5 Computer Architectures for Health Care and Biomedicine

Jonathan C. Silverstein and Ian T. Foster

After reading this chapter, you should know the answers to these questions:

- What are the components of computer architectures?
- How are medical data stored and manipulated in a computer?
- How can information be displayed clearly?
- What are the functions of a computer's operating system?
- What is the technical basis and business model of cloud computing?
- What advantages does using a database management system provide over storing and manipulating your own data directly?
- How do local area networks facilitate data sharing and communication within health care institutions?
- What are the maintenance advantages of Software-as-a-Service?
- How can the confidentiality of data stored in distributed computer systems be protected?
- How is the Internet used for medical applications today?
- How are wireless, mobile devices, federated and hosted systems changing the way the Internet will be used for biomedical applications?

5.1 Computer Architectures

Architectures are the designs or plans of systems. Architectures have both physical and conceptual aspects. Computer architectures for health care and biomedicine are the physical designs and conceptual plans of computers and information systems that are used in biomedical applications.

Health professionals and the general public encounter computers constantly. As electronic health records are increasingly deployed, clinicians use information systems to record medical observations, order drugs and laboratory tests, and review test results. Physicians and patients use personal computing environments such as desktop computers, laptops, and mobile devices to access the Internet, to search the medical literature, to communicate with colleagues and friends, and to do their clinical and administrative work. In fact, computers are ubiquitous, touching every aspect of human life, and increasingly image-intense, interactive, and collaborative.

J.C. Silverstein, MD, MS (✉)
Research Institute, NorthShore University HealthSystem, 1001 University Place, Evanston 60201, IL, USA
e-mail: jcs@uchicago.edu

I.T. Foster, PhD
Searle Chemistry Laboratory, Computation Institute, University of Chicago and Argonne National Laboratory, 5735 South Ellis Avenue, Chicago 60637, IL, USA
e-mail: foster@uchicago.edu

This chapter is adapted from an earlier version in the third edition authored by GioWiederhold and Thomas C. Rindfleisch.

Individual computers differ in speed, storage capacity, and cost; in the number of users that they can support; in the ways that they are interconnected; in the types of applications that they can run; in the way they are managed and shared; in the way they are secured; and in the way people interact with them. On the surface, the differences among computers can be bewildering, but the selection of appropriate hardware, software, and the architecture under which they are assembled is crucial to the success of computer applications. Despite these differences, however, computers use the same basic mechanisms to store and process information and to communicate with the outside world whether desktop, laptop, mobile device, gaming system, digital video recorder, or massive computer cluster. At the conceptual level, the similarities among all these computers greatly outweigh the differences.

In this chapter, we will cover fundamental concepts related to computer hardware, software, and distributed systems (multiple computers working together), including data acquisition, processing, communications, security, and sharing. We assume that you use computers but have not been concerned with their internal workings or how computers work together across the global Internet. Our aim is to give you the background necessary for understanding the underpinning technical architecture of the applications discussed in later chapters. We will describe the component parts that make up computers and their assembly into complex distributed systems that enable biomedical applications.

5.2 Hardware

Early computers were expensive to purchase and operate. Only large institutions could afford to acquire a computer and to develop its software. In the 1960s, the development of integrated circuits on silicon chips resulted in dramatic increases in computing power per dollar. Since that time, computer hardware has become dramatically smaller, faster, more reliable and personal. As computation, storage, and communication capabilities have increased so have data volumes, particularly genomic and imaging data. At the same time, software packages have been developed that remove much of the burden of writing the infrastructure of applications via encapsulation or abstraction of underlying software to “higher level” commands. The result is that computers are increasingly complex in their layered hardware and software architectures, but simpler for individuals to program and to use.

Essentially all modern general-purpose computers have similar base hardware (physical equipment) architectures. This is generally true whether they are large systems supporting many users, such as hospital information systems, individual personal computers, laptops, mobile devices, or even whole computers on one silicon “chip” embedded in medical devices. The scale of computing, memory, display and style of usage largely distinguish different individual hardware devices. Later we will discuss assemblies of computers for complex applications.

General computer architectures follow principles expressed by John von Neumann in 1945. Figure 5.1 illustrates the configuration of a simple **von Neumann machine**. Extending this to modern computers, they are composed of one or more

- **Central processing units** (**CPUs**) that perform general computation
- **Computer memories** that store programs and data that are being used actively by a CPU
- **Storage devices**, such as **magnetic disks** and tapes, **optical disks**, and **solid state drives**, that provide long-term storage for programs and data
- **Graphics processing units** (**GPUs**) that perform graphic displays and other highly parallel computations
- **Input** and **output** (I/O) devices, such as keyboards, pointing devices, touch screens, controllers, video displays, and printers, that facilitate user interaction and storage
- **Communication** equipment, such as network interfaces, that connect computers to networks of computers
- **Data buses**, electrical pathways that transport encoded information between these subsystems

Most computers are now manufactured with multiple CPUs on a single chip, and in some

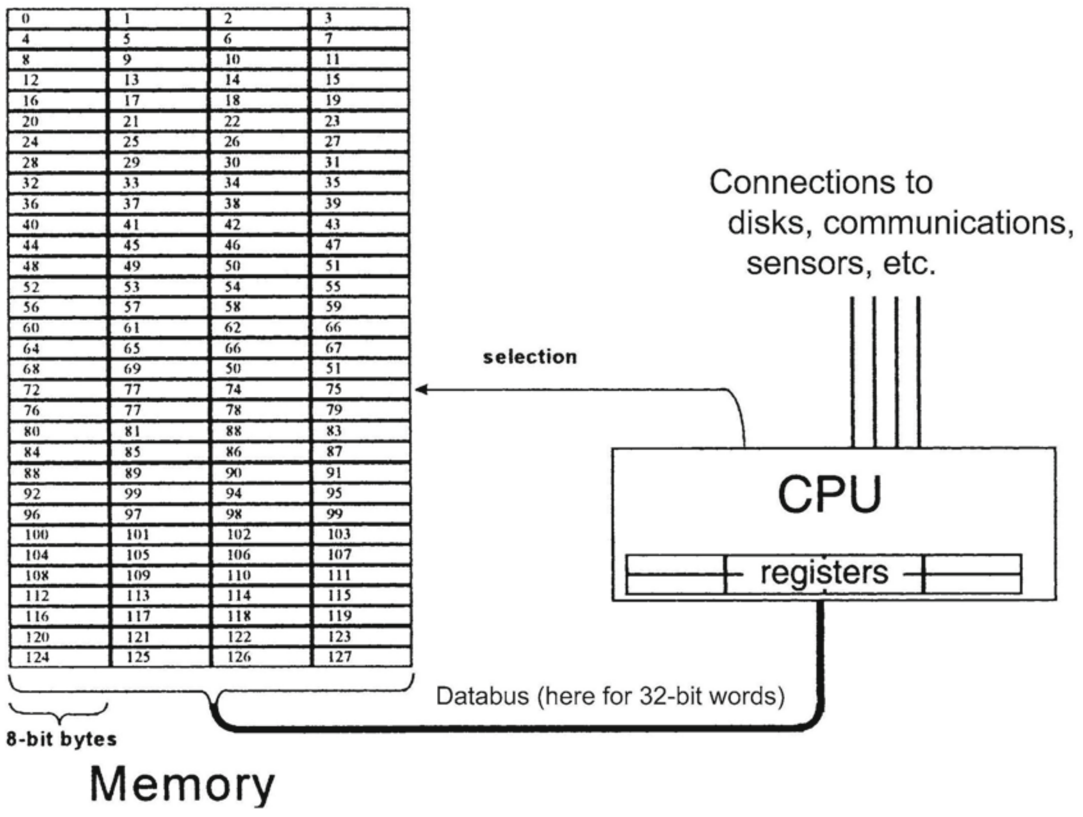


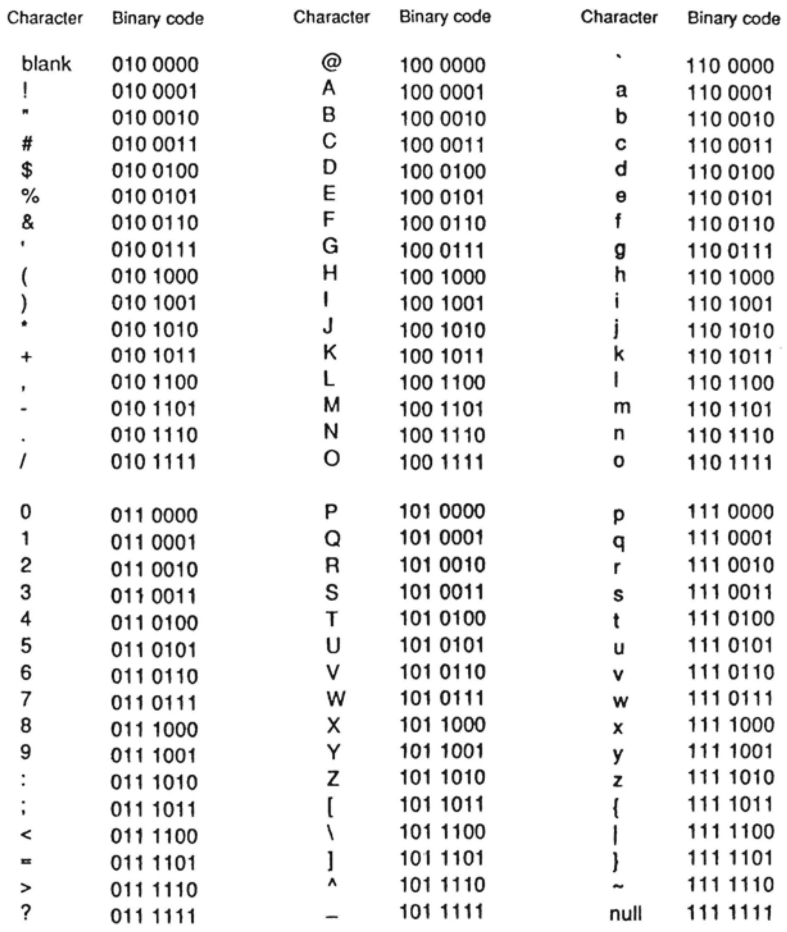
Fig. 5.1 The von Neumann model: the basic architecture of most modern computers. The computer comprises a single central processing unit (CPU), an area for memory, and a data bus for transferring data between the two

cases multiple GPUs, as well as multiple layers of memory, storage, I/O devices and communication interfaces. Multiple interconnected CPUs with shared memory layers further enable **parallel processing** (performing multiple computations simultaneously). The challenge then is for the software to distribute the computation across these units to gain a proportionate benefit.

5.2.1 Central Processing Unit

Although complete computer systems appear to be complex, the underlying principles are simple. A prime example is a processing unit itself. Here simple components can be carefully combined to create systems with impressive capabilities. The structuring principle is that of hierarchical organization: primitive units (electronic switches) are combined to form basic units that can store letters and numbers, add digits, and compare values with one another. The basic units are assembled into **registers** capable of storing and manipulating text and large numbers. These registers in turn are assembled into the larger functional units that make up the central component of a computer: the CPU.

The logical atomic element for all digital computers is the *binary digit* or **bit**. Each bit can assume one of two values: 0 or 1. An electronic switch that can be set to either of two states stores a single bit value. (Think of a light switch that can be either on or off.) These primitive units are the building blocks of computer systems. Sequences of bits (implemented as sequences of switches) are used to represent larger numbers and other kinds of information. For example, four switches can store 2^4, or 16, different combination of values: 0000, 0001, 0010, 0011, 0100, 0101, 0110, and so on, up to 1111. Thus, 4 bits



Character	Binary code	Character	Binary code	Character	Binary code
blank	010 0000	@	100 0000	`	110 0000
!	010 0001	A	100 0001	a	110 0001
"	010 0010	B	100 0010	b	110 0010
#	010 0011	C	100 0011	c	110 0011
$	010 0100	D	100 0100	d	110 0100
%	010 0101	E	100 0101	e	110 0101
&	010 0110	F	100 0110	f	110 0110
'	010 0111	G	100 0111	g	110 0111
(	010 1000	H	100 1000	h	110 1000
)	010 1001	I	100 1001	i	110 1001
*	010 1010	J	100 1010	j	110 1010
+	010 1011	K	100 1011	k	110 1011
,	010 1100	L	100 1100	l	110 1100
-	010 1101	M	100 1101	m	110 1101
.	010 1110	N	100 1110	n	110 1110
/	010 1111	O	100 1111	o	110 1111
0	011 0000	P	101 0000	p	111 0000
1	011 0001	Q	101 0001	q	111 0001
2	011 0010	R	101 0010	r	111 0010
3	011 0011	S	101 0011	s	111 0011
4	011 0100	T	101 0100	t	111 0100
5	011 0101	U	101 0101	u	111 0101
6	011 0110	V	101 0110	v	111 0110
7	011 0111	W	101 0111	w	111 0111
8	011 1000	X	101 1000	x	111 1000
9	011 1001	Y	101 1001	y	111 1001
:	011 1010	Z	101 1010	z	111 1010
;	011 1011	[	101 1011	{	111 1011
<	011 1100	\	101 1100	|	111 1100
=	011 1101	]	101 1101	}	111 1101
>	011 1110	^	101 1110	~	111 1110
?	011 1111	_	101 1111	null	111 1111

Fig. 5.2 The American Standard Code for Information Interchange (ASCII) is a standard scheme for representing alphanumeric characters using 7 bits. The upper-case and lower-case alphabet, the decimal digits, and common punctuation characters are shown here with their ASCII representations

can represent any decimal value from 0 to 15; e.g., the sequence 0101 is the **binary** (base 2) representation of the decimal number 5—namely, $0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 5$. A byte is a sequence of 8 bits; it can take on 2^8 or 256 values (0–255).

Groups of bits and bytes can represent not only decimal integers but also fractional numbers, general characters (upper-case and lower-case letters, digits, and punctuation marks), instructions to the CPU, and more complex data types such as pictures, spoken language, and the content of a medical record. Figure 5.2 shows the **American Standard Code for Information Interchange** (**ASCII**), a convention for representing 95 common characters using 7 bits. These 7 bits are commonly placed into an 8-bit unit, a byte, which is the common way of transmitting and storing these characters. The eighth bit may be used for formatting information (as in a word processor) or for additional special characters (such as currency and mathematical symbols or characters with diacritic marks), but the ASCII base standard does not cover its use. Not all characters seen on a keyboard can be encoded and stored as ASCII. The Delete and Arrow keys are often dedicated to edit functions, and the Control, Escape, and Function keys are used to modify other keys or to interact directly with programs. A standard called **Unicode** represents characters needed for foreign languages using up to 16 bits; ASCII is a small subset of Unicode.

