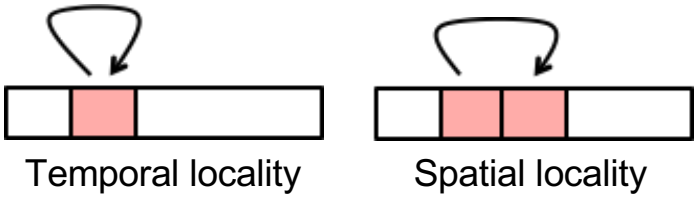
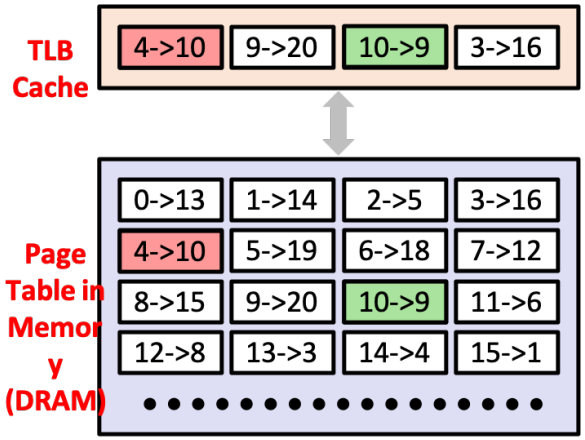
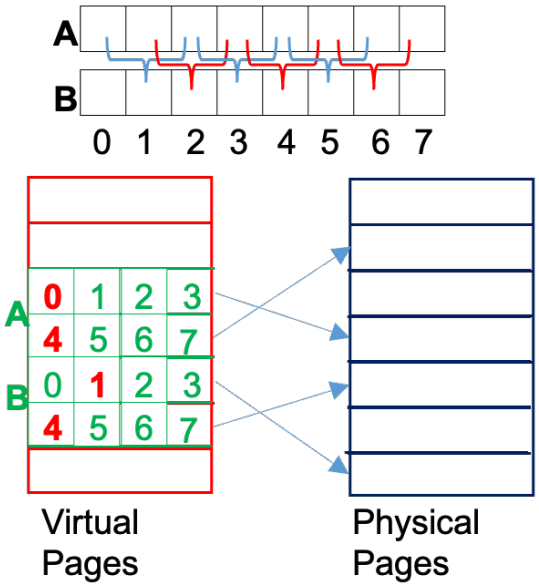
Lecture 19: Multi-Level Page Table

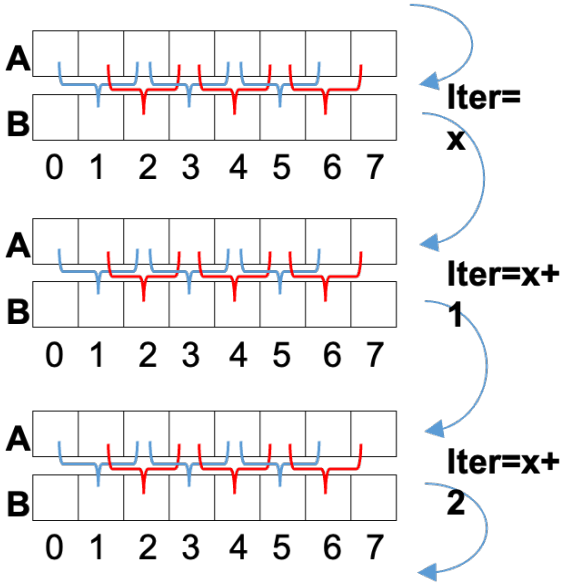
Vivek Kumar
Computer Science and Engineering
IIIT Delhi
vivekk@iiitd.ac.in

