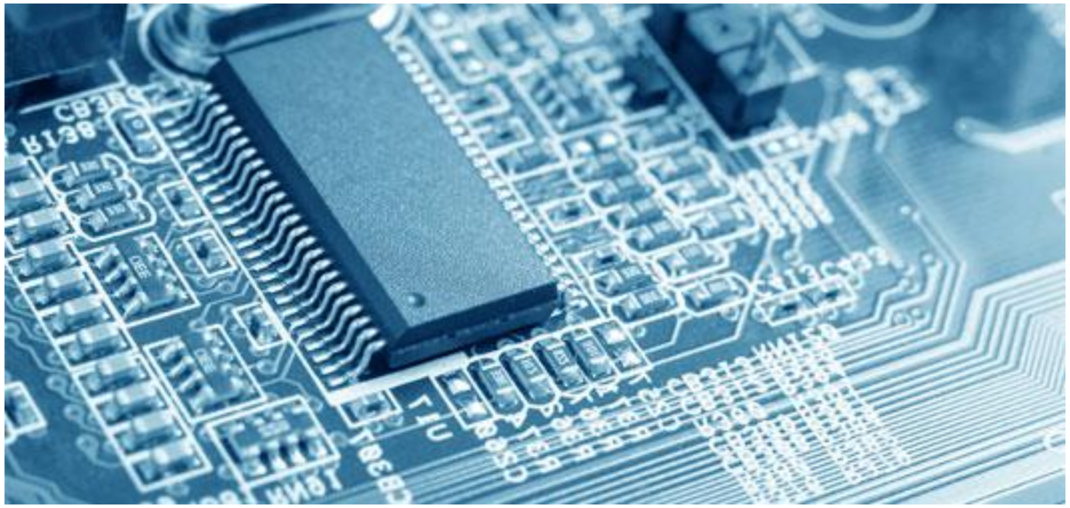
NAZARBAYEV UNIVERSITY
SCHOOL OF SCIENCE AND TECHNOLOGY



ROBT305 - Embedded Systems

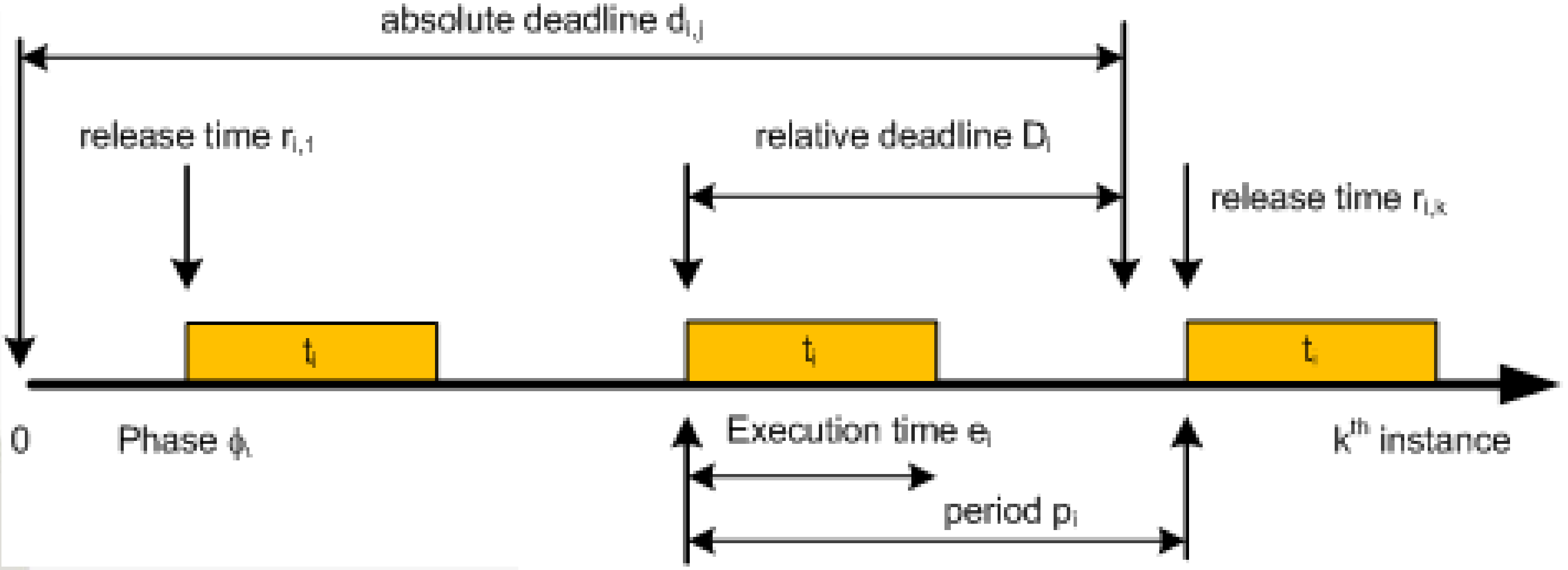
Lectures 9-10 – Clock Driven and Rate Monotonic Scheduling

