Principles of Database Systems (CS307)

Lecture 12: Storage System and Structure

Zhong-Qiu Wang

Department of Computer Science and Engineering
Southern University of Science and Technology

- Most contents are from slides made by Stéphane Faroult and the authors of Database System Concepts (7th Edition).
- Their original slides have been modified to adapt to the schedule of CS307 at SUSTech.
- The slides are largely based on the slides provided by Dr. Yuxin Ma

Outline

Part 1: Relational Language

Basic SQL
- Create Tables and Insert Data
- Basic Queries

Intermediate SQL
- Advanced Join
- Subquery
- Window Function
- *Transactions*

Advanced SQL
- Function
- Procedure
- Trigger
- View

Lab Only

Part 2: DB Design and Application

Relational DB Design
- The Entity-Relationship Model
- E-R Diagram Design

Relational Algebra

Normalization
- Functional Dependency
- Normal Forms
- FD Theory
- Decomposition w/ FD

Application Design
- Use Databases in Programs
- Complex Data Types (JSON, XML, etc.)

Part 3: Internal Mechanism Of Relational DB Systems

Storage Management
- Physical and Logical Storage Structure
- Indexing

Query Management
- Query Processing
- Query Optimization

Transaction Management
- Transaction
- Concurrency

Part 4: Miscellaneous Topics

Big Data
- Big Data Storage
- Query and Process Big Data

Advanced Topics
- Beyond Relational DB
- Data Analytics

Physical Storage System

Physical Storage System

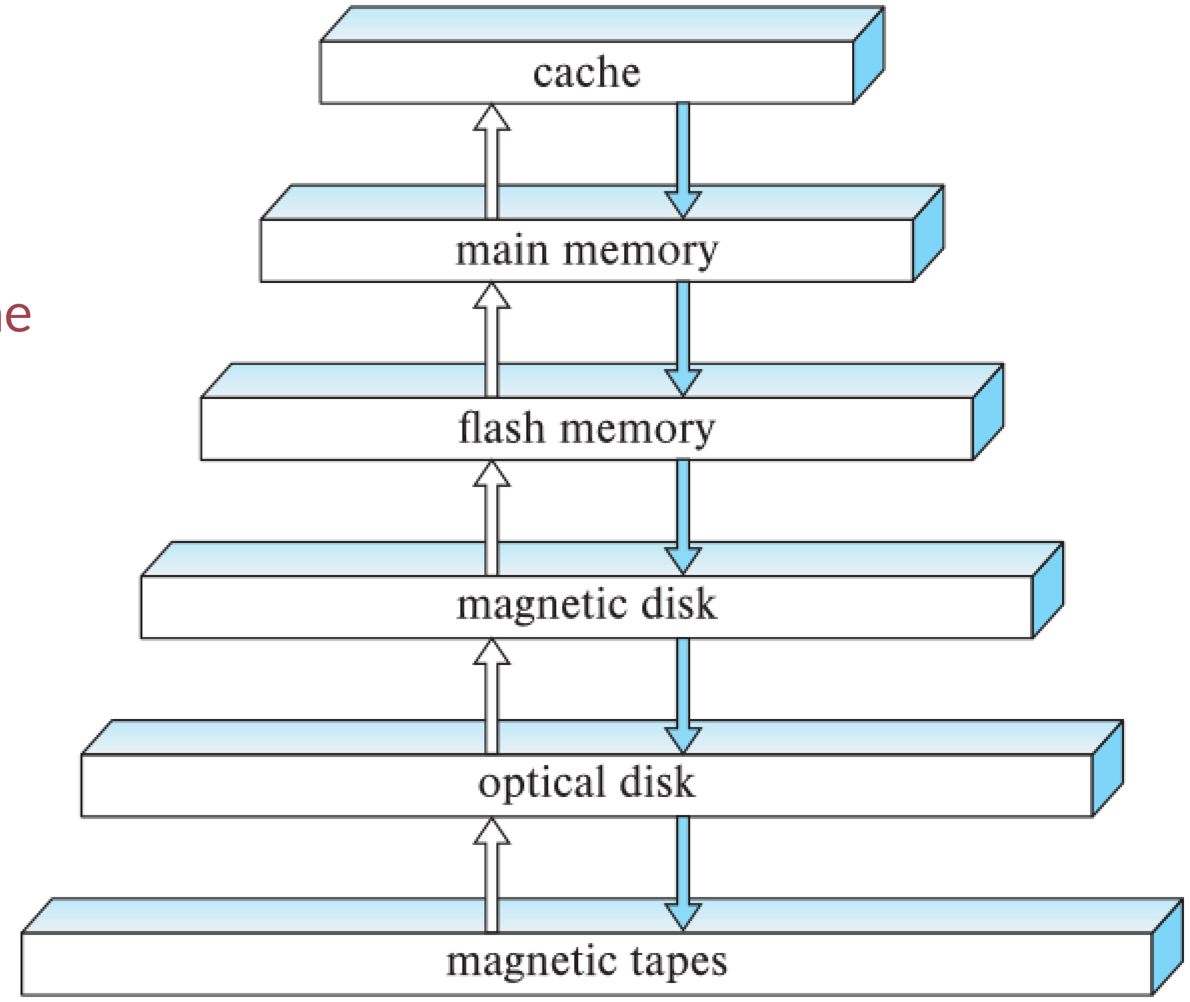
- The hardware where data is recorded

Classification of Physical Storage Media

- 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
- Factors affecting choice of storage media include
  - Speed with which data can be accessed
  - Cost per unit of data
  - Reliability

