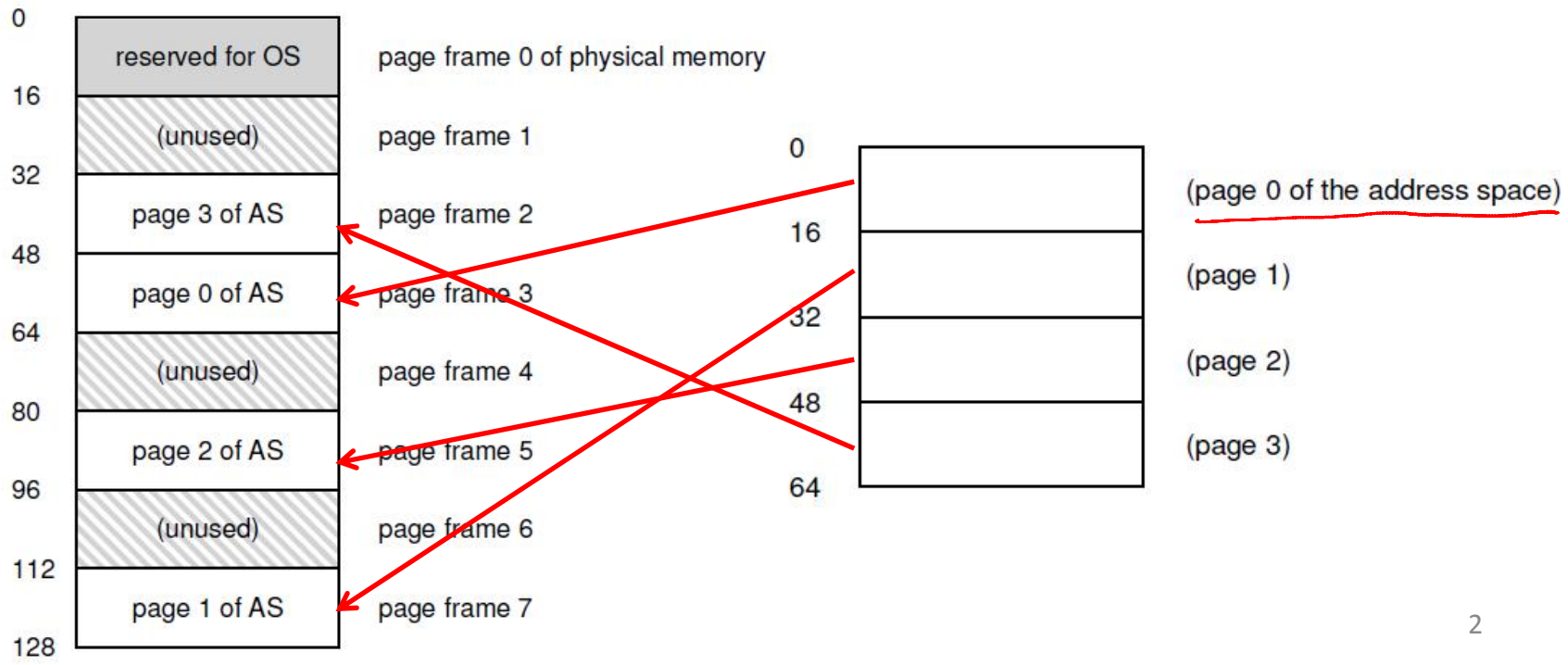
Lecture 9: Paging

Mythili Vutukuru
IIT Bombay

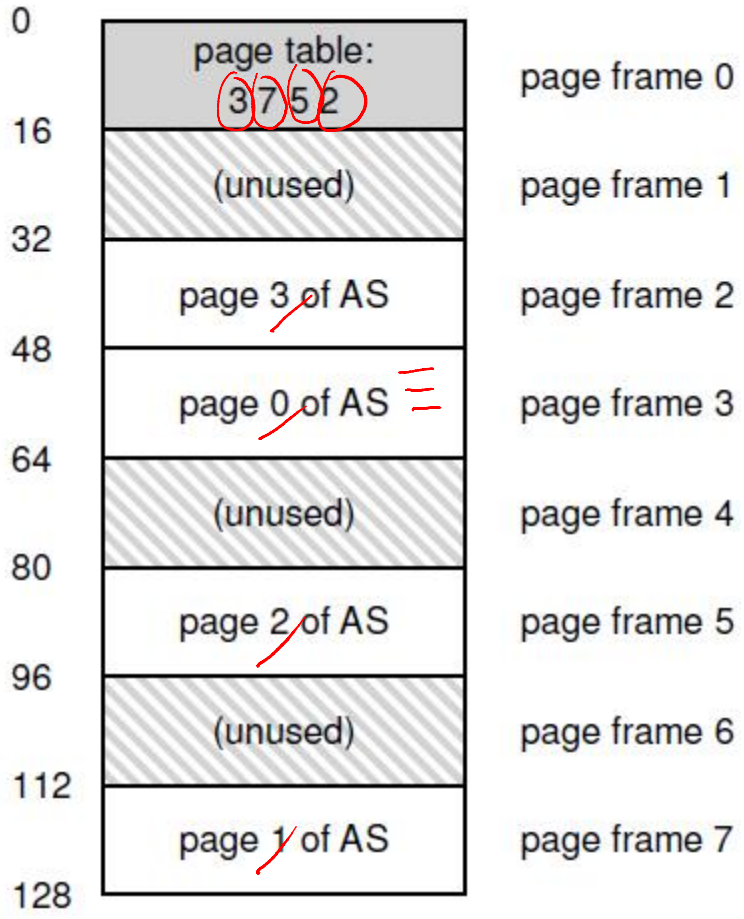
Paging

- Allocate memory in fixed size chunks (“pages”)
- Avoids external fragmentation (no small “holes”)
- Has internal fragmentation (partially filled pages)

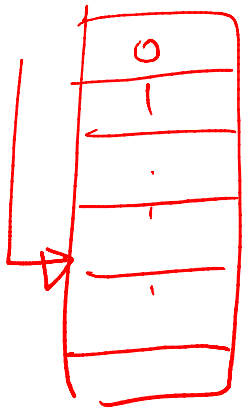


Page table

