SNS COLLEGE OF TECHNOLOGY

Coimbatore-35
An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A++' Grade

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

19ECT312 – EMBEDDED SYSTEM DESIGN

III YEAR/ VI SEMESTER

UNIT 4 : EMBEDDED OPERATING SYSTEM AND MODELING

TOPIC 4.3: MEMORY MANAGEMENT

Memory management

- We have seen how CPU can be shared by a set of processes
  - Improve system performance
  - Process management
- Need to keep several process in memory
  - Share memory
- Learn various techniques to manage memory
  - Hardware dependent

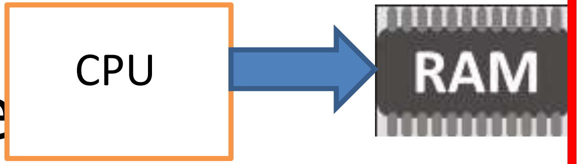
Memory management

What are we going to learn?

- ***Basic Memory Management:*** logical vs. physical address space, protection, contiguous memory allocation, paging, segmentation, segmentation with paging.
- ***Virtual Memory:*** background, demand paging, performance, page replacement, page replacement algorithms (FCFS, LRU), allocation of frames, thrashing.

Background

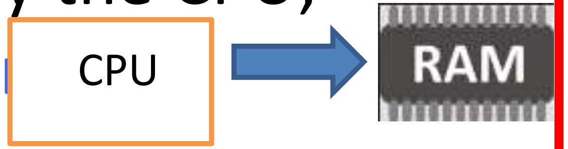
- Program must be brought (from disk) into memory
- Fetch-decode-execute cycle



- Memory unit only sees a stream of addresses + read requests, or address + data and write requests
- Sequence of memory addresses generated by running program

Logical vs. Physical Address Sp

Logical address – generated by the CPU; also referred to as **virtual addr**



Physical address – address seen by the memory unit

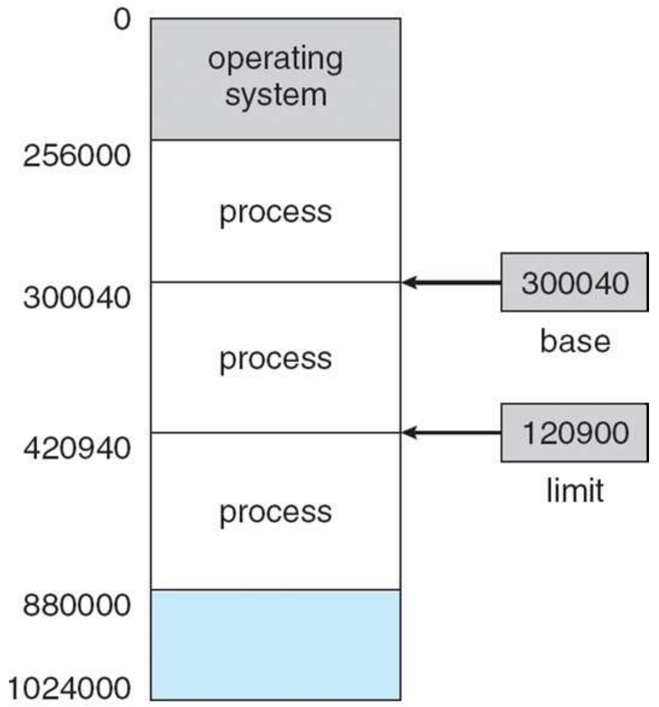
- **Logical address space** is the set of all logical addresses generated by a program
- **Physical address space** is the set of all physical addresses generated by a program

Background

Multiple processes resides in memory

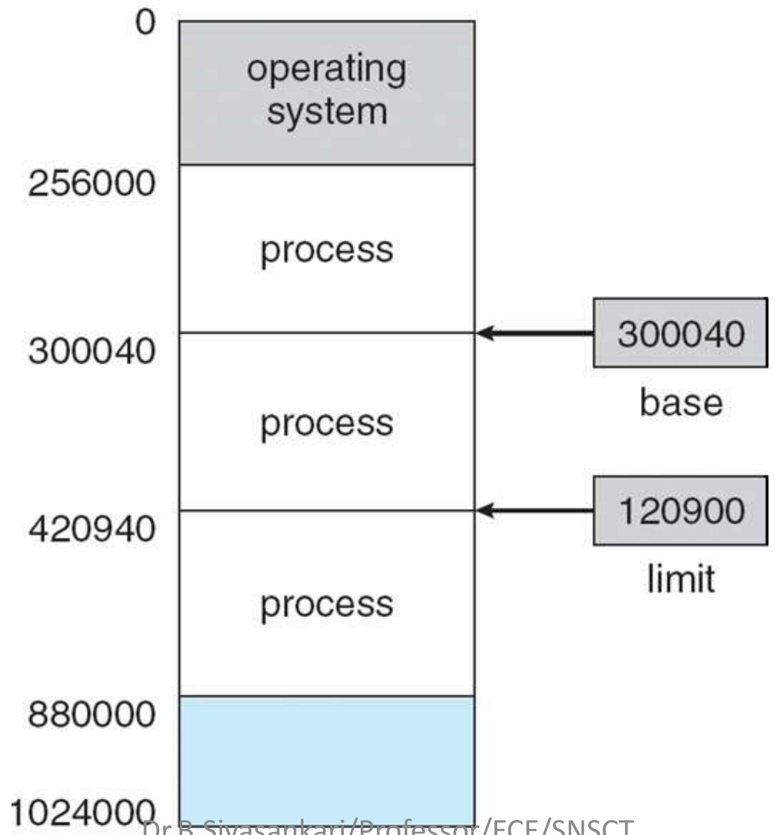
- Protection of memory required to ensure correct operation

