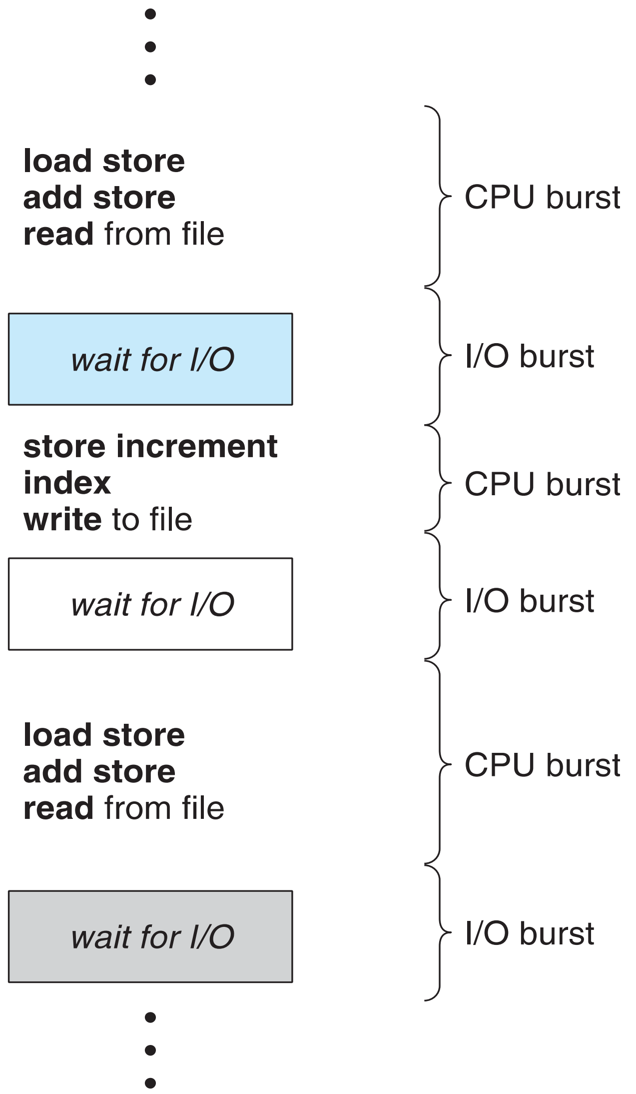
Amirkabir University of Technology
(Tehran Polytechnic)
Department of Computer Engineering

CPU Scheduling
زمان‌بندی CPU

Hamid R. Zarandi
h_zarandi@aut.ac.ir

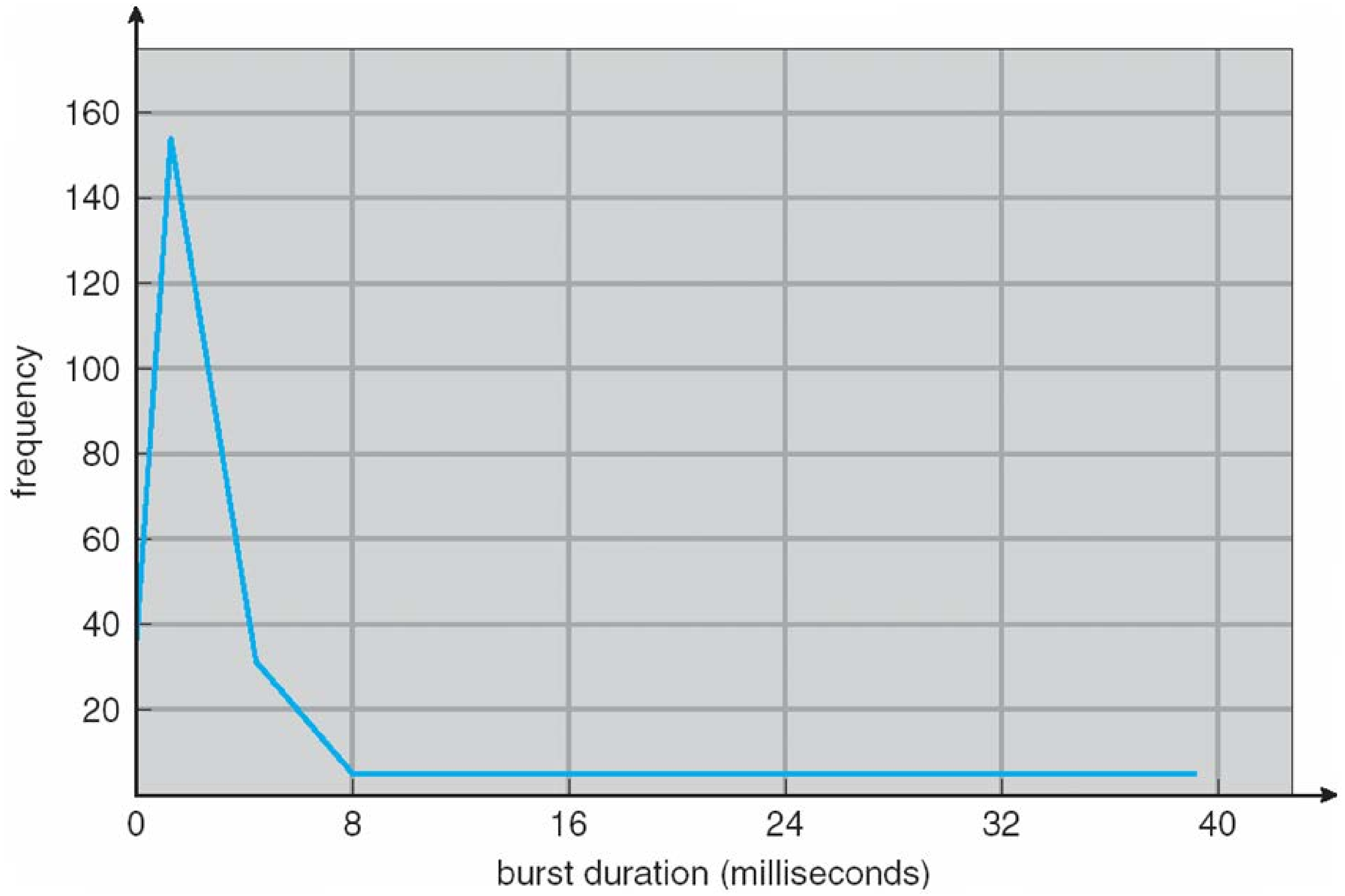
Motivation

- To make computer more productive
  - Maximum CPU utilization obtained with multiprogramming
- Process scheduling or Thread scheduling
- Having different sequence of IO or CPU
  - CPU–I/O Burst Cycle – Process execution consists of a cycle of CPU execution and I/O wait
  - CPU burst followed by I/O burst
  - CPU burst distribution is of main concern



