Embedded Systems - Real-Time Scheduling

part 1

Timing constraints: origin and characterization

Basic concepts of *real-time*
Real-Time Systems requirements

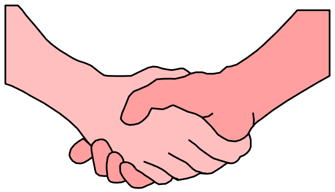
Real-Time Computing

The **results of the computations** must be

- Logicaly correct
- Produced in time

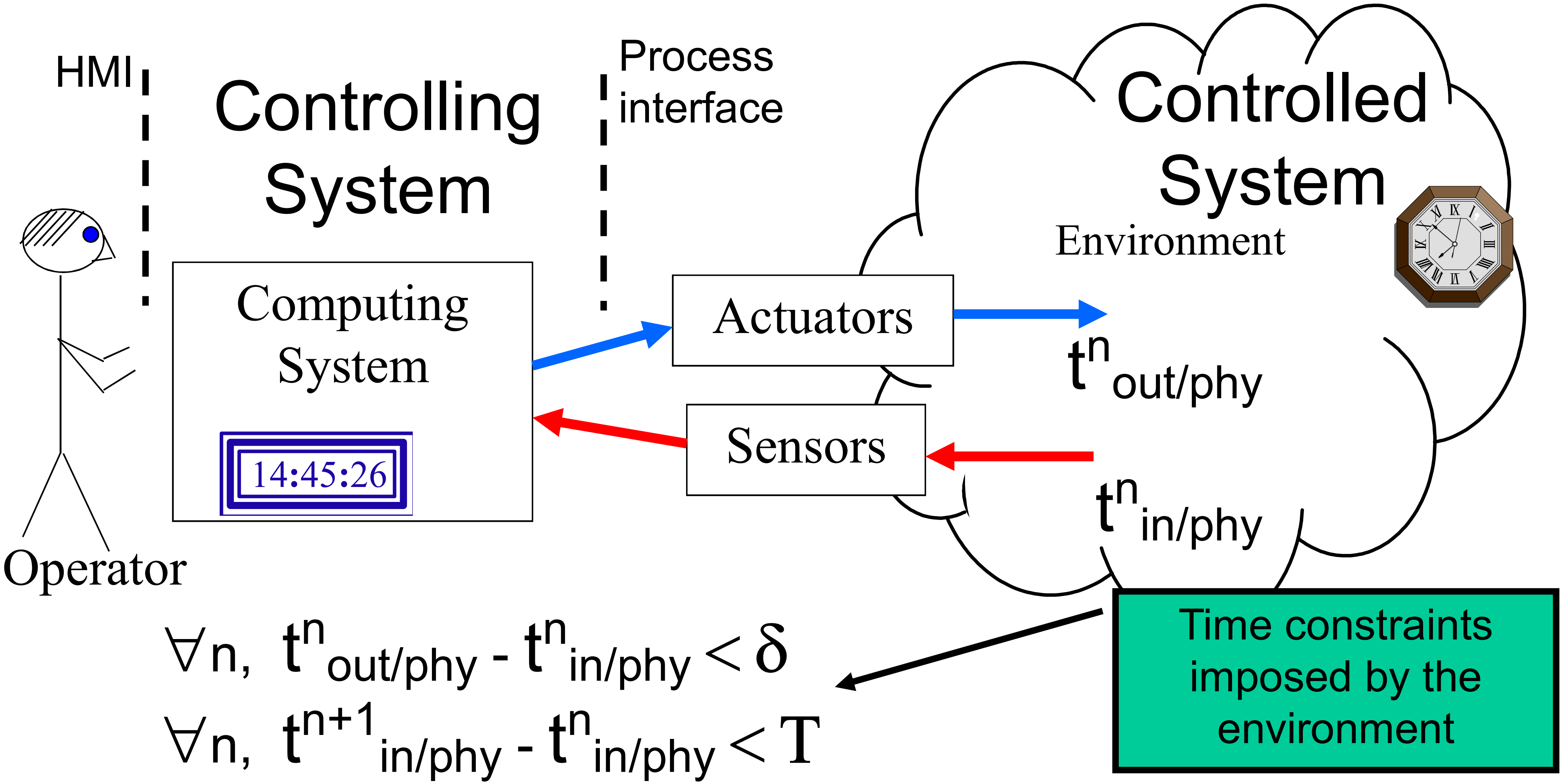
(Stankovic, 1988)