22-29 September, 2015

Course Logistics

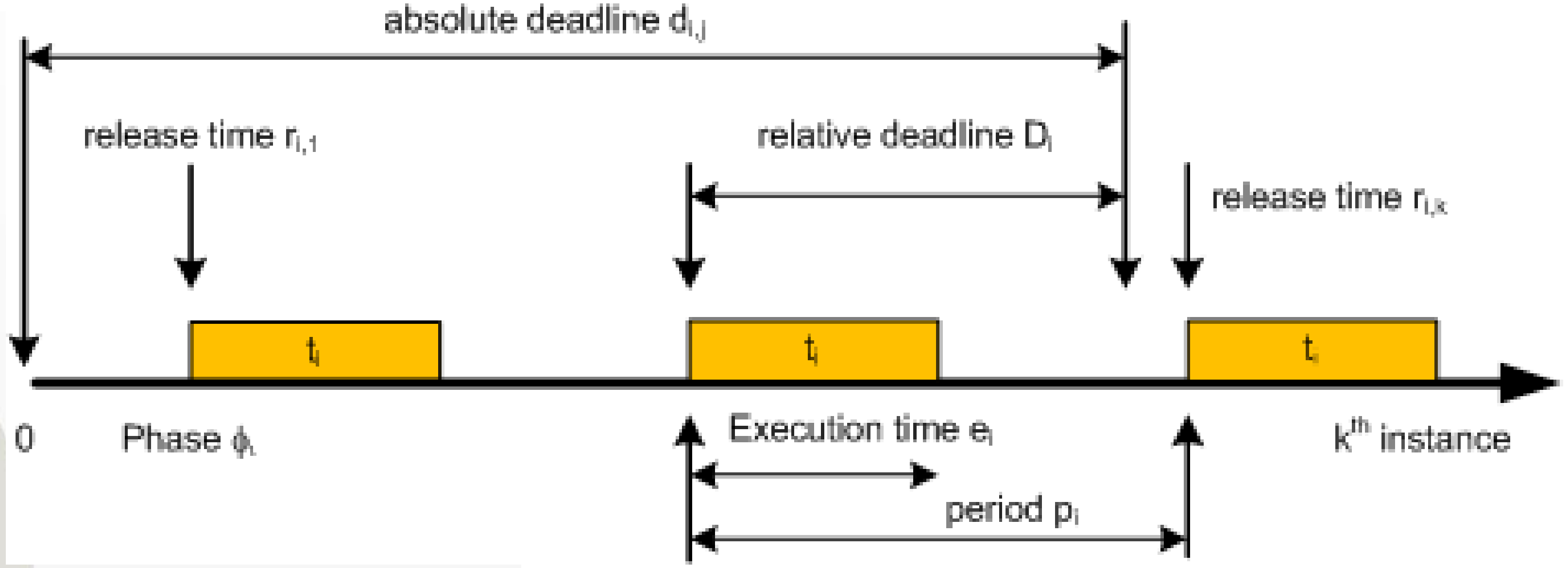
Reading Assignment:
Chapter 5 of the Operating Systems Concept textbook **Chapter 3** of the Real-Time Systems Design and Analysis textbook
Homework Assignment #2 is out in Moodle and due to end of 26 September (Sunday)
Quiz #3 is on 1 October – Semaphores, Scheduling

Clock Driven Scheduling: Assumptions



- Clock-driven scheduling applicable to deterministic systems
- A restricted periodic task model:
  - The parameters of all periodic tasks are known a priori
  - For each mode of operation, system has a fixed number, *n*, periodic tasks
    - For task T_i each job $J_{i,k}$ is ready for execution at its release time $r_{i,k}$ and is released p_i units of time after the previous job in T_i such that $r_{i,k} = r_{i,k-1} + p_i$
    - Variations in the inter-release times of jobs in a periodic task are negligible

Clock Driven Scheduling: Assumptions



- Aperiodic jobs may exist
  - Assume that the system maintains a single queue for aperiodic jobs
  - Whenever the processor is available for aperiodic jobs, the job at the head of this queue is executed

- There are no sporadic jobs
  - Recall: sporadic jobs have hard deadlines, aperiodic jobs do not

Static Cyclic Code (CC) Scheduling

- A simple approach that generates a complete and highly predictable schedule.
- CC scheduler deterministically interleaves and sequentializes the execution of periodic tasks on a processor according to a pre-run-time schedule.
- In general terms, the CC scheduler is a table of procedure calls, where each task is a procedure, and it executes a single do loop.
- Scheduling decisions are made periodically, rather than at arbitrary times.
- Time intervals during scheduling decision points are referred to as **frames** or **minor cycles**, and every frame has a length ***f*** called the **frame size**.

