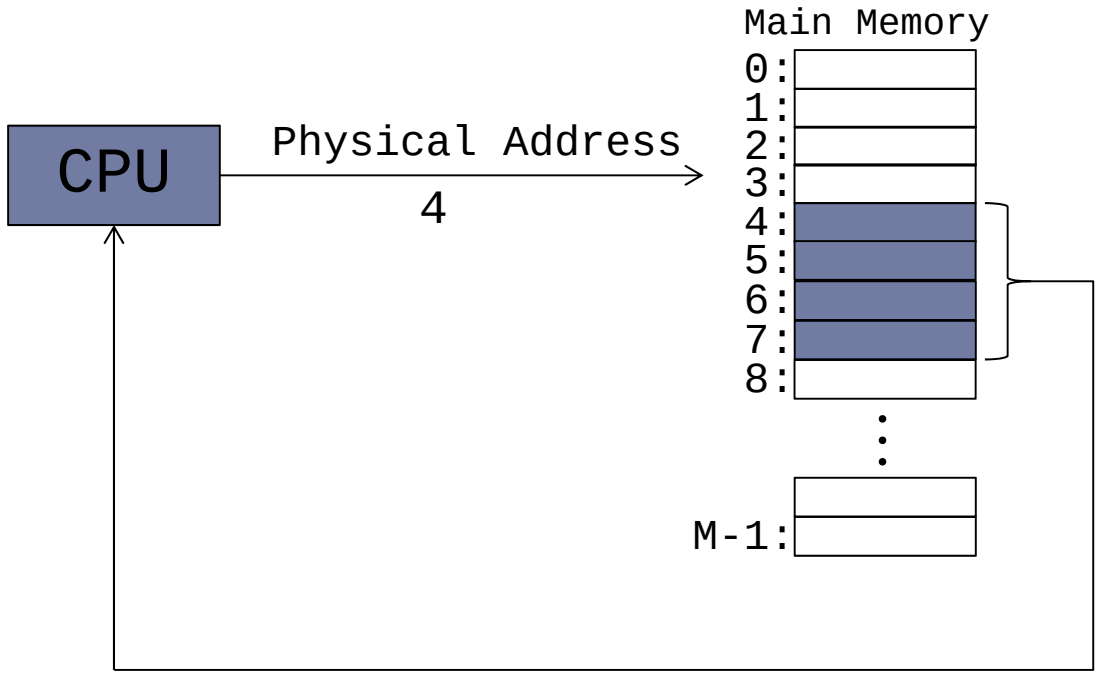
Memory Management
内存管理

Chapter 7

Uniprogrammed OS
单道程序操作系统

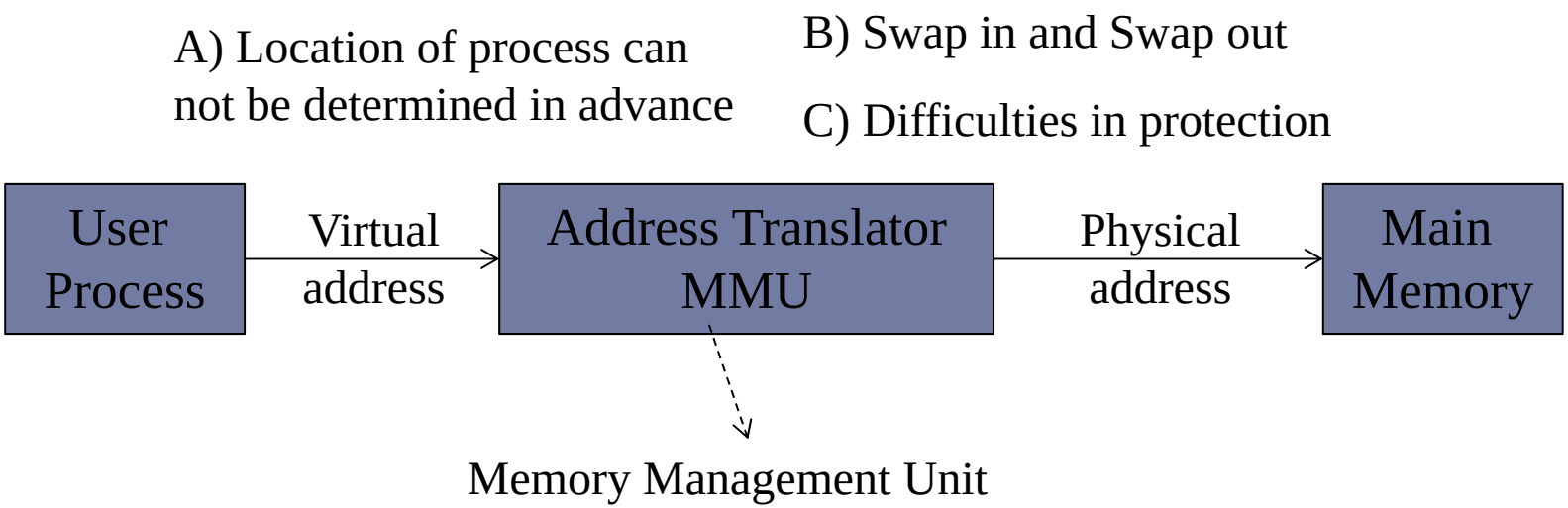
- In uniprogrammed system, two parts in main memory
  - One for the OS (resident monitor, kernel), and
  - The other for the program currently being executed
- Static address translation
  - Physical addresses reference by a user process are pre-computed before its running

Physical Addressing

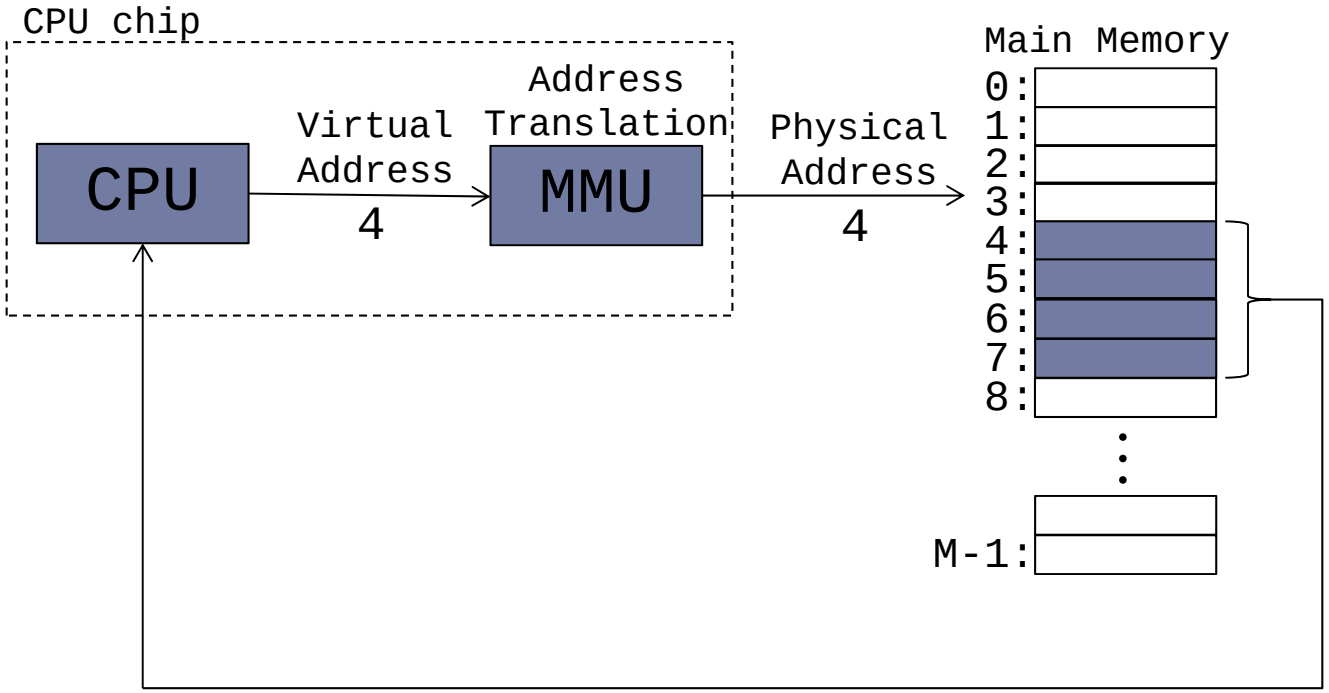


Multiprogrammed OS
多道程序操作系统

- **No free lunch**
  - **all benefits have their own costs**
- **Dynamic address translation**



Virtual Addressing



Address Spaces
地址空间

No Memory Abstraction

- **Every program simply saw the physical memory**
  - **An instruction MOV REGISTER1, 1000 moves the contents of physical memory location 1000 to REGISTER1.**
  - **It was very hard to have two running programs at the same time: one process would erase the value the second process wrote.**

Memory Abstraction: Address Space

- **What is address space?**
  - An address space is the set of addresses that a process can use to address memory.
  - Each process has its own address space, independent of those belonging to other processes
    - Except in some special cases where processes want to share their address spaces.

Address Spaces in Other Contexts

- Telephone number: the address space runs from 0000000 to 9999999
- I/O ports on the Pentium: the address space runs from 0 to 16383.
- IPv4 addresses are 32-bit numbers: the address space runs from 0 to 2^{32}-1.
- Address spaces do not have to be numeric

Memory Management
存储管理

- **Modern memory manager exploits *memory hierarchy*, which normally holds multiple copies of information**
- **Memory needs to be allocated efficiently to pack as many processes into memory as possible, in order to *increase CPU utilization*.**

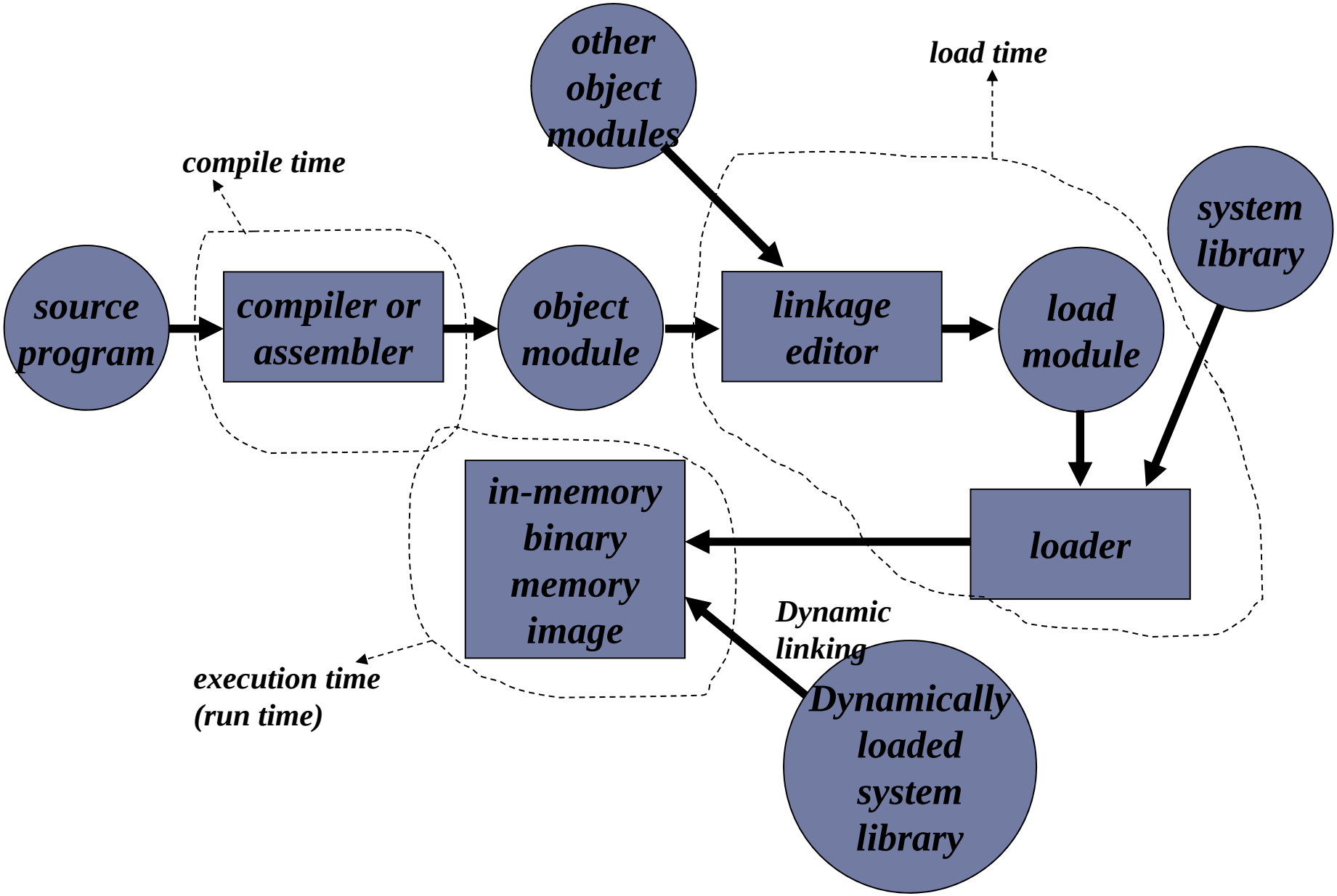
Main Contents

- Background Knowledge
- Contiguous Memory Allocation (连续内存分配)
- Non-contiguous Memory Allocation
  - Paging
  - Segmentation

Background Knowledge

Example: Multistep Processing of a User Program

- **A user program will go through several steps before being executed**
- **Addresses may be represented in different ways during these steps**
  - *Addresses in the source program are generally symbolic*
  - *A compiler will typically **bind** these symbolic addresses to relocatable addresses (such as "14 bytes from the beginning of this module)*
  - *The linkage editor or loader will in turn **bind** these relocatable addresses to absolute addresses (such as 74014)*
- ***Each binding is a mapping from one address space to another***


other object modules
load time
compile time
system library
source program
compiler or assembler
object module
linkage editor
load module
in-memory binary memory image
loader
Dynamic linking
execution time (run time)
Dynamically loaded system library

Loading
装入

- *Loader (装入程序) places the load module (装入模块) in main memory*
- *In general, three approaches can be taken:*
  - *Absolute loading (绝对装入)*
  - *Relocatable loading (可重定位装入)*
  - *Dynamic run-time loading (动态运行时装入)*

Absolute Loading
绝对装入

- ***An absolute loader requires that a given load module always be loaded into the same location in main memory***
  - *In the load module presented to the loader, all address references must be to absolute main memory addresses*
  - *The assignment of specific address values to memory references within a program can be done either by the programmer or at compile or assembly time.*

Relocatable Loading
可重定位装入

- ***The loader can place the module in the desired location***
  - *All the memory references within the load module are **expressed relative to some known points** (such as the start of the load module)*
  - *If the module is to be loaded beginning at location x, then the loader must **simple add x to each memory reference** as it loads the module into memory*

Dynamic Run-Time Loading
动态运行时装入

- ***To be able to swap a process image back into different locations at different time, we have to defer the calculation of an absolute address until it is actually needed at run time***
  - ***The load module is loaded into main memory with all memory references in relative form.***
  - ***The absolute address must be done by special hardware, in order not to degrade performance***
- ***Dynamic address calculation provides complete flexibility***

Linking
链接

- *The function of linker is to take as input a collection of object modules (目标模块) and produce a load module (装入模块)*
- *The linker creates a single load module that is the contiguous joining of all of the object modules*
  - *Each intra-module reference must be changed from a symbolic address to a reference to a location within the overall load module.*

