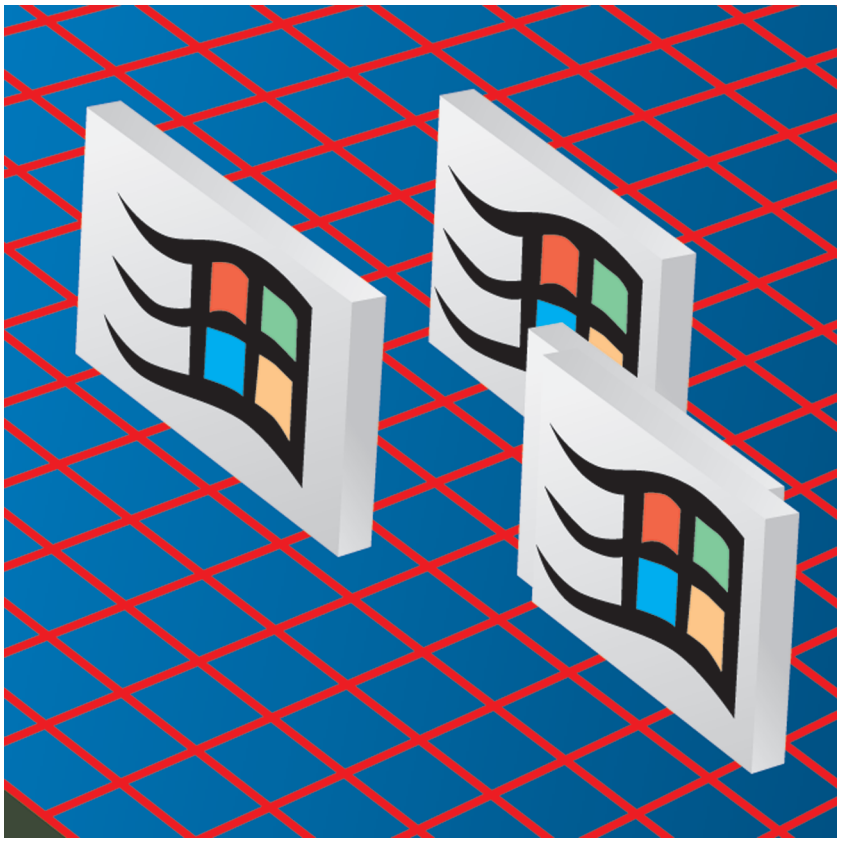
CHAPTER 7

How Windows Works



THE Windows operating system is more than just a pretty face. Sure, it's cute, with all the tiny pictures called icons, sound effects, your personal color scheme, and the capability to drag things around like pull-toys.

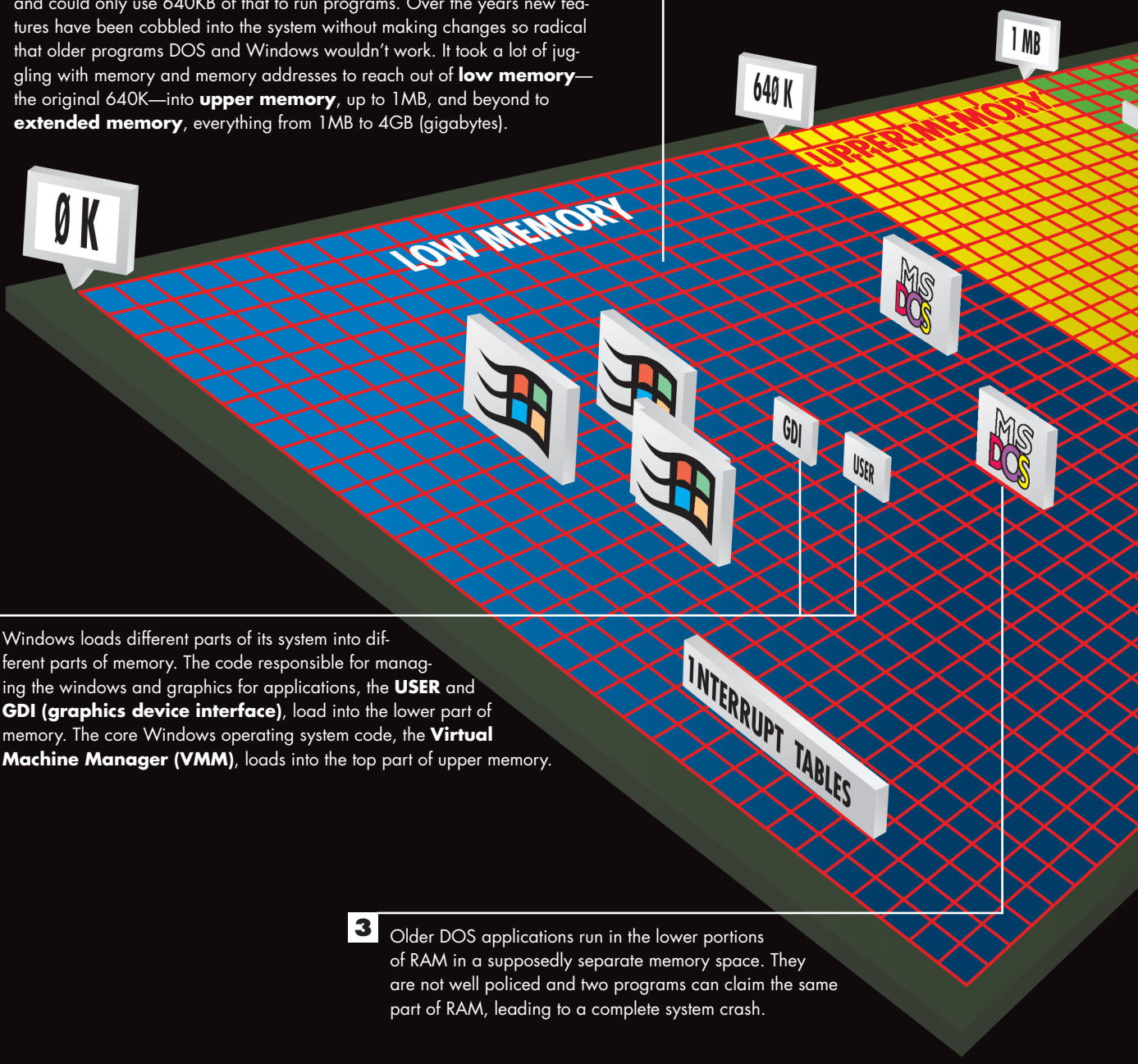
But behind that face is a stern taskmaster. Windows XP, Microsoft's advanced version of Windows, is not your father's operating system. Earlier versions of the consumer level of Windows were built on top of DOS, an elementary operating system for small computers and modest programs that came with the first PCs. DOS was inept at graphics, color, and sound, features that are commonplace in Windows.