Protect OS
Protect user processes

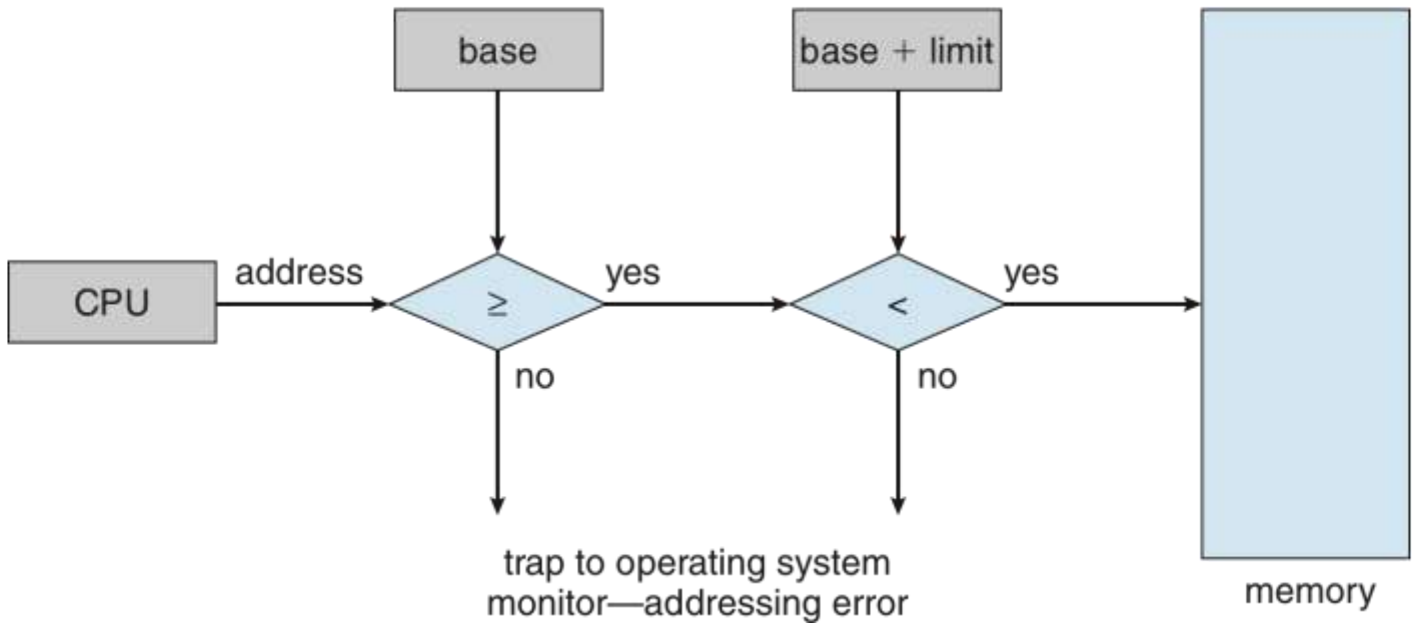


Base and Limit Registers

- A pair of **base** and **limit** registers define the logical address space



HARDWARE ADDRESS PROTECTION WITH BASE AND LIMIT REGISTERS

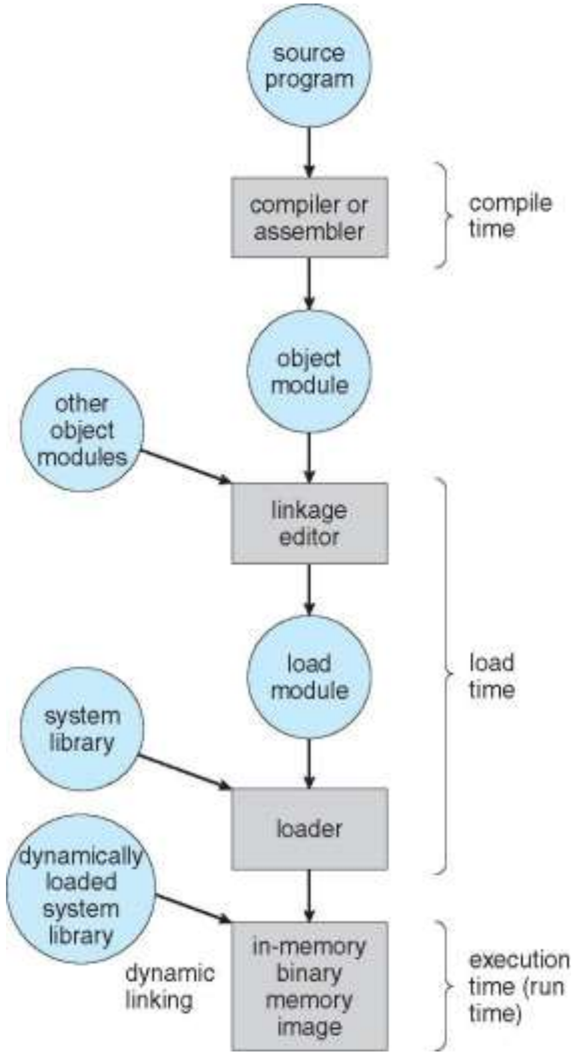


- OS loads the base & limit reg.
- Privileged instruction

Address Binding

- Process resides in main memory
- Associate each data element with memory address
- Further, addresses represented in different ways at different stages of a program's life
  - Source code addresses usually symbolic
  - Compiled code addresses **bind** to relocatable addresses
    - i.e. "14 bytes from beginning of this module"
  - Linker or loader will bind relocatable addresses to absolute addresses
    - i.e. 74014

