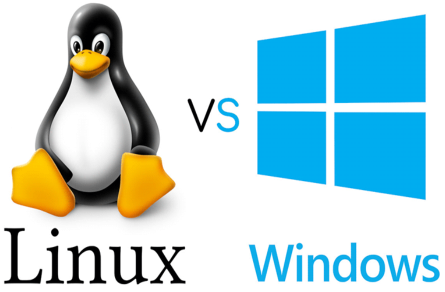
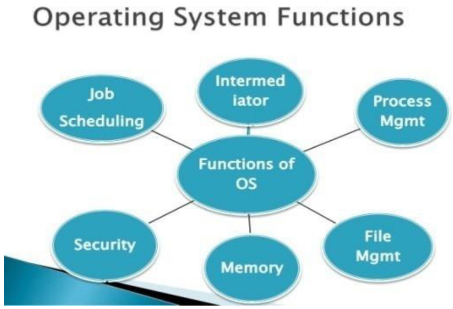
CIT210 OP. SYS. MANAG.

CSUN

COLLEGE OF
ENGINEERING AND
COMPUTER SCIENCE

Week6Day1 : Memory Management Techniques, Fixed Partitioning, Dynamic Partitioning, Simple Paging, Simple Segmentation, Placement Algorithms, Virtual Memory




Operating System Functions

Job Scheduling

Intermed iator

Process Mgmt

Functions of OS

Security

Memory

File Mgmt

Objectives (Memory Management)

- Memory Partitioning
- Paging
- Structure of the Page Table
- Segmentation
- Virtual Memory Management

Memory Management Techniques

1. Fixed Partitioning
   - Divide memory into partitions at boot time, partition sizes may be equal or unequal but don't change
   - Simple but has internal fragmentation
2. Dynamic Partitioning
   - Create partitions as programs loaded
   - Avoids internal fragmentation, but must deal with external fragmentation
3. Simple Paging
   - Divide memory into equal-size pages, load program into available pages
   - No external fragmentation, small amount of internal fragmentation

Memory Management Techniques

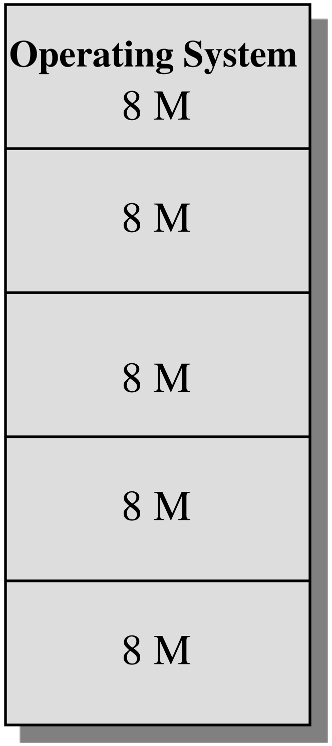
4. Simple Segmentation
   - Divide program into segments
   - No internal fragmentation, some external fragmentation
5. Virtual-Memory Paging
   - Paging, but not all pages need to be in memory at one time
   - Allows large virtual memory space
   - More multiprogramming, overhead
6. Virtual Memory Segmentation
   - Like simple segmentation, but not all segments need to be in memory at one time
   - Easy to share modules
   - More multiprogramming, overhead

Fixed Partitioning

- Any process whose size is less than or equal to the partition size can be loaded into an available partition
- If all partitions are full, the operating system can swap a process out of a partition

1. Fixed Partitioning

- Main memory divided into static partitions
- Simple to implement
- Inefficient use of memory
  - Small programs use entire partition
  - Maximum active processes fixed
  - Internal Fragmentation



Operating System
8 M

8 M

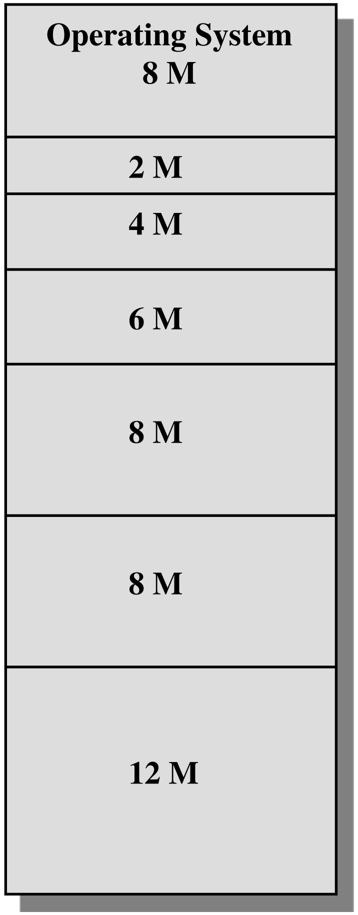
8 M

8 M

8 M

Fixed Partitioning

- Variable-sized partitions
  - assign smaller programs to smaller partitions
  - lessens the problem, but still a problem
- Placement
  - Which partition do we use?
    - Want to use smallest possible partition
    - What if there are no large jobs waiting?
  - Can have a queue for each partition size, or one queue for all partitions
- Used by IBM OS/MFT, obsolete
- Smaller partition by using overlays



Problems with Fixed Partitioning

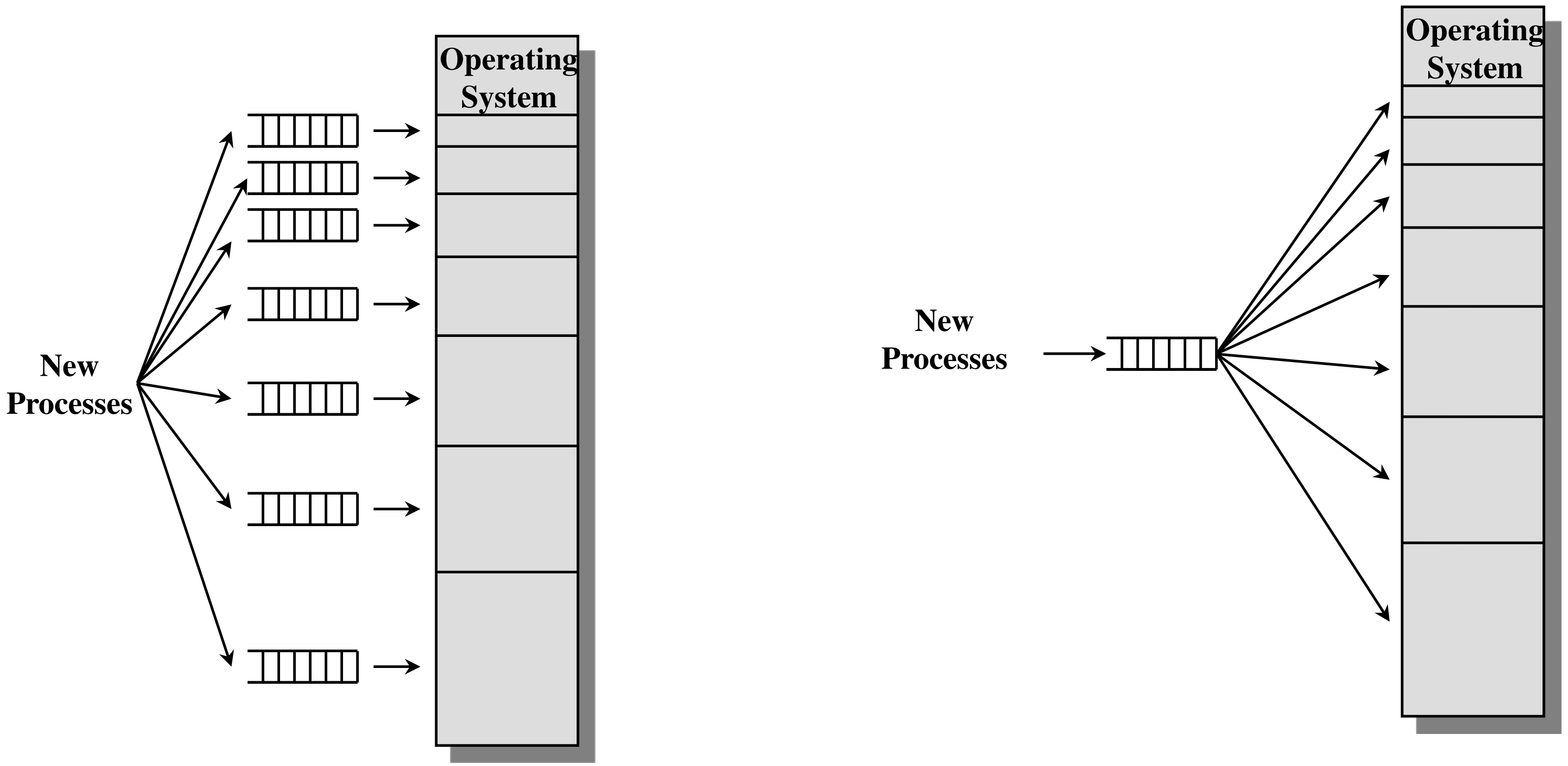
- A program may not fit in a partition. The programmer must design the program with overlays
- Main memory use is inefficient.

