Virtual Memory

Introduction

1. Discuss how computer architecture supports virtual memory to allow modern operating systems to be much more efficient

Why Virtual Memory?

1. Programmer and hardware have different view of memory
    - Hardware views memory as multiple DIMMs
    - Programmer views memory as one big array
        - Split between system, code, data, heap, stack
2. Actual memory a programmer uses may be less or more than what is physically available
    - Multiple programs run simultaneously (file system, word processor, etc)
        - Each program thinks it has access to the entire memory space
3. Virtual memory: Method of reconciling how programmer and hardware view memory

Virtual Memory Quiz

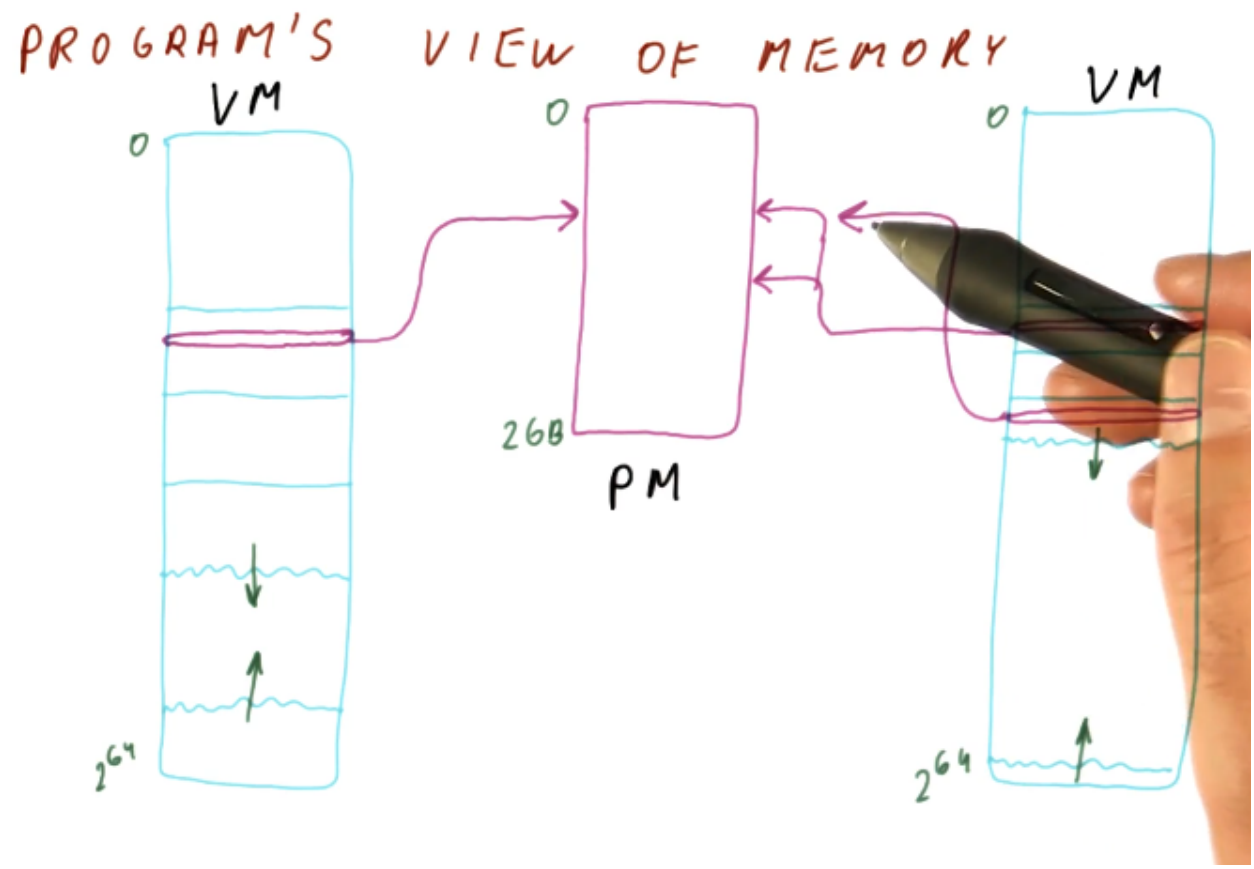
1. Computer has 16 active applications, each with 32-bit address space (4 GB)
2. What does the system actually have for memory?
    - Two 2GB memory modules (true)
    - Four 4GB memory modules (true)
    - Eight 8GB memory modules (true)
    - One 16GB memory module (true)
3. Virtual memory allows each program to suppose it has access to the entire physical memory