The CPU works on data that it retrieves from memory, placing them in working registers. By manipulating the contents of its registers, the CPU performs the mathematical and logical functions that are basic to information processing: addition, subtraction, and comparison ("is greater than," "is equal to," "is less than"). In addition to registers that perform computation, the CPU also has registers that it uses to store instructions—a computer program is a set of such instructions—and to control processing. In essence, a computer is an instruction follower: it fetches an instruction from memory and then executes the instruction, which usually is an operation that requires the retrieval, manipulation, and storage of data into memory or registers. The processor often performs a simple loop, fetching and executing each instruction of a program in sequence. Some instructions can direct the processor to begin fetching instructions from a different place in memory or point in the program. Such a transfer of flow control provides flexibility in program execution. Parallel flows may also be invoked.

5.2.2 Memory

The computer's working memory stores the programs and data currently being used by the CPU. Working memory has two parts: **read-only memory** (**ROM**) and **random-access memory** (**RAM**).

ROM, or fixed memory, is permanent and unchanging. It can be read, but it cannot be altered or erased. It is used to store a few crucial programs that do not change and that must be available at all times. One such predefined program is the **bootstrap** sequence, a set of initial instructions that is executed each time the computer is started. ROM also is used to store programs that must run quickly—e.g., the base graphics that run the Macintosh computer interface.

More familiar to computer programmers is RAM, often just called **memory**. RAM can be both read and written into. It is used to store the programs, control values, and data that are in current use. It also holds the intermediate results of computations and the images to be displayed on the screen. RAM is much larger than ROM. For example, we might speak of a 2 gigabyte memory chip. A **kilobyte** is 2^{10}, or 10^3, or 1,024 bytes; a **megabyte** is 2^{20}, or 10^6, or 1,048,576 bytes; and a **gigabyte** is 2^{30}, or 10^9, or 1,073,741,824 bytes. Increasing powers of 2^{10}, or 10^3, are **terabytes**, **petabytes**, and **exabytes** (10^{18}).

A sequence of bits that can be accessed by the CPU as a unit is called a **word**. The **word size** is a function of the computer's design. Early computers had word sizes of 8 or 16 bits; newer, faster computers had 32-bit and now 64-bit word sizes that allow processing of larger chunks of information at a time. The bytes of memory are numbered in sequence. The CPU accesses each word in memory by specifying the sequence number, or **address**, of its starting byte.

5.2.3 Long-Term Storage

The computer's memory is relatively expensive, being specialized for fast read–write access; therefore, it is limited in size. It is also **volatile**: its contents are changed when the next program runs, and memory contents are not retained when power is turned off. For many medical applications we need to store more information than can be held in memory, and we want to save all that information for a long time. To save valuable programs, data, or results we place them into **long-term storage**.

Programs and data that must persist over longer periods than in volatile memory are stored on long-term storage devices, such as hard disks, flash memory or solid state disks, optical disks, or magnetic tape, each of which provide persistent storage for less cost per byte than memory and are widely available. The needed information is loaded from such storage into working memory whenever it is used. Conceptually, long-term storage can be divided into two types: (1) **active storage** is used to store data that may need to be retrieved with little delay, e.g., the medical record of a patient who currently is being treated within the hospital; and (2) **archival storage** is used to store data for documentary or legal purposes, e.g., the medical record of a patient who is deceased.

Computer storage also provides a basis for the sharing of information. Whereas memory is dedicated to an executing program, data written on storage in **file systems** or in **databases** is available to other users or processes that can access the computer's storage devices. Files and databases complement direct communication among computer users and have the advantage that the writers and readers need not be present at the same time in order to share information.

5.2.4 Graphics Processing Unit

Graphics processing units are specialized for handling computation over many bytes simultaneously. They can be thought of as enabling of simultaneous processing of many data elements with tight coupling to their memory, or frame buffers, but with limited instruction sets. This allows rapid manipulation of many memory locations simultaneously, essential to their power for computer graphics, but leaves less flexibility in general. For example, GPUs are much more efficient than CPUs for video display, the primary purpose for which they were invented, or for highly parallel computation on a single machine, such as comparing genomic sequences or image processing. However, GPUs generally require different, less mature, programming languages and software architectures than CPUs making them harder to use to date. Newer languages, such as Open CL, enable some computer codes to run over either CPUs or GPUs. When GPUs are used for general computation, rather than for video or graphics display, they are referred to as general purpose GPUs or **GPGPUs**.

5.2.5 Input Devices

Data and user-command entry remain the most costly and awkward aspects of medical data processing. Certain data can be acquired automatically; e.g., many laboratory instruments provide electronic signals that can be transmitted to computers directly, and many diagnostic radiology instruments produce output in digital form. Furthermore, redundant data entry can be minimized if data are shared among computers over networks or across direct interfaces. The most common instrument for data entry is the **keyboard** on which the user types. A **cursor** indicates the current position on the screen. Most programs allow the cursor to be moved with a **pointing device**, such as a **mouse**, **track pad**, **or touch screen**, so that making insertions and corrections is convenient. Systems developers continue to experiment with a variety of alternative input devices that minimize or eliminate the need to type.

There are also three-dimensional pointing devices, where an indicator, or just the user's own body, using optical capture, is positioned in front of the screen, and a three-dimensional display may provide feedback to the user. Some three-dimensional pointing devices used in medical virtual-reality environments also provide computer-controlled force or **tactile feedback**, so that a user can experience the resistance, for example, of a simulated needle being inserted for venipuncture or a simulated scalpel making a surgical incision.

In automatic speech recognition, digitized voice signals captured through a microphone are matched to the patterns of a vocabulary of known words, and use grammar rules to allow recognition of sentence structures. The speech input is then stored as ASCII-coded text. This technology is improving in flexibility and reliability, but error rates remain sufficiently high that manual review of the resulting text is needed. This is easier for some users than typing.

5.2.6 Output Devices

The presentation of results, or of the **output**, is the complementary step in the processing of medical data. Many systems compute information that is transmitted to health care providers and is displayed immediately on local personal computers or printed so that action can be taken. Most immediate output appears at its destination on a display screen, such as the flat-panel **liquid crystal display** (**LCD**) or **light-emitting diode** (**LED**) based displays of a personal computer (PC). The near-realistic quality of computer displays enables unlimited interaction with text, images, video, and interactive graphical elements. Graphical output is essential for summarizing and presenting the information derived from voluminous data.

A graphics screen is divided into a grid of picture elements called **pixels**. One or more bits in memory represent the output for each pixel. In a black-and-white monitor, the value of each pixel on the screen is associated with the level of intensity, or gray scale. For example, 2 bits can distinguish 2^2 or 4 display values per pixel: black, white, and two intermediate shades of gray. For color displays, the number of bits per pixel, or **bit depth**, determines the **contrast** and **color resolution** of an image. Three sets of multiple bits are necessary to specify the color of pixels on LCDs, giving the intensity for red, green, and blue components of each pixel color, respectively. For instance, three sets of 8 bits per pixel provide 2^{24} or 16,777,216 colors. The number of pixels per square inch determines the **spatial resolution** of the image (Fig. 5.3). Both parameters determine the size requirements for storing images. LCD color projectors are readily available so that the output of a workstation can also be projected onto a screen for group presentations. Multiple standard hardware interfaces, such as VGA and HDMI also enable computers to easily display to high definition and even stereoscopic televisions (3DTVs). Much diagnostic information is produced in image formats that can be shown on graphics terminals. Examples are ultrasound observations, magnetic resonance images (MRIs), and computed tomography (CT) scans.

For portability and traditional filing, output is printed on paper. Printing information is slower than displaying it on a screen, so printing is best done in advance of need. In a clinic, relevant portions of various patient records may be printed on high-volume printers the night before scheduled visits. **Laser printers** use an electromechanically controlled laser beam to generate an image on a xerographic surface, which is then used to produce paper copies, just as is done in a copier. Their spatial resolution is often better than that of displays, allowing 600 dots (pixels) per inch (commercial typesetting equipment may have a resolution of several thousand dots per inch). Color **ink-jet printers** are inexpensive, but the ink cartridges raise the cost under high use. Liquid ink is sprayed on paper by a head that moves back and forth for each line of pixels. Ink-jet printers have lower resolution than laser printers and are relatively slow, especially at high resolution. Ink-jet printers that produce images of photographic quality are also readily available. Here the base colors are merged while being sprayed so that true color mixes are placed on the paper.

Other output mechanisms are available to computer systems and may be used in medical applications, particularly sound, for alerts, feedback and instruction, such as for Automatic External Defibrillators (AEDs). A computer can produce sound via digital-to-analog conversion (see Sect. 5.3). There are also 3D printers which create objects.

5.2.7 Local Data Communications

Information can be shared most effectively by allowing access for all authorized participants whenever and wherever they need it. Transmitting data electronically among applications and computer systems facilitates such sharing by minimizing delays and by supporting interactive collaborations. Videoconferencing is also supported on PCs. Transmitting paper results in a much more passive type of information sharing. Data communication and integration are critical functions of health care information systems. Modern computing and communications are deeply intertwined.

Computer systems used in health care are specialized to fulfill the diverse needs of health professionals in various areas, such as physicians' offices, laboratories, pharmacies, intensive care units, and business offices. Even if their hardware is identical, their content will differ, and some of that content must be shared with other applications in the health care organization. Over time, the hardware in the various areas will also diverge—e.g., imaging departments will require more capable displays and larger storage, other areas will use more processor power. Demand for growth and funding to accommodate change

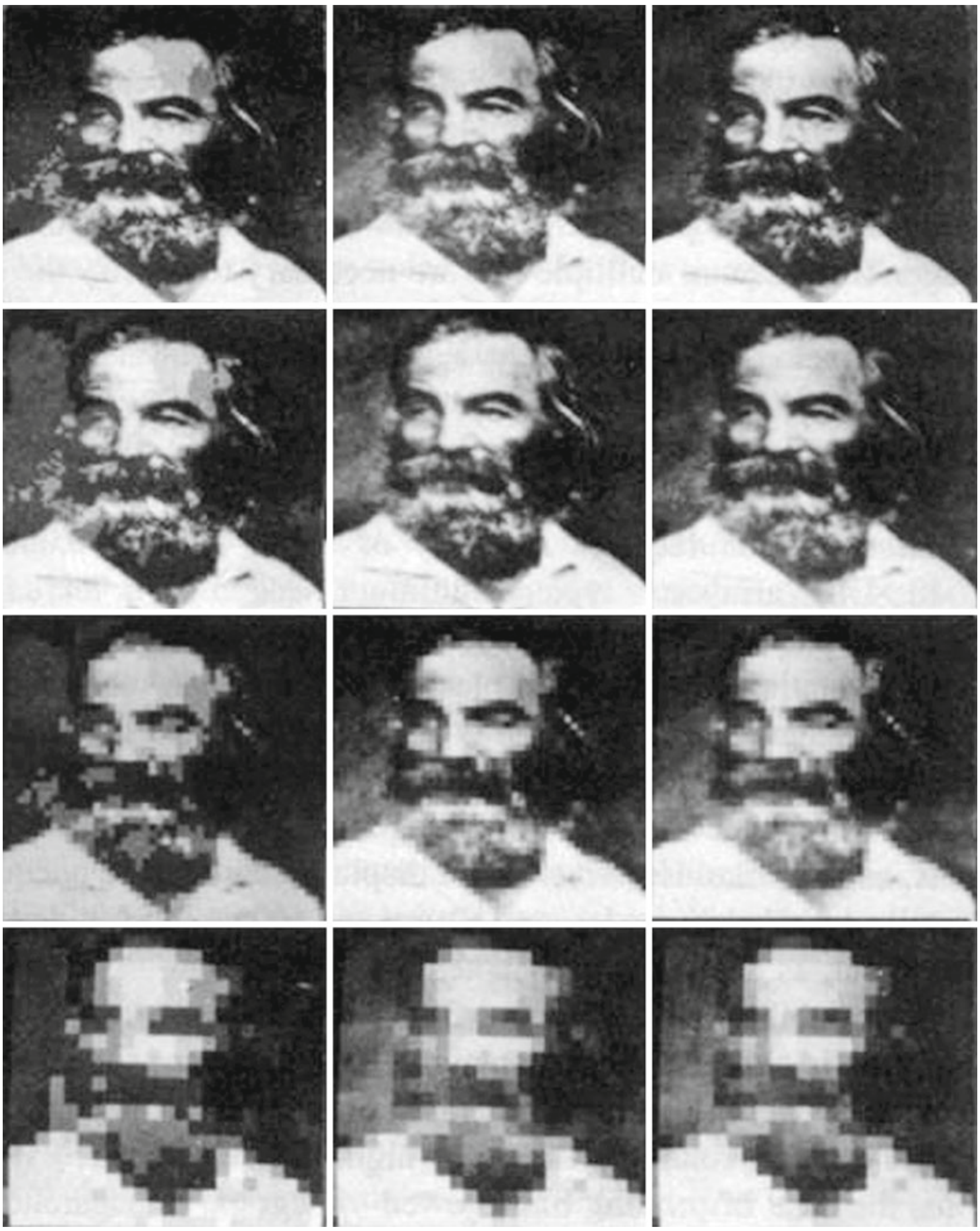


Fig. 5.3 Demonstration of how varying the number of pixels and the number of bits per pixel affects the spatial and contract resolution of a digital image. The image in the upper right corner was displayed using a 256 x 256 array of pixels, 8 bits per pixel; the subject (Walt Whitman) is easily discernible (Source: Reproduced with permission from Price R.R., & James A.E. (1982). Basic principles and instrumentation of digital radiography. In: Price R.R., et al. (Eds.). *Digital radiography: A focus on clinical utility*. Orlando: WB Saunders)

occurs at different times. Communication among diverse systems bridges the differences in computing environments.

Communication can occur via telephone lines, dedicated or shared wires, fiber-optic cables, infrared, or radio waves (wireless). In each case different communication interfaces must be enabled with the computer, different conventions or communication protocols must be obeyed, and a different balance of performance and reliability can be expected. For example, wired connections are typically higher volume communication channels and more reliable. However, communication paths can reach capacity; so for example, a wired computer in a busy hotel where the network is overloaded may communicate less reliably than a wireless smart phone with a strong wireless signal. Thus, specific communication needs and network capabilities and loads must always be considered when designing applications and implementing them.

The overall **bit rate** of a digital communication link is a combination of the rate at which symbols can be transmitted and the efficiency with which digital information (in the form of bits) is encoded in the symbols. There are many available data transmission options for connecting local networks (such as in the home). **Integrated services digital network** (**ISDN**) and, later, **digital subscriber line** (**DSL**) technologies allow network communications using conventional telephone wiring (twisted pairs). These allow sharing of data and voice transmission ranging from 1 to 10 **megabits** per second (Mbps), depending on the distance from the distribution center.

