

CYB 510 OPERATING SYSTEMS

Prepared by Prof. Dr. Lamiaa Fattouh

Lecture 7

جامعة الجميع الذكية

تسعى لتكون الكيان الأول الذي يتبنى مبدأ المجتمع التعليمي
التعاوني ويقدم العلم مقابل العمل لتنمية وبناء المجتمعات.

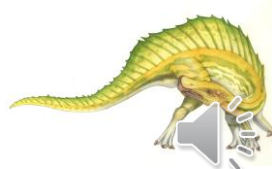
CYB 510

MAIN MEMORY



جامعة الجميع الذكية
EVERYONE'S SMART UNIVERSITY

- ❑ Background
- ❑ Swapping
- ❑ Contiguous Memory Allocation
- ❑ Segmentation
- ❑ Paging
- ❑ Structure of the Page Table



- ❑ **CPU** can be **shared by** a set of **processes**.
- ❑ As a result of **CPU scheduling**, we can **improve** both the **utilization** of the **CPU** and the **speed** of the computer's response to its users.
- ❑ To **increase performance**, we must keep several **processes** in **memory** and we must **share memory**



Register

- is a temporary storage area built into a CPU.
- There are many **types of registers** like **Accumulator**(Arithmetic, logic operations), **Data Register**, **Program Counter**, etc.

CACHE:

- It's a **temporary** storage, resides within the processor chip.
- It's both very **fast** as well as **nearer** to cpu **than ram**.
- The main aim is to try to **fill it with the data** which might be **needed again soon** to **speed up the computations**

Main memory

- The **programs** that should **execute** are brought from the **hard disk** into the **RAM** so that they can be **executed faster**



- ❑ **Main memory** and the **registers** built into the processor itself are the **only storage** that the **CPU** can **access directly**.
- ❑ Therefore, any **instructions in execution**, and any **data** being used by the instructions, **must be in one of these** direct-access **storage** devices.
- ❑ **CPUs** can **decode instructions** on **register** contents at the **rate** of **one or more operations per clock tick**. The **same cannot be said of main memory**, which is accessed via a **transaction** on the **memory bus**.
- ❑ There are **machine instructions** that **take memory addresses** as arguments, but none that take disk addresses.



Base and Limit Registers

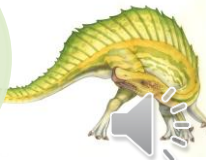
- We first need to **make sure** that **each process** has a **separate memory space**.
- **Separate per-process memory space** protects the processes from each other
- A pair of **base** and **limit registers** define the **logical address space** for a process

- **Holds** the **smallest** legal **physical memory address**;

The base register

- **specifies** the **size** of the **range**.

