EE5903 RTS
Chapter 4
Real Time Scheduling Policies

Bharadwaj Veeravalli

elebv@nus.edu.sg

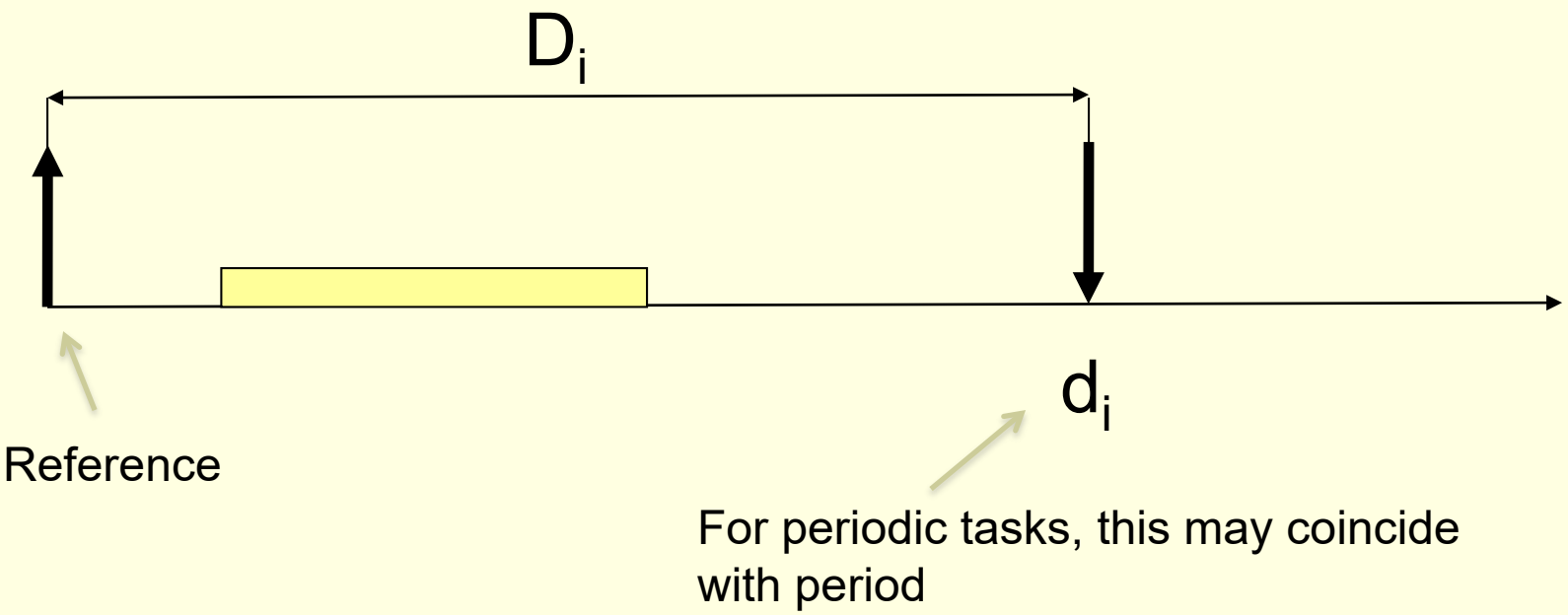
Acknowledgements: Parts of this material from Hard Real-Time Computing Systems, G.C, Buttazzo, Springer, 3rd Edition; RTS, J W Liu;

Outline of this Chapter

- RT Scheduling Algorithms
  -- EDD, EDF, LLF
- Non-Preemptive Scheduling
- Time-Triggered Systems
- Scheduling Anomalies – Priority, # of CPUs, task sizes, resource constraints, etc
- Handling Overloaded Conditions
- Imprecise task scheduling strategies (uniprocessor)
- Handling Periodic Tasks with EDF, RMA and DMA
- Quick note on Event Driven Schedulers
- Precedence Constraints – Static Scheduling algorithm
- Soft-real time issues (if time permits…)

Real-Time Aperiodic/Periodic Scheduling Algorithms

- Tasks can be scheduled by
  - relative deadlines $\mathbf{D_i}$ **(static)**
  - absolute deadlines $\mathbf{d_i}$ (**dynamic**)



RT Scheduling algorithms

- Earliest Due Date (EDD)
- Earliest Deadline First (EDF)
- Least Laxity First (LLF)
- Rate Monotonic Algorithm (RMA)

- Classification of Scheduling Problems – Notational ease!

  α / β / γ [Graham et. Al 1979]

machine infrastructure / task information / metric

Classification Examples

- Example 1:

α / β / γ: 1 / prec / L_{max}

Denotes the problem of scheduling a set of tasks with precedence constraints on a uniprocessor machine in order to minimize the maximum lateness.

If no additional constraints are indicated in the second field, preemption is allowed at any time, and tasks can have arbitrary arrivals.

Classification

- Example 2:

α / β / γ: 3 / no-preem / Σf_i

Denotes the problem of scheduling a set of tasks on a three-processor machine. Preemption is not allowed and the objective is to minimize the sum of the finishing times. Since no other constraints are indicated in the second field, tasks do not have precedence nor resource constraints but have arbitrary arrival times.

Classification

- Example 3:

α / β / γ: 2 / sync / $\Sigma Late_i$

Denotes the problem of scheduling a set of tasks on a two processor machine. Tasks have synchronous arrival times and do not have other constraints.

