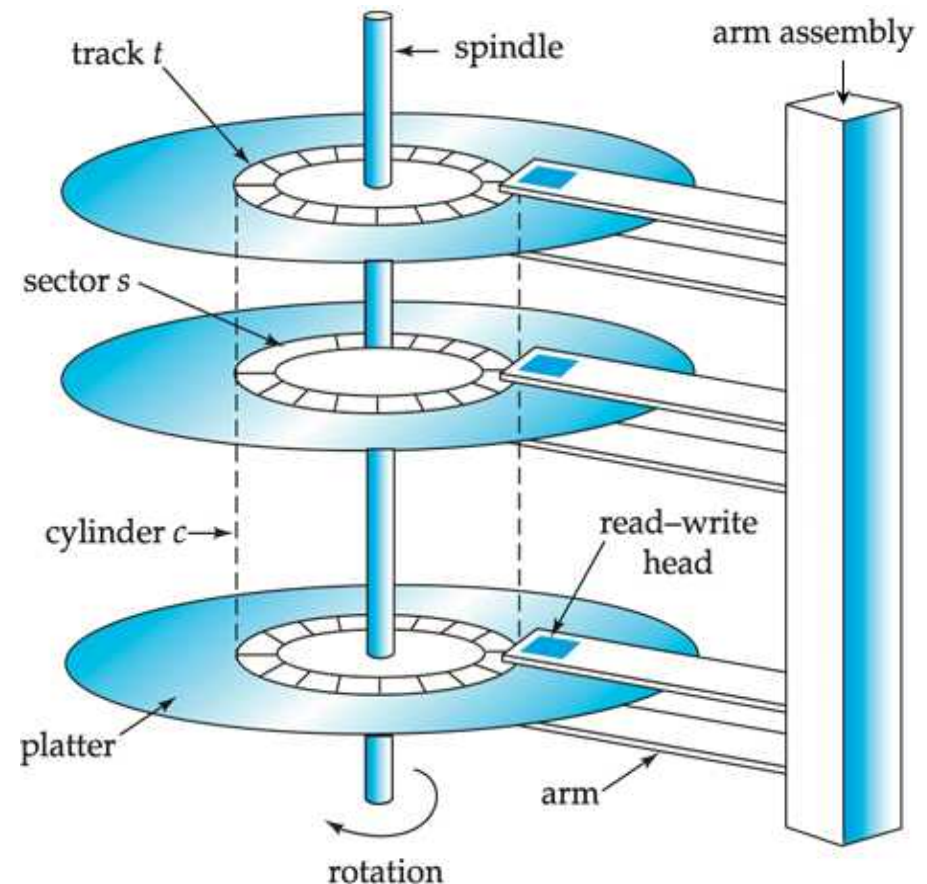
- Positioned very close to the platter surface
- Reads or writes magnetically encoded information

