Lecture Note #4: Task Scheduling (1)
EECS 571
Principles of Real-Time Embedded Systems

Kang G. Shin
EECS Department
University of Michigan

Reading Assignment

- Liu and Layland's paper
- Chapter 3 of the text
- HW#2 has already been posted.

Main Question

Will my real-time application really meet its timing constraints/requirements?

- **Task Assignment and Scheduling: Given a set of tasks, precedence constraints, resource requirements, their execution times, release times, and deadlines, and a processing system, design a feasible/optimal allocation/scheduling of tasks on the processing system.**
- **Terminologies: feasibility, optimality, lateness, absolute/relative/effective deadlines, absolute/effective release times.**

Components of Task Assigment & Scheduling

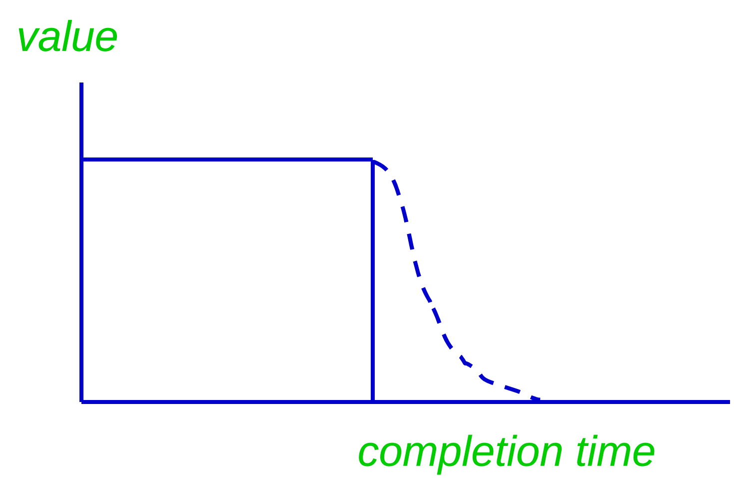
- **Precedence relation $\prec$ (T) = the set of tasks that must be completed before task T can begin its execution**

- **Resource requirements: processor, memory, bus, disk,...**
  - Exclusive
  - Shared (read-only, read-write)

- **Schedule *S*: set of processors $\times$ time $\rightarrow$ set of tasks**
  - off-line or online
  - static or dynamic priority alg
  - preemptive or nonpreemtive
  - uniprocessor or multiprocessor

Terminology

- **Hard deadline: late result is of little or no value, or may lead to catastrophe**
  - need to guarantee it
- **Soft deadline: late result may still be useful**
  - probability of missing deadlines
  - With prob. 0.95 a telephone switch connects in 10 seconds
- **How serious is serious?**
- **Tardiness:**
  - min{ 0, deadline - completion time}
- **Utility:**
  - function of tardiness



Terminology: Temporal Parameters

- **Release time:**
  - fixed (*r*), jitter [$r-\Delta$, $r+\Delta$], sporadic or aperiodic
- **Execution time:**
  - Unpredictability due to memory refresh, contention due to DMA, pipelining, cache misses, interrupts, OS overhead
  - Execution-path variations
- **WCET: a "deterministic" parameter for the worst-case execution time**
  - a conservative measure
  - an assumption to make scheduling and validation feasible
  - how can you measure the WCET of a job?

Effective release time and deadline

- **Release time of a job can be later than that of its successor**
- **Deadline of a job can be earlier than that of its predecessor**
- **Effective release time$_i$ = max {release time$_i$, effective release times of all its precedessors}**

  Effective release time = release time if no predecessor

- **Effective deadline$_i$ = min {deadline$_i$, deadlines of all its successors}**

