Memory Management(2)

Dept. of Computer Science

Hanyang University

Structure of the Page Table

- Hierarchical Paging
  - Break up the logical address space into multiple page tables
- Hashed Page Tables
- Inverted Page Tables

Two-Level Page-Table Scheme (Hierarchical Paging)

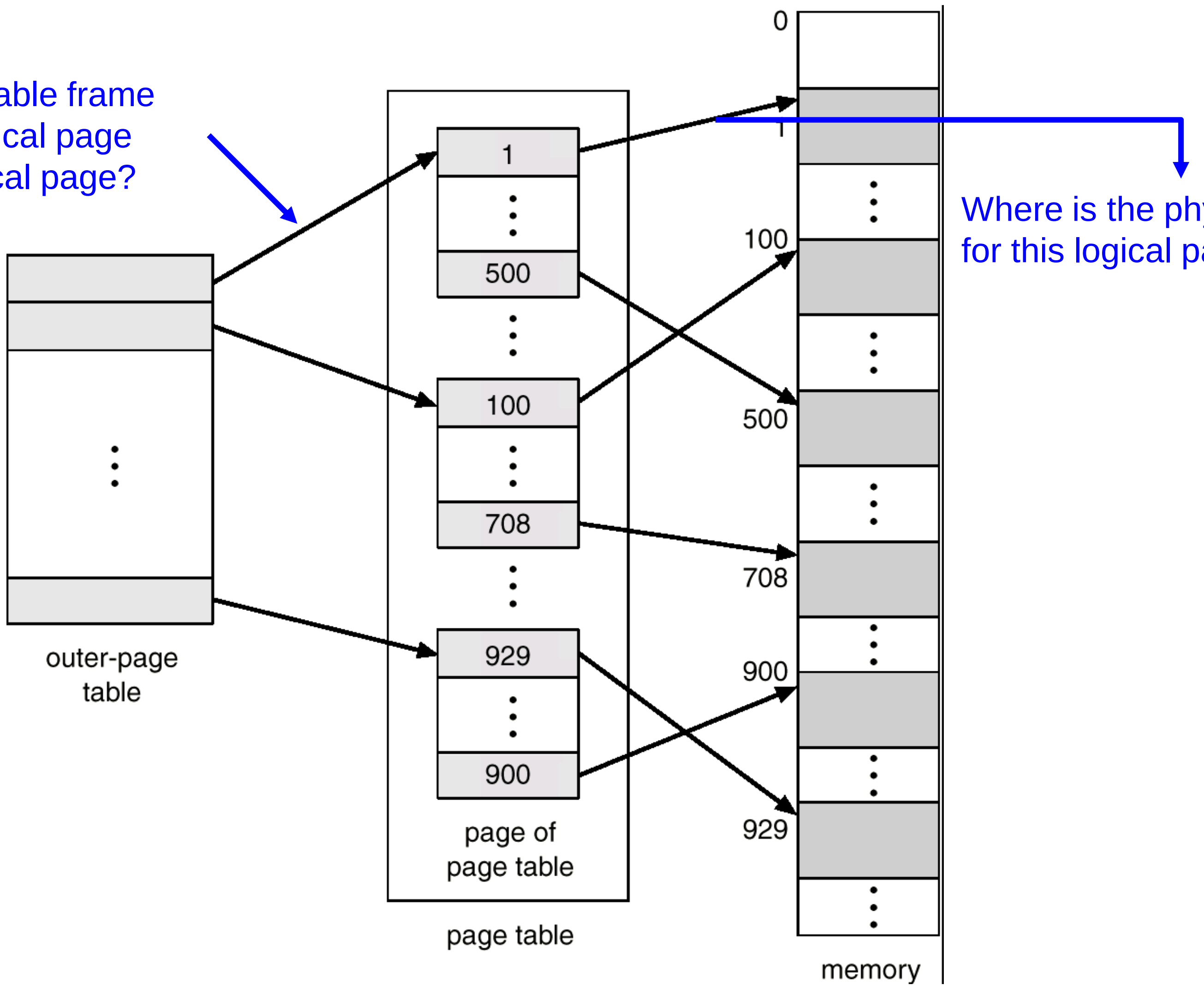
- Problem: program has a large address space
  - For 32 bit address and 4 KB (= 2^{12}) page size (m = 32, n = 12)
  - m-n = 20, hence 2^{20} logical pages
  - 1 million (2^{20}) page table entries
  - 4 MB (2^{22} bytes) page table size (if each entry is 4 Bytes)
  - To store 4 MB page table, 1 K (= 2^{10}) page frames needed per a process

- Solution
  - Store entire page table in disk
  - Page table itself is on-demand loaded in a page unit ➔ Needs page table for page table

Two-Level Page-Table Scheme

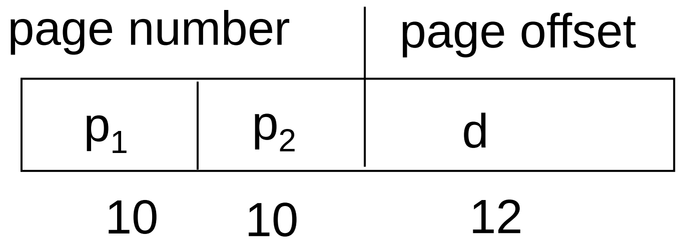
Where is the page table frame that stores the physical page number for this logical page?

