

# **Basic Concepts and Aperiodic Task Scheduling**

CPEN 432 Real-Time System Design

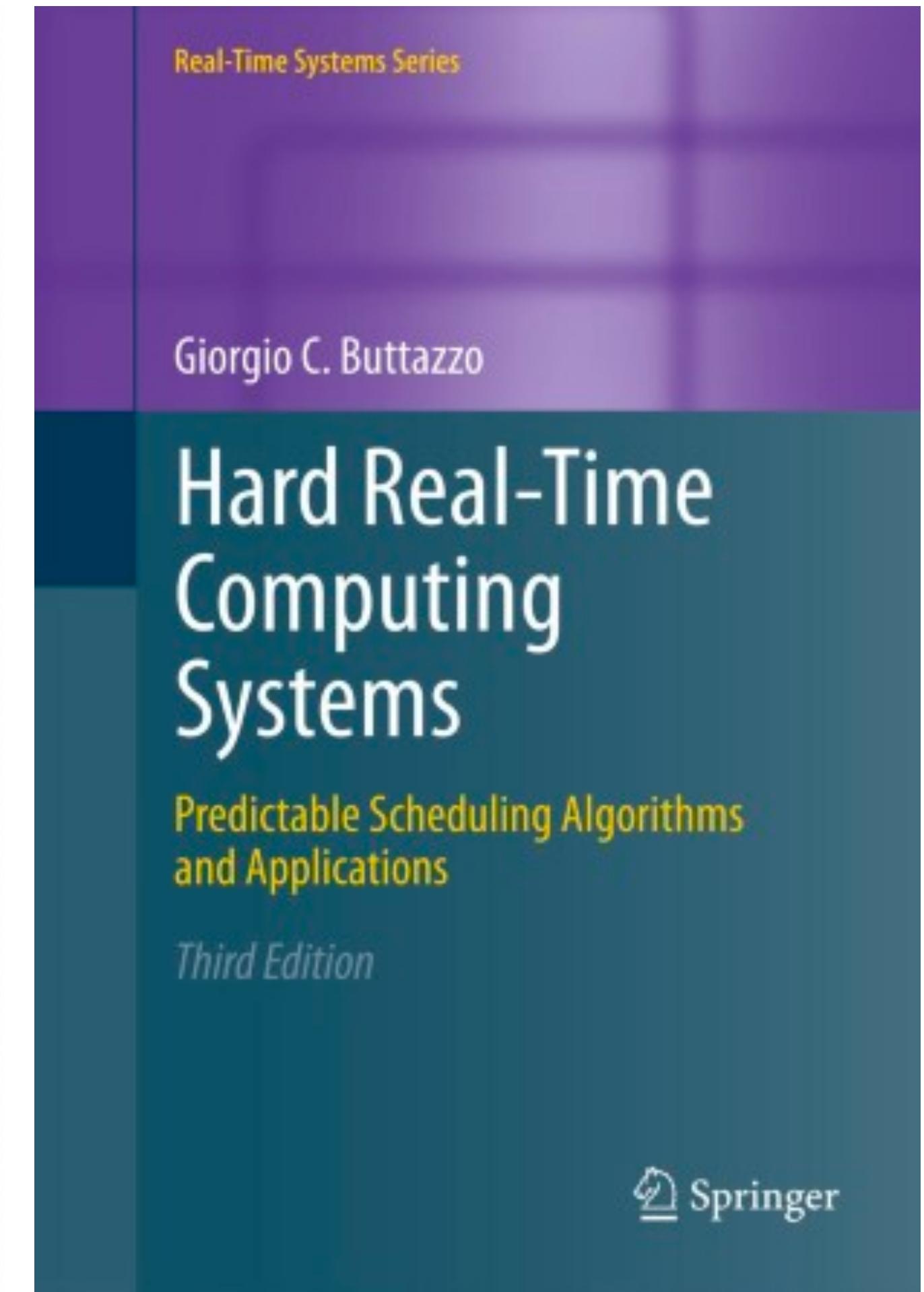
Arpan Gujarati  
University of British Columbia

# What are Real-Time Systems? [1/3]

## 1.1 INTRODUCTION

Real-time systems are computing systems that must react within precise time constraints to events in the environment. As a consequence, the correct behavior of these systems depends not only on the value of the computation but also on the time at which the results are produced [SR88]. A reaction that occurs too late could be useless or even dangerous. Today, real-time computing plays a crucial role in our society, since an increasing number of complex systems rely, in part or completely, on computer control. Examples of applications that require real-time computing include the following:

- Chemical and nuclear plant control,
- control of complex production processes,
- railway switching systems,
- automotive applications,



# What are Real-Time Systems? [2/3]

- The **time** it takes to perform a task is
  - not just an issue of performance
  - but **critical to the correct functioning** of the system
- Examples
  - Airbag deployment in cars, processing of sensor data in drones, etc.
- Challenges
  1. Can we **engineer** the system such that it always satisfies “timing constraints”?
  2. Can we **prove in advance** that the system **will always satisfy** “timing constraints”?