Linkage Editor
链接编辑器

- ***A linker that produces a relocatable load module is often referred to as linkage editor***
  - ***Each compiled or assembled object module is created with references relative to the beginning of the object module***
  - ***All of these object modules are put together into a single relocatable load module with all references relative to the original of the load module***

Dynamic Linker
动态链接器

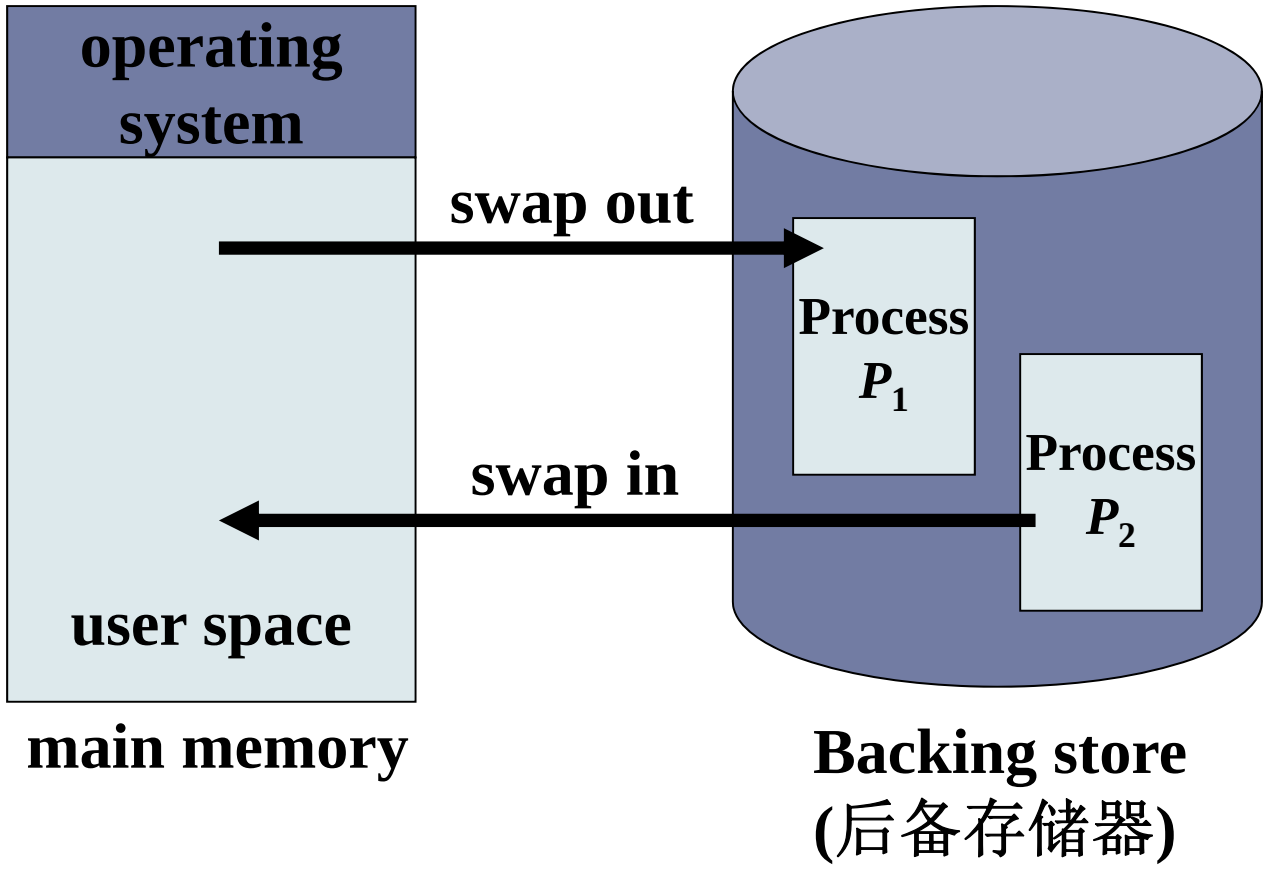
- ***Load-time dynamic linking (装入时动态链接)***
  - *The load module (application module) to be loaded is read into memory*
  - *Any reference to an external module (target module) causes the loader to find the target module, load it, and alter the reference to a relative address in memory from the beginning of the application module*
- ***Run-time dynamic linking (运行时动态链接)***
  - *Some of the linking is postponed until execution time. (external references to target modules remain in the loaded program)*
  - *When a call is made to the absent module, the OS locates the module, loads it, and links it to the calling module.*

Swapping (交换)
Why Swapping?

- **Enough processes should be kept in main memory to keep the CPU busy.**
  - A process need to be in main memory to be executed
- **What if the main memory is not enough to hold all the currently active memory**
  - Swapping (交换)
  - Virtual memory (虚拟存储器)

Swapping vs Virtual Memory

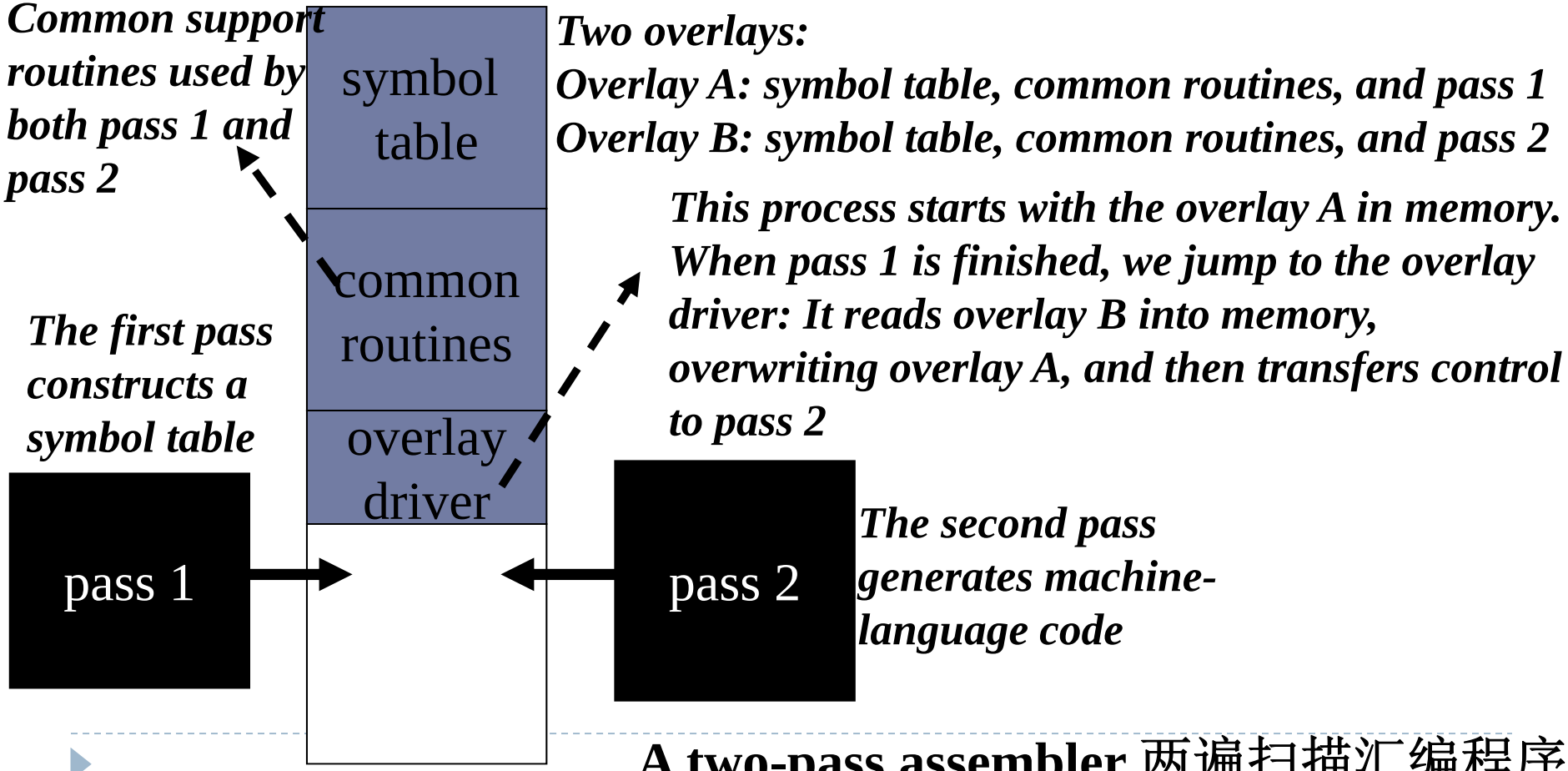
- **Swapping is the simpler solution:**
  - It brings in each process *in its entirety*, running it for a while, then putting it back on the disk
  - Swapping strategy makes use of dynamic relocation, which influenced the development of virtual memory
- **Virtual memory is a more complex solution:**
  - It allows programs to run even when they are only *partially* in main memory
  - Modern operating systems usually adopt virtual memory as the memory management strategy


operating system
swap out
Process P_1
Process P_2
swap in
user space
main memory
Backing store
(后备存储器)

Overlay
覆盖

- **Overlay is to enable a process to be larger than the amount of main memory allocated to it.**
  - **The idea of overlay is to keep in memory only those instructions and data that are needed at any given time.**
- **Overlays do not require any special support from the operating system**
  - **The programmer must design and program the overlay structure properly.**

Overlay
An Example



A two-pass assembler 两遍扫描汇编程序

Contiguous Memory Allocation
连续内存分配

Introduction

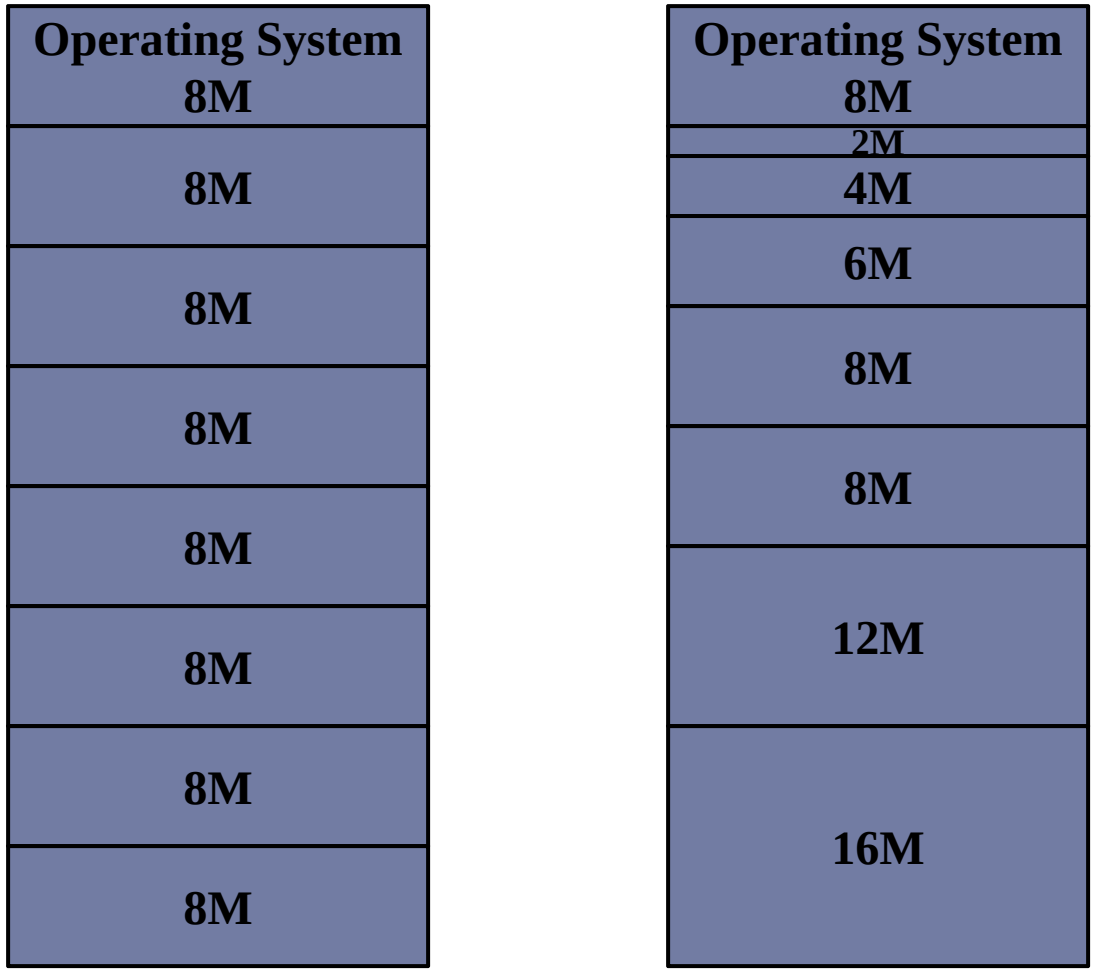
- **Memory allocation**
  - Fixed partitioning
  - Dynamic partitioning
- **Fragmentation**
  - Internal fragmentation
  - External fragmentation
- **Address translation and memory protection**

Memory Partitioning

- Fixed Partitioning (固定分区)
- Dynamic Partitioning (动态分区)
- Buddy System (伙伴系统)
- Relocation (重定位)

Fixed Partitioning (1)
Two Alternatives

- **Fixed partitioning**
  - **Partitions the available memory into regions with *fixed boundaries***
  - ***Each partition can hold only one process.***
- **Do these regions be all of the same size?**
  - ***Yes: Equal-size partitions***
  - ***No: Unequal-size partitions***



Equal-size partitions

Unequal-size partitions

Fixed Partitioning (2)
Difficulties with Equal-Size Partitions

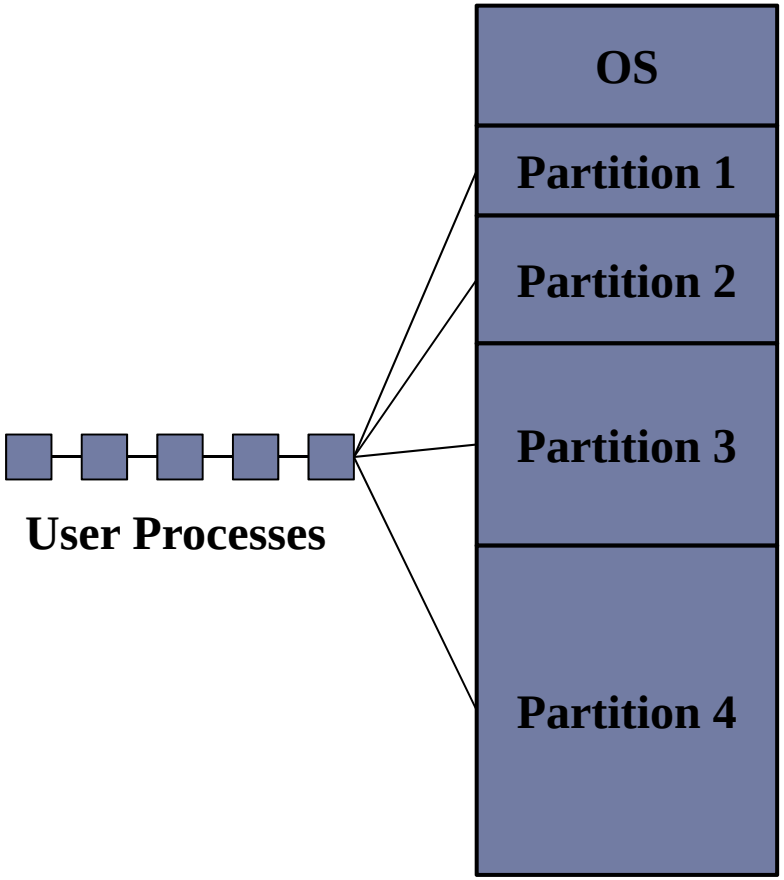
- **A program may be too big to fit into a partition**
  - Programmer must design the program with the use of overlay
- **Main memory utilization is extremely inefficient**
  - Any program, no matter how small, occupies an entire partition
  - **Internal fragmentation** (內部碎片):There is wasted space internal to a partition due to the fact that the block of data loaded is smaller than the partition.
- **Both of these problems can be lessoned by using unequal-size partitions**

