VIRTUAL ADDRESSES --> PHYSICAL ADDRESSES

Before we start a discussion of virtual memory and virtual addressing, let's first review the machine instructions cycle and the machine pipeline to recall our understanding of communication between the CPU and Main Memory (RAM)

Typical Machine-Instruction Cycle for PIPELINE

Dr. Joseph Wunderlich

Phase 1: “FETCH”

CPU puts address of the “next” machine instruction onto the Address bus
CPU sends “READ” signal to memory (cache’s first, then main memory)
Instruction read into CPU via data bus from first memory where it is found (i.e., L1 cache, or L2 cache, or main memory)
If it is not located in any of them, this is a “Page Fault” and a new page must be put into main memory from disk storage

Phase 2: “DECODE”

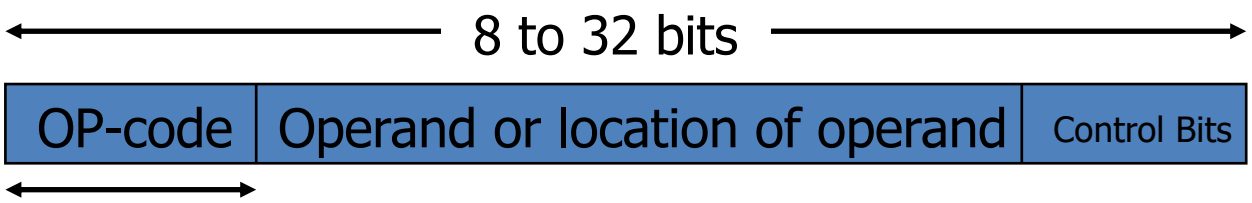
CPU decodes instruction put into it’s
Instruction Register during the “FETCH

Machine Instructions have two main parts:

1. **OP-Code:** Identifies which instruction to execute
2. **Operand:** Data to be used during execution

Phase 2: "DECODE" continued

Typical Instruction Format:



5 to 11 bits in OP-Code
Therefore 2^5=32 to 2^11=2048 different machine instructions in "**Instruction Set**"

- Some simple Microcontrollers (e.g., PIC's) have only 32 instructions
- Some large-scale machines (e.g., IBM S/390) have close to 2000 instructions

Phase 2: “DECODE” continued

Location of operand:

1. “IMMEDIATE:” Data encoded into machine instruction (Fastest to execute)
2. “MEMORY-REFERENCED:” Data located in memory at a location defined by address encoded into machine instruction (Slowest to execute)
3. “REGISTER-REFERENCED:” Data is located in an internal CPU register and it’s register number is encoded into machine instruction

Phase 3: "EXECUTE"

If necessary, read operand data from cache's or main memory:

1. CPU puts address of operand onto address bus
2. CPU exerts a "READ" signal
3. Data read into CPU via data bus from first memory where it is found (i.e., L1 cache, or L2 cache, or main memory)
4. If it is not located in any of them, this is a "Page Fault" and a new page must be put into main memory from disk storage

Many different types of data manipulations are carried out depending on the type of instruction (e.g., ADD, SUBTACT, MULTIPLY, MOVE, JUMP, etc.)

