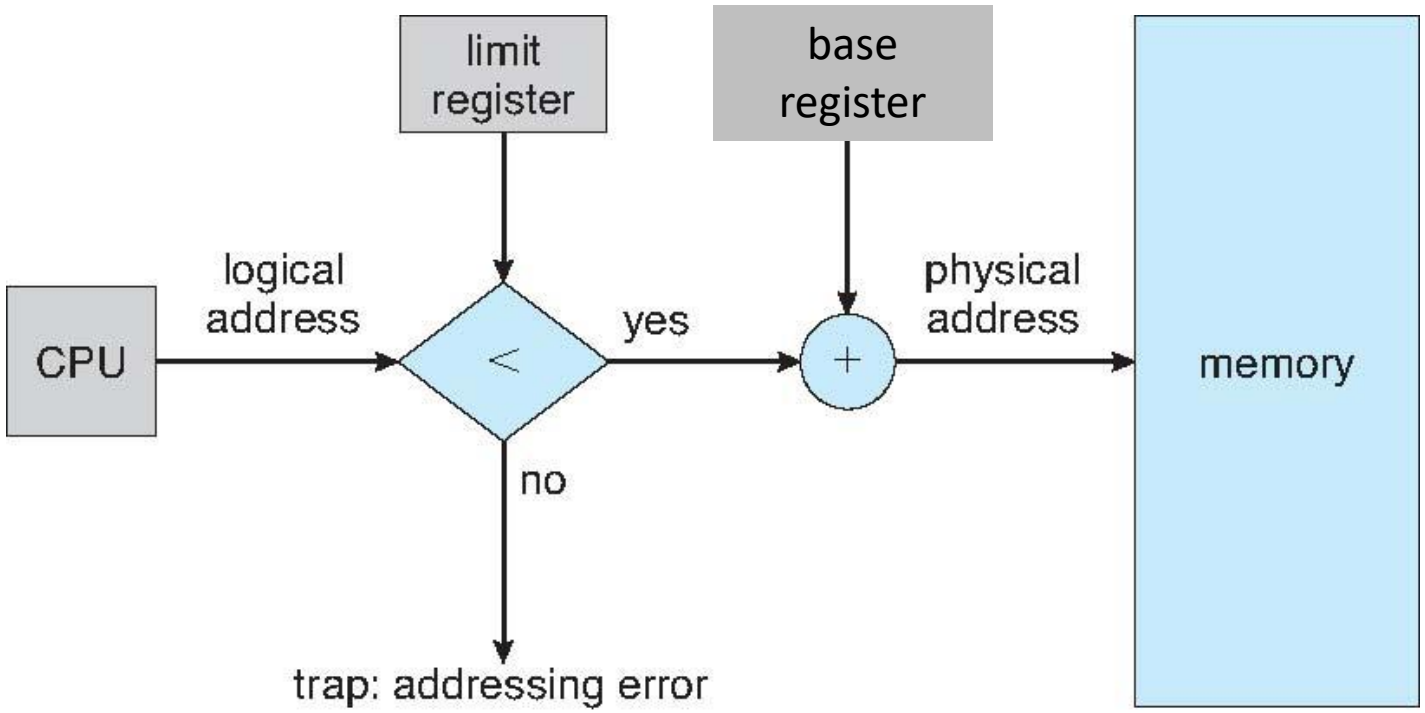
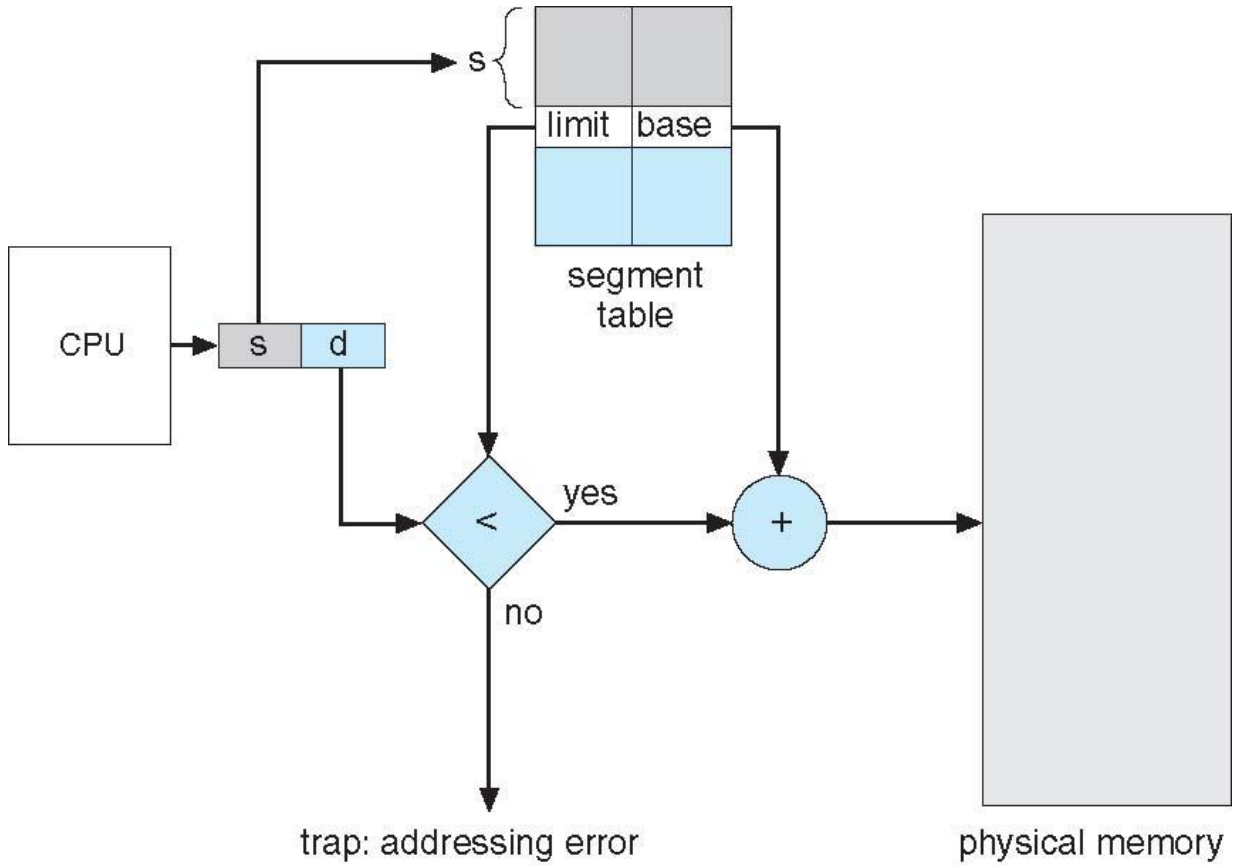
Virtualizing Memory: Smaller Page Tables