# What are Real-Time Systems? [3/3]

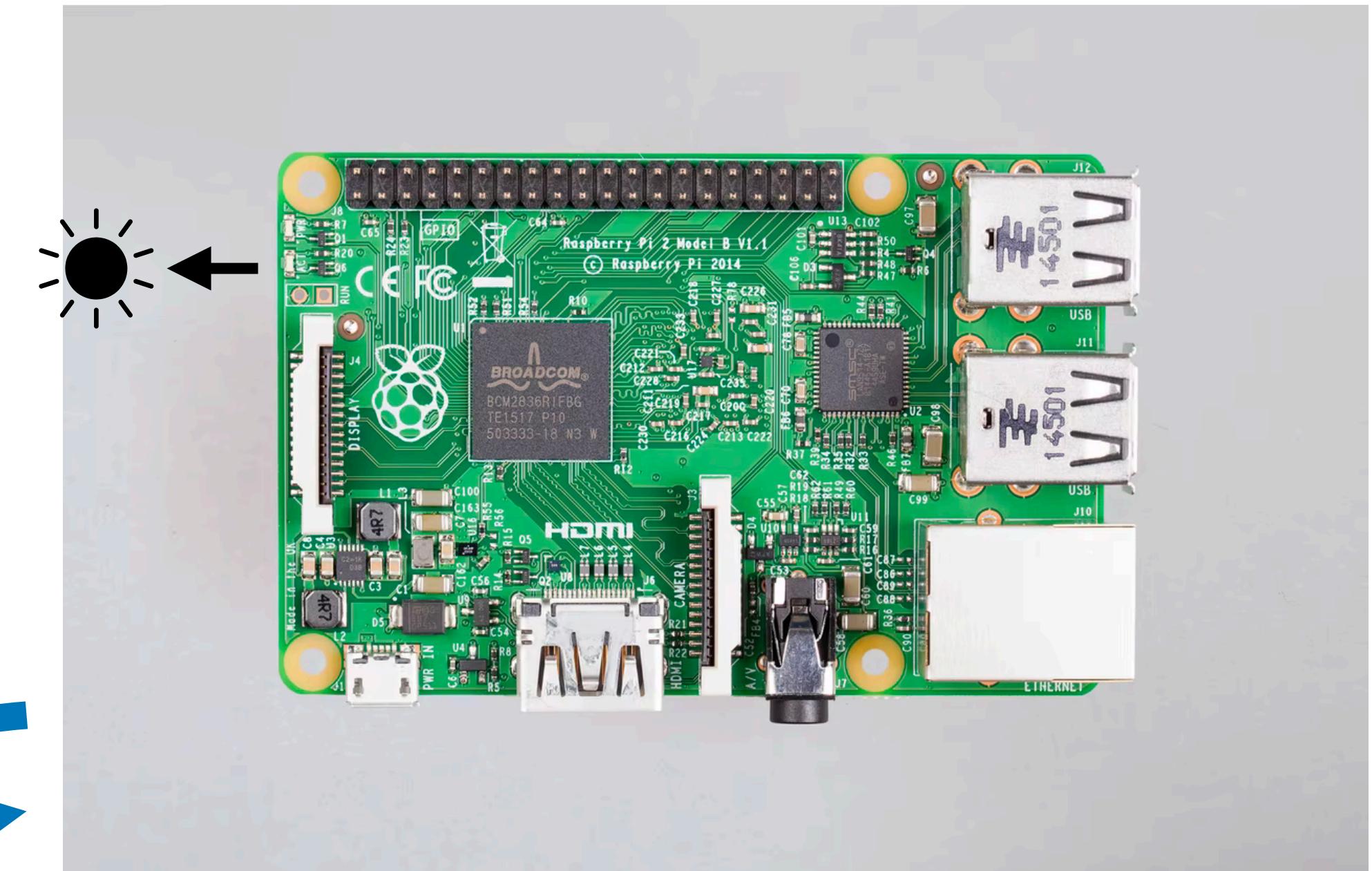
## Model the system and the workload

- ▶ # instructions in the blinking code?
- ▶ Is there an OS? # instructions between calls to the blinking code?
- ▶ Processor speed? Time to execute a single instruction? Caching effects?
- ▶ Ignore unnecessary details ...
  - Can we ignore the GPU?
  - Disable interrupts and ignore?

