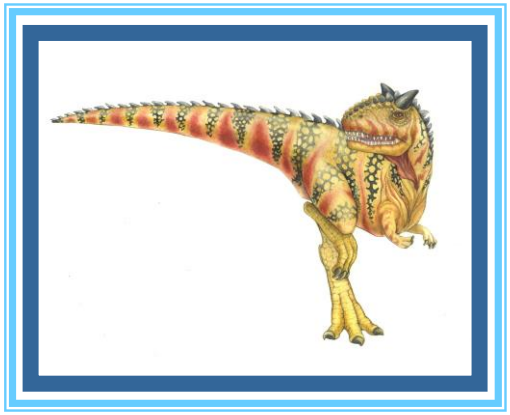
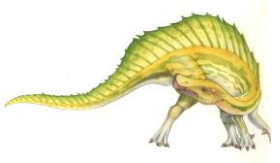
Chapter 7: Memory Management



Chapter 7: Memory Management

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



Objectives

- To provide a detailed description of various ways of organizing memory hardware
- To discuss various memory-management techniques, including paging and segmentation

Background

program is passive
process is active

passive

active.

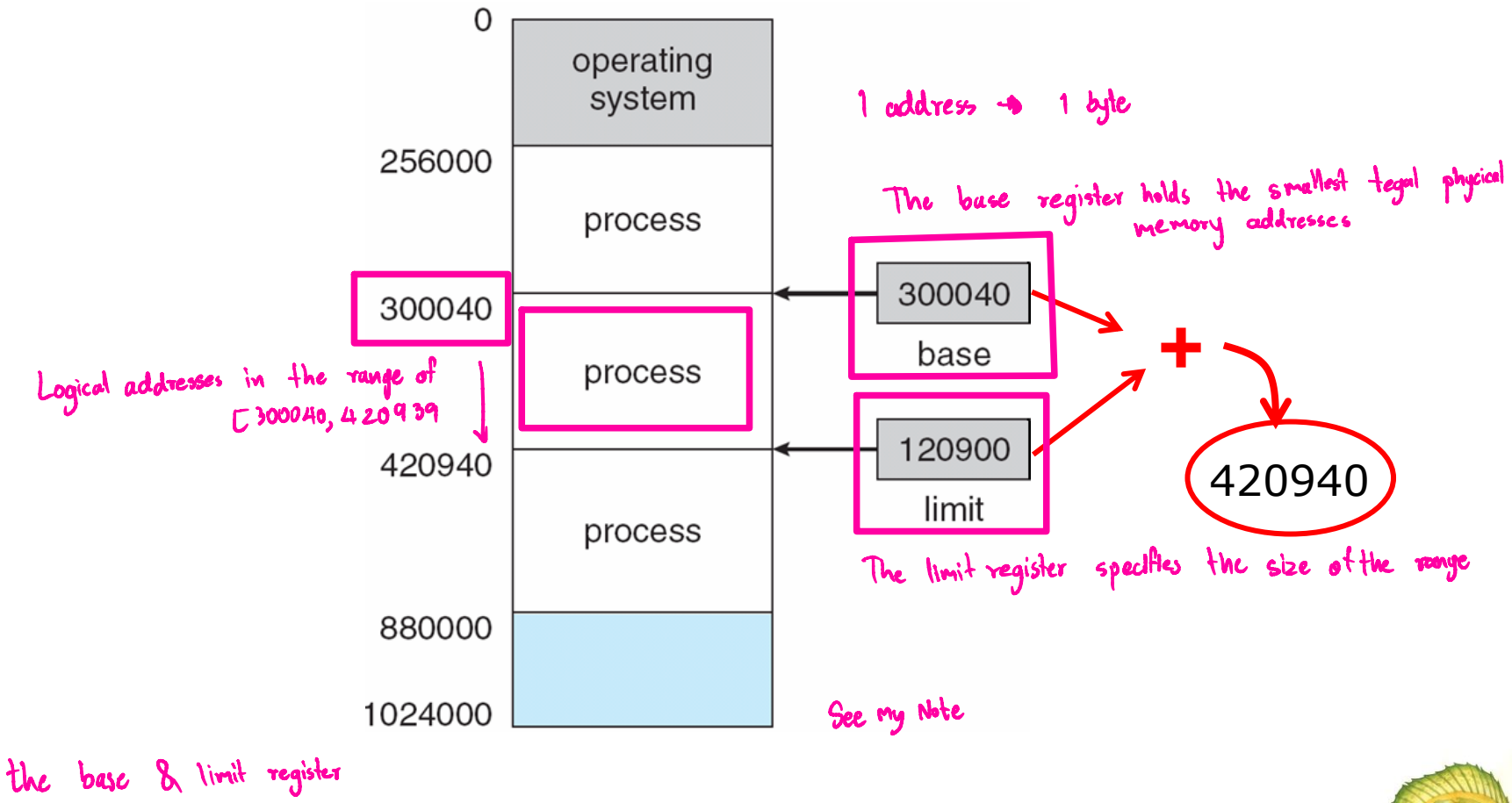
- Program must be brought (from disk) into memory and placed within a process for it to be run
- Main memory and registers are only storage CPU can access directly
- Register access in one CPU clock (or less)
- Main memory can take many cycles of the CPU clock
- Cache sits between main memory and CPU registers
- Protection of memory required to ensure correct operation

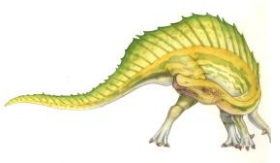
Having a cache built the CPU. the H/W automatically speeds up memory access witout any OS control

brought : ถูกนำพา

Base and Limit Registers

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





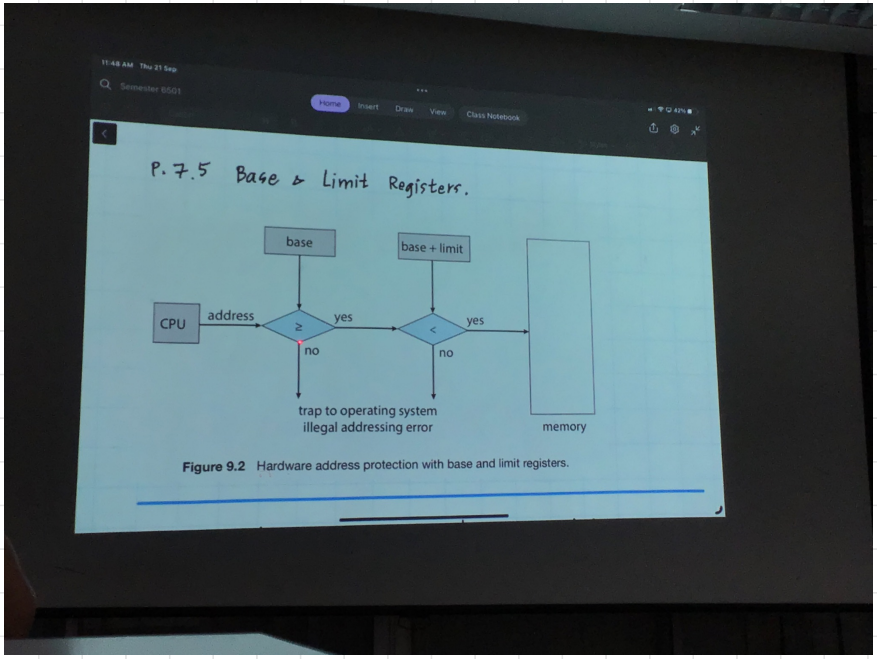


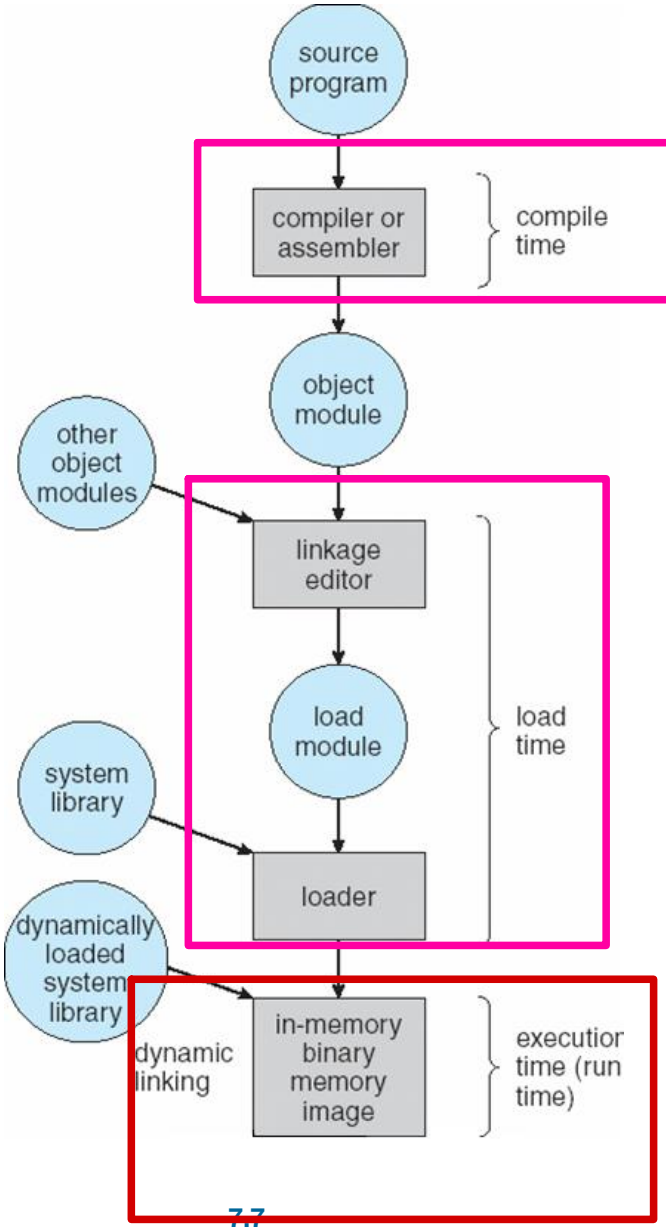
Figure 9.2 Hardware address protection with base and limit registers.

