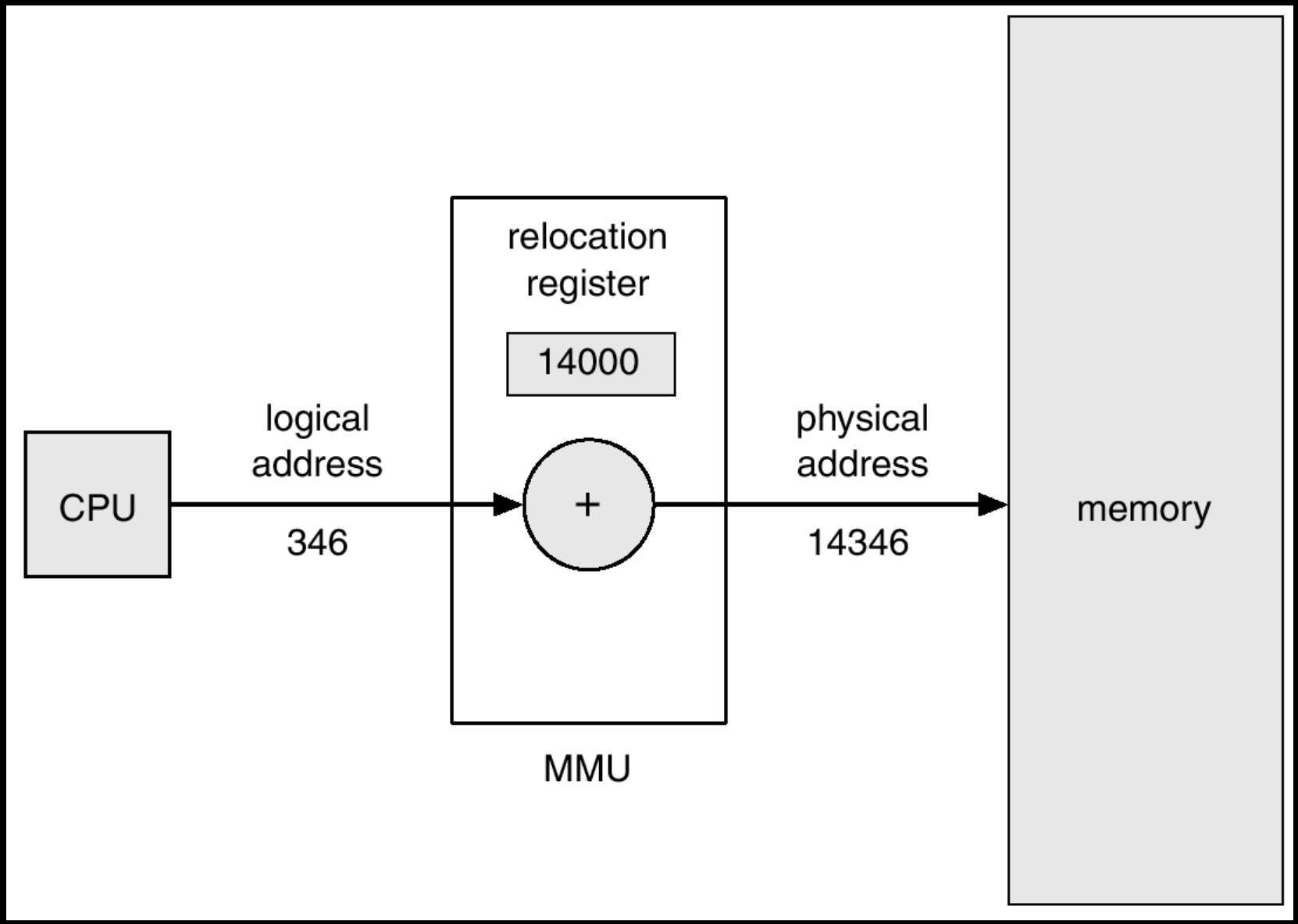
Memory Management

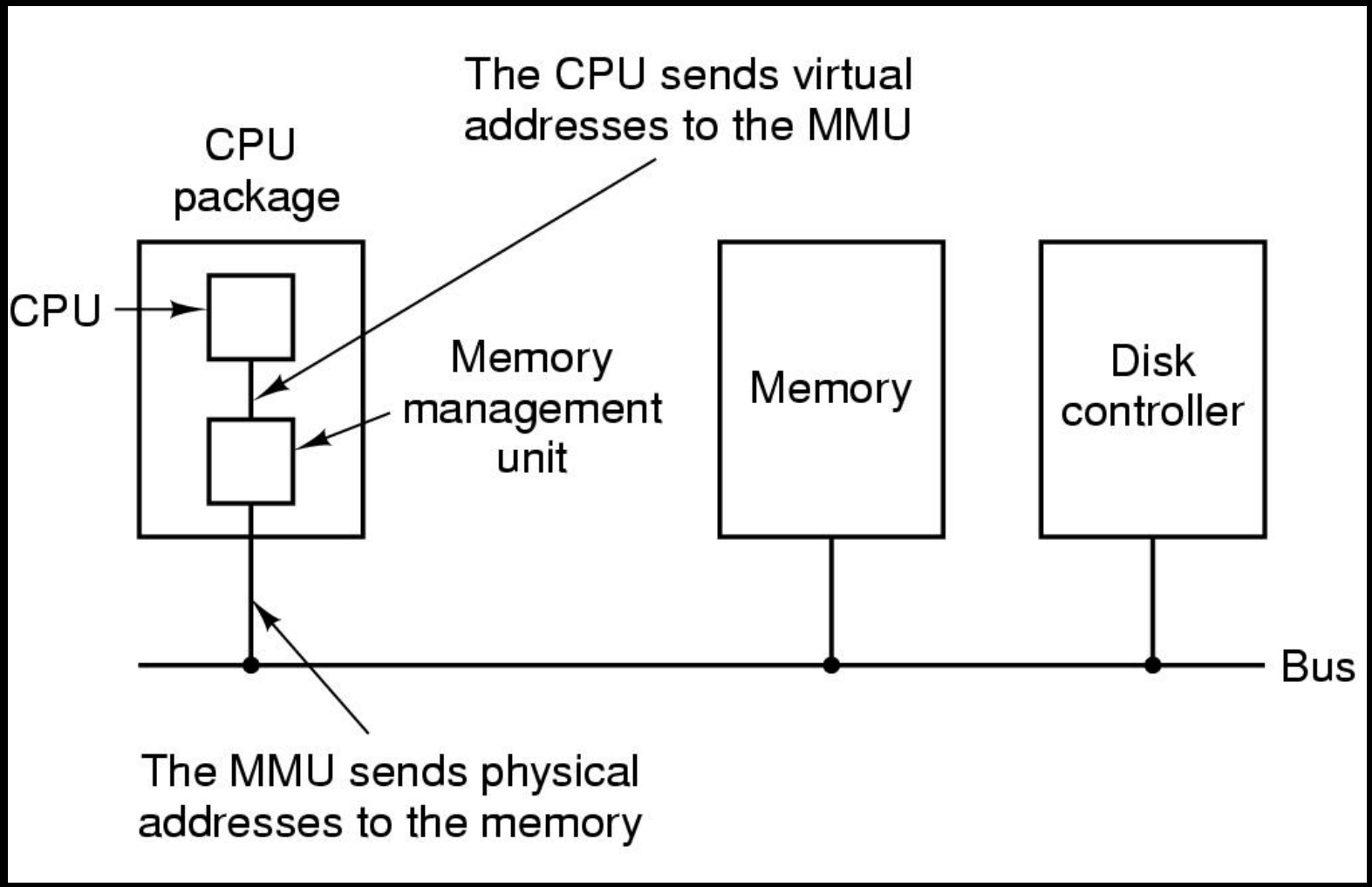
Memory Management – Virtual Space

- Run applications larger than physical memory.
- Run partially loaded programs.
- Multiprogramming: More than one program simultaneously reside in memory.
- Allow relocatable programs – anywhere, anytime
- Application Portability:
  - Applications should not have to manage memory resources
  - Write machine independent code – program should not depend on memory architecture.
- Permit sharing of memory segments or regions. For example, read-only code segments should be shared between program instances.

