Please SCAN

Address Translation

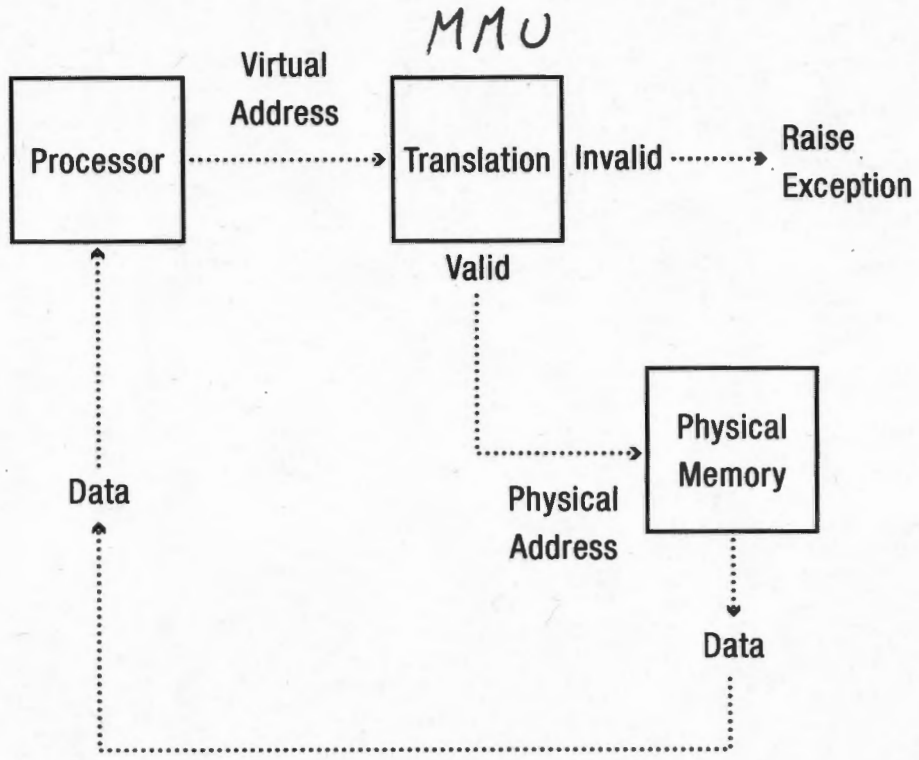
part 2

Main Points

- Address Translation Concept
  - How do we convert a virtual address to a physical address?
- Flexible Address Translation
  - Segmentation — REGIONS OF VIRTUAL MEMORY; COMPLEX MEM MGT
  - Paging — - LINEAR COVERAGE OF ADDRESS SPACE; - BAD FOR SPARSE ADDRESS USAGE
  - Multilevel translation
- Efficient Address Translation
  - Translation Lookaside Buffers
  - Virtually and physically addressed caches

BOTH-
COPY ON WRITE
STACK GROWTH
LOAD ON DEMAND

Address Translation Concept



LINEAR PAGE TABLES

Beyond Paging: Sparse Address Spaces

- Might want many separate segments
  - Per-processor heaps
  - Per-thread stacks
  - Memory-mapped files
  - Dynamically linked libraries
- What if virtual address space is large?
  - 32-bits, 4KB pages => 500K page table entries
  - 64-bits => 4 quadrillion page table entries

Multi-level Translation

- Tree of translation tables
  - Paged segmentation
  - Multi-level page tables
  - Multi-level paged segmentation
- All have pages as lowest level; why?

Multilevel Translation with Pages at Lowest Level

- Efficient memory allocation (vs. segments)
- Efficient for sparse addresses (vs. 1 level paging)
- Efficient disk transfers (fixed size units)
- Easier to build translation lookaside buffers
- Efficient reverse lookup (from physical -> virtual)
- Variable granularity for protection/sharing CORE MAP

