Unit-03

Deadlock and Memory Management

Unit-03/Lecture-01

The Deadlock Problem

1. A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set.

2. Example

a. System has 2 tape drives.

b. P1 and P2 each hold one tape drive and each needs another one.

3. Example

a. semaphores A and B, initialized to 1

P0 P1
wait (A); wait(B)
wait (B); wait(A)

Bridge Crossing Example

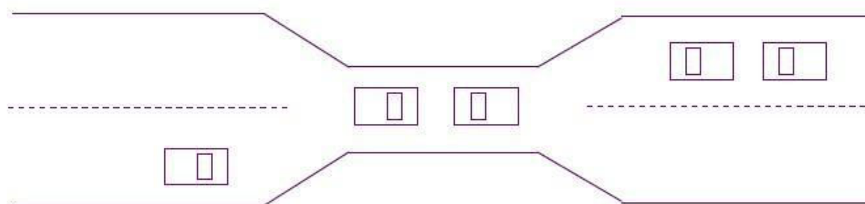
1. Traffic only in one direction.

2. Each section of a bridge can be viewed as a resource.

3. If a deadlock occurs, it can be resolved if one car backs up (preempt resources and rollback).

4. Several cars may have to be backed up if a deadlock occurs.

5. Starvation is possible.



System Model

1. Resource types R1, R2, . . ., Rm

2. CPU cycles, memory space, I/O devices

3. Each resource type Ri has Wi instances.

we dont take any liability for the notes correctness. http://www.rgpvonline.com

4. Each process utilizes a resource as follows:

a. request

b. use

c. release

Deadlock Characterization

Deadlock can arise if four conditions hold simultaneously.

1. **Mutual exclusion:** only one process at a time can use a resource.

2. **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes.

3. **No preemption:** a resource can be released only voluntarily by the process holding it, after that process has completed its task.

4. **Circular wait:** there exists a set {P0, P1, …, P0} of waiting processes such that P0 is waiting for a resource that is held by P1, P1 is waiting for a resource that is held by

P2, …, Pn–1 is waiting for a resource that is held by Pn, and P0 is waiting for a resource that is held by P0.

Resource-Allocation Graph

A set of vertices V and a set of edges E.

1. V is partitioned into two types:
   a. P = {P1, P2, …, Pn}, the set consisting of all the processes in the system.
   b. R = {R1, R2, …, Rm}, the set consisting of all resource types in the system.
2. request edge – directed edge P1 Rj
3. assignment edge – directed edge Rj Pi
4. Process
5. Resource Type with 4 instances
6. Pi requests instance of Rj
7. Pi is holding an instance of Rj

.

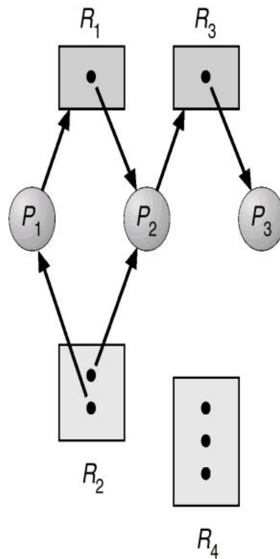
S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	List and explain four necessary conditions simultaneously hold for deadlock?	Dec 2011	10
Q.2	What is Deadlock	Dec 2013	7

we dont take any liability for the notes correctness. http://www.rgpvonline.com

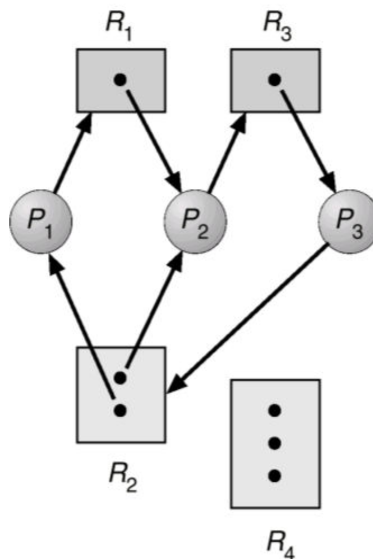
Unit-03/Lecture-02

. **Resource-Allocation Graph (contd.)**

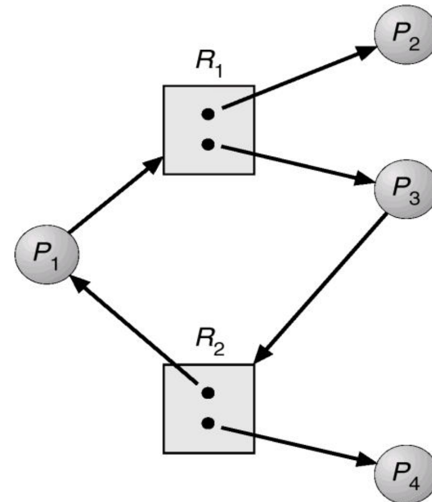
Example of a Resource Allocation Graph



Resource Allocation Graph With A Deadlock



Resource Allocation Graph With A Cycle But No Deadlock



Basic Facts

1. If graph contains no cycles no deadlock.
2. If graph contains a cycle
a. if only one instance per resource type, then deadlock.
b. if several instances per resource type, possibility of deadlock.

Methods for Handling Deadlocks

1. Ensure that the system will never enter a deadlock state.
2. Allow the system to enter a deadlock state and then recover.
3. Ignore the problem and pretend that deadlocks never occur in the system; used by most operating systems, including UNIX.

Deadlock Prevention
Restrain the ways request can be made.

1. **Mutual Exclusion** – not required for sharable resources; must hold for nonsharable resources.
2. **Hold and Wait** – must guarantee that whenever a process requests a resource, it does not hold any other resources.

  a. Require process to request and be allocated all its resources before it begins execution, or allow process to request resources only when the process has none.
  b. Low resource utilization; starvation possible.

we dont take any liability for the notes correctness.

3. **No Preemption** –

If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released.

Preempted resources are added to the list of resources for which the process is waiting.

Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting.

4. **Circular Wait** – impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration.

Deadlock Avoidance

Requires that the system has some additional a priori information available.

1. Simplest and most useful model requires that each process declare the maximum number of resources of each type that it may need.

2. The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition.

3. Resource-allocation state is defined by the number of available and allocated resources, and the maximum demands of the processes.

Safe State

1. When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state.

2. System is in safe state if there exists a safe sequence of all processes.

3. Sequence <P1, P2, …, Pn> is safe if for each Pi, the resources that Pi can still request can be satisfied by currently available resources + resources held by all the Pj, with j<I.

a. If Pi resource needs are not immediately available, then Pi can wait until all Pj have finished.

b. When Pj is finished, Pi can obtain needed resources, execute, return allocated resources, and terminate.

c. When Pi terminates, Pi+1 can obtain its needed resources, and so on.

Basic Facts

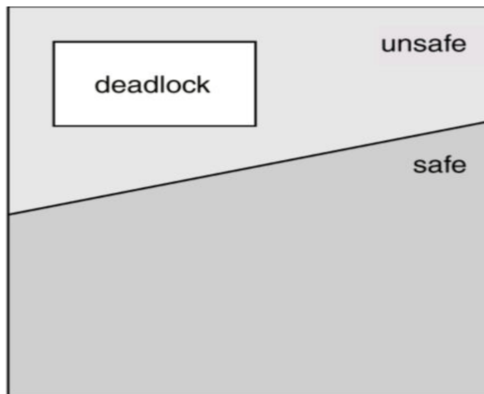
1. If a system is in safe state no deadlocks.

2. If a system is in unsafe state possibility of deadlock.

3. Avoidance ensure that a system will never enter an unsafe state.

we dont take any liability for the notes correctness. http://www.rgpvonline.com

Safe, Unsafe , Deadlock State



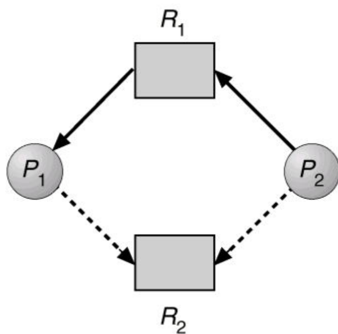
S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	Explain resource allocation graph for deadlock avoidance	Dec 2012	10

Unit-03/Lecture-03

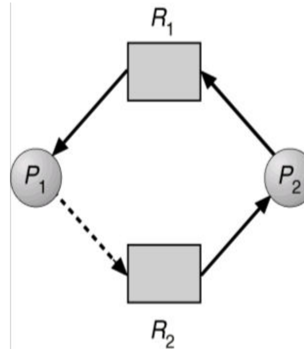
Resource-Allocation Graph Algorithm

1. Claim edge Pi Rj indicated that process Pj may request resource Rj; represented by a dashed line.
2. Claim edge converts to request edge when a process requests a resource.
3. When a resource is released by a process, assignment edge reconverts to a claim edge.
4. Resources must be claimed a priori in the system.

