# CIS 721 - Real-Time Systems
# **Lecture 1: Introduction**

Mitch Neilsen

neilsen@ksu.edu

Office: 219D Nichols Hall

# Course Outline

- Scheduling Theory
  - Real-Time Scheduling Algorithms
  - Feasibility/Schedulability Analysis
- Design Methodologies
- Verification and Validation

# What is a real-time system?

- A real-time system is a system whose specification includes both **logical** and **temporal** correctness criteria.
  - Logical correctness means that the system **produces correct output**. System correctness can be checked using standard Hoare logic.
  - Temporal correctness means that the output is **generated at the “right” time**, and can be checked using interval logics.

# Outline for Today

- References - where to get more information
- Typical Real-Time Applications (Ch. 1)
- Basic Terms and Concepts (Ch. 2-3)
  - Definitions
  - Misconceptions
- Read Chapters 1-3

# References

- Book:
  - Jane W.S. Liu, Real-Time Systems, 2000, Prentice Hall (Pub.).
  - Phillip A. Laplante, Seppo J. Ovaska, Real-Time Systems Design and Analysis, 2014, Wiley (Pub.).
- Course web site: CIS 721 - Real-Time Systems
  - **http://online.ksu.edu/**
- Conference Papers:
  - Various, handouts or papers available on-line in the Papers folder

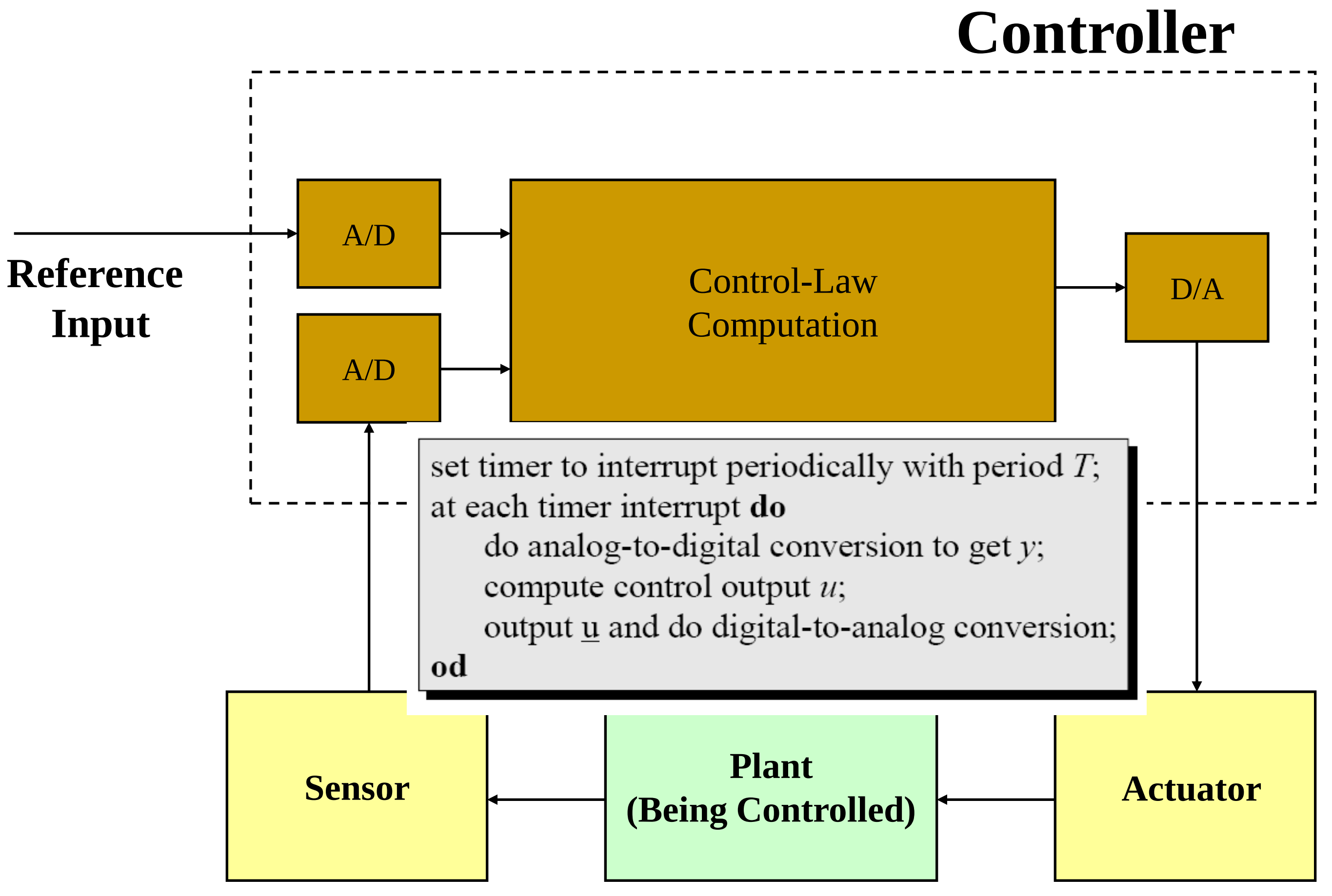
# Outline

- References - where to get more info.
- **Typical Real-Time Applications**
- Terms and Concepts

# Typical Real-Time Applications

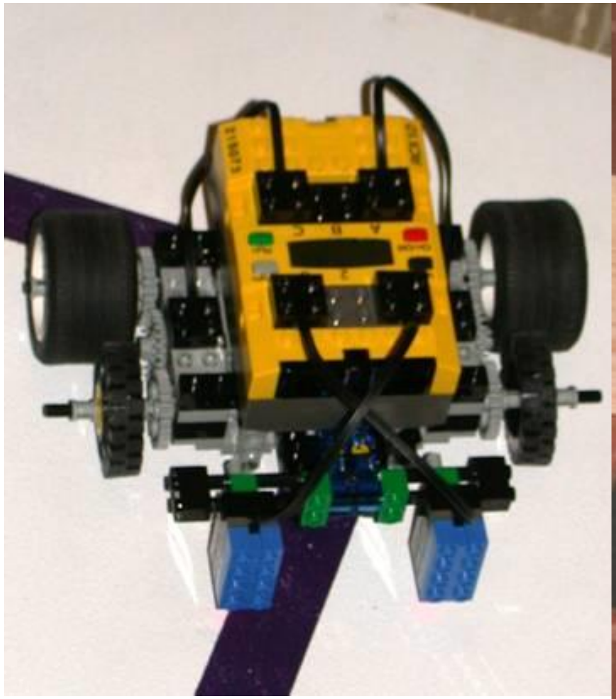
- Digital Controllers
  - Automotive Controllers
  - Industrial Automation
- High-Level Controllers
  - Command and Control Systems
  - Air Traffic Control Systems
  - Avionic Systems
- Image and Signal Processing
- Real-Time Databases and Multimedia

# Digital Controller

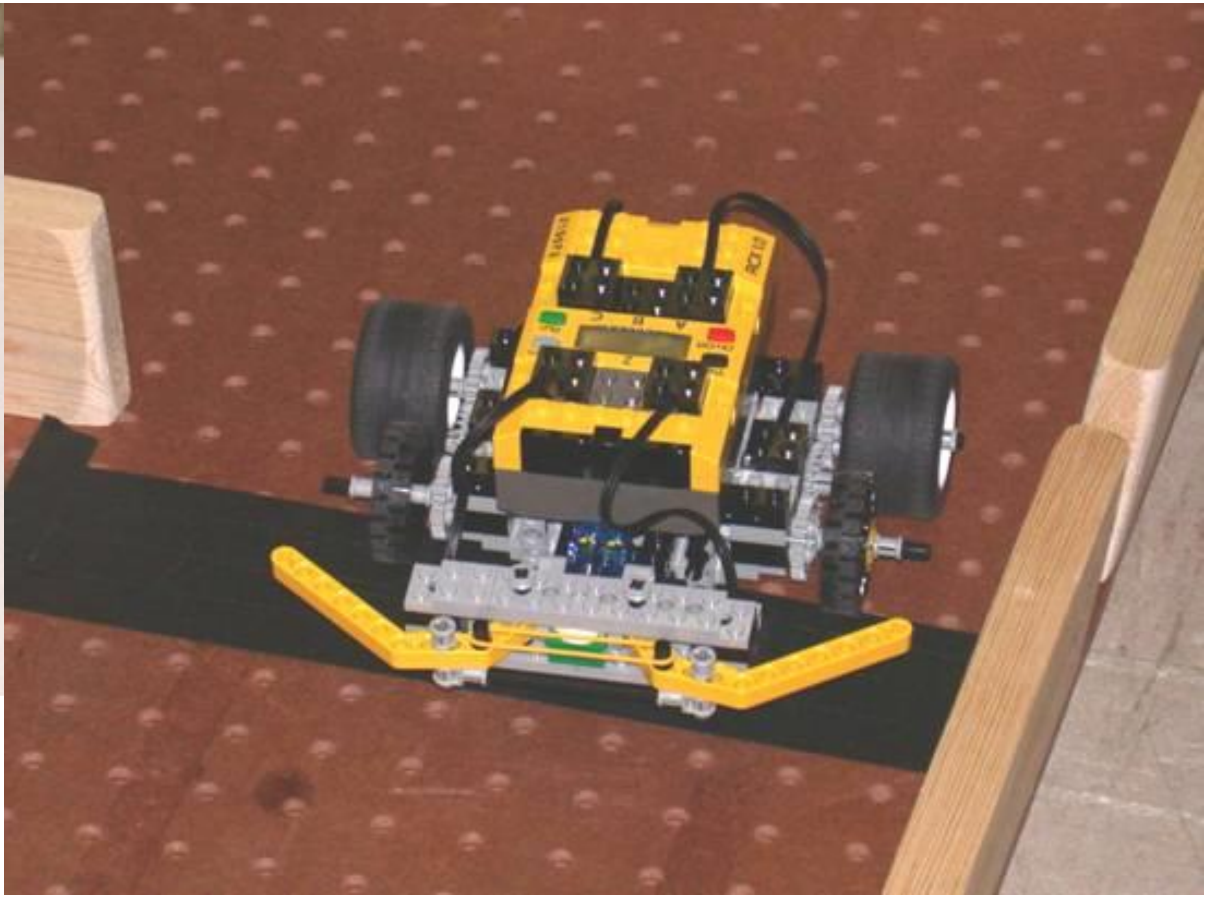



- purely cyclic application executes periodically.