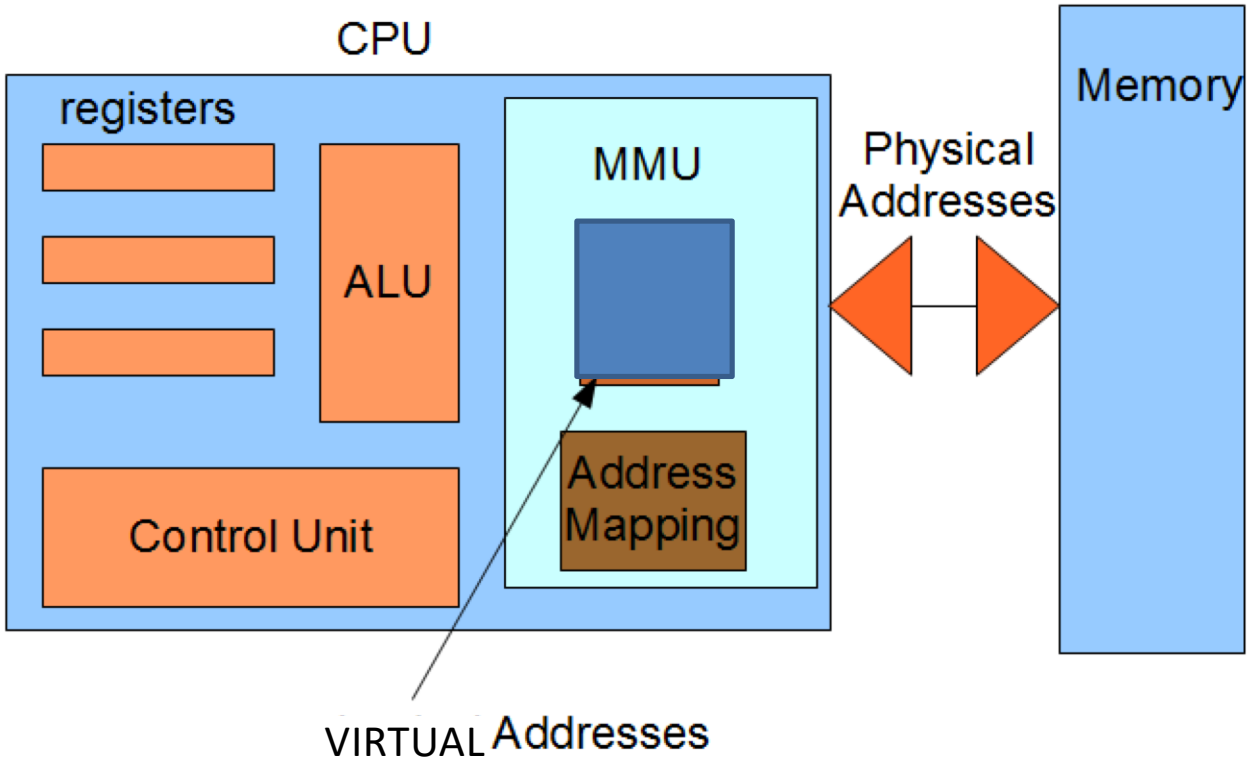
Phase 4: “WRITE-BACK”

This phase is only necessary for memory referenced instructions which write results back to memory:

1. CPU puts address onto address bus of where data is to be written to
2. CPU puts Data onto data bus
3. CPU exerts a “WRITE” signal
4. Data written into memory

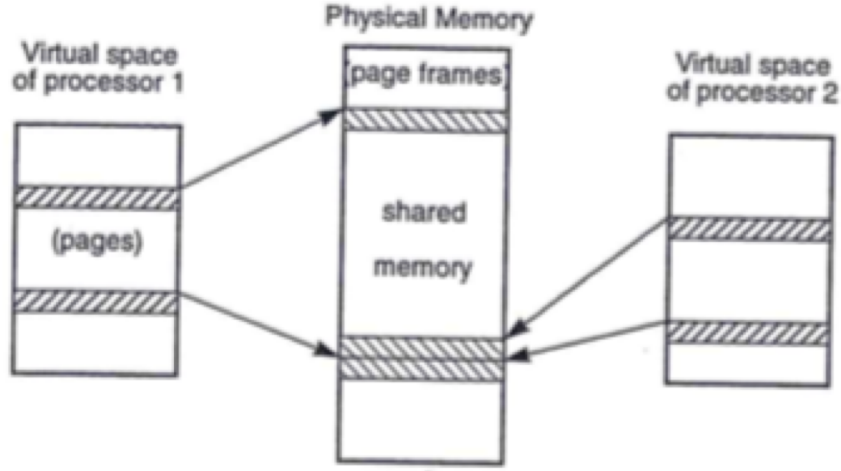
VIRTUAL ADDRESSES --> PHYSICAL ADDRESSES

The CPU, operating system, and all application programs use a **64-bit** address space (VIRTUAL addressing); but 2^64 is ~16,000,000,000 GigaBytes (i.e., 2^4 x 2^30 x 2^30) of memory which is much more than the Physical Memory of most computers. For example the motherboard of most PCs, and the number of address pins coming out of the processor is typically only 40 which corresponds to 2^4**0** = 1 TeraByte (i.e., 2^10 x 2^30 = 1000 GigaBytes)

