Unit 5: Memory Management

5.1. Memory Management for Multiprogramming

Roadmap for Section 5.1.

- Memory Management Principles
- Logical vs Physical Address Space
- Swapping vs Segmentation
- Paging

Memory Management Principles

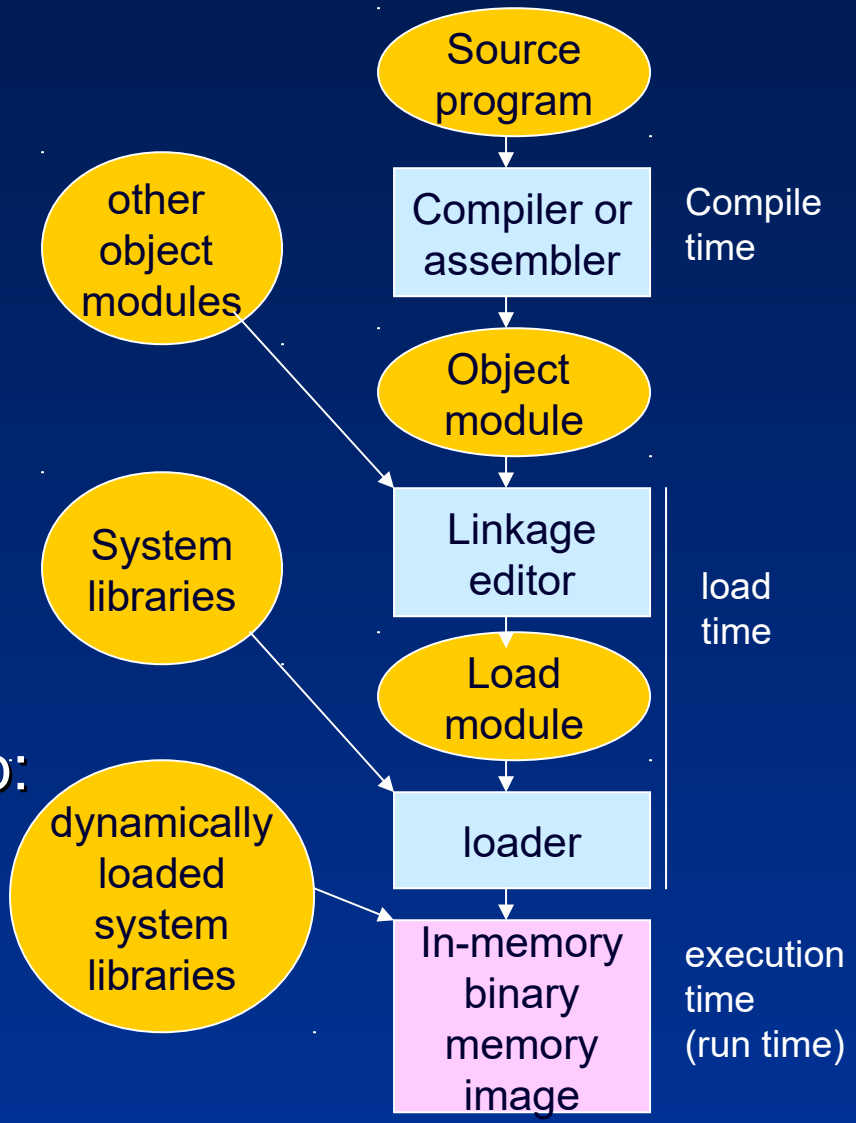
- Memory is central to the operation of a modern computer system
- Memory is a large array of words/bytes
- CPU fetches instructions from memory according to the value of the program counter
- Instructions may cause additional loading from and storing to specific memory addresses

Address Binding

- Addresses in source programs are symbolic
- Compiler binds symbolic to relocatable addresses
- Linkage editor/loader binds relocatable addresses to absolute addresses

Binding can be done at any step:

- i.e., compiler may generate absolute code (as for MS-DOS .COM programs)



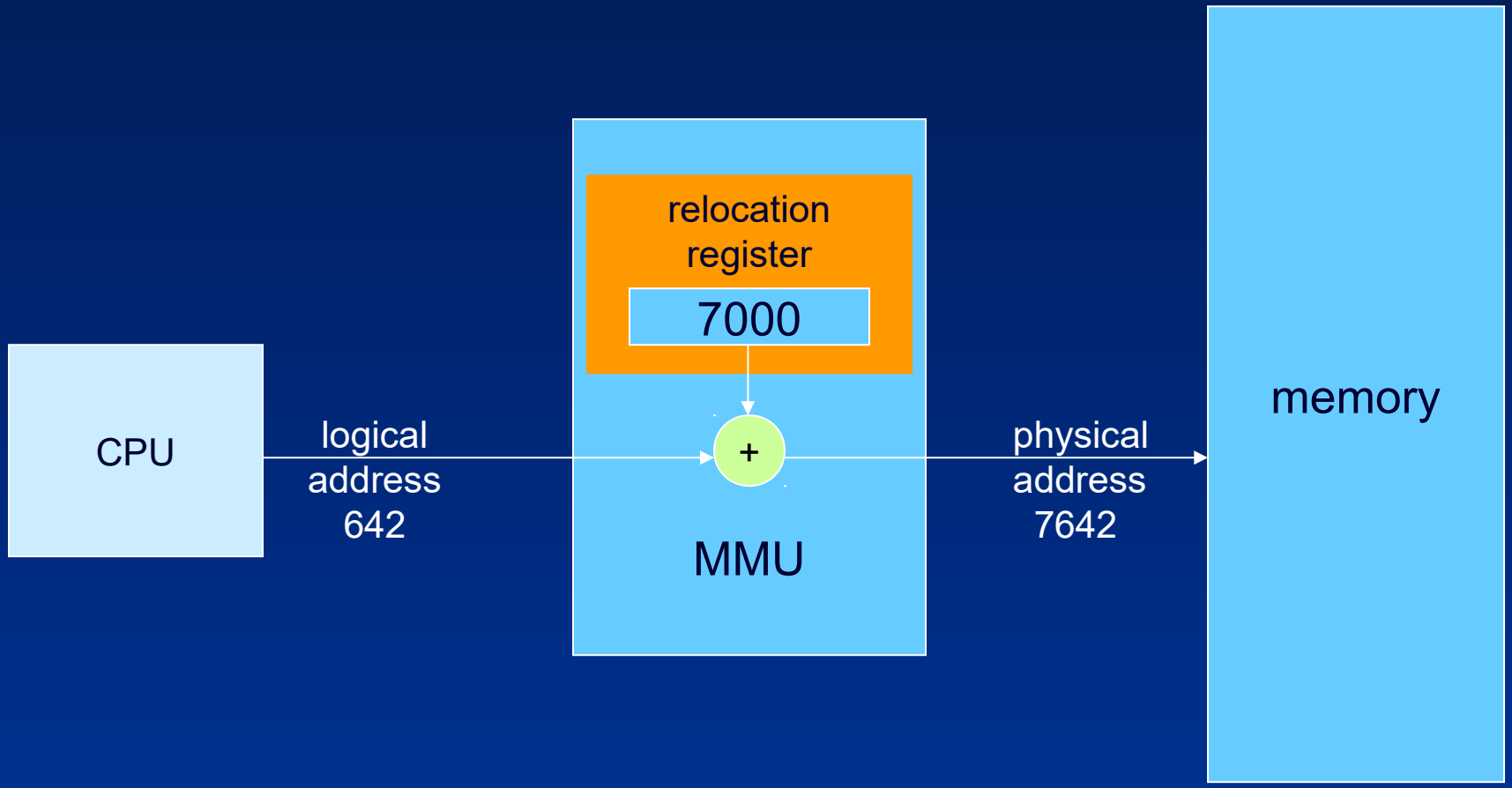
Logical vs. Physical Address Space

- Address generated by CPU is called a *logical address*
- Memory unit deals with *physical addresses*
- **compile-time** and **load-time address-binding**:
  - Logical and physical addresses are identical
- **execution-time address-binding**:
  - Logical addresses are different from physical addresses
  - Logical addresses are also called *virtual addresses*
  - Run-time mapping from virtual to physical addresses is done by *Memory Management Unit* (MMU) – a hardware device
- The concept of a ***logical address space*** that is bound to a different ***physical address space*** is central to Memory Management!

