Jin-Soo Kim
(*jinsoo.kim@snu.ac.kr*)

Systems Software &
Architecture Lab.

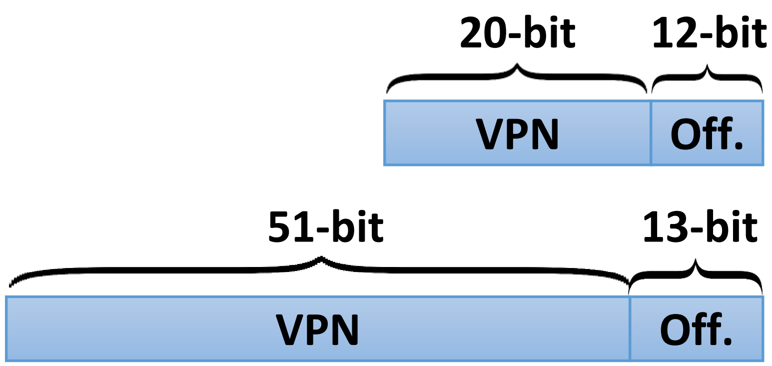
Seoul National University

Spring 2024

Page Tables

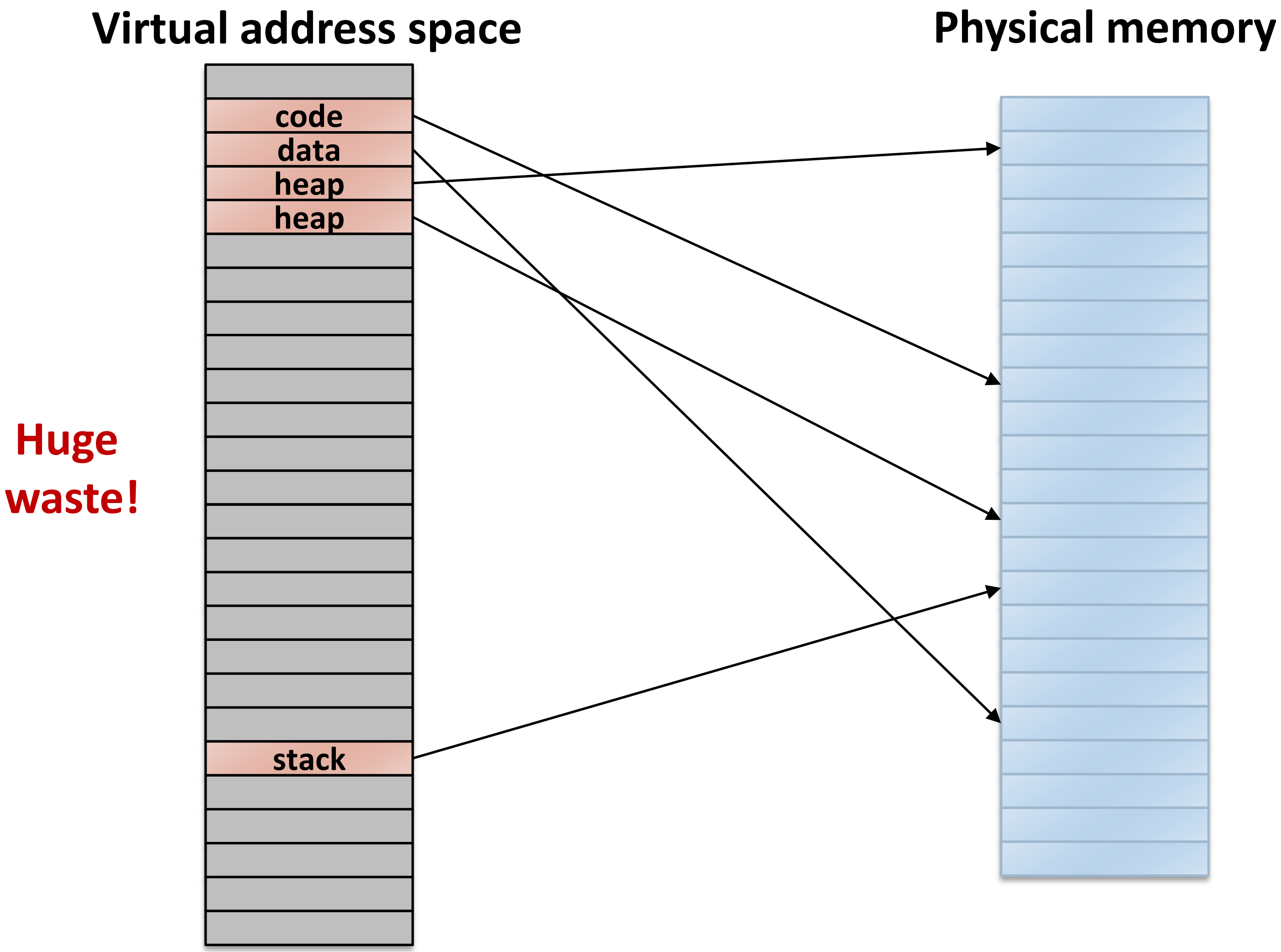
The Problem

- Space overhead of page tables
  - A 32-bit address space with 4KB pages (4 bytes/PTE): $2^{20} * 4 = 4MB$ (per process)
  - A 64-bit address space with 8KB pages (8 bytes/PTE): $2^{51} * 8 = 2^{54} = 16PB$ (per process)



- How can we reduce this overhead?
  - Observation: many invalid PTEs
