

Chapter 10: Storage and File Structure

- Overview of Physical Storage Media
- Magnetic Disks
- RAID
- Storage Access
- File Organization
- Organization of Records in Files
- Data-Dictionary Storage
- Disk Performance Models
- I/O Efficient Sorting

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 battery backed up main-memory.

Physical Storage Media

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

Physical Storage Media (Cont.)

■ Flash memory

- Data survives power failure
- Data can be written at a location only once, but location can be erased and written to again
 - ▶ Can support only a limited number (10K – 1M) of write/erase cycles.
 - ▶ Erasing of memory has to be done to an entire bank of memory
- Reads are almost as fast as main memory (but not quite)
- But writes are slow (few microseconds), erase is slower
- Actual performance depends on file system and details
- Widely used in embedded devices such as digital cameras, phones, and USB keys

Physical Storage Media (Cont.)

■ Magnetic-disk

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

Physical Storage Media (Cont.)

■ Optical storage

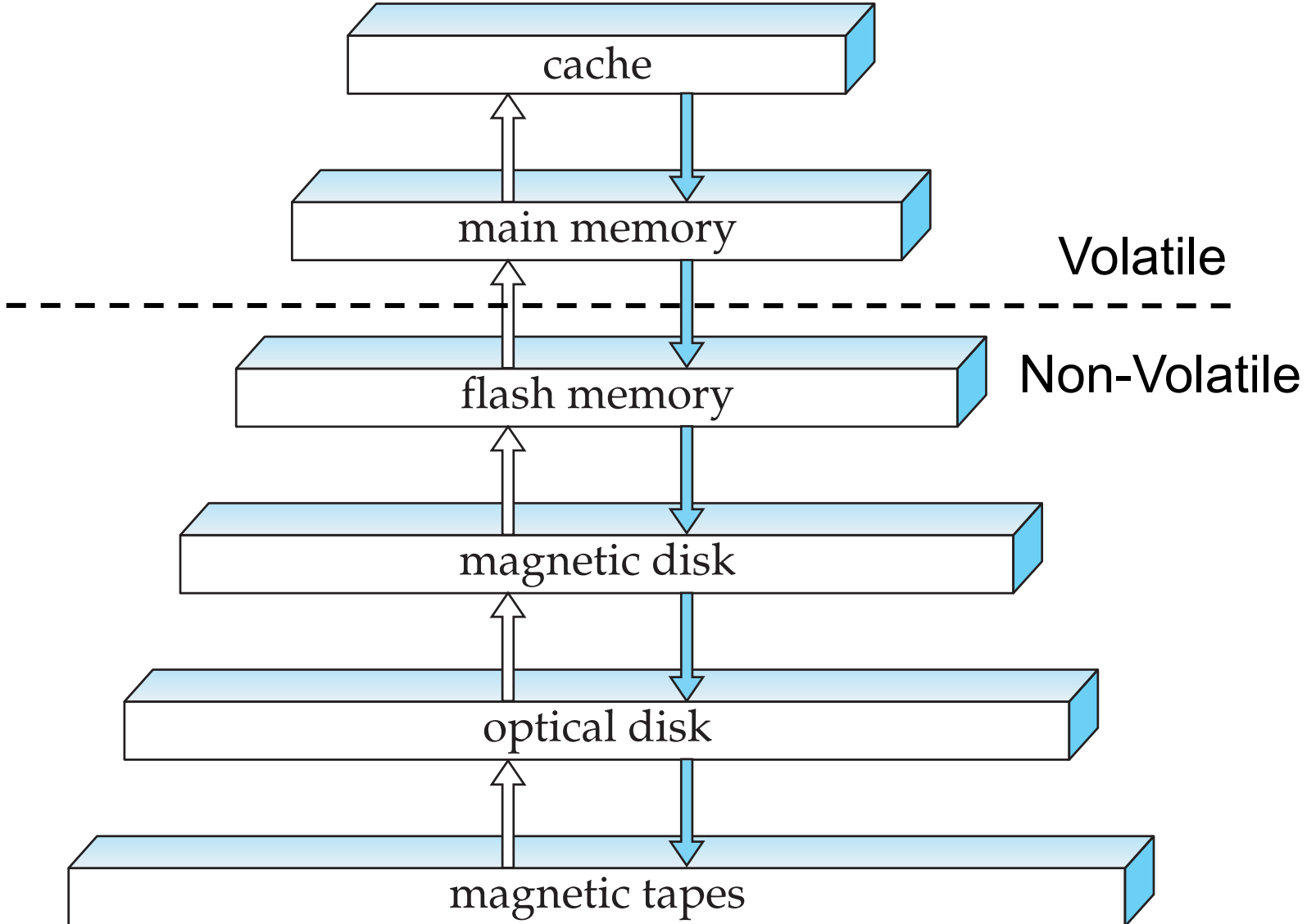
- Non-volatile, data is read optically from a spinning disk using a laser
- CD-ROM (640 MB) and DVD (4.7 to 17 GB) most popular forms
- Blu-ray disks: 27 GB to 54 GB
- Write-one, read-many (WORM) optical disks used for archival storage (CD-R, DVD-R, DVD+R)
- Multiple write versions also available (CD-RW,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

Physical Storage Media (Cont.)

