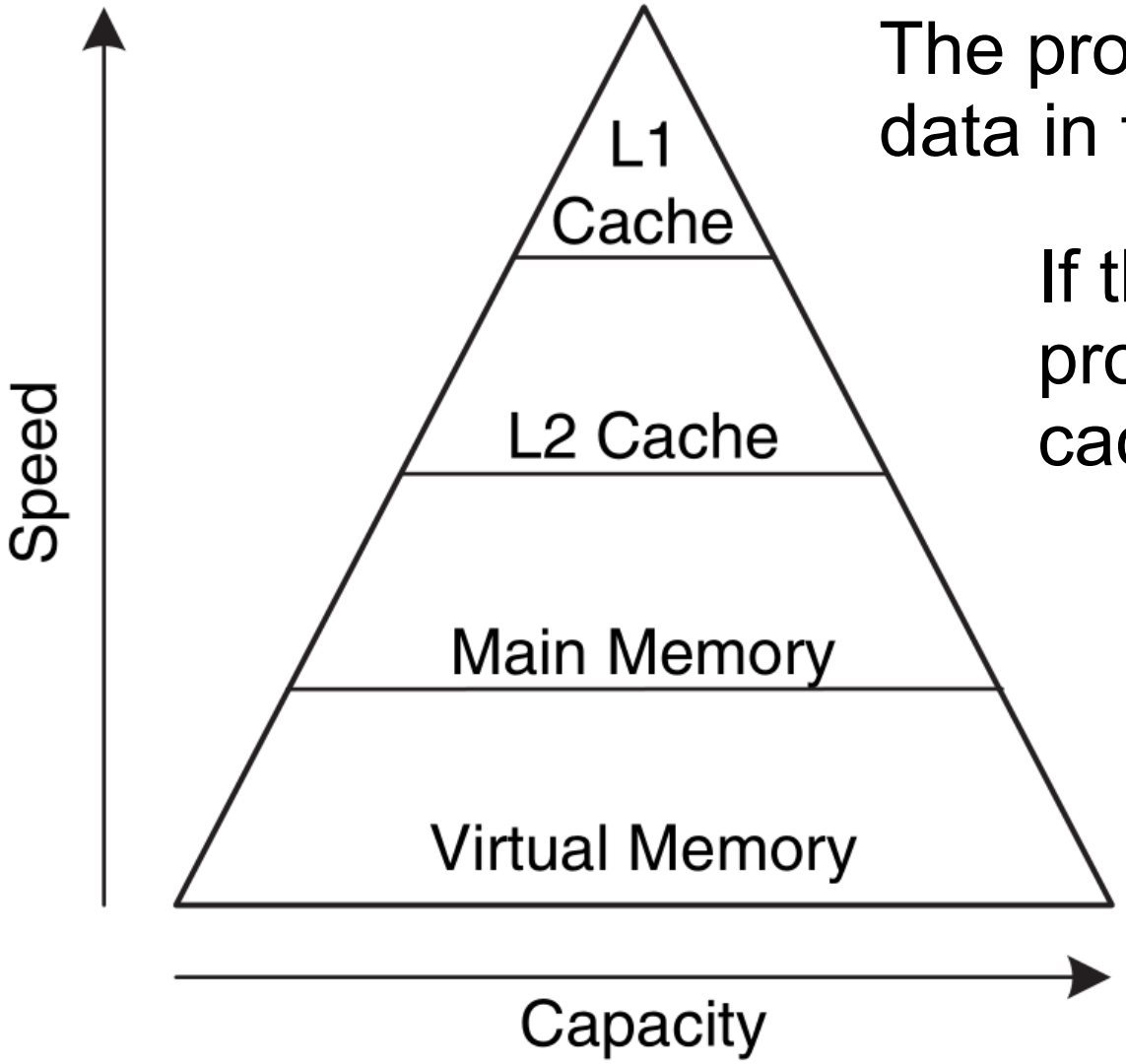
Computer organization and architecture

Lesson 20

Memory systems

Part 2



The processor first looks for the data in the L1 cache.

If the L1 cache misses, the processor looks in the L2 cache.

If the L2 cache misses, the processor fetches the data from main memory.

Many modern systems add even more levels of cache to the memory hierarchy, because accessing main memory is slow.

Example: L1 cache, L2 cache, and main memory, with access times of 1, 10, and 100 cycles, respectively.

Assume that the L1 and L2 caches have miss rates of 5% and 20%, respectively.

What is the average memory access time (AMAT)?

$$AMAT = 1 + 0.05\left(10 + 0.2 \times 100\right) = 2.5\,cycles$$

The misses can be classified as

- compulsory
- capacity
- conflict

The first request to a cache block is called a **compulsory miss**, because the block must be read from memory regardless of the cache design.

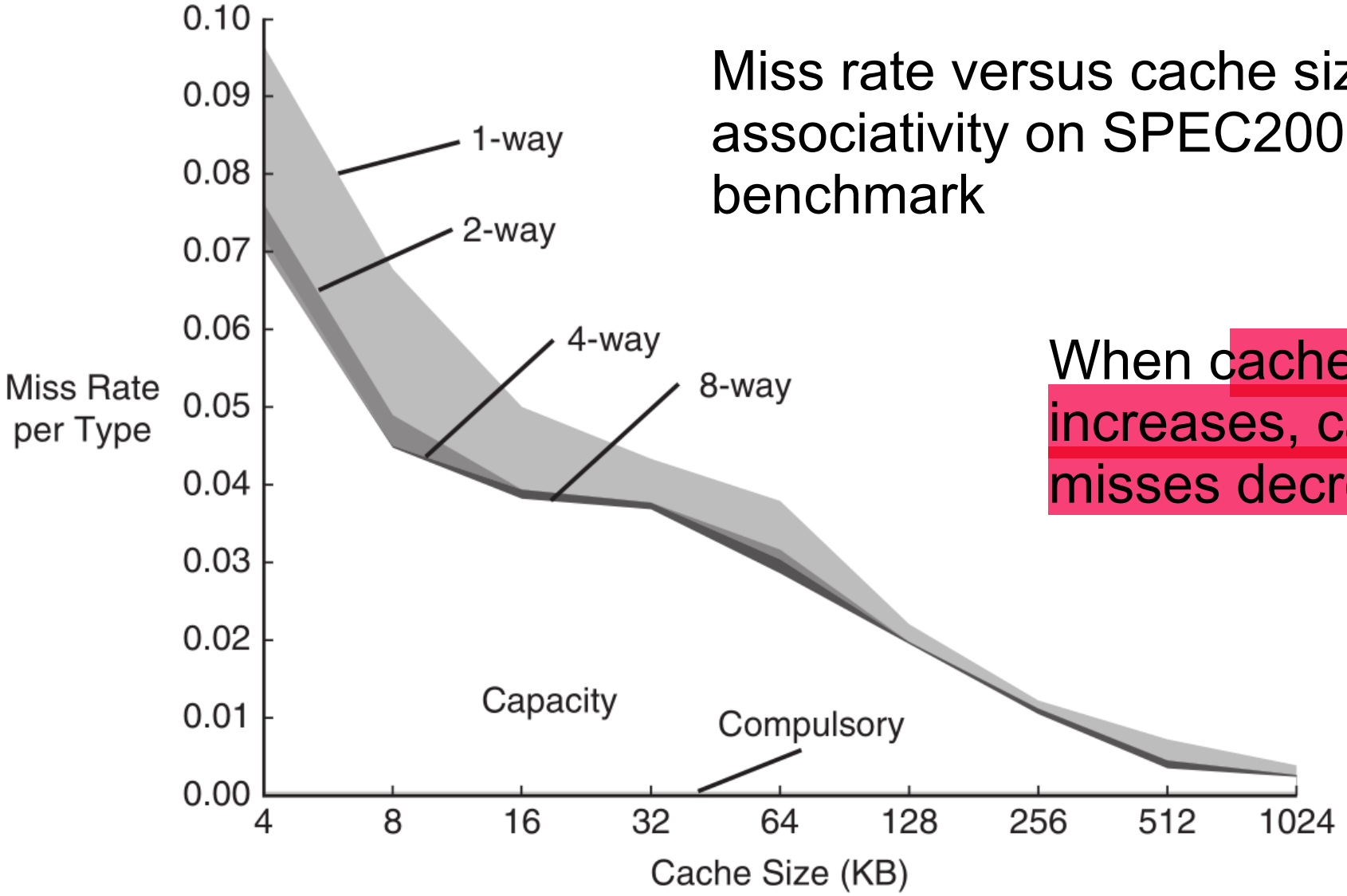
Capacity misses occur when the cache is too small to hold all concurrently used data.