Processor's View of Memory

1. Processor sees physical memory (actual memory modules)
    - Sometimes < 4GB
    - Almost never 4GB per process
    - Never 16 exabytes per process (each process in a 64-bit machine)
2. Amount of physical memory << memory programs can access
3. Addresses have a 1:1 mapping to bytes/words in physical memory

Program's View of Memory

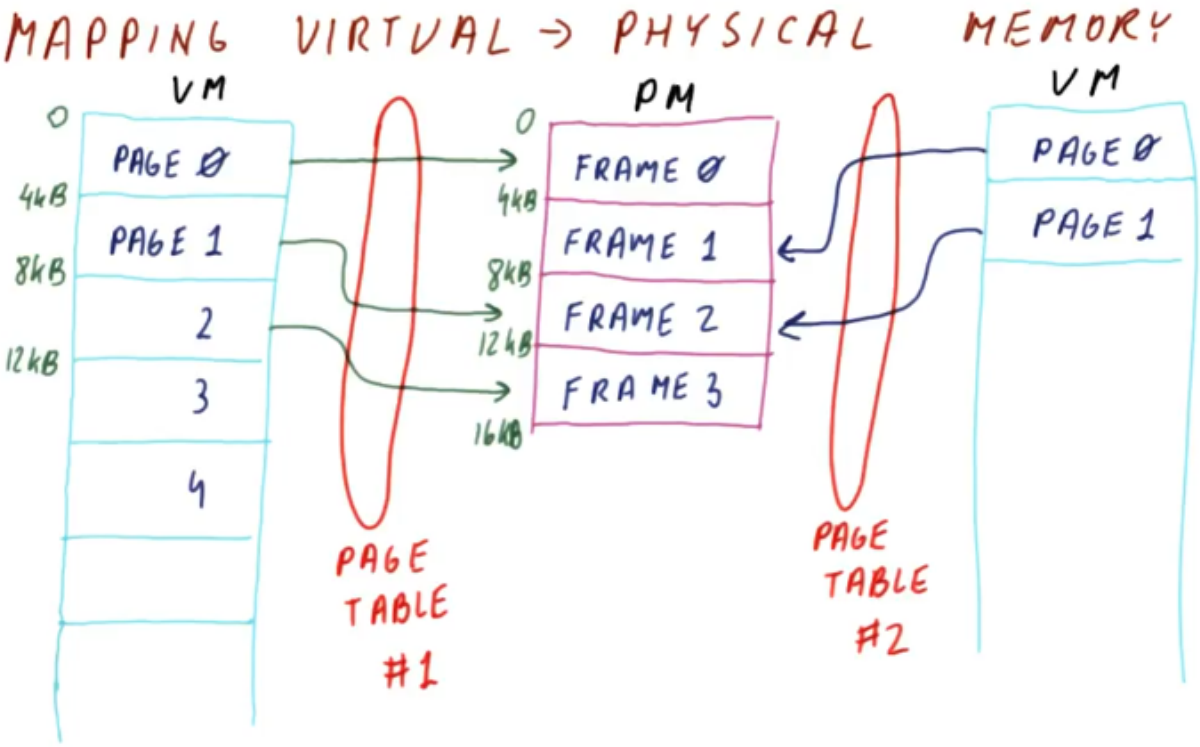
1. Program thinks it has lots of memory (2^64 for a 64-bit machine), but in reality, there isn't enough physical memory to support this (virtual view)
2. All programs share this view of virtual memory
3. Need to figure out how to map virtual addresses to physical addresses
    - Data sharing across programs is not constrained to using the same virtual address for a shared physical address