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**: Binding delayed until run time if the process can be moved during its execution from one memory segment to another. Need hardware support for address maps (e.g., base and limit registers)

Binding: การกำหนดค่า

Multistep Processing of a User Program



main.c

gcc -c main.c

generates

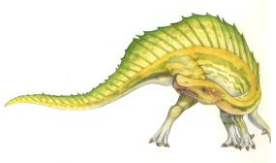
main.o (object file)

gcc -o main main.o -lm

generates

main (execute file)

./main



Logical vs. Physical Address Space

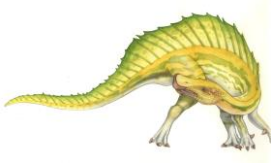
- The concept of a logical address space that is bound to a separate **physical address space** is central to proper memory management
  - **Logical address** – generated by the CPU; also referred to as **virtual address**
  - **Physical address** – address seen by the memory unit
- Logical and physical 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

Memory-Management Unit (MMU)

Logical Address

- 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

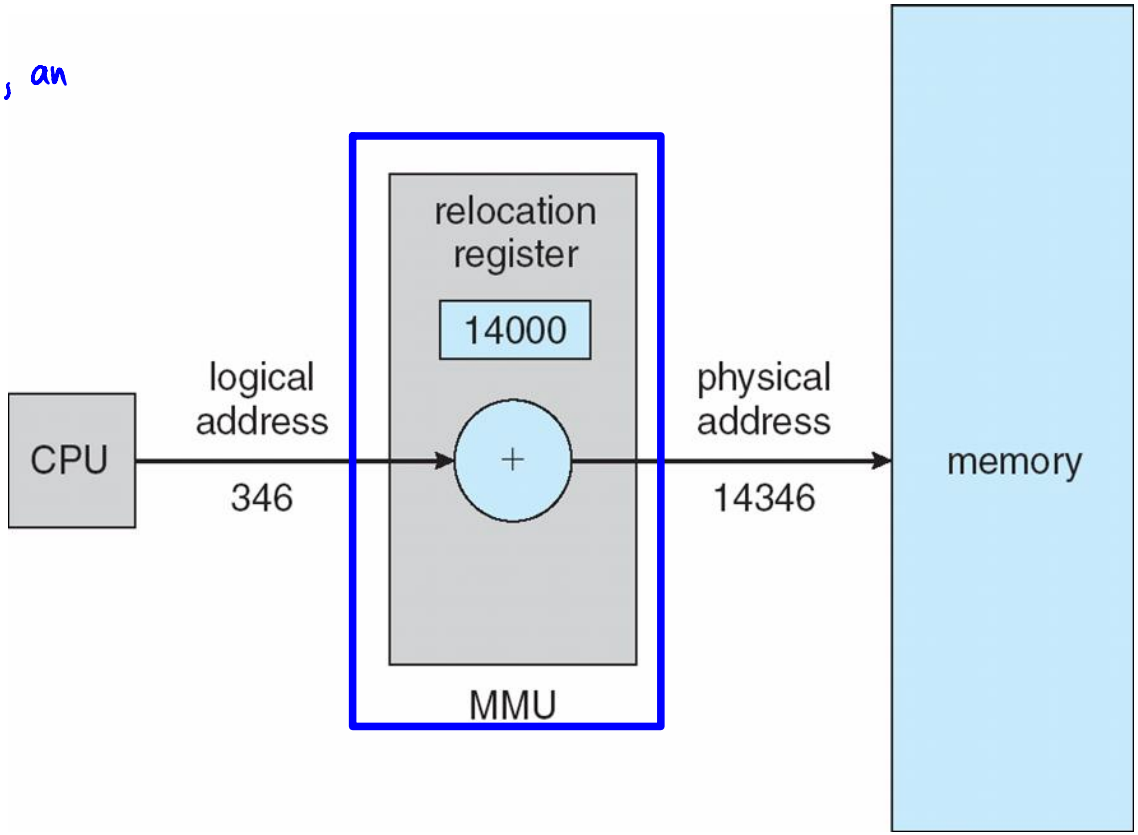
the user program never accesses the real physical addresses.
the user program deals with logical addresses
Then, the memory

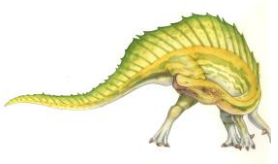


Dynamic relocation using a relocation register

Run-time or Execution-time binding

thus, an





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

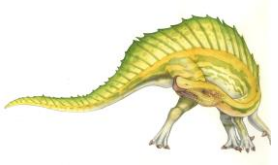
All routine are kept on disk in relocatable load format. The main program is loaded into memory and is executed

Ex Error routines.

It is responsibility of the users to design their program

When routine needs to call another routine, the calling routine first check to see whether the other routine has

Routine: โปรแกรมย่อย



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

  Use the routine address to link to routien code for execution
- Operating system needed to check if routine is in processes' memory address
- Dynamic linking is particularly useful for libraries
- System also known as **shared libraries**

Dynamic linking (compared to static linking)

Pro 1 Smaller size of the executable image

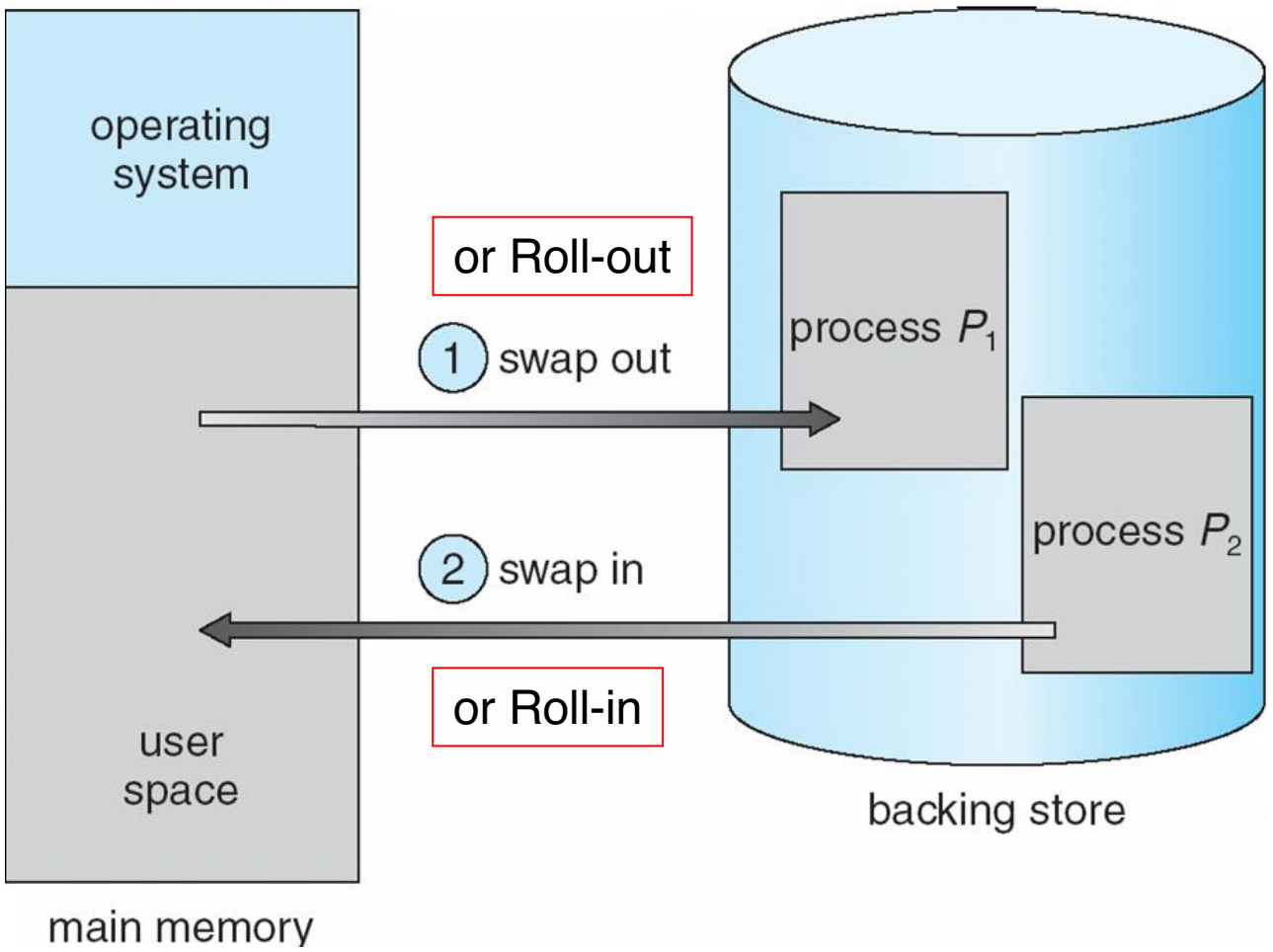
Pro 2 DLLs (libraries) can be shared among multiple processes, so that only 1 instance DLL in main memory

Pro 3 A library may be replaced by a new version, and all programs that reference the 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
- Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped
- System maintains a **ready queue** of ready-to-run processes which have memory images on disk
- Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)
  - Swapping normally disabled
  - Started if more than threshold amount of memory allocated
  - Disabled again once memory demand reduced below threshold

Schematic View of Swapping



Contiguous Allocation

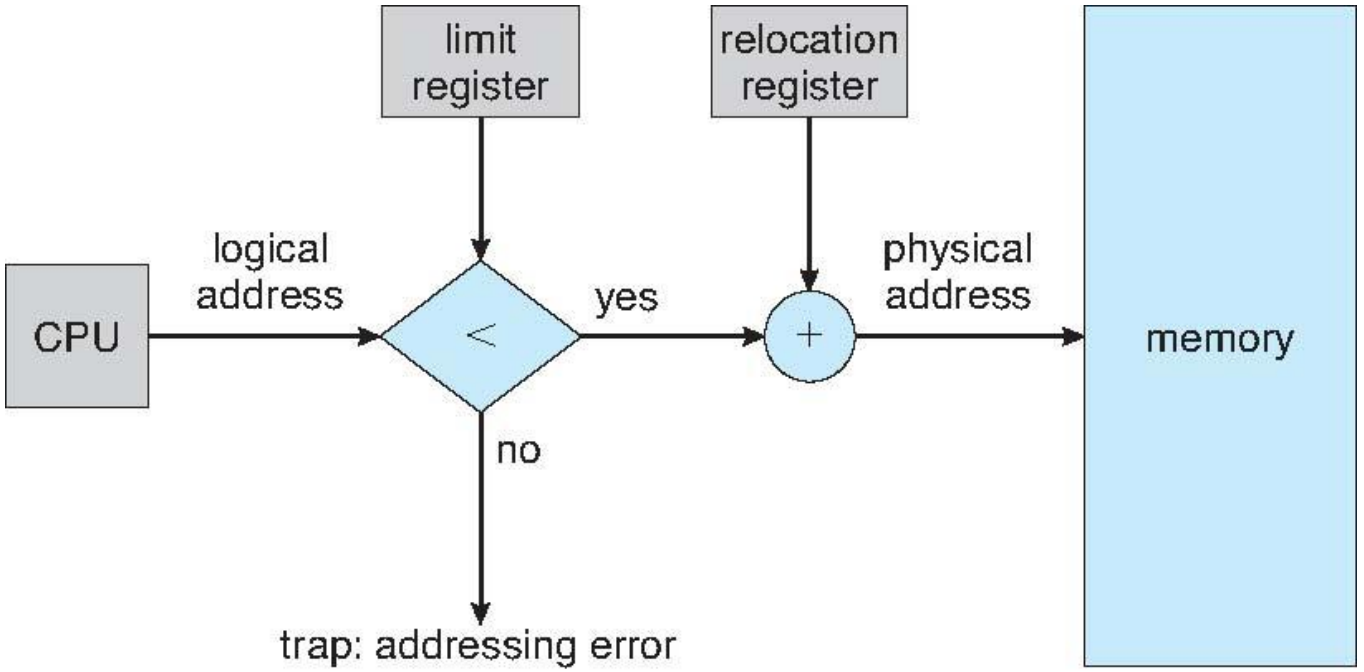
(การจัดสรรพื้นที่ที่ติดกัน)

