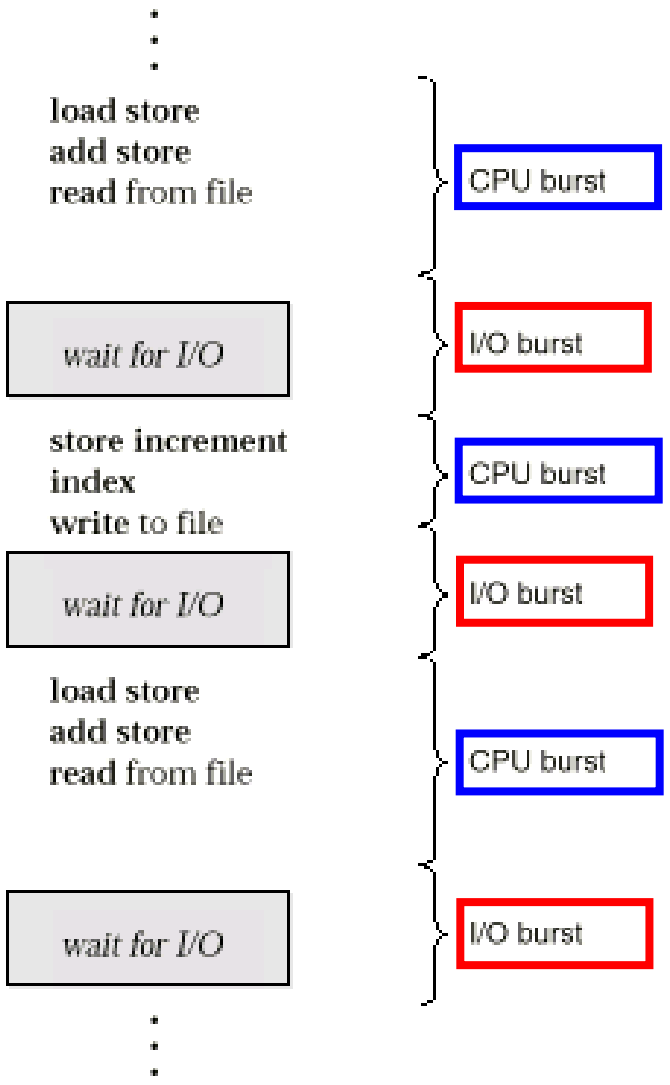
CPU Scheduling

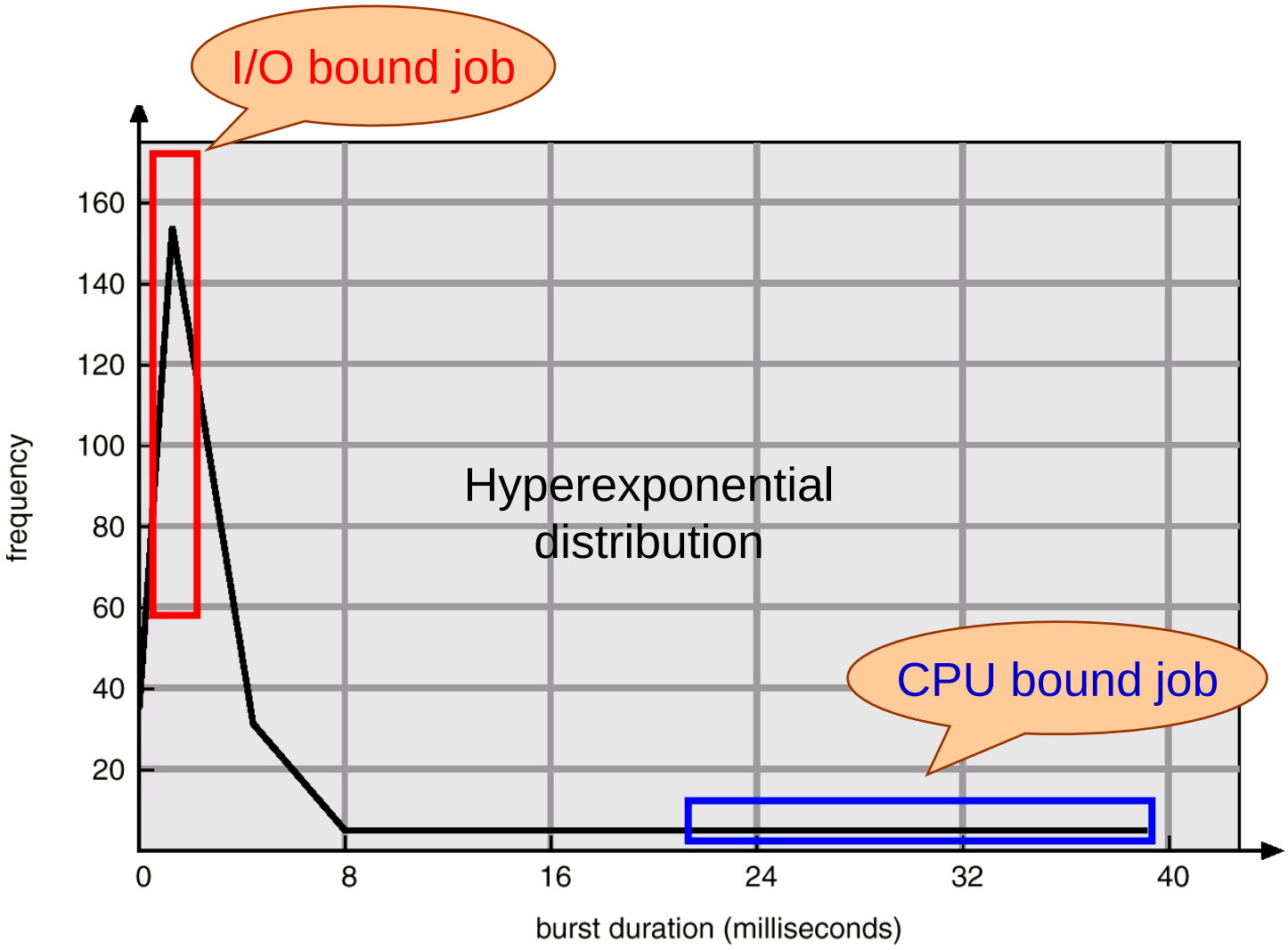
Dept. of Computer Science

Hanyang University

Alternating sequence of CPU and I/O Bursts



Histogram of CPU-burst Times



State Transition Diagram - revisited

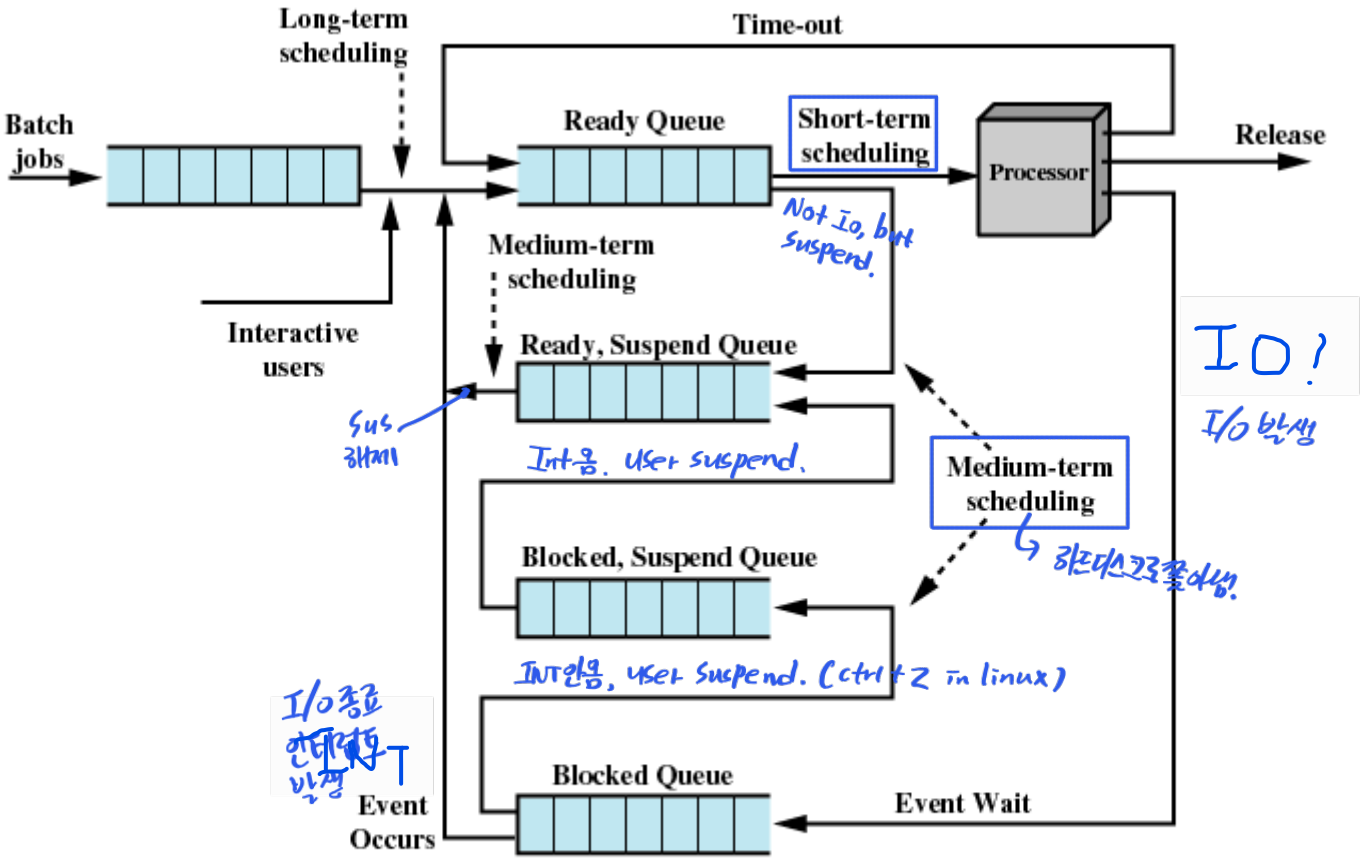


Figure 9.3 Queuing Diagram for Scheduling

CPU Scheduler

- Multiprogramming environment
- ***CPU Scheduler*** *selects* 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. (eg I/O request)
  2. Switches from running to ready state. (eg timerunout)
  3. Switches from waiting to ready. (eg I/O finished interrupt)
  4. Terminates
- If scheduling takes place only under 1 and 4 :
  - The scheduling scheme is *nonpreemptive*
- Otherwise, *preemptive* scheduling scheme

Dispatcher

Ready 에 있는 Process중 하나를 고르면, 실제 CPU를 넘기는 일을 함

