OPERATING SYSTEMS
Topic-Paging

- Non-contiguous Allocation
- Paging
  - Principles of Operation
  - Hardware Support

Non-Contiguous Memory Allocations

- In this type of allocation, the memory is allocated in such a way that parts of a single logical object may be placed in non-contiguous areas of physical memory.
- Address translation performed during execution instructions establishes the necessary correspondence between *a contiguous virtual space address* and *possible discontiguous memory locations*.

Paging

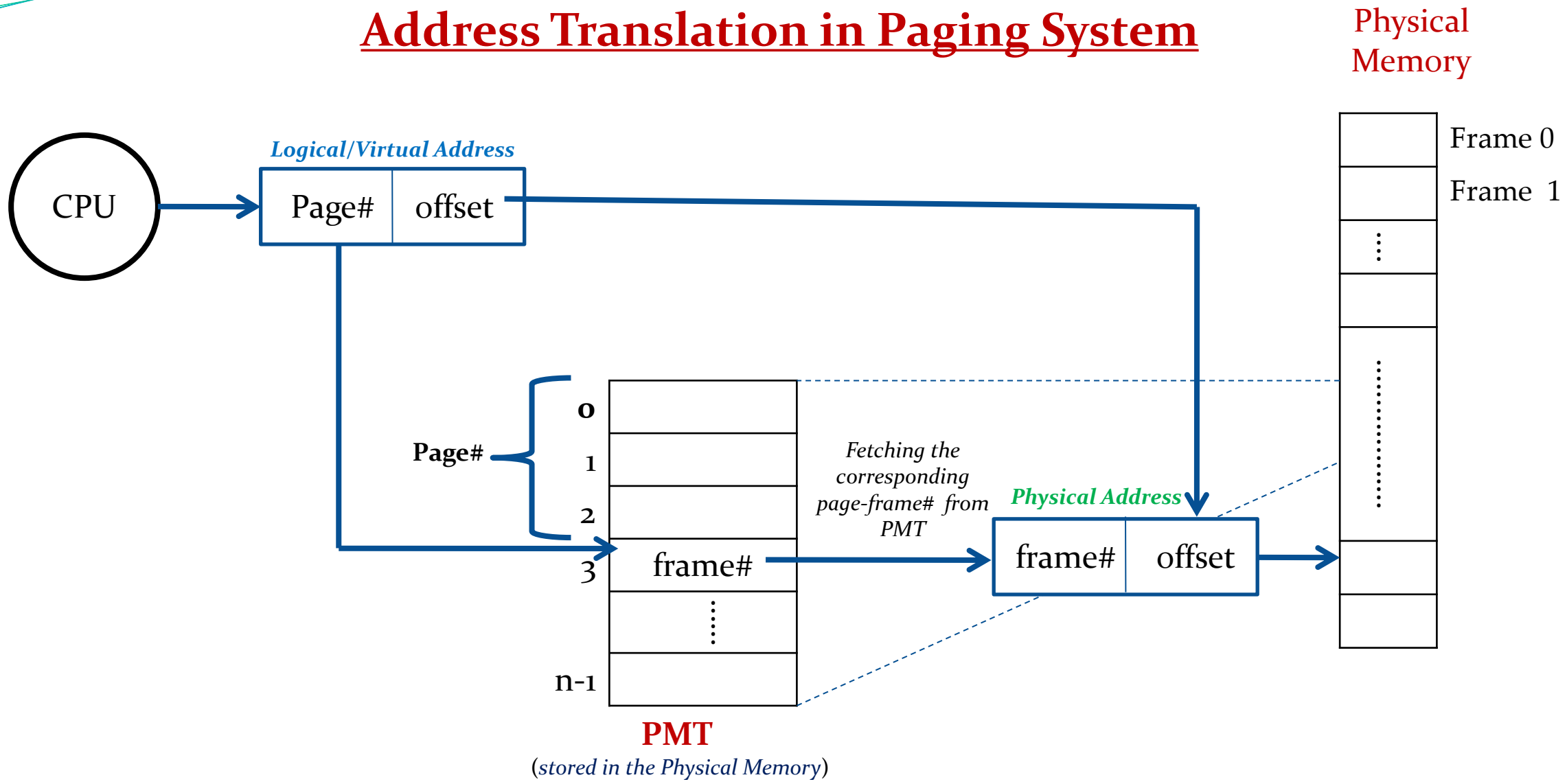
- Paging memory management scheme removes the requirement of contiguous allocation of physical memory
- Address mapping is used to maintain the illusion contiguity of virtual address space of a process despite of discontiguous placement in the physical memory
- In paging, the physical memory is conceptually divided in to a number of fixed size slots called *page frames*
- Similarly, the virtual address space of a process is also split in to fixed size block of same size(as that of frame) called *pages*.
- Allocation of memory consist of finding a sufficient number unused page frames for loading of requesting process's pages.