Resource-Allocation Graph For Deadlock Avoidance



Unsafe State In Resource-Allocation Graph



Banker's Algorithm

1. Multiple instances.

2. Each process must a priori claim maximum use.

3. When a process requests a resource it may have to wait.

4. When a process gets all its resources it must return them in a finite amount of time.

Data Structures for the Banker's Algorithm

Let n = number of processes, and m = number of resources types.

1. Available: Vector of length m. If available [j] = k, there are k instances of resource type Rj available.

2. Max: n x m matrix. If Max [i,j] = k, then process Pi may request at most k instances of resource type Rj.

3. Allocation: n x m matrix. If Allocation[i,j] = k then Pi is currently allocated k instances of Rj.

4. Need: n x m matrix. If Need[i,j] = k, then Pi may need k more instances of Rj to complete its task.

we dont take any liability for the notes correctness. http://www.rgpvonline.com

5. Need [i,j] = Max[i,j] – Allocation [i,j].

Safety Algorithm

1.Let Work and Finish be vectors of length m and n, respectively. Initialize:
Work = Available
Finish [i] = false for i - 1,3, …, n.
2.Find and i such that both:
(a) Finish [i] = false
(b) Needi Work
If no such i exists, go to step 4.
3.Work = Work + Allocationi Finish[i] = true go to step 2.
4.If Finish [i] == true for all i, then the system is in a safe state.

Resource-Request Algorithm for Process Pi

Request = request vector for process Pi. If Requesti [j] = k then process Pi wants k instances of resource type Rj.
1.If Requesti Needi go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim.
2.If Requesti Available, go to step 3. Otherwise Pi must wait, since resources are not available.
3.Pretend to allocate requested resources to Pi by modifying the state as follows:
Available = Available = Requesti;
Allocationi = Allocationi + Requesti;
Needi = Needi – Requesti;;

- If safe the resources are allocated to Pi.

- If unsafe Pi must wait, and the old resource-allocation state is restored

Example of Banker's Algorithm

1. 5 processes P0 through P4; 3 resource types A (10 instances), B (5instances, and C (7 instances).

2. Snapshot at time T0:

Allocation Max Available
A B C A B C A B C
P0 0 1 0 7 5 3 3 3 2
P1 2 0 0 3 2 2
P2 3 0 2 9 0 2
P3 2 1 1 2 2 2
P4 0 0 2 4 3 3

3. The content of the matrix. Need is defined to be Max – Allocation.

Need
A B C
P0 7 4 3
P1 1 2 2
P2 6 0 0

P3 0 1 1
P4 4 3 1

4. The system is in a safe state since the sequence < P1, P3, P4, P2, P0> satisfies safety criteria.

Example P1 Request (1,0,2) (Cont.)

1. Check that Request Available (that is, (1,0,2) (3,3,2) true.

Allocation Need Available
A B C A B C A B C
P0 0 1 0 7 4 3 2 3 0
P1 3 0 2 0 2 0
P2 3 0 1 6 0 0
P3 2 1 1 0 1 1
P4 0 0 2 4 3 1

1. Executing safety algorithm shows that sequence <P1, P3, P4, P0, P2> satisfies safety requirement.

2. Can request for (3,3,0) by P4 be granted?

3. Can request for (0,2,0) by P0 be granted?

S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	Describe Bankers Algorithm with example	Dec 2011, Dec 2013	10 7
Q.2	Consider following snapshot Allocation: A B C D — P_0 0 0 1 2; P_1 1 0 0 0; P_2 1 3 5 4; P_3 0 6 3 2; P_4 0 0 1 4 Max: A B C D — P_0 0 0 1 2; P_1 1 7 5 0; P_2 2 3 5 6; P_3 0 6 5 2; P_4 0 6 5 6 Available: A B C D — 1 5 2 0 i) What is the content of the matrix need ii) Is the system in a safe state. iii) If a request from process P_1 arrives from (0, 4, 2, 0) can the request be granted immediately. Answer the above questions using Banker's algorithm.	Dec 2012	14

Unit-03/Lecture-04

Deadlock Detection

1. Allow system to enter deadlock state
2. Detection algorithm
3. Recovery scheme

Single Instance of Each Resource Type

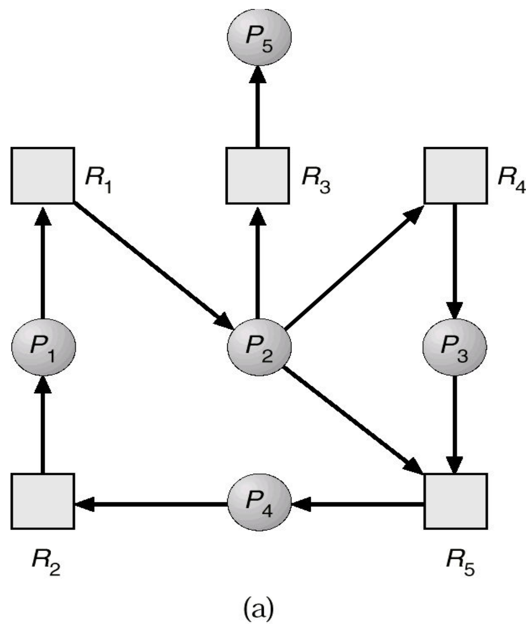
1. Maintain wait-for graph

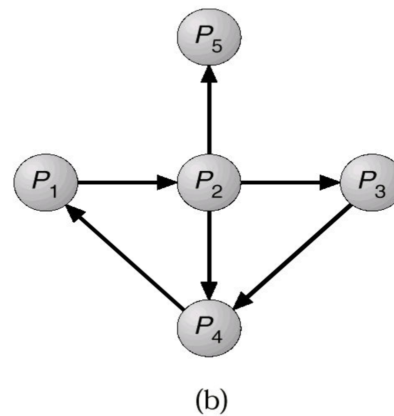
a. Nodes are processes.
b. Pi Pj if Pi is waiting for Pj.

2. Periodically invoke an algorithm that searches for a cycle in the graph.

3. An algorithm to detect a cycle in a graph requires an order of n2 operations, where n is the number of vertices in the graph.

Resource-Allocation Graph and Wait-for Graph





a. Resource-Allocation Graph b.Corresponding wait-for graph

Several Instances of a Resource Type

1. Available: A vector of length m indicates the number of available resources of each type.
2. Allocation: An n x m matrix defines the number of resources of each type currently allocated to each process.
3. Request: An n x m matrix indicates the current request of each process. If Request [ij] = k, then process Pi is requesting k more instances of resource type. Rj.

Detection Algorithm
1. Let Work and Finish be vectors of length m and n, respectively Initialize:
(a) Work = Available
(b) For i = 1,2, …, n, if Allocationi 0, then Finish[i] = false;otherwise, Finish[i] = true.
2. Find an index i such that both:
(a) Finish[i] == false
(b) Requesti Work
If no such i exists, go to step 4.
3. Work = Work + Allocationi Finish[i] = true go to step 2.
4. If Finish[i] == false, for some i, 1 i n, then the system is in deadlock state. Moreover, if Finish[i] == false, then Pi is deadlocked.
Algorithm requires an order of O(m x n2) operations to detect whether the system is in deadlocked state.

Example of Detection Algorithm
1.Five processes P0 through P4; three resource types A (7 instances), B (2 instances), and C (6 instances).

2.Snapshot at time T0:
Allocation Request Available
A B C A B C A B C
P0 0 1 0 0 0 0 0 0 0
P1 2 0 0 2 0 2
P2 3 0 3 0 0 0
P3 2 1 1 1 0 0
P4 0 0 2 0 0 2

3.Sequence <P0, P2, P3, P1, P4> will result in Finish[i] = true for all i.

