Storage and File Structure

Outline

- Overview of Physical Storage Media
- Magnetic Disks
- RAID
- File Organization
- Organization of Records in Files

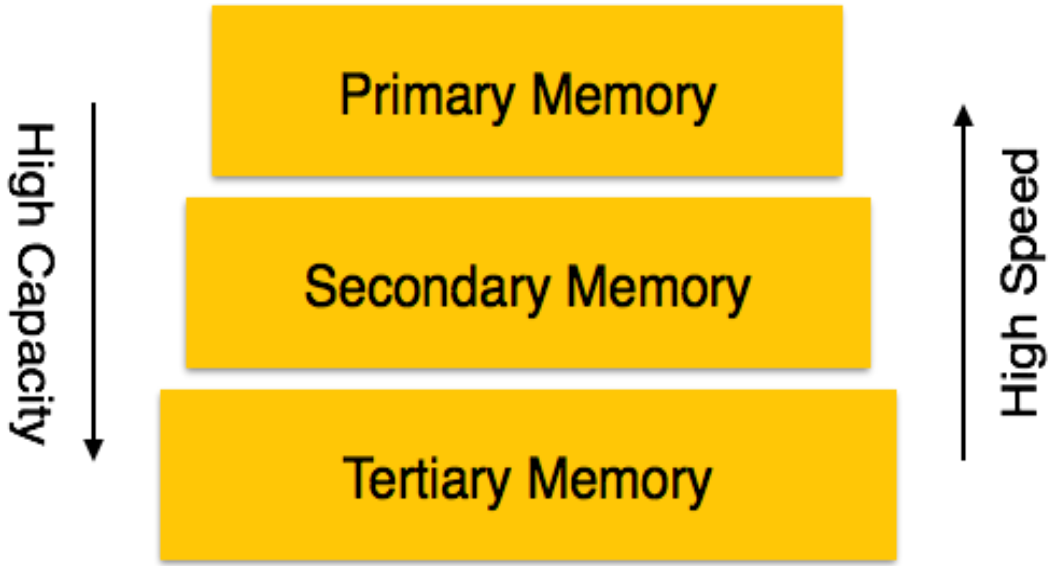
Overview of Physical Storage Media

Storage System

- Databases are stored in file formats, which contain records.

- At physical level, the actual data is stored in electromagnetic format on some device.

These storage devices can be broadly categorized into three types −



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.

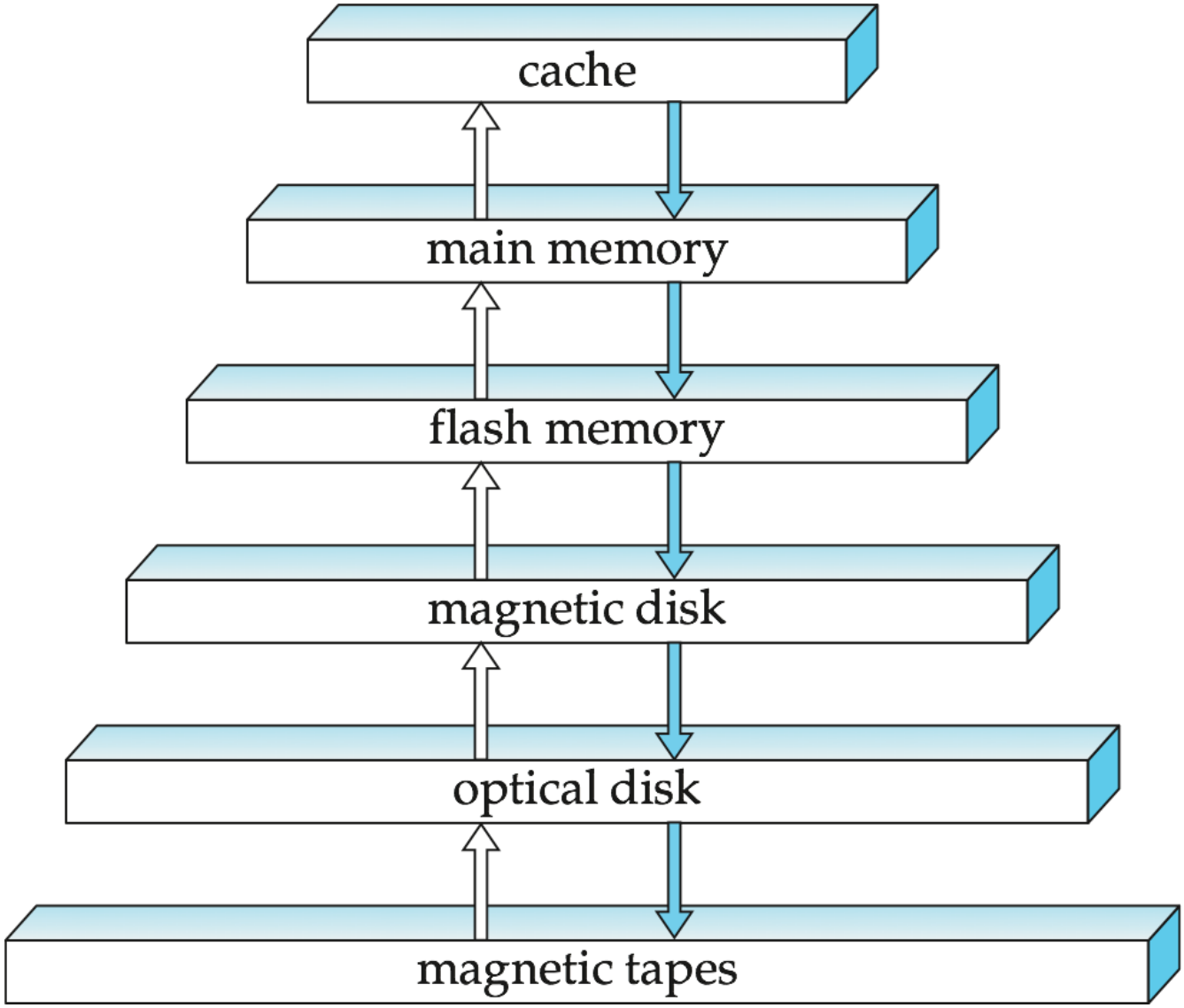
contd.,

- Large number of disks are needed to store multiple applications (Web, multimedia), even though disk-drive capacities have been growing very fast.
- Large number of disks in a system presents opportunities for improving the rate access of data.
- Furthermore, this setup offers the potential for improving the reliability of data storage, since redundant information can be stored on multiple disks.
- Failure of one disk does not lead to loss of data.

contd.,

- in the past, designer uses small disk rather than larger one.
- Now a days, all storages are relatively small and cheep.
- RAID stands for inexpensive.
- RAID systems are used for higher reliability and higher performance rate, rather than for economic reasons.
- Easier management and operations.

