SoC Architecture : Lecture 7

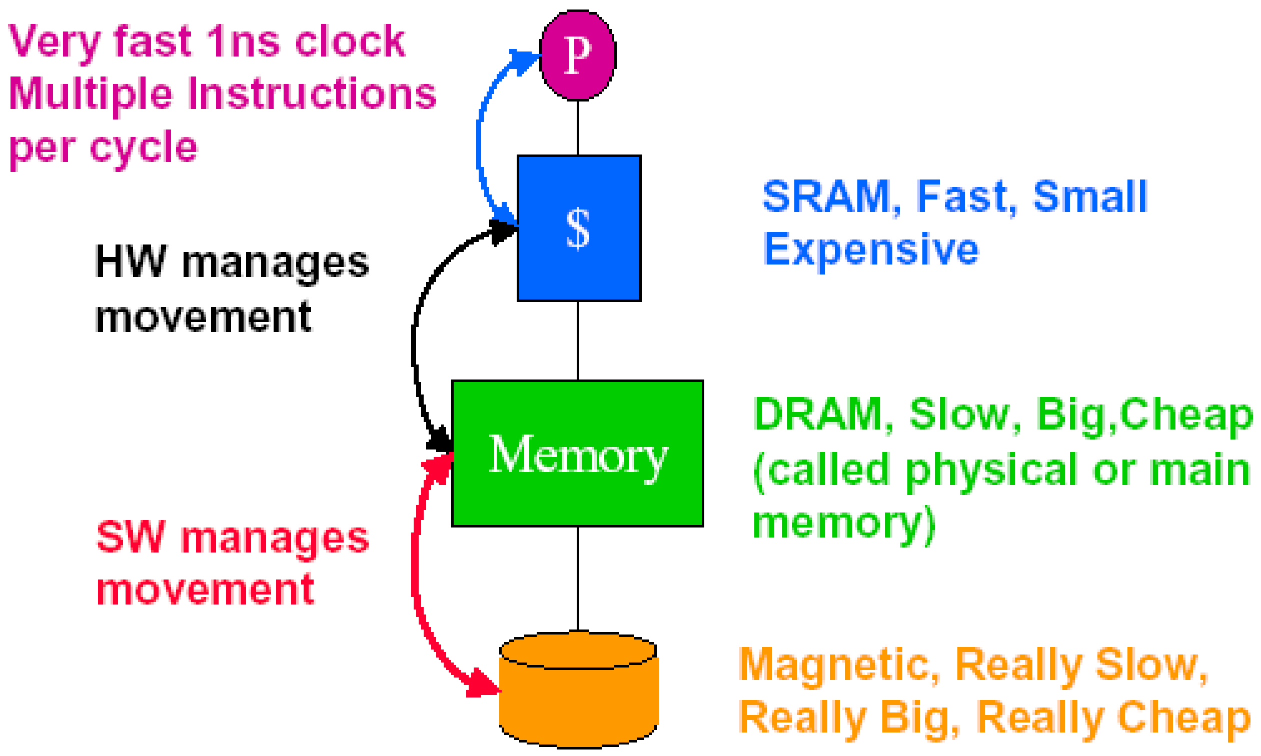
(Virtual Memory)

Prof. Jong-Myon Kim

School of Electrical Engineering

Extending the Memory Hierarchy

We know about registers, cache and RAM.
We can add a hard drive for virtual memory.
The HD will be really slow, really big and really cheap.