Last Lecture







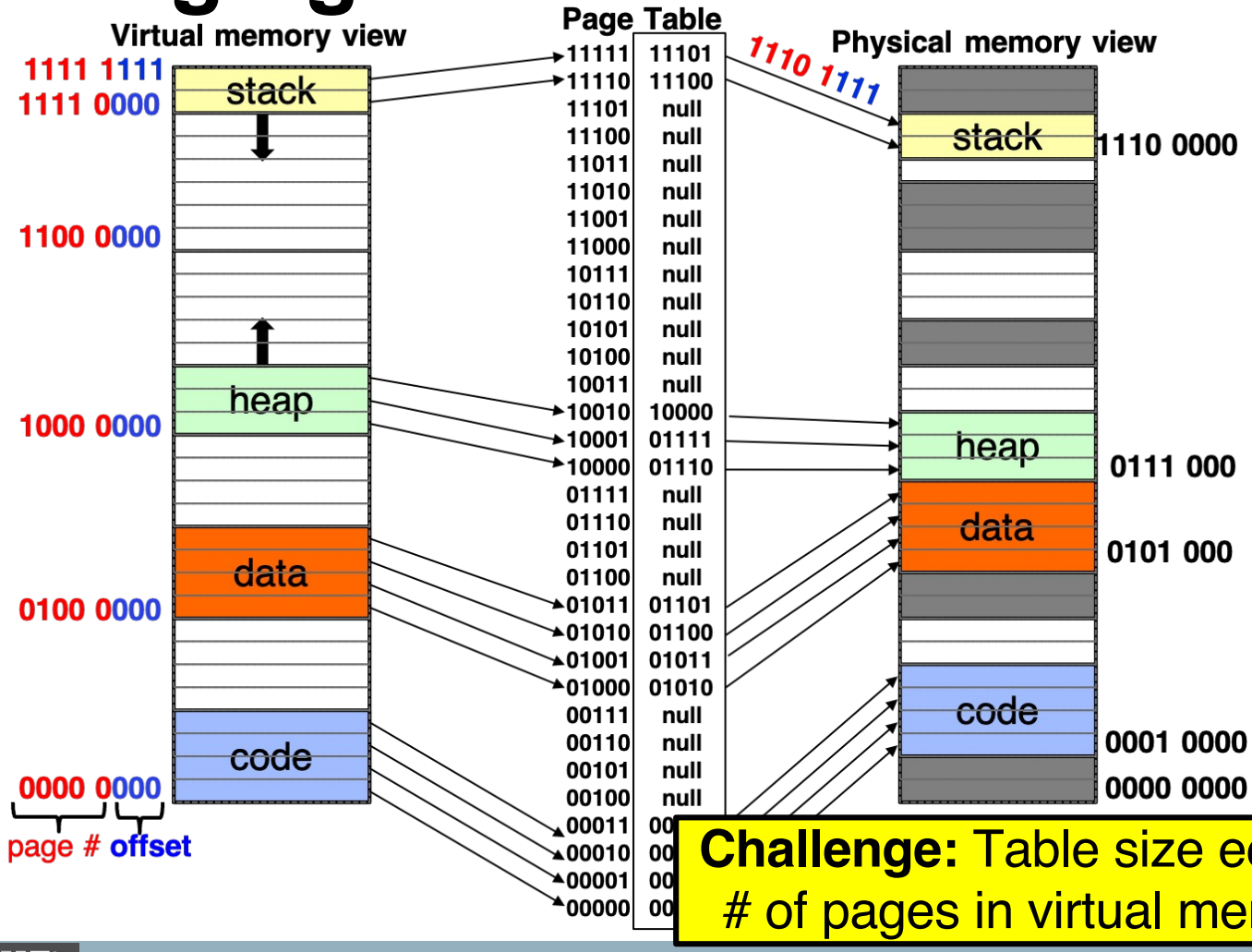


- Time overheads with page table access
  - PT base address in PTBR (1-cycle)
  - Accessing PT index in DRAM (several cycles)
- Translation look aside buffer (TLB)
  - hardware cache per CPU for saving frequently accessed PTE entries (part of MMU)
- Locality helps in caching
  - Temporal locality
  - Spatial locality

Today's Class

- Challenges with simple page table
- Multi-Level Page Table (MLPT)
- MLPT with TLB

Paging



Challenge: Table size equal to # of pages in virtual memory!

- Recall, process address space is not contiguous
  - Inter-segment memory is not contiguous
  - Only intra-segment virtual memory is contiguous (the physical memory may/may-not be contiguous)
- Page table until now is an **array** where the length of the array will be the total number of VPN being used in the process address space
  - Not all the VPNs will be used as the segments are not contiguous (there could be gaps between them)

Overheads with the Linear Page Table

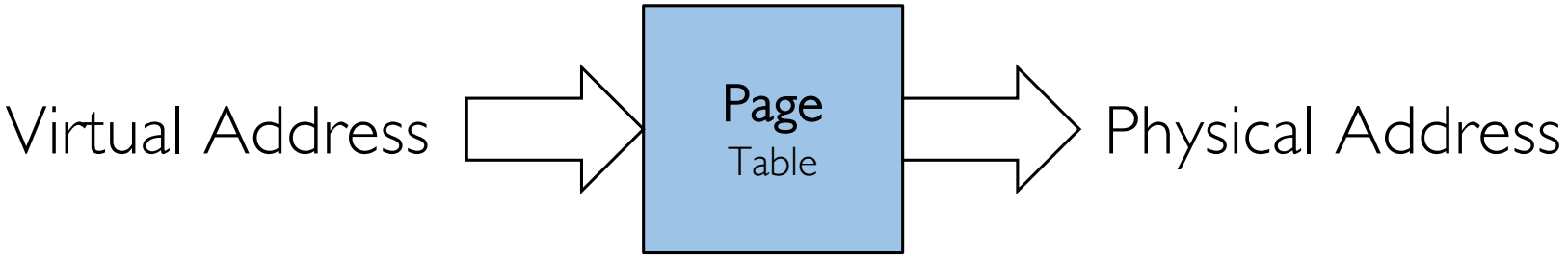
- **Time overhead**
  - Covered in last lecture

Discussion for today

- Space overhead (array based implementation)
  - Single entry in PTE on x86 (**32-bit**) = 4 bytes
  - Total number of pages = 2^{32}/4KB = $2^{32}/2^{12}$
  - Total number of unique VPNs = 2^{20}
  - Total memory for one page table = 2^{20} x 4 = **4MB**
  - If there are 100 processes, total memory requirement for page table = **400MB**!
  - On 64-bit: 2^{52} unique VPNs x 8 bytes = **36 Exa-bytes!**

Challenge: How to Structure a Page Table?

- Page Table is a *map* (function) from VPN to PPN



- Simple (linear) page table corresponds to a *very large* lookup table
  - VPN is index into table, each entry contains PPN
- What other map structures can you think of?
  - Trees?
  - Hash Tables?