Storage Hierarchy

- 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 and used for archival storage
  - E.g., magnetic tape, optical storage (DVD, blueray DVD)
    - Magnetic tape
      - Sequential access, 1 to 12 TB capacity
      - A few drives with many tapes
      - Juke boxes with petabytes (1000's of TB) of storage

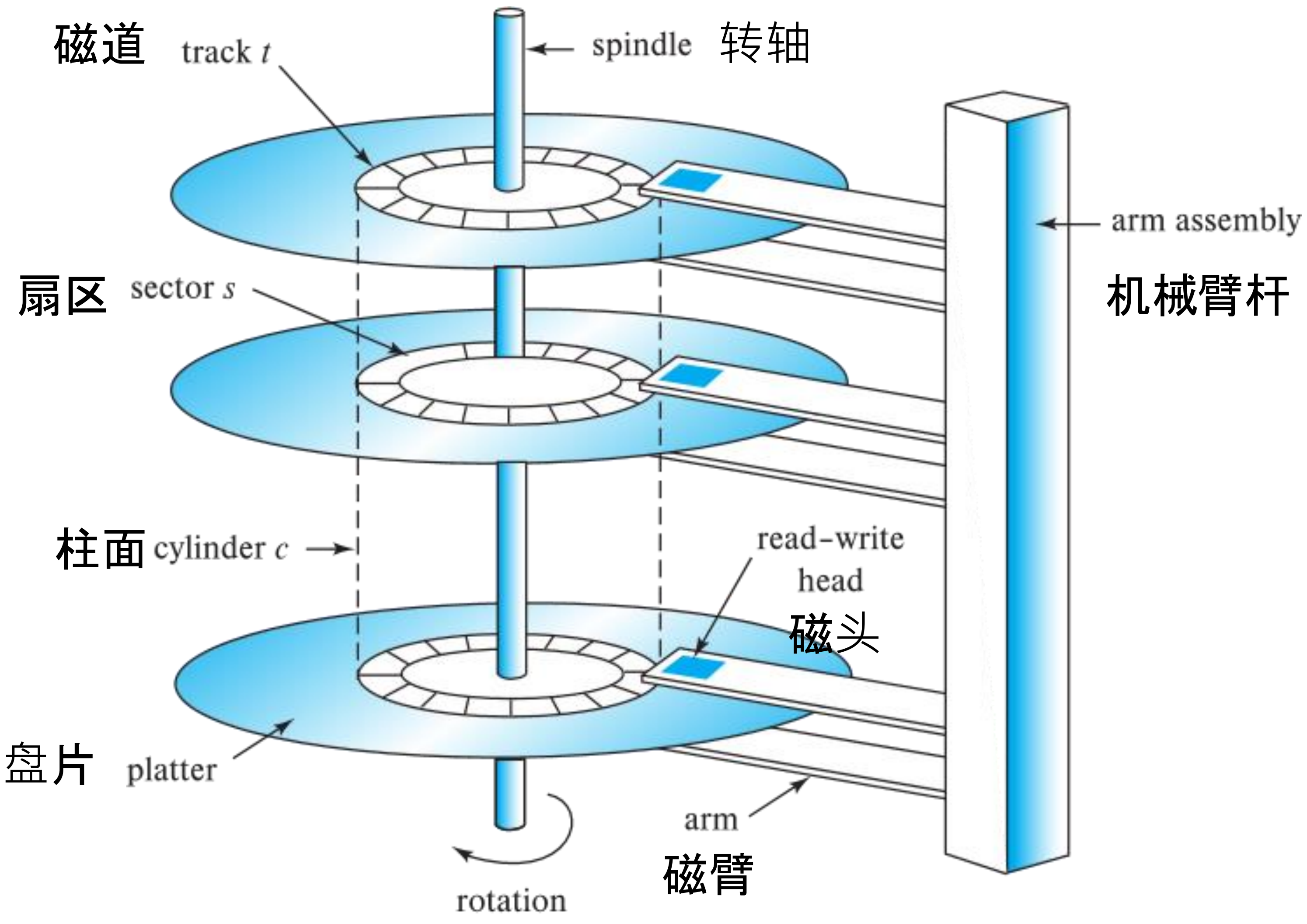


Storage Interfaces

- 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, however…
  - 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