- Main memory usually into two partitions:
  - Resident operating system, usually held in low memory with interrupt vector (โปรแกรมของระบบปฏิบัติการเอง)
  - 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*

contiguous : ติดกัน

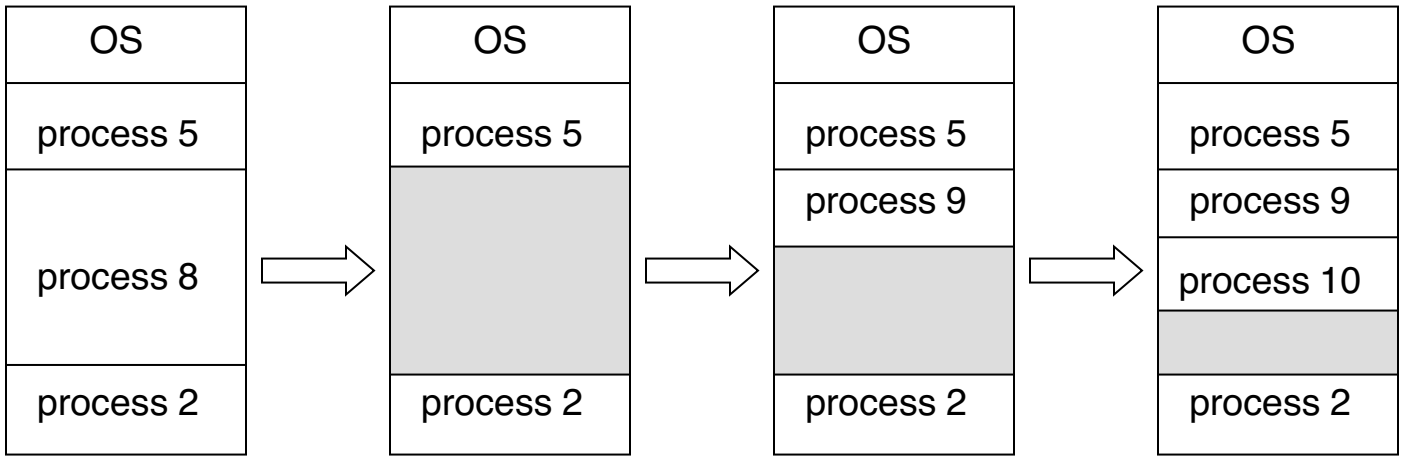
Hardware Support for Relocation and Limit Registers



Contiguous Allocation (Cont)

- Multiple-partition allocation (การจัดสรรเนื้อที่แบบหลายส่วน)
  - Degree of multiprogramming limited by number of partitions
  - **Variable-partition** sizes for efficiency (sized to a given process' needs)
  - **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)

อาจใช้การจัด Schedule แบบ FCFS

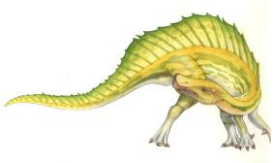


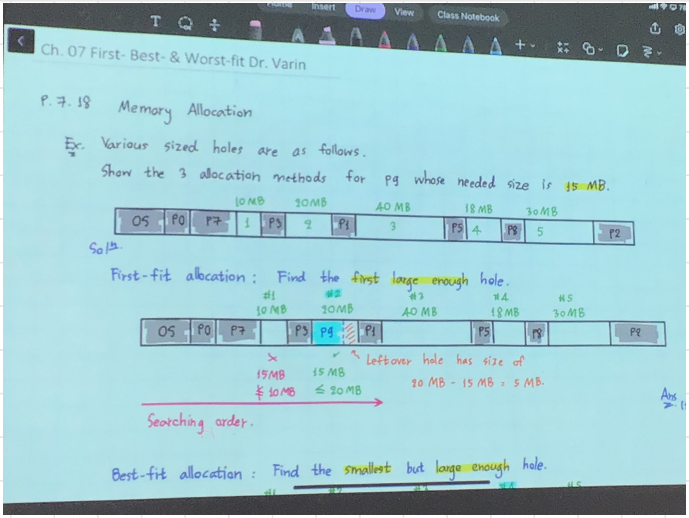
Dynamic Storage-Allocation Problem

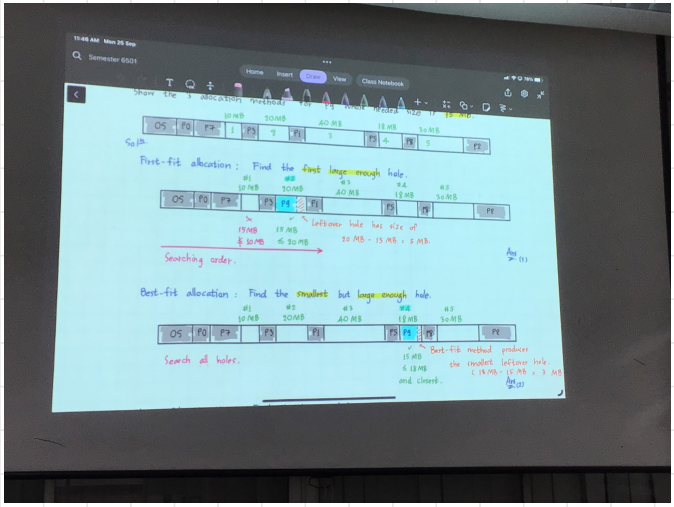
How to satisfy a request of size ***n*** from a list of free holes

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

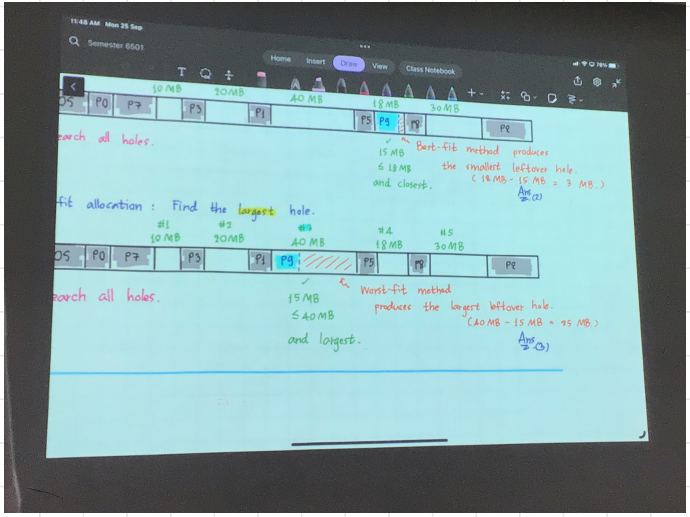
First-fit and best-fit better than worst-fit in terms of speed and storage utilization




Ch. 07 First- Best- & Worst-fit Dr. Varin
P. 7. 18 Memory Allocation
Ex. Various sized holes are as follows.
Show the 3 allocation methods for Pg whose needed size is 15 MB.
10 MB 20 MB 40 MB 18 MB 30 MB
OS P0 P7 1 P3 2 P1 3 P5 4 P8 5 P2
Sol:
First-fit allocation : Find the first large enough hole.
#1 #2 #3 #4 #5
10 MB 20 MB 40 MB 18 MB 30 MB
OS P0 P7 P3 Pg P1 P5 P8 P2
15MB ≤ 20MB
15MB ≤ 20MB
Leftover hole has size of
20 MB - 15 MB = 5 MB.
Searching order.
Best-fit allocation : Find the smallest but large enough hole.

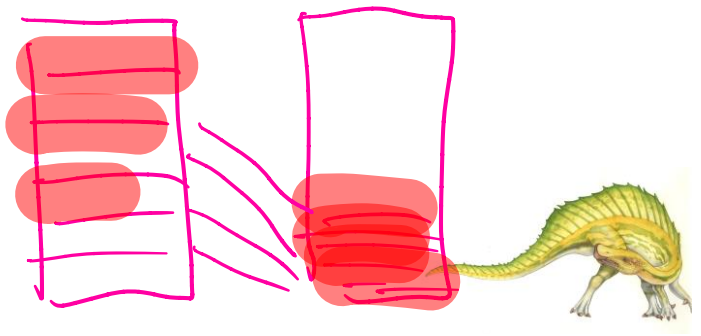

Semester 6501
Home Insert Draw View Class Notebook
First-fit allocation : Find the first large enough hole.
Searching order.
Best-fit allocation : Find the smallest but large enough hole.
Search all holes.
Best-fit method produces the smallest leftover hole.
(18 MB - 15 MB = 3 MB)

worst- fit


Semester 6501
Home Insert Draw View Class Notebook
10 MB 20 MB 40 MB 18 MB 30 MB
OS P0 P7 P3 P1 P5 Pg P8 P2
Search all holes.
15 MB ≤ 18 MB and closest.
Best-fit method produces the smallest leftover hole.
(18 MB - 15 MB = 3 MB)
Ans (2)
fit allocation : Find the largest hole.
#1 #2 #3 #4 #5
10 MB 20 MB 40 MB 18 MB 30 MB
OS P0 P7 P3 P1 Pg P5 P8 P2
Search all holes.
15 MB ≤ 40 MB and largest.
Worst-fit method produces the largest leftover hole.
(40 MB - 15 MB = 25 MB.)
Ans (3)

