

Operating Systems Concepts

Review I

Final & Resit exam structure

I. Fundamentals

II. CPU scheduling, Memory management, Disk scheduling

III. Resource allocation

IV. Operating System in C Language

The final exam covers the lectures, tutorials and labs of week 1 → 13

NO Mcqs in the final exam!

The final exam is OPEN-BOOK

Resources managed by an OS

- CPUs (processes)
- Main memory and Virtual memory
- Secondary storage
- I/O devices
- File system
- Protection and security

Virtual Machines & Distributed Systems

PROCESSES

Process – a program in execution;

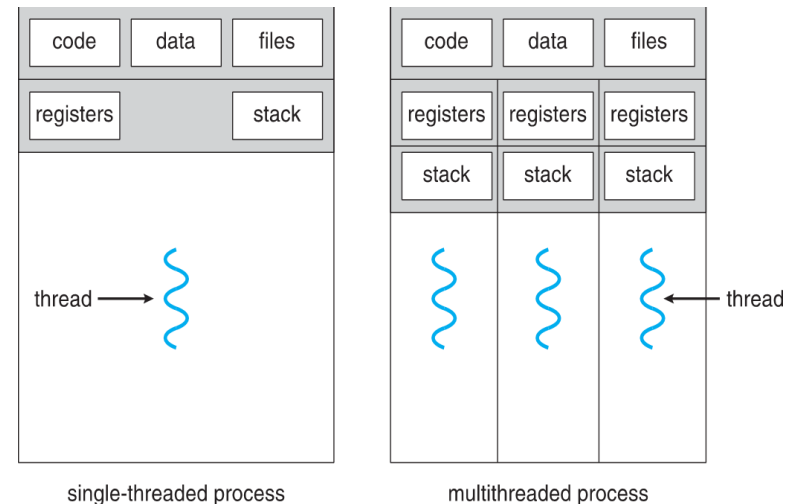
The states of a process :

- **new**: The process is being created
- **running**: Instructions are being executed
- **waiting**: The process is waiting for some event to occur
- **ready**: The process is waiting to be assigned to a processor
- **terminated**: The process has finished execution

process state
process number
program counter
registers
memory limits
list of open files
...

Thread *fundamental unit of CPU utilization that forms the basis of multithreaded computer systems*

Multiple threads can exist within one process, executing concurrently and sharing resources.



USER LEVEL & KERNEL LEVEL THREAD

User Level Threads	Kernel Level Thread
User level threads are faster to create and manage.	Kernel level threads are slower to create and manage.
Implementation is by a thread library at the user level.	Operating system supports creation of Kernel threads
User level thread is generic and can run on any operating system	Kernel level thread is specific to the operating system.
Multi-threaded application cannot take advantage of multiprocessing.	Kernel routines themselves can be multithreaded.

Explicit threading - the programmer creates and manages threads.

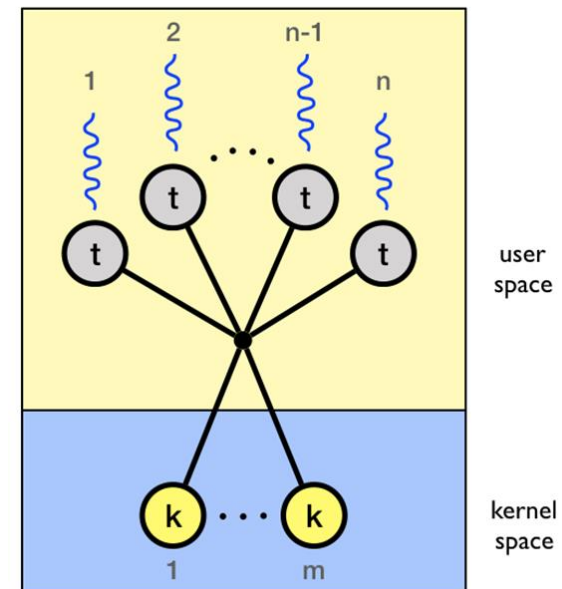
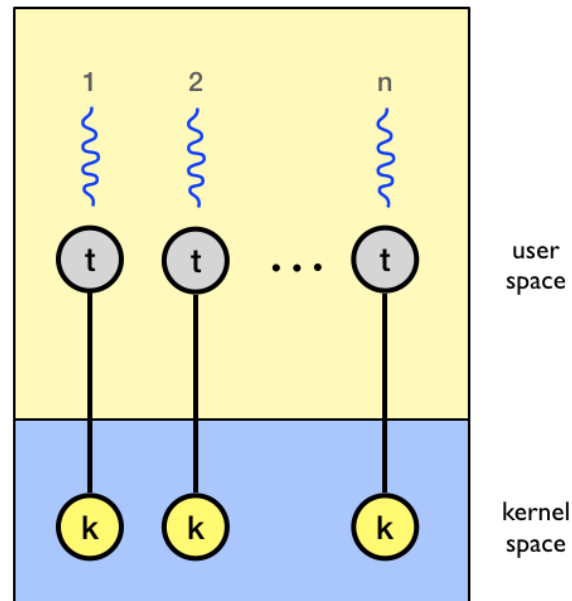
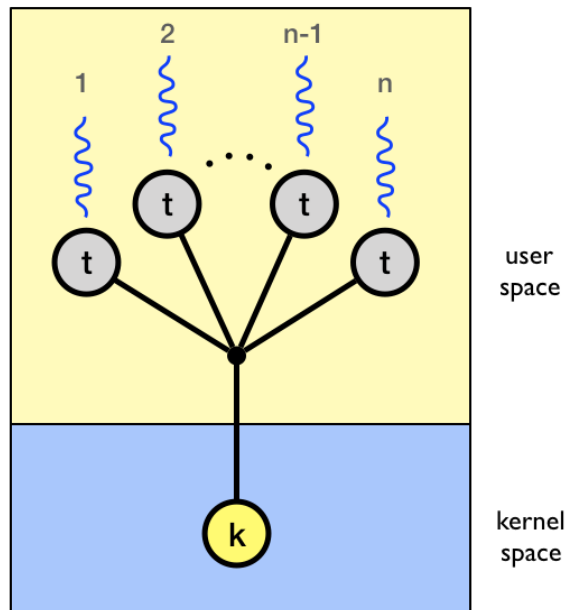
Implicit threading - compilers and run-time libraries create and manage threads.