Schematic diagram of magnetic disk drive

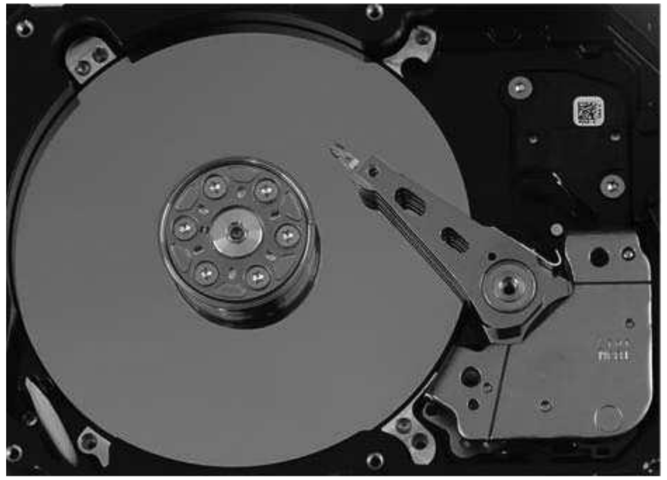
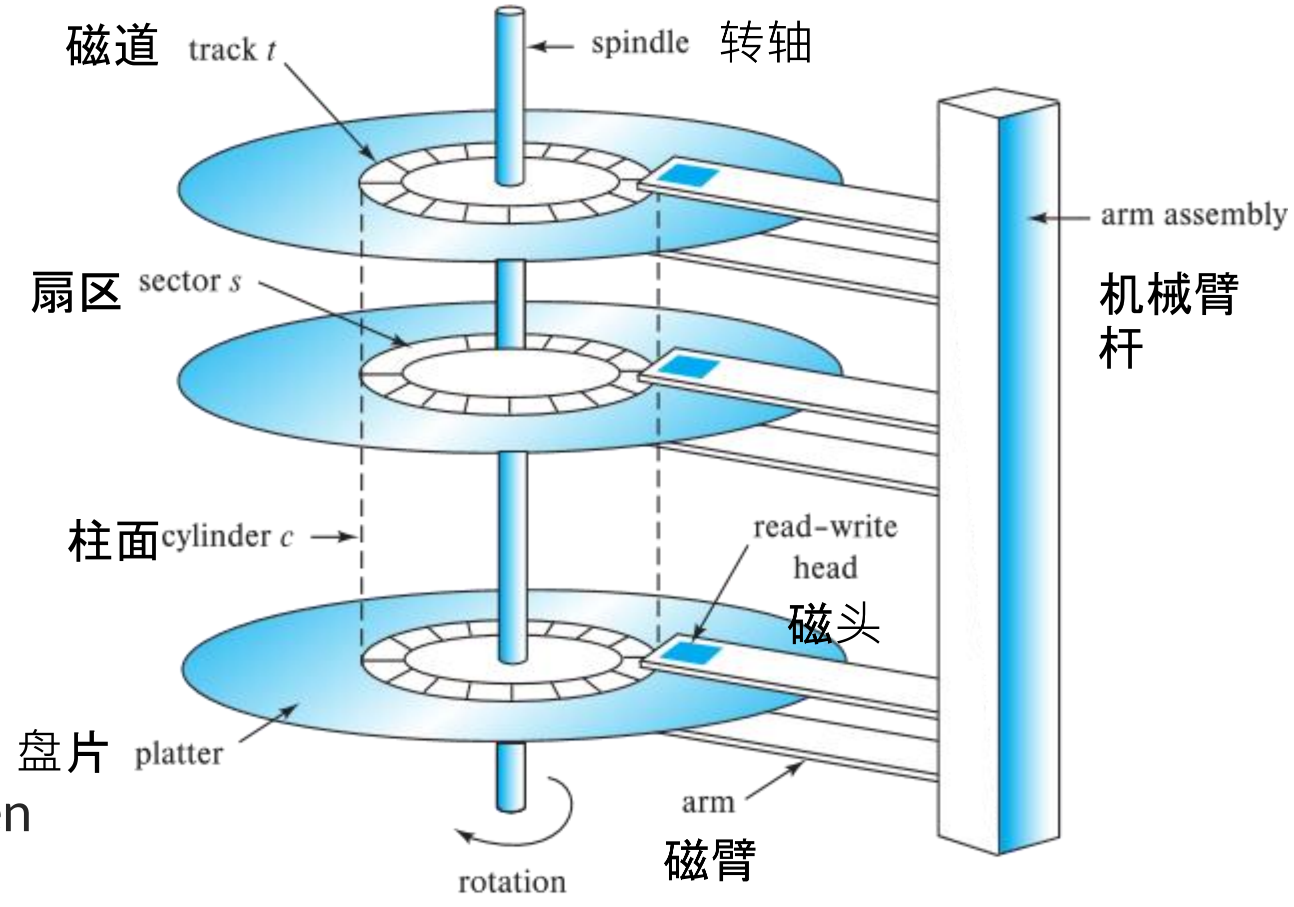


Photo of a magnetic disk drive

Magnetic Hard Disk Mechanism



- Read-write head
- 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 (modern OS requires 4KB)
  - 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 ith track of all the platters