Where is the physical page for this logical page?



Two-Level Paging Example

- A logical address (on ***32-bit*** machine with ***4K page*** size) is divided into:
  - a ***page number*** consisting of ***20 bits***
  - a ***page offset*** consisting of ***12 bits***
- Since the page table is paged, the page number is further divided into:
  - a ***10-bit page number*** : for 1 K pages (for page table)
  - a ***10-bit page offset*** : for 1 K entries in a page (of page table)
- Thus, a logical address is as follows:

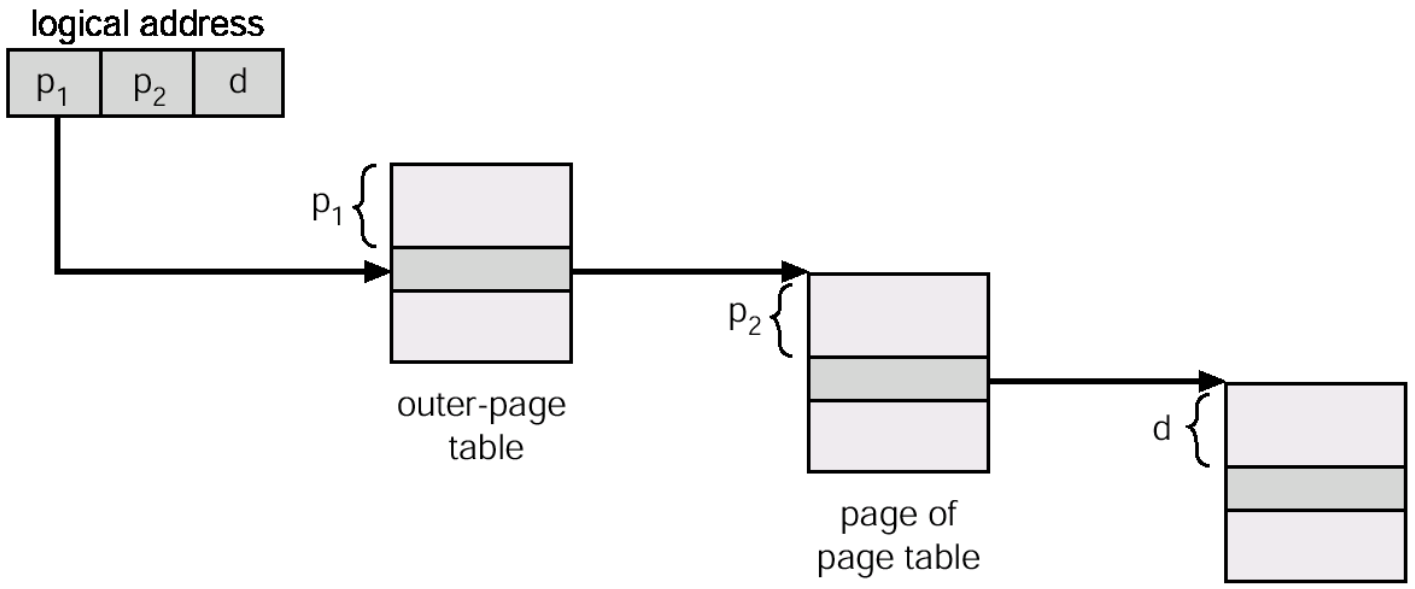


where p_1 is an index into the outer page table, and

p_2 is the displacement within the page of the inner page table

Address-Translation Scheme

- Address-translation scheme for a two-level 32-bit paging architecture



<Forward-mapped page table>

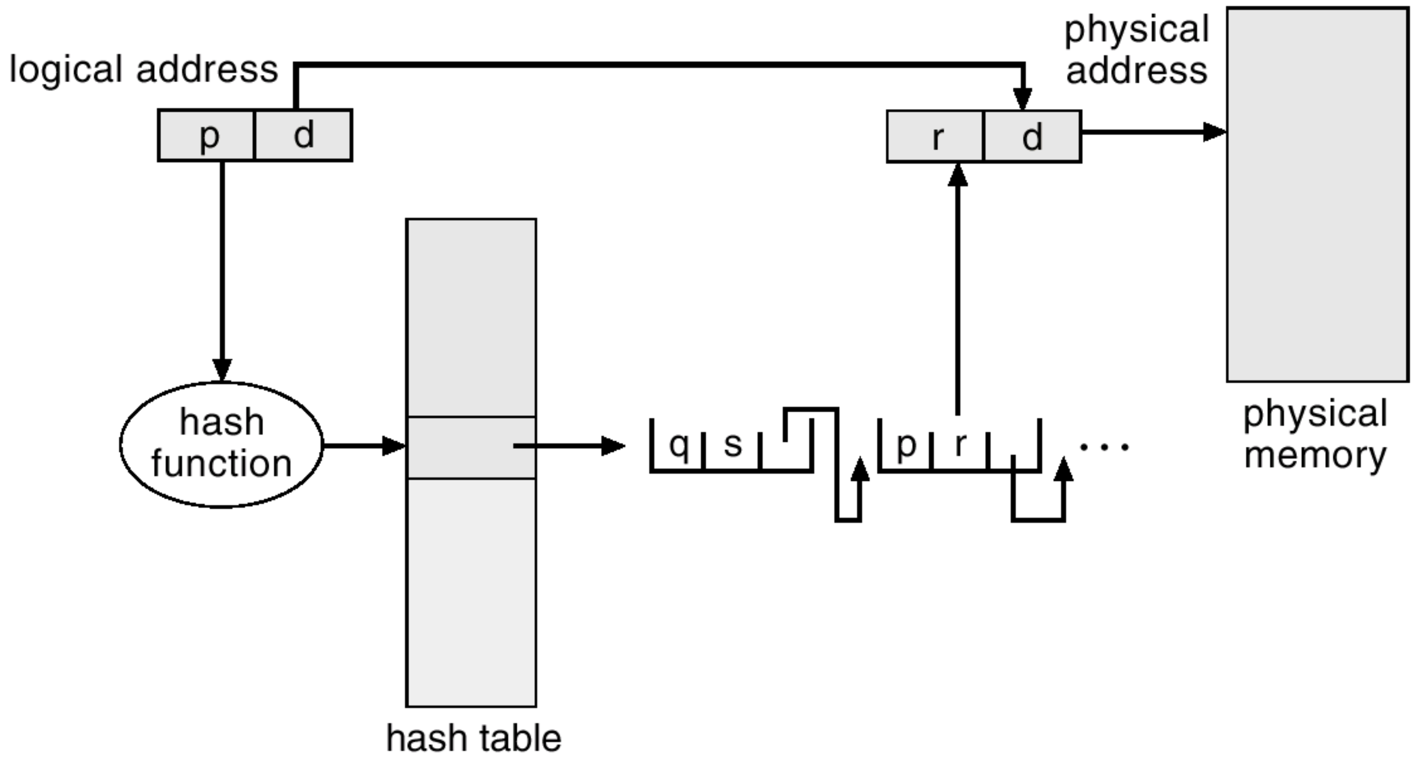
Multilevel Paging and Performance

- For even larger address space,
  - The third page is (like B-tree) not data block, but next level page table
  - Since each level is stored as a separate table in memory, converting a logical address to a physical one may take three, four memory accesses
- Even though time needed for one memory access is quintupled, TLB permits performance to remain reasonable
  - TLB hit rate of 98 percent with 4-level paging yields:

    EAT = 0.98 x 120 + 0.02 x 520 = 128 nanoseconds,
    which is only a 28 percent slowdown in memory access time (assuming TLB access time = 20 ns, memory access time = 100 ns)
- Nevertheless, for 64 bits addressing, 6-levels of paging is required
  - Which is inappropriate
  - In current 64-bits OSes, only 48 bits are used for addressing : 4-levels

Hashed Page Tables

- For address spaces > 32 bits
- The virtual page number is hashed into a page table
  - This page table contains a chain of elements hashing to the same location
  - Virtual page numbers are compared in this chain searching for a match. If a match is found, the corresponding physical frame is extracted.



Inverted Page Table

- Problem: why page table so large?
  - Page table size is proportional to the number of pages
  - One page-table-entry per "logical" page
  - However, only a small number of pages are needed (memory resident) at a time

- Cure: let one page-table-entry per "physical" page frame
  - Each page table entry includes process-id
  - One system-wide page table
    - Required number of entries are the number of page frames
    - All processes share the page table

