CMP325
Operating Systems
Lecture 21-23

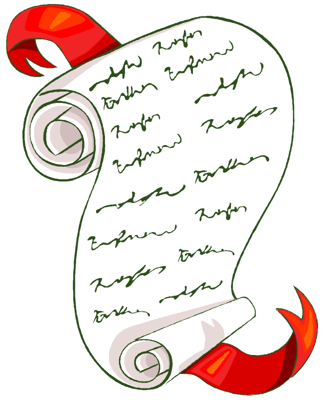
Non-Contiguous Memory Allocation

Muhammad Arif Butt, PhD

Note:
Some slides and/or pictures are adapted from course text book and Lecture slides of
- Dr Syed Mansoor Sarwar
- Dr Kubiatowicz
- Dr P. Bhat
- Dr Hank Levy
- Dr Indranil Gupta

For practical implementation of operating system concepts discussed in these slides, students are advised to watch and practice following video lectures:
OS with Linux:
https://www.youtube.com/playlist?list=PL7B2bn3G_wfBuJ_WtHADcXC44piWLRzr8
System Programming:
https://www.youtube.com/playlist?list=PL7B2bn3G_wfC-mRpG7cxJMnGWdPAQTViW

Today's Agenda



- Review of Previous Lecture
- Introduction to Paging
  - Address Translation in Paging
  - Implementation of Page Table
    - In CPU Registers
    - In Associative Memory
    - In main memory
  - Structure of Page Table
    - Hierarchical
    - Inverted
    - Hashed
- Introduction to Segmentation
- Introduction to Paged Segmentation
- Intel 80386 Address Translation

LOGICAL/VIRTUAL ADDRESS

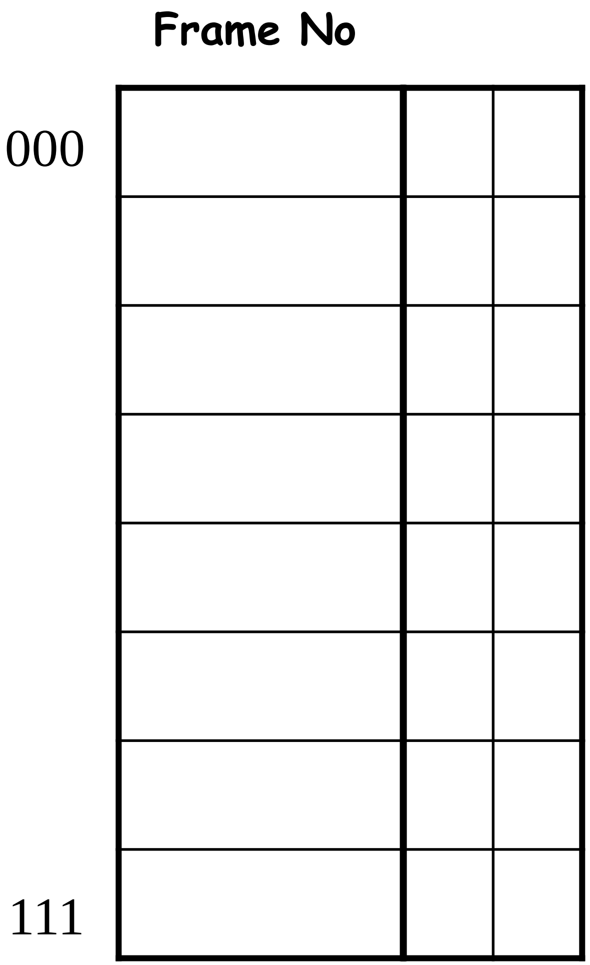
- A Logical address is the address generated by the process/CPU OR is the address used by the process to access its own address space
  - Logical Address ≠ Actual Physical RAM Address
  - When a process access a logical address, MMU hardware translates the Logical Address to Physical Address
  - Operating System determines the mapping from Logical Address to Physical address
- Benefits
  - Isolation
  - Illusion of larger memory space
  - Relocation (A program does not need to know which physical addresses it will use when it is run. Can even change physical location when it is running)

INTRODUCTION TO PAGING

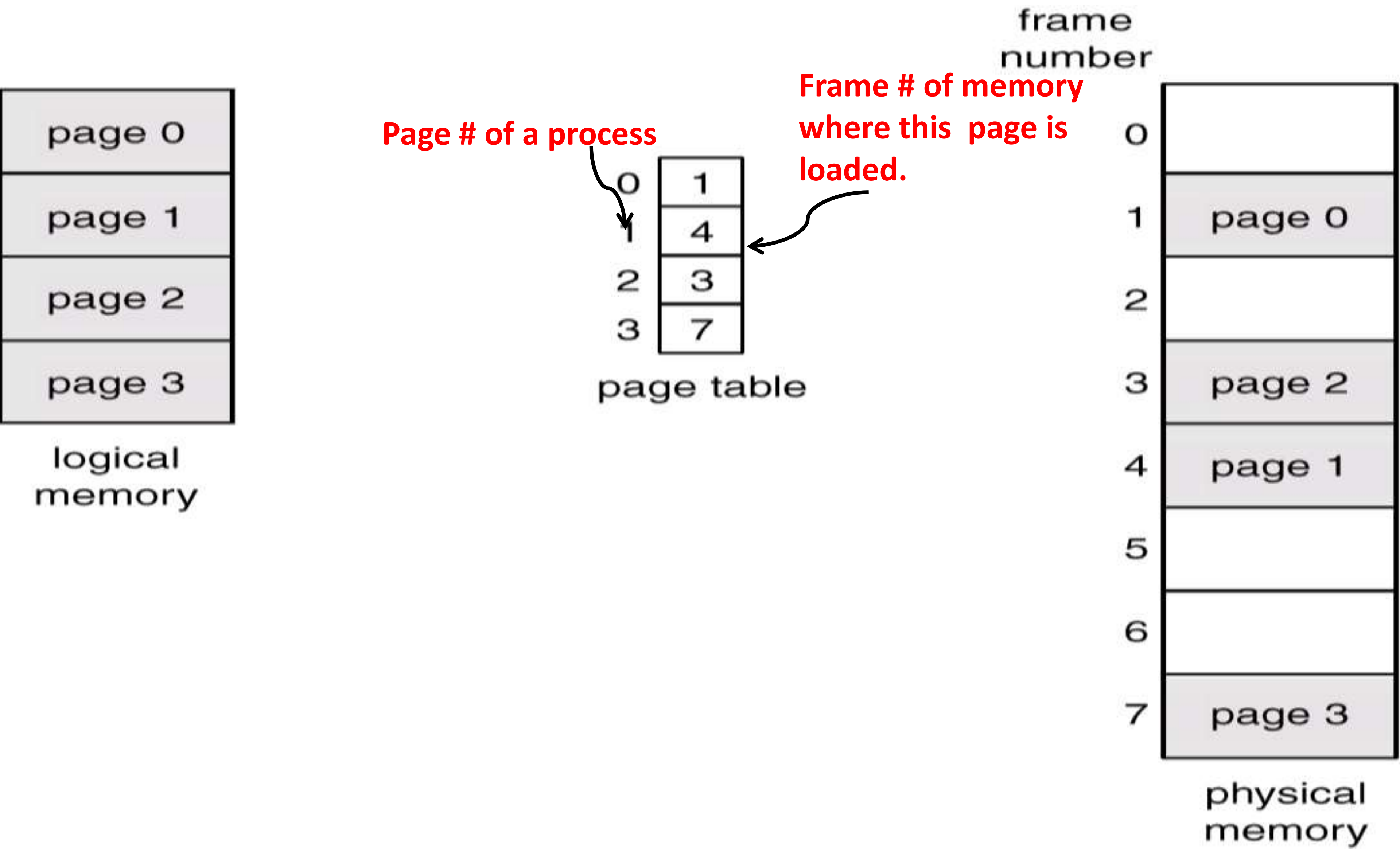
- Paging is like reading a book. At any time we do not need all pages except ones we are reading. The analogy suggest that pages we are reading are in the main memory and the rest can be in the secondary memory
- **Logical/Virtual address space** is the set of addresses that programs use for load and store operations on disk. Logical address space is divided into **pages**
- **Physical address space** is the set of addresses used to reference locations in the main memory. Physical address space is divided into **frames**
- Pages and frames must be of **same size**
- Typical page sizes range from 1KB to 64 KB. (different for different architectures)
- Pages that have been loaded into the main memory form disk are said to have been **mapped** into the main memory
- A program / process is loaded by loading its pages into available not necessarily contiguous frames
- To run a program having **n** pages, find **n** free frames and load the pages into these frames. These frames need not to be contiguous. For example a program comprising of 10 pages need 10 free frames in main memory, which need not to be contiguous. For this to work, we need to store the mapping information of which page is loaded in which frame, in some data structure called "**Page Table**"
- No external fragmentation. Internal Fragmentation in paging is half a page per process

PAGE TABLES

- **Page Tables** are used to keep track of how logical addresses map to physical addresses. Page table entries generally contain the frame number where the particular page is loaded. In addition a PTE may also contain a **modify bit, resident bit, valid bit and protection bits**
- Page table size depends on the maximum number of pages of a process that a CPU support. E.g. if a system support a process of maximum 8 pages then the page table will contain 8 rows (length is debatable). So the address of page table will consist of 3 bits



USE OF PAGE TABLES



LOGICAL & PHYSICAL ADDRESS FORMAT IN PAGING

Logical Address format (p, d)

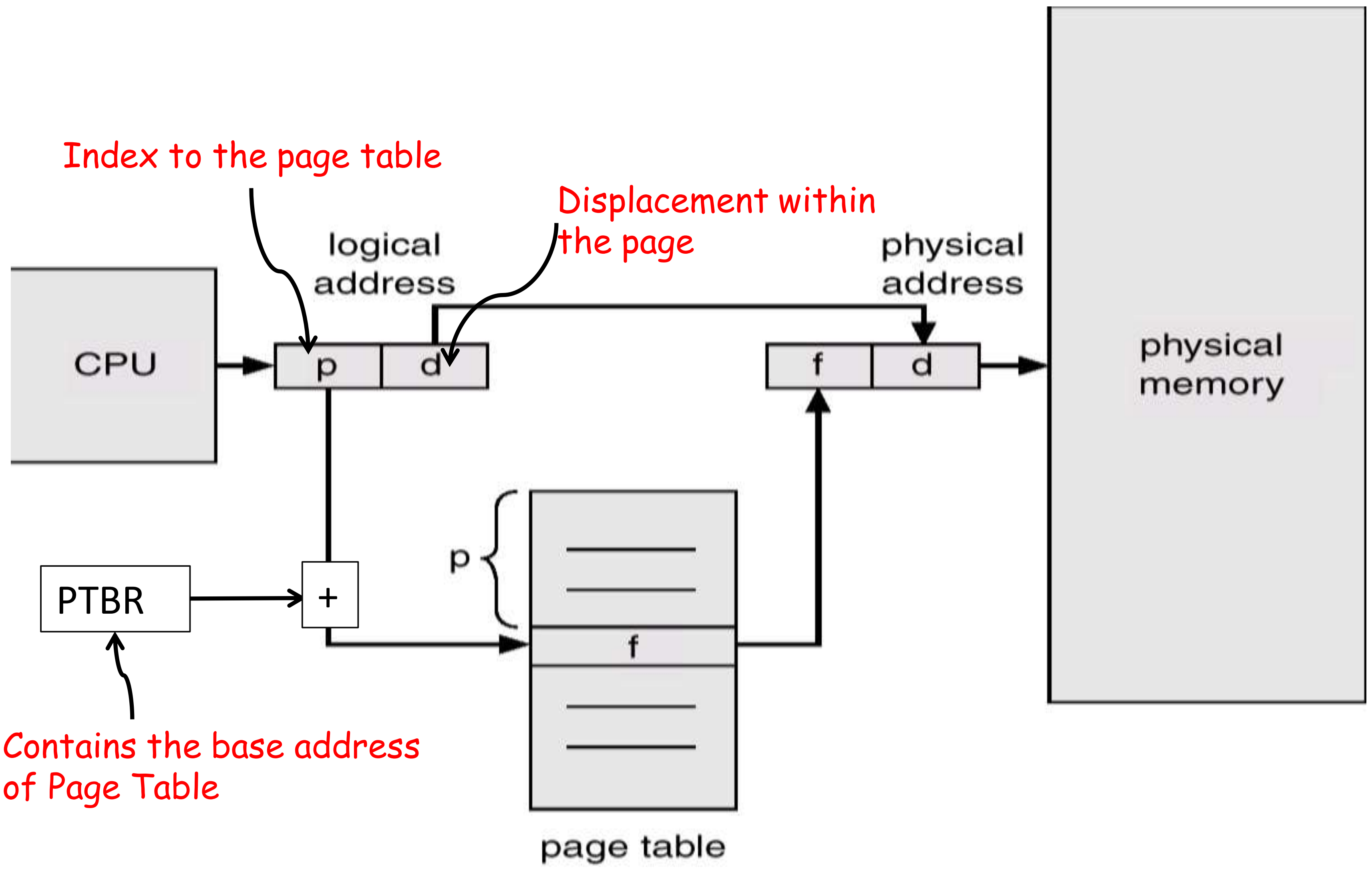
- Address generated by CPU is divided into:
  - **Page number (p)** – used as an index into a page table which contains base address of each page in physical memory.
  - **Page offset (d)** – combined with base address of frame to define the physical memory address that is sent to the memory unit.