  - Only need to map the portion of the address space actually being used which is a tiny fraction of the entire address space

(Typical) Linear Page Table



Paging with Segmentation

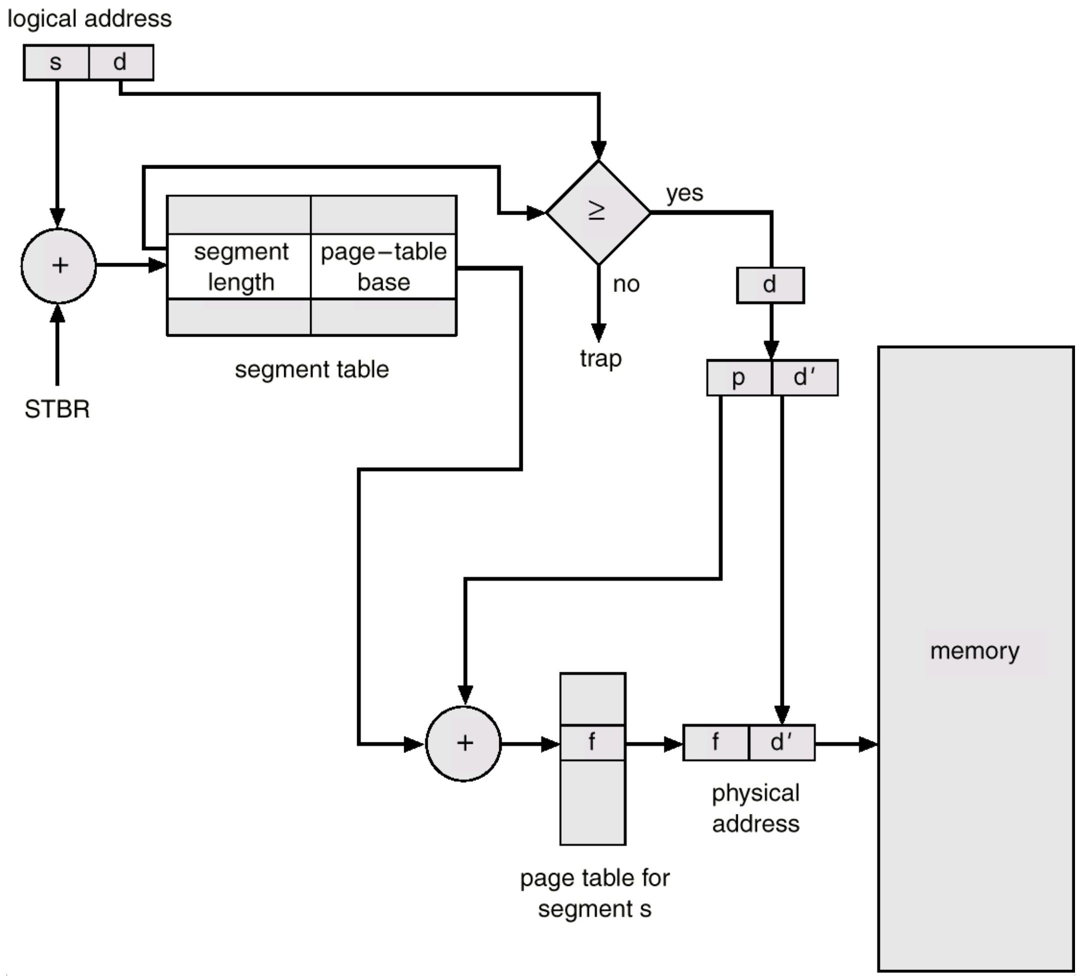
- A segment represents a region of valid address space
  - Segmentation:
    - Divide virtual address space into segments
    - Each segment can have variable length
  - Paging:
    - Divide each segment into fixed-sized pages
    - Each segment has a page table
    - Each segment tracks base (physical address) and limit of the page table for that segment