Conflict misses are caused when several addresses map to the same set and evict blocks that are still needed.

Increasing cache capacity can reduce conflict and capacity misses, but it does not affect compulsory misses.

Changing cache parameters can affect one or more type of cache miss.

Increasing block size could reduce compulsory misses (due to spatial locality) but might actually increase conflict misses (because more addresses would map to the same set and could conflict).



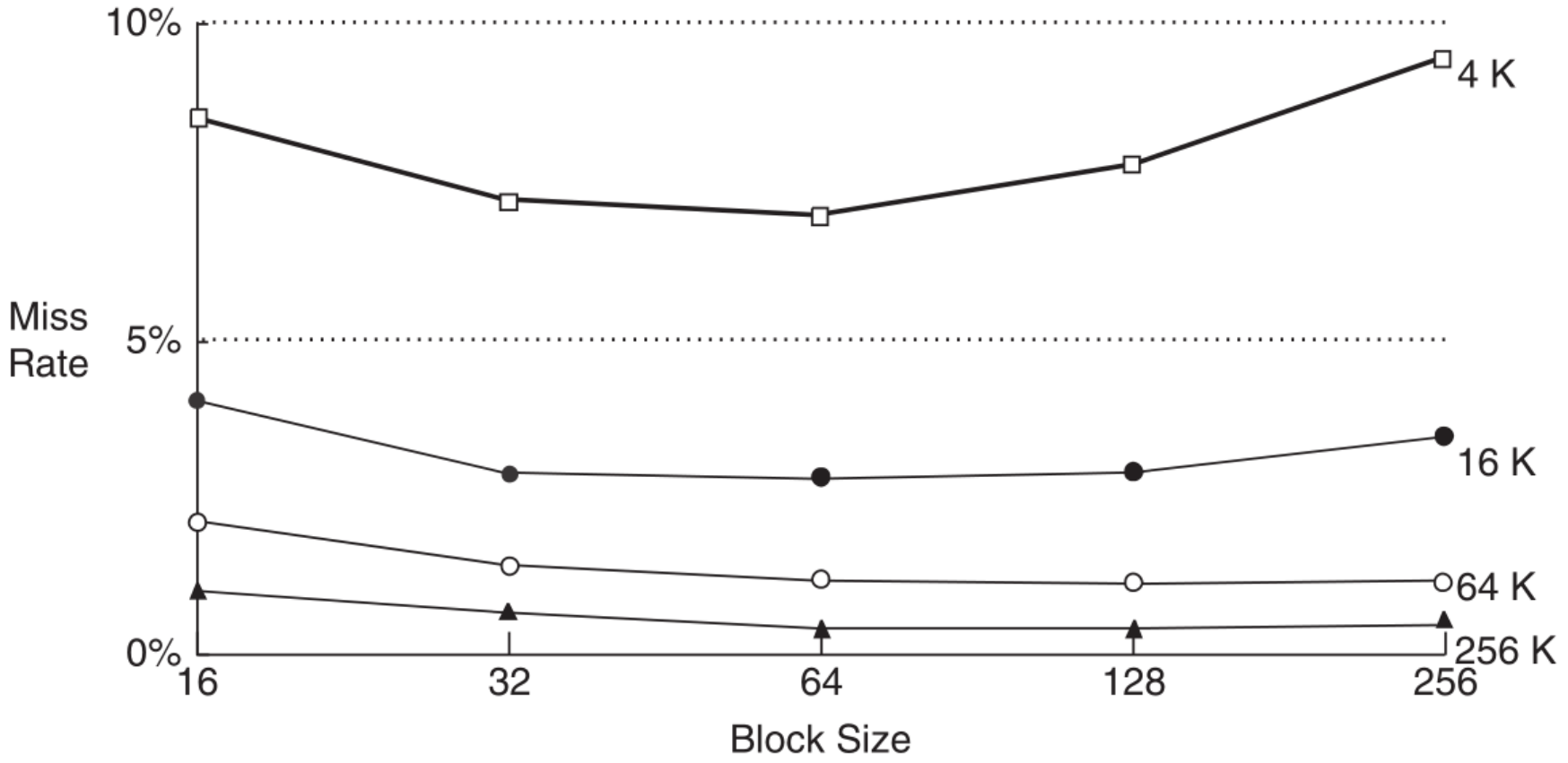
Miss rate versus cache size and associativity on SPEC2000 benchmark

When cache size increases, capacity misses decrease.

Increased associativity, especially for small caches, decreases the number of conflict misses.

Increasing associativity beyond 4 or 8 ways provides only small decreases in miss rate.

Miss rate versus block size and cache size on SPEC92 benchmark



Miss rate can be decreased by using larger block sizes that take advantage of spatial locality.

As block size increases, the number of sets in a fixed-size cache decreases, increasing the probability of conflicts.

For small caches, such as the 4-KB cache, increasing the block size beyond 64 bytes increases the miss rate because of conflicts.

For larger caches, increasing the block size beyond 64 bytes does not change the miss rate.

However, large block sizes might still increase execution time because of the larger miss penalty, the time required to fetch the missing cache block from main memory.

Write policy

Memory stores, or writes, follow a similar procedure as loads.

Upon a memory store, the processor checks the cache.

If the cache misses, the cache block is fetched from main memory into the cache, and then the appropriate word in the cache block is written.

If the cache hits, the word is simply written to the cache block.

In a **write-through cache**, the data written to a cache block is simultaneously written to main memory.

In a **write-back cache**, a **dirty bit** (D) is associated with each cache block.

D is 1 when the cache block has been written and 0 otherwise.

Dirty cache blocks are written back to main memory only when they are evicted from the cache.

A write-through cache requires no dirty bit but usually requires more main memory writes than a write-back cache.

Modern caches are usually write-back, because main memory access time is so large.

Virtual memory

It is using a hard drive space to expand main memory.

Programs can access data anywhere in **virtual memory**, so they must use **virtual addresses** that specify the location in virtual memory.

The physical memory holds a subset of most recently accessed virtual memory.

Physical memory acts as a cache for virtual memory.