Multistep Processing of a User Program

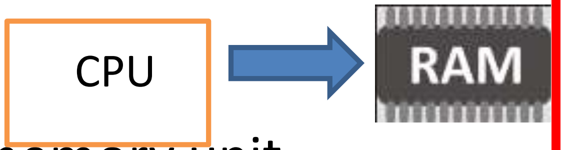


Binding of Instructions and Data to Memory

- Address binding of instructions and data to memory addresses can happen at three different stages
  - **Compile time**: If memory location known a priori, **absolute code** can be generated; must recompile code if starting location changes
  - **Load time**: Must generate **relocatable code** if memory location is not known at compile time
  - **Execution time**: If the process can be moved during its execution from one memory segment to another
    - Binding delayed until run time
    - Need hardware support for address maps (e.g., base and limit registers)

Logical vs. Physical Address Space

Logical address – generated by the CPU; also referred to as **virtual address**



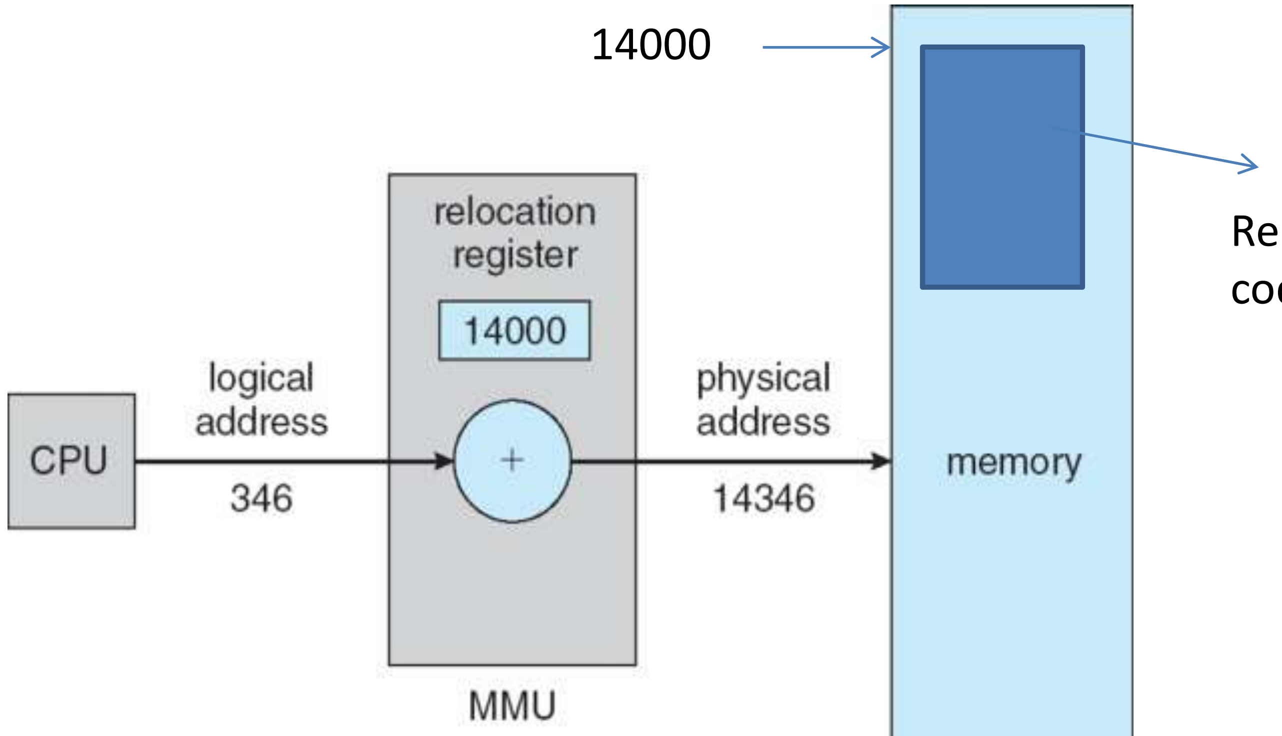
Physical address – address seen by the memory unit

- Logical and physical addresses are the same in compile-time and load-time address-binding schemes;
- logical (virtual) and physical addresses differ in execution-time address-binding scheme
- **Logical address space** is the set of all logical addresses generated by a program
- **Physical address space** is the set of all physical addresses generated by a program

Memory-Management Unit (MMU)

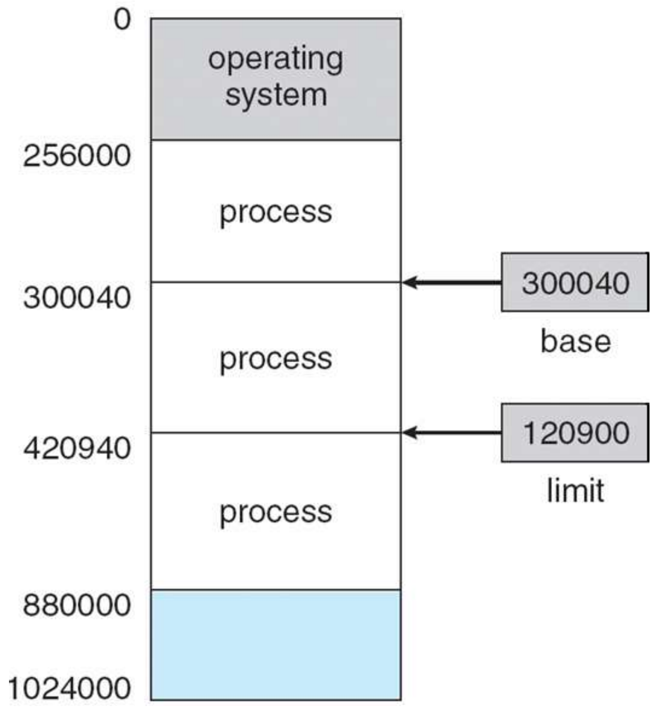
- Hardware device that at run time maps virtual to physical address
- Many methods possible
- To start, consider simple scheme where the value in the **relocation register** is added to every address generated by a user process at the time it is sent to memory
  - **relocation register**
  - MS-DOS on Intel 80x86 used 4 relocation registers
- The user program deals with *logical* addresses (0 to max); it never sees the *real* physical addresses (R to R+max)
  - Say the logical address 25
  - Execution-time binding occurs when reference is made to location in memory
  - Logical address bound to physical addresses

