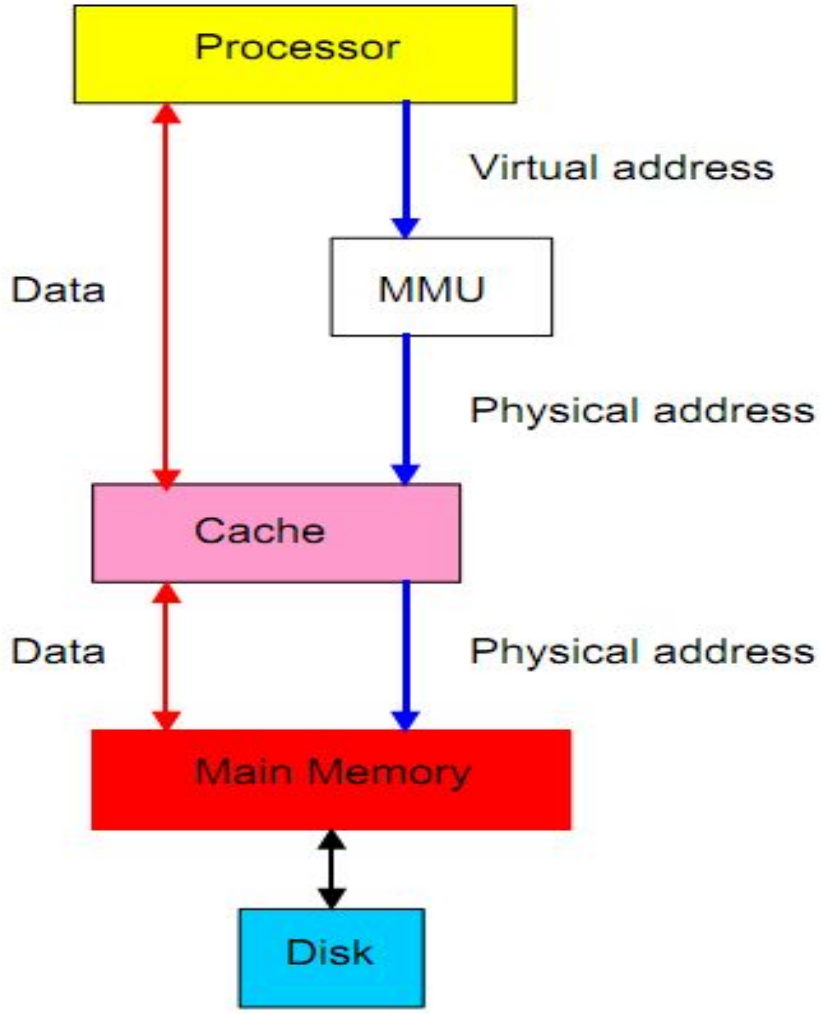
Module IV: Memory Management

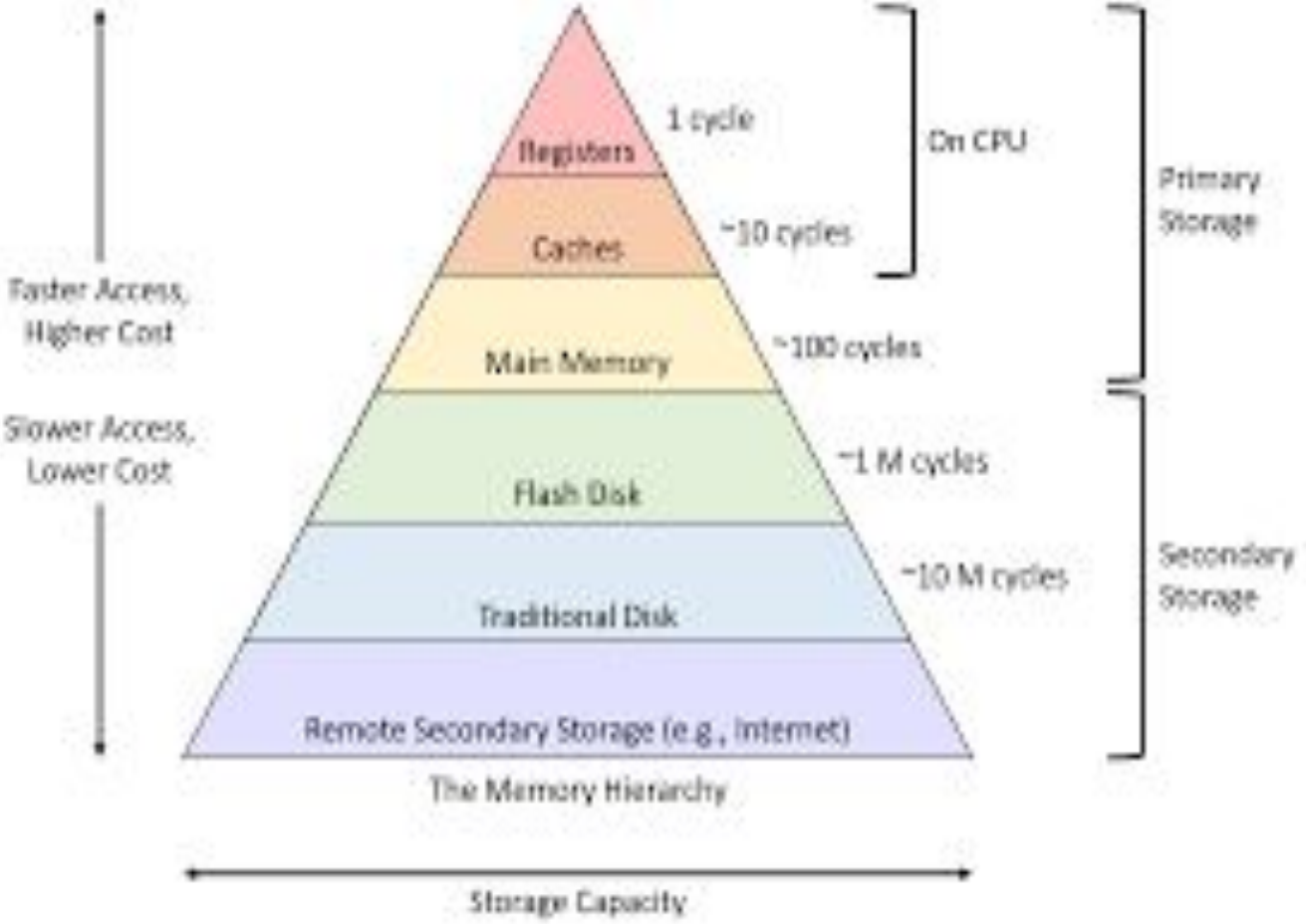
- Memory Management: Basics - Swapping -Memory Allocation (fixed partitions, variable partitions) Fragmentation - Paging - Segmentation - Virtual memory concepts – Demand paging - Page replacement algorithms (FIFO, Optimal, LRU) – Allocation of frames - Thrashing.

Memory Management Basics

- Don't have infinite RAM
- Do have a memory hierarchy-
  - Cache (fast)
  - Main(medium)
  - Disk(slow)
- Memory manager has the job of using this hierarchy to create an abstraction (illusion) of easily accessible memory



Memory Hierarchy



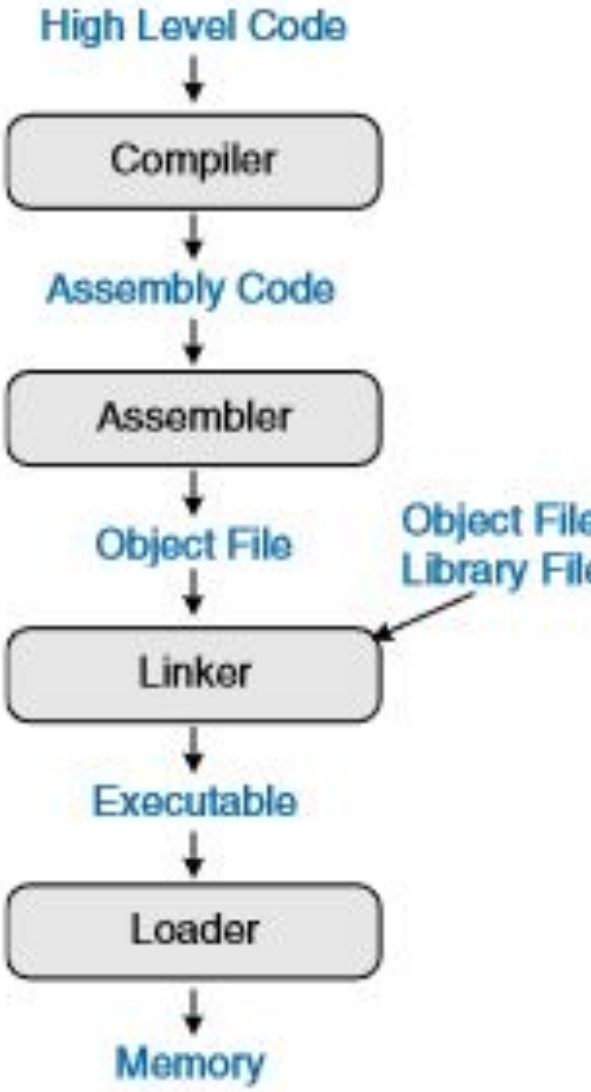
The Memory Hierarchy

Background

- Program must be brought into memory and placed within a process for it to be executed.
- *Input queue* – collection of processes on the disk that are waiting to be brought into memory for execution.
- User programs go through several steps before being executed.

Steps before program execution

- Step-1: compilation
- Compiler translates high-level code into assembly language
- Step 2: Assembling
- Assembler turns assembly language code into an object file containing machine language code.
- Step 3: Linking
- Linker combines all of the object files into one machine language file called the executable.
- Step 4: Loader
- Loader loads the program into memory and starts execution.
- In practice, most compilers perform all three steps



Binding of Instructions and Data to Memory

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

- **Compile time**: If memory location known a priori, absolute code can be generated; must recompile code if starting location changes.
- **Load time**: Compiler must generate *relocatable* code if memory location is not known at compile time and final binding is delayed until load time. Only reload user code if starting location changes.
- **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*).

Dynamic Linking

- Linking postponed until execution time.
- A **dynamic linker** is part of OS that loads and links shared libraries needed by an executable when it is executed (at "run time"), by copying the content of libraries from persistent storage to RAM, filling jump tables and relocating pointers.
- Small piece of code, *stub*, used to locate the appropriate memory-resident library routine.
- Stub replaces itself with address of routine, and executes routine.
- OS need to check if routine is in processes' memory address.

Overlays

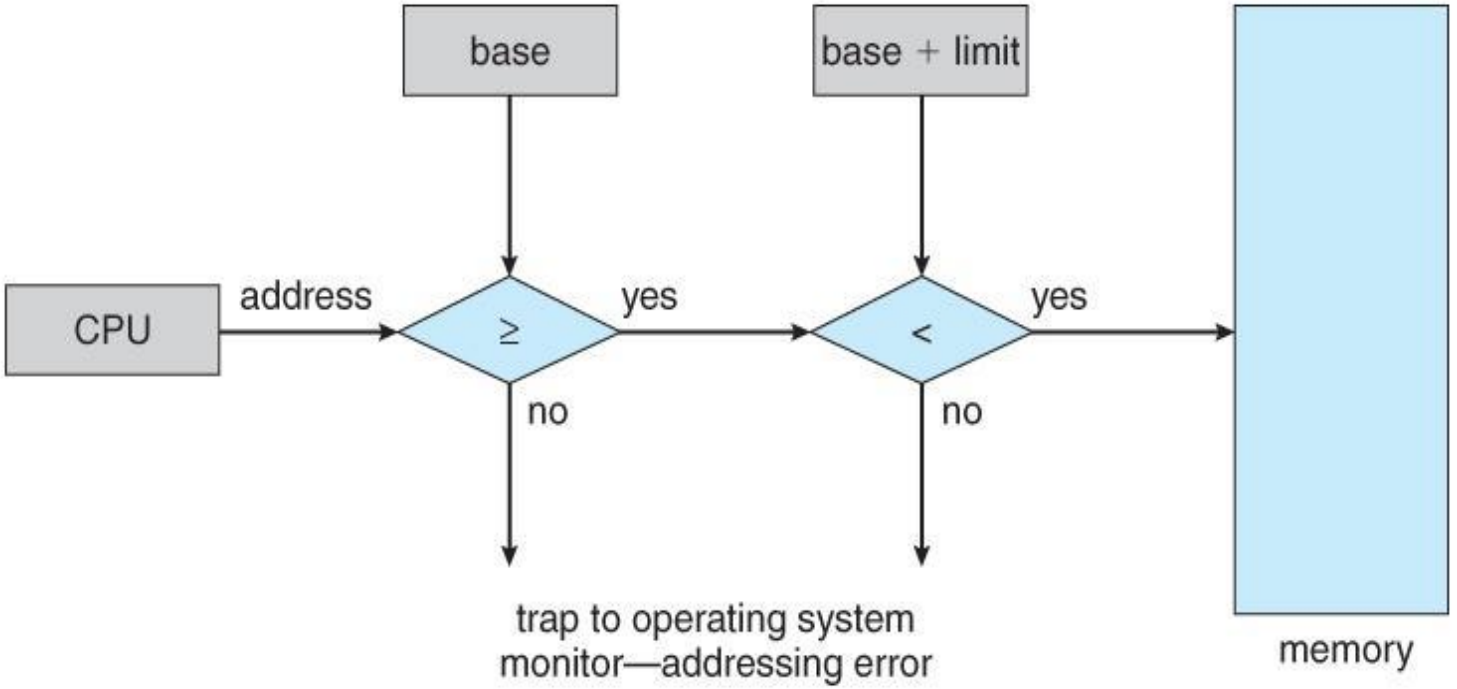
- Overlays-programmer breaks program into pieces which are swapped in by overlay manager
- Needed when process is larger than memory allocated to it.
- Idea of overlay is to keep in memory only those instructions and data that are needed at any given time.
- When other instructions are needed, they are loaded into space that was occupied previously by instructions that are no longer needed.
- Implemented by user, no special support needed from operating system, programming design of overlay structure is complex.
- Define overlays in the program (divide program codes into

Memory Protection

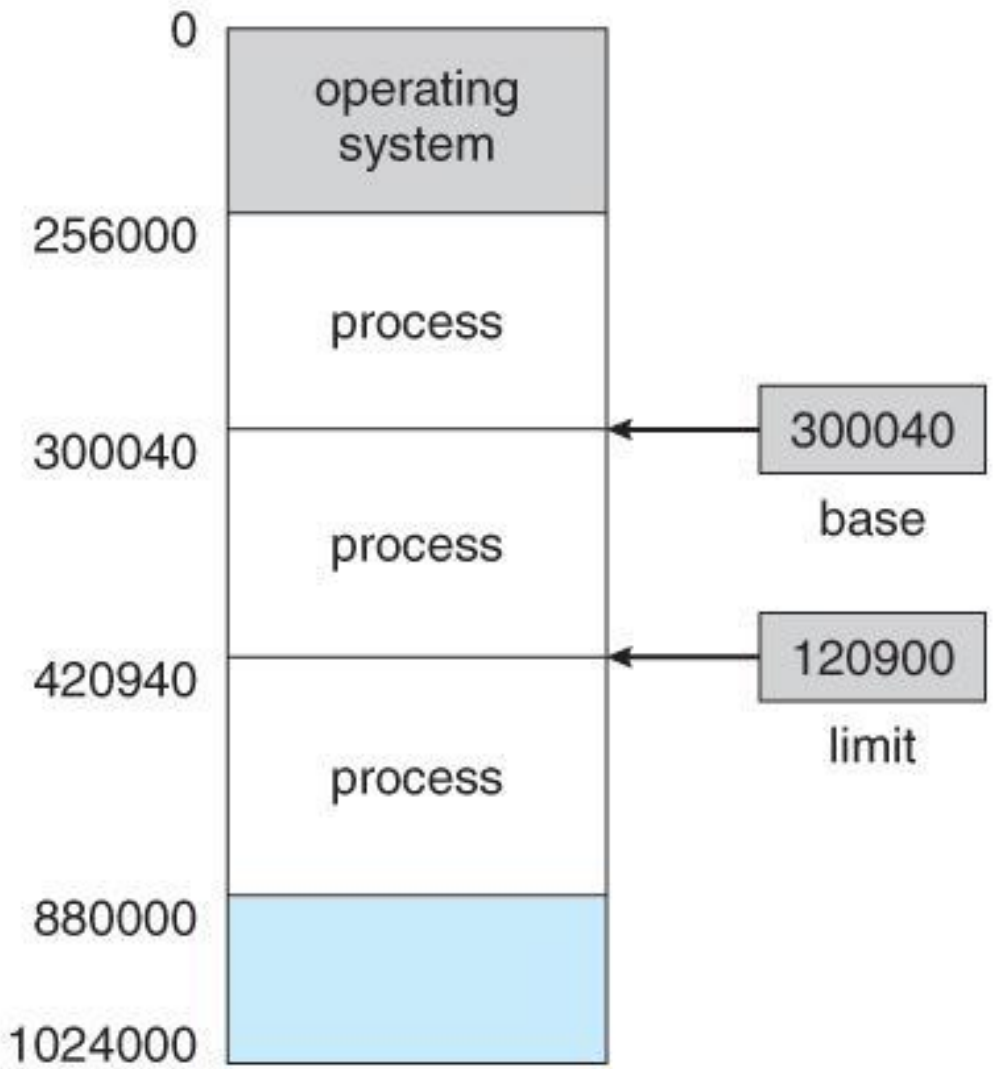
- We can prevent a process from accessing memory it does not own with a relocation register, together with a limit register
- This scheme is used to protect user processes from each other, and from changing operating-system code and data.
- Relocation register contains value of smallest physical address;
- limit register contains range of logical addresses – each logical address must be less than the limit register.
- The MMU maps the logical address dynamically by adding the value in the relocation register. This mapped addressis sent to memory

Hardware support for relocation and limit registers

- The relocation-register scheme provides an effective way to allow the operating system's size to change dynamically.



Example


0
operating system
256000
process
300040
process
420940
process
880000
1024000
300040
base
120900
limit

Logical vs. Physical Address Space

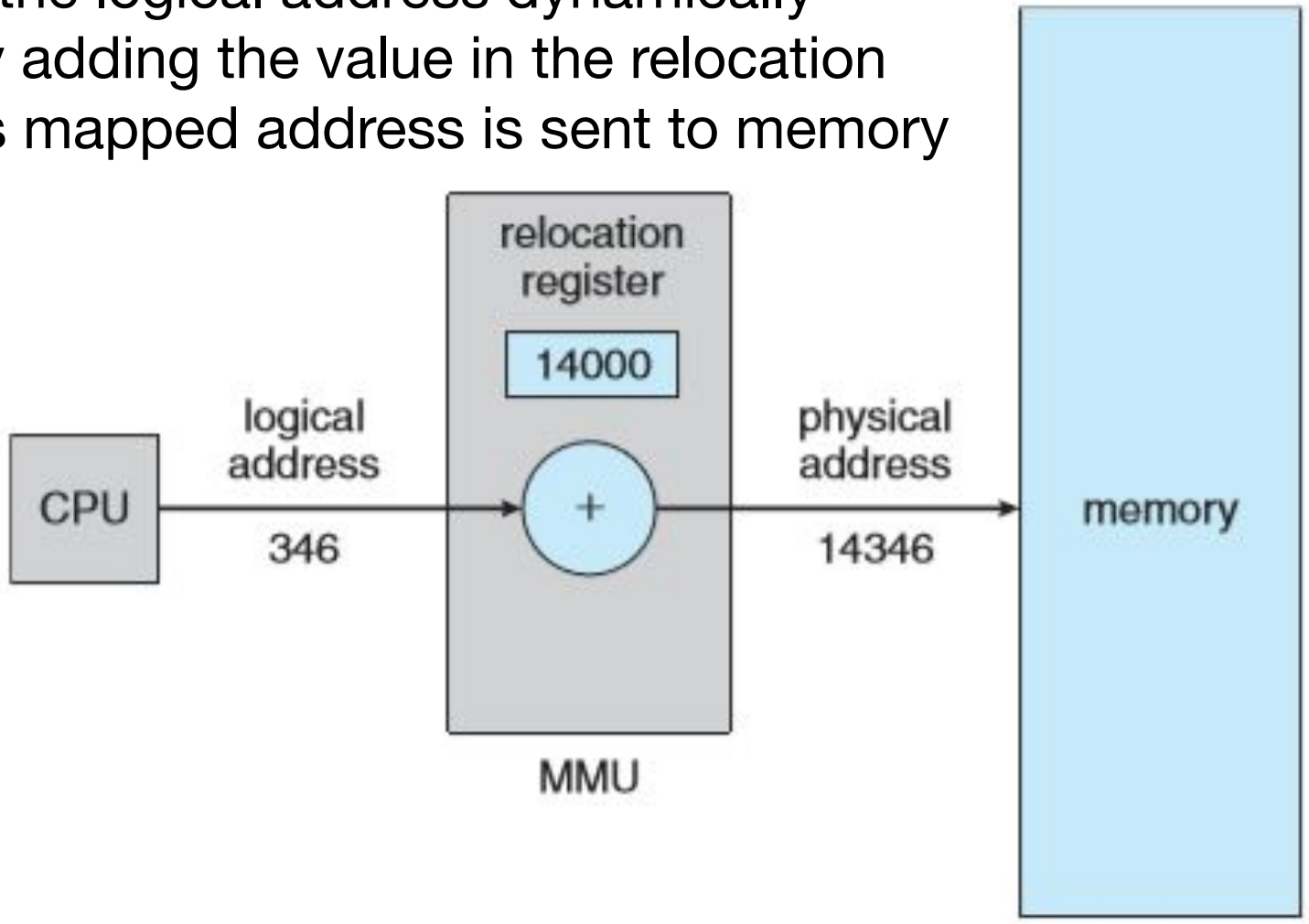
- The concept of a logical *address space* that is bound to a separate *physical address space* is central to proper memory management.
  - *Logical address* – generated by the CPU; also referred to as *virtual address*.
  - *Physical address* – address seen by the memory unit.
- Logical and physical addresses are the same in compile-time and load-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme.

Memory-Management Unit (MMU)

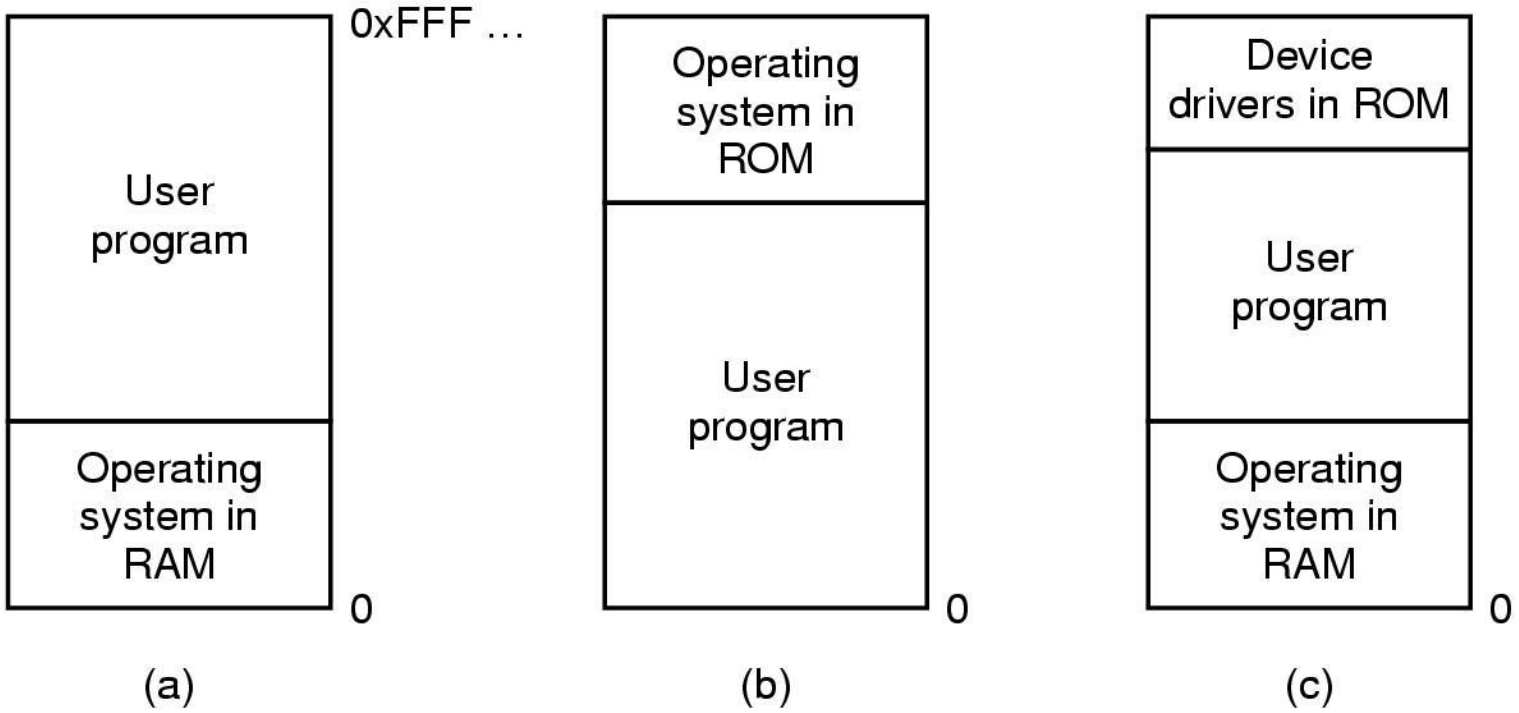
- Hardware device that maps virtual to physical address.
- MMU generates physical address from virtual address provided by the program
- In MMU scheme, the value in the relocation register is added to every address generated by a user process at the time it is sent to memory.
- The user program deals with *logical* addresses; it never sees the *real* physical addresses.
- The final address of a referenced memory address is determined at the time of reference (execution time binding).

Dynamic relocation using a relocation register

MMU maps the logical address dynamically (run-time) by adding the value in the relocation register. This mapped address is sent to memory



One program at a time in memory



OS reads program in from disk and it is executed

How to run more programs then fit in main memory at once

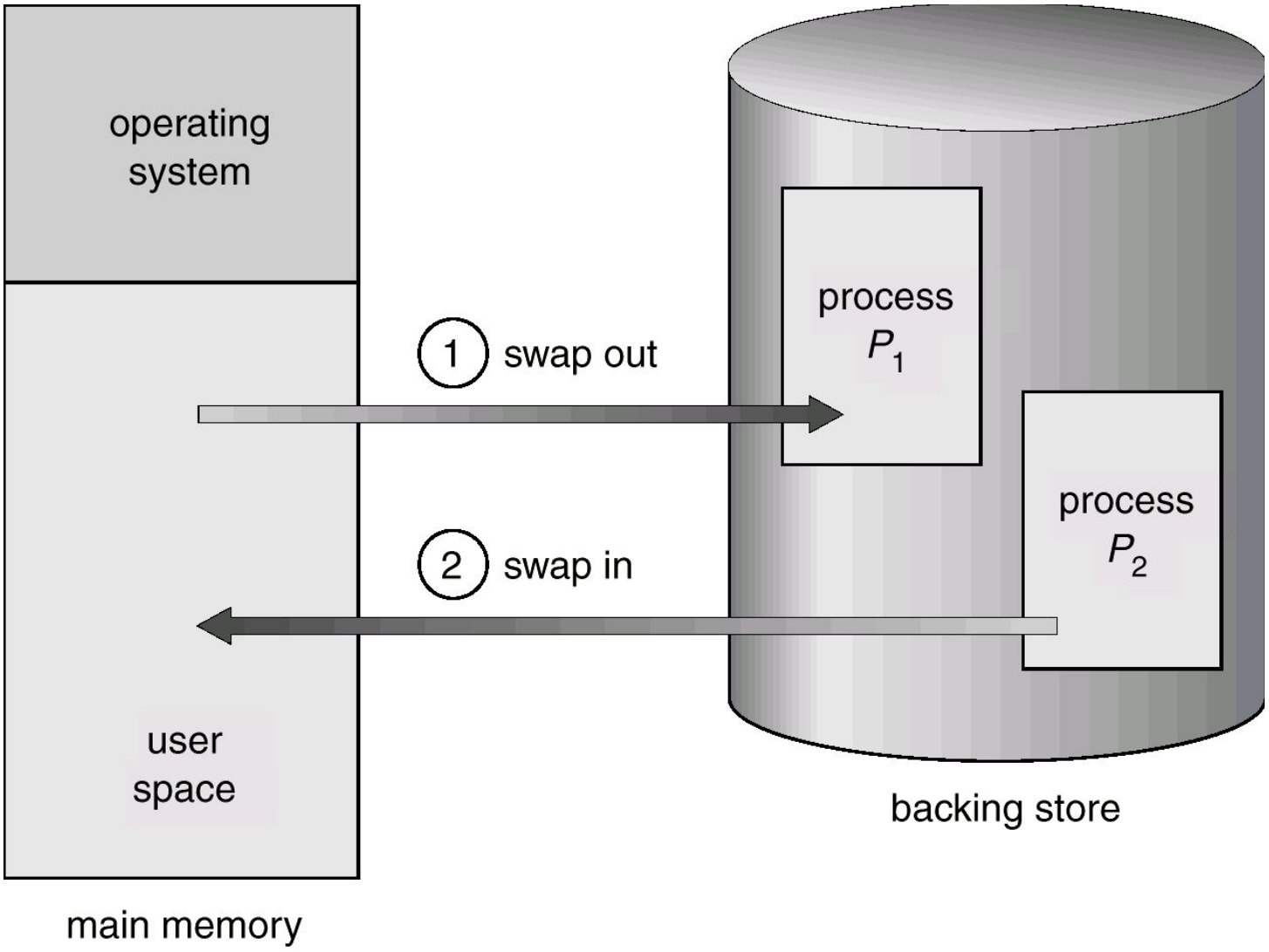
We usually want several user processes to reside in memory at the same time.

- Can't keep all processes in main memory
  - Too many (hundreds)
  - Too big (e.g. 200 MB program)
- Two approaches
  - Swap-bring program in and run it for awhile
  - Virtual memory-allow program to run even if only part of it is in main memory

Swapping

- A process can be *swapped* temporarily out of memory to a *backing store*, and then brought back into memory for continued execution.
- Backing store – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images.
- *Roll out, roll in* – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed.
- Major part of swap time is transfer time; total transfer time is directly proportional to the *amount* of memory swapped.

Schematic View of Swapping



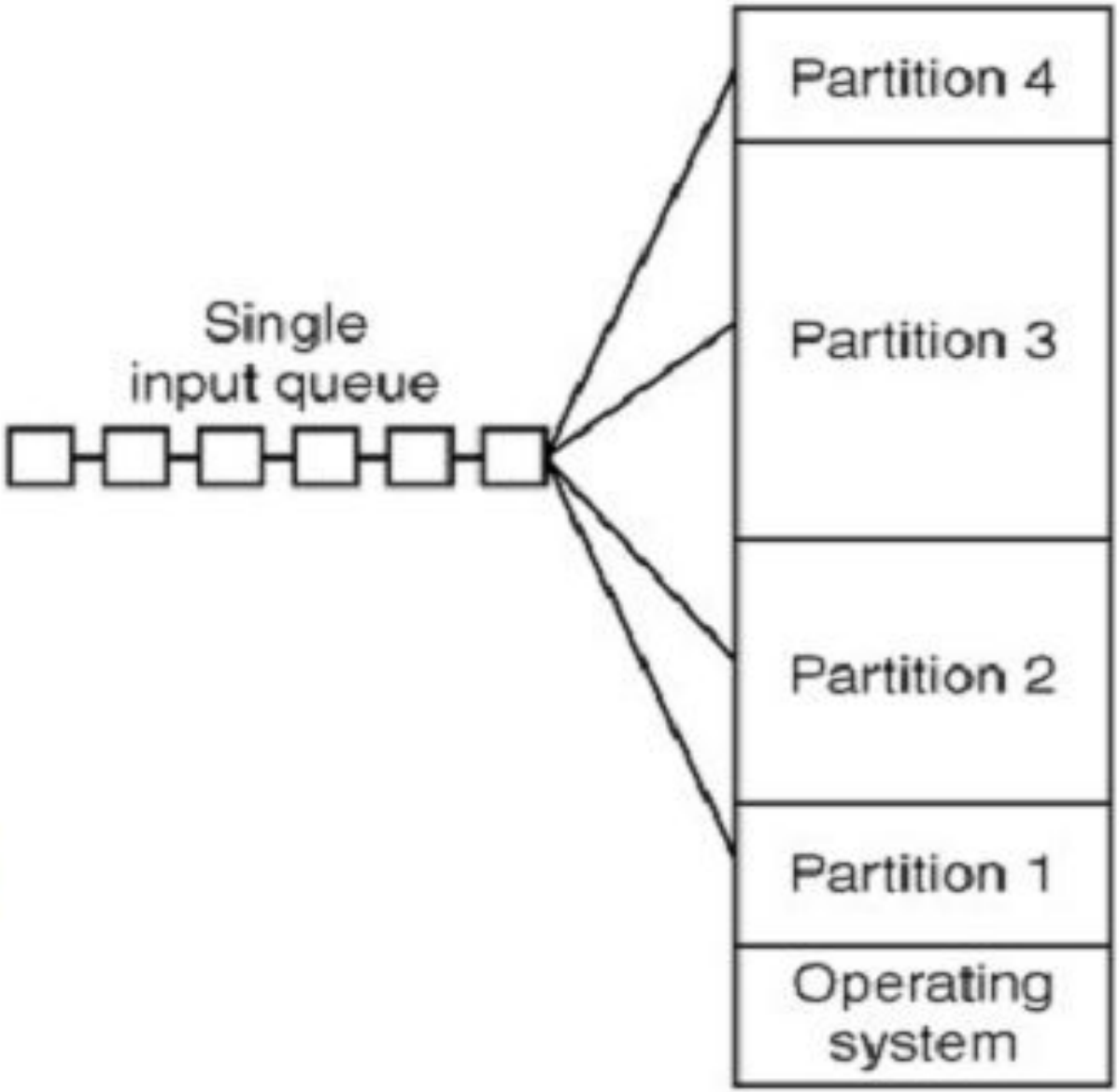
Memory Allocation : Multiple-partition

In contiguous memory allocation, each process is contained in a single section of memory that is contiguous to section containing next process.

1. Fxed-sized partitions

- The simplest methods for allocating memory is to divide memory into several fixed-sized partitions. Each partition may contain exactly one process.
- Thus, the degree of multiprogramming is bound by the number of partitions.
- In multiple partition method, when a partition is free, a process is selected from the input queue and is loaded into the free partition. When the process terminates, the partition becomes available for another process.