Magnetic Hard Disk Mechanism

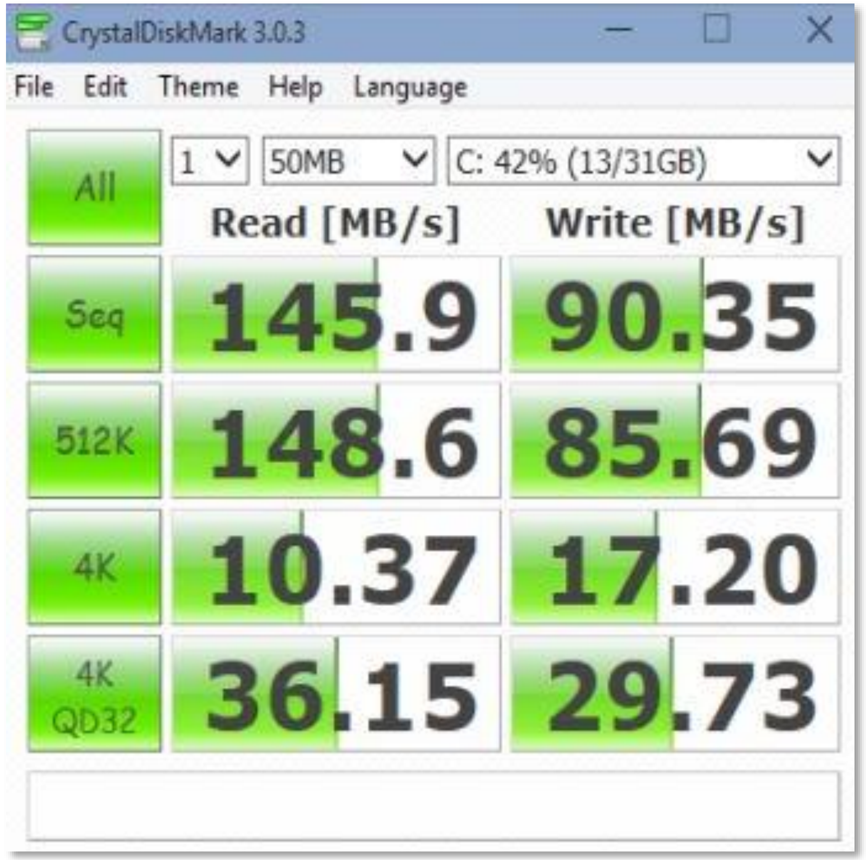
- Disk controller: An interface between the computer system and the disk drive hardware
  - Accept high-level commands to read or write a sector
  - Initiate actions such as moving the disk arm to the right track and reading or writing the data
  - Compute and attach checksums（校验和）to each sector during writing
    - It can verify the data that is later read back is correct
    - If data is corrupted, with very high probability stored checksum won't match recomputed checksum
  - Ensure successful writing by reading back sector after writing it
  - Perform remapping of bad sectors
    - Map the address of bad sectors to that of good sectors

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
    - 4 to 11 milliseconds on typical disks (5400 to 15000 r.p.m.)
    - Average latency is 1/2 of the above latency
  - Overall latency is 5 to 20 msec depending on disk model
- 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 (which have fewer sectors)

Performance Measures of Disks

- Each request specifies the address on the disk to be referenced, and that address is in the form of a block number
- 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



Performance Measures of Disks

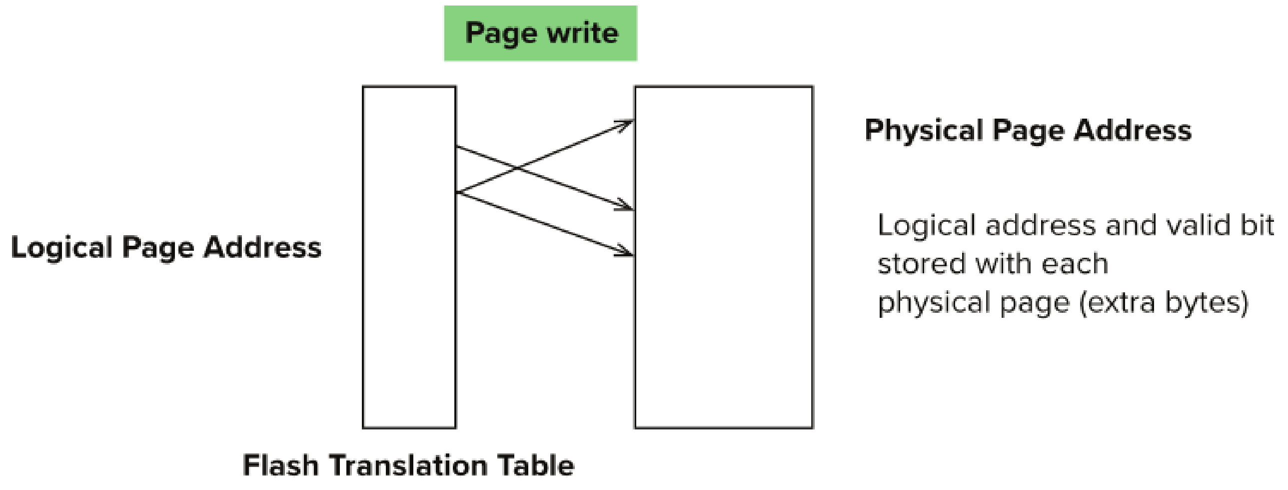
- Mean time to failure (MTTF) – the average time the disk is expected to run continuously without any failure (i.e., reliability)
  - Typically, 3 to 5 years
  - Probability of failure of new disks is quite low, corresponding to a “theoretical MTTF” of 500,000 to 1,200,000 hours for a new disk
    - 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

- NOR flash vs NAND flash
- 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 (SSD)
  - 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

- 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 (balancing the number of times each block is erased)
  - Can use remapping to store data in good blocks

SSD Performance Metrics

- Random reads/writes per second
  - Typical 4KB 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

Magnetic Tapes

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

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

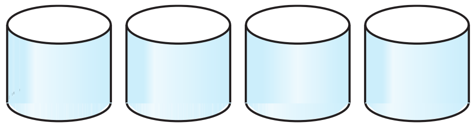
- Redundancy – Store extra information that can be used to rebuild information lost in a disk failure
  - E.g., Mirroring (or shadowing)
    - Duplicate every disk. Logical disk consists of two physical disks.
    - Every write is carried out on both disks
      - Reads can take place from either disk
    - 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*106 hours (or 57,000 years) for a mirrored pair of disks (ignoring dependent failure modes)

