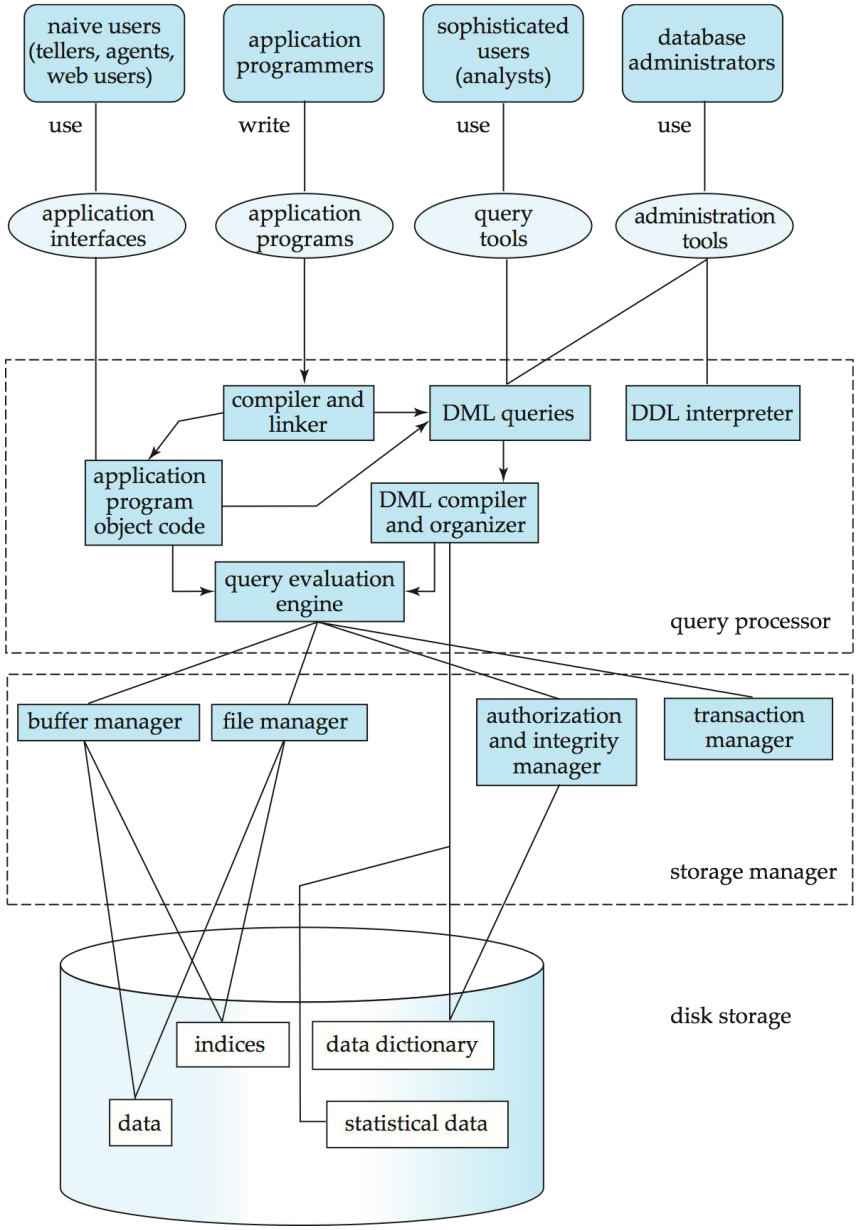
Chapter 12，13: Storage and File Structure

Database System Concepts, 7th Ed.

©Silberschatz, Korth and Sudarshan
See www.db-book.com for conditions on re-use


naive users (tellers, agents, web users)
application programmers
sophisticated users (analysts)
database administrators
use
write
use
use
application interfaces
application programs
query tools
administration tools
compiler and linker
DML queries
DDL interpreter
application program object code
DML compiler and organizer
query evaluation engine
query processor
buffer manager
file manager
authorization and integrity manager
transaction manager
storage manager
disk storage
indices
data dictionary
data
statistical data

Chapter 10: Storage and File Structure

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

©Silberschatz, Korth and Sudarshan

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.

©Silberschatz, Korth and Sudarshan

Physical Storage Media

- **Cache** – fastest and most costly form of storage; volatile; managed by the computer system hardware.
- **Main memory**:
  - fast access (10 to 100 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

©Silberschatz, Korth and Sudarshan

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 15 TB as of 2019
    - 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