Memory Management – Dynamic Reallocation



Memory Management



Memory Management - Paging

- Job address space is divided into equal pieces called pages
- Pages are units of memory that are swapped in and out of Primary memory
- Pages are grouped together for a given user job and are placed in Page tables
- Physical memory is also divided into pieces of equal size called Blocks / frames
- A mapping is provided between the pages and the frames
- The pages remain logically contiguous

Memory Management - Paging

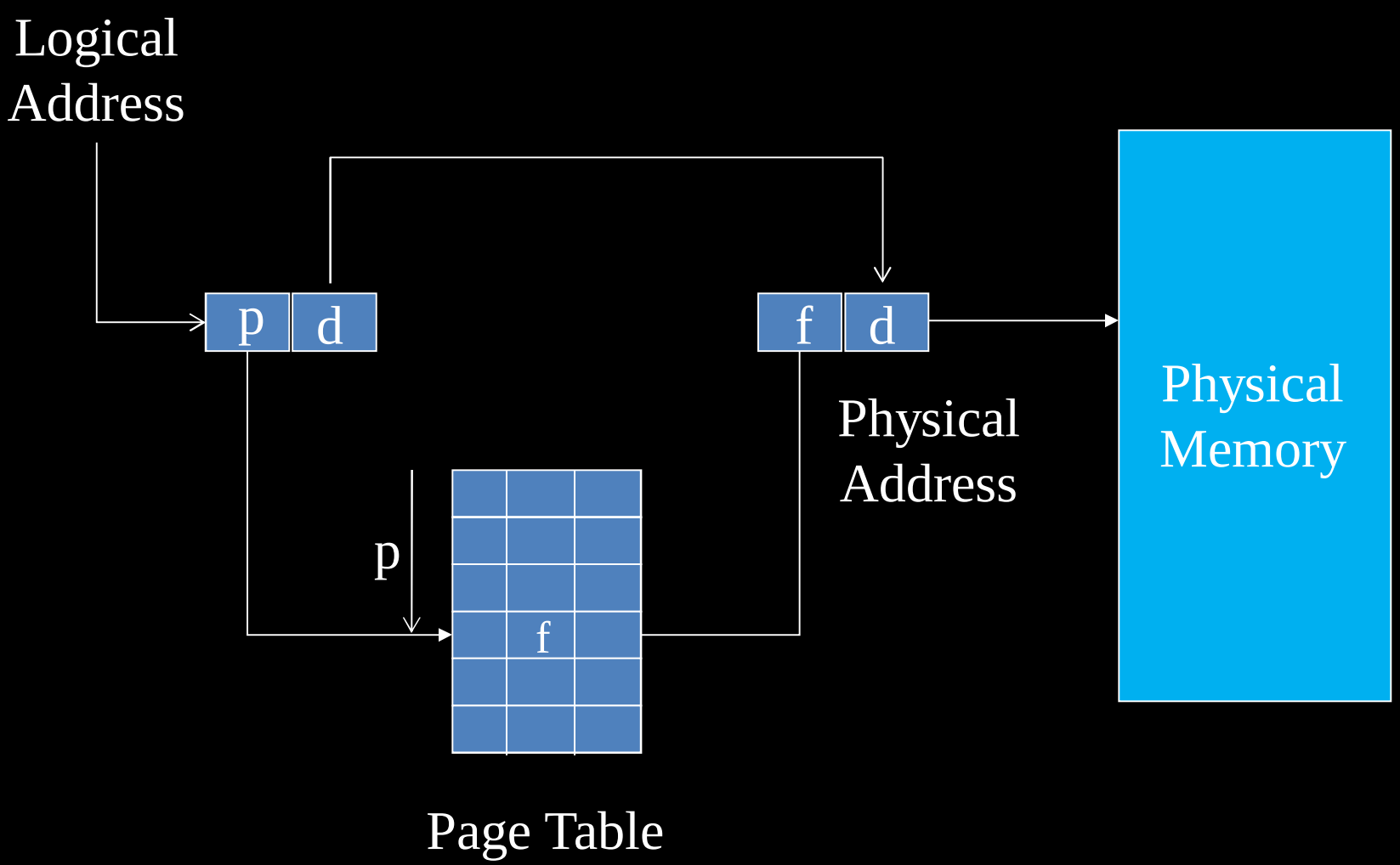
Advantage:

- As each page is separately allocated, the users job need not be contiguous.
- Decreases or eliminates the fragmentation
- High degree of multiprogramming
- Increased processor and memory utilization

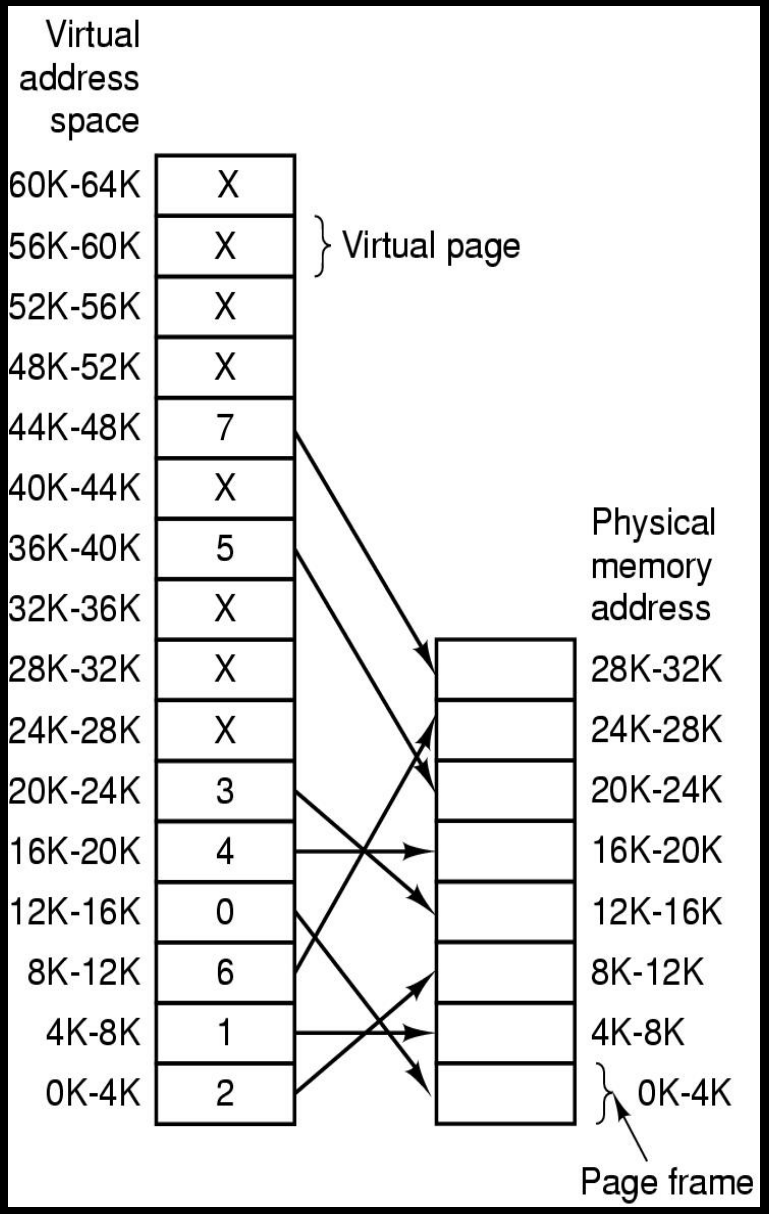
Disadvantages:

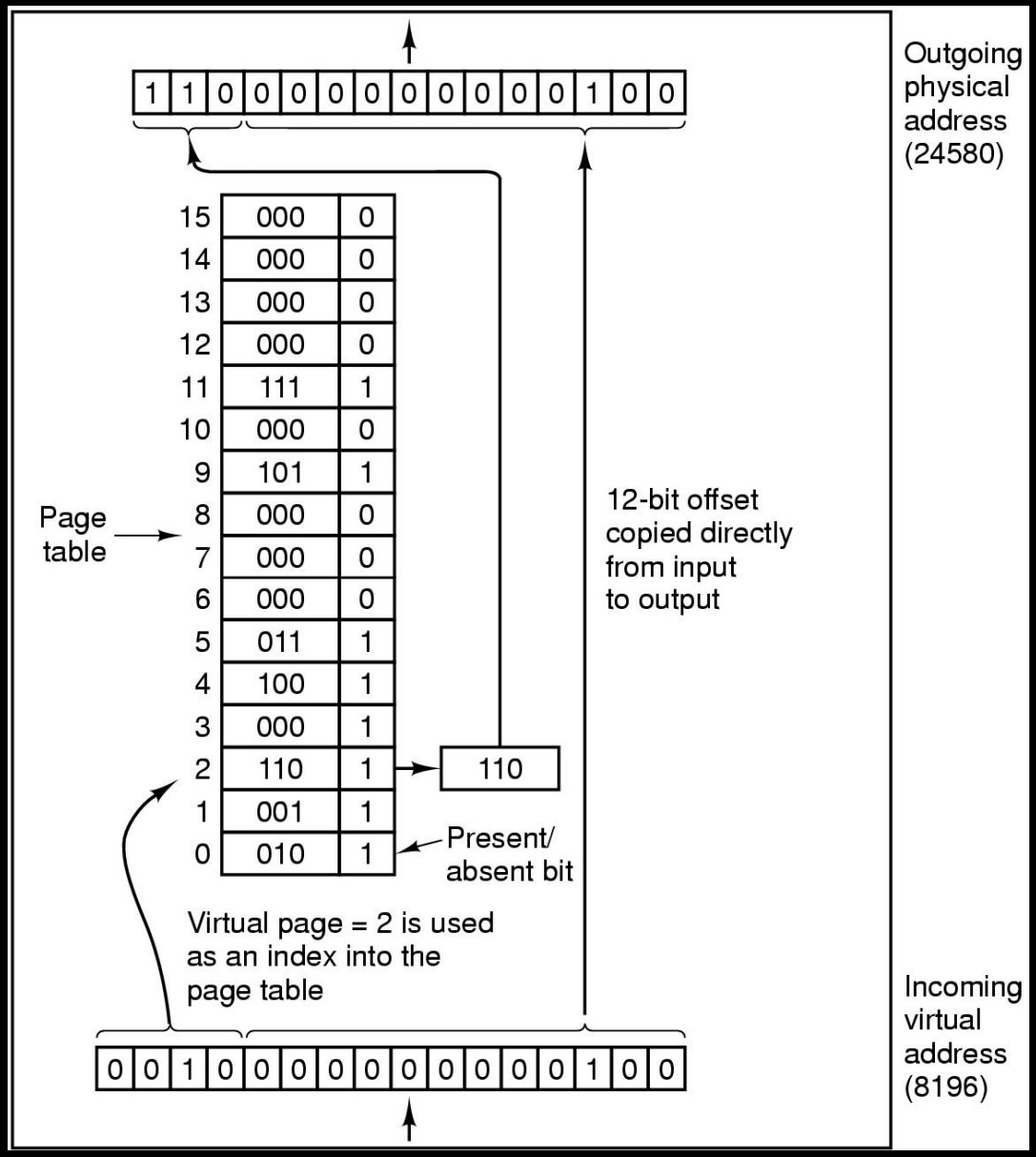
- Extra memory required for storing page tables
- Considerable amount of hardware support is required for address transformations etc.
- All pages of entire job must be in memory