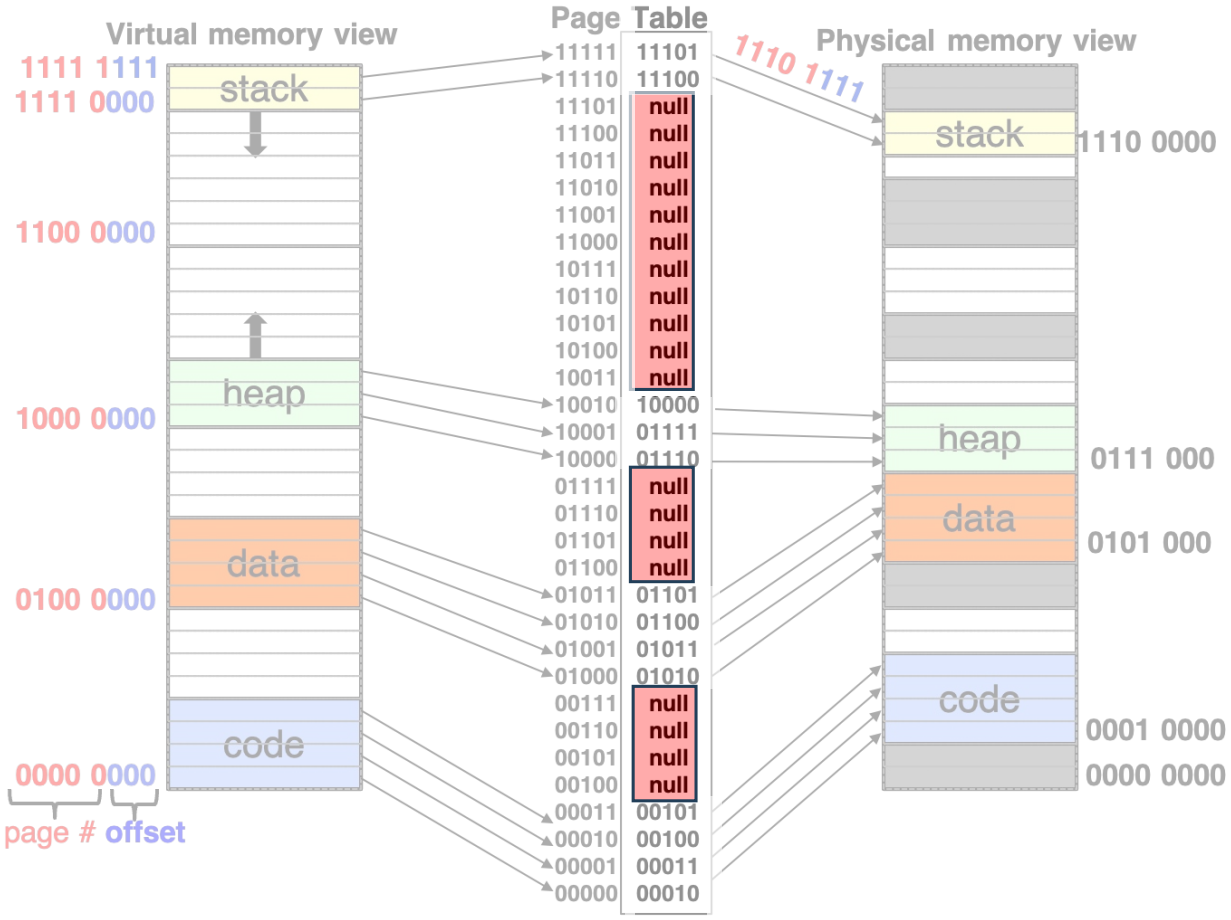
A Simple Fix

- How about using a page size bigger than 4KB?
- Let's try using a 16KB page size in 32-bit addressing
  - Total unique VPNs are $2^{32}/16KB = 2^{18}$
  - Space overhead
    - Single entry in PTE on x86 (32-bit) = 4 bytes
    - Total number of unique VPNs = 2^{18}
    - Total memory for one page table = 2^{18} x 4 = **1MB**
    - We are able to reduce single PT size by a factor of 4!
- Although we are able to alleviate the space overheads by increasing the page size, but do you see any issues in this approach?
  - Internal fragmentation!

Mapping Virtual to Physical Address

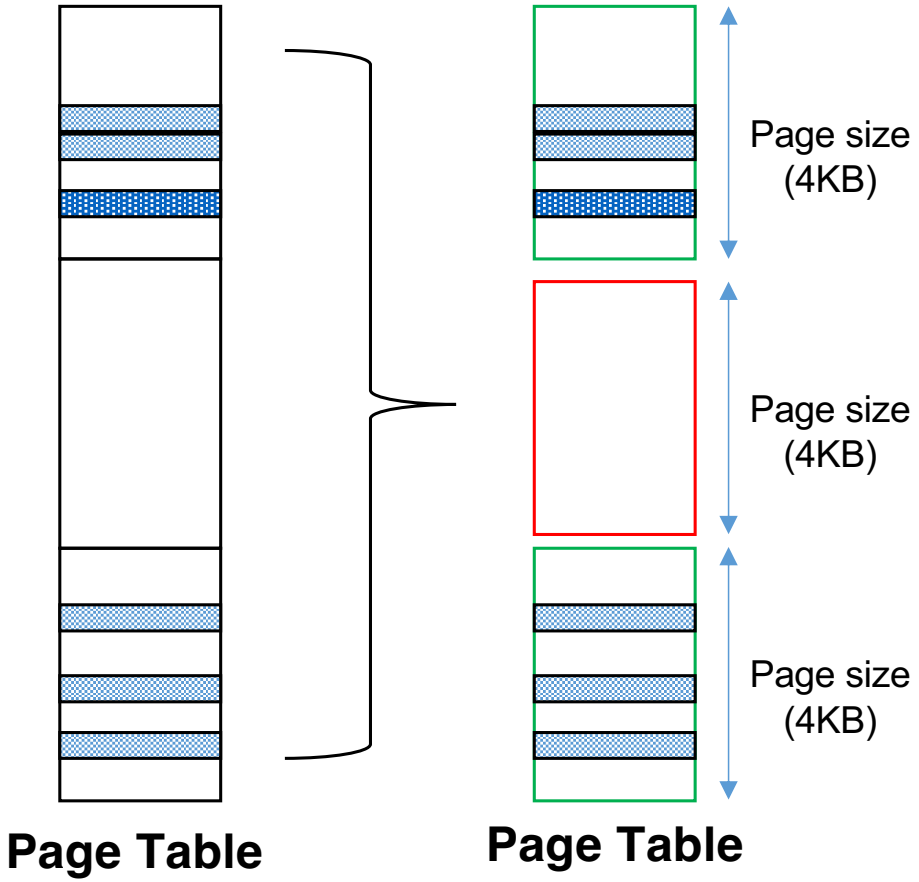
- Approaches
  1. Virtual address is same as the physical address
  2. Virtual address is not the same as physical address
     a) Using base & bound for VA to PA mapping
     b) Using segmentation for VA to PA mapping
     c) Paging
        a) Using Page Table (PT) in RAM and having a base register for PT
        b) Using Page Table (PT) in RAM and having a base register for PT as well as a TLB cache
        c) Using Multi-level Page Table