Physical Address format (f, d)

- Physical address translated from the L.A is divided into:
  - **Frame number (f)** – kept in the page table against the page number.
  - **Frame offset (d)** – is the same as the page offset

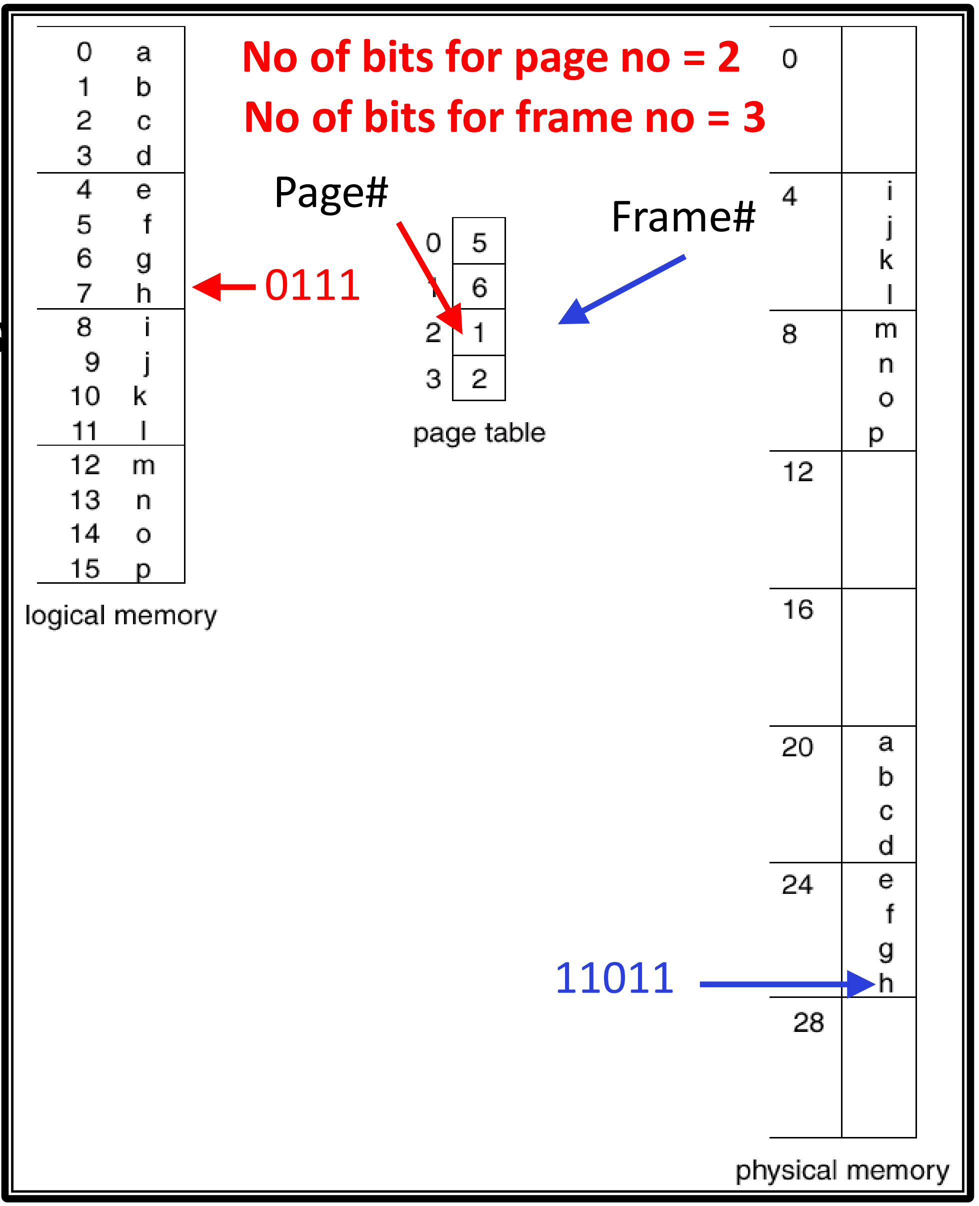
ADDRESS TRANSLATION ARCHITECTURE IN PAGING

- Address translation means conversion of logical address into the equivalent physical address.



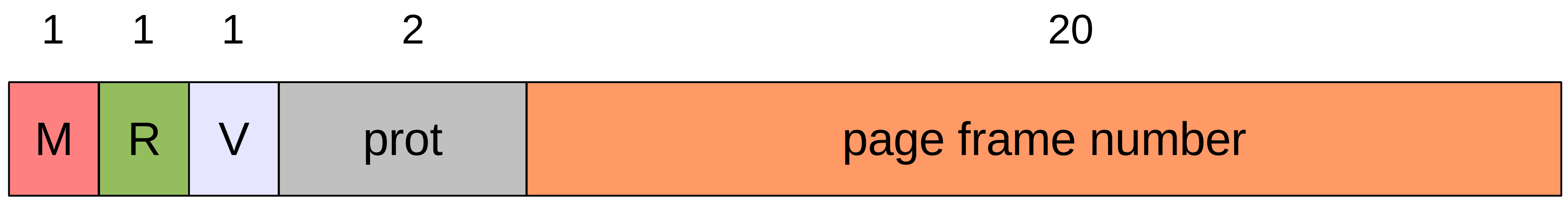
PAGING EXAMPLE

- Page size = 4 bytes
- Process address space = 4pages
- Physical address space = 8frames
- Logical address: $\overset{(p,d)}{(1,3)}$ = 0111
- Physical address: $\overset{(f,d)}{(6,3)}$ = 11011



PAGE TABLE ENTRY

Typical PTE format (dependent on processor architecture)



- Various bits accessed by MMU on each page access:
  - **Modify/Dirty bit:** Indicates whether a page is dirty or modified
  - **Resident bit:** Indicates whether the page is resident in memory or not (may be on disk). Its not an error for a program to access a non-resident page. A page fault occurs and the page is brought into memory from disk
  - **Valid bit:** Page is legal for the program to access, e.g., is the page a process has requested is part of its address space. (Abort)
  - **Protection bits:** Specify if a page is readable, writable or executable

SAMPLE PROBLEMS

Problem 12

- Consider a logical address space of 16 pages each of 1024 words (each word of 2 Bytes) mapped into a physical memory of 32 frames
- Give the Logical and Physical address format
- Also give the total Logical and Physical address space
- Compute the required page table size for this situation

Problem 13

- A system has 48 bit L.A & a main memory of 64 GBs. Page size is 4096 bytes. Compute the number of pages and frames that exist in the system. Also give L.A & P.A format.

Problem 14

- Consider a system with
  - L.A = 32 bits ; Page Size = 4 K ; Main memory = 512 MB.
- Compute the total process address space and maximum number of pages in a process address space. Also give the logical and physical address format. Also give the page table size for this situation.

Problem 15

- Consider a LA space of 8 pages of 1024 words mapped into memory of 32 frames.
  - How many bits are there in the LA?
  - How many bits are there in PA?

SAMPLE PROBLEMS

Problem 16

- In a system with a logical address space of 64 pages, each of 512 bytes mapped into physical memory of 1024 frames. Compute lengths (in bits) of p, d, f, logical and physical address format.

Problem 17

- A system has 48 bit logical address, physical address space is 32 bits and page size is 4 KB. Determine the lengths of p, d, f, logical and physical address formats, maximum number of pages per process and maximum number of frames in the system, page table entry size (PTES) and the size of the page table.

Problem 18

- Consider a system that allows maximum 2 Mega pages per process with 2 KB page size. Determine the length of the logical address only.

Problem 19

- Consider a system with 24 bits physical address space that supports a frame size of 512 Bytes. Calculate the page table entry size (PTES) and the length of physical address.

SAMPLE PROBLEMS

Problem 20

- For each of the following logical addresses (given in decimal), compute the page number and offset within the page; if the page size is 4 KB
  - 20000
  - 32768
  - 60000
- Repeat for an 8 KB page

Problem 21

- A machine has a 32 bit address space and an 8 KB page. The page table is entirely in hardware, with one 32 bit word per entry. When a process starts, the page table is copied to the hardware from memory, at one word every 100 nsec. If each process runs for 100 msec (including the time to load the page table), what fraction of the CPU time is devoted to loading the page tables?

Problem 22

- A machine has a 48 bit virtual addresses and 32 bit physical addresses. Page size is 8 KB. How many entries are needed for the page table?

SAMPLE PROBLEMS

Problem 23

- Let 14000 is a logical address, in which page does it exist if the page size is 1 KB?

Problem 24

- In a system with 34 bits logical address and 32 bits physical address and a 16 KB page size. How many entries will be there in the page table?

Problem 25

- Consider a virtual address 40808. Compute the virtual page number and offset for a 4 KB page

PAGED ARCHITECTURES

Paging Parameters in Intel P4

- 32 bit linear address. (Intel used the term linear instead of logical)
- 4 K page size
- Maximum pages in a process address space = $2^{32} / 2^{12} = 1$ M
- No of bits for **d** = 12
- No of bits for **p** = 32 – 12 = 20
- **What about the physical address format? / What about no of bits for f?**

Paging Parameters in PDP 11

- 16 bit Logical address.
- 8 K page size.
- Maximum pages in a process address space = $2^{16} / 2^{13} = 8$
- No of bits for **d** = 13
- No of bits for **p** = 16 – 13 = 3
- **What about the physical address format? / What about no of bits for f ?**

IMPLEMENTATION OF PAGE TABLE

IMPLEMENTATION OF PAGE TABLES

IN CPU REGISTERS

- Design CPU in such a way that page table can be kept / maintained within the CPU, using its registers. (costly affair)
- Feasible for small process address space with less number of pages which may be of large size
- Effective Memory Access Time (time to convert L.A to P.A) is almost the same as the Physical memory access time
- Example is PDP-11, which has eight pages each of size 8 KB

IN MAIN MEMORY

- Page table is kept in main memory
- Page-table base register (PTBR) points to the starting address of page table
- Page-table length register (PRLR) indicates size of the page table
- In this scheme every data/instruction access requires two memory accesses. One for the page table (that resides inside the main memory) and one for the data / instruction.

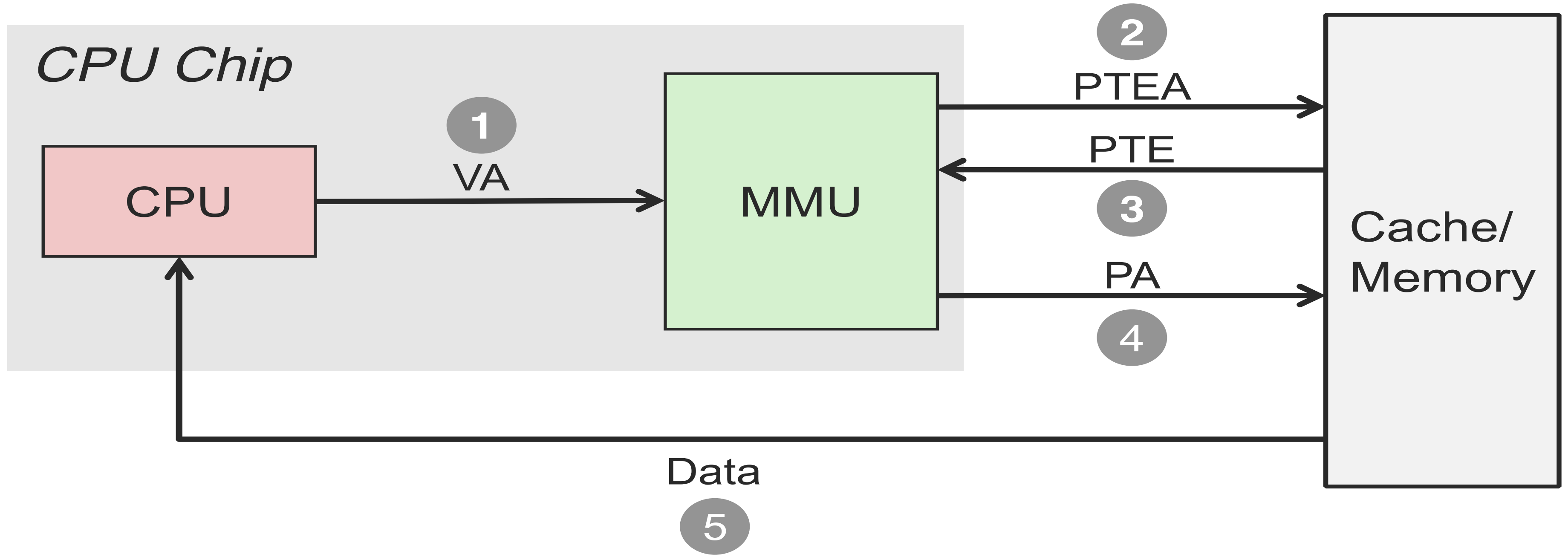
$$T_{EFFECTIVE} = 2\,T_{MEM}$$

IMPLEMENTATION OF PAGE TABLES (cont…)

IN ASSOCIATIVE CACHE / TLB

- Use a Cache / Translation Look Aside Buffer (TLB)
- Place references of some of the recently used pages in TLB, i.e. a portion of page table resides in TLB and the rest in main memory
- On a context switch, the TLB is flushed and loaded with values for the scheduled processes. (Normally TLB contains the page numbers of the currently running process)
- If mapping is found page hit else page fault

ADDRESS TRANSLATION-PAGE HIT



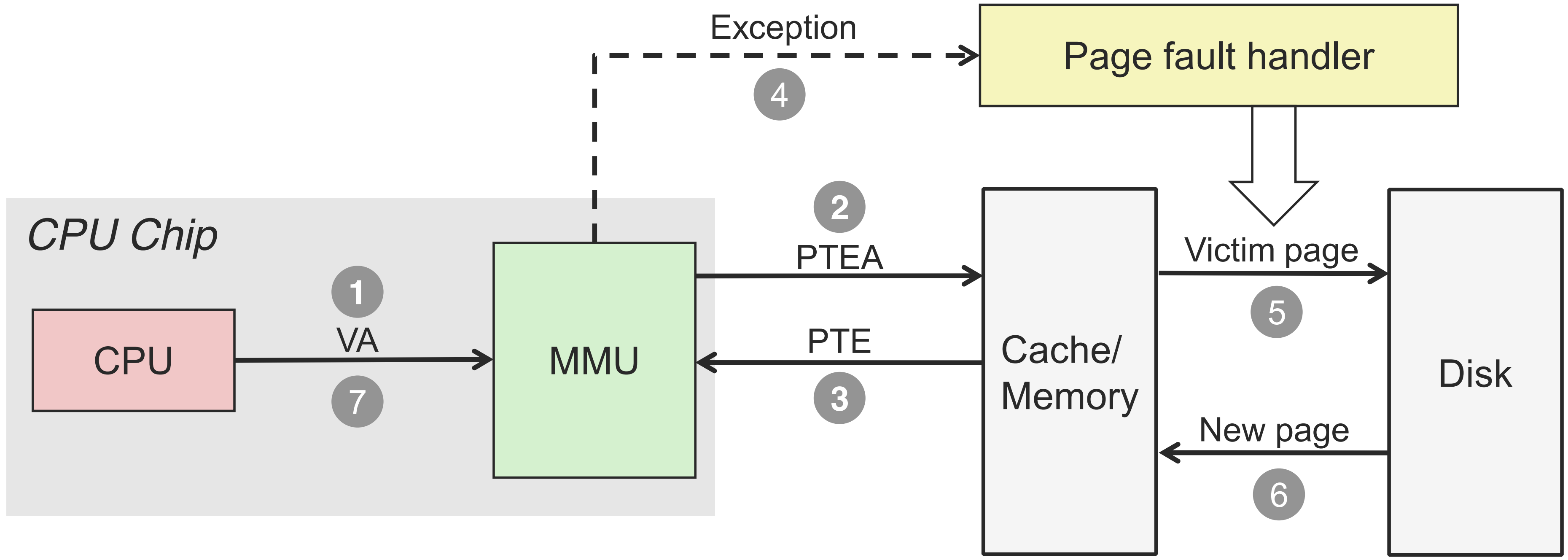
1. CPU sends virtual address/logical address to MMU

