Multiprocessor On-Line Scheduling of Hard-Real-Time Systems

Michael Dertouzos and Aloysius Mok

Published in

IEEE Transactions on Software Engineering, Vo. 15, No. 12, Dec 1989

Gurulingesh Raravi

CISTER/ISEP

29/10/2010

Overview

- Significance of the Paper
- Past Results
- The Scheduling Game Representation
- Uniprocessor Scheduling
  - Optimality of EDF
- On-Line Multiprocessor Scheduling
  - Why EDF is not optimal
  - The Insufficient Knowledge Problem
- Conclusions

Significance of the Paper

- The paper showed that it is impossible to design an optimal online algorithm for multiprocessor scheduling
  - In other words, *a priori* knowledge of all of the following parameters is essential for designing an optimal multiprocessor scheduling algorithm:
    1. Deadlines
    2. Computation times and
    3. Start-times

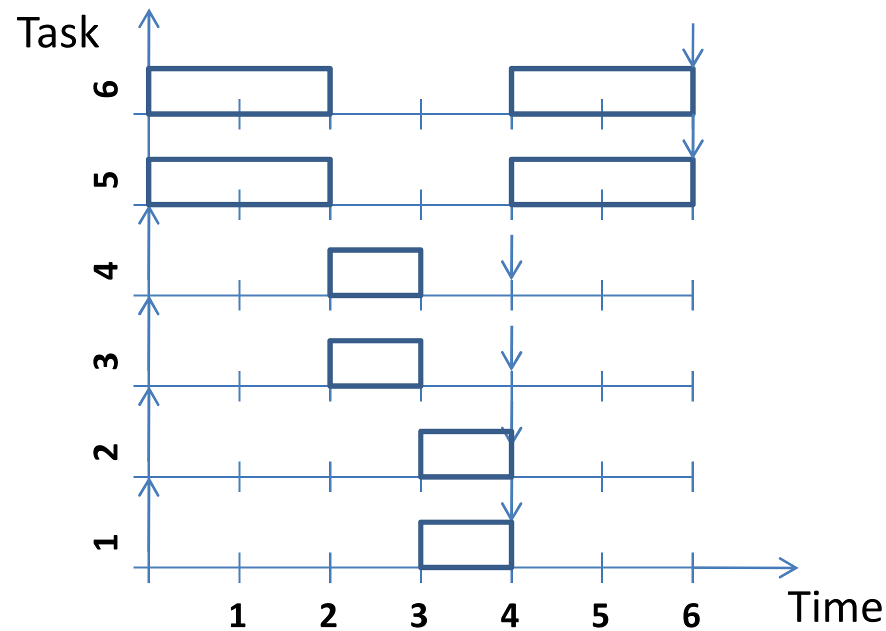
Past Results

- Uniprocessor:
  - Liu/Layland's sufficient and necessary condition for scheduling periodic task sets
  - EDF shown to be optimal for scheduling arbitrary task sets (not necessarily periodic) by Dertouzos
- Multiprocessors:
  - EDF is not optimal
  - Optimal scheduling algorithms for two processor by Garey and Johnson
  - The scheduling problem often becomes intractable for more than two processors
    - Except two special cases
    - However, the algorithms for those exceptions are not optimal anyway (when used online).

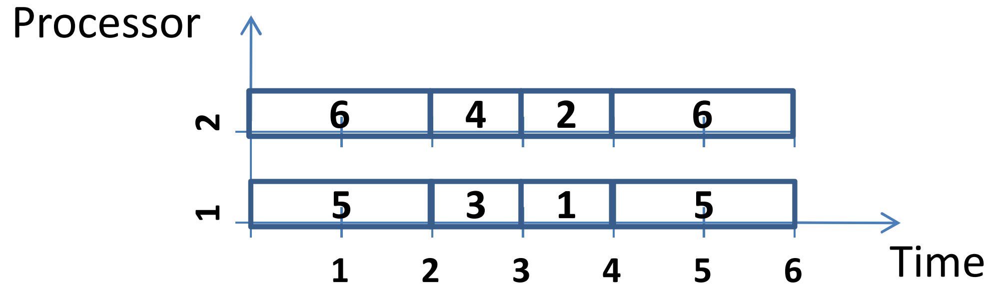
Scheduling Game Representation (1/6)