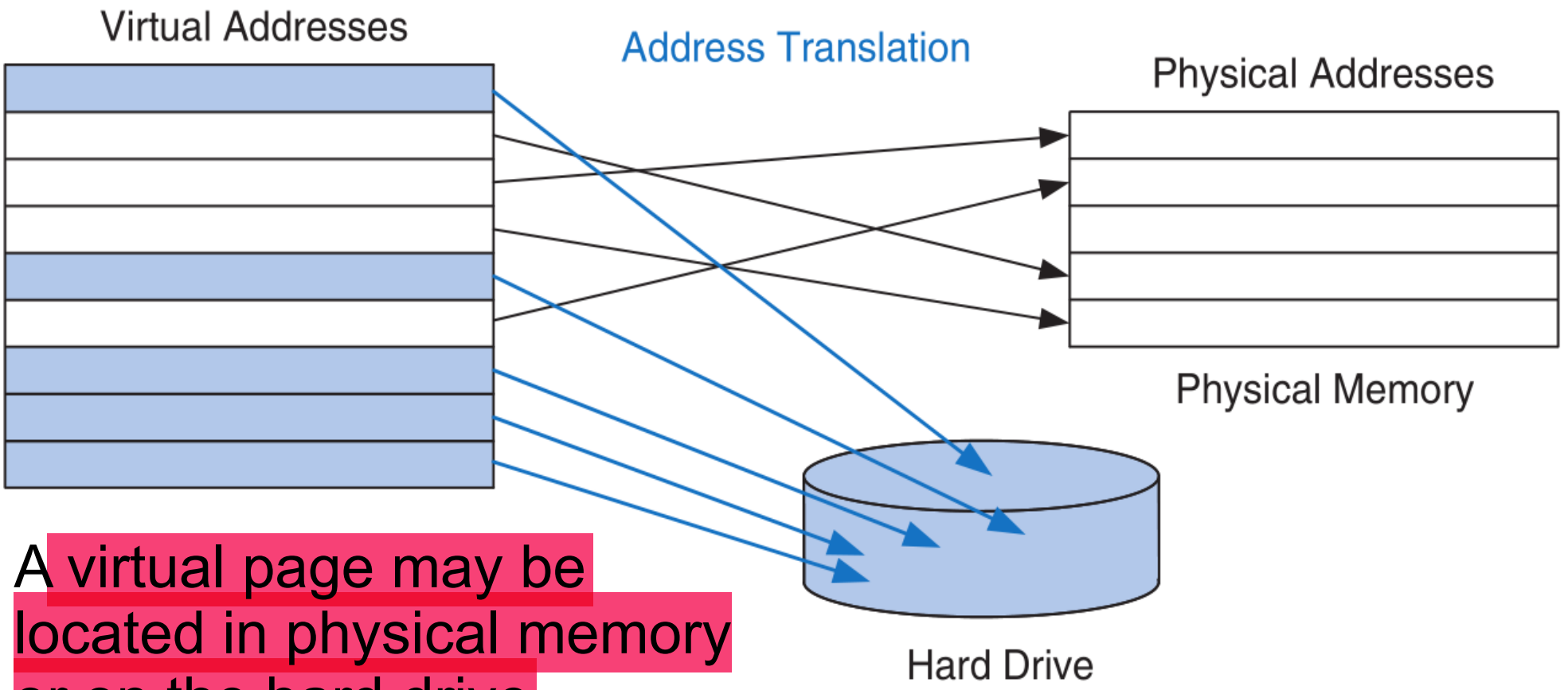
Thus, most accesses hit in physical memory at the speed of DRAM, yet the program enjoys the capacity of the larger virtual memory.

Virtual memory systems use different terminologies for the same caching principles

Cache	Virtual Memory
Block	Page
Block size	Page size
Block offset	Page offset
Miss	Page fault
Tag	Virtual page number

Virtual memory is divided into **virtual pages**, typically 4 KB

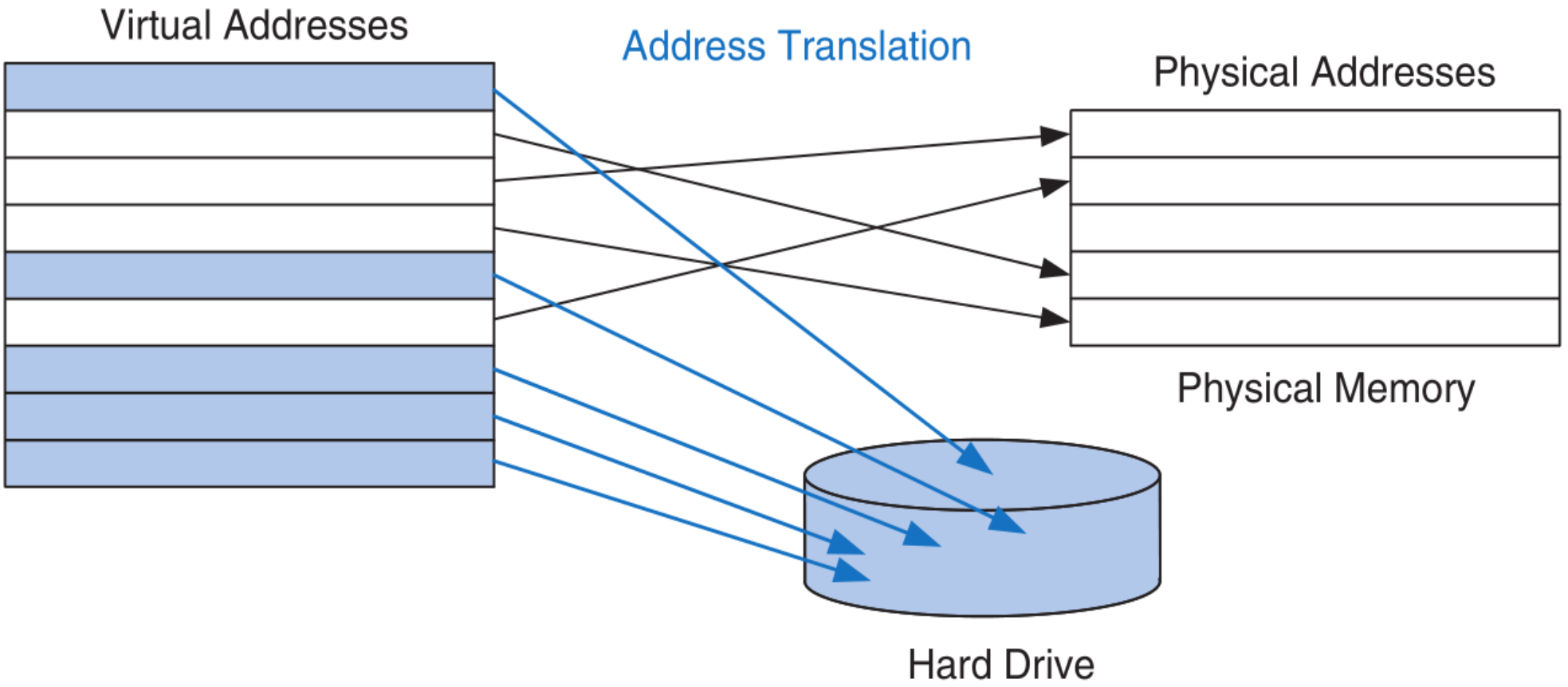
Physical memory is likewise divided into **physical pages**, called **frames**, of the same size.



A virtual page may be located in physical memory or on the hard drive.