Memory Management – Address Translation



Memory Management – Virtual Memory





Memory Management - TLB

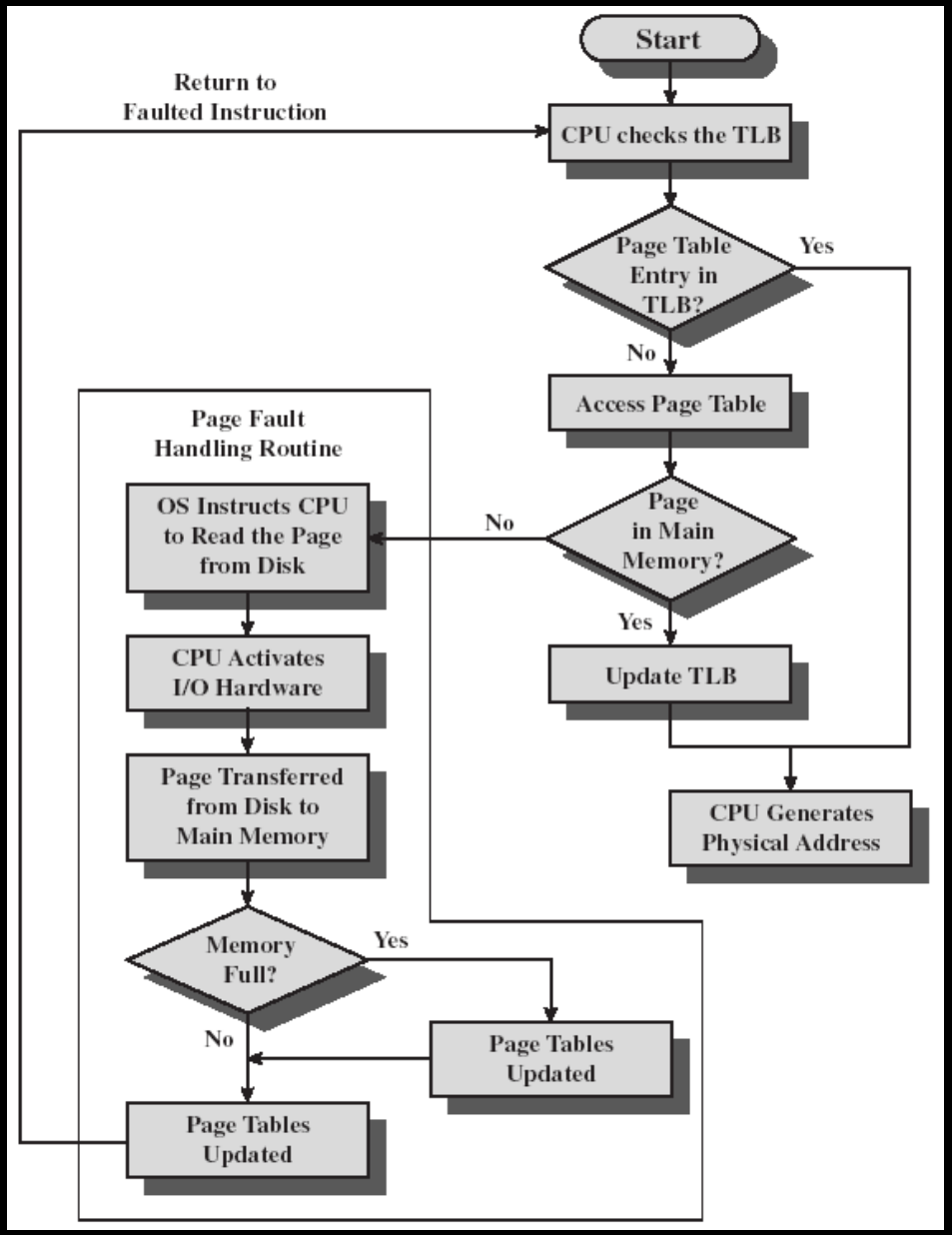
Each virtual memory reference can cause two physical memory Access

- one to fetch the page table
- one to fetch the data

To overcome this problem a high-speed cache is set up for page table entries ,called the TLB - Translation Look-aside Buffer