Program's View of Memory

Mapping Virtual Physical Memory

1. Mapping VA -> PA would be very difficult if any virtual address could map to any physical address
   - Instead, we split virtual memory into pages (typically 4KB)
   - Virtual memory pages map to physical memory frames
2. Two processes can share memory by mapping their virtual address to the same physical frame
3. The operating system determines how to make the mapping
   - This is accomplished through a data structure called a page table



Mapping Virtual -> Physical Memory

Page Table Quiz

1. Consider a system with the following specifications:
   - Physical memory is 2GB
   - Virtual memory is 4GB
   - Page size is 4KB
2. How many page frames?
   - 2^31 / 2^12 = 2^19
3. How many entries in each page table?
   - 2^32 / 2^12 = 2^20
4. Each process requires a page table with 2^20 entries

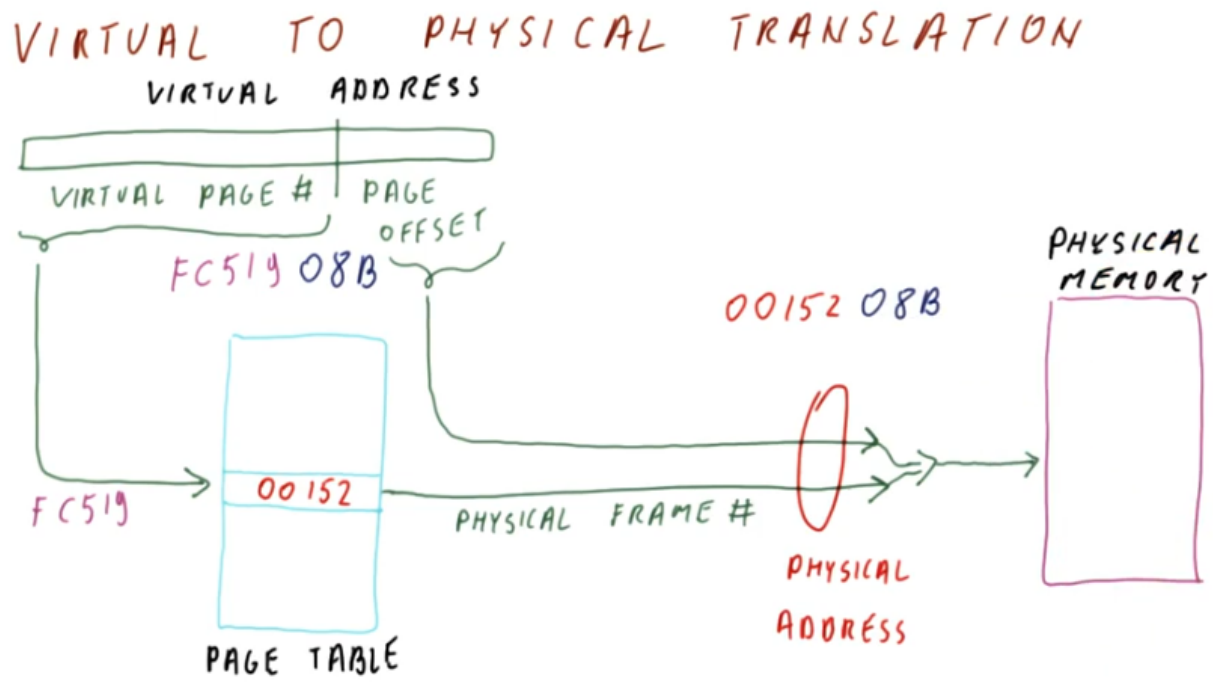
Where Is the Missing Memory?

