COP 4710: Database Systems
Fall 2007

Chapter 9 – Data Storage

Instructor :   Dr. Mark Llewellyn
                      markl@cs.ucf.edu
                      HEC 236, 407-823-2790
                      http://www.cs.ucf.edu/courses/cop4710/fall2007

School of Electrical Engineering and Computer Science
University of Central Florida

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.

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 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
  - Cost per unit of storage roughly similar to main memory
  - Widely used in embedded devices such as digital cameras
  - also known as EEPROM (Electrically Erasable Programmable Read-Only Memory)

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
  - Hard disks vs floppy disks
  - Capacities range up to roughly 300+ GB currently
    - 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 very 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

  - Write-one, read-many (WORM) optical disks used for archival storage (CD-R and DVD-R)

  - Multiple write versions also available (CD-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 a petabyte (1 petabyte = 10^{12} bytes)

Storage Hierarchy
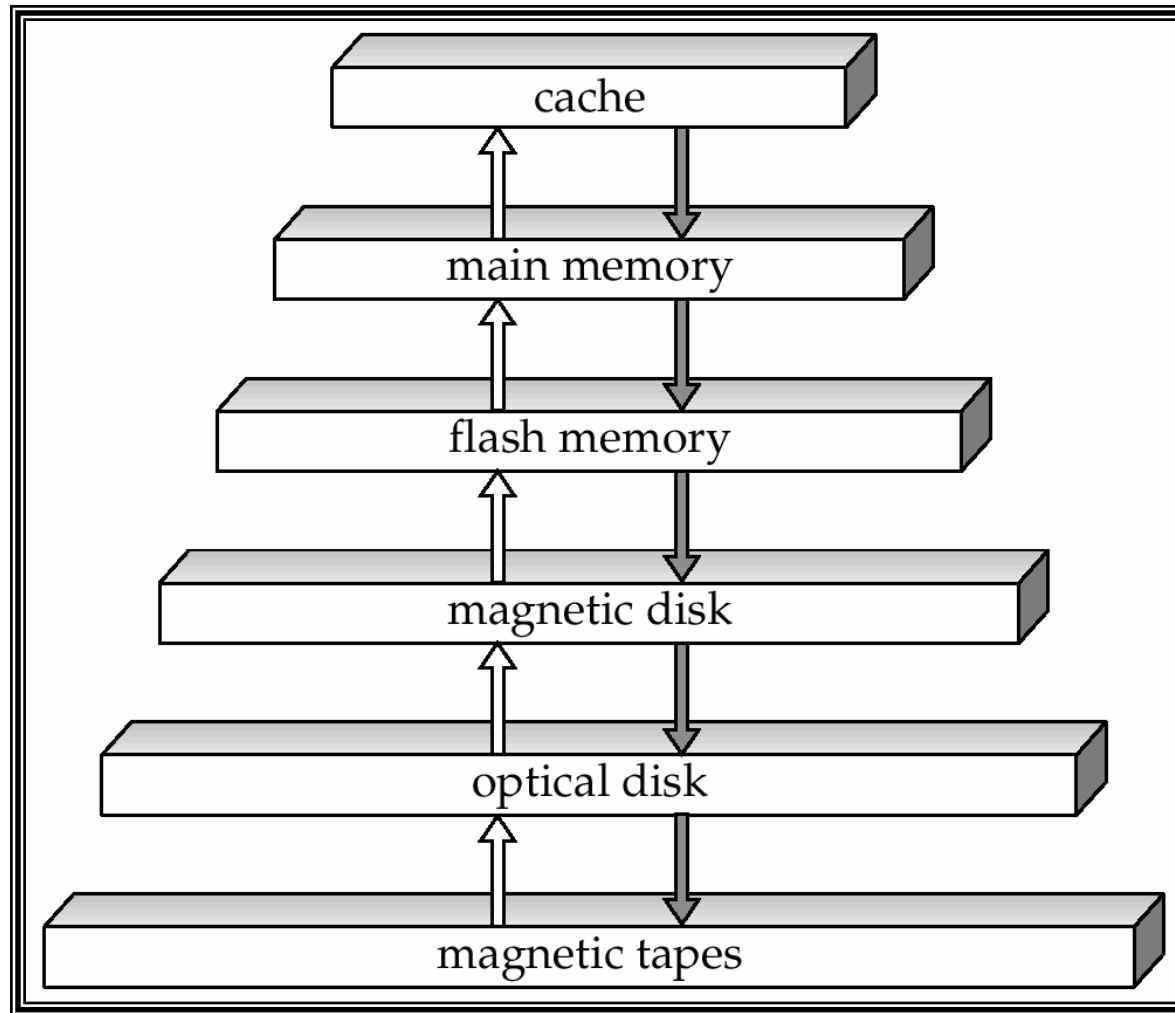
Storage Hierarchy (cont.)

- **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

  - examples: flash memory, magnetic disks

- **tertiary storage:** lowest level in hierarchy, non-volatile, slow access time

  - also called off-line storage

  - examples: magnetic tape, optical storage

Magnetic Hard Disk Mechanism
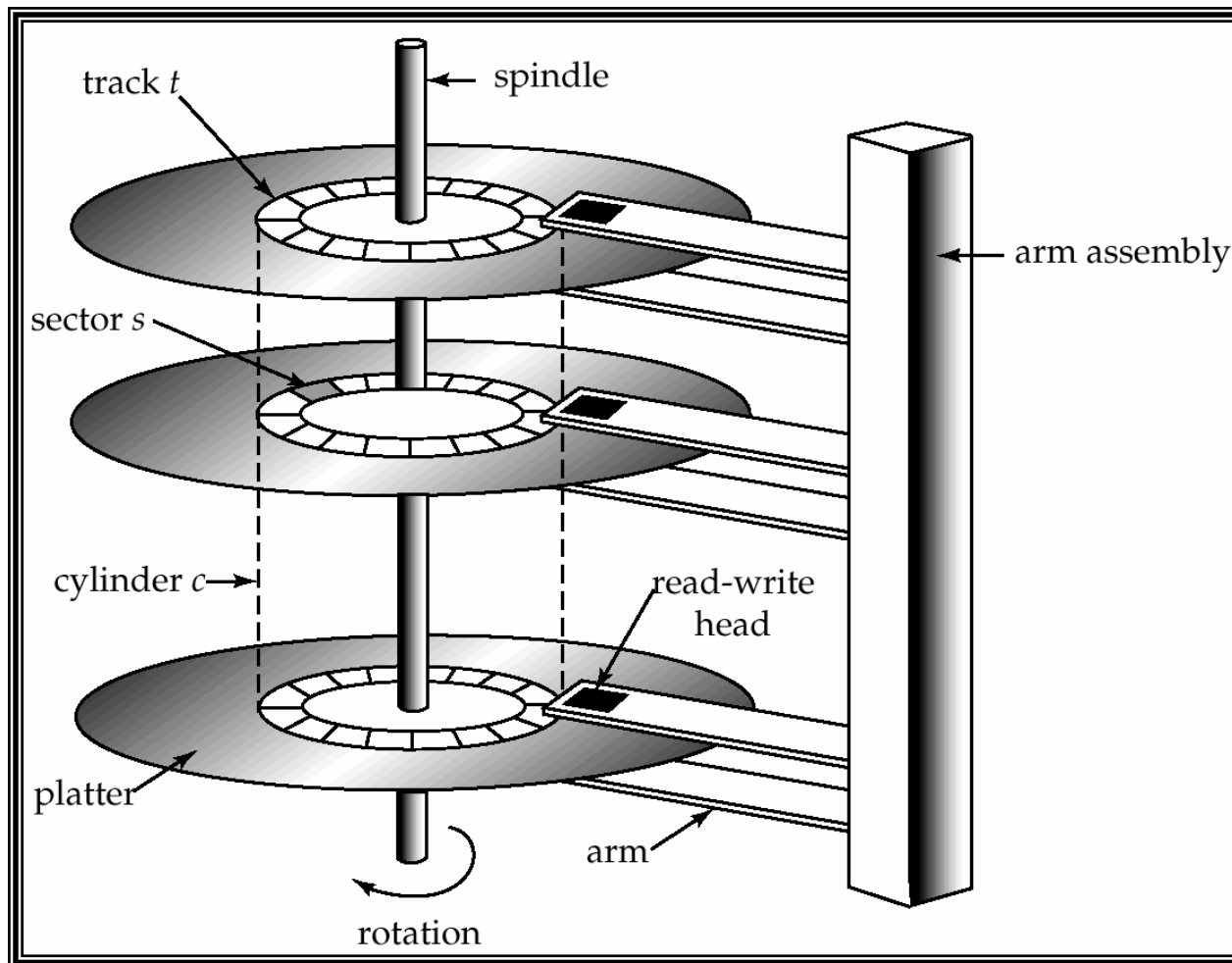
Magnetic Disks