TLB Contains page table entries that have been most recently used. Functions same way as a memory cache

Memory Management - TLB



Given a virtual address, processor examines TLB

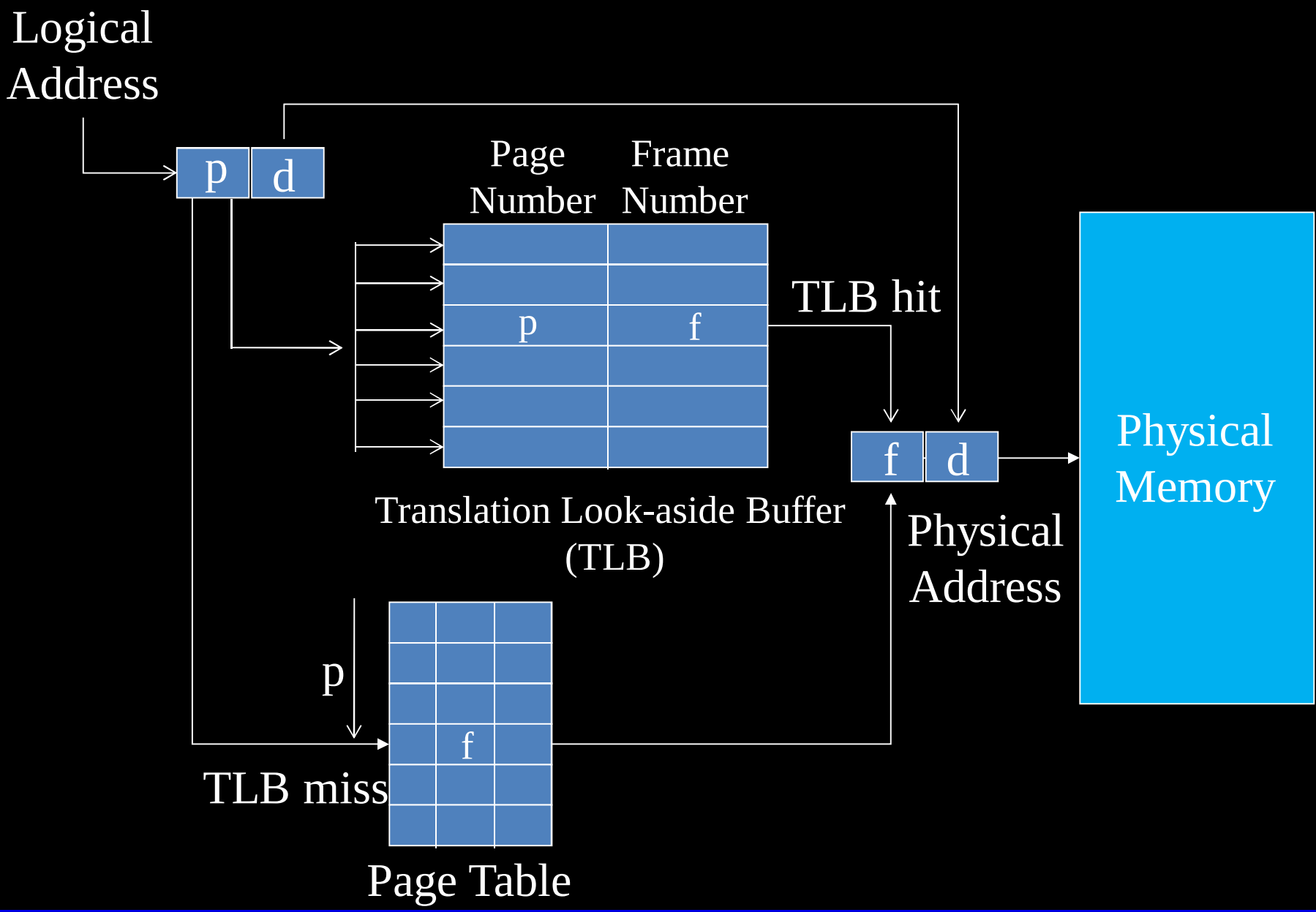
If page table entry is presents (a hit), the frame number is retrieved and the real address is formed

If page table entry is not found in the TLB (a miss), the page number is used to index the process page table

First checks if page is already in main memory if not in main memory a page fault is issued

The TLB is updated to include the new page entry

Memory Management – TLB / Associative Register



Question & Answer

Consider a paging system with the page table stored in memory. If a memory reference takes 200 nanoseconds, how long does a paged memory reference take? 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.)

400 nanoseconds; 200 nanoseconds to access the page table and 200 nanoseconds to access the word in memory.

