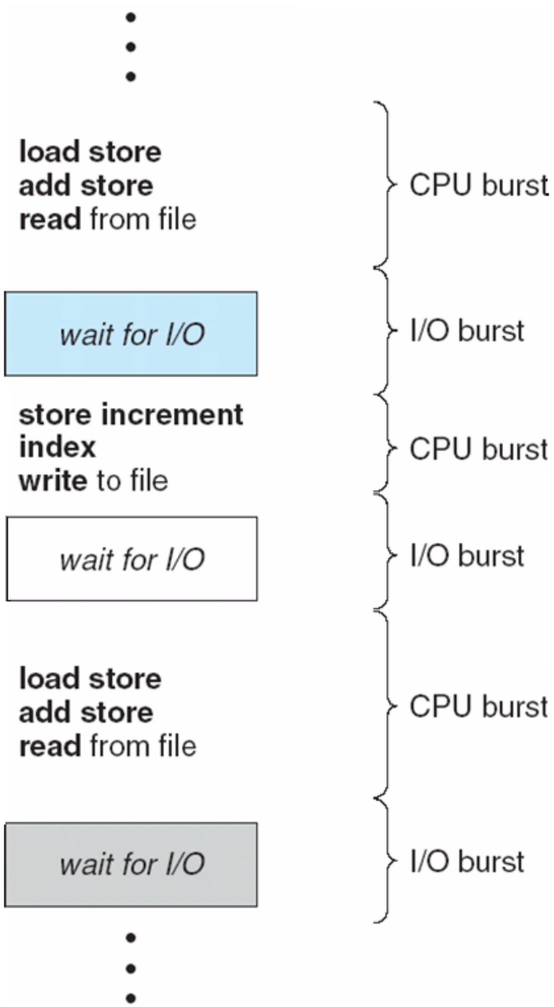
CPU Scheduling (I)

CPU Scheduling



- Decide which process should be run on the CPU
- Purpose: To maximize CPU utilization obtained with multiprogramming/multitasking
- Model of a process: CPU–I/O Burst Cycle – Process execution consists of a *cycle* of CPU execution and I/O wait