More importantly, DOS was intended to run only one program at a time. There was no competition by different programs for the attention of the processor, memory, and drive storage. Computers back then might not have had much memory, but any program loaded into that memory had the run of the place. Now, these programs were expected to play nice and share such toys as RAM and the CPU. They didn't, of course. And when programs trampled on another program's RAM, Windows didn't know what to do and collapsed into the dreaded Blue Screen of Death.

With XP, Windows finally got smart—and sneaky. As you'll discover in this chapter, Windows became enormously more stable when it used pretend computers so that each program thought it was the only program running on the PC. You'll also see how Windows is more than one program. It's a complex, intertwined *organization* of scores of programs, more like a hive than a single worker.

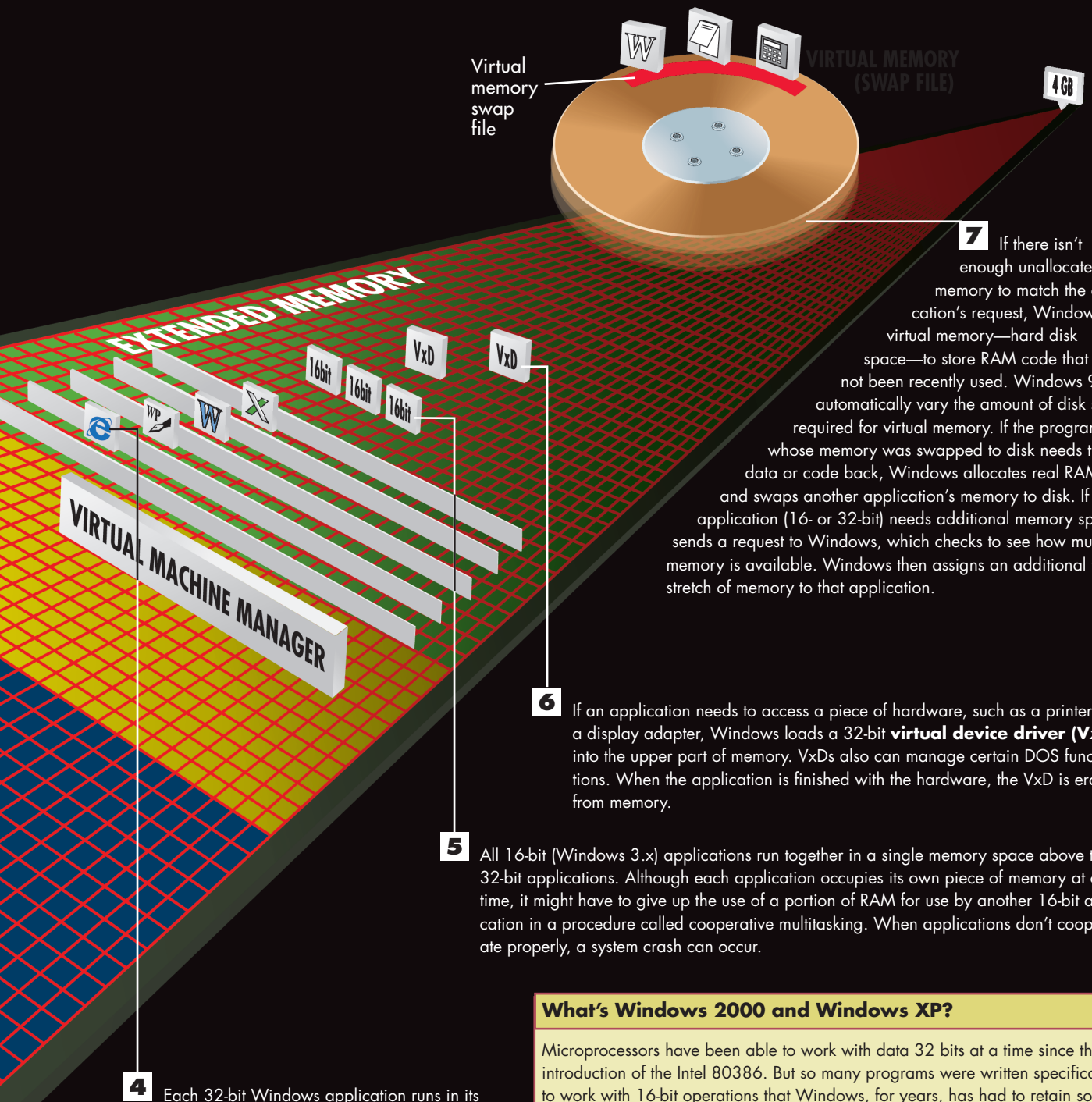
How Older Windows Uses a PC's Memory

1 When you boot your PC with Windows 98 or 95, your computer is using memory in a way developed when PCs had at most a gigabyte of memory, and could only use 640KB of that to run programs. Over the years new features have been cobbled into the system without making changes so radical that older programs DOS and Windows wouldn't work. It took a lot of juggling with memory and memory addresses to reach out of **low memory**—the original 640K—into **upper memory**, up to 1MB, and beyond to **extended memory**, everything from 1MB to 4GB (gigabytes).



2 Windows loads different parts of its system into different parts of memory. The code responsible for managing the windows and graphics for applications, the **USER** and **GDI (graphics device interface)**, load into the lower part of memory. The core Windows operating system code, the **Virtual Machine Manager (VMM)**, loads into the top part of upper memory.

3 Older DOS applications run in the lower portions of RAM in a supposedly separate memory space. They are not well policed and two programs can claim the same part of RAM, leading to a complete system crash.



7 If there isn't enough unallocated memory to match the application's request, Windows uses virtual memory—hard disk space—to store RAM code that has not been recently used. Windows 95 can automatically vary the amount of disk space required for virtual memory. If the program whose memory was swapped to disk needs that data or code back, Windows allocates real RAM for it and swaps another application's memory to disk. If any application (16- or 32-bit) needs additional memory space, it sends a request to Windows, which checks to see how much memory is available. Windows then assigns an additional free stretch of memory to that application.

6 If an application needs to access a piece of hardware, such as a printer or a display adapter, Windows loads a 32-bit **virtual device driver (VxD)** into the upper part of memory. VxDs also can manage certain DOS functions. When the application is finished with the hardware, the VxD is erased from memory.

5 All 16-bit (Windows 3.x) applications run together in a single memory space above the 32-bit applications. Although each application occupies its own piece of memory at one time, it might have to give up the use of a portion of RAM for use by another 16-bit application in a procedure called cooperative multitasking. When applications don't cooperate properly, a system crash can occur.

4 Each 32-bit Windows application runs in its own protected memory space above the system and DOS code. Each application is guaranteed the use of what is supposed to be its own block of memory.

