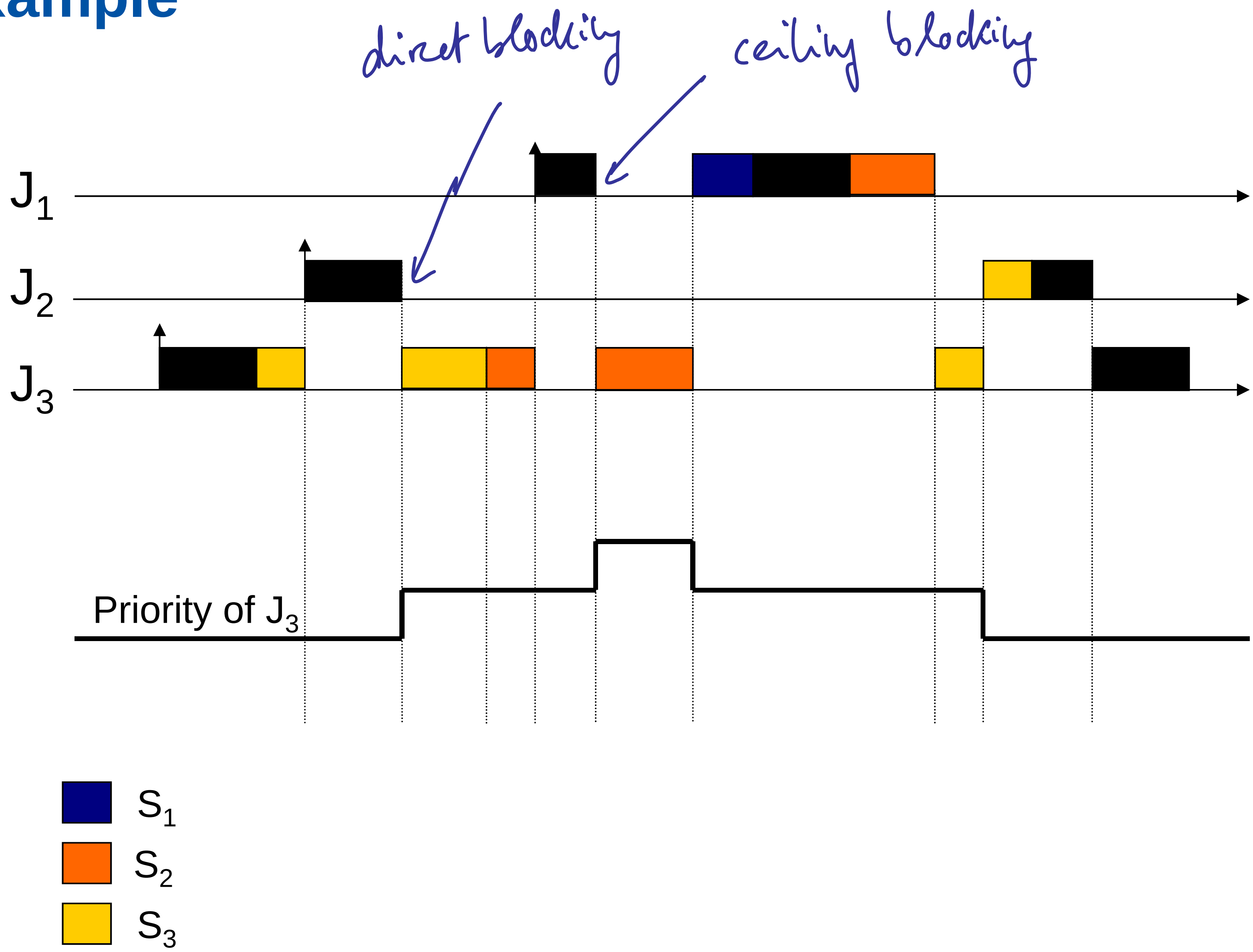
Embedded Systems 19

REVIEW: Priority Ceiling Protocol

- The processor is assigned to a ready job J with highest priority.
- To **enter** a critical section, J needs priority > C(S*),
  where S* is the currently locked semaphore with max C.
  $\rightarrow$ otherwise J „blocks on semaphore“ and
  priority of J is inherited by job J‘ holding S*.
- When J‘ **exits** critical section, its priority is updated to the highest priority of some job that is blocked by J‘ (or to the nominal priority if no such job exists).