- Virtual address divided into three portions

Seg #	Page number	Page offset

Paging with Segmentation: Example

- Multics address translation

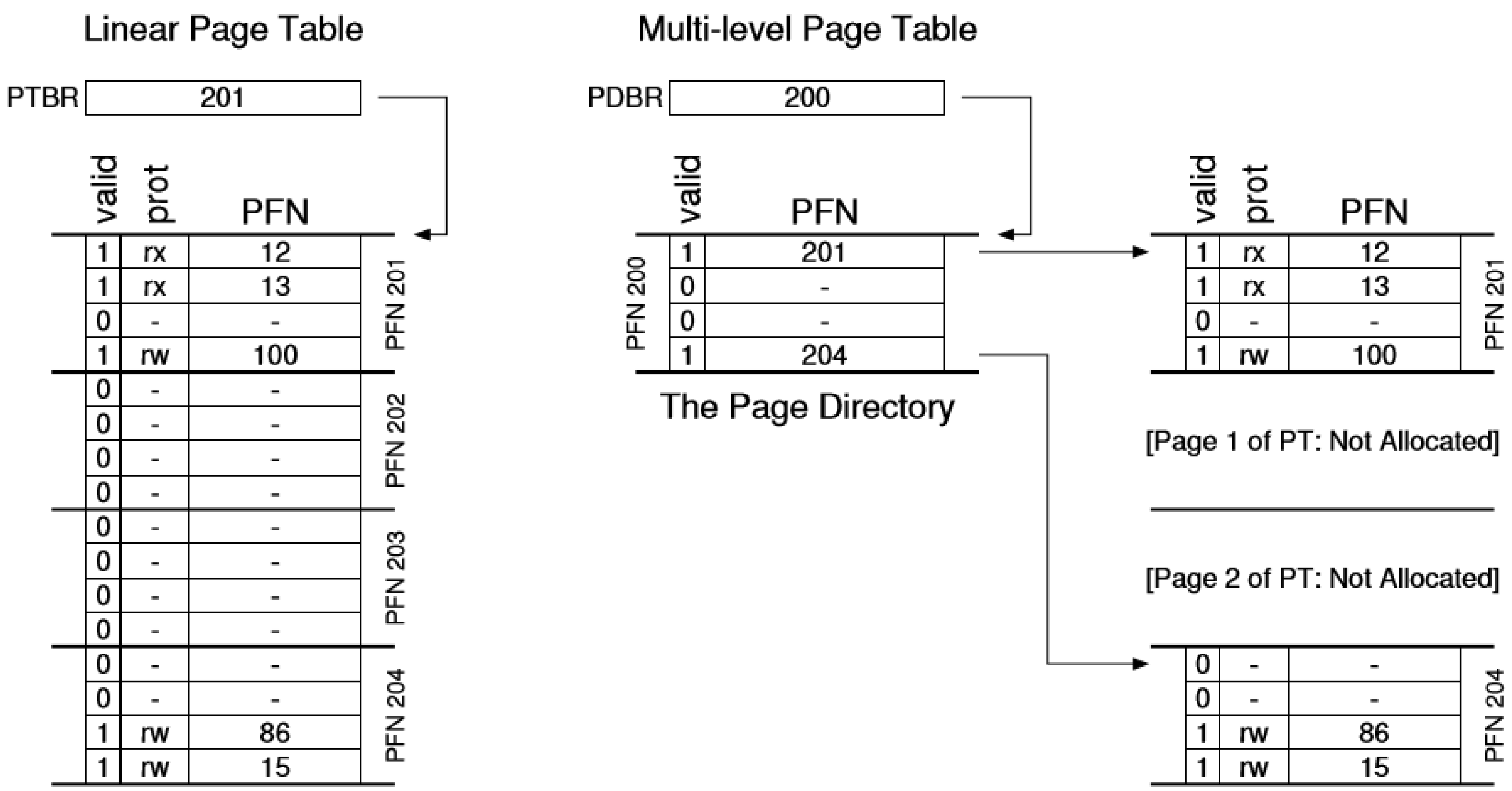


Summary: Paging with Segmentation

- Pros
  - Can decrease the size of page tables
  - Segments can grow without any reshuffling
  - Can run process when some pages are swapped to disk
  - Increases flexibility of sharing: share either single page or entire segment