Memory-Management Unit (MMU)

- Hardware device that maps virtual to physical address.
  - The MMU is part of the processor
  - Re-programming the MMU is a privileged operation that can only be performed in privileged (kernel) mode
- In MMU scheme, the value in the relocation register is added to every address generated by a user process at the time it is sent to memory.
  - The user program deals with *logical* addresses; it never sees the *real* physical addresses.

Dynamic relocation using a relocation register



Dynamic Loading

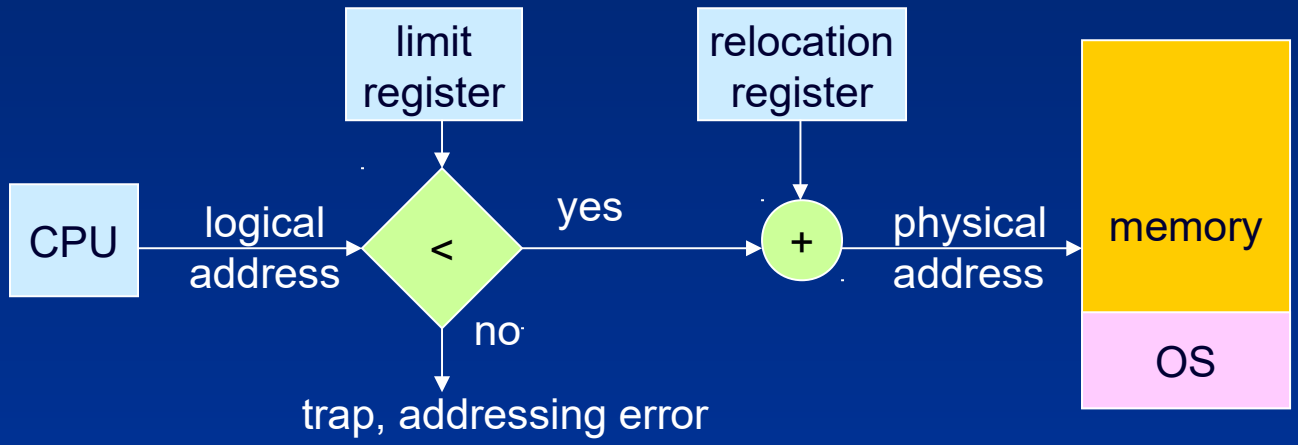
- A routine is not loaded until it is called
- All routines are kept on disk in a relocatable load format
- When a routine calls another routine:
  - It checks, whether the other routine has been loaded
  - If not, it calls the relocatable linking loader to load desired routine
  - Loader updates program‘s address tables to reflect change
  - Control is passed to newly loaded routine
- Better memory-space utilization
  - Unused routines are never loaded
- No special OS support required

Dynamic Linking

- Similar to dynamic loading:
  - Rather than loading being postponed until run time, linking is postponed
  - Dynamic libraries are not statically attached to a program‘s object modules (only a small stub is attached)
  - The stub indicates how to call (load) the appropriate library routine
- All programs may use the same copy of a library (code) (shared libraries - .DLLs)
- Dynamic linking requires operating system support
  - OS is the only instance which may locate a library in another process‘s address space

Memory Allocation Schemes

- Main memory must accommodate OS + user processes
  - OS needs to be protected from changes by user processes
  - User processes must be protected from each other
- Single partition allocation:
  - User processes occupy a single memory partition
  - Protection can be implemented by limit and relocation register (OS in low memory, user processes in high memory, see below)



Memory Allocation Schemes (contd.)

- Multiple-Partition Allocation
  - Multiple processes should reside in memory simultaneously
  - Memory can be divided in multiple partitions (fixed vs. variable size)
    Problem: **What is the optimal partition size**?
- Dynamic storage allocation problem
  - Multiple partitions with holes in between
  - Memory requests are satisfied from the **set of holes**
- Which hole to select?
  - **First-fit**: allocate the *first* hole that is big enough
  - **Best-fit**: allocate the *smallest* hole that is big enough
  - **Worst-fit**: allocate the *largest* hole (produces largest leftover hole)
  - First-fit & best-fit are better than worst-fit (time & storage-wise)
  - First-fit is generally faster than best-fit

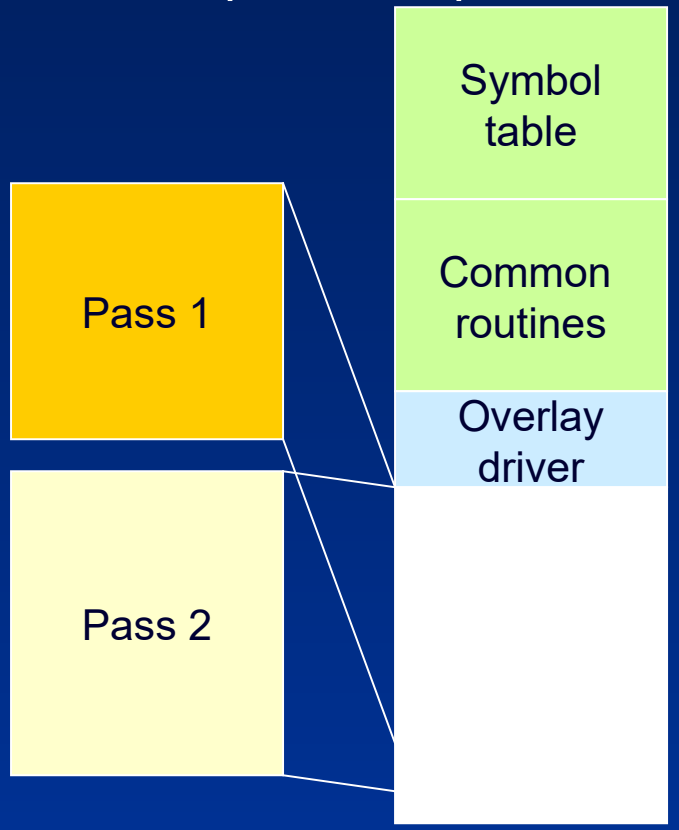
Overlays

- Size of program and data may exceed size of memory

Concept:

- Separate program in modules
- Load modules alternatively
- Overlay driver locates modules on disk
- Overlay modules are kept as absolute memory images
- Compiler support required

Example:
multi-pass compiler



Swapping

In a multiprogramming environment:

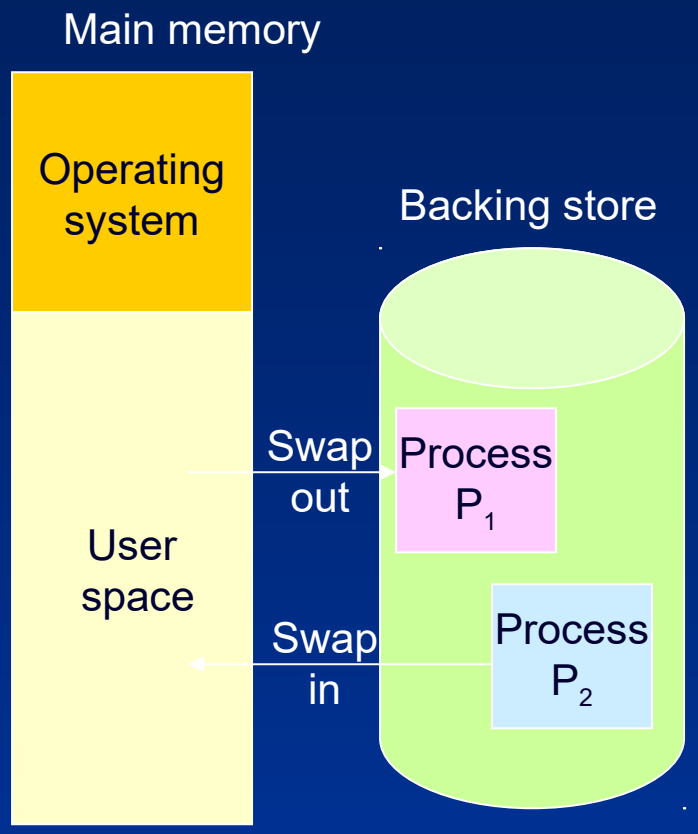
- Processes can temporarily be ***swapped out*** of memory to backing store in order to allow for execution of other processes

On the basis of physical addresses:

- Then, processes will be ***swapped in*** into same memory space that they occupied previously

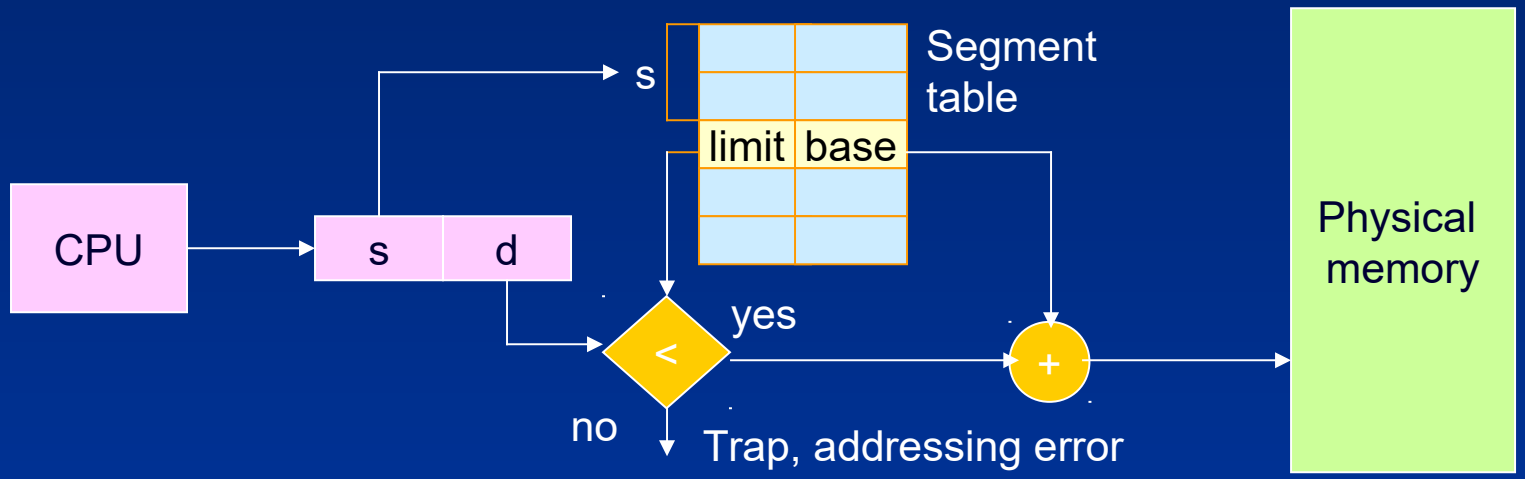
On the basis of logical addresses:

- What current OSes call swapping is rather paging out whole processes.
- Then, processes can be ***swapped in*** at arbitrary physical addresses.



Segmentation

- What is the programmer‘s view of memory?
  - Collection of variable-sized segments (text, data, stack, subroutines,..)
  - No necessary ordering among segments
  - Logical address: <segment-number, offset>
- Hardware:
  - Segment table containing base address and limit for each segment



Fragmentation

- **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous.
- **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used.
- Reduce external fragmentation by compaction
  - Shuffle memory contents to place all free memory together in one large block.
  - Compaction is possible *only* if relocation is dynamic, and is done at execution time.
  - I/O problem
    - Latch job in memory while it is involved in I/O.
    - Do I/O only into OS buffers.

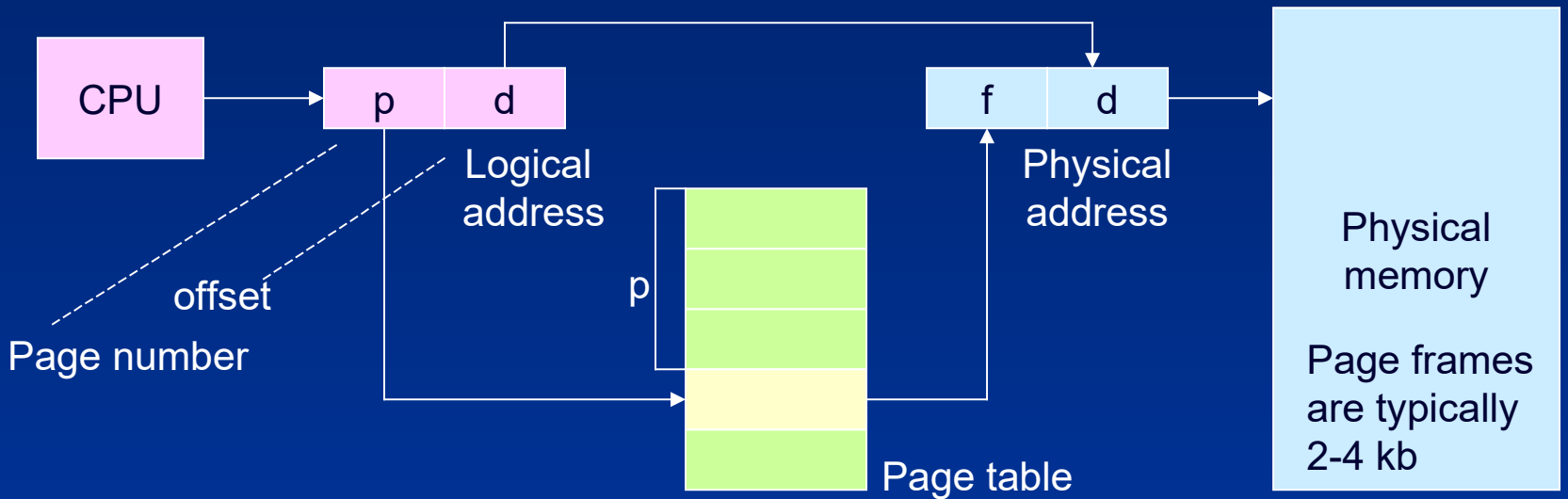
Paging

- Dynamic storage allocation algorithms for varying-sized chunks of memory may lead to fragmentation
- Solutions:
  - Compaction – dynamic relocation of processes
  - Noncontiguous allocation of process memory in equally sized **pages** (this avoids the memory fitting problem)

