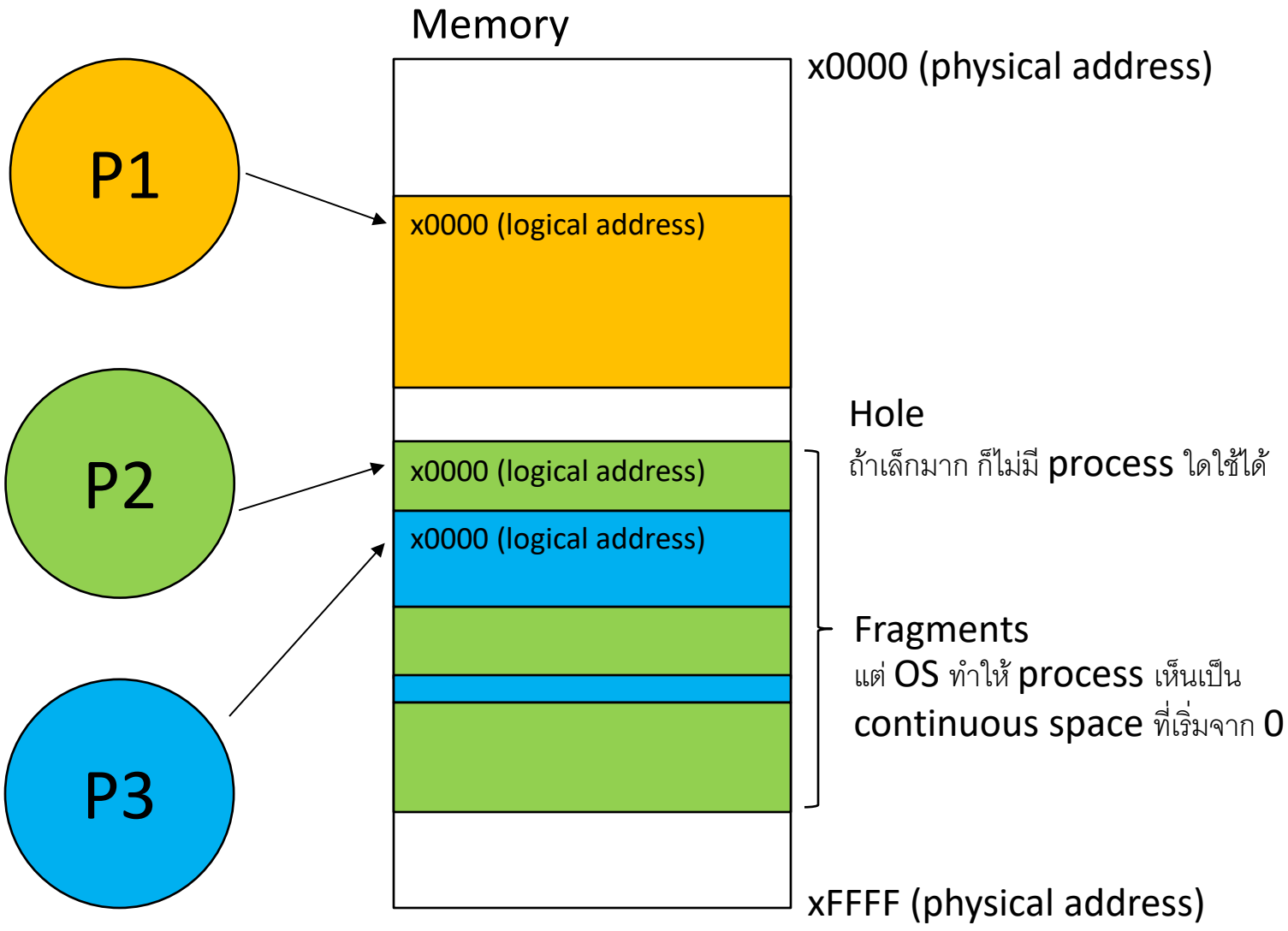
แก้ไขครั้งล่าสุดเมื่อวันที่ 9 ตุลาคม 2567 (หลังสอน)

Memory Management



Memory Separation

แบ่ง memory ให้แต่ละ process

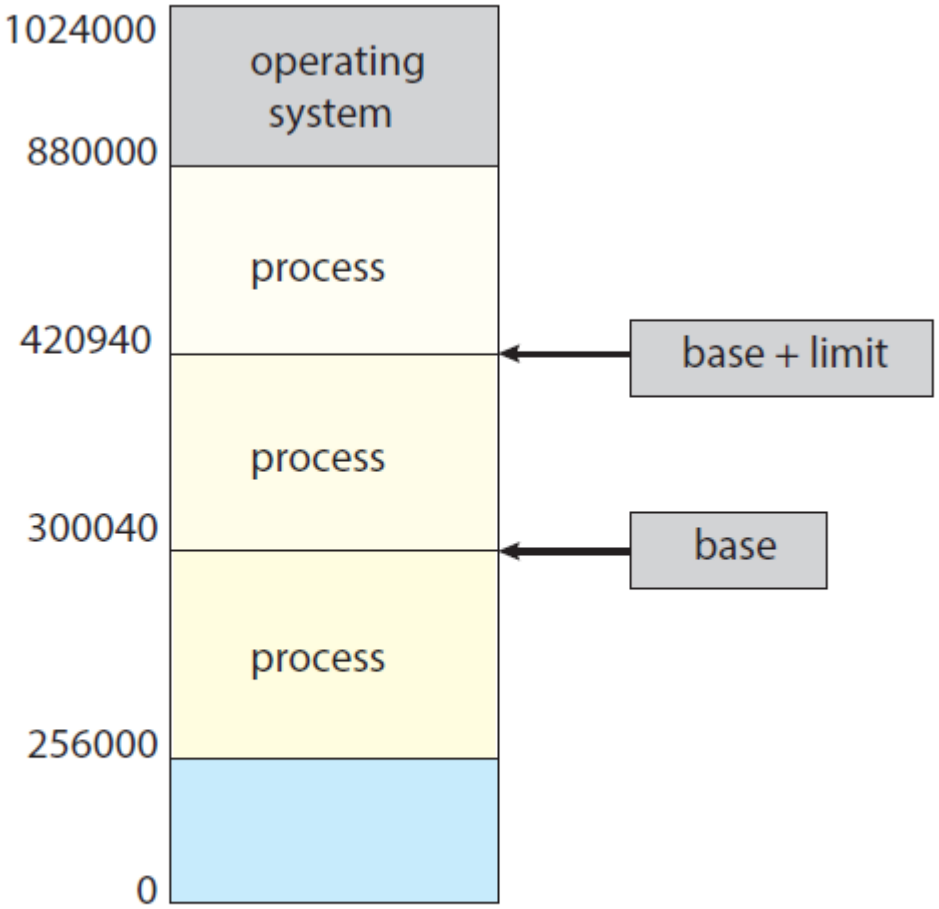


Figure 9.1 A base and a limit register define a logical address space.

Memory Protection

ป้องกันไม่ให้อ่าน/เขียน memory เกินขอบเขตที่จองไว้

ใน kernel mode เท่านั้น ที่จะแก้ไขค่า base และ limit ได้

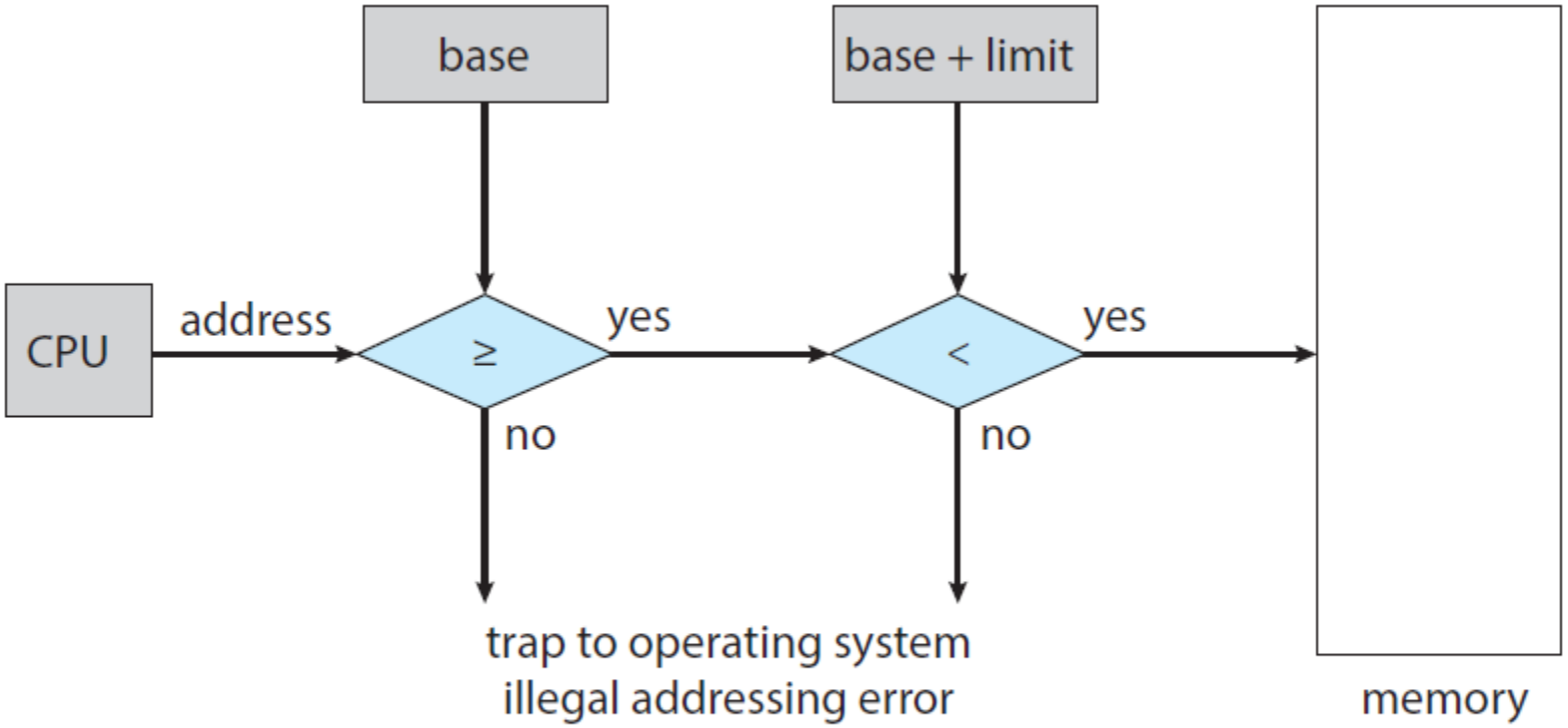
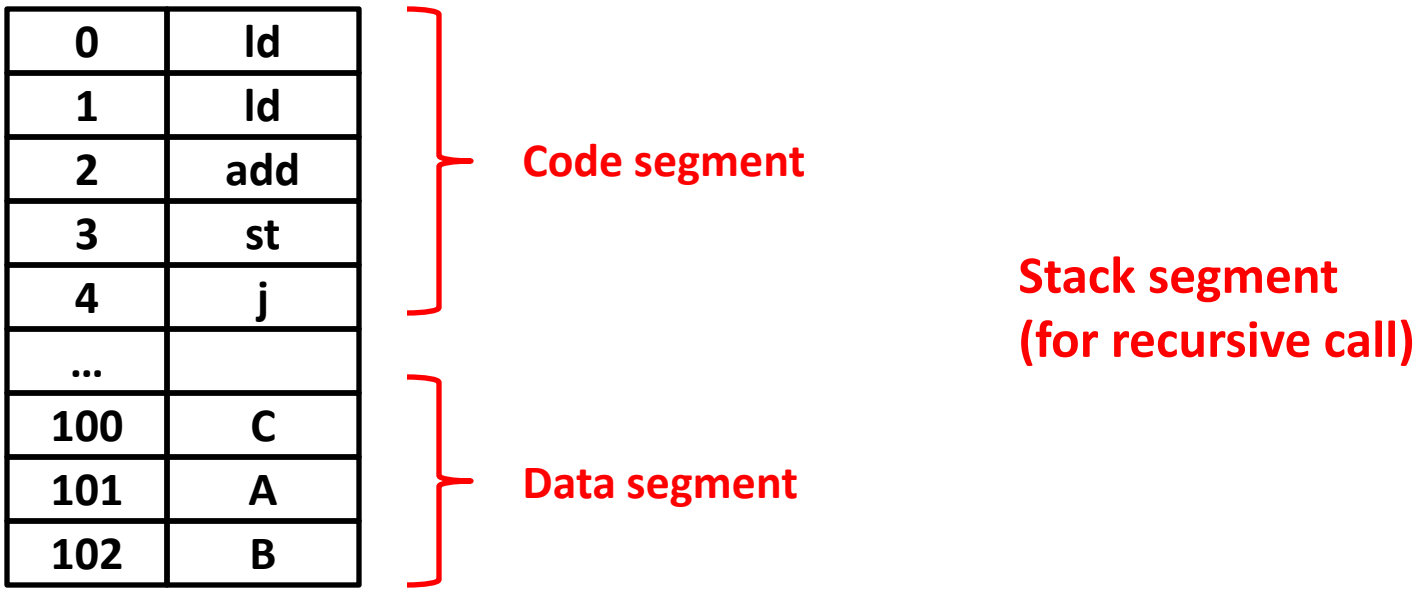