- Cons
  - Page tables potentially can be large
    - e.g., large but sparsely used heaps will have a lot of waste
  - External fragmentation due to page tables
    - Each page table should be allocated contiguously

Linear vs. Multi-level Page Table

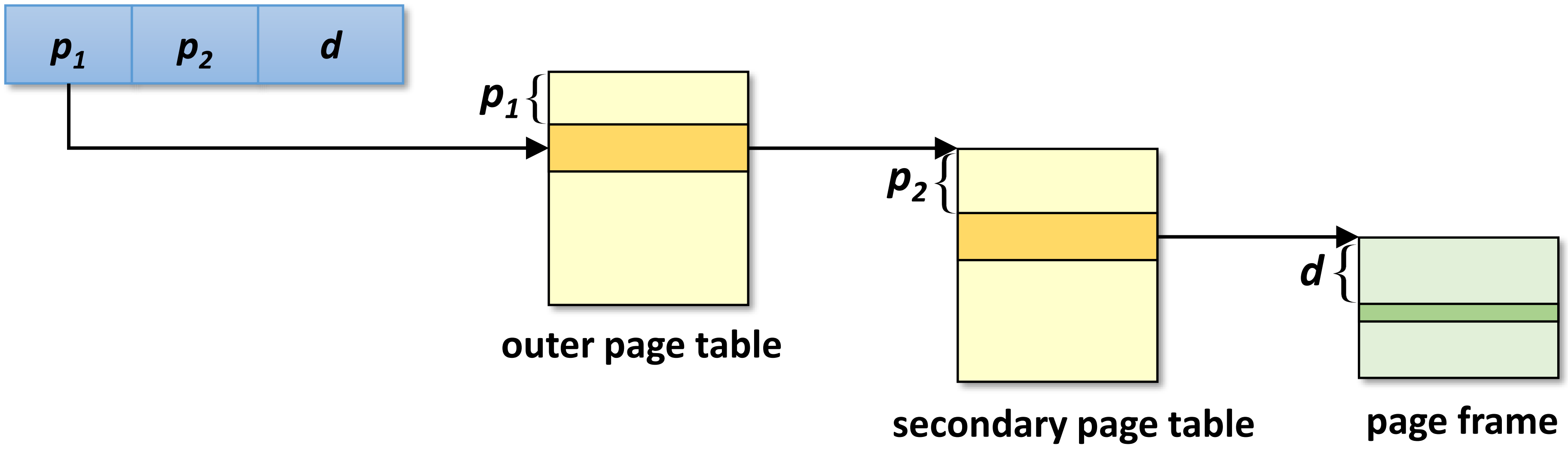


Multi-level Page Table

- Allow each page table to be allocated non-contiguously
- Virtual addresses have 3 parts