RAID Levels

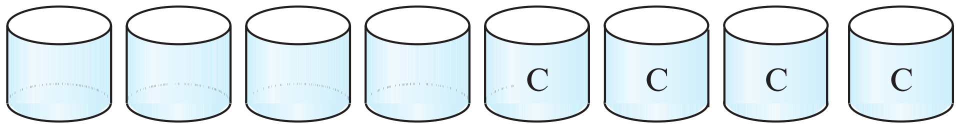
- Schemes to provide redundancy at lower cost by using disk striping combined with parity bits
  - Different RAID organizations, or RAID levels, have differing cost, performance and reliability characteristics

  - RAID 0: Block striping (e.g., bit-level striping); non-redundant
  - RAID 1: Mirrored disks with block striping
  - RAID 10: Combination of striping and mirroring

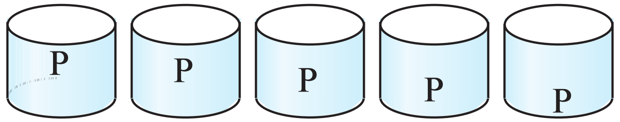
  - RAID 5: Block-interleaved distributed parity
  - RAID 6: P+Q Redundancy scheme



(a) RAID 0: nonredundant striping



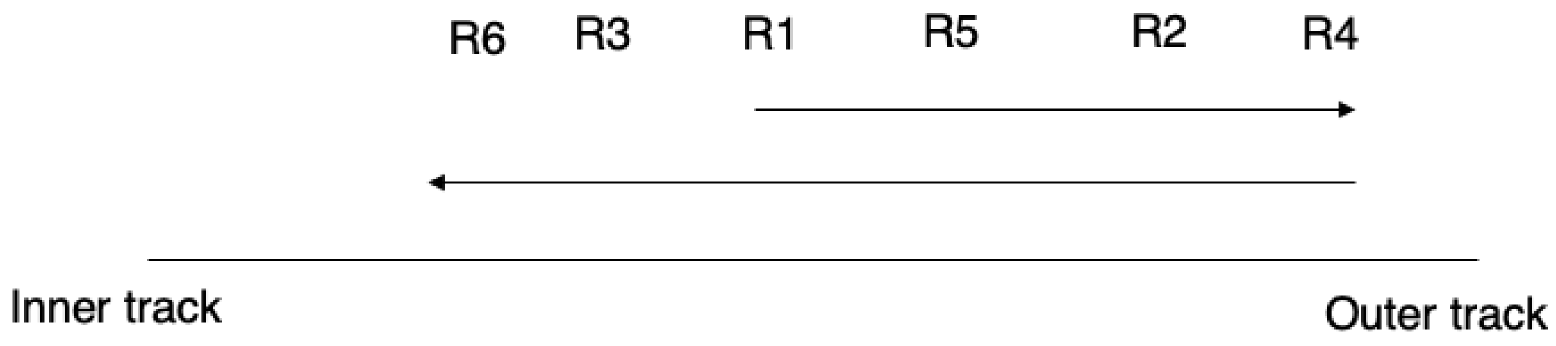
(b) RAID 1: mirrored disks



(c) RAID 5: block-interleaved distributed parity

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
  - Useful for sequential access
- Disk-arm-scheduling algorithms
  - Re-order block requests so that disk arm movement is minimized
  - E.g., elevator algorithm (like elevators do)



Optimization of Disk-Block Access

- 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
  - Temporarily store the written data
    - … and immediately notifies the OS that writing is completed without errors
  - Write data into the disk when idle
    - … with some optimizations

(Logical) Data Storage Structure

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; we will consider variable length records later
- Magnetic disks and SSDs are block structured devices
  - Data are read or written in units of a block (4-8 KB)
- Databases deal with records, which are usually much smaller than a block
- We assume that records are smaller than a disk block
  - Each record is entirely contained in a single block

File Organization

- Goals: Time and Space
  - Support CURD operations as fast as possible
  - Save storage space as much as possible
  - Also, to some extent, maintain data integrity

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 if block size is not a multiple of n bytes
    - Modification: do not allow records to cross block boundaries
    - Divide block size (e.g., 4 KB) by record size (e.g., 53 bytes), and discard the fractional part

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

Fixed-Length Records

- Deletion of record i
  - Way #1: move records $i + 1, \ldots, n$ to $i, \ldots, n - 1$

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

Fixed-Length Records

- Deletion of record i
  - Way #2: move record n to i
    - Record 3 is removed and replaced by record 11

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

record 11	98345	Kim	Elec. Eng.	80000

Fixed-Length Records

- Deletion of record i
  - Way #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