Timeliness



Logical correctness

Real-Time System



$$\forall n,\ t^n{}_{out/phy} - t^n{}_{in/phy} < \delta$$

$$\forall n,\ t^{n+1}{}_{in/phy} - t^n{}_{in/phy} < T$$

Purpose of studying Real-Time Systems

- The main **objective** of a Real-Time Systems (RTS) discipline is to develop **appropriate techniques** for the
  - **Design / synthesis,**
  - **Analysis and**
  - **Verification**

  of such systems that allow obtaining **guarantees** that their **temporal behavior is adequate to the dynamics** of the process they must interact with.

Special features of Real-Time Systems

Given their **dependence with time**, some of the most important aspects that characterize a real-time system are:

- **Execution time of its computations**
- **Response time to events**
- **Regularity of generating periodic events**

Special features of Real-Time Systems

These aspects are influenced by several internal parameters:

- **Execution time**
  - Code structure (language, conditionals, cycles)
  - Operating system or kernel (system calls)
  - DMA, caches, instructions pipeline
- **Response time and regularity**
  - Multi-tasking
  - Access to shared resources (buses, disks, comm ports,...)
  - HW Interrupts

Requirements of Real-Time Systems

Typical **requirements** imposed on real-time systems:

- **Functional**
- **Temporal**
- **Dependability**
- **…**

Non-Functional

Functional requirements

Example

Acquiring environment data

- Sample process variables (**real-time entities**) either continuous or discrete

Direct Digital Control (DDC)

- Realize low-level feedback control (sensor/actuators loops)

Human-Machine Interface (HMI)

- Provide information on the system state, carry out logging, perform configuration

Functional requirements

Acquiring environment data

The **process real-time entities**, upon acquisition, are internally accessible to the computing system through **local images** (internal variables).

Each **local image** of a real-time entitiy has a **limited temporal validity** due to the process dynamics.

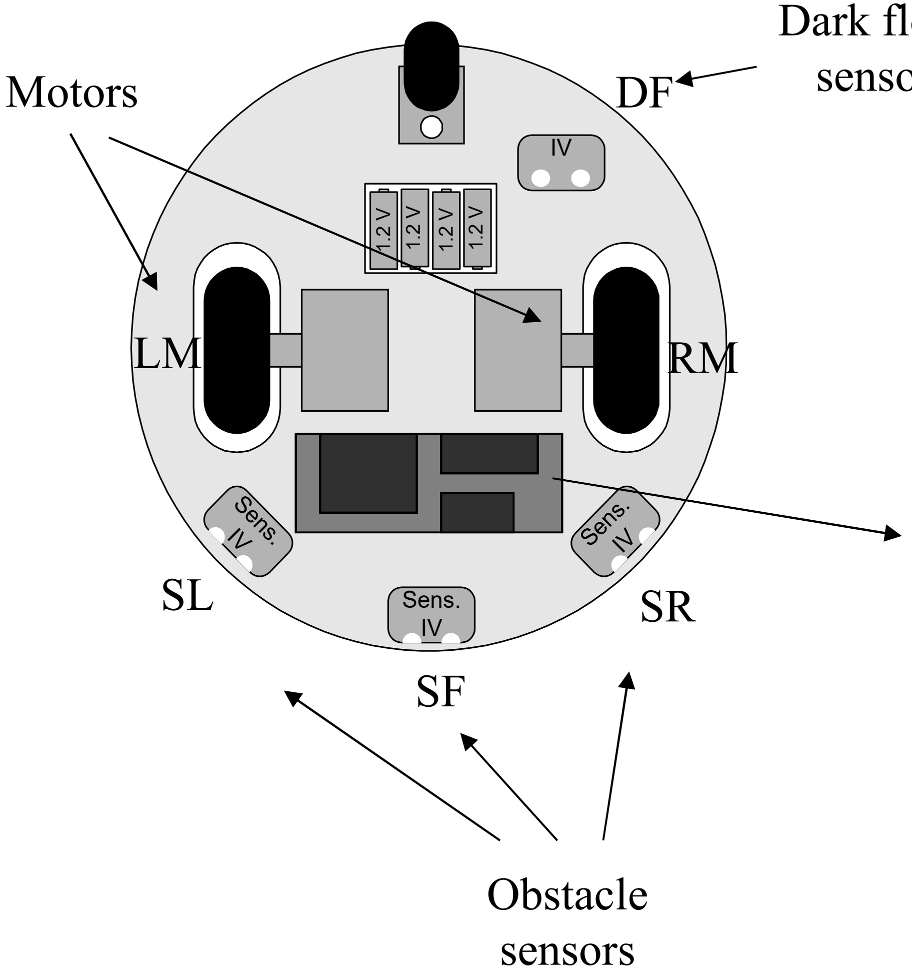
The **set of local images** of the real-time entities forms the **real-time database**.