CPU burst curve

- **Exponential or hyperexponential**

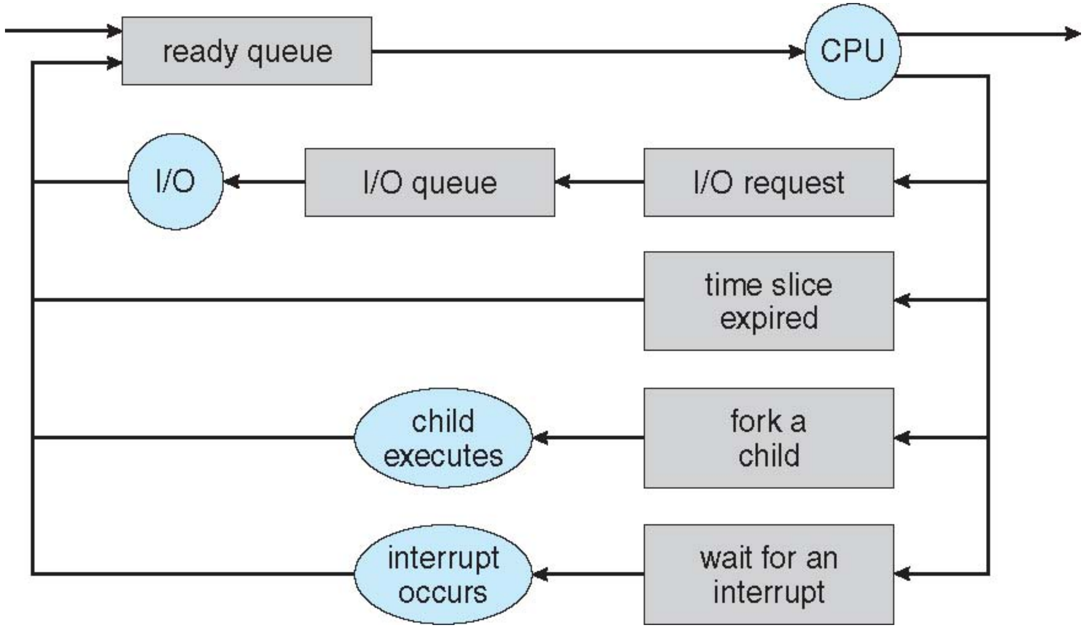


CPU scheduler

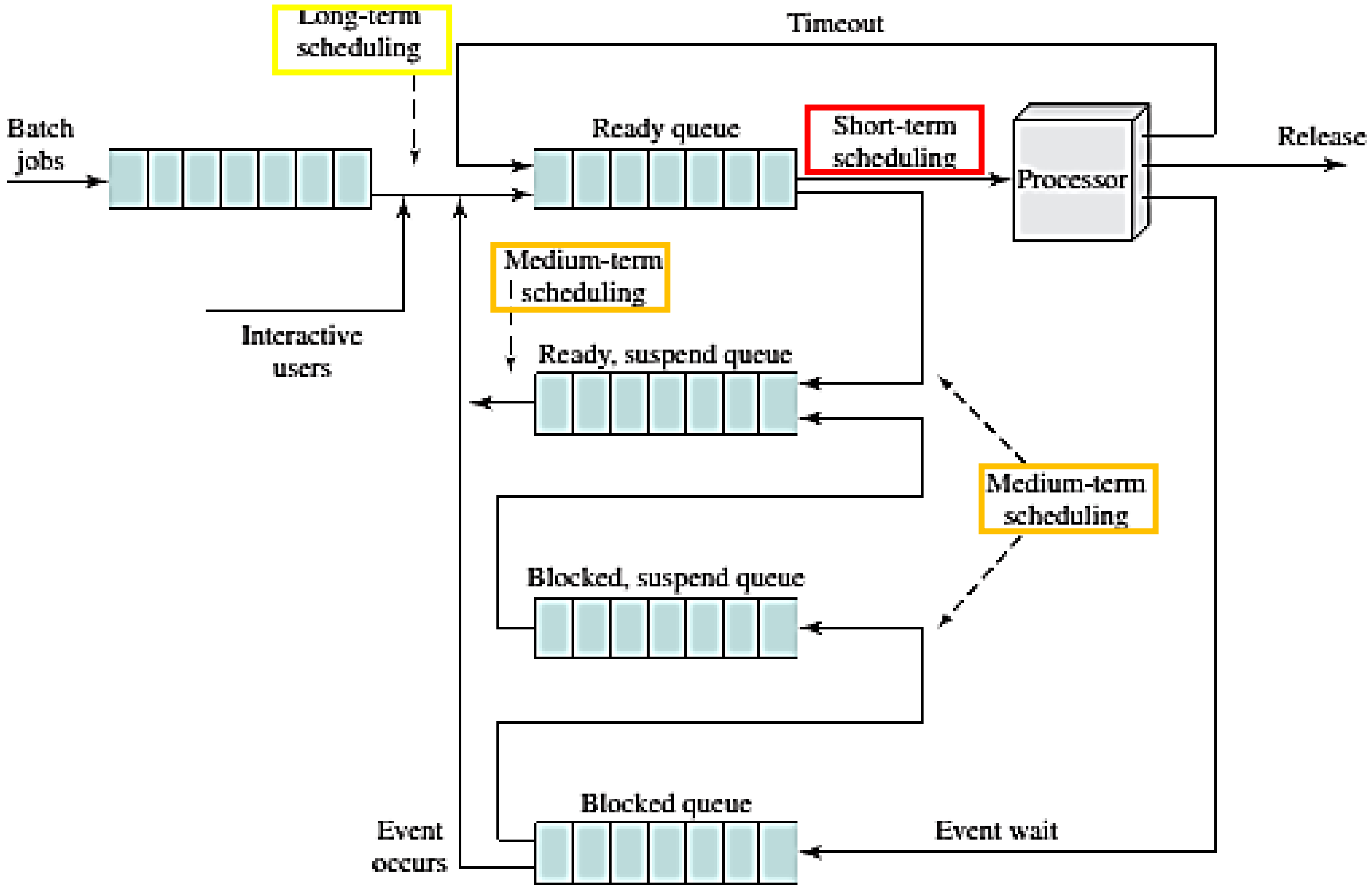
- Whenever CPU is idle, it must select another process from ready queue (short-term scheduler).

- Short-term scheduler selects from among the processes in ready queue, and allocates the CPU to one of them

- Ready queue: FIFO, priority queue, tree, unordered linked list!
  - Consisted of PCBs of processes

