OPERATING SYSTEMS (CS-3000)

Lecture-2

DR. JAISHREE MAYANK

ASSISTANT PROFESSOR

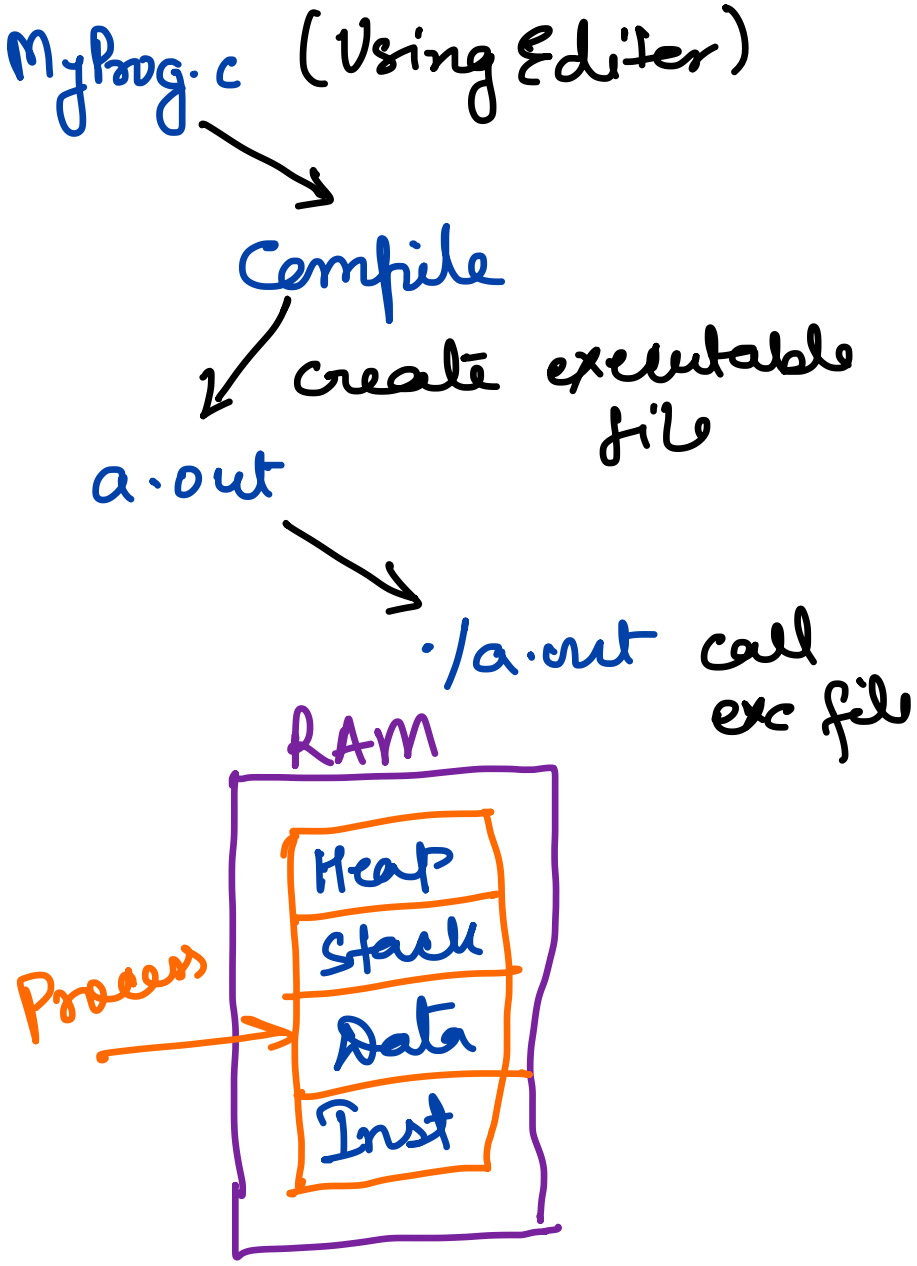
IIITDM KANCHEEPURAM

Objective of this lecture

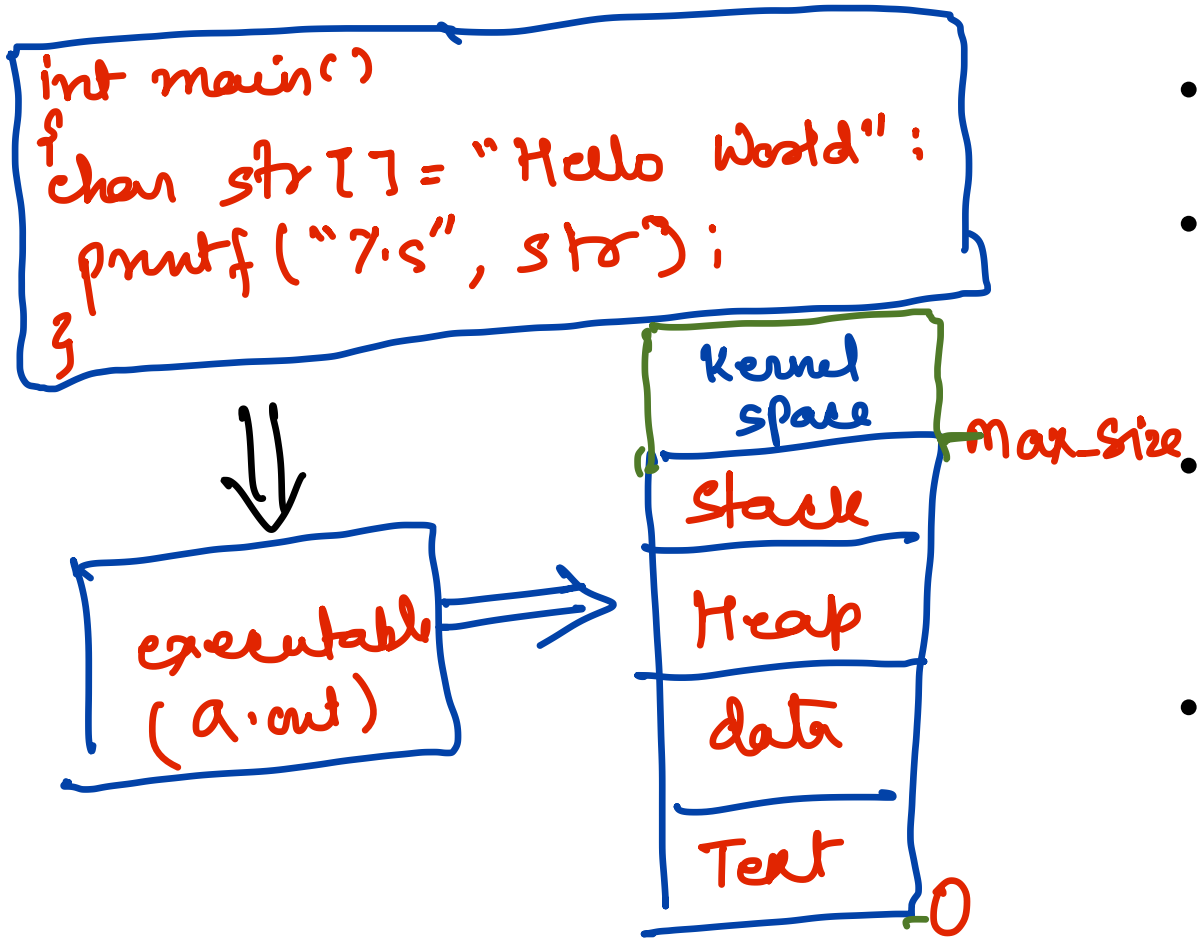
- Processes
- Virtual address space of a process
- Metadata about the Process
  - PCB
  - Kernel stack of the Process
  - Page table

What is a Process?

- Program in execution is know as a process.

