Priority-Based Scheduling

- Jobs have priorities
- Scheduler selects highest-priority ready job
- Two possibilities:
  - **Dynamic-priority scheduling**
    - Different jobs of task may be assigned different priorities
    - Example
      - Job $J_{i,k}$ of task T_i has higher priority than job $J_{j,m}$ of T_j
      - Job $J_{i,l}$ of T_i has lower priority than job $J_{j,n}$ of T_j
  - Static-priority scheduling

Dynamic-Priority Scheduling

- Consider a task's characteristics:
  - Phase
  - Period
  - Execution time
  - Relative deadline
- What derived characteristics does each job have?
  - Release time
  - Absolute deadline
- Main goal: schedule jobs so that no job misses deadline
  - What would be the most straightforward way to do this?

Earliest Deadline First (EDF)

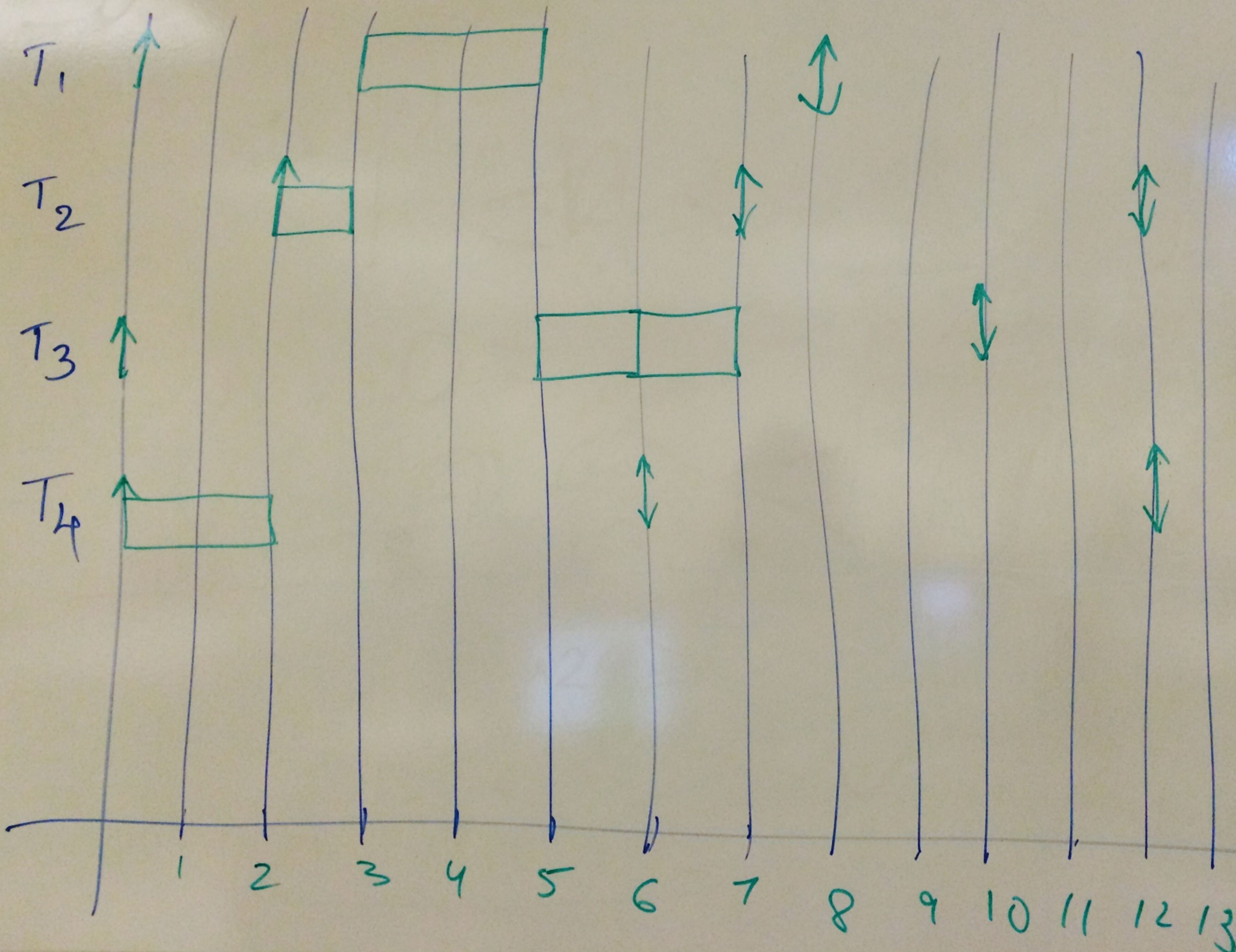
- Whenever scheduling decision needs to be made
  - Job with earliest absolute deadline given highest priority