Dynamic relocation using a relocation register



Contiguous Allocation

Multiple processes resides in memory



Contiguous Allocation

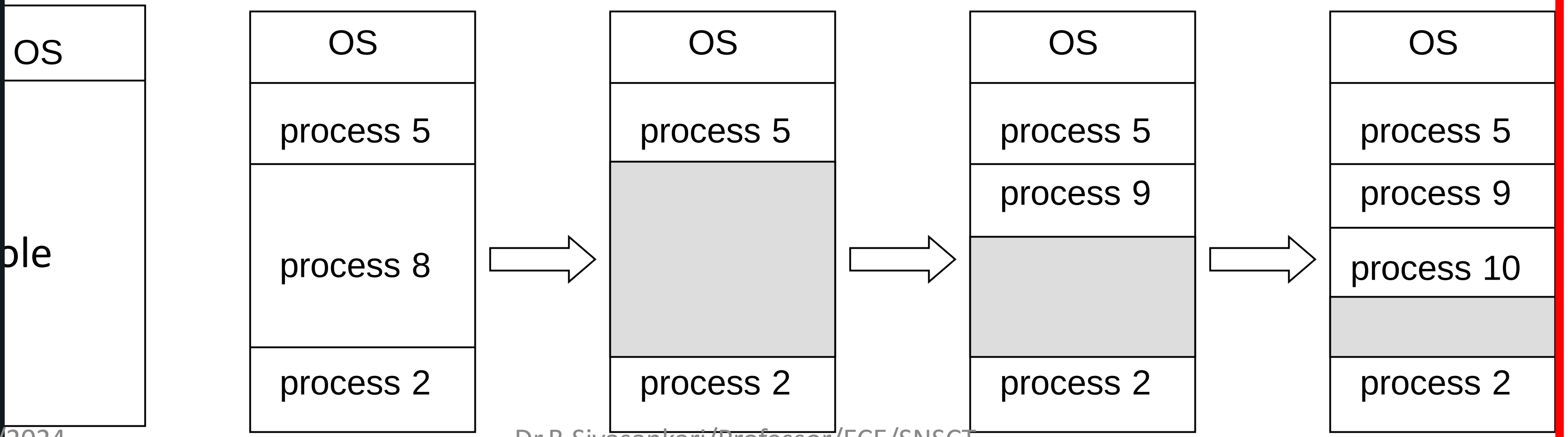
- Main memory usually divided into two partitions:
  - Resident operating system, usually held in low memory
  - User processes then held in high memory
  - Each process contained in **single contiguous section** of memory

Contiguous Allocation (Cont.)

- Multiple-partition allocation
  - Divide memory into several **Fixed size partition**
  - Each partition stores one process
  - Degree of multiprogramming limited by number of partitions
  - If a partition is free, load process from job queue
  - MFT (IBM OS/360)

Contiguous Allocation (Cont.)

- Multiple-partition allocation
  - **Variable partition scheme**
  - Hole – block of available memory; holes of various size are scattered throughout memory
  - Keeps a table of free memory
  - When a process arrives, it is allocated memory from a hole large enough to accommodate it
  - Process exiting frees its partition, adjacent free partitions combined
  - Operating system maintains information about:
    a) allocated partitions b) free partitions (hole)



Dynamic Storage-Allocation Problem

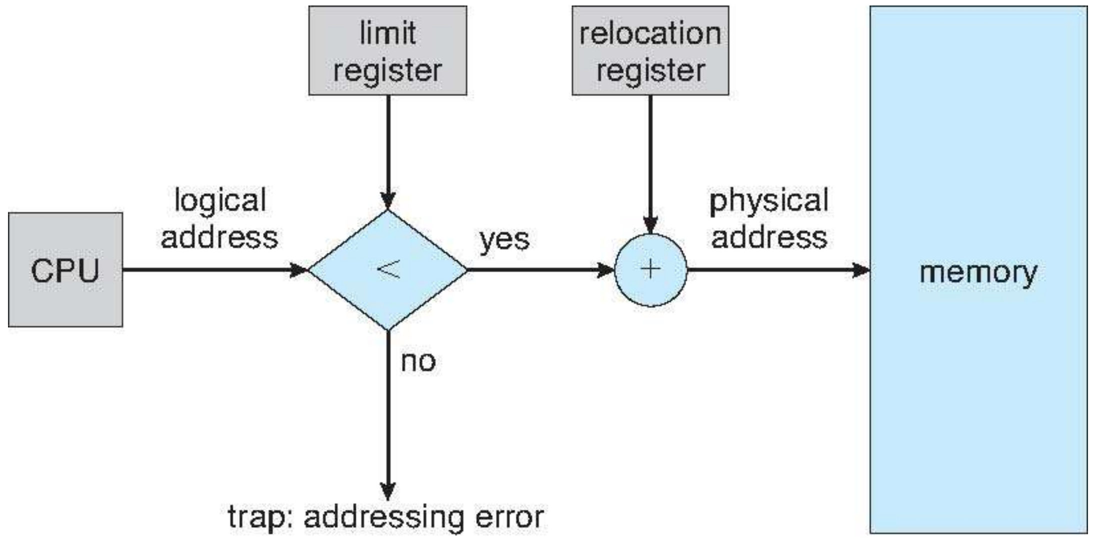
How to satisfy a request of size *n* from a list of free holes?