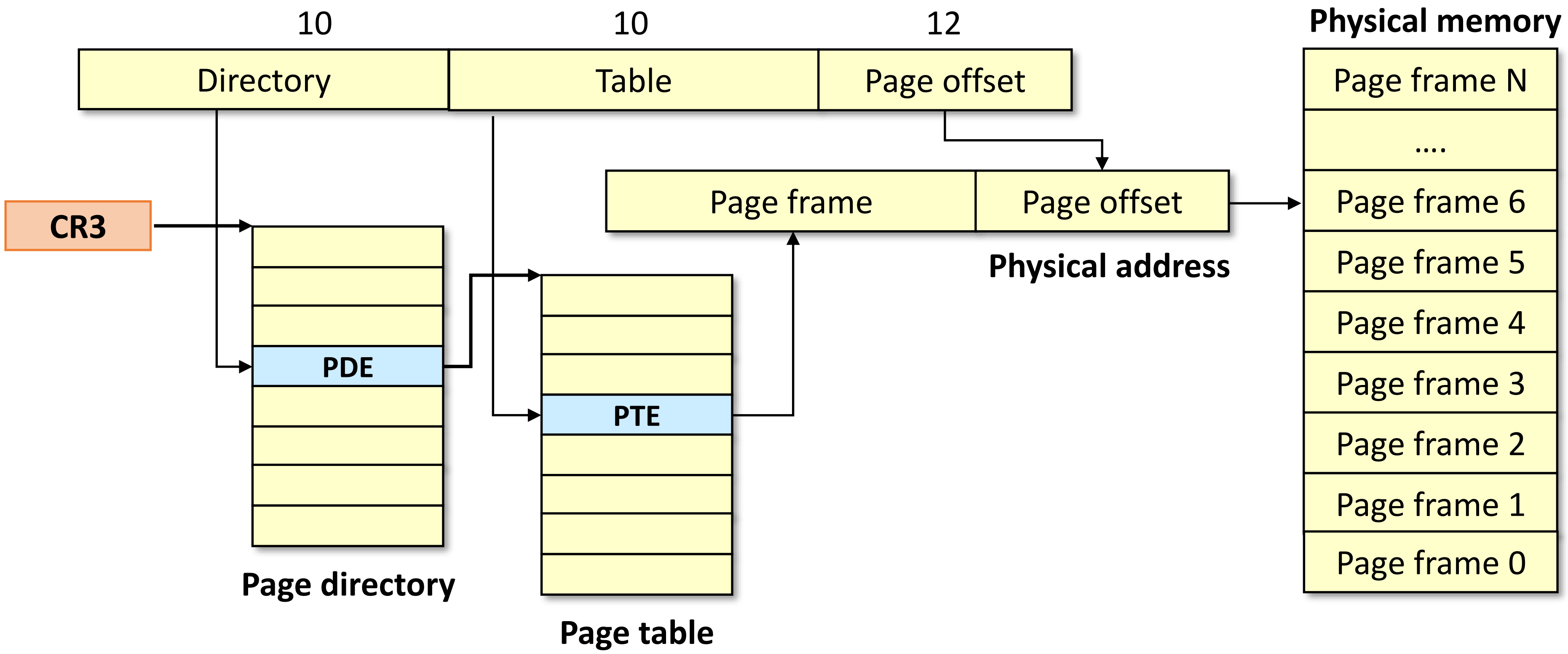
Outer page #	Secondary page #	Page offset

  - Outer page table: outer page number → secondary page table
  - Secondary page table: secondary page # → page frame #