In areas remote to wired lines, these digital services may be unavailable, but the communications industry is broadening access to digital services over wireless channels. In fact, in many countries, usable wireless bandwidth exceeds wired bandwidth in many areas. Transmission for rapid distribution of information can occur via cable modems using coaxial cable (up to 30 Mbps) or direct satellite broadcast. These alternatives have a very high capacity, but all subscribers then share that capacity. Also for DSL, digital cable services and wireless services, transmission speeds are often asymmetrical, with relatively low-speed service used to communicate back to the data source. This design choice is rationalized by the assumption that most users receive more data than they send, for example, in downloading and displaying graphics, images, and video, while typing relatively compact commands to make this happen. This assumption breaks down if users generate large data objects on their personal machine that then have to be sent to other users. In this case it may be more cost-effective in terms of user time to purchase a symmetric communication service. Home networking has been further expanded with the use of WiFi (IEEE 802.11 standard for wireless communications). Computers, cell phones, and many personal health and fitness devices with embedded computers now support WiFi to establish access to other computers and the Internet, while Bluetooth is another wireless protocol generally used for communicating among computer components (such as wireless headsets to cell phones and wireless keyboards to computers).

Frame Relay is a network protocol designed for sending digital information over shared, **wide-area networks** (**WANs**). It transmits variable-length messages or packets of information efficiently and inexpensively over dedicated lines that may handle aggregate speeds up to 45 Mbps. **Asynchronous Transfer Mode** (**ATM**) is a protocol designed for sending streams of small, fixed-length cells of information over very high-speed dedicated connections—most often digital optical circuits. The underlying optical transmission circuit sends cells synchronously and supports multiple ATM circuits. The cells associated with a given ATM circuit are queued and processed asynchronously with respect to each other in gaining access to the multiplexed (optical) transport medium. Because ATM is designed to be implemented by hardware switches, information bit rates over 10 gigabits per second (Gbps) are typical.

For communication needs within an office, a building, or a campus, installation of a **local-area network** (**LAN**) allows local data communication without involving the telephone company or **network access provider**. Such a network is dedicated to linking multiple computer nodes together at high speeds to facilitate the sharing of resources—data, software, and equipment—among multiple users. Users working at individual workstations can retrieve data and programs from network **file servers**: computers dedicated to storing local files, both shared and private. The users can process information locally and then save the results over the network to the file server or send output to a shared printer.

There are a variety of **protocols** and technologies for implementing LANs, although the differences should not be apparent to the user. Typically data are transmitted as messages or **packets** of data; each packet contains the data to be sent, the network addresses of the sending and receiving nodes, and other control information. LANs are limited to operating within a geographical area of at most a few miles and often are restricted to a specific building or a single department. Separate remote LANs may be connected by **bridges**, routers, or switches (see below), providing convenient communication between machines on different networks. The information technology department of a health care organization typically takes responsibility for implementing and linking multiple LANs to form an enterprise network. Important services provided by such network administrators include integrated access to WANs, specifically to the Internet (see later discussion on Internet communication), service reliability, and security.

Early LANs used coaxial cables as the communication medium because they could deliver reliable, high-speed communications. With improved communication signal–processing technologies, however, **twisted-pair wires** (Cat-5 and better quality) have become the standard. Twisted-pair wiring is inexpensive and has a high **bandwidth** (capacity for information transmission) of at least 100 Mbps. An alternate medium, **fiber-optic cable**, offers the highest bandwidth (over 1 billion bps or 1 Gbps) and a high degree of reliability because it uses light waves to transmit information signals and is not susceptible to electrical interference. Fiber-optic cable is used in LANs to increase transmission speeds and distances by at least one order of magnitude over twisted-pair wire. Splicing and connecting into optical cable is more difficult than into twisted-pair wire, however, so in-house delivery of networking services to the desktop is still easier using twisted-pair wires. Fiber-optic cable, twisted-pair wires, and WiFi are often used in a complementary fashion—fiber-optic cable for the high-speed, shared backbone of an enterprise network or LAN and twisted-pair wires extending out from side-branch hubs to bring service to small areas and twisted-pair wires or WiFi to the individual workstation.

Rapid data transmission is supported by LANs. Many LANs still operate at 10 Mbps, but 100-Mbps networks are now cost-effective. Even at 10 Mbps, the entire contents of this book could be transmitted in a few seconds. Multiple users and high-volume data transmissions such as video may congest a LAN and its servers, however, so the effective transmission speed seen by each user may be much lower. When demand is high, LANs can be duplicated in parallel. Gateways, routers, and switches shuttle packets among these networks to allow sharing of data between computers as though the machines were on the same LAN. A **router** or a **switch** is a special device that is connected to more than one network and is equipped to forward packets that originate on one network segment to machines on another network. **Gateways** perform routing and can also translate packet formats if the two connected networks run different communication protocols.

Messages also can be transmitted through the air by radio, microwave, infrared, satellite signal, or line-of-sight laser-beam transmission. Application of these have many special trade-offs and considerations. **Broadband** has a specific technical meaning related to parallelization of signal transmission, but has been used more recently as a term to refer to any relatively high bandwidth network connection, such as cable service or third generation (3G) or fourth generation (4G) wireless cellular telephone services, which are now widespread means of broadband Internet access and communication.

Wireless users of a hospital or clinic can use these radio signals from portable devices to communicate with the Internet and, when authorized, to servers that contain clinical data and thus can gain entry to the LANs and associated services. Hospitals have many instruments that generate electronic interference, and often have reinforced concrete walls, so that radio transmission may not be reliable internally over long distances. In fact, in many medical settings the use of cellular or similar radio technologies was also prohibited for a time for perhaps justified fear of electromagnetic interference with telemetry or other delicate instruments. You experience electromagnetic interference when proximity of a cell phone causes public address speakers to make chattering sounds. These features and short battery life had made portable wireless devices weakly fit computers in medical applications until recently. Now, with: (1) better battery life due to improvements in battery technology; (2) smarter wireless radio use by the portable devices (e.g. cellular phones, tablet computers), which decrease the radio transmission power when they have stronger wireless signals (also reducing the risk for electromagnetic interference); (3) more reliable hospital wireless networks; and (4) better applications; portable wireless devices (e.g. tablet computers) are exploding in use in hospital environments.

5.3 Data Acquisition and Signal Processing Considerations

A prominent theme of this book is that capturing and entering data into a computer manually is difficult, time-consuming, error-prone, and expensive. **Real-time acquisition** of data from the actual source by direct electrical connections to instruments can overcome these problems. Direct acquisition of data avoids the need for people to measure, encode, and enter the data manually. Sensors attached to a patient convert biological signals—such as blood pressure, pulse rate, mechanical movement, and electrocardiogram (ECG)—into electrical signals, which are transmitted to the computer. The signals are sampled periodically and are converted to digital representation for storage and processing. Automated data-acquisition and signal-processing techniques are particularly important in patient-monitoring settings (see Chap. 19). Similar techniques also apply to the acquisition and processing of human voice input.

Most naturally occurring signals are **analog signals**—signals that vary continuously. The first bedside monitors, for example, were wholly analog devices. Typically, they acquired an analog signal (such as that measured by the ECG) and displayed its level on a dial or other continuous display (see, for example, the continuous signal recorded on the ECG strip shown in Fig. 19.4).

The computers with which we work are **digital computers**. A digital computer stores and processes values in discrete values collected at discrete points and at discrete times. Before computer processing is possible, analog signals must be converted to digital units. The conversion process is called **analog-to-digital conversion** (**ADC**). You can think of ADC as sampling and rounding—the continuous value is observed (sampled) at some instant and is rounded to the nearest discrete unit (Fig. 5.4). You need one bit to distinguish between two levels (e.g., on or off); if you wish to discriminate among four levels, you need two bits (because $2^2=4$), and so on.

Two parameters determine how closely the digital data represent the original analog signal:

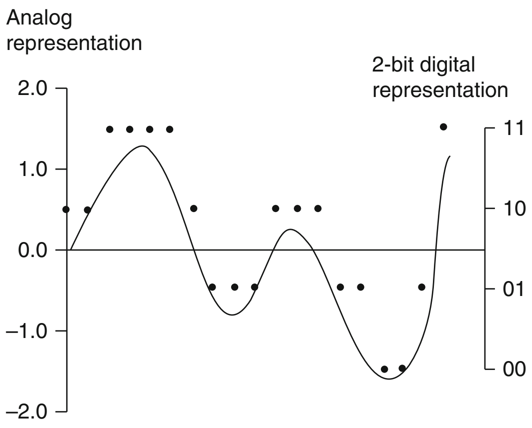


Fig. 5.4 Analog-to-digital conversion (ADC). ADC is a technique for transforming continuous-valued signals to discrete values. In this example, each sampled value is converted to one of four discrete levels (represented by 2 bits)

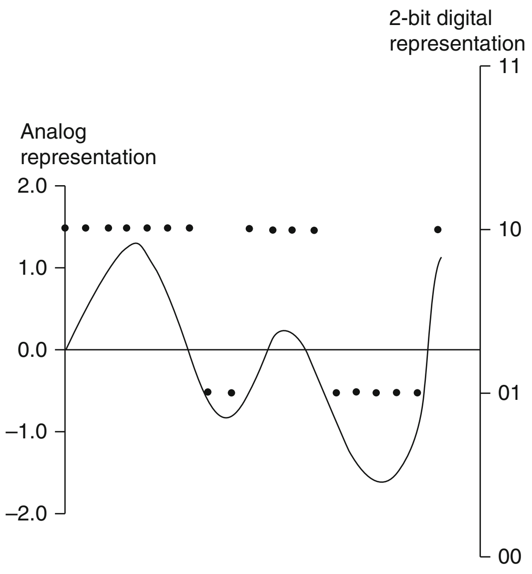


Fig. 5.5 Effect on precision of ranging. The amplitude of signals from sensors must be ranged to account, for example, for individual patient variation. As illustrated here, the details of the signal may be lost if the signal is insufficiently amplified. On the other hand, over amplification will produce clipped peaks and troughs

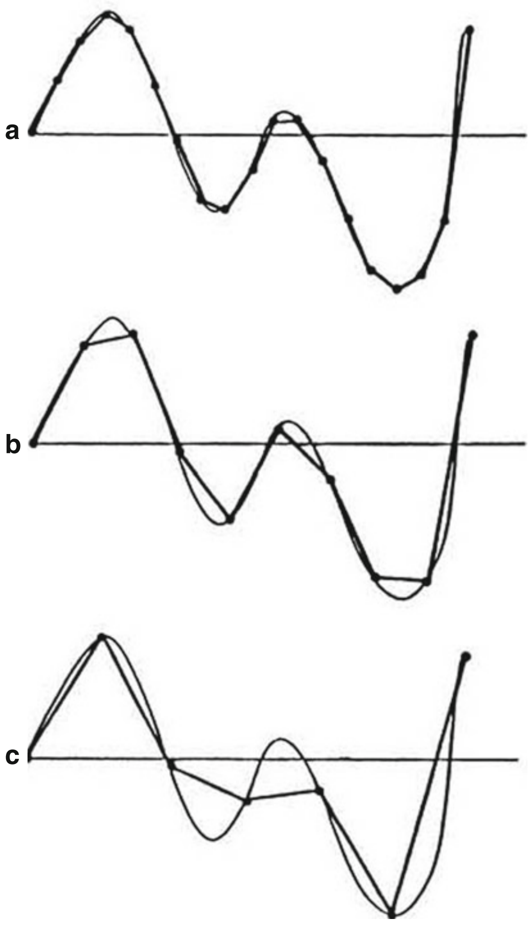


Fig. 5.6 The greater the sampling rate is, the more closely the sampled observations will correspond to the underlying analog signal. The sampling rate in (**a**) is highest; that in (**b**) is lower; and that in (**c**) is the lowest. When the sampling rate is very low (as in **c**), the results of the analog-to-digital conversion (ADC) can be misleading. Note the degradation of the quality of the signal from (**a**) to (**c**)

the precision with which the signal is recorded and the frequency with which the signal is sampled. The **precision** is the degree to which a digital estimate of a signal matches the actual analog value. The number of bits used to encode the digital estimate and their correctness determines precision; the more bits, the greater the number of levels that can be distinguished. Precision also is limited by the accuracy of the equipment that converts and transmits the signal. Ranging and calibration of the instruments, either manually or automatically, is necessary for signals to be represented with as much accuracy as possible. Improper ranging will result in loss of information. For example, a change in a signal that varies between 0.1 and 0.2 V will be undetectable if the instrument has been set to record changes between −2.0 and 2.0 in 0.5 V increments. Figure 5.5 shows another example of improper ranging.

The **sampling rate** is the second parameter that affects the correspondence between an analog signal and its digital representation. A sampling rate that is too low relative to the rate with which a signal changes value will produce a poor representation (Fig. 5.6). On the other hand, oversampling increases the expense of processing and storing the data. As a general rule, you need to sample at least twice as frequently as the highest-frequency component that you need to observe in a signal. For instance, looking at an ECG, we find that the basic contraction repetition frequency is at most a few per second, but that the **QRS wave** within each beat (see Sect. 17.5) contains useful frequency components on the order of 150 cycles per second, i.e., the QRS signal rises and falls within a much shorter interval than the basic heart beat. Thus, the ECG data-sampling rate should be at least 300 measurements per second. The rate calculated by doubling the highest frequency is called the **Nyquist frequency**. The ideas of sampling and signal estimation apply just as well to spatially varying signals (like images) with the temporal dimension replaced by one or more spatial dimensions.

Another aspect of signal quality is the amount of **noise** in the signal—the component of the acquired data that is not due to the specific phenomenon being measured. Primary sources of noise include random fluctuations in a signal detector or electrical or magnetic signals picked up from nearby devices and power lines. Once the signal has been obtained from a sensor, it must be transmitted to the computer. Often, the signal is sent through lines that pass near other equipment. En route, the analog signals are susceptible to electromagnetic interference. Inaccuracies in the sensors, poor contact between sensor and source (e.g., the patient), and disturbances from signals produced by processes other than the one being studied (e.g., respiration interferes with the ECG) are other common sources of noise.

Three techniques, often used in combination, minimize the amount of noise in a signal before its arrival in the computer:

1. **Shielding**, isolation, and grounding of cables and instruments carrying analog signals all reduce electrical interference. Often, two twisted wires are used to transmit the signal—one to carry the actual signal and the other to transmit the ground voltage at the sensor. At the destination, a differential amplifier measures the difference. Most types of interference affect both wires equally; thus, the difference should reflect the true signal. The use of glass fiber-optic cables, instead of copper wires, for signal transmission eliminates interference from electrical machinery, because optical signals are not affected by relatively slow electrical or magnetic fields.
2. For robust transmission over long distances, analog signals can be converted into a frequency-modulated representation. An FM signal represents changes of the signal as changes of frequency rather than of amplitude. **Frequency modulation** (**FM**) reduces noise greatly, because interference directly disturbs only the amplitude of the signal. As long as the interference does not create amplitude changes near the high carrier frequency, no loss of data will occur during transmission.

