

Figure 1.8 Symmetric multiprocessing architecture.

In addition, one chip with multiple cores uses significantly less power than multiple single-core chips, an important issue for mobile devices as well as laptops.

In Figure 1.9, we show a dual-core design with two cores on the same processor chip. In this design, each core has its own register set, as well as its own local cache, often known as a level 1, or L1, cache. Notice, too, that a level 2 (L2) cache is local to the chip but is shared by the two processing cores. Most architectures adopt this approach, combining local and shared caches, where local, lower-level caches are generally smaller and faster than higher-level shared

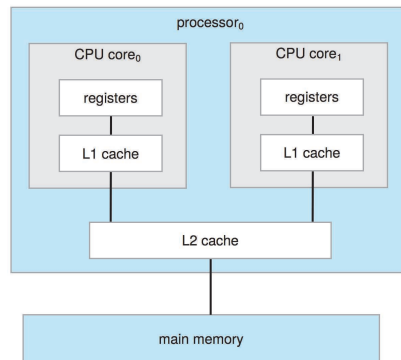


Figure 1.9 A dual-core design with two cores on the same chip.

DEFINITIONS OF COMPUTER SYSTEM COMPONENTS

- **CPU**—The hardware that executes instructions.
- **Processor**—A physical chip that contains one or more CPUs.
- **Core**—The basic computation unit of the CPU.
- **Multicore**—Including multiple computing cores on the same CPU.
- **Multiprocessor**—Including multiple processors.

Although virtually all systems are now multicore, we use the general term *CPU* when referring to a single computational unit of a computer system and *core* as well as *multicore* when specifically referring to one or more cores on a CPU.

caches. Aside from architectural considerations, such as cache, memory, and bus contention, a multicore processor with N cores appears to the operating system as N standard CPUs. This characteristic puts pressure on operating-system designers—and application programmers—to make efficient use of these processing cores, an issue we pursue in Chapter 4. Virtually all modern operating systems—including Windows, macOS, and Linux, as well as Android and iOS mobile systems—support multicore SMP systems.

Adding additional CPUs to a multiprocessor system will increase computing power; however, as suggested earlier, the concept does not scale very well, and once we add too many CPUs, contention for the system bus becomes a bottleneck and performance begins to degrade. An alternative approach is instead to provide each CPU (or group of CPUs) with its own local memory that is accessed via a small, fast local bus. The CPUs are connected by a **shared system interconnect**, so that all CPUs share one physical address space. This approach—known as **non-uniform memory access**, or **NUMA**—is illustrated in Figure 1.10. The advantage is that, when a CPU accesses its local memory, not only is it fast, but there is also no contention over the system interconnect. Thus, NUMA systems can scale more effectively as more processors are added.

A potential drawback with a NUMA system is increased latency when a CPU must access remote memory across the system interconnect, creating a possible performance penalty. In other words, for example, CPU₀ cannot access the local memory of CPU₃ as quickly as it can access its own local memory, slowing down performance. Operating systems can minimize this NUMA penalty through careful CPU scheduling and memory management, as discussed in Section 5.5.2 and Section 10.5.4. Because NUMA systems can scale to accommodate a large number of processors, they are becoming increasingly popular on servers as well as high-performance computing systems.

Finally, **blade servers** are systems in which multiple processor boards, I/O boards, and networking boards are placed in the same chassis. The difference between these and traditional multiprocessor systems is that each blade-processor board boots independently and runs its own operating system. Some blade-server boards are multiprocessor as well, which blurs the lines between

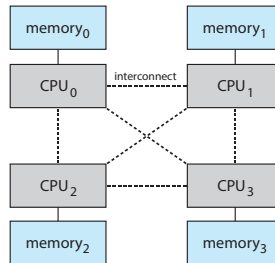


Figure 1.10 NUMA multiprocessing architecture.

types of computers. In essence, these servers consist of multiple independent multiprocessor systems.