Sridhar Alagar

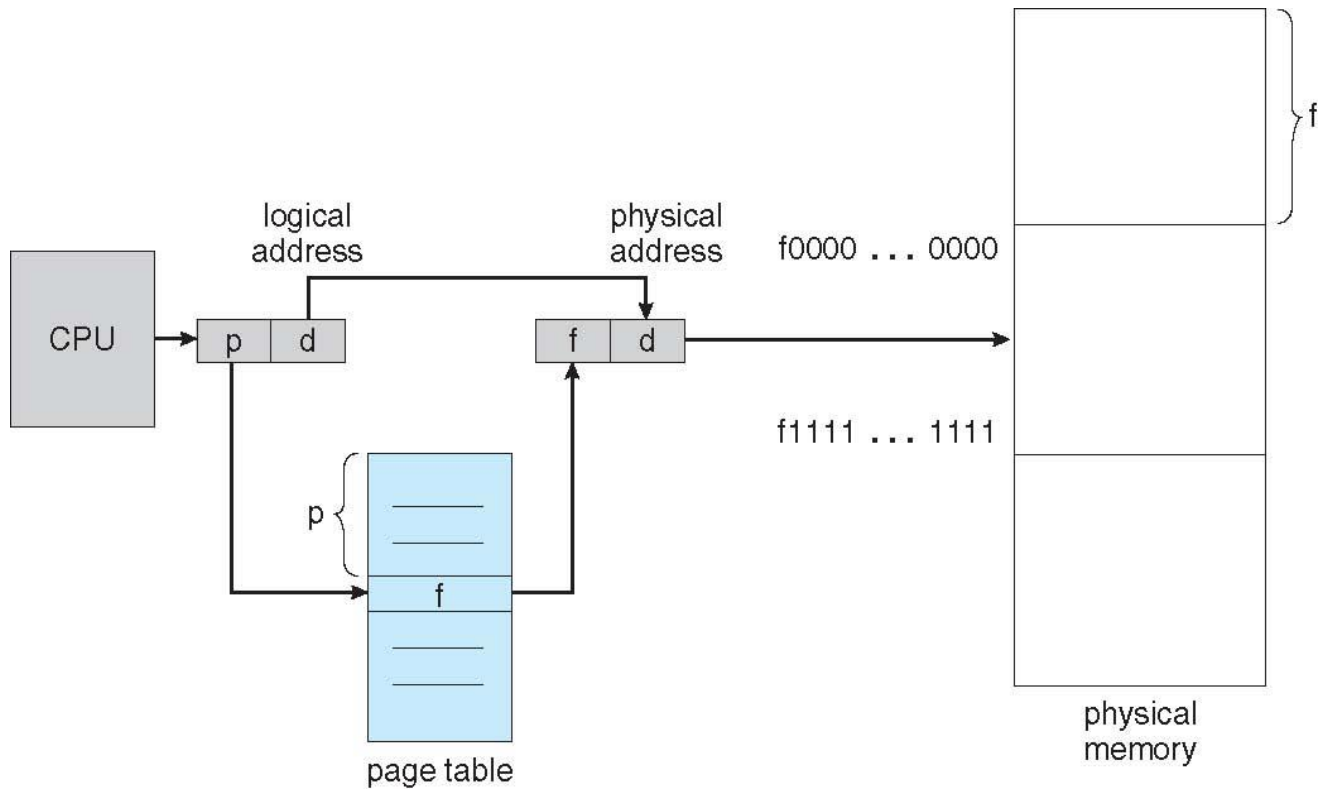
Translation with Base and Limit



Segmentation



Paging


CPU
logical address
p
d
physical address
f
d
f0000 . . . 0000
f1111 . . . 1111
f
p
f
page table
physical memory

Disadvantages of Paging

1. Additional memory reference to look up in page table (inefficient)
   - Page table must be stored in memory
   - MMU stores only base address of page table
   - **Avoid extra memory reference for lookup with TLBs**

2. Storage for page tables may be substantial
   - Simple page table: Requires PTE for all pages in address space
     - Entry needed even if page not allocated
   - Requires contiguous memory space

How big are page tables?

1. PTE's are **2 bytes**, and **32** possible virtual page numbers

   32 * 2 bytes = 64 bytes

2. PTE's are **2 bytes**, virtual addr are **24 bits**, pages are **16 bytes**

   2 bytes * 2^(24 – lg 16) = **2^21 bytes** (2 MB)

3. PTE's are **4 bytes**, virtual addr are **32 bits**, and pages are **4 KB**

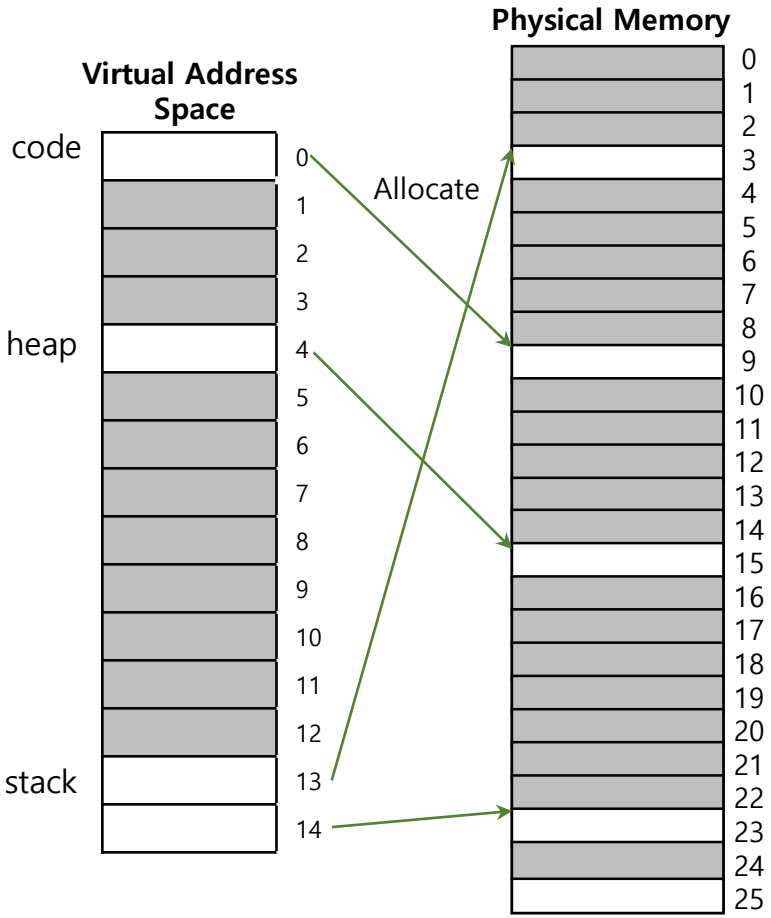
   4 bytes * 2^(32 – lg 4K) = **2^22 bytes** (4 MB)