The objective is to minimize the number of late tasks.

Earliest Due Date (EDD): 1 /synch / L_{max}

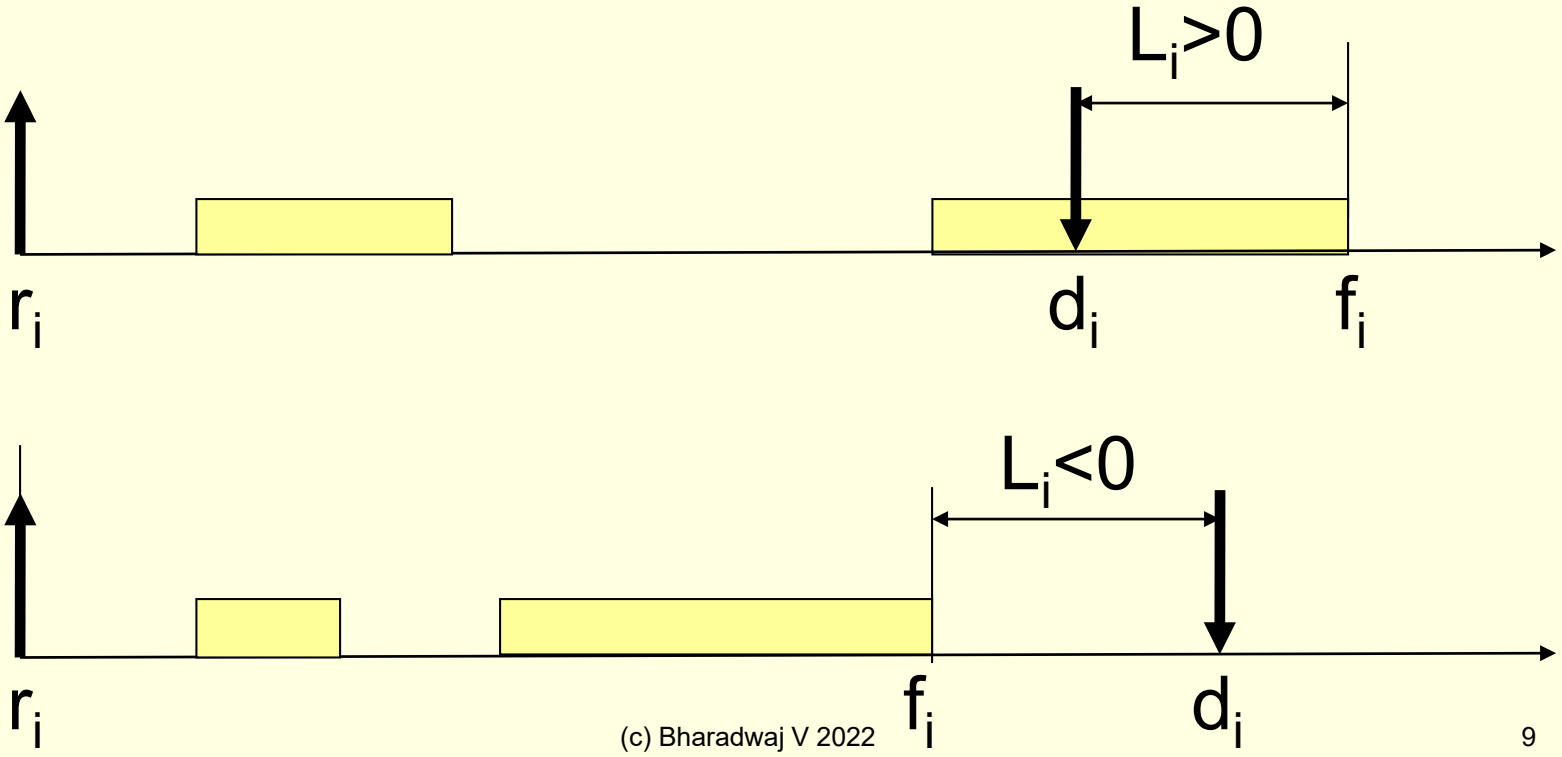
- It selects the task with the earliest **relative** deadline [Jackson's Algorithm].
  - **All tasks arrive simultaneously**
  - **Fixed priority** (D_i is known in advance)
  - **Preemption** is not an issue (*why?*)
  - **It minimizes the maximum lateness (L_{max})**

Theorem -- (Jackson's rule) *Given a set of n independent tasks, any algorithm that executes the tasks in order of nondecreasing deadlines is optimal with respect to minimizing the maximum lateness.*

In this case D_i is also the absolute deadline.