- **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 16,000 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: 200 (on inner tracks) to 400 (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 (typically 2 to 4)
  - one head per platter, mounted on a common arm.
- Cylinder i consists of ith 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
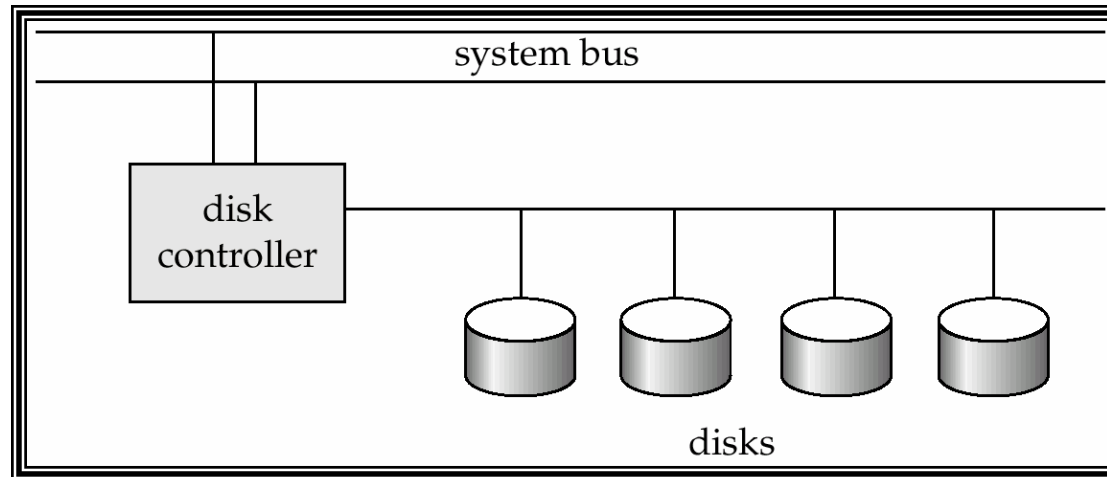
- Multiple disks connected to a computer system through a controller
  - Controllers functionality (checksum, bad sector remapping) often carried out by individual disks; reduces load on controller
- Disk interface standards families
  - ATA (AT adaptor) range of standards
  - SCSI (Small Computer System Interconnect) range of standards
  - 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
  - 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.)

Performance Measures of Disks (cont.)

- **Data-transfer rate** – the rate at which data can be retrieved from or stored to the disk.

  - 4 to 8 MB per second is typical

  - Multiple disks may share a controller, so rate that controller can handle is also important

    - E.g. ATA-5: 66 MB/second, SCSI-3: 40 MB/s

    - Fiber Channel: 256 MB/s

Performance Measures of Disks (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 30,000 to 1,200,000 hours for a new disk

    - E.g., an MTTF of 1,200,000 hours for a new disk means that given 1000 relatively new disks, on an average one will fail every 1200 hours

    – MTTF decreases as disk ages.

Optimization of Disk Block Access

- **Block** – a contiguous sequence of sectors from a single track
  - data is transferred between disk and main memory in blocks
  - sizes range from 512 bytes to several kilobytes
    - Smaller blocks: more transfers from disk
    - Larger blocks: more space wasted due to partially filled blocks
    - Typical block sizes today range from 4 to 16 kilobytes

- **Disk-arm-scheduling** algorithms order pending accesses to tracks so that disk arm movement is minimized

  - elevator algorithm : move disk arm in one direction (from outer to inner tracks or vice versa), processing next request in that direction, till no more requests in that direction, then reverse direction and repeat

Optimization of Disk Block Access (cont.)

- **File organization** – 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
    - E.g. if data is inserted to/deleted from the file
    - Or 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.

Optimization of Disk Block Access (cont.)

- **Nonvolatile write buffers** speed up disk writes by writing blocks to a non-volatile RAM buffer immediately
  - Non-volatile RAM: battery backed up RAM or flash memory
    - Even if power fails, the data is safe and will be written to disk when power returns
  - Controller then writes to disk whenever the disk has no other requests or request has been pending for some time
  - Database operations that require data to be safely stored before continuing can continue without waiting for data to be written to disk
  - *Writes can be reordered to minimize disk arm movement*
- **Log disk** – a disk devoted to writing a sequential log of block updates
  - Used exactly like nonvolatile RAM
    - Write to log disk is very fast since no seeks are required
    - No need for special hardware (NV-RAM)
- File systems typically reorder writes to disk to improve performance
  - Journaling file systems write data in safe order to NV-RAM or log disk
  - Reordering without journaling: risk of corruption of file system data
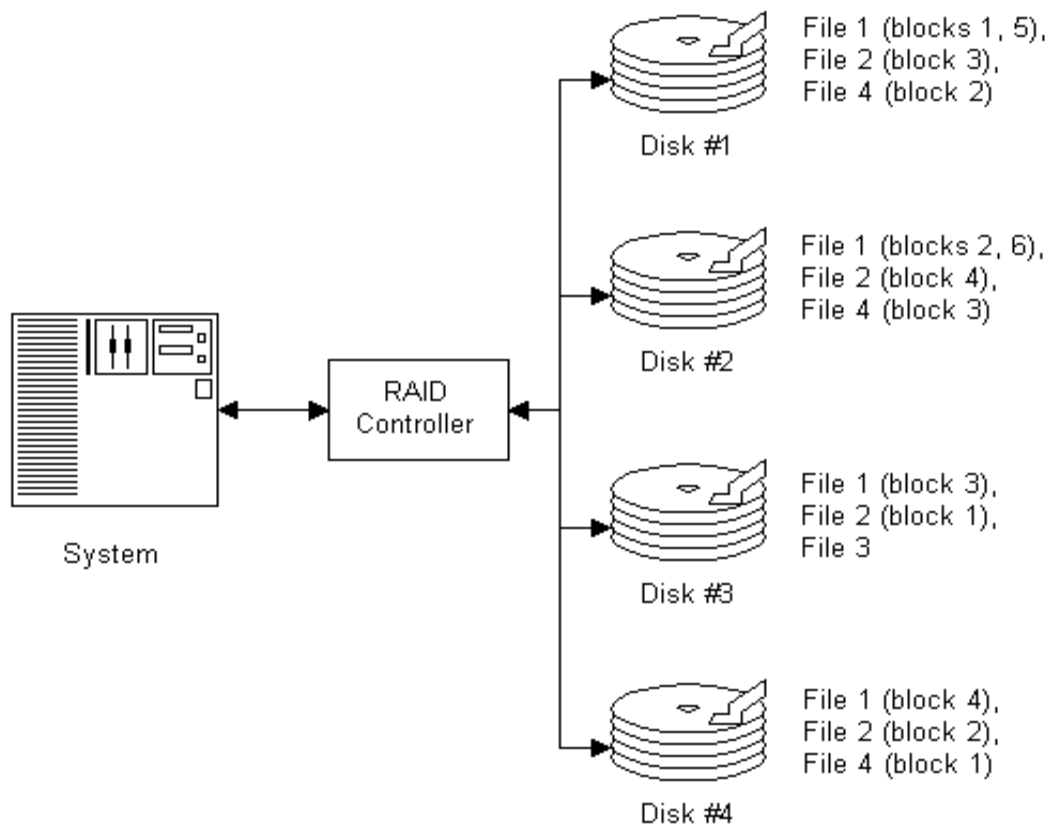
RAID