Multithreading models are:

Many - to - One

One - to - One

Many - to - Many



Processes



```
graph LR; A[Processes] --- B[Communication]; A --- C[Synchronization]; A --- D[Scheduling]; A --- E[Deadlock];
```

Communication

Synchronization

Scheduling

Deadlock


```
graph LR; A[Processes] --- B[Communication]; A --- C[Synchronization]; A --- D[Scheduling]; A --- E[Deadlock];
```

Processes

Communication

Synchronization

Scheduling

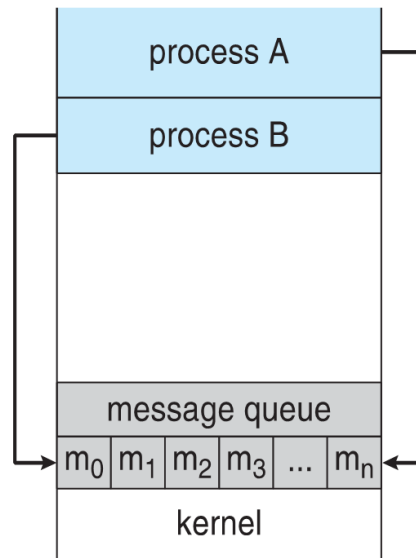
Deadlock

INTER-PROCESS COMMUNICATION

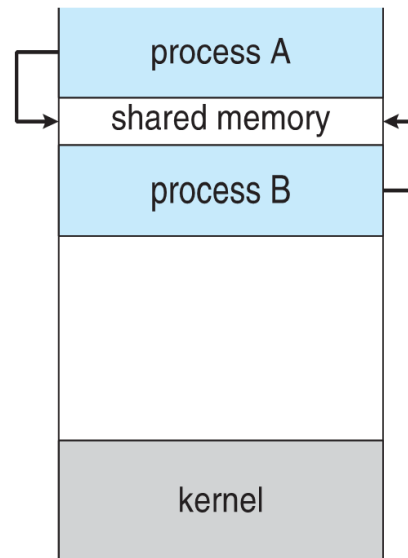
Independent Processes - neither affect other processes or be affected by other processes.