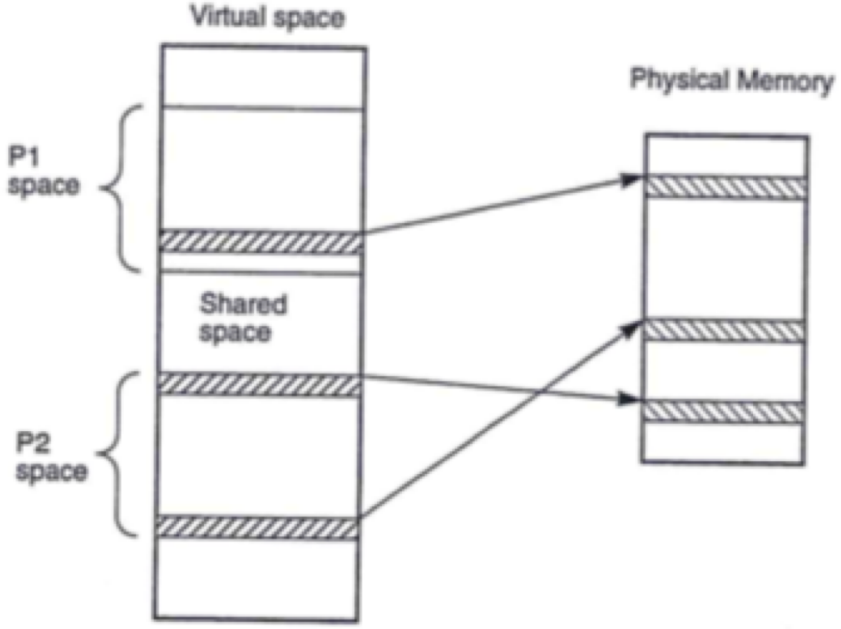


Before memory addresses are loaded on to the system bus, they are translated to physical addresses by the MMU.

When the CPU wants to read or write from **MAIN MEMORY** (RAM), it creates a 64-bit **VIRTUAL ADDRESS** that needs to be translated into a **PHYSICAL ADDRESS** in the **MEMORY MANGEMENT UNIT (MMU)**. The location of this address is then first looked for in the caches located between the CPU and Memory and if there is a cache miss of all those caches', then Memory is searched; however if it is not there, this is called a **PAGE FAULT** and a page of data & instructions is copied (by the MMU) from a storage device (**DISK** or **FLASH HARDDRIVE**) into memory.



(a) Private virtual memory spaces in different processors



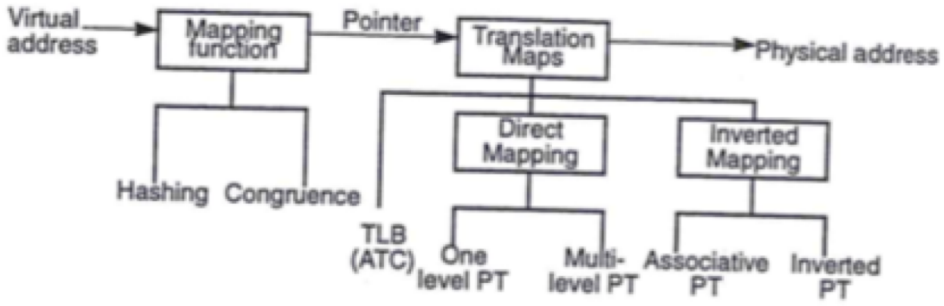
(b) Globally shared virtual memory space

Figure 4.20 Two virtual memory models for multiprocessor systems. (Courtesy o Dubois and Briggs, tutorial, *Annual Symposium on Computer Architecture*, 1990)

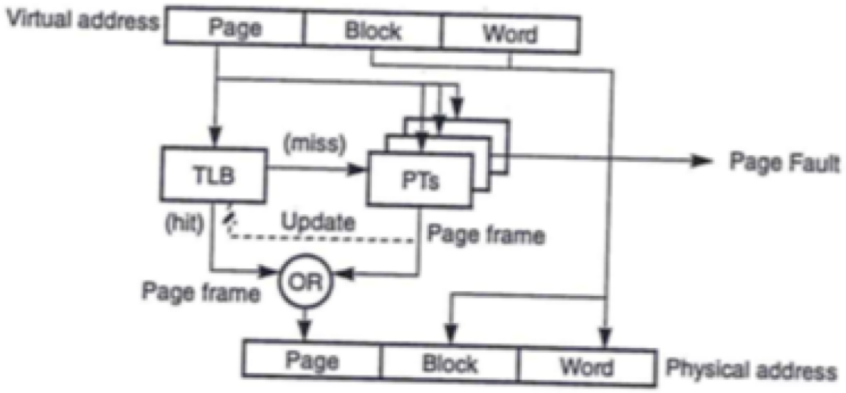
Translation maps are stored in the cache, in associative memory, or in the main memory. To access these maps, a mapping function is applied to the virtual address. This function generates a pointer to the desired translation map. This mapping can be implemented with a *hashing* or *congruence* function.