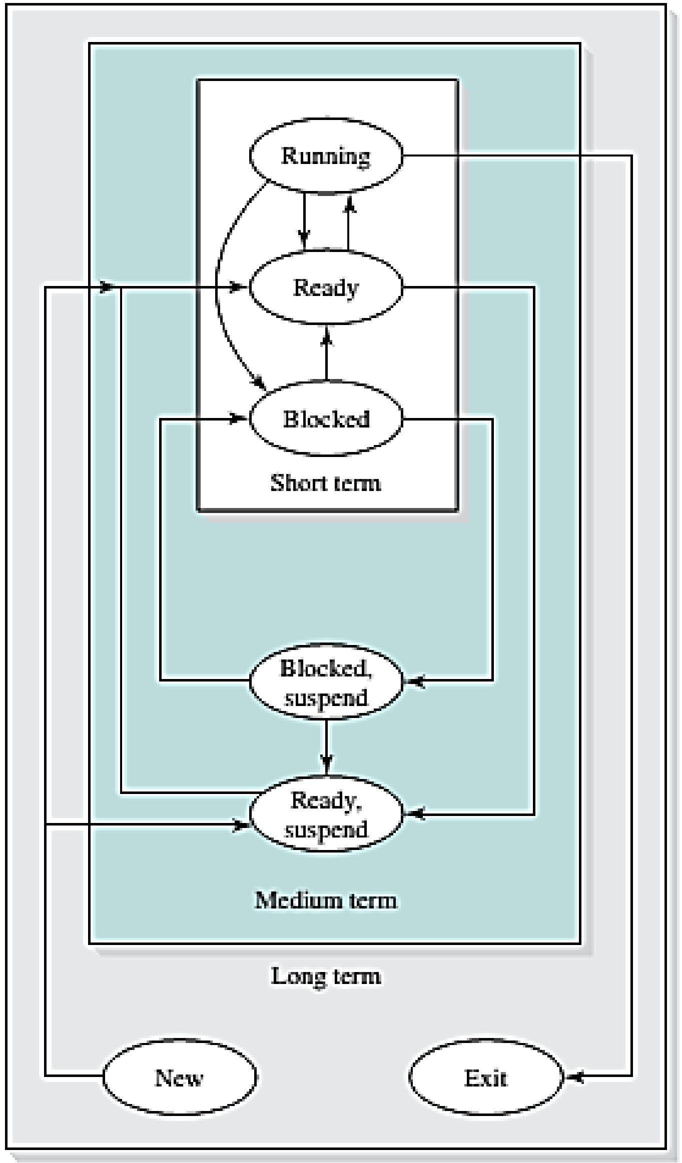


Schedulers

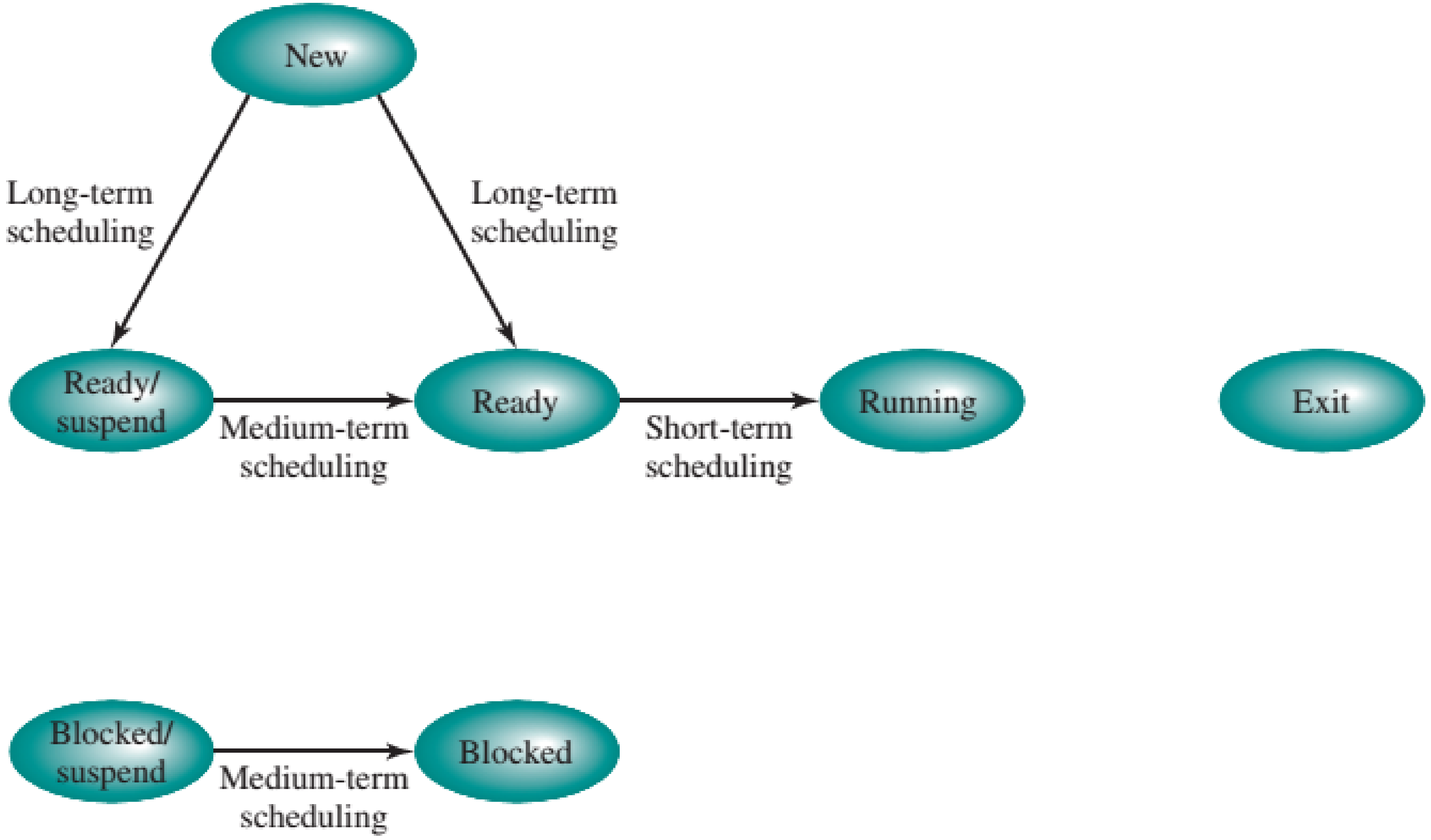


Long-term scheduling
Timeout
Batch jobs
Ready queue
Short-term scheduling
Processor
Release
Medium-term scheduling
Interactive users
Ready, suspend queue
Medium-term scheduling
Blocked, suspend queue
Blocked queue
Event occurs
Event wait

Level of scheduling



Scheduling and process state transition



Preemption and preemptive scheduling

- **Preemption**
  - The act of temporarily interrupting a task being carried out by a computer system, without requiring its cooperation, and with the intention of resuming the task at a later time [wiki]
- **Scheduler**
  - Preemptive vs. Nonpreemptive (cooperative)
  - When CPU can switch?
    1. A process switches from running → waiting state (IO request, wait())
    2. A process switches from running → ready state (interrupt occurs)
    3. A process switches from waiting → ready state (completion of IO)
    4. A process terminates!
- Scheduling under 1 and 4 is nonpreemptive
- All other scheduling is preemptive

Which one is good? preemptive or nonpreemptive

Nonpremtive scheduler

- Windows 3.1
- No need of any special hardware mechanisms (timer, etc.)

Preemptive scheduler

- Windows 95, 98, ME, XP, 7, 8, 10
- Mac OS X
- Can result Race Condition! (why?)

Dispatcher

- An OS module gives control of CPU to the process selected by short-term scheduler
  - Switching context
  - Switching to user mode
  - Jumping to proper location in the user program to resume it

- Should be fast.

- Dispatch latency
  - The time to stop one process and start another running

Which scheduler is the best?

- **Criteria**
  - CPU utilization (بهره‌وری)
    - As busy as possible
    - A value from 0 to 100 (real system 40 to 90)
  - Throughput (گذردهی، برون‌دهی)
    - Number of processes that are completed.
  - Turnaround time
    - Time from submission of a process to time of completion
    - Sum of periods spent waiting {to get memory, IO, CPU}, running in CPU, doing IO
  - Waiting time
    - Sum of periods spent waiting in the ready queue
  - Response time
    - Time from submission of a request until first response is produced.
    - Time it takes to start responding
- **Which one is better?**

Best scheduler?

- For interactive systems (desktop systems)
  - Minimizing variance in response time

- The main question:
  - Which one of processes in Ready Queue is to be allocated to CPU?

Scheduling Algorithms

1) First-Come, First Served scheduling

- Simplest
- Average waiting time is much!

Process	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 Gantt Chart for the schedule is:

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

1) FCFS Scheduling (Cont.)

- Suppose that the processes arrive in the order:

$$P_2\,,\ P_3\,,\ P_1$$

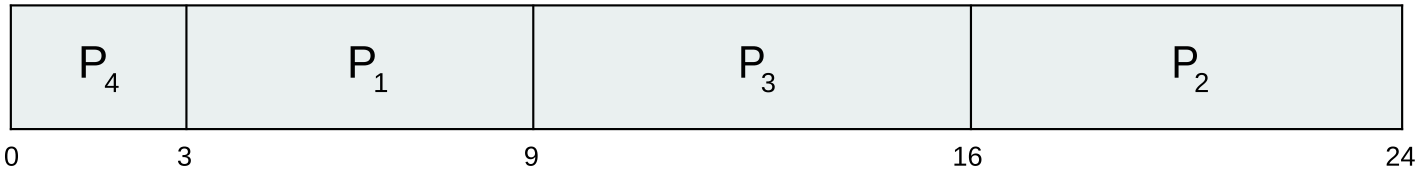
- Waiting time for $P_1 = 6$; $P_2 = 0$; $P_3 = 3$
- Average waiting time: $(6 + 0 + 3)/3 = 3$
- Much better than previous case (why?)
- Convoy effect - short process behind long process
  - Consider one CPU-bound and many I/O-bound processes

2) Shortest-Job-First scheduling

- Shortest-next-CPU-burst
- Decides based on the length of process's next CPU burst
- Is optimal; has min average waiting time!
- It cannot be implemented in short-term scheduler (why?)

