Real Time Systems
Unit-2

RTOS UNIT - 2

Real Time Scheduling

Common Approaches to Real Time Scheduling: Clock Driven Approach, Weighted Round Robin Approach, Priority Driven Approach, Dynamic Versus Static Systems.

Optimality of Effective-Deadline-First (EDF) and Least-Slack-Time-First (LST) Algorithms, Rate Monotonic Algorithm, Offline Versus Online Scheduling, Scheduling Aperiodic and Sporadic jobs in Priority Driven and Clock Driven Systems.

Clock-Driven Scheduling: Basics

- Decision regarding which job to run next is made only at clock interrupt instants:
  - Interval timers determine the scheduling points.

- Which task to be run when and for how long is stored in a table.

Clock-Driven Scheduling

- Round robin scheduling:
  - An example of clock-driven scheduling.

- Popularly used:
  - Basic Timer-Driven Scheduler (Table-driven)
  - Cyclic Scheduler

Clock-Driven Schedulers

- Also called:
  - Offline schedulers.
  - Static schedulers

- Used extensively in embedded applications.

Clock-Driven Scheduling

- For scheduling n periodic tasks:
  - The schedule is stored in a table.
    - Repeated forever.
  - The designer needs to develop a schedule for what period?
    - LCM(P_1,P_2,….,P_n)
    - Proof in class

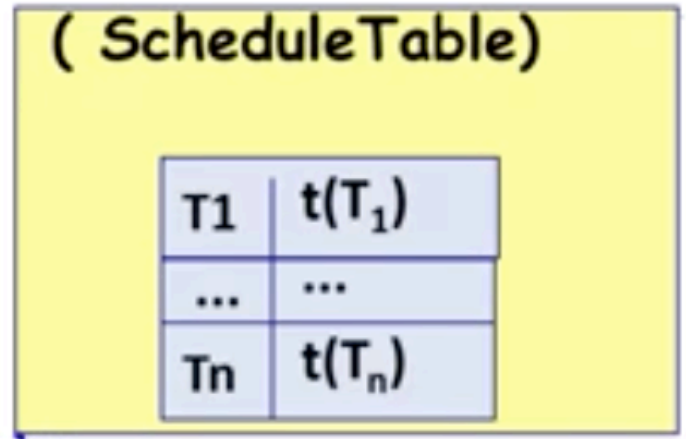
Clock-Driven Scheduling

- Used in low-cost applications:

- Pro:
  - Compact: Require very little storage space
  - Efficient: Incur very little runtime overhead.

- Con:
  - Inflexible: Very difficult to accommodate aperiodic or sporadic tasks.

Basic Table-Driven Scheduler




```
int SchedTableSize= 10;
int entry = -1;
timer_handler () {
  time_t next_time;
  current   = SchedTable[entry].task;
  entry = (entry+1) % SchedTableSize;
  next_time = Table[entry].time + gettime();
 set_timer(next_time);
  execute_task(current);
  if (gettime() < next_time)
       handle_aperiodic();
  return;
}
```

Schedule Table

Task	Start Time	Stop Time
T1	0	100
T2	101	150
T3	151	225
T2	226	300
T1	301	345

Disadvantage of Table-Driven Schedulers

- When the number of tasks are large:
  - Requires setting a timer large number of times.
  - The overhead is significant:
    - Remember that a task instance runs only for a few mili or microseconds

Cyclic Schedulers

- Cyclic schedulers are very popular:
  - Extensively being used in the industry.
  - A large majority of small embedded applications being manufactured at present use cyclic schedulers.
- The schedule is Precomputed and stored for one major cycle:
  - This schedule is repeated.
- A major cycle is divided into:
  - One or more minor cycles (frames)..
- Scheduling points for a cyclic scheduler:
  - Occur at the beginning of frames.

Cyclic Scheduler Basics

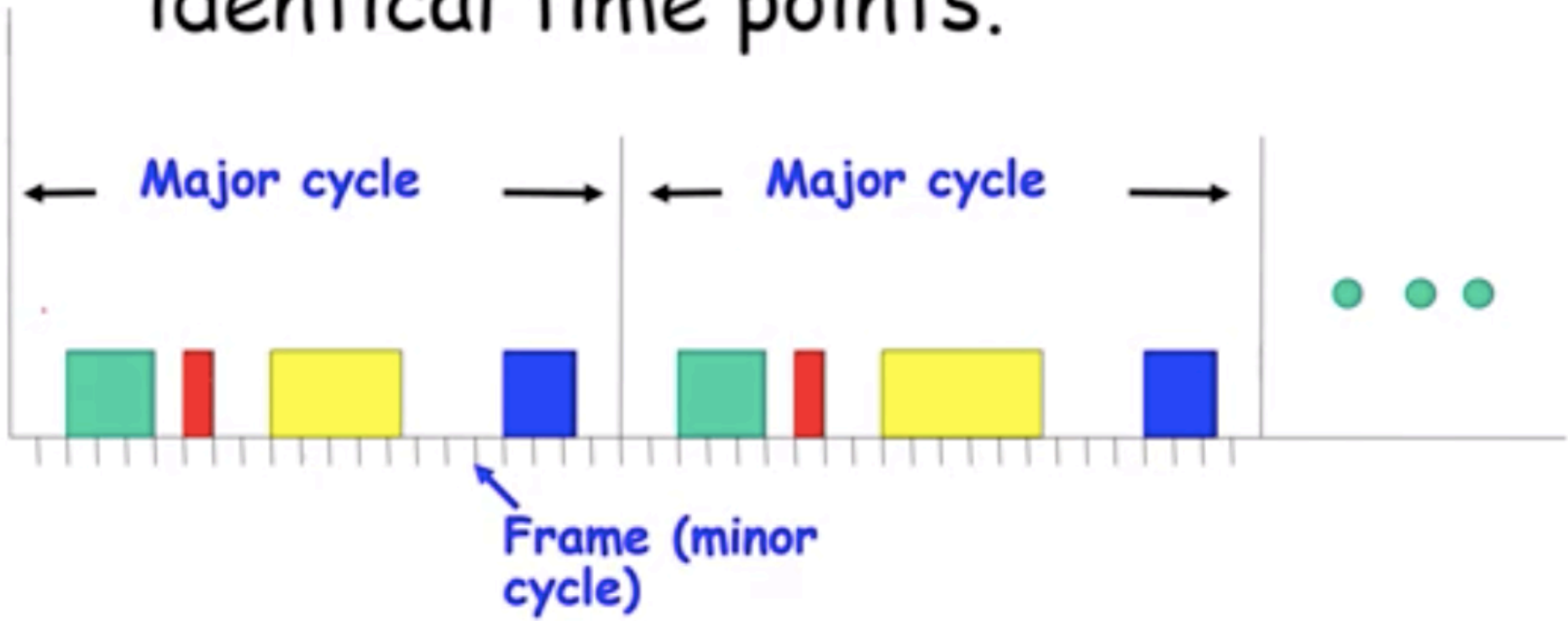
- Scheduling decisions are only made at frame boundaries.
  - Exact start and completion time of jobs within frame is not known
  - But must know the max computation time
- Jobs are allocated to specific frames
- Major cycle is also called a Hyperperiod.

Cyclic Scheduler Basics

- If a schedule can not be found for the set of predefined jobs:
  - Then these are divided into job slices.
  - Essentially, divide a job into a sequence of smaller jobs.

Major Cycle

- In each major cycle:
  - The different jobs recur at identical time points.



