

---

# Basic Task Models in Real-Time Systems and Applications

Prof. Dr. Jian-Jia Chen

**LS 12, TU Dortmund**

# Fundamentals

---

- Algorithm:
- Program:
- Process/job/task:

# Fundamentals

---

- Algorithm:
  - It is the logical procedure to solve a certain **problem**
  - It is informally specified a sequence of elementary steps that an “execution machine” must follow to solve the problem
  - It is not necessarily (and usually not) expressed in a formal programming language
- Program:
- Process/job/task:

# Fundamentals

---

- Algorithm:
  - It is the logical procedure to solve a certain **problem**
  - It is informally specified a sequence of elementary steps that an “execution machine” must follow to solve the problem
  - It is not necessarily (and usually not) expressed in a formal programming language
- Program:
  - It is the implementation of an algorithm in a programming language
  - It can be executed several times with different inputs
- Process/job/task:

# Fundamentals

---

- Algorithm:
  - It is the logical procedure to solve a certain **problem**
  - It is informally specified a sequence of elementary steps that an “execution machine” must follow to solve the problem
  - It is not necessarily (and usually not) expressed in a formal programming language
- Program:
  - It is the implementation of an algorithm in a programming language
  - It can be executed several times with different inputs
- Process/job/task:
  - An instance of a program that given a sequence of inputs produces a set of outputs

# Operating System

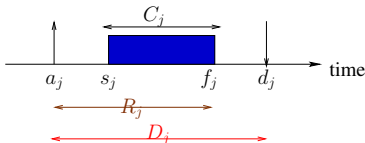
---

An operating system is a program that

- acts as an intermediary between a user of a computer and the computer hardware by providing interfaces
- provides an “abstraction” of the physical machine (for example, a file, a virtual page in memory, etc.)
- **manages the access to the physical resources of a computing machine**
- makes the computer system convenient to use
- executes user programs and makes solving user problems easier
- and more . . . . .

# Timing parameters of a job $J_j$

- Arrival time ( $a_j$ ) or release time ( $r_j$ ) is the time at which the job becomes ready for execution
- Computation (execution) time ( $C_j$ ) is the time necessary to the processor for executing the job without interruption (= WCET).
- Absolute deadline ( $d_j$ ) is the time at which the job should be completed.
- Relative deadline ( $D_j$ ) is the time length between the arrival time and the absolute deadline.
- Start time ( $s_j$ ) is the time at which the job starts its execution.
- Finishing time ( $f_j$ ) is the time at which the job finishes its execution.
- Response time ( $R_j$ ) is the time length at which the job finishes its execution after its arrival, which is  $f_j - a_j$ .



# Multi-Tasking (Recap)

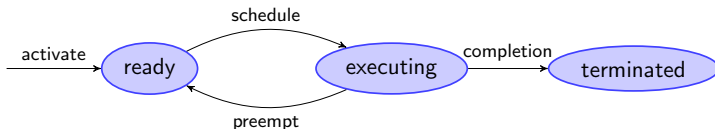
---

- The execution entities (tasks, processes, threads, etc.) are competing from each other for shared resources
- Scheduling policy is needed
  - When to schedule an entity?
  - Which entity to schedule?
  - How to schedule entities?



# Scheduling Concepts

- **Scheduling Algorithm:** determines the order that jobs execute on the processor
- Jobs (a simplified version) may be in one of three states:



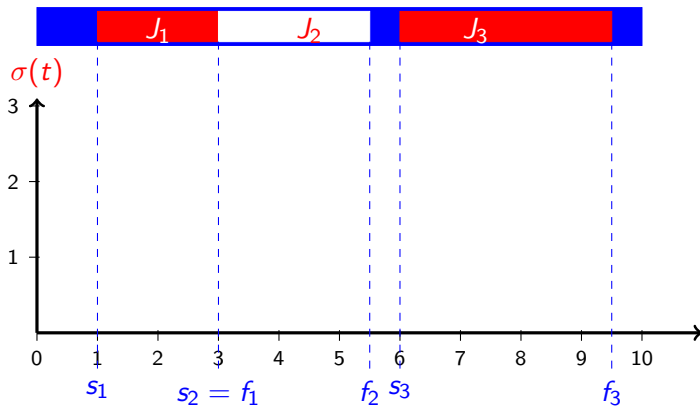
# Schedules for a set of jobs $\{J_1, J_2, \dots, J_N\}$

---

- A schedule is an assignment of jobs to the processor, such that each job is executed until completion.
- A schedule can be defined as an integer step function  $\sigma : \mathbb{R} \rightarrow \mathbb{N}$ , where  $\sigma(t) = j$  denotes job  $J_j$  is executed at time  $t$ , and  $\sigma(t) = 0$  denotes the system is idle at time  $t$ .
- If  $\sigma(t)$  changes its value at some time  $t$ , then the processor performs a context switch at time  $t$ .
- Non-preemptive scheduling: there is only one interval with  $\sigma(t) = j$  for every  $J_j$ , where  $t$  is covered by the interval.
- Preemptive scheduling: there could be more than one interval with  $\sigma(t) = j$ .