2-3. MMU fetches PTE from page table in memory

4. MMU sends physical address to memory

5. Memory sends data word to CPU

ADDRESS TRANSLATION-PAGE FAULT

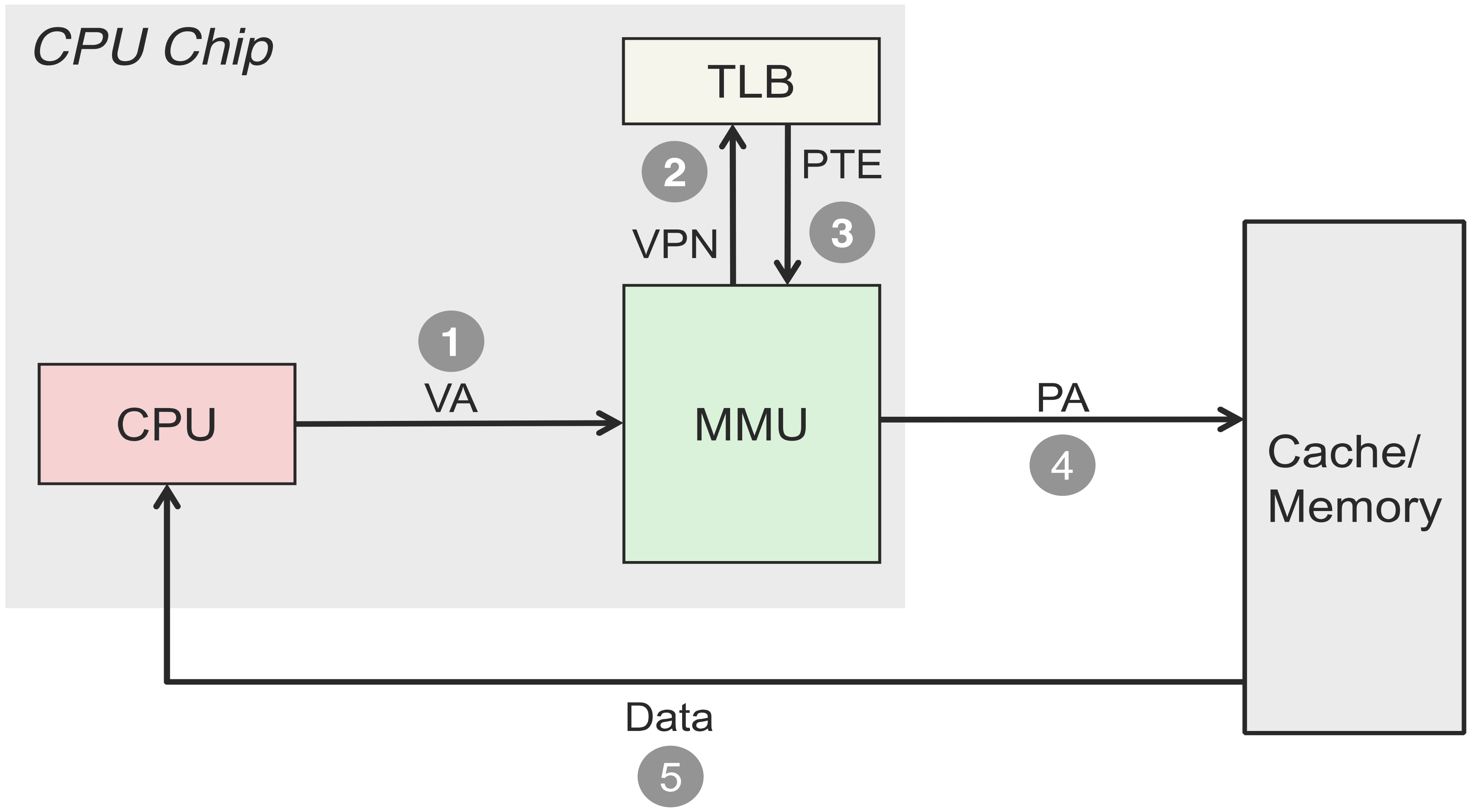


1. CPU sends virtual address/logical address to MMU
2-3. MMU fetches PTE from page table in memory
4. Valid bit is zero, so MMU triggers page fault exception
5. Handler identifies victim (and if dirty pages it out to disk)
6. Handler pages in new page and updates PTE in memory
7. Handler returns to original process, restarting fault instruciton

QUESTION 1: MEMORY ACCESSES

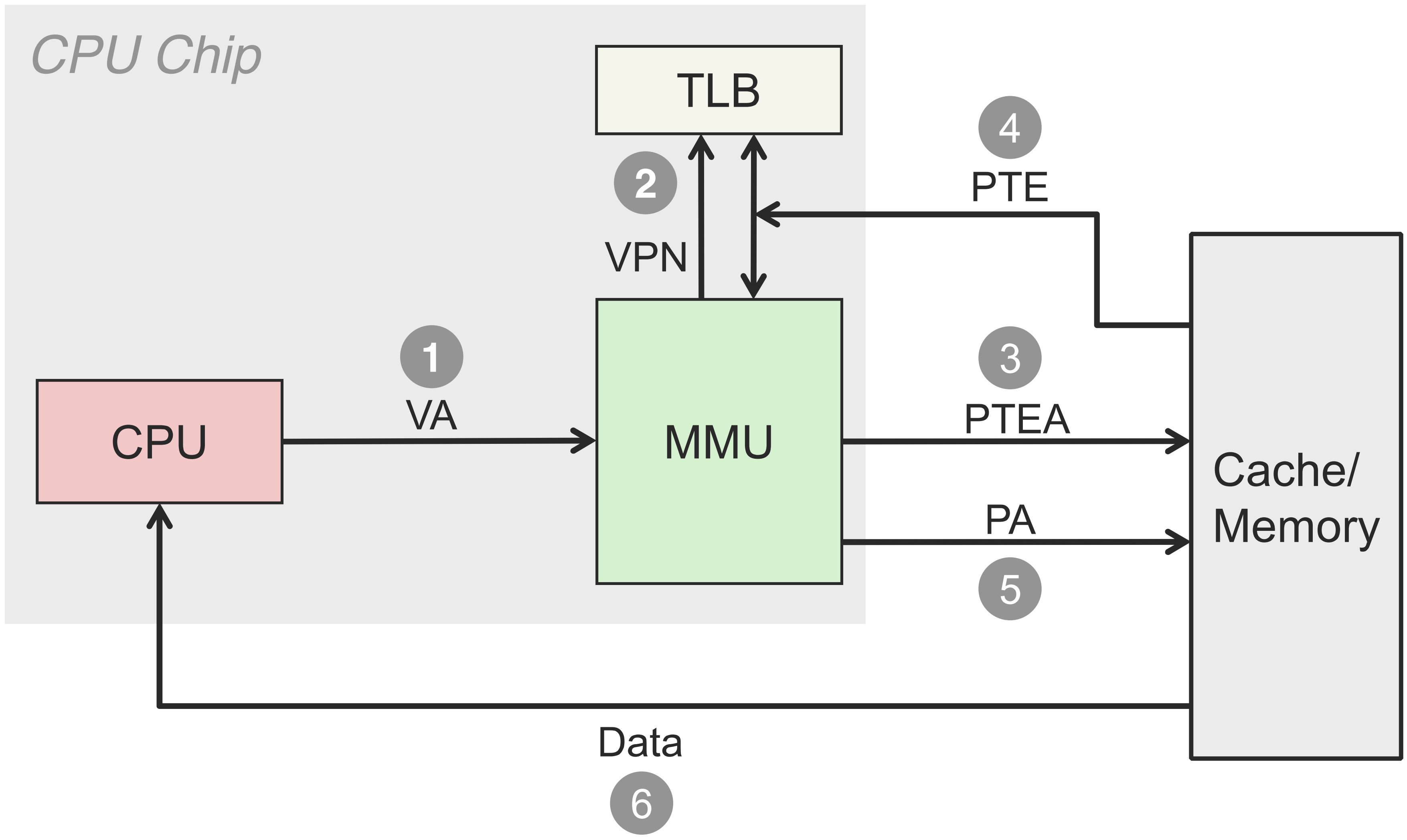
- Isn't it slow to have to go to memory twice every time?
- Yes, it would be..., so real MMUs don't
- **Solution 1:** PTEs are cached in L1 like any other memory word
  - PTEs may be evicted by any other data references
  - PTE hit still requires a small L1 delay
- **Solution 2:** TLB
  - Small, dedicated, superfast hardware cache of PTEs in MMU
  - Contains complete Page Table Entries for small number of pages

ADDRESS TRANSLATION-TLB HIT



A TLB hit eliminates a memory access

ADDRESS TRANSLATION-TLB MISS



A TLB miss incurs an additional memory access (the PTE).

Fortunately, TLB misses are rare

IMPLEMENTATION OF TLB

- Caches employing associative mapping stores complete address as well as content of memory word in cache.
- So any location in cache (using associative mapping) can store any word from main memory.
- **Example**.
  - Carry out associative mapping of main memory(32 X 8) and cache (8 words)
  - Width of Cache = Address bits of memory + Data width

Main memory

00000	**54**
00001	**90**
00010	**29**
00011	**25**
00100	**91**
	-
	-
	-
11111	**96**

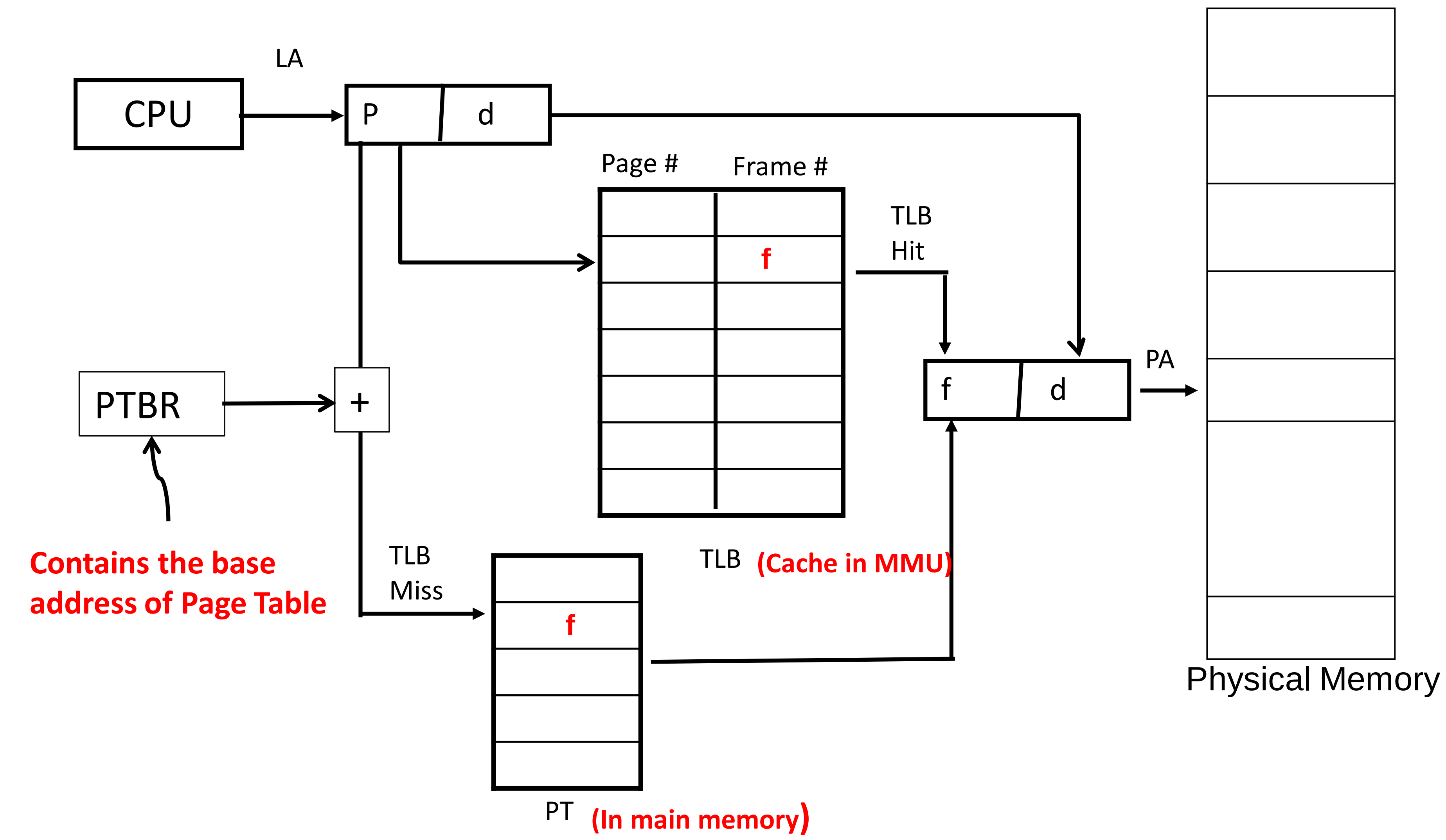
A.R (5) K.R (5)

Main memory Addr (5 bits)	Data (8bits)
00010	29
11111	96
00000	54

- CPU generates physical addr of 5 bits
- It is placed in A.R
- Associative memory is searched for the matching address
- If found the corresponding bit of the M.R is set & the corresponding data is sent to CPU
- If not found the memory is accessed & the addr-data pair is transferred to cache & to CPU
- If the cache is full an addr-data pair must be displaced to make room as per the predetermined replacement algo

IMPLEMENTATION OF PAGE TABLE (cont…)

PAGING HARDWARE



IMPLEMENTATION OF PAGE TABLE (cont…)

Performance In Paging

Effective Memory Access Time (Page hit) = Time to access TLB + Time to access memory

Effective memory Access Time (Page Fault) = Time to access TLB + 2 x Time to access memory

- If HR is the hit ratio and MR is the miss ratio then

$$T_{EFFECTIVE} = HR\,(T_{TLB} + T_{MEM}) + MR\,(T_{TLB} + 2\,T_{MEM})$$

IMPLEMENTATION OF PAGE TABLE (cont…)

Problem 26

- Consider a system with memory access time of 100 nsec. Page table is implemented using associative memory. The TLB access time is 20 ns. Hit ratio is 80%. Calculate the Effective memory access time. Calculate the Effective memory access time if there is no TLB, i.e. the entire page table is kept in memory.

Problem 27

- Repeat above example with a hit ratio of 95% and compare

Problem 28

Consider a paging system with the page table stored in memory.

a. If a memory reference takes 200 nanoseconds, how long does a paged memory reference take?

b. If we add associative registers, and 75 percent of all page-table references are found in the associative registers, what is the effective memory reference time? (Assume that finding a page-table entry in the associative registers takes zero time, if the entry is there.)

IMPLEMENTATION OF PAGE TABLE (cont…)

Problem 29

- If the hit ratio to a TLB is 80%, and it takes 15 nanoseconds to search the TLB and 150 nanoseconds to access the main memory, then what must be the effective memory access time in nanoseconds?

Problem 30

- If the hit ratio to register is 30% and hit ratio to TLB is 50%, and it takes 1 and 10 nanoseconds to search the register and the TLB respectively and 150 nanoseconds to access the main memory, then what must be the Effective Memory Access Time in nanoseconds?

Problem 31

- Consider a system with 80% hit ratio, 50 nsec time to search the associative registers, 750 nsec time to access main memory. Find the time to access a page:
  a. When the page number is found in associative memory
  b. When the page number is not found in associative memory
  c. Find the effective memory access time

STRUCTURE OF PAGE TABLE

QUESTION 2: PAGE TABLE SIZE

- **Isn't the page table huge? How can it be stored in RAM?**
- Yes, it would be…, so real real page tables aren't simple arrays
- For example, consider a logical address space of 64 bits and page size of 4 KB. Also consider the 8 byte PTE. How big the page table need to be?