EDD (Cont'd)…

Lateness: $L_i = f_i - d_i$



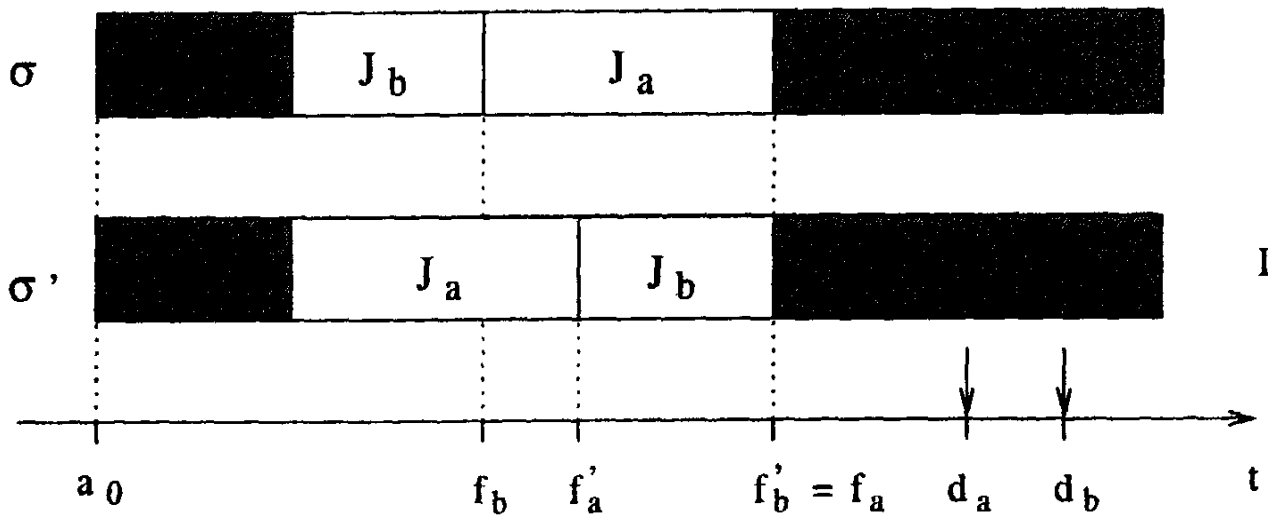
EDD (Cont'd)…

Maximum Lateness: $L_{max}=max_i(L_i)$

if ($L_{max} < 0$) **then**
no task misses its deadline

How do we prove Jackson's rule? We will use a simple interchange logic and applying this logic until optimal schedule is reached.

EDD Optimality – swapping argument



$L_{max}_{ab} = f_a - d_a$

$L'_{max}_{ab} = \max (L'_a, L'_b)$

if $(L'_a \geq L'_b)$ then $L'_{max}_{ab} = f'_a - d_a < f_a - d_a$

if $(L'_a \leq L'_b)$ then $L'_{max}_{ab} = f'_b - d_b < f_a - d_a$

in both cases: $L'_{max}_{ab} < L_{max}_{ab}$

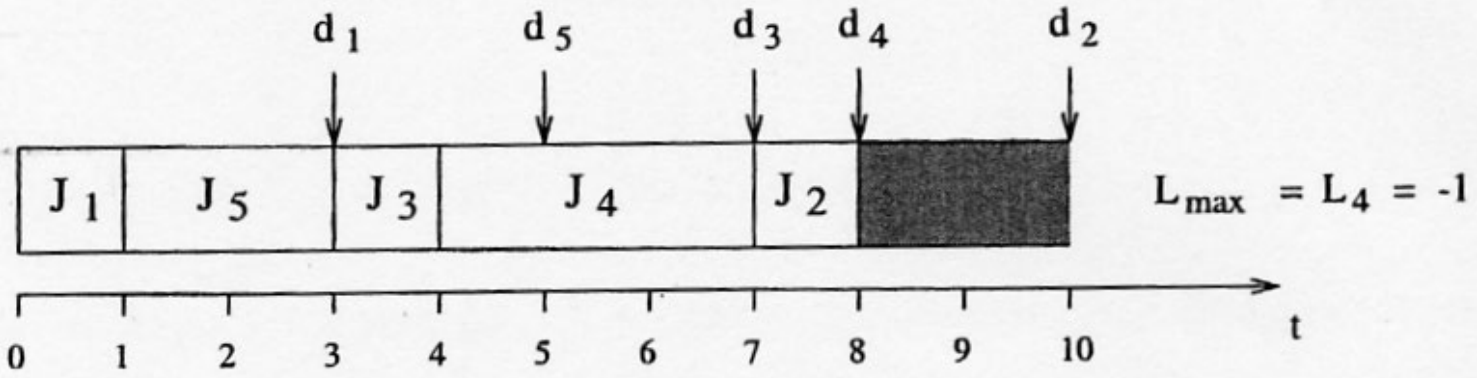
In the first case - (since $f'_a < f_a$)