- Well-known (previous) representations:

1. Timing diagram



2. Gantt chart

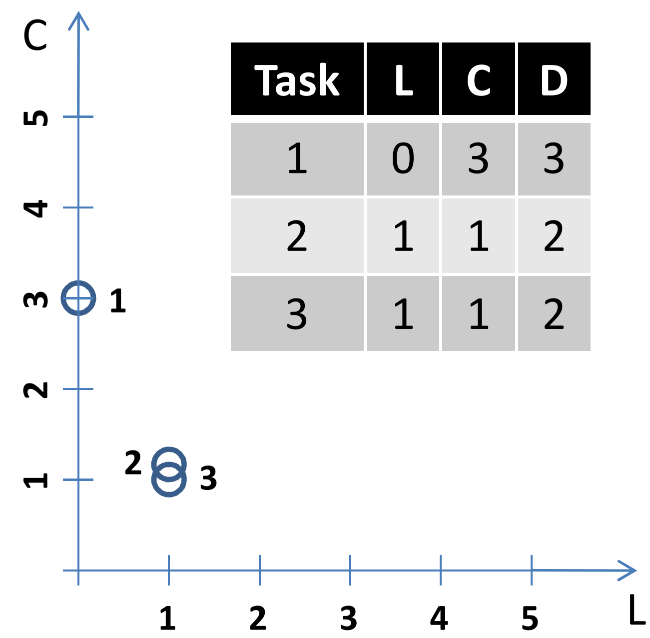


Scheduling Game Representation (2/6)

- Few Notations:
  - The status of each task whose start-time has elapsed can be characterized at time=i by:
    - Remaining Computation: C(i) and
    - Deadline: D(i)
  - *Laxity* of a task at time=i:
    - L(i) = D(i) – C(i)
    - Laxity is a measure of task's urgency. A task with:
      - zero laxity => execute immediately without interruption
      - negative laxity => a deadline will be missed

Scheduling Game Representation (3/6)

- The scheduling problem at time=i can be modelled by configuration of "tokens" in the first quadrant of Cartesian plane:
  - Y-axis: C
  - X-axis: L
  - Token: represents a task
- Task `j´ with $C_j(i)$ and $L_j(i)$:
  - $L = L_j(i)$ and $C = C_j(i)$



Task	L	C	D
1	0	3	3
2	1	1	2
3	1	1	2

Scheduling Game Representation (4/6)

- Consider *m* tasks and *n* processors ($m > n$)
  - At most *n* tasks can be executed at a time
- On L-C plane: scheduling corresponds to moving:
  - *n* tokens one step downwards
    - L(i+1) = L(i), C(i+1) = C(i) - 1
  - Rest (*m*-*n* tokens) one step leftwards
    - L(i+1) = L(i) - 1, C(i+1) = C(i)
  - Scheduling algorithm decides the direction of token movement at each step
  - If a token reaches
    - 2^{nd} quadrant => algorithm failed
    - L-axis (horizontal axis) => task met deadline

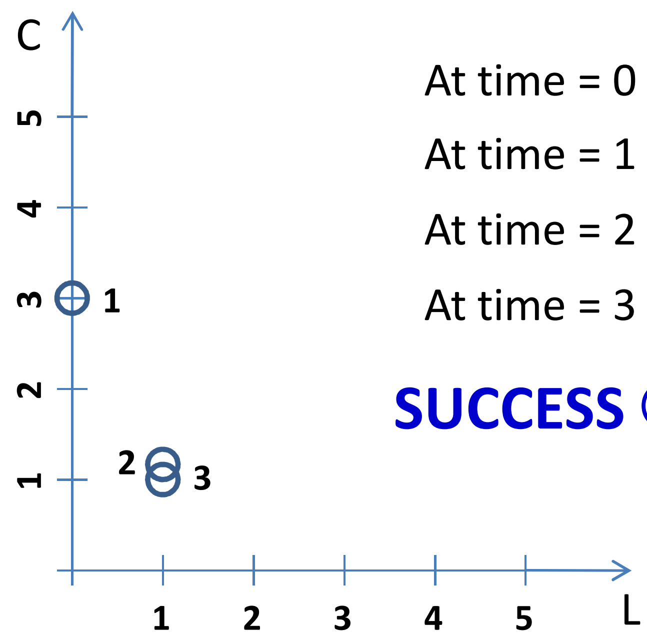
Scheduling Game Representation (5/6)