Storage Hierarchy



Physical Storage Media

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

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

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
    - 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

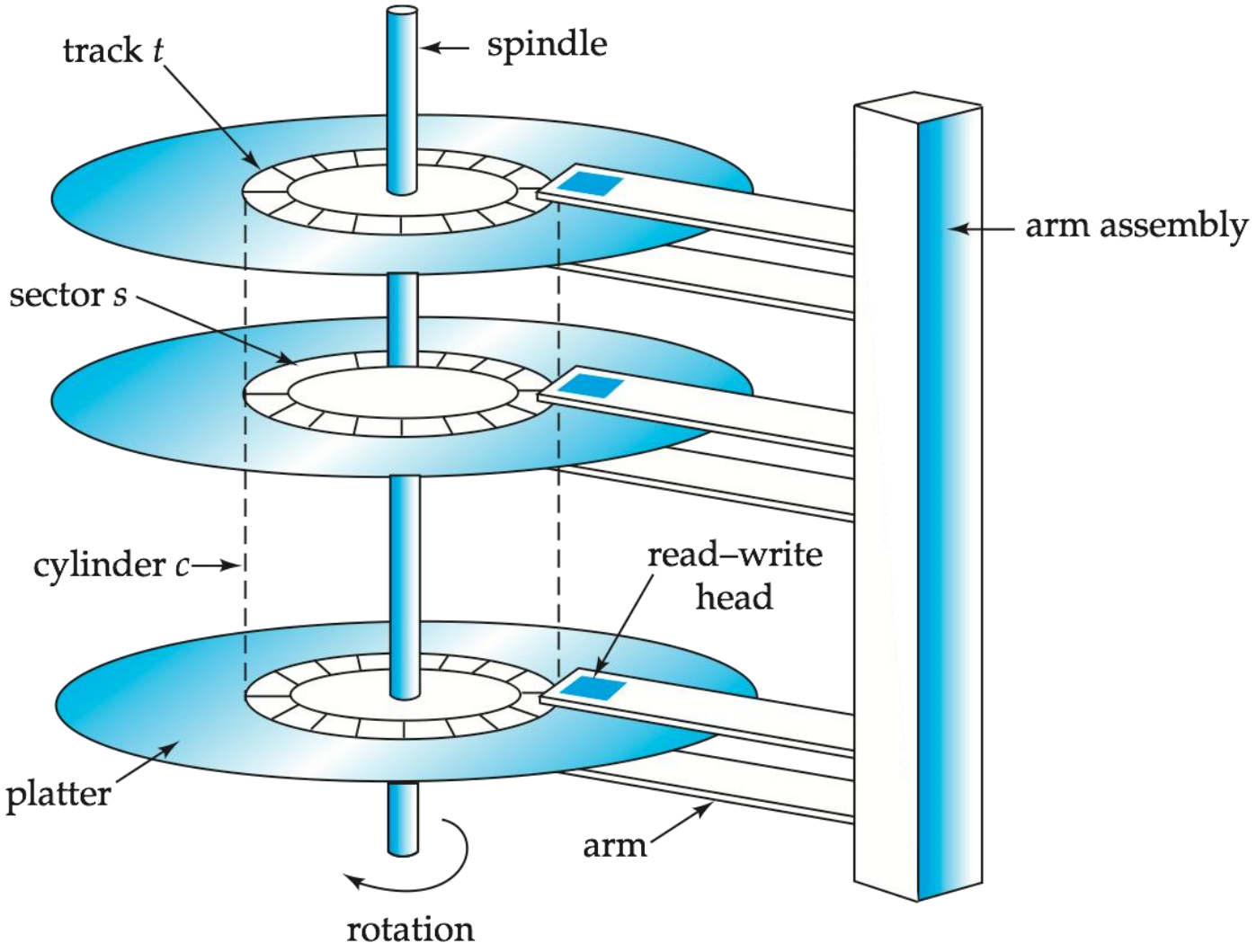
Physical Storage Media (Cont.)

- **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 ⇒ 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 (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

Magnetic Hard Disk Mechanism



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

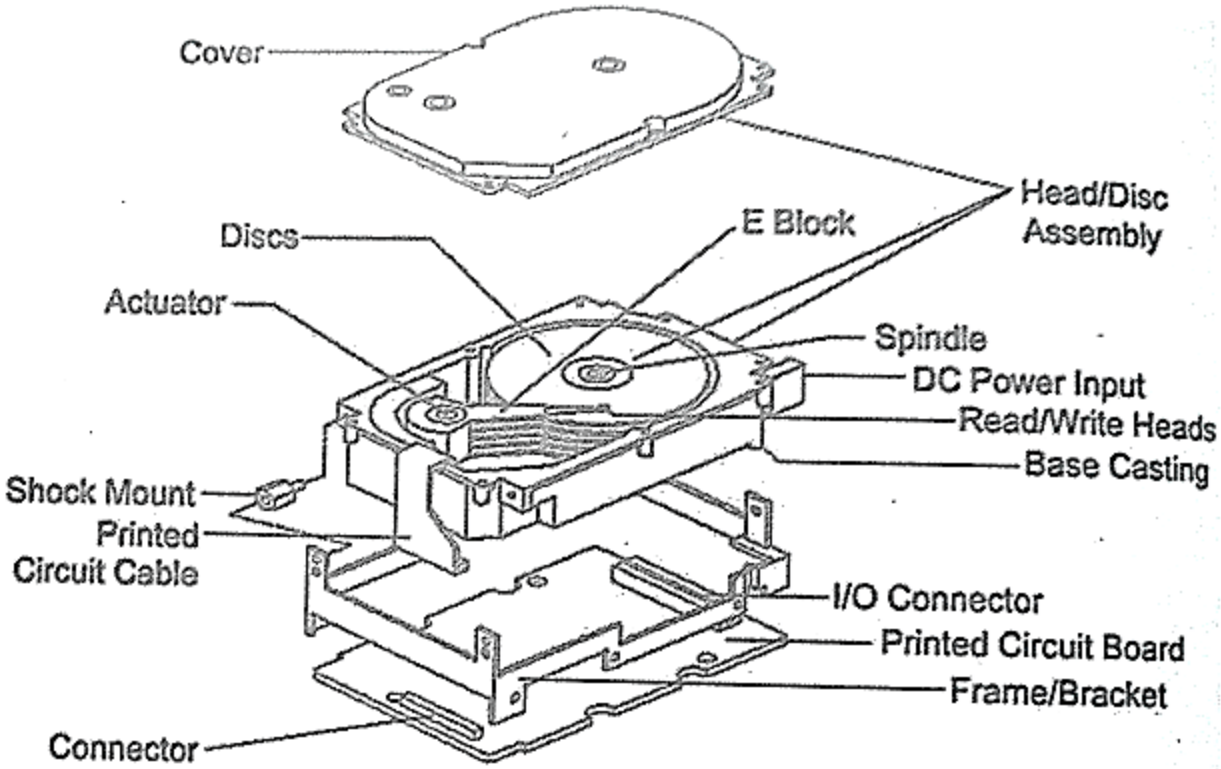


Fig: Components of Hard Disk

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

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 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
  - MTTF decreases as disk ages

RAID

- Data Storage requirements – Web, database and some multimedia application.
- Disk drive capacity growing fast.
- Reliability of data storage-Redundant information stored on multiple disks.
- Thus, the failure of one disk does not lead to loss of data.
- A variety of disk-organization techniques, collectively called **redundant arrays of independent disks** (**RAID)**, have been proposed to achieve improved performance and reliability.

Redundant Array of Independent Disks (RAID)

- RAID stands for **R**edundant **A**rray of **I**ndependent **D**isks, which is a technology to connect multiple secondary storage devices and use them as a single storage media.

- RAID consists of an array of disks in which multiple disks are connected together to achieve different goals. RAID levels define the use of disk arrays.

RAID

- **RAID: Redundant Arrays of Independent Disks**
  - disk organization techniques that manage a large numbers of disks, providing a view of a single disk of
    - high capacity and high speed by using multiple disks in parallel,
    - high reliability by storing data redundantly, so that data can be recovered even if a disk fails
- Originally a cost-effective alternative to large, expensive disks
  - I in RAID originally stood for ``inexpensive''
  - Today RAIDs are used for their higher reliability and bandwidth.
    - The “I” is interpreted as independent