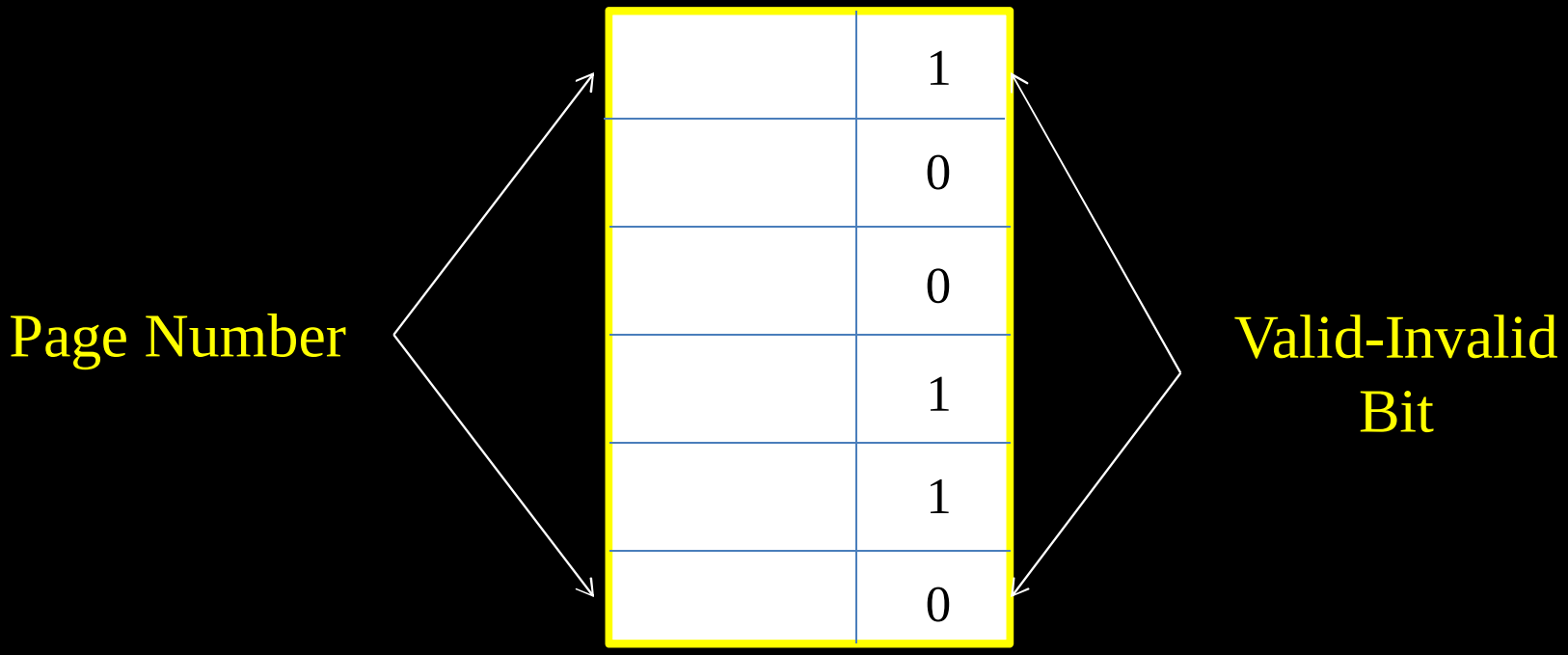
Effective access time = 0.75 × (200 nanoseconds) + 0.25 × (400 nanoseconds) = 250 nanoseconds.

Memory Management – Valid-Invalid Bit

With each page table entry a valid–invalid bit is associated (1 ⇒ in-memory, 0 ⇒ not-in-memory)

Initially valid–invalid bit is set to 0 on all entries.

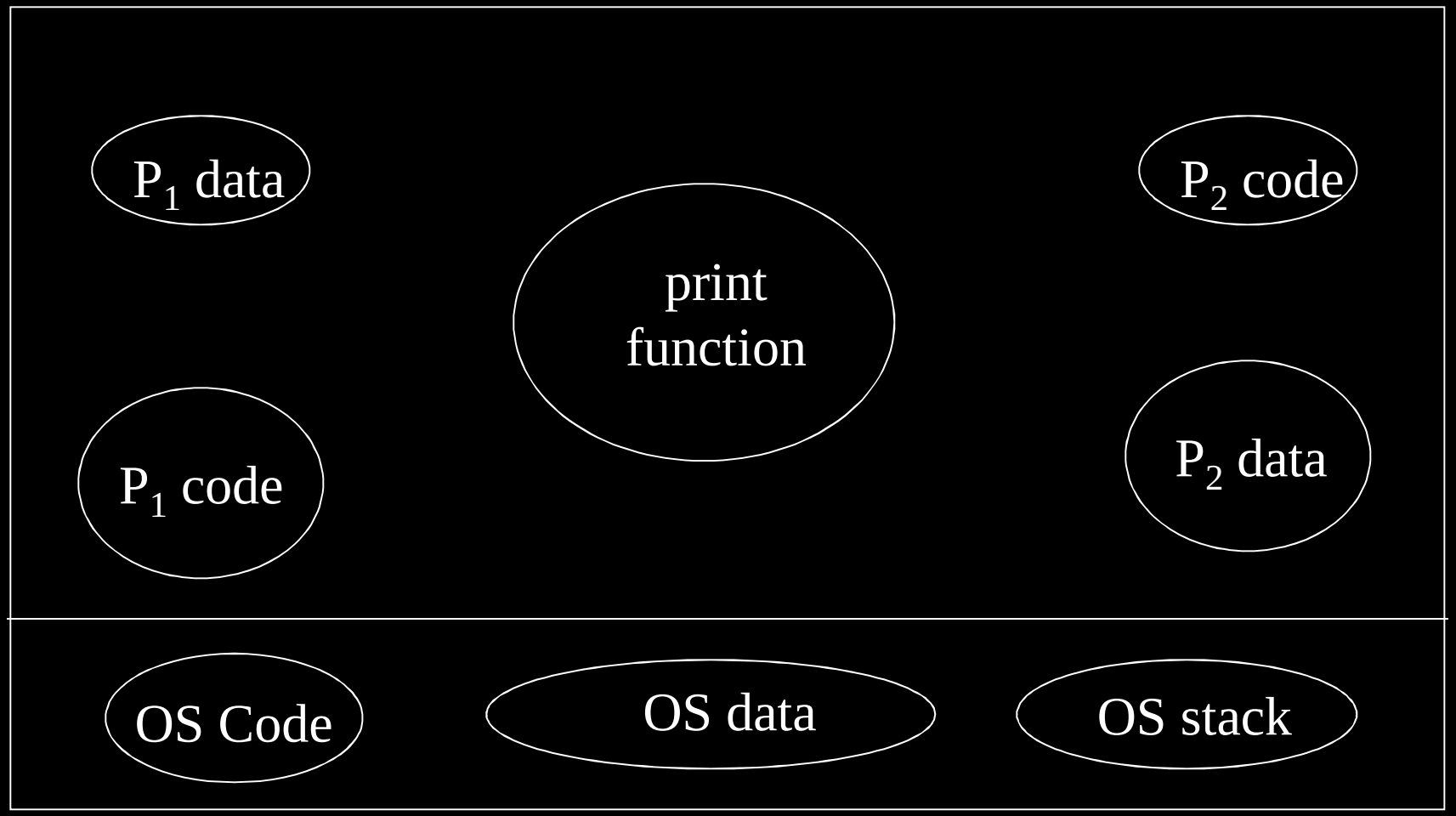
During address translation, if valid–invalid bit in page table entry is 0 ⇒ page fault.