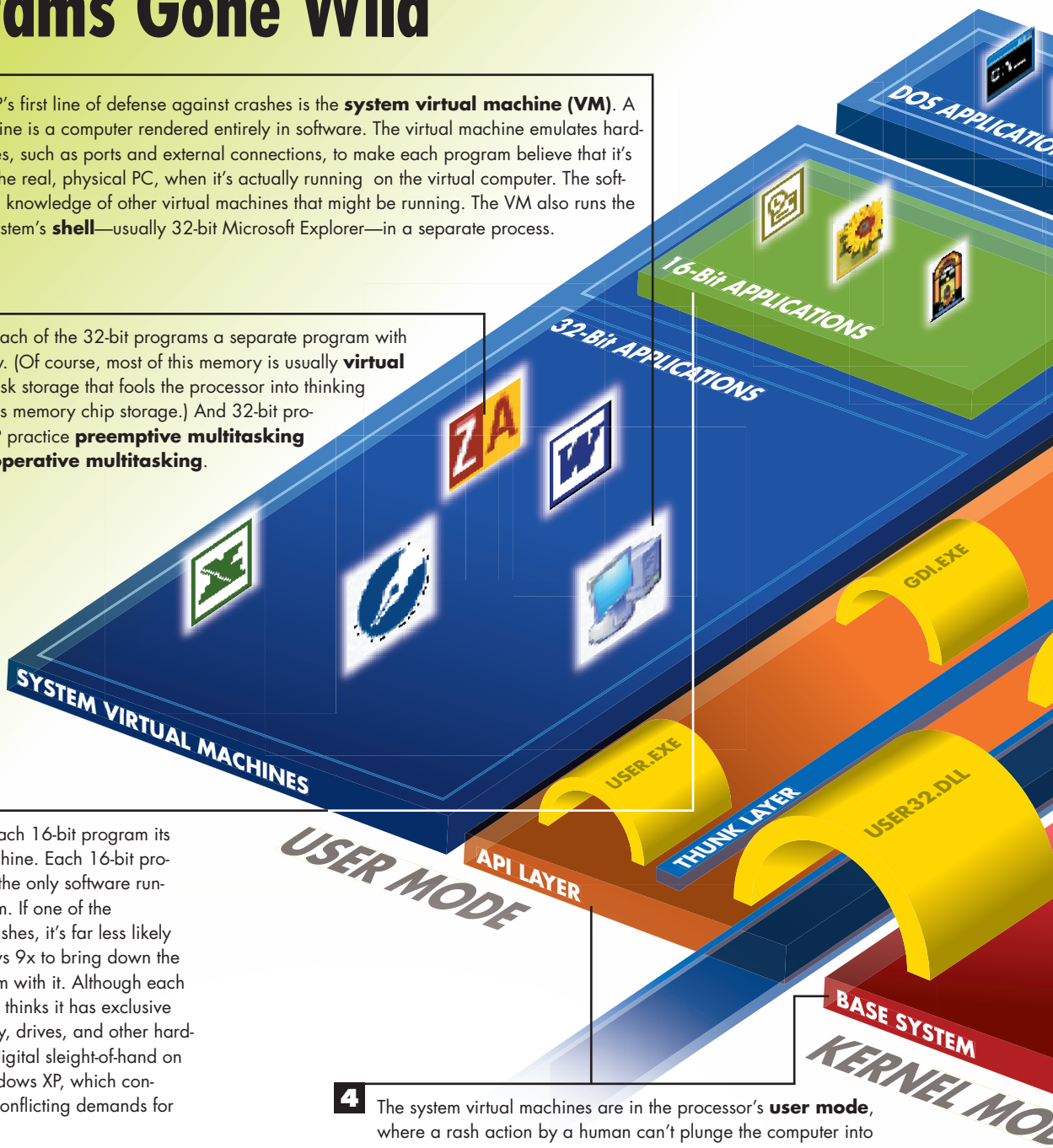
What's Windows 2000 and Windows XP?

Microprocessors have been able to work with data 32 bits at a time since the introduction of the Intel 80386. But so many programs were written specifically to work with 16-bit operations that Windows, for years, has had to retain some of that 16-bit code for compatibility with older, "legacy" applications. Windows 2000, used primarily for networking applications, is pure 32-bit code, which gives it many advantages in speed, reliability, and versatility. But it falters at running older 16-bit Windows programs. The newest version of Windows—XP—is also 32-bit but features a compatibility mode to accomodate older software that needs 16-bit functions.

How Windows Controls Programs Gone Wild