- When must scheduling decisions be made?
  1) When currently executing job completes
  2) When new job is released/activated

- Consequence of 2)
  - Jobs can be interrupted during execution
  - I.e., jobs can be preempted

Example

- Derive preemptive EDF schedule for task set below
  - T1(8, 2), T2(2, 5, 1, 5), T3(10, 2) and T4(6, 2)



T1
T2
T3
T4
1
2
3
4
5
6
7
8
9
10
11
12
13

Earliest Deadline First (EDF)

Theorem 4-1: [Liu and Layland] When preemption is allowed and jobs do not contend for resources, the EDF algorithm can produce a feasible schedule of a set **J** of jobs with arbitrary release times and deadlines on a processor if and only if **J** has feasible schedules.

In other words, preemptive EDF is **optimal**

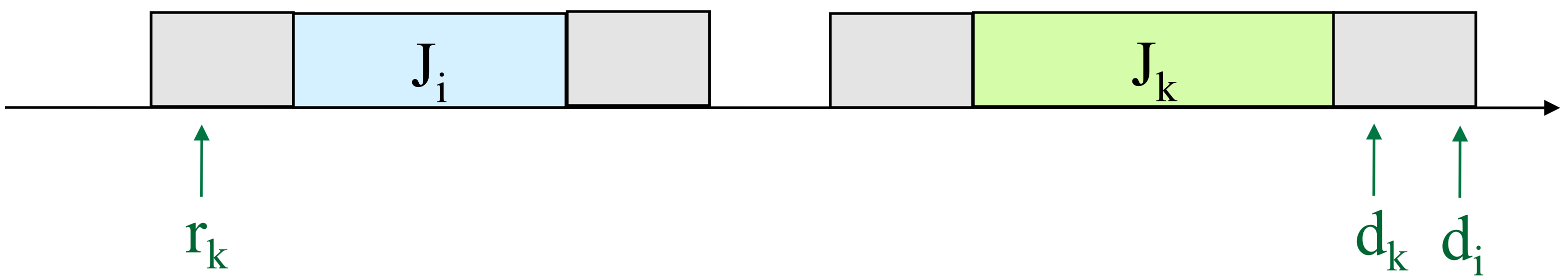
Notes:

- Applies even if tasks are not periodic
- If periodic, relative deadline could be <, = or > period
- Precedence constraints are allowed

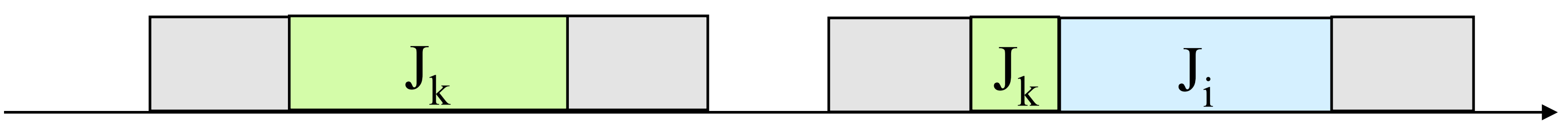
Proof sketch for Theorem 4-1

Any feasible schedule of **J** can be systematically transformed into an EDF schedule

Suppose parts of two jobs J_i and J_k are executed out of EDF order:



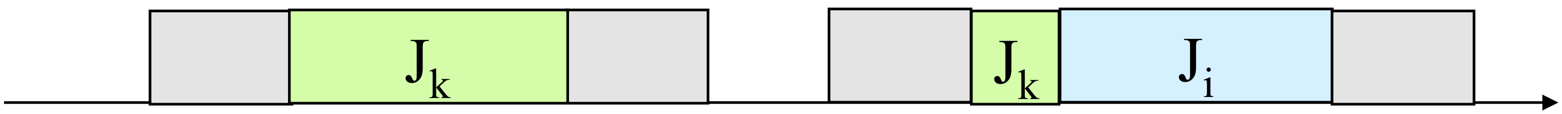
This situation can be corrected by performing a “swap”:



Proof sketch for Theorem 4-1

Inductively repeating this can eliminate all out-of-order violations

Resulting schedule may still have idle intervals even when job is ready:

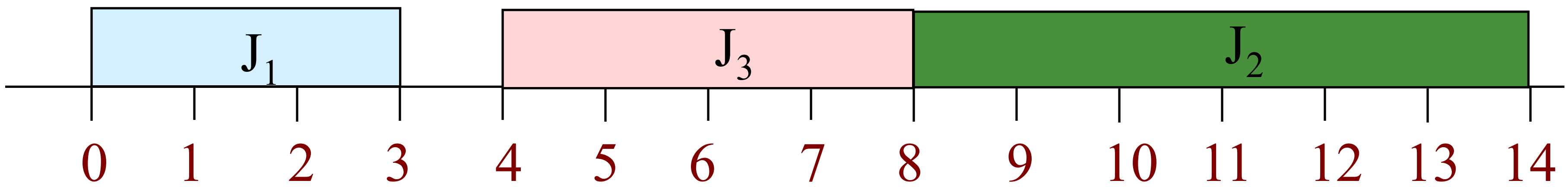


Such idle intervals can be eliminated by moving some jobs forward:

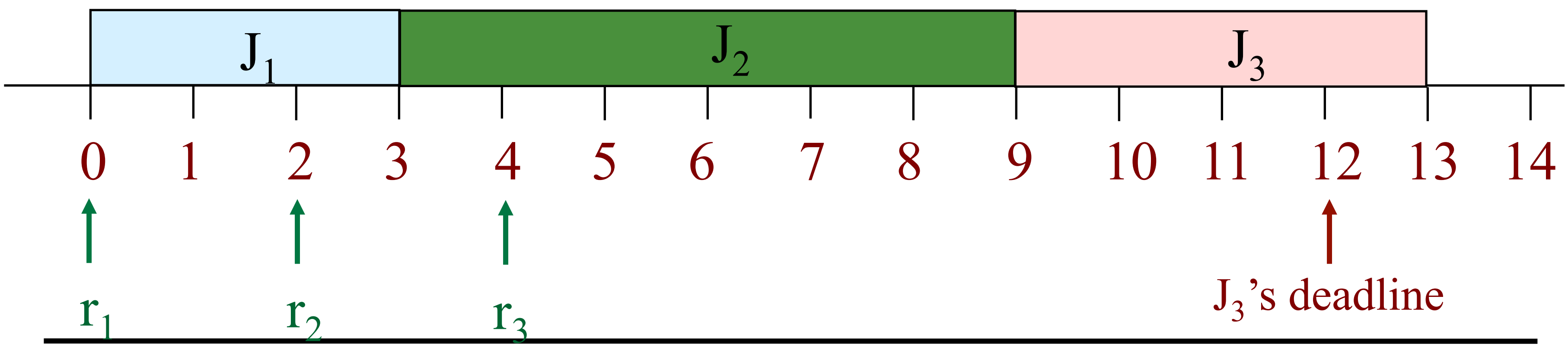
What about non-preemptive EDF?

Consider a system of three jobs J_1, J_2, and J_3 such that
$(r_1, e_1, d_1) = (0, 3, 10)$, $(r_2, e_2, d_2) = (2, 6, 14)$, $(r_3, e_3, d_3) = (4, 4, 12)$.

Here's a feasible schedule:



Now, what happens if we use non-preemptive EDF?



So, what's the conclusion?

Theorem: Non-preemptive EDF is not optimal.

Question: Does this mean preemptive EDF is always better than non-preemptive EDF in practice?

Not necessarily!
EDF optimality proof assumes there is no penalty due to preemption

Note: from now on, "EDF" means preemptive EDF, unless specified otherwise

Schedulability test for (preemptive) EDF

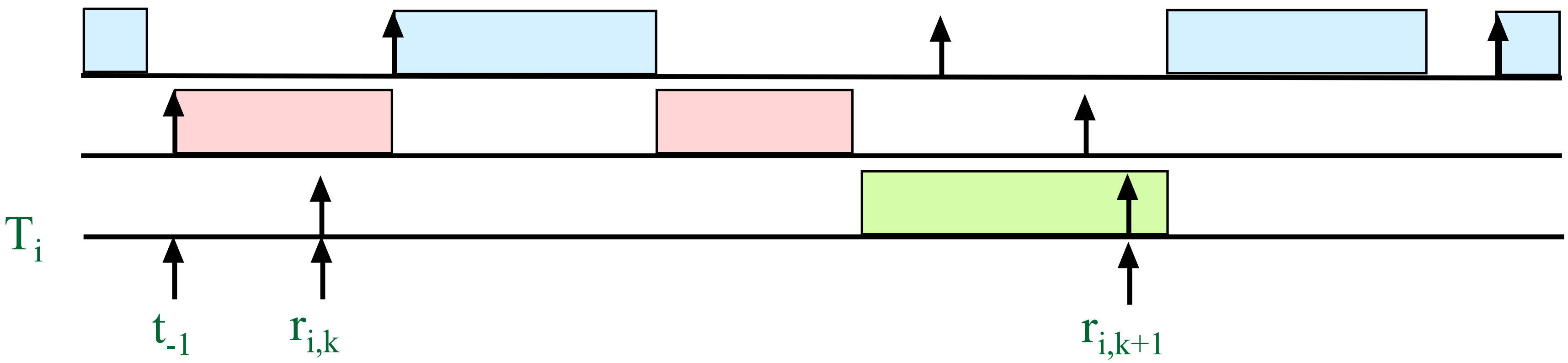
- Obvious observation
  - Total utilization (U) must be less than or equal to 1 (100%)

