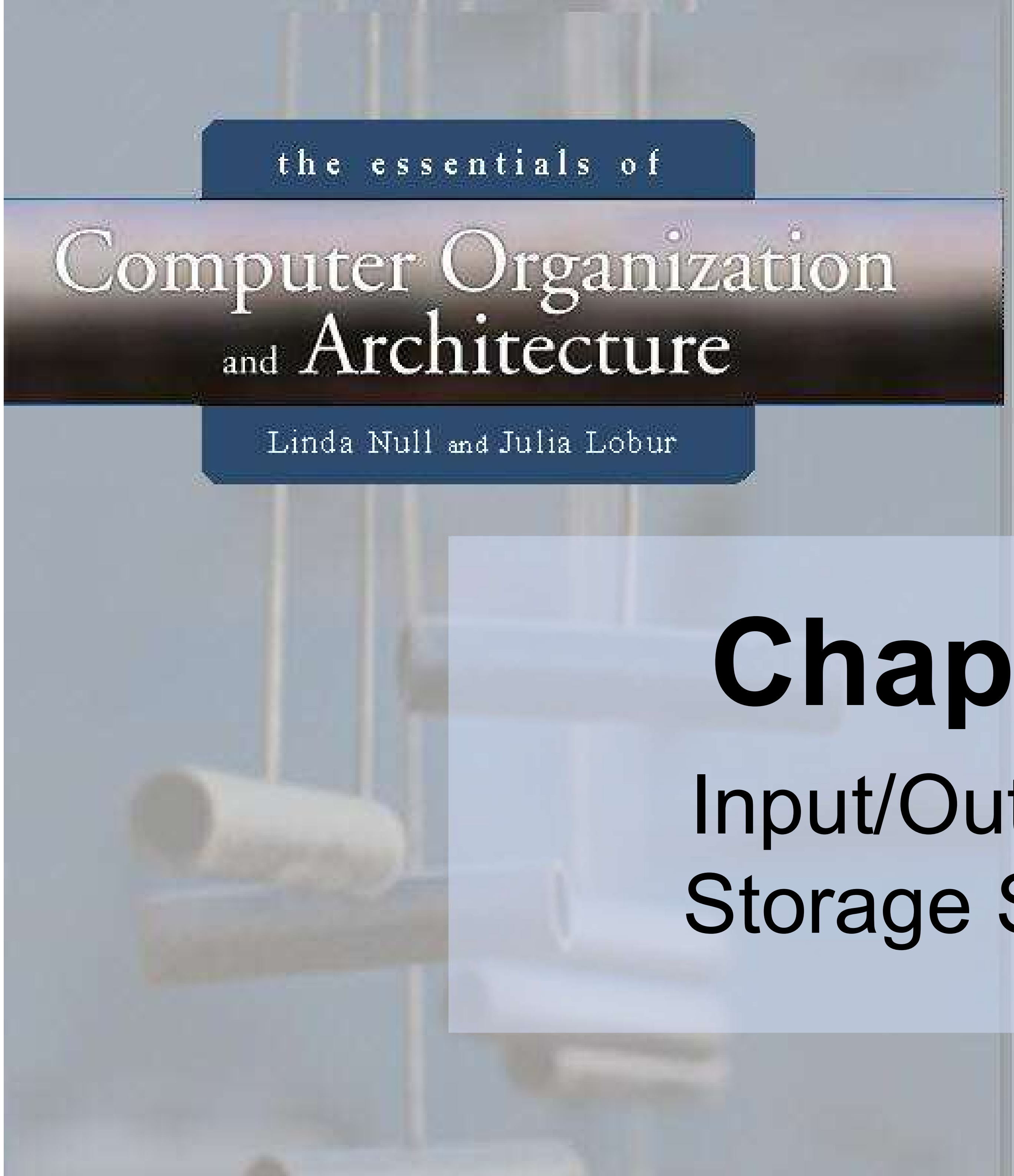


the essentials of

Computer Organization and Architecture

Linda Null and Julia Lobur

Chapter 7

Input/Output and Storage Systems

Chapter 7 Objectives

- Understand how I/O systems work, including I/O methods and architectures.
- Become familiar with storage media, and the differences in their respective formats.
- Understand how RAID improves disk performance and reliability.
- Become familiar with the concepts of data compression and applications suitable for each type of compression algorithm.

7.1 Introduction

- Data storage and retrieval is one of the primary functions of computer systems.
- Sluggish I/O performance can have a ripple effect, dragging down overall system performance.
- This is especially true when virtual memory is involved.
- The fastest processor in the world is of little use if it spends most of its time waiting for data.

7.2 Amdahl's Law

- Suppose the daytime processing load consists of 60% CPU activity and 40% disk activity. Your customers are complaining that the system is slow. After doing some research, you have learned that you can upgrade your disks for $8,000 to make them 2.5 times as fast as they are currently. You have also learned that you can upgrade your CPU to make it 1.4 as fast for $5,000.
  - a. Which would you choose to yield the best performance improvement for the least amount of money?
  - b. Which option would you choose if you don't care about the money, but want a faster system?

7.2 Amdahl's Law

- The overall performance of a system is a result of the interaction of all of its components.
- System performance is most effectively improved when the performance of the most heavily used components is improved.
- This idea is quantified by Amdahl's Law（阿姆达尔定律）:

$$S = \frac{1}{(1-f) + \frac{f}{k}}$$

where S is the overall speedup; f is the fraction of work performed by a faster component; and k is the speedup of the faster component.

7.2 Amdahl’s Law

- Amdahl’s Law gives us a handy way to estimate the performance improvement we can expect when we upgrade a system component.
- On a large system, suppose we can upgrade a CPU to make it 50% faster for $10,000 or upgrade its disk drives for $7,000 to make them 250% faster.
- Processes spend 70% of their time running in the CPU and 30% of their time waiting for disk service.
- An upgrade of which component would offer the greater benefit for the lesser cost?

7.2 Amdahl's Law

- The processor option offers a speedup of 1.3 times, or 30%:

$$f = 0.70, \quad k = 1.5 \qquad S = \frac{1}{(1 - 0.7) + 0.7/1.5}$$

- And the disk drive option gives a speedup of 1.22 times, or 22%:

$$f = 0.30, \quad k = 2.5 \qquad S = \frac{1}{(1 - 0.3) + 0.3/2.5}$$

- Each 1% of improvement for the processor costs \$333, and for the disk a 1% improvement costs \$318.

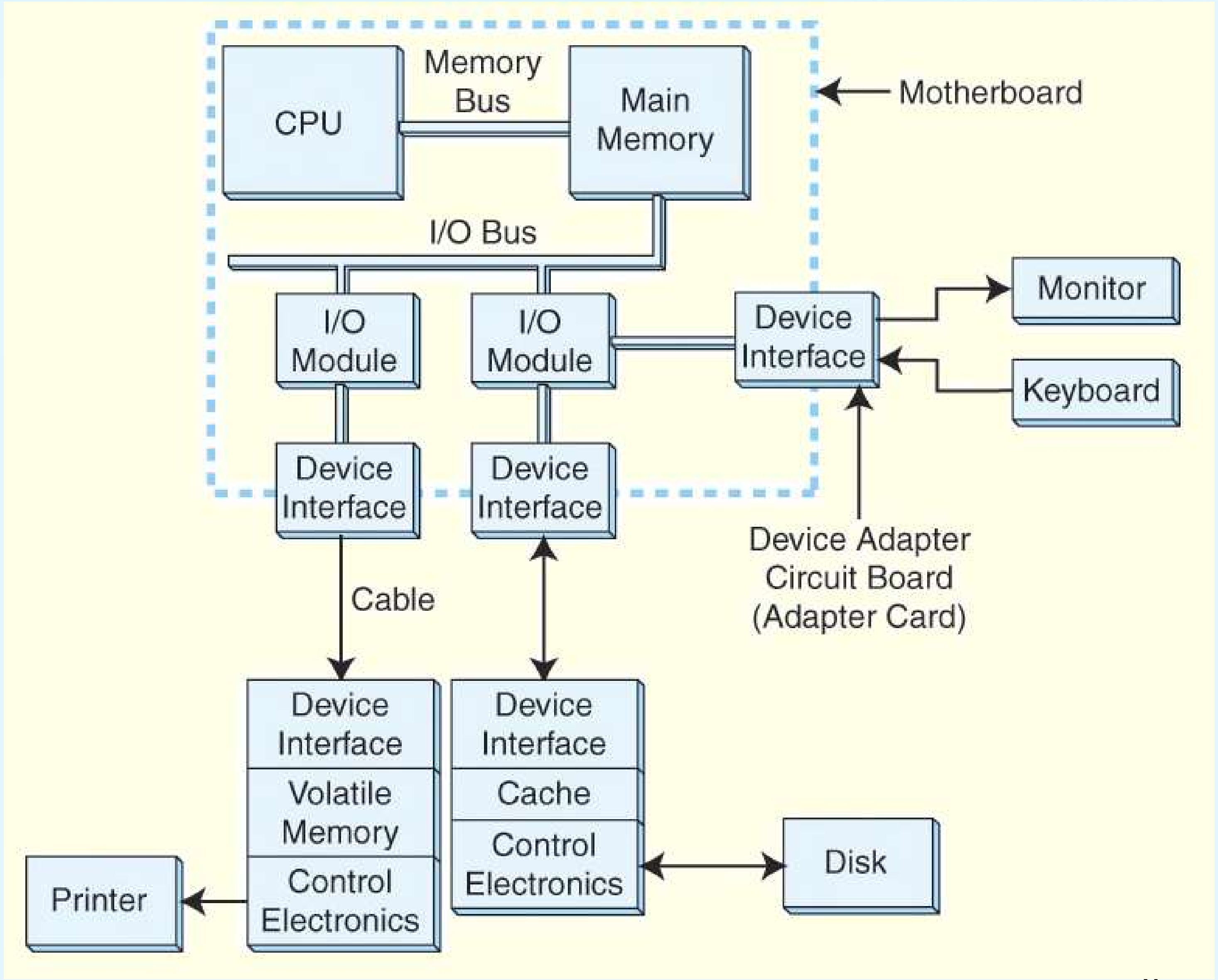
Should price/performance be your only concern?

7.3 I/O Architectures

- We define input/output as a subsystem of components that moves coded data between external devices and a host system.
- I/O subsystems include:
  - Blocks **of main memory** that are devoted to I/O functions.
  - **Buses** that move data into and out of the system.
  - **Control modules** in the host and in peripheral devices
  - **Interfaces** to external components such as keyboards and disks.
  - **Cabling or communications links** between the host system and its peripherals.

7.3 I/O Architectures

This is a model I/O configuration.



7.3 I/O Architectures

- I/O can be controlled in four general ways.
- **Programmed I/O** reserves a register for each I/O device. Each register is continually polled to detect data arrival.
- **Interrupt-Driven I/O** allows the CPU to do other things until I/O is requested.
- **Direct Memory Access** (DMA) offloads I/O processing to a special-purpose chip that takes care of the details.
- **Channel I/O** uses dedicated I/O processors.

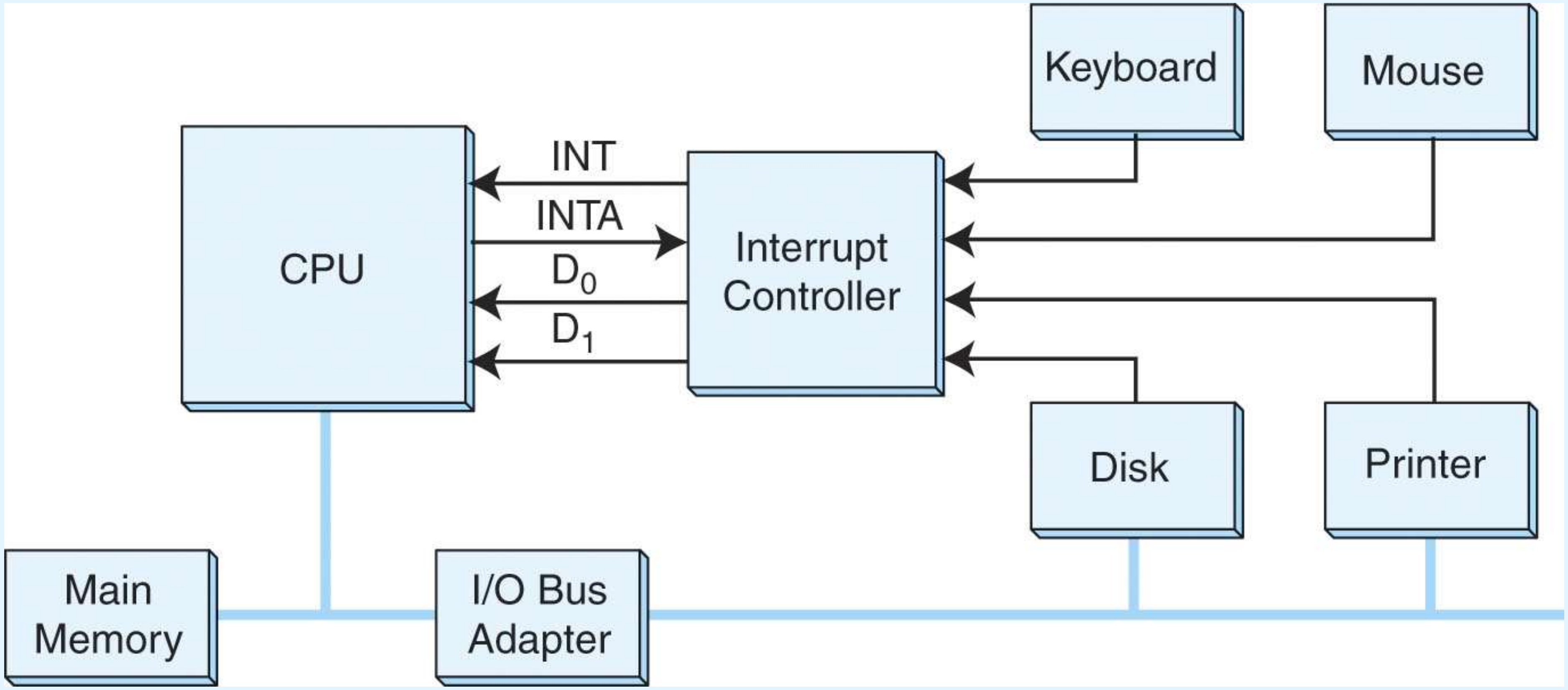
7.3 I/O Architectures

- Systems using *programmed I/O* devote at least one register for the exclusive use of each I/O device
- The CPU continually monitors each register, waiting for data to arrive. **Polling**
- Once the CPU detects a "data ready" condition, it acts according to instructions programmed for that particular register.
- Benefit: easy programmed control of each device, change of priority, polling interval
- Problem: time waste on polling and waiting.

7.3 I/O Architectures

- Instead of the CPU continually asking its attached devices whether they have any input, the devices tell the CPU when they have data to send
- The CPU proceeds with other tasks until a device requesting service interrupts it
- Interrupts are usually signaled with a bit in the CPU flags register called an *interrupt flag*.
- The system fetches the *address vector that points to the address of the I/O* service routine.
- Virus writers could replace the I/O vectors with pointers to their own nefarious code.
- I/O with FIFO

7.3 I/O Architectures



An I/O subsystem using interrupts

Interrupt request, Interrupt acknowledge, Interrupt number, Interrupt priority

7.3 I/O Architectures

- A CPU with interrupt-driven I/O is busy servicing a disk request. While the CPU is midway through the disk-service routine, another I/O interrupt occurs.
  - a. What happens next?
  - b. Is it a problem?
  - c. If not, why not? If so, what can be done about it?
  - d. If a device generates interrupt request every 1ms, another device service routine needs 1ms.
    What will happen?