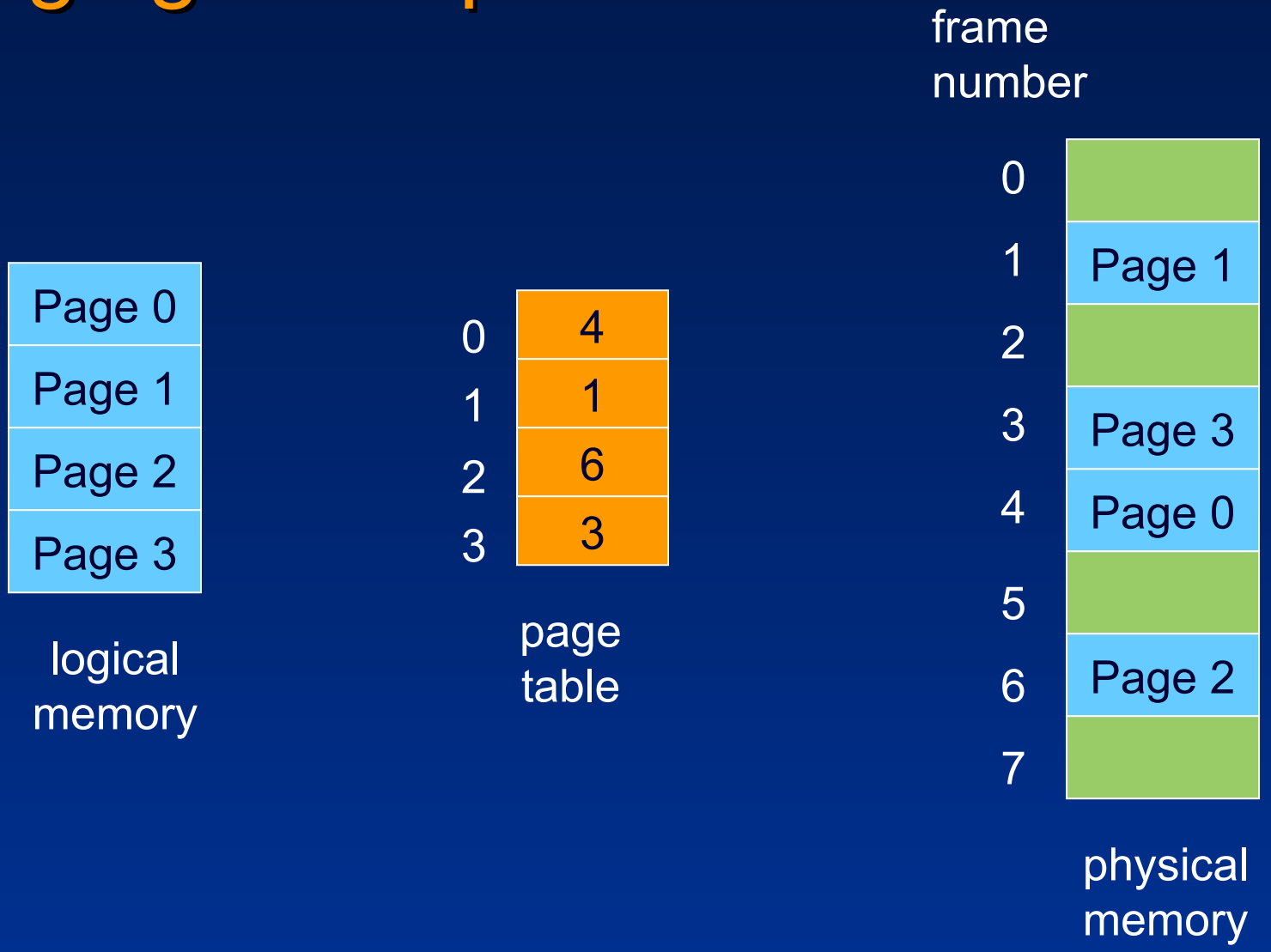
- **Paging** breaks physical memory into fixed-sized blocks (called *frames*)
- Logical memory is broken into *pages* (of the same size)

Paging: Basic Method

- When a process is executed, its pages are loaded into any available frames from backing store (disk)
- Hardware support for paging consists of a page table
- Logical addresses consist of page number and offset



Paging Example



Free Frames

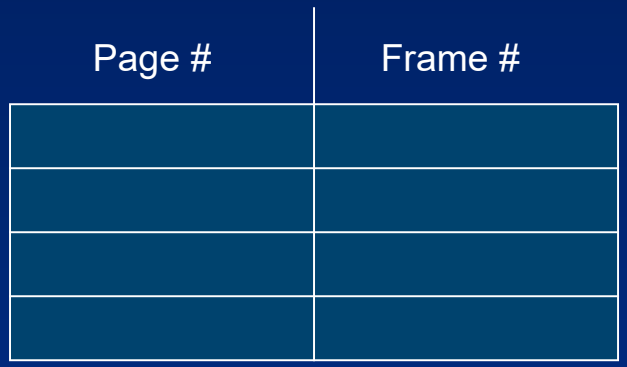

7
8
9
10
11
12
13
14
15
16
17
Free frame list
7, 8, 10, 11,13, 16
frame number
Process creation
0
1
2
3
11
8
16
13
New process page table
Free frame list
7, 10
7
8
9
10
11
12
13
14
15
16
17
Page 1
Page 0
Page 3
Page 2
Before allocation
After allocation
physical memory

Paging: Hardware Support

- Every memory access requires access to page table
  - Page table should be implemented in hardware
  - Page tables exist on a per-user process basis
- Small page tables can be just a set of registers
  - Problem: size of physical memory, # of processes
- Page tables should be kept in memory
  - Only base address of page table is kept in a special register
  - Problem: speed of memory accesses
- Translation look-aside buffers (TLBs)
  - Associative registers store recently used page table entries
  - TLBs are fast, expensive, small: 8..2048 entries
  - TLB must be flushed on process context switches

