Operating Systems

Lecture 7

Memory management

Summary

- This lecture shows the problem of memory management:
  - Introduction
  - Continuous allocation
  - External fragmentation and compactification
  - Memory allocation strategies
  - Segmentation
  - Paging
  - Internal fragmentation
  - Memory management in systems with a lot of memory

Summary

- This lecture shows the problem of memory management:
  - Introduction
  - Continuous allocation
  - External fragmentation and compactification
  - Memory allocation strategies
  - Segmentation
  - Paging
  - Internal fragmentation
  - Memory management in systems with a lot of memory

Introduction

- Today we're discussing operating (primary) memory only.
- Memory is an array of words, ordered according to their addresses.
  - Each word consists of bits, depending on the machine's processor architecture (32, 64, 128). Words usually are divided into bytes (usually 8-bit long).
  - Each memory location is identified by its address, which is a binary number.

What is „memory management”?

- There may be several processes running in a computer system.
- Each process should have a dedicated, separate memory space allocated to it.
  - If processes' memory spaces are not separated, they can interfere one with another, causing errors in execution.
- In a multi-program computer system, it is hard to determine process location in a memory at compile-time. Therefore processes must be compiled in such way, that they can flexibly adapt to their placement in memory.
  - In order to do it, a process should use **virtual addresses**, which are converted to physical ones by the CPU's part caled **Memory Management Unit** (MMU).

Memory Management Unit (MMU)

- Hardware device that at run time maps virtual to physical address.
- Many methods possible, covered in the rest of this lecture.
- The user program deals with logical addresses; it never sees the real physical addresses.
  - Execution-time binding occurs when reference is made to location in memory.
  - Logical address bound to physical addresses.

Summary

- This lecture shows the problem of memory management:
  - Introduction
  - Continuous allocation
  - External fragmentation and compactification
  - Memory allocation strategies
  - Segmentation
  - Paging
  - Internal fragmentation
  - Memory management in systems with a lot of memory

Continuous allocation

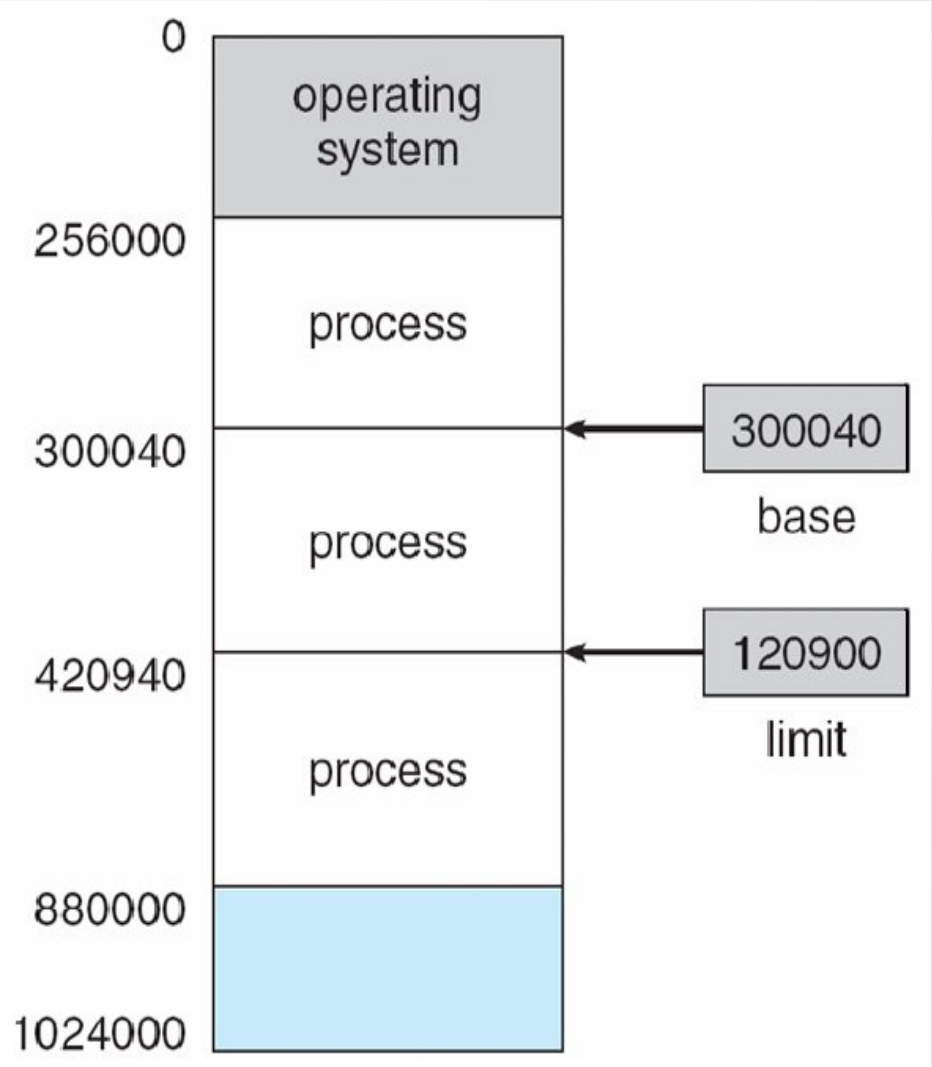
- Main memory must support both OS and user processes.
- Limited resource, must allocate efficiently.
- Main memory usually into two partitions:
  - Resident operating system, usually held in low memory with interrupt vector.
  - User processes then held in high memory.
  - Each process contained in single continuous section of memory.

Continuous allocation (cont.)