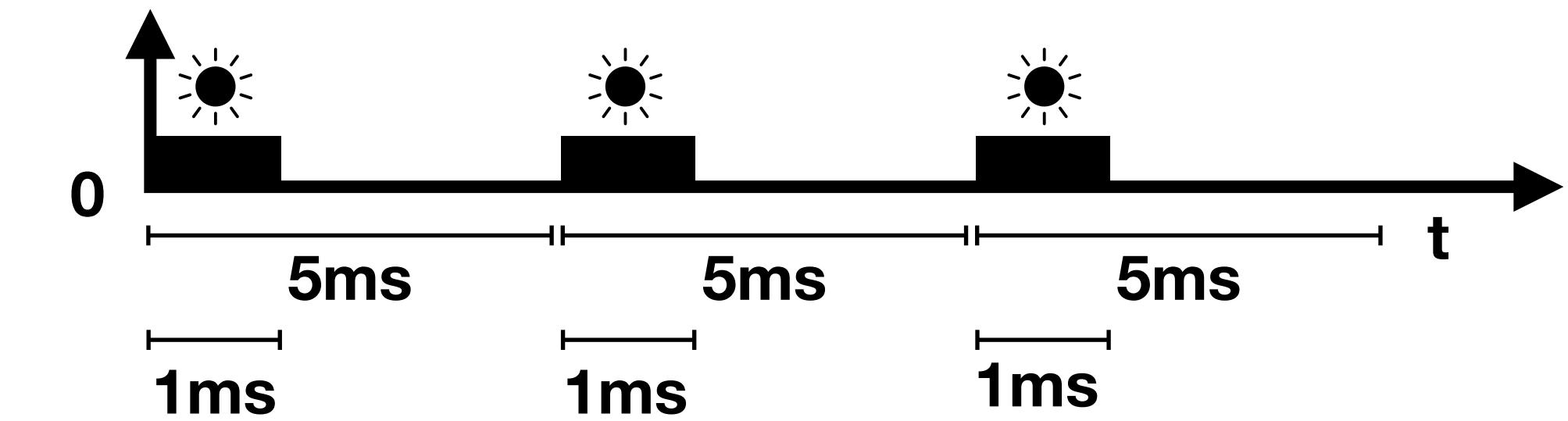
$\approx$

For the given model

- ▶ Prove that the specified timing constraint is always satisfied



Timing constraint: The status LED blinks every 5ms, and continues blinking for precisely 1ms



# **Basic Concepts**

# Operating System

- **Process, Task, Thread**
  - Computation executed by the CPU in a sequential fashion
  - Assumption: **Process = Task = Thread**
- **Scheduling**
  - **Policy** or set of rules that determine how tasks are mapped to processors and the order in which they execute
- **Dispatching**
  - **Mechanism** through which a CPU is allocated to a task (e.g., context switch)
- **Ready task**
  - Task waiting for CPU allocation
- **Running task**
  - Task executing on a CPU
- **Ready queue**
  - Where all ready tasks are kept
- **Preemption**
  - Suspending the running task and inserting into the ready queue

# Schedule as an Integer Step Function

- Given a set of tasks  $\tau = \{\tau_1, \tau_2, \dots, \tau_n\}$  and a **uniprocessor** CPU
  - A **schedule** can be defined using an integer step function  $\sigma(t)$
- Formally,  $\sigma : \mathbb{R}^+ \rightarrow \mathbb{N}$  such that  $\forall t \in \mathbb{R}^+, \exists t_1, t_2$  such that
  - $t \in [t_1, t_2)$  and  $\forall t' \in [t_1, t_2)$  we have  $\sigma(t) = \sigma(t')$
- In other words,
  - $\sigma(t) = k$ , with  $k > 0$ , means that task  $\tau_k$  is executing at time  $t$
  - $\sigma(t) = 0$  means that the CPU is idle