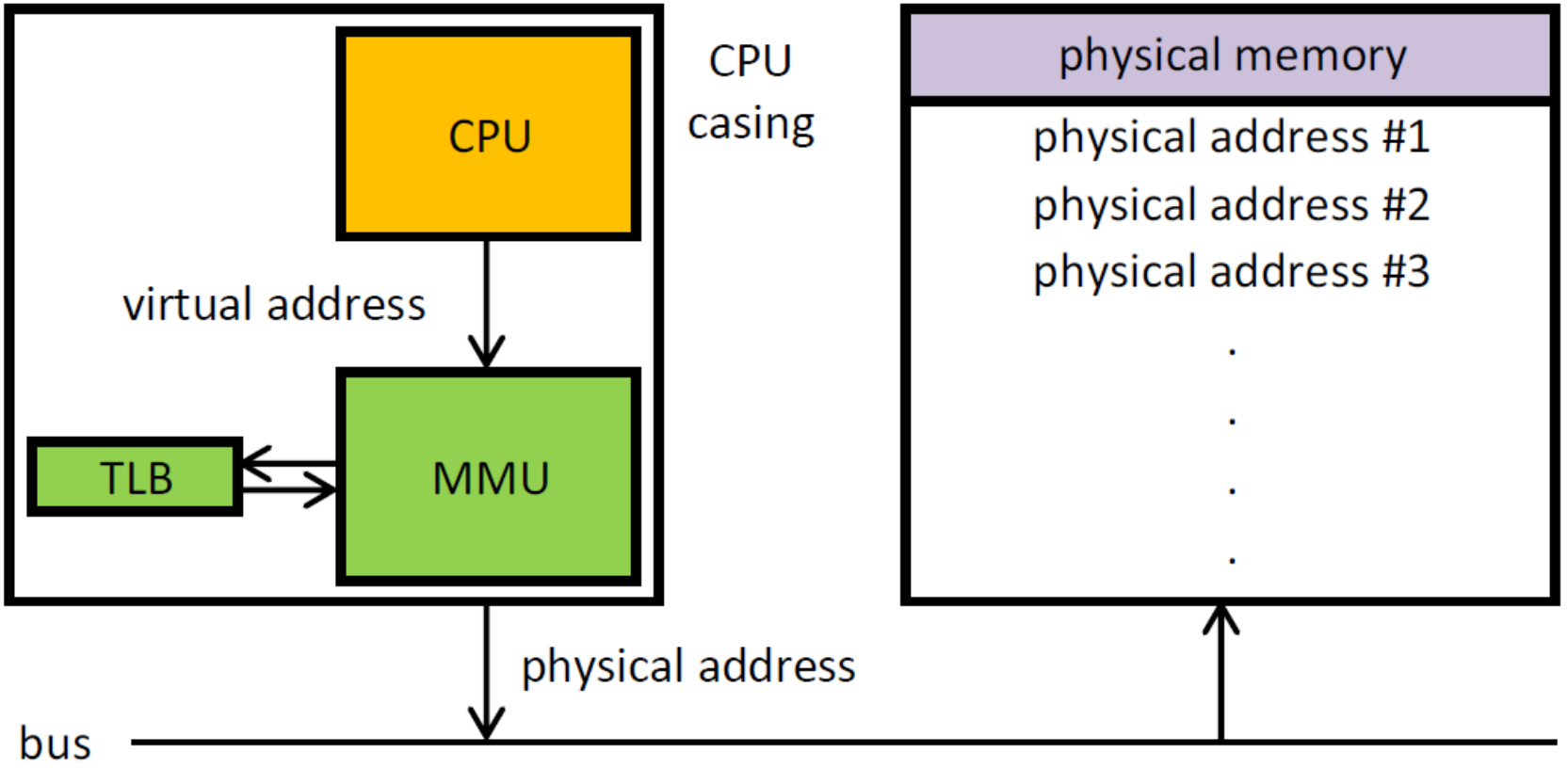
Address translation – the process of determining the physical address from the virtual address

If the processor attempts to access a virtual address that is not in physical memory, a **page fault** occurs, and the page is loaded from the hard drive into physical memory.



Any virtual page can map to any physical page.

That is, physical memory behaves as a fully associative cache for virtual memory.



A **memory management unit** (**MMU**) is a hardware unit, primarily performing the translation of virtual memory addresses to physical addresses.

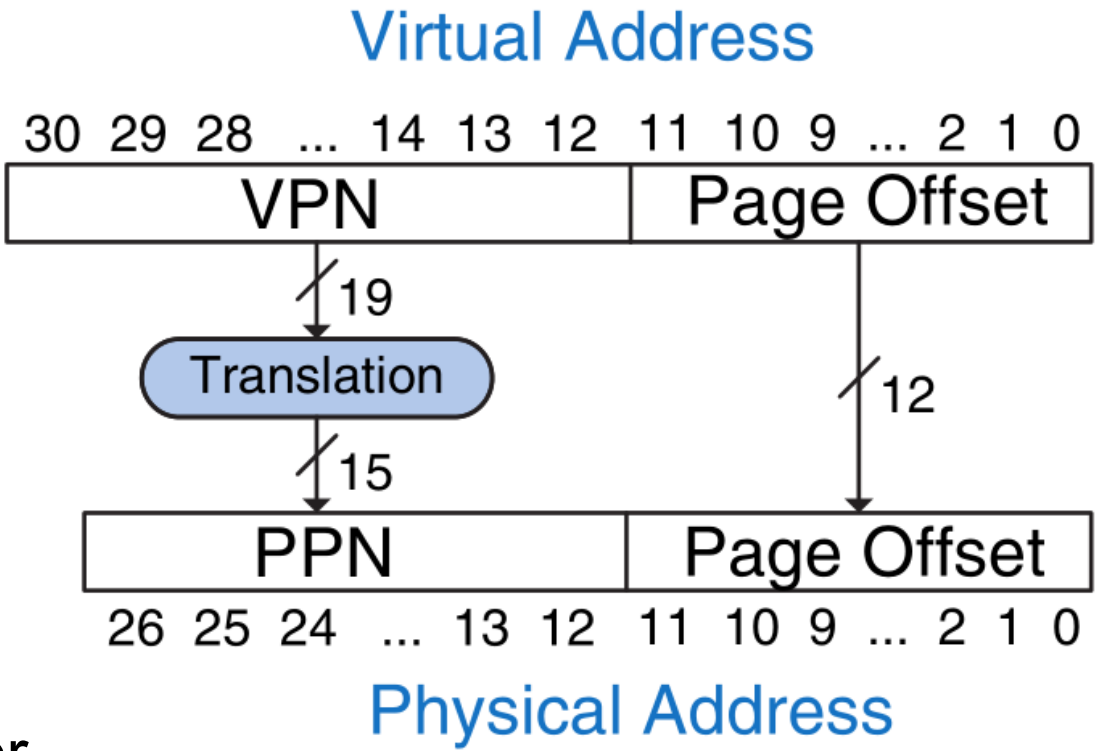
It is usually implemented as part of the central processing unit (CPU), but it also can be in the form of a separate integrated circuit.

MMU divides the virtual address space into pages, each having a size which is a power of 2, usually a few kilobytes, but they may be much larger.

The bottom bits of the address (the offset within a page) are left unchanged.

The upper address bits are the virtual page numbers.

Only the page numbers need to be translated to obtain the physical addresses from the virtual addresses.



