## 64-Bit Address Space Layouts

The theoretical 64-bit virtual address space is 16 exabytes (18,446,744,073,709,551,616 bytes, or approximately 18.44 billion billion bytes). Unlike on x86 systems, where the default address space is divided in two parts (half for a process and half for the system), the 64-bit address is divided into a number of different size regions whose components match conceptually the portions of user, system, and session space. The various sizes of these regions, listed in Table 10-8, represent current implementation limits that could easily be extended in future releases. Clearly, 64 bits provides a tremendous leap in terms of address space sizes.

**TABLE 10-8** 64-Bit Address Space Sizes

| Region | IA64 | x64 |
|---|---|---|
| Process Address Space | 7,152 GB | 8,192 GB |
| System PTE Space | 128 GB | 128 GB |
| System Cache | 1 TB | 1 TB |
| Paged Pool | 128 GB | 128 GB |
| Nonpaged Pool | 75% of physical memory | 75% of physical memory |

Also, on 64-bit Windows, another useful feature of having an image that is large address space aware is that while running on 64-bit Windows (under Wow64), such an image will actually receive all 4 GB of user address space available—after all, if the image can support 3-GB pointers, 4-GB pointers should not be any different, because unlike the switch from 2 GB to 3 GB, there are no additional bits involved. Figure 10-11 shows TestLimit, running as a 32-bit application, reserving address space on a 64-bit Windows machine, followed by the 64-bit version of TestLimit leaking memory on the same machine.

```
C:\temp>testlimit -r

Testlimit v5.1 - test Windows limits
Copyright (C) 2012 Mark Russinovich
Sysinternals - wwww.sysinternals.com

Reserving private bytes (MB)...
Leaked 4031 MB of reserved memory (4031 MB total leaked). Lasterror: 8
Not enough storage is available to process this command.
```

```
C:\temp>testlimit64 -r

Testlimit v5.1 - test Windows limits
Copyright (C) 2012 Mark Russinovich
Sysinternals - wwww.sysinternals.com

Reserving private bytes (MB)...
Leaked 8388548 MB of reserved memory (8388548 MB total leaked). Lasterror: 8
Not enough storage is available to process this command.
```

**FIGURE 10-11** 32-bit and 64-bit TestLimit reserving address space on a 64-bit Windows computer

Note that these results depend on the two versions of TestLimit having been linked with the /LARGEADDRESSAWARE option. Had they not been, the results would have been about 2 GB for each. 64-bit applications linked without /LARGEADDRESSAWARE are constrained to the first 2 GB of the process virtual address space, just like 32-bit applications.