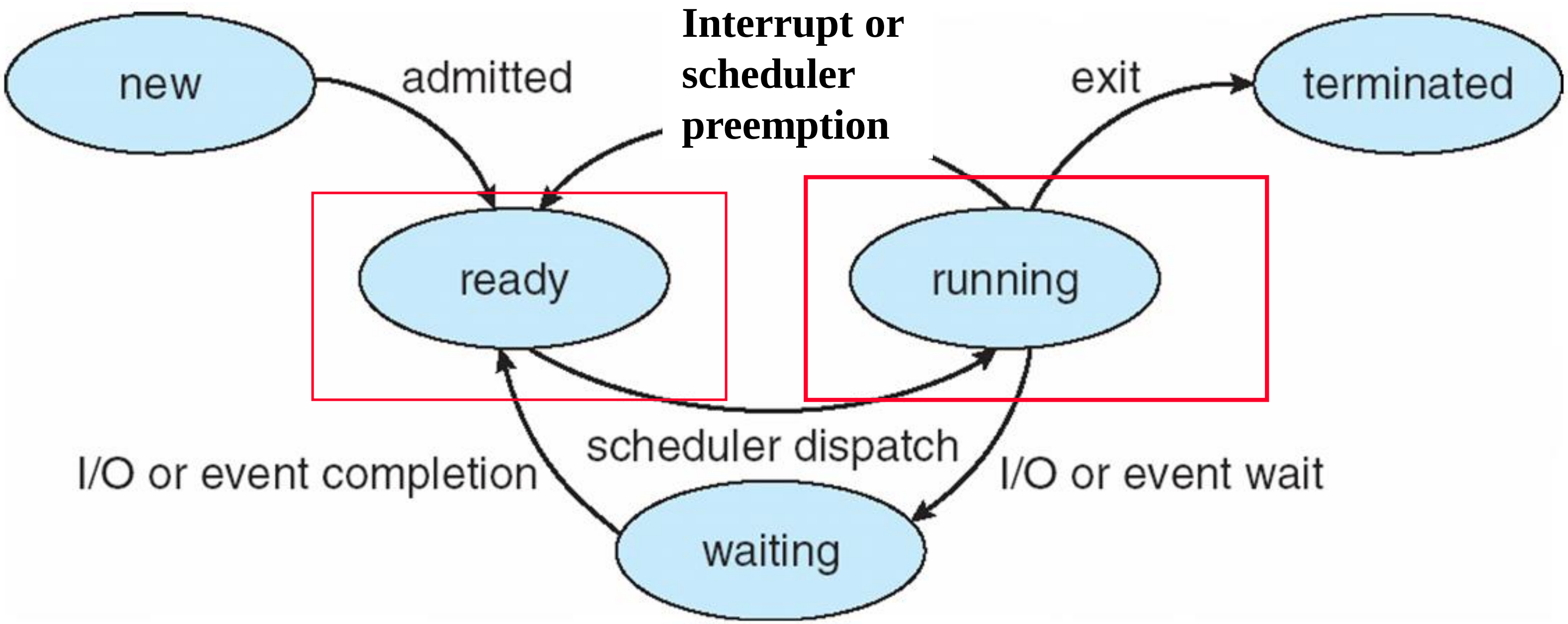
Notes

- In this discussion of CPU scheduling, we use the term "process" to represent a "traditional" process of only a single thread.

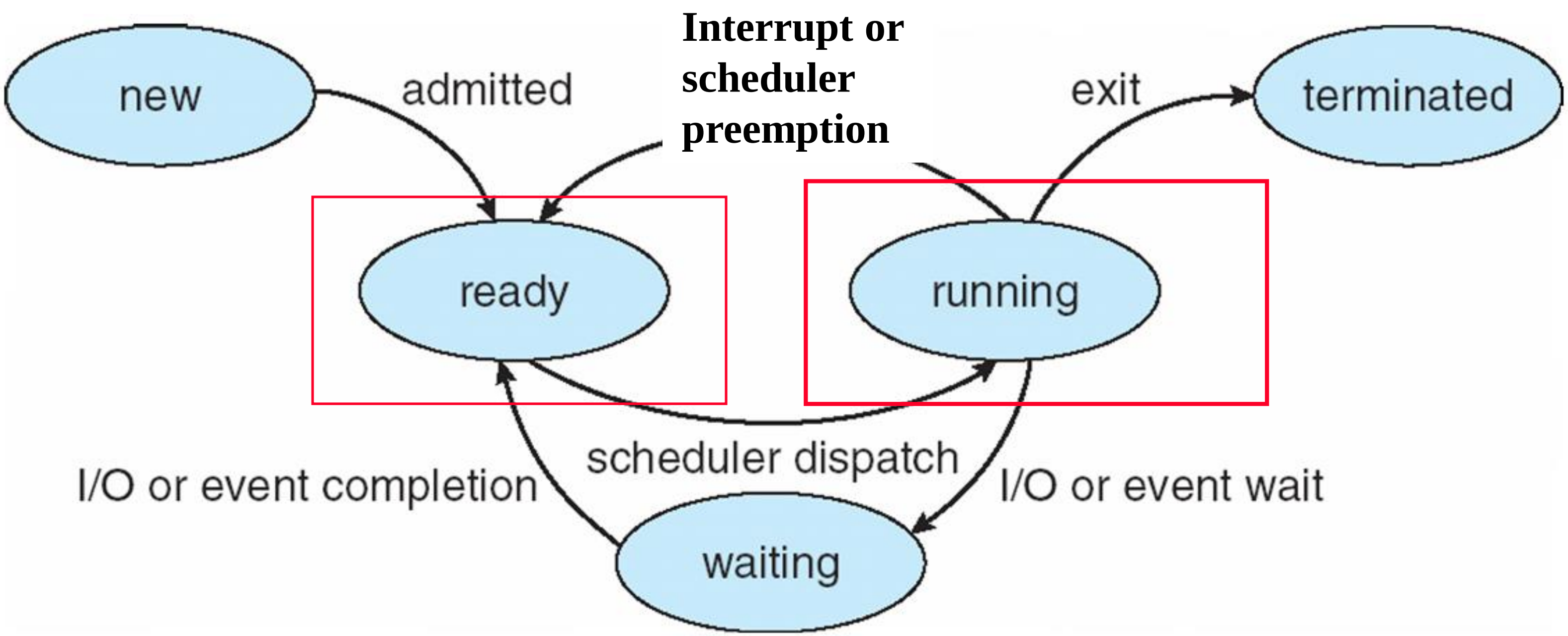
- The principles and algorithms also apply to kernel threads, if the OS supports kernel threads. Note: if a process is allowed to have multiple kernel threads, it is the kernel threads (not processes) that are scheduled by the OS.