Principles of Operation

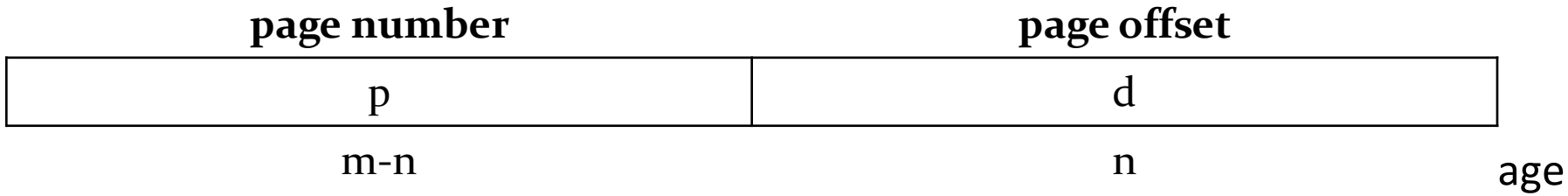
- The mapping of virtual address to physical address in paging system is paging system at the page level
- The basic method for implementing paging involves breaking physical memory into fixed-sized blocks called frames and breaking logical memory into blocks of the same size called pages. When a process is to be executed, its pages are loaded into any available memory frames
- Every address generated by the CPU is divided into two parts: a page number(p) and a page-offset(d) within that page. The page number is used as an index into a page table. The page table contains the base address of each page in physical memory. This base address is combined with the page offset to define the physical memory address that is sent to the memory unit.

- The offset within the page and the page-frame are identical and need not be mapped (because page and the page-frame have identical size)
- The address translation is performed with the aid of a mapping table called Page-Map Table (PMT).
  - The PMT is constructed at process loading time in order to establish the correspondence between the virtual and physical addresses.
  - The value of each entry in PMT is the page-frame number (the higher order bits of the page-frame's base address)in the physical memory where the corresponding virtual page is placed. So PMT entries are filled with page-frame numbers of the region where the corresponding pages are actually loaded.