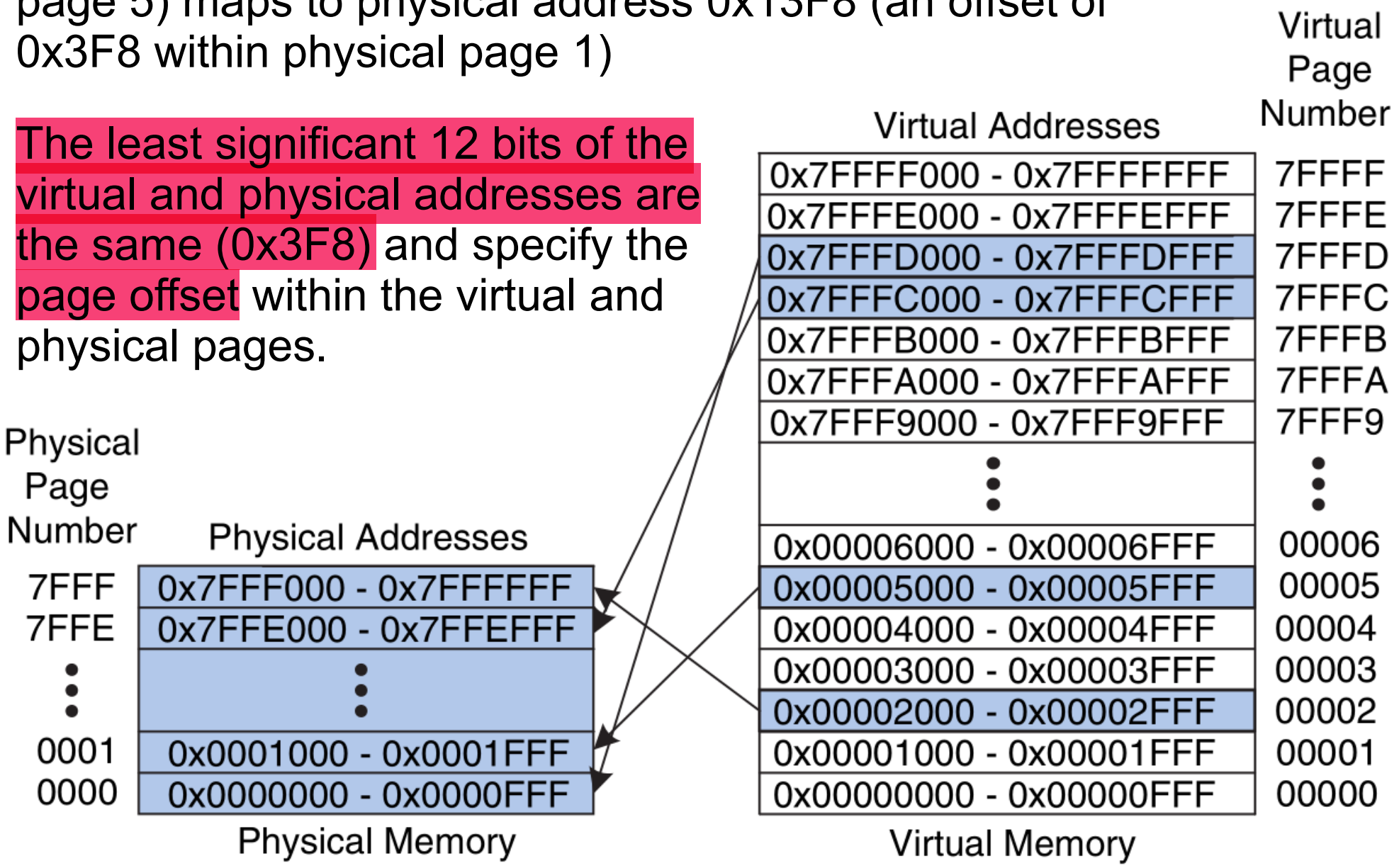
VPN – virtual page number

PPN – physical page number

2 GB of virtual memory, 128 MB of physical memory, 4-KB pages

Virtual address 0x53F8 (an offset of 0x3F8 within virtual page 5) maps to physical address 0x13F8 (an offset of 0x3F8 within physical page 1)

The least significant 12 bits of the virtual and physical addresses are the same (0x3F8) and specify the page offset within the virtual and physical pages.



Most MMUs use an in-memory table of items called a **page table**, containing one **page table entry** (**PTE**) per page, to map virtual page numbers to physical page numbers in main memory.

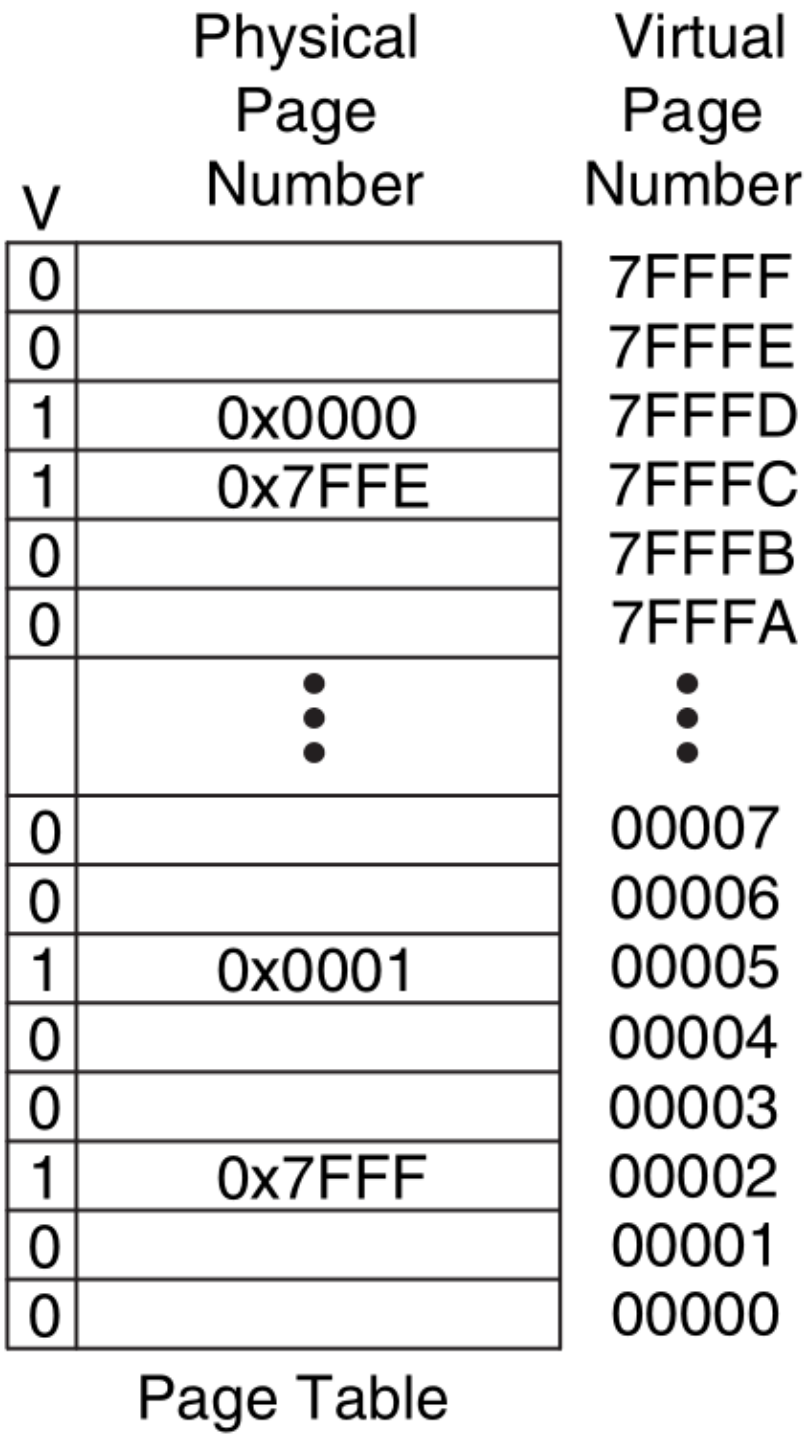
Each load or store instruction requires a page table access followed by a physical memory access.