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

SJF scheduling chart



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

Determining length of next CPU burst

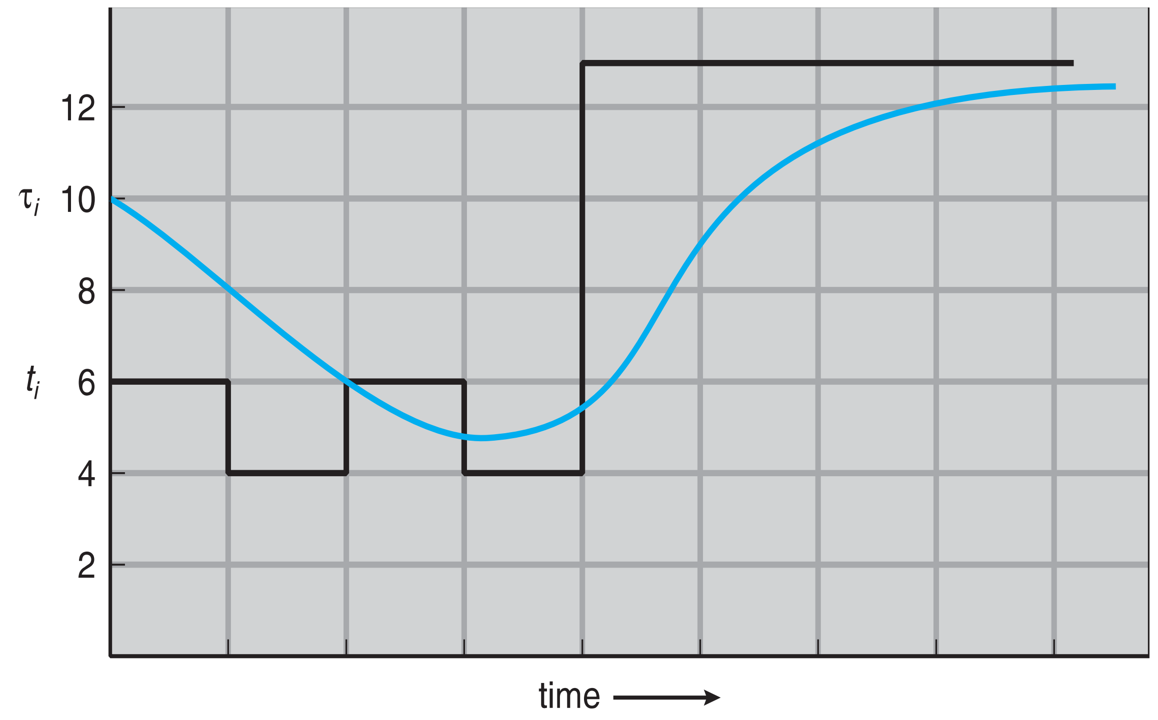
- **Prediction as exponential average**

1. $t_n =$ actual length of n^{th} CPU burst
2. $\tau_{n+1} =$ predicted value for the next CPU burst
3. $\alpha, 0 \le \alpha \le 1$
4. Define : $\tau_{n+1} = \alpha\, t_n + (1-\alpha)\tau_n$

- **Commonly, α set to ½**

- **Two implementations: Preemptive, Nonpreemptive**
  - **Preemptive SJF: shortest-remaining-time-first**

Example



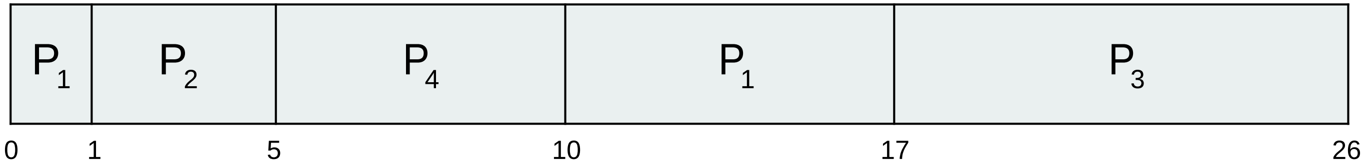
CPU burst (t_i)		6	4	6	4	13	13	13	…
"guess" (τ_i)	10	8	6	6	5	9	11	12	…

Preemptive SJF

Now we add the concepts of varying arrival times and preemption to the analysis

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

Preemptive SJF Gantt Chart



Average waiting time = [(10-1)+(1-1)+(17-2)+5-3)]/4 = 26/4 = 6.5 msec

3) Priority scheduling

- General case of SJF (how?)

- A priority (number) is associated with each process
  - Internal: time limits, memory requirements, number of open files, ratio of IO burst to average CPU burst
  - External: outside of OS (importance of process, type of funds being paid, etc)

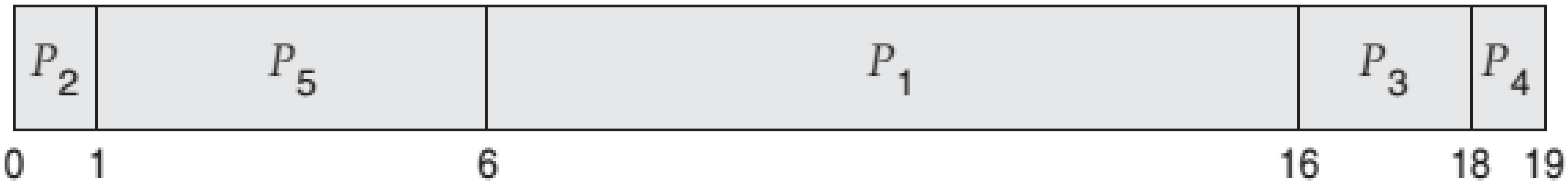
- Can be:
  - preemptive
  - nonpreemptive

3) Priority scheduling (cont'd)

- Main problem? Indefinite blocking or starvation
- Solution? To include aging

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

Priority scheduling Gantt Chart



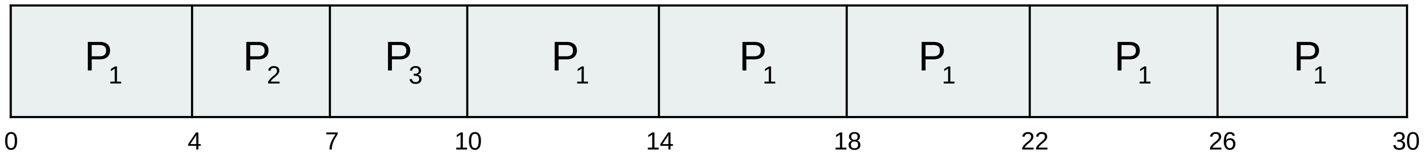
Average waiting time = 8.2 msec

4) Round-Robin scheduler

- **Time quantum = q (time slice)**
  - A small unit of time (usually 10-100 ms)
  - After this time has elapsed, the process is preempted and added to the end of the ready queue

Process	Burst Time
P_1	24
P_2	3
P_3	3

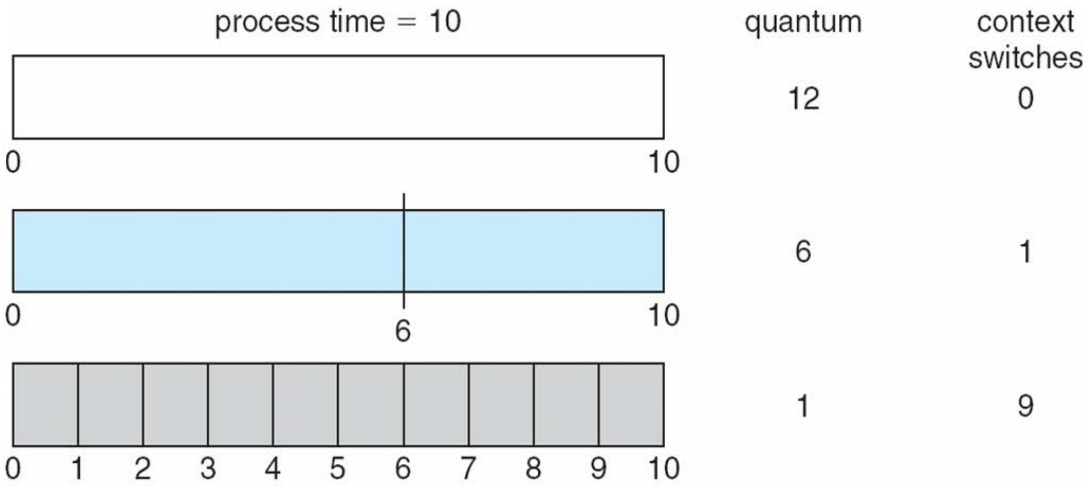
The Gantt chart is:

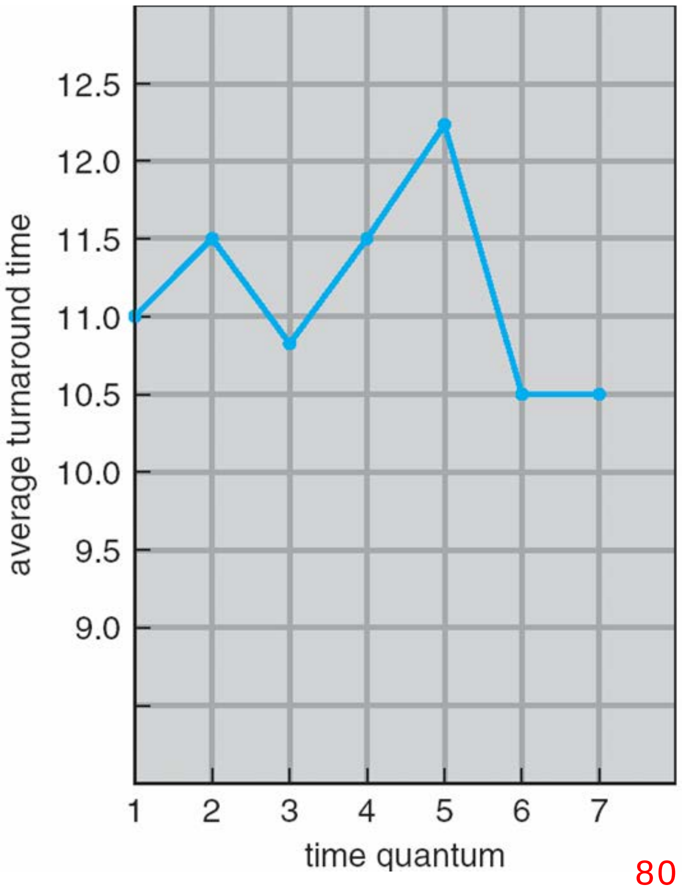


Typically, higher average turnaround than SJF, but better ***response***
q should be large compared to context switch time
q usually 10ms to 100ms, context switch < 10 usec

- **Small time slice is better or large?**
  - q large $\Rightarrow$ FIFO
  - q small $\Rightarrow$ q must be large with respect to context switch, otherwise overhead is too high

Time quantum & context switch time





process	time
P_1	6
P_2	3
P_3	1
P_4	7

80% of CPU bursts should be shorter than q

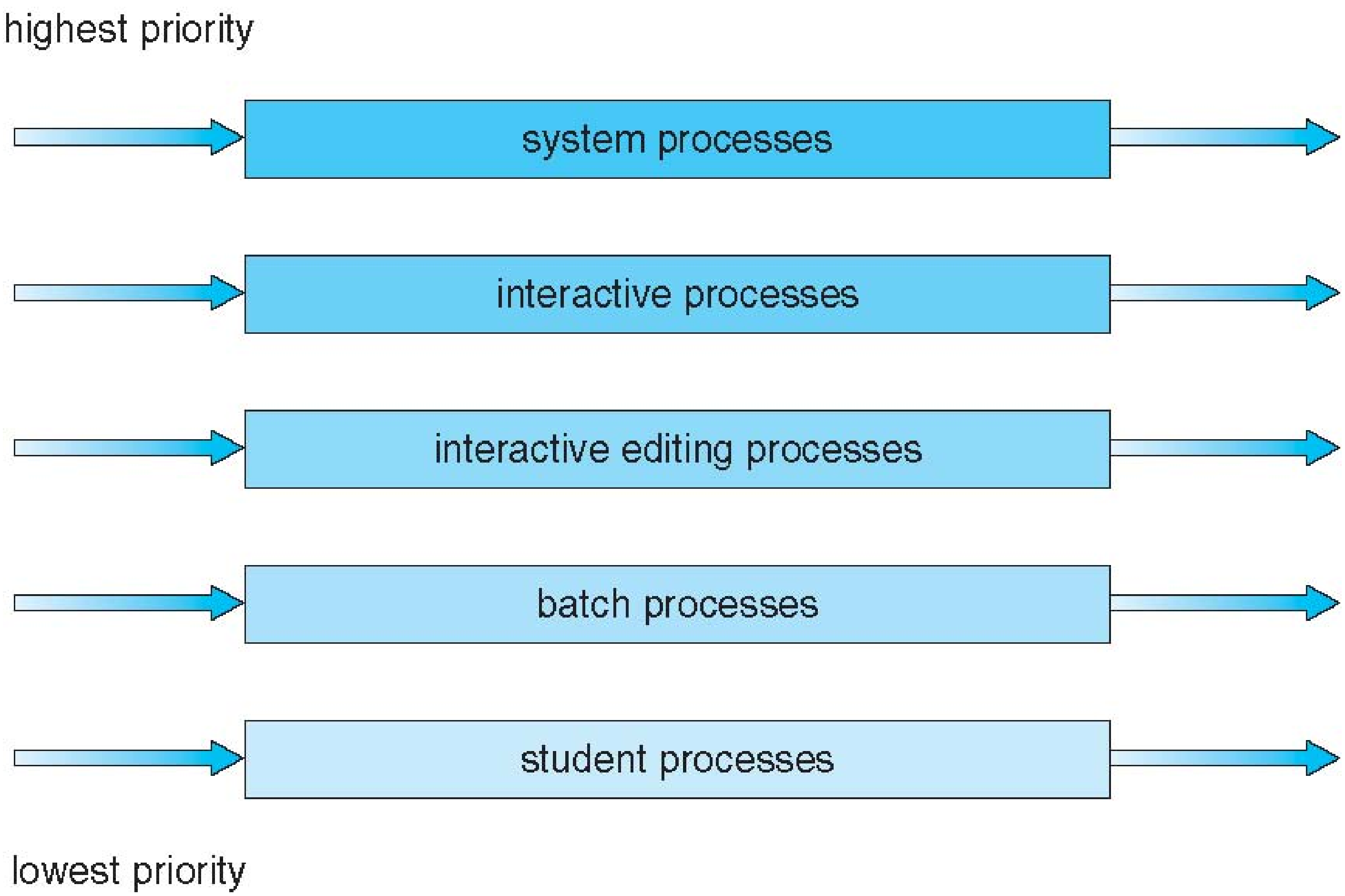
5) Multilevel Queue scheduler

- Ready queue is partitioned into separate queues, eg:
  - foreground (interactive)
  - background (batch)

- Each queue has its own scheduling algorithm:
  - foreground – RR
  - background – FCFS

- For example, it has 5 queues:
  - System processes
  - Interactive processes
  - Interactive editing processes
  - Batch processes
  - Student processes

Example of multilevel queue scheduler



6) Multilevel Feedback Queue scheduler

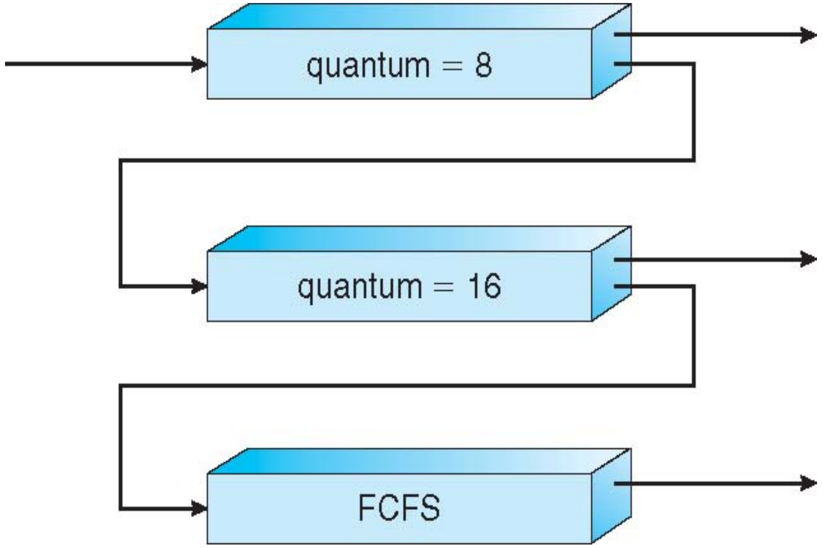
- A process can move between the various queues; aging can be implemented this way

- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue
  - method used to determine when to upgrade a process
  - method used to determine when to demote a process
  - method used to determine which queue a process will enter when that process needs service

Example of multilevel feedback queue

Three queues:

- Q_0 – RR with time quantum 8 milliseconds
- Q_1 – RR time quantum 16 milliseconds
- Q_2 – FCFS