Fixed memory partitions with a single input queue



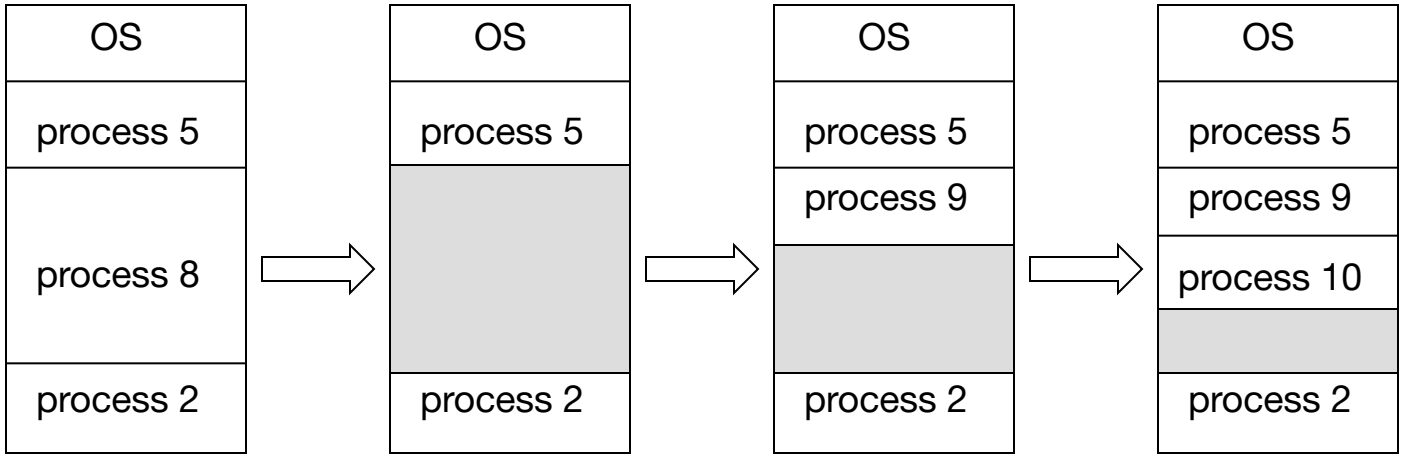
2. Variable-sized partitions

- In the variable-partition scheme, OS keeps a table indicating which parts of memory are available and which are occupied.
- Initially, all memory is available for user processes and is considered one large block of available memory, a hole.
- As processes enter the system, they are put into an input queue.
- OS takes care of the memory requirements of each process and the amount of available memory space in determining which processes are allocated memory.
- When a process terminates, it releases its memory, which the operating system may then fill with another process from the input queue.
- After certain time memory blocks available comprise a set of holes of various sizes scattered throughout memory.
- When a process arrives, the system searches for a hole that is large enough for this process.

Contiguous Allocation (Cont.)

Multiple-partition allocation

- *Hole* – block of available memory; holes of various size are scattered throughout memory.
- When a process arrives, it is allocated memory from a hole large enough to accommodate it.
- Operating system maintains information about:
  a) allocated partitions b) free partitions (hole)



Dynamic Storage-Allocation Problem

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

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

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

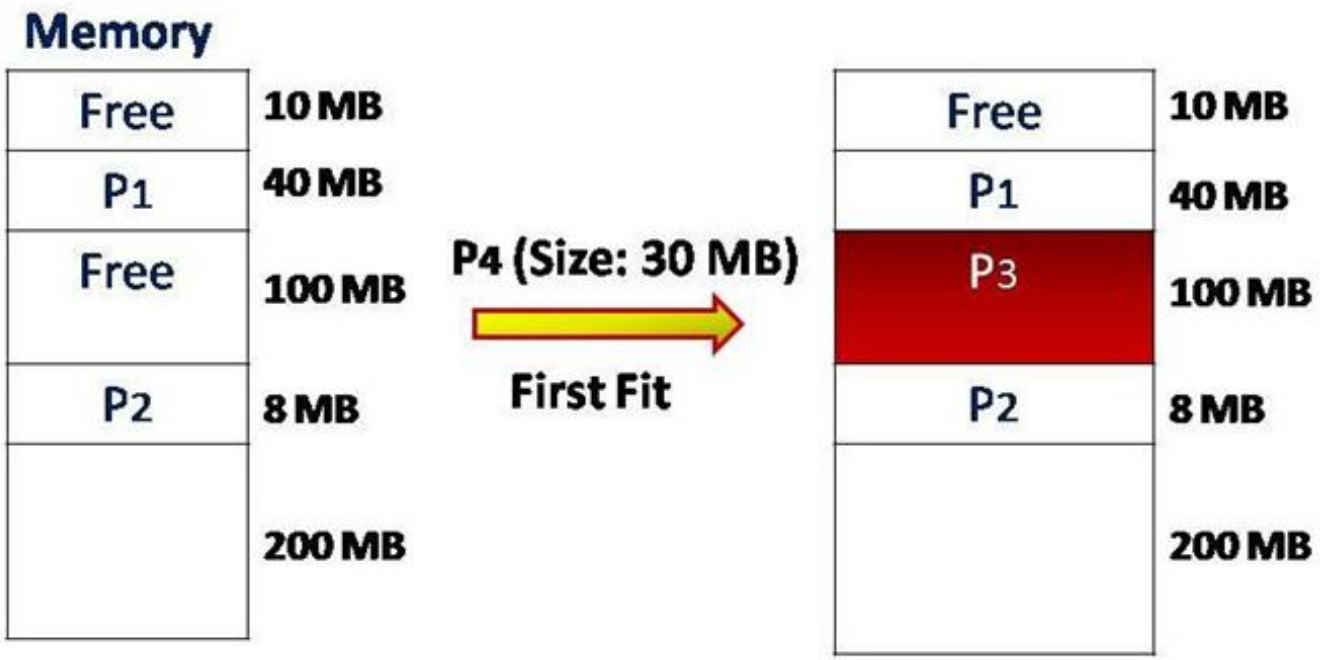
If most of the requests are of similar size, a worst fit policy tends to minimize external fragmentation.

Memory Allocation Policies

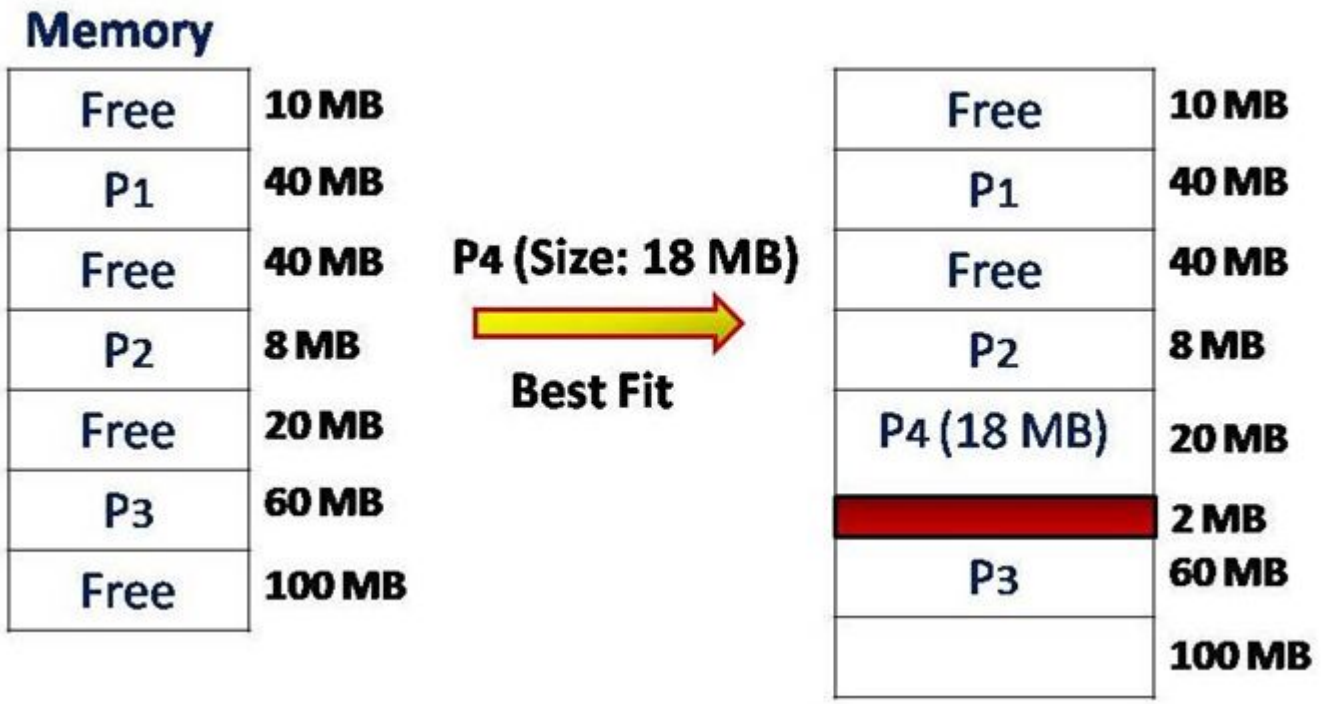
- **First Fit:** first fit allocation is that we begin searching the list and take the first block whose size is greater than or equal to the request size.
- **Best Fit**: This policy allocates the process to the smallest available free block of memory that is closest in size to the request. The best fit may result into a bad fragmentation, but in practice this is not commonly observed.
- **Worst Fit:** This policy allocates the process to the largest available free block of memory. This leads to elimination of all large blocks of memory, thus requests of processes for large memory cannot be met.
-

First fit

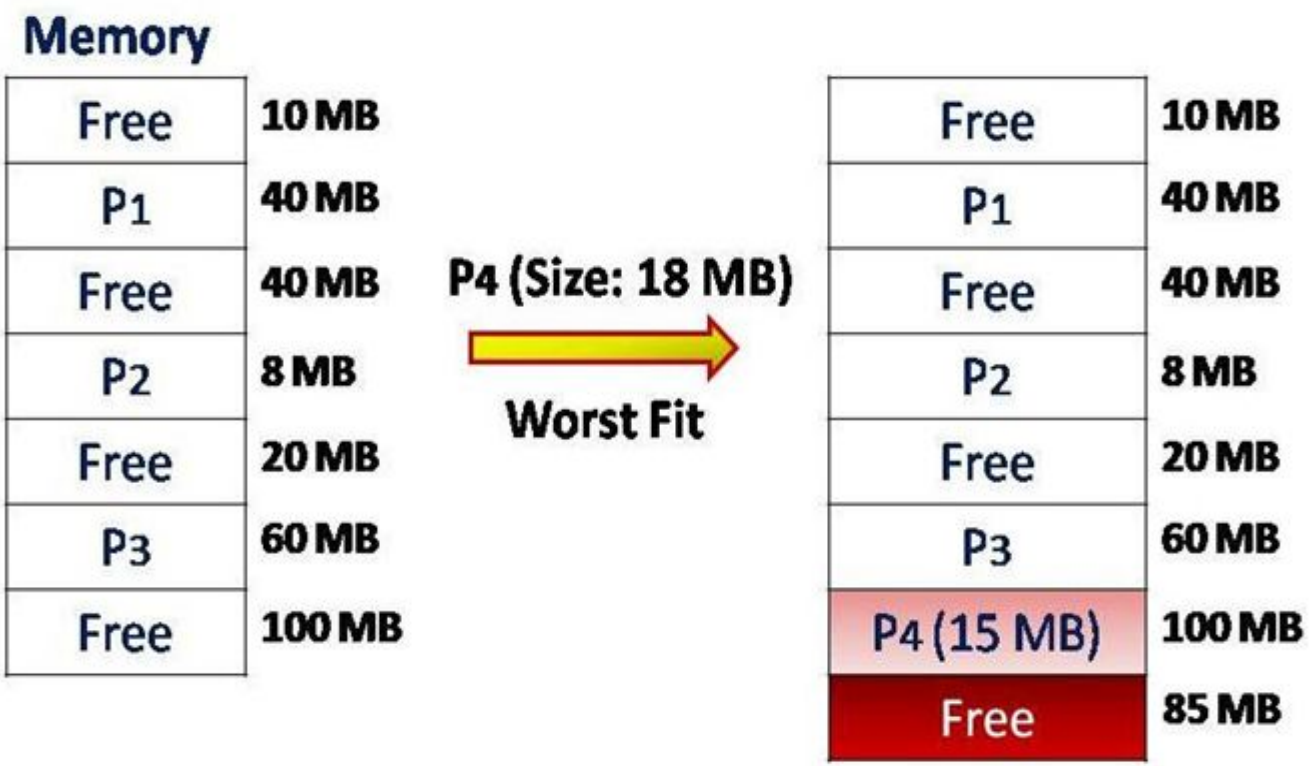
If the P3 size is 30MB then it fit the first location of the memory, first appropriate free location size is 100MB so, it is appropriate for P3



Best fit


Memory
Free
10 MB
P1
40 MB
Free
40 MB
P2
8 MB
Free
20 MB
P3
60 MB
Free
100 MB
P4 (Size: 18 MB)
Best Fit
Free
10 MB
P1
40 MB
Free
40 MB
P2
8 MB
P4 (18 MB)
20 MB
2 MB
P3
60 MB
100 MB

Worst fit


Memory
Free
10 MB
P1
40 MB
Free
40 MB
P2
8 MB
Free
20 MB
P3
60 MB
Free
100 MB
P4 (Size: 18 MB)
Worst Fit
Free
10 MB
P1
40 MB
Free
40 MB
P2
8 MB
Free
20 MB
P3
60 MB
P4 (15 MB)
100 MB
Free
85 MB

Problem

Given six memory partitions of 300 KB, 600 KB, 350 KB, 200 KB, 750 KB, and 125 KB (in order), how would the first-fit, best-fit, and worst-fit algorithms place processes of size 115 KB, 500 KB, 358 KB, 200 KB, and 375 KB (in order)? Rank the algorithms in terms of how efficiently they use memory

Fragmentation

- In contiguous memory allocation whenever the processes come into RAM, space is allocated to them.
- These spaces in RAM are divided either on the basis of fixed partitioning(the size of partitions are fixed before the process gets loaded into RAM) or dynamic partitioning (the size of the partition is decided at the run time according to the size of the process).
- As the process gets loaded and removed from the memory these spaces get broken into small pieces of memory that it can't be allocated to the coming processes. This problem is called fragmentation.

Fragmentation

- As processes are loaded and removed from memory, the free memory space is broken into little pieces.
- For a given set of processes, we can increase the multiprogramming level only by packing more processes into memory. To accomplish this task, we must reduce memory waste, or fragmentation.
- Fragmentation refers to the waste space in memory, which cannot be allocated to any process.

Two types: Internal, external

Fragmentation

- Fragmentation is the inability to reuse memory that is free
- External fragmentation occurs when enough free memory is available but isn't contiguous
  - Many small holes
- Internal fragmentation arises when a large enough block is allocated but it is bigger than needed
  - Blocks are usually split to prevent internal fragmentation

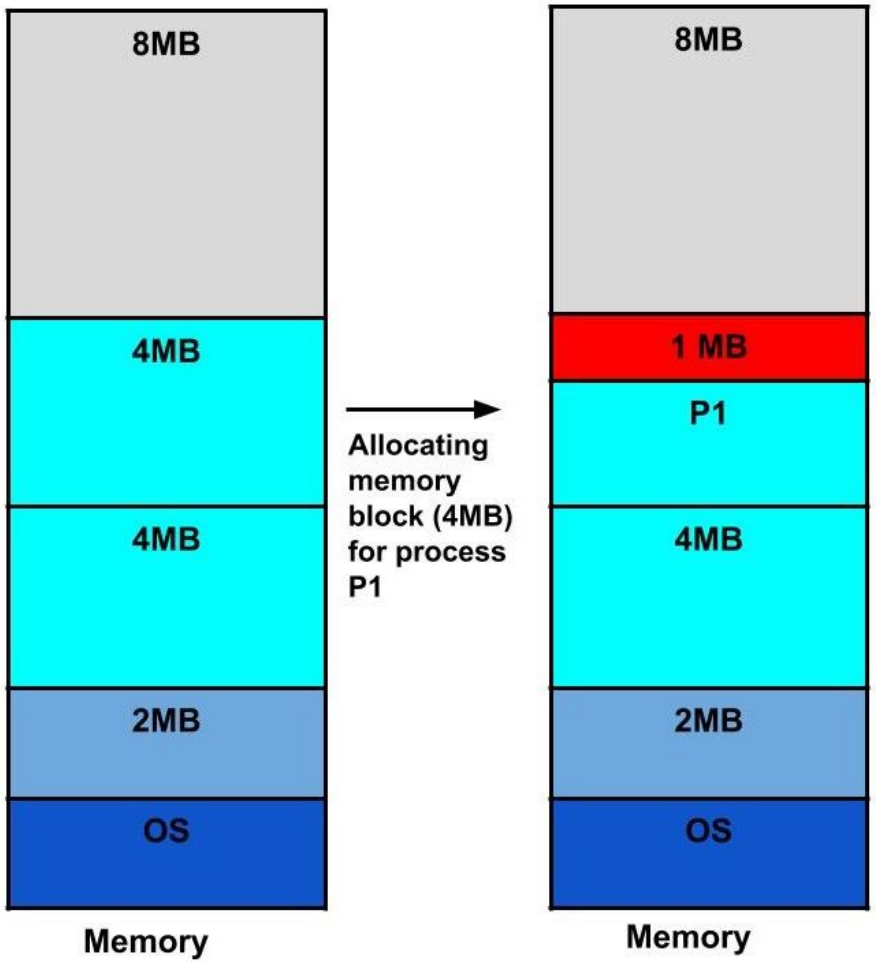
Internal fragmentation

- Internal fragmentation – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used.
- Internal fragmentation—unused memory that is internal to a partition.
- Systems with fixed-sized allocation units, such as the single-partition scheme and paging, suffer from internal fragmentation.

Example

- Example: Suppose there is fixed partitioning (i.e. the memory blocks are of fixed sizes) is used for memory allocation in RAM. These sizes are 2MB, 4MB, 4MB, 8MB. Some part of this RAM is occupied by the Operating System (OS).

- Now, suppose a process P1 of size 3MB comes and it gets memory block of size 4MB. So, the 1MB that is free in this block is wasted and this space can't be utilized for allocating memory to some other process. This is called internal fragmentation.

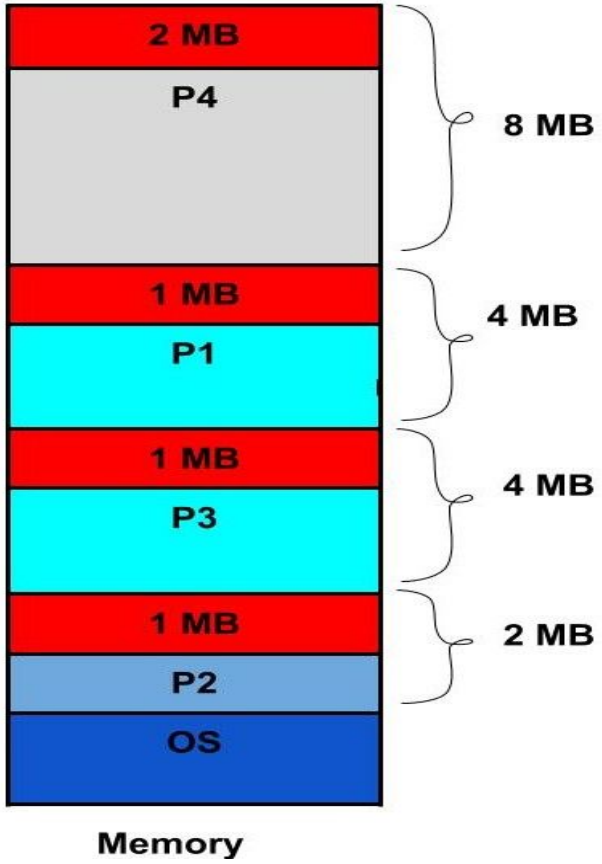

8MB
4MB
4MB
2MB
OS
Memory
Allocating memory block (4MB) for process P1
8MB
1 MB
P1
4MB
2MB
OS
Memory

How to remove internal fragmentation?

- This problem is occurring because we have fixed the sizes of the memory blocks.
- This problem can be removed if we use dynamic partitioning for allocating space to the process.
- In dynamic partitioning, the process is allocated only that much amount of space which is required by the process. So, there is no internal fragmentation.

External Fragmentation

- Total memory space exists to satisfy a request, but it is not contiguous. storage is fragmented into a large number of small holes.
- In the worst case, we could have a block of free (or wasted) memory between every two processes. If all these small pieces of memory were in one big free block instead, we might be able to run several more processes.
- Both the first-fit and best-fit strategies for memory allocation suffer from external fragmentation


2 MB
P4
8 MB
1 MB
P1
4 MB
1 MB
P3
4 MB
1 MB
P2
2 MB
OS
Memory

How to remove external fragmentation?

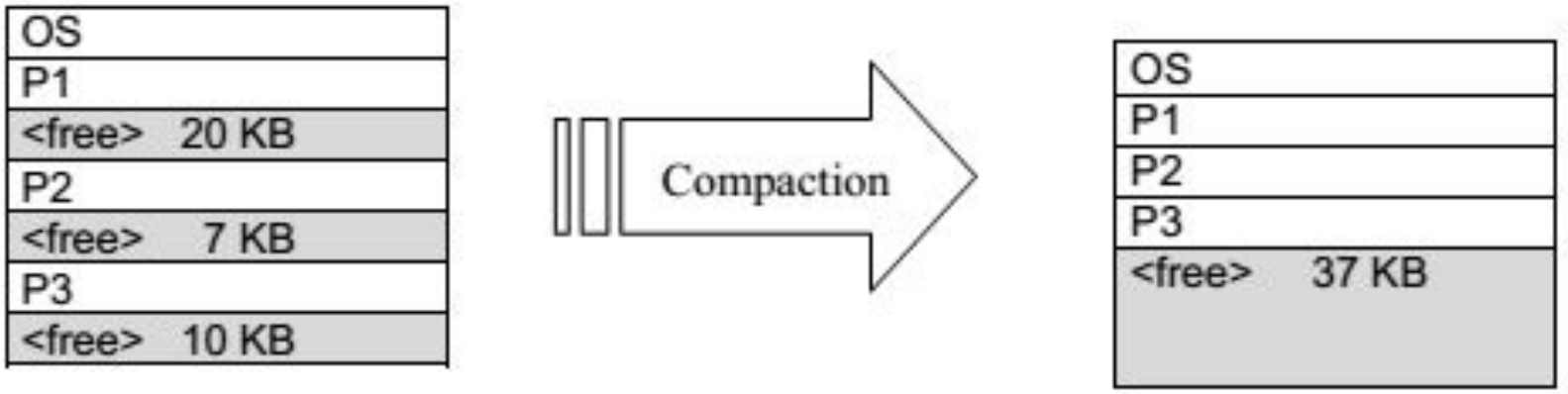
- This problem is occurring because we are allocating memory continuously to the processes. So, if we remove this condition external fragmentation can be reduced. This is what done in **paging** & **segmentation**(non-contiguous memory allocation techniques) where memory is allocated non-contiguously to the processes

- Another way to remove external fragmentation is **compaction**. When dynamic partitioning is used for memory allocation then external fragmentation can be reduced by merging all the free memory together in one large block. This technique is also called defragmentation. This larger block of memory is then used for allocating space according to the needs of the new processes.

Compaction

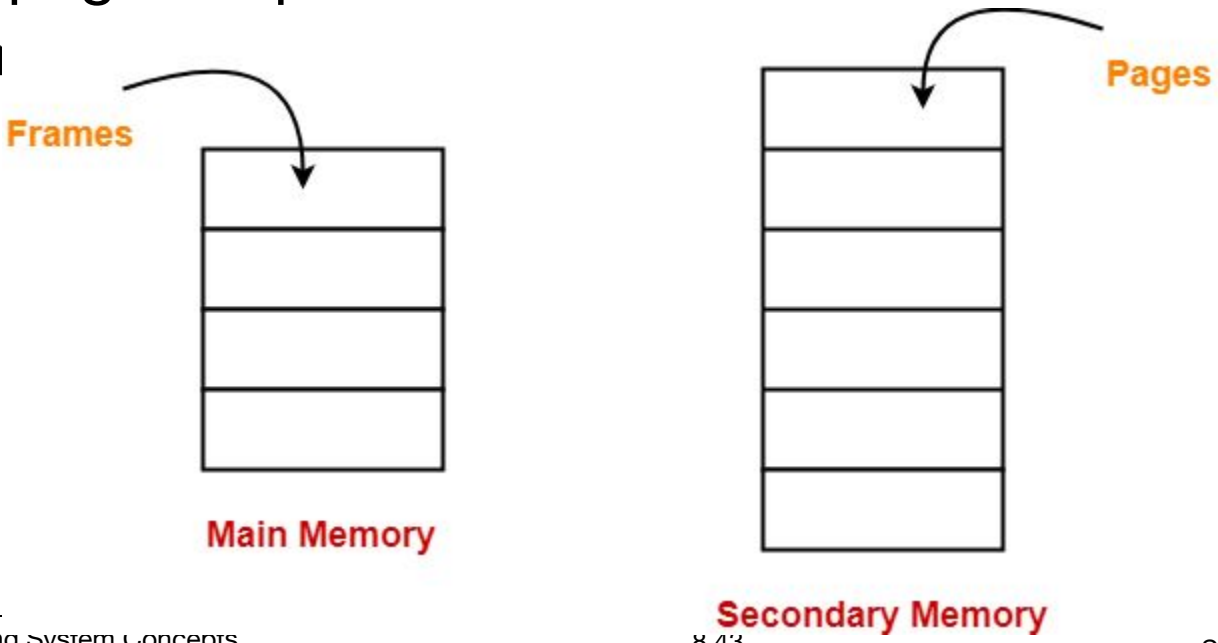
- Reduce external fragmentation by compaction
  - Shuffle memory contents to place all free memory together in one large block.
  - Compaction is possible *only* if relocation is dynamic, and is done at execution time.
  - If addresses are relocated dynamically, relocation requires only moving the program and data and then changing the base register to reflect the new base address.
  - The simplest compaction algorithm is to move all processes toward one end of memory; all holes move in the other direction, producing one large hole of available memory. This scheme can be expensive.
- Another possible solution is to permit the logical address space of the processes to be noncontiguous, thus allowing a process to be allocated physical memory wherever available.
- Two complementary techniques to this solution: segmentation and paging

Memory Compaction Example



Paging

- Paging is a fixed size partitioning scheme.
- Divide physical memory into fixed-sized blocks called frames
- Divide logical memory into blocks of same size called pages.
- The pages of process are stored in the frames of main men

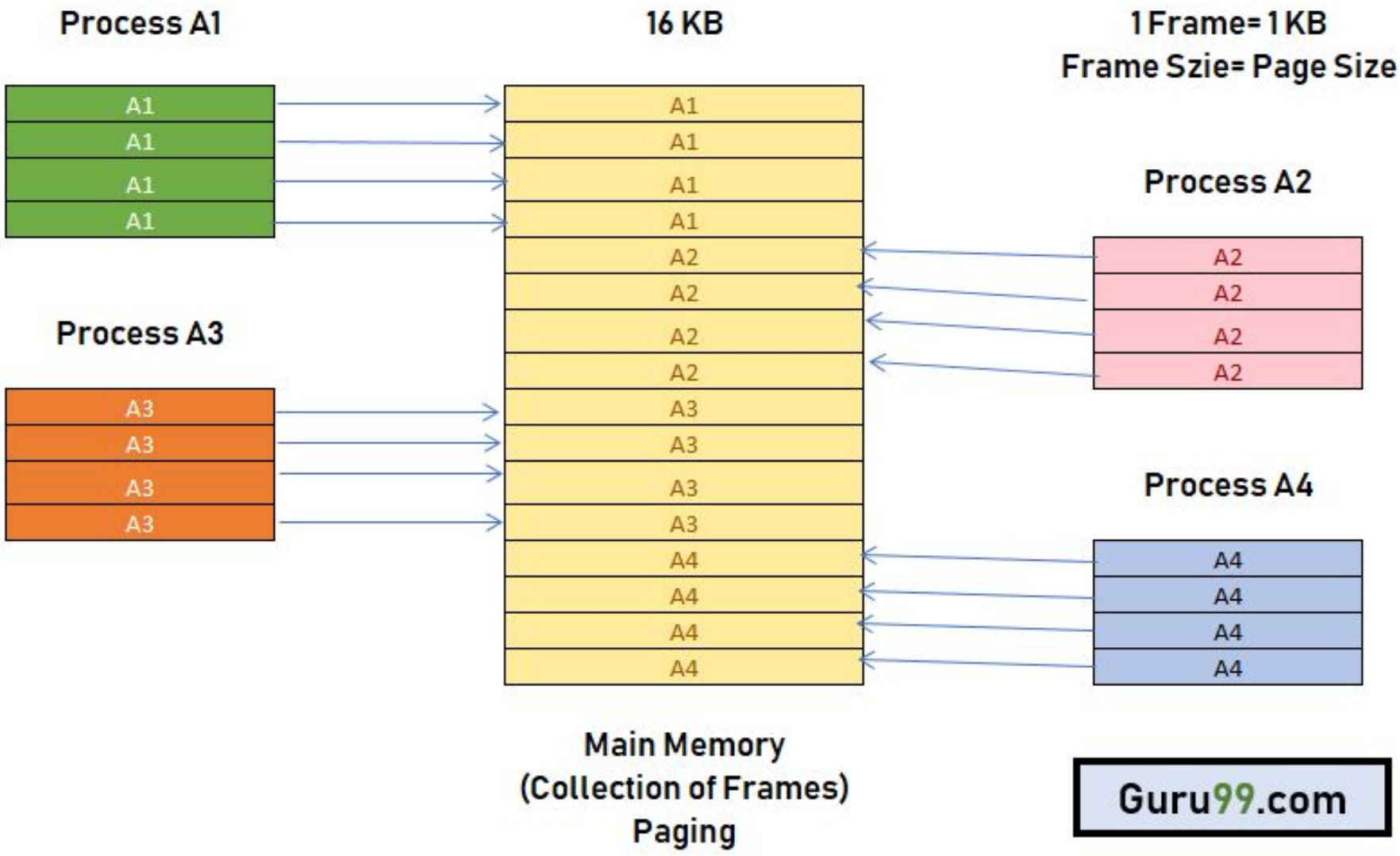


- Each process is divided into parts where size of each part is same as page size.
- The size of the last part may be less than the page size.
- Depending upon the availability, these pages may be stored in the main memory frames in a non-contiguous fashion.

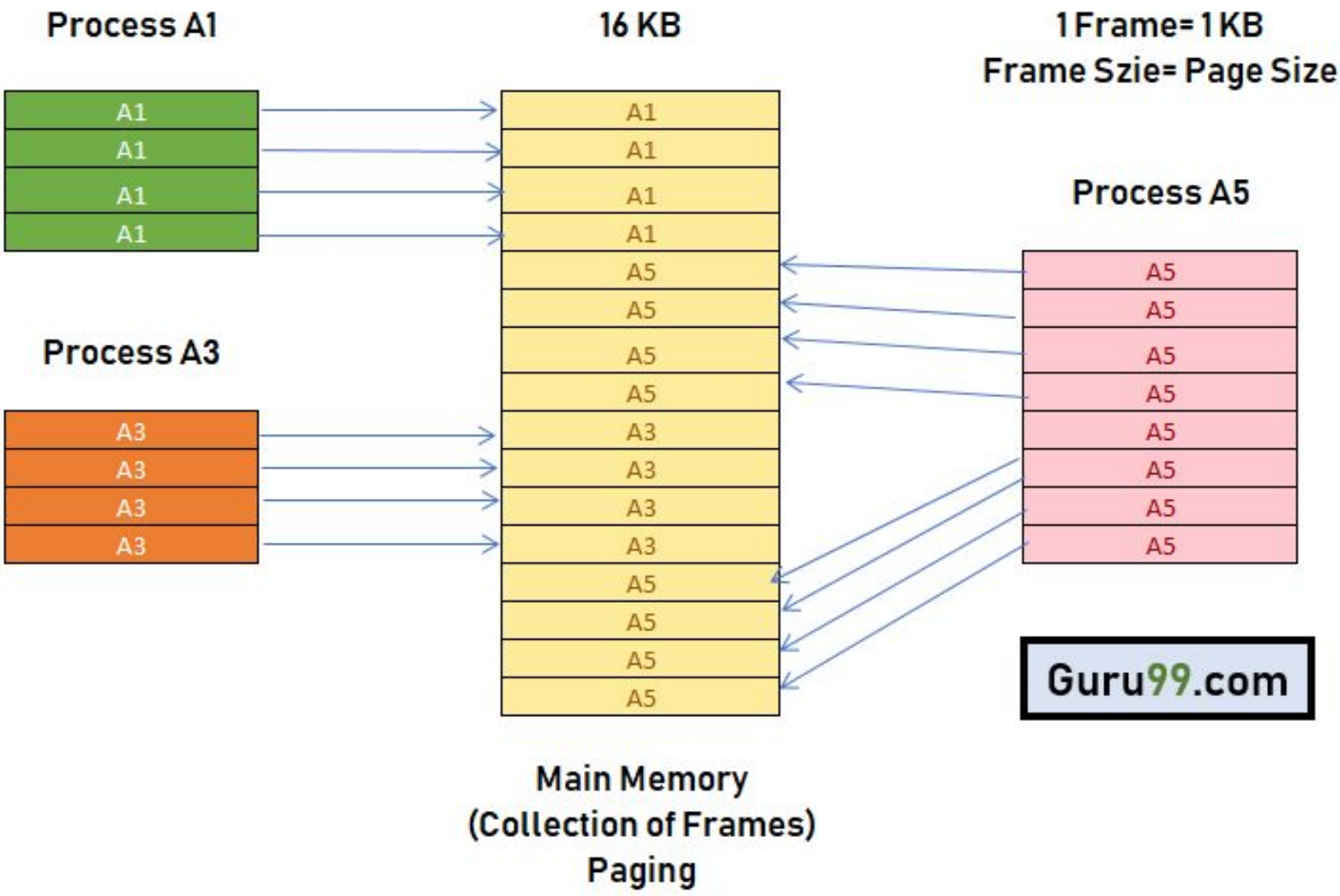
Advantages

- Paging avoids external fragmentation whereas segmentation does not.
- It allows to store parts of a single process in a non-contiguous fashion.

Paging Example



Paging Example (contd....)

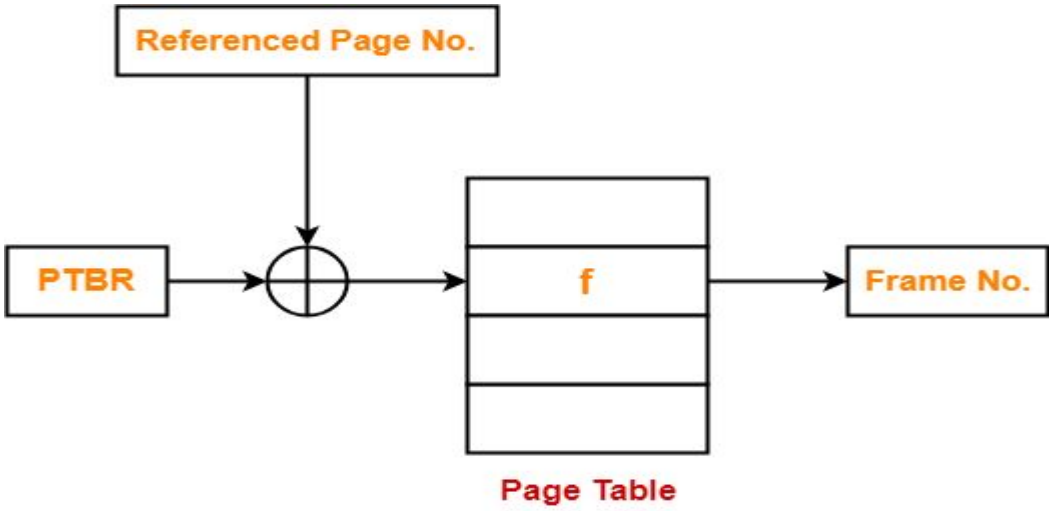


Page Table

A **page table** is the data structure used by a virtual memory system in a computer operating system to store the mapping between virtual addresses and physical addresses.