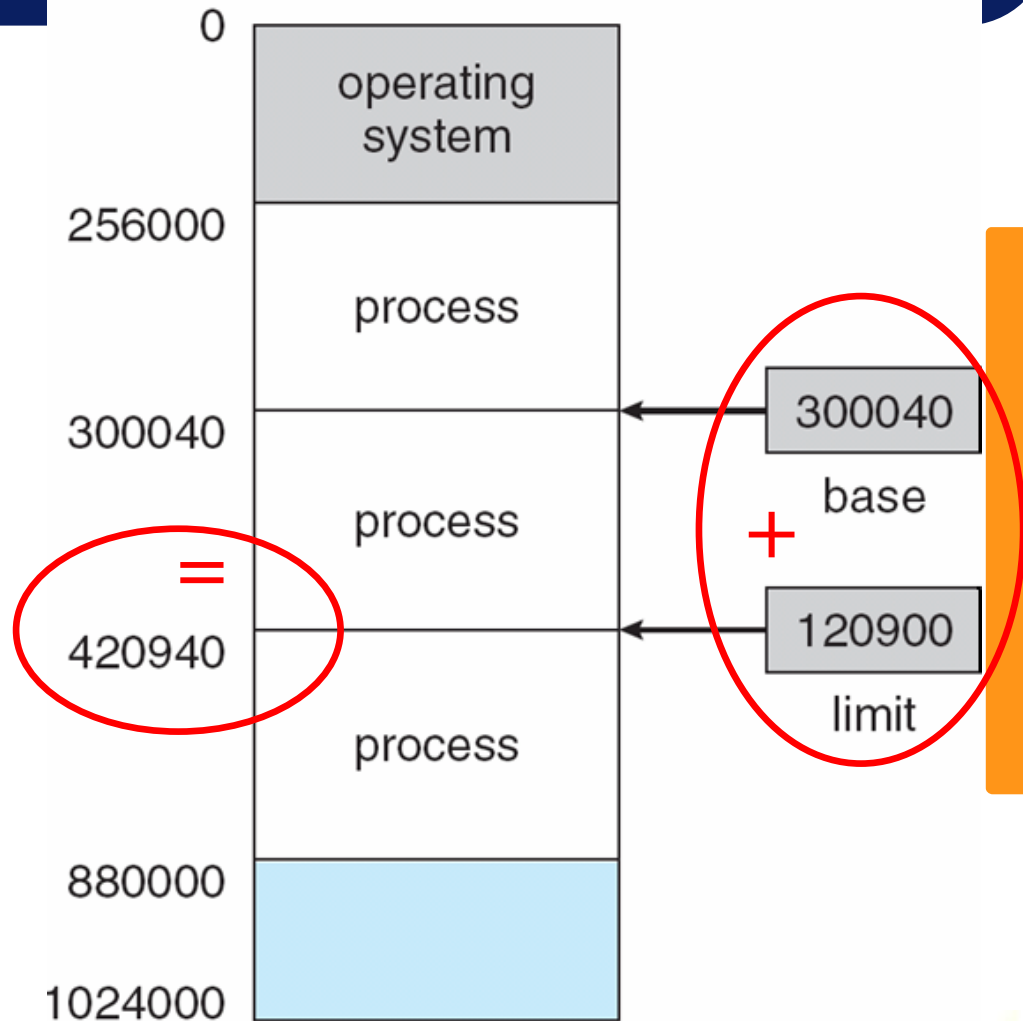
The limit register



Lecture 7

Base and Limit Registers

- For example, if the **base** register holds **300040** and
- The **limit** register is **120900**,
- Then the program can legally **access all addresses** from 300040 through **420939** (**inclusive**)



Lecture 7

Binding of Instructions and Data to Memory

Address binding of instructions and data to memory addresses can happen at **three different stages**

**Compile
time**

**Load
time**

**Execution
time**



Binding of Instructions and Data to Memory

Address binding of instructions and data to memory addresses can happen at **three different stages**

□ **Compile time:**

- is when the **code** is being **processed by a compiler**, that is transforming your code into an **executable binary**.
- must **recompile** code if **starting location changes**

□ **Load time:**

- Must **generate relocatable code** if memory location is **not known at compile time**

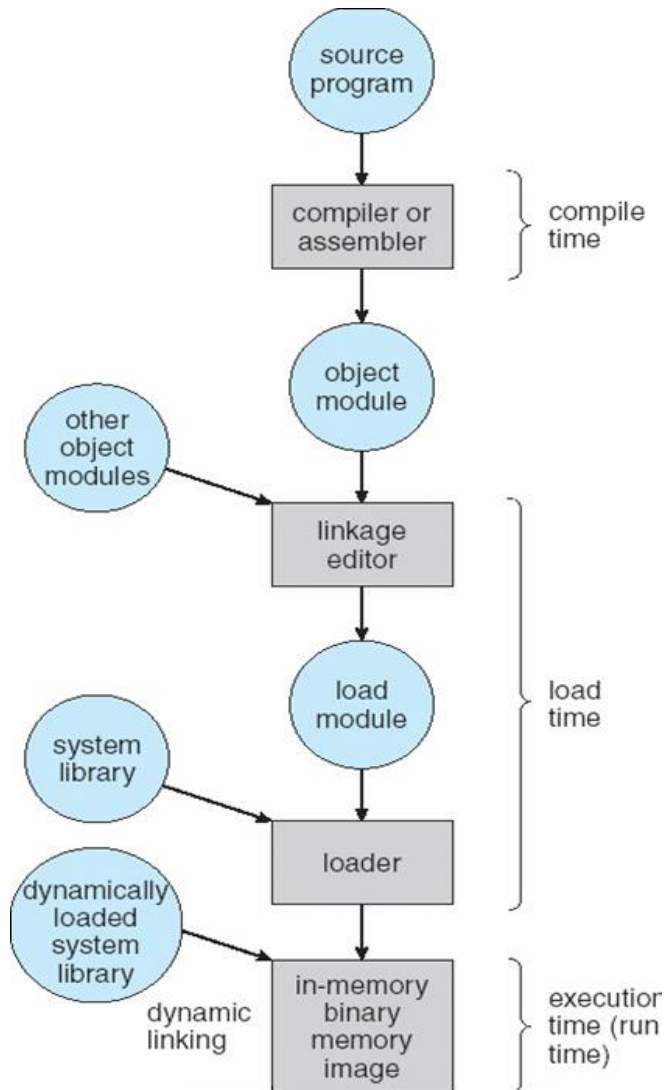
□ **Execution time:**

- **Binding delayed until run time** if the **process** can be **moved** during its **execution** from one **memory segment** to another.

- During program **execution**, the instructions are **in memory** and are being processed by the CPU. **Additional memory may be allocated** and/or **deallocated** at this time.