4.P2 requests an additional instance of type C.
Request
A B C
P0 0 0 0
P1 2 0 1
P2 0 0 1
P3 1 0 0
P4 0 0 2
5.State of system

o Can reclaim resources held by process P0, but insufficient resources to fulfill other processes; requests.

o Deadlock exists, consisting of processes P1, P2, P3, and P4.

Detection-Algorithm Usage
1. When, and how often, to invoke depends on:
a. How often a deadlock is likely to occur?
b. How many processes will need to be rolled back?
i. one for each disjoint cycle

we dont take any liability for the notes correctness. http://www.rgpvonline.com

2. If detection algorithm is invoked arbitrarily, there may be many cycles in the resource graph and so we would not be able to tell which of the many deadlocked processes “caused” the deadlock.

Recovery from Deadlock: Process Termination

1. Abort all deadlocked processes.
2. Abort one process at a time until the deadlock cycle is eliminated.
3. In which order should we choose to abort?

a. Priority of the process.
b. How long process has computed, and how much longer to completion.
c. Resources the process has used.
d. Resources process needs to complete.
e. How many processes will need to be terminated.
f. Is process interactive or batch?

Recovery from Deadlock: Resource Preemption

1. Selecting a victim – minimize cost.
2. Rollback – return to some safe state, restart process for that state.
3. Starvation – same process may always be picked as victim, include number of rollback in cost factor

Combined Approach to Deadlock Handling

1. Combine the three basic approaches

a. prevention
b. avoidance
c. detection

allowing the use of the optimal approach for each of resources in the system.

2. Partition resources into hierarchically ordered classes.
3. Use most appropriate technique for handling deadlocks within each class.

S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	What are deadlock detection and recovery schemes? Describe deadlock detection algorithm in detail	Dec 2012	10

Unit-03/Lecture-05

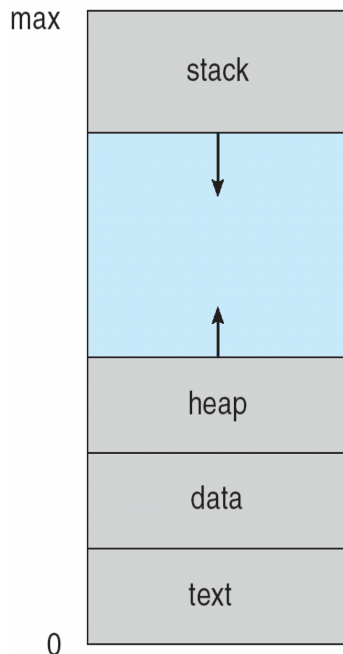
Memory Management Background

Program must be brought into memory and placed within a process memory space for it to be executed

Input queue – collection of processes on the disk that are waiting to be brought into memory to run the program

User programs (Applications) go through several steps before being run

Applications' view of the Memory:

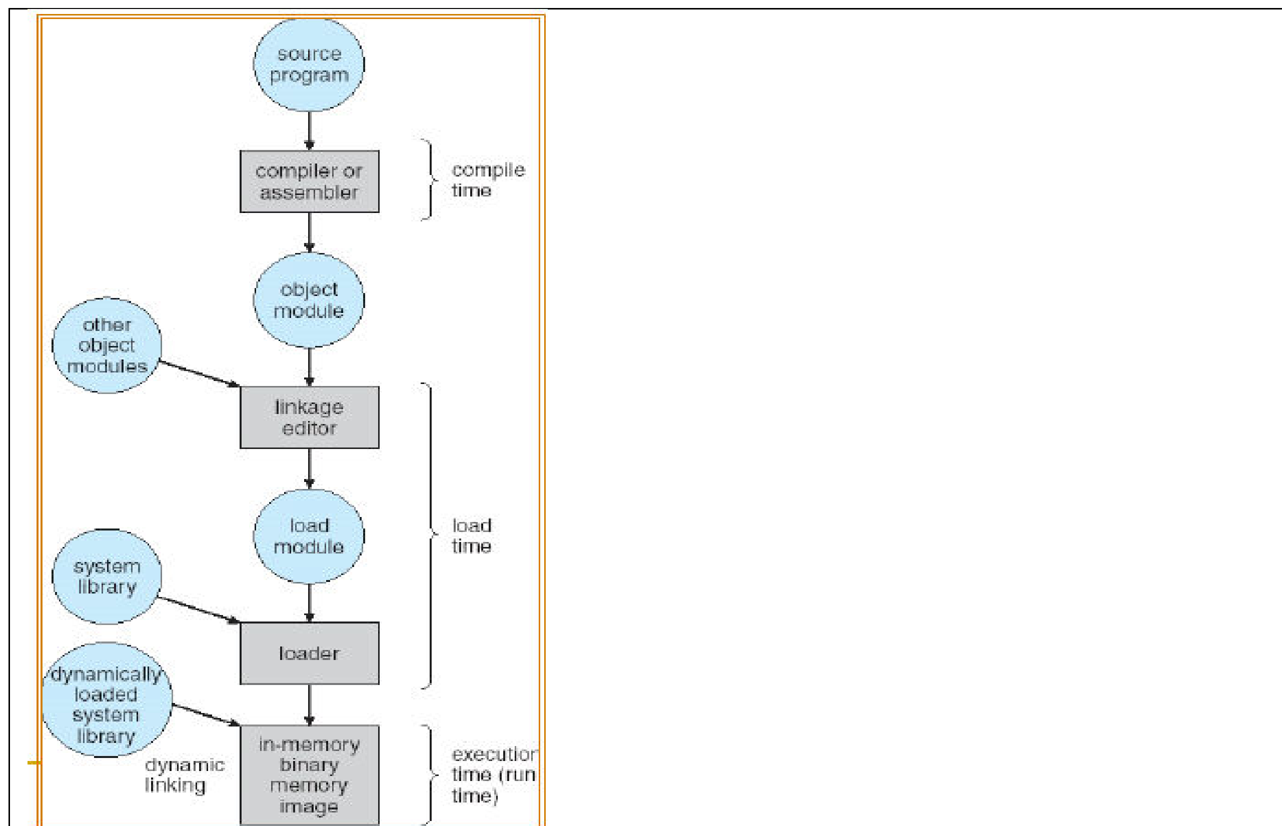


Binding of Instructions and Data to Memory

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

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

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).



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 – the address seen by the memory unit

Logical and physical addresses are the same in compiletime and load-time address-binding schemes

Logical (virtual) and physical addresses differ within the execution-time address-binding scheme

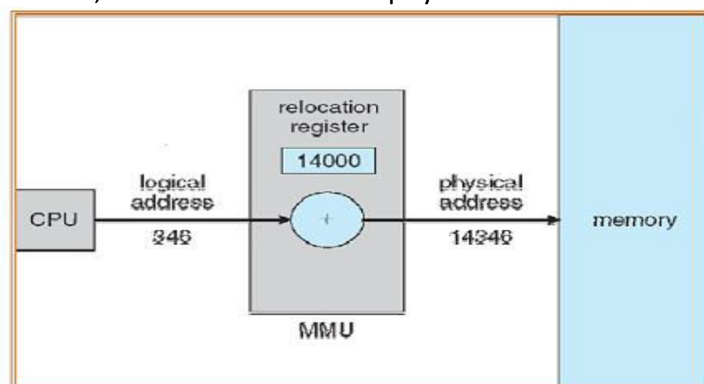
Memory-Management Unit (MMU)

Hardware device that maps virtual to physical address

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

Dynamic relocation using a relocation register



Dynamic Loading

_ Routine is not loaded until it is called

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

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

_ No special support from the operating system is required;
implemented through program design (overlays)

Dynamic Linking

_ Linking postponed until execution time
_ Small piece of code, **stub**, placed instead of the real procedure call – used to locate the appropriate memory resident library routine
_ Stub replaces itself with the address of the routine, and executes the routine
_ Operating system support needed to check if the routine is in memory and addressable by the process
_ Dynamic linking is particularly useful for libraries

