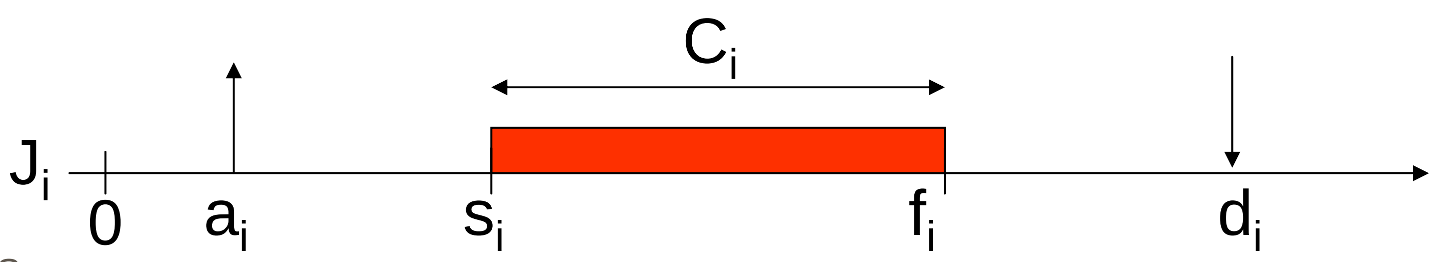
Embedded Systems 18

Timing parameters

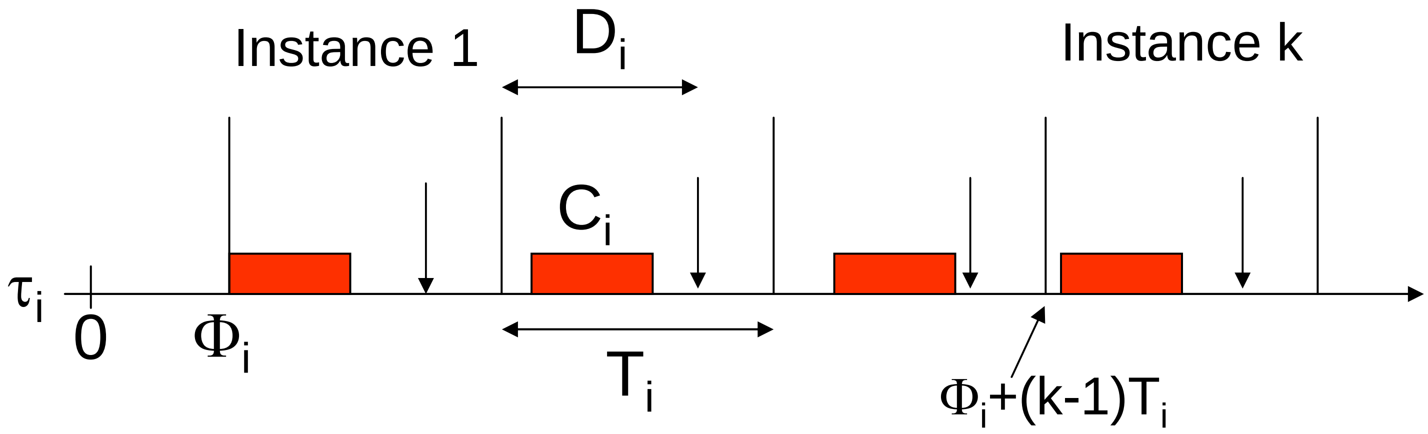
REVIEW

- Timing parameters of a real-time task J_i:
  - **Arrival time a_i**: time at which task becomes ready for execution
  - **Computation time C_i**: time necessary to the processor for executing the task without interruption
  - **Deadline d_i**: time before which a task should be complete to avoid damage to the system
  - **Start time s_i**: time at which a tasks starts its execution
  - **Finishing time f_i**: time at which task finishes its execution



Timing parameters of periodic tasks **REVIEW**

- Phase Φ_i: activation time of first periodic instance
- Period T_i: time difference between two consecutive activations
- Relative deadline D_i: time after activation time of an instance at which it should be complete



Aperiodic scheduling: EDF – Earliest Deadline First

REVIEW

- EDF: At every instant execute the task with the earliest absolute deadline among all the ready tasks.

- **Theorem (Horn '74):**
  Given a set of n independent task with arbitrary arrival times, any algorithm that at every instant executes the task with the earliest absolute deadline among all the ready tasks is optimal with respect to minimizing the maximum lateness.

Aperiodic scheduling: Non-preemptive version

REVIEW

- **Theorem** (Jeffay et al. '91): EDF is an optimal *non-idle* scheduling algorithm also in a non-preemptive task model.

- When idle schedules are allowed: problem is NP-hard.
- Possible approaches:
  - Heuristics
  - Bratley's algorithm: branch-and-bound

Periodic scheduling: EDF

REVIEW

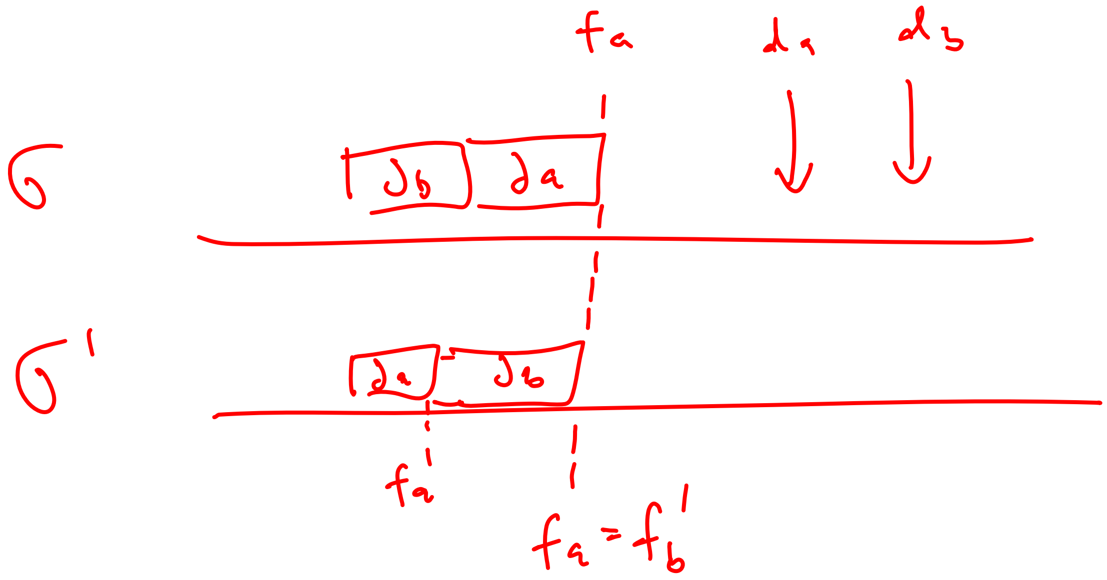
- **Theorem: A set of periodic tasks $\tau_1, ..., \tau_n$ with $D_i = T_i$ is schedulable with EDF iff $U \leq 1$.**

- EDF is applicable to both periodic and a-periodic tasks.