Example



direct blocking
ceiling blocking
J1
J2
J3
Priority of J3
S1
S2
S3

Priority Ceiling Protocol

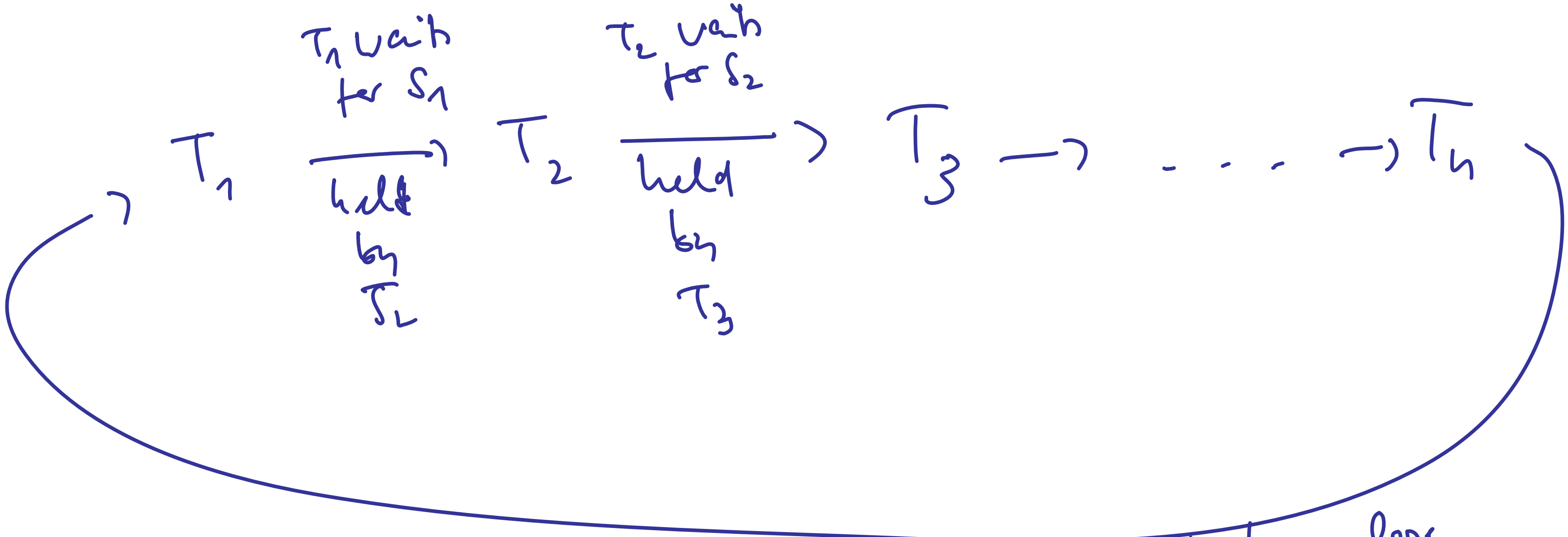
Theorem (Sha/Rajkumar/Lehoczky): Under the Priority Ceiling Protocol, a job can be blocked by at most one lower priority task.

- Suppose τ_i is blocked by τ_1, τ_2 with <u>lower</u> priority.
- Let τ_1 be the task that enters the critical region first.
- Let $C_1^* = \max_{S \text{ locked by } \tau_1} C(S)$
- When τ_2 enters the critical section, $P_2 > C_1^*$
- If τ_i is blocked by τ_1, $P_i \leq C_1^*$

Hence, $P_i \leq C_1^* < P_2$ ↯

Priority Ceiling Protocol

The Priority Ceiling Protocol prevents deadlocks.



Let S_j be the last semaphore acquired to close the circle

$\Rightarrow$ task $J_{j \oplus 1}$ must have priority $> C(S_k)$
$\forall k = 1 .. j-1, j+1, ... n$

↯ Impossible, because $J_{j \oplus 1}$ can lock $S_{j \oplus 1}$

Incorporating aperiodic tasks

- In real systems, not all tasks are periodic
  - Environmental events to be processed
  - Exceptions raised
  - Background tasks running whenever CPU time budget permits
- Thus, real systems tend to be a **combination** of
  - periodic and
  - aperiodic tasks

  and of
  - hard real-time and
  - soft real-time tasks.

Aperiodic and periodic tasks together (1)

- **Aperiodic and periodic tasks together**
  - can be handled by dynamic-priority schedulers like EDF
- **Problem:**
  - Off-line guarantees can not be given without assumptions on aperiodic tasks.
  - If deadlines for aperiodic tasks are hard, aperiodic tasks need to be characterized by a minimum interarrival time between consecutive instances
    $\Rightarrow$ bounds on the aperiodic load
  - Aperiodic tasks with maximum arrival rate may be modeled as periodic tasks with this rate

  $\Rightarrow$ periodic scheduling
  - Aperiodic tasks with maximum arrival rate are called sporadic tasks.

Aperiodic and periodic tasks together (2)

- Other solutions for the case that periodic tasks have hard deadlines, aperiodic tasks have soft deadlines.
  - Simplest solution: **Background scheduling**
    - Aperiodic tasks are only executed when no periodic task is ready
    - Guarantees for periodic tasks do not change
    - Only applicable when load is not too high
  - **Other solutions:**
    - Define new periodic tasks, a so-called **server**
    - Aperiodic tasks are executed during “execution time” of server process
    - Independent scheduling strategies possible for periodic tasks and aperiodic tasks “inside the server”