Multistep Processing of a User Program



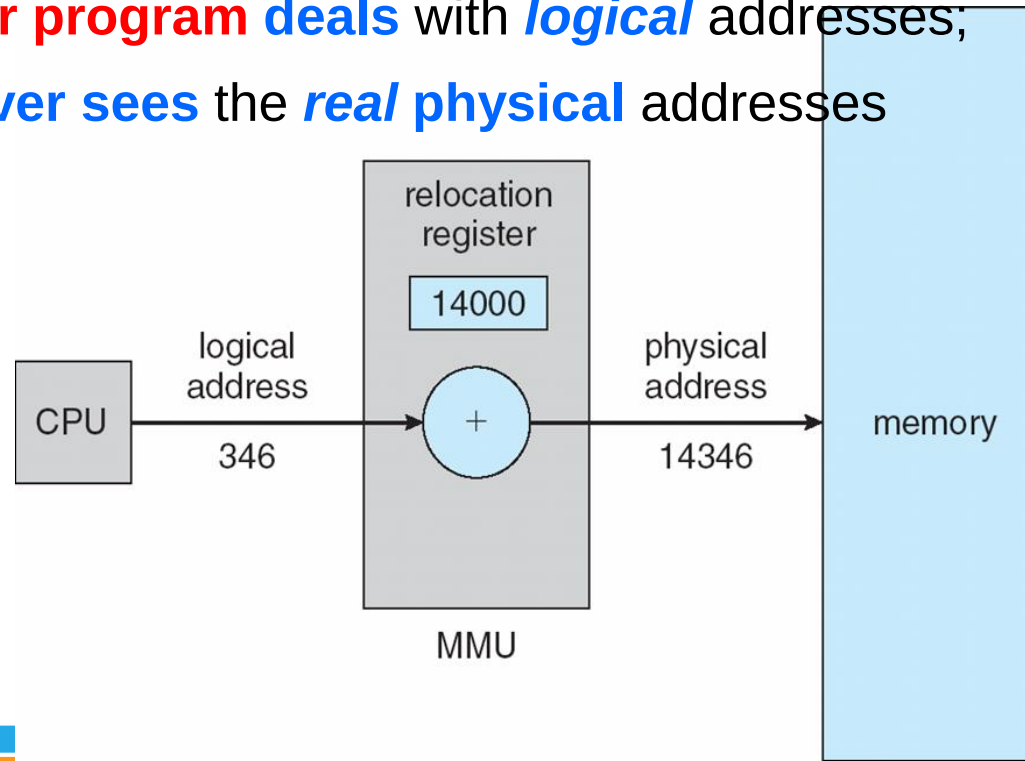
Logical vs. Physical Address Space

- The concept of a **logical address space** that is **bound to** a separate **physical address space** is central to proper memory management
 - **Logical address**
 - ▶ **generated** by the **CPU**;
 - ▶ also referred to as **virtual address**
 - **Physical address**
 - ▶ **address seen** by the **memory unit**
- They are **the same** in **compile-time** and **load-time** address-binding schemes;
- They **differ in execution-time** address-binding scheme