Associative Memory

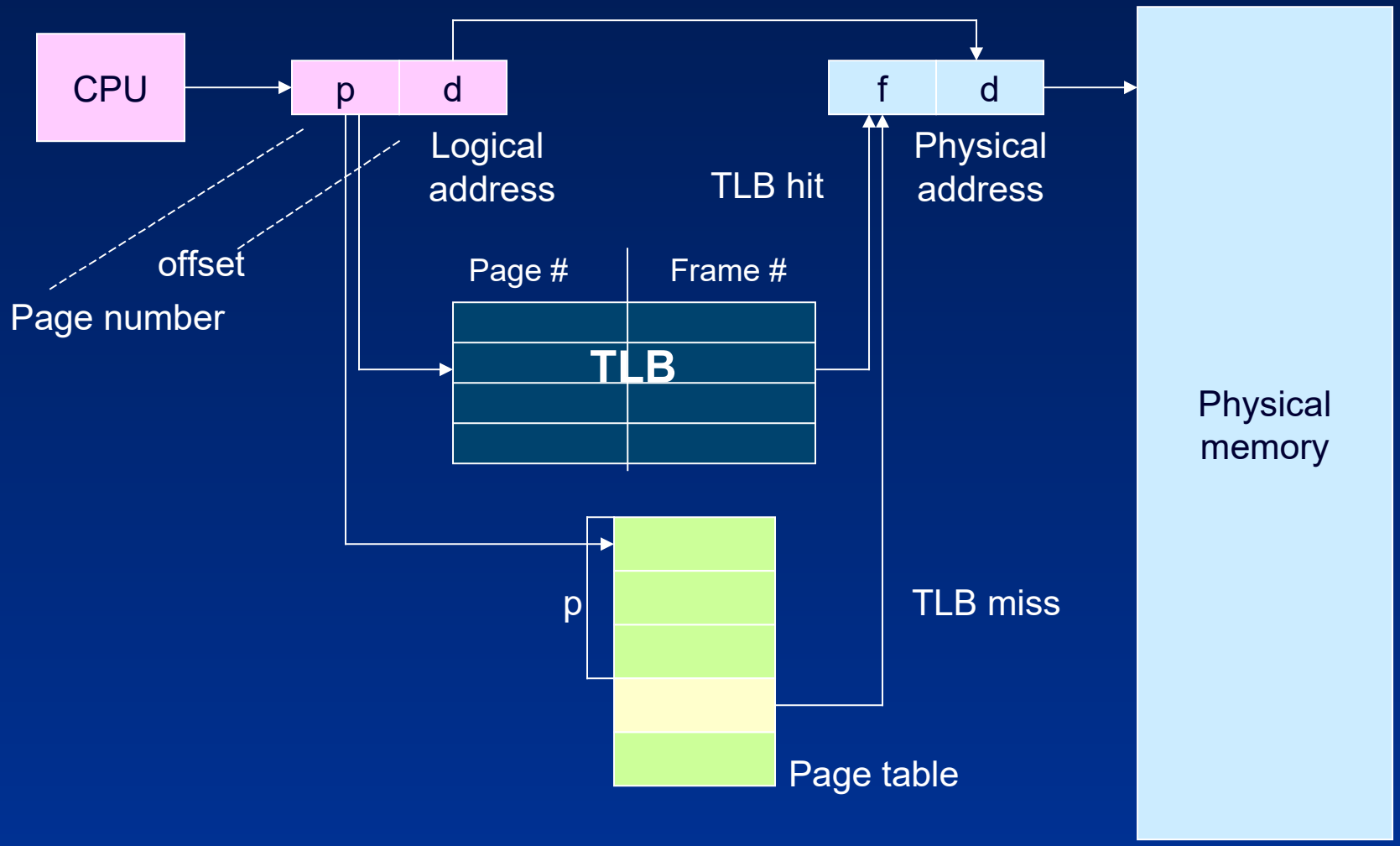
- Associative memory – parallel search



Address translation (A´, A´´)

- If A´ is in associative register, get frame # out.
- Otherwise get frame # from page table in memory

Paging Hardware With TLB



Effective Access Time with TLB

- Associative Lookup in TLB = ε time unit
- Assume memory cycle time is 1 microsecond
- Hit ratio – percentage of times that a page number is found in the associative registers;
  - ratio related to number of associative registers.
  - Let us assume a hit ratio = α

- Effective Access Time (EAT)

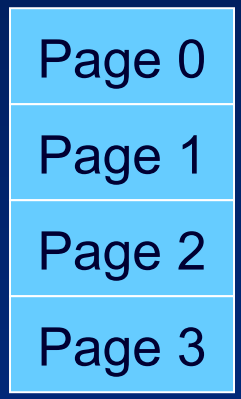
  $EAT = (1 + \varepsilon)\,\alpha + (2 + \varepsilon)(1 - \alpha)$

  $= 2 + \varepsilon - \alpha$

Memory Protection

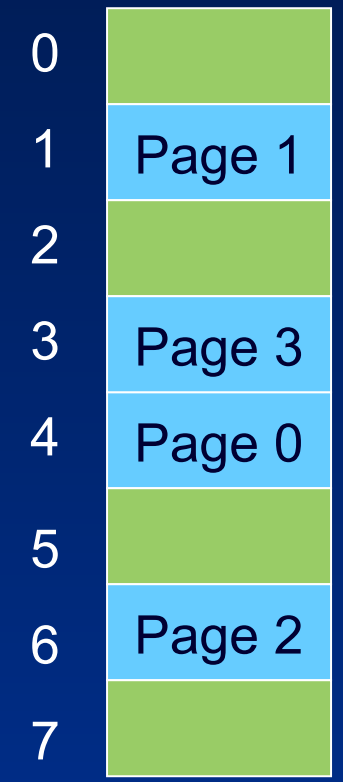
- Memory protection implemented by associating control bits with each frame
  - Isolation of processes in main memory
- *Valid-invalid* bit attached to each entry in the page table:
  - “valid” indicates that the associated page is in the process’ logical address space, and is thus a legal page
  - “invalid” indicates that the page is not in the process’ logical address space

Valid (v) or Invalid (i) Bit in a Page Table



logical memory

page table



physical memory

- Invalid pages may be paged out

Page Table Structure

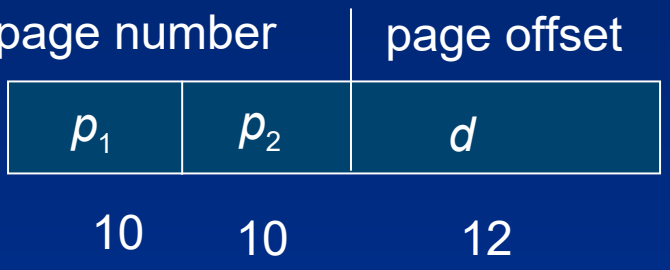
- Hierarchical Paging
- Hashed Page Tables
- Inverted Page Tables

Hierarchical Page Tables

- Break up the logical address space into multiple page tables
- A simple technique is a two-level page table
  - Used with 32-bit CPUs