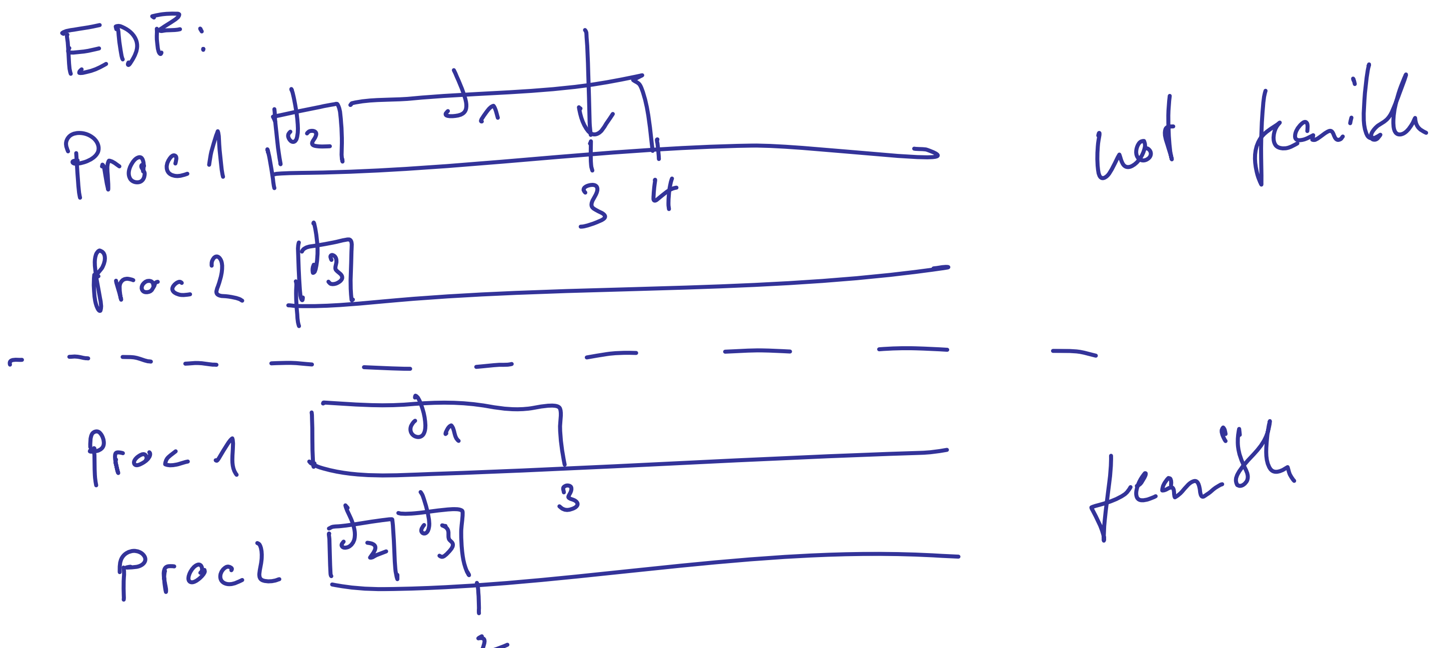
Multiprocessor scheduling

EDF with multiple processors?

$C_1 = 3, \; d_1 = 3$

$C_2 = 1, \; d_2 = 2$

$C_3 = 1, \; d_3 = 2$



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:

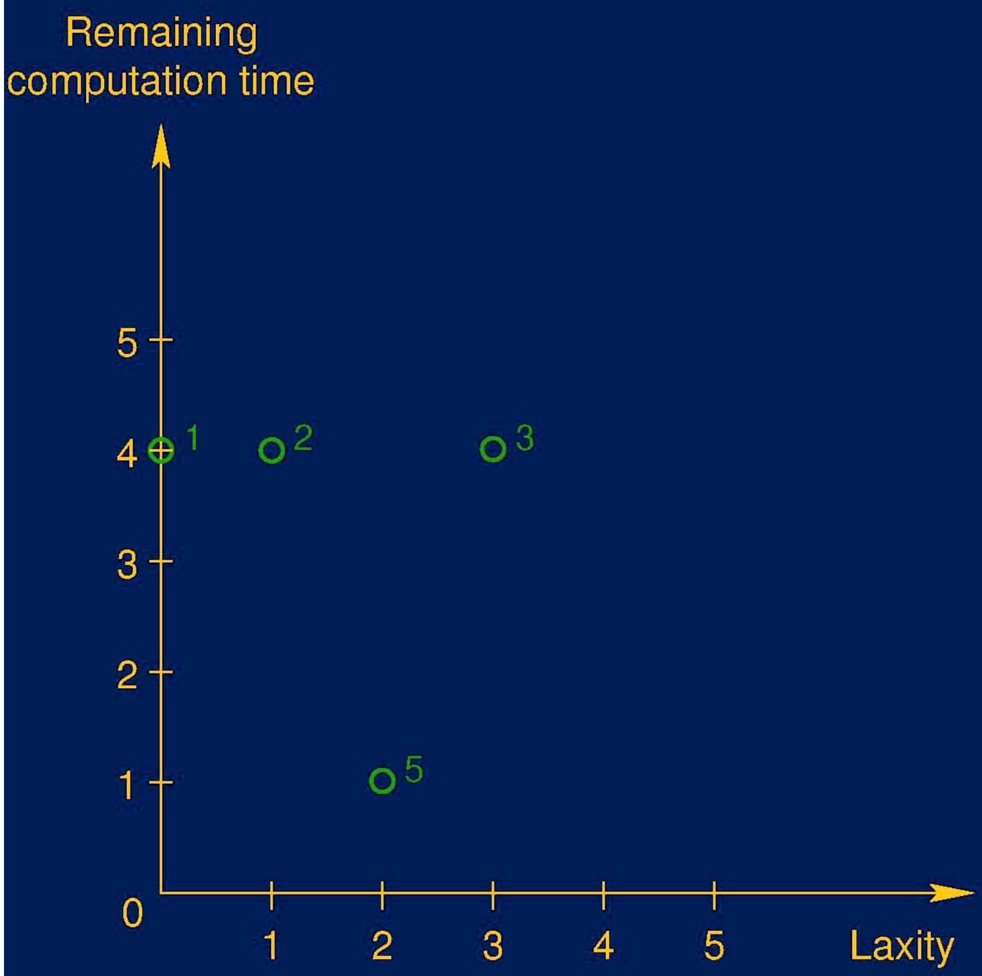
- Tasks can freely be migrated between processors
  - at any integer time instant, without overhead
  - however: no task may run on two processors simultaneously
- All tasks are preemptable
  - at any integer time instant, without overhead