S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	Explain memory management in multiprogramming environment.	Dec 2013	7
Q.2	Differentiate b/w a)Logical & physical address space b)Static & dynamic relocation	Dec 2012	10

we dont take any liability for the notes correctness. http://www.rgpvonline.com

Unit-03/Lecture-06

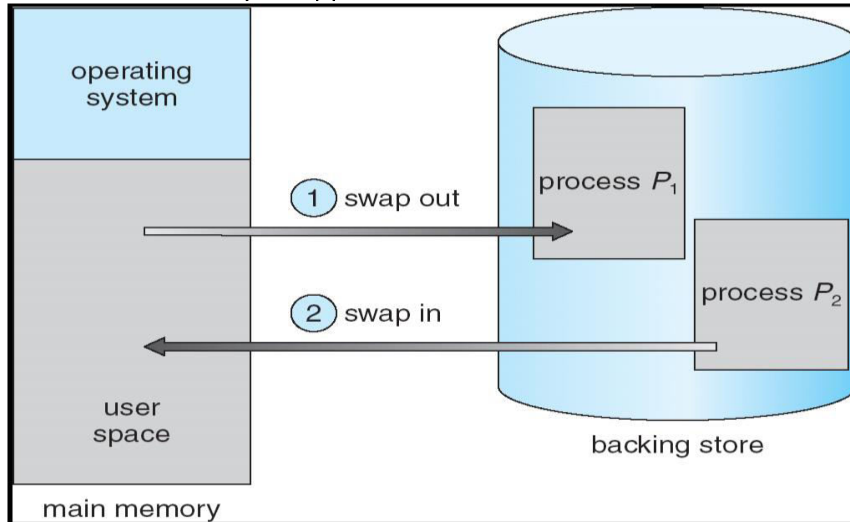
Swapping

_ A process' memory 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



Contiguous Allocation

Main memory is usually split into two partitions:

Resident operating system, usually held in low memory with interrupt vector

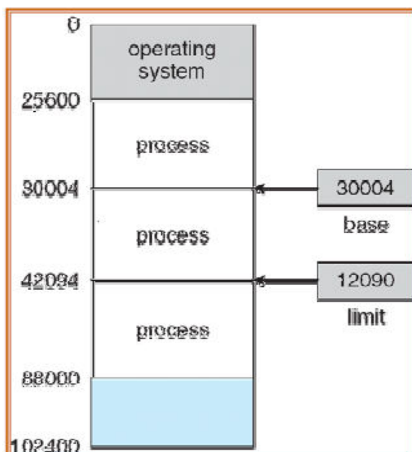
User processes then held in high memory

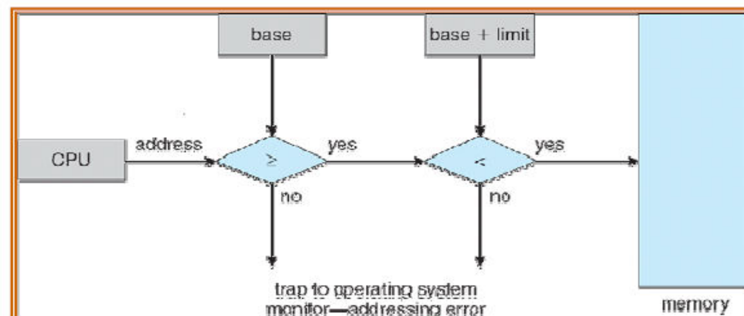
Single-partition allocation

_ Relocation-register scheme 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





A base and a limit register define a logical address space

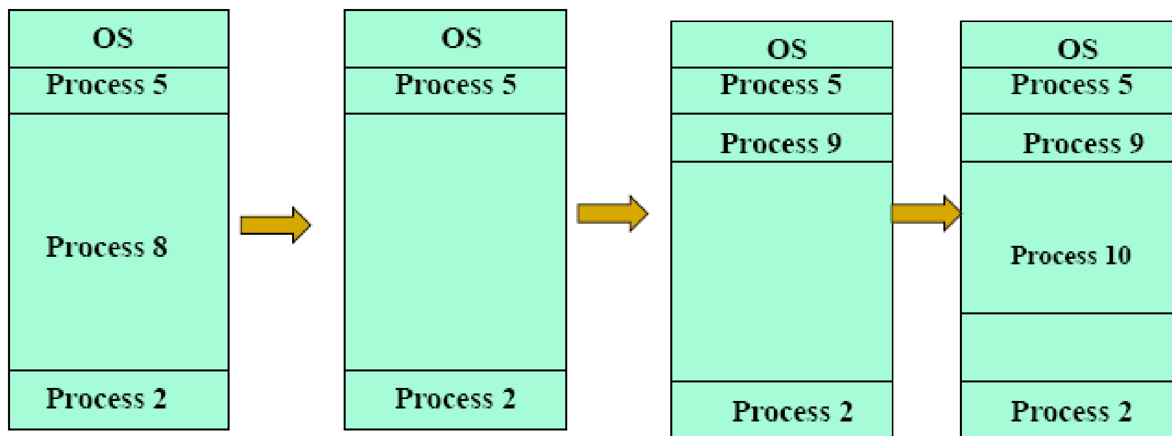
Multiple-partition allocation

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:

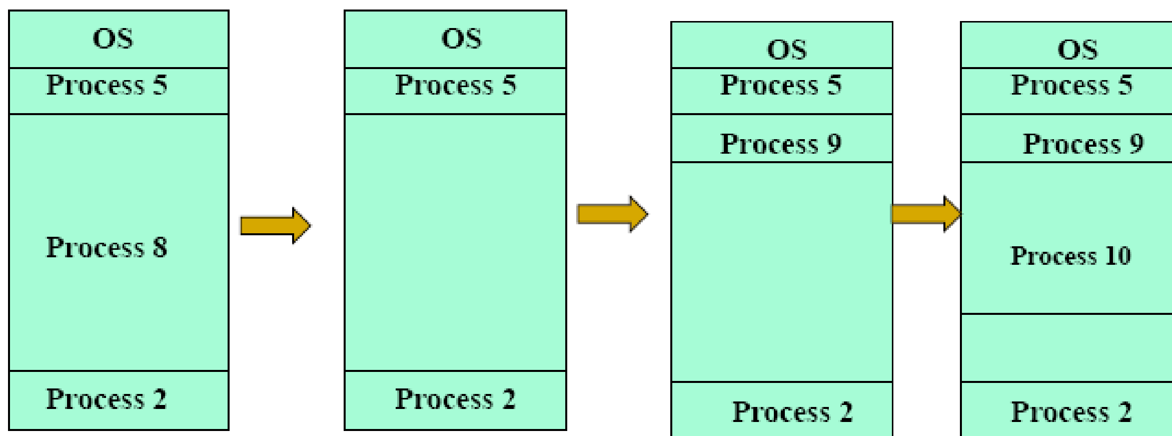
a) allocated partitions b) free partitions (holes)



Holes in Memory Space

_ Contiguous allocation produces holes

_ Holes are produced also by other allocation strategies



Dynamic Storage-Allocation Problem

First-fit: Allocate the first hole that is big enough

_ Fastest method

Best-fit: Allocate the smallest hole that is big enough;

_ Must search entire list, unless ordered by size. Produces the smallest leftover hole

_ Good storage utilization

Worst-fit: Allocate the largest hole; must also search entire list. Produces the largest leftover hole.

_ Low storage fragmentation

Fragmentation

External Fragmentation

_ Total free memory space exists to satisfy a request, but it is not contiguous and contiguous space is required

Internal Fragmentation

Memory is allocated using some fixed size memory "partitions"

Allocation size often defined by hardware
Allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used
A compromise is needed:
Large partitions – too high internal fragmentation
Small partitions – too many of partitions to administer
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**