- **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, and

    - 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

RAID (cont.)

File 1 (blocks 1, 5),
File 2 (block 3),
File 4 (block 2)

Disk #1

File 1 (blocks 2, 6),
File 2 (block 4),
File 4 (block 3)

Disk #2

RAID
Controller

System

File 1 (block 3),
File 2 (block 1),
File 3

Disk #3

File 1 (block 4),
File 2 (block 2),
File 4 (block 1)

Disk #4

Block diagram of a RAID striping configuration. One controller (which can be hardware or software) splits files into blocks or bytes and distributes them across several hard disks. The block size determines how many "pieces" files will be split into. In this example, the first block of file 1 is sent to disk #1, then the second block to disk #2, etc. When all four disks have one block of file 1, the fifth block goes back to disk #1, and this continues until the file is completed. Note that file 3 is only on one disk; this means it was smaller than the block size in this case.

RAID (cont.)

- Originally a cost-effective alternative to large, expensive disks

  – The "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 Through Redundancy

- **Redundancy** – store extra information that can be used to rebuild information lost in a disk failure
- **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 of Reliability Through Parallelism

- Two main goals of parallelism in a disk system:
    1. Load balance multiple small accesses to increase throughput
    2. Parallelize large accesses to reduce response time.
- Improve transfer rate by striping data across multiple disks.
- **Bit-level striping** – split the bits of each byte across multiple disks
    - In an array of eight disks, write bit i of each byte to disk i.
    - Each access can read data at eight times the rate of a single disk.
    - But seek/access time worse than for a single disk
        - Bit level striping is not used much any more
- **Block-level striping** – with n disks, block i of a file goes to disk (i mod n) + 1
    - Requests for different blocks can run in parallel if the blocks reside on different disks
    - A request for a long sequence of blocks can utilize all disks in parallel
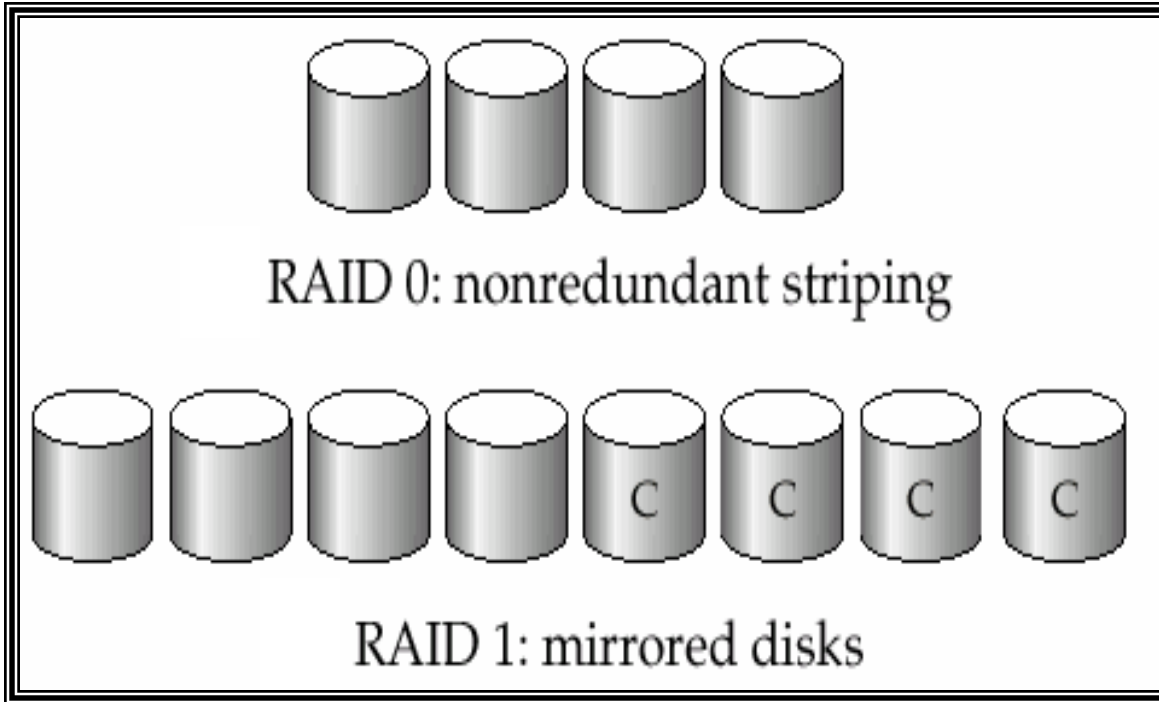
RAID Levels

- 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 lost 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.

RAID Levels (cont.)

RAID 0: nonredundant striping
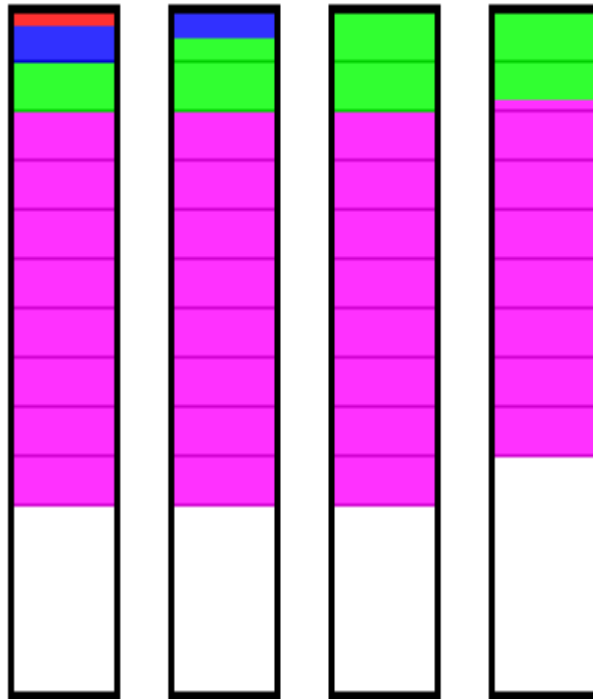
RAID 1: mirrored disks

RAID Levels (cont.)