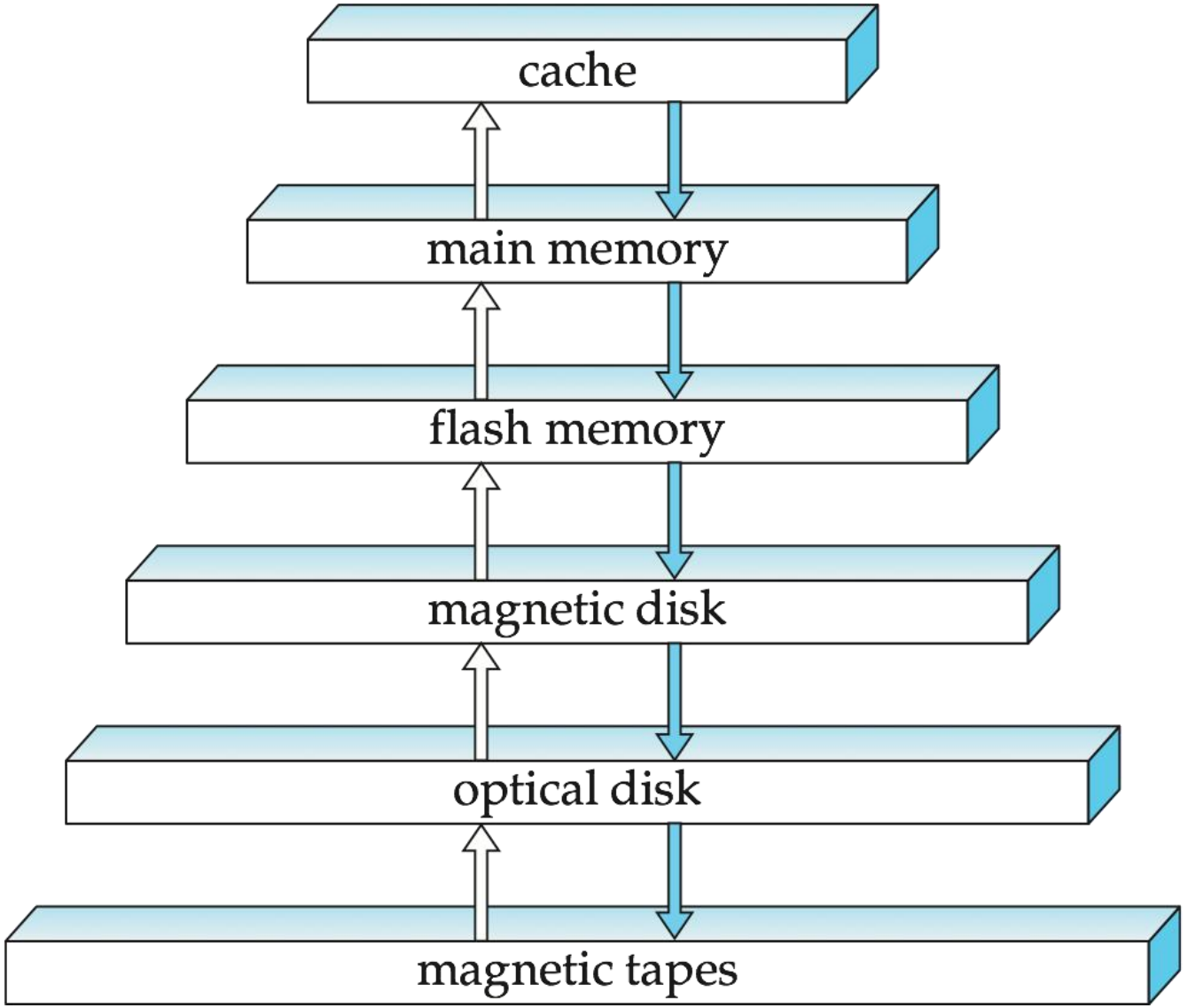
Physical Storage Media (Cont.)

- **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 (300 TB per tape 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



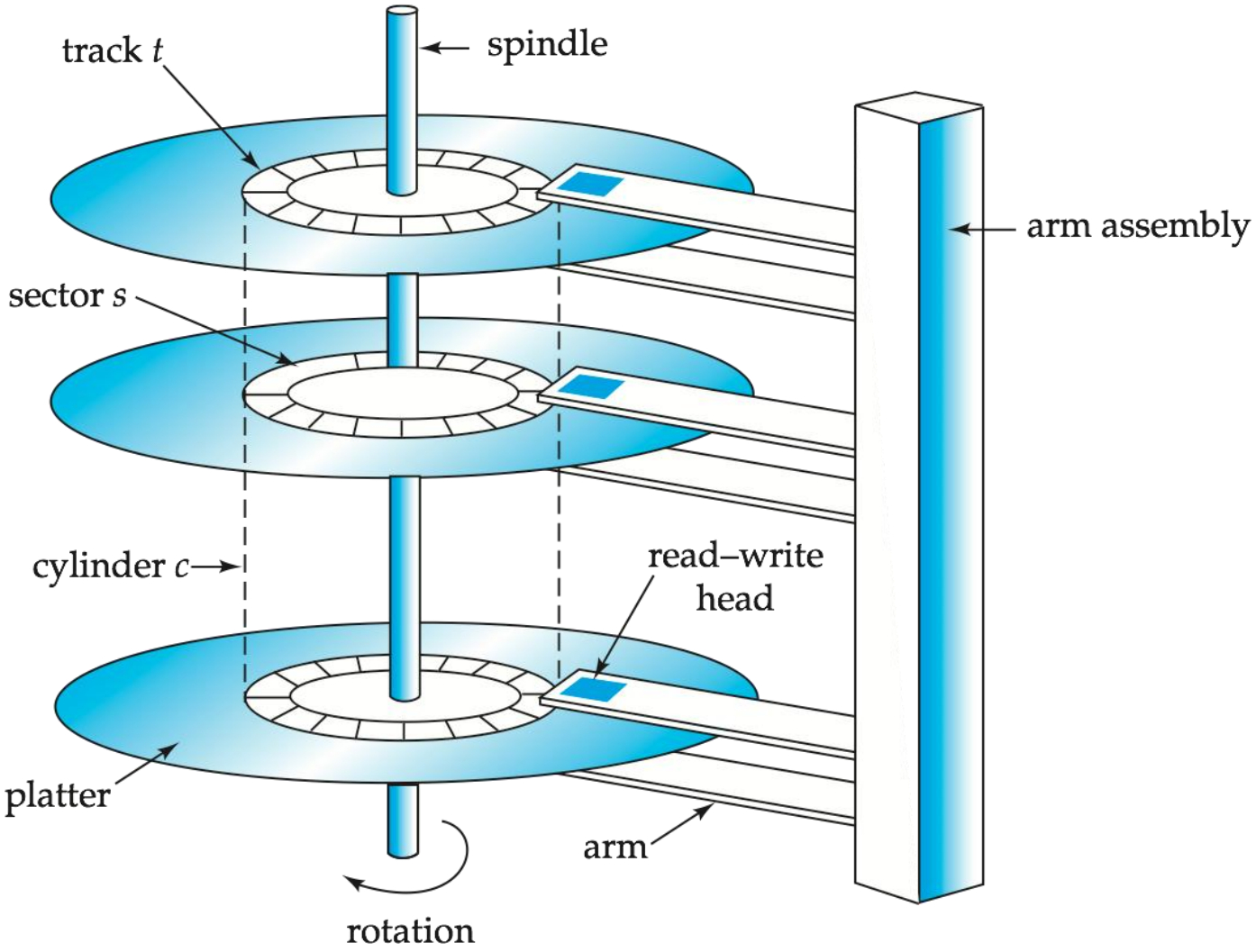
Storage Hierarchy (Cont.)