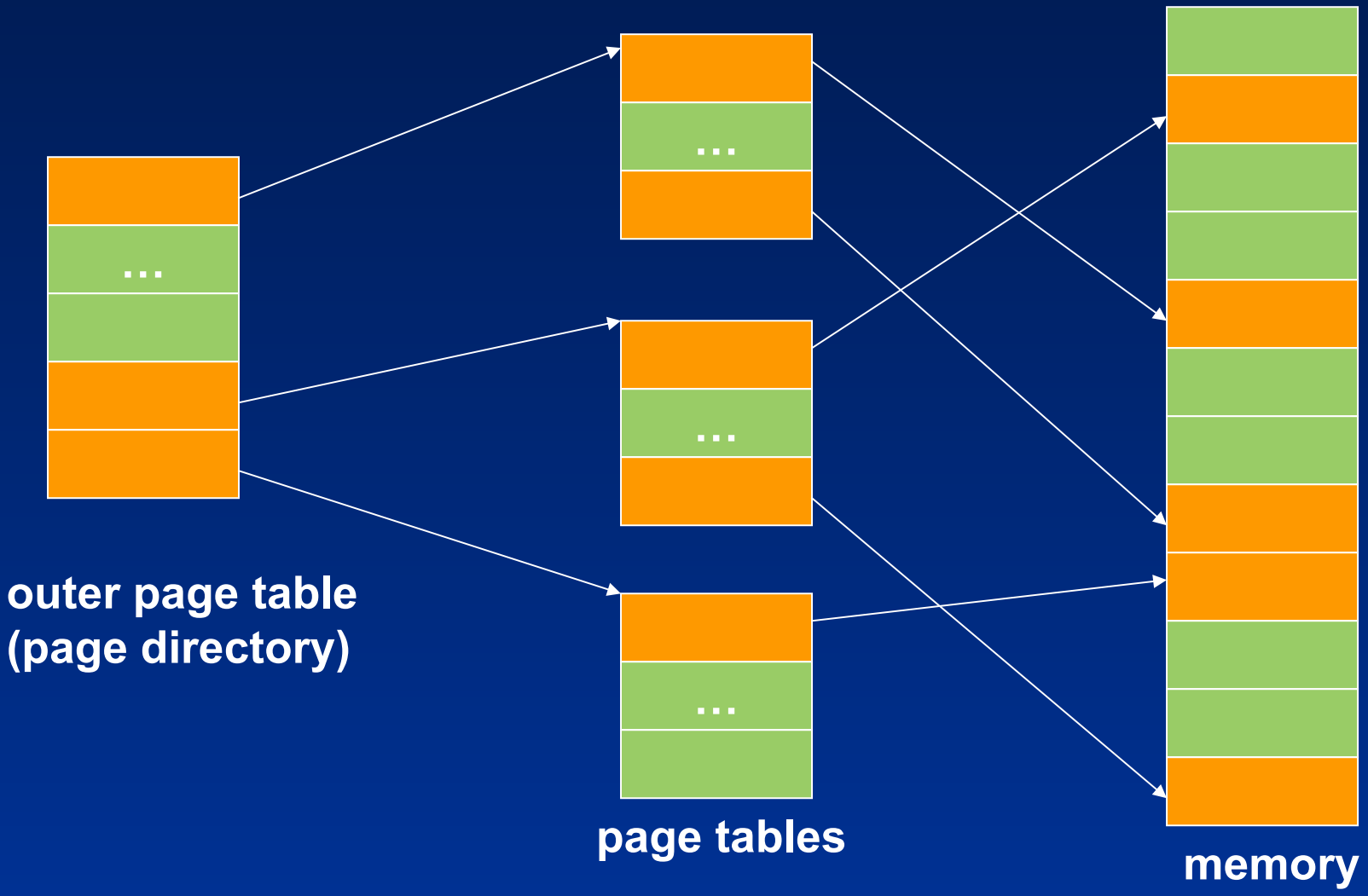
Two-Level Paging Example

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



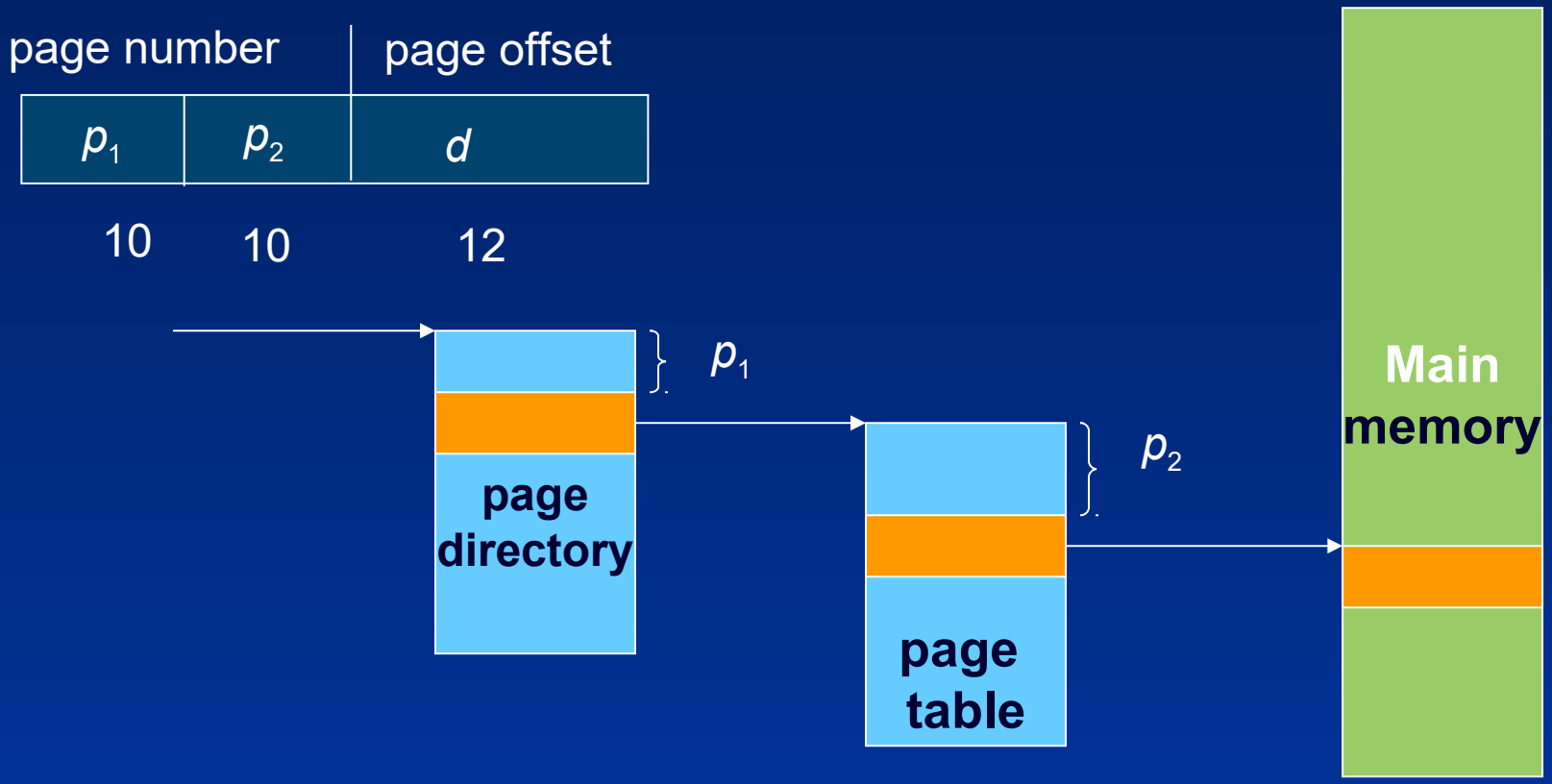
where p_1 is an index into the outer page table, and p_2 is the displacement within the page of the outer page table

Two-Level Page-Table Scheme



Address-Translation Scheme

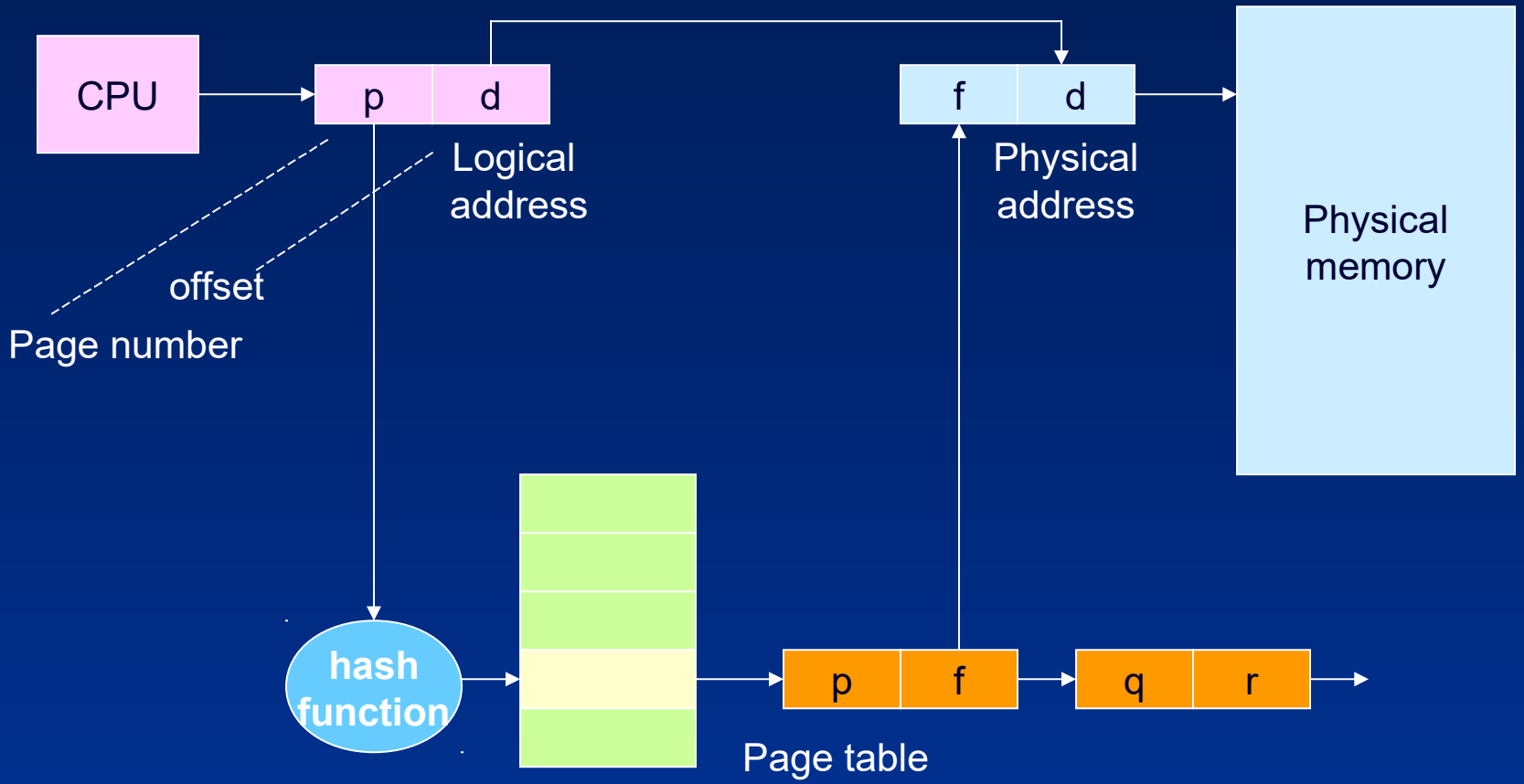
- Address-translation scheme for a two-level 32-bit paging architecture