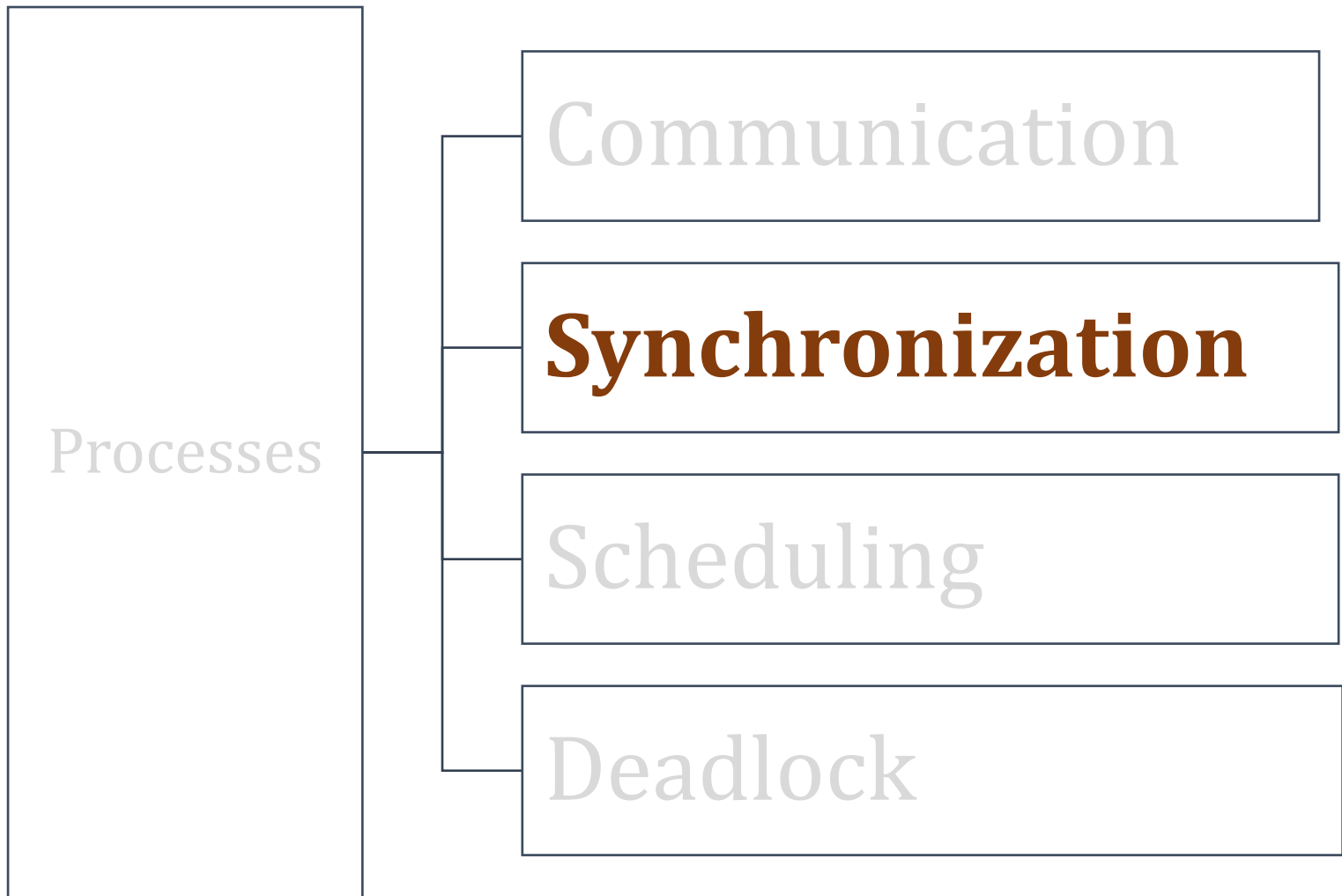
Cooperating Processes - can affect or be affected by other processes.

(a) Message passing.



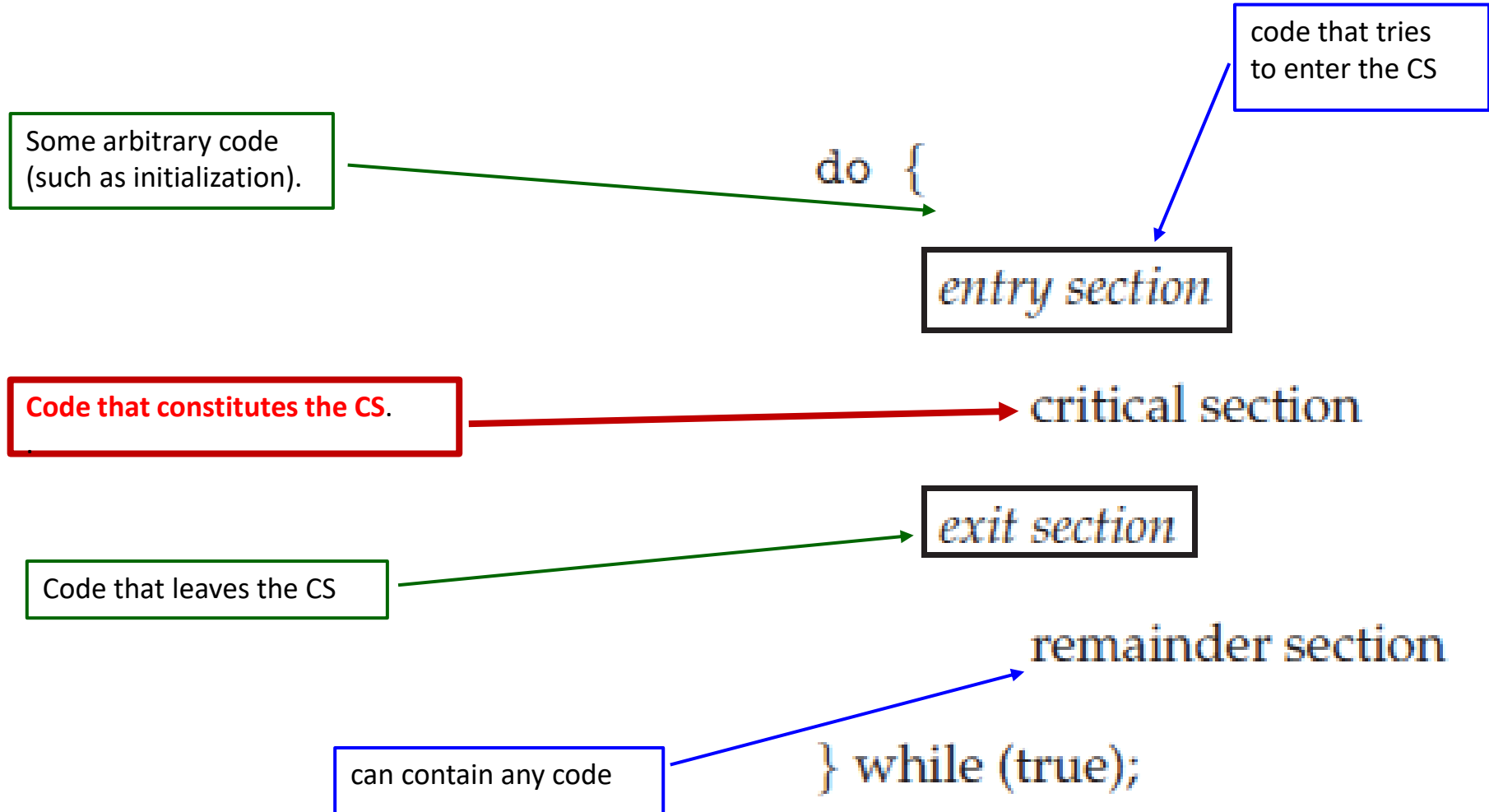
(b) Shared memory.