- If there are only periodic tasks, priority-based schemes like "rate monotonic scheduling (RM)" (see later) are often preferred, since
  - They are simpler due to fixed priorities
    $\Rightarrow$ use in "standard OS" possible
  - sorting wrt. to deadlines at run time is not needed

Proof of Jackson‘s Theorem and Horn‘s Theorem

REVIEW



$$L'_{max} = \max\{ f'_a - d_a, f'_b - d_b \}$$

$$f'_a - d_a < f_a - d_a$$

$$f'_b - d_b = f_a - d_b \leq f_a - d_a$$

$$\Rightarrow L'_{max} \leq L_{max}$$

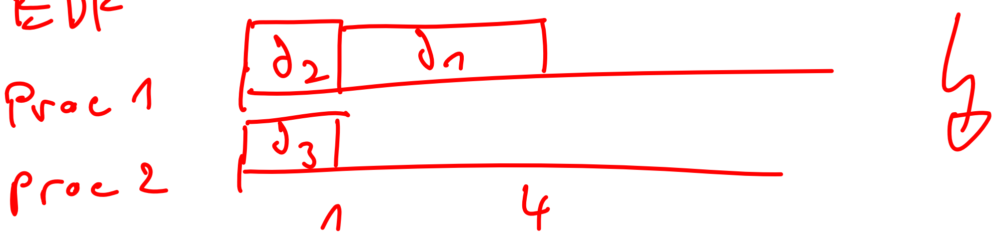
EDF with multiple processors?

$c_1 = 3, \quad d_1 = 3$

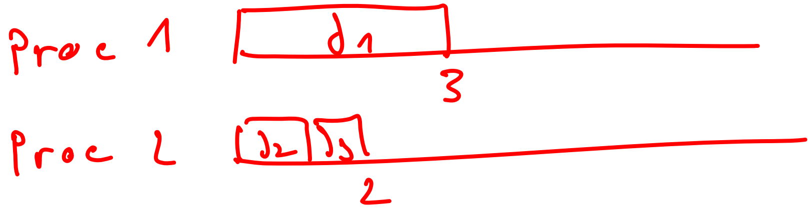
$c_2 = 1, \quad d_2 = 2$

$c_3 = 1, \quad d_3 = 2$

EDF



Feasible schedule:



Multiprocessor Scheduling

Given

- n equivalent processors,
- a finite set M of aperiodic/periodic tasks

find a schedule such that each task always meets its deadline.

Assumptions:

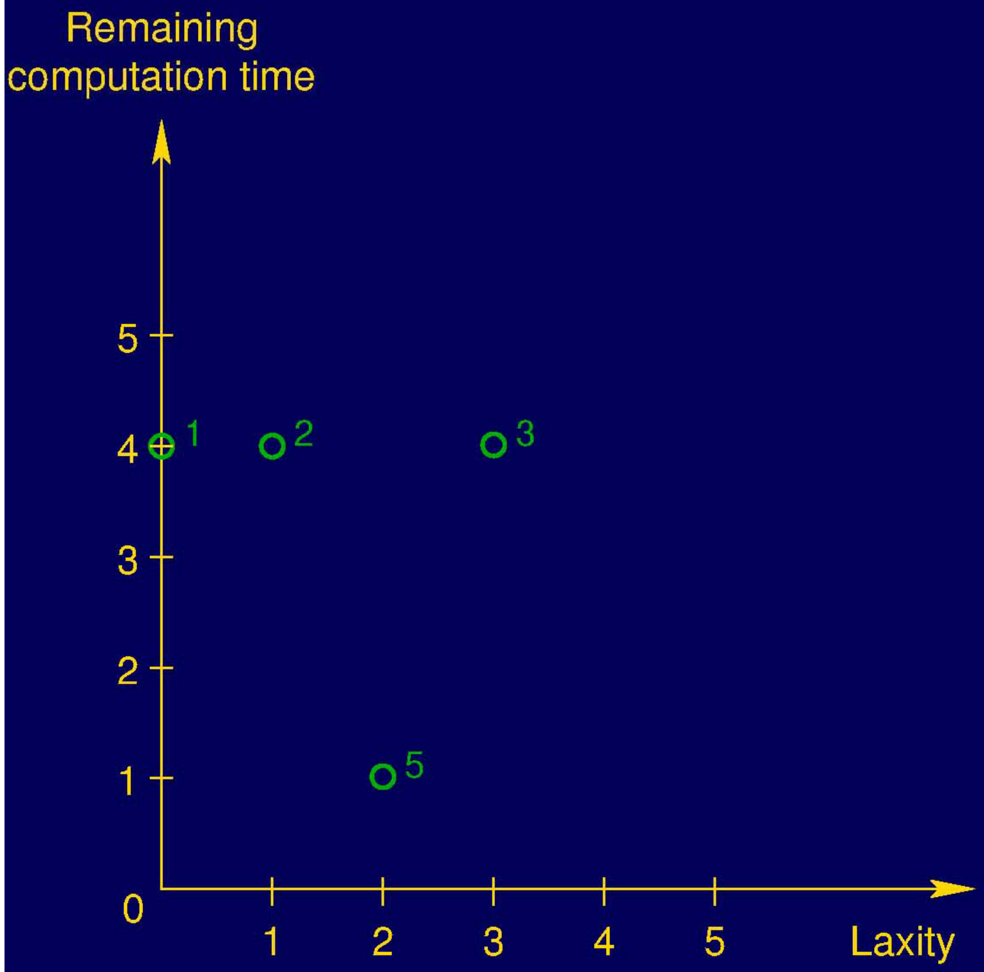
- Tasks can freely be migrated between processors
  - at any integer time instant, without overhead
  - however: no task may run on two processors simultaneously
- All tasks are preemptable
  - at any integer time instant, without overhead

Game-theoretic problem formulation

- Associate possible **states of the system** with positions on a **game board.**
- Associate **choices one can influence** in order to solve the problem with **own moves on the game board.**
- Associate **choices one cannot influence** with **opponent‘s moves.**
- Identify **feasible solutions** with **winning positions**.