# Schedule as an Integer Step Function

$J_1$  is preempted, and  $J_2$  is scheduled

$\sigma(t) = 2$  now, because  $J_2$  is scheduled

$\sigma(t) = 1$  here because  $J_1$  is scheduled

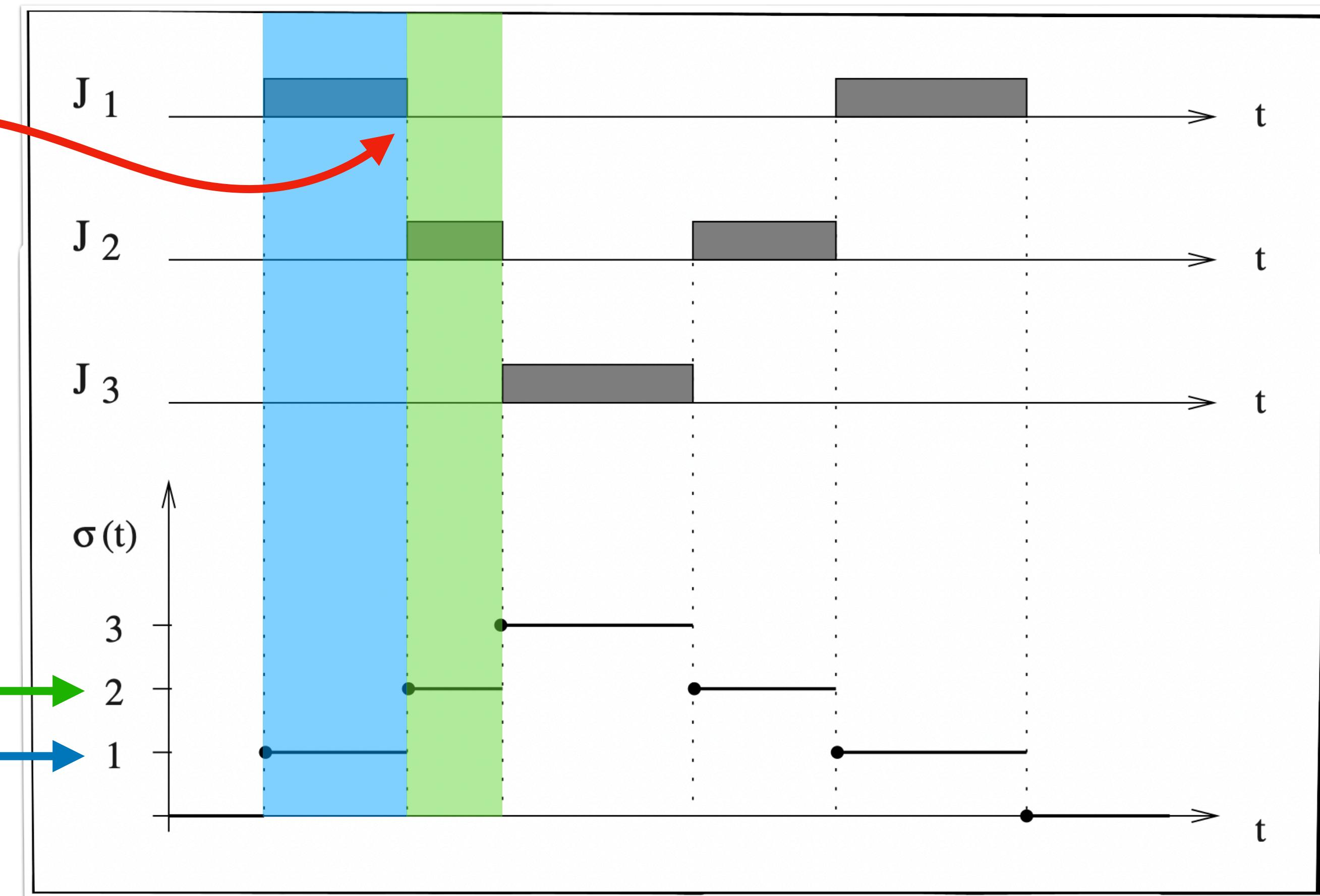


Figure 2.3 from the textbook – Example of a preemptive schedule

# Task Parameters

- **Arrival time**  $a_i$  is the time at which task  $\tau_i$  becomes ready for execution; also denoted as **release time**  $r_i$
- **Computation or execution time**  $C_i$  is the time needed by the processor to execute the task without interruption
- **Absolute deadline**  $d_i$  is the time before which a task should be executed to avoid damage to the system
- **Relative deadline**  $D_i$  is the difference  $d_i - r_i$  between the absolute deadline and the arrival time
- **Start time**  $s_i$  is the time at which the task starts its execution
- **Finishing time**  $f_i$  is the time at which the task finishes its execution
- **Response time**  $R_i$  is the difference  $f_i - r_i$  between the finishing time and the arrival time
- **Laxity or slack**  $X_i$  is the maximum time a task's execution can be delayed so that it can still meet its deadline:  $d_i - r_i - C_i$
- **Lateness**  $L_i$  is the delay  $f_i - d_i$  of a task with respect to its deadline; **Tardiness** is simply  $\max(0, L_i)$

# Activation Frequency

- **Periodic** tasks consist of an infinite sequence of identical iterations
  - These iterations are called instances or **jobs**, and are regularly **activated at a constant rate**
  - Example: Periodic sensing of environment
  - Notation:  $\tau_{i,k}$  denotes the  $k^{th}$  job of a periodic task  $\tau_i$
- **Aperiodic** tasks
  - May consist of an infinite sequence of identical jobs that are activated arbitrarily
  - Or may consist of a finite number of jobs that are activated arbitrarily
  - Or may consist of just a single job
  - Example: Event-triggered tasks, interrupts, etc.
- **This lecture: Scheduling aperiodic tasks with single jobs**
  - Notation: Set of tasks / jobs  $J = \{J_1, J_2, \dots, J_n\}$

# Timing Constraints

- What are the implications of a **deadline miss** for ...
  - ▶ airbag deployment?
  - ▶ an automatic plant watering device?
  - ▶ audio streaming application?
  - ▶ musical notes in a live orchestra

