How is Virtual Memory Translated to Physical Memory?

Table of contents

How is Virtual Memory Translated to Physical Memory?

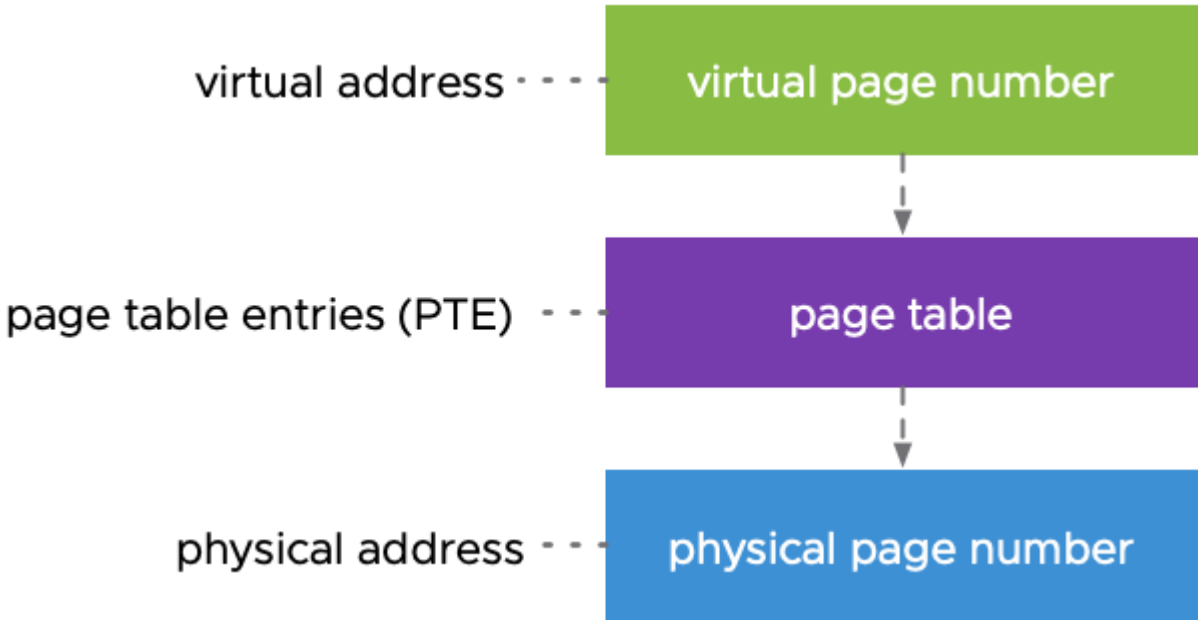
Overview

Memory is one of the most important host resources. For workloads to access global system memory, we need to make sure virtual memory addresses are mapped to the physical addresses. There are several components working together to perform these translations as efficient as possible. This blog post will cover the basics on how virtual memory addresses are translated.

Memory Translations

The physical address space is your system RAM, the memory modules inside your ESXi hosts, also referred to as the global system memory. When talking about virtual memory, we are talking about the memory that is controlled by an operating system, or a hypervisor like vSphere ESXi. Whenever workloads access data in memory, the system needs to look up the physical memory address that matches the virtual address. This is what we refer to as memory translations or mappings.

To map virtual memory addresses to physical memory addresses, page tables are used. A page table consists of numerous page table entries (PTE).