Scheduling

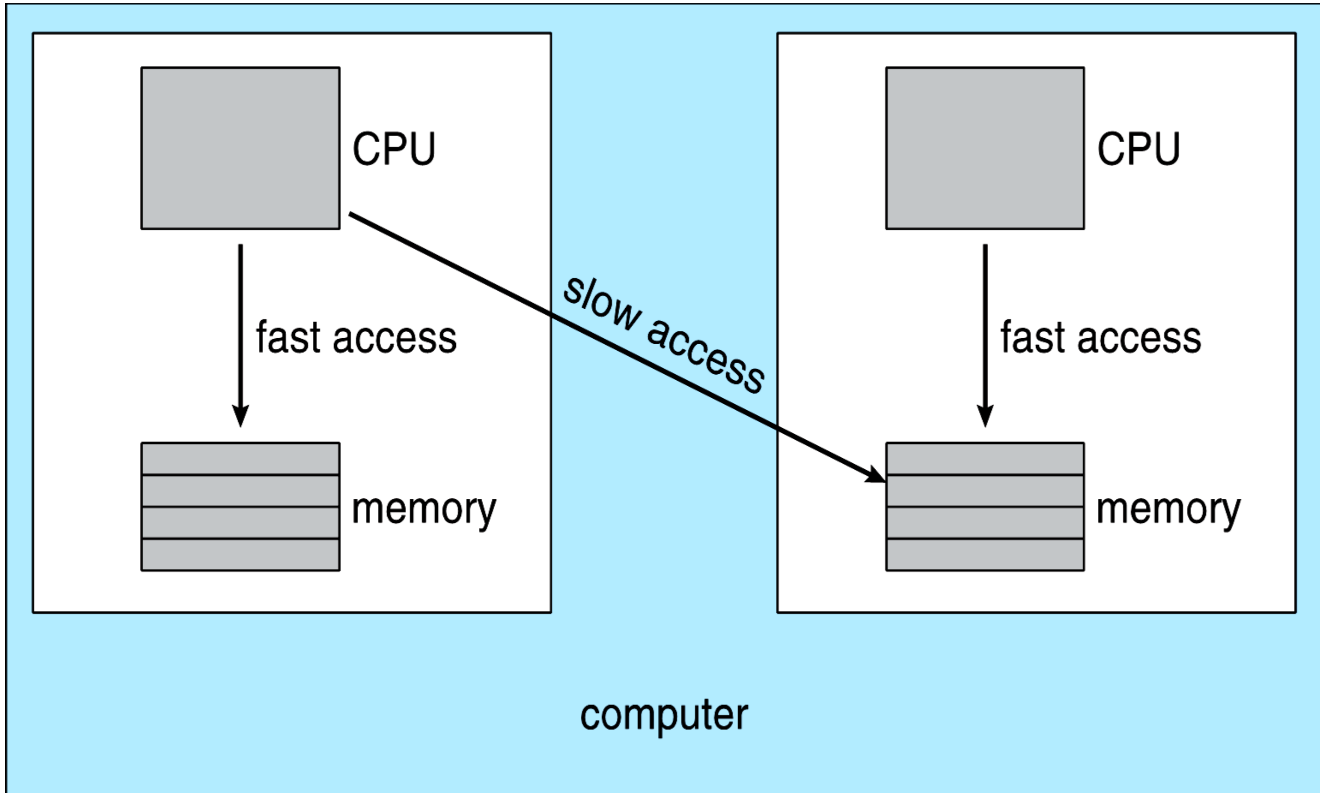
- A new job enters queue Q_0 which is served FCFS
  - When it gains CPU, job receives 8 milliseconds
  - If it does not finish in 8 milliseconds, job is moved to queue Q_1
- At Q_1 job is again served FCFS and receives 16 additional milliseconds
  - If it still does not complete, it is preempted and moved to queue Q_2

Multiple Processor Scheduling

Multiple-processor scheduling

- **Multiple processors**
  - Load sharing
- **Multiple-processor scheduling**
  - AMP: only one processor accesses the system data structures, alleviating the need for data sharing
    - Master server (master processor)
  - SMP
    - Common ready queue
    - Private ready queue
      - ✓ Processor affinity: a process has an affinity for the processor on which it is currently running
        - Soft affinity
        - Hard affinity
    - Load balancing
      - ✓ To get best CPU utilization

NUMA and CPU scheduling



Note that memory-placement algorithms can also consider affinity

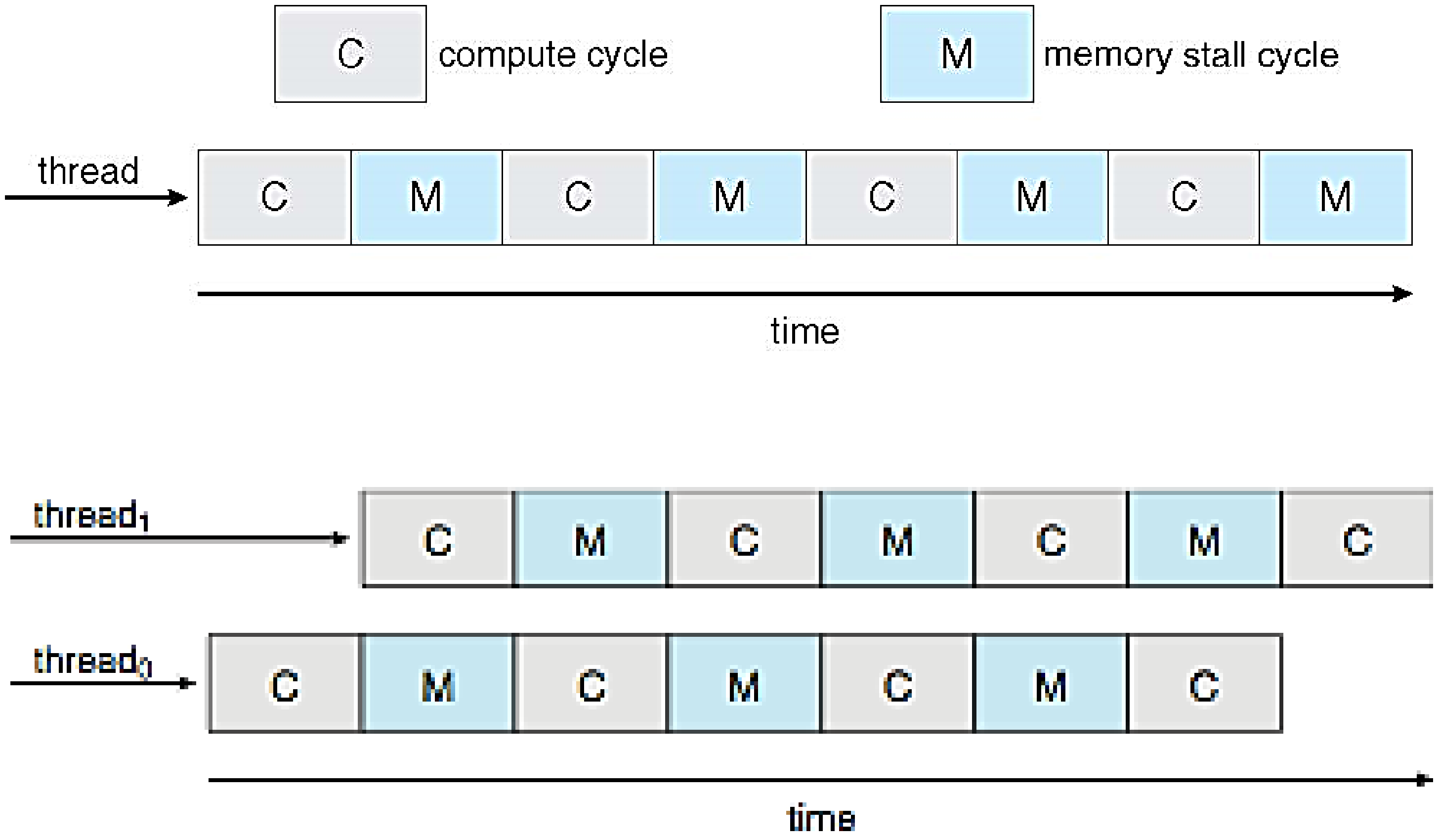
Load balancing in SMP

- If SMP, need to keep all CPUs loaded for efficiency
- Load balancing attempts to keep workload evenly distributed
  - Push migration
    - Task periodically check the loads on each processor
  - Pull migration
    - Idle processor pulls a waiting task
- Problem to affinity
  - Using threshold for imbalancing

Multicore processor

- Faster, less power consumption (why?)
- Memory stall is costly
  - Hardware threads for each core
    - UlteraSPARC T3 CPU (16*8)
    - Intel Itanium (dual core)
  - Coarse-grained vs. fine-grained scheduling

Multithreaded multicore system



Real-Time Scheduling

Real-time CPU scheduling

- **Events**
  - SW: timer
  - HW: external interrupts
- **Soft real-time systems**
  - No guarantee as to when critical real-time process will be scheduled
  - Guarantee only critical processes have preference over noncritical ones.
- **Hard real-time systems**
  - Task must be serviced by its deadline
  - Service after deadline is the same as no service at all.