# Example: Lego Mindstorm Robots



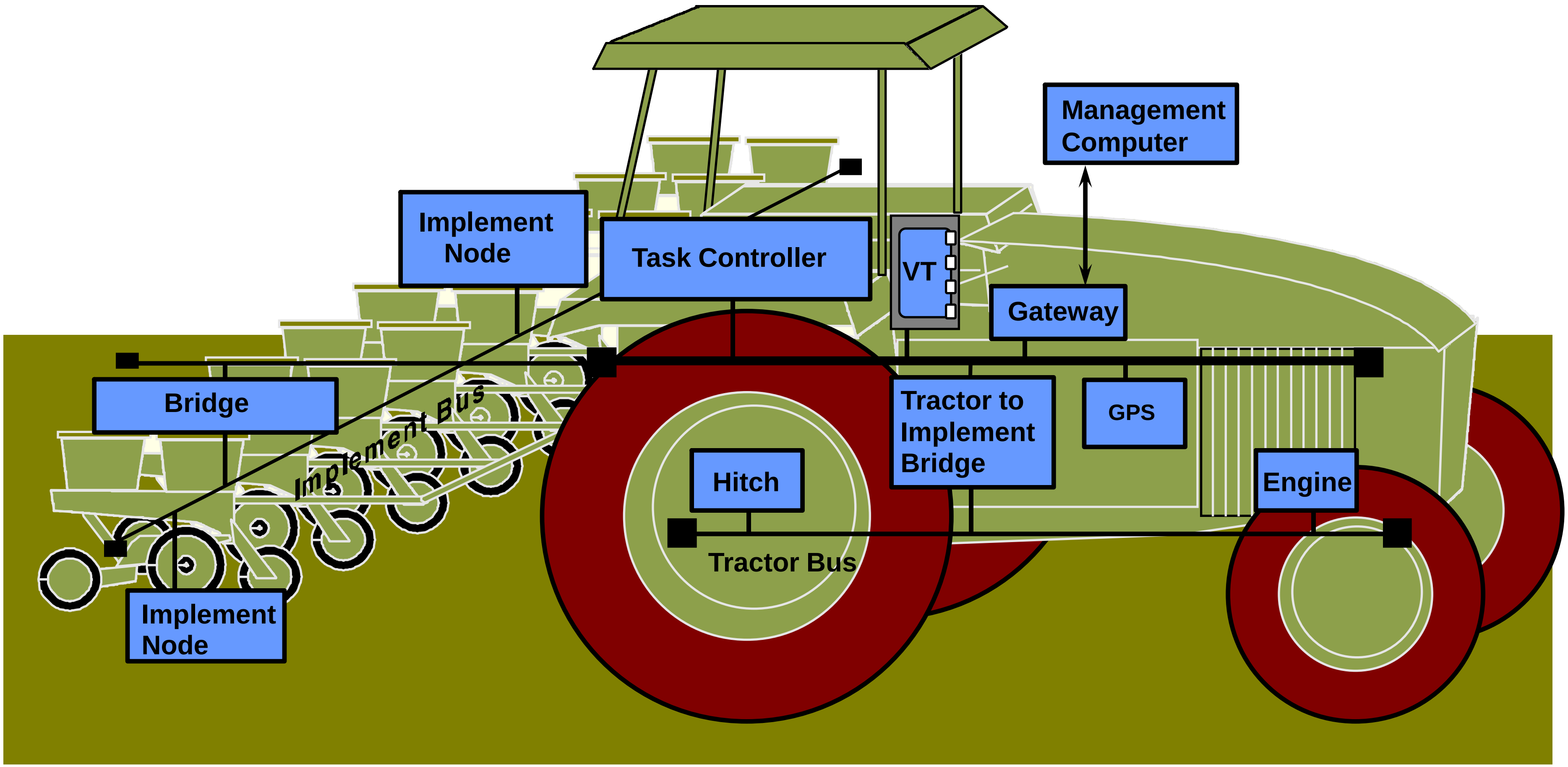
Light and touch sensors



# Variable-Rate Technology

Precision agricultural application

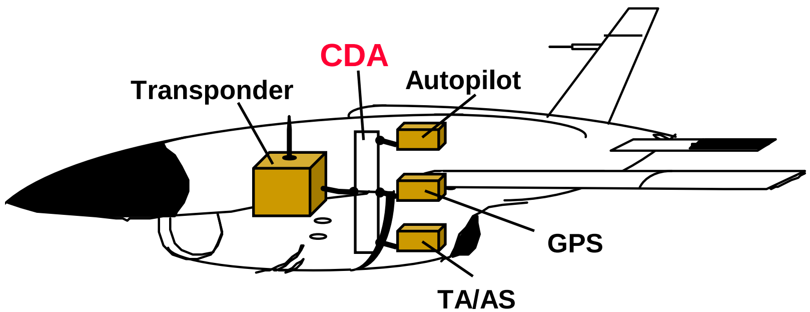
- mostly cyclic process control system.

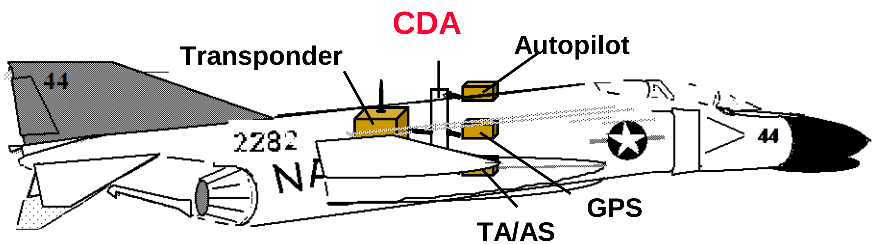


# Variable Rate Technology

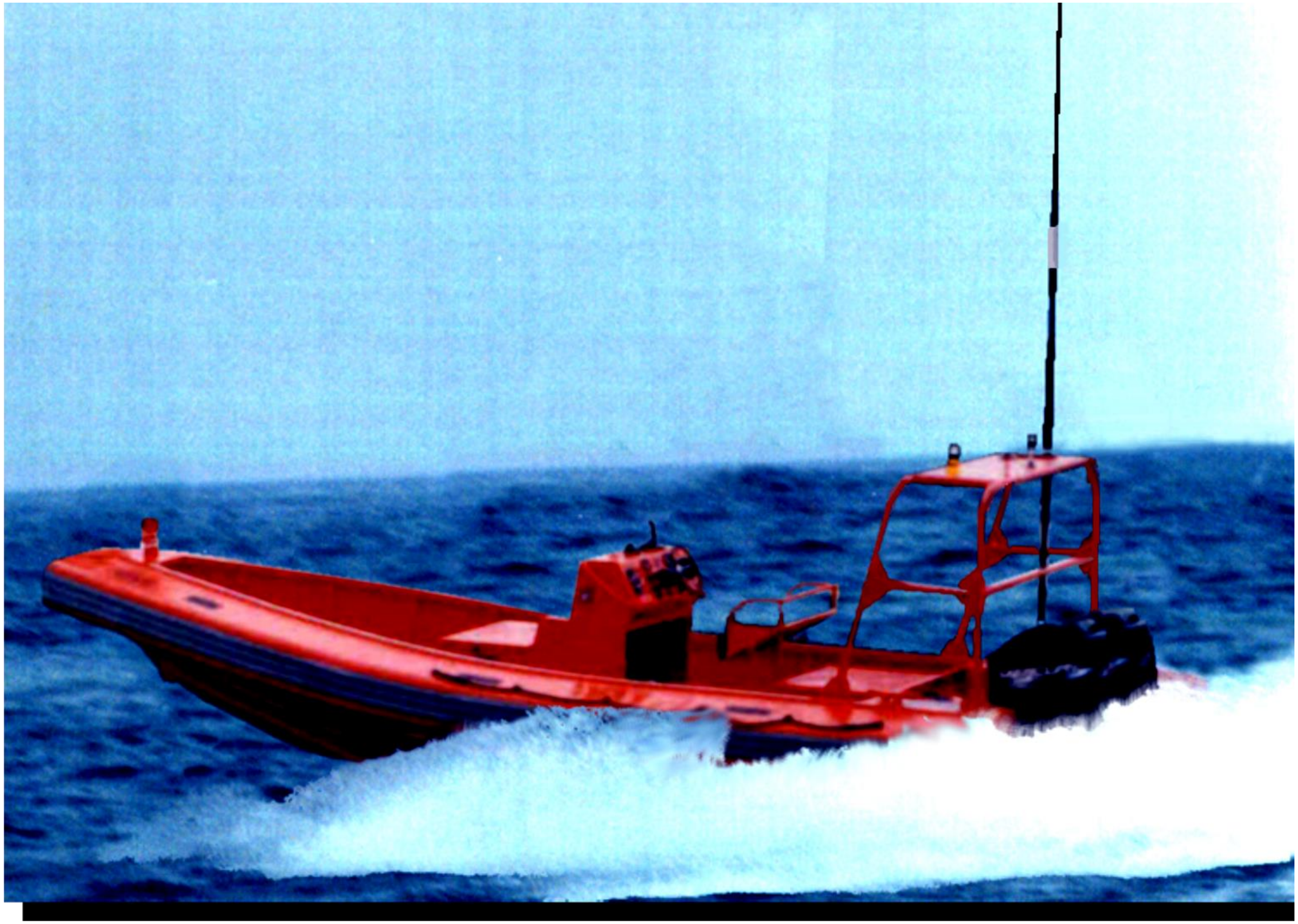
# Target Vehicle Electronics

- The **Common Digital Architecture (CDA 101)** is a standard architecture for interconnecting target vehicle electronics.

