# CIS 721 - Real-Time Systems

## Quiz #1 Review

Mitch Neilsen
**neilsen@ksu.edu**

# Outline

- Quiz #1: Fri., Oct. 16, in class
- Topics
  - Real-Time Scheduling Theory and Algorithms
  - Ch. 1-7
- Open book, open notes

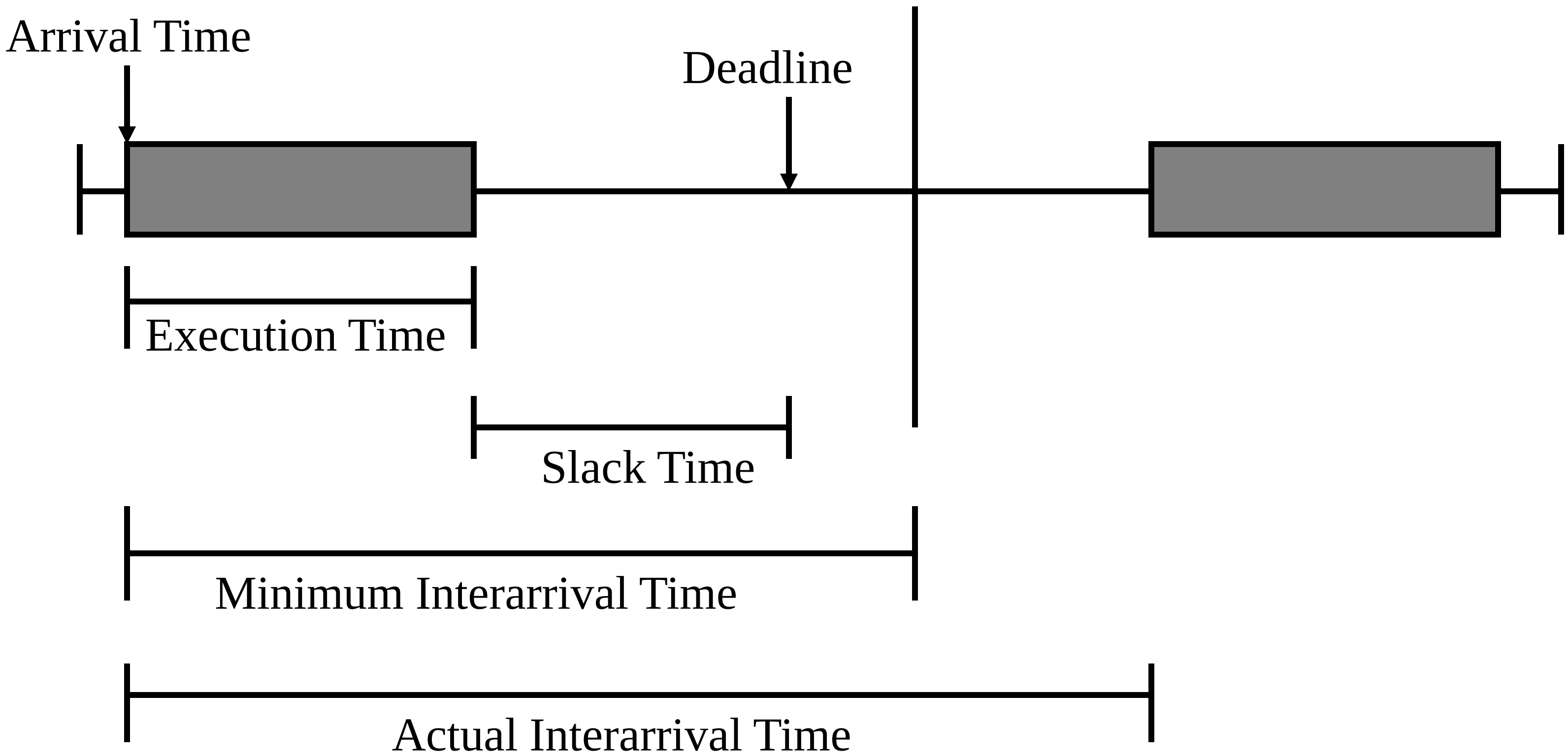
# Terms and Concepts

- A **real-time system** is a system with performance deadlines on computations and actions; that is, *system **correctness** depends on the **timeliness** of the results*.
- An **embedded system** is a system that exists within a larger system.
- A **job** is a unit of work that is scheduled and executed by the system ($J_{i,k}$).
- A **task** is a set of related jobs that provide some system function $\tau_i = \{ J_{i,1}, J_{i,2}, \ldots , J_{i,n} \}$.; e.g., the reception of a data frame could be a job that is part of a task that provides time service.
- The **deadline** of a job is the time at which a job must be completed.

