Basic Concepts and Aperiodic Task Scheduling

CPEN 432 Real-Time System Design

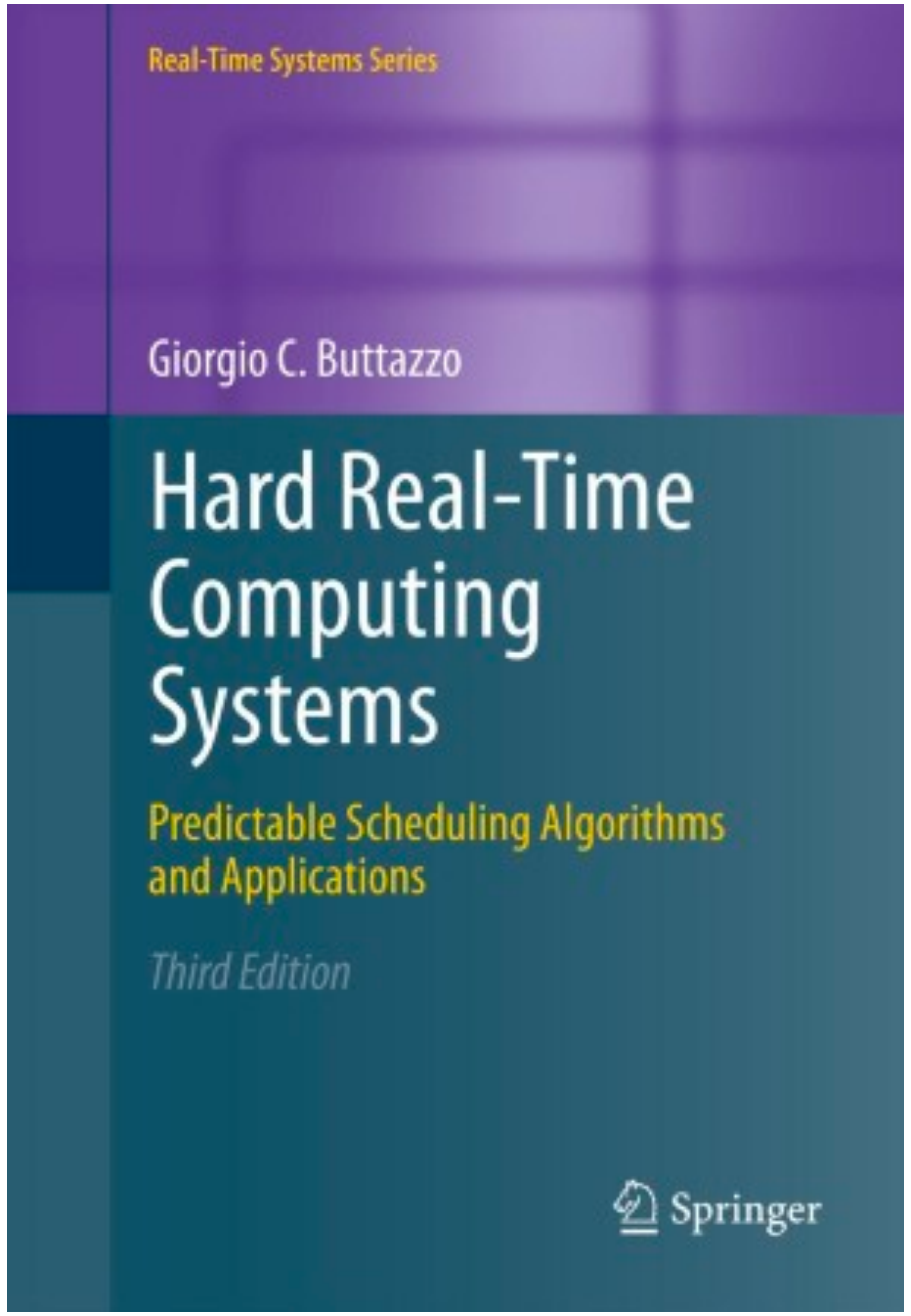
Arpan Gujarati
University of British Columbia

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.

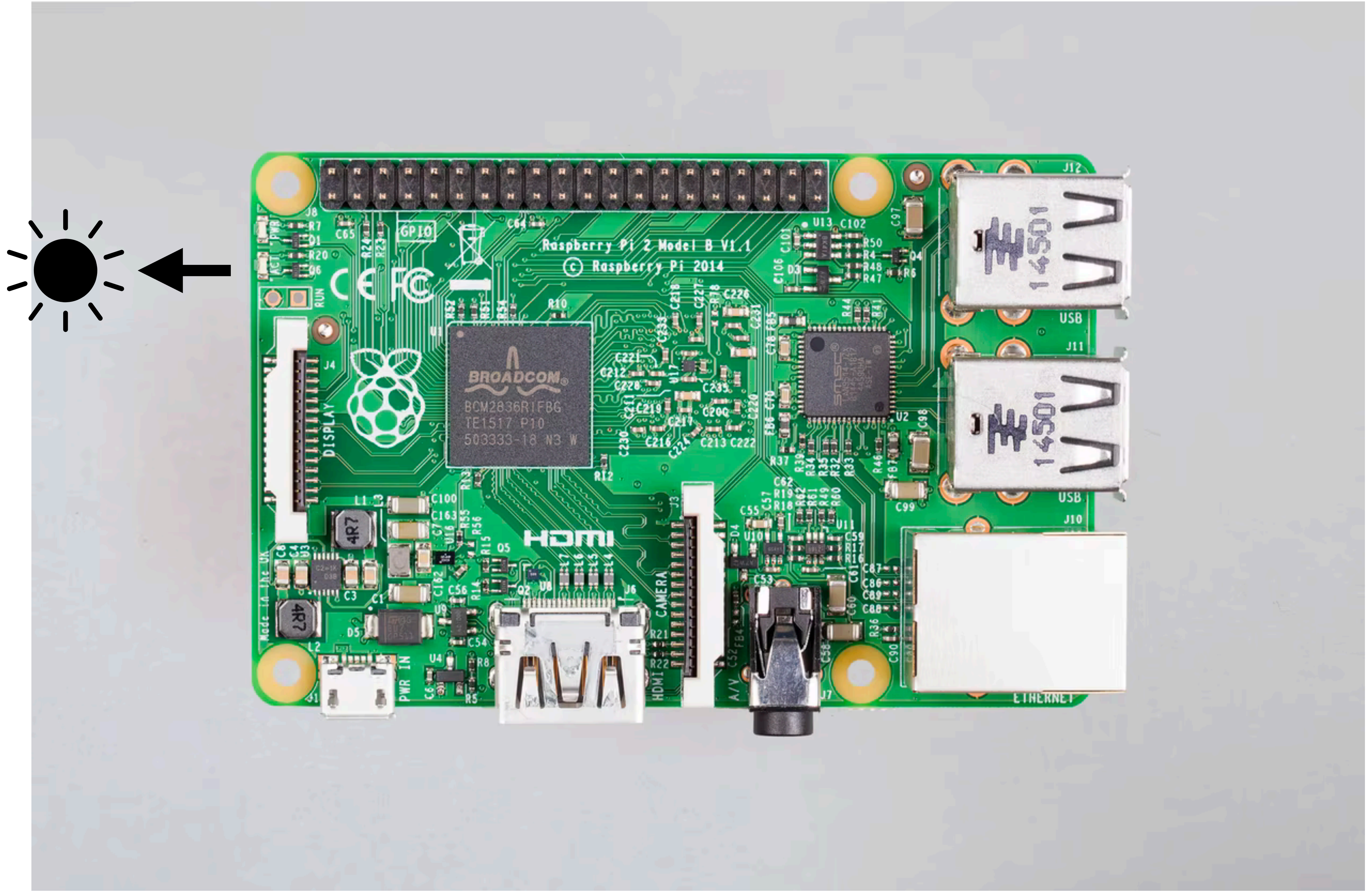
- Challenges
  1. Can we **engineer** the system such that it always satisfies “timing constraints”?
  2. Can we **prove in advance** that the system **will always satisfy** “timing constraints”?