- A schedule can be simulated by a sequence of configurations of tokens on the L-C plane:
  1. Initial configuration of *m* tokens in Q-1
  2. At each step, at most *n* tokens are moved one step downwards and the rest one step leftwards
  3. A token that reaches L-axis (X-axis) is ignored
  4. A scheduler fails if a token enters Q-2
  5. The scheduler wins if all tokens eventually reach L-axis without entering Q-2

Scheduling Game Representation (6/6)

- An Example:
  - n=2 (processors), m=3 (tasks)

Task	L	C	D
1	0	3	3
2	1	1	2
3	1	1	2



At time = 0

At time = 1

At time = 2

At time = 3

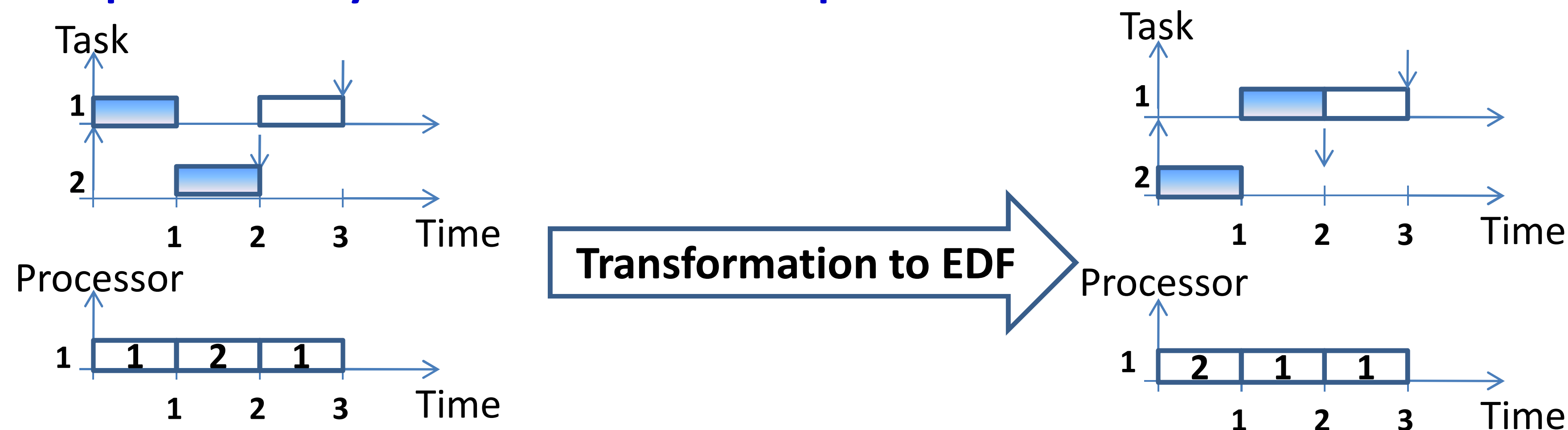
SUCCESS ☺

EDF Scheduling Properties (1/3)