■ Tape storage

- non-volatile, used primarily for backup (to recover from disk failure), and for archival data
- **sequential-access** – much slower than disk
- very high capacity (40 to 300 GB tapes available)
- tape can be removed from drive \Rightarrow 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)

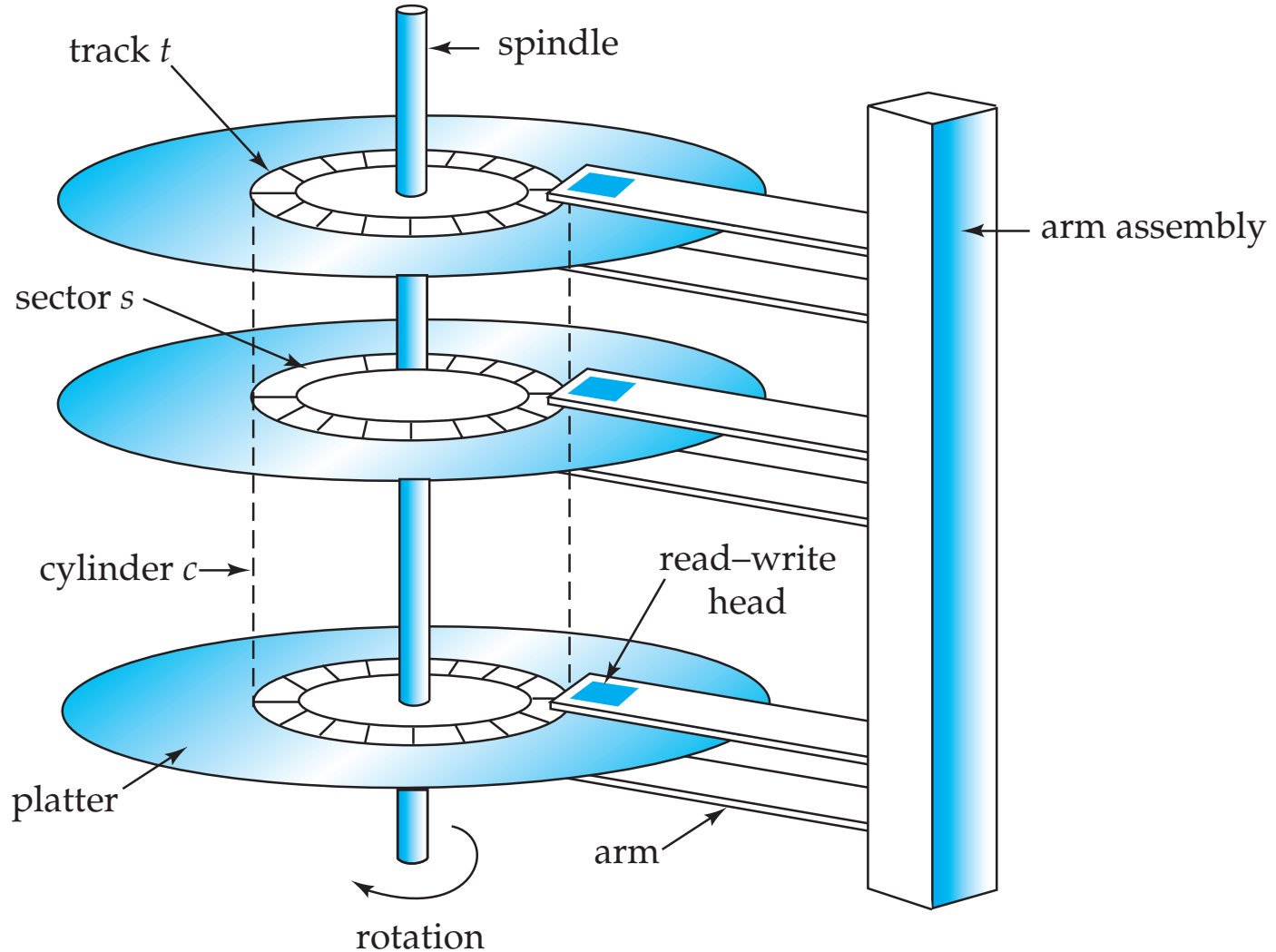
Storage Hierarchy



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
- Another issue: remote storage, clouds, etc.

Magnetic Hard Disk Mechanism



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

Actual Hard Disk Enclosed in Casing



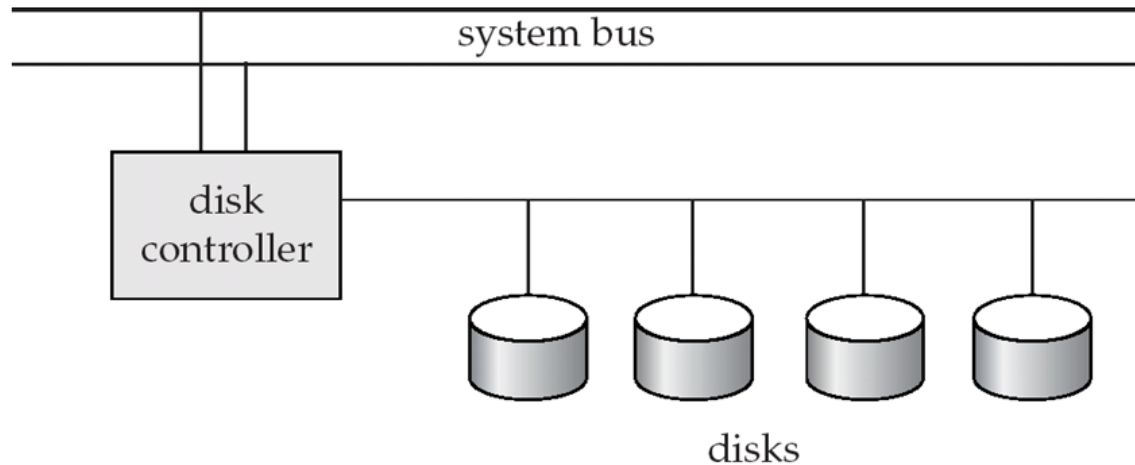
Magnetic Disks

- 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 side, mounted on a common arm.
- **Cylinder** i consists of i^{th} track of all the platters

Magnetic Disks (Cont.)