Major Cycle

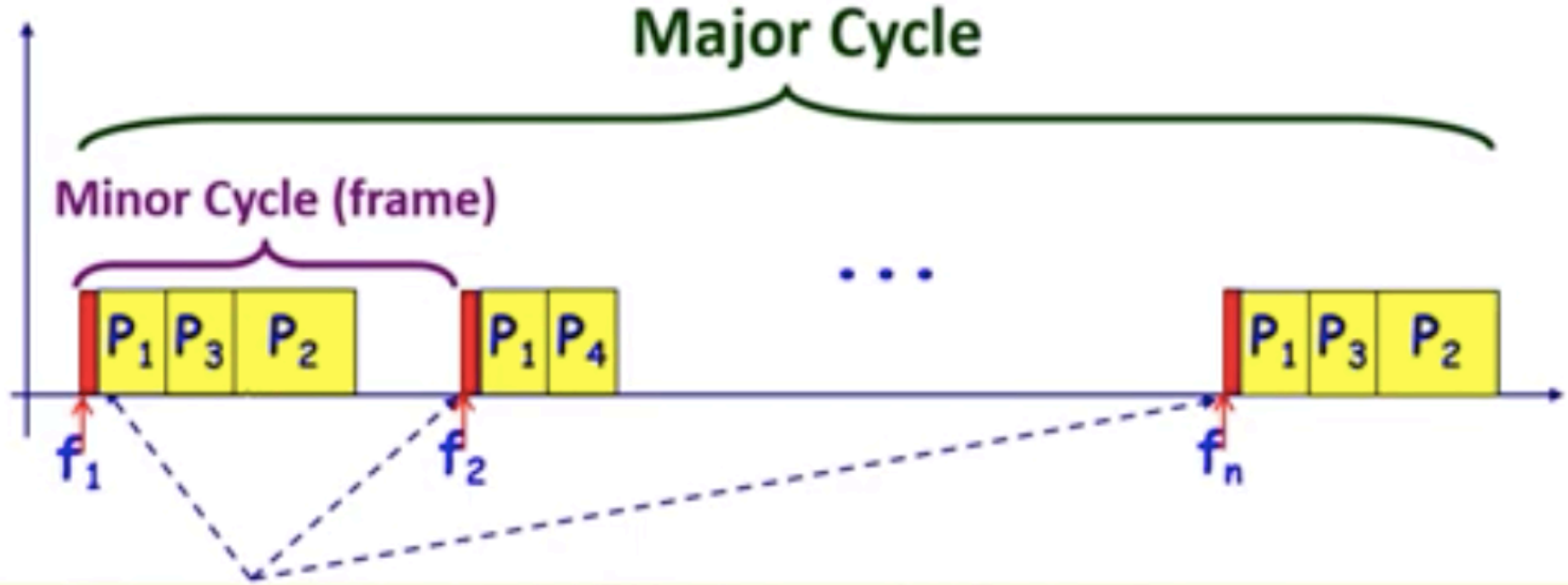
- The major cycle of a set of tasks ST={T1, T2,...,Tn} is at least:
  - LCM(p1,p2,...,pn)
- Holds even when tasks have arbitrary phasings.
- Can be greater than LCM when F does not divide major cycle.

Minor Cycle (Frame)

- Each major cycle:
  - Usually contains an integral number of minor cycles(frames).

- Frame boundaries are marked:
  - Through interrupts generated from a periodic timer

Major and Minor Cycle



- Cyclic Executive runs in response to a tick event
- Red bar shows the time to execute the scheduler

A Typical Schedule

Frame	Task
F1	T2
F2	T3
F3	T1

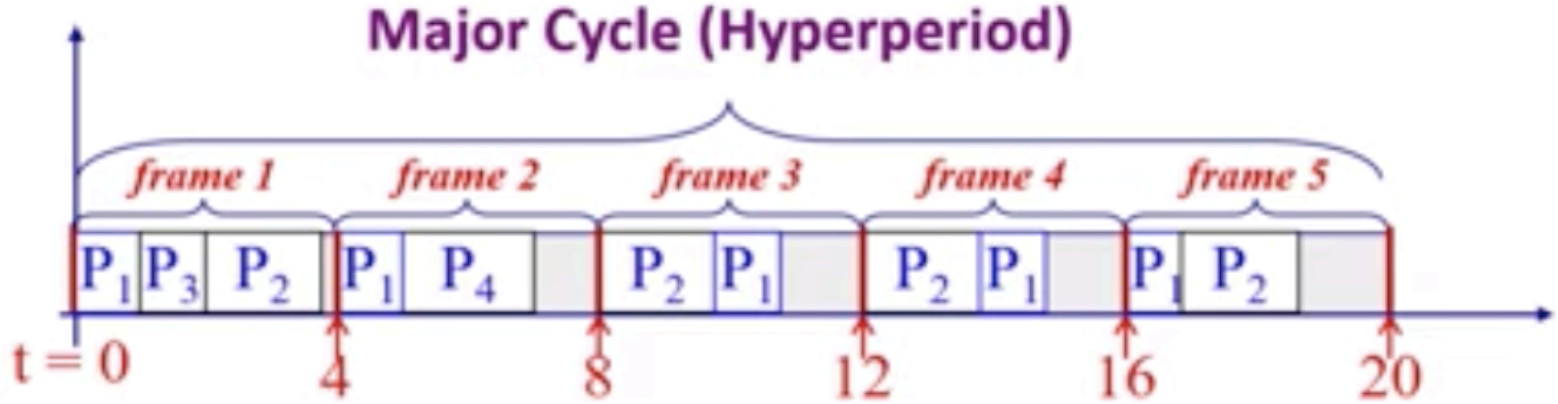
Constructing a Schedule

- Construct static schedule for a Major Cycle
- Cyclic Executive repeats this schedule
- There may be resulting idle intervals
  - if so attempt to arrange so they occur periodically

Cyclic Schedule

f_1	$tasks_1$
...	...
f_5	$tasks_5$

Task	=	(r, e)
T_1	=	(4, 1)
T_2	=	(5, 1.8)
T_3	=	(20, 1)
T_4	=	(20, 2)
H	=	20



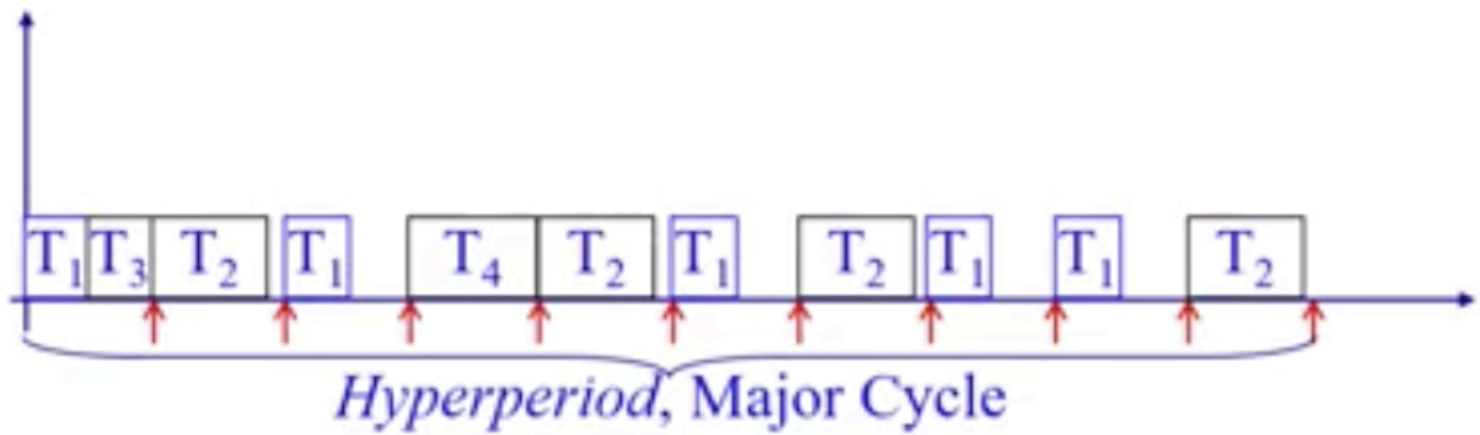
Cyclic Executive

```
int MajorCycle = 10;
int MinorCycle = 1;
int frame=-1;

// tick handler,
// called at start of frame
tick_handler() {
  // ensure previous frame completed
  if (previous task running)
    handle_overrun(); // error

  // determine frame and schedule
  frame      = (frame + 1) % MajorCycle;
  schedule   = CyclicTable[frame].sched;

  // sequentially run each job in schedule
  execute_schedule(schedule);

  // done with this frame
  return;
}
```

```
execute_schedule (schedule) {
   job_t job;
   for each job in schedule
   execute job
   return
}
```