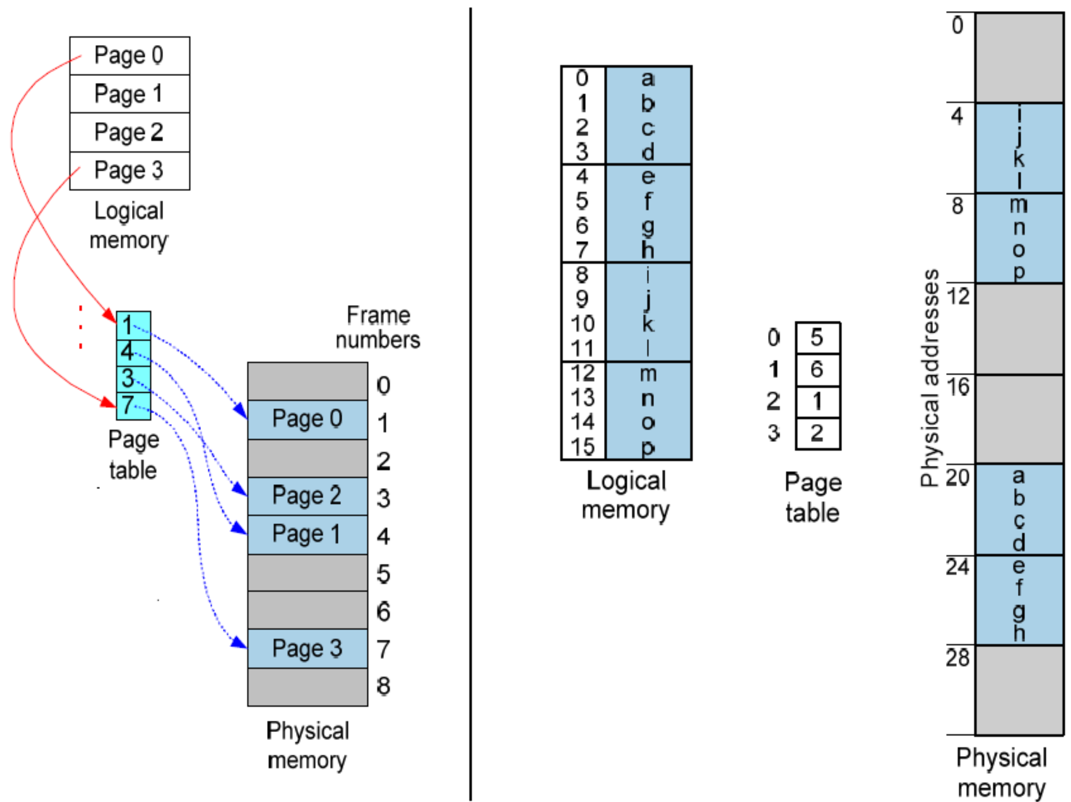
Paging

Contiguous logical address space of a process can be mapped to **noncontiguous** physical allocation
process is allocated physical memory whenever the latter is available
Divide physical memory into fixed-sized blocks called **frames** (size is power of 2, between 512 bytes and 8192 bytes)
Divide logical memory into blocks of same size called **pages**
_ Keep track of all free frames
_ To run a program of size n pages, need to find n free frames and load program
_ Set up a page table to translate logical to physical addresses
_ Internal fragmentation may occur

S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	Differentiate b/w a)Internal & External Fragmentation b)First fit, best fit & worst fit	Dec 2012 Dec 2011	10 8
Q.2	Given memory partitions of 100k, 500k, 200k and 600k (in order), how would each of the first-fit, Best-fit, and worst-fit algorithm place process of 212k, 417k, 112k and 426k (in order). Which algorithm makes the most efficient use of memory.	Dec 2013	7

Unit-03/Lecture-07

Paging Address Translation Principle

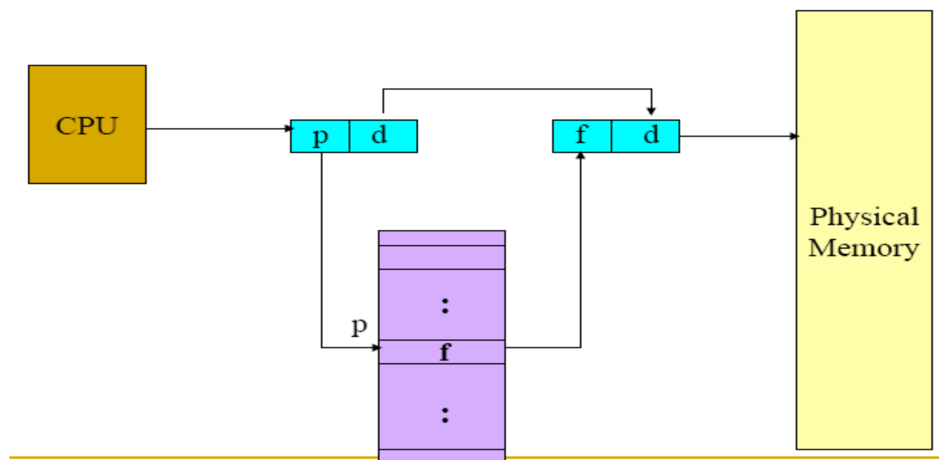


Address Translation Scheme

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) – combined with base address to define the physical memory address that is sent to the memory unit



Implementation of Page Table

Paging is implemented in hardware

Page table is kept in main memory

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

Page-table length register (PTLR) 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 memory** or **translation look-aside buffers (TLBs)**

Associative Memory

Associative memory – parallel search – VERY COMPLEX CIRCUITRY

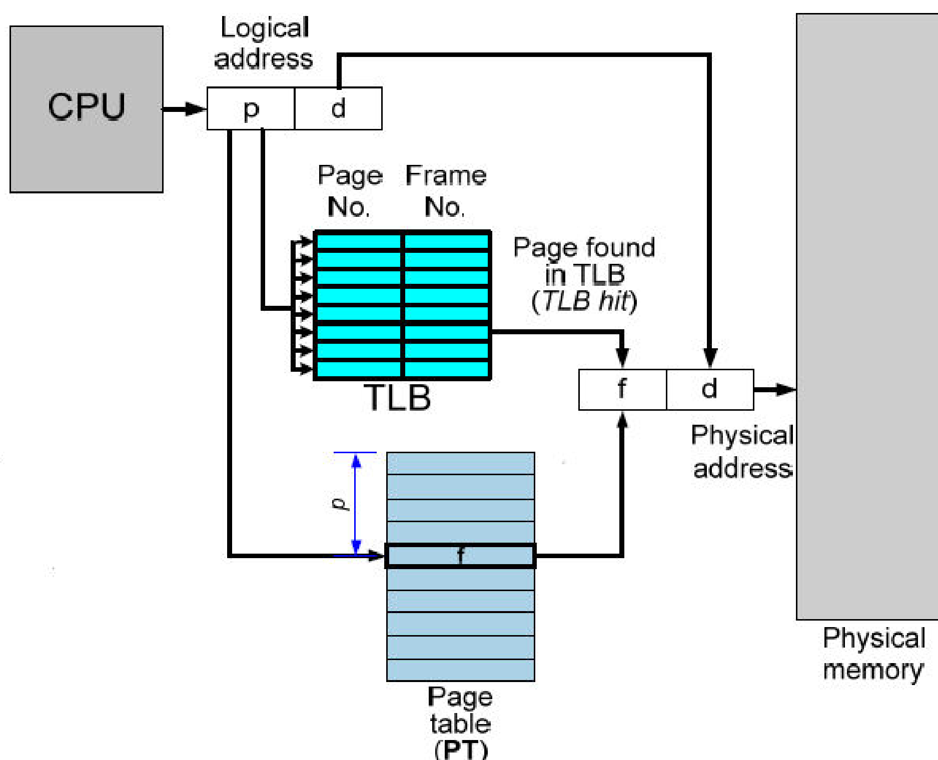
Page #	Frame #

Address translation (P# → F#)

If P# is in associative register, get F# out

Otherwise get F# from page table in memory

Paging Hardware With TLB



Paging Properties

Effective Access Time with TLB

Associative Lookup = e time unit

Assume memory cycle time is 1 microsecond

Hit ratio – percentage of times that a page number is found in the associative registers; ration related to number of associative registers,

Hit ratio = a

Effective Access Time (EAT)

EAT = (1 + e) a + (2 + e)(1 – a)