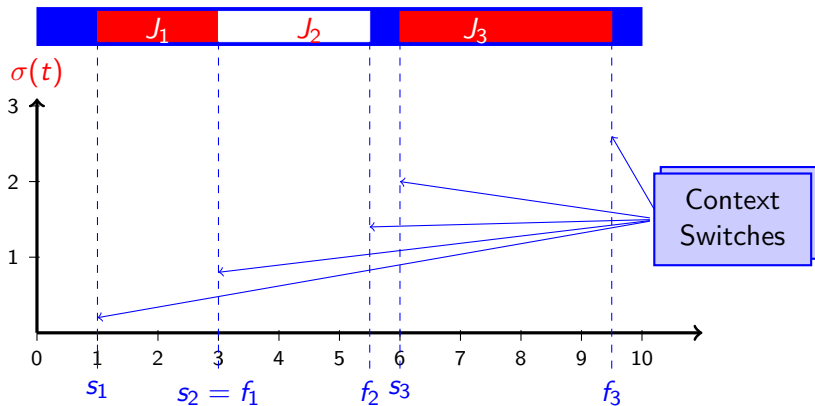
# Scheduling Concept: Non-preemptive

**Schedule:**  $\sigma : \mathbb{R} \rightarrow \mathbb{N}$  function of processor time to jobs



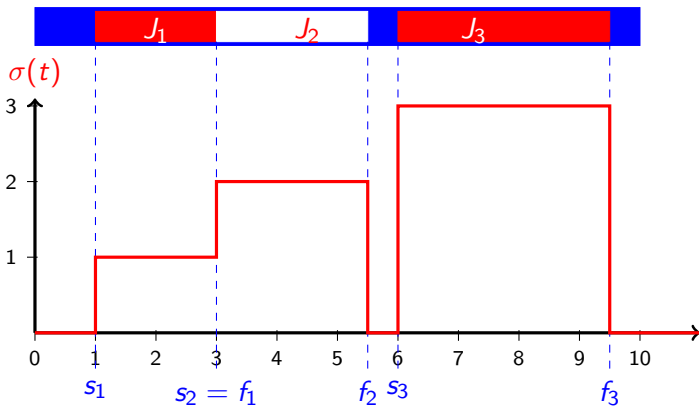
# Scheduling Concept: Non-preemptive

**Schedule:**  $\sigma : \mathbb{R} \rightarrow \mathbb{N}$  function of processor time to jobs



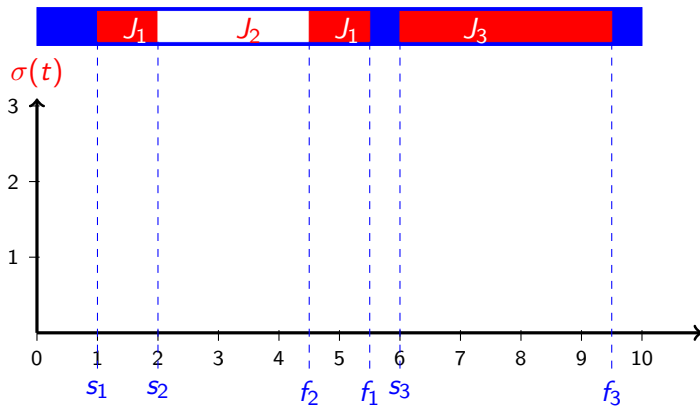
# Scheduling Concept: Non-preemptive

**Schedule:**  $\sigma : \mathbb{R} \rightarrow \mathbb{N}$  function of processor time to jobs



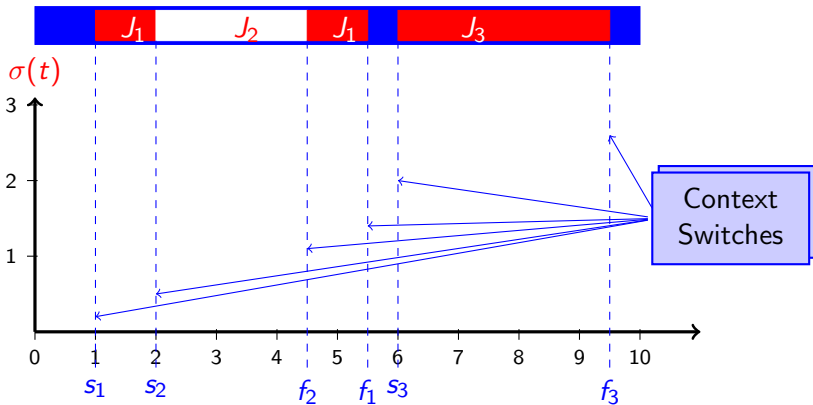
# Scheduling Concept: Preemptive

**Schedule:**  $\sigma : \mathbb{R} \rightarrow \mathbb{N}$  function of processor time to jobs



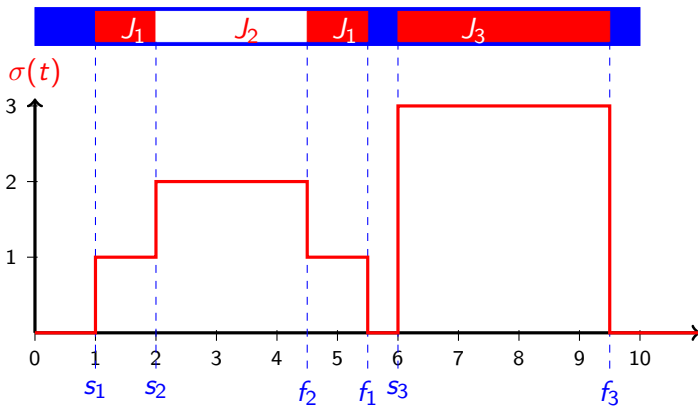
# Scheduling Concept: Preemptive

**Schedule:**  $\sigma : \mathbb{R} \rightarrow \mathbb{N}$  function of processor time to jobs



# Scheduling Concept: Preemptive

**Schedule:**  $\sigma : \mathbb{R} \rightarrow \mathbb{N}$  function of processor time to jobs





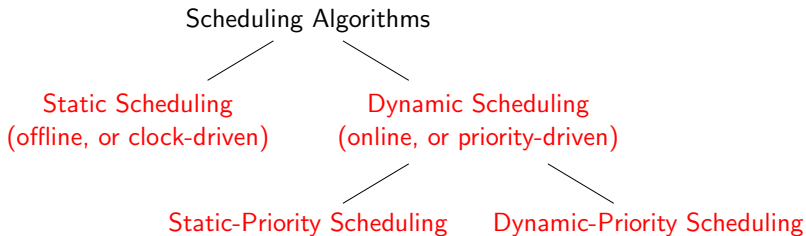
# Feasibility of Schedules and Schedulability

---

- A schedule is **feasible** if all jobs can be completed according to a set of specified constraints.
- A set of jobs is **schedulable** if there exists a feasible schedule for the set of jobs.
- A scheduling algorithm is **optimal** if it always produces a feasible schedule when one exists (under any scheduling algorithm).

