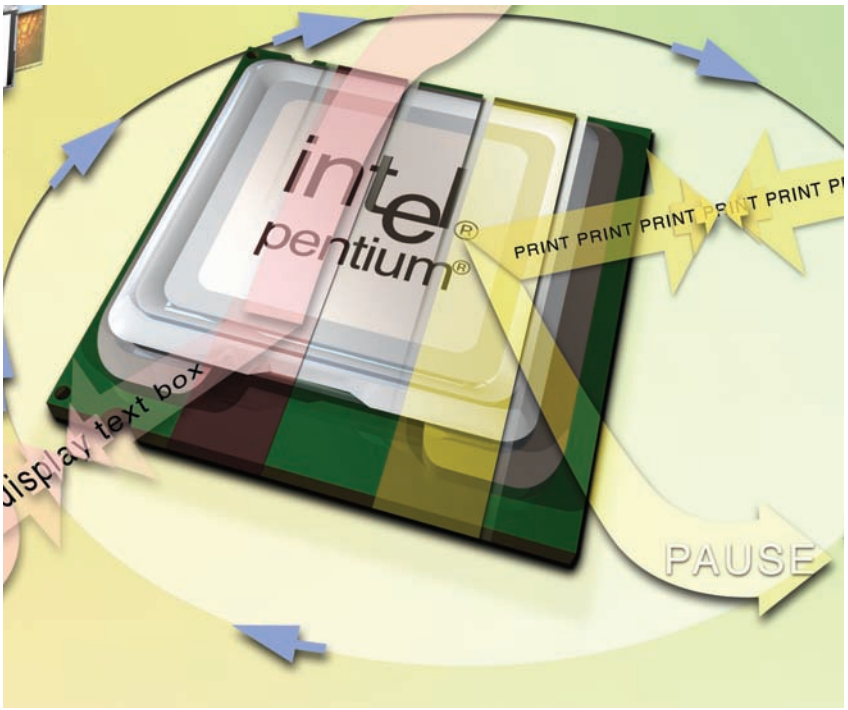
CHAPTER 5

Managing System Resources



THE part of Windows Vista that you see—the Vista desktop—is just part of the operating system. Behind this interface (under the hood, if you will) is the guts of the beast. Vista is more than just a pretty interface; it's a robust engine that makes all the components of your computer system run.

The Windows Vista engine works by managing the data flow to all the different pieces of hardware (including key subsystems) of your PC. Vista manages the instructions that are fed to the central processing unit; the applications and drivers that are stored in system memory; the external and internal devices that are connected to your computer; and the disk drives that your computer uses to store your data. Think of Windows as a virtual traffic cop, managing the flow of data and instructions; it's all quite complex, yet Windows handles any given operation in the blink of an eye.

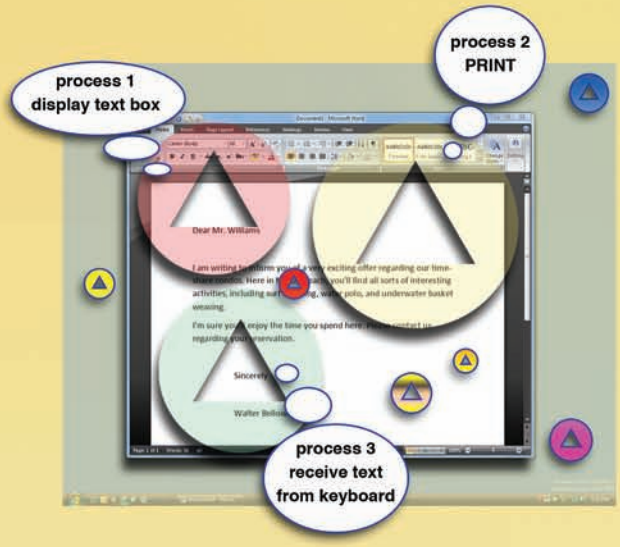
Consider, for example, the simple act of clicking your mouse to open a dialog box. When you press your finger down on that mouse button, it sends an electric signal from the mouse to your computer. That signal is translated into a specific instruction in binary code, thanks to a small software program called a device driver, which is part of the Windows operating system. Windows takes the instruction from the device driver, interprets what it means, and then forwards the instruction to your computer's CPU. The CPU processes the instruction, and then feeds the result back to Windows. Windows then accesses the currently running program, which is temporarily stored in system memory, and tells it to open the dialog box. The program does as it's told, and feeds back to Windows the necessary information about what dialog box to open, and where. Windows takes that instruction, processes it as necessary, and then feeds the graphic information about the dialog box to a different device driver—this one for your PC's video card. The video device driver translates Windows' instruction into the appropriate electronic signal, and the dialog box appears on your computer monitor screen. This whole process occurs in the blink of an eye.

Although this sounds rather complicated, it's actually an example of a very simple—and very common—operation. Windows manages dozens, if not hundreds, of these operations every hour, all in the background, all without you knowing what's going on behind the scenes. The operating system just does its thing, routing the proper instructions to the proper devices and systems, making sure that no one operation gets in the way of any other one. There's a lot of interrupting and pausing and restarting, but that's the nature of the game—and it all happens behind the scenes, without troubling you, the user.

It's all in a days work, as far as Windows Vista is concerned.