The **real-time database** needs to be **updated** whenever there is a **change** in the state/value of a real-time entity.

Functional requirements

<u>Example:</u>

A simple mobile robot



<u>RT entities:</u>
sensors:
SL, SF, SR, DF

Motors speeds:
LM, RM

<u>Internal images:</u>
sensors:
SL', SF', SR', DF'

motors speeds:
LM', RM'

RT database

Temporal requirements

Arise from the **dynamics of the process** to be controlled/monitored

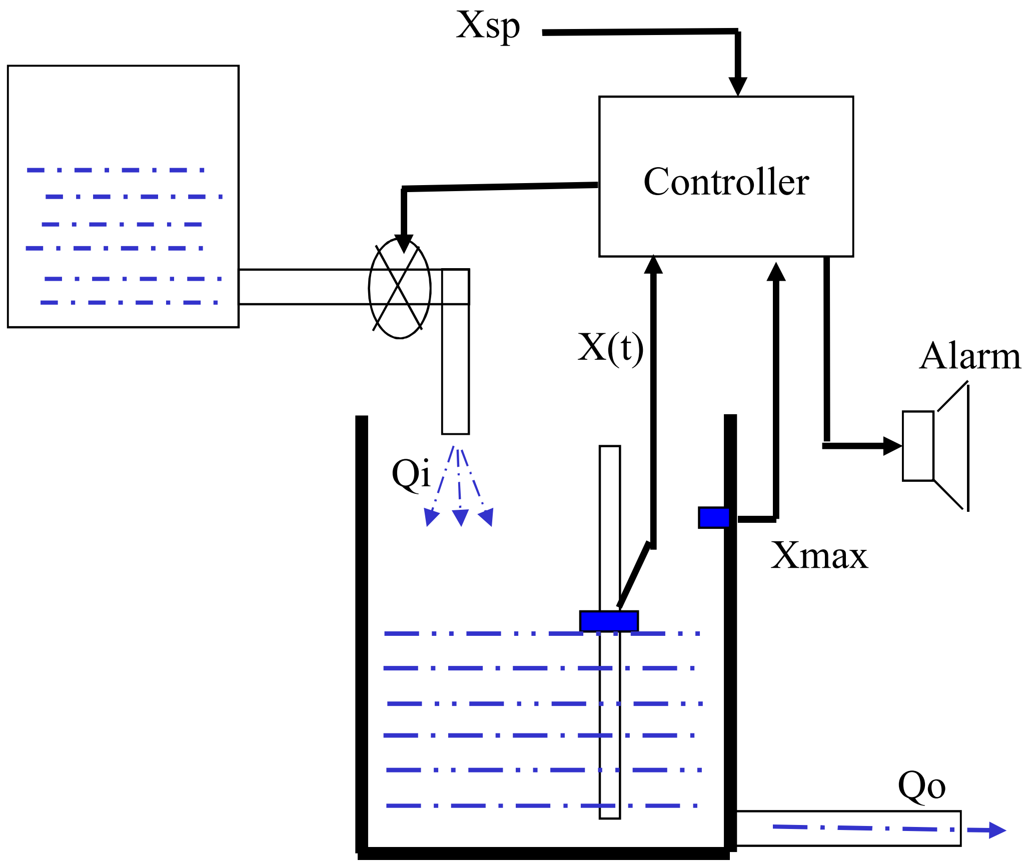
Sets **constraints**:

- To the **observation delays** of the system state
- To the **computing delays** of the control/actuation values
- To the **delay variations** of the previous (***jitter***)

In some cases, such constraints must be met in
all instances (including the worst-case)
and **not only on average terms.**

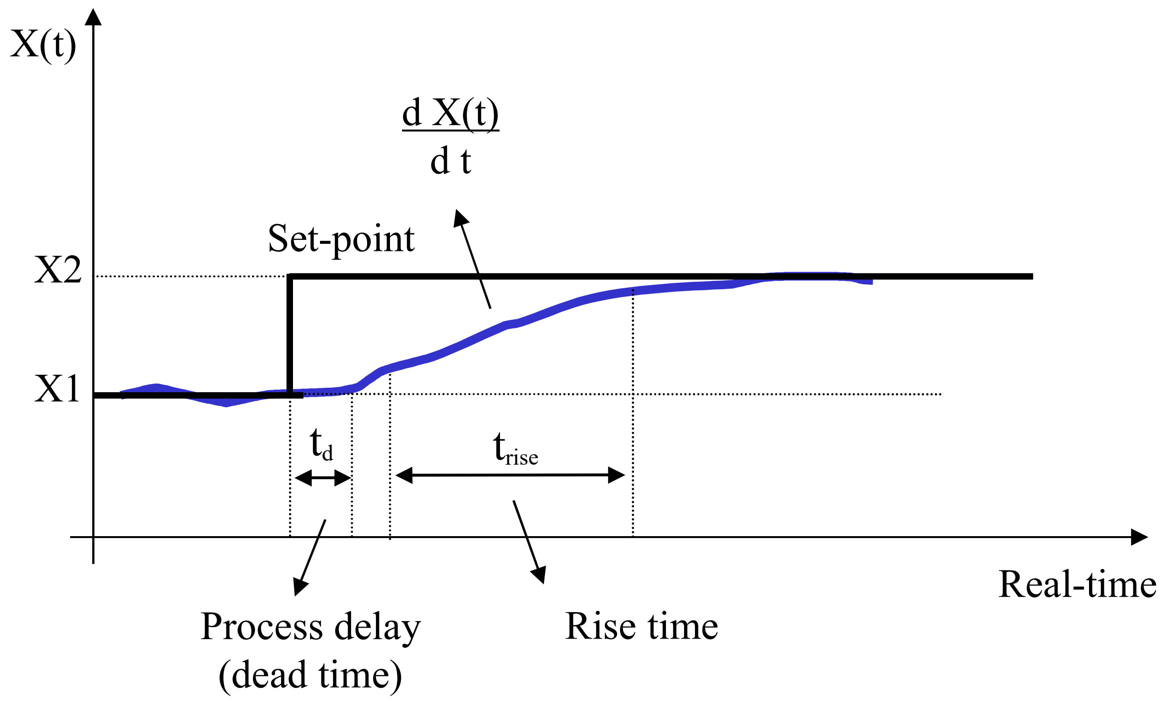
Temporal requirements

Control of the level of a liquid in a tank