= 2 + e – a

Memory protection implemented by associating protection bit with each page

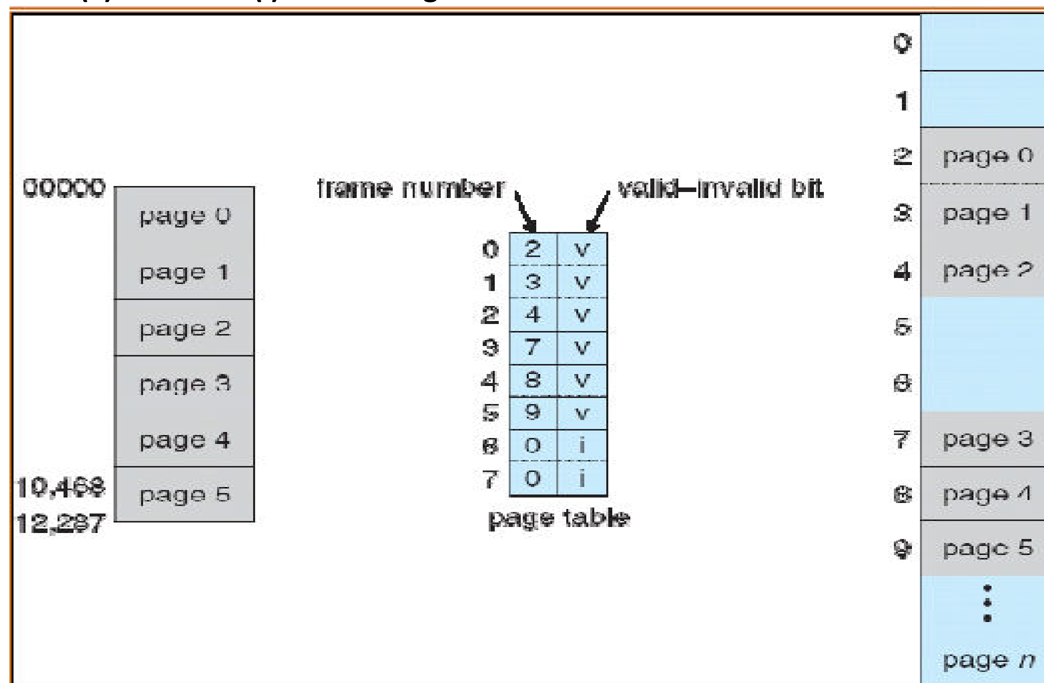
Usually two bits: Read-Only bit, Dirty bit (used as described later)

Valid-invalid bit attached to each entry in the page table:

"valid" indicates that the associated page is in the process' logical address space, and is thus a legal page

"invalid" indicates that the page is not in the process' logical address space

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

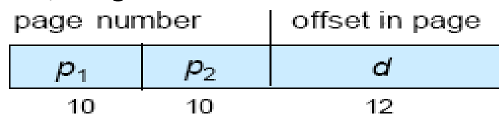


S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	Explain paging and implementation of page table	Dec 2009	8

Unit-03/Lecture-08

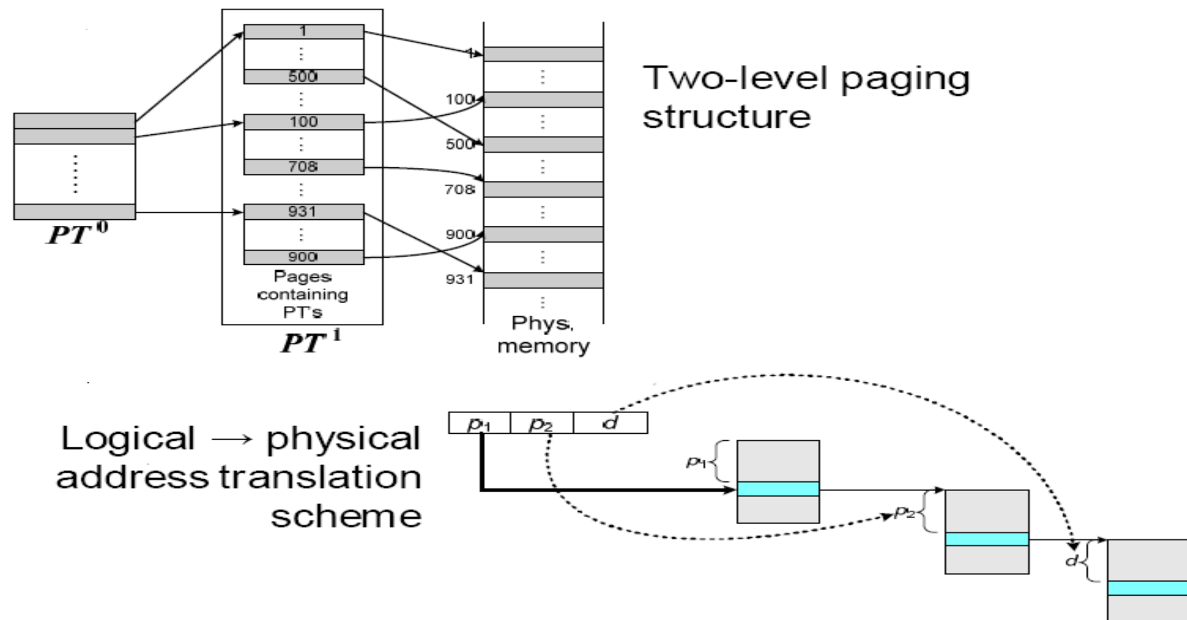
Page Table Structures
**Hierarchical Paging
**Hashed Page Tables
**Inverted Page Tables

Hierarchical Page Tables
Break up the logical address space into multiple page tables
A simple technique is a two-level page table
A logical address (on 32-bit machine with 4K page size) is divided into:
_ a page number consisting of 20 bits
_ an offset within page consisting of 12 bits
The page table itself can also be paged, the page number can be further divided into:
_ a 10-bit page number
_ a 10-bit page offset
Thus, a logical address is:

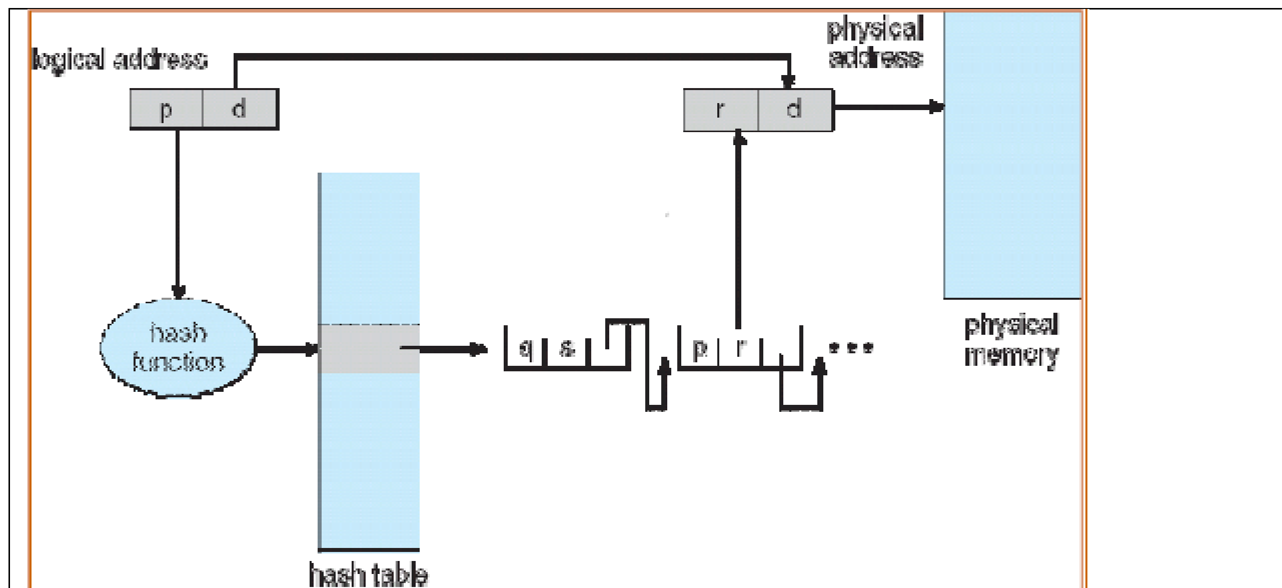


where pi is an index into the outer page table, and p2 is the displacement within the page of the outer page table

Two-Level Page-Table Scheme



Hashed Page Tables
_ Common for address spaces > 32 bits
_ The virtual page number is hashed into a page table. This page table contains a chain of elements hashing to the same location.
_ Virtual page numbers are compared in this chain searching for a match. If a match is found, the corresponding physical frame is extracted.