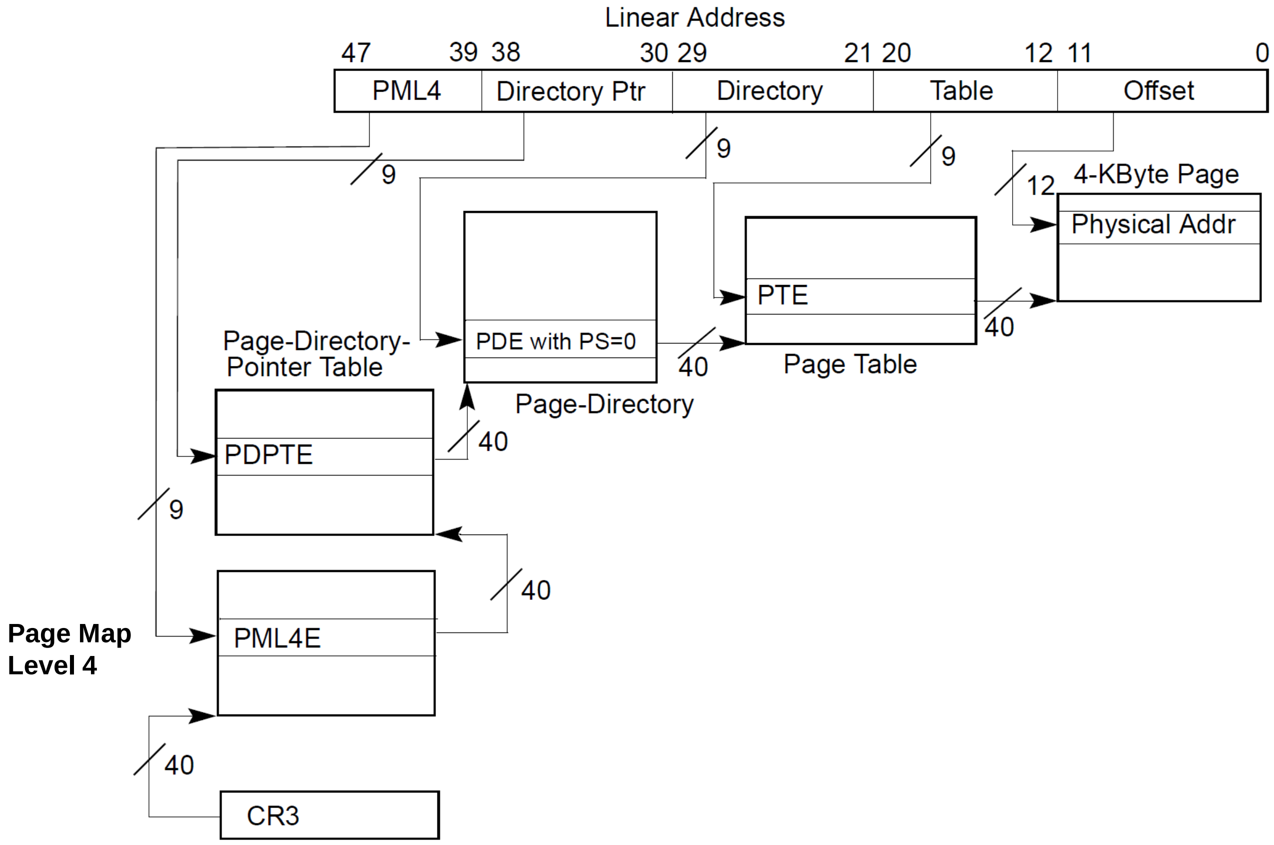
Multi-level Page Table: IA-32

- 32-bit paging
  - 32-bit address space, 4KB pages, 4 bytes/PTE
  - Want every page table fit into a page