Address Translation in Paging System

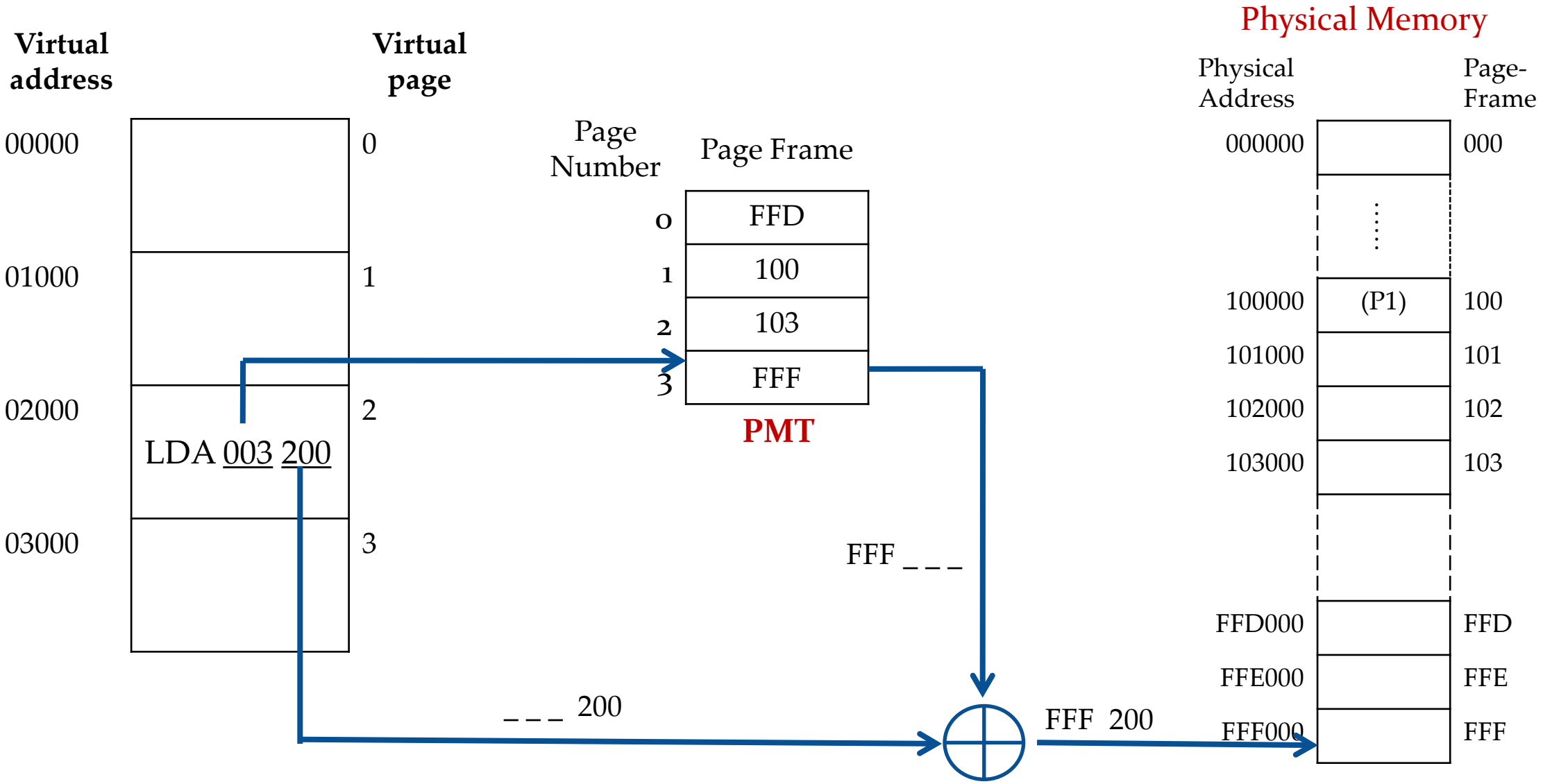


The page size (like the frame size) is defined by the hardware. The size of a page is a power of 2, varying between 512 bytes and 1 GB per page, depending on the computer architecture. The selection of a power of 2 as a page size makes the translation of a logical address into a page number and page offset particularly easy. If the size of the logical address space is 2^m, and a page size is 2^n bytes, then the high-order m− n bits of a logical address designate the page number, and the n low-order bits designate the page offset. Thus, the logical address is as follows:



age

Illustration of Address Translation



Problems-1&2

Consider a logical address space of 64 pages of 1,024 words each, mapped onto a physical memory of 32 frames.

a. How many bits are there in the logical address?

   Answer: 16 bits

a. How many bits are there in the physical address?

   Answer: 15 bits

Consider a logical address space of 256 pages with a 4-KB page size, mapped onto a physical memory of 64 frames.

a. How many bits are required in the logical address? Answer: 20 bits
b. How many bits are required in the physical address? Answer: 18 bits

Memory Map table

- The operating system keeps track of the status of each page frame by means of physical memory map table (stored as static table).
  - This table is referred as Memory-map Table
  - Each entry of the MMT describe the status FREE or ALLOCATED of the page-frames in the physical memory
  - Therefore the MMT has fixed number of entries

$$f = \frac{m}{p}$$

Where

$f \rightarrow$ the total number of page-frames in the physical memory

$m \rightarrow$ the capacity of the installed physical memory

p $\rightarrow$ the page size(= page-frame size)