Hashing is a simple computer technique for converting a long page number into a short one with fewer bits. The hashing function should randomize the virtual page number and produce a unique hashed number to be used as the pointer. The congruence function provides hashing into a linked list.

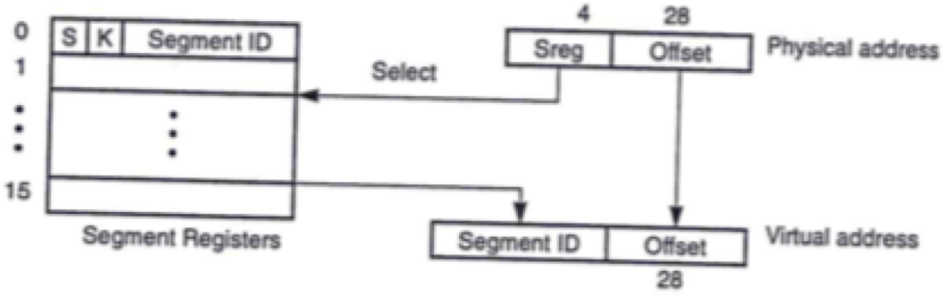
Translation Lookaside Buffer Translation maps appear in the form of a *translation*



(a) Virtual address translation schemes (PT = page table)



(b) Use of a TLB and PTs for address translation



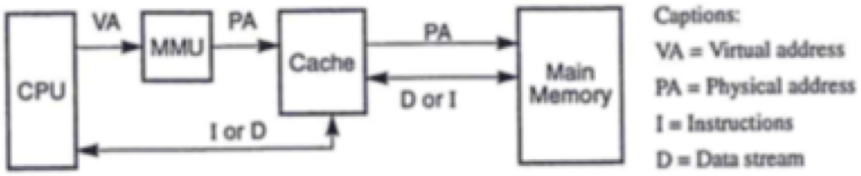
(c) Inverted address mapping

Figure 4.21 Address translation mechanisms using a TLB and various forms of page tables.

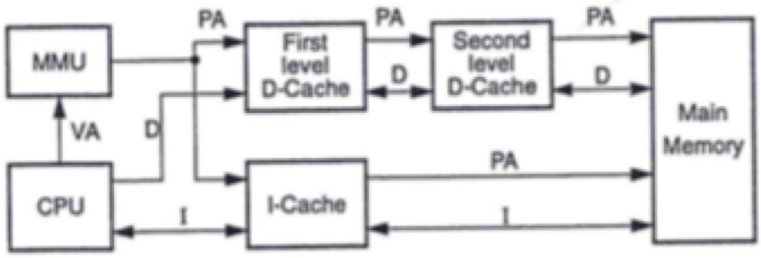
lookaside buffer (TLB) and *page tables* (PTs). Based on the principle of locality in memory references, a particular *working set* of pages is referenced within a given context or time window.

The TLB is a high-speed lookup table which stores the most recently or likely referenced page entries. A *page entry* consists of essentially a (virtual page number, page frame number) pair. It is hoped that pages belonging to the same working set will be directly translated using the TLB entries.

The use of a TLB and PTs for address translation is shown in Fig 4.21b. Each



(a) A unified cache accessed by physical address



(b) Split caches accessed by physical address in the Silicon Graphics workstation

Figure 5.8 Physical address models for unified and split caches.

Example 5.2 Cache design in a Silicon Graphics workstation

Figure 5.8b demonstrates the split cache design using the MIPS R3000 CPU in the Silicon Graphics 4-D Series workstation. Both data cache and instruction cache are accessed with a physical address issued from the on-chip MMU. A two-level data cache is implemented in this design.