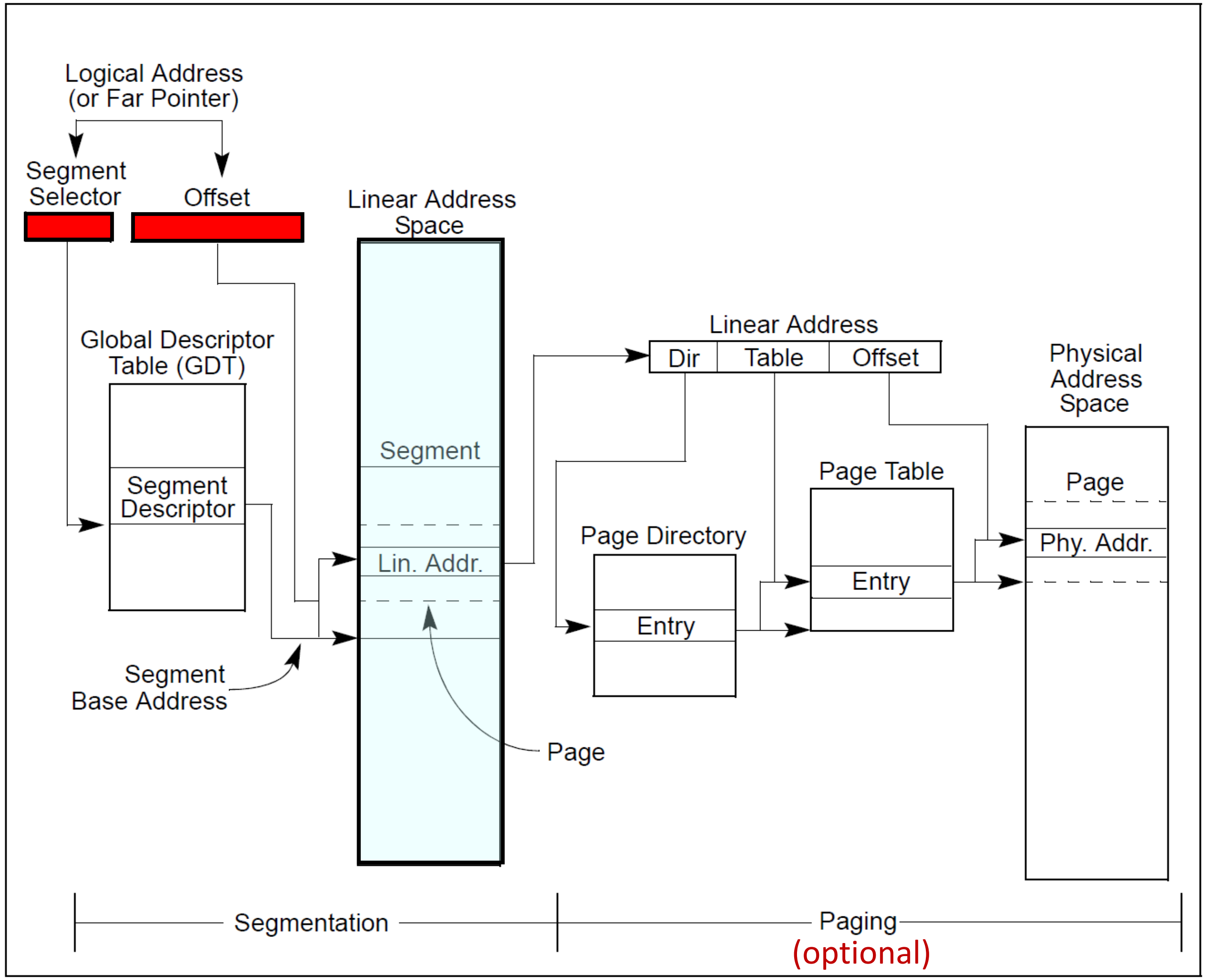


Multi-level Page Table: Intel 64

- Address translation in Intel 64 architecture
  - 48-bit "*linear*" address → 52-bit physical address (4KB page)



Intel VM Architecture (IA-32)