  Effective deadline = deadline if no successor

Classical Uniprocessor Scheduling Algorithms

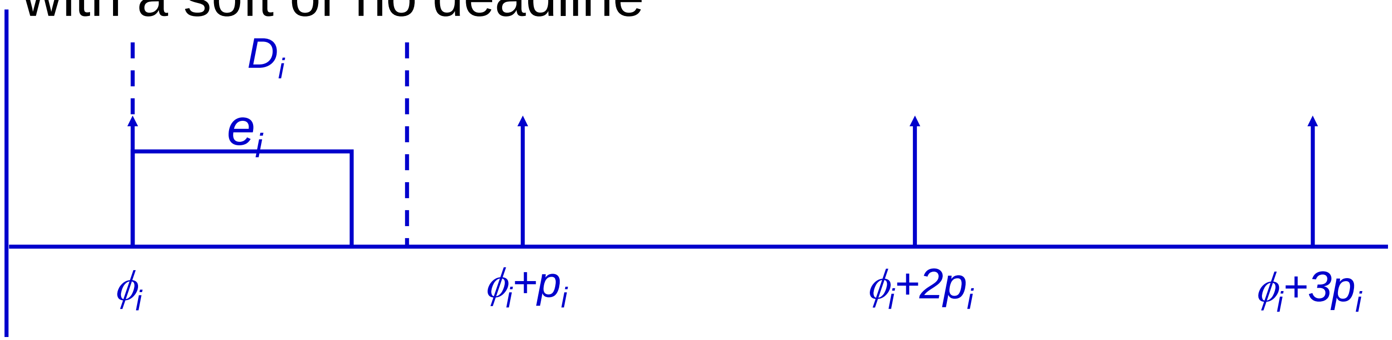
- **Rate Monotonic (RM):** ***statically*** **assign higher priorities to tasks with lower periods**
- **Deadline Monotonic (DM):** **the smaller *relative deadline* the higher priority.**
- **Earliest Deadline First (EDF):** **the earlier the deadline, the higher the priority; optimal if preemption is allowed and jobs do not contend for resources.**
- **Minimum-Laxity-First (MLF):** **the smaller the laxity the higher priority; optimal just like EDF.**

Assumptions and Task Models for Classical Sched Algs

- **Assumptions:**
  - Fully preemtable with negligible costs,
  - Independent tasks, i.e., no precedence constraints between tasks
  - CPU is the only resource to deal with.
- **Task Model:**
  - Characterized by a subset of period/interarrival time, phase, execution time, absolute/effective release time, absolute/relative/effective deadline.
  - Example: Periodic task $T_i = (\phi_i, P_i, e_i, d_i)$ = (phase, period, execution time, relative deadline)
  - Task vs. job

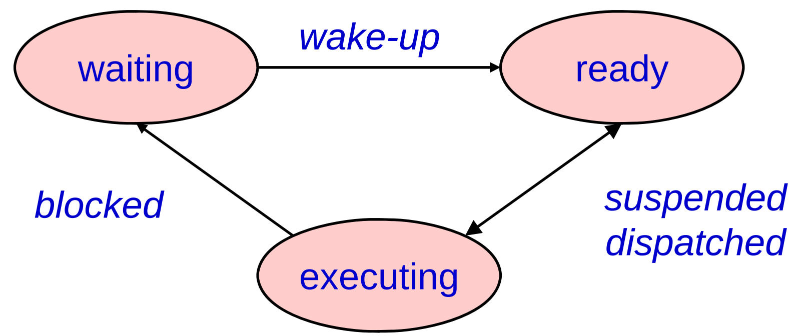
More on Task model

- **Periodic task T_i: (examples?)**
  - constant (or bounded) period, p_i: inter-release time between two consecutive jobs
  - phase ϕ_i, utilization $u_i = e_i / p_i$, deadline (relative) D_i
- **Aperiodic and sporadic: (examples?)**
  - Sporadic: uncertain interarrival times but with a minimum separation and with a hard deadline
  - aperiodic: non-periodic with no minimum separation and usually with a soft or no deadline



Task Functional Parameters

- **Preemptivity: suspend the executing job and switch to a different job**
  - should a job (or a portion of job) be preemptable
  - context switch: save the current process status (PC, registers, etc.) and initiate a ready job
- **Preemptivity of resources: concurrent use of resources or critical section**
  - lock, semaphore, disable interrupts
- **How can a context switch be triggered?**
  - Assume you want to preempt an executing job, why?
    - a higher priority job arrives
    - Use up the assigned time quantum