Recall: Process State and Transitions

- **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



- 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/new to ready
  4. Terminates



CPU Scheduler

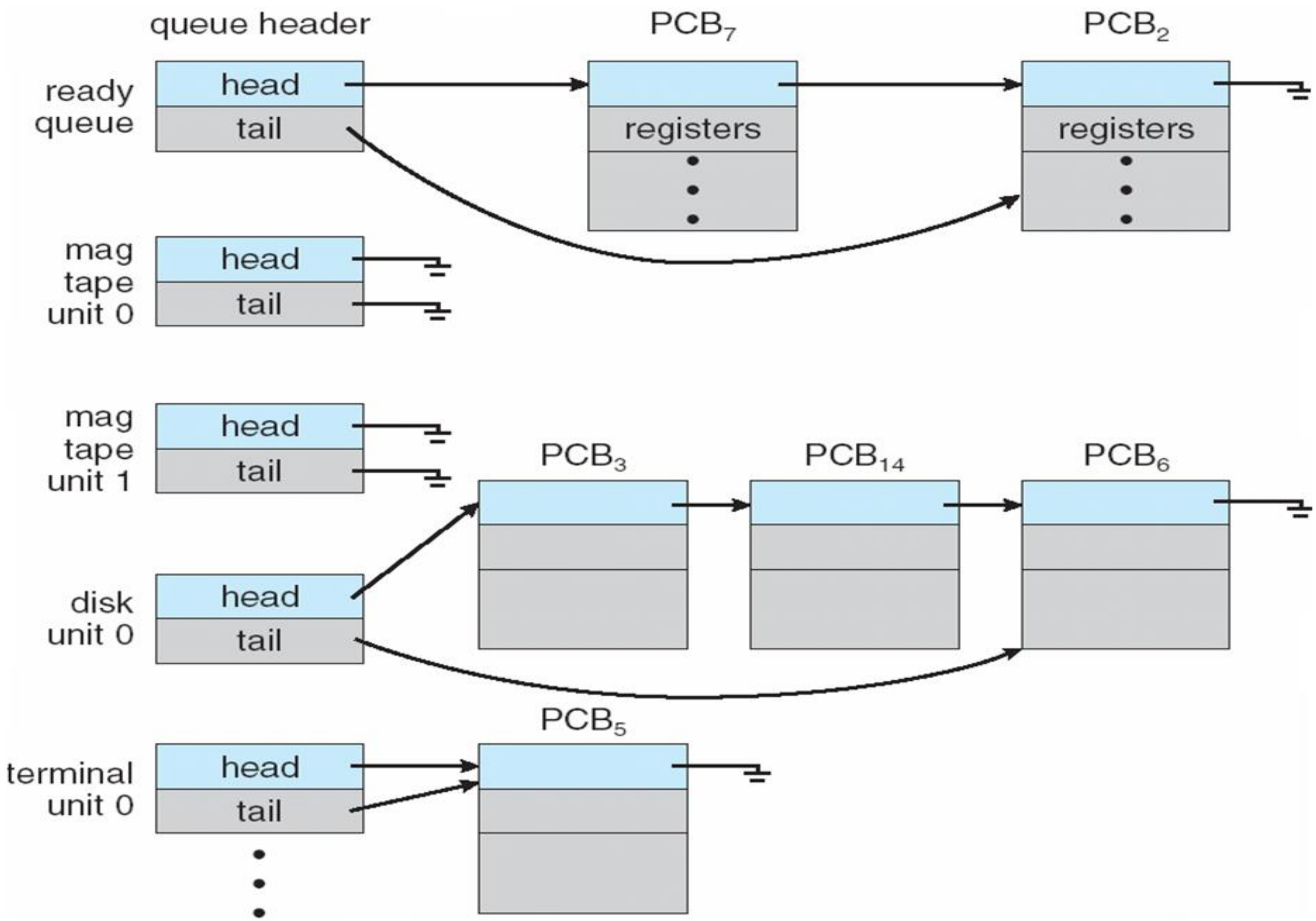
- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them

- 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/new to ready
  4. Terminates
- Scheduling under 1 and 4 is **nonpreemptive**
- All other scheduling is **preemptive**

Process Queues

- **Job queue** – the queue of the PCBs of all the processes in the system
- **Ready queue** – the queue of the PCBs of all processes residing in main memory, ready and waiting to execute
- **Device queues** – the queue of PCBs of processes waiting for an I/O device
- Processes (i.e., their PCBs) migrate among the various queues
- Note: When it is threads that are scheduled, the queues are composed of TCBs.

Ready Queue And Various I/O Device Queues



Context Switch

- When CPU switches from one process to another one, the system (hardware/OS) must save the state of the old process and load the saved state of the new one via a context switch

Context of a Running Process

- User memory space

- Value of “program counter register”
  - storing the address of the instruction to be executed next
- Value of “stack pointer register”
  - pointing to the top of current stack
- Values of other registers
  - storing intermediate results of ongoing computations

(When a process is running, the above information is stored on the registers.)