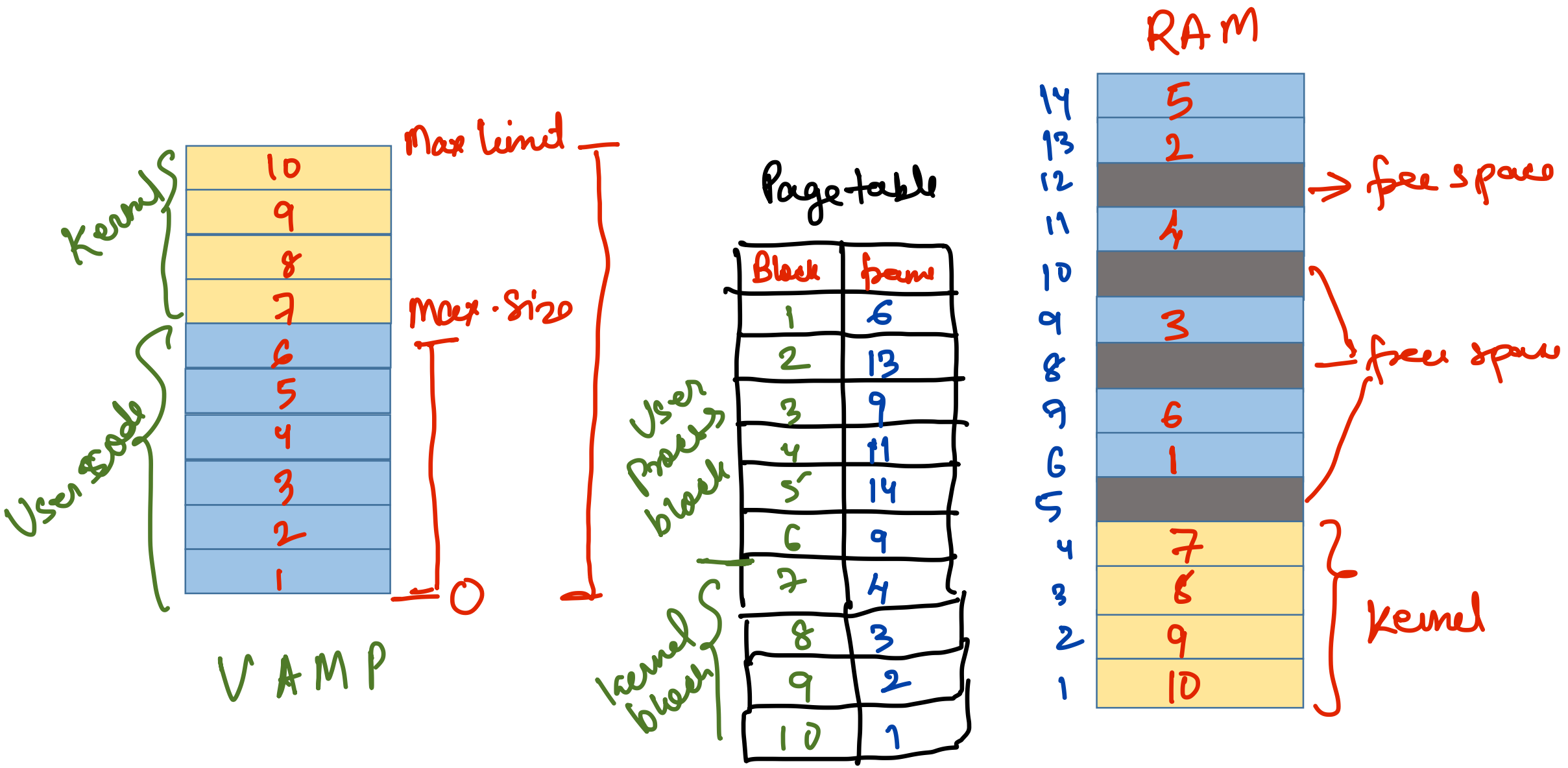


Virtual address space of a process



- When compiled with gcc hello.c it creates an executable a.out.
- When a.out is executed a process is created. Part of this process will be in the RAM and it is identified by a virtual address map.
- So the virtual address map is a **sequence of contiguous addressable** memory locations starting from 0 to a limit of MAX_SIZE.
- So, within this virtual address map we have various aspects of the process including the instructions, global and static data, heap, as well as the stack

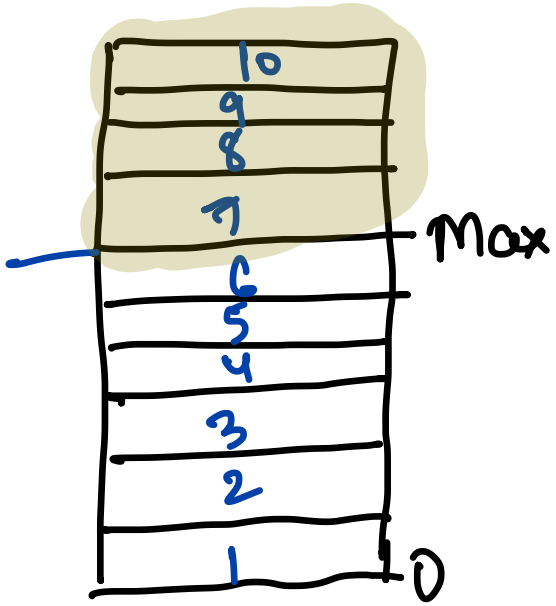
Where does the kernel reside?



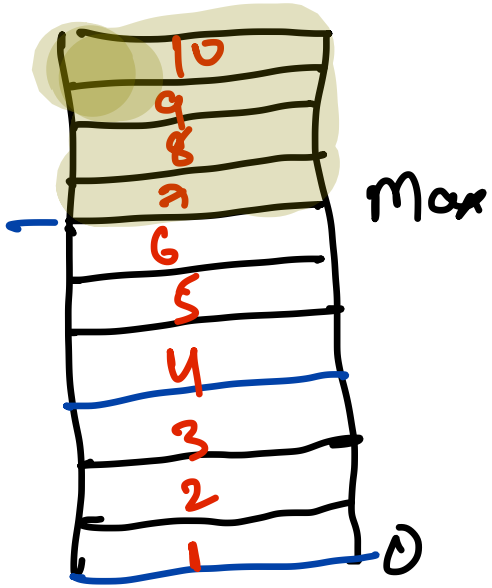
- The kernel resides in the lower part of the memory starting from page frames 1, 2, 3, and so on.