kernel code

- ***Dispatcher module*** gives *control of the CPU* to the process selected by the *short-term scheduler*; this involves:
  - *switching context* — context switching 할 때의 오버헤드 (overhead)
  - *switching to user mode* — mode switching
  - *jumping to the proper location in the user program to restart that program*
- ***Dispatch latency***
  - time it takes for the dispatcher to stop one process and start another running. (Mostly, context switch overhead)

Scheduling Criteria (Performance Index) "성능지표"

- ***CPU utilization*** : maximize
  - keep the *CPU as busy as possible*
- ***Throughput*** : maximize
  - *# of processes* that *complete* their execution per time unit
- ***Turnaround time*** : minimize
  - amount of time to *complete a particular process*
- ***Waiting time*** : minimize
  - sum of the periods spent *waiting in the ready queue*
- ***Response time*** : minimize Request (user Action) 후 response까지
  - amount of time it takes *from when a request was submitted until the first response is produced*, **not** output (for time-sharing environment)

Scheduling Algorithms

- FCFS (First-Come First-Served)
- SJF (Shortest-Job-First)
- SRTF (Shortest-Remaining-Time-First)
- Priority Scheduling
- RR (Round Robin)
- Multilevel Queue
- Multilevel Feedback Queue

FCFS Scheduling

- Example:

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 (FCFS is **nonpreemptive**):

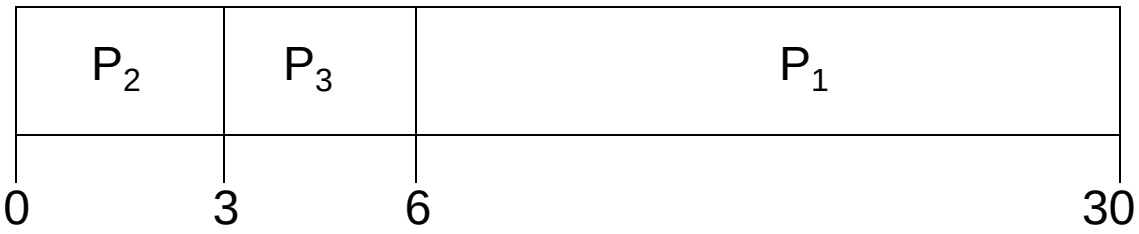
- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

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. waiting time이 들쭉날쭉...
- ***Convoy effect*** : short process behind long process
  호위병