- Per process data structure to help VA-PA translation
- Array stores mappings from virtual page number (VPN) to physical frame number (PFN)
  - E.g., VP 0 → PF 3, VP 1 → PF 7
- Part of OS memory (in PCB)
- MMU has access to page table and uses it for address translation
- OS updates page table upon context switch



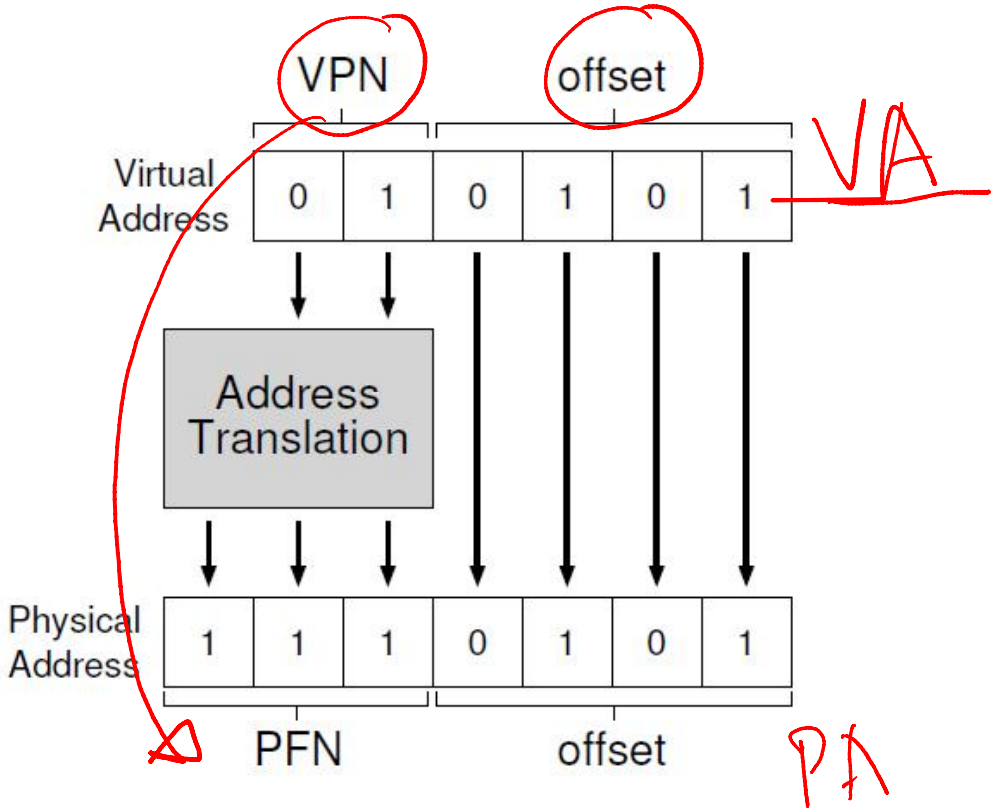
Page table entry (PTE)