Fixed Partitioning (3)
Placement (放置) Algorithm

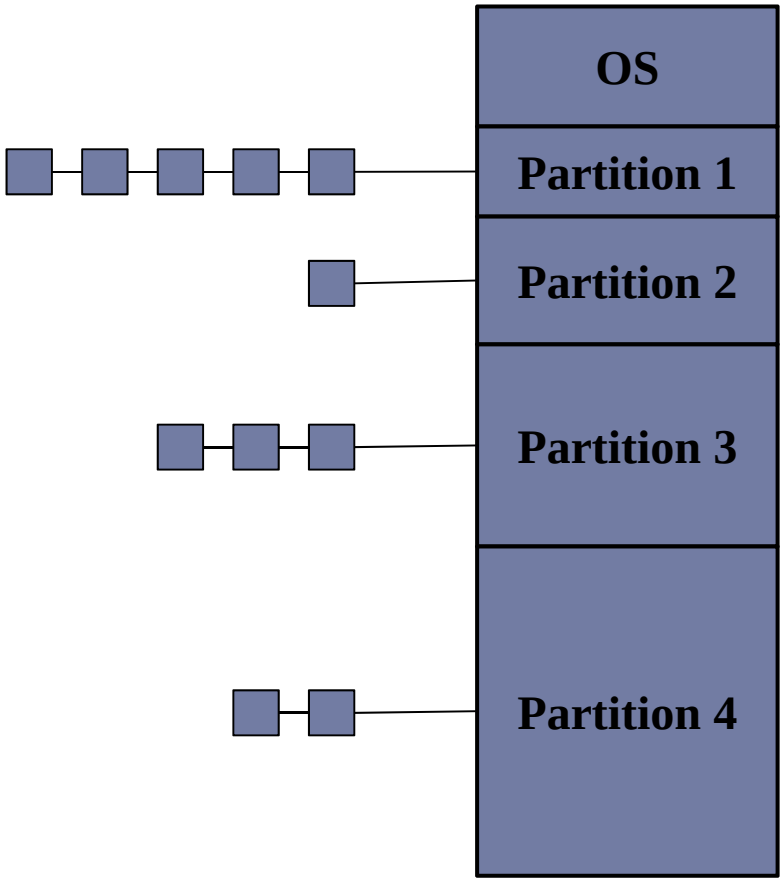
- **With equal-size partitions: The placement is trivial, because all partitions are of equal size, it does not matter which partition is used**
  - **As long as there is any available partition, a process can be loaded into it.**
  - **If all partitions are occupied with processes that are not ready to run, then one of these processes must be swapped out to make room for a new process.**

Fixed Partitioning (4)
Placement (放置) Algorithm

- **With unequal-size partitions: Two possible ways to assign processes to partitions**
  - The simplest way is to assign each process to the **smallest partition** within which it will fit
    - ***A scheduling queue is needed for each partition***, to hold swapped-out processes destined for that partition
    - To minimize wasted memory within a partition (internal fragmentation) – Seems optimum from the point of view of an individual partition, but not from the point of view of the whole system
  - A single queue for all processes (**smallest available partition**)



Single Queue



Single Queue Per Partition

Fixed Partitioning (5)
Advantages

- Relatively simple
- Require minimal operating system software and processing overhead

Fixed Partitioning (6)
Disadvantages

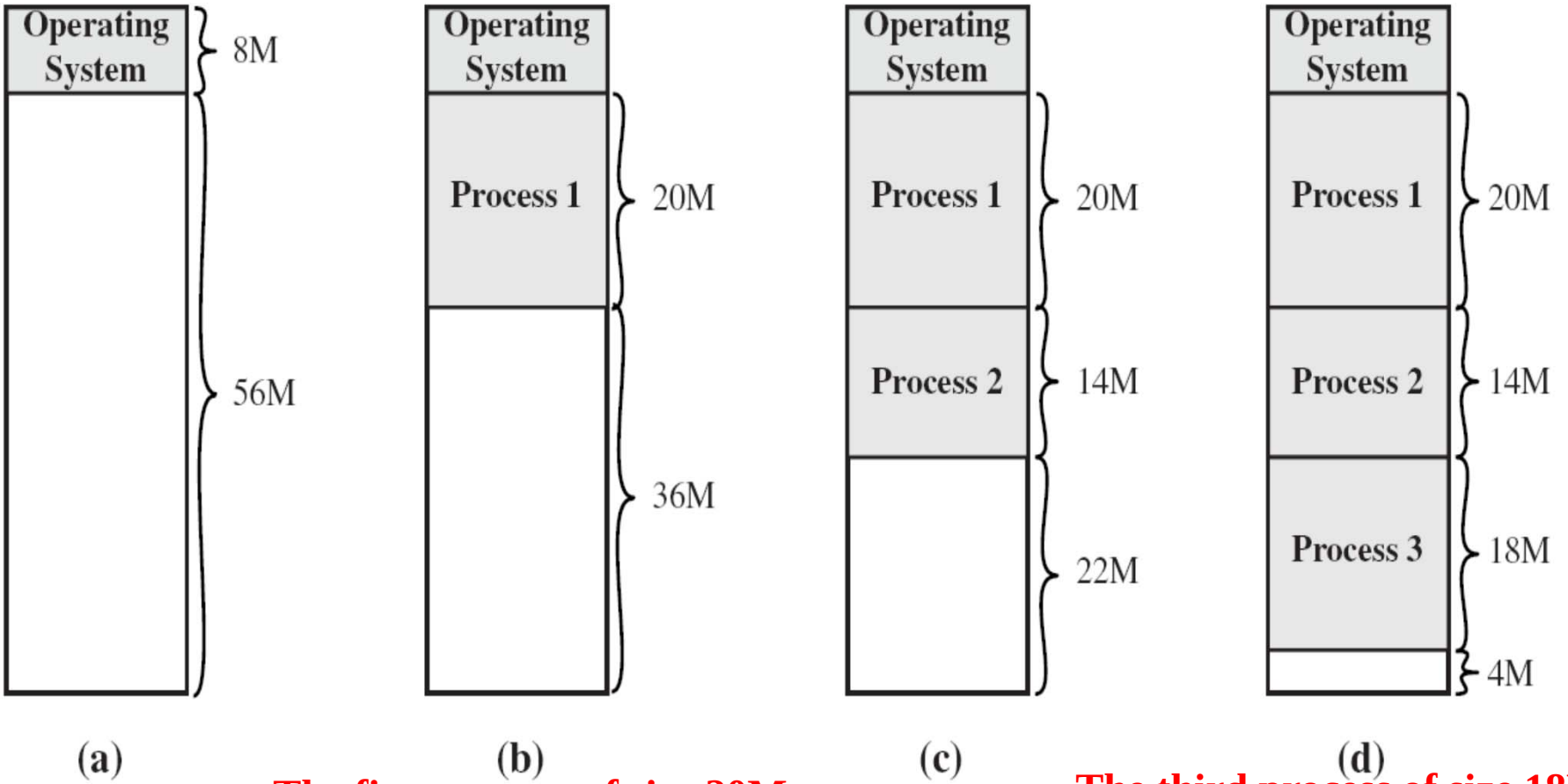
- **Limits the number of active (not suspended) processes in the system**
- **Memory space is used inefficiently if there is a poor match between available partition sizes and process sizes**
  - **However, it may be reasonable if the main storage requirement of all jobs is known beforehand**

Dynamic Partitioning (1)

- **With dynamic partitioning, partitions are of *variable length and number***
  - **Process is allocated exactly as much memory as it requires.**
- **This method start out well, but eventually lead to a lot of small holes in the memory.**
  - **It is called external fragmentation (外部碎片): referring to the fact that the memory external to all partitions becomes increasingly fragmented**

64M main memory, where
8M is occupied by the OS

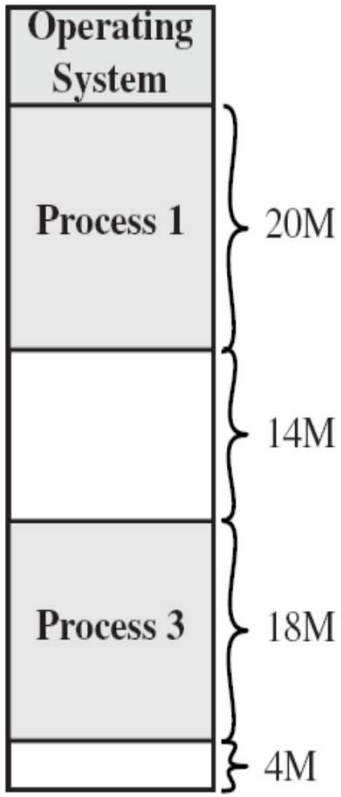
The second process of size
14M is loaded



The first process of size 20M
is loaded

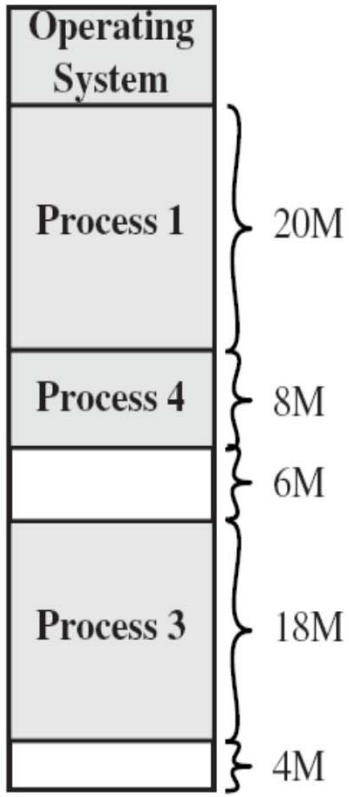
The third process of size 18M
is loaded

The hole at the end of memory is too small for the 4th process of 8M size.



(e)

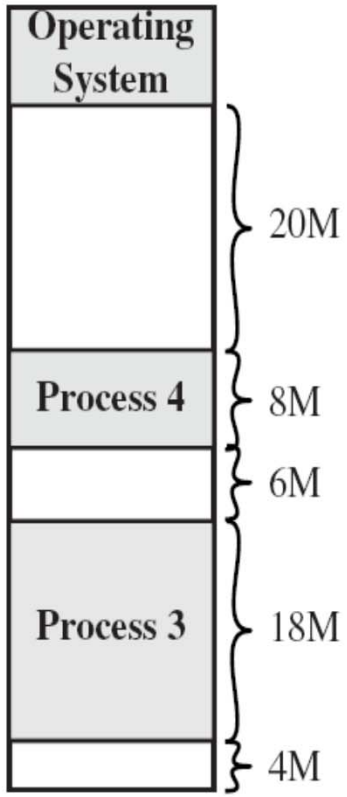
None of the 1st, 2nd, and 3rd processes is ready.
The second process is swapped out.



(f)

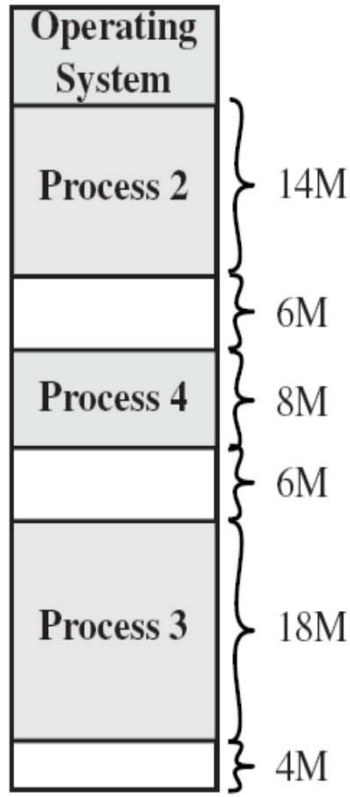
The 4th process is loaded into memory.

Another hole is created



(g)

None of the 1st, 3rd, and 4th processes is ready, while 2nd is in ready-suspend state.
The 1st process is swapped out.



(h)

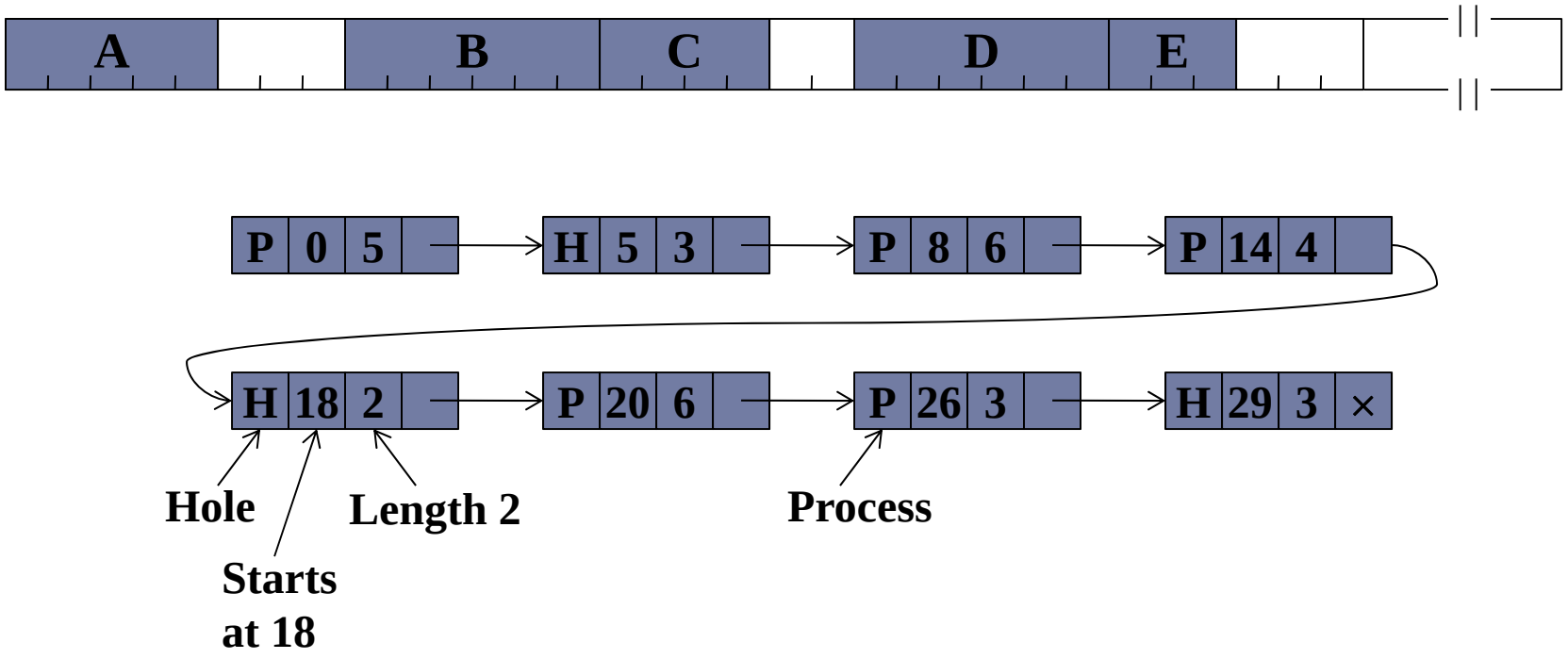
The 2nd process is swapped back in.