## ■ 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)

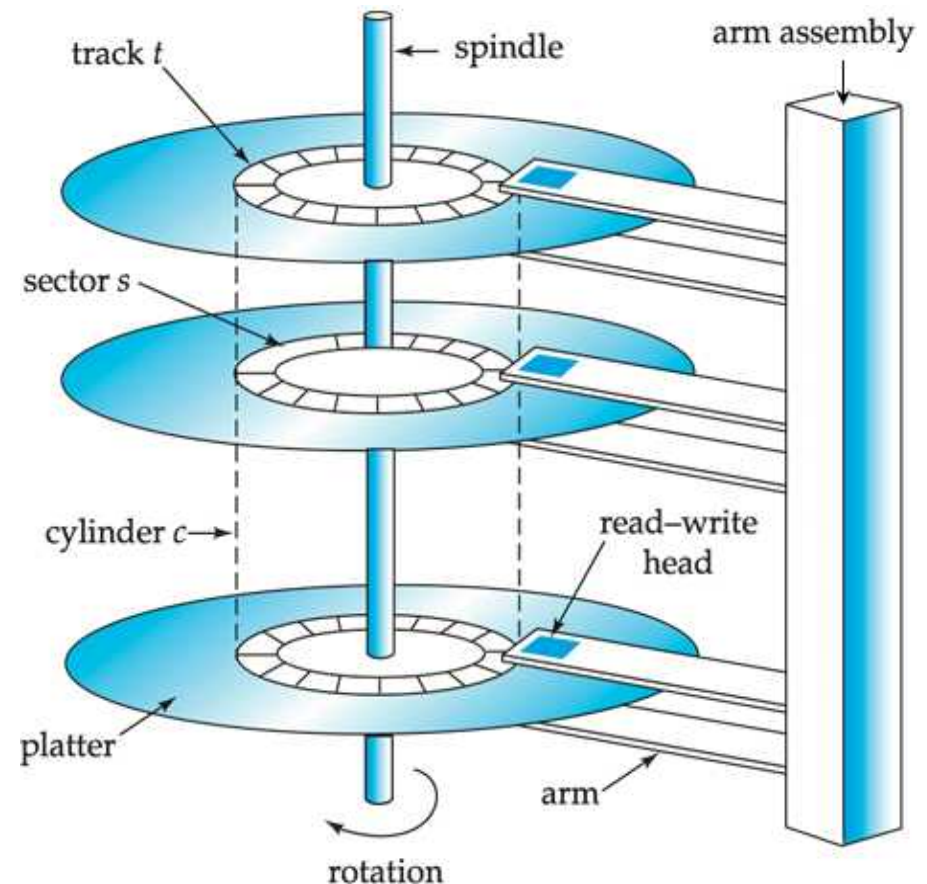






# Magnetic Disk Mechanism

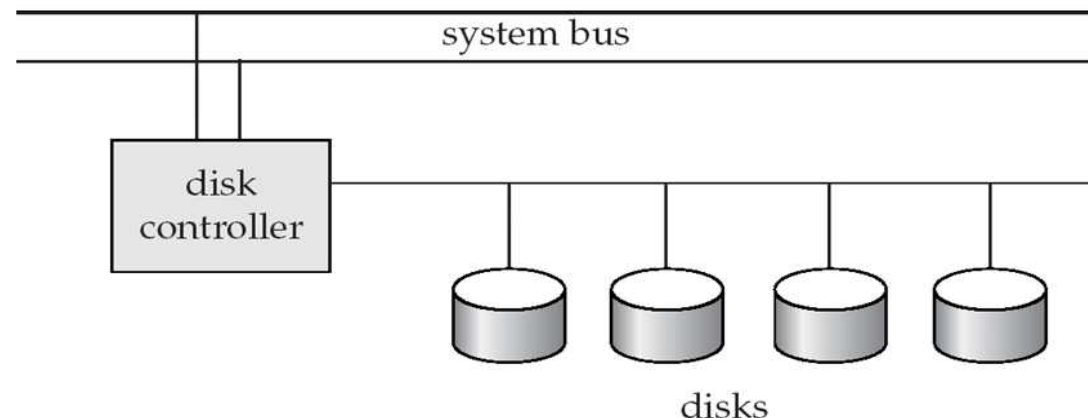
- To read/write a sector
  - Disk arm swings to position head on right track
  - Platter spins continually; data is read/written as sector passes under head
- Head-disk assemblies
  - Multiple disk platters on a single spindle (1 to 5 usually)
  - One head per platter, mounted on a common arm
- **Cylinder**  $i$  consists of  $i^{th}$  track of all the platters