7.3 I/O Architectures

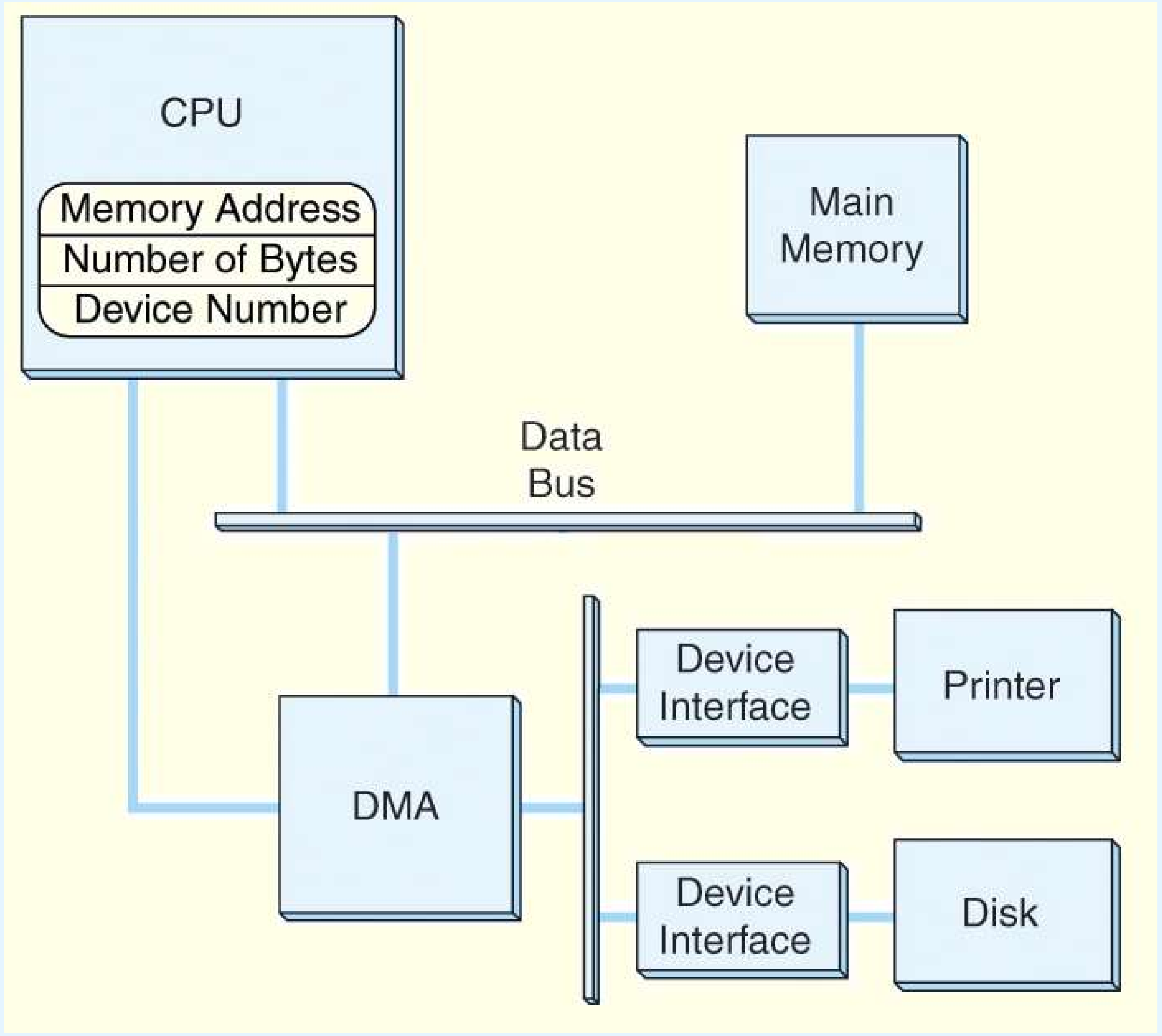
- Clearly, data moving is simple enough to be programmed in a dedicated chip
- When a system uses DMA, the CPU offloads execution of tedious I/O instructions
- CPU provides the DMA controller with the location of the bytes to be transferred, the number of bytes to be transferred, and the destination device or memory address.
- After the I/O is complete (or ends in error), the DMA subsystem signals the CPU by sending it an interrupt

7.3 I/O Architectures

This is a DMA configuration.

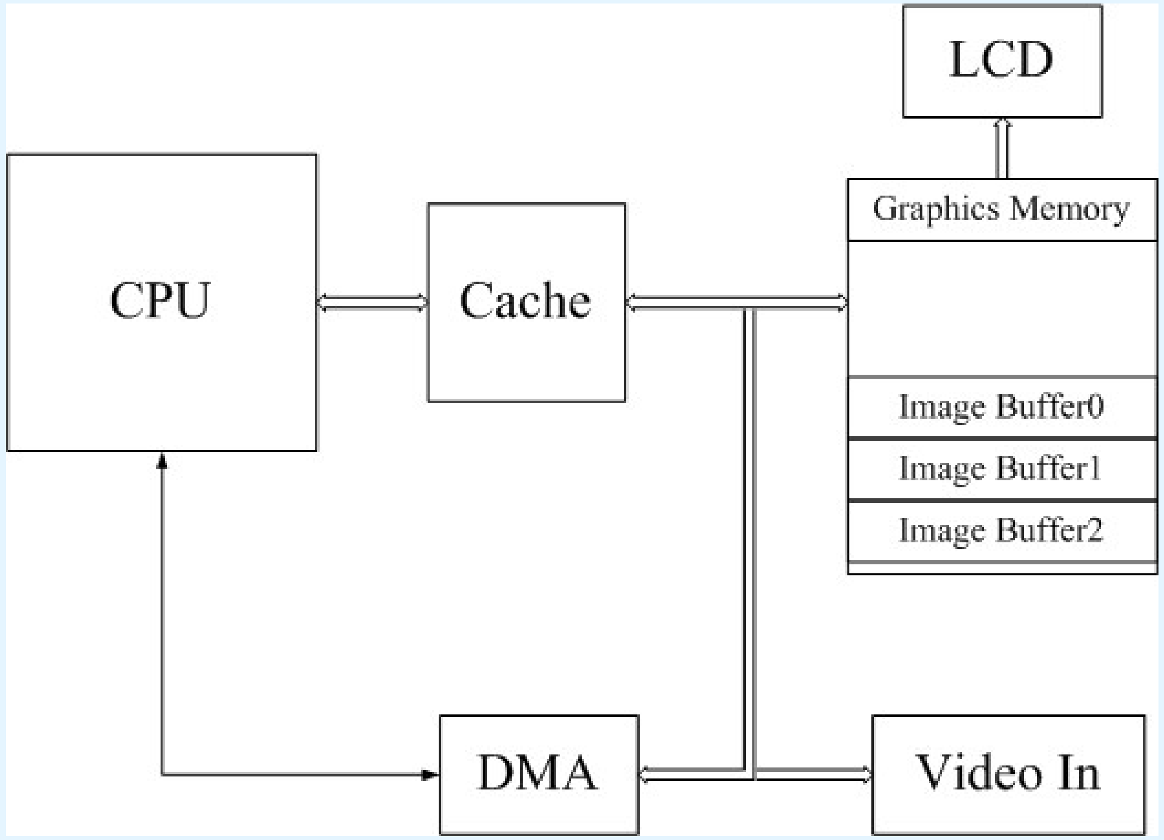
Notice that the DMA and the CPU share the bus.

The DMA runs at a higher priority and steals memory cycles from the CPU.



7.3 I/O Architectures

- Data coherency problem when using cache and DMA
  - Why has the problem?
  - How to solve it?
    - Write Through
    - Write Back
    - Flush
    - Invalidate



7.3 I/O Architectures

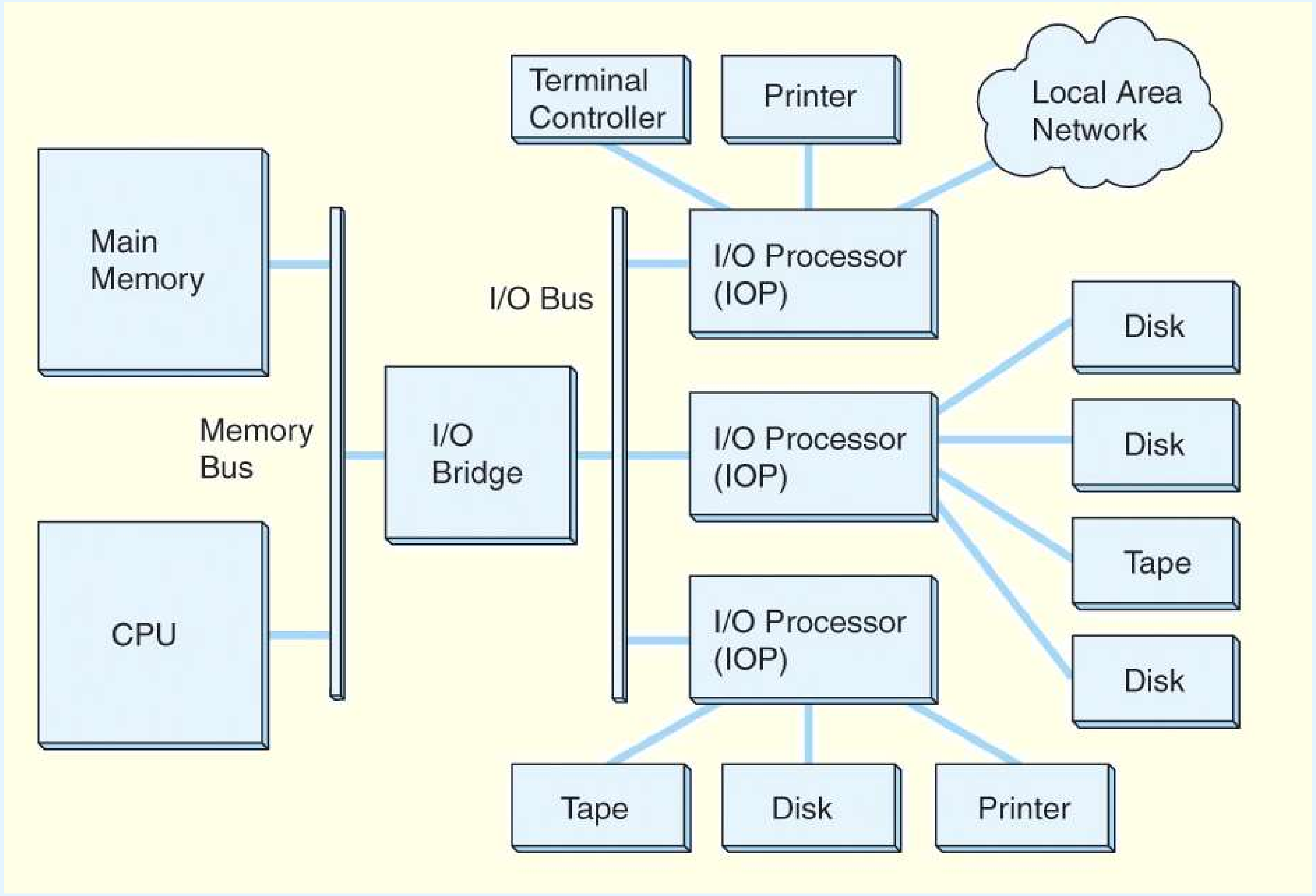
- Very large systems employ channel I/O.
- Channel I/O consists of one or more I/O processors (IOPs) that control various channel paths.
- Slower devices such as terminals and printers are combined (*multiplexed*) into a single faster channel.
- On IBM mainframes, multiplexed channels are called *multiplexor channels*, the faster ones are called *selector channels*.

7.3 I/O Architectures

- Channel I/O is distinguished from DMA by the intelligence of the IOPs.
- The IOP negotiates protocols, issues device commands, translates storage coding to memory coding, and can transfer entire files or groups of files independent of the host CPU.
- The host has only to create the program instructions for the I/O operation and tell the IOP where to find them.

7.3 I/O Architectures

- This is a channel I/O configuration.

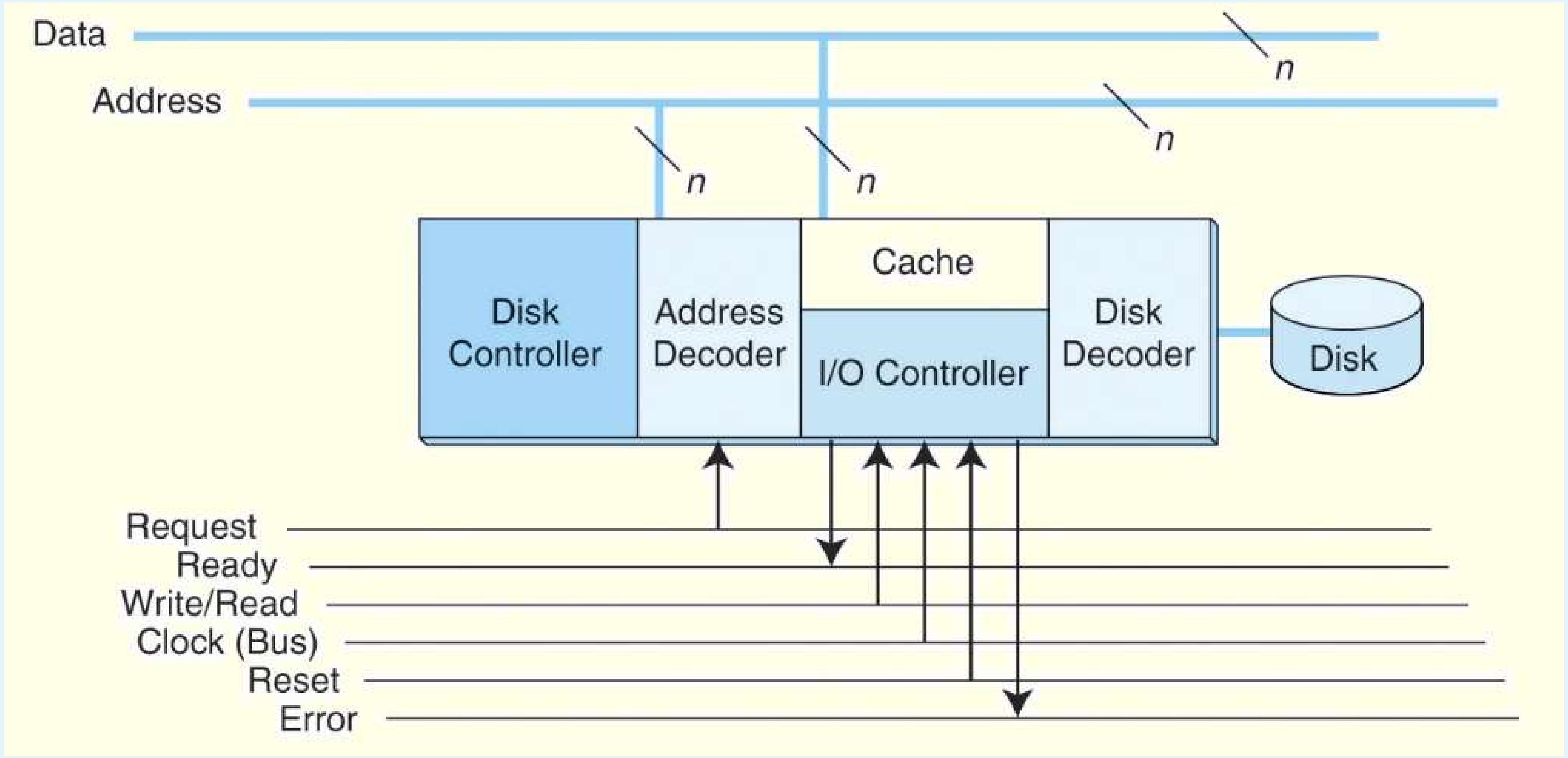


7.3 I/O Architectures

- Memory transfers can be synchronous since memory is always on-line whenever CPU accesses memory
- I/O buses, unlike memory buses, operate **asynchronously**. I/O devices cannot always be ready. Handshakes should be used every time a device is to be accessed.
- Bus control lines activate the devices when they are needed, raise signals when errors have occurred, and reset devices when necessary.
- The number of data lines is the *width* of the bus.
- A bus clock coordinates activities and provides bit cell boundaries.

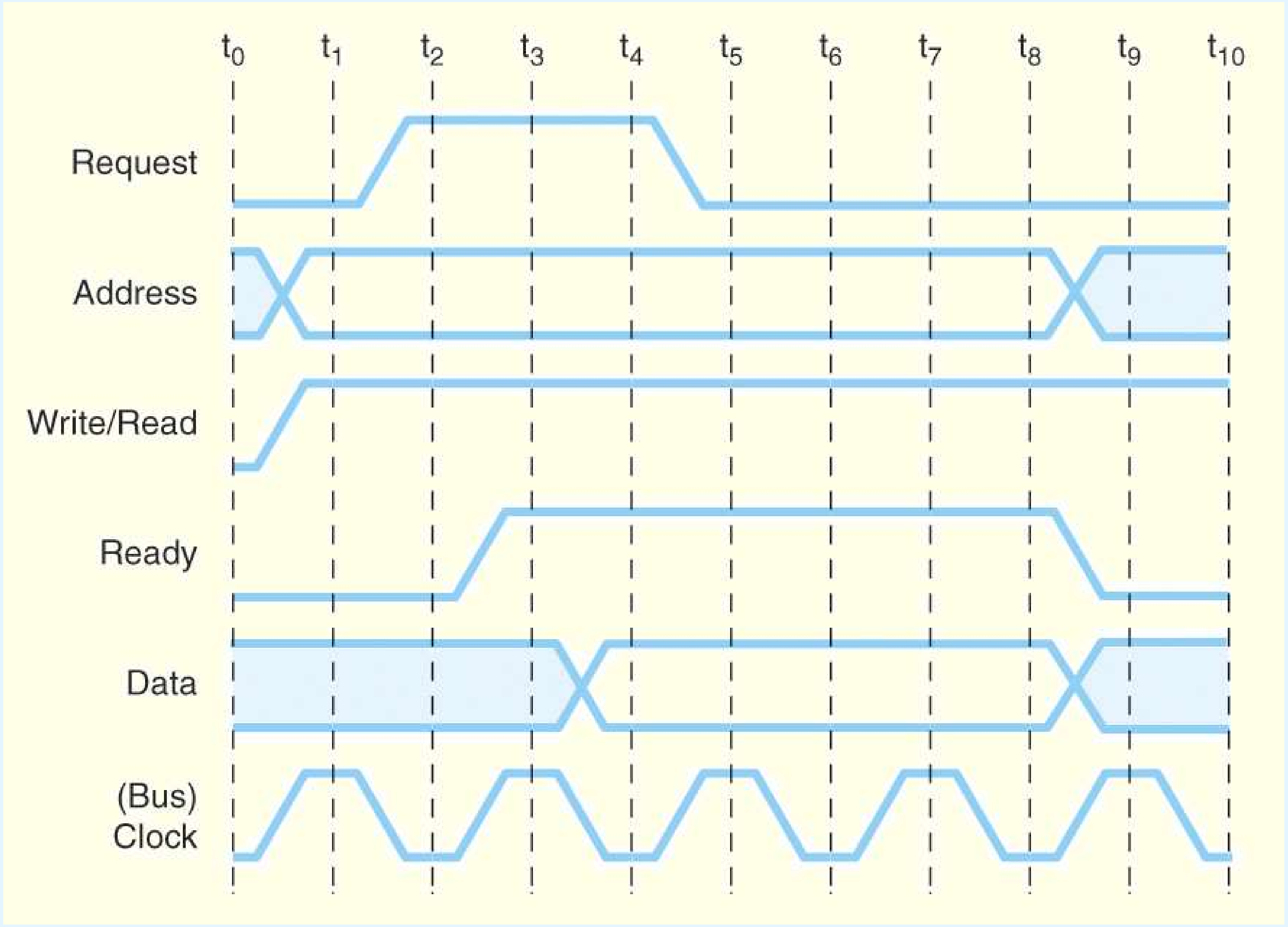
7.3 I/O Architectures

This is how a bus connects to a disk drive.