Placement Algorithm w/Partitions

- Equal-size partitions
  - because all partitions are of equal size, it does not matter which partition is used
  - Placement is trivial
- Unequal-size partitions
  - can assign each process to the smallest partition within which it will fit
  - queue for each partition
  - processes are assigned in such a way as to minimize wasted memory within a partition

Process Queues

- When its time to load a process into main memory the smallest available partition that will hold the process is selected

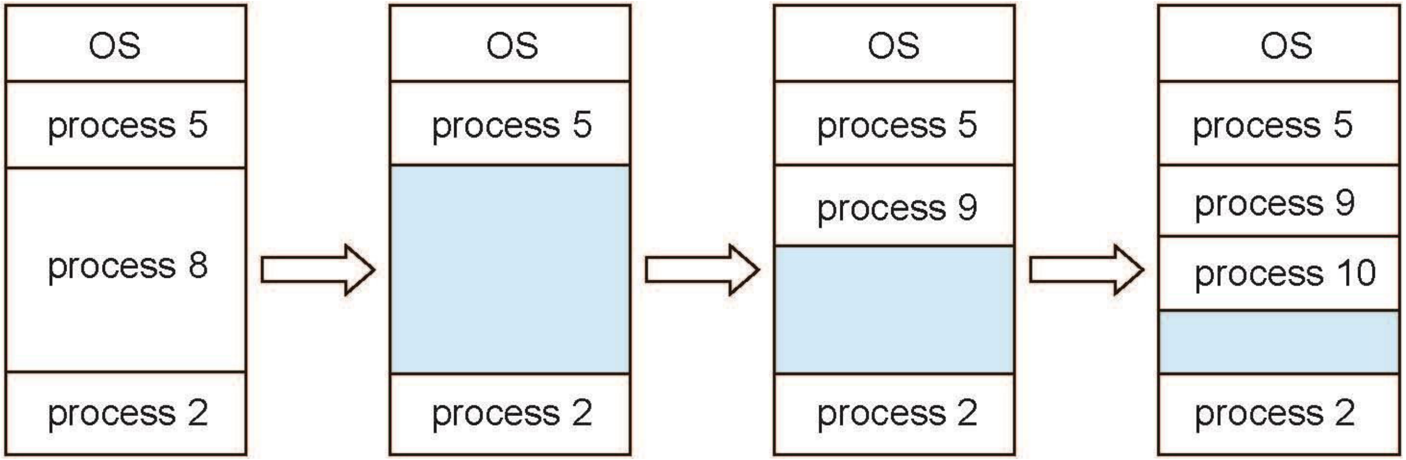


Dynamic Partitioning

- Partitions are of variable length and number
- Process is allocated exactly as much memory as required
- Eventually get holes in the memory.
  - external fragmentation
- Must use compaction to shift processes so they are contiguous and all free memory is in one block

Multiple-partition allocation

- **Multiple-partition allocation**
  - Degree of multiprogramming limited by number of partitions
  - Variable-partition sizes for efficiency (sized to a given process' needs)
  - Hole – block of available memory; holes of various size are scattered throughout memory
  - When a process arrives, it is allocated memory from a hole large enough to accommodate it
  - Process exiting frees its partition, adjacent free partitions combined
  - Operating system maintains information about:
    a) allocated partitions   b) free partitions (hole)



Allocation Strategies

- First Fit
- Best Fit
- Next Fit
- Worst Fit

First-fit algorithm

- Scans memory form the beginning and chooses the first available block that is large enough
- Fastest
- May have many process loaded in the front end of memory that must be searched over when trying to find a free block

Best-fit algorithm

- Search through all the spots, allocate the spot in memory that most closely matches requirements.
- Chooses the block that is closest in size to the request
- Since smallest block is found for process, the smallest amount of fragmentation is left
- Lot of little holes …
- Memory compaction must be done more often

Next-fit

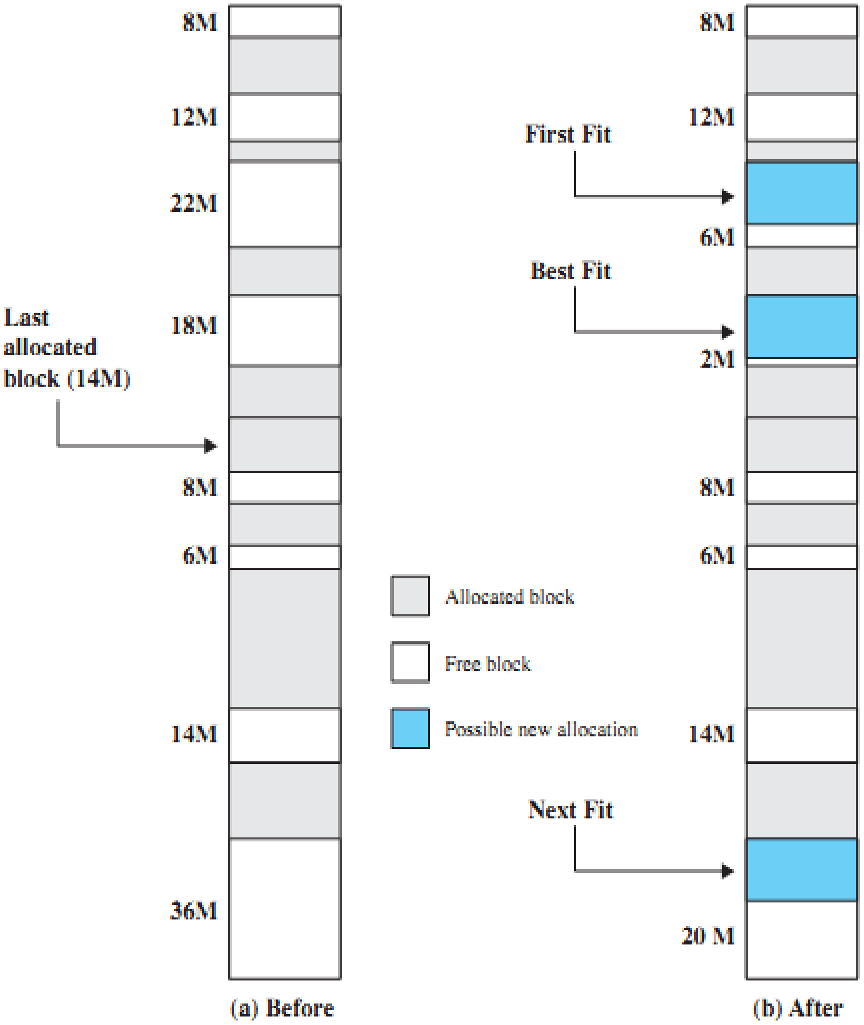
- Scans memory from the location of the last placement
- More often allocate a block of memory at the end of memory where the largest block is found
- The largest block of memory is broken up into smaller blocks
- Compaction is required to obtain a large block at the end of memory

Worst - Fit

- Allocate the *largest* hole; must also search entire list.
- Produces the largest leftover hole.

Placement Algorithms

- Used to decide which free block to allocate to a process of 16MB.
- Goal: reduce usage of compaction procedure (its time consuming).
- Example algorithms:
  - First-fit
  - Next-fit
  - Best-fit
  - Worst-fit (to imagine)



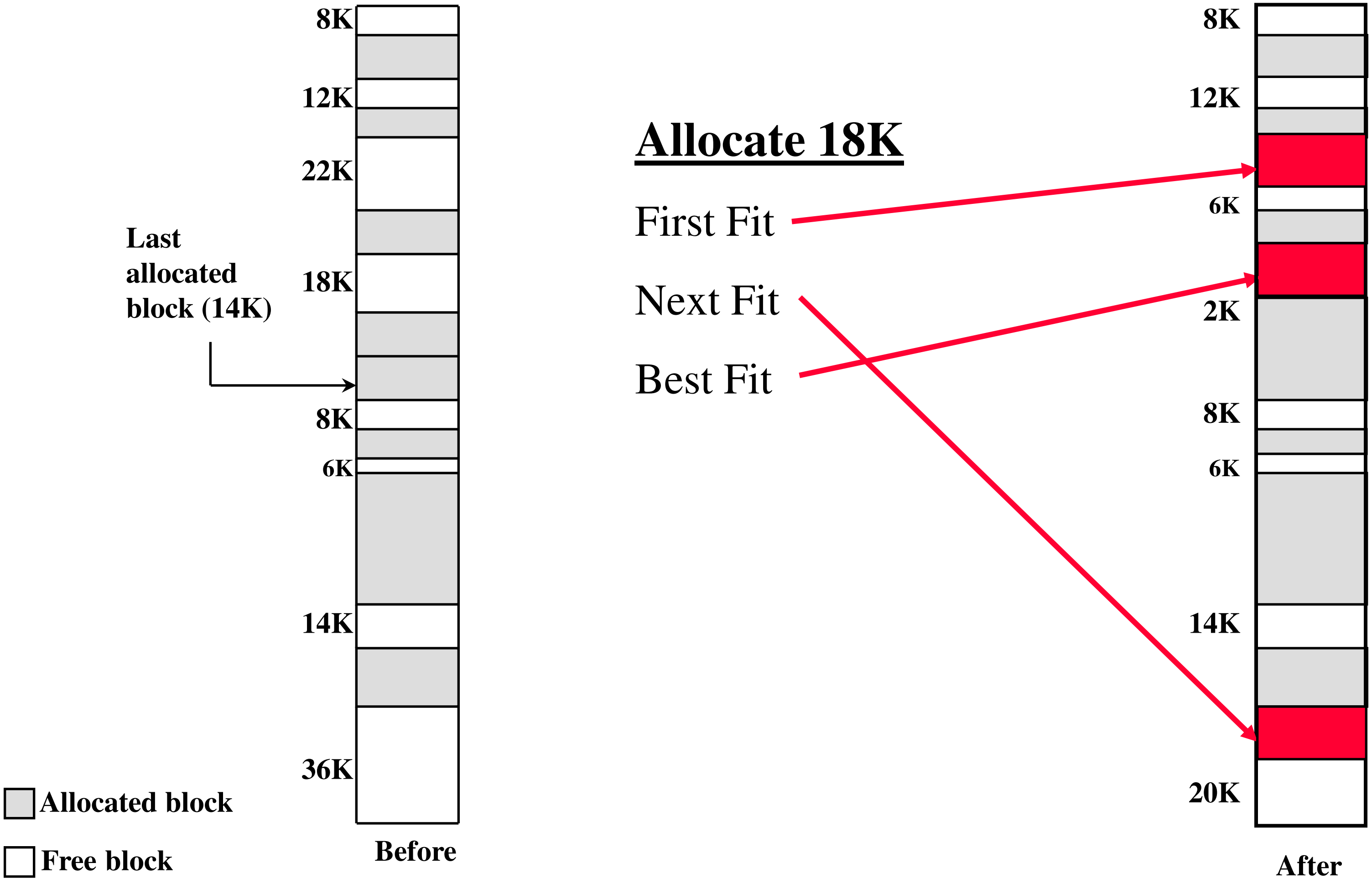
Comments on Placement Algorithms

- First-fit favors allocation near the beginning: tends to create less fragmentation then Next-fit.
- Next-fit often leads to allocation of the largest block at the end of memory.
- Best-fit searches for smallest block: the fragment left behind is small as possible –
  - main memory quickly forms holes too small to hold any process: compaction generally needs to be done more often.
- First/Next-fit and Best-fit better than Worst-fit (name is fitting) in terms of speed and storage utilization.

Which Allocation Strategy?

- The first-fit algorithm is not only the simplest but usually the best and the fastest as well.
  - May litter the front end with small free partitions that must be searched over on subsequent first-fit passes.
- The next-fit algorithm will more frequently lead to an allocation from a free block at the end of memory.
  - Results in fragmenting the largest block of free memory.
  - Compaction may be required more frequently.
- Best-fit is usually the worst performer.
  - Guarantees the fragment left behind is as small as possible.
  - Main memory quickly littered by blocks too small to satisfy memory allocation requests.