- The virtual address space or virtual address map of a process is divided into equally sized blocks. So typically the size of each block is 4KB.
- Again there is each process would also have a process **page table** in memory (Kernel space) which maps each block of the process into a corresponding page frame.
- The RAM, as we have seen is divided into page frames of size 4 KB similar to the block size. And these page frames contain the actual code and data of the process which is being executed.

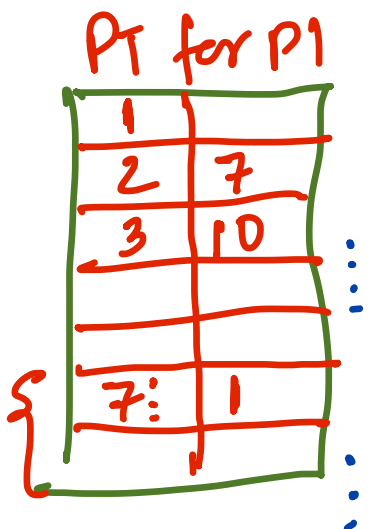
Kernel and multiple process

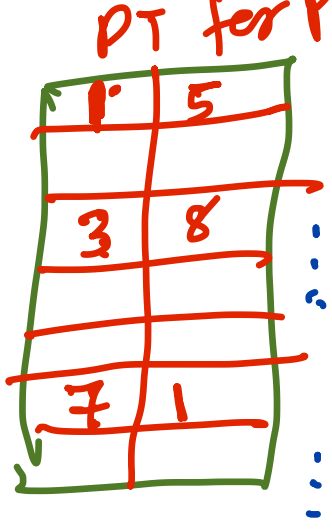


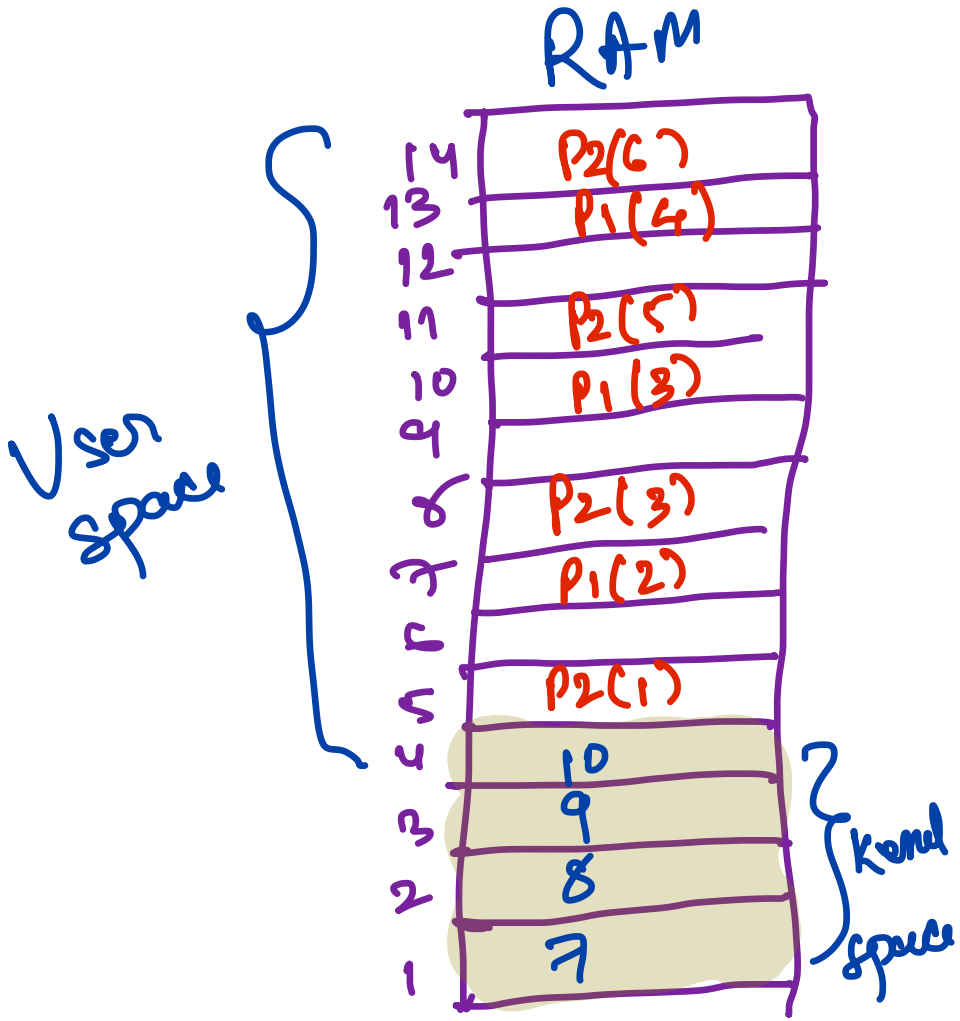
P1



P2





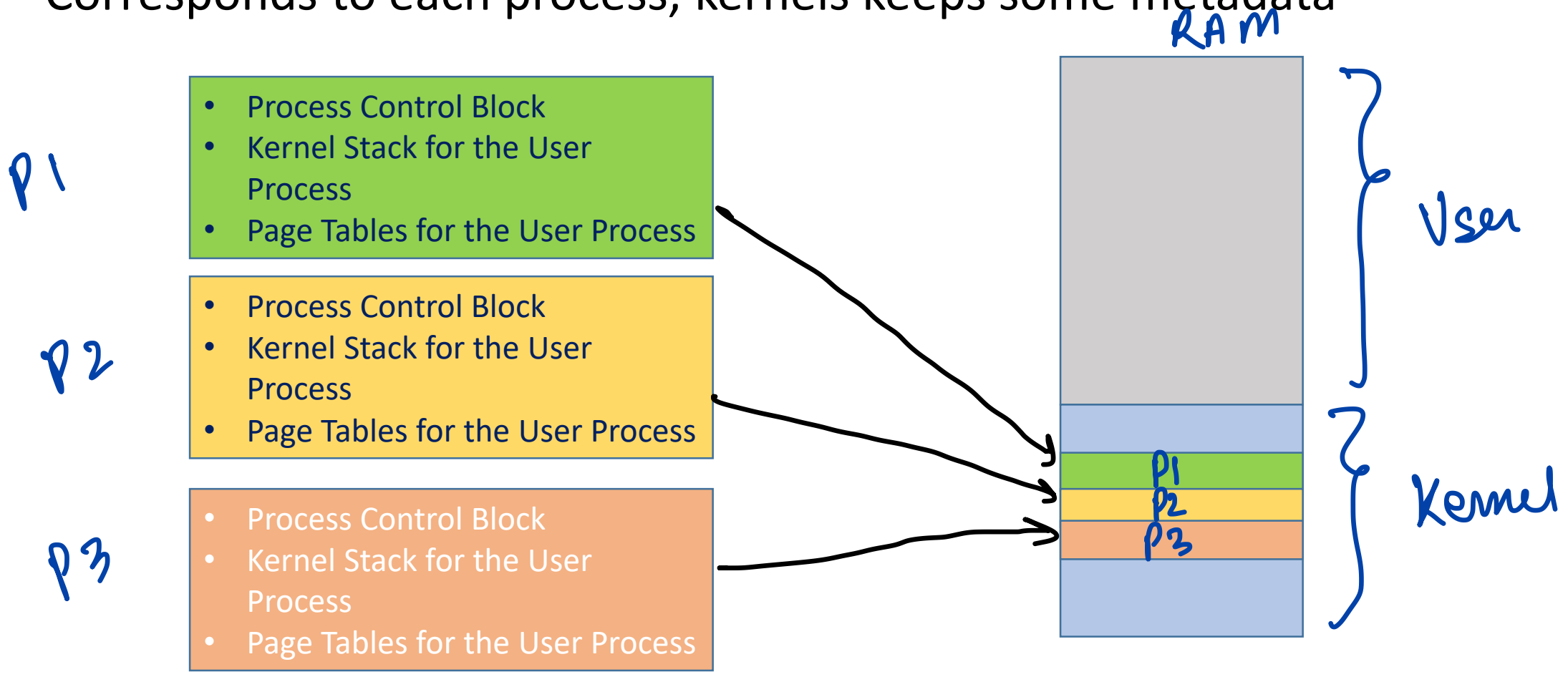


What happens when we have multiple processes in the system?

- The kernel space is mapped identically in all virtual address spaces of every process.
- For instance, above MAX_SIZE and below Max Limit the kernel space is present in all processes.
- Similarly, the page table in each process also has an identical mapping between the kernel page tables and the corresponding page frames that the kernel occupies, as can be seen (in above mentioned slide) .
- The virtual address space of each process has different entries for the kernel.
- However all processes eventually map their kernel space into the same page frames in the RAM. So, what this means is that, we have just **a single copy of the kernel present in the RAM.**
- Thus, there can be identical entries in each processes page table corresponding to the kernel code and data.