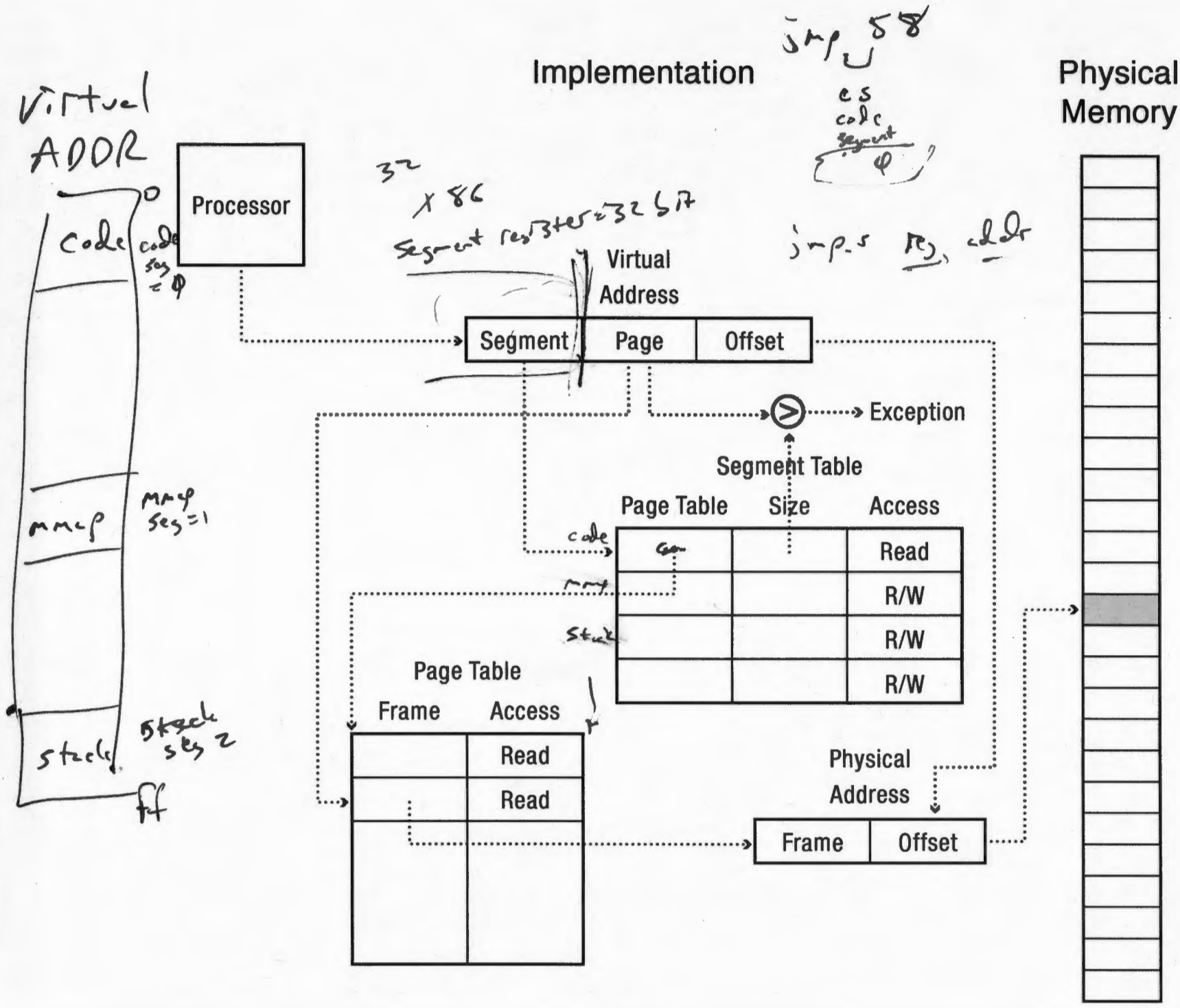
- Except: see discussion of superpages

Paged Segmentation

- Process memory is segmented
- Segment table entry:
  - Pointer to page table
  - Page table length (# of pages in segment)
  - Access permissions
- Page table entry:
  - Page frame
  - Access permissions
- Share memory or set access permissions at either page or segment-level

16 bit
x86

32 bit
x86
(points to
2 level
PT)


Implementation
Physical Memory
Virtual ADDR
Processor
Virtual Address
Segment
Page
Offset
Exception
Segment Table
Page Table
Size
Access
Read
R/W
R/W
R/W
Page Table
Frame
Access
Read
Read
Physical Address
Frame
Offset