Yet another hole is created

Dynamic Partitioning (2) Compaction (紧凑)

- ***Compaction*** **is one technique for overcoming external fragmentation**
  - *It shifts processes so that they are contiguous and all free memory is in one block*
- ***It is a time-consuming procedure and wasteful of CPU time.***
  - *It implies the needs for a dynamic relocation capability so that it is possible to move a process in main memory*

Dynamic Partitioning (3)
Memory Management with Linked List



Dynamic Partitioning (4) Allocation Algorithms

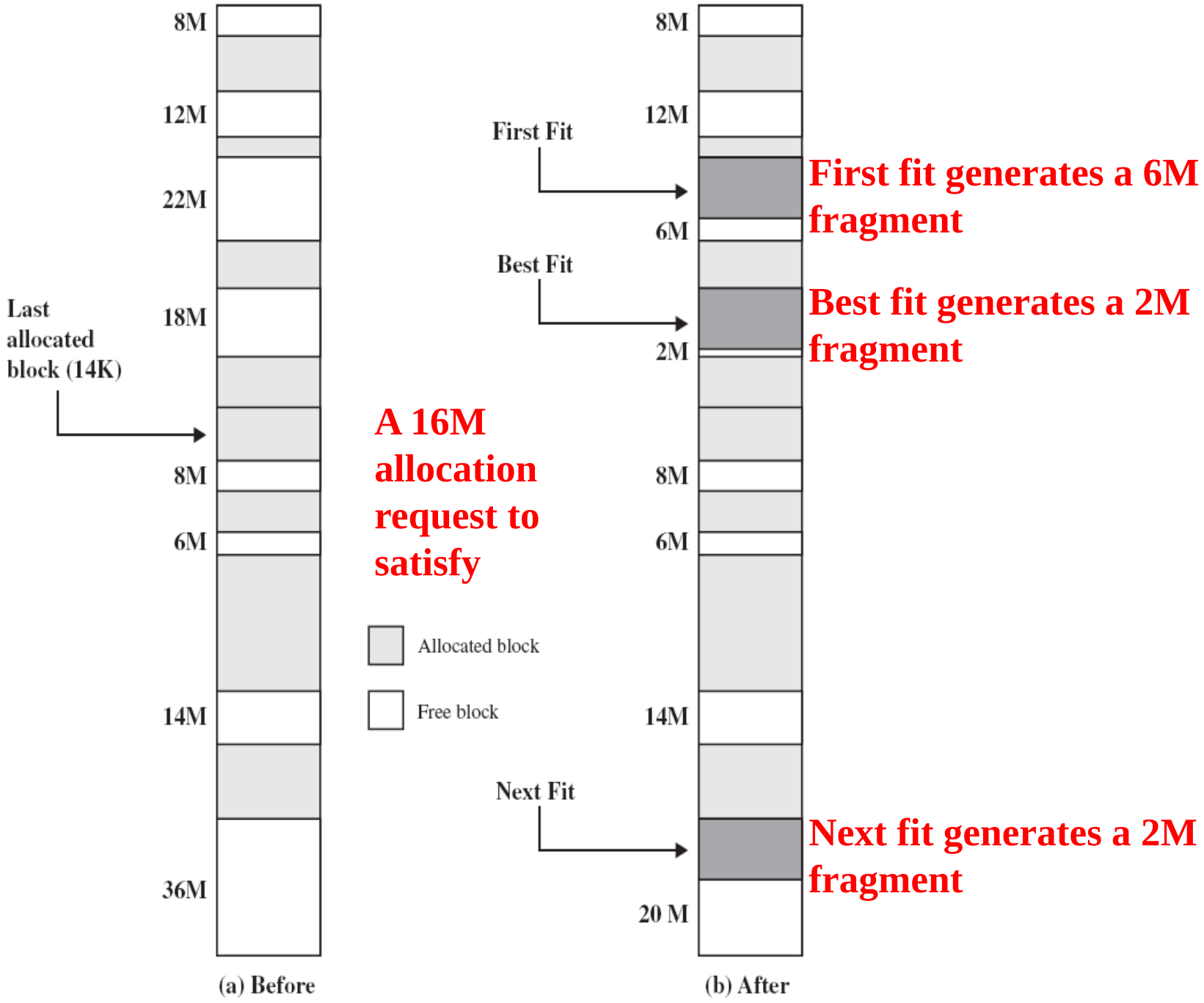
- **For compaction is time consuming, OS must be clever in deciding how to assign processes to memory**
  - **Best-fit (最佳适配):** *Choose the block that is closest in size to the request*
  - **First-fit (首次适配):** *Scan the memory from the beginning and choose the first available block that is large enough*
  - **Next-fit (邻近适配):** *Scan the memory from the location of the last placement and choose the next available block that is large enough*

Dynamic Partitioning (5)
Comments on Allocation Algorithms

- **The first-fit algorithm is not only the simplest but usually the best and fastest as well**
- **The Next-fit tends to be slightly worse than First-Fit**
  - **More frequent to allocate a block of memory at the end of memory where the largest block is usually found**
  - **The largest block of memory is quickly broken up into smaller blocks**
  - **Compaction is required more frequently to obtain a large block at the end of memory**

Dynamic Partitioning (6)
Comments on Allocation Algorithms

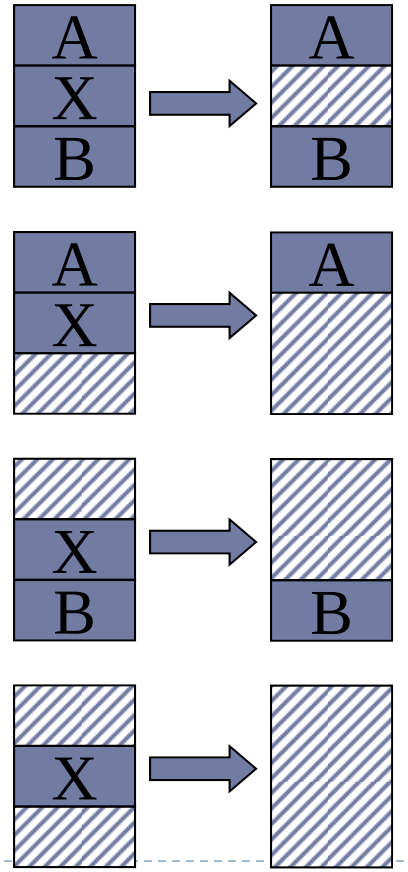
- **The best-fit algorithm is usually the worst performer**
  - **Since smallest block is found for process, the *smallest amount of fragmentation is left behind.***
  - **Memory compaction must be done more often**


8M
12M
22M
18M
Last allocated block (14K)
8M
6M
14M
36M
(a) Before
A 16M allocation request to satisfy
Allocated block
Free block
8M
12M
First Fit
6M
Best Fit
2M
8M
6M
14M
Next Fit
20 M
(b) After
First fit generates a 6M fragment
Best fit generates a 2M fragment
Next fit generates a 2M fragment

Dynamic Partitioning (7)
Drawbacks

- **Compared with fixed partitioning:**
  - A dynamic partitioning scheme is more complex to maintain
  - Overhead of compaction (内存紧缩的额外开销)

Dynamic Partitioning (8)
Memory Deallocation (内存回收)



A terminating process X normally has two neighbors (except when it is at the very top or bottom of memory)

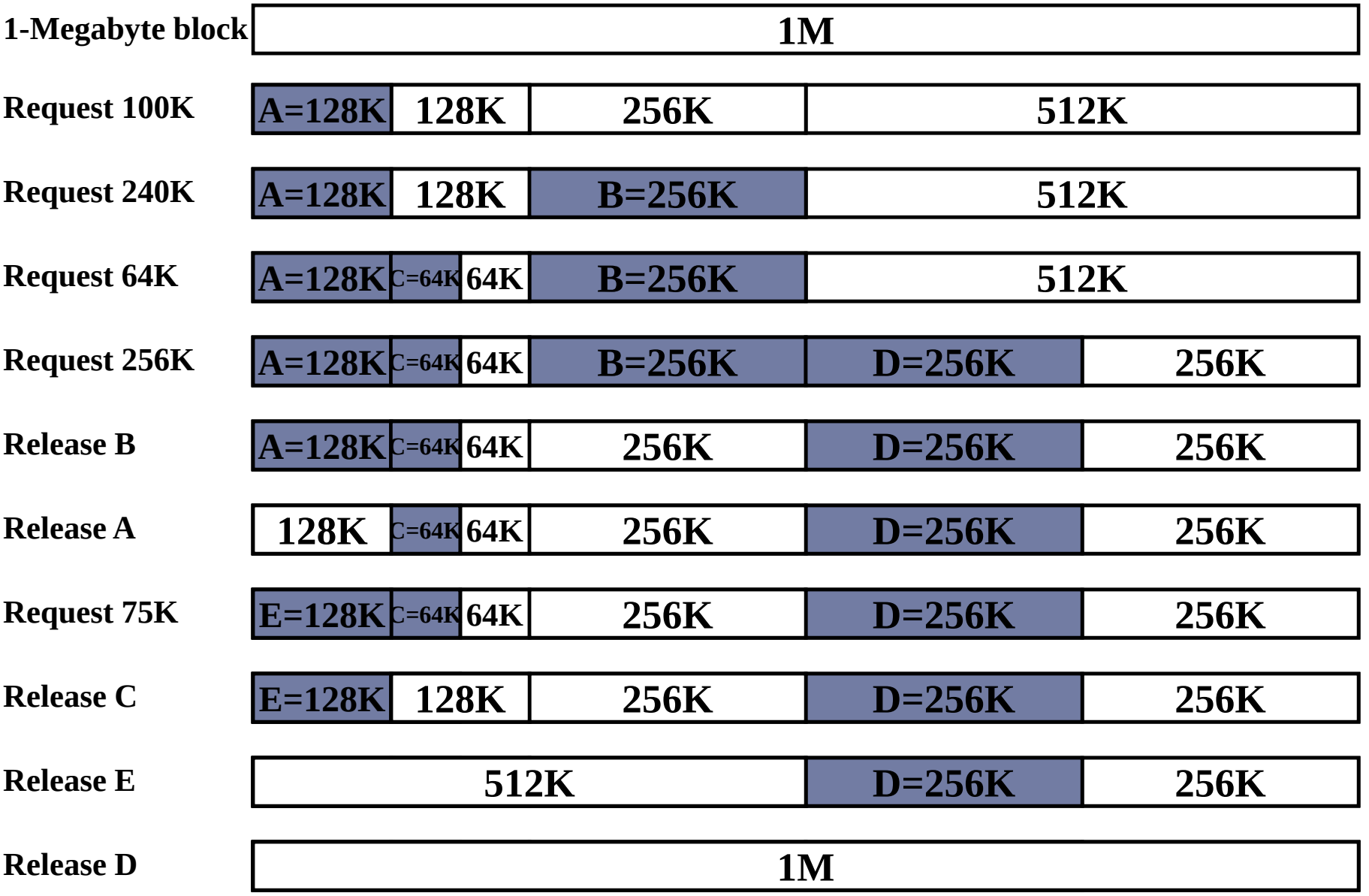
Each neighbor may be either a process or a hole, so there are four combinations

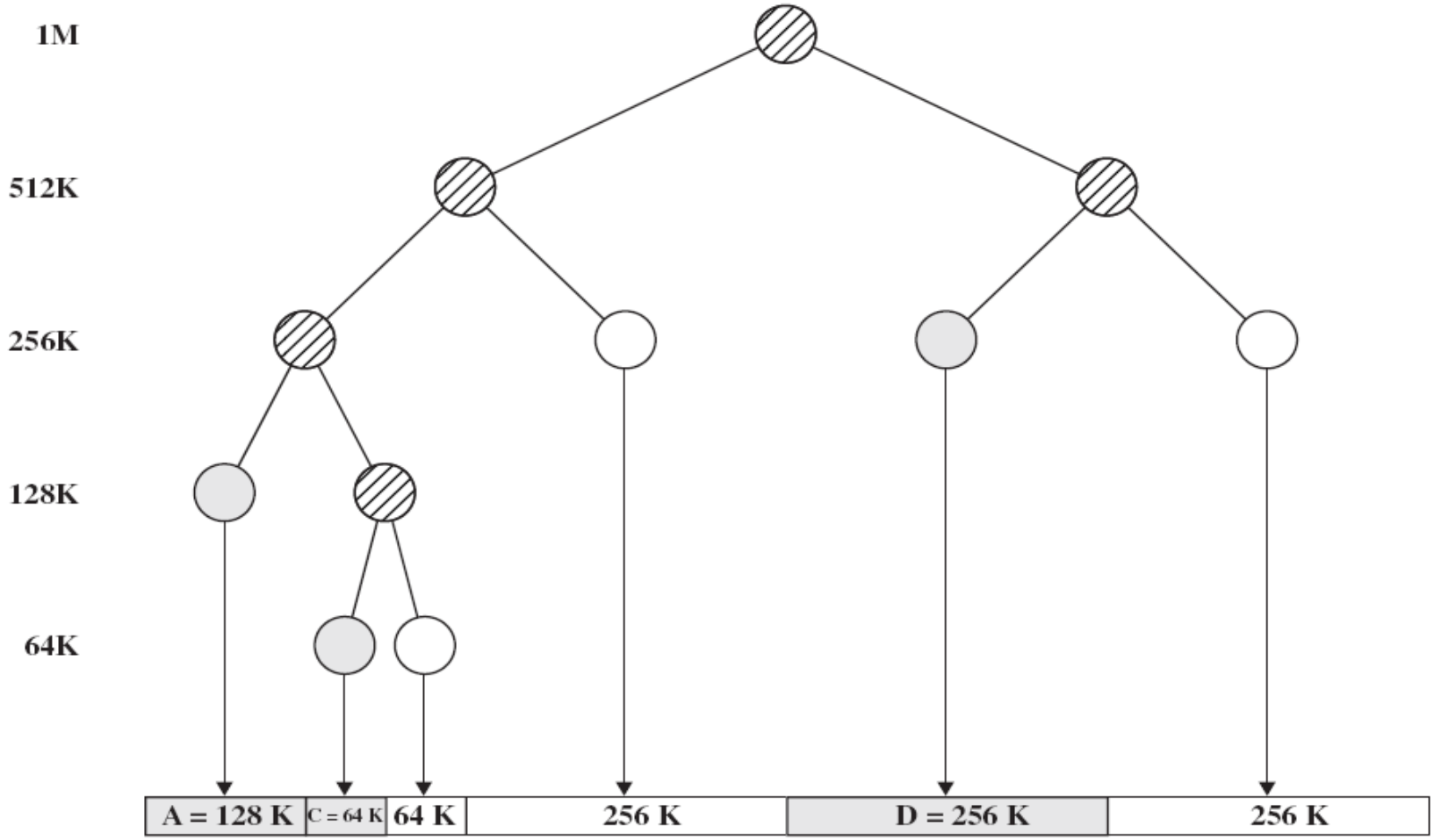
Buddy System (1)
伙伴系统

- The buddy system is a compromise between fixed and dynamic partitioning
  - Drawbacks of fixed and dynamic partitioning
- Entire space available is treated as a single block of 2^U
- If a request of size s such that $2^{U-1} < s <= 2^U$, entire block is allocated
  - Otherwise block is split into two equal buddies
  - Process continues until smallest block greater than or equal to s is generated