This illustration shows how files of different sizes are distributed between the drives on a four-disk, 16 kB stripe size RAID 0 array. The red file is 4 kB in size; the blue is 20 kB; the green is 100 kB; and the magenta is 500 kB. They are shown *drawn to scale* to illustrate how much space they take up in relative terms in the array--one vertical pixel represents 1 kB. Thus, the 500 kB files needs a total of 31+ stripes, the 100 kB file needs 6+ stripes.

Raid 0

RAID Levels (cont.)
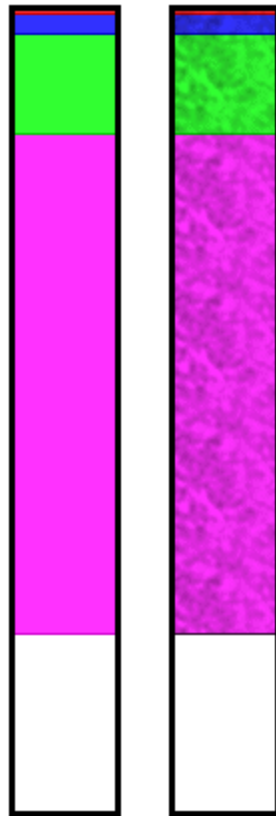
Illustration of a pair of mirrored hard disks, showing how the files are duplicated on both drives. (The files are the same as those in the RAID 0 illustration on the previous page, except that the scale is reduced here so one vertical pixel represents 2 kB.)

Raid 1

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).
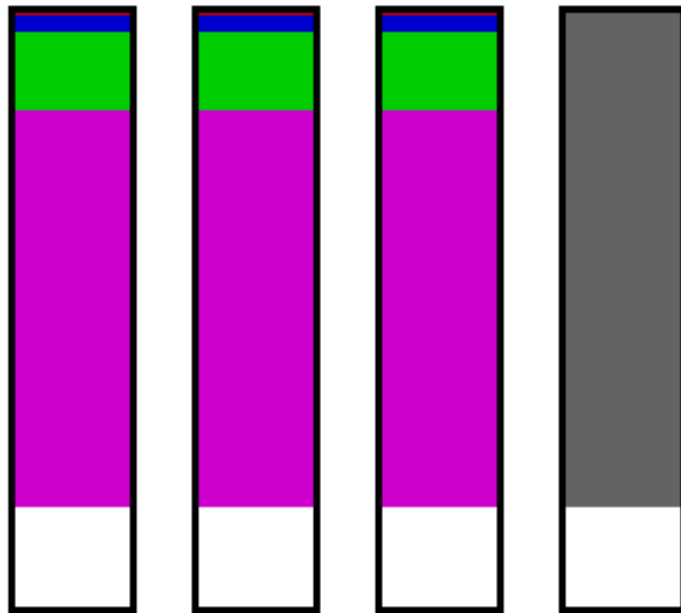
RAID Levels (cont.)
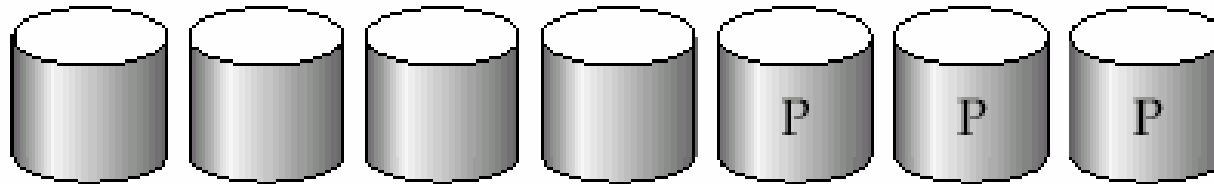
RAID 2: memory-style error-correcting codes
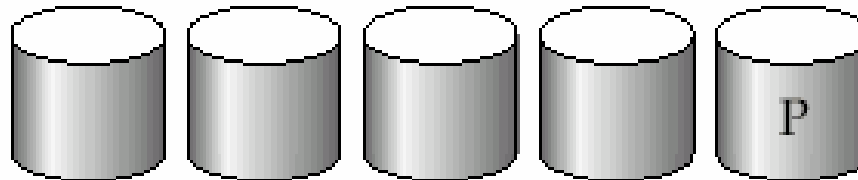
RAID 3: bit-interleaved parity

RAID Levels (cont.)

This illustration shows how files of different sizes are distributed between the drives on a four-disk, byte-striped RAID 3 array. As with the RAID 0 illustration, the red file is 4 kB in size; the blue is 20 kB; the green is 100 kB; and the magenta is 500 kB, with each vertical pixel representing 1 kB of space. Notice that the files are evenly spread between three drives, with the fourth containing parity information (shown in dark gray). Since the blocks are so tiny in RAID 3, the individual boundaries between stripes can't be seen. You may want to compare this illustration to the one for RAID 4 on page 34.

Raid 3

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
    - Can be done by using old parity block, old value of current block and new value of current block (2 block reads + 2 block writes)
    - Or by recomputing the parity value using the new values of blocks corresponding to the parity block
      - More efficient for writing large amounts of data sequentially
  - Parity block becomes a bottleneck for independent block writes since every block write also writes to parity disk.
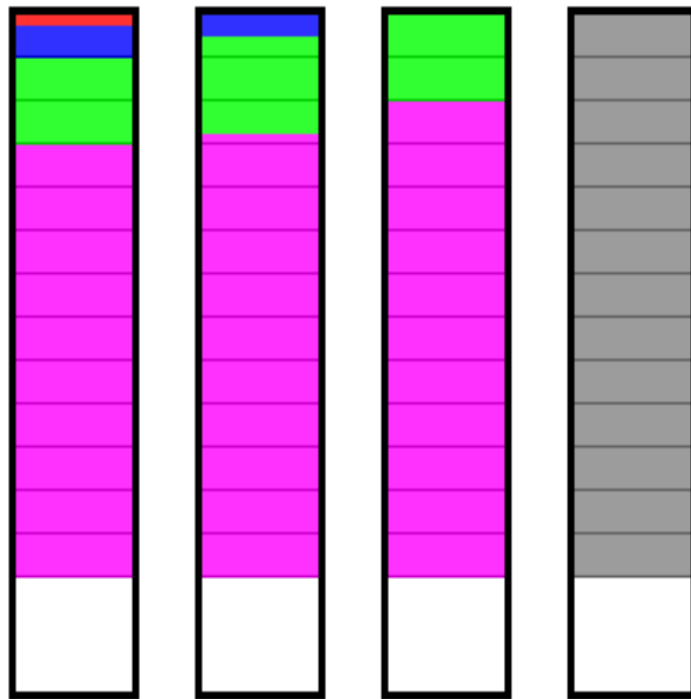
RAID Levels (cont.)
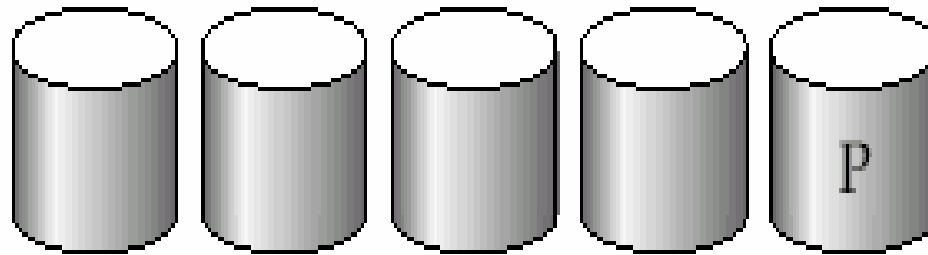
RAID 4: block-interleaved parity

RAID Levels (cont.)