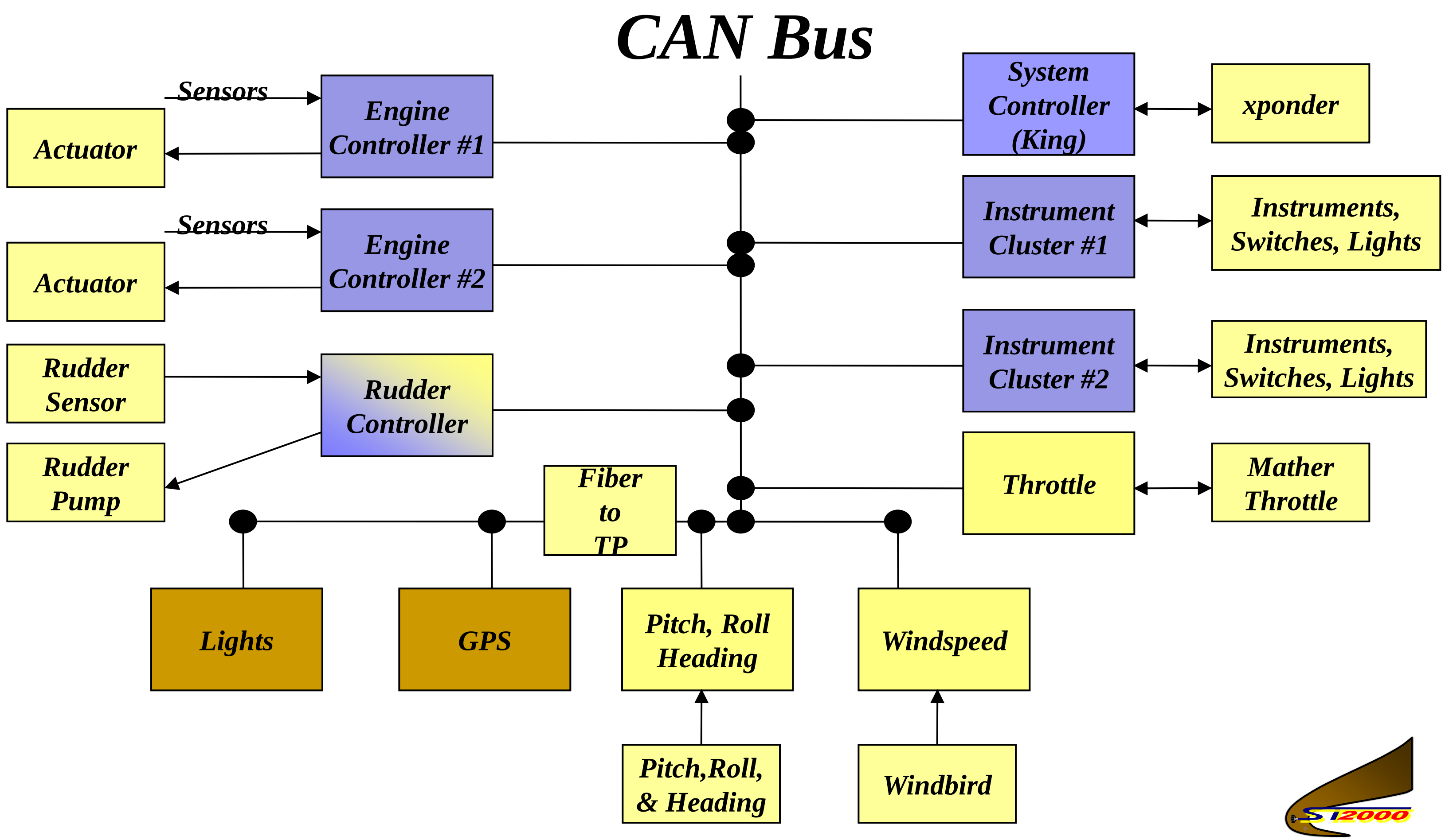


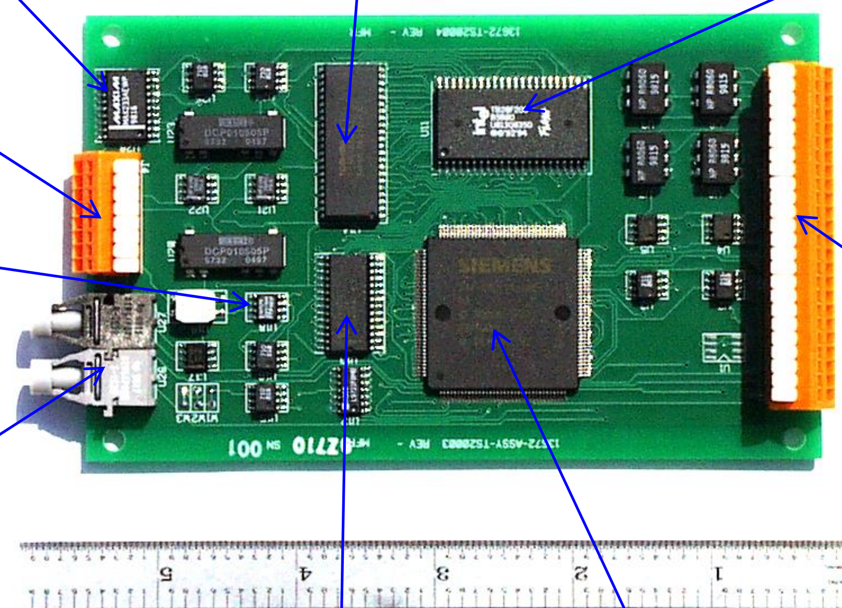
# Seaborne Target ST2000



# Seaborne Target ST2000



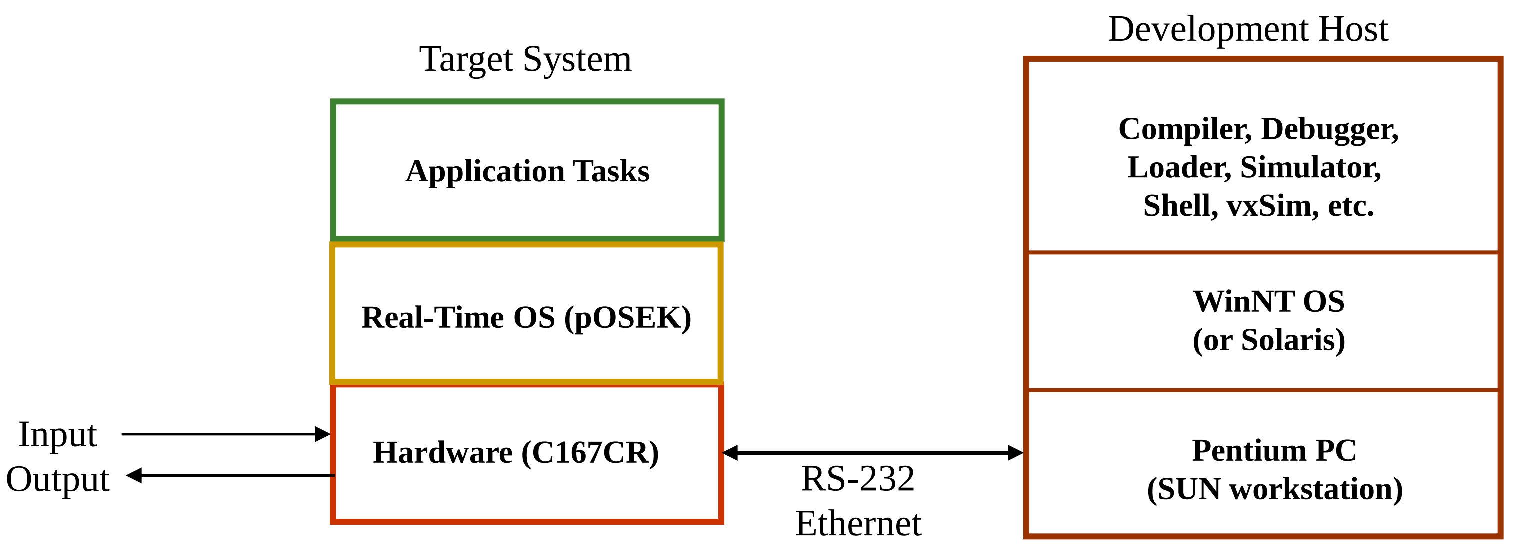
# Typical CAN Node



# For this class!

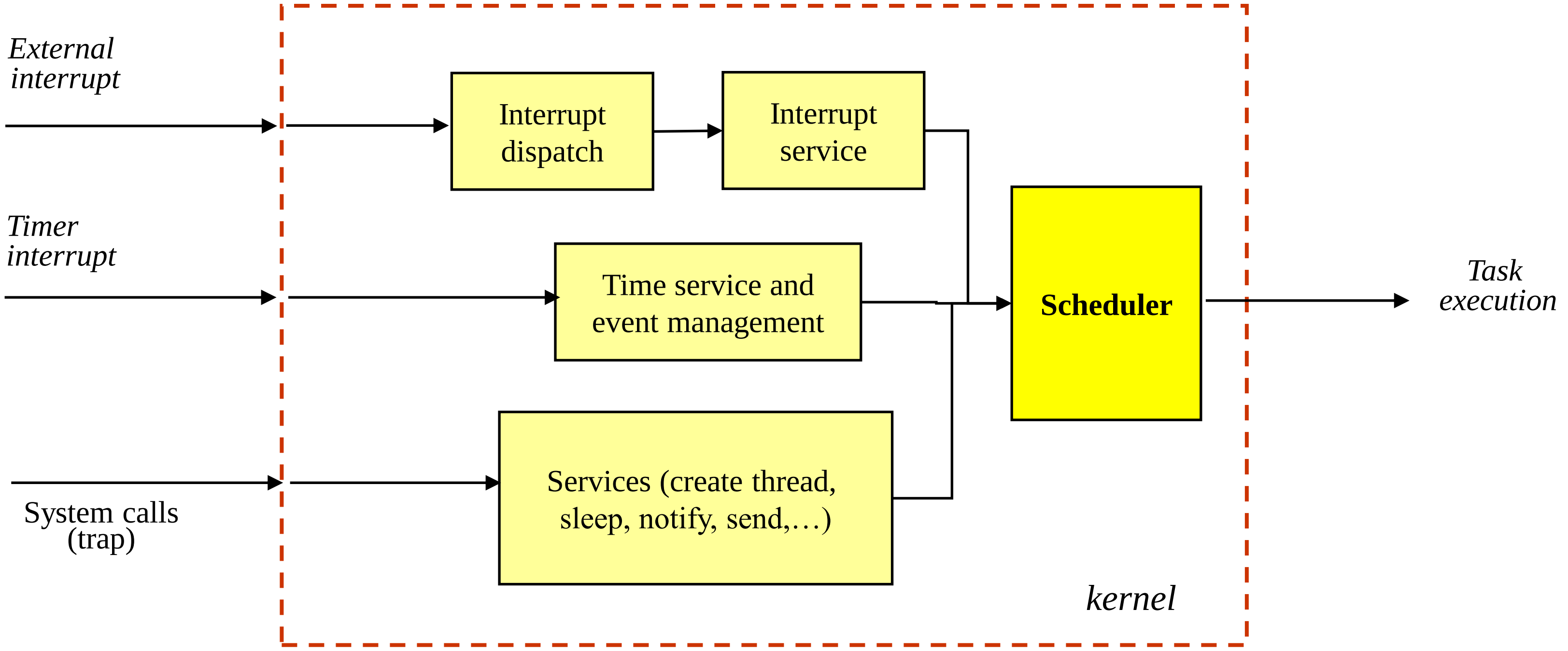
# Complex System Development

- High-Level Development Environment
- Real-Time Operating System



# Real-Time Operating System

**Functions:** task management, memory management, time management, device drivers, and interrupt service.



# Outline

- References - where to get more info.
- Typical Real-Time Applications
- **Terms and Concepts**

# Terms and Concepts

- A **real-time system** is a system with performance deadlines on computations and actions; that is, *system correctness depends not only on logical correctness, but also on the **timeliness** of the results*.
- An **embedded system** is a system that exists within a larger system.