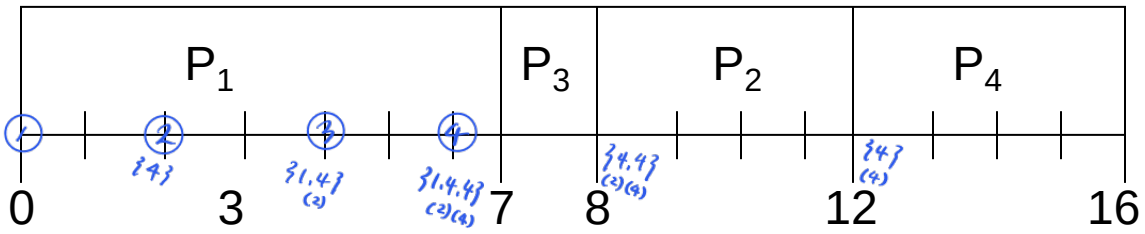
Shortest-Job-First (SJF) Scheduling

- Associate with each process *the length of its next CPU burst*
  - Use these lengths to schedule the process with the shortest time
- Two schemes:
  - ***Nonpreemptive***
    - Once CPU given to the process it cannot be preempted until completes its CPU burst
  - ***Preemptive***
    - 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)
- SJF is optimal
  - Gives *minimum average waiting time* for a given set of processes

Example of Non-Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (non-preemptive)



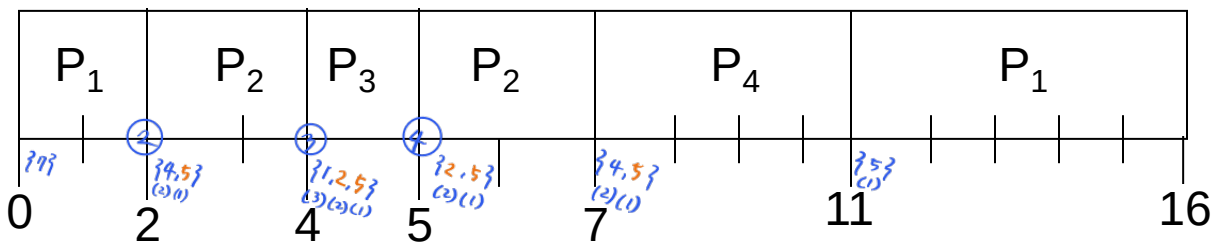
- Average waiting time = (0 + 6 + 3 + 7)/4 = 4

$P_1 = 0 \quad P_2 = 8-2=6 \quad P_3 = 7-4=3 \quad P_4 = 12-5=7$

Example of Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (preemptive) : SRTF "most optimal in waiting time"



- Average waiting time = (9 + 1 + 0 +2)/4 = 3

$(0+9) + (0+1) + (0) + (0+2)$
$P_1 \quad P_2 \quad P_3 \quad P_4$

Determining Length of Next CPU Burst

- But how do you know the *length of next CPU burst?*
  얼마나 걸릴지 어케암?
  (input data, branch, user …)
- Can only *estimate* the length
  예측해.
- Can be done by using the length of *previous* CPU bursts, using exponential averaging.

1. t_n = actual lenght of n^{th} CPU burst
   n번째로 CPU를 잡았을때 burst의 길이 (과거data)
2. τ_{n+1} = predicted value for the next CPU burst
   예측값.
3. $\alpha, 0 \le \alpha \le 1$
   가중치.
4. Define : $\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n$
   그래 얘도 모르겠어?

Exponential averaging
t_n: 직전값
τ_n: 그 전의 예측값

Examples of Exponential Averaging

- $\alpha = 0$
  - $\tau_{n+1} = \tau_n$ 예측값은 항상 동일.
  - Recent history does not count.
- $\alpha = 1$
  - $\tau_{n+1} = t_n$
  - Only the actual last CPU burst counts.
- If we expand the formula, we get:

$$\tau_{n+1} = \alpha\, t_n + (1-\alpha)\,\alpha\, t_{n-1} + \ldots$$
$$+ (1-\alpha)^j \alpha\, t_{n-j} + \ldots$$
$$+ (1-\alpha)^{n+1} \tau_0 = t_0$$

최근 데이터 중요성↑ 과거 데이터 중요성↓

α가 0과 1사이라 앞 계수는 점점 작아짐

- Since both α and $(1-\alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor

Examples of Exponential Averaging

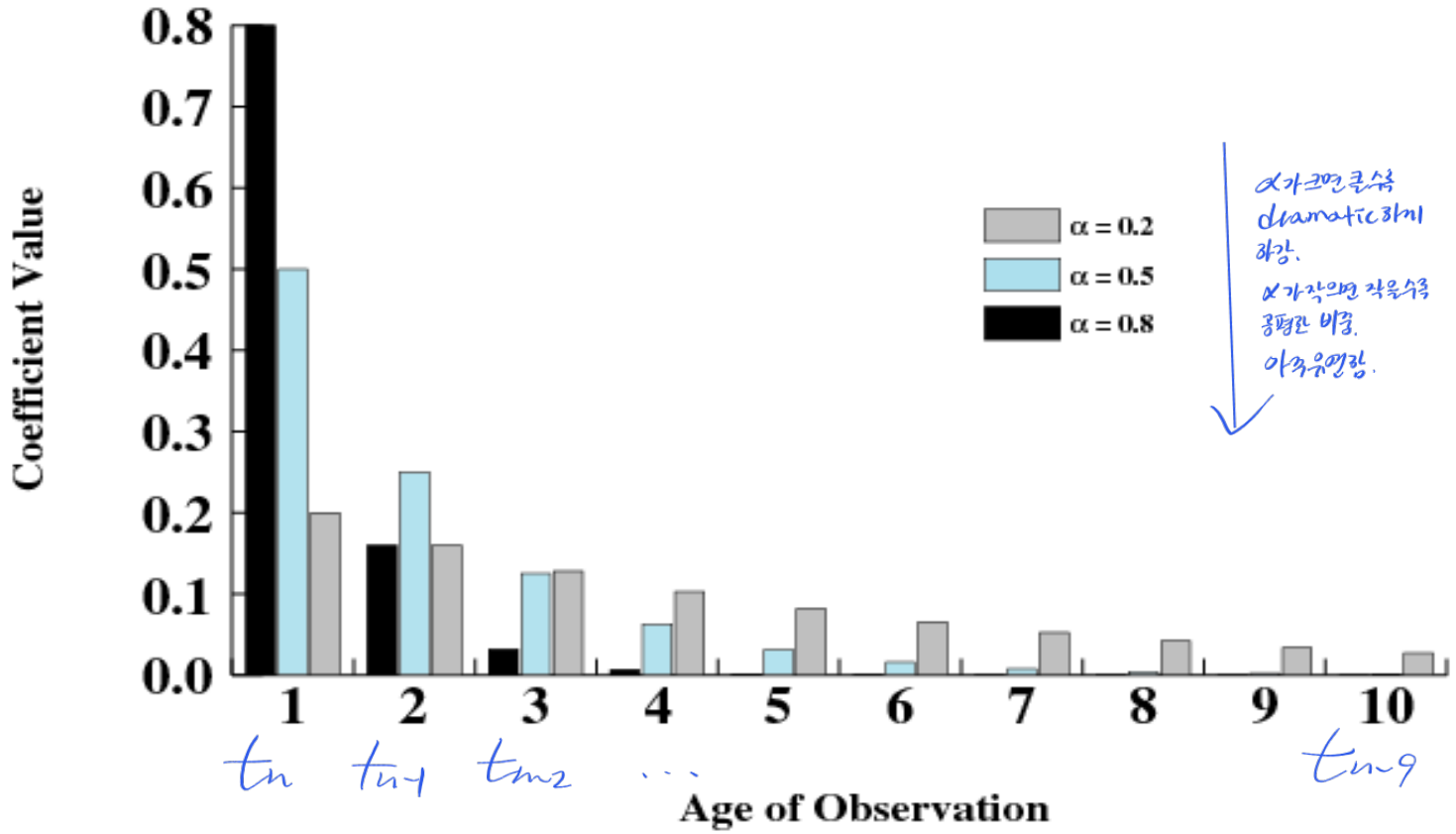
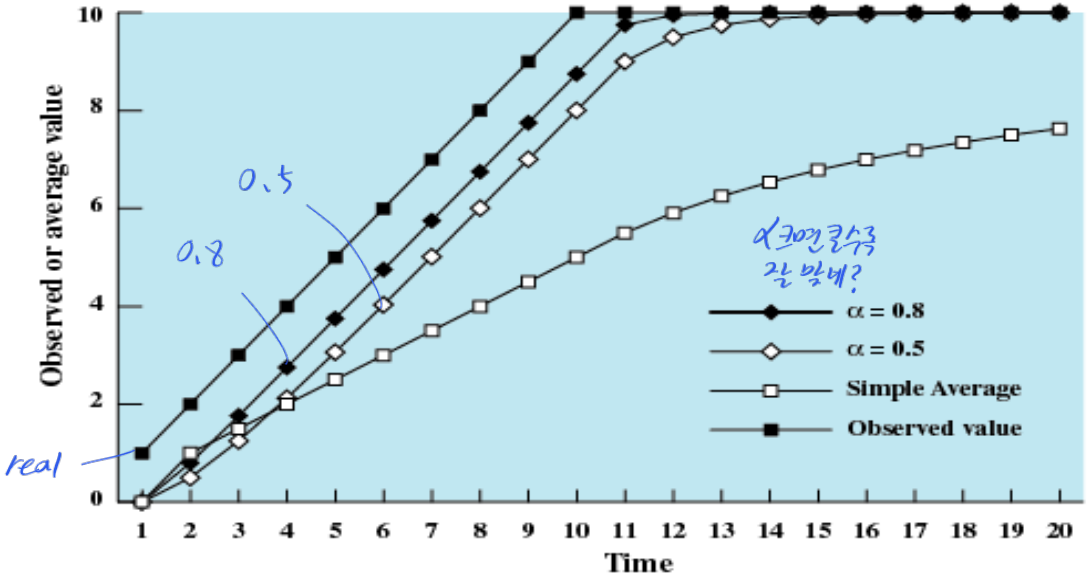
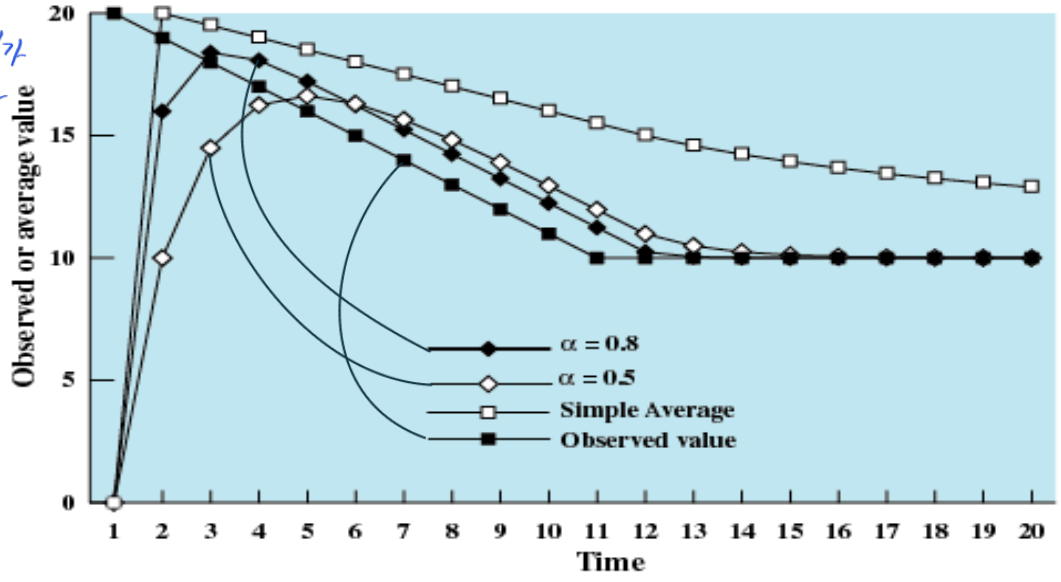