Task Scheduling

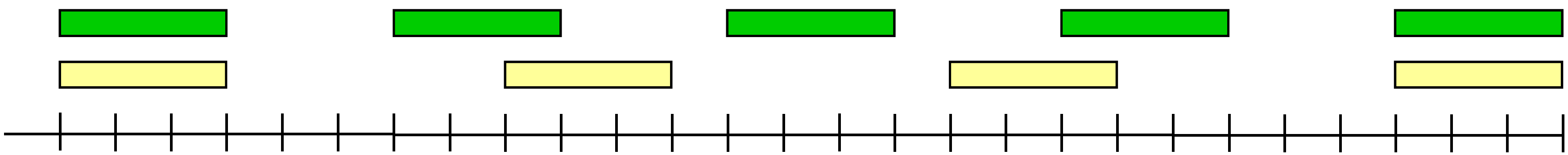
- **Schedule: to determine which job is assigned to a processor at any given time**
  - valid schedule: satisfies constraints (release time, WCET, precedence constraints, etc.)
  - feasible schedule: meet job deadlines
- **Need an algorithm to generate a schedule**
  - optimal scheduling algorithm: can always find a feasible schedule if any other alg can
- **Scheduler or dispatcher: the mechanism to implement a schedule**
- **Interaction between schedulers**

Commonly-Used Real-Time Scheduling Approaches

- **Clock-Driven: determines which job to execute when. All parameters of hard RT jobs are *fixed* and *known*; a schedule is computed *off-line* and stored for use at runtime.**
- **Weighted Round-Robin: for high-speed networks, where length of a round = sum of all weights.**
- **Priority-Driven: assigns priorities to jobs and executes jobs in priority order,**
  - Static priority assignment: Rate or Deadline Monotonic
  - Dynamic priority assignment: Earliest Deadline First (EDF), Minimum Laxity First (MLF).

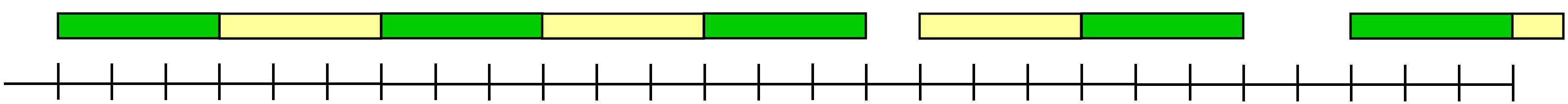
Clock-Driven Task Scheduling

- **Clock-driven**
  - a schedule determines (off-line) which job to be executed at each instant
  - static or *cyclic*
  - predictable and deterministic
  - scheduler: invoked by a timer
  - multiple tables for different operation modes



$p_1 = 6,\ e_1 = 3,\ d_1 = 6$
$p_2 = 8,\ e_2 = 3,\ d_2 = 8$

Major cycle = lcm (6,8) = 24



Clock-Driven RT Scheduling, cont'd

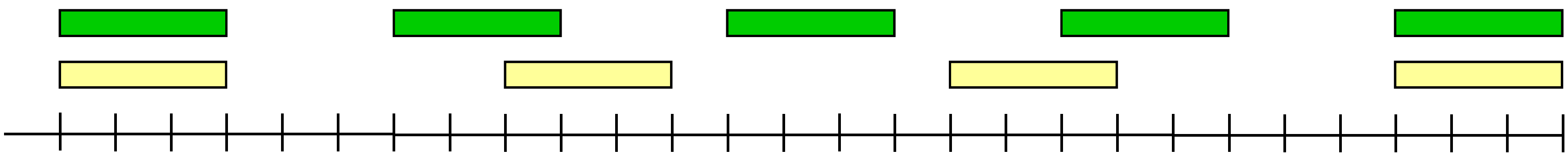
- **Time line is partitioned into *frames*, each with length $f \geq \max_{1 \leq i \leq n} e_i$ and f must also be a divisor of the planning (major) cycle, $F = \lceil L/f \rceil$.**
- **Scheduling decisions are made at the *beginning* of each frame, *not within* a frame.**
- **The first job of each task is released at the beginning of some frame.**
- **Cyclic executive: table-driven scheduler.**
- **Scheduling block L(k): names of job slices scheduled to execute within frame k.**