- **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

Disk Storage Interfaces

- Disk interface standards families
  - SATA (Serial ATA)
    - SATA 3 supports data transfer speeds of up to 6 gigabits/sec
  - SAS (Serial Attached SCSI)
    - SAS Version 3 supports 12 gigabits/sec
  - NVMe (Non-Volatile Memory Express) interface
    - Works with PCIe connectors to support lower latency and higher transfer rates
    - Supports data transfer rates of up to 24 gigabits/sec
- Disks usually connected directly to computer system
- In **Storage Area Networks (SAN)**, a large number of disks are connected by a high-speed network to a number of servers
- In **Network Attached Storage (NAS)** networked storage provides a file system interface using networked file system protocol, instead of providing a disk system interface

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.)

- **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

©Silberschatz, Korth and Sudarshan

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.)
- **Data-transfer rate** – the rate at which data can be retrieved from or stored to the disk.
  - 25 to 200 MB per second max rate, lower for inner tracks
  - Multiple disks may share a controller, so rate that controller can handle is also important

Performance Measures (Cont.)

- **Disk block** is a logical unit for storage allocation and retrieval
  - 4 to 16 kilobytes typically
    - Smaller blocks: more transfers from disk
    - Larger blocks: more space wasted due to partially filled blocks
- **Sequential access pattern**
  - Successive requests are for successive disk blocks
  - Disk seek required only for first block
- **Random access pattern**
  - Successive requests are for blocks that can be anywhere on disk
  - Each access requires a seek
  - Transfer rates are low since a lot of time is wasted in seeks
- **I/O operations per second (IOPS)**
  - Number of random block reads that a disk can support per second
  - 50 to 200 IOPS on current generation magnetic disks

©Silberschatz, Korth and Sudarshan

Performance Measures (Cont.)

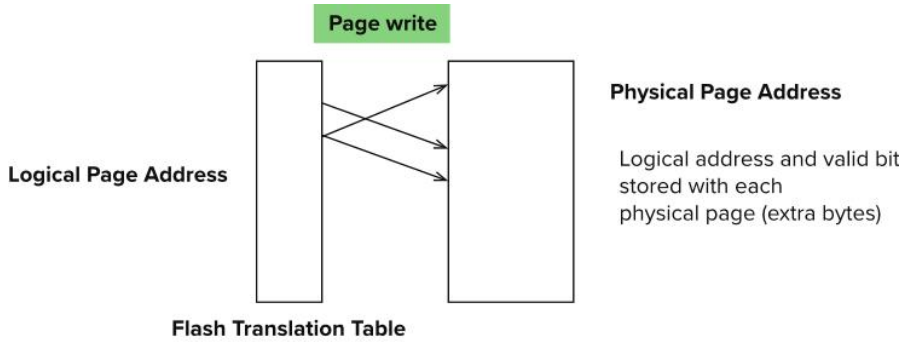
- **Mean time to failure (MTTF)** – the average time the disk is expected to run continuously without any failure.
  - Typically 3 to 5 years
  - Probability of failure of new disks is quite low, corresponding to a "theoretical MTTF" of 500,000 to 1,200,000 hours for a new disk
    - E.g., an MTTF of 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

Flash Storage

- NAND flash
  - used widely for storage, cheaper than NOR flash
  - requires page-at-a-time read (page: 512 bytes to 4 KB)
    - 20 to 100 microseconds for a page read
    - Not much difference between sequential and random read
  - Page can only be written once
    - Must be erased to allow rewrite
- **Solid state disks**
  - Use standard block-oriented disk interfaces, but store data on multiple flash storage devices internally
  - Transfer rate of up to 500 MB/sec using SATA, and up to 3 GB/sec using NVMe PCIe

Flash Storage (Cont.)

- Erase happens in units of **erase block**
  - Takes 2 to 5 millisecs
  - Erase block typically 256 KB to 1 MB (128 to 256 pages)
- **Remapping** of logical page addresses to physical page addresses avoids waiting for erase
- **Flash translation table** tracks mapping
  - also stored in a label field of flash page
  - remapping carried out by **flash translation layer**



- After 100,000 to 1,000,000 erases, erase block becomes unreliable and cannot be used
  - **wear leveling**

SSD Performance Metrics

- Random reads/writes per second
  - Typical 4 KB reads: 10,000 reads per second (10,000 IOPS)
  - Typical 4KB writes: 40,000 IOPS
  - SSDs support parallel reads
    - Typical 4KB reads:
      - 100,000 IOPS with 32 requests in parallel (QD-32) on SATA
      - 350,000 IOPS with QD-32 on NVMe PCIe
    - Typical 4KB writes:
      - 100,000 IOPS with QD-32, even higher on some models
- Data transfer rate for sequential reads/writes
  - 400 MB/sec for SATA3, 2 to 3 GB/sec using NVMe PCIe
- **Hybrid disks**: combine small amount of flash cache with larger magnetic disk