Buddy System (2)
An Example

- 1-Mbyte initial block
  - The first request, A, is for 100Kbytes
  - The next request, B, is for 240Kbytes
  - The third request, C, is for 64Kbytes
  - The fourth request, D, is for 256Kbytes
  - Release B
  - Release A
  - The fifth request, E, is for 75Kbytes
  - Release C
  - Release E
  - Release D


1-Megabyte block
1M
Request 100K
A=128K
128K
256K
512K
Request 240K
A=128K
128K
B=256K
512K
Request 64K
A=128K
C=64K
64K
B=256K
512K
Request 256K
A=128K
C=64K
64K
B=256K
D=256K
256K
Release B
A=128K
C=64K
64K
256K
D=256K
256K
Release A
128K
C=64K
64K
256K
D=256K
256K
Request 75K
E=128K
C=64K
64K
256K
D=256K
256K
Release C
E=128K
128K
256K
D=256K
256K
Release E
512K
D=256K
256K
Release D
1M



If two buddies are leaf nodes, then at least one must be allocated; otherwise they should be coalesced (合并) into a larger block

Relocation (1)
Two Situations

- When we use the fixed partitioning scheme with one process queue per partition, a process is always assigned to the same partition
  - A simple relocating loader can be used: it replaces all relative memory references in the code by absolute main memory addresses when the process is first loaded
- In the case of equal-size partitions, or a single process queue for unequal-size partitions, or dynamic partitioning, a process may occupy different partitions during the course of its life
  - How to handle? Use the dynamic run-time loader

Relocation (2)
Types of Addresses

- **Logical Address**
  - Reference to a memory location independent of the current assignment of data to memory
  - Translation (转换) must be made to the physical address
- **Relative Address**
  - *A particular example of logical address*
  - Address expressed as a location relative to some known point (usually the beginning of a program)
- **Physical Address or Absolute Address**
  - The absolute address or actual location in main memory

Registers Used during Execution

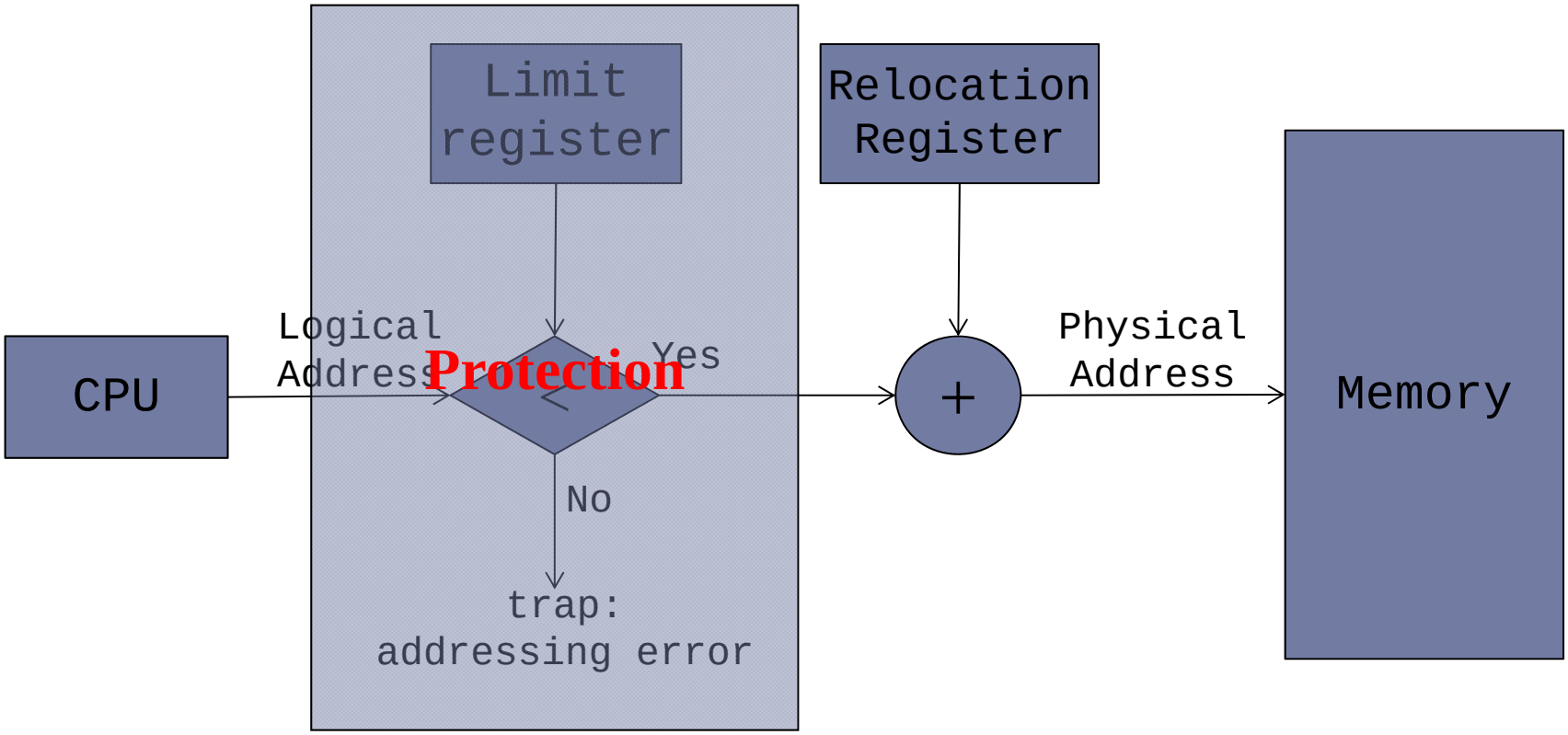
- **Relocation Register (or Base Register)**
  - Starting (physical) address for the process
- **Limit Register (or Bounds Register)**
  - The Range of logical addresses of the process (Ending location of the process)

These values are set when the process is loaded and when the process is swapped in

Registers Used during Execution

- The value of the base register is added to a relative address to produce an absolute address
- The resulting address is compared with the value in the bounds register
- If the address is not within bounds, an interrupt is generated to the operating system

Hardware Support for Relocation and Limit Registers



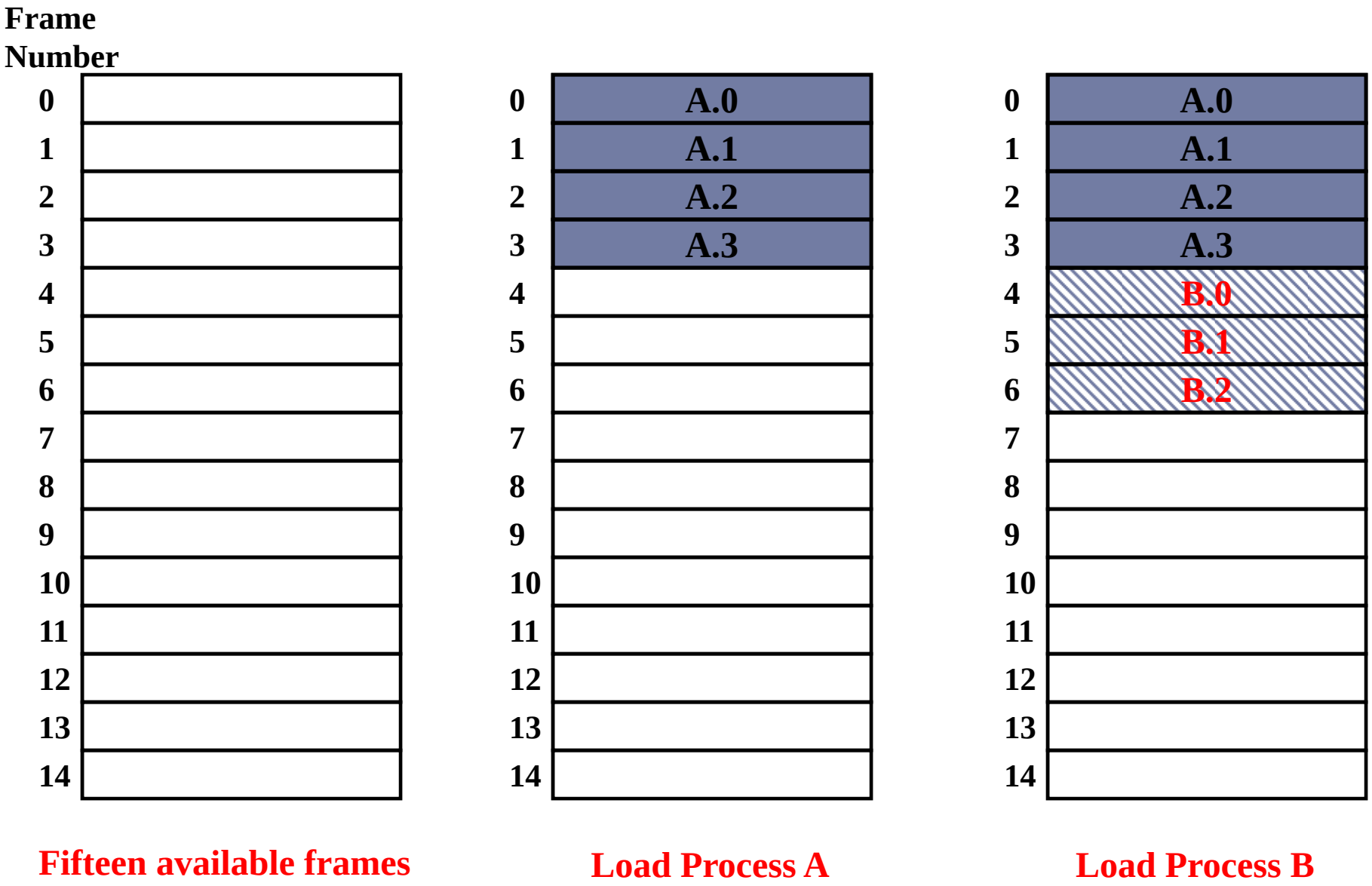
Paging
分页

Paging

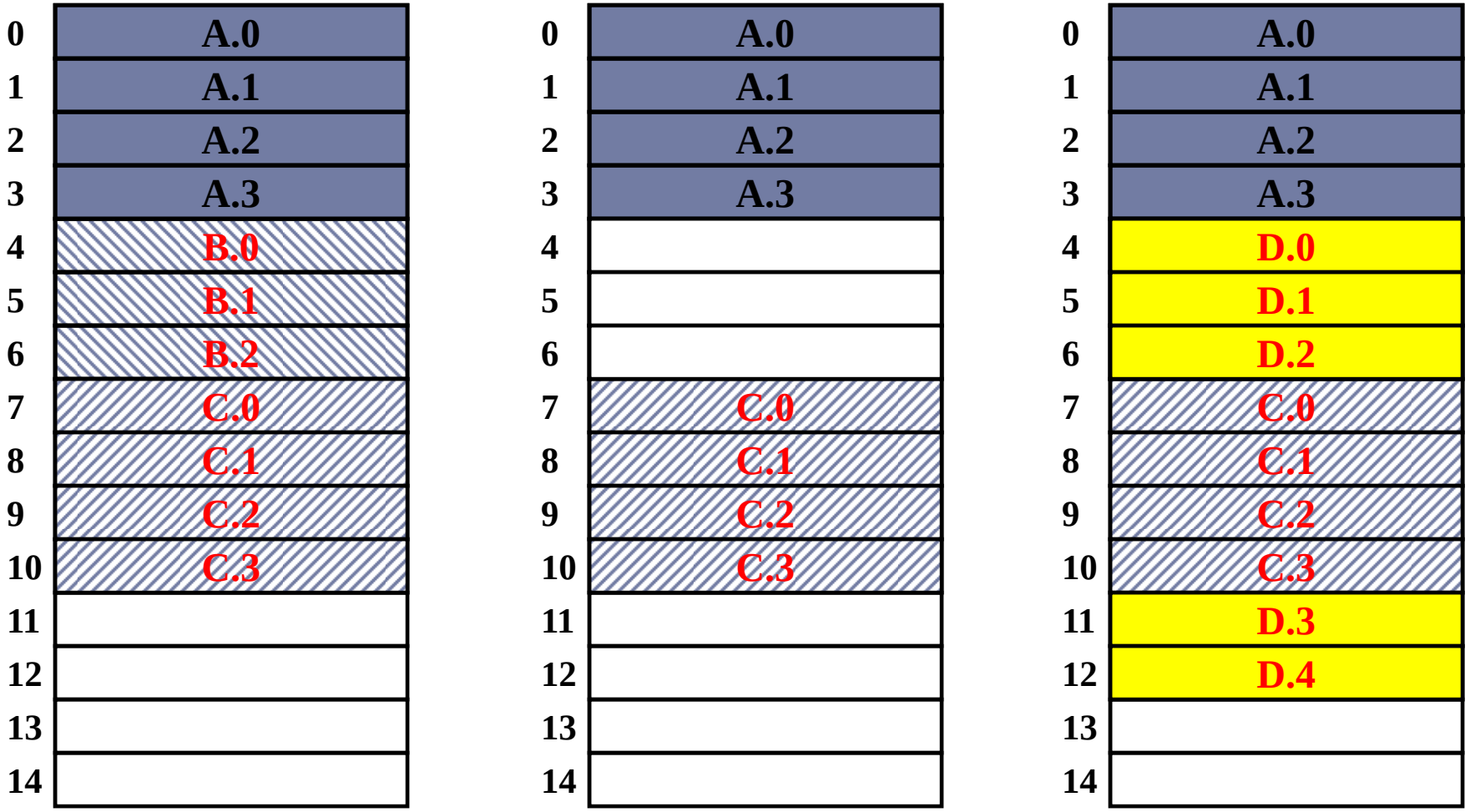
- **Paging permits the physical-address space of a process to be noncontiguous.**
  - Physical memory is broken into (*relatively*) small fixed-sized *small*) chunks called **frames** (帧或页框)
  - Logical memory is also broken into chunks of same size called **pages** (页或页面)
- **Wasted space for each process is due to internal fragmentation consisting of only a fraction of the last page of a process.**
  - There is *NO external fragmentation*

An Example

- Four processes
  - A: 4 pages
  - B: 3 pages
  - C: 4 pages
  - D: 5 pages
- 15 available frames in total


Frame Number
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
A.0
A.1
A.2
A.3
B.0
B.1
B.2
Fifteen available frames
Load Process A
Load Process B

Processes A, B, and C are all blocked, the OS has to swap one of them out and bring in another new process

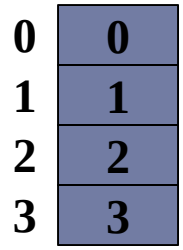


Load Process C **Swap Out B** **Load Process D**

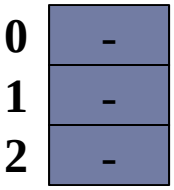
Page Table and Logical Address

- **The frames holding a process need NOT be contiguous.**
  - **A simple base address register is not enough**
- **The OS maintains a page table for each process**
  - **The page table shows the frame location for each page of the process**
- **Each logical address consists of a page number and an offset within the page.**