Cyclic Executive

```
Input: stored schedule: L(k) for k=0,1,⋯, F-1; /*F=# of frames per major cycle*/
    Aperiodic job queue
Task CYCLIC_EXECUTIVE:
     current time t=0;  current frame k=0;
     do forever
        accept clock interrupt at time tf;
        currentBlock = L(k);
        t := t+1;    k:= t mod F;
        if the last job is not completed, take appropriate action;
        if any of the slices in currentBlock is not released, take action;
        wake up the periodic server to execute the slices in current Block;
        sleep until the periodic server completes;/*completes periodic job slices*/
        while the aperiodic job queue is nonempty,
           wake up the job at the head of the aperiodic queue;
           sleep until the aperiodic job completes:
           remove the aperiodic job from the queue:
       endwhile;
       sleep until the next clock interrupt;
   enddo;
end CYCLIC_EXECUTIVE
```

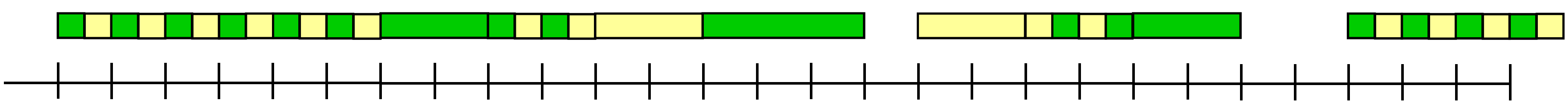
Round-Robin Task Scheduling

- **Weighted Round-robin**
  - interleave job executions
  - allocate a time slice to each job in the FIFO queue
  - time slice may vary while sharing the processor
  - good for pipelined jobs, e.g., network packets



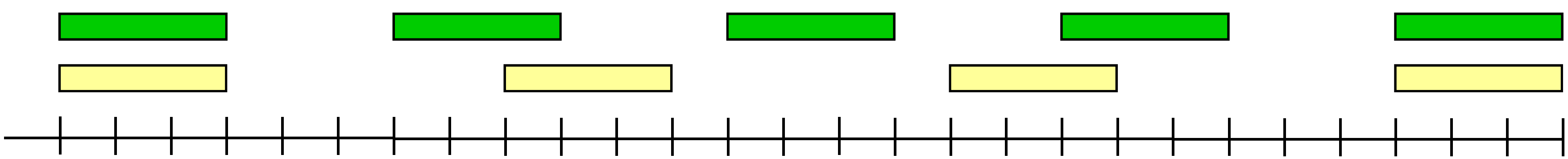
$p_1 = 6, e_1 = 3, d_1 = 6$
$p_2 = 8, e_2 = 3, d_2 = 8$

Major/planning cycle = LCM (6,8) = 24



Priority-Driven Task Scheduling

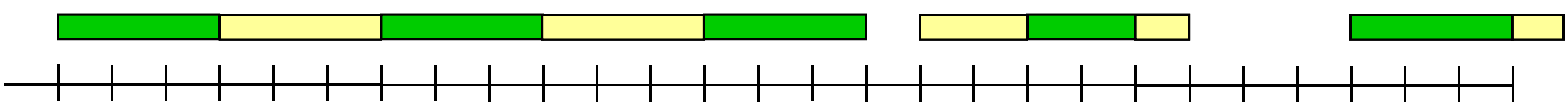
- **Priority-driven**
  - The highest-priority job gets to run until completion or blocked
  - A processor is never idle if ready jobs are waiting (*work-conserving*)
  - *preemptive* or *non-preemptive*
  - priority assignment can be *static* or *dynamic*
  - Scheduler just looks at the priority queue for waiting jobs (*list schedule*)