Reason for Large Page Table



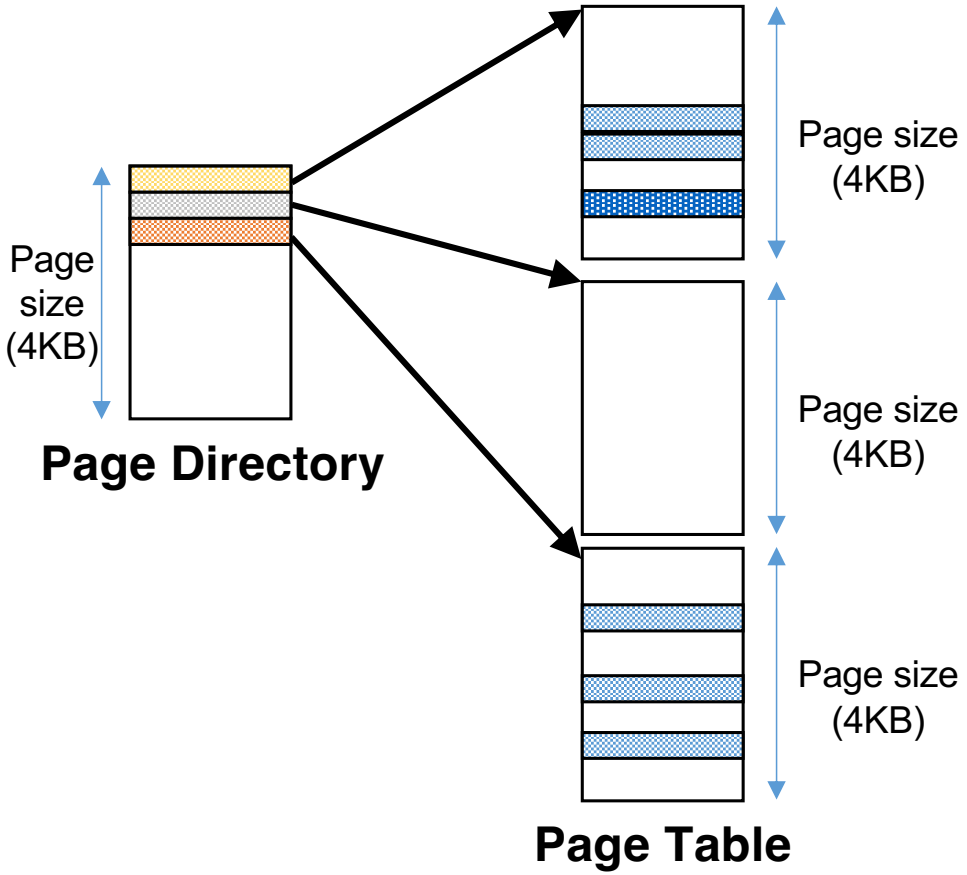
- There are several invalid ("null") entries in an array-based implementation of the page table
  - Not all indices are going to be valid in the look up table
- We can reduce the size of the page table if we can use a data structure that can help in removing the invalid "null" entries inside the page table
  - **Multi-Level Page Table (MLPT)**
  - We can also combine paging along with segmentation
    - We are skipping it in our discussion as anyway segmentation leads to complexity as we have already seen

Multi-Level Page Table (MLPT)



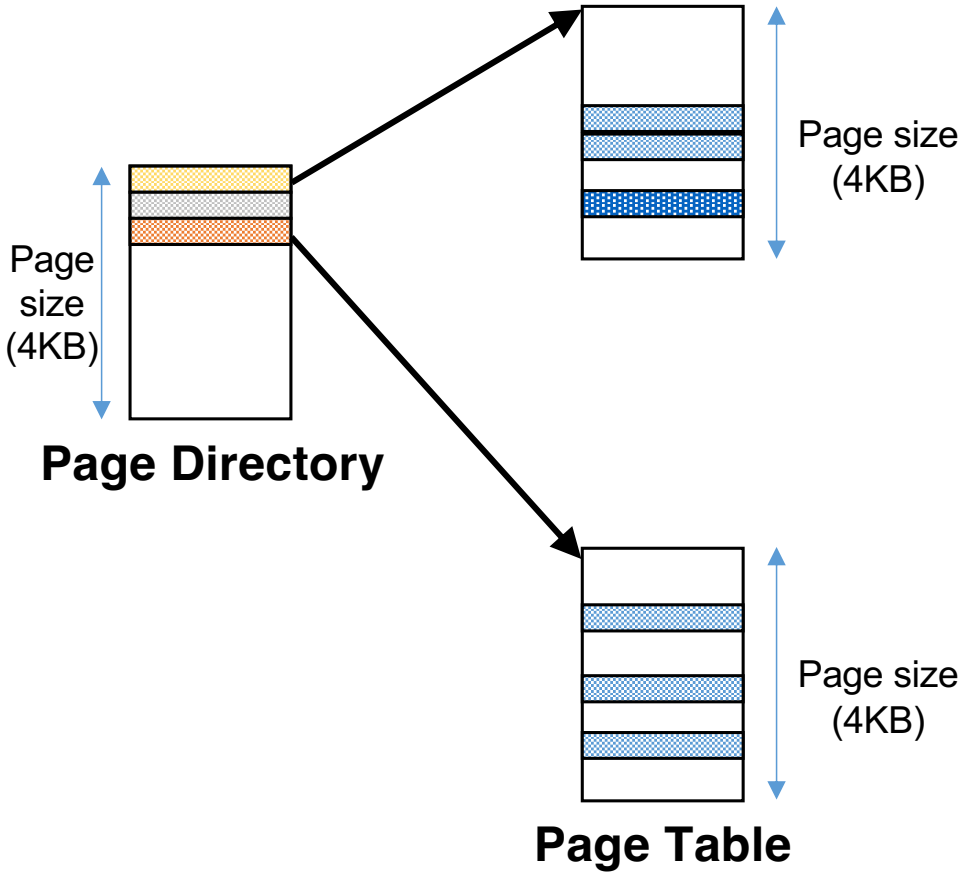
- Idea-1
  - Split the entire page table into page size entries
  - How many entries inside each page size entries?
    - On x86 with 4KB page size, there would be 1024 VPN entries inside each page size blocks (4KB/4bytes)
  - Types of “Pages”
    - All invalid (“null”) VPNs
    - At least one valid VPN
  - Guess what we can do next?