Improvement of Reliability via Redundancy

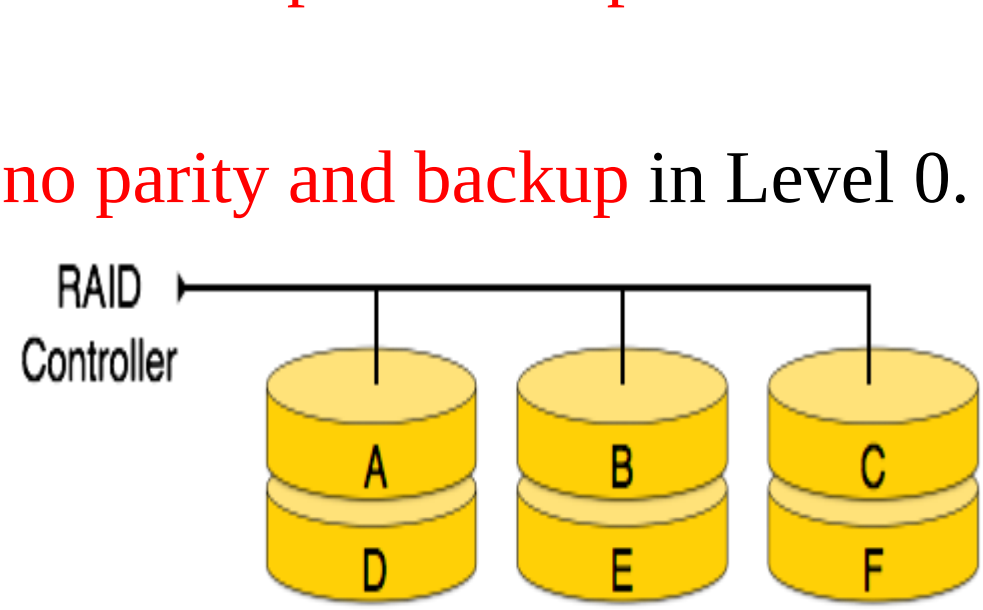
- **Redundancy** – store extra information that can be used to rebuild information lost in a disk failure
- E.g., **Mirroring** (or **shadowing**)
  - Duplicate every disk. Logical disk consists of two physical disks.
  - Every write is carried out on both disks
    - Reads can take place from either disk
  - If one disk in a pair fails, data still available in the other
- If **MTTF** is not long- disk failure result in loss of data -> **solution is redundancy**
- **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 is 100,000^2/(2*10)= $500*10^6$ hours (or 57,000 years) for a mirrored pair of disks

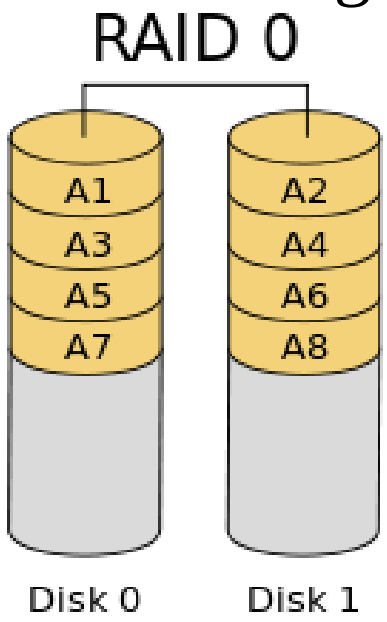
RAID Levels

- 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.

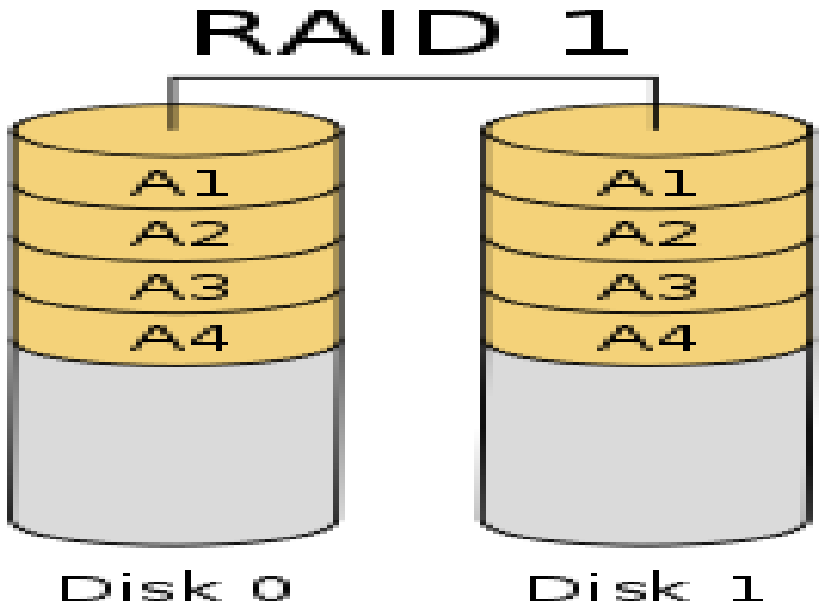
RAID Level 0: Block striping; non-redundant.
Used in high-performance applications where data loss is not critical.

- In this level, a striped array of disks is implemented.
- The data is broken down into **blocks** and the blocks are distributed among disks.
- Each disk receives a block of data to write/read in parallel.
- It enhances the speed and performance of the storage device.
- There is no parity and backup in Level 0.





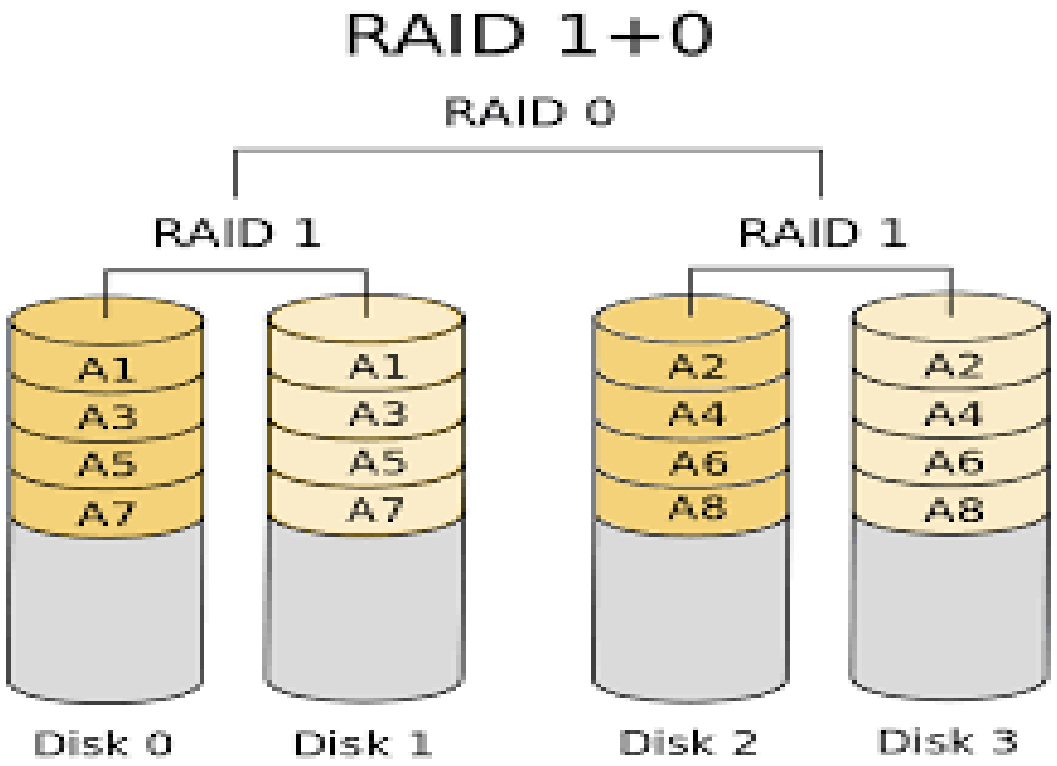
- **RAID Level 1**: Mirrored disks
  - is the replication of data to two or more disks
  - RAID level 1 is also called **mirroring** and provides **100% redundancy in case of a failure.**
  - Popular for applications such as storing **log files** in a database system.



