Process Management

Prof J P Misra
BITS, Pilani

Program & Process

- Program is set of logically arranged instruction to perform a specific task.
- Program is written using some kind of language and compiled to produce machine executable code.
- Program is passive entity containing list of instruction stored on disk

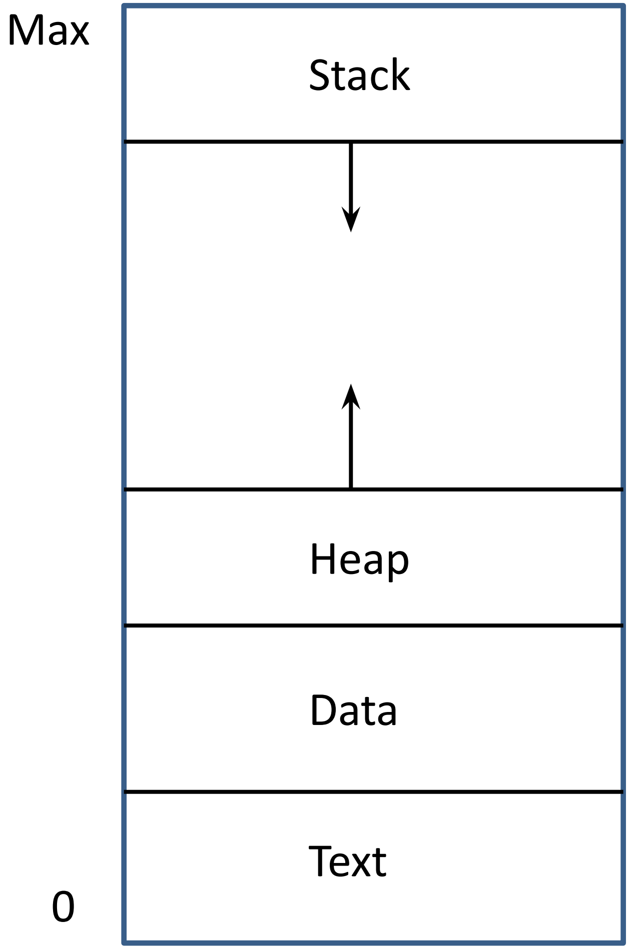
Process Management

- A process is a program in execution. It is a unit of work within the system.
- Program is a *passive entity*, process is an *active entity*.
- Process needs resources to accomplish its task
  - CPU, memory, I/O, files
  - Initialization data
- Process termination requires reclaim of any reusable resources
- Single-threaded process has one **program counter** specifying location of next instruction to execute
  - Process executes instructions sequentially, one at a time, until completion
- Multi-threaded process has one program counter per thread
- Typically system has many processes, some user, some operating system running concurrently on one or more CPUs
  - Concurrency by multiplexing the CPUs among the processes / threads

Process Management

- Needs to be loaded into memory for execution
- Processor Sequentially executes the instructions one by one until it encounters a branch instruction
- Process is an instance of executing program
- Process is characterized by
  - Its code, Data, stack , Heap and set of register

Process in memory



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

Process creation

- Process Is created
  - When a new job is submitted
  - When a user attempts to login
  - to provide a service
- Process can be spawned by existing process

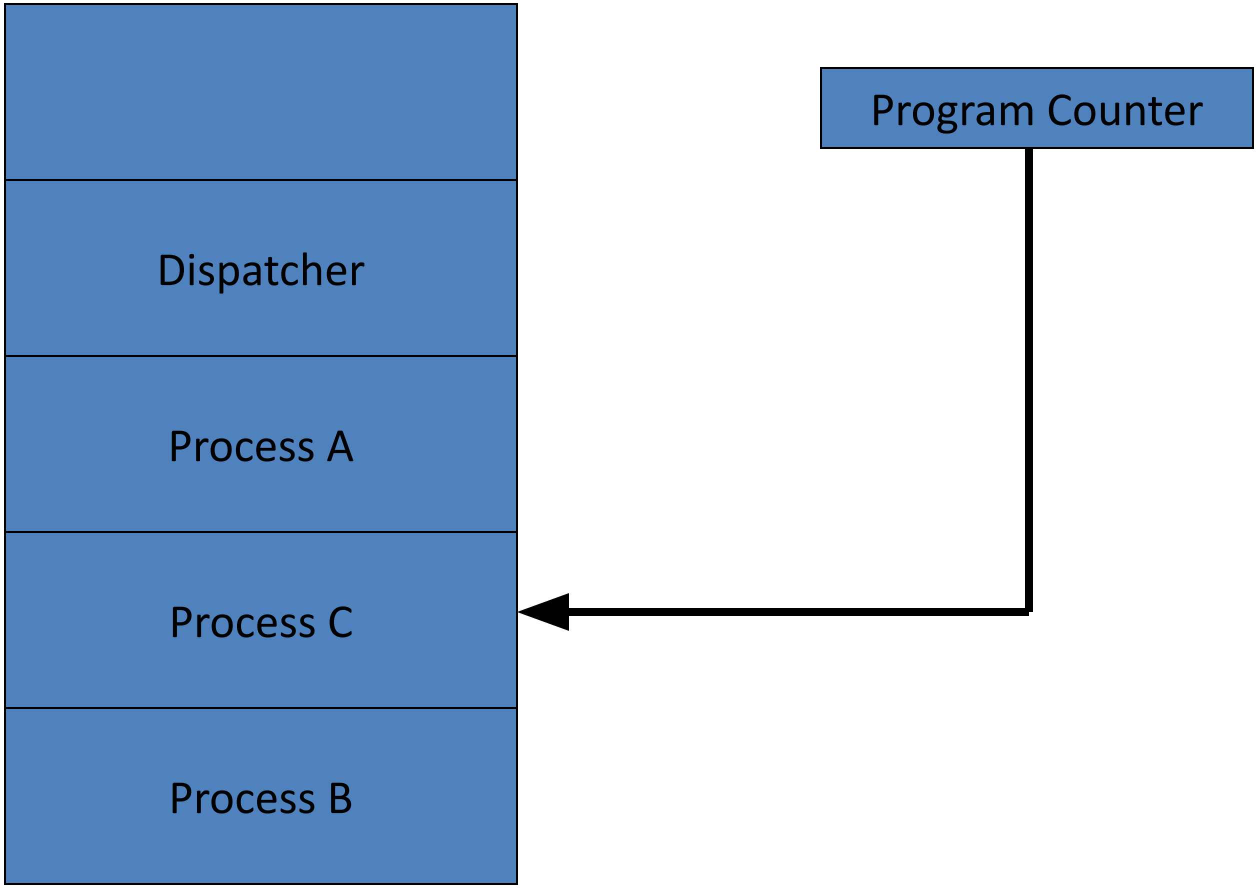
How a program/job is executed ?