STRUCTURE OF PAGE TABLE

- L.A space increases day by day due to the large size of processes, thus increasing the size of the Page Table
- Thus there is a dire need to structure Page Table in a better way especially in situation where the Page Table becomes larger in size than a single page size, i.e. a page table cannot be contained by a single page
- We can use following techniques for the structure of our page tables:

  a. Multi level / Hierarchical Page Table

  b. Hashed Page Table

  c. Inverted Page Table

Multi level / Hierarchical Page Tables

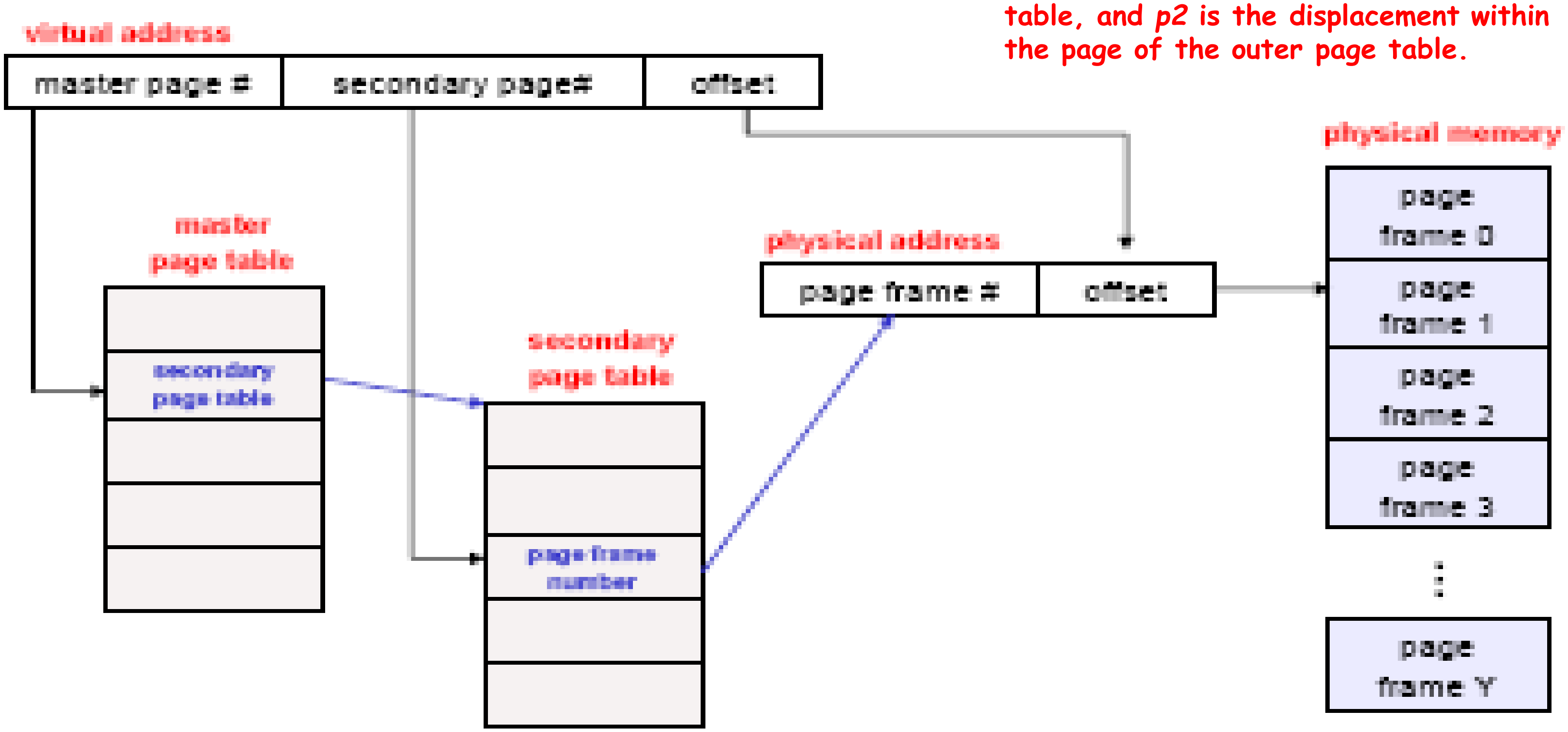
- Consider a system with a logical address of 32 bits, and page size of 4K and each page entry of 4 Bytes
- Maximum pages in a process address space = 1M
- Page Table size = 4 M (because each entry in page table is of 4 Bytes). Since total no of pages are 2^{20} i.e. 1MB, so there will be 2^{20} rows/ tupples /entries, each tupple of 4 Bytes. So page table size will be 4MB
- Since the system has a page size of 4K, therefore, the page table of 4M can't be accommodated in a single page of 4K. Thus we have to **make pages of the page table**
- We keep two page tables:
  - **Outer page table / page directory** (which keep track of the pages of the inner page table)
  - **Inner Page Table** which actually maps the frames
- No of pages in the outer page table = 4M / 4K = 1K
- So size of outer page table = 1K entries of 4 bytes each = 4KB
- This outer page table will now fit in one page

Multi level / Hierarchical Page Tables (cont…)

- 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
  - a 10-bit page offset
- Thus, a logical address is as follows:

page number		page offset
p_1	p_2	d
10	10	12

where *p1* is an index into the outer page table, and *p2* is the displacement within the page of the outer page table.

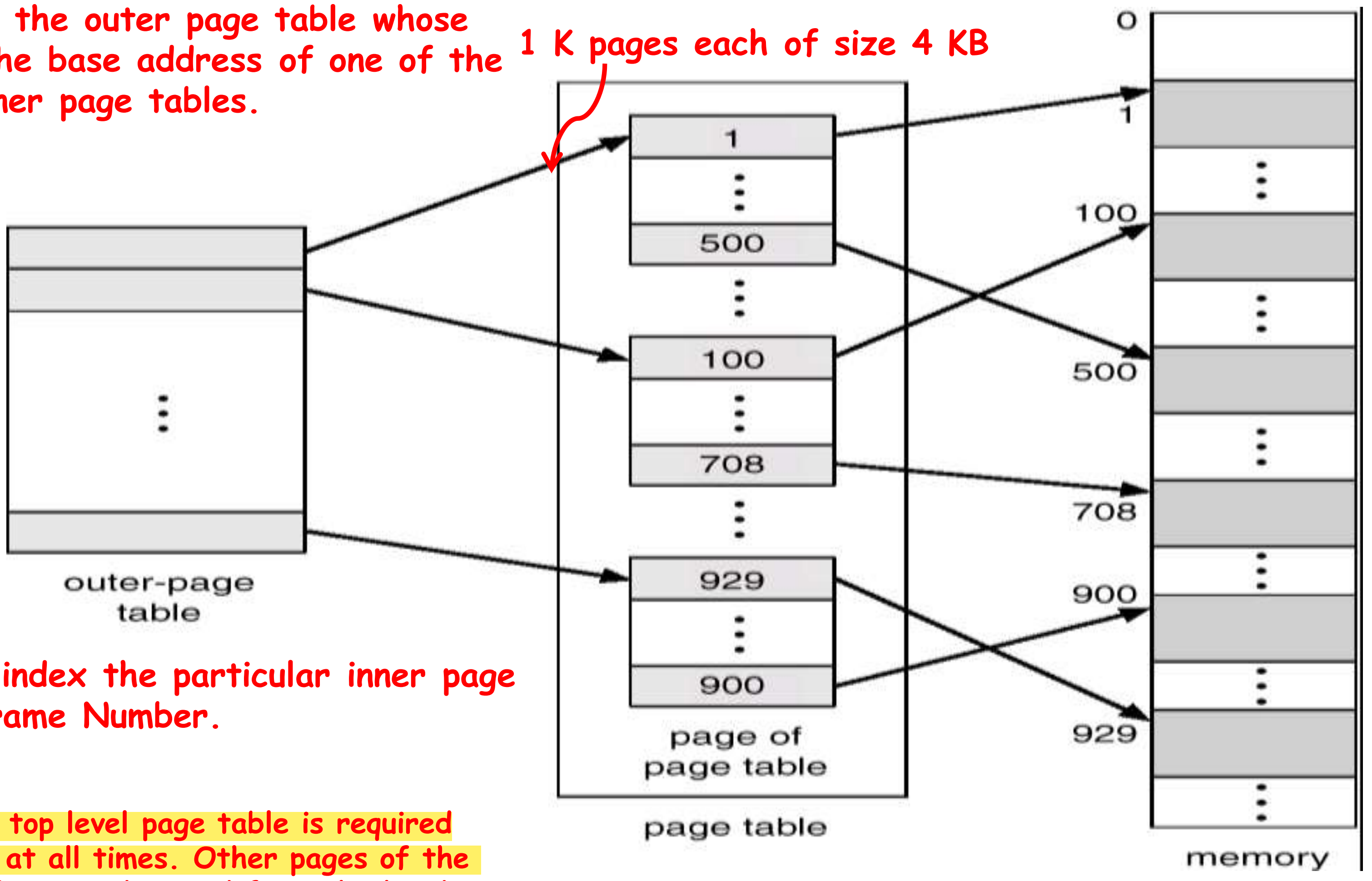


Two-Level Page Table Scheme

Outer page table is used to get hold of various pages inside the inner page table.

P1 is used to index the outer page table whose entry will give us the base address of one of the pages inside the inner page tables.

1 K pages each of size 4 KB



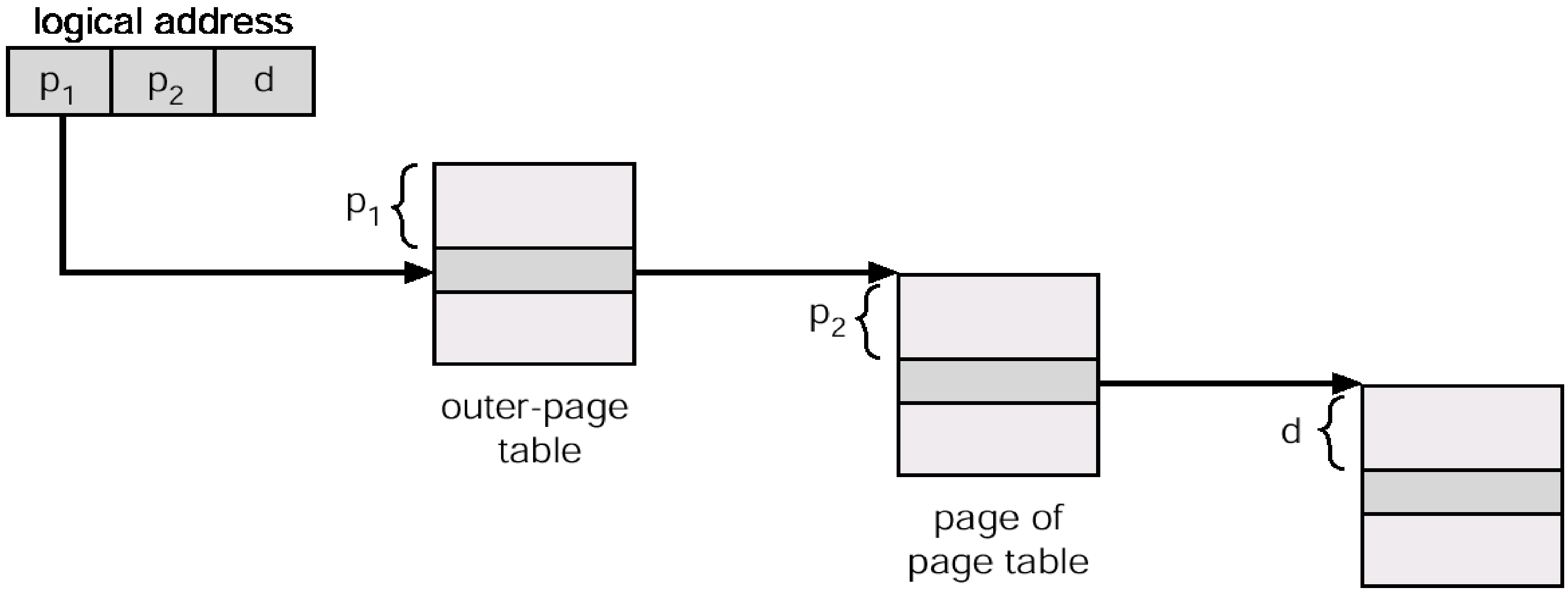
P2 is then used to index the particular inner page table to get the Frame Number.

Note:

Only the outer most / top level page table is required To be kept in memory at all times. Other pages of the inner page tables can be copied to and from the hard disk as needed.

Two-Level Page Table Scheme (cont…)

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



Since address translation works from the outer page table inwards, this scheme is also known as forward mapped page table. Pentium II uses this architecture

Multi level / Hierarchical Page Tables (cont…)

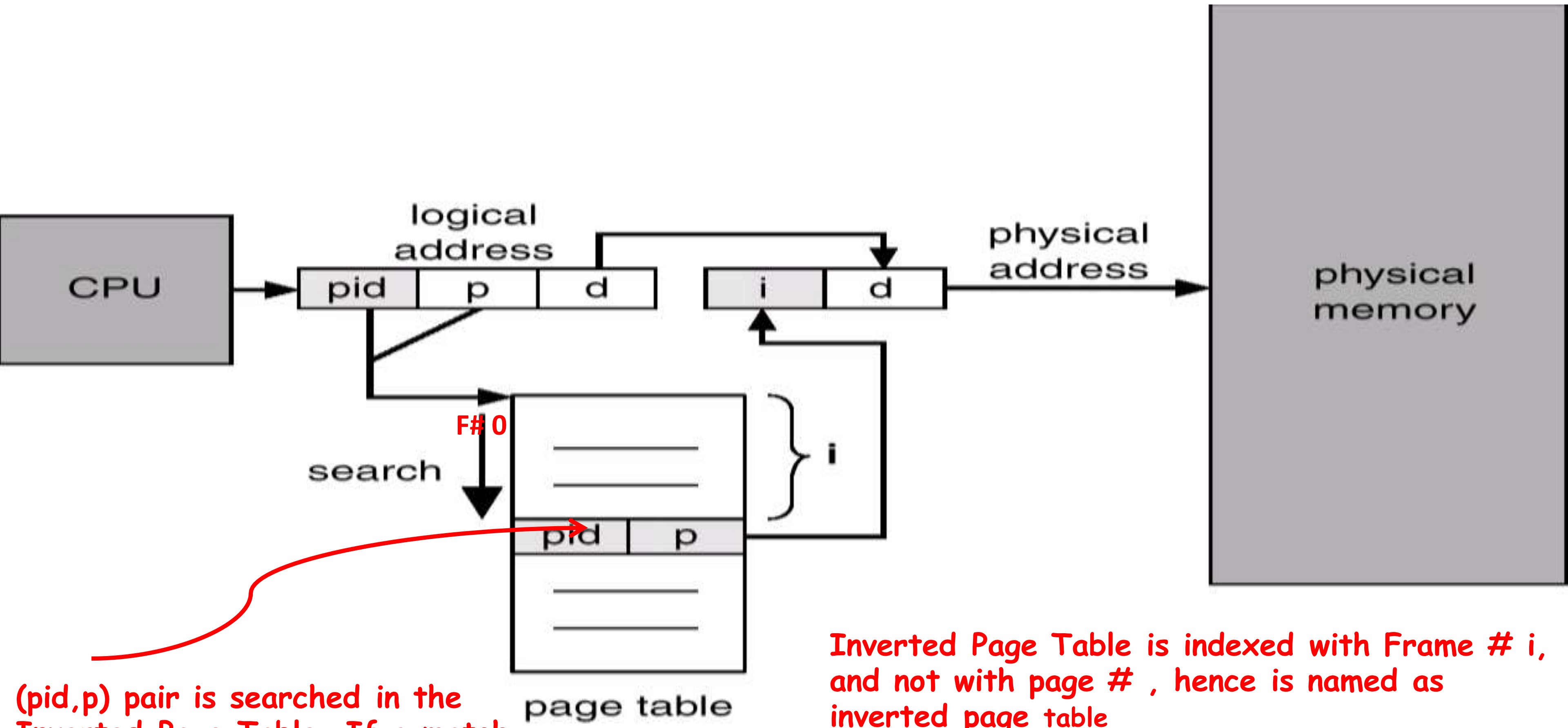
- Some other examples are:
  - 32 bit Sun SPARC support 3 Level Paging
  - 32 bit Motorola 68030 support 4 Level Paging
  - 64 bit Sun Ultra SPARC support 7 Level Paging
- Since each level is stored as a separate table in memory, converting a logical address to a physical one may take more than three memory accesses
- As the no of levels increases, too many memory references needed for address translation
- But at times this is required to support large process address space, so that larger processes can be executed in larger applications

Inverted Page Tables

- A Page Table has one entry for each Page in the Logical Address space of process
- An Inverted Page Table has one entry for each Frame in the Physical Address space (Physical Memory)
- Entries of a Page Table contains Frame numbers
- Entries of a Inverted Page Table contains Page numbers and pid (information about the process that owns that page)
- Page Table is indexed with page number, p
- Inverted Page Table is indexed with Frame number, f
- Inverted Page Tables are used to reduce the size of page table
- Only one page table is required in the system. (All processes will have / share a single Inverted Page Table)
- Page table size is limited by the number of frames (i.e. the physical / main memory) and not Process Address Space
- Each entry in the Page Table contains pid and p#. There is a possibility that there are two processes executing at a time and each process has a page no 3. So to avoid this confusion we have to keep pid as well
- 64 bit Ultra SPARC and IBM Power PC uses this technique

Inverted Page Tables (cont…)

This scheme decreases the amount of memory needed to store each page table but increases the time needed to search the table. It is because inverted page table is sorted by Physical address, but looks up occur on Virtual address. So the whole table might need to be searched for a match.