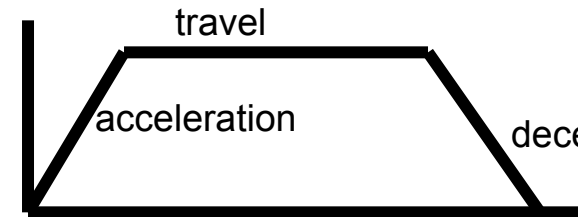
- Earlier generation disks were susceptible to head-crashes
 - Surface of earlier generation disks had metal-oxide coatings which would disintegrate on head crash and damage all data on disk
 - Current generation disks are less susceptible to such disastrous failures, although individual sectors may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

Disk Subsystem



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

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
- 

The graph illustrates the speed profile of a disk head during a seek operation. The vertical axis represents speed, and the horizontal axis represents distance. The profile starts with a vertical line, followed by a diagonal line labeled 'acceleration', a horizontal line labeled 'travel', and finally a diagonal line labeled 'deceleration'.
- **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 100 MB per second max rate, lower for inner tracks
 - Multiple disks may share a controller, so rate that controller can handle is also important
 - ▶ E.g. SATA: 150 MB/sec, SATA-II 3Gb (300 MB/sec)
 - ▶ Ultra 320 SCSI: 320 MB/s, SAS (3 to 6 Gb/sec)
 - ▶ Fiber Channel (FC2Gb or 4Gb): 256 to 512 MB/s

Performance Measures (Cont.)

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

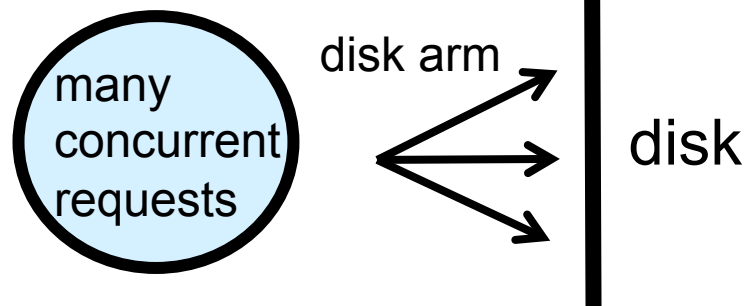
Disk Subsystem

- 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

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**

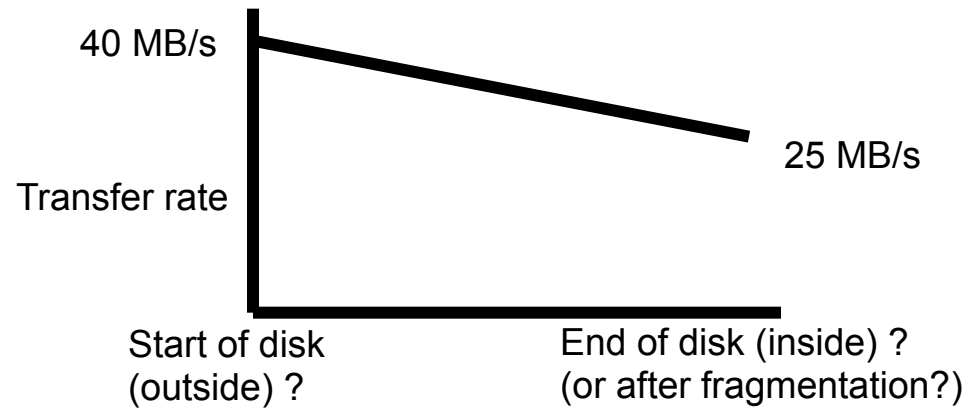


- **File organization** – optimize block access time by organizing the blocks to correspond to how data will be accessed
 - E.g. Store related information on the same or nearby cylinders.
 - Files may get **fragmented** over time
 - ▶ Sequential access to fragmented file means increased arm movement
 - Some systems have utilities to **defragment** the file system, in order to speed up file access

EXAMPLE

■ SEAGATE BARRACUDA (circa 2003)

- 80GB CAPACITY
- 2 PLATTERS
- 7200 RPM → 120 RPS → 8.33 ms/rotation
- 2048 KB CACHE, IDE
- Access time 13.7ms



■ Today's disks:

- 70-120 MB/s (cheap SATA disks)
- 3TB capacity at \$150
- Access time between 5 and 10 ms

Disk Performance Model

- Seek Time (5ms)
- Rotational Latency (5ms)
- Transfer Rate (80MB/s)
- File of size 400KB modeled as having sequential layout



Time to read: t_R = seek time + rotational latency + transfer

Time to read: t_R = 5ms + 5ms + 5ms = 15ms

- FILE of size 4KB (or 8, 16..)

t_B = 10.05ms

NOTE: often have blocks of 4/8/16 KB

5400 RPM = 90 RPS
⇒ 11ms/rotation
⇒ 5.5ms half rotation

HARD DISK MODELS

1. **BLOCK MODEL:** It takes x ms to read each block of size 4KB (or 8KB, or 16KB, but block size is fixed).

⇒ it take $10x$ ms to read 40KB

2. **LATENCY TRANSFER-RATE (LTR) MODEL:** It takes LATENCY + TRANSFER TIME to read a file.

LTR more precise when reading large pieces of data sequentially

For small random reads, no difference!

EXAMPLE

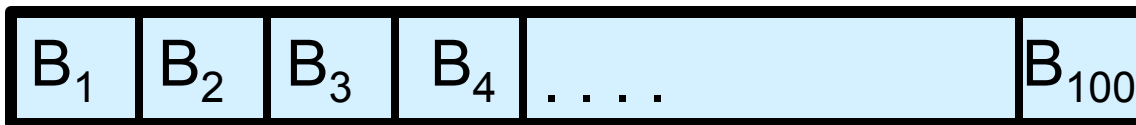
■ 7200 RPM, 5ms SEEK, 80MB/s TRANSFER RATE

■ **BLOCK MODEL** : Block size 4KB, file size 400KB

⇒ $5\text{ms} + 1000/240 \text{ ms} + 0.05\text{ms}$

⇒ 9.216ms to read block

⇒ $100 * 9.21666 = 921.666\text{ms}$ to read 400KB file.



Note: Vast overestimate of the actual time needed!

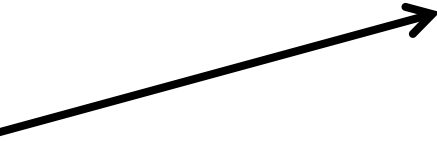
EXAMPLE (Cont.)