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?

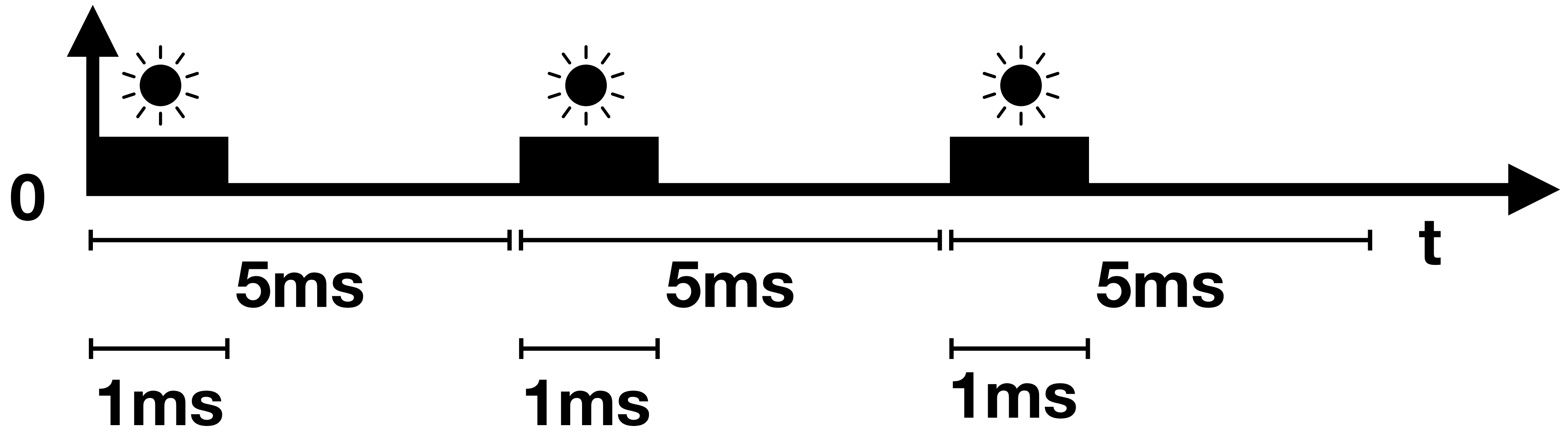
$\approx$



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, $\sigma : \mathbb{R}^+ \to \mathbb{N}$ 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 τ_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 is J_2 scheduled

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

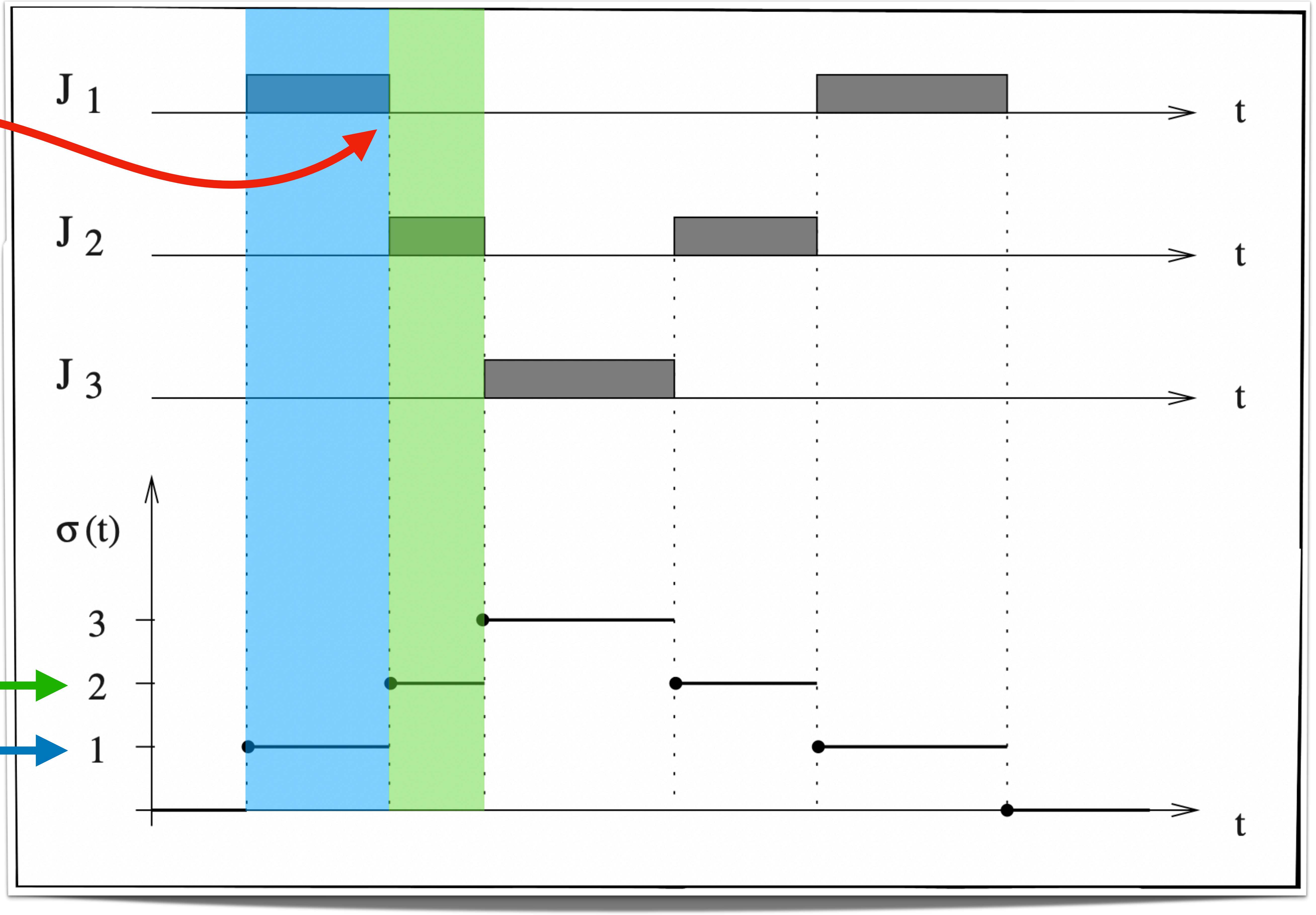


Figure 2.3 from the textbook — Example of a preemptive schedule

Task Parameters

- **Arrival time** a_i is the time at which task τ_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