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



compaction: การบีบอัด
satisfy: ปฏิบัติตาม
shuffle:สับเปลี่ยน

Paging การสลับหน้า

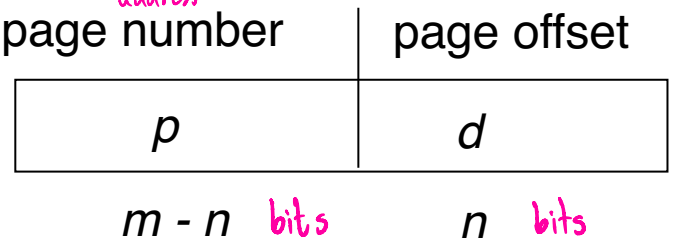
- Physical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available
- Divide physical memory into fixed-sized blocks called **frames** (size is power of 2, between 512 bytes and 16 Mbytes)
- Divide logical memory into blocks of same size called **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
- 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

first part of the logical address

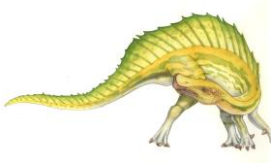
Second part of the logical address



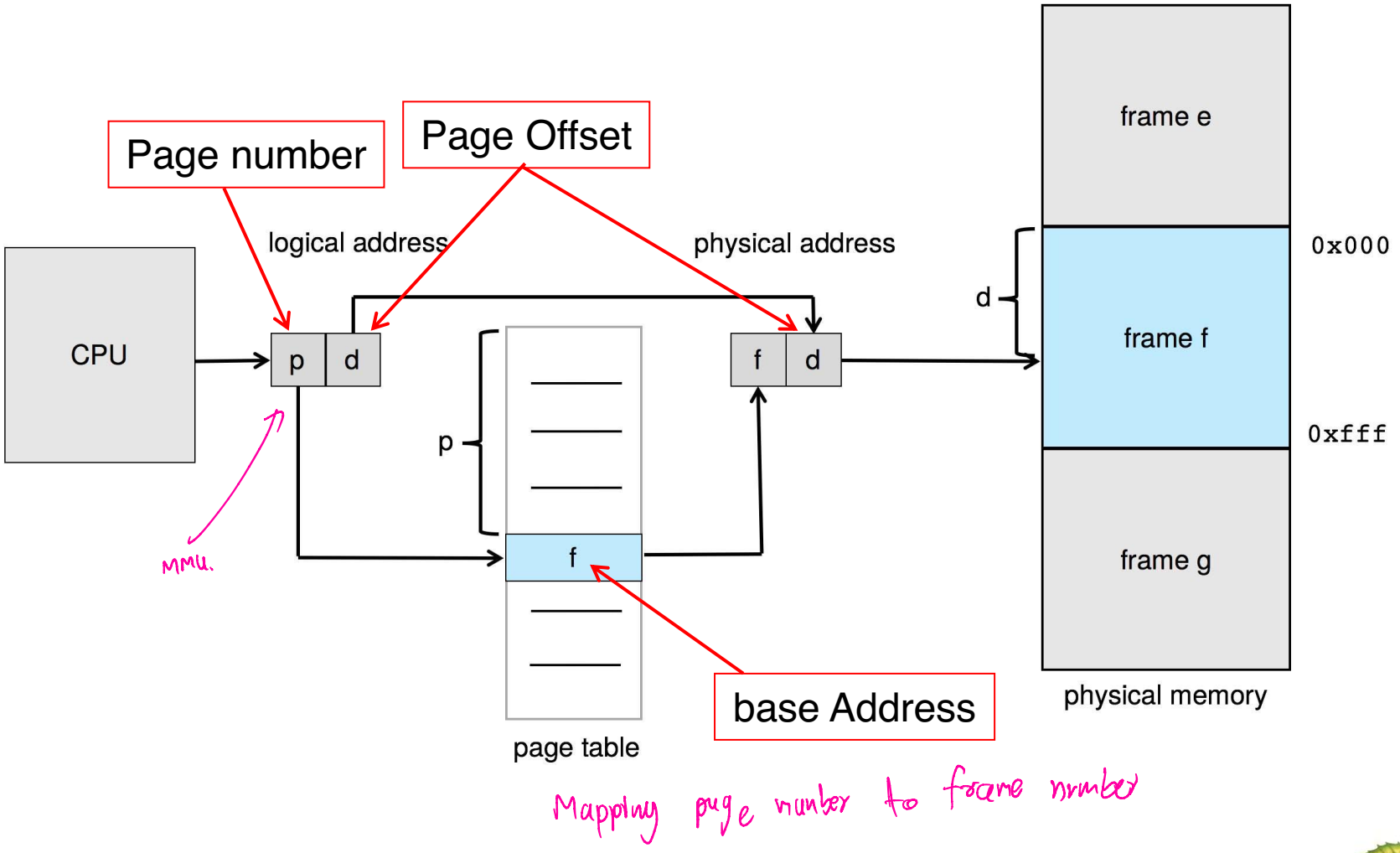
total number of bits of each logical address = (m - n) + n bits = m bits