Partitioning A Major Cycle into Frames

- Designer:
  1. choose frame size,
  2. partition jobs into slices (if needed),
  3. place jobs/slices into frames.
- At Frames boundaries :
  - Cyclic executive performs scheduling
- There is no preemption within a frame



Cyclic Executive Example

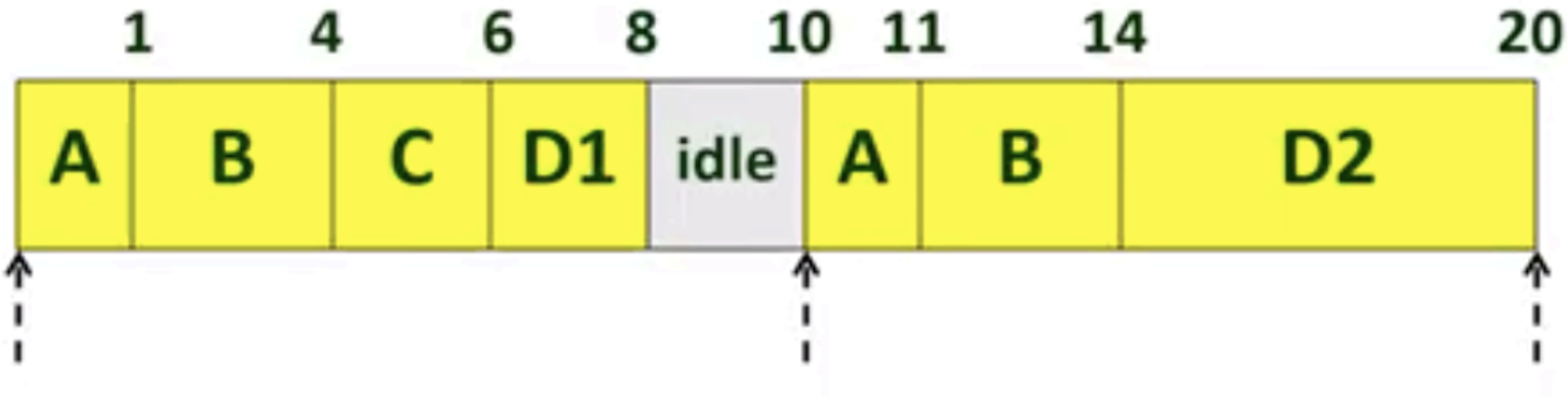
- Job=(computation time, period, relative deadline)
  - A = (1, 10, 10)
  - B = (3, 10, 10)
  - C = (2,20,20)
  - (8, 20, 20)

- Two slices

$$c(D1) = 2,\ c(D2) = 6$$

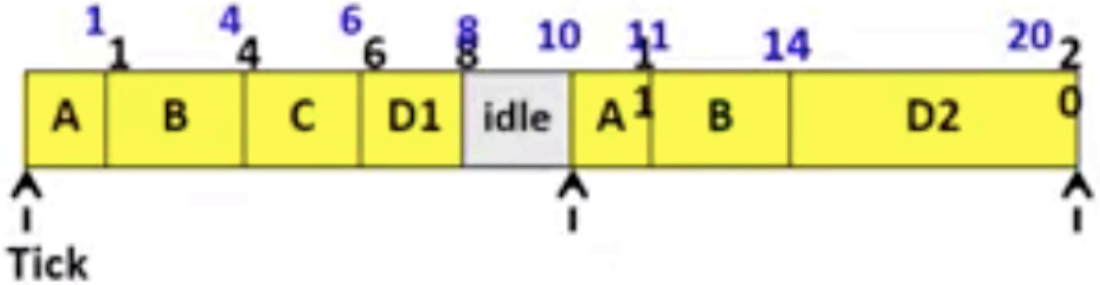
- Major Cycle = 20msec.
- We select frame size = 10 msec

Schedule Example



- Could we create a different schedule?
- Is it better to distribute the idle time?

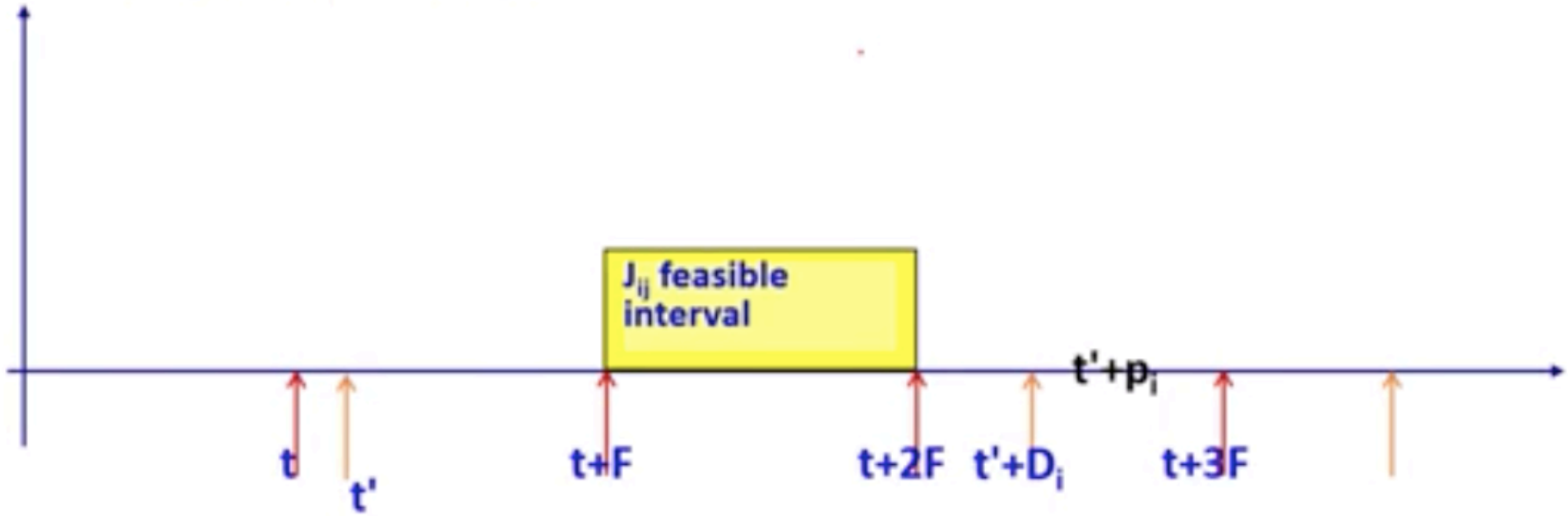
Schedule Example




```
mct = 10;
next_time = Clock() + mct;
Frame_no = 1;
loop
        Frame_no = (Frame_no+1) mod 2;
        if Frame_no = 0 → A ; B ; C ; D1;
        else → A ; B ; D2;
        if Clock < next_time → wait (next_time – Clock);
        else if Clock > next_time → Handle Frame Overrun;
end Loop
```

Frame Size Constraints

1. Every job needs to start and complete within a frame
   - $F \geq \max(Te_i), 1 \leq i \leq n$
2. Frame size F divides H (the Hyperperiod)
3. Between the release time and deadline of every job there is at least one frame



Minor Cycle (Frame)

- Each task is assigned to run in one or more frames.
- Frame size (F) is an important design parameter while using a cyclic scheduler.
- A selected frame size has to satisfy a few constraints.