- Simplest page table: linear page table
- Page table is an array of page table entries, one per virtual page
- VPN (virtual page no.) is index into this array
- Each PTE contains PFN (physical frame number) and few other bits
  - Valid bit: is this page used by process?
  - Protection bits: read/write permissions
  - Present bit: is this page in memory? (more later)
  - Dirty bit: has this page been modified?
  - Accessed bit: has this page been recently accessed?

Address translation in hardware

- Most significant bits of VA give the VPN
- Page table maps VPN to PFN
- PA is obtained from PFN and offset within a page
- MMU stores (physical) address of start of page table, not all entries.
- "Walks" the page table to get relevant PTE



What happens on memory access?

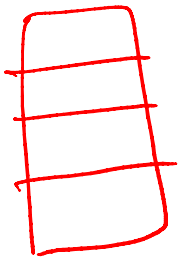
- CPU requests code or data at a virtual address
- MMU must translate VA to PA
  - First, access memory to read page table entry
  - Translate VA to PA
  - Then, access memory to fetch code/data
- Paging adds overhead to memory access
- Solution? A cache for VA-PA mappings

Translation Lookaside Buffer (TLB)

- A cache of recent VA-PA mappings
- To translate VA to PA, MMU first looks up TLB
- If TLB hit, PA can be directly used
- If TLB miss, then MMU performs additional memory accesses to "walk" page table
- TLB misses are expensive (multiple memory accesses)
  - Locality of reference helps to have high hit rate
- TLB entries may become invalid on context switch and change of page tables