The page table access translates the virtual address used by the program to a physical address.

The physical address is then used to actually read or write the data.

Hence, each load or store involves two physical memory accesses: a page table access, and a data access.

To speed up address translation, a **translation lookaside buffer** (**TLB**) caches the most commonly used page table entries.



The page table contains an entry for each virtual page.

This entry contains a physical page number and a valid bit.

If the valid bit is 1, the virtual page maps to the physical page specified in the entry.

Otherwise, the virtual page is found on the hard drive.

The page table is stored in physical memory.

The processor uses a dedicated register, called the **page table register**, to store the base address of the page table in physical memory.

In real programs, the vast majority of accesses hit in the TLB, avoiding the time-consuming page table reads from physical memory.

A TLB is organized as a fully associative cache and typically holds 16 to 512 entries.

Each TLB entry holds a virtual page number and its corresponding physical page number.

The TLB is accessed using the virtual page number.

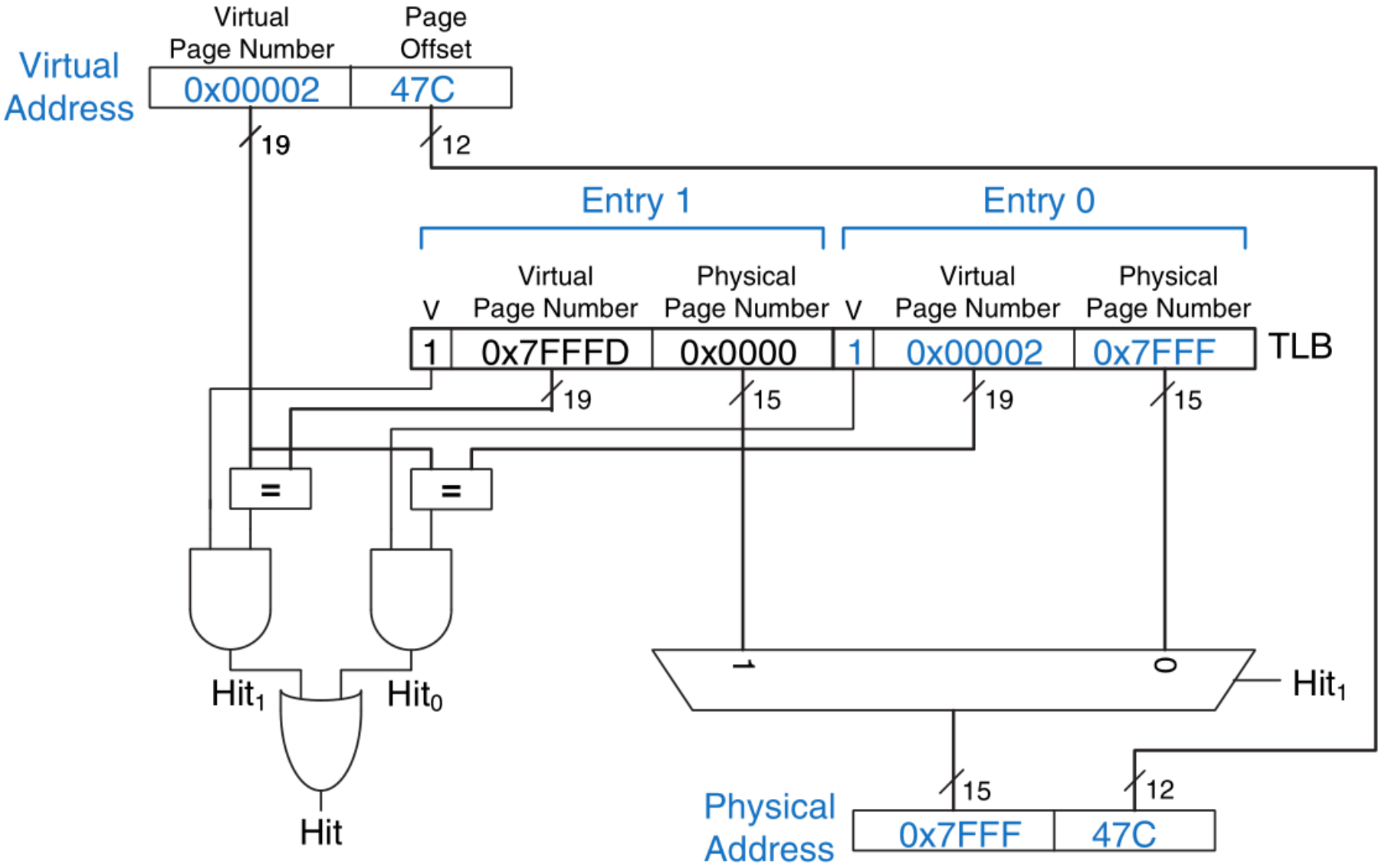
If the TLB hits, it returns the corresponding physical page number.

Otherwise, the processor must read the page table in physical memory.

The TLB is designed to be small enough that it can be accessed in less than one cycle.

TLBs typically have a hit rate of greater than 99%.

The 2-entry TLB with the request for virtual address 0x247C



Memory protection

Two reasons for using virtual memory:

1) to provide a fast, inexpensive, large memory

2) to provide protection between concurrently running programs

No program should be able to access another program's memory without permission.

Virtual memory systems provide memory protection by giving each program its own **virtual address space**.

Each program can use as much memory as it wants in that virtual address space, but only a portion of the virtual address space is in physical memory at any given time.

A program can access only those physical pages that are mapped in its page table.

In this way, a program cannot accidentally or maliciously access another program's physical pages, because they are not mapped in its page table.

In some cases, multiple programs access common instructions or data.

The operating system adds control bits to each page table entry to determine which programs, if any, can write to the shared physical pages.

Replacement policies

Virtual memory systems use **writeback** and an **approximate least recently used** (**ALRU**) replacement policy.

Under the writeback policy, the physical page is written back to the hard drive only when it is evicted from physical memory.

The processor pages out to the hard drive one of the least recently used physical pages when a page fault occurs, then replaces that page with the missing virtual page.

To support these replacement policies, each page table entry contains two additional status bits: a dirty bit D and a use bit U.

The dirty bit is 1 if any store instructions have changed the physical page since it was read from the hard drive.

When a physical page is paged out, it needs to be written back to the hard drive only if its dirty bit is 1.

Otherwise, the hard drive already holds an exact copy of the page.

The use bit is 1 if the physical page has been accessed recently.

As in a cache system, exact LRU replacement would be impractically complicated.

Instead, the OS approximates LRU replacement by periodically resetting all the use bits in the page table.

When a page is accessed, its use bit is set to 1.

Upon a page fault, the OS finds a page with U=0 to page out of physical memory.

Thus, it does not necessarily replace the least recently used page, just one of the least recently used pages.

Multilevel page tables

Page tables can occupy a large amount of physical memory.

To conserve physical memory, page tables can be broken up into multiple (usually two) levels.

The 1st-level page table is always kept in physical memory.

It indicates where small second-level page tables are stored in virtual memory.

The second-level page tables each contain the actual translations for a range of virtual pages.

If a particular range of translations is not actively used, the corresponding second-level page table can be paged out to the hard drive so it does not waste physical memory.