# Disk Controller

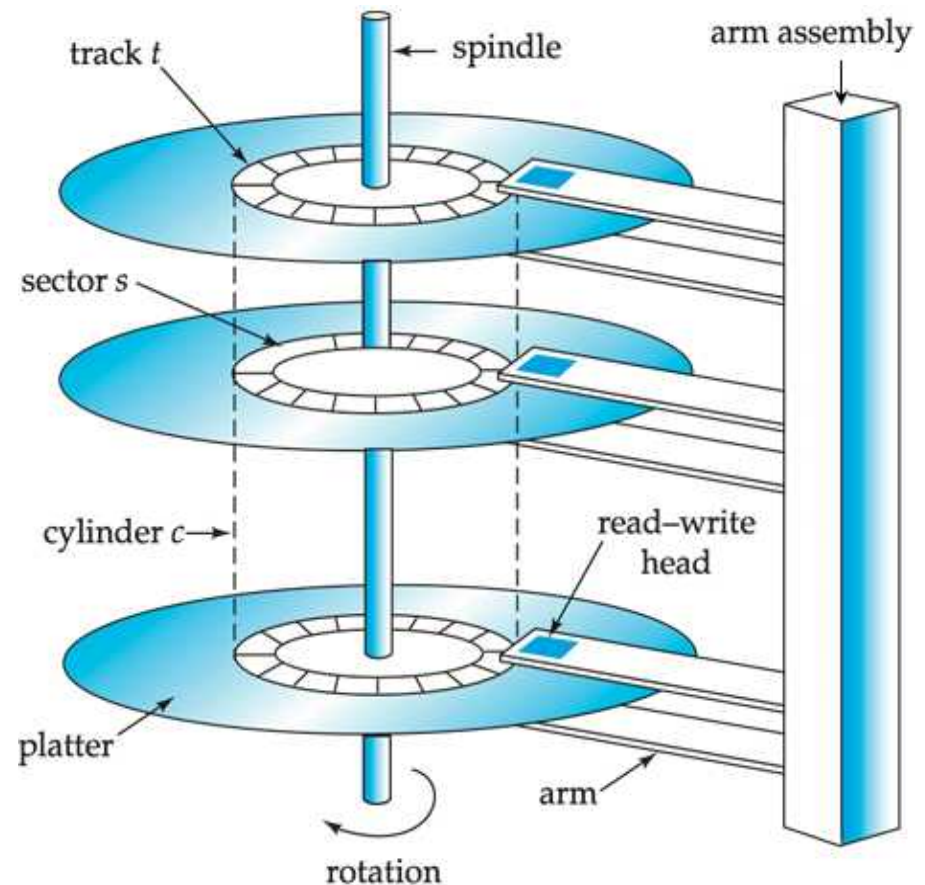
- Interfaces between the computer system and the disk drive hardware
  - Accepts high-level commands to read or write a sector
  - **Initiates actions** such as moving the disk arm to the right track and actually reading or writing the data
  - Manages data quality and robustness
    - ▶ Computes and attaches **checksums** to each sector to verify that data is read back correctly
    - ▶ Ensures successful writing by reading back sector after writing it
    - ▶ Performs **remapping of bad sectors**
- **Multiple disks** connected to a computer system through a controller





# Performance Measures of Disks

- **Access time** – the time it takes from when a read or write request is issued to when data transfer begins
  - **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
    - ▶ 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 rpm)





# Performance Measures (Cont.)

- **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, Fiber Channel (FC2Gb or 4Gb): 256 to 512 MB/s
  
- **Mean time to failure (MTTF)** – the average time the disk is expected to run continuously without any failure.
  - Typically 3 to 5 years (1 year = 8760 hrs)
  - 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



# Optimization of Disk-Block Access

- **Block** – a 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-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



# Optimization of Disk Block Access (Cont.)

## ■ File organization

- Need to optimize block access time by organizing 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 (Cont.)