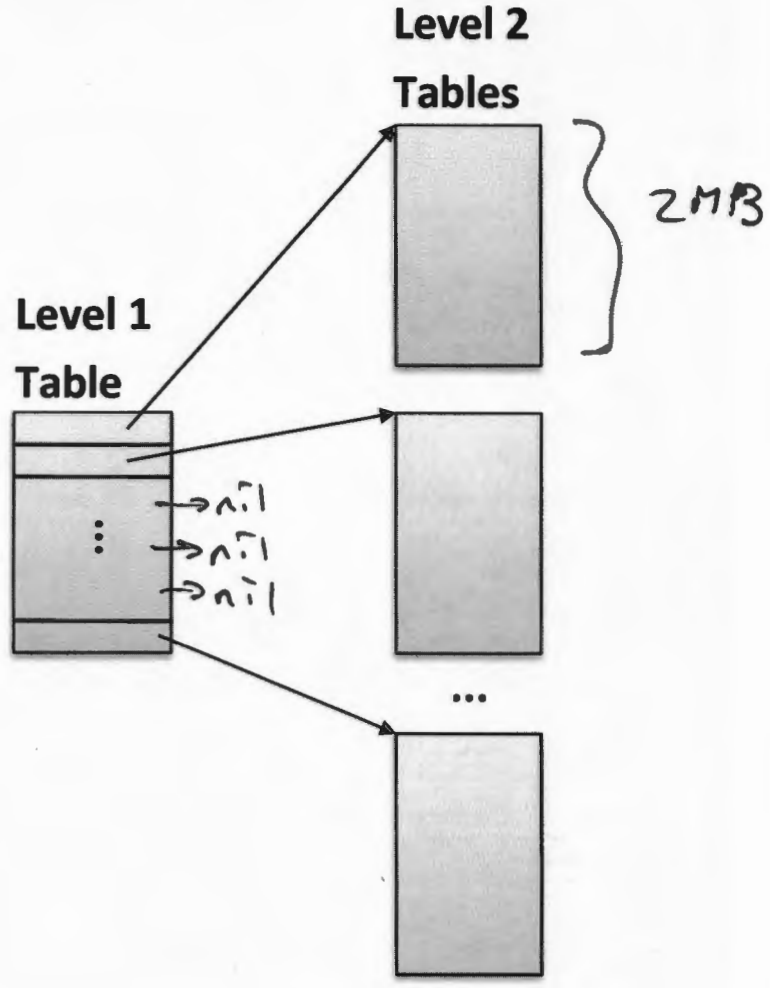
Question

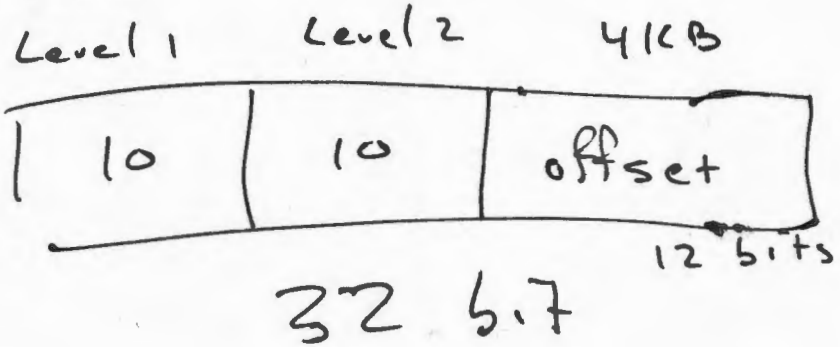
- With paged segmentation, what must be saved/restored across a process context switch?

Multi-level or hierarchical page tables

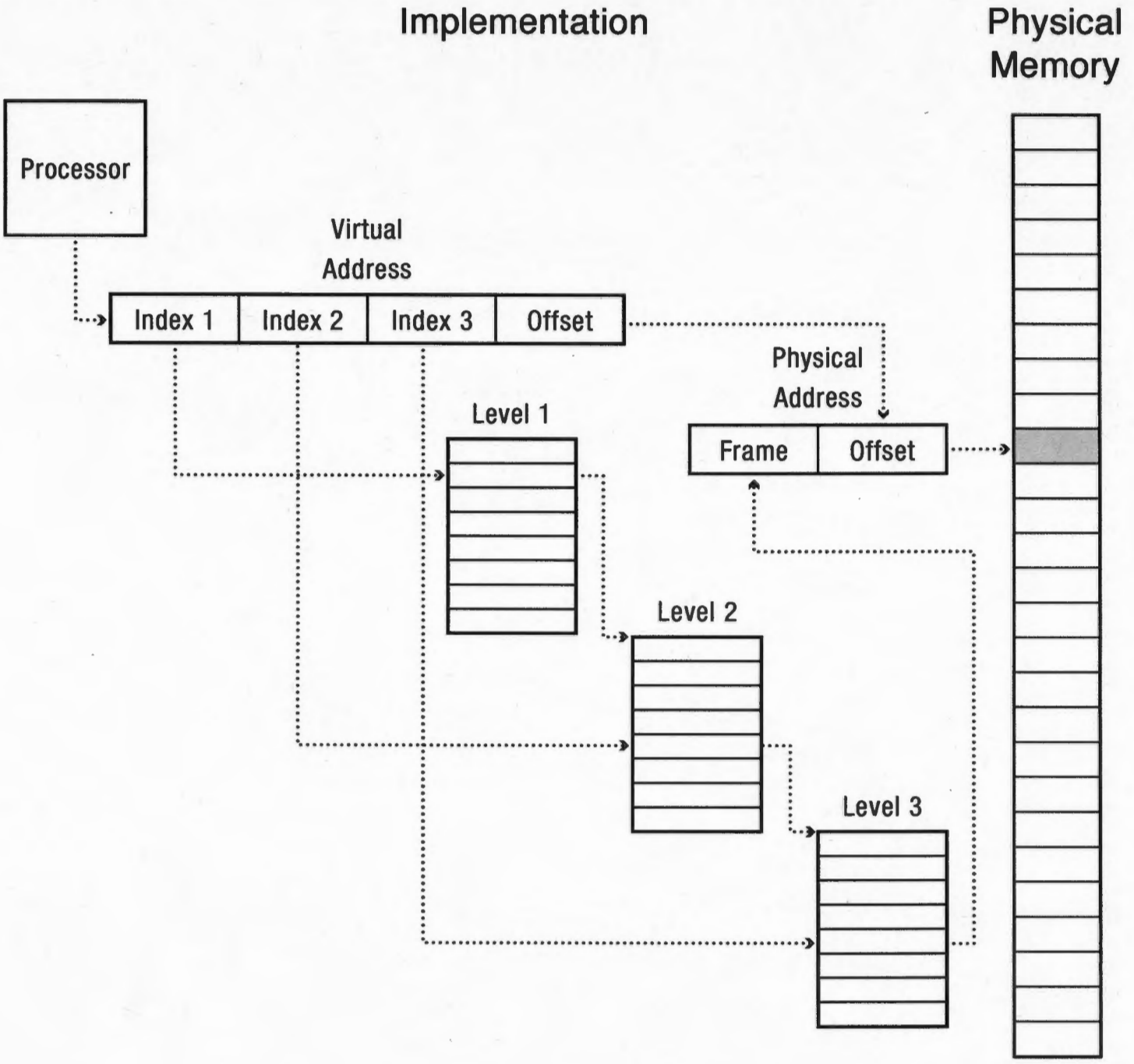
32 bit
x86

- Ex: 2-level page table
  - Level 1 table: each PTE points to a page table
  - Level 2 table: each PTE points to a page
- Can share/protect/page in/out at either level 1 or level 2