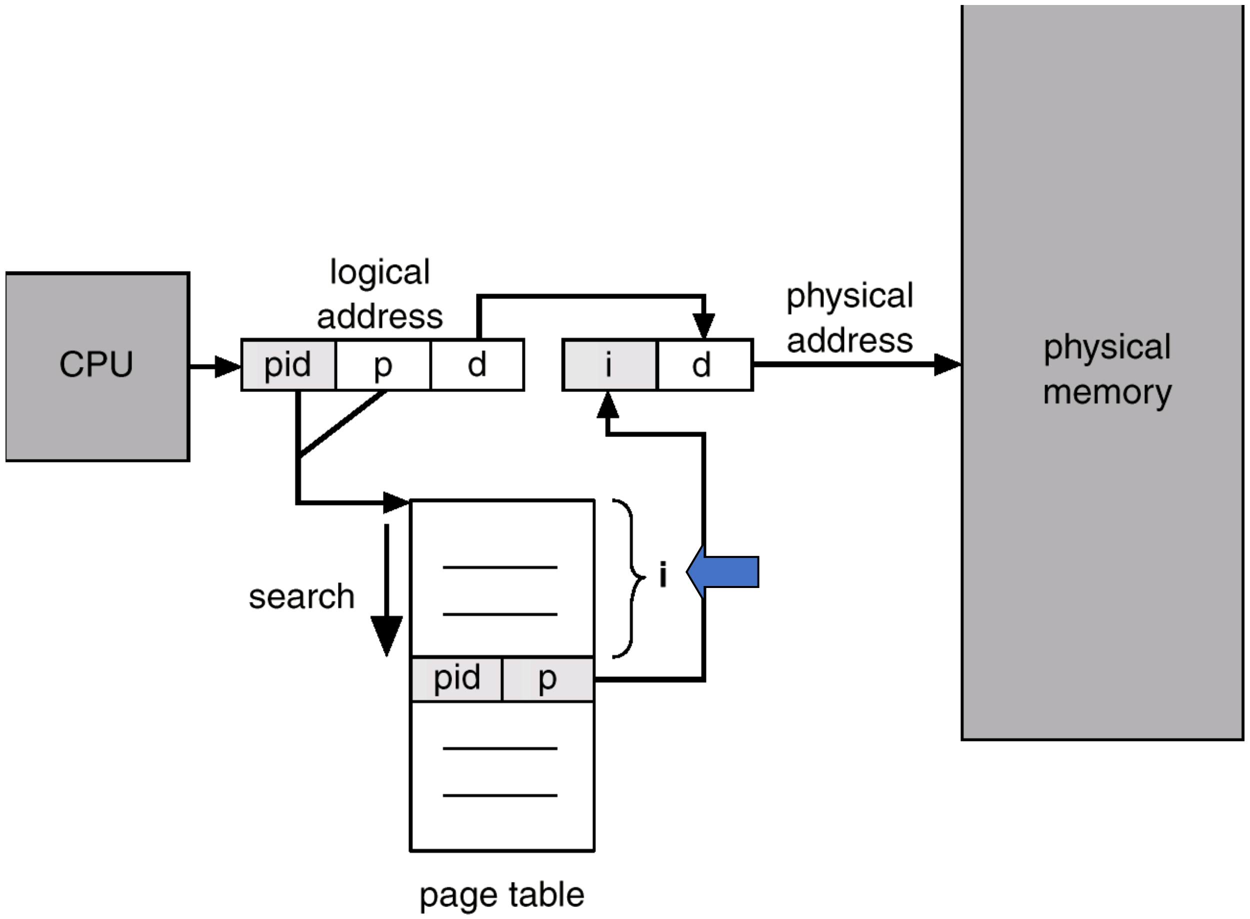
Inverted Page Table Architecture

Drawback:

- Need to search entire table
- Page sharing is impossible

Remedy:

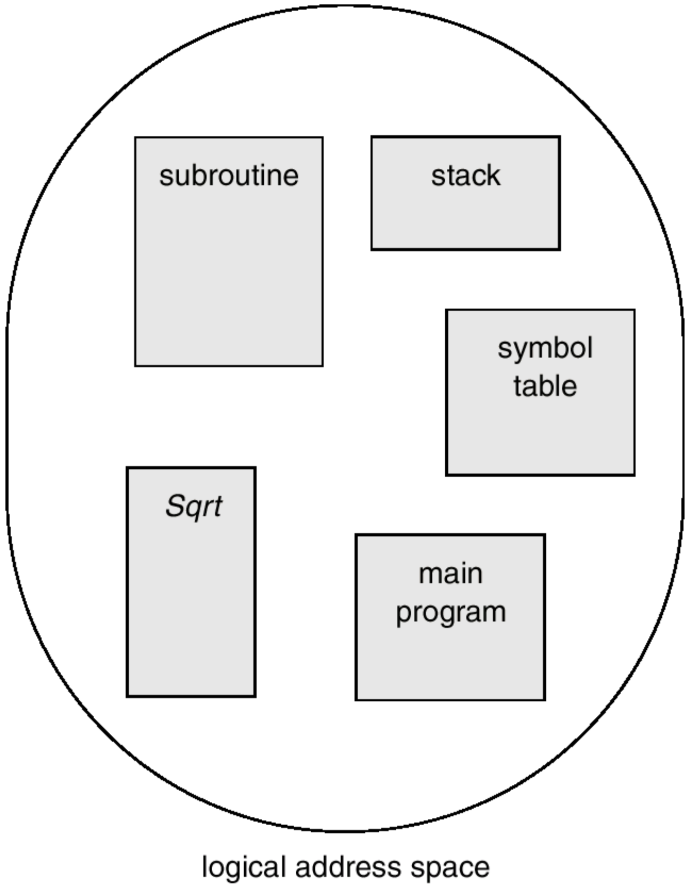
- Use hash table to limit the search : one more memory lookup
- Use TLB for speedup



Segmentation

- Memory management scheme that supports user view of memory
- A program is a collection of variable length segments
- A segment is a *logical unit* such as:
  main (), function, global variables, stack, symbol table, arrays