Dynamic storage allocation problem

- **First-fit**: Allocate the *first* hole that is big enough
- **Best-fit**: Allocate the *smallest* hole that is big enough; must search entire list, unless ordered by size
  - Produces the smallest leftover hole
- **Worst-fit**: Allocate the *largest* hole; must also search entire list
  - Produces the largest leftover hole

Hardware Support for Relocation and Limit Registers

- **Relocation registers** used to protect user processes from each other, and from changing operating-system code and data
  - Relocation register contains value of smallest physical address
  - Limit register contains range of logical addresses – each logical address must be less than the limit register
  - Context switch
  - MMU maps logical address *dynamically*



Fragmentation

- Processes loaded and removed from memory
  - Memory is broken into little pieces

- **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous

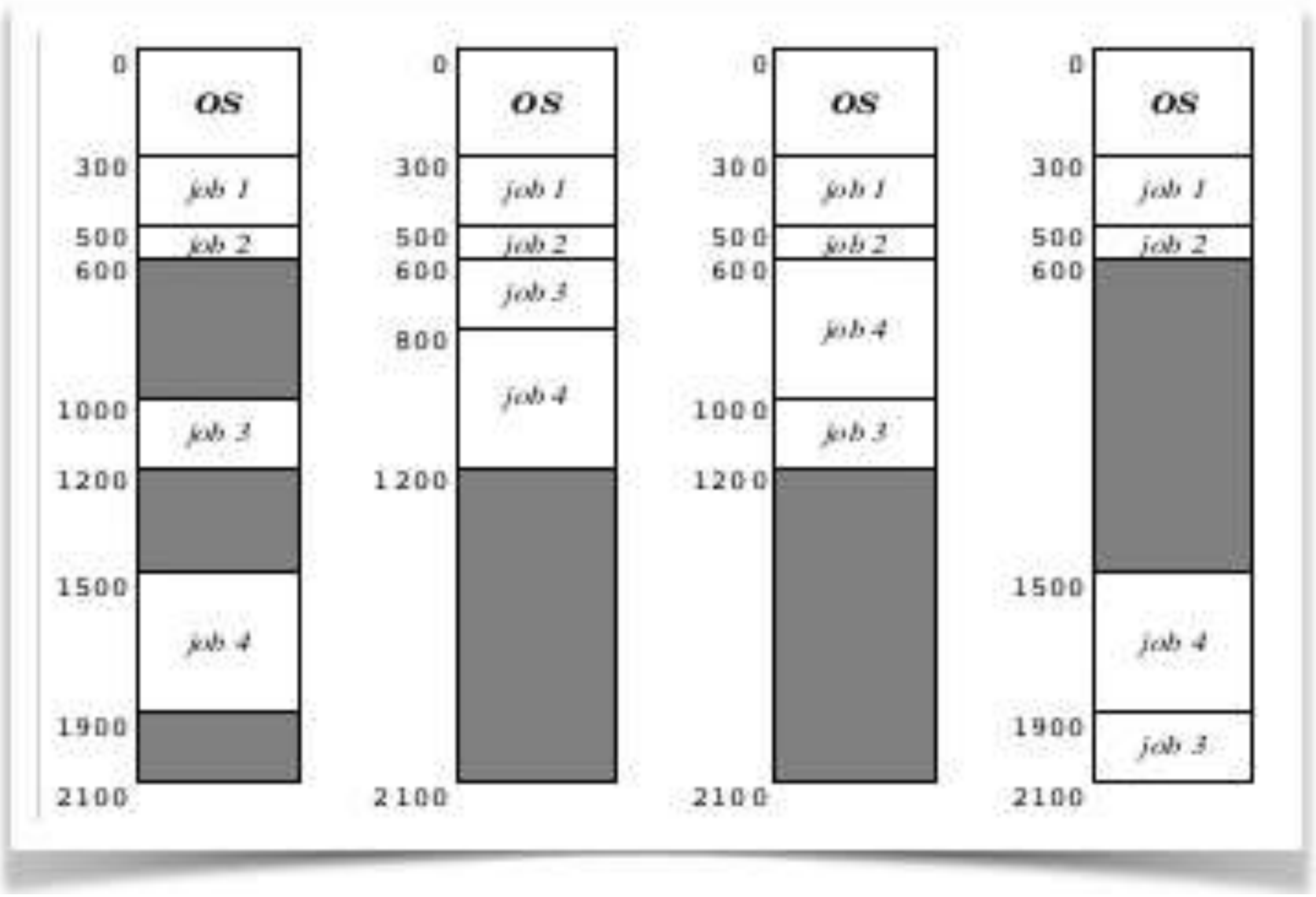
- First fit analysis reveals that given *N* blocks allocated, 0.5 *N* blocks lost to fragmentation
  - 1/3 may be unusable -> **50-percent rule**

Fragmentation (Cont.)

- 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
    - Change relocation reg.
  - Cost

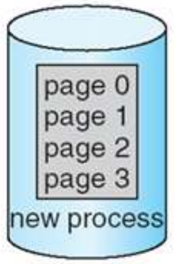
- **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used

Fragmentation (Cont.)