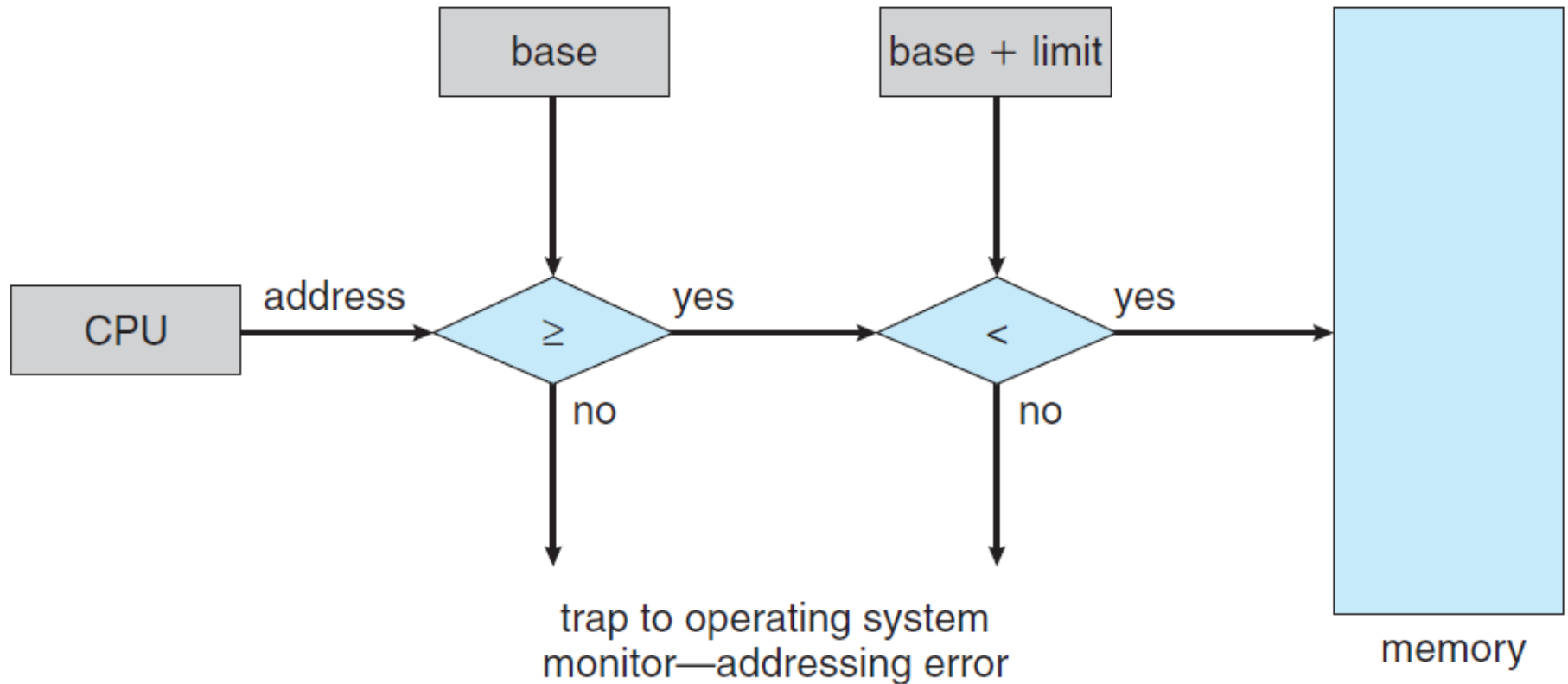


Memory-Management Unit (MMU)

- ❑ **MMU** is the **hardware device** that **maps virtual** to **physical** address
- ❑ 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



Address protection



Hardware address protection with base and limit registers.



Dynamic Loading

- ❑ **Routine** is **not loaded until** it is **called**
- ❑ **Better** **memory-space utilization**;
 - ❑ **unused** routine is **never loaded**
- ❑ **Useful** when **large amounts of code** are needed to handle **infrequently** occurring cases
- ❑ **No special support** from the **operating system** is required
 - ❑ **implemented through program design**



Dynamic Linking

- ❑ **Linking postponed** until **execution** time
- ❑ **Small piece of code**, **stub**, used to **locate** the **appropriate** memory-resident library **routine**
- ❑ **Stub replaces itself** with the **address of the routine**, and **executes** the **routine**
- ❑ **Operating system** needed to **check if routine** is **in processes' memory address**
- ❑ **Dynamic linking** is particularly **useful** for **libraries**



❑ also **known as shared libraries**



Swapping

- A **process** can be **swapped** temporarily **out of memory** to a **backing store**, and then **brought back into memory** for **continued execution**
- **Backing store** – **fast** disk **large enough** to accommodate **copies** of all memory images for all users;
 - **must** provide **direct access** to these memory images
- **Roll out, roll in** – **swapping** variant used **for priority-based scheduling** algorithms;
 - **lower-priority** process is **swapped out** so **higher-priority** process can be **loaded** and **executed**

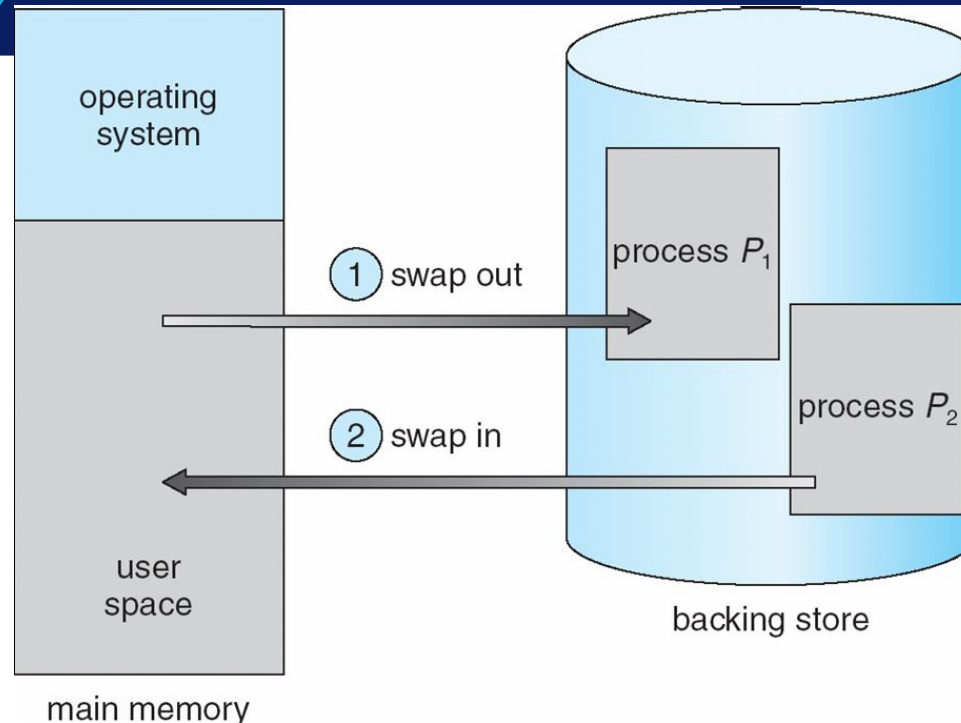


Swapping

- ❑ **System** maintains a **ready queue** of **ready-to-run processes** which **have memory images on disk**
- ❑ **Swapped process** should be **idle**
- ❑ **I/O problem**
 - ❑ Assume that the **I/O operation** is queued because the device is busy.
 - ❑ If we were to **swap out** process P_1 and swap in process P_2 ,
 - ❑ **I/O operation might** then attempt to **use memory** that now **belongs** to **process P_2** ,
- ❑ **Solution**
 - ▶ **Keep job** in memory while it is **involved in I/O**
 - ▶ **Do I/O only** into **OS buffers**



Schematic View of Swapping



- Notice that the **major** part of the **swap time** is **transfer time**. The **total transfer time** is directly **proportional** to the **amount of** memory **swapped**.