This illustration shows how files of different sizes are distributed between the drives on a four-disk RAID 4 array using a 16 kB stripe size. As with the RAID 0 illustration, the red file is 4 kB in size; the blue is 20 kB; the green is 100 kB; and the magenta is 500 kB, with each vertical pixel representing 1 kB of space. Notice that as with RAID 3, the files are evenly spread between three drives, with the fourth containing parity information (shown in gray). You may want to contrast this illustration to the one for RAID 3 (which is very similar except that the blocks are so tiny you can't see them) and the one

Raid 4

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 $(n \bmod 5) + 1$, 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.

RAID Levels (cont.)
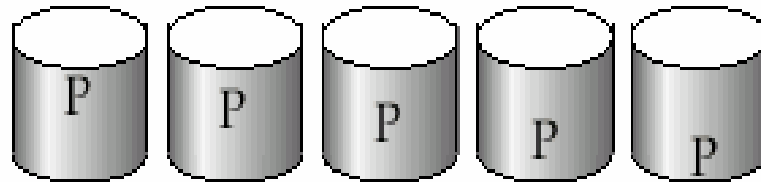
(f) RAID 5: block-interleaved distributed parity

RAID Levels (cont.)
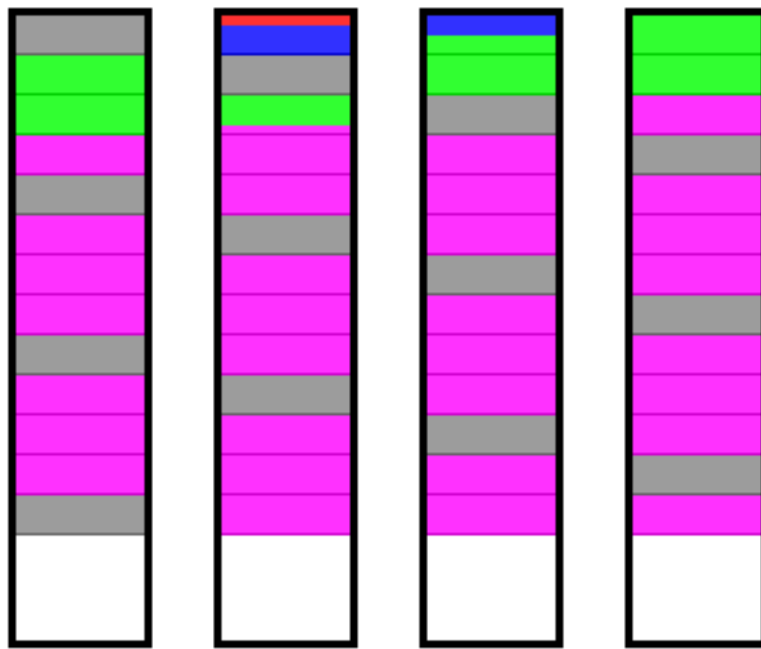
Raid 5

This illustration shows how files of different sizes are distributed between the drives on a four-disk RAID 5 array using a 16 kB stripe size. As with the RAID 0 illustration, the red file is 4 kB in size; the blue is 20 kB; the green is 100 kB; and the magenta is 500 kB, with each vertical pixel representing 1 kB of space. Contrast this diagram to the one for RAID 4, which is identical except that the data is only on three drives and the parity (shown in gray) is exclusively on the fourth drive.

RAID Level 5 Example

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

Assume an array of 5 disks.

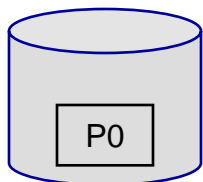The parity block, labeled Pk, for logical blocks 4k, 4k+1, 4k+2, and 4k+3 is stored in disk (k mod 5)+1

The corresponding blocks of the other four disks store the 4 data blocks 4k to 4k+3.

The table on the left illustrates how the first 20 blocks, numbered 0 to 19, and their parity blocks are laid out in the disk array.

Example: parity block P0, for logical blocks 0, 1, 2, and 3, is stored in disk (0 mod 5)+1 = 1

parity block P1, for logical blocks 4, 5, 6, and 7, is stored in disk (1 mod 5)+1 = 2

parity block P2, for logical blocks 8, 9, 10, and 11, is stored in disk (2 mod 5)+1 = 3

parity block P3, for logical blocks 12, 13, 14, and 15, is stored in disk (3 mod 5)+1 = 4

parity block P4, for logical blocks 16, 17, 18, and 19, is stored in disk (4 mod 5)+1 = 5
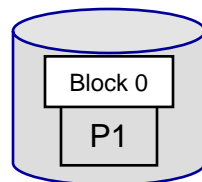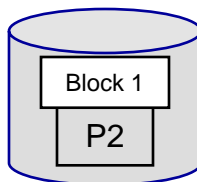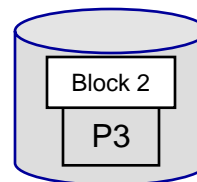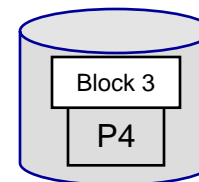
	Block 0	Block 1	Block 2	Block 3
P0	P1	P2	P3	P4
Disk 1	Disk 2	Disk 3	Disk 4	Disk 5

RAID Level 5 Example (cont.)