Selecting an Appropriate Frame Size (F)

- Minimum scheduling overhead and chances of inconsistency:
  - F should be larger than each task size.
  - Sets a lower bound.
- Minimization of table size:
  - F should squarely divide major cycle.
  - Allows only a few discrete frame sizes
- Satisfaction of task deadline:
  - Between the arrival of a task and its deadline:
    - At least one full frame must exist.
  - Sets an upper bound

Minimize Scheduler Overhead

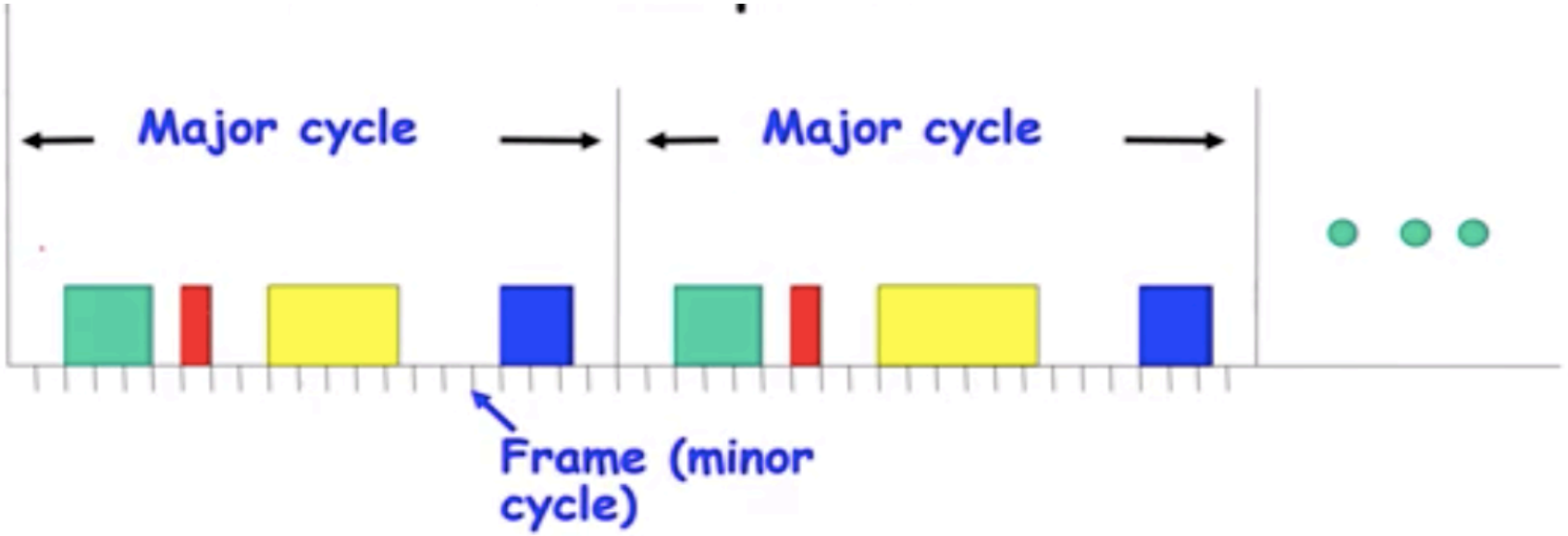
- A task instance must complete running within its assigned frame:
  - The scheduler would otherwise get invoked many times.

Minimize Inconsistency

- Unless a job runs to completion:
  - Its partial results might be used by other jobs, leading to inconsistency
- To avoid scheduler overhead:
  - Selected frame size should be larger than execution time of each task.
  - Sets a lower bound for frame size.

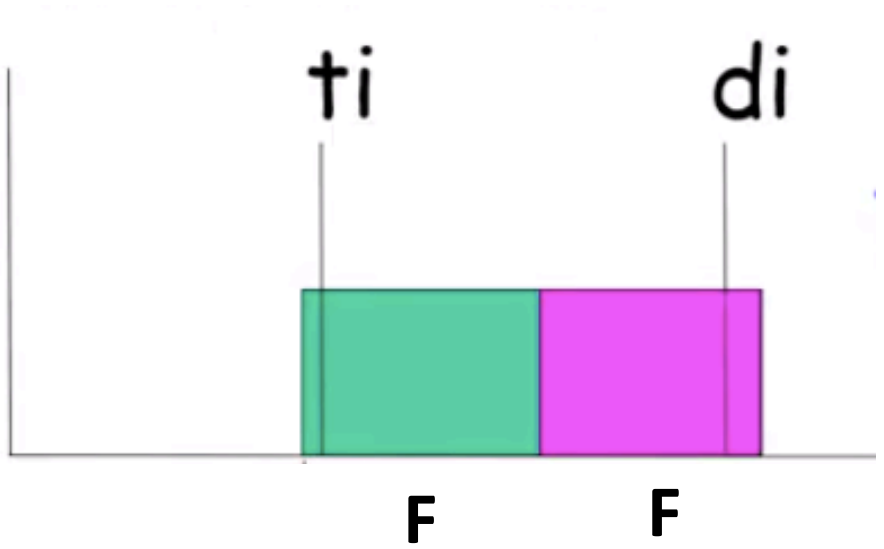
Minimization of Table Size

- Unless the minor cycle squarely divides the major cycle:
  - Storing schedule for one major cycle would not be sufficient.
  - Schedules in the major cycle would not repeat:
  - This would make the size of the table large.



Satisfaction of Task Deadline

- Between the arrival of a task and its deadline:
  - At least one full frame must exist.
- If there is not even a single frame:
  - The task would miss its deadline,
  - By the time it could be taken up for scheduling, the deadline could be imminent.

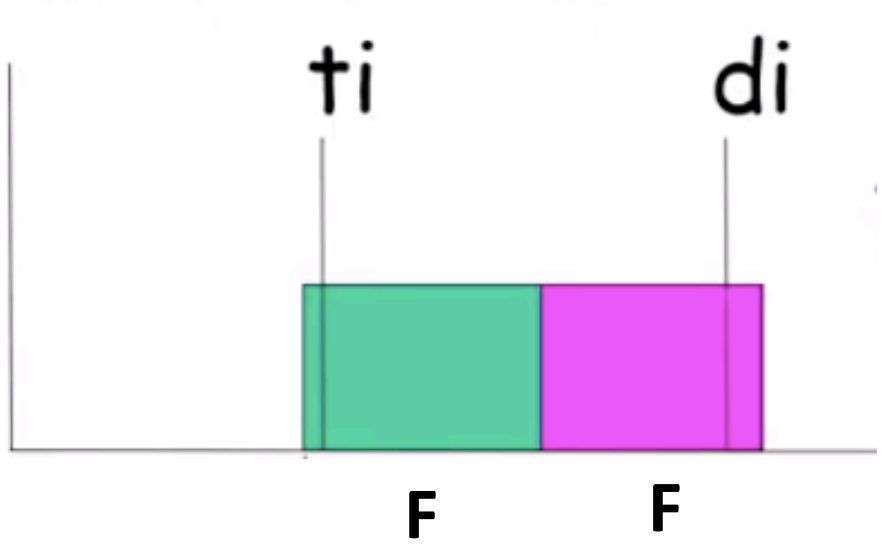


Satisfaction of Task Deadline

- The worst case for a task occurs when the task arrives just after a frame has started.

Satisfaction of Task Deadline

- The minimum separation of an arrival time for ti from a framestart:-
  - GCD(F,pi)



- Thus, for all ti
  - 2F-GCD(F,pi) < di must be satisfied

Frame Size Constraints

- $F \geq \max(e;)$ for all I
- $2F - \gcd(p_1, F) \leq Di$
- F divides the major cycle

Selection of a Suitable Frame Size

- Several frame sizes may satisfy the constraints:
  - Plausible frames.
- A plausible frame size has been found:
  - Does not mean that the task set is schedulable

What If No Frame Size is Suitable?

- Possible culprit is a task with large execution time e.g. (20,100,100):
  - Prevents a smaller frame from being chosen.
  - Try splitting task with large execution time into two or three sub- tasks.
  - Example: T1=(20,100,100) can be split into (10,100,100) and (10,100,100)

Response Times of Aperiodic Tasks

- Aperiodic tasks run when the processor is not being used by periodic tasks.
- No advantage in completing periodic tasks early.
- Try to minimize the response time of aperiodic tasks.

Slack Stealing

- Originally proposed for Priority-Driven systems.
- Used in Clock Driven scheduling:
  - Run aperiodic tasks before periodic ones where possible.