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).

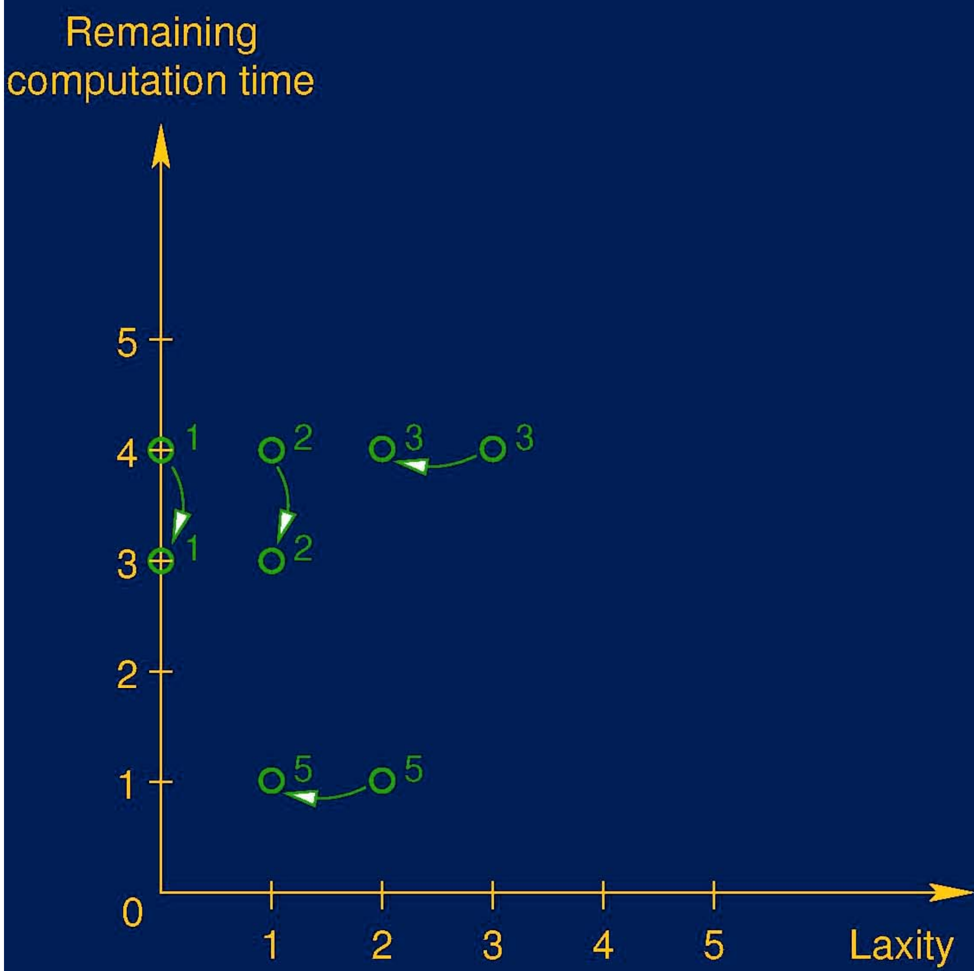
In every time scheduling step / turn of the game:
– at most n nodes go down by 1
– the rest moves 1 to the left

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).

