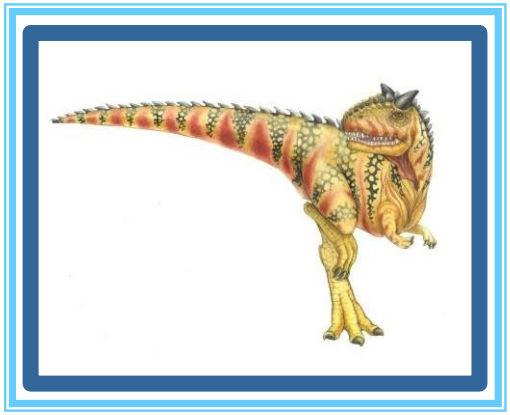
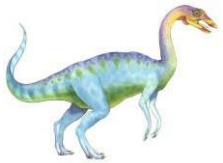
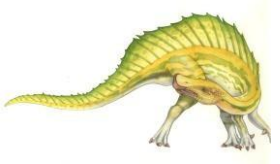
Chapter 7: Memory Management

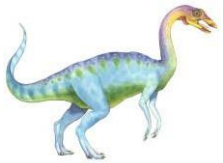




Chapter 7: Memory Management

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





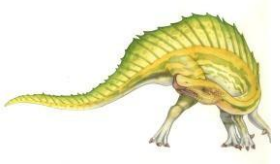
Objectives

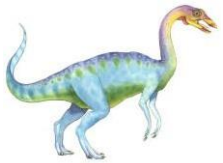
เพื่อให้คำอธิบายโดยละเอียดเกี่ยวกับวิธีการต่างๆ ในการจัดระเบียบฮาร์ดแวร์หน่วยความจำ

- To provide a detailed description of various ways of organizing memory hardware

เพื่อหารือเกี่ยวกับเทคนิคการจัดการหน่วยความจำต่างๆ รวมถึงการแบ่งหน้าและการแบ่งส่วน

- To discuss various memory-management techniques, including paging and segmentation





Background

program is passive

process is active

passive. active.

ต้องนำโปรแกรม (จากดิสก์) เข้าสู่หน่วยความจำและวางไว้ภายในกระบวนการจึงจะรันได้

- Program must be brought (from disk) into memory and placed within a process for it to be run

หน่วยความจำหลักและรีจิสเตอร์เป็นเพียงหน่วยเก็บข้อมูล CPU เท่านั้นที่สามารถเข้าถึงได้โดยตรง

- Main memory and registers are only storage CPU can access directly

ลงทะเบียนการเข้าถึงในหนึ่งนาฬิกา CPU (หรือน้อยกว่า)

- Register access in one CPU clock (or less)

หน่วยความจำหลักอาจใช้เวลาหลายรอบ

- Main memory can take many cycles of the CPU clock

แคชอยู่ระหว่างหน่วยความจำหลักและการลงทะเบียน CPU

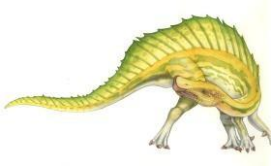
- **Cache** sits between main memory and CPU registers

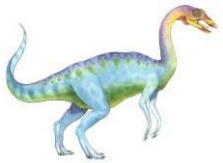
การป้องกันหน่วยความจำที่จำเป็นเพื่อให้แน่ใจว่าการทำงานถูกต้อง

- Protection of memory required to ensure correct operation

Having a cache built the CPU. the H/W automatically speeds up memory access without any OS control

brought : ถูกนำพา

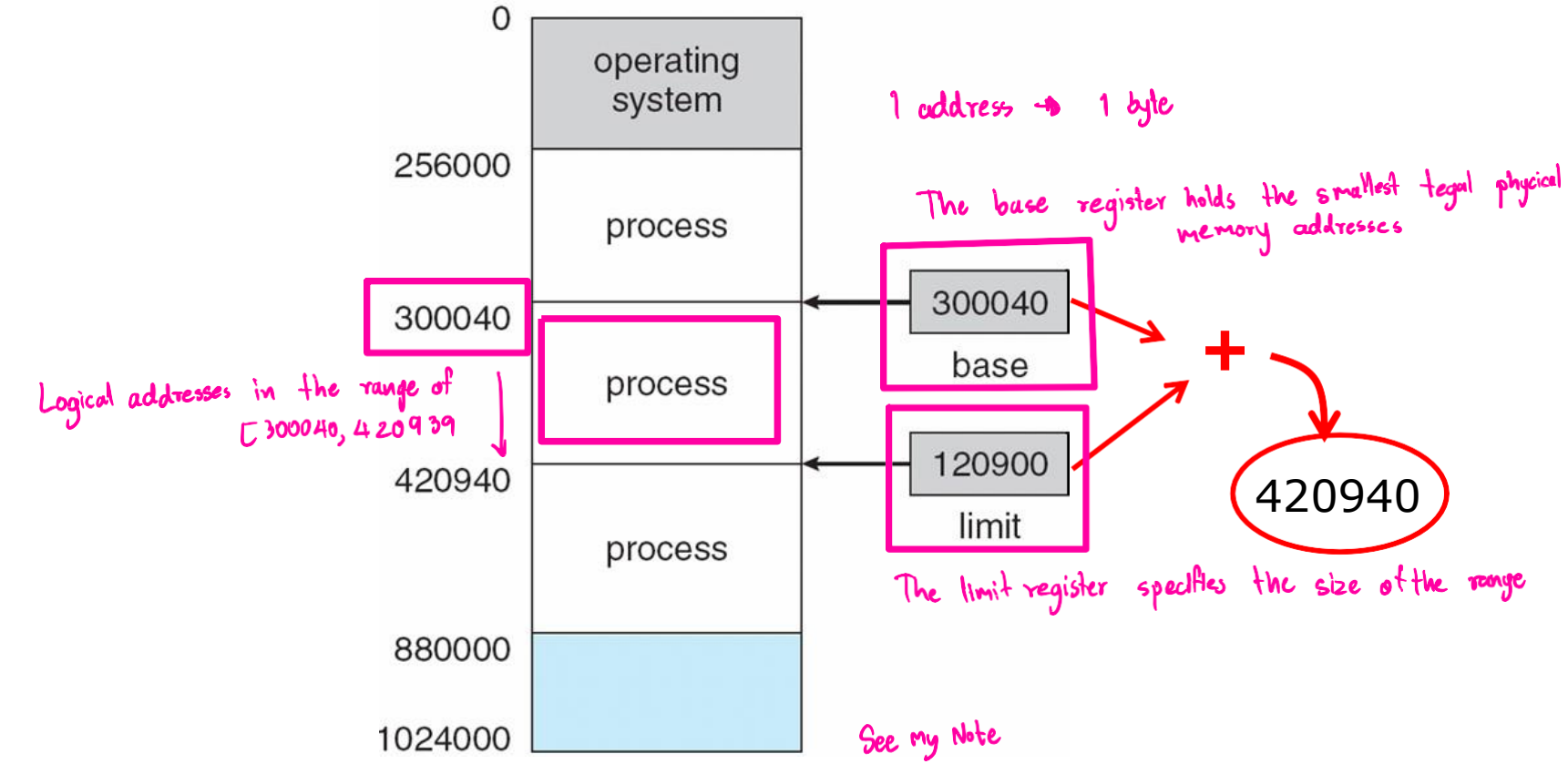


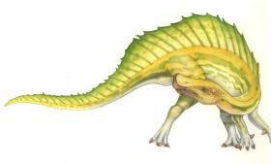


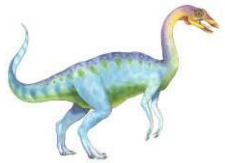
Base and Limit Registers

คู่ของการลงทะเบียนฐานและขีดจำกัดจะกำหนดพื้นที่ที่อยู่แบบลอจิคัล

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







การเชื่อมโยงคำสั่งและข้อมูลเข้ากับหน่วยความจำ

Binding of Instructions and Data to Memory

การเชื่อมโยงที่อยู่ของคำสั่งและข้อมูลไปยังที่อยู่หน่วยความจำสามารถเกิดขึ้นได้ในสามขั้นตอนที่แตกต่างกัน

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

The binding can be done at any step along the way

Step1

- **Compile time**: If memory location known a priori, **absolute code** can be generated; must recompile code if starting location changes

If you know at compile time where the process will reside in memory the absolute code can be generated.

If, at some (the) time, the starting location changes, then it will be necessary to recompile this code

เวลาในการโหลด: ต้องสร้างโค้ดที่ย้ายตำแหน่งได้หากไม่ทราบตำแหน่งหน่วยความจำในขณะคอมไพล์

Step2

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

Step3

- **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)

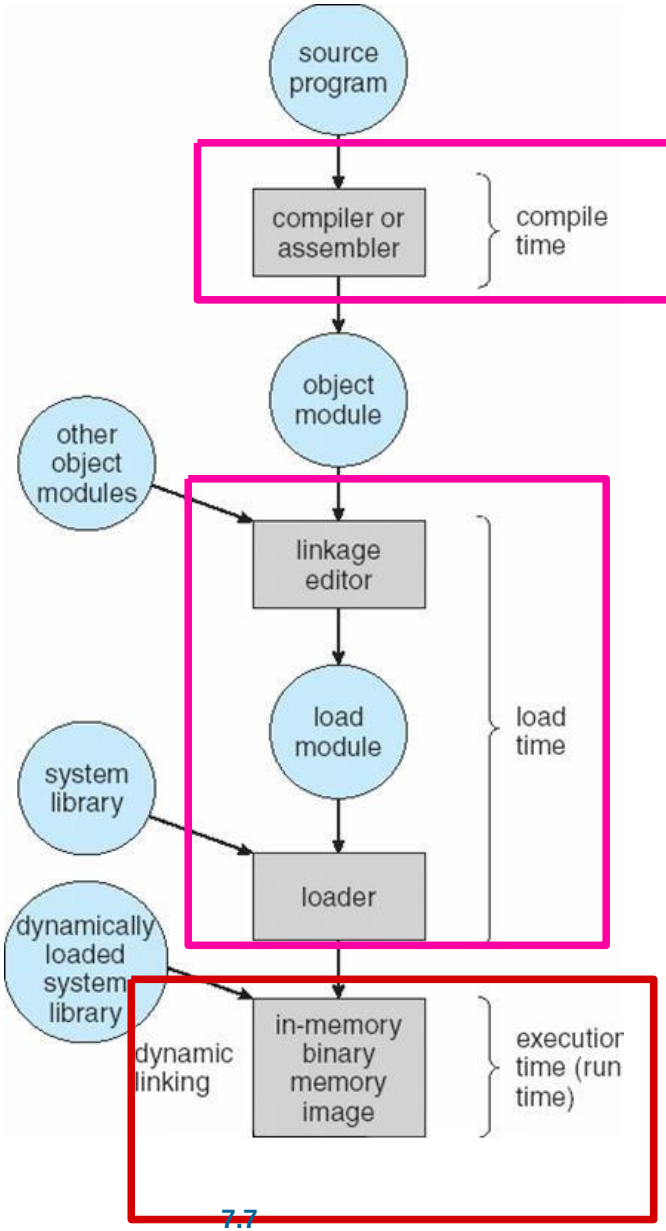
เวลาในการคอมไพล์: หากตำแหน่งหน่วยความจำรู้จักนิรนัย สามารถสร้างโค้ดที่สมบูรณ์ได้ ต้องคอมไพล์โค้ดใหม่หากตำแหน่งเริ่มต้นเปลี่ยนแปลง

เวลาดำเนินการ: การเชื่อมโยงจะล่าช้าไปจนถึงเวลารันหากกระบวนการสามารถย้ายระหว่างการดำเนินการจากเซ็กเมนต์หน่วยความจำหนึ่งไปยังอีกส่วนหนึ่งได้ ต้องการการสนับสนุนด้านฮาร์ดแวร์สำหรับการแมปที่อยู่ (เช่น การลงทะเบียนฐานและขีดจำกัด)

Binding: การกำหนดค่า

การประมวลผลหลายขั้นตอนของโปรแกรมผู้ใช้

Multistep Processing of a User Program



main.c

gcc -c main.c

generates

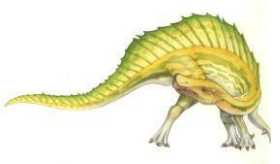
main.o (object file)

gcc -o main main.o -lm

generates

main (execute file)

./main



พื้นที่ที่อยู่แบบลอจิคัลกับฟิสิคัล

Logical vs. Physical Address Space

แนวคิดของพื้นที่ที่อยู่แบบลอจิคัลที่ถูกผูกไว้กับพื้นที่ที่อยู่ทางกายภาพที่แยกต่างหากเป็นศูนย์กลางของการจัดการหน่วยความจำที่เหมาะสม

