Multicore Performance #1
- CPU Scheduling -

Minsoo Ryu

Department of Computer Science and Engineering
Hanyang University

Outline

CPU Scheduling

- CPU scheduling is to decide when and which process to run among ready processes

- Scheduling criteria
  - Performance objectives
    - Maximize CPU utilization or throughput
    - Minimize completion time, waiting time, or response time
  - Real-time constraints
    - Satisfy deadlines
  - Fairness
    - Provide CPU cycles proportional to weights

Classification of Scheduling Policies

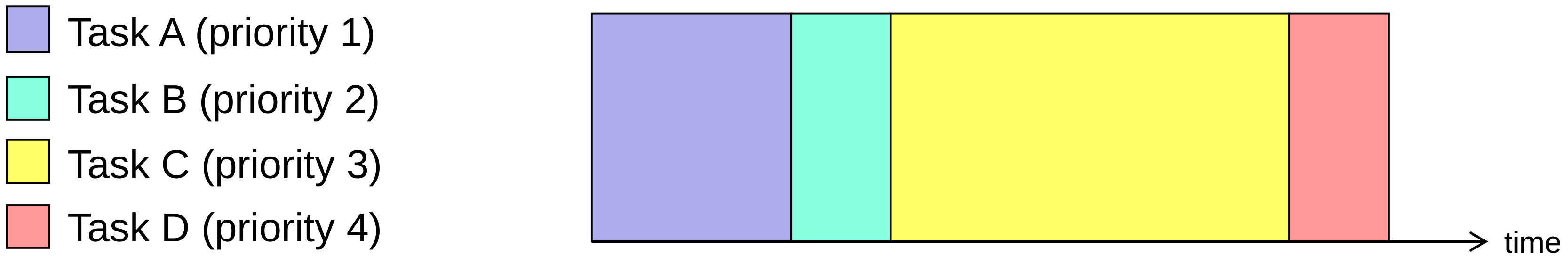
- Two paradigms
  - Priority-based scheduling
  - Proportional share scheduling

- Number of processors
  - Uniprocessor scheduling
  - Multiprocessor scheduling

Priority-Based Scheduling

Priority-Based Scheduling

- A priority number (integer) is associated with each process



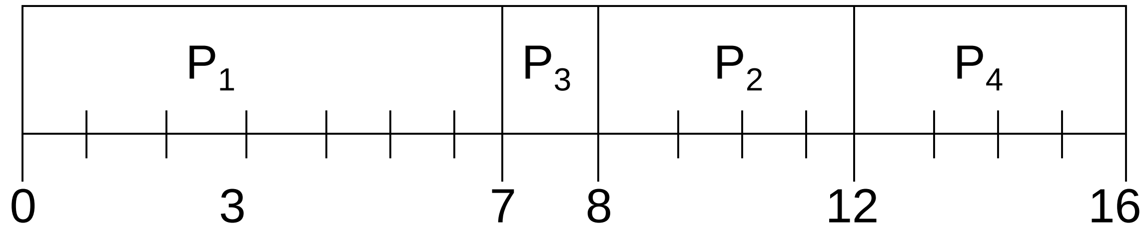
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority)
  - Preemptive
  - Nonpreemptive

Priority-Based Scheduling Policies

- Non-real-time policies
  - FCFS (First-Come, First-Served)
  - SJF (Shortest-Job-First)
  - SRTF (Shortest-Remaining-Time-First)
- Real-time policies
  - RM (Rate Monotonic)
  - EDF (Earliest Deadline First)

SJF (Shortest-Job-First)

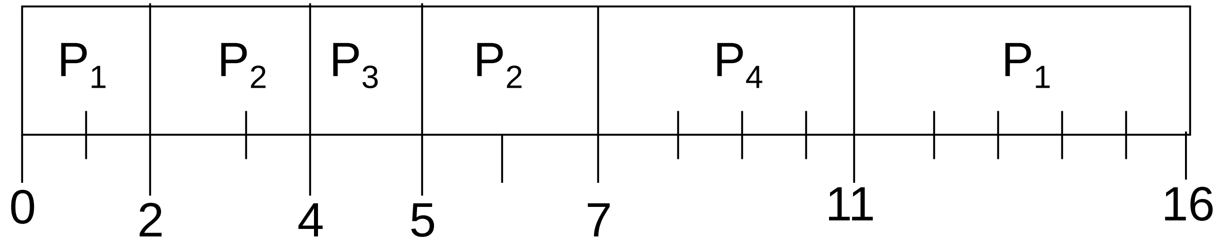
Process	Arrival Time	Execution Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4



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

SRTF (Shortest-Remaining-Time-First)

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



- Average waiting time = (9 + 1 + 0 +2)/4 = 3
  - SRTF is optimal in that it achieves minimum average waiting time for a given set of processes

RM (Rate Monotonic)

빈도가 높은 일에 우선순위를 주는 방식.