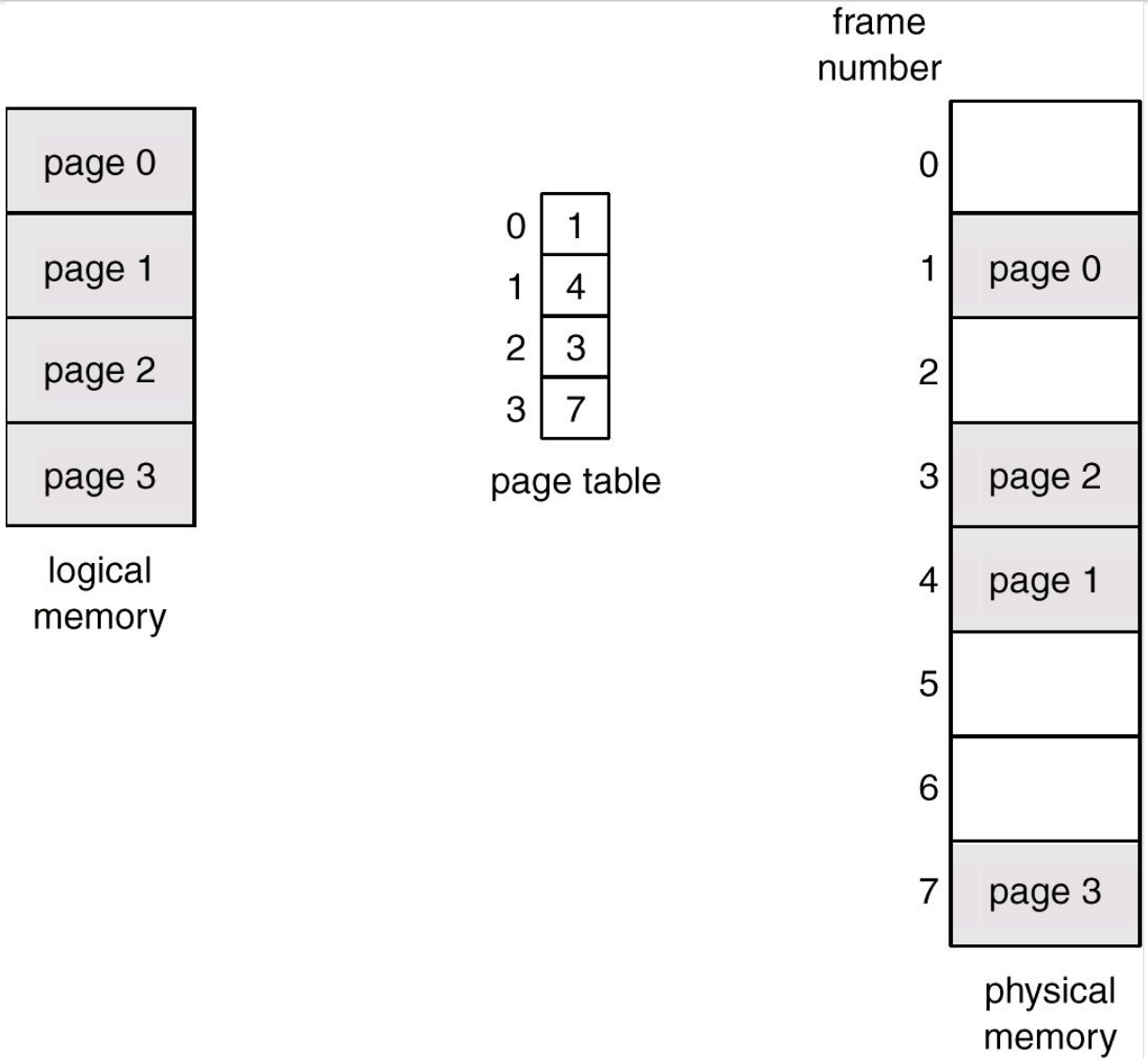
Number of entries in a page table = Number of pages in which the process is divided.

Each process has its own independent page table.



Obtaining Frame Number Using Page Table

Paging Example

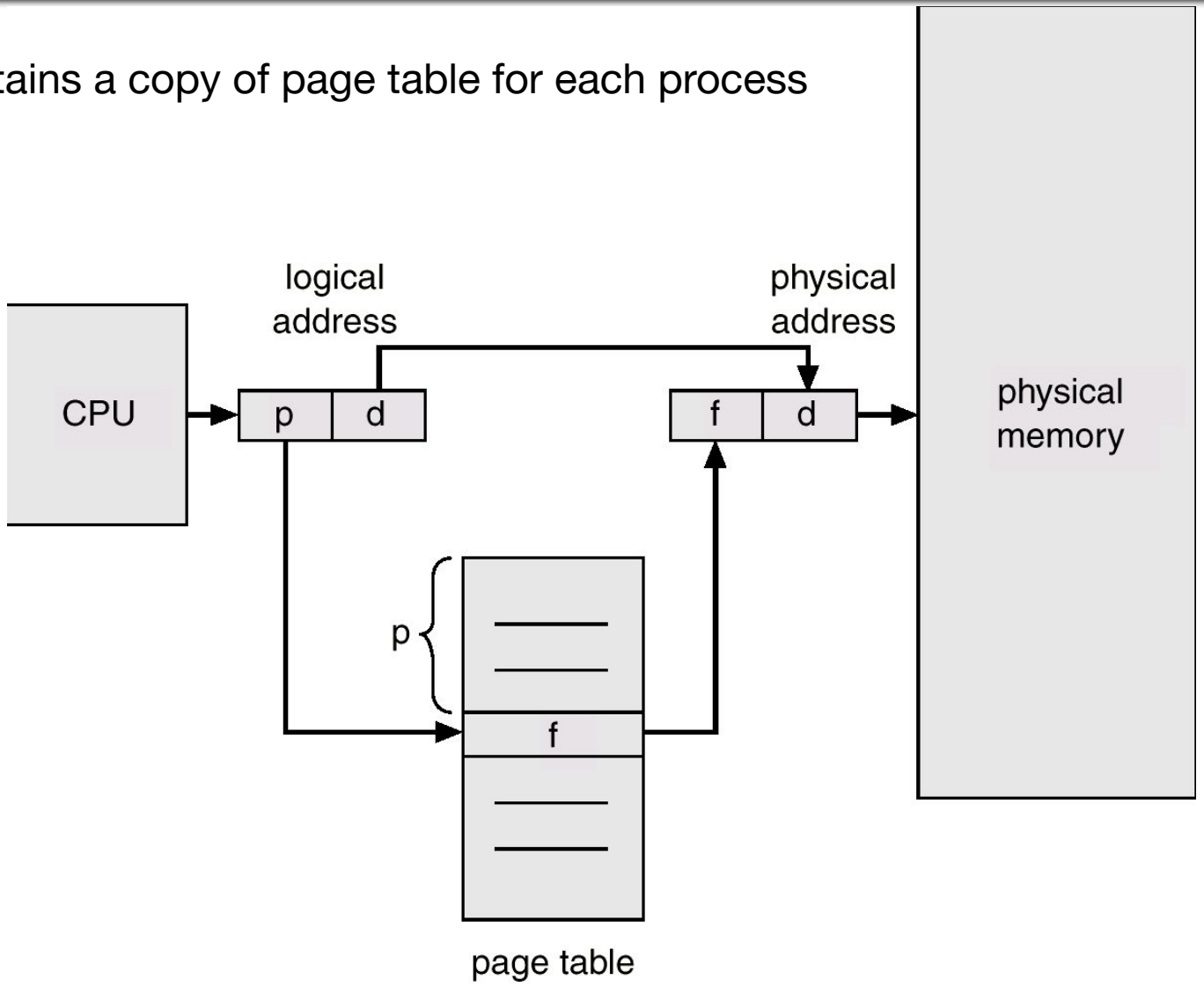


Address Translation Scheme

- Address generated by CPU is divided into:
  - *Page number (p)* – used as an index into a *page table* which contains base address of each page in physical memory.
  - *Page offset (d)* – specifies the specific word on the page that CPU wants to read.
- Page Table provides the corresponding frame number (base address of the frame) where that page is stored in the main memory.
- Page offset combined with base address to define the physical memory address that is sent to the memory unit.
- Average Internal fragmentation in paging is one-half page

Address Translation Architecture

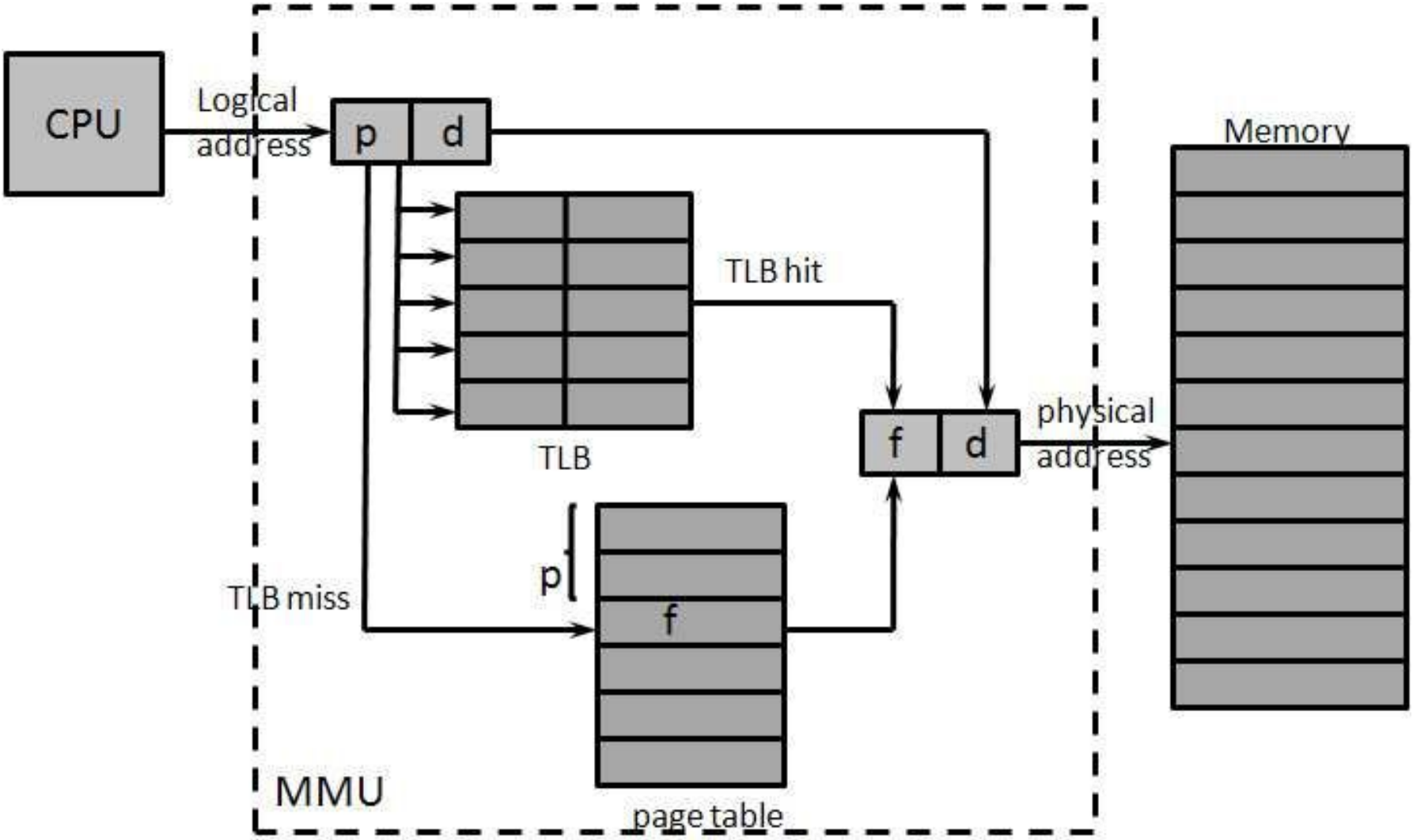
OS maintains a copy of page table for each process



Translation lookaside buffer or TLB.

- Page table information is used by the MMU for every read and write access, so ideally page table should be situated within the MMU.
- But due to its large size, it is impossible to include a complete page table on the processor chip. Therefore the page table is kept in the main memory.
- A copy of the small portion of the most recently used page table entries are cached in MMU to optimize address translation. This cache is commonly called a ***translation lookaside buffer*** or TLB.

Address translation using TLB

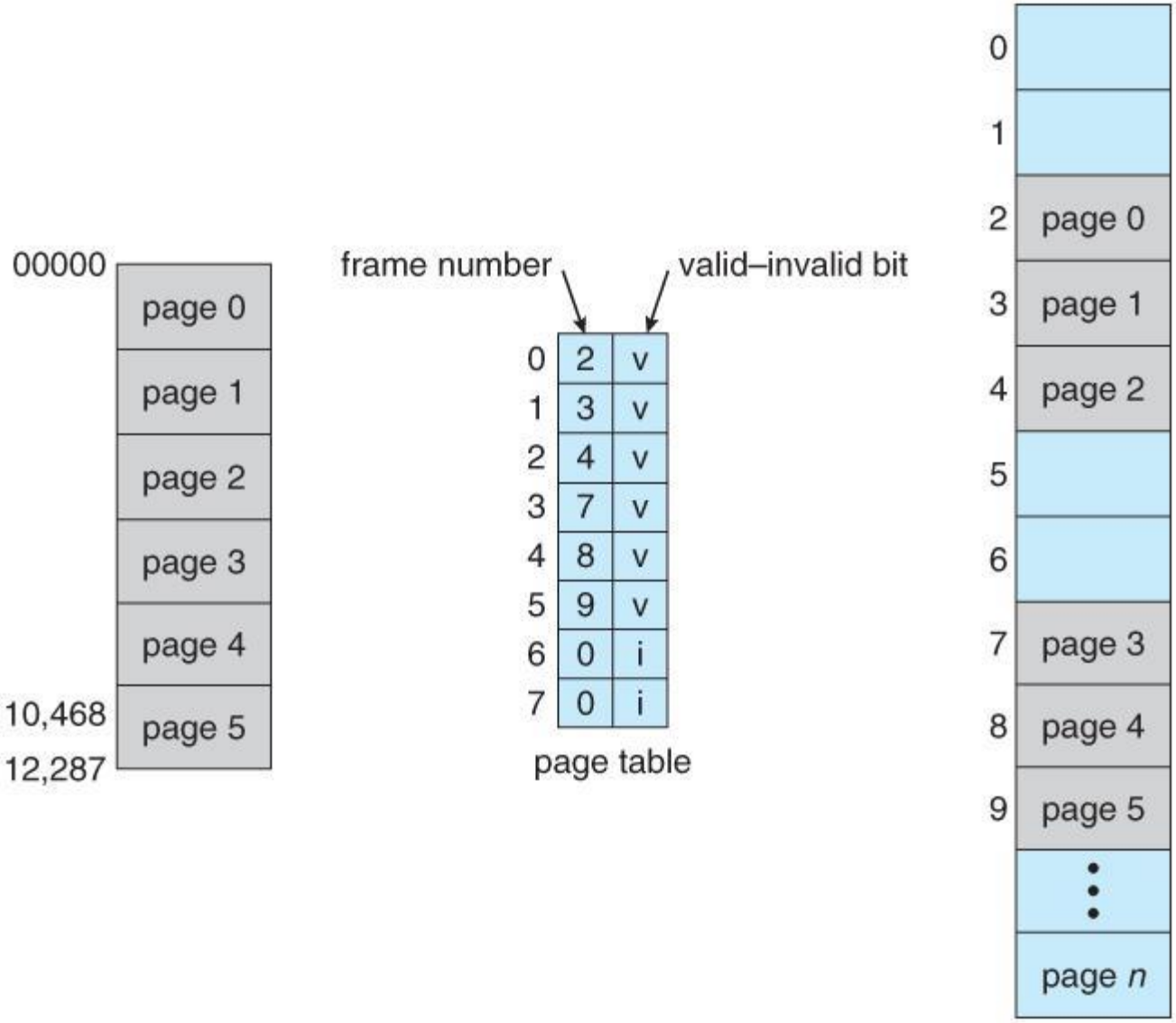


Address translation procedure using TLB

1. Given a virtual address, the MMU looks in the TLB, the physical address is obtained immediately.
2. If there is a miss in the TLB, then the required entry is obtained from the page table in the main memory and the TLB is updated.
3. If requested page is not in main memory, a page fault occurred. The whole page must be brought from disk into the memory before access can proceed.
4. The OS then copies the requested page from the disk into the main memory.
5. Because a long delay occurs while the page transfer takes place, OS may suspend execution of the task that caused page fault and begin execution of another task whose pages are in main memory.

Memory Protection

- Memory protection implemented by associating protection bit with each frame.
- *Valid-invalid* bit attached to each entry in the page table:
  - "valid" indicates that the associated page is in the process' logical address space, and is thus a legal page.
  - "invalid" indicates that the page is not in the process' logical address space.



The disadvantages of paging are-

- It suffers from internal fragmentation.
- There is an overhead of maintaining a page table for each process.
- The time taken to fetch the instruction increases since now two memory accesses are required.

Logical Address in paging scheme

- If the size of the logical address space is 2^m, and a page size is 2^n bytes, then the high-order ***m–n*** bits of a logical address designate the page number, and the ***n*** low-order bits designate the page offset. Thus, the logical address is:

Example

1. Consider a logical address space of 32 pages with 1,024 words per page, mapped onto a physical memory of 16 frames.

Calculate:

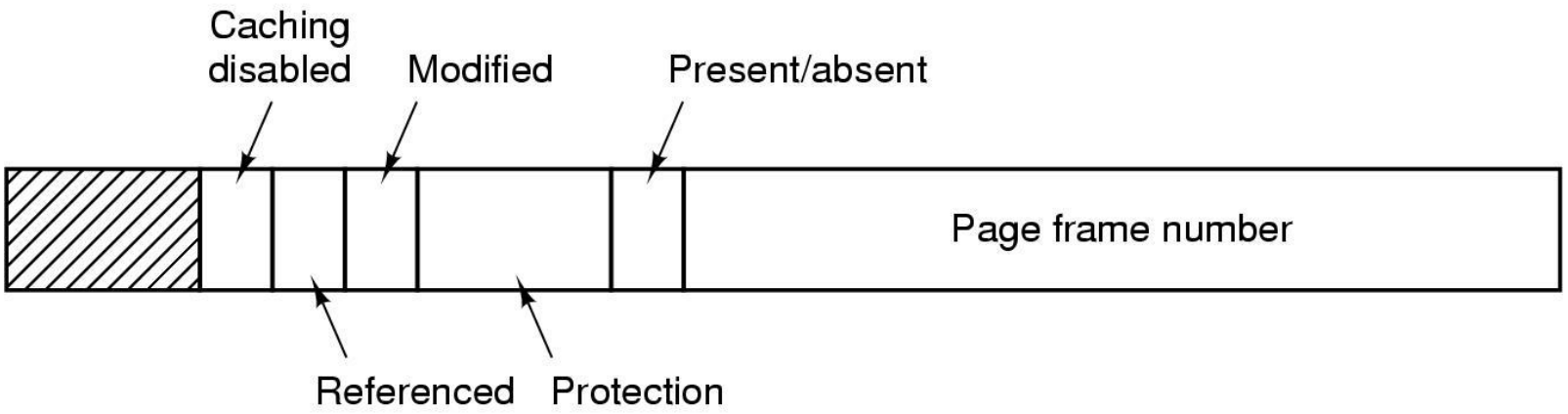
i) How many bits are required in the logical address?

ii) How many bits are required in the physical address?

2. Assuming a 1-KB page size, what are the page numbers and offsets for the following address references (provided as decimal numbers):

a) 3085 b) 42095 c) 215201 d) 650000 e) 2000001

Structure of Page Table Entry



- Modified (dirty) bit: 1 means written to => have to write it to disk.
  0 means don't have to write to disk.
- Referenced bit: 1 means it was either read or written. Used to pick page to evict. Don't want to get rid of page which is being used.
- Present (1) / Absent (0) bit
- Protection bits: r, w, r/w

.

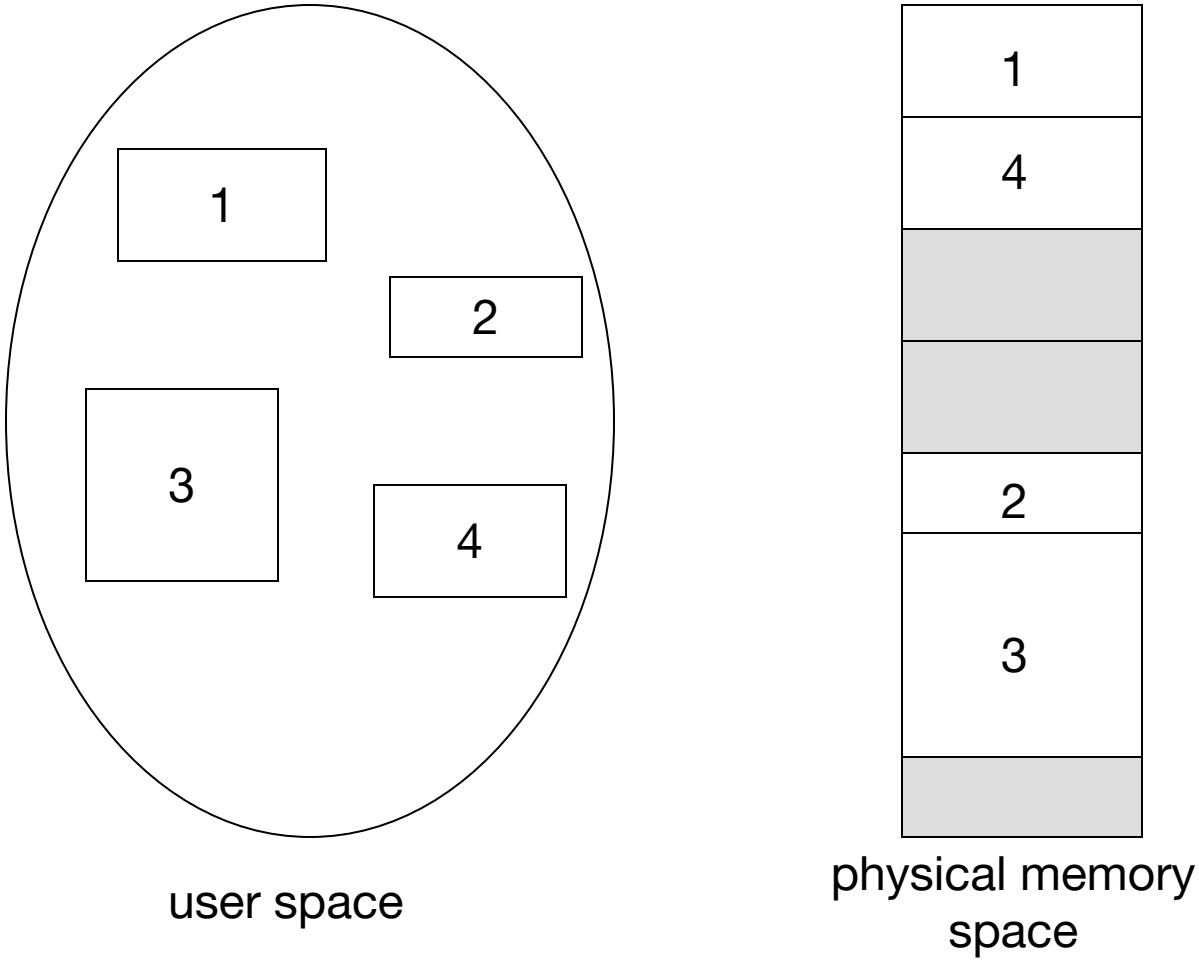
Implementation of Page Table

- Page table is kept in main memory.
- *Page-table base register (*PTBR) points to the page table.
- *Page-table length register* (PRLR) indicates size of the page table.
- In this scheme every data/instruction access requires two memory accesses. One for the page table and one for the data/instruction.
- The two memory access problem can be solved by the use of a special fast-lookup hardware cache called *associative registers* or *translation look-aside buffers (TLBs)*

Segmentation

- Segmentation is a memory-management scheme that supports this programmer view of memory. A logical address space is a collection of segments.
- Segments vary in length, and length of each is defined by its purpose in the program
- Address mapping is done by a segment table. Each entry in the segment table has a segment base and a segment limit.
- The segment base contains the starting physical address where the segment resides in memory, and the segment limit specifies the length of the segment.
- Logical address consists of two parts: segment number, s, offset, d.
- The segment number is used as an index to the segment table. The offset d of the logical address must be between 0 and the segment limit.
- If it is not, trap to OS (logical addressing attempt beyond end of segment).
- When an offset is legal, it is added to the segment base to produce the address in physical memory of the desired byte.

Logical View of Segmentation



Memory segmentation

- Memory **segmentation** is the division of a **computer**'s primary memory into **segments** or sections.
- Memory addresses consist of a segment id and an offset within the segment.
- A hardware memory management unit (MMU) is responsible for translating the segment and offset into a physical address, and for performing checks to make sure the translation can be done and that the reference to that segment and offset is permitted.
- Each segment has a length and set of permissions (for example, *read*, *write*, *execute*) associated with it. A process is only allowed to make a reference into a segment if the type of reference is allowed by the permissions, and if the offset within the segment is within the range specified by the length of the segment. Otherwise, a hardware exception such as a segmentation fault is raised.

Segmentation Architecture

- Logical address consists of a two tuple:

  <segment-number, offset>,

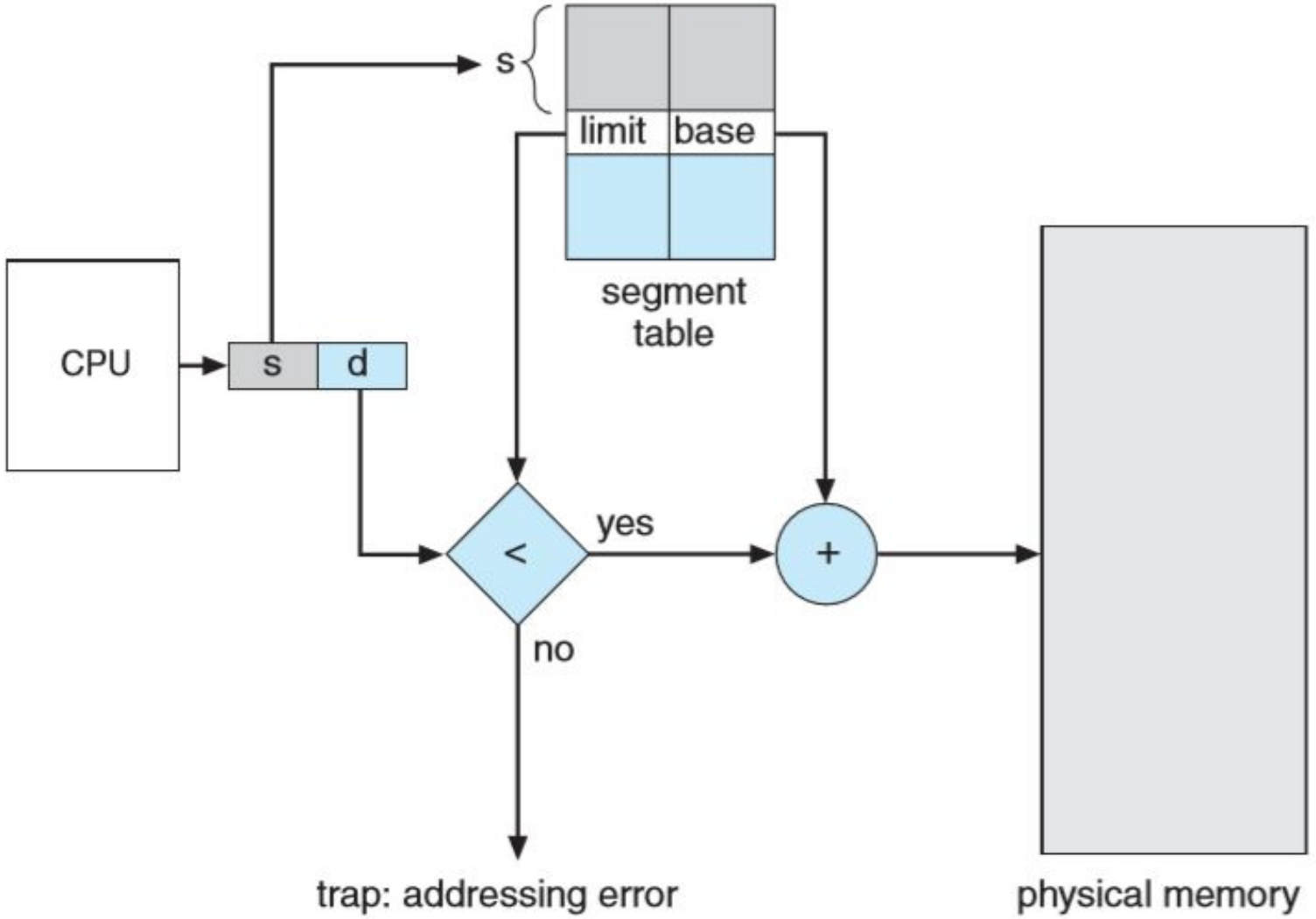
- *Segment table* – maps two-dimensional physical addresses; each table entry has:
  - base – contains the starting physical address where the segments reside in memory.
  - *limit* – specifies the length of the segment.

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

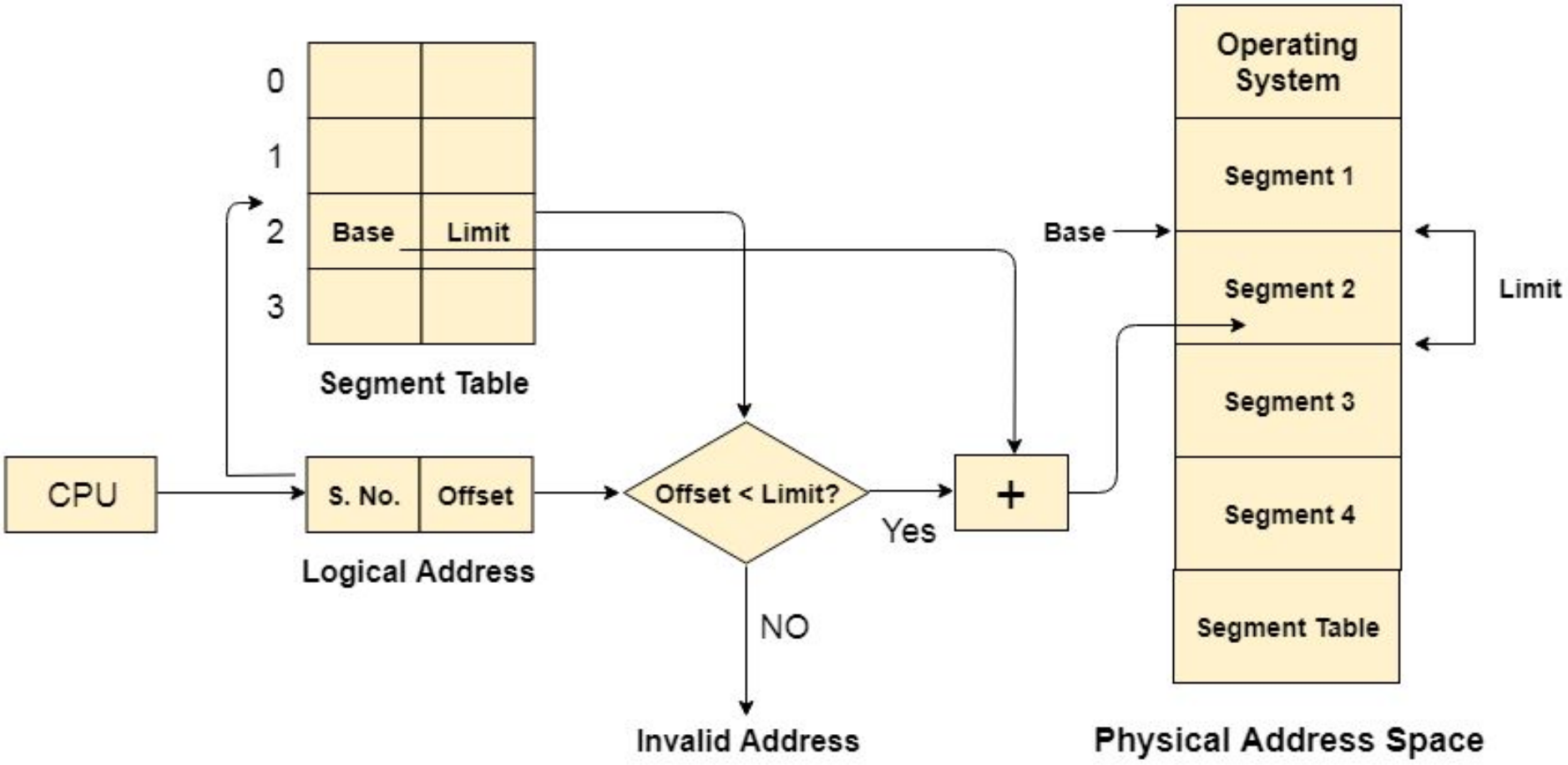
- *Segment-table length register (STLR)* indicates number of segments used by a program;

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

Segmentation hardware



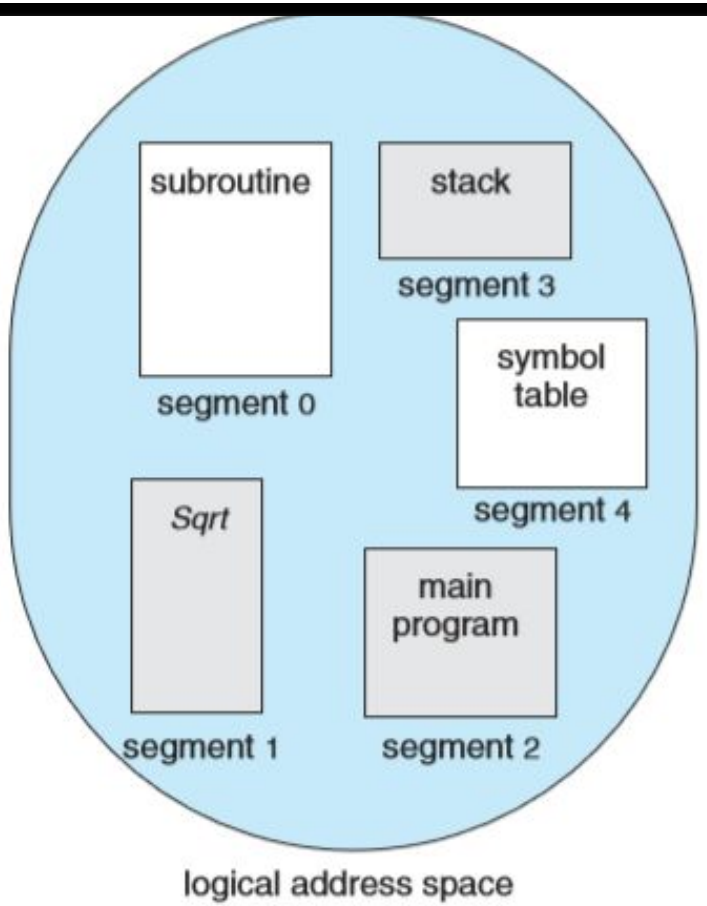
Translation of Logical address into physical address by segment table



Segmentation Architecture (Cont.)