Figure 9.2 Hardware address protection with base and limit registers.

มี **separation & protection** แล้ว แต่ยังต้องโปรแกรมด้วย **physical address**

```
ld      r1    101          // data segment (variable A)
ld      r2    102          // data segment (variable B)
add     r0    r1    r2
st      r0    100          // data segment (variable C = A + B)
J       50                 // code segment
```

This program assumes that base address is zero.
If not, the program may encounter illegal memory access.



Logical Address vs. Physical Address

Logical address must begin from zero.

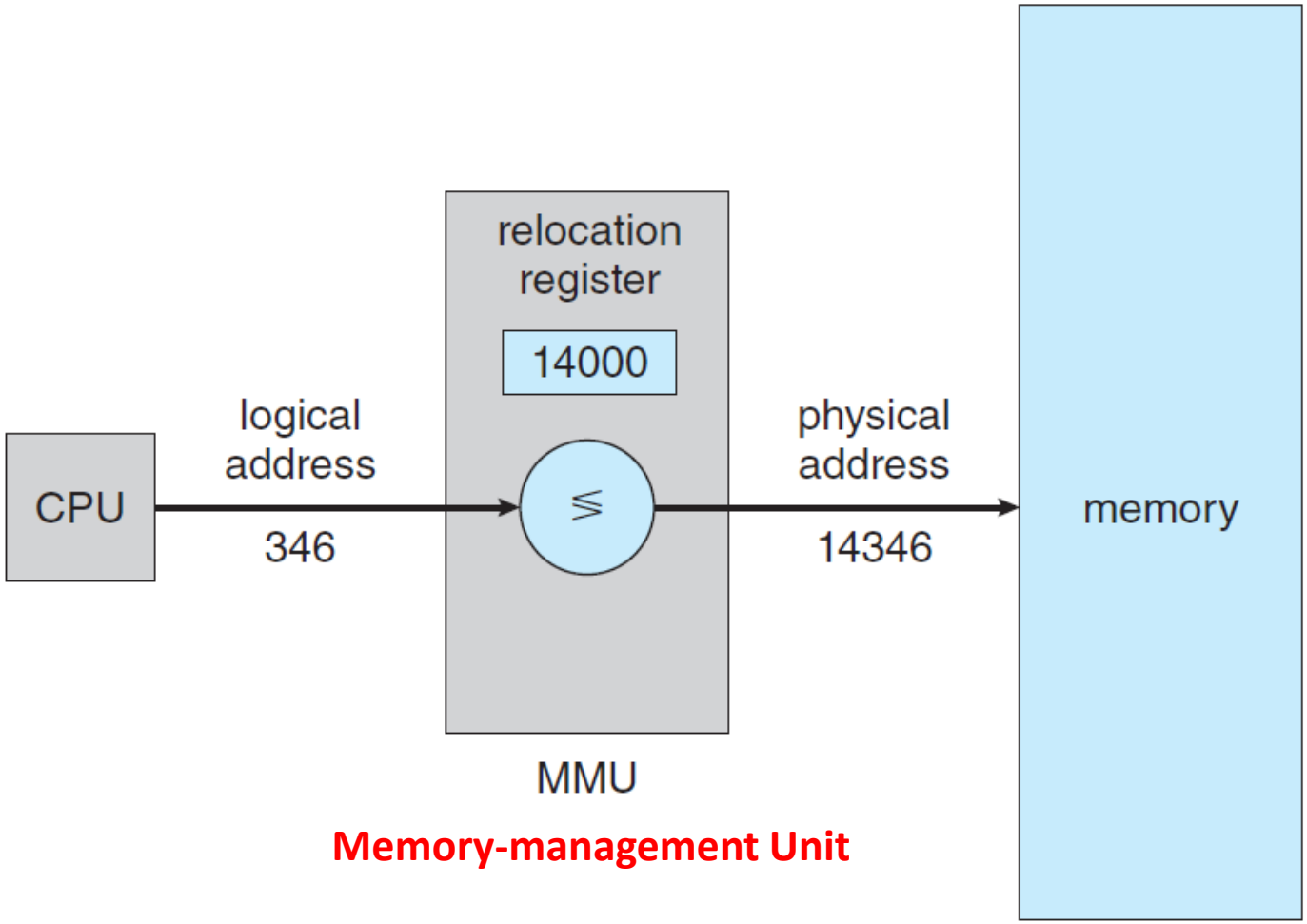


Figure 9.5 Dynamic relocation using a relocation register.

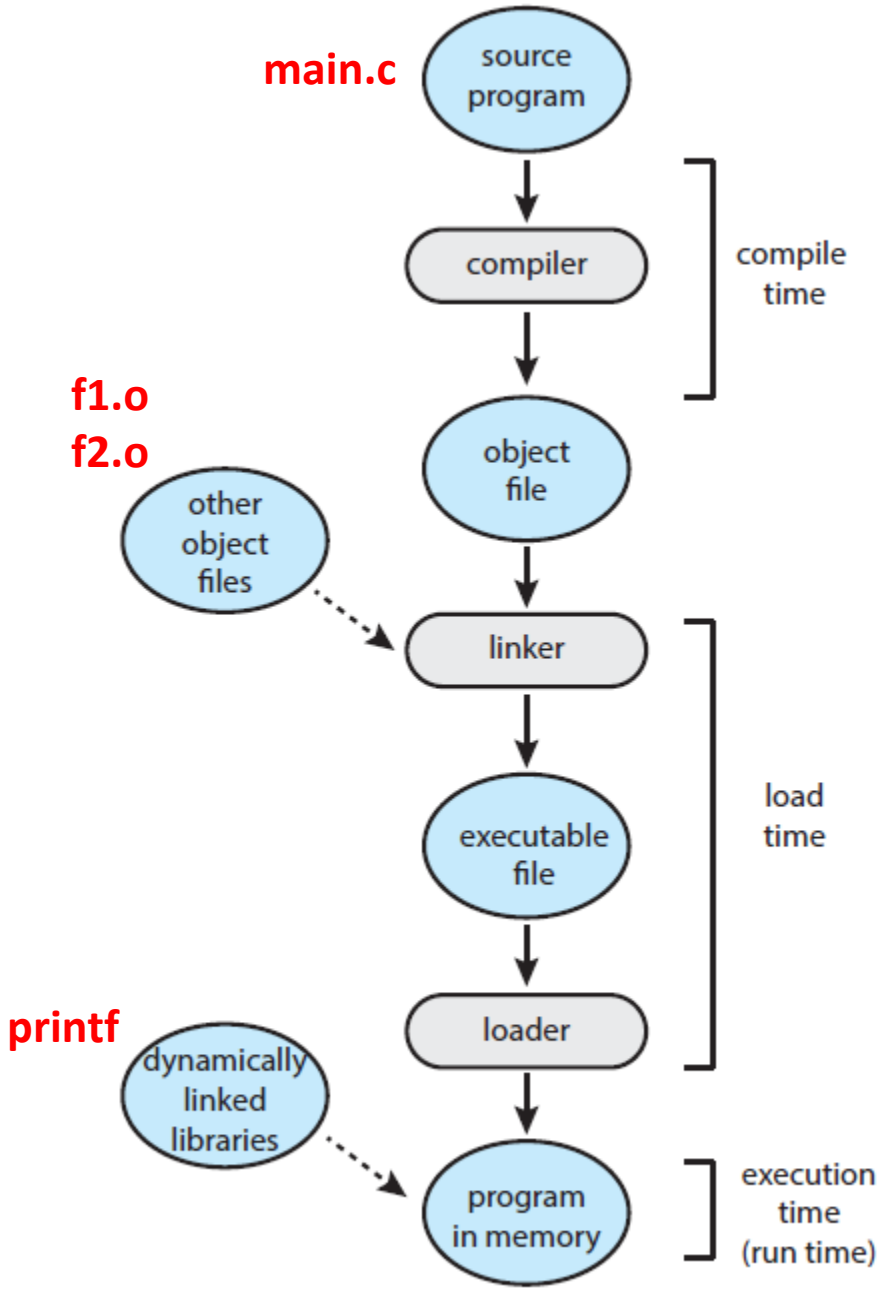


Figure 9.3 Multistep processing of a user program.

Address Binding

- Compile time

```
ld    r1   101
ld    r2   102
add   r0   r1   r2
st    r0   100
J    50
```

- Load time

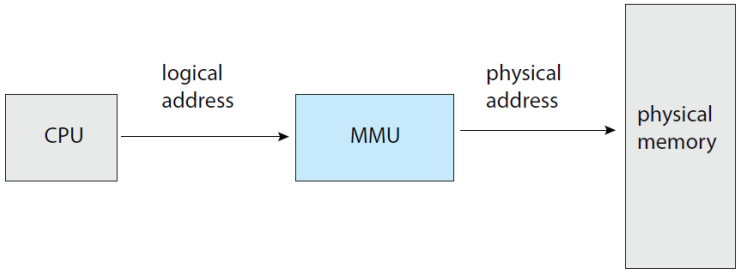
```
ld    r1   [ ]
ld    r2   [ ]
add   r0   r1   r2
st    r0   [ ]
J    [ ]
```

เว้น address ไว้ก่อน
เมื่อรู้ว่าจะโหลดลงที่ใด
ค่อยเติม addr ให้ถูกต้อง

- Execution time

เกิดจาก dynamic loading หรือ swapping

ใช้ logical address เริ่มต้นจาก 0
ต้องมี relocation reg.