# Characteristics of a real-time system

- Event-driven (reactive)/time-driven (periodic)
- High cost of failure
- Concurrency/multiprogramming
- Stand-alone/continuous operation
- Reliable/fault tolerant requirements
- **Predictable behavior**

# Misconceptions regarding real-time systems

- There is no science involved in designing real-time systems
- Advances in hardware will take care of real-time requirements
- Real-time computing is just fast computing
  - Only to ad agencies, for our purpose it means **PREDICTABLE** computing
- Real-time programming is assembly language coding
  - The current trend is to automate code generation

# Misconceptions regarding real-time systems

- “Real-time” is just performance engineering
  - In “real-time”, timeliness is generally more important than raw performance
- All of the important “real-time” problems have been solved
  - Many unsolved problems remain
- It is not meaningful to talk about real-time performance when things can fail
  - Designing the system to tolerate faults is usually an important criteria in real-time systems

# Definitions – used throughout the course

- A **job** is a unit of work that is scheduled and executed by the system ($J_{i,k}$).
- A **task** is a set of related jobs that provide some system function $\tau_i = \{ J_{i,1}, J_{i,2}, \ldots , J_{i,n} \}$.; e.g., the reception of a data frame could be a job that is part of a task that provides time service.
- The **deadline** of a job is the time at which a job must be completed.

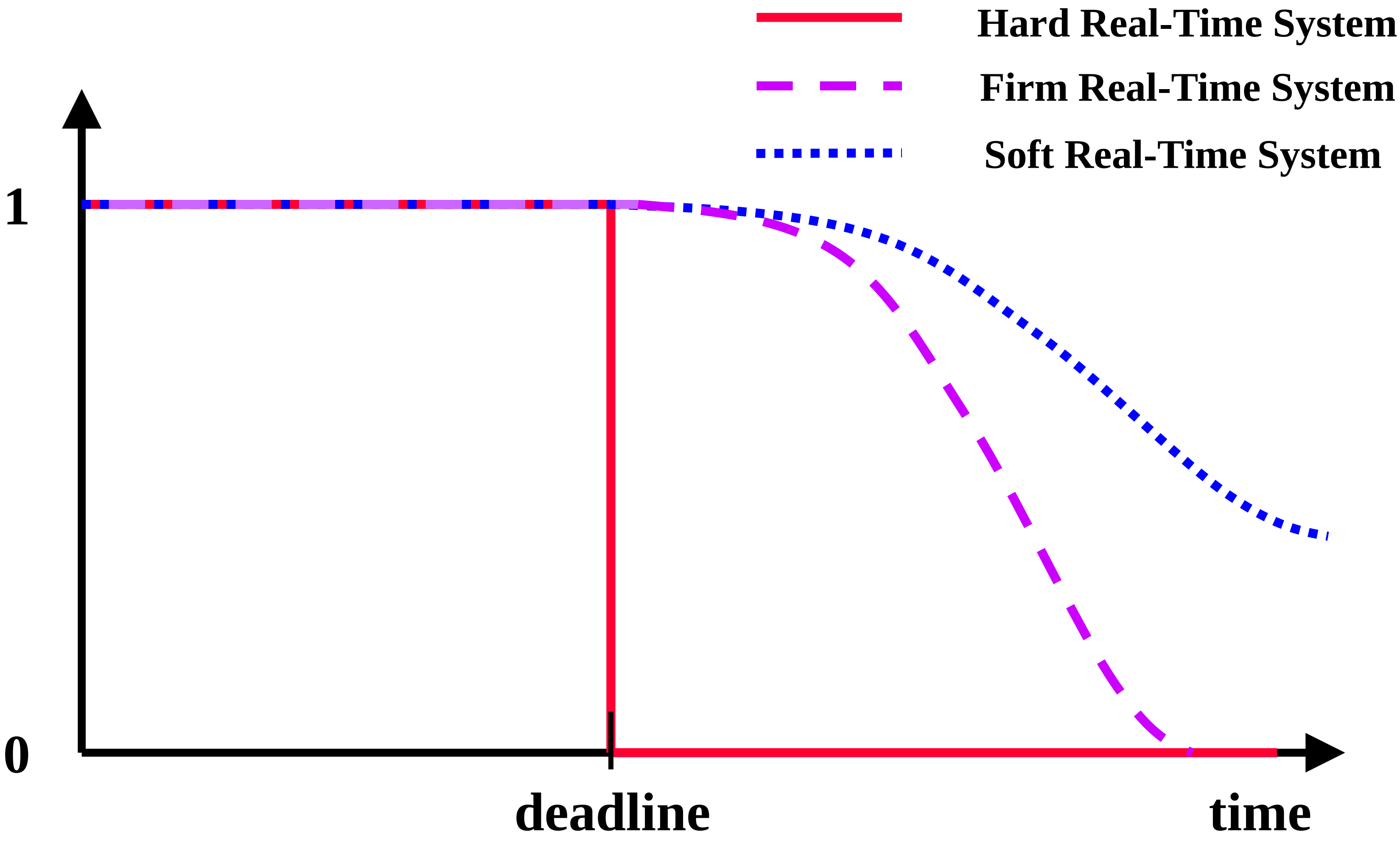
# Deadlines

- The **release time** (or **arrival time**) of a job is the time at which the job becomes available for execution ( $r_i$ or $R_i$ ).
- The **response time** of a job is the length of time between the release time of the job and the time instant when it completes.
- The **relative deadline** of a job is the maximum allowable response time of a job ( $D_i$ ).
- The **absolute deadline** of a job is the time at which a job must be completed ( $d_i = r_i + D_i$ ).

# Hard vs. Soft Constraints

- A timing constraint or deadline is **hard** if the failure to meet it is consider a fatal fault.
- The failure to meet a **soft** deadline is undesirable, but not fatal.
- Another way of defining hard and soft timing constraints is in terms of the value of the result (to the system) relative to time.