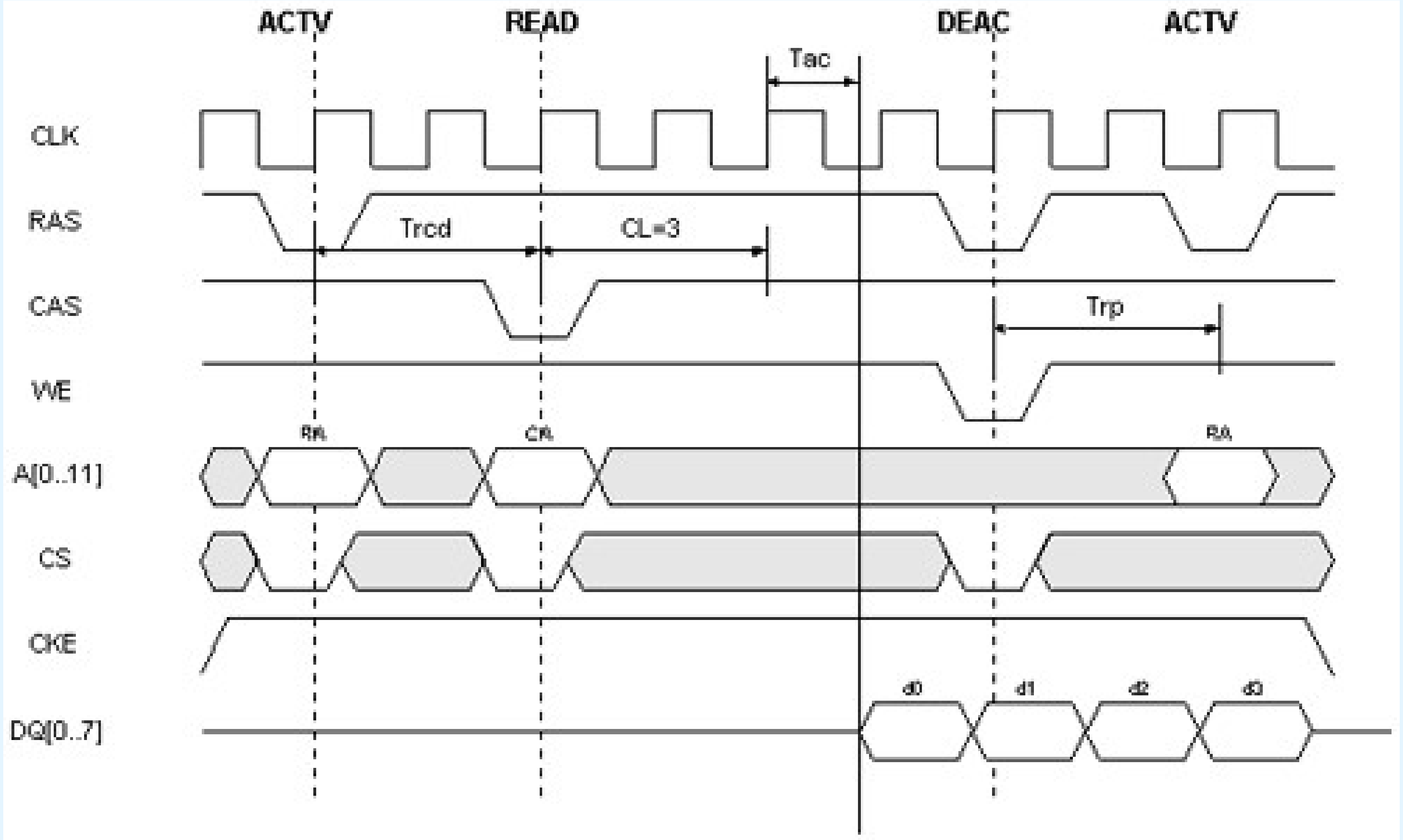


7.3 I/O Architectures

Timing diagrams, such as this one, define bus operation in detail.



7.3 I/O Architectures



SDRAM burst read timing

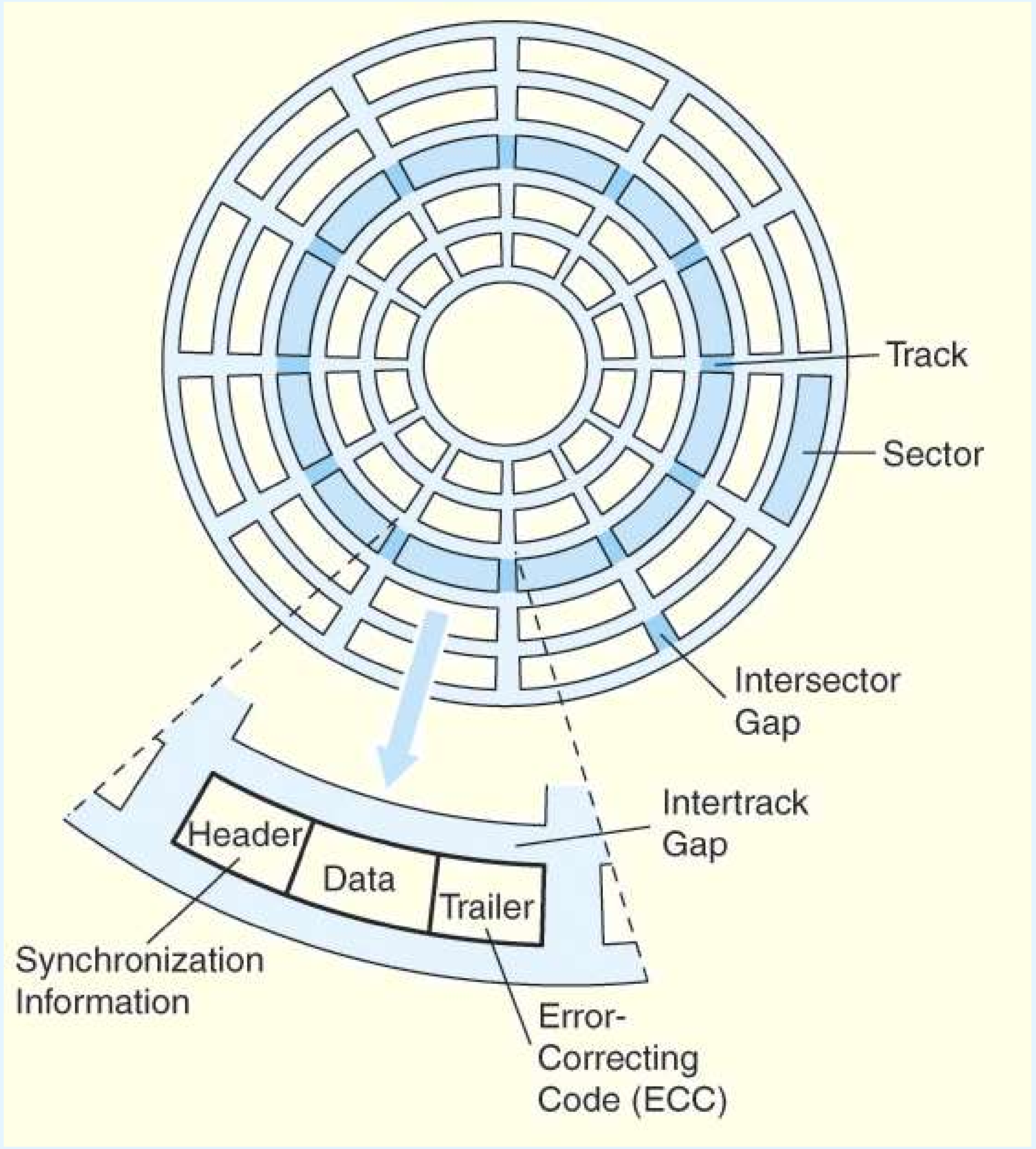
Example of Synchronous bus timing diagram

7.4 Magnetic Disk Technology

- Magnetic disks offer large amounts of durable storage that can be accessed quickly.
- Disk drives are called *random* (or *direct*) *access storage devices,* because blocks of data can be accessed according to their location on the disk.
  - This term was coined when all other durable storage (**e.g., tape**) was sequential.
- Magnetic disk organization is shown on the following slide.

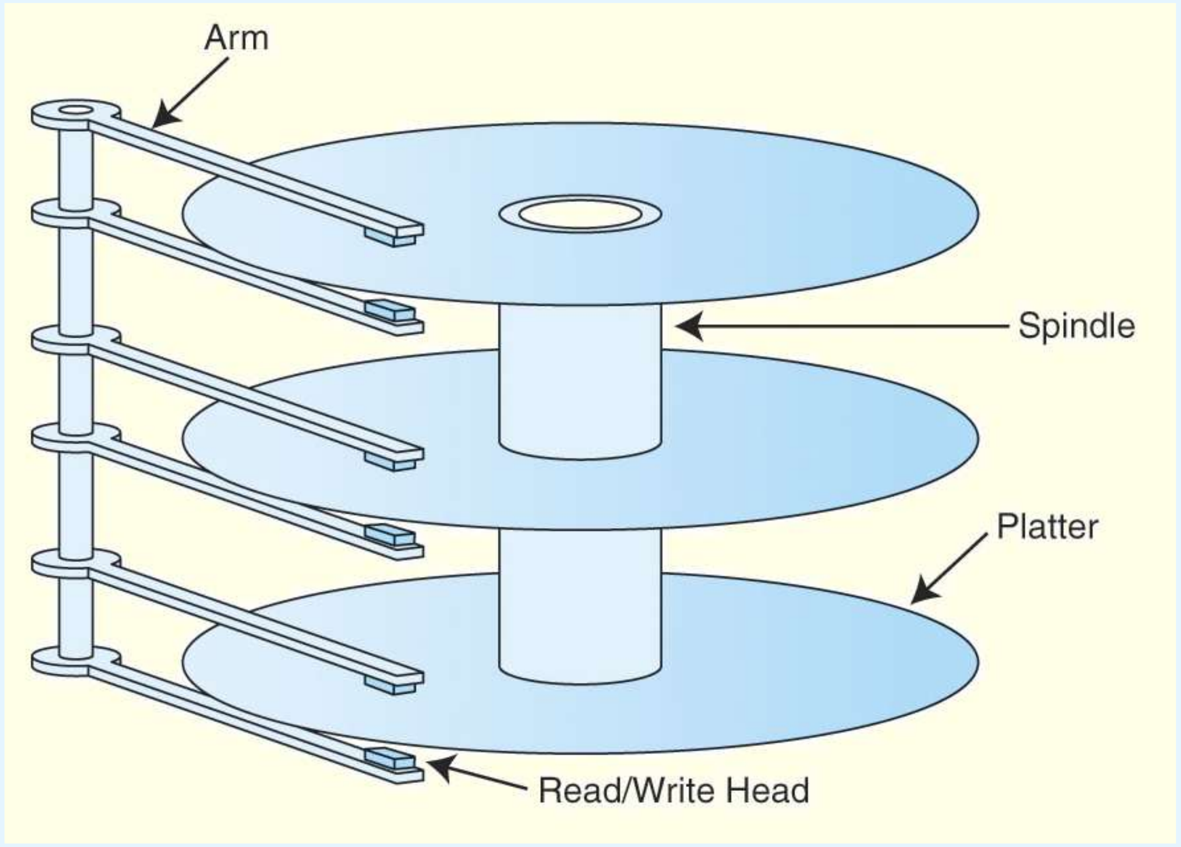
7.4 Magnetic Disk Technology

Disk tracks are numbered from the outside edge, starting with zero.



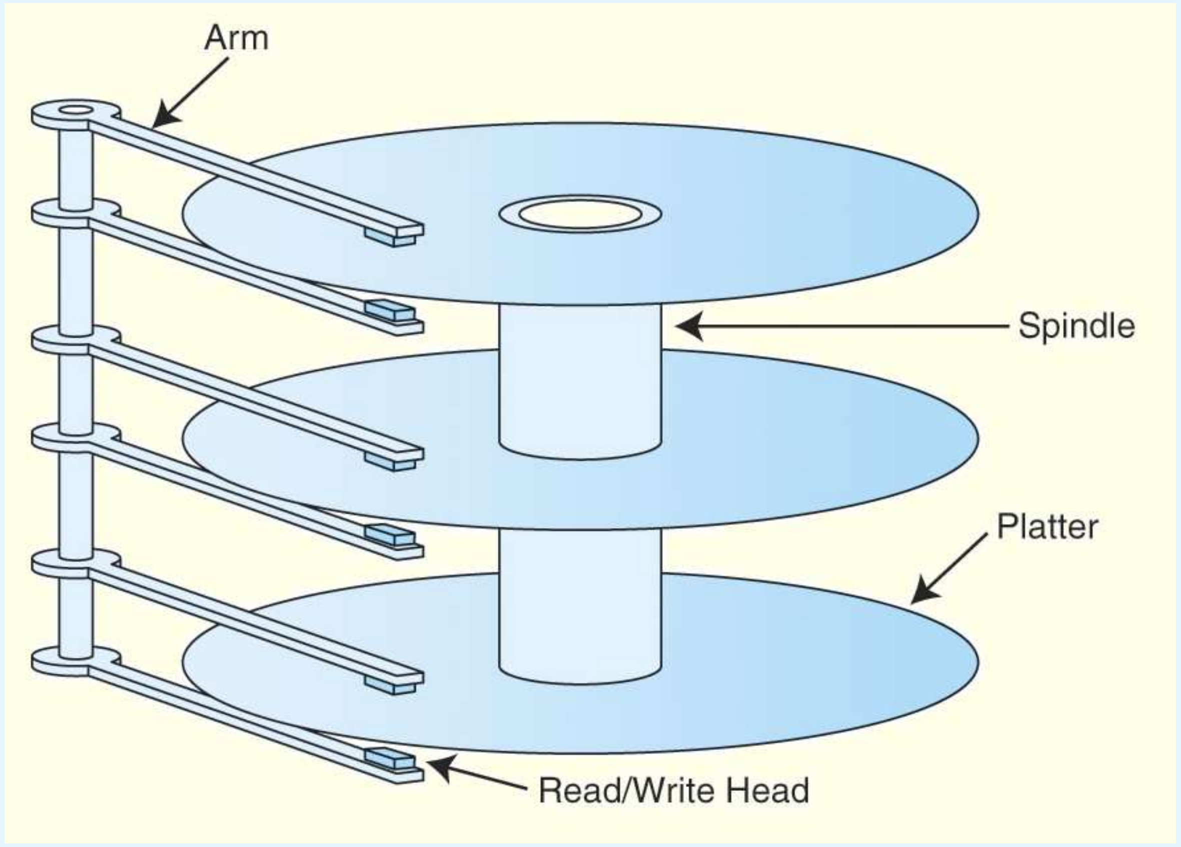
7.4 Magnetic Disk Technology

- Hard disk platters are mounted on spindles.
- Read/write heads are mounted on a comb that swings radially to read the disk.



7.4 Magnetic Disk Technology

- The rotating disk forms a logical cylinder beneath the read/write heads.
- Data blocks are addressed by their cylinder, surface, and sector.



7.4 Magnetic Disk Technology

- There are a number of electromechanical properties of hard disk drives that determine how fast its data can be accessed.
- ***Seek time*** is the time that it takes for a disk arm to move into position over the desired cylinder.
- ***Rotational delay*** is the time that it takes for the desired sector to move into position beneath the read/write head.
- Seek time + rotational delay = *access time*.