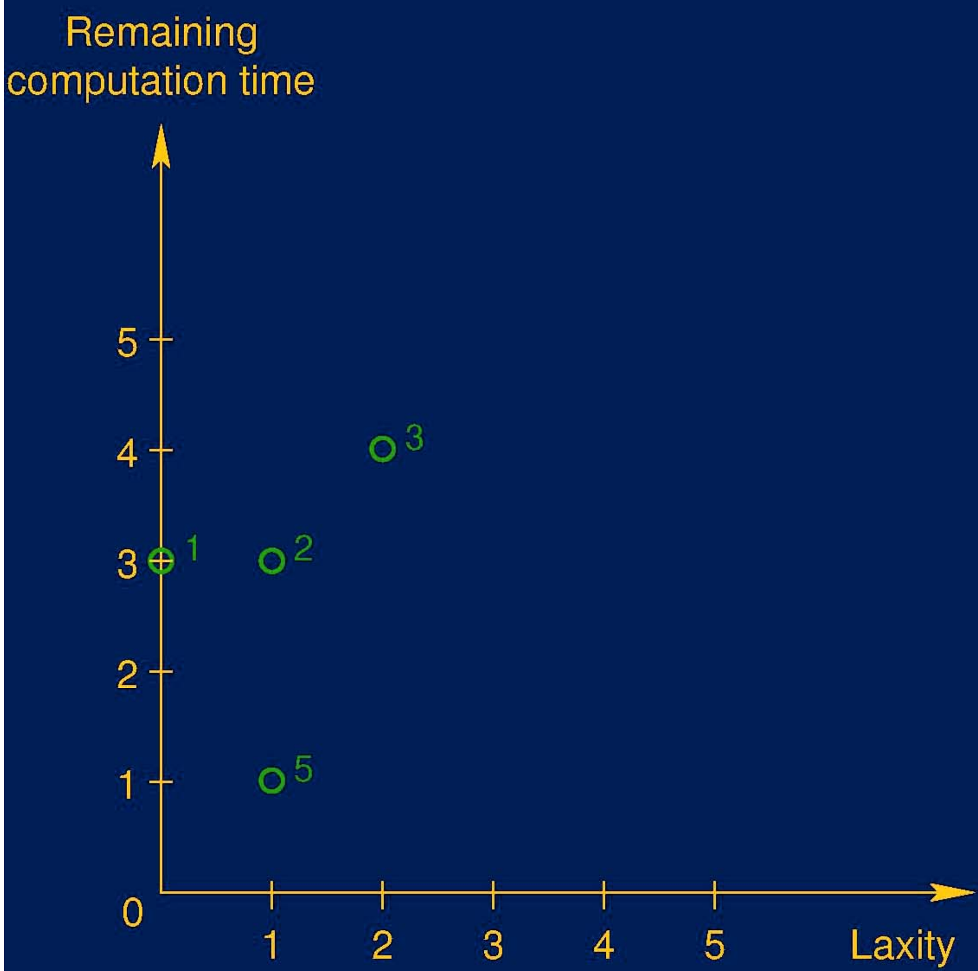
In every time scheduling step / turn of the game:
– at most n nodes go down by 1
– the rest moves 1 to the left

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).

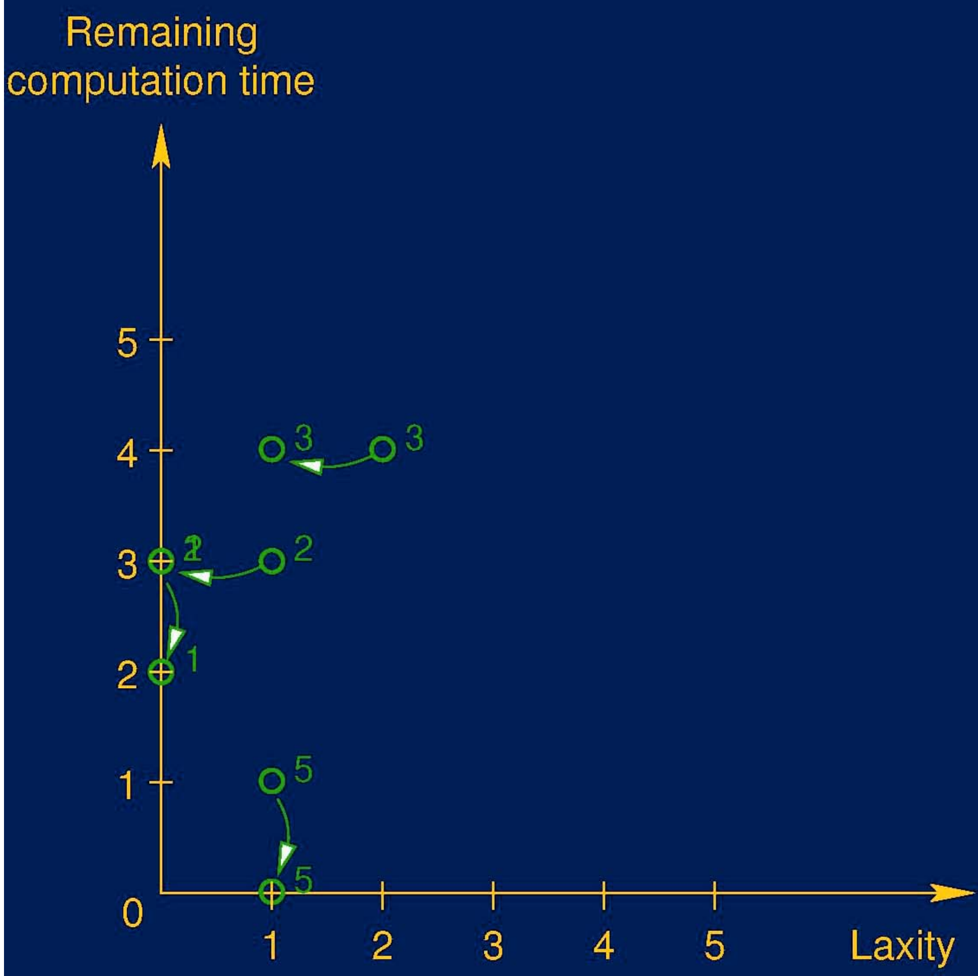
In every time scheduling step / turn of the game:
– at most n nodes go down by 1
– the rest moves 1 to the left

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).