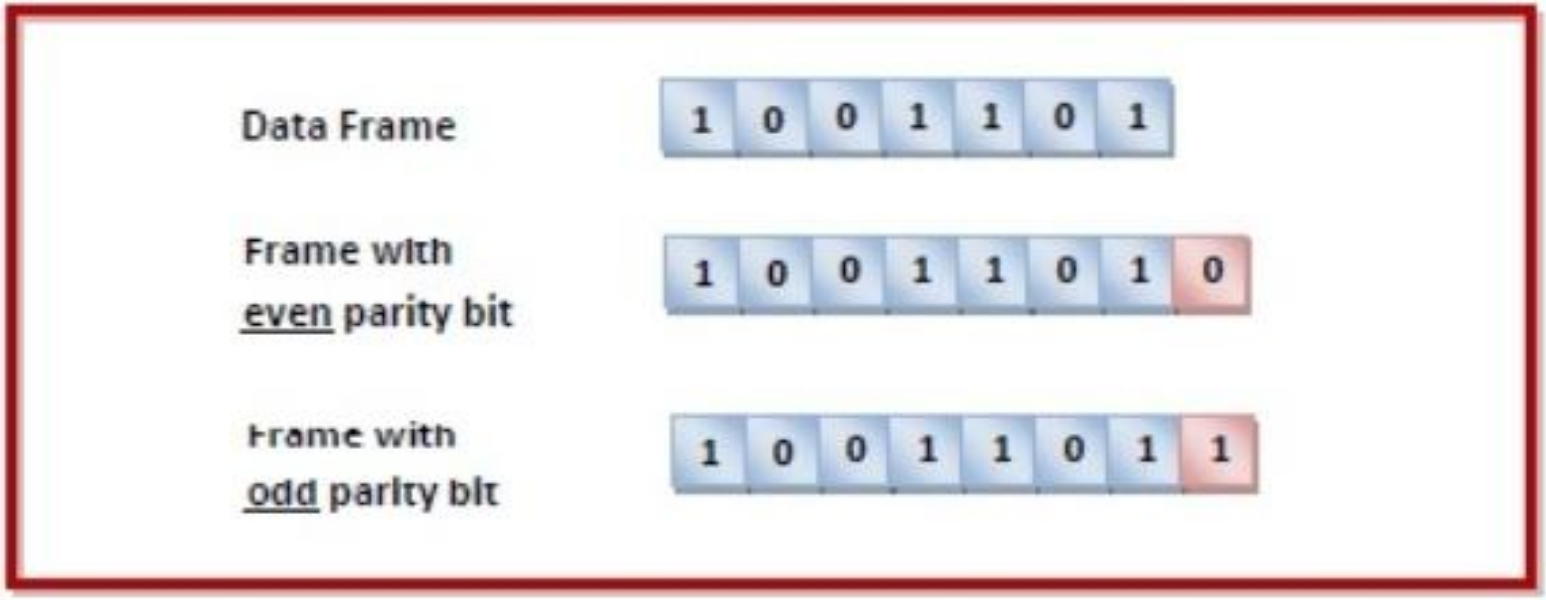
RAID Levels (Cont.)

RAID Level 10 (1+0)

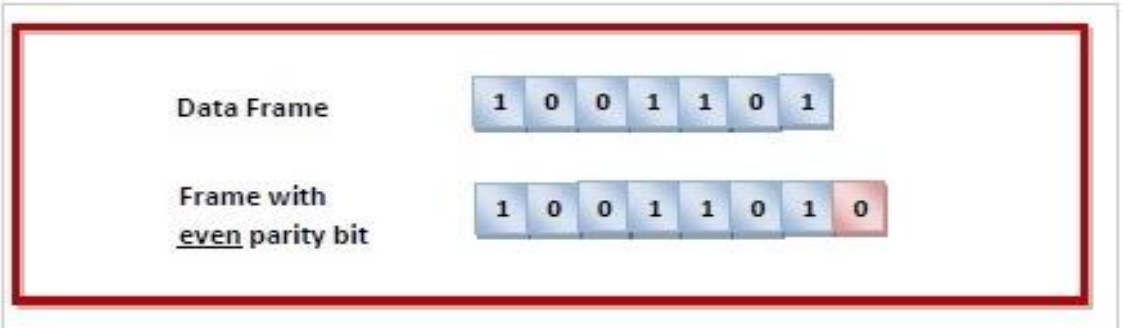
RAID 10, also known as **RAID** 1+0, is a **RAID** configuration that combines disk mirroring and disk striping to protect data.

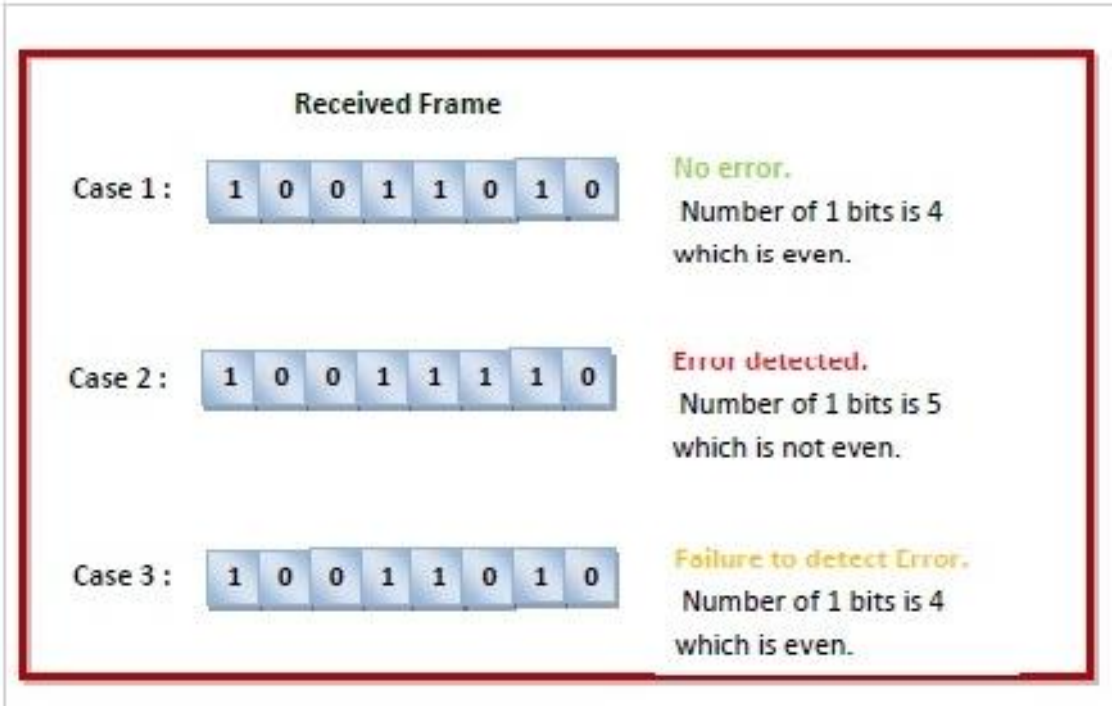


Error Detection by parity check



Suppose that a sender wants to send the data 1001101 using even parity check method. It will add the parity bit as shown below.





Let us find the Hamming code for binary code, $d_4d_3d_2d_1$ = 1000. Consider even parity bits.

The number of bits in the given binary code is n=4.