1. Additional pages that don't fit in memory are stored on the disk
2. These pages can not be directly accessed by the processor; they must first be brought into memory

Virtual to Physical Translation

1. Program generates a virtual address (what a load will compute)
   - Contains a virtual page number and page offset
   - Virtual page number: Which page we want
     - Index into the page table
     - The entry in the page table contains the physical frame number
   - Page offset: Where in the page data should be retrieved

  - Is log2(page size) bits (12 for 4KB page)
  - Does not change when creating the physical address
- The physical frame number and page offset are combined to get a physical address



Address Translation

Address Translation Quiz

1. Consider a system with the following specifications:
   - 16-bit virtual address space
   - 20-bit physical address space
2. The page table contains the following entries:

Page Table
0x1F
0x3F
0x23
0x17

3. What physical addresses do the following virtual address translate to?
   - 0xF0F0 -> 0001 0111 11 0000 1111 0000 -> 0x5F0F0
   - 0x001F -> 0001 1111 00 0000 0001 1111 -> 0x7C01F
4. Solution:
   - 4 entries in page table, so 2 most significant bits of virtual address are the virtual page number
   - We drop the additional most significant bits to get to the 20-bit address

Size of Flat Page Table

1. 1 entry per page in the virtual address space, even for pages the program never uses

- Entry contains frame number and bits that tell us if the page is accessible
- Entry is similar in size to actual physical address

2. Overall size of a page table
   - Virtual memory / page size * size of entry
   - 4GB / 4KB * 4B = 4MB
3. A process might not be using all of its available virtual memory
4. A system with 64-bit address space would have a larger page table than could fit in memory

Flat Page Table Size Quiz

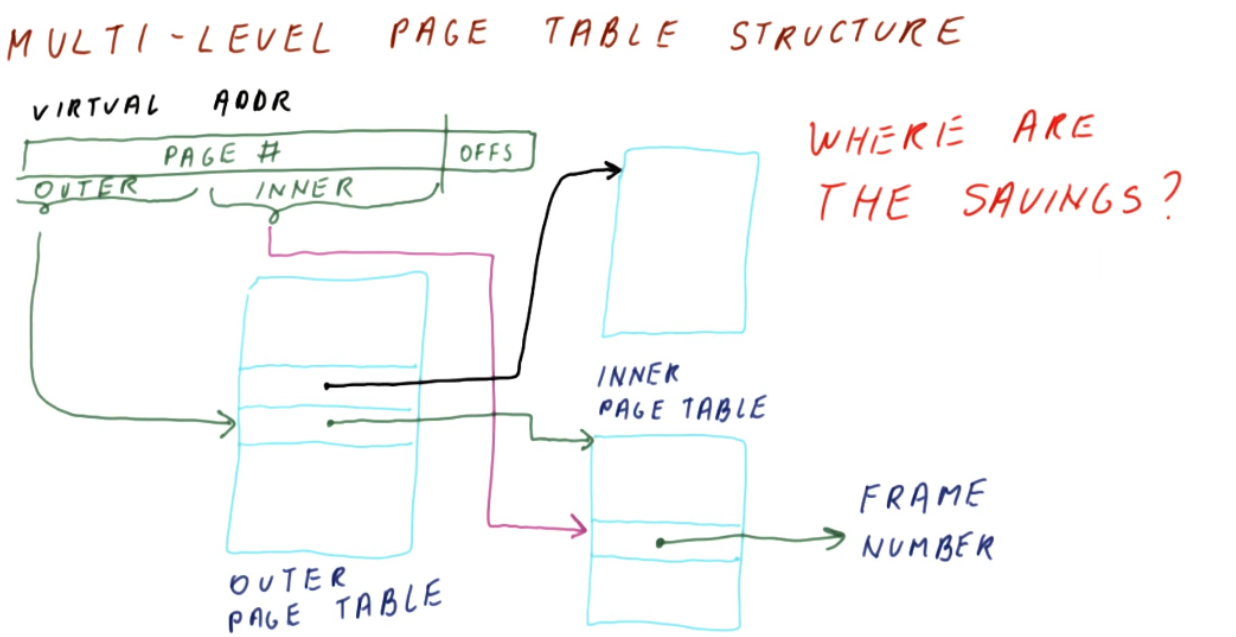
1. Consider a system with the following specifications:
   - 8B/entry (64-bit physical address space)
   - Page size of 4KB
   - Two processes in system
   - Physical memory: 2GB
   - 32-bit virtual address
   - Process #1 is 1MB of memory
   - Process #2 is using 1GB of memory
2. What is the total size of the page tables in MB?
   - 2^32 / 2^12 * 2^3 = 8MB
   - Since we have two processes, we need 16 MB of memory for page tables
   - The amount of memory the processes are using and the size of the physical memory is irrelevant