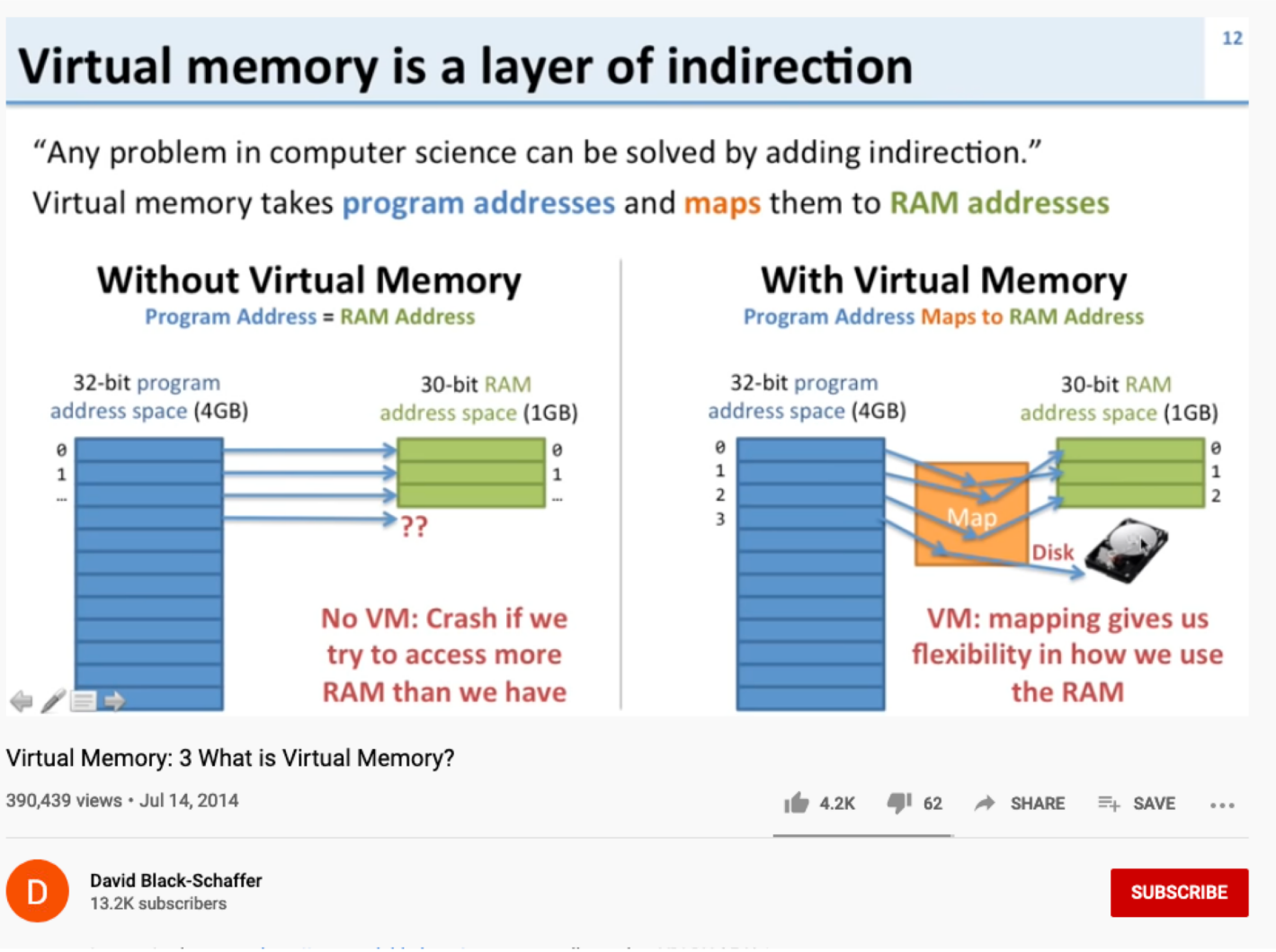
The first level uses 64 Kbytes of WT D-cache. The second level uses 256 Kbytes of WB D-cache. The single-level I-cache is 64 Kbytes. By the inclusion property, the first-level cache is always a subset of the second-level cache. Most manufacturers put the first-level caches on the processor chip and leave the second-level caches as options on the processor board. Hardware consistency needs to be enforced between the two cache levels.

■

The major advantages of physical address caches include no need to perform cache flushing, no aliasing problems, and thus fewer cache bugs in the OS kernels. The shortcoming is the slowdown in accessing the cache until the MMU/TLB finishes translating the address. This motivates the use of a virtual address cache. Integration of the MMU and caches on the same VLSI chip can alleviate some of these problems.

Most conventional system designs use a physical address cache because of its simplicity and because it requires little intervention from the OS kernel. When physical address caches are used in a UNIX environment, no flushing of data caches is needed if bus watching is provided to monitor the system bus for DMA requests from I/O devices or from other CPUs. Otherwise, the cache must be flushed for every I/O without proper bus watching.