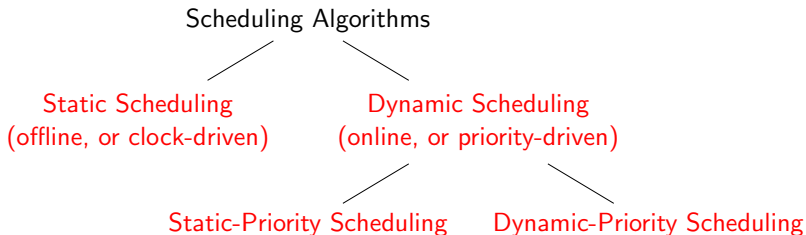
# Scheduling Algorithms

---



# Scheduling Algorithms

---

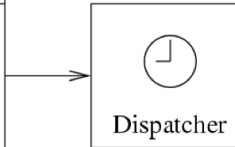


- Preemptive vs. Non-preemptive
- Guarantee-Based vs. Best-Effort
- Optimal vs. Non-optimal

# Static/offline scheduling

Scheduling taking a priori knowledge about arrival times, execution times, and deadlines into account. Dispatcher allocates processor when interrupted by timer. Timer controlled by a table generated at design time.

Time	Action	WCET
10	<i>Start Task<sub>1</sub></i>	12
17	send M5	
22	<i>Start Task<sub>3</sub></i>	20
38	<i>Start Task<sub>2</sub></i>	
47	send M3	



# Time-Triggered Systems

---

*In an entirely time-triggered system, the temporal control structure of all tasks is established a priori by off-line support-tools. This temporal control structure is encoded in a Task-Descriptor List (TDL) that contains the cyclic schedule for all activities of the node. This schedule considers the required precedence and mutual exclusion relationships among the tasks such that an explicit coordination of the tasks by the operating system at run time is not necessary...*

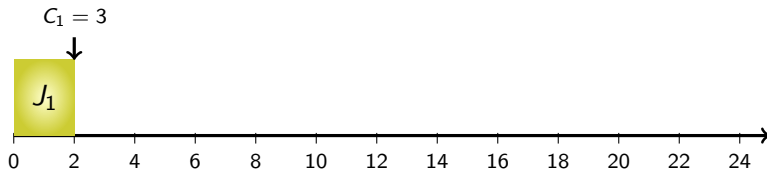
The dispatcher is activated by the synchronized clock tick. It looks at the TDL, and then performs the action that has been planned for this instant [Kopetz].

The disadvantage is that the response to sporadic events may be poor.

# An Example: Shortest-Job-First (SJF)

- At any moment, the system executes the job with the *shortest* remaining time among the jobs in the ready queue.

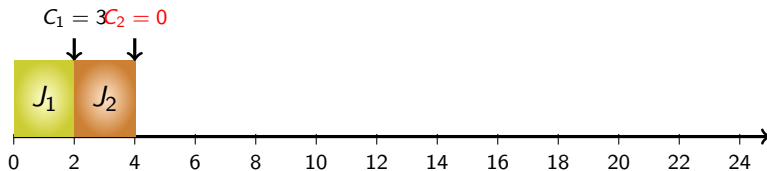
	$J_1$	$J_2$	$J_3$	$J_4$	$J_5$
$a_j$	0	2	8	10	15
$C_j$	5	2	6	3	4
$d_j$	6	8	20	14	22



# An Example: Shortest-Job-First (SJF)

- At any moment, the system executes the job with the *shortest* remaining time among the jobs in the ready queue.

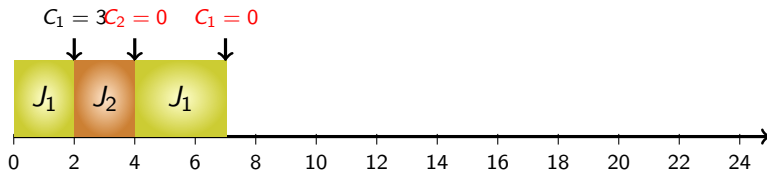
	$J_1$	$J_2$	$J_3$	$J_4$	$J_5$
$a_j$	0	2	8	10	15
$C_j$	5	2	6	3	4
$d_j$	6	8	20	14	22



# An Example: Shortest-Job-First (SJF)

- At any moment, the system executes the job with the *shortest* remaining time among the jobs in the ready queue.

	$J_1$	$J_2$	$J_3$	$J_4$	$J_5$
$a_j$	0	2	8	10	15
$C_j$	5	2	6	3	4
$d_j$	6	8	20	14	22

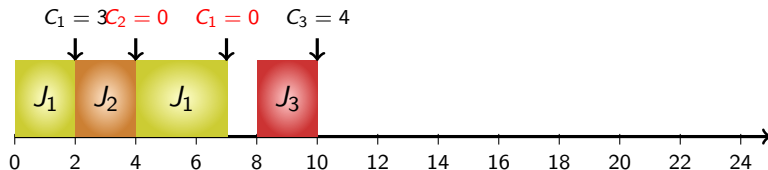




# An Example: Shortest-Job-First (SJF)

- At any moment, the system executes the job with the *shortest* remaining time among the jobs in the ready queue.