Dynamic Partitioning Placement Algorithm



Allocate 18K

First Fit

Next Fit

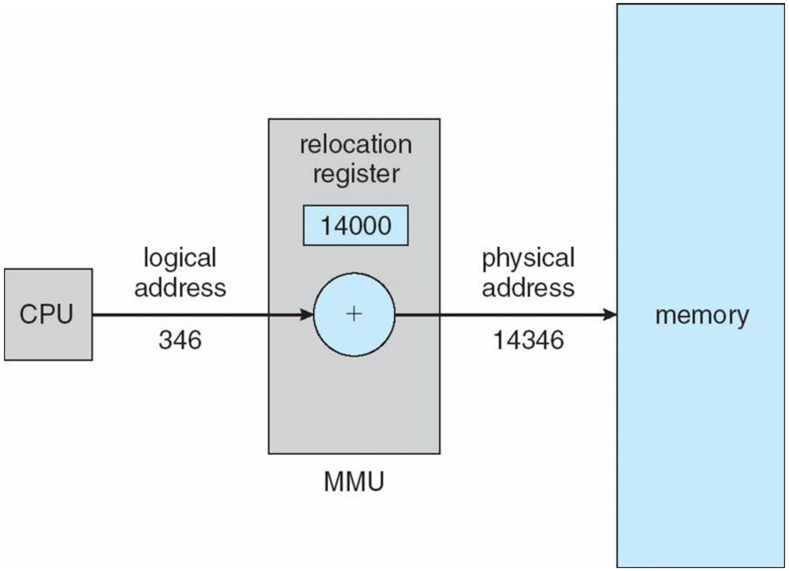
Best Fit

▨ Allocated block

□ Free block

Dynamic relocation using a relocation register

- Routine is not loaded until it is called
- Better memory-space utilization; unused routine is never loaded
- All routines kept on disk in relocatable load format
- Useful when large amounts of code are needed to handle infrequently occurring cases
- No special support from the operating system is required
  - Implemented through program design
  - OS can help by providing libraries to implement dynamic loading



Protection

- Protection
  - Processes should not be able to reference memory locations in another process without permission
  - Impossible to check absolute addresses at compile time
  - Must be checked at run time

Sharing

- Sharing
  - Allow several processes to access the same portion of memory
  - Better to allow each process to access the same copy of the program rather than have their own separate copy

Modules

- Logical Organization
  - Programs are written in modules
  - Modules can be written and compiled independently
  - Different degrees of protection given to modules (read-only, execute-only)
  - Share modules among processes

Memory Management

- Memory references are dynamically translated into physical addresses at run time
  - A process may be swapped in and out of main memory such that it occupies different regions
- A process may be broken into a number of pieces (pages or segments)

Execution of a Program

- Operating system brings into main memory a few pieces of the program
- Resident set - portion of process that is in main memory
- An interrupt is generated when an address is needed that is not in main memory
- Operating system places the process in a blocking state

Execution of a Program (ctd.)

- Piece of process that contains the logical address is brought into main memory
  - Operating system issues a disk I/O Read request
  - Another process is dispatched to run while the disk I/O takes place
  - An interrupt is issued when disk I/O complete which causes the operating system to place the affected process in the Ready state

Improved System Utilization

- More processes may be maintained in main memory
  - Only load in some of the pieces of each process
  - With so many processes in main memory, it is very likely a process will be in the Ready state at any particular time
- A process may be larger than all of main memory

Simple/Basic Paging

- Idea: Physical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available:
  - Avoids external fragmentation.
  - Avoids problem of varying sized memory chunks.
- Divide physical memory into fixed-sized chunks/blocks called frames (size is power of 2, usually between 512 bytes and 16 MB).
- Divide logical memory into blocks of same size pages.

Paging

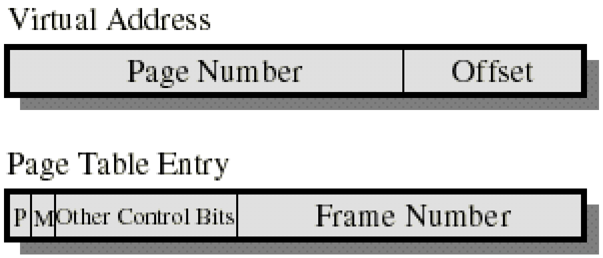
- **Partition memory into small equal fixed-size chunks and divide each process into the same size chunks**
- **The chunks of a process are called pages and chunks of memory are called (page) frames**
- **Divide physical memory into fixed-sized blocks called** frames

  Size is power of 2, between 512 bytes and 16 Mbytes

Divide logical memory into blocks of same size called pages

Paging

- Typically, each process has its own page table



P= present bit
M = Modified bit

- Each page table entry contains a present bit to indicate whether the page is in main memory or not.
  - If it is in main memory, the entry contains the frame number of the corresponding page in main memory
  - If it is not in main memory, the entry may contain the address of that page on disk or the page number may be used to index another table (often in the PCB) to obtain the address of that page on disk

Paging

- A modified bit indicates if the page has been altered since it was last loaded into main memory
  - If no change has been made, the page does not have to be written to the disk when it needs to be swapped out
- Other control bits may be present if protection is managed at the page level
  - a read-only/read-write bit
  - protection level bit: kernel page or user page (more bits are used when the processor supports more than 2 protection levels)

Simple/Basic Paging

- The process pages can thus be assigned to any free frames in main memory; a process does not need to occupy a contiguous portion of physical memory.
- Need to keep track of all free frames.
- To run a program of size n pages, need to find *n* free frames and load program.
- Need to set up a page table to translate logical to physical pages/addresses.
- Internal fragmentation possible only for page at end of program.

Demand Paging

- Bring a page into memory only when it is needed:
  - Less I/O needed, no unnecessary I/O
  - Less memory needed
  - Faster response
  - More users
- Page is needed ⇒ reference to it:
  - invalid reference ⇒ abort
  - not-in-memory ⇒ bring to memory
- Similar to paging system with swapping.
- Lazy swapper – never swaps a page into memory unless page will be needed; Swapper that deals with pages is a pager.

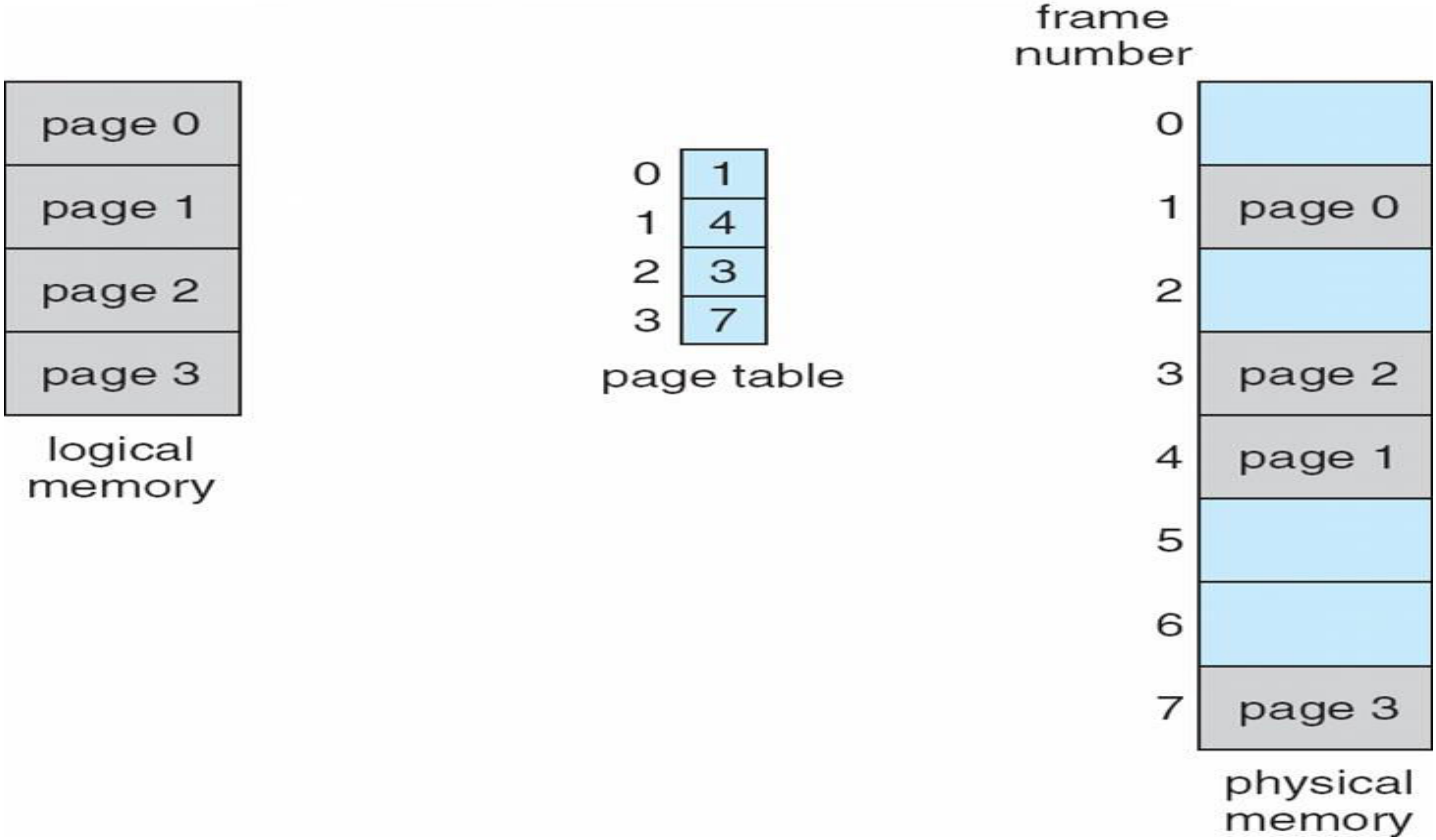
Page Table

- Operating system maintains a page table for each process
  - Contains the frame location for each page in the process
  - Memory address consist of a page number and offset within the page
  - Page tables are variable in length (depends on process size) then must be in main memory instead of registers
  - A single register holds the starting physical address of the page table of the currently running process

Sharing Pages

- If we share the same code among different users, it is sufficient to keep only one copy in main memory
- Shared code must be reentrant (ie: non self-modifying) so that 2 or more processes can execute the same code
- If we use paging, each sharing process will have a page table who’s entry points to the same frames: only one copy is in main memory
- But each user needs to have its own private data pages