Assumptions

- Processes have periods, worst-case execution times (WCETs), and deadlines

Scheduling policy

- Give higher priorities to tasks with shorter periods
- Preemptive static priority scheduling

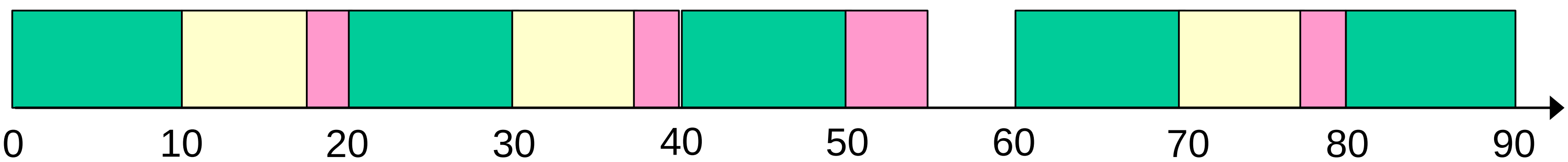
Optimality

- If a feasible static priority assignment exists for some process set, the RM priority assignment is feasible for that process set

RM (Rate Monotonic)

- **Consider the following tasks**
  - **Process X: period = 20, WCET = 10, deadline = 20**
  - **Process Y: period = 30, WCET = 8, deadline = 30**
  - **Process Z: period = 40, WCET = 4, deadline = 40**

period : 주기



- **Schedulability test**
  - **CPU utilization:** $U = \sum_{i=1}^{m} e_i / p_i$
  - **A set of m processes is schedulable if** $U \le m(2^{1/m} - 1)$
    - For large m, $m(2^{1/m} - 1) \approx \ln 2 \approx 0.69$

m : CPU count
U : CPU utilization

cpu utilization이 69%를 넘지 않으면 rate monotonic을 할 수 있다.

EDF (Earliest Deadline First)

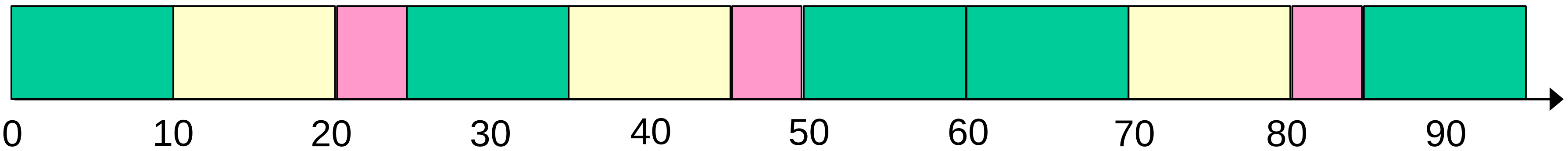
- Scheduling policy
  - Give higher priorities to tasks with earlier absolute deadlines
  - Preemptive dynamic priority scheduling

- Optimality
  - If a feasible dynamic priority assignment exists for some process set, the EDF priority assignment is feasible for that process set

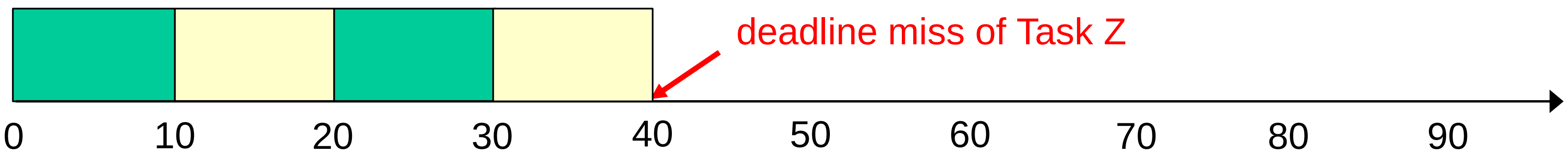
- Schedulability test
  - A set of m processes is schedulable if and only if $U \le 1$

EDF (Earliest Deadline First)

- Consider the following tasks
  - Process X: period = 20, WCET = 10, deadline = 20
  - Process Y: period = 30, WCET = 10, deadline = 30
  - Process Z: period = 40, WCET = 5, deadline = 40