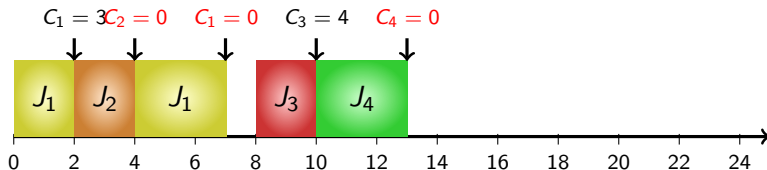
	$J_1$	$J_2$	$J_3$	$J_4$	$J_5$
$a_j$	0	2	8	10	15
$C_j$	5	2	6	3	4
$d_j$	6	8	20	14	22



# An Example: Shortest-Job-First (SJF)

- At any moment, the system executes the job with the *shortest* remaining time among the jobs in the ready queue.

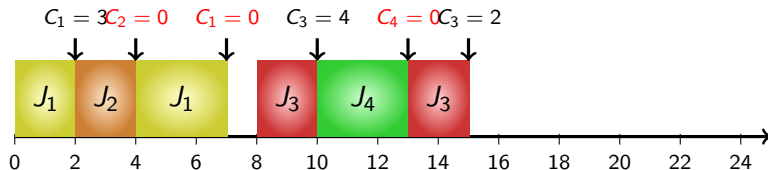
	$J_1$	$J_2$	$J_3$	$J_4$	$J_5$
$a_j$	0	2	8	10	15
$C_j$	5	2	6	3	4
$d_j$	6	8	20	14	22



# An Example: Shortest-Job-First (SJF)

- At any moment, the system executes the job with the *shortest* remaining time among the jobs in the ready queue.

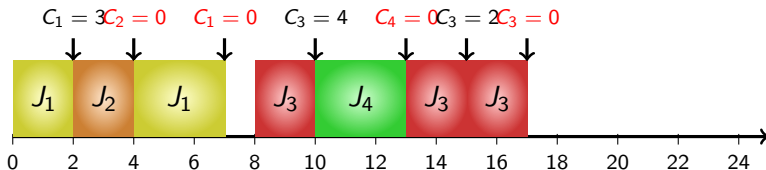
	$J_1$	$J_2$	$J_3$	$J_4$	$J_5$
$a_j$	0	2	8	10	15
$C_j$	5	2	6	3	4
$d_j$	6	8	20	14	22



# An Example: Shortest-Job-First (SJF)

- At any moment, the system executes the job with the *shortest* remaining time among the jobs in the ready queue.

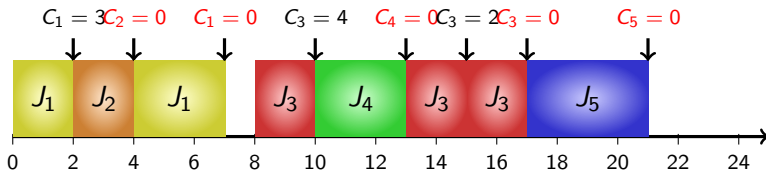
	$J_1$	$J_2$	$J_3$	$J_4$	$J_5$
$a_j$	0	2	8	10	15
$C_j$	5	2	6	3	4
$d_j$	6	8	20	14	22



# An Example: Shortest-Job-First (SJF)

- At any moment, the system executes the job with the *shortest* remaining time among the jobs in the ready queue.

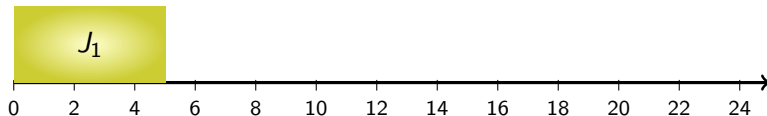
	$J_1$	$J_2$	$J_3$	$J_4$	$J_5$
$a_j$	0	2	8	10	15
$C_j$	5	2	6	3	4
$d_j$	6	8	20	14	22



# An Example: Earliest-Deadline-First (EDF)

- At any moment, the system executes the job with the *earliest absolute deadline* among the jobs in the ready queue.

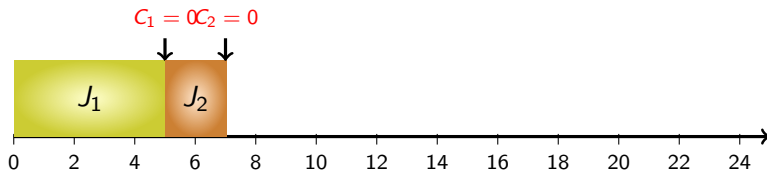
	$J_1$	$J_2$	$J_3$	$J_4$	$J_5$
$a_j$	0	2	8	10	15
$C_j$	5	2	6	3	4
$d_j$	6	8	20	14	22



# An Example: Earliest-Deadline-First (EDF)

- At any moment, the system executes the job with the *earliest absolute deadline* among the jobs in the ready queue.

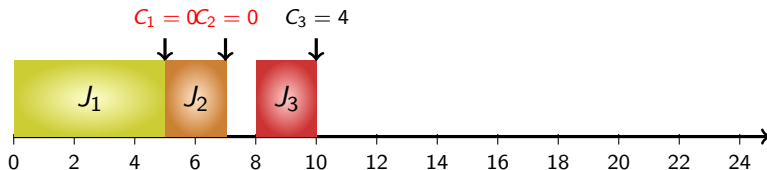
	$J_1$	$J_2$	$J_3$	$J_4$	$J_5$
$a_j$	0	2	8	10	15
$C_j$	5	2	6	3	4
$d_j$	6	8	20	14	22



# An Example: Earliest-Deadline-First (EDF)

- At any moment, the system executes the job with the *earliest absolute deadline* among the jobs in the ready queue.