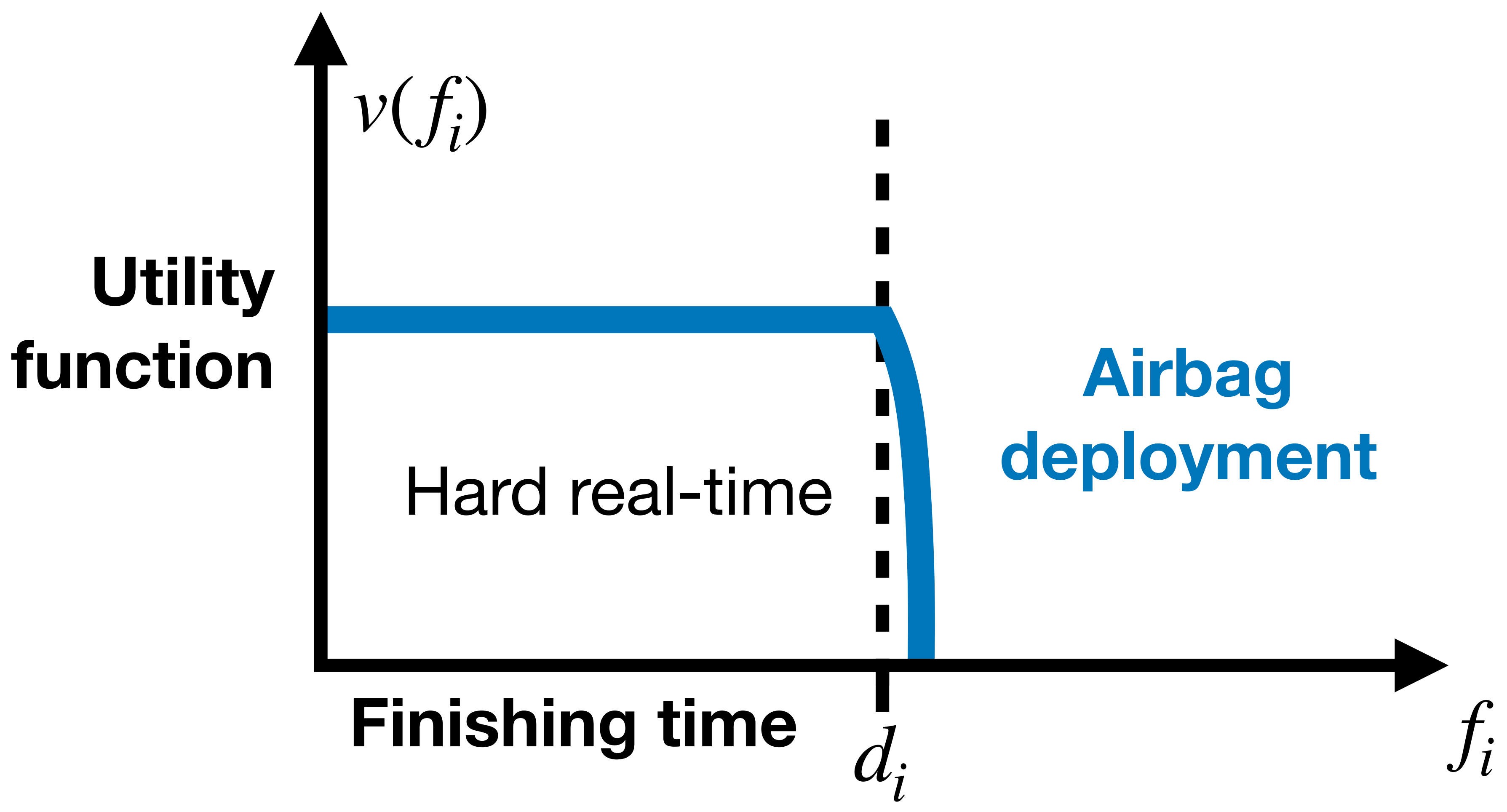
- **Periodic** tasks consist of an infinite sequence of identical iterations
  - These iterations are called instances or **jobs,** and are regularly **activated at a constant rate**
  - Example: Periodic sensing of environment
  - Notation: $\tau_{i,k}$ denotes the k^{th} job of a periodic task τ_i

- **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 $J = \{J_1, J_2, \ldots, J_n\}$

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



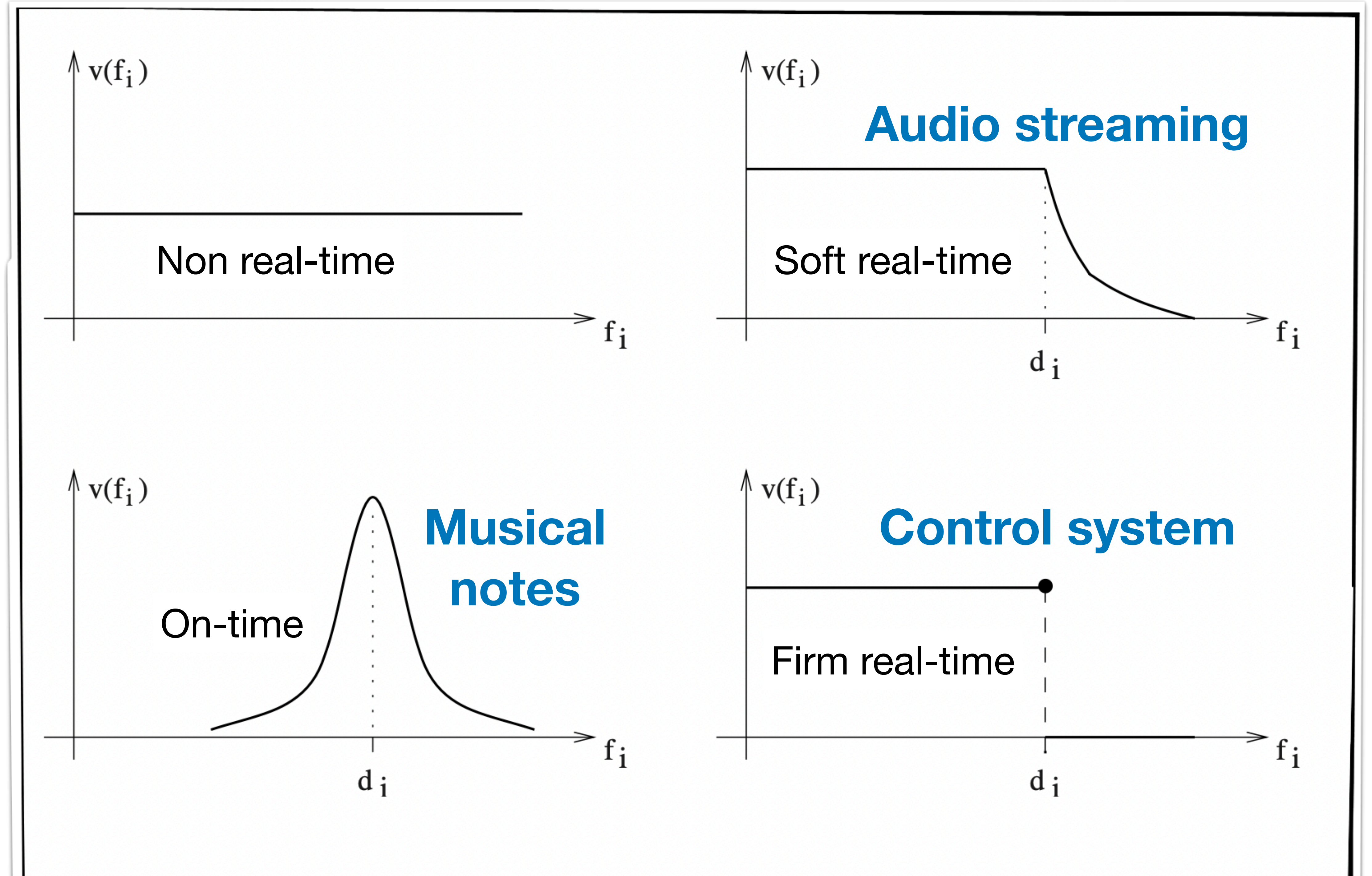


Figure 2.17 Example of cost functions for different types of tasks.