- For executing a job , OS first creates data structure for holding the context of process
- Loads the job in memory
- At some point of time the scheduler schedules the job and it starts executing.
- Dispatcher process dispatches ready to execute process for execution
  - once executing process terminates,
  - its time slice expires
  - Executing process makes I/O request and gets blocked
- Dispatcher is a system process

Process & Multiprogramming

- In multiprogramming environment, many jobs can be in memory
- Jobs in memory are ready to execute
- At any point of time one Job would be in running state and other jobs would be in not running state.

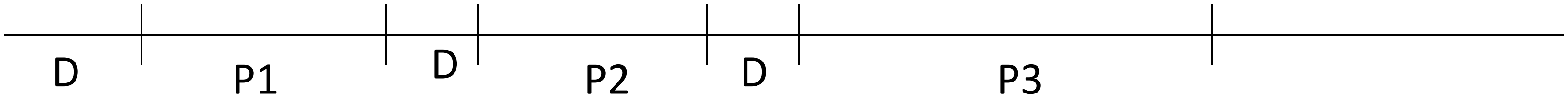
Memory Layout



Execution Of Ready Processes

P1, P2, P3 are user processes

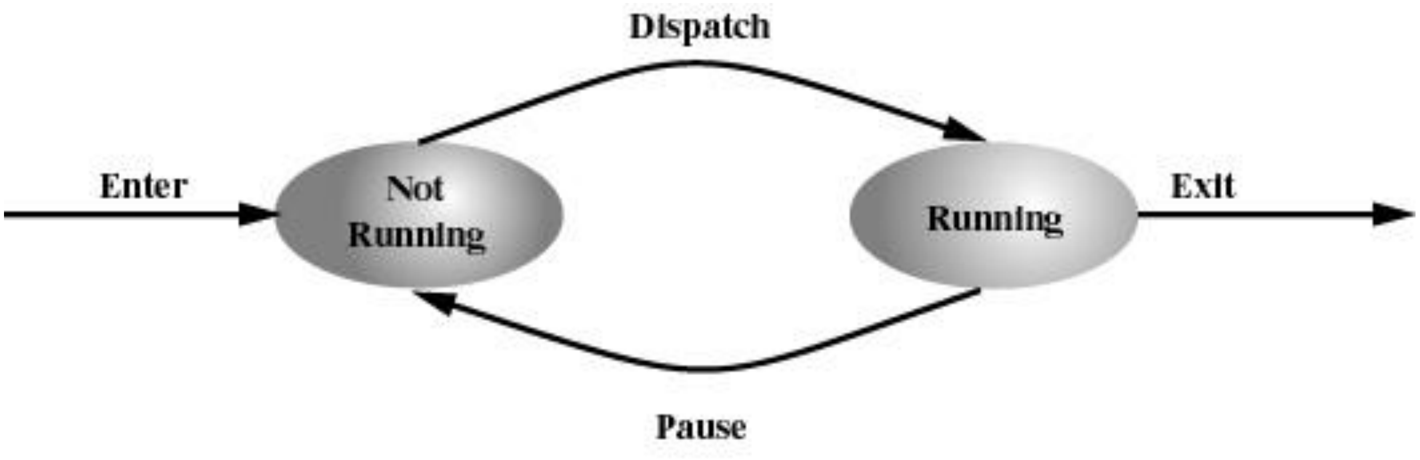
D is Dispatcher Process. It takes constant time to dispatch



- Computing system can be considered as assemblage of set of processes.
- The processes can be system processes or user processes
- At any given time the processes can be in different states.
- The system can be modeled with the help of state transition diagram