- **Non-volatile RAM:** 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
  - Database operations that require data to be safely stored before continuing can continue without waiting for data to be written to disk
  - Writes can be reordered to minimize disk arm movement
  
- **Log disk:** a disk 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
  - No need for special hardware such as NV-RAM



# RAID

## ■ 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,
  - ▶ high reliability by storing data redundantly, so that data can be recovered even if a disk fails
  
- 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





# Improvement of Reliability via Redundancy

- **Redundancy** – store extra information that can be used to rebuild information lost in a disk failure
- E.g., **Mirroring** (or **shadowing**)
  - Duplicate every disk
  - Write on both disks / Read from either disk
  - If one disk 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
- **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  $500 \times 10^6$  hours (or 57,000 years) for a mirrored pair of disks (ignoring dependent failure modes)



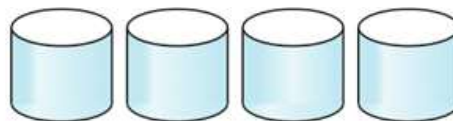
# Improvement in Performance via Parallelism

- 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  $(i \bmod n) + 1$ 
  - 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

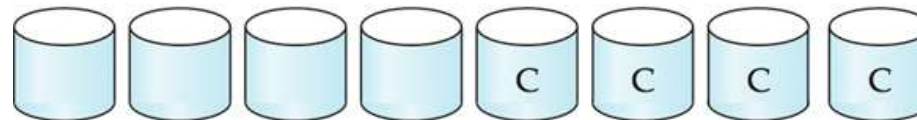


# RAID Levels

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



(a) RAID 0: nonredundant striping



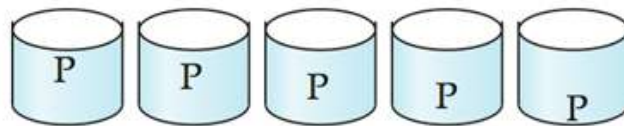
(b) RAID 1: mirrored disks



## RAID Levels (Cont.)

### ■ 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  $n$ th set of blocks is stored on **one disk** ( $n \bmod 5$ ) + 1, with the data blocks stored on **the other 4 disks**



(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

- Compared to level 1, lower storage overhead but higher time overhead for writes: popular for applications with frequent reads with rare writes



# File Organization

- The database is stored as a collection of **files**
  - Each file is a sequence of **records**
  - A record is a sequence of **fields**
  - These are stored in units of **blocks**
  
- How to represent records in a file structure
  - Fixed-length records
    - ▶ Assume record size is fixed
    - ▶ Each file has records of one particular type only
    - ▶ Different files are used for different relations
  - Variable length records
    - ▶ Multiple record types are stored in a file
    - ▶ Record types with variable lengths are allowed such as strings (**varchar**)



# Fixed-Length Records

- Simple approach:
  - 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

- **Alternatives** for deletion of record  $i$ :

1. Move records  $i + 1, \dots, n$  to  $i, \dots, n - 1$
2. Move record  $n$  to  $i$
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

1. Move records  $i + 1, \dots, n$  to  $i, \dots, n - 1$

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

2. Move record  $n$  to  $i$

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





# 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

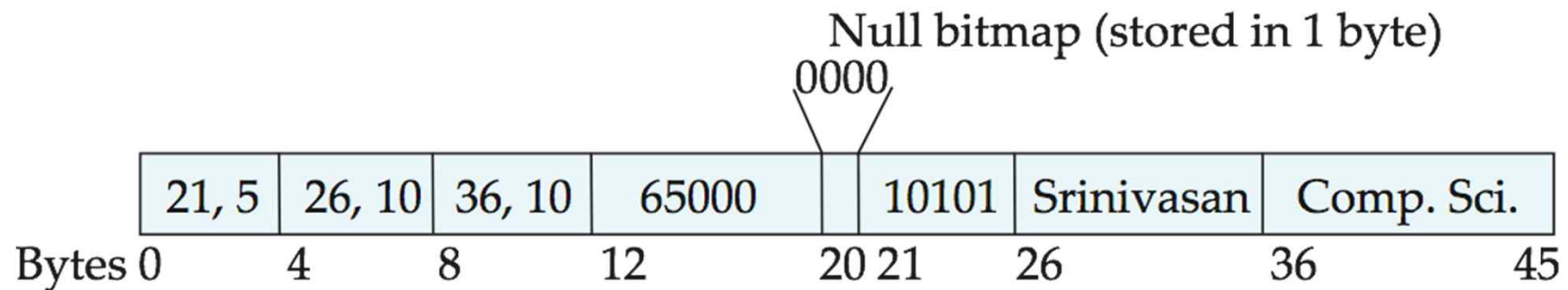
3. Do not move records, but link all free records on a *free list*