User's view of a program



Segmentation Architecture

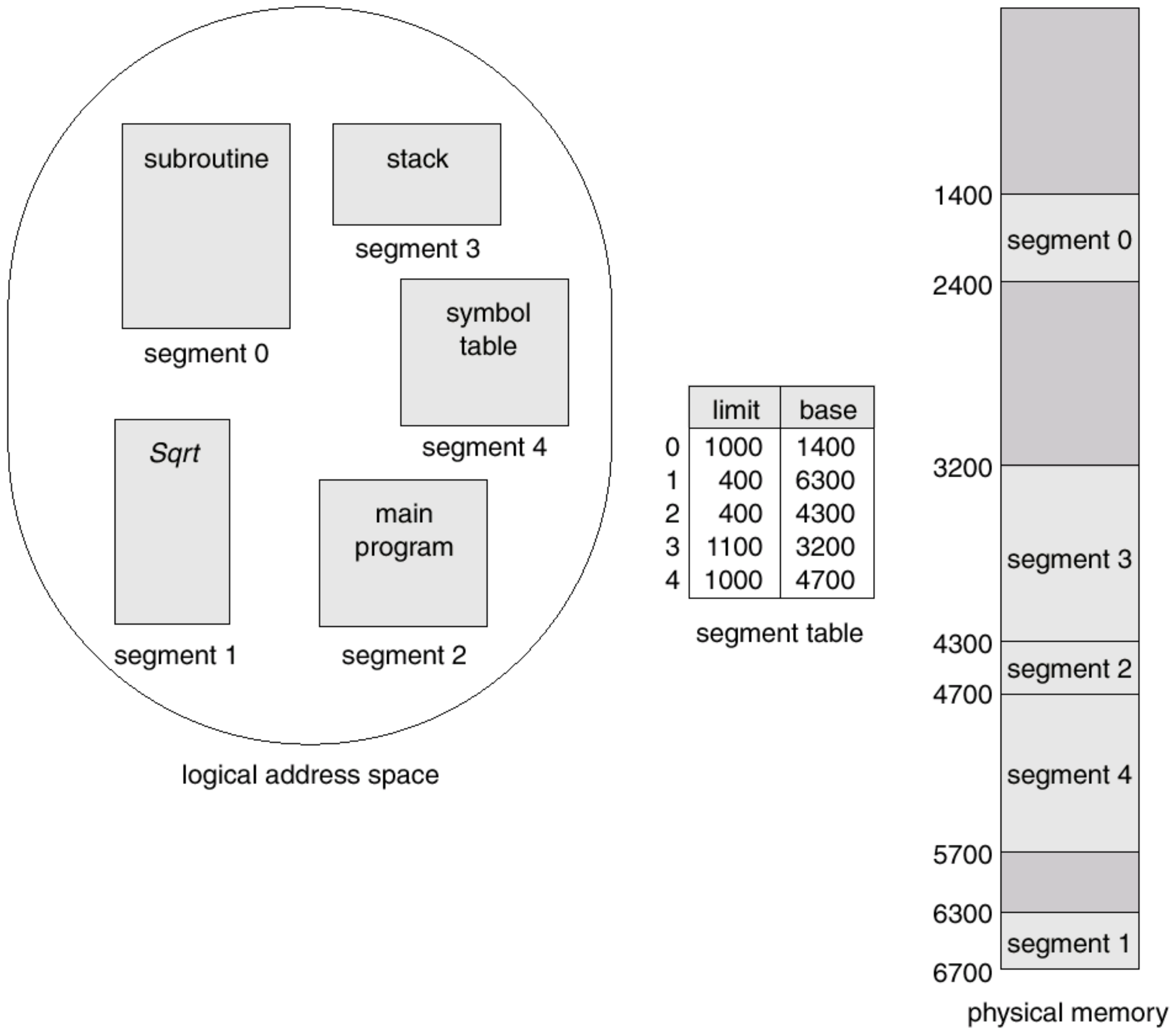
- Logical address structure:

  < segment-number, offset >

- Segment table
  - maps *two dimensional logical address into physical addresses*
  - each table entry has:
    - ***base*** – contains the *starting physical address* where the segments reside in memory
    - ***limit*** – specifies the *length* of the segment
- Segment-table base register (STBR)
  - points to the *segment table's location* in memory
- Segment-table length register (STLR)
  - indicates *number of segments* used by a program;