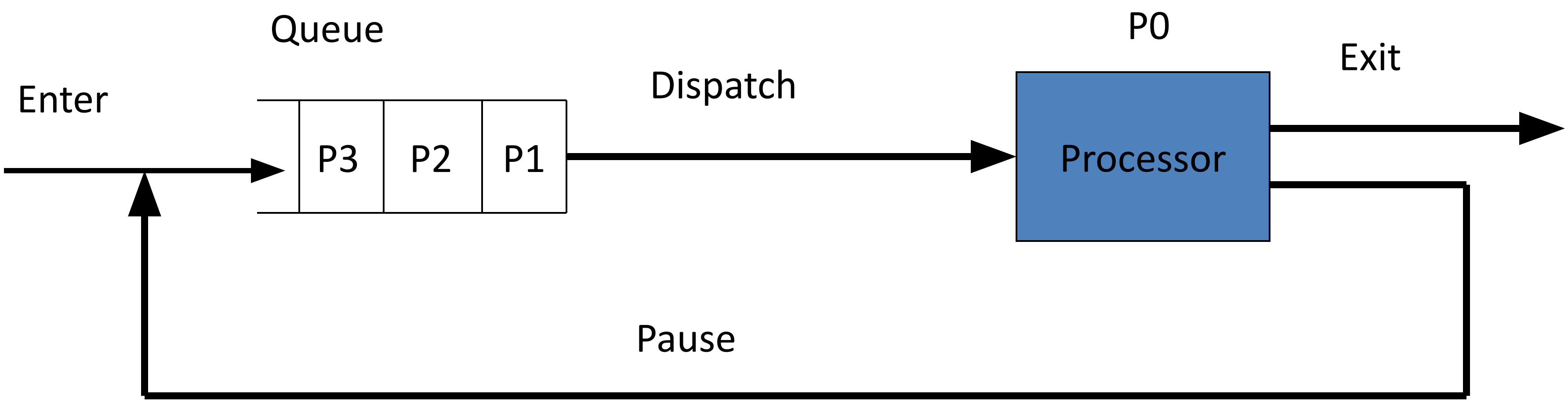
Two State Process Model

- Process may in any of the 2 states
  - Running
  - Not running



(a) State transition diagram

Queuing Diagram



Queue May contain ready and blocked processes

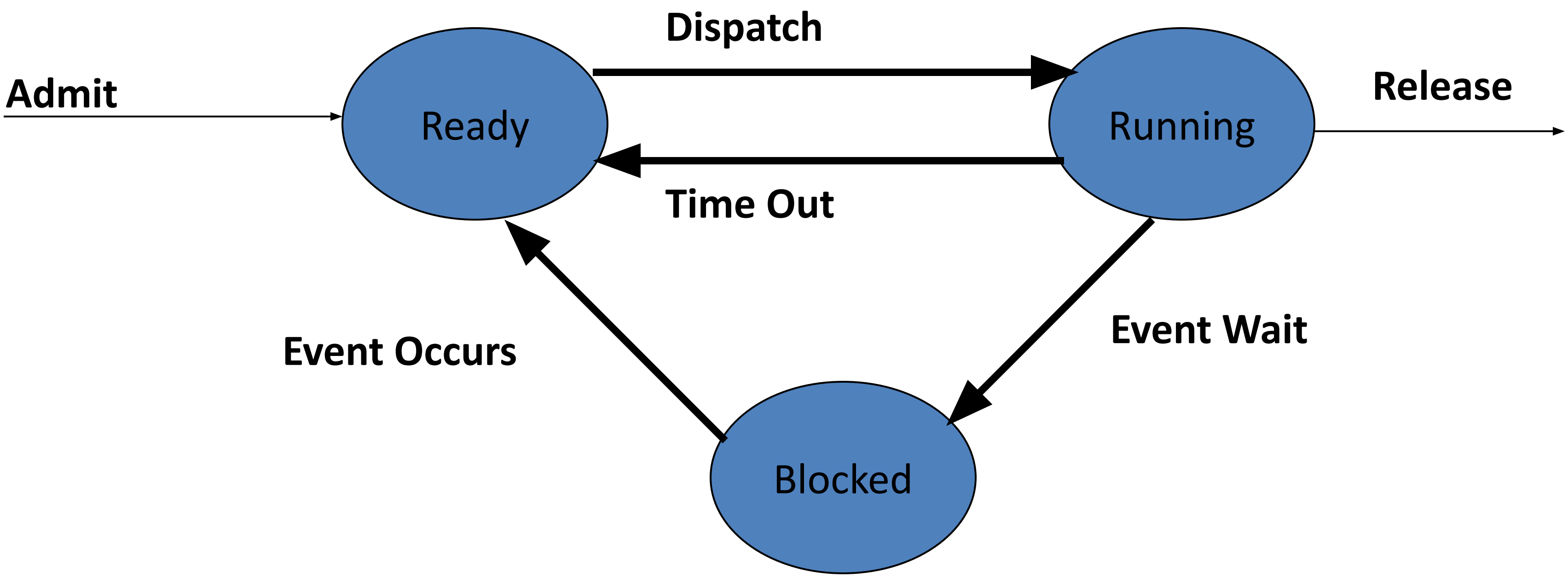
State Transition

- Not running to running state , transition occurs when
  - process in running state finishes execution
  - Makes an I/O request
  - Time slice for executing process expires
- Running to not running state transition occurs when
  - Running process makes an I/O request
  - Time slice of executing process expires

Dispatcher

- Dispatcher schedules a Ready process to CPU for execution.
- In Two State model Not running state contains processes which are :
  - Ready to run
  - Blocked
- Dispatcher is required to linearly search the queue to find Ready to run process which Increases dispatcher overhead.
- Solution : Split not running state into Ready state & Blocked state