Metadata about the process

- Corresponds to each process, kernels keeps some metadata



Kernel Stack for User Process

- kernel stack is used when
  - the kernel executes in the context of a process.
  - to store the context of a process.
- Example: when the process executes a system call (service from the OS) it results in some kernel code executing and these kernel code would use the kernel stack for it's local variables as well as function calls.
- So, why do we have two separate stacks? Why do we have a user stack for the process as well as the kernel stack?
  - the kernel can execute even if the user stack is corrupted.
  - For Example:- Attacks that target the stack, such as buffer overflow attack will not affect the kernel in such a case.

Process Control Block (PCB)

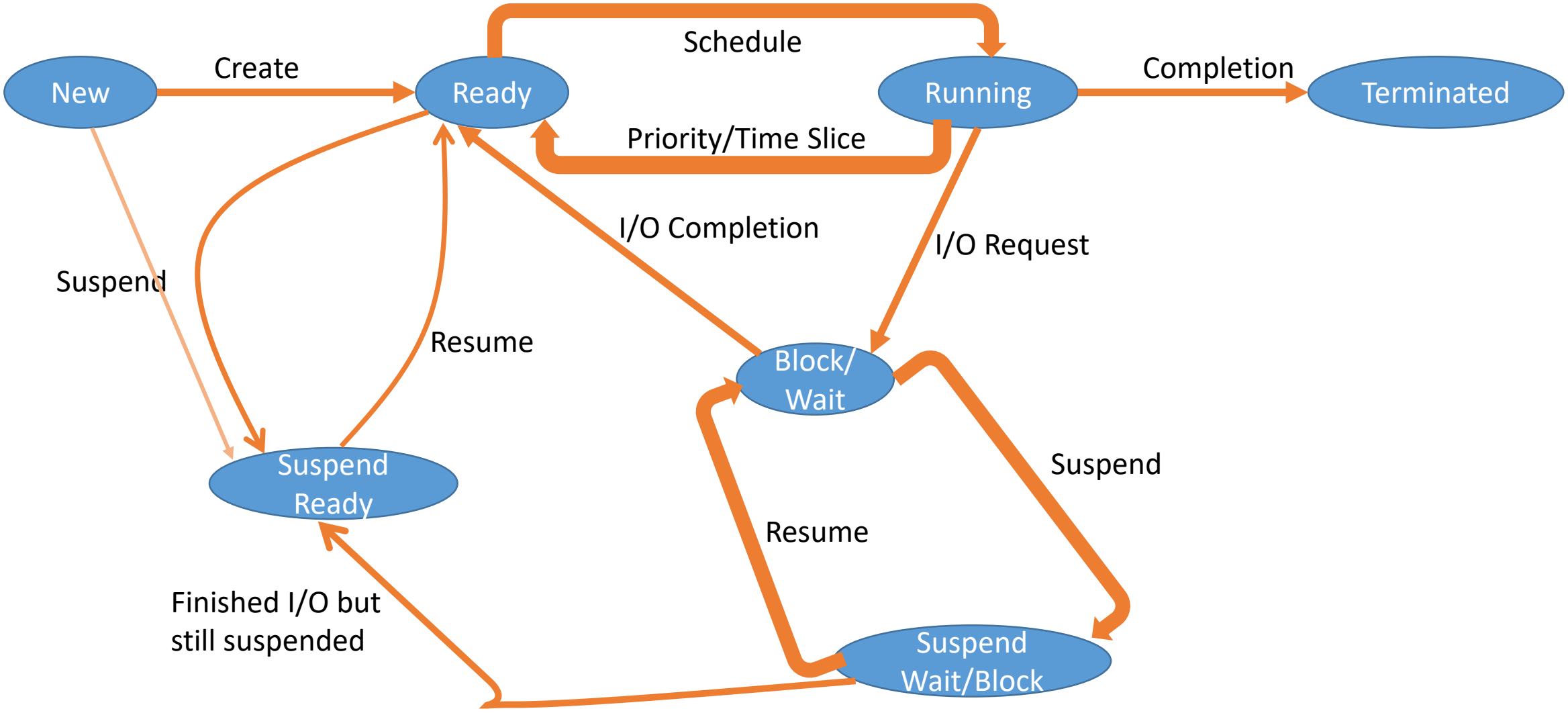
- PCB is a data structure that stores information associated with each process
  - **Process id:** (unique for each process)
  - **Process state :–** It stores the respective state of the process.
  - **Process number: –** Every process is assigned with a unique id known as process ID or PID which stores the process identifier.
  - **Program counter; –** It stores the value which contains the address of the next instruction that is to be executed for the process.
  - **Register: –** These are the CPU registers which includes: accumulator, base, registers and general purpose registers.
  - Memory management info:- Size of process memory ,Page Table, Memory Limit, …)
  - List of opened files
  - List of opened devices
  - Accounting and status data – This field includes information about the amount of CPU used, time constraints, jobs or process number, etc.