Dynamic Loading

1. Load main function into memory.
2. Load other functions on demand.

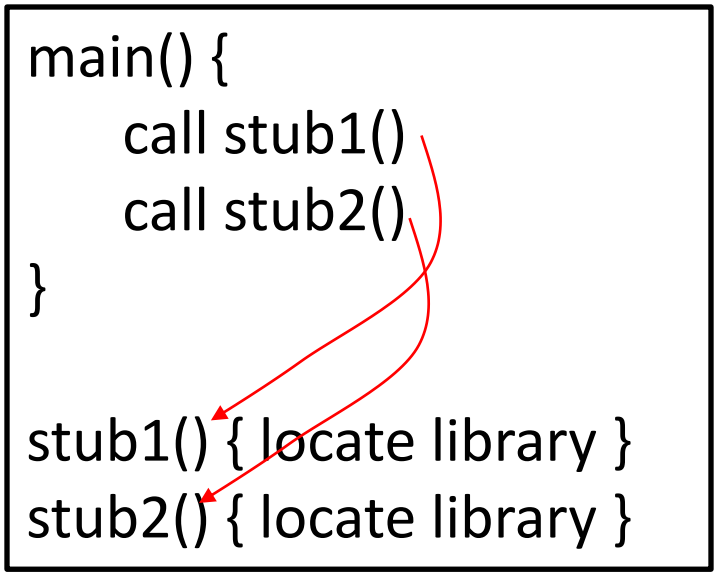
This method is particularly useful when large amounts of code are needed to handle infrequently occurring cases, such as error routines.

There may be multiple copies of the same routine in memory.

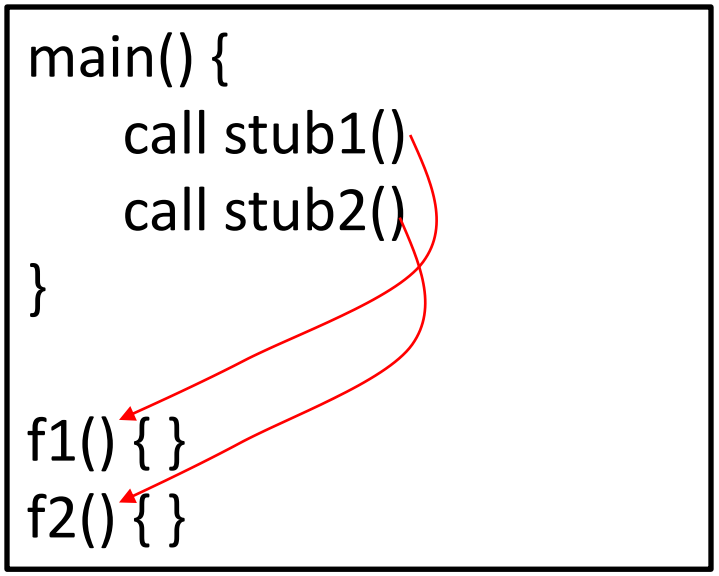
Dynamic Linking and Shared Libraries

1. Static linking exe = main.o + f1.o + f2.o
2. Dynamic linking exe = main.o + stub1 + stub2

Before calling



After calling



There is only single copy of f1() and f2() in memory.

Swapping

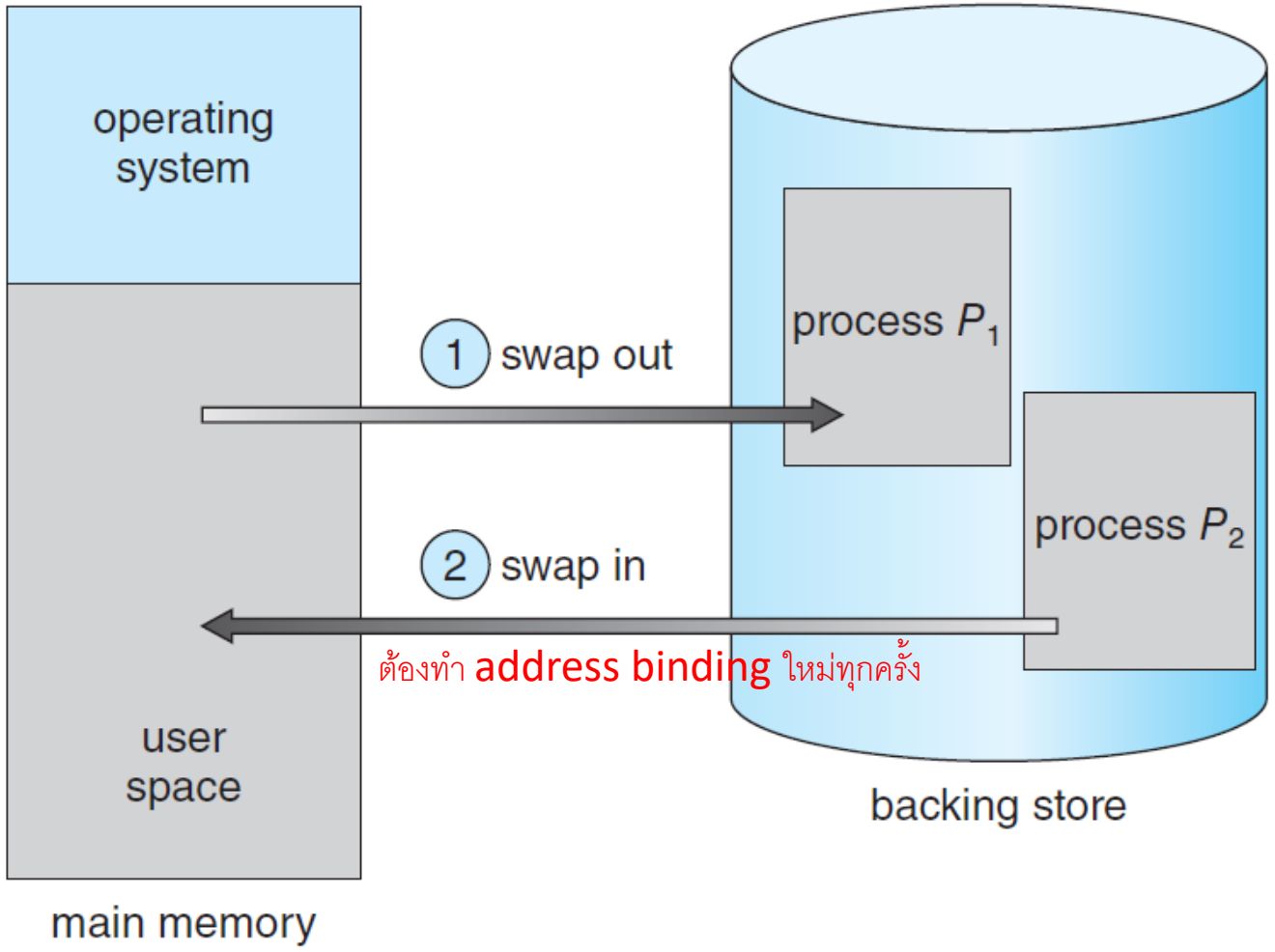


Figure 9.19 Standard swapping of two processes using a disk as a backing store.

Memory Mapping & Protection

ยังเป็น **Contiguous Memory Allocation**

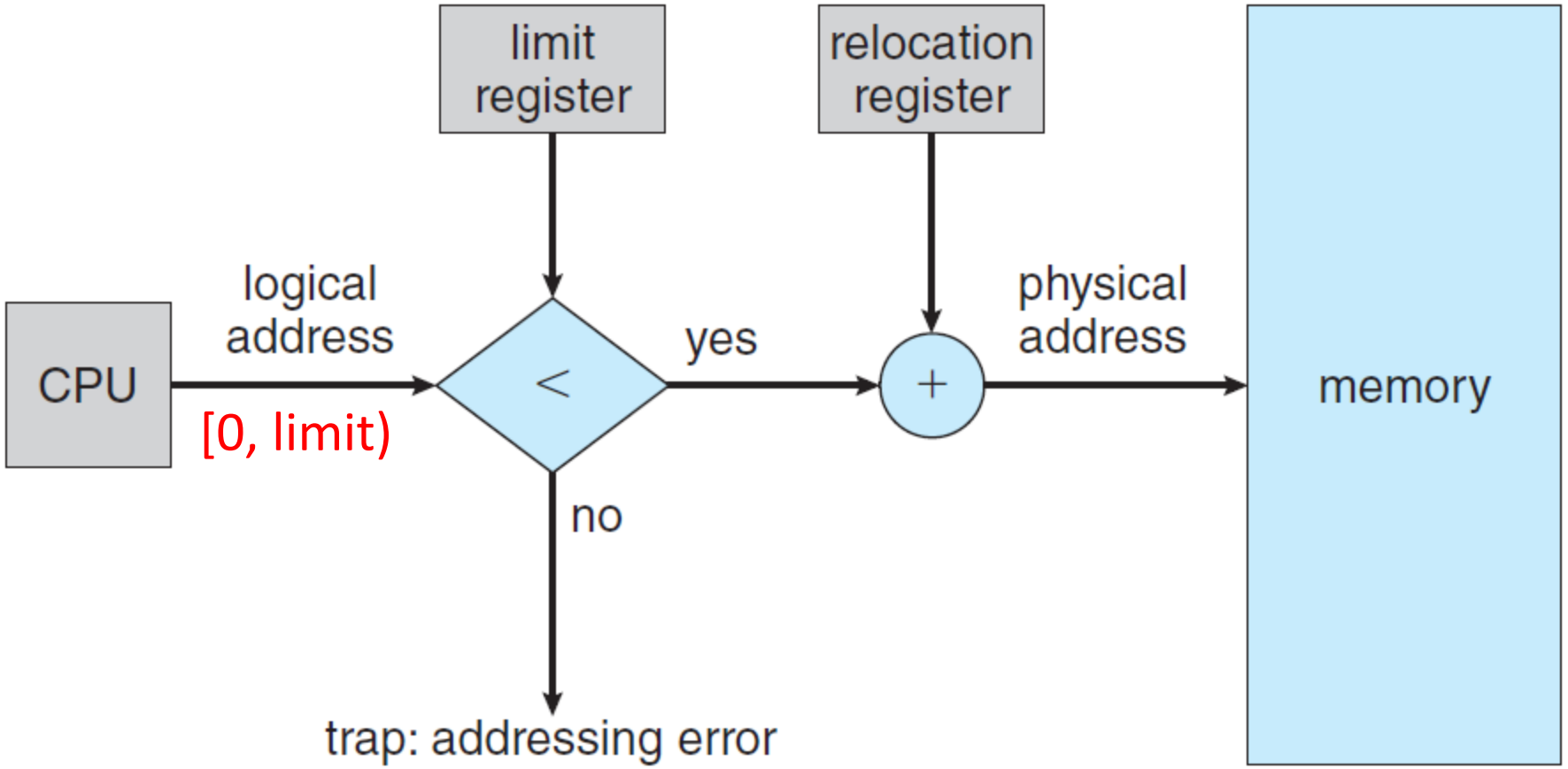
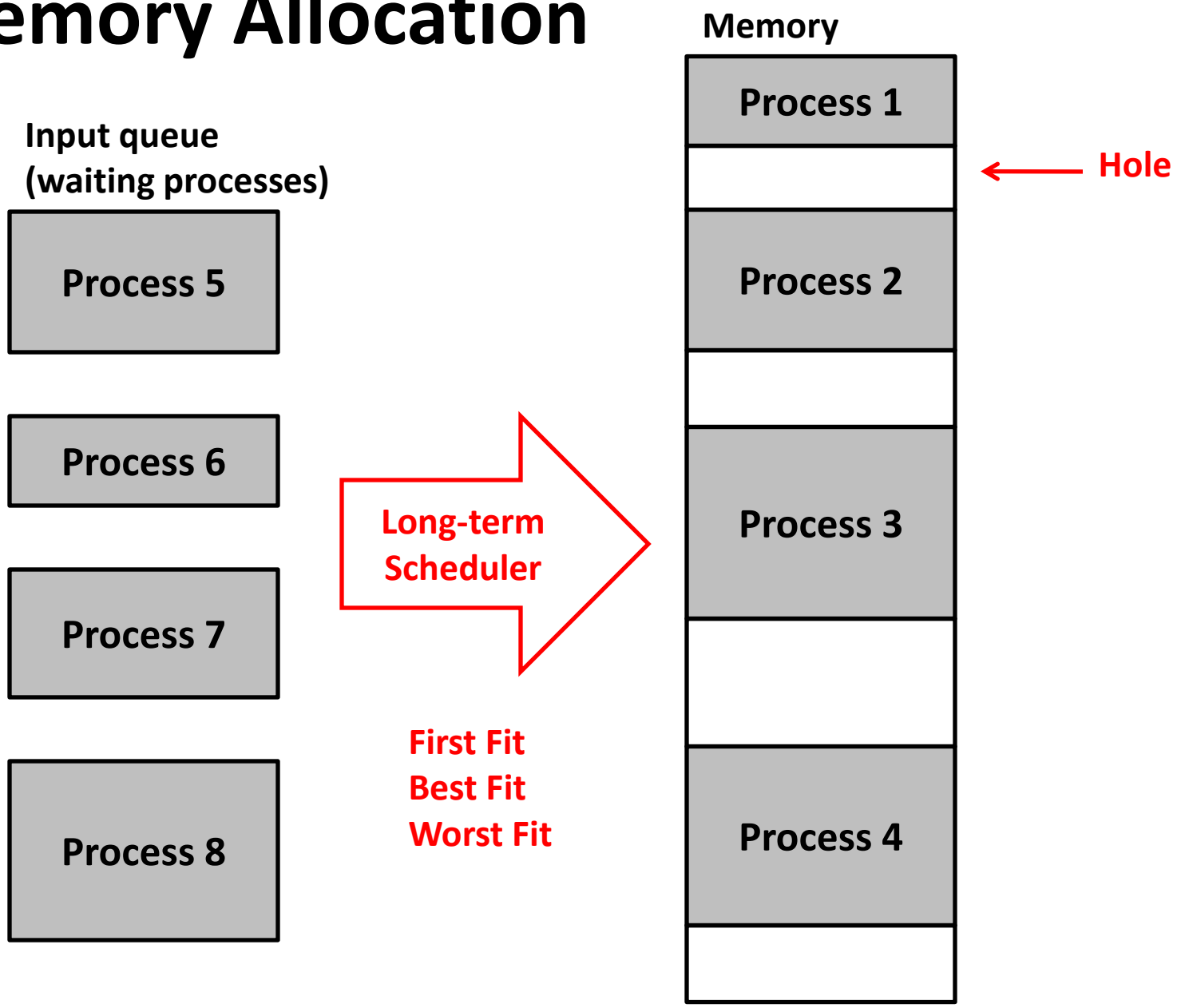