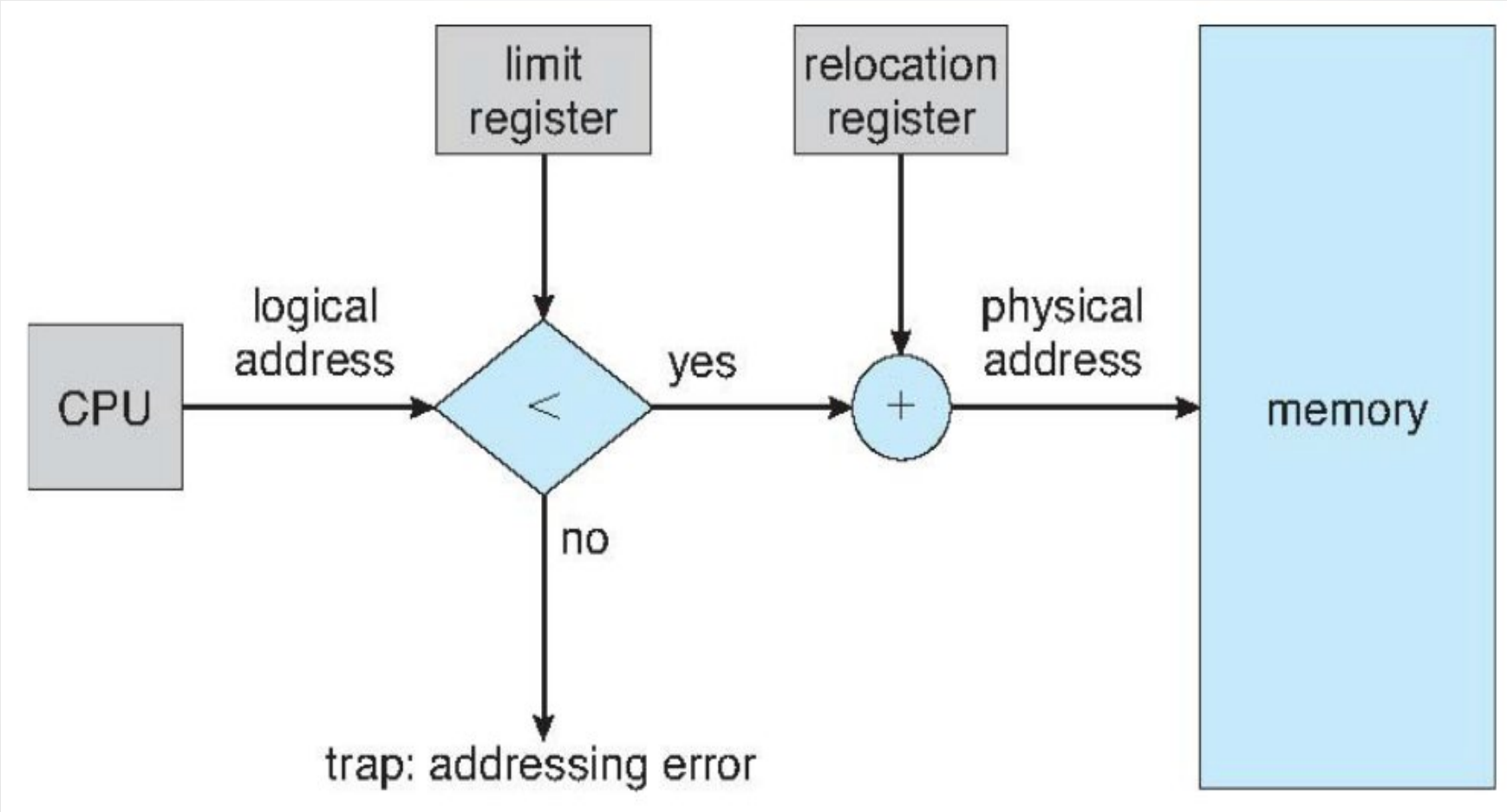
- Relocation registers used to protect user processes from each other, and from changing operating-system code and data:
  - Base register contains value of smallest physical address.
  - Limit register contains range of logical addresses – each logical address must be less than the limit register.
  - MMU maps logical address dynamically.
  - Can then allow actions such as kernel code being transient and kernel changing size.

Base and Limit Registers

- A pair of **base (relocation)** and **limit** registers define the logical address space.
- CPU must check every memory access generated in user mode to be sure it is between base and limit for that user.



MMU

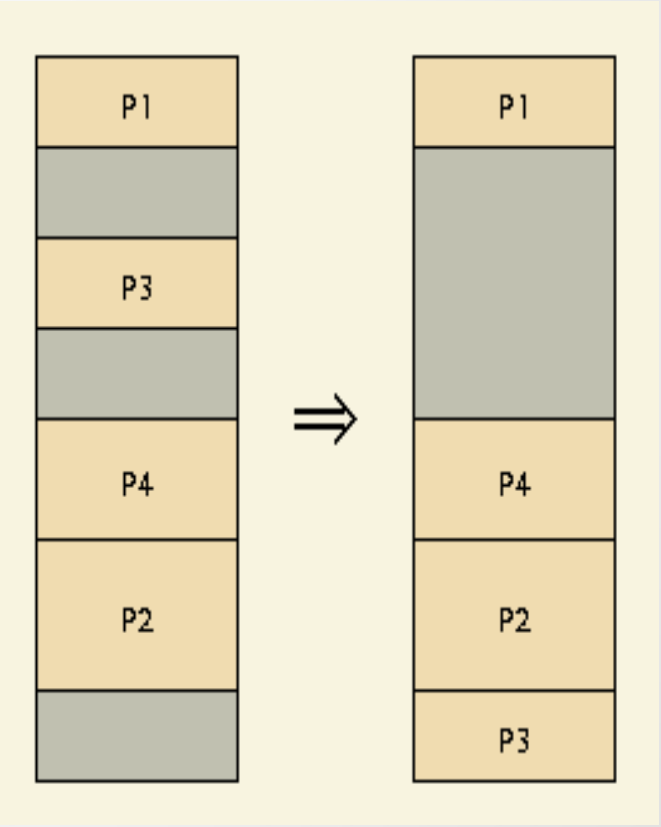

limit register
relocation register
logical address
CPU
<
yes
+
physical address
memory
no
trap: addressing error