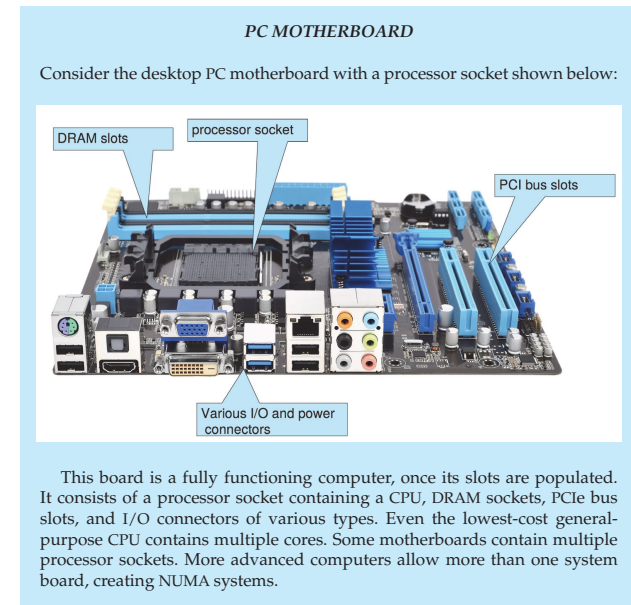
1.3.3 Clustered Systems

Another type of multiprocessor system is a **clustered system**, which gathers together multiple CPUs. Clustered systems differ from the multiprocessor systems described in Section 1.3.2 in that they are composed of two or more individual systems—or nodes—joined together; each node is typically a multicore system. Such systems are considered **loosely coupled**. We should note that the definition of **clustered** is not concrete; many commercial and open-source packages wrestle to define what a clustered system is and why one form is better than another. The generally accepted definition is that clustered computers share storage and are closely linked via a local-area network LAN (as described in Chapter 19) or a faster interconnect, such as InfiniBand.

Clustering is usually used to provide **high-availability service**—that is, service that will continue even if one or more systems in the cluster fail. Generally, we obtain high availability by adding a level of redundancy in the system. A layer of cluster software runs on the cluster nodes. Each node can monitor one or more of the others (over the network). If the monitored machine fails, the monitoring machine can take ownership of its storage and restart the applications that were running on the failed machine. The users and clients of the applications see only a brief interruption of service.

High availability provides increased reliability, which is crucial in many applications. The ability to continue providing service proportional to the level of surviving hardware is called **graceful degradation**. Some systems go beyond graceful degradation and are called **fault tolerant**, because they can suffer a failure of any single component and still continue operation. Fault tolerance requires a mechanism to allow the failure to be detected, diagnosed, and, if possible, corrected.

Clustering can be structured asymmetrically or symmetrically. In **asymmetric clustering**, one machine is in **hot-standby mode** while the other is running the applications. The hot-standby host machine does nothing but monitor the active server. If that server fails, the hot-standby host becomes the active



server. In **symmetric clustering**, two or more hosts are running applications and are monitoring each other. This structure is obviously more efficient, as it uses all of the available hardware. However, it does require that more than one application be available to run.

Since a cluster consists of several computer systems connected via a network, clusters can also be used to provide **high-performance computing** environments. Such systems can supply significantly greater computational power than single-processor or even SMP systems because they can run an application concurrently on all computers in the cluster. The application must have been written specifically to take advantage of the cluster, however. This involves a technique known as **parallelization**, which divides a program into separate components that run in parallel on individual cores in a computer or computers in a cluster. Typically, these applications are designed so that once each computing node in the cluster has solved its portion of the problem, the results from all the nodes are combined into a final solution.