Foreground/Background

- Scheduling aperiodic jobs with periodic jobs
  - Foreground jobs => periodic jobs
  - background jobs => aperiodic jobs

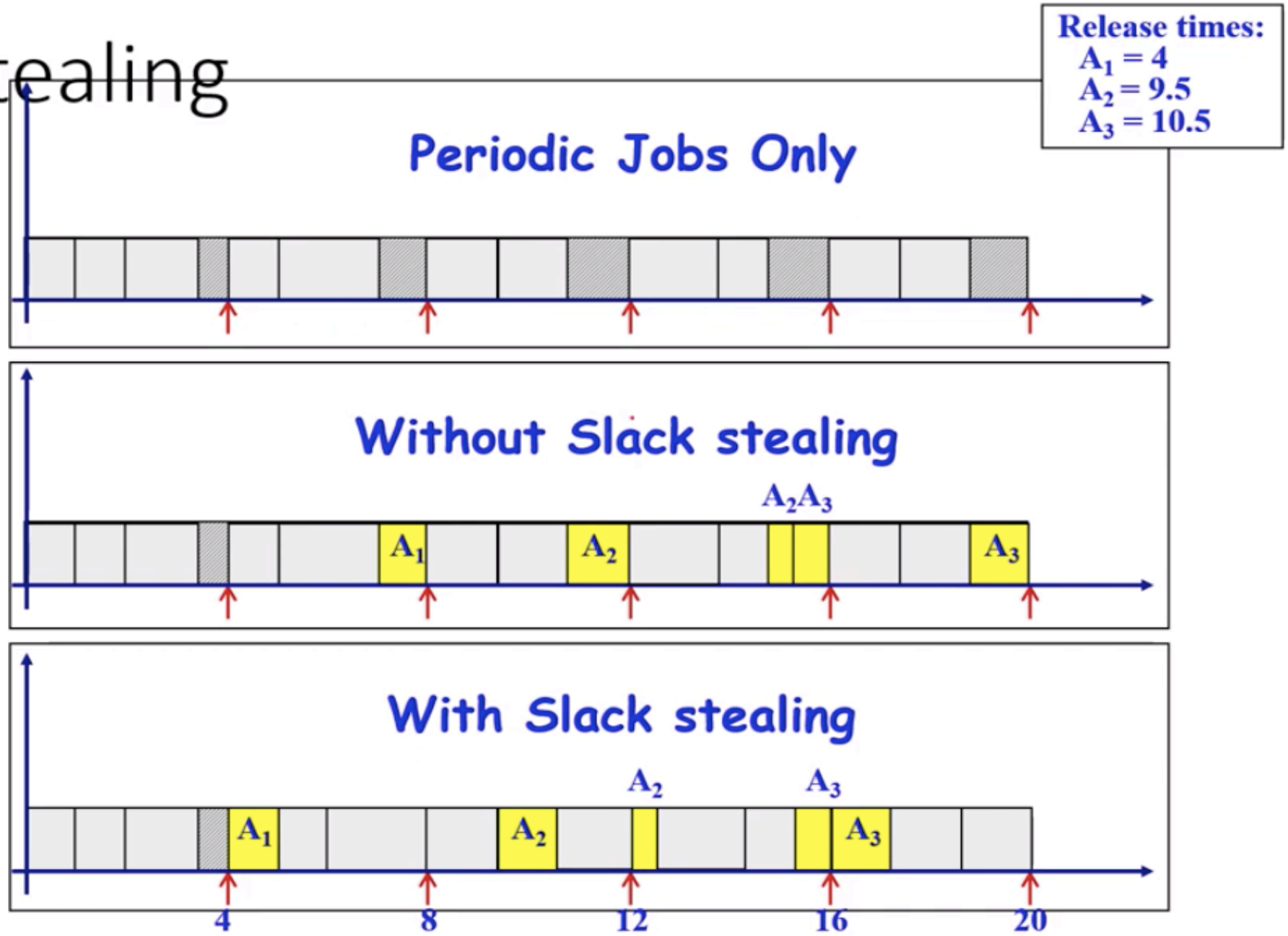
- Aperiodic jobs
  - executed within the idle blocks.
  - Aperiodic jobs are preemptable

- Optimize response time by slack stealing.

Slack Stealing

- Let :
- X_k be the amount of time used by slices in frame k
- The slack available in k is F -x_k at the start of the frame
- If y units of slack time are used the available slack reduces to F - X_k-Y.
- Checks and run aperiodic tasks after each slice completes if slack time remains.

Slack Stealing

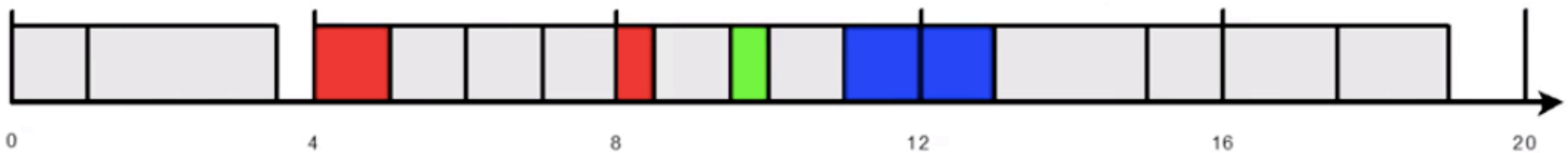


Example 2

Cyclic Schedule

Aperiodic Tasks

No Slack Stealing



With Slack Stealing

Handling Frame Overruns

- An overrun may occur on unusual input data, hardware fault, logic error
  - Job execution time exceeds maximum execution time
  - Abort the overrun job and report the premature termination of the job
  - Unfinished portion may execute as aperiodic job in a later frames
    - But this may cause other jobs to be late, too!

Mode Changes

- Mode changes result in change of schedule
  - Processes may be added or removed
    - Eg: switching from iPod to Calling mode
- Computation time may be adjusted
  - Eg: low power vs. normal mode
- When to change schedule
  - Immediately by interrupting the action
  - After completion of current action
  - After completion of current frame
  - After completion of current major cycle

Pros of Cyclic Schedulers

- Simple and efficient
- Complex requirements can be handled,such as:
  - Precedence constraints
  - Minimization of jitter or context switches

Cons of Cyclic Schedulers

- As number of tasks increases:
  - It becomes very difficult to select a suitable frame size.
- Results in suboptimal schedules.
  - CPU time wasted in many frames.
  - Results in poor schedulability.

Cons of Cyclic Schedulers

- Inflexible, difficult to modify and maintain
- Fragile:
  - Overrun may cause system to fail
- Difficult to handle sporadic and aperiodic tasks.

A Generalized Task Scheduler

- Many practical systems:
  - Consist of a mixture of periodic, aperiodic, and sporadic tasks.
  - How can sporadic and aperiodic tasks be scheduled using a cyclic scheduler?
- Use the slack time and the unused frames:
  - As and when the spordic and aperiodic tasks arise.

Cyclic Scheduler : Pseudocode

```
cyclic-scheduler() {
        current-task T = SchedTab[k]
        k = k + 1;
        k = k mod N;
        dispatch-current-Task(T);
        schedule-sporadic-tasks();
        schedule-aperiodic-tasks()
        idle(),
}
```

Preemptive vs. Non-preemptive Scheduling Algorithms

- Preemptive Scheduling:
  - Event driven.
    - Each event causes interruption of running tasks.
    - Choice of running tasks reconsidered after each interruption.
  - Benefits
    - Can minimize response time to events.
  - Disadvantages:
    - Requires considerable computational resources for scheduling

Non-preemptive Scheduling Algorithm

- Tasks remain active till completion
  - Scheduling decisions only made after task completion.
- Benefits:
  - Reasonable when
    - task execution times ~= task switching times.
  - Less computational resources needed for scheduling
- Disadvantages:
  - Can leads to starvation (not met the deadline) especially for those real time tasks (or high priority tasks).