4. PTE's are **4 bytes**, virtual addr are **64 bits**, and pages are **4 KB**

   4 bytes * 2^(64 – lg 4K) = **2^54 bytes**

Problem?

- Most of the page table is unused, full of invalid entries.



A 16KB Address Space with 1KB Pages

PFN	valid	prot	present	dirty
10	1	r-x	1	0
-	0	-	-	-
-	0	-	-	-
-	0	-	-	-
15	1	rw-	1	1
...	...	...	...	...
-	0	-	-	-
3	1	rw-	1	1
23	1	rw-	1	1

A Page Table For 16KB Address Space

Avoid storing invalid PTEs

PFN	valid	prot
10	1	r-x
-	0	-
23	1	rw-
-	0	-
-	0	-
-	0	-
-	0	-
...many more invalid...		
-	0	-
-	0	-
-	0	-
-	0	-
28	1	rw-
4	1	rw-

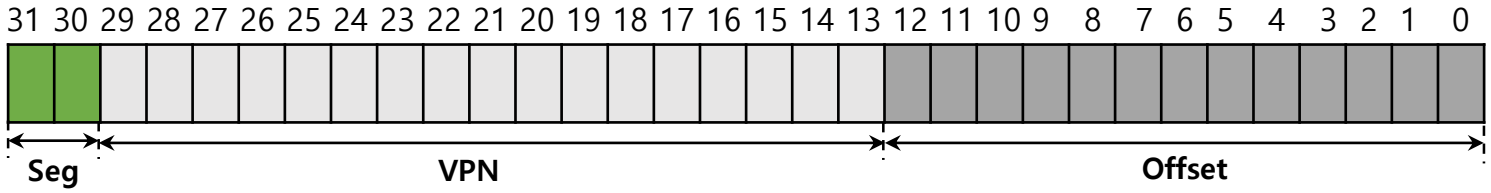
how to avoid storing these?

How did OS avoid allocating holes in physical memory?

Segmentation

Combine Paging and Segmentation

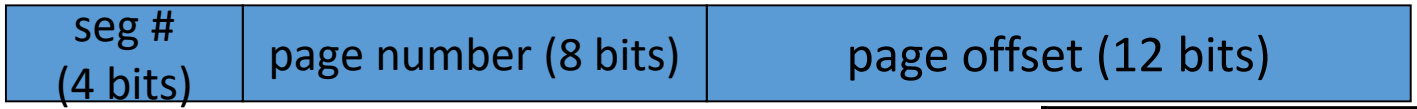
- Divide address space into segments (code, heap, stack)
- Divide each segment into fixed sized pages
- Logical address divided into 3 parts



Implementation Paging and Segmentation

- Base of segment
  - points to beginning of page table of that segment
- Bounds of segment
  - end/size of page table

Quiz: Paging and Segmentation



seg	base	bounds	R W
0	0x002000	0xff	1 0
1	0x000000	0x00	0 0
2	0x001000	0x0f	1 1

```
0x002070 read:    0x004070
0x202016 read:    0x003016
0x104c84 read:      error
0x010424 write:     error
0x203568 read:    0x02a568
```

...	
0x01f	0x001000
0x011	
0x003	
0x02a	
0x013	
...	
0x00c	0x002000
0x007	
0x004	
0x00b	
0x006	
...	

Advantages of Paging and Segmentation

- Advantages of Segments
  - Supports sparse address spaces
    - Decreases size of page tables
    - If segment not used, no need for page table
- Advantages of Pages
  - No external fragmentation
  - Segments can grow without any reshuffling
- Advantages of Both
  - Increases flexibility of sharing
    - Share either single page or entire segment. How?