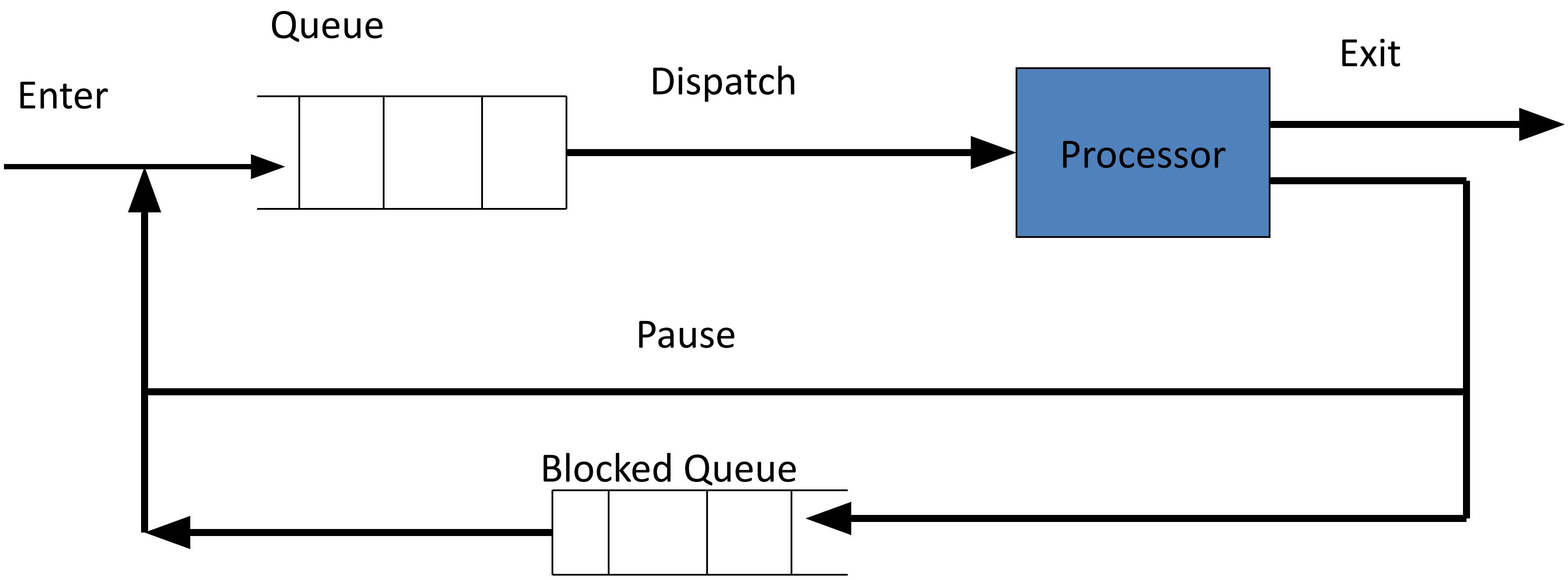
Three State Model



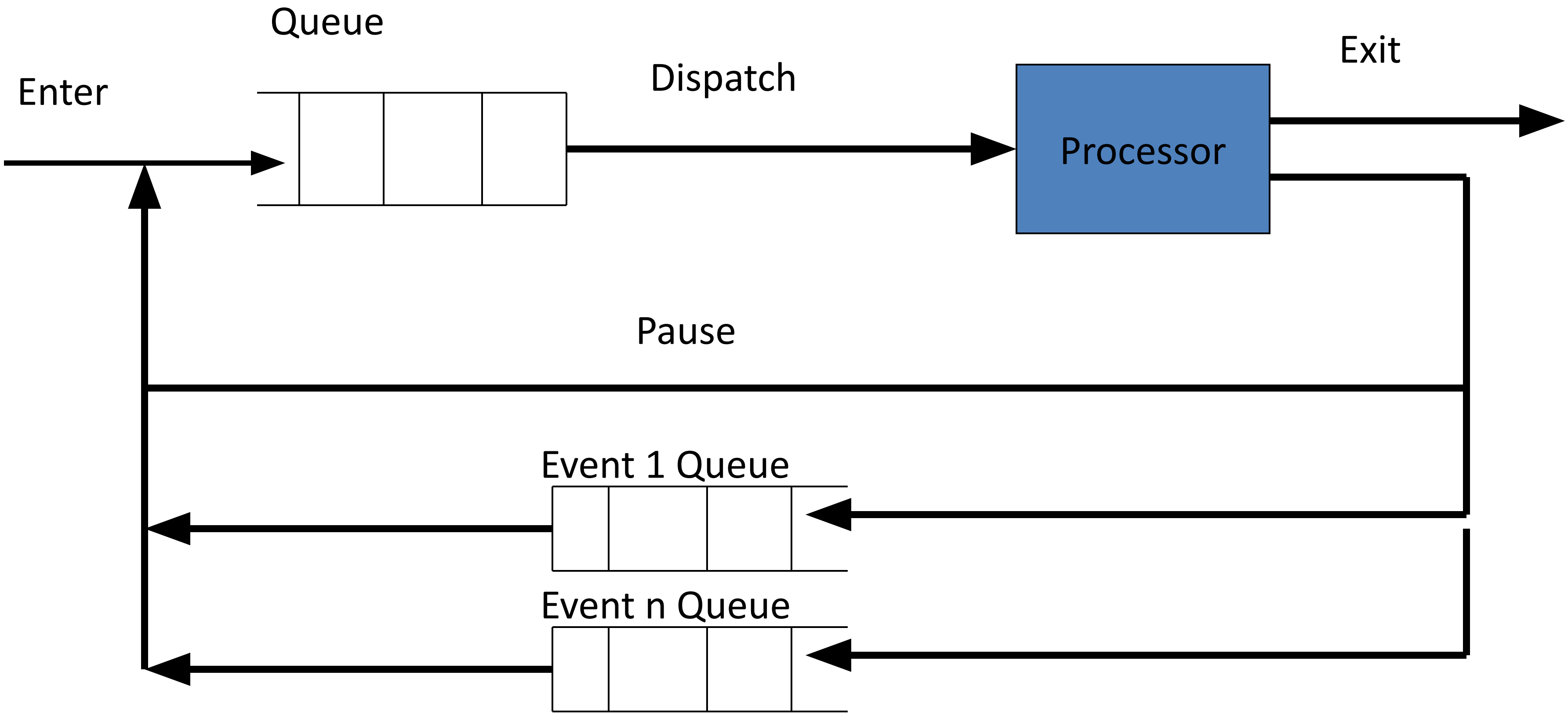
Process States

- **Running**: The process that is currently being executed
- **Ready**: A process that is prepared to execute when given the opportunity
- **Blocked** : A process can not execute until some events occur
- Occurrence of event is usually indicated by interrupt signal