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 $J = \{J_1, J_2, \ldots, J_n\}$
  - 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_1	J_2	J_3	J_4	J_5
C_i	1	1	1	3	2
d_i	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}?**

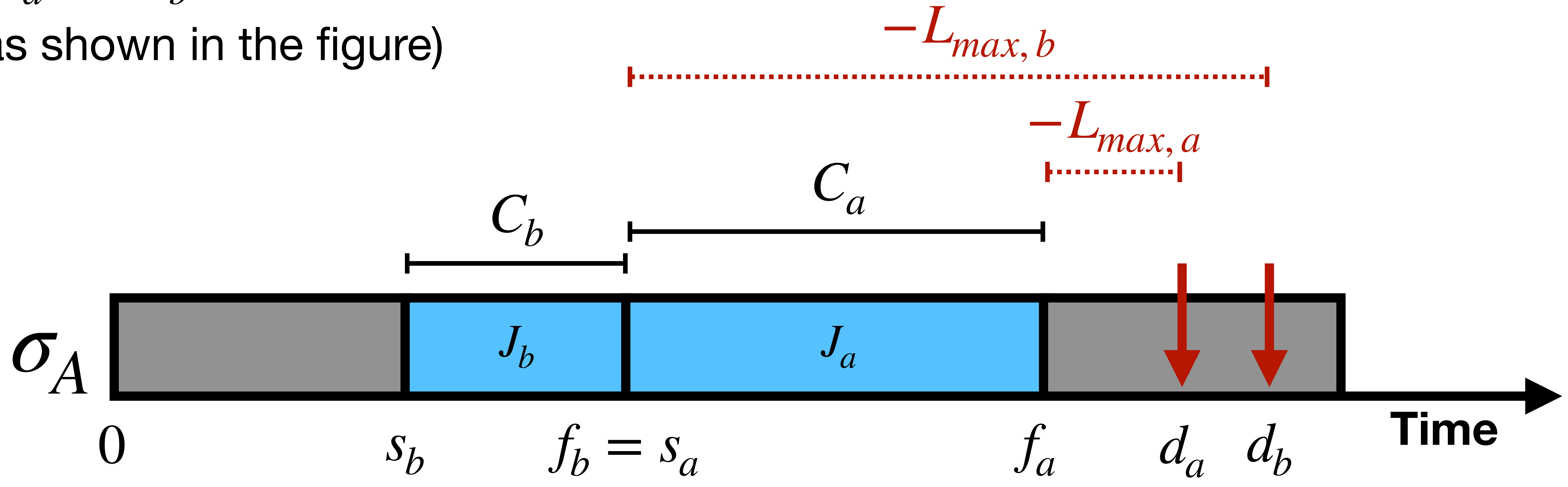
Jackson's Algorithm is Optimal w.r.t. minimizing L_{max} [1/3]

(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 σ_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 transposition ($\rightsquigarrow$) schedule σ_A to σ'_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

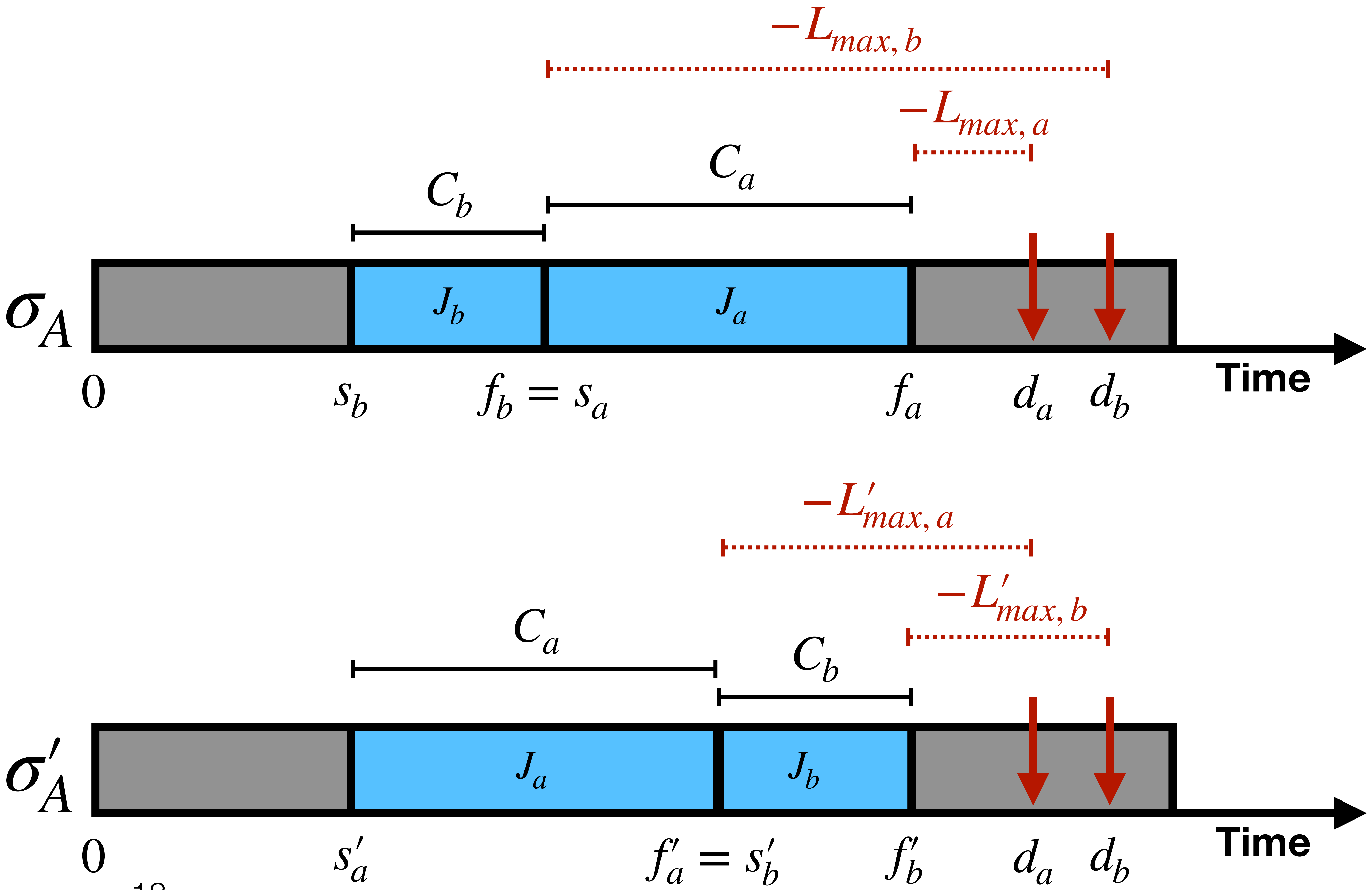
- (I) $-L_{max,a} < -L'_{max,a}$
- (II) $-L_{max,a} < -L'_{max,b}$
- (III) $-L_{max,a} < -L_{max,b}$