Memory Management - Segmentation

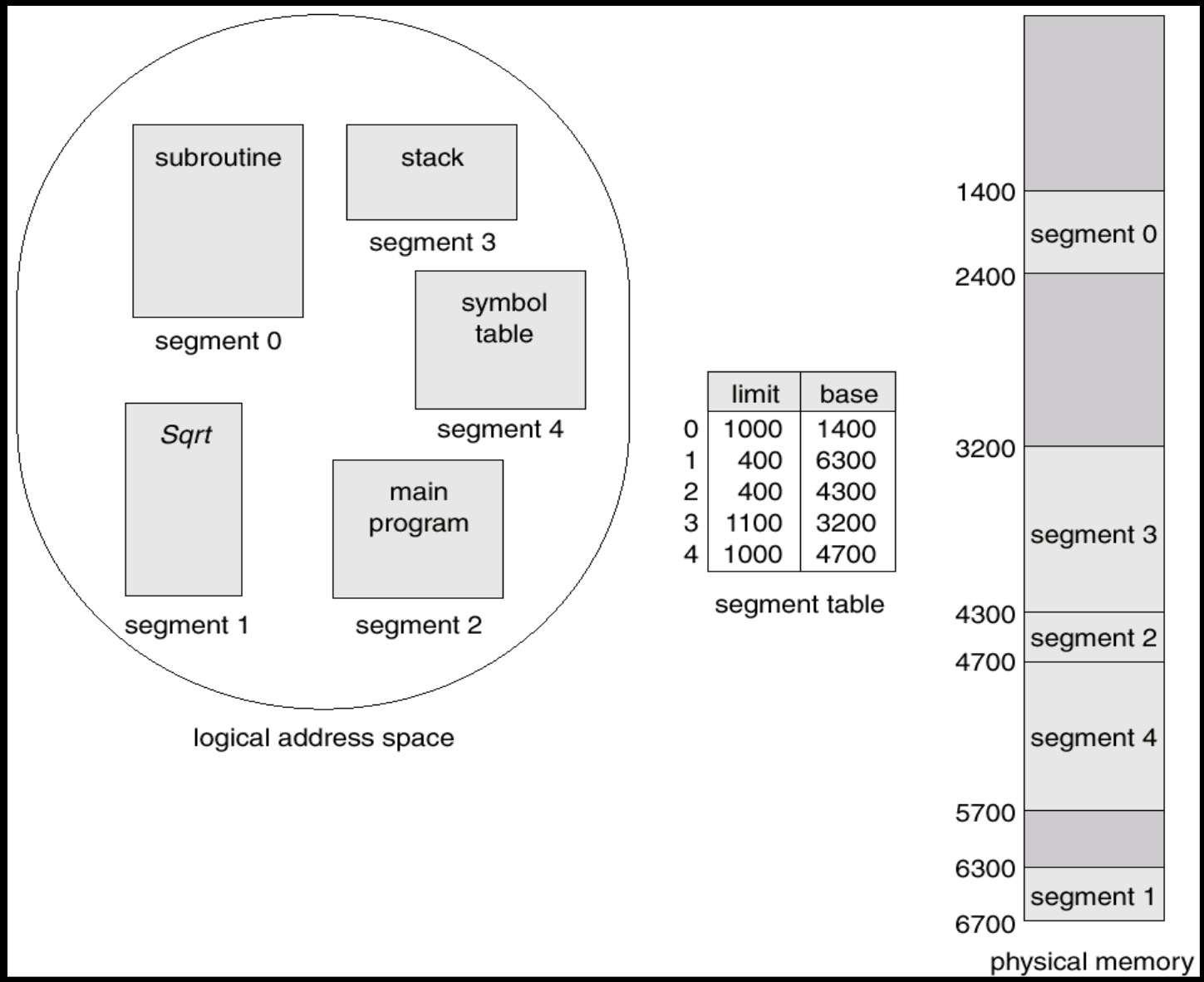
- Segmentation is a technique for breaking memory up into logical pieces
- Each “piece” is a grouping of related information
  - data segments for each process
  - code segments for each process
  - data segments for the OS
  - etc.
- Like paging, use virtual addresses and use disk to make memory look bigger than it really is
- Segmentation can be implemented with or without paging