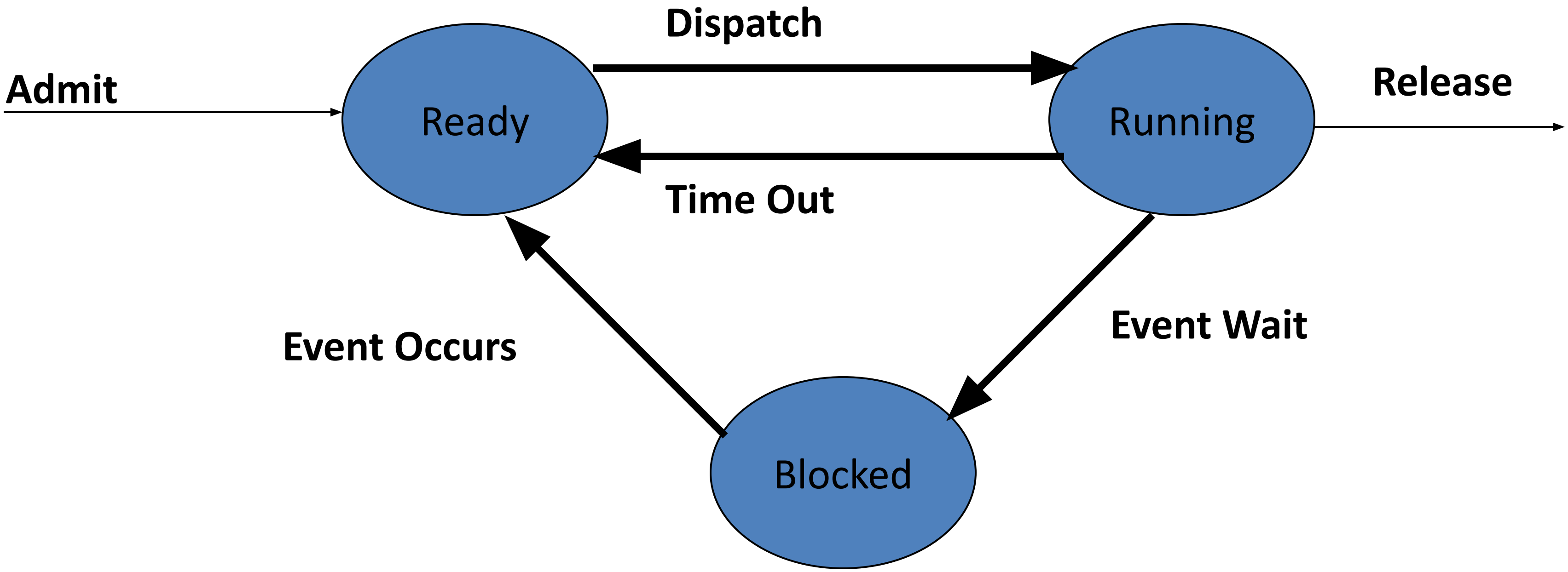
Queuing Diagram



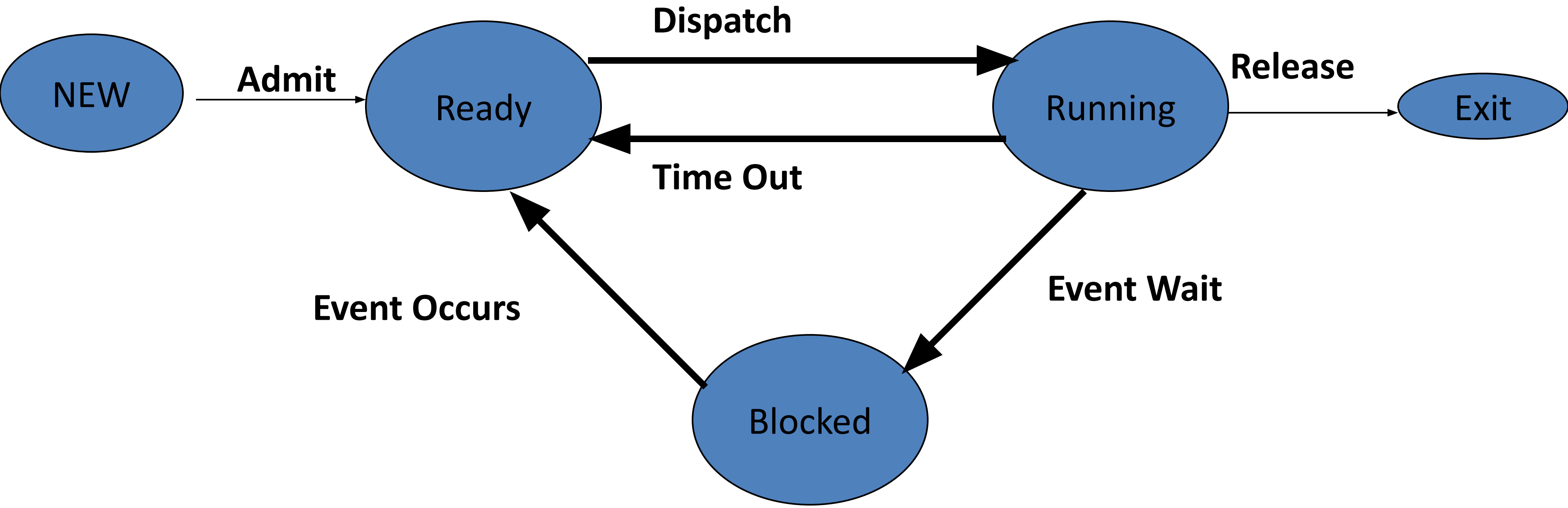
Queuing Diagram



Three State Model



Five State Model



Why do we need “New” state

- Whenever a job is submitted , OS creates data structure for keeping track of the process context and then it tries to load the process. While loading the process
  - System May not have enough memory to hold the process
  - To efficiently execute processes, system may put a maximum limit on processes in ready queue

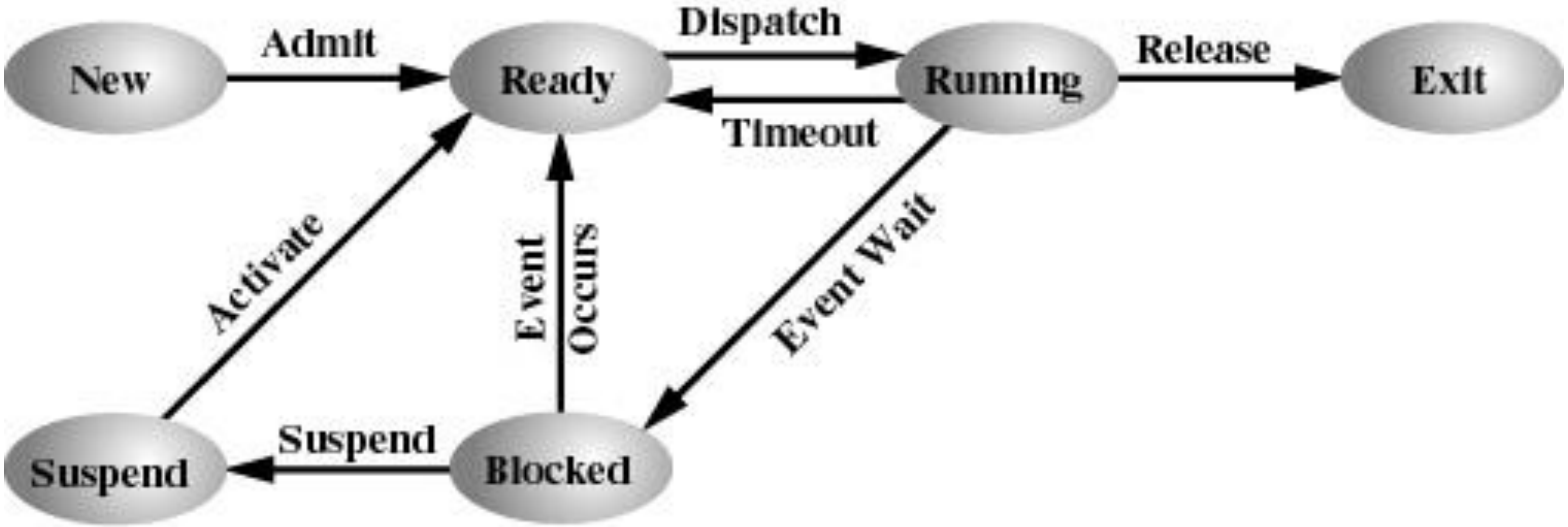
Consider a scenario

All ready processes get blocked on I/O one by one, the system tries to bring in a process from new to ready state and it is found no memory is available to accommodate this process