- **time** could be **reduced if exact memory use** is **known**.



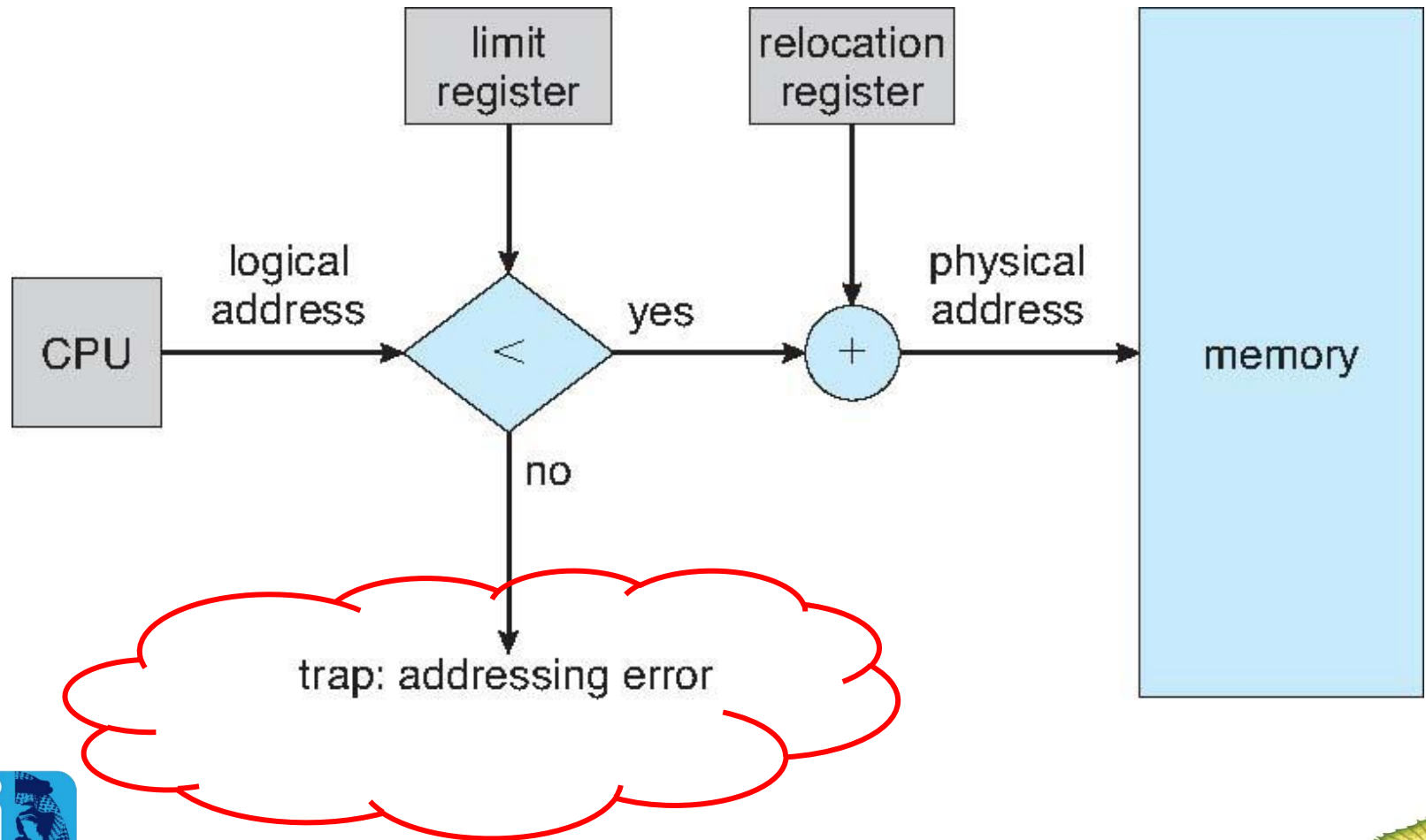
Contiguous Allocation

- **Main memory** usually into **two partitions**:
 - **Resident operating system**,
 - ▶ **usually** held in **low memory** with **interrupt vector**
 - **User processes**
 - ▶ then held in **high memory**
- **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**