(pid,p) pair is searched in the Inverted Page Table. If a match is found say at entry i, then that is the Frame # where that page is loaded in main memory.

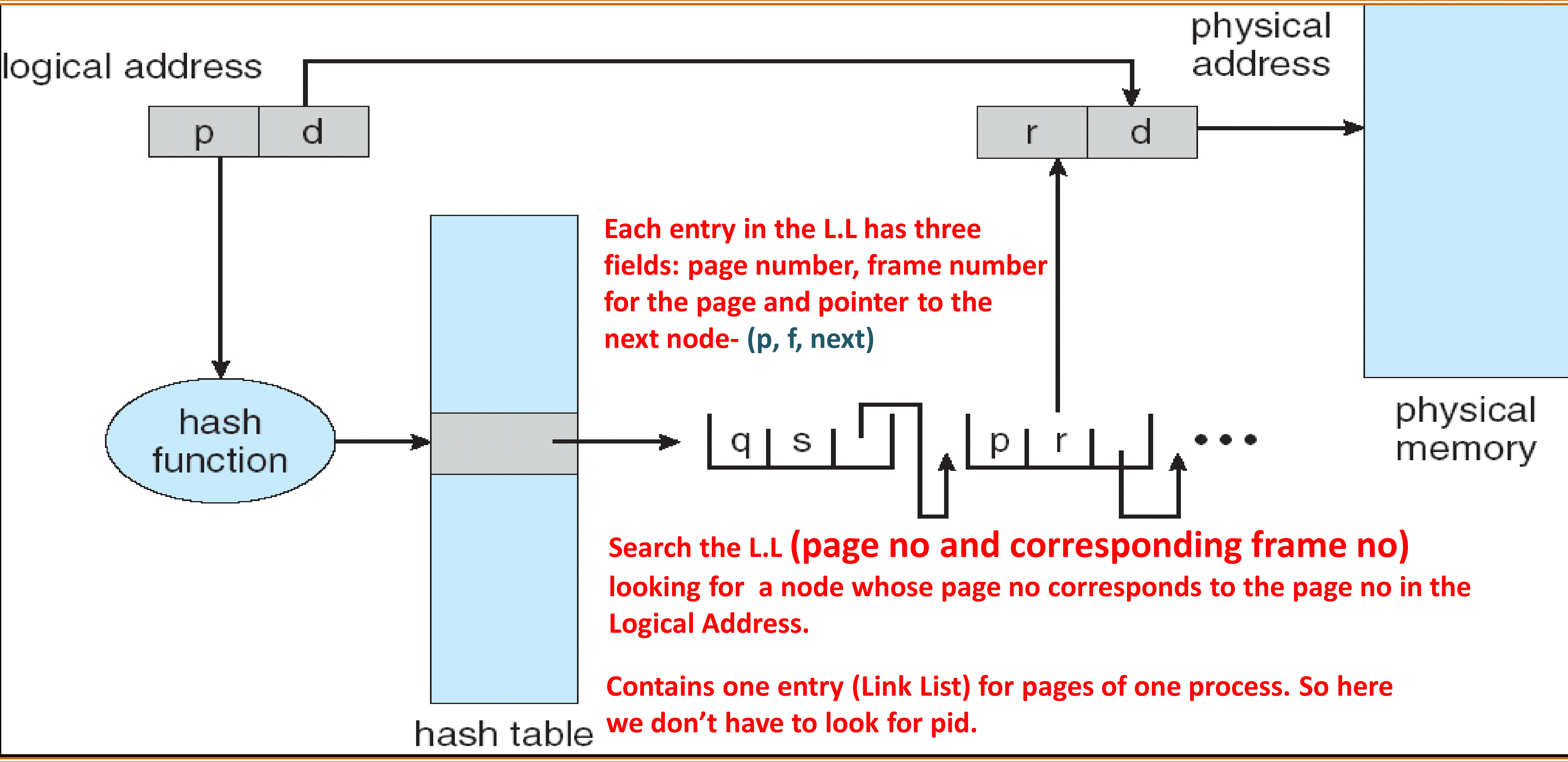
Inverted Page Table is indexed with Frame # i, and not with page # , hence is named as inverted page table

Inverted Page Table has entries equal to the number of frames in Main memory. Each entry contain the pid and its page number.

Hashed Page Tables

- Another common approach to handle address spaces larger than 32 bits is Hashed Page Table
- **Hashing.** A key is given, you pass it through a function that returns an integer value known as hash value, which is used to index a table / array (hash value must be within that array) where the required record will be lying or may contain a pointer to the record (link list)
- **Hash Function.** A function used to manipulate the key of an element in a list to identify its location in the list

Hashed Page Tables (cont…)



Each entry in the L.L has three fields: page number, frame number for the page and pointer to the next node- (p, f, next)

Search the L.L (page no and corresponding frame no) looking for a node whose page no corresponds to the page no in the Logical Address.

Contains one entry (Link List) for pages of one process. So here we don't have to look for pid.

- Hashing can be combined with Inverted Page Tables, to overcome its limitation of complex / time consuming search.
- For 64 bit address space **Clustered Page Tables** are used.

SAMPLE PROBLEMS

Problem 32

- A computer with a 32 bit address uses a two level page table. Virtual addresses are split into a 9 bit top level page table field, an 11 bit second level page table field and an offset. How large are the pages and how many are there in the address space?

Problem 33

- Suppose that a 32 bit virtual address is broken up into four fields: a, b, c and d. The first three are used for a three level page table system. The fourth field, d, is the offset. Does the number of pages depend on the sizes of all four fields? If not, which ones matter and which ones do not?

Problem 34

- A computer has 32 bit logical addresses and 4 KB pages. How many entries are needed in the page table if traditional (one level) paging is used?
- How many page table entries are needed for two level paging, with 10 bits in each part?

SAMPLE PROBLEMS

Problem 35

- Consider a system with 20 bit logical address and a page size of8 KB and a **single** level page table. Let the page table base register (PTBR) contains 14000 and PTES is 4 bytes. What will be the starting address of the last entry of the page table? Also determine the maximum number of pages per process supported by the system. What will be the format of the logical address? Can you calculate the number of frames with the provided information?

Problem 36

- Repeat above problem if logical address is of 32 bit and a **two level page table** is used.

Problem 37

- Consider a system with 36 bits logical address that supports 4 KB page size. Available physical memory is 64 MB. Calculate p, d, f, PTES, minimum number of levels of the page table per process, format of logical and physical addresses.

SAMPLE PROBLEMS

Problem 38

- Consider a system with 36 bits logical address space that supports 4 KB page size. Available physical memory is 64 GB. OS running on this system supports 32 bits PIDs. Calculate p, d, f, PTES in an inverted page table, size of the inverted page table, and the format of logical and physical addresses.

Problem 39

- In a system with 2048 MB RAM and a 4 KB page size, how many entries will be there in an inverted page table?

Problem 40

- In a 64 bit machine, with 256 MB RAM and a 4 KB page size, how many entries will there be in the page table if it is inverted?

Problem 41

- Consider a system that has a Process ID of 32 bits and uses inverted page table for address translation. The 34 bits logical address is viewed as a 22 bits page identifier (p) and 12 bits offset (d). Size of physical address is 32 bits. Compute page size, PTES, number of pages, and size of inverted page table.

SHARED PAGES

Shared Code

- One copy of read-only (reentrant) code shared among processes e.g. text editors
- Shared code must appear in same location in the logical address space of all processes
- Multiple invocations of vi editor or *gcc;* when multiple users invoke vim the code part can be shared by multiple processes, while data can be private for each process

Private Code and Data

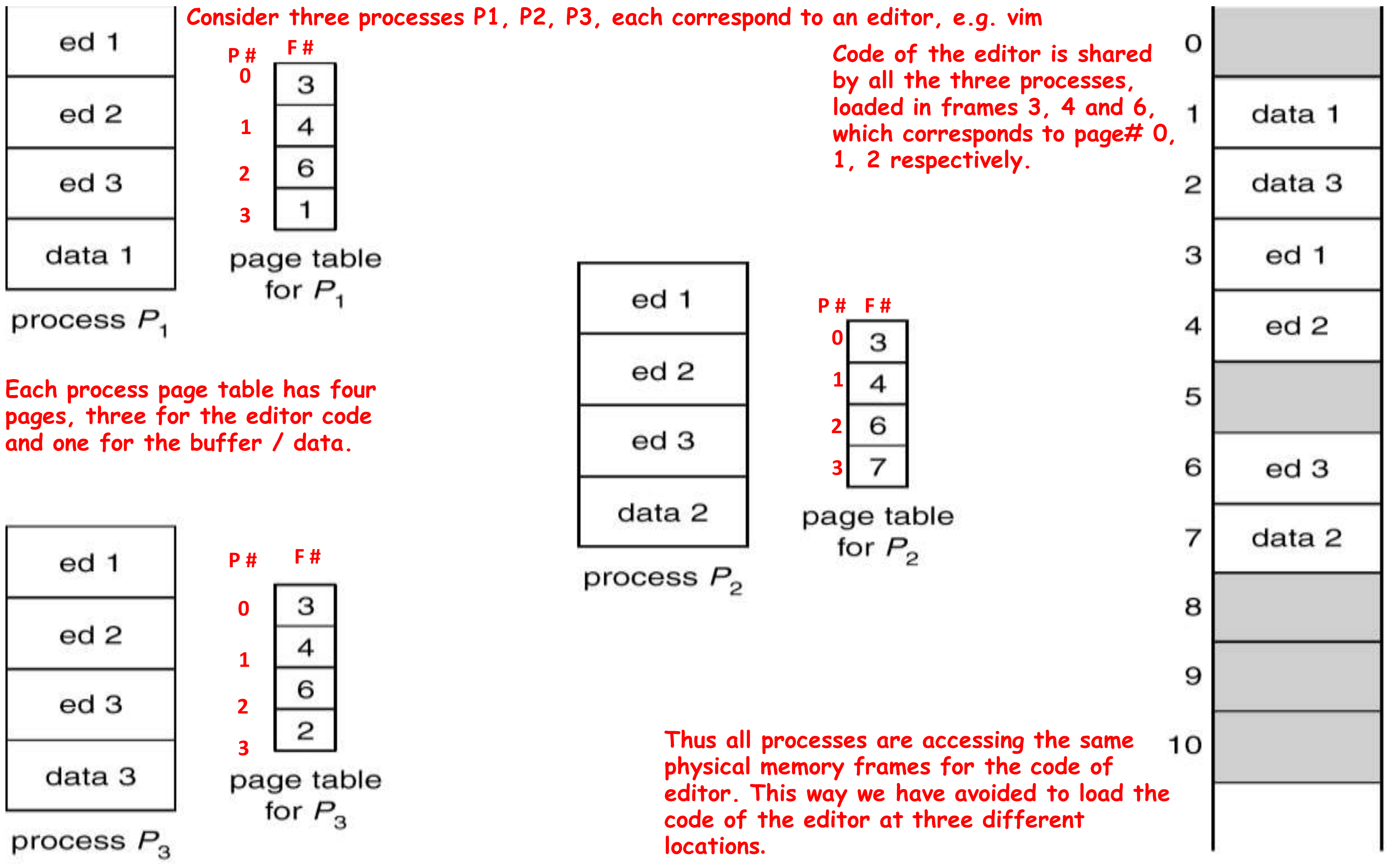
- Each process keeps a separate copy of the code and data
- The pages for the private code and data can appear anywhere in the logical address space

Shared Pages Example

Consider three processes P1, P2, P3, each correspond to an editor, e.g. vim

Code of the editor is shared by all the three processes, loaded in frames 3, 4 and 6, which corresponds to page# 0, 1, 2 respectively.

Each process page table has four pages, three for the editor code and one for the buffer / data.

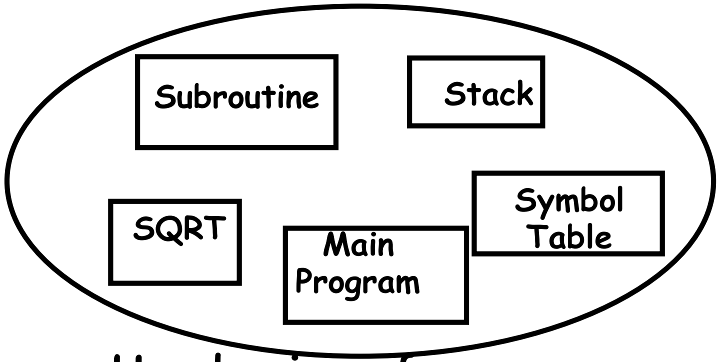


Thus all processes are accessing the same physical memory frames for the code of editor. This way we have avoided to load the code of the editor at three different locations.