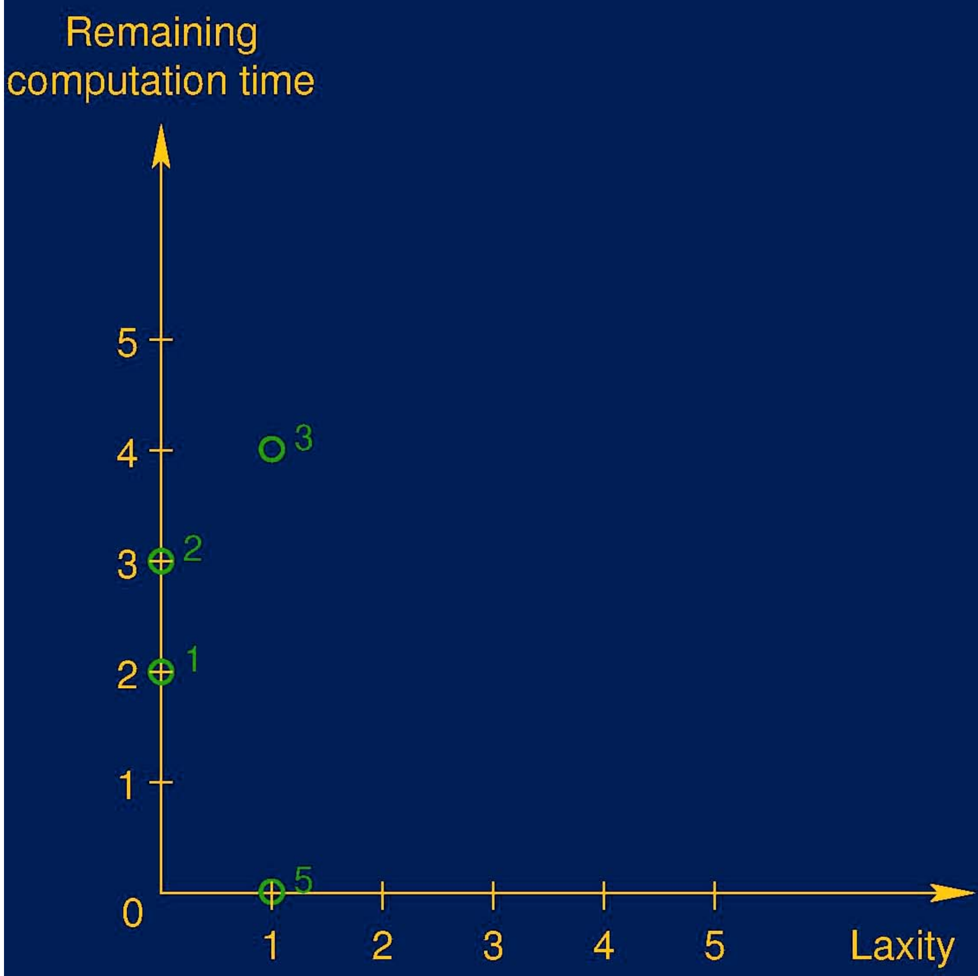
In every time scheduling step / turn of the game:
– at most n nodes go down by 1
– the rest moves 1 to the left

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).

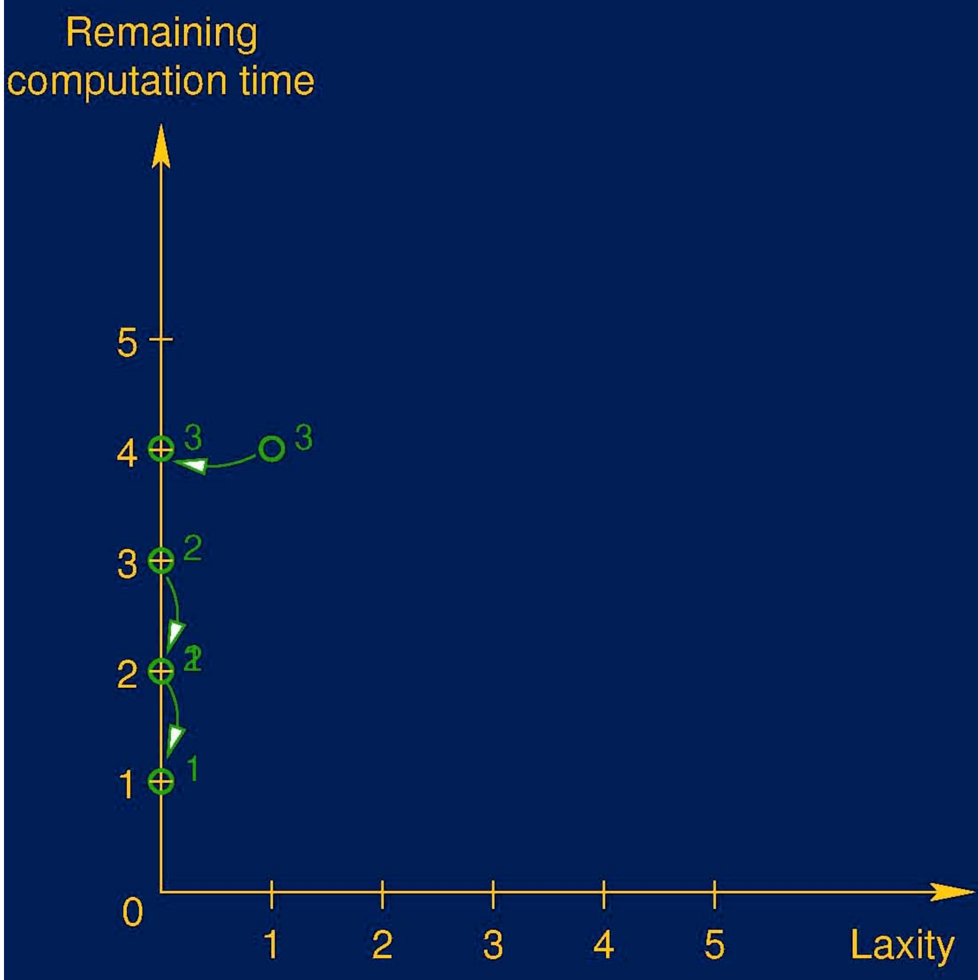
In every time scheduling step / turn of the game:
– at most n nodes go down by 1
– the rest moves 1 to the left

