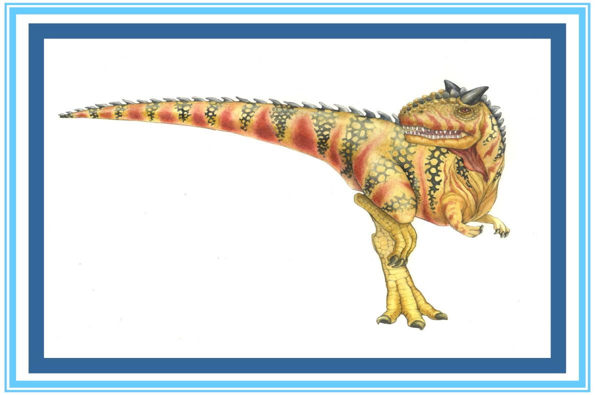
Chapter 5: Process Scheduling

School of Computing, Gachon Univ.
Jungchan Cho



Most slides from "Operating System Concepts – 10th Edition".
Many slides are taken from lecture notes of Prof. Joon Yoo.

Objectives

- To introduce CPU scheduling, which is the basis for multiprogrammed operating systems

- To describe various CPU-scheduling algorithms

- To discuss evaluation criteria for selecting a CPU-scheduling algorithm for a particular system

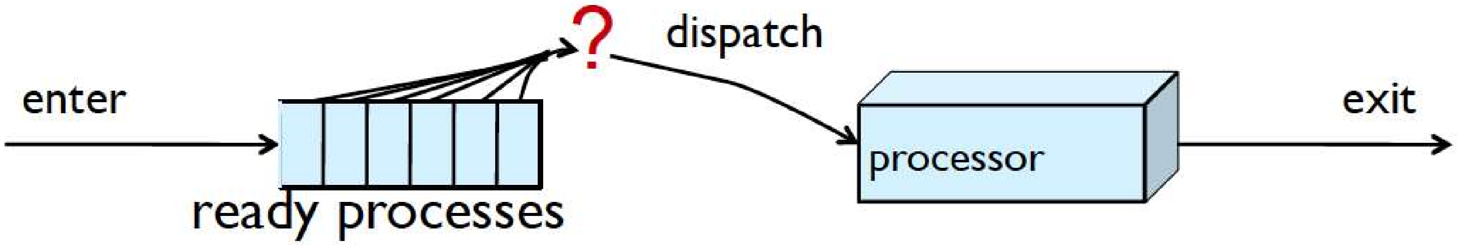
Chapter 5: Process Scheduling

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Advanced Scheduling
  - Multiple-Processor Scheduling
  - Real-Time CPU Scheduling

Introduction

- Multitasking: multiple processes in the system with one (or more) processor(s)
- The role of OS
  - Increases CPU utilization by organizing processes so that the CPU always has one to execute
  - Process scheduling
    - Allocates processor (CPU) time slots to processes
  - **Note:** Terms "**Process/Thread/CPU/Job/Task** scheduling" will be used inter-changeably in this chapter.

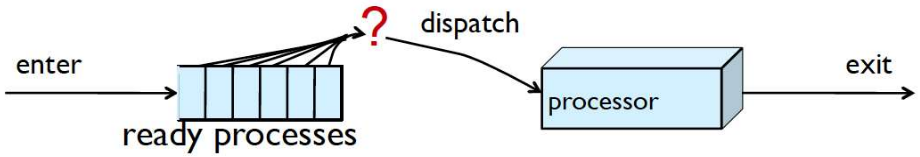
Introduction : Basic Concept



- **CPU Scheduling**
  - **<u>The action</u>** that selects among the processes (threads) in memory that are ready to execute, and allocates the CPU to one of them
  - Given a set of ready processes (or threads),
    - Which one should I run next?
    - How long should it run?

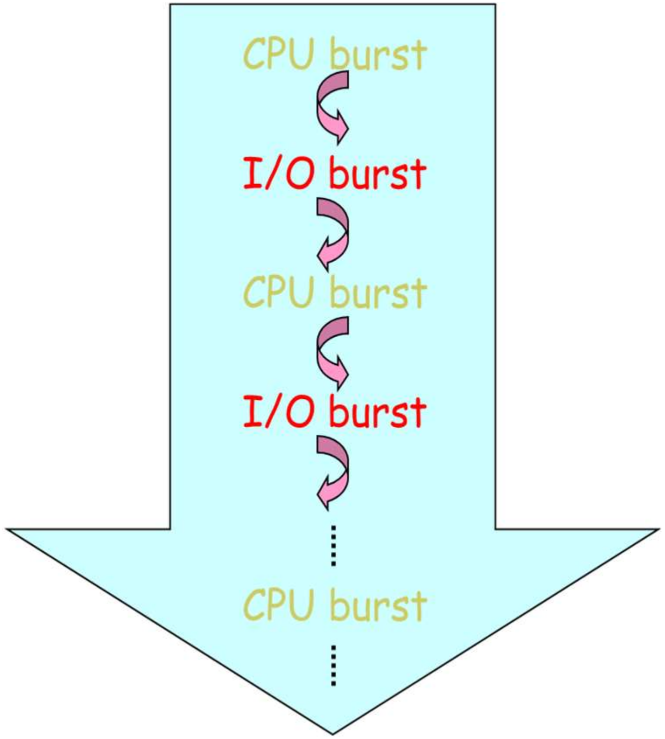
Introduction : Basic Concept

- **CPU scheduler**
  - **An OS component module** that performs the process scheduling
  - Allocates CPU time slots to processes
- **Dispatcher** is a module (in the scheduler) that gives control of the CPU to the process selected by the scheduler
  - **context switching**
  - switching to **user mode**
  - jumping to the proper location (**PC**) in the user program to restart that program



CPU & I/O Bursts

- Types of workloads : CPU burst vs. I/O burst
  - **CPU burst**
    - Cycles of CPU executions
    - e.g., sorting a million-entry array in RAM
  - **I/O burst**
    - Cycles of I/O waits
    - e.g., disk read/write, networking, printing
- Process execution consists of cycles of CPU bursts and I/O bursts (or waits)
  - **CPU burst time** is an important factor(criteria) for scheduling algorithms



CPU & I/O Bursts (cont.)

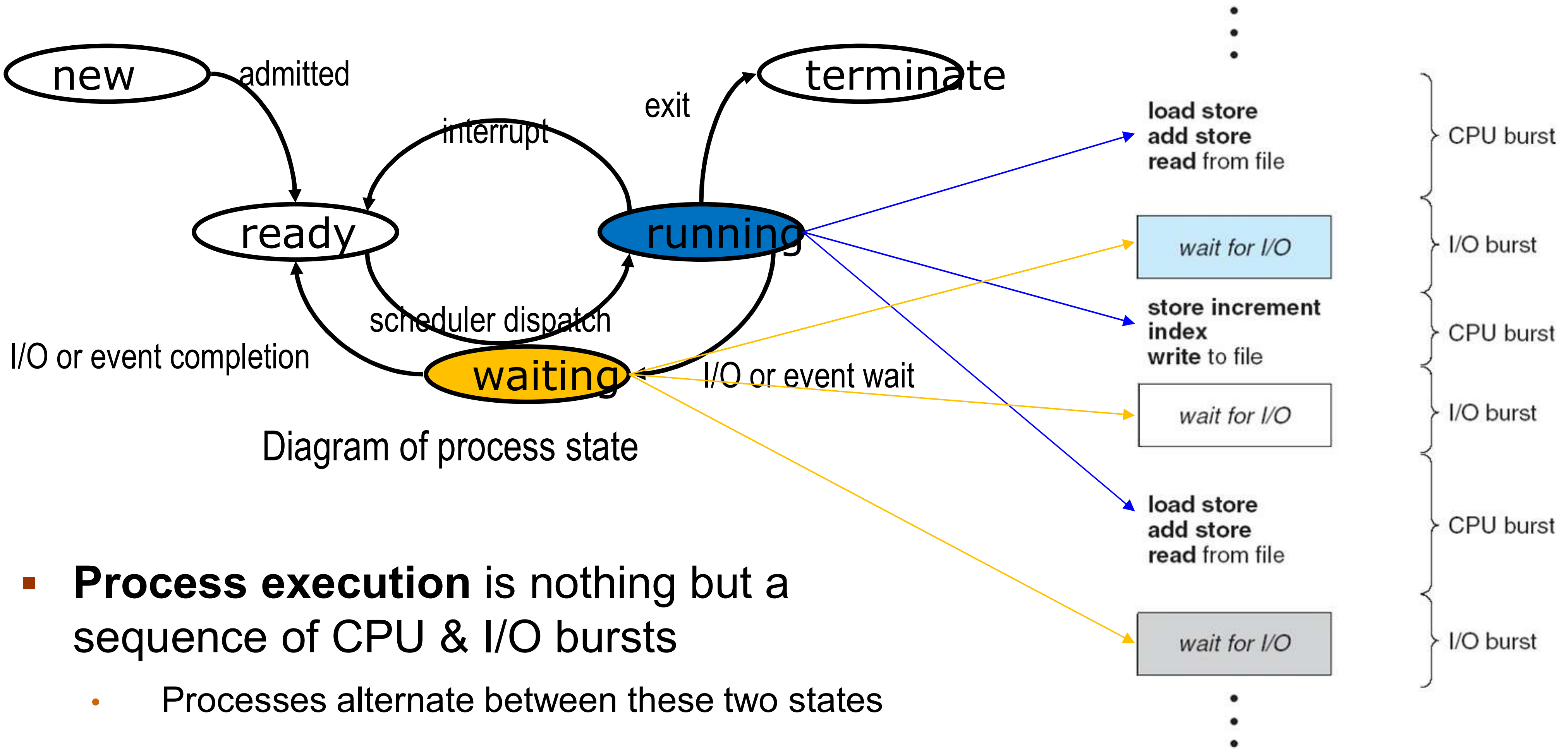
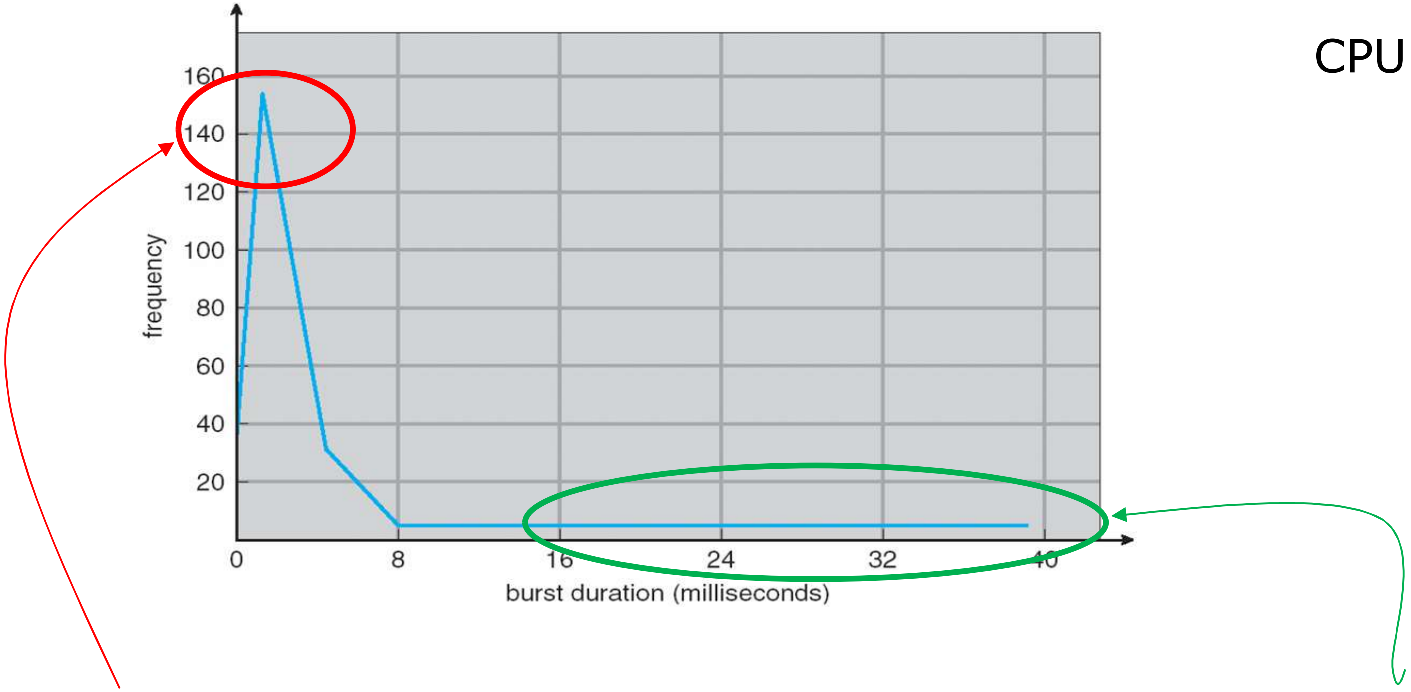