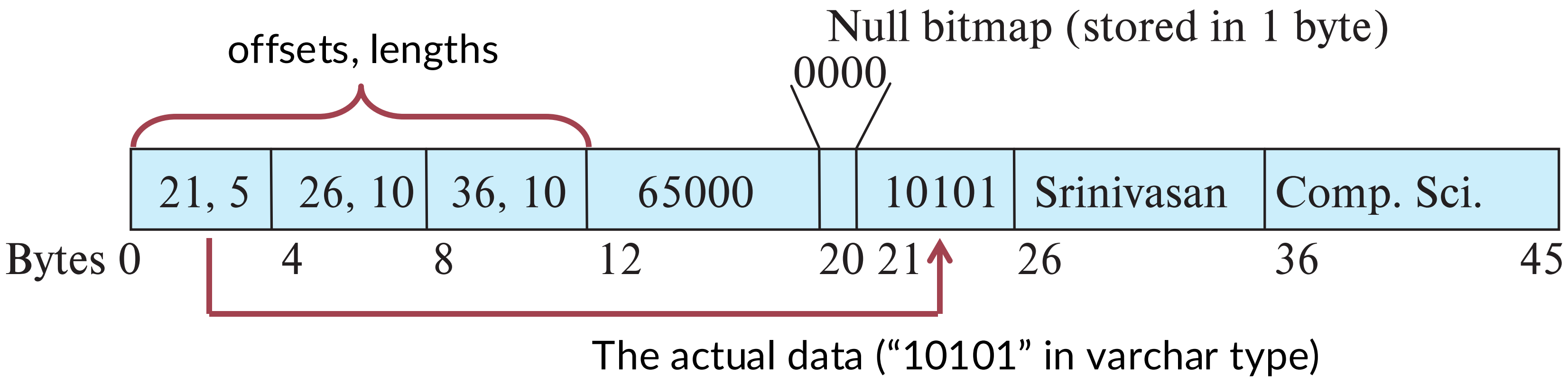
Variable-Length Records

- Variable-length records arise in database systems in several ways:
  - Record types that allow variable lengths for one or more fields such as strings (`varchar`)
  - Storage of multiple record types in a file
  - Record types that allow repeating fields (arrays, multisets used in some older data models).

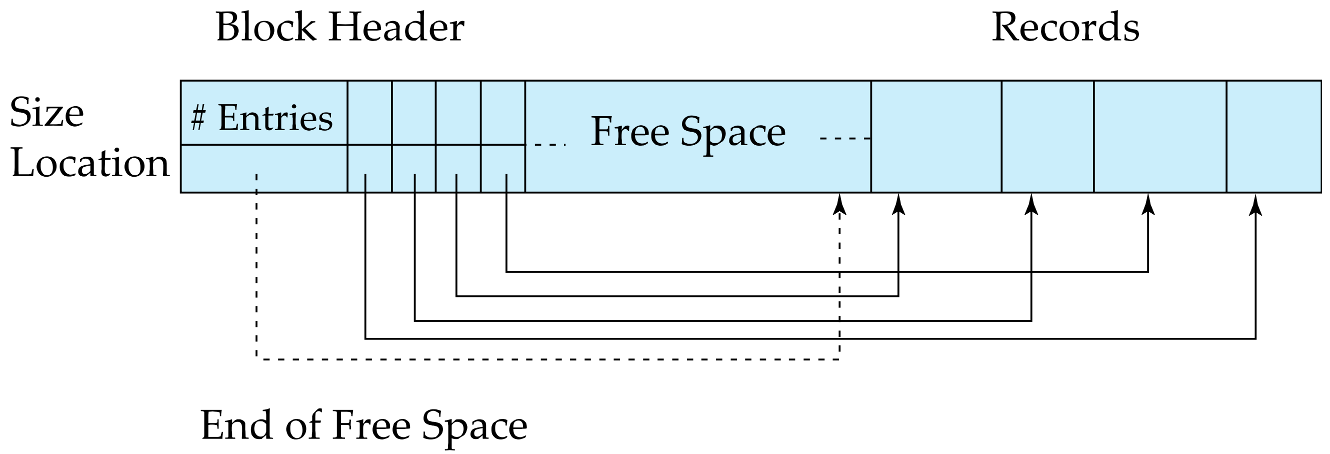
- Problem with variable-length records
  - How can we retrieve the data in an easy way without wasting too much space
    - `varchar(1000)`
      - Do we really need to allocate 1000 bytes for this field, even if most of the actual data items only costs less than 10 bytes?

Variable-Length Records

- 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 in Block



- 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 records — instead, they should point to the entry for the record in header

- Page size is usually aligned with the disk block size (4KB-8KB)

Storing Large Objects

- E.g., BLOB/CLOB types
  - BLOB: Binary Large OBject
  - CLOB: Character Large OBject
- Records must be smaller than pages
- Alternatives:
  - Store as files in file systems
    - Store the file name (usually a path in the file system) as an attribute of a record in the database
  - Store as files managed by databases via B+ tree
    - B+-tree file organizations permit us to read an entire object, or specified byte ranges in the object, as well as to insert and delete parts of the object
  - Break into pieces and store in multiple tuples in separate relation
    - PostgreSQL TOAST

Organization of Records in Files

- Heap – records 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
- 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 even with inserts/deletes
- Hashing – a hash function computed on search key; the result specifies in which block of the file the record should be placed

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 map

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)

Sequential File Organization

- Store records in sequential order, based on the value of the search key of each record
- 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

- It is difficult, to maintain physical sequential order as records are inserted and deleted
- 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

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

Multitable Clustering File Organization

- Store several relations in one file using a multitable clustering file organization
  - Good for queries involving:
    - *department* ⋈ *instructor*
    - or: one single department and its instructors (one-to-many correspondense)
  - Bad for queries involving only *department*
    - E.g., select * from department
  - Results in variable size records
    - Can add pointer chains to link records of a particular relation
  - Bad for large databases
    - where other operations than joins are required