■ LTR MODEL :

⇒ 5ms + 1000/240 ms + 5ms

⇒ 14.16ms to read 400KB file

Transfer time for
400KB at 80 MB/s



Much more realistic!

Why is the block model more popular?

- Simpler? (no reasoning about data layout involved)
- OK if mostly small random reads and writes (transactions)
- But not good for other scenarios (data mining/OLAP/search)

DISK WORKLOAD EXAMPLES

- **TRANSACTIONS:** Many small reads and writes (typical DB transaction workload): block or LTR model fine
- **DATA ANALYSIS:** data mining, search engines, scientific computing: reading (scanning) and writing large files: block model not good at all → use LTR
- **INTERACTIVE** (e.g., UNIX users) : Many repeated reads, few writes. Caching works. Motivation for log structured file systems

SSD: Solid State Drives

- Non-volatile, starting to replace hard disk in servers + laptops
- Still more expensive than hard drives (HDD)
- But getting cheaper: now maybe \$200 for 256GB
- Much faster: 50-100 times faster random reads!
- Transfer rate $>200\text{MB/s}$, $<100\text{ us}$ per random access
- But tricky to model performance, limited # writes issue
- Big impact on large data and I/O-efficient computing
- Great for database transaction processing

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
- Originally a cost-effective alternative to large, expensive disks
 - I in RAID originally stood for “inexpensive” (now “independent”)
 - Today RAIDs are used for their higher reliability and bandwidth.

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)

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** – 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

- 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.)

- **RAID Level 2: Memory-Style Error-Correcting-Codes (ECC) with bit striping.**
- **RAID Level 3: Bit-Interleaved Parity**
 - A single parity bit is enough for error correction, not just detection, since we know which disk has failed
 - ▶ When writing data, corresponding parity bits must also be computed and written to a parity bit disk
 - ▶ To recover data in a damaged disk, compute XOR of bits from other disks (including parity bit disk)
 - Faster data transfer than with a single disk, but fewer I/Os per second since every disk has to participate in every I/O.
 - Subsumes Level 2 (provides all its benefits, at lower cost).



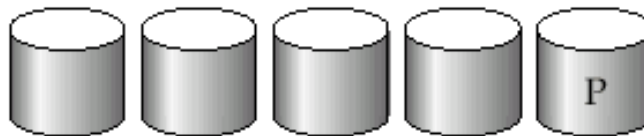
(c) RAID 2: memory-style error-correcting codes



(d) RAID 3: bit-interleaved parity

RAID Levels (Cont.)

- **RAID Level 4: Block-Interleaved Parity**; uses block-level striping, and keeps a parity block on a separate disk for corresponding blocks from N other disks.
 - When writing data block, corresponding block of parity bits must also be computed and written to parity disk
 - To find value of a damaged block, compute XOR of bits from corresponding blocks (including parity block) from other disks.
 - Provides higher I/O rates for independent block reads than Level 3
 - ▶ block read goes to a single disk, so blocks stored on different disks can be read in parallel
 - Provides high transfer rates for reads of multiple blocks than no-striping
 - Before writing a block, parity data must be computed
 - Parity disk becomes a bottleneck for independent block writes since every block write also writes to parity disk



(e) RAID 4: block-interleaved parity

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 $(n \bmod 5) + 1$, with the data blocks stored on the other 4 disks.
 - Higher I/O rates than Level 4.
 - ▶ Block writes occur in parallel if the blocks and their parity blocks are on different disks.
 - Subsumes Level 4: provides same benefits, but avoids bottleneck of parity disk.

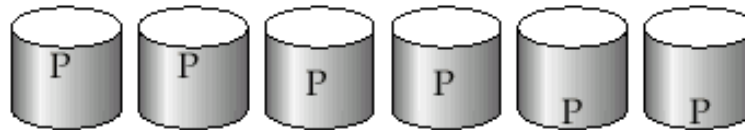


(f) RAID 5: block-interleaved distributed parity

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

RAID Levels (Cont.)

- **RAID Level 6: P+Q Redundancy** scheme; similar to Level 5, but stores extra redundant information to guard against multiple disk failures.
 - Better reliability than Level 5 at a higher cost; not used as widely.



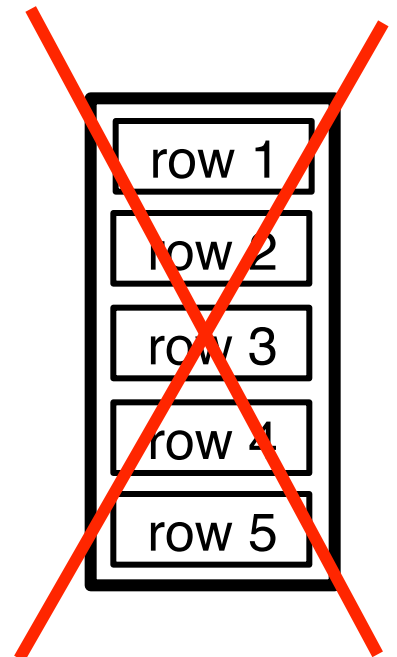
(g) RAID 6: P + Q redundancy

Choice of RAID Level

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

How are Tables Stored on Disk?

- The database is stored as a collection of *files*.
- In fact, each table is stored in one or more files, where each file consists of multiple 4KB or 16KB pages (blocks).
- E.g., 1GB relation = 50 files of 20MB each: file1, file2, file3, ..
- Simple solution: each page has 20 rows of size 200 bytes:
- But this is too naive:
 - How about insertions and deletions?
 - How about sorted tables?
 - How about variable-length rows?
- Need to use more complicated schemes
- Compare to memory allocation (but on disk)



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.