Summary

- This lecture shows the problem of memory management:
  - Introduction
  - Continuous allocation
  - External fragmentation and compactification
  - Memory allocation strategies
  - Segmentation
  - Paging
  - Internal fragmentation
  - Memory management in systems with a lot of memory

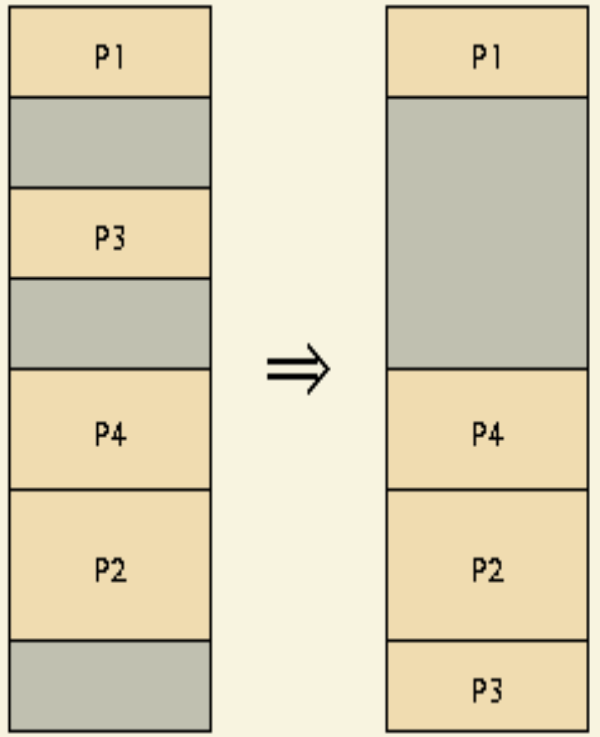
Memory fragmentation

- Processes appear and vanish in the operating system memory (they are executed and finished). After a process is finished, its memory is released. But it may happen, that – due to an order of allocate/free operations – the available space is scattered across the physical memory.
- The same situation can be observed in memory space of one process – dynamic memory allocation/deallocation corresponds to process creation/termination.
- A situation, when certain memory is not occupied although divided into parts, separated with memory assigned to processes, is calle **fragmentation**.
  - Fragmentation is **external**, when it concerns physical memory.
  - Framentation is **internal**, when it concerns logical memory of a process.



Compactification

- In order to reclaim all the unassigned space and create one block of unused memory, an operating system can perform **compactification** (memory defragmentation).
- This operation can be easily applied on the level of processes, although is hard to be used in case of internal process memory.
  - It is time-consuming – whole blocks of allocated memory must be moved.



Summary

- This lecture shows the problem of memory management:
  - Introduction
  - Continuous allocation
  - External fragmentation and compactification
  - Memory allocation strategies
  - Segmentation
  - Paging
  - Internal fragmentation
  - Memory management in systems with a lot of memory

Memory allocation strategies

- Memory allocation strategy should aim at minimalization of fragmentation.
- There are many possible strategies:
  - **First-fit** – assign memory in the first (according to addresses) memory area, which is large enough to fulfill the request requirements
    - May be used with „cyclic” modification – algorithm seeks for available block from the address, where last assignment was made.
  - **Best-fit** – assign memory in the smallest memory block, which is still large enough to fulfill the request requirements
  - **Worst-fit** – assign memory in the largest memory block available

Memory allocation strategies

- All these allocation strategies can be applied for allocation of memory for the whole program, as well as for allocation of dynamic memory inside the program (and many other situations like heap management).
- All these strategies are only heuristics – their actual performance depends on parameters of consecutive allocation/deallocation requests.
  - We may show sequences of requests, for which any of these strategies will be better than others.