Cyclic Code Scheduling

- The **major cycle** is the minimum time required to execute tasks allocated to the processor ensuring the deadlines and periods of all processes are met.
- The major cycle or the **hyperperiod** is equal to the least common multiple of the periods, i.e., **lcm(p_1, … p_n).**
- As scheduling decisions are made only at the beginning of every frame, there is no preemption within each frame.
- The phase of each periodic task is a nonnegative integer multiple of the frame size.
- The scheduler carries out monitoring and enforcement actions at the beginning of each frame.

Cyclic Code Scheduling: Notations

The 4-tuple $T_i = (\phi_i, p_i, e_i, D_i)$ refers to a periodic task T_i with phase ϕ_i, period p_i, execution time e_i, and relative deadline D_i

- Default phase of T_i is $\phi_i = 0$, default relative deadline is the period $D_i = p_i$
- Omit elements of the tuple that have default values
- Examples:

$T_1 = (1, 10, 3, 6) \Rightarrow$ $\phi_1 = 1$ $p_1 = 10$ $e_1 = 3$ $D_1 = 6$

$J_{1,1}$ released at 1, deadline 7
$J_{1,2}$ released at 11, deadline 17
…

$T_2 = (10, 3, 6) \Rightarrow$ $\phi_2 = 0$ $p_2 = 10$ $e_2 = 3$ $D_2 = 6$

$J_{1,1}$ released at 0, deadline 6
$J_{1,2}$ released at 10, deadline 16
…

$T_3 = (10, 3) \Rightarrow$ $\phi_3 = 0$ $p_3 = 10$ $e_3 = 3$ $D_3 = 10$

$J_{1,1}$ released at 0, deadline 10
$J_{1,2}$ released at 10, deadline 20
…

Cyclic Code Scheduling: Example

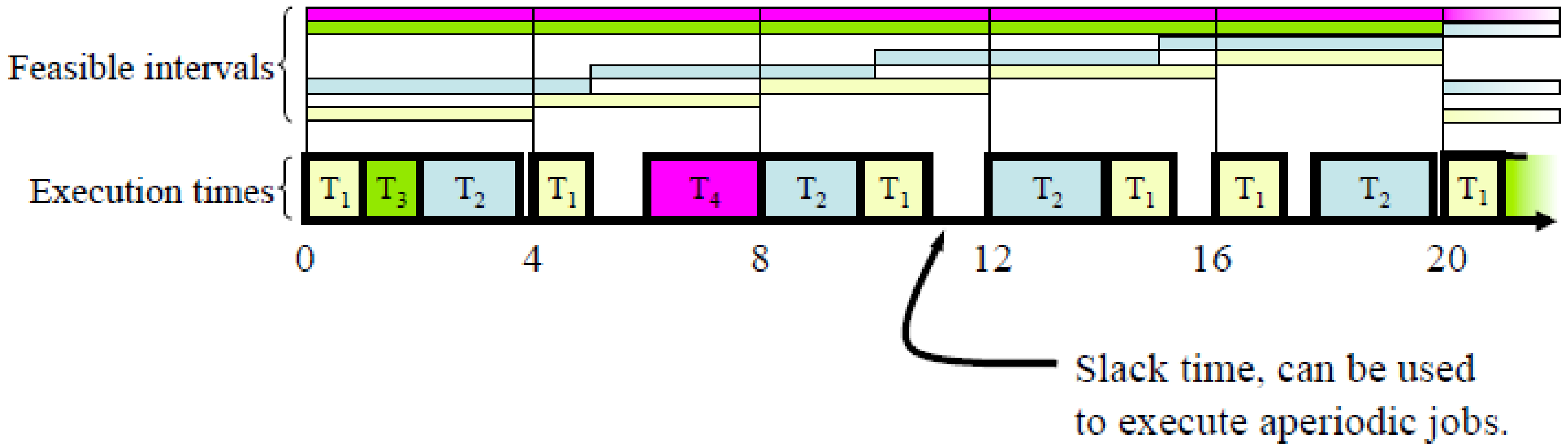
- Consider a system with 4 independent periodic tasks:
  - $T_1 = (4, 1.0)$
  - $T_2 = (5, 1.8)$
  - $T_3 = (20, 1.0)$
  - $T_4 = (20, 2.0)$

Execution time

Period

[Phase and deadline take default values]

- Hyper-period $H = 20$ (least common multiple of 4, 5, 20, 20)
- Can construct an arbitrary static schedule to meet all deadlines:



Frame Size Calculation

Frames must be sufficiently long so that every task can start and complete within a single frame. This implies that the frame size, f, is to be longer than the execution time, e_i, of every task, τ_i, that is,

$$C_1 : f \geq \max_{i \in [1,n]}(e_i). \qquad (1)$$

In order to keep the length of the cyclic schedule as short as possible, the frame size, f, should be chosen so that the hyperperiod has an integer number of frames:

$$C_2 : \lfloor p_{\text{hyper}} / f \rfloor - p_{\text{hyper}} / f = 0. \qquad (2)$$

Moreover, to ensure that every task completes by its deadline, frames must be short so that between the release time and deadline of every task, there is at least one frame. The following relation is derived for a worst-case scenario, which occurs when the period of a task starts just after the beginning of a frame, and, consequently, the task cannot be released until the next frame:

$$C_3 : 2f - \gcd(p_i, f) \leq D_i. \qquad (3)$$

where "gcd" is the greatest common divisor, and D_i is the relative deadline of task τ_i. This condition should be evaluated for all schedulable tasks.