Multi-Level Page Table (MLPT)



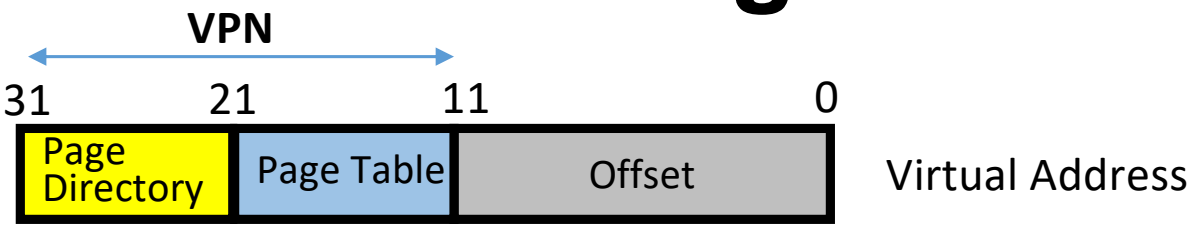
- Idea-2
  - Create a Page Directory (PD)
    - Per-process array-based data structure
    - Each index in PD points to a page of page table

Multi-Level Page Table (MLPT)



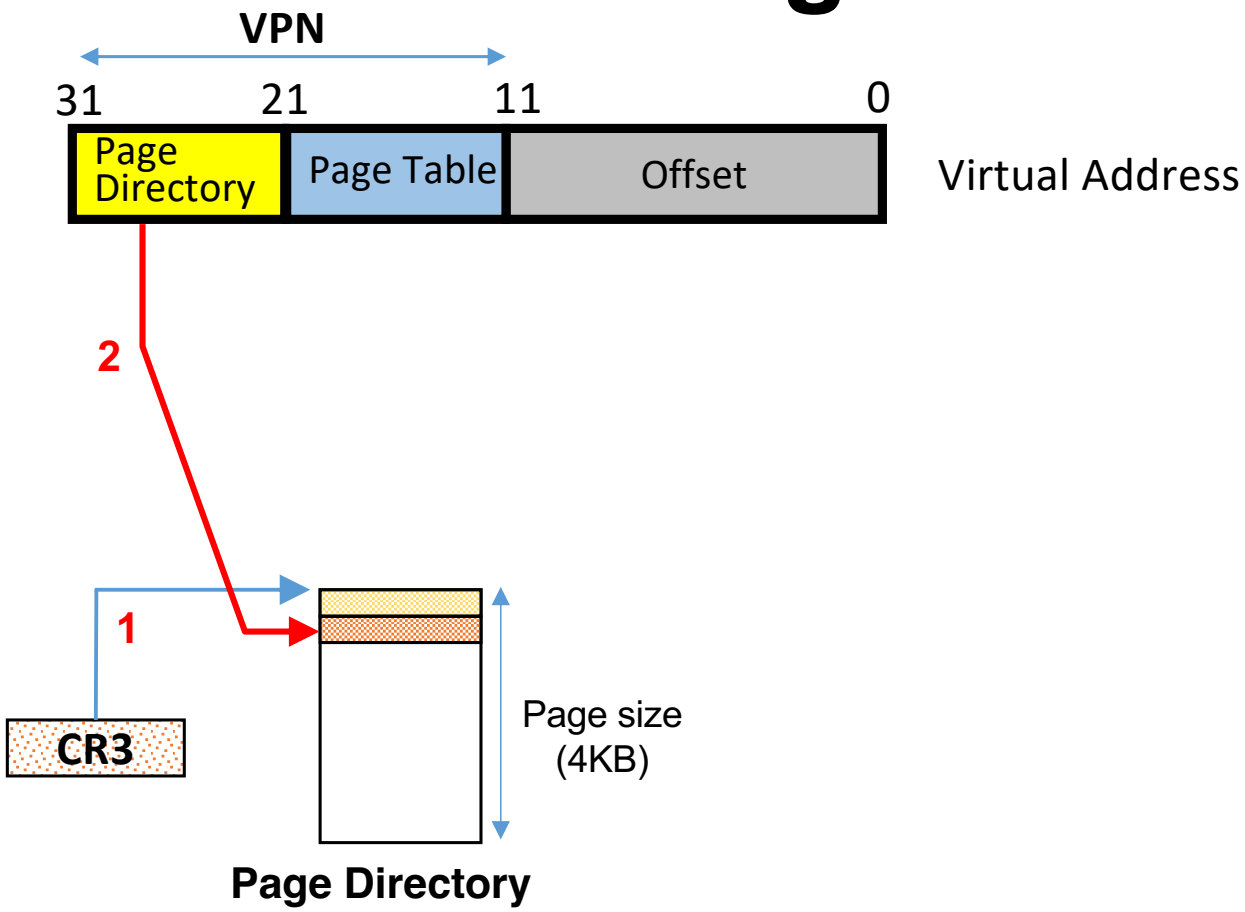
- Idea-3
  - Don't allocate a page of page table if there is not a single valid entry inside it
  - PD entry also stores this information apart from the base address of the page of a page table
- This example of MLPT is a two-level page table
- The unused page table in MLPT can easily be swapped to the disk
  - Only the page directory and the page table in use is needed to be on the physical memory
- There could be more than two-levels of page table!
  - Out of scope of this course