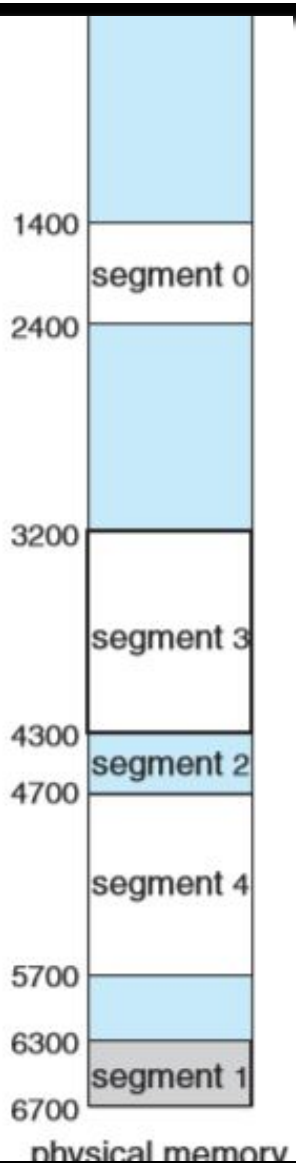
- Protection. With each entry in segment table associate:
  - validation bit = 0 ⇒ illegal segment
  - read/write/execute privileges
- Protection bits associated with segments; code sharing occurs at segment level.
- Since segments vary in length, memory allocation is a dynamic storage-allocation problem.
- A segmentation example is shown in the following diagram

Example of segmentation

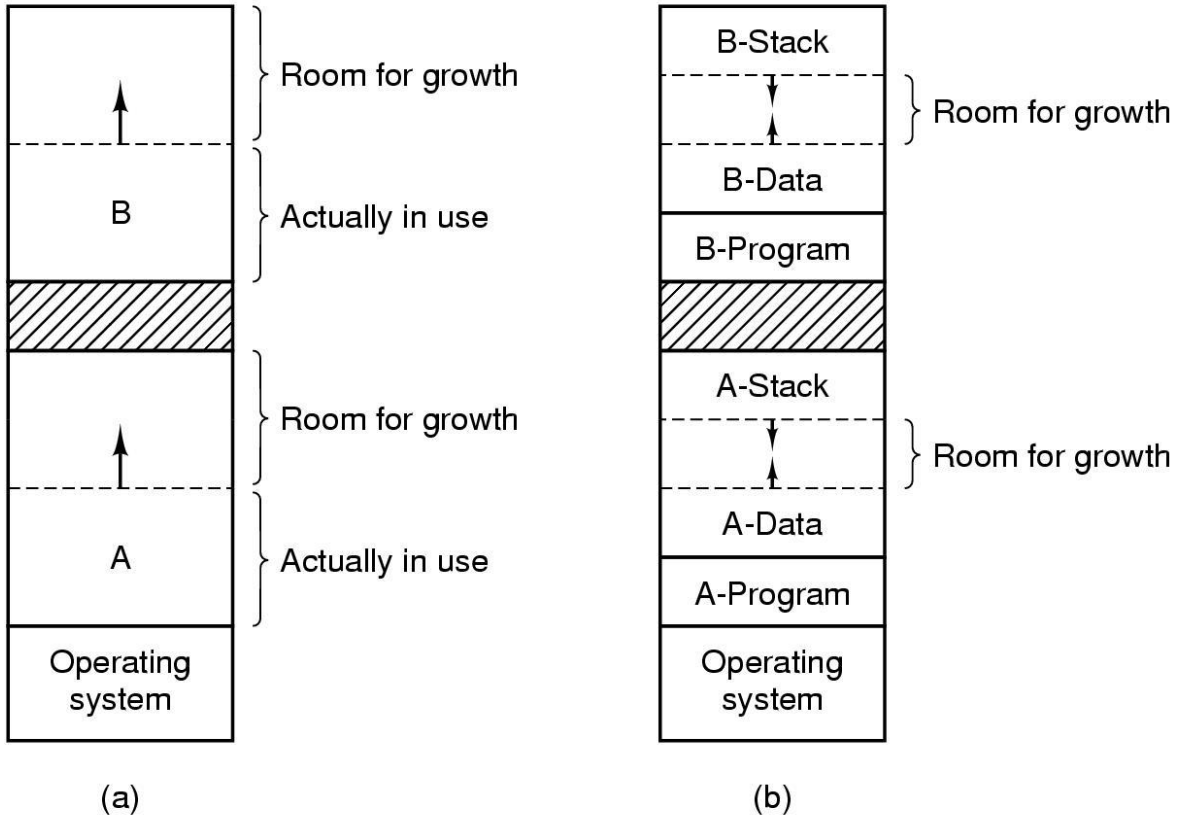


	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

segment table



2 ways to allocate space for growth



(a) Just add extra space

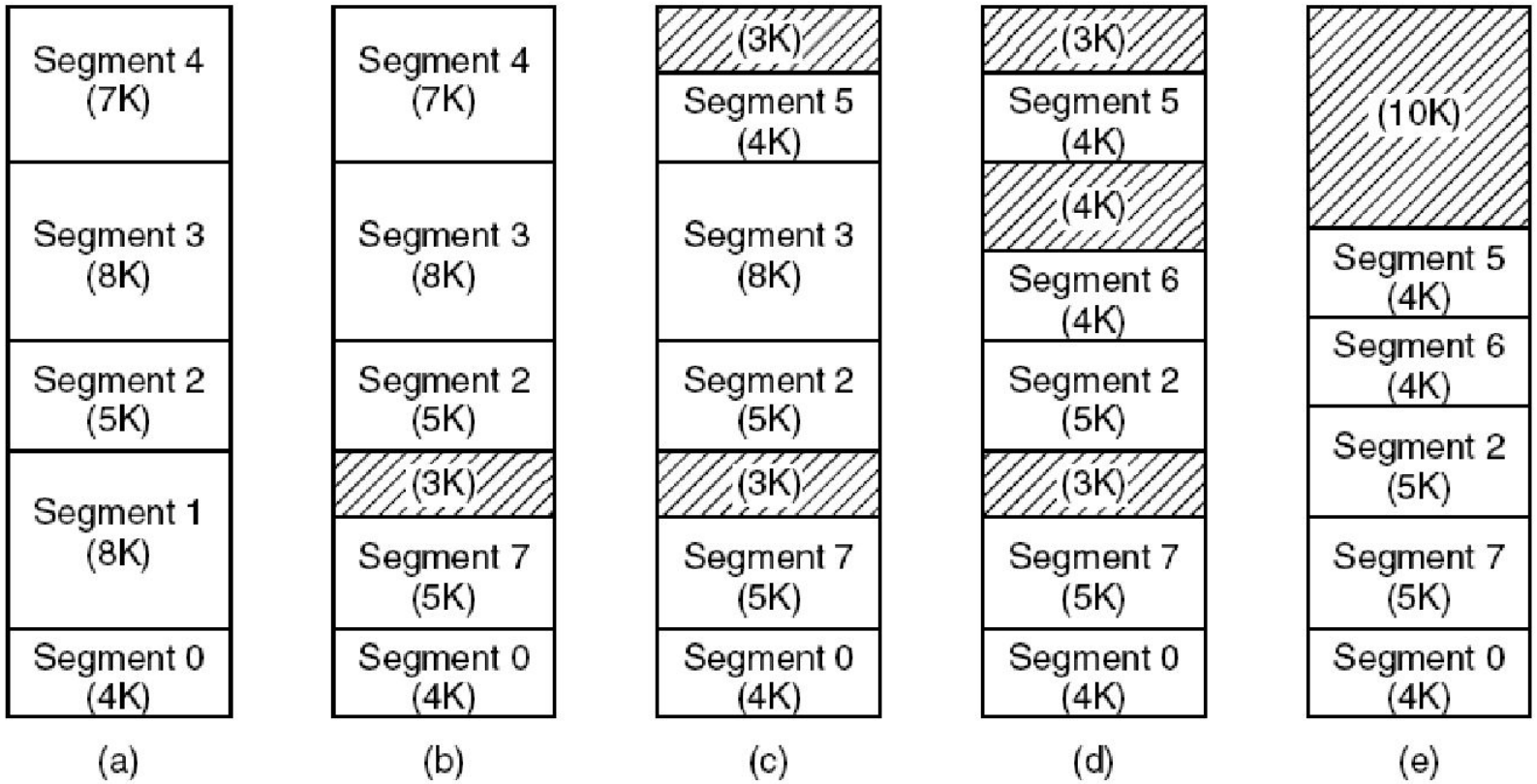
(b) Stack grows downwards, data grows upwards

Logical to physical address translation

Sample Problem

- A reference to byte 53 of segment 2 is mapped onto location 4300 (the base of segment 2) + 53 = 4300 +53=4353.
- A reference to segment 3, byte 852, is mapped to 3200 (the base of segment 3) + 852 = 4052.
- A reference to byte 1222 of segment 0 would result in a trap to OS, as this segment is only 1,000 bytes long.

External fragmentation



(a)-(d) Development of checkerboarding. (e) Removal of the checkerboarding by compaction.

Logical to physical address translation

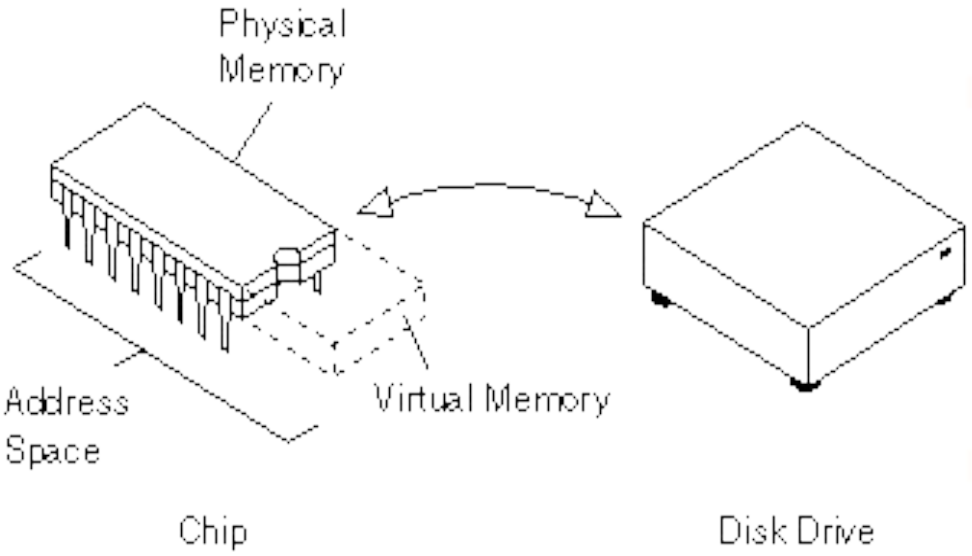
- Consider the following segment table:

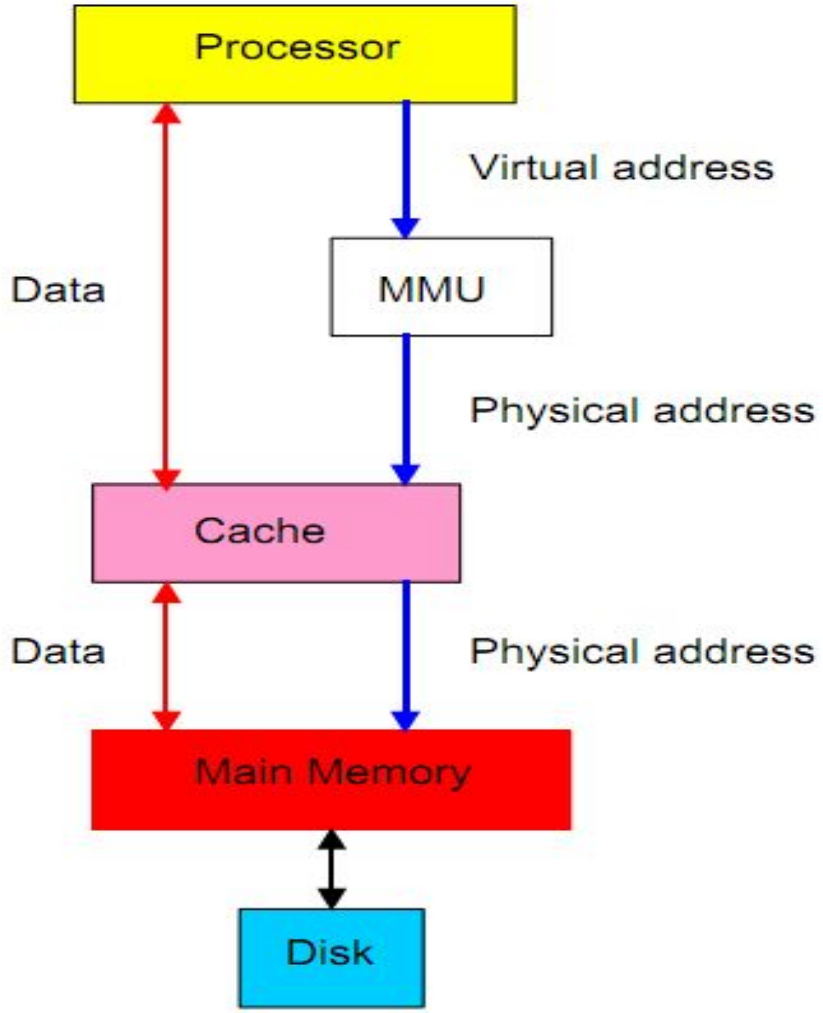
Segment	Base	Length
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

- What are the physical addresses for the following logical addresses?

a) 0,430 b) 1,10 c) 2,500 d) 3,400 e) 4,112

Virtual memory organization





Virtual Memory

- Virtual memory is used to increase the apparent size of physical memory.
- Virtual memory combines computer's RAM with temporary space on your hard disk.
- Virtual memory is a feature of OS that enables a process to use a memory (RAM) address space that is independent of other processes running in the same system, and use a space that is larger than the actual amount of RAM present, temporarily transferring some contents from RAM to a disk, with little or no overhead.
- Virtual memory enables each process to act as if it has whole memory space to itself.
- Data may be stored in physical memory locations that have address different from those specified by the program.

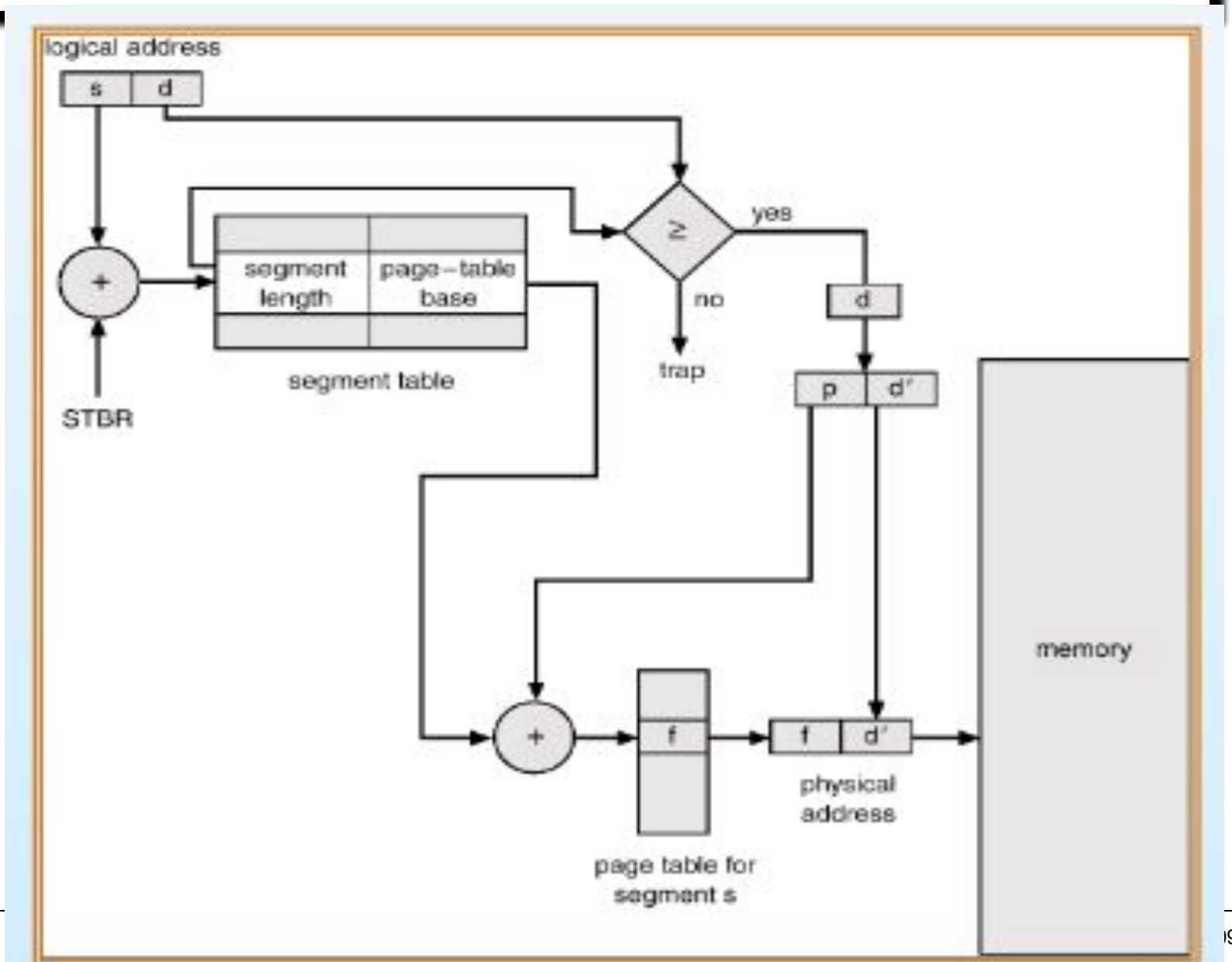
Virtual Memory

- Program's address space is broken up into fixed size pages
- Pages are mapped to physical memory
- If instruction refers to a page in memory, fine
- Otherwise OS gets the page, reads it in, and re-starts the instruction
- While page is being read in, another process gets the CPU

Segmentation with paging

- An implementation of virtual memory on a system using segmentation without paging requires that entire segments be swapped back and forth between main memory and secondary storage. When a segment is swapped in, the operating system has to allocate enough contiguous free memory to hold the entire segment.
- Often memory fragmentation results if there is not enough contiguous memory even though there may be enough in total.
- In Segmented Paging, the main memory is divided into variable size segments which are further divided into fixed size pages.
- Using segmentation with paging usually only moves individual pages back and forth between main memory and secondary storage, similar to a paged non-segmented system.
- Pages of the segment can be located anywhere in main memory and need not be contiguous. This usually results in a reduced amount of input/output between primary and secondary storage and reduced

Segmentation with Paging –Address Translation Scheme



Inverted Page Table

- A single global page table.
- One entry for each real page of memory.
- Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page.
- The physical page number is not stored, since the index in the table corresponds to it
- Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs.
- Use hash table to limit the search to one — or at most a few

Inverted Page Table

- Virtual address in the system consists of a triple:

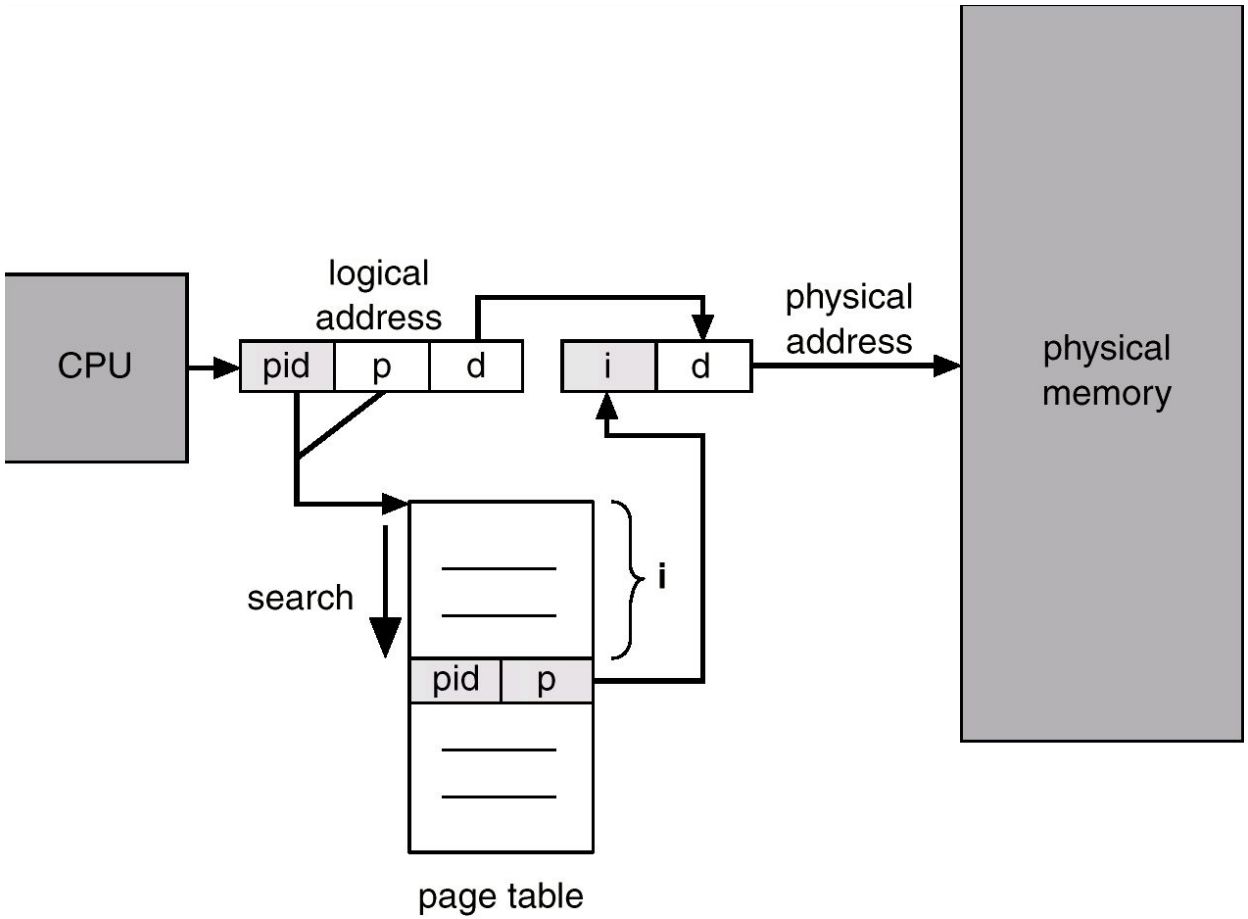
  <process-id, page-number, offset>.

- Each inverted page-table entry is a pair:<process-id, page-number> where the process-id assumes the role of the address-space identifier.

- When a memory reference occurs, part of the virtual address, consisting of <process-id, page number>, is presented to the memory subsystem.

- The inverted page table is then searched for a match. If a match is found—say, at entry i—then the physical address <i, offset> is generated. If no match is found, then an illegal address access has been attempted.

Inverted Page Table Architecture



Demand Paging

- Pages should only be brought into memory if the executing process demands them.
- This is often referred to as lazy evaluation as only those pages demanded by the process are swapped from secondary storage to main memory.
- To achieve this process a page table implementation is used. The page table maps logical memory to physical memory.
- The page table uses a bitwise operator to mark if a page is valid or invalid.
- A valid page is one that currently resides in main memory.
- An invalid page is one that currently resides in secondary memory.

Steps involved in demand paging

When a process tries to access a page, the following steps are generally followed:

- Attempt to access page.
- If page is valid (in memory) then continue processing instruction as normal.
- If page is invalid then a **page-fault trap** occurs.
- Schedule disk operation to read the desired page into main memory.
- Restart the instruction that was interrupted by the operating system trap.

Advantages

- Demand paging, as opposed to loading all pages immediately:
- Only loads pages that are demanded by the executing process.
- As there is more space in main memory, more processes can be loaded, reducing the context switching time, which utilizes large amounts of resources.
- Less loading latency occurs at program startup, as less information is accessed from secondary storage and less information is brought into main memory.

As main memory is expensive compared to secondary memory, this technique helps significantly reduce the bill of material (BOM) cost in smart phones.

Disadvantages

- Individual programs face extra latency when they access a page for first time.
- Low-cost, low-power embedded systems may not have a memory management unit that supports page replacement.
- Memory management with page replacement algorithms becomes slightly more complex.
- Thrashing which may occur due to repeated page faults.

Paging Vs Segmentation

Sno.	Paging	Segmentation
1	Block replacement easy Fixed-length blocks	Block replacement hard Variable-length blocks Need to find contiguous, variable-sized, unused part of main memory
2	Invisible to application programmer	Visible to application programmer.
3	No external fragmentation, But there is Internal Fragmentation unused portion of page.	No Internal Fragmentation, But there is external Fragmentation unused portion of main memory.
4	Units of code and date are broken into separate pages.	Keeps blocks of code or data as a single units.
5	paging is a physical unit invisible to the user's view and is of fixed size	segmentation is a logical unit visible to the user's program and id of arbitrary size
6	Paging maintains one address space.	Segmentation maintains multiple address spaces per process.

Page Replacement Algorithms

Introduction

- When a page fault occurs and no free page frames available, then OS has to choose a page to remove from memory to make room for the page that has to be brought in.
- The page replacement is done by swapping the required pages from backup storage to main memory and vice-versa.
- If the page to be removed has been modified while in memory, it must be rewritten to disk to bring disk copy up to date.
- If, the page has not been changed (e.g., it contains program text), disk copy is already up to date, so no rewrite is needed.
- The page to be read in just overwrites the page being evicted.

Page Replacement Algorithms

- If new page is brought in, need to chose a page to evict
- Don't want to evict heavily used pages
- Each operating system uses different page replacement algorithms.
- A page replacement algorithm is evaluated by running the particular algorithm on a string of memory references (references string) and compute the page faults.
- To select the particular algorithm, the algorithm with lowest page fault rate is considered.

Page Replacement Algorithms

- First-in, first-out page replacement
- Not recently used page replacement
- Least recently used page replacement
- Optimal page replacement algorithm
- Second chance page replacement
- Clock page replacement
- Working set page replacement
- WSClock page replacement

FIFO

- The oldest page in the physical memory is the one selected for replacement.
- Very simple to implement. - Keep a list
- On a page fault, the page at the head is removed and the new page added to the tail of the list

Issues

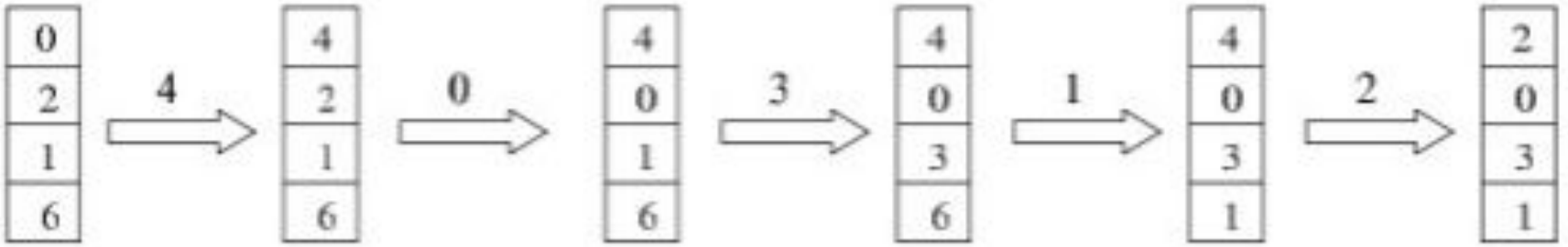
- Poor replacement policy
- Oldest might be most heavily used! No knowledge of use is included in FIFO.
- Doesn't consider the page usage.

FIFO Example with 4 memory frames

- Consider the following reference string: 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

X X X X X X X X X

Compulsory Misses



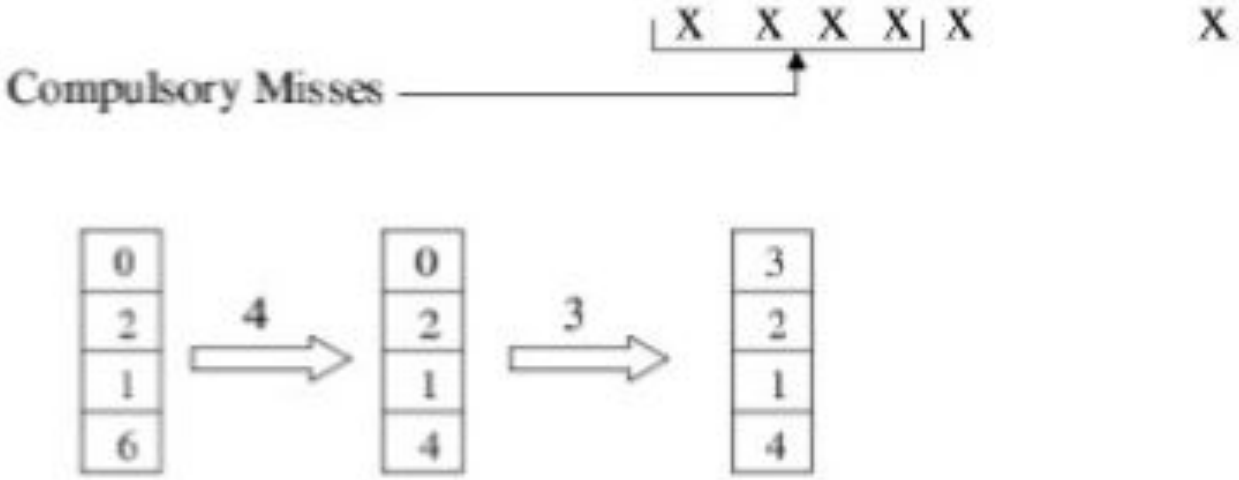
- Fault Rate = 9 / 12 = 0.75

Optimal Page Replacement

- Pages are replaced which would not be used for the longest duration of time in the future.
- Not possible unless know when pages will be referenced (crystal ball)
- Used as ideal reference algorithm
- Difficult to implement, because it requires future knowledge of the reference string.
- Mainly used for comparison studies
- The algorithm has lowest page fault rate of all algorithm

Optimal Page Replacement

- Consider the following reference string: 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1



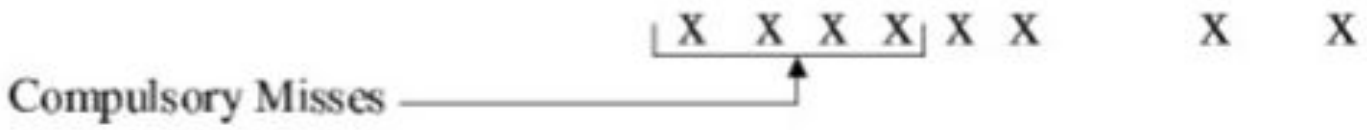
- Fault Rate = 6 / 12 = 0.50
- With the above reference string, this is the best we can hope to do

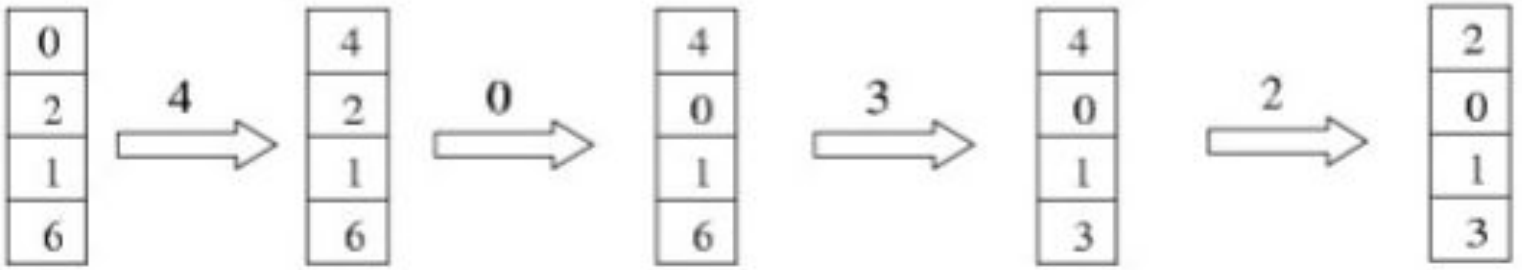
Least Recently Used: LRU

- LRU replacement associates with each page the time of that page's last use.
- When a page must be replaced, LRU chooses the page that has not been used for the longest period of time.
- Approximate LRU by assuming that recent page usage approximates long term page usage
- Could associate counters with each page and examine them but this is expensive
- Maintain a linked list of all pages in memory with the MRU page at front and LRU page at the rear.
- The difficulty is that the list must be updated on every memory reference.

LRU Example

- Consider the following reference string: 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1





- Fault Rate = 8 / 12 = 0.67

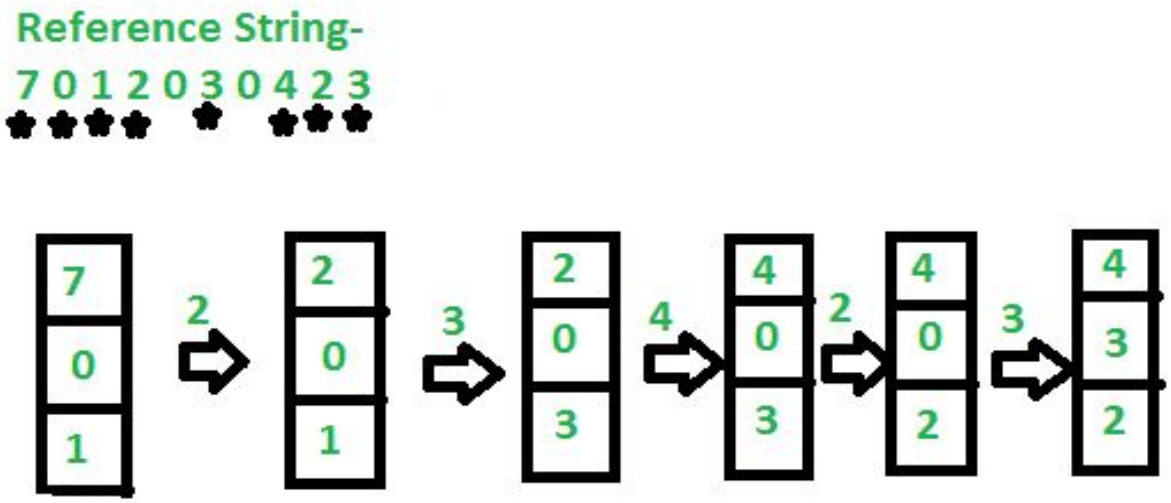
Not recently used (NRU)

- Two status bits R and M associated with each page
- R is set when a page is referenced; M is set when a page is modified.
- When a page fault occurs OS inspects all pages and divide them into following categories.
  - Class 0: not referenced, not modified
  - Class 1: not referenced, modified
  - Class 2: referenced, not modified
  - Class 3: referenced, modified
- Pick lowest priority page to evict (removes page at random with lowest class)

Not recently used (NRU): Example

From the given reference string NRU will remove a page at random from the lowest numbered nonempty class. Implicit in this algorithm is that it is better to remove a modified page that has not been referenced in at least one clock tick (typically 20 msec) than a clean page that is in heavy use.