Disadvantages of Paging and Segmentation

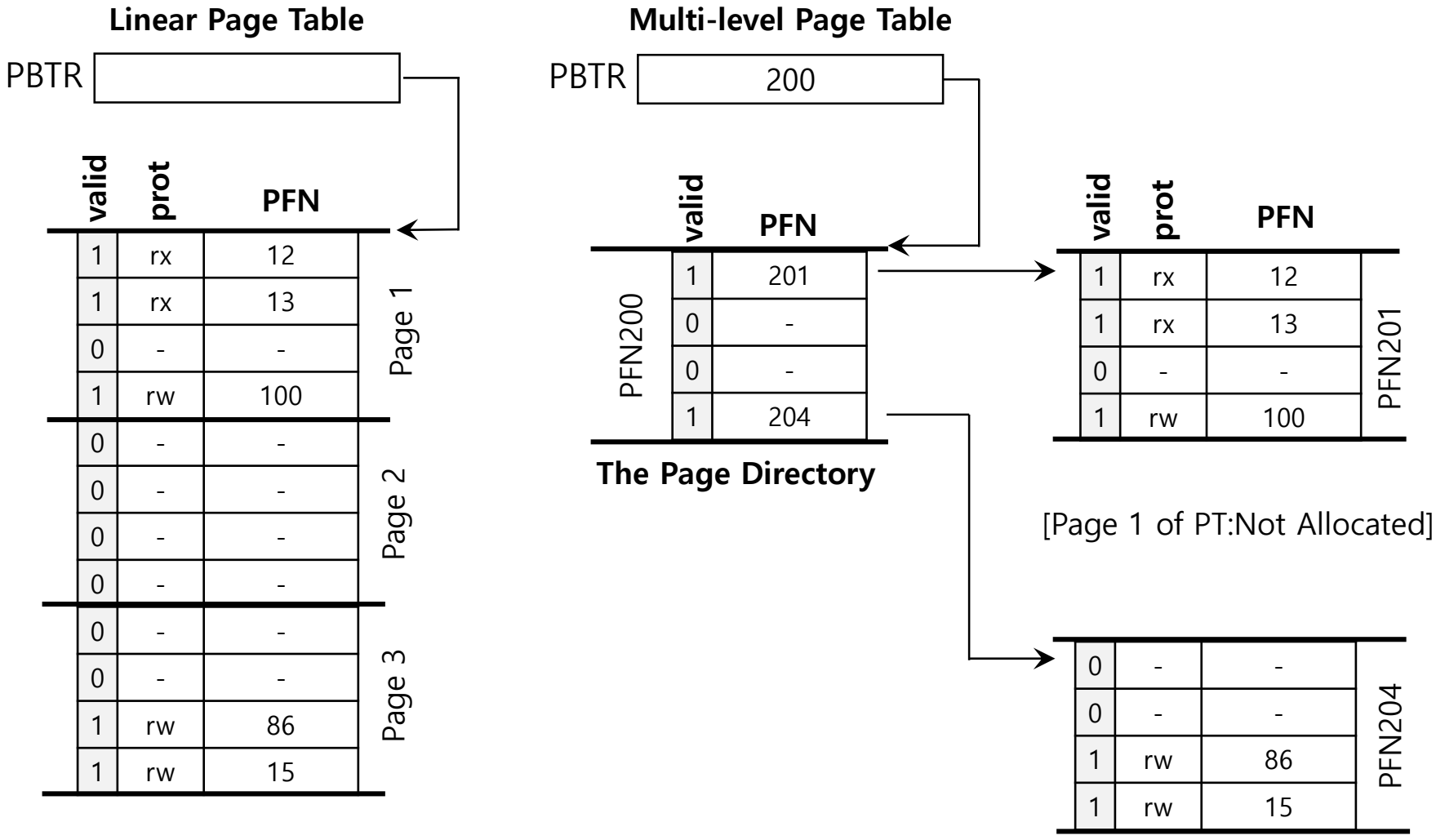
- Potentially large page tables (for each segment)
  - Must allocate each page table contiguously
  - More problematic with more address bits
  - Page table size?
    - Assume 2 bits for segment, 18 bits for page number, 12 bits for offset

Each page table size:
= Number of entries * size of each entry
= Number of pages * 4 bytes
= 2^18 * 4 bytes = 2^20 bytes = 1 MB!!!

How to allocate memory for page table?

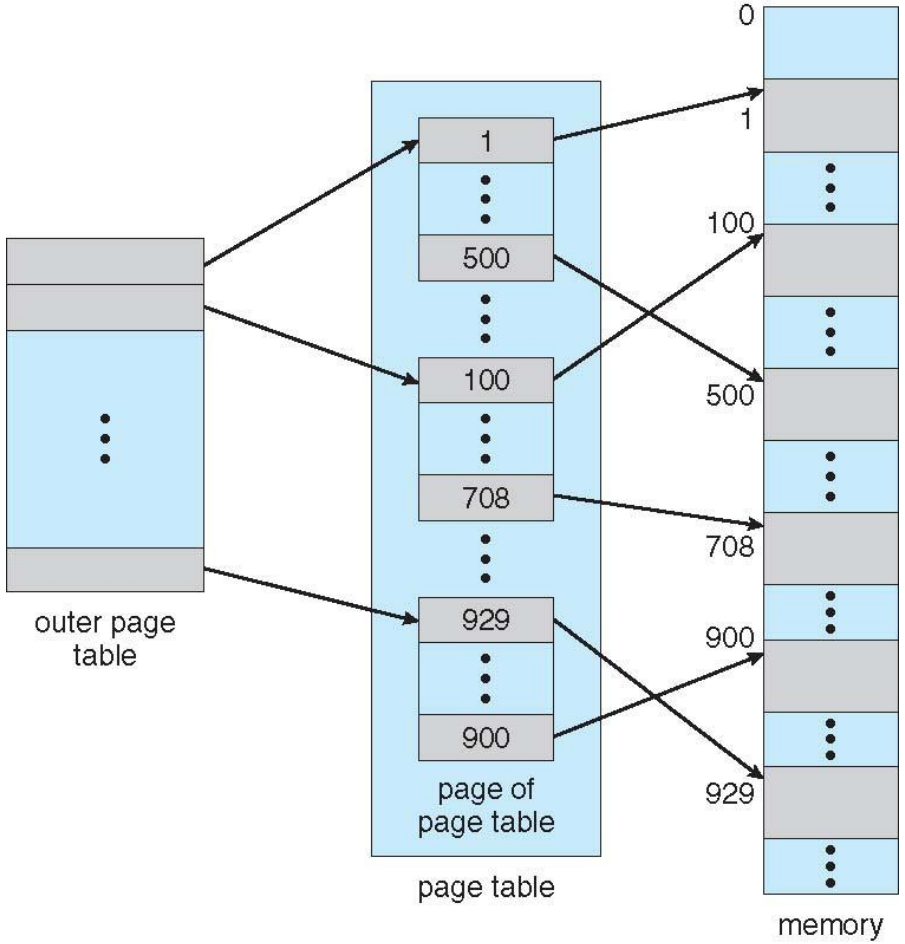
- Physical page size is 4K
- Page table size can be > 1MB
- How to allocate memory non-contiguously?
- Page the page table