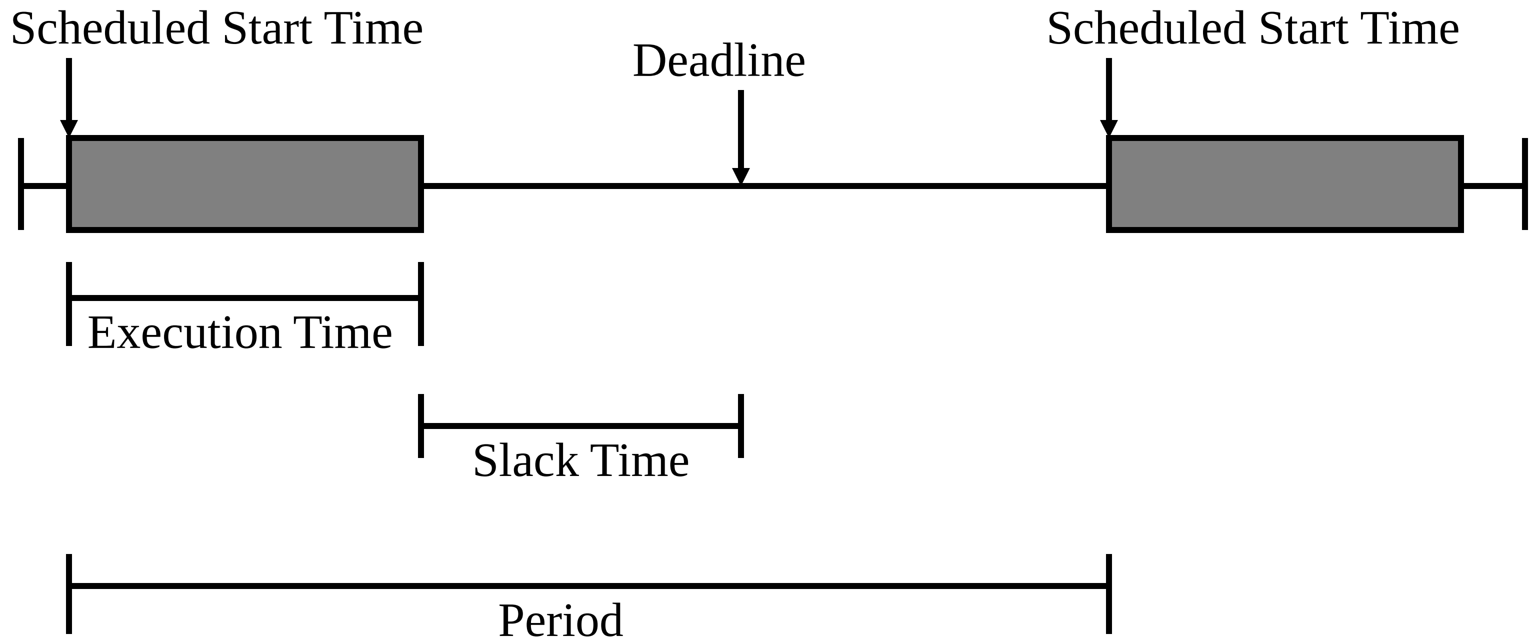
# Deadlines

- The **release time** (or **arrival time**) of a job is the time at which the job becomes available for execution ( $r_i$ or $R_i$ ).
- The **response time** of a job is the length of time between the release time of the job and the time instant when it completes.
- The **relative deadline** of a job is the maximum allowable response time of a job ( $D_i$ ).
- The **absolute deadline** of a job is the time at which a job must be completed ( $d_i = r_i + D_i$ ).
- Failure to meet a **hard deadline** is considered a fatal fault, whereas a **soft deadline** can be missed as long as the average performance is optimized.

# Event-Driven Task



# Time-Driven Task



# Scheduler

- A **scheduler** assigns jobs to processors.
- A **schedule** is an assignment of all jobs in the system on available processors (produced by scheduler).
- The **execution time** (or **run-time**) of a job is the amount of time required to complete the execution of a job once it has been scheduled ( $e_i$ or $C_i$ ).
- A constraint imposed on the timing behavior of a job is called a **timing constraint**.

# Assumptions

- The scheduler works correctly; e.g., it only produces **valid schedules** that satisfy the following conditions:
  - each processor is assigned to at most one job at a time,
  - each job is assigned to at most one processor,
  - no job is scheduled before its release time, and
  - all precedence constraints and resource usage constraints are satisfied.

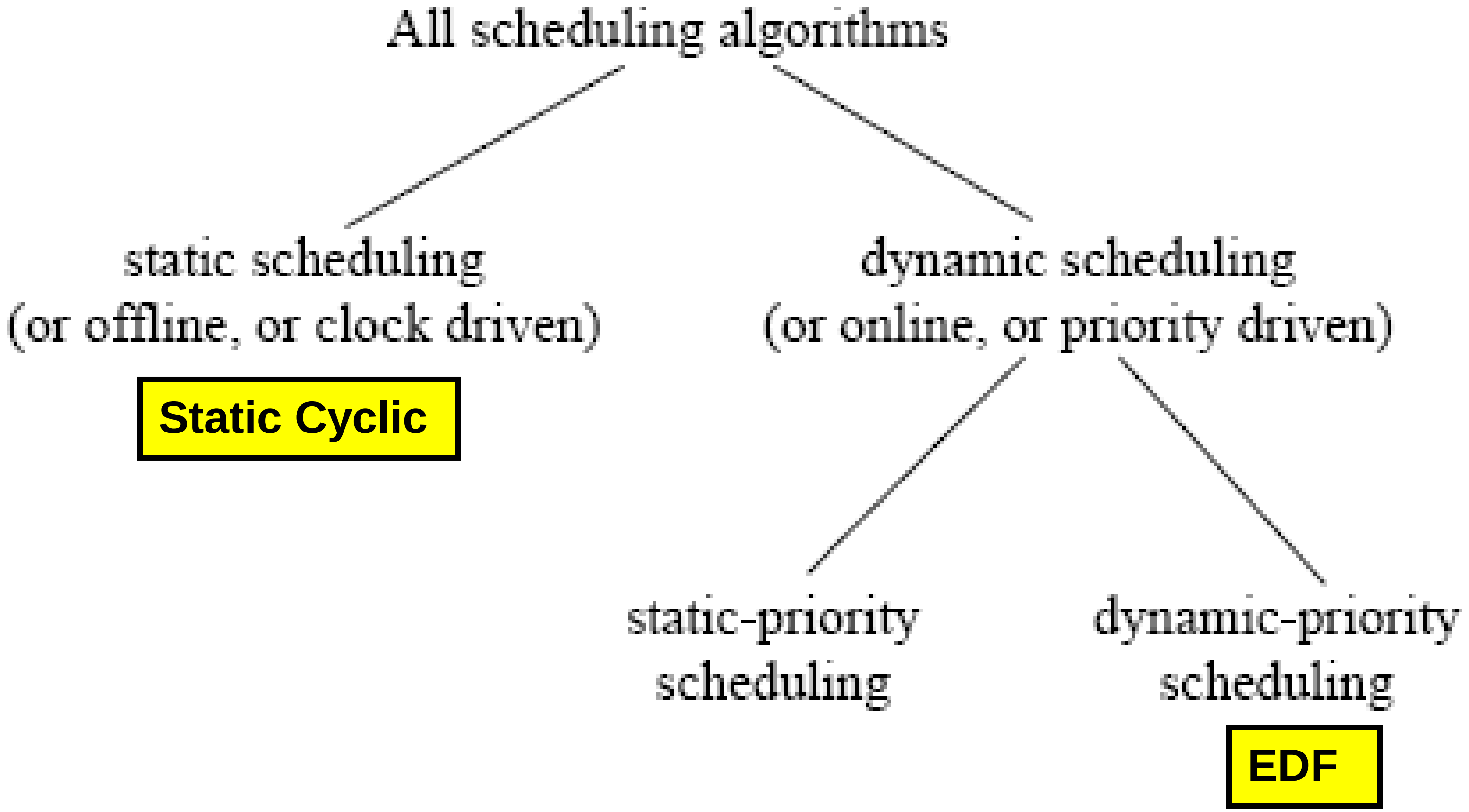
# Feasible Schedule

- A valid schedule is a **feasible schedule** if every job meets its timing constraints; e.g., completes executing by its deadline.
- A set of jobs is **schedulable** according to a scheduling algorithm if (when) using the algorithm (the scheduler) always produces a feasible schedule.

# Common Approaches For Real-Time Scheduling

- **Clock-Driven (Time-Driven) Approach –** scheduling decisions are made at specific time instants.
- **Weighted Round-Robin Approach -** every job joins a FIFO queue; when a job reaches the front of the queue, its weight refers to the fraction of processor time (number of time slices) allocated to the job.
- **Priority-Driven (Event-Driven) Approach -** ready jobs with highest priorities are scheduled for execution first.
  - Scheduling decisions are made when particular events occur; e.g., a job is released or a processor becomes idle. A **work-conserving** processor is busy whenever there is work to be done.