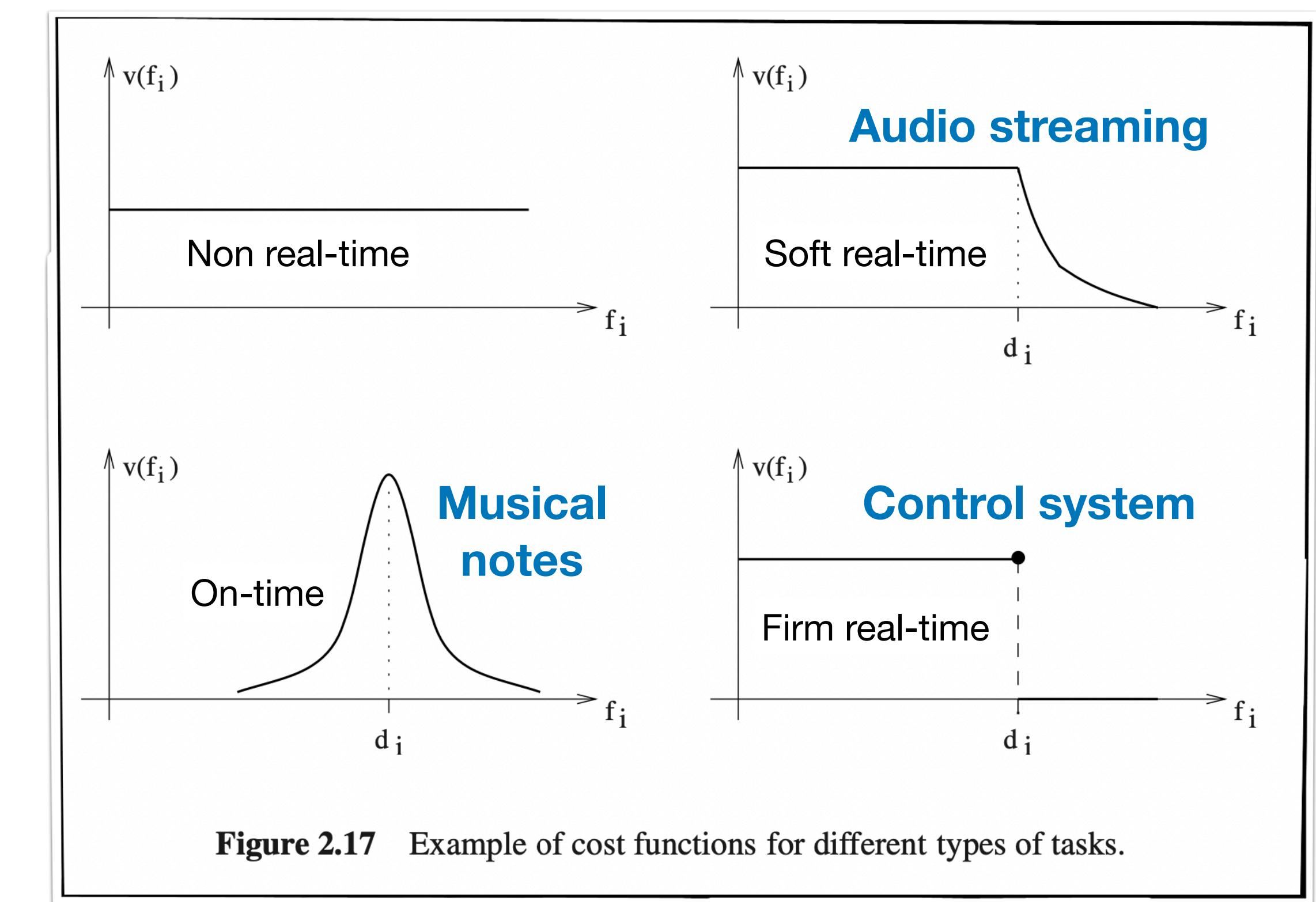
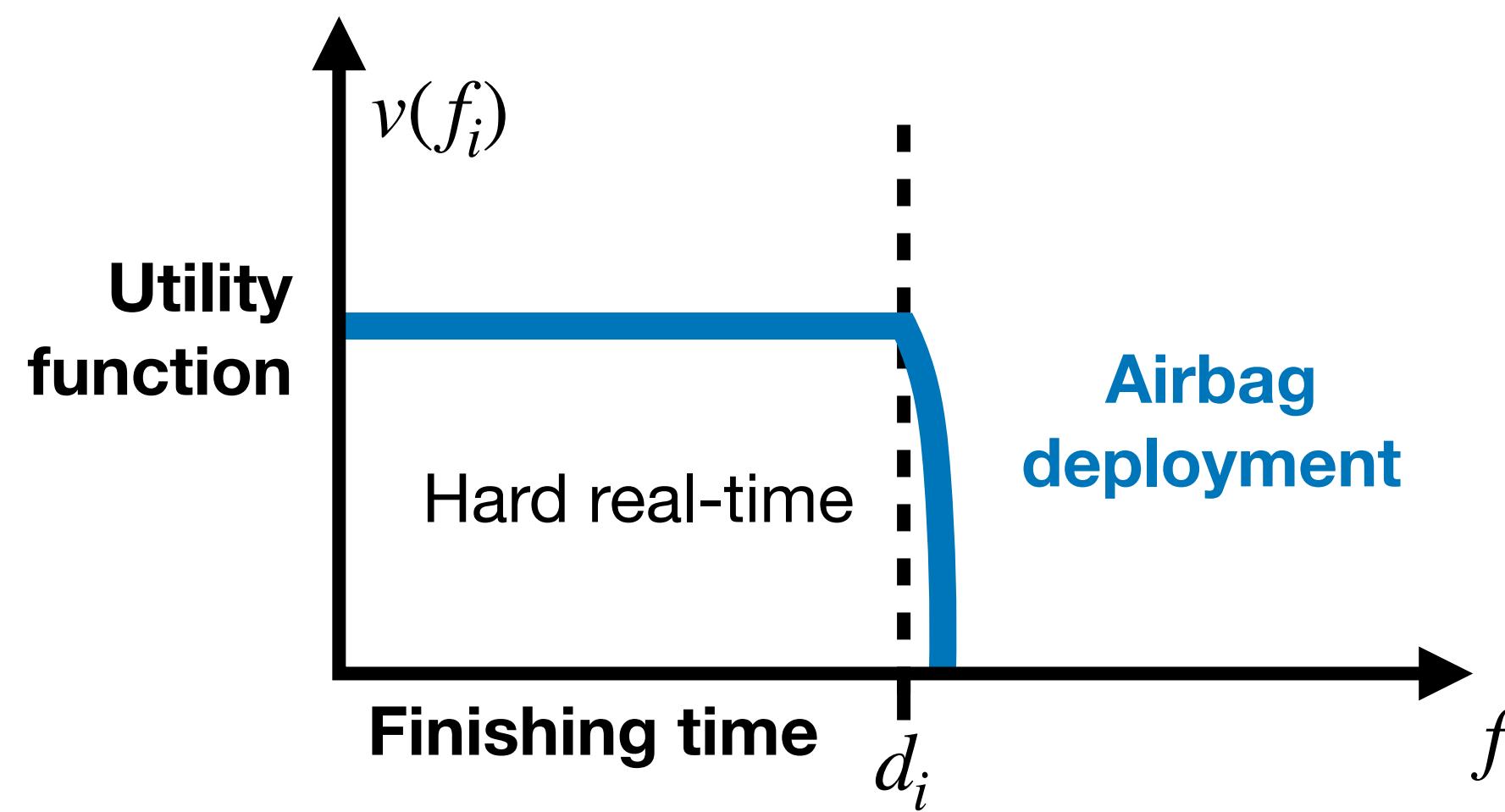