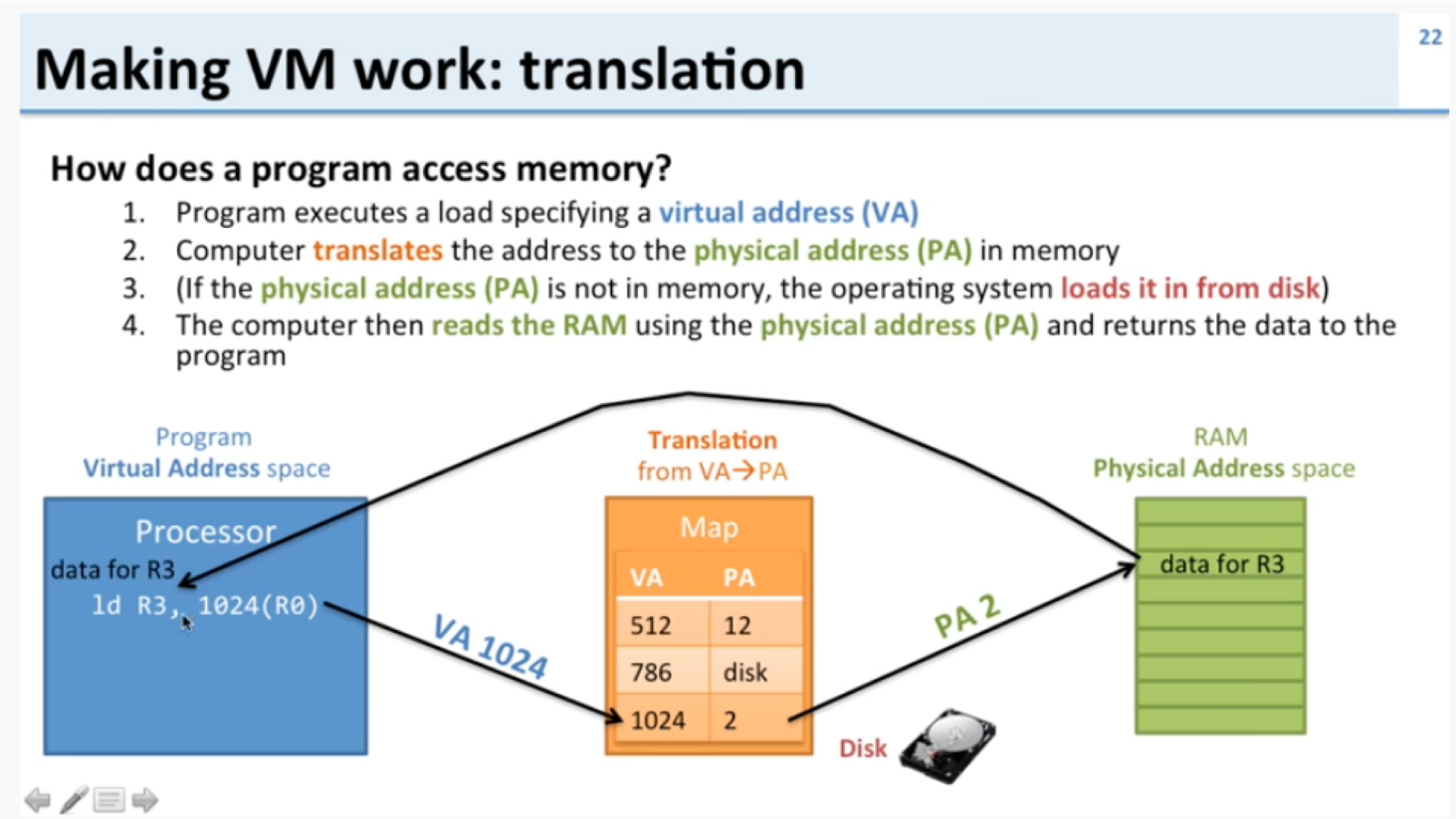
Watch this video:
https://www.youtube.com/watch?v=qlH4-oHnBb8



VIRTUAL ADDRESSES --> PHYSICAL ADDRESSES

Watch this Video:

https://www.youtube.com/watch?v=59rEMnKWoS4



Virtual Memory: 4 How Does Virtual Memory Work?