Inverted Page Table

_ One entry for each frame of real memory

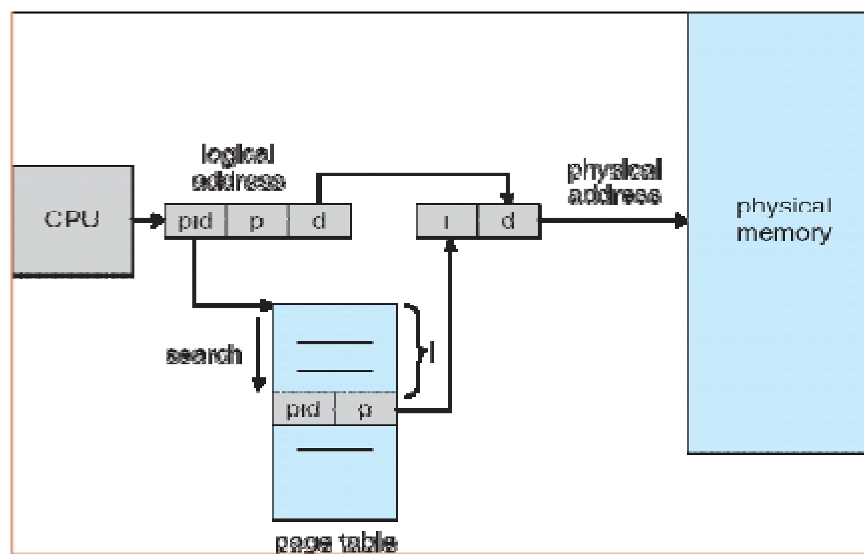
_ Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page

_ Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs

Main advantage:

_ only one PT for all processes

_ Use hash table to limit the search to one –
or at most a few –page-table entries



.

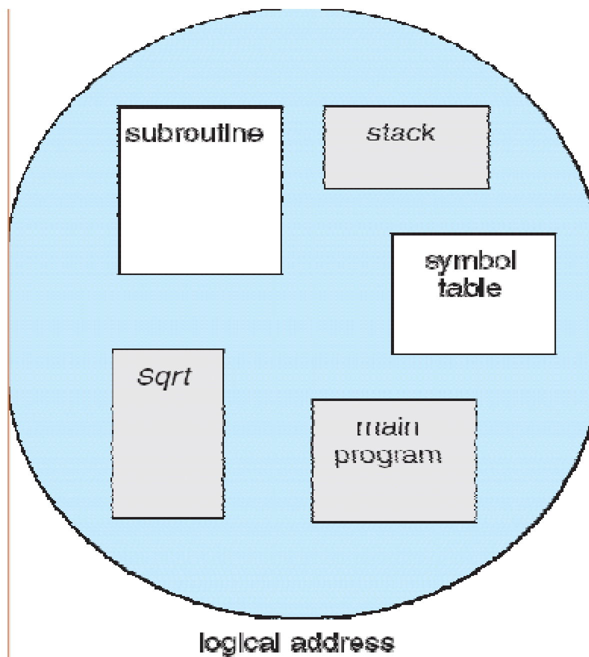
S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	Discuss structure of page table	Dec 2010	10

Unit-03/Lecture-09

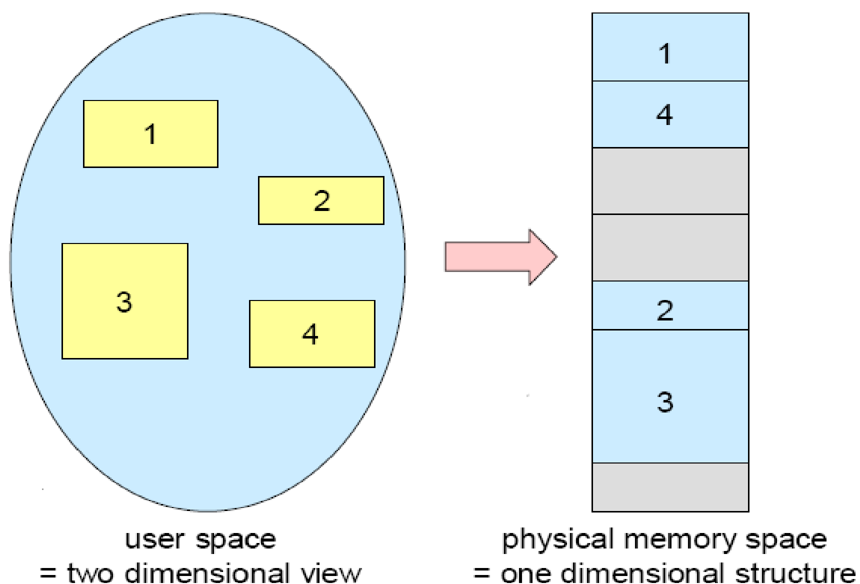
Segmentation

Memory-management scheme that supports user view of memory
A program is a collection of segments. A segment is a logical unit such as: main program, procedure, function, method, object, local variables, global variables, common block, stack, symbol table, arrays

User's View of a Program



Logical View of Segmentation



we dont take any liability for the notes correctness.

Segmentation Architecture

Logical address consists of a couple:
<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

Relocation

dynamic by segment table

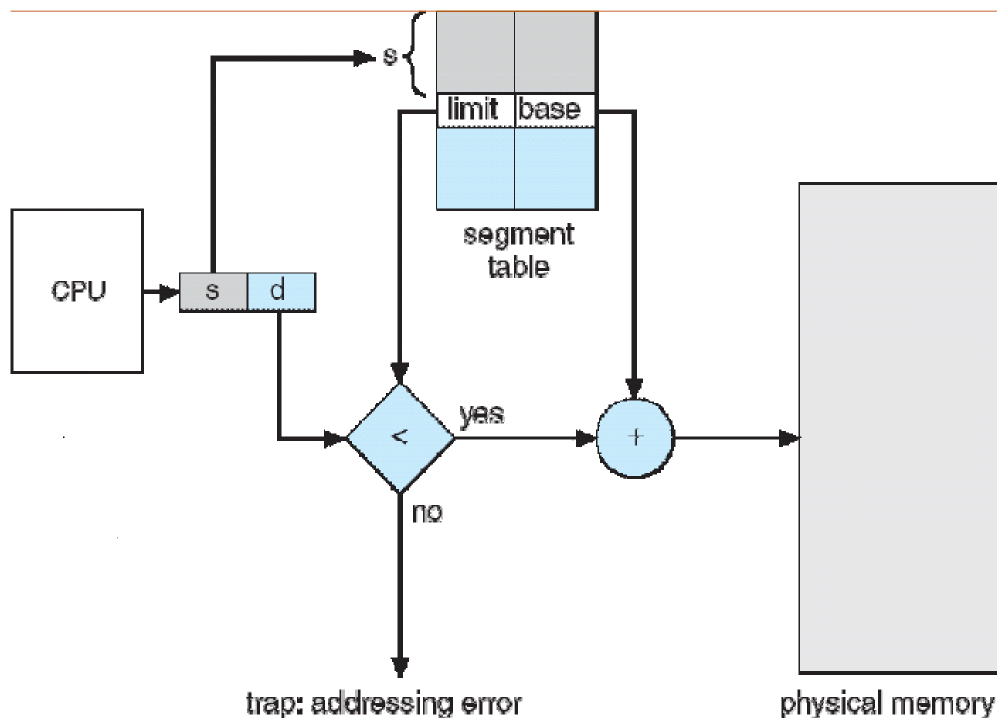
Sharing

shared segments – same segment number

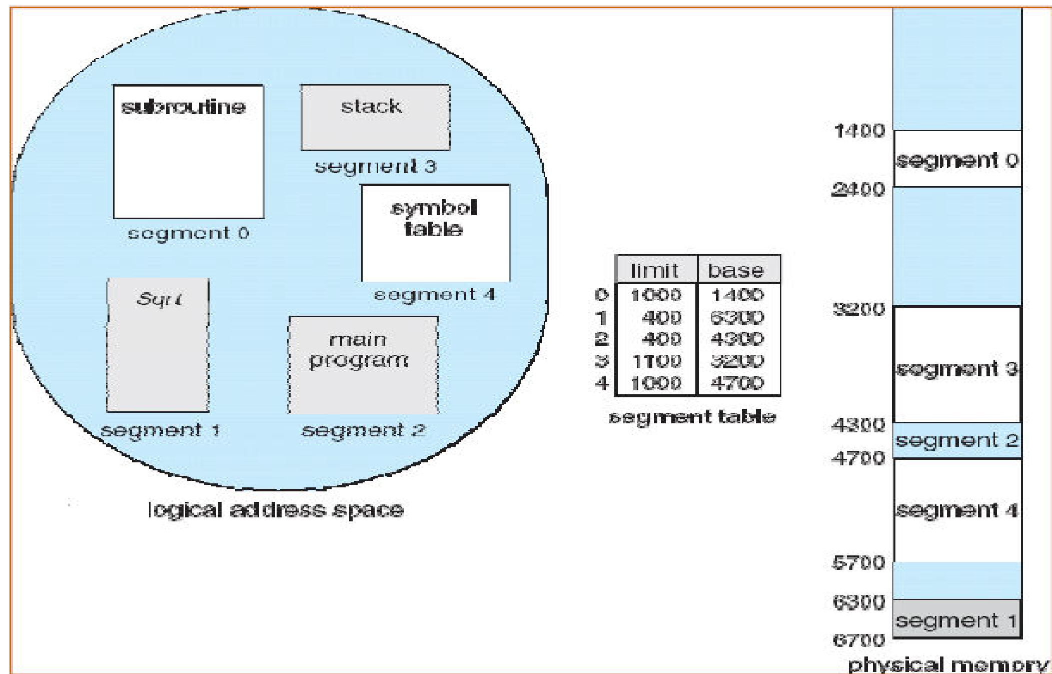
Allocation.