$p_1 = 6,\ e_1 = 3,\ d_1 = 6$

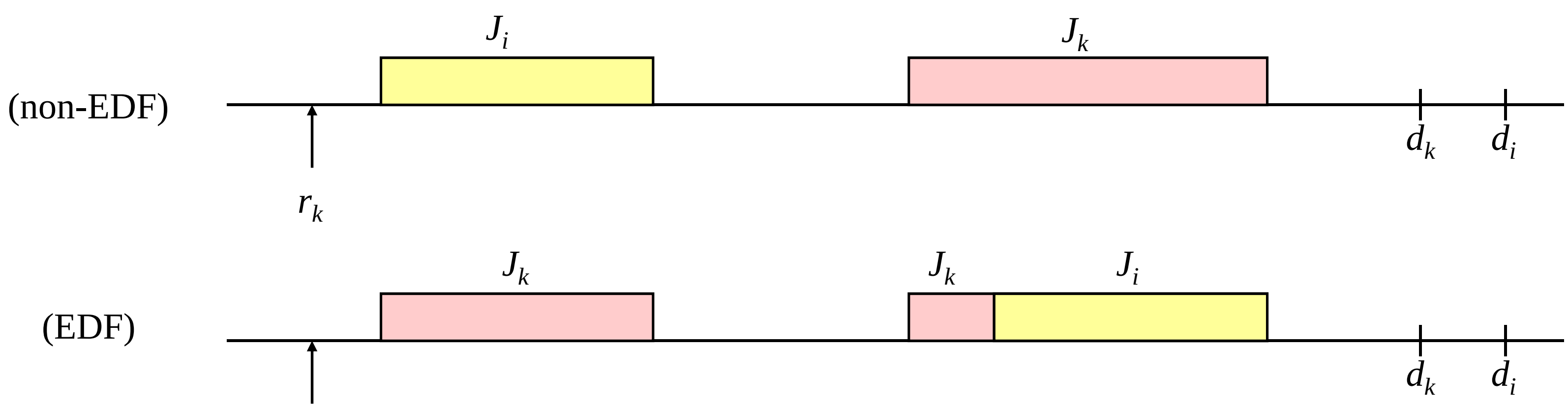
$p_2 = 8,\ e_2 = 3,\ d_2 = 8$

Major cycle = lcm (6,8) = 24



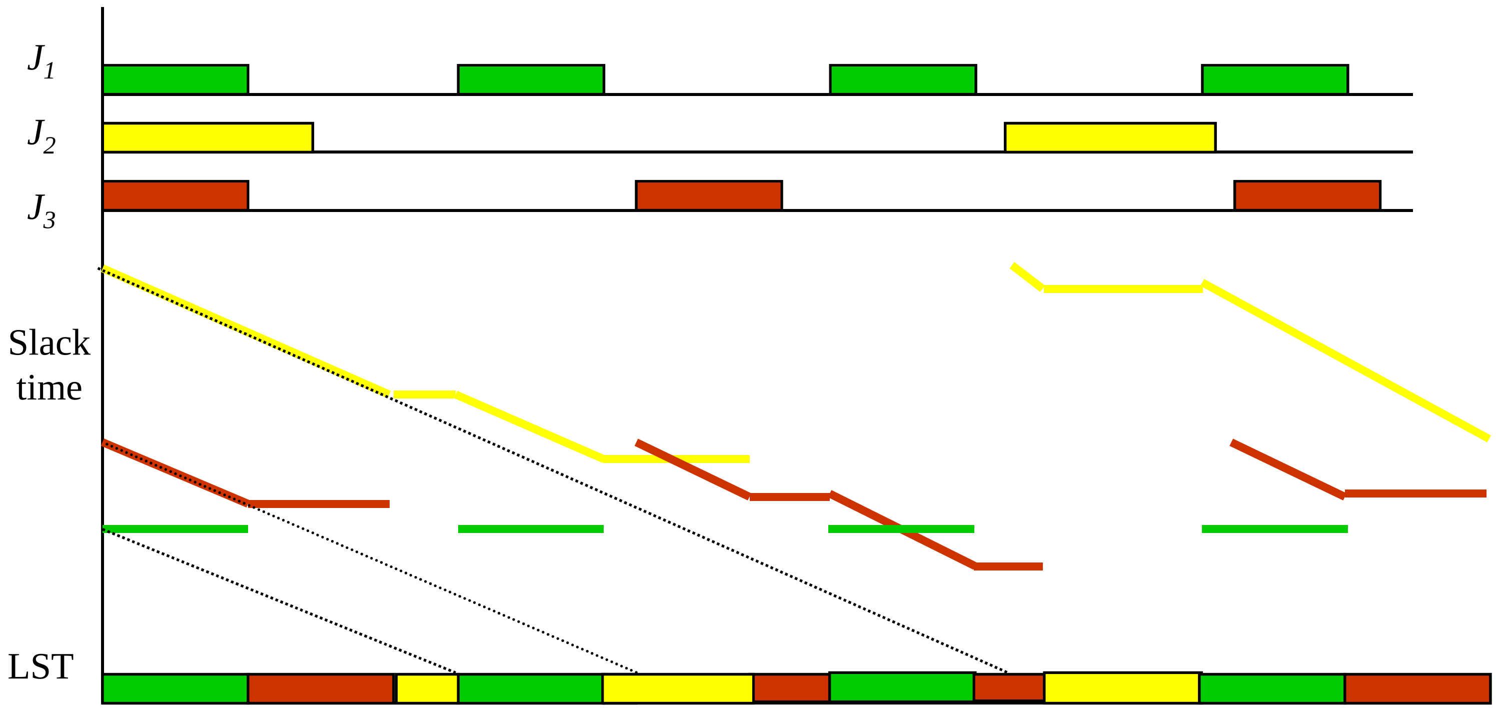
Earliest-Deadline-First (EDF) Schedule

- **Preemptive dynamic priority scheduling**
  - a job with earliest (absolute) deadline has the highest priority
  - does not require the knowledge of execution time
- **Optimal if**
  - single processor, no resource contention, preemptive
  - why is this optimal? assume a feasible schedule