Q. What do we do ?

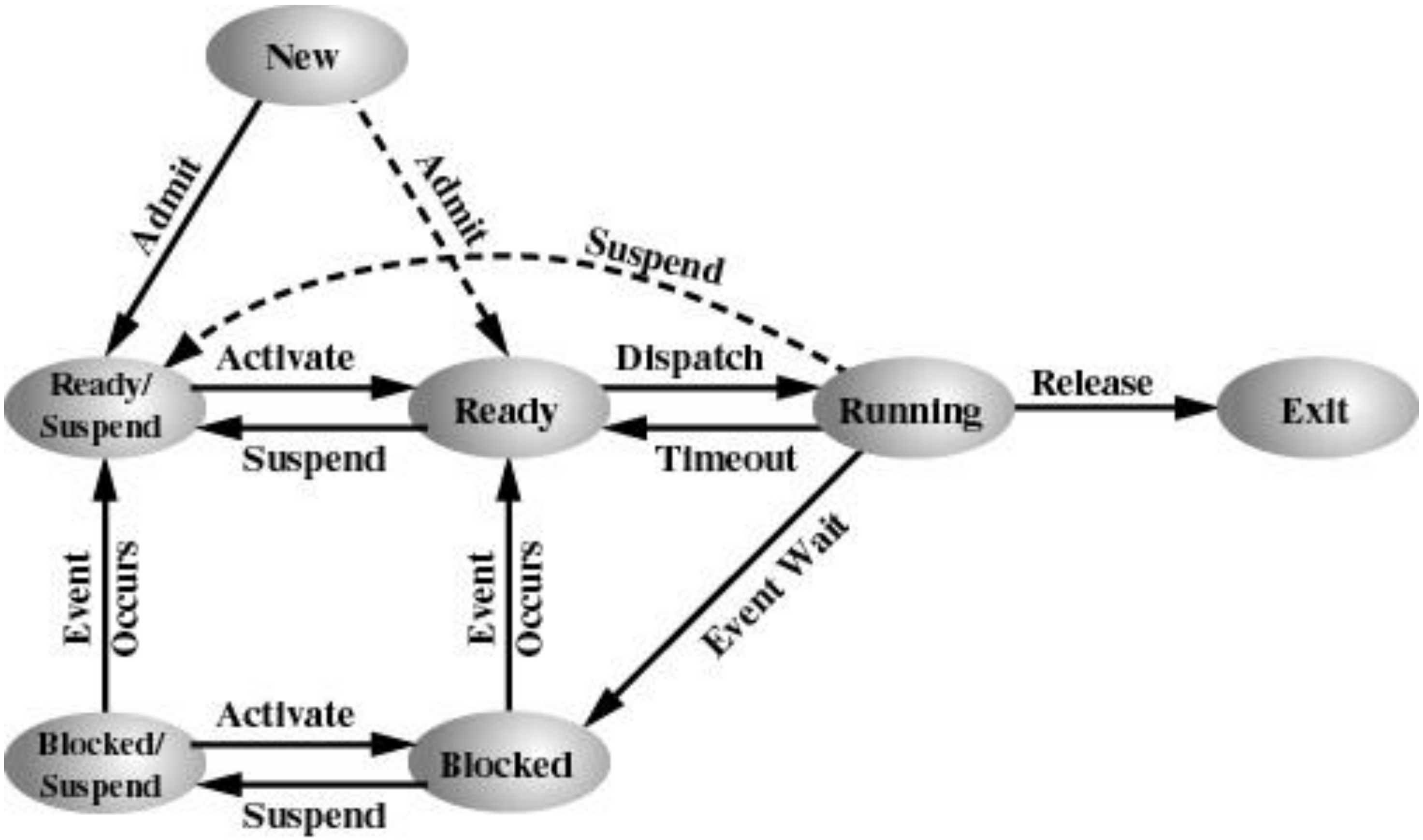
Ans. Swapping

One Suspend State



(a) With One Suspend State

Two Suspend State



(b) With Two Suspend States

State Transitions

- Blocked → Blocked/suspended:
  - If Ready queue is empty and insufficient memory is available then one of the blocked process can be swapped out
  - If currently running process requires more memory

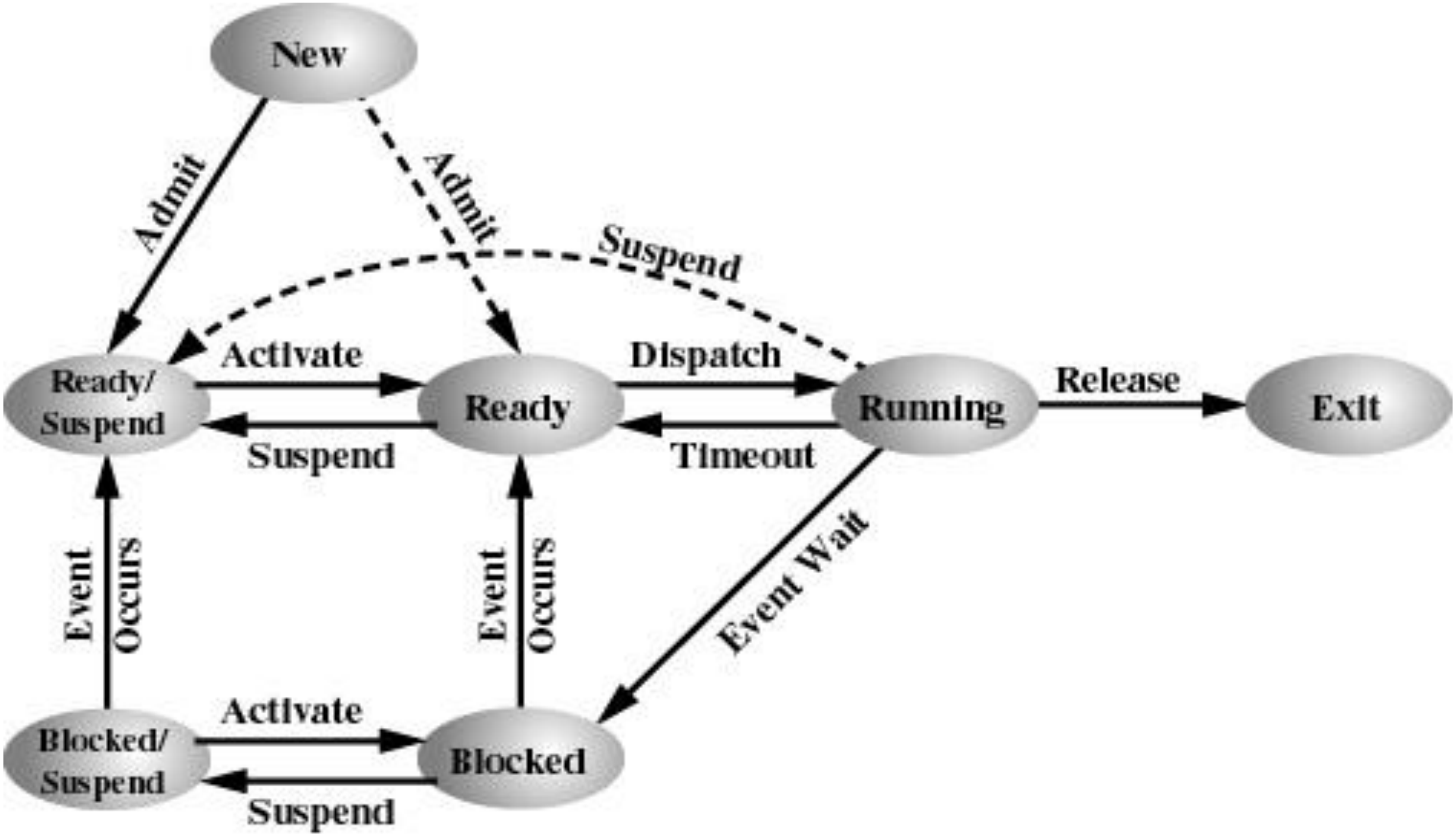
State Transitions

- Blocked/ Suspended → Ready/suspended
  - When the event for which process has been waiting occurs
- Note
  - State information concerning to suspended process must be accessible to the OS.
- Ready-suspended → Ready
  - If Ready queue is empty
  - If Process in Ready-suspended state has higher priority than process in Ready state