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 $i + 1, \dots, n$ to $i, \dots, n - 1$
- 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

Free Lists

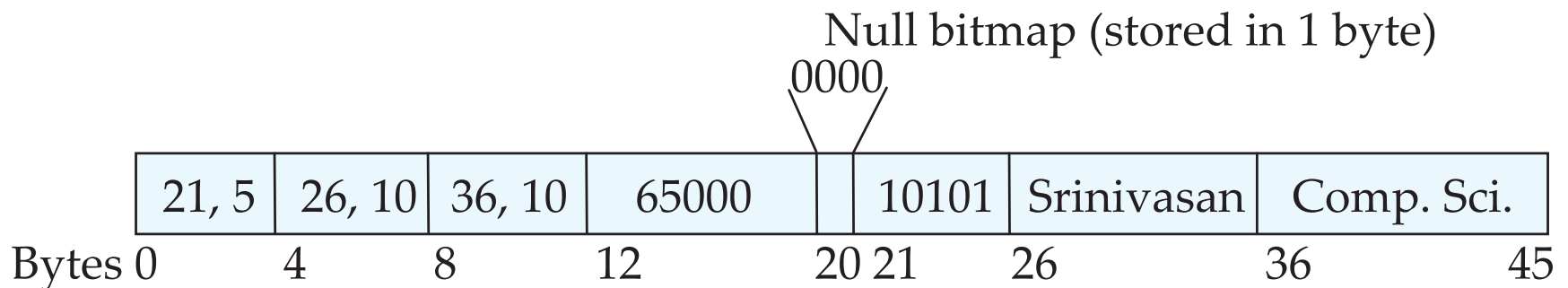
- Store the address of the first deleted record in the file header.
- Use this first record to store the address of the second deleted record, and so on
- Can think of these stored addresses as **pointers** since they “point” to the location of a record.
- More space efficient representation: reuse space 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

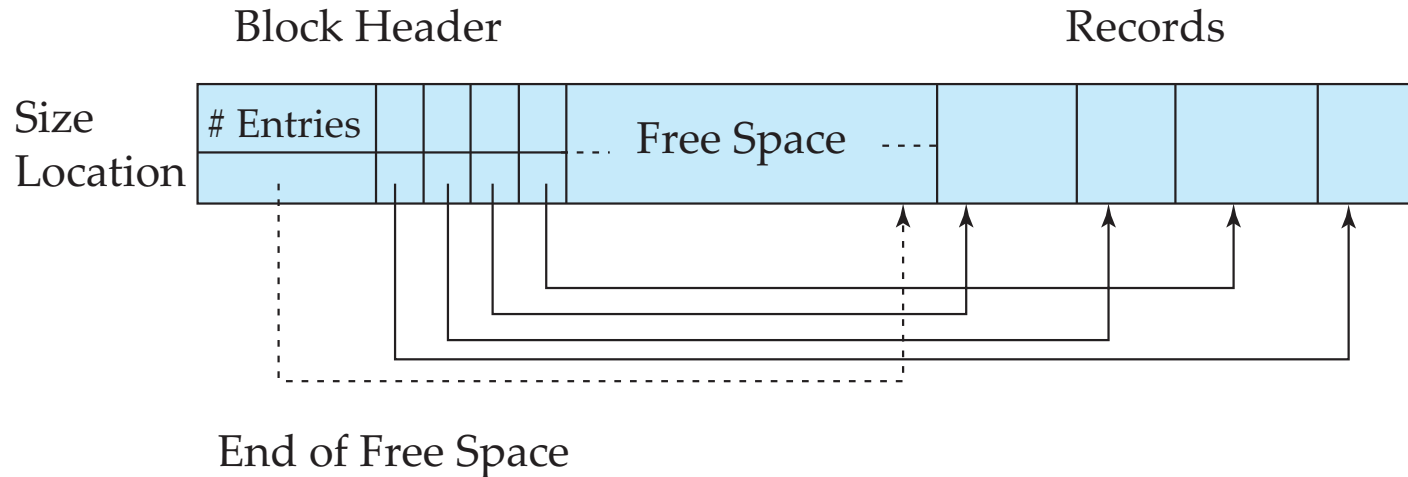
The diagram illustrates a linked list of free records. Arrows show the sequence: record 0 points to record 1, record 1 points to record 4, record 4 points to record 6, and record 6 points to a ground symbol, indicating the end of the list.

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 – arrays or multi-sets).
- 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.

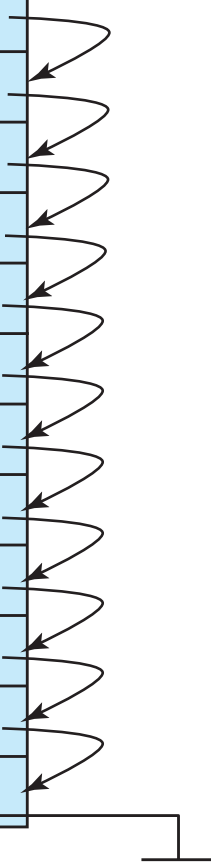
Organization of Records in Files

- **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

Sequential File Organization

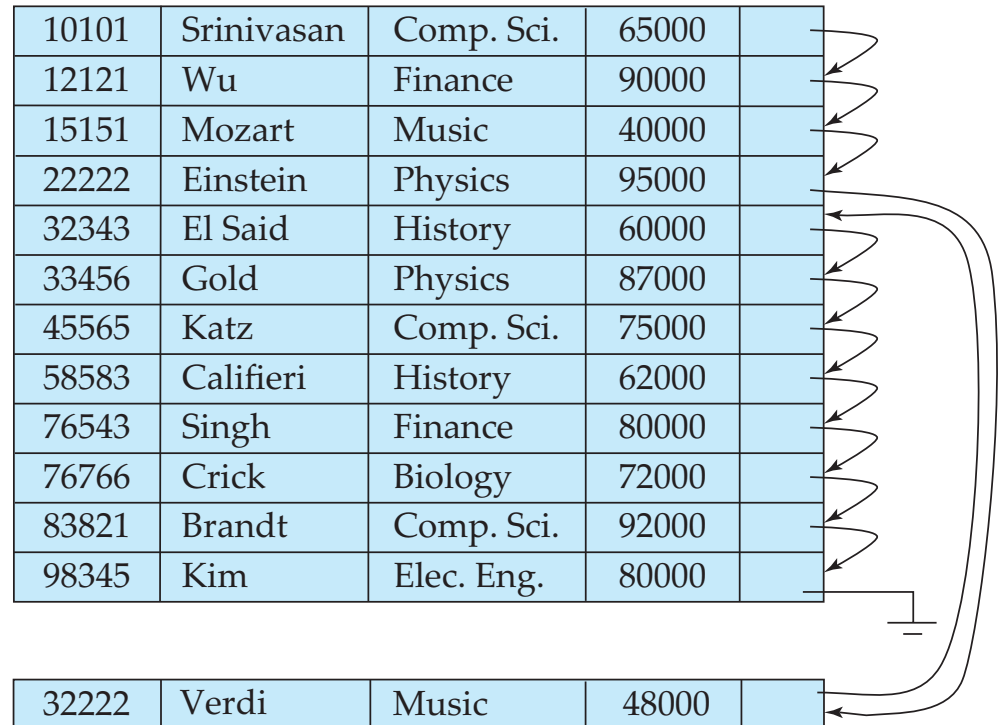
- Suitable for applications that require sequential processing of the entire file
- The records in the file are ordered by a **search-key**