- Process state, memory info, I/O info, … : stored in PCB

Context of a Non-Running Process

- User memory space

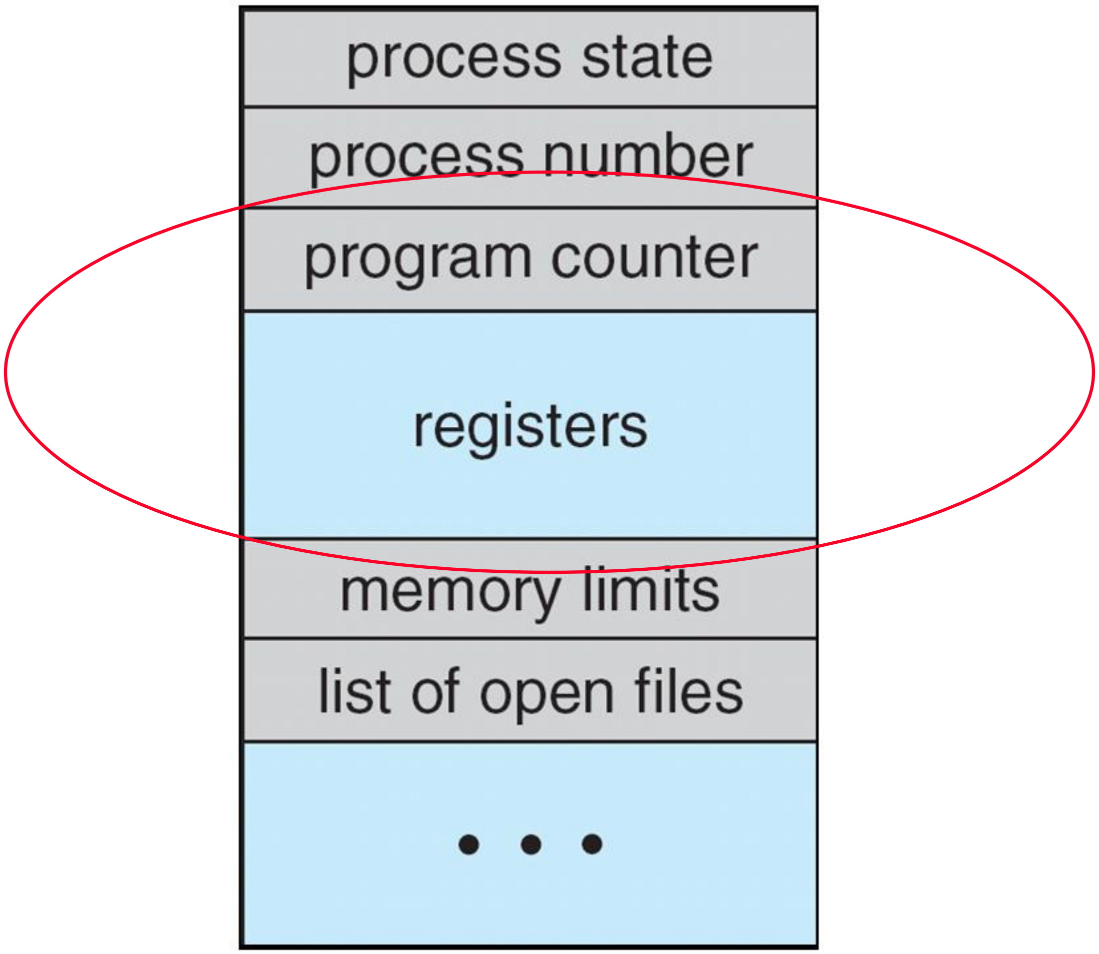
- Value of "program counter register"
  - storing the address of the instruction to be executed next
- Value of "stack pointer register"
  - pointing to the top of current stack
- Values of other registers
  - storing intermediate results of ongoing computations

(When a process is not running, the above information is stored in PCB.)