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

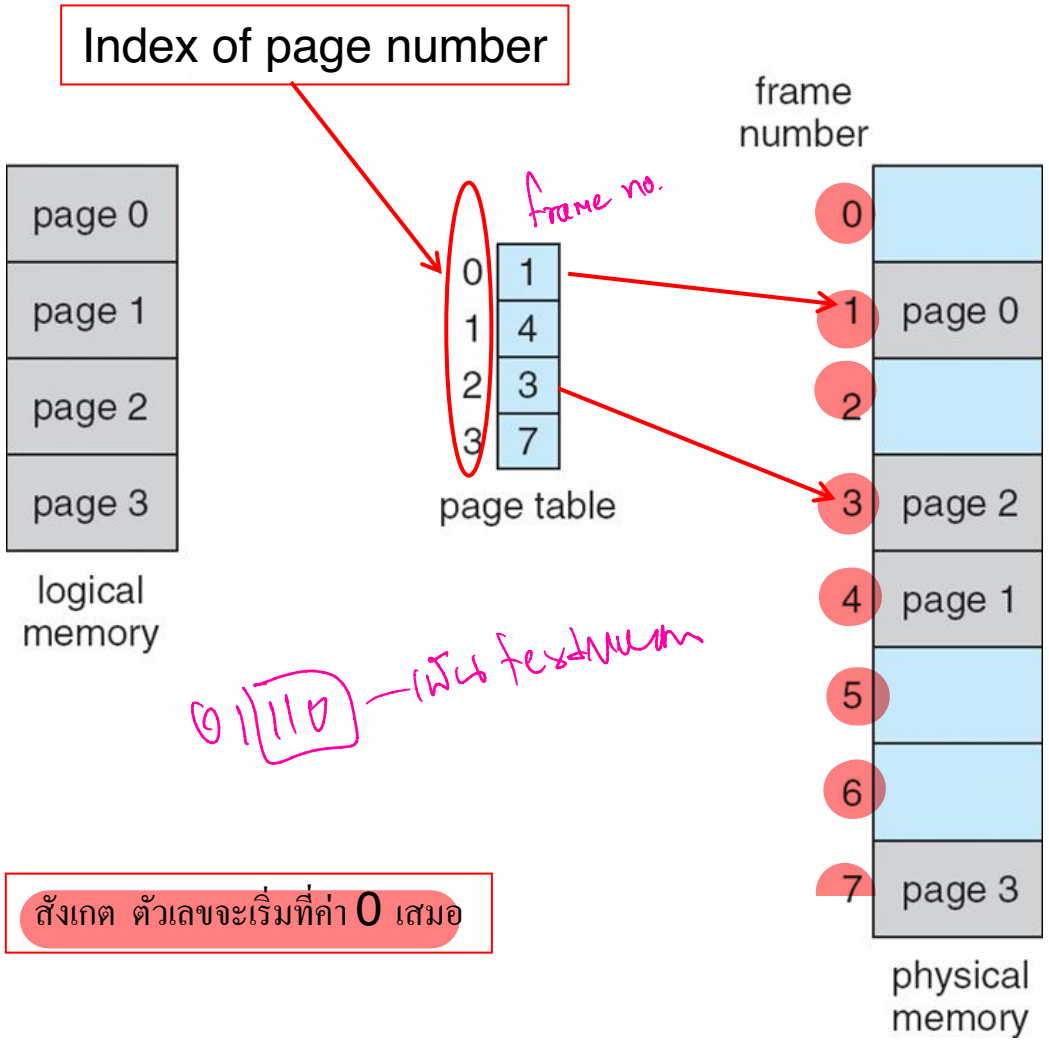
Since each address is m-bit long



Paging Hardware

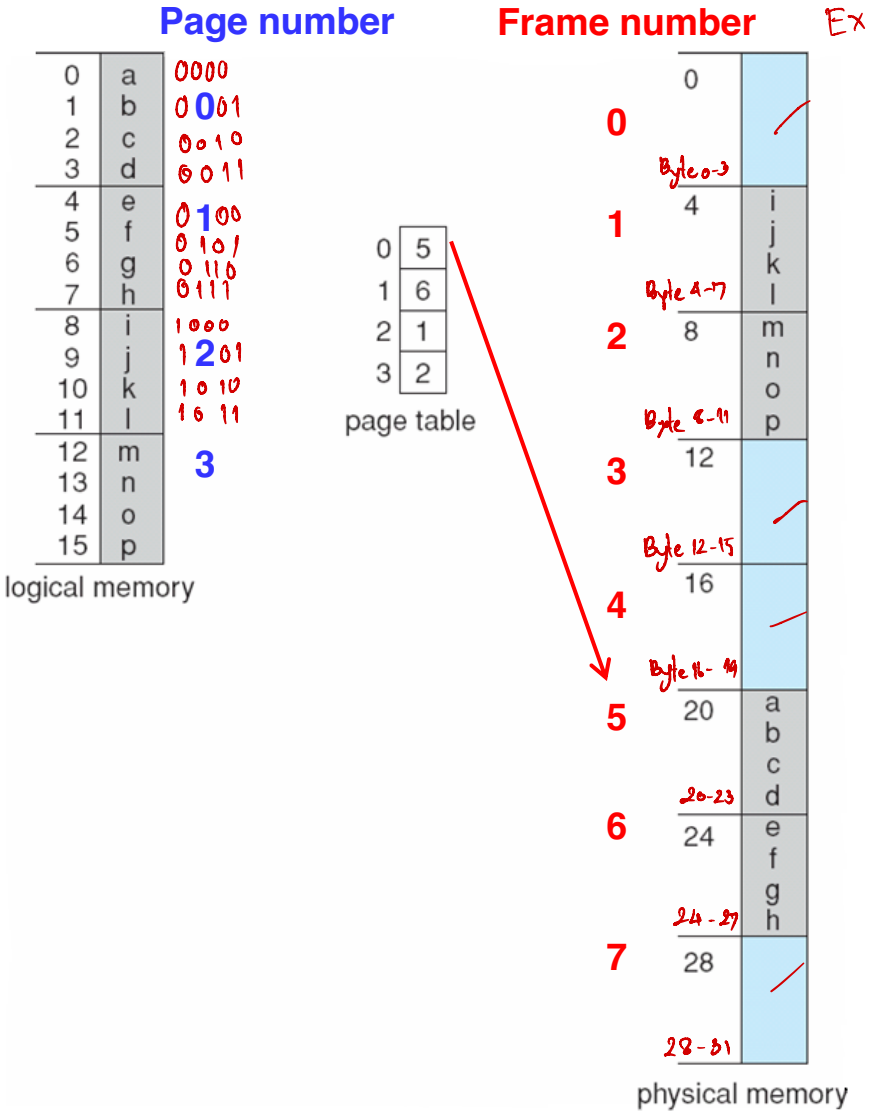


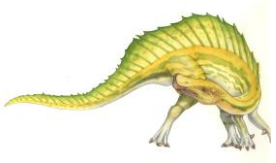
Paging Model of Logical and Physical Memory



Paging Example

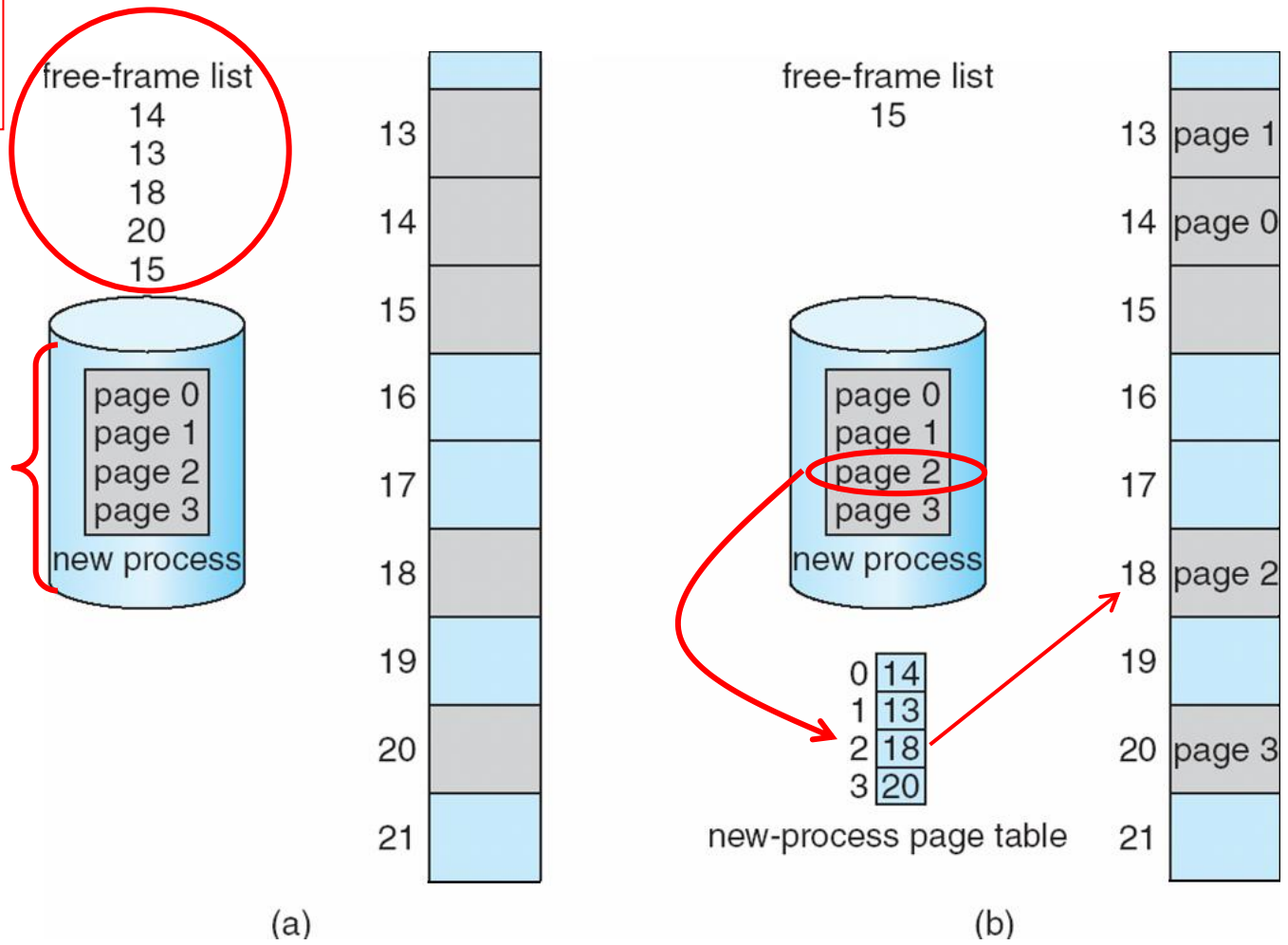
- Logical address: n = 2 and m = 4. Using a page size of 4 bytes and a physical memory of 32 bytes (8 pages)





Free Frames

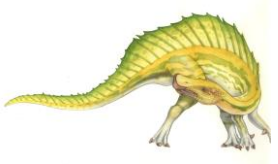
เรียงตามลำดับ
Free-frame
ที่ว่าง



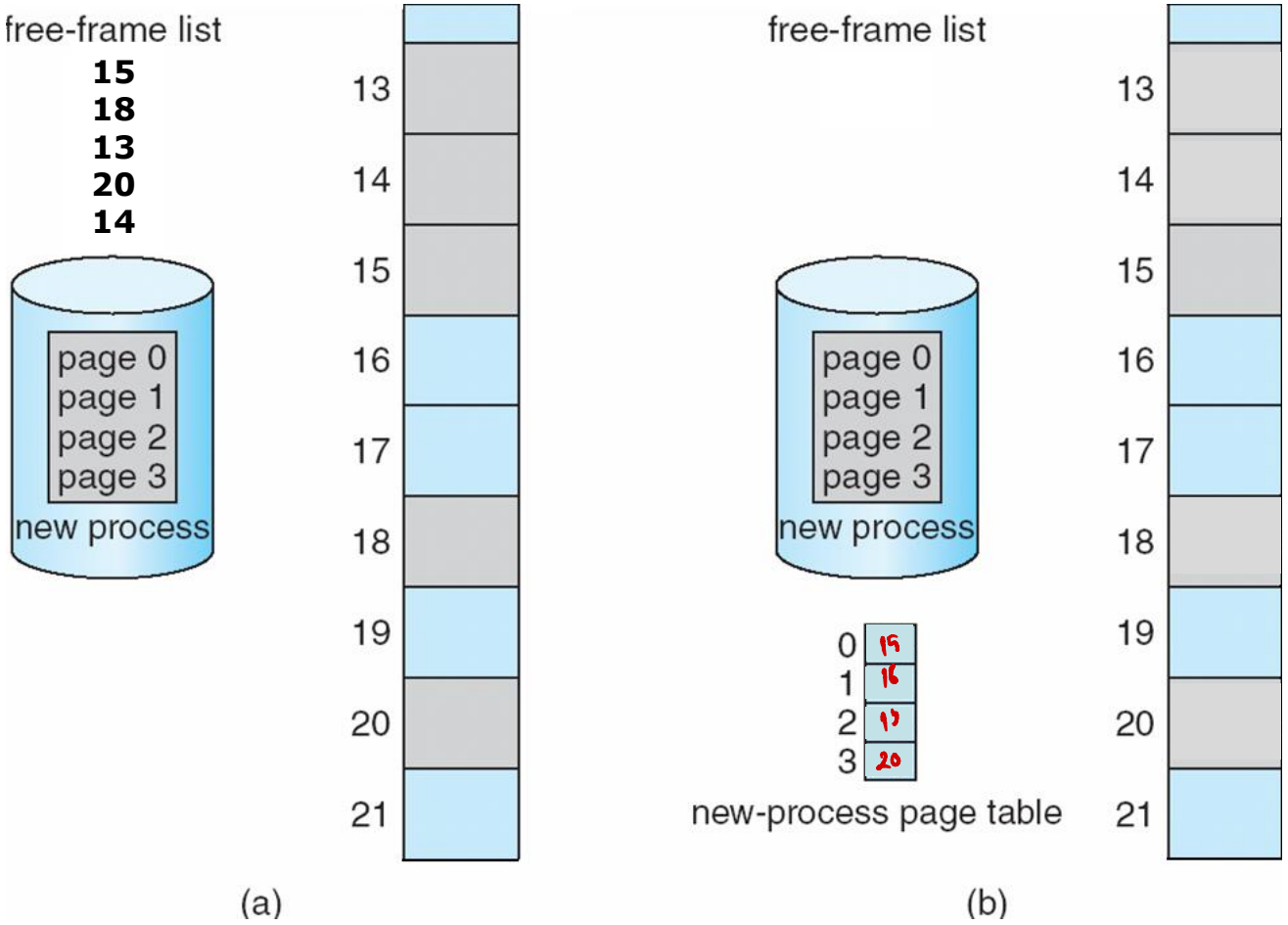
4 Pages

Before allocation

After allocation



Exercise: Free Frames

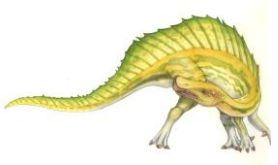


Before allocation

After allocation

Implementation of Page Table

- Page table is kept in main memory
- **Page-table base register (PTBR)** points to the page table
- **Page-table length register (PRLR)** 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)**
- Some TLBs store **address-space identifiers (ASIDs)** in each TLB entry – uniquely identifies each process to provide address-space protection for that process
  - TLBs typically small (64 to 1,024 entries)