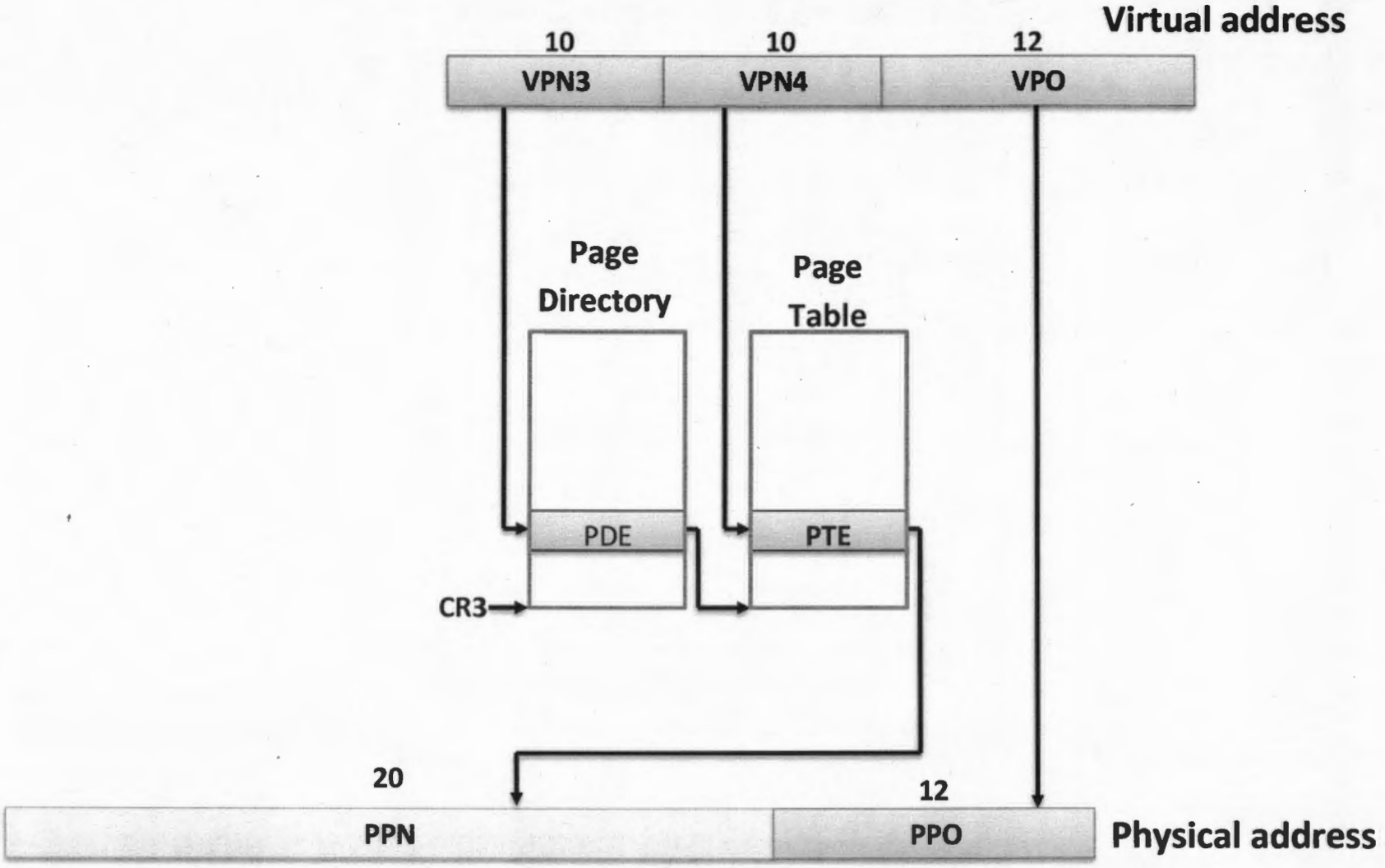
32 bit


Implementation
Physical Memory
Processor
Virtual Address
Index 1
Index 2
Index 3
Offset
Physical Address
Frame
Offset
Level 1
Level 2
Level 3