THE CRITICAL SECTION PROBLEM

The critical section refers to the **segment of code where processes access shared resources**, such as common variables and files, memory space, etc.



Formally, the following requirements should be satisfied:

- ❑ **Mutual exclusion** - no two process/thread can be simultaneously present inside critical section at any point in time
- ❑ **Progress** - no process/thread running outside the critical section should block the other interesting process from entering into a critical section when, in fact, the critical section is free
- ❑ **Bounded waiting** - No process/thread should have to wait forever to enter into the critical section.

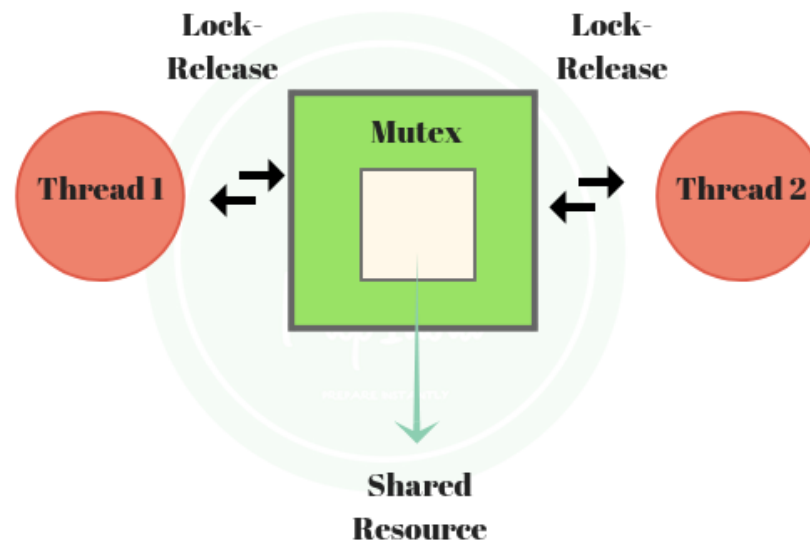
```
while (true) {  
    entry section  
    critical section  
    exit section  
    non-critical section  
}
```

SOLUTIONS TO THE CRITICAL SECTION PROBLEM

- **Peterson's solution** - restricted to **two processes** that alternate execution between their critical sections and remainder sections.
- **Hardware solution** - rely on some special machine instructions
- **Mutex lock** - software to protect critical regions and avoid race conditions.
- **Semaphores** - like the mutex lock, provide sophisticated ways for the process to synchronize their activities.

MUTEX LOCKS / MUTUAL EXCLUSION

- a mutex is locking mechanism used to synchronize access to a resource.