Two-Level Page Table in x86



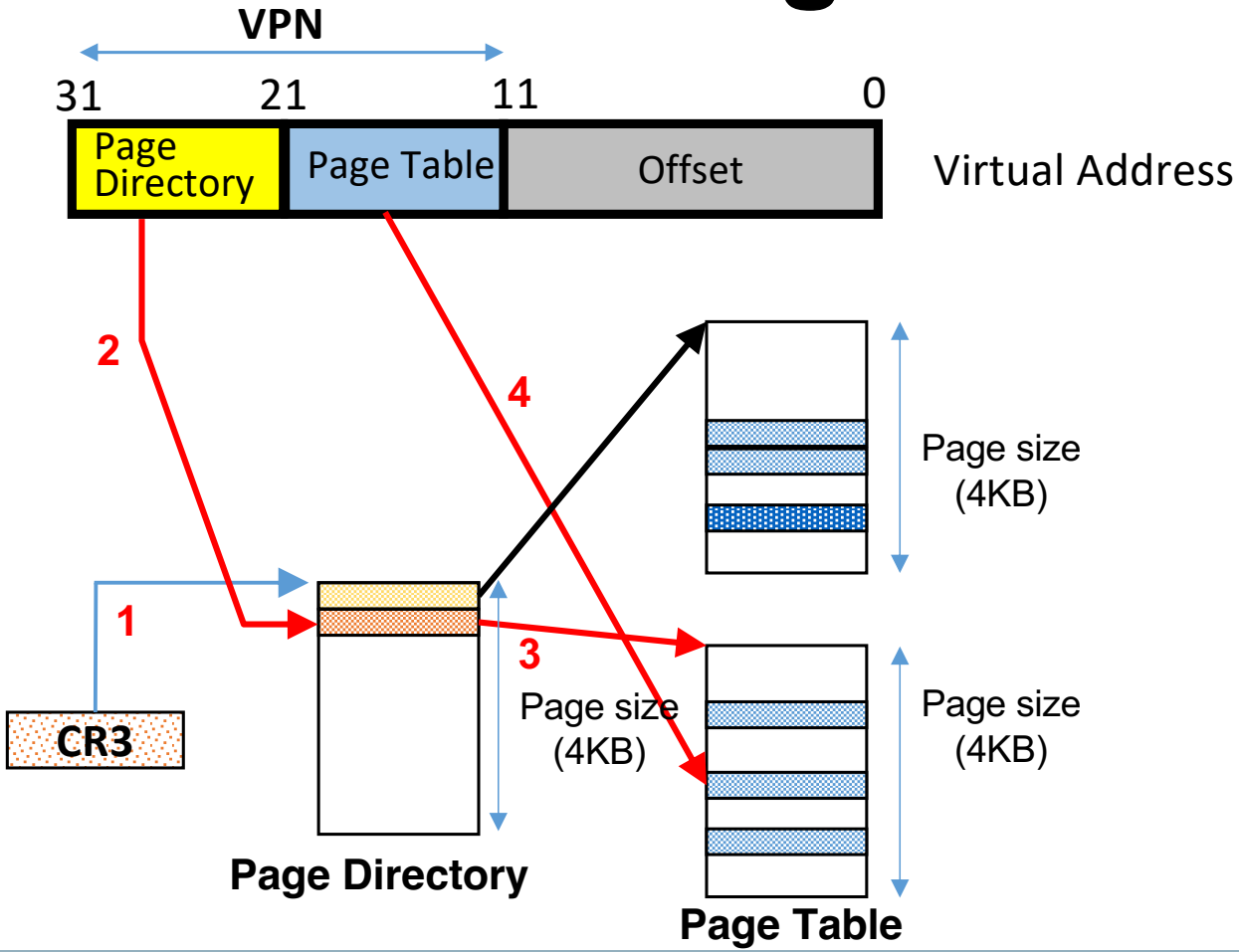
- Virtual address (32-bit) contains three different entries (instead of two that we saw earlier in case of linear page table)
  - Page Directory index (PDIndex)
    - Total entries 1024
    - Bits needed = 10
  - Index into page table corresponding to the above entry inside page directory
    - Total entries 1024
    - Bits needed = 10
  - Offset to address in the physical page

Two-Level Page Table in x86



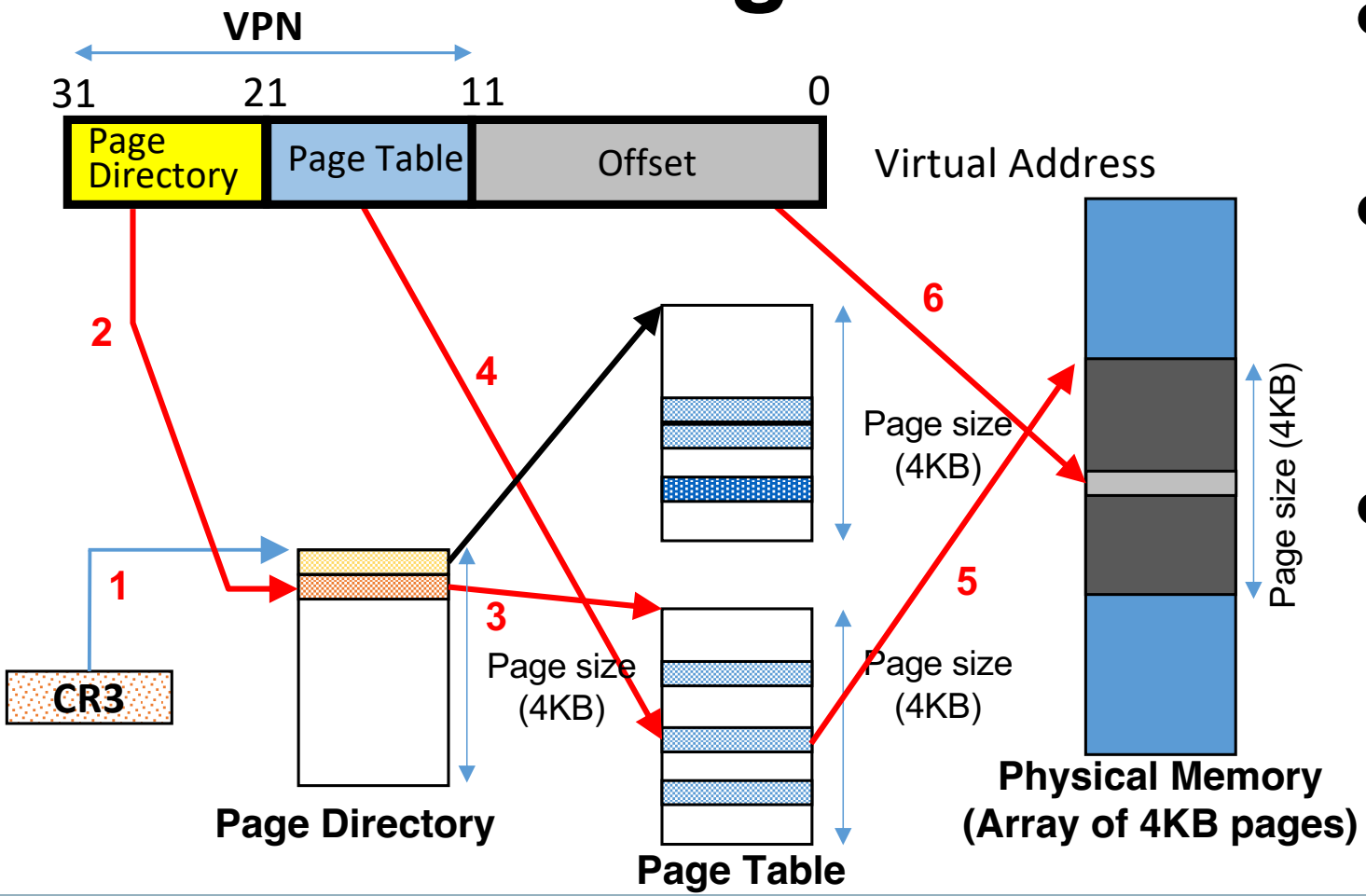
- Control Register (CR3) register (part of MMU) stores the starting address of the Page Directory (PD)
- PD contains 1024 Page Directory Entry (PDEs)
  - Each PDE contains the Page Frame Number and a status bit
  - Status=0 if there is not a single valid entry in the PFN, otherwise status =1 (valid PFN only if status=1)
- Address of the PDE associated with this VPN
  - PDE = CR3 + PDIndex * 4 bytes