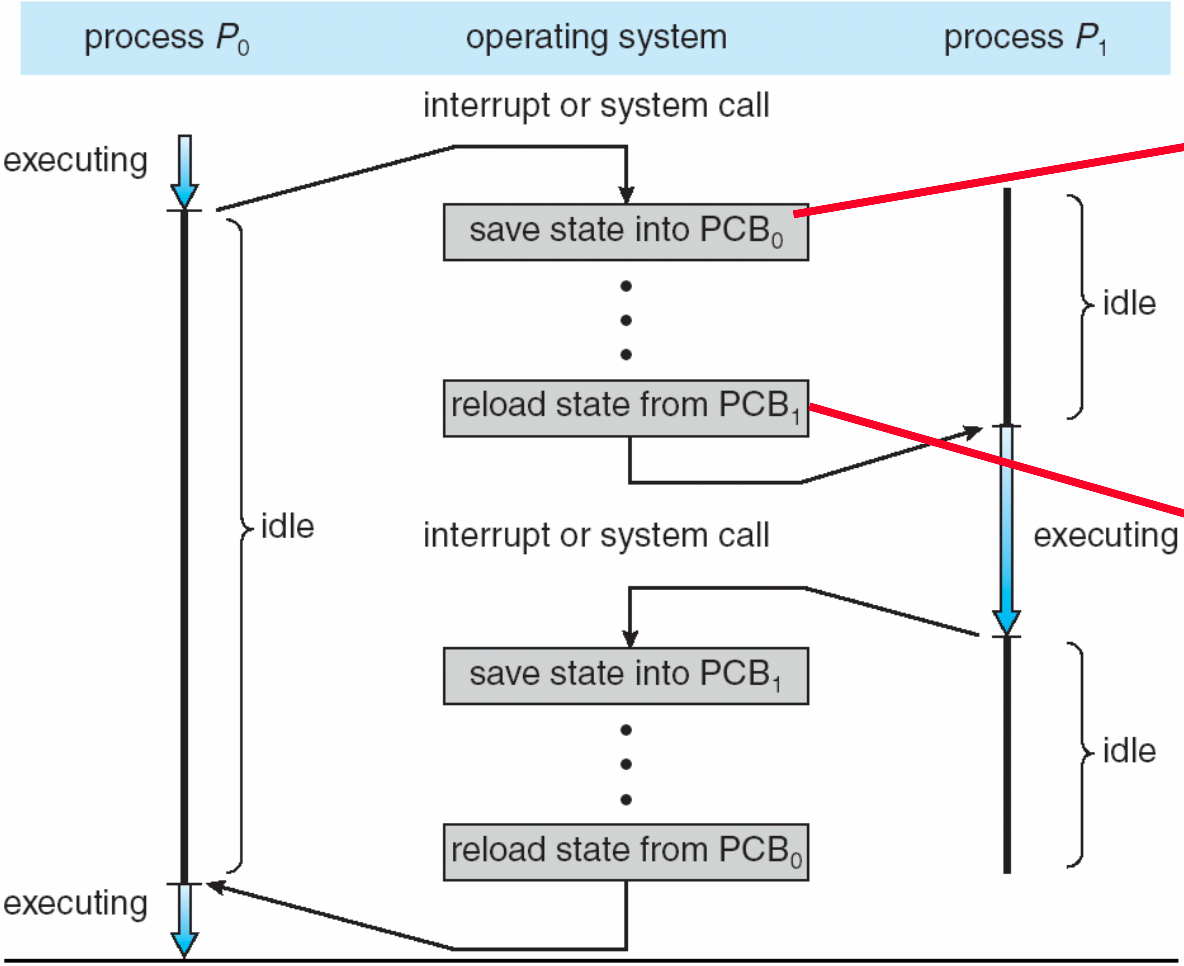
- Process state, memory info, I/O info, … : stored in PCB

Note: for a thread, it is TCB.

Process Control Block (PCB)



CPU Switch between Processes



Save current values of CPU registers into PCB_0

Load the values of CPU registers saved in PCB_1 to the registers

Context Switch

- Context-switch time is overhead
- Time depends on hardware support
  - memory speed;
  - number of registers whose values should be copied;
  - the existence of special instructions for storing/loading all registers
  - Typically, it takes a few microseconds

Evaluation Metrics

- **CPU utilization** – the fraction of valued CPU time over the whole time window
- **Throughput** – # of processes that complete their execution per time unit
- **Turnaround time** – amount of time to execute a particular process (from when a process is created/submitted to when it completes)
- **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 (for time-sharing environment)

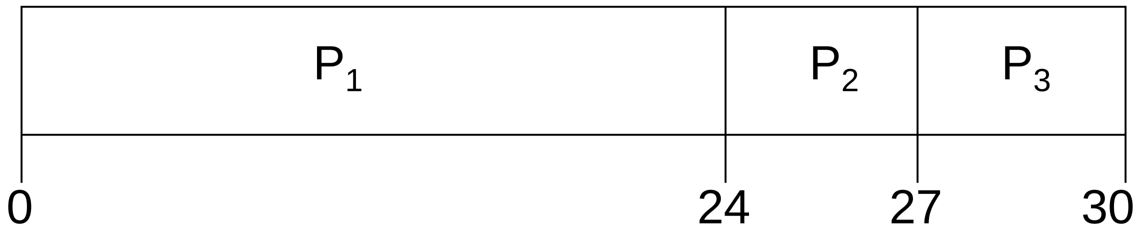
Scheduling Algorithm Optimization Criteria

- Maximize CPU utilization
- Maximize throughput
- Minimize turnaround time
- Minimize waiting time
- Minimize response time

First-Come, First-Served (FCFS) Scheduling

(Ready) Process	(Next) 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 time interval between arrivals is negligible*. The Gantt Chart for the schedule is:



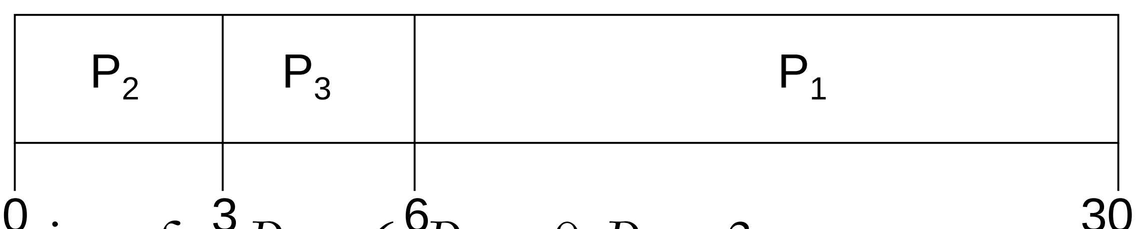
- Waiting time for $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- Average waiting time: $(0 + 24 + 27)/3 = 17$

FCFS Scheduling (Cont)

Suppose that the processes arrive in the order

$$P_2 , P_3 , P_1$$

- The Gantt chart for the schedule is:



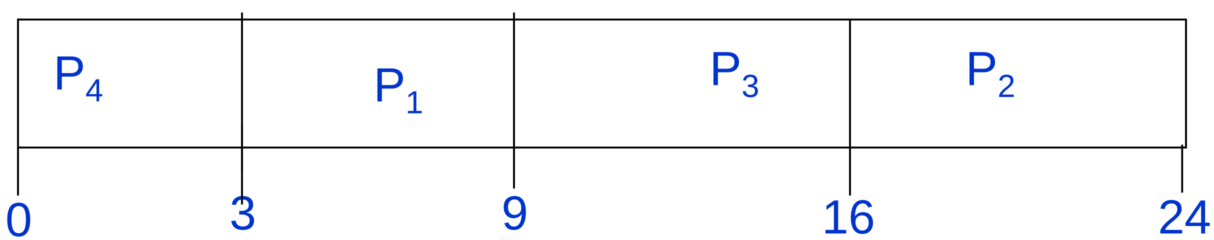
- Waiting time for $P_1 = 6; P_2 = 0; P_3 = 3$
- Average waiting time: $(6 + 0 + 3)/3 = 3$
- Much better than previous case
- *Convoy effect* when short processes behind a long process

Shortest-Job-First (SJF) Scheduling

- Schedule first the process with the shortest burst time

Process	(CPU) Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF scheduling chart



- Average waiting time = (3 + 16 + 9 + 0) / 4 = 7

SJF: Preemptive or Nonpreemptive

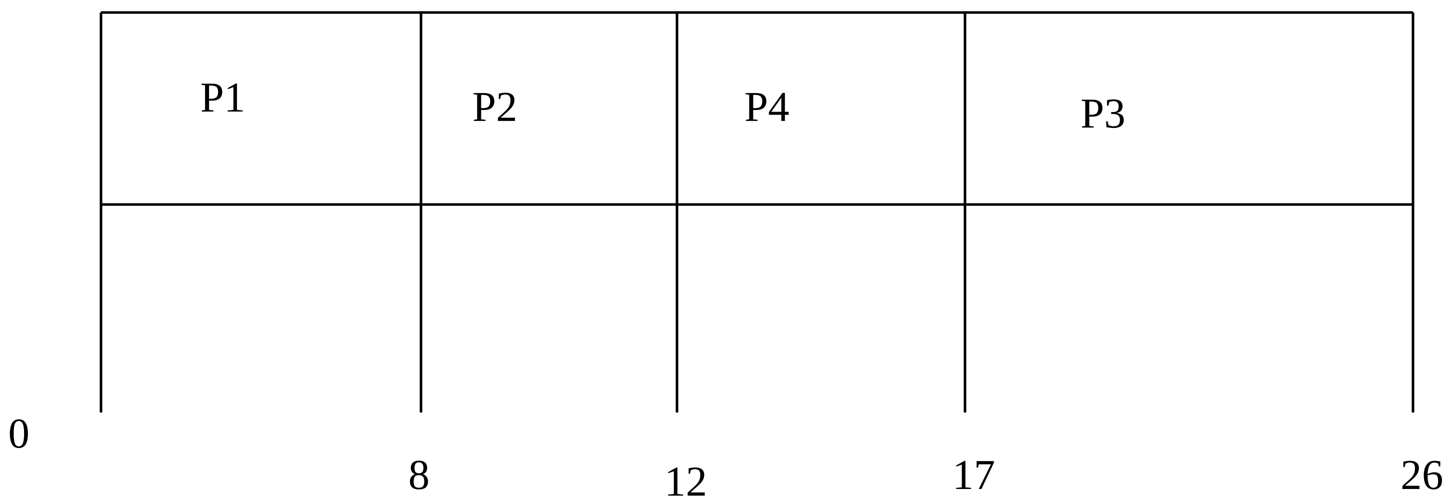
- SJF can be either preemptive or nonpreemptive
  - Preemptive (Shortest-remaining-time-first): the currently running process can be preempted by a newly arriving process that has shorter CPU burst
  - Non-preemptive: even a newly arriving process has shorter CPU burst than the currently running one, the running process is not preemptive.