Hashed Page Tables

- Common in address spaces > 32 bits
  - IA64 supports hashed page tables
- 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

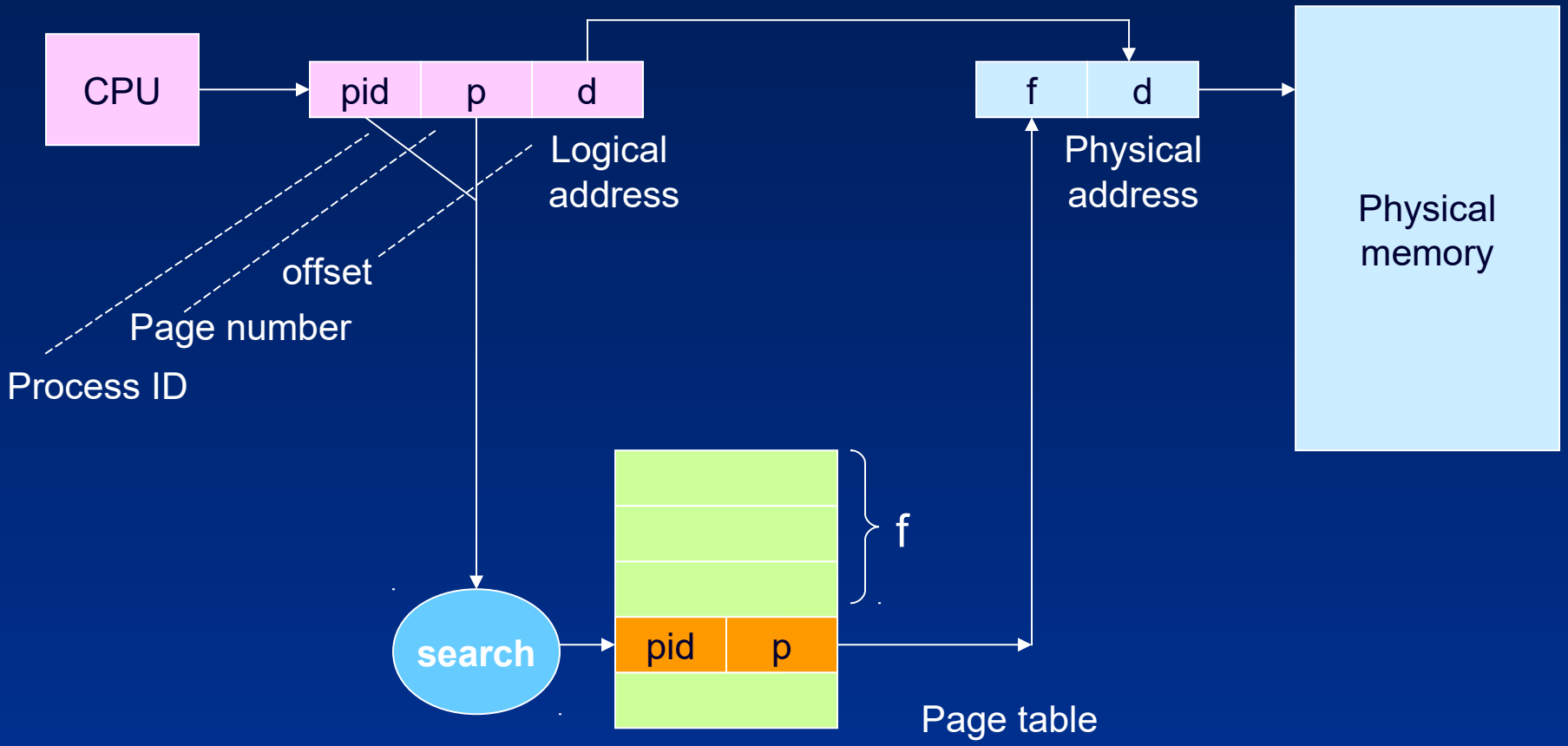
Hashed Page Table



Inverted Page Table

- 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

Inverted Page Table Architecture



Shared Pages

- Shared code
  - One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems)
  - Shared code must appear in same location in the logical address space of all processes
- 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