# Classification of Scheduling Algorithms




**EDF** is optimal for scheduling preemptive tasks on a single processor.

# EDF Algorithm

- **Earliest-Deadline-First (EDF) algorithm:**
  - At any time, execute the available job with the earliest deadline.
- **Theorem: (Optimality of EDF):** In a system with **one processor** and **preemption** allowed, EDF is optimal; that is, EDF can produce a feasible schedule for a given job set J with arbitrary release times and deadlines, if a feasible schedule exists.
- **Proof:** Apply schedule transformations and remove idle time.

# EDF may not be optimal

- When preemption is not allowed:

$$
\begin{array}{lcl}
 & & \quad r_i \quad d_i \quad e_i \\
J_1 & = & ( \; 0, \; 10, \; 3 \; ) \\
J_2 & = & ( \; 2, \; 14, \; 6 \; ) \\
J_3 & = & ( \; 4, \; 12, \; 4 \; )
\end{array}
$$

- When more than one processor is used:

$$
\begin{array}{lcl}
 & & \quad r_i \quad d_i \quad e_i \\
J_1 & = & ( \; 0, \; 4, \; 1 \; ) \\
J_2 & = & ( \; 0, \; 4, \; 1 \; ) \\
J_3 & = & ( \; 0, \; 5, \; 5 \; )
\end{array}
$$

# Periodic Task Model

- **Tasks:** $T_1, \ldots\ldots, T_n$
- Each consists of a set of **jobs**: $T_i = \{J_{i1}, J_{i2}, \ldots\ldots\ldots\}$
- $\phi_i$: p**hase** of task $T_i$ = time when its first job is released
- $p_i$: p**eriod** of $T_i$ = minimum inter-release time
- H: h**yperperiod** $H = \mathrm{lcm}(p_1, \ldots\ldots, p_n)$
- $e_i$: e**xecution time** of $T_i$
- $u_i$: **utilization** of task $T_i$ is given by $u_i = e_i / p_i$
- $D_i$: (relative) **deadline** of $T_i$, typically $D_i = p_i$

# Periodic Task

- We refer to a periodic task $T_i$ with phase $\phi_i$ , period $p_i$ , execution time $e_i$ , and relative deadline $D_i$ by the 4-tuple ( $\phi_i$ , $p_i$ , $e_i$ , $D_i$ ).
- Example: ( 1, 10, 3, 6 )
- By default, the phase of each task is 0, and its relative deadline is equal to its period.
- Example: ( 0, 10, 3, 10 ) = ( 10, 3 ).

# Static Cyclic Scheduling

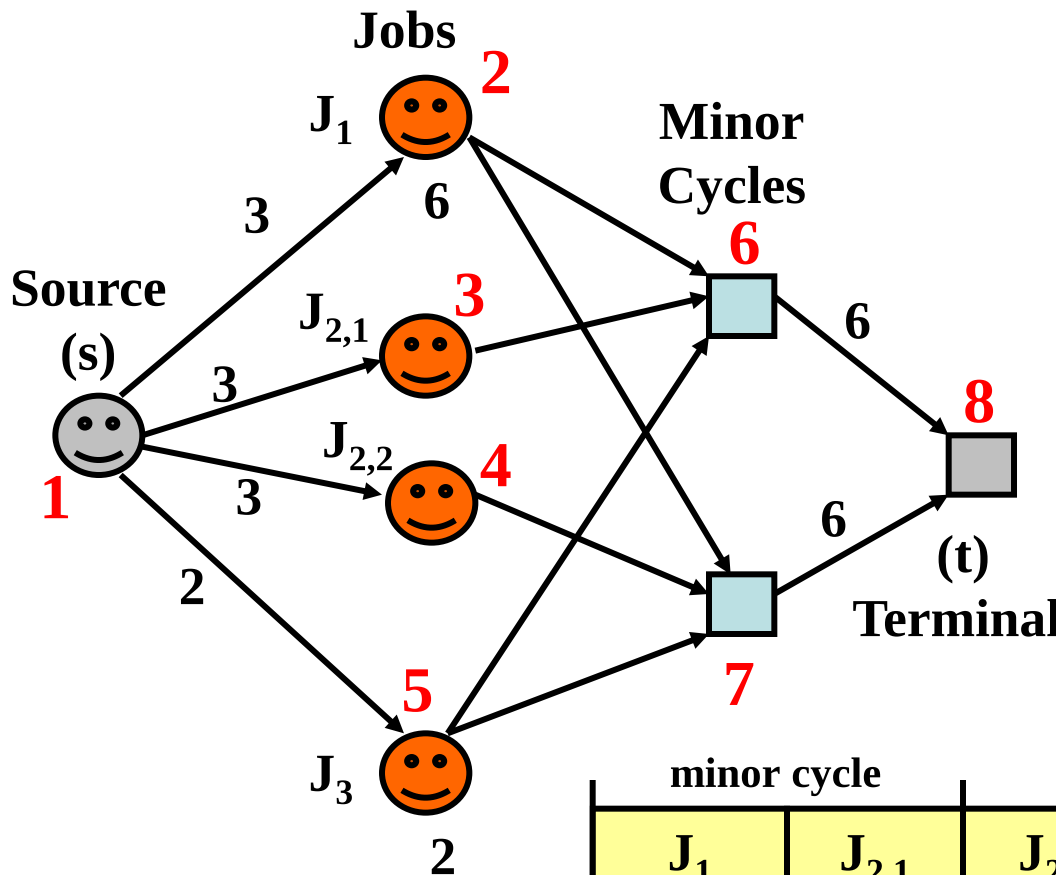
- Periodic Task 1: (12, 3)
- Periodic Task 2: (6, 3)
- Periodic Task 3: (12, 2)

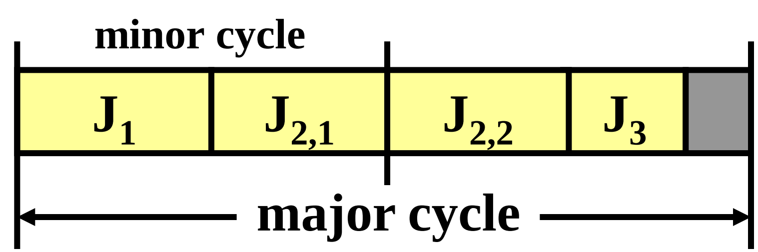
- Hyperperiod H = lcm(12, 6, 12) = 12
- Potential frame sizes? 3 or 6

# Clock-Driven Example

**$ hi_pr < cyclic2.input > cyclic2.output**

**INPUT:**
p max 8 12
n 1 s
n 8 t
a 1 2 3
a 1 3 3
a 1 4 3
a 1 5 2
a 2 6 6
a 2 7 6
a 3 6 6
a 4 7 6
a 5 6 6
a 5 7 6
a 6 8 6
a 7 8 6