In the second case: $f_b' = f_a$ and $d_b > d_a$

EDD – Example 1

	J_1	J_2	J_3	J_4	J_5
C_i	1	1	1	3	2
d_i	3	10	7	8	5

Note: Under EDD –
All tasks arrive simultaneously

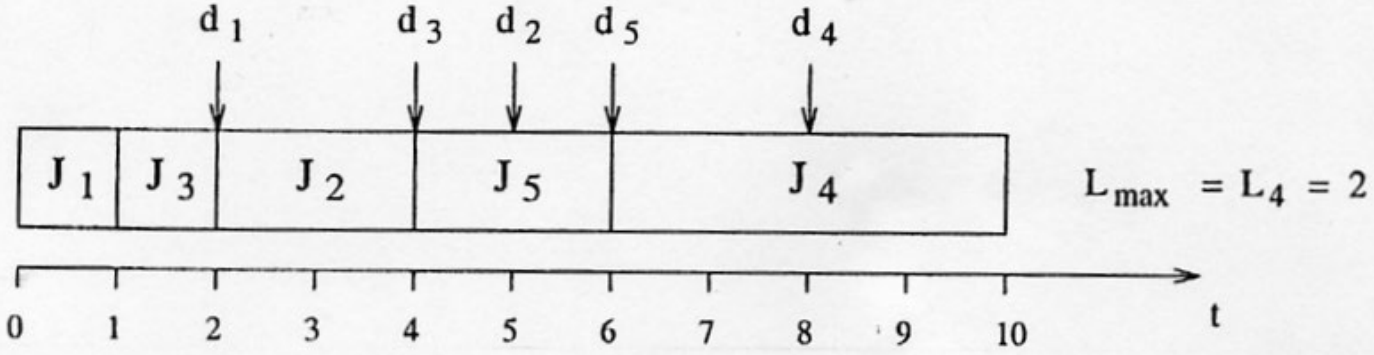


A feasible schedule produced by Jackson's algorithm.

EDD Example 2

This is an example wherein we demonstrate that EDD can result in an infeasible schedule

	J_1	J_2	J_3	J_4	J_5
C_i	1	2	1	4	2
d_i	2	5	4	8	6



An infeasible schedule produced by Jackson's algorithm.

EDD - Guaranteed feasibility

Important Note: The optimality claim of the EDD algorithm cannot guarantee the feasibility of the schedule for any task set.

It only guarantees that **if** a feasible schedule exists for a task set, then EDD will find it. (Our Example 2 demonstrated this!)

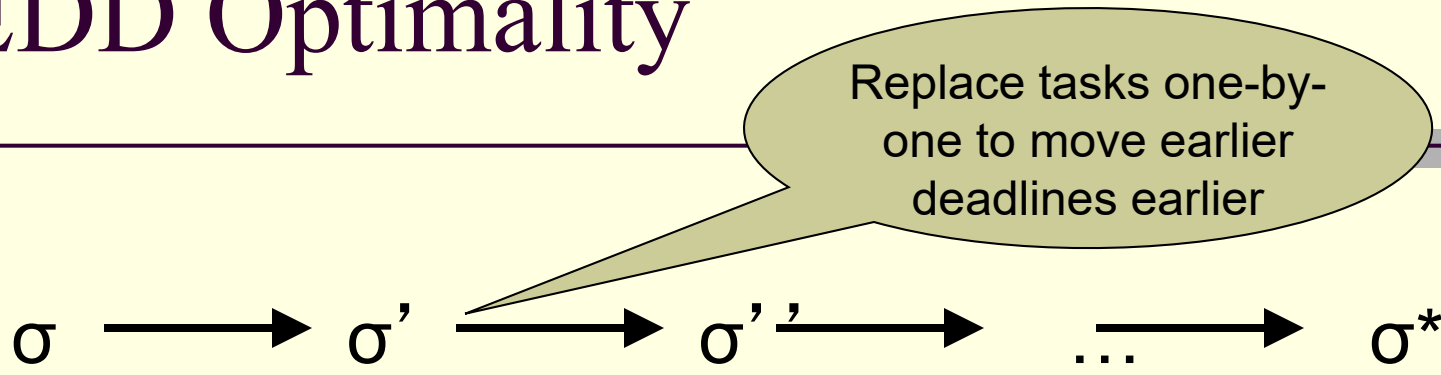
Order tasks by increasing deadlines. Then:

$$\forall i = 1, \ldots, n \quad f_i \leq d_i$$

This implies, $f_i = \sum_{k=1}^{i} C_k$

Which implies, $\forall i = 1, \ldots, n \quad \sum_{k=1}^{i} C_k \leq d_i$

EDD Optimality



$$L_{\max}(\sigma) \geq L_{\max}(\sigma') \geq L_{\max}(\sigma'') \ldots \geq L_{\max}(\sigma^*)$$