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



Multitable Clustering File Organization

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

department

<i>dept_name</i>	<i>building</i>	<i>budget</i>
Comp. Sci.	Taylor	100000
Physics	Watson	70000

instructor

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


multitable clustering
of *department* and
instructor

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

Multitable Clustering File Organization (cont.)

- Good for queries involving *department* ⋈ *instructor*, and for queries involving one single department and its instructors
- Bad for queries involving only *department*
- Can add pointer chains to link records of a particular relation

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



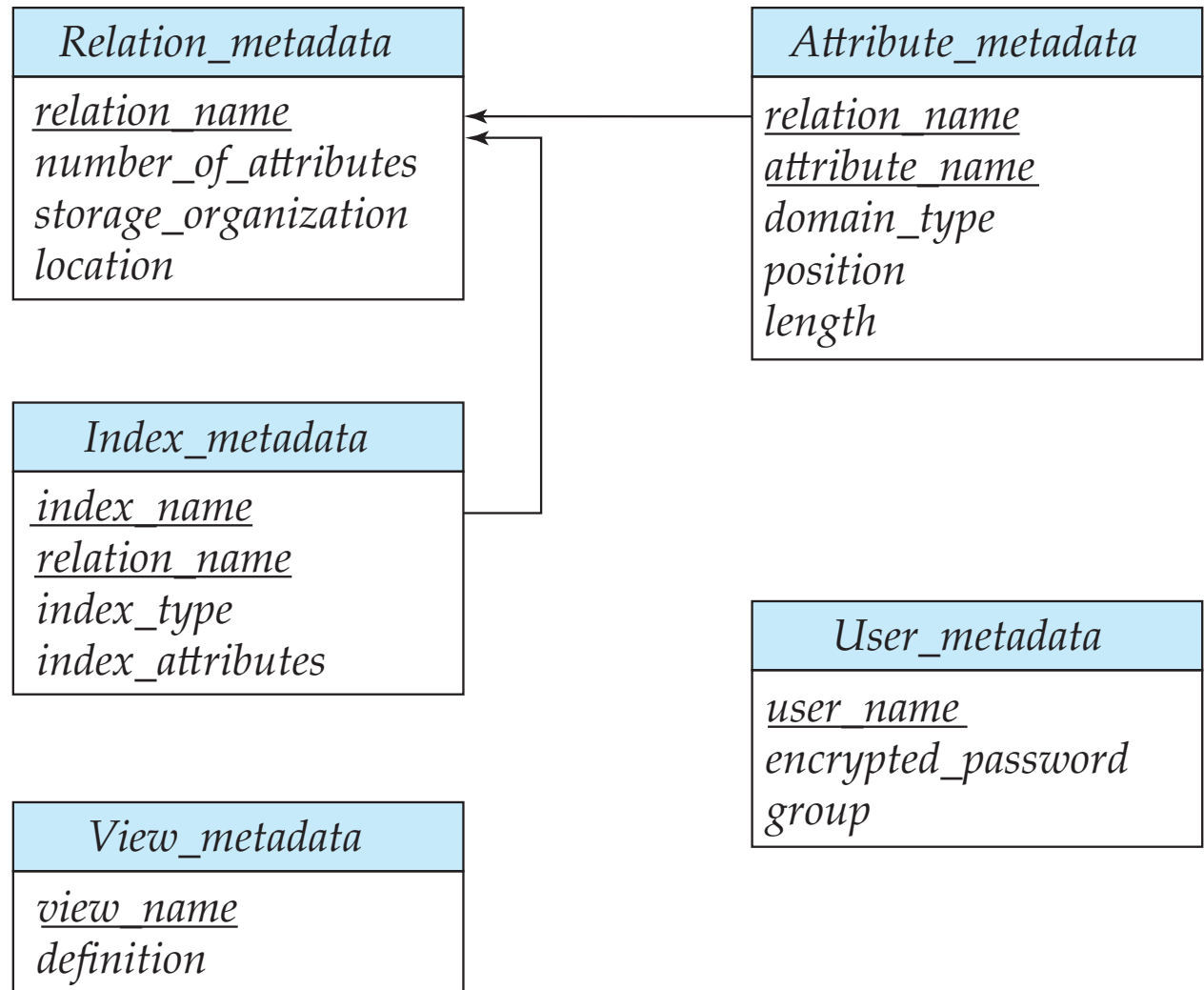
Data Dictionary Storage

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 (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.
- 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.