©Silberschatz, Korth and Sudarshan

Storage Class Memory

- 3D-XPoint memory technology pioneered by Intel
- Available as Intel Optane
  - SSD interface shipped from 2017
    - Allows lower latency than flash SSDs
  - Non-volatile memory interface announced in 2018
    - Supports direct access to words, at speeds comparable to main-memory speeds

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
- 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)
  - Techniques for using redundancy to avoid data loss are critical with large numbers of disks

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. Logical 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 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)

©Silberschatz, Korth and Sudarshan

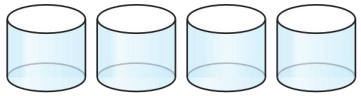
Improvement in Performance via Parallelism

- 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

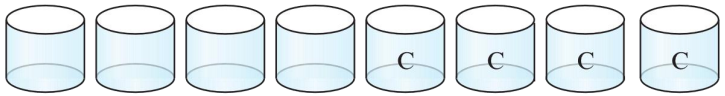
©Silberschatz, Korth and Sudarshan

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.
- **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

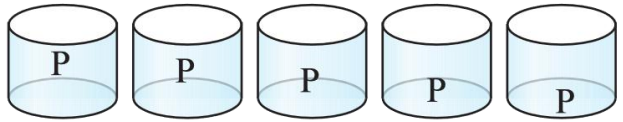
RAID Levels (Cont.)

- **Parity blocks**: Parity block *j* stores XOR of bits from block *j* of each disk
  - When writing data to a block *j*, parity block *j* must also be computed and written to 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 recover data for a block, compute XOR of bits from all other blocks in the set including the parity block

©Silberschatz, Korth and Sudarshan

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.
  - E.g., with 5 disks, parity block for *n*th set of blocks is stored on disk (*n mod* 5) + 1, with the data blocks stored on the other 4 disks.



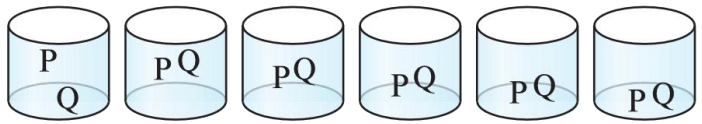
(c) RAID 5: block-interleaved distributed parity

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

©Silberschatz, Korth and Sudarshan

RAID Levels (Cont.)

- **RAID Level 5** (Cont.)
  - Block writes occur in parallel if the blocks and their parity blocks are on different disks.
- **RAID Level 6**: P+Q Redundancy scheme; similar to Level 5, but stores two error correction blocks (P, Q) instead of single parity block to guard against multiple disk failures.
  - Better reliability than Level 5 at a higher cost
    - Becoming more important as storage sizes increase



(d) RAID 6: P + Q redundancy

RAID Levels (Cont.)

- **Other levels (not used in practice):**
  - **RAID Level 2**: Memory-Style Error-Correcting-Codes (ECC) with bit striping.
  - **RAID Level 3**: Bit-Interleaved Parity
  - **RAID Level 4:** Block-Interleaved Parity; uses block-level striping, and keeps a parity block on a separate ***parity disk*** for corresponding blocks from *N* other disks.
    - RAID 5 is better than RAID 4, since with RAID 4 with random writes, parity disk gets much higher write load than other disks and becomes a bottleneck

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

Choice of RAID Level (Cont.)

- Level 1 provides much better write performance than level 5
  - Level 5 requires at least 2 block reads and 2 block writes to write a single block, whereas Level 1 only requires 2 block writes
- Level 1 had higher storage cost than level 5
- Level 5 is preferred for applications where writes are sequential and large (many blocks), and need large amounts of data storage
- RAID 1 is preferred for applications with many random/small updates
- Level 6 gives better data protection than RAID 5 since it can tolerate two disk (or disk block) failures
  - Increasing in importance since latent block failures on one disk, coupled with a failure of another disk can result in data loss with RAID 1 and RAID 5.

©Silberschatz, Korth and Sudarshan

Hardware Issues

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

©Silberschatz, Korth and Sudarshan

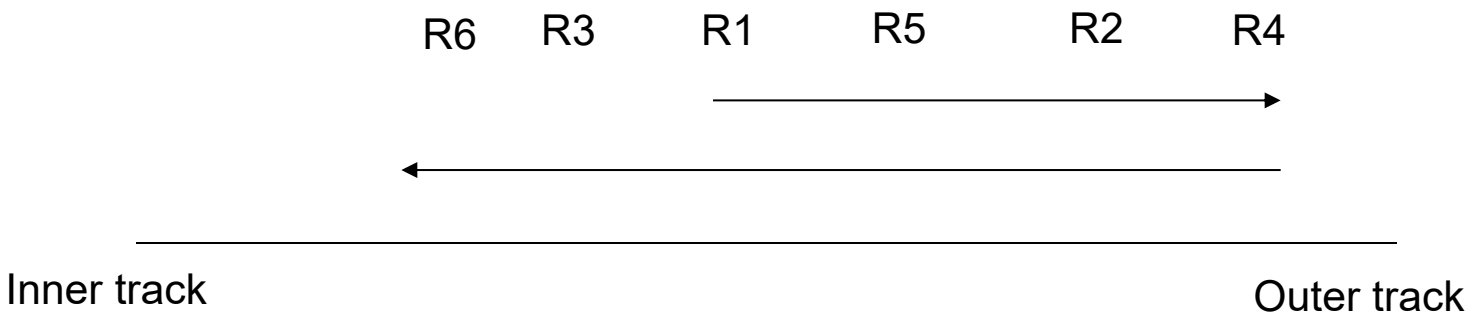
Hardware Issues (Cont.)