(6) First case: The maximum lateness in σ_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 σ'_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 σ_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 σ'_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 $\sigma_A \rightsquigarrow \sigma'_A \rightsquigarrow \sigma''_A \rightsquigarrow \ldots \rightsquigarrow \sigma_{Jackson}$

(10) From (4)-(8), $L_{max}(\sigma_A) \geq L_{max}(\sigma'_A) \geq L_{max}(\sigma''_A) \geq \ldots \geq L_{max}(\sigma_{Jackson})$

(11) This contradicts (3). Hence, Jackson's algorithm is indeed optimal and results in the minimum possible

Basic Concepts and Aperiodic Task Scheduling (contd.)

CPEN 432 Real-Time System Design

Arpan Gujarati
University of British Columbia

Recap: Task Parameters

Arrival or release time:

- Job J_i — *"I am ready for execution!"*

Absolute deadline:

- Job J_i — *"I better be done by now!"*

Start time:

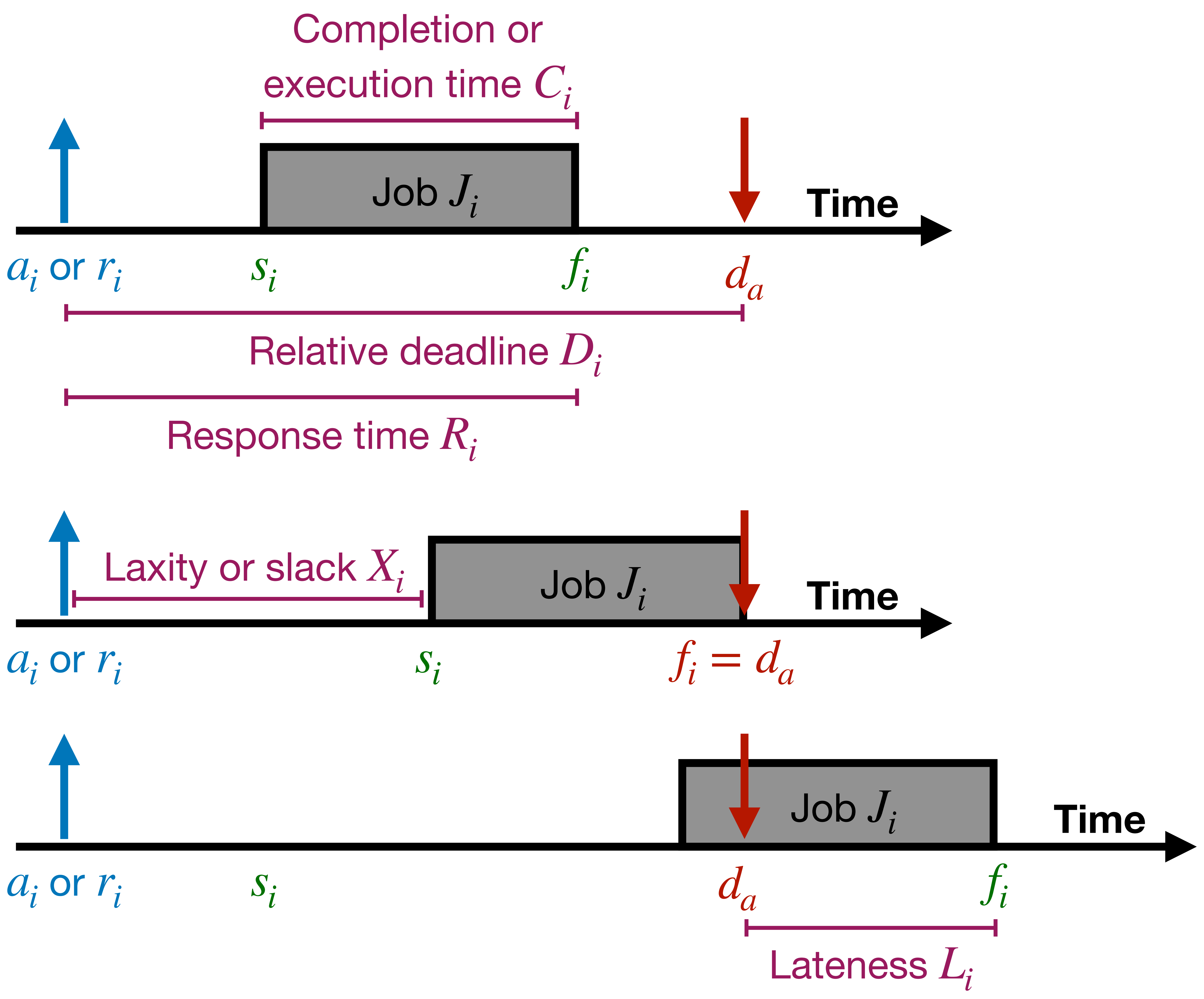
- Operating system — *"Start working!"*

Finishing time:

- Job J_i — *"I am done!"*

Relative time durations

- $C_i = f_i - s_i$
- $D_i = d_i - a_i$
- $R_i = f_i - a_i$
- $X_i = D_i - C_i$
- $L_i = f_i - d_i$



Recap: Activation Frequencies

- **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 τ_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.

Recap: Scenario #1

- **Given**
  - Set of n aperiodic jobs $J = \{J_1, J_2, \ \ldots, J_n\}$
  - 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

Jackson's Algorithm is Optimal w.r.t. minimizing L_{max}

(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 σ_A (as shown in the figure)
  - (ii) and $d_a < d_b$