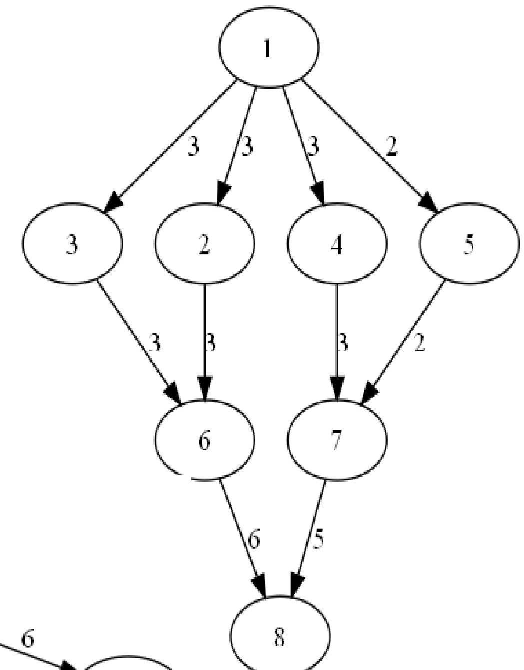



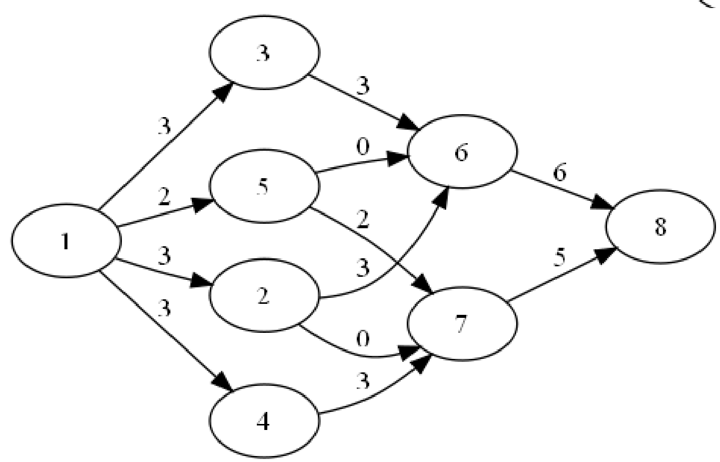

**OUTPUT:**
max flow:11
c flow values
f 1 2 3
f 1 4 3
f 1 3 3
f 1 5 2
f 2 6 3
f 2 7 0
f 3 6 3
f 4 7 3
f 5 7 2
f 5 6 0
f 6 8 6
f 7 8 5
c

# Resulting Graphics File

```
digraph g {
1 -> 2 [label=3]
1 -> 4 [label=3]
1 -> 3 [label=3]
1 -> 5 [label=2]
2 -> 6 [label=3]
3 -> 6 [label=3]
4 -> 7 [label=3]
5 -> 7 [label=2]
6 -> 8 [label=6]
7 -> 8 [label=5]
{ rank=same;  2; 3; 4; 5;}
{ rank=same;  6; 7;}
}
```