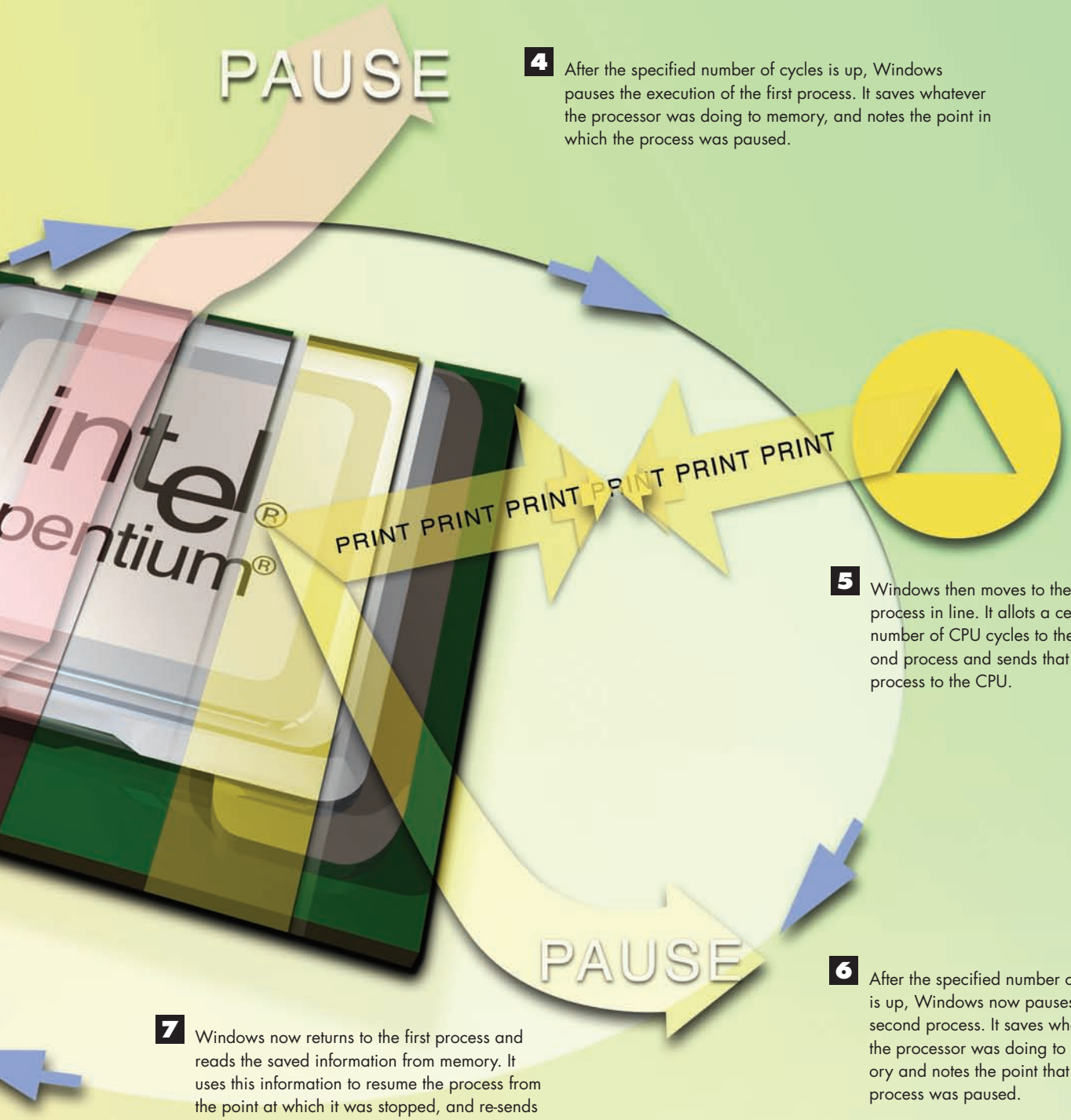
How Windows Manages the CPU

1 All operations that your computer undertakes are broken down into processes that perform some individual action. In the case of an application, such as Microsoft Word or Internet Explorer, several processes are typically involved. The application itself may contain one or more processes, but also cause several other processes to begin—typically for related tasks, such as accessing the modem, activating the printer, and so on. Your system's central processing unit (CPU) manages these processes. At any given time, Windows is running dozens of background processes to handle your system's memory management, disk management, networking, virus checking, and so on.



2 Windows is a multitasking operating system. This means that multiple processes are run at virtually the same time; this is how you can simultaneously surf the Web, listen to digital music, and print a document. Windows' job is to arrange the execution of all these processes so that they seem to be running concurrently—when in fact, they're being processed sequentially.

3 When multiple processes are running at the same time, Windows assigns each process a slice of the CPU's time. It starts by allotting a certain number of CPU execution cycles to the first process and sends that process to the CPU.



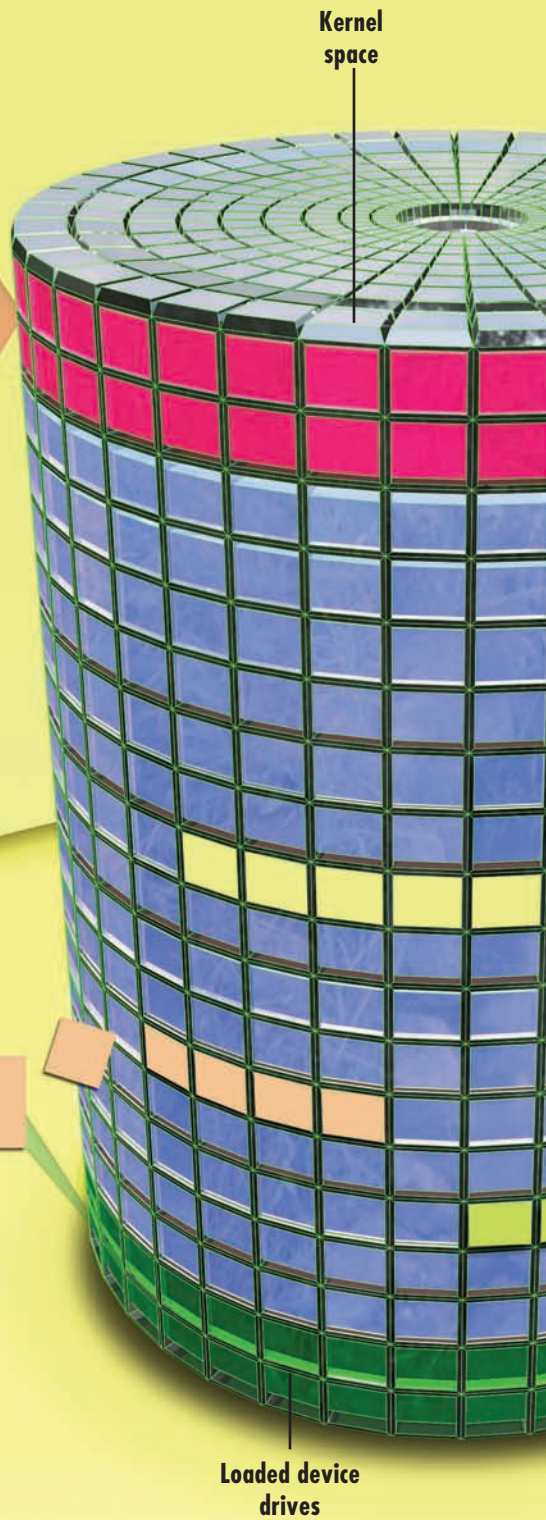
4 After the specified number of cycles is up, Windows pauses the execution of the first process. It saves whatever the processor was doing to memory, and notes the point in which the process was paused.