Process	Arrival Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

What are the Gantt charts for the preemptive SJF and the nonpreemptive SJF?

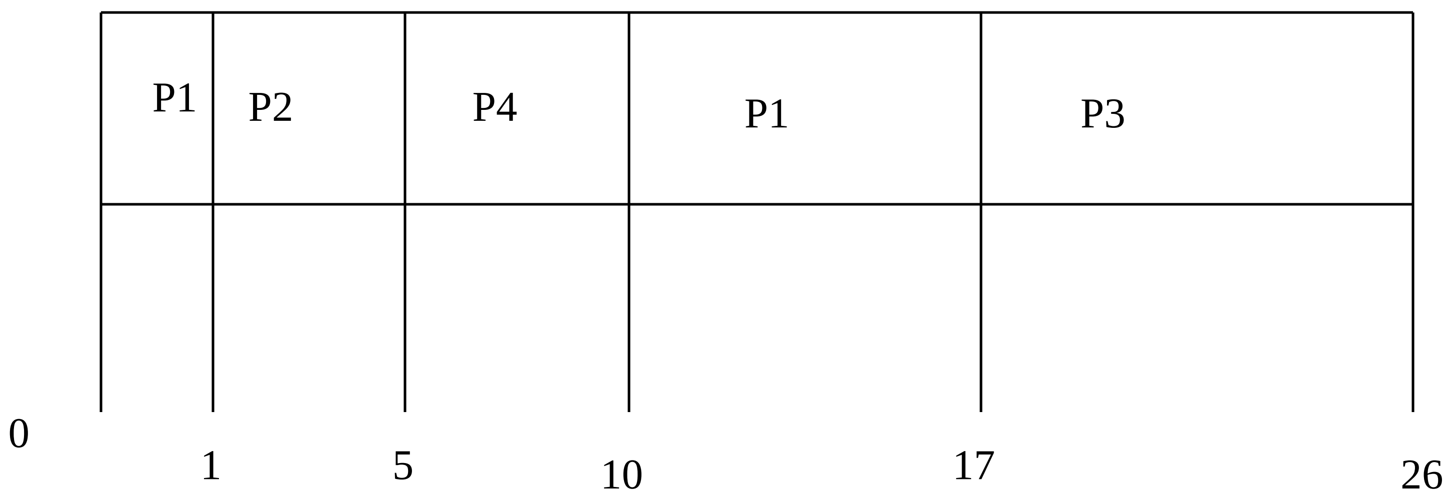
Non-preemptive SJF

Process	Arrival Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5



Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0	8
P_2	*1*	4
P_3	2	9
P_4	3	5



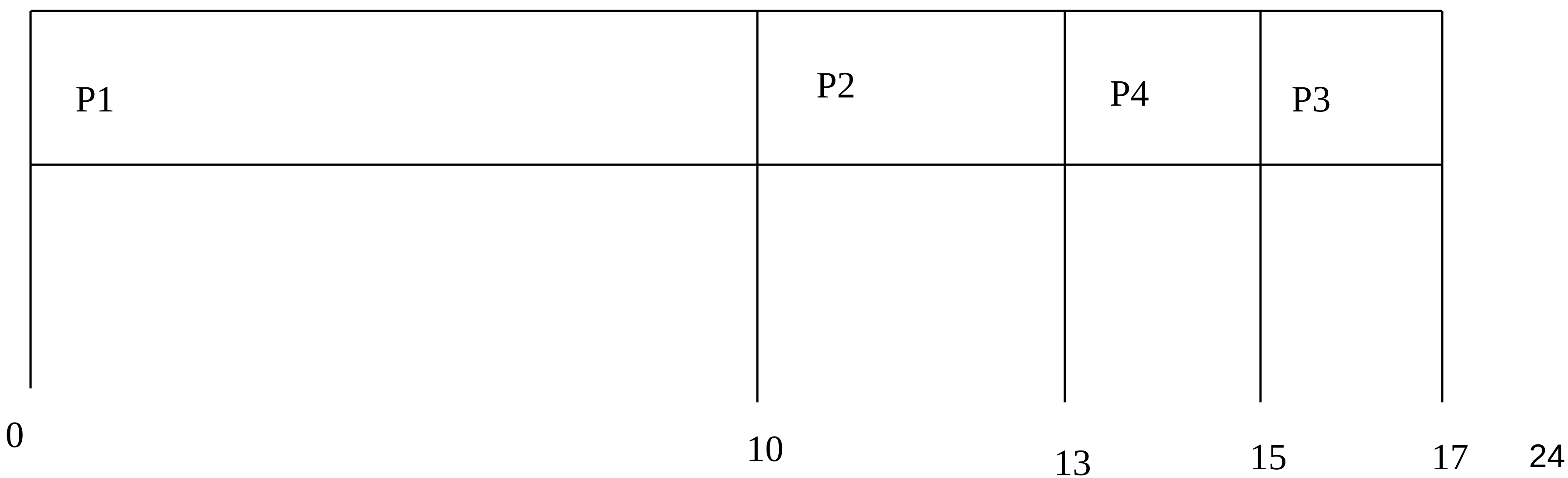
Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority)
  - Preemptive
  - Non-preemptive
- SJF is a priority scheduling where priority is the predicted next CPU burst time