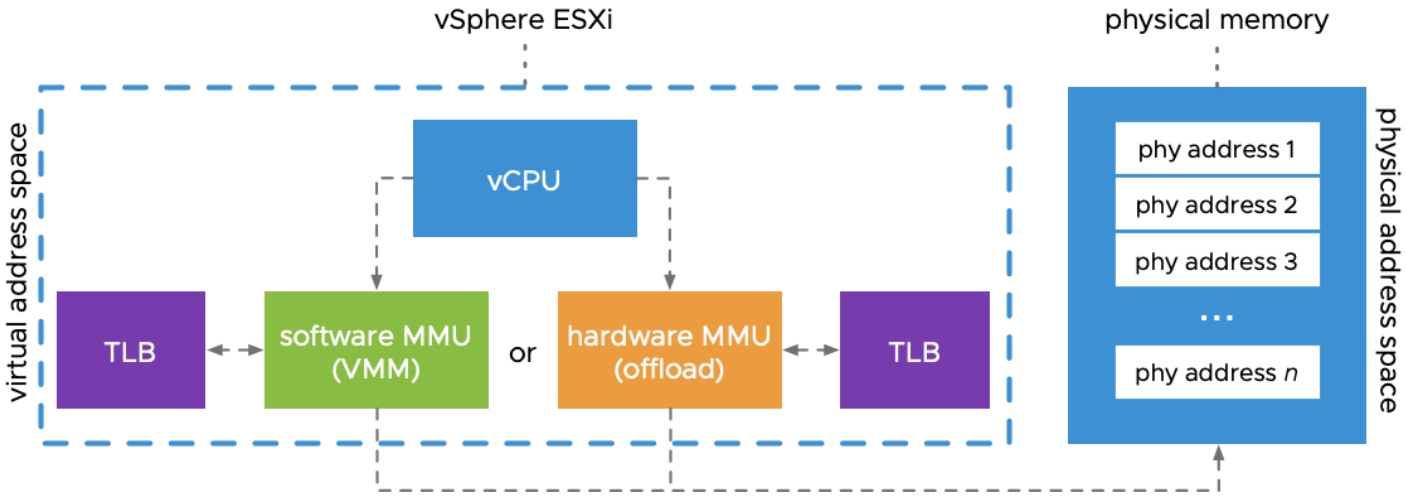


One memory page in a PTE contains data structures consisting of different sizes of 'words'. Each type of word contains multiple bytes of data (*WORD* (16 bits/2 bytes), *DWORD* (32 bits/4 bytes) and *QWORD* (64 bits/8 bytes)). Executing memory translations for every possible word, or virtual memory page, into physical memory address is not very efficient as this could potentially be billions of PTE's. We need PTE's to find the physical address space in the system's global memory, so there is no way around them.

To make memory translations more efficient, we use page tables to group chunks of memory addresses in one mapping. Looking at an example of a DWORD entry of 4 bytes; A page table covers 4 kilobytes instead of just the 4 bytes of data in a single page entry. For example, using a page table, we can translate virtual address space 0 to 4095 and say this is found in physical address space 4096 to 8191. Now we no longer need to map all the PTE's separately, and be far more efficient by using page tables.

MMU and TLB

The page tables are managed by a Memory Management Unit (MMU). All the physical memory references are passed through the MMU. The MMU is responsible for the translation between virtual memory addresses and physical memory addresses. With vSphere ESXi, a virtual machine's vCPU will call out to MMU functionality by the Virtual Machine Monitor (VMM) process, or a hardware MMU supported by a vendor specific CPU offloading instruction.



The Memory Management Unit (MMU) works with the Translation Lookaside Buffer (TLB) to map the virtual memory addresses to the physical memory layer. The page table always resides in physical memory, and having to look up the memory pages directly in physical memory, can be a costly exercise for the MMU as it introduces latency. That is where the TLB comes into play.

TLB in Detail

The TLB acts as a cache for the MMU that is used to reduce the time taken to access physical memory. The TLB is a part of the MMU. Depending on the make and model of a CPU, there's more than one TLB, or even multiple levels of TLB like with memory caches to avoid TLB misses and ensuring as low as possible memory latency.