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



# Variable-Length Records

- Variable-length records arise in database systems in several ways:
  - Storage of multiple record types in a file
  - Record types that allow variable lengths for one or more fields such as strings (**varchar**)
  - Record types that allow repeating fields (used in some older data models)
- Attributes are stored in order
- Variable length attributes represented by fixed size (offset, length), with actual data stored after all fixed length attributes
- Null values represented by **null bitmap**





# Representation Methods for Variable-Length Records

## ■ Byte string representation

- Attach an *end-of-record* ( $\perp$ ) control character to the end of each record
- Difficulty with deletion / growth

## ■ Fixed length representation

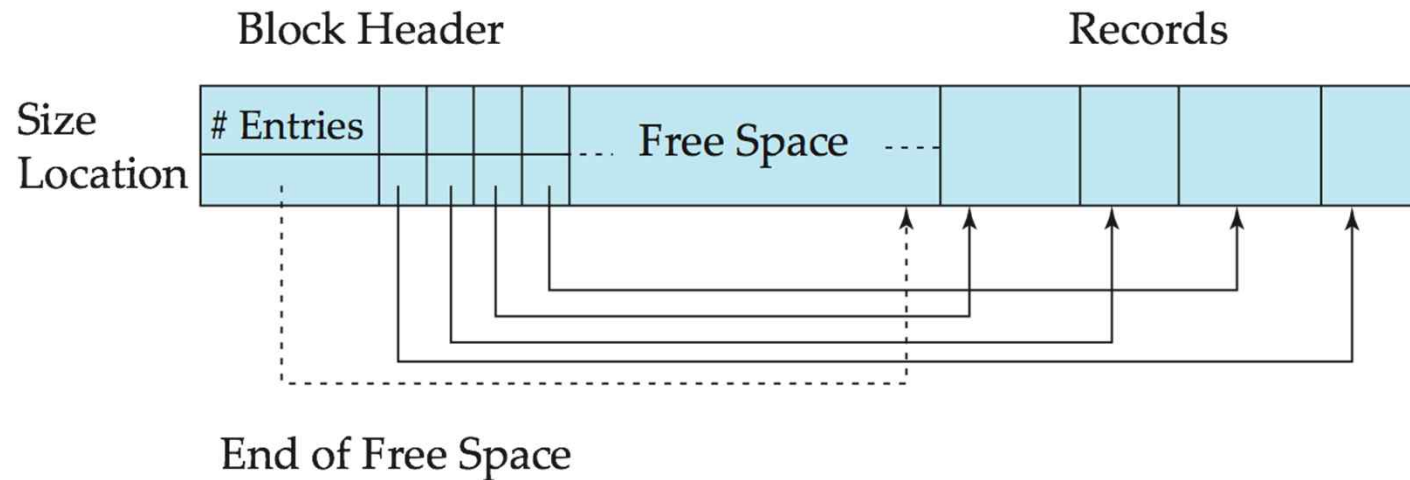
- Reserved space method
  - ▶ **Reserved space** – can use fixed-length records of a known maximum length
  - ▶ unused space in shorter records filled with a null or end-of-record symbol
- Pointer method

## ■ Slotted page method

- The most widely used method



# Variable-Length Records: Slotted Page Structure



- **Slotted page** header contains:
  - number of record entries
  - end of free space in the block
  - location and size of each record
- Records can be moved 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.