# Utility or "Usefulness" Function - Value of Result

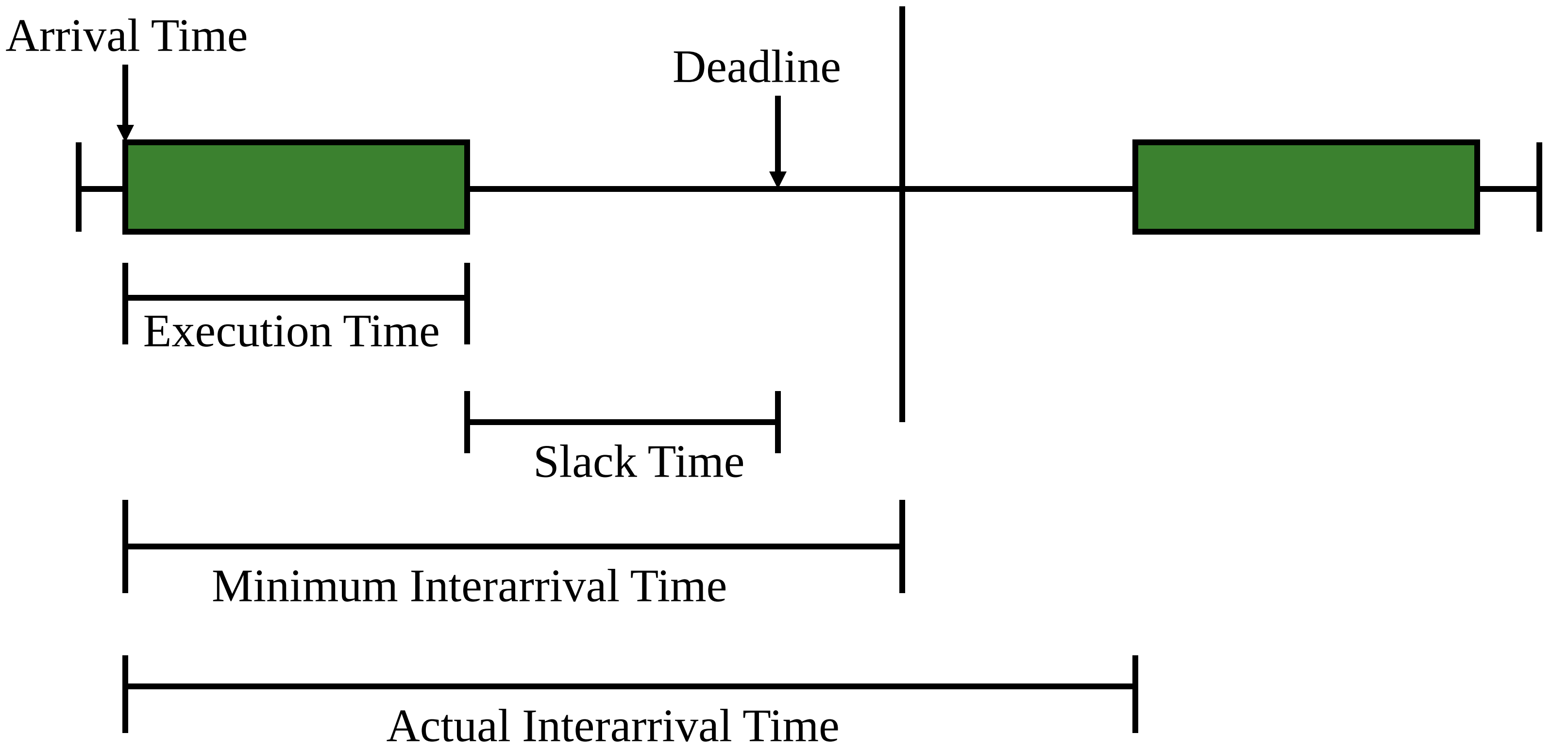


# Task Model

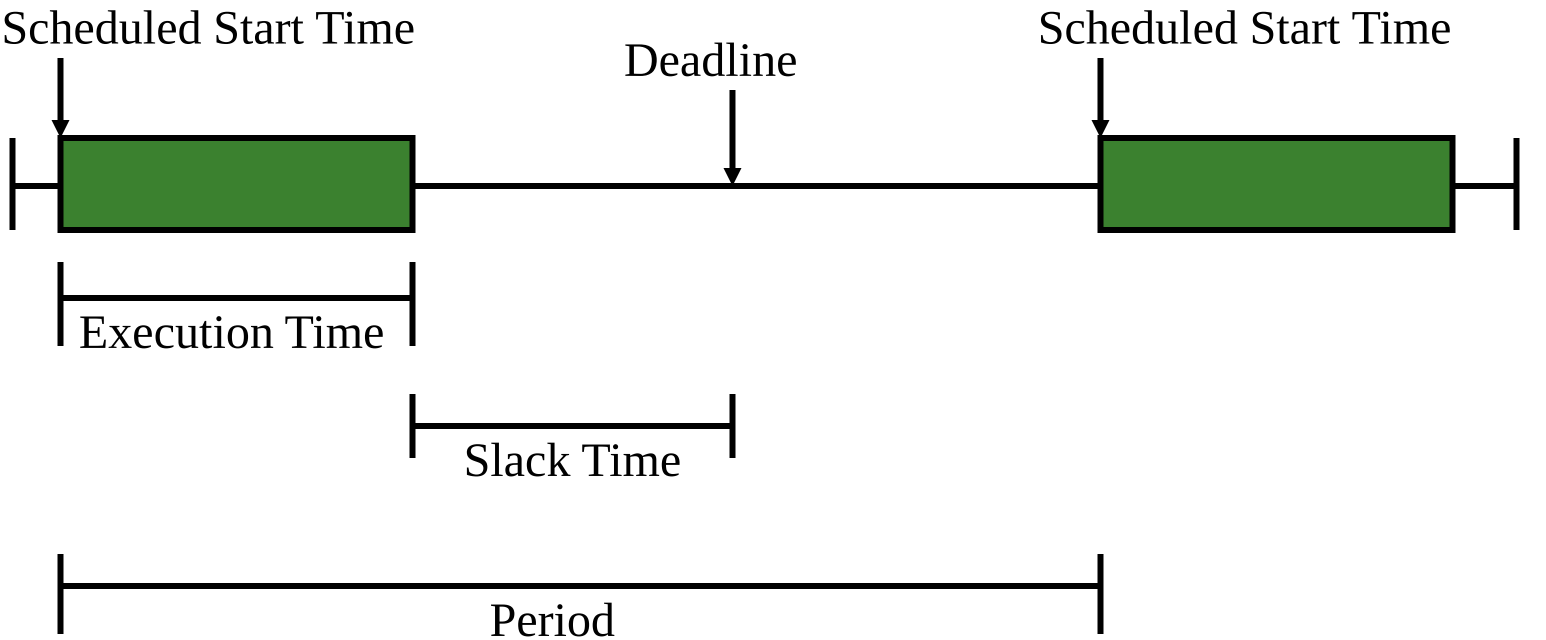
- **Event-Driven (Reactive) Tasks** primarily react to external events which are generally aperiodic (sporadic).

- **Time-Driven Tasks** are driven by the passage of time or time epochs; generally periodic tasks.

# Event-Driven Task



# Time-Driven (Periodic) Task



# Scheduler

- A **scheduler** assigns jobs to processors.
- A **schedule** is an assignment of all jobs in the system on available processors (produced by scheduler).
- The **execution time** (or **run-time**) of a job is the amount of time required to complete the execution of a job once it has been scheduled ( $e_i$ or $C_i$ ).
- A constraint imposed on the timing behavior of a job is called a **timing constraint**.

# Assumptions

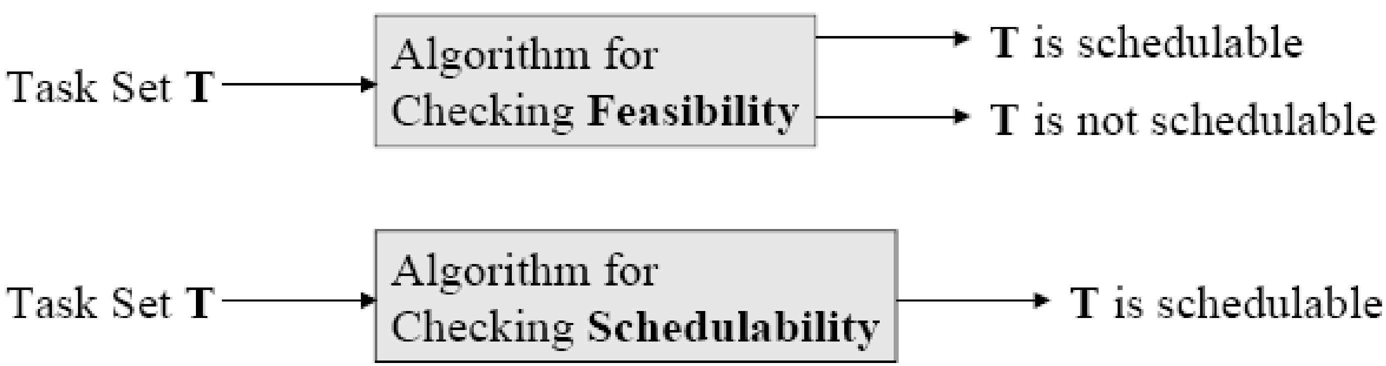
- The scheduler works **correctly**; e.g., it only produces **valid schedules** that satisfy the following conditions:
  - each processor is assigned to at most one job at a time,
  - each job is assigned to at most one processor,
  - no job is scheduled before its release time, and
  - all precedence constraints and resource usage constraints are satisfied.

# Feasible Schedule

- A valid schedule is a **feasible schedule** if every job meets its timing constraints; e.g., completes executing by its deadline.
- A set of jobs is **schedulable** according to a scheduling algorithm if (when) using the algorithm (the scheduler) always produces a feasible schedule.
- The **lateness** of a job is the difference between its completion time and its deadline. If the job completes early, its lateness will be negative.

# Feasibility vs. Schedulability

- Most people in the real-time research community use "**feasibility**" to refer to an **exact schedulability test**, whereas "**schedulability**" to refer to a **sufficient test**.
- However, as we shall see, the terms are used inconsistently and interchangeably depending on the author.



# Timing Constraints

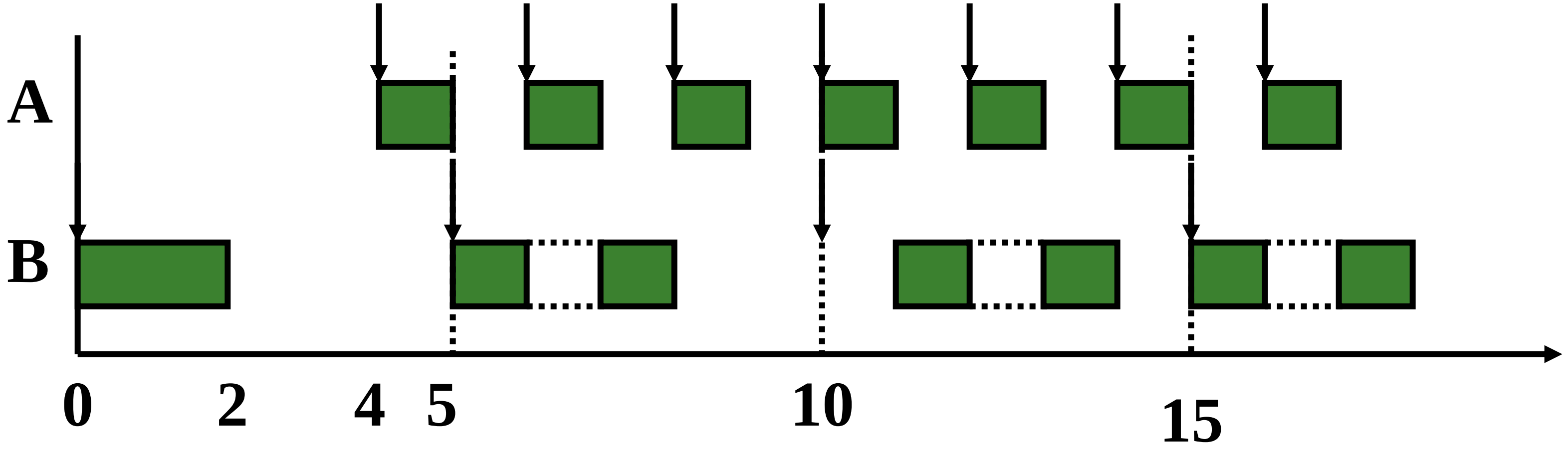
- **Periodic** - tasks arrive at **fixed intervals**, called **periods**. Note that the author of our text uses a slightly more general definition for periodic, but we will stick with this more commonly adopted definition.
- **Aperiodic (Sporadic)** - tasks may arrive at any time after a minimum interval.
  - Sporadic = hard deadline
  - Aperiodic = soft deadline or no deadline

# Periodic Task

- A periodic task $\tau_i$ = { $J_{i,1}$, $J_{i,2}$, ... , $J_{i,n}$ } is a sequence of jobs with identical parameters with:
  - a **period** ( $p_i$ or $T_i$ ) equal to the **exact** (the text uses "minimum", and here's where we differ from the text) length of time between the release times of consecutive jobs,
  - an **execution time** ( $e_i$ or $C_i$ ) equal to the maximum execution time of any job in the task, and
  - a **phase** ( $\varphi_i$ ) equal to the release time of the first job in $\tau_i$ .