Process 1 virtual memory

cpp
cc1
cc2
data1

Process 1 page table

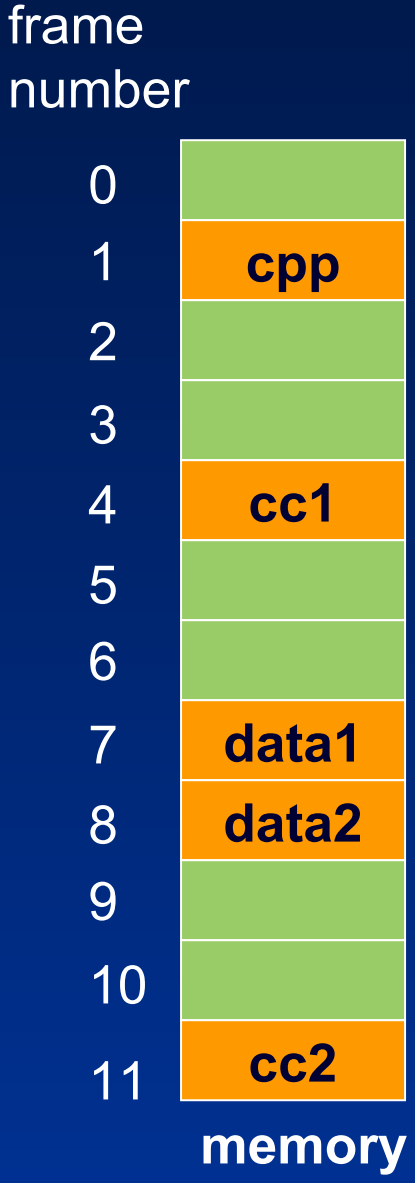
1
4
11
7

Process 2 virtual memory

cpp
cc1
cc2
data2

Process 2 page table

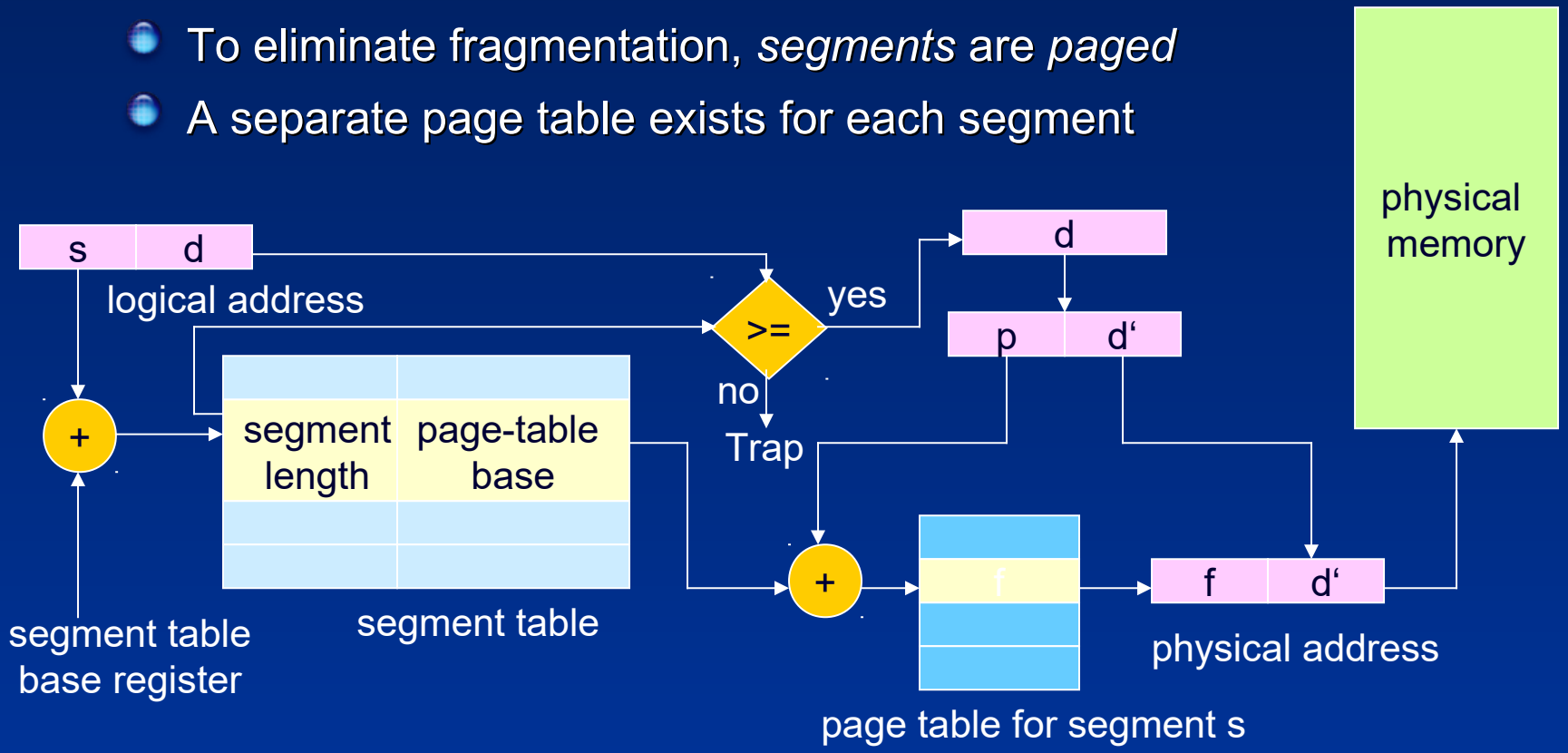
1
4
11
8



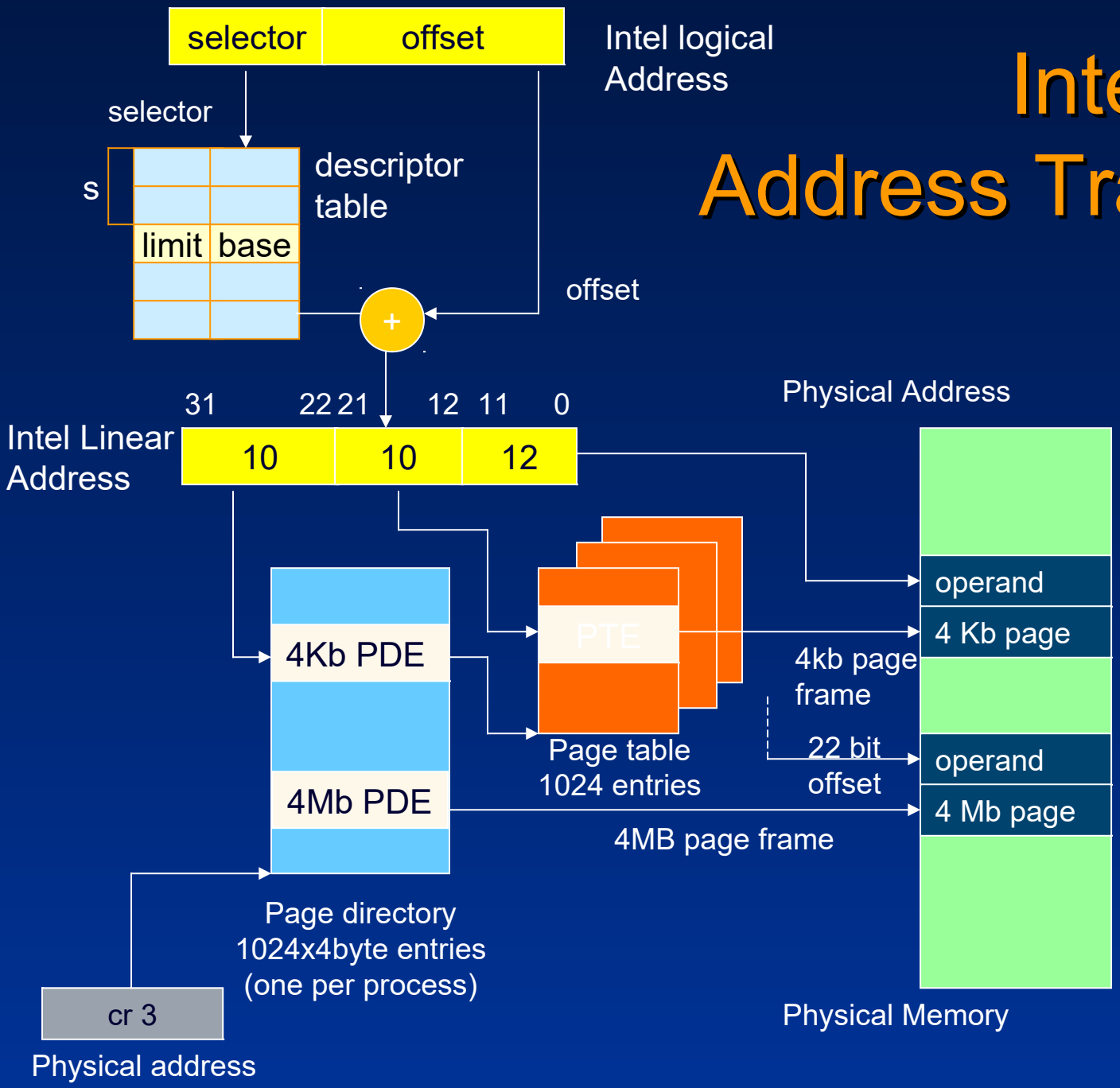
Segmentation with Paging
- paged segmentation on the GE 645 (Multics)

The innovative MULTICS operating system introduced:

- Logical addresses: 18-bit segment number, 16-bit offset
- (relatively) small number of 64k segments
- To eliminate fragmentation, *segments* are *paged*
- A separate page table exists for each segment



Intel 80386 Address Translation



The Intel 386 uses segmentation with paging for memory management, with a two-level paging scheme.

Summary

- In a multiprogrammed OS, every memory address generated by the CPU must be checked for legality and possibly mapped to a physical address
  - Checking cannot be implemented (efficiently) in software
  - Hardware support is essential
- A pair of registers is sufficient for single/multiple partition schemes
  - Paging/segmentation need mapping tables to define address maps
- Paging and segmentation can be fast
  - Tables have to be implemented in fast registers (Problem: size)
  - Set of associative registers (TLB) may reduce performance degradation if tables are kept in memory
- Most modern OS combine paging and segmentation

Further Reading

- Abraham Silberschatz, Peter B. Galvin, and Greg Gagne, "*Operating System Concepts*", John Wiley & Sons, 9th Ed., 2013.
  - Chapter 8 – Main Memory
  - Chapter 9 – Virtual Memory