Two-Level Page Table in x86



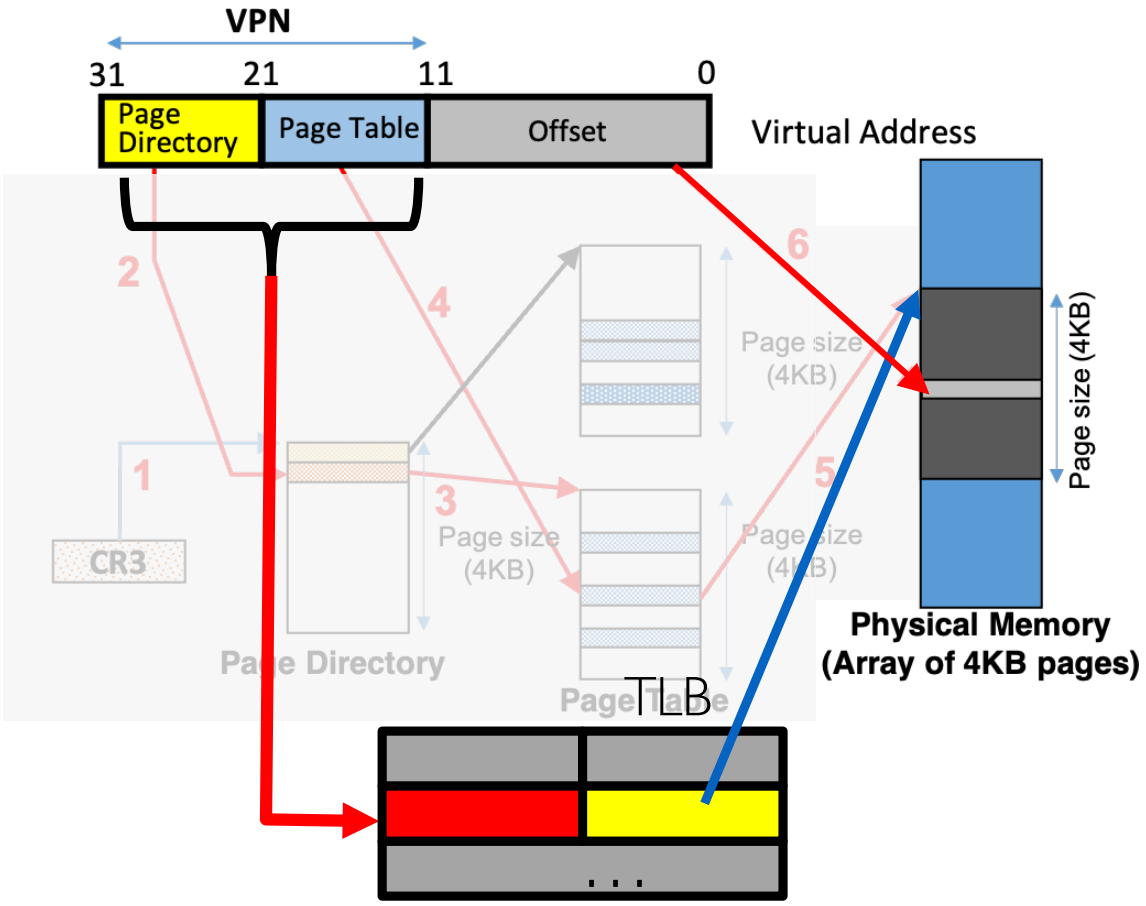
- Page Table Entry (PTE) can be now found from the PDE
  - PTE = PDE + (Page Table index * 4 bytes)