Problem solution: find a winning strategy

Game-board representation



When tasks are released, they are inserted into the game board according to their WCET and laxity (= deadline – remain. comp. time).

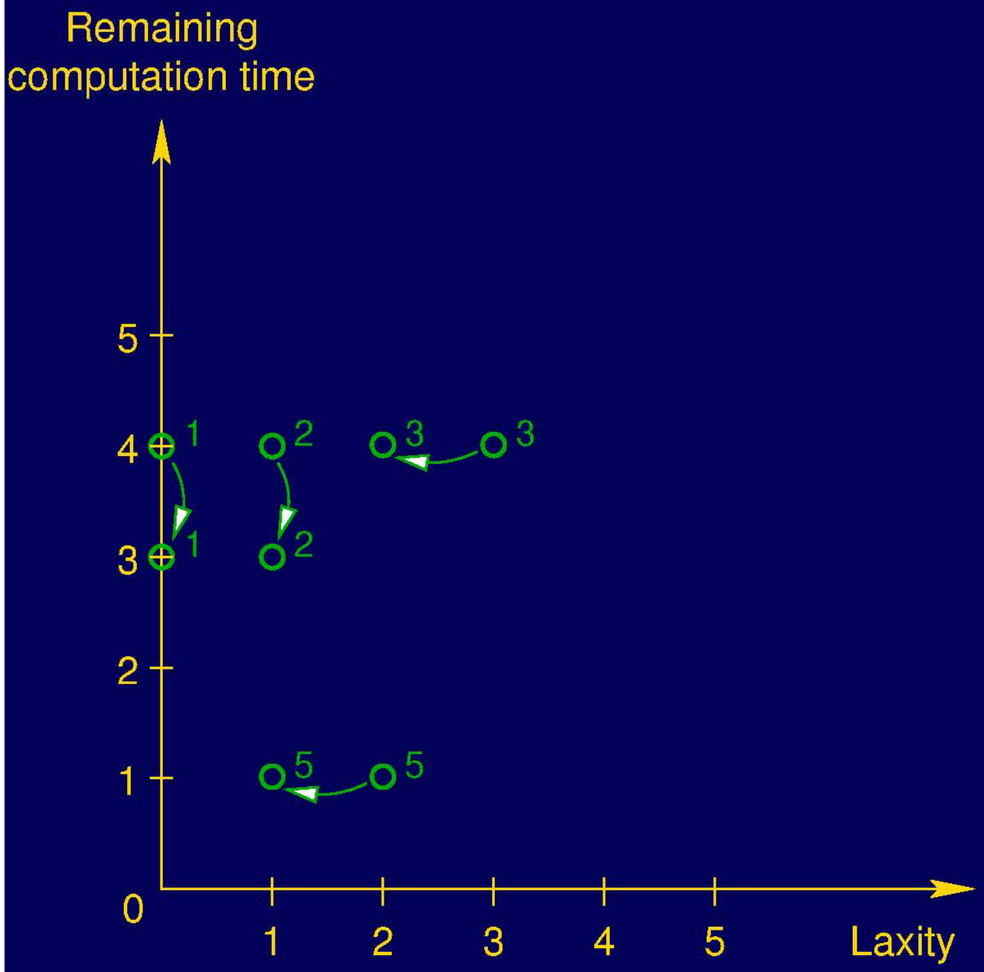
In every time scheduling step / turn of the game:
– at most n nodes go down by 1
– the rest moves 1 to the left

Nodes reaching the x–axis have been allocated all the computation time they need and are thus removed from the game board, as they don't represent scheduling constraints any longer.

The game is lost (i.e., the schedule is infeasible) if some node reaches the second quadrant.

It is won if no node remains on the board.

Game-board representation



When tasks are released, they are inserted into the game board according to their WCET and laxity (= deadline – remain. comp. time).

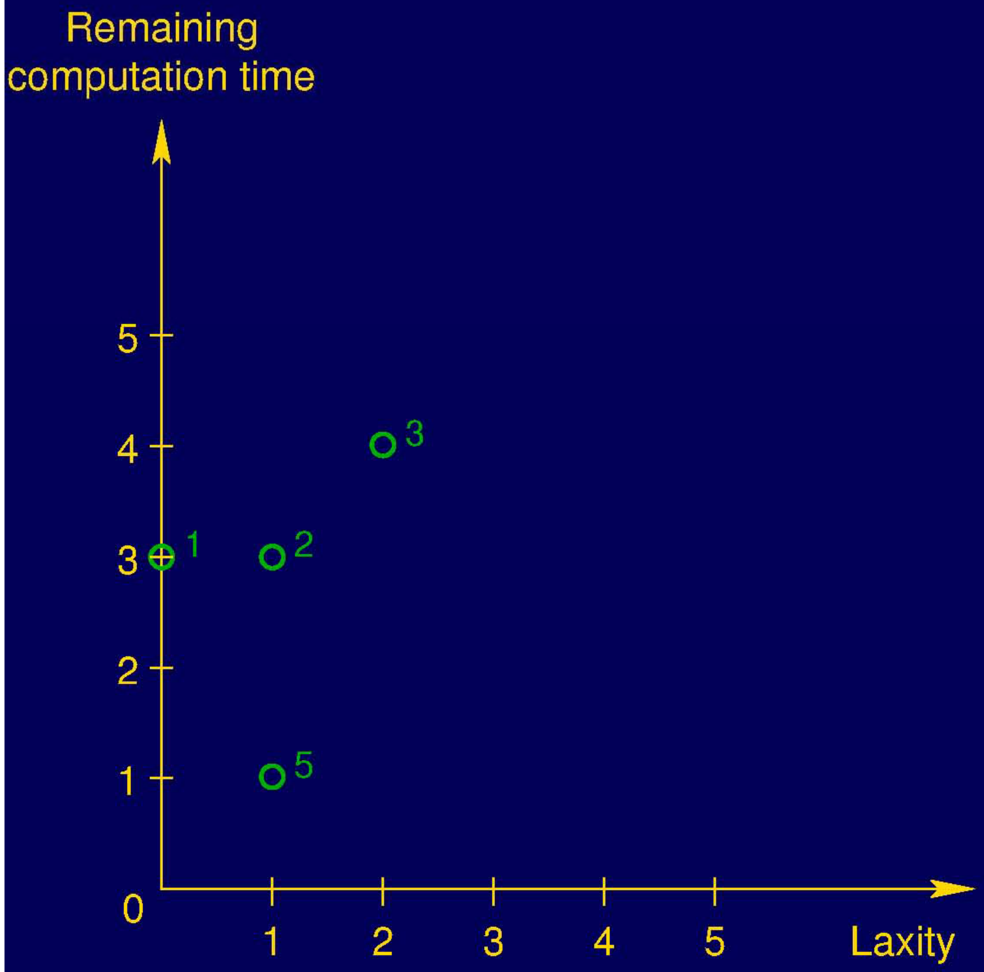
In every time scheduling step / turn of the game:
– at most n nodes go down by 1
– the rest moves 1 to the left