1 Windows XP's first line of defense against crashes is the **system virtual machine (VM)**. A virtual machine is a computer rendered entirely in software. The virtual machine emulates hardware features, such as ports and external connections, to make each program believe that it's running on the real, physical PC, when it's actually running on the virtual computer. The software has no knowledge of other virtual machines that might be running. The VM also runs the operating system's **shell**—usually 32-bit Microsoft Explorer—in a separate process.

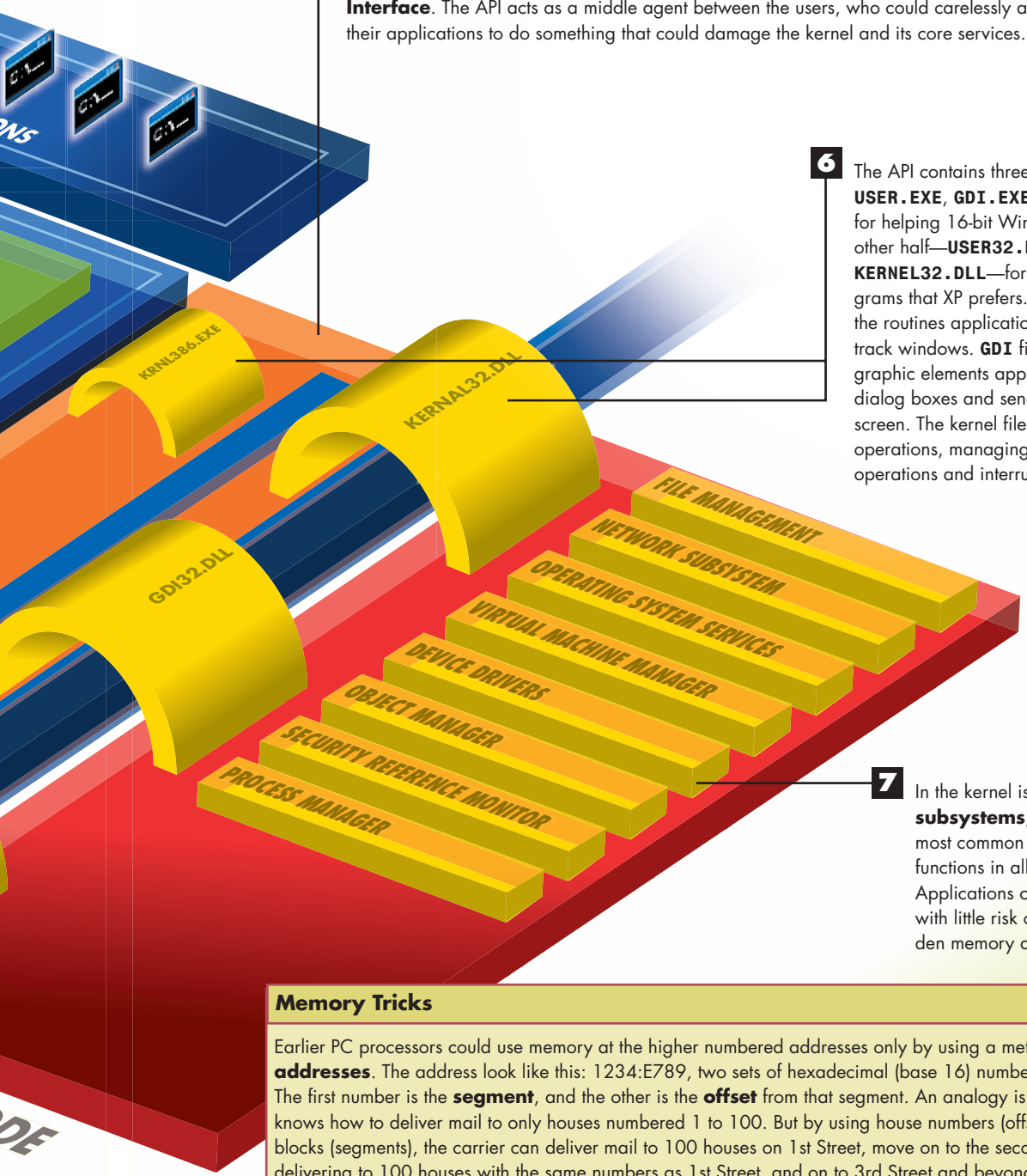
2 The VM gives each of the 32-bit programs a separate program with 4GB of memory. (Of course, most of this memory is usually **virtual RAM**—drive disk storage that fools the processor into thinking the disk space is memory chip storage.) And 32-bit programs under XP practice **preemptive multitasking** rather than **cooperative multitasking**.