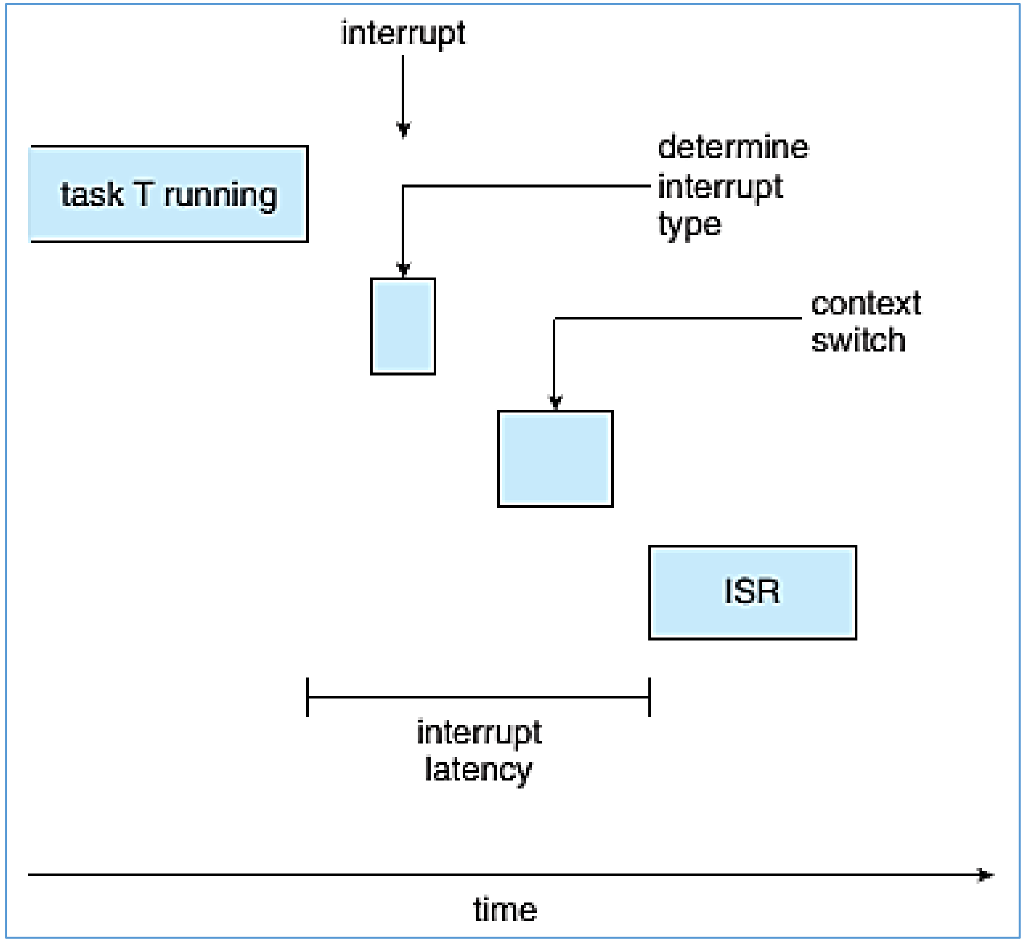
Event latency

➢Event latency

- Time from when an event occurs to when it is serviced
- For ABS: 3-5 ms

- Interrupt latency
  - time from arrival of interrupt to start of routine that services interrupt

- Dispatch latency
  - time for schedule to take current process off CPU and switch to another