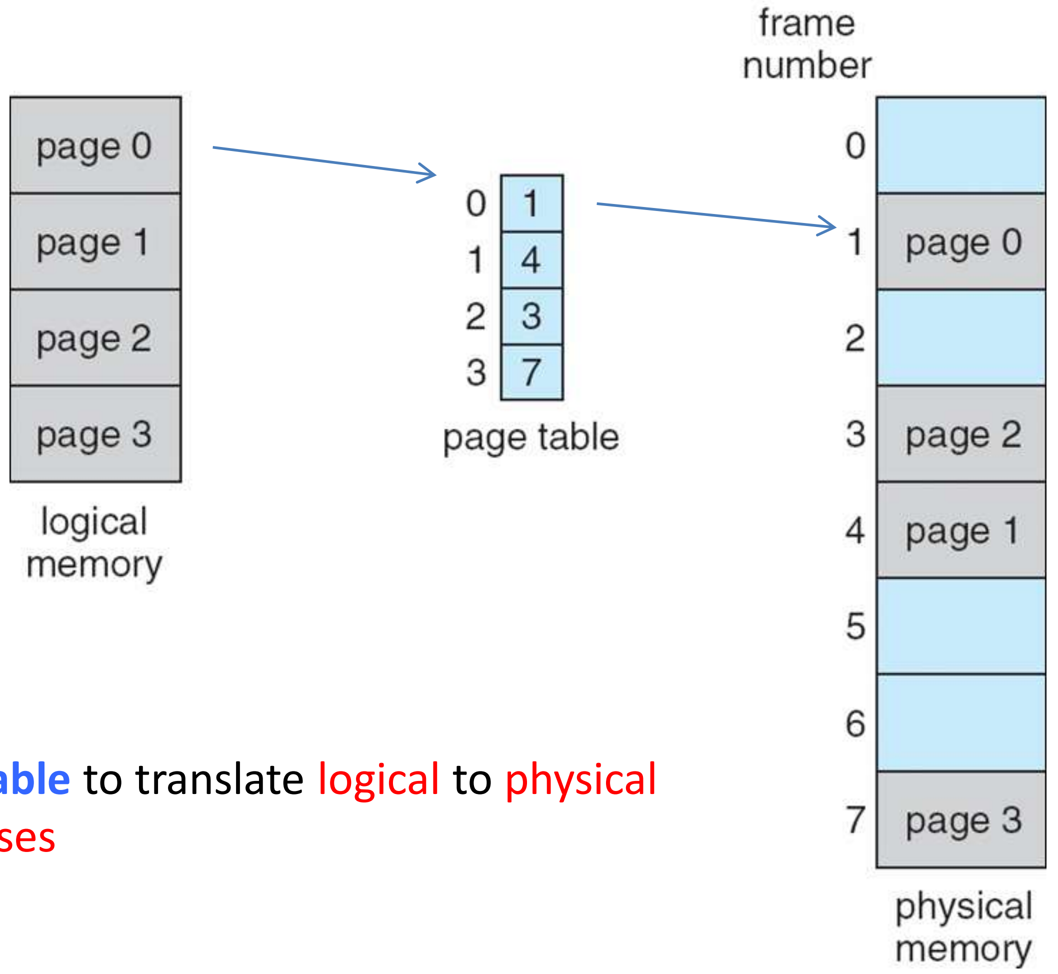
Paging

- Physical address space of a process can be noncontiguous;
  - process allocates physical memory whenever the latter is available
- 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**
  - To run a program of size *N* pages, need to find *N* free frames and load program
- Backing store likewise split into pages



- Set up a **page table** to translate logical to physical addresses
- System keeps track of all free frames

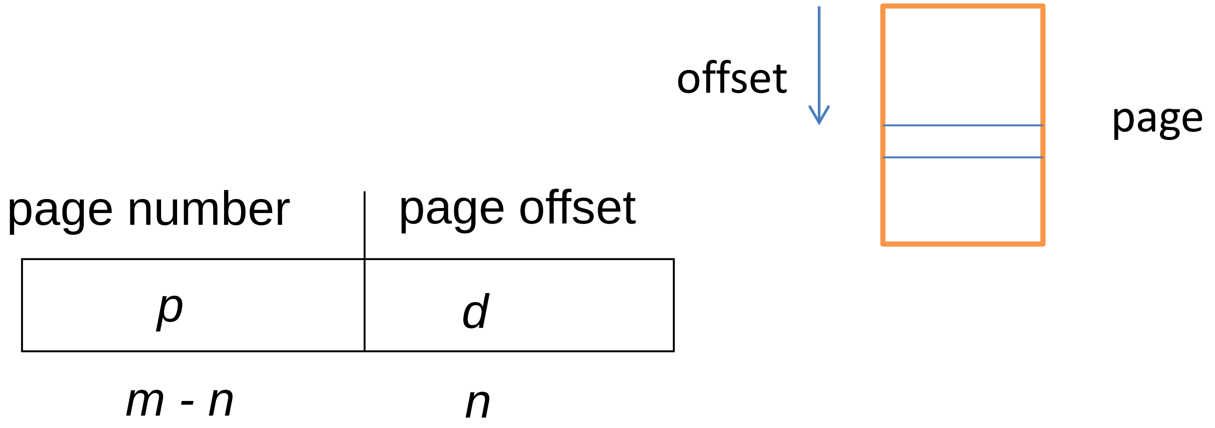
Paging Model of Logical and Physical Memory