3 The VM gives each 16-bit program its own virtual machine. Each 16-bit program thinks it's the only software running on a system. If one of the applications crashes, it's far less likely than in Windows 9x to bring down the rest of the system with it. Although each of the programs thinks it has exclusive rights to memory, drives, and other hardware, it's only digital sleight-of-hand on the part of Windows XP, which constantly juggles conflicting demands for resources.

4 The system virtual machines are in the processor's **user mode**, where a rash action by a human can't plunge the computer into disaster. Dangerous commands are stored in the **kernel mode** in a rich set of instructions called the **kernel**, or **base system**. The kernel mode is off-limits to applications. To call on any 16-bit or 32-bit operations, apps must send requests through the API.

5 The applications in their virtual machines are a rowdy bunch, digitally speaking. When any of the applications wants to call on any of Windows XP's core services, such as writing a file, the app must first go through the **API**—the **Application Programming Interface**. The API acts as a middle agent between the users, who could carelessly ask their applications to do something that could damage the kernel and its core services.



6 The API contains three paired programs, half—**USER.EXE**, **GDI.EXE**, and **KRNL386.EXE**—for helping 16-bit Windows programs, and the other half—**USER32.DLL, GDI32.DLL**, and **KERNEL32.DLL**—for the full-blown 32-bit programs that XP prefers. The **USER** files contain the routines applications need to control and track windows. **GDI** files are collections of graphic elements applications use to build their dialog boxes and send information to the screen. The kernel files work with low-level operations, managing memory, input/output operations and interrupts.

7 In the kernel is a set of **services**, or **subsystems**, code that powers the most common and most necessary functions in all Windows programs. Applications can use these services with little risk of treading on forbidden memory addresses.

Memory Tricks

Earlier PC processors could use memory at the higher numbered addresses only by using a method called **segmented addresses**. The address look like this: 1234:E789, two sets of hexadecimal (base 16) numbers with four digits each. The first number is the **segment**, and the other is the **offset** from that segment. An analogy is a postal carrier who knows how to deliver mail to only houses numbered 1 to 100. But by using house numbers (offsets) combined with blocks (segments), the carrier can deliver mail to 100 houses on 1st Street, move on to the second segment, 2nd Street, delivering to 100 houses with the same numbers as 1st Street, and on to 3rd Street and beyond. Reading and writing memory was slower because the processor had to take an extra step to work with the segment and the offset.