5 Windows then moves to the second process in line. It allots a certain number of CPU cycles to the second process and sends that process to the CPU.

6 After the specified number of cycles is up, Windows now pauses the second process. It saves whatever the processor was doing to memory and notes the point that the process was paused.

7 Windows now returns to the first process and reads the saved information from memory. It uses this information to resume the process from the point at which it was stopped, and re-sends the process to the CPU. This "process-swapping" procedure is then repeated over and over until both processes end.

How Windows Manages Memory with SuperFetch

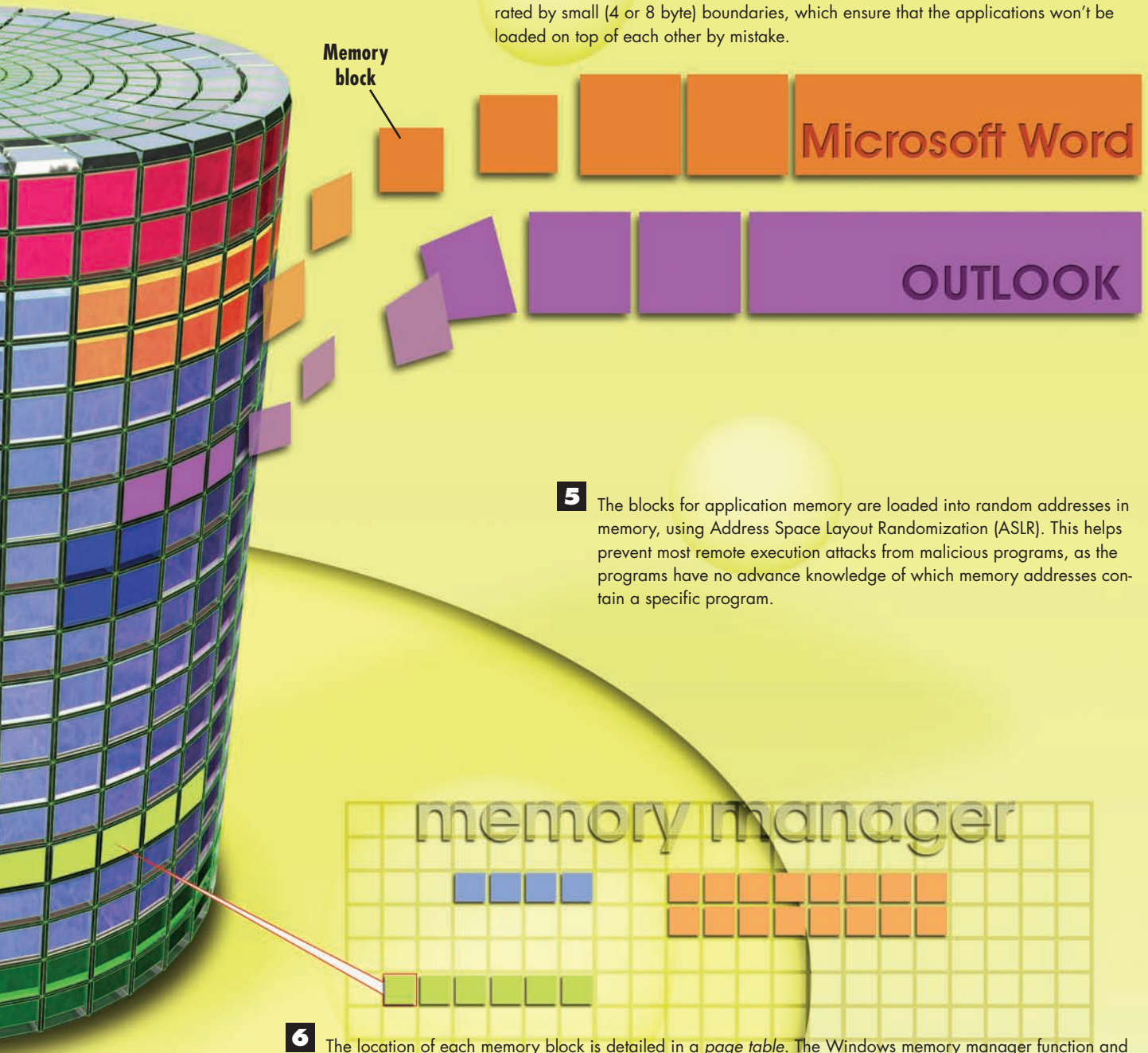


1 When Windows first starts up, the Windows kernel (that part of the operating system responsible for securely managing running programs) is the first item loaded into system memory. The kernel loads at the very top of the available system memory, "backing up" far enough to meet the needs of the operating system. This area of memory is called the *system space* or *kernel space*.

2 After loading the kernel, Windows now moves to the bottom of the pool of system memory and starts loading the various device drivers needed to control your computer's hardware subsystems.

3 The remaining memory between the device drivers and the Windows kernel is free for the loading of software applications. In Windows Vista, SuperFetch technology automatically loads your most-used applications into memory when Windows first launches—instead of waiting for you to open the program manually. By pre-loading a program into a memory, it starts up much quicker when you later open the program. SuperFetch uses an intelligent prioritization scheme to understand which programs you use most often, and it can even differentiate which programs you're likely to use at different times.

4 Each application is subdivided into smaller blocks of memory, each about 2 kilobytes in size; each block is loaded separately, into an individual location. The blocks are separated by small (4 or 8 byte) boundaries, which ensure that the applications won't be loaded on top of each other by mistake.