Figure 9.6 Hardware support for relocation and limit registers.

Memory Allocation



Fragmentation

External fragmentation

Holes in memory.

แก้ด้วยวิธี 1 ทำ compaction

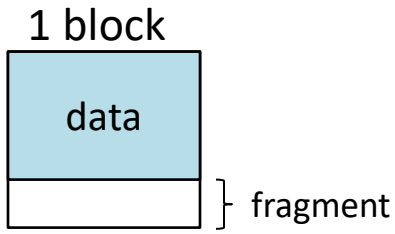
แก้ด้วยวิธี 2 ใช้ non-contiguous memory

50-percent rule (first fit), given *n* blocks allocated, 0.5*n* will be lost.

One-third of memory may be unusable.

Internal fragmentation

เกิดจากการจอง memory เป็น block แล้วใช้ไม่เต็ม block



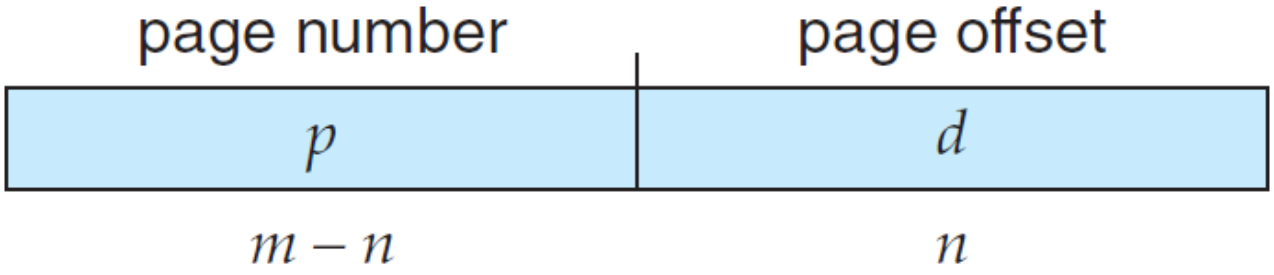
Paging

Frame	fixed-size block on physical memory
Page	fixed-size block on logical memory
Page number (p)	left side of logical address
Page offset (d)	right side of logical address

Page size = frame size

Windows 10 supports page sizes of **4KB** (typical) and 2MB.
Linux also supports two page sizes.

m	number of bits (logical address space)
n	number of bits (page size)



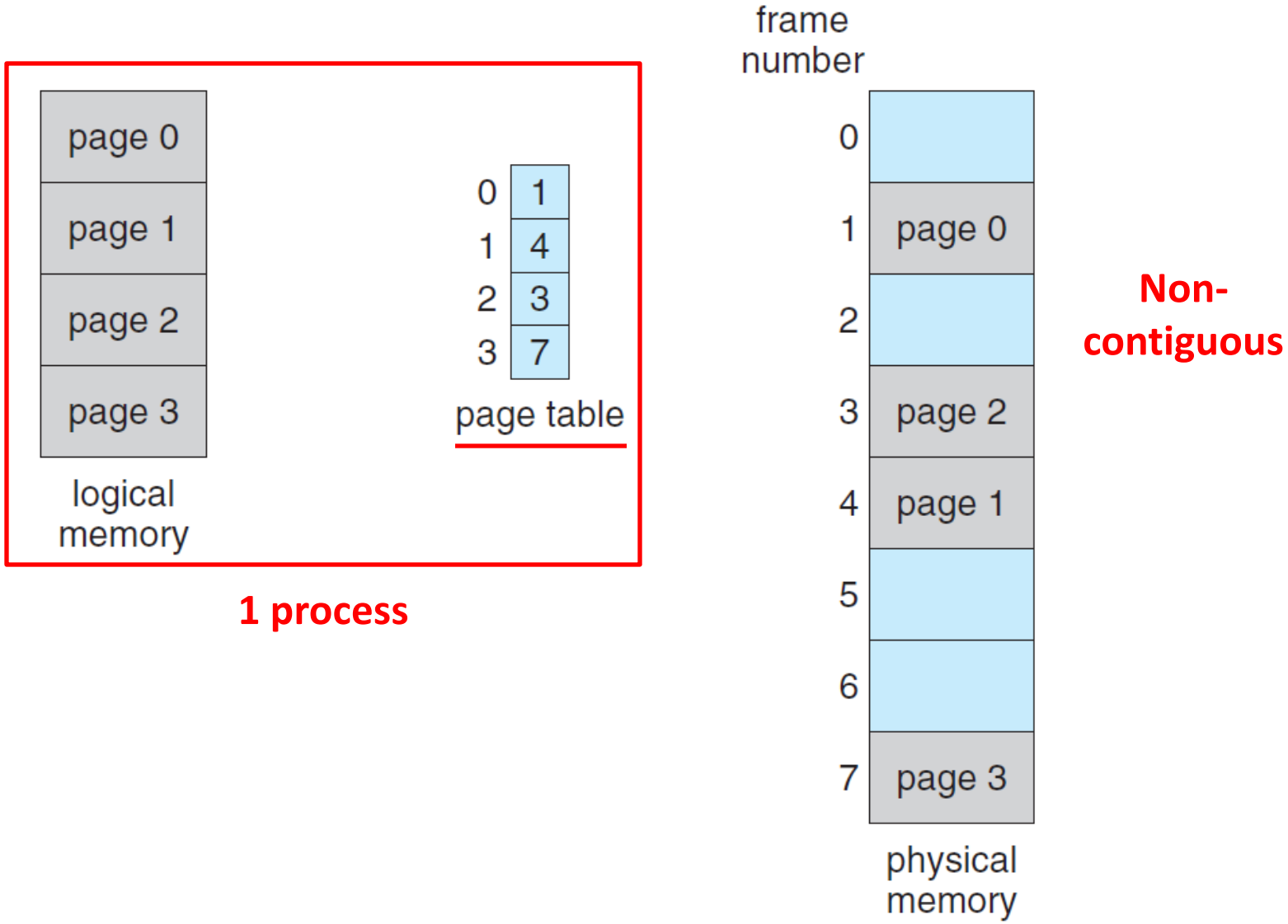


Figure 9.9 Paging model of logical and physical memory.

Page table is in memory, and it is indexed by PTBR (page-table base register) ที่จริงแล้ว page table ก็อยู่ใน memory นั่นเอง ทำให้ทุกครั้งที่โปรแกรมจะ read/write memory ก็ต้องอ่าน page table ก่อน ทำให้ต้องอ่าน memory 2 รอบ

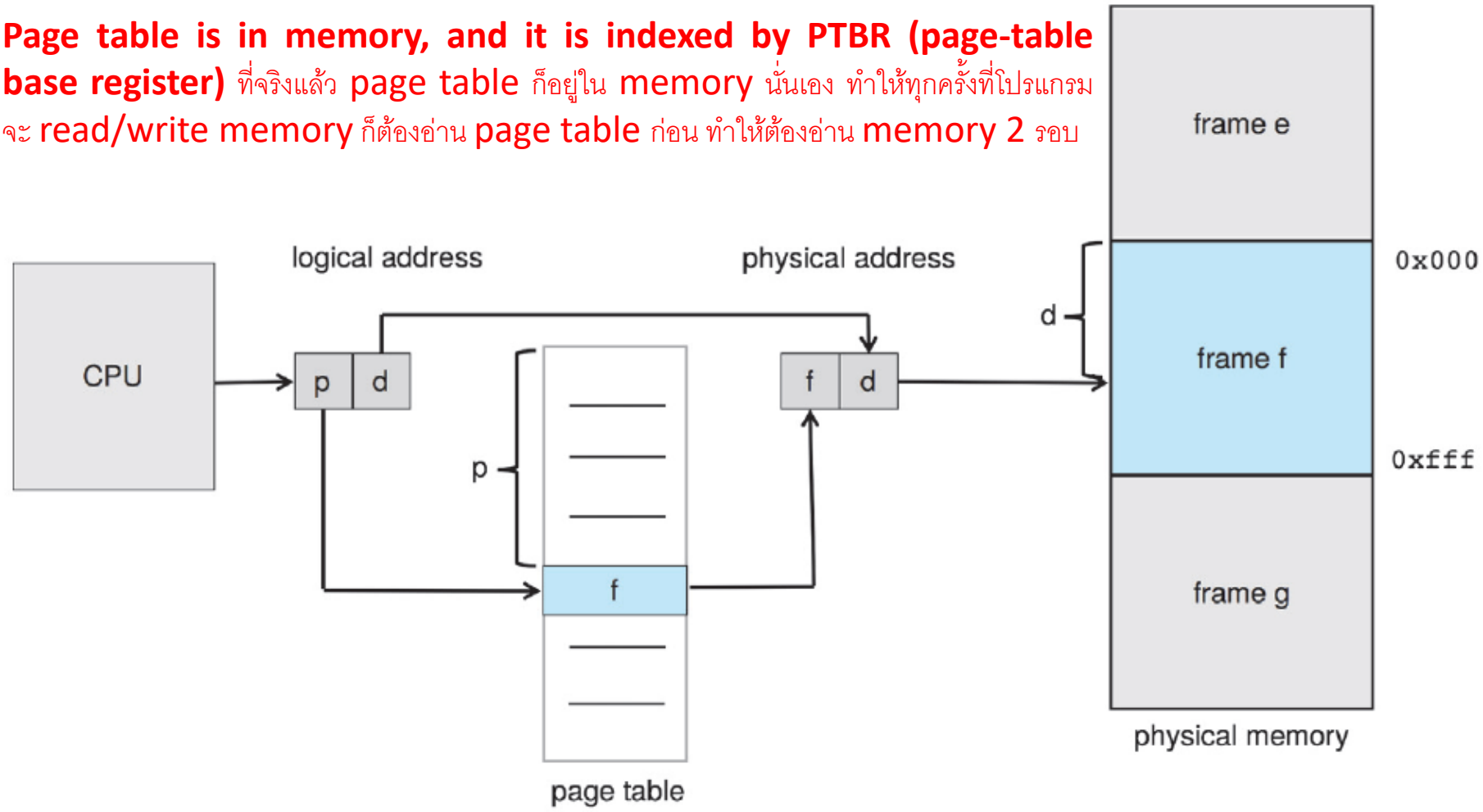
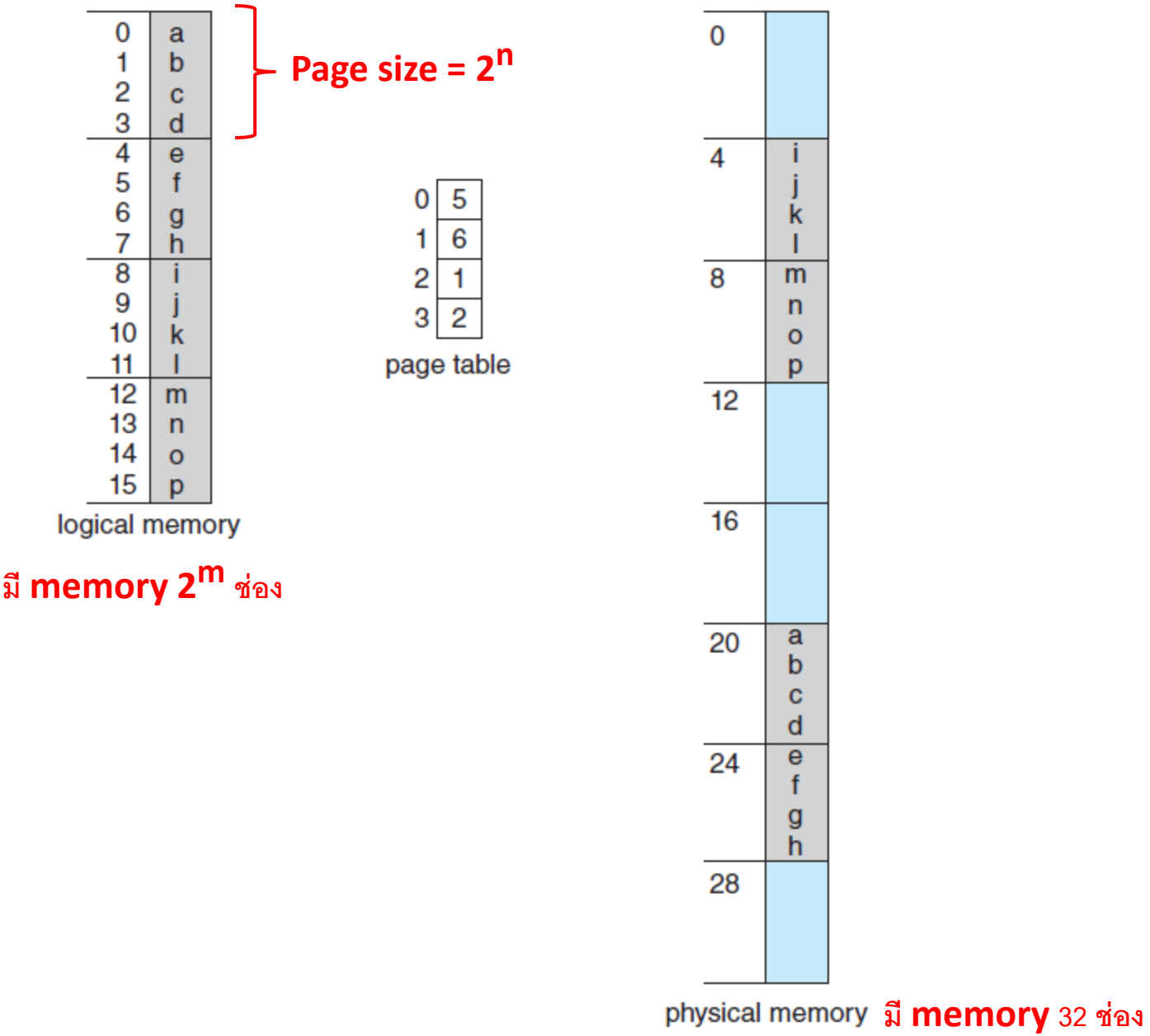


Figure 9.8 Paging hardware.

m = 4, n = 2



มี **memory 2^m** ช่อง

มี **memory** 32 ช่อง

Figure 9.10 Paging example for a 32-byte memory with 4-byte pages.

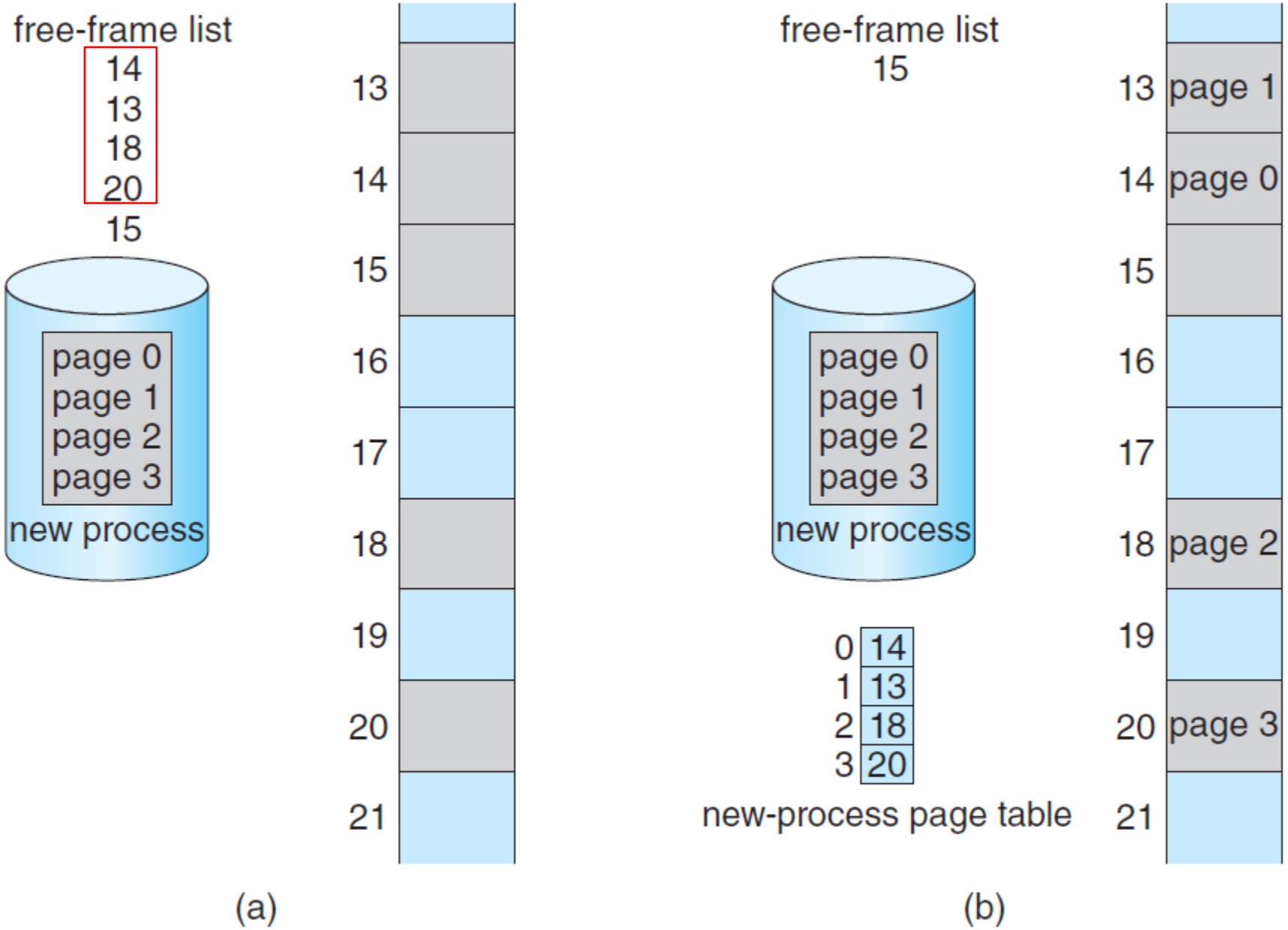


Figure 9.11 Free frames (a) before allocation and (b) after allocation.

Translation Look-aside Buffer (TLB)

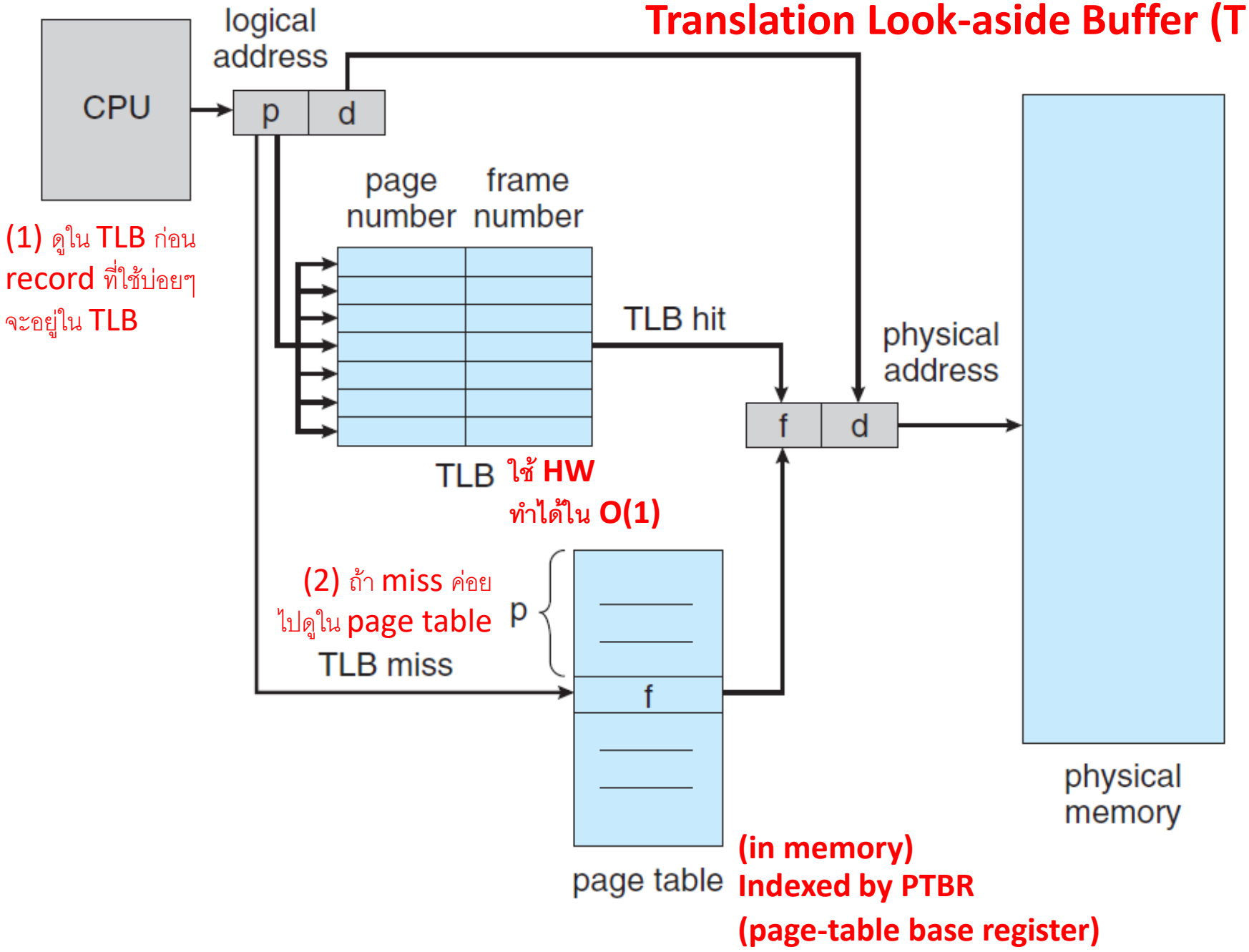


Figure 9.12 Paging hardware with TLB.

Protection

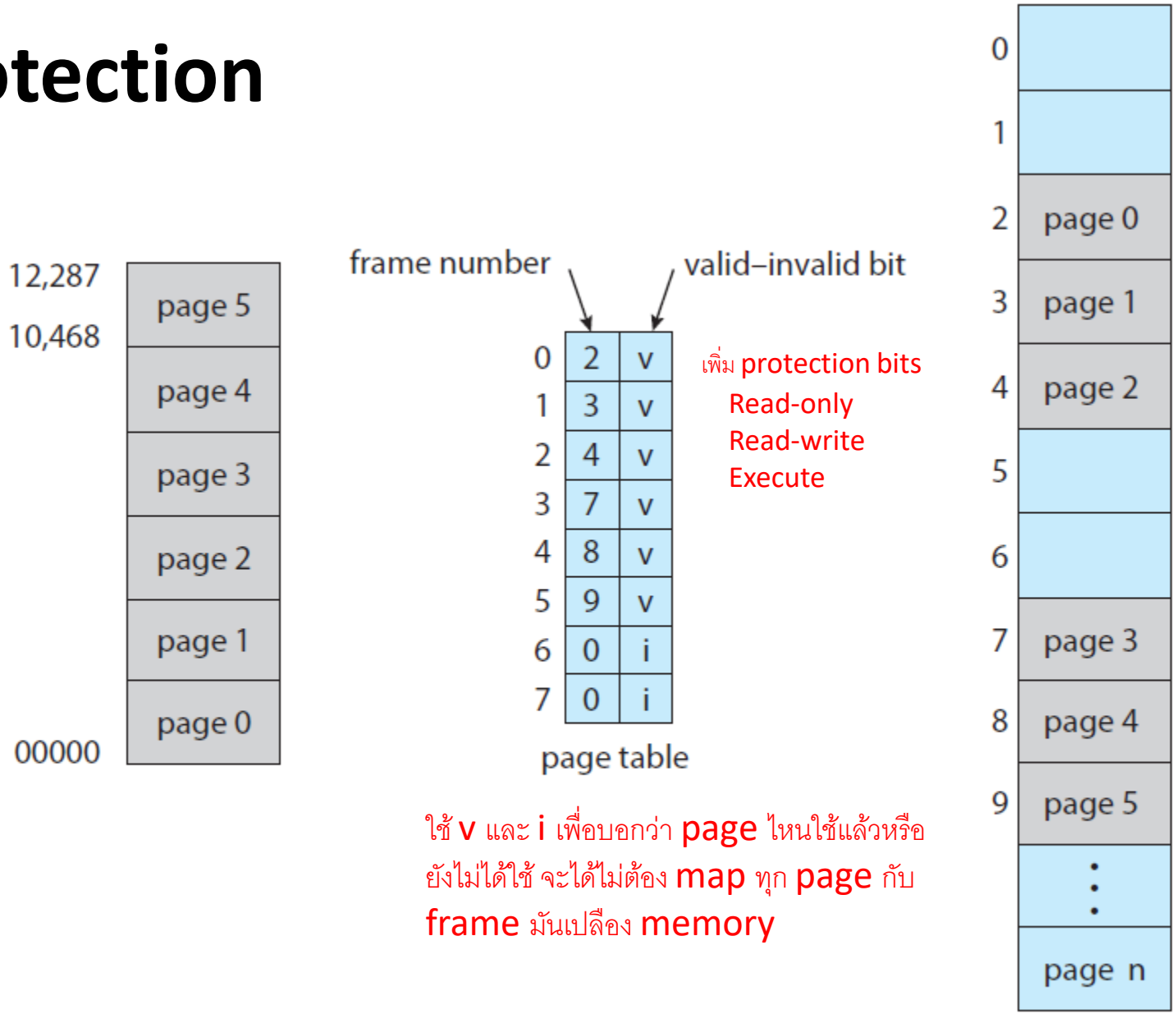


Figure 9.13 Valid (v) or invalid (i) bit in a page table.

If the code is **reentrant code**, however, it can be shared.

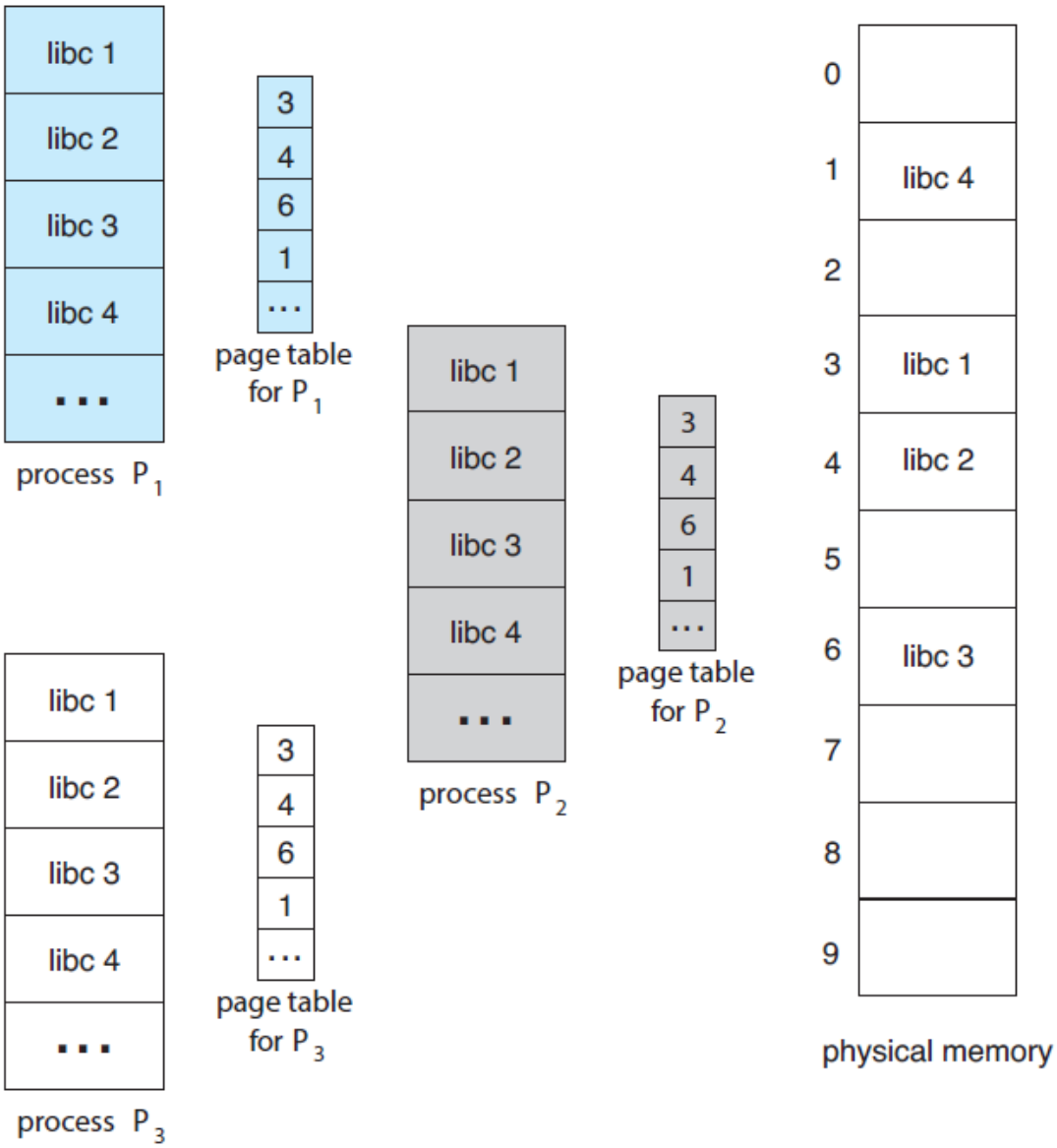
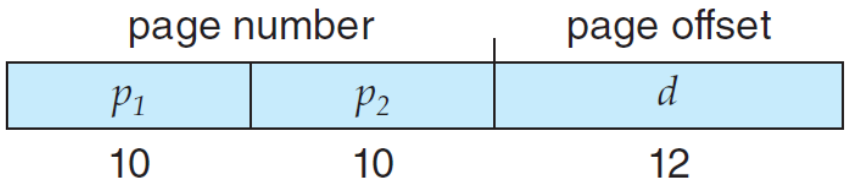


Figure 9.14 Sharing of standard C library in a paging environment.

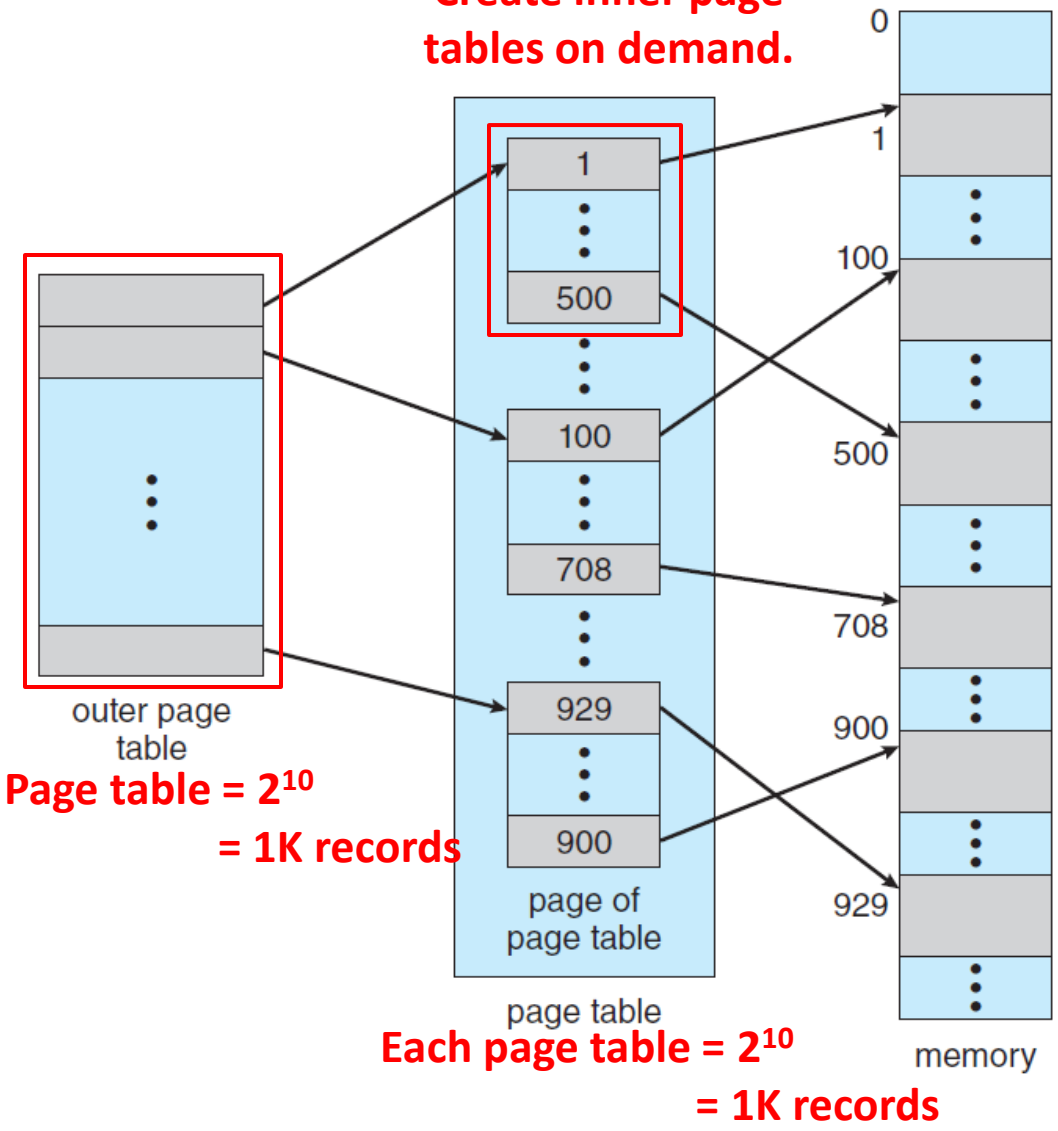
https://en.wikipedia.org/wiki/Reentrancy_(computing)

Hierarchical Paging (32-bit)



1 page = 2^{12} entries

Create inner page tables on demand.



Page table = 2^{10} = 1K records

Each page table = 2^{10} = 1K records

Figure 9.15 A two-level page-table scheme.

No hierarchical paging

page table = 2^{20} = 1 million entries

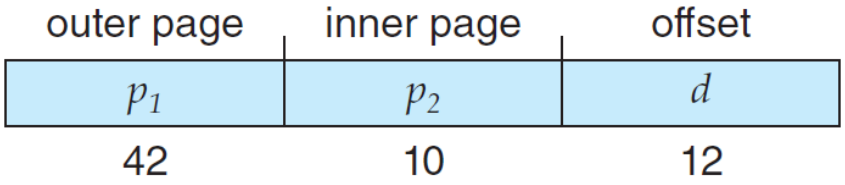
A new process costs 1M.

Hierarchical paging

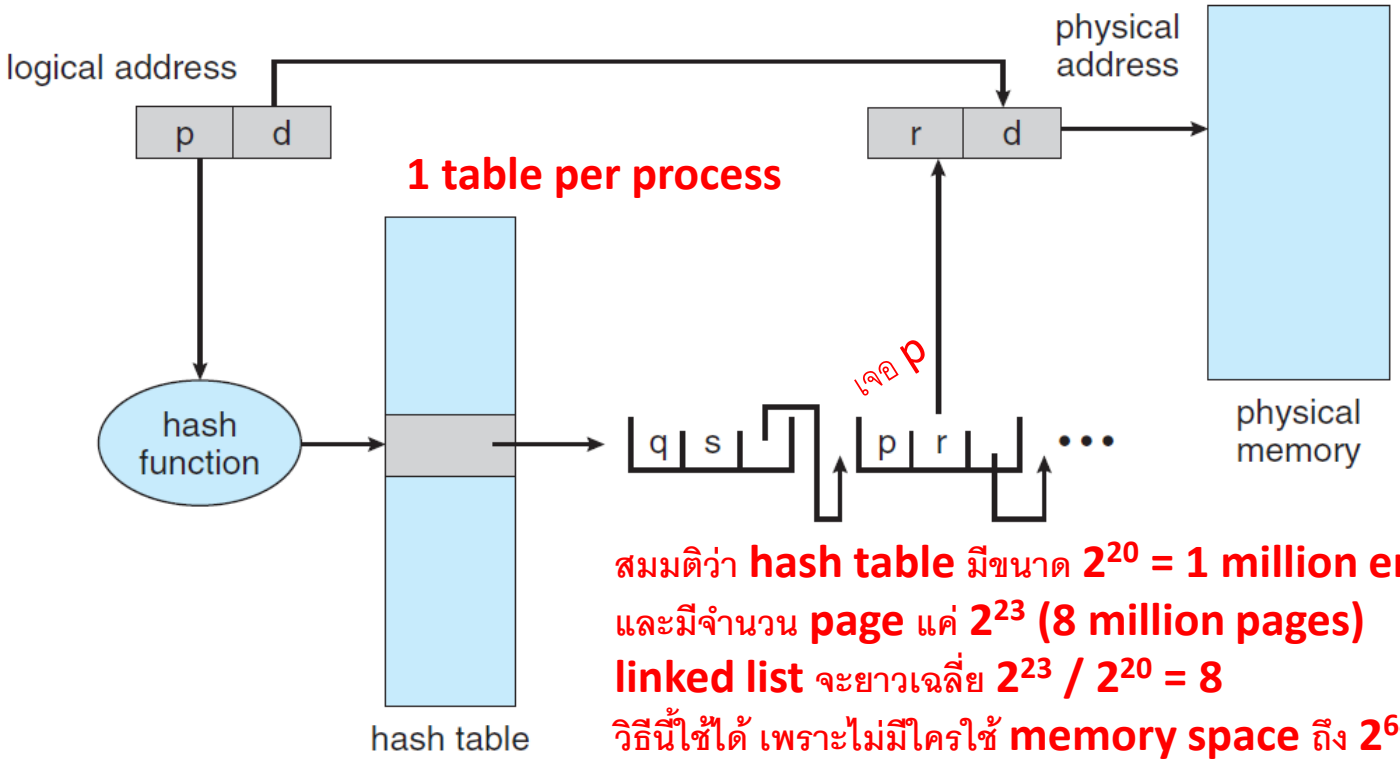
new process = $2^{10} + 2^{10}$ = 2K entries

A new process costs 2K.

Hashed Page Tables (64-bit)



ทำแบบเดิมไม่ได้แล้ว เพราะ 2^{42} ก็ยังใหญ่เกินไป
หรือแบ่งครึ่ง 2^{26} = 64M entries ก็ยังใหญ่อยู่ดี

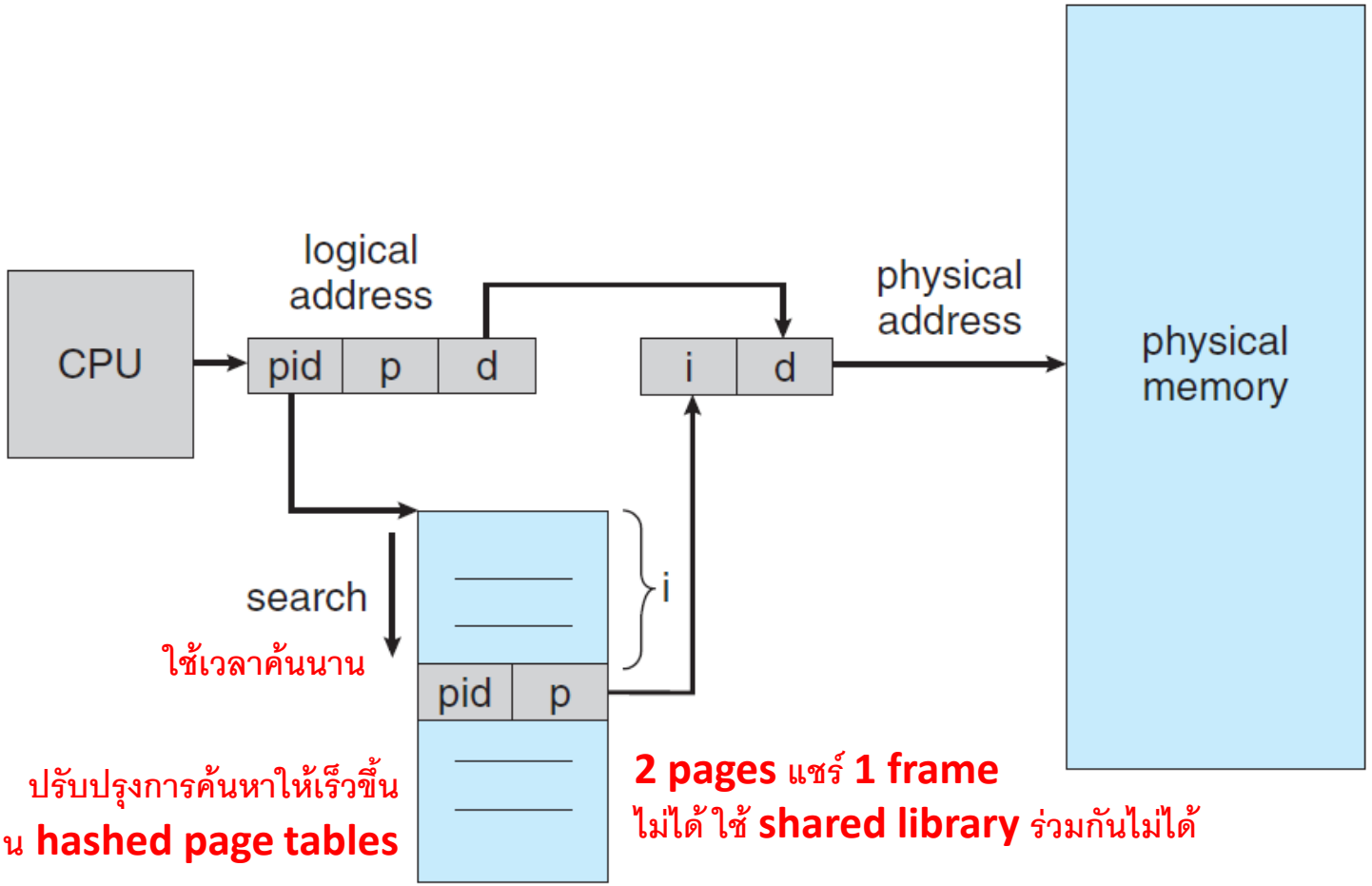


สมมติว่า **hash table** มีขนาด 2^{20} = 1 million entries
และมีจำนวน **page** แค่ 2^{23} (8 million pages)
linked list จะยาวเฉลี่ย $2^{23} / 2^{20} = 8$
วิธีนี้ใช้ได้ เพราะไม่มีใครใช้ **memory space** ถึง 2^{64} จริง ๆ
ถ้าใช้ **clustered page tables** จะทำให้ **linked list** มี **max length** คงที่

Figure 9.17 Hashed page table.

Inverted Page Tables (64-bit)

ทั้งระบบมี **page table** แค่ตารางเดียว ไม่ **per process**



ปรับปรุงการค้นหาให้เร็วขึ้น
โดยทำเหมือน **hashed page tables**

2 pages แชร์ **1 frame**
ไม่ได้ ใช้ **shared library** ร่วมกันไม่ได้

page table ไม่ใหญ่มาก
เพราะไม่มีใครใช้ **memory space** ถึง 2^{64} จริง ๆ

Figure 9.18 Inverted page table.

64-bit UltraSPARC, PowerPC, IBM RT

Segmentation

<segment-number, offset>.

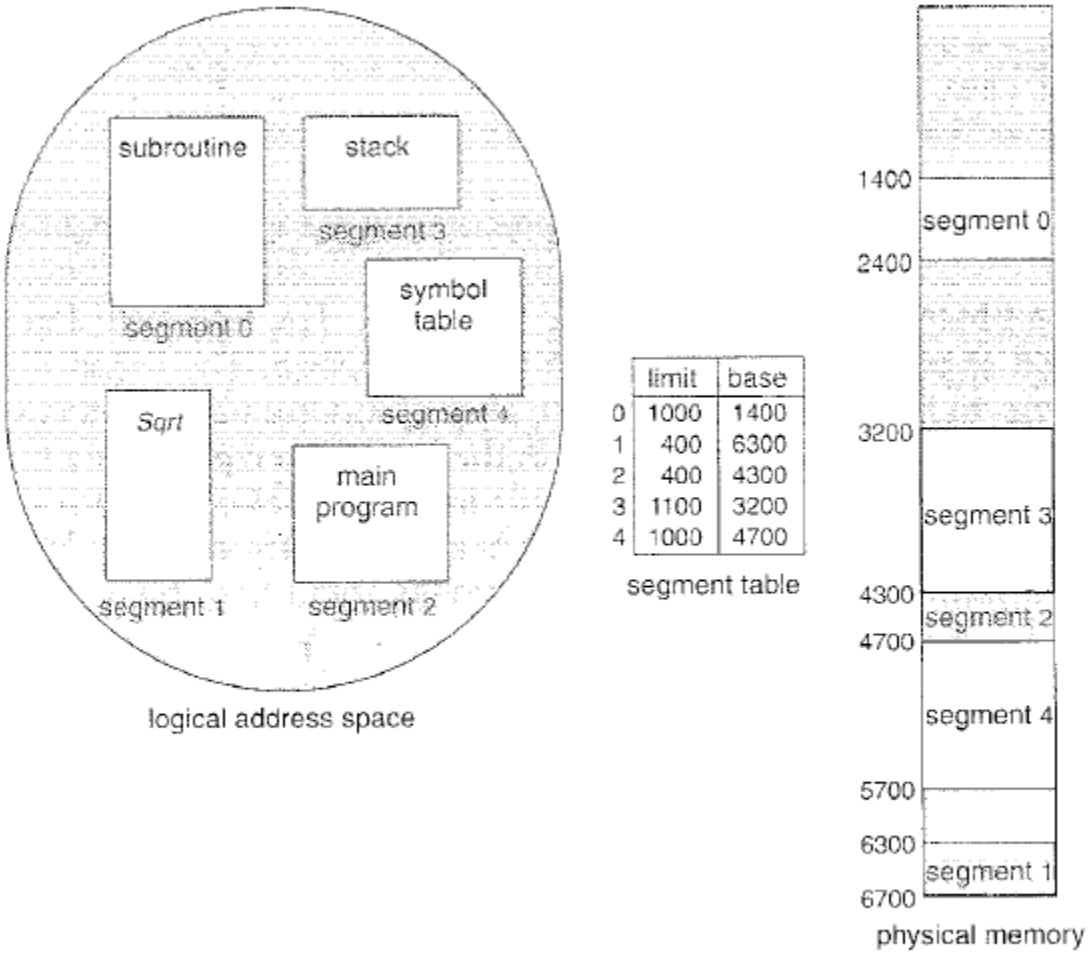


Figure 8.20 Example of segmentation.

รูปนี้มีแค่ใน textbook เล่มเก่า
แต่อธิบายเข้าใจง่ายกว่า

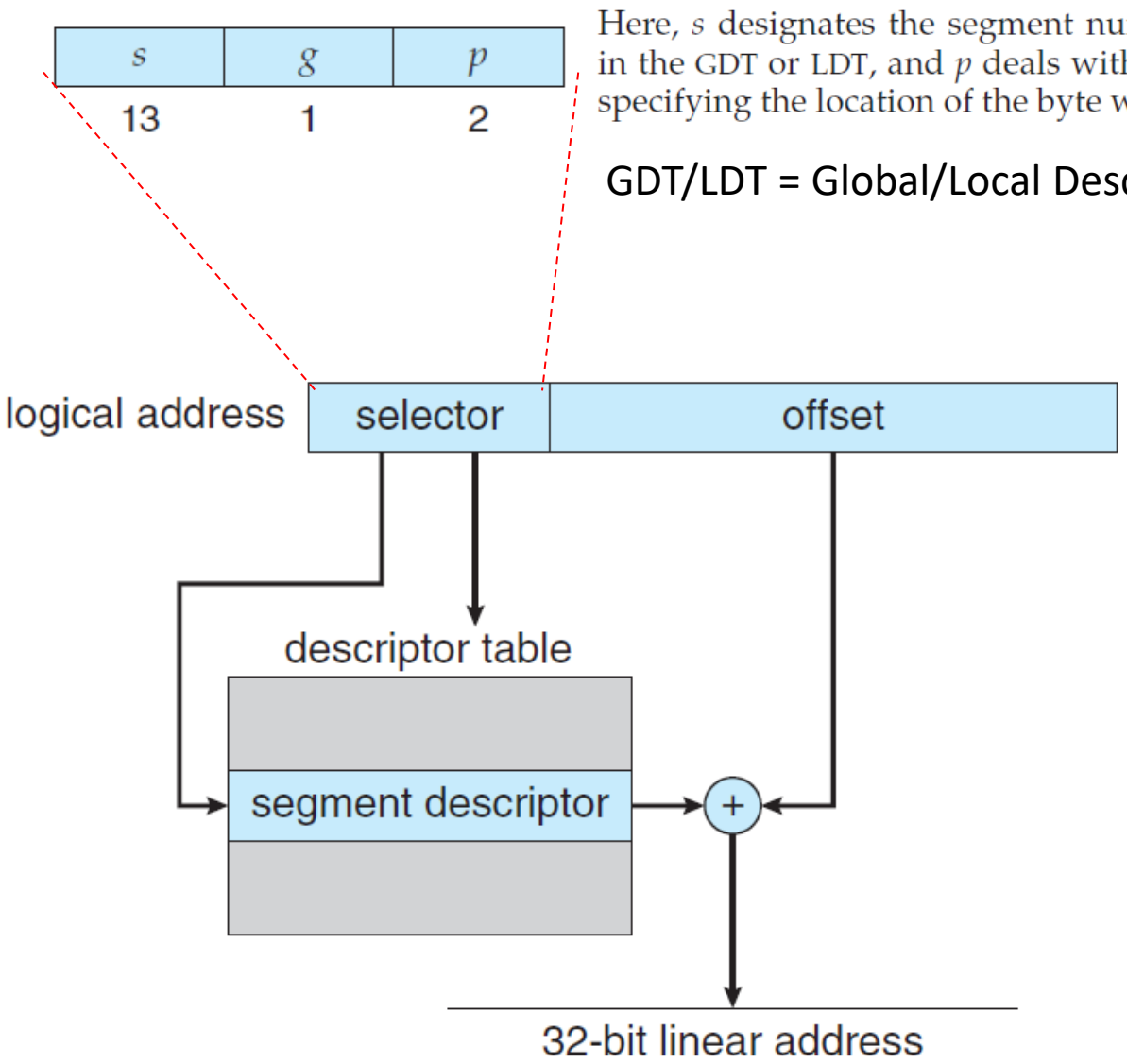


Figure 9.22 IA-32 segmentation.

ข้ามไปก็ได้

IA-32 Architecture

32-bit processors generally compatible with the Intel Pentium® II processor

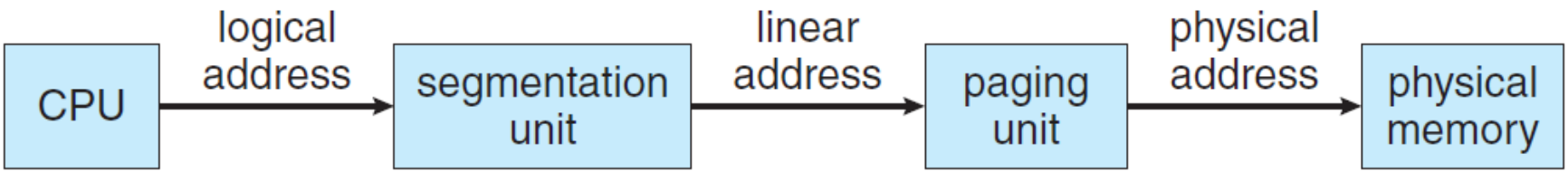


Figure 9.21 Logical to physical address translation in IA-32.

Chapter 9 Main Memory

ลองพิมพ์คำสั่ง getconf PAGE_SIZE บน Linux ค่าที่ได้คืออะไร?

Wired down ใน TLB คืออะไร?

Effective access time ของ memory คืออะไร? คำนวณยังไง?

Reentrant code คืออะไร?