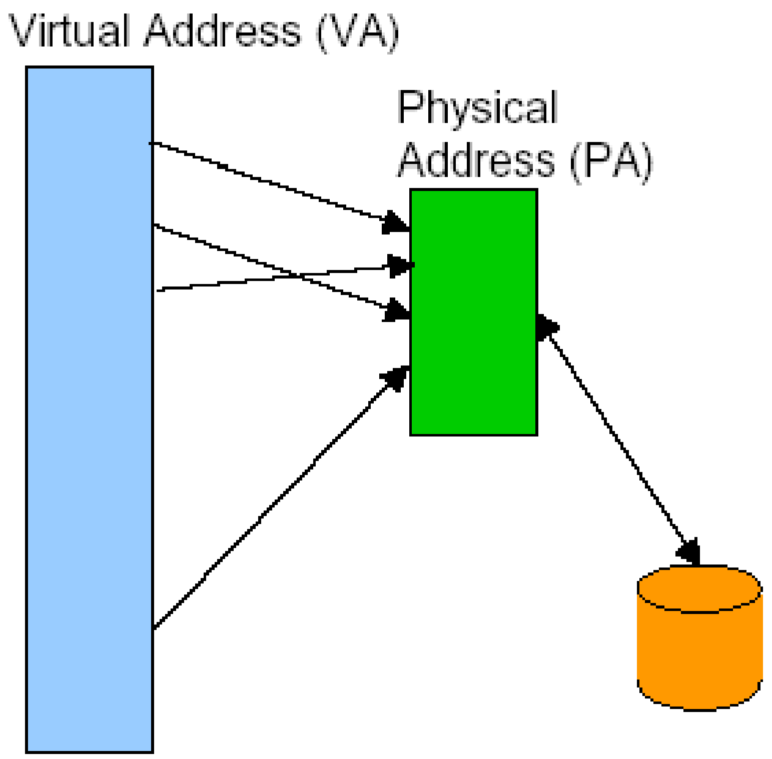

Very fast 1ns clock
Multiple Instructions per cycle
P
$
SRAM, Fast, Small Expensive
HW manages movement
Memory
DRAM, Slow, Big,Cheap (called physical or main memory)
SW manages movement
Magnetic, Really Slow, Really Big, Really Cheap

Why do we need Virtual Memory?

- In our programs, we have addresses of 32 bits. That means there can be about 4 billion addresses we can use.
- If each entry uses 8 bits (1 byte), then we would need about 4 GBs of memory to hold all the spaces.
- Most RAM and cache combined is not more than 300 MB, so what if we actually need all that space?
- <u>Virtual memory</u> uses the hard drive to save memory
- When we are using virtual memory, it will slow the computer down a lot. A lot of cycles will be wasted waiting for the data or instruction to come all the way from the hard drive.
- Virtual Memory is an important concept that not only gives us a lot more space, but also lets programs believe they have all the memory to themselves

Virtual Memory

- The processor will give a memory space (virtual addresses) to each program,
- The program will not know if the data is in physical memory or if it is in virtual memory
- There is a mechanism that can translate the address into either physical or virtual memory.
- The program using the memory doesn't care about where the memory actually is