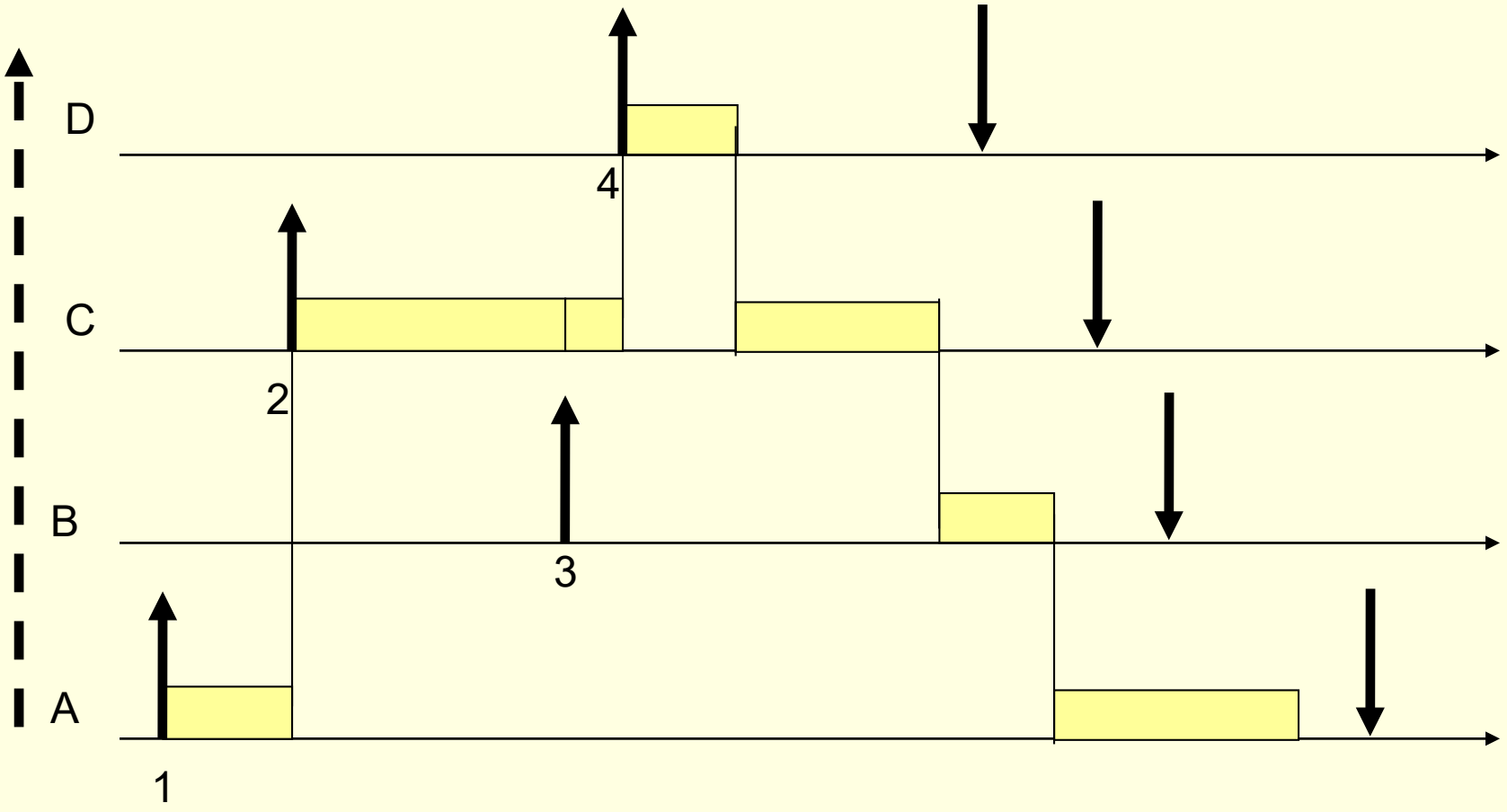
$\sigma^* = \sigma_{EDD}$

$L_{\max}(\sigma_{EDD})$ is the minimum value achievable by any algorithm

Earliest Deadline First (EDF): 1 /preem/ L_{max}

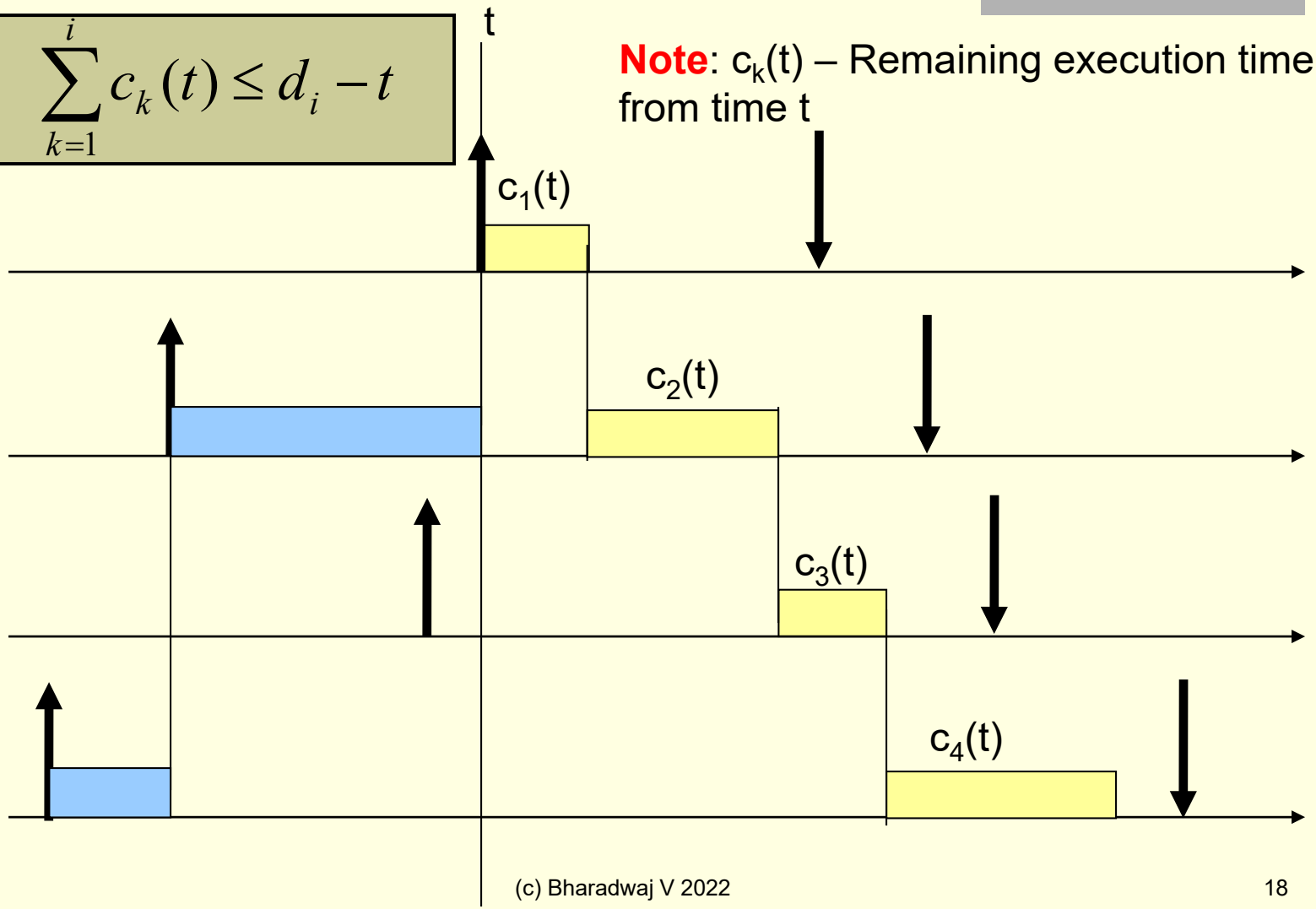
- It selects the task with the earliest **absolute** deadline [*Horn's Algorithm*].
  - Tasks may arrive at any time
  - Dynamic priority (d_i depends on arrival)
  - Fully preemptive tasks
  - It minimizes the maximum lateness (L_{max})
- Newly arrived task is inserted into a queue of ready tasks, sorted by their absolute deadlines. Task at head of queue is executed.
- If a newly arrived task is inserted at the head of the queue, the currently executing task is preempted.