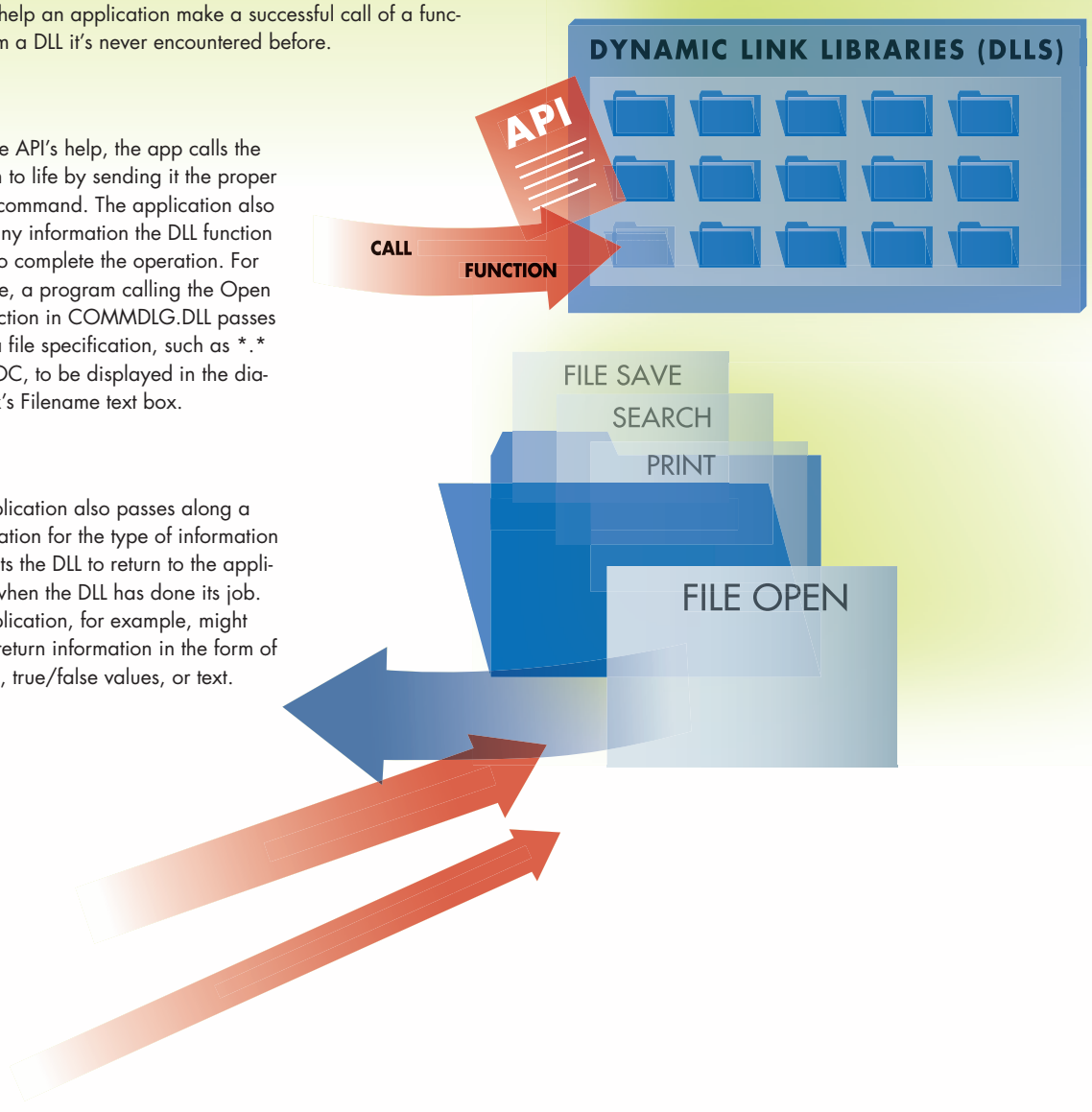
How Windows Shares Program Code

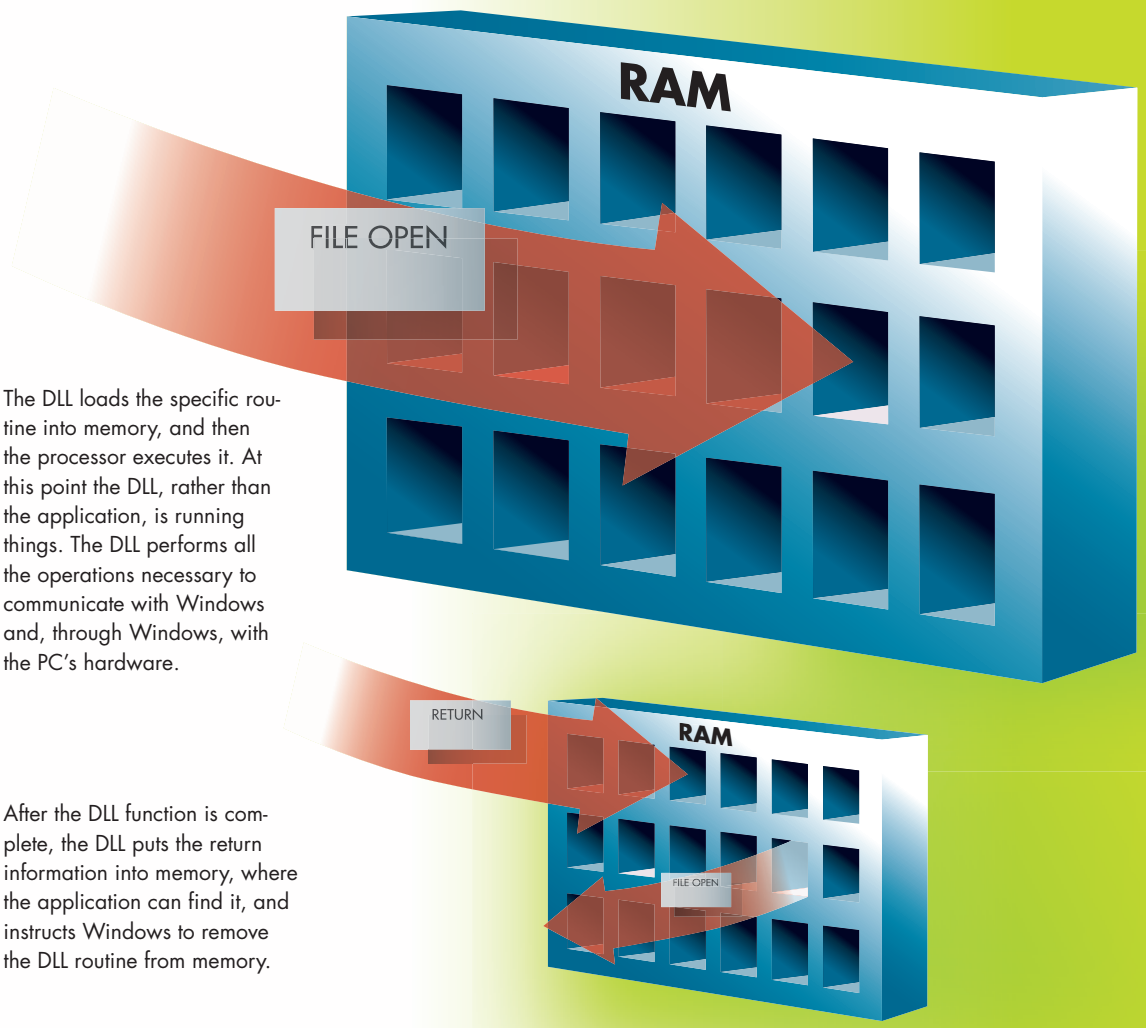
1 Windows 9x and Windows XP both provide several files called **DLLs (dynamic link libraries)**. These are collections of software code that perform common software functions. One of the most frequently used DLLs is Windows' COMMDLG.DLL. As the name suggests, the DLL specializes in commonly used dialog boxes. Its functions include displaying File Open, File Save, Search, and Print dialog boxes. Microsoft is not the only creator of DLLs for Windows. Other companies have created DLLs to provide functions such as file compression or added printing abilities.

2 An application that wants to take advantage of a DLL function first checks with an **API (application programming interface)** to find out how to call the function. All DLLs have APIs to help an application make a successful call of a function from a DLL it's never encountered before.