Diagram of process state

- **Process execution** is nothing but a sequence of CPU & I/O bursts
  - Processes alternate between these two states

Histogram of CPU-burst Durations



CPU bursts distribution

- Large number of **short CPU bursts** and small number of **long CPU bursts**
  - **I/O bound process**: Short CPU burst
  - **CPU bound process**: Long CPU burst
- Important for CPU scheduling algorithm selection

Chapter 5: Process Scheduling

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Advanced Scheduling
  - Multiple-Processor Scheduling
  - Real-Time CPU Scheduling

Scheduling Criteria

- **Goal** of CPU scheduling
  - Improving **system performance**

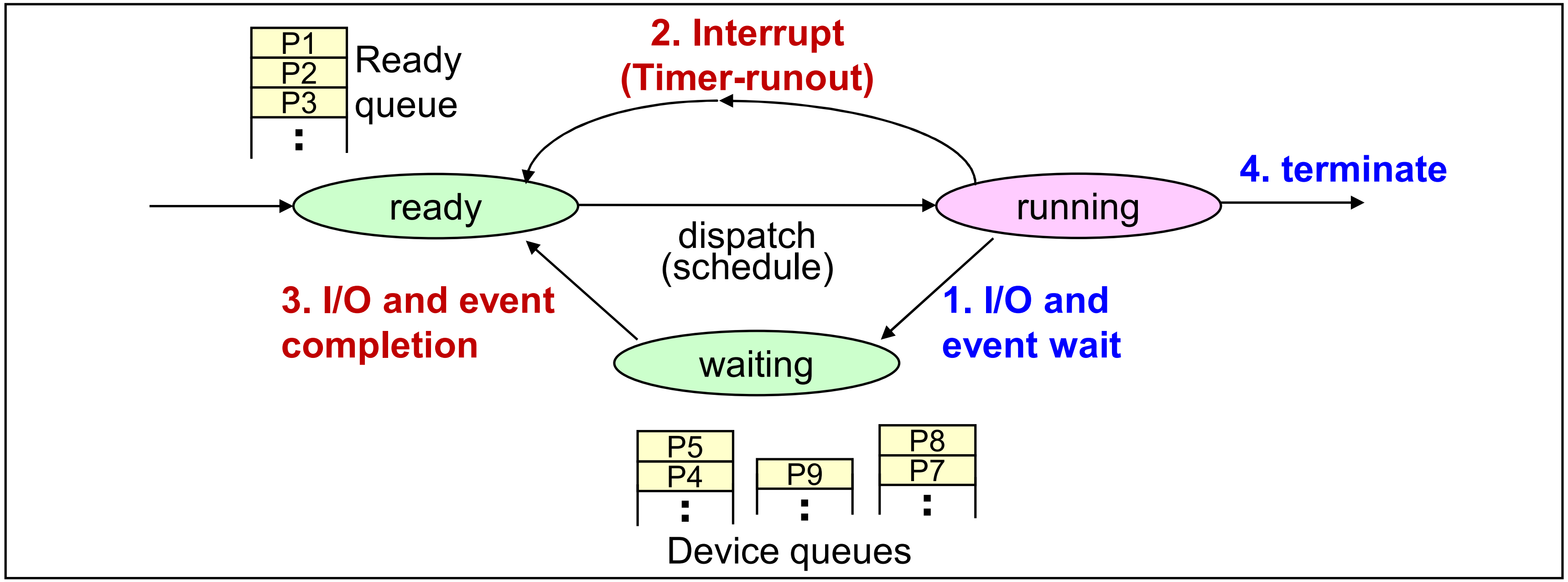
- System performance?
  - **CPU utilization, Throughput** (Jobs/time)**, Turnaround time, ...**
  - **Waiting time** – sum of time spent waiting only in the ready queue
  - **Response time** – amount of time it takes from when a request was submitted until the first ***response*** is produced (not final output results)
    - e.g., Interactive system

Scheduling Algorithm Optimization Criteria

- So, what is the objective?
  - **Max CPU utilization, Max throughput, Min turnaround time, Min waiting time, Min response time**

- Each system selects a scheduling policy considering the application
  - e.g., real-time system vs. general-purpose system
  - Special purpose (e.g., “real-time”) scheduling exists

Re: Process and State Transition



- Three states : ready, running, and waiting
- CPU scheduling decisions may take place when a process:

Must dispatch a new process

1. **Switches from running to waiting state**
2. Switches from running to ready state
3. Switches from waiting to ready
4. **Terminates**

Can make a decision!
- should scheduler dispatch a new process or continue with old one?

Preemptive vs. Non-preemptive

1. Switches from running to waiting state
2. Switches from running to ready state
3. Switches from waiting to ready
4. Terminates

- **Non-preemptive**
  - Once CPU is allocated to a process, it holds it till it
    - It gives up by itself (스스로): Terminates (case 4) or blocks itself to wait for I/O – (case 1)
    - It does not give up for case 2 & 3
  - Windows 3.0

- **Preemptive**
  - Currently running process may be **interrupted** and moved to the ready state by the OS
    - Most modern OSes: Windows 95 – Windows 10, Mac OS X, Linux, ...

Preemptive vs. Non-preemptive

- **Non-preemptive scheduling**
  - Process uses the CPU until it voluntarily releases it - No preemption
  - Pros: Low context switch overhead
  - Cons : May result in longer mean response time
- **Preemptive scheduling**
  - CPU may be preempted to another process independent of the intention of the running process
  - Pros: Low response time - for time-sharing systems and real-time systems
  - Cons: High context switching overhead, race conditions (need synchronization – Ch. 6)

Preemptive vs. Non-preemptive

- Which scheduling policy
  - can immediately take care of Interrupts?
  - has less context-switching overhead?
  - has less response time?
  - can use time-sharing?

Chapter 5: Process Scheduling

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Advanced Scheduling
  - Multiple-Processor Scheduling
  - Real-Time CPU Scheduling

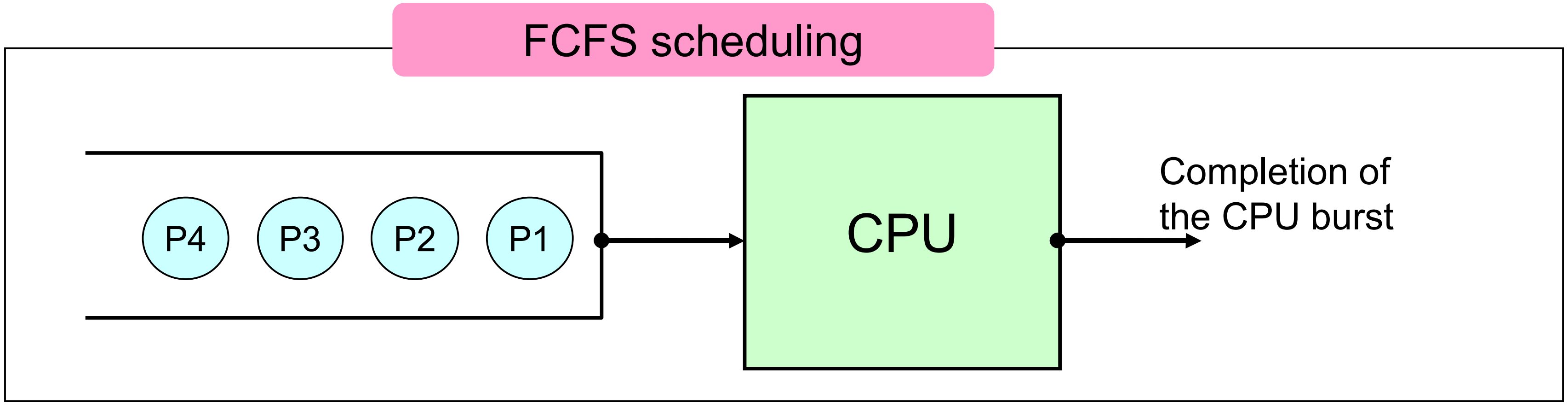
Scheduling Algorithms

- CPU Scheduling Algorithm
  - Algorithm that decides which of the processes in the ready queue is to be allocated the CPU

- Various algorithms
  - First-Come, First-Served (FCFS) Scheduling
  - Shortest-Job-First (SJF) Scheduling
  - Priority Scheduling
  - Round-Robin Scheduling
  - Earliest Deadline First Scheduling

First-Come, First-Served (FCFS)

- FCFS(First-Come-First-Served) scheduling
  - Simplest scheme using FIFO queue



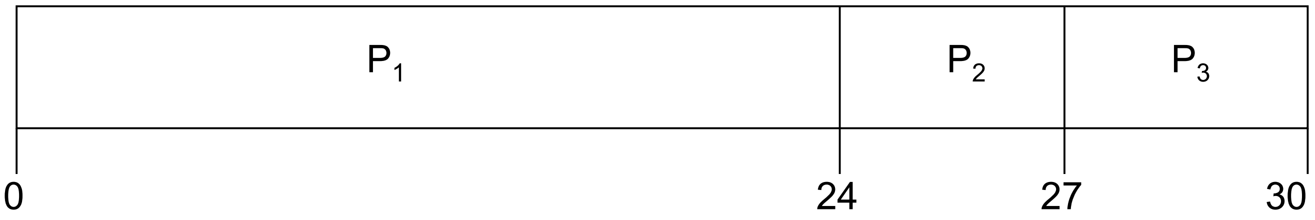
  - Non-preemptive scheduling
  - Scheduling criteria
    - Arrival time (at the ready queue)
    - Faster arrival time process first

First-Come, First-Served (FCFS) Scheduling

All arrived at time 0 in order P_1, P_2, P_3

Process	CPU 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**

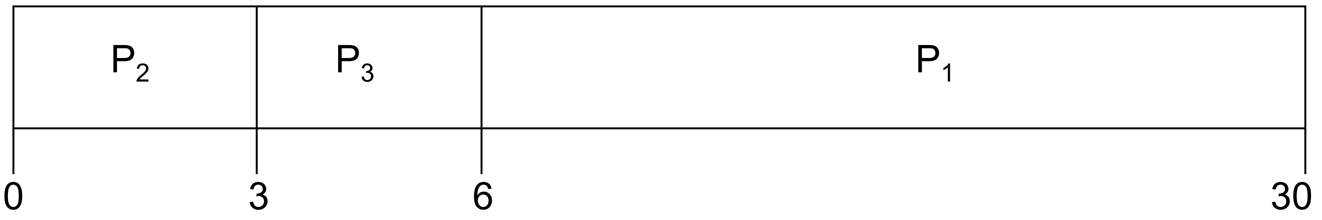
Can we do better?