Two-Level Page Table in x86



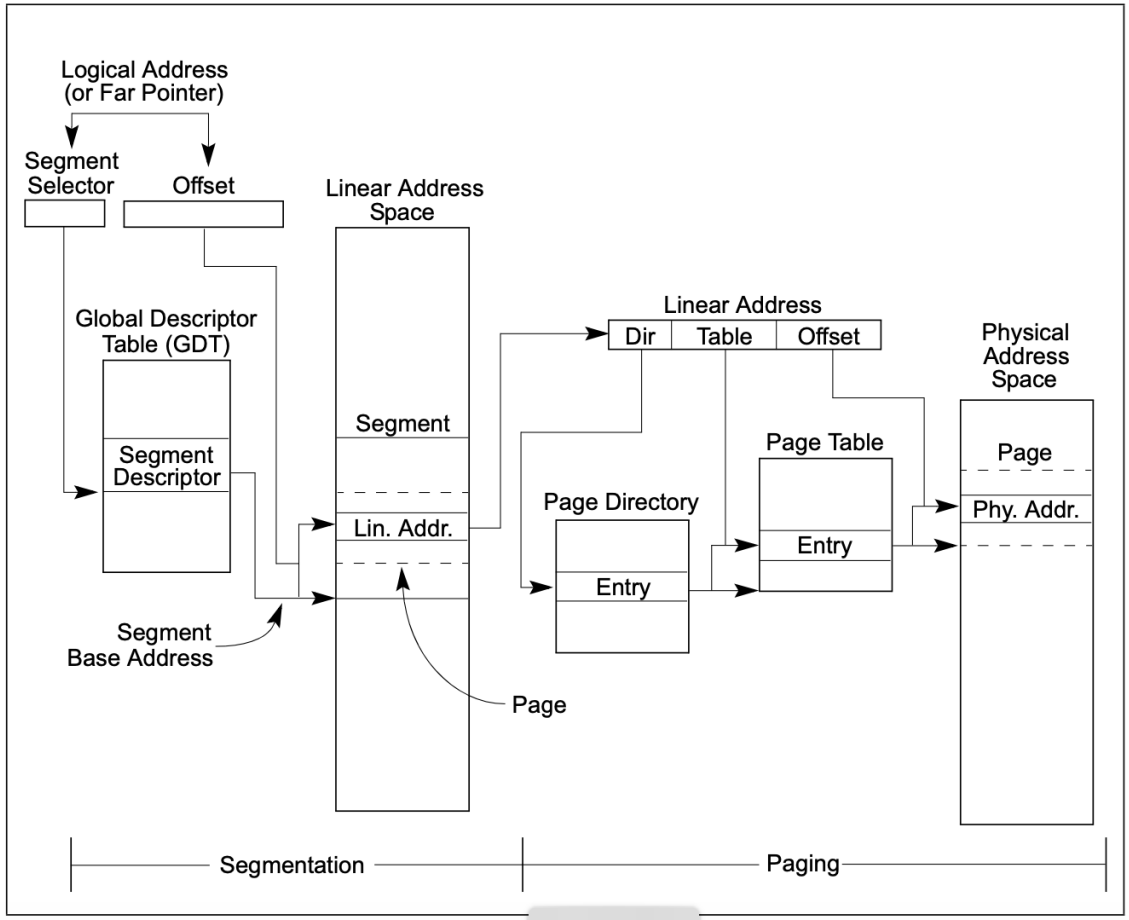
- Physical Page Frame Number (PFN) can be found from the PTE
- The physical address is obtained by adding the "offset" into the starting address of PFN
- What would happen when considering TLB?
  - None of these complex lookups are required if TLB contains the VPN->PPN

MLPT Along with TLB



- Look ups are required only in case of TLB misses
- Access latency significantly low when TLB hits
  - See Lecture-18

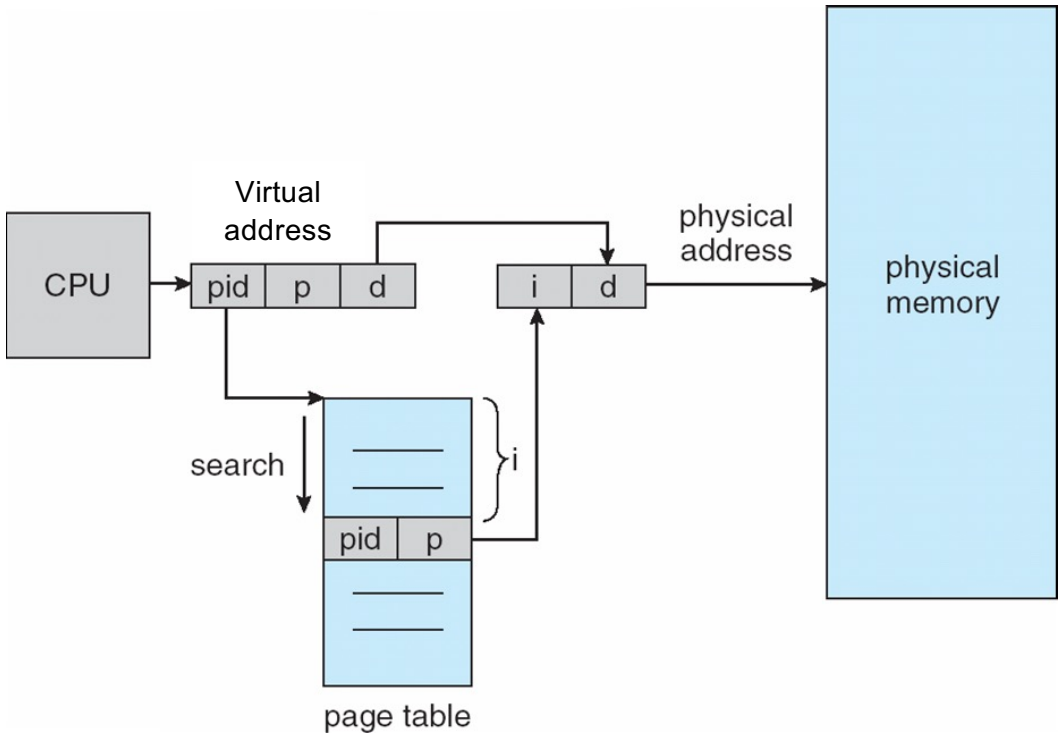
Memory Management in Intel x86



Mapping Virtual to Physical Address

- Approaches
  1. Virtual address is same as the physical address
  2. Virtual address is not the same as physical address
     a) Using base & bound for VA to PA mapping
     b) Using segmentation for VA to PA mapping
     c) Paging
        a) Using Page Table (PT) in RAM and having a base register for PT
        b) Using Page Table (PT) in RAM and having a base register for PT as well as a TLB cache
        c) Using Multi-level Page Table
        d) Using Inverted Page Table

Inverted Page Table



Picture source: https://www.os-book.com/OS10/

- Issues with MLPT
  - Easy implementation but many page tables
- Inverted Page Table
  - Size is independent of the virtual address space, and is directly related to the amount of physical memory
    - Each entry points to a page in the physical memory
    - Each entry corresponds to the VPN (“p”) and process (“pid”) that owns that page
    - The tuple (pid, p) inside a virtual address is used as a hash key to find the entry in the page table
  - Single page table contains entries for all the processes
    - Complex hash function!
  - It is called as inverted page table, as total entries is equal to the total number of PPN
    - In contrast, the page tables we have seen so far (a.k.a. forward page tables) the entries are according to the VPN

Next Lecture

- Demand paging
  - Last remaining topic in virtualization of memory