Process	Arrival Time	Burst Time	Priority
P_1	0	10	3
P_2	*1*	3	1
P_3	2	2	4
P_4	3	2	2

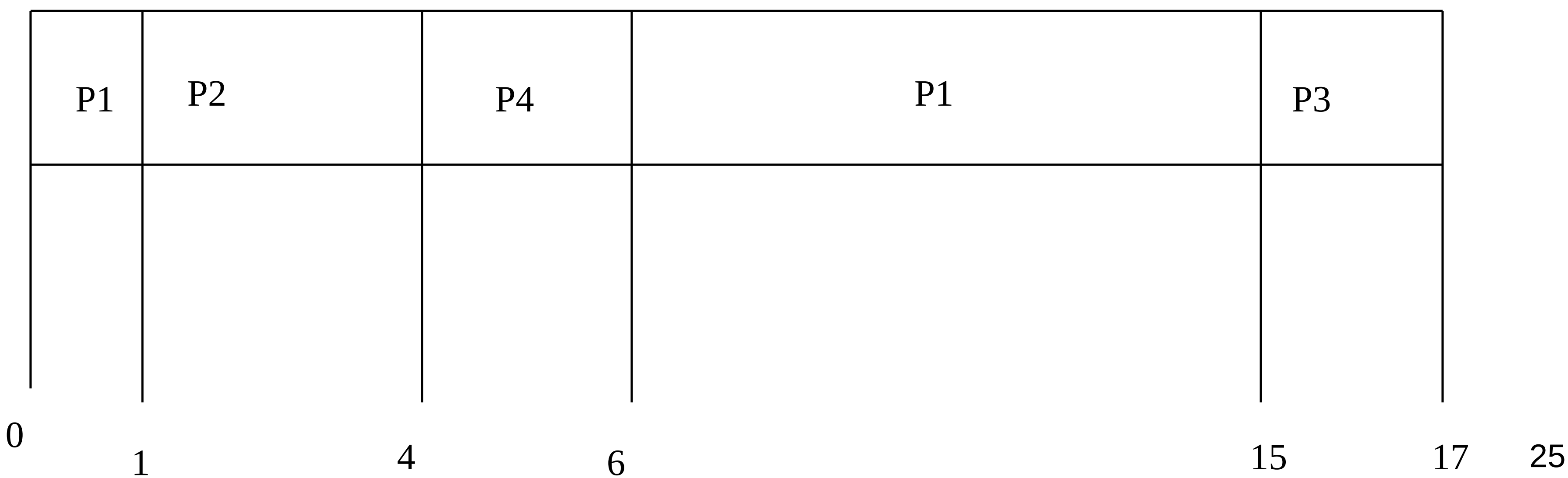
Priority Scheduling: Non-preemptive

Process	Arrival Time	Burst Time	Priority
P_1	0	10	3
P_2	1	3	1
P_3	2	2	4
P_4	3	2	2



Priority Scheduling: Preemptive

Process	Arrival Time	Burst Time	Priority
P_1	0	10	3
P_2	*1*	3	1
P_3	2	2	4
P_4	3	2	2



Priority Scheduling

- Problem: **Starvation** – low priority processes may never execute
- Solution: **Aging** – as time progresses increase the priority of the waiting process

Process	Arrival Time	Burst Time	Priority
P_1	0	10	1
P_2	*1*	5	10
P_3	2	2	2
P_4	3	2	3
P_5	4	2	3
P_6	5	2	3
P_7	6	1	3
P_8	7	1	2
…	…	…	…