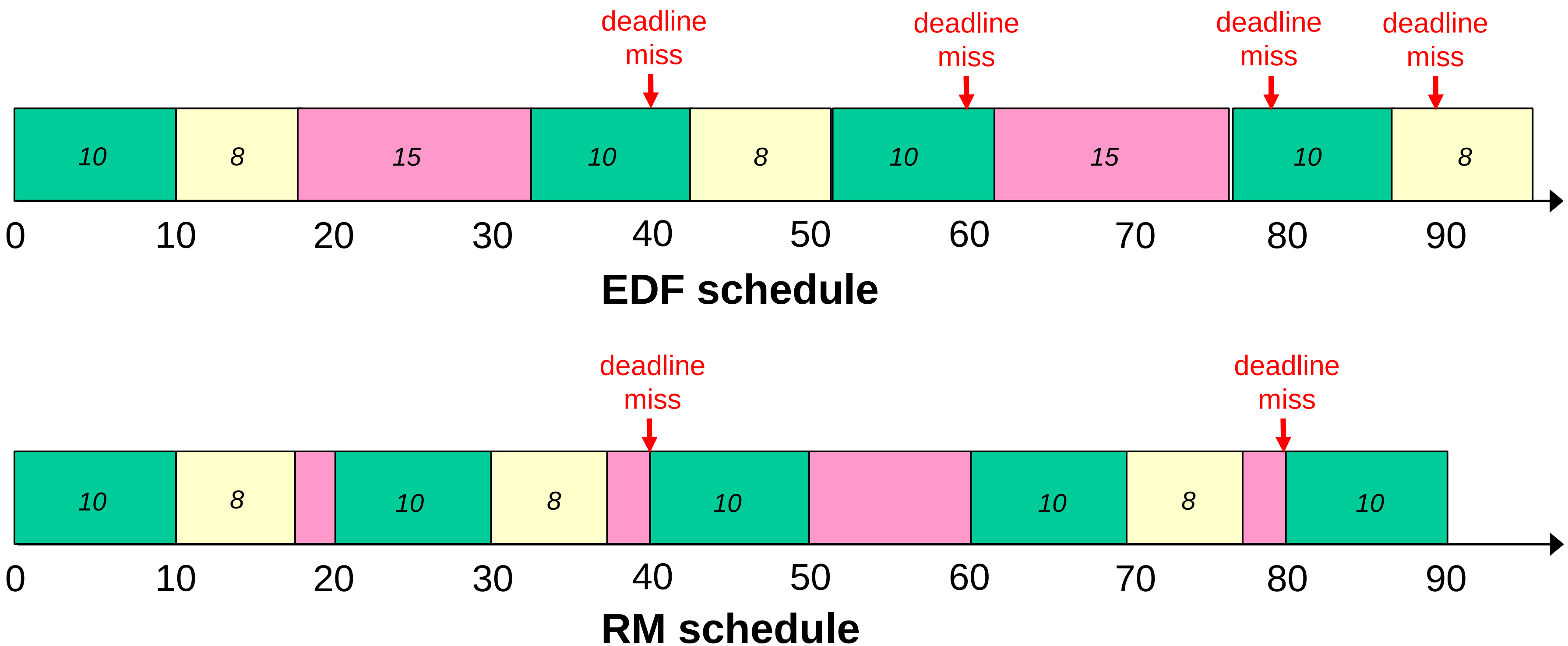
EDF schedule



RM schedule

Non-schedulable Behavior

- Consider the following tasks
  - Process X: period = 20, WCET = 10, deadline = 20
  - Process Y: period = 30, WCET = 8, deadline = 30
  - Process Z: period = 40, WCET = 15, deadline = 40

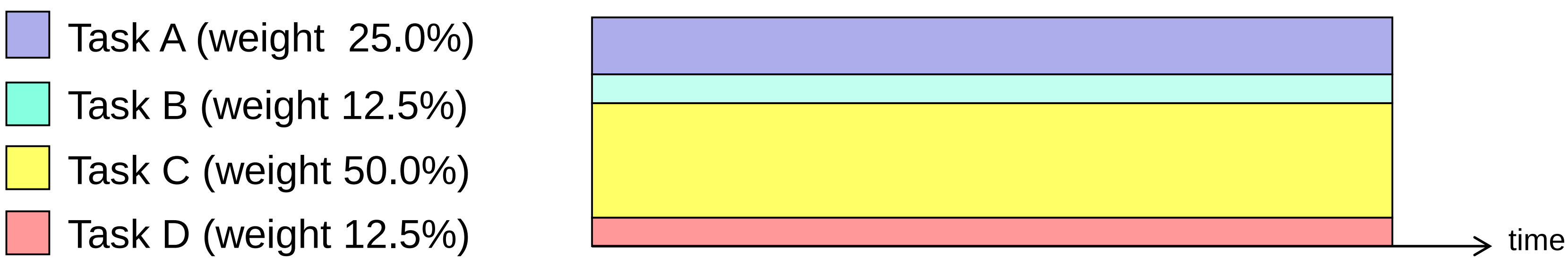


Proportional Share Scheduling

Proportional Share Scheduling

➢ Basic concept

- **A weight value is associated with each process**
- **The CPU is allocated to the process in proportion to its weight**



➢ Two contexts

- **Fair queueing (in the context of communication networks)**
  - Packet scheduling
- **Proportional share (in the context of operating systems)**
  - Process scheduling