Event-Driven Schedulers

- Compared to clock-driven scheduler
  - More proficient
  - Can handle handle sporadic and a periodic task
  - Used in more complex applications
- Scheduling Point
  - Defined by task arrival and completion events
- Preemptive scheduling
  - On arrival of higher priority task, the running task may be preempted
- Simplest event driven scheduler
  - Fore ground-back ground scheduler

Scheduling Point for EDS

- Scheduling decision are made when certain events occur:
  - A task become ready
  - A task completed

Time-sliced Round Robin Scheduling: A Hybrid Scheduler

- Time-sliced round robin schedulers:
  - Commonly used in traditional operating system
  - We shall not discuss much about it
- Scheduling Points
  - Task completing or suspending
  - Clock interrupts
- Less proficient than even cyclic schedulers:
  - Task deadlines are ignored

Event-Driven Schedulers

- Preemptive Schedulers:
  - When a higher priority task becomes ready any executing lower priority task is preempted
- Greedy Schedulers:
  - Never keep the processor idle if a task is ready

Preemptive Scheduling

- A simplifying Assumption:
  - Independent tasks execute on a uniprocessor
  - Two Algorithms pretty much summaries all important results
    - EDF: Earliest Deadline First
    - RMA: Rate Monotonic Algorithm

Foreground-Background Scheduler

- Real-time tasks are run as foreground tasks.
  - Sporadic, aperiodic, and non-real-time tasks are run as background tasks.
- Among the foreground tasks, at every scheduling point:
  - The highest priority foreground task is scheduled.
- A background task can run:
  - When no foreground task is ready

Background Task Completion Time

- Let T_B be the only background task
  - Its processing time be e_B
  - The time for it to complete would be:

$$ct_B = \frac{e_B}{1 - \sum \frac{e_i}{p_i}}$$

Example 1

- Consider a real-time system:
  - Tasks are scheduled using foreground-background scheduling.
- There is only one periodic foreground task:
  - P1=100 msec, e1=50 msec, d1=100 msec
- Background task TB, eB=1020msec.
- Compute the completion time for the background task.

Event-Driven Static Priority Schedulers

- The task priorities once assigned by the programmer:
  - Do not change during runtime.
  - RMA (Rate Monotonic Algorithm) is the optimal static priority scheduling algorithm.

Event-Driven Dynamic Schedulers

- The task priorities can change during runtime:
  - Based on the relative urgency of completion of tasks.
  - EDF (Earliest Deadline First) is the optimal uniprocessor scheduling algorithm.

Event-Driven Schedulers

- First let us consider the simplest scenario:
  - Uniprocessor
  - Independent tasks:
    - Tasks do not share resources
    - There is no precedence ordering among the tasks

EDF

- EDF is the optimal uniprocessor scheduling algorithm:
- If EDF cannot feasibly schedule a set of tasks:
  - Then, there can exist no other scheduling algorithm to do that.
- Can schedule both periodic and aperiodic tasks.

EDF

- At any scheduling point:
  - The scheduler dispatches the task having the shortest deadline among all ready tasks.
  - Preempts any task (with longer deadline) that might be running.

EDF Schedulability Check

- Sum of utilizations due to all tasks is less than one:

$$\sum_{i=1}^{n} \frac{e_i}{p_i} = \sum u_i \leq 1$$

- Both the necessary and sufficient condition for schedulability.

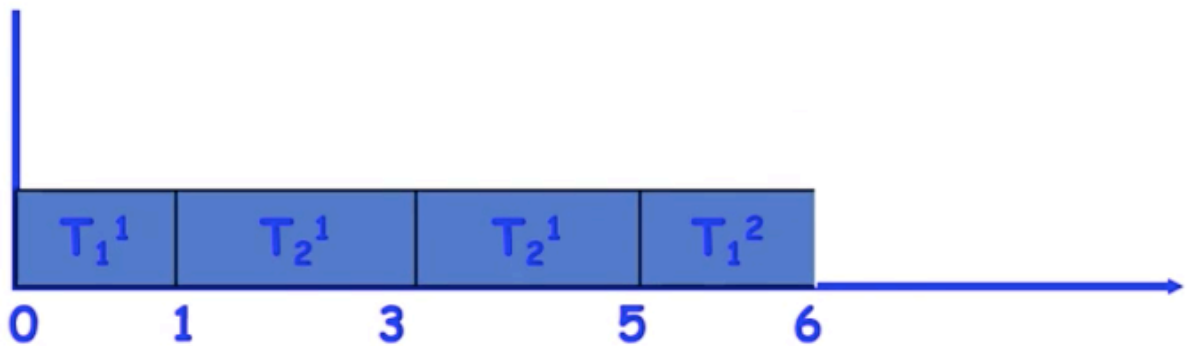
EDF Scheduler – Example 1

Task set: $T_i = (c_i, p_i, d_i)$

T1 = (1,3,3) and T2 = (4,6,6)

Schedulability check:

1/3 + 4/6 = 0.33 + 0.67 = 1.0



Unlike RMS, Only those task sets which pass the schedulability test are schedulable under EDF

Example 2

- Compute a suitable frame size of the following task set:
  - e1=1, p1=4, d1=4
  - e2=1, p2=5, d2=5
  - e3=5, p3=20, d3=20

Dynamic Priority Scheduling Algorithm

- Is EDF Really a Dynamic Priority Scheduling Algorithm?
- If EDF were a dynamic priority algorithm:
  - At any point of time, the priority value of a task can be determined. dynamic priorities
  - Also we should be able to show how it changes with time.

Dynamic Priority

- The longer a task waits in ready queue:
  - The higher the chance (probability) of its being taken up for scheduling.
- We can imagine a virtual priority value associated with a task:
  - Keeps increasing with time until the task is taken up for scheduling

EDF: An Evaluation

- EDF is:
  - Simple
  - Optimal
- But, is rarely used:
  - No commercial operating system directly supports EDF scheduling.
  - Why?
    - Let us examine the disadvantages of EDF.

Disadvantages of EDF

- Main disadvantages of EDF:
  - Poor transient overload handling
  - Runtime inefficiency
  - Poor support for resource sharing among tasks.

Poor Transient Overload Handling

- Why does transient overload occur?
  - A task might take more time than estimated.
  - Too many tasks might arise on some event.
- When an executing task takes more time:
  - It is extremely difficult to predict which task would miss its deadline.

Runtime Inefficiency

- EDF is not very efficient:
  - In an implementation of EDF:
    - The tasks need to be maintained sorted.
    - A priority queue based on task deadlines is required.
- The complexity of maintaining a priority queue is:
  - log(n), n is the number of tasks.

Poor Resource Sharing Support

- Support for tasks to share non- premptable resources (critical sections):
  - Extremely inefficient.
  - Tasks often miss their deadlines.
  - We shall elaborate this problem later in our discussions.

General EDF Schedulability

- When task deadlines differ from task periods:
- The schedulability criterion needs to be generalized:

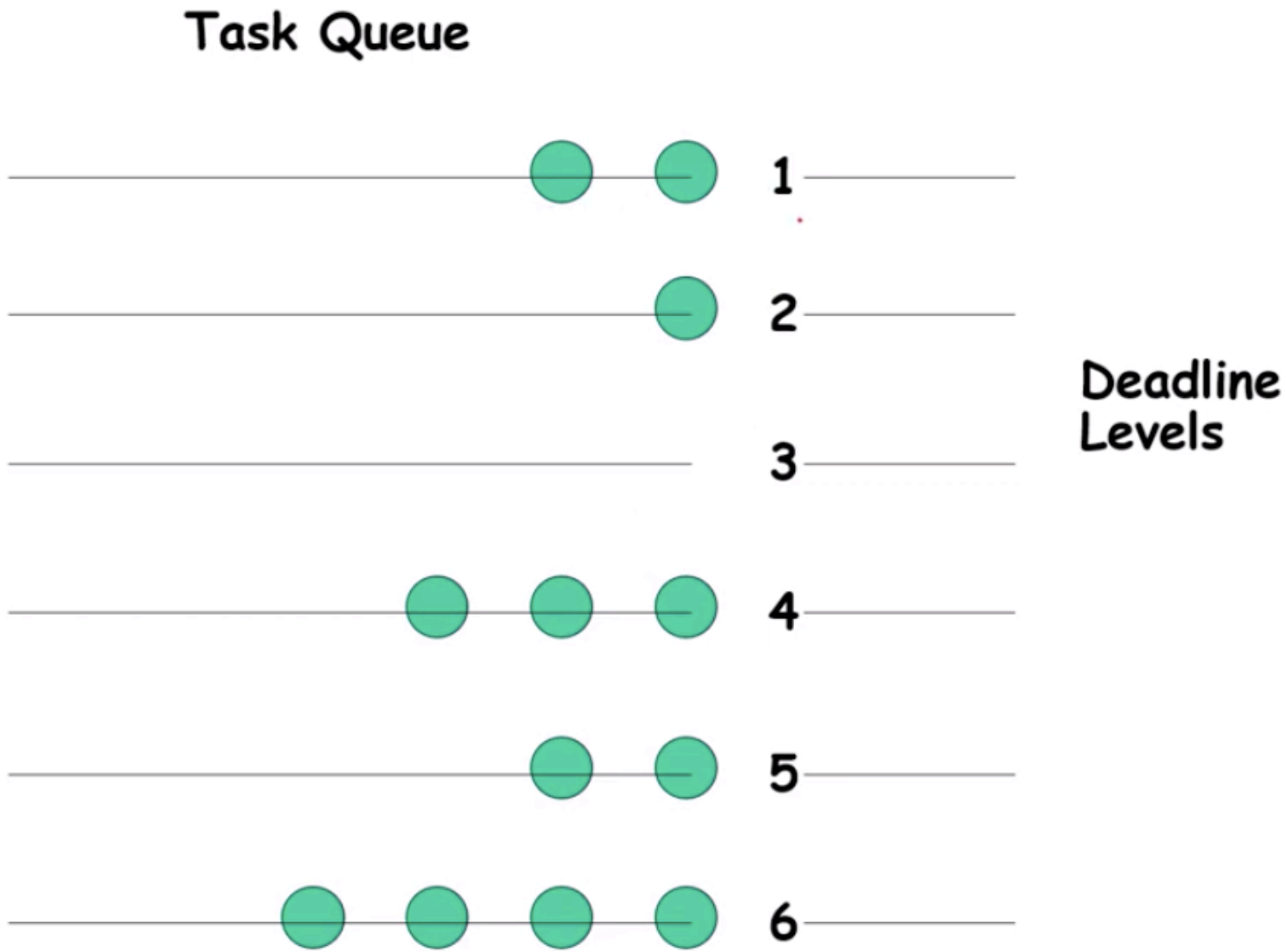
$$\sum_{i=1}^{n} \frac{e_i}{\min(p_i, d_i)} \leq 1$$

- If pidi, it becomes sufficient:
  - But not a necessary condition

An Efficient Implementation of EDF

- Maximum number of distinct deadlines is fixed.
- A queue is maintained for each distinct deadline.
- When a task arrives:
  - Its absolute deadline is computed and inserted in Q.

Efficient EDF Implementation



MLF: Variation of EDF

- Minimum Laxity First:
  - Laxity= relative deadline - time required to complete task execution
  - The task that is most likely to fail first is assigned highest priority.
  - EDF=MLF when tasks have equal execution times and deadlines.

Static and Dynamic System

- Static System:
  - Output of the system depends only on the present values of input
    - Ex- $y(t)=f\{x(t)\}$
- Dynamic System:
  - Output of the system depends on past or future values of input at any instance of time
    - Ex- $y(t)=f\{x(t)*x(t-1)\}$

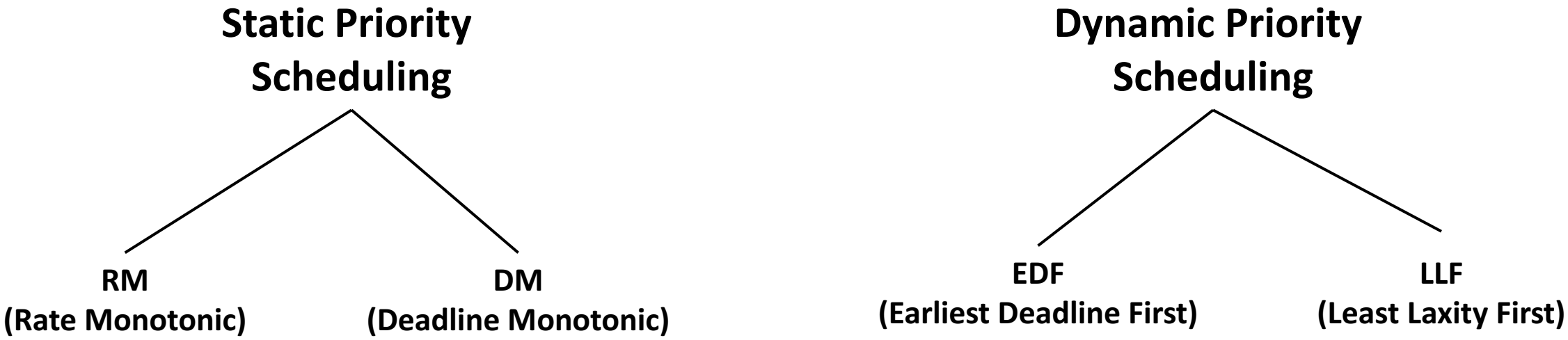
Static Scheduling

- All Scheduling decisions at compile time
- Used to determine schedule offline
- All jobs, their arrival, execution, deadline known in advance
- Create a schedule and execute it
- Cannot change dynamically
- They are clock driven
- Example: Round Robin Scheduling

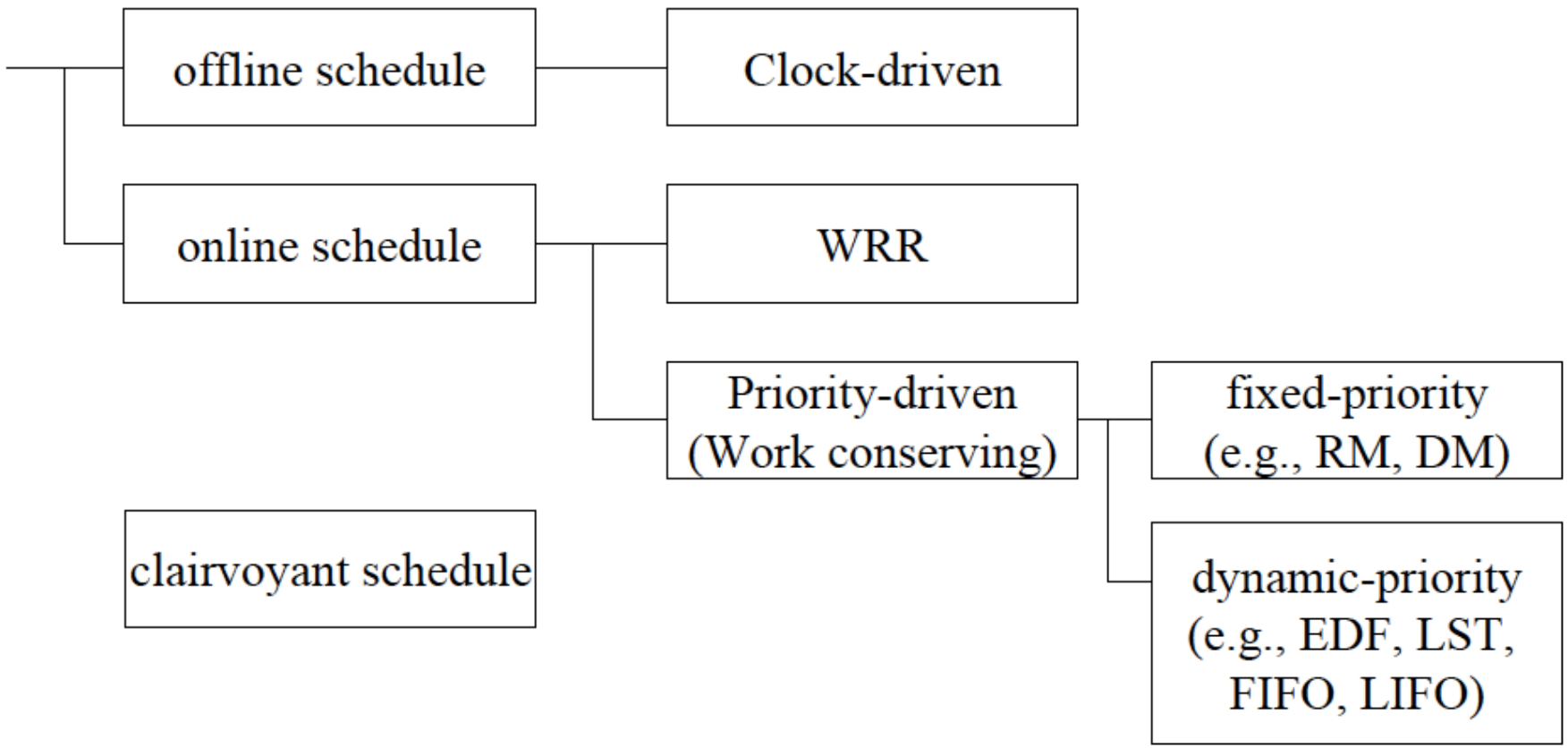
Dynamic Scheduling

- All Scheduling decisions at run time.
- Used to determine schedule online
- Priorities assign statically as well as dynamically to the process
- Not predictable(difficult to predict)
- Accommodate all dynamic changes
- Priority driven and quality driven; In priority driven it either have static or dynamic priority