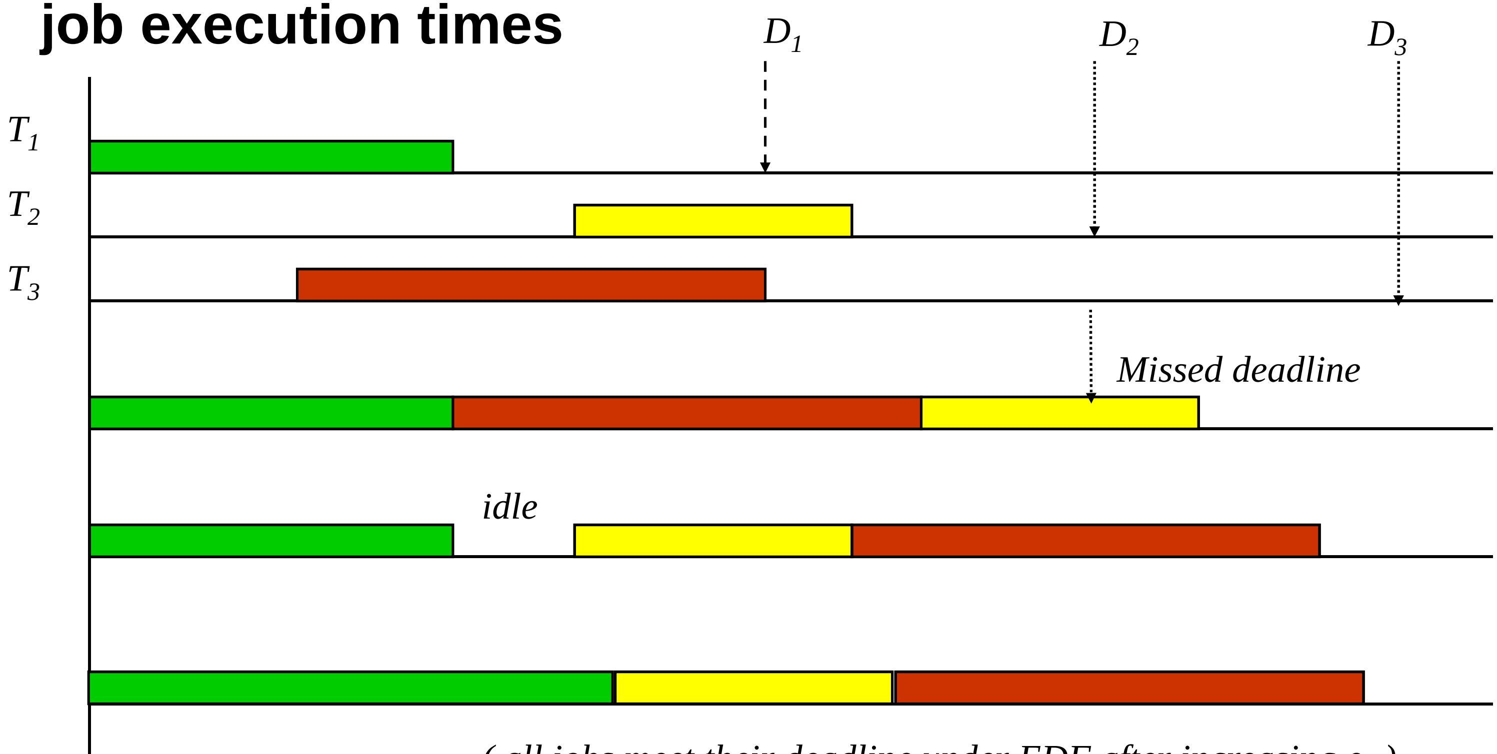
Least Slack Time (LST) Schedule

- **Preemptive priorityscheduling based on slack time (d_i - e_i^*)**
  - schedule instants: when jobs are released or completed.
  - optimal for preemptive single processor schedule



Non-optimality of EDF

- Non-preemptive or multiple processors
- scheduling anomaly --- the schedule fails even after we reduce job execution times



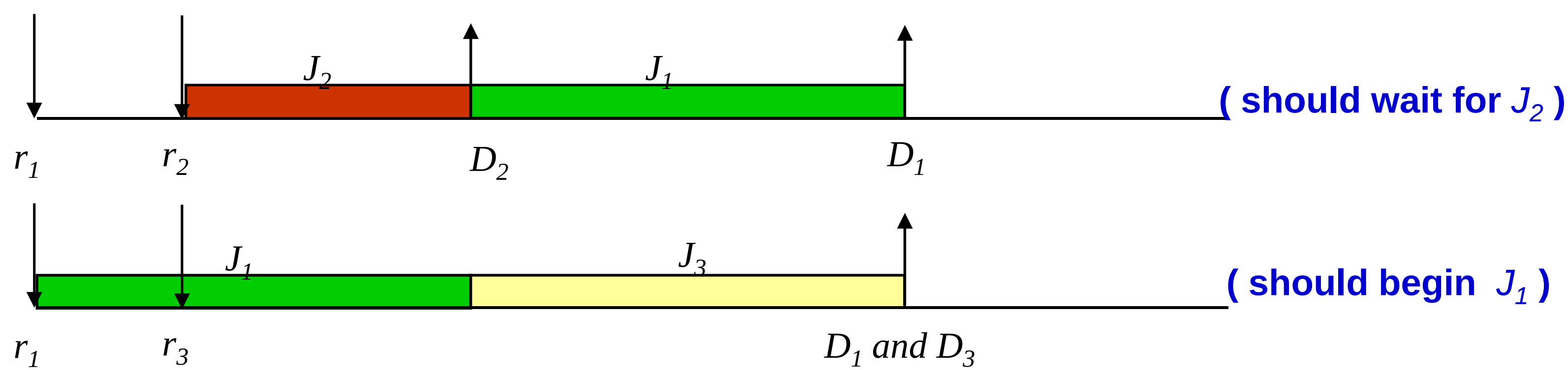
(all jobs meet their deadline under EDF after increasing e_1)

Predictable System

- **With variant job execution times, do we know when a task is started or completed?**
- **If the start & completion times and the deadline are known, then we can determine whether a schedule is feasible or not**
- **Two extreme conditions:**
  - maximal schedule: all jobs take their maximal execution times
  - minimal schedule: all jobs take their minimal execution times
- **A job is predictable iff its start and complete times are predictable:**
  - $s^{-}(J_i) \le s(J_i) \le s^{+}(J_i)$
  - $f^{-}(J_i) \le f(J_i) \le f^{+}(J_i)$
- **The execution of every job in a set of independent, preemptive jobs with fixed release times is predictable when scheduled in a priority-driven manner on a single processor**

On-line vs. Off-line Scheduling

- ***Off-line scheduling:*** **the schedule is computed off-line and is based on the knowledge of the release times and execution times of all jobs.**
  - A system with fixed sets of functions and job characteristics does not vary or vary only slightly.
- ***On-line scheduling:*** **a scheduler makes each scheduling decision without knowledge about the jobs that will be released in future.**
  - there is no optimal on-line schedule if jobs are non-preemptive
  - when a job is released, the system can serve it, or wait for future jobs