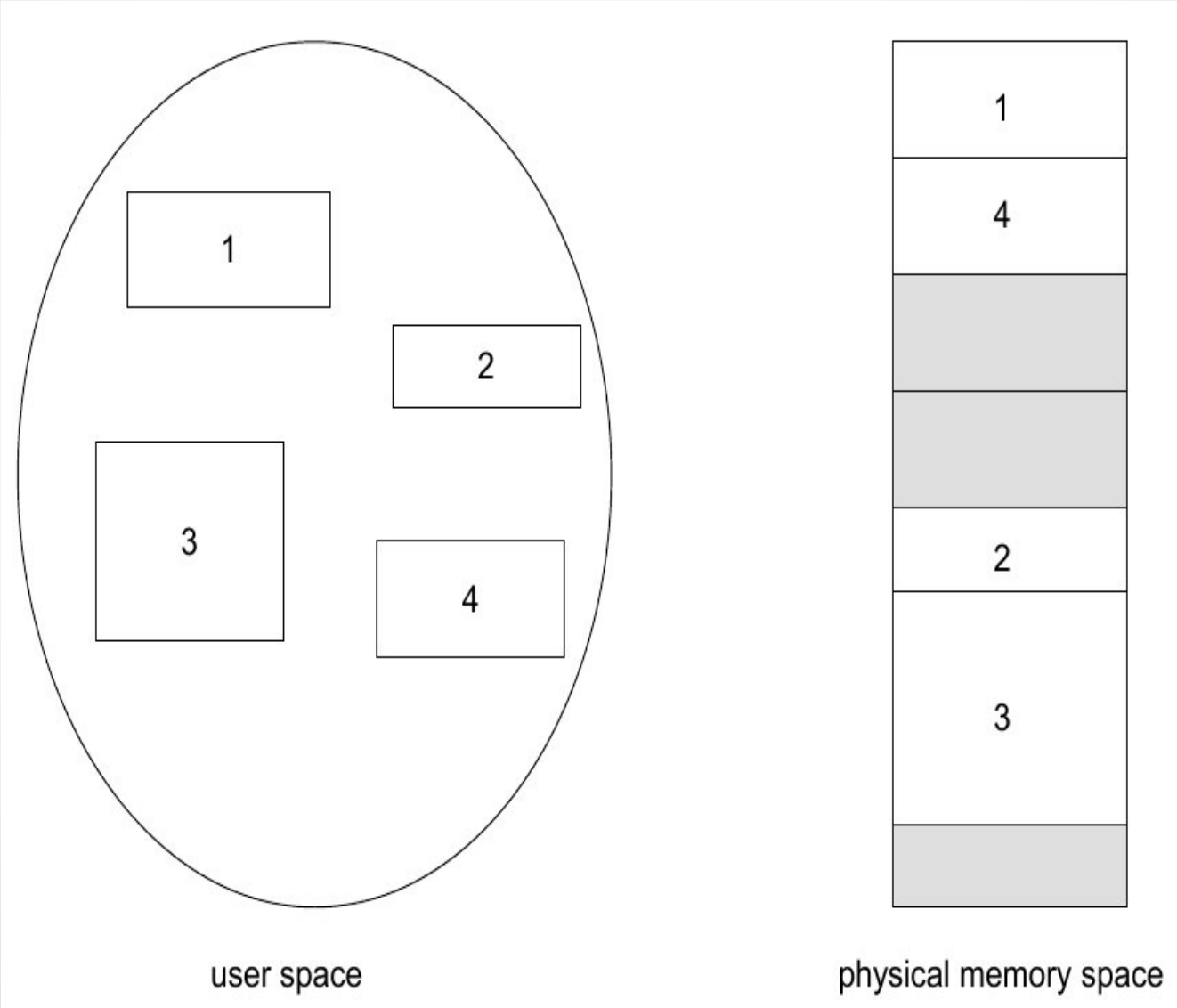
Summary

- This lecture shows the problem of memory management:
  - Introduction
  - Continuous allocation
  - External fragmentation and compactification
  - Memory allocation strategies
  - Segmentation
  - Paging
  - Internal fragmentation
  - Memory management in systems with a lot of memory

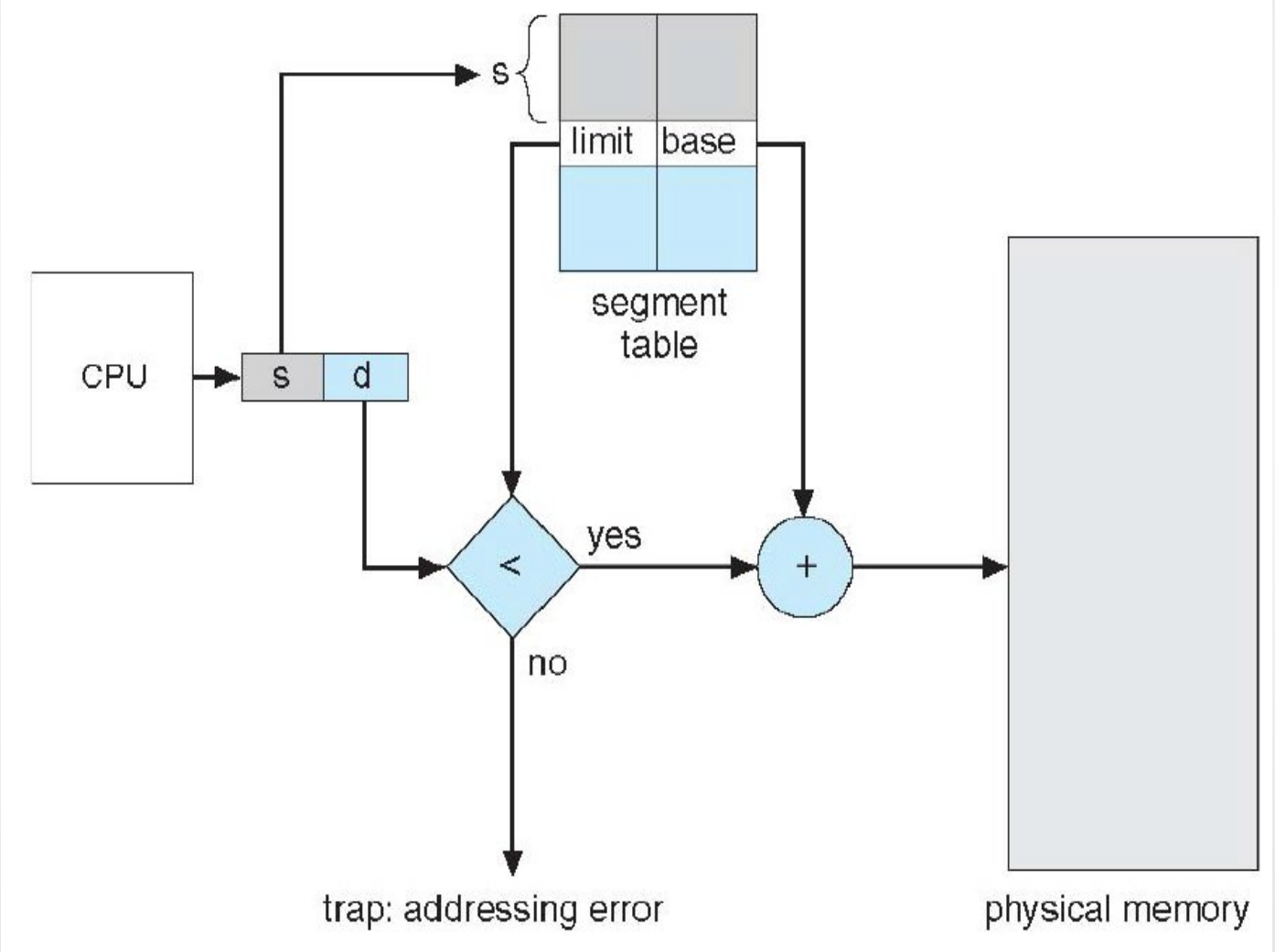
Segmentation

- Memory-management scheme that supports user view of memory
- A program is a collection of segments
  - A segment is a logical unit such as:
    - Main program
    - Local variables, global variables
    - Stack
    - Symbol table
    - Procedure, function, method
    - Object
    - Common block
    - Arrays
    - ...