Dynamic Scheduling



Classification of Scheduling Algorithm



Round Robin(Revision)

- Preemptive
- Arrival time, Burst Time, Quantum time
- Gantt chart, Queue
- Turn around time, Weight time, Throughput

Round Robin(Revision)

- Jobs join a FIFO queue. The job at the head of the queue executes for at most one time slice.
- If the job does not complete by the end of the time slice, it is preempted and placed at the end of the queue.
- Because the length of the slice is relatively short (typically ~10^{-3} – 10^{-2} sec), the execution of every job begins almost immediately after it becomes ready.
- Each job get $1/n^{th}$ share of the process – processor-sharing algorithm

Weighted Round Robin

- Weighted round-robin algorithm: Rather than giving all the ready jobs equal shares of the processor, different jobs may be given different weights.
- The weight of a job refers to the fraction of processor time allocated to the job. A job with *wt* get *w* time slices in every round.
- (Weighted) round-robin scheduling is a reasonable approach for a *pipe*, since a successor job may be able to incrementally consume what is produced by a predecessor, and the two jobs can execute concurrently.

Priority Driven Scheduling

- Priority-driven algorithms are event-driven: Scheduling decisions are made when events such as releases and completions of jobs occur.
- Priority-driven algorithms are locally greedy: The algorithms never leave any resource idle intentionally.
- Jobs ready for execution are placed in one or more queues ordered by the priorities. The priority list and rules such as whether preemption is allowed define the scheduling algorithm.
- Usually, preemptive scheduling with the ability of job migration among processors is better than non-preemptive scheduling.

Priority Driven Approach

- A fixed-priority algorithm assigns the same priority to all the jobs in each task.

- A dynamic-priority algorithm assigns different priorities to the individual jobs in each task. By dynamic, we mean task-level dynamic and job level fixed.

- Most real-time scheduling algorithms of practical interests assign job-level fixed priorities.

Priority Driven Approach

- Fixed Priority
  - The priority of the task(w.r.t. other tasks) remains same during execution
  - Example: Priority(T_1) < Priority(T_2), always true

- Dynamic Priority
  - The priority of the task(w.r.t. other tasks) may change during execution
  - Example: Priority(T_3) < Priority(T_2) < Priority(T_1), may change to Priority(T_3) < Priority(T_1) < Priority(T_2)

Assigning Priority – EDF

- A way to assign priorities to jobs is on the basis of their deadlines.
- The earlier the deadline, the higher the priority.
- The priority-driven scheduling algorithm based on this priority assignment is called the Earliest-Deadline-First (EDF) algorithm.
- EDF algorithm is optimal when used to schedule jobs on a processor as long as preemption is allowed and jobs do not contend for resources.

Assigning Priority – LST

- The Least-Slack-Time-First (LST) algorithm assigns priorities to jobs based on their slacks: the smaller the slack, the higher the priority.
- At any time t, the slack (or laxity) of a job with deadline at d is equal to d-t minus the time required to complete the remaining portion of the job.
- LST algorithm is also optimal for scheduling preemptive jobs on one processor.
- Note: The EDF and the LST algorithms are optimal only when preemption is allowed.

Least Slack Time

- Slack Time: Time span between the completion of a request and its deadline is referred to as the slack time of the request.
- Lower the slack time higher the priority.
- (D,t,e’):
  - D: Deadline of a task
  - t : Real time when the cycle start
  - e’ : Remaining execution time of task
- Example: Problem solved in class

Advantages of Priority-Driven Scheduling

- Priority-driven scheduling is easy to implement. It does not require the information on the release times and execution times of the jobs a priori.

- The run-time overhead due to maintaining a priority queue of ready jobs can be made small.

Disadvantages of Priority-Driven Scheduling

- However, the timing behavior of a priority-driven system is nondeterministic.

- It is difficult to validate that all jobs scheduled in a priority-driven manner meet their deadlines when the job parameters vary.

Rate Monotonic Algorithm

- RMA is a static priority (event driven) algorithm and is extensively used in practical applications. RMA assigns priorities based on their rate of occurrence:
  - The lower the occurrence rate of a task , the lower the priority assigned to it.
  - A task having the highest occurrence rate (lowest period) is accorded the highest priority.
- RMA has been proven to be the optimal static priority real-time task scheduling algorithm.
- In RMA the priority of a task Ti is computed as: priority = k/pi, where pi is the period of a task Ti and k is a constant.

Rate Monotonic Algorithm

- RMA is a static priority (event driven) algorithm and is extensively used in practical applications. RMA assigns priorities based on their rate of occurrence:
  - The lower the occurrence rate of a task , the lower the priority assigned to it.
  - A task having the highest occurrence rate (lowest period) is accorded the highest priority.
- RMA has been proven to be the optimal static priority real-time task scheduling algorithm.
- In RMA the priority of a task Ti is computed as: priority = k/pi, where pi is the period of a task Ti and k is a constant.

SCHEDULABILITY TEST FOR RMA

- Schedulability of a task set under RMA can be determined from a knowledge of the worst-case execution times and periods of the tasks.
- QUESTION: how can a system developer determine the worst-case execution time of a task even before the system is developed?
- The worst-case execution times are usually determined experimentally or through simulation.

SCHEDULABILITY TEST FOR RMA

- Necessary condition: a set of periodic real-time tasks would not be schedulable under RMA unless they satisfy the following necessary condition:

$$\sum_{i=1}^{n} \frac{e_i}{p_i} = \sum u_i \leq 1$$

ADVANTAGES OF RMA

- Unlike EDF, RMA requires very few special data structures
- Most commercial real-time operating systems support real-time (static) priority levels for tasks. Tasks having real-time priority levels are arranged in multilevel feedback queues
  - Among tasks in a single level, the RTOS provide an option of either round-robin or FIFO scheduling
- RMA possesses good transient overload handling capability
  - Will a delay in completion by a lower priority task affect a higher priority task? --- NO --- a lower priority task even when it exceeds its planned execution time, cannot make a higher priority task wait -- according to the basic principles of RMA a high priority task preempts any lower priority task => execution overshooting in lower priority tasks can not make a high priority task miss its deadline

Disadvantages of RMA

- It is very difficult to support scheduling of aperiodic and sporadic tasks under RMA; RMA is not optimal when task periods and deadlines differ.
- RMA no longer remains an optimal scheduling algorithm for periodic real-time tasks, when pi and di of tasks differ.
- For such task sets DMA is more proficient than RMA

DMA

- DMA is a variant of RMA and assigns priorities to tasks based on their relative deadlines, rather than assigning priorities based on task periods.
  - DMA assigns higher priorities to tasks with shorter relative deadlines
  - When the relative deadline of every task is proportional to its period, RMA and DMA produce identical solutions
  - When the relative deadlines are arbitrary, DMA is more proficient than RMA in that it can sometimes produce a feasible schedule when RMA fails -- on the other hand RMA always fails when DMA fails.

Unit 2 finished here.
For Queries contact at
saurabh.mishra@kiet.edu