Scheduling Algorithms

- Network scheduling
  - PGPS (= WFQ), Demers *et al.*, 1989
  - Virtual Clock, Lixia Zhang, 1990
  - SCFQ, Golestani, 1994
  - SFQ, Goyal *et al.*, 1996
  - WF^2Q, Bennett *et al.*, 1996
- CPU scheduling
  - Lottery and Stride, Waldspurger, 1995
  - Hierarchical SFQ, Goyal *et al.*, 1996
  - BVT, Duda *et al.*, 1999
  - VTRR, Nieh *et al.*, 2001

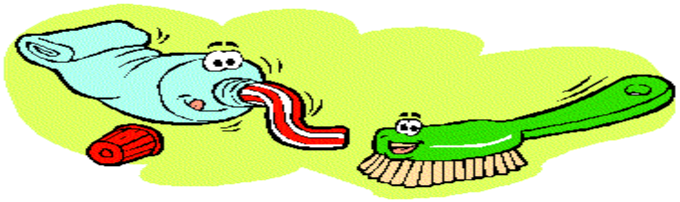
GPS (Generalized Processor Sharing)

- A GPS server is defined by (Kleinrock, 1976)

$$\frac{W_i(t_1,t_2)}{W_j(t_1,t_2)} \geq \frac{r_i}{r_j},\ j=1,2,\ldots,N$$

  - $W_i(t_1,t_2)$: the amount of session i traffic served in an $(t_1,t_2]$ interval
  - r_i : weight of session i

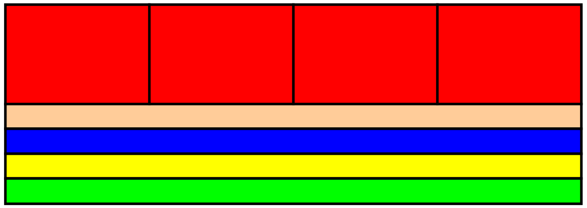
- Packets of all sessions are served simultaneously
  - Idealized fluid-flow system or bit-by-bit weighted round-robin



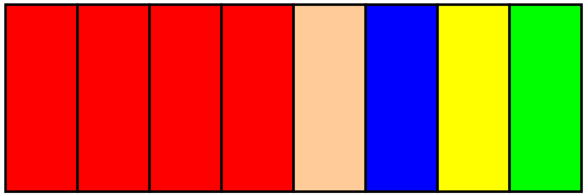
PGPS (Packet-by-Packet GPS)

- PGPS is an approximation to GPS
  - Fair queueing on a packet-by-packet basis
  - Also known as WFQ (Weighted Fair Queueing)

- Algorithm
  - F_p : the time at which packet p will depart (finish service) under GPS
  - Serve packets in increasing order of F_p



GPS



PGPS

Fairness Bounds of PGPS

Bound on lag

- $\hat{F}_p - F_p \leq \dfrac{L_{\max}}{r}$
  - $\hat{F}_p$: the time at which packet p departs under PGPS
  - F_p : the time at which packet p will depart under GPS
  - $L_{\max}$: the maximum packet length
  - If $r = 1Gbps$ and $L_{\max} = 1Kb$, then the lag is 1 ms

Bound on difference of services received

- $W_i(0,\tau) - \hat{W}_i(0,\tau) \leq L_{\max}$

Worst-case Fair WFQ

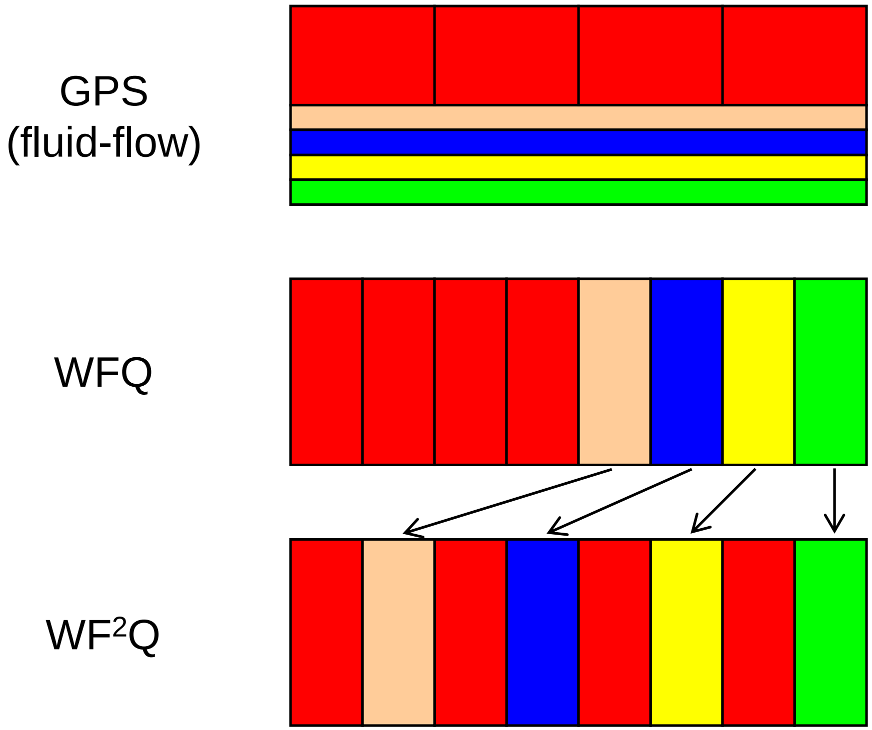
- WF²Q only considers
  - Packets that have started receiving service under GPS