- **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

©Silberschatz, Korth and Sudarshan

Optimization of Disk-Block Access

- **Buffering:** in-memory buffer to cache disk blocks
- **Read-ahead:** Read extra blocks from a track in anticipation that they will be requested soon
- **Disk-arm-scheduling** algorithms re-order block requests so that disk arm movement is minimized
  - **elevator algorithm**



Optimization of Disk Block Access (Cont.)

- **File organization**
  - Allocate blocks of a file in as contiguous a manner as possible
  - Allocation in units of **extents**
  - Files may get **fragmented**
    - E.g., if 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
- **Non-volatile write buffers**

File Organization, Record Organization and Storage Access

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.
- One approach:
  - assume record size is fixed
  - each file has records of one particular type only
  - different files are used for different relations

  This case is easiest to implement; will consider variable length records later.
- We assume that records are smaller than a disk block

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

- 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

©Silberschatz, Korth and Sudarshan

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

©Silberschatz, Korth and Sudarshan

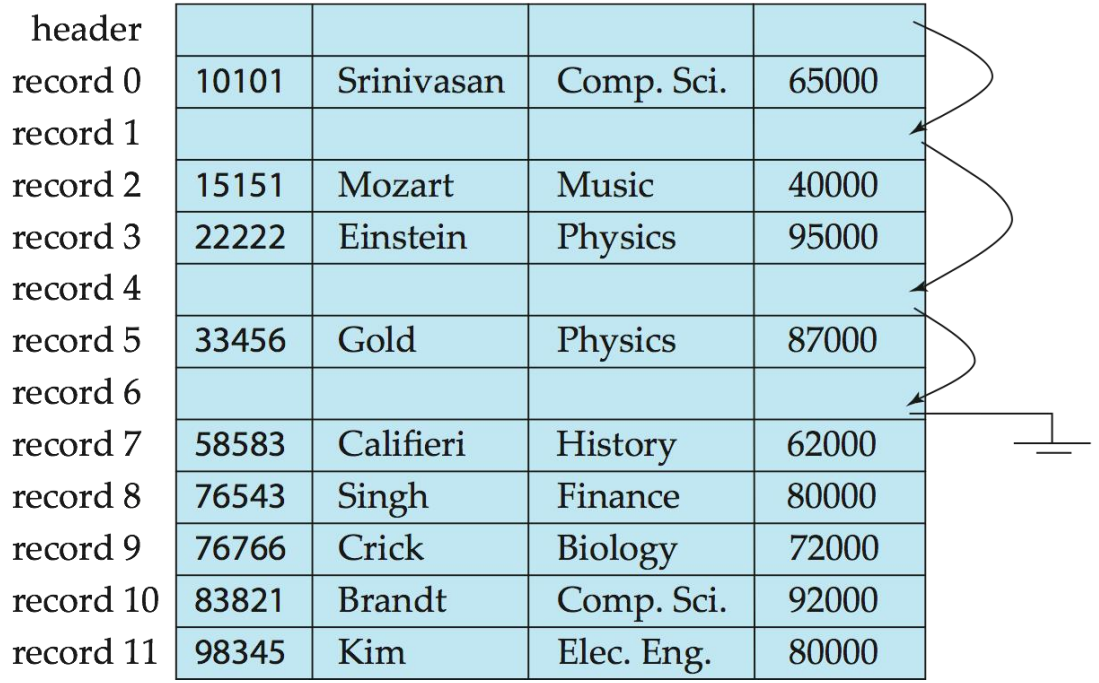
Deleting record 3 and moving last record

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

©Silberschatz, Korth and Sudarshan

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.)

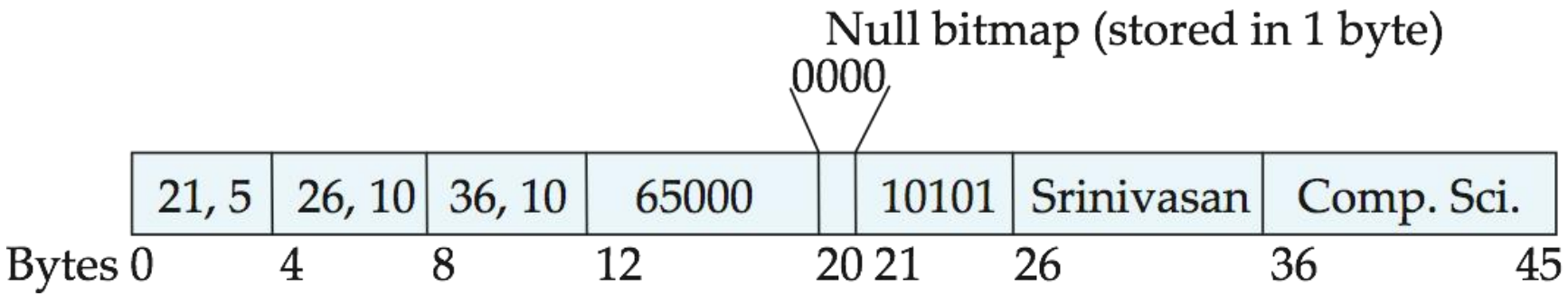


header				
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1				
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4				
record 5	33456	Gold	Physics	87000
record 6				
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology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