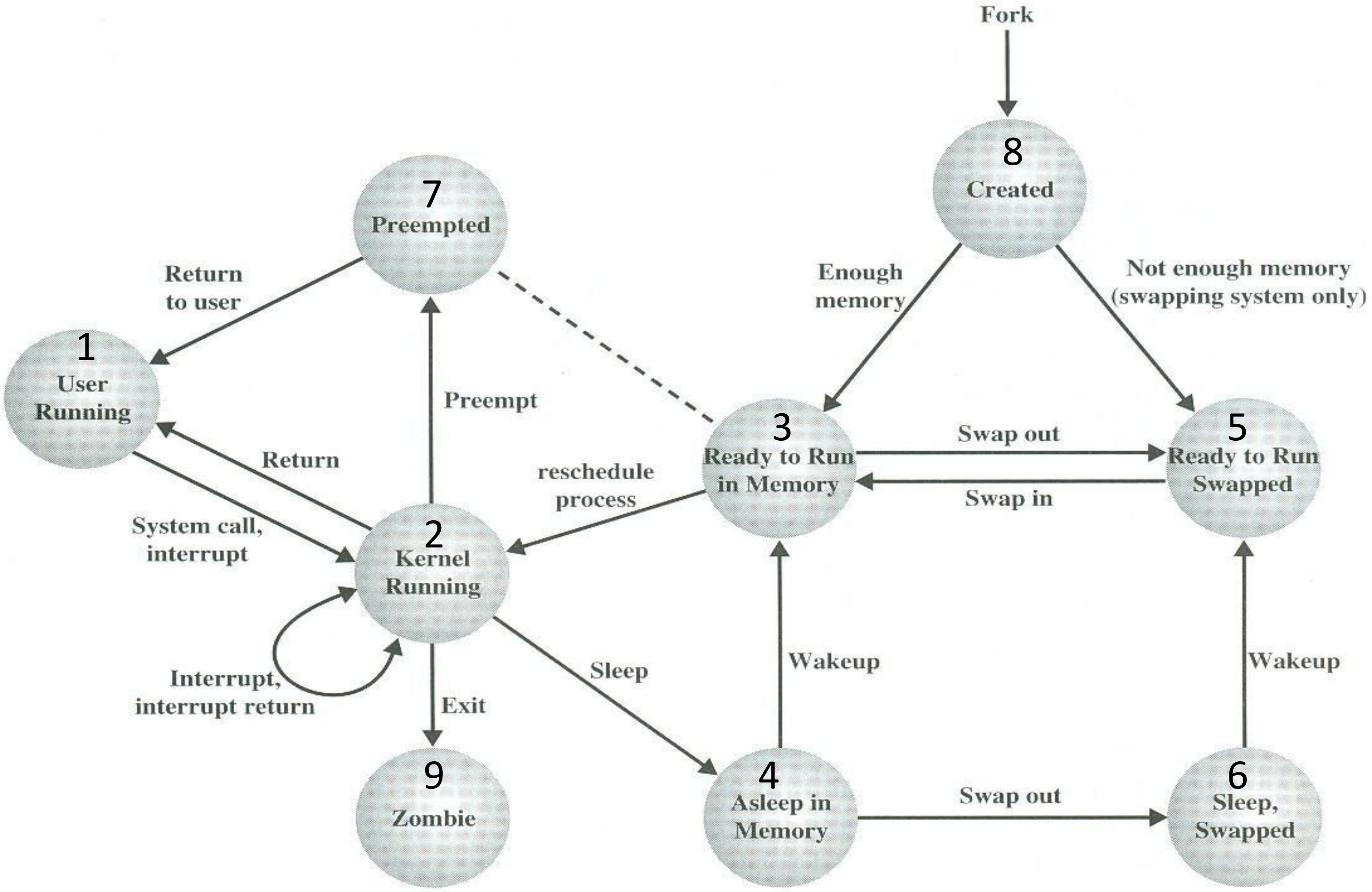
State Transitions

- Ready → Ready/Suspend
  - Normally a blocked process is suspended
  - Suspend Ready process if it is the only way to free memory
  - Suspend a lower priority ready process than higher priority blocked process ?
- Blocked Suspended → Blocked
- Running → Ready suspended
- Various → Exit

Seven state process state transition Diagram



Unix Process State Transition Diagram



UNIX Process State Transition Diagram

Unix System V

- Unix uses two categories of process
  - **System Process** :
    - Runs in kernel mode and
    - performs administrative and house keeping functions such as memory allocation, process swapping, scheduling etc.
  - **User Process** :
    - Runs in user Mode to execute user program and utilities
    - Runs in kernel mode to execute instructions belonging to kernel
- A user process enters in kernel mode
  - by issuing a system call or
  - exception / interrupt

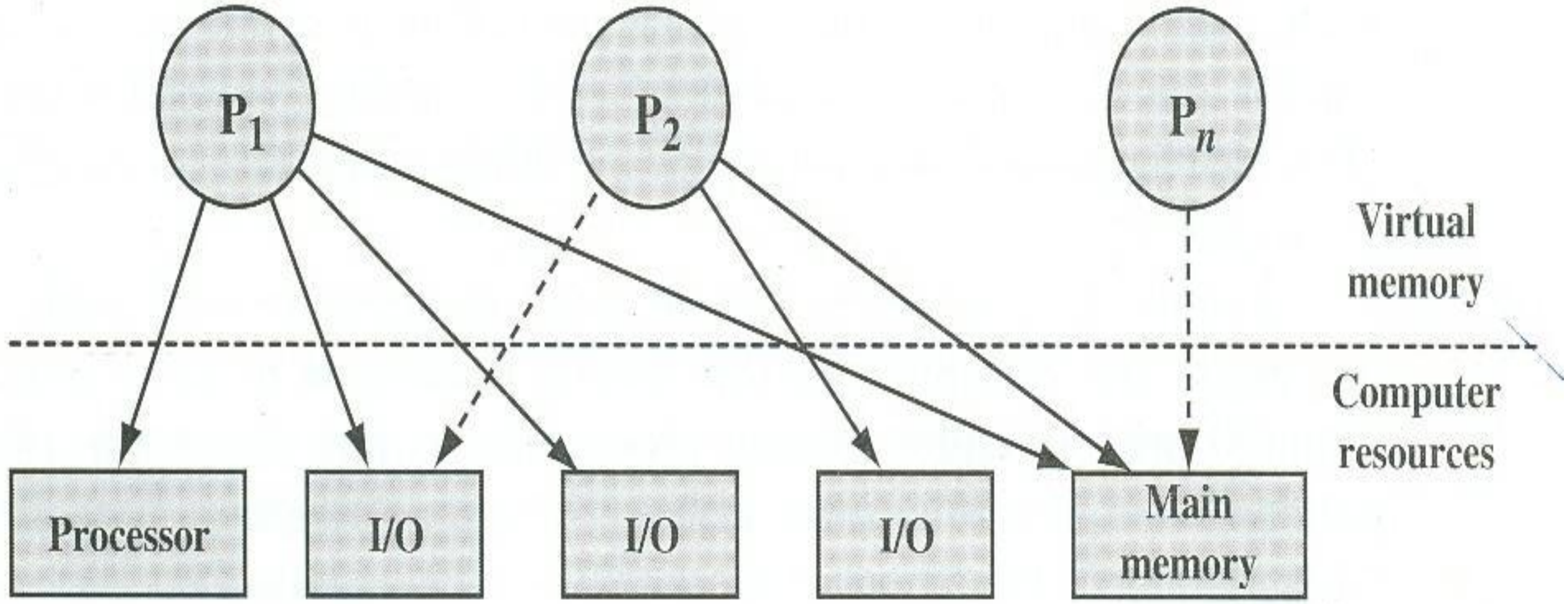
Process 0 & Process 1

- Process 0 (Swapper Process) is predefined as data structure, loaded at boot time .
- Process 0 spawns process 1
- Process 1(Init Process) is ancestor of all other processes except process 0