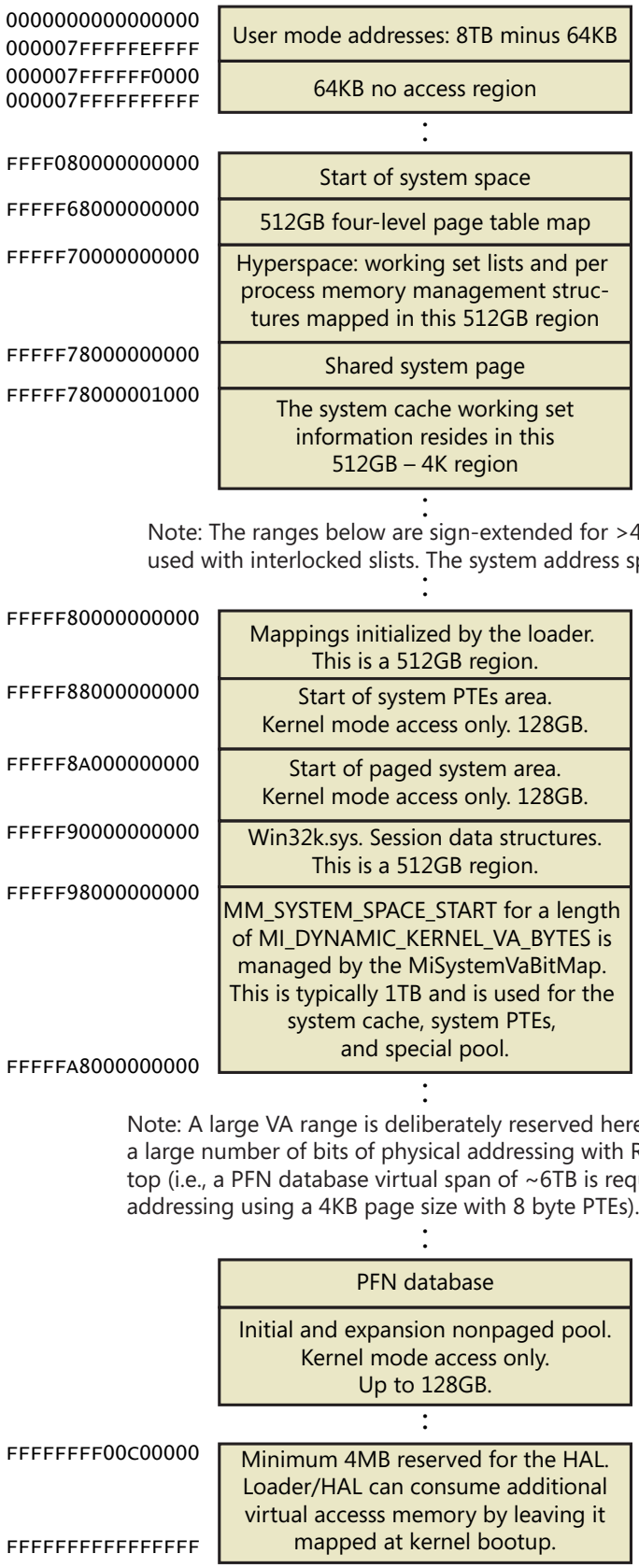


**FIGURE 10-13** x64 address space layout

## x64 Virtual Addressing Limitations

As discussed previously, 64 bits of virtual address space allow for a possible maximum of 16 exabytes (EB) of virtual memory, a notable improvement over the 4 GB offered by 32-bit addressing. With such a copious amount of memory, it is obvious that today's computers, as well as tomorrow's foreseeable machines, are not even close to requiring support for that much memory.

Accordingly, to simplify chip architecture and avoid unnecessary overhead, particularly in address translation (to be described later), AMD's and Intel's current x64 processors implement only 256 TB of virtual address space. That is, only the low-order 48 bits of a 64-bit virtual address are implemented. However, virtual addresses are still 64 bits wide, occupying 8 bytes in registers or when stored in memory. The high-order 16 bits (bits 48 through 63) must be set to the same value as the highest order implemented bit (bit 47), in a manner similar to sign extension in two's complement arithmetic. An address that conforms to this rule is said to be a "canonical" address.

Under these rules, the bottom half of the address space thus starts at 0x0000000000000000, as expected, but it ends at 0x00007FFFFFFFFFFF. The top half of the address space starts at 0xFFFF800000000000 and ends at 0xFFFFFFFFFFFFFFFF. Each "canonical" portion is 128 TB. As newer processors implement more of the address bits, the lower half of memory will expand upward, toward 0x7FFFFFFFFFFFFFFF, while the upper half of memory will expand downward, toward 0x8000000000000000 (a similar split to today's memory space but with 32 more bits).

### Windows x64 16-TB Limitation

Windows on x64 has a further limitation: of the 256 TB of virtual address space available on x64 processors, Windows at present allows only the use of a little more than 16 TB. This is split into two 8-TB regions, the user mode, per-process region starting at 0 and working toward higher addresses (ending at 0x000007FFFFFFFFFF), and a kernel-mode, systemwide region starting at "all Fs" and working toward lower addresses, ending at 0xFFFFF80000000000 for most purposes. This section describes the origin of this 16-TB limit.

A number of Windows mechanisms have made, and continue to make, assumptions about usable bits in addresses. Pushlocks, fast references, Patchguard DPC contexts, and singly linked lists are common examples of data structures that use bits within a pointer for nonaddressing purposes. Singly linked lists, combined with the lack of a CPU instruction in the original x64 CPUs required to "port" the data structure to 64-bit Windows, are responsible for this memory addressing limit on Windows for x64.

Here is the SLIST_HEADER, the data structure Windows uses to represent an entry inside a list:

```
typedef union _SLIST_HEADER {
    ULONGLONG Alignment;
    struct {
        SLIST_ENTRY Next;
        USHORT Depth;
        USHORT Sequence;
    } DUMMYSTRUCTNAME;
} SLIST_HEADER, *PSLIST_HEADER;
```