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 = **<u>3</u>**
- Much better than previous case

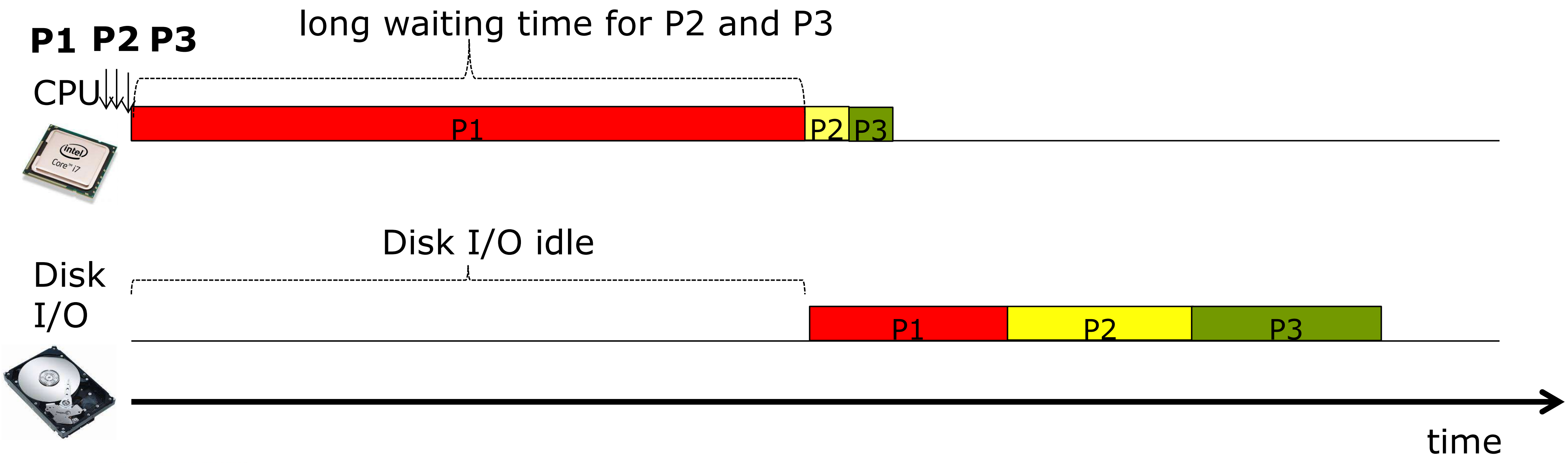
FCFS

- FCFS is non-preemptive
  - Once a CPU has been allocated a process, that process keeps the CPU until either by **terminating** or by **requesting I/O**
  - not good for time-sharing systems – why?

- **Convoy Effect** (next slide example)
  - Larger waiting time for I/O bound process
  - Low CPU and device utilization

Convoy Effect

- I/O bound process: P2, P3
  - 10ms CPU, 100ms disk I/O
- CPU bound process: P1
  - 200ms CPU, 100ms disk I/O



P1 P2 P3

long waiting time for P2 and P3

CPU

P1 P2 P3

Disk I/O idle

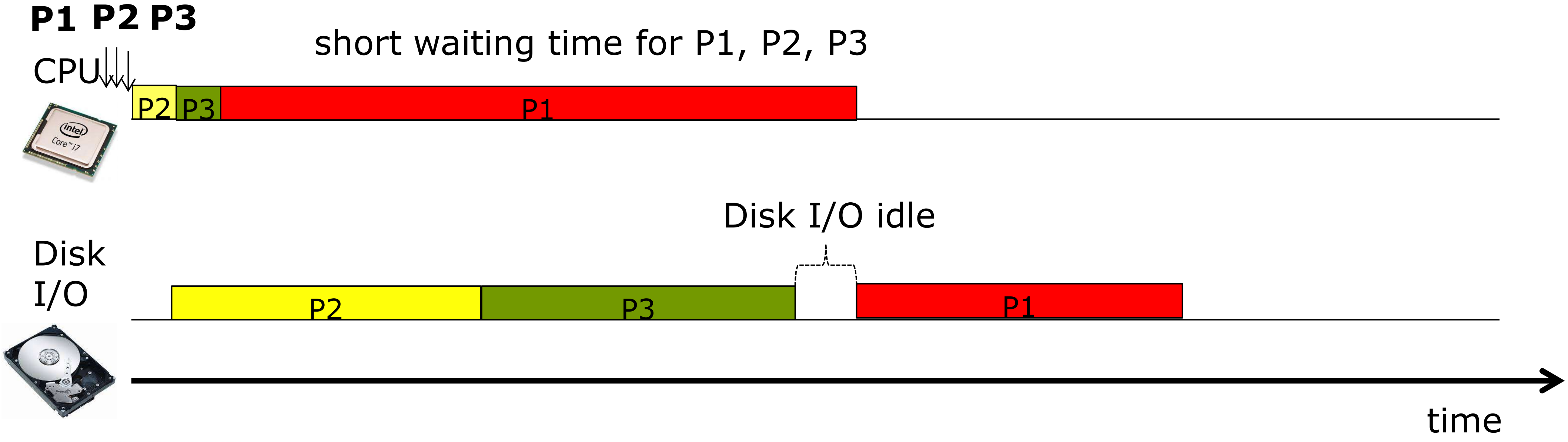
Disk I/O

P1 P2 P3

time

What should be better?

- I/O bound process: P2, P3
  - 10ms CPU, 100ms disk I/O
- CPU bound process: P1
  - 200ms CPU, 100ms disk I/O

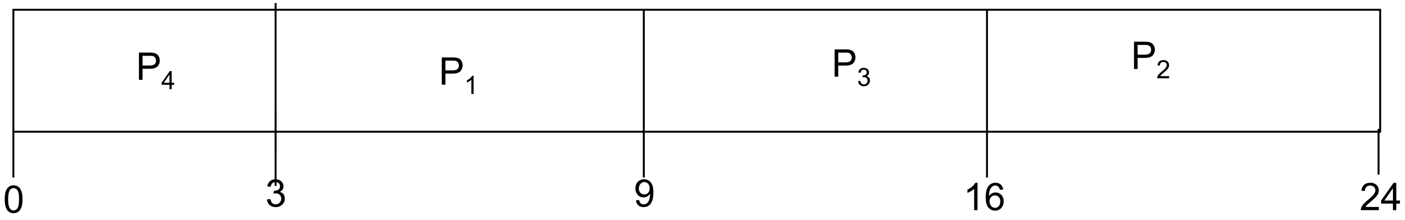


Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst
  - schedule the process with the shortest time

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF scheduling chart



P_4 P_1 P_3 P_2

0 3 9 16 24

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

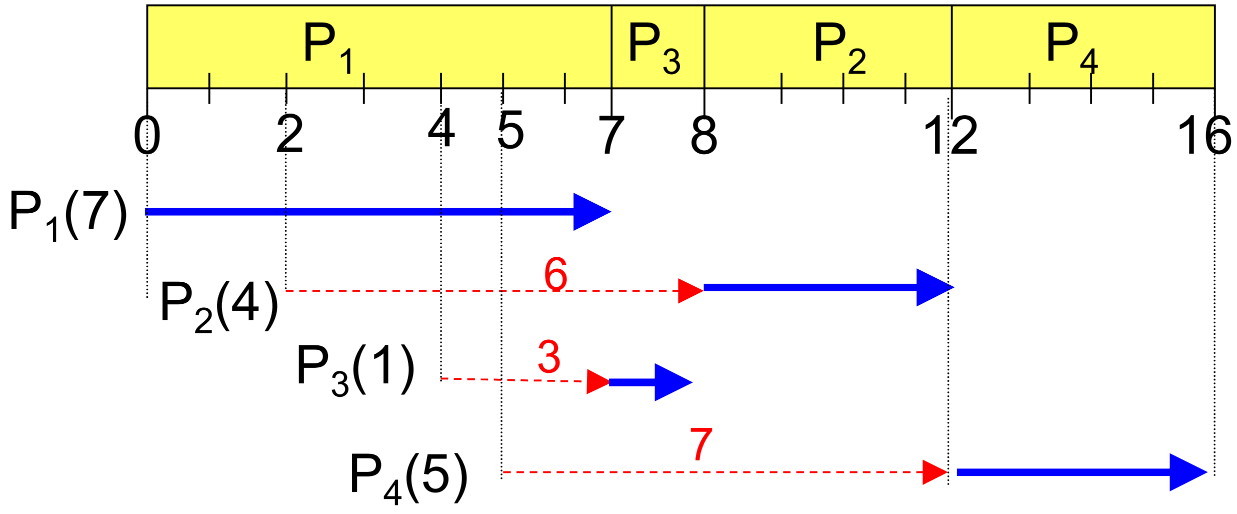
SJF Scheduling (Cont.)