EDF Example

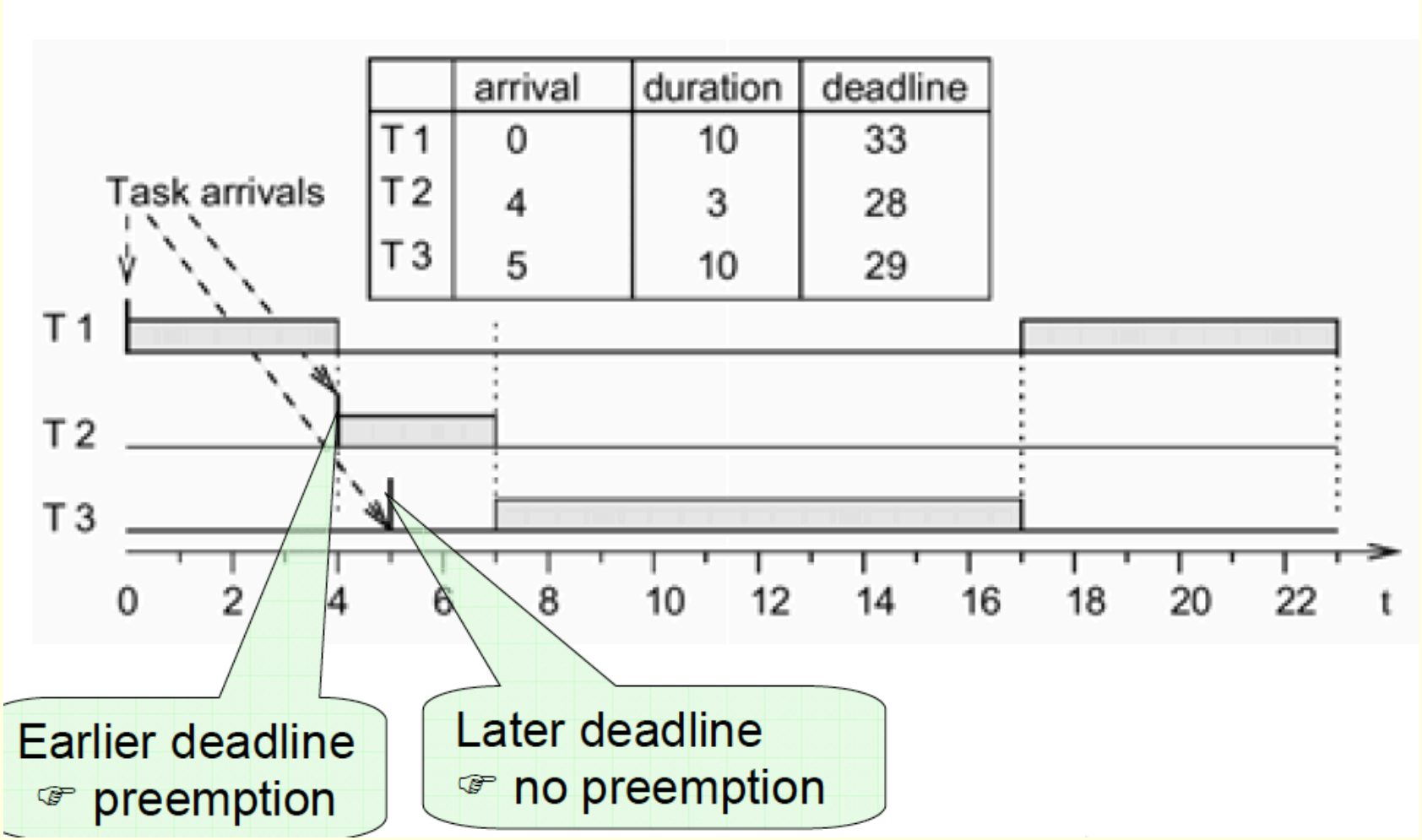


EDF Guarantee test (online)