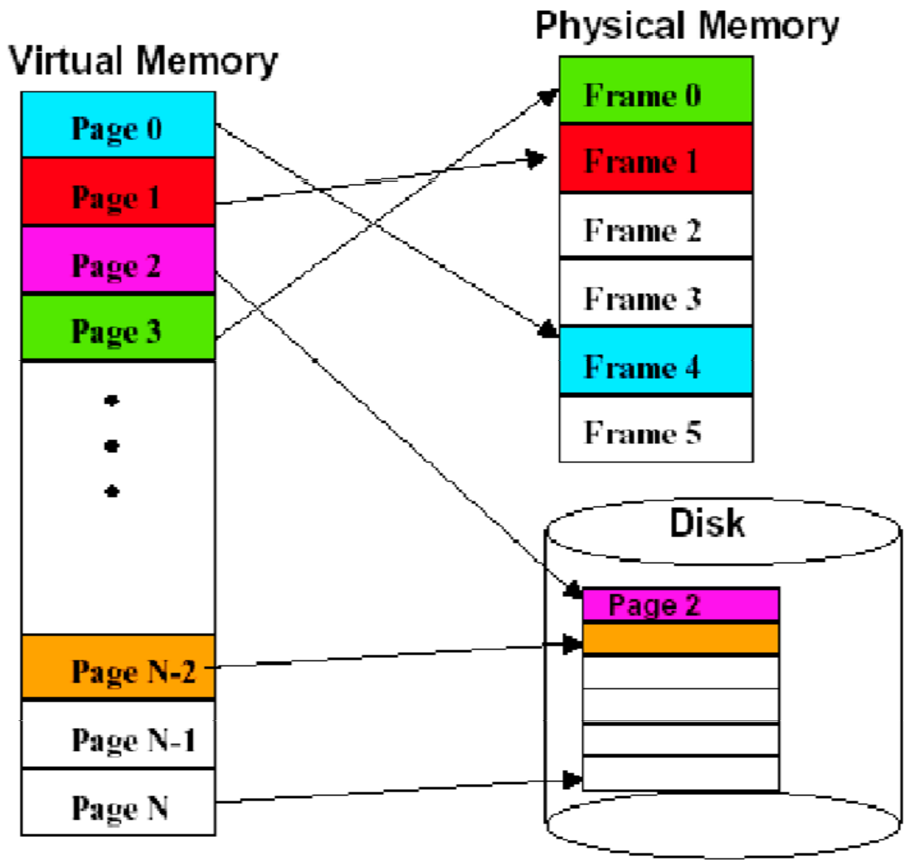


Virtual Memory

- We have always thought about virtual memory as memory that is used on a hard drive.
- Virtual memory, in operating systems, is really known as the memory that is given to a program so that it believes it has access to the full range of 32 bits of memory addresses
- The memory might be on the hard drive or the main memory. The program does not care where it is though, because it does not know where it is

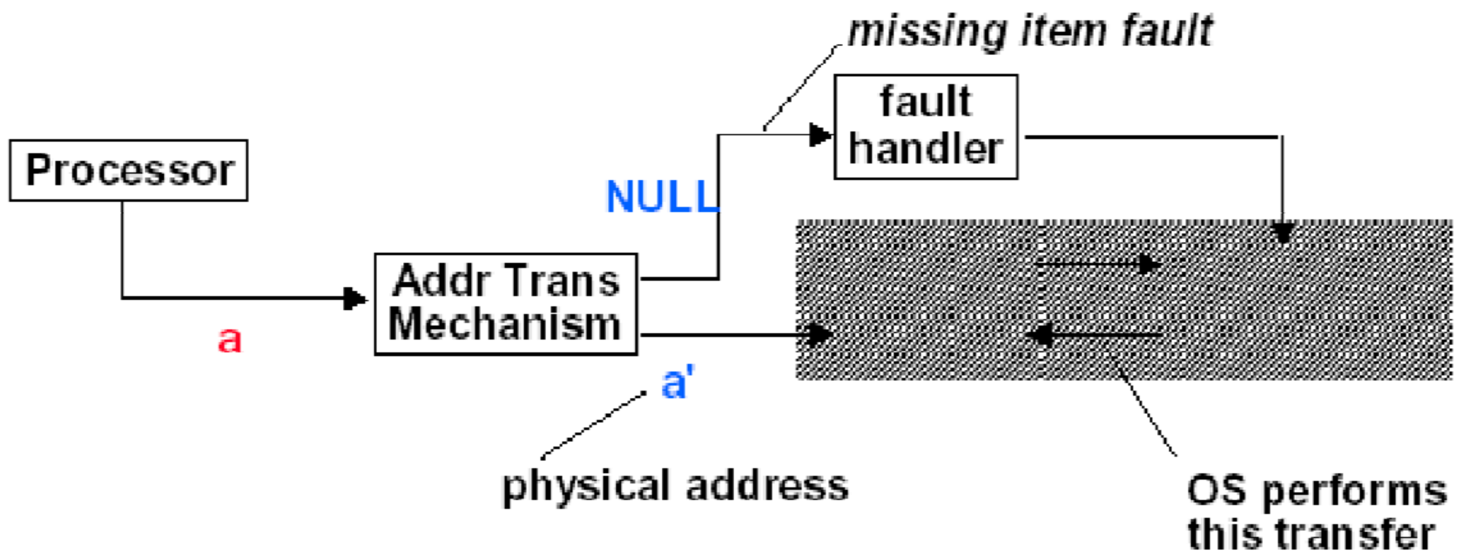
Pages

- The program will use the virtual address space and only know about things in memory called "pages."
- Each page will be connected to either a place in physical memory or virtual memory.
- There is a mechanism that can change a Page address to a physical frame address in RAM or in the HD



Address Mapping

- V = {0…n-1} – Virtual Address Space
- M = {0…m-1} – Physical address space
  - n > m (there are more virtual addresses than physical)
- MAP function: V → M to get the real address