WFQ satisfies only

$$W_p(t_1,t_2) - \hat{W}_p(t_1,t_2) \le L_{\max}$$

not

$$|W_p(t_1,t_2) - \hat{W}_p(t_1,t_2)| \le L_{\max}$$



Lottery and Stride Scheduling

- Tickets: relative resource rights
  - Task τ_i has m_i tickets
  - A total of M tickets
- Lottery scheduling: probabilistic algorithm
  - Use random number generator to select a winning ticket
  - Task τ_i is probabilistically guaranteed a rate of $p = m_i / M$
- Stride scheduling: deterministic algorithm
  - "Stride" is inversely proportional to tickets
  - Task with minimum "pass" value is selected and its pass is advanced by its stride

Stride Scheduling

- Task τ_1 : tickets = 3, stride = 2
- Task τ_2 : tickets = 2, stride = 3
- Task τ_3 : tickets = 1, stride = 6

Initial pass values are set to stride values

pass \ time	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
τ_1's pass value	2	4	4	6	8	8	8	10	10	12	14	14	14	16	16
τ_2's pass value	3	3	6	6	6	9	9	9	12	12	12	15	15	15	18
τ_3's pass value	6	6	6	6	6	6	12	12	12	12	12	12	18	18	18

Smallest value is chosen
Pass is advanced by stride 3

Ties are broken arbitrarily

Multiprocessor Scheduling

Uniprocessor vs. Multiprocessor Scheduling

- **Uniprocessor scheduling**
  - **It is to decide when and which job will run**

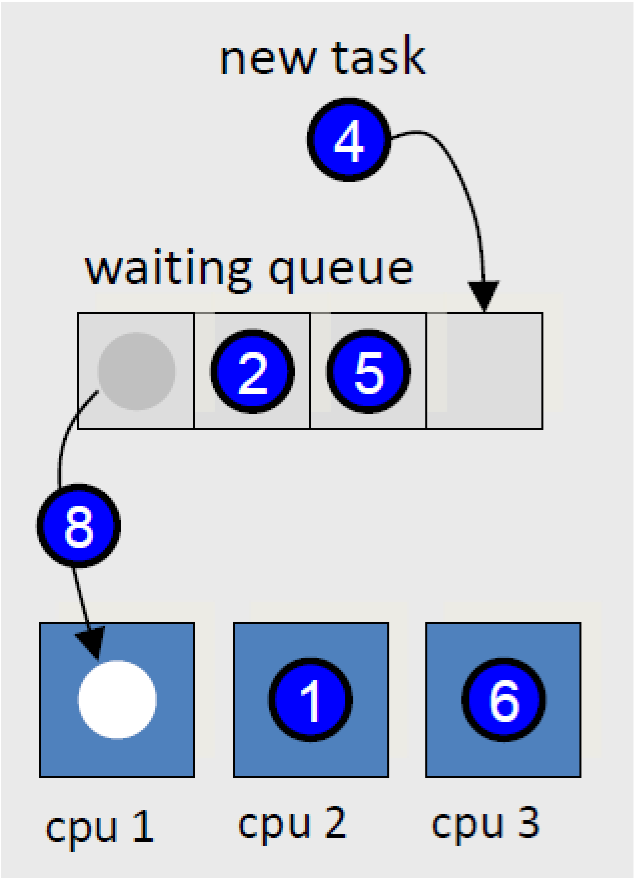
- **Multiprocessor scheduling**
  - **It is to decide not only when but also where a job will run**
  - **Almost the same goals as those of uniprocessor scheduling**
  - **But it raises new issues**
    - How to assign applications to multiple processors?
    - How to balance workload among processors?
    - How to define and exploit affinity?
    - How to manage processor heterogeneity?