5 The blocks for application memory are loaded into random addresses in memory, using Address Space Layout Randomization (ASLR). This helps prevent most remote execution attacks from malicious programs, as the programs have no advance knowledge of which memory addresses contain a specific program.

6 The location of each memory block is detailed in a *page table*. The Windows memory manager function and your computer's CPU use the page table to map the location of memory blocks. After the memory manager finds a specific page table entry, it can then locate and access that memory block in physical memory.

How Windows Manages System Devices

1 To manage all the hardware not on your PC's motherboard, Windows uses a special software program, called a *device driver*. The driver functions as a translator between the instructions issued by Windows (and Windows' applications) and the electrical signals that run the hardware subsystems. Without device drivers, all the instructions for managing every possible hardware device would have to be hard-coded into the Windows operating system. Because of the driver architecture, you only need to load those drivers necessary for the hardware on your specific computer system.

2 When a new device is added to your computer system, Windows installs a device driver for that item. If it's a common driver, it may be included with Windows itself; however, it may instead be included on the device's installation CD; or it may have to be downloaded from the manufacturer's website.

3 Once installed, each driver is added to the Windows Vista *Driver Store*. This component, new to Vista, is a repository of all installed driver packages for your particular installation of Windows. The Driver Store ensures that if you need to repair or reinstall a particular device driver, you won't need to locate and use the original installation disk; Windows will automatically access the original driver stored in the Driver Store.

4 In regular use, all installed device drivers are loaded into system memory when Windows starts up. Once loaded, the driver passes information to Windows on which device it's talking to and what that device can do. It then becomes an adjunct to the operating system, sitting dormant while it waits for a request from Windows.

Function Dispatch Table

Task Driver

Task Driver

Task Driver

Print Driver

5 Information about a particular device is stored in system memory in what is called a *driver object*. The driver object supplies data that completes the *Function Dispatch Table*, which is a database that tells Windows about each driver.

6 When Windows needs to perform a particular function, it first examines the Function Dispatch Table to determine which device can best do the job.

INPUT/OUTPUT REQUEST PACKET

print
print
print
print
print
print
print

7 After identifying the appropriate device, Windows sends the instruction to the device driver, in the form of an Input/Output Request Packet (IRP).

Task Driver

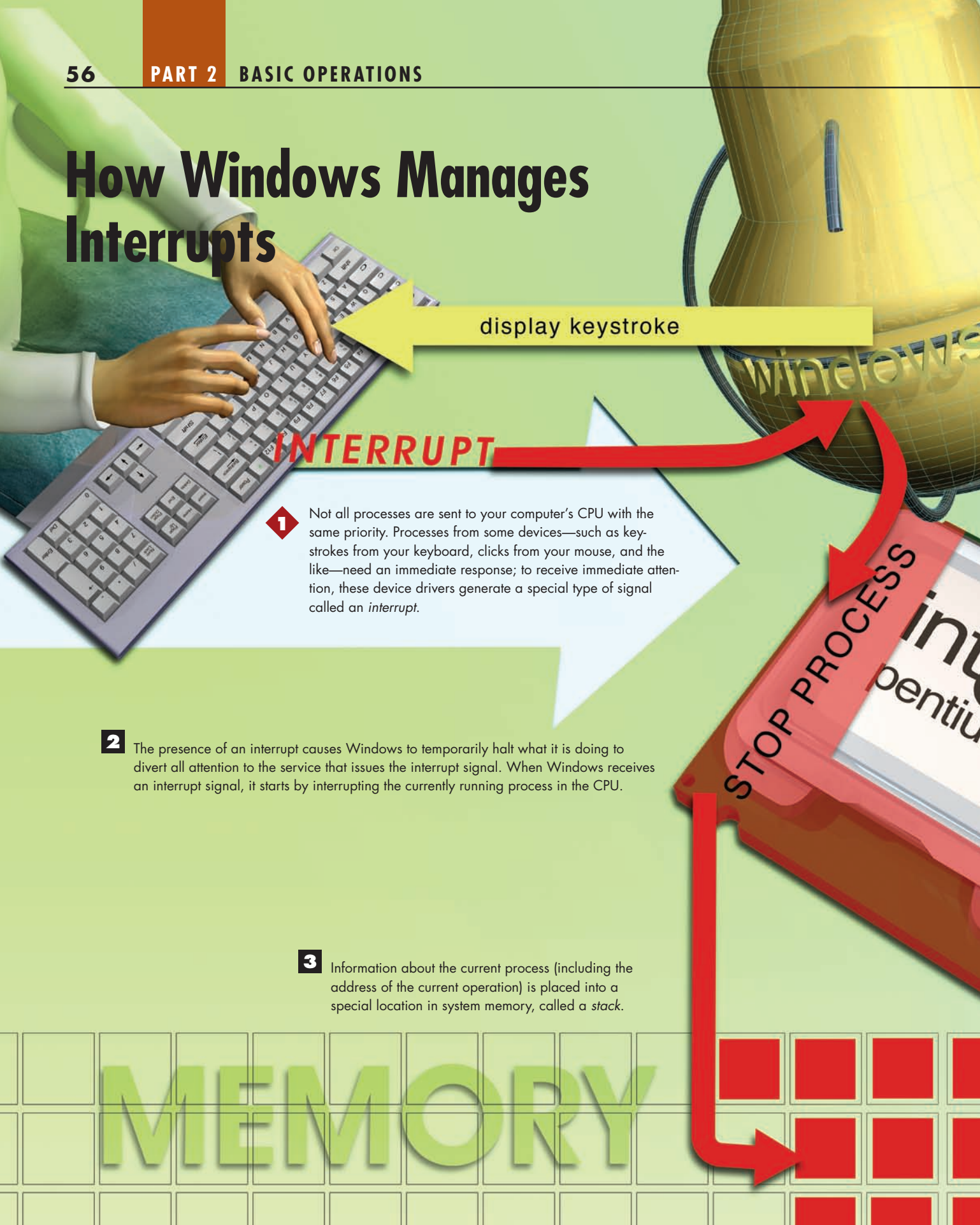
Task Driver

print driver

print

8 The device driver translates the instructions from Windows, and then commands the device to perform the appropriate function.