	$J_1$	$J_2$	$J_3$	$J_4$	$J_5$
$a_j$	0	2	8	10	15
$C_j$	5	2	6	3	4
$d_j$	6	8	20	14	22

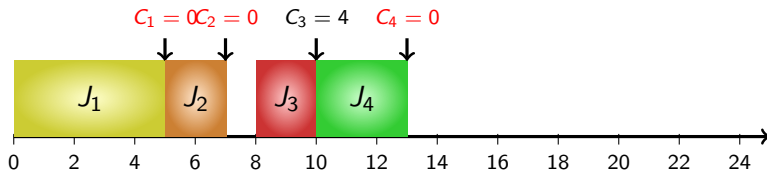




# An Example: Earliest-Deadline-First (EDF)

- At any moment, the system executes the job with the *earliest absolute deadline* among the jobs in the ready queue.

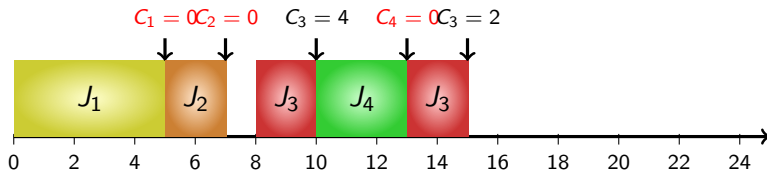
	$J_1$	$J_2$	$J_3$	$J_4$	$J_5$
$a_j$	0	2	8	10	15
$C_j$	5	2	6	3	4
$d_j$	6	8	20	14	22



# An Example: Earliest-Deadline-First (EDF)

- At any moment, the system executes the job with the *earliest absolute deadline* among the jobs in the ready queue.

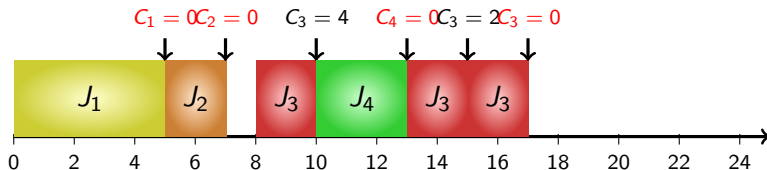
	$J_1$	$J_2$	$J_3$	$J_4$	$J_5$
$a_j$	0	2	8	10	15
$C_j$	5	2	6	3	4
$d_j$	6	8	20	14	22



# An Example: Earliest-Deadline-First (EDF)

- At any moment, the system executes the job with the *earliest absolute deadline* among the jobs in the ready queue.

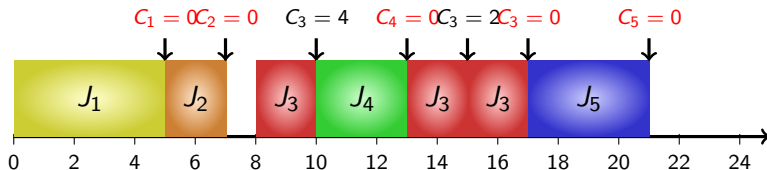
	$J_1$	$J_2$	$J_3$	$J_4$	$J_5$
$a_j$	0	2	8	10	15
$C_j$	5	2	6	3	4
$d_j$	6	8	20	14	22



# An Example: Earliest-Deadline-First (EDF)

- At any moment, the system executes the job with the *earliest absolute deadline* among the jobs in the ready queue.

	$J_1$	$J_2$	$J_3$	$J_4$	$J_5$
$a_j$	0	2	8	10	15
$C_j$	5	2	6	3	4
$d_j$	6	8	20	14	22



# Evaluating A Schedule

---

For a job  $J_j$ :

- Lateness  $L_j$ : delay of job completion with respect to its deadline.

$$L_j = f_j - d_j$$

- Tardiness  $E_j$ : the time that a job stays active after its deadline.

$$E_j = \max\{0, L_j\}$$

- Laxity (or Slack Time)( $X_j$ ): The maximum time that a job can be delayed and still meet its deadline.

$$X_j = d_j - a_j - C_j$$

# Metrics of Scheduling Algorithms (for Jobs)

Given a set  $\mathbb{J}$  of  $n$  jobs, common metrics are to minimize

- Average response time:

$$\sum_{J_j \in \mathbb{J}} \frac{f_j - a_j}{|\mathbb{J}|}$$

- Makespan (total completion time):

$$\max_{J_j \in \mathbb{J}} f_j - \min_{J_j \in \mathbb{J}} a_j$$

- Total weighted response time:

$$\sum_{J_j \in \mathbb{J}} w_j (f_j - a_j)$$

- Maximum latency:

$$L_{\max} = \max_{J_j \in \mathbb{J}} (f_j - d_j)$$

- Number of late jobs:

$$N_{late} = \sum_{J_j \in \mathbb{J}} miss(J_j),$$

where  $miss(J_j) = 0$  if  $f_j \leq d_j$ , and  $miss(J_j) = 1$  otherwise.

# Hard/Soft Real-Time Systems

---

- Hard Real-Time Systems
  - If any hard deadline is ever missed, then the system is incorrect
  - The tardiness for any job must be 0
  - **Examples:** Nuclear power plant control, flight control
- Soft Real-Time Systems
  - A soft deadline may **occasionally** be missed
  - Various definitions for “occasionally”
    - minimize the number of tardy jobs, minimize the maximum lateness, etc.
  - **Examples:** Telephone switches, multimedia applications

# Hard/Soft Real-Time Systems

---

- Hard Real-Time Systems
  - If any hard deadline is ever missed, then the system is incorrect
  - The tardiness for any job must be 0
  - **Examples:** Nuclear power plant control, flight control
- Soft Real-Time Systems
  - A soft deadline may **occasionally** be missed
  - Various definitions for “occasionally”
    - minimize the number of tardy jobs, minimize the maximum lateness, etc.
  - **Examples:** Telephone switches, multimedia applications

We mostly consider hard real-time systems in this course.