Nodes reaching the x–axis have been allocated all the computation time they need and are thus removed from the game board, as they don't represent scheduling constraints any longer.

The game is lost (i.e., the schedule is infeasible) if some node reaches the second quadrant.

It is won if no node remains on the board.

Game-board representation



When tasks are released, they are inserted into the game board according to their WCET and laxity (= deadline – remain. comp. time).

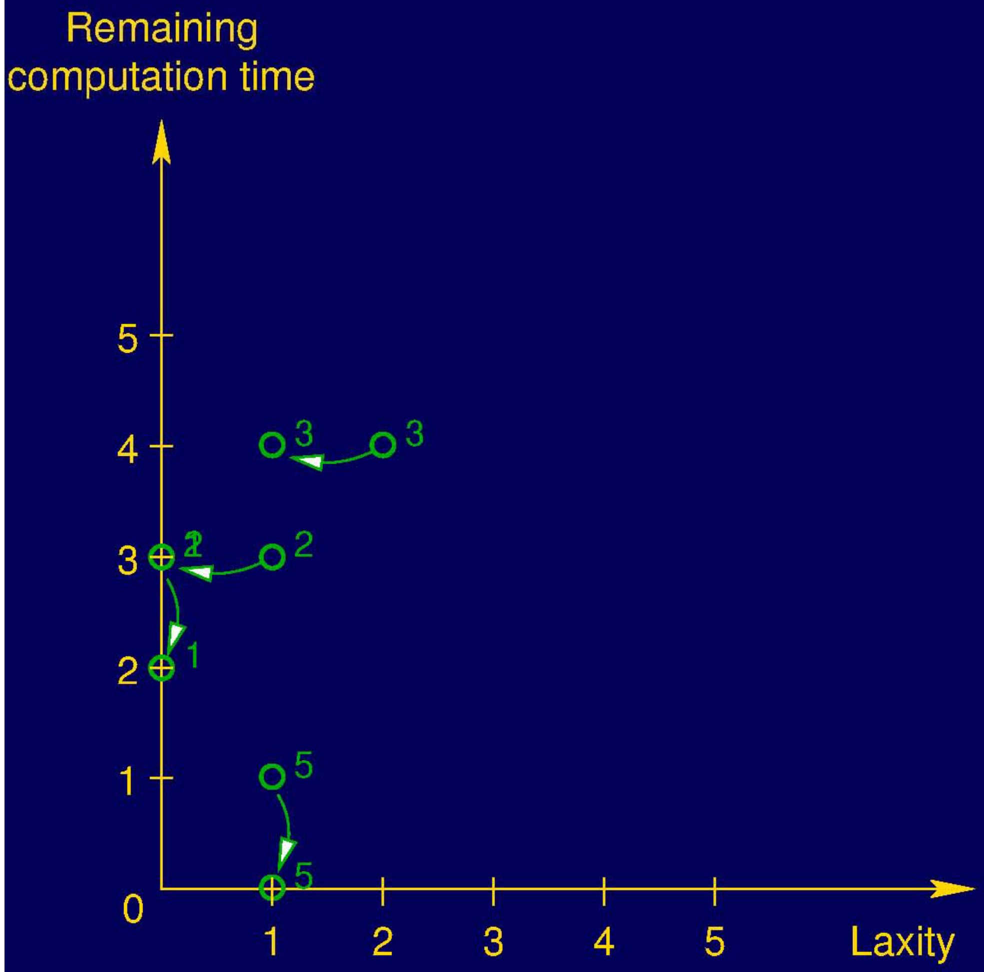
In every time scheduling step / turn of the game:
– at most n nodes go down by 1
– the rest moves 1 to the left

Nodes reaching the x–axis have been allocated all the computation time they need and are thus removed from the game board, as they don't represent scheduling constraints any longer.

The game is lost (i.e., the schedule is infeasible) if some node reaches the second quadrant.

It is won if no node remains on the board.

Game-board representation



When tasks are released, they are inserted into the game board according to their WCET and laxity (= deadline – remain. comp. time).