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,
clustering key is dept_name

Comp. Sci.	Taylor	100000	
10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000
Physics	Watson	70000	
33456	Gold	Physics	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
    - since the cost of some operations, such as finding free space for a record, increase with relation size
  - 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

Column-Oriented Storage

- Also known as columnar representation
- Store each attribute of a relation separately

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

Column-Oriented Storage

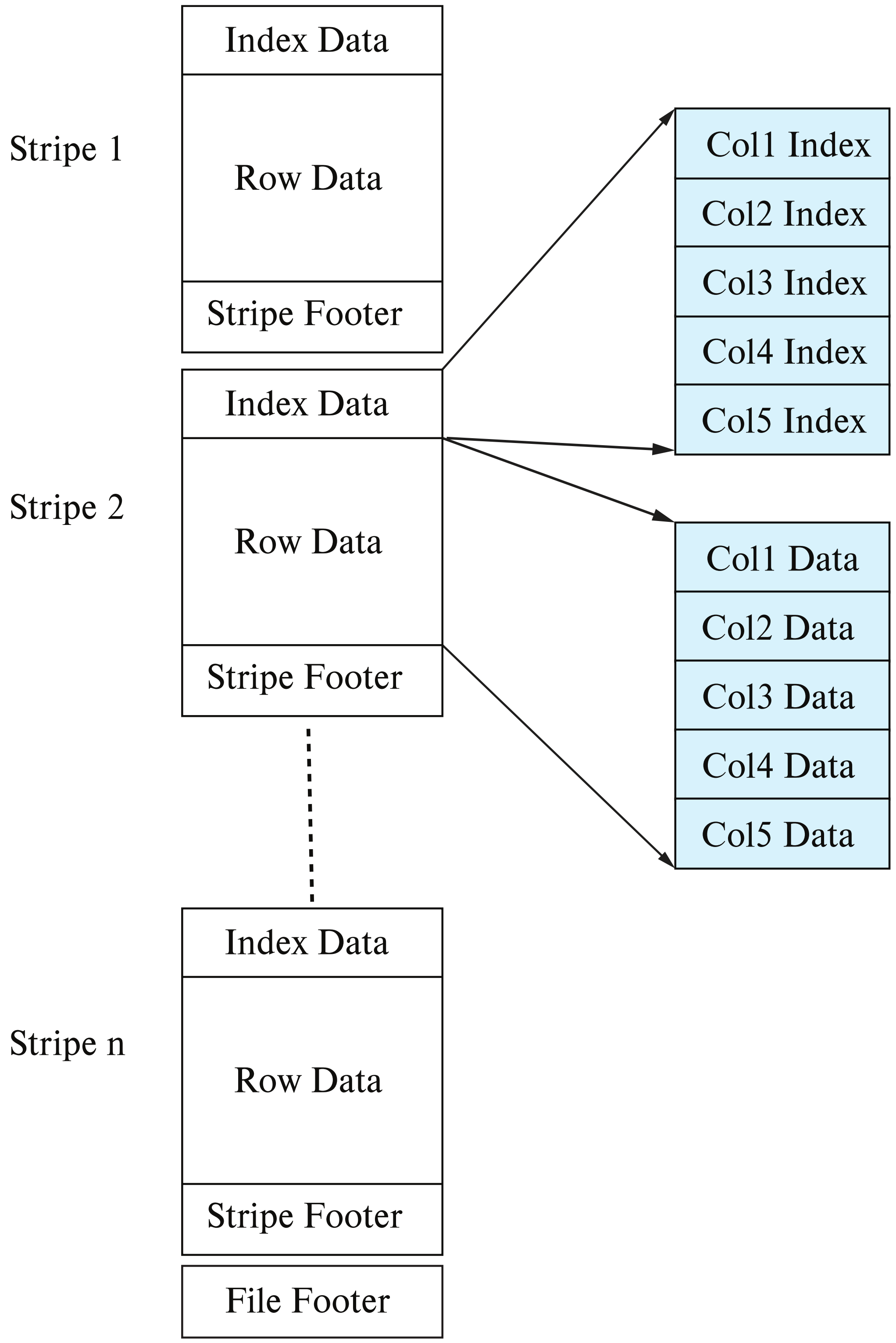
- Benefits:
  - Reduced IO if only some attributes are needed
  - Improved CPU cache performance
    - data analysis queries usually access many values of an attribute consecutively
  - Improved compression
    - Data in the same type can be compressed more efficiently
  - Vector processing on modern CPU architectures
- Drawbacks
  - Cost of tuple reconstruction from columnar representation
  - Cost of tuple deletion and update
  - Cost of decompression

Column-Oriented Storage

- 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

Column-Oriented Storage

- ORC (Optimized Row Columnar) File Format
  - A row-oriented representation is converted to column-oriented representation
  - Each stripe contains values of a subset of columns
  - File format with columnar storage inside file
    - Row Data
    - Index Data
- Details if you are interested in big data processing:
  - https://cwiki.apache.org/confluence/display/hive/languagemanual+orc



Data Dictionary Storage

- The Data dictionary (also called system catalog) stores metadata
  - … that is, data about data – “meta” means “of”
- It includes
  - 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
    - Authorization for each user, default schemas of the users, passwords etc
  - Statistical and descriptive data
    - Number of tuples in each relation
    - Number of distinctive values for each attribute