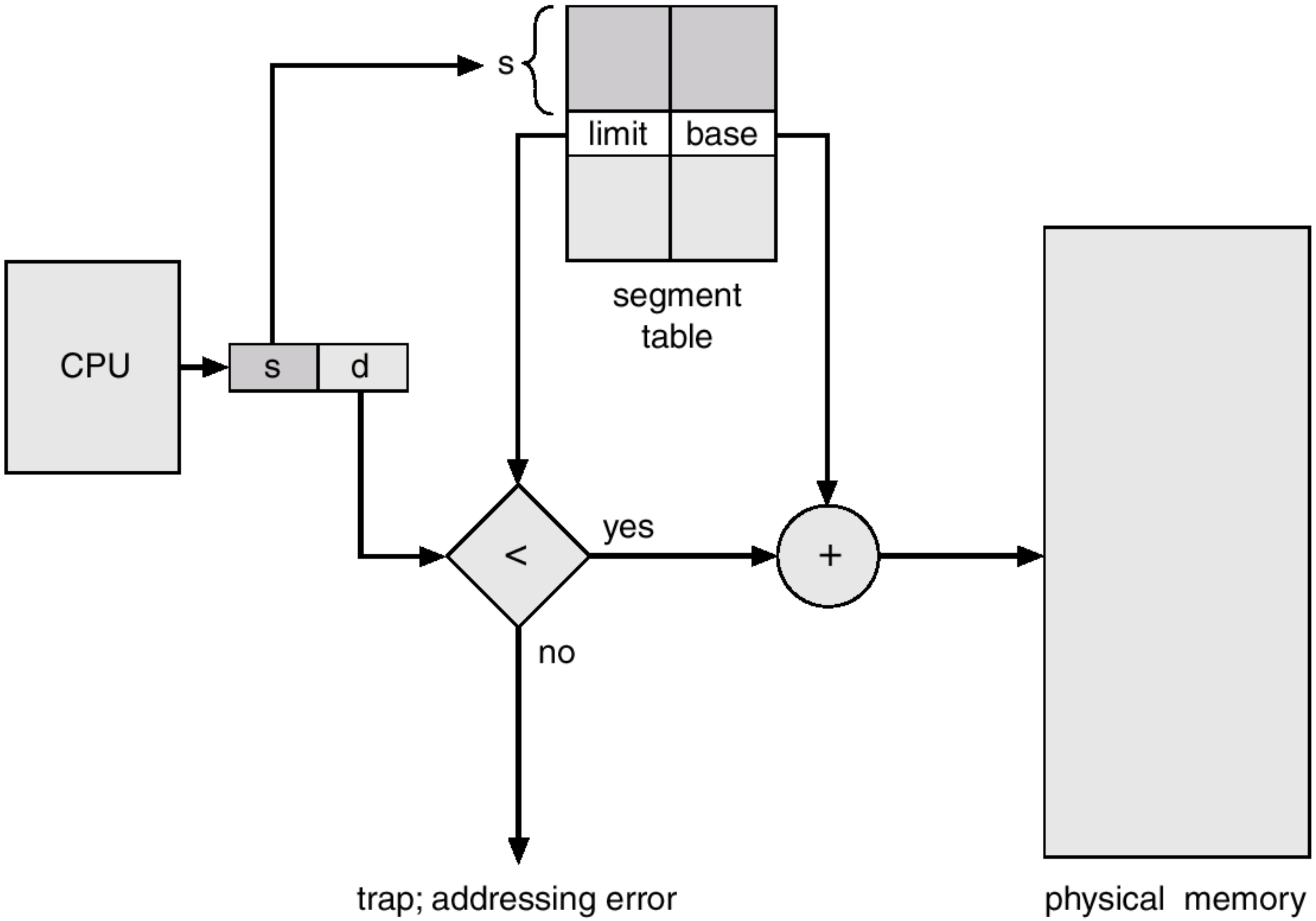
    segment number *s* is legal if *s* < *STLR*