- A new process arrives at the ready queue – what can we do next?
- Two schemes:
  - **Nonpreemptive SJF**
    - Once CPU given to the process it cannot be preempted until completes its CPU burst
  - **Preemptive SJF**
    - If a new process arrives with CPU burst length less than **remaining time** of current executing process, preempt.
    - This scheme is known as the **Shortest-Remaining-Time-First (SRTF)**

Example of Non-Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

- **SJF (non-preemptive)**



P_1's wating time = 0

P_2's wating time = 6

P_3's wating time = 3

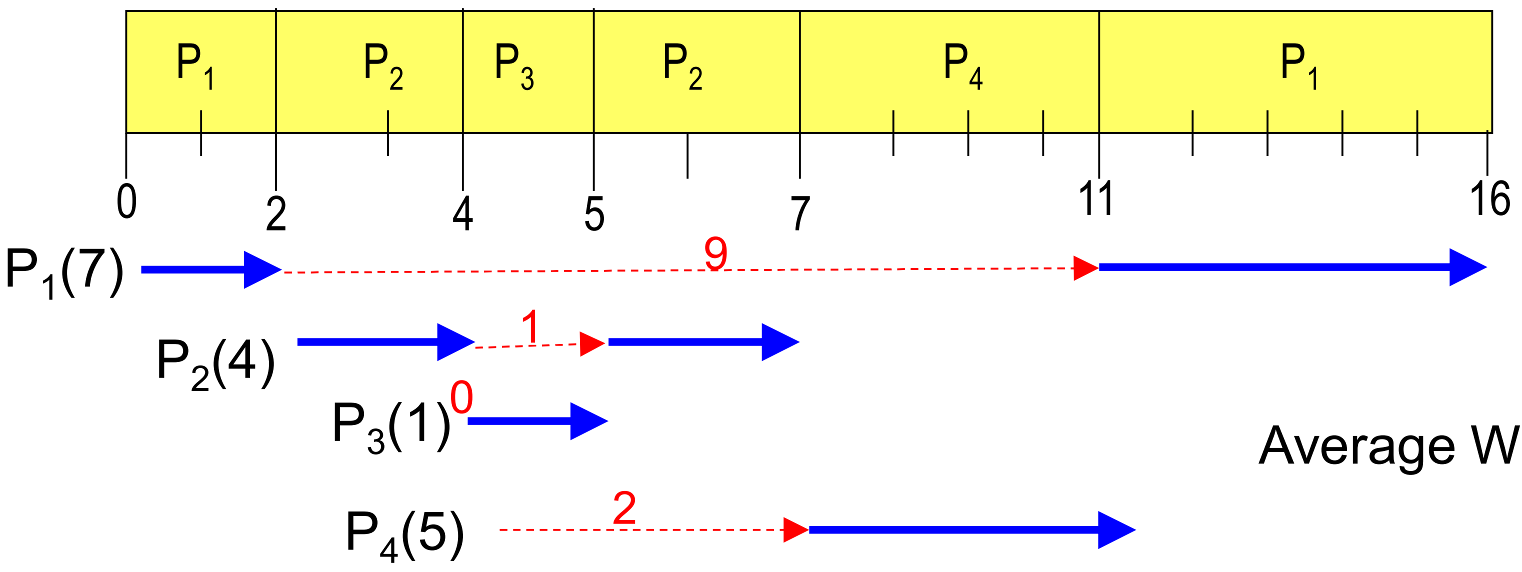
P_4's wating time = 7

Average WT = (0 + 6 + 3 + 7)/4 = 4

Example of Preemptive SJF (SRTF)

Process	Arrival Time	Burst Time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

- **SRTF (preemptive SJF)**



P_1's wating time = 9

P_2's wating time = 1

P_3's wating time = 0

P_4's wating time = 2

Average WT = (9 + 1 + 0 +2)/4 = 3

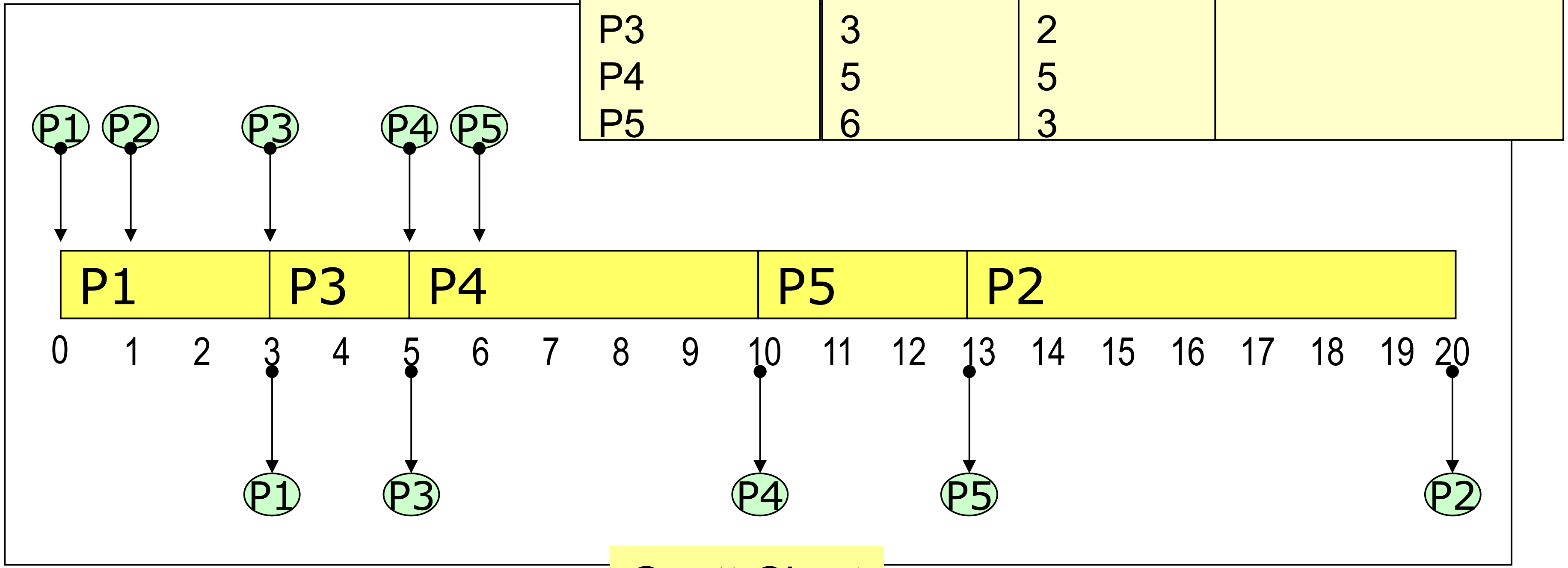
미래창조과학부 SW중심대학
가천대학교 소프트웨어학과
Gachon University Department of Software

Example of SJF

- Preemptive or Non-preemptive ?

Example 2:
SJF Scheduling

Process ID	Arrival time	Burst time	Wait Time
P1	0	3	
P2	1	7	
P3	3	2	
P4	5	5	
P5	6	3	



Gantt Chart

SJF Scheduling (Cont.)

- Pros
  - Gives **minimum average waiting time** for a given set of processes
  - Minimizes the number of processes in the system
    - Reduces the size of the ready queue
    - Reduces the overall space requirements
  - Fast responses to many processes

- Cons
  - **Starvation**, indefinite postponement (blocking)
    - Long burst-time processes
    - Can be solved by **aging**

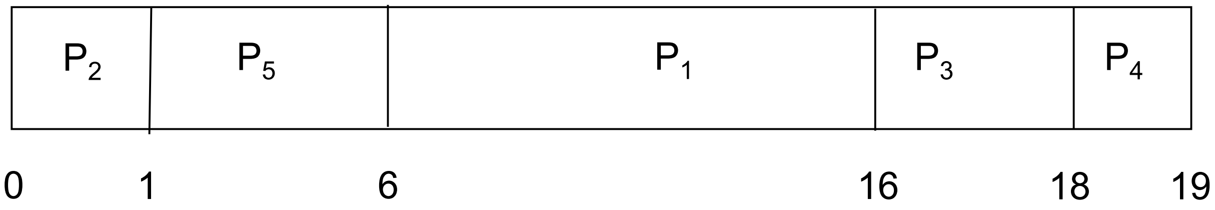
Priority Scheduling

- SJF algorithm is a special case of **priority scheduling**
- A **priority** is associated with each process
- The CPU is allocated to the process with the **highest priority** (smallest integer ≡ highest priority)
  - Preemptive
  - Nonpreemptive
- SJF is priority scheduling where priority is the inverse of predicted next CPU burst time

Example of Priority Scheduling

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



P_2 P_5 P_1 P_3 P_4

0 1 6 16 18 19

- Average waiting time = 8.2 msec

- SJF : Special case of the general priority scheduling
- FCFS : Equal priority

Priority Scheduling

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

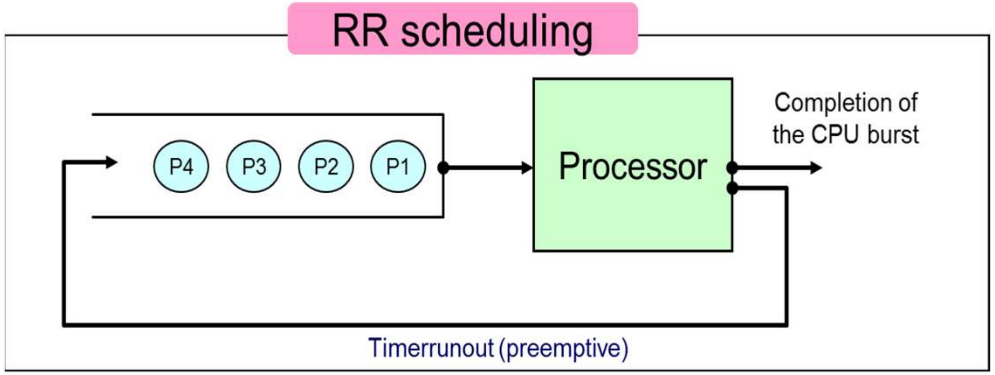
Example of Priority Scheduling

- HRRN(Highest Response Ratio Next) Scheduling
  - Nonpreemptive
  - modification of shortest job first (SJF) to mitigate the problem of process starvation.

$$\mathit{response\ ratio} = \frac{\mathit{waiting\ time} + \mathit{estimated\ run\ time}}{\mathit{estimated\ run\ time}} = 1 + \frac{\mathit{waiting\ time}}{\mathit{estimated\ run\ time}}$$

Round Robin (RR)

- **RR (Round-Robin) scheduling**



- Preemptive scheduling
- Scheduling criteria
  - Arrival time (at the ready queue) = FIFO

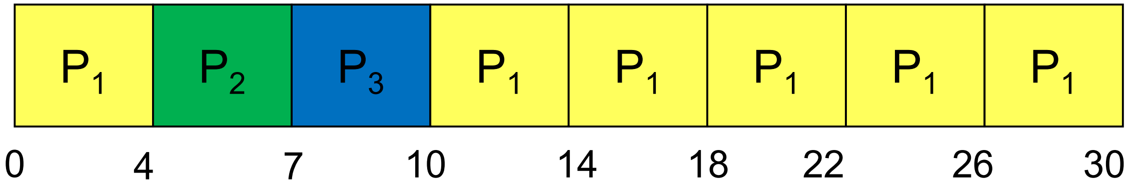
RR (Round-Robin) scheduling

- RR (Round-Robin) scheduling
  - **Time quantum** (=**time slice**) for each process
    - System parameter (generally from 10 to 100 ms in length)
  - The CPU scheduler sets a timer to 1 time quantum
    - An interrupt will occur after 1 time quantum
    - The (running) process that has exhausted his time quantum releases the CPU and goes to the ready state (timerrunout)
    - A context switch occurs!

Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:



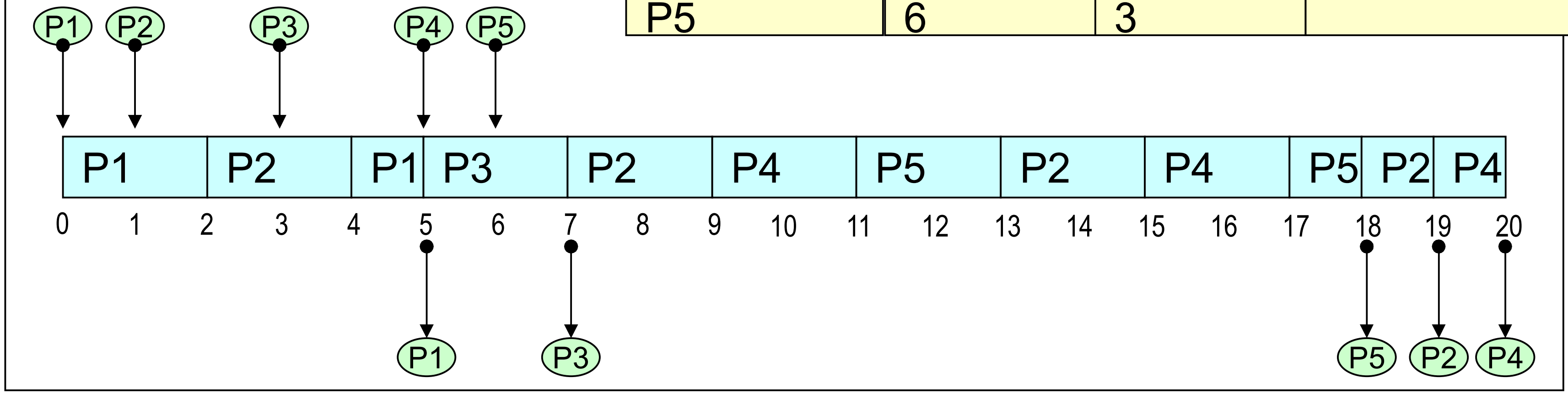
- Typically, higher average **turnaround time** than SJF, but better ***response time***
- q should be large compared to context switch time
- q usually 10ms to 100ms, context switch < 10 usec

Round Robin Scheduling

Example:
RR Scheduling

Time Quantum = 2

Process ID	Arrival time	Burst time	Wait Time
P1	0	3	
P2	1	7	
P3	3	2	
P4	5	5	
P5	6	3	



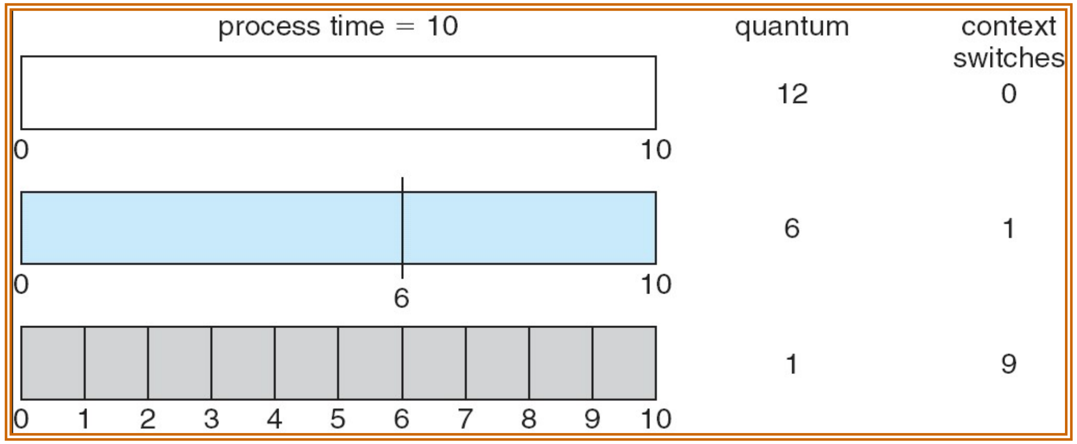
Gantt Chart

RR (Round-Robin) scheduling

- RR (Round-Robin) scheduling
  - Pros
    - Prevents monopoly of the CPU by a process
    - Adequate for interactive/time-sharing system
  - Cons
    - High context switching overhead due to preemptions

Round Robin Scheduling

- Performance
  - quantum (q) large ⇒ FCFS
  - q small ⇒ **time-sharing** ⇒ q must be large with respect to **context switch time**, otherwise overhead is too high.



➢ Number of Context Switching
  - quantum = 12 ➔ FCFS
  - quantum = 6 ➔ 2 quanta, 1 Context Switching
  - quantum = 1 ➔ 10 quanta, 9 Context Switching

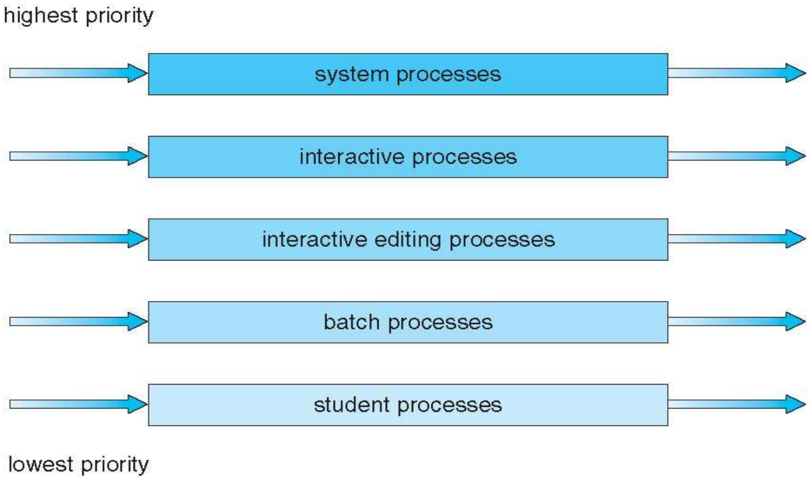
In practice,
- Quantum: 10~100ms
- Context switch time: 10us
 (0.1~0.01% overhead)

Multilevel Queue

- **Multilevel Queue:** Ready queue is partitioned into separate queues, e.g.:
  - Each queue can have its own scheduling algorithm:
    - e.g., foreground – RR, background – FCFS

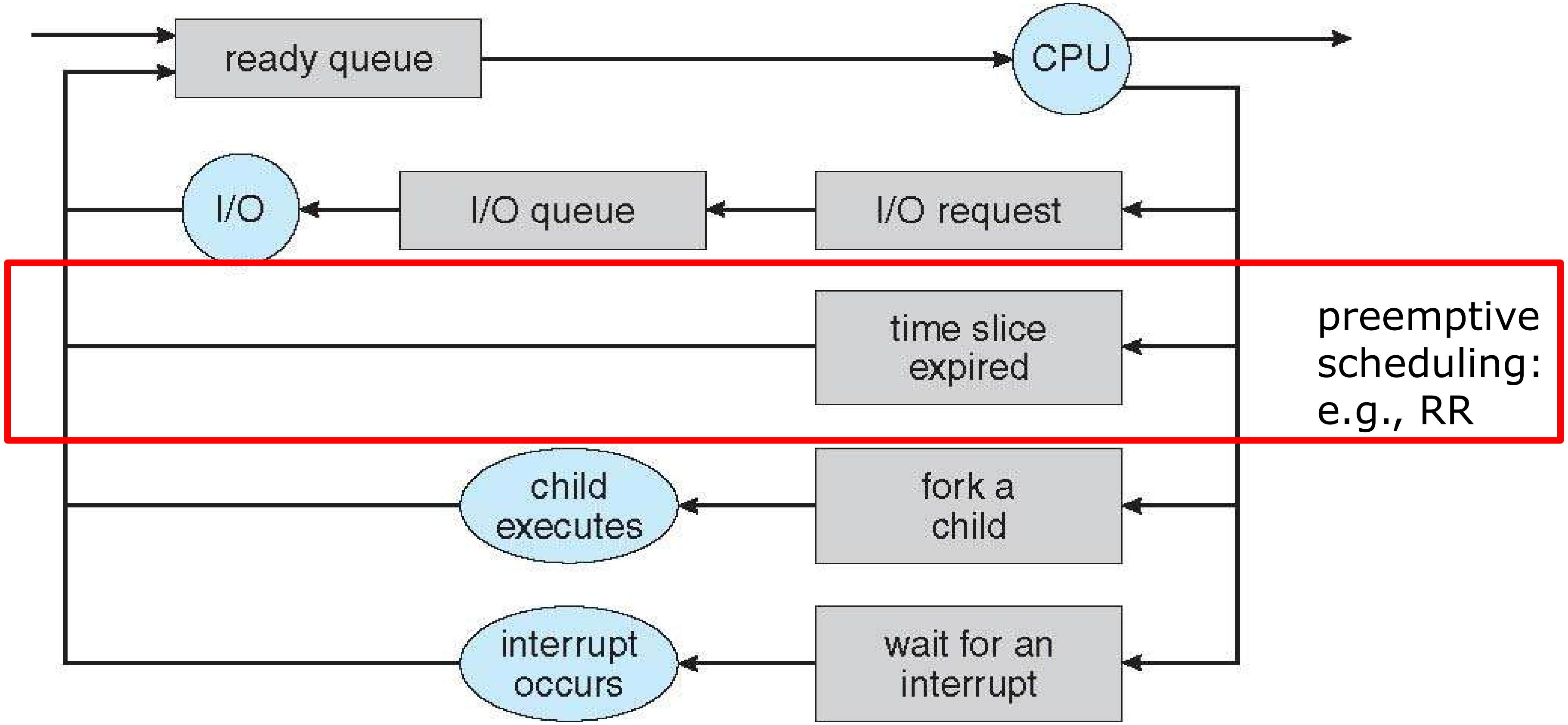
  Scheduling must be done between the queues:

  - Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of starvation.
  - Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes;
    - e.g., 80% to foreground in RR, 20% to background in FCFS



Process Sceduling

- Queueing diagram representation of process scheduling



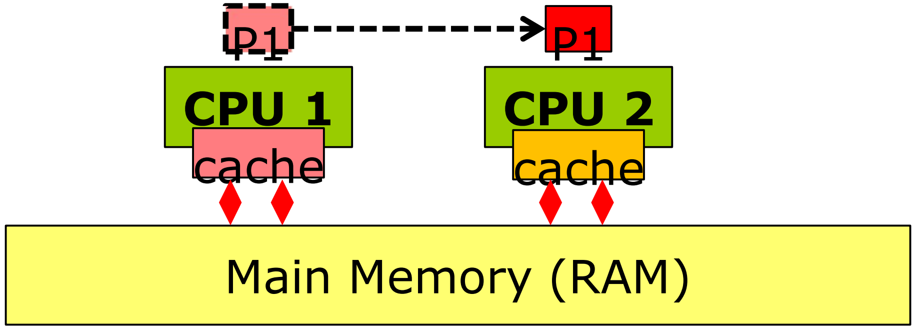
preemptive scheduling: e.g., RR

Chapter 5: Process Scheduling

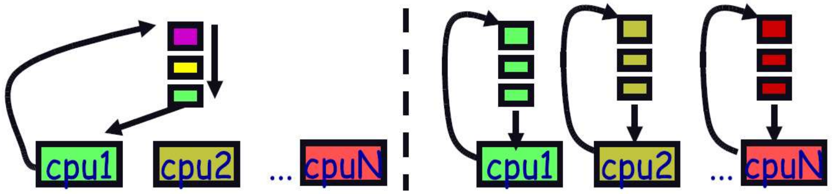
- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Advanced Scheduling
  - Multiple-Processor Scheduling
  - Real-Time CPU Scheduling