How Windows Manages Interrupts



1 Not all processes are sent to your computer's CPU with the same priority. Processes from some devices—such as keystrokes from your keyboard, clicks from your mouse, and the like—need an immediate response; to receive immediate attention, these device drivers generate a special type of signal called an *interrupt*.

2 The presence of an interrupt causes Windows to temporarily halt what it is doing to divert all attention to the service that issues the interrupt signal. When Windows receives an interrupt signal, it starts by interrupting the currently running process in the CPU.

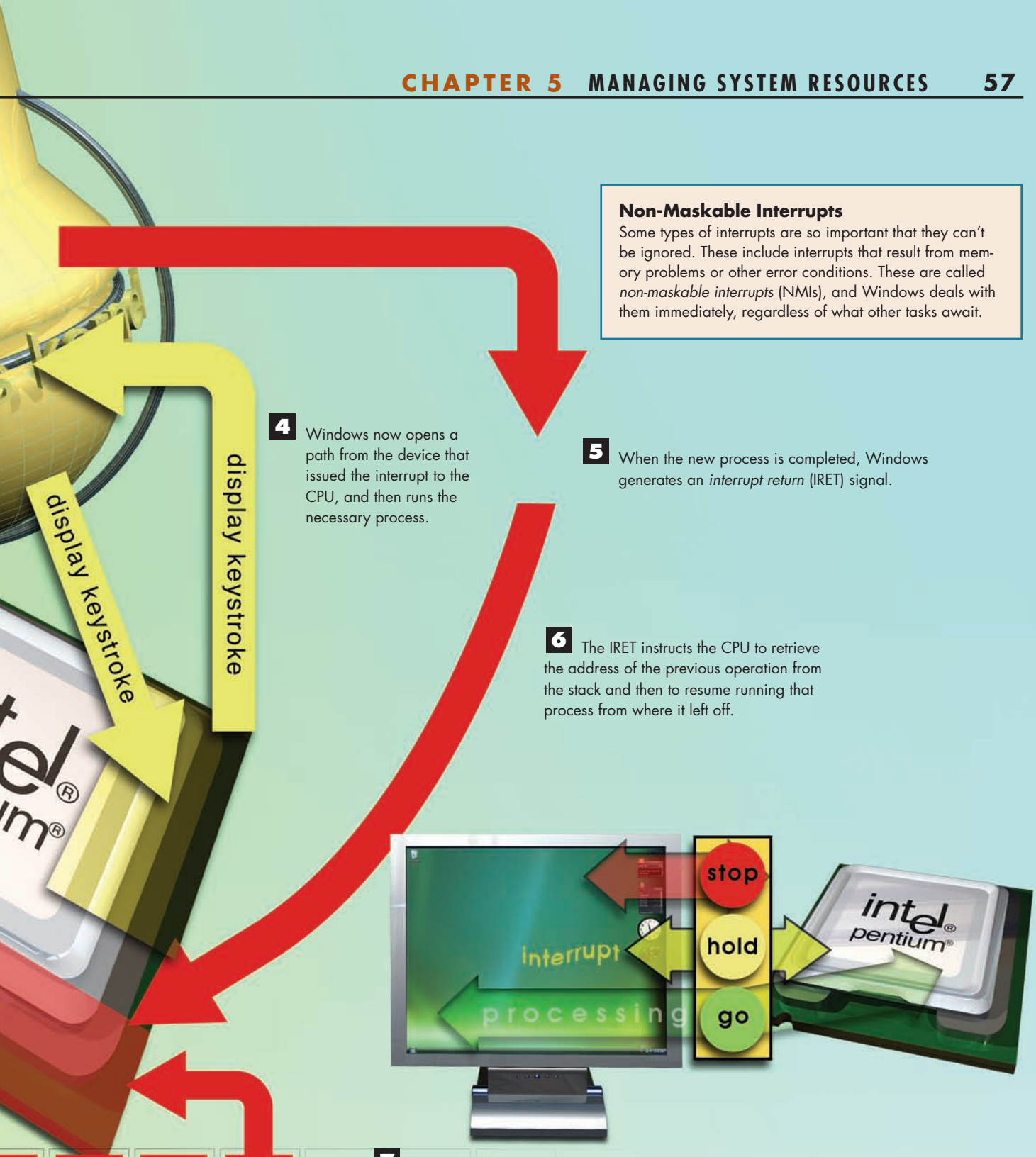
3 Information about the current process (including the address of the current operation) is placed into a special location in system memory, called a *stack*.

Non-Maskable Interrupts
Some types of interrupts are so important that they can't be ignored. These include interrupts that result from memory problems or other error conditions. These are called *non-maskable interrupts* (NMIs), and Windows deals with them immediately, regardless of what other tasks await.

4 Windows now opens a path from the device that issued the interrupt to the CPU, and then runs the necessary process.

5 When the new process is completed, Windows generates an *interrupt return* (IRET) signal.

6 The IRET instructs the CPU to retrieve the address of the previous operation from the stack and then to resume running that process from where it left off.