3 With the API's help, the app calls the function to life by sending it the proper digital command. The application also sends any information the DLL function needs to complete the operation. For example, a program calling the Open File function in COMMDLG.DLL passes along a file specification, such as *.* or *.DOC, to be displayed in the dialog box's Filename text box.

4 The application also passes along a specification for the type of information it expects the DLL to return to the application when the DLL has done its job. The application, for example, might expect return information in the form of integers, true/false values, or text.



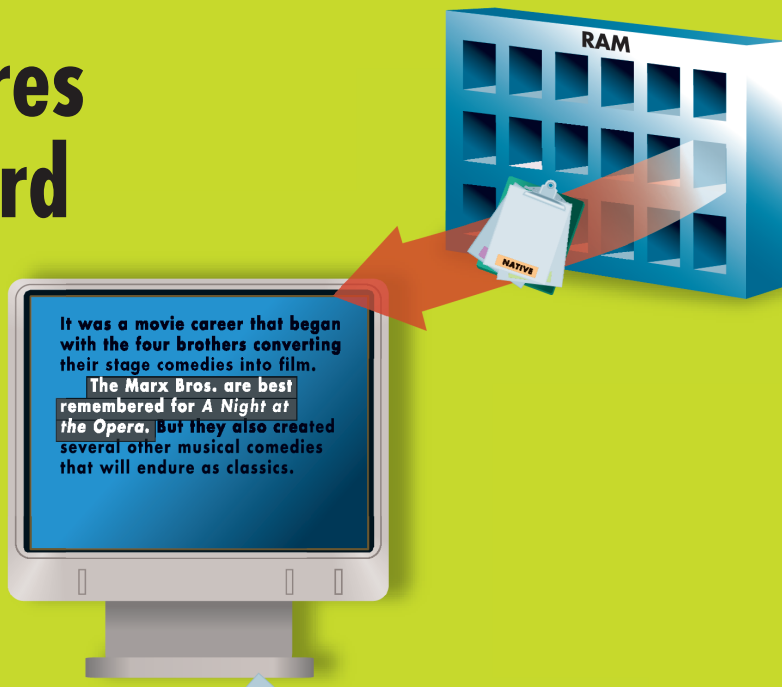


5 The DLL loads the specific routine into memory, and then the processor executes it. At this point the DLL, rather than the application, is running things. The DLL performs all the operations necessary to communicate with Windows and, through Windows, with the PC's hardware.