**`dot -Tpng fn.gr -o fn.png`**

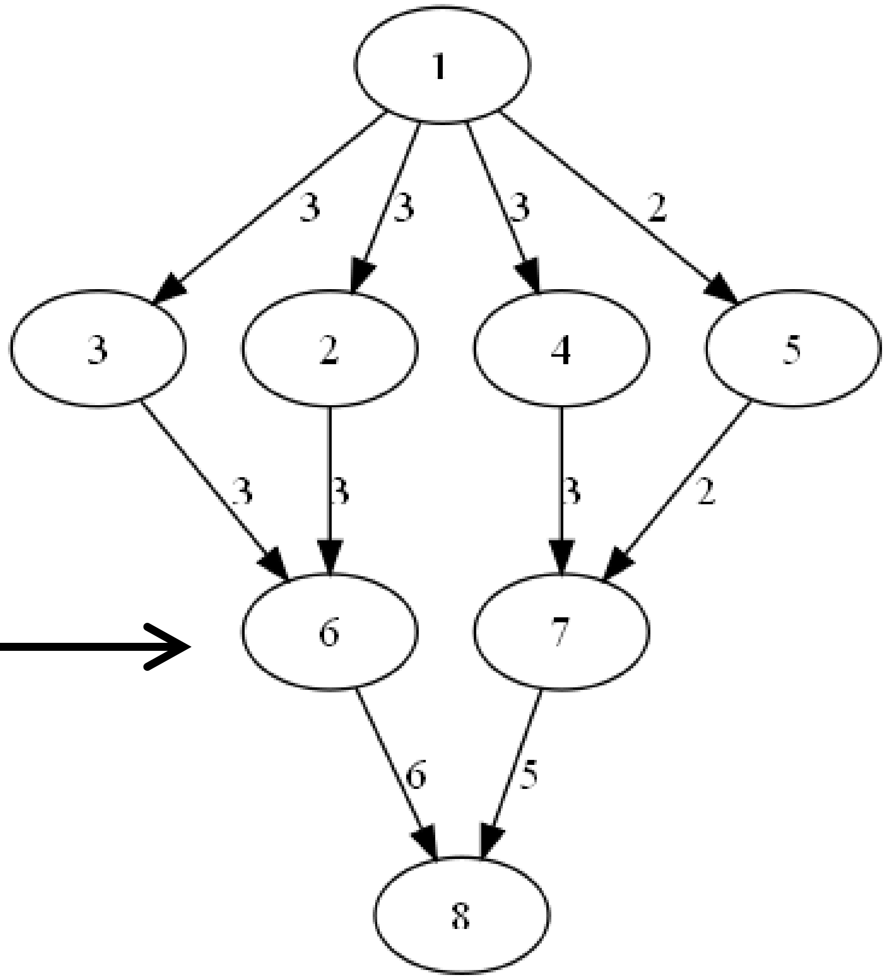


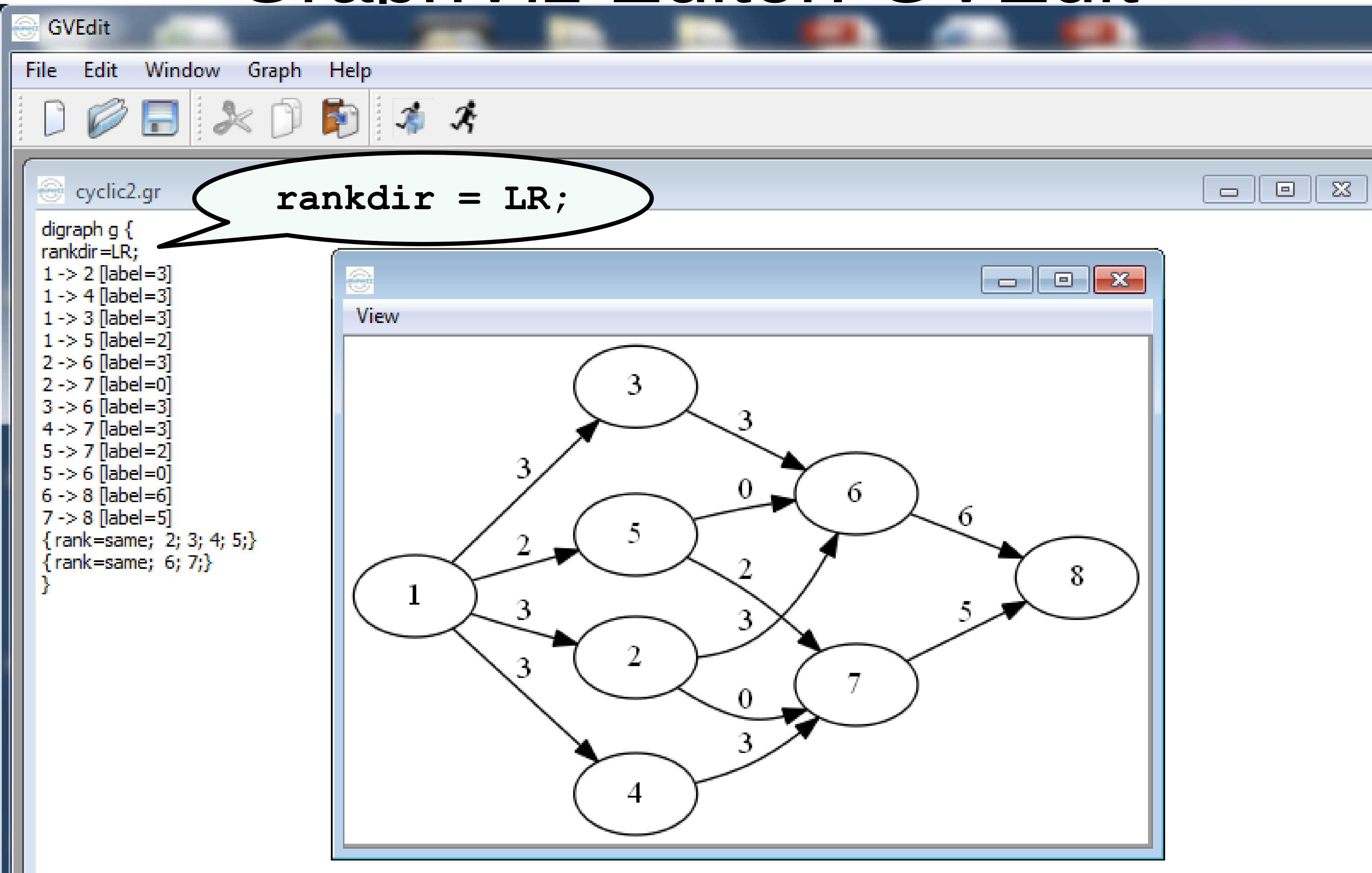
# Output generated

Jobs

Frames



# GraphViz Editor: GVEdit

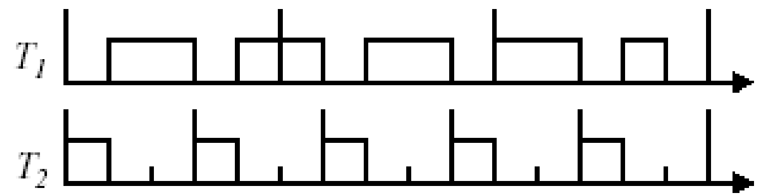


# Static-Priority *vs.* Dynamic Priority

- **Static-Priority**: All jobs in task have same priority.
- example:

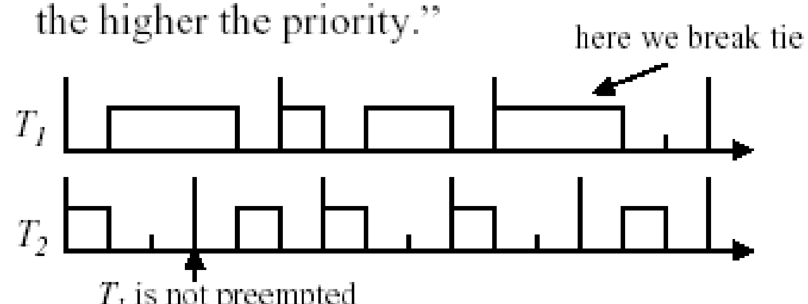
  **Rate-Monotonic**: "The shorter the period, the higher the priority."

$$T_1 = (5, 3, 5)$$
$$T_2 = (3, 1, 3)$$




- **Dynamic-Priority**: May assign different priorities to individual jobs.
- example:

  **Earliest-Deadline-First**: "The nearer the absolute deadline, the higher the priority."



# Rate-Monotonic Algorithm (RM)

- The **rate** of a task is the inverse of its period ( $f_i = 1 / p_i$ ).
- Tasks with **higher rates (shorter periods)** are assigned **higher priorities**.
- C. L. Liu and J. W. Layland, “Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment”,  JACM, Vol. 20, No. 1, pages 46-61, 1973.

# Deadline-Monotonic Algorithm (DM)

- Tasks with **shorter relative deadlines** are assigned **higher priorities**.
- When tasks have relative deadlines ( $D_i$ ) equal to their periods ( $p_i$ ), the rate-monotonic algorithm is the same as the deadline-monotonic algorithm.

# Optimal Priority Assignment

- A given priority assignment algorithm is **optimal** if whenever a task set can be scheduled by some fixed priority assignment, it can also be scheduled by the given algorithm.
- Liu and Layland show that the rate-monotonic (RM) algorithm is optimal, for preemptive, periodic task sets, with phase 0 and relative deadlines equal to their periods.