Frame Size Calculation Example

- Review the system of independent periodic tasks from our earlier example:

  $T_1 = (4, 1.0)$ $T_2 = (5, 1.8)$

  $T_3 = (20, 1.0)$ $T_4 = (20, 2.0)$

Hyper-period $H = \text{lcm}(4, 5, 20, 20) = 20$

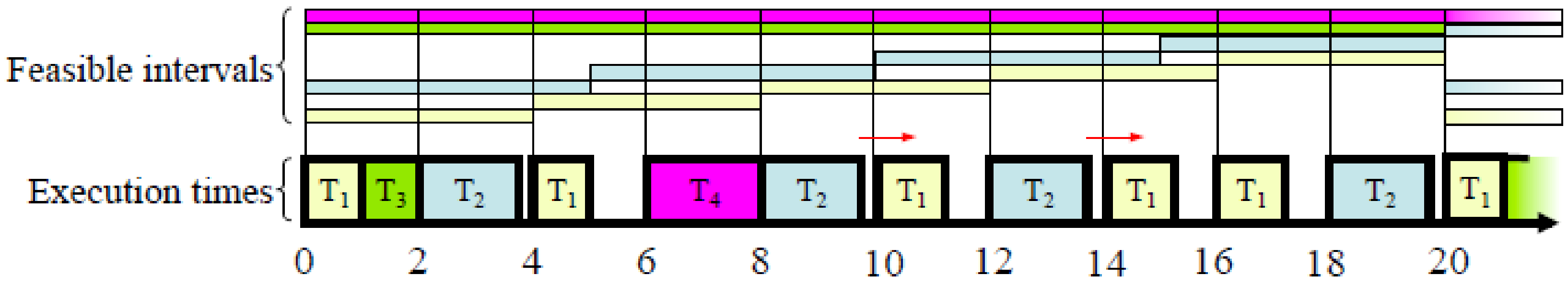
- Constraints:

$$\text{Eq.1} \Rightarrow f \geq \max(1, 1.8, 1, 2) \geq 2$$

$$\text{Eq.2} \Rightarrow f \in \{ 2, 4, 5, 10, 20 \}$$

$$\begin{aligned} \text{Eq.3} \Rightarrow\ & 2f - \gcd(4, f) \leq 4 \quad (T_1) \\ & 2f - \gcd(5, f) \leq 5 \quad (T_2) \\ & 2f - \gcd(20, f) \leq 20 \quad (T_3, T_4) \end{aligned}$$

Value of *f* that satisfy all the constrains is *f* = 2



Frame Size Calculation Example 2

- Consider a set of three tasks specified in the table

τ_i	p_i	e_i	D_i
τ_1	15	1	15
τ_2	20	2	20
τ_3	22	3	22

- The hyperperiod is equal to 660 since the least common multiple of 15, 20 and 22 is 660.
- The three required conditions are evaluated as follows:

$$C_1 : f \geq \max_{i \in [1,3]} (e_i) \Rightarrow f \geq 3$$

$$C_2 : \lfloor p_{\text{hyper}} / f \rfloor - p_{\text{hyper}} / f = 0 \Rightarrow f = 2, 3, 4, 5, 6, 10, \ldots$$

$$C_3 : 2f - \gcd(p_i, f) \leq D_i \Rightarrow f = 2, 3, 4, 5, 6, 7$$

- Possible value for ***f*** could be any of the values of 3, 4, 5 or 6.