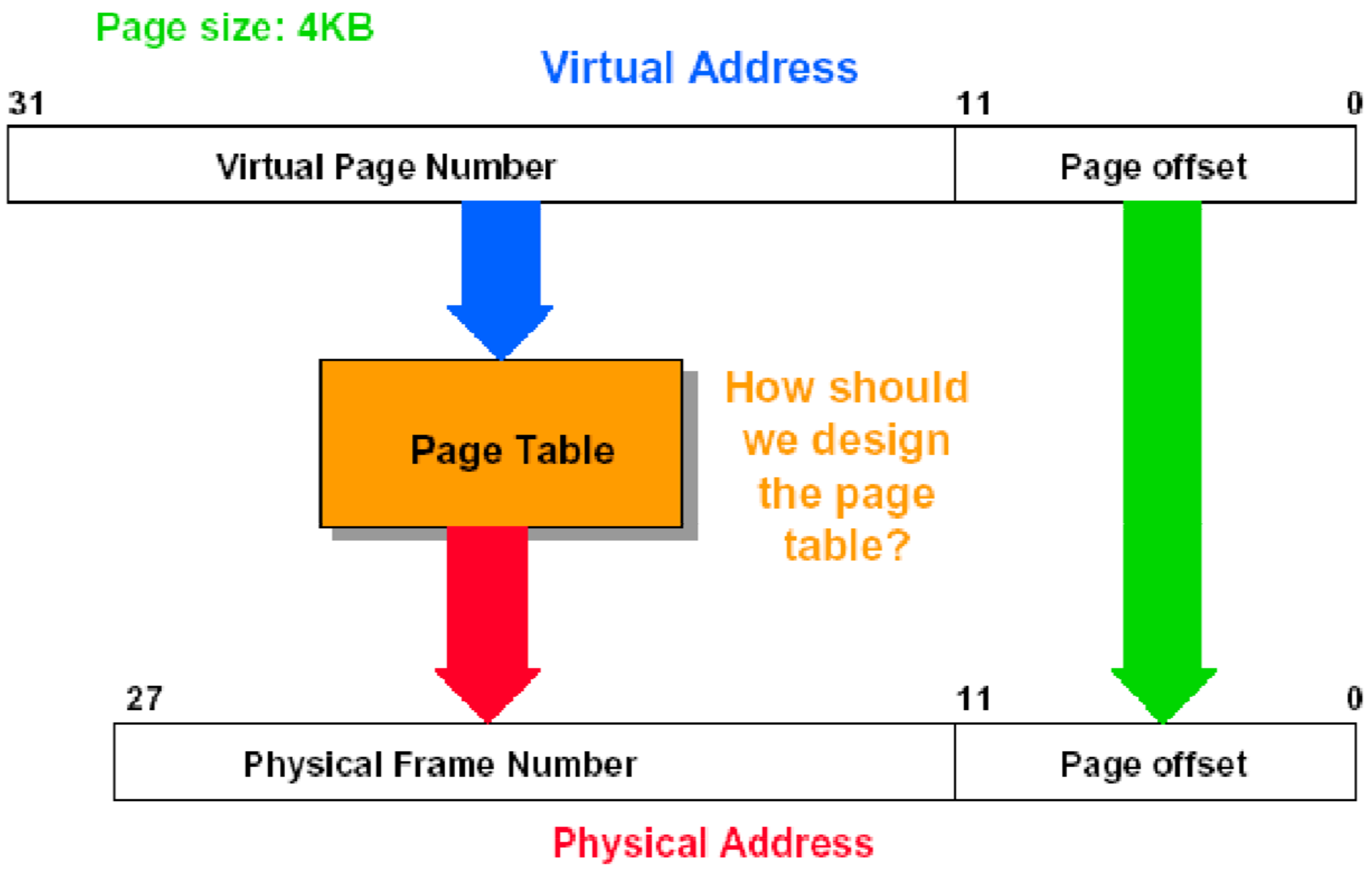
Page Table

- **The operating system will keep a separate Page Table for each program that runs. The program has only a small part of the memory and it must share with other running programs**
- **A Page Table Entry helps keep important information:**
  - Virtual to Physical mapping
  - Valid Bit – make sure the data is good
  - Access Rights – the operating system will say if the program is allowed to read, write, or delete a page of memory
- **There are many ways to design this Page Table**