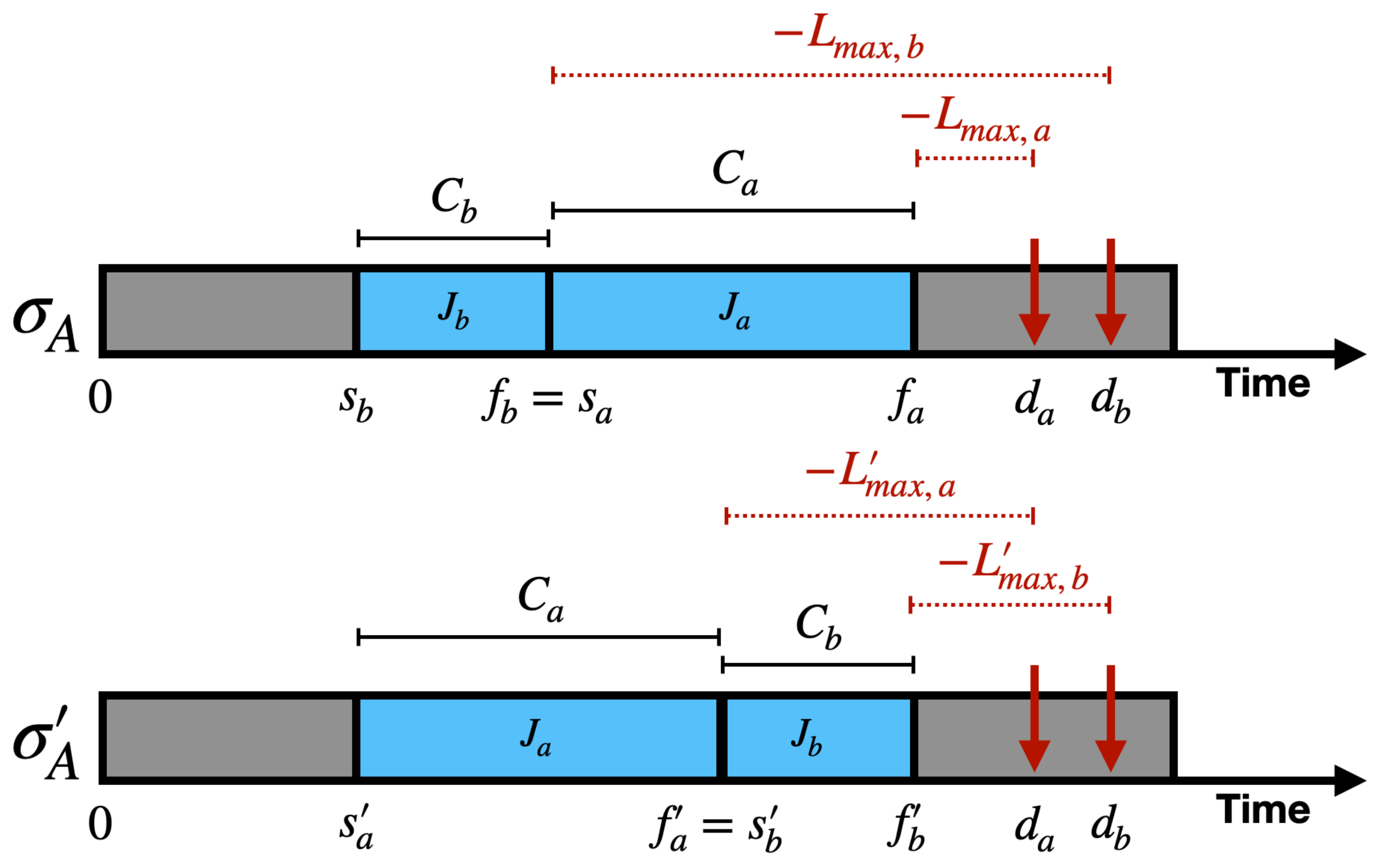
(4) We transposition ($\rightsquigarrow$) schedule σ_A to σ'_A by interchanging the time slots of J_a and J_b

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

(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 $\sigma_A \rightsquigarrow \sigma'_A \rightsquigarrow \sigma''_A \rightsquigarrow \ldots \rightsquigarrow \sigma_{Jackson}$

(10) From (4)-(8), $L_{max}(\sigma_A) \geq L_{max}(\sigma'_A) \geq L_{max}(\sigma''_A) \geq \ldots \geq L_{max}(\sigma_{Jackson})$

(11) This contradicts (3). Hence, Jackson's algorithm is indeed optimal and results in the minimum possible L_{max}

Scenario #1, Example #2

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

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

Scenario #1

Scheduling policy

- Jackon's algorithm 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.

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

Schedulability analysis

Yes?

No?

Design time claim:

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

Scenario #2

- **Given**
  - Set of n aperiodic jobs $J = \{J_1, J_2, \ \ldots, J_n\}$
  - **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 misses its deadline
- **Scheduling policy?**
  - **Earliest Deadline First (EDF):** At any instant, execute a ready task with the earliest absolute deadline
    - If there is a tie, execute the task with the smaller ID, e.g., if $d_1 = d_2$, then execute J_1 before J_2

Scenario #2, Example #1

	J_1	J_2	J_3	J_4	J_5
a_i	0	0	2	3	6
C_i	1	2	2	2	2
d_i	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}?**

Jackson's Algorithm is Optimal w.r.t. minimizing L_{max}

Do these arguments work for EDF?

(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 $L_{max}(\sigma_{Jackson})$

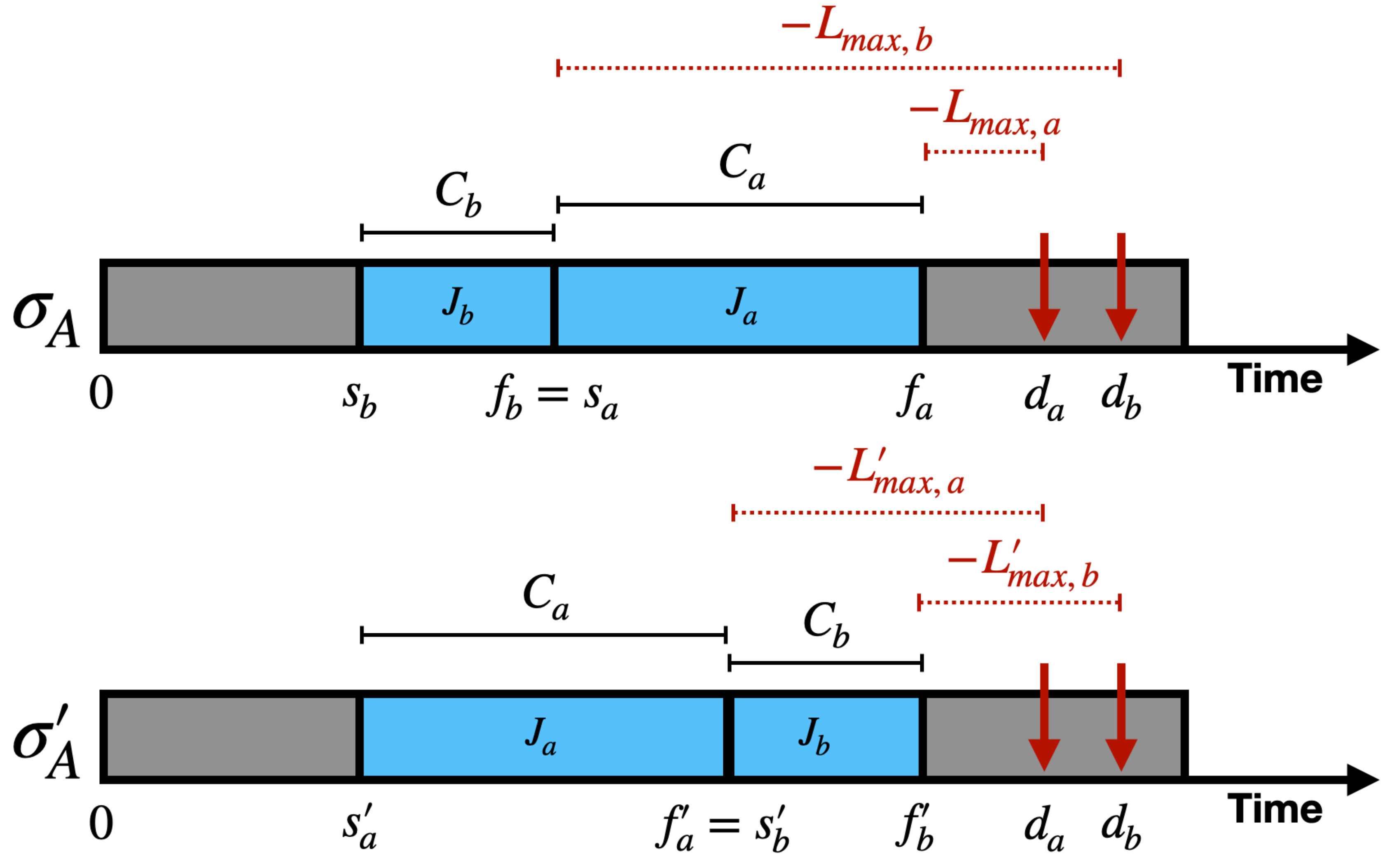
(3) Suppose there exists another algorithm A that produces schedule $\sigma_A \neq \sigma_{Jackson}$ with maximum lateness $L_{max}(\sigma_A) < L_{max}(\sigma_{Jackson})$