We can find the required number of parity bits by using the following mathematical relation.

$$2^k \geq n + k + 1$$

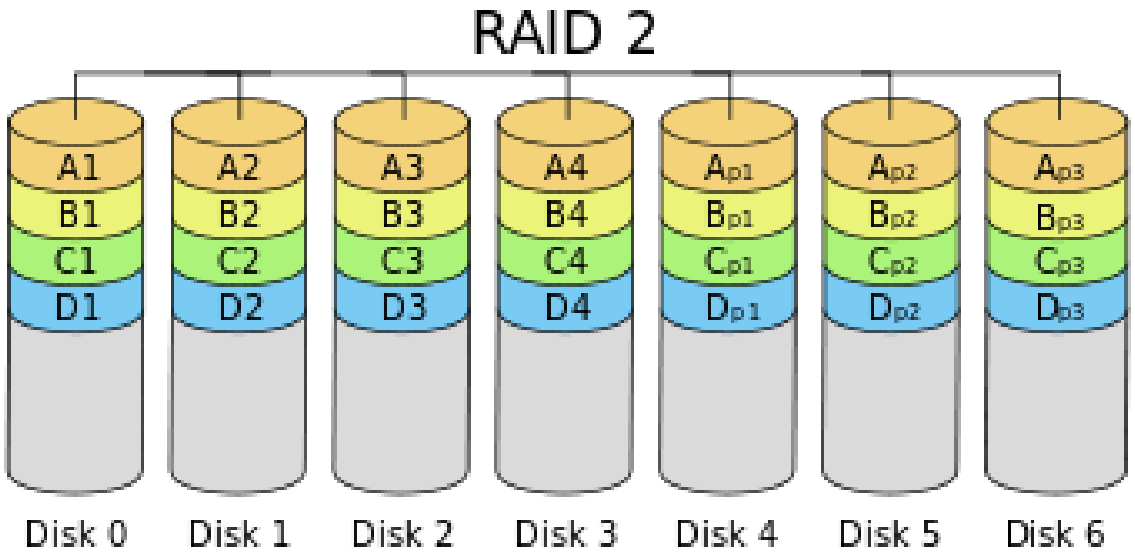
Substitute, n=4 in the above mathematical relation.

$$\Rightarrow 2^k \geq 4 + k + 1$$

$$\Rightarrow 2^k \geq 5 + k$$

RAID Levels (Cont.)

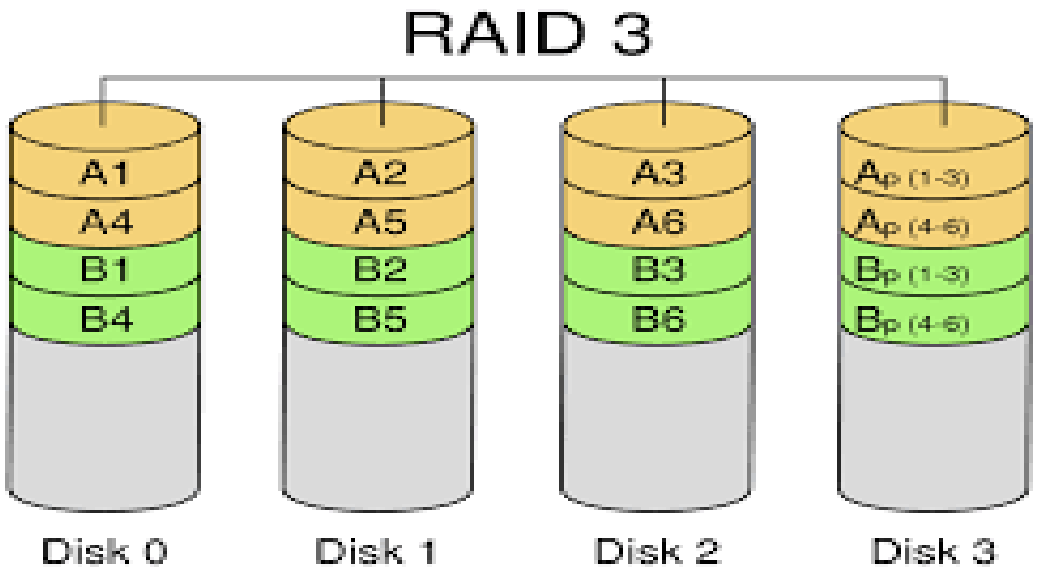
- **RAID Level 2**: Memory-Style Error-Correcting-Codes (ECC) with bit striping.
  - Use 2 or more extra bits, and can reconstruct the data if a single bits gets damaged



RAID Levels (Cont.)

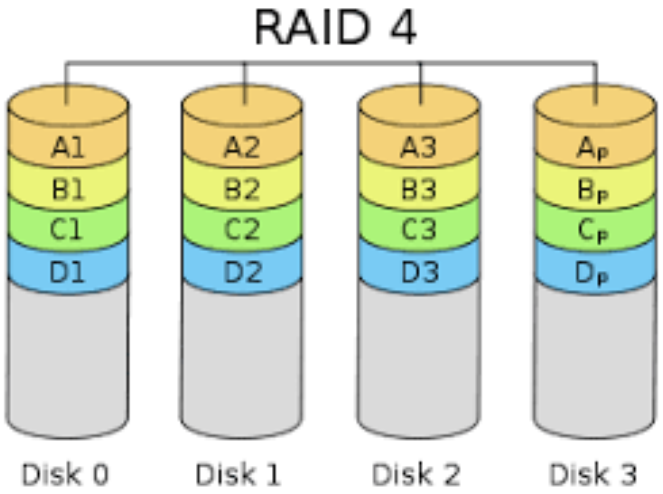
RAID Level 3: Bit-Interleaved Parity

- a single parity bit is enough for error correction, not just detection, since we know which disk has failed done by controller
- If Disk sector is failed the system will identify which sector it is and each bit in the sector. later it can be replaced.
  - 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.
- Subsume



RAID Levels (Cont.)

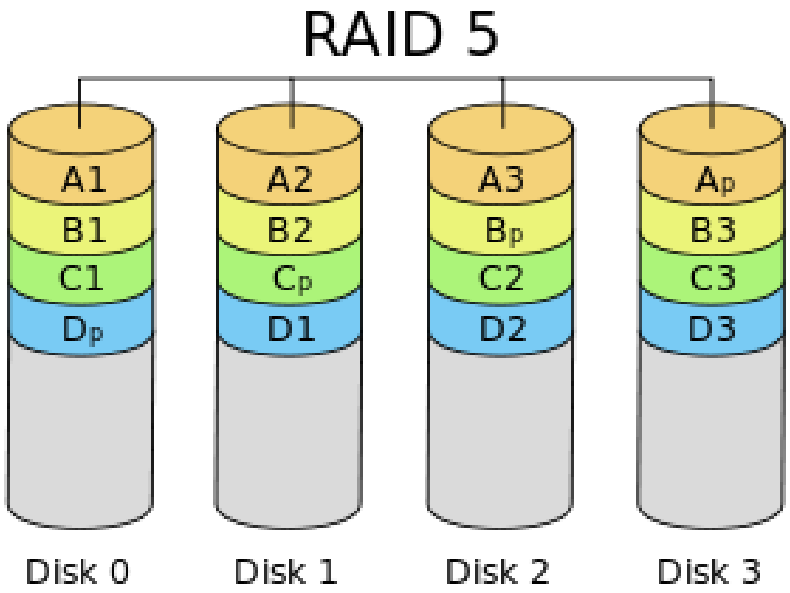
- **RAID Level 4:** Block-Interleaved Parity; uses block-level striping, and keeps a parity block on a separate disk for corresponding blocks from *N* other disks.
  - Provides higher I/O rates for independent block reads than Level 3
    - block read goes to a single disk, so blocks stored on different disks can be read in parallel
  - Before writing a block, parity data must be computed



RAID Levels (Cont.)