Example –



Given page reference string:

1,2,3,4,2,1,5,6,2,1,2,3,7,6,3,2,1,2,3,6

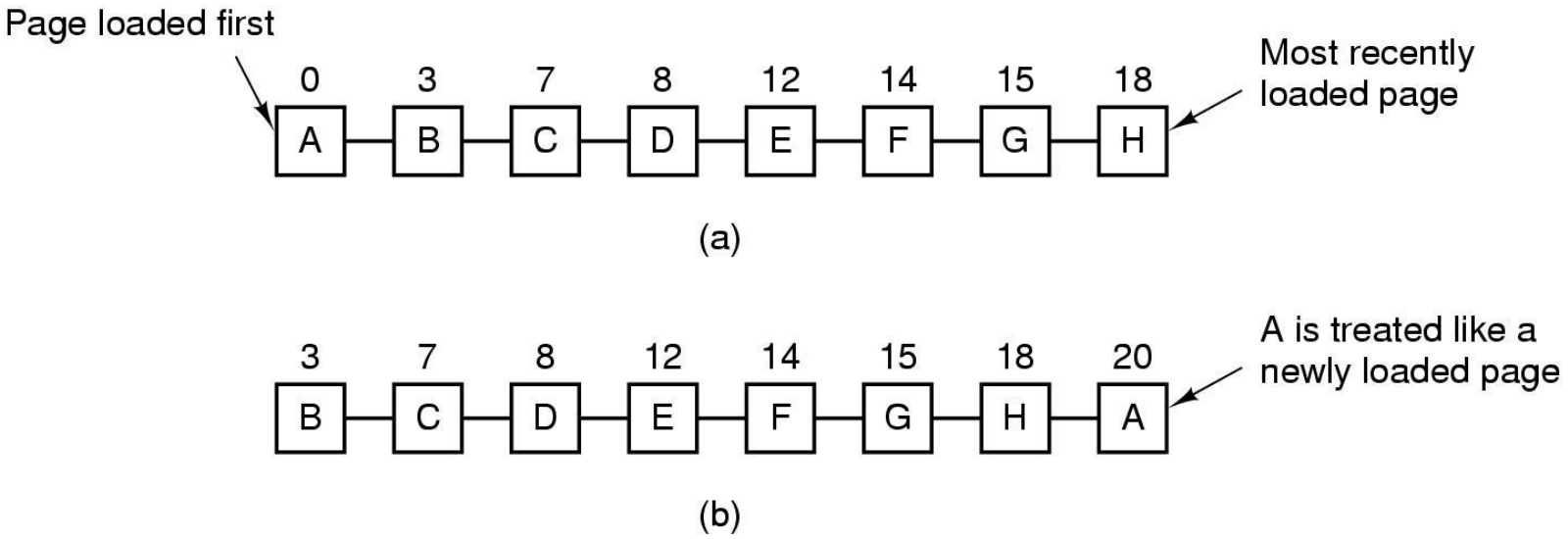
Compare the number of page faults for LRU,

FIFO and Optimal page replacement algorithm

second-chance page-replacement algorithm

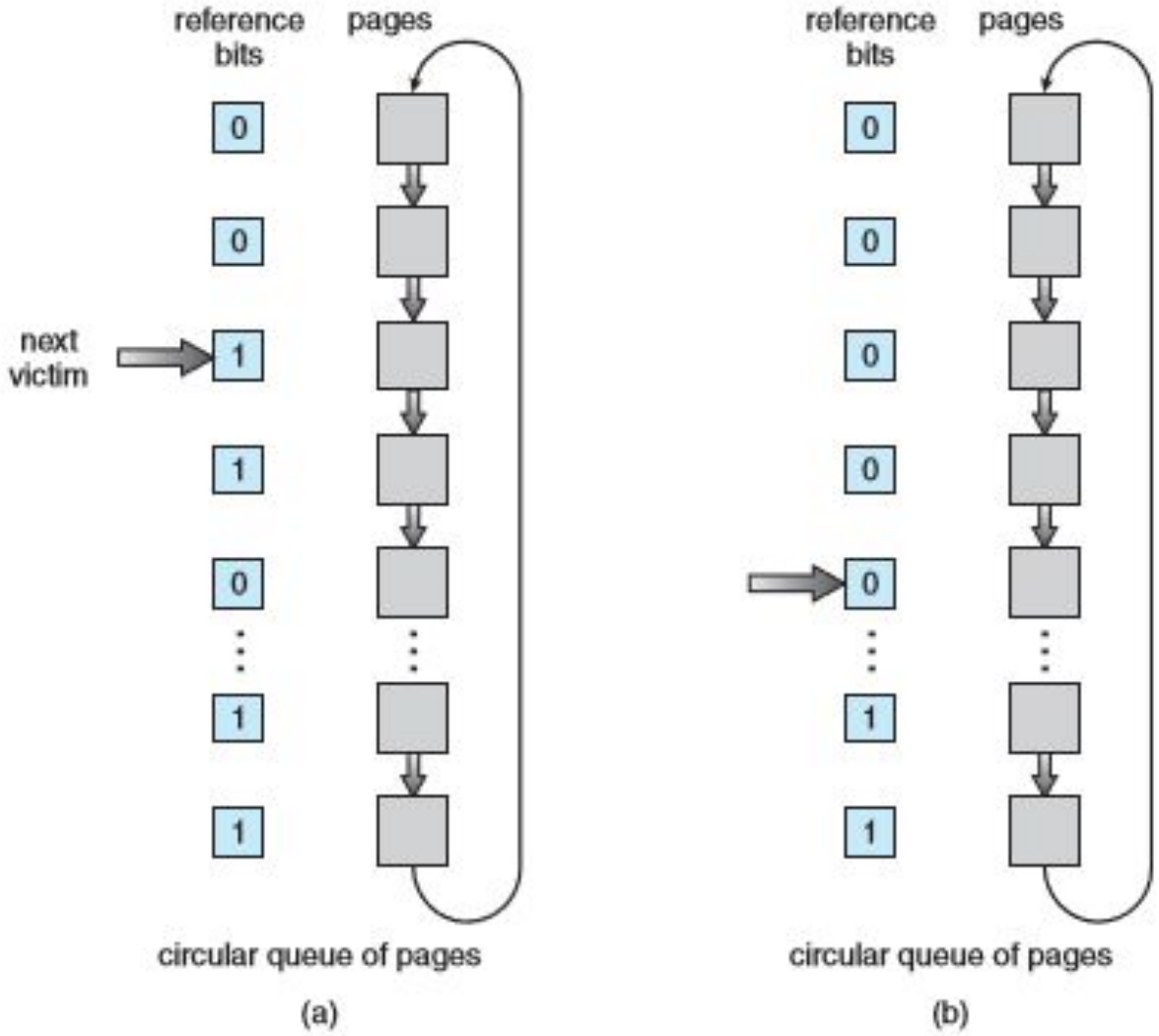
- The basic algorithm of second-chance replacement is a refinement of FIFO algorithm.
- It replaces page that is both oldest as well as unused instead of oldest page that may be heavily used.
- To keep track of page usage, it uses a reference bit (R) associated with each page and set when page is accessed.
- When a page has been selected, if its reference bit value is 0, we proceed to replace this page; but if the reference bit is set to 1, we give the page a second chance and move on to select the next FIFO page.
- When a page gets a second chance, its reference bit is cleared, and its arrival time is reset to the current time.
- Thus, a page that is given a second chance will not be replaced until all other pages have been replaced

Second Chance Algorithm

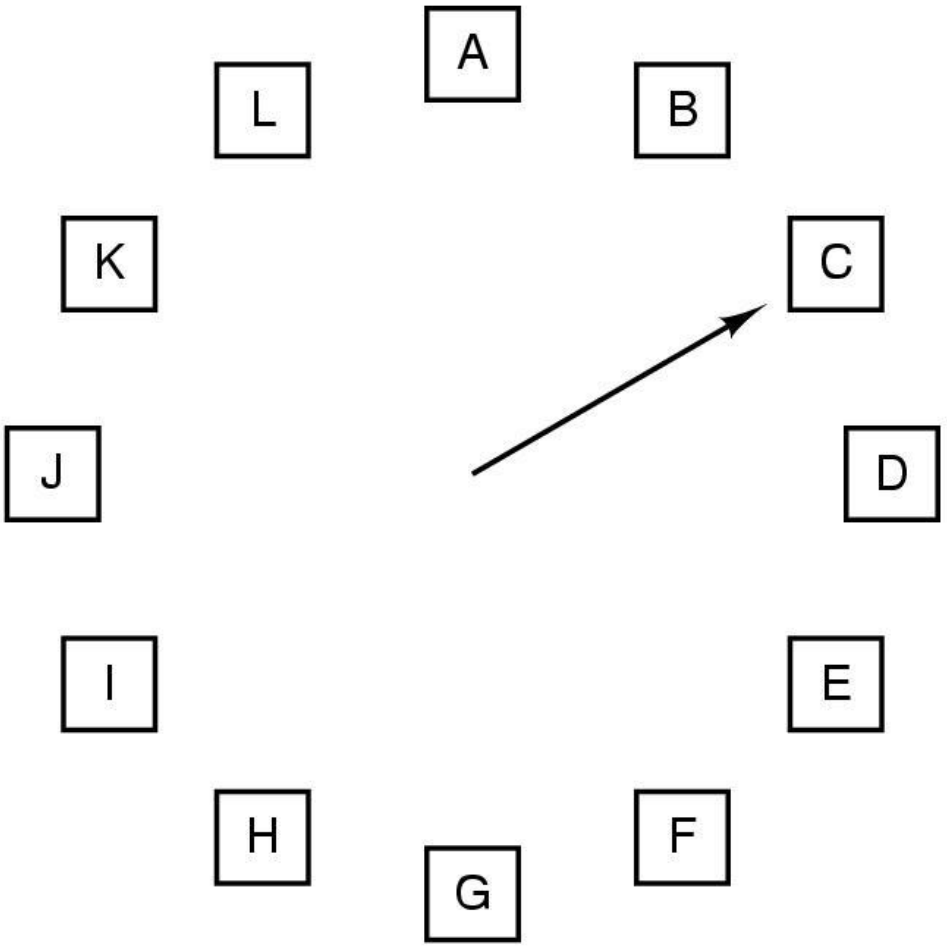


- Pages sorted in FIFO order by arrival time.
- Examine R bit. If zero, evict. If one, put page at end of list and R is set to zero.
- If change value of R bit frequently, might still evict a heavily used page

Second-chance (clock) page-replacement algorithm



Clock Page Replacement Algorithm



When a page fault occurs, the page the hand is pointing to is inspected. The action taken depends on the R bit:

R = 0: Evict the page

R = 1: Clear R and advance hand

CLOCK

Clock is a more efficient version of FIFO than Second-chance because pages don't have to be constantly pushed to back of the list, but it performs same general function as Second-Chance.

The clock algorithm keeps a circular list of pages in memory, with the "hand" (iterator) pointing to the last examined page frame in the list. When a page fault occurs and no empty frames exist, then the R (referenced) bit is inspected at the hand's location.

If R is 0, new page is put in place of the page "hand" points to, and the hand is advanced one position.

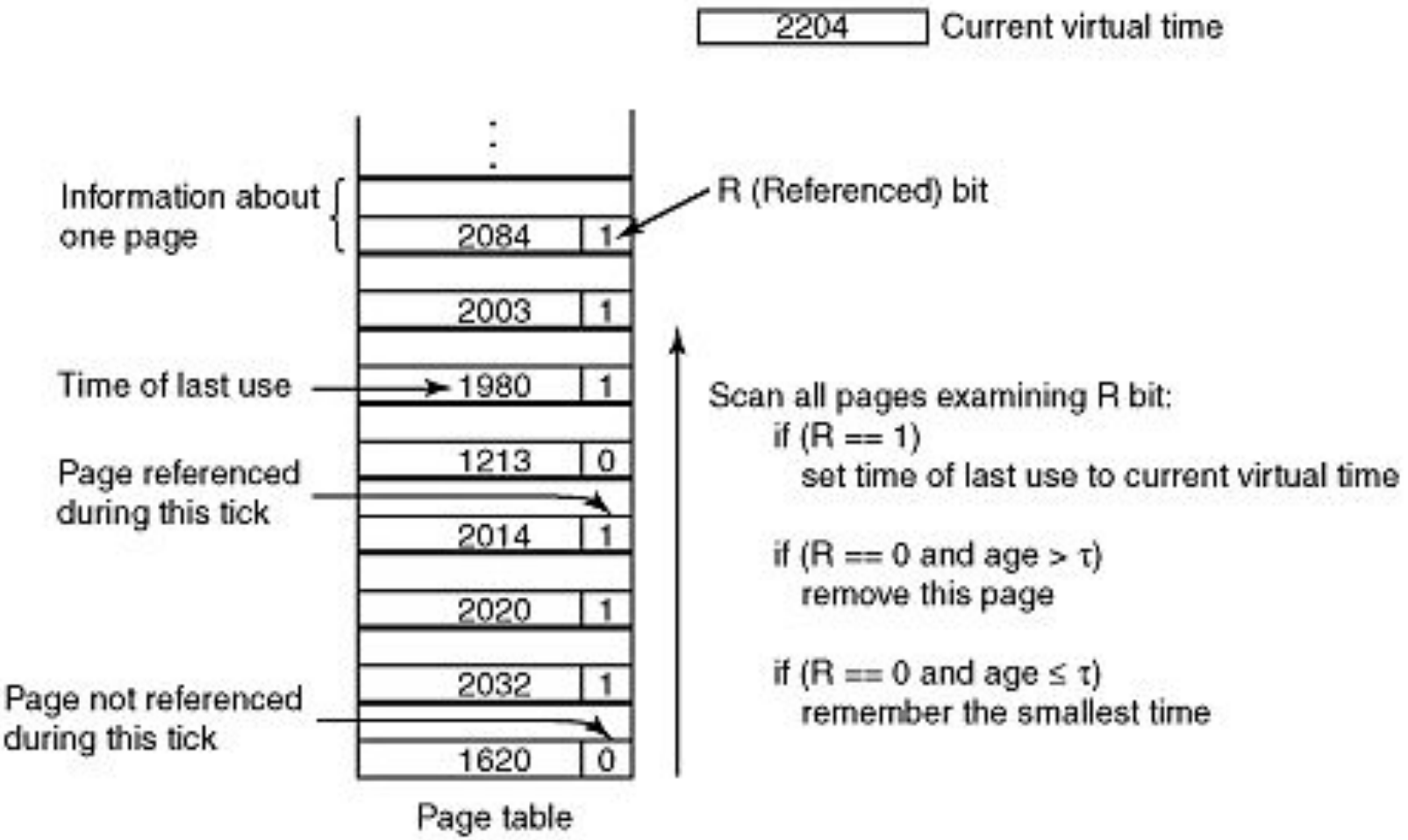
Otherwise, R bit is cleared, then clock hand is incremented and process is repeated until a page is replaced.

Thrashing

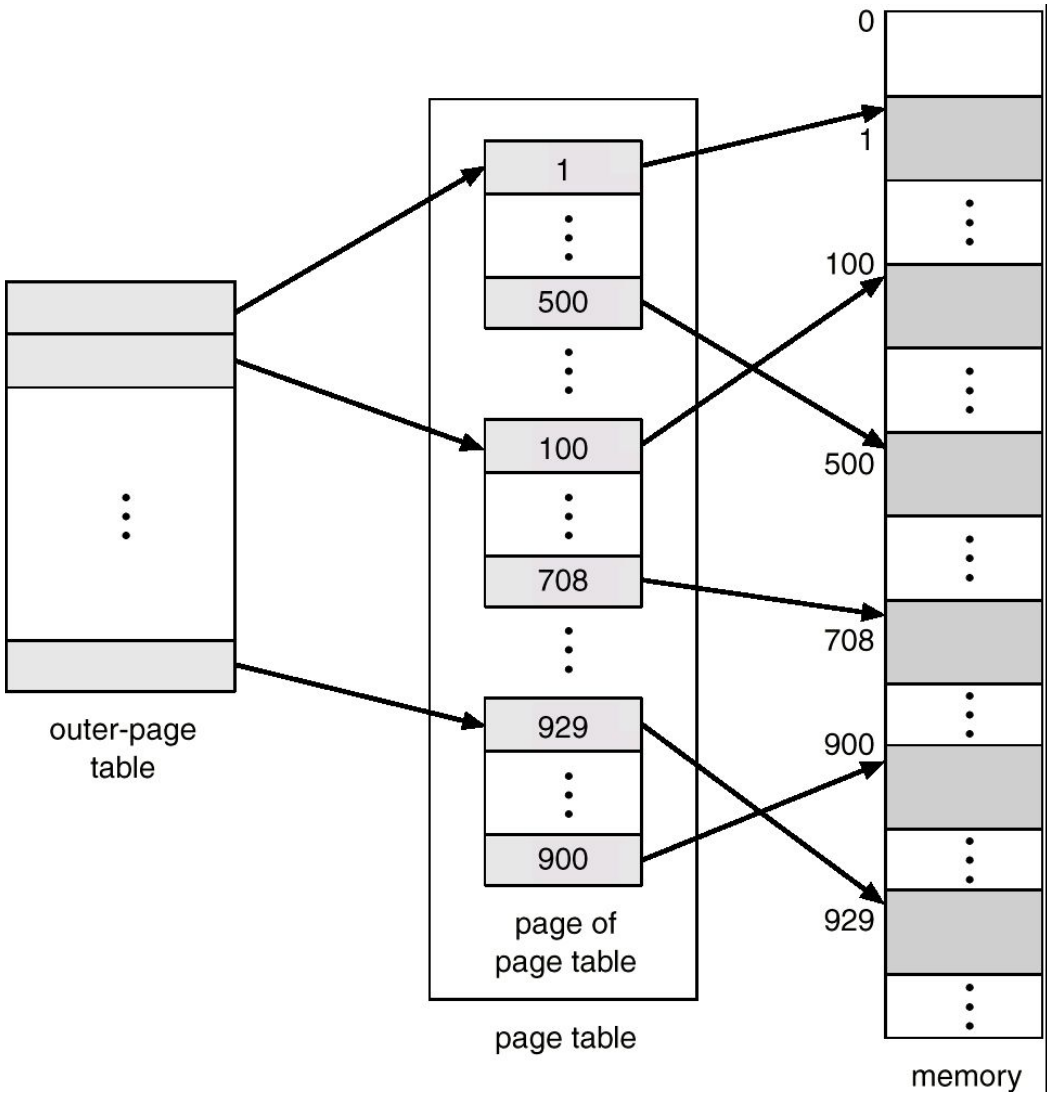
- Thrashing is excessive swapping of pages of data between memory and hard disk, causing application to respond slowly
- A process is thrashing if it is spending more time paging than executing.
- When each page in execution demands a page that is not currently in real memory (RAM) it places some pages on virtual memory and adjusts the required page on RAM.
- If the CPU is too busy in doing this task, thrashing occurs.
- If the process does not have the number of frames it needs to support pages in active use, it will quickly page-fault. At this point, it must replace some page.
- However, since all its pages are in active use, it must replace a page that will be needed again right away.
- Consequently, it quickly faults again, and again, and again, replacing pages that it must bring back in immediately.

Working Set Model

- The set of pages that a process is currently using is called its **working set**
- Demand paging-bring a process into memory by trying to execute first instruction and getting page fault. Continue until all pages that process needs to run are in memory (the working set)
- Try to make sure that working set is in memory before letting process run (pre-paging)
- Thrashing-memory is too small to contain working set, so page fault all of the time

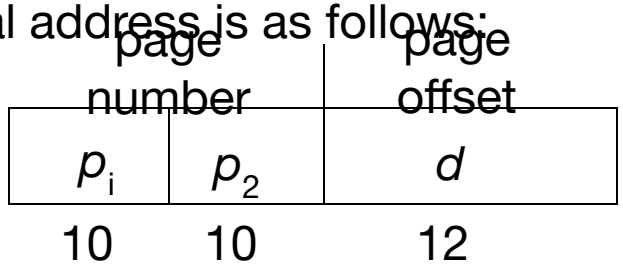

2204
Current virtual time
Information about one page
R (Referenced) bit
2084
1
2003
1
Time of last use
1980
1
1213
0
Page referenced during this tick
2014
1
2020
1
2032
1
Page not referenced during this tick
1620
0
Page table
Scan all pages examining R bit:
if (R == 1)
set time of last use to current virtual time
if (R == 0 and age > τ)
remove this page
if (R == 0 and age ≤ τ)
remember the smallest time

Two-Level Page-Table Scheme



Two-Level Paging Example

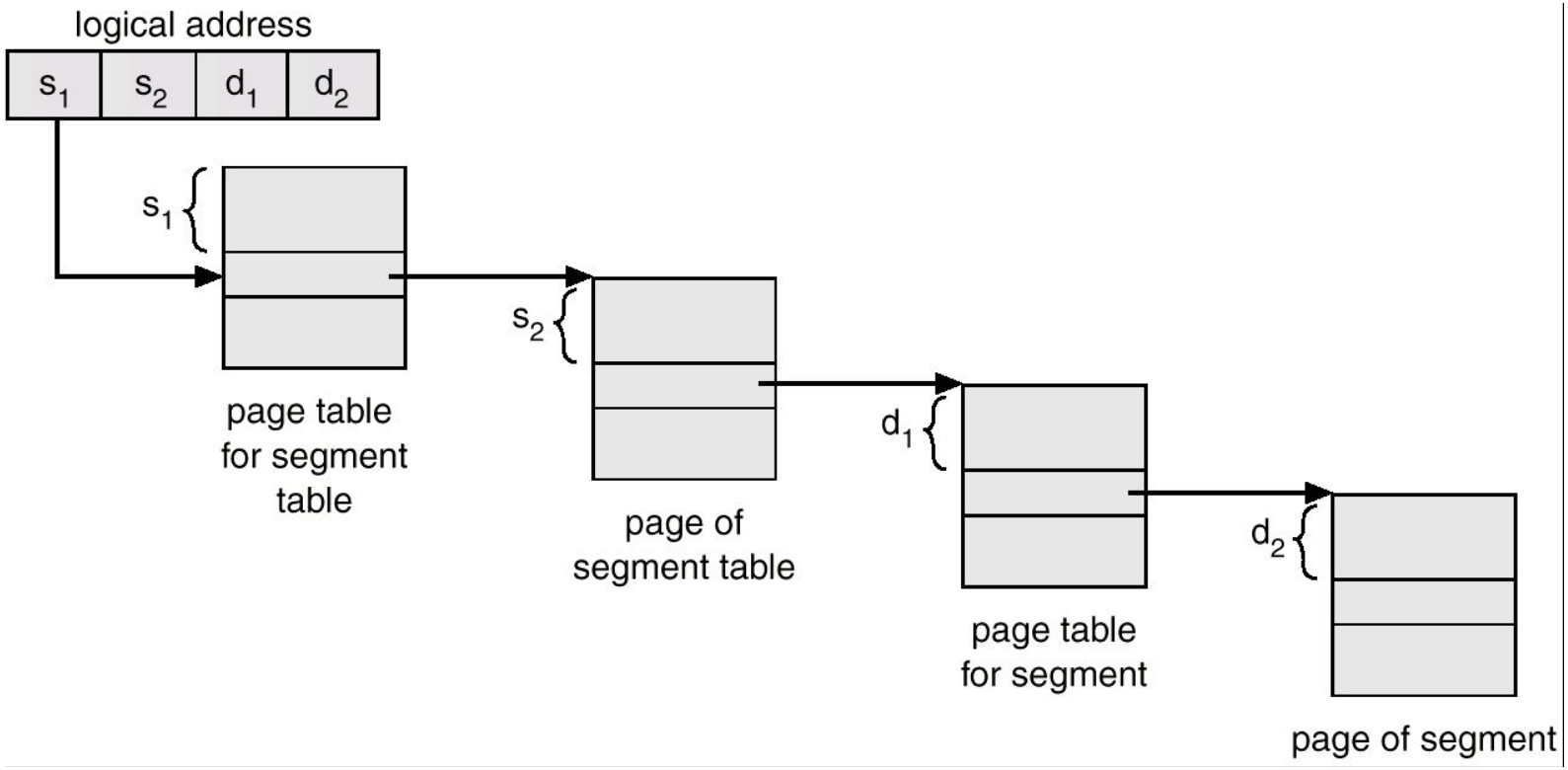
- A logical address (on 32-bit machine with 4K page size) is divided into:
  - a page number consisting of 20 bits.
  - a page offset consisting of 12 bits.
- Since the page table is paged, the page number is further divided into:
  - a 10-bit page number.
  - a 10-bit page offset.
- Thus, a logical address is as follows:



where p_i is an index into the outer page table, and p_2 is the displacement within the page of the outer page table.

Address-Translation Scheme

- Address-translation scheme for a two-level 32-bit paging architecture



Multilevel Paging and Performance

- Since each level is stored as a separate table in memory, covering a logical address to a physical one may take four memory accesses.
- Even though time needed for one memory access is quintupled, caching permits performance to remain reasonable.
- Cache hit rate of 98 percent yields:

  effective access time = 0.98 x 120 + 0.02 x 520

  = 128 nanoseconds.

  which is only a 28 percent slowdown in memory access time.

Bélády's anomaly

- In computer storage, Bélády's anomaly is the phenomenon in which increasing the number of page frames results in an increase in the number of page faults for certain memory access patterns. This phenomenon is commonly experienced when using the first-in first-out (FIFO) page replacement algorithm.

- Generally, on increasing the number of frames to a process' virtual memory, its execution becomes faster as less number of page faults occur. Sometimes the reverse happens, i.e. more number of page faults occur when more frames are allocated to a process. This most unexpected result is termed as Belady's Anomaly.

Belady's Anomaly in FIFO –Example

- Assuming a system that has no pages loaded in the memory and uses the FIFO Page replacement algorithm. Consider the following reference string:
- 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- **Case-1:** If the system has 3 frames, the given reference string on using FIFO page replacement algorithm yields a total of 9 page faults. The diagram below illustrates the

1	1	1	2	3	4	1	1	1	2	5	5
	2	2	3	4	1	2	2	2	5	3	3
		3	4	1	2	5	5	5	3	4	4
PF	PF	PF	PF	PF	PF	PF	X	X	PF	PF	X

Belady's Anomaly in FIFO –Example

- **Case-2:** If the system has 4 frames, the given reference string on using FIFO page replacement algorithm yields a total of 10 page faults. The diagram below illustrates the pattern of the page faults occurring in the example. 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

1	1	1	1	1	1	2	3	4	5	1	2
	2	2	2	2	2	3	4	5	1	2	3
		3	3	3	3	4	5	1	2	3	4
			4	4	4	5	1	2	3	4	5
PF	PF	PF	PF	X	X	PF	PF	PF	PF	PF	PF

- In this example on increasing the number of frames while using the FIFO page replacement algorithm, the number of **page faults increased** from 9 to 10.
- **Note –** It is not necessary that every string reference pattern cause Belady anomaly in FIFO but there are certain kind of string references

RSET

RAJAGIRI SCHOOL OF
ENGINEERING & TECHNOLOGY

ITT303
OPERATING SYSTEM CONCEPTS

Module I

Course Out line

- No of Hours: 4(3+1)
- (1) Introduction to OS,
- (2) Process Management,
- (3) Process Synchronization,
- (4) Memory Management
- (5) Storage Management

Text Books

1. Andrew S. Tanenbaum, “Modern Operating Systems”, Prentice Hall
2. 2. J. L. Peterson and A. Silberschatz , Operating System Concepts, Addison Wesley.

Course Outcome

- Explain the concepts and functionality of operating systems.
- Describe the concepts of process management and process synchronization and apply them to solve problems.
- Illustrate deadlock and deadlock – prevention and avoidance techniques.
- Illustrate the memory management techniques.
- Explain the file system and its implementation
- Use the disk scheduling algorithms to solve problems.

Syllabus

1. Introduction: Operating Systems-different types, System kernel, Shell,
2. Processes- . Process Scheduling methods, Inter process Communication,
3. Memory management : fixed &variable partitions - paging & segmentation - virtual memory concepts - demand paging - page replacement
4. Device management : disk scheduling algorithms - sector queuing -device drivers.
5. Dead locks - conditions for deadlock - prevention - avoidance - detection – recovery from dead lock -bankers' algorithm. - resource trajectories –starvation,
6. File system concepts – Access methods – Directory structure – Directory implementation – Linear list, Hash table

- OPERATING SYSTEM AND NETWORK PROGRAMMING LAB

Course Code : ITL 331

Course Outcome

- Analyse CPU Scheduling Algorithms like FCFS, Round Robin, SJF and Priority.
- Implement inter process communication and process synchronization problems.
- Implement memory management schemes - first fit, best fit and worst fit.
- Implement client server communication using sockets.
- Implement MAC protocols.
- Familiarization of network simulation tool.

Assessment Pattern:

Mark distribution

Total Marks	CIE	ESE	ESE Duration
150	75	75	2.5 hours

- **Continuous Internal Evaluation Pattern:**
- Attendance : 15 marks
- Continuous Assessment : 30 marks
- Internal Test (Immediately before the second series test) : 30 marks

End Semester Examination Pattern: The following guidelines should be followed regarding award of marks

(a) Preliminary work	: 15 Marks
(b) Implementing the work/Conducting the experiment	: 10 Marks
(c) Performance, result and inference (usage of equipments and trouble shooting)	: 25 Marks
(d) Viva voce	: 20 marks
(e) Record	: 5 Marks

Module-I Contents

- Operating Systems: Introduction, Functions of OS, Types of OS (Batch, Multi programmed, Time-sharing and Real time systems) –System calls – System Programs –– System structure (Simple structure, Layered approach, Microkernel system structure, Modules)– Kernel, Shell.

Introduction

What Is An Operating System

A modern computer consists of:

- One or more processors
- Main memory
- Disks
- Printers
- Various input/output devices

Managing all these components requires a layer of software – the **operating system**

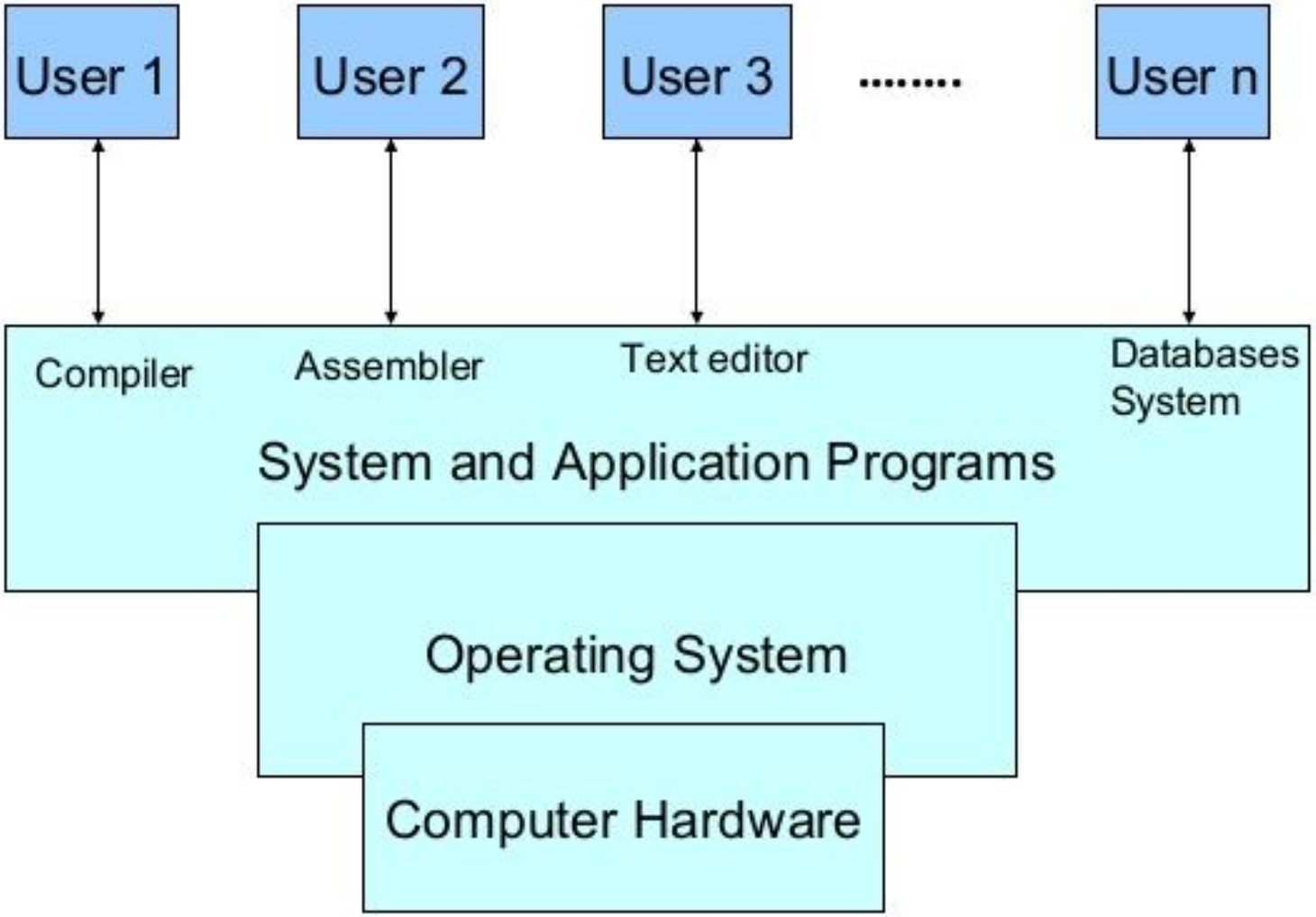


Figure: Abstract view of the components of a computer system

Computer System Structure

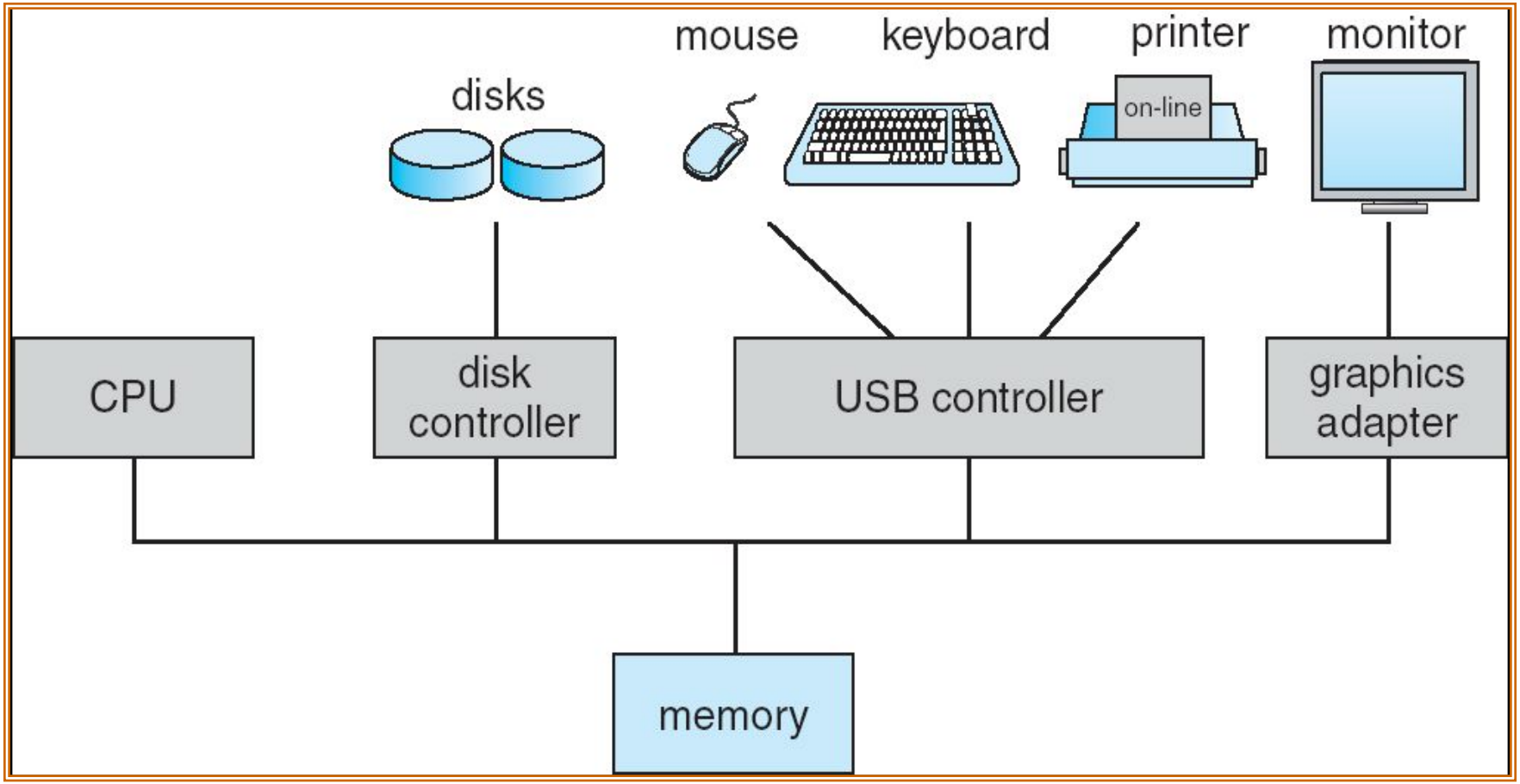
Computer system can be divided into four components

1. Hardware – provides basic computing resources
   - CPU, memory, I/O devices
2. Operating system
   - Controls and coordinates use of hardware among various applications and users
3. Application programs – define the ways in which the system resources are used to solve the computing problems of the users
   - Word processors, compilers, web browsers, database systems, video games
4. Users
   - People, machines, other computers

What is an Operating System?