Multi Level Page Tables

1. The problem with flat page tables is that their size is proportional to the address space
   - 32-bit virtual address space requires several MB for page tables
     - This is true even for a process that only needs a few KB of memory
   - For 64-bit virtual address, the page table would be many gigabytes
     - Unusable
   - Multi-level page tables attempt to rectify these issues
     - Need to work for 64-bit address spaces
     - Size of page table should be proportional to the memory we're using
2. In a virtual address space, the code, data, and heap are stored near the top and the stack is stored near the bottom
   - Even if both use several GB, there will be terabytes of unused memory in between
3. Multi-level page tables...
   - Use bits from VA to index tables
   - Avoid use of table entries for unused portions of address space

Multi Level Page Table Structure

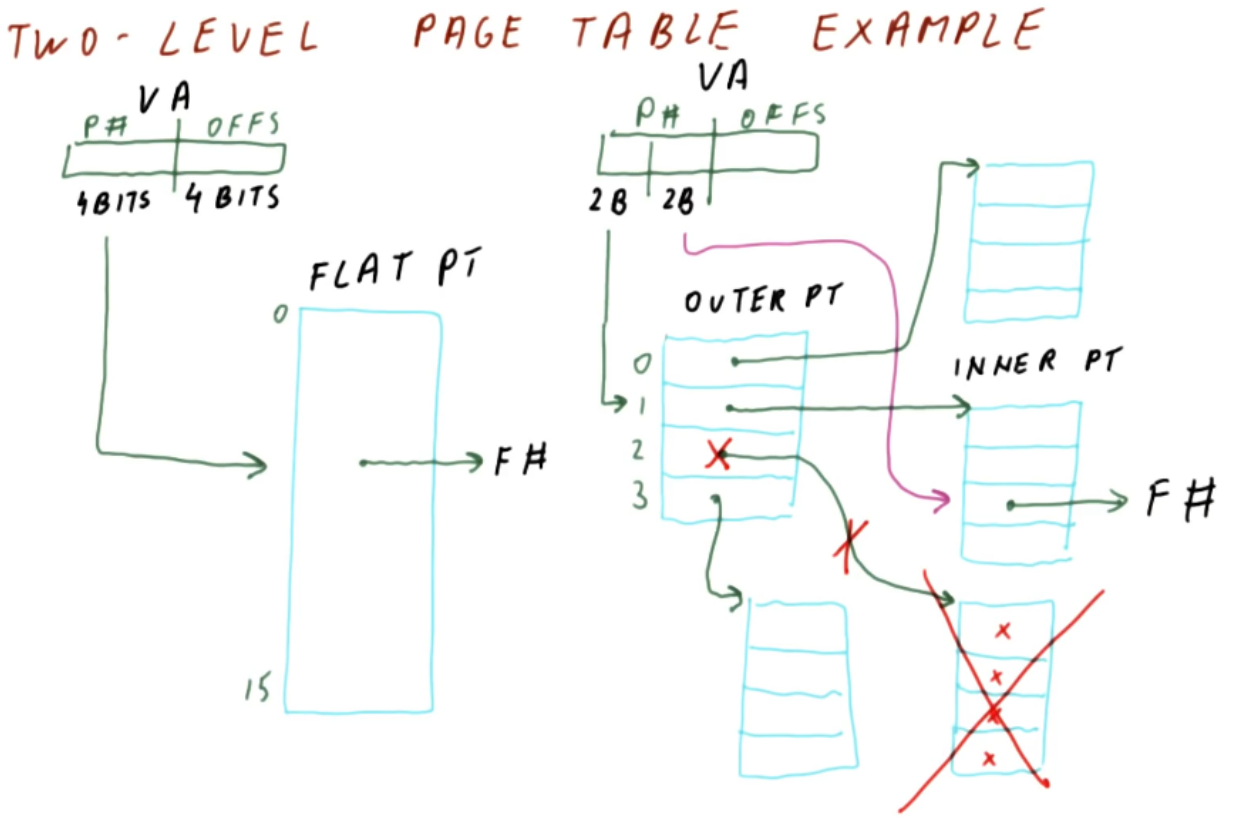
1. A virtual address still consists of page number and offset
   - The page number is split into an inner and outer page number
2. Maintain multiple page tables (another level of indirection)
   - Outer page table points to another page table
   - Inner page table contains a frame number