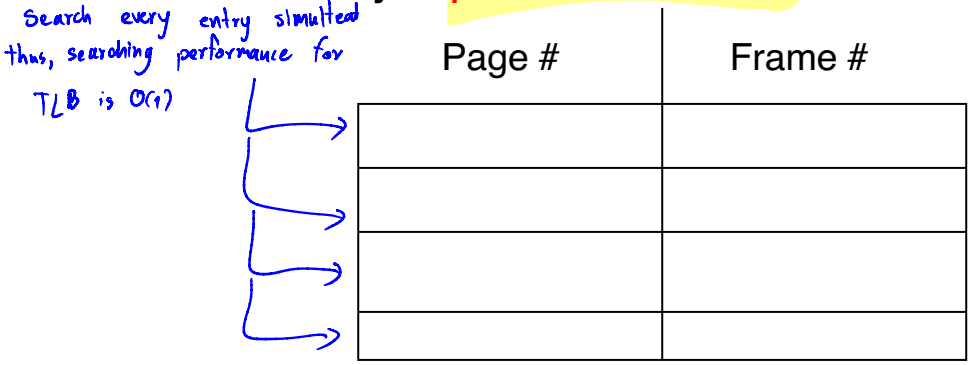
Associative Memory

Stored in a cache

Mapping Page # → Frame

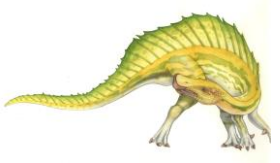
- Associative memory – parallel search

Search every entry simultaneously
thus, searching performance for TLB is O(1)

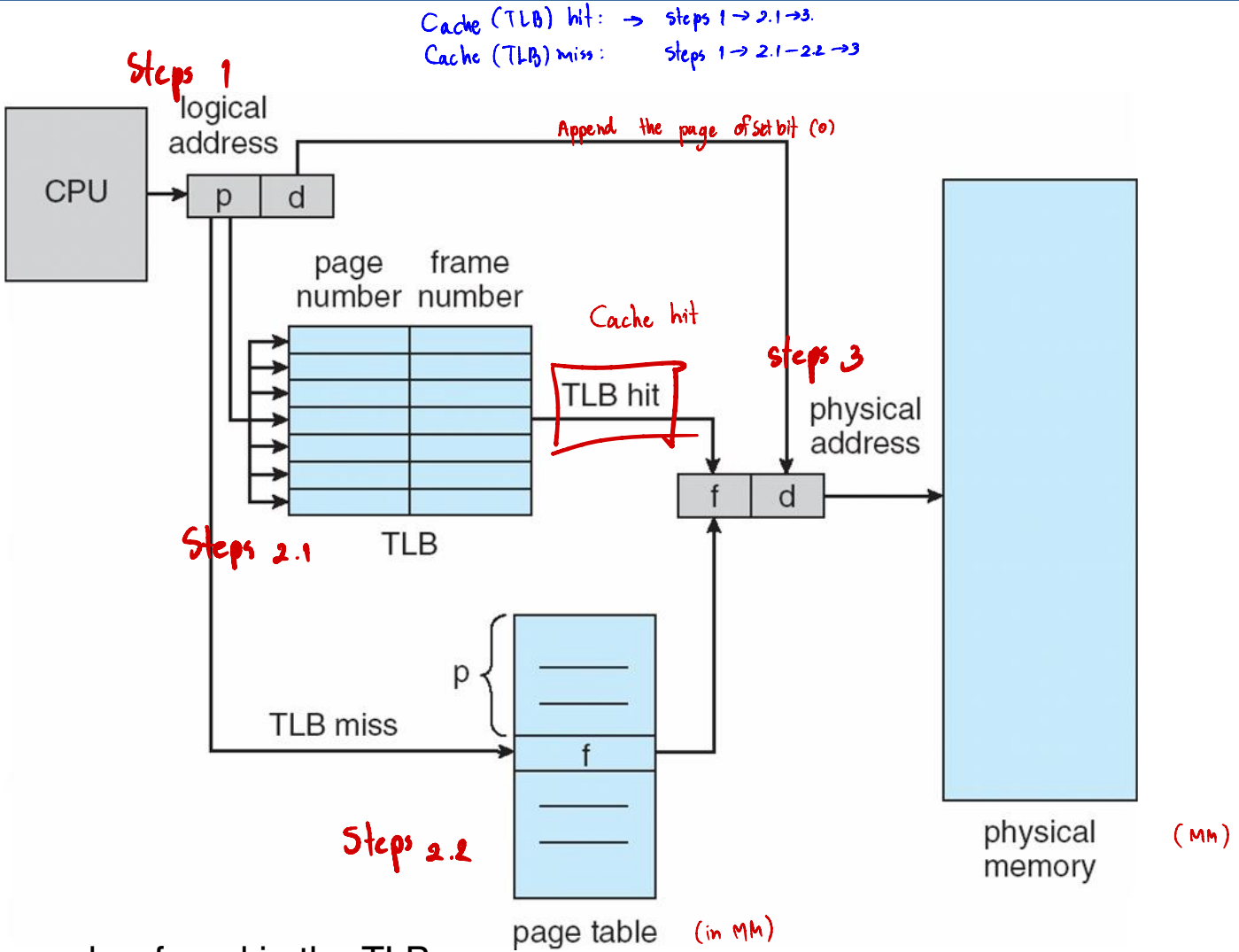


Address translation (p, d)

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



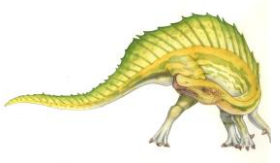
Paging Hardware With TLB



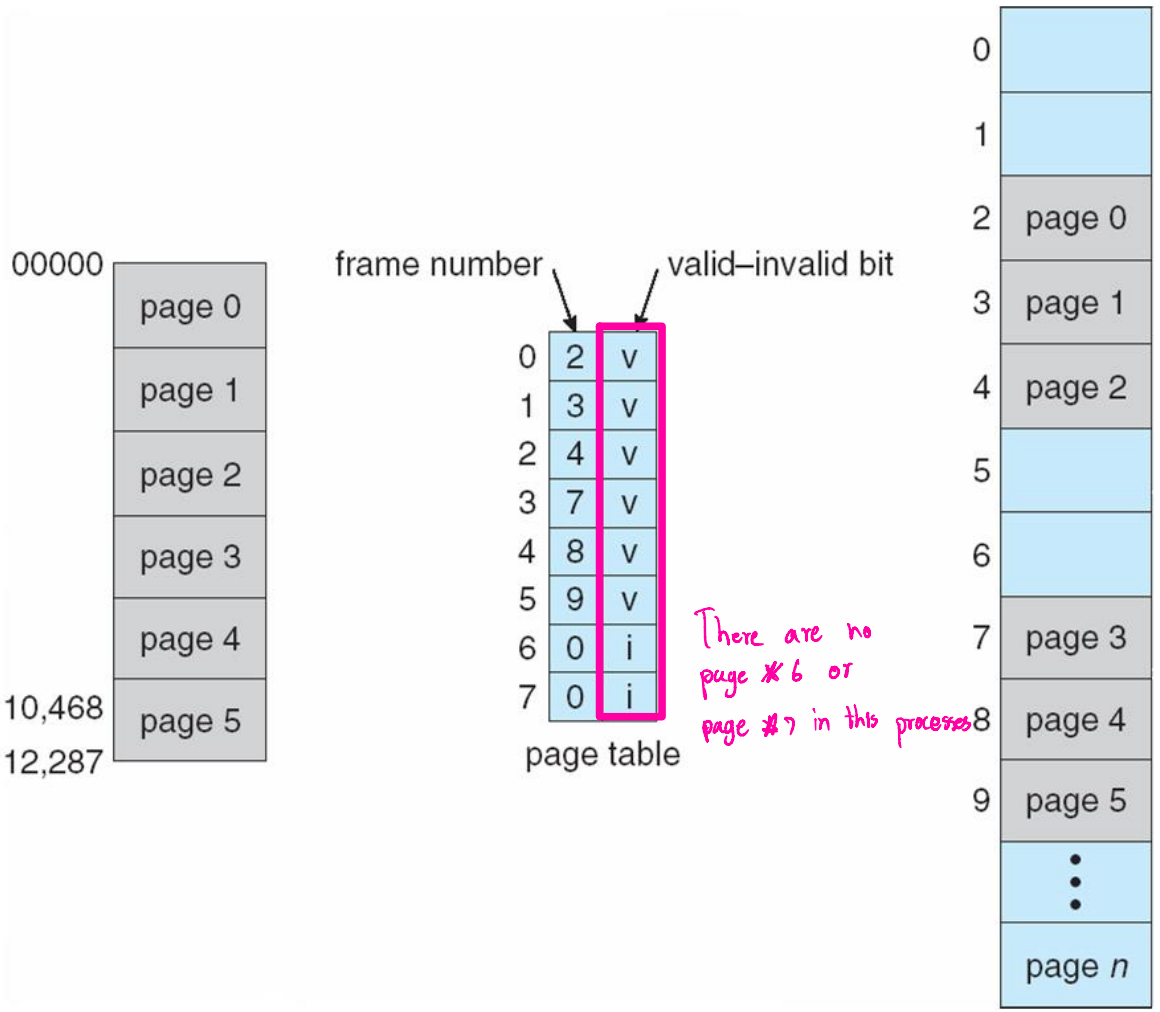
hit : page number found in the TLB
miss : page number not in the TLB

Memory Protection

- Memory protection implemented by associating protection bit with each frame
- **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
  - Or use **page-table length register (PTLR)**
- Any violations result in a trap to the kernel