- A collection of programs that acts as an interface between a user of a computer and the computer hardware.
- Provides an environment within which other programs can do useful work.
- Operating system goals:
  - Execute user programs and make solving user problems easier.
  - Make the computer system convenient to use.
  - Use the computer hardware in an efficient manner.
  - Security and protection to the user activities (prevent illegal accesses)

Computer System Organization



- Computer-system operation
  - One or more CPUs, device controllers connect through common bus providing access to shared memory
  - Concurrent execution of CPUs and devices competing for memory cycles

Operating System Definition

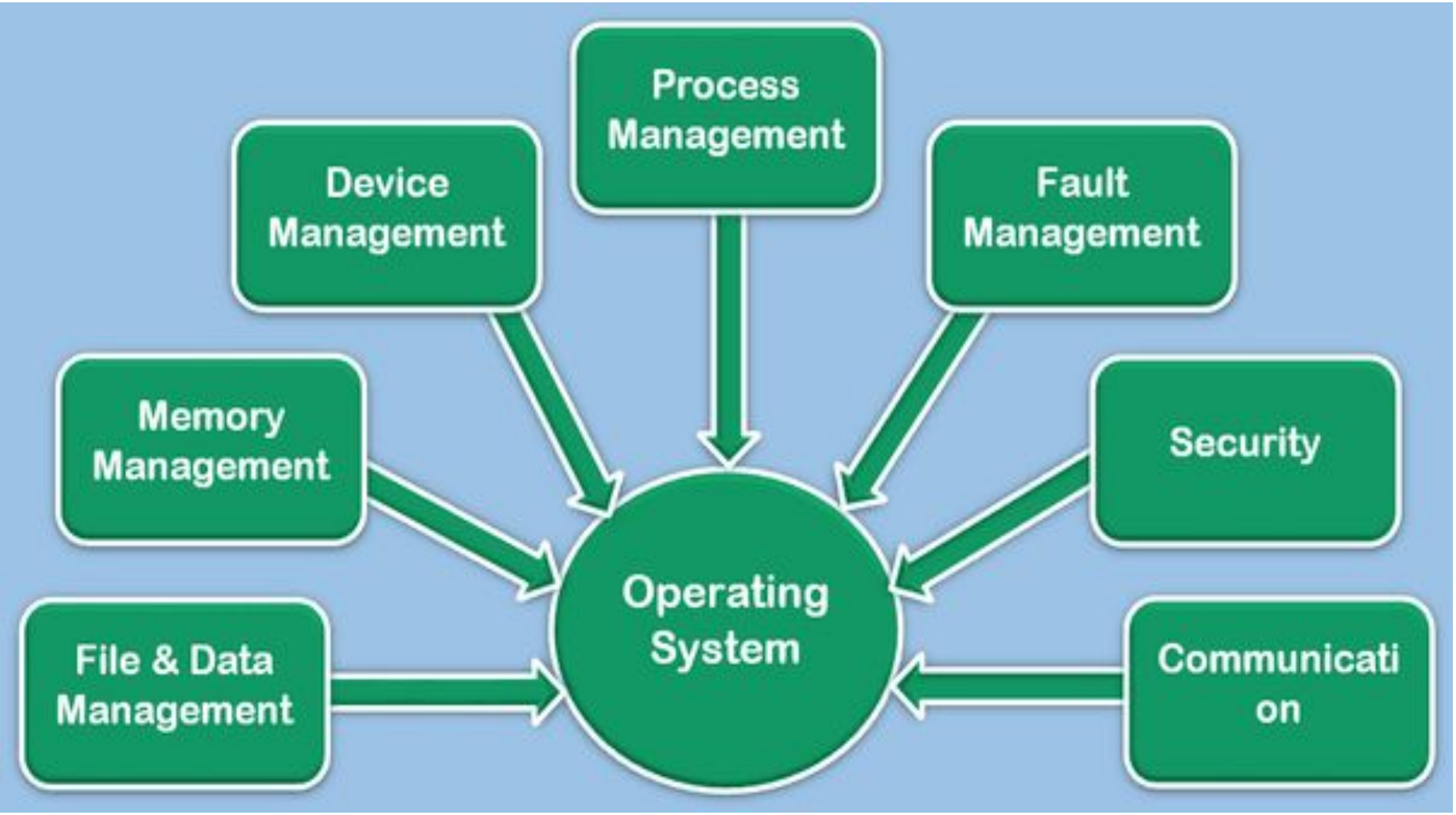
- OS is a **resource allocator**
  - Manages all resources (hardware and software)
  - Decides between conflicting resource requests for efficient and fair resource use.
- OS is a **control program**
  - Controls execution of programs to prevent errors and improper use of the computer.

OS as a Resource Manager

- Allow multiple programs to run at the same time
- Manage and protect memory, I/O devices, and other resources
- Includes multiplexing (sharing) resources in two different ways:
  - In time
  - In space

OS Functionalities/roles

- OS is responsible for all the functions of hardware as well as software



Function of OS

OS provides the environment within which programs are executed.

Function of OS:

- Process Management
- Main Memory Management
- Secondary-Storage Management
- I/O System Management
- File Management
- Command-Interpreter System
- Protection System
- Networking

Process Management

- A process is a program in execution. It is a unit of work within the system. A program is a passive entity, while a process is an active entity.
- A process needs resources to accomplish its task
- CPU, memory, I/O, files
- Initialization data
- Process termination requires the reclaiming of any reusable resources
- Typically a system has many processes, some user, some operating system processes, all running concurrently on one or more CPUs
- Concurrency is achieved by multiplexing the CPUs among the processes / threads

Process Management Activities

- The operating system is responsible for the following activities in connection with process management:
- Creating and deleting both user and system processes
- Suspending and resuming processes
- Providing mechanisms for process synchronization
- Providing mechanisms for process communication
- Providing mechanisms for deadlock handling

Memory Management

- Main memory is a repository of quickly accessible data shared by the CPU and I/O devices
- The CPU reads instructions from main memory during the instruction fetch-execute cycle and both reads and writes data from main memory
- For a program to be executed, it must be mapped to absolute addresses and loaded into memory
- The operating system is responsible for the following memory management activities:
- Keeping track of which parts of memory are currently being used and by whom
- Deciding which processes (or parts thereof) and data to move into and out of memory
- Allocating and deallocating memory space as needed

Storage Management

- The operating system provides a uniform, logical view of information storage
- It abstracts from the physical properties of a storage device to a logical storage unit called a file
- The operating system maps files onto physical media and accesses these files by way of the storage devices
- File Management
- Visible component of an OS
- Storage of information possible in magnetic disc, magnetic tapes etc
- Each storage medium has varying properties that include access speed, capacity, data-transfer rate, an access method (sequential or random

- File
- collection of related information by its creator,
- Files can be of any type
- Files are usually organized into directories
- Access control on most systems determines who can access what

- The OS is responsible for the following file management activities:
- Creating and deleting files and directories
- Supporting operations manipulate files and directories
- Mapping files onto secondary storage
- Backing up files onto stable (non-volatile) storage media

Mass Storage Management

- Main memory too small to accommodate all data and programs
- Also data that it holds are lost when power is lost
- Computer must provide secondary storage to back up main memory.
- Most computers use disks as the on-line storage medium for both programs and data.
- Most programs like compilers, word processors etc are stored in disk until loaded into memory.
- So management of disk storage is important.

Mass Storage Management

- The entire speed of computer operation hinges on disk subsystem and its algorithms
- The OS is responsible for the following disk management activities
- Free-space management
- Storage allocation
- Disk scheduling

- Some storage need not be fast-Tertiary Storage
- There are storage that is slower and lower in cost, but sometimes of higher capacity than secondary storage.
- Eg: Backups of disk data, seldom-used data, long term archival storage.
- Tertiary storage is not crucial to system performance, but still must be managed.
- like mounting and unmounting media in devices, migrating data from secondary to tertiary storage etc.

I/O Subsystem

- One of the purposes of an operating system is to hide peculiarities of hardware devices from the user
- In UNIX, the peculiarities of I/O devices are hidden from the bulk of the operating system itself by the I/O subsystem
- The I/O subsystem consists of several components
- A memory-management component that includes buffering, caching, and spooling
- A general device-driver interface
- Drivers for specific hardware devices
- Only the device driver knows the peculiarities of the specific device to which it is assigned

Protection and Security

- If a computer system has multiple users and allows for concurrent execution of multiple processes, access to data must be regulated.
- Mechanisms ensure that files, memory segments, CPU and other resources can be operated only by one of the processes that have gained proper authorization from the OS
- Eg:
- A process can execute only within its own address space
- A timer ensures that no process can gain control of CPU without eventually relinquishing control.
- Device control registers are not accessible to users

- Protection – any mechanism for controlling access of processes or users to the resources defined by the operating system
- This must provide means to specify the controls to be imposed and to enforce the controls.
- It can improve reliability by detecting latent errors at the interfaces.

Protection and Security

- Security – defends a system from internal and external attacks
- This is a huge range of threats, including denial-of-service, viruses, identity theft, and theft of service
- Systems generally first distinguish among users, to determine who can do what
- User identities (user IDs) include name and associated number, one per user
- The user ID then is associated with all files and processes of that user to determine access control
- A group identifier (group ID) allows a set of users to be defined an associated with each process, directory and file
- Privilege escalation allows user to change to an effective ID with more rights for a short period of time

Process Management

- A *process* is a program in execution. A process needs certain resources, including CPU time, memory, files, and I/O devices, to accomplish its task.
- The operating system is responsible for the following activities in connection with process management.
  - Process creation and deletion.
  - process suspension and resumption.
  - Provision of mechanisms for:
    - process synchronization
    - process communication

Main-Memory Management

- Memory is a large array of words or bytes, each with its own address. It is a repository of quickly accessible data shared by the CPU and I/O devices.
- Main memory is a volatile storage device. It loses its contents in the case of system failure.
- The operating system is responsible for the following activities in connections with memory management:
  - Keep track of which parts of memory are currently being used and by whom.
  - Decide which processes to load when memory space becomes available.
  - Allocate and deallocate memory space as needed.

Secondary-Storage Management

- Since main memory (*primary storage*) is volatile and too small to accommodate all data and programs permanently, the computer system must provide *secondary storage* to back up main memory.
- Most modern computer systems use disks as the principle on-line storage medium, for both programs and data.
- The operating system is responsible for the following activities in connection with disk management:
  - Free space management
  - Storage allocation
  - Disk scheduling

I/O System Management

- The I/O system consists of:
  - A buffer-caching system
  - A general device-driver interface
  - Drivers for specific hardware devices
- I/O system management activities are:
  - I/O interrupt servicing,
  - initiation of I/O operations,
  - optimization of I/O device performance

File Management

- A file is a collection of related information defined by its creator. Commonly, files represent programs (both source and object forms) and data.
- The operating system is responsible for the following activities in connections with file management:
  - File creation and deletion.
  - Directory creation and deletion.
  - Support of primitives for manipulating files and directories.
  - Mapping files onto secondary storage.
  - File backup on stable (nonvolatile) storage media.

Protection System

- *Protection* refers to a mechanism for controlling access by programs, processes, or users to both system and user resources.
- The protection mechanism must:
  - distinguish between authorized and unauthorized usage.
  - specify the controls to be imposed.
  - provide a means of enforcement.

Networking (Distributed Systems)

- A *distributed* system is a collection processors that do not share memory or a clock. Each processor has its own local memory.
- The processors in the system are connected through a communication network.
- A distributed system provides user access to various system resources.
- Access to a shared resource allows:
  - Computation speed-up
  - Increased data availability
  - Enhanced reliability

Command-Interpreter System

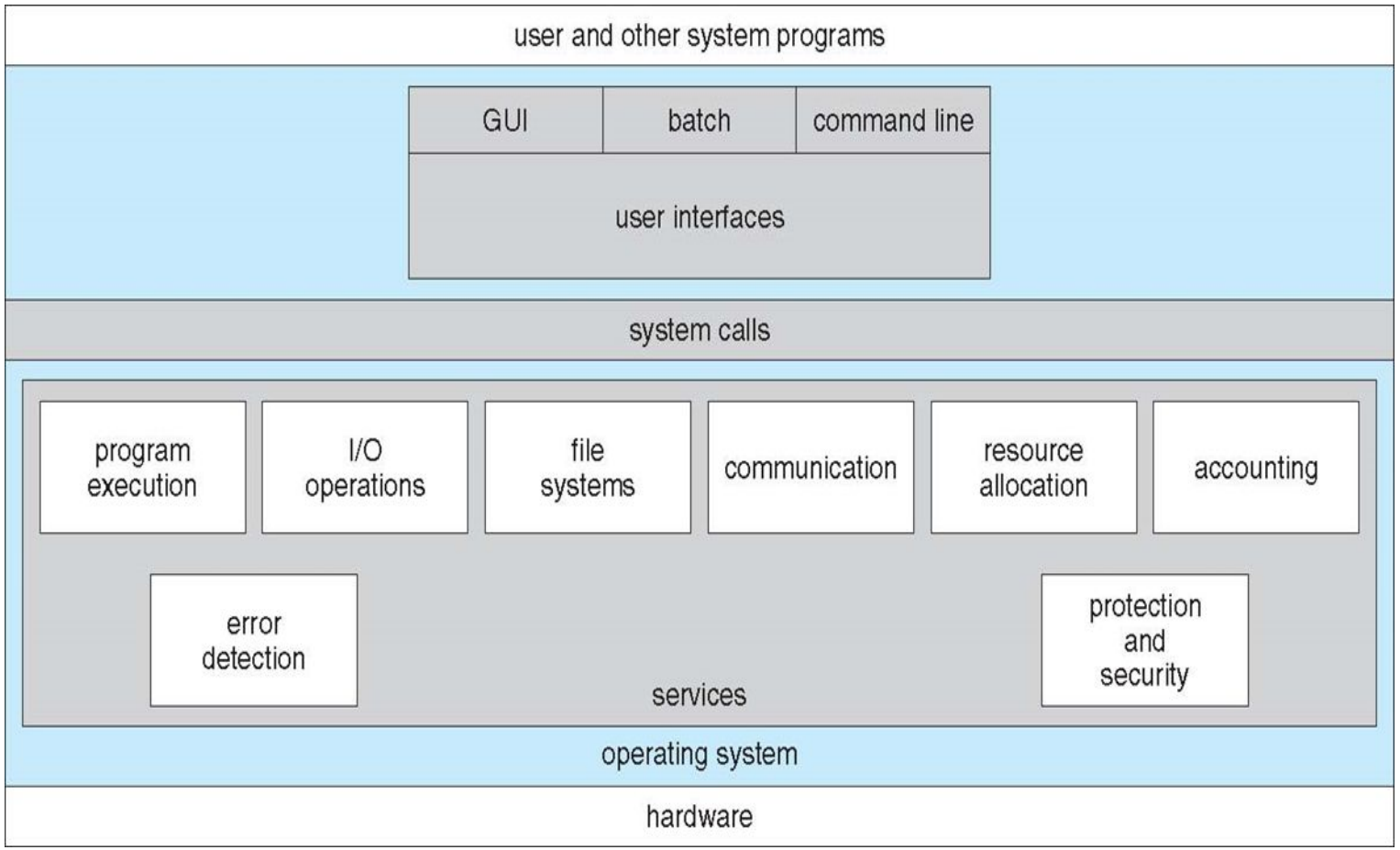
- Many commands are given to the operating system by control statements which deal with:
  - process creation and management
  - I/O handling
  - secondary-storage management
  - main-memory management
  - file-system access
  - protection
  - networking

Additional Operating System Functions

Additional functions exist not for helping the user, but rather for ensuring efficient system operations.

- Resource allocation – allocating resources to multiple users or multiple jobs running at the same time.
- Accounting – keep track of and record which users use how much and what kinds of computer resources for account billing or for accumulating usage statistics.
- Protection – ensuring that all access to system resources is controlled.

OS Services


user and other system programs
GUI
batch
command line
user interfaces
system calls
program execution
I/O operations
file systems
communication
resource allocation
accounting
error detection
protection and security
services
operating system
hardware

OS Services

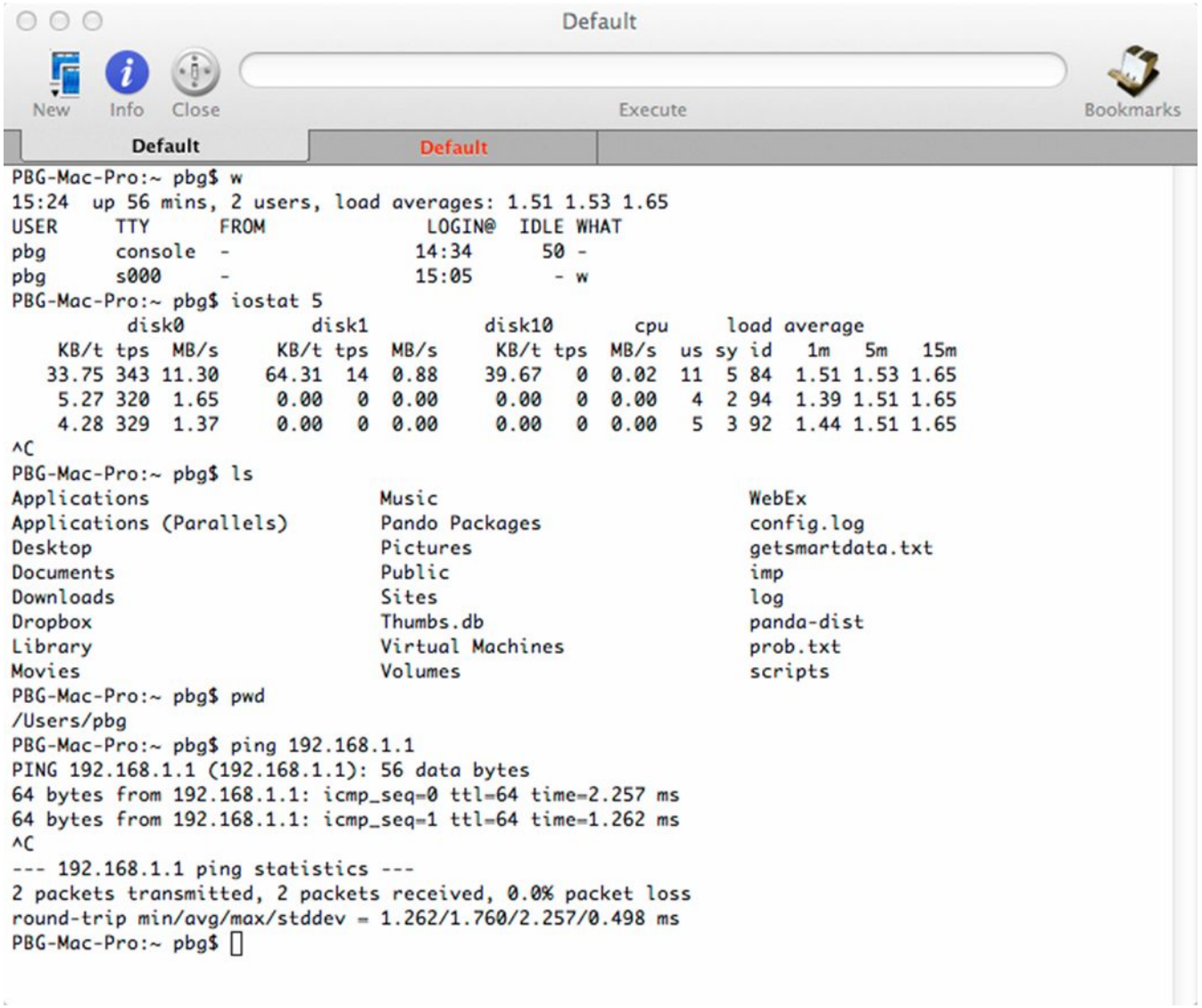
- User Interface
- Program Execution
- I/O operation
- File system manipulation
- Communication between processes
- Error Detection
- Resource allocation
- Accounting: keeping usage statistics
- Protection and security: control resource access

User Operating System Interface - CLI

CLI or **command interpreter** allows direct command entry

- Sometimes implemented in kernel, sometimes by systems program
- Sometimes multiple flavors implemented – **shells**
- Primarily fetches a command from user and executes it
- Sometimes commands built-in, sometimes just names of programs
  - If the latter, adding new features doesn't require shell modification

Bourne Shell Command Interpreter



User Operating System Interface - GUI

- User-friendly desktop metaphor interface
- Usually mouse, keyboard, and monitor
- Icons represent files, programs, actions, etc
- Various mouse buttons over objects in the interface cause various actions (provide information, options, execute function, open directory (known as a folder)
- Invented at Xerox PARC
- Many systems now include both CLI and GUI interfaces
- Microsoft Windows is GUI with CLI “command” shell
- Apple Mac OS X is “Aqua” GUI interface with UNIX kernel underneath and shells available
- Unix and Linux have CLI with optional GUI interfaces (CDE, KDE, GNOME)

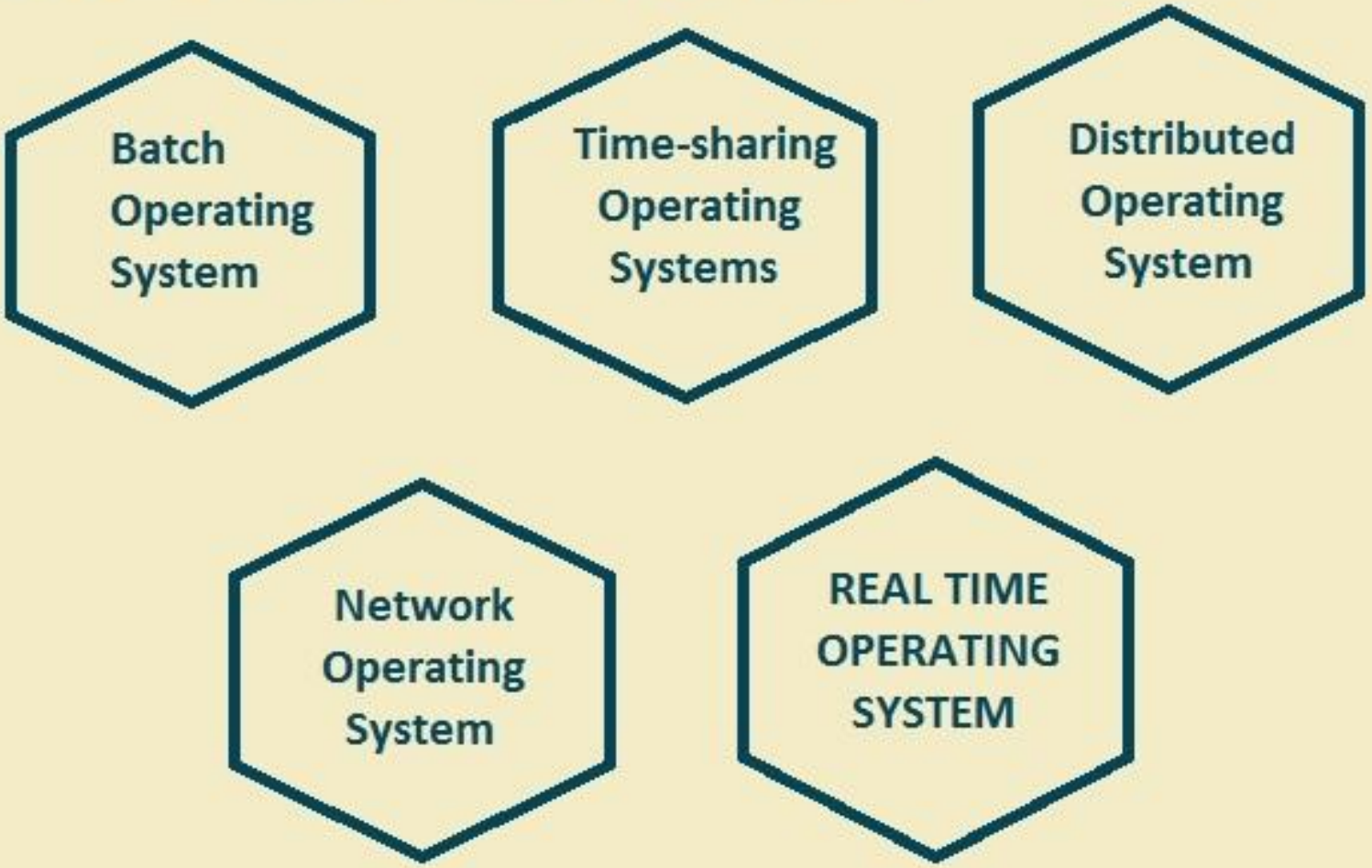
Touchscreen Interfaces

- Touchscreen devices require new interfaces
- Mouse not possible or not desired
- Actions and selection based on gestures
- Virtual keyboard for text entry
- Voice commands.

Operating System Services

- Program execution – system capability to load a program into memory and to run it.
- I/O operations – since user programs cannot execute I/O operations directly, the operating system must provide some means to perform I/O.
- File-system manipulation – program capability to read, write, create, and delete files.
- Communications – exchange of information between processes executing either on same computer or on different systems tied together by a network. Implemented via *shared memory* or *message passing*.
- Error detection – ensure correct computing by detecting errors in CPU and memory hardware, in I/O devices, or in user programs.

TYPES OF OPERATING SYSTEM



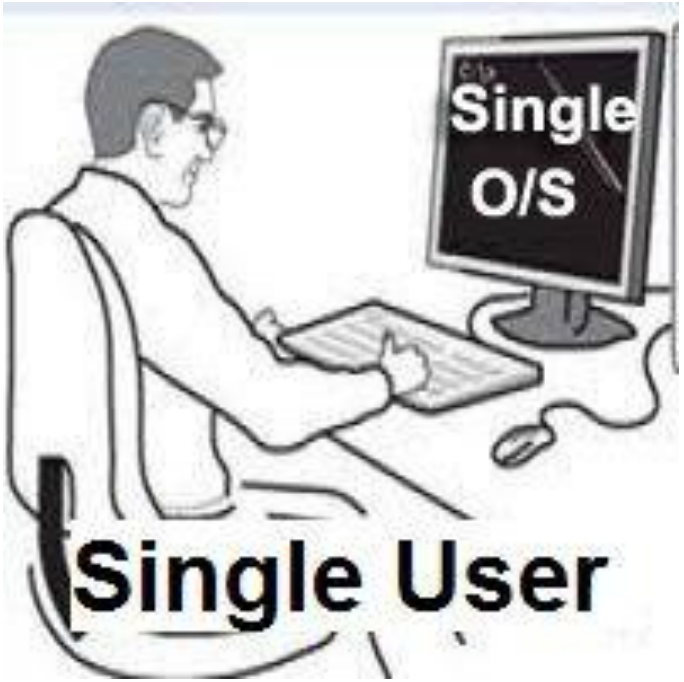
Types of Operating Systems

Following are some of the most widely used types of Operating system.

- single user/single task system
- Simple Batch System
- Multiprogramming Batch System
- Real time Operating System
- Time sheard OS

Single-user, single task

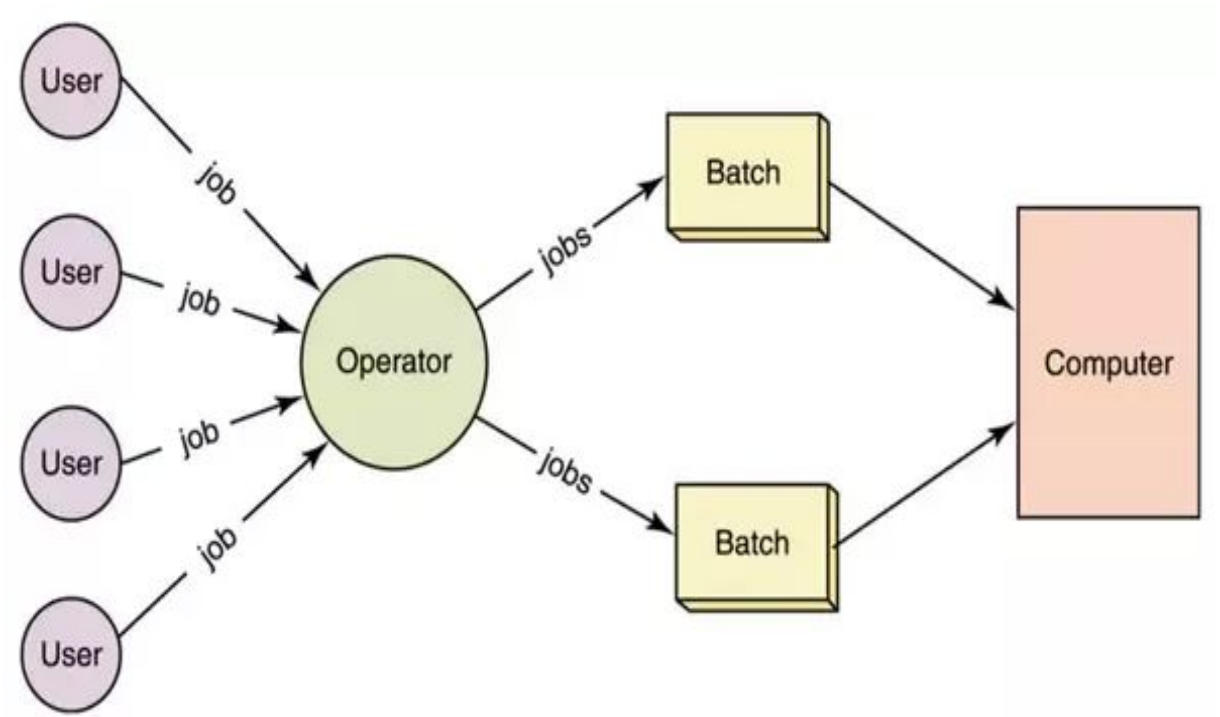
- As the name implies, this operating system is designed to manage the computer so that one user can effectively do one thing at a time. The Palm OS for Palm handheld computers is a good example of a modern single-user, single-task operating system.



Simple Batch Systems

- In this type of system, there is no direct interaction between user and the computer.
- The user has to submit a job (written on cards or tape) to a computer operator.
- Then computer operator sorts the programs with similar requirements into batches and places a batch of several jobs on an input device.
- Then a special program, called monitor, manages the execution of each program in the batch.
- The output from each job would be sent back to the appropriate programmer.

Simple Batch System Memory layout

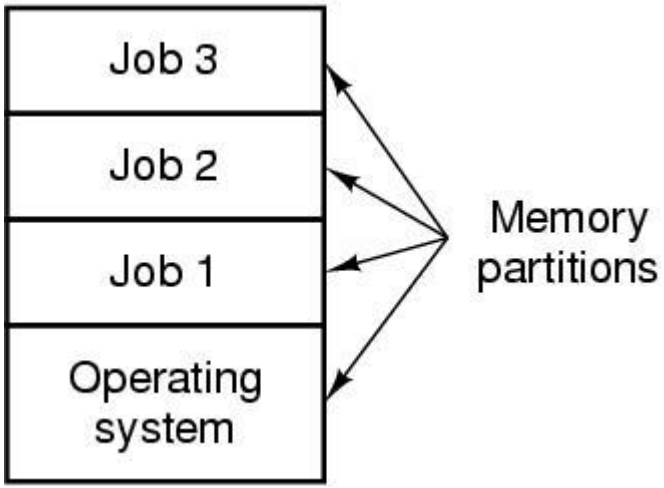


Disadvantages of Simple Batch OS

- No interaction between user and computer.
- CPU is often idle, because the speed of the mechanical I/O devices is slower than the CPU.
- No mechanism to prioritize the processes.

Multiprogramming batch Systems

- Multiprogramming needed for efficiency
- Single program cannot keep CPU and I/O devices busy always.
- Multiprogramming organizes jobs (code and data) so CPU always has one to execute.
- OS keeps several jobs in memory simultaneously.
- One job selected and run via job scheduling

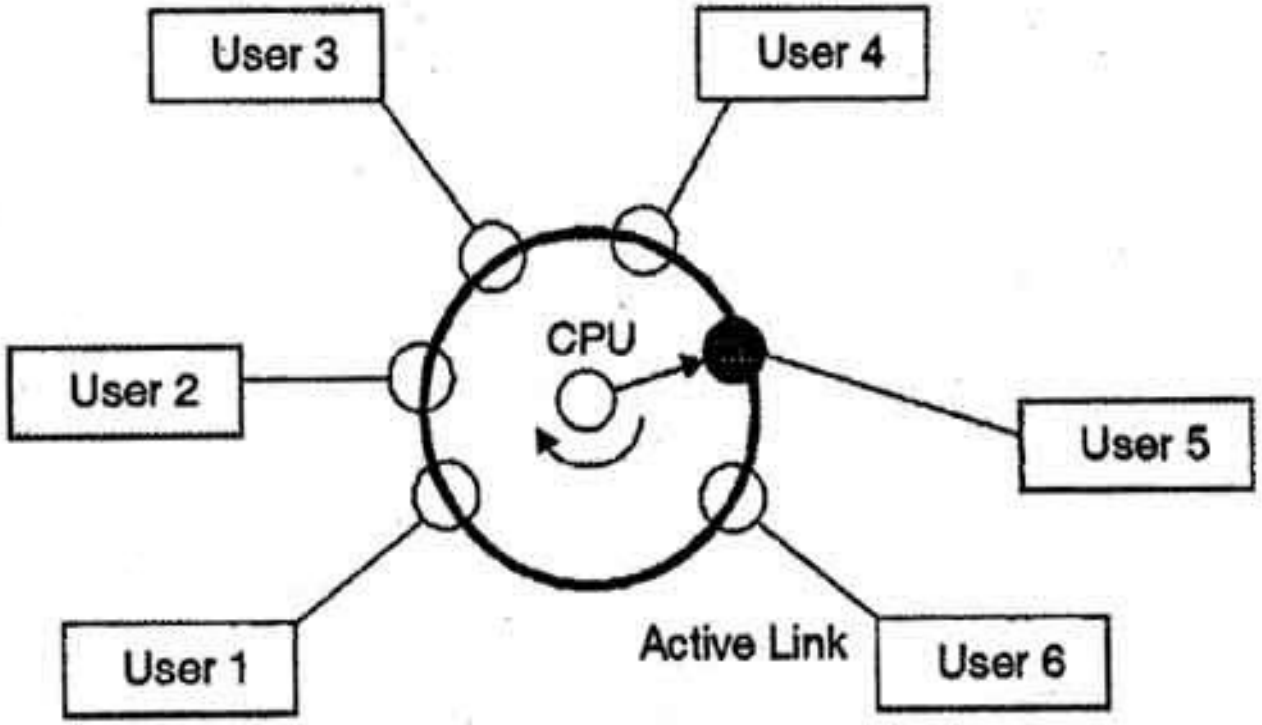


Multiprogramming Batch Systems

- In this the operating system picks up and begins to execute one of the jobs from memory.
- Once this job needs an I/O operation operating system switches to another job (CPU and OS always busy).
- Jobs in the memory are always less than the number of jobs on disk(Job Pool).
- If several jobs are ready to run at the same time, then the system chooses which one to run through the process of CPU Scheduling.
- In Non-multiprogrammed system, there are moments when CPU sits idle and does not do any work.
- In Multiprogramming system, CPU will never be idle and keeps on processing.