Conversion of analog signals to digital form provides the most robust transmission. The closer to the source the conversion occurs, the more reliable the data become. Digital transmission of signals is inherently less noise-sensitive than is analog transmission: interference rarely is great enough to change a 1 value to a 0 value or vice versa. Furthermore, digital signals can be coded, permitting detection and correction of transmission errors. Placing a microprocessor near the signal source is now the most common way to achieve such a conversion. The development of **digital signal processing** (**DSP**) chips—also used for computer voice mail and other applications—facilitates such applications.

3. **Filtering algorithms** can be used to reduce the effect of noise. Usually, these algorithms are applied to the data once they have been stored in memory. A characteristic of noise is its relatively random pattern. Repetitive signals, such as an ECG, can be integrated over several cycles, thus reducing the effects of random noise. When the noise pattern differs from the signal pattern, Fourier analysis can be used to filter the signal; a signal is decomposed into its individual components, each with a distinct period and amplitude. (The article by Wiederhold and Clayton (1985) in the Suggested Readings explains Fourier analysis in greater detail.) Unwanted components of the signal are assumed to be noise and are eliminated. Some noise (such as the 60-cycle interference caused by a building's electrical circuitry) has a regular pattern. In this case, the portion of the signal that is known to be caused by interference can be filtered out.

Once the data have been acquired and cleaned up, they typically are processed to reduce their volume and to abstract information for use by interpretation programs. Often, the data are analyzed to extract important parameters, or features, of the signal—e.g., the duration or intensity of the ST segment of an ECG. The computer also can analyze the shape of the waveform by comparing one part of a repetitive signal to another, e.g., to detect ECG beat irregularities, or by comparing a waveform to models of known patterns, or templates. In **speech recognition**, the voice signals can be compared with stored profiles of spoken words. Further analysis is necessary to determine the meaning or importance of the signals—e.g., to allow automated ECG-based cardiac diagnosis or to respond properly to the words recognized in a spoken input.

5.4 Internet Communication

External routers link the users on a LAN to a **regional network** and then to the **Internet**. The Internet is a WAN that is composed of many regional and local networks interconnected by long-range **backbone links**, including international links. The Internet was begun by the National Science Foundation in the 1980s, a period in which various networking approaches were overtaken by a common protocol designed and inspired by military considerations to enable scalability and robustness to failure of individual links in the network.

All Internet participants agree on many standards. The most fundamental is the protocol suite referred to as the **Transmission Control Protocol/Internet Protocol** (**TCP/IP**). Data transmission is always by structured packets, and all machines are identified by a standard for IP addresses. An **Internet address** consist of a sequence of four 8-bit numbers, each ranging from 0 to 255— most often written as a dotted sequence of numbers: a.b.c.d. Although IP addresses are not assigned geographically (the way ZIP codes are), they are organized hierarchically, with a first component identifying a network, a second identifying a subnet, and a third identifying a specific computer. Computers that are permanently linked into the Internet may have a fixed IP address assigned, whereas users whose machines reach the Internet by making a wireless connection only when needed, may be assigned a temporary address that persists just during a connected session. The Internet is in the process of changing to a protocol (IPv6) that supports 64-bit IP addresses, because the worldwide expansion of the Internet, the block address assignment process, and proliferation of networked individual computer devices are exhausting the old 32-bit address space. While the changeover is complex and has been moving slowly for more than a decade, much work has gone into making this transition transparent to the user.

Because 32-bit (and 64-bit) numbers are difficult to remember, computers on the Internet also have names assigned. Multiple names may be used for a given computer that performs distinct services. The names can be translated to IP addresses—e.g., when they are used to designate a remote machine—by means of a hierarchical name management system called the **Domain Name System** (DNS). Designated computers, called **name-servers**, convert a name into an IP address before the message is placed on the network; routing takes place based on only the numeric IP address. Names are also most often expressed as dotted sequences of name segments, but there is no correspondence between the four numbers of an IP address and the parts of a name. The Internet is growing rapidly; therefore, periodic reorganizations of parts of the network are common. Numeric IP addresses may have to change, but the logical name for a resource can stay the same and the (updated) DNS can take care of keeping the translation up to date. This overall process is governed today by **the Internet Corporation for Assigned Names and Numbers** (**ICANN**). Three conventions are in use for composing Internet names from segments:

1. Functional convention: Under the most common convention for the United States, names are composed of hierarchical segments increasing in specificity from right to left, beginning with one of the top-level domain-class identifiers—e.g., computer.institution.class (ci.uchicago.edu) or institution.class (whitehouse.gov).

Initially the defined top-level domain classes were .com, .edu, .gov, .int, .mil, .org, and .net (for commercial, educational, government, international organizations, military, non-profit, and ISP organizations, respectively). As Internet use has grown, many more classes have been added. Other conventions have evolved as well: www was often used as a prefix to name the **World Wide Web** (**WWW**) services on a computer (e.g., www.nlm.nih.gov).

2. Geographic convention: Names are composed of hierarchical segments increasing in specificity from right to left and beginning with a two-character top-level country domain identifier—e.g., institution.town.state.country (cnri.reston.va.us or city.paloalto.ca.us). Many countries outside of the United States use a combination of these conventions, such as csd.abdn.ac.uk, for the Computer Science Department at the University of Aberdeen (an academic institution in the United Kingdom). Note that domain names are **case-insensitive**, although additional fields in a URL, such as file names used to locate content resources, may be **case-sensitive**.
3. Attribute list address (X.400) convention: Names are composed of a sequence of attribute-value pairs that specifies the components needed to resolve the address— e.g.,/C=GB/ADMD = BT/PRMD = AC/O = Abdn/OU=csd/, which is equivalent to the address csd.abdn.ac.uk. This convention derives from the X.400 address standard that is used mainly in Europe. It has the advantage that the address elements (e.g., /C for Country name, /ADMD for Administrative Management Domain name, and /PRMD for Private Management Domain name) are explicitly labeled and may come in any order. Country designations differ as well. However, this type of address is generally more difficult for humans to understand and has not been adopted broadly in the Internet community.

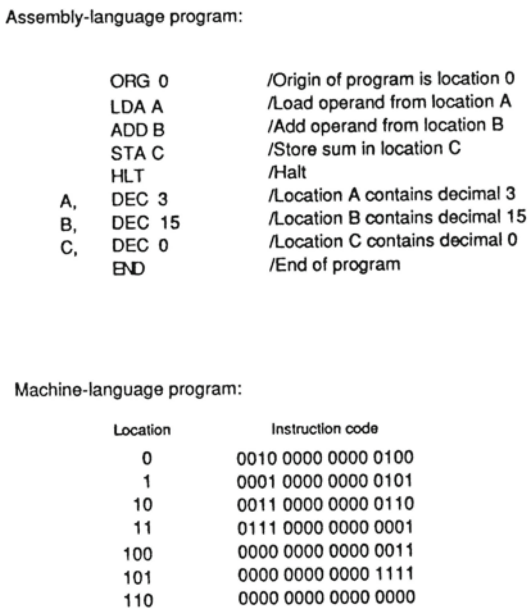
An institution that has many computers may provide a service whereby all communications (e.g., incoming e-mails) go to a single address (e.g., uchicago.edu or apple.com), and then local tables are used to direct each message to the right computer or individual. Such a scheme insulates outside users from internal naming conventions and changes, and can allow dynamic machine selection for the service in order to distribute loading. Such a central site can also provide a firewall—a means to attempt to keep viruses and unsolicited and unwanted connections or messages (spam) out. The nature of attacks on networks and their users is such that constant vigilance is needed at these service sites to prevent system and individual intrusions.

The routing of packets of information between computers on the Internet is the basis for a rich array of information services. Each such service—be it resource naming, electronic mail, file transfer, remote computer log in, World Wide Web, or another service— is defined in terms of sets of protocols that govern how computers speak to each other. These worldwide inter-computer linkage conventions allow global sharing of information resources, as well as personal and group communications. The Web's popularity and growing services continues to change how we deal with people, form communities, make purchases, entertain ourselves, and perform research. The scope of all these activities is more than we can cover in this book, so we restrict ourselves to topics important to health care. Even with this limitation, we can only scratch the surface of many topics.

Regional and national networking services are now provided by myriad commercial communications companies, and users get access to the regional networks through their institutions or privately by paying an **Internet service provider** (**ISP**), who in turn gets WAN access through a **network access provider** (NAP). There are other WANs besides the Internet, some operated by demanding commercial users, and others by parts of the federal government, such as the Department of Defense, the National Aeronautics and Space Administration, and the Department of Energy. Nearly all countries have their own networks connected to the Internet so that information can be transmitted to most computers in the world. Gateways of various types connect all these networks, whose capabilities may differ. It is no longer possible to show the Internet map on a single diagram.

5.5 Software

All the functions performed by the hardware of a computer system are directed by computer programs, or software (e.g. data acquisition from input devices, transfer of data and programs to and from working memory, computation and information processing by the CPU, formatting and presentation of results via the GPU, exchange of data across networks).



Assembly-language program:

```
          ORG 0       /Origin of program is location 0
          LDA A       /Load operand from location A
          ADD B       /Add operand from location B
          STA C       /Store sum in location C
          HLT         /Halt
     A,   DEC 3       /Location A contains decimal 3
     B,   DEC 15      /Location B contains decimal 15
     C,   DEC 0       /Location C contains decimal 0
          END         /End of program
```

Machine-language program:

```
Location      Instruction code
   0          0010 0000 0000 0100
   1          0001 0000 0000 0101
  10          0011 0000 0000 0110
  11          0111 0000 0000 0001
 100          0000 0000 0000 0011
 101          0000 0000 0000 1111
 110          0000 0000 0000 0000
```

Fig. 5.7 An assembly-language program and a corresponding machine-language program to add two numbers and to store the result.

5.5.1 Programming Languages

In our discussion of the CPU in Sect. 5.2, we explained that a computer processes information by manipulating words of information in registers. Instructions that tell the processor which operations to perform also are sequences of 0's and 1's, a binary representation called **machine language** or **machine code** or just **code**. Machine-code instructions are the only instructions that a computer can process directly. These binary patterns, however, are difficult for people to understand and manipulate. People think best symbolically. Thus, a first step toward making programming easier and less error prone was the creation of an assembly language. **Assembly language** replaces the sequences of bits of machine-language programs with words and abbreviations meaningful to humans; a programmer instructs the computer to LOAD a word from memory, ADD an amount to the contents of a register, STORE it back into memory, and so on. A program called an **assembler** translates these instructions into binary machine-language representation before execution of the code. There is a one-to-one correspondence between instructions in assembly and machine languages. To increase efficiency, we can combine sets of assembly instructions into **macros** and thus reuse them. An assembly-language programmer must consider problems on a hardware-specific level, instructing the computer to transfer data between registers and memory and to perform primitive operations, such as incrementing registers, comparing characters, and handling all processor exceptions (Fig. 5.7).

On the other hand, the problems that the users of a computer wish to solve are real-world problems on a higher conceptual level. They want to be able to instruct the computer to perform tasks such as to retrieve the latest trends of their test results, to monitor the status of hypertensive patients, or to order new medications. To make communication with computers more understandable and less tedious, computer scientists developed higher-level, user-oriented **symbolic-programming languages**.

Using a higher-level language, such as one of those listed in Table 5.1, a programmer defines variables to represent higher-level entities and specifies arithmetic and symbolic operations without worrying about the details of how the hardware performs these operations. The details

Table 5.1 Distinguishing features of 17 common programming languages

Programming language	First year	Primary application domain	Type	Operation	Type checks	Procedure call method	Data management method
FORTRAN	1957	Mathematics	Procedural	Compiled	Weak	By reference	Simple files
COBOL	1962	Business	Procedural	Compiled	Yes	By name	Formatted files
Pascal	1978	Education	Procedural	Compiled	Strong	By name	Record files
Smalltalk	1976	Education	Object	Interpreted	Yes	By defined methods	Object persistence
PL/l	1965	Math, business	Procedural	Compiled	Coercion	By reference	Formatted files
Ada	1980	Math, business	Procedural	Compiled	Strong	By name	Formatted files
Standard ML	1989	Logic, math	Functional	Compiled	Yes	By value	Stream files
MUMPS (M)	1962	Data handling	Procedural	Interpreted	No	By reference	Hierarchical files
LISP	1964	Logic	Functional	Either	No	By value	Data persistence
C	1976	Data handling	Procedural	Compiled	Weak	By reference	Stream files
C++	1986	Data handling	Hybrid	Compiled	Strong	By reference	Object files
Java	1995	Data display	Object	Either	Strong	By value	Object classes
JavaScript	1995	Interactive Web	Object	Interpreted	Weak	By value or reference	Context-specific object classes
Perl	1987	Text processing	Hybrid	Interpreted	Dynamic	By reference	Stream files
Python	1990	Scripting	Hybrid	Interpreted	Dynamic	By reference	Stream files
Erlang	1986	Real-time systems	Functional; concurrent	Compiled	Dynamic	By reference	Stream files

of managing the hardware are hidden from the programmer, who can specify with a single statement an operation that may translate to thousands of machine instructions. A compiler is used to translate automatically a high-level program into machine code. Some languages are interpreted instead of compiled. An **interpreter** converts and executes each statement before moving to the next statement, whereas a compiler translates all the statements at one time, creating a binary program, which can subsequently be executed many times. MUMPS (M) is an interpreted language, LISP may either be interpreted or compiled, and FORTRAN routinely is compiled before execution. Hundreds of languages have been developed—here we touch on only a few that are important from a practical or conceptual level.

Each statement of a language is characterized by syntax and semantics. The syntactic rules describe how the statements, declarations, and other language constructs are written—they define the language's grammatical structure. Semantics is the meaning given to the various syntactic constructs. The following sets of statements (written in Pascal, FORTRAN, COBOL, and LISP) all have the same semantics:

```
  C := A + B        C := A + B              LN IS "The value          (SETQ C
  PRINTF(C)         WRITE 10,                 is NNN.FFF"             (PLUS A B))
  no layout         6 c10                ADD A TO B, GIVING C         (format file
  choice            FORMAT                  MOVE C TO LN              6 "The value
                    ("The value is" F5.2)   WRITE LN                  is ~ 5,2F" C)
```

Each set of statements instructs the computer to add the values of variables A and B, to assign the result to variable C, and to write the result onto a file. Each language has a distinct syntax for indicating which operations to perform. Regardless of the particular language in which a program is written, in the end, the computer executes similar (perhaps exactly the same) instructions to manipulate sequences of 0's and 1's within its registers.