- The concept of a logical address space that is bound to a separate physical address space is central to proper memory management

  ที่อยู่แบบลอจิคัล – สร้างโดย CPU; เรียกอีกอย่างว่าที่อยู่เสมือน

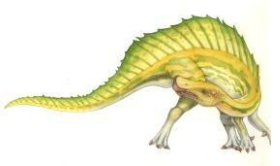
  - Logical address – generated by the CPU; also referred to as virtual address

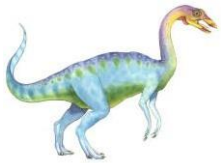
  ที่อยู่ทางกายภาพ - ที่อยู่ที่หน่วยหน่วยความจำเห็น

  - 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

ที่อยู่แบบลอจิคัลและฟิสิคัลจะเหมือนกันในรูปแบบการผูกที่อยู่เวลาคอมไพล์และเวลาในการโหลด ที่อยู่แบบลอจิคัล (เสมือน) และฟิสิคัลแตกต่างกันในรูปแบบการผูกที่อยู่เวลาดำเนินการ





หน่วยการจัดการหน่วยความจำ

Memory-Management Unit (MMU)

Logical Address

อุปกรณ์ฮาร์ดแวร์ที่แมปเสมือนกับที่อยู่จริง

- Hardware device that maps virtual to physical address

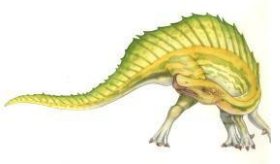
ในรูปแบบ MMU ค่าในทะเบียนการย้ายตำแหน่งจะถูกเพิ่มไปยังทุกที่อยู่ที่สร้างโดยกระบวนการผู้ใช้ ณ เวลาที่ถูกส่งไปยังหน่วยความจำ

- 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

โปรแกรมผู้ใช้เกี่ยวข้องกับที่อยู่แบบลอจิคัล มันไม่เคยเห็นที่อยู่ทางกายภาพที่แท้จริง

- 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

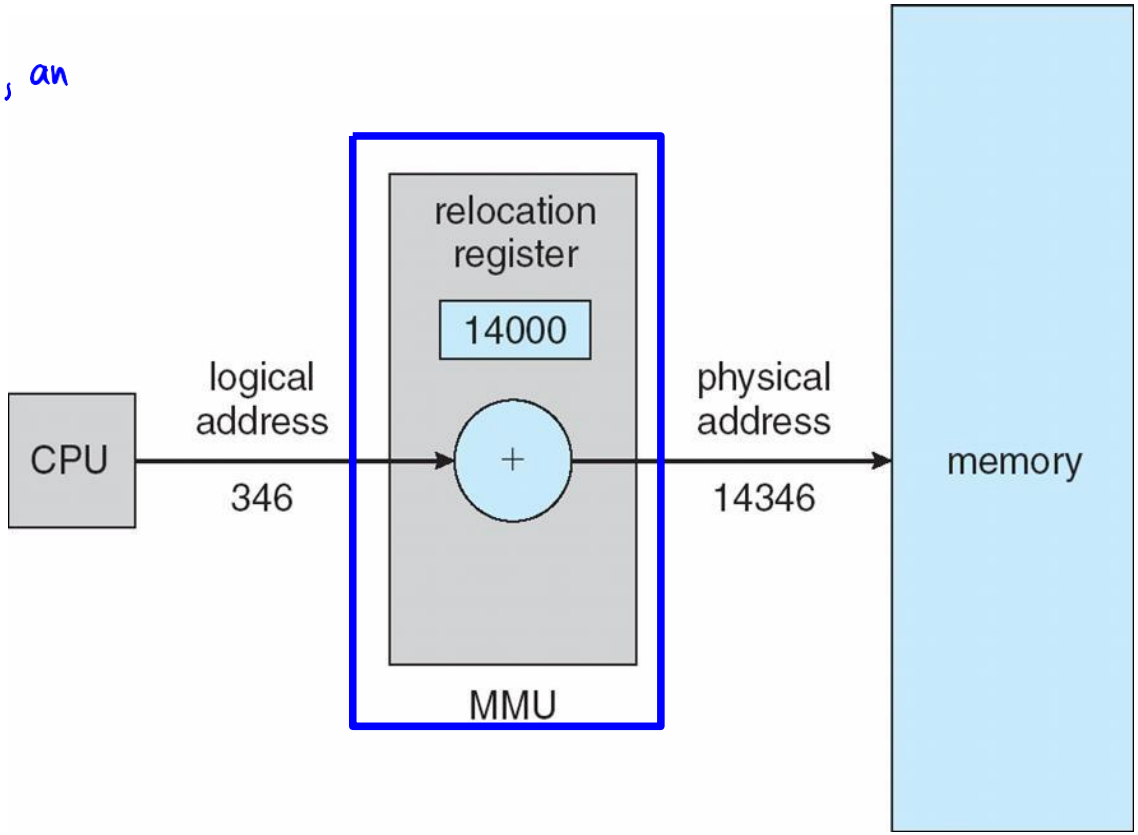


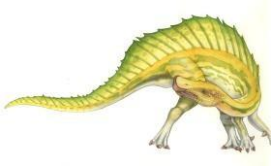
การย้ายตำแหน่งแบบไดนามิกโดยใช้ทะเบียนการย้ายตำแหน่ง

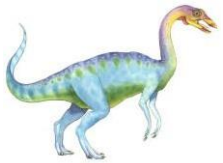
Dynamic relocation using a relocation register

Run-time or Execution-time binding

thus, an







Dynamic Loading

รูทีนจะไม่โหลดจนกว่าจะถูกเรียก

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

- Routine is not loaded until it is called

การใช้พื้นที่หน่วยความจำดีขึ้น รูทีนที่ไม่ได้ใช้จะไม่ถูกโหลด

- Better memory-space utilization; unused routine is never loaded

มีประโยชน์เมื่อจำเป็นต้องใช้โค้ดจำนวนมากเพื่อจัดการกับกรณีที่ไม่ค่อยเกิดขึ้น

- Useful when large amounts of code are needed to handle infrequently occurring cases

Ex Error routines.

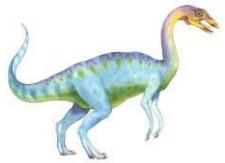
ไม่จำเป็นต้องมีการสนับสนุนพิเศษจากระบบปฏิบัติการผ่านการออกแบบโปรแกรม

- No special support from the operating system is required implemented through program design

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

การเชื่อมโยงถูกเลื่อนออกไปจนกว่าจะถึงเวลาดำเนินการ

- Linking postponed until execution time

โค้ดชิ้นเล็กๆ stub ใช้เพื่อค้นหารูทีนไลบรารีในหน่วยความจำที่เหมาะสม

- Small piece of code, *stub*, used to locate the appropriate memory-resident library routine

Stub จะแทนที่ตัวเองด้วยที่อยู่ของรูทีน และดำเนินการรูทีน

- Stub replaces itself with the address of the routine, and executes the routine

Use the routine address to link to routien code for execution

ระบบปฏิบัติการที่จำเป็นในการตรวจสอบว่ารูทีนอยู่ในที่อยู่หน่วยความจำของกระบวนการหรือไม่

- Operating system needed to check if routine is in processes' memory address

การเชื่อมโยงแบบไดนามิกมีประโยชน์อย่างยิ่งสำหรับไลบรารี

- Dynamic linking is particularly useful for libraries

ระบบเรียกอีกอย่างว่าไลบรารีแบบแบ่งใช้

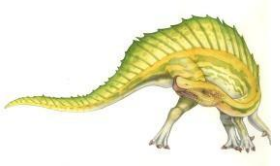
- System also known as **shared libraries**

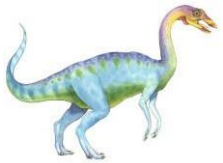
Dynamic linking (compared to static linking)

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

กระบวนการสามารถสลับจากหน่วยความจำชั่วคราวไปเป็นที่เก็บสำรอง จากนั้นนำกลับเข้าสู่หน่วยความจำเพื่อดำเนินการต่อไป

- A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution

Backing store – ดิสก์ที่รวดเร็วมีขนาดใหญ่พอที่จะรองรับสำเนาของอิมเมจหน่วยความจำทั้งหมดสำหรับผู้ใช้ทุกคน จะต้องให้การเข้าถึงโดยตรงไปยังอิมเมจหน่วยความจำเหล่านี้

- **Backing store** – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images

เปิดตัว ม้วนเข้า – ตัวแปรการสลับที่ใช้สำหรับอัลกอริธึมการตั้งเวลาตามลำดับความสำคัญ กระบวนการที่มีลำดับความสำคัญต่ำกว่าจะถูกสลับออกเพื่อให้สามารถโหลดและดำเนินการกระบวนการที่มีลำดับความสำคัญสูงกว่าได้

- **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

เวลาสวอปส่วนใหญ่คือเวลาโอน เวลาในการถ่ายโอนทั้งหมดเป็นสัดส่วนโดยตรงกับจำนวนหน่วยความจำที่สลับ

- Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped

ระบบจะรักษาคิวที่พร้อมของกระบวนการที่พร้อมใช้งานซึ่งมีอิมเมจหน่วยความจำบนดิสก์

- System maintains a **ready queue** of ready-to-run processes which have memory images on disk

พบการสลับเวอร์ชันที่แก้ไขแล้วในหลายระบบ (เช่น UNIX, Linux และ Windows)

- Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)

การแลกเปลี่ยนถูกปิดใช้งานตามปกติ

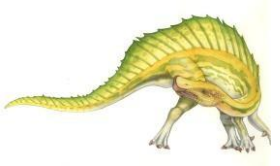
  - Swapping normally disabled

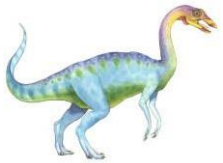
เริ่มต้นหากจัดสรรหน่วยความจำเกินขีดจำกัด

  - Started if more than threshold amount of memory allocated

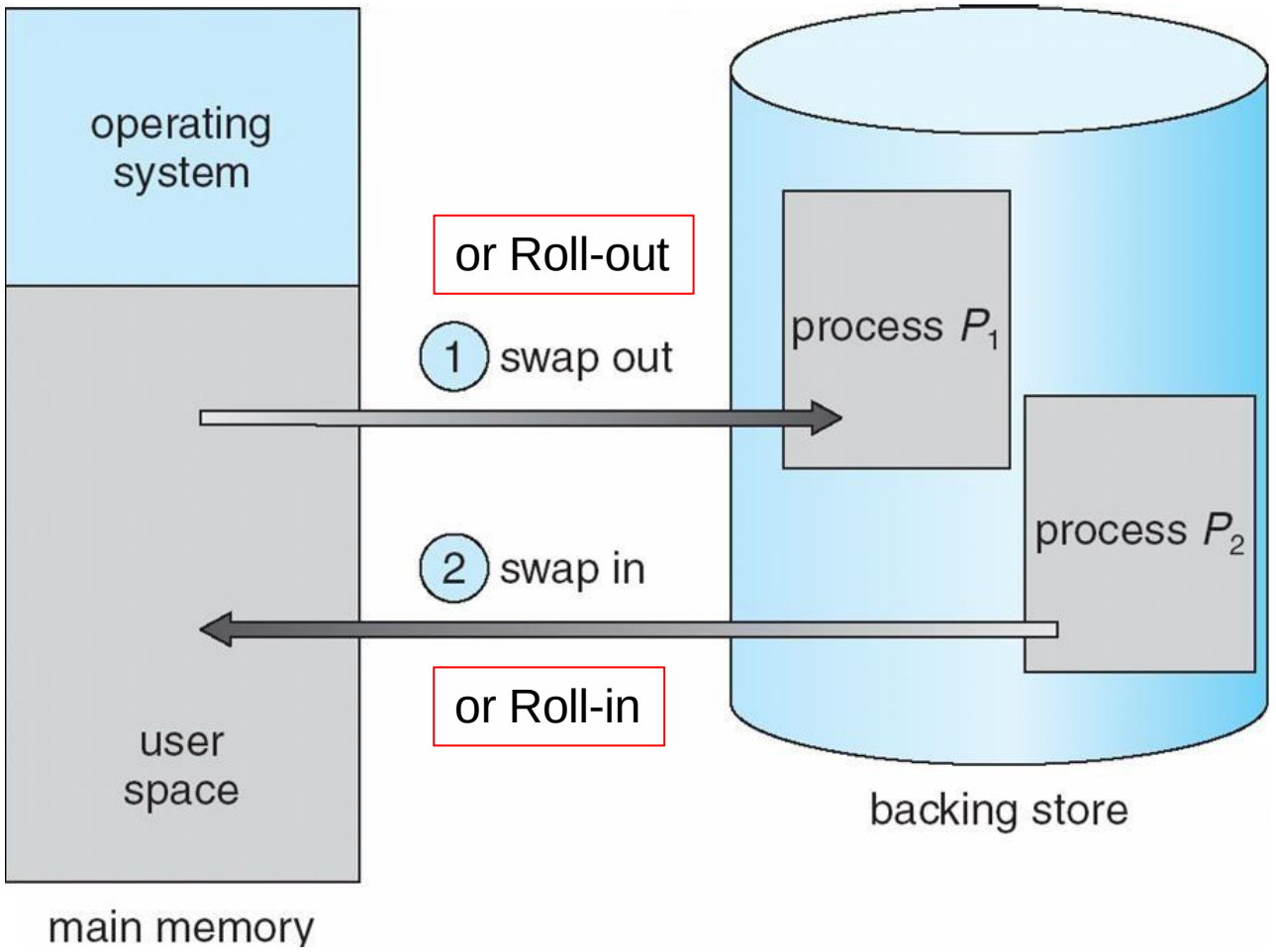
ปิดใช้งานอีกครั้งเมื่อความต้องการหน่วยความจำลดลงต่ำกว่าเกณฑ์

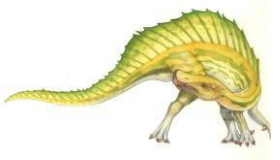
  - Disabled again once memory demand reduced below threshold

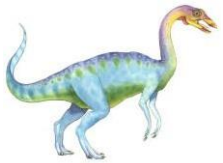




Schematic View of Swapping







Contiguous Allocation

(การจัดสรรพื้นที่ที่ติดกัน)

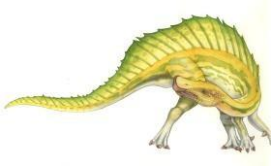
หน่วยความจำหลักมักจะแบ่งออกเป็นสองพาร์ติชัน:

- Main memory usually into two partitions:
  - ระบบปฏิบัติการประจำเครื่อง มักจะเก็บไว้ในหน่วยความจำเหลือน้อยและมีเวกเตอร์ขัดจังหวะ
    - Resident operating system, usually held in low memory with interrupt vector (โปรแกรมของระบบปฏิบัติการเอง)
  - กระบวนการของผู้ใช้นั้นถูกเก็บไว้ในหน่วยความจำสูง
    - User processes then held in high memory

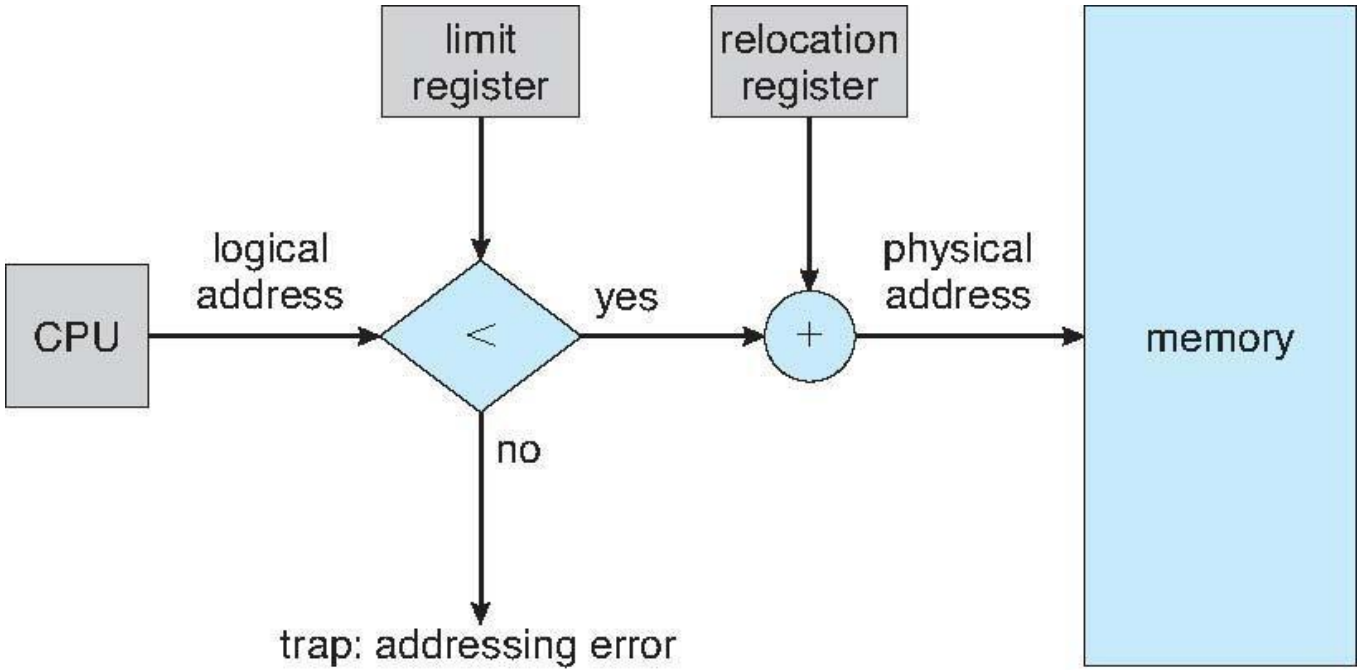
การลงทะเบียนการย้ายตำแหน่งที่ใช้เพื่อปกป้องกระบวนการของผู้ใช้จากกันและกัน และจากการเปลี่ยนแปลงรหัสและข้อมูลระบบปฏิบัติการ

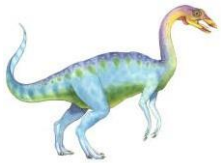
- Relocation registers used to protect user processes from each other, and from changing operating-system code and data
  - การลงทะเบียนฐานประกอบด้วยค่าที่อยู่ทางกายภาพที่เล็กที่สุด
    - Base register contains value of smallest physical address
  - การลงทะเบียนแบบจำกัดประกอบด้วยช่วงของที่อยู่แบบลอจิคัล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

contiguous : ติดกัน



Hardware Support for Relocation and Limit Registers





การจัดสรรต่อเนื่อง

Contiguous Allocation (Cont)

การจัดสรรหลายพาร์ติชั่น

- Multiple-partition allocation (การจัดสรรเนื้อที่แบบหลายส่วน)

ระดับของการเขียนโปรแกรมหลายโปรแกรมถูกจำกัดด้วยจำนวนพาร์ติชัน

  - Degree of multiprogramming limited by number of partitions

ขนาดพาร์ติชั่นแบบแปรผันเพื่อประสิทธิภาพ (ขนาดตามความต้องการของกระบวนการที่กำหนด)

  - **Variable-partition** sizes for efficiency (sized to a given process' needs)

Hole – บล็อกของหน่วยความจำที่มีอยู่ รูขนาดต่างๆ กระจัดกระจายไปทั่วหน่วยความจำ

  - **Hole** – block of available memory; holes of various size are scattered throughout memory

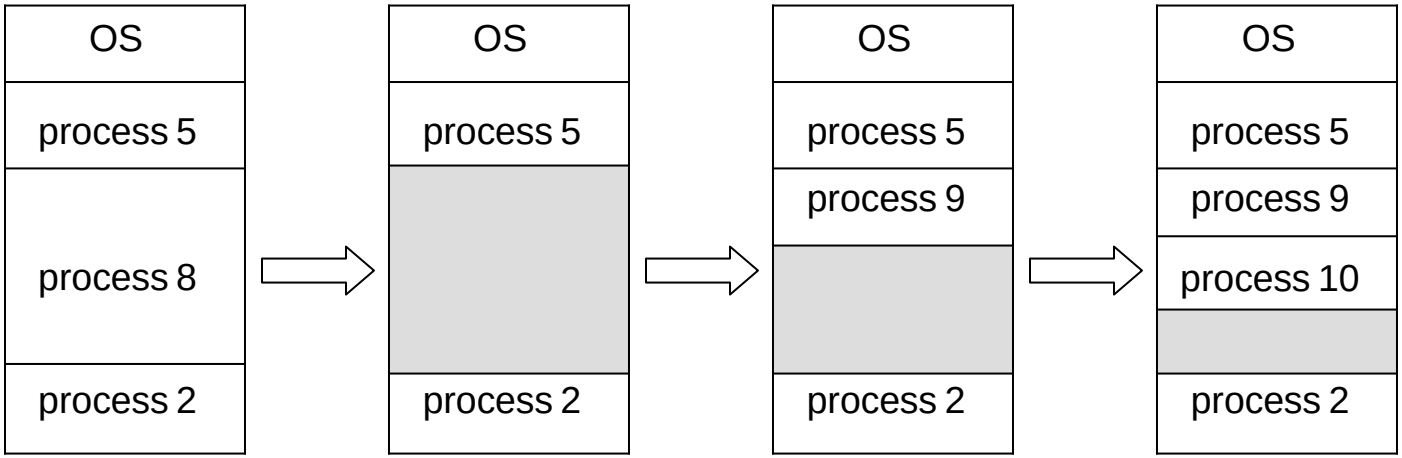
เมื่อกระบวนการมาถึง กระบวนการจะถูกจัดสรรหน่วยความจำจากรูที่มีขนาดใหญ่พอที่จะรองรับได้

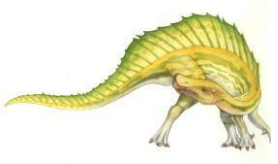
  - When a process arrives, it is allocated memory from a hole large enough to accommodate it

ระบบปฏิบัติการเก็บรักษาข้อมูลเกี่ยวกับ:

  - Operating system maintains information about:
    a) allocated partitions (พาร์ติชันที่จัดสรร) b) free partitions (hole) (พาร์ติชันฟรี (รู))

อาจใช้การจัด Schedule แบบ FCFS





ปัญหาการจัดสรรพื้นที่เก็บข้อมูลแบบไดนามิก

Dynamic Storage-Allocation Problem

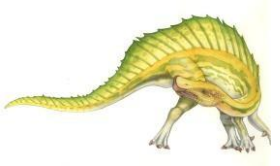
วิธีตอบสนองคำขอขนาด n จากรายการรูว่าง

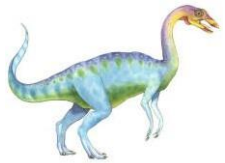
How to satisfy a request of size ***n*** from a list of free holes

- **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)
  - ทำให้มีรูเหลือน้อยที่สุด
  - Produces the smallest leftover hole
- **Worst-fit**: Allocate the ***largest*** hole; must also search entire list (หาพื้นที่ที่ใหญ่ที่สุดก่อน)
  - ทำให้มีรูเหลือที่ใหญ่ที่สุด
  - Produces the largest leftover hole

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

First-fit และ best-fit ดีกว่า worst-fit ในแง่ของความเร็วและการใช้พื้นที่จัดเก็บข้อมูล





Fragmentation การกระจายตัว

การกระจายตัวภายนอก - พื้นที่หน่วยความจำทั้งหมดมีอยู่เพื่อตอบสนองคำขอ แต่ไม่ต่อเนื่องกัน

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

การกระจายตัวภายใน - หน่วยความจำที่จัดสรรอาจมีขนาดใหญ่กว่าหน่วยความจำที่ร้องขอเล็กน้อย

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

ความแตกต่างของขนาดนี้คือหน่วยความจำภายในพาร์ติชัน แต่ไม่ได้ใช้งาน

ลดการกระจายตัวภายนอกด้วยการบดอัด

- Reduce external fragmentation by **compaction**
  - Shuffle memory contents to place all free memory together in one large block

สับเปลี่ยนเนื้อหาหน่วยความจำเพื่อรวมหน่วยความจำที่ว่างทั้งหมดไว้ด้วยกันในบล็อกขนาดใหญ่บล็อกเดียว

  - Compaction is possible *only* if relocation is dynamic, and is done at execution time

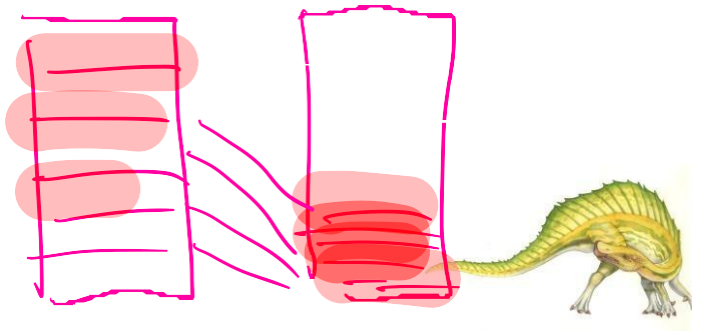
การบดอัดสามารถทำได้เฉพาะในกรณีที่การย้ายตำแหน่งเป็นแบบไดนามิก และเสร็จสิ้นในเวลาดำเนินการ

  - I/O problem

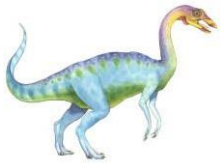
งานสลักในหน่วยความจำขณะเกี่ยวข้องกับ I/O

    - Latch job in memory while it is involved in I/O
    - Do I/O only into OS buffers

ทำ I/O ลงในบัฟเฟอร์ OS เท่านั้น



compaction: การบีบอัด
satisfy: ปฏิบัติตาม
shuffle:สับเปลี่ยน



Paging การสลับหน้า

พื้นที่ที่อยู่ทางกายภาพของกระบวนการอาจไม่ต่อเนื่องกัน กระบวนการจะถูกจัดสรรหน่วยความจำกายภาพทุกครั้งที่มีหน่วยความจำหลัง

- Physical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available

แบ่งหน่วยความจำกายภาพออกเป็นบล็อกขนาดคงที่ที่เรียกว่าเฟรม (ขนาดคือกำลัง 2 ระหว่าง 512 ไบต์ถึง 16 เมกะไบต์)

- Divide physical memory into fixed-sized blocks called **frames** (size is power of 2, between 512 bytes and 16 Mbytes)

แบ่งหน่วยความจำลอจิคัลออกเป็นบล็อกที่มีขนาดเท่ากันเรียกว่าเพจ

- Divide logical memory into blocks of same size called **pages**

ติดตามเฟรมฟรีทั้งหมด

- Keep track of all free frames

ในการรันโปรแกรมขนาด N เพจ จำเป็นต้องค้นหาเฟรมว่าง N เฟรมและโหลดโปรแกรม

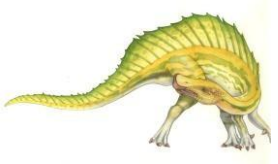
- To run a program of size ***N*** pages, need to find ***N*** free frames and load program

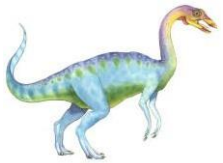
ตั้งค่าตารางหน้าเพื่อแปลตรรกะเป็นที่อยู่จริง

- Set up a **page table** to translate logical to physical addresses

ยังคงมีการกระจายตัวภายใน

- Still have Internal fragmentation





โครงการแปลที่อยู่

Address Translation Scheme

ที่อยู่ที่สร้างโดย CPU แบ่งออกเป็น:

- Address generated by CPU is divided into:

หมายเลขหน้า (p) - ใช้เป็นดัชนีในตารางหน้าซึ่งมีที่อยู่ฐานของแต่ละหน้าในหน่วยความจำกายภาพ

- **Page number (*p*)** – used as an index into a ***page table*** which contains base address of each page in physical memory

Page offset (d) - รวมกับที่อยู่ฐานเพื่อกำหนดที่อยู่หน่วยความจำกายภาพที่ส่งไปยังหน่วยหน่วยความจำ

- **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

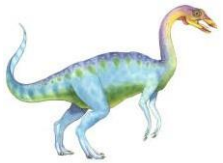
page number	page offset
p	*d*
m - *n* bits	*n* bits

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

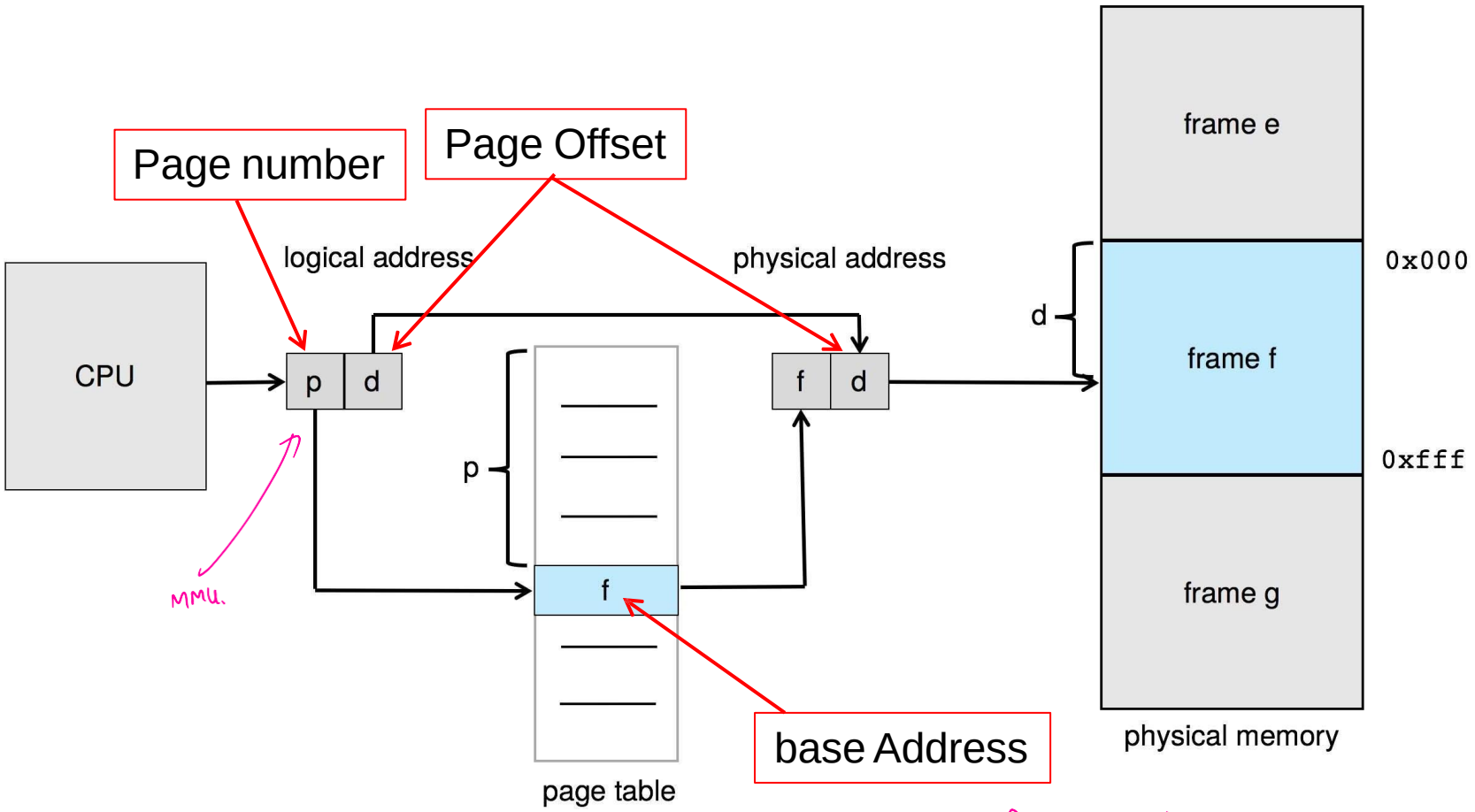
สำหรับพื้นที่ที่อยู่โลจิคัลที่กำหนด 2^m และขนาดหน้า 2^n

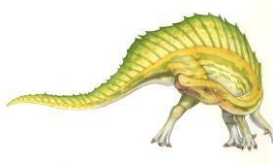
- For given logical address space 2^m and page size 2^n bytes

Since each address is m-bit long



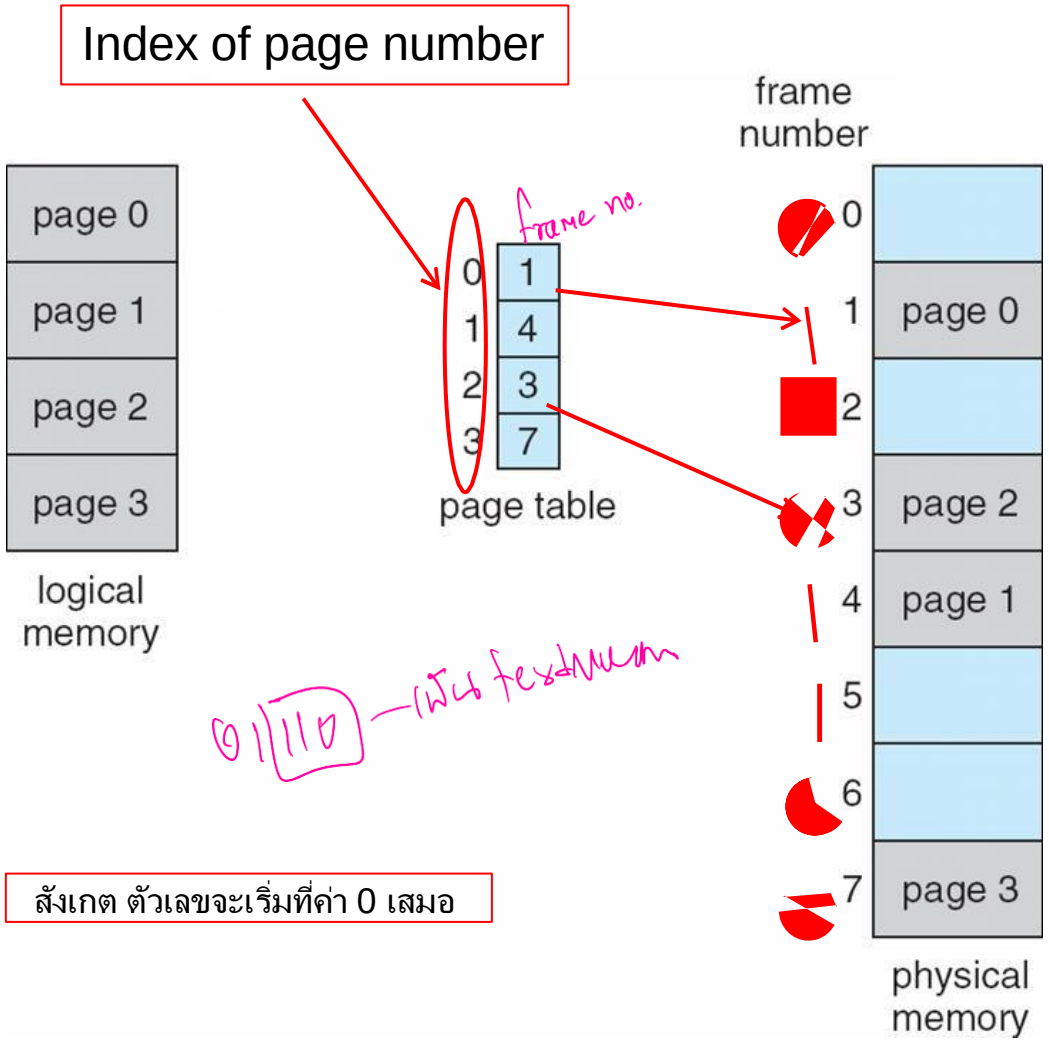
Paging Hardware



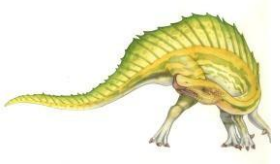


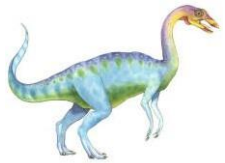
แบบจำลองเพจจิ้งของหน่วยความจำโลจิคัลและฟิสิคัล

Paging Model of Logical and Physical Memory



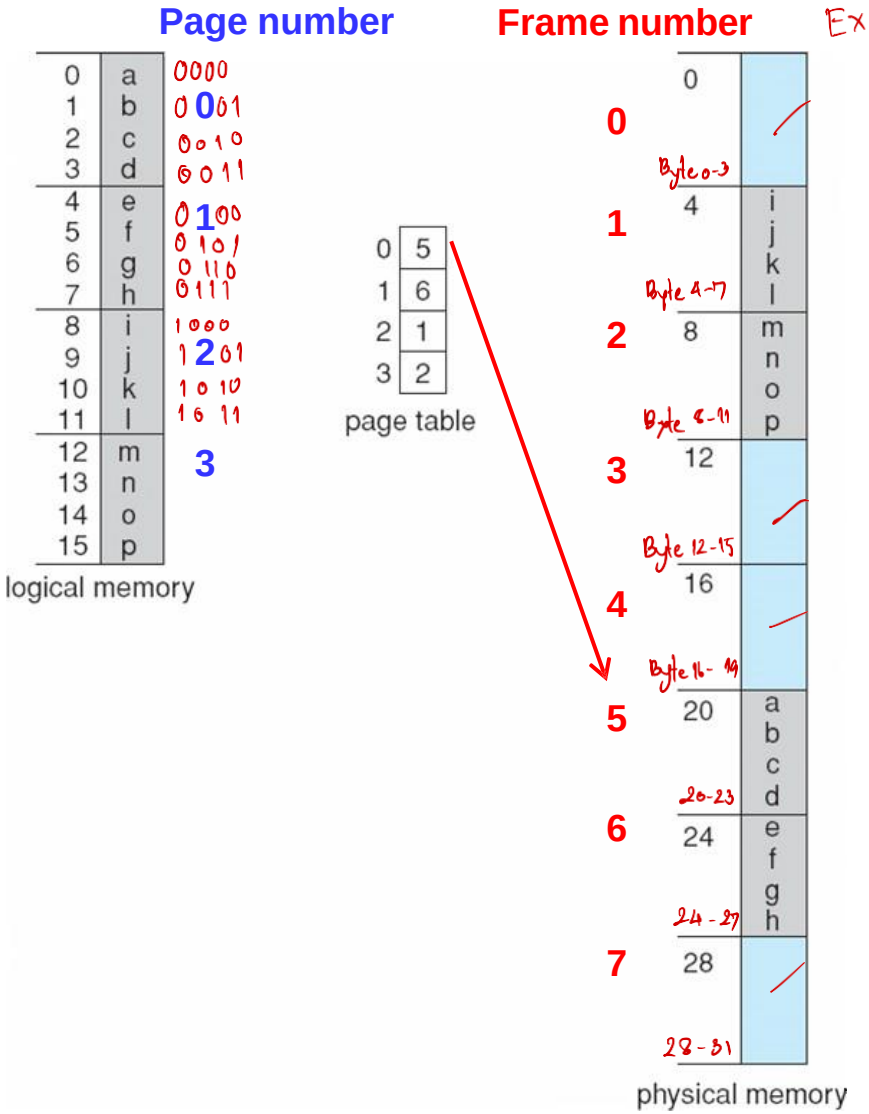
สังเกต ตัวเลขจะเริ่มที่ค่า 0 เสมอ

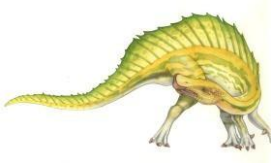




Paging Example

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



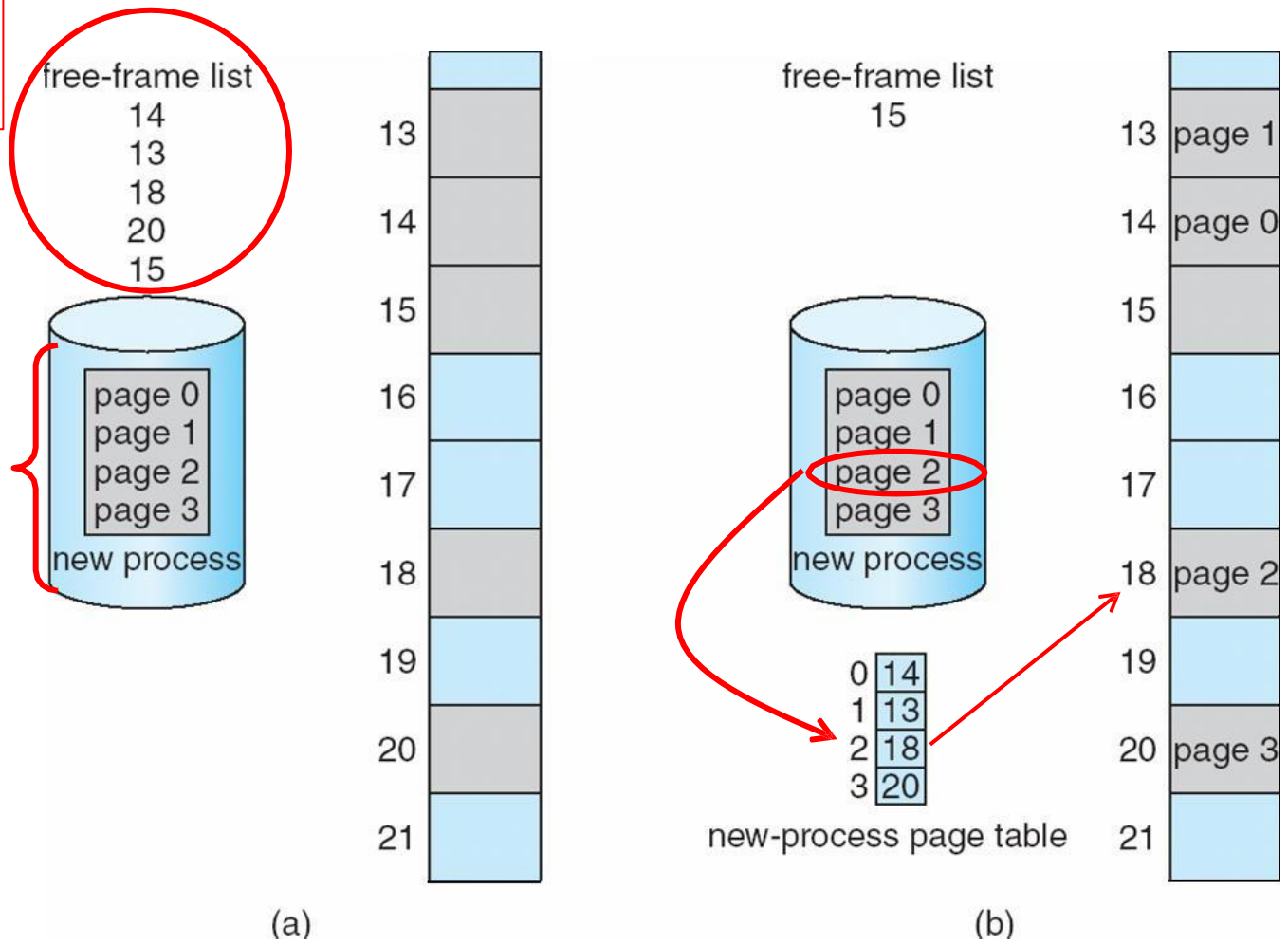


Free Frames

เรียงตามลำดับ

Free-frame

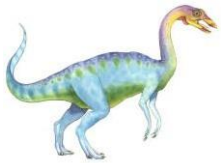
ที่ว่าง



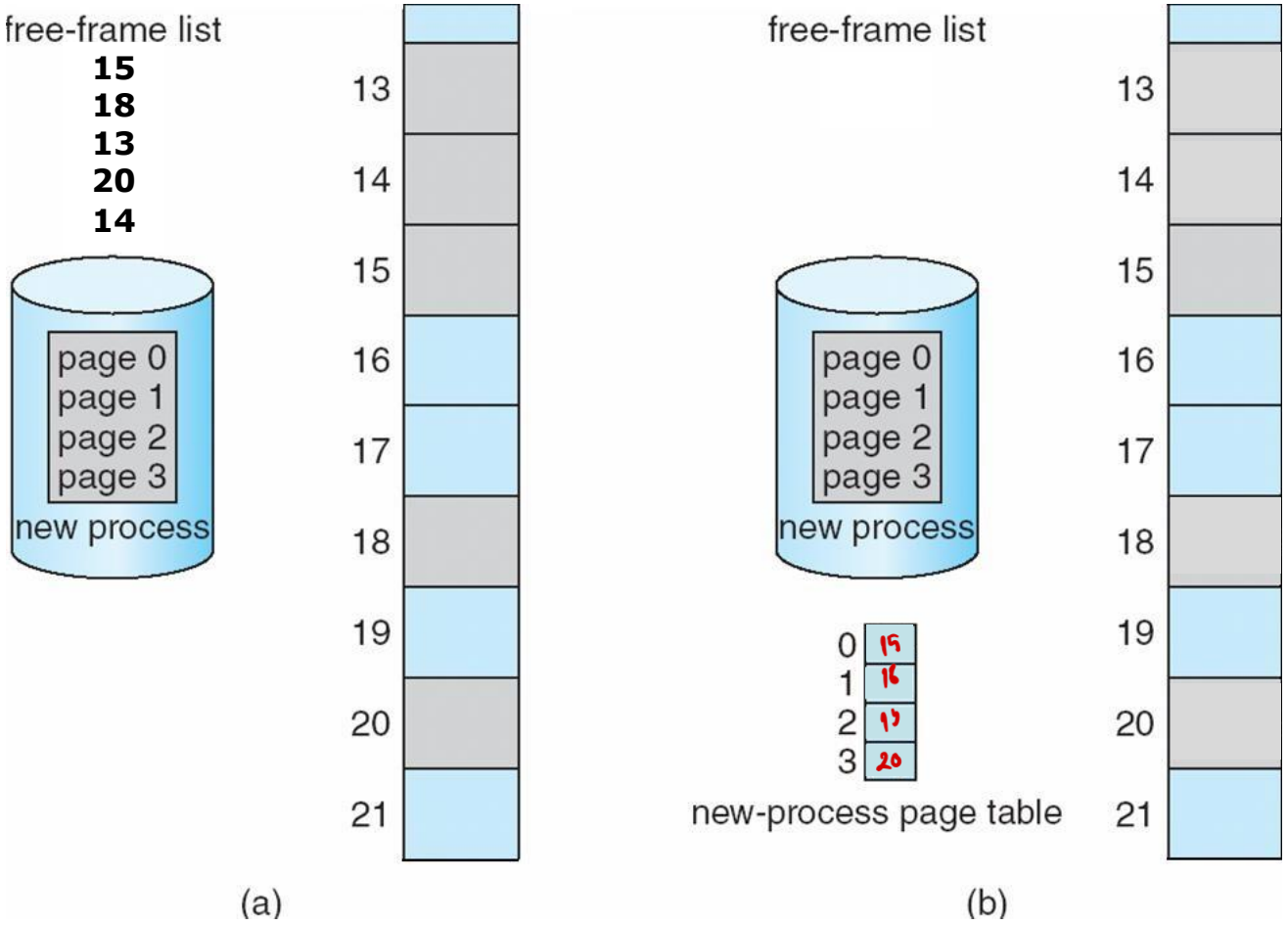
4 Pages

Before allocation

After allocation

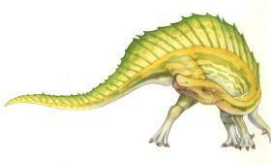


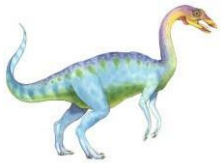
Exercise: Free Frames



Before allocation

After allocation





การดำเนินการตารางเพจ

Implementation of Page Table

ตารางหน้าถูกเก็บไว้ในหน่วยความจำหลัก

- Page table is kept in main memory

รีจิสเตอร์ฐานตารางเพจ (PTBR) ชี้ไปที่ตารางเพจ

- **Page-table base register (PTBR)** points to the page table

การลงทะเบียนความยาวหน้าตาราง (PRLR) ระบุขนาดของตารางหน้า

- **Page-table length register (PRLR)** indicates size of the page table

ในรูปแบบนี้ การเข้าถึงข้อมูล/คำสั่งทุกครั้งจำเป็นต้องมีการเข้าถึงหน่วยความจำสองครั้ง หนึ่งรายการสำหรับตารางหน้าและอีกรายการหนึ่งสำหรับข้อมูล/คำสั่ง

- In this scheme every data/instruction access requires two memory accesses. One for the page table and one for the data/instruction.

ปัญหาการเข้าถึงหน่วยความจำทั้งสองสามารถแก้ไขได้โดยการใช้แคชฮาร์ดแวร์การค้นหาแบบพิเศษที่เรียกว่าหน่วยความจำแบบเชื่อมโยงหรือบัฟเฟอร์มองข้างการแปล (TLB)

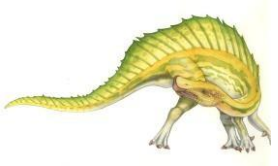
- The two memory access problem can be solved by the use of a special fast-lookup hardware cache called **associative memory** or **translation look-aside buffers (TLBs)**

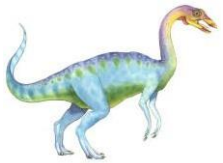
TLB บางตัวจัดเก็บตัวระบุพื้นที่ที่อยู่ (ASID) ไว้ในแต่ละรายการ TLB โดยจะระบุแต่ละกระบวนการโดยไม่ซ้ำกันเพื่อให้การป้องกันพื้นที่ที่อยู่สำหรับกระบวนการนั้น

- Some TLBs store **address-space identifiers (ASIDs)** in each TLB entry – uniquely identifies each process to provide address-space protection for that process

TLB โดยทั่วไปมีขนาดเล็ก (64 ถึง 1,024 รายการ)

  - TLBs typically small (64 to 1,024 entries)





หน่วยความจำเชื่อมโยง

Associative Memory

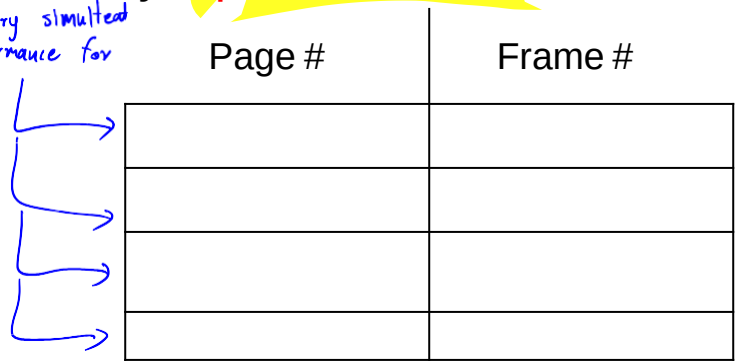
Stored in a cache

หน่วยความจำเชื่อมโยง - การค้นหาแบบขนาน

Mapping Page # → Fram

- Associative memory – parallel search

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



การแปลที่อยู่ (p, d)

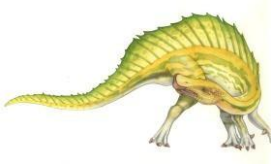
Address translation (p, d)

ถ้า p อยู่ใน associative register ให้ดึง frame # ออก

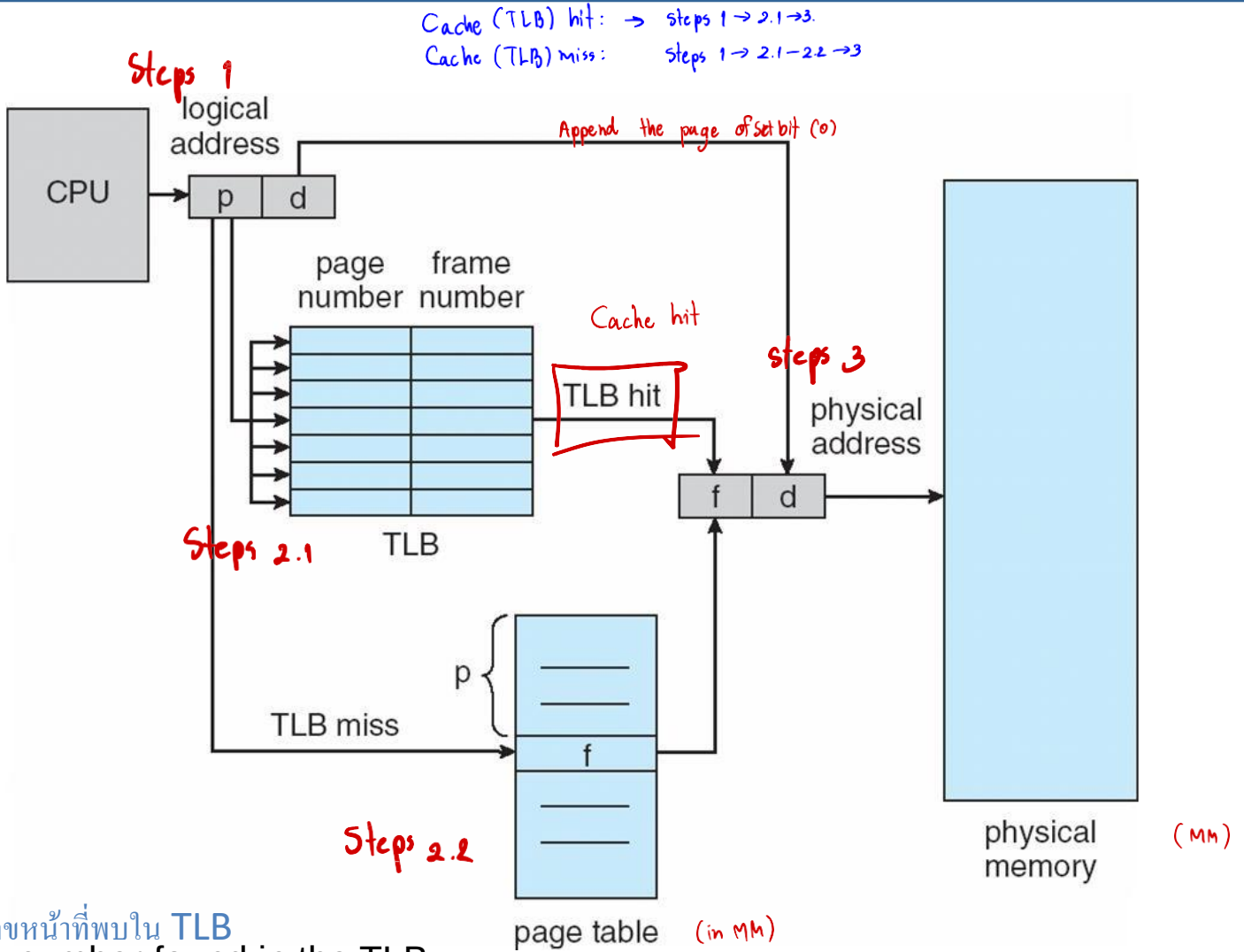
- If p is in associative register, get frame # out

มิฉะนั้นรับ frame # จากตารางหน้าในหน่วยความจำ

- Otherwise get frame # from page table in memory



Paging Hardware With TLB

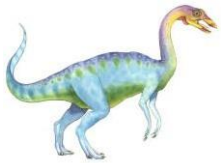


หมายเลขหน้าที่พบใน TLB

hit : page number found in the TLB

miss : page number not in the TLB

หมายเลขหน้าไม่อยู่ใน TLB



การป้องกันหน่วยความจำ

Memory Protection

การป้องกันหน่วยความจำดำเนินการโดยการเชื่อมโยงบิตการป้องกันกับแต่ละเฟรม

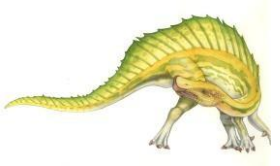
- Memory protection implemented by associating protection bit with each frame

บิตที่ถูกต้องและไม่ถูกต้องที่แนบมากับแต่ละรายการในตารางหน้า:

- **Valid-invalid** bit attached to each entry in the page table:
  - "ถูกต้อง" บ่งชี้ว่าเพจที่เกี่ยวข้องอยู่ในพื้นที่ที่อยู่แบบลอจิคัลของกระบวนการ และด้วยเหตุนี้จึงเป็นเพจทางกฎหมาย
    - "valid" indicates that the associated page is in the process' logical address space, and is thus a legal page
  - "ไม่ถูกต้อง" บ่งชี้ว่าเพจไม่อยู่ในพื้นที่ที่อยู่โลจิคัลของกระบวนการ
    - "invalid" indicates that the page is not in the process' logical address space
  - หรือใช้การลงทะเบียนความยาวหน้าตาราง (PTLR)
    - Or use **page-table length register** (**PTLR**)

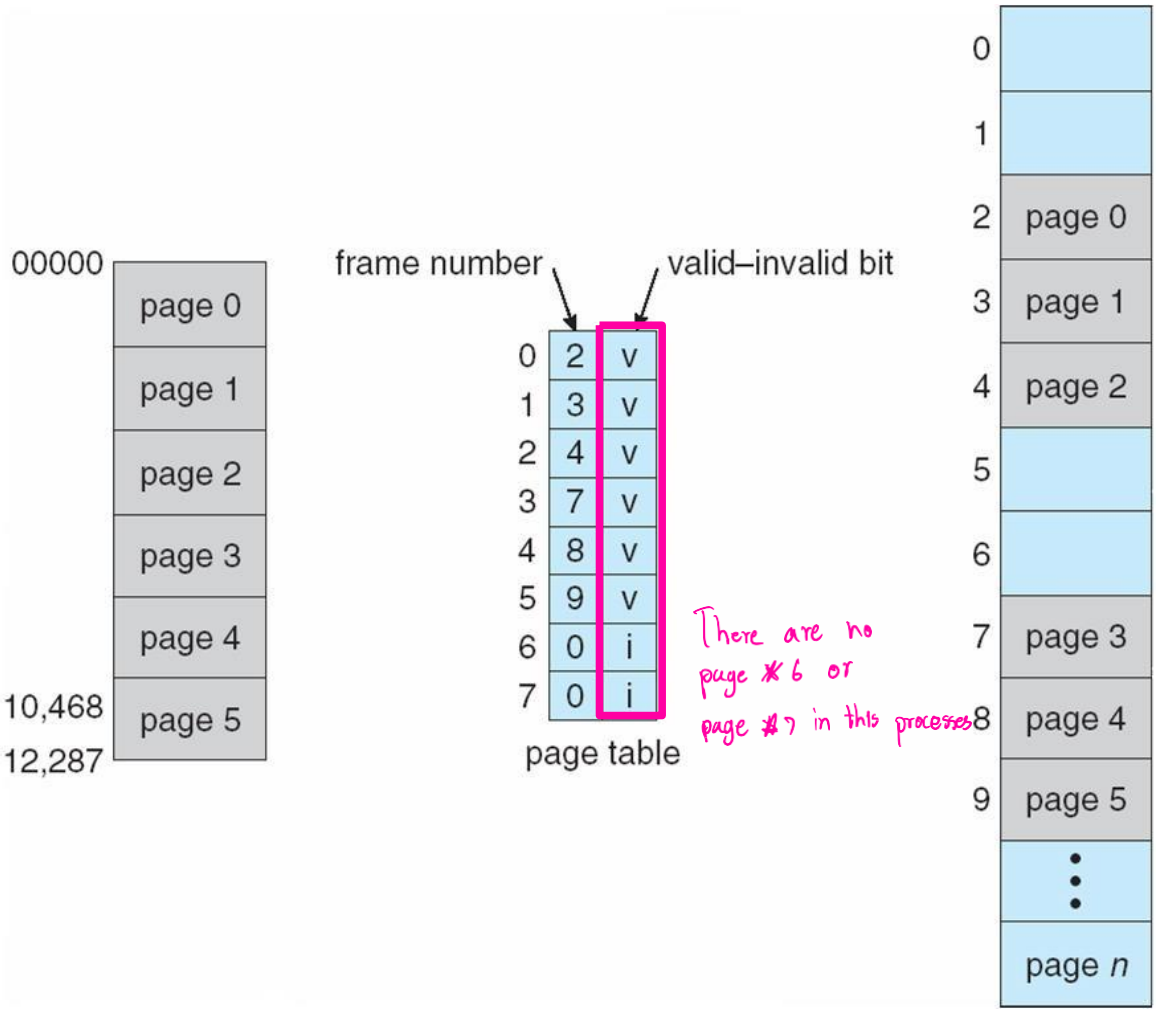
การละเมิดใดๆ ส่งผลให้เกิดกับดักกับเคอร์เนล

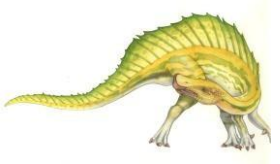
- Any violations result in a trap to the kernel

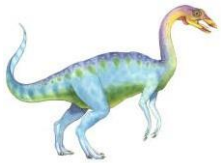


บิตที่ถูกต้อง (v) หรือไม่ถูกต้อง (i) ในตารางเพจ

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

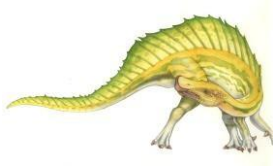


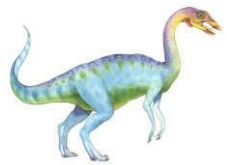




Shared Pages เพจที่ใช้ร่วมกัน

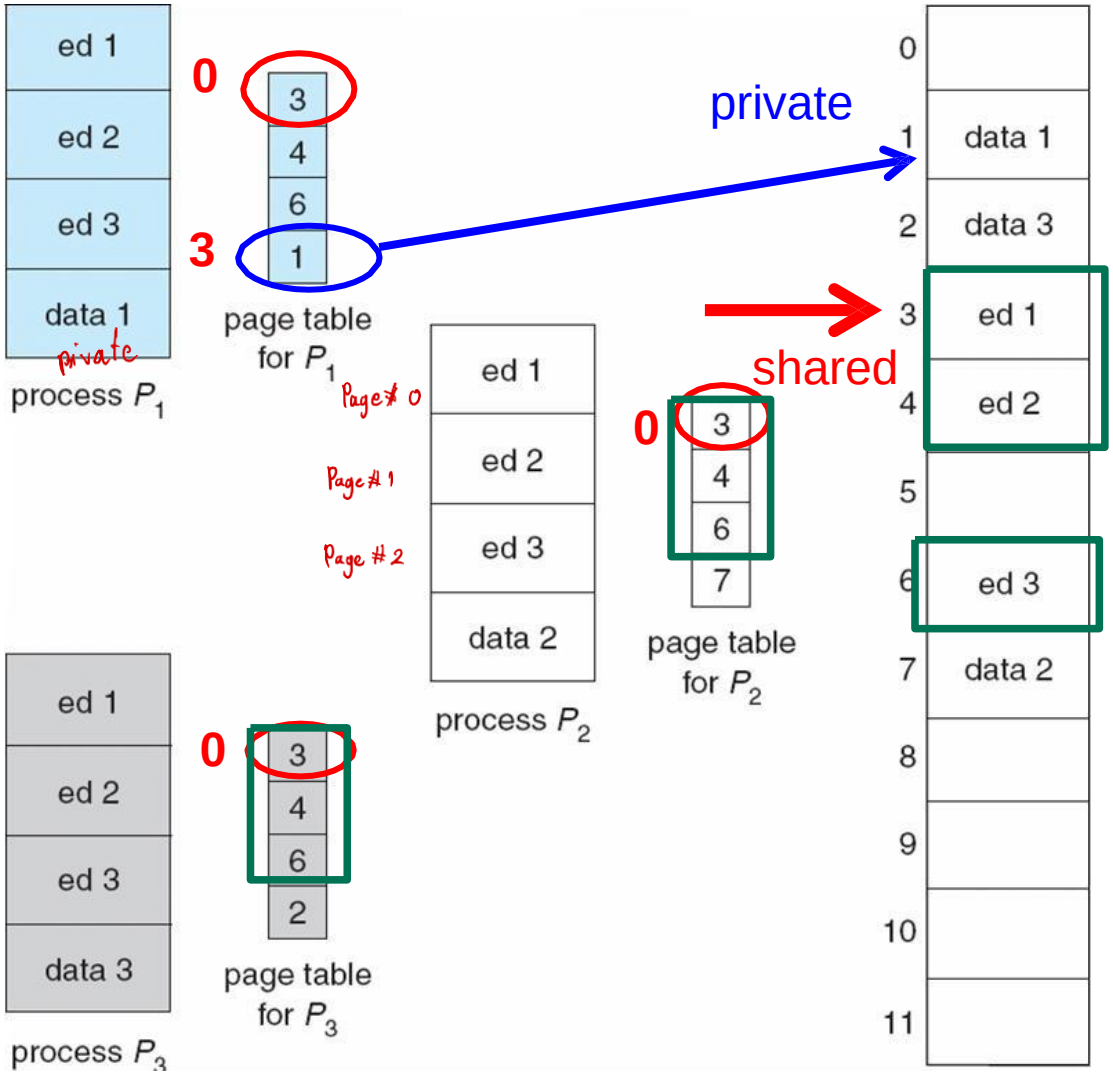
- **Shared code** รหัสที่ใช้ร่วมกัน
  - One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems).
  - Shared code must appear in same location in the logical address space of all processes
- **Private code and data** รหัสส่วนตัวและข้อมูล
  - Each process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

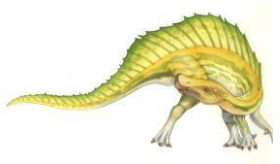


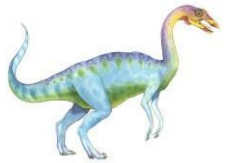


ตัวอย่างเพจที่ใช้ร่วมกัน

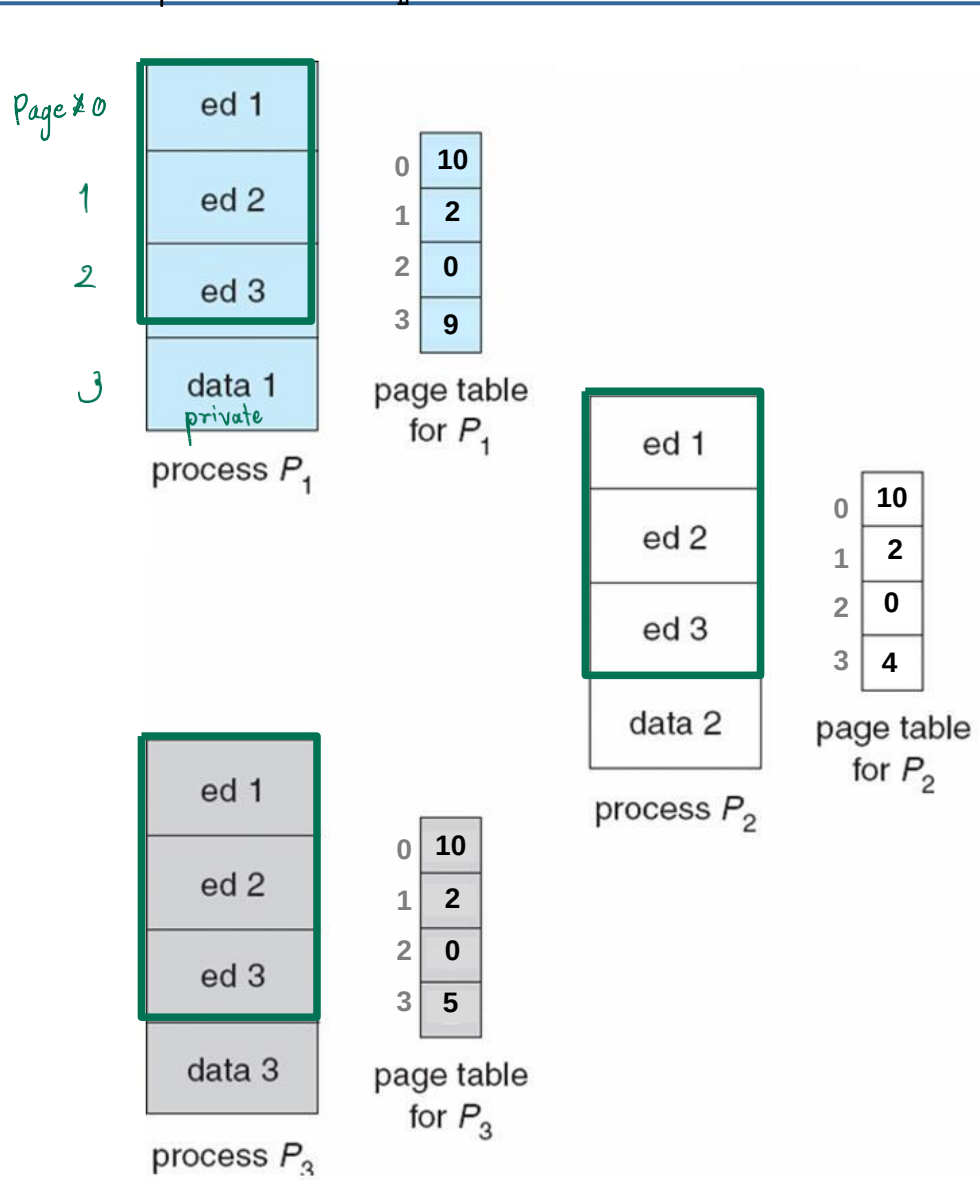
Shared Pages Example

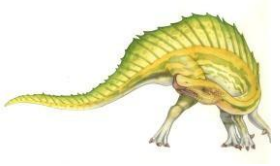


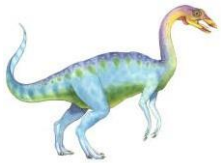




Exercise: จงเติมข้อมูลในส่วนของ Frame ที่สัมพันธ์กับโพรเซสต่างๆ ที่ทำงานอยู่ในระบบที่มีการใช้งาน Shared Pages ด้วย







Segmentation (แบ่งเป็นตอน)

รูปแบบการจัดการหน่วยความจำที่รองรับการดูหน่วยความจำของผู้ใช้

- Memory-management scheme that supports user view of memory

โปรแกรมคือชุดของเซ็กเมนต์

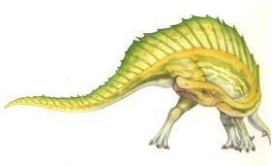
- A program is a collection of segments

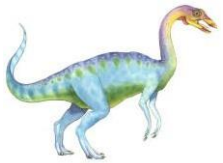
เซ็กเมนต์เป็นหน่วยทางลอจิคัล เช่น:

- A segment is a logical unit such as:

ส่วน ประกอบ ของ program

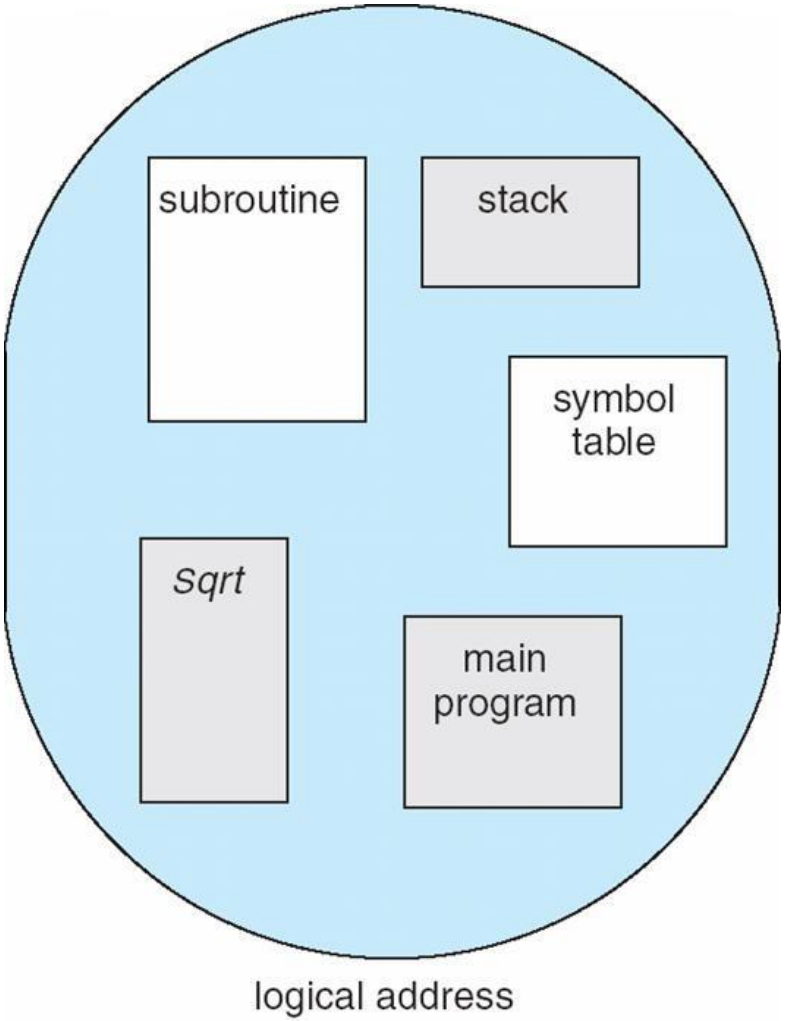
main program	โปรแกรมหลัก
procedure	ขั้นตอน
function	การทำงาน
method	วิธี
object	วัตถุ
local variables, global variables	ตัวแปรท้องถิ่น ตัวแปรโกลบอล
common block	บล็อกทั่วไป
stack	ซ้อนกัน
symbol table	ตารางสัญลักษณ์
arrays	อาร์เรย์

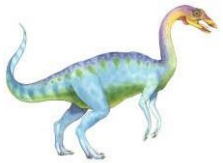




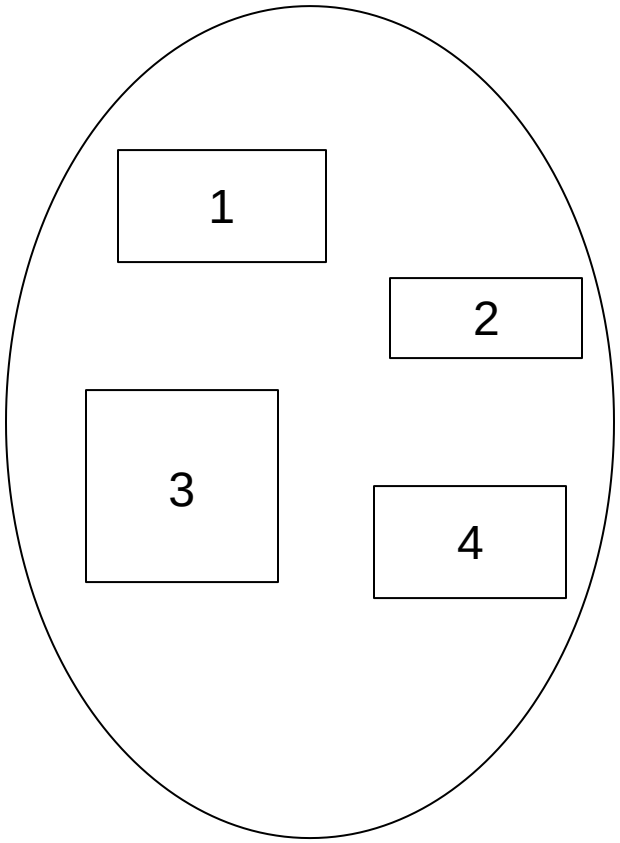
มุมมองของผู้ใช้โปรแกรม

User's View of a Program



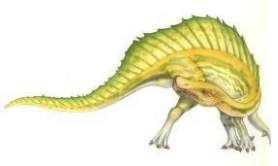


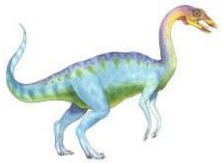
Logical View of Segmentation



user space

physical memory space





สถาปัตยกรรมการแบ่งส่วน

Segmentation Architecture

อ้างอิงความเข้าใจในเรื่อง paging

ที่อยู่แบบลอจิคัลประกอบด้วยสองสิ่งอันดับ:

- Logical address consists of a two tuple:
  <segment-number, offset>,

ตารางเซ็กเมนต์ – แม็ปที่อยู่ทางกายภาพสองมิติ แต่ละรายการในตารางมี:

- **Segment table** – maps two-dimensional physical addresses; each table entry has:
  - **base** – contains the starting physical address where the segments reside in memory
  - **limit** – specifies the length of the segment

การลงทะเบียนฐานตารางส่วน (STBR) ชี้ไปที่ตำแหน่งของตารางส่วนในหน่วยความจำ

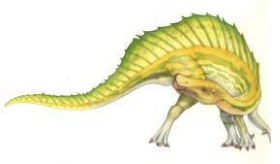
- **Segment-table base register (STBR)** points to the segment table's location in memory

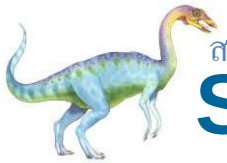
การลงทะเบียนความยาวเซ็กเมนต์ตาราง (STLR) ระบุจำนวนเซ็กเมนต์ที่ใช้โดยโปรแกรม

- **Segment-table length register (STLR)** indicates number of segments used by a program;

  หมายเลขเซ็กเมนต์ s นั้นถูกกฎหมายถ้า s < STLR

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





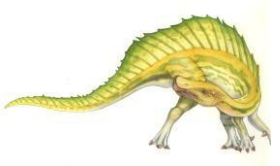
Segmentation Architecture (Cont.)

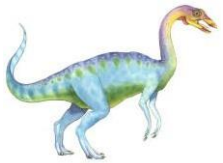
การป้องกัน
- Protection
  - ด้วยแต่ละรายการในการเชื่อมโยงตารางส่วน:
  - With each entry in segment table associate:
    - บิตการตรวจสอบ = 0 ⇒ ส่วนที่ไม่ถูกต้อง
    - validation bit = 0 ⇒ illegal segment
    - สิทธิ์การอ่าน/เขียน/ดำเนินการ
    - read/write/execute privileges

บิตป้องกันที่เกี่ยวข้องกับเซ็กเมนต์ การแบ่งปันรหัสเกิดขึ้นในระดับเซ็กเมนต์
- Protection bits associated with segments; code sharing occurs at segment level

เนื่องจากเซ็กเมนต์มีความยาวต่างกัน การจัดสรรหน่วยความจำจึงเป็นปัญหาการจัดสรรพื้นที่เก็บข้อมูลแบบไดนามิก
- Since segments vary in length, memory allocation is a dynamic storage-allocation problem

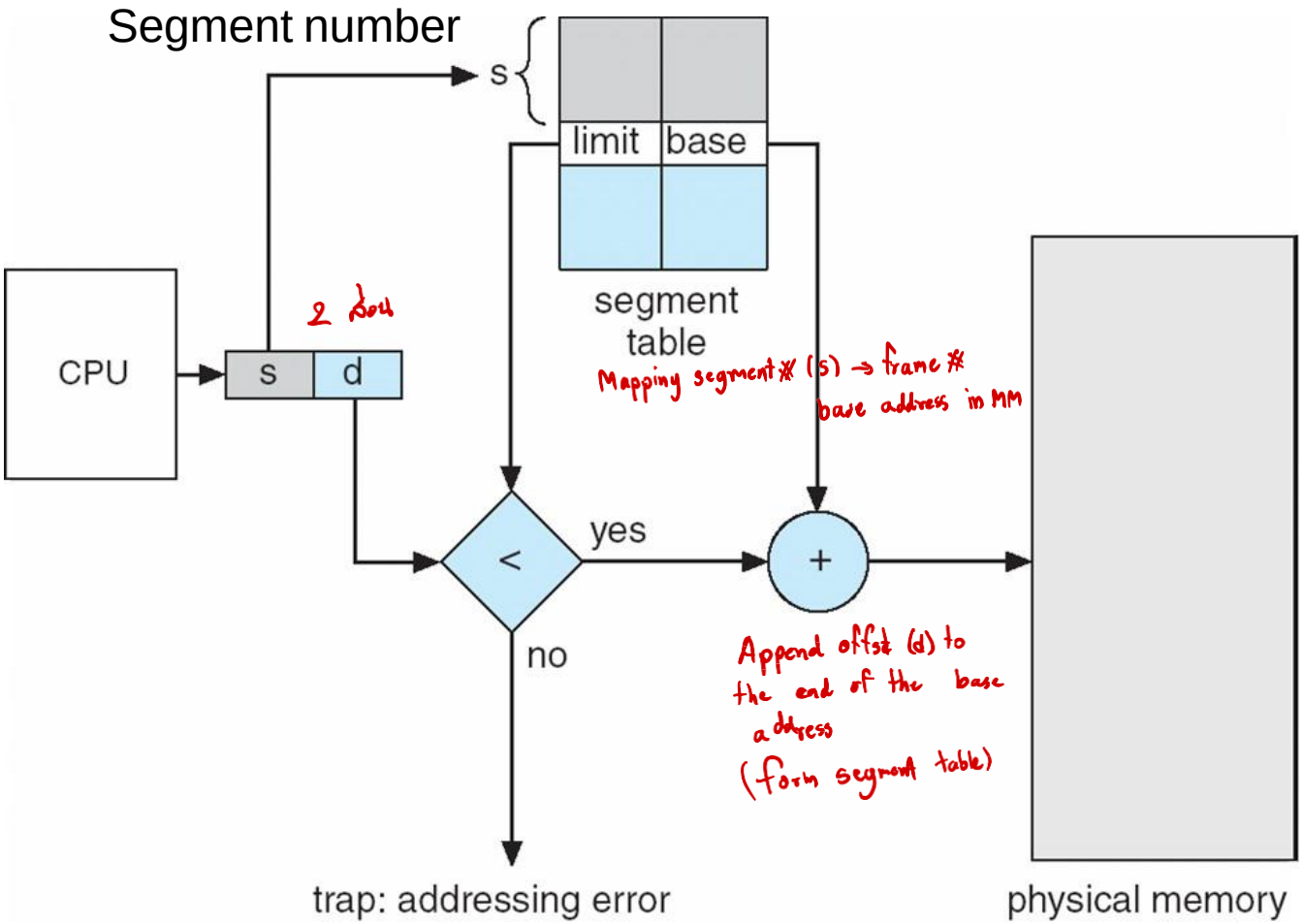
ตัวอย่างการแบ่งส่วนจะแสดงในแผนภาพต่อไปนี้
- A segmentation example is shown in the following diagram

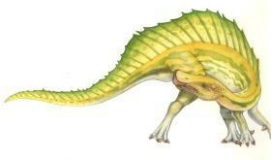


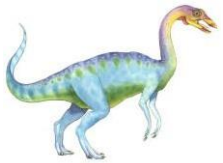


ฮาร์ดแวร์การแบ่งส่วน

Segmentation Hardware

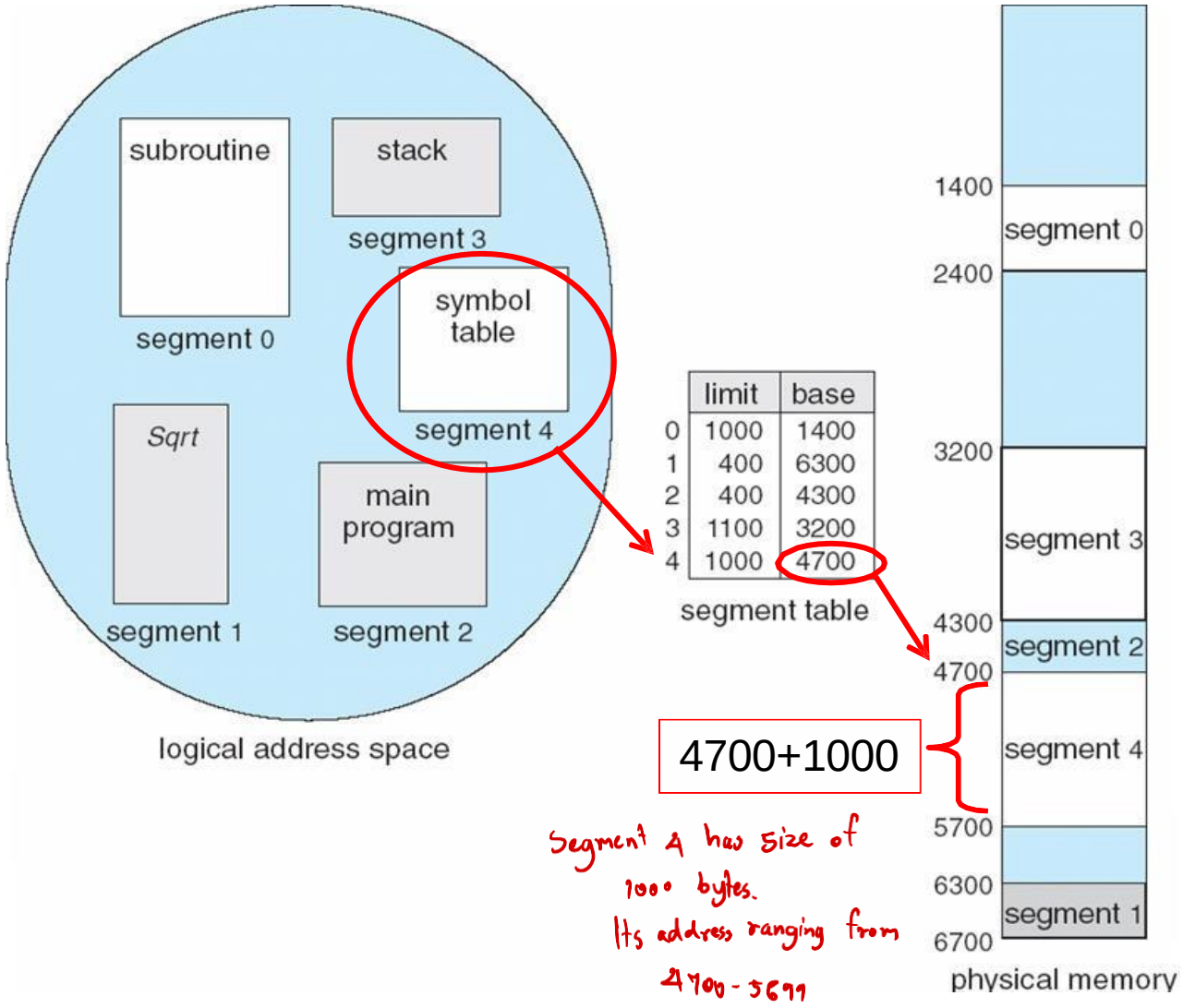


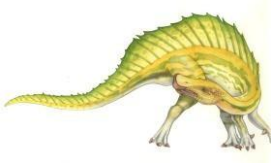




ตัวอย่างการแบ่งส่วน

Example of Segmentation





End of Chapter 7