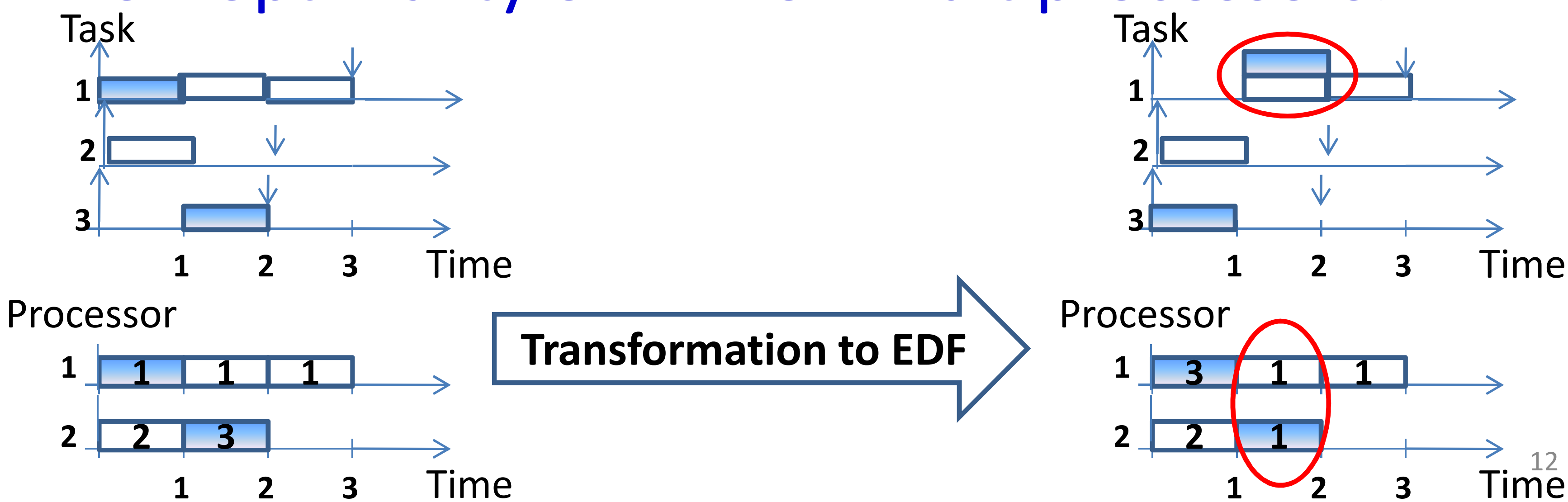
- Uniprocessor
  - Optimal scheduling algorithms:
    - Earliest Deadline First (EDF), Least Laxity first (LLF)
  - The optimality of EDF is proven by Dertouzos
    - by showing that a feasible schedule can always be transformed into EDF schedule
      - If at any time the processor executes some task other than the one which has the closest deadline, then it is possible to interchange their order of execution
- Multiprocessor
  - EDF is not optimal

EDF Scheduling Properties (2/3)

- Optimality of EDF on Uniprocessor:



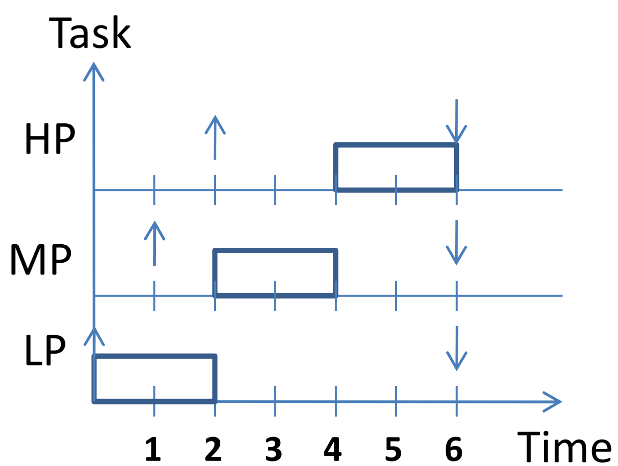
- Non-optimality of EDF on Multiprocessors:



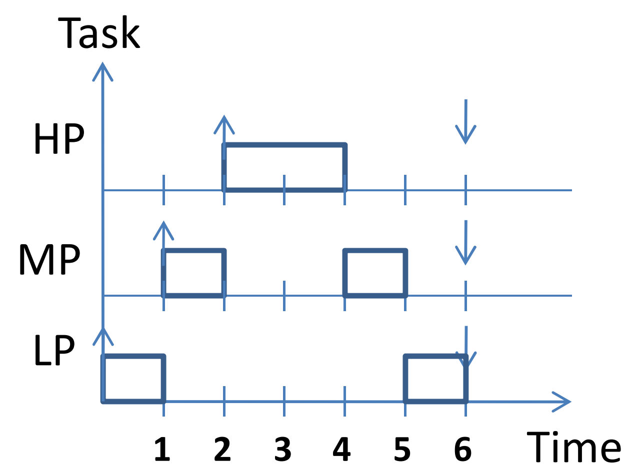
EDF Scheduling Properties (3/3)

- The system overhead due to context switching required by EDF is at most twice that required by any algorithm
  - Loading of a task is considered as context switch
  - An example to illustrate the concept:

A non-preemptive schedule



EDF schedule



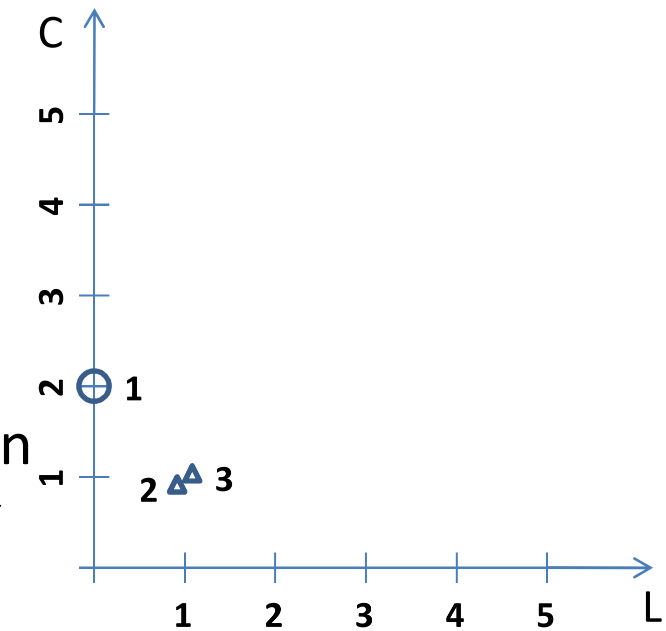
The Insufficient Knowledge Problem (1/7)

- Another interesting thing about (optimal) EDF is:
  - It is driven only by *D* and
  - *a priori* information about *C* or *S* not required
- Whether such an algorithm exists for MPs?
  - Unfortunately, NOT ☹
- No optimal algorithm can be designed for multiprocessors without *a priori* information of:
  1. Computation times
  2. Deadlines and
  3. Start-times

The Insufficient Knowledge Problem (2/7)

- Lemma: No optimal algorithm can exist if the computation time of tasks are not known *a priori*
  - An example: 2 processors, 3 tasks

Task	L	C	D
1	0	2	**2**
2	1	1	**2**
3	1	1	**2**



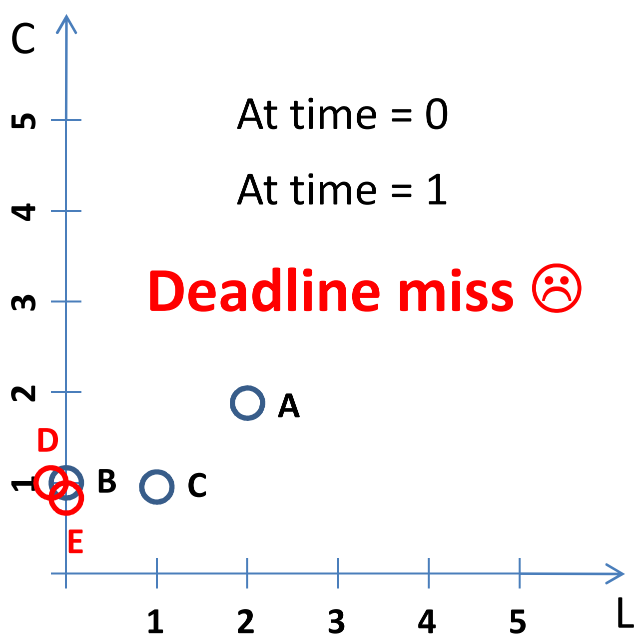
    - If scheduler picks task `j´ then we can always arrange our example so that `j´ is represented by triangular token

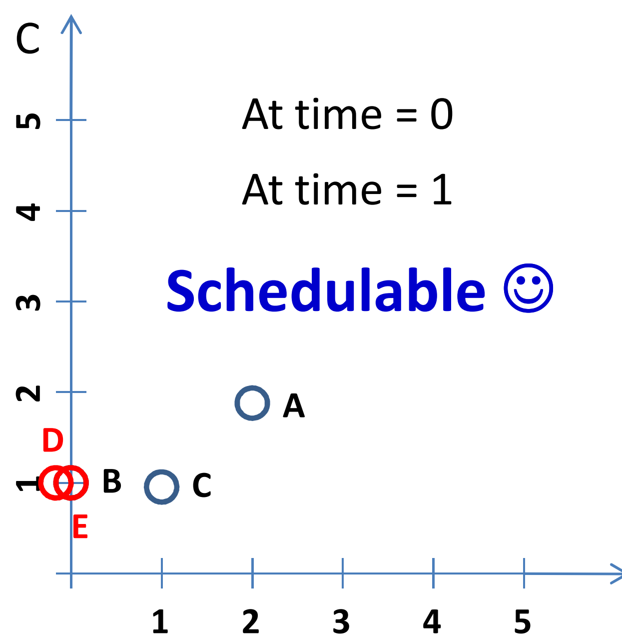
- Lemma: No optimal algorithm can exist if the deadlines of tasks are not known *a priori*