Paging Example



page 0
page 1
page 2
page 3

logical memory

0	1
1	4
2	3
3	7

page table

frame number

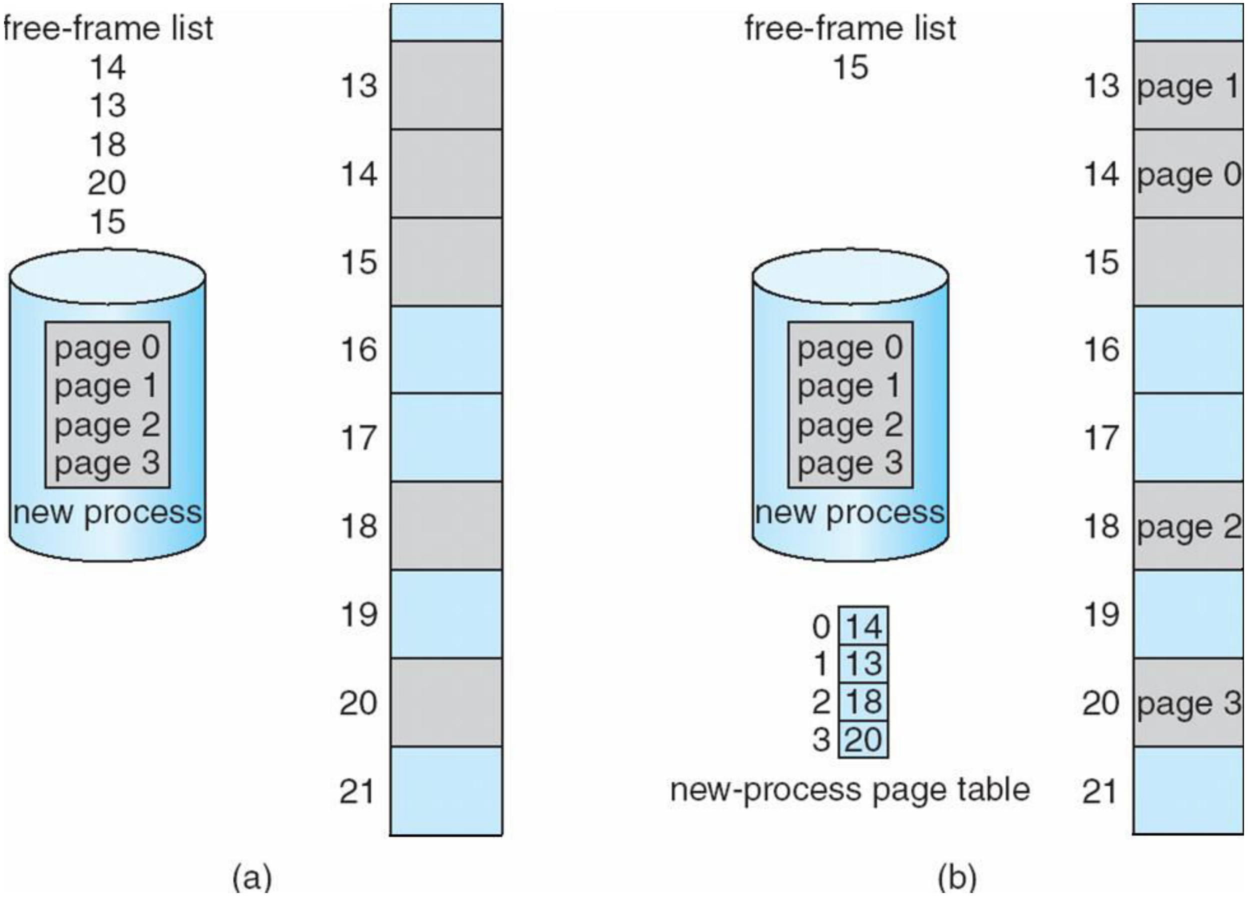
0	
1	page 0
2	
3	page 2
4	page 1
5	
6	
7	page 3

physical memory

Simple/Basic Paging

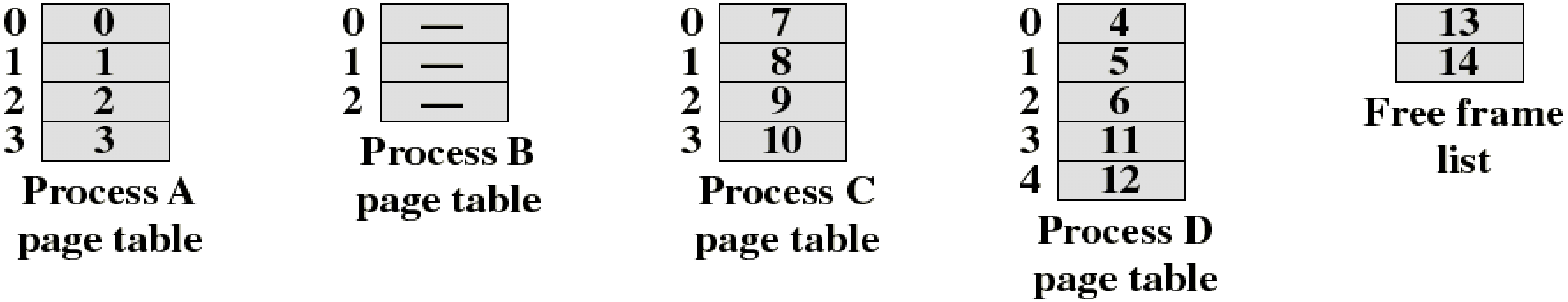
- To run a program of size *n* pages, need to find any *n* free frames and load all the program (pages).
- So need to keep track of all free frames in physical memory – use free-frame list.
- Free-frame list example in next slide.

Free Frames



Before allocation **After allocation**

Processes loading



0	0
1	1
2	2
3	3

Process A page table

0	—
1	—
2	—

Process B page table

0	7
1	8
2	9
3	10

Process C page table

0	4
1	5
2	6
3	11
4	12

Process D page table

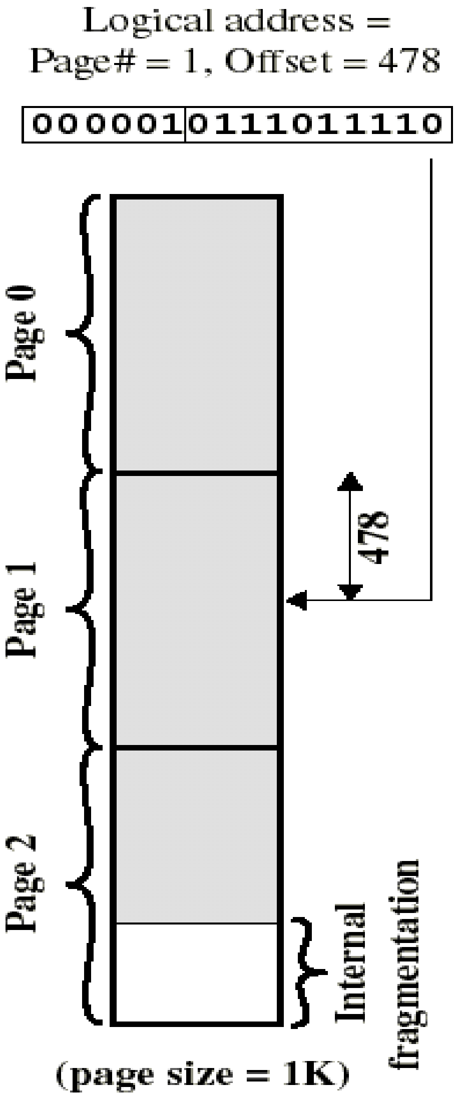
13
14

Free frame list

- The OS now needs to maintain (in main memory) a page table for each process.
- Each entry of a page table consists of the frame number where the corresponding page is physically located.
- The corresponding page table is indexed by the page number to obtain the frame number.
- A free frame table/list, of available pages, is maintained.

Logical address in paging

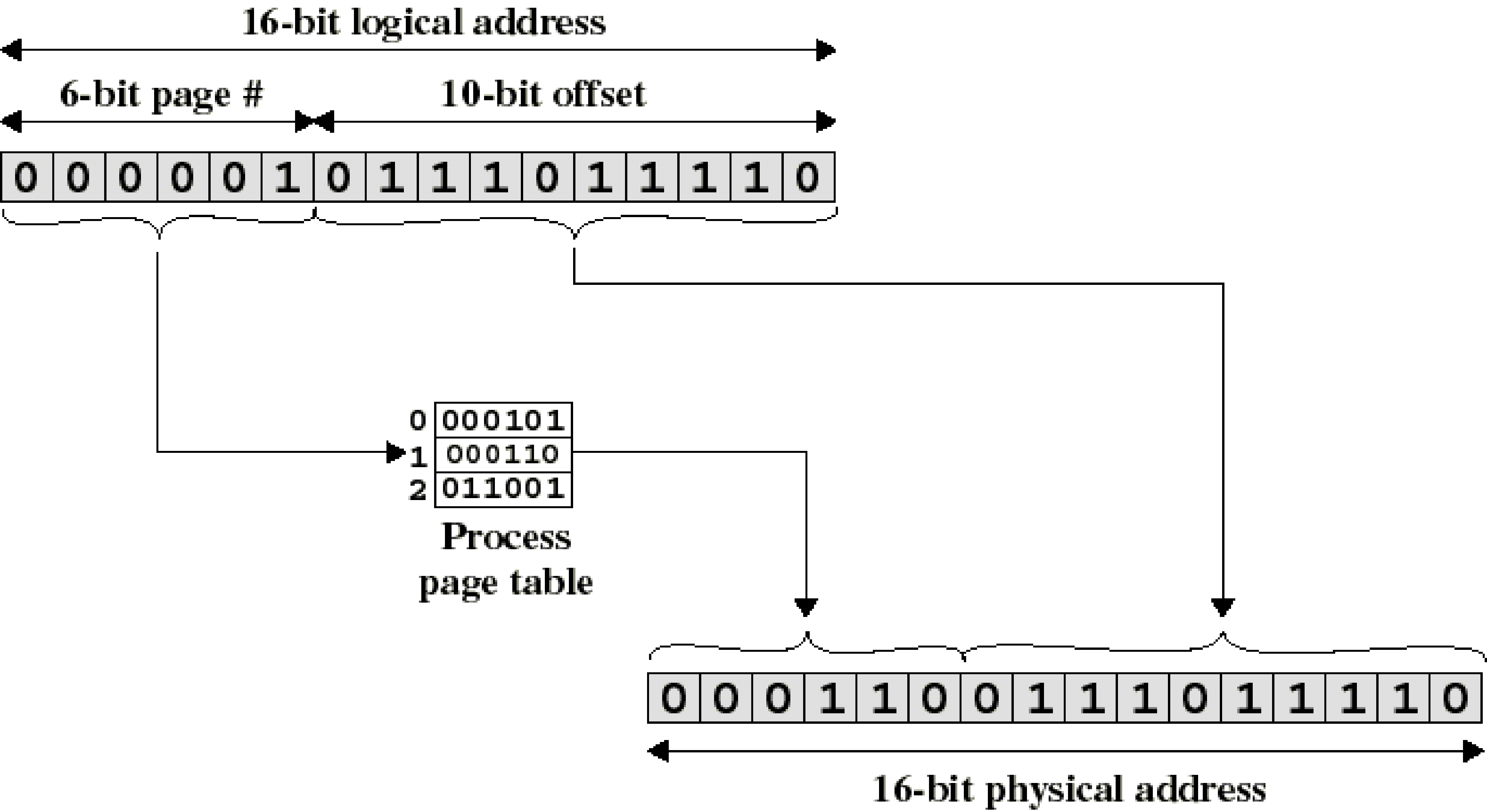
- The logical address becomes a relative address when the page size is a power of 2.
- Example: if 16 bits addresses are used and page size = 1K, we need 10 bits for offset and have 6 bits available for page number.
- Then the 16 bit address, obtained with the 10 least significant bits as offset and 6 most significant bits as page number, is a location relative to the beginning of the process.