Computer languages are tailored to handle specific types of computing problems, as shown in Table 5.1, although all these languages are sufficiently flexible to deal with nearly any type of problem. Languages that focus on a flexible, general computational infrastructure, such as C or Java have to be augmented with large collections of libraries of procedures, and learning the specific libraries takes more time than does learning the language itself. Languages also differ in usability. A language meant for education and highly reliable programs will include features to make it foolproof, by way of checking that the types of values, such as integers, decimal numbers, and strings of characters, match throughout their use—this is called **type checking**. Such features may cause programs to be slower in execution, but more reliable. Without type checking, smart programmers can instruct the computers to perform some operations more efficiently than is possible in a more constraining language.

Sequences of statements are grouped into *procedures*. Procedures enhance the clarity of larger programs and also provide a basis for reuse of the work by other programmers. Large programs are in turn mainly sequences of invocations of such procedures, some coming from libraries (such as format in LISP) and others written for the specific application. These procedures are called with *arguments*—e.g., the medical record number of a patient—so that a procedure to retrieve a value, such as the patient's age might be: age (medical record number). An important distinction among languages is how those arguments are transmitted. Just giving the value in response to a request is the safest method. Giving the name provides the most information to the procedure, and giving the reference (a pointer to where the value is stored) allows the procedure to go back to the source, which can be efficient but also allows changes that may not be wanted. Discussions about languages often emphasize these various features, but the underlying concern is nearly always the trade-off of protection versus power.

Programmers work in successively higher levels of abstraction by writing, and later invoking, standard procedures in the form of functions and subroutines. Within these they may also have routines that spawn other routines, called **threads**. Threads allow multiple execution units, or concurrency, in programming, and as systems scale they become increasingly important, particularly as multiple functions are being run over the same data simultaneously. Built-in functions and subroutines create an environment in which users can perform complex operations by specifying single commands. Tools exist to combine related functions for specific tasks—e.g., to build a forms interface that displays retrieved data in a certain presentation format.

Specialized languages can be used directly by nonprogrammers for well-understood tasks, because such languages define additional procedures for specialized tasks and hide yet more detail. For example, users can search for, and retrieve data from, large databases using the Structured Query Language (SQL) of database management systems (discussed later in this section). With the help of statistical languages, such as SAS or R, users can perform extensive statistical calculations, such as regression analysis and correlation. Other users may use a spreadsheet program, such as Excel, to record and manipulate data with formulas in the cells of a spreadsheet matrix. In each case, the physical details of the data storage structures and the access mechanisms are hidden from the user.

The end users of a computer may not even be aware that they are programming per se, if the language is so natural that it matches their needs in an intuitive manner. Moving icons on a screen and dragging and dropping them into boxes or onto other icons is a form of programming supported by many layers of interpreters and compiler-generated code. If the user saves a **script** (a keystroke-by-keystroke record) of the actions performed for later reuse, then he or she has created a program. Some systems allow such scripts to be viewed and edited for later updates and changes; e.g., there is a macro function available in the Microsoft Excel spreadsheet and on Macintosh computers via AppleScript.

Even though many powerful languages and packages handle these diverse tasks, we still face the challenge of incorporating multiple functions into a larger system. It is easy to envision a system where a Web browser provides access to statistical results of data collected from two related databases. Such interoperation is not simple; however, modern layers of software coupled with programming expertise now make such complex interactions routine in health information systems and our everyday lives.

5.5.2 Data Management

Data provide the infrastructure for recording and sharing information. Data become information when they are organized to affect decisions, actions, and learning (see Chap. 2). Accessing and moving data from the points of collection to the points of use are among the primary functions of computing in medicine. These applications must deal with large volumes of varied data and manage them, for persistence, on external storage. The mathematical facilities of computer languages are based on common principles and are, strictly speaking, equivalent. The same conceptual basis is not available for data management facilities. Some languages allow only internal structures to be made persistent; in that case, external library programs are used for handling storage.

Handling data is made easier if the language supports moving structured data from internal memory to external, persistent storage. Data can, for instance, be viewed as a stream, a model that matches well with data produced by some instruments, by TCP connections over the Internet, or by a ticker tape. Data can also be viewed as records, matching well with the rows of a table (Table 5.2); or data can be viewed as a hierarchy, matching well with the structure of a medical record, including patients, their visits, and their findings during a visit.

If the language does not directly support the best data structure to deal with an application, additional programming must be done to construct the desired structure out of the available facilities. The resulting extra layer, however, typically costs effort (and therefore money) and introduces inconsistencies among applications trying to share information.

Table 5.2 A simple patient data file containing records for four pediatric patients

Record number	Name	Sex	Date of Birth
22-546-998	Adams, Clare	F	11 Nov 1998
62-847-991	Barnes, Tanner	F	07 Dec 1997
47-882-365	Clark, Laurel	F	10 May 1998
55-202-187	Davidson, Travis	M	10 Apr 2000

Note: The key field of each record contains the medical record number that uniquely identifies the patient. The other fields of the record contain demographic information

5.5.3 Operating Systems

Users ultimately interact with the computer through an **operating system** (**OS**): a program that supervises and controls the execution of all other programs and that directs the operation of the hardware. The OS is software that is typically included with a computer system and it manages the resources, such as memory, storage, and devices for the user. Once started, the **kernel** of the OS resides in memory at all times and runs in the background. It assigns the CPU to specific tasks, supervises other programs running in the computer, controls communication among hardware components, manages the transfer of data from input devices to output devices, and handles the details of file management such as the creation, opening, reading, writing, and closing of data files. In shared systems, it allocates the resources of the system among the competing users. The OS insulates users from much of the complexity of handling these processes. Thus, users are able to concentrate on higher-level problems of information management. They do get involved in specifying which programs to run and in giving names to the directory structures and files that are to be made persistent. These names provide the links to the user's work from one session to another. Deleting files that are no longer needed and archiving those that should be kept securely are other interactions that users have with the OS.

Programmers can write **application programs** to automate routine operations that store and organize data, to perform analyses, to facilitate the integration and communication of information, to perform bookkeeping functions, to monitor patient status, to aid in education—in short, to perform all the functions provided by medical computing systems. These programs are then filed by the OS and are available to its users when needed.

PCs typically operate as **single-user systems**, whereas servers are **multiuser systems**. In a multiuser system, all users have simultaneous access to their **jobs**; users interact through the OS, which switches resources rapidly among all the jobs that are running. Because people work slowly compared with CPUs, the computer can respond to multiple users, seemingly at the same time. Thus, all users have the illusion that they have the full attention of the machine, as long as they do not make very heavy demands. Such shared resource access is important where databases must be shared, as we discuss below in Database Management Systems. When it is managing sharing, the OS spends resources for queuing, switching, and re-queuing jobs. If the total demand is too high, the overhead increases disproportionately and slows the service for everyone. High individual demands are best allocated to distributed systems (discussed in Sect. 5.6), or to dedicated machines where all resources are prioritized to a primary user.

Because computers need to perform a variety of services, several application programs reside in main memory simultaneously. **Multiprogramming** permits the effective use of multiple devices; while the CPU is executing one program, another program may be receiving input from external storage, and another may be generating results on a printer. With the use of multiple simultaneous programs executing in one system it becomes important to ensure one program does not interfere with another or with the OS. Thus, **protected memory** comes in to play. Protected memory is only available to the program that allocated it. Web browsers and the Apple iOS (operating system for smart phones and tablets) similarly protect one process from another for security (unless authorized by the user). In **multiprocessing** systems, several processors (CPUs) are used by the OS within a single computer system, thus increasing the overall processing power. Note, however, that multiprogramming does not necessarily imply having multiple processors.

Memory may still be a scarce resource, especially under multiprogramming. When many programs and their data are active simultaneously, they may not all fit in the physical memory on the machine at the same time. To solve this problem, the OS will partition users' programs and data into **pages**, which can be kept in tempo-

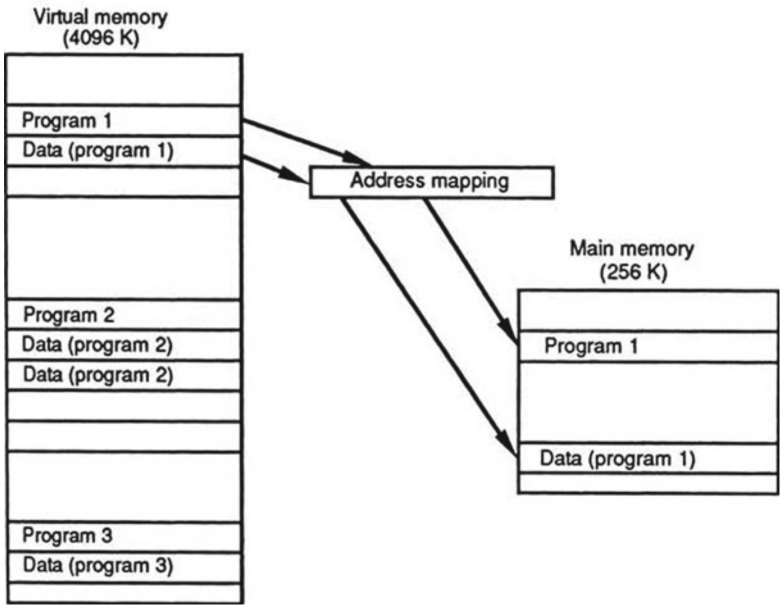


Fig. 5.8 Virtual-memory system. Virtual memory provides users with the illusion that they have many more addressable memory locations than there are in real memory—in this case, more than five times as much. Programs and data stored on peripheral disks are swapped into main memory when they are referenced; logical addresses are translated automatically to physical addresses by the hardware

rary storage on disk and are brought into main memory as needed. Such a storage allocation is called **virtual memory**. Virtual memory can be many times the size of real memory, so users can allocate many more pages than main memory can hold. Also individual programs and their data can use more memory than is available on a specific computer. Under virtual memory management, each address referenced by the CPU goes through an address mapping from the **virtual address** of the program to a physical address in main memory (Fig. 5.8). When a memory page is referenced that is not in physical storage, the CPU creates space for it by swapping out a little-used page to secondary storage and bringing in the needed page from storage. This mapping is handled automatically by the hardware on most machines but still creates significant delays, so the total use of virtual memory must be limited to a level that permits the system to run efficiently.

A large collection of **system programs** are generally associated with the **kernel** of an OS. These programs include utility programs, such as **graphical user interface** (**GUI**) routines; security management; compilers to handle programs written in higher-level languages; **debuggers** for newly created programs; communication software; diagnostic programs to help maintain the computer system; and substantial libraries of standard routines (such as for listing and viewing files, starting and stopping programs, and checking on system status). Modern software libraries include tools such as sorting programs and programs to perform complex mathematical functions and routines to present and manipulate graphical displays that access a variety of application programs, handle their point-and-click functions, allow a variety of fonts, and the like.

5.5.4 Database Management Systems

Throughout this book, we emphasize the importance to good medical decision making of timely access to relevant and complete data from diverse sources. Computers provide the primary means for organizing and accessing these data; however, the programs to manage the data are complex and are difficult to write. Database technology supports the integration and organization of data and assists users with data entry, long-term storage, and retrieval. Programming data management software is particularly difficult when multiple users share data (and thus may try to access data simultaneously), when they must search through voluminous data rapidly and at unpredictable times, and when the relationships among data elements are complex. For health care applications, it is important that the data be complete and virtually error-free. Furthermore, the need for long-term reliability makes it risky to entrust a medical database to locally written programs. The programmers tend to move from project to project, computers will be replaced, and the organizational units that maintain the data may be reorganized.

Not only the individual data values but also their meanings and their relationships to other data must be stored. For example, an isolated data element (e.g., the number 99.7) is useless unless we know that that number represents a human's body temperature in degrees Fahrenheit and is linked to other data necessary to interpret its value—the value pertains to a particular patient who is identified by a unique medical record number, the observation was taken at a certain time (02:35, 7 Feb 2000) in a certain way (orally), and so on. To avoid loss of descriptive information, we must keep together clusters of related data throughout processing. These relationships can be complex; e.g., an observation must be linked not only to the patient but also to the person recording the observation, to the instrument that he used to acquire the values, and to the physical state of the patient.

The meaning of data elements and the relationships among those elements are captured in the structure of the database. **Databases** are collections of data, typically organized into fields, records, and files (see Table 5.2), as well as descriptive metadata. The **field** is the most primitive building block; each field represents one data element. For example, the database of a hospital's registration system typically has fields such as the patient's identification number, name, date of birth, gender, admission date, and admitting diagnosis. Fields are usually grouped together to form **records**. A record is uniquely identified by one or more **key fields**—e.g., patient identification number and observation time. Records that contain similar information are grouped in **files**. In addition to files about patients and their diagnoses, treatments, and drug therapies, the database of a health care information system will have separate files containing information about charges and payments, personnel and payroll, inventory, and many other topics. All these files relate to one another: they may refer to the same patients, to the same personnel, to the same services, to the same set of accounts, and so on.

Metadata describes where in the record specific data are stored, and how the right record can be located. For instance, a record may be located by searching and matching patient ID in the record. The metadata also specifies where in the record the digits representing the birth date are located and how to convert the data to the current age. When the structure of the database changes—e.g., because new fields are added to a record—the metadata must be changed as well. When data are to be shared, there will be continuing requirements for additions and reorganizations to the files and hence the metadata. The desire for **data independence**—i.e., keeping the applications of one set of users independent from changes made to applications by another group—is the key reason for using a database management system for shared data.

A **database management system** (**DBMS**) is an integrated set of programs that helps users to store and manipulate data easily and efficiently. The conceptual (logical) view of a database provided by a DBMS allows users to specify what the results should be without worrying too much about how they will be obtained; the DBMS handles the details of managing and accessing data. A crucial part of a database kept in a DBMS is the **schema**, containing the needed metadata. A schema is the machine-readable definition of the contents and organization of the records of all the data files. Programs are insulated by the DBMS from changes in the way that data are stored, because the programs access data by field name rather than by address. A DBMS also provides facilities for entering, editing, and retrieving data. Often, fields are associated with lists or ranges of valid values; thus, the DBMS can detect and request correction of some data-entry errors, thereby improving database integrity.

```
SELECT      Patient ID, Name, Age, Systolic

FROM        Patients

WHERE       Sex = 'M'  and
            Age >= 45 and
            Age <= 64 and
            Systolic > 140
```

Fig. 5.9 An example of a simple database query written in Structured Query Language (SQL). The program will retrieve the records of males whose age is between 45 and 64 years and whose systolic blood pressure is greater than 140 mmHg

Users retrieve data from a database in either of two ways. Users can query the database directly using a query language to extract information in an ad hoc fashion—e.g., to retrieve the records of all male hypertensive patients aged 45–64 years for inclusion in a retrospective study. Figure 5.9 shows the syntax for such a query using SQL. Query formulation can be difficult, however; users must understand the contents and underlying structure of the database to construct a query correctly. Often, database programmers formulate the requests for health professionals.