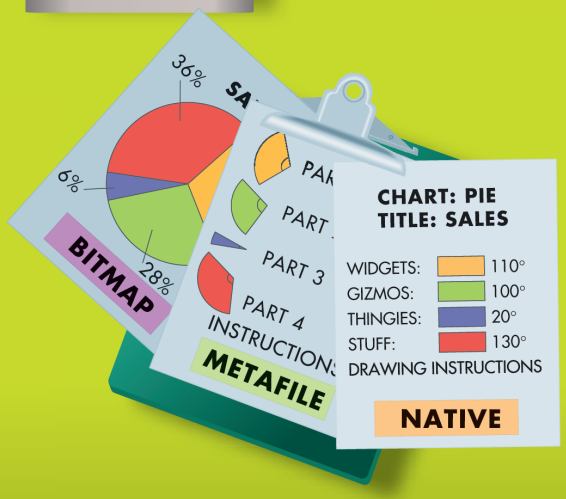
6 After the DLL function is complete, the DLL puts the return information into memory, where the application can find it, and instructs Windows to remove the DLL routine from memory.

7 The application inspects the return information, which usually tells whether the DLL function was able to execute correctly. If the operation was a success, the application continues from where it left off before issuing the function call. If the operation failed, the application displays an error message.

How Windows Shares Data on the Clipboard

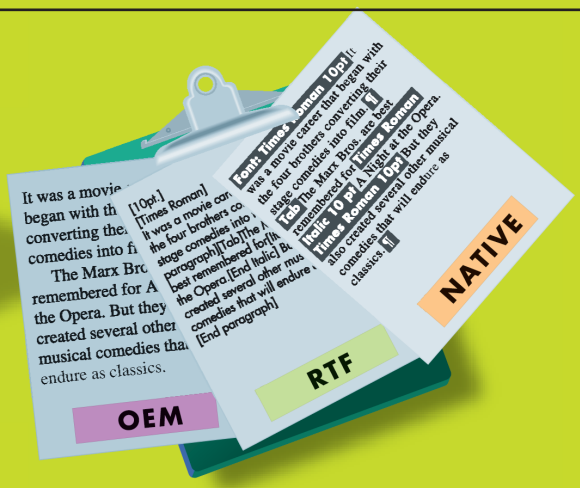


1 The simplest way to share the same data among different documents and different applications is through the Windows Clipboard. Any time you select some data—text, graphics, spreadsheet cells—and copy it, Windows places a replica of that data into a section of memory reserved for its Clipboard. Actually, Windows creates three different versions of the data.



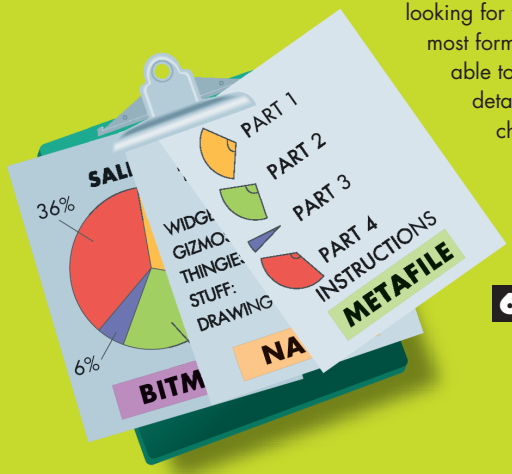
2 Applications save their documents in different **formats**—the exact coding that defines how the data is structured. When you copy or cut data, Windows transfers the selection to the Clipboard in multiple formats so that it then can be pasted into applications that use different formats. One format is that of the application that created the data. The second is a translation of the application's formatting codes for boldfacing, justification, fonts, and so on into a generic form called **rich text format (RTF)**, which is recognized by all Windows applications. The third format is called **OEM (original equipment manufacturer)** text, which is used to paste text into DOS applications or when you prefer no formatting.

3 For example, if the data is a graphic, Windows saves it in three formats—the original format, such as .TIF or .PCX; a bitmap format; and a metafile format. A *bitmap* is a record of the specific pattern of display pixels that need to be turned on to re-create the image in its original size. A **metafile** is a collection of commands that can be used by Windows's **graphic device interface (GDI)** to re-create the image. Metafiles are resolution-independent; that is, they aren't locked into a specific array of pixels, as a bitmap is. This lets metafiles take advantage of all the resolution your display or printer can provide, and it lets you resize images without distorting them. (A metafile often is called an **object-oriented** graphic because it is stored as a series of distinct objects—lines, rectangles, arcs—rather than as a map of pixels.)



4 When you paste data from the Clipboard, the application receiving the data inspects the various formats in which the data has been copied. If you are pasting data into the application from which it was copied, the application will choose its native format.

5 If you are pasting from one application into another, the receiving application inspects all the formats saved on the Clipboard. The application is looking for formats it understands, and which one of them retains the most formatting information. For example, a metafile graphic is preferable to a bitmapped graphic, because a metafile contains more detailed information that the receiving application can use to change the graphic's size or placement on a page.



6 To paste data that's in a format other than its native one, the receiving application first translates any information about the format of the data—such as boldfacing or fonts—into the formatting codes the receiving application uses. If it is receiving a metafile graphic, the application sends the commands contained in the Clipboard to Windows's GDI.exe and GDI32.dll, which control the graphic look of Windows XP and its ancestors. They, in turn, send the display driver the information the driver needs to create the graphic onscreen.