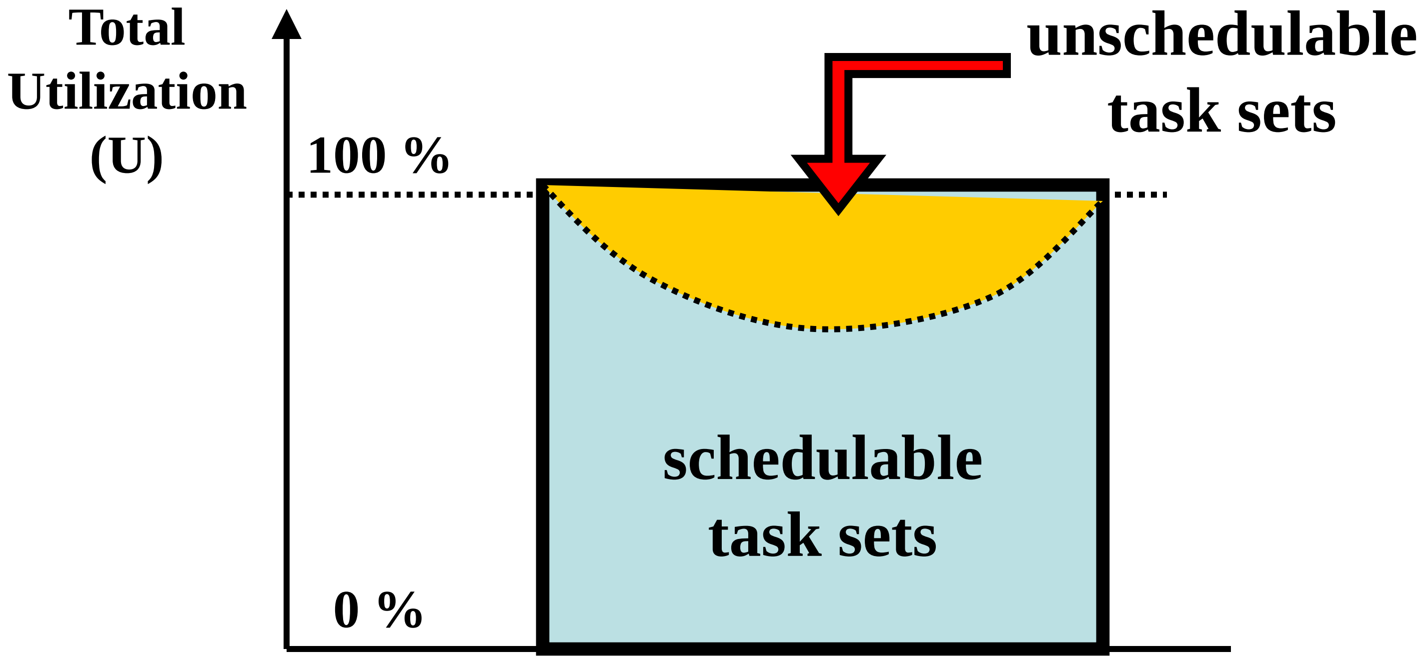
# Processor Utilization

- Given a periodic task $T_i$ , the ratio $u_i = e_i / p_i$ is called the **utilization of task $T_i$** .
- The **total utilization U** of all tasks in a system is the sum of the utilizations of all individual tasks:

$$\boxed{U = \sum_{i=1}^{n} \frac{e_i}{p_i}}$$

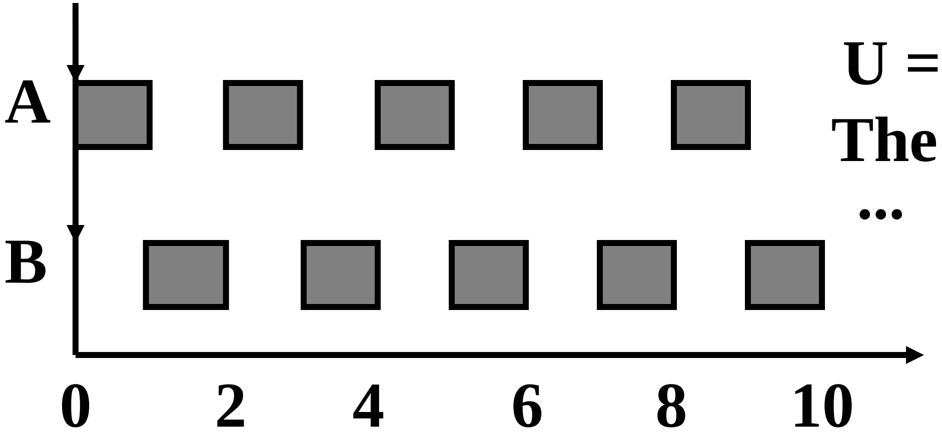
# Maximum Achievable Utilization

A task set is **fully utilized** if any increase in run-time would result in a missed deadline.



# Example #1

| Task $T_i$ | | Period $p_i$ | Deadline $D_i$ | Run-Time $e_i$ | Phase $\varphi_i$ |
| --- | --- | --- | --- | --- | --- |
| A | (High Priority) | 2 | 2 | 1 | 0 |
| B | (Low Priority) | 2 | 2 | 1 | 0 |



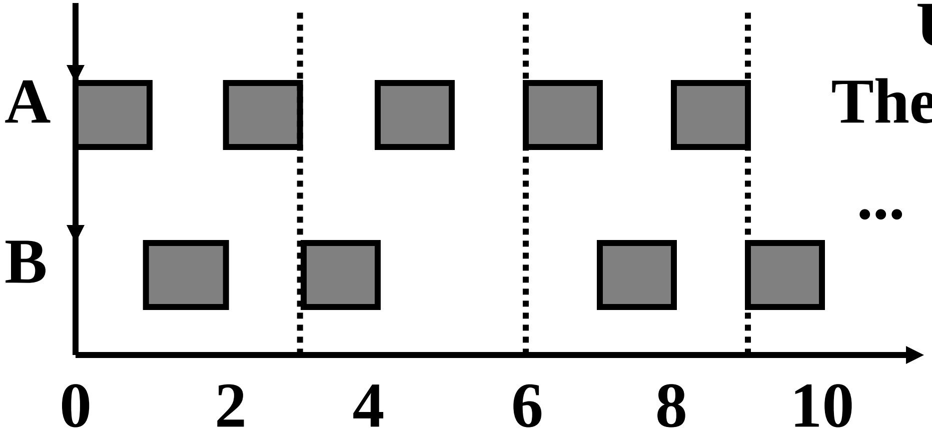

U = 1/2 + 1/2 = 1.0 = 100 %

The task set is fully utilized.

...