- (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 σ_A (as shown in the figure)
  - (ii) and $d_a < d_b$

(4) We transposition ($\rightsquigarrow$) schedule σ_A to σ'_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 σ_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 σ'_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 σ_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 σ'_A
- (II) Thus, $L_{max}(\sigma_A) = L_{max}(\sigma'_A)$ in the second case

(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 $\sigma_A \rightsquigarrow \sigma'_A \rightsquigarrow \sigma''_A \rightsquigarrow \ldots \rightsquigarrow \sigma_{Jackson}$

(10) From (4)-(8), $L_{max}(\sigma_A) \geq L_{max}(\sigma'_A) \geq L_{max}(\sigma''_A) \geq \ldots \geq L_{max}(\sigma_{Jackson})$

(11) This contradicts (3). Hence, Jackson's algorithm is indeed optimal and results in the minimum possible L_{max}

EDF is Optimal w.r.t. minimizing L_{max} [1/3]

(1) For simplicity, assume that each job has a unit execution time, i.e., $\forall i, C_i = 1$

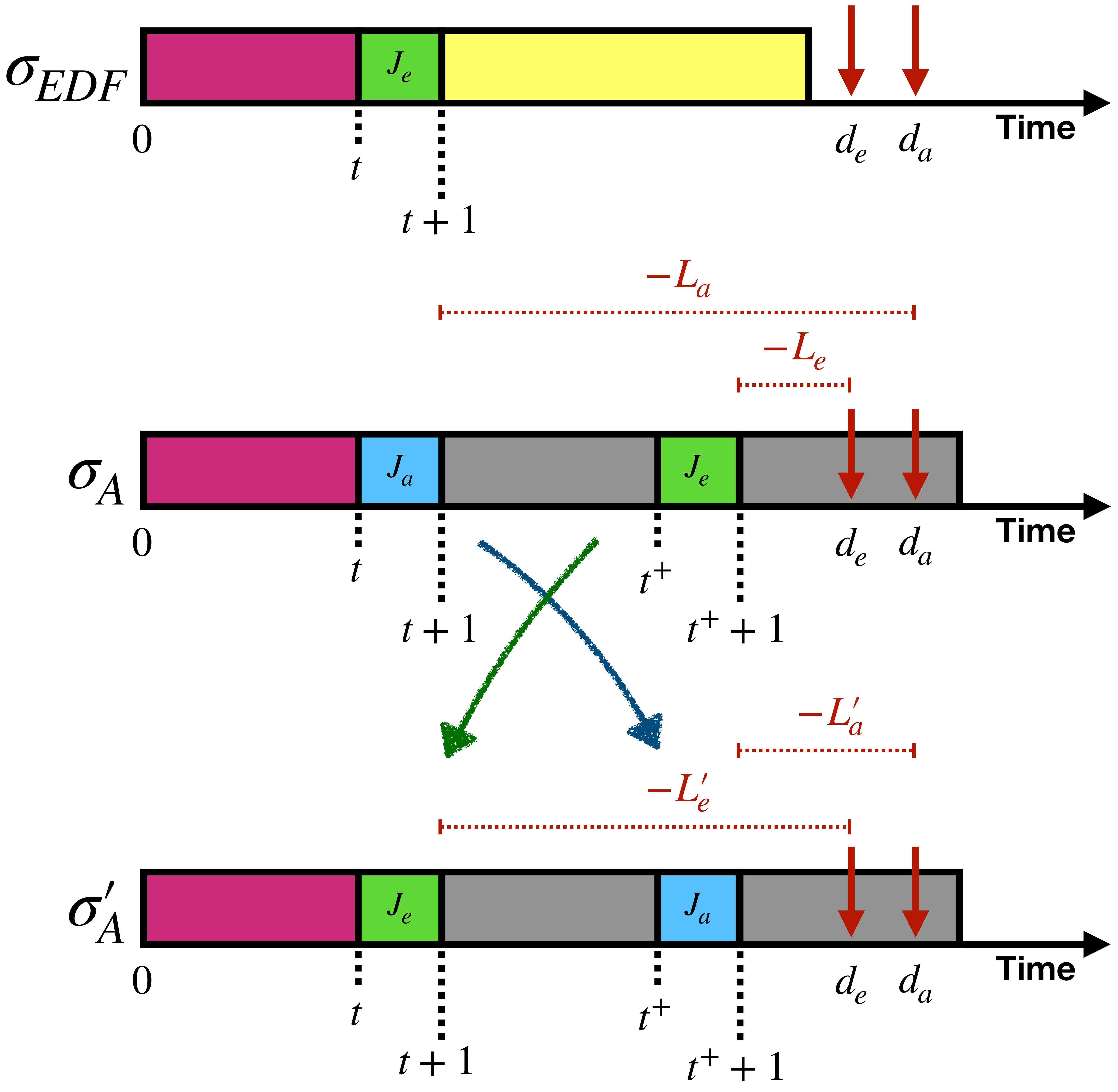
(2) Let σ_{EDF} be the schedule produced by EDF, and let $\sigma_A \neq \sigma_{EDF}$ be the schedule produced by another algorithm A

(3) We will show that EDF is optimal because

- (I) σ_A can be transformed into σ_{EDF} using a finite number of transpositions ($\rightsquigarrow$), that is, $\sigma_A \rightsquigarrow \sigma'_A \rightsquigarrow \sigma''_A \rightsquigarrow \ldots \rightsquigarrow \sigma_{EDF}$
- (II) After each transposition, the maximum lateness cannot increase

EDF is Optimal w.r.t. minimizing L_{max} [2/3]