INTRODUCTION TO SEGMENTATION

INTRODUCTION TO SEGMENTATION

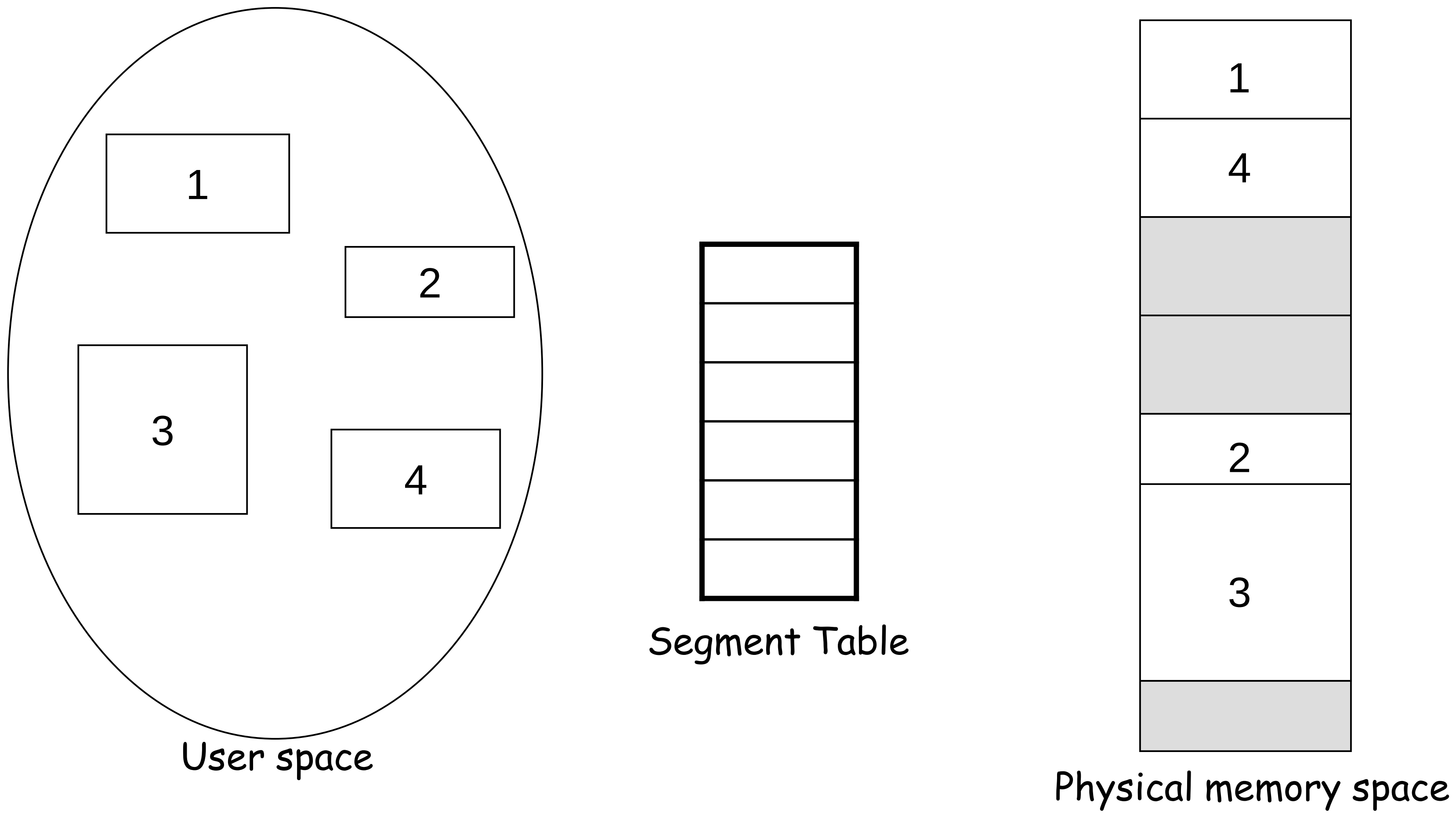
- A segment is a set of logically related instructions or data elements associated with a given name
- A segment is a logical unit such as: Main program, procedure, function, local & global variables, stack, symbol table, arrays
- A program is a collection of segments. Each program is divided into a number of segments which can be of different lengths, however, there is a maximum segment length
- A program is loaded by loading all of its segments into dynamic partitions that need not to be contiguous
- Memory-management scheme that supports programmer's view of memory
- **Private Code and Data**
  - Each process keeps a separate copy of the code and data
  - The pages for the private code and data can appear anywhere in the logical address space



User's view of a program

The user does not think of memory as a linear array of words. Rather the user prefers to view memory as a collection of variable sized segments, with no necessary ordering among segments.

LOGICAL VIEW OF SEGMENTATION



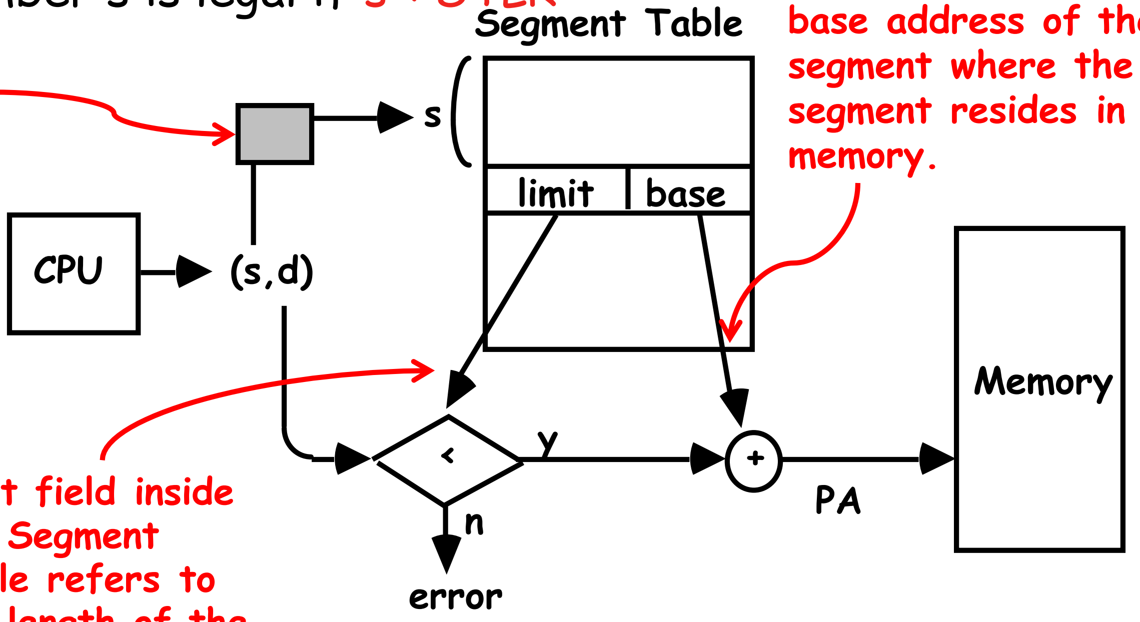
- In paging we store the pages in frames and keep the mapping in page table
- In segmentation instead of pages we have segments that are kept in the frames and we keep the mapping in segmentation table
- Unlike pages segments can be of variable size

SEGMENTATION ARCHITECTURE

- Logical address consists of a two tuple:
- <segment-number, offset>,
- 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

Ensure that s < STLR. If yes then adds STBR to it and then index the Segment table.

Base contains the base address of the segment where the segment resides in memory.

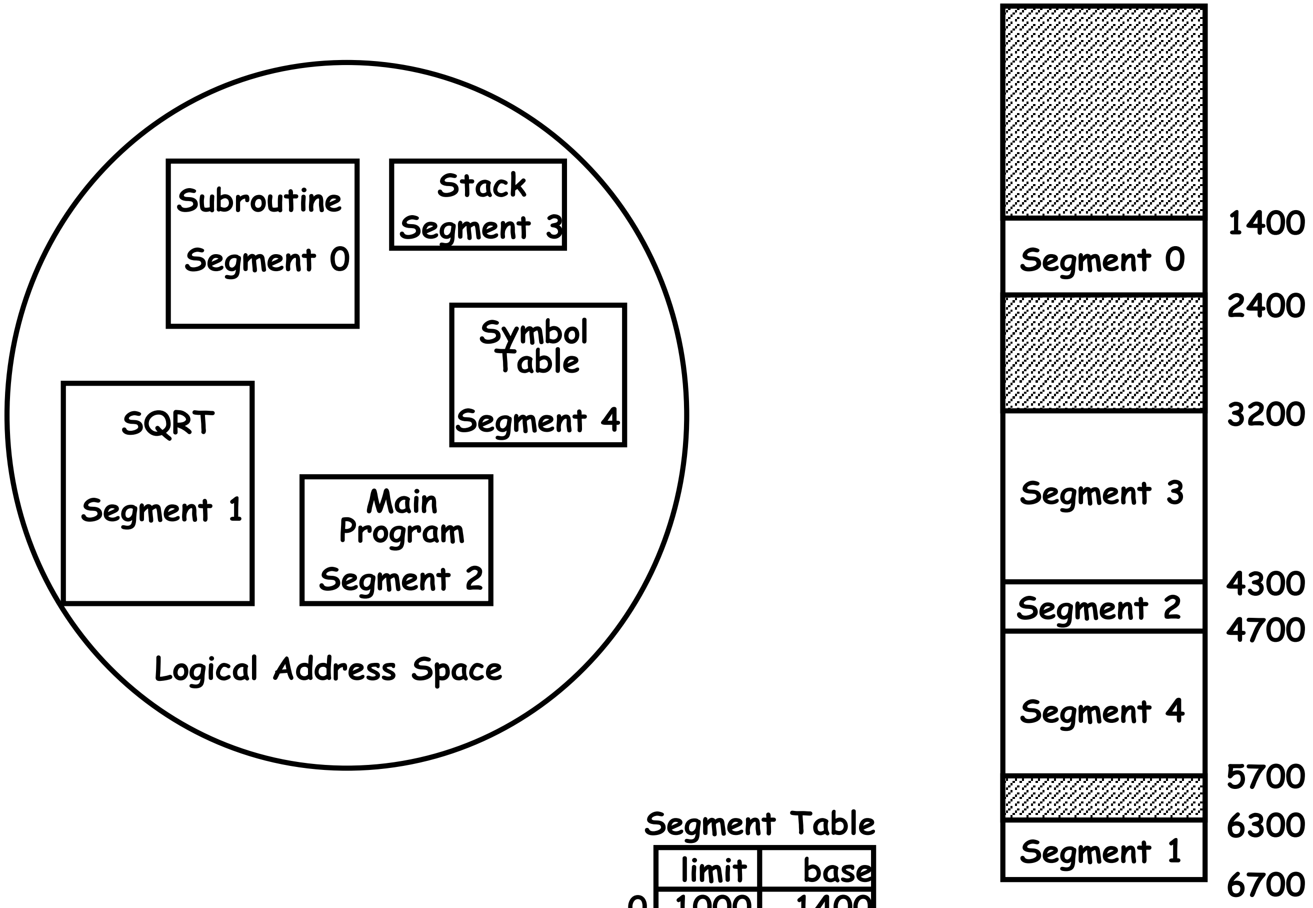


Limit field inside the Segment Table refers to the length of the segment inside the main memory.

Each segment table entry has:
- **Base** contains the starting physical address where the segment resides in memory.

- **Limit** specifies the length of the segment.

SEGMENTATION EXAMPLE



Segment Table

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

SAMPLE PROBLEMS

Problem 42

Consider the given segment table, What are the Physical addresses for the following logical addresses:

- (2,399)
- (4,0)
- (4,1000)
- (3,1300)
- (6,297)

Segment Table

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

Problem 43

Consider the given segment table, What are the Physical addresses for the following logical addresses:

- 0,430
- 1,10
- 2,500
- 3,400
- 4,112

Seg	Base	Len
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

SAMPLE PROBLEMS

Problem 44

Consider the given segment table, What are the Physical addresses for the following logical addresses:

- (0, 0)
- (2, 120)
- (6, 10)
- (5, 3399)
- (4, 1200)
- (0, 99)

Segment #	Length	Base
0	100	12000
1	1200	12100
2	190	13300
3	444	15500
4	19308	18008
5	3400	5000

Problem 45

Consider the above segment table, if segment table base register (STBR) contains 36500 and segment table entry size (STES) is 64 bits then what will be size of segment table? Also compute the address of the last entry?

SAMPLE PROBLEMS

Problem 46

- Compare between Segmentation and Base/Bound Address Translation Techniques

1. Segmentation supports generalized base and bound with support of multiple segments at once
2. Protection: Different segments can have different protections
3. Flexibility: Can separately grow both stack and heap. Enables sharing of code and other segments if needed

Problem 47

- What is the main difference between paging of memory and segmentation of memory? List two ways in which this difference affects the address translation hardware required with each scheme.

EXTERNAL FRAGMENTATION IN SEGMENATION

- External fragmentation exist in segmentation
- Holes can be adjusted by compaction, i.e. shuffle segments to place free memory together in one block
- Compaction is possible only if relocation is dynamic and is done at execution time
- So in segmentation we need run time address binding, i.e. dynamically a segment of process can be shifted to another location within the process address space within the main memory

PROTECTION IN SEGMENATION

- With each entry in segment table we associate specific bits for protection like we do in page table, i.e. valid bit, read-write, read only and execute bits.
- Protection information can be included in the segment table or segment register of the memory management hardware.
- Format of a typical segment descriptor is

Base address	Length	Protection

The protection field in a segment descriptor specifies the *Access Rights* to the particular segment

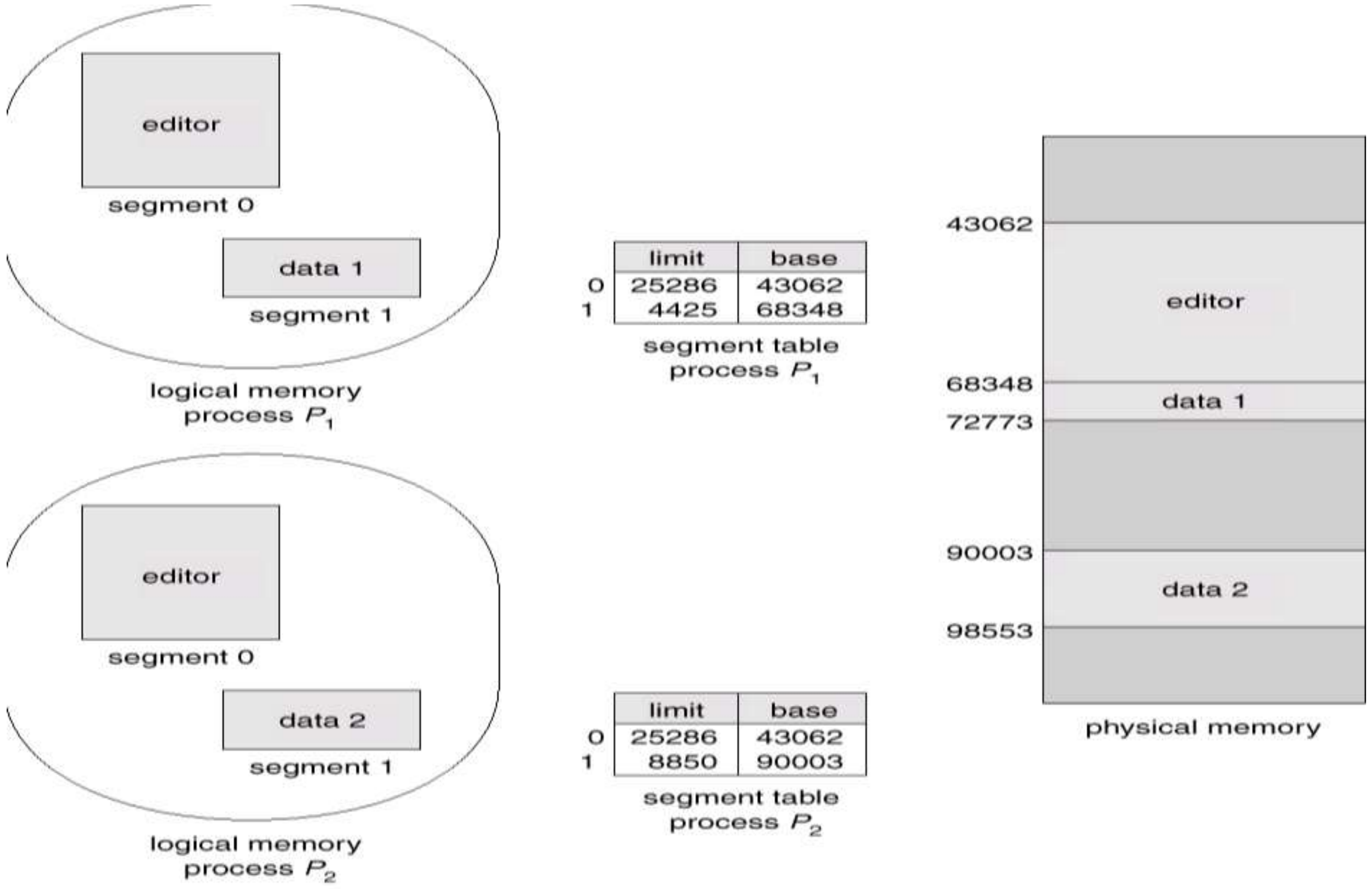
- Access Rights:

Full read and write privileges
Read only (write protection)
Execute only (program protection)
System only (O.S. Protection)

SHARING OF SEGMENTS

- Sharing can be implemented at segment level
- Segment table of multiple processes point to the same segment
- Consider two processes P1 and P2 each correspond to an editor
- In the main memory both will load the editor code at a single location and will share its code
- P1 and P2 will both have their private data/buffers as shown on next slide

SHARING OF SEGMENTS



	limit	base
0	25286	43062
1	4425	68348

segment table process P_1

	limit	base
0	25286	43062
1	8850	90003

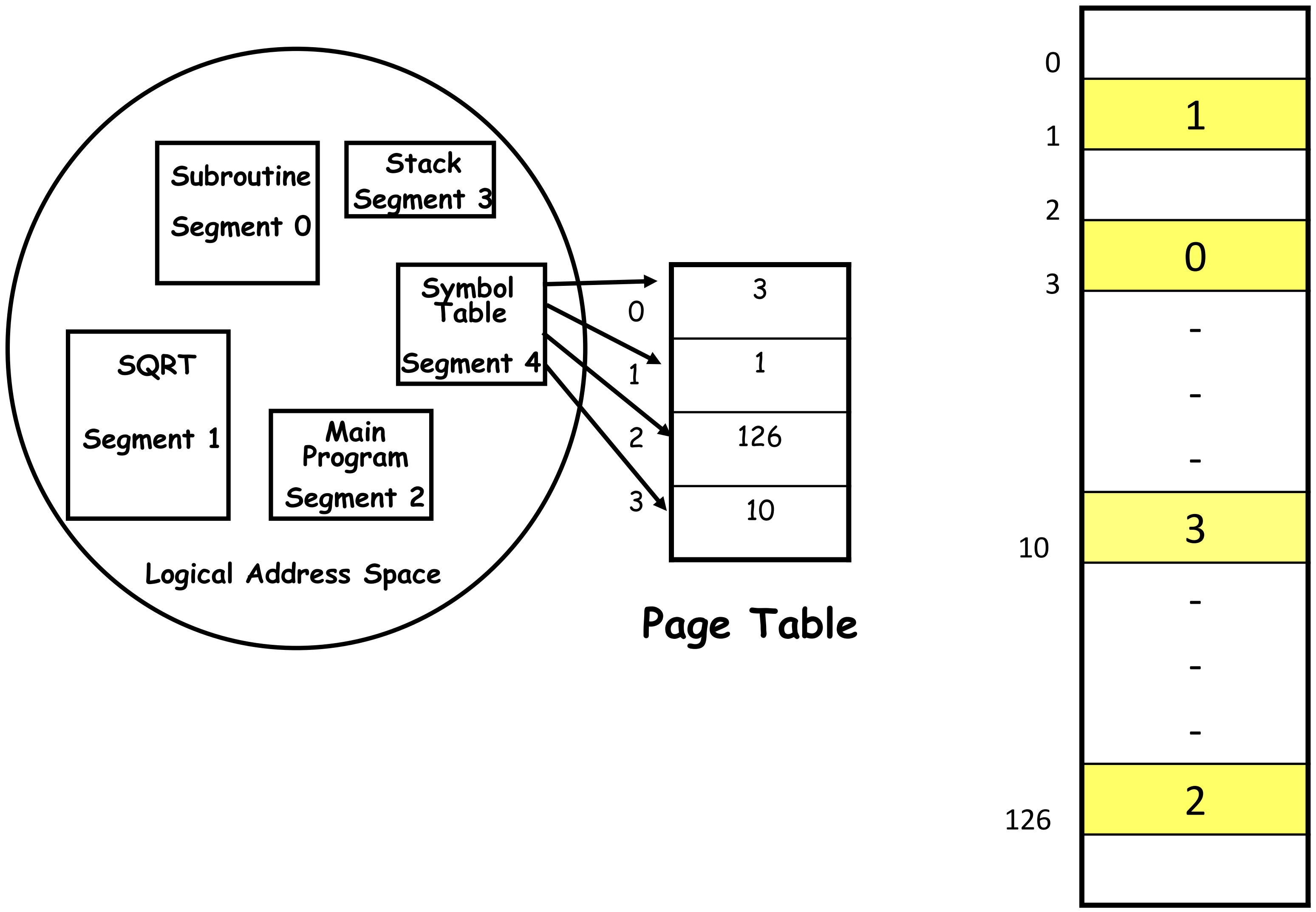
segment table process P_2

INTRODUCTION TO PAGED SEGMENTATION

PAGED SEGMENTATION

- When the segments size become too large the larger is the External Fragmentation
- To prevent it, we page the segments (like we paged the Page Table)
- In paging, against every process we had a Page Table which used to map the pages of that particular process with the frames in the Physical memory where those pages were loaded
- Here every segment will be further divided into pages and their mapping information is kept in the Page Table
- **So every segment will have a Page Table. This way there will be no External fragmentation**

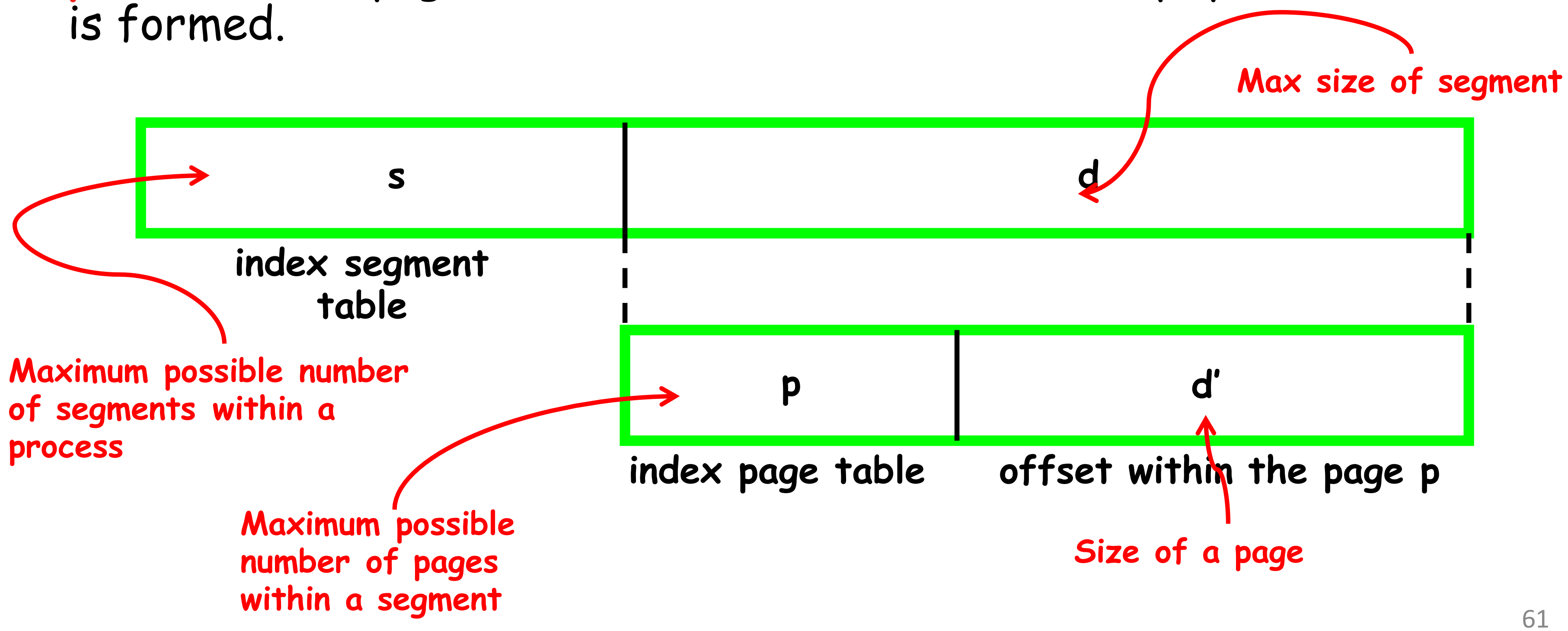
PAGED SEGMENTATION EXAMPLE



Here we have divided segment 4 into four pages and kept that information in page table.

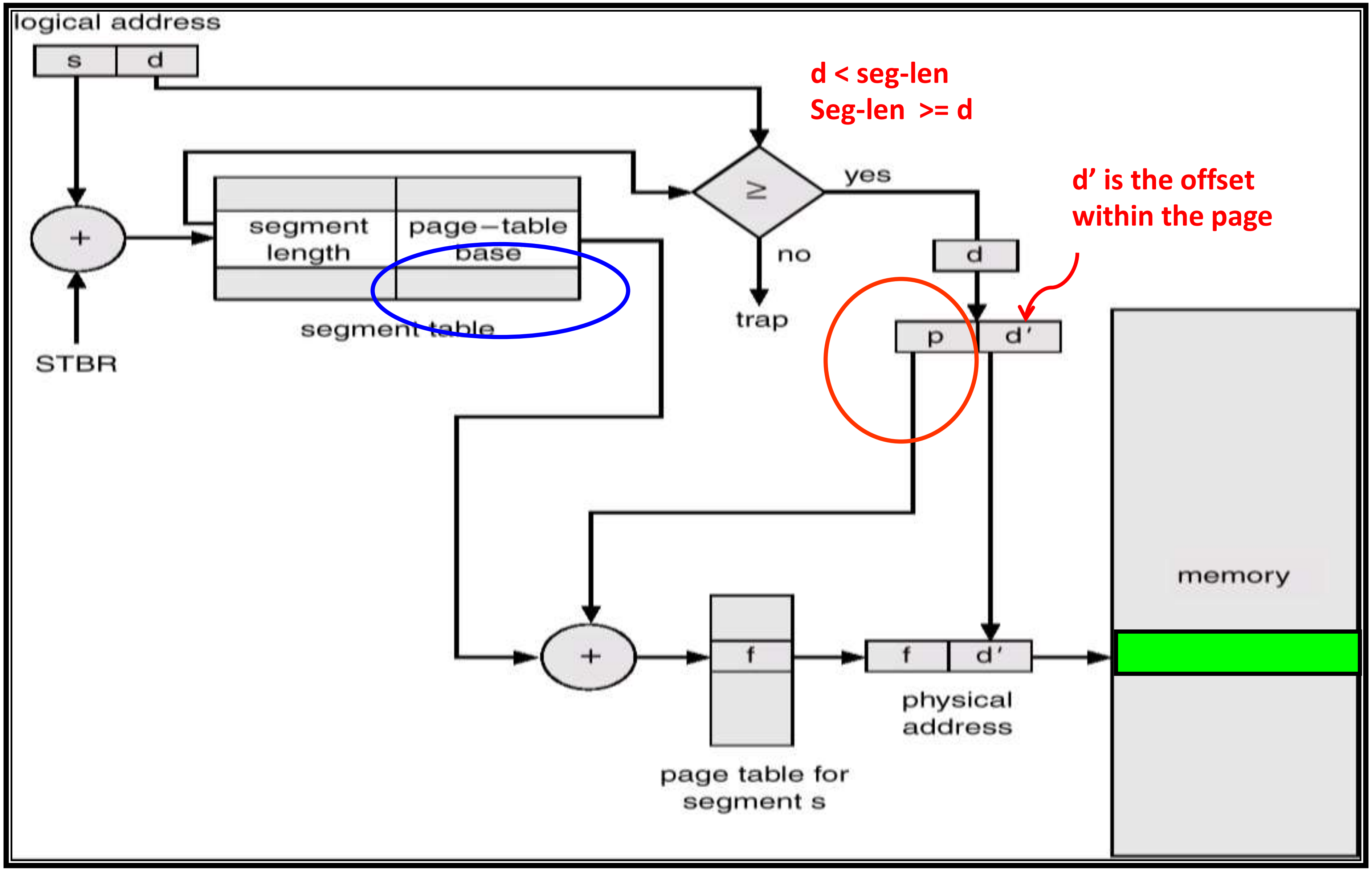
PAGED SEGMENTATION EXAMPLE

- Logical address is still **<s,d>**, with **s** used to index the segment table
- Each segment table entry consist of the tupple **<seg-length, page-table-base>**
- The logical address is legal if **d < seg-length**.
- Segment offset, d, is partitioned into two parts: **p** and **d'**
- **p** is used to index the page table associated with segment **s**
- **d'** is used as offset within a page.
- **p** indexes the page table to retrieve frame, **f**, and physical address **(f,d')** is formed.



Max size of segment
s
d
index segment table
Maximum possible number of segments within a process
p
d'
index page table
offset within the page p
Maximum possible number of pages within a segment
Size of a page

TRANSLATION (LA - PA) IN PAGED SEGMENTATION



SAMPLE PROBLEM

Problem 48

Consider MULTICS on a GE 345 processor, with Logical Address of 34 bits and page size of 1 KB. **s** is of 18 bits and **d** is of 16 bits.

- What is the largest segment size?
- What is the maximum number of segments per process?
- Give maximum number of pages per segment.
- Give the LA format including the no of bits for p and d’

SAMPLE PROBLEM

Problem 49

- Consider a process in MULTICS with its segment # 15 having 5096 bytes. The page size is 1 KB. The process generates a Logical Address of (15, 3921).
  - Is it a legal address? If yes why?
  - How many pages does the segment have?
  - What page does the logical address refer to, and what is its offset?
  - What is the P.A if page#3 (i.e. fourth page) is in frame 12?

SAMPLE PROBLEMS

Problem 50

Consider the given segment table. How many page tables will be constructed for the process whose segments are shown in the segment table?