Disk 1	Disk 2	Disk 3	Disk 4	Disk 5
	Block 0	Block 1	Block 2	Block 3
Block 4		Block 5	Block 6	Block 7
Block 8	Block 9		Block 10	Block 11
Block 12	Block 13	Block 14		Block 15
Block 16	Block 17	Block 18	Block 19	
P0	P1	P2	P3	P4
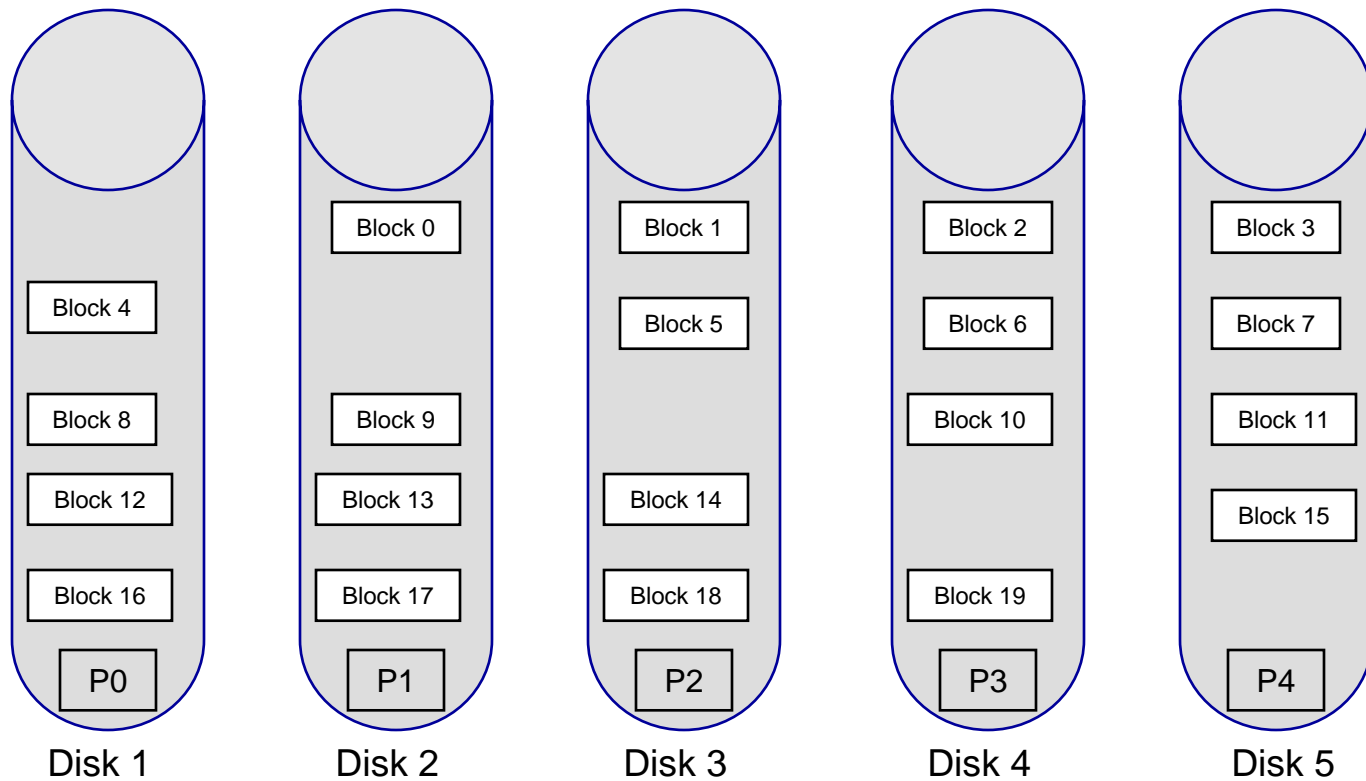
RAID Levels (cont.)

- **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.

RAID Levels (cont.)
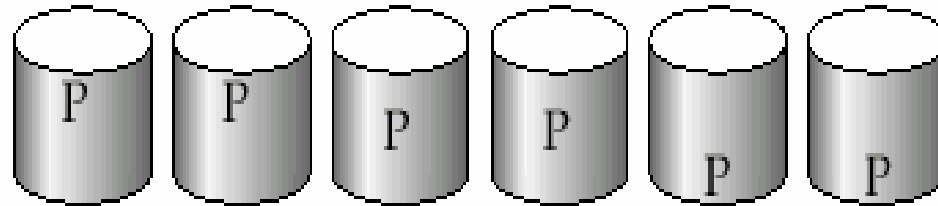
RAID 6: P + Q redundancy

RAID Levels (cont.)
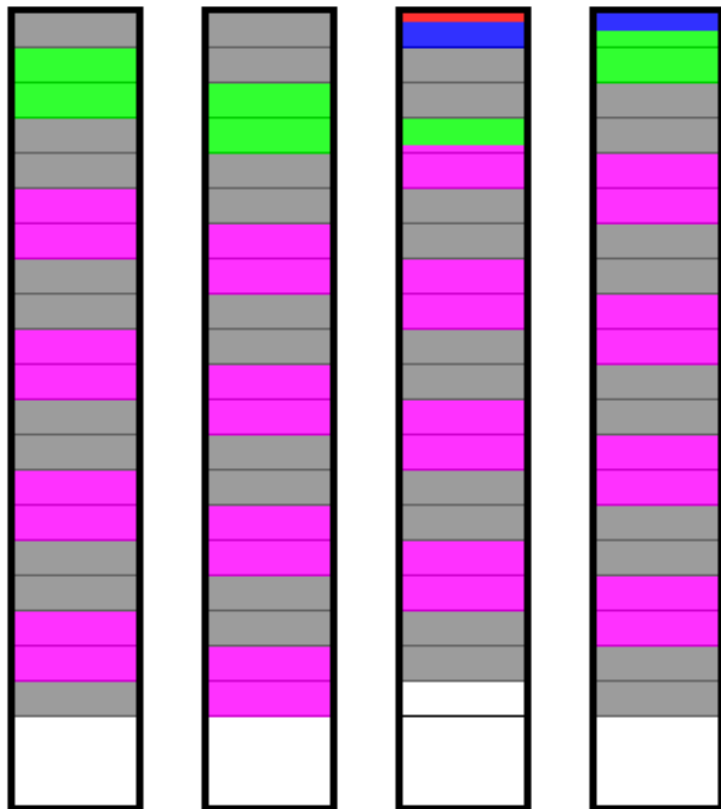
This illustration shows how files of different sizes are distributed between the drives on a four-disk RAID 6 array using a 16 kB stripe size. As with the RAID 0 illustration, the red file is 4 kB in size; the blue is 20 kB; the green is 100 kB; and the magenta is 500 kB, with each vertical pixel representing 1 kB of space. This diagram is the same as the RAID 5 one, except that you'll notice that there is now twice as much gray parity information, and as a result, more space taken up on the four drives to contain the same data than the other levels that use striping.

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 almost all applications
- So competition is between 1 and 5 only.

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

Additional RAID Levels

- The single RAID levels have distinct advantages and disadvantages, which is why most of them are used in various parts of the market to address different application requirements.

- It wasn't long after RAID began to be implemented that engineers looked at these RAID levels and began to wonder if it might be possible to get some of the advantages of more than one RAID level by designing arrays that use a combination of techniques.

- These RAID levels are called variously *multiple*, *nested*, or *multi*-RAID levels. They are also sometimes called *two-dimensional*, in reference to the two-dimensional schematics that are used to represent the application of two RAID levels to a set of disks.

- Multiple RAID levels are most commonly used to improve performance, and they do this well. Nested RAID levels typically provide better performance characteristics than either of the single RAID levels that comprise them. The most commonly combined level is RAID 0, which is often mixed with redundant RAID levels such as 1, 3 or 5 to provide fault tolerance while exploiting the performance advantages of RAID 0. There is never a "free lunch", and so with multiple RAID levels what you pay is a cost in complexity: many drives are required, management and maintenance are more involved, and for some implementations a high-end RAID controller is required.

- Not all combinations of RAID levels exist. Typically, the most popular multiple RAID levels are those that combine single RAID levels that complement each other with different strengths and weaknesses. Making a multiple RAID array marrying RAID 4 to RAID 5 wouldn't be the best idea, since they are so similar to begin with.

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, the requesting program is given the address of the block in main memory

  2. If the block is not in the buffer,

     1. The buffer manager allocates space in the buffer for the block, replacing (throwing out) some other block, if required, to make space for the new block.

     2. The block that is thrown out is written back to disk only if it was modified since the most recent time that it was written to/fetched from the disk.

     3. Once space is allocated in the buffer, the buffer manager reads the block from the disk to the buffer, and passes the address of the block in main memory to requester.