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).

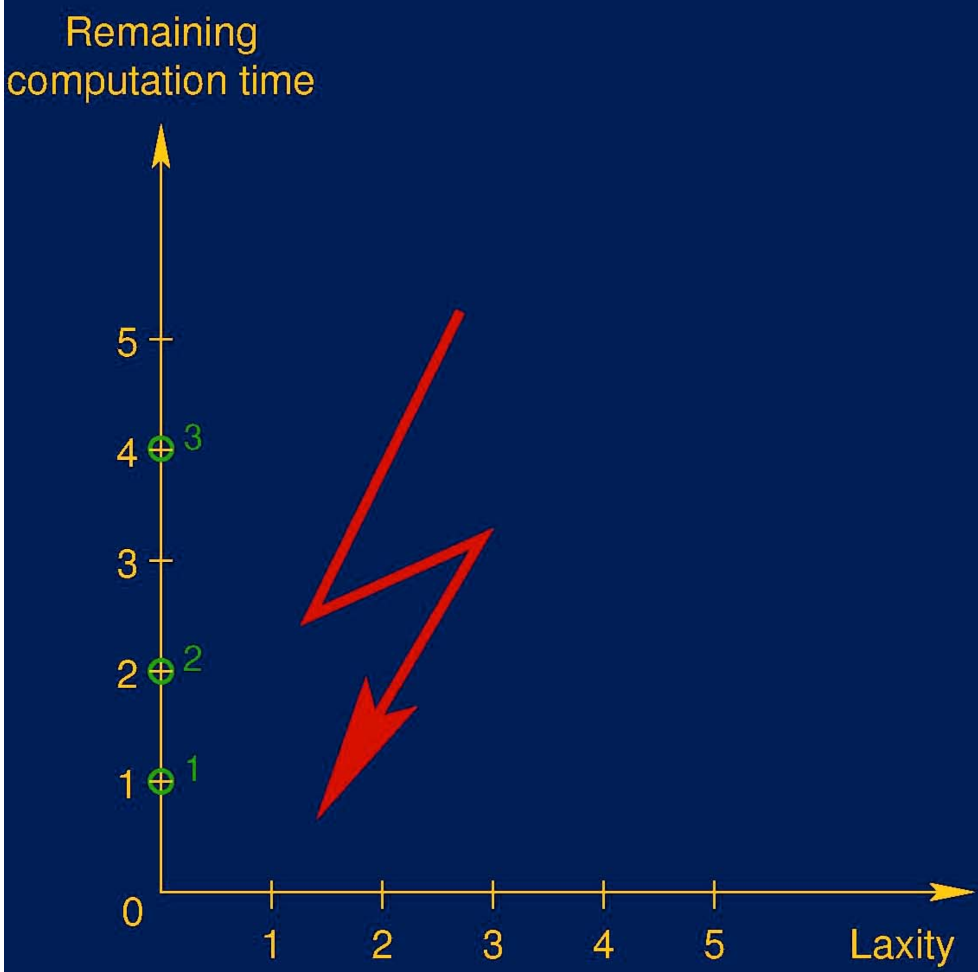
In every time scheduling step / turn of the game:
– at most n nodes go down by 1
– the rest moves 1 to the left