Address Mapping Algorithm

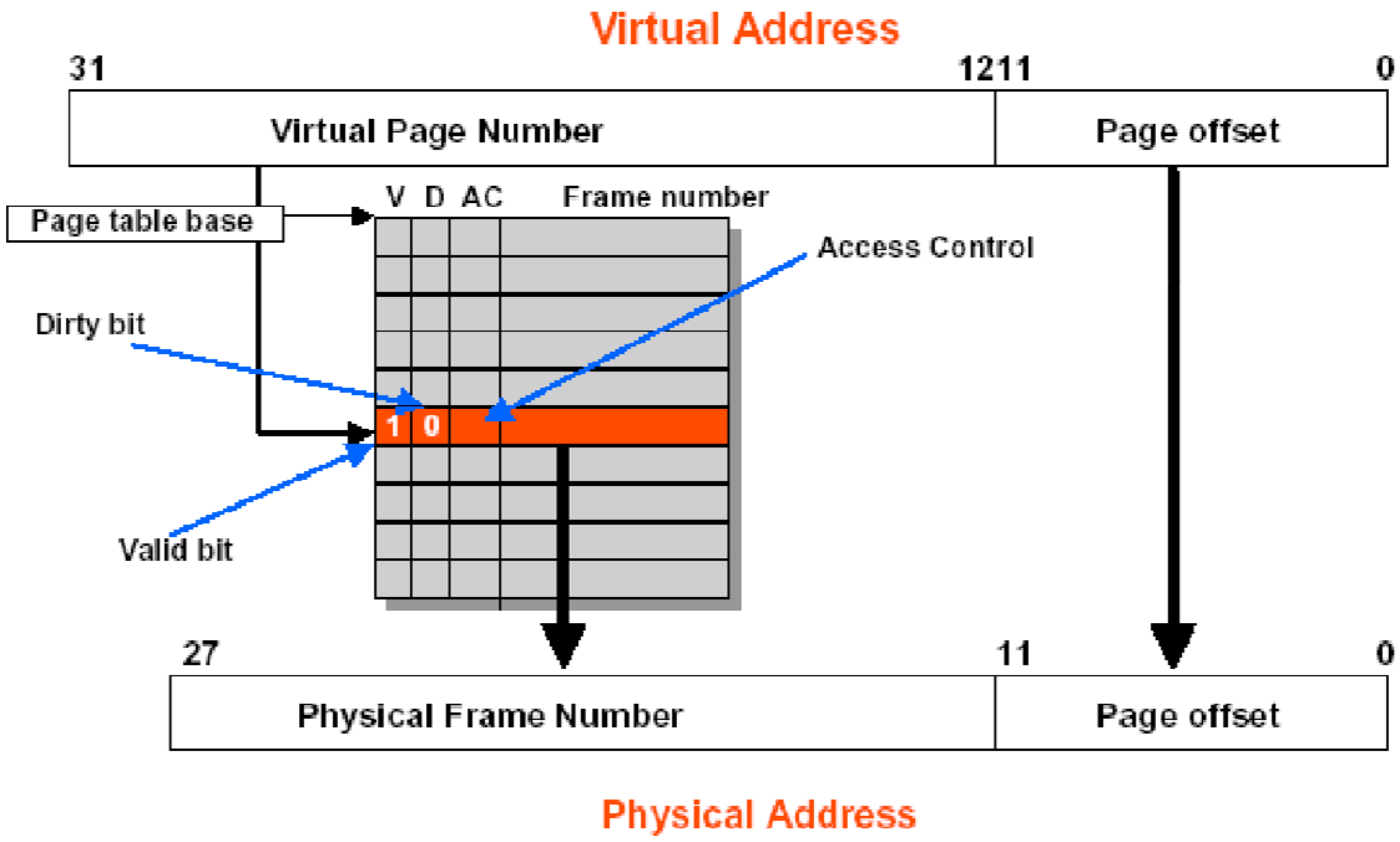
- **If (ValidBit == good)**
  - Then the data is in main memory. If it is not good, then the data is in secondary storage (hard drive)
- **If (ValidBit != good)**
  - Called a Page Fault because the data is not in main memory and the data must be search for on the hard drive.
  - The program will have to wait while the data is retrieved. (While the program waits, the operating system takes over to get the data. This is called a context switch)

Virtual to Physical Translation



Page Table Design: Linear Design

Simplest design. Virtual page number is an index right into the page table. Then it gets the real address



How do we choose a page size?