Example of Segmentation



	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

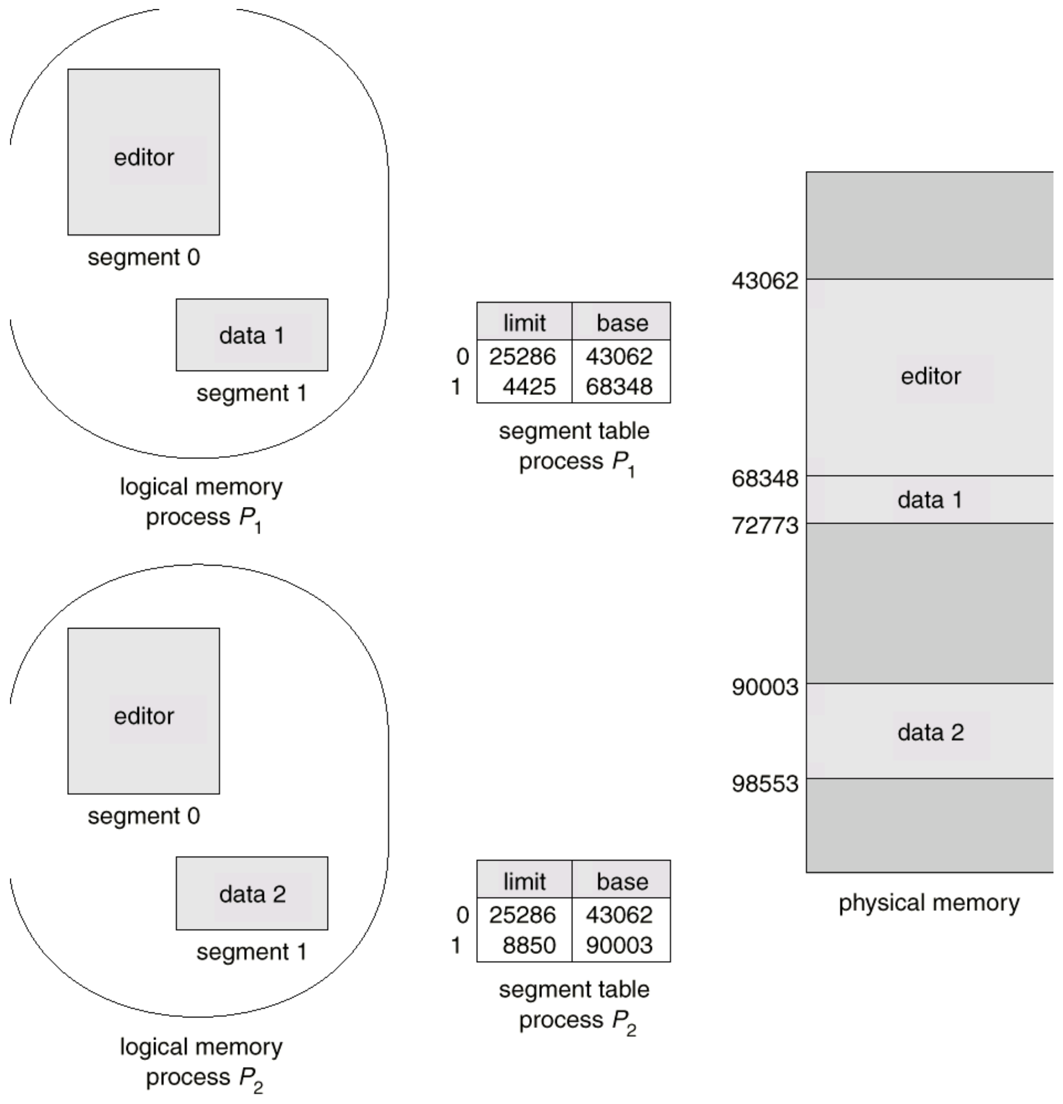
Segmentation Hardware



Segmentation Architecture (Cont.)

- Protection
  - With each entry in segment table associate:
    - Valid/invalid bit = 0 $\Rightarrow$ illegal segment
    - R/W/X bits
- Sharing
  - Protection bits associated with segments: code sharing occurs at segment level
  - Each process should have the same segment number for the shared segment
    - Ex. Self-referencing code segment references itself using segment number and offset
- Allocation