Multiple-Processor Scheduling

- CPU scheduling more complex when multiple CPUs are available
  - Must decide on which CPU to run which process
- Moving processes between CPUs has costs
  - More cache misses



- **Affinity scheduling**
  - try to keep processes on same CPU



Multiple-Processor Scheduling

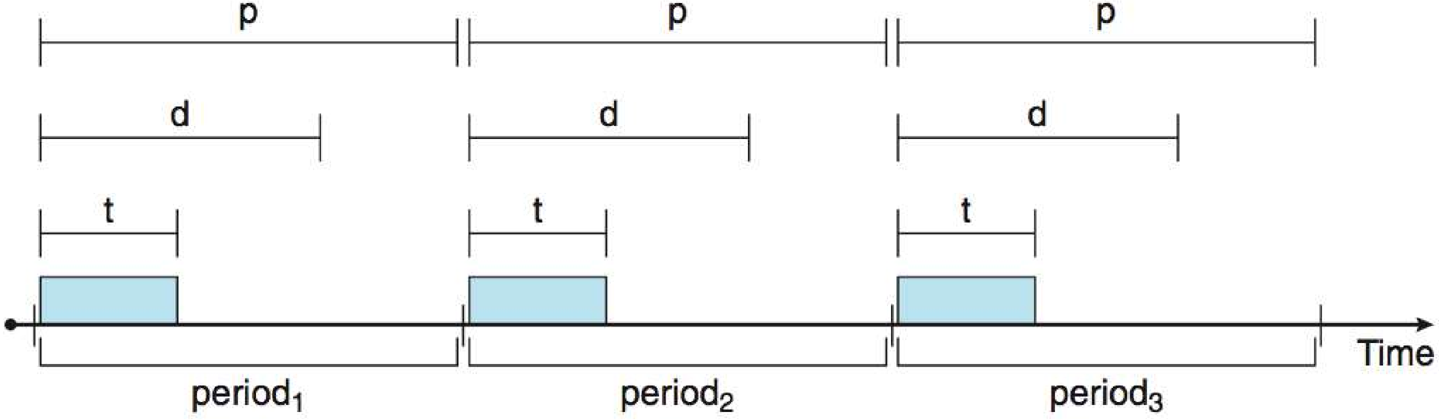
- **Load balancing**
  - Keep the workload between processors balanced
  - Migration
    - Move tasks from a busy processor to an idle processor
  - Load balancing may counteract the benefits of Affinity Scheduling
    - Can you understand why?

Chapter 5: Process Scheduling

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Advanced Scheduling
  - Multiple-Processor Scheduling
  - Real-Time CPU Scheduling

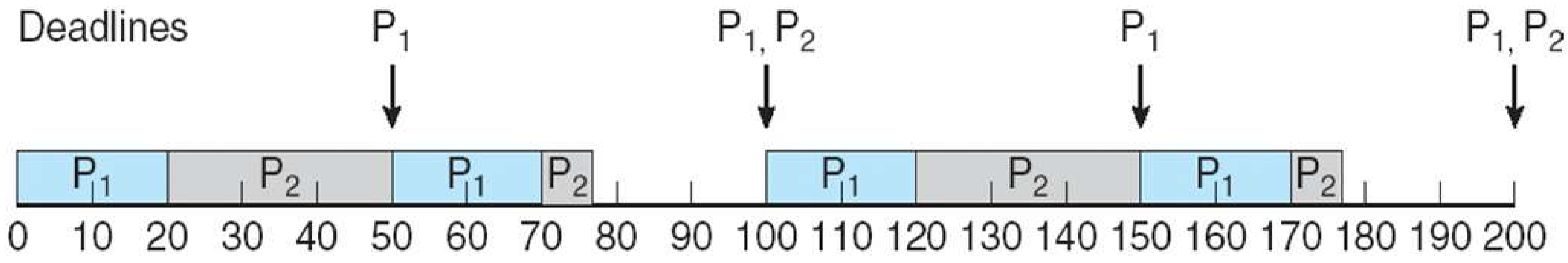
Real-time scheduling

- **Real-time scheduling** applications must provide ability to meet **deadlines**
- Processes have new characteristics: **periodic** ones require CPU at constant intervals
  - Has processing **time *t***, **deadline *d*, period *p***
  - $0 \le t \le d \le p$
  - **Rate** of periodic task is $1/p$



Rate Monotonic Scheduling

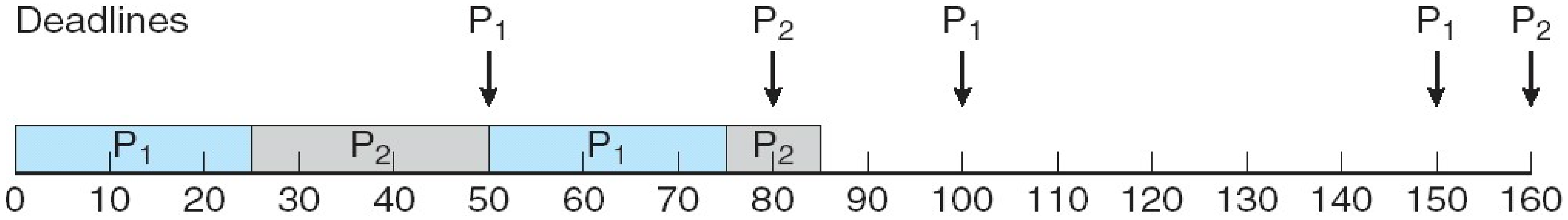
- A priority is assigned based on the inverse of its period (=rate)
  - Shorter periods = higher priority;
  - Longer periods = lower priority
- Example
  - periods: p_1 = 50, p_2 = 100, CPU burst t_1 = 20, t_2 = 35
  - deadline = complete CPU burst before start of next period
  - P_1 is assigned a higher priority than P_2.



Missed Deadlines with Rate Monotonic Scheduling

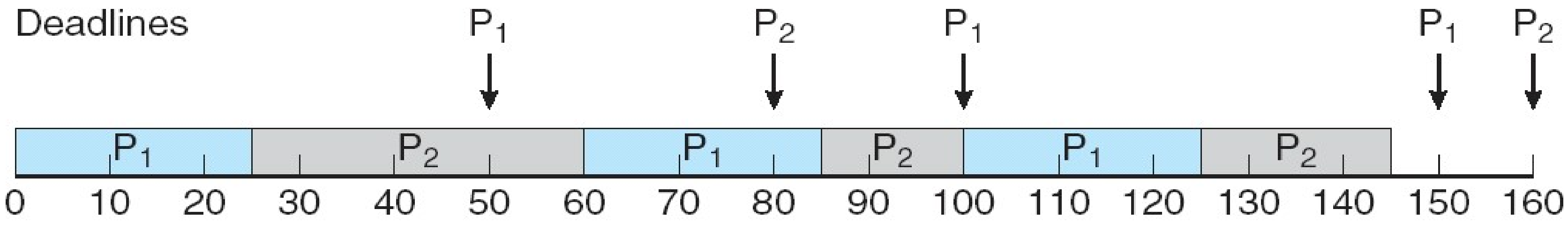
periods: p_1 = 50, p_2 = 80, CPU burst t_1 = 25, t_2 = 35

deadline = complete CPU burst before start of next period



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



  - periods: p_1 = 50, p_2 = 80, CPU burst t_1 = 25, t_2 = 35
  - deadline = complete CPU burst before start of next period

- Appendix
  - CPU Burst Prediction - Moving Average
    - Shortest Job First (SJF) is an optimal scheduling algorithm as it gives maximum Throughput and minimum average waiting time(WT) and turn around time (TAT) but it is not practically implementable because Burst-Time of a process can’t be predicted in advance.
    - We may not know the length of the next CPU burst, but we may be able to predict its value. We expect the next CPU burst will be similar in length to the previous ones.
    - By computing an approximation of the length of the next CPU burst, we can pick the process with the shortest predicted CPU burst.

https://www.geeksforgeeks.org/shortest-job-first-cpu-scheduling-with-predicted-burst-time/

CPU Burst Prediction

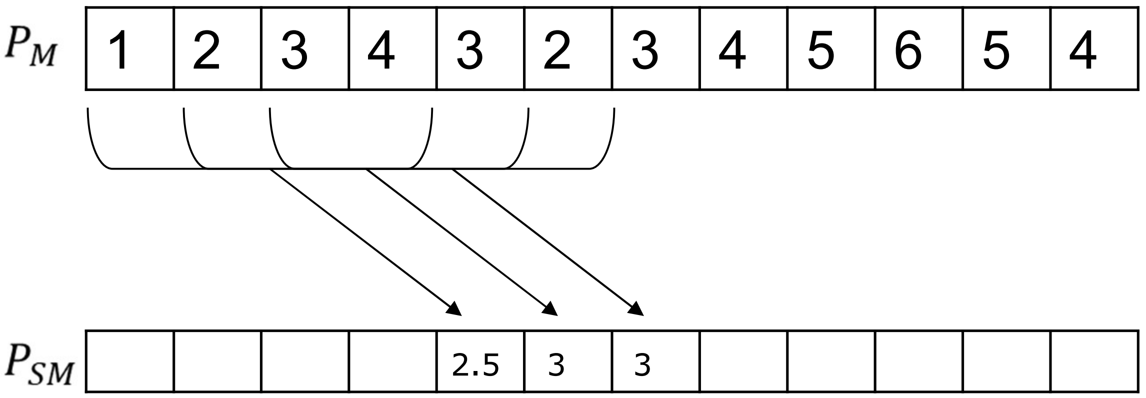
- Moving average
  - In statistics, a moving average (rolling average or running average) is a calculation to analyze data points by creating a series of averages of different subsets of the full data set.
  - Simple Moving Average (SMA)
  - Weighted Moving Average (WMA)
  - Exponential Moving Average (EMA)

CPU Burst Prediction

- Moving average
  - Simple Moving Average
  - simple moving average (SMA) is the unweighted mean of the previous n data.

$$\bar{p}_{\mathrm{SM}} = \frac{p_M + p_{M-1} + \cdots + p_{M-(n-1)}}{n} = \frac{1}{n}\sum_{i=0}^{n-1} p_{M-i}$$