What is the use of PCB?

- Context Switching: PCBs are essential for context switching, where the operating system saves the state of the currently running process and loads the state of the next scheduled process. This allows multiple processes to share the CPU effectively.
- Process Management: By maintaining all necessary information about processes, PCBs enable efficient process creation, termination, and scheduling to support multiprogramming.
- System Stability and Efficiency: PCBs help ensure that processes are managed in an organized manner, contributing to the stability and efficiency of the operating system.

Process State

- From the time process is created till it finishes, it passes through several states.
  - New
  - Ready
  - Running
  - Wait (Block)
  - Terminated (Completed)
  - Suspend Ready
  - Suspend Wait (Block)


New
Create
Ready
Schedule
Running
Completion
Terminated
Priority/Time Slice
I/O Completion
I/O Request
Suspend
Resume
Block/
Wait
Suspend
Ready
Suspend
Resume
Finished I/O but
still suspended
Suspend
Wait/Block

- **New:** Newly Created Process (or) being-created process.
  - In this step, the process is about to be created but not yet created, it is the program which is present in secondary memory that will be picked up by OS to create the process.
- **Ready:** After creation process moves to Ready state, i.e. the process is ready for execution.
  - New -> Ready to run. After the creation of a process, the process enters the ready state i.e. the process is loaded into the main memory. The process here is ready to run and is waiting to get the **CPU time** for its execution. Processes that are ready for execution by the CPU are maintained in a queue for ready processes called READY QUEUE.
- **Running:** Currently running process in CPU (only one process at a time can be under execution in a single processor).
  - The process is chosen by CPU for execution and the instructions within the process are executed by any one of the available CPU cores.
- **Wait (or Blocked):** When a process requests I/O access.
  - Whenever the process requests access to I/O or needs input from the user or needs access to a critical region (the lock for which is already acquired) it enters the blocked or wait state. The process continues to wait in the main memory and does not require CPU. Once the I/O operation is completed the process goes to the **ready state**.

- **Complete (or Terminated):** The process completed its execution. Process is killed as well as PCB is deleted.
- **Suspend Ready:** When the ready queue becomes full(that means no sufficient memory is available for new process), then some processes are moved to **suspend ready** state to give space at RAM for new high priority process
  - Process that was initially in the ready state but were swapped out of main memory and placed onto external storage by scheduler are said to be in **suspend ready state**.
  - The process will transition back to **ready state** whenever the process is again brought onto the main memory.
- **Suspend Block:** When all processes is in waiting queue, then some of the waiting processes are moved to **suspended block** to give space at RAM for new processes or high priority process
  - Similar to suspend ready but remove the process which was waiting to perform I/O operation
  - When work is finished it may go to suspend ready or blocked state

Process Scheduling Queues

- **Three** types of queue
- **Job queue** – set of all processes in the system which are ready to come to main memory, maintain in secondary memory
- **Ready queue** – set of all processes residing in main memory, ready and waiting to execute in processor, maintain in main memory
- **Device queues** – set of processes waiting for an I/O device, maintain in main memory
- Suspend block and suspend ready queue is maintained in secondary memory

Processes migrate among the various queues

Process Scheduler

- Brings a set of processes to main memory
- Handles the removal of the running process from the CPU and the selection of another process from ready queue

CPU-Bound vs I/O-Bound Processes

- A CPU-bound process requires more CPU time
  - spends more time in the running state.
- An I/O-bound process requires more I/O time and less CPU time.
  - spends more time in the wait state.

Types of Process Scheduler.

Long Term or job scheduler :