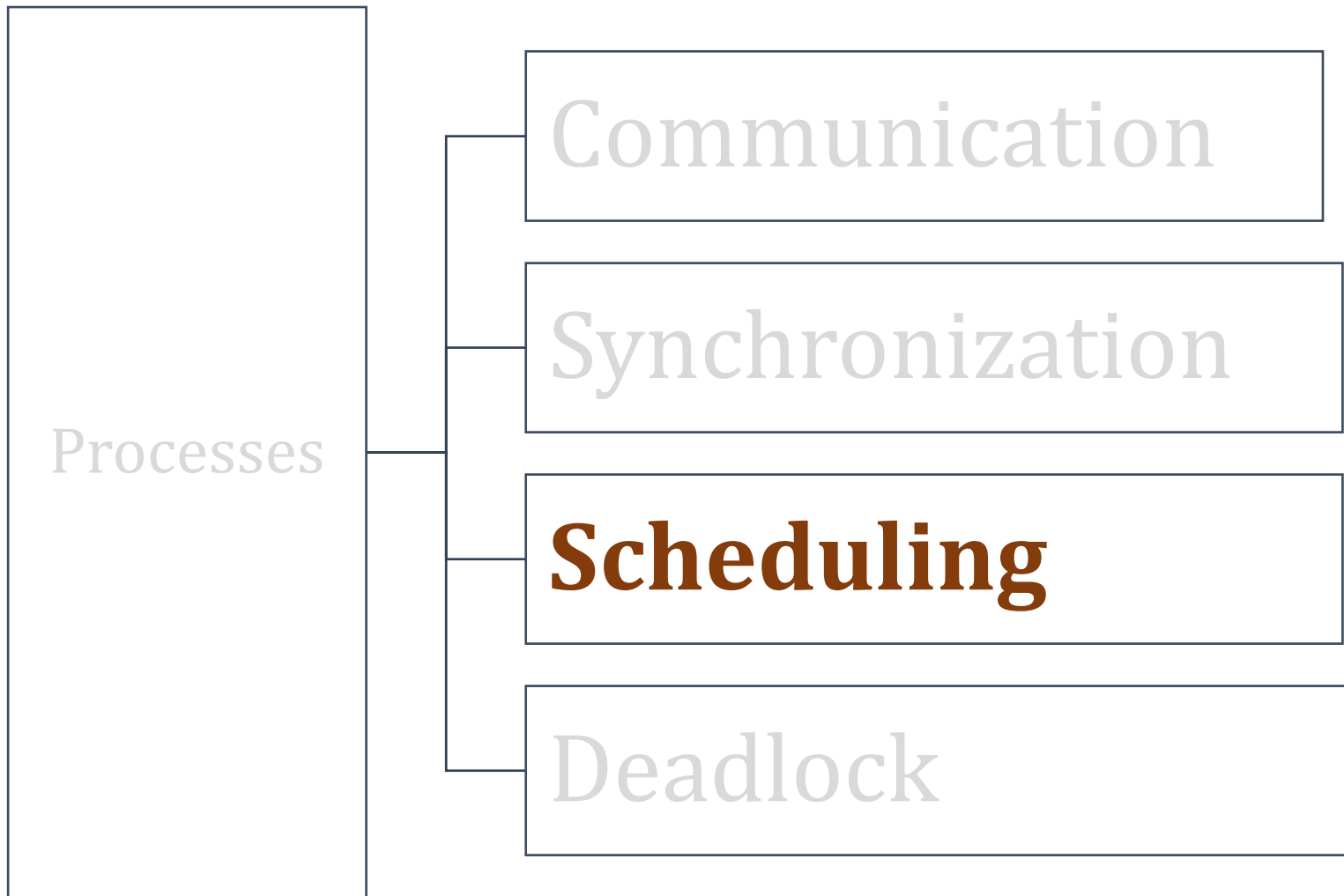
- **Mutex object is locked or unlocked by the process/thread requesting or releasing the resource**

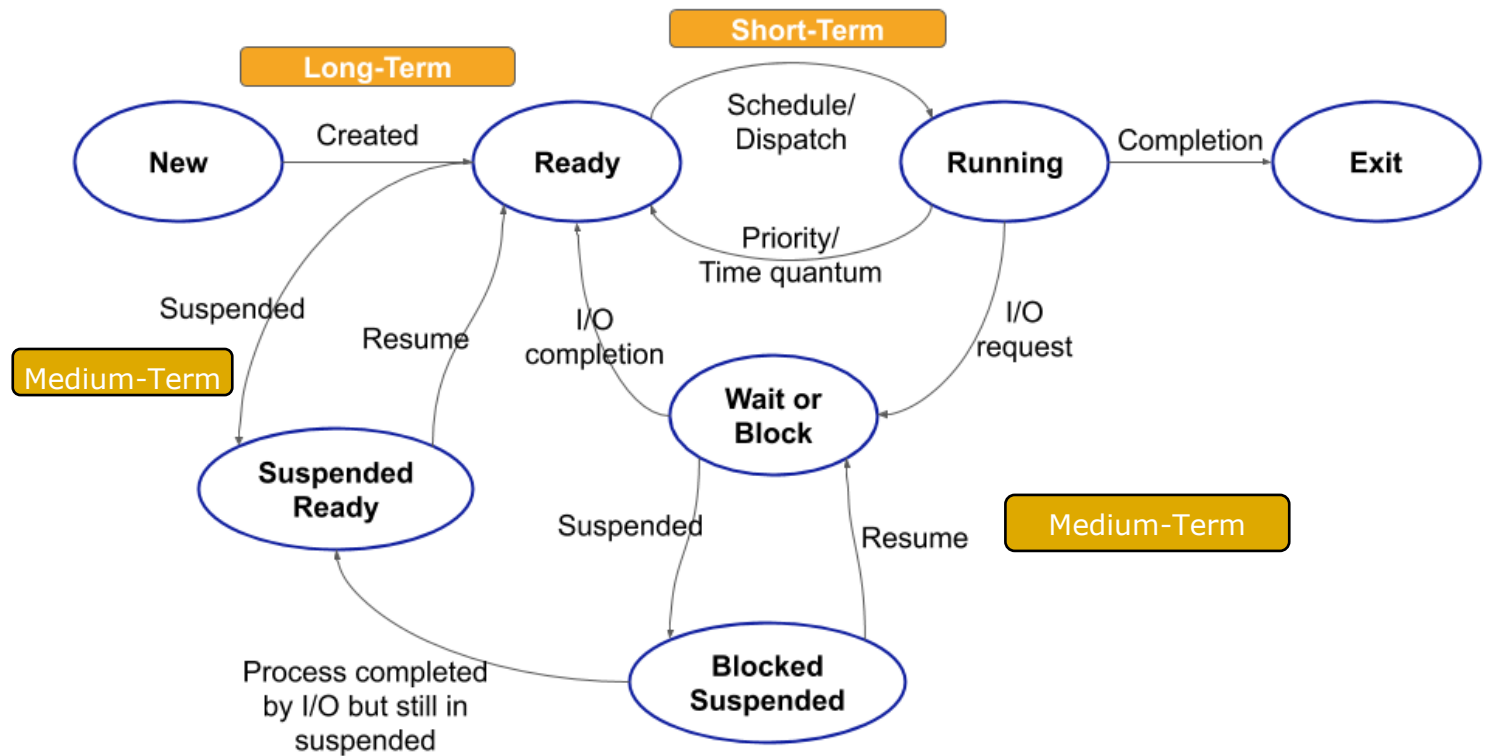
SEMAPHORE

There are two main types of semaphores:

- **COUNTING SEMAPHORE** – allow an arbitrary resource count. Its **value can range over an unrestricted domain**. It is used to control access to a **resource that has multiple instances**.
- **BINARY SEMAPHORE** – is also known as **mutex lock**. It can have only **two values: 0 and 1**.

Its **value is initialized to 1**. It is used to implement the solution of critical section problem with multiple processes.





PREEMPTIVE SCHEDULING - a running process can be interrupted by any other process.

NON-PREEMPTIVE SCHEDULING - a running process cannot be interrupted by any other process.

NON – Real -Time System

- is not associated with the timing constraint.
- not associated with any time bound.
- just need to be completed in whatever time system may take.
e.g. desktops, mobile running Android, online web services etc.

Real - Time System

- is associated with the quantitative expression of time.
- tasks / process are scheduled to finish all the computation events involved in it into timing constraint.
- the timing constraint related to the real-time tasks/process is deadline.
- all the real-time tasks/process need to be completed before deadline.
e.g. flight control, nuclear power plant control, certain (e.g., radiological) medical instruments, railway crossing signals, etc.

NON – Real -Time System

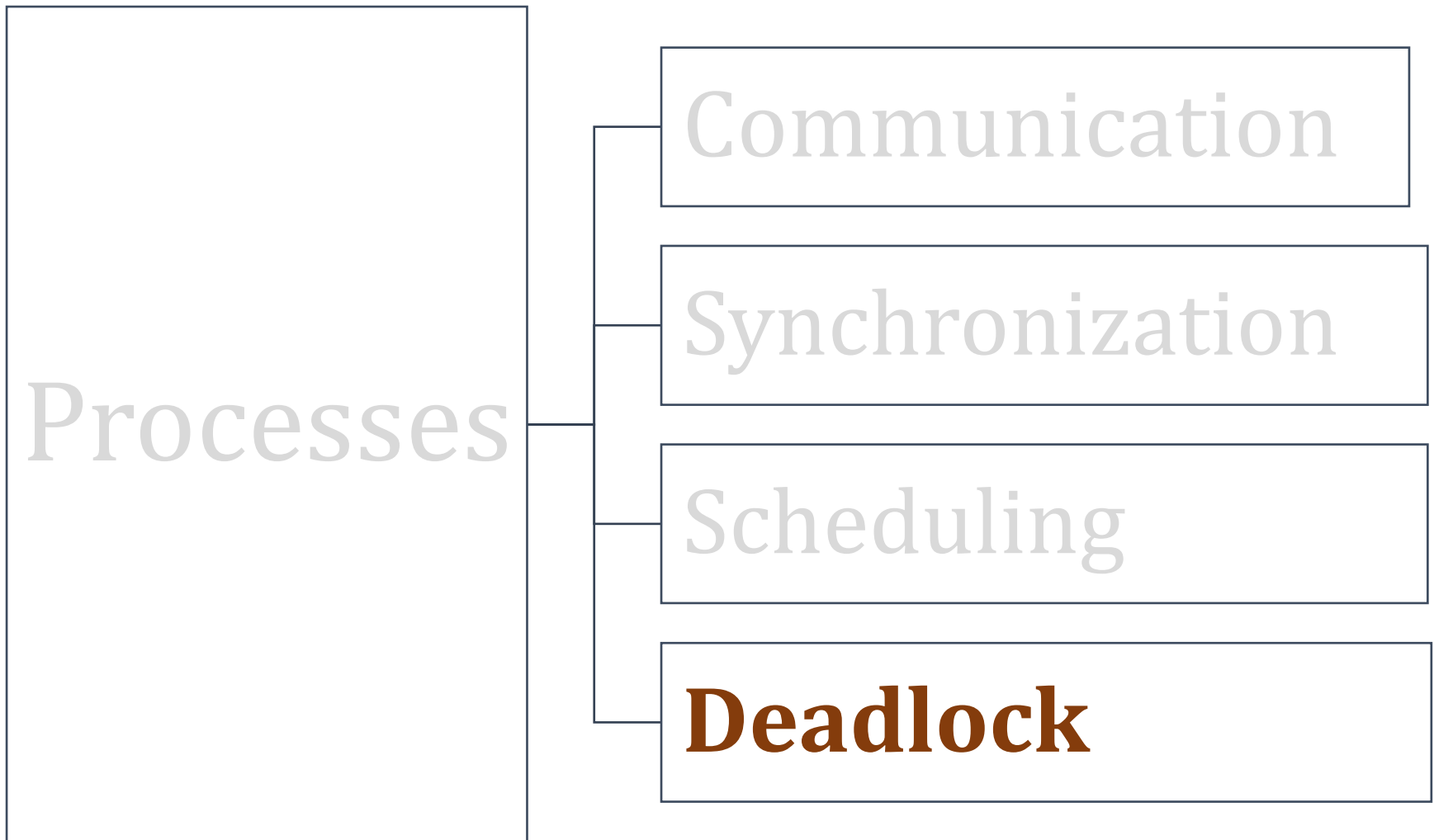
SCHEDULING ALGORITHMS

- ☐ **First-Come, First-Served (FCFS)**
- ☐ **Shortest-Job-First (SJF)**
- ☐ **Shortest Remaining Time First (SRTF)**
- ☐ **Priority Scheduling**
- ☐ **Round Robin(RR)**
- ☐ **Multiple-Level Queues**
- ☐ **Multilevel Feedback Queue (MLFQ)**

Real -Time System

SCHEDULING ALGORITHMS

- **Priority-Based Scheduling** - each process a priority based on its importance
- **Rate-Monotonic Scheduling** - *static priority* (the shorter the period = the higher the priority) with preemption.
 - *Missed Deadlines with Rate Monotonic Scheduling*
- **Earliest-Deadline-First Scheduling** – *dynamic priority* (the earlier the deadline = the higher the priority) with preemption.
- **Proportional Share Scheduling** - T shares are allocated among all processes in the system.



CONDITIONS OF DEADLOCK