Figure 2.17 Example of cost functions for different types of tasks.

# Performance Metrics

- Question: If the goal is to satisfy timing constraints, why do we care about performance metrics?
  - For comparing different scheduling algorithms
  - More efficient algorithm can support more tasks
  - Embedded platforms have other constraints
    - E.g., minimize power consumption

**Average response time:**

**Generally not useful for real-time systems**

$$\overline{t_r} = \frac{1}{n} \sum_{i=1}^n (f_i - a_i)$$

**Total completion time:**

**Implies good system utilization**

$$t_c = \max_i(f_i) - \min_i(a_i)$$

**Weighted sum of completion times:**

**Useful if tasks have different importance**

$$t_w = \sum_{i=1}^n w_i f_i$$

**Maximum lateness:**

**Useful for soft real-time tasks**

$$L_{max} = \max_i(f_i - d_i)$$

**Maximum number of late tasks:**

**Useful for soft firm-time tasks**

$$N_{late} = \sum_{i=1}^n miss(f_i)$$

where

$$miss(f_i) = \begin{cases} 0 & \text{if } f_i \leq d_i \\ 1 & \text{otherwise} \end{cases}$$

# **Aperiodic Task Scheduling (Job Scheduling)**

# Classification of Scheduling Algorithms

Property	Yes?	No?
Can a task be interrupted at any time?	Preemptive	Non-preemptive
Are scheduling decisions based on fixed parameters, such as fixed task priorities that do not change at runtime?	Static	Dynamic
Is the entire schedule generated in advance and stored in a table?	Offline	Online
Does the scheduling algorithm always minimizes a specified performance metric for a task set?	Optimal	Heuristic
Does the schedule repeat if the task set repeats?	Deterministic	Random

# Scenario #1

- **Given**
  - Set of  $n$  aperiodic jobs  $J = \{J_1, J_2, \dots, J_n\}$
  - Synchronous arrival times  $\forall i : a_i = 0$
  - Uniprocessor system
- **Objective**
  - Minimize the maximum lateness  $L_{max} = \max_i (f_i - d_i)$
- **Constraints**
  - No job must misses its deadline
- **Scheduling policy?**
  - **Jackson's algorithm:** Execute the tasks in order of non-decreasing deadlines

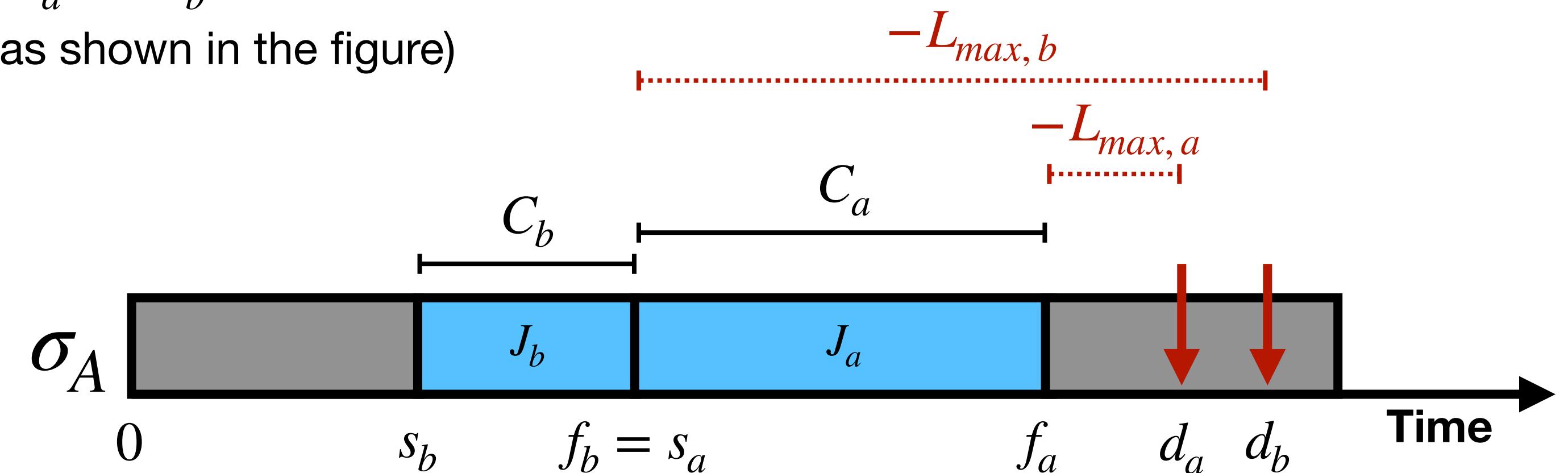
# Scenario #1, Example #1

	J <sub>1</sub>	J <sub>2</sub>	J <sub>3</sub>	J <sub>4</sub>	J <sub>5</sub>
C <sub>i</sub>	1	1	1	3	2
d <sub>i</sub>	3	10	7	8	5