Frame Size Calculation: Job Slices

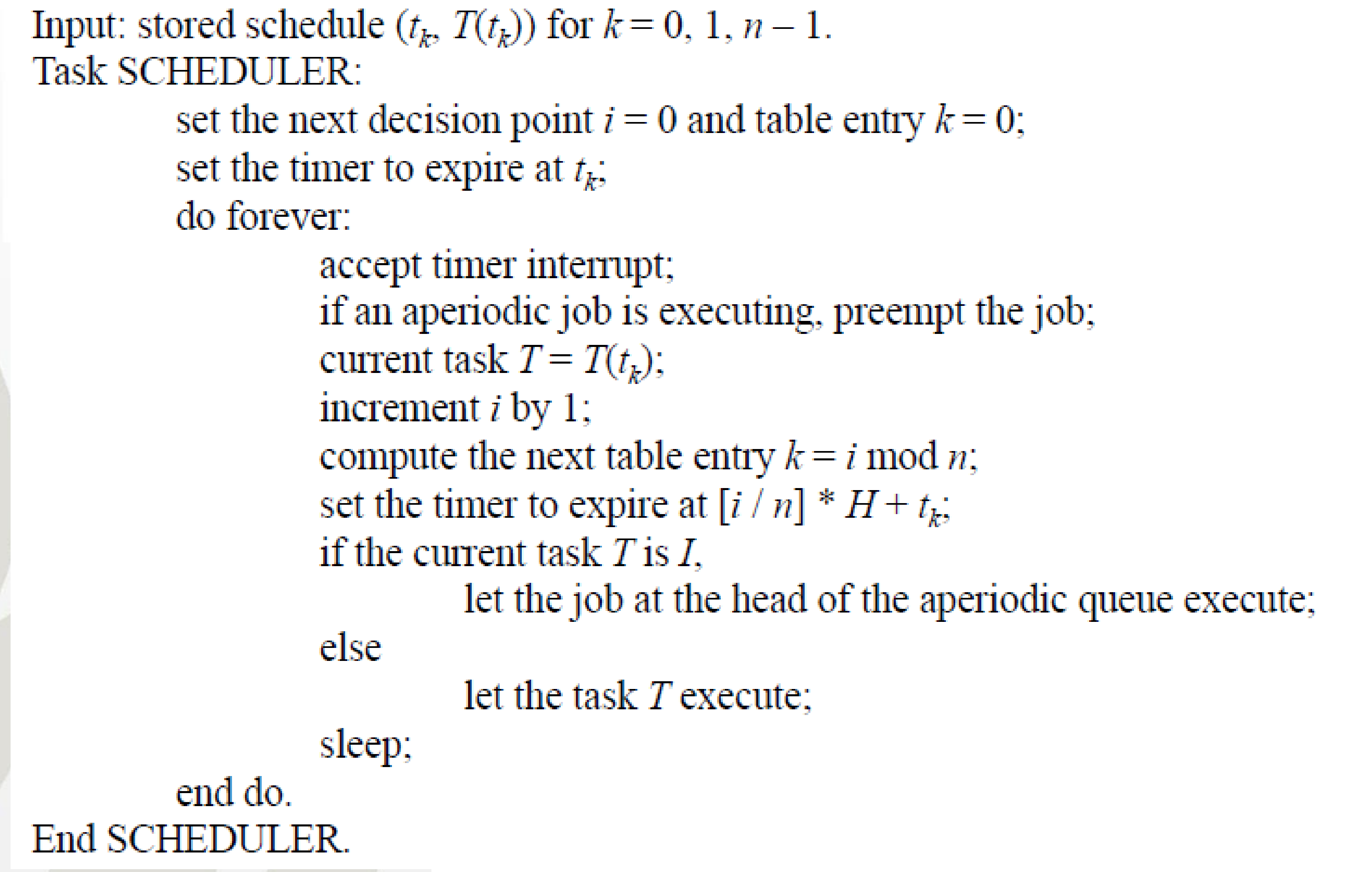
- Sometimes, a system cannot meet all three frame size constraints simultaneously
- Can often solve by partitioning a job with large execution time into slices (sub-jobs) with shorter execution times/deadlines
  - Gives the effect of preempting the large job, so allow other jobs to run
  - Sometimes need to partition jobs into more slices than required by the frame size constraints, to yield a feasible schedule
- Example:
  - Consider a system with $T_1 = (4, 1)$, $T_2 = (5, 2, 7)$, $T_3 = (20, 5)$
  - Cannot satisfy constraints: Eq.1 $\Rightarrow f \geq 5$ but Eq.3 $\Rightarrow f \leq 4$
  - Solve by splitting T_3 into $T_{3,1} = (20, 1)$, $T_{3,2} = (20, 3)$ and $T_{3,3} = (20,1)$
    - Other possible splits exist; pick based on application domain knowledge
  - Result can be scheduled with $f = 4$

Implementation of Cyclic Scheduler

- Store pre-computed schedule as a table
- The system creates all the tasks that are to be executed:
  - Allocates sufficient memory for the code and data of every task
  - Brings the code executed by the task into memory
- Scheduler sets the hardware timer to interrupt at the first decision time, $t_{k=0}$
- On receipt of an interrupt at t_k:
  - Scheduler sets the timer interrupt to expire at t_{k+1}
  - If previous task overrunning, handle failure
  - If $T(t_k) = I$ and aperiodic job waiting, start aperiodic job
  - Otherwise, start next job in task $T(t_k)$ executing

k	t_k	$T(t_k)$
0	0.0	T_1
1	1.0	T_3
2	2.0	T_2
3	3.8	I
4	4.0	T_1
5	5.0	I
6	6.0	T_4
7	8.0	T_2
8	9.8	T_1
9	10.8	I
10	12.0	T_2
11	13.8	T_1
12	14.8	I
13	17.0	T_1
14	17.0	I
15	18.0	T_2
16	19.8	I

Implementation of Cyclic Scheduler



```
Input: stored schedule $(t_k, T(t_k))$ for $k = 0, 1, n - 1$.
Task SCHEDULER:
        set the next decision point $i = 0$ and table entry $k = 0$;
        set the timer to expire at $t_k$;
        do forever:
                accept timer interrupt;
                if an aperiodic job is executing, preempt the job;
                current task $T = T(t_k)$;
                increment $i$ by 1;
                compute the next table entry $k = i \bmod n$;
                set the timer to expire at $[i / n] * H + t_k$;
                if the current task $T$ is $I$,
                        let the job at the head of the aperiodic queue execute;
                else
                        let the task $T$ execute;
                sleep;
        end do.
End SCHEDULER.
```