Page tables and free frame list

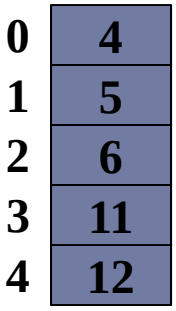


Process A
page table

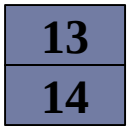


Process B
page table

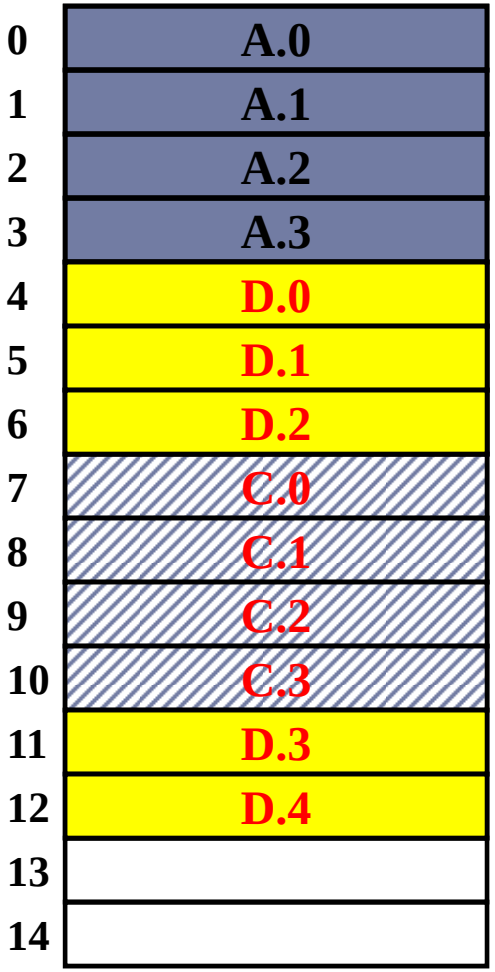
Process C
page table



Process D
page table



Free frame
list

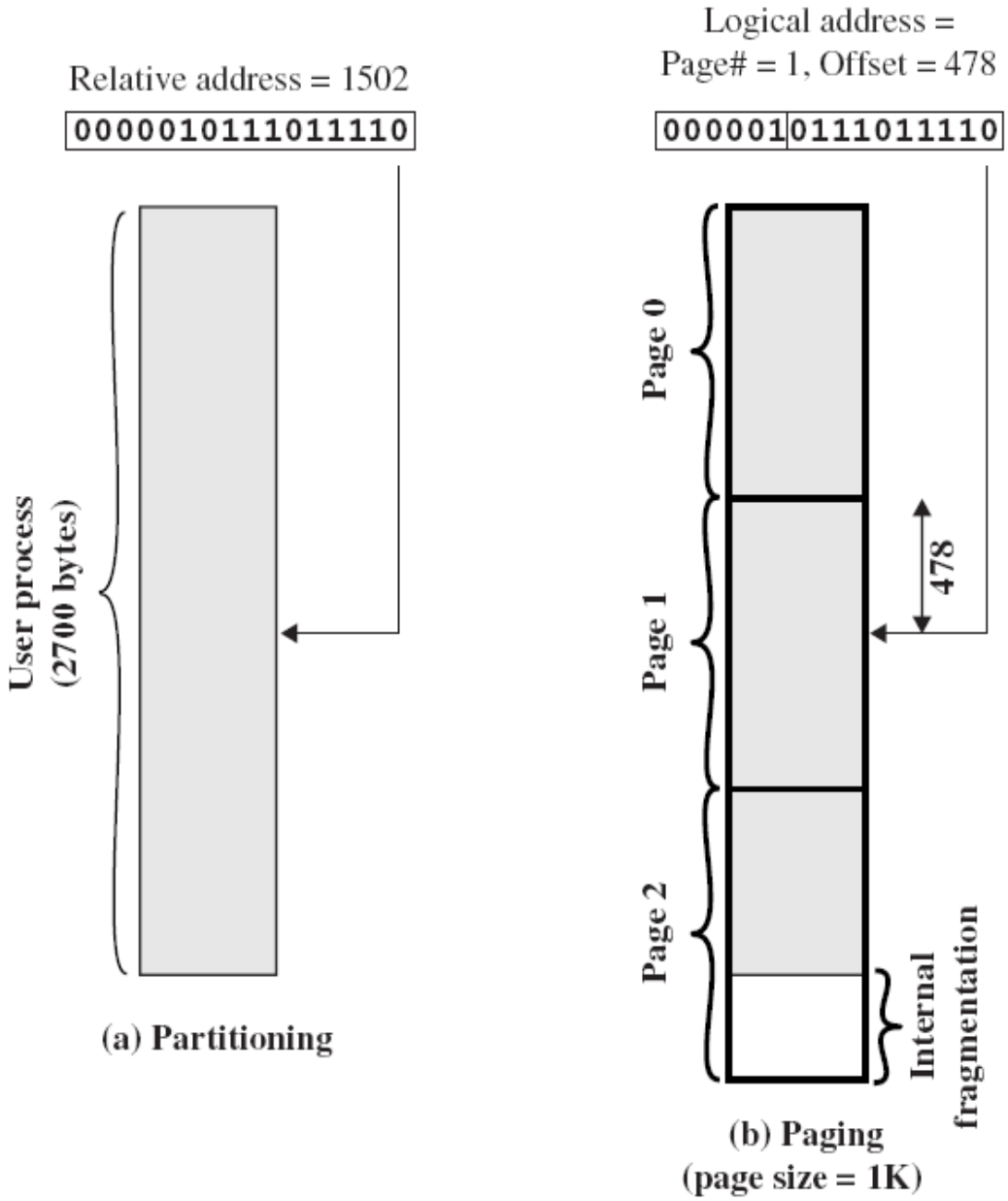


Paging v.s Partitioning

- **Simple paging is similar to fixed partitioning, but it is different in that:**
  - **With paging, the partitions are rather small**
  - **A program may occupy more than one partition**
  - **The partitions occupied by a single program need not be contiguous**
  - **The internal fragmentation is less**

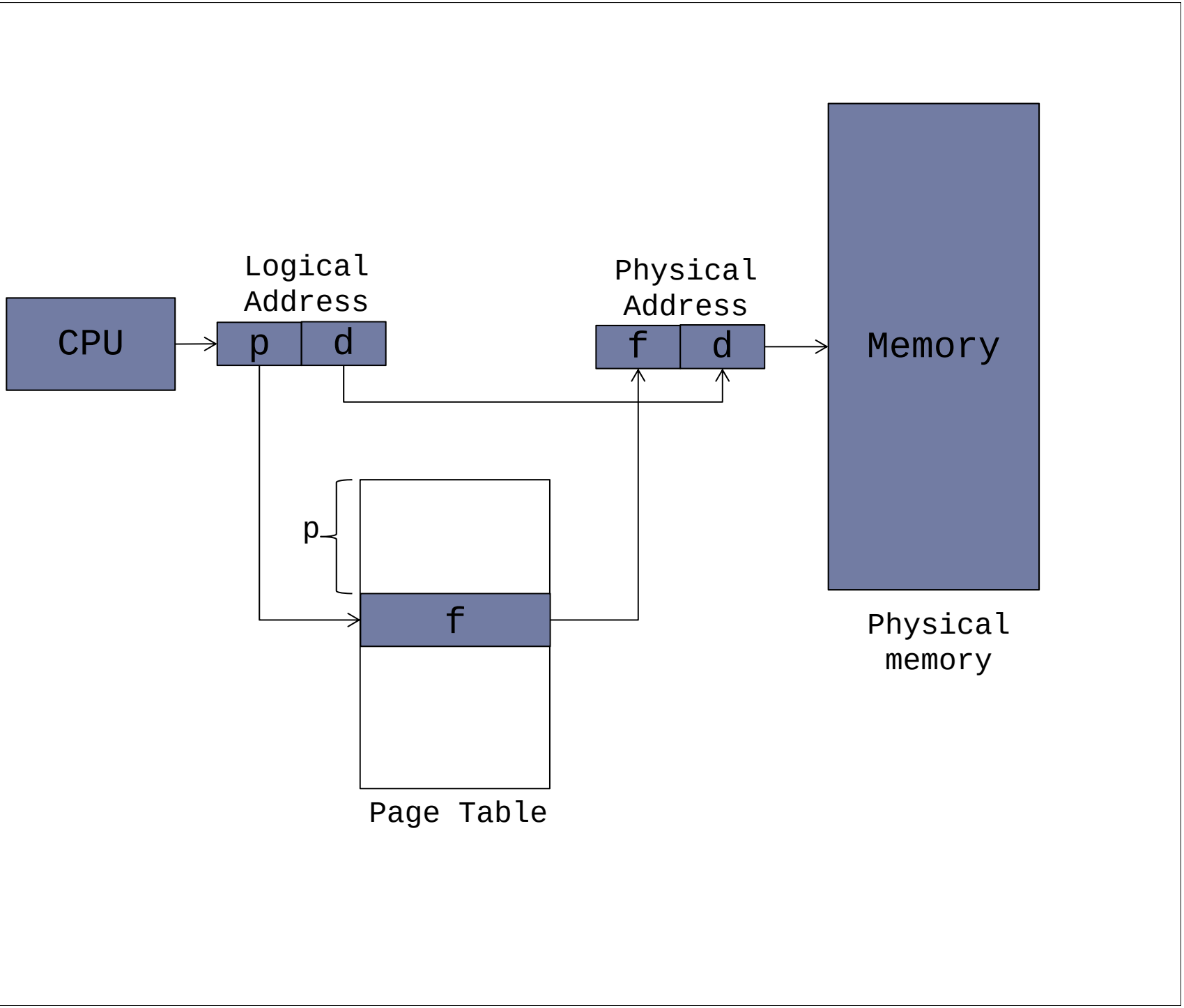
Page Size

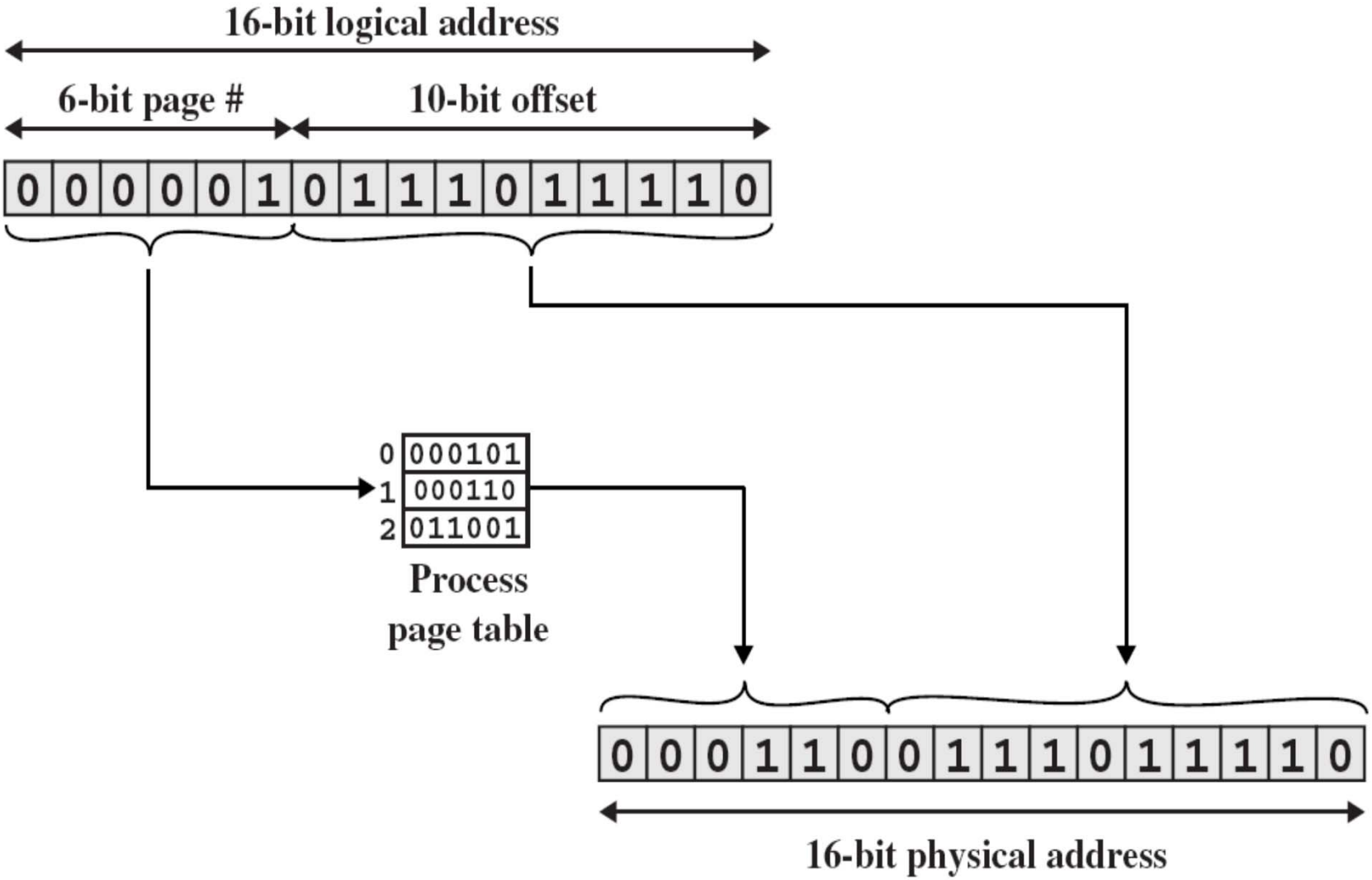
- *The page size (or the frame size) is typically a power of 2*
  - *It is defined by the hardware.*
- *Two advantages of using a power of 2 as page size:*
  - *The logical addressing scheme is transparent to the programmer, the assembler, and the linker. Each logical address is identical to its relative address*
  - *It is relatively easy to implement a function in hardware to perform dynamic address translation at run time.*


Relative address = 1502
0000010111011110
User process
(2700 bytes)
(a) Partitioning
Logical address =
Page# = 1, Offset = 478
000001 0111011110
Page 0
Page 1
Page 2
478
Internal
fragmentation
(b) Paging
(page size = 1K)

Address Translation

- **Consider an address of $n+m$ bits, where the leftmost n bits are the page number and the rightmost m bits are the offset**
- **Address Translation:**
  - **Extract the page number as the leftmost n bits of the logical address**
  - **Use the page number as an index into the process page table to find the frame number, k.**
  - **The starting address of the frame is $k \times 2^m$, and the physical address of the reference byte is that number plus the offset.**


CPU
Logical
Address
p
d
Physical
Address
f
d
Memory
p
f
Page Table
Physical
memory

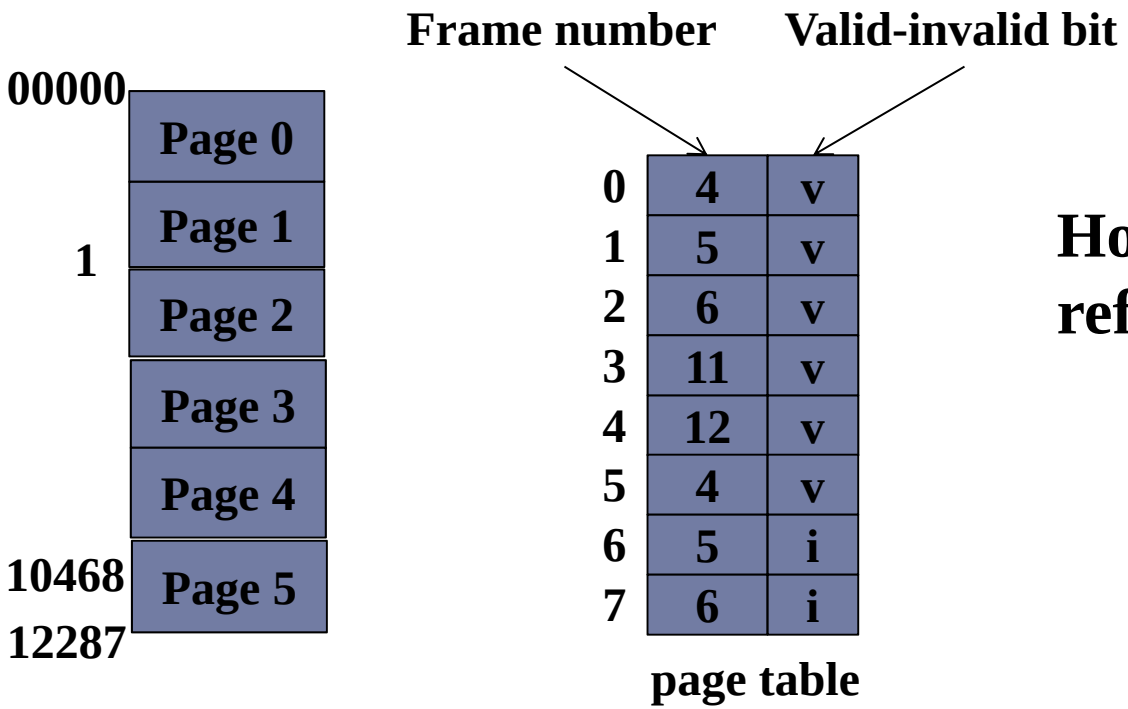

16-bit logical address
6-bit page #
10-bit offset
0 0 0 0 0 1 0 1 1 1 0 1 1 1 1 0
0 000101
1 000110
2 011001
Process
page table
0 0 0 1 1 0 0 1 1 1 0 1 1 1 1 0
16-bit physical address