Conditions that give rise to a deadlock:

- **Mutual exclusion:** Some resource types cannot allow multiple processes to access it at the same time \Rightarrow only one process at a time can use a resource
- **Hold and wait:** When all the processes are holding some resources and waiting for other resources, a deadlock may occur
- **No preemption:** If a resource cannot be pre-empted, it may lead to a deadlock.
- **Circular wait:** The Resource Allocation Graph RAG has a cycle.

Ensure that the system will *never* enter a deadlock state:

- **Deadlock Prevention** \Rightarrow eliminating any of the four conditions
- **Deadlock Avoidance** \Rightarrow ensure that a system will never enter an unsafe state.
- **Deadlock Detection And Recovery** \Rightarrow can run an algorithm to check for the cycle in the Resource Allocation Graph.
 - traditional OS (e.g. Windows) doesn't deal with deadlock recovery as it is a time and space-consuming process.
 - Real-Time OS use Deadlock recovery.

If a system is in **safe state** \Rightarrow no deadlocks

If a system is in **unsafe state** \Rightarrow possibility of deadlock

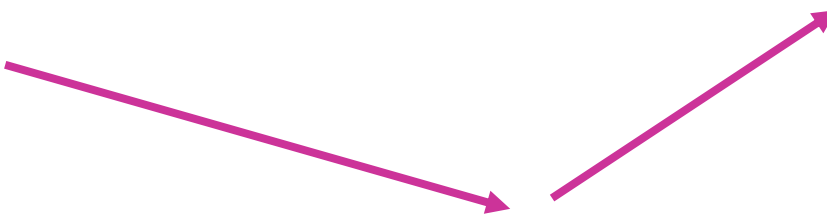
How to check if system is in safe state

Total resources

R1	R2	R3
15	8	8

Available

R1	R2	R3
3	3	2



Process	Max			Alloc			Need(Max – Alloc)		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	5	6	3	2	1	0	3	5	3
P2	8	5	6	3	2	3	5	3	3
P3	4	8	2	3	0	2	1	9	0
P4	7	4	3	3	2	0	4	2	3
P5	4	3	3	1	0	1	3	3	2

$Available[i] = Available[i] - Request[i];$

$Allocation[i] = Allocation[i] + Request[i];$

$Need[i] = Need[i] - Request[i];$

MEMORY

There are two types of memory:

- real memory (main memory)
- virtual memory.

Duties of a Memory Management (MM) System

- Protection
- Relocation
- Sharing
- Logical Organization of memory
- Physical Organization of memory

Logical address – generated by the CPU; also referred to as *virtual address*.

When the process starts executing, *relative or logical addresses* are generated by the CPU.

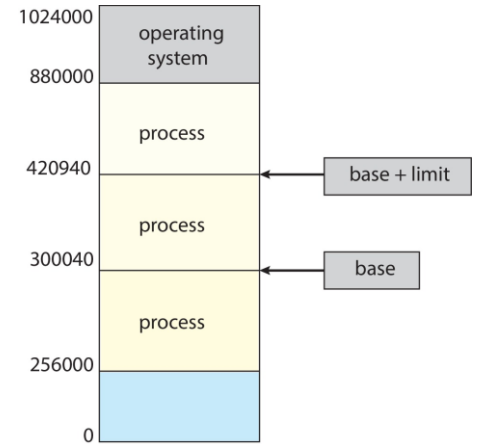
Physical address – address seen by the memory unit (*the absolute addresses*)

Hardware must provide a mechanism to **protect the memory allocated to each process** → *registers*