  - Since segments vary in length, memory allocation is a dynamic storage-allocation problem
  - first fit / best fit
  - external fragmentation, but no internal fragmentation

Sharing of Segments



	limit	base
0	25286	43062
1	4425	68348

segment table process P_1

	limit	base
0	25286	43062
1	8850	90003

segment table process P_2

Segmentation with Paging

- The problem of *external fragmentation* can be solved by paging the segments

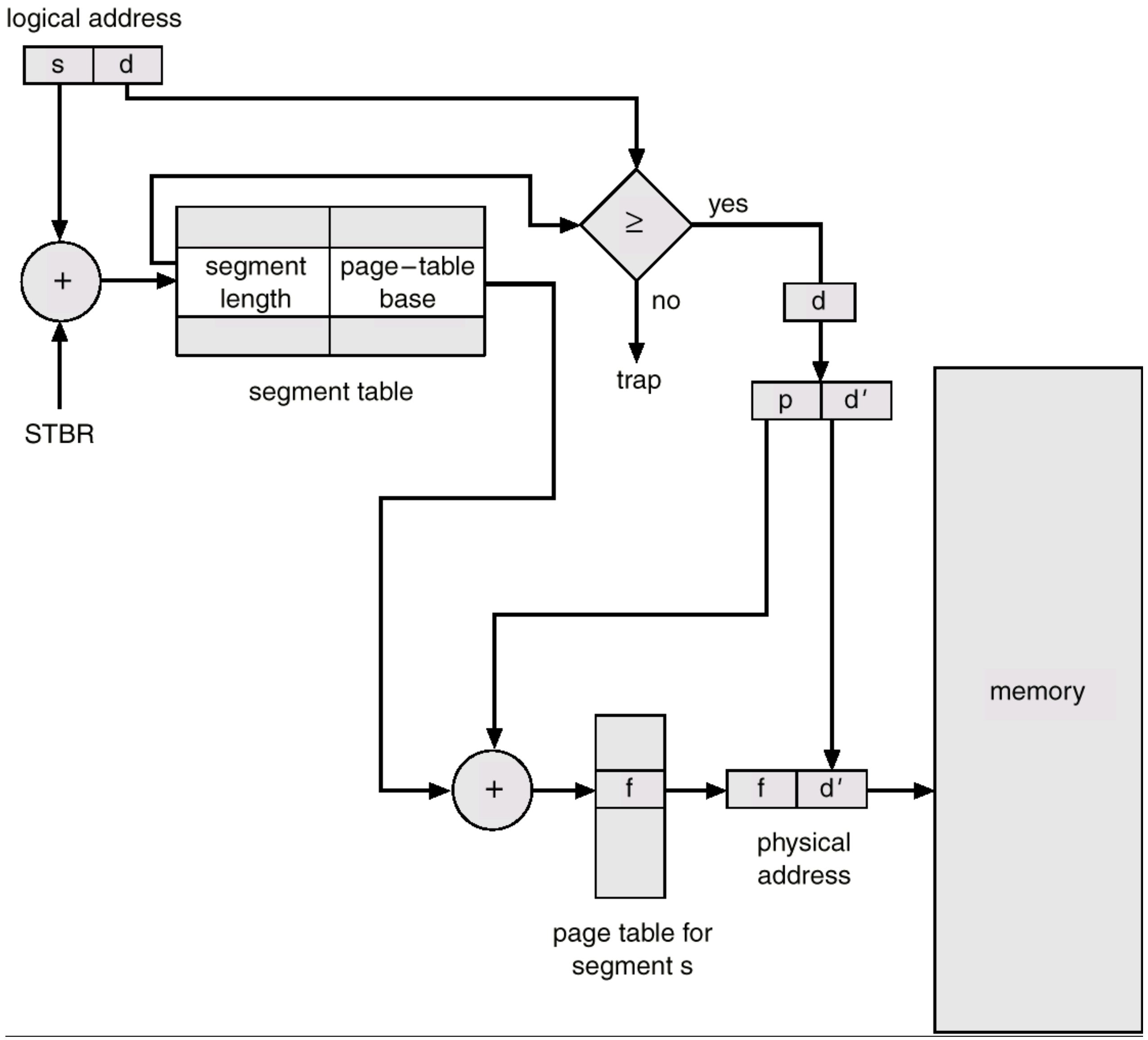
- Solution differs from pure segmentation in that the *segment-table entry* contains
  - *not* the *base address* of the segment
  - *but* rather the *base address of a page table* for this segment