# Periodic Task Set - Priority-Driven Scheduler

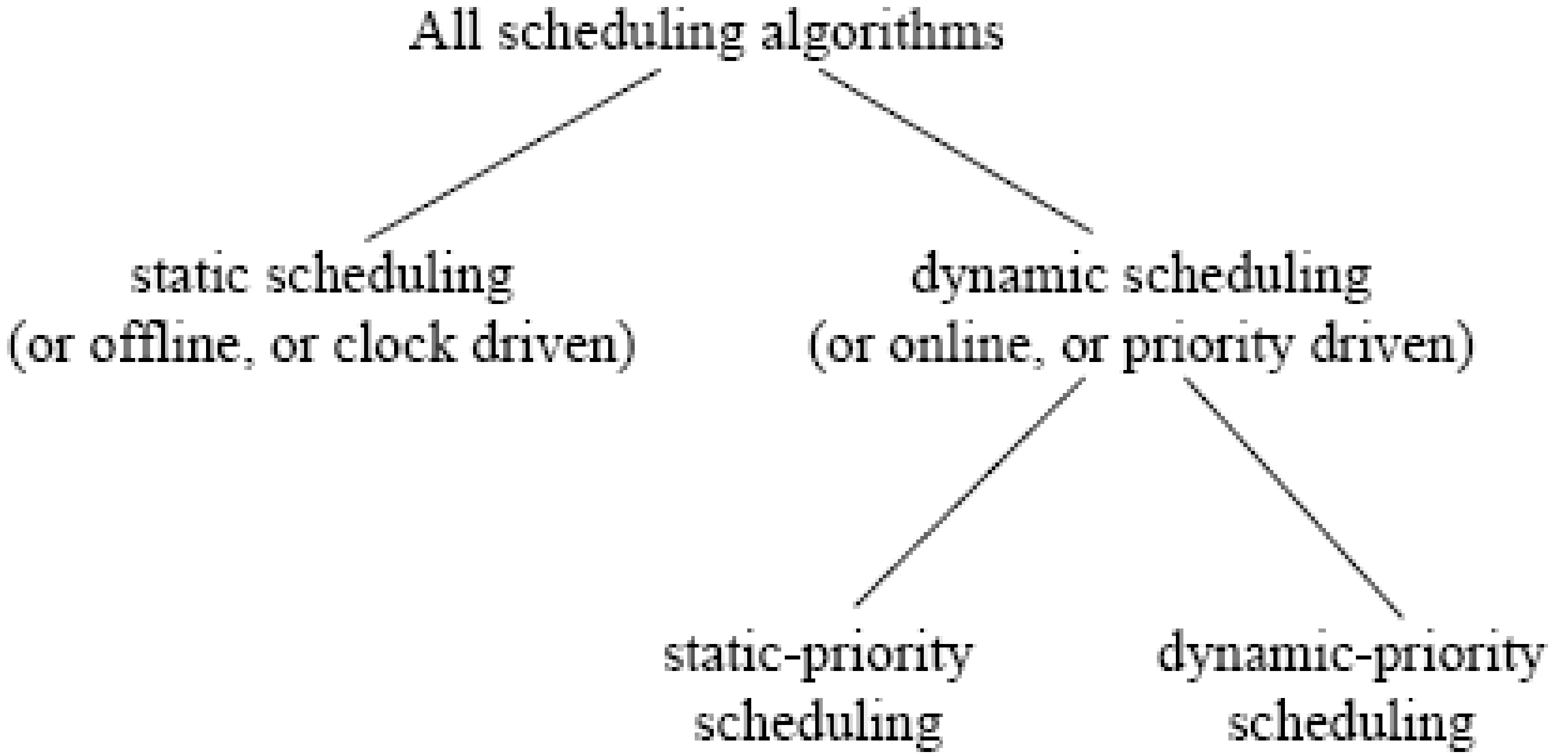
| Task $\tau_i$ | | Period $T_i$ | Deadline $D_i$ | Run-Time $C_i$ | Phase $\varphi_i$ |
|---|---|---|---|---|---|
| A | (High Priority) | 2 | 2 | 1 | 4 |
| B | (Low Priority) | 5 | 5 | 2 | 0 |



# Algorithms

- We’re interested in two types of algorithms:
  - **Scheduling algorithms** are used to generate to a schedule or priority assignment that can be used by a scheduler to schedule tasks at run time.
  - **Feasibility/schedulability analysis algorithms** are used to determine if a given task set is schedulable.
  - Normally, the second class of algorithms are much more complex than the first class.

# Classification of Scheduling Algorithms



# Static Scheduling Algorithms

- **Static scheduling algorithms** can be used if the scheduling algorithm has complete knowledge of the task set and all timing constraints such as deadlines, execution times, precedence, and **future** arrival times.
- The static algorithm operates on the set of tasks and constraints to generate a single, fixed schedule.

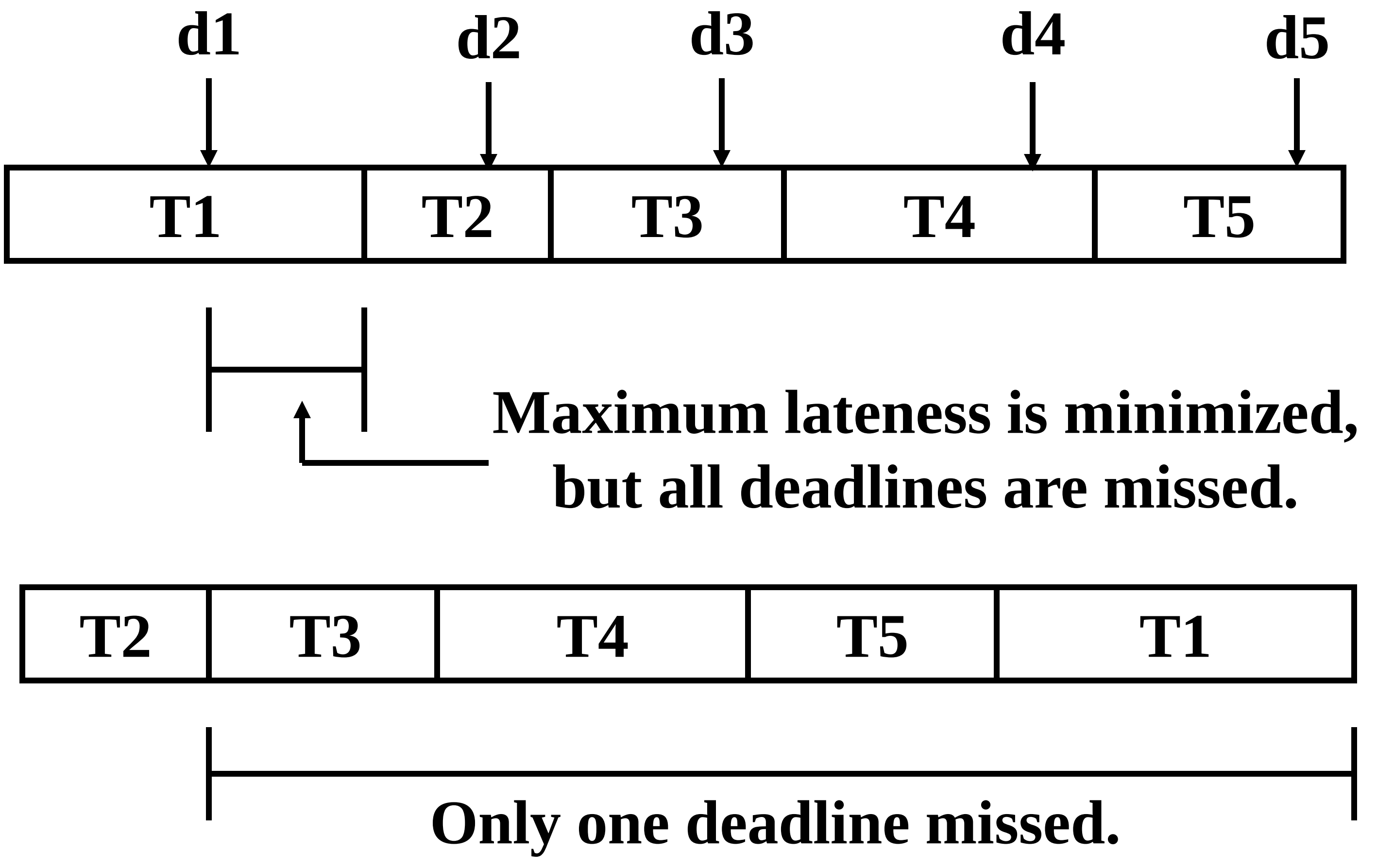
# Dynamic Scheduling Algorithms

- **Dynamic scheduling algorithms** have complete knowledge of **currently active** jobs, but new jobs may arrive at any time in the future.
- Dynamic scheduling is performed at run-time (online); however, offline analysis is usually performed to constrain the dynamic schedule; e.g., assign priorities, etc.

# Metrics used to evaluate scheduling algorithms

- processor utilization
- throughput
- weighted sum of task completion times
- schedule length
- number of processors required
- maximum lateness
- **missed deadlines**

# Minimize maximum lateness



# Missed Deadlines

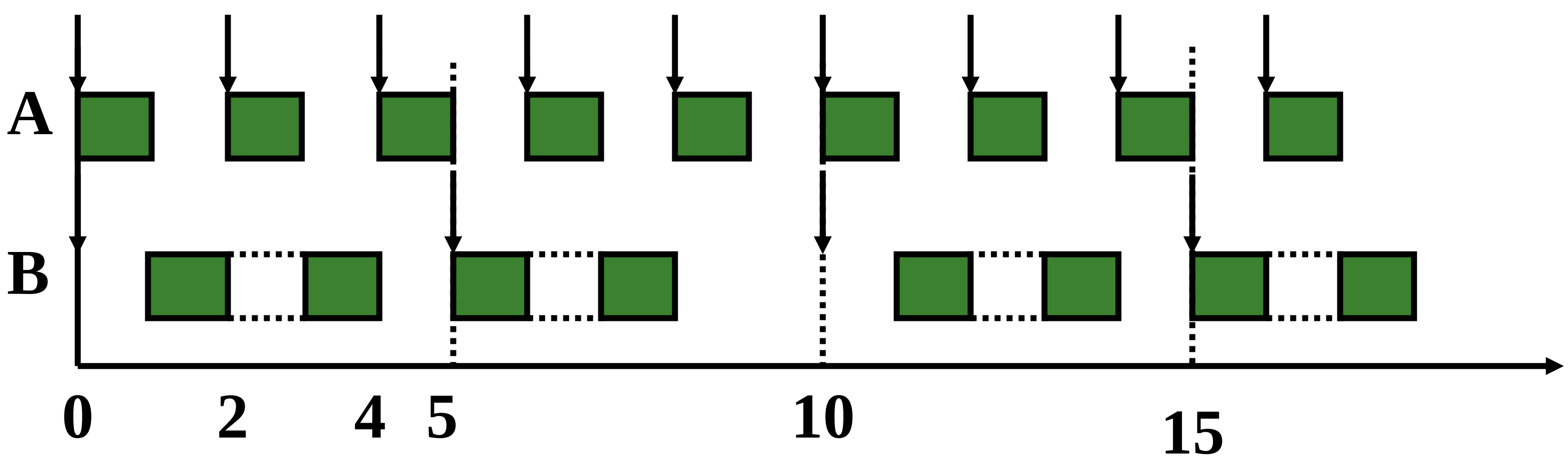
- Much real-time work is only concerned with missed deadlines; e.g., for **hard real-time systems all deadlines must be met**.
- In which case, an **optimal** scheduling algorithm is one that will fail to meet a deadline for any given task set only if no other scheduling algorithm can meet the deadlines.