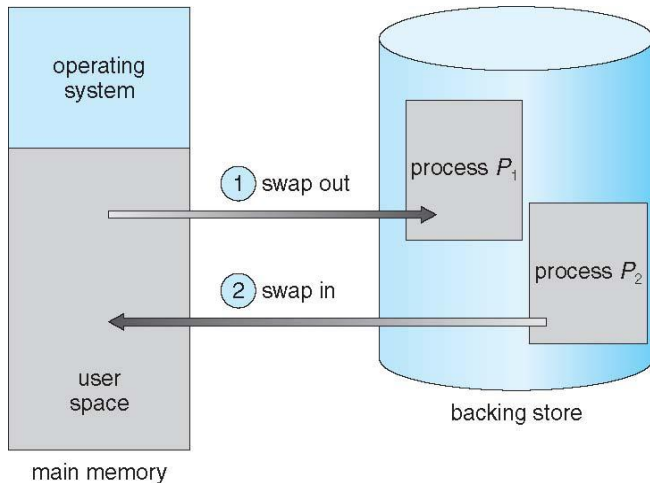
Processor registers (hardware support) hold:

- **base register** or **relocation register**
- **limit** or **bounce register**

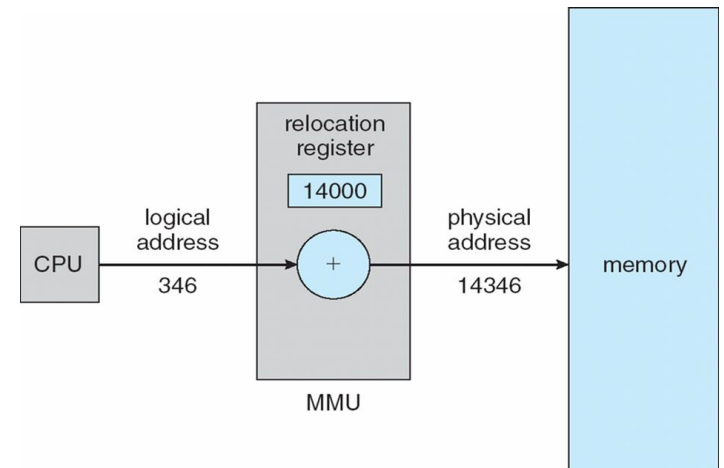
Hardware checks every address generated in user mode.



Swapping



Relocation



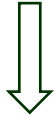
Logical Organization of Memory: Allocation

The memory allocation can be classified into two methods:

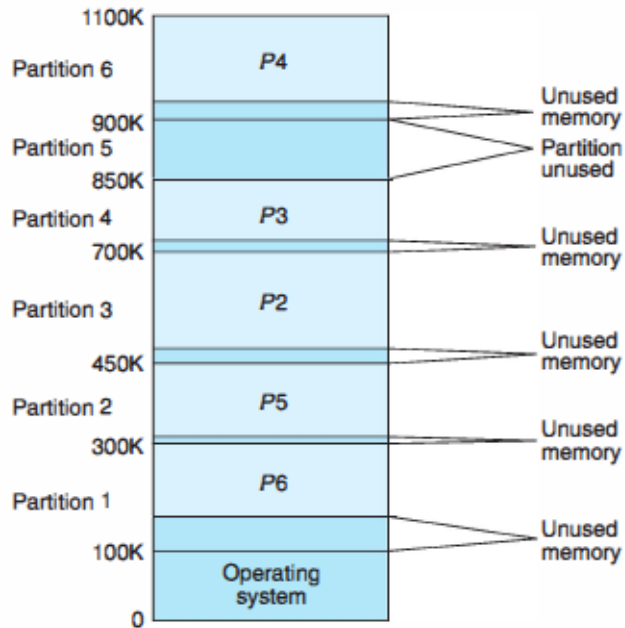
- **contiguous memory allocation** - assigns consecutive memory blocks to a process.
- **non-contiguous memory allocation** - assigns different blocks of memory in a nonconsecutive manner to a process.

CONTIGUOUS ALLOCATION

Fixed/Static Partitioning



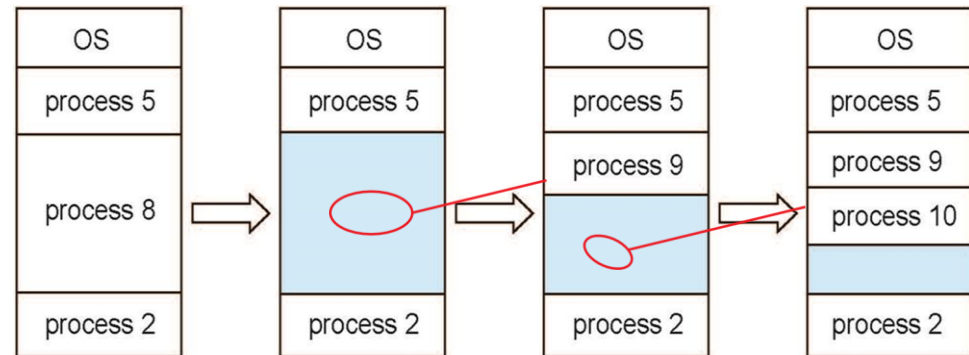
Internal fragmentation



Variable/Dynamic Partitioning



External fragmentation



Placement Algorithms:

First-fit: Allocate the first hole that's big enough

Best-fit: Allocate the smallest hole that's big enough

Worst-fit: Allocate the largest hole

To Be Continued