7.4 Magnetic Disk Technology

- *Transfer rate* gives us the rate at which data can be read from the disk.
- *Average latency* is a function of the rotational speed:

$$\frac{\dfrac{60 \text{ seconds}}{\text{disk rotation speed}} \times \dfrac{1000 \text{ ms}}{\text{second}}}{2}$$

- *Mean Time To Failure (MTTF)* is a statistically-determined value often calculated experimentally.
  - It usually doesn't tell us much about the actual expected life of the disk. *Design life* is usually more realistic.

7.4 Magnetic Disk Technology

- Floppy (flexible) disks are organized in the same way as hard disks, with concentric tracks that are divided into sectors.
- Physical and logical limitations restrict floppies to much lower densities than hard disks.
- A major logical limitation of the DOS/Windows floppy diskette is the organization of its file allocation table (FAT).
  - The FAT gives the status of each sector on the disk: Free, in use, damaged, reserved, etc.

7.4 Magnetic Disk Technology

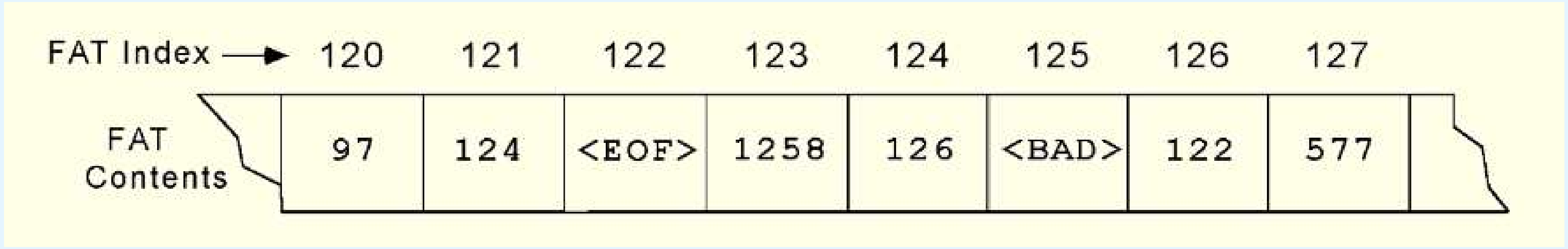
- On a standard 1.44MB floppy, the FAT is limited to nine 512-byte sectors.
  - There are two copies of the FAT.
- There are 18 sectors per track and 80 tracks on each surface of a floppy, for a total of 2880 sectors on the disk. So each FAT entry needs at least 14 bits ($2^{14}=4096 < 2^{13} = 2048$).
  - FAT entries are actually 16 bits, and the organization is called FAT16.

7.4 Magnetic Disk Technology

- The disk directory associates logical file names with physical disk locations.
- Directories contain a file name and the file’s first FAT entry.
- If the file spans more than one sector (or cluster), the FAT contains a pointer to the next cluster (and FAT entry) for the file.
- The FAT is read like a linked list until the <EOF> entry is found.

7.4 Magnetic Disk Technology

- A directory entry says that a file we want to read starts at sector 121 in the FAT fragment shown below.



FAT Index →	120	121	122	123	124	125	126	127
FAT Contents	97	124	<EOF>	1258	126	<BAD>	122	577

  – Sectors 121, 124, 126, and 122 are read. After each sector is read, its FAT entry is to find the next sector occupied by the file.
  – At the FAT entry for sector 122, we find the end-of-file marker <EOF>.

How many disk accesses are required to read this file?

7.5 Optical Disks

- Optical disks provide large storage capacities very inexpensively.
- They come in a number of varieties including CD-ROM, DVD, and WORM.
- Many large computer installations produce document output on optical disk rather than on paper. This idea is called COLD-- *Computer Output Laser Disk*.
- **It is estimated that optical disks can endure for a hundred years. Other media are good for only a decade-- at best**.

7.5 Optical Disks

- CD-ROMs were designed by the music industry in the 1980s, and later adapted to data.
- This history is reflected by the fact that data is recorded in a single spiral track, starting from the center of the disk and spanning outward.
- Binary ones and zeros are delineated by bumps in the polycarbonate disk substrate. The transitions between **pits** and **lands** define binary ones.
- If you could unravel a full CD-ROM track, it would be nearly **five miles** long!

7.5 Optical Disks

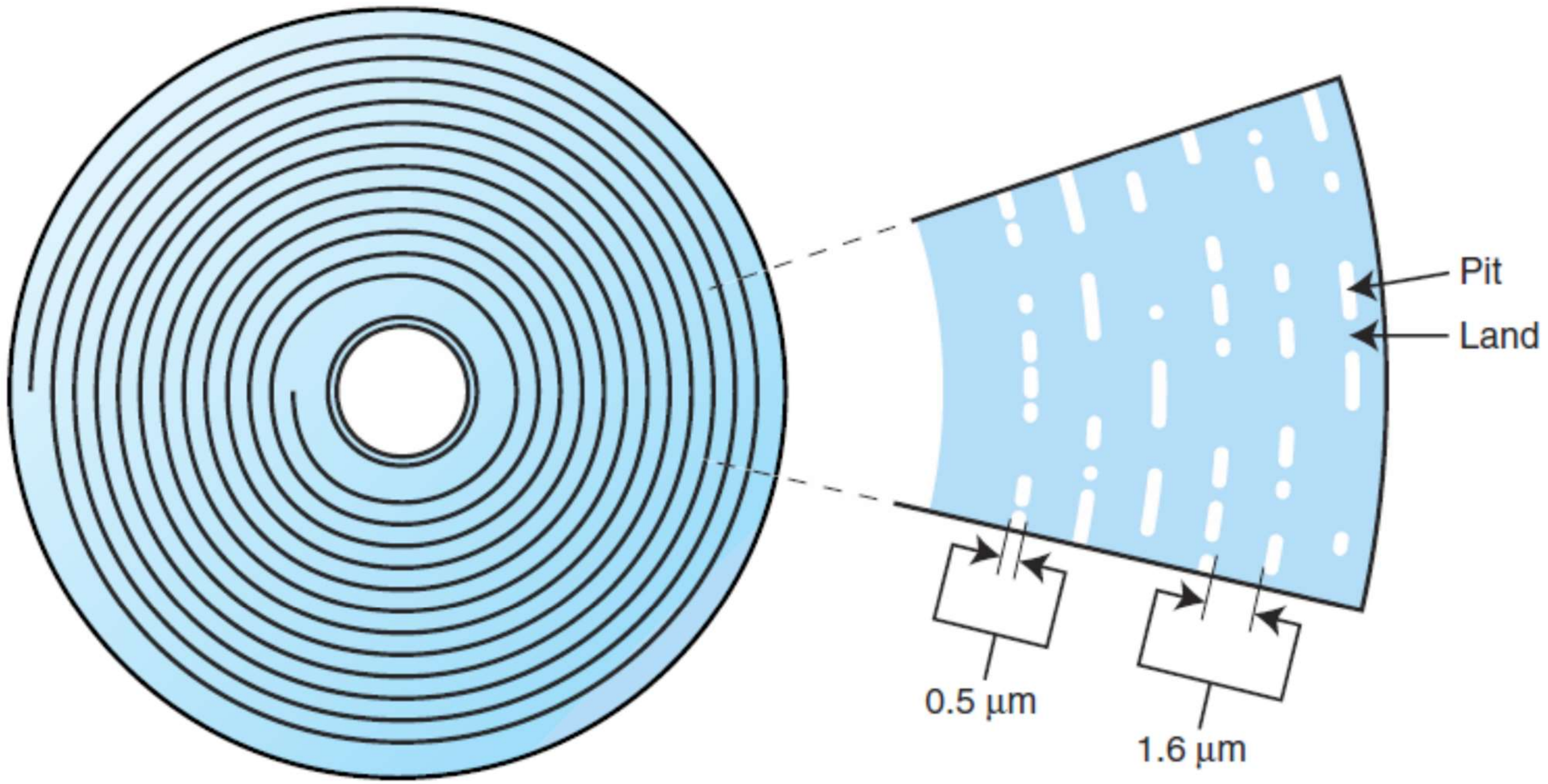


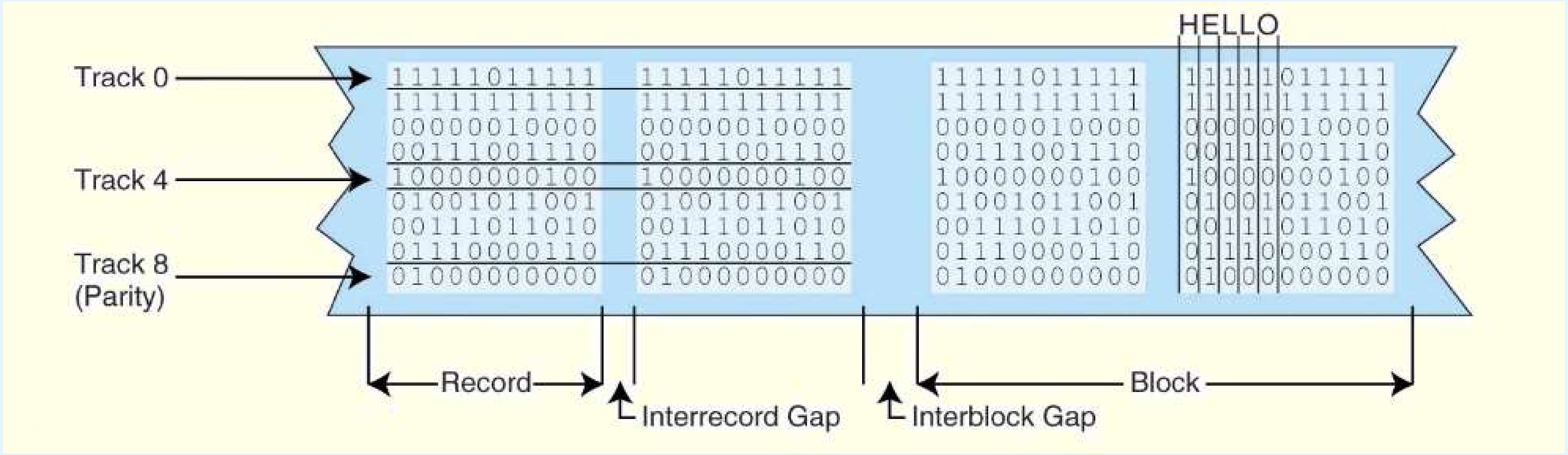
FIGURE 7.14 CD Track Spiral and Track Enlargement

7.5 Optical Disks

- The logical data format for a CD-ROM is much more complex than that of a magnetic disk. (See the text for details.)
- Different formats are provided for data and music.
- Two levels of error correction are provided for the data format.
- DVDs can be thought of as quad-density CDs.
- Where a CD-ROM can hold at most 650MB of data, DVDs can hold as much as 8.54GB.
- It is possible that someday DVDs will make CDs obsolete.

7.6 Magnetic Tape

- First-generation magnetic tape was not much more than wide analog recording tape, having capacities under 11MB.
- Data was usually written in nine vertical tracks:

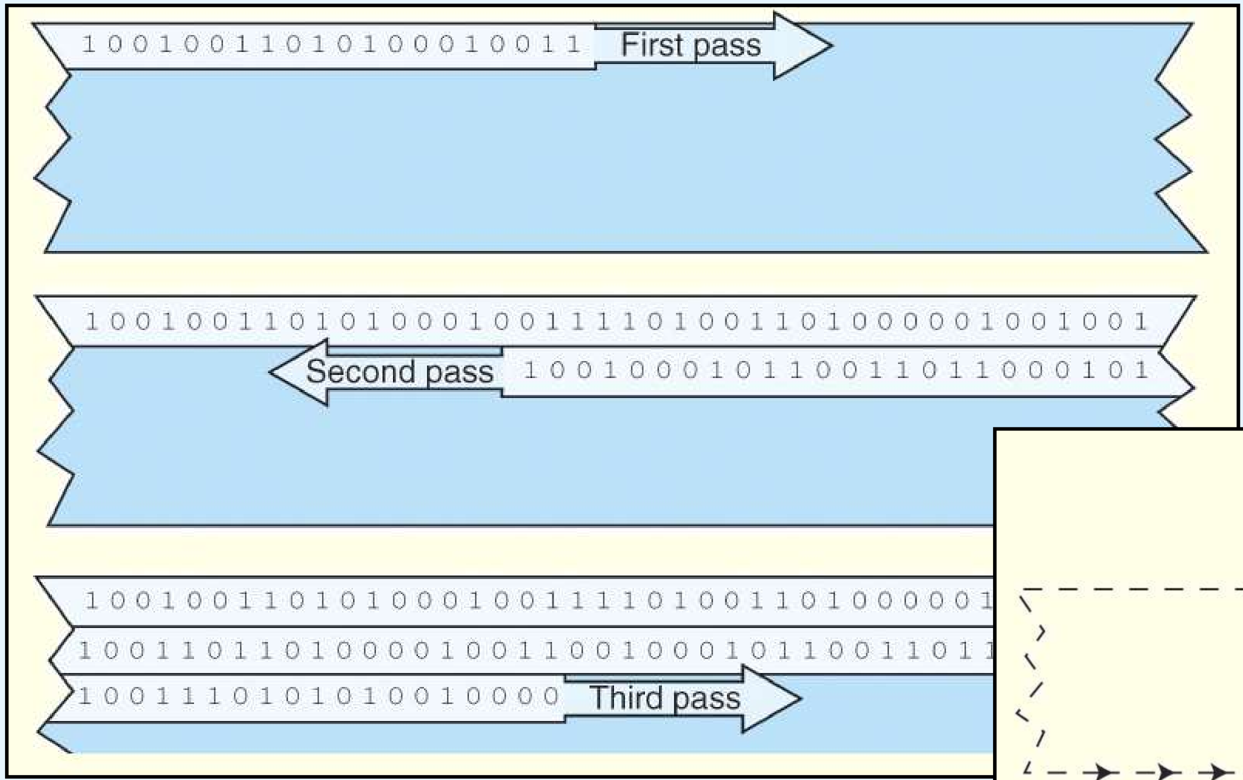


7.6 Magnetic Tape

- Today’s tapes are digital, and provide multiple gigabytes of data storage.
- Two dominant recording methods are *serpentine* and *helical scan*, which are distinguished by how the read-write head passes over the recording medium.
- Serpentine recording is used in *digital linear tape* (DLT) and *Quarter inch cartridge* (QIC) tape systems.
- *Digital audio tape* (DAT) systems employ helical scan recording.

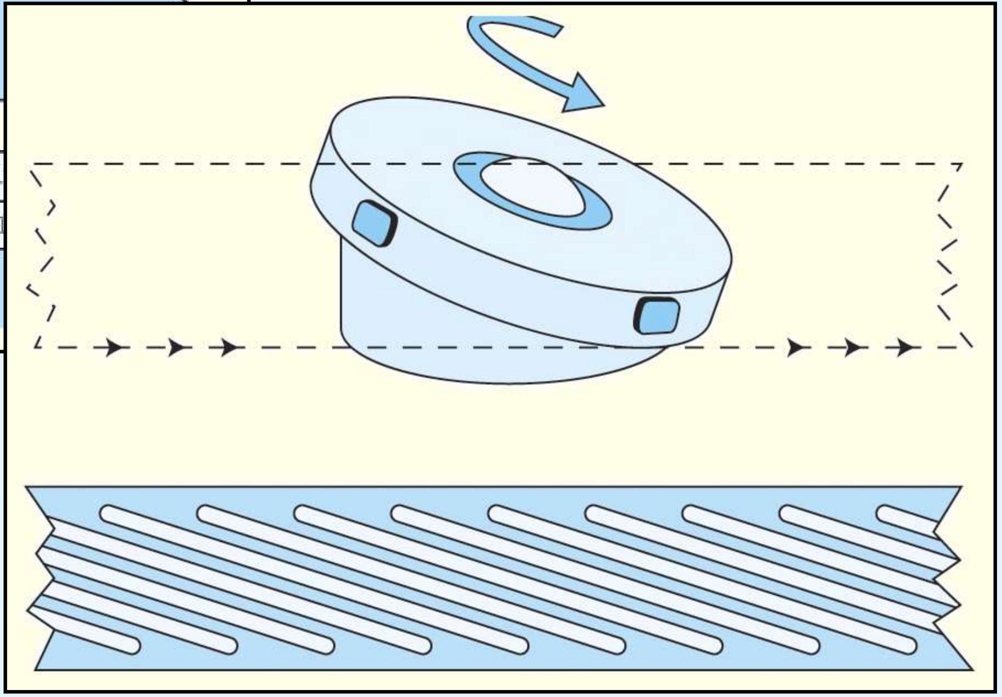
These two recording methods are shown on the next slide.