Page the page table



Linear (Left) And Multi-Level (Right) Page Tables

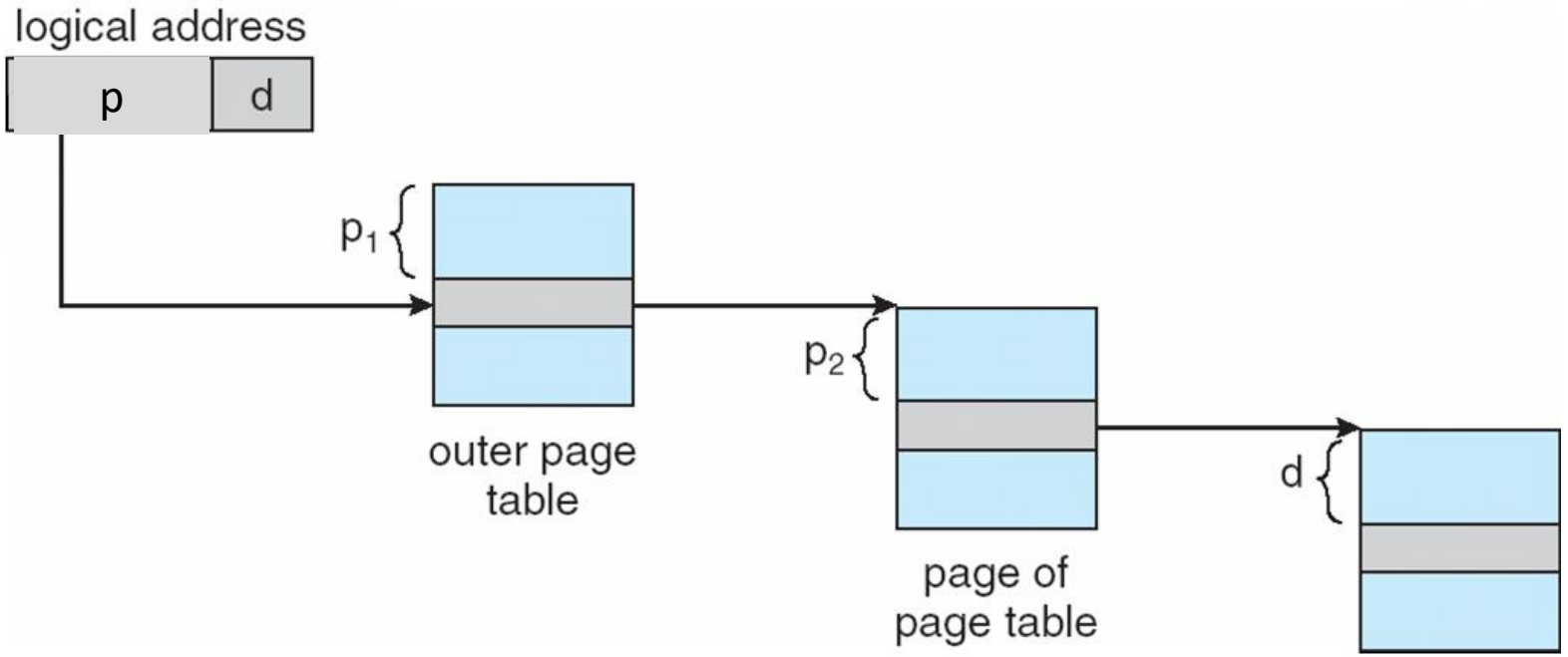
Two-level Page Table Scheme



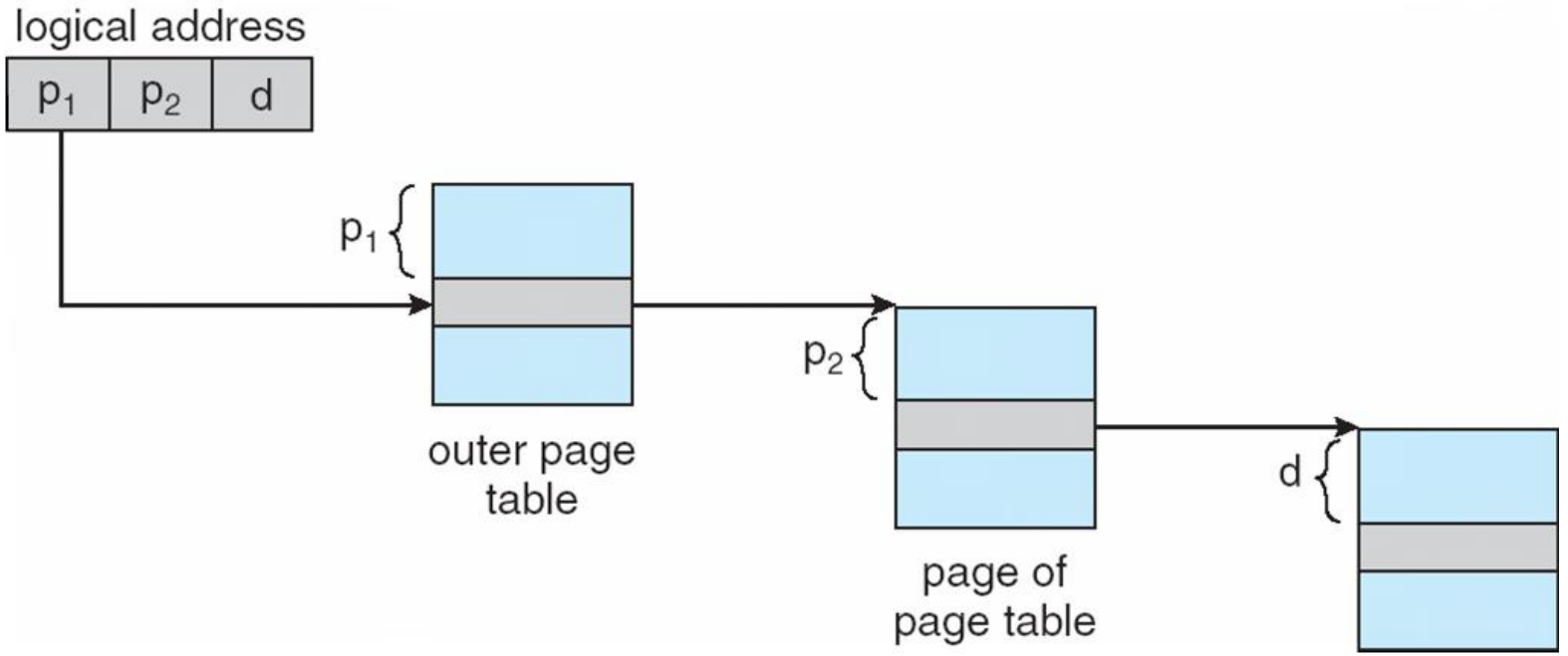
Multi-level Page Table

- Turns a linear page table to a tree
- Divide the page table into multiple pages (page-sized page tables)
- Create a page directory (outer level PT) to track the pages of the page table (inner-level PTs)

Multi-level Address Translation Scheme

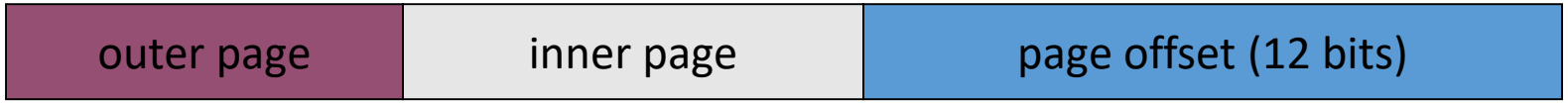


Multi-level Address Translation Scheme



How many bits for each paging level?