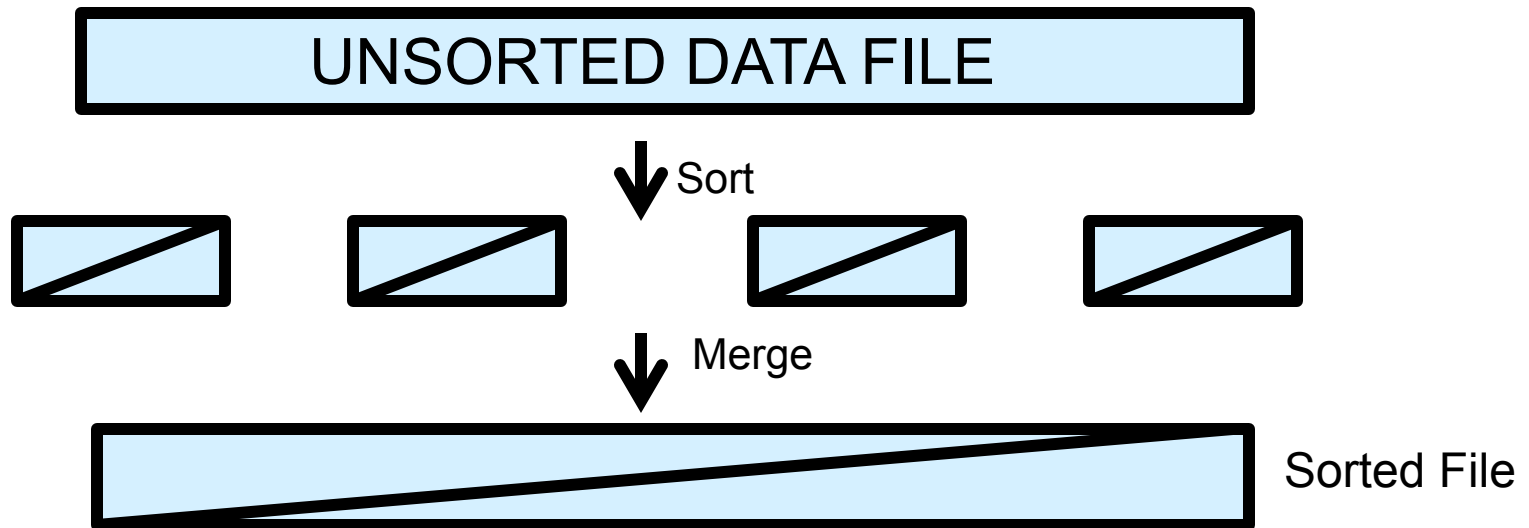
I/O – Efficient Sorting

■ Sorting needed in many cases:

- Output in sorted order
- Sort – based join algorithms
- Offline B-tree index construction
- Inverted index construction (*contains* op, search)
- Duplicate elimination
- Group-by operations

I/O – Efficient Sorting

- Data may not fit in main memory
 - Many algorithms will be inefficient if data on disk
 - Most popular I/O-efficient method: Merge Sort



MERGE SORT EXAMPLE

- Data file (256 million records of 100 bytes each) → 25.6 GB
- Main memory available for sorting: 100 MB
- Algorithm:
 - Load pieces of size 100 MB, sort each piece, and write to a file
 - Use d-way merging to merge d sorted files into one larger sorted file, until only one file left



- How to choose d?
- How to merge d lists?

IN MORE DETAIL

25.6 GB of data (256 million records of 100 Bytes)

100 MB of work space in main memory

Phase 1: Repeat:

- read 100 MB data
- sort in main memory using any sorting algo
- write into a new file

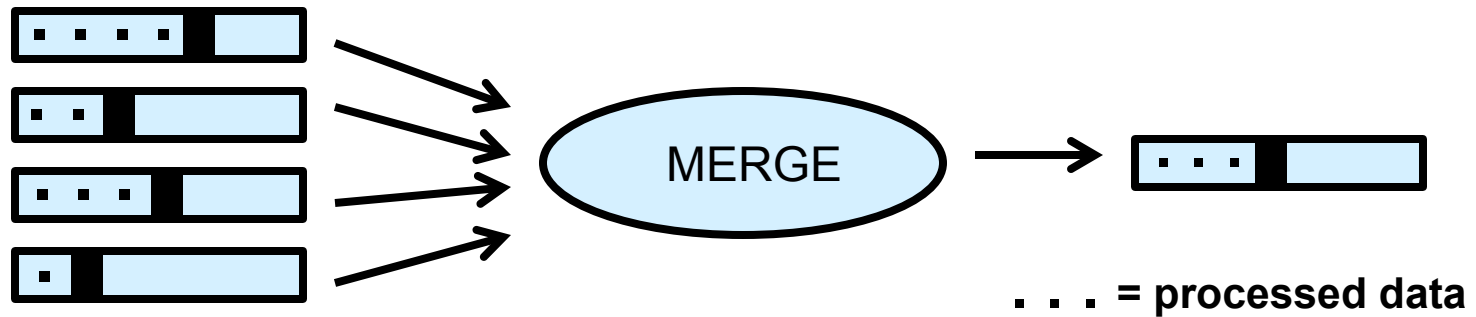
Until all data read

Phase 2: Merge the 256 files created in Phase 1

- in 1 pass: merge 256 files into one
- in 2 passes: merge 256 files into 16, then 16 into 1

DATA ACCESS/MOVEMENT DURING MERGE

d-way merge: need d input buffers and 1 output buffer

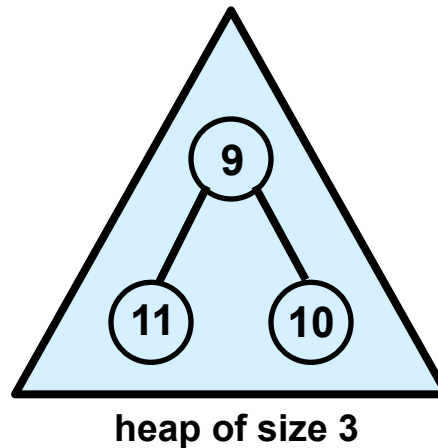


- Files sorted in ascending order (left to right)
- If output buffer full, write it out: append to output file
- If input buffer empty, read next chunk of data from that file