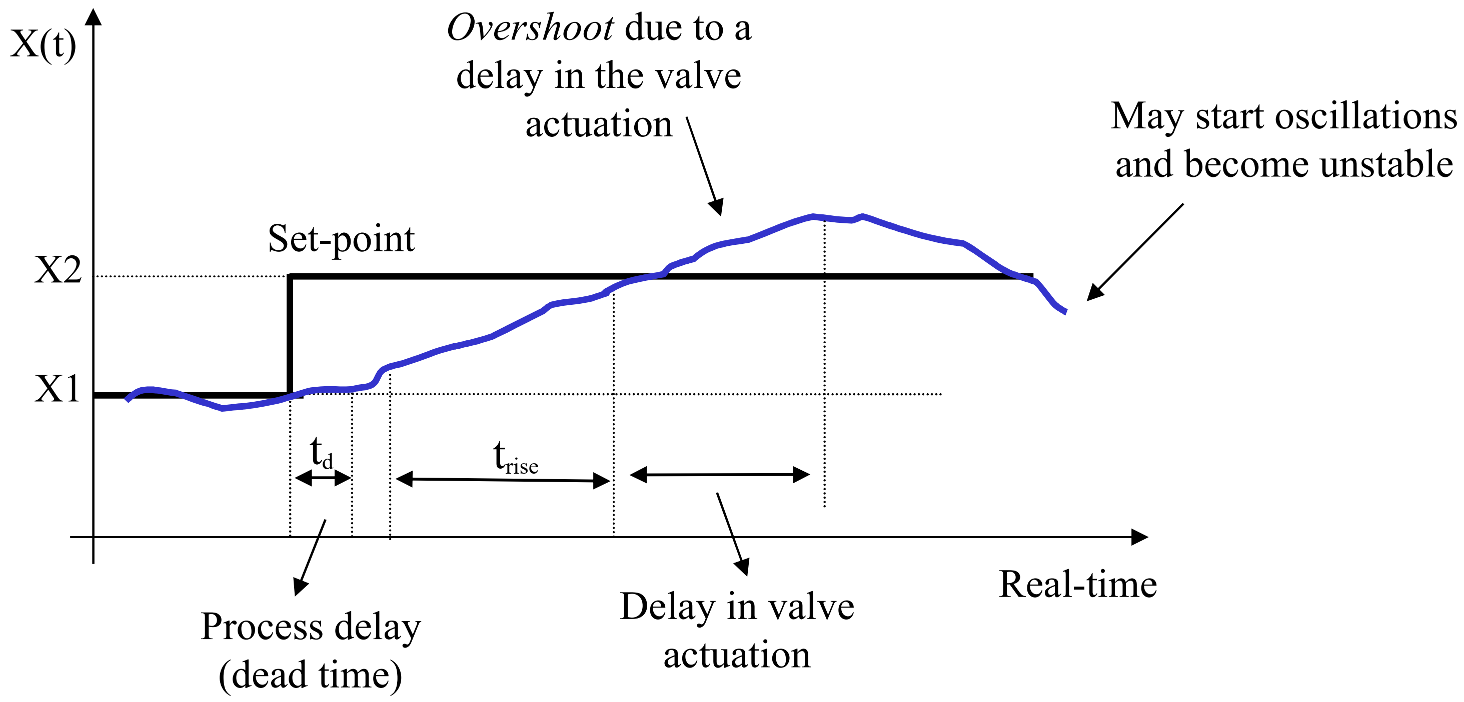
Temporal requirements

Control of the level of a liquid in a tank



Temporal requirements

Delayed actuation –control degradation



Temporal requirements

(Control systems)

Sampling period – $\mathbf{T_s}$ (~< 1/10 t_{rise} - *quasi-continuous control*)

Maximum delay in the valve response – $\mathbf{d_{max}}$ (< $\mathbf{T_s}$)

(control delay – can be compensated in many cases with a model)