203,098 views • Jul 14, 2014 1.7K 21 SHARE SAVE

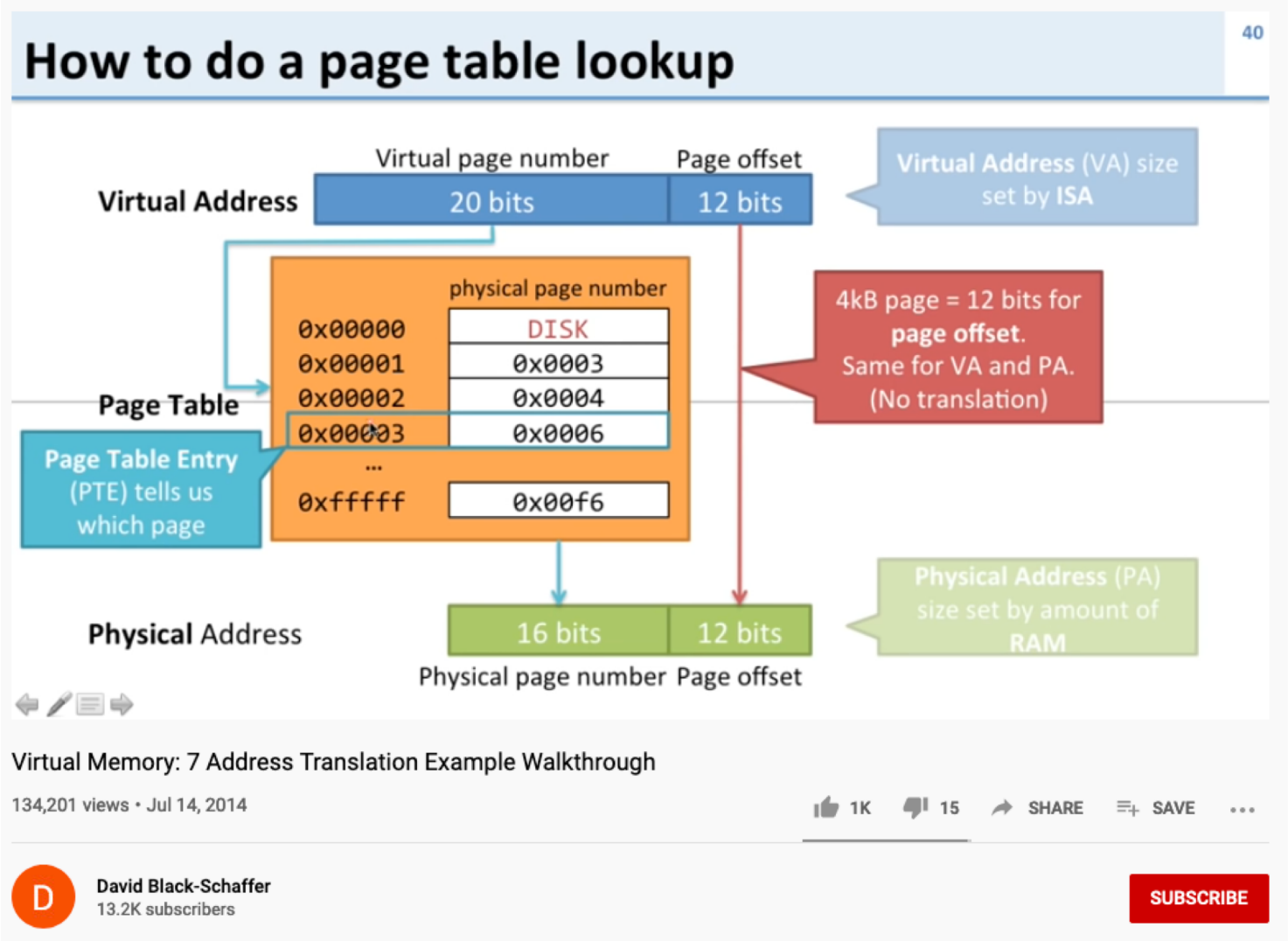
David Black-Schaffer
13.2K subscribers

SUBSCRIBE

VIRTUAL ADDRESSES --> PHYSICAL ADDRESSES

Watch this Video:

https://www.youtube.com/watch?v=6neHHkI0Z0o



Virtual Memory: 7 Address Translation Example Walkthrough

134,201 views • Jul 14, 2014 1K 15 SHARE SAVE

David Black-Schaffer
13.2K subscribers

SUBSCRIBE