Multiprocessor Scheduling Policies

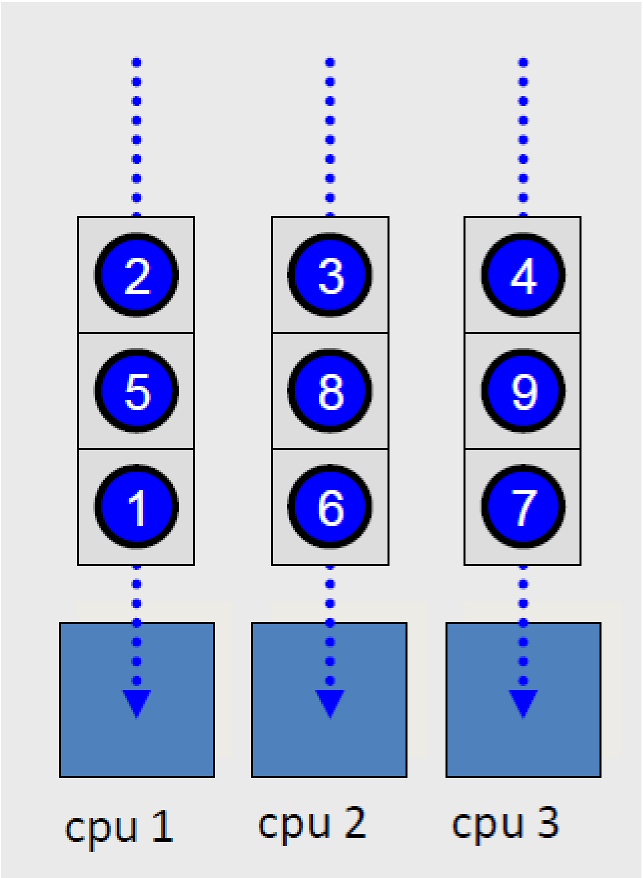
- **The same policies as uniprocessor policies**
  - **Priority-based scheduling: FCFS, SJF, SRTF, RM, EDF**
  - **Proportional share scheduling: PGPS, SFQ, WF^{2}Q, Lottery and Stride, BVT, VTRR**

- **Two approaches**
  - **Global scheduling**
    - The system has a single global process queue
    - Processes are dispatched to any available processors
  - **Partitioned scheduling**
    - Each processor has a separate process queue
    - Each queue is scheduled by an independent scheduler
    - Process migration may be allowed or not

Global vs. Partitioned Scheduling



Global Scheduling



Partitioned Scheduling

Global vs. Partitioned Scheduling

Global scheduling

migration issue, cache problem

- It is generally believed that global scheduling can achieve better performance
- However, it can be inefficient due to the contention at the single queue and increased cache misses

Partitioned scheduling

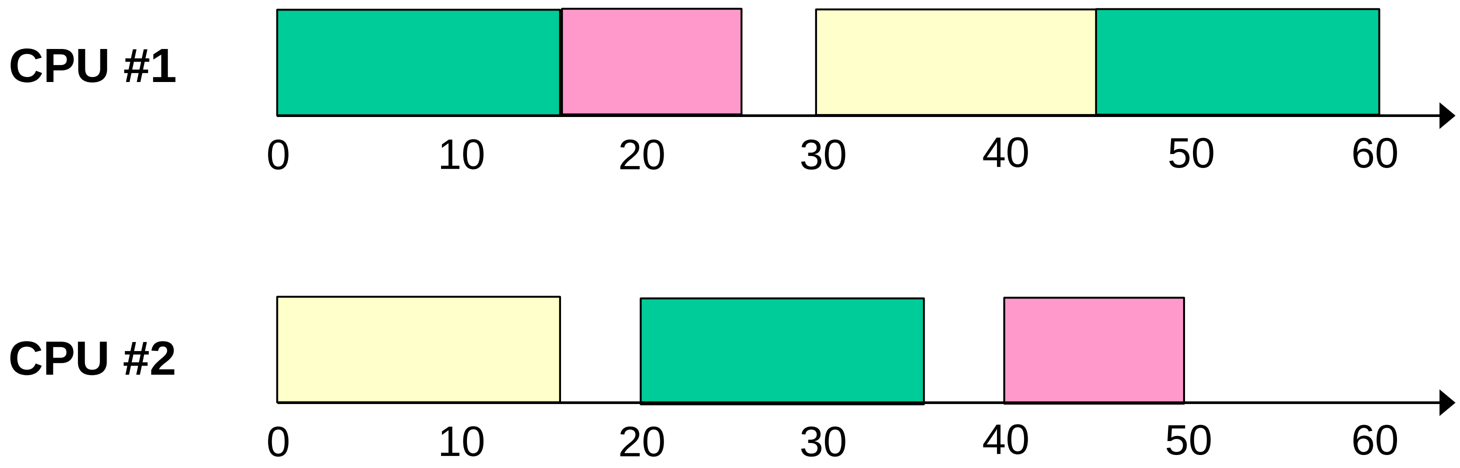
실제로는 이 방식을 많이 사용한다. 그 이유는, RM scheduling 방식은 single core에서 증명된 스케줄링 방식이기 때문이다.