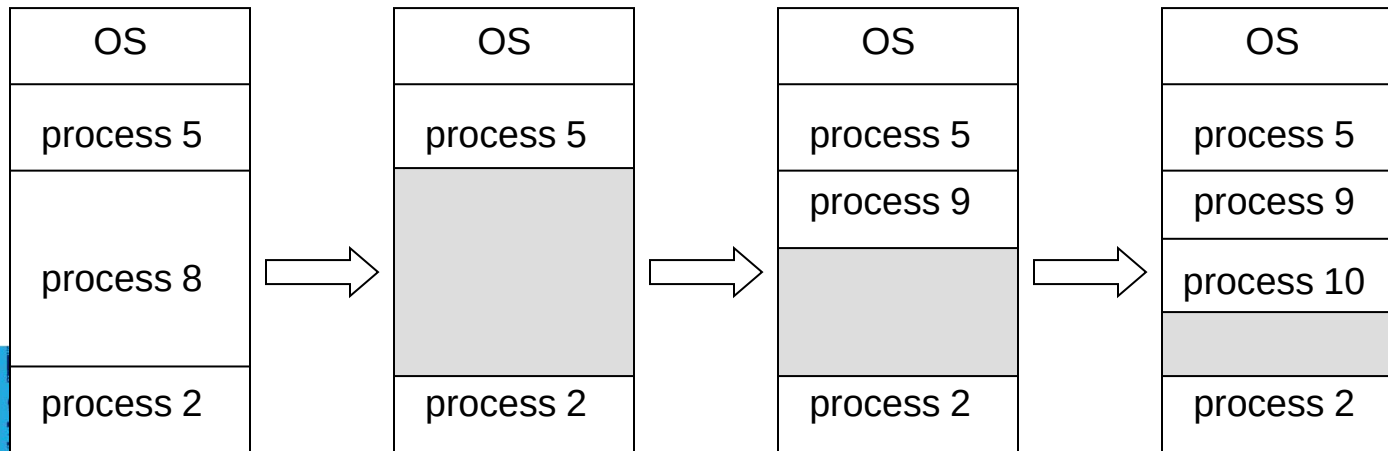
Lecture 7

Hardware Support for Relocation and Limit Registers



Contiguous Allocation (Cont)

- ❑ **Multiple-partition allocation**
 - ❑ **Hole – block** of **available** memory;
 - ▶ **holes** of **various size** are scattered throughout memory
 - ❑ When a **process arrives**, it is **allocated memory** from a **hole large enough** to accommodate it
 - ❑ **Operating system maintains information** about:
 - a) **allocated** partitions b) **free** partitions (hole)



Dynamic Storage-Allocation Problem

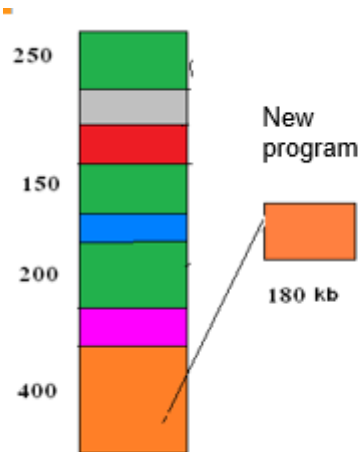
How to satisfy a **request** of size n from a **list of free holes**

- **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**
- **First-fit** and **best-fit** **better** than worst-fit in terms of **speed** and **storage utilization**.
- **First-fit** generally **fastest**.

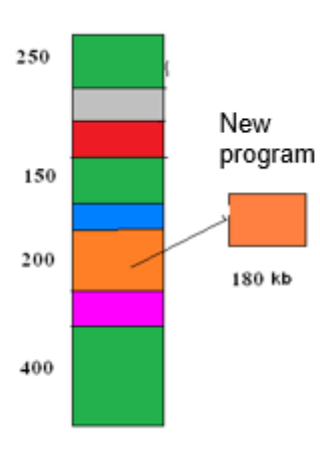


Dynamic Storage-Allocation

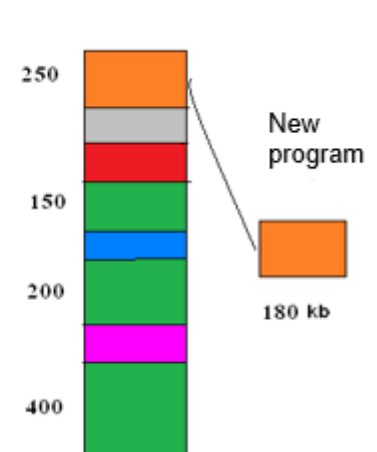
Worst-fit



Best-fit



First-fit



Example

Dynamic Storage-Allocation

Suppose we have memory spaces ordered as follows

500 ، 250 ، 300 ، 400

Load the following programs according to the following sizes

230 - 450 - 350 - 100

Using first-fit, best-fit and worst fit methods

then explain which is better in memory management



Lecture 7

230 - 450 - 350 - 100

First fit

230

450

350

best fit

230

450

350

100

worst fit

230

450

350

100

230

100

350

500

250

300

400

230

100

350

450

230

100

350

230

100

350



Lecture 7



First fit

$$\begin{aligned}\text{Left space} &= (500 - 230) + (250 - 100) + 300 + (400 - 350) \\ &= 270 + 150 + 300 + 50 = 770 \text{ KB}\end{aligned}$$

Best fit

$$\begin{aligned}\text{Left space} &= (500 - 450) + (250 - 230) + (300 - 100) + (400 - 350) \\ &= 50 + 20 + 200 + 50 = 320 \text{ KB}\end{aligned}$$