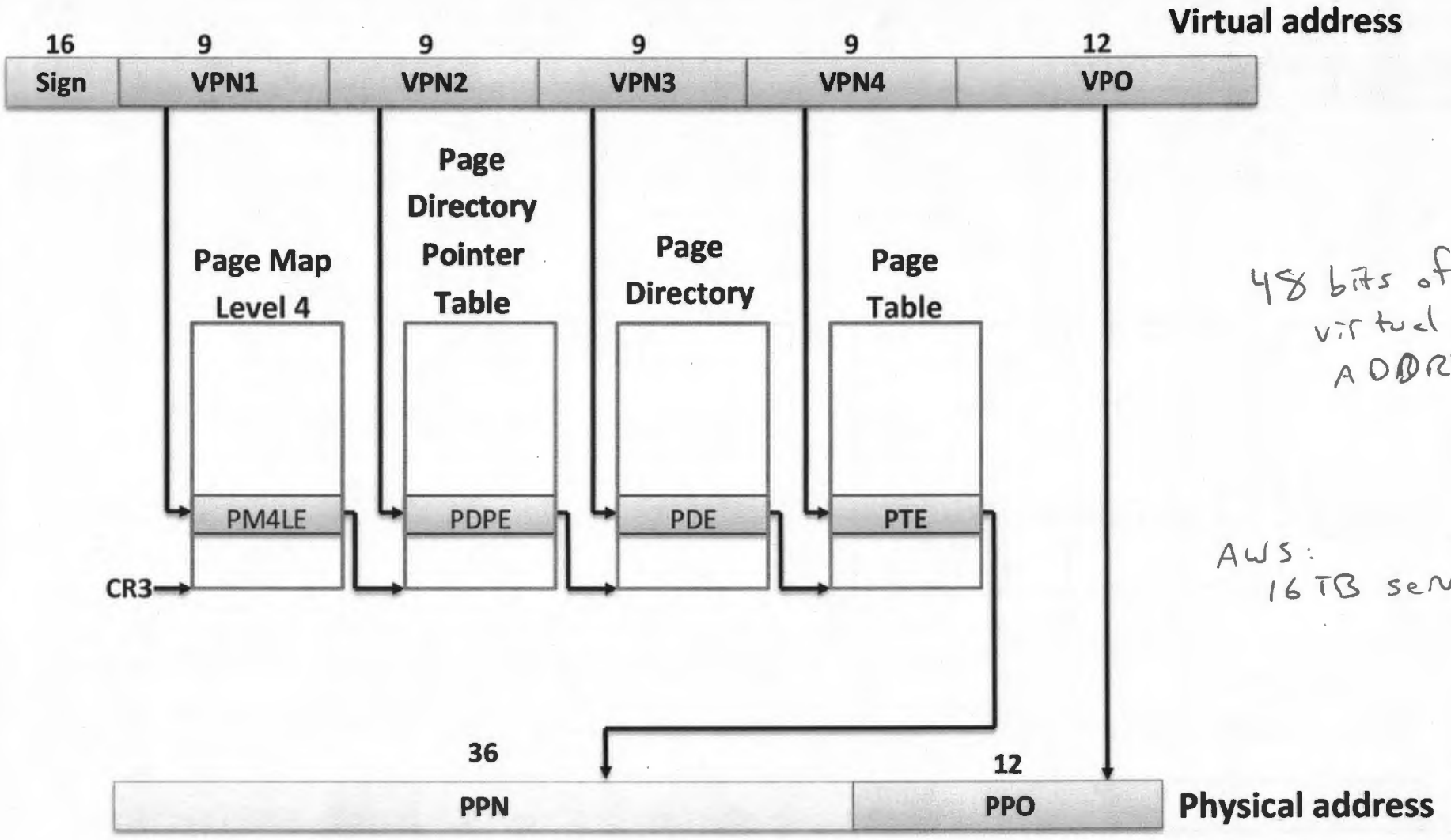
x86 Multilevel Paging

- Omit sub-tree if no valid addresses
  - Good for sparse address space
- 4KB pages
- Each level of page table fits in one page
- 32-bit: two level page table (per segment)
- 64-bit: four level page table (per segment)