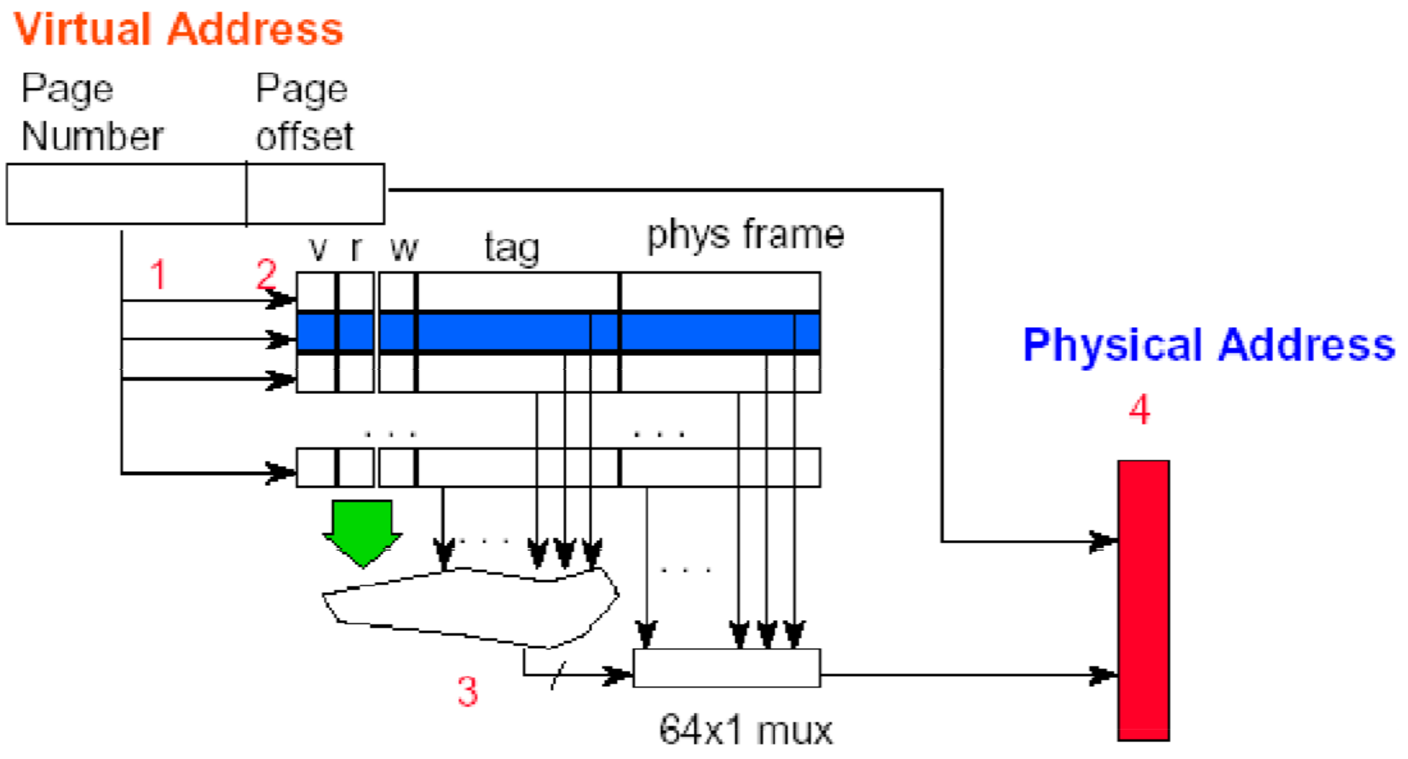
- **If page is too small:**
  - We have a lot of misses and a big page table
- **If page is too big:**
  - Fragmentation: we only use part of the page and don’t need the other memory inside because it's too large
  - Smaller page tables
- **As computers get faster, pages are increasing in size**
- **We also need a page replacement policy, which will be just like the cache replacement policy (least recently used)**
- **Important: Virtual memory is virtual, because it is managed by the operating system.**
- **Cache and physical memory is implemented in hardware**

Translation Lookaside Buffer

- We want a fast way to get the translated address.
- This can be accomplished if there is a "buffer" or a list of already translated addresses.
- This is called the TLB (translation lookaside buffer). It is like a cache for page table entries
- We want to make the common case very fast, and we want to use locality.
- If we use something often, it should be very fast to get to.
- The TLB helps us find the things we use often

Translation Buffer (fast translation)

- We can keep a cache of translated addresses.
- For example, if we have 64 translated addresses, we can increase the speed of our address translation by a lot

