memory can be stored on the hard drive without incurring a performance penalty. When those instructions are ready to be executed, the OS kernel will have to transfer the data into physical memory. This slows the system down but makes it more flexible without requiring huge quantities of DRAM. Part of the kernel's memory management function is to decide which virtual pages should be held in DRAM and which should be *swapped* out to the disk. Pages that have not been used for a while can be swapped out to make room for new pages that are currently needed. If a process subsequently accesses a page that has been moved to the disk, that page can be swapped back into DRAM to replace another page that is not needed at the time. A computer with 256 MB of DRAM could, for example, have a 512-MB swap file on its hard drive, enabling processes to share a combined 768 MB of used virtual memory.

This scheme of expanding virtual memory onto a disk effectively turns the computer's DRAM into a large cache for an even larger disk-based memory. As with all caches, certain behavioral characteristics exist. A virtual memory page that is not present in DRAM is effectively a cache miss with a large penalty, because hard disks are much slower than DRAM. Such misses are called *page faults*. The MMU detects that the requested virtual memory address from a particular PID is not present in DRAM and causes an exception that must be handled by the OS kernel. Instead of performing a cache line fill and flush, it is the kernel's responsibility to swap pages to and from the disk. For a virtual memory system to function with reasonable performance, the *working set* of memory across all the processes running should be able to fit into the computer's physical memory. The working set includes any instructions and data that are accessed within a local time interval. This is directly analogous to a microprocessor cache's exploitation of locality. Processes with good locality characteristics will do well in a cache and in a virtual memory system. Processes with poor locality may result in thrashing as many sequential page faults are caused by random accesses throughout a large virtual memory space.

The virtual to physical address mapping process is guided by the kernel using a *page table*, which can take various forms but must somehow map each PID/VPN combination to either a physical memory page or one located on the disk drive's swap area. Virtual page mapping is illustrated in Fig. 7.6, assuming 4-kB pages, a 32-bit address space, and an 8-bit PID. In addition to basic mapping information, the page table also contains status information, including a dirty bit that indicates when a page held in memory has been modified. If modified, the page must be saved to the disk before being flushed to make room for a new virtual page. Otherwise, the page can be flushed without further action.

Given a 4-kB page size and a 32-bit address space, each process has access to 2^{20} = 1,048,576 pages. With 256 PIDs, a brute-force page table would contain more than 268 million entries! There are a variety of schemes to reduce page table size, but there is no escaping the fact that a page table

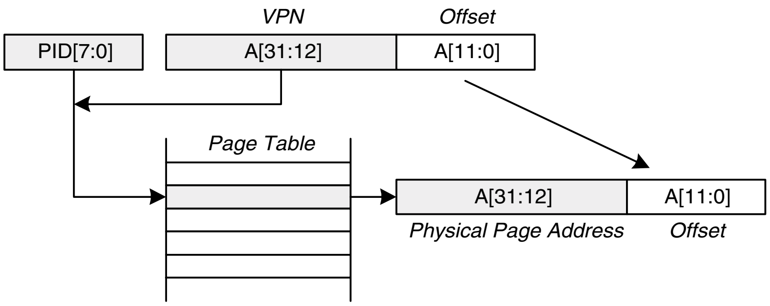

FIGURE 7.6 Virtual page mapping.

will be large. Page table management schemes are largely an issue of OS architecture and are outside the scope of this discussion. The fact that the page table is large and is parsed by software means that the mapping process will be extremely slow without hardware assistance. Every access to virtual memory, in other words almost every access performed on the computer, requires mapping, which makes hardware acceleration critical to the viability of virtual memory.

Within the MMU is a *translation lookaside buffer* (TLB), a small, fully associative cache that allows the MMU to rapidly locate recently accessed virtual page mappings. Typical sizes for a TLB are just 16 to 64 entries because of the complexity of implementing a fast fully associative cache. When a process is first spawned, it has not yet performed virtual memory accesses, so its first access will result in a TLB miss. When a TLB miss occurs, an exception is generated that invokes the kernel's memory management routine to parse the page table in search of the correct physical address mapping. The kernel routine loads a TLB entry with the mapping information and exits. On subsequent memory accesses, the TLB will hit some and miss some. It is hoped that the ratio of hits to misses will decline rapidly as the process executes. Once again, locality of reference is key to a well performing application, but the TLB and MMU are not as sensitive to locality as a normal cache, because they map multiple-kilobyte pages rather than 16 or 32 byte lines. Yet, as more processes actively vie for resources in a multitasking system, they may begin to fight each other for scarce TLB entries. The resources and architecture of a computer must be properly matched to its intended application. A typical desktop or embedded computer may get along fine with a small TLB, because it may not have many demanding processes running concurrently. A more powerful computer designed to simultaneously run many memory-intensive processes may require a larger TLB to take full advantage of its microprocessor and memory resources. The ever-present trade-off between performance and cost does not go away!

The TLB is usually located between the microprocessor and its cache subsystem as shown in Fig. 7.7, such that physical addresses are cached rather than virtual addresses. Such an arrangement adds latency to microprocessor transactions, because the virtual-to-physical mapping must take place before the L1 cache can respond. A TLB can be made very fast because of its small size, thereby limiting its time penalty on transactions. Additionally, microprocessors may implement a pipelined interface where addresses are presented each clock cycle, but their associated data are returned one or more clock cycles later, providing time for the TLB lookup.

7.5 SUPERPIPELINED AND SUPERSCALAR ARCHITECTURES

At any given time, semiconductor process technology presents an intrinsic limitation on how fast a logic gate can switch on and off and at what frequency a flip-flop can run. Other than relying on semiconductor process advances to improve microprocessor and system throughput, certain basic techniques have been devised to extract more processing power from silicon with limited switching

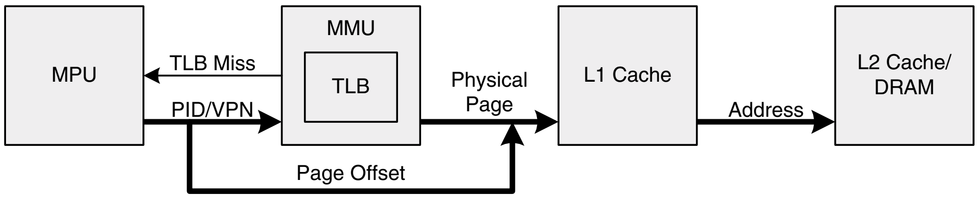

FIGURE 7.7 Location of TLB.