- But, if $U \leq 1$, is the task set surely schedulable?
  - The contrapositive of a true statement is always true
    - Statement: If $U \leq 1$, then the task set is schedulable
    - Contrapositive: if task set is not schedulable, then $U > 1$
  - Our approach
    - Start with assumption that task set is not schedulable
    - Work backwards and prove the contrapositive

Proving the contrapositive

Assume task set is not schedulable

Let $J_{i,k}$ be the first job to miss its deadline.



this is the last "idle instant" (each task whose next job has a deadline at or before $r_{i,k+1}$ either has no ready job, or has *just* released a job, and no job with a deadline after $r_{i,k+1}$ executes in $[t_{-1}, r_{i,k+1}]$.)

Proving the contrapositive

$J_{i,k}$ missed its deadline…

➔ demand placed on processor in $[t_{-1}, r_{i,k+1})$ by jobs with deadlines $\leq r_{i,k+1}$ is greater than the available processor time in $[t_{-1}, r_{i,k+1}]$

$r_{i,k+1} - t_{-1}$ = available processor time within $[t_{-1}, r_{i,k+1}]$

$r_{i,k+1} - t_{-1}$ < demand placed on processor within $[t_{-1}, r_{i,k+1})$ by jobs with deadlines $\leq r_{i,k+1}$

$$r_{i,k+1} - t_{-1} < \sum_{j=1}^{N} \{\#\text{ jobs of } T_j \text{ with deadlines} \leq r_{i,k+1} \text{ released within } [t_{-1}, r_{i,k+1})\}\, e_j$$

$$r_{i,k+1} - t_{-1} < \sum_{j=1}^{N} \left\lfloor \frac{r_{i,k+1} - t_{-1}}{p_j} \right\rfloor e_j \;\Rightarrow\; r_{i,k+1} - t_{-1} < \sum_{j=1}^{N} \frac{r_{i,k+1} - t_{-1}}{p_j} e_j$$

Proving the contrapositive

We have

$$r_{i,k+1} - t - 1 < \sum_{j=1}^{N} \frac{r_{i,k+1} - t - 1}{p_j} e_j$$

Canceling $r_{i,k+1} - t - 1$ yields

$$1 < \sum_{j=1}^{N} \frac{e_j}{p_j}$$

i.e.,

$$1 < U$$

Note: This proof is valid even if **relative deadlines are larger than periods**

So, what's the conclusion?

If $U > 1$, task set is not schedulable
If $U \leq 1$, task set is schedulable

Here's the formal theorem:

Theorem 6-1: [Liu and Layland] A system **T** of independent, preemptable, periodic tasks with relative deadlines equal to their periods can be feasibly scheduled (under EDF) on one processor if and only if its total utilization U is at most one.

EDF with Deadlines < Periods

If deadlines are less than periods then $U \leq 1$ is no longer a sufficient schedulability condition

Consider two tasks such that, for both, $e_i = 1$ and $p_i = 2$

If both have deadlines at 1.9, then the system is not schedulable, even though $U = 1$.

Here, **densities** used instead of utilizations

Definition: The **density of task T_k** is defined as $\delta_k = \frac{e_k}{min(D_k, p_k)}$

The **density of the system** is defined as $\Delta = \sum_{k=1..N} \delta_k$

EDF with Deadlines < Periods

Theorem 6-2: A system **T** of independent, preemptable, periodic tasks can be feasibly scheduled on one processor if its density is at most one.