Worst fit

$$\begin{aligned}\text{Left space} &= (500 - 230) + (250) + (300 - 100) + (400 - 350) \\ &= 270 + 250 + 200 + 50 = 770 \text{ KB}\end{aligned}$$



Fragmentation

- **External Fragmentation** – **total memory space exists** to **satisfy** a request, but it is **not contiguous**
 - e.g. first-fit can on average use 2/3 of memory.
- **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**



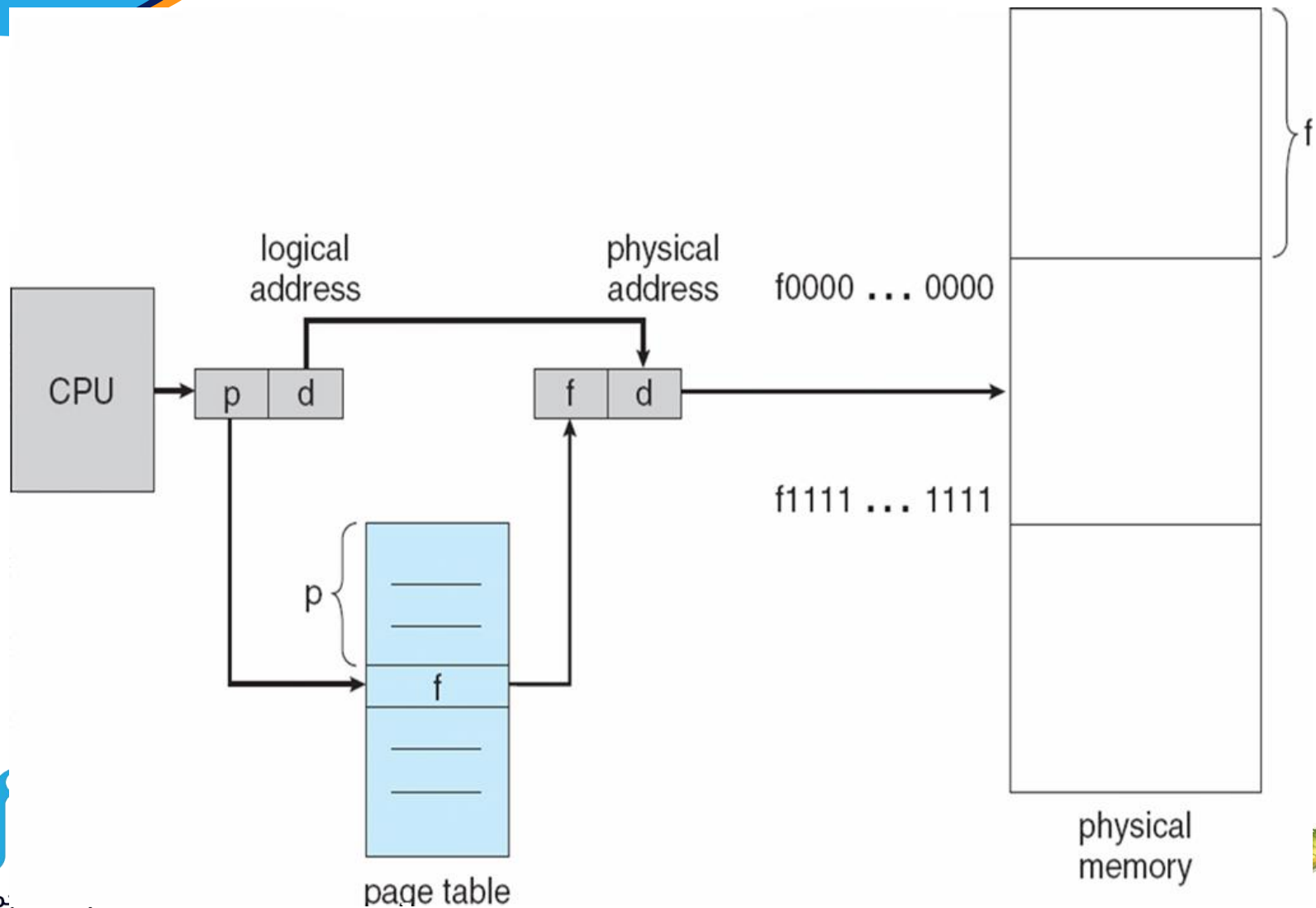
Paging

- ❑ **Logical address space** of a process can be **noncontiguous**;
 - ❑ **process** is allocated 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 **8,192** bytes
- ❑ **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