Figure 9.8 Exponential Smoothing Coefficients



(a) Increasing function



(b) Decreasing function

Figure 9.9 Use of Exponential Averaging

Priority Scheduling

Not algorithm. but 정책.

- 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*
  - *nonpreemptive*
- *SJF* is a priority scheduling where

  priority is the predicted next CPU burst time
- Problem
  - ***Starvation***: low priority processes may *never execute*
- Solution
  - ***Aging***: as time progresses *increase the priority* of the process

Round Robin (RR)

"CPU를 갖을수 있는 시간의 한계치"

- Each process gets a small unit of CPU time (***time quantum***)
  - Usually 10-100 milliseconds
  - After this time has elapsed, the process is preempted and added to the end of the ready queue
- If there are *n* processes in the ready queue and the time quantum is ***q sec***, then each process gets 1/*n* of the CPU time in chunks of at most *q* sec at once

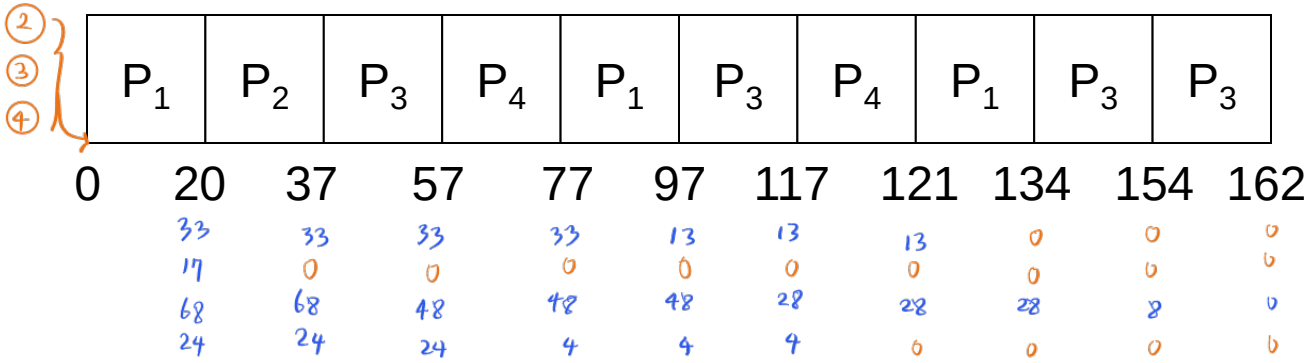
  "upper bound"