x86-32 Paging



x86-64 Paging



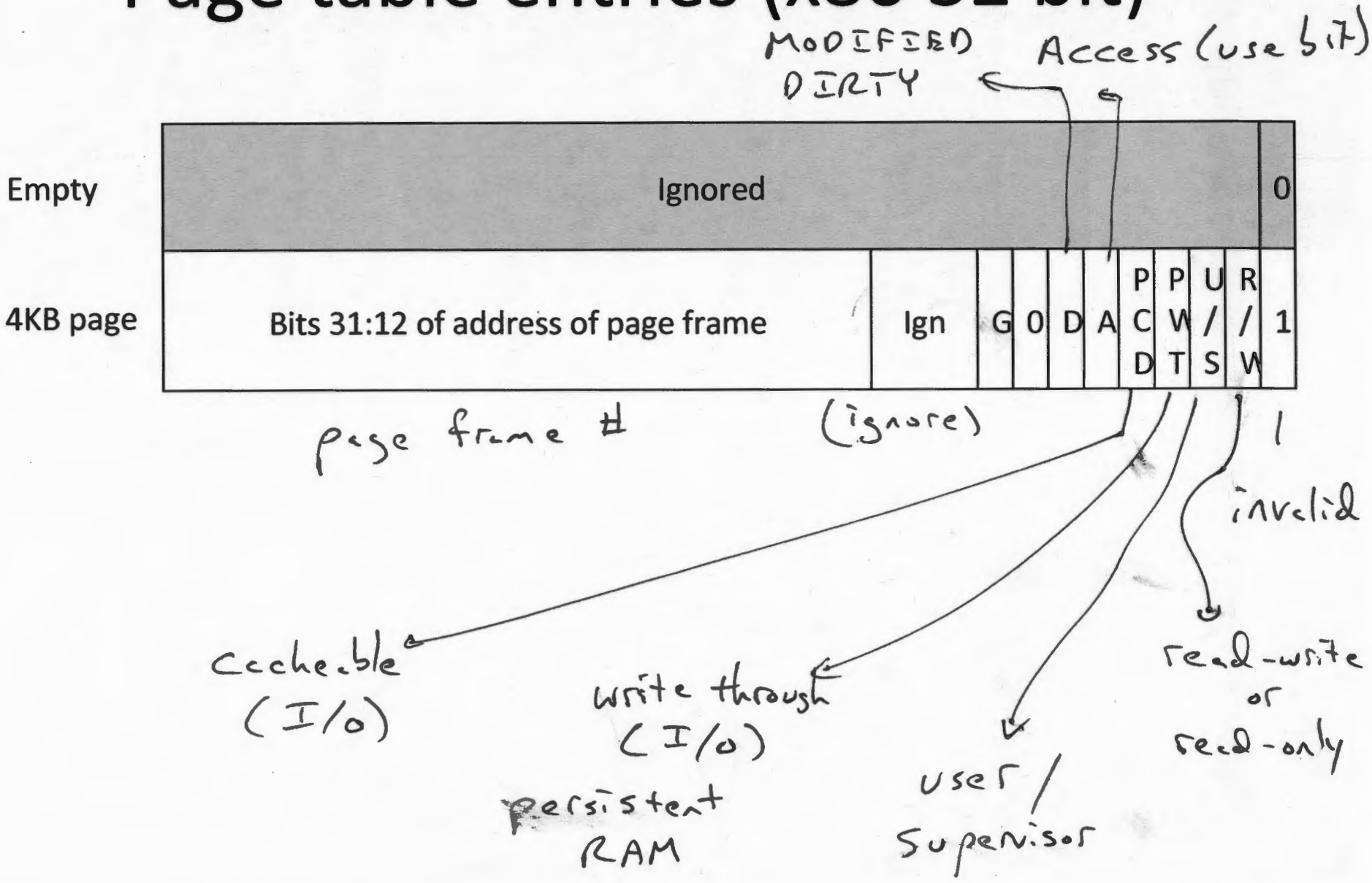
48 bits of virtual ADDRESS

AWS: 16 TB servers