Buffer Replacement Protocols

- Most operating systems replace the block **least recently used** (LRU strategy)

- Idea behind LRU – use past pattern of block references as a predictor of future references

- Queries have well-defined access patterns (such as sequential scans), and a database system can use the information in a user's query to predict future references

  - LRU can be a bad strategy for certain access patterns involving repeated scans of data

    - e.g. when computing the join of 2 relations r and s by a nested loops

      for each tuple *tr* of *r* do

        for each tuple *ts* of *s* do

          if the tuples *tr* and *ts* match …

    - Mixed strategy with hints on replacement strategy provided by the query optimizer is preferable.

Buffer Replacement Protocols (cont.)

- **Pinned block** – memory block that is not allowed to be written back to disk.

- **Toss-immediate** strategy – frees the space occupied by a block as soon as the final tuple of that block has been processed

- Most recently used (MRU) strategy – system must pin the block currently being processed. After the final tuple of that block has been processed, the block is unpinned, and it becomes the most recently used block.

- Buffer manager can use statistical information regarding the probability that a request will reference a particular relation
  - E.g., the data dictionary is frequently accessed. Heuristic: keep data-dictionary blocks in main memory buffer

- Buffer managers also support forced output of blocks for the purpose of recovery.

File Organizations

- 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 $i + 1, \ldots, n$ to $i, \ldots, n - 1$
  - move record n to i
  - do not move records, but link all free records on a *free list*

record 0	A-102	Perryridge	400
record 1	A-305	Round Hill	350
record 2	A-215	Mianus	700
record 3	A-101	Downtown	500
record 4	A-222	Redwood	700
record 5	A-201	Perryridge	900
record 6	A-217	Brighton	750
record 7	A-110	Downtown	600
record 8	A-218	Perryridge	700

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	A-102	Perryridge	400	
record 1				
record 2	A-215	Mianus	700	
record 3	A-101	Downtown	500	
record 4				
record 5	A-201	Perryridge	900	
record 6				
record 7	A-110	Downtown	600	
record 8	A-218	Perryridge	700	
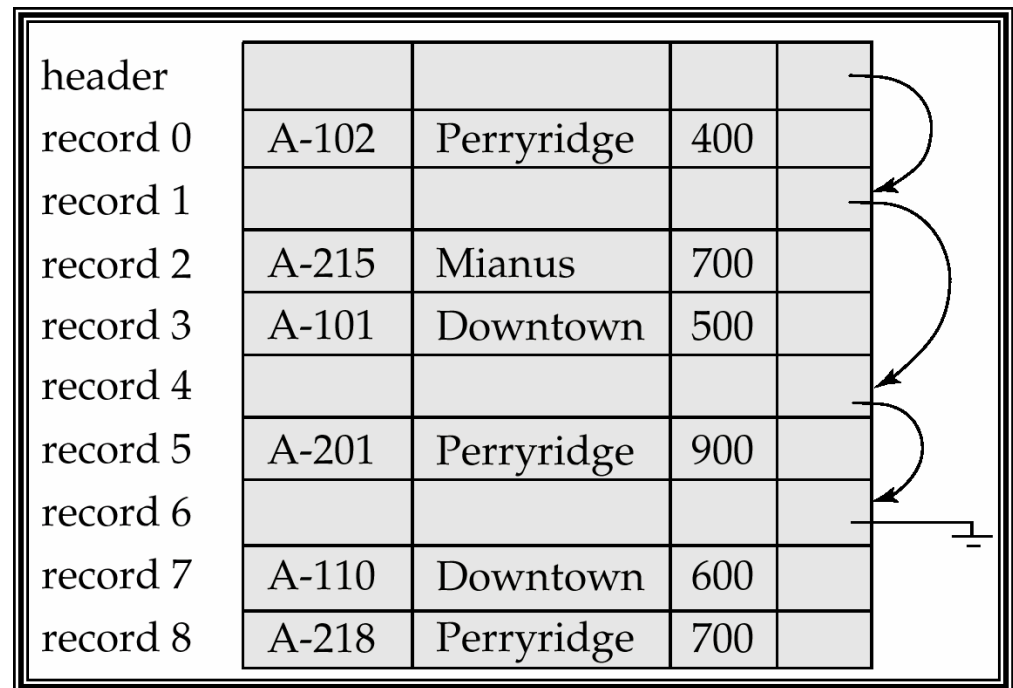
Variable-Length Records

- 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.
  - Record types that allow repeating fields (used in some older data models).

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

Variable-Length Records – Slotted Page

- 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.

Variable-Length Records – Slotted Page (cont.)
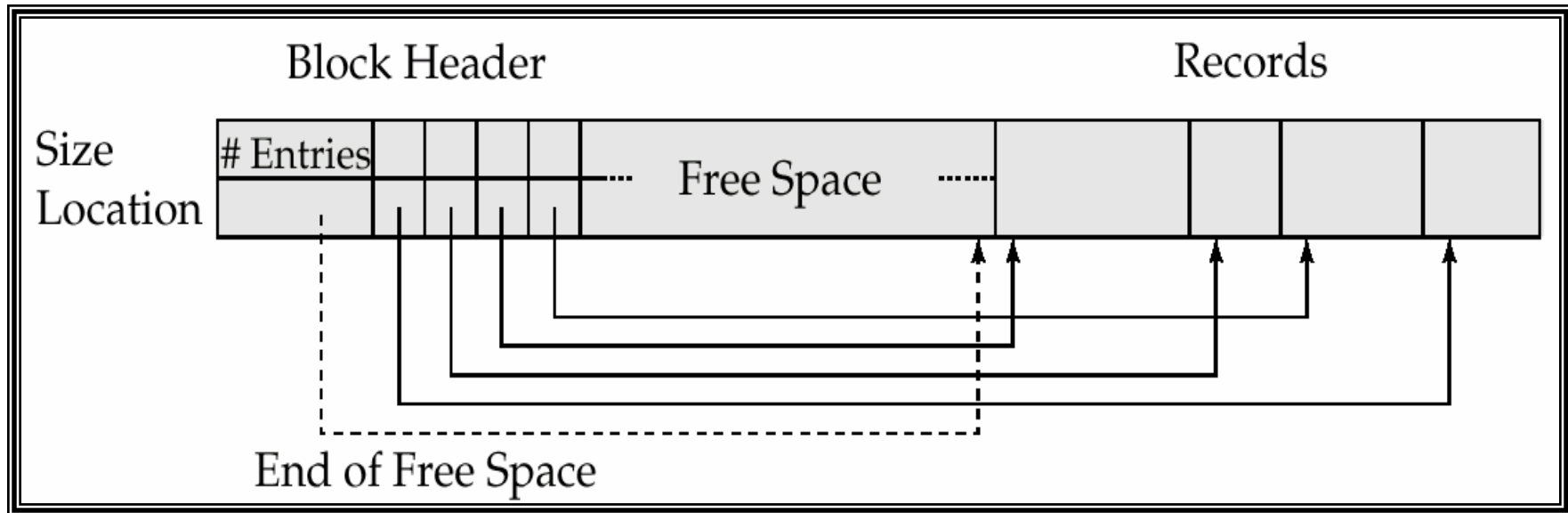
Variable-Length Records (cont.)

- Fixed-length representation:
  - reserved space
  - pointers

- Reserved space – can use fixed-length records of a known maximum length; unused space in shorter records filled with a null or end-of-record symbol.