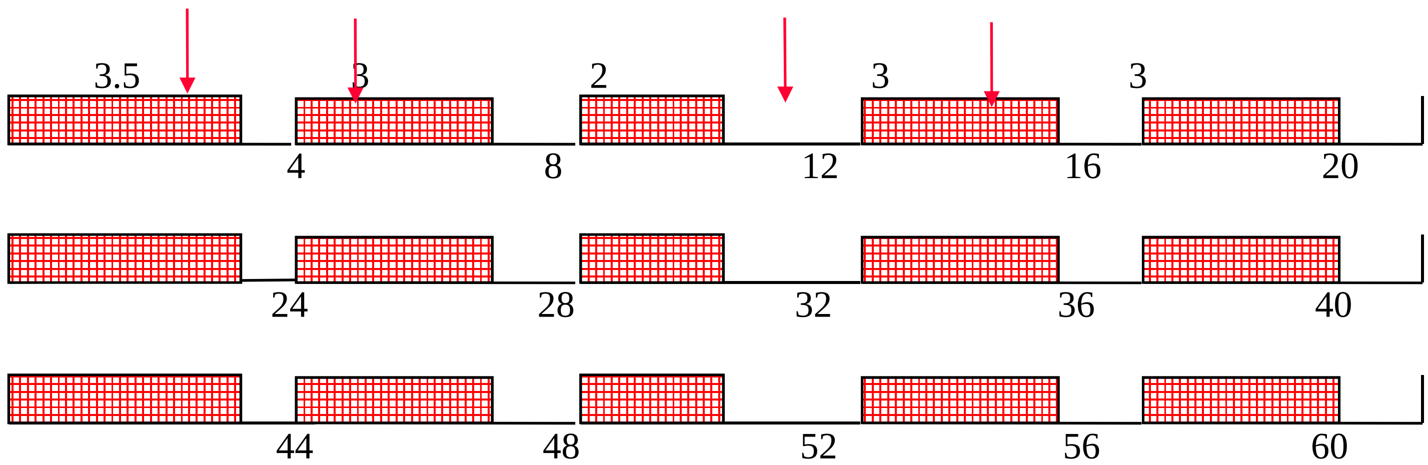
Aperiodic Tasks

- **A periodic server follows the cyclic schedule**
- **A aperiodic server looks at the aperiodic task queue**
  - runs at the background
- **Slack stealing**
  - slack time: how much each periodic task can be delayed
- **Assume all tasks must be completed before the end of their frames and aperiodic tasks are not preemptable**
  - at frame k, e_k is allocated to periodic tasks
  - slack time: $s= f - e_k$
  - at the beginning of frame k, find an aperiodic task j with an execution time e_j that is less than s
  - try to run the other aperiodic task with a slack time: $s=s - e_j$
- **Do slack stealing at the beginning of each frame and then examine the queue when idle**

Sporadic Tasks

- **Accept if the sporadic task can be done before its deadline**
- **If more than one sporadic task $\Rightarrow$ EDF**
- **Assume tasks are preemptable (run across frame boundary)**
- **When a sporadic task arrives ---- schedule it immediately or at the beginning of the next frame**
  - is there enough slack time before its deadline and for every existing sporadic task
- **Bookkeeping**
  - slack time from frame i to l: $\sigma(i,l)$, l=1,2,3,…, F
  - for each sporadic task, remaining execution time and slack time
- **Let the deadline of an arriving task is in frame l+1**
  - is there enough slack time for all tasks with a deadline before frame l+1?
  - for each task with a deadline later than frame l+1, can it be delayed by the new arrival?

Sporadic Tasks -- Example



- **A table of σ(i,l), i,l=1,2,3,…, F**
- **Assume**
  - S1(17,4.5) arrives at time 3 --- checks at time 4
  - S2(29,4) arrives at time 5
  - S3(22,1.5) arrives at time 11
  - S4(44,5.0) arrives at time 14

Summary of Cyclic Schedule

- **Pros**
  - simple, table-driven, easy to validate (knows what is doing at any moment)
  - fits well for harmonic periods and small system variations
  - static schedule $\Rightarrow$ deterministic, static resource allocation, no preemption
  - small jitter
  - no scheduling anomalies
- **Cons**
  - difficult to change (need to re-schedule all tasks)
  - fixed released times for the set of tasks
  - difficult to deal with different temporal dependencies
  - schedule algorithm may get complex (NP-hard)
  - doesn't support aperiodic and sporadic tasks efficiently