Larger d: fewer passes but smaller buffers, thus slower disk I/O

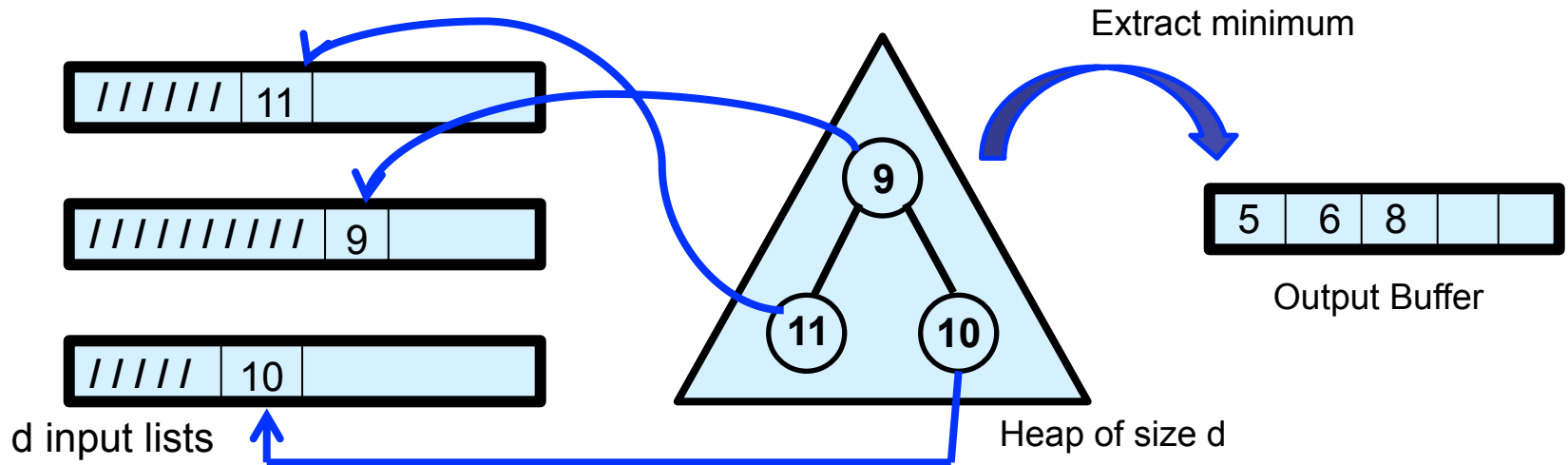
USING HEAPS

RECALL: HEAPS (PRIORITY QUEUES)



- Basic data structure supporting insert, delete, and extracting the minimum (maximum) in $O(\lg n)$ time
- Minimum stored in root
- But often laid out in simple array structure

HOW TO MERGE d LISTS



- Initially insert leftmost (smallest) item from each list into heap
- Extract minimum and write out to output buffer
- Replace extracted element with the next element from the list where the minimum came from, then heapify again
- Repeat steps until heap is empty \rightarrow all d lists are merged

EXAMPLE

Back to our example:

25.6 GB of data to be sorted

100 MB of main memory available for sorting

⇒ after sort phase, we need to merge 256 sorted files of size 100 MB each

Disk with 10ms access time (seek time plus rotational latency) and 50 MB/s maximum transfer rate

Thus it takes $10 + x/50$ ms to read x KB of data

What is the best choice of d ?

$d = 2$ (8 passes since $2^8 = 256$)

$d = 16$ (2 passes since $16^2 = 256$)

$d = 256$ (1 pass)

EXAMPLE (Cont.)

$d = 2$:

- 2 input and 1 output buffer of 33.33MB each
- Reading/writing one buffer of data takes :
 $10 + 33333/50 = 676.66 \text{ ms}$

⇒ Reading all 25.6 GB in 768 pieces of 33.33 MB takes:
 $768 * 676.66 \sim 520 \text{ s}$

⇒ Each pass (read in + write out)
 1040 s

⇒ All 8 passes
 $8320 \text{ seconds} = 2.3 \text{ hours}$

EXAMPLE (Cont.)

d = 16:

- 16 input and 1 output buffer of 5.88 MB each
- Reading/writing one buffer of data takes :
 $10 + 5.888/50 = 127.6 \text{ ms}$

⇒ Reading all 25.6 GB in 4354 pieces of 5.88 MB takes:
 $4354 * 127.6 \text{ ms} = 555.6 \text{ s}$

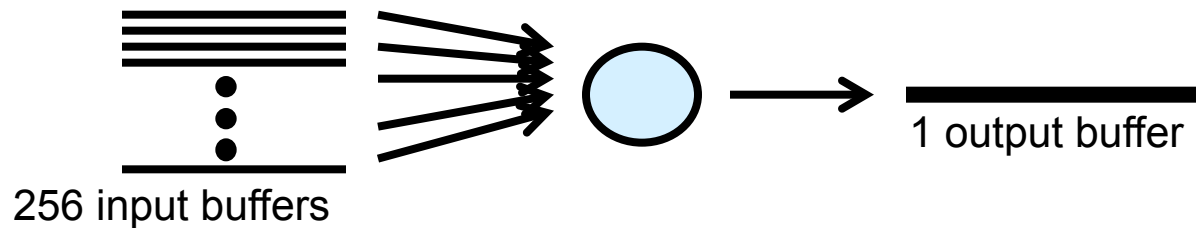
⇒ Each pass (read in + write out)
1111.2 sec

⇒ Total time for 2 passes
2222.4 sec or about 40 minutes

EXAMPLE (Cont.)

$d = 256$:

- 257 buffers of 389 KB each



- Reading/writing one buffer of data takes:
 $10 + 389/50 \sim 17.78 \text{ ms}$ (more than half of time on seeks)

⇒ Reading all 25.6 GB takes:
1170 s

⇒ Total (read in + write out)
2340 seconds, or slightly slower than $d=16$

Conclusion

⇒ Choose d:

- Not too small (Few passes)
- Not too large (Fast disk access)
- Typically 1 or 2 passes for current machines
- Make the output buffer larger than the input buffer

E.g.:

- Choose 16 input buffers of 5 MB each
- 1 output buffer of 20 MB