To support occasional use, **front-end applications** to database systems can help a user retrieve information using a menu based on the schema. More often, transactional applications, such as a drug order–entry system, will use a database system without the pharmacist or ordering physician being aware of the other's presence. The medication-order records placed in the database by the physician create communication transactions with the pharmacy; then, the pharmacy application creates the daily drug lists for the patient care units.

Some database queries are routine requests—e.g., the resource utilization reports used by health care administrators and the end-of-month financial reports generated for business offices. Thus, DBMSs often also provide an alternative, simpler means for formulating such queries, called **report generation**. Users specify their data requests on the input screen of the report-generator program. The report generator then produces the actual query program using information stored in the schema, often at predetermined intervals. The reports are formatted such that they can be distributed without modification. The report-generation programs can extract header information from the schema. Routine report generation should, however, be periodically reviewed in terms of its costs and benefits. Reports that are not read are a waste of computer, natural, and people resources. A reliable database will be able to provide needed and up-to-date information when that information is required.

Many DBMSs support multiple **views**, or models, of the data. The data stored in a database have a single physical organization, yet different user groups can have different perspectives on the contents and structure of a database. For example, the clinical laboratory and the finance department might use the same underlying database, but only the data relevant to the individual application area are available to each group. Basic patient information will be shared; the existence of other data is hidden from groups that do not need them. Application-specific descriptions of a database are stored in such **view schemas**. Through the views, a DBMS controls access to data. Thus, a DBMS facilitates the integration of data from multiple sources and avoids the expense of creating and maintaining multiple files containing redundant information. At the same time, it accommodates the differing needs of multiple users. The use of database technology, combined with communications technology (see the following discussion on Software for Network Communications), will enable health care institutions to attain the benefits both of independent, specialized applications and of large integrated databases.

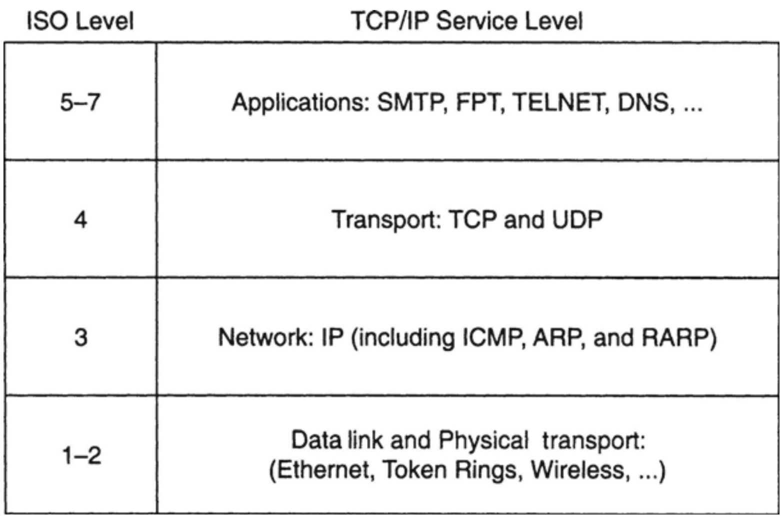
Database design and implementation has become a highly specialized field. An introduction to the topic is provided by Garcia-Molina et al. (2002). Wiederhold's book (1981) discusses the organization and use of databases in health care settings. Most medical applications use standard products from established vendors. However, these databases and application architectures are inherently oriented toward the transactions needed for the workflows of the applications, one patient at a time. Thus, these are called **on line transaction processing** (**OLTP**) systems, typically designed for use by thousands of simultaneous users doing simple repetitive queries. Data warehousing or **on line analytic processing** (**OLAP**) systems focus on use of DBMS differently, for querying across multiple patients simultaneously, typically by few users for infrequent, but complex queries, often for research. To achieve both of these architectural goals, hospital information systems will duplicate the data in two separate DBMS with different data architectures. Thus computer architectures may require the coordination of multiple individual computer systems, especially when both systems use data derived from a single source.

5.5.5 Software for Network Communications

The ability of computers to communicate with each other over local and remote networks brings tremendous power to computer users. Internet communications make it possible to share data and resources among diverse users and institutions around the world. Network users can access shared patient data (such as a hospital's medical records) or international databases (such as bibliographic databases of scientific literature or genomics databases describing what is known about the biomolecular basis of life and disease). Networks make it possible for remote users to communicate with one another and to collaborate. In this section, we introduce the important concepts that will allow you to understand network technology.

5.5.5.1 The Network Stack

Network power is realized by means of a large body of communications software. This software handles the physical connection of each computer to the network, the internal preparation of data to be sent or received over the network, and the interfaces between the network data flow and applications programs. There are now tens of millions of computers of different kinds on the Internet and hundreds of programs in each machine that service network communications. Two key ideas make it possible to manage the complexity of network software: network service stacks and network protocols. These strategies



ISO Level	TCP/IP Service Level
5–7	Applications: SMTP, FPT, TELNET, DNS, ...
4	Transport: TCP and UDP
3	Network: IP (including ICMP, ARP, and RARP)
1–2	Data link and Physical transport: (Ethernet, Token Rings, Wireless, ...)

Fig. 5.10 TCP/IP network service level stack and corresponding levels of the Open Systems Interconnection (OSI) Reference model developed by the International Standards Organization (ISO). Each level of the stack specifies a progressively higher level of abstraction. Each level serves the level above and expects particular functions or services from the level below it. *SMTP* Simple Mail Transport Protocol, *FTP* File Transfer Protocol, *DNS* Domain Name System, *TCP* Transmission Control Protocol, *UDP* User Datagram Protocol, *IP* Internet Protocol, *ICMP* Internet Control Message Protocol, *ARP* Address Resolution Protocol, *RARP* Reverse Address Resolution Protocol

allow communication to take place between any two machines on the Internet, ensure that application programs are insulated from changes in the network infrastructure, and make it possible for users to take advantage easily of the rapidly growing set of information resources and services. The **network stack** serves to organize communications software within a machine. Network software is made modular by dividing the responsibilities for network communications into different levels, with clear interfaces between the levels. The four-level network stack for TCP/IP is shown in Fig. 5.10, which also compares that stack to the seven-level stack defined by the International Standards Organization.

At the lowest level—the Data Link and Physical Transport level—programs manage the physical connection of the machine to the network, the physical-medium packet formats, and the means for detecting and correcting errors. The Network level implements the IP method of addressing packets, routing packets, and controlling the timing and sequencing of transmissions. The Transport level converts packet-level communications into several services for the Application level, including a reliable serial byte stream (TCP), a transaction-oriented User Datagram Protocol (UDP), and newer services such as real-time video.

The Application level is where programs run that support electronic mail, file sharing and transfer, Web posting, downloading, browsing, and many other services. Each layer communicates with only the layers directly above and below it and does so through specific interface conventions. The network stack is machine- and OS-dependent—because it has to run on particular hardware and to deal with the OS on that machine (filing, input–output, memory access, etc.). But its layered design serves the function of modularization. Applications see a standard set of data-communication services and do not each have to worry about details such as how to form proper packets of an acceptable size for the network, how to route packets to the desired machine, how to detect and correct errors, or how to manage the particular network hardware on the computer. If a computer changes its network connection from a wired to a wireless network, or if the **topology** of the network changes, the applications are unaffected. Only the lower level Data Link and Network layers need to be updated.

Internet protocols(see Sect. 5.3) are shared conventions that serve to standardize communications among machines—much as, for two people to communicate effectively, they must agree on the syntax and meaning of the words they are using, the style of the interaction (lecture versus conversation), a procedure for handling interruptions, and so on. Protocols are defined for every Internet service (such as routing, electronic mail, and Web access) and establish the conventions for representing data, for requesting an action, and for replying to a requested action. For example, protocols define the format conventions for e-mail addresses and text messages (RFC822), the attachment of multimedia content (Multipurpose Internet Mail Extensions (MIME)), the delivery of e-mail messages (Simple Mail Transport Protocol (SMTP)), the transfer of files (File Transfer Protocol (FTP)), connections to remote computers (SSH), the formatting of Web pages (Hypertext Markup Language (HTML)), the exchange of routing information, and many more. By observing these protocols, machines of different types can communicate openly and can interoperate with each other. When requesting a Web page from a server using the Hypertext Transfer Protocol (HTTP), the client does not have to know whether the server is a UNIX machine, a Windows machine, or a mainframe running VMS—they all appear the same over the network if they adhere to the HTTP protocol. The layering of the network stack is also supported by protocols. As we said, within a machine, each layer communicates with only the layer directly above or below. Between machines, each layer communicates with only its peer layer on the other machine, using a defined protocol. For example, the SMTP application on one machine communicates with only an SMTP application on a remote machine. Similarly, the Network layer communicates with only peer Network layers, for example, to exchange routing information or control information using the Internet Control Message Protocol (ICMP).

We briefly describe four of the basic services available on the Internet: electronic mail, FTP, SSH, and access to the World Wide Web.

5.5.5.2 Electronic Mail

Users send to and receive messages from other users via electronic mail, mimicking use of the postal service. The messages travel rapidly: except for queuing delays at gateways and receiving computers, their transmission is nearly instantaneous. Electronic mail was one of the first protocols invented for the Internet (around 1970, when what was to become the Internet was still called the **ARPANET**). A simple e-mail message consists of a **header** and a **body**. The header contains information formatted according to the RFC822 protocol, which controls the appearance of the date and time of the message, the address of the sender, addresses of the recipients, the subject line, and other optional header lines. The body of the message contains free text. The user addresses the e-mail directly to the intended reader by giving the reader's account name or a personal alias followed by the IP address or domain name of the machine on which the reader receives mail—e.g., JohnSmith@domain.name. If the body of the e-mail message is encoded according to the MIME standard it may also contain arbitrary multimedia information, such as drawings, pictures, sound, or video. Mail is sent to the recipient using the SMTP standard. It may either be read on the machine holding the addressee's account or it may be downloaded to the addressee's local computer for reading.

It is easy to broadcast electronic mail by sending it to a **mailing list** or a specific **list-server**, but electronic mail etiquette conventions dictate that such communications be focused and relevant. **Spamming**, which is sending e-mail solicitations or announcements to broad lists, is annoying to recipients, but is difficult to prevent. Conventional e-mail is sent in clear text over the network so that anyone observing network traffic can read its contents. Protocols for encrypted e-mail, such as Privacy-Enhanced Mail (PEM) or encrypting attachments, are also available, but are not yet widely deployed. They ensure that the contents are readable by only the intended recipients. Because secure email is generally not in use, communication of protected health information from providers to patients is potentially in violation of the HIPAA regulations in that the information is not secure in transit. It remains less clear, however, if it is appropriate for physicians to answer direct questions from patients in email, given that the patient has begun the insecure communication. Large health systems have generally deployed secure communication portals over the Web (where both participants must authenticate to the same system) to overcome this.

5.5.5.3 File Transfer Protocol (FTP)

FTP facilitates sending and retrieving large amounts of information—of a size that is uncomfortably large for electronic mail. For instance, programs and updates to programs, complete medical records, papers with many figures or images for review, and the like could be transferred via FTP. FTP access requires several steps: (1) accessing the remote computer using the IP address or domain name; (2) providing user identification to authorize access; (3) specifying the name of a file to be sent or fetched using the file-naming convention at the destination site; and (4) transferring the data. For open sharing of information by means of FTP sites, the user identification is by convention "anonymous" and the requestor's e-mail address is used as the password. **Secure FTP** (**SFTP**) uses the same robust security mechanism as SSH (below), but provides poor performance. **Globus Online** is a SaaS data movement solution (see Sect. 5.7.4) that provides both security and high performance.

5.5.5.4 SSH

Secure Shell allows a user to log in on a remote computer securely over unsecured networks using public-key encryption (discussed in next section). If the log-in is successful, the user becomes a fully qualified user of the remote system, and the user's own machine becomes a relatively passive terminal in this context. The smoothness of such a terminal emulation varies depending on the differences between the local and remote computers. Secure Shell replaced Telnet which was used for terminal emulation until network security became important. Secure Shell enables complete command line control of the remote system to the extent the user's account is authorized.

5.5.5.5 World Wide Web (WWW)

Web **browsing** facilitates user access to remote information resources made available by Web servers. The user interface is typically a **Web browser** that understands the World Wide Web protocols. The **Universal Resource Locator** (**URL**) is used to specify where a resource is located in terms of the protocol to be used, the domain name of the machine it is on, and the name of the information resource within the remote machine. The **Hyper Text Markup Language** (**HTML**) describes what the information should look like when displayed. HTML supports conventional text, font settings, headings, lists, tables, and other display specifications. Within HTML documents, highlighted **buttons** can be defined that point to other HTML documents or services. This **hypertext** facility makes it possible to create a web of cross-referenced works that can be navigated by the user. HTML can also refer to subsidiary documents that contain other types of information—e.g., graphics, equations, images, video, speech— that can be seen or heard if the browser has been augmented with **helpers** or **plug-ins** for the particular format used. The **Hyper Text Transfer Protocol** (**HTTP**) is used to communicate between browser clients and servers and to retrieve HTML documents. Such communications can be **encrypted** to protect sensitive contents (e.g., credit card information or patient information) from external view using protocols such as **Secure Sockets Layer** (**SSL**: recently renamed to **Transport Level Security**, **TLS**) which is used by the HTTPS protocol (and generally shows a "lock" icon when the browser is securely communicating with the host in the URL).

HTML documents can also include small programs written in the Java language, called **applets**, which will execute on the user's computer when referenced. Applets can provide animations and also can compute summaries, merge information, and interact with selected files on the user's computer. The Java language is designed such that operations that might be destructive to the user's machine environment are blocked, but downloading remote and untested software still represents a substantial security risk (see Sect. 5.3). The JavaScript language, distinct from the Java language (unfortunately similarly named) runs in the browser itself (much like protected memory), thus reducing this security risk, and is becoming increasingly powerful, enhancing substantially the capabilities and capacity of the Web browser as a computer platform.

HTML captures many aspects of document description from predefined markup related to the appearance of the document on a display to markup related to internal links, scripts, and other semantic features. To separate appearance-related issues from other types of markup, to provide more flexibility in terms of markup types, and to work toward more open, self-defining document descriptions, a more powerful markup framework, called **eXtensible Markup Language** (XML), has emerged. Coupled with JavaScript, XML further enhances the capabilities of the Web browser itself as a computer platform.