Time Sharing/Multitasking Systems

- Time Sharing Systems are very similar to Multiprogramming batch systems. In fact time sharing systems are an extension of multiprogramming systems.
- Also known as multitasking; is logical extension in which CPU switches jobs so frequently that users can interact with each job while it is running, creating interactive computing
- In time sharing systems the prime focus is on minimizing the response time, while in multiprogramming the prime focus is to maximize the CPU usage.
- Quick response is achieved by giving fair execution opportunity to each process by two means:
  1. Round-robin scheduling : services all process by turn
  2. Time-slicing: prevents a process from using too much CPU time



Time-sharing is a technique which enables many people, located at various terminals, to use a particular computer system at the same time.

TIME SHARING SYSTEMS

Advantages

- Provides the advantage of quick response.
- Avoids duplication of software.
- Reduces CPU idle time.

Real Time Operating System

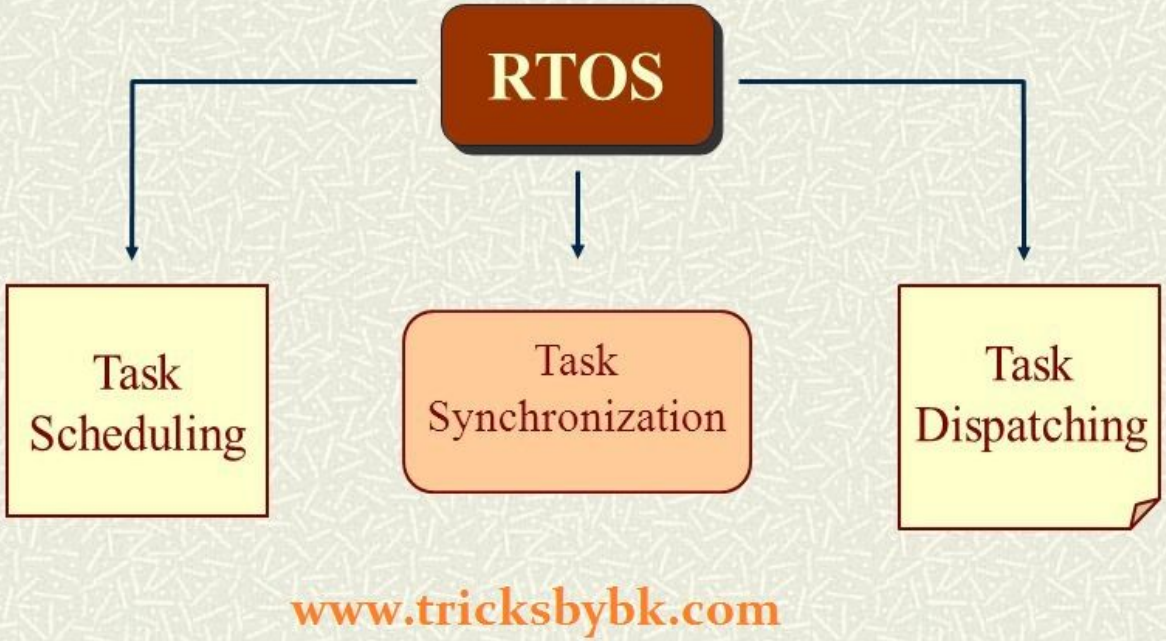
- Used to implement a computer application for controlling or tracking of real-world activities.
- Application needs to complete its computational tasks in a timely manner.
- It is defined as an operating system known to give maximum time for each of the critical operations that it performs, like OS calls and interrupt handling.
- A real-time operating system must have well-defined, fixed time constraints, otherwise the system will fail.

Examples of RTOS

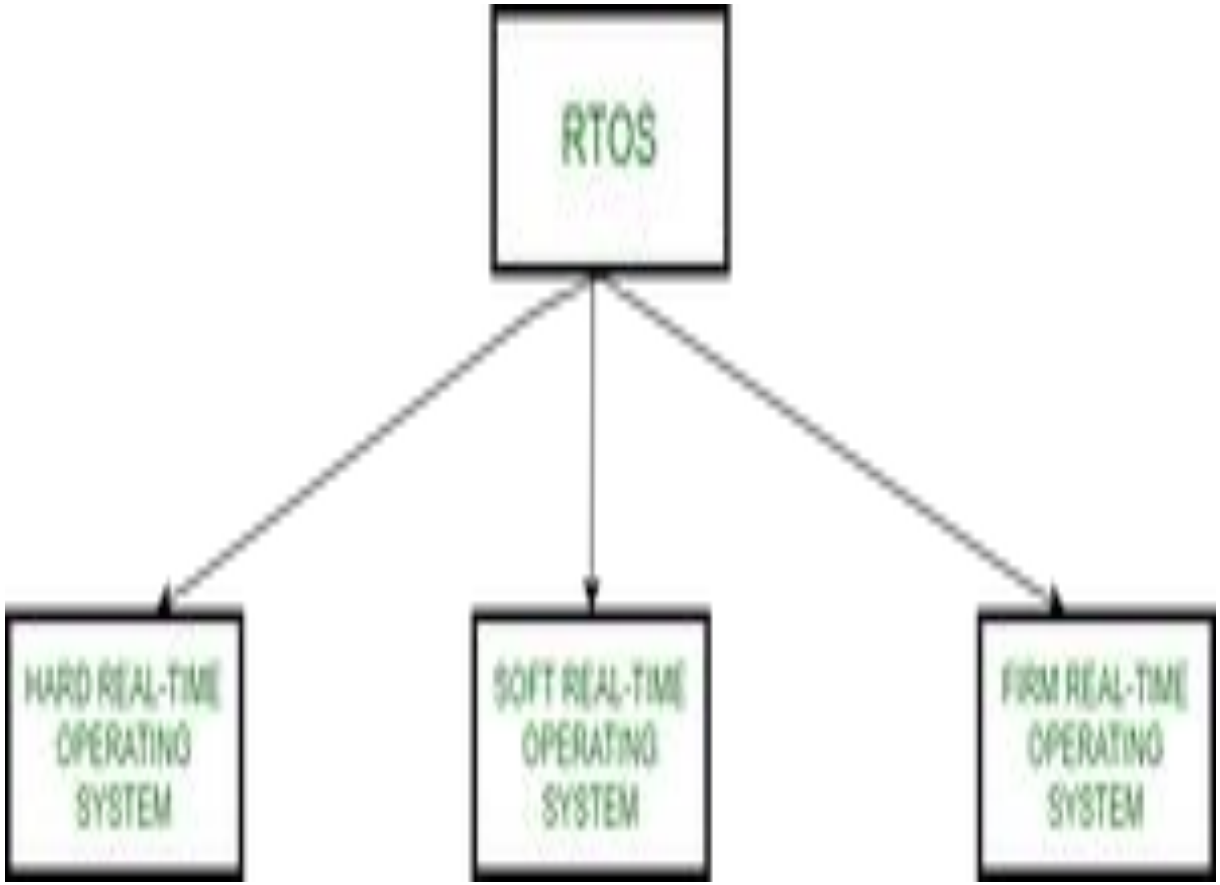
- Embedded systems
- Industrial control systems
- Robots
- Air traffic control systems, etc.,

Real-Time Operating Systems

Overview



- **Real-time** systems are systems in which **correctness** of the system depends **not only** on the **logical result** of computation, **but also** on the **time** at which the results are produced.


RTOS
HARD REAL-TIME OPERATING SYSTEM
SOFT REAL-TIME OPERATING SYSTEM
FIRM REAL-TIME OPERATING SYSTEM

Two types of real-time OS

1. Hard Real-Time Operating Systems
2. Soft Real-Time Operating Systems.

- The RTOS which guarantees the maximum time for critical operations and complete them on time are referred to as Hard Real-Time Operating Systems.
- Delays are bounded
- Secon storage limited,or absent data stored in RAM
- A hard real time system is very restrictive and safety is critical.

Example

1. Air Traffic Control
2. Medical System: Telesurgery

Soft Real-Time Operating Systems.

- The RTOS that can only guarantee a maximum of the time, i.e. the critical task will get priority over other tasks, but no assurity of completing it in a defined time. These systems are referred to as Soft Real-Time Operating Systems.
- Soft real time system is a system whose operation is degrade if results are not produce according to the specified timing requirement.
- A Soft real time system is less restrictive and safety is not critical.

For example

- Computer games
- Virtual reality

System Call

- A system call is a way for a user program to interface with the operating system.
- The program requests several services, and the OS responds by invoking a series of system calls to satisfy the request.
- A system call can be written in assembly language or a high-level language like C or Pascal.
- System calls are predefined functions that the operating system may directly invoke if a high-level language is used.

What is a System Call?

- A system call is a method for a computer program to request a service from the kernel of the operating system on which it is running.
- A system call is a method of interacting with the operating system via programs.
- A system call is a request from computer software to an operating system's kernel.

- The Application Program Interface (API) connects the operating system's functions to user programs. It acts as a link between the operating system and a process, allowing user-level programs to request operating system services.
- The kernel system can only be accessed using system calls. System calls are required for any programs that use resources.

Why do you need system calls in Operating System?

- It is must require when a file system wants to create or delete a file.
- Network connections require the system calls to sending and receiving data packets.
- If you want to read or write a file, you need to system calls.
- If you want to access hardware devices, including a printer, scanner, you need a system call.
- System calls are used to create and manage new processes.

System Call

- System call provide an interface to the OS services
- Computer work in two modes:
  - Kernel mode
  - User Mode (safer mode)
- If user need to access some resource, makes a call that switches from user mode to kernel mode (context switch) it is called system call.
- System call is a programmatic way in which a computer program requests a service from the kernel of OS.
- These call are generally available as routines written in C or C++
- Example: Copy one file content to another file.

System Calls: Examples

System Calls are routine services of OS.

Examples

- Create: OS creates a new process with the specified attributes
- Delete: destroy the designated process
- Abort: terminate the process forcibly
- Fork/join: split and merge sequence of instruction
- Suspend: suspended indefinitely
- Resume: resume the target process which is presumably suspended.
- Delay: suspended for a specified time period
- Level of privilege, priority, size, data area, access rights
- Get_attributes
- Change priority

System Call Categories

1. Process control:
2. File manipulation:
3. Device manipulation:
4. Information Maintenance:
5. Communications:

1. Process control System Call

Used in controlling the processes

- end, abort: Running program needs to halt execution normally or abnormally
- load, execute: Process executing one program may need to load and execute another program
- Create process, terminate process:
- Get process attributes, set process attributes (priority, execution time)
- wait event, signal event,
- allocate and free memory

2. File manipulation System call

- Create file, delete file: Requires file name and attributes (type, protection)
- open, close:
- read, write:
- get file attributes, set file attributes

  (Name, type, location, size….)

3. Device manipulation System call

To acquire additional resources needed for a running process. System call manages resource access

- Request device, release device,
- read, write,
- get device attributes, set device attributes,
- logically attach or detach devices (OS understand the device is attached/detached with the system)

4. Information Maintenance System call

Maintaining the information of the system

Transferring information between user program and OS

- get time or date, set time or date
- get system data, set system data
- get process, file, device attributes
- set process, file, device attributes

5. Communications System Call

Used to make Communication between different processes

Two communication models of communication

1. Message-passing model:
   - Information exchanged through interprocess communication
   - Useful for small data transfer.
   - Easy to implement
2. Shared-memory model: Two or more process exchange information by reading and writing data in shared memory area
   - create, delete communication connection
   - send, receive messages
   - transfer status information
   - attach or detach remote devices

System Call Implementation

- In most systems, system calls can only be made from userspace processes, while in some systems, OS/360 and successors for example, privileged system code also issues system calls.
- OS executes at the highest level of privilege, and allows applications to request services via system calls, which are often initiated via interrupts.
- An interrupt automatically puts the CPU into some elevated privilege level, and then passes control to the kernel, which determines whether the calling program should be granted the requested service.
- If service is granted, kernel executes a specific set of instructions to which calling program has no direct control
- Returns privilege level to that of the calling program, and then returns control to the calling program.

Typical System Call Implementations

- RISC Processor:
  - set up some register with the system call number needed, and execute the software interrupt (trap).
  - transfer control to the operating system kernel.
- CISC Processor:
  - instruction set contains the instructions SYSCALL/SYSRET and SYSENTER/SYSEXIT (AMD and Intel).
- "fast" control transfer instructions that are designed to quickly transfer control to the kernel for a system call without the overhead of an interrupt.
- Linux 2.5
- formerly used the INT instruction, where the system call number was placed in the EAX register before interrupt 0x80 was executed

Typical System Call Implementations

- Multics
  - An older mechanism is the call gate;
  - It allows a program to call a kernel function directly using a safe control transfer mechanism, which the OS sets up in advance
- IA-64 architecture
  - EPC (Enter Privileged Code) instruction is used.
  - The first eight system call arguments are passed in registers, and the rest are passed on the stack.
- IBM System/360 mainframe family
  - Supervisor Call instruction (SVC), with the number in the instruction rather than in a register, implements a system call for most of[1] IBM's OS, and for all system calls in Linux.
  - In IBM's OS, the Program Call (PC) instruction is used for newer facilities. In particular, PC is used when the caller might be in Service Request Block (SRB) mode

System Programs

- System programs provide a convenient environment for program development and execution.
- For example, a compiler is a complex system program.

Categories:

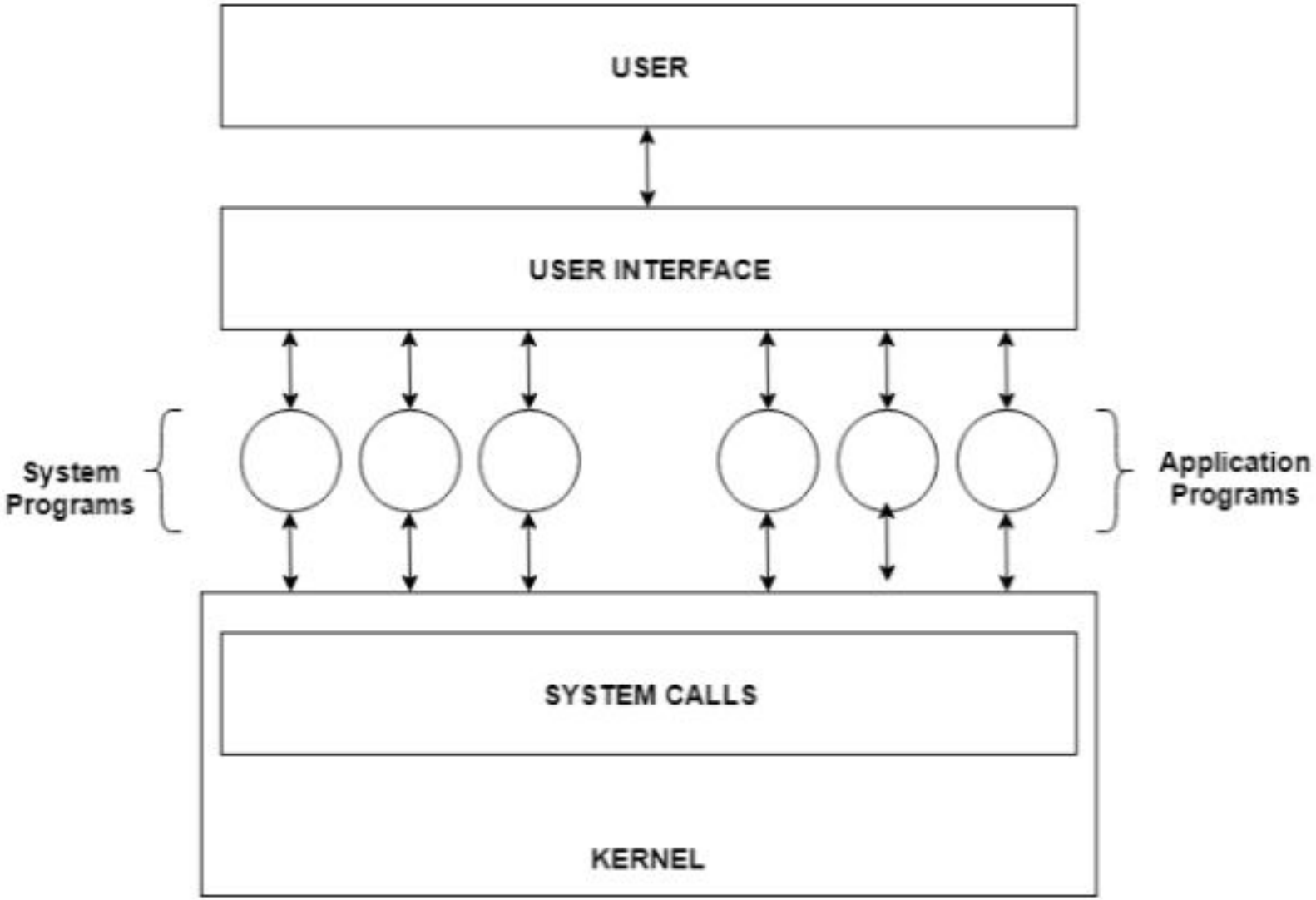
- File management: create, delete, copy, rename, print, dump, list
- Status information: no. of users, available memory or disk space
- File modification: modify content of file using text editors
- Programming language support: compilers, assemblers, interpreters for C, C++, Java etc.
- Program loading and execution: provide loaders and linkers
- Communications: provide virtual connection among processes, users and computers

- Most users' view of OS is defined by system programs, not the actual system calls.

System Calls

- Provide interface between a running program and OS.
  - Generally available as assembly-language instructions.
  - Some system allow system calls directly from HLL programs resemble predefined functions or subroutine calls
- Three general methods are used to pass parameters between a running program and OS.
  - Pass parameters in *registers*.
  - Store the parameters in a table in memory, and the table address is passed as a parameter in a register.
  - *Push* (store) the parameters onto the *stack* by the program, and *pop* off the stack by OS.

Operating system hierarchy



System Call Vs System Program

- User program receives OS services through system calls while system program provide a convenient environment for program development and execution.
- System calls, are the calls made by applications or the processors for a particular execution of a code block;
- System Program is the program that is run in the kernel space. System call is the API exposed from kernel space to user space.
- So, your program running in user space calls to system program via API which is called as system call.
- These are the programs that are used and required to run the system - machine, input output devices and other connected peripherals. They are also known as System softwares.

OS Structure

Structure of OS concerns the nature of OS core (Kernel) and other parts of OS and their interaction.

1. Monolithic structure:
   - OS formed a single software layer between user and the bare machine.
   - User interface was provided by command interpreter.
   - Both command interpreter and user processes invoked OS functionalities and services through system calls
   - Poor portability, high cost of maintenance and enhancement.
2. Layered structure: Structuring OS into no. of layers
3. Kernel-based structure:
   - Small portion of OS code constitutes Kernel
   - Portability increased
4. Microkernel-based OS structure
5. Modular structure

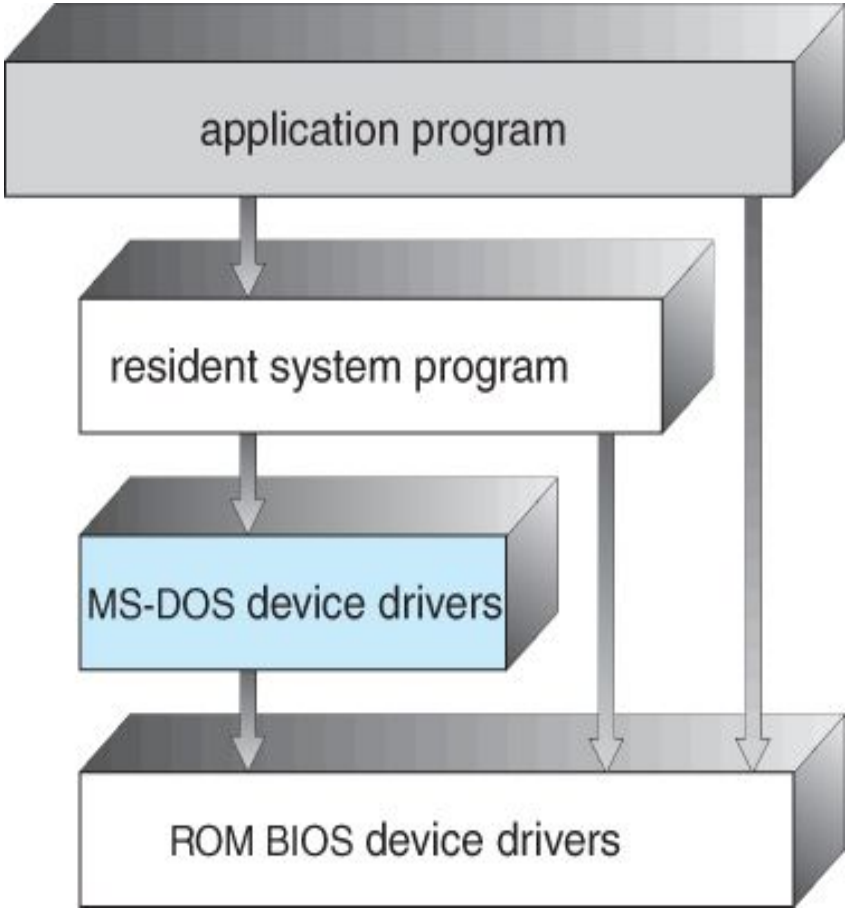
System Structure – Simple Approach

A small, simple and limited systems that do not have a well defined structure

All layers has direct access to base hardware.

Example

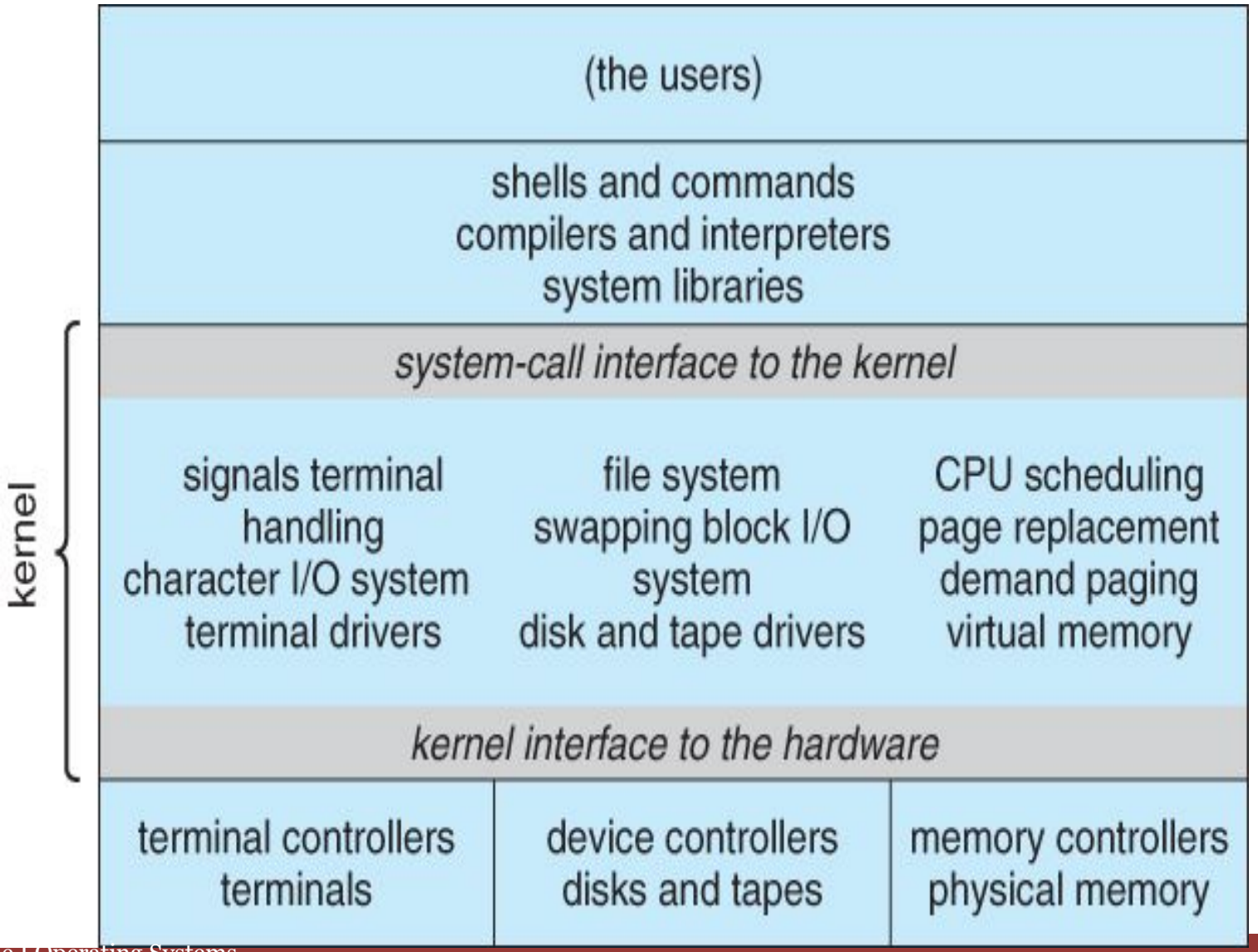
- MS-DOS – written to provide most functionality in least space
  - not divided into modules
  - Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated



MS-DOS Layer structure

1. Monolithic Structure (UNIX Structure)

Too many functions packed into one level called kernel



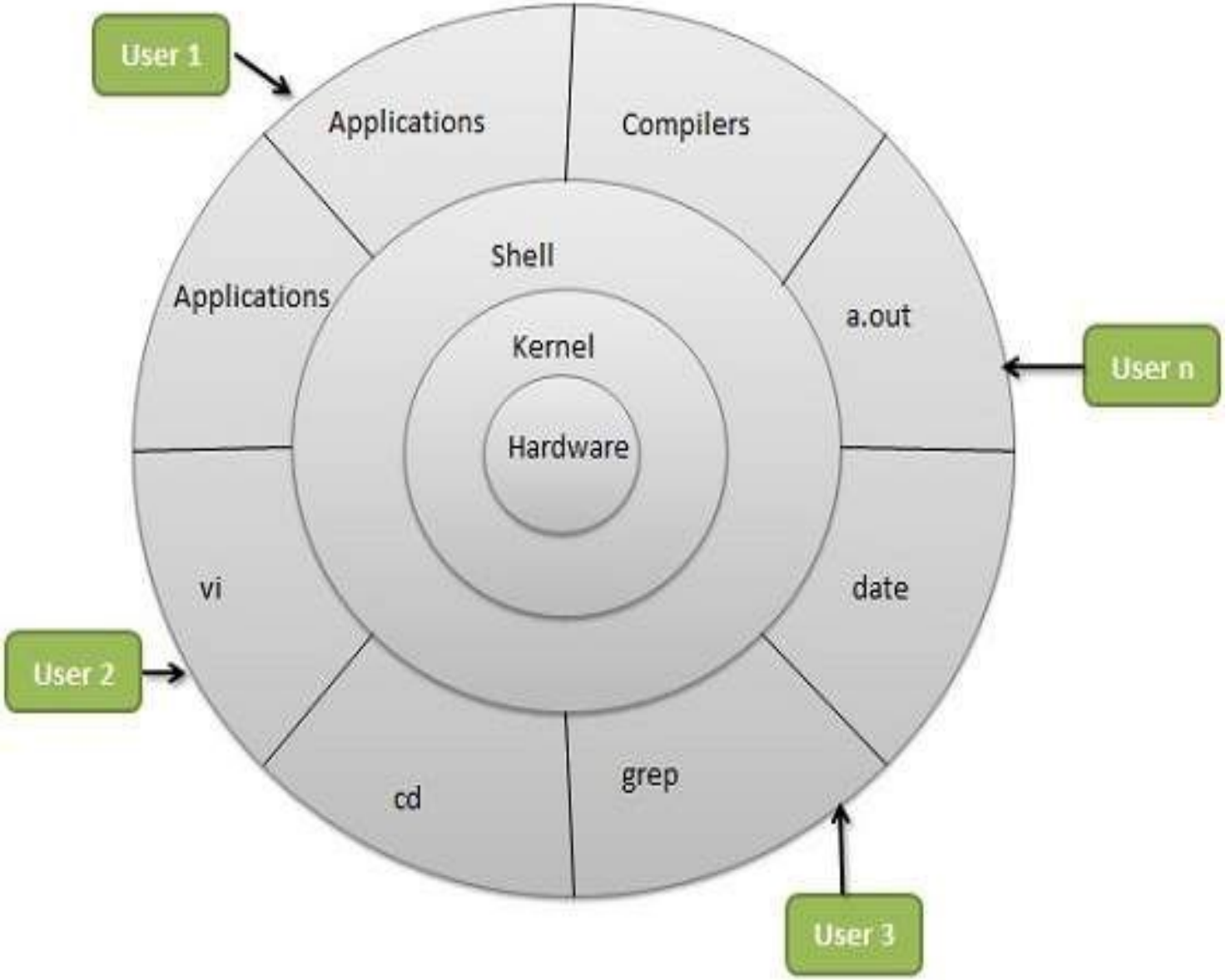
UNIX System Structure

- UNIX – limited by hardware functionality, the original UNIX OS had limited structuring. The UNIX OS consists of two separable parts.
  - Systems programs: use kernel supported system calls, provide functions such as compilation, file manipulation.
  - The kernel: separated into interfaces and device drivers
    - Consists of everything below the system-call interface and above the physical hardware
    - Provides the file system, CPU scheduling, memory management, and other OS functions through system call; a large number of functions for one level.

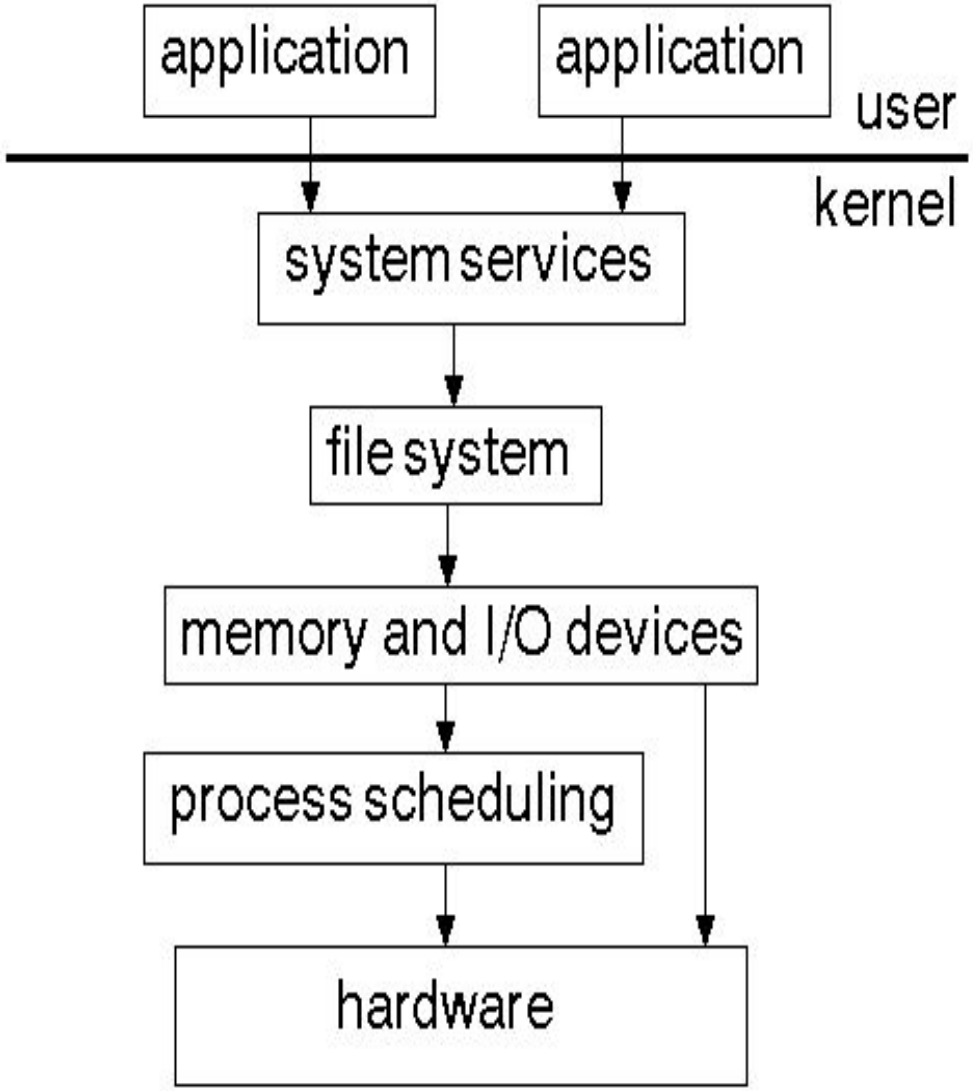
2. System Structure – Layered Approach

- The OS is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers.
- Design and implementation of system is simplified.
- Simplifies debugging and system verification
- Each layer hides the data structures, operations and hardware from higher-level layers (ensures protection)
- Disadvantages:
- Difficulty in deciding the layer position
- Slow in getting services (system calls has to pass to each lower layer)

An Operating System Layer



Layered Model



Advantages

- Simplicity of Construction
- Debugging
- With modularity, layers are selected such that each layer uses functions (operations) and services of only lower-level layers. This simplifies debugging and system verification.
- 1st layer can be debugged without any concern from the rest of the system since layer 1 uses only the basic hardware to implement its functions.
- The correct functioning of layer 1 can be assumed while debugging layer 2 and so on…
- If an error is found during the debugging of a particular layer, the error must be on that layer since the layers below are already debugged,
- Thus the design and implementation of the system are simplified

Disadvantages

- Proper definition of layers is difficult
- Since a layer can use only lower-level layers, careful planning is necessary.
- Eg: The device driver for the disk space and virtual memory should be at a lower level than the memory management routines since memory management requires the ability to use the disk space.
- Less efficient than other types.
- When a user program executes an I/O operation, it executes a system call, which calls the memory management layer, which in turn calls the CPU scheduling layer, which is then passed to the hardware.
- At each layer, the parameters may be modified, data may need to be passed etc.
- Each layer adds overhead to the system call . Hence system calls take longer time than ones in non-layered system.