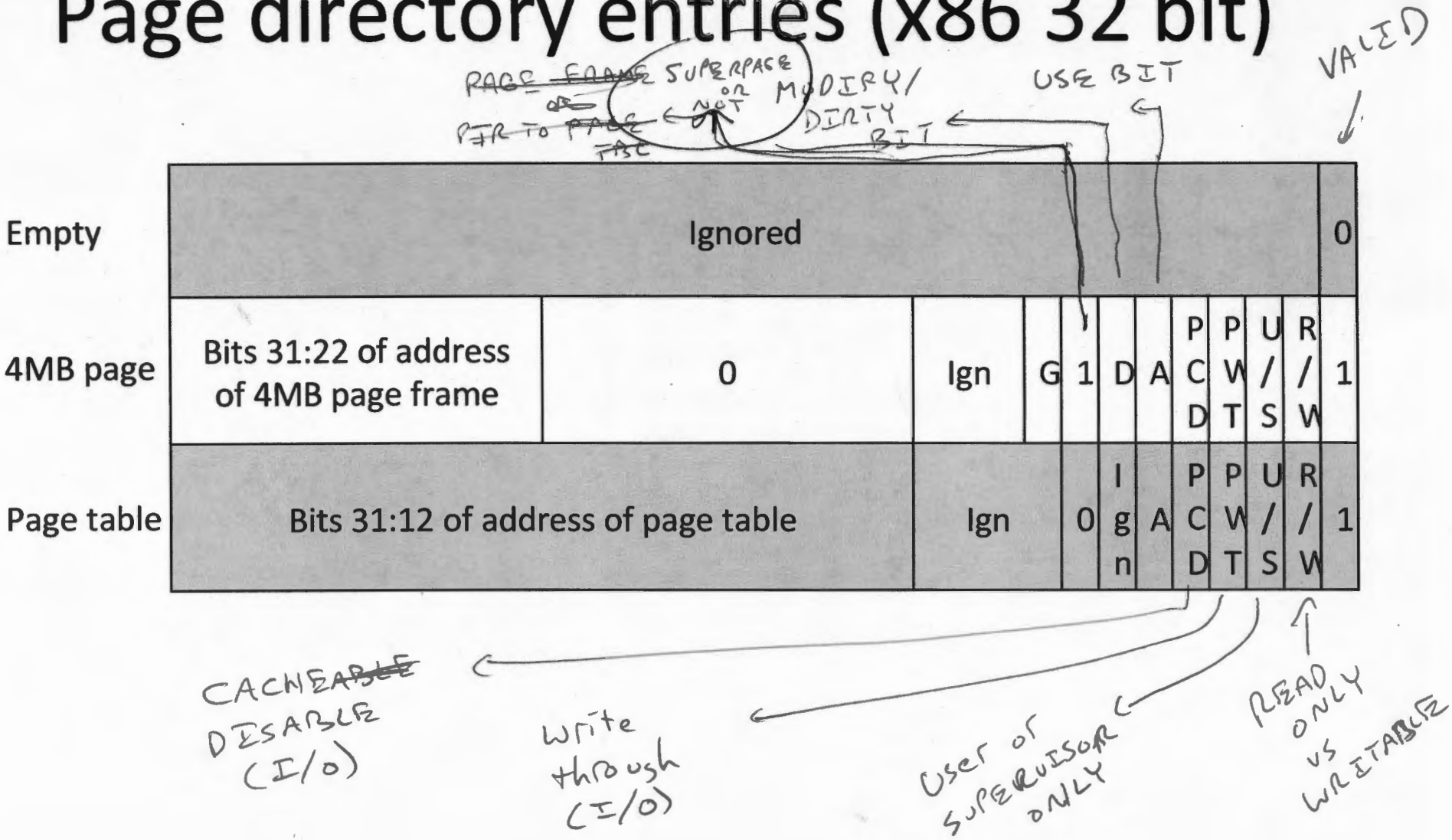
48 bits of Physical ADDRESS

256 TB

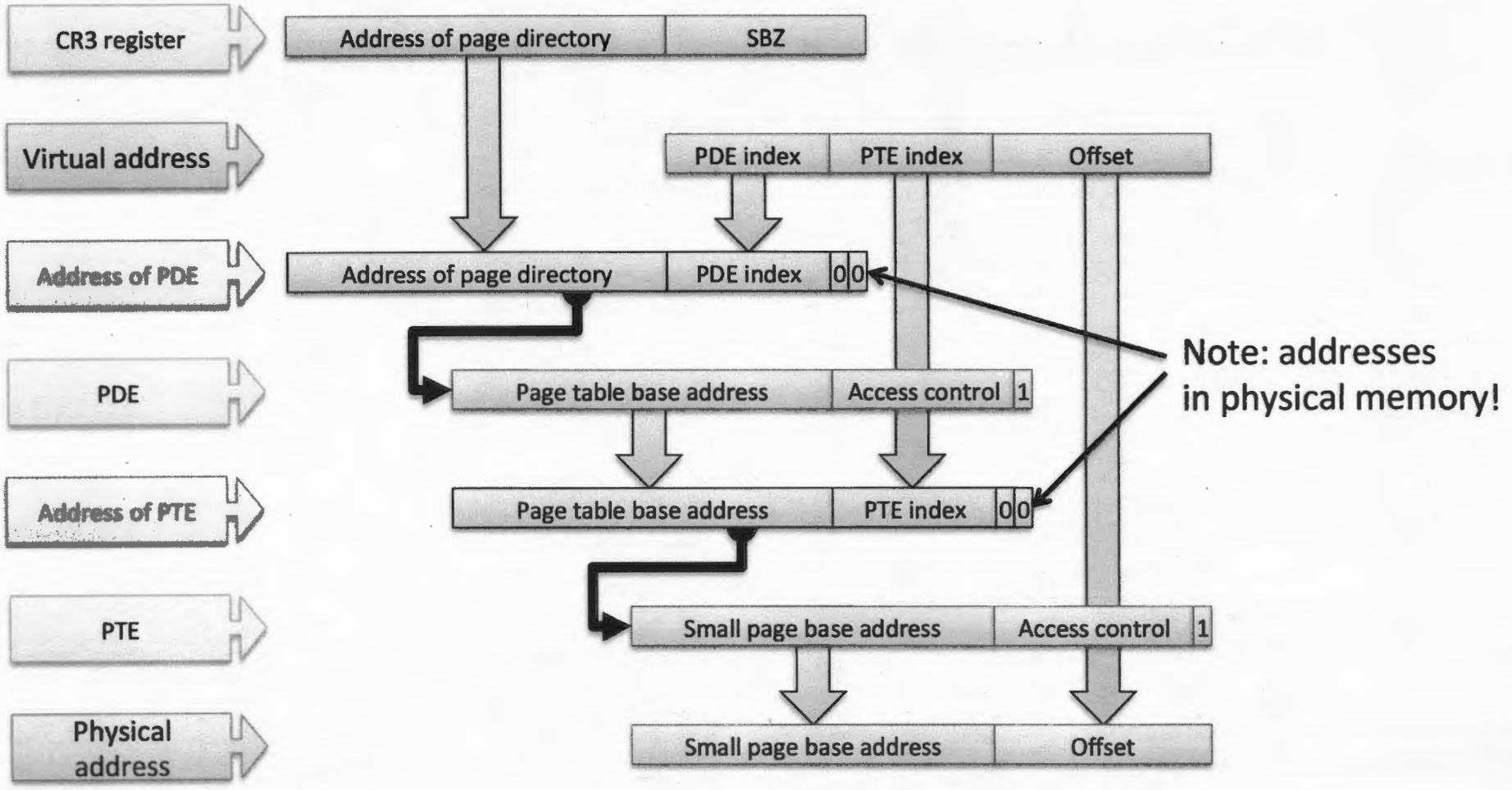
Page table entries (x86 32 bit)