32-bit address:



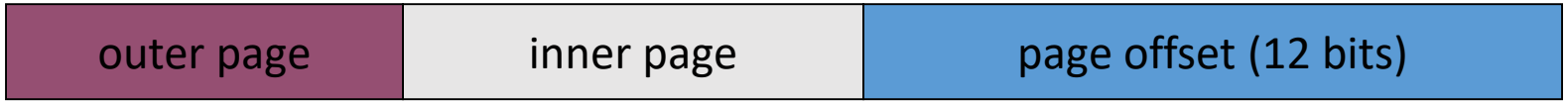
- Each inner page table fits within a page
- number of PTE = page size/ PTE size
  - Assume PTE size = 4 bytes
  - Page size = 2^12 bytes = 4KB
  - number PTE = 2^12 bytes/ 2^2 bytes
  - → number PTE = 2^10
  - → # bits for selecting inner page = 10

Remaining bits for outer page:

- 32 – 10 – 12 = 10 bits

How many bits for each paging level?

32-bit address:



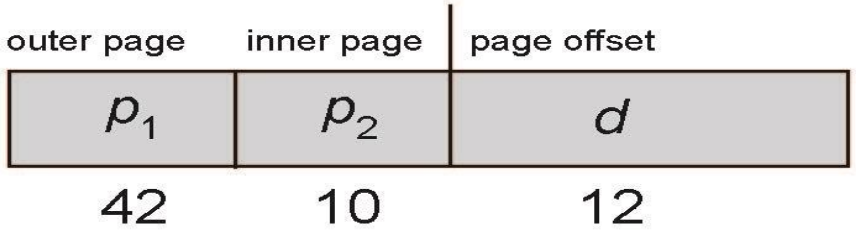
- Outer page table has one entry per inner page table
- Number of inner page tables = linear page table size/ page size
  - Assume PTE size = 4 bytes
  - Linear page table size = # PTE * PTE size
  - Linear page table size = 2^20 * 4 bytes = 2^22 bytes
- Number of inner page tables = 2^22 bytes/ 2^12 bytes
  - → # bits for selecting outer page = 10

Remaining bits for inner page:

- 32 – 10 – 12 = 10 bits

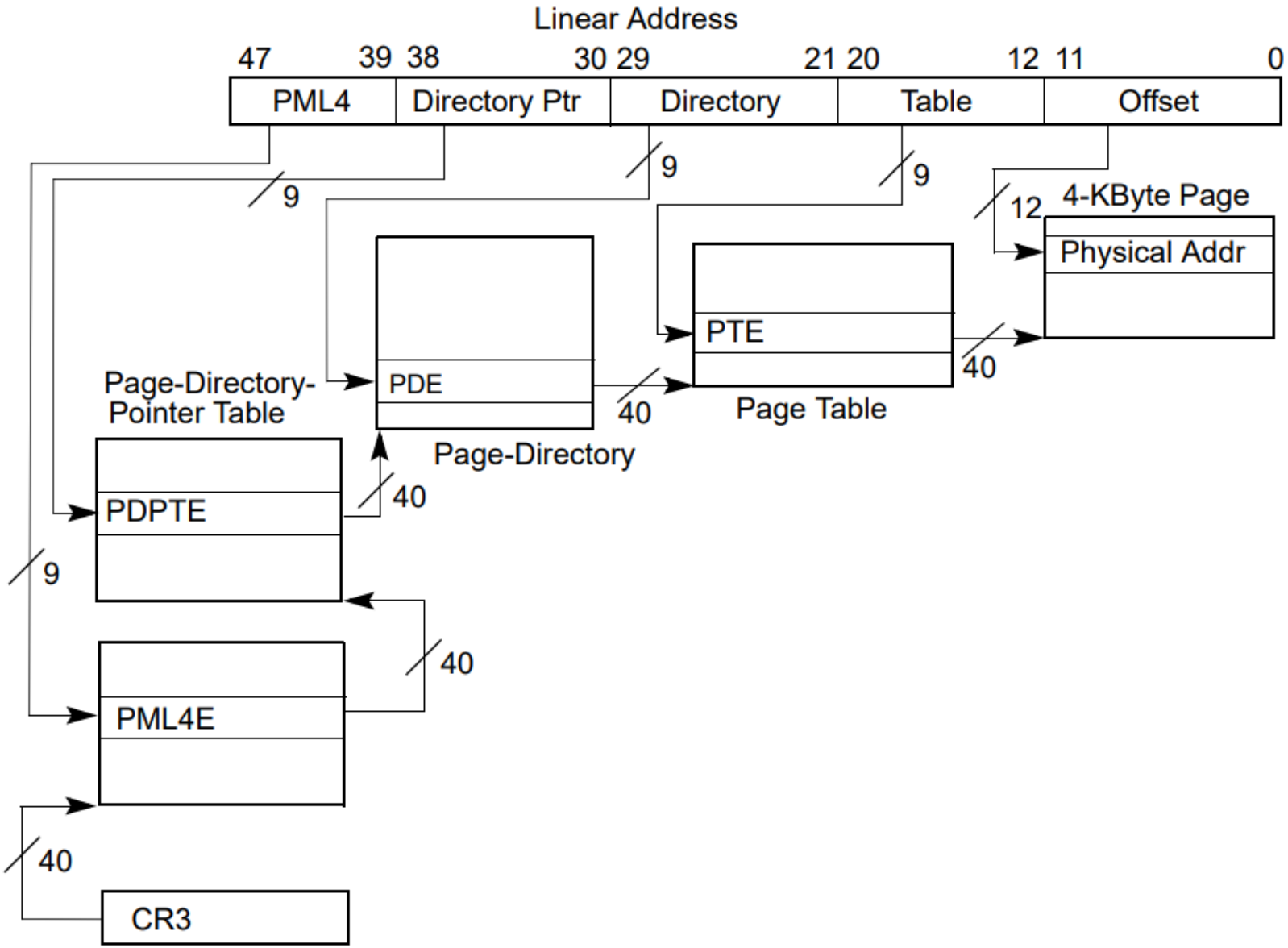
64-bit Address Space

- Page size is 4 KB (2^{12})
  - page table has 2^{52} entries
  - In two level scheme, inner page tables could be 2^{10} 4-byte entries