5.5.5.6 Client-Server Interactions

A client–server interaction is a generalization of the four interactions we have just discussed, involving interactions between a client (requesting) machine and a server (responding) machine. A **client–server** interaction, in general, supports collaboration between the user of a local machine and a remote computer. The server provides information and computational services according to some protocol, and the user's computer—the client—generates requests and does complementary processing (such as displaying HTML documents and images). A common function provided by servers is database access. Retrieved information is transferred to the client in response to requests, and then the client may perform specialized analyses on the data. The final results can be stored locally, printed, or mailed to other users.

5.6 Data and System Security

Medical records contain much information about us. These documents and databases include data ranging from height and weight measurements, blood pressures, and notes regarding bouts with the flu, cuts, or broken bones to information about topics such as fertility and abortions, emotional problems and psychiatric care, sexual behaviors, sexually transmitted diseases, human immunodeficiency virus (HIV) status, substance abuse, physical abuse, and genetic predisposition to diseases. Some data are generally considered to be mundane, others highly sensitive. Within the medical record, there is much information about which any given person may feel sensitive. As discussed in Chap. 10, health information is confidential, and access to such information must be controlled because disclosure could harm us, for example, by causing social embarrassment or prejudice, by affecting our insurability, or by limiting our ability to get and hold a job. Medical data also must be protected against loss. If we are to depend on electronic medical records for care, they must be available whenever and wherever we need care, and the information that they contain must be accurate and up to date. Orders for tests or treatments must be validated to ensure that they are issued by authorized providers. The records must also support administrative review and provide a basis for legal accountability. These requirements touch on three separate concepts involved in protecting health care information.

Privacy refers to the desire of a person to control disclosure of personal health and other information. **Confidentiality** applies to information—in this context, the ability of a person to control the release of his or her personal health information to a care provider or information custodian under an agreement that limits the further release or use of that information. **Security** is the protection of privacy and confidentiality through a collection of policies, procedures, and safeguards. Security measures enable an organization to maintain the integrity and availability of information systems and to control access to these systems' contents. Health privacy and confidentiality are discussed further in Chap. 10.

Concerns about, and methods to provide, security are part of most computer systems, but health care systems are distinguished by having especially complex considerations for the use and release of information. In general, the security steps taken in a health care information system serve five key functions (National Research Council, 1997), namely availability, accountability, perimeter control, role-limited access, and comprehensibility and control. We discuss each of these functions in turn.

5.6.1 Availability

Availability ensures that accurate and up-to-date information is available when needed at appropriate places. It is primarily achieved to protect against loss of data by ensuring redundancy—performing regular system backups. Because hardware and software systems will never be perfectly reliable, information of long-term value is copied onto archival storage, and copies are kept at remote sites to protect the data in case of disaster. For short-term protection, data can be written on duplicate storage devices. Critical medical systems must be prepared to operate even during environmental disasters. If one of the storage devices is attached to a remote processor, additional protection is conferred. Therefore, it is also important to provide secure housing and alternative power sources for CPUs, storage devices, network equipment, etc. It is also essential to maintain the integrity of the information system software to ensure availability. Backup copies provide a degree of protection against software failures; if a new version of a program damages the system's database, the backups allow operators to roll back to the earlier version of the software and database contents.

Unauthorized software changes—e.g., in the form of **viruses** or **worms**—are also a threat. A virus may be attached to an innocuous program or data file, and, when that program is executed or data file is opened, several actions take place:

1. The viral code copies itself into other files residing in the computer.
2. It attaches these files to outgoing messages, to spread itself to other computers.
3. The virus may collect email addresses to further distribute its copies.
4. The virus may install other programs to destroy or modify other files, often to escape detection.
5. A program installed by a virus may record keystrokes with passwords or other sensitive information, or perform other deleterious actions.

A software virus causes havoc with computer operations, even if it does not do disabling damage, by disturbing operations and system access and by producing large amounts of Internet traffic as it repeatedly distributes itself (what is called a denial of service attack). To protect against viruses, all programs loaded onto the system should be checked against known viral codes and for unexpected changes in size or configuration. It is not always obvious that a virus program has been imported. For example, a word-processing document may include macros that help in formatting the document. Such a macro can also include viral codes, however, so the document can be infected. Spreadsheets, graphical presentations, and so on are also subject to infection by viruses.

5.6.2 Accountability

Accountability for use of medical data can be promoted both by surveillance and by technical controls. It helps to ensure that users are responsible for their access to, and use of, information based on a documented need and right to know. Most people working in a medical environment are highly ethical. In addition, knowledge that access to, and use of, data records are being watched, through scanning of access **audit trails**, serves as a strong impediment to abuse. Technical means to ensure accountability include two additional functions: authentication and authorization.

1. The user is **authenticated** through a positive and unique identification process, such as name and password combination.
2. The authenticated user is **authorized** within the system to perform only certain actions appropriate to his or her role in the health care system—e.g., to search through certain medical records of only patients under his or her care.

Authentication and authorization can be performed most easily within an individual computer system, but, because most institutions operate multiple computers, it is necessary to coordinate these access controls consistently across all the systems. Enterprise-wide and remote access-control standards and systems are available and are being deployed extensively.

5.6.3 Perimeter Definition

Perimeter definition requires that you know who your users are and how they are accessing the information system. It allows the system to control the boundaries of trusted access to an information system, both physically and logically. For health care providers within a small physician practice, physical access can be provided with a minimum of hassle using simple name and password combinations. If a clinician is traveling or at home and needs remote access to a medical record, however, greater care must be taken to ensure that the person is who he or she claims to be and that communications containing sensitive information are not observed inappropriately. But where is the boundary for being considered a trusted insider? Careful control of where the network runs and how users get outside access is necessary. Most organizations install a **firewall** to define the boundary: all sharable computers of the institution are located within the firewall. Anyone who attempts to access a shared system from the outside must first pass through the firewall, where strong authentication and protocol access controls are in place. Having passed this authentication step, the user can then access enabled services within the firewall (still limited by the applicable authorization controls). Even with a firewall in place, it is important for enterprise system administrators to monitor and ensure that the firewall is not bypassed. For example, a malicious intruder could install a new virus within the perimeter, install or use surreptitiously a wireless base station, or load unauthorized software.

Virtual Private Network (**VPN**) technologies offer a powerful way to let bona fide users access information resources remotely. Using a client–server approach, an encrypted communication link is negotiated between the user's client machine and an enterprise server. This approach protects all communications and uses strong authentication to identify the user. No matter how secure the connection is, however, sound security ultimately depends on responsible users and care that increasingly portable computers (laptops, tablets, or handheld devices) are not lost or stolen so that their contents are accessible by unauthorized people. This is the single most common mechanism by which HIPAA violations occur. Health systems are therefore requiring whole machine encryption on portable devices.

Strong authentication and authorization controls depend on cryptographic technologies. **Cryptographic encoding** is a primary tool for protecting data that are stored or are transmitted over communication lines. Two kinds of cryptography are in common use— secret-key cryptography and public-key cryptography. In **secret-key cryptography**, the same key is used to encrypt and to decrypt information. Thus, the key must be kept secret, known to only the sender and intended receiver of information. In **public-key cryptography**, two keys are used, one to encrypt the information and a second to decrypt it. One is kept secret. The other one can be made publicly available. It is thus asymmetric and one end of the transaction can be proven to have been done by a specific entity. By using this twice (four keys), one can certify both the sender and receiver. This arrangement leads to important services in addition to the exchange of sensitive information, such as digital signatures (to certify authorship), content validation (to prove that the contents of a message have not been changed), and nonrepudiation (to ensure that an action cannot be denied as having been done by the actor). Under either scheme, once data are encrypted, a key is needed to decode and make the information legible and suitable for processing.

Keys of longer length provide more security, because they are harder to guess. Because powerful computers can help intruders to test millions of candidate keys rapidly, single-layer encryption with keys of 56-bit length (the length prescribed by the 1975 **Data Encryption Standard** (**DES**)) are no longer considered secure, and keys of 128 and even 256 bits are routine. If a key is lost, the information encrypted with the key is effectively lost as well. If a key is stolen, or if too many copies of the key exist for them to be tracked, unauthorized people may gain access to information. Holding the keys in **escrow** by a trusted party can provide some protection against loss.

Cryptographic tools can be used to control authorization as well. The authorization information may be encoded as digital **certificates**, which then can be validated with a certification authority and checked by the services so that the services do not need to check the authorizations themselves. Centralizing authentication and authorization functions simplifies the coordination of access control, allows for rapid revocation of privileges as needed, and reduces the possibility of an intruder finding holes in the system. A central authentication or authorization server must itself be guarded and managed with extreme care, however, so that the chain of access-control information is not broken. Modern browsers contain the public certificates for major certificate authorities thus enabling them to check the veracity of Web sites using HTTPS (enabling the locked browser icon discussed earlier).

5.6.4 Role-Limited Access

Role-limited access control is based on extensions of authorization schemes. It allows access for personnel to only that information essential to the performance of their jobs and limits the real or perceived temptation to access information beyond a legitimate need. Even when overall system access has been authorized and is protected, further checks must be made to control access to specific data within the record. A medical record is partitioned according to access criteria based upon use privileges; the many different collaborators in health care all have diverse needs for the information collected in the medical record. Examples of valid access privileges include the following:

- Patients: the contents of their own medical records
- Community physicians: records of their patients
- Specialty physicians: records of patients referred for consultations
- Public health agencies: incidences of communicable diseases
- Medical researchers: consented data, or by waiver of authorization for patient groups approved by an **Institutional Review Board** (**IRB**)
- Billing clerks: records of services, with supporting clinical documentation as required by insurance companies
- Insurance payers: justifications of charges

Different types of information kept in medical records have different rules for release, as determined by state and federal law such as provisions of the Health Insurance Portability and Accountability Act (HIPAA) and as set by institutional policy following legal and ethical considerations and the IRB.

5.6.5 Comprehensibility and Control

Comprehensibility and Control ensures that record owners, data stewards, and patients can understand and have effective control over appropriate aspects of information confidentiality and access. From a technological perspective, while authentication and access control are important control mechanisms, audit trails are perhaps the most important means for allowing record owners and data stewards to understand whether data is being accessed correctly or incorrectly. Many hospitals also allow employees to review who has accessed their personal records. This ability both reassures employees and builds awareness of the importance of patient privacy.

An audit trail contains records indicating what data was accessed, when, by who, from where, and if possible some indication of the reason for the access. If backed by strong authentication and protected against improper deletion, such records can both provide a strong disincentive for improper access (because individuals will know that accesses will be recorded) and allow responsible parties to detect inadequate controls.

5.7 Distributed System Architectures

As noted in Sect. 5.5.5, computer networks have become fundamental to both health care and biomedical research, allowing for quasi-instantaneous information sharing and communication within and between clinical and research institutions. Applications such as the Web, email, videoconferencing, and SSH for remote access to computers are all widely used to reduce barriers associated with physical separation.

As health care becomes more information driven, we see increased interest in not just enabling ad hoc access to remote data, but in linking diverse information systems into **distributed systems**. A distributed system links multiple computer systems in such a way that they can function, to some extent at least, as a single information system. The systems to be linked may live within a single institution (e.g., an electronic medical record system operated by a hospital's information technology services organization, a pathology database operated by the Department of Pathology, and a medical PACS system operated by the Department of Radiology); within multiple institutions of similar type (e.g., electronic medical record systems operated by different hospitals); or within institutions of quite different types (e.g., a hospital and a third-party cloud (see Sect. 5.7.5) service provider). They may be linked for a range of reasons: for example, to federate database systems, so as to permit federated queries (see Sect. 5.7.2) across different databases—e.g., to find records corresponding to multiple admissions for a single patient at different hospitals, or to identify potential participants in a clinical trial; to enable cross-institutional workflows, such as routing of infectious disease data to a public health organization; or to outsource expensive tasks to third-party providers, as when a medical center maintains an off-site backup of its databases in a cloud provider.

Distributed systems can be challenging to construct and operate. They are, first of all, networked systems, and thus are inevitably subject to a wide range of failures. Yet more seriously, they commonly span administrative boundaries, and in the absence of strong external control and/or incentives, different administrative units tend to adopt distinct and incompatible technical solutions, information representations, policies, and governance structures. These problems seem to be particularly prevalent within health care, which despite much effort towards standardization over many decades, remains dominated by non-standards-based products. This situation can make even point-to-point integration of two systems within a single hospital challenging; the creation of large-scale distributed systems that span many institutions remains difficult.

To illustrate the types of problems that may exist, consider a distributed system that is intended to provide a unified view of electronic medical records maintained in databases D_A and D_B at two hospitals, H_A and H_B. We may find that:

- Databases D_A and D_B use different database management systems (see Sect. 5.4), with different access protocols and query languages
- Databases D_A and D_B use different schemas and vocabularies to represent patient information
- Hospitals H_A and H_B assign different identifiers to the same patient
- Hospital H_A does not allow remote access to its electronic medical record system, restricting access to computers located within its firewall (see Sect. 5.4)
- Hospital H_B interprets HIPAA rules as preventing information about its patients to be shared with personnel who are not H_B employees
- Hospitals H_A and H_B both have a governance structure that reviews proposed changes to computer and data systems ahead of implementation, but neither includes representation from the developers of the distributed system.

As this brief and partial list shows, the development of a successful distributed system can require solutions to a wide range of heterogeneity problems, including syntactic (e.g., different formats used to represent data in databases), semantic (e.g., different meanings assigned to a clinical diagnosis), policy, and governance. The use of appropriate distributed system technologies can assist with overcoming these problems, but any effective and robust solution must take into account all aspects of the problem.

5.7.1 Distributed System Programming

To build a distributed system, it must be possible for a program running on one computer to issue a request that results in a program being run in another computer. Various approaches have been pursued over the years to this **distributed system programming** problem. All seek to hide heterogeneity across different platforms via the introduction of a standardized remote procedure call mechanism. Under the covers, communication typically occurs via the Internet protocols described in Sect. 5.5.5; distributed system programming methods build on that base to allow a programmer to name a remote procedure that is to be called (e.g., "query database"), specify arguments to that procedure (e.g., "find patients with influenza"), specify how results are to be returned, provide required security credentials, and so forth—all without the programmer needing to know anything about how the remote program is implemented. Other related methods may allow a programmer to discover what procedures are supported by a particular remote system, or alternatively what remote system should be contacted for a particular purpose.

One distributed system programming approach on which much effort has been spent is **CORBA**, the Common Object Request Broker Architecture. Starting in the mid 1990s, numerous technology providers and adopters formed the Object Management Group (OMG) to define standards for describing remotely accessible procedures (an Interface Definition Language: **IDL**) and for invoking those procedures over a network. Building on that base, a wide range of other standards have been developed defining interaction patterns important for different fields. In the field of health care, the CORBAmed activity defined a range of specifications for such things as personal identification and medical image access. Unfortunately, while CORBA has had some success in certain industries (e.g., manufacturing), a combination of technical limitations in its specifications and inter-company conflict (e.g., Microsoft never adopted CORBA) had prevented it from having broad impact as a distributed system programming technology.