# Recurrent Task Models

---

- When jobs (usually with the same computation requirement) are released recurrently, these jobs can be modeled by a recurrent task
- **Periodic Task**  $\tau_i$ :
  - A job is released exactly and periodically by a period  $T_i$
  - A phase  $\phi_i$  indicates when the first job is released
  - A relative deadline  $D_i$  for each job from task  $\tau_i$
  - $(\phi_i, C_i, T_i, D_i)$  is the specification of periodic task  $\tau_i$ , where  $C_i$  is the worst-case execution time.

# Recurrent Task Models

---

- When jobs (usually with the same computation requirement) are released recurrently, these jobs can be modeled by a recurrent task
- **Periodic Task  $\tau_i$ :**
  - A job is released exactly and periodically by a period  $T_i$
  - A phase  $\phi_i$  indicates when the first job is released
  - A relative deadline  $D_i$  for each job from task  $\tau_i$
  - $(\phi_i, C_i, T_i, D_i)$  is the specification of periodic task  $\tau_i$ , where  $C_i$  is the worst-case execution time.
- **Sporadic Task  $\tau_i$ :**
  - $T_i$  is the minimal time between any two consecutive job releases
  - A relative deadline  $D_i$  for each job from task  $\tau_i$
  - $(C_i, T_i, D_i)$  is the specification of sporadic task  $\tau_i$ , where  $C_i$  is the worst-case execution time.

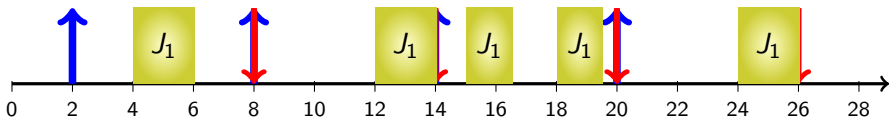
# Recurrent Task Models

---

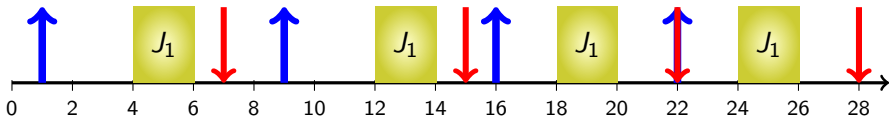
- When jobs (usually with the same computation requirement) are released recurrently, these jobs can be modeled by a recurrent task
- **Periodic Task**  $\tau_i$ :
  - A job is released exactly and periodically by a period  $T_i$
  - A phase  $\phi_i$  indicates when the first job is released
  - A relative deadline  $D_i$  for each job from task  $\tau_i$
  - $(\phi_i, C_i, T_i, D_i)$  is the specification of periodic task  $\tau_i$ , where  $C_i$  is the worst-case execution time.
- **Sporadic Task**  $\tau_i$ :
  - $T_i$  is the minimal time between any two consecutive job releases
  - A relative deadline  $D_i$  for each job from task  $\tau_i$
  - $(C_i, T_i, D_i)$  is the specification of sporadic task  $\tau_i$ , where  $C_i$  is the worst-case execution time.
- **Aperiodic Task**: Identical jobs released arbitrarily (we will revisit this part in Real-Time Calculus).

# Examples of Recurrent Task Models

**Periodic task:**  $(\phi_i, C_i, T_i, D_i) = (2, 2, 6, 6)$



**Sporadic task:**  $(C_i, T_i, D_i) = (2, 6, 6)$



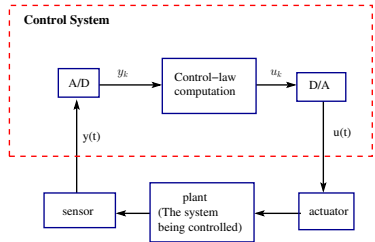
# Example: Sporadic Control System

## Pseudo-code for this system

**while (true)**

- `start := get the system tick;`
- `perform analog-to-digital conversion to get  $y$ ;`
- `compute control output  $u$ ;`
- `output  $u$  and do digital-to-analog conversion;`
- `end := get the system tick;`
- `$timeToSleep := T - (end - start)$ ;`
- `sleep  $timeToSleep$ ;`

**end while**



# Example: Periodic Control System

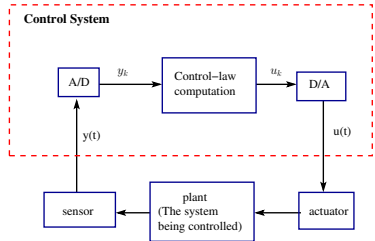
## Pseudo-code for this system

set timer to interrupt periodically  
with period  $T$ ;

at each timer interrupt  
do

- perform analog-to-digital conversion to get  $y$ ;
- compute control output  $u$ ;
- output  $u$  and do digital-to-analog conversion;

od



# Evaluating A Schedule for Tasks

---

For a job  $J_j$ :

- Lateness  $L_j$ : delay of job completion with respect to its deadline.

$$L_j = f_j - d_j$$

- Tardiness  $E_j$ : the time that a job stays active after its deadline.

$$E_j = \max\{0, L_j\}$$

- Laxity (or Slack Time)( $X_j$ ): The maximum time that a job can be delayed and still meet its deadline.

$$X_j = d_j - a_j - C_j$$

For a task  $\tau_i$ :

- Lateness  $L_i$ : maximum latency of jobs released by task  $\tau_i$
- Tardiness  $E_i$ : maximum tardiness of jobs released by task  $\tau_i$
- Laxity  $X_i$ :  $D_i - C_i$

# Relative Deadline $\leq$ Period

---

For a task set, we say that the task set is with

- **implicit deadline** when the relative deadline  $D_i$  is equal to the period  $T_i$ , i.e.,  $D_i = T_i$ , for every task  $\tau_i$ ,
- **constrained deadline** when the relative deadline  $D_i$  is no more than the period  $T_i$ , i.e.,  $D_i \leq T_i$ , for every task  $\tau_i$ , and
- **arbitrary deadline** when the relative deadline  $D_i$  could be larger than the period  $T_i$  for some task  $\tau_i$ .