7.6 Magnetic Tape



← **Serpentine**

Helical Scan →

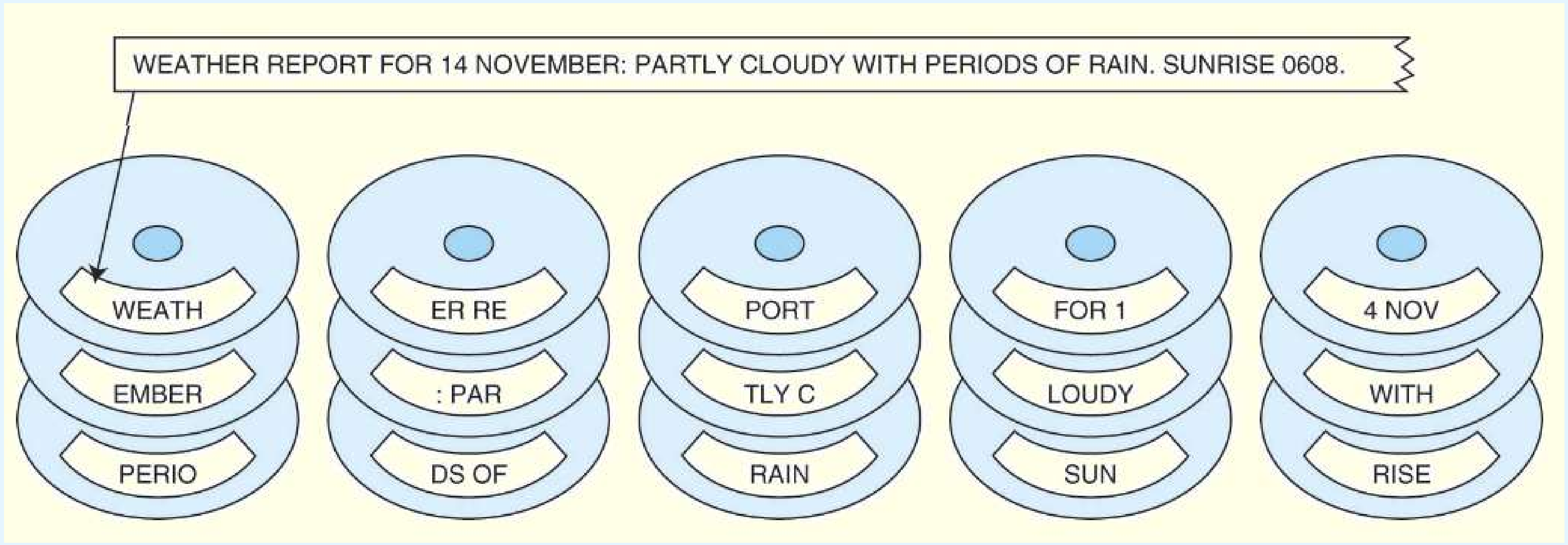


7.7 RAID

- RAID(独立硬盘冗余阵列), an acronym for *Redundant Array of Independent Disks* was invented to address problems of **disk reliability, cost, and performance**.
- In RAID, data is stored across many disks, with extra disks added to the array to **provide error correction** (redundancy).
- The inventors of RAID, David Patterson, Garth Gibson, and Randy Katz, provided a RAID taxonomy that has persisted for a quarter of a century, despite many efforts to redefine it.

7.7 RAID

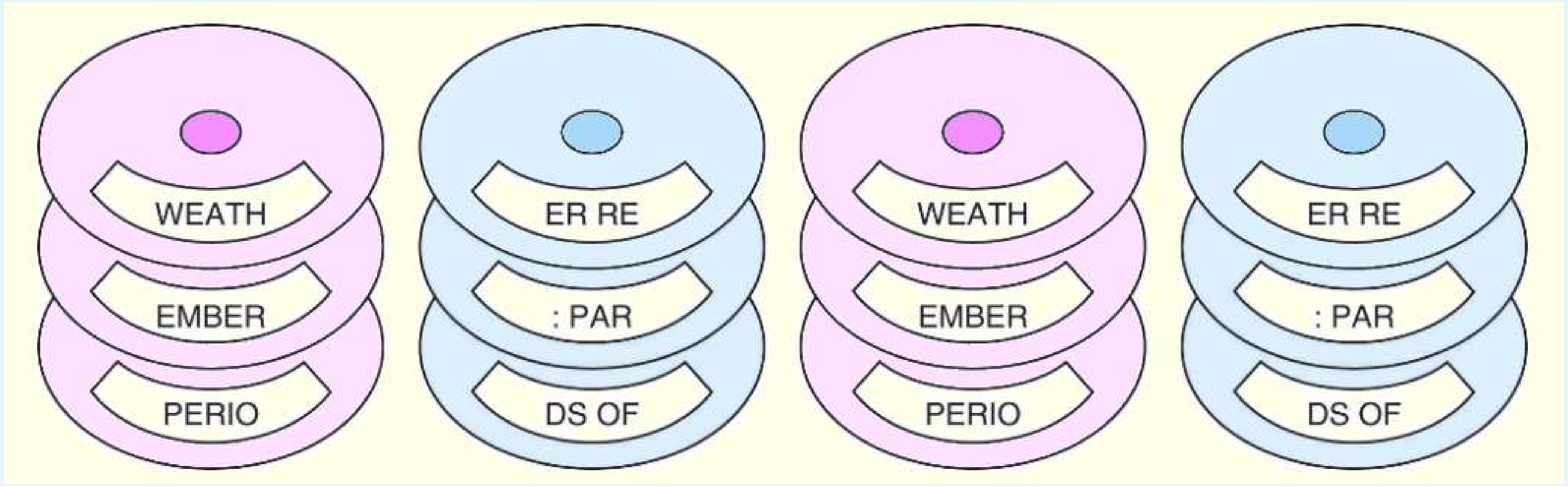
- RAID Level 0, also known as *drive spanning*, provides improved performance, but no redundancy.
  – Data is written in blocks across the entire array


WEATHER REPORT FOR 14 NOVEMBER: PARTLY CLOUDY WITH PERIODS OF RAIN. SUNRISE 0608.
WEATH
EMBER
PERIO
ER RE
: PAR
DS OF
PORT
TLY C
RAIN
FOR 1
LOUDY
SUN
4 NOV
WITH
RISE

  – The disadvantage of RAID 0 is in its low reliability.

7.7 RAID

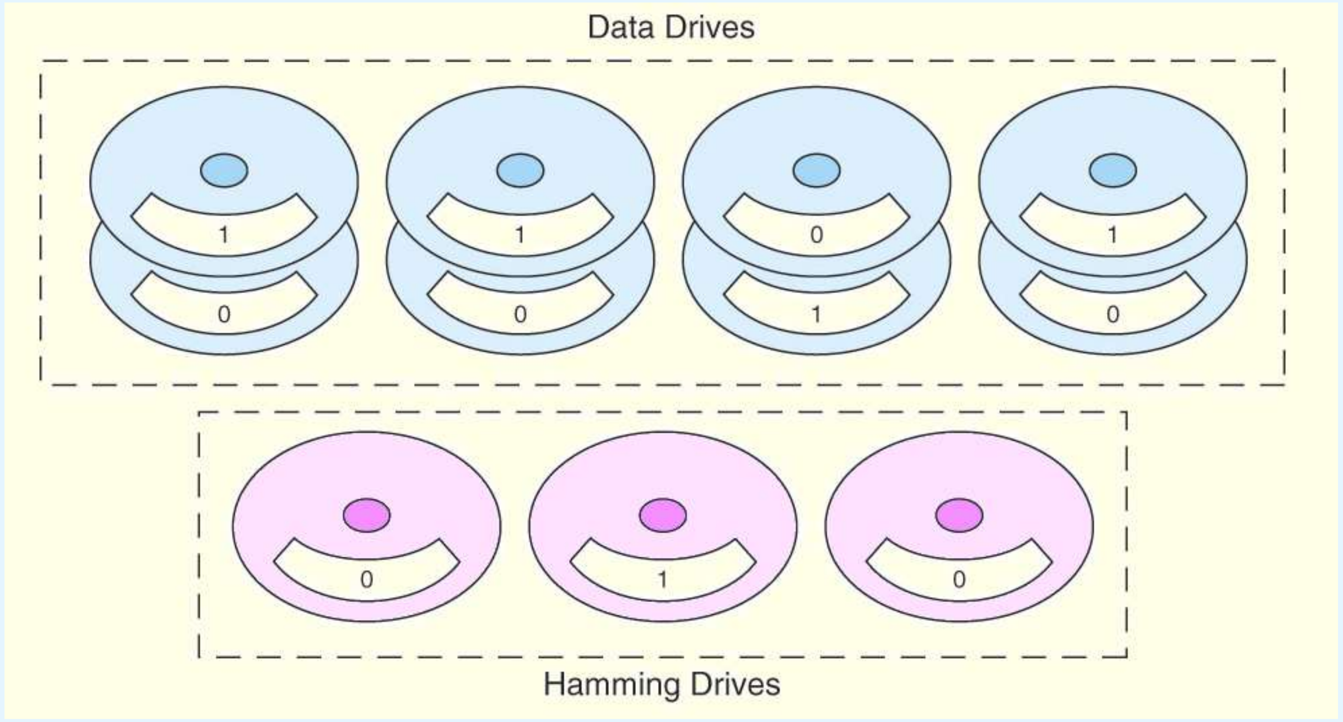
- RAID Level 1, also known as *disk mirroring*, provides 100% redundancy, and good performance.
  - Two matched sets of disks contain the same data.



  - The disadvantage of RAID 1 is cost.

7.7 RAID

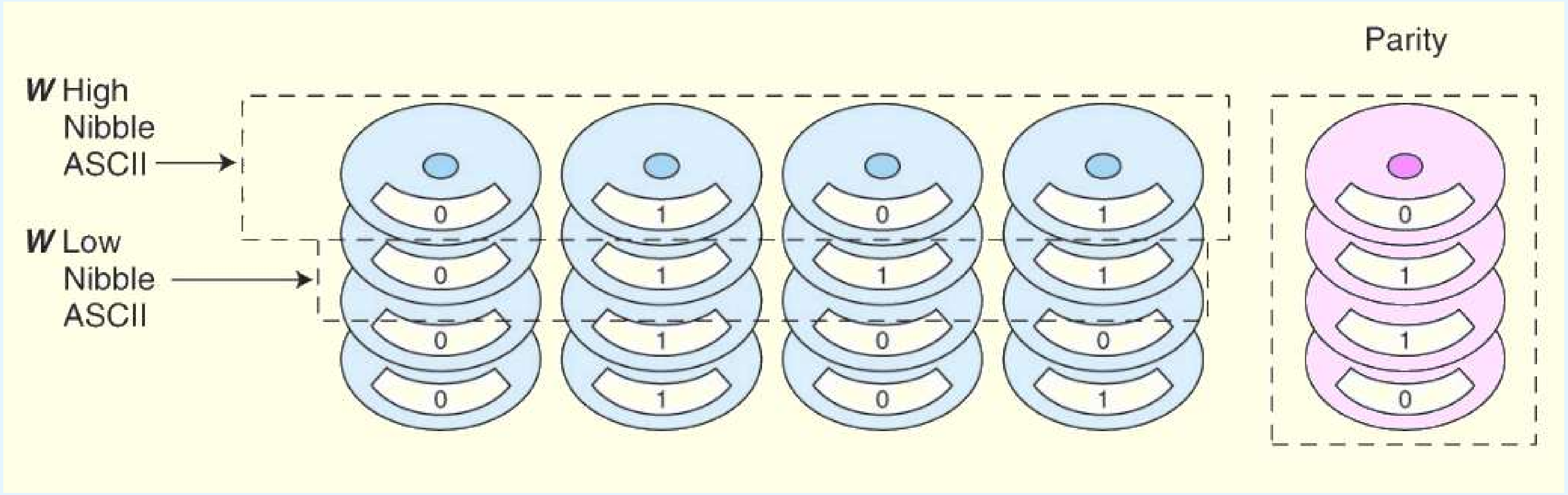
- A RAID Level 2 configuration consists of a set of data drives, and a set of Hamming code drives.
  - Hamming code drives provide error correction for the data drives.
  - 8bits data and 3bits hamming code



  - RAID 2 performance is poor and the cost is relatively high.

7.7 RAID

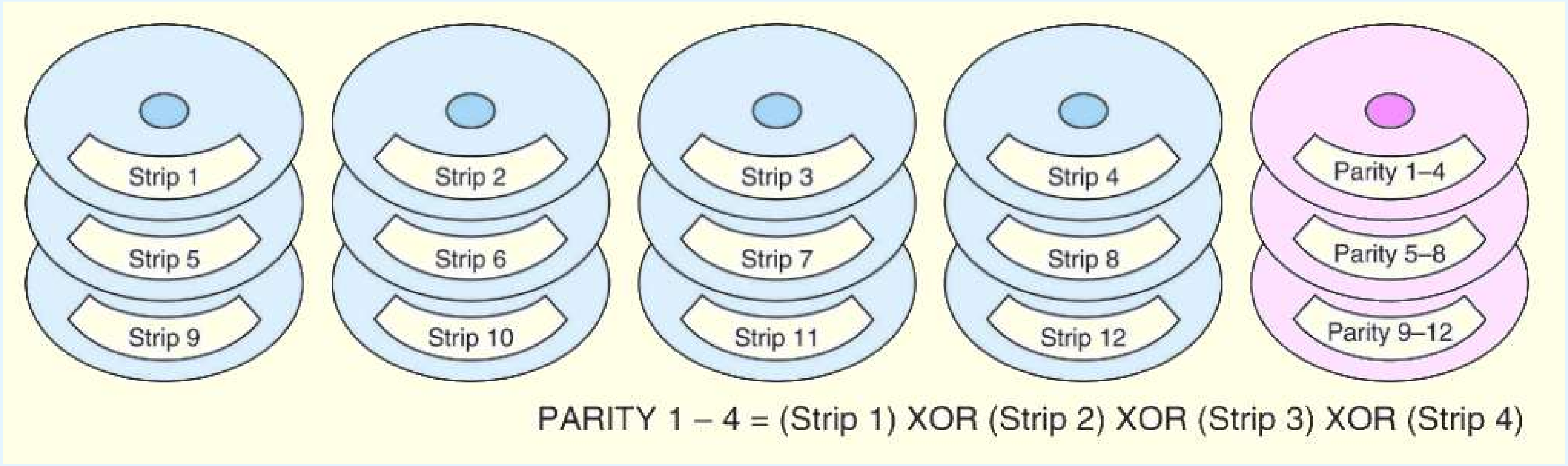
- RAID Level 3 stripes bits across a set of data drives and provides a separate disk for parity.
  - Parity is the XOR of the data bits.(可以纠正确定位置的数据丢失)



  - RAID 3 is not suitable for commercial applications, but is good for personal systems.

7.7 RAID

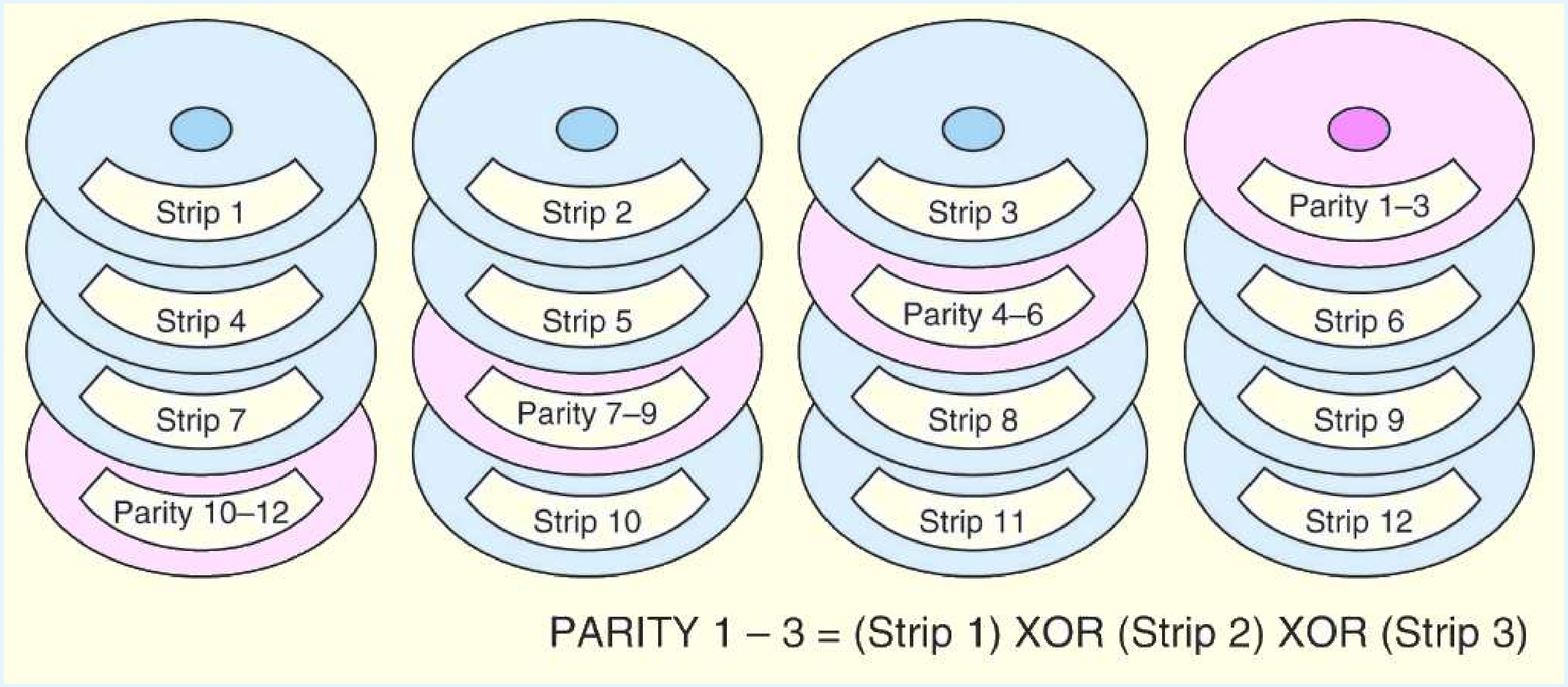
- RAID Level 4 is like adding parity disks to RAID 0.
  - Data is written in blocks across the data disks, and a parity block is written to the redundant drive.