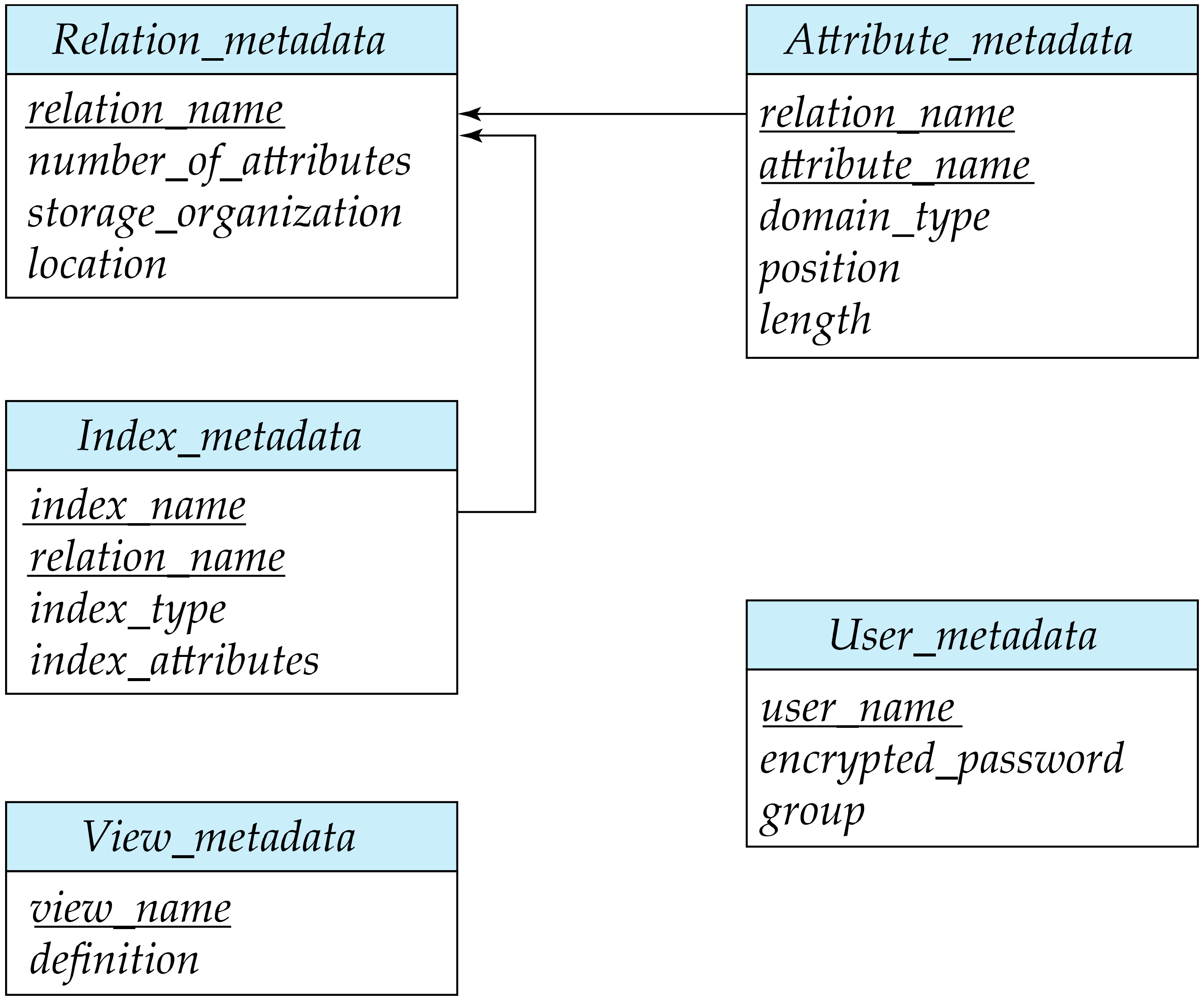
Data Dictionary Storage

- It includes (continue)
  - Physical file organization information
    - How relation is stored (sequential/hash/…)
    - Physical location of relation
  - Information about indices/indexes

(we will learn it later)

Relational Representation of System Metadata

- We could store metadata by using special-purpose data structures and code
  - But a preferable way is to store them as relations in database
- An example of the relational representation of the metadata
  - Relational representation on disk
  - Specialized data structures designed for efficient access, in memory
  - Different DBMS may have their own implementation
- First consult meta-data, then fetch records
  - Maybe better to put meta-data in memory to achieve fast access



Storage Access

- Many large databases are much larger than the available memory on servers
  - Database data reside primarily on disk in most databases
  - The data must be brought into memory to be read or updated; and updated data blocks must be written back to disk subsequently
- Blocks are <u>units</u> of both <u>storage allocation</u> and <u>data transfer</u>
- Database system seeks to minimize <u>the number of block transfers between the disk and memory</u>
  - We can reduce the number of disk accesses by <u>keeping as many blocks</u> as possible in <u>main memory</u>
  - Buffer: <u>Portion of main memory</u> available to store <u>copies of disk blocks</u>
  - Buffer Manager: Subsystem responsible for <u>allocating buffer space</u> in main memory

Self Study

- Database System Concepts , 7^{th} Edition
  - Chapter 13.5 “Database Buffer”