Segment number s	Page number p	Displacement d

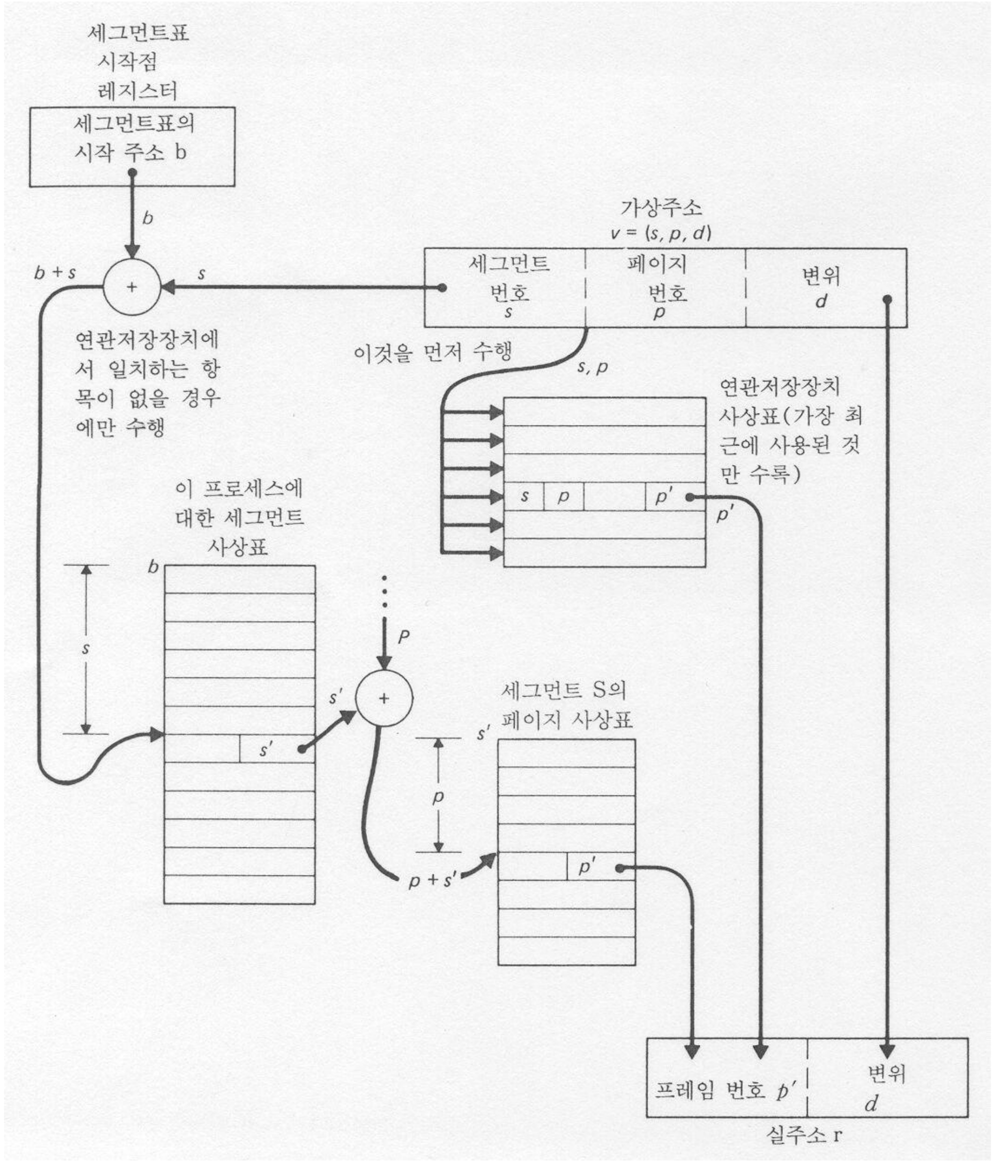
Virtual address v = (s, p, d)

(Virtual address format in a paged and segmented system)

Segmentation with Paging: Address Translation



Segmentation with Paging (TLB incorporated)


세그먼트표
시작점
레지스터
세그먼트표의
시작 주소 b
b
b + s
+
s
가상주소
v = (s, p, d)
세그먼트
번호
s
페이지
번호
p
변위
d
이것을 먼저 수행
s, p
연관저장장치에
서 일치하는 항
목이 없을 경우
에만 수행
연관저장장치
사상표(가장 최
근에 사용된 것
만 수록)
s
p
p′
p′
이 프로세스에
대한 세그먼트
사상표
b
s
s′
s′
+
P
세그먼트 S의
페이지 사상표
s′
p
p + s′
p′
프레임 번호 p′
변위
d
실주소 r