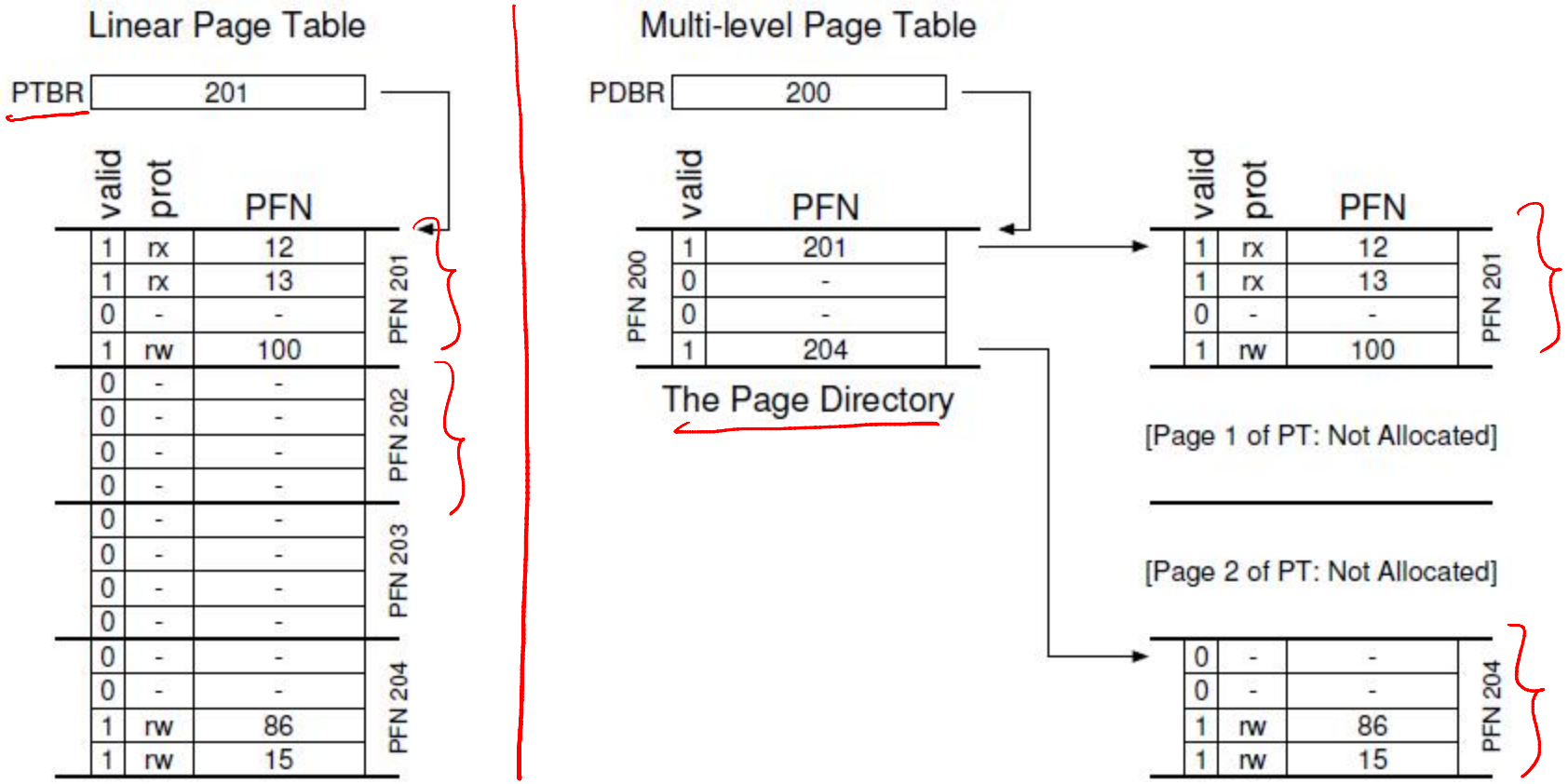
How are page tables stored in memory?

- What is typical size of page table?
  - 32 bit VA, 4 KB pages, so 2^32 / 2^12 = 2^20 entries
  - If each PTE is 4 bytes, then page table is 4MB
  - One such page table per process!
- How to reduce the size of page tables?
  - Larger pages, so fewer entries
- How does OS allocate memory for such large tables?
  - Page table is itself split into smaller chunks!



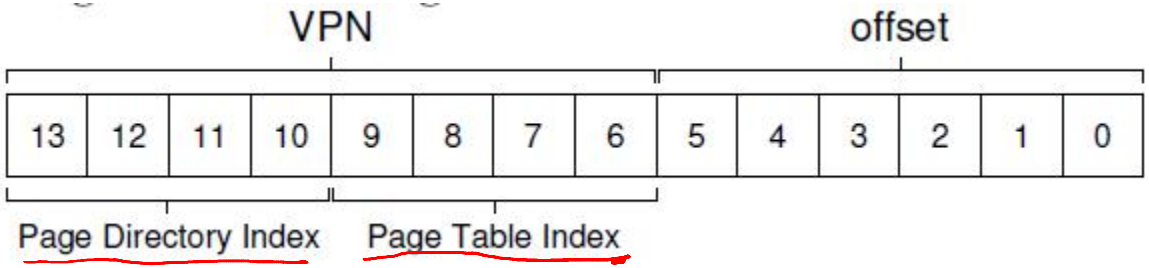
Multilevel page tables (1)

- A page table is spread over many pages
- An "outer" page table or page directory tracks the PFNs of the page table pages



Multilevel page tables (2)

- Depending on how large the page table is, we may need more than 2 levels also
  - 64-bit architectures may need 7 levels
- What about address translation?
  - First few bits of VA to identify outer page table entry
  - Next few bits to index into next level of PTEs



- In case of TLB miss, multiple accesses to memory required to access all the levels of page tables