Logical view of segmentation



Segmentation hardware (MMU)



Segmentation

- The MMU scheme suggests, that memory access is 2x longer than in a standard case (first, access to segments array must be made, then the actual memory call).
  - In order to improve access, we can limit the number of segments and keep proper offsets in registers, thus reducing access time (there is no need to access memory).
  - Switching between processes includes storing segment addresses and reloading them for another process. Segment array may be long, therefore it may be profitable to keep in register only a reference to this array and reload only one particular register.

Protection and segmentation

- Protection
  - With each entry in segment table associate:
    - validation bit = 0 ⇒ illegal segment
    - read/write/execute privileges
- Protection bits associated with segments; code sharing occurs at segment level
- Since segments vary in length, memory allocation is a dynamic storage-allocation problem

Summary

- This lecture shows the problem of memory management:
  - Introduction
  - Continuous allocation
  - External fragmentation and compactification
  - Memory allocation strategies
  - Segmentation
  - Paging
  - Internal fragmentation
  - Memory management in systems with a lot of memory

Paging

- Physical address space of a process can be noncontinuous; process is allocated physical memory whenever the latter is available
  - Avoids external fragmentation
  - Avoids problem of varying sized memory chunks
- Divide physical memory into fixed-sized blocks called frames
  - Size is power of 2, between 512 bytes and 16 Mbytes
- Divide logical memory into blocks of same size called pages
- Keep track of all free frames
- To run a program of size N pages, need to find N free frames and load program
- Set up a **page table** to translate logical to physical addresses
- Backing store likewise split into pages
- Still have Internal fragmentation.

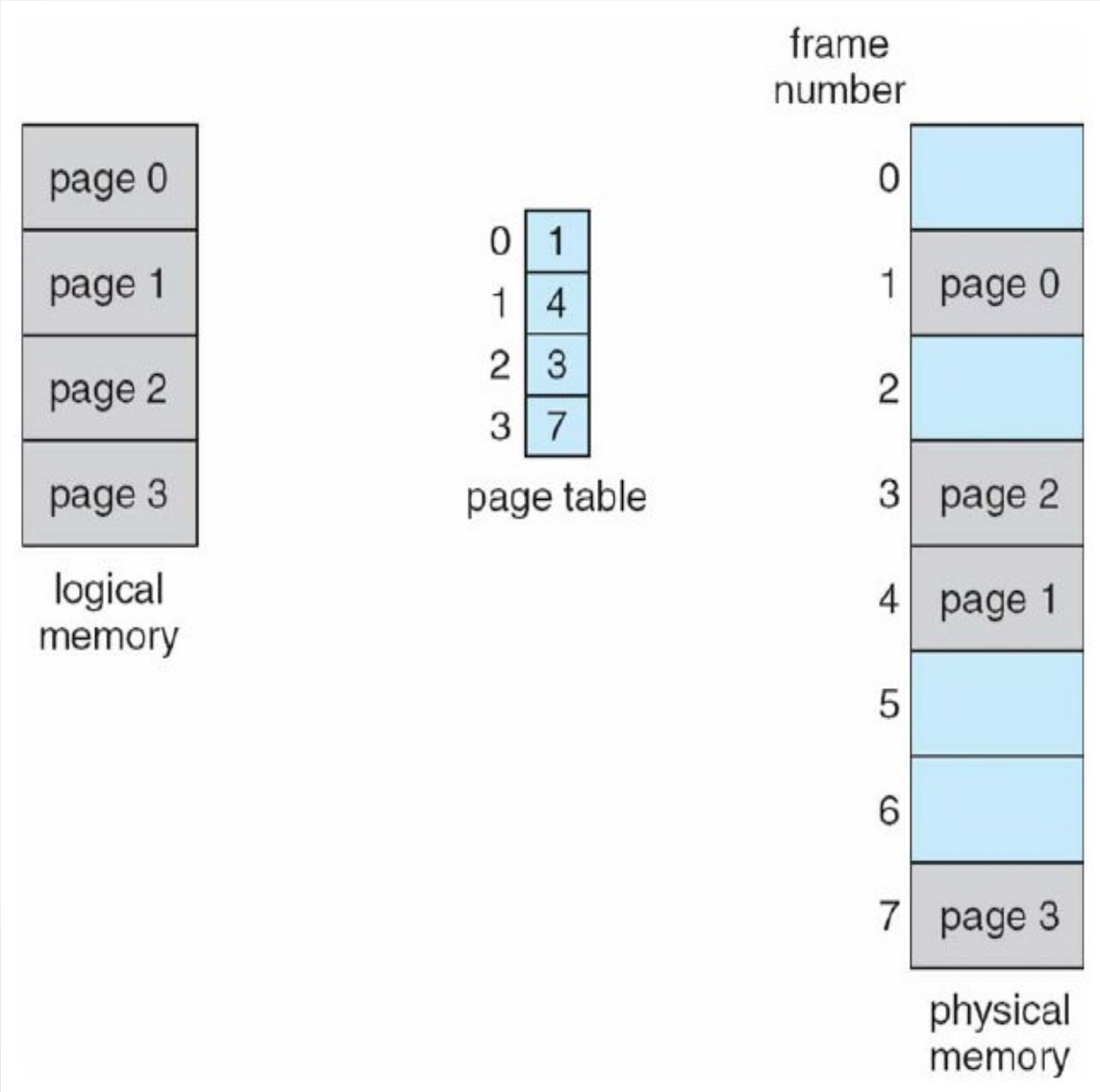
Address translation scheme

- 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 to define the physical memory address that is sent to the memory unit