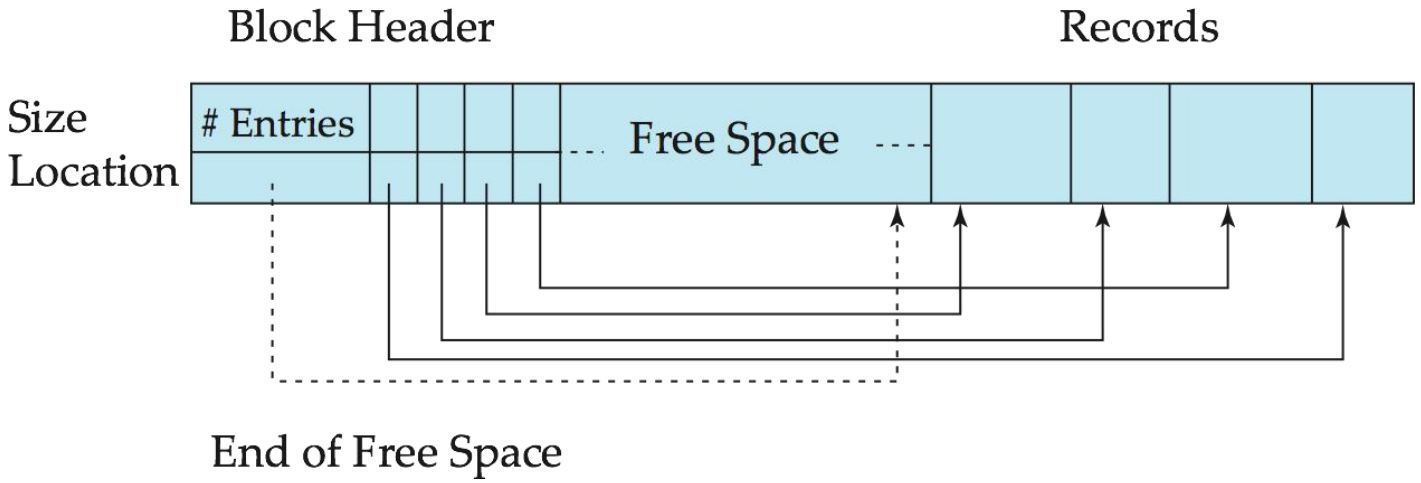
©Silberschatz, Korth and Sudarshan

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-value bitmap



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.

Storing Large Objects

- E.g., blob/clob types
- Records must be smaller than pages
- Alternatives:
  - Store as files in file systems
  - Store as files managed by database
  - Break into pieces and store in multiple tuples in separate relation
    - PostgreSQL TOAST

©Silberschatz, Korth and Sudarshan

Organization of Records in Files

- **Heap** – a record can be placed anywhere in the file where there is space
- **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. In a **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
- **B^+-tree file organization**
  - Ordered storage

Heap File Organization

- Records can be placed anywhere in the file where there is free space
- Records usually do not move once allocated
- Important to be able to efficiently find free space within file
- **Free-space map**
  - Array with 1 entry per block. Each entry is a few bits to a byte, and records fraction of block that is free
  - In example below, 3 bits per block, value divided by 8 indicates fraction of block that is free

4	2	1	4	7	3	6	5	1	2	0	1	1	0	5	6

  - Can have second-level free-space map
  - In example below, each entry stores maximum from 4 entries of first-level free-space

4	7	2	6

- Free space map written to disk periodically, OK to have wrong (old) values for some entries (will be detected and fixed)

©Silberschatz, Korth and Sudarshan

Sequential File Organization

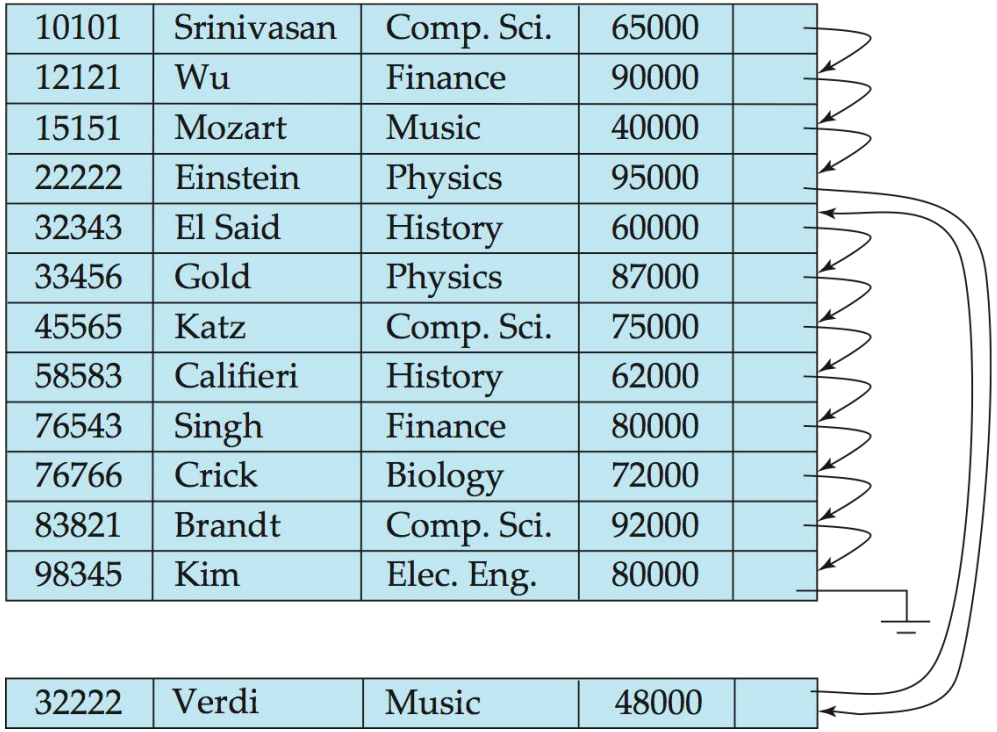
- Suitable for applications that require sequential processing of the entire file
- The records in the file are ordered by a search-key

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	

©Silberschatz, Korth and Sudarshan

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



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	

Multitable Clustering File Organization

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

©Silberschatz, Korth and Sudarshan

Multitable Clustering File Organization (cont.)