In the mid 2000s, a new technology called **Web Services** emerged that implemented similar concepts to those found in CORBA, but in a simpler and more flexible form. Web Services uses XML (Sect. 5.5.5.5) to encode remote procedure calls and HTTP (Sect. 5.5.5.5) to communicate them, and defines an IDL called Web Services Description Language (**WSDL**). These technologies have seen wide use in many areas, including biomedical research, with many research datasets and analytic procedures being made accessible over the network via Web Service interfaces. However, adoption within clinical systems remains modest for reasons listed above.

Over the past 5 years, the commercial and consumer Web/Cloud market — as typified, for example, by the likes of Google, Facebook, Twitter, and Amazon — has largely converged on a yet simpler architectural approach for defining interfaces to services. This approach is based on a small set of protocol standards and methodologies, namely REST and HTTP, JSON and XML, TLS, and O Auth 2.0. These same methods are seeing increasingly widespread use in health care as well, due to their simplicity and the substantial investment in relevant technologies occurring within industry. At the core of this approach is **REpresentational State Transfer** (**REST**), an HTTP-based approach to distributed system architecture in which components are modeled as *resources* that are named by URLs and with which interaction occurs through standard HTTP actions (POST, GET, PUT, DELETE). For example, Table 5.3 illustrates a simple REST encoding of an interface to a medical record system. This interface models patients as resources with the following form, where {patient-id} is the patient identifier:
/patients/{patient-id}

Table 5.3 Example REST representation of a patient record interface. On the left, the format of each request; on the right, a brief description

Request	Description
GET/patients	Retrieve list of patients
GET or POST/patients/{patient-id}	Get/put profile for a specific patient
GET/patients/{patient-id}/lab results	Request lab results for a specific patient

Thus, to request all patients that we have permission to see we send the following HTTP request:

GET/patients/

while to obtain the profile for a specific patient named NAME we send this request:

GET/patients/NAME

and to update that profile we send the request:

POST/patients/NAME

REST and HTTP define how to name resources (URLs) and messaging semantics, but not how to encode message contents. Two primary message encoding schemes are used: **Java Script Object Notation** (**JSON**) and XML. The following is a potential JSON encoding of a response to a GET/patients/request. The response provides a list of patient identifiers plus (because there may be many more patients that can fit in a single response) information regarding the number of patients included in this response (X), an offset that can be provided in a subsequent request to get new patients (Y), and the number of patients remaining (Z):

```
{"patients":
{"list":[list of patients],
"count":X,
"offset":Y,
"remain":Z}
}
```

5.7.2 Distributed Databases

Distributed databases are a special case of a distributed system. The problem here is to enable queries against data located in multiple databases. Two different methods are commonly used. In a **data warehouse** approach, an **extract-transform-load** (**ETL**) process is used to extract data from the various sources, transform it as required to fit the schema and semantics used by the data warehouse, and then load the transformed data into the data warehouse. In a **federated query** approach, a query is dispatched to the different databases, applied to each of them independently, and then the results combined to get the complete answer. Intermediate components called **mediators** may be used to convert between different syntaxes and semantics used in different systems.

These two approaches have various advantages and disadvantages. The data warehouse requires a potentially expensive ETL process, requires storage for a separate copy of all relevant data, and may not be up to date with all source databases. However, it can permit highly efficient queries against the entirety of the data. The federated query approach can provide access to the latest data from each source database, but requires potentially complex mediator technology.

5.7.3 Parallel Computing

Parallel computers combine many microprocessors and/or GPUs (see Sect. 5.2) to provide an aggregate computing capacity greater than that of a single workstation. The largest such systems available today have more than one million processing cores. While systems of that scale are not used in medicine, parallel computers are becoming more commonly used in biomedical computing as a means of performing large-scale computational simulations and/or analyzing large quantities of data. In basic research, parallel computers are used for such purposes as mining clinical records, genome sequence analysis, protein folding, simulation studies of cell membranes, and modeling of blood flow. In translational research, parallel computers are commonly used to compute parameters for computer programs that are then used in clinical settings, for example, for computed aided diagnosis of mammograms.

When computing over large quantities of data, it is often useful to employ a **parallel database management system**. These systems support the same SQL query language as sequential database management systems (see Sect. 5.5.4.) but can run queries faster when using multiple processors. Another increasingly popular approach to parallel data analysis is to use the **MapReduce** model, popularized by Google and widely available via the free **Hadoop** software. MapReduce programs may be less efficient than equivalent SQL programs but do not require that data be loaded into a database prior to processing.

5.7.4 Grid Computing

Grid computing technologies allow for the federation of many computers and/or data resources in such a way that they can be used in an integrated manner. Grid computing is the foundation, for example, for the worldwide distributed system that analyzes data from the Large Hadron Collider (LHC) in Geneva, Switzerland. The 10 or more petabytes (10×10^{15} bytes) produced per year at the LHC is distributed to several hundred institutions worldwide for analysis. Each institution that participates in this worldwide system has its own local computer system administration team, user authentication system, accounting system, and so forth. Grid technologies bridge these institutional barriers, allowing a user to authenticate once (to "the grid"), and then submit jobs for execution at any or all computers in the grid.

Grid computing is used in many academic campuses to link small and large computer clusters and even idle desktop computers for parallel computing applications. But its biggest use in research is to enable sharing of large quantities of data across institutional boundaries. By addressing the challenges of authentication, access control, and high-speed data movement, grid computing technologies make it possible, for example, to acquire genome sequence from a commercial sequencing provider, transport that data over the network to a cloud computing provider (see Sect. 5.7.5), perform analysis there, and then load results into a database at a researcher's home institution.

5.7.5 Cloud Computing

The late 2000s saw the emergence of successful commercial providers of on-demand computing and software services. This concept is certainly not new: for example, McCarthy first referred to "utility computing" in 1961, various time sharing services provided computing over the network in the 1970s and 1980s, and grid computing provided such services in the 1990s and 2000s. However, it is clear that—perhaps driven by a combination of quasi-ubiquitous high-speed Internet, vastly increased demand from e-commerce, powerful lightweight Web protocols, and an effective business model—cloud computing has achieved large-scale adoption in ways that previous efforts had not. The implications of these developments for medical informatics will surely be profound.

The **National Institutes of Standards and Technology** (NIST) defines cloud computing as "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" (Mell and Grance, 2011). They distinguish between three distinct types of cloud service (see Fig. 5.11):

- *Software as a Service* (*SaaS*) allows the consumer to use the provider's applications running on a cloud infrastructure. Examples include Google mail, Google Docs, Salesforce.com customer relationship management, and a growing number of electronic medical record and practice management systems.
- *Platform as a Service* (*PaaS*) allows the consumer to deploy consumer-created or acquired applications onto the cloud infrastructure, created using programming languages, libraries, services, and tools supported by the provider. Google's App Engine and Salesforce's Force.com are examples of such platforms.
- *Infrastructure as a Service* (*IaaS*) allows the consumer to provision processing, storage, networks, and other fundamental computing resources on which the consumer is able to deploy and run arbitrary software. Amazon Web Services and Microsoft Azure are popular IaaS providers.

Each level of the stack can, and often does, build on services provided by the level below. For example, Google Mail is a SaaS service that runs on compute and storage infrastructure services operated by Google; Globus Online is a SaaS research data management service that runs on infrastructure services operated by Amazon Web Services.

NIST further distinguishes between public cloud providers, which deliver such capabilities

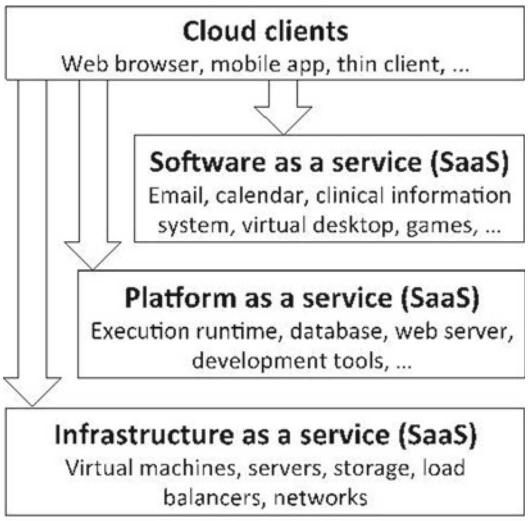

Cloud clients
Web browser, mobile app, thin client, ...
Software as a service (SaaS)
Email, calendar, clinical information system, virtual desktop, games, ...
Platform as a service (SaaS)
Execution runtime, database, web server, development tools, ...
Infrastructure as a service (SaaS)
Virtual machines, servers, storage, load balancers, networks

Fig. 5.11 The NIST taxonomy of cloud providers. SaaS, PaaS, and IaaS providers each offer different types of services to their clients. Cloud services are distinguished by their Web 2.0 interfaces, which can be accessed either via Web browsers or (for access from other programs or scripts) via simple APIs

to anyone, and private cloud providers, which provide such on-demand services for consumers within an organization.

Benefits claimed for cloud computing include increased reliability, higher usability, and reduced cost relative to equivalent software deployed and operated within the consumer's organization, due to expert operations and economies of scale. (IaaS providers such as Amazon charge for computing and storage on a per-usage basis.) Potential drawbacks include security challenges associated with remote operations, lock-in to a remote cloud provider, and potentially higher costs if usage becomes large.

Outside health care, cloud computing has proven particularly popular among smaller businesses, who find that they can outsource essentially all routine information technology functions (e.g., email, Web presence, accounting, billing, customer relationship management) to SaaS providers. Many companies also make considerable use of IaaS from the likes of Amazon Web Services and Microsoft Azure for compute- and data-intensive computations that exceed local capacity. In research, we see the emergence of a growing number of both commercial and non-profit SaaS offerings designed to accelerate common research tasks. For example, Mendeley organizes bibliographic information, while Globus Online provides research data management services.

Similarly, in health care, we see many independent physicians and smaller practices adopting SaaS electronic medical record systems. The relatively high costs and specialized expertise required to operate in-house systems, plus a perception that SaaS providers do a good job of addressing usability and security concerns, seem to be major drivers of adoption. Similarly, a growing number of biomedical researchers are using IaaS for data- and compute-intensive research. Meanwhile, some cloud providers (e.g., Microsoft) are prepared to adhere to security and privacy provisions defined in HIPAA and the HITECH act. Nevertheless, while some hospitals are using IaaS for remote backup (e.g., by storing encrypted database dumps), there is not yet any significant move to outsource major hospital information systems.

5.8 Summary

As we have discussed in this chapter, the synthesis of large-scale information systems is accomplished through the careful construction of hierarchies of hardware and software. Each successive layer is more abstract and hides many of the details of the preceding layer. Simple methods for storing and manipulating data ultimately produce complex information systems that have powerful capabilities. Communication links that connect local and remote computers in arbitrary configurations, and the security mechanisms that span these systems, transcend the basic hardware and software hierarchies. Thus, without worrying about the technical details, users can access a wealth of computational resources and can perform complex information management tasks, such as storing and retrieving, communicating, authorizing, and processing information. As the technology landscape evolves and computer architectures continuously increase in complexity, they will also increasingly, necessarily, hide that complexity from the user. Therefore, it is paramount that systems designers, planners, and implementers remain sufficiently knowledgeable about the underlying mechanisms and distinguishing features of computing architectures so as to make optimal technology choices.

Suggested Readings

Council, N.R. (1997). *For the record: Protecting electronic health information*. Washington, DC: National Academy Press. This report documents an extensive study of current security practices in US health care settings and recommends significant changes. It sets guidelines for policies, technical protections, and legal standards for acceptable access to, and use of, health care information. It is well suited for lay, medical, and technical readers who are interested in an overview of this complex topic.

Garcia-Molina, H., Ullman, J.D., & Widom, J.D. (2008). *Database systems: The complete book* (2nd ed.). Englewood Cliffs: Prentice-Hall. The first half of the book provides in-depth coverage of databases from the point of view of the database designer, user, and application programmer. It covers the latest database standards SQL:1999, SQL/PSM, SQL/CLI, JDBC, ODL, and XML, with broader coverage of SQL than most other texts. The second half of the book provides in-depth coverage of databases from the point of view of the DBMS implementer. It focuses on storage structures, query processing, and transaction management. The book covers the main techniques in these areas with broader coverage of query optimization than most other texts, along with advanced topics including multidimensional and bitmap indexes, distributed transactions, and information-integration techniques.

Hennessy, J.L., & Patterson, D.A. (2011). *Computer architecture: a quantitative approach* (5th ed.). San Francisco: Morgan Kaufmann. This technical book provides an in-depth explanation of the physical and conceptual underpinnings of computer hardware and its operation. It is suitable for technically oriented readers who want to understand the details of computer architecture.

Mell, P. and Grance, T. (2011). *The NIST definition of cloud computing*. NIST Special Publication 800–145, National Institute of Standards and Technology. This brief document provides a concise and clear definition of cloud computing.

Tanenbaum, A., & Wetherall, D. (2010). *Computer networks* (5th ed.). Englewood Cliffs: Prentice-Hall. The heavily revised edition of a classic textbook on computer communications, this book is well organized, clearly written, and easy to understand. It first describes the physical layer of networking and then works up to network applications, using real-world example networks to illustrate key principles. Covers applications and services such as email, the domain name system, the World Wide Web, voice over IP, and video conferencing.

Teorey, T., Lightstone, S., Nadeau, T., & Jagadish, H. (2011). *Database modeling and design: Logical design* (5th ed.). San Francisco: Elsevier. This text provides and excellent and compact coverage of multiple topics regarding database architectures, including core concepts, universal modeling language, normalization, entity-relationship diagrams, SQL, and data warehousing.

Wiederhold, G., & Clayton, P.D. (1985). Processing biological data in real time. M.D. *Computing, 2*(6), 16–25. This article discusses the principles and problems of acquiring and processing biological data in real time. It covers much of the material discussed in the signal-processing section of this chapter and it provides more detailed explanations of analog-to-digital conversion and Fourier analysis.

Questions for Discussion

1. What are four considerations in deciding whether to keep data in active versus archival storage?
2. Explain how operating systems and cloud architectures insulate users from hardware changes.
3. Discuss what characteristics determine whether computer clusters or cloud architectures are better for scaling a given computational problem.
4. Explain how grid computing facilitates federation of resources.
5. Describe the architectural advantages and disadvantages of different computing environments.
6. Explain how REST and XML enable flexibility, modularity, and scale.
7. How can you prevent inappropriate access to electronic medical record information? How can you detect that such inappropriate access might have occurred?
8. You are asked whether a medical practice should outsource its information technology functions to third party cloud providers. What factors would enter into your recommendation?