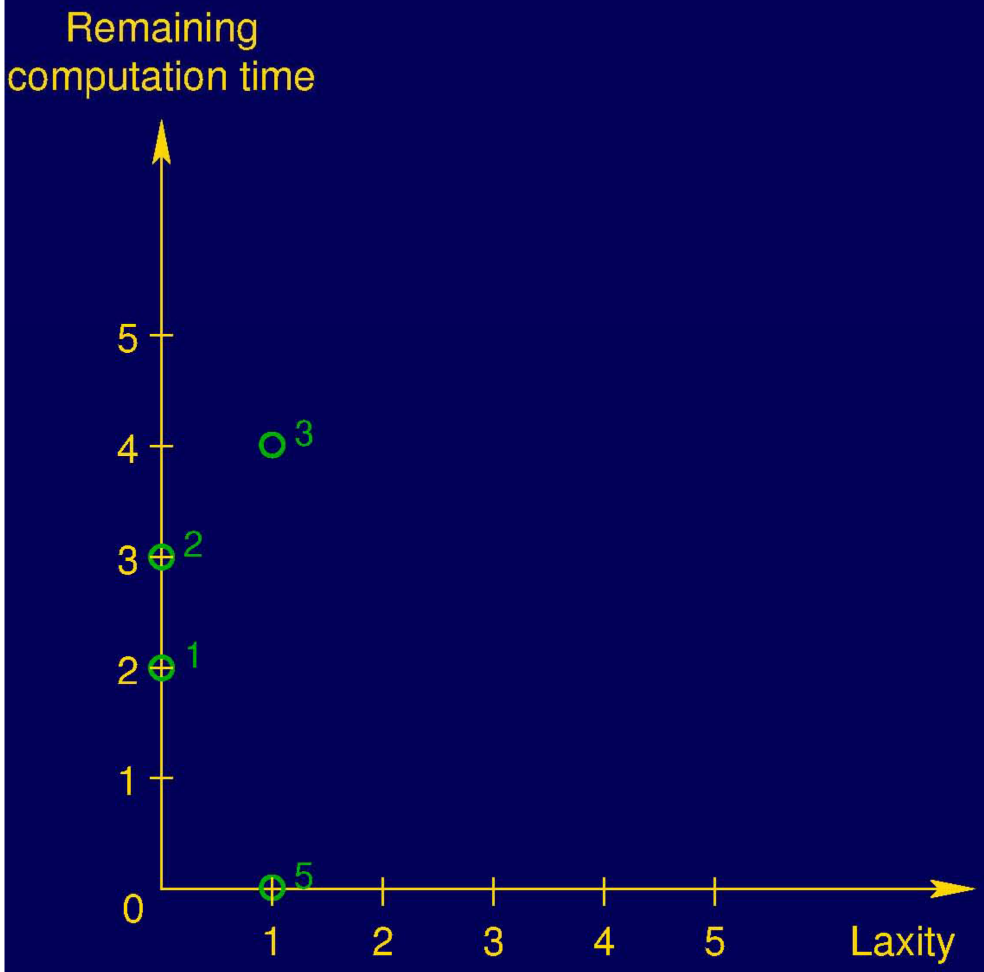
In every time scheduling step / turn of the game:
– at most n nodes go down by 1
– the rest moves 1 to the left

Nodes reaching the x–axis have been allocated all the computation time they need and are thus removed from the game board, as they don't represent scheduling constraints any longer.

The game is lost (i.e., the schedule is infeasible) if some node reaches the second quadrant.

It is won if no node remains on the board.

Game-board representation



When tasks are released, they are inserted into the game board according to their WCET and laxity (= deadline – remain. comp. time).

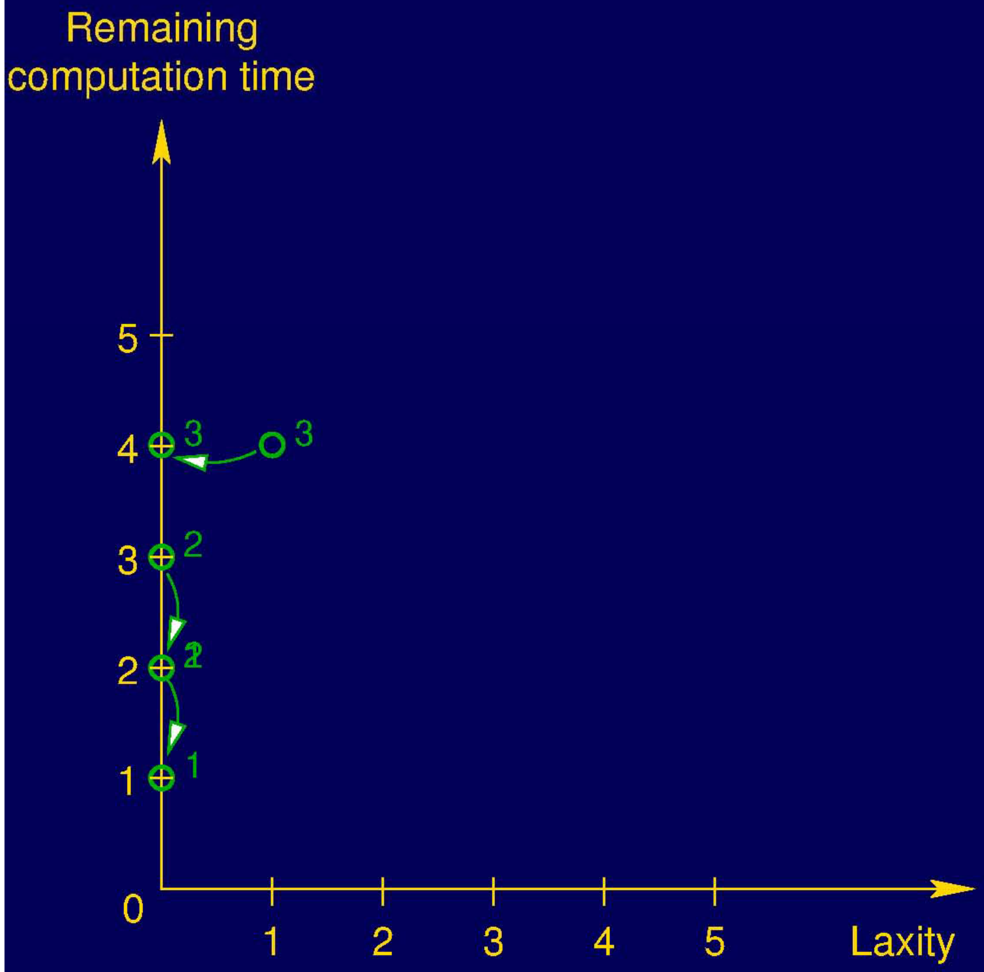
In every time scheduling step / turn of the game:
– at most n nodes go down by 1
– the rest moves 1 to the left

Nodes reaching the x–axis have been allocated all the computation time they need and are thus removed from the game board, as they don't represent scheduling constraints any longer.

The game is lost (i.e., the schedule is infeasible) if some node reaches the second quadrant.

It is won if no node remains on the board.

Game-board representation



When tasks are released, they are inserted into the game board according to their WCET and laxity (= deadline – remain. comp. time).