- Classroom assignment
  - ▶ Can you draw the schedule?
  - ▶ Is the schedule feasible? Why?
  - ▶ What is  $L_{max}$ ?
  - ▶ **Is this the minimum possible  $L_{max}$ ?**

# Jackson's Algorithm is Optimal w.r.t. minimizing $L_{max}$ [1/3]

- (1) For simplicity, assume that each deadline is unique:  $\forall i, k, d_i \neq d_k$
- (2) Let Jackson's Algorithm produce schedule  $\sigma_{Jackson}$  with maximum lateness  $L_{max}(\sigma_{Jackson})$
- (3) Suppose there exists another algorithm A that produces schedule  $\sigma_A \neq \sigma_{Jackson}$  with maximum lateness  $L_{max}(\sigma_A) < L_{max}(\sigma_{Jackson})$ 
  - (I) Since  $\sigma_A \neq \sigma_{Jackson}$ , there exists at least two jobs  $J_a$  and  $J_b$  such that:
    - (i)  $J_b$  immediately precedes  $J_a$  in schedule  $\sigma_A$  (as shown in the figure)
    - (ii) and  $d_a < d_b$



# Jackson's Algorithm is Optimal w.r.t. minimizing $L_{max}$ [2/3]

(4) We transpose ( $\rightsquigarrow$ ) schedule  $\sigma_A$  to  $\sigma'_A$  by interchanging the time slots of  $J_a$  and  $J_b$

(5) From the figure,  $L_{max,a}$  is the maximum among all four lateness values illustrated, since

- (I)  $-L_{max,a} < -L'_{max,a}$
- (II)  $-L_{max,a} < -L'_{max,b}$
- (III)  $-L_{max,a} < -L_{max,b}$

(6) First case: The maximum lateness in  $\sigma_A$

corresponded to  $J_a$ , i.e.,  $L_{max}(\sigma_A) = L_{max,a}$

- (I) From (5) above, since the maximum lateness of  $J_a$  and  $J_b$  in  $\sigma'_A$  can only reduce w.r.t.  $L_{max,a}$ ,

(II) Thus,  $L_{max}(\sigma_A) \geq L_{max}(\sigma'_A)$  in the first case

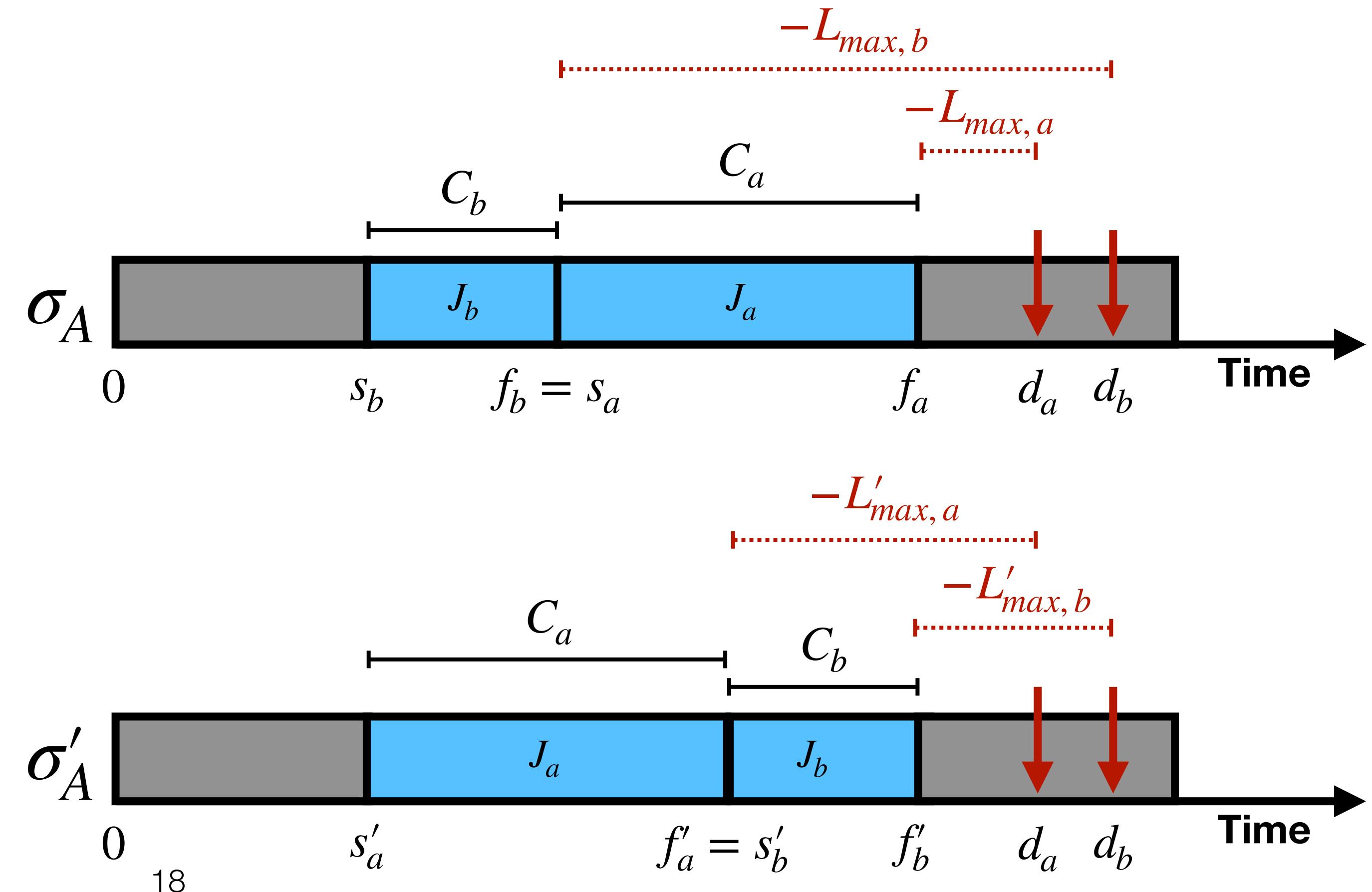
(7) Second case: The maximum lateness in  $\sigma_A$