(a) Paging

Protection
保护

- **Protection bits** associated with each frame
  - One bit can define a page to be **read-write or read-only**.
    - Can be extended to a finer level of protection: read-only, read-write, or execute-only
  - One additional bit is generally attached to each entry in the page table: a **valid-invalid bit**.
    - An "invalid" page indicates that the page is not in the process's logical address space
- **Violation of the memory protection will cause a hardware trap to the OS**

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

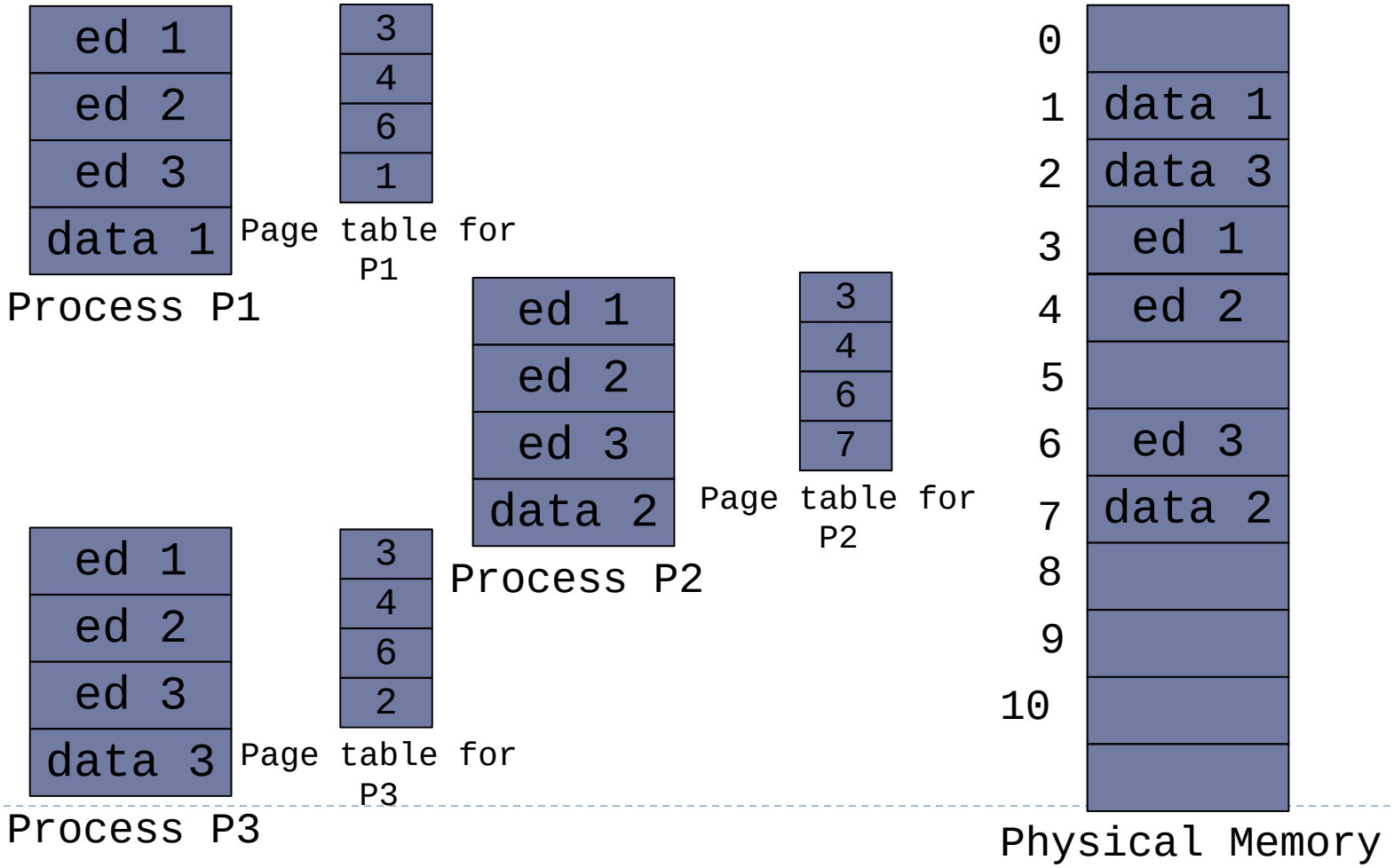


How about address reference at 12000?

A system with a 14-bit address space (0 to 16383), and page size of 2KB. We have a program that should use only addresses 0 to 10468.

Any attempt to generate an address in pages 6 or 7 will find that the valid-invalid bit is set to invalid, and the computer will trap to the OS (invalid page reference)

Shared Pages
共享页面

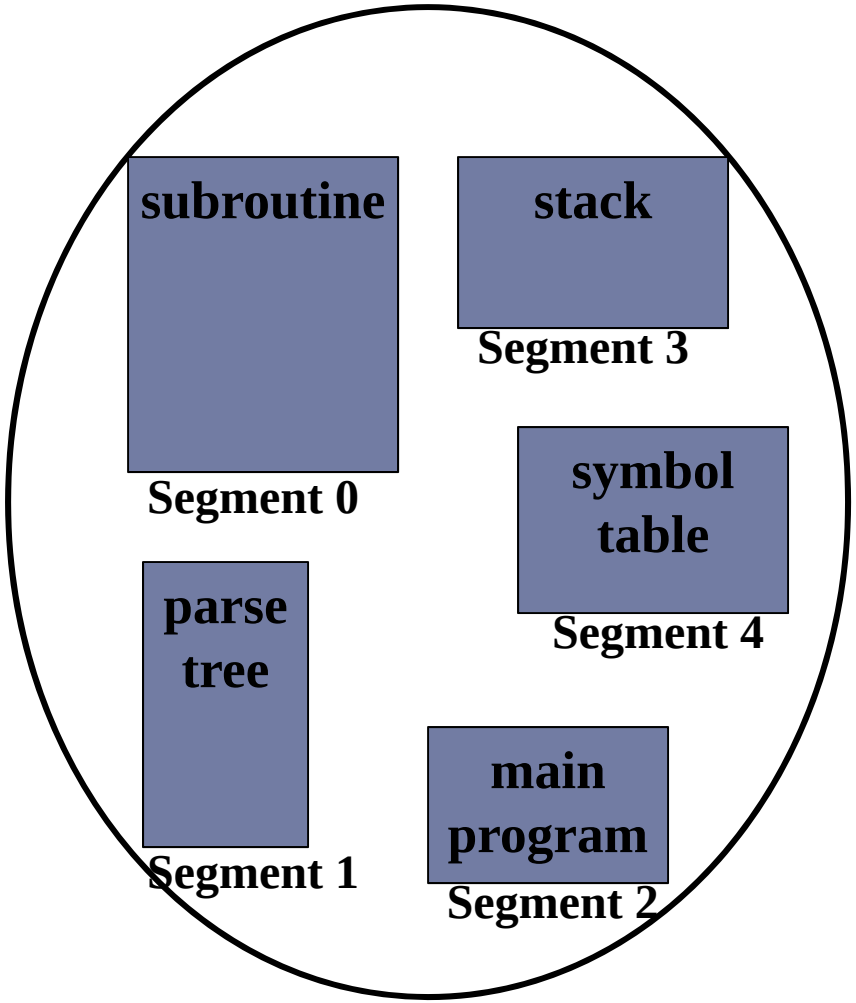


Segmentation
分段

User's View of a Program

- **Users prefer to view memory as a collection of variable-sized segments, with no necessary ordering among segments.**
  - **User normally thinks of a program as a main program with a set of subroutines, procedures, or functions.**
- **Each of these modules or data elements is referred to by name**

User's View of a Program



logical address space

Segmentation

- ***Segmentation is a memory management scheme that supports the user view of memory***
  - *A logical address space is a collection of variable-sized segments. (Each segment has a name and a length)*
  - *For simplicity of implementation, segments are numbered and referred to by a segment number*
- ***Logical address consists of: a segment number and an offset into the segment***

Segmentation v.s Dynamic Partitioning

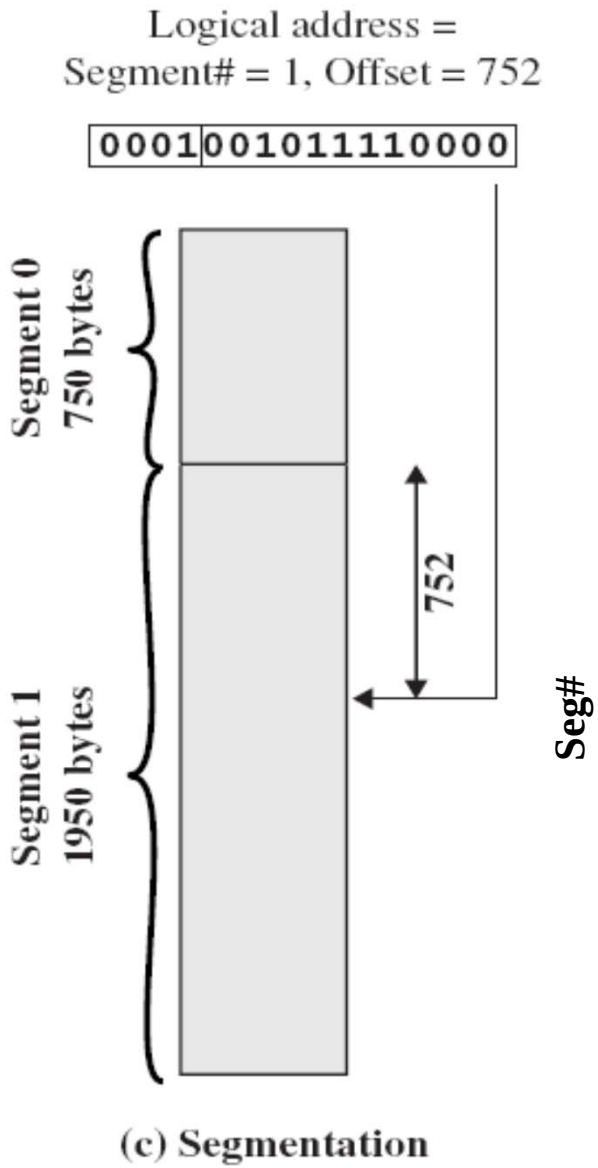
- **Similarities:**
  - Unequal-size segments
  - Segmentation eliminates internal fragmentation, but suffers from external fragmentation,
- **However, with segmentation:**
  - A program can occupy more than one partition, and these partitions need not be contiguous
  - The external fragmentation is less than dynamic partitioning because a process is broken up into a number of smaller pieces

Segmentation v.s Paging

- **Paging is invisible to the programmer, while segmentation is usually visible**
  - **Segmentation is provided as a convenience for organizing programs and data**
- **With segmentation, there is no simple relationship between logical addresses and physical addresses**

Segment Table Entry

- **How to map two-dimensional user-defined addresses into one-dimensional physical address?**
  - The mapping is done with the help of a segment table.
  - Each segment table entry should give the **starting address** in main memory of the corresponding segment.
  - The entry should also provide **the length of the segment**, to assure that invalid (无效的) addresses are not used