- Performance can vary depending on the initial distribution of processes, i.e., a bin-packing problem
- Different scheduling policies can be employed across processors
- We can use the rich and extensive results from the uniprocessor scheduling theory

bean-packing problem => NP problem

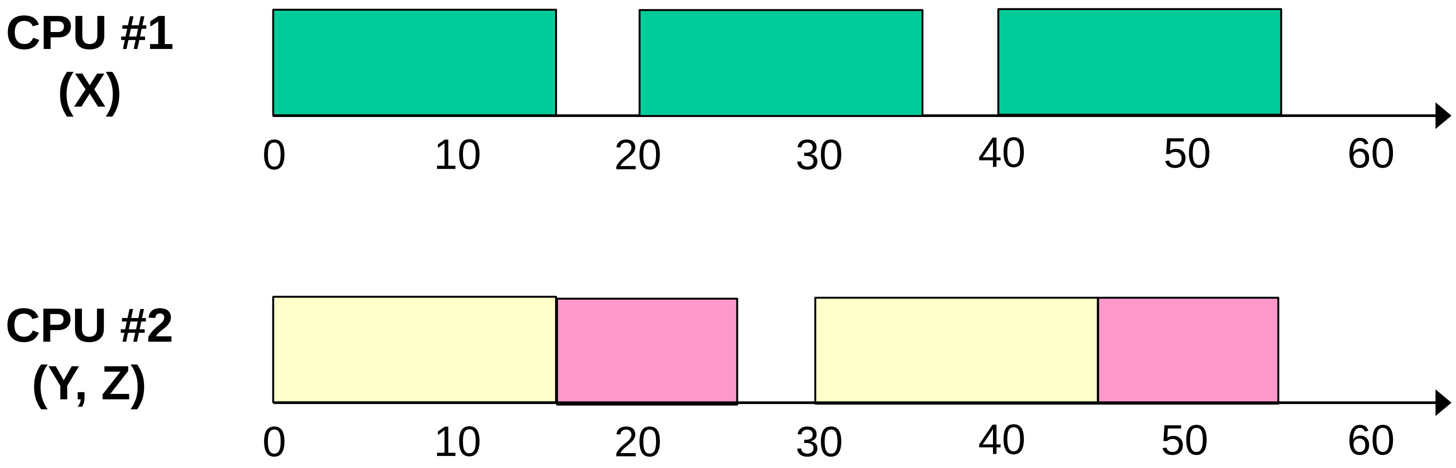
Global EDF

- Consider the following tasks
  - Process X: period = 20, WCET = 15, deadline = 20
  - Process Y: period = 30, WCET = 15, deadline = 30
  - Process Z: period = 40, WCET = 10, deadline = 40



Partitioned EDF

- Consider the following tasks
  - Process X: period = 20, WCET = 15, deadline = 20
  - Process Y: period = 30, WCET = 15, deadline = 30
  - Process Z: period = 40, WCET = 10, deadline = 40



Schedulability Analysis

➢ Global EDF

- There is no single efficient test
- Most tests are very complex

Theorem 2 (*GFB*). *A task set τ is schedulable with global* EDF *if*

$$\lambda_{\text{tot}} \leqslant m(1-\lambda_{\max}) + \lambda_{\max}. \qquad (1)$$

Theorem 3 (*BAK*, from [3]). *A task set τ is schedulable with global* EDF *if, for all $\tau_k \in \tau$, there is a $\lambda \in \{\lambda_k\} \cup \{U_\ell | U_\ell \geqslant \lambda_k, \ell < k\}$ such that*

$$\sum_{\tau_i \in \tau} \min(1, \beta_{i,k}(\lambda)) \leqslant m(1-\lambda) + \lambda, \qquad (2)$$

where

$$\beta_{i,k}(\lambda) = \begin{cases} U_i\left(1 + \frac{\max(0, T_i - D_i)}{D_k}\right) & \text{if} \quad U_i \leqslant \lambda \\ U_i\left(1 + \frac{T_i}{D_k}\right) - \lambda\frac{D_i}{D_k} & \text{if} \quad U_i > \lambda. \end{cases}$$

Theorem 6 (*FFDBF from* [10]). *A task set τ is schedulable with global* EDF *if $\exists \sigma | \lambda_{\max} \leqslant \sigma < \frac{m - U_{\text{tot}}}{m-1} - \epsilon$ (with an arbitrarily small ϵ), such that $\forall t \geqslant 0$,*

$$\mathit{ffdbf}(t, \sigma) \leqslant (m - (m-1)\sigma)t \qquad (10)$$

It can be proved that it is sufficient to check only those values of t in $\{kT_i + D_i | \, k \in \mathbb{N}\}_{i=1}^{n}$ that are smaller than[2]

$$\frac{\sum_{\tau_i \in \tau} C_i\left(1 - \frac{D_i}{T_i}\right)}{m - (m-1)\sigma - U_{\text{tot}}}. \qquad (11)$$