  ⇒ *No process waits more than (n-1)q time units*
- Performance
  - *q* large ⇒ FIFO
  - *q* small ⇒ *q* must be large with respect to context switch, otherwise overhead is too high

Example: RR with Time Quantum = 20

Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

- The Gantt chart is:



- Typically, higher average turnaround than SJF, but better *response*

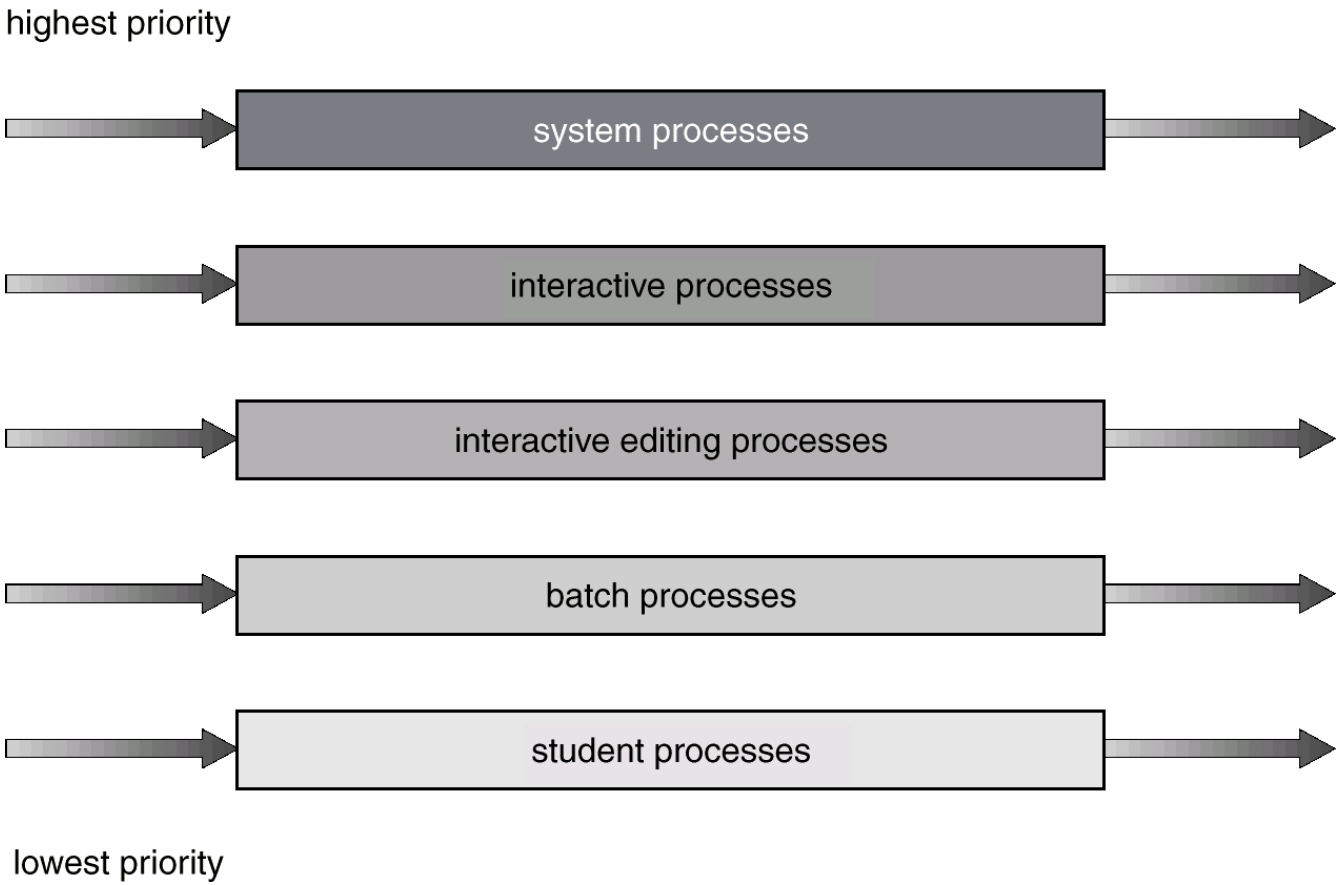
process가 시작하고 끝나는 시간.