page table to translate logical to physical addresses

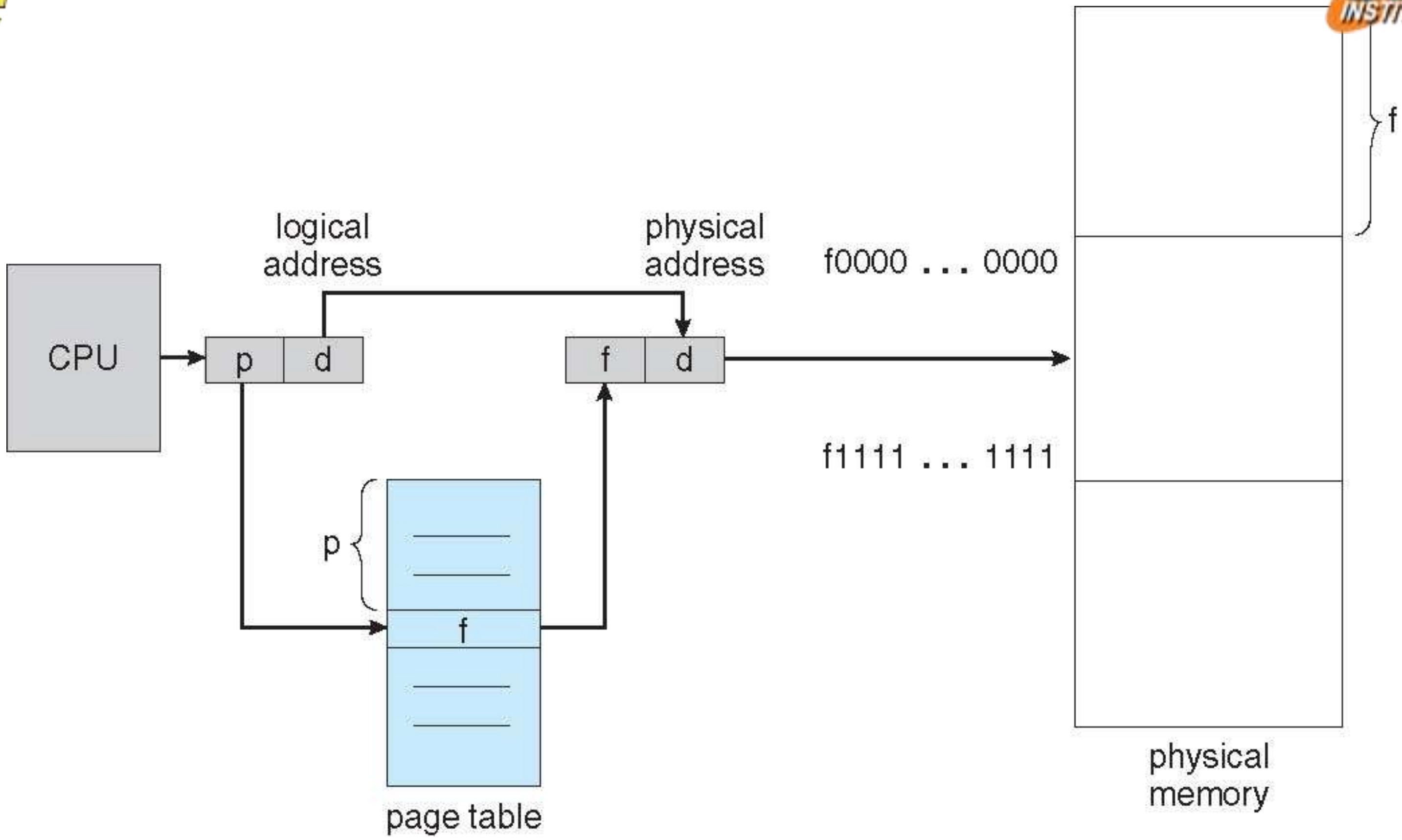
Address Translation Scheme

- 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*)** – offset within a page
    - combined with base address to define the physical memory address that is sent to the memory unit



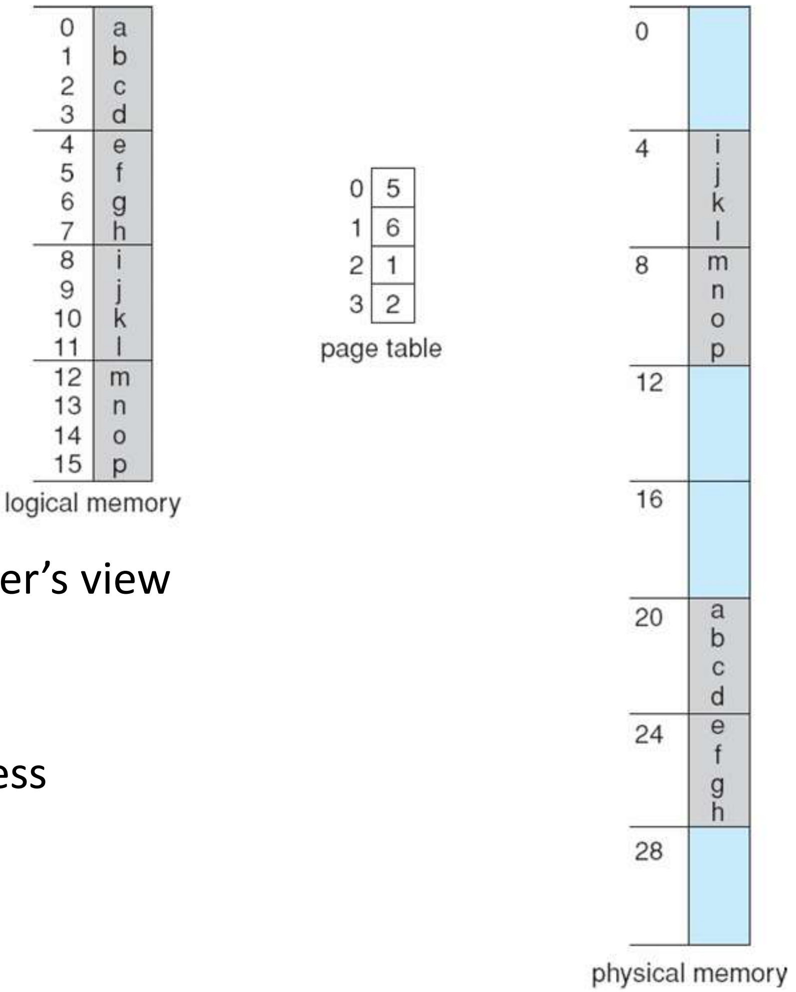
  - **For given logical address space 2^m and page size 2^n**