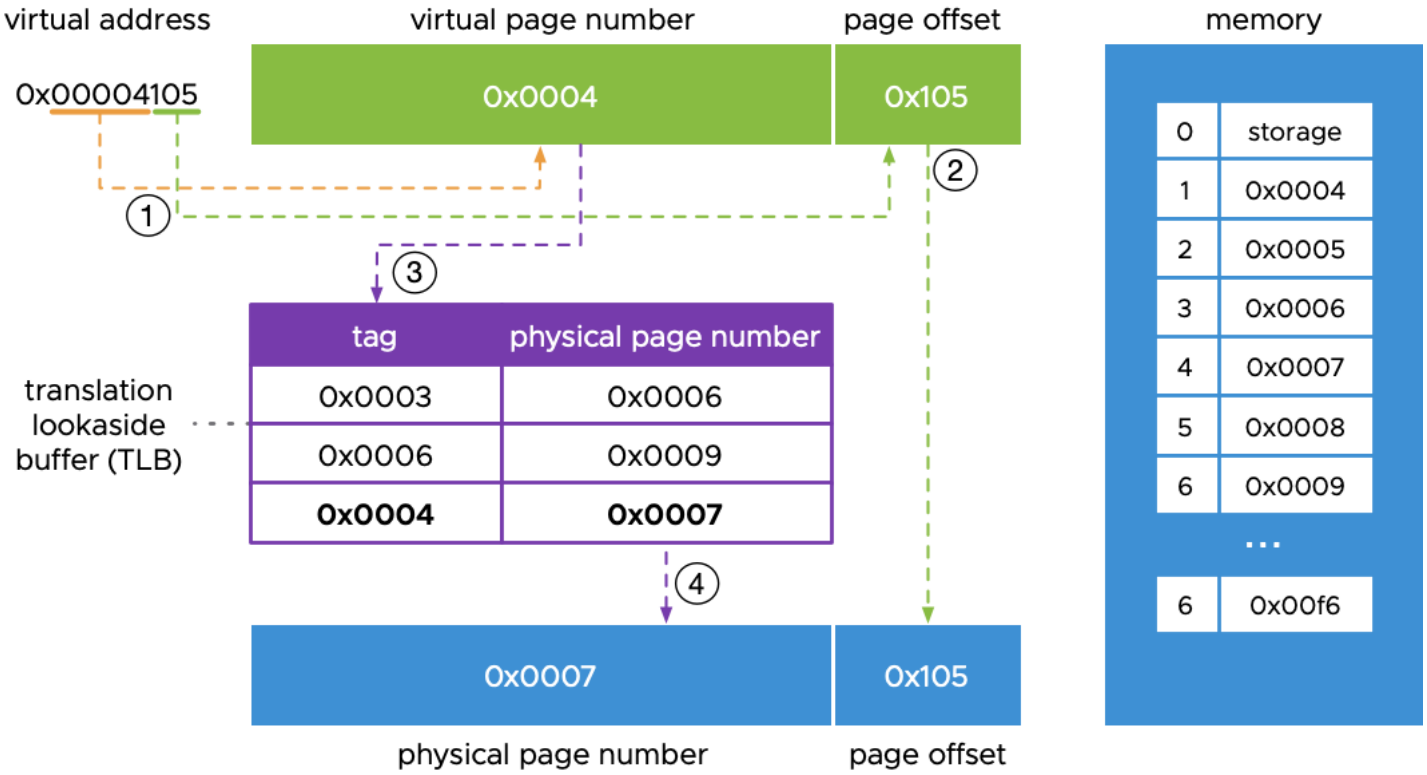
In essence, the TLB stores recent memory translations of virtual to physical. It is a cache for page tables. Because it is part of the MMU, the TLB lives inside the CPU package. This is why the TLB is faster than main memory, which is where the page tables exists. Typically access times for a TLB is ~10 ns where main memory access times are around 100 ns.

Now that we covered the basics on memory translation, let's take a look at some example scenarios for the TLB.

TLB hit

A virtual memory address comes in, and needs to be translated to the physical address. The first step is always to dissect the virtual address into a virtual page number, and the page offset. The offset consists of the last bits of the virtual address. The offset bits are not translated and passed through to the physical memory address. The offset contains bits that can represent all the memory addresses in a page table.

So, the offset is directly mapped to the physical memory layer, and the virtual page number matches a tag already in the TLB. The MMU now immediately knows what physical memory page to access without the need to look into the global memory.