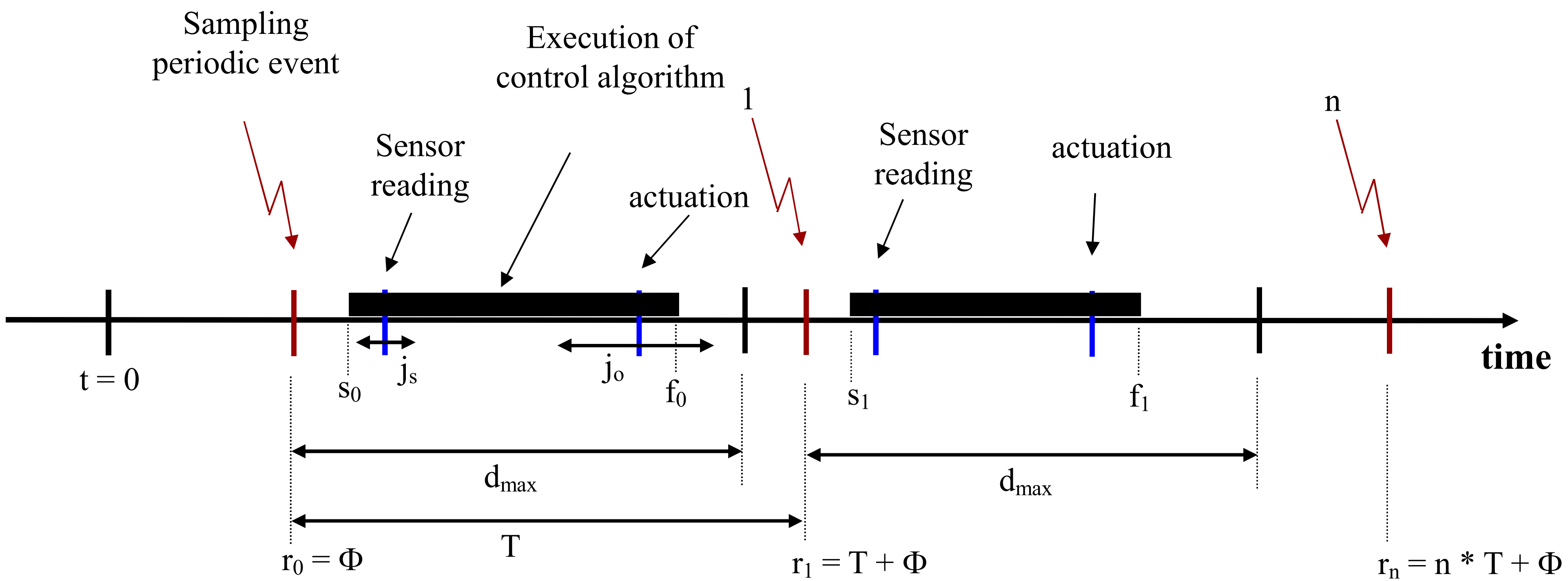
Variations in the level reading (***jitter***) – $\mathbf{j_{s,max}}$ (<< $\mathbf{d_{max}}$)

Variations in the valve actuation delay (***jitter***) – $\mathbf{j_{o,max}}$ (<< $\mathbf{d_{max}}$)

(harder to compensate – degradation of the quality of the control)

Maximum delay in alarm signaling – $\mathbf{d_{al,max}}$

Temporal requirements



r_n – activation

s_n – start of execution

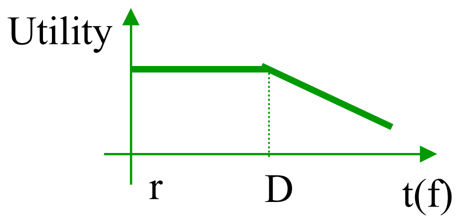
f_n – end of execution

Temporal requirements

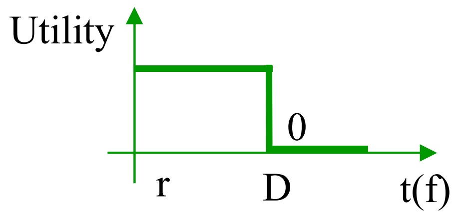
Classification of real-time constraints:

(according to the utility of the result to the application)

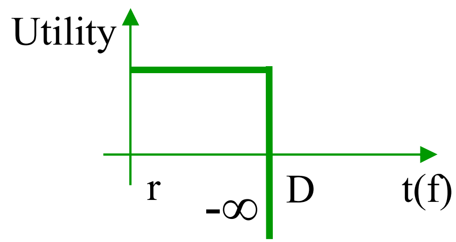
- ***Soft*** – Time constraint according to which the associated value keeps **some utility** to the application even after a deadline D, despite a possible degradation of the quality of service



- ***Firm*** – Time constraint according to which the associated value loses **any utility** to the application after a deadline D



- ***Hard*** – Time constraint that, when not met, may generate a **catastrophic failure**



Temporal requirements

Classification of Real-Time Systems:

(according to the type of time constraints they have to cope with)

- ***Soft Real-Time*** – There are ***firm*** or ***soft*** time constraints, only (e.g., simulators, multimedia systems)

- ***Hard Real-Time*** – There is at least one ***hard*** time constraint. These are **safety-critical** systems (e.g. transportation systems, weapons control, process control of dangerous materials…)

Dependability Requirements

Real-Time Systems are frequently used in **critical applications** either in terms of **safety** and **economy** (e.g., power plants, traffic control (air, rail, road), process industry…).

This leads to a specific requirement for:

High reliability – In *hard real-time* systems one typically finds **ultra high reliability** requirements ($\lambda < 10^{-9}$ failures/hour – cannot be verified experimentally! 1 failure in 114155 years!).