Multi-level Page Table

Two Level Page Table Example

1. The total number of potential memory usage is actually greater than in a flat page table
   - Inner page tables need to be able to address the same amount of memory, plus the additional overhead of the outer page table
2. However, we don't have to allocate an inner page table if none of the entries will be used
   - This is where the savings comes from



Multi-level Page Table Example

Two Level Page Table Size

1. Consider a system with the following specifications:
   - 32-bit address space
   - 4KB page
   - 1024-entry outer page table, 1024-entry inner page table
   - PT entry is 8 bytes
   - Program uses virtual memory at 0x0..0x00100000 and 0xFFFF0000..0xFFFFFFFF
2. Flat page table size
   - 2^32 / 2^12 = 2^20 * 2^3 = 8MB
3. 2-level page table size
   - Address is 10-bit outer page number, 10-bit inner page number, 12-bit offset
   - Outer page table is 2^10 * 8 = 8KB
   - The first portion of memory only needs one inner page table since the most significant 10 bits are identical for all addresses
   - The second portion of memory also only needs one inner page table since the most significant 10 bits are identical for all addresses
   - Inner page table is 2 * 2^10 * 2^3 = 16 KB
   - Total size is 24 KB (much less than the flat page table)
4. Multi-level page tables are used almost exclusively in today's processors
   - Only way to enable 64-bit address spaces

4 Level Page Table Quiz

1. Consider a system with the following specifications:

- 64-bit virtual address space (2^64)
- 64KB page size (2^16)
- 8B page table entry
- Program only uses addresses 0..4GB (2^32 bytes -> 2^16 pages)

2. Flat page table size is...
   - 2^64 / 2^16 * 2^3 = 2^51
3. 4-level page table (page number split equally) size is...
   - 12 bits per page table ((64 - 16) / 4) = 12
   - Outer page table: 2^16 * 2^3 = 2^19 = 512 KB
   - 1 inner-most page table covers 2^12 addresses
     - Need 2^16 pages, so we need 2^16 / 2^12 = 4 inner-most page tables
   - Only need 1 second-inner-most page table (and therefore all others)
   - Need a total of 19 page tables
     - Each page table is 2^12 * 2^3 = 2^15 = 32KB
   - Total number of bytes is 19 * 32 KB = 608 KB

Choosing the Page Size

1. Larger pages
   - Smaller page table (1 per page and larger -> fewer pages)
2. Smaller pages
   - Larger page table
3. Internal fragmentation: Application requests some amount of memory, but we can only give memory in units of pages
   - This is why we shouldn't use huge pages
4. Compromise: Somewhere between a few KB and a few MB
   - Designers were worried about wasting memory, leading to the 4KB standard
   - A processor nowadays has more memory, so a larger page is more plausible

Memory Access Time With V-P Translation

1. For load/store:
   - Compute the virtual address (fast; addition of two numbers)
   - Compute the page number (fast; indexing using virtual address)
   - Compute physical address of page table entry (fast; addition)
   - Read page table entry (this can be large, so it will be in memory)
   - Compute physical address (fast; combine offset with frame number)
2. The issue with this process is that reading the page table results in going to memory
   - This means that, even though we cache the results, we might still have to go to memory (slow), rendering the cache useless
   - For a multi-level page table, this will happen multiple times

V-P Translation Quiz 1

1. Consider a system with the following specifications:
   - 1 cycle to compute virtual address
   - 1 cycle to access cache
   - 10 cycles to access memory
   - 90% hit rate for data
   - Page table entries are not cached
2. How many cycles for a load if using a three-level page table?
   - 1 for VA
   - 10 to access outer-most page table
   - 10 to access next page table