Problem 51

Consider the given segment table. If the system implements paged-segmentation with the page size of 2 KB, then compute the page number and offset for the logical address of (4, 12765). Also compute the number of pages in segment 4 and the address of the 3rd entry in the page table (Assume PTES is 4 Bytes)

Segment #	Length	Base
0	100	12000
1	1200	12100
2	190	13300
3	444	15500
4	19308	18008
5	3400	5000

Problem 52

Consider the above segment table, if segment table base register (STBR) contains 36500 and segment table entry size (STES) is 64 bits then what will be size of segment table? Also compute the address of the last entry?

SAMPLE PROBLEM

Problem 53

Consider a logical address in paged segmentation (16, 7865), where 16 is segment number and 7865 is offset. Suppose segment 16 has length of 15690 bytes. Page size is 1 KB. Calculate following:

- Number of pages in segment number 16
- In which page the above said offset will reside?
- How would you represent the above address in <s, p, d'> format?
- Let the page p is stored at frame 30, what would be the physical address?

REAL MODE AND PROTECTED MODE ADDRESSING

- Earlier PCs like i8086 having an address bus of 20 bits and support only 1 MB of memory
- i80286 has an address bus of 24 bits and can support 16 MB of memory
- i80386 has an address bus of 32 bits and can support 4 GB of memory
- Higher PCs (PCs with address bus greater than 20 bits) can operate in two modes:
- **Real Mode**
  - Only first 1 MB of RAM can be accessed in Real Mode
  - The Real Mode Address is a 20 bit address, stored and represented in the form of **segment:offset** (as a 20 bit addr can't be stored in memory). Give a 4 bit left shift to seg register and add offset in it to get 20 bits address
- **Protected Mode**
  - In Protected Mode whole of the RAM is accessible that includes the Conventional RAM (640 KB), System RAM (384 KB) and Extended memory (Memory higher than 1 MB)
  - PCs initially boots up in Real Mode. It may be shifted to protected mode during the booting process using drivers like HIMEM.SYS.

PROTECTED MODE ADDRESSING IN WINDOWS

- Real Mode supports only contiguous allocation
- Protected mode support non-contiguous allocation
- i80286 and higher processors support non-contiguous allocation
- i80286 support segmentation in Protected Mode
- i80386 and higher processors also support paging
- We must give credit to Pentium designers, as they have designed it to support pure paging, pure segmentation and paged segments, while at the same time being compatible with the i80286
- The key to such non contiguous allocation systems is the Addressing Technique, that is supported only in Protected mode

LA TO PA TRANSLATION IN PROTECTED MODE

- The heart of Pentium Virtual Memory consists of two tables: the LDT (Local Descriptor Table) and the GDT (Global Descriptor Table). Each program has its own LDT, but there is a single GDT, shared by all the programs on the computer. The LDT describes segments local to each program, including its code, data, stack, where as the GDT describes system segments
- **Selectors**
  - In protected mode the segment registers are used as selectors
  - As the name suggest they are used to select descriptor entry from some descriptor table
  - To access a segment, a Pentium program first loads a 16 bit selector for that segment into one of the machine's six segment registers
  - During execution, the CS register holds the selector for the code segment and the DS register holds the selector for the data segment

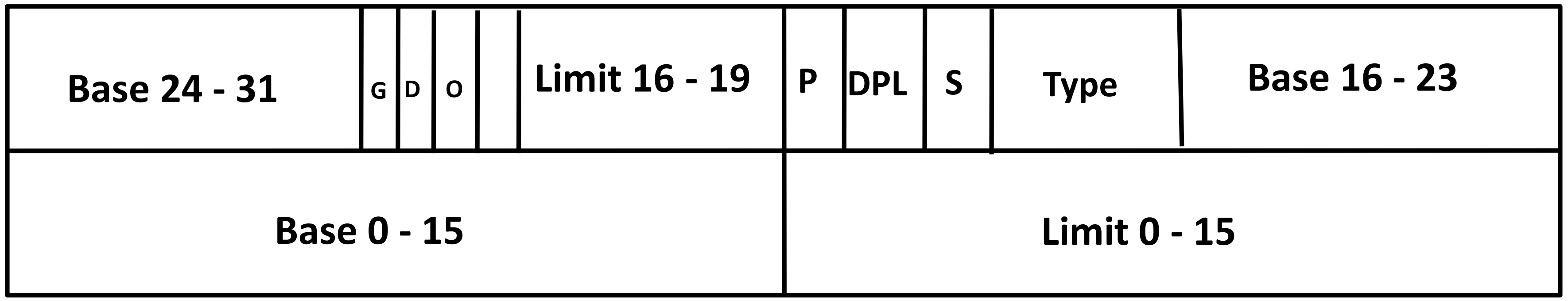
13	1	2
Segment #	g	p

i. 13 bits selector index selects a descriptor table entry out of 8K entries of LDT or GDT (each entry of 8 bytes, so a total of 64 KB)
ii. g = 0 means GDT, g = 1 means LDT.
iii. p is 2bit privilege level with 00 as highest privilege

LA TO PA TRANSLATION IN PROTECTED MODE

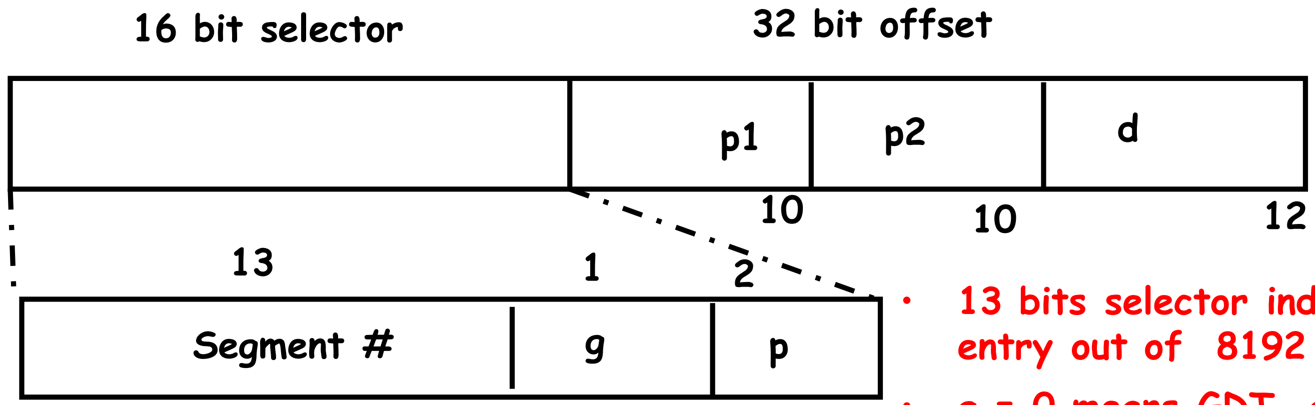
Descriptor

- A descriptor describes a memory segment by storing attributes related to a memory segment. For every segment there is a descriptor
- Significant attributes of a memory segment can be its base address, its length and its access rights
- For address of 80286 we need 24 bit, while for 80386 we need 32 bits. But in both cases the descriptor size is 8 Bytes



Intel 80386 ADDRESS TRANSLATION

- IBM OS/2, MS Windows, Linux are the most famous operating systems that run on i386
- It uses paged segmentation with two level paging and a TLB with 32 entries.
- L.A = 48 bits
- Page size = 4K. (each page table entry is of 4 Bytes)
- Each entry of TLB can refer to a page. So 32 entries of TLB can address to a memory of 32 X 4K = 128 K (which is also known as TLB reach)
- A bigger picture of V.A / L.A in Intel 80386 is given below:



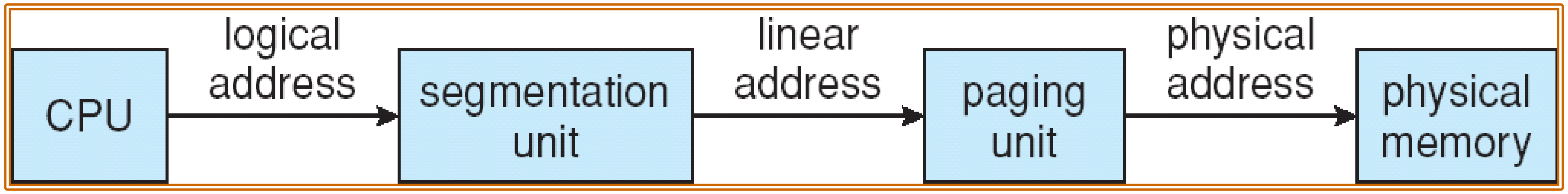
- 13 bits selector index selects a descriptor table entry out of 8192 entries of LDT or GDT
- g = 0 means GDT, g = 1 means LDT
- p is 2bit privilege level with 00 as highest privilege

Intel 80386 ADDRESS TRANSLATION (cont…)

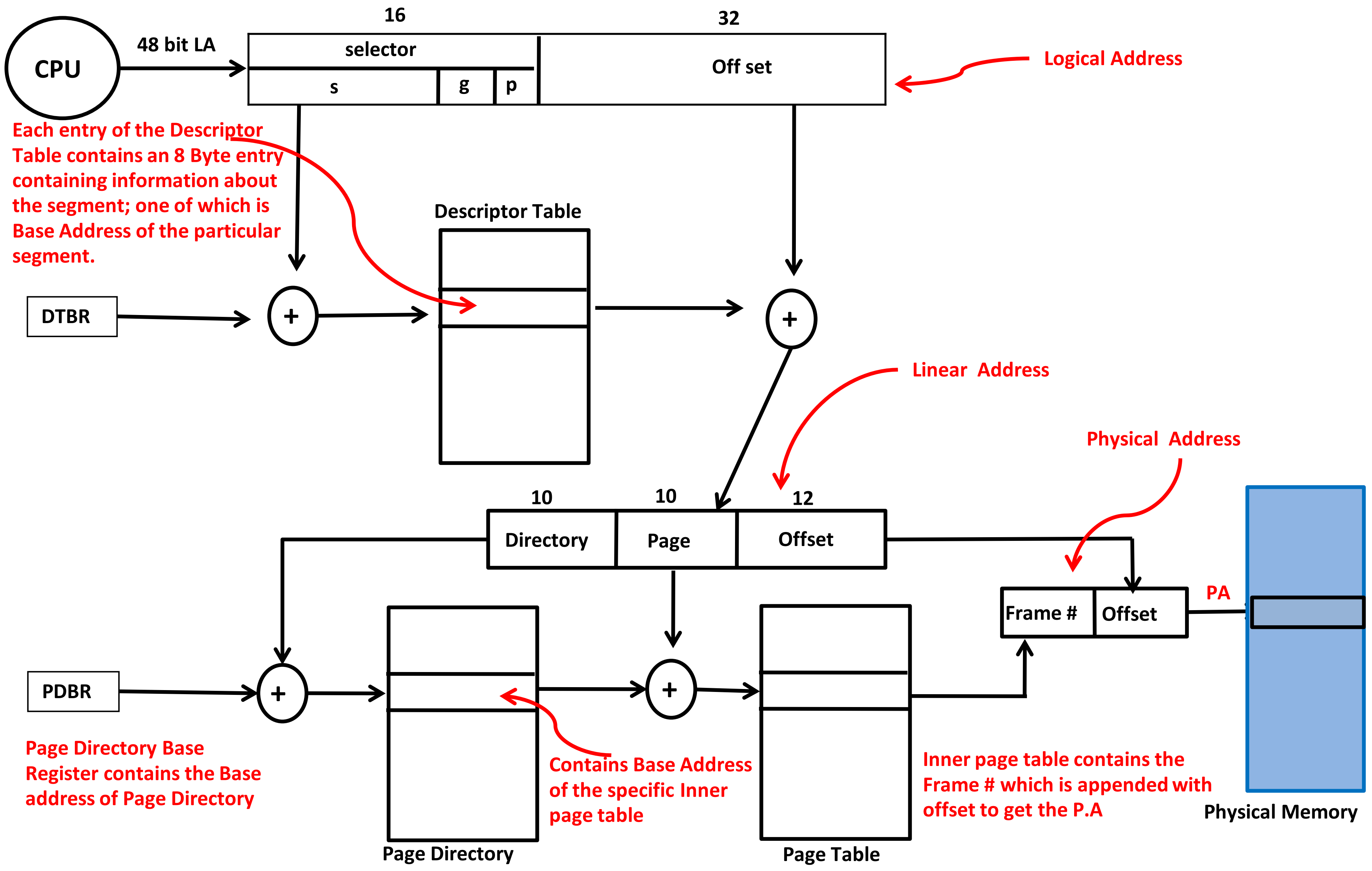
- We have 48 bit Logical address, which in Intel literature is also called Virtual Address
- Linear Address Space = 2^{32} Bytes (max size of a segment)
- Max Segment size = 4 GB
- Maximum number of segments per process = 16 K (13 bits selector and one bit for the selection of LDT or GDT)
- At any one time the Intel CPU can access six segments out of these because of six segment registers (CS, DS, SS, ES,   )
- Selector is used to index a segment descriptor table to obtain an 8 byte segment descriptor entry. Base address and offset are added to get a 32 bit linear address, which is partitioned into **p1**, **p2** and **d** for supporting two level paging.
  - **p1** is used to index the page directory / outer page table (having 1K entries), which gives us the appropriate page table base address
  - **p2** is used to index this selected inner page table (having 1K entries) from where we get the frame number of the page of the given L.A
  - This frame number is appended with **d** part of the Linear Address to get the Physical address

Intel 80386 ADDRESS TRANSLATION (cont…)

- **Supports both segmentation and segmentation with paging**
- **CPU generates 48 bits Logical Address**
  - Given to segmentation unit:
    - » Which produces 32 bits Linear Address
  - Linear address given to two level paging unit:
    - » Which generates Physical Address

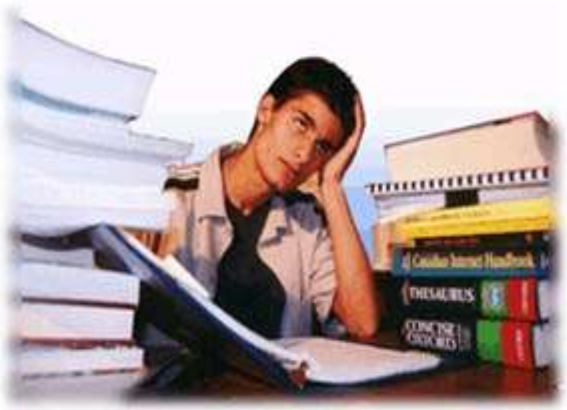


Intel 80386 ADDRESS TRANSLATION (cont…)



CPU
48 bit LA
16
32
selector
s
g
p
Off set
Logical Address
Each entry of the Descriptor Table contains an 8 Byte entry containing information about the segment; one of which is Base Address of the particular segment.
Descriptor Table
DTBR
+
+
Linear Address
Physical Address
10
10
12
Directory
Page
Offset
Frame #
Offset
PA
PDBR
+
+
Page Directory Base Register contains the Base address of Page Directory
Contains Base Address of the specific Inner page table
Inner page table contains the Frame # which is appended with offset to get the P.A
Physical Memory
Page Directory
Page Table

We're done for now, but Todo's for you after this lecture…



- Go through the slides and Book Sections: 8.4 - 8.8
- Solve all the sample problems given in slides to understand the concepts discussed in class