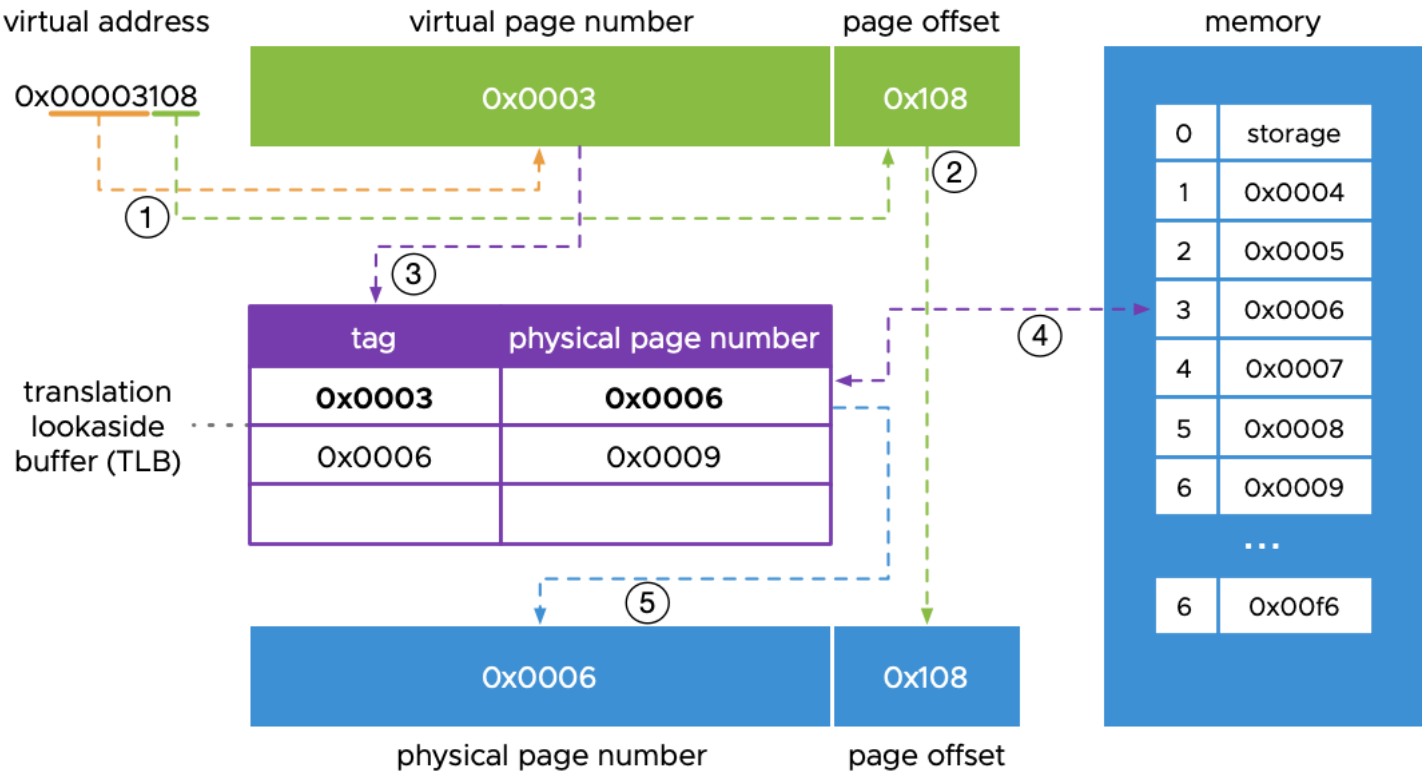


In the example provided in the above diagram, the virtual page number is found in the TLB, and immediately translated to the physical page number.

1. The virtual address is dissected in the virtual page number and the page offset.
2. The page offset is passed through as it is not translated.
3. The virtual page number is looked up in the TLB, looking for a tag with the corresponding number.
4. There is an entry in the TLB (hit), meaning we immediately can translate the virtual to the physical address.