- 10 to access inner-most page table
- 1 to access cache
- 10% of 10 cycles to account for cache misses
- Total = 33 cycles (30 of which are page table accesses)

V-P Translation Quiz 2

1. Consider a system with the following specifications:
   - 1 cycle to compute virtual address
   - 1 cycle to access cache
   - 10 cycles to access memory
   - 90% hit rate for data
   - Page table entries are cached and have same hit rate as data
2. How many cycles for a load if using a three-level page table?
   - 1 for VA
   - 1 to access cache + 0.1 * 10 for cache misses in outer page table (2)
   - 1 to access cache + 0.1 * 10 for cache misses in outer page table (2)
   - 1 to access cache + 0.1 * 10 for cache misses in outer page table (2)
   - 1 to access cache + 0.1 * 10 for cache misses in data cache (2)
   - Total = 9 cycles (6 of which are page table accesses)

Translation Lookaside Buffer (TLB)

1. TLB is a cache for translations
2. Cache is relatively large; could fit many translations
3. TLB is small -> extremely fast (faster than using data cache)
   - 16 kB -> 4 entries
4. Cache accessed for each level of page table
   - TLB stores only the final translation
   - In a cache for a 4-level page table, it would take 4 cycles
   - In a TLB, this is only one cycle
5. What if we have a TLB miss?
   - Perform translation using page table(s)
   - Put translation in TLB for future references

What If We Have A TLB Miss?

1. When a TLB miss occurs, who should be in charge of performing the translation and updating the TLB?
   - The operating system
   - The processor automatically reads page tables and updates TLB
2. Both are acceptable options
   - OS is called software TLB miss handling; allows OS to use any type of page table it wants; processor just maintains TLB
     - Could be tree or hash page table
   - Processor is called hardware TLB miss handling
     - More hardware, but faster
     - Most high-performance processors use this approach, but some embedded processors using software-miss handling

TLB Size Quiz

1. Consider a system with the following specifications:
   - 32KB cache, 64B block size
   - 4KB page size

2. How many TLB entries do we need if we want similar miss rates in cache and TLB?
    - 8 to 512 entries (correct)
    - 64 to 32768 entries
    - Fewer than 64 entries
    - Greater than 256 entries
3. Solution:
    - TLB needs to cover the same amount of memory that the cache does
        - 32 / 4 = 8 entries
    - However, the 64B block can fit 512 different blocks which are spread throughout memory (not necessarily aligned)
        - To account for this amount of memory, we need up to 512 entries

TLB Organization

1. Associativity?
    - Small -> fast -> fully (or highly) associative
        - Direct mapped cache would sacrifice hit rate for speed, but we don't need that
2. Size
    - Want hits as often as our cache -> 64..512 entries
    - If we need more space, create a two-level TLB
        - L1: Small and fast
        - L2: Hit time several cycles, but large

TLB Performance Quiz

1. Consider a program with a 1MB array, read one byte at a time from start to end. This is repeated 10 times
    - No other memory is accessed
    - 4KB page, 128-entry L1 TLB, 1024-entry L2 TLB
    - TLB is initially empty and the array is page-aligned
2. Track the hits and misses for L1 and L2 TLB
    - L1 TLB hits: 10 * 4095 * 256
    - L1 TLB misses: 10 * 256
    - L2 TLB hits: 9 * 256
    - L2 TLB misses: 256
3. Solution:
    - 1MB = 2^20 / 2^12 = 2^8 (256) pages
    - In the first pass, we will have 256 L1 TLB misses (one per page) and (4096 - 1) * 256 TLB hits
    - After one pass, the L1 TLB will have entries for the latter half
        - The L2 TLB will have all of the data
    - In the first pass, the L2 TLB has 256 TLB misses and no hits
    - In each of the subsequent 9 passes, the L2 TLB has 100% hits

Conclusion

1. Understanding virtual memory is important for understand advanced cache optimizations