The Insufficient Knowledge Problem (3/7)

- Lemma: No optimal algorithm can exist if the start-times of tasks are not known *a priori*
  - An example: 2 processors, 3 tasks
  - Depending on scheduler decision, there are 3 cases:

Case-1: A and B are moved down at time=0

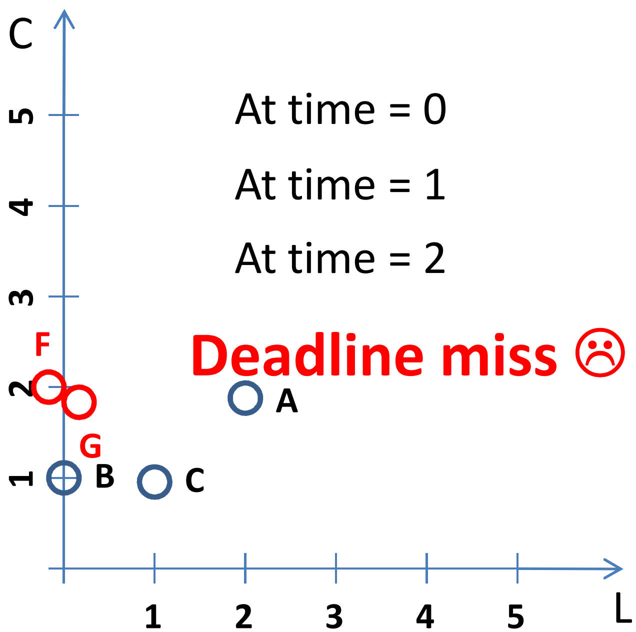


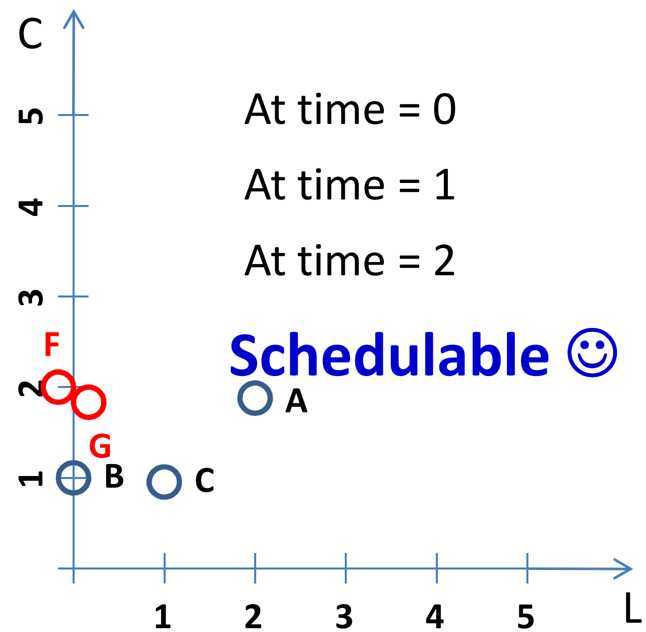


The Insufficient Knowledge Problem (4/7)

- Lemma: No optimal algorithm can exist if the <u>start-times</u> of tasks are not known *a priori*
  - An example: 2 processors, 3 tasks
  - Depending on scheduler decision, there are 3 cases:

Case-2: B and C are moved down at time=0

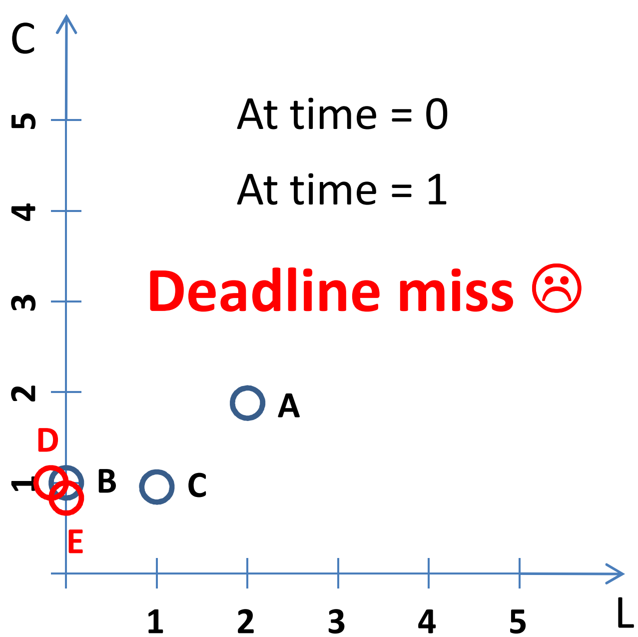


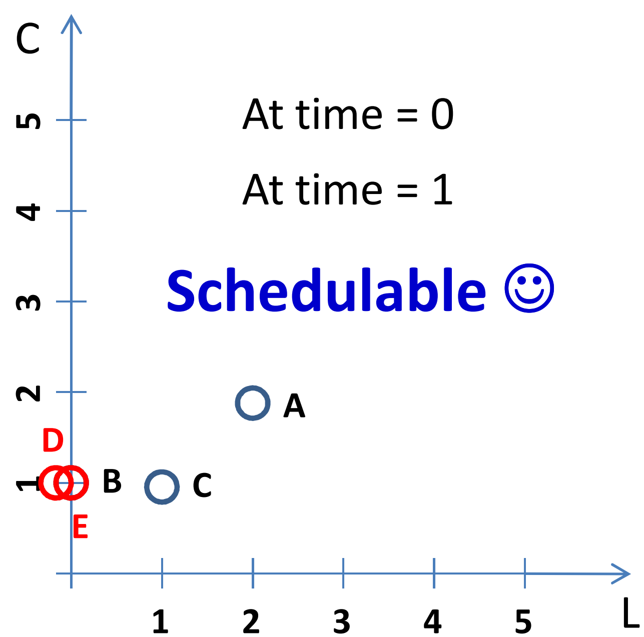


The Insufficient Knowledge Problem (5/7)

- Lemma: No optimal algorithm can exist if the start-times of tasks are not known *a priori*
  - An example: 2 processors, 3 tasks
  - Depending on scheduler decision, there are 3 cases:

Case-3: Only B is moved down at time=0





The Insufficient Knowledge Problem (6/7)

- The above reasoning can be generalized to more than two processors
  - since the extra processors can be kept busy by introducing zero-laxity tasks
  - Theorem: For two or more processors, no deadline scheduling algorithm can be optimal without complete *a priori* knowledge of:
    1. Deadlines
    2. Computation times and
    3. Start-times of the tasks

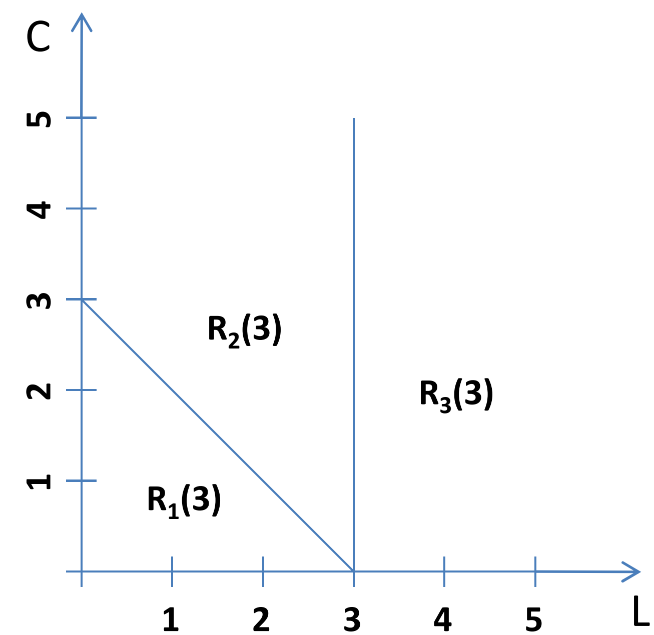
The Insufficient Knowledge Problem (7/7)