Dependability Requirements

Important aspects concerning critical safety:

- **Stable interfaces** between the critical and non-critical subsystems in order to prevent **error propagation**.
- **Well defined worst-case scenarios**. The system must possess the **adequate resources** to withstand a worst-case operational scenario without need for probabilistic arguments, i.e., it should provide **service guarantees** even in such cases.
- **Architecture** composed by **autonomous subsystems**, whose properties can be verified independently from the other subsystems (composability).

Wrapping up

- Notion of **real-time** and **real-time system**
- Purpose of studying RTS – obtain **guarantees** of **adequate temporal behavior**
- Aspects to consider: **execution time**, **response time**, **regularity** of generating/handling periodic events
- RTS requirements: **functional**, **temporal** and **dependability**
- Notion of **real-time database**
- **Soft**, **firm** and **hard** time constraints
- **Hard real time** vs **soft real time**
- Importance of the **worst-case scenario**

part 2

Computing models

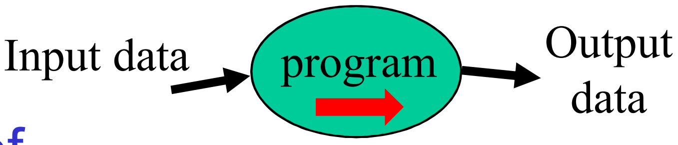
Models of tasks with explicit time constraints
Logic and temporal control (by events -ET and by time -TT)

Computing models

Transformational model

- A program starts and finishes, transforming input data in output results.

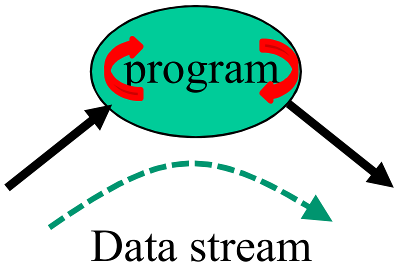
Reactive model



- A program executes indefinitely a sequence of interactions, for example operating over a continuous input data stream.

Real-time model

- **Reactive model** in which the output stream has to be synchronized with the input data stream, thus imposing time constraints on the program execution.



Determinism versus Predictability

(within reactive systems)

<u>Determinism</u>

- Feeding a program with the **same sequence of data inputs** originates the **same sequence of data outputs**.
  (logical property)

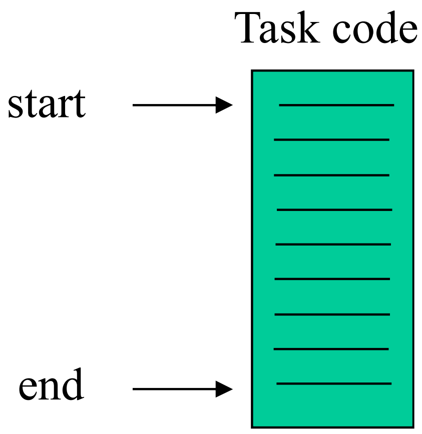
<u>Predictability</u>

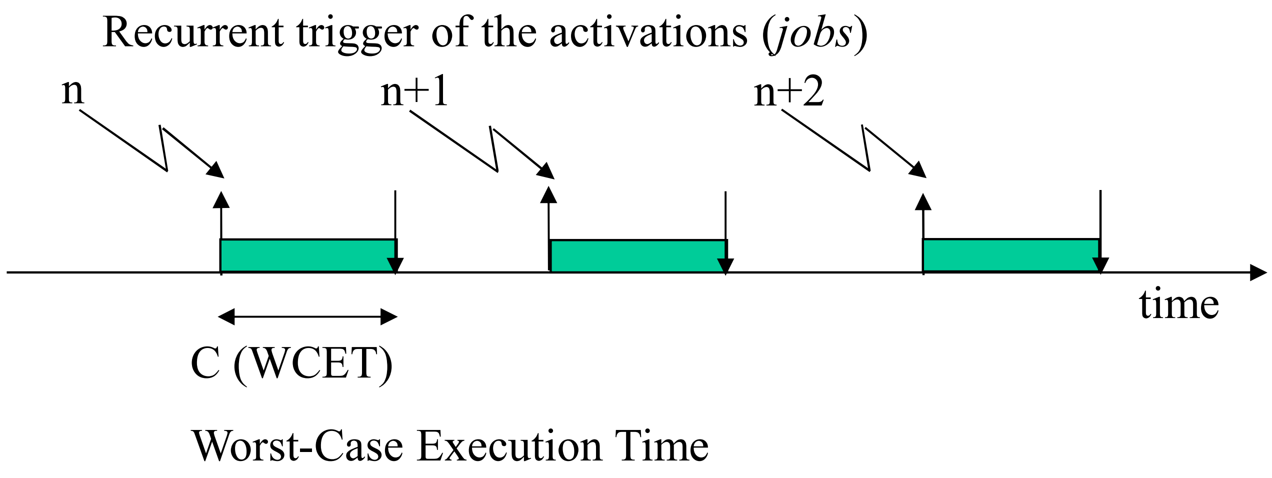
- feeding a program with the **same input sequence** will generate the **same output sequence** with a known **delay** or within a know **time window.**
  (temporal determinism – logical and temporal property)