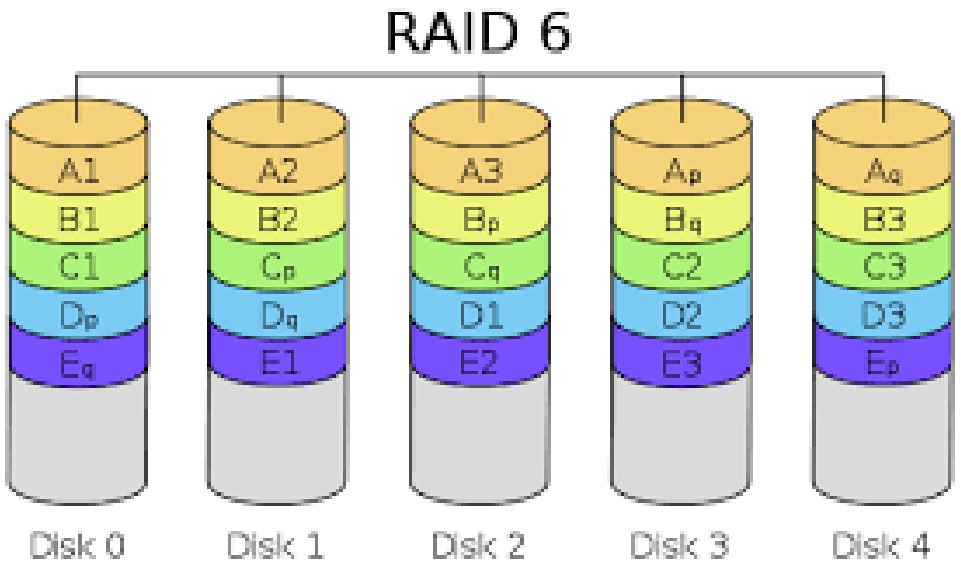
- **RAID Level 5:** Block-Interleaved Distributed Parity; partitions data and parity among all $N + 1$ disks, rather than storing data in N disks and parity in 1 disk.
  - E.g., with 5 disks, parity block for nth set of blocks is stored on disk (n *mod* 5) + 1, with the data blocks stored on the other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

- Higher I/O rates than Level 4.
- Block writes occur in parallel if the blocks and their parity blocks are on different disks.
- Subsumes Level 4: provides same benefits, but avoids bottleneck of parity disk.

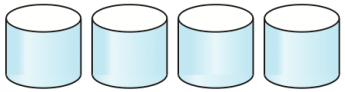


RAID Levels (Cont.)

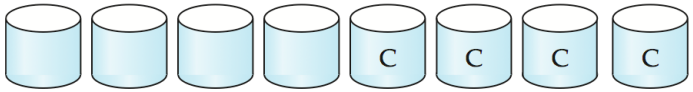
- **RAID Level 6**: P+Q Redundancy scheme; similar to Level 5, but stores extra redundant information to guard against multiple disk failures.
  - Better reliability than Level 5 at a higher cost; not used as widely.



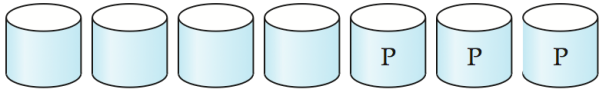
RAID Levels



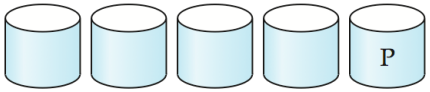
(a) RAID 0: nonredundant striping



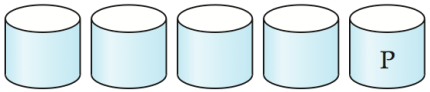
(b) RAID 1: mirrored disks



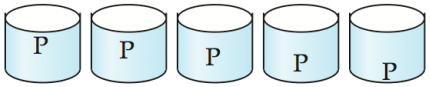
(c) RAID 2: memory-style error-correcting codes



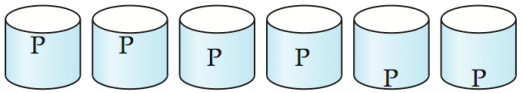
(d) RAID 3: bit-interleaved parity



(e) RAID 4: block-interleaved parity



(f) RAID 5: block-interleaved distributed parity



(g) RAID 6: P + Q redundancy

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
- Level 2 and 4 never used since they are subsumed by 3 and 5
- Level 3 is not used anymore since bit-striping forces single block reads to access all disks, wasting disk arm movement, which block striping (level 5) avoids
- Level 6 is rarely used since levels 1 and 5 offer adequate safety for most applications

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 preferred for high update environments such as log disks
- Level 1 had higher storage cost than level 5
  - disk drive capacities increasing rapidly (50%/year) whereas disk access times have decreased much less (x 3 in 10 years)
  - I/O requirements have increased greatly, e.g. for Web servers
  - When enough disks have been bought to satisfy required rate of I/O, they often have spare storage capacity
    - so there is often no extra monetary cost for Level 1!
- Level 5 is preferred for applications with low update rate, and large amounts of data
- Level 1 is preferred for all other applications

File Organization

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.
- Records are mapped onto disk blocks.
- Each file is logically partitioned into fixed-length storage unit called blocks. (4 to 8KB)
- A block may contain several records
- The exact count of records that the block contains is determined by the form of physical data organization being used.
- Assume that no record is larger than a block.(Some cases its not true i.e images)
- Each record is entirely in single block
- Approaches:
  - Fixed length records
  - Variable length records

ID	ENAME	SALARY	BONUS	DEPT
1	James Potter	75000	1000	ICP
2	Ethan McCarty	90000	2000	ETA
3	Emily Rayner	25000		ETA
4	Jack Abraham	30000	1000	ETA

Relation is usually represented as:
Employee(ID, ENAME, SALARY, BONUS, DEPT)

Fixed-Length Records

- Simple approach:
  - assume record size is fixed
  - each file has records of one particular type only
  - different files are used for different relations
  - Instructor record-53 bytes

type instructor = ***record***
ID ***varchar*** *(5);*
name ***varchar****(20);*
dept name ***varchar*** *(20);*
salary ***numeric*** *(8,2);*
end

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

Disadvantages

- 2 Problems with Fixed length Record:

- i) If the block size is less than the record size in the Disk. We need two block access to be done

- Example: If the record size is 53 bytes and the disk block size is 40 then **2 blocks** are needed to store.

- ii) Difficult to **delete** the record from this structure
  - i.e i) will be deleted record must be filled with some other record
  - ii) marking deleted record that can be ignored.

Solution

- i) Allocate only as many records to block would fit entirely in the block(Computed by dividing the block size by record size), and discard the fractional part. Any remaining bytes of each block are left unused.

- ii) for second problem, we would move the record that came after it into the space formerly occupied by the deleted record.

Deleting record 3 and compacting

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

- It is undesirable to move records to occupy the space freed by a deleted record. (Additional block access)

Solution:

- It is recommeded to leave the space occupied by deleted record (Since insertion tend to be more frequent than deletion in disk).

- But simple marker is not enough for free space identification.

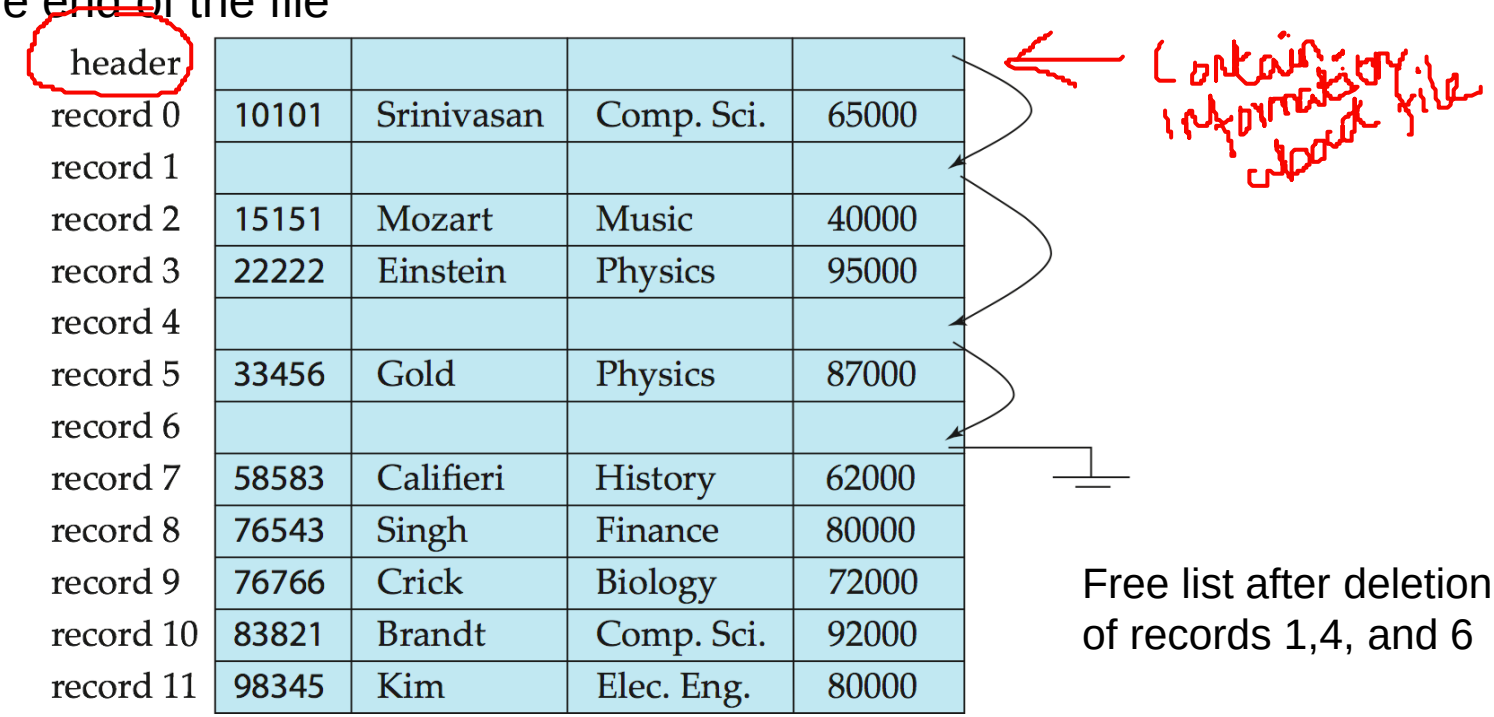
Deleting record 3 and moving last record