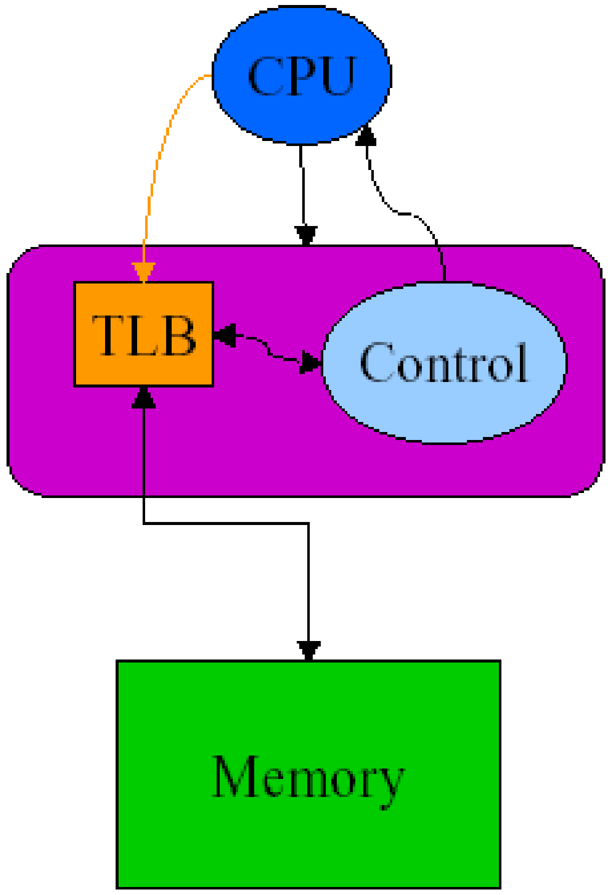


The TLB

- **What do we need to do to make it work well?**
  - Must be fast and not make the processor slow down
  - Should have high hit ratio
  - Must be able to change with physical memory (when something is removed, it should be updated)
  - It must allow for multiple programs. (Each program will have it’s own page table where it thinks it own all of memory)
  - Should only have a small number of entries so it can be fast

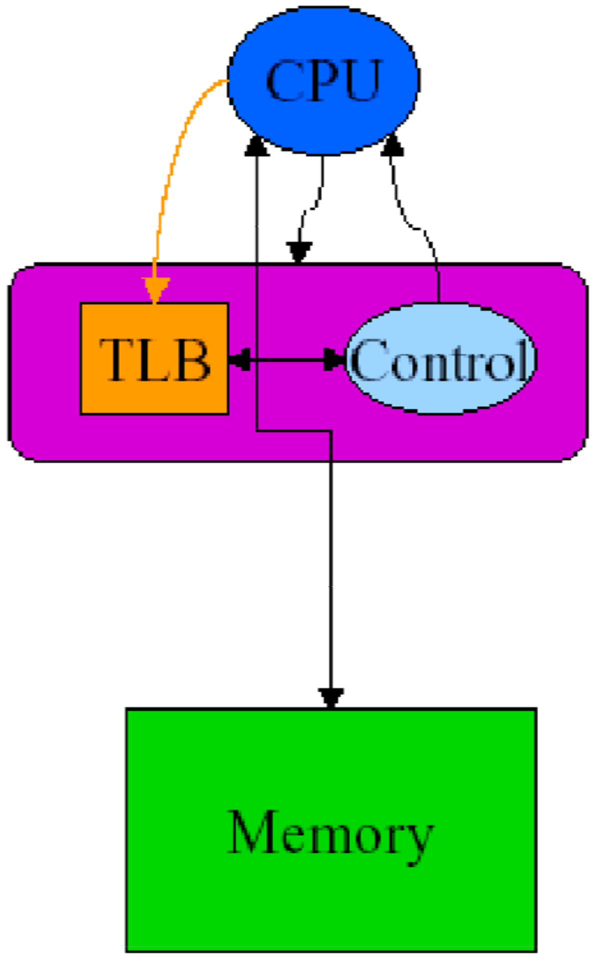
Hardware TLB

- The hardware will handle TLB misses and tell the page table how it should be organized.
- Lots of circuits are needed in the hardware to keep data up to date.
- Forces the page table to be one way and cannot improve if a new method is invented



Software TLB

- Software handles the misses.
- It will put in entries from the page table into the TLB.
- Allows for flexible page table.
- With hardware, the page table is built into hardware and cannot be changed.