Logical address used in paging

- Within each program, each logical address must consist of a page number and an offset within the page.
- A dedicated register always holds the starting physical address of the page table of the currently running process.
- Presented with the logical address (page number, offset) the processor accesses the page table to obtain the physical address (frame number, offset).

Logical-to-Physical Address Translation in Paging



0	000101
1	000110
2	011001

Address Translation Scheme

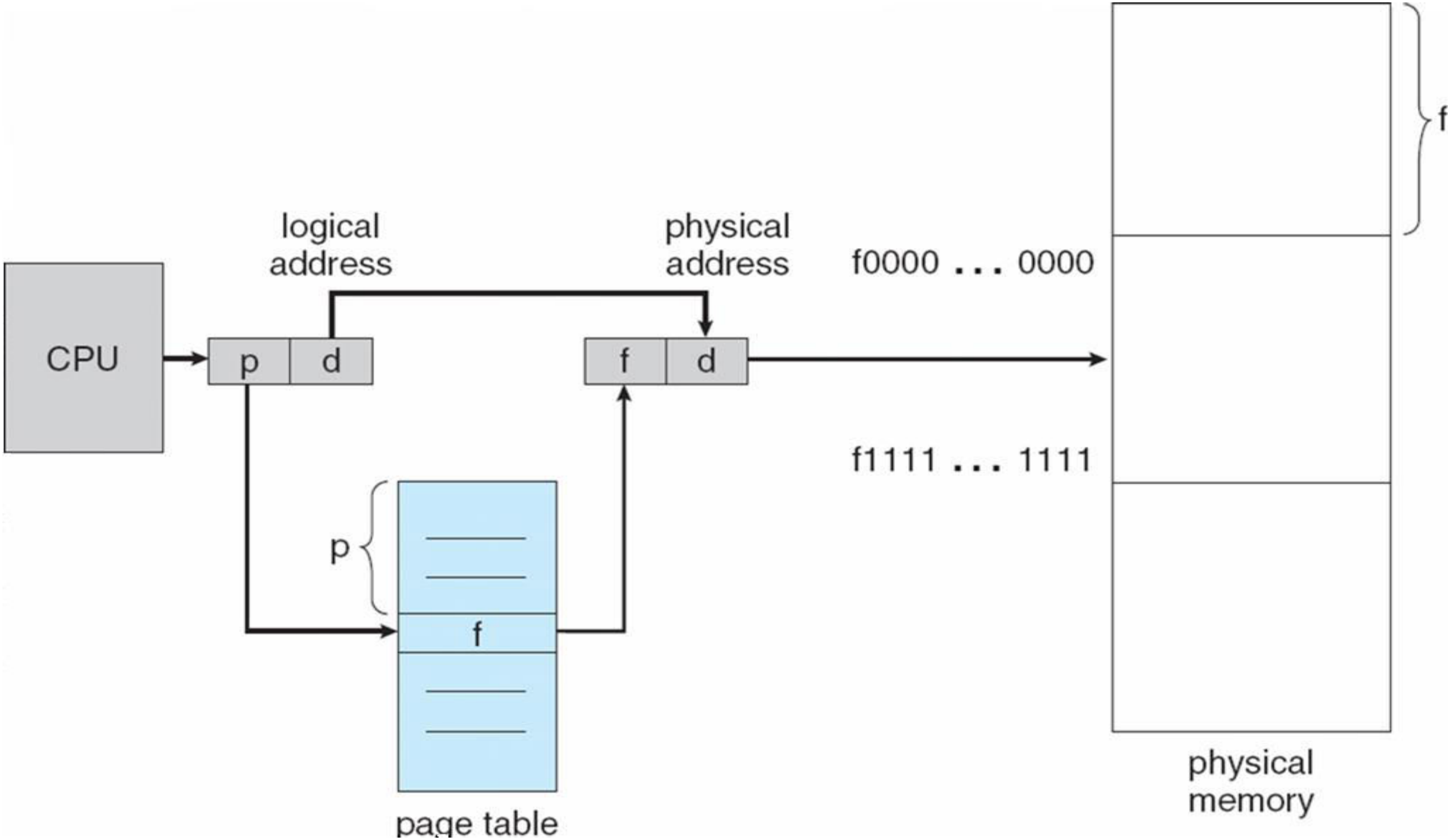
- Logical address generated by CPU is divided into two parts:
  - *Page number (p)* – used as an index into a *page table* which contains the base address of each page in physical memory.
  - *Page offset/displacement (d)* – combined with base address to define the physical memory address that is sent to the memory unit.
- For given logical address space 2^m and page size 2^n.

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

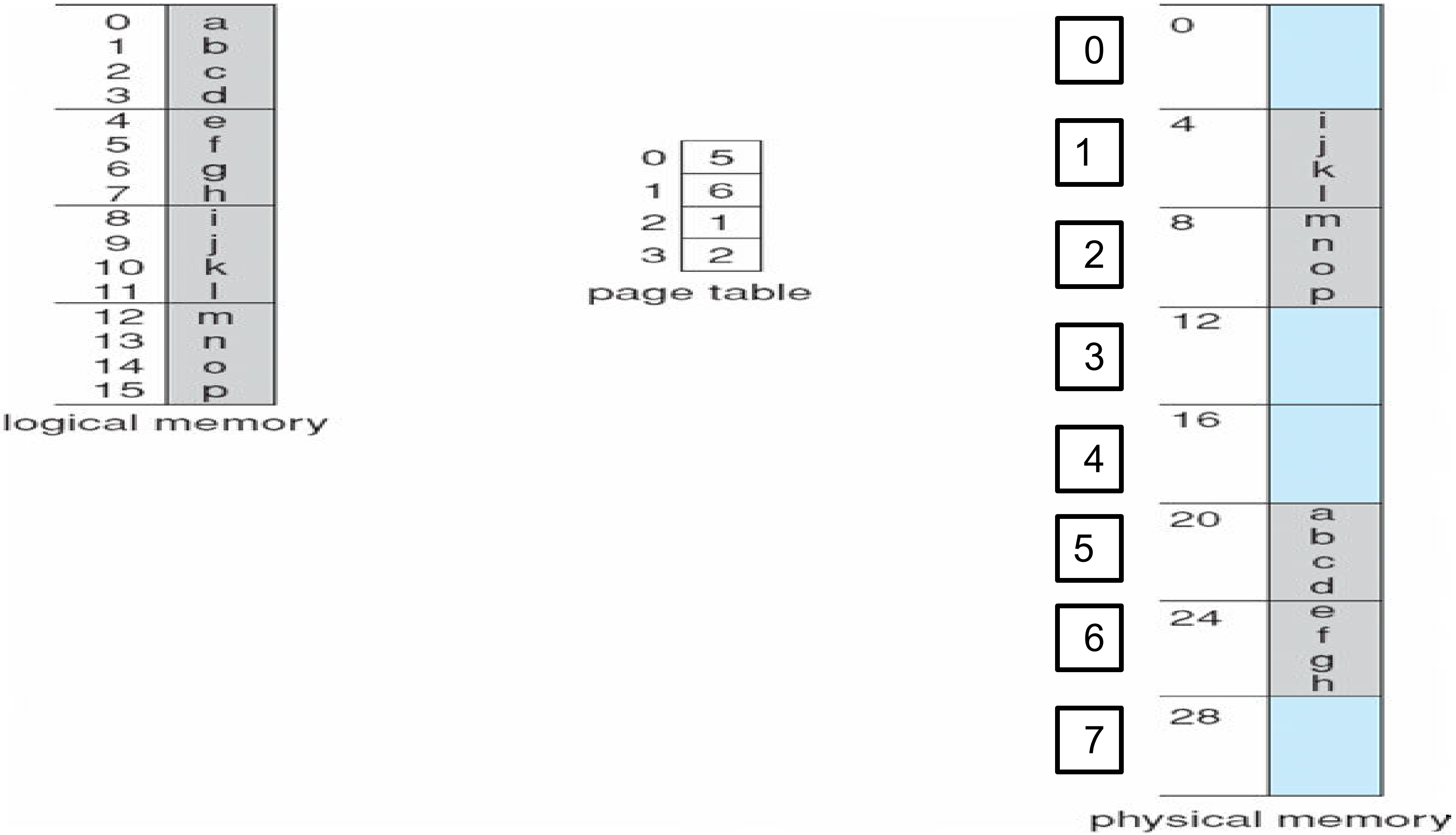
Address Translation Scheme

- By using a page size of a power of 2, the pages are invisible to the programmer, compiler/assembler, and the linker.
- Address translation at run-time is then easy to implement in hardware:
  - logical address (p, d) gets translated to physical address (f, d) by indexing the page table with p and appending the same displacement/offset d to the frame number f.