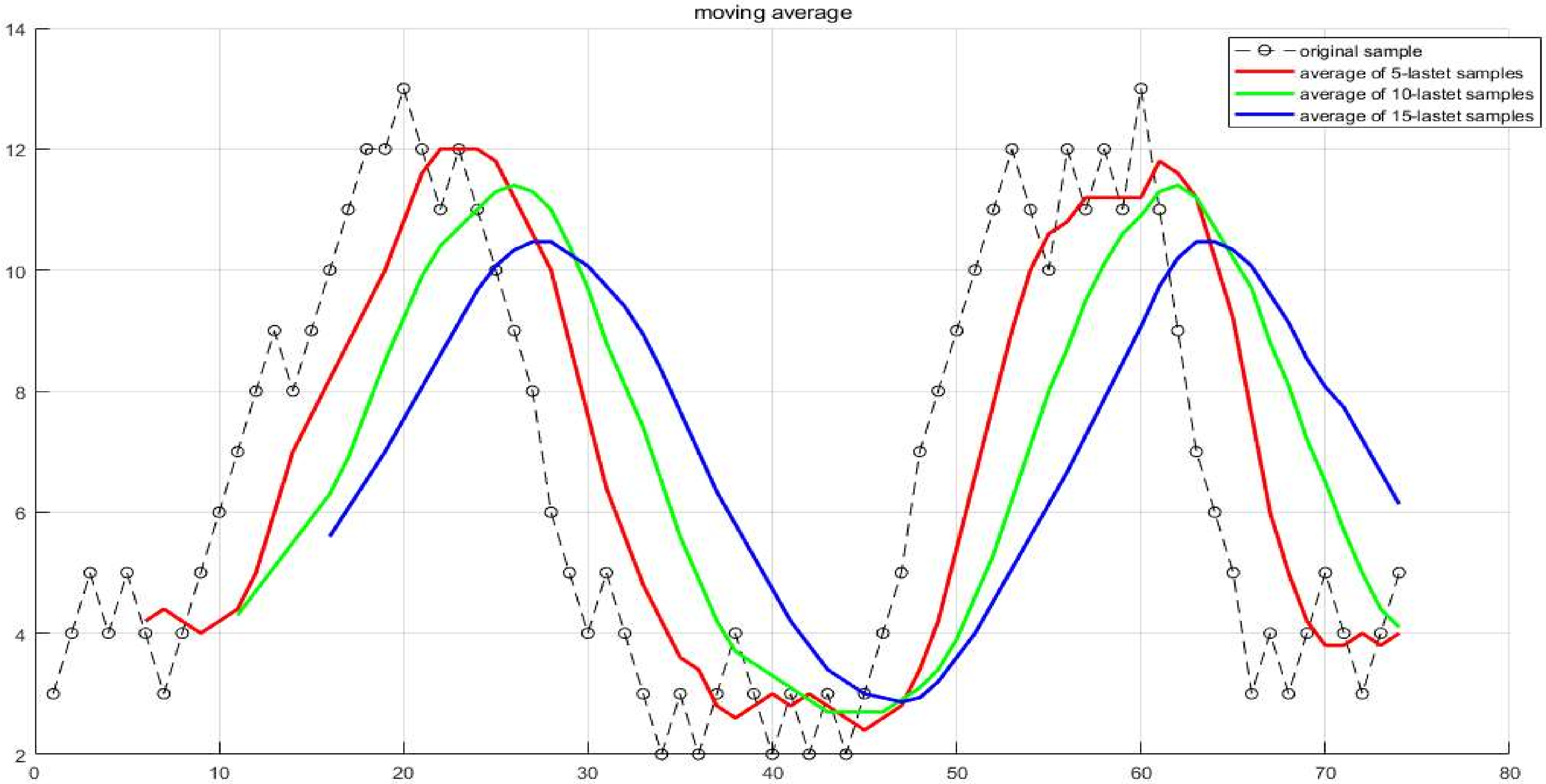
$$\bar{p}_{\mathrm{SM}} = \bar{p}_{\mathrm{SM,prev}} + \frac{p_M}{n} - \frac{p_{M-n}}{n}$$



P_M	1	2	3	4	3	2	3	4	5	6	5	4
P_{SM}					2.5	3	3					

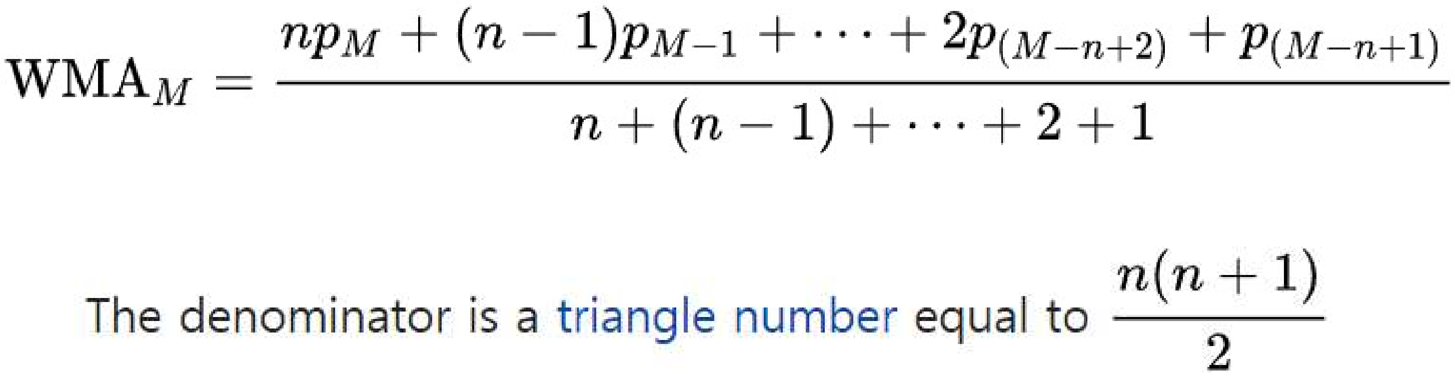
CPU Burst Prediction

- Moving average
  - Example



CPU Burst Prediction

- Moving average
  - Weighted Moving Average
  - a weighted moving average (WMA) has the specific meaning of weights that decrease in arithmetical progression.

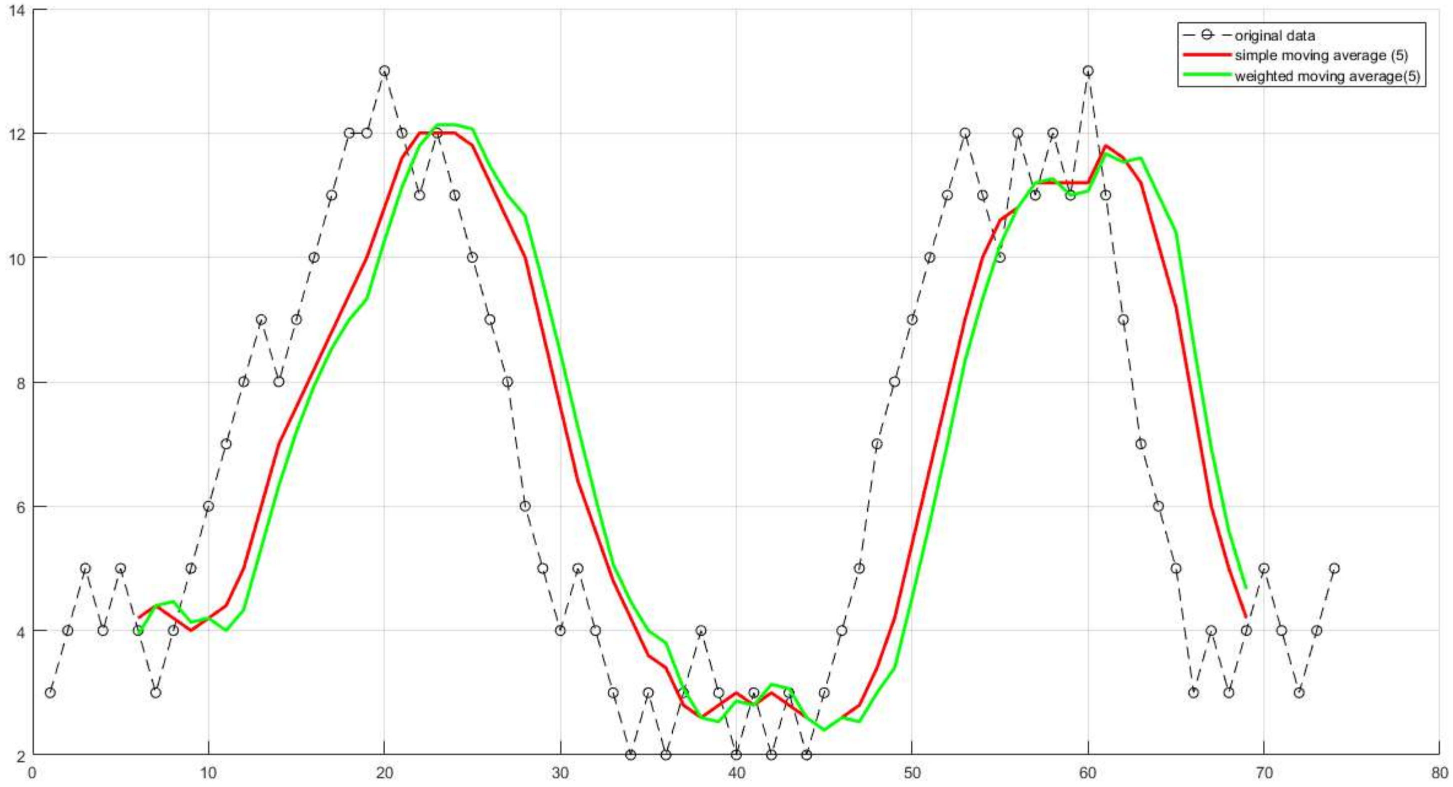


$$\text{WMA}_M = \frac{np_M + (n-1)p_{M-1} + \cdots + 2p_{(M-n+2)} + p_{(M-n+1)}}{n + (n-1) + \cdots + 2 + 1}$$