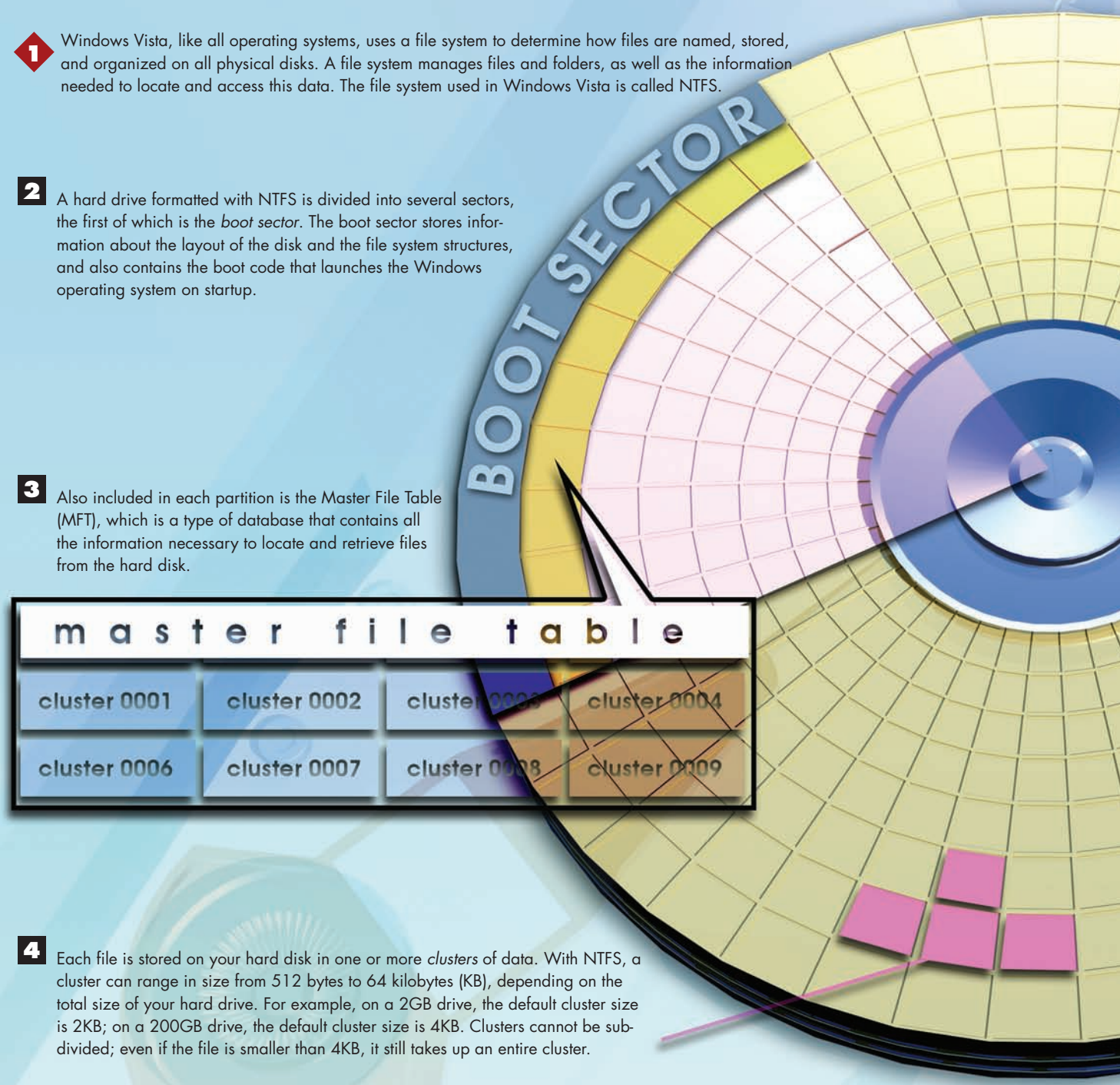
7 In many instances, Windows is capable of *masking* the interrupt so as not to interfere with already running processes. When the CPU is running an important process, Windows intercepts the interrupt signal and holds it so that the current process can be finished as quickly as possible. As soon as the process is done running, Windows then puts through the interrupt request.

How Windows Manages Disk Drives and Data with NTFS

1 Windows Vista, like all operating systems, uses a file system to determine how files are named, stored, and organized on all physical disks. A file system manages files and folders, as well as the information needed to locate and access this data. The file system used in Windows Vista is called NTFS.

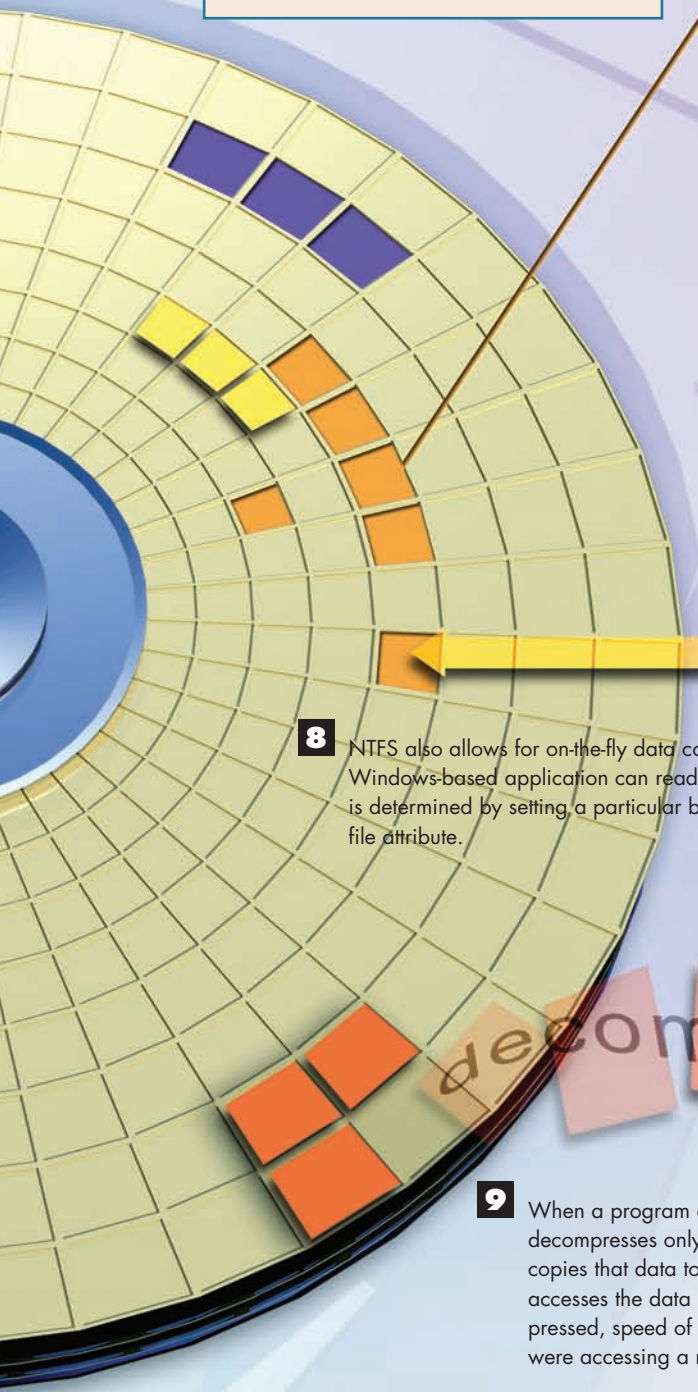
2 A hard drive formatted with NTFS is divided into several sectors, the first of which is the *boot sector*. The boot sector stores information about the layout of the disk and the file system structures, and also contains the boot code that launches the Windows operating system on startup.