TLB miss

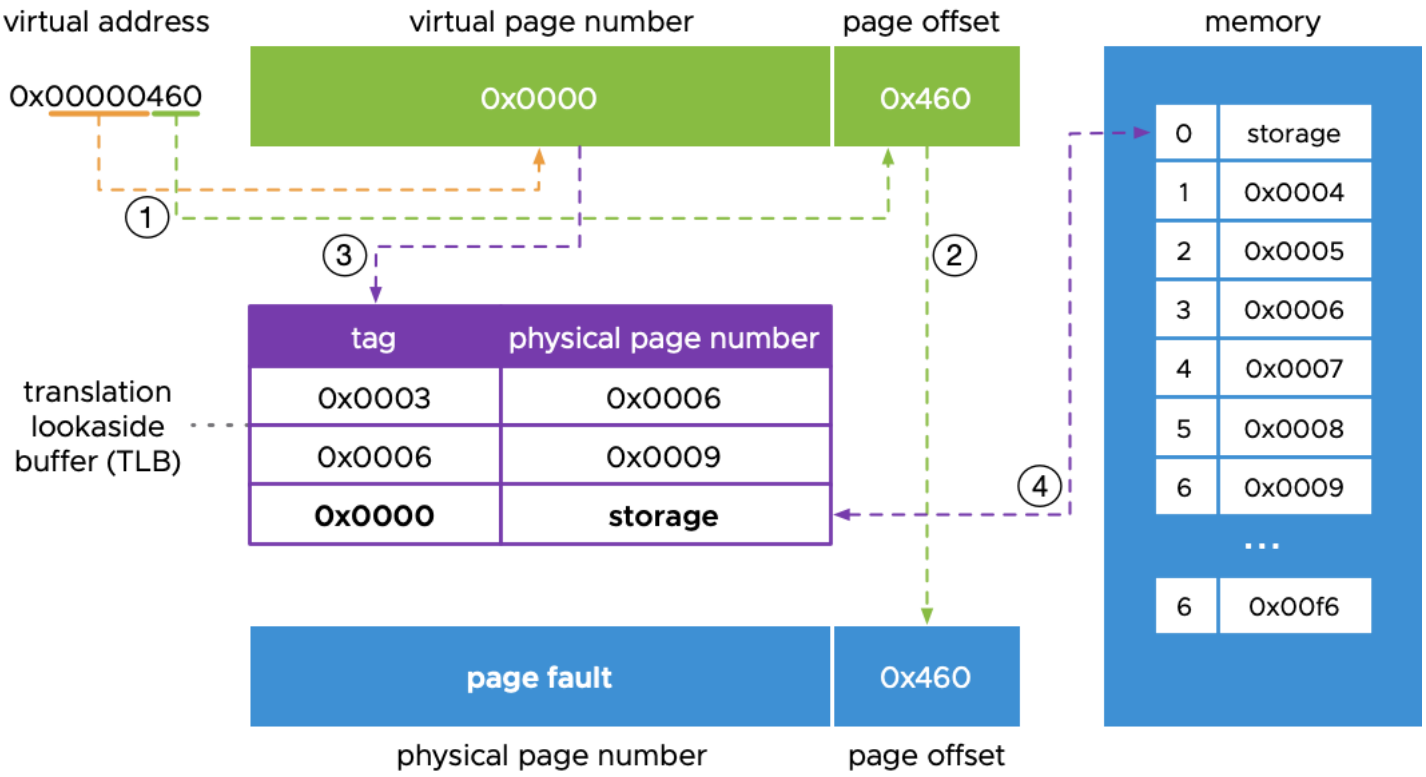
What happens when a virtual page number is not found in the TLB, also referred to as a TLB miss? The TLB needs to consult the system's global memory to understand what physical page number is used. Reaching out to physical memory means higher latency compared to a TLB hit. If the TLB is full and a TLB miss occurs, the least recent TLB entry is flushed, and the new entry is placed instead of it. In the following example, the virtual page number is not found in the TLB, and the TLB needs to look into memory to get the page number.



1. The virtual address is dissected in the virtual page number and the page offset.
2. The page offset is passed through as it is not translated.
3. The virtual page number is looked up in the TLB, looking for a tag with a corresponding number. In this example, the TLB does not yet have a valid entry.
4. TLB reaches out to memory to find page number 3 (because of the tag, derived from the virtual page number). Page number 3 is retrieved in memory with value 0x0006.
5. The memory translation is done and the entry is now cached in the TLB.

Retrieve from storage

A TLB miss is not ideal, but the worst-case scenario is data that is not residing in memory but on storage media (flash or disk). Where we are talking nanoseconds to retrieve data in caches or global memory, getting data from storage media will quickly run into milliseconds or seconds depending on the media used.



1. The virtual address is dissected in the virtual page number and the page offset.
2. The page offset is passed through as it is not translated.
3. The virtual page number is looked up in the TLB, looking for a tag with a corresponding number. In this example, the TLB does not yet have a valid entry.
4. TLB reaches out to memory to find page number 0 (because of the tag, derived from the virtual page number). Page number 0 is retrieved in memory but finds that the data does not resides in memory, but on storage. A page fault is triggered, because we cannot translate memory pages for data that is not in memory. We need to wait for the data from storage.

To Conclude

We only covered the basics about how memory translations work. The information we discussed should provide you with a basic understanding about what is going on under the hood of your virtual machine and ESXi hosts when it comes to memory access.

To understand how the MMU and TLB process memory access, is fundamental for memory-centric vSphere features, like vMotion. vMotion uses memory constructs like PTE and TLB. During a vMotion operation, PTE's are set to read-only access and the TLB is flushed.