Virtual Memory: Mini-Review

- **Provides illusion of large memory**
  - Sum of memory of all programs greater than physical memory
  - Address space of each program bigger than physical memory
- **Allows us to use main memory efficiently**
- **Uses memory hierarchy to try to make memory fast**
- **<u>Virtual Address</u> – address used by the program**
- **<u>Virtual Address Space</u> – all the addresses the program thinks it can use**
- **<u>Memory Address</u> – address in physical memory. It is the real address**

Paged Virtual Memory

- **We can divide memory into pages, blocks, and frames.**
- **They are different words but just mean the same thing, a way to divide up memory into separate pieces**
  - Pages are used in virtual memory
  - Frames are used for real memory
- **Pages can be mapped into frames in real memory (TLB)**
- **Pages can live anywhere: main memory or secondary memory (the hard disk)**

Paged Virtual Memory

- All programs are written to use the virtual memory space.
- Programs are not allowed to directly control main memory, they can only change virtual memory
- This is because it would be bad if one program could change another program’s memory.
- Each program can only see it's own memory.
- Hardware or software will do the translation “on-the-fly” which means it will translate the address when it is being used.
- We use a page table to translate between virtual and physical.
- A TLB helps make the translation go faster.

Summary

- Virtual memory does a few important tasks for a computer
  - It increases the size of memory far beyond what is actually in the RAM by using the hard disk
  - It lets each program think they have their own special world of memory.
  - Makes sure programs cannot access the memory of other running programs
- Virtual memory uses pages, page tables and TLBs in order to accomplish these tasks.

Virtual Memory Example 1

1. *Assume an Inverted Page Table (8-entry IPT) is used by a 32-bit OS. The memory page size is 2MB. The complete IPT content is shown below. The Physical Page Number (PPN) starts from 0 to 7 from the top of the table. There are three active processes, P1 (PID=1), P2 (PID=2) and P3 (PID=3) running in the system and the IPT holds the translation for the entire physical memory. Answer the following questions.*

Valid	Process ID (PID)	Virtual Page Number (VPN)
1	1	0x3fe
1	3	0x001
1	2	0x1ad
1	3	0x7fd
1	2	0x3fe
1	1	0x2bf
0	2	0x7fd
1	2	0x0bf

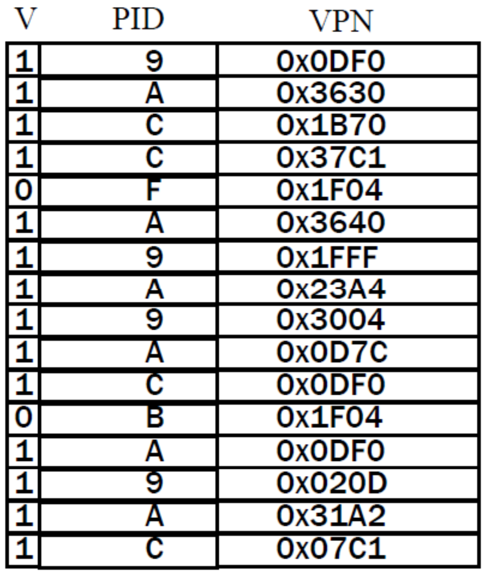
① *Based on the size of the Inverted Page Table above, what is the size (in MB) of the physical memory?*

② *To which "physical address" does the "virtual address" 0x7fdd8f64 of P2 map? Please derive and write down the complete address in Hex value. (if there is no valid mapping, please answer "page fault")*

③ *To which "virtual address", of "which process", does the physical address 0x78e968 map? Please derive and write down the complete address in Hex value. If you cannot find a valid mapping, please answer "address not found".*

Virtual Memory Example 2

1. *Given a 32-bit processor with 4 active processes being executed concurrently. Please answer the following questions. Show all the addresses of your answer in hex number. If a translation cannot be found, enter page fault.*
   ① *Assume an inverted page table (IPT) is used by the OS. The IPT is shown below (only Valid, PID and VPN are shown). Each page size is 4 MB. What "virtual address" of which "process" maps to the physical address "0x363055B"?*



V	PID	VPN
1	9	0x0DF0
1	A	0x3630
1	C	0x1B70
1	C	0x37C1
0	F	0x1F04
1	A	0x3640
1	9	0x1FFF
1	A	0x23A4
1	9	0x3004
1	A	0x0D7C
1	C	0x0DF0
0	B	0x1F04
1	A	0x0DF0
1	9	0x020D
1	A	0x31A2
1	C	0x07C1