❑ **Internal fragmentation** s handled



Paging Hardware



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

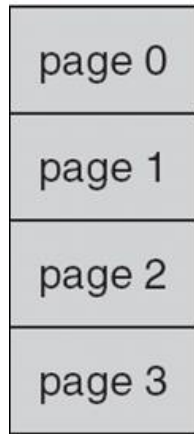
page number	page offset
p	d
$m - n$	n

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



Lecture 7

Paging Model of Logical and Physical Memory

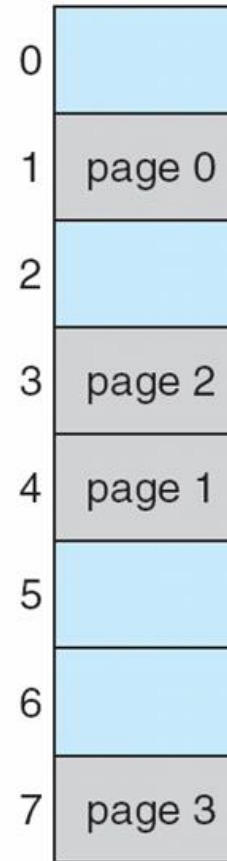


logical
memory

0	1
1	4
2	3
3	7

page table

frame
number



physical
memory



Paging Example

0	a
1	b
2	c
3	d
4	e
5	f
6	g
7	h
8	i
9	j
10	k
11	l
12	m
13	n
14	o
15	p

logical memory

0	5
1	6
2	1
3	2

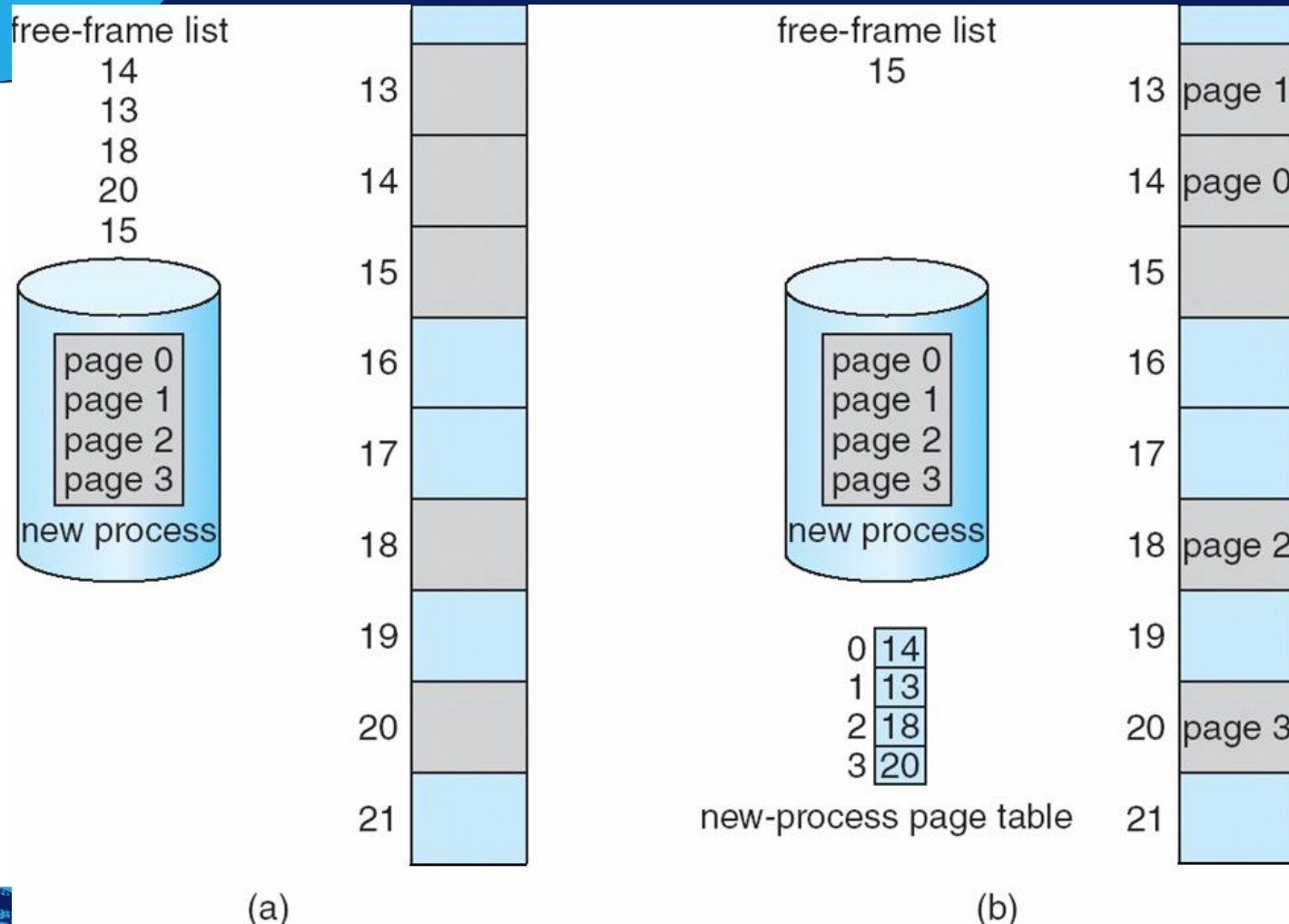
page table

0	
4	i j k l
8	m n o p
12	
16	
20	a b c d
24	e f g h
28	

physical memory



Free Frames



END OF LECTURE 7