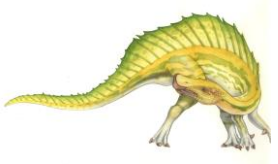


Valid (v) or Invalid (i) Bit In A Page Table

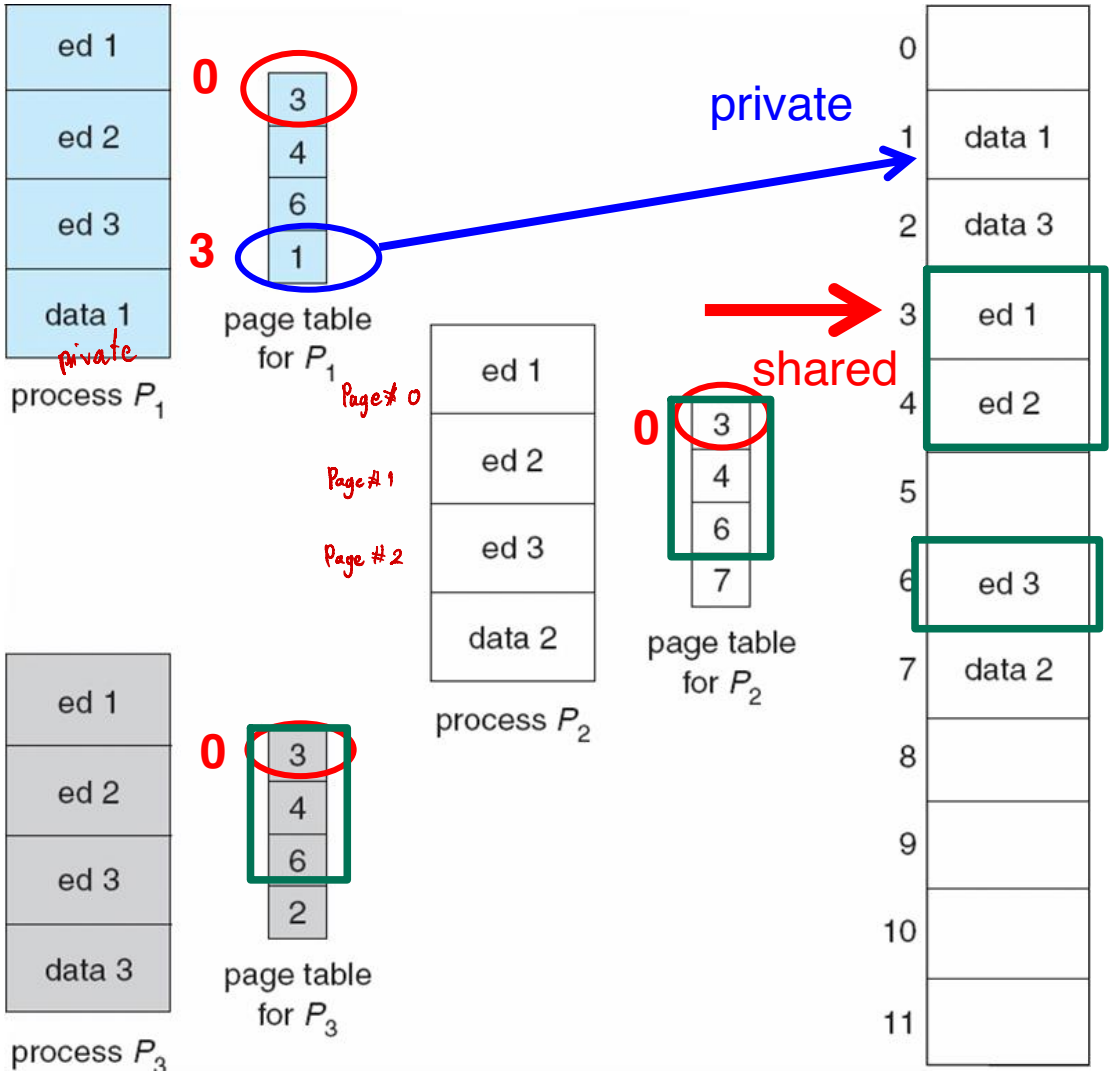


Shared Pages

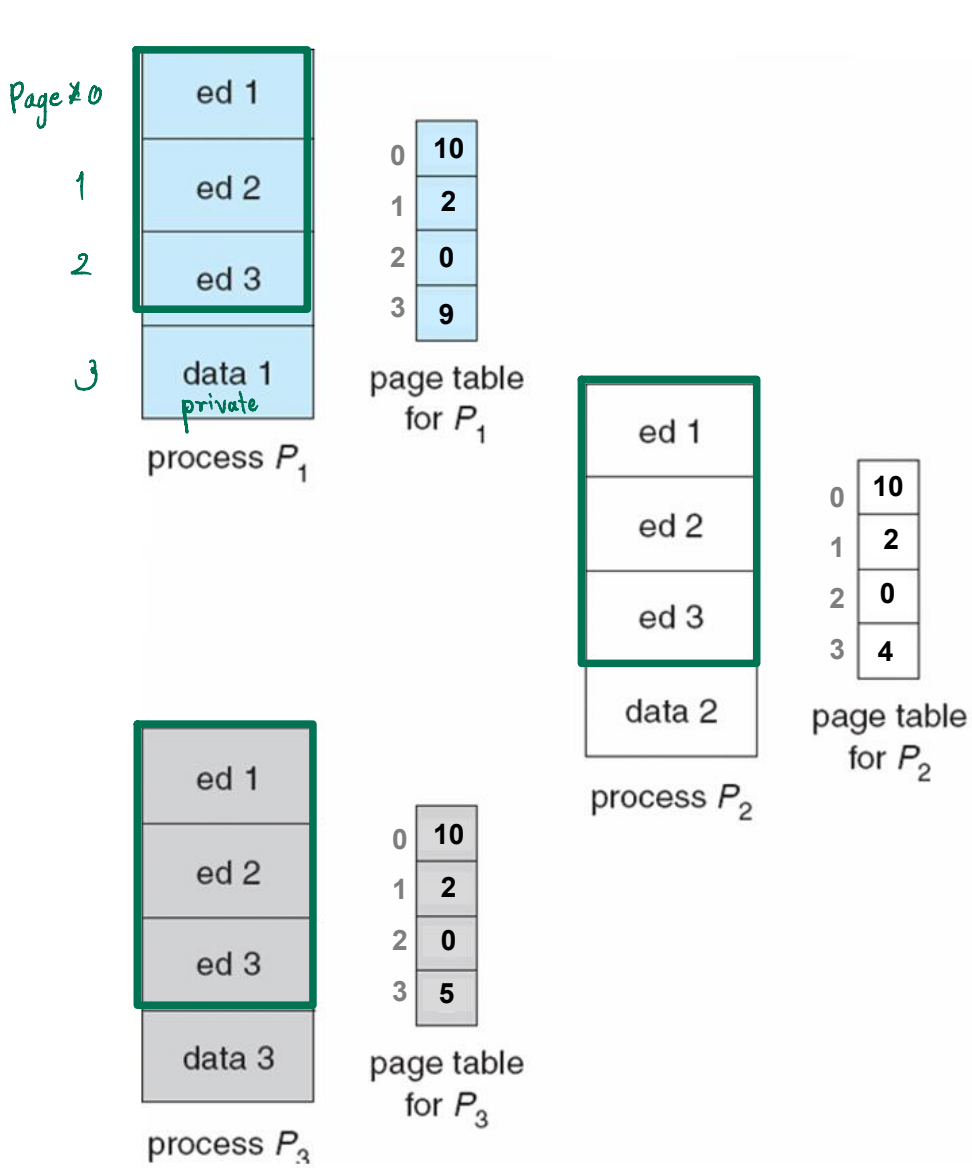
- **Shared code**
  - One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems).
  - Shared code must appear in same location in the logical address space of all processes
- **Private code and data**
  - Each process keeps a separate copy of the code and data
  - The pages for the private code and data can appear anywhere in the logical address space

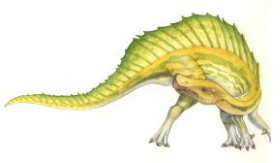


Shared Pages Example



Exercise: จงเติมข้อมูลในส่วนของ Frame ที่สัมพันธ์กับโพรเซสต่างๆ ที่ทำงานอยู่ในระบบที่มีการใช้งาน Shared Pages ด้วย





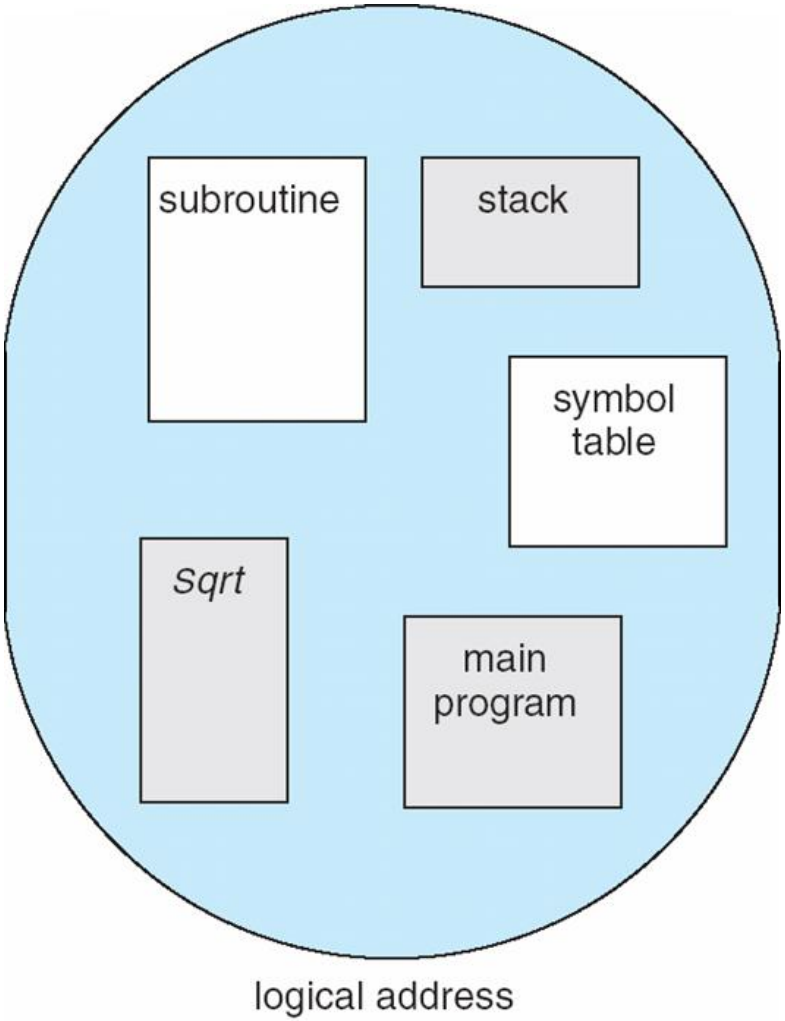
Segmentation (แบ่งเป็นตอน)

- Memory-management scheme that supports user view of memory
- A program is a collection of segments
  - A segment is a logical unit such as:

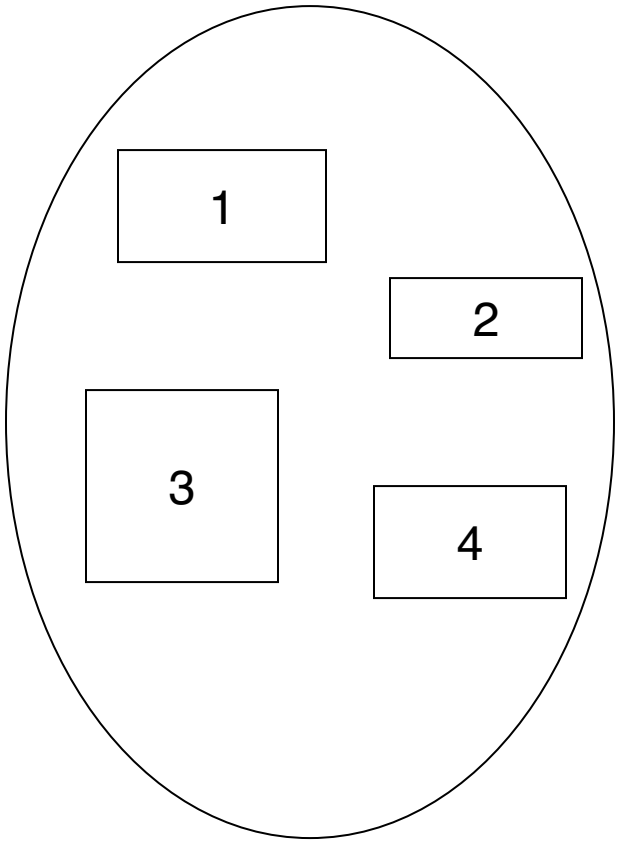
ส่วน ประกอบ ของ program

main program
procedure
function
method
object
local variables, global variables
common block
stack
symbol table
arrays

User's View of a Program



Logical View of Segmentation



user space

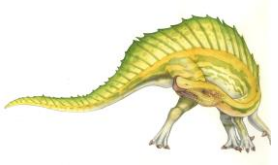
physical memory space

Segmentation Architecture

อ้างอิงความเข้าใจในเรื่อง paging

- Logical address consists of a two tuple:
  <segment-number, offset>,
  Part 1 (page #) Part 2 (page offset)
- **Segment table** – maps two-dimensional physical addresses; each table entry has:
  Page table
  - **base** – contains the starting physical address where the segments reside in memory
  - **limit** – specifies the length of the segment ขนาดของ segment
- **Segment-table base register (STBR)** points to the segment table's location in memory
- **Segment-table length register (STLR)** indicates number of segments used by a program;

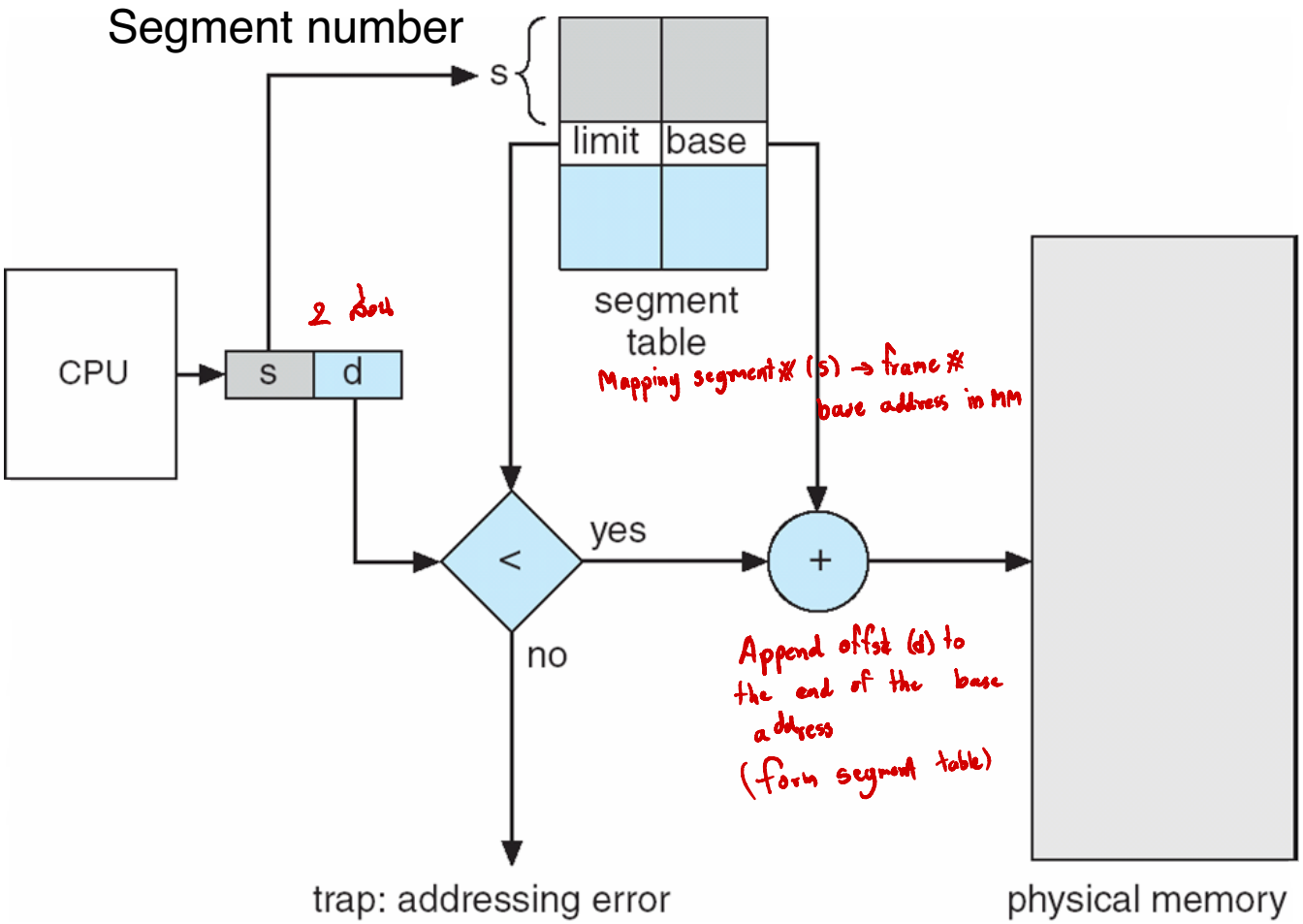
  segment number ***s*** is legal if ***s*** < **STLR**



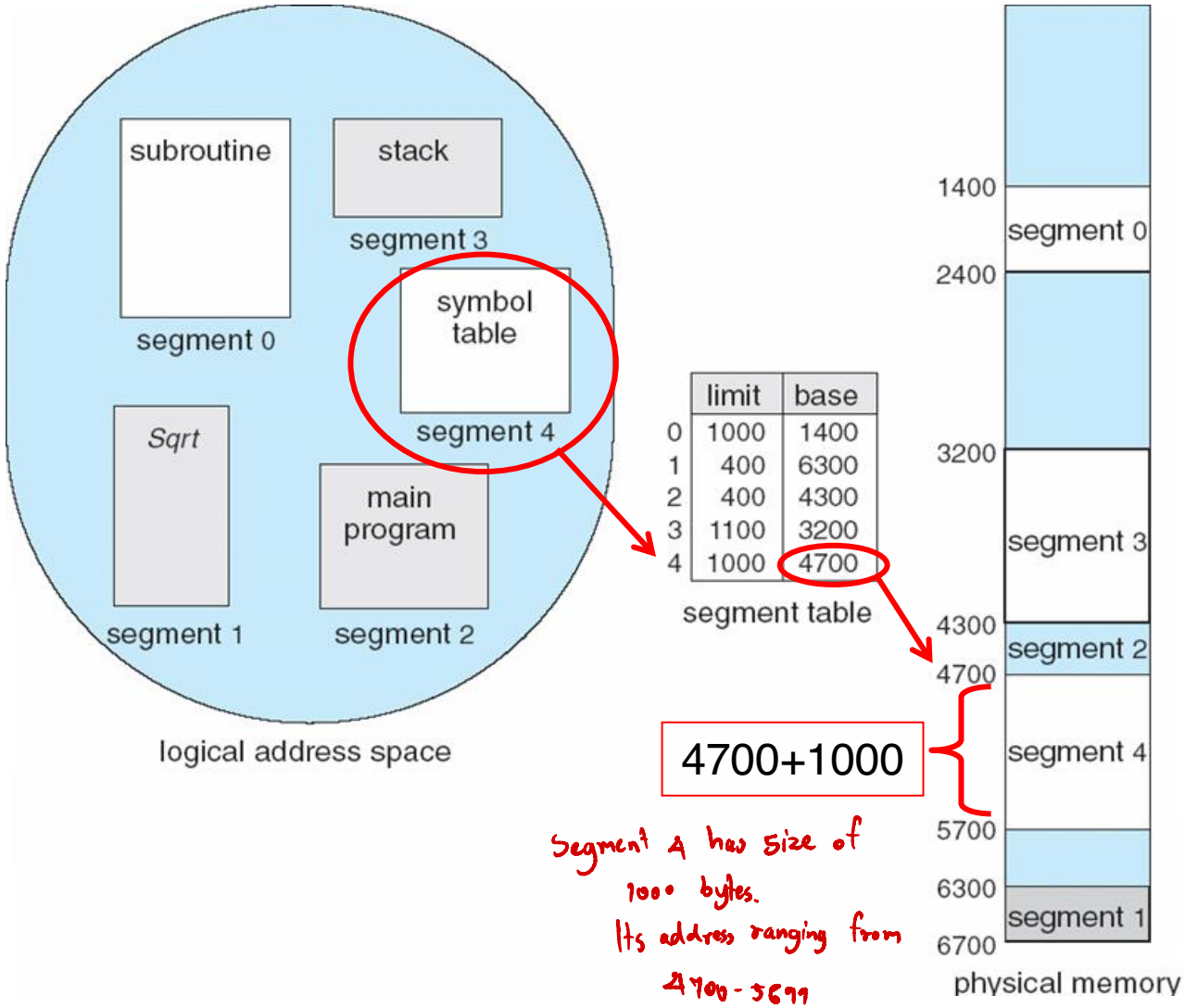
Segmentation Architecture (Cont.)

- 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
- A segmentation example is shown in the following diagram

Segmentation Hardware



Example of Segmentation



End of Chapter 7