3 Also included in each partition is the Master File Table (MFT), which is a type of database that contains all the information necessary to locate and retrieve files from the hard disk.



4 Each file is stored on your hard disk in one or more *clusters* of data. With NTFS, a cluster can range in size from 512 bytes to 64 kilobytes (KB), depending on the total size of your hard drive. For example, on a 2GB drive, the default cluster size is 2KB; on a 200GB drive, the default cluster size is 4KB. Clusters cannot be subdivided; even if the file is smaller than 4KB, it still takes up an entire cluster.

NTFS
NTFS originated with the Windows NT operating system, and stands for *NT File System*. Previous versions of Windows used either the FAT or FAT32 file systems, which utilized larger cluster sizes.



5 Large files are broken into multiple clusters. Although Windows tries to find contiguous storage space that will hold all the clusters for a file, those clusters may end up scattered in different physical locations on the hard disk.

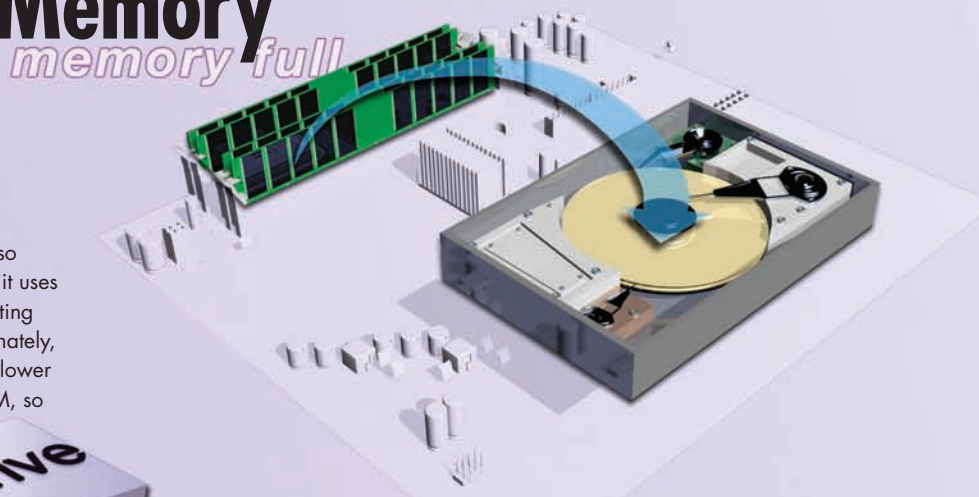
6 When a file is stored on your hard disk, a record of that file—and the location of all its clusters—is created in the Master File Table.

7 When it needs to access a file (to open it within an application, copy it, move it, and so on), Windows accesses the MFT to locate all the clusters associated with that file.

8 NTFS also allows for on-the-fly data compression. Because this compression is implemented within the file system, any Windows-based application can read and write compressed files just as it would noncompressed files. Compression is determined by setting a particular bit within the file header; information about the compression is stored in the data file attribute.

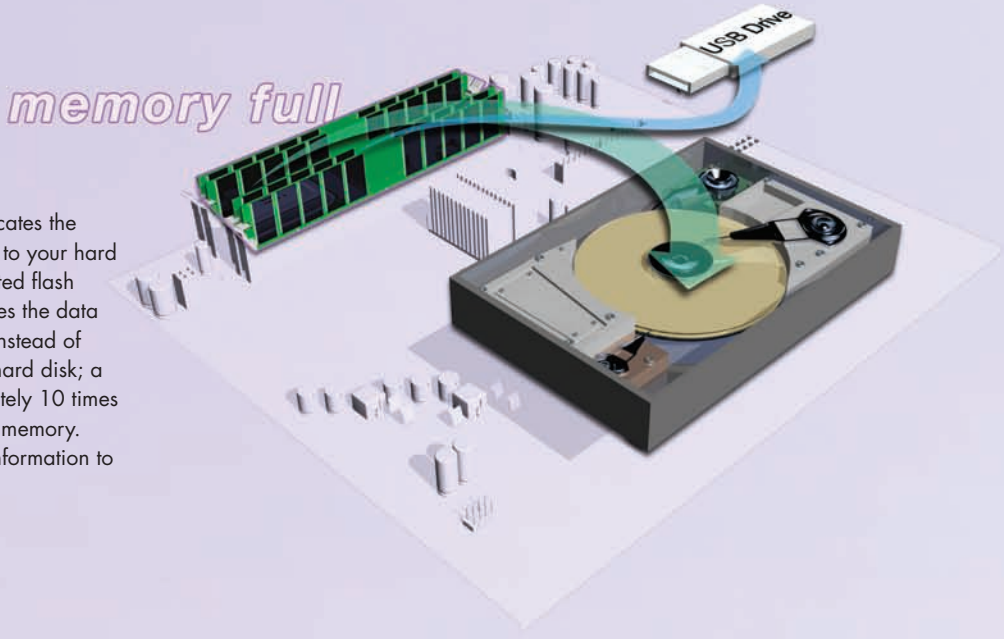
9 When a program opens a compressed file, NTFS automatically decompresses only the portion of the file being read, and then copies that data to system memory. Because the application only accesses the data in memory, which has already been decompressed, speed of data access is just as fast as if the program were accessing a noncompressed file.