Multilevel Queue

- Ready queue is partitioned into separate queues:
  - ***foreground*** ***(interactive)***
  - ***background*** ***(batch – no human interaction)***
- Each queue has its own scheduling algorithm
  - ***foreground*** – **RR**
  - ***background*** – **FCFS**
- Scheduling must be done between the queues.
  - **Fixed priority scheduling**
    - 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
    - Eg., *80% to foreground in RR*, *20% to background in FCFS*

Multilevel Queue Scheduling

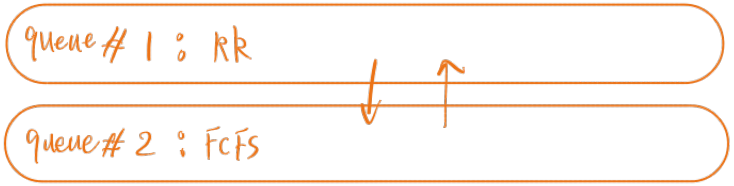
Ready queue example : queue 가 5개.



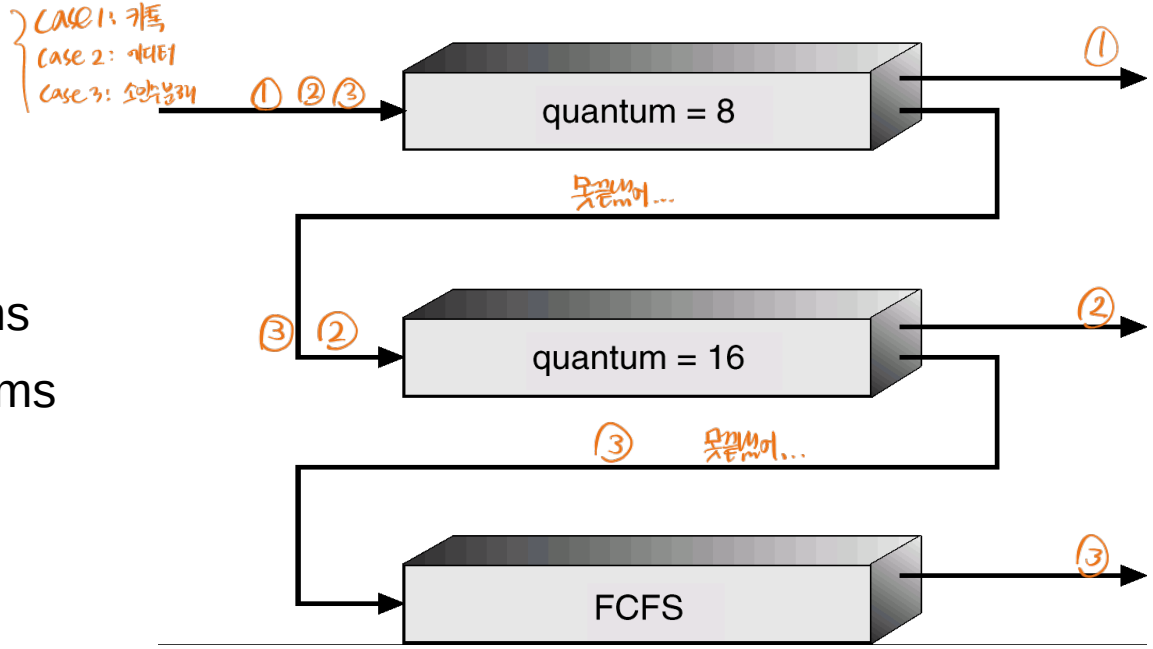
Multilevel Feedback Queue

어떻게 프로세스를 구별해서 queue에 넣나? → 프로세스가 속한 queue를 이동할수 있게해야 practical함.

- A process *can move* between the *various queues*
- 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 – time quantum 8 ms
  - Q_1 – time quantum 16 ms
  - 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 ms
  - If it still does not complete, it is preempted and moved to queue Q_2

Real-Time Scheduling

실시간 스케줄링 이야기

- Hard real-time systems
  - Required to *complete a critical task* within a guaranteed amount of time
    - deadline이 존재. 이를 보장.
- Soft real-time computing
  - Requires that critical processes receive priority over less fortunate ones
    - 일반적 운영체제에 포함되는 개념.