# Example #2

| Task $T_i$ | | Period $p_i$ | Deadline $D_i$ | Run-Time $e_i$ | Phase $\varphi_i$ |
| --- | --- | --- | --- | --- | --- |
| A | (High Priority) | 2 | 2 | 1 | 0 |
| B | (Low Priority) | 3 | 3 | 1 | 0 |

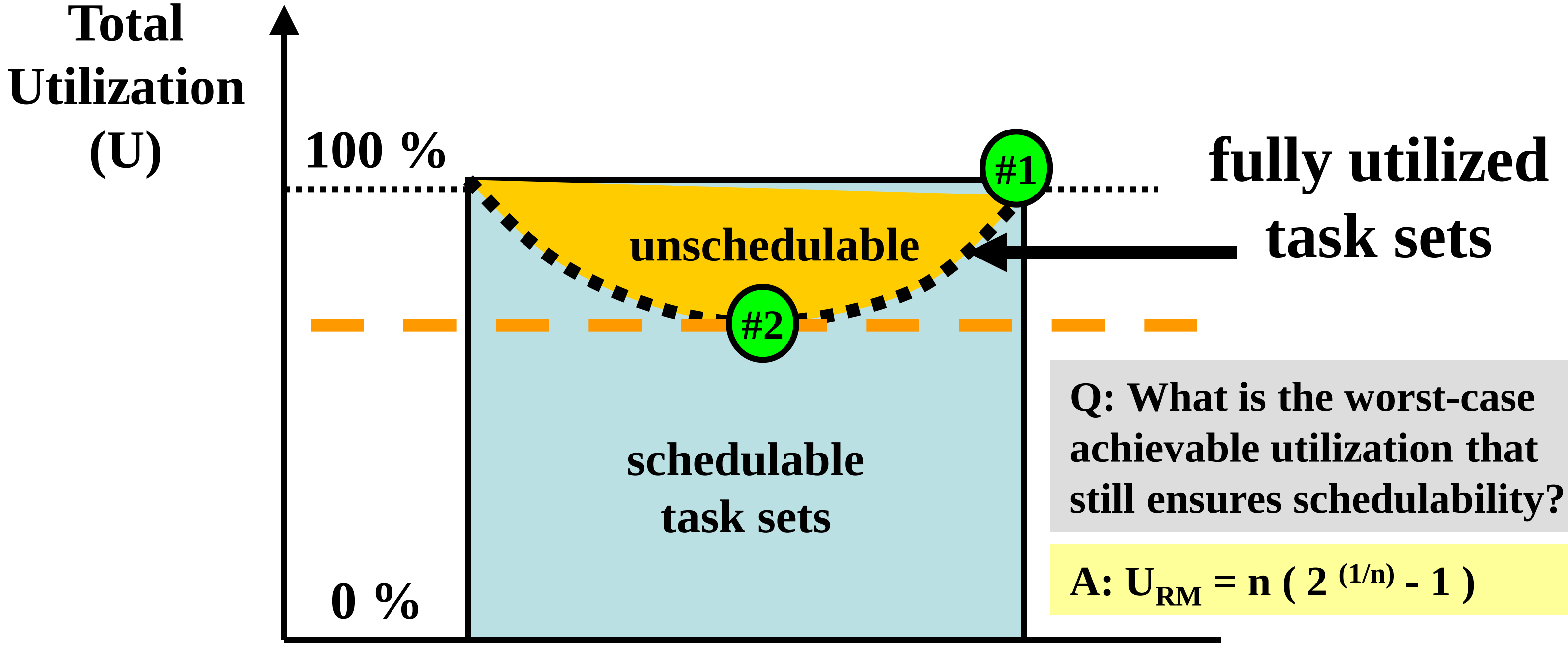



U = 1/2 + 1/3 = 0.8333

The task set is fully utilized, even though U < 1.0.

# Fully utilized task sets

A task set is **fully utilized** if any increase in run-time would result in a missed deadline.




**Q: What is the worst-case achievable utilization that still ensures schedulability?**

**A:** $U_{RM} = n \left( 2^{(1/n)} - 1 \right)$

# Utilization-Based Test

- **A sufficient, but not necessary,** test that can be used to test the schedulability of a task set that is assigned priorities using the rate-monotonic algorithm, with deadlines equal to periods.
- Compute total task utilization U(n) = U.
- Compare with worst-case utilization bound (also called **schedulable utilization**) $U_{RM}(n) = U_{RM}$:
  - If **U > 1**, then the task set is not schedulable.
  - If $\mathbf{U \le U_{RM}}$, then the task set is schedulable.
  - Otherwise, no conclusion can be made.

# Schedulability Analysis

- **Utilization-Based Tests**
  - Not exact (sufficient, but not necessary).
  - Not applicable to more general task models.
- **Time-Based Tests (Response Time Analysis)**
  - Use analytic approach to predict worst-case response time of each task.
  - Compare computed worst-case response times with deadlines.
  - Exact (sufficient and necessary)

# Example

| Task $\tau_i$ | | Period $T_i$ | Deadline $D_i$ | Run-Time $C_i$ | Phase $\varphi_i$ |
|---|---|---|---|---|---|
| A | (High Priority) | 7 | 7 | 3 | 0 |
| B | | 12 | 12 | 3 | 0 |
| C | (Low Priority) | 20 | 20 | 5 | 0 |

- $U = 3/7 + 3/12 + 5/20 = 13/14 \approx 0.93$
- $U_{RM} = 3\,(2^{1/3} - 1) \approx 0.78$
- Since $U_{RM} < U \leq 1.0$, no conclusion can be drawn using the Utilization-Based Test.

# Response Time

- The **response time** ($R_i$) for task $T_i$ is given by $R_i = e_i + I_i$ where:
  - $e_i$ is the execution time of each job in $T_i$, and
  - $I_i$ is the **maximum interference** caused by higher priority tasks in any interval $[t, t + R_i)$.

# Maximum Interference ($I_i$)

- The number of releases of task $T_j$ in [ t, t + $R_i$ ) is given by $\left\lceil \frac{R_i}{p_j} \right\rceil$
- The interference caused by task $T_j$ is $\left\lceil \frac{R_i}{p_j} \right\rceil * e_j$
- The maximum interference caused by all higher priority tasks is given by

$$I_i = \sum_{j \in hp(i)} \left\lceil \frac{R_i}{p_j} \right\rceil * e_j$$

where *hp(i)* = set of all tasks with priority greater than task $T_j$ .

# Response Time Analysis

- The (**worst-case**) **response time** ($w_i$) for task $T_i$ is given by the implicit equation:

$$R_i = e_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{p_j} \right\rceil * e_j$$

- Solve by forming a recurrence relation:

$$w_i^{n+1} = e_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{p_j} \right\rceil * e_j$$

$$w_i^0 = e_i$$

# Solving Recurrence

- The sequence $w_i^0, w_i^1, w_i^2, \ldots, w_i^n$

  is non-decreasing:

  - If $w_i^{n+1} = w_i^n$, then a fixed point (solution) has been found.

  - If $w_i^{n+1} > D_i$, then no solution exists.

Algorithm

Input: $e_1,..,e_m,\ p_1,..,\ p_m, D_1,..,D_m$

Output: $R_1, R_2,...,R_m$

for $i = 1$ to $m$

$n = 0$

$w_i^n = e_i$

loop

$$w_i^{n+1} = e_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{p_j} \right\rceil * e_j$$

if $w_i^{n+1} = w_i^n$ then

$R_i = w_i^n$

break out of loop {solution found}

if $w_i^{n+1} > D_i$ then

break out of loop {no solution}

$n = n + 1$

end loop

end for

# Response Time Analysis

- For each task, $T_i$ , compute worst-case response time ( $R_i$ ).
- If ( $R_i \leq D_i$ ) for each task $T_i$ **,** then the task set is feasible (schedulable).
- Response Time Analysis is both necessary and sufficient, and works for task sets with deadlines different than periods. If deadlines are greater than periods, then need to test jobs in task $T_i$ over a level-i busy period and assign priorities from lowest to highest.