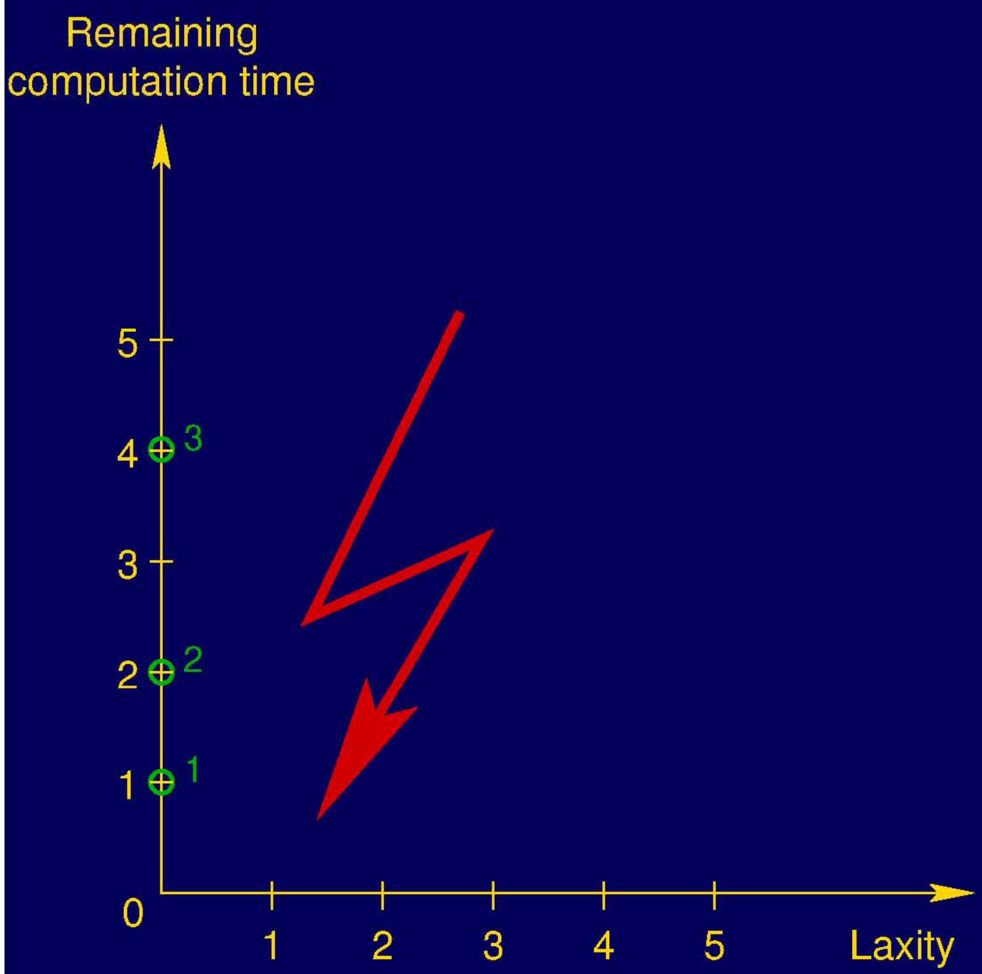
In every time scheduling step / turn of the game:
- at most n nodes go down by 1
- the rest moves 1 to the left

Nodes reaching the x–axis have been allocated all the computation time they need and are thus removed from the game board, as they don't represent scheduling constraints any longer.

The game is lost (i.e., the schedule is infeasible) if some node reaches the second quadrant.

It is won if no node remains on the board.

Game-board representation



When tasks are released, they are inserted into the game board according to their WCET and laxity (= deadline – remain. comp. time).

In every time scheduling step / turn of the game:
– at most n nodes go down by 1
– the rest moves 1 to the left

Nodes reaching the x–axis have been allocated all the computation time they need and are thus removed from the game board, as they don't represent scheduling constraints any longer.

The game is lost (i.e., the schedule is infeasible) if some node reaches the second quadrant.

It is won if no node remains on the board.

Extensions

- **Resource conflicts:** restricted move rules
- **Precedence constraints:** restricted move rules
- **Periodic tasks:** opponent‘s moves insert new nodes; game won if no task ever reaches second quadrant