# Organization of Records in Files

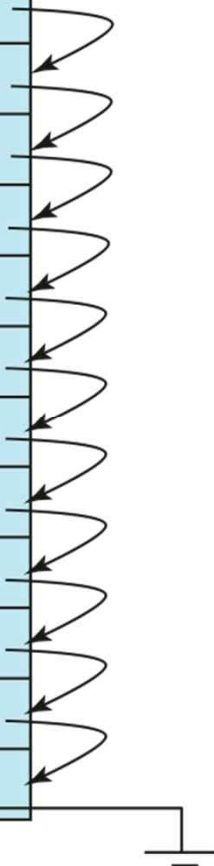
- How to organize records in files
- **Heap**
  - A record can be placed anywhere in the file where there is space
  - Sometimes called, **Pile**
- **Sequential**
  - Store records in sequential order, based on the value of the search key of each record
- **Hashing**
  - A hash function computed on some attribute of each record
  - The result specifies in which block of the file the record should be placed
- Records of each relation may be stored in a separate file
- Records of several different relations can be stored in the same file
  - **Multi-table clustering file organization**
    - ▶ Motivation: store related records on the same block to minimize I/O



# Sequential File Organization

- Suitable for applications that require sequential processing of the entire file
- The records in the file are ordered by a **search-key**

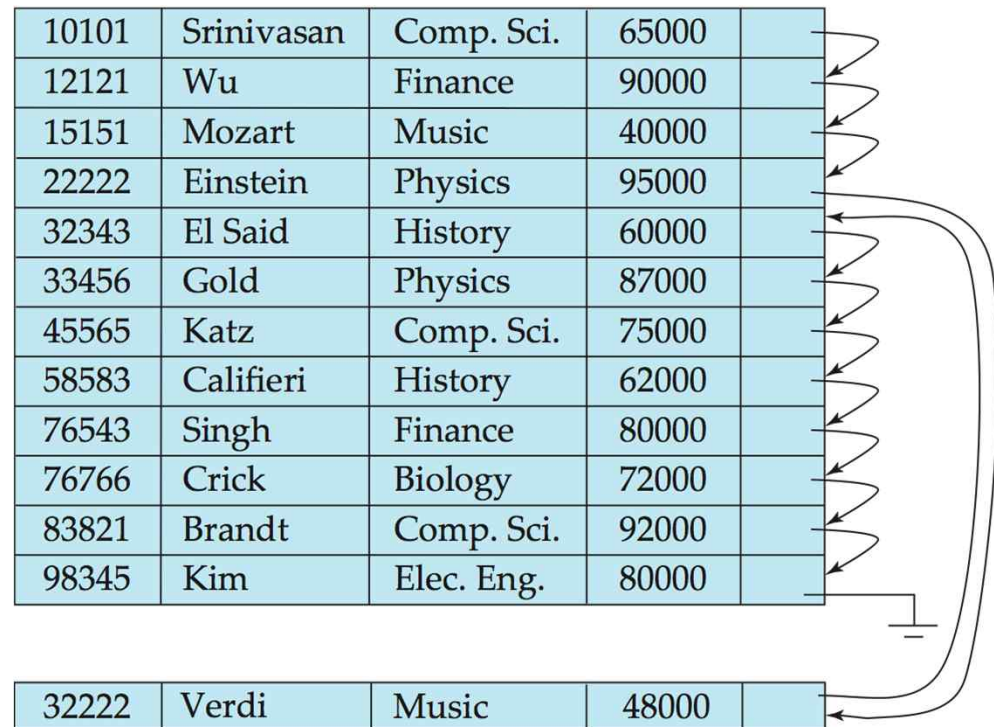
10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp.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





## Sequential File 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
- Need to reorganize the file from time to time to restore sequential order