  - RAID 4 would be feasible if all record blocks were the same size.
  - 不能同时写入不同的strip数据，性能低

7.7 RAID

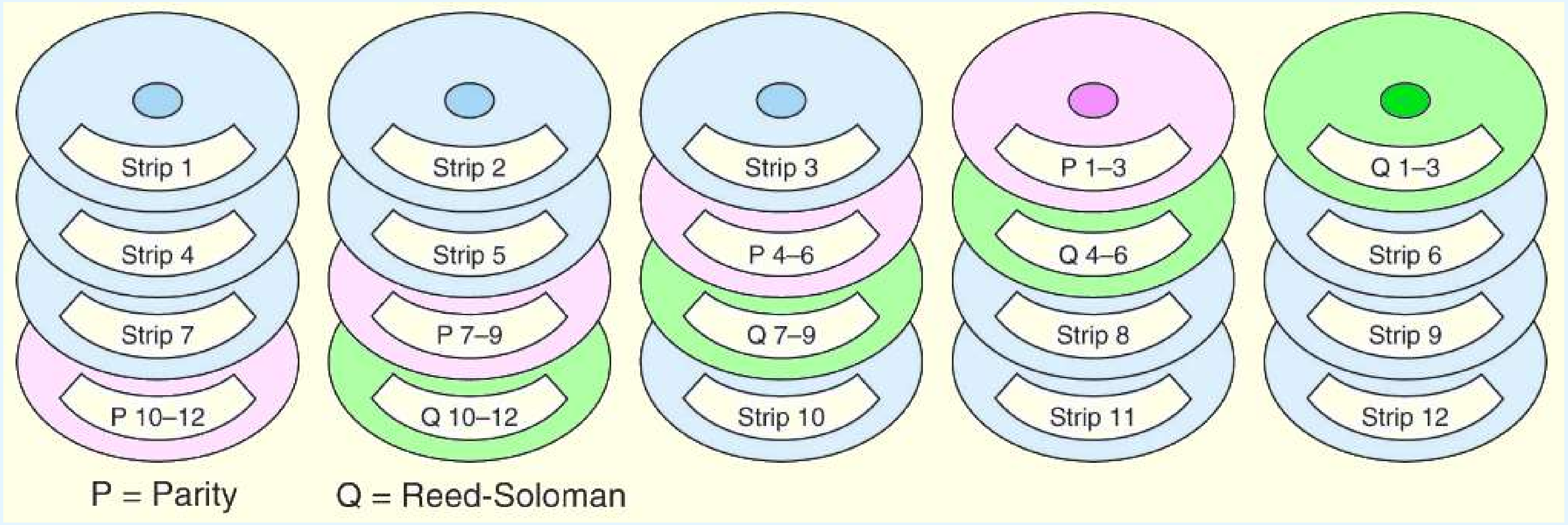
- RAID Level 5 is RAID 4 with distributed parity.
  - With distributed parity, some accesses can be serviced concurrently, giving good performance and high reliability.



  - RAID 5 is used in many commercial systems, file and application servers, email and news servers, database servers

7.7 RAID

- RAID Level 6 carries two levels of error protection over striped data: Reed-Soloman and parity.
  - It can tolerate **the loss of two disks**.



  - RAID 6 is write-intensive, but highly fault-tolerant.

7.7 RAID

- Large systems consisting of many drive arrays may employ various RAID levels, depending on the criticality of the data on the drives.
  - A disk array that provides program workspace (say for file sorting) does not require high fault tolerance.
- Critical, high-throughput files can benefit from combining RAID 0 with RAID 1, called RAID 10.
- **Keep in mind that a higher RAID level does not necessarily mean a "better" RAID level. It all depends upon the needs of the applications that use the disks.**

7.8 Data Compression

- Data compression is important to storage systems because it allows more bytes to be packed into a given storage medium than when the data is uncompressed.
- Some storage devices (notably tape) compress data automatically as it is written, resulting in less tape consumption and significantly faster backup operations.
- Compression also reduces Internet file transfer time, saving time and communications bandwidth.

7.8 Data Compression

- A good metric for compression is the *compression factor* (or *compression ratio*) given by:

$$\text{Compression factor} = 1 - \left[\frac{\text{compressed size}}{\text{uncompressed size}}\right] \times 100\%$$

- If we have a 100KB file that we compress to 40KB, we have a compression factor of:

$$1 - \left[\frac{40\text{KB}}{100\text{KB}}\right] \times 100\% = 60\%$$

7.8 Data Compression

- Compression is achieved by removing data redundancy while preserving information content.
- The information content of a group of bytes (a message) is its *entropy*.
  - Data with low entropy permit a larger compression ratio than data with high entropy.
- Entropy, *H*, is a function of symbol frequency. It is the weighted average of the number of bits required to encode the symbols of a message:

$$H = -P(x) \times \log_2 P(x)$$

7.8 Data Compression

- The entropy of the entire message is the sum of the individual symbol entropies.

$$\sum -P(x_i) \times \log_2 P(x_i)$$

- The average redundancy for each character in a message of length l is given by:

$$\sum P(x_i) \times l_i - \sum -P(x_i) \times \log_2 P(x_i)$$

7.8 Data Compression

- Consider the message: **`HELLO WORLD!`**
  - The letter **`L`** has a probability of 3/12 = 1/4 of appearing in this message. The number of bits required to encode this symbol is $-\log_2(1/4) = 2$.
- Using our formula, $\sum -P(x) \times \log_2 P(x_i)$, the average entropy of the entire message is 3.022.
  - This means that the theoretical minimum number of bits per character is 3.022.
- Theoretically, the message could be sent using only 37 bits. ($3.022 \times 12 = 36.26$)

7.8 Data Compression

- The entropy metric just described forms the basis for statistical data compression.
- Two widely-used statistical coding algorithms are *Huffman coding* and *arithmetic coding*.
- Huffman coding builds a binary tree from the letter frequencies in the message.
  - The binary symbols for each character are read directly from the tree.
- Symbols with the highest frequencies end up at the top of the tree, and result in the shortest codes.

An example is shown on the next slide.

7.8 Data Compression

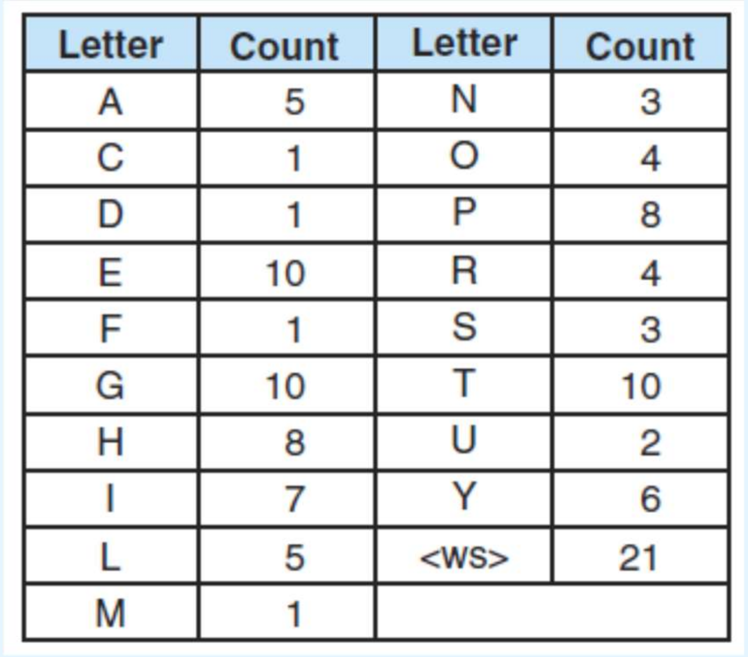
- Example.

  HIGGLETY PIGGLETY POP

  THE DOG HAS EATEN THE MOP

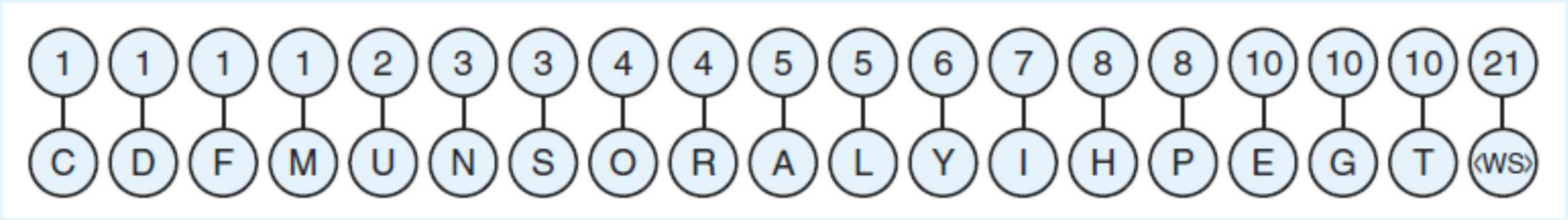
  THE PIGS IN A HURRY THE CATS IN A FLURRY

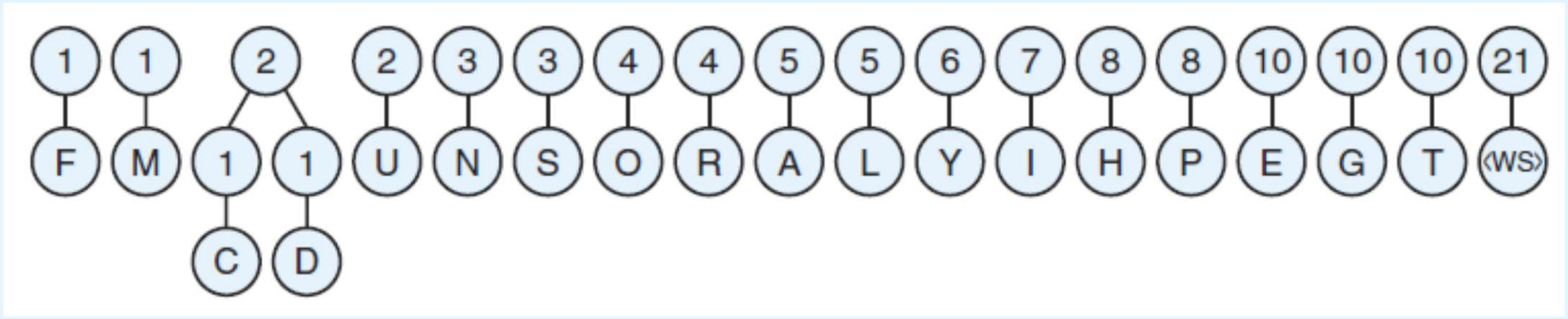
  HIGGLETY PIGGLETY POP

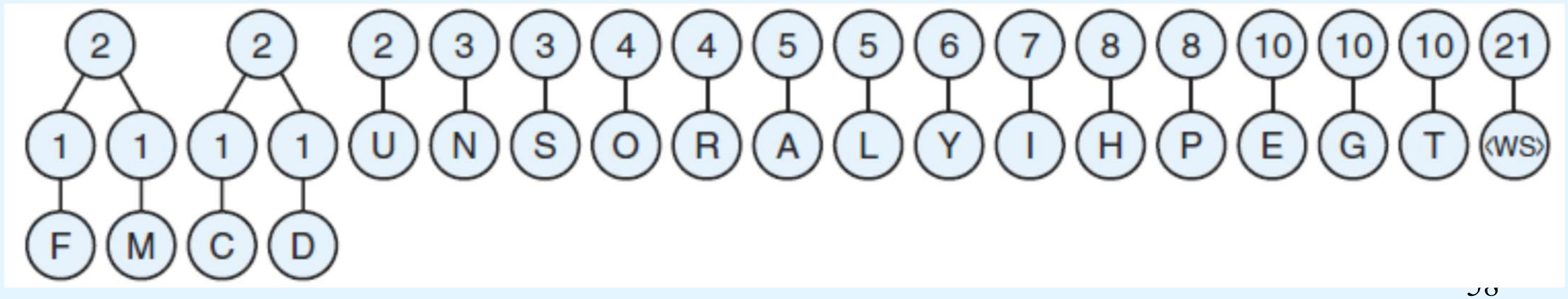


Letter	Count	Letter	Count
A	5	N	3
C	1	O	4
D	1	P	8
E	10	R	4
F	1	S	3
G	10	T	10
H	8	U	2
I	7	Y	6
L	5	<WS>	21
M	1		