- Inevitable failure of an online algorithm is due to:
  - The possible existence of two or more sets of future "conflicting" tasks
    - Scheduler is forced to make an early commitment to meet deadlines of one set of tasks at the expense of all others
- **If** no *a priori* information is available to decide which one of the conflicting sets occur next **then**
  - Optimal scheduling is possible only if the set of tasks does not have conflicting subsets
    - E.g., if C = 1 for all tasks then EDF is optimal run-time algorithm ("swapping" argument)

Sufficient Condition for Conflict Free Task Sets (1/3)

- In the rest of the paper, it is shown that:
  - **if** a feasible schedule exists for a task set when their start-times are same, **then** that task set can be scheduled even when their start-times are different
    - furthermore it is not necessary to know their start- times
- Some Notations:
  - `j´th job: J_j
  - L-C plane is divided into 3 regions

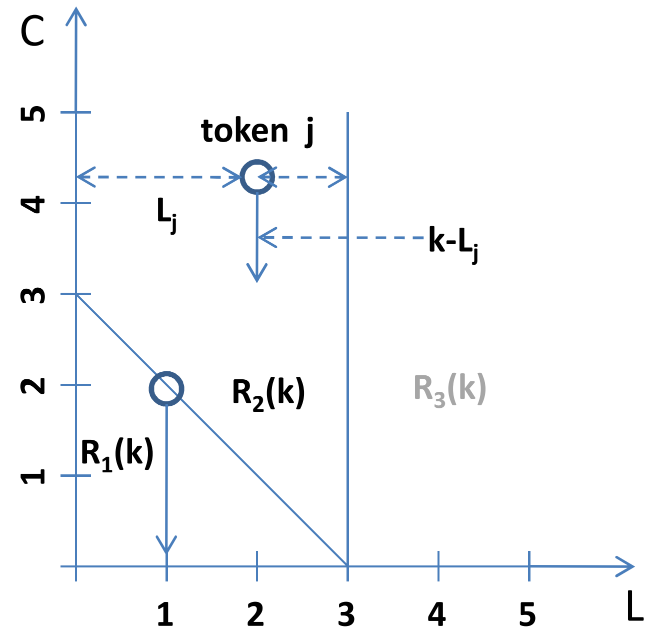
  For all positive integer k:
    - $R_1(k) = \{J_j : D_j <= k\}$
    - $R_2(k) = \{J_j : L_j <= k \text{ and } D_j > k\}$
    - $R_3(k) = \{J_j : L_j > k\}$



Sufficient Condition for Conflict Free Task Sets (2/3)

- "Surplus" computing power in next k time units:

$$F(k) = k \cdot n - \sum_{R_1} C_j - \sum_{R_2} (k - L_j)$$



  - F(k) is a function of time and should be denoted as F(k, i) to signify that F(k) is computed at time=i

- <u>Lemma:</u>
  - A <u>necessary condition</u> for scheduling of a task set whose start-times are the same (at time i=0) is that <u>F(k,0) >= 0</u>

Sufficient Condition for Conflict Free Task Sets (3/3)

- Theorem (Sufficient Condition):
  - **If** a feasible schedule exists for task set whose start-times are same, **then** the same task set can be scheduled at run-time even if their start-times are different and not known *a priori*.
  - Only knowledge of pre-assigned D and C is enough
    - E.g., Least Laxity First
- Periodic Task Sets:
  - LLF is non-optimal at run-time for periodic task sets
  - Theorem (for periodic tasks):
    - Let $T=GCD(D_1, ..., D_m)$ and $t=GCD(T, T*C_1/D_1, ..., T*C_m/D_m)$ and $U <= n$.
    - A sufficient condition for scheduling task set on *n* processors is that *t* be integral

Conclusions

- Contributions of the Paper
  1. It is impossible to design an optimal run-time algorithm for multiprocessor scheduling
     - *A priori* knowledge of all the following parameters is essential :
       1. Deadlines
       2. Computation times
       3. Start-times
  2. If
     - a task set can be successfully scheduled when their start-times are the same (necessary condition: F(k, 0) >=0)

     then

     - they can be scheduled at run-time even if their start-times are different and not known *a priori* (using LLF)
     - Hence, LLF is optimal online algorithm if the above sufficient condition (**if part**) is satisfied.