Clock-Driven Scheduling: Advantages

- Conceptual simplicity
  - Ability to consider complex dependencies, communication delays, and resource contention among jobs when constructing the static schedule, guaranteeing absence of deadlocks and unpredictable delays
  - Entire schedule is captured in a static table
  - Different operating modes can be represented by different tables
  - No concurrency control or synchronization required
  - If completion time jitter requirements exist, can be captured in the schedule
- When workload is mostly periodic and the schedule is cyclic, timing constraints can be checked and enforced at each frame boundary
- Choice of frame size can minimize context switching and communication overheads
- Relatively easy to validate, test and certify

Clock-Driven Scheduling: Disadvantages

- Inflexible
  - Pre-compilation of knowledge into scheduling tables means that if anything changes materially, have to redo the table generation
  - Best suited for systems which are rarely modified once built
- Other disadvantages:
  - Release times of all jobs must be fixed
  - All possible combinations of periodic tasks that can execute at the same time must be known a priori, so that the combined schedule can be pre-computed
  - The treatment of aperiodic jobs is very primitive
    - Unlikely to yield acceptable response times if a significant amount of soft real-time computation exists

Fixed Priority Scheduling

- Priority of each periodic task is fixed relative to other tasks
- Liu and Layland in their seminal work "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment" dated 1973 presented an optimal fixed priority **Rate-Monotonic Algorithm**.

Rate-Monotonic Scheduling

Theorem (Rate-monotonic) [Liu and Layland, 1973]

Given a set of periodic tasks and preemptive priority scheduling, then by assigning priorities such that the tasks with shorter periods have higher priorities (rate monotonic) yields an optimal scheduling algorithm.

Theorem (RMA Bound)
Any set of *n* periodic tasks is RM-schedulable if the processor utilization, *U*, is no greater than $n\left(2^{\frac{1}{n}}-1\right)$.

This means that whenever *U* is at or below the given utilization bound, a schedule can be constructed with RM. In the limit when the number of tasks $n=\infty$, the maximum utilization limit is

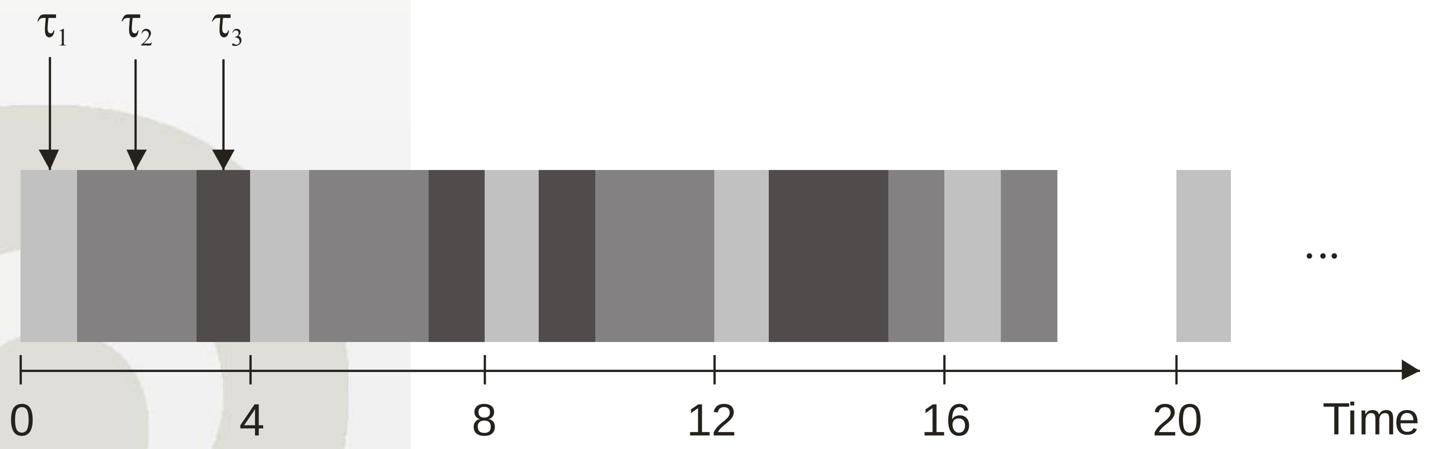
$$\lim_{n\to\infty} n\left(2^{\frac{1}{n}}-1\right)=\ln 2\approx 0.69$$

Rate-Monotonic Scheduling

To illustrate rate-monotonic scheduling, consider the task set shown

τ_i	e_i	p_i	$u_i = e_i/p_i$
τ_1	1	4	0.25
τ_2	2	5	0.4
τ_3	5	20	0.25

. All tasks are released at time 0. Since task τ_1 has the smallest period, it is the highest priority task and is scheduled first. Note that at time 4 the second instance of task τ_1 is released and it preempts the currently running task τ_3 that has the lowest priority.