corresponded to some  $J_c$  (grey region in the figure),

i.e.,  $L_{max}(\sigma_A) = L_{max,c} > L_{max,a}$

- (I) The schedule of all other jobs and their maximum lateness remains intact in  $\sigma'_A$

(II) Thus,  $L_{max}(\sigma_A) = L_{max}(\sigma'_A)$  in the second case



# Jackson's Algorithm is Optimal w.r.t. minimizing $L_{max}$ [3/3]

- (8) From (6) and (7), transposition  $\sigma_A \rightsquigarrow \sigma'_A$  cannot result in a higher  $L_{max}$
- (9) We can perform a finite number of such transpositions, such that  $\sigma_A \rightsquigarrow \sigma'_A \rightsquigarrow \sigma''_A \rightsquigarrow \dots \rightsquigarrow \sigma_{Jackson}$
- (10) From (4)-(8),  $L_{max}(\sigma_A) \geq L_{max}(\sigma'_A) \geq L_{max}(\sigma''_A) \geq \dots \geq L_{max}(\sigma_{Jackson})$
- (11) This contradicts (3). Hence, Jackson's algorithm is indeed optimal and results in the minimum possible

# Scenario #1, Example #2

- Classroom assignment
  - Can you draw the schedule?
  - Is the schedule feasible? Why?
  - **How can we guarantee feasibility?**

	J <sub>1</sub>	J <sub>2</sub>	J <sub>3</sub>	J <sub>4</sub>	J <sub>5</sub>
C <sub>i</sub>	1	2	1	4	2
d <sub>i</sub>	2	5	4	8	6

# Scenario #1

## Scheduling policy

- Jackson's algorithms is optimal with respect to  $L_{max}$  for all job sets, but it does not guarantee feasibility for all job sets!
- However, Jackson's algorithm guarantees feasibility if the job set  $J$  satisfies the following conditions.

$$\forall i = 1, \dots, n \quad \sum_{k=1}^i C_k \leq d_i .$$

## Schedulability analysis

Design time claim:

**"job set  $J$  is (not) schedulable using Jackson's algorithm."**

Yes?  
No?

# Scenario #2

- **Given**
  - Set of  $n$  aperiodic jobs  $J = \{J_1, J_2, \dots, J_n\}$
  - **Arbitrary** arrival times  $\forall i, k : a_i \neq a_k$
  - Uniprocessor system with **preemption**
- **Objective**
  - Minimize the maximum lateness  $L_{max} = \max_i (f_i - d_i)$
- **Constraints**
  - No job must miss its deadline
- **Scheduling policy?**
  - **Earliest Deadline First (EDF):** At any instant, execute a ready task with the earliest absolute deadline

# Scenario #2, Example #1

	J <sub>1</sub>	J <sub>2</sub>	J <sub>3</sub>	J <sub>4</sub>	J <sub>5</sub>
a <sub>i</sub>	0	0	2	3	6
C <sub>i</sub>	1	2	2	2	2
d <sub>i</sub>	2	5	4	10	9

- Classroom assignment
  - ▶ Can you draw the schedule?
  - ▶ Is the schedule feasible? Why?
  - ▶ What is  $L_{max}$ ?
  - ▶ **Is this the minimum possible  $L_{max}$ ?**

# Scenario #3

- **Given**
  - Set of  $n$  aperiodic jobs  $J = \{J_1, J_2, \dots, J_n\}$
  - Arbitrary arrival times  $\forall i, k : a_i \neq a_k$
  - Uniprocessor system with **NO** preemption
- **Objective**
  - Minimize the maximum lateness  $L_{max} = \max_i (f_i - d_i)$
- **Constraints**
  - No job must misses its deadline
- **Scheduling policy?**
  - **Earliest Deadline First (EDF)? Bratley's algorithm from [Buttazzo 3.4.1]?**