For all i $\sum_{k=1}^{i} c_k(t) \leq d_i - t$



EDF - Example


arrival
duration
deadline
T 1
0
10
33
T 2
4
3
28
T 3
5
10
29
Task arrivals
T 1
T 2
T 3
0
2
4
6
8
10
12
14
16
18
20
22
t
Earlier deadline
preemption
Later deadline
no preemption

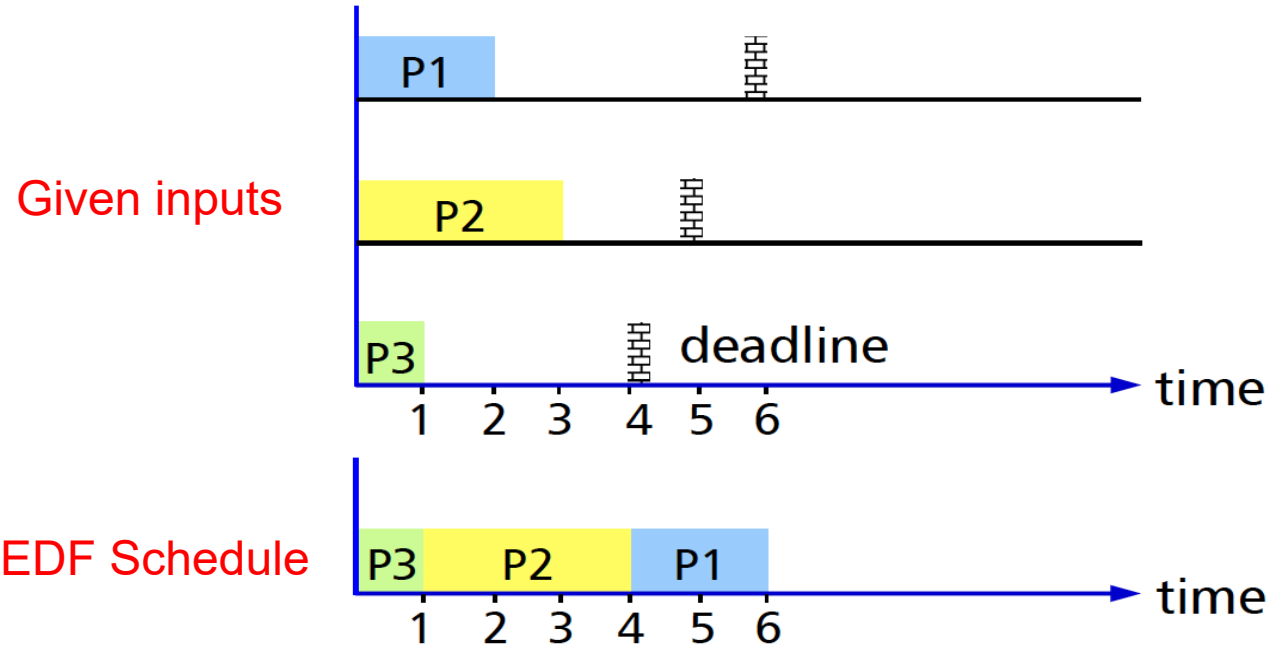
EDF for tasks with Equal Ready Times

- **Static EDF with Equal Ready Times**
  - Sort processes with ascending deadlines [Jackson]
  - Whenever a process is scheduled, check if its deadline is met or violated
  - For the k-th process it must hold that

$$\sum_{i=1}^{k} C_i \leq D_k \qquad k = 1 \cdots n$$

EDF Optimality for Identical Ready Times: Single CPU

- **Example**



- EDF is **optimal** for identical ready times!

Proof of Optimality

- **Proof** (indirect)
  - The processor is always utilized
  - If the schedule has been timely before adding P_k, it is not any more afterwards:
    $$\sum_{i=1}^{k} C_i > D_k$$
  - The only possibility is then to exchange P_k with one of the already planned processes P_j ($j < k$)
  - The sum of execution times remains unchanged
  - We know that $D_j \leq D_k$ and thus
    $$\sum_{i=1}^{k} C_i > D_j$$
  - Therefore the exchange makes things worse
  - Thus: If this algorithm does not find a solution, there is none