Memory Management - Segmentation



Logical Address Space

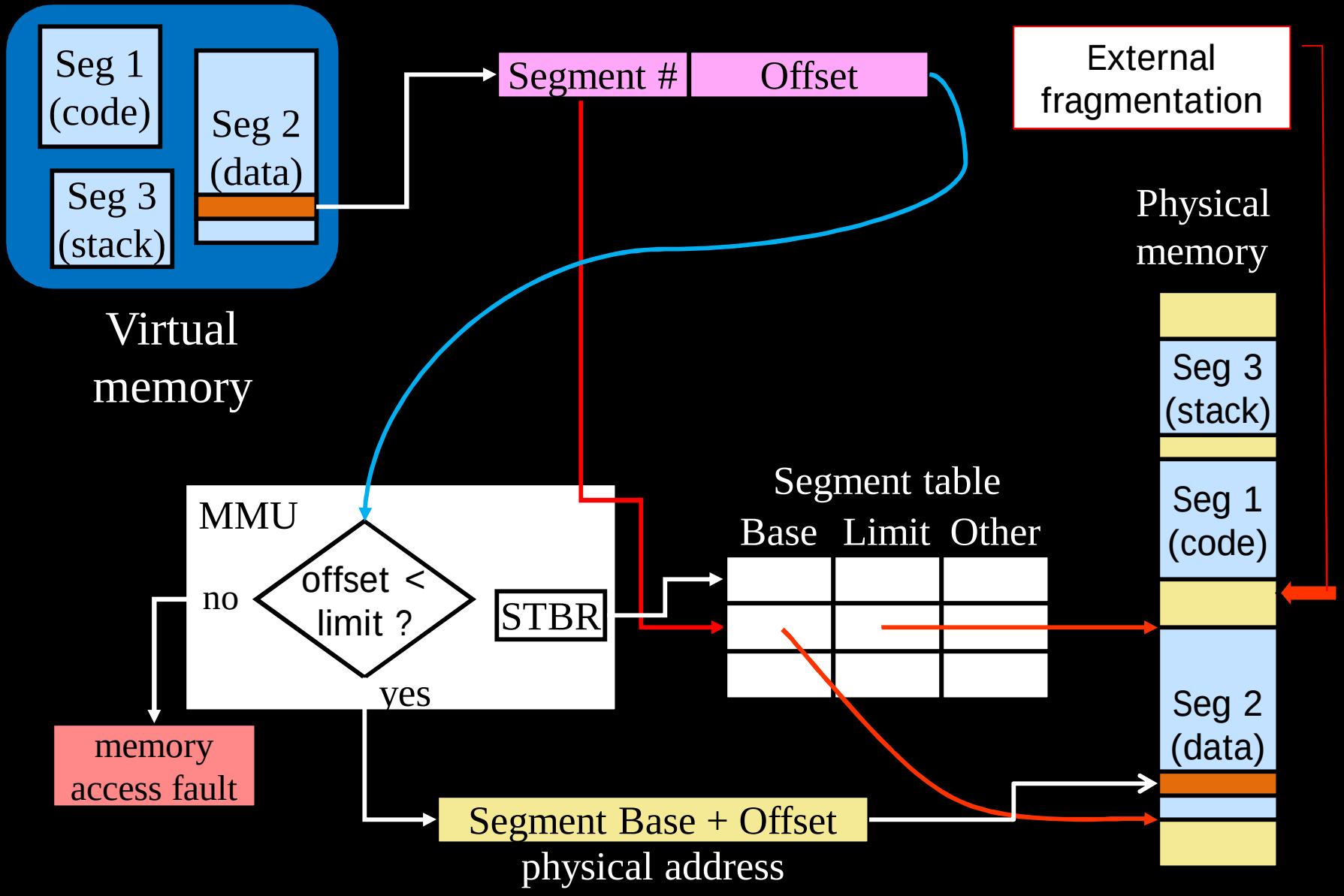
Memory Management - Segmentation Example



Memory Management - Segmentation

- Sounds very similar to paging
- Big difference – segments can be variable in size
- As with paging, to be effective hardware must be used to translate logical address
- Most systems provide segment registers
- If a reference isn't found in one of the segment registers
  - trap to operating system
  - OS does lookup in segment table and loads new segment descriptor into the register
  - return control to the user and resume
- Again, similar to paging

Memory Management - Segmentation



Memory Management - Segmentation

Advantage

- As opposed to paging:
  - No internal fragmentation
  - May save memory if segments are very small and should not be combined into one page (e.g. for reasons of protection)
  - Segment tables: only one entry per actual segment as opposed to per page in VM
  - Average segment size >> average page size
    $\Rightarrow$ less overhead (smaller tables)

Disadvantage

- External fragmentation
- Costly memory management algorithms
  - Segmentation: find free memory area big enough (search!)
  - Paging: keep list of free pages, any page is ok (take first!)
- Segments of unequal size not suited as well for swapping

Question & Answer

Consider the following segment table:

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

What are the physical addresses for the following logical addresses?

a. 0,430 b. 1,10 c. 2,500 d. 3,400 e. 4,112

a. 219 + 430 = 649 b. 2300 + 10 = 2310
c. illegal reference, trap to operating system
d. 1327 + 400 = 1727
e. illegal reference, trap to operating system

Question & Answer

Compare the main memory organization schemes between Contiguous Memory Allocation, Pure Segmentation, and Pure Paging.

Contiguous Memory Allocation scheme suffers from external fragmentation as address spaces are allocated contiguously and holes develop as old processes dies and new processes are initiated. It also does not allow processes to share code.

Pure Segmentation also suffers from external fragmentation, however, it enables processes to share code, for instance, two different processes could share a code segment but have distinct data segments.

Pure Paging does not suffer from external fragmentation, but instead, it suffers from internal fragmentation. Processes are allocated in pages and if a page is not completely utilized, it results in internal fragmentation and a corresponding wastage of space occur. Paging also enables processes to share code at the granularity of pages.

Question & Answer

On a system with paging, a process cannot access memory that it does not own; why? How could the operating system allow access to other memory? Why should it or should it not?

An address on a paging system is a logical page number and an offset. The physical page is found by searching a table based on the logical page number to produce a physical page number. Because the operating system controls the contents of this table, it can limit a process to accessing only those physical pages allocated to the process. There is no way for a process to refer to a page it does not own because the page will not be in the page table. To allow such access, an operating system simply needs to allow entries for non-process memory to be added to the process's page table. This is useful when two or more processes need to exchange data. They just read and write to the same physical addresses (which may be at varying logical addresses). This makes for very efficient inter-process communication.

Question & Answer

Compare paging with segmentation with respect to the amount of memory required by the address translation structures in order to convert virtual addresses to physical addresses.

Paging requires more memory overhead to maintain the translation structures. Segmentation requires just two registers per segment: one to maintain the base of the segment and the other to maintain the extent of the segment. Paging on the other hand requires one entry per page, and this entry provides the physical address in which the page is located.

Question & Answer

Why are page sizes always power of 2? Consider a logical address space of eight pages of 1024 words each, mapped onto a physical memory of 32 frames.

a) How many bits are there in the logical address?
b) How many bits are there in the physical address?

All addresses are binary and are divided into page or page frame number and offset. By making the page size a power of 2, the page number, the page frame number and the offset can be determined by looking at particular bits in the address. No mathematical calculation is required. The physical address can be generated by concatenating the offset to the frame number.