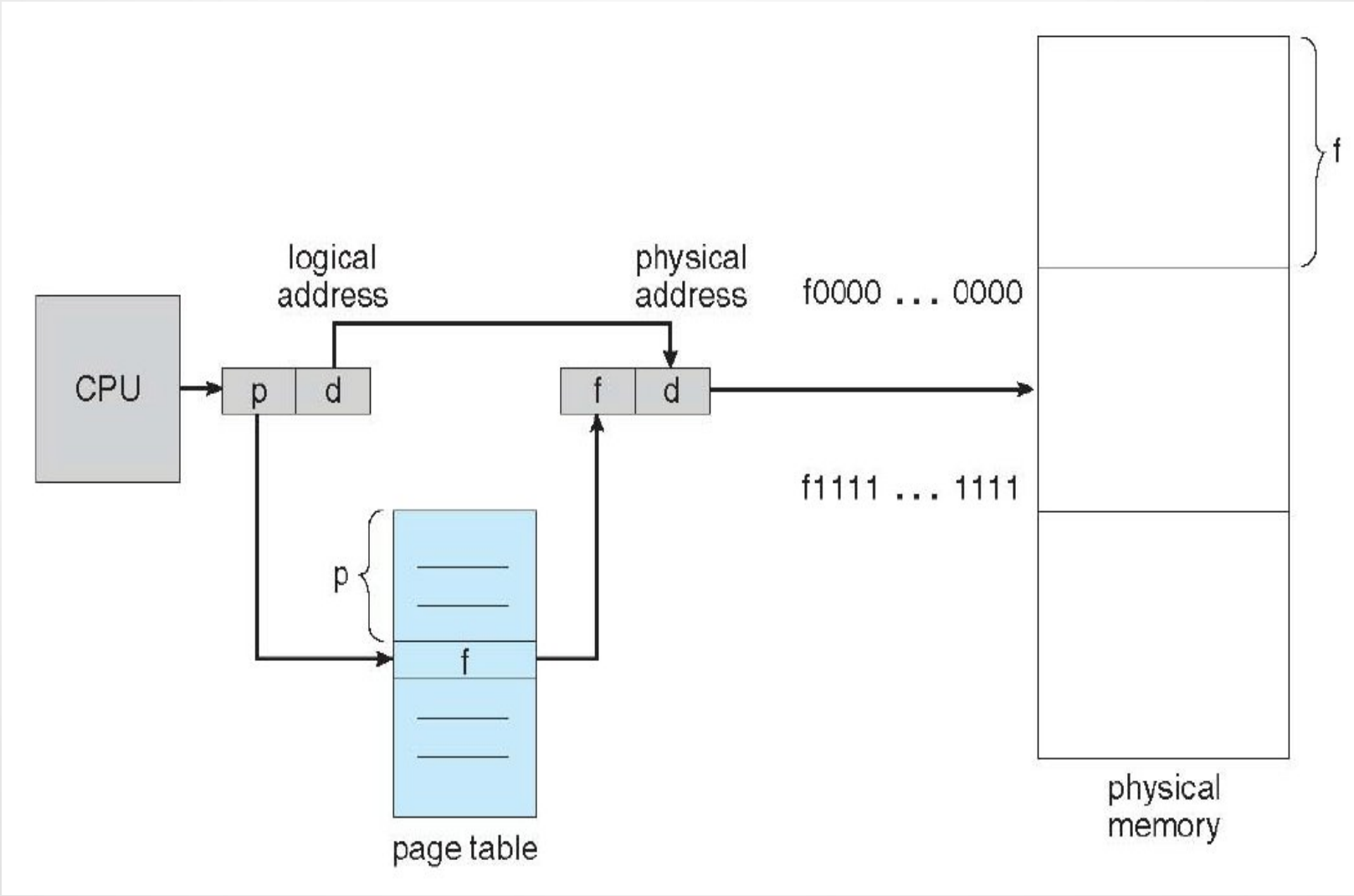
Page number (p) (m-n bits)	Page offset (d) (n bits)

For given logical address space 2^m and page size 2^n

Paging model of logical and physical memory



Paging hardware



Summary

- This lecture shows the problem of memory management:
  - Introduction
  - Continuous allocation
  - External fragmentation and compactification
  - Memory allocation strategies
  - Segmentation
  - Paging
  - Internal fragmentation
  - Memory management in systems with a lot of memory

Internal fragmentation

- Paging gets rid of external fragmentation.
  - No matter where the free pages are located, a system can assign them (allocated pages don't have to be assigned in a sequence).
  - Still, we can observe fragmentation in logical process memory – allocated data structures are usually mapped in such way, that they all reside on the same page (if not too big). It means, that parts of pages can be left unassigned, causing fragmentation.