# Periodic Task

- A periodic task $\boldsymbol{\tau_i}$ **= {** $\mathbf{J_{i,1}, J_{i,2}, \ldots , J_{i,n}}$ **}** is a sequence of jobs with identical parameters with:
  - a **period** ( $p_i$ or $T_i$ ) equal to the length of time between the release times of consecutive jobs,
  - an **execution time** ( $e_i$ or $C_i$ ) equal to the maximum execution time of any job in the task, and
  - a **phase** ( $\varphi_i$ ) equal to the release time of the first job in $\boldsymbol{\tau_i}$ .

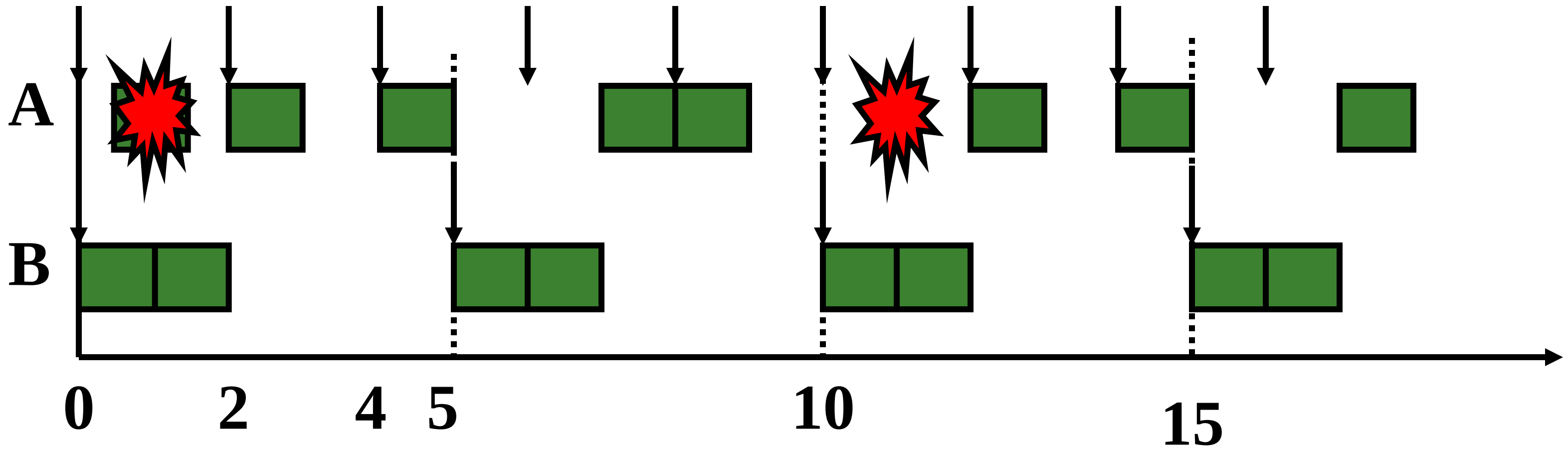
# Example #1 - Priority-Driven Scheduler

| Task $\tau_i$ | | Period $T_i$ | Deadline $D_i$ | Run-Time $C_i$ | Phase $\varphi_i$ |
|---|---|---|---|---|---|
| A | (High Priority) | 2 | 2 | 1 | 0 |
| B | (Low Priority) | 5 | 5 | 2 | 0 |



# Example #2

| Task $\tau_i$ | | Period $T_i$ | Deadline $D_i$ | Run-Time $C_i$ | Phase $\varphi_i$ |
|---|---|---|---|---|---|
| A | (Low Priority) | 2 | 2 | 1 | 0 |
| B | (High Priority) | 5 | 5 | 2 | 0 |



# Example #3

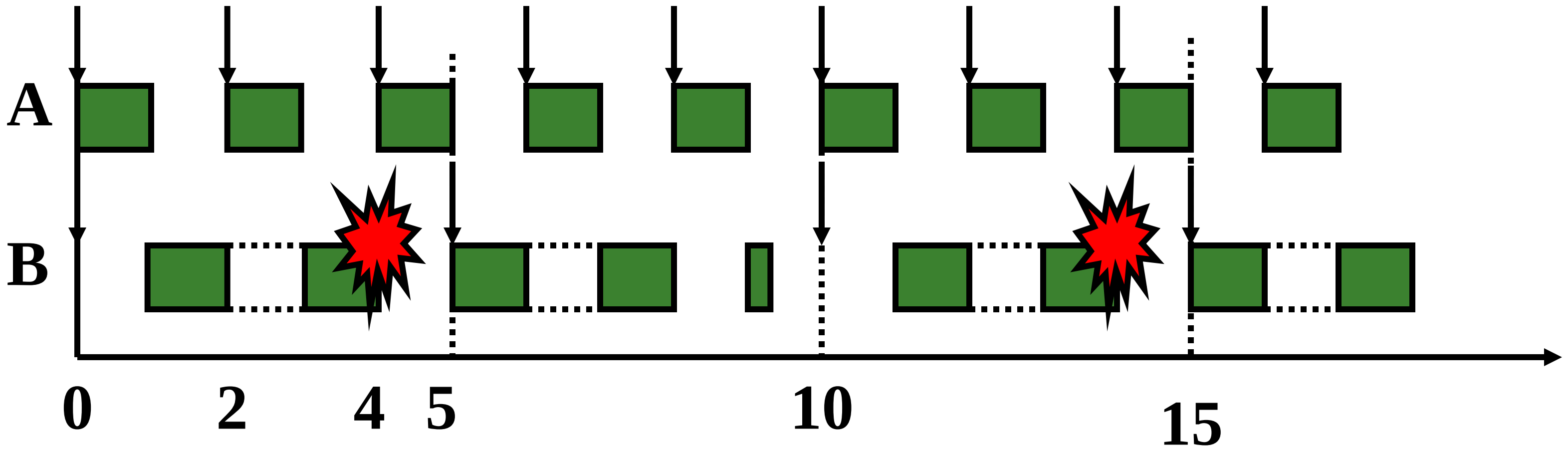
| Task $\tau_i$ | | Period $T_i$ | Deadline $D_i$ | Run-Time $C_i$ | Phase $\varphi_i$ |
|---|---|---|---|---|---|
| A | (High Priority) | 2 | 2 | 1 | 0 |
| B | (Low Priority) | 5 | 5 | 2.1 | 0 |

$U = C_1 / T_1 + C_2 / T_2 = 1 / 2 + 2.1 / 5 = 0.92$

Even if U < 1, a task set **may not be schedulable** using fixed priority scheduling.

# Example #3

| Task $\tau_i$ | | Period $T_i$ | Deadline $D_i$ | Run-Time $C_i$ | Phase $\varphi_i$ |
|---|---|---|---|---|---|
| A | (High Priority) | 2 | 2 | 1 | 0 |
| B | (Low Priority) | 5 | 5 | 2.1 | 0 |



# Observations

- The schedulability of a task set depends on priority assignment (Example 1 is schedulable, but Example 2 is not).
- Even if the utilization of a task set is less than one, it may not be schedulable by any fixed priority assignment (Example 3 is not).

# Priority-Driven Scheduling Algorithms for Periodic Tasks

- **Fixed-Priority** - assigns the same priority to all jobs in a task.
- **Dynamic-Priority** - assigns different priorities to the individual jobs in each task.

- After looking at Static Scheduling Algorithms, we will investigate Dynamic Scheduling Algorithms first by considering **fixed-priority algorithms**.

# Issues in Fixed Priority Assignment

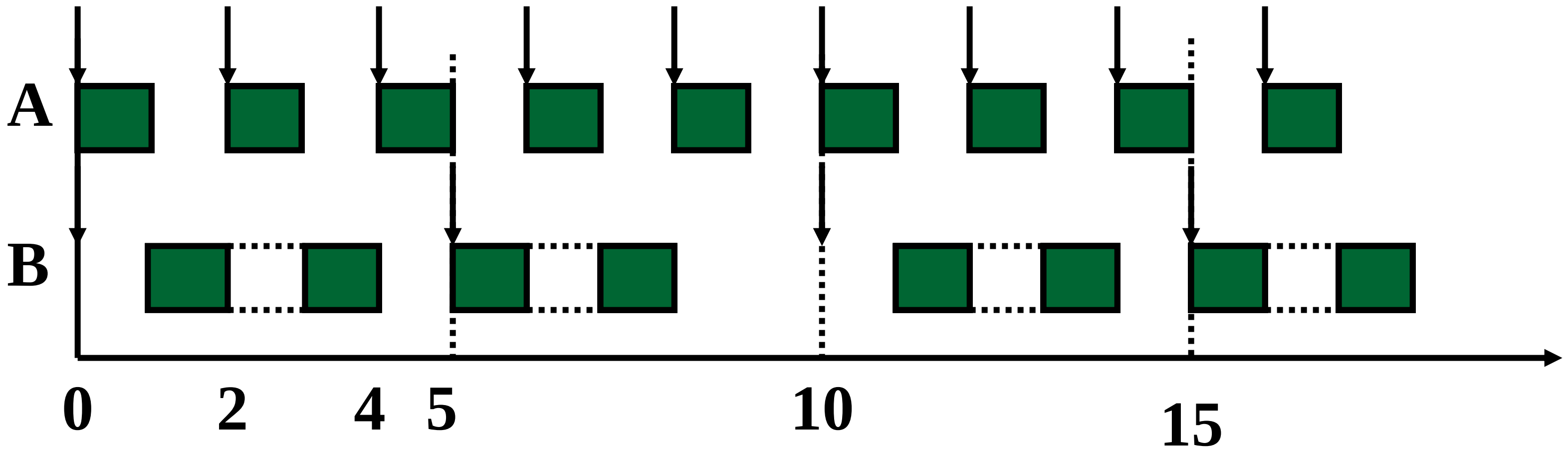
- How to assign priorities?
- How to determine which assignment is the best; e.g., how to evaluate a priority assignment algorithm (method)?
- How to compare different priority assignment algorithms?