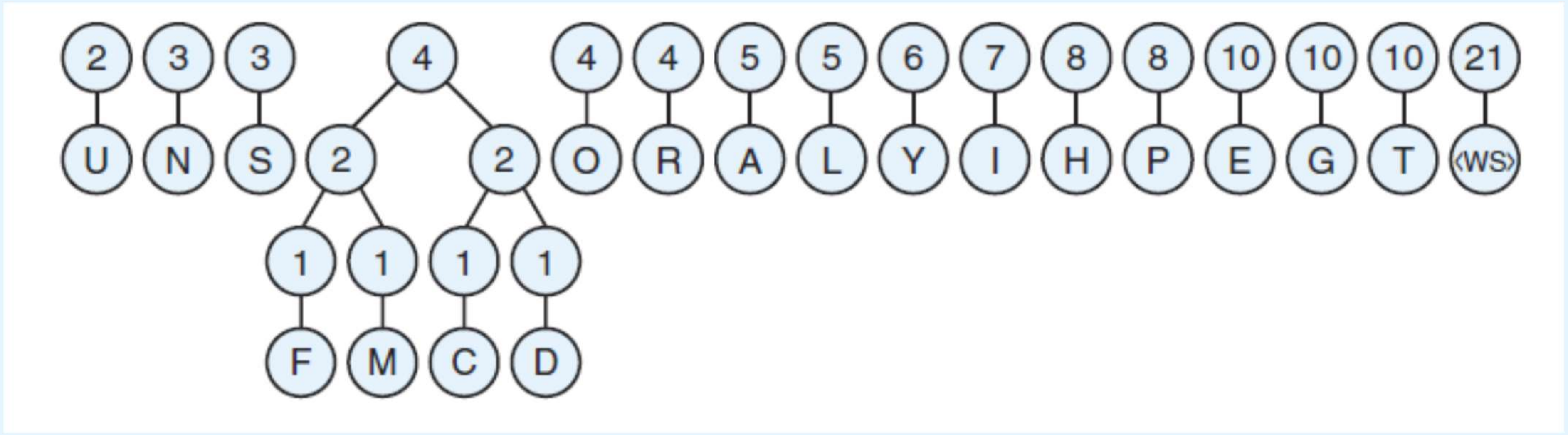
7.8 Data Compression

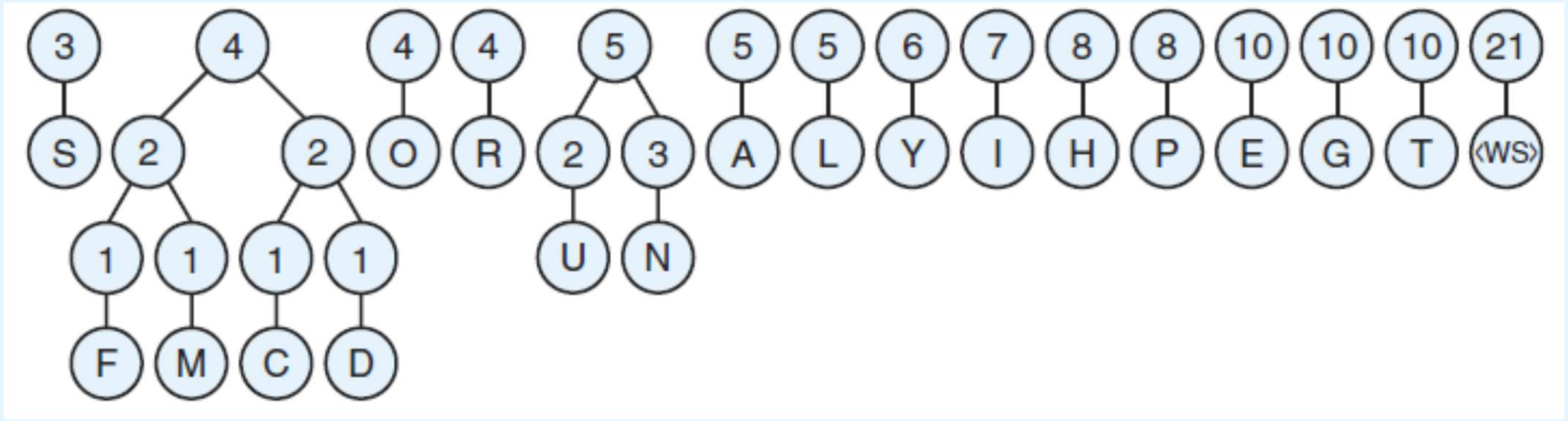




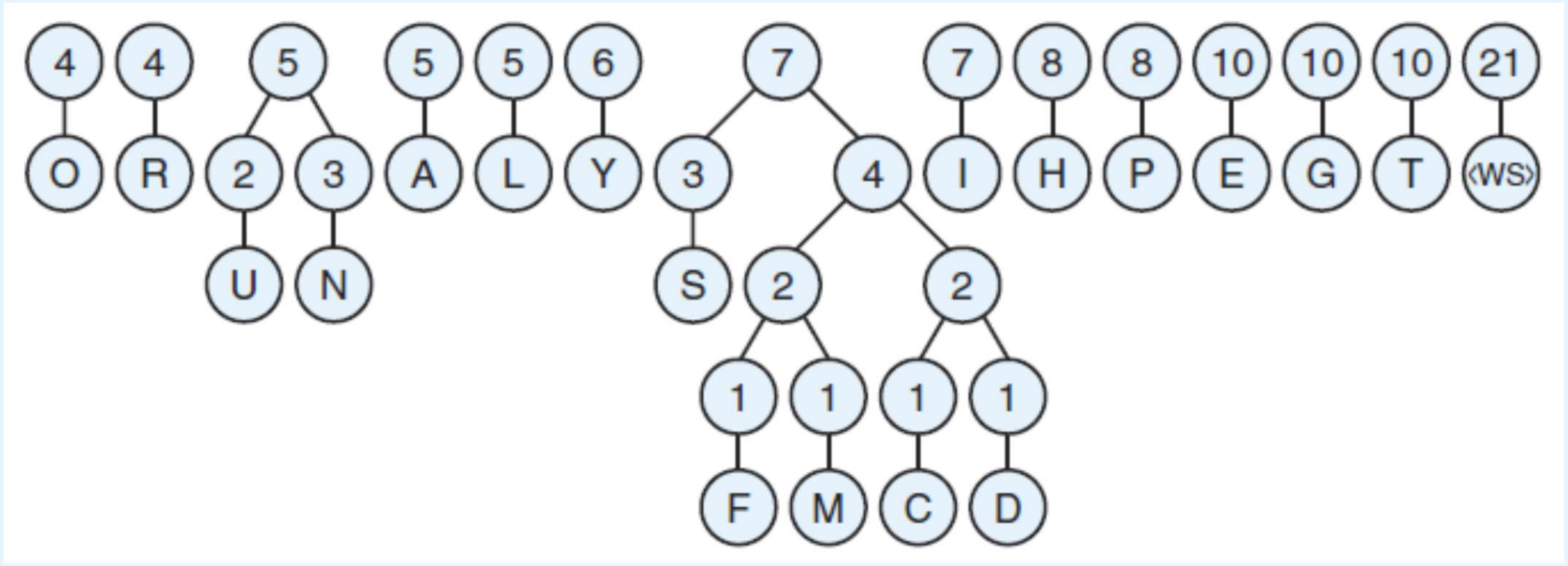


7.8 Data Compression

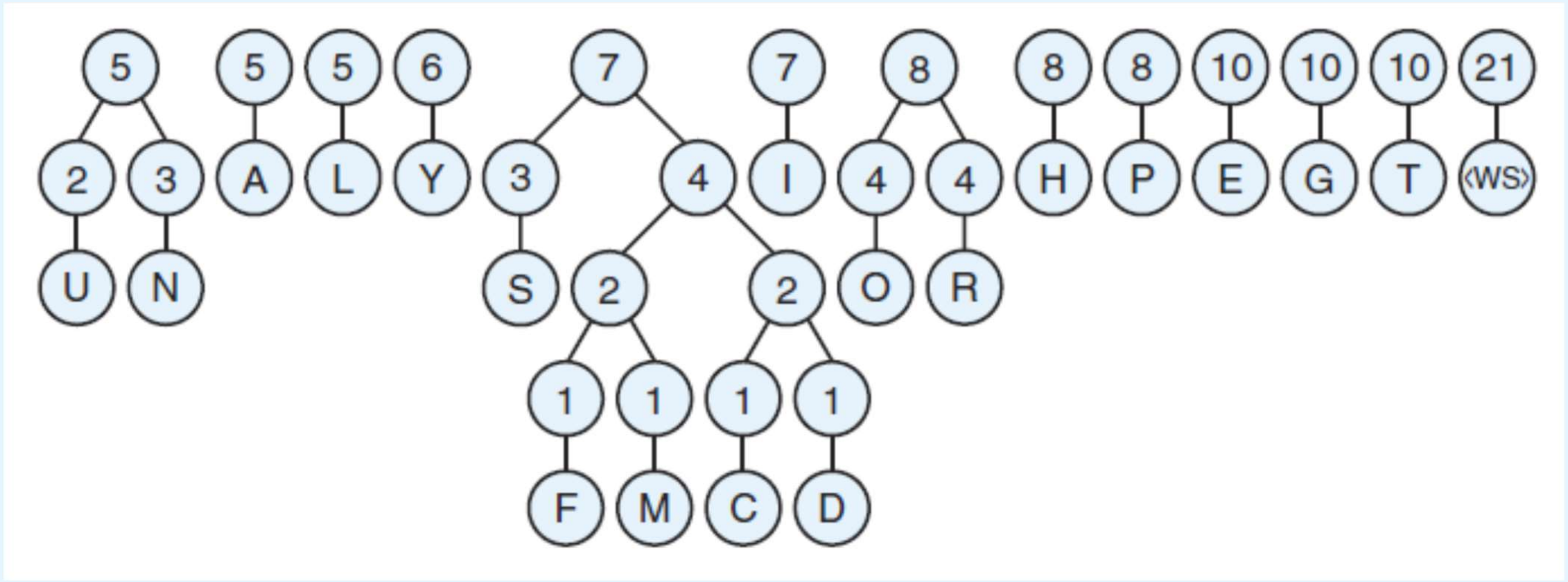




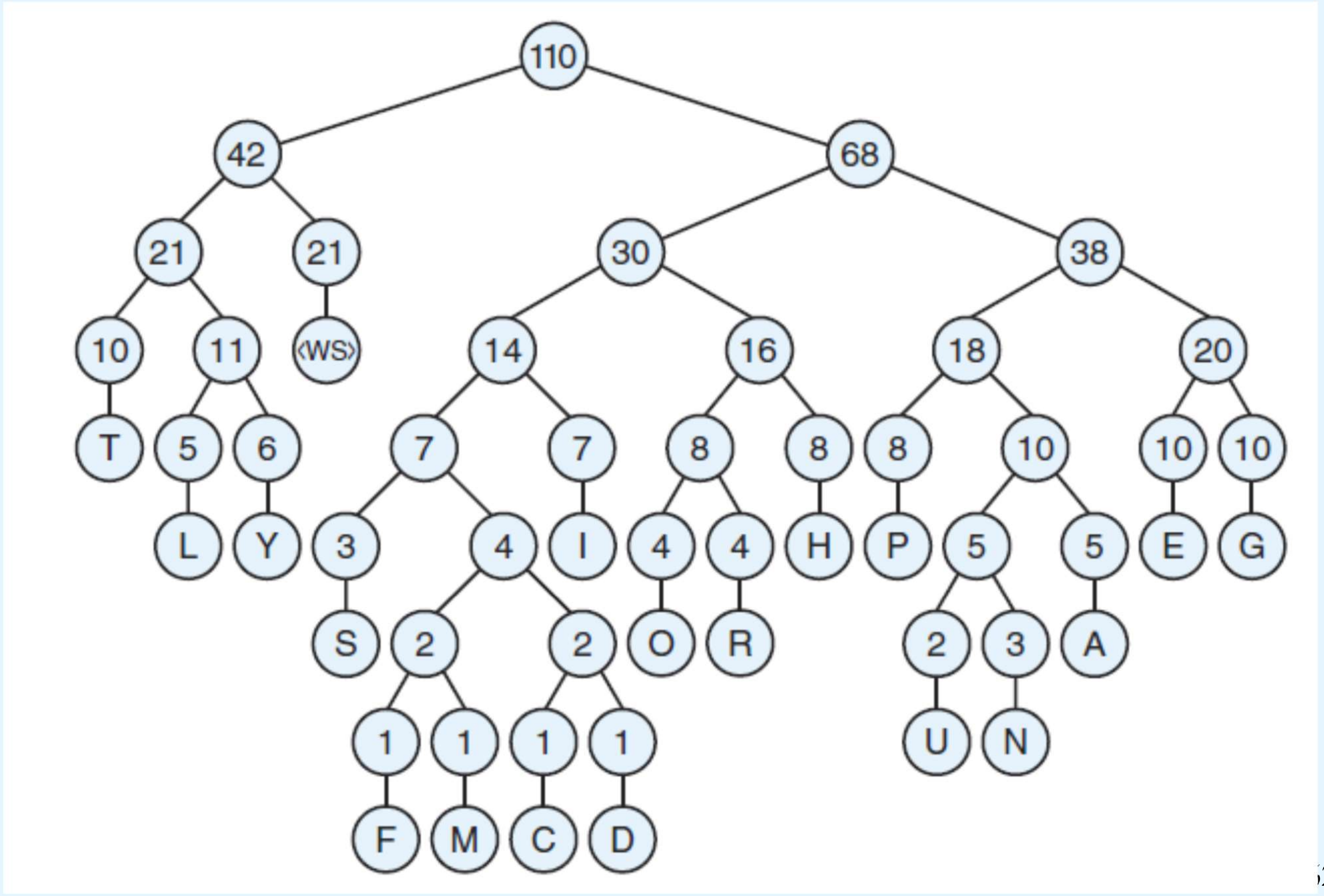
7.8 Data Compression



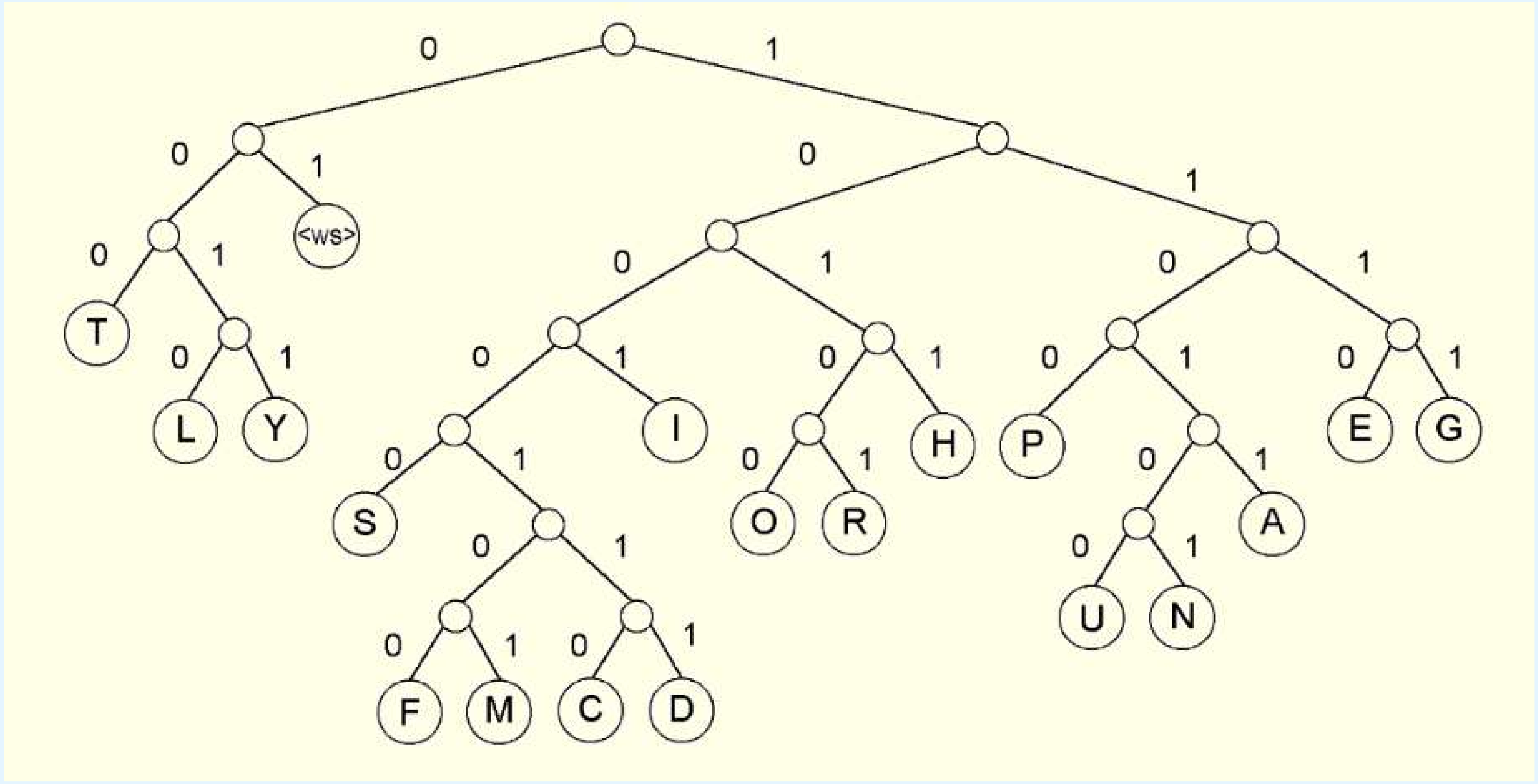
7.8 Data Compression



7.8 Data Compression

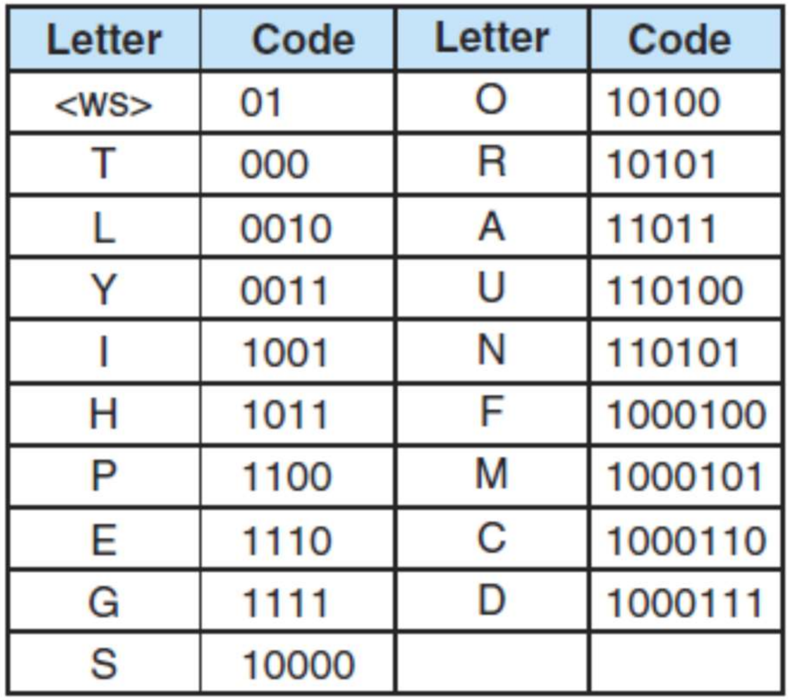


7.8 Data Compression



labeling every right branch with a binary 1, then each left branch with a binary 0

7.8 Data Compression



Letter	Code	Letter	Code
<WS>	01	O	10100
T	000	R	10101
L	0010	A	11011
Y	0011	U	110100
I	1001	N	110101
H	1011	F	1000100
P	1100	M	1000101
E	1110	C	1000110
G	1111	D	1000111
S	10000		

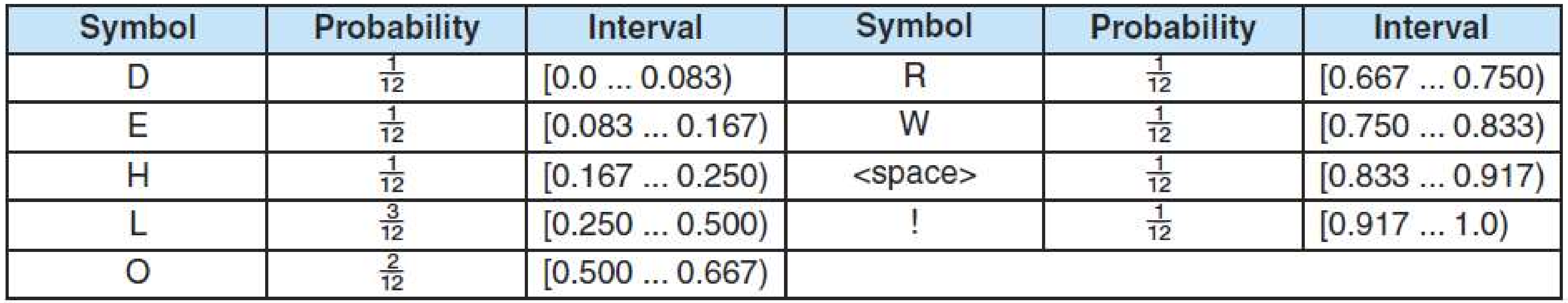
TABLE 7.2 **The Coding Scheme**

7.8 Data Compression

- The second type of statistical coding, arithmetic coding, partitions the real number interval between 0 and 1 into segments according to symbol probabilities.
  - An abbreviated algorithm for this process is given in the text.
- Arithmetic coding is computationally intensive and it runs the risk of causing divide underflow.
- Variations in floating-point representation among various systems can also cause the terminal condition (a zero value) to be missed.