- 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
- 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	

Partitioning

- **Table partitioning**: Records in a relation can be partitioned into smaller relations that are stored separately
- E.g., *transaction* relation may be partitioned into *transaction_2018, transaction_2019, etc.*
- Queries written on *transaction* must access records in all partitions
  - Unless query has a selection such as *year*=2019, in which case only one partition in needed
- Partitioning
  - Reduces costs of some operations such as free space management
  - Allows different partitions to be stored on different storage devices
    - E.g., *transaction* partition for current year on SSD, for older years on magnetic disk

©Silberschatz, Korth and Sudarshan

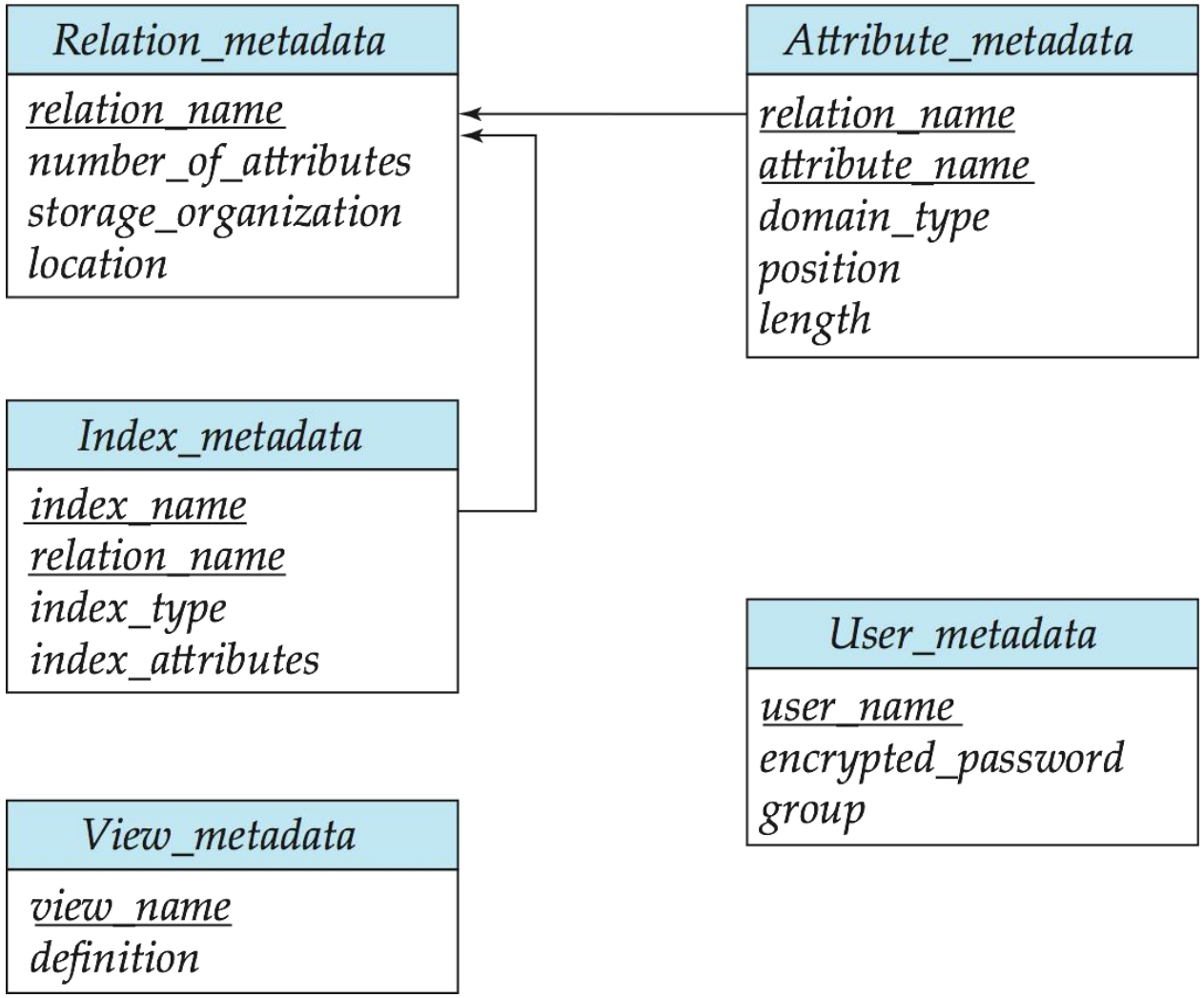
Data Dictionary Storage

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
- Information about indices

Relational Representation of System Metadata

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



©Silberschatz, Korth and Sudarshan

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.

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.

©Silberschatz, Korth and Sudarshan

Buffer-Replacement Policies

- Most operating systems replace the block **least recently used** (LRU strategy)
- Idea behind LRU – use past pattern of block references as a predictor of future references
- Example, Four pages, Three buffers:

1	3	4	3	1	2
		4	3	1	2
	3	3	4	3	1
1	1	1	1	4	3

©Silberschatz, Korth and Sudarshan

Buffer-Replacement Policies (Cont.)

- 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

Buffer-Replacement Policies (Cont.)