The denominator is a triangle number equal to $\frac{n(n+1)}{2}$

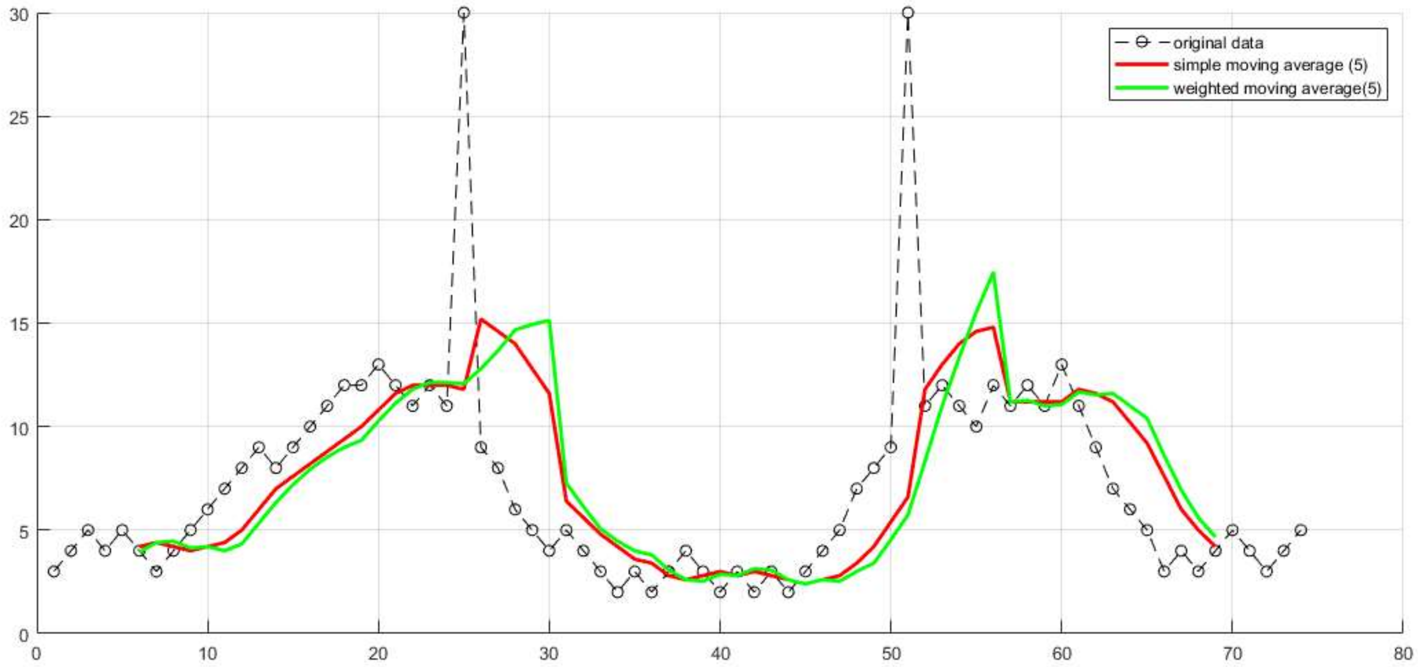
CPU Burst Prediction

- Moving average
  - Example



CPU Burst Prediction

- Characteristics of SMA, WMA
  - We should use previous value
  - A specific value do not suddenly affect

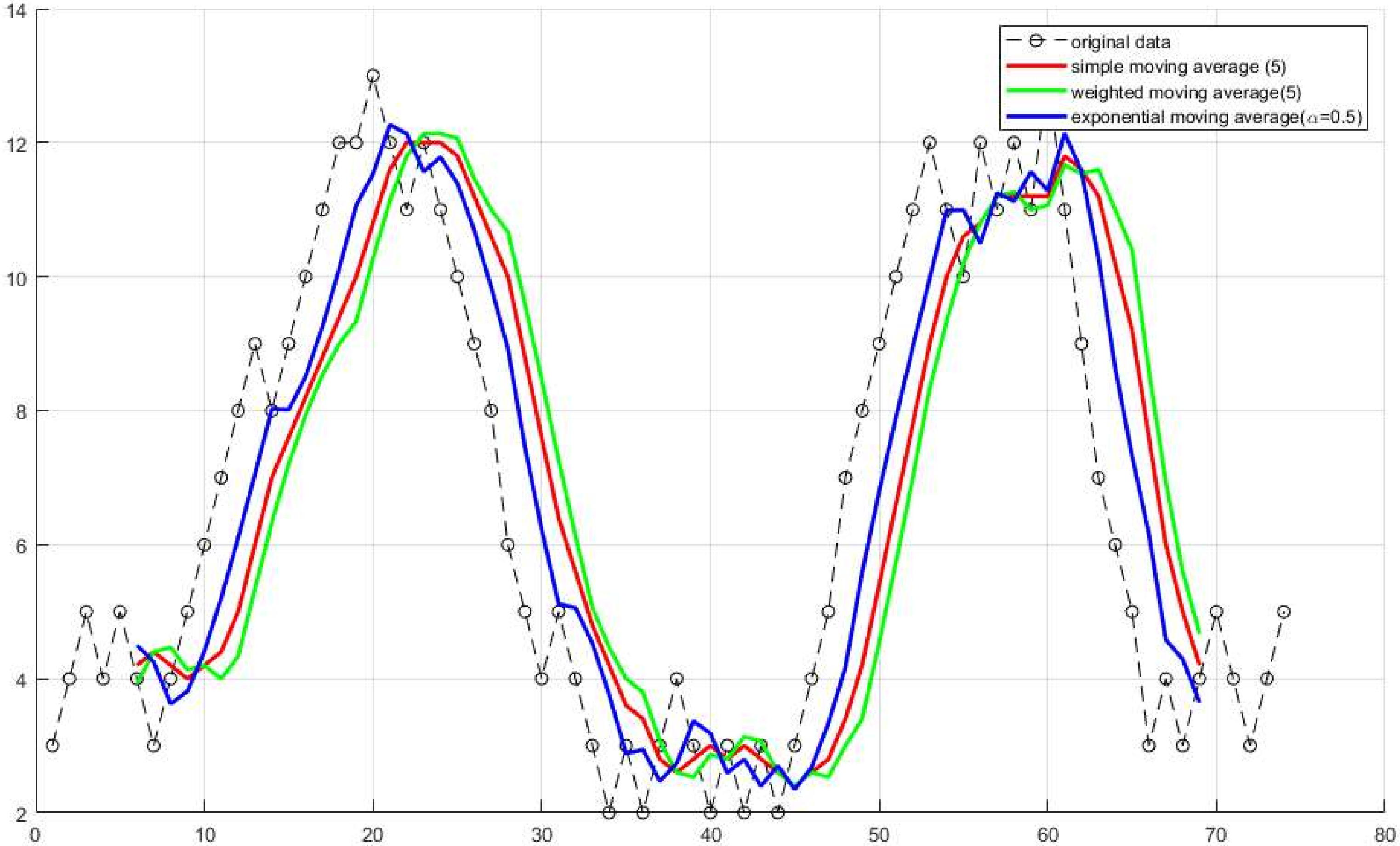


CPU Burst Prediction

- Moving average
  - Exponential Moving Average
  - $EMA_2 = P_1$
  - $EMA_3 = 0.5P_1 + 0.5P_2$
  - $EMA_4 = 0.5^2P_1 + 0.5^2P_2 + 0.5P_3$
  - $EMA_5 = 0.5^3P_1 + 0.5^3P_2 + 0.5^2P_3 + 0.5P_4$
  - $EMA_6 = 0.5^4P_1 + 0.5^4P_2 + 0.5^3P_3 + 0.5^2P_4 + 0.5P_5$
  - In the same equation,
  - $EMA_3 = 0.5EMA_2 + 0.5P_2$
  - $EMA_4 = 0.5EMA_3 + 0.5P_3$
  - $EMA_5 = 0.5EMA_4 + 0.5P_4$

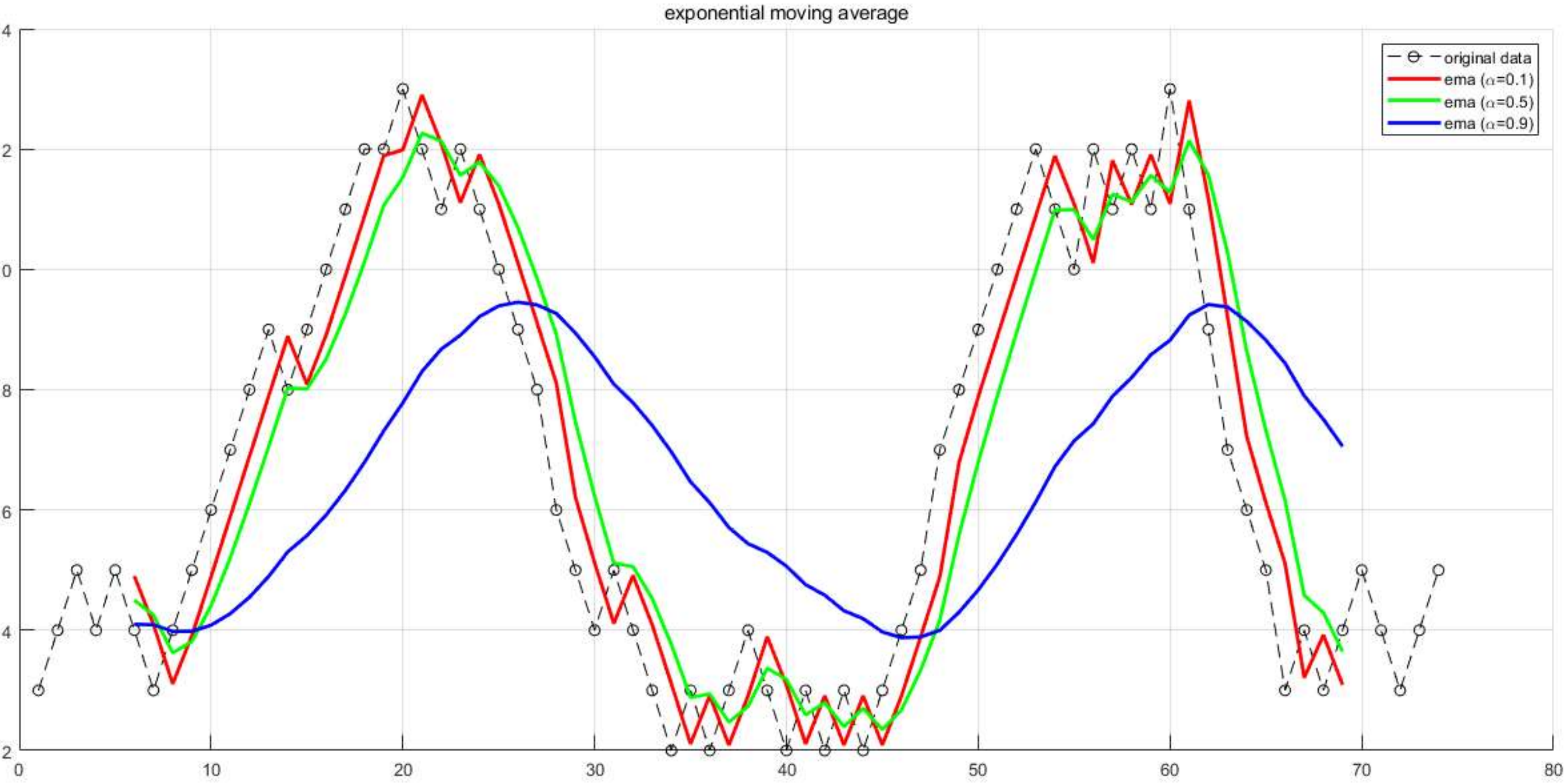
CPU Burst Prediction

- Moving average (SMA, WMA, EMA)



CPU Burst Prediction

- Moving average (EMA with various alpha)


exponential moving average
original data
ema (α=0.1)
ema (α=0.5)
ema (α=0.9)