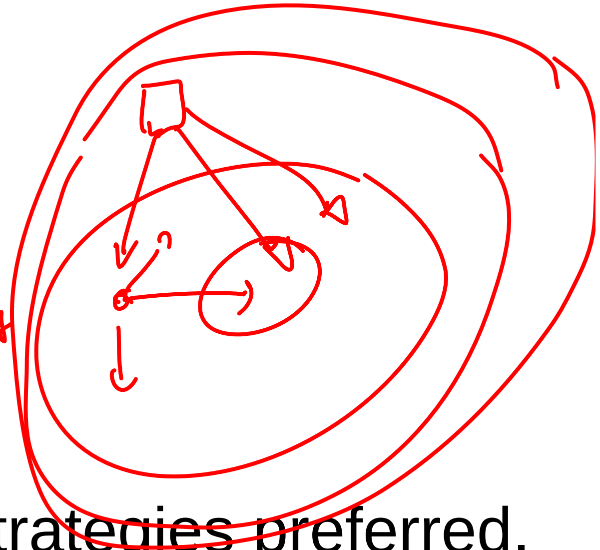
Game-theoretic solution

Theorem: In games with

- **finitely many positions** on the game board, and
- **complete** information

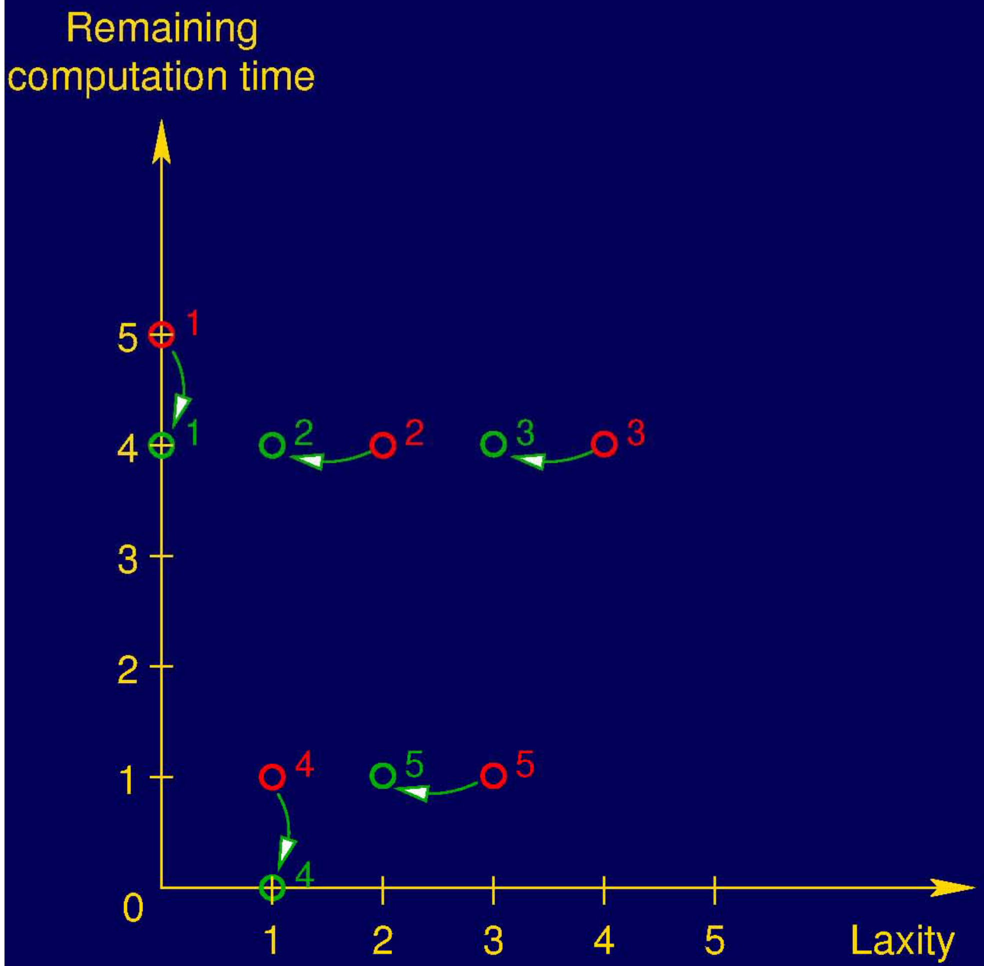
there is a always a winning strategy for one of the two players; it can be constructed effectively.

Fixpoint construction:
- Start with the winning positions
- Add all positions where we can move into set
- Add all positions where the opponent must move into set
- repeat until no more change



However: high complexity ⇒ predefined strategies preferred.

LLF (Least Laxity First)

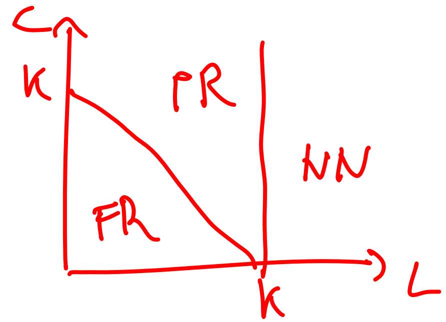


When tasks are released, they are inserted into the game board according to their WCET and laxity (= deadline – remain. comp. time).

In every time scheduling step / turn of the game:
– at most n nodes go down by 1
– the rest moves 1 to the left

LLF is optimal.

Schedulability



Within a set M of aperiodic tasks, we identify three classes with respect to the next k time units starting at time t:

1. Tasks that have to be fully run within the next k time units:

$$FR(t,k) = \{i \in M \mid D_i(t) \leq k\}$$

2. Tasks that have to be partially run within the next k time units:

$$PR(t,k) = \{i \in M \mid L_i(t) \leq k \wedge D_i(t) > k\}$$

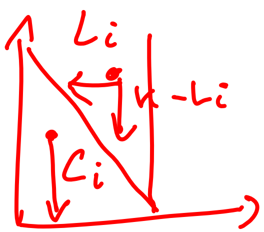
3. Tasks that need not be run within the next k time units:

$$NN(t,k) = \{i \in M \mid L_i(t) > k\}$$

Surplus computing power

$$SCP(t,k) = \underbrace{kn}_{\text{avail. computing power}} - \underbrace{\sum_{i \in FR(t,k)} C_i(t)}_{\text{needed for full runs}} - \underbrace{\sum_{i \in PR(t,k)} (k - L_i(t))}_{\text{needed for partial runs}}$$

Lemma: SCP(0,k)≥0 for all k>0 is a **necessary** condition for schedulability.



• total amount of available processor time: kn

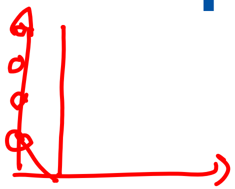
Executing FR(0,k) requires $\sum_{i \in FR(0,k)} C_i$

Tasks in PR(0,k) may move horizontally for at most L_i steps, must move downward during the rest

=> requires $\sum_{i \in PR(0,k)} (k - L_i)$

Surplus computing power

$SCP(t,k) = kn - \sum_{i \in FR(t,k)} C_i(t) - \sum_{i \in PR} k - L_i(t)$



1 **Theorem:** If all tasks are released at time 0, then SCP(0,k)≥0 for all k>0 is a **necessary and sufficient** condition for schedulability.

We show that no deadline is missed using LLF.

Proof by induction over t:

1) $SCP(t,k) \geq 0 \;\forall k \Rightarrow$ # of token on the C-axis is $\leq n$.
2) $SCP(t,k) \geq 0 \;\forall k$ implies that $SCP(t+1,k) \geq 0 \;\forall k$ (if LLF is used).