In a two-level page table, the virtual page number is split into two parts:

1) the **page table number**

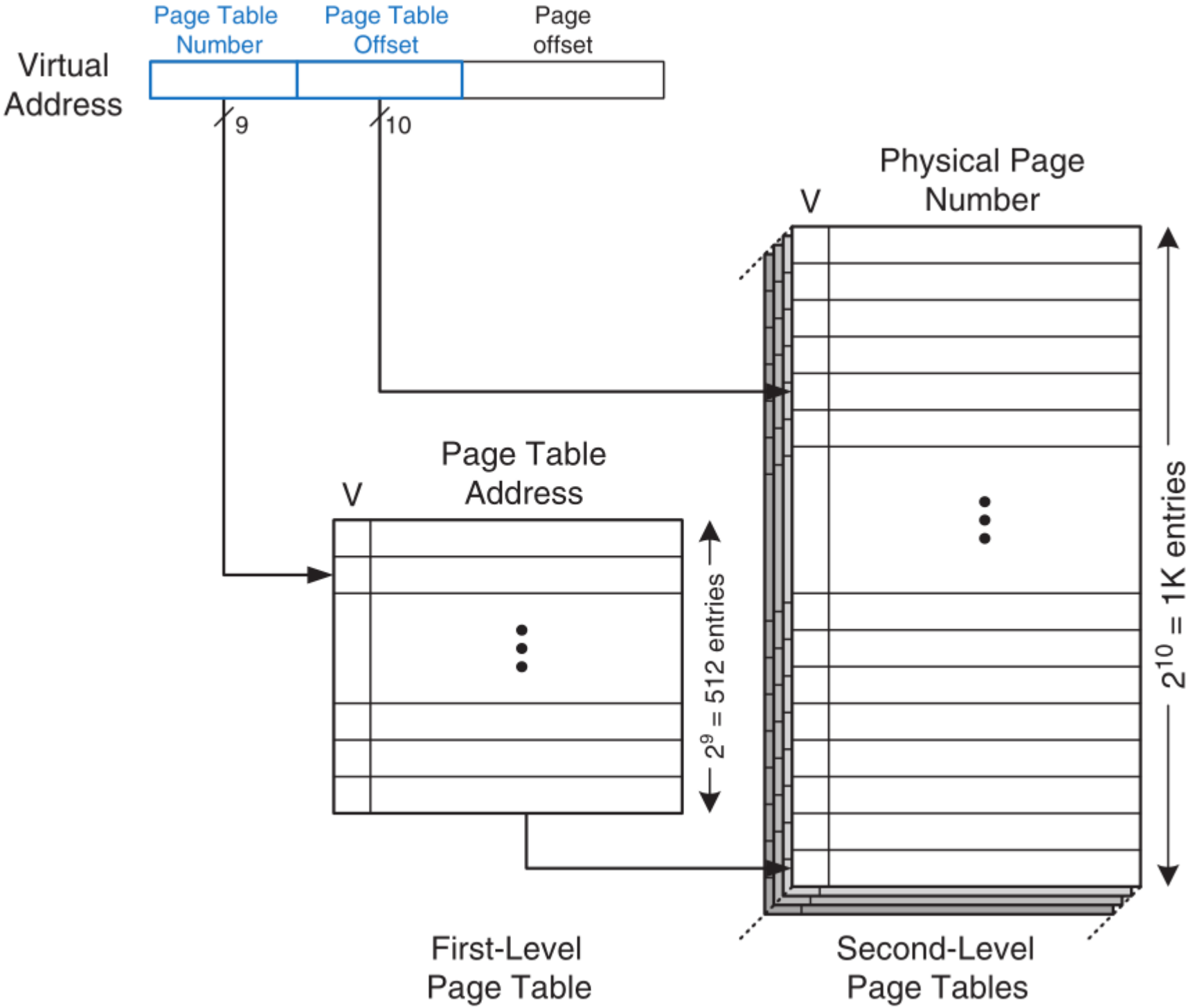
2) the **page table offset**

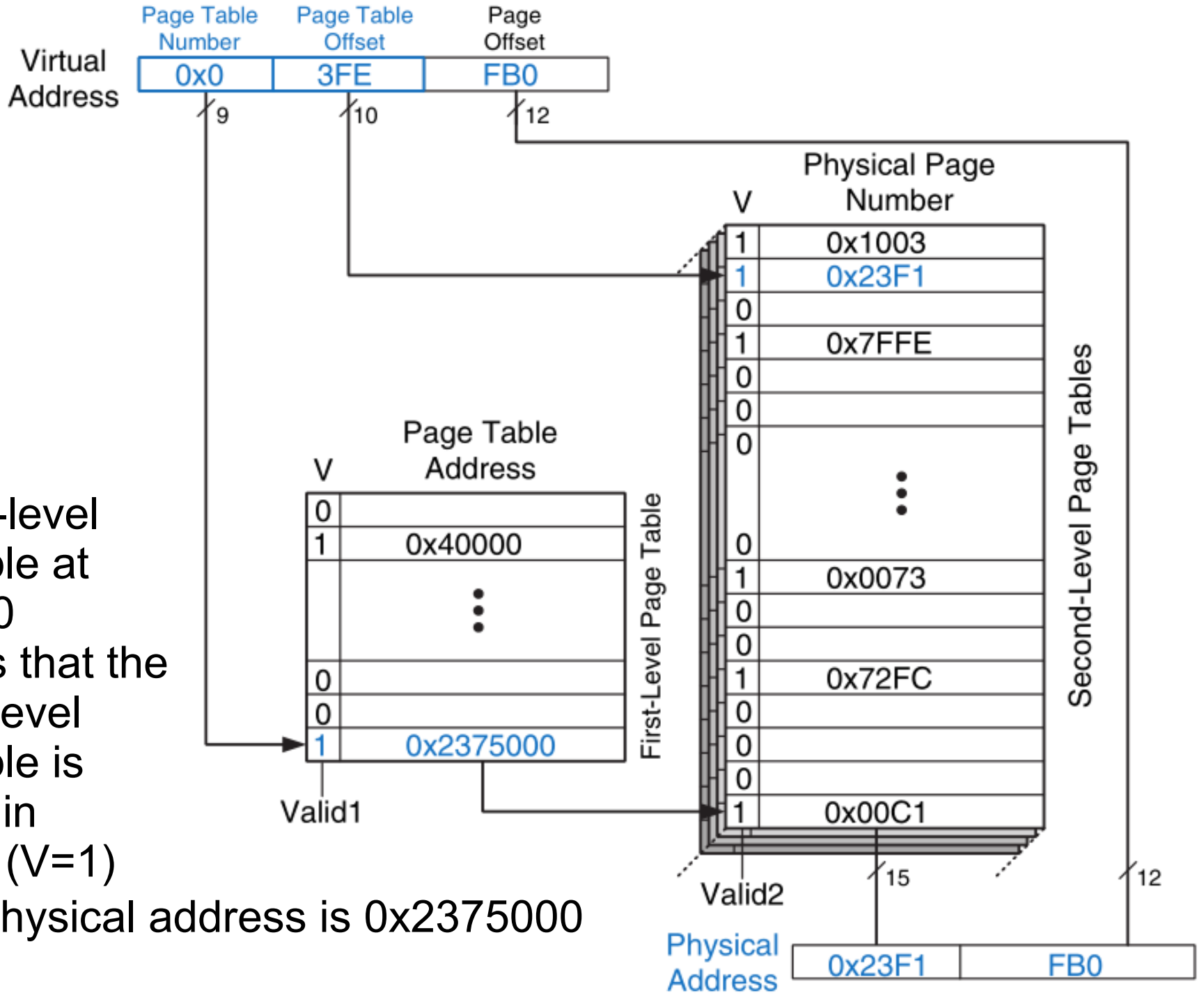
The page table number indexes the first-level page table, which must reside in physical memory.