Real-Time model

Definition of task (recurrent)

Infinite sequence of activations (instances or jobs), each executing a set of operations (a **given function**).



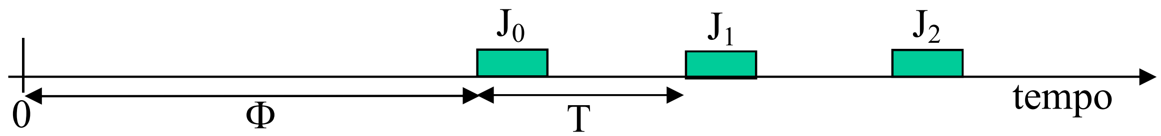


Real-Time model

Concerning recurrency of arrivals, a task can be

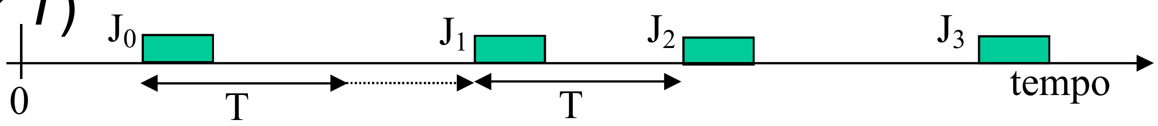
- **periodic**

  Instance *n* activated at $a_n = n*T+\Phi$



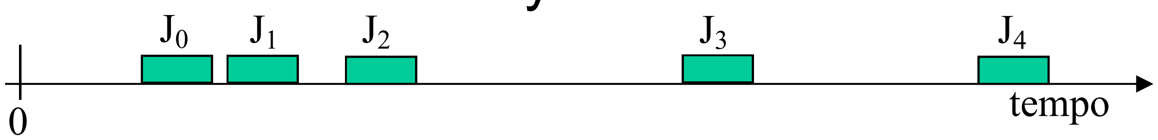
- **sporadic**

  Minimal time between consecutive activations (minimum inter-arrival time: *mit or T*)

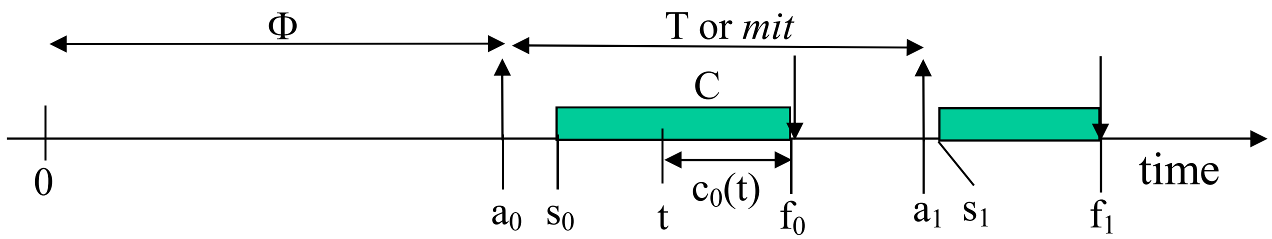


- **aperiodic**

  Can only be characterized stochastically



Real-