Other forms of clusters include parallel clusters and clustering over a wide-area network (WAN) (as described in Chapter 19). Parallel clusters allow multiple hosts to access the same data on shared storage. Because most oper-

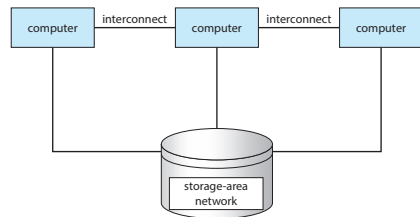


Figure 1.11 General structure of a clustered system.

ating systems lack support for simultaneous data access by multiple hosts, parallel clusters usually require the use of special versions of software and special releases of applications. For example, Oracle Real Application Cluster is a version of Oracle's database that has been designed to run on a parallel cluster. Each machine runs Oracle, and a layer of software tracks access to the shared disk. Each machine has full access to all data in the database. To provide this shared access, the system must also supply access control and locking to ensure that no conflicting operations occur. This function, commonly known as a **distributed lock manager (DLM)**, is included in some cluster technology.

Cluster technology is changing rapidly. Some cluster products support thousands of systems in a cluster, as well as clustered nodes that are separated by miles. Many of these improvements are made possible by **storage-area networks (SANs)**, as described in Section 11.7.4, which allow many systems to attach to a pool of storage. If the applications and their data are stored on the SAN, then the cluster software can assign the application to run on any host that is attached to the SAN. If the host fails, then any other host can take over. In a database cluster, dozens of hosts can share the same database, greatly increasing performance and reliability. Figure 1.11 depicts the general structure of a clustered system.

1.4 Operating-System Operations

Now that we have discussed basic information about computer-system organization and architecture, we are ready to talk about operating systems. An operating system provides the environment within which programs are executed. Internally, operating systems vary greatly, since they are organized along many different lines. There are, however, many commonalities, which we consider in this section.

For a computer to start running—for instance, when it is powered up or rebooted—it needs to have an initial program to run. As noted earlier, this initial program, or bootstrap program, tends to be simple. Typically, it is stored within the computer hardware in firmware. It initializes all aspects of the system, from CPU registers to device controllers to memory contents. The bootstrap program must know how to load the operating system and how to

HADOOP

Hadoop is an open-source software framework that is used for distributed processing of large data sets (known as **big data**) in a clustered system containing simple, low-cost hardware components. Hadoop is designed to scale from a single system to a cluster containing thousands of computing nodes. Tasks are assigned to a node in the cluster, and Hadoop arranges communication between nodes to manage parallel computations to process and coalesce results. Hadoop also detects and manages failures in nodes, providing an efficient and highly reliable distributed computing service.

Hadoop is organized around the following three components:

1. A distributed file system that manages data and files across distributed computing nodes.
2. The YARN (“Yet Another Resource Negotiator”) framework, which manages resources within the cluster as well as scheduling tasks on nodes in the cluster.
3. The **MapReduce** system, which allows parallel processing of data across nodes in the cluster.

Hadoop is designed to run on Linux systems, and Hadoop applications can be written using several programming languages, including scripting languages such as PHP, Perl, and Python. Java is a popular choice for developing Hadoop applications, as Hadoop has several Java libraries that support MapReduce. More information on MapReduce and Hadoop can be found at <https://hadoop.apache.org/docs/r1.2.1/mapred.tutorial.html> and <https://hadoop.apache.org>

start executing that system. To accomplish this goal, the bootstrap program must locate the operating-system kernel and load it into memory.

Once the kernel is loaded and executing, it can start providing services to the system and its users. Some services are provided outside of the kernel by system programs that are loaded into memory at boot time to become **system daemons**, which run the entire time the kernel is running. On Linux, the first system program is “**systemd**,” and it starts many other daemons. Once this phase is complete, the system is fully booted, and the system waits for some event to occur.

If there are no processes to execute, no I/O devices to service, and no users to whom to respond, an operating system will sit quietly, waiting for something to happen. Events are almost always signaled by the occurrence of an interrupt. In Section 1.2.1 we described hardware interrupts. Another form of interrupt is a **trap** (or an **exception**), which is a software-generated interrupt caused either by an error (for example, division by zero or invalid memory access) or by a specific request from a user program that an operating-system service be performed by executing a special operation called a **system call**.