The first-level page table entry gives the base address of the second-level page table or indicates that it must be fetched from the hard drive when V is 0.

The page table offset indexes the 2^{nd}-level page table.

The drawback of a two-level page table is that it adds yet another memory access for translation when the TLB misses.


Page Table Number
Page Table Offset
Page offset
Virtual Address
9
10
Physical Page Number
V
Page Table Address
V
2^9 = 512 entries
2^{10} = 1K entries
First-Level Page Table
Second-Level Page Tables

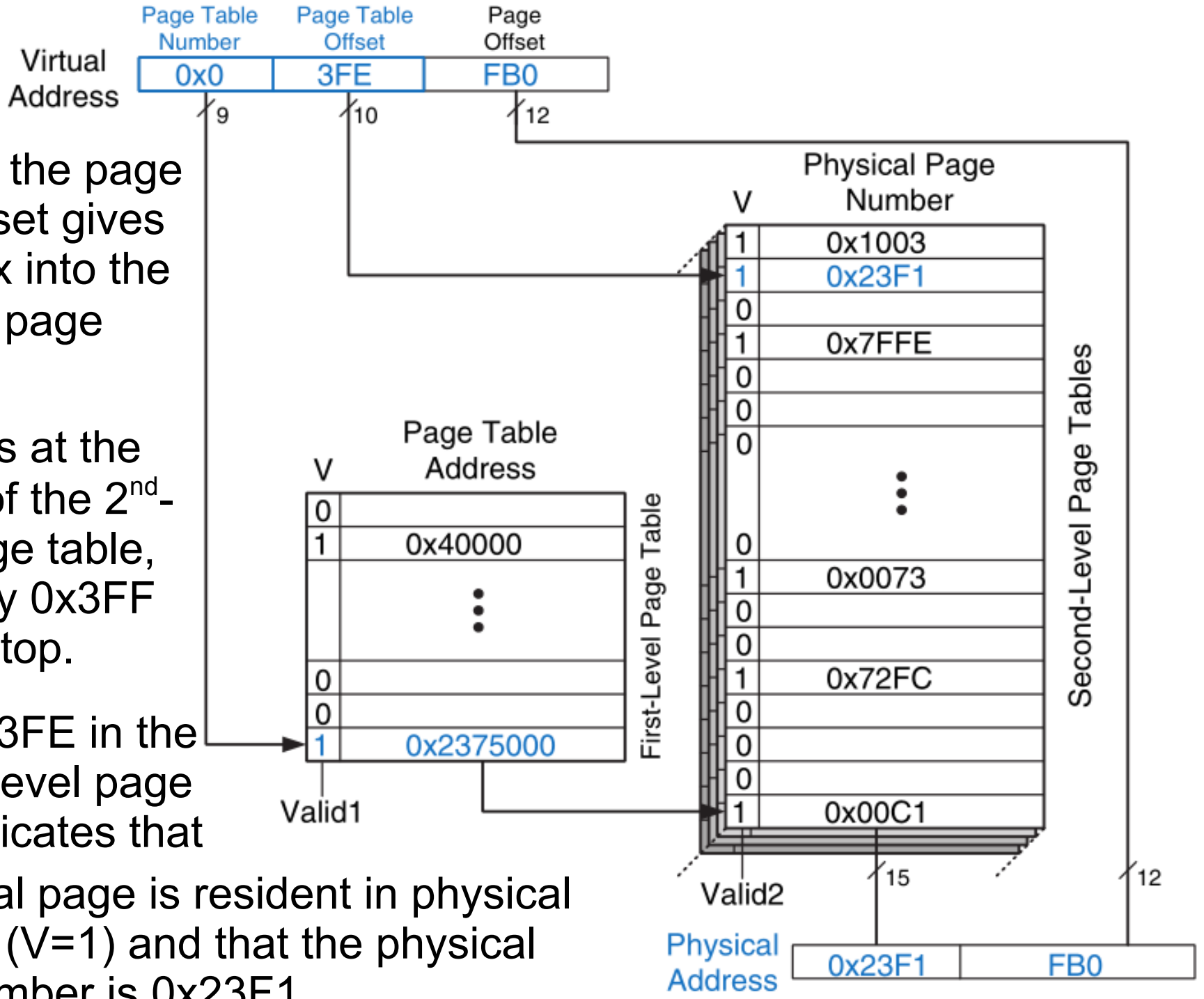


The first-level page table at entry 0x0 indicates that the second-level page table is resident in memory (V=1) and its physical address is 0x2375000

0x3FE – the page table offset gives the index into the 2nd-level page table.

Entry 0 is at the bottom of the 2nd-level page table, and entry 0x3FF is at the top.

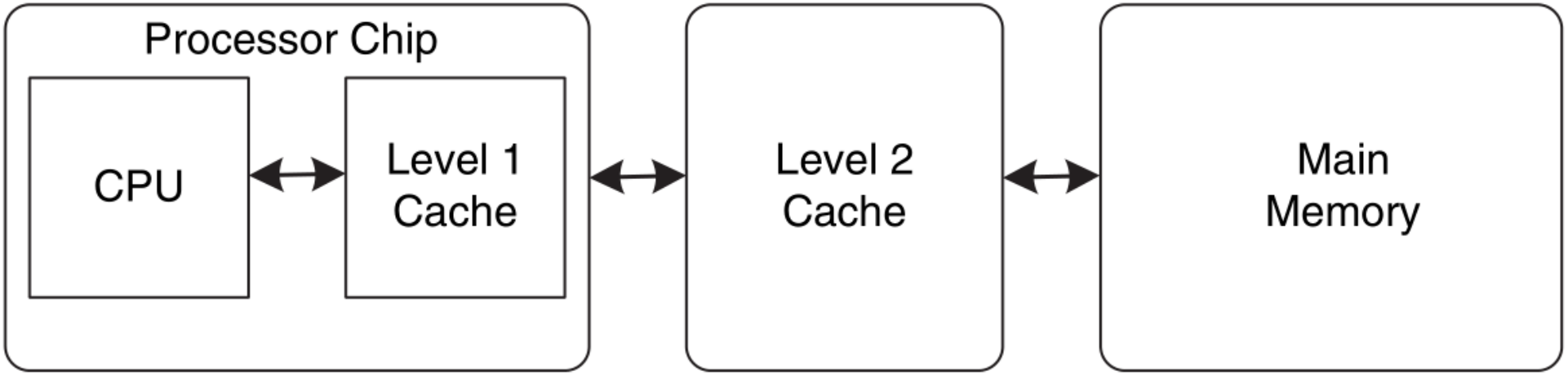
Entry 0x3FE in the second-level page table indicates that the virtual page is resident in physical memory (V=1) and that the physical page number is 0x23F1



Exercise 20.1

You've joined a hot new Internet startup to build wrist watches with a built-in Web browser.

It uses an embedded processor with the following multilevel cache scheme



The processor includes a small on-chip cache in addition to a large off-chip second-level cache.

Assume that the processor uses 32-bit physical addresses but accesses data only on word boundaries.

The caches have the following characteristics

Characteristic	On-chip Cache	Off-chip Cache
Organization	Four-way set associative	Direct mapped
Hit rate	A	B
Access time	t_a	t_b
Block size	16 bytes	16 bytes
Number of blocks	512	256K

The DRAM has an access time of t_m and a size of 512 MB.

a) For a given word in memory, what is the total number of locations in which it might be found in the on-chip cache and in the second-level cache?

b) What is the size, in bits, of each tag for the on-chip cache and the second-level cache?

c) Give an expression for the average memory read access time.

The caches are accessed in sequence.