(4) How do we define a transposition ($\rightsquigarrow$) in the case of EDF?
- (I) Since $\sigma_A \neq \sigma_{EDF}$, there exists a time t such that
    - (i) $\sigma_A(t) \neq \sigma_{EDF}(t)$ but $\forall t^- < t, \sigma_A(t^-) = \sigma_{EDF}(t^-)$
- (II) Let $\sigma_{EDF}(t) = J_e$ and $\sigma_A(t) = J_a$
    - (i) Can $\sigma_{EDF}(t) = 0$ or $\sigma_A = 0$?
        - $\sigma_{EDF}(t) = 0$ and $\sigma_A \neq 0$ is not possible …
        - $\sigma_{EDF}(t) \neq 0$ and $\sigma_A = 0$ is possible …
- (III) Suppose that J_e is executed in schedule σ_A at time $t^+ > t$
    - (i) What if J_e's execution in schedule σ_A has already completed before t? Not possible …
- (IV) Transposition $\sigma_A \rightsquigarrow \sigma'_A$
    - (i) Exchange the time slices at $[t, t+1)$ and $[t^+, t^+ + 1)$ in schedule σ_A to obtain σ'_A



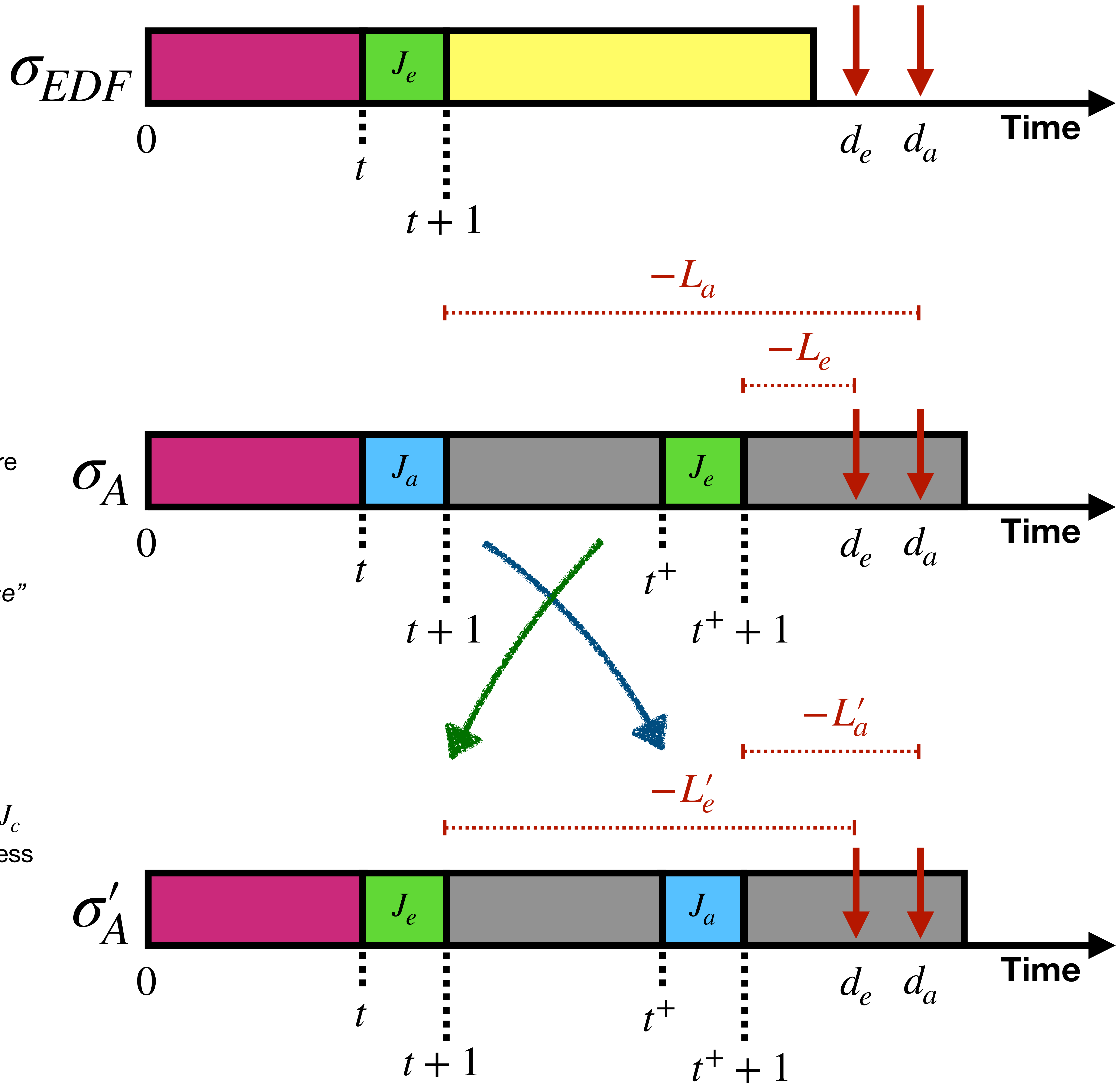
EDF is Optimal w.r.t. minimizing L_{max} [3/3]

(5) Proof of (3)(I), *"σ_A can be transformed into σ_{EDF} using a finite number of transpositions ($\rightsquigarrow$), that is, $\sigma_A \rightsquigarrow \sigma'_A \rightsquigarrow \sigma''_A \rightsquigarrow \ldots \rightsquigarrow \sigma_{EDF}$"*

- (I) After the transposition at time t, the prefixes of schedules σ_A and σ_{EDF} corresponding to time slice $[0, t+1)$ are identical
- (II) The length of prefixes identical in σ_A and σ_{EDF} increases after every transposition
- (III) Since the number of jobs is finite, beyond a certain time t_{end}, no jobs are pending, and thus $\forall t' > t_{end}, \sigma_A(t') = \sigma_{EDF}(t') = 0$

(6) Proof of (3)(II), *"After each transposition, the maximum lateness cannot increase"*

- (I) For the special case ($\sigma_{EDF}(t) \neq 0$ and $\sigma_A = 0$), the proof is trivial …
- (II) For the normal case ($\sigma_{EDF}(t) \neq 0$ and $\sigma_A \neq 0$)
  - (i) If the maximum lateness in σ_A corresponded to J_e
    - Since $L_e = \max(L_e, L_a, L'_e, L'_a)$, transposition $\sigma_A \rightsquigarrow \sigma'_A$ **can only reduce** the maximum lateness from L_e to $\max(L'_a, L'_e)$
  - (ii) If the maximum lateness in σ_A corresponded to some other job J_c
    - Transposition $\sigma_A \rightsquigarrow \sigma'_A$ has no effect on the maximum lateness



Runtime Schedulability Test for EDF

- Arbitrary arrival times
  - Schedulability test has to be done dynamically, when a new task arrives
- Jobs $J = \{J_1, J_2, \ldots, J_n\}$ are activated before t, new job J_{n+1} arrives at t
  - Is the system still schedulable?

Scenario #3

- **Given**
  - Set of n aperiodic jobs $J = \{J_1, J_2, \ldots, J_n\}$
  - 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)?**

Scenario #3, Example #1

	J_1	J_2
a_i	0	1
C_i	4	2
d_i	12	10

- Classroom assignment
  - Can you draw the EDF schedule?
  - Is the schedule feasible? Why?
  - What is L_{max}?
  - **Is this the minimum possible L_{max}?**
  - **EDF is work-conserving, hence is not optimal**
  - Why was this not a problem earlier?
  - Alternative algorithms?
    - Exhaustive search, heuristics