# Rate-Monotonic Algorithm (RM)

- The **rate** of a task is the inverse of its period.
- Task with **higher rates** are assigned **higher priorities**.
- C. L. Liu and J. W. Layland, “Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment”, JACM, Vol. 20, No. 1, pages 46-61, 1973.

# Example #1 - Rate Monotonic Assignment

| Task $\tau_i$ | | Period $T_i$ | Deadline $D_i$ | Run-Time $C_i$ | Phase $\varphi_i$ |
|---|---|---|---|---|---|
| A | (High Priority) | 2 | 2 | 1 | 0 |
| B | (Low Priority) | 5 | 5 | 2 | 0 |



# Real-Time Reference Model (Ch. 3)

- Idea: Abstract away functional characteristics and focus on **timing requirements and resource requirements**.
- **Reference Model Components**
  - **Resource Graph** – identify available system resources, resource types, and dependencies
  - **Task Graph** – identify task dependencies
  - **Scheduling and Resource Management** – identify algorthithms for scheduling and resource management

# Processors and Resources

- **Processors ($P_i$)** are **active** system resources, such as computers, transmission (tx) links, and database servers
- **Resources ($R_i$)** are **passive** system resources, such as memory, mutexes, semaphores, and database locks
- **Example: Sliding Window Protocol**
  - Job = transmit a message
  - Processor = data link
  - Resource = valid sequence number

# Types of Resources

- **Reusable** – most resources are reusable; e.g., they can be reused by subsequent jobs after being released.
  - Ex: a mutex is a serially reusable resource
- **Plentiful** – a resource is plentiful if no job is ever prevented from executing due to a lack of this resource.
  - Ex: a read-only (immutable) configuration file
  - Plentiful resources are typically removed from the model.

# Resource or Processor?

- For some problems, it is hard to classify system resources as processors or resources.
- This is where experience and the “art” of modeling comes in to play.
- **Example: I/O Bus**
  - In many cases the I/O Bus is viewed as a plentiful resource and ignored in the model
  - However, if we want to study how I/O activities impact real-time performance of an I/O arbitration scheme, then the bus must be modeled as a resource or processor.

# Temporal Parameters

- $J_i$ : **job** – a unit of work
- $T_i$ (or $\tau_i$ ): **task** - a set of related jobs
- A **periodic task** is sequence of invocations of jobs with identical parameters.
- $r_i$: **release time** of job $J_i$
- $d_i$: **absolute deadline** of job $J_i$
- $D_i$: **relative deadline** (or just **deadline**) of job $J_i$
- $e_i$: (Maximum) **execution time** of job $J_i$

# Periodic Task Model

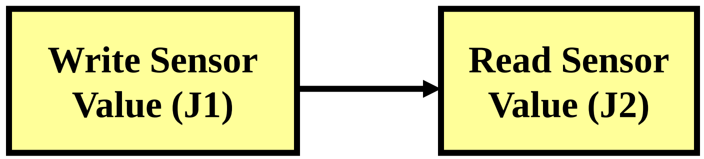
- **Tasks:** $T_1, \ldots.., T_n$
- Each consists of a set of **jobs**: $T_i = \{J_{i1}, J_{i2}, \ldots\}$
- $\varphi_i$ : p**hase** of task $T_i$ = time when its first job is released
- $p_i$: p**eriod** of $T_i$ = inter-release time
- H: h**yperperiod** $H = \mathrm{lcm}(p_1, \ldots.., p_n)$
- $e_i$: e**xecution time** of $T_i$
- $u_i$: **utilization** of task $T_i$ is given by $u_i = e_i / p_i$
- $D_i$: (relative) **deadline** of $T_i$, typically $D_i = p_i$

# Types of Release Times

- **Fixed** – release times are known values (**periodic**)
- **Jittered:** $r_i \in [r_i^-, r_i^+]$ : release time of job $J_i$ falls within a known interval
- **Sporadic or aperiodic** – release times are unknown
  - A(x) = interarrival time (time between two consecutive jobs) probability distribution
  - B(x) = execution time distribution
- Definitions
  - **Sporadic tasks** have jobs with hard relative deadlines, but **aperiodic tasks** have either soft or no deadlines

# Precedence Constraints/Graphs

- A **precedence graph** reflects data/control dependencies
- **Example: Sensor/actuator (producer/consumer)**

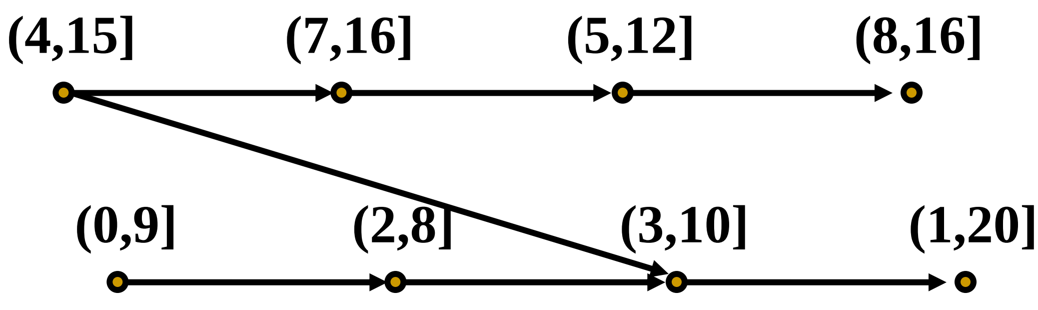



**J1 < J2**

- A precedence relation, denoted < , defines a partial order on the set of jobs.
- $J_i < J_k$ if $J_i$ is a predecesor of $J_k$
- Precedence graph: $G = (\mathbf{J}, <)$, $\mathbf{J} = \{J_1, J_2, \ldots\}$
- Precedence constraints can include AND/OR constraints.
- Some dependencies **cannot** be captured by task graphs
  - Example: access to shared data

# Task Graph

- A **task graph** is an extended precedence graph:
  - Vertices denote jobs
  - Edges denote dependencies
  - The label in brackets above each job give its feasible interval ( $r_i$, $d_i$ ] = ( release time, absolute deadline ].



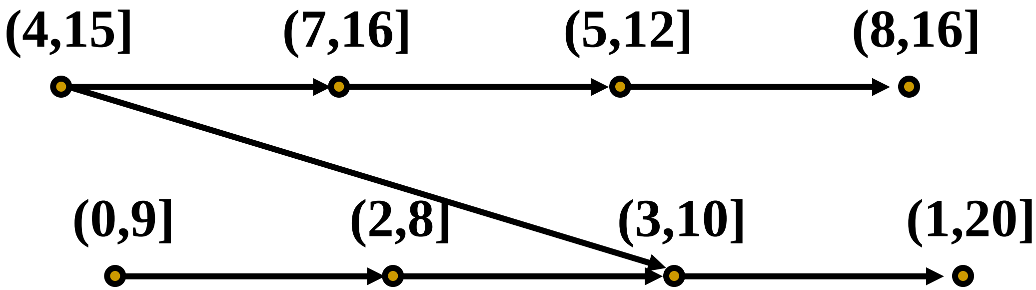
# Effective Timing Constraints

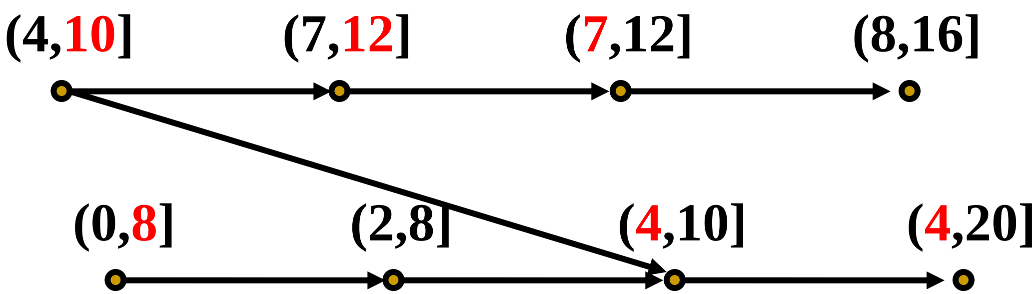
- Timing constraints are often inconsistent with precedence constraints; e.g., $d_1 > d_2$ , but $J_1 < J_2$
- Effective timing constraints on a single processor:
  - **Effective release time**:

    $r_i^{eff} = \mathbf{max}( r_i , \{ r_k^{eff} \mid J_k < J_i \} )$
  - **Effective deadline**:

    $d_i^{eff} = \mathbf{min}( d_i , \{ d_k^{eff} \mid J_i < J_k \} )$
- **Theorem:** A set of jobs **J** can be feasibly scheduled on a processor iff it can be feasibly scheduled to meet all **effective** release times and all **effective** deadlines.

# Effective Release Times and Deadlines



# Note

- Unless otherwise specified, we will use the terms release time and effective release time interchangeably; likewise, we will use the terms deadline and effective deadline interchangeably.

# System Characterization

- **Preemptivity** - are the jobs preemptable; e.g., can the current task be suspended to assign the processor to a more urgent task?
- **Context-switching time** - is the time required to switch between tasks negligible?
- **Laxity type** - are deadlines hard or soft?
- **Resource requirements** - are any resources required by the job to execute, and for what time interval are these resources required (e.g., critical sections).
- **Criticalness –** can jobs be assigned weights to indicate their importance relative to other jobs? If so, algorithms can be used to optimize weighted performance metrics.

# Schedules

- A **schedule** is an assignment of jobs to available processors. In a **feasible schedule**, every job starts at or after its release time and completes by its deadline.
- In a hard real-time system, a scheduling algorithm is **optimal** if it always produces a feasible schedule if such a schedule exists.
- In a soft real-time system, we can consider different **performance metrics:**
  - Number of **missed deadlines** (tardy jobs).
  - Maximum (or average) **tardiness or lateness**.
  - Maximum (or average) **response time.**

# Common Approaches For Real-Time Scheduling

- **Clock-Driven (Time-Driven) Approach –** scheduling decisions are made at specific time instants.
- **Weighted Round-Robin Approach -** every job joins a FIFO queue; when a job reaches the front of the queue, its weight refers to the fraction of processor time (number of time slices) allocated to the job.
- **Priority-Driven (Event-Driven) Approach -** ready jobs with highest priorities are scheduled for execution first.
  - Scheduling decisions are made when particular events occur; e.g., a job is released or a processor becomes idle. A **work-conserving** processor is busy whenever there is work to be done.

# For Next Time

- Read Ch. 1-5.
- Static Cyclic Scheduling (Ch. 5)
- After that, Real-Time Scheduling – Commonly Used Approaches (Ch. 4)