7.8 Data Compression

- Huffman coding cannot always achieve theoretically optimal compression because it is restricted to using an integral number of bits in the resulting code

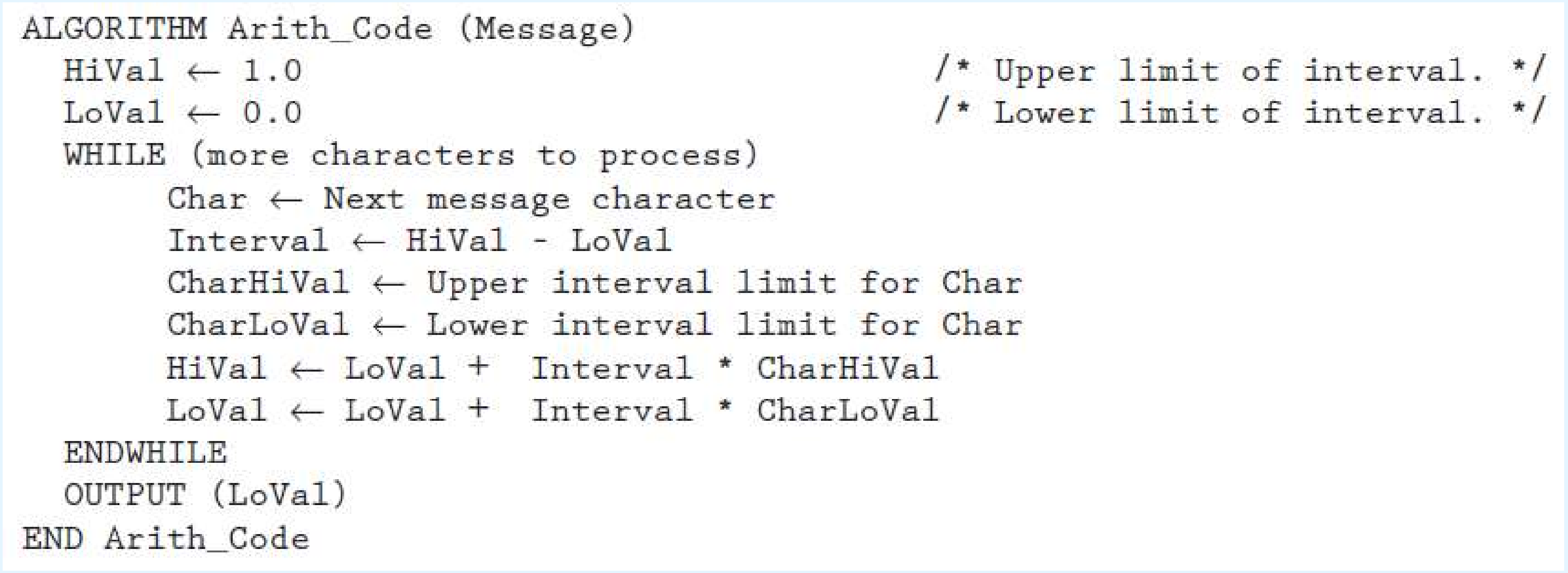


Symbol	Probability	Interval	Symbol	Probability	Interval
D	$\frac{1}{12}$	[0.0 ... 0.083)	R	$\frac{1}{12}$	[0.667 ... 0.750)
E	$\frac{1}{12}$	[0.083 ... 0.167)	W	$\frac{1}{12}$	[0.750 ... 0.833)
H	$\frac{1}{12}$	[0.167 ... 0.250)	<space>	$\frac{1}{12}$	[0.833 ... 0.917)
L	$\frac{3}{12}$	[0.250 ... 0.500)	!	$\frac{1}{12}$	[0.917 ... 1.0)
O	$\frac{2}{12}$	[0.500 ... 0.667)			

TABLE 7.3 **Probability Interval Mapping for *HELLO WORLD!***

7.8 Data Compression

- A trace of the pseudocode for *HELLO WORLD! is given below*



```
ALGORITHM Arith_Code (Message)
  HiVal ← 1.0                                  /* Upper limit of interval. */
  LoVal ← 0.0                                  /* Lower limit of interval. */
  WHILE (more characters to process)
       Char ← Next message character
       Interval ← HiVal - LoVal
       CharHiVal ← Upper interval limit for Char
       CharLoVal ← Lower interval limit for Char
       HiVal ← LoVal +  Interval * CharHiVal
       LoVal ← LoVal +  Interval * CharLoVal
  ENDWHILE
  OUTPUT (LoVal)
END Arith_Code
```

7.8 Data Compression

Symbol	Interval	CharLoVal	CharHiVal	LoVal	HiVal
				0.0	1.0
H	1.0	0.167	0.25	0.167	0.25
E	0.083	0.083	0.167	0.173889	0.180861
L	0.006972	0.25	0.5	0.1756320	0.1773750
L	0.001743	0.25	0.5	0.17606775	0.17650350
O	0.00043575	0.5	0.667	0.176285625	0.176358395
<sp>	0.00007277025	0.833	0.917	0.1763462426	0.1763523553
W	0.00000611270	0.75	0.833	0.1763508271	0.1763513345
O	0.00000050735	0.5	0.667	0.1763510808	0.1763511655
R	0.00000008473	0.667	0.75	0.1763511373	0.1763511444
L	0.00000000703	0.25	0.5	0.1763511391	0.1763511409
D	0.00000000176	0	0.083	0.1763511391	0.1763511392
!	0.00000000015	0.917	1	0.176351139227	0.176351139239
				0.176351139227	

FIGURE 7.28 Encoding ***HELLO WORLD!*** with Arithmetic Coding

7.8 Data Compression

```
ALGORITHM Arith_Decode (CodedMsg)
    Finished ← FALSE
    WHILE NOT Finished
       FoundChar ← FALSE            /*  We could do this search much more    */
       WHILE NOT FoundChar          /* efficiently in a real implementation. */
          PossibleChar ← next symbol from the code table
          CharHiVal ← Upper interval limit for PossibleChar
          CharLoVal ← Lower interval limit for PossibleChar
          IF CodedMsg < CharHiVal AND CodedMsg > CharLoVal THEN
             FoundChar ← TRUE
          ENDIF                     / * We now have a character whose interval  */
       ENDWHILE                     /* surrounds the current message value.     */
       OUTPUT(Matching Char)
       Interval ← CharHiVal - CharLoVal
    CodedMsgInterval ← CodedMsg - CharLoVal
    CodedMsg ← CodedMsgInterval / Interval
    IF CodedMsg = 0.0 THEN
       Finished ← TRUE
    ENDIF
END WHILE
END Arith_Decode
```

7.8 Data Compression

Symbol	LowVal	HiVal	Interval	CodedMsg-Interval	CodedMsg
					0.176351139227
H	0.167	0.25	0.083	0.009351139227	0.112664328032
E	0.083	0.167	0.084	0.029664328032	0.353146762290
L	0.25	0.5	0.250	0.103146762290	0.412587049161
L	0.25	0.5	0.250	0.162587049161	0.650348196643
O	0.5	0.667	0.167	0.15034819664	0.90028860265
<sp>	0.833	0.917	0.084	0.0672886027	0.8010547935
W	0.75	0.833	0.083	0.051054793	0.615117994
O	0.5	0.667	0.167	0.11511799	0.6893293
R	0.667	0.75	0.083	0.0223293	0.2690278
L	0.25	0.5	0.250	0.019028	0.076111
D	0	0.083	0.083	0.0761	0.917
!	0.917	1	0.083	0.000	0.000

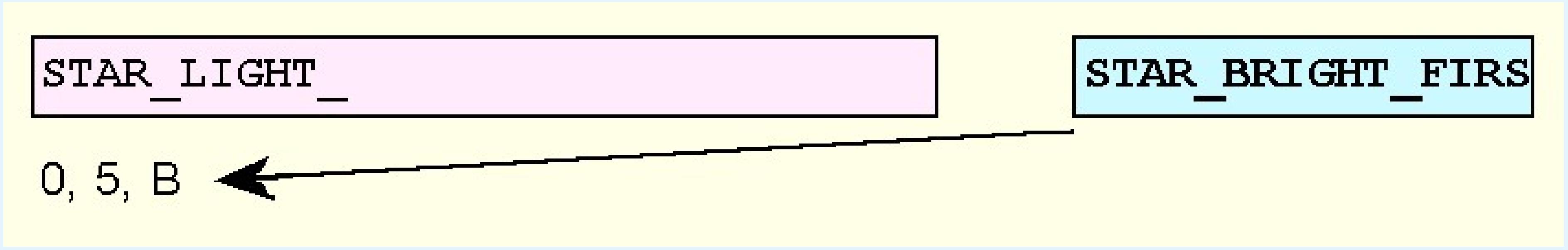
FIGURE 7.29 **A Trace of Decoding *HELLO WORLD!***

7.8 Data Compression

- For most data, statistical coding methods offer excellent compression ratios.
- Their main disadvantage is that they require two passes over the data to be encoded.
  - The first pass calculates probabilities, the second encodes the message.
- This approach is unacceptably slow for storage systems, where data must be read, written, and compressed within one pass over a file.

7.8 Data Compression

- *Ziv-Lempel* (LZ) dictionary systems solve the two-pass problem by using values in the data as a dictionary to encode itself.
- The LZ77 compression algorithm employs a text window in conjunction with a lookahead buffer.
  - The text window serves as the dictionary. If text is found in the lookahead buffer that matches text in the dictionary, the location and length of the text in the window is output.



7.8 Data Compression

- The LZ77 implementations include PKZIP and IBM’s RAMAC RVA 2 Turbo disk array.
  - The simplicity of LZ77 lends itself well to a hardware implementation.
- LZ78 is another dictionary coding system.
- It removes the LZ77 constraint of a fixed-size window. Instead, it creates a trie as the data is read.
- Where LZ77 uses pointers to locations in a dictionary, LZ78 uses pointers to nodes in the trie.

7.8 Data Compression

- GIF compression is a variant of LZ78, called LZW, for Lempel-Ziv-Welsh.
- It improves upon LZ78 through its efficient management of the size of the trie.
- Terry Welsh, the designer of LZW, was employed by the Unisys Corporation when he created the algorithm, and Unisys subsequently patented it.
- Owing to royalty disputes, development of another algorithm PNG, was hastened.

7.8 Data Compression

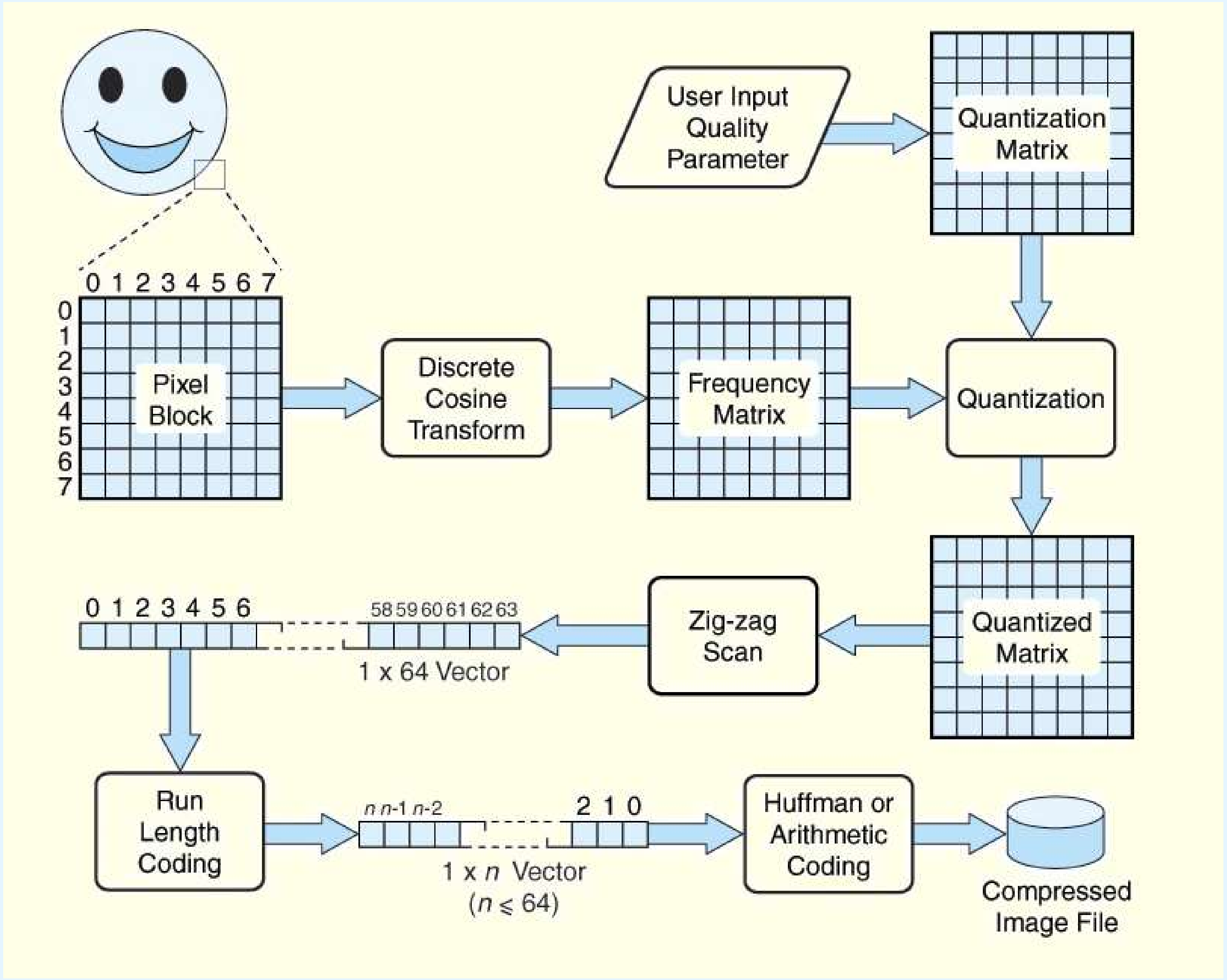
- PNG employs two types of compression, first a Huffman algorithm is applied, which is followed by LZ77 compression.
- The advantage that GIF holds over PNG, is that GIF supports multiple images in one file.
- MNG is an extension of PNG that supports multiple images in one file.
- GIF, PNG, and MNG are primarily used for graphics compression. To compress larger, photographic images, JEPG is often more suitable.

7.8 Data Compression

- Photographic images incorporate a great deal of information. However, much of that information can be lost without objectionable deterioration in image quality.
- With this in mind, JPEG allows user-selectable image quality, but even at the "best" quality levels, JPEG makes an image file smaller owing to its multiple-step compression algorithm.
- It's important to remember that JPEG is lossy, even at the highest quality setting. It should be used only when the loss can be tolerated.

The JPEG algorithm is illustrated on the next slide.

7.8 Data Compression



Chapter 7 Conclusion

- I/O systems are critical to the overall performance of a computer system.
- Amdahl’s Law quantifies this assertion.
- I/O systems consist of memory blocks, cabling, control circuitry, interfaces, and media.
- I/O control methods include programmed I/O, interrupt-based I/O, DMA, and channel I/O.
- Buses require control lines, a clock, and data lines. Timing diagrams specify operational details.

Chapter 7 Conclusion

- Magnetic disk is the principal form of durable storage.
- Disk performance metrics include seek time, rotational delay, and reliability estimates.
- Optical disks provide long-term storage for large amounts of data, although access is slow.
- Magnetic tape is also an archival medium. Recording methods are track-based, serpentine, and helical scan.

Chapter 7 Conclusion

- RAID gives disk systems improved performance and reliability. RAID 3 and RAID 5 are the most common.
- Many storage systems incorporate data compression.
- Two approaches to data compression are statistical data compression and dictionary systems.
- GIF, PNG, MNG, and JPEG are used for image compression.