…

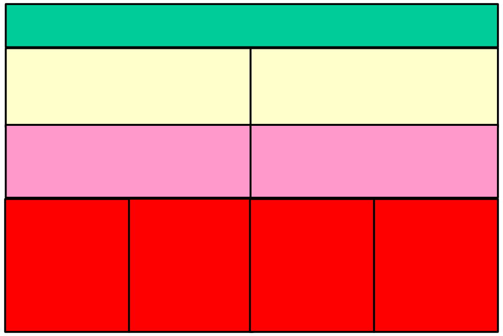
…

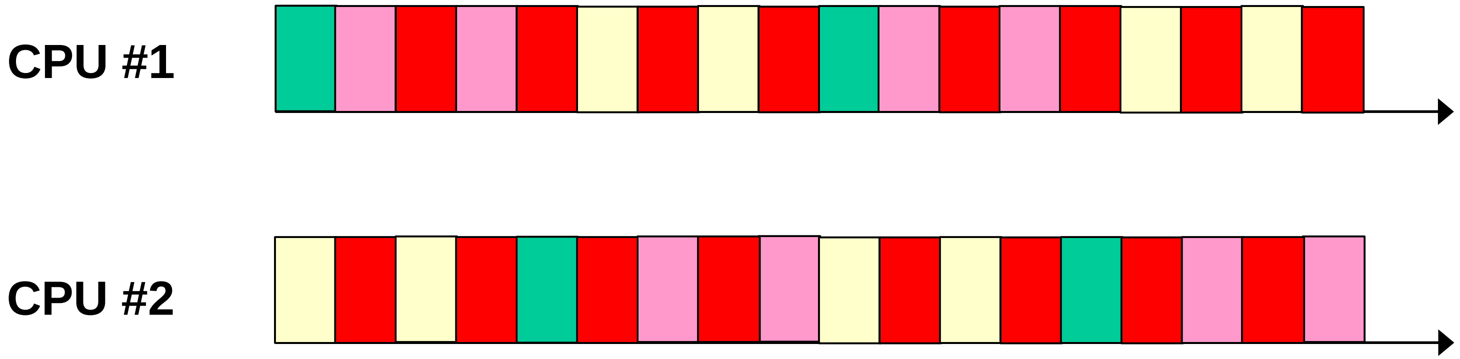
➢ Partitioned EDF

- Sufficient to check if the CPU utilization does not exceed 100% for each processor

Global WFQ

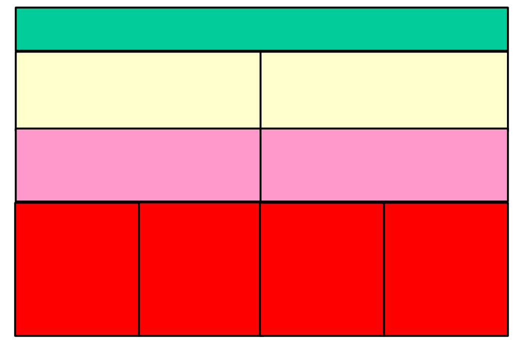
- Consider the following tasks
  - Process A: weight = 1
  - Process B: weight = 2
  - Process C: weight = 2
  - Process D: weight = 4

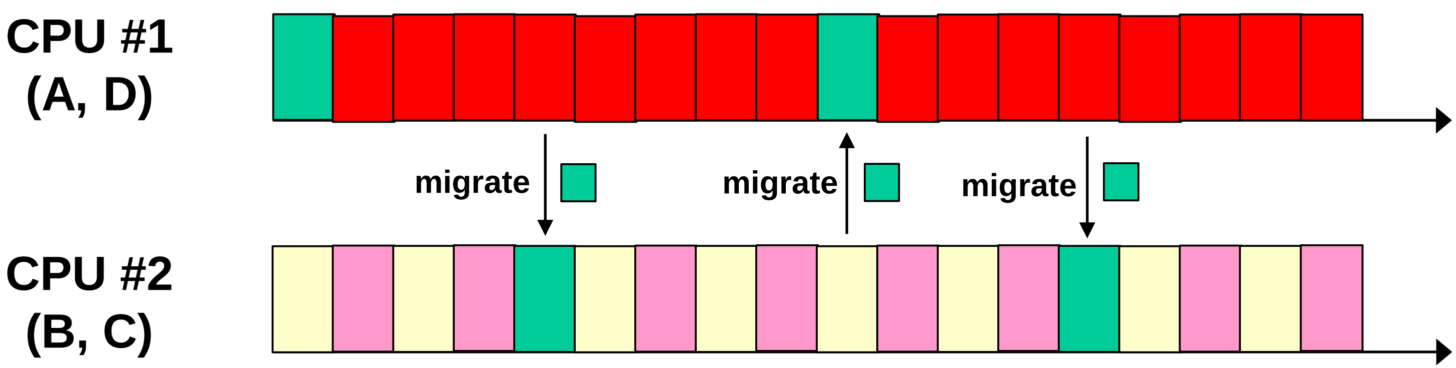




Partitioned WFQ with Load Balancing

- Consider the following tasks
  - Process A ■: weight = 1
  - Process B □: weight = 2
  - Process C □: weight = 2
  - Process D ■: weight = 4






Q&A