move record *n* to *i*

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

Free Lists

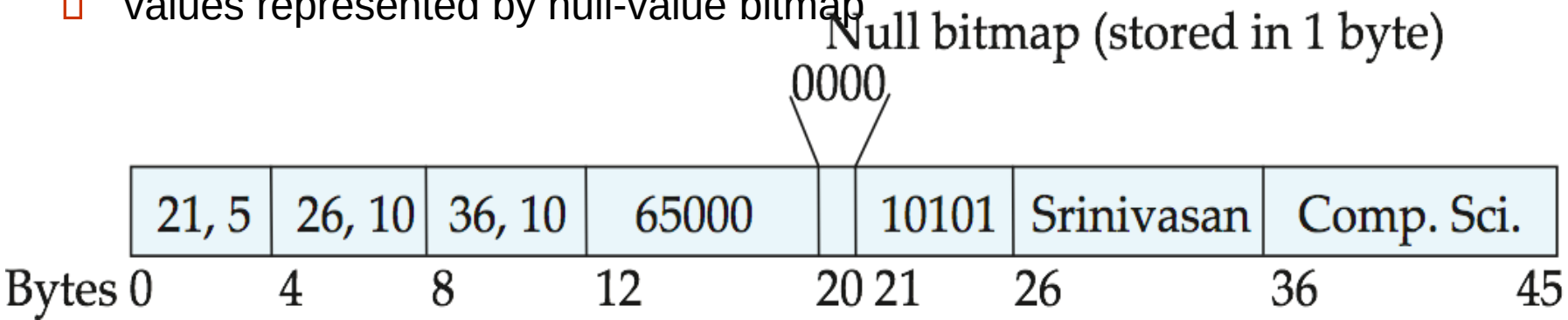
- Store the address of the first deleted record in the file header.
- Use this first record to store the address of the second deleted record, and so on
- Can think of these stored addresses as pointers since they “point” to the location of a record.(**Linked List called free list**)
- **Insertion of new record** - use the pointer in file header and update the pointer. if no space available i.e no pointer then new record is add to the end of the file



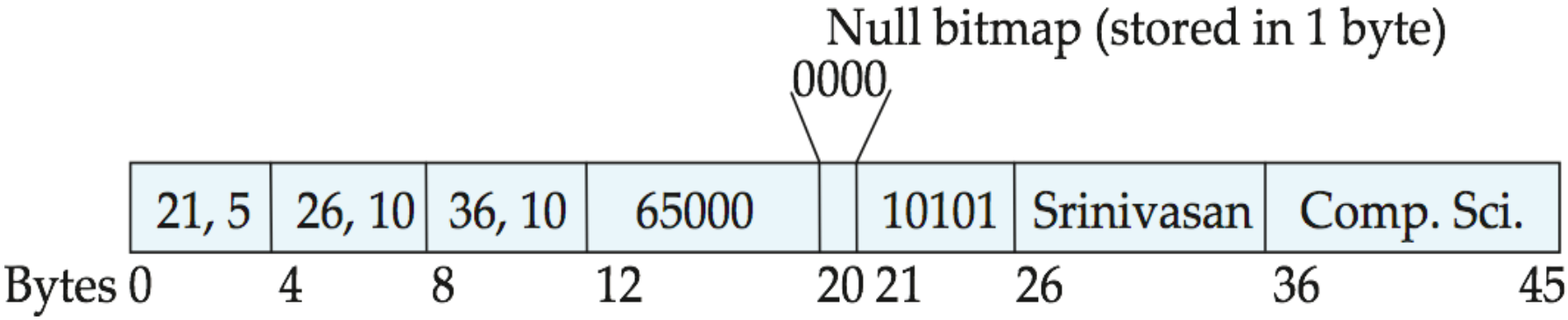
Free list after deletion of records 1,4, and 6

Variable-Length Records

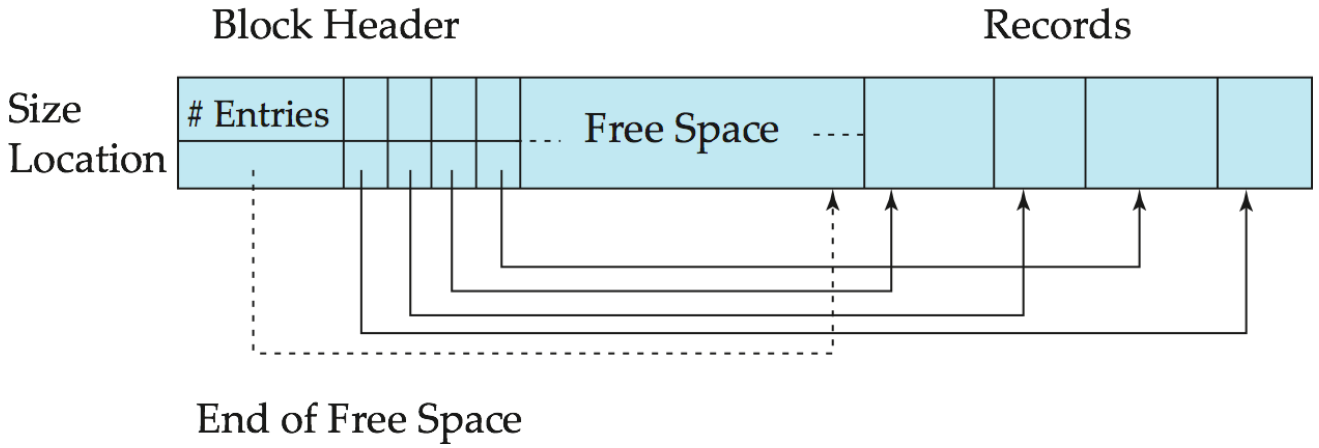
- Variable-length records arise in database systems in several ways:
  - Storage of multiple record types in a file.
  - Record types that allow variable lengths for one or more fields such as strings (**varchar**)
  - Record types that allow repeating fields (used in some older data models).
- **Two parts**: An initial part with fixed length attributes, followed by data for variable-length attributes
- Fixed length attributes such as numeric values, dates. (allocate as many bytes required)
- Variable length attributes represented by fixed size (offset-where the data begins, length-in bytes for attributes)
- values represented by null-value bitmap



- Let assume offset and length values are stored in two bytes each.
- ID, name and dept_name are variable length strings.
- Salary is fixed length store in two bytes.
- Null bitmap- used to indicate null values. if salary is NULL, then fourth bit would be set to 1. So the bytes from 12 to 19 would be ignored.