Rate Monotonic Scheduling Example

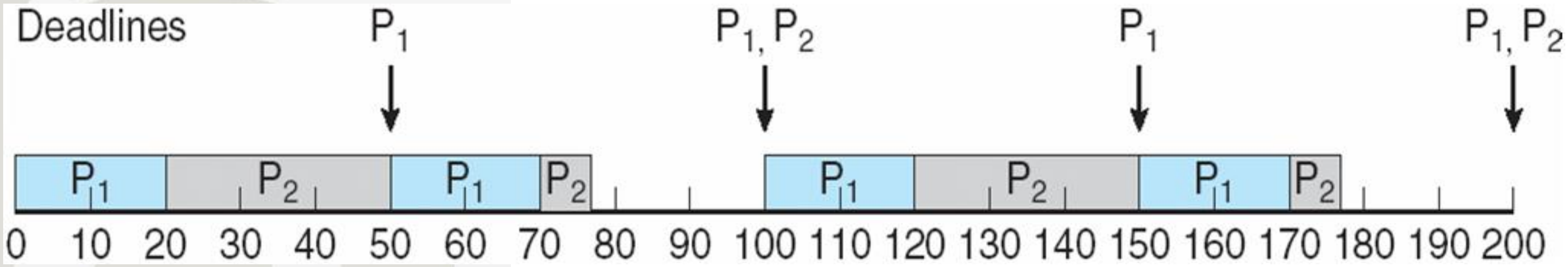
Task	Execution time, e_i	Period, p_i
P_1	20	50
P_2	35	100

- A priority is assigned based on the inverse of its period
- Shorter periods = higher priority;
- Longer periods = lower priority
- Draw the scheduling diagram
- Find CPU utilization factor

Rate Monotonic Scheduling Example

Task	Execution time, e_i	Period, p_i
P_1	20	50
P_2	35	100

- 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.
- CPU utilization U = 20/50 + 35/100 = 75%



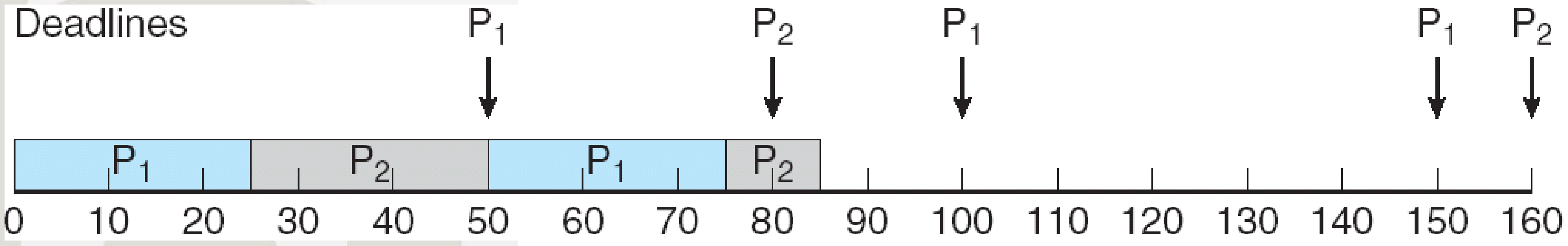
Missed Deadlines With Rate Monotonic Scheduling