EDF Properties

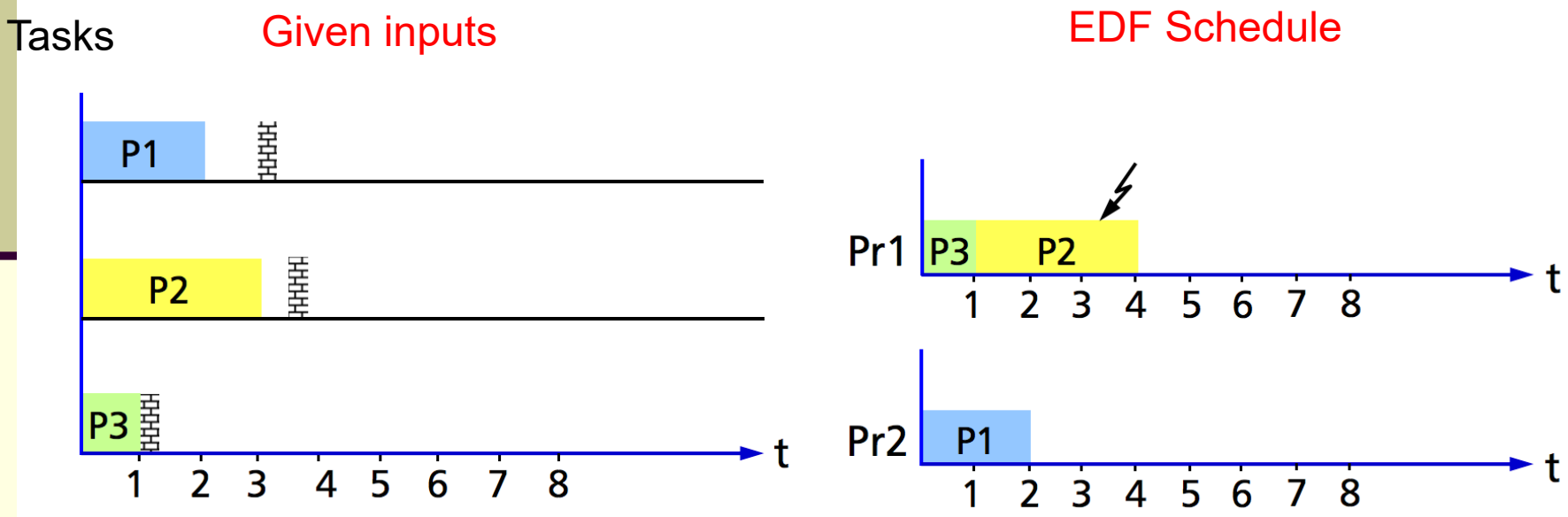
- EDF is optimal for a uniprocessor schedule under pre-emption;

  - If there exists a feasible schedule then EDF will schedule the tasks

- EDF achieves 100% processor utilization – If under EDF a deadline is missed then this means system is overloaded.

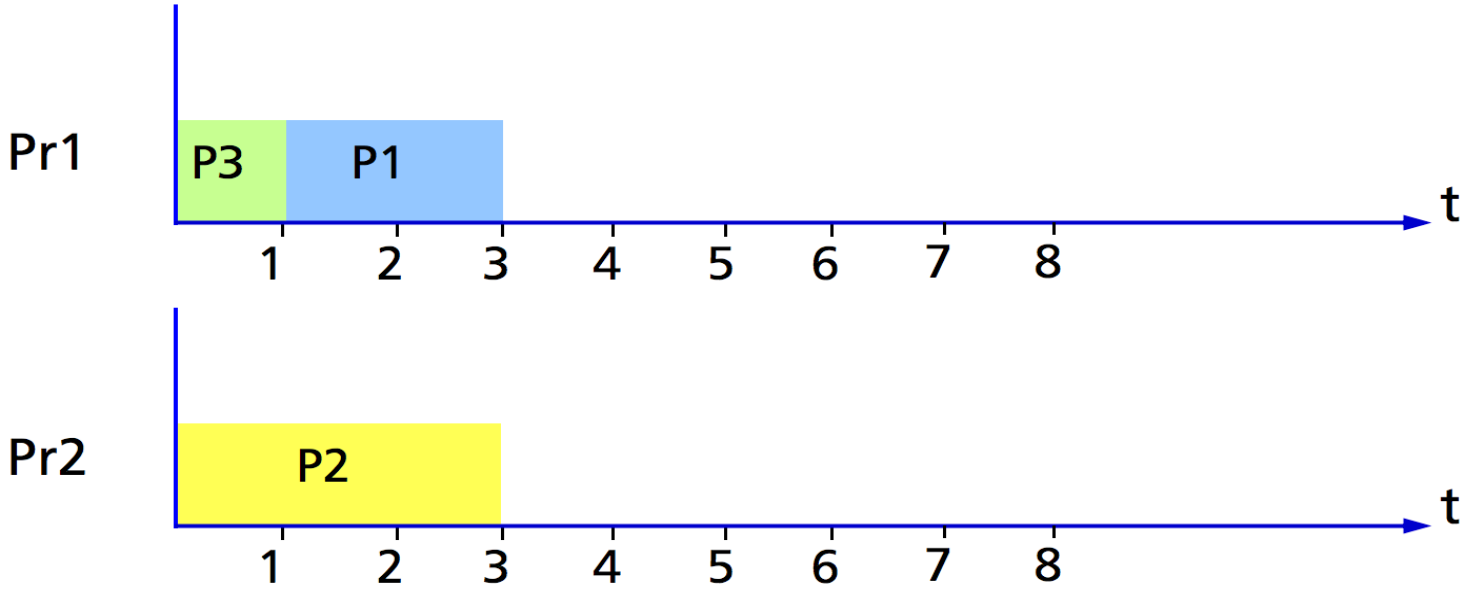
EDF on Multiprocessors - Example

- **What About Multi Processor Systems?**
  - An example shows that EDF is not optimal even for the simple case of equal ready times



EDF on MPS

- There is a timely schedule: ☺
  (not following EDF!)



- Obvious cause of the problem:?

Where exactly is the problem?