Multi-level Page Table: RISC-V

- Sv32 (used for RV32)
  - 32-bit virtual address → 34-bit physical address

31–22	21–12	11–0
VPN[1]	VPN[0]	page offset
10	10	12

satp register holds the physical page number (PPN) of the root page table

- Sv39 (used in xv6)
  - 39-bit virtual address → 56-bit physical address

38–30	29–21	20–12	11–0
VPN[2]	VPN[1]	VPN[0]	page offset
9	9	9	12

- Sv48
  - 48-bit virtual address → 56-bit physical address

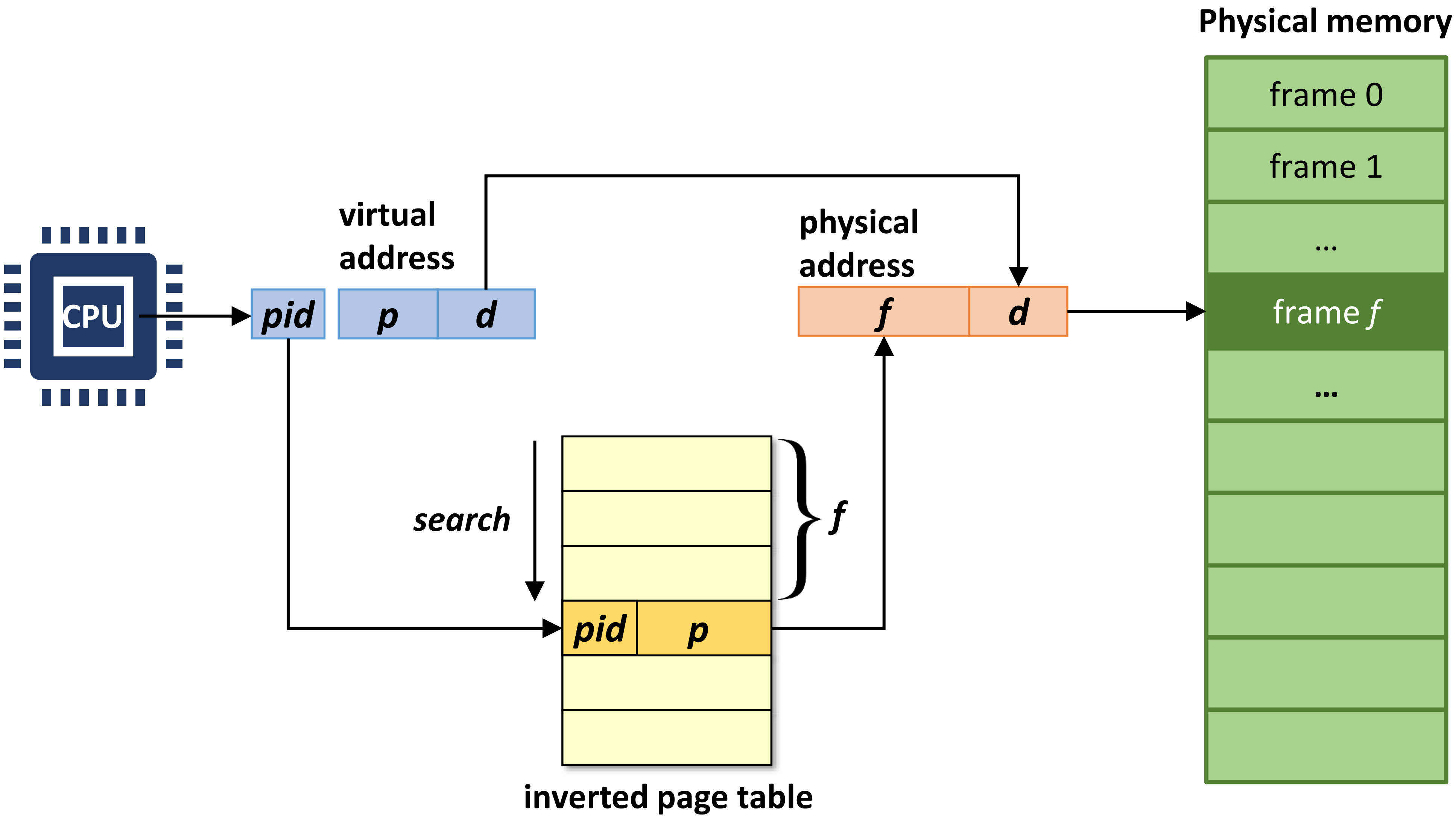
47–39	38–30	29–21	20–12	11–0
VPN[3]	VPN[2]	VPN[1]	VPN[0]	page offset
9	9	9	9	12

Summary: Multi-level Page Table

- Pros
  - Compact while supporting sparse address space
    - Page-table space in proportion to the amount of address space used
  - Easier to manage physical memory
    - Each page table usually fits within a page
  - Easier for hardware to walk through page tables
  - No external fragmentation

- Cons
  - More memory accesses on a TLB miss
  - More complex than a simple linear page-table lookup

Inverted Page Table



Summary: Inverted Page Table

- Reverse mapping from PFN → <VPN, PID>
  - One entry for each page frame in physical memory
  - Entry consists of the virtual page number with information about the process that owns that page
  - Need to search through the table to find match
  - Use hashing to limit the search to one, or at most a few, page-table entries

- Pros & Cons
  - Decrease memory needed to store page tables:
    No need to have per-process page tables
  - Increase time needed to search the table on a TLB miss

Paging Page Tables

- Store page tables in ________ address space
  - Cold (unused) page table pages can be paged out to disk
  - But, now addressing page tables requires translation
  - Outer page table is usually pinned into physical memory
  - Outer page table points to the ________ addresses (in the kernel address space) of secondary page tables
  - Need to handle nested page faults

- What if we page kernel code and data too?