# Preemption Thresholds Task Model

- Task Set $\Gamma = \{\tau_1, \tau_2, \tau_3, \ldots, \tau_n\}$
  - Each task $\tau_i$ is characterized by ( $C_i$, $T_i$, $D_i$ ), denoted $\tau_i \sim$ ( $C_i$, $T_i$, $D_i$ ).
  - Each task $\tau_i$ is assigned a priority $\pi_i \in \{1,2,..,n\}$
  - and a preemption threshold $\gamma_i \in \{\pi_i, \pi_i + 1,..,n\}$.
- **Notes:**
  - 1 = lowest priority, n = highest priority.
  - $\pi_i$ = static priority.
  - $\gamma_i$ = dynamic priority.

# Run-Time Model

- Modified fixed-priority, preemptive scheduling.
- When task $\tau_i$ is released, it is scheduled using its static priority $\pi_i$ .
- After task $\tau_i$ starts executing, another task $\tau_j$ can preempt $\tau_i$ only if $\pi_j > \gamma_i \geq \pi_i$ .

# Extremes

- If $\gamma_i = \pi_i$ for each i, then the result is preemptive, priority-based scheduling.
- If $\gamma_i$ = n (max. priority) for each i, then the result is non-preemptive, priority-based scheduling.

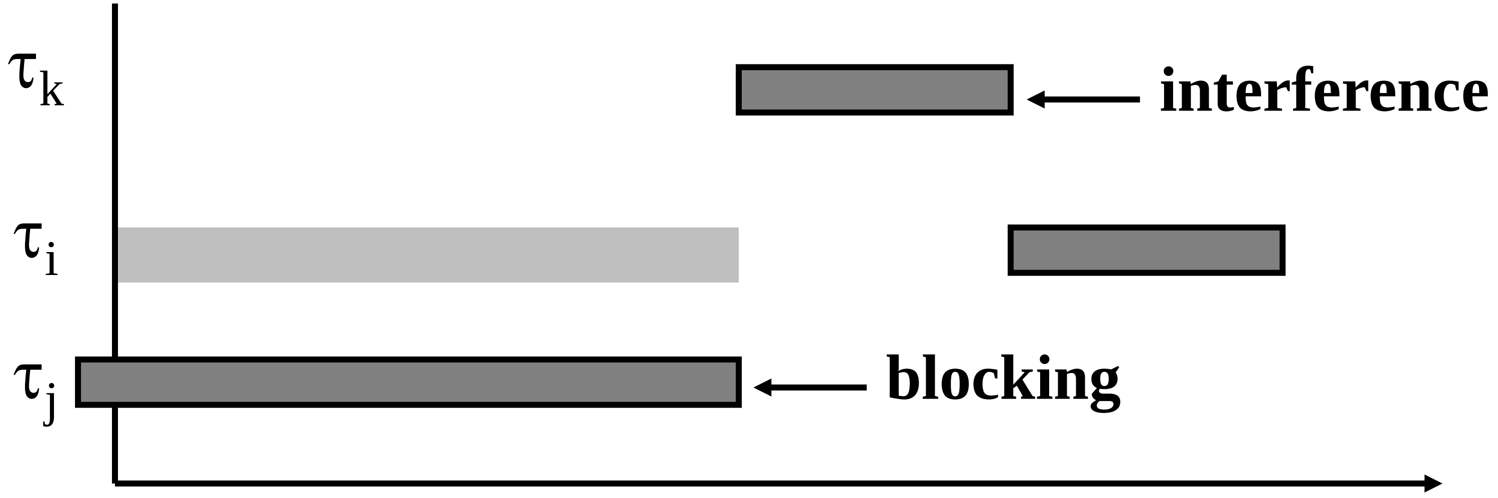
# Response Time Analysis

- The **blocking time** of task $\tau_i$ is denoted $B(\tau_i)$. Blocking occurs if a lower priority task is running and task $\tau_i$ cannot preempt it.

$$B(\tau_i) = \max_j \{C_j | \gamma_j \geq \pi_i > \pi_j\}$$

# Busy Period Analysis

- A **critical instant** occurs when all higher priority tasks arrive at the same time, and the task that contributes to the maximum blocking arrives at the critical instant - $\varepsilon$.



# Divide Busy Period

- Divide the busy period for $\tau_i$ into two parts:
  - the length of time from the critical instant (time 0) to the point when $\tau_i$ starts executing its $q^{th}$ job ( $\mathbf{S_i(q)}$ ).
  - the length of time from the time $\tau_i$ starts executing its $q^{th}$ job until it finishes executing its $q^{th}$ job ($\mathbf{F_i(q)-S_i(q)}$ ).
- Let q = 1, 2, ... , m until we reach q = m s.t. $F_i(m) \leq m T_i$ that is, the $m^{th}$ job completes before the next job is released.
- Then,

$$R_i = \max_{q \in \{1,..,m\}} \{F_i(q) - (q-1)T_i\}$$

# Worst-Case Start Time ( $\mathbf{S_i(q)}$ )

$$S_i(q) = B(\tau_i) + (q-1)C_i + \sum_{\substack{j \in \{1,..,n\} \\ \pi_j > \pi_i}} \left(1 + \left\lfloor \frac{S_i(q)}{T_j} \right\rfloor\right) C_j$$

# Worst-Case Finish Time ( $\mathbf{F_i(q)}$ )

$$F_i(q) = S_i(q) + C_i + \sum_{\substack{j \in \{1,..,n\} \\ \pi_j > \gamma_i}} \left( \left\lceil \frac{F_i(q)}{T_j} \right\rceil - \left(1 + \left\lfloor \frac{S_i(q)}{T_j} \right\rfloor \right) \right) C_j$$

Algorithm to compute $R_i$

Input: $C_1, \ldots, C_m, T_1, \ldots, T_m, \pi_1, \ldots, \pi_m, \gamma_1, \ldots, \gamma_m$

Output: $R_1, R_2, \ldots, R_m$

done = FALSE

q = 1

while (not done)

  compute $S_i$(q) and $F_i$(q)

  if $F_i(q) \le q\, T_i$ then

    done = TRUE

    m = q

  else

    q = q + 1

  end if

end while

$R_i = \max_{q \in \{1,..,m\}} (F_i(q) - (q-1)\, T_i)$

# Preemption Threshold Assignment

- Assign preemption thresholds from lowest priority to highest priority.
- Only elevate preemption threshold as much as needed to make a given task feasible.

# Summary

- Quiz #1
  - Fri., Oct. 16, in class
  - Open book, open notes
  - 50 minutes