- **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

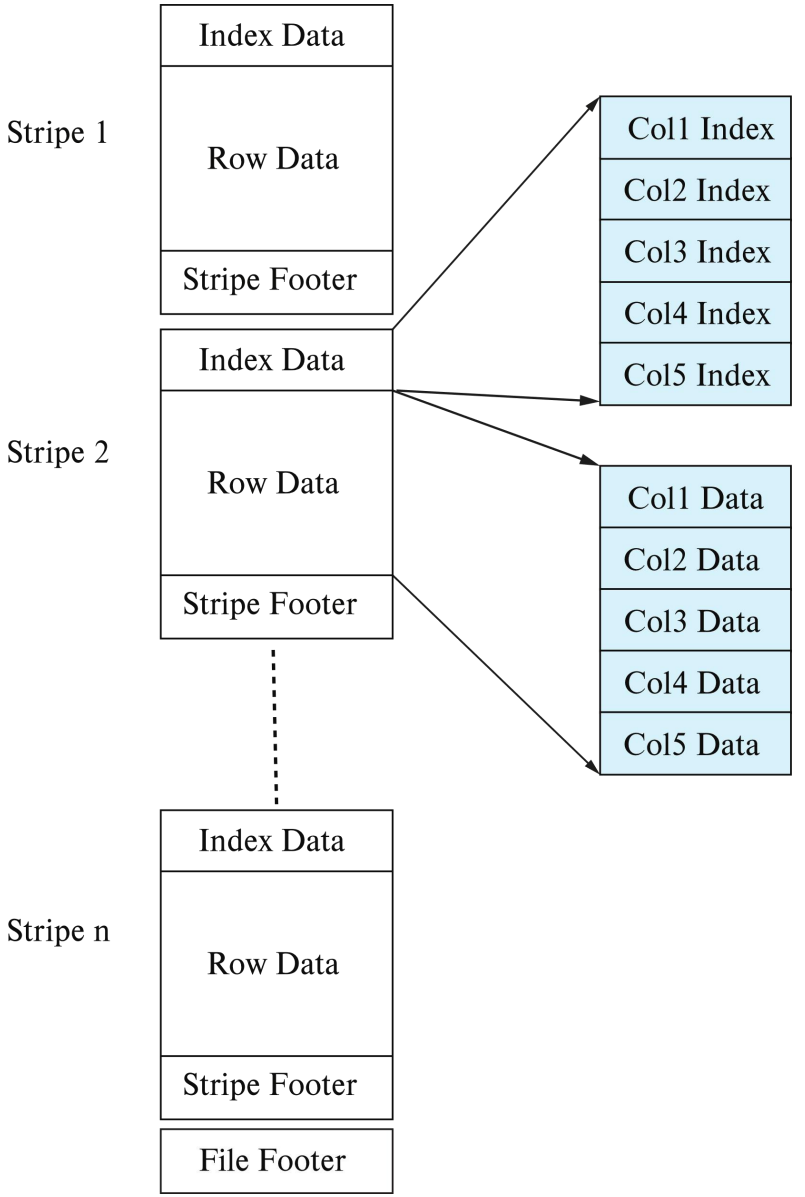
Columnar Representation

- Benefits:
  - Reduced IO if only some attributes are accessed
  - Improved CPU cache performance
  - Improved compression
  - **Vector processing** on modern CPU architectures
- Drawbacks
  - Cost of tuple reconstruction from columnar representation
  - Cost of tuple deletion and update
  - Cost of decompression
- Columnar representation found to be more efficient for decision support than row-oriented representation
- Traditional row-oriented representation preferable for transaction processing
- Some databases support both representations
  - Called **hybrid row/column stores**

©Silberschatz, Korth and Sudarshan

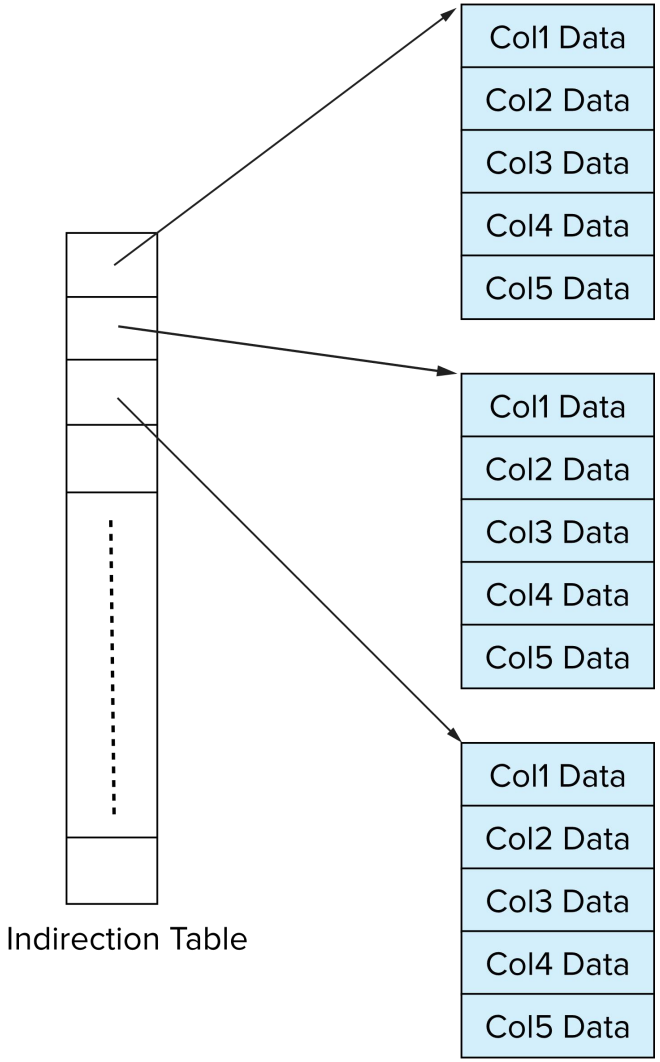
Columnar File Representation

- ORC and Parquet: file formats with columnar storage inside file
- Very popular for big-data applications
- Orc file format shown on right:



Storage Organization in Main-Memory Databases

- Can store records directly in memory without a buffer manager
- Column-oriented storage can be used in-memory for decision support applications
  - Compression reduces memory requirement



End of Chapter 12, 13

Execise:12.1, 13.5, 13.9, 13.11

©Silberschatz, Korth and Sudarshan
See www.db-book.com for conditions on re-use