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

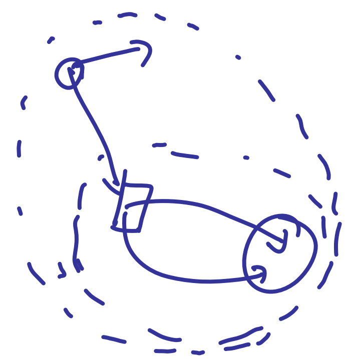
Game-theoretic solution

Theorem: In games with

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

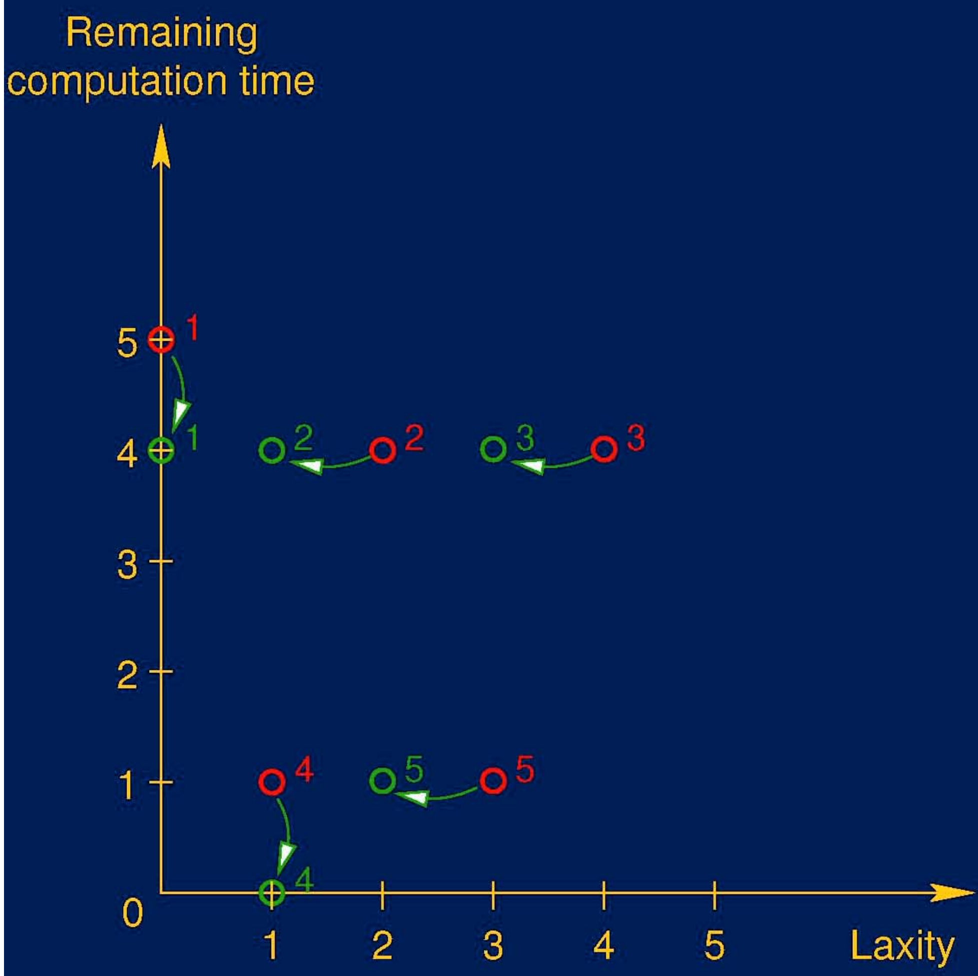
there is a always a winning strategy for one of the two players; it can be constructed effectively.

- Start with the losing positions
- Add all positions where we cannot avoid moving into this set
- Add all positions wher the opponent can move into this set
- Repeat until no more change



However: high complexity $\Rightarrow$ predefined strategies preferred.

LLF (Least Laxity First)

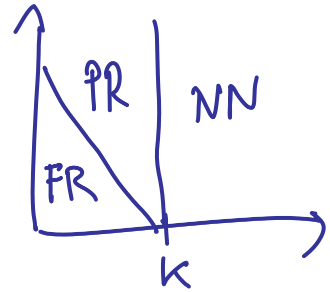


When tasks are released, they are inserted into the game board according to their WCET and laxity (= deadline – remain. comp. time).

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:

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

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

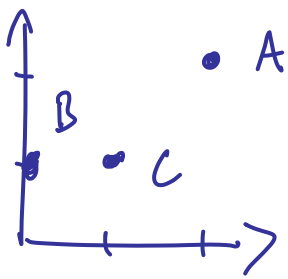
$$SCP(t,k) = \underbrace{kn}_{\text{avail. computing power}} - \underbrace{\sum_{i \in FR(t,k)} C_i(t)}_{\text{needed for full runs}} - \underbrace{\sum_{i \in PR(t,k)} (k - L_i(t))}_{\text{needed for partial runs}}$$

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

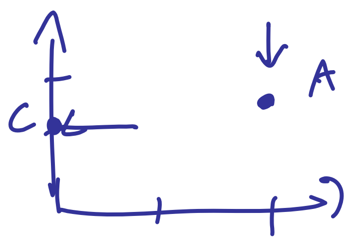
- Executing FR(0,k) requires $\sum_{i \in FR(0,k)} C_i$
- Executing PR(0,k) requires $\sum_{i \in PR(0,k)} (k - L_i(0))$
- kn total computing power in k steps

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

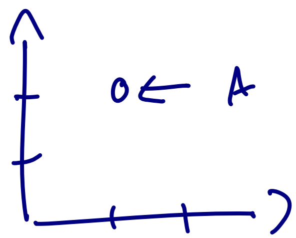


New tasks D, E arrive with L = 0, c = 1 at t = 1

=> no schedule

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

Case 2: Schedule selects B + C



Now tasks D, E arrive with

L = 0, C = 2 at t = 2

↯ no schedule

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