Variable-Length Records: Slotted Page Structure



- **Slotted page used for organizing records within a block.**
- 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.
- Large objects(blob,clob) are stored in special file –pointer is stored in record

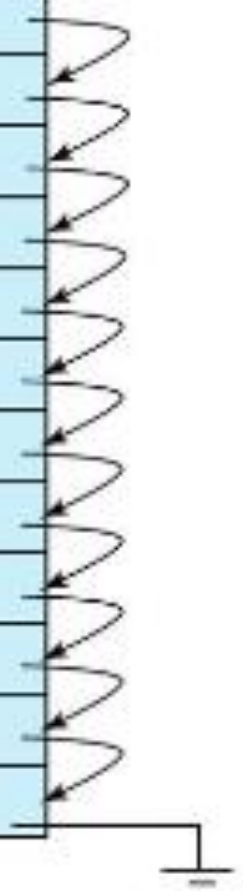
Organization of Records in Files

- **Heap** – a record can be placed anywhere in the file where there is space. There is no ordering of records.
- **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

1. Sequential File Organization

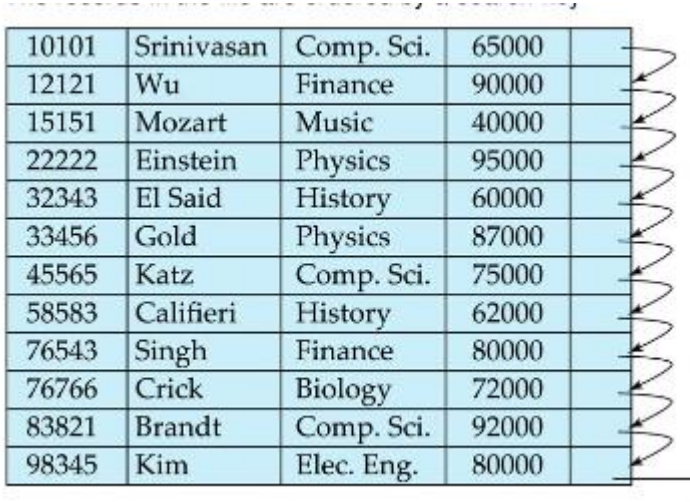
- **Sequential file** – is designed for efficient processing of records in sorted order based on search key.
- A search key is any attribute or set of attributes.
- It need not be the primary key or even a super key.

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

- The pointer is used for fast retrival of records.
- The pointer in each record points to the next record in search- key order.
- To minimize the number of block accesses, we store records physically in search-key order or close to search-key order as possible.
- Here, the image uses ID as 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	

Limitations

- However, it is costly to move many records as a results of single insertion and deletion.
- Deletion - Uses pointer chain as we discussed earlier.

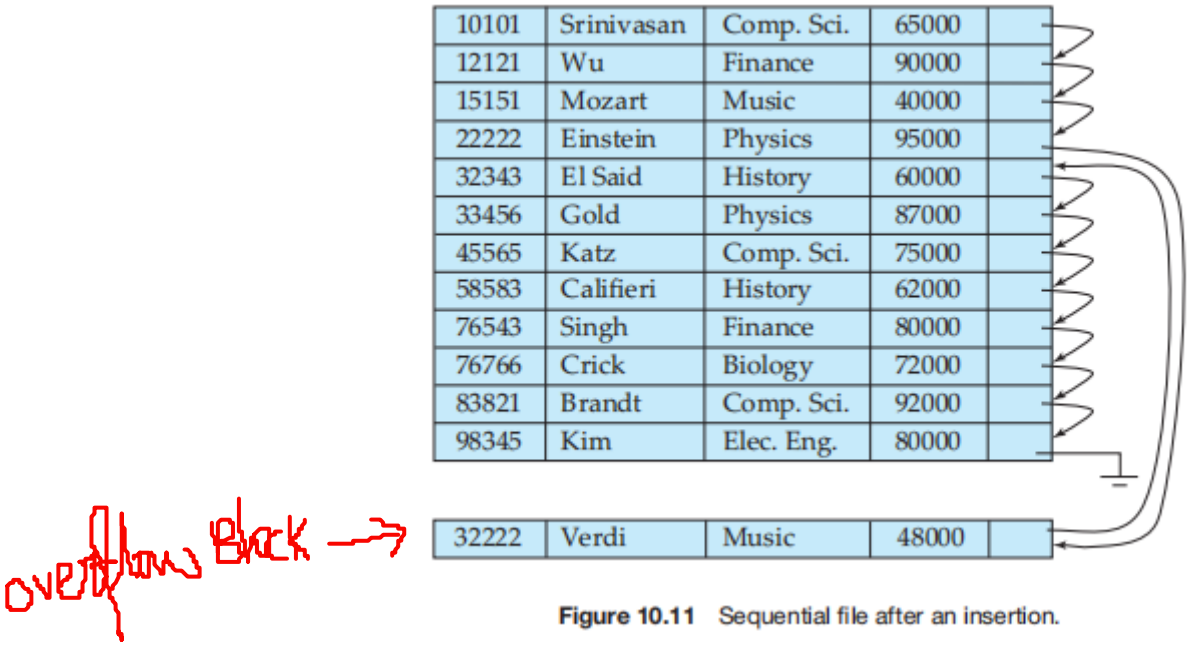


Figure 10.11 Sequential file after an insertion.

Sequential File Organization

- Deletion – use pointer chains
- Insertion – must locate the position in the file where the record is to be record
  - 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

2. Multitable Clustering File Organization

- Many relational database systems store each relation in a separate file, so that they
- can take full advantage of the file system that the operating system provides.

- Tuples of a relation can be represented as fixed-length records.

- low-cost database implementations.
- Example, Embedded systems or Portable devices

Low Cost

- Size of the database is small, so little is gained from a sophisticated file structure.
- Furthermore, in such environments, it is essential that the overall size of the object code for the database system be small.
- A simple file structure reduces the amount of code needed to implement the system

2. Multitable Clustering File Organization

- This simple approach to relational database implementation becomes less satisfactory as the size of the database increases.

- Instead, one large operating system file is allocated to the database system. The database system stores all relations in this one file, and manages the file itself.

- Even if multiple relations are stored in a single file, by default most databases store records of only one relation in a given block.

2. Multitable Clustering File Organization

- However, in some cases it can be useful to store records of more than one relation in a single block. To see the advantage of storing records of multiple relations in one block.

2. Multitable Clustering File Organization

```
select dept_name, building, budget, ID, name, salary
from department natural join instructor;
```

Ideally, these records will be located with the help of indices, which we shall discuss in next class.

Regardless of how these records are located, however, they need to be transferred from disk into main memory. In the worst case, each record will reside on a different block, forcing us to do one block read for each record required by the query

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Physics	Watson	70000

Figure 10.12 The *department* relation.

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

Figure 10.13 The *instructor* relation.

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

Figure 10.14 Multitable clustering file structure.

The instructor tuples for each ID are stored near the department tuple for the corresponding dept name. This structure mixes together tuples of two relations, but allows for efficient processing of the join.

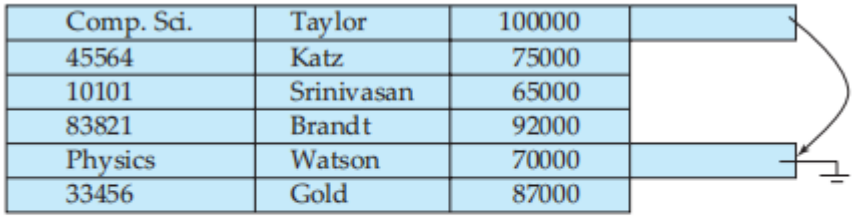


Figure 10.15 Multitable clustering file structure with pointer chains.

select *
from *department;*

2. Multitable Clustering File Organization