0	Perryridge	A-102	400	A-201	900	A-218	700
1	Round Hill	A-305	350	⊥	⊥	⊥	⊥
2	Mianus	A-215	700	⊥	⊥	⊥	⊥
3	Downtown	A-101	500	A-110	600	⊥	⊥
4	Redwood	A-222	700	⊥	⊥	⊥	⊥
5	Brighton	A-217	750	⊥	⊥	⊥	⊥

Variable-Length Records – Pointer Method

- Pointer method

  - A variable-length record is represented by a list of fixed-length records, chained together via pointers.

  - Can be used even if the maximum record length is not known

0	Perryridge	A-102	400	
1	Round Hill	A-305	350	
2	Mianus	A-215	700	
3	Downtown	A-101	500	
4	Redwood	A-222	700	
5		A-201	900	
6	Brighton	A-217	750	
7		A-110	600	
8		A-218	700	

Variable-Length Records – Pointer Method
(cont.)

- Disadvantage to pointer structure; space is wasted in all records except the first in a a chain.

- Solution is to allow two kinds of block in file:

  - Anchor block – contains the first records of chain

  - Overflow block – contains records other than those that are the first records of chairs.

anchor block	Perryridge	A-102	400	
	Round Hill	A-305	350	
	Mianus	A-215	700	
	Downtown	A-101	500	
	Redwood	A-222	700	
	Brighton	A-217	750	

overflow block	A-201	900	
	A-218	700	
	A-110	600	
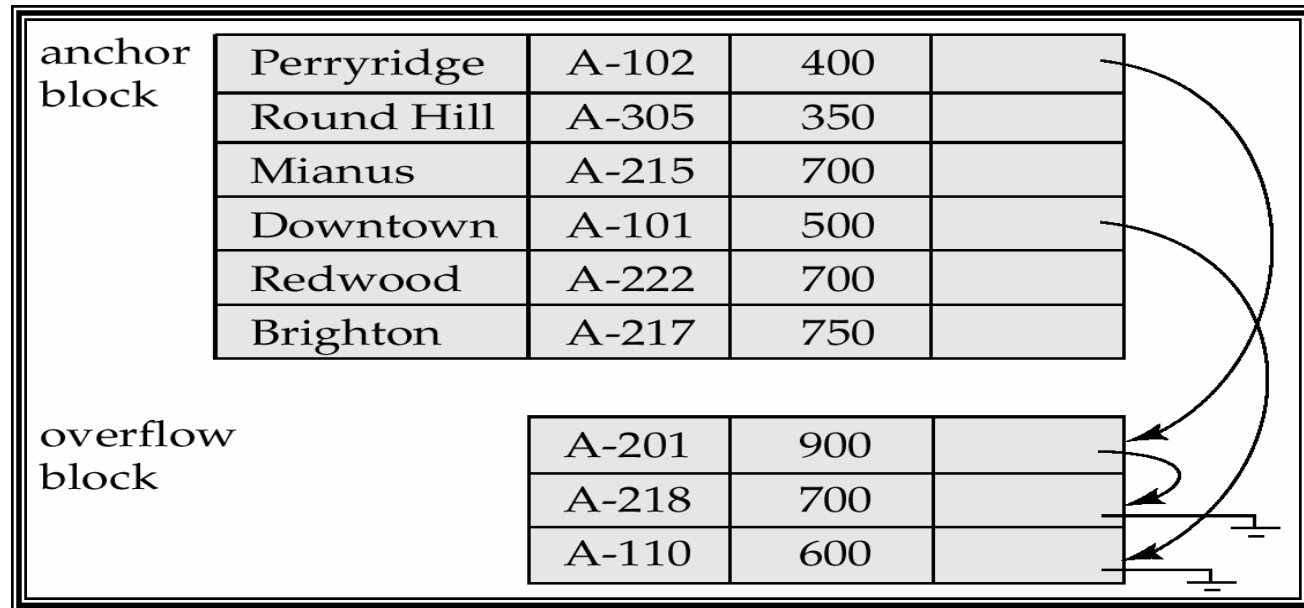
Organization of Records in Files

- **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 **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

A-217	Brighton	750	
A-101	Downtown	500	
A-110	Downtown	600	
A-215	Mianus	700	
A-102	Perryridge	400	
A-201	Perryridge	900	
A-218	Perryridge	700	
A-222	Redwood	700	
A-305	Round Hill	350	

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.

A-217	Brighton	750	
A-101	Downtown	500	
A-110	Downtown	600	
A-215	Mianus	700	
A-102	Perryridge	400	
A-201	Perryridge	900	
A-218	Perryridge	700	
A-222	Redwood	700	
A-305	Round Hill	350	

A-888	North Town	800	
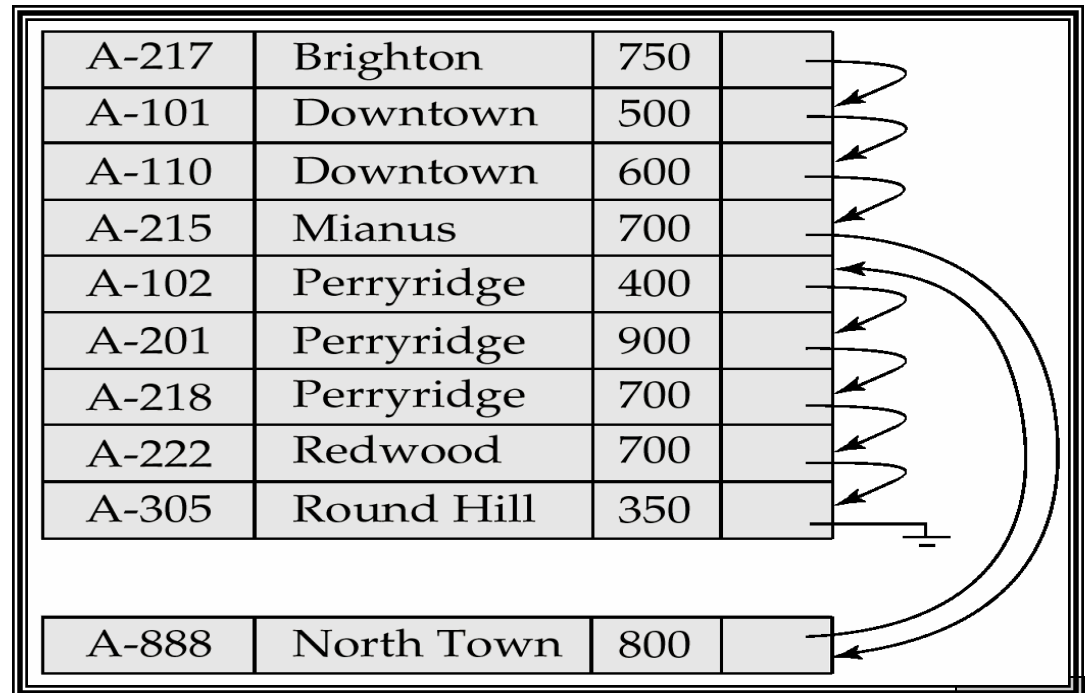
Clustering File Organization

- Simple file structure stores each relation in a separate file

- Can instead store several relations in one file using a **clustering** file organization

- E.g., clustering organization of *customer* and *depositor:*

- Good for queries involving depositor customer, and for queries involving one single customer and his accounts

- Bad for queries involving only customer

- Results in variable size records

Hayes	Main	Brooklyn
Hayes	A-102	
Hayes	A-220	
Hayes	A-503	
Turner	Putnam	Stamford
Turner	A-305	