Summary

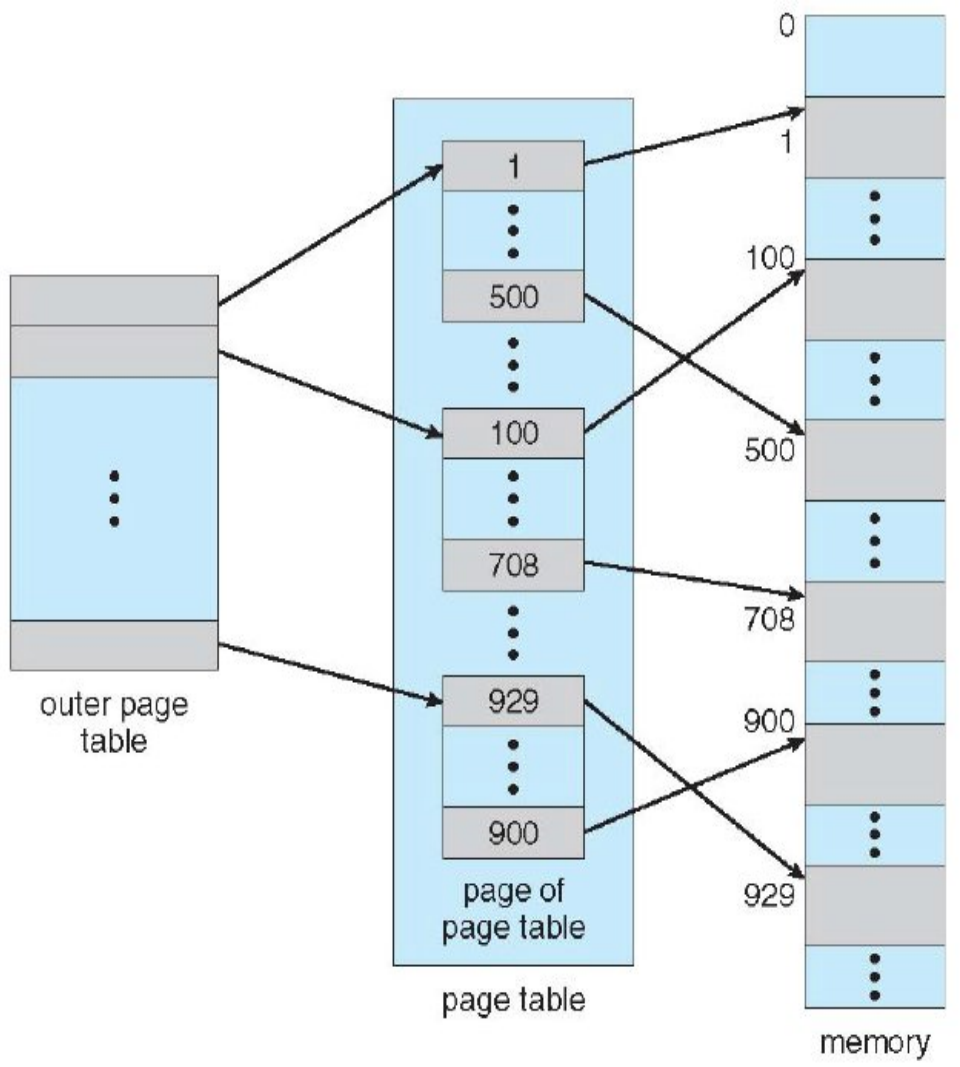
- This lecture shows the problem of memory management:
  - Introduction
  - Continuous allocation
  - External fragmentation and compactification
  - Memory allocation strategies
  - Segmentation
  - Paging
  - Internal fragmentation
  - Memory management in systems with a lot of memory

What to do, when a system can address a lot of memory?

- A system with 64-bit addressing can address up to 16EB (17,179,869,184 GB). A page table for such address space would occupy about 32PB. Therefore, in a system with 64-bit addresses we can use other ways of address translation:
  - Hierarchical page tables
  - Hashed page table
  - Inverted page table

Hierarchical page tables

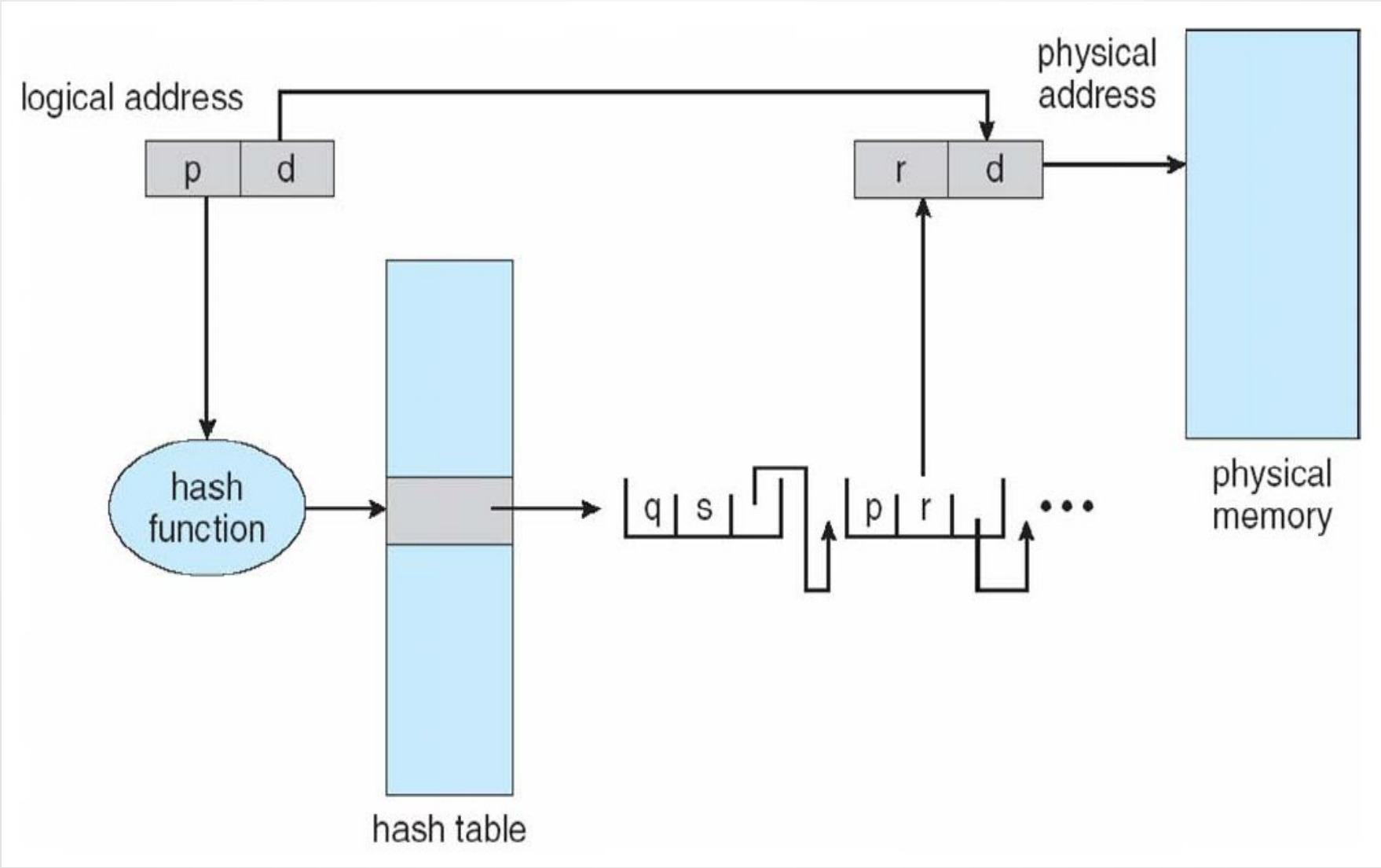
- In a system with 32-bit addressing and 4kB page, the whole page table can occupy up to 4MB of memory. It means, that it cannot fit in a single memory page.
- Therefore, a system can implement a multi-level hierarchy of page tables
  - Address is divided to more segments (3 or more)
  - Each part of address corresponds to consecutive level of arrays in hierarchy