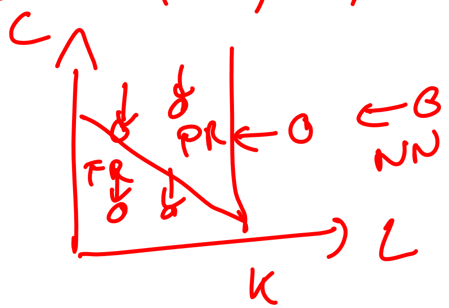
Ad 1): $SCP(t,1) \geq 0 \Rightarrow n \geq \underbrace{\sum_{i \in FR(t,1)} C_i(t) + \sum_{i \in PR(t,1)} (1 - L_i(t))}_{\text{\# of token on C-axis.}}$

$$SCP(t,k) = kn - \sum_{i \in FR(t,k)} C_i(t) - \sum_{i \in PR(t,k)} (k - L_i(t))$$

Ad 2) : Claim For all $k \geq 0$ $\exists$ k' s.t.

$$SCP(t+1,k) \geq SCP(t,k')$$

<u>Case 1:</u>



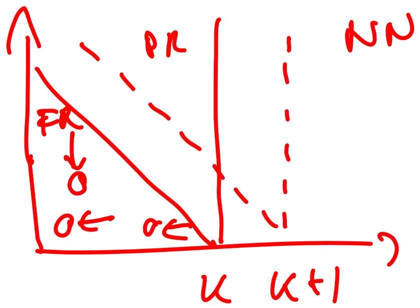
All token left of $L=k$ arrived by vertical move

Pick $k' = k$

$\Delta SCP = SCP(t+1,k) - SCP(t,k)$

- Token is in FR : add 1 to ΔSCP
- Token is in PR : contribution to ΔSCP is 0.
- Token that moved from NN to the line $L=k$. $k - L_i(t) = 0$: contribution to ΔSCP is 0.
- Token that moved from PR to $L+C=k$ in FR $k - L_i(t+1) = C_i(t+1) = k - L_i(t)$: contribution to ΔSCP is 0.

Case 2: Some of the token left of $L=k$ have arrived by horizontal move.



$SCP(t,k) = kn - \sum_{i \in FR} C_i(t) - \sum_{i \in PR} k - L_i(t)$

Pick $k' = k+1$: $\Delta SCP = SCP(t+1, k) - SCP(t, k+1)$

By LCF, n token must have moved downward left of $L = k+1$,

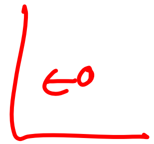
Consider a token that has moved downward

in PR $\Rightarrow$ contributes $-(k - L_i(t+1)) + (k+1 - L_i(t))$
$= 1$

in FR $\Rightarrow$ contributes 1

Consider a firm that has moved horizontally:

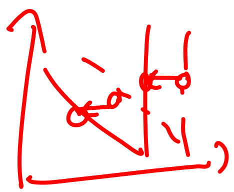
has in PR $\Rightarrow$



$$SCP = kn - \sum_{FR} c_i - \sum_{PR} k - L_i$$

$$-(k - L_i(t+1)) + (k+1 - L_i(t))$$

$$= 0$$

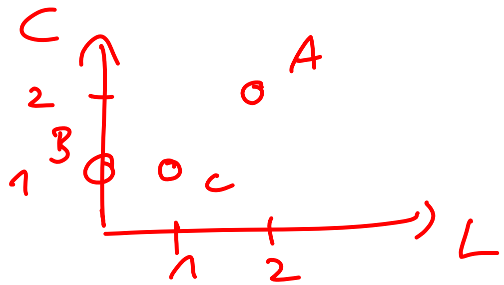


$$\Rightarrow \Delta SCP \geq kn - (k+1)n + \text{at least } n$$

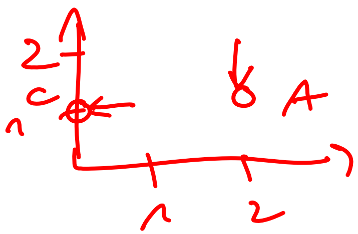
$$\geq 0.$$

Online scheduling?

Theorem: There can be no optimal scheduling algorithm if the release times are not known a priori.

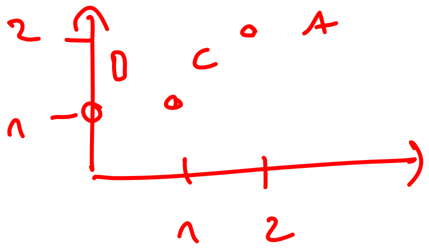


Case 1: Scheduler selects A + B



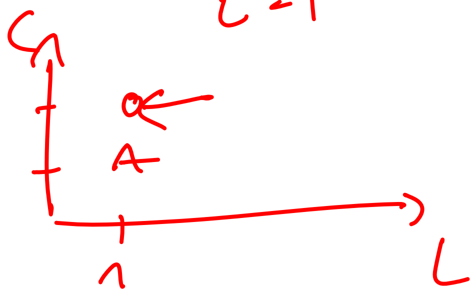
New tasks D, E with L = 0, C = 1 arrive at t = 1
=> no schedule

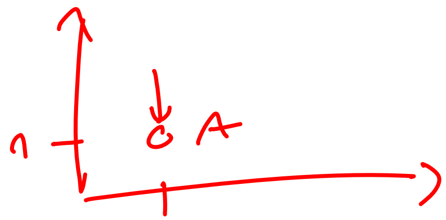
Feasible schedule: B + C, D + E, A



Case 2: Scheduler selects B + C

t=1 t=2





New tasks D, E arrive with L=0, C=2 at t=2

↯ No schedule

Feasible schedule: A+B, A+C, D+E, D+E

Periodic periodic tasks

Theorem: A necessary and sufficient condition for the schedulability of periodic tasks is that $U \leq n$.

necessary ✓.

Scheduling idea

1. Divide the time line into time slices such that each period of each process is divided into an integral number of time slices.
   Slice length $T = \text{GCD}(T_1, \ldots, T_n)$.
2. Within each time slice, allocate processor time in proportion to the utilization $U_i = \frac{C_i}{T_i}$ originating from the various tasks.
   Processing time per slice $r_i = TU_i = T\frac{C_i}{T_i}$.
   Hence, each task runs $\frac{T_i}{T} r_i = \frac{T_i}{T} T \frac{C_i}{T_i} = C_i$ time units within its period.
3. Allocate r_i according to the following algorithm
   (a) Look for the first processor proc_j that has free capacity in its time slices.
   (b) Allocate that portion of r_i to proc_j that proc_j can accommodate.
   (c) If all of r_i has been allocated then proceed with the next task (goto step a).
   (d) Otherwise allocate the remainder of r_i to proc_{j+1}.
   proc_{j+1} has enough spare capacity as it has not previously been used and $r_i \leq T$ due to $U_i \leq 1$. Furthermore, due to $r_i \leq T$, we don't generate temporal overlap between the two partial runs of task i.