4. Kernel

- It is a bridge between applications and the actual data processing done at the hardware level.
- It manages operations of computer and hardware - most notably memory and CPU time.
- Manages system's resources (the communication between hardware and software components)

There are two types of kernels:

- A microkernel, which only contains basic functionality;
- A monolithic kernel, which contains entire OS.

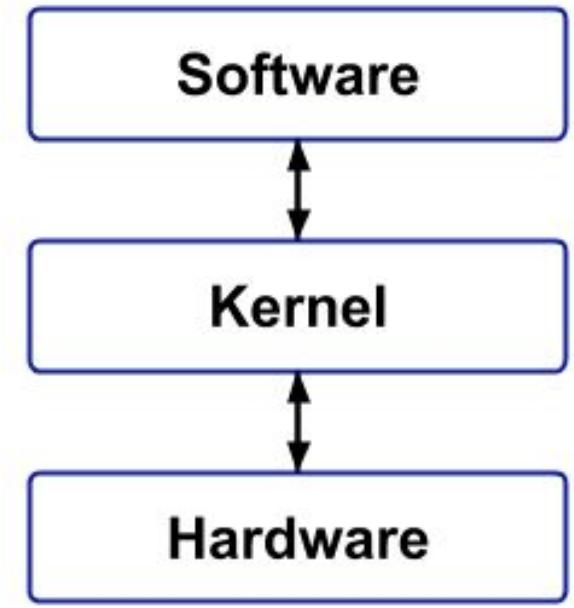
Functions of kernel

- Access Computer resource
- Resource Management
- Memory Management
- Device Management

Monolithic kernel

- Monolithic Kernels are those Kernels where the user services and the kernel services are implemented in the same memory space i.e. different memory for user services and kernel services are not used in this case.
- By doing so, the size of the Kernel is increased and this, in turn, increases the size of the Operating System. As there is no separate User Space and Kernel Space, so the execution of the process will be faster in Monolithic Kernels.

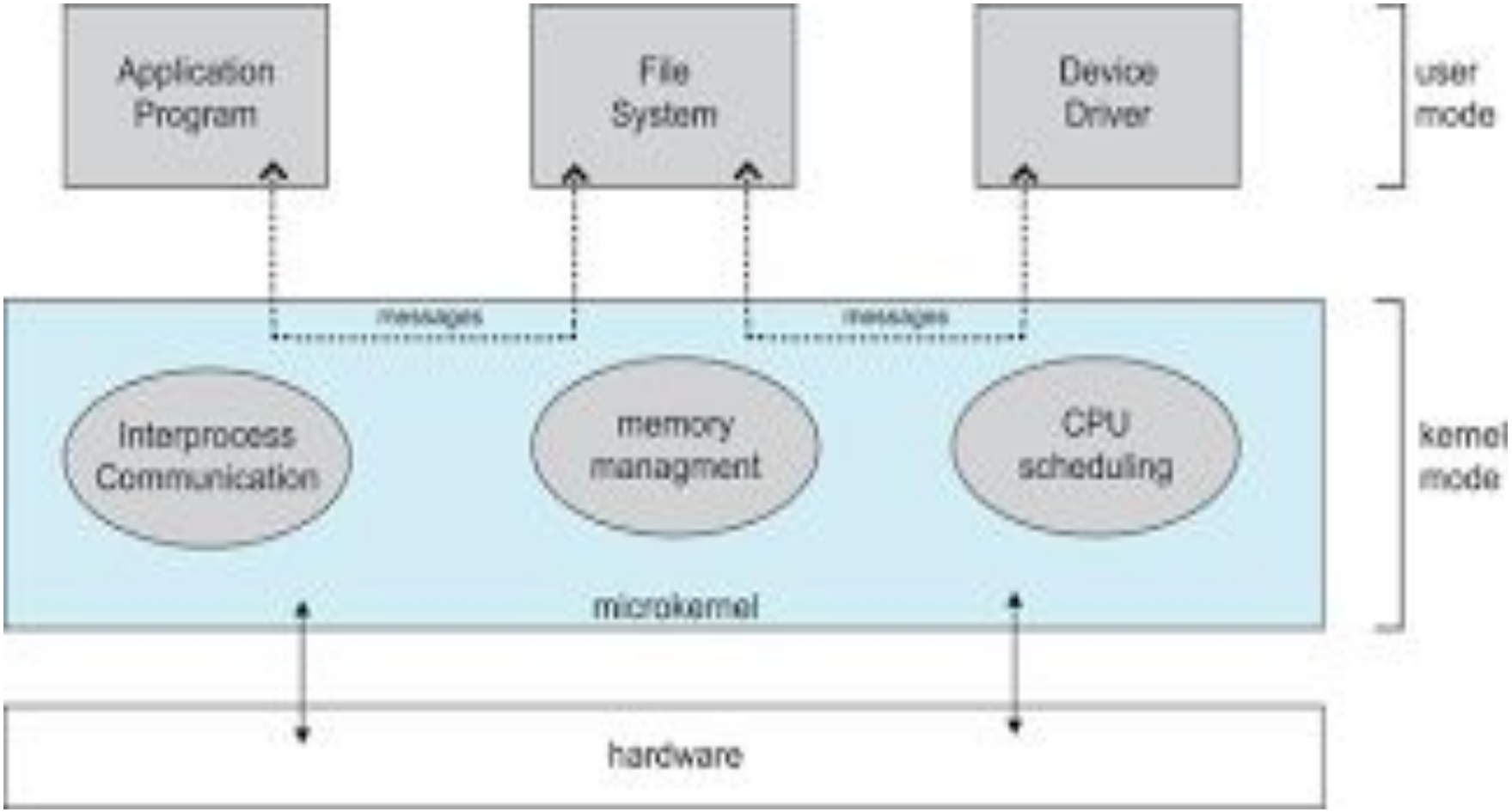
Monolithic kernel



- **Advantages:**
- It provides CPU scheduling, memory scheduling, file management through System calls only.
- Execution of the process is fast because there is no separate memory space for user and kernel.
- **Disadvantages:**
- If any service fails, then it leads to system failure.
- If new services are to be added then the entire Operating System needs to be modified.

Microkernel (Layered)

- To reduce kernel complexity, nonessential components removed from the kernel and implement as system and user-level programs; results smaller kernel.
- Main function is to provide communication facility between client program and various services running in user space as message passing.
- Microkernels provide minimal process and memory management, in addition to communication facility.
- Benefit is the ease of extending the OS i.e. all new services are added to user space, hence no kernel modification required.
- Easier to port one hardware design to another.
- Provides more security and reliability; if a service fails rest of the OS remain untouched.
- Performance issue


Application Program
File System
Device Driver
user mode
messages
messages
Interprocess Communication
memory managment
CPU scheduling
kernel mode
microkernel
hardware

- Communication provided through message passing.
- Eg: If the client pgm needs to access a file, it must interact with the file server.
- The client program and service never interact directly, they communicate indirectly by exchanging messages with the microkernel.
- Disadvantage: The performance of micro kernels can suffer due to increase in system-function overhead.

Advantage

(i) Extension of OS is easier.

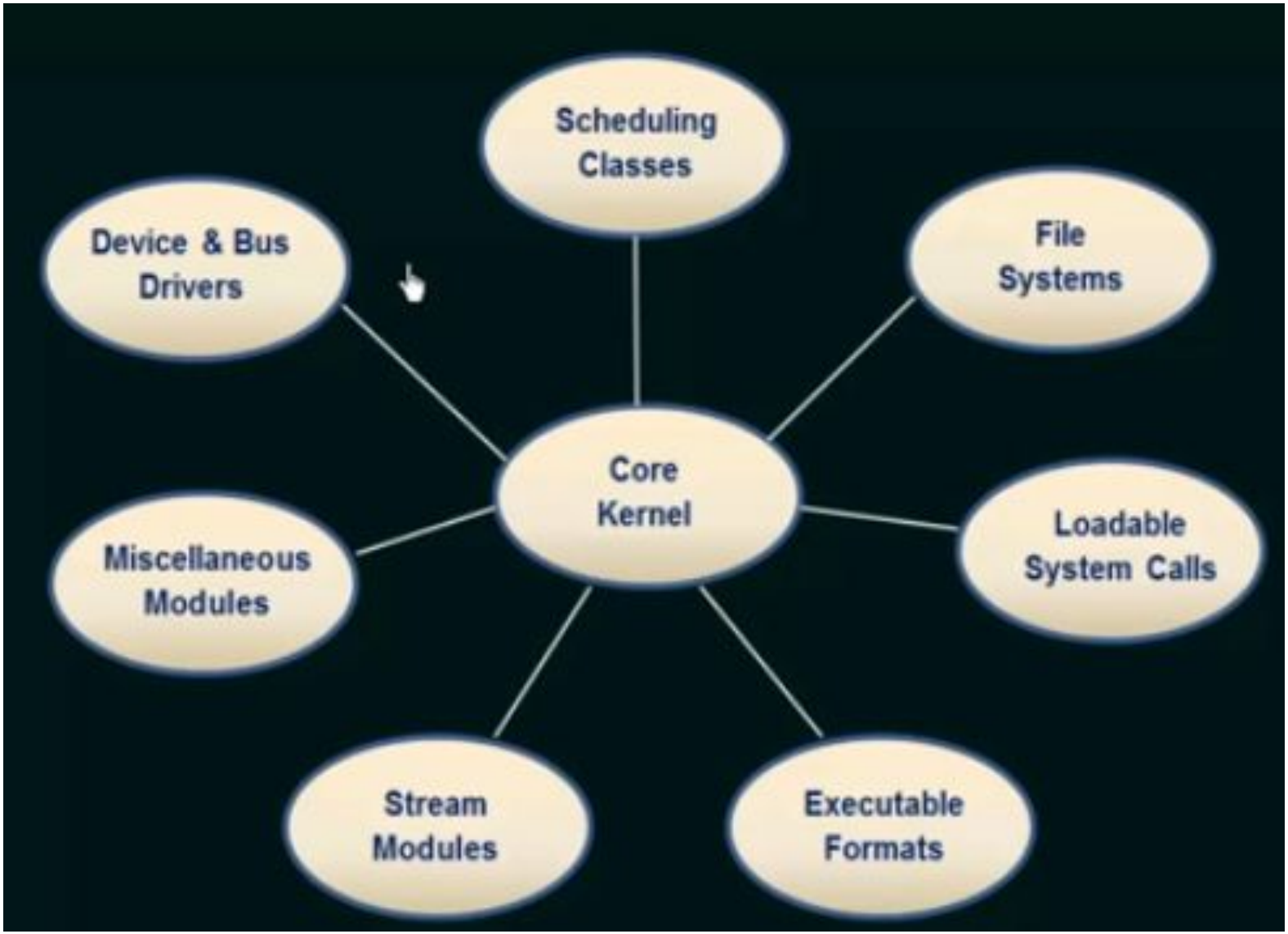
- All new services are added to the user space and do not require modification to the kernel.
- If the kernel needs to be modified, the microkernel tends to be small. So changes made will also be small.
- The kernel is easy to port from one hardware design to another.

(ii) Microkernel provides more security and reliability. Since most services are running as user process rather than kernel process.

- If a service fails, most of the OS remains untouched.

Modular kernel

- Better than layered structure
- Each module can be directly loaded and interacted
- Provides security
- Microkernel requires message passing


Scheduling Classes
Device & Bus Drivers
File Systems
Core Kernel
Miscellaneous Modules
Loadable System Calls
Stream Modules
Executable Formats

- The best current methodology for OS design involves using loadable kernel modules.
- Here, the kernel has a set of core components and links to additional services via modules, either at boot time or at run time.
- This is most common in UNIX as well as Windows.
- The idea is for the kernel to provide core services while other services are implemented dynamically, as the kernel is running.
- Linking services dynamically is preferable to adding new features directly to the kernel. Otherwise the kernel needs to be recompiled every time a change was made.

Shell

- The shell acts as an interface between the user and the kernel. When a user logs in, the login program checks the username and password, and then starts another program called the shell.
- OS shells use either a command-line interface (CLI) or GUI, depending on a computer's role and particular operation.
- It interprets commands the user types in and arranges for them to be carried out.
- It is named a shell because it is the outermost layer around the OS kernel.

CLI shell and GUI shell

CLI shells

- Uses alphanumeric characters typed on a keyboard to provide instructions and data to the operating system, interactively.
- Require user to be familiar with commands and their calling syntax, and to understand concepts about shell-specific scripting language.

Graphical shells

- Allowing for operations such as opening, closing, moving and resizing windows, as well as switching focus between windows
- Place a low burden on beginning computer users, and are characterized as being easy to use. Most GUI-enabled operating systems also provide CLI shells

Summary

- An operating system is software that manages the computer hardware, as well as providing an environment for application programs to run.
- Perhaps the most visible aspect of an operating system is the interface to the computer system it provides to the human user.

References

1. Andrew S. Tanenbaum, “Modern Operating Systems”, Prentice Hall

2. 2. J. L. Peterson and A. Silberschatz , Operating System Concepts, Addison Wesley.


Thank
you

Module 2: CPU Scheduling

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Round Robin Scheduling
- Priority scheduling
- Multiple queues
- Shortest Job First
- Guaranteed scheduling
- Two- level scheduling
- Preemptive scheduling,
- Multiple-Processor Scheduling

Topics

- Process Management: Process concept, Process State, PCB, Operations on processes, Multithreading-Benefits.
- Process Scheduling: Basic concepts, Preemptive Scheduling, Dispatcher, Scheduling criteria, Scheduling Algorithms (FCFS, SJF, Priority scheduling, Round Robin Scheduling, Multi level queue scheduling, Multi level feedback queue scheduling).
- Inter process communication (Shared memory, message passing, pipes and socket).

PROCESSES

Definitions

PROCESS CONCEPT:

- ✔ Program is a group of instruction whereas process is the activity.
- ✔ A **program** is passive; such as the contents of a file stored on disk.
- ✔ A **process** is active; with program counter and a set of resources associated with it.

A process is an instance of a program in execution.

Attributes held by a process include :

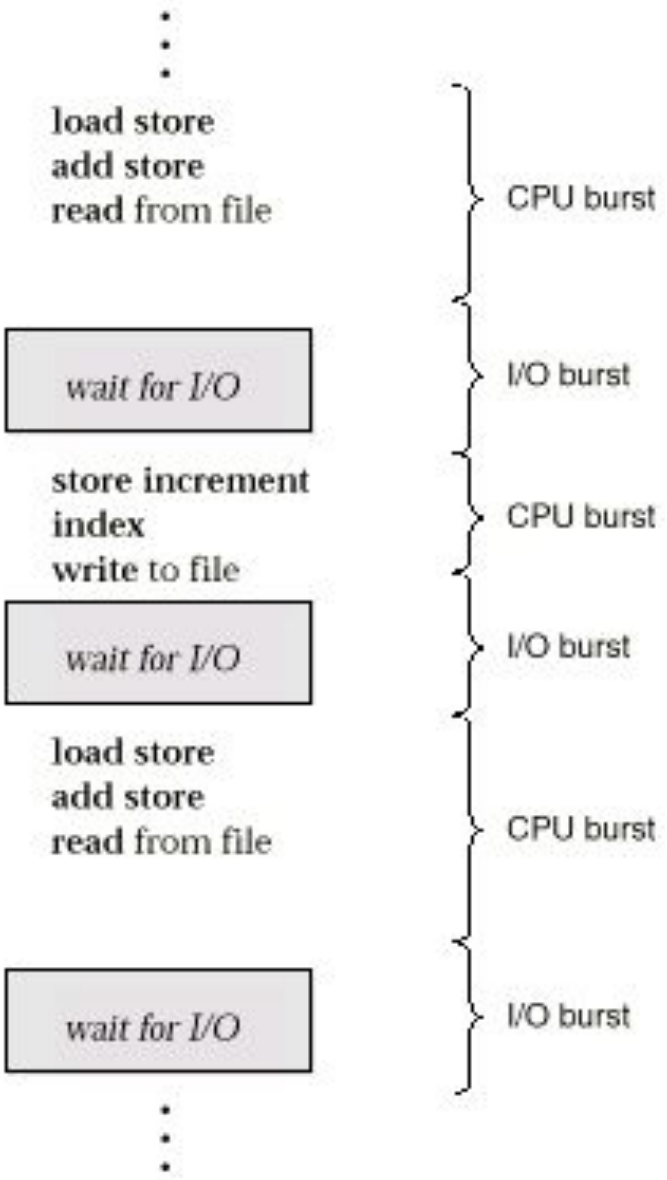
- Hardware state,
- Memory,
- CPU,
- Progress (executing)

WHY HAVE PROCESSES?

- Resource sharing (logical (files) and physical(hardware)
- Computation speedup - taking advantage of multiprogramming
- Modularity for protection.

- Maximum CPU utilization obtained with multiprogramming
- CPU–I/O Burst Cycle – Process execution consists of a cycle of CPU execution and I/O wait.
- CPU burst distribution

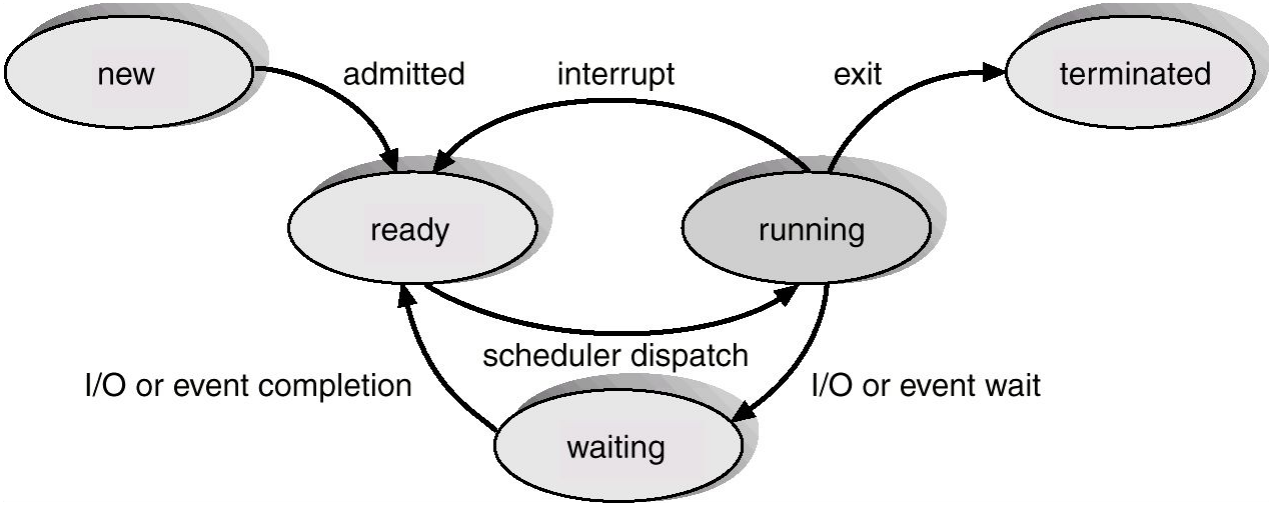
Alternating Sequence of CPU And I/O Bursts



PROCESSES

PROCESS STATES

- **New** The process is just being put together.
- **Running** Instructions being executed. This running process holds the CPU.
- **Waiting** For an event (hardware, human, or another process.)
- **Ready** The process has all needed resources - waiting for CPU only.
- **Suspended** Another process has explicitly told this process to sleep. It will be awakened when a process explicitly awakens it.
- **Terminated** The process is being torn apart.



How a process moves between these states?

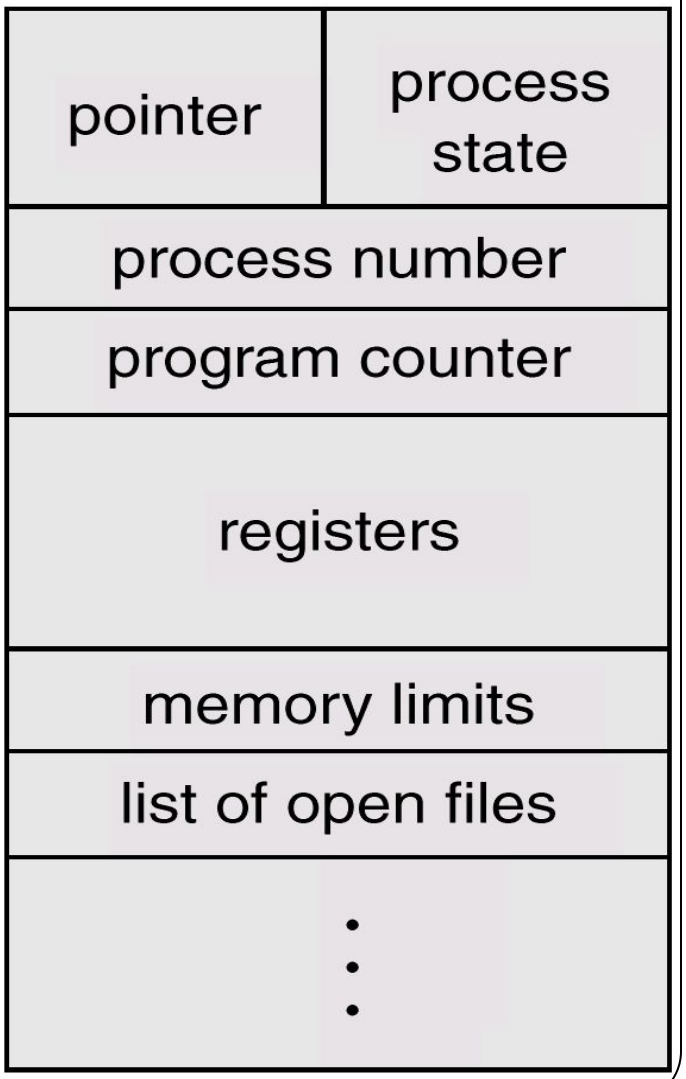
- A process moves from ready to running, when it is dispatched by the scheduler.
- A process moves from running to ready, when it is pre-empted by the scheduler.
- A process moves from running to blocked, when it issues a request for a resource.
- A process moves from blocked to ready, when completion of request is signaled.

Process Control Block (PCB)

Contains information associated with each process:

It's a data structure holding:

- PC, CPU registers,
- memory management information,
- accounting (time used, ID, ...)
- I/O status (such as file resources),
- scheduling data (relative priority, etc.)
- Process State (so running, suspended, etc. is simply a field in the PCB).



Multithreading-Benefits.

- Multithreading is a function of the CPU that permits multiple threads to run independently while sharing the same process resources. A thread is a conscience sequence of instructions that may run in the same parent process as other threads.

- Multithreading allows many parts of a program to run simultaneously. These parts are referred to as threads, and they are lightweight processes that are available within the process. As a result, multithreading increases CPU utilization through multitasking. In multithreading, a computer may execute and process multiple tasks simultaneously.

- Multithreading needs a detailed understanding of these two terms: process and thread. A process is a running program, and a process can also be subdivided into independent units called threads.

- Multithreading allows the execution of multiple parts of a program at the same time.
- These parts are known as threads and are lightweight processes available within the process. So multithreading leads to maximum utilization of the CPU by multitasking.

Benefits of Multithreading

- Responsiveness
- Resource Sharing
- Economy
- Scalability
- Better Communication
- Minimized system resource usage

Disadvantages of Multithreading

- It needs more careful synchronization.
- It can consume a large space of stocks of blocked threads.
- It needs support for thread or process.
- If a parent process has several threads for proper process functioning, the child processes should also be multithreaded because they may be required.
- It imposes context switching overhead.

CPU Scheduler

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state.
  2. Switches from running to ready state.
  3. Switches from waiting to ready.
  4. Terminates.
- Scheduling under 1 and 4 is *nonpreemptive*.
- All other scheduling is *preemptive.*

Scheduling Categories

- CPU Scheduling deals with the problem of deciding which of the processes in ready queue is to be allocated CPU.
- Major division among the several scheduling algorithms is preemptive or non-preemptive discipline.

1. Non-preemptive scheduling

- Once a process has been allocated to the CPU, the CPU cannot be taken away from the process
- Jobs are made to wait by longer jobs, but treatment of all process is fairer.

Algorithms used are:

- First Come First Served (FCFS), Shortest Job First (SJFS)

2. Preemptive scheduling

- Preemption means OS moves a process from running to ready without the process requesting it.
- Algorithm switches to the processing of a new request before completing the processing of the current request.
- Preempted request is put back to the pending request list and resumed again when it get rescheduled.
- Preemptive scheduling is useful in high priority process which requires immediate response.
- Algorithms used are:
- Round Robin Scheduling, Priority based scheduling, Shortest Remaining Time Next scheduling (SRTN)

The decision whether to schedule preemptive or not, depends on the environment and type of application most likely to be supported by OS.

Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - switching context : save the process states and register contents in PCB and reload contents of next program
  - switching to user mode
  - jumping to the proper location in the user program to restart that program
- *Dispatch latency* – time it takes for the dispatcher to stop one process and start another running.
- Context switch: switching of CPU from one process to another

Scheduling Criteria

- CPU utilization – keep the CPU as busy as possible
- Throughput – # of processes that complete their execution per time unit
- Turnaround time – amount of time to execute a particular process; interval from time of submission to time of completion
- Waiting time – amount of time a process has been waiting in the ready queue
- Response time – amount of time it takes from when a request was submitted until the first response is produced, **not** output (for time-sharing environment)

Optimization Criteria

✔ Max CPU utilization ✓ Max throughput

✔ Min turnaround time ✓ Min waiting time

✔ Min response time

$$\text{CPU utilization} = \frac{\text{(processor busy time)}}{\text{(processor busy time + processor idle time)}}$$

Throughput = (no. of process completed)/(time unit)

Turnaround time = t(process completed) – t(process submitted)

Waiting time = Turnaround time – Processing time

Response time = t(first response) – t(submission of request)

1. LONG TERM SCHEDULER: Job scheduler

- Selects processes from disk (spool)
- Run seldom (when job comes into memory)
- Controls degree of multiprogramming
- Tries to balance arrival and departure rate through an appropriate job mix (I/O bound, CPU bound).

SHORT TERM SCHEDULER

Called CPU scheduler

Selects process from memory (ready queue) for execution

Contains three functions:

- Code to remove a process from the processor at the end of its run.
  a)Process may go to ready queue or to a wait state.

- Code to put a process on the ready queue –
  a)Process must be ready to run.
  b)Process placed on queue based on priority.

- Code to take a process off the ready queue and run that process (also called **dispatcher**).
  a) Always takes the first process on the queue (no intelligence required)
  b) Places the process on the processor.

This code runs frequently and so should be as short as possible

MEDIUM TERM SCHEDULER

- Mixture of CPU and memory resource management.
- Swap out/in jobs to improve mix and to get memory.
- Controls change of priority.

Categories of Scheduling Algorithms

- Batch
- Interactive
- Real time

Scheduling Algorithm Goals

All systems

- Fairness - giving each process a fair share of the CPU
- Policy enforcement - seeing that stated policy is carried out
- Balance - keeping all parts of the system busy

Batch systems

- Throughput - maximize jobs per hour
- Turnaround time - minimize time between submission and termination
- CPU utilization - keep the CPU busy all the time

Interactive systems

- Response time - respond to requests quickly
- Proportionality - meet users' expectations

Real-time systems

- Meeting deadlines - avoid losing data
- Predictability - avoid quality degradation in multimedia systems

Some goals of the scheduling algorithm under different circumstances.

Scheduling in Batch Systems

- First-come first-served
- Shortest job first
- Shortest remaining Time next

First-Come, First-Served (FCFS) Scheduling

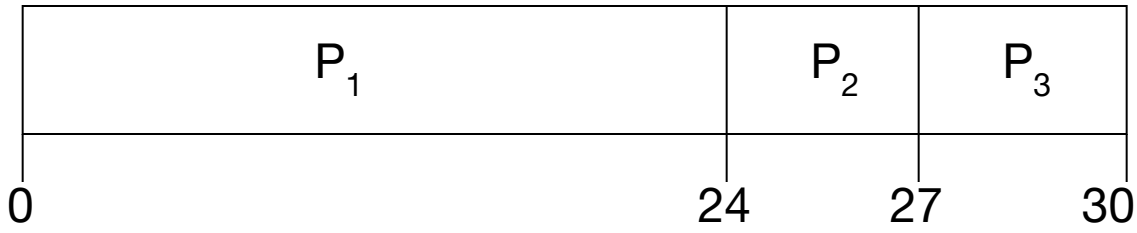
- Example: Process Burst Time

  P_1 24

  P_2 3

  P_3 3

- Suppose that the processes arrive in the order: P_1 , P_2 , P_3
  The Gantt Chart for the schedule is:



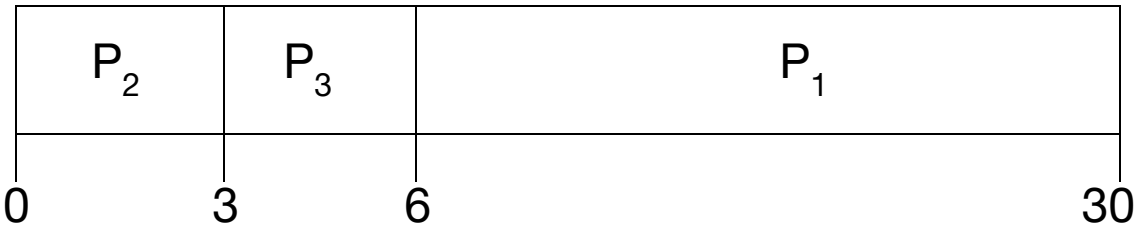
- Waiting time and response time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17
- Turnaround time of P1 = 24; P2 = 27; P3 = 30

FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order

P_2 , P_3 , P_1 .

- The Gantt chart for the schedule is:



- Waiting time and response time for $P_1 = 6$; $P_2 = 0$; $P_3 = 3$
- Average waiting time: (6 + 0 + 3)/3 = 3
- Turnaround time of P1 = 3, P2 = 6; P3 = 30
- Much better than previous case.

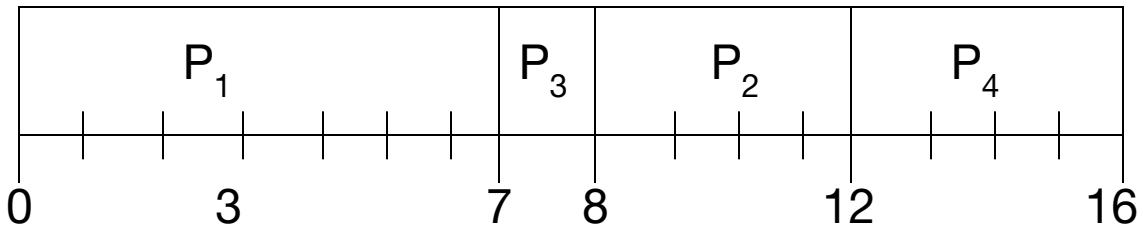
Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time.
- Two schemes:
  - nonpreemptive – once CPU given to the process it cannot be preempted until completes its CPU burst.
  - Preemptive – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is know as the Shortest-Remaining-Time-First (SRTF).
- SJF is optimal – gives minimum average waiting time for a given set of processes.

Example of Non-Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (non-preemptive)
- Waiting time and response time of P1 = 0; P2 = 8 – 2 = 6; P3 = 7 – 4 = 3



- Average waiting time = (0 + 6 + 3 + 7)/4 = 4

Turnaround time = Completion time – arrival time

- Turnaround time (P1) = 7 - 0 = 7
- Turnaround time (P2) = 12 – 2 = 10
- Turnaround time (P3) = 8 – 4 = 4
- Turnaround time (P4) = 16 – 5 = 11

Scheduling in Interactive Systems

- Round-robin scheduling
- Priority scheduling
- Multiple queues
- Shortest process next
- Guaranteed scheduling
- Lottery scheduling
- Fair-share scheduling

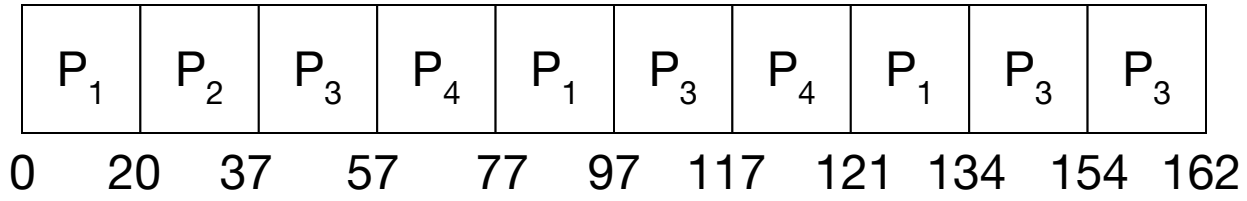
Round Robin (RR)

- Each process gets a small unit of CPU time (*time quantum*), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are *n* processes in the ready queue and the time quantum is *q*, then each process gets 1/*n* of the CPU time in chunks of at most *q* time units at once. No process waits more than (*n*-1)*q* time units.
- Performance
  - *q* large ⇒ FIFO
  - *q* small ⇒ *q* must be large with respect to context switch, otherwise overhead is too high.

Example: RR with Time Quantum = 20

Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

- The Gantt chart is:



- Typically, higher average turnaround than SJF, but better *response*.
- Assume all process arrived at time zero

Turnaround time = Completion time – arrival time

- Turnaround time (P1) = 134 – 0 = 134
- Turnaround time (P2) = 37 – 0 = 37
- Turnaround time (P3) = 162 – 0 = 162
- Turnaround time (P4) = 121 – 0 = 121

Waiting time = Turnaround time – Burst Time

- Waiting time (P1) = 134 – 53 = 81
- Waiting time (P2) = 37 – 17 = 20
- Waiting time (P3) = 162 – 68 = 94

Response Time = First execution start time – arrival time

- Response Time (P1) = 0 – 0 = 0
- Response Time (P2) = 20 – 0 =20
- Response Time (P3) = 37 – 0 = 37
- Response Time (P4) = 57 – 0 = 57

What is Preemptive Scheduling?

- Preemptive Scheduling is a scheduling method where the tasks are mostly assigned with their priorities. Sometimes it is important to run a task with a higher priority before another lower priority task, even if the lower priority task is still running.

- At that time, the lower priority task holds for some time and resumes when the higher priority task finishes its execution.

What is Non- Preemptive Scheduling?

- In this type of scheduling method, the CPU has been allocated to a specific process. The process that keeps the CPU busy will release the CPU either by switching context or terminating.

- It is the only method that can be used for various hardware platforms. That's because it doesn't need specialized hardware (for example, a timer) like preemptive Scheduling.

- Non-Preemptive Scheduling occurs when a process voluntarily enters the wait state or terminates.

Advantages of Preemptive Scheduling

- Preemptive scheduling method is more robust, approach so one process cannot monopolize the CPU
- Choice of running task reconsidered after each interruption.
- Each event cause interruption of running tasks
- The OS makes sure that CPU usage is the same by all running process.
- In this, the usage of CPU is the same, i.e., all the running processes will make use of CPU equally.
- This scheduling method also improvises the average response time.

Advantages of Non-preemptive Scheduling

- Offers low scheduling overhead
- Tends to offer high throughput
- It is conceptually very simple method
- Less computational resources need for Scheduling

Disadvantages of Preemptive Scheduling

- Need limited computational resources for Scheduling
- Takes a higher time by the scheduler to suspend the running task, switch the context, and dispatch the new incoming task.
- The process which has low priority needs to wait for a longer time if some high priority processes arrive continuously

Disadvantages of Non-Preemptive Scheduling

- It can lead to starvation especially for those real-time tasks
- Bugs can cause a machine to freeze up
- It can make real-time and priority Scheduling difficult
- Poor response time for processes