Address Translation Architecture



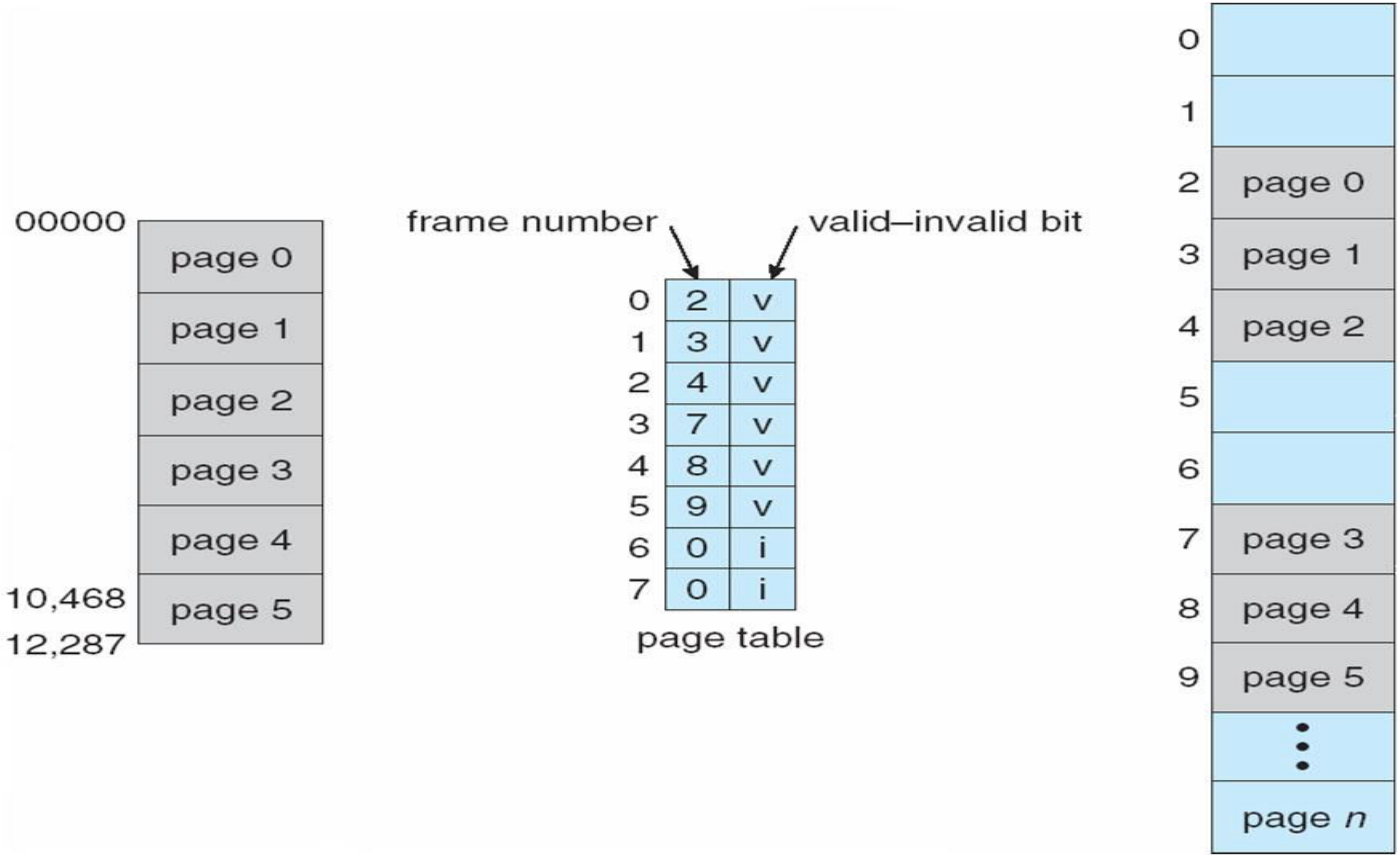
Paging Example



Memory Protection

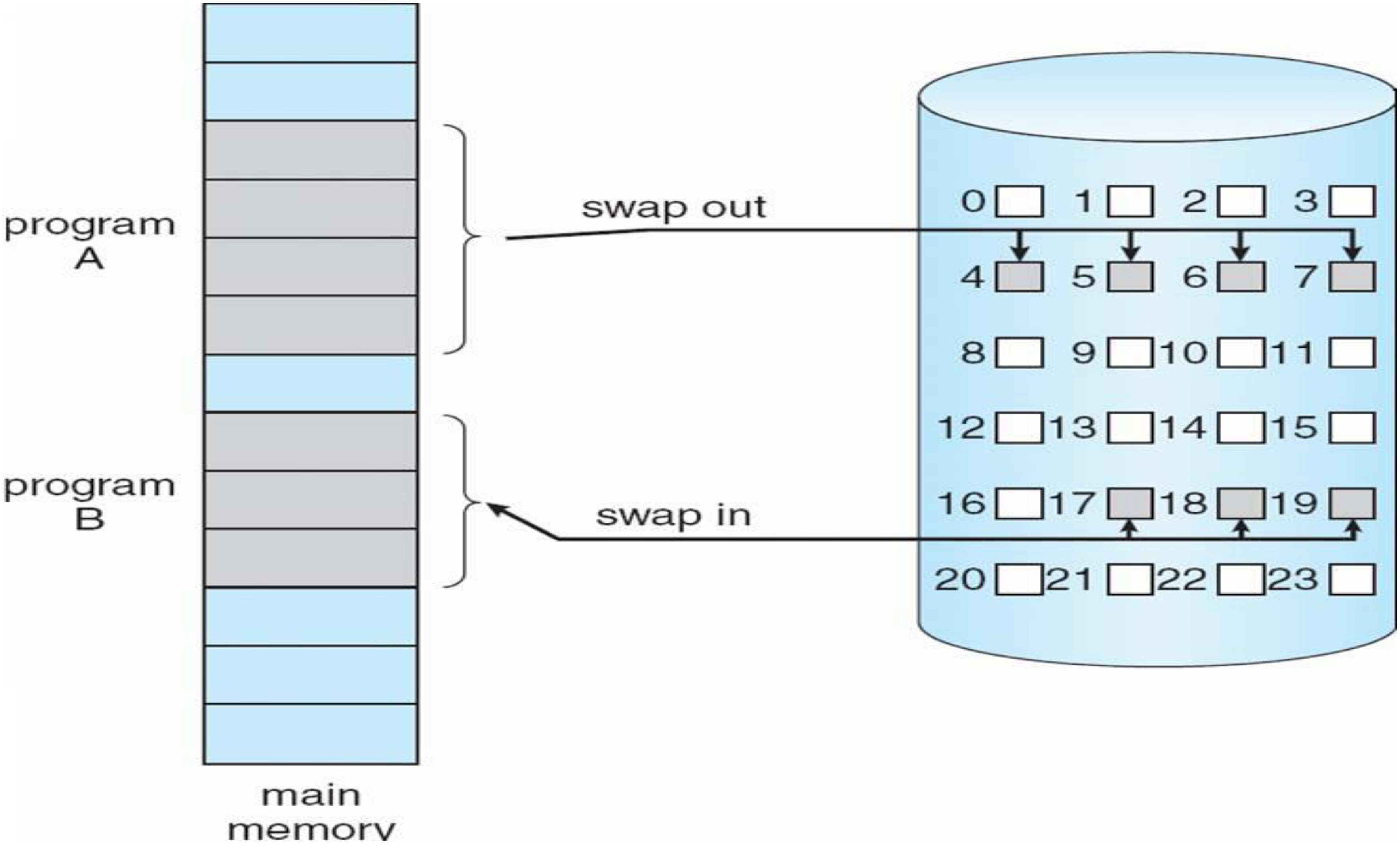
- Memory protection implemented by associating protection bit with each frame to indicate if read-only or read-write access is allowed:
  - Can also add more bits to indicate page execute-only, and so on.
- Valid-invalid bit attached to each entry in the page table:
  - "valid" indicates that the associated page is in the process' logical address space, and is thus a legal page.
  - "invalid" indicates that the page is not in the process' logical address space.

Valid (v) or Invalid (i) Bit in a Page Table



	frame number	valid–invalid bit
0	2	v
1	3	v
2	4	v
3	7	v
4	8	v
5	9	v
6	0	i
7	0	i

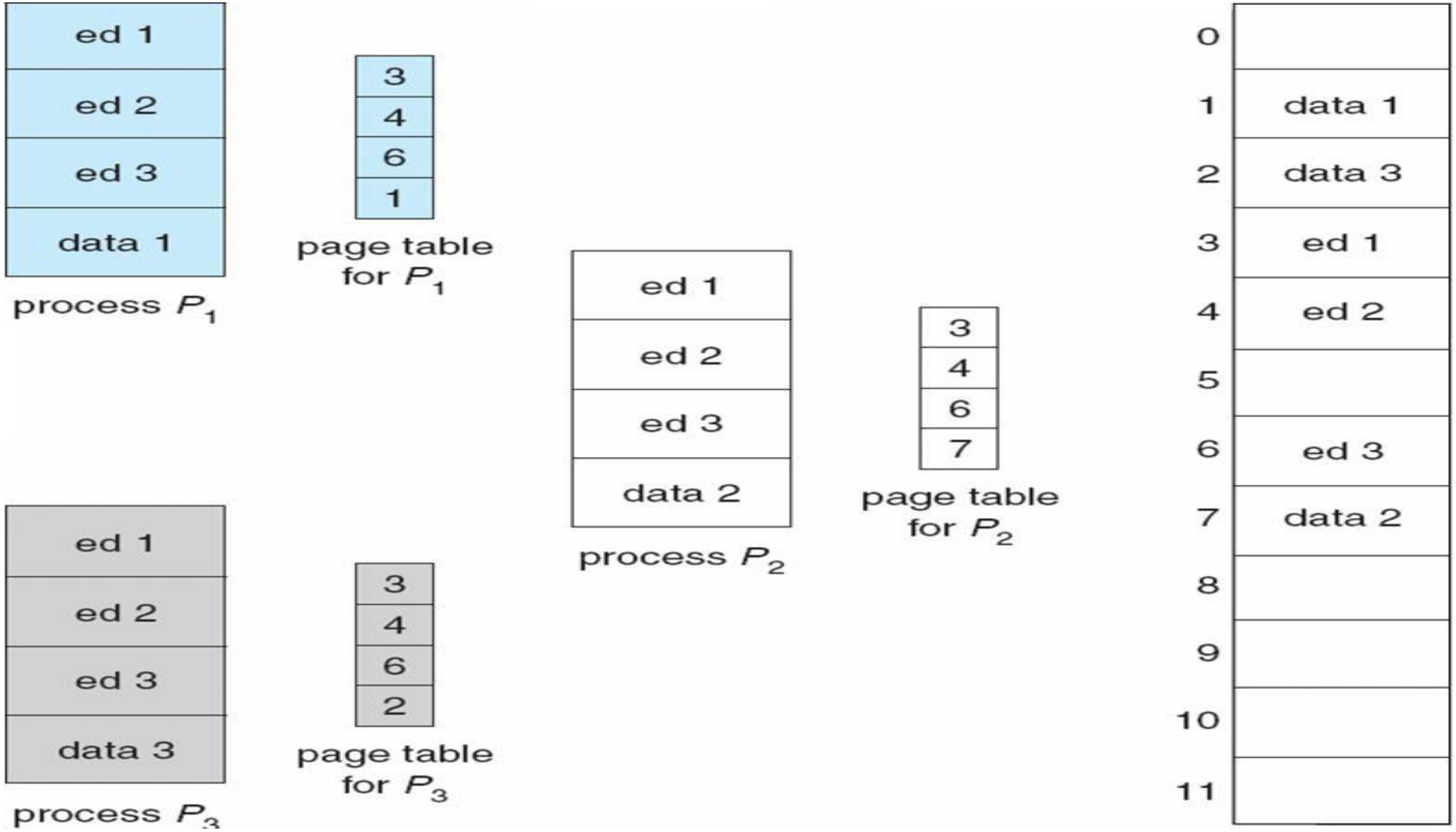
Transfer of a Paged Memory to Contiguous Disk Space