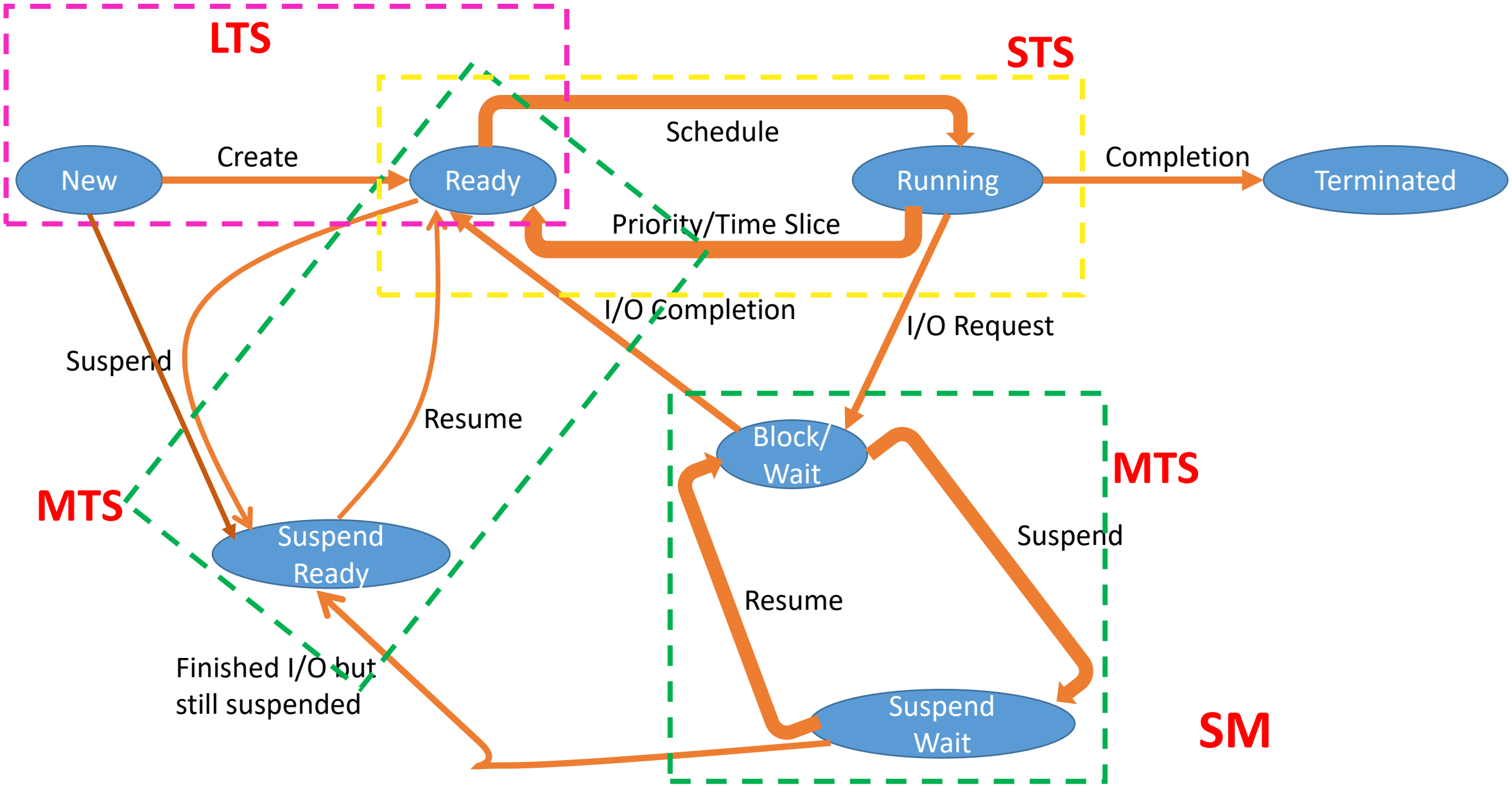
- It brings the new process to the ‘Ready State’.
- It controls ***Degree of Multi-programming***, i.e., number of process present in ready state at any point of time
- A careful selection of both IO and CPU bound process.
- The job scheduler increases efficiency by maintaining a balance between the two.

Short term or CPU scheduler :

- It is responsible for selecting one process from ready state for scheduling it on the running state.
- Note: Short-term scheduler only selects the process to schedule it doesn’t load the process on running.
- The CPU scheduler is responsible for ensuring there is no starvation owing to high burst time processes.
  Dispatcher is responsible for loading the process selected by Short-term scheduler on the CPU (Ready to Running State)
- Context switching is done by dispatcher only.

Medium-term scheduler :

- It is responsible for suspending and resuming the process.
- It mainly does swapping (moving processes from main memory to disk and vice versa).
- It is helpful in maintaining a perfect balance between the I/O bound and the CPU bound.
- It reduces the degree of multiprogramming


LTS
STS
Schedule
Create
New
Ready
Running
Completion
Terminated
Priority/Time Slice
I/O Completion
I/O Request
Suspend
Resume
MTS
Block/
Wait
MTS
Suspend
Ready
Suspend
Resume
Finished I/O but
still suspended
Suspend
Wait
SM

Context Switching

- The process of saving the context of one process and loading the context of another process.

- When does context switching happen?
  - When process moves from Running to ready state
    - Preemptive scheduling used
      - When a high-priority process comes to a ready state (higher priority than the running process)
    - When time slice expires

  - When process moves from running to block state for I/O operation
  - When process moves from running to terminate (no need to save state of running process, but need to load state of new process from ready to running)