# Feasibility and Schedulability for Recurrent Tasks

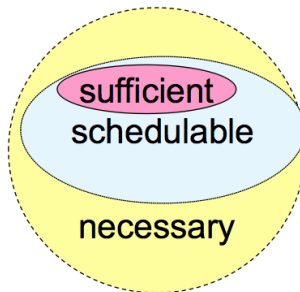
---

- A schedule is **feasible** if all the jobs of all tasks can be completed according to a set of specified constraints.
- A set of tasks is **schedulable** if there exists a feasible schedule for the set of tasks.
- A set of tasks is **schedulable by a scheduling algorithm** if the schedule is always feasible.
- A scheduling algorithm is **optimal** if it always produces a feasible schedule when one exists (under any scheduling algorithm).

# Different Tests

The issue for timing analysis is on how to analyze the schedulability.

- *Sufficient Test*: If A holds, then the task set is schedulable (by a scheduling algorithm).
- *Necessary Test*: If the task set is schedulable (by a scheduling algorithm), then B holds.
- *Exact Test*: the task set is schedulable (by a scheduling algorithm) if and only if  $A^*$  holds.



# Monotonicity of Scheduling Algorithms

A good scheduling algorithm should be monotonic

- If a scheduling algorithm derives a feasible schedule, it should also guarantee the feasibility with
  - less execution time of a task/job,
  - less number of tasks/jobs, or
  - more number of processors/machines.

# Why is Real-Time Scheduling Hard?

Single-processor (Ekberg and Wang, in ECRTS 2015)

EDF Scheduling: Exact schedulability test for constrained-deadline periodic task systems  $\text{co}\mathcal{NP}$ -Hard in the strong sense.

Single-processor (Eisenbrand and Rothvoß, in RTSS 2008)

Fixed-Priority Real-Time Scheduling: Response Time Computation Is  $\mathcal{NP}$ -Hard (in the weak sense).

Multiprocessor (Graham 1976)

Changing the priority order, increasing the number of processors, reducing execution times, or weakening precedence constraints can result in a deadline miss.

Many Cases: Scheduling problems in multiprocessor systems are usually  $\mathcal{NP}$ -Hard.

# Fundamentals: Computational Complexity

---

- $\mathcal{NP}$ -Complete for a problem  $\Pi$ :
  - If  $\Pi$  can be solved in polynomial time by using a **non-deterministic Turing machine**, the problem is said in the computational complexity class  $\mathcal{NP}$ .
  - $\Pi$  is  $\mathcal{NP}$ -Complete if  $\Pi$  is in  $\mathcal{NP}$  and any problem in the  $\mathcal{NP}$  class can **reduce** to  $\Pi$  in polynomial time (or log space).
  - $\Pi$  is  $\mathcal{NP}$ -hard if any problem in the  $\mathcal{NP}$  class can **reduce** to  $\Pi$  in polynomial time (or log space).

# Fundamentals: Computational Complexity

---

- $\mathcal{NP}$ -Complete for a problem  $\Pi$ :
  - If  $\Pi$  can be solved in polynomial time by using a **non-deterministic Turing machine**, the problem is said in the computational complexity class  $\mathcal{NP}$ .
  - $\Pi$  is  $\mathcal{NP}$ -Complete if  $\Pi$  is in  $\mathcal{NP}$  and any problem in the  $\mathcal{NP}$  class can **reduce** to  $\Pi$  in polynomial time (or log space).
  - $\Pi$  is  $\mathcal{NP}$ -hard if any problem in the  $\mathcal{NP}$  class can **reduce** to  $\Pi$  in polynomial time (or log space).
- More intuitively (informally)
  - The computing machines we have developed so far are deterministic Turing machines.
  - If  $\Pi$  can be solved in polynomial time by using a **deterministic Turing machine**, the problem is said in the computational complexity class  $\mathcal{P}$ .
  - If a problem is  $\mathcal{NP}$ -Complete or  $\mathcal{NP}$ -hard, there is no efficient (polynomial-time) algorithm to derive optimal/feasible solutions unless  $\mathcal{P} = \mathcal{NP}$ .

# Multiprocessor Anomalies

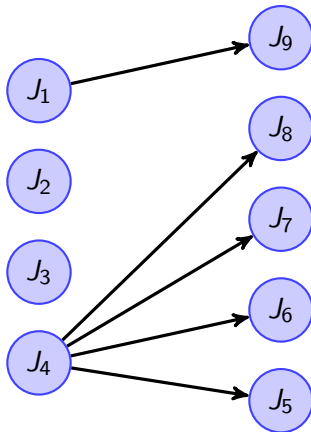
---

- Partitioned scheduling (Each task/job is on a processor)
  - As most partitioning algorithms are not optimal, a system might become infeasible with
    - Less execution time of a task/job
    - Less number of tasks/jobs
    - More number of processors/machines
- Global scheduling
  - As most priority-assignment algorithms are not optimal, a system might become infeasible with
    - Less execution time of a task/job
    - Less number of tasks/jobs
    - More number of processors/machines

# Precedence Constraints

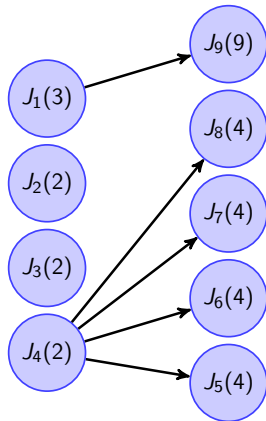
---

Jobs (and tasks) may have to execute in a pre-specified order.