Paging Hardware



Paging Example

Logical address = 16
Page size=4
Physical memory= 32



User's view

Run time address binding

n=2 and *m*=4 32-byte memory and 4-byte pages

Logical address 0
(0*4+0)
Physical address:
(5*4+0)=20

Logical address 3
(0*4+3)
Physical address:
(5*4+0)=23

Logical address 4
(1*4+0)
Physical address:
(6*4+0)=24

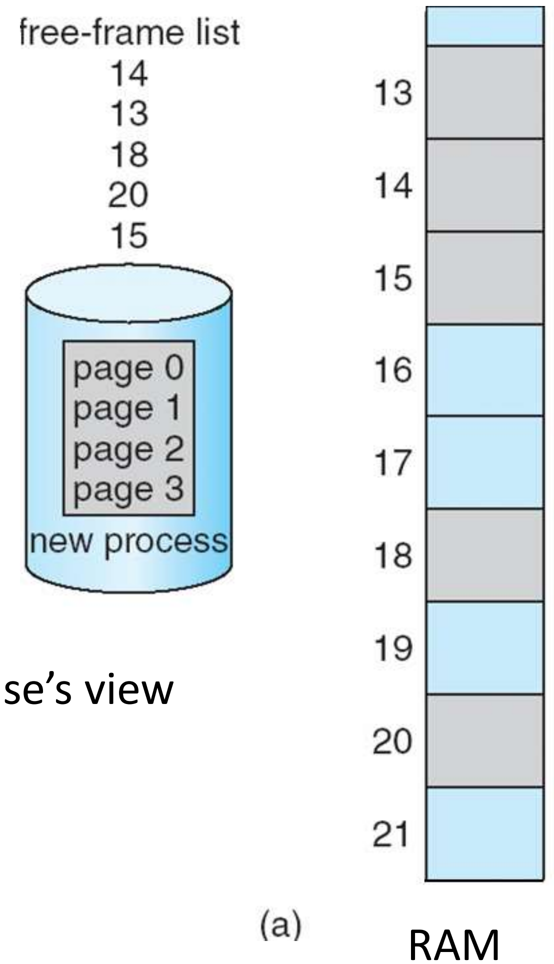
Logical address 13
(3*4+1)
Physical address:
(2*4+1)

Paging

- External fragmentation??
- Calculating internal fragmentation
  - Page size = 2,048 bytes
  - Process size = 72,766 bytes
  - 35 pages + 1,086 bytes
  - Internal fragmentation of 2,048 - 1,086 = 962 bytes
- So small frame sizes desirable?
  - But increases the page table size
  - Poor disk I/O
  - Page sizes growing over time
    - Solaris supports two page sizes – 8 KB and 4 MB
- User's view and physical memory now very different
  - user view=> process contains in single contiguous memory space
- By implementation process can only access its own memory
  - protection

e table

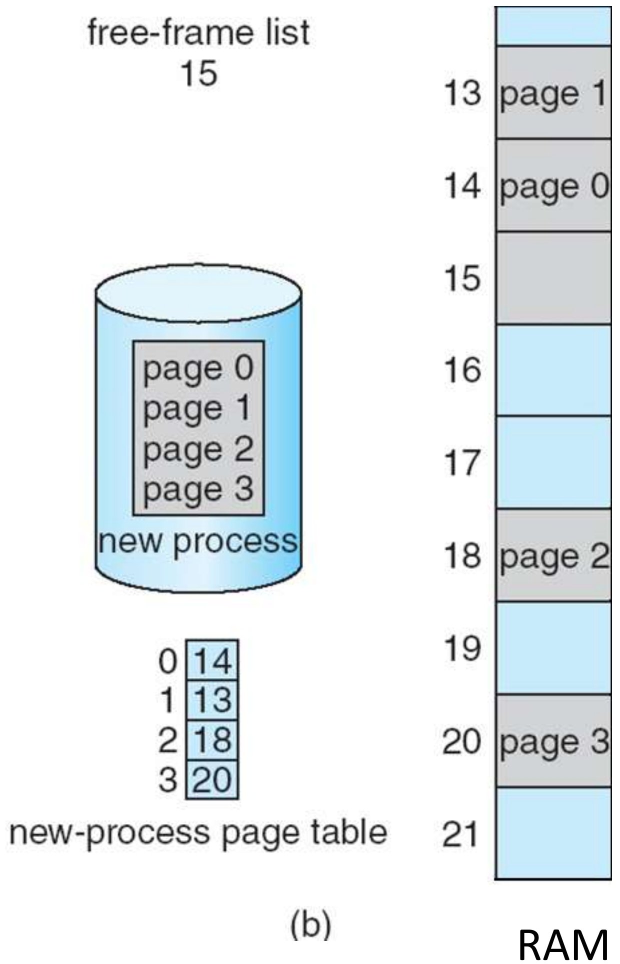
Free Frames



Use's view

RAM

Before allocation



System's view

RAM

After allocation

Implementation of Page Table

- For each process, Page table is kept in main memory
- **Page-table base register (PTBR)** points to the page table
- **Page-table length register (PTLR)** indicates size of the page table
- In this scheme every data/instruction access requires two memory accesses
  - One for the page table and one for the data / instruction
- The two memory access problem can be solved by the use of a special fast-lookup hardware cache called **associative memory** or **translation look-aside buffers (TLBs)**