Shared Pages

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

Shared Pages Example



ed 1
ed 2
ed 3
data 1
process P_1

3
4
6
1
page table for P_1

ed 1
ed 2
ed 3
data 2
process P_2

3
4
6
7
page table for P_2

ed 1
ed 2
ed 3
data 3
process P_3

3
4
6
2
page table for P_3

0	
1	data 1
2	data 3
3	ed 1
4	ed 2
5	
6	ed 3
7	data 2
8	
9	
10	
11	

Segmentation

- A program can be subdivided into segments
  - Segments may vary in length
  - There is a maximum segment length
- Addressing consist of two parts
  - a segment number and
  - an offset
- Segmentation is similar to dynamic partitioning

Simple/Basic Segmentation

- Paging division is arbitrary; no natural/logical boundaries for protection/sharing.
- Segmentation supports user’s view of a program.
- A program is a collection of segments – logical units – such as:

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

Segmentation

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

main program

procedure

function

method

object

local variables, global variables

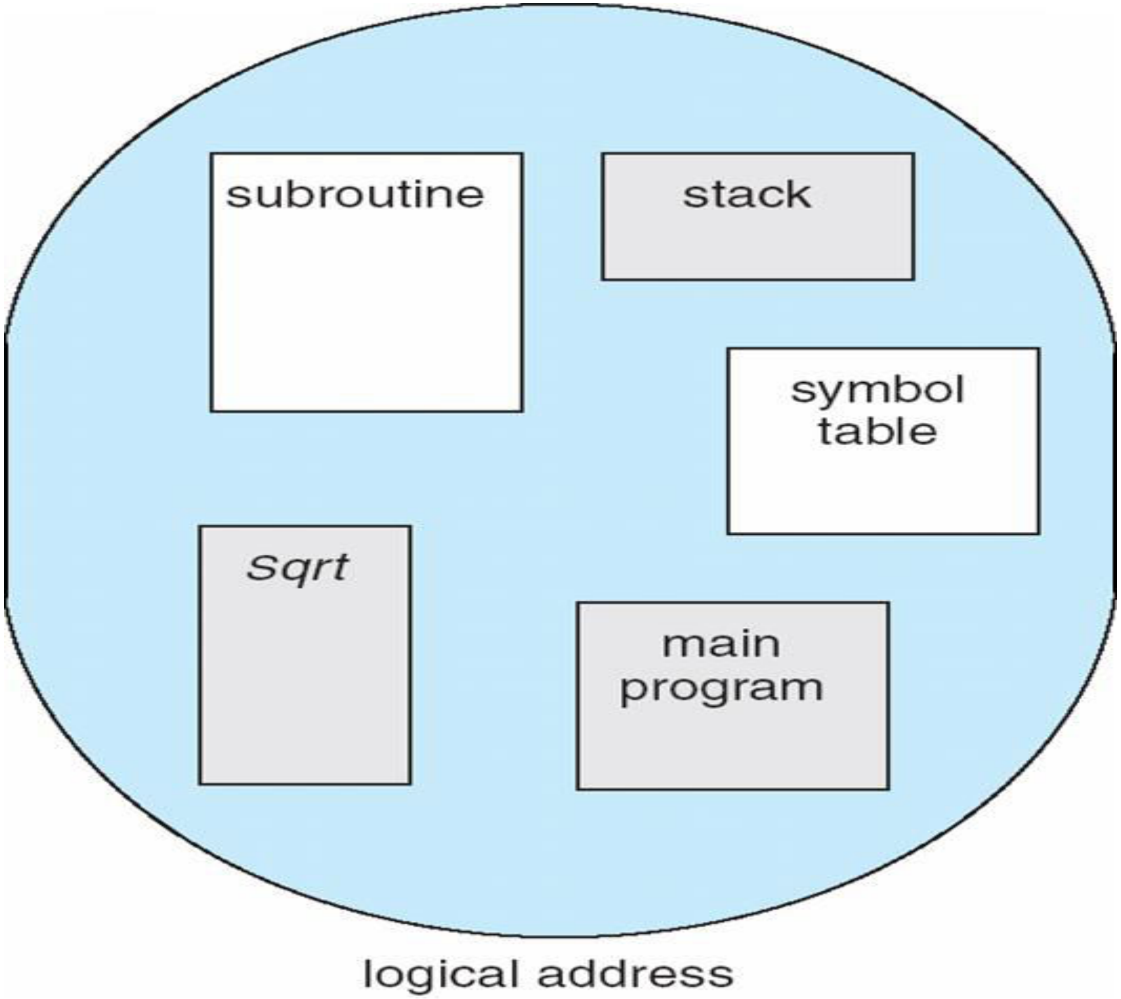
common block

stack

symbol table

arrays

User’s View of a Program

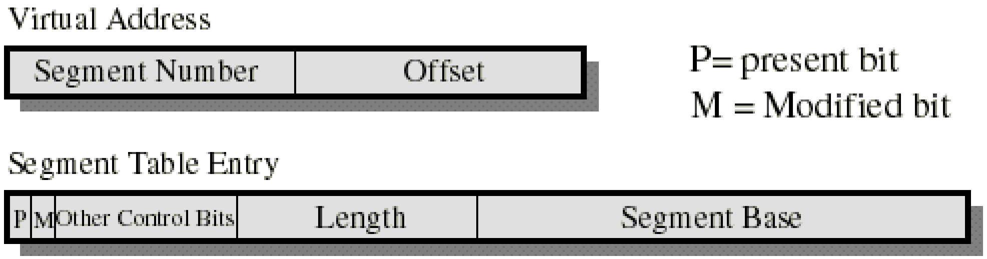


Dynamics of Simple Segmentation

- Each program is subdivided into blocks of non-equal size called segments.
- When a process gets loaded into main memory, its different segments can be located anywhere.
- Each segment is fully packed with instructions/data; no internal fragmentation.
- There is external fragmentation; it is reduced when using small segments.

Dynamics of Segmentation

- Typically, each program has its own segment table.



Virtual Address

Segment Number	Offset

P= present bit
M = Modified bit

Segment Table Entry

P	M	Other Control Bits	Length	Segment Base

- Similarly to paging, each segment table entry contains a present (valid-invalid) bit and a modified bit.
- If the segment is in main memory, the entry contains the starting address and the length of that segment.
- Other control bits may be present if protection and sharing is managed at the segment level.
- Logical to physical address translation is similar to paging except that the offset is added to the starting address (instead of appended).

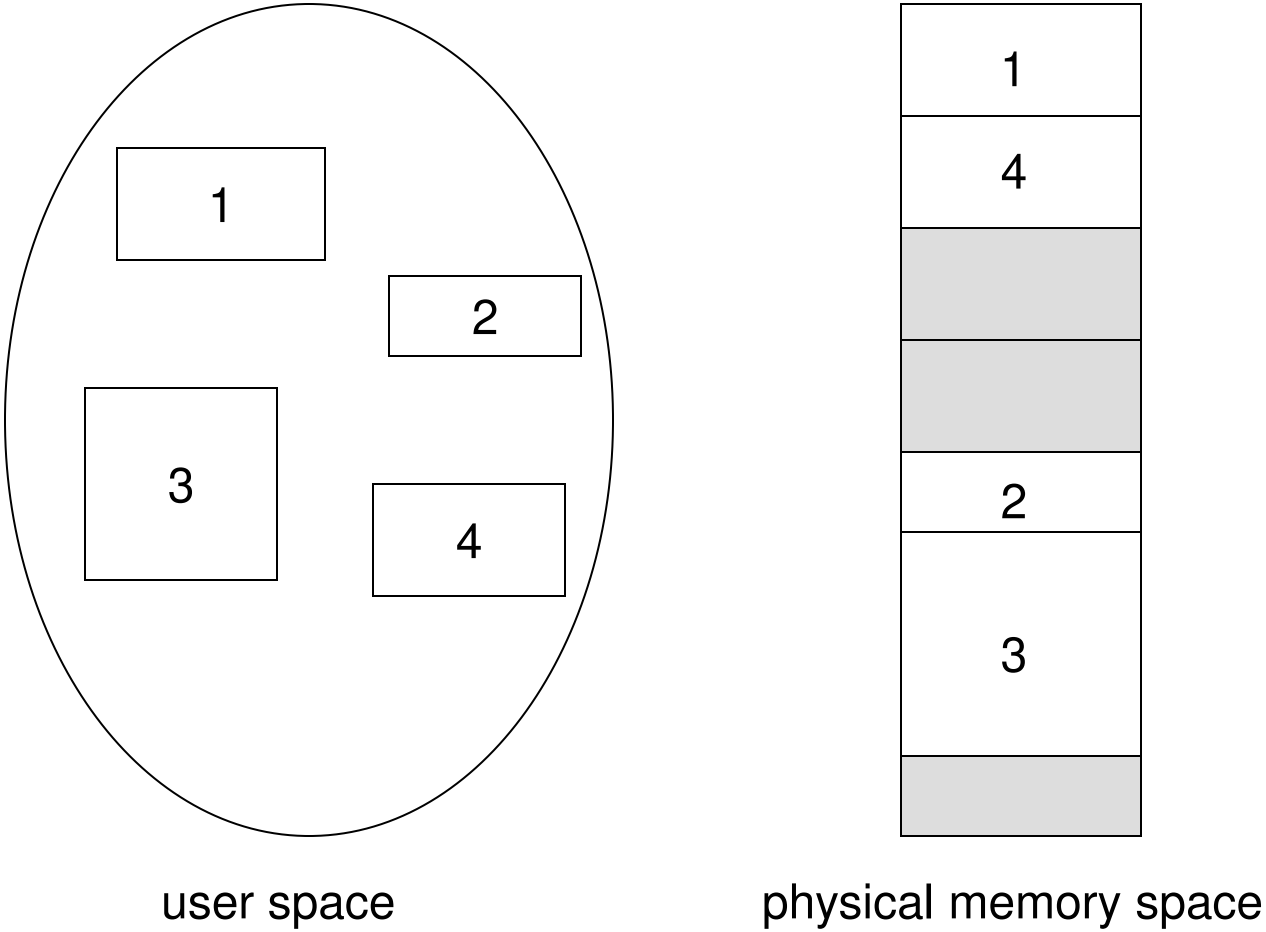
Segmentation

- All segments of all programs do not have to be of the same length
- There is a maximum segment length
- Addressing consist of two parts - a segment number and an offset
- Since segments are not equal, segmentation is similar to dynamic partitioning