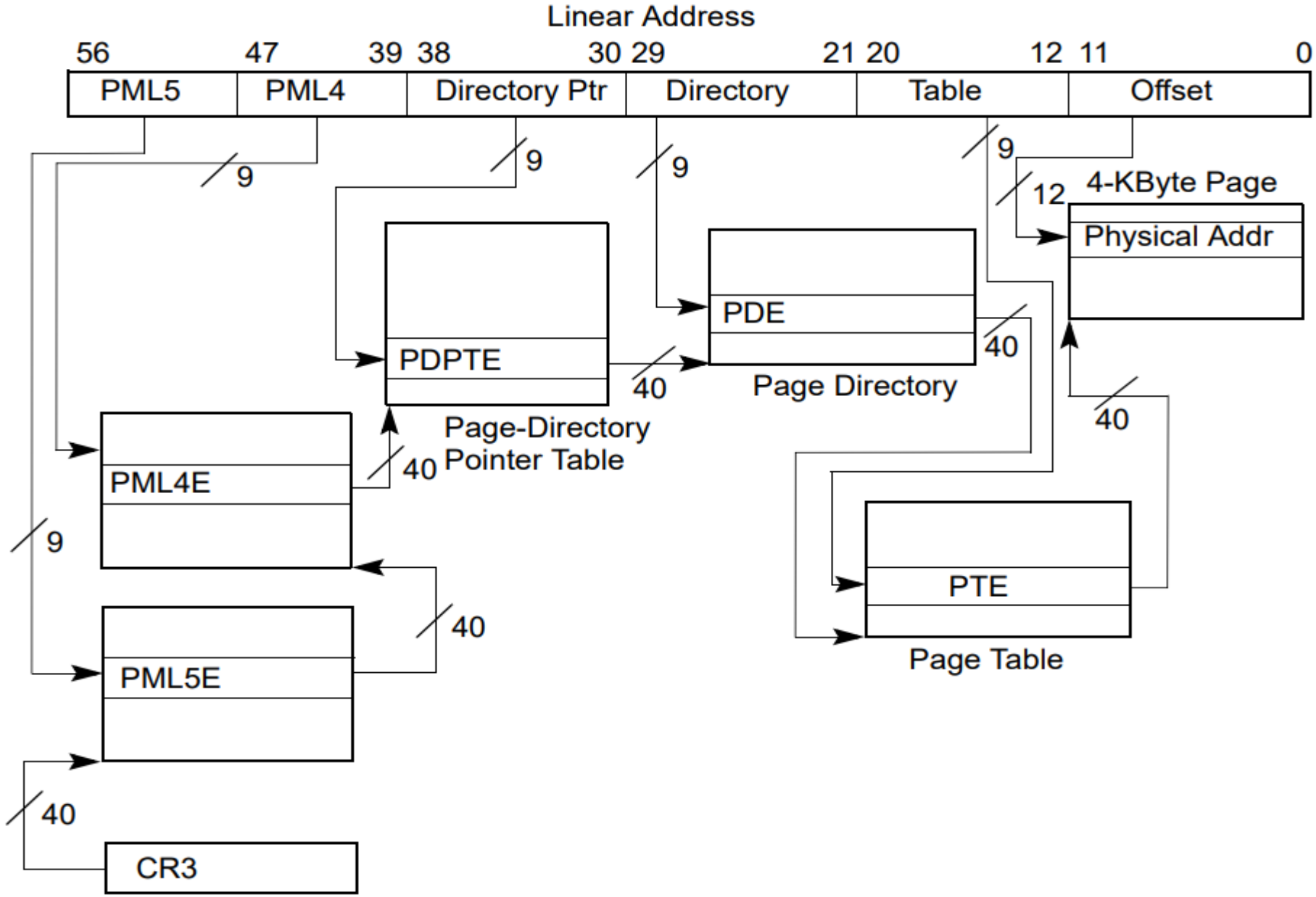


  - Outer page table has 2^{42} entries or 2^{44} bytes
  - One solution is to add a 2^{nd} outer page table
  - The 2^{nd} outer page table is still 2^{34} bytes in size
    - And possibly 4 memory access to get to one physical memory location