Hashed page table

- Common in 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.
- Each element contains (1) the virtual page number (2) the value of the mapped page frame (3) a pointer to the next element.
- Virtual page numbers are compared in this chain searching for a match:
  - If a match is found, the corresponding physical frame is extracted.
- Variation for 64-bit addresses is clustered page tables:
  - Similar to hashed but each entry refers to several pages (such as 16) rather than 1.
  - Especially useful for sparse address spaces (where memory references are non-continuous and scattered).

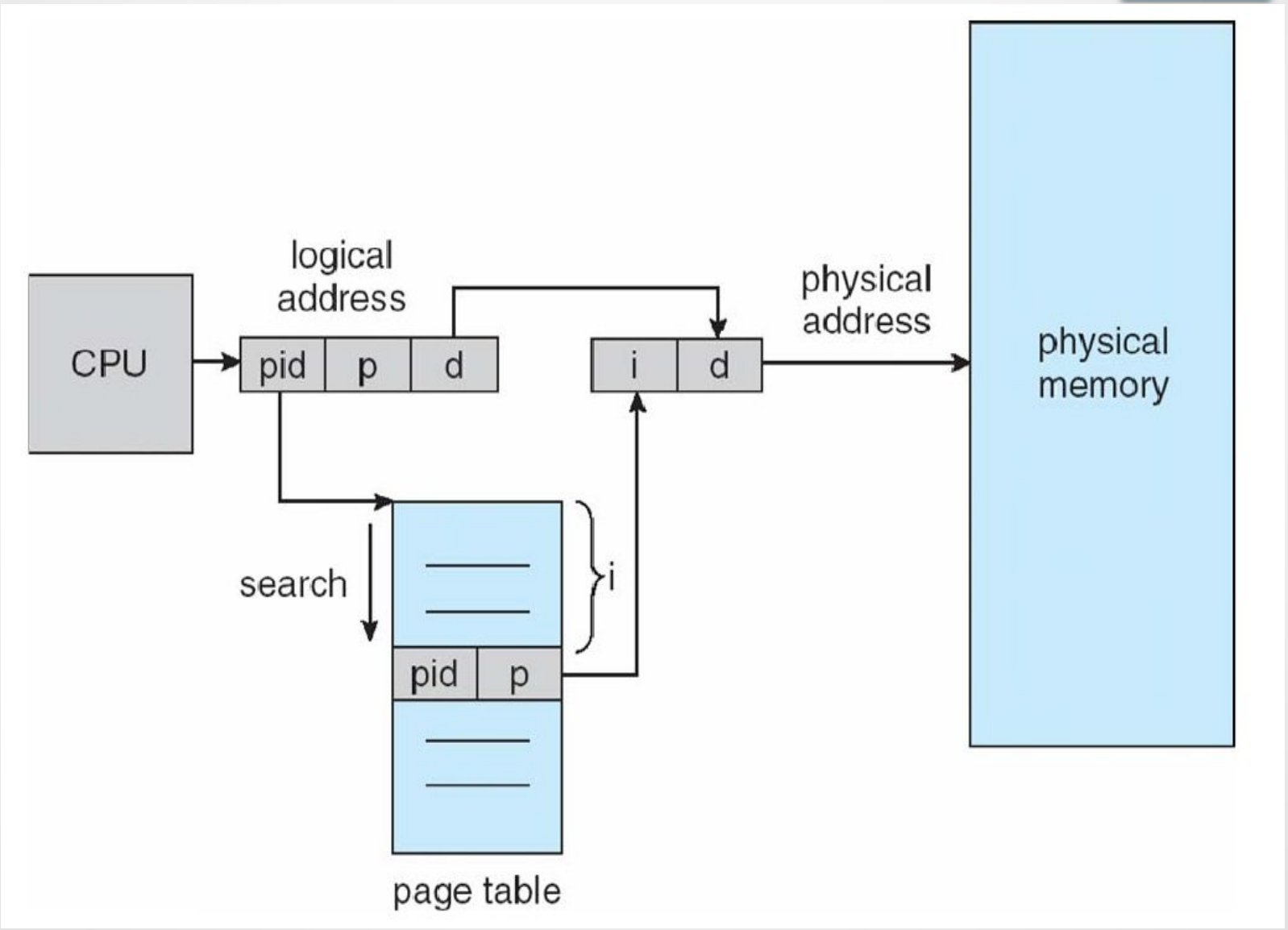
Hashed page table



Inverted page table

- Rather than each process having a page table and keeping track of all possible logical pages, track all physical pages.
- One entry for each real page of memory.
- Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page.
- Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs.
- Use hash table to limit the search to one — or at most a few — page-table entries.
  - TLB can accelerate access.
- But how to implement shared memory?
  - One mapping of a virtual address to the shared physical address.

Inverted page table



Thank You