Task	Execution time, e_i	Period, p_i
P_1	25	50
P_2	35	80

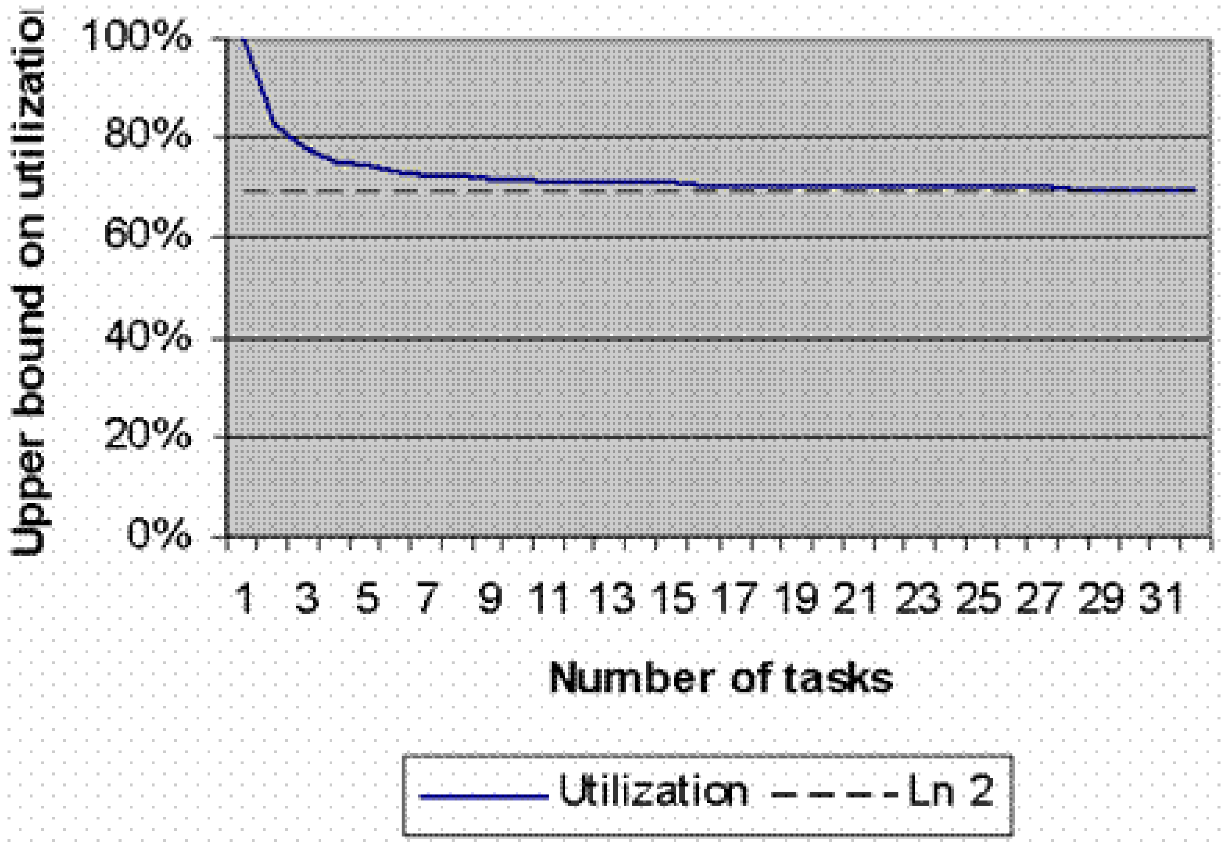
CPU utilization U = 25/50 + 35/80 = 94%

Recall RMA bound theorem

$$n = 2 \rightarrow n\left(2^{\frac{1}{n}} - 1\right) = 2\left(\sqrt{2} - 1\right) \approx 0.83 = 83\%$$



Rate-Monotonic Scheduling



Upper bound on utilization in a rate-monotonic system as a function of the number of tasks. Notice how it rapidly converges to 0.69.

Analysis of Fixed-Period Systems

- A necessary and sufficient condition for schedulability analysis is based on worst-case response time calculation

- For a general task, response time, R_i, is given as $R_i = e_i + I_i$ where e_i is max execution time and I_i is the maximum amount of delay in execution, caused by higher priority tasks.

- By solving as a recurrence relation, the response time of the nth iteration can be found as

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

- Here *hp(i)* is the set tasks of a higher priority than task *i*.

Response Time Analysis for Fixed Period Systems

When using the recurrence relation to find response times, it is necessary to compute consecutive values of R_i^{n+1} iteratively until the first value of m is found such that $R_i^{m+1} = R_i^m$. This R_i^m is then the desired response time, R_i. It is important to note that if the recursive equation does not have a solution, then the value of R_i^{n+1} will continue to grow, as in the overloaded case when a task set has a CPU utilization factor greater than 100%.

Response Time Analysis - RM Example

- Consider the task set, to be scheduled rate-monotonically

τ_i	e_i	p_i
τ_1	3	9
τ_2	4	12
τ_3	2	18

- The highest priority task ***T₁*** will have a response time equal to its execution time so, $R_1 = 3$.
- T_2 will have its response time calculated as follows:
- First let $R_2^0 = 4$, then

$$R_2^1 = 4 + \lceil 4/9 \rceil 3 = 7$$

$$R_i^{n+1} = e_i + \sum_{j \in hp(i)} \lceil R_i^n / p_j \rceil e_j$$

$$R_2^2 = 4 + \lceil 7/9 \rceil 3 = 7$$

The equality $R_2^1 = R_2^2$ implies that $R_2 = 7$.

Response Time Analysis - RM Example

- Similarly, the lowest priority task ***T*₃**'s response is derived as

$$R_3^1 = 2 + \lceil 2/9 \rceil 3 + \lceil 2/12 \rceil 4 = 9$$

$$R_3^2 = 2 + \lceil 9/9 \rceil 3 + \lceil 9/12 \rceil 4 = 9$$

As $R_3^1 = R_3^2$, the response time $R_3 = 9$.

Deadline-Monotonic Scheduling

- The *deadline monotonic* algorithm assigns task priority according to relative deadlines – the shorter the relative deadline, the higher the priority
- When relative deadline of every task matches its period, then rate monotonic and deadline monotonic give identical results
- When the relative deadlines are arbitrary:
  - Deadline monotonic can sometimes produce a feasible schedule in cases where rate monotonic cannot
  - But, rate monotonic always fails when deadline monotonic fails
- Deadline monotonic preferred to rate monotonic
  - If deadline ≠ period

Any Questions?