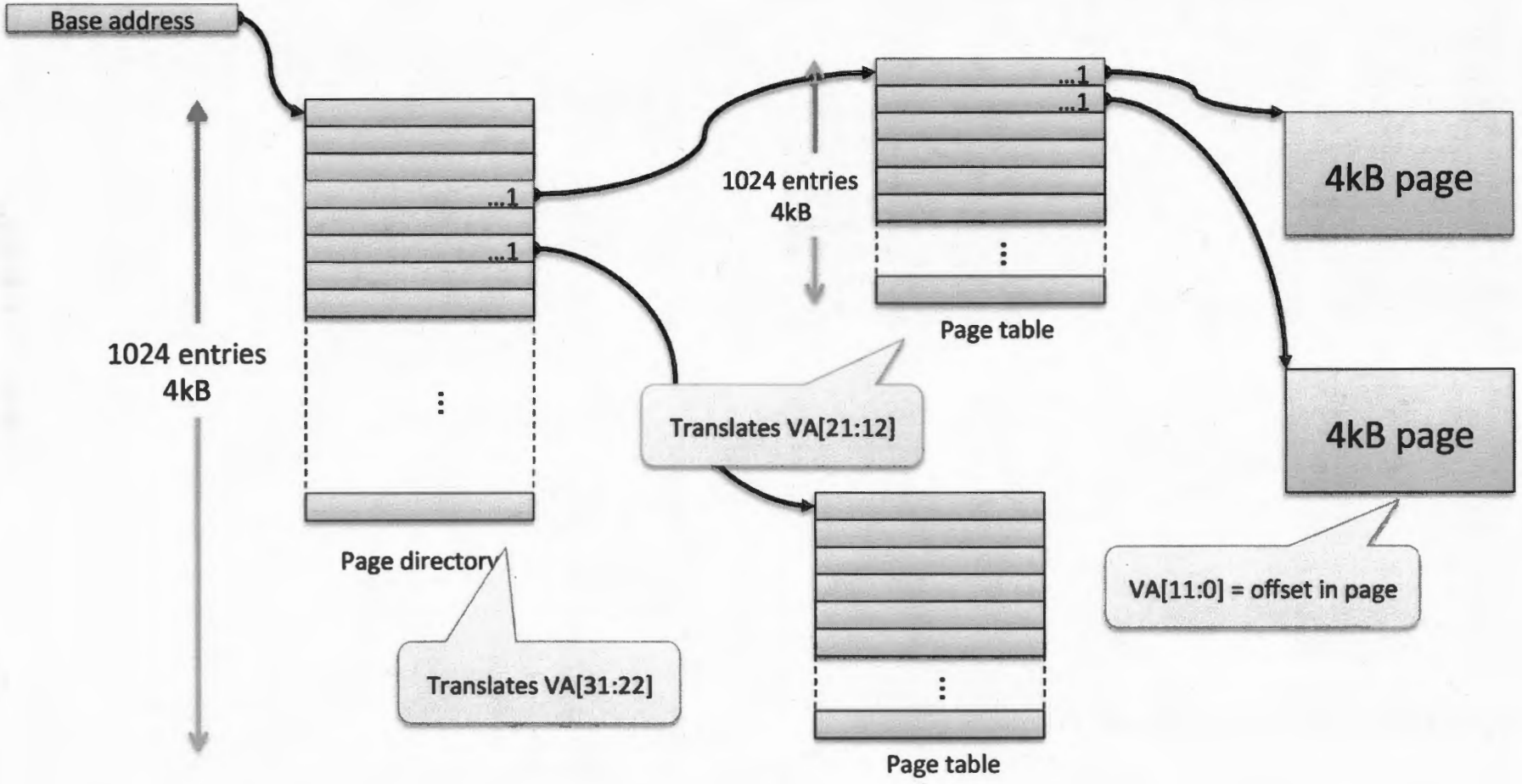
Page directory entries (x86 32 bit)



Small page translation



Page table for small pages



Question

- Write pseudo-code for translating a virtual address to a physical address for a system using 3-level paging, with 8 bits of address per level

PTD[ADDR >> 24], PT[ADDR >> 16 & 0xFF]

x86 Multilevel Paged Segmentation

- Global Descriptor Table (segment table)
- Each segment descriptor
  - Pointer to (multilevel) page table
  - Segment length
  - Segment access permissions
- Context switch
  - change global descriptor table register (GDTR), pointer to global descriptor table
  - Side effect: invalidates TLB

Multilevel Translation

- Pros:
  - Allocate/fill only page table entries that are in use
  - Simple memory allocation
  - Share at segment or page level
- Cons:
  - Space overhead: one pointer per virtual page
  - Multiple lookups per memory reference

Page Translation in the OS

- OS's need to keep their own data structures
  - List of memory objects (segments)
  - Virtual page -> physical page frame
  - Physical page frame -> set of virtual pages
  - Keep track of copy on write, load on demand, ...
- Why not just use the hardware page tables?

Kernel Page Translation

- Kernel maintains its own page translation data structures
  - Portable, flexible
  - Copy changes down into hardware page tables
- Example: Inverted page table
  - Hash from virtual page -> physical page
  - Space proportional to # of physical pages
- Example: virtual/shadow page table
  - Linux kernel tables mirror x86 structure, even on ARM