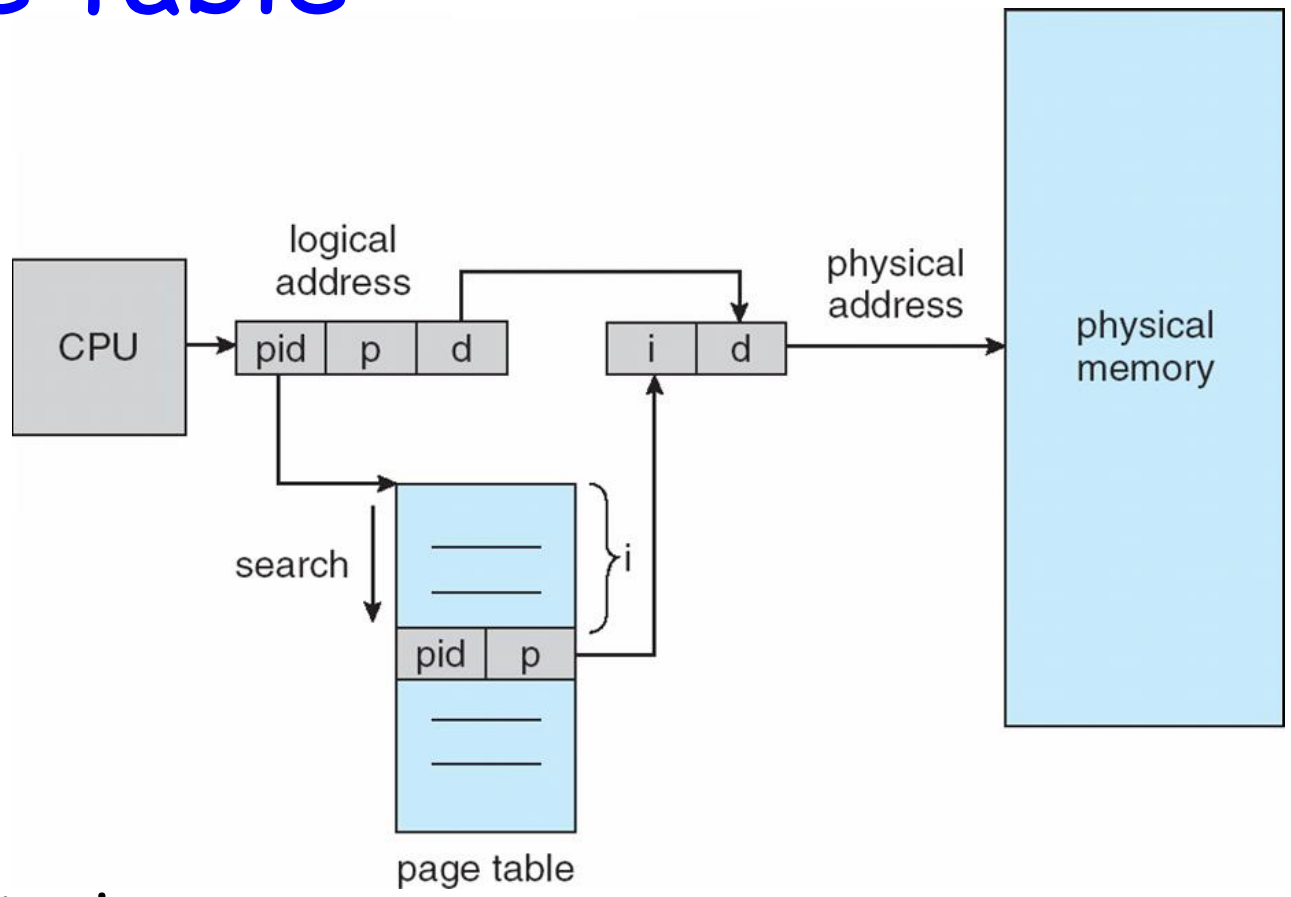
Intel x86-64 (4 level)



Intel x86-64 (5 -level)

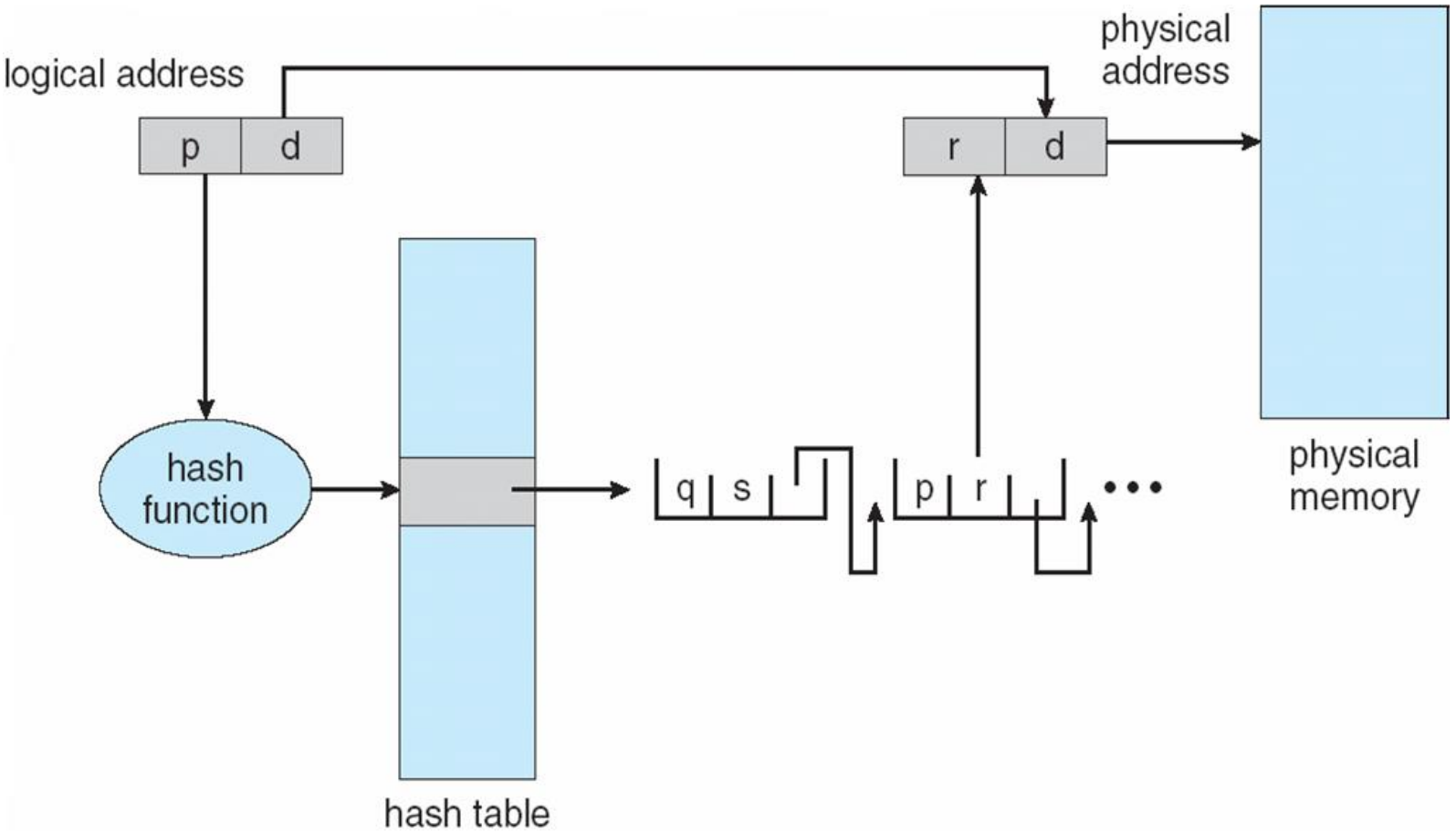


Inverted Page table



One entry for each physical page

Hashed Page Table



Avoid Simple linear page table

- Use complex page table instead of big array
- Inverted page table
- Hashing
- TLB should be software managed