Here, both *m* and p are integer power of 2, so *f* is an integer

Page-frame Allocation to a Requesting Process

- When requested to load a process of size ***s***, the operating system must allocate ***n*** free frames, so that

$$n = \left\lceil \frac{s}{p} \right\rceil$$

Where

p $\rightarrow$ the page size

$\lceil\ \rceil$ $\rightarrow$ the ceiling function

- The operating system allocates memory in terms of an integral number of page-frames
- If the size of a given process is not a multiple of the page size, the last frame may be partially used. This phenomenon is known as page fragmentation or page breakage.
- Any placement strategy (first-fit, best-fit and worst-fit) is as good as others since all frames fit all pages

Hardware Support for Paging

- Each Page-Map Table (PMT) must be large enough to accommodate the maximum size allowed for a address space of a process in a given system, for example :

  Memory Size:16MB and Page Size: 256 B
  Total Number of Page frame: $2^{24}/2^{8} = 2^{16}$ =64K
  Individual PMT entries are Frame Number= 16 bits=2 B
  Therefore the size of the PMT is 128KB

  - This is the size of one PMT. The total storage for all PMTs consume a significant port of the physical memory.

- Normally the actual address space is well below its allowable maximum, so it will be reasonable to construct each PMT only with as many entries as its related process has pages

Hardware Support for Paging (Cont.)

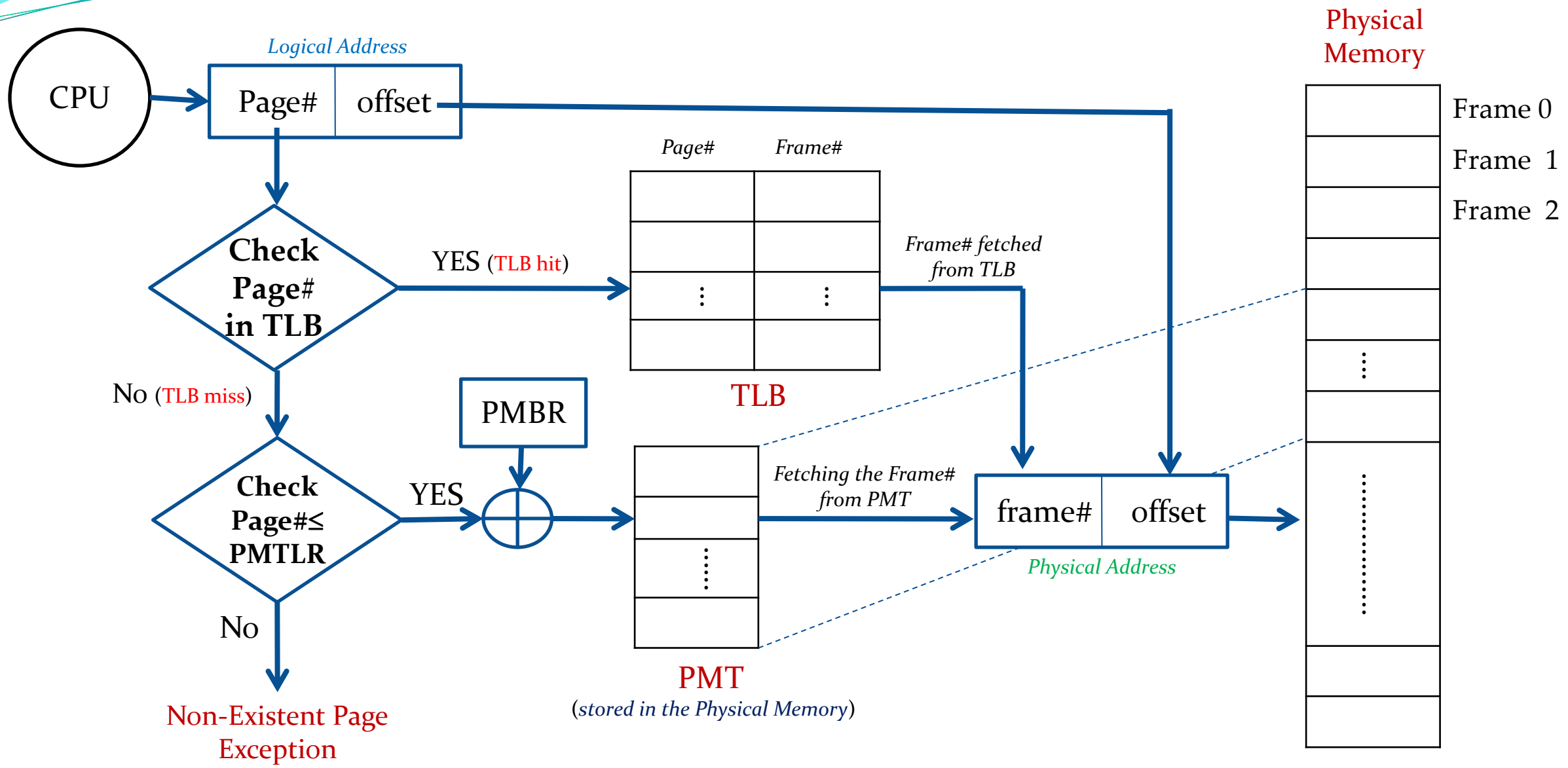
- This can be implemented with a dedicated hardware consist of two special registers namely:

  Page Map Table Limit Register (**PMTLR**)

  Page Map Table Base Register (**PMTBR**)

  Here these values are defined at the time of process loading and stored in PCB

- Even after this, the address translation in paging system still required two memory references.
  - One for accessing the PMT for mapping and
  - Other for accessing the target item in the physical memory.
  - This results in a 50% reduction of memory bandwidth.
- This problem can be alleviated by providing the specialized hardware to speedup the address translation process
  - One approach is to use a high speed associative memory for storing a subset of often used page map table entries and this memory is called Translation Look-aside Buffer(TLB) or mapping cache


CPU
Logical Address
Page#
offset
Check Page# in TLB
YES (TLB hit)
Page#
Frame#
Frame# fetched from TLB
TLB
No (TLB miss)
PMBR
Check Page#≤ PMTLR
YES
Fetching the Frame# from PMT
frame#
offset
Physical Address
No
Non-Existent Page Exception
PMT
(stored in the Physical Memory)
Physical Memory
Frame 0
Frame 1
Frame 2

Effective Memory Access Time using TLB

The effective memory access time, t_{eff} is the sum of the address translation time, t_{TR} and the subsequent access time needed to fetch the target item from memory, t_M

$$t_{eff} = t_{TR} + t_M$$

With TLB used to assist in address translation. t_{TR} becomes

$$t_{TR} = ht_{TLB} + (1 - h)(t_{TLB} + t_M) = t_{TLB} + (1 - h)t_M$$

Where

$h \rightarrow$ TLB *hit ratio (*the ratio of address translations that are completed through the TLB*)*

$t_M \rightarrow$ the memory access time

$t_{TLB} \rightarrow$ the TLB access time

Effective Memory Access Time with TLB is

$$t_{eff} = t_{TLB} + (2 - h)t_M$$

Problem-3

- If TLB access time is 10 times as fast as that of the main memory and the hit ratio is 90% then what will be the t_{eff} ?

  Given $t_{TLB} = 0.1t_M$ *and* $h = 0.9$

Answer

$$t_{eff} = t_{TLB} + (2 - h)\, t_M$$

$$t_{eff} = 0.1t_M + (2-0.9)\, t_M = 1.21t_M$$

Problem-4

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

a. If a memory reference takes 50 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 2 nanoseconds time, if the entry is there.)

Answer:

a. 100 nanoseconds;

50 nanoseconds to access the page table and

50 nanoseconds to access the word in memory.

b. Effective access time = (2 nanoseconds) + (1.25 × 50 nanoseconds)

Problem-5

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.)

Answer:

a. 400 nanoseconds; 200 nanoseconds to access the page table and 200 nanoseconds to access the word in memory.
b. Effective access time = 0.75 × (200 nanoseconds) + 0.25 × (400 nanoseconds) = 250 nanoseconds.