Proof is similar to that for Theorem 6-1

Note: This theorem only gives **sufficient** condition

We refer to the following as the **schedulability condition for EDF**:

$$\sum_{k=1}^{n} \frac{e_k}{min(D_k, p_k)} \leq 1$$

Proof of non-tightness

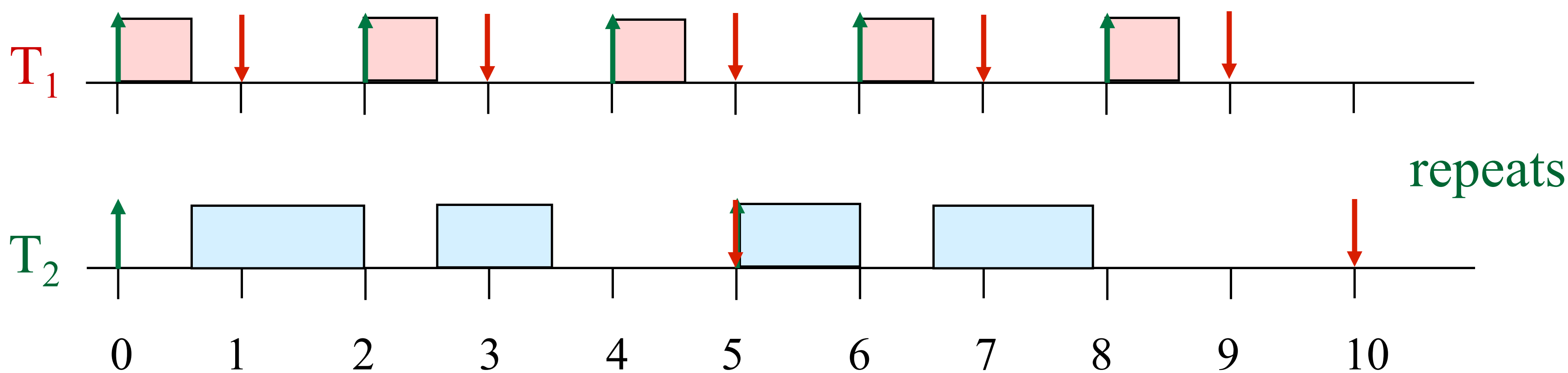
Note that $\Delta > 1$ doesn't imply non-schedulability

Example:

Consider two tasks $T_1 = (2, 0.6, 1)$ and $T_2 = (5, 2.3)$

$\Delta = 0.6/1 + 2.3/5 = 1.06$

Nonetheless, we can schedule this task set under EDF:



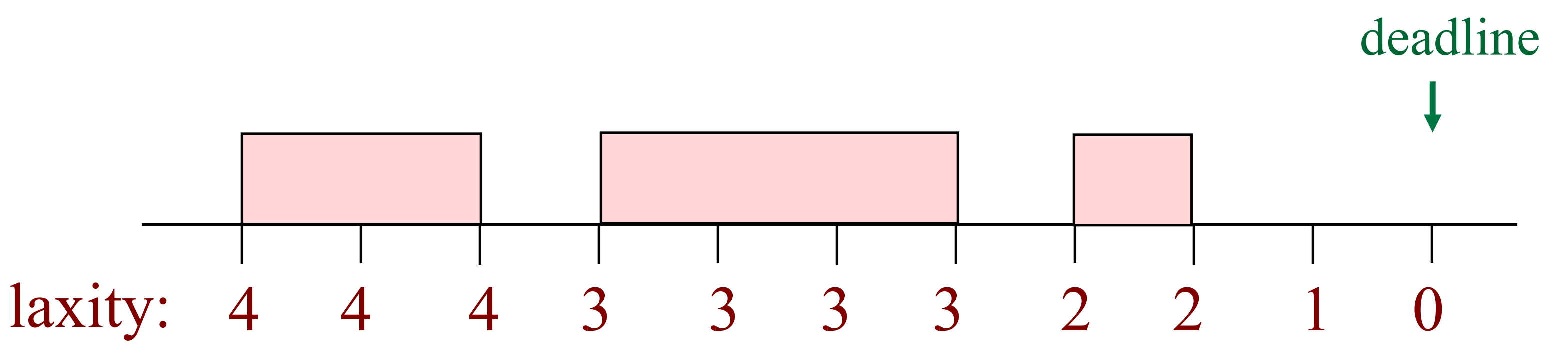
Properties of EDF

- Priority calculation is dynamic at task level
  - Priorities of different jobs of task are different
  - Priority of specific job does not change during its execution

- Priority calculation can also be job level dynamic…

Least Laxity First (LLF)

- **Definition:** At any time t, the **slack** (or **laxity**) of a job with deadline d is equal to d – t minus the time required to complete the remaining portion of the job



- **LLF Scheduling:** job with smallest laxity has highest priority

Optimality of LLF

Theorem 4-3: When preemption is allowed and jobs do not contend for resources, the LLF algorithm can produce a feasible schedule of a set **J** of jobs with arbitrary release times and deadlines on a processor if and only if **J** has feasible schedules.

Proof is similar to that for EDF

Question: Which of EDF and LLF would be preferable in practice?