How Windows ReadyBoost Adds Instant Memory to Your PC



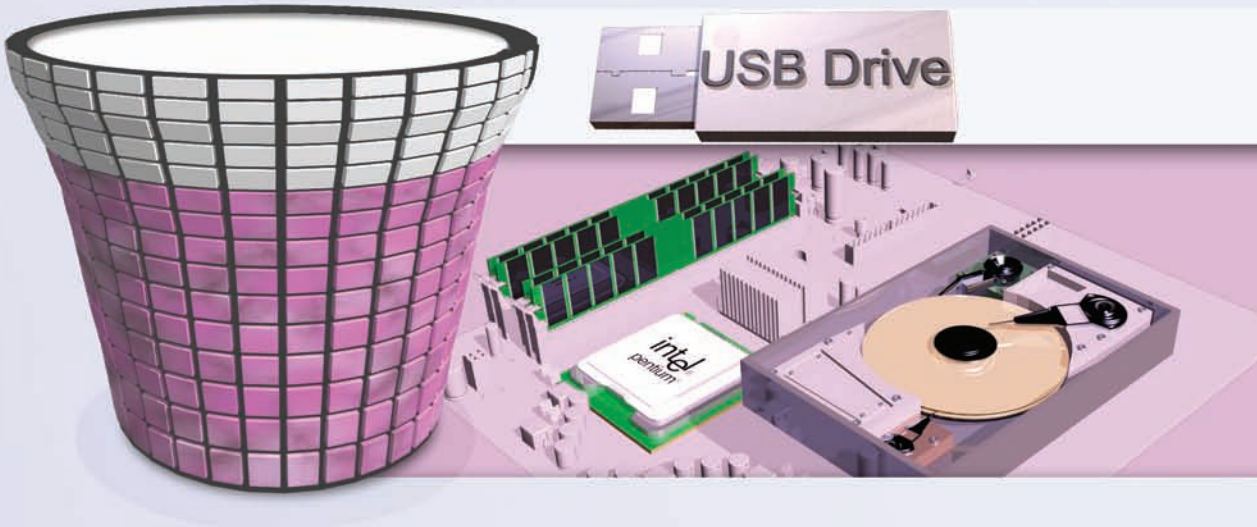
1 When Windows runs short on memory (also called Random Access Memory, or RAM), it uses your PC's hard disk as virtual memory, writing temporary code to the hard drive. Unfortunately, reading and writing to hard disk is much slower than reading and writing to electronic RAM, so system performance suffers.

2 Windows Vista lets you add an instant memory upgrade to your PC via ReadyBoost technology. With ReadyBoost, you can use a flash memory device to temporarily increase the amount of RAM on your personal computer. Insert one of these devices into the appropriate slot on your PC, and your system's memory is automatically increased—and your system's performance is automatically improved. Vista supports USB flash memory drives, as well as CompactFlash (CF) and Secure Digital (SD) memory cards; it can handle devices that hold between 256MB and 4GB of RAM.

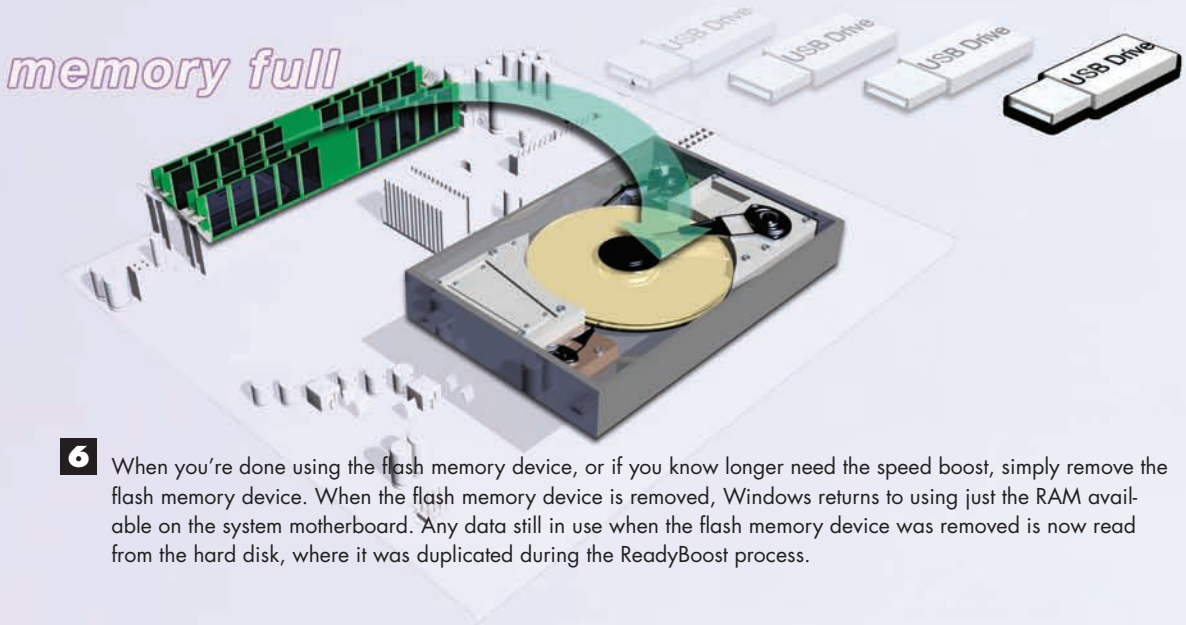


3 With ReadyBoost, Windows duplicates the overflow data that is typically sent to your hard disk by also sending it to the inserted flash memory device. Windows then uses the data stored in the flash device's RAM, instead of accessing the data on the slower hard disk; a flash memory device is approximately 10 times faster than hard disk-based virtual memory. (Windows continues to write the information to the hard disk, as a backup.)

4 The RAM on the flash memory device is added to the available RAM on your computer's motherboard, thus providing more memory to run applications and open documents.



5 You can configure Vista to use all or just part of the available memory on the USB drive for your system's RAM. Just right-click on the USB drive in Windows Explorer and select Properties; from the Properties dialog box, select the ReadyBoost tab and adjust the slider to select how much space to use.



6 When you're done using the flash memory device, or if you know longer need the speed boost, simply remove the flash memory device. When the flash memory device is removed, Windows returns to using just the RAM available on the system motherboard. Any data still in use when the flash memory device was removed is now read from the hard disk, where it was duplicated during the ReadyBoost process.