Process Summary

- Process is an instance of executing program
- In multiprogramming environment number of process can reside in system
- At any instance of time processes can be in different state such as ready, Blocked, Running etc. and processes move from one state to another state depending upon certain conditions.
- Processes use available system resources
- OS is responsible for managing all the processes in the system

Process & Resource at some instance of time



Process & resource snapshot at some instance of time

OS as Process Manager

- OS must have information about
  - the current state of processes
  - the current state of Resources in the system
- OS must keep track of utilization of resources by processes
- OS must constructs tables (control structure) to maintain information about each entity it is managing

OS as Process Manager

- To manage processes, OS maintains following tables
  - Memory Tables
  - I/O Table
  - File Table
  - Process table

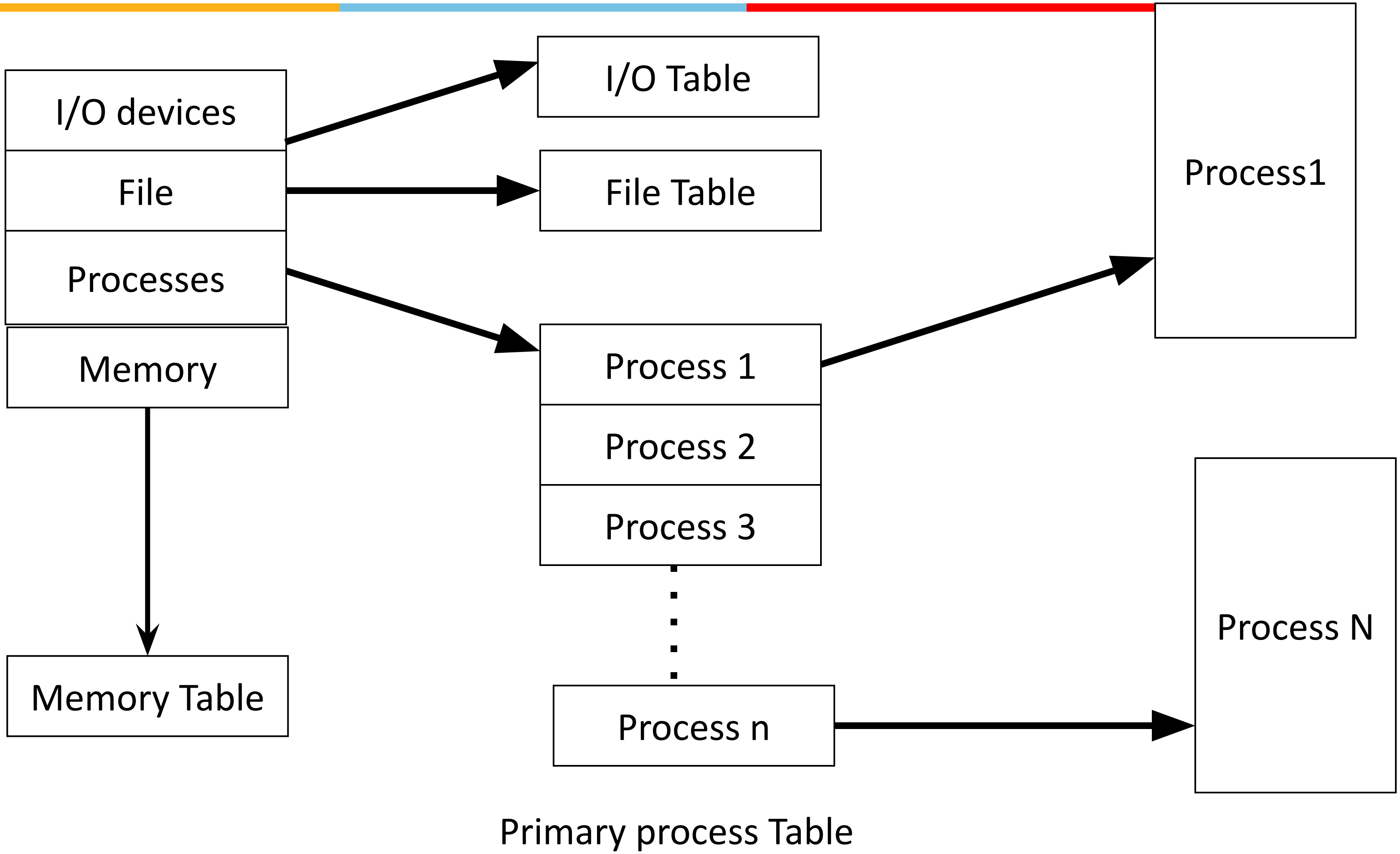
Memory Table & I/O Table

- **Memory Tables keep track**
  - of allocation of main memory to processes
  - of Allocation of secondary memory to processes
  - Of protection attributes of main & secondary memory such as which processes can access certain shared memory region
  - Information needed to manage ***virtual*** memory
- **I/O Table:** To keep track of
  - resource allocation,
  - resource availability
  - I/O request

File Table & Process Table

- **File Table**
  - Provide information about
    - Existence of file
    - Location on secondary memory
    - Current status and attributes of file
- **Process table**
  - keep track of processes

Operating Systems Control table



Process Image

- Process image consists of
  - User Data
  - User program
  - System Stack
  - Process control block
    - Containing processes attributes

Process Management

- To manage/control a process OS must know
  - Where process is located
  - Attributes of processes eg. Process id, state etc.
  - Collection of attributes are held in Process control block

Process Attributes

- Process identification
- Processor State Information
- Processes control information

Process Identification

- Identifier of the process
- Identifier of the process that created this process
- User identifier

Processor State Information

- User Visible registers
- Control and status registers
  - Program counter
  - Flags
  - Status register
- Stack pointer

Processes Control Information

- Scheduling and state information
- Inter process communication
- Process privileges
- Memory management
- Resource ownership & utilization

Functions of OS kernel

- Process Management
  - Process creation & termination
  - Process scheduling & dispatching
  - Process switching
  - Process synchronization & support for IPC
  - Management of process control block
- Memory Management
  - Allocation of address space to process
  - Swapping
  - Page & segment management

Functions of OS kernel Cont…

- I/O Management
  - Buffer management
  - Allocation of I/O channels & devices to processes
- Support Services
  - Interrupt handling
  - Accounting
  - monitoring

Thank you