first fit/best fit – external fragmentation
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

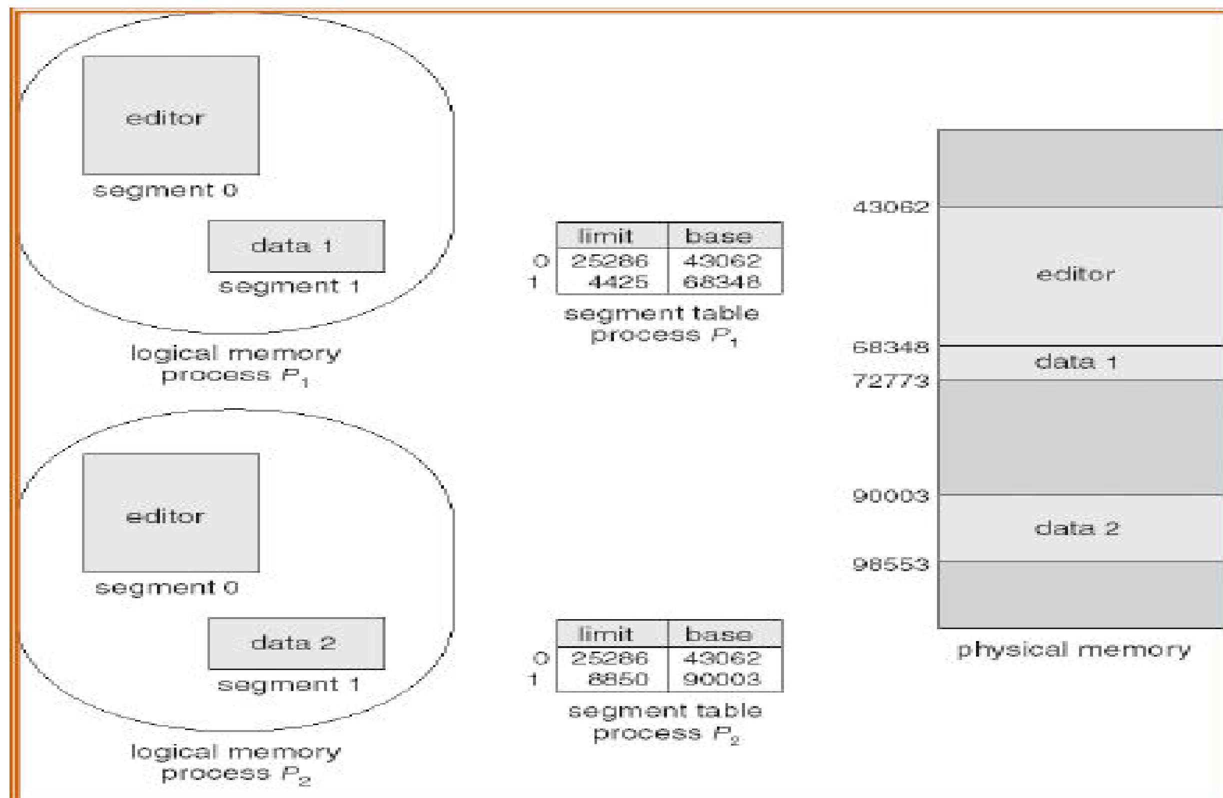
Address Translation Architecture



Example of Segmentation



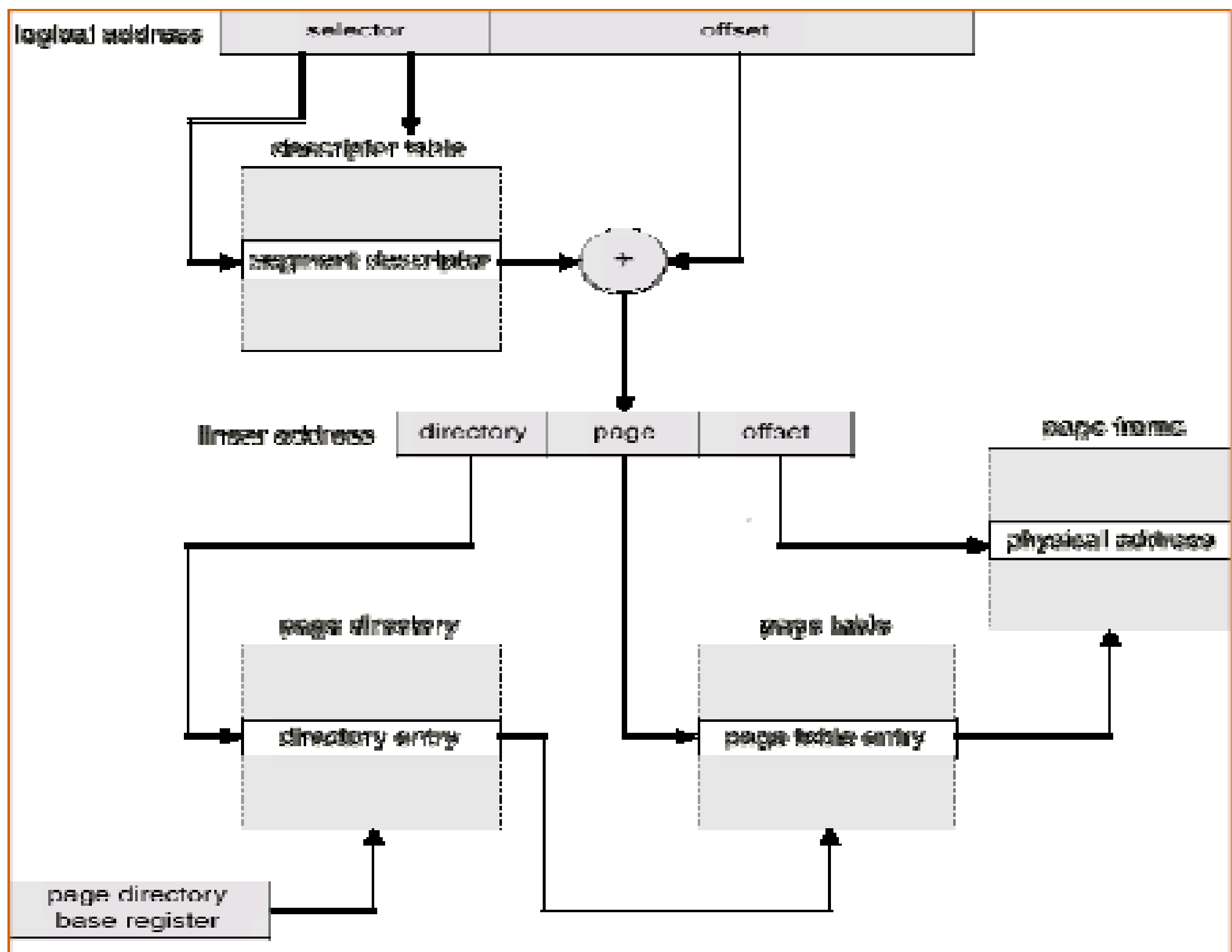
Sharing of Segments



Segmentation with Paging – Intel IA32

IA32 architecture

uses segmentation with paging for memory management with a two-level paging scheme



S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	Explain segmentation with paging.	Dec 2012	7
Q.2	Explain why sharing of segmentation is used when pure paging is used.	Dec 2012	10