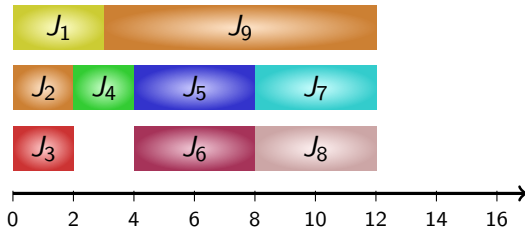




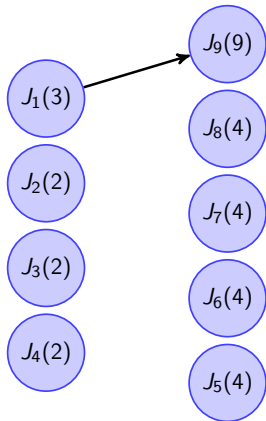
# Multiprocessor Anomaly: Case 1



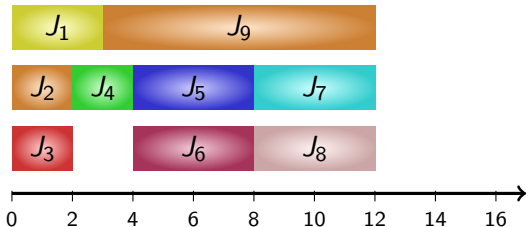
On 3 processors



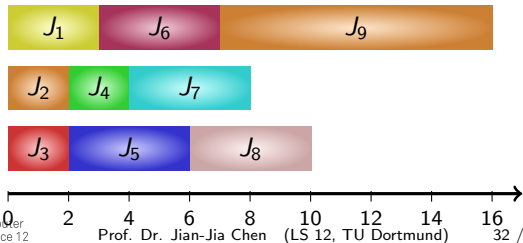
# Multiprocessor Anomaly: Case 1



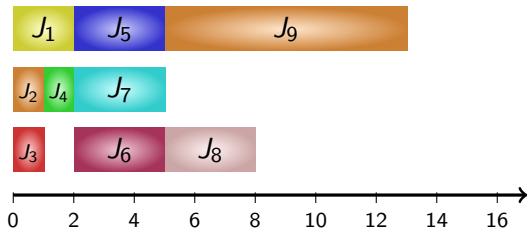
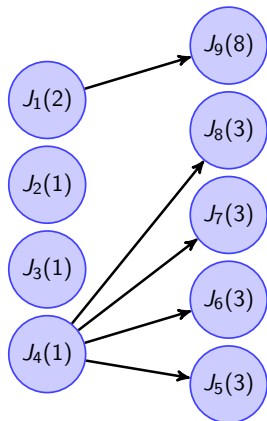
On 3 processors



Removing the precedence constraints on  $J_4$ ...

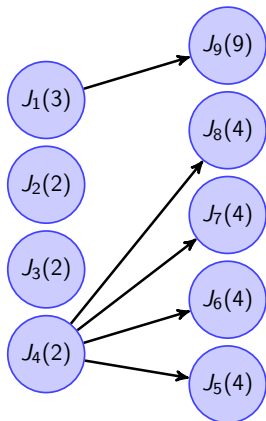


# Multiprocessor Anomaly: Case 2

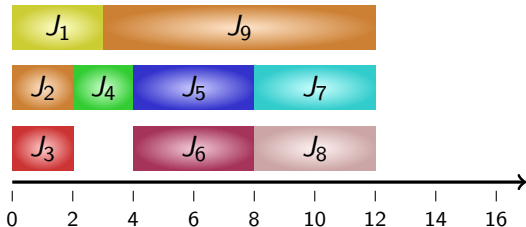


Reduce the execution  
time by 1, and schedule  
on 3 processors

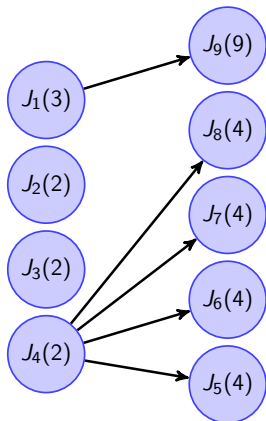
# Multiprocessor Anomaly: Case 3



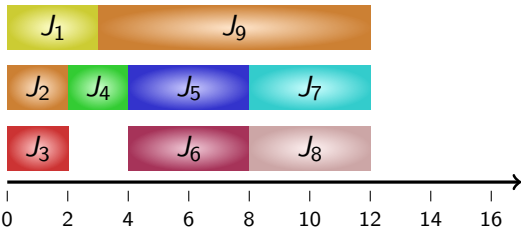
On 4 processors



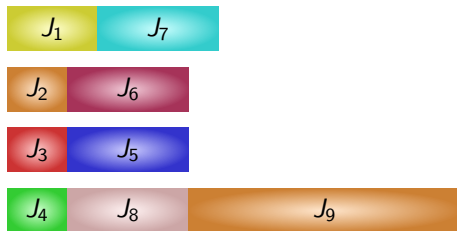
# Multiprocessor Anomaly: Case 3



On 4 processors



Use 4 processors



# Note about the Material in the Course

---

- For the rest of the course, if the context is not emphasized, we will focus on the design and analysis for scheduling algorithms with
  - preemptive scheduling and
  - independent tasks/jobs
- We will have a few sessions, discussing
  - non-preemptive scheduling or
  - tasks with shared resources (namely, critical sections),
  - tasks (jobs) with precedence constraints (namely, one job has to wait until another job finishes).

# Note about the Material in the Course

---

- For the rest of the course, if the context is not emphasized, we will focus on the design and analysis for scheduling algorithms with
  - preemptive scheduling and
  - independent tasks/jobs
- We will have a few sessions, discussing
  - non-preemptive scheduling or
  - tasks with shared resources (namely, critical sections),
  - tasks (jobs) with precedence constraints (namely, one job has to wait until another job finishes).

The material this week has covered the corresponding contents in Chapters 2 and 3 in Buttazzo's textbook.