Segmentation: comments

- In each segment table entry we have both the starting address and length of the segment
  - the segment can thus dynamically grow or shrink as needed
  - address validity easily checked with the length field
- But variable length segments introduce external fragmentation and are more difficult to swap in and out...
- It is natural to provide protection and sharing at the segment level since segments are visible to the programmer (pages are not)
- Useful protection bits in segment table entry:
  - read-only/read-write bit
  - Supervisor/User bit

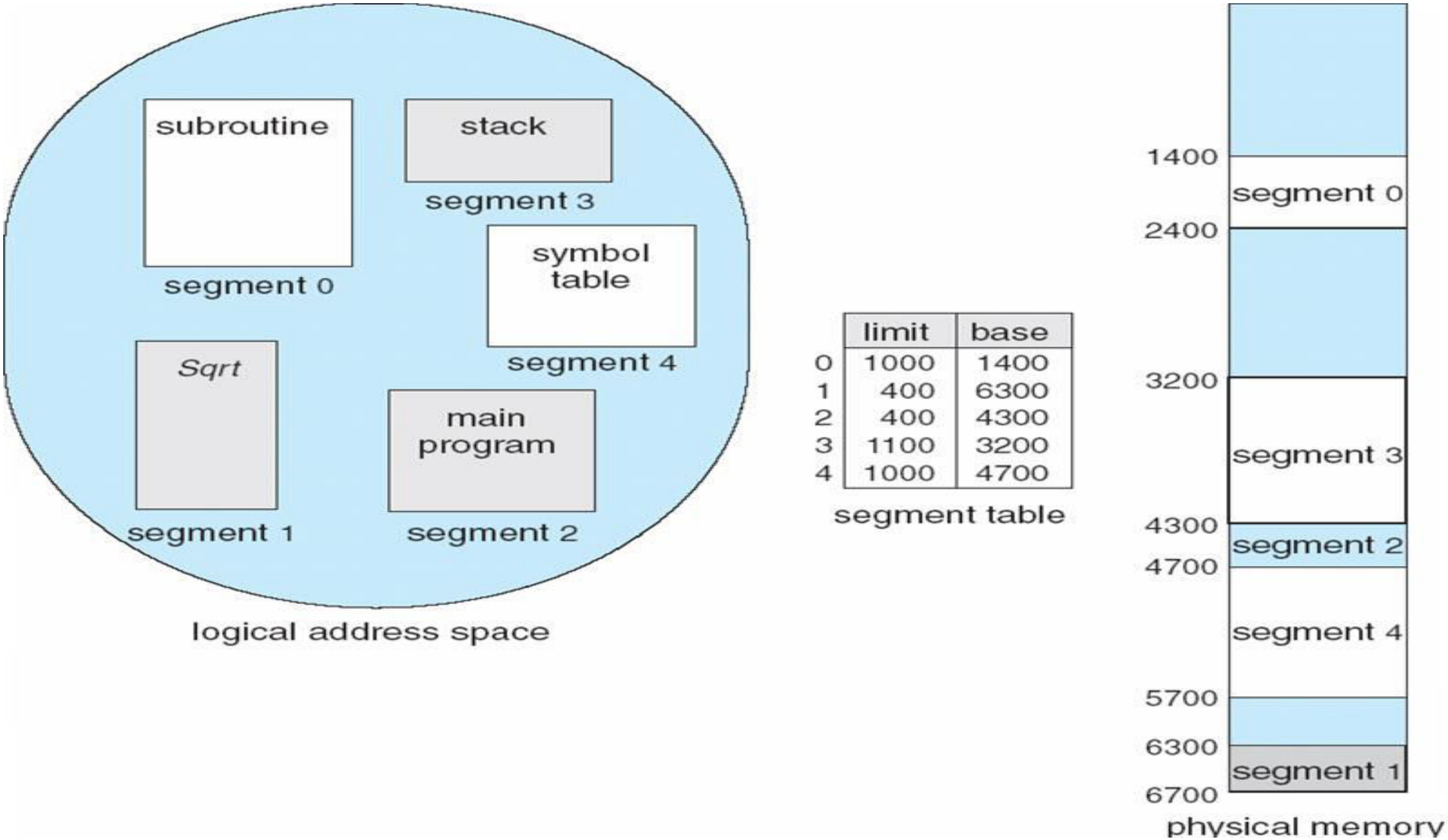
Logical view of simple segmentation



Dynamics of Simple Segmentation

- In contrast with paging, segmentation is visible to the programmer:
  - provided as a convenience to organize logically programs (example: data in one segment, code in another segment).
  - must be aware of segment size limit.
- The OS maintains a segment table for each process. Each entry contains:
  - the starting physical addresses of that segment.
  - the length of that segment (for protection).

Example of Segmentation



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

segment table

Logical address used in segmentation

- When a process enters the Running state, a dedicated register gets loaded with the starting address of the process's segment table.
- Presented with a logical address (segment number, offset) = (s, d), the CPU indexes (with s) the segment table to obtain the starting physical address b and the length l of that segment.
- The physical address is obtained by adding d to b (in contrast with paging):
  - the hardware also compares the offset d with the length l of that segment to determine if the address is valid.

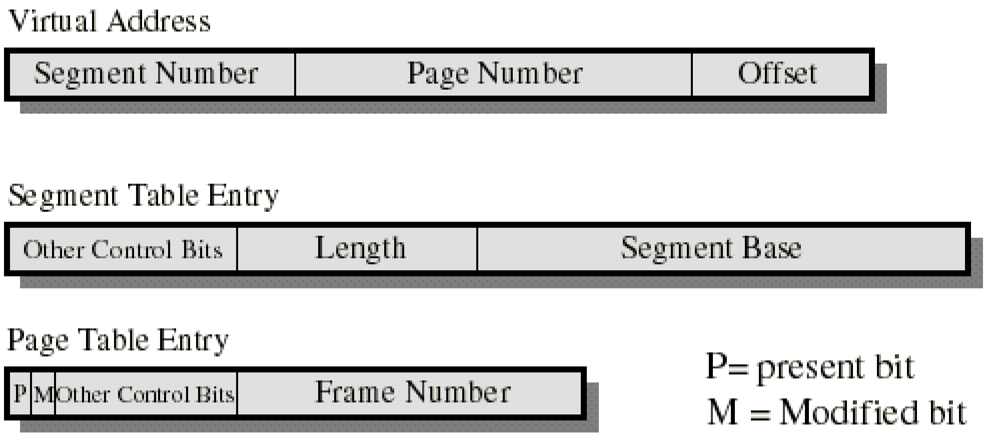
Sharing in Segmentation Systems

- Segments are shared when entries in the segment tables of 2 different processes point to the same physical locations
- Ex: the same code of a text editor can be shared by many users
  - Only one copy is kept in main memory
- but each user would still need to have its own private data segment

Combined Segmentation and Paging

- To combine their advantages some processors and OS page the segments.
- Several combinations exists. Here is a simple one
- Each process has:
  - one segment table
  - several page tables: one page table per segment
- The virtual address consist of:
  - a segment number: used to index the segment table who's entry gives the starting address of the page table for that segment
  - a page number: used to index that page table to obtain the corresponding frame number
  - an offset: used to locate the word within the frame

Simple Combined Segmentation and Paging



Virtual Address

Segment Number	Page Number	Offset

Segment Table Entry

Other Control Bits	Length	Segment Base

Page Table Entry

P	M	Other Control Bits	Frame Number

P= present bit
M = Modified bit

- The Segment Base is the physical address of the page table of that segment
- Present and modified bits are present only in page table entry
- Protection and sharing info most naturally resides in segment table entry
  - Ex: a read-only/read-write bit, a kernel/user bit...

Simple segmentation and paging comparison

- Segmentation requires more complicated hardware for address translation
- Segmentation suffers from external fragmentation
- Paging only yield a small internal fragmentation
- Segmentation is visible to the programmer whereas paging is transparent
- Segmentation can be viewed as commodity offered to the programmer to organize logically a program into segments and using different kinds of protection (ex: execute-only for code but read-write for data)
    - for this we need to use protection bits in segment table entries

Paging vs Segmentation

1. Paging is a memory management technique in which process address space is broken into blocks of the same size called pages (size is power of 2, between 512 bytes and 8192 bytes).

In paging, program is divided into fixed or mounted size pages.

2.For paging operating system is accountable.

3.Page size is determined by hardware.

4.It is faster in the comparison of segmentation.

1. Segmentation is a memory management technique in which each job is divided into several segments of different sizes, one for each module that contains pieces that perform related functions.

In segmentation, program is divided into variable size sections.

2. For segmentation compiler is accountable.

3. Here, the section size is given by the user.

4. Segmentation is slow.

Paging vs Segmentation

5.Paging could result in internal fragmentation.

6.In paging, logical address is split into page number and page offset.

7.Paging comprises a page table which encloses the base address of every page.

8.Page table is employed to keep up the page data.

5. Segmentation could result in external fragmentation.

6. Here, logical address is split into section number and section offset.

7. While segmentation also comprises the segment table which encloses segment number and segment offset.

8. Section Table maintains the section data.

Paging vs Segmentation

9. In paging, operating system must maintain a free frame list.

10. Paging is invisible to the user.

11. In paging, processor needs page number, offset to calculate absolute address.

9. In segmentation, operating system maintain a list of holes in main memory.

10. Segmentation is visible to the user.

11. In segmentation, processor uses segment number, offset to calculate full address.