# Data Dictionary Storage

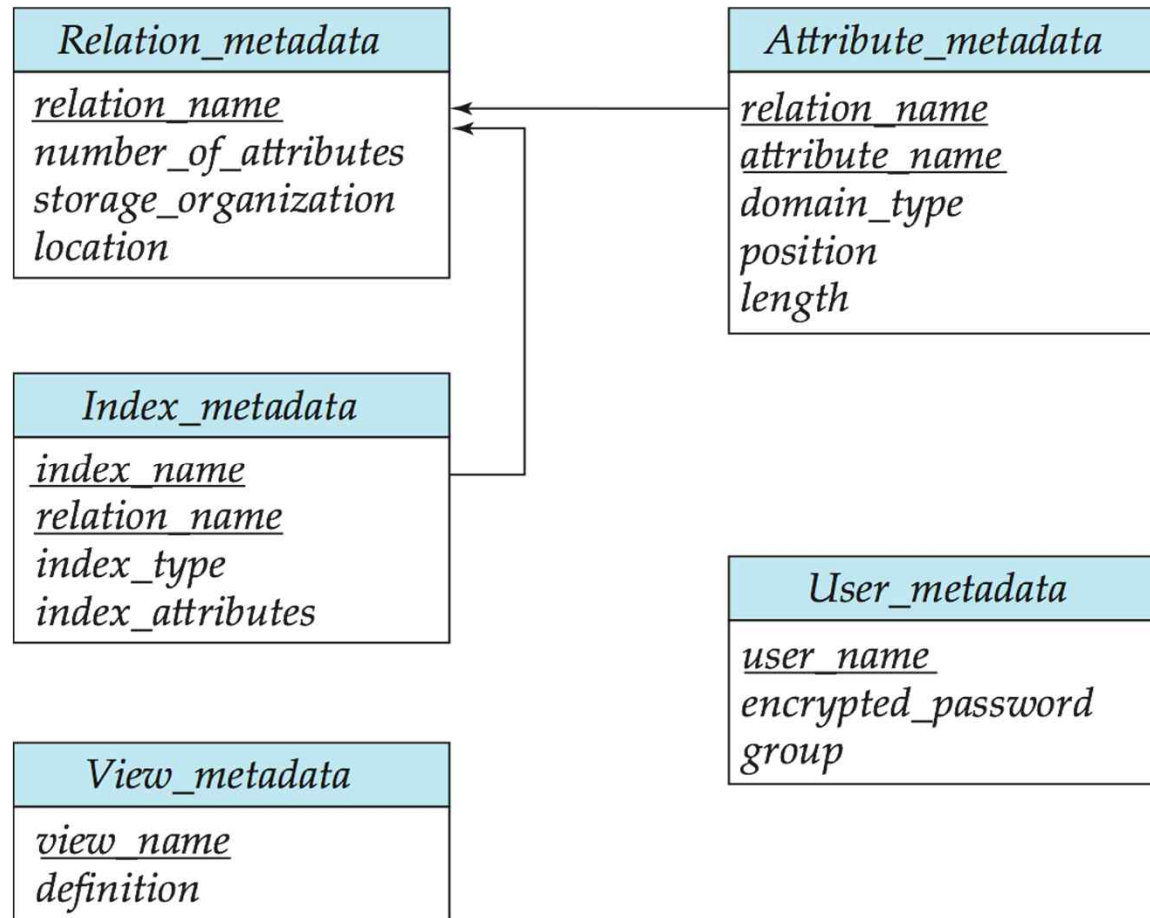
- The **Data dictionary** (also called **system catalog**) stores **metadata**; that is, data about data, such as
  - **Information about relations**
    - names of relations
    - names, types and lengths of attributes of each relation
    - names and definitions of views
    - integrity constraints
  - **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)





# Relational Representation of System Metadata

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





# Storage Access

- A database file is partitioned into fixed-length storage units called **blocks**
  - Blocks are units of both **storage allocation** and **data transfer**
- DBMS 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**



# 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

- 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 DBMS 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
  - 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 ...



# Buffer-Replacement Policies (Cont.)

- **The Problem of LRU can be solved**
  - **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



# End of Chapter 10

**Database System Concepts, 6<sup>th</sup> Ed.**

©Silberschatz, Korth and Sudarshan

See [www.db-book.com](http://www.db-book.com) for conditions on re-use