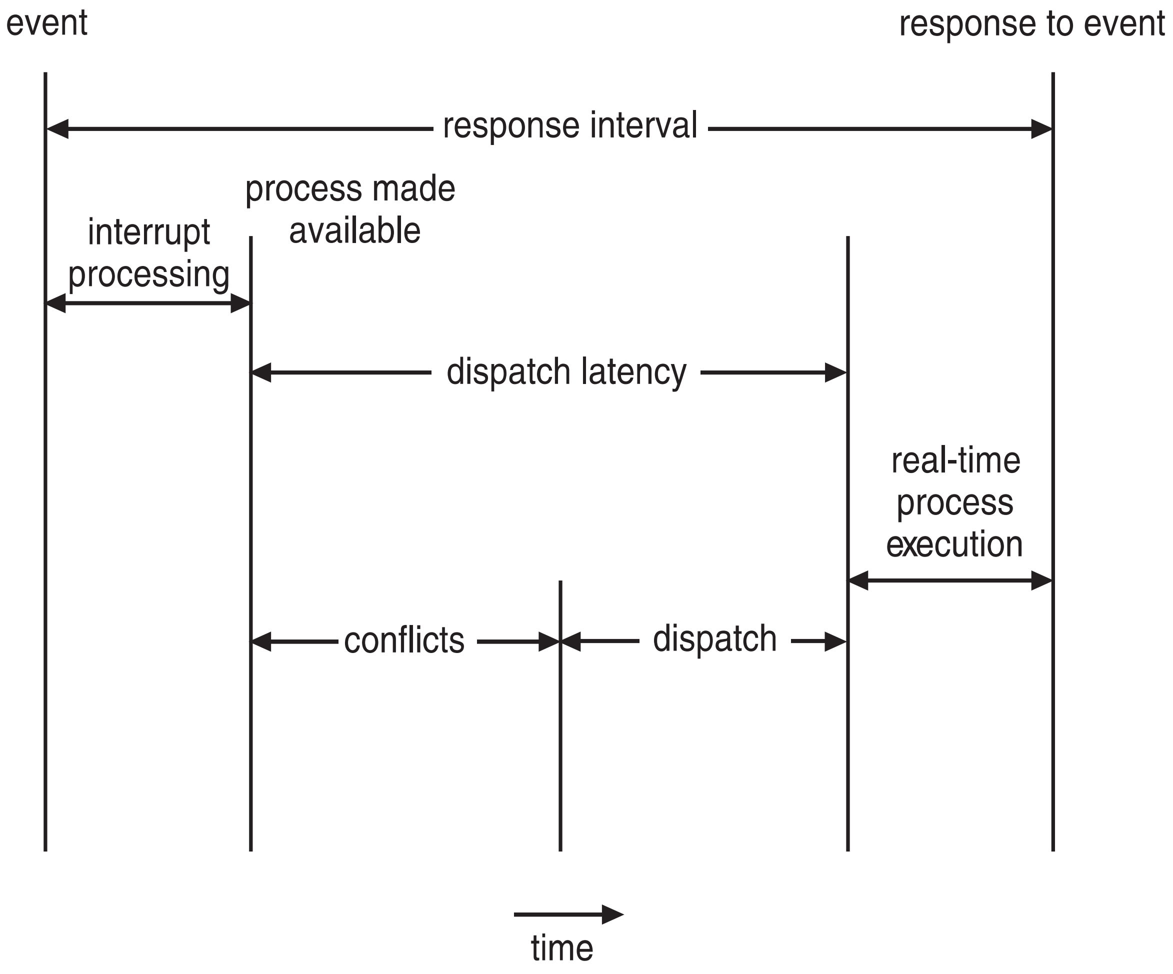


Dispatch latency

- **Conflict phase of dispatch latency:**

  1. Preemption of any process running in kernel mode

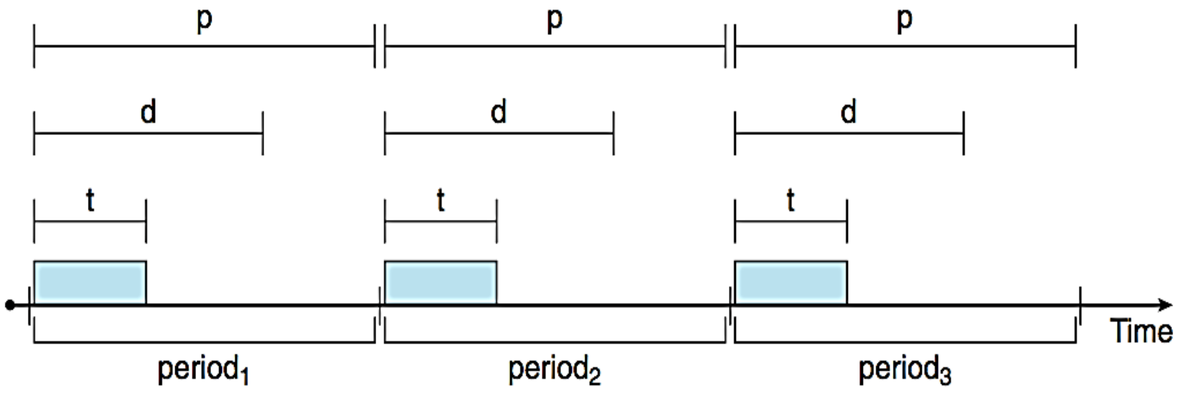
  2. Release by low-priority process of resources needed by high-priority processes



Priority-based scheduling

- For real-time scheduling, scheduler must support preemptive, priority-based scheduling
  - But only guarantees soft real-time
  - For hard real-time must also provide ability to meet deadlines

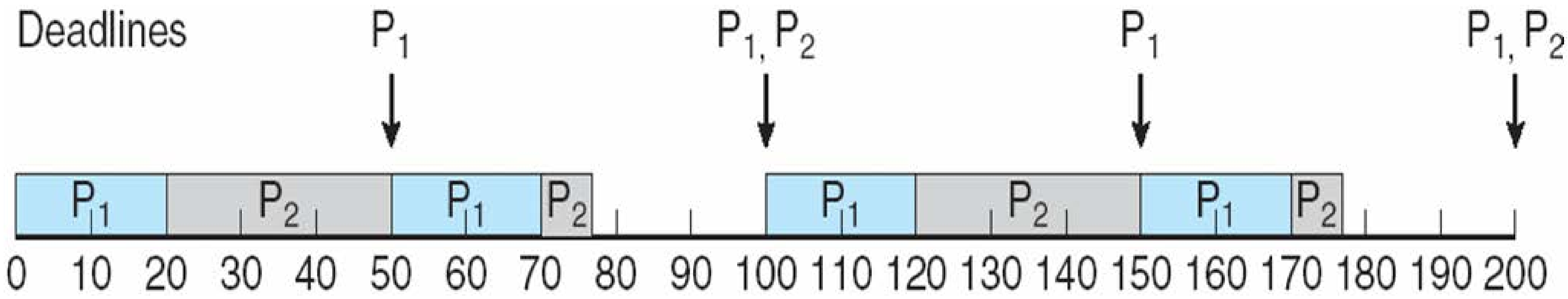
- Processes have new characteristics: periodic ones require CPU at constant intervals
  - Has processing time t, deadline d, period p
  - $0 \leq t \leq d \leq p$
  - Rate of periodic task is $1/p$



1) Rate-monotonic scheduling

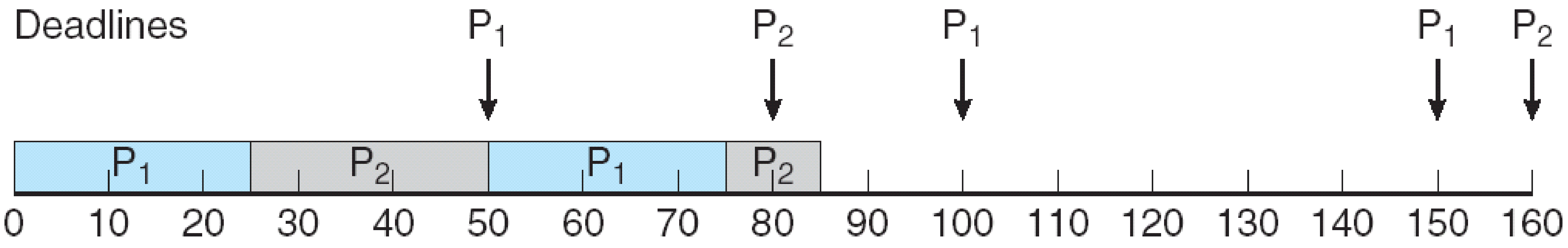
- A priority is assigned based on the inverse of its period
- Shorter periods = higher priority;
- Longer periods = lower priority
- P_1 is assigned a higher priority than P_2

Process	Period	CPU burst time	Utilization
P_1	50	20	20/50=0.4
P_2	100	35	35/100=0.35
			total=0.75



Missed deadlines with Rate Monotonic scheduling

Process	Period	CPU burst time	Utilization
P_1	50	25	25/50=0.5
P_2	80	35	35/80=0.44
			total=0.94



Limitation of CPU utilization in Rate-Monotonic

- **CPU utilization in RM is bounded!**

$$U_{max}(N) = N\,(2^{1/N}-1)$$

- N=1 → U(1) = 100%
- N=2 → U(2) = 83%
- N=inf → U(inf) = 69%

if $U_{sys} <= U_{max}$ ➔ Rate Monotonic is feasible
Otherwise you need to check (possible)

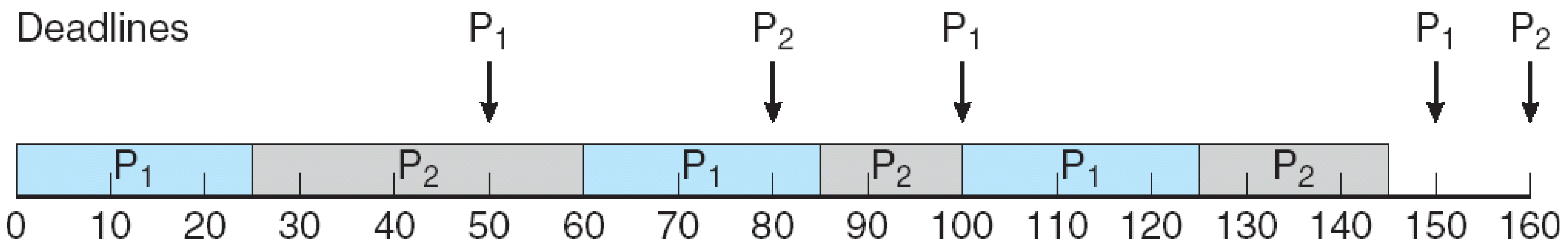
2) Earliest-deadline-first scheduling (EDF)

- Priorities are assigned according to deadlines:

  the earlier the deadline, the higher the priority;

  the later the deadline, the lower the priority

Process	Period	CPU burst time	Utilization
P_1	50	25	25/50=0.5
P_2	80	35	35/80=0.44
			total=0.94



3) Proportional share scheduling

- T shares are allocated among all processes in the system
- An application receives N shares where $N < T$
- This ensures each application will receive N / T of the total processor time

Algorithm evaluation

- How to select CPU-scheduling algorithm for an OS?
  - Determine criteria, then evaluate algorithms
- Deterministic modeling
  - FCFS is 28ms
  - Non-preemptive SJF is 13ms
  - RR is 23ms

Process	Burst Time
P_1	10
P_2	29
P_3	3
P_4	7
P_5	12

- Queueing models
  - n = average queue length
  - W = average waiting time in queue
  - λ = average arrival rate into queue
  - Little's law – in steady state, processes leaving queue must equal processes arriving, thus:
    $n = \lambda \times W$
- Simulations
  - Programmed model of computer system

Implementation

- Even simulations have limited accuracy
- Just implement new scheduler and test in real systems
- High cost, high risk
- Environments vary
- Most flexible schedulers can be modified per-site or per-system
- Or APIs to modify priorities
- But again environments vary

Questions?