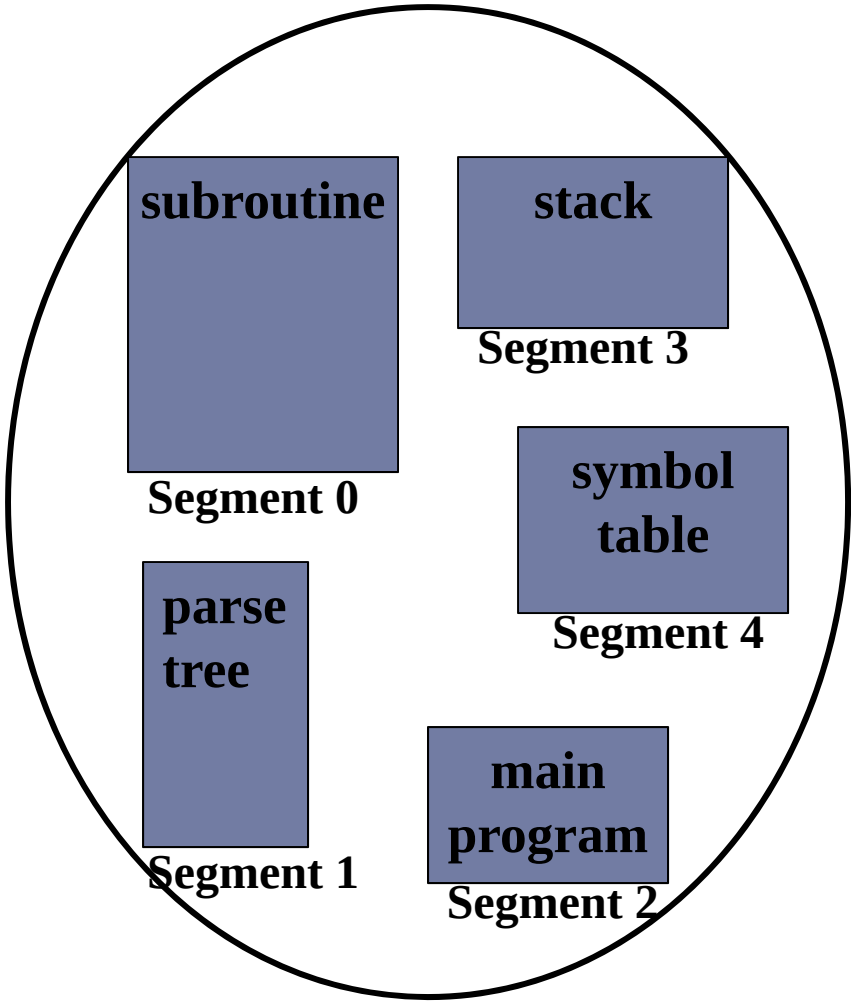


(c) Segmentation

Leftmost 4 bits: segment number

Rightmost 12 bits: offset

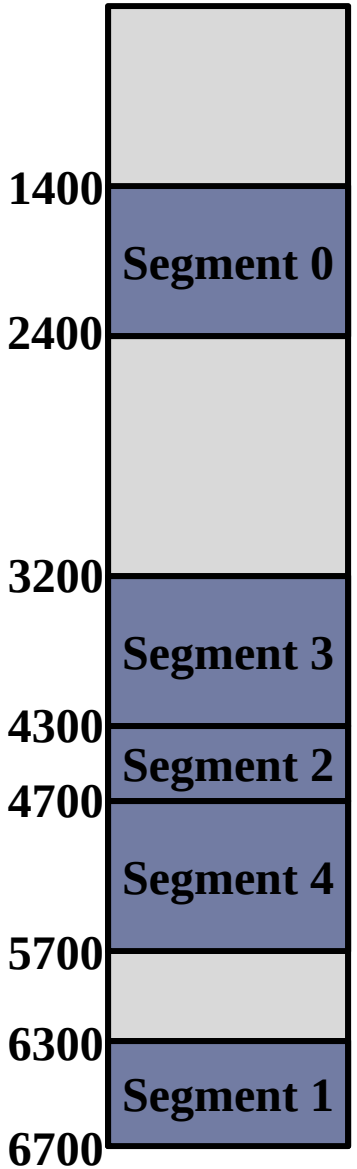
Maximum segment size: 2^{12}=4096



logical address space

	length	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

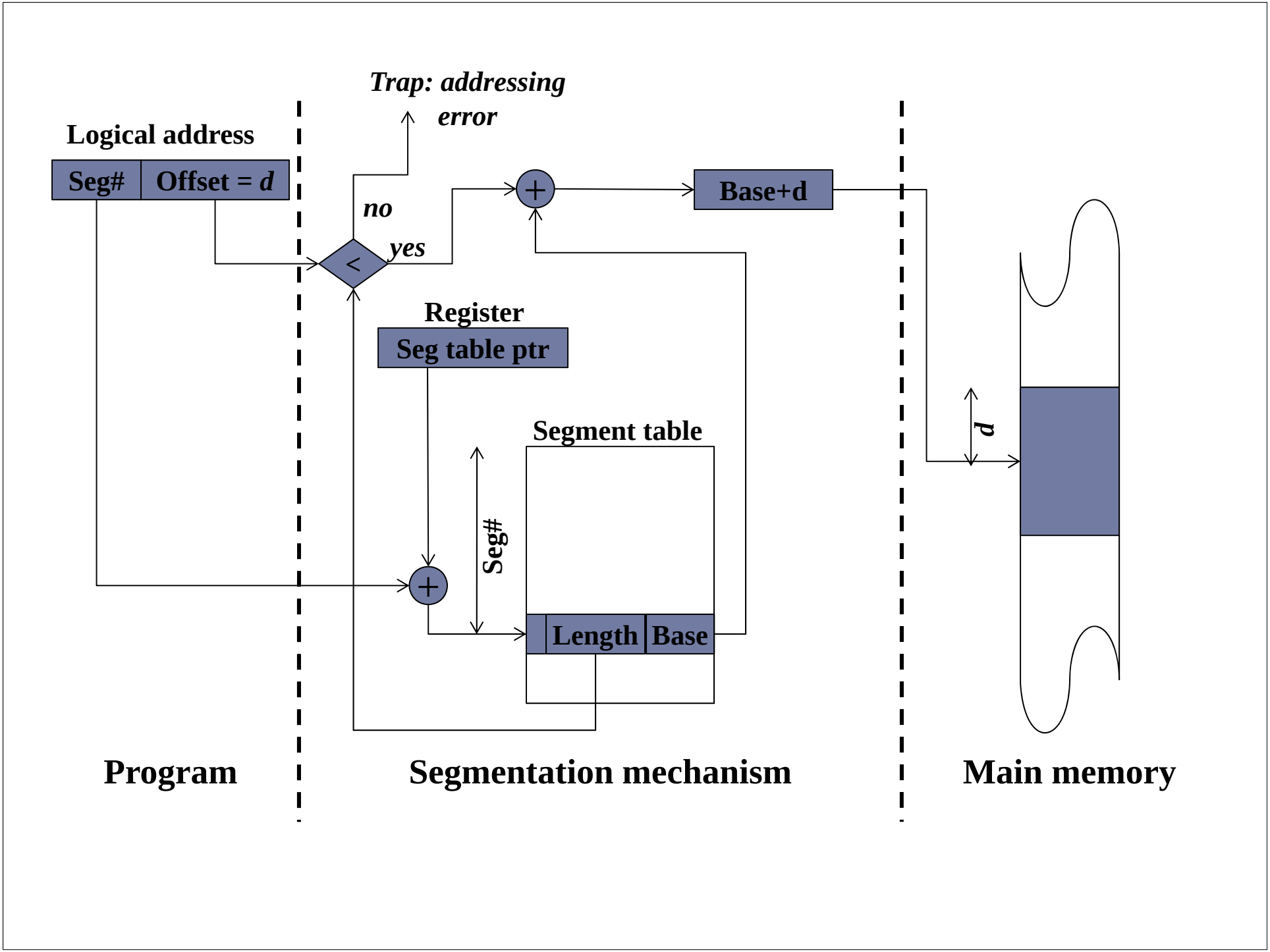
segment table

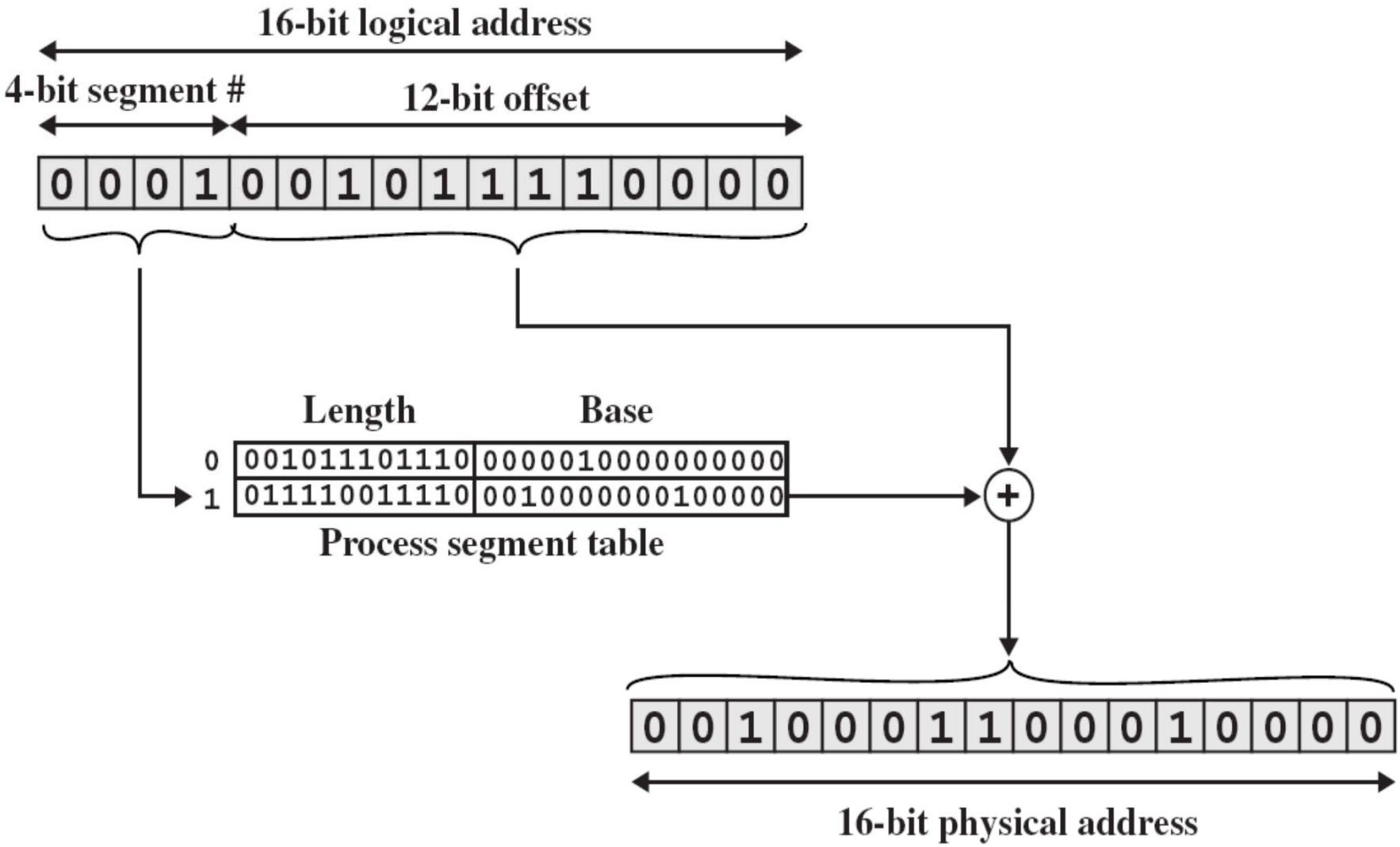


physical memory

Address Translation

- **Consider an address of *n*+*m* bits, where the leftmost *n* bits are the segment number and the rightmost *m* bits are the offset**
  - **Maximum segment size is 2^m**
- **Address Translation:**
  - **Extract the segment number as the leftmost *n* bits of the logical address**
  - **Use the segment number as an index into the process segment table to find the starting physical address of the segment.**
  - **Compare the offset (the rightmost *m* bits) to the length of the segment. If the offset is greater, the address is invalid.**
  - **The desired physical address is the sum of the starting physical address of the segment plus the offset.**


Trap: addressing error
Logical address
Seg#
Offset = d
no
yes
<
Base+d
Register
Seg table ptr
Segment table
Seg#
Length
Base
d
Program
Segmentation mechanism
Main memory



(b) Segmentation

Dynamic Growth of Segments

- **Each segment consists of a linear sequence of addresses, from 0 to some maximum**
  - **Different segments usually have different lengths.**
- **Segment lengths may change during execution**
  - **Because each segment constitutes a separate address space, different segments can grow or shrink independently without affecting each other**

Dynamic Linking

- *If each procedure occupies a separate segment, with address 0 as its starting address*
  - *A procedure call to the procedure in segment n will use the two-part address (n, 0) to address the entry point*
- *If the procedure in segment n is modified later, no other procedures need to be changed*
  - *Because no starting addresses have been modified.*

Protection

- ***Each segment forms a logical entity of which the programmer is aware.***
  - *Such as: a procedure, or an array, or an stack*
  - *Different segments can have different kinds of protection.*
- ***All the information in a segment will normally be used in the same way***
  - *A floating-point array can be specified as read/write but not execute*
  - *A procedure segment can be specified as execute-only, prohibiting attempts to read from or store into it*

Sharing

- ***Segmentation also facilitates sharing procedures or data between several processes***
  - ***Segments are the logical units of information, while pages are only the physical units of storing information***
  - ***Clearly, sharing should be based on the logical units of information***

Paging VS Segmentation

	Paging	Segmentation
Need the programmer be aware that this technique is being used?	NO	YES
How many linear address spaces are there?	1	Many
Can procedures and data be distinguished and separately protected?	NO	YES
Can tables whose size fluctuates be accommodated easily?	NO	YES
Is sharing of procedures between users facilitated?	NO	YES
Why was this technique invented?	To get a large linear address space without buying more physical memory	To break up programs and data into logically independent address spaces and to aid sharing and protection

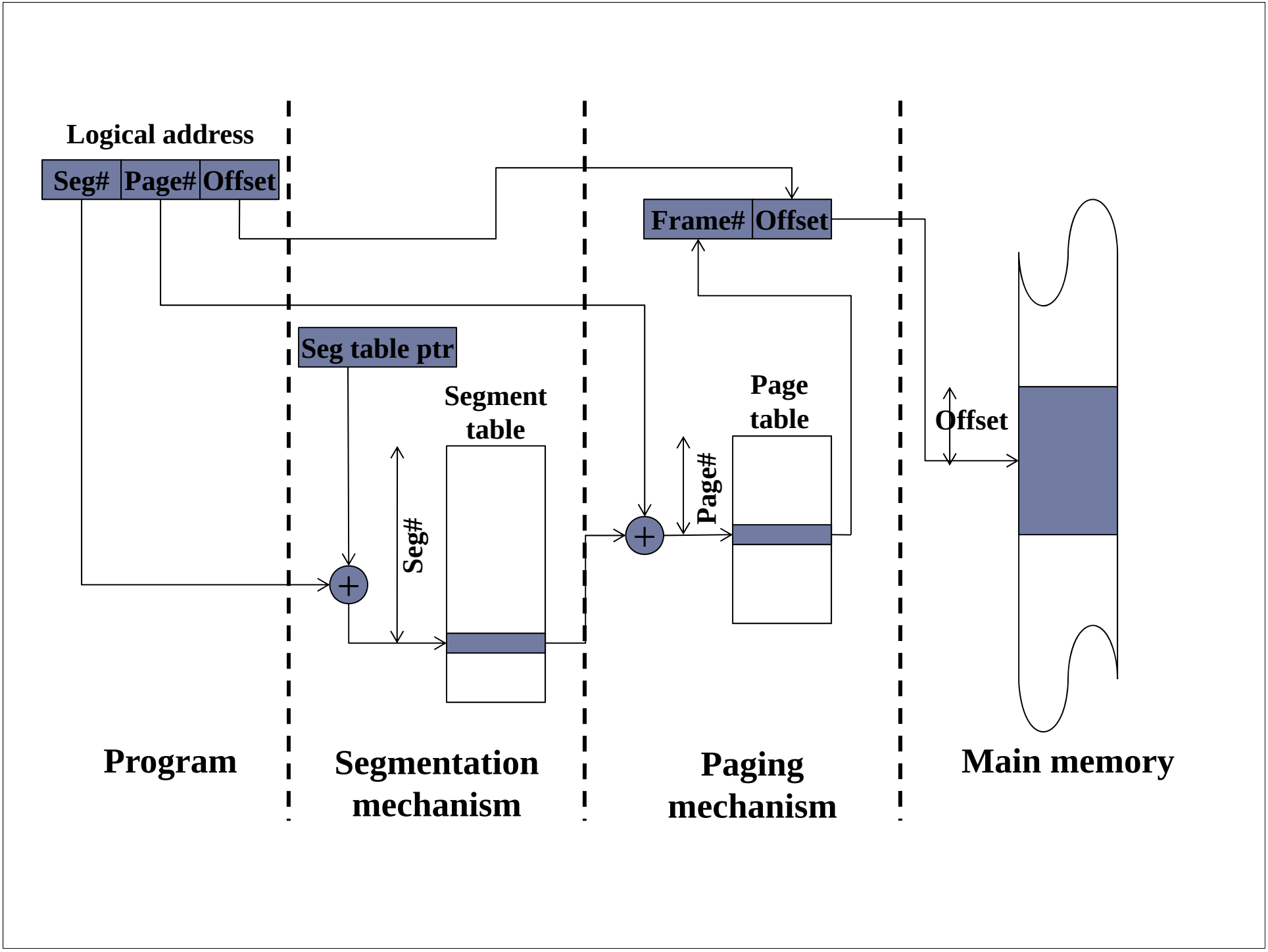
Segmentation with Paging
段页式内存管理

Basic Idea

- *A user's address space is broken up into a number of segments, and each segment is, in turn, broken up into a number of fixed-size pages*
- *Logical address consists of three parts:*
  - *The segment number*
  - *The page number in the segment*
  - *The offset within the page*

Segment Table and Page Tables

- ***Each process is associated with a segment table and a number of page tables, one per process segment.***
  - *When a process is running, a register holds the starting address of the segment table for that process*
  - *The segment table entry contains the length of the segment, and a base field which refers to a page table*
  - *Each page number is mapped into a corresponding frame number.*


Logical address
Seg#
Page#
Offset
Seg table ptr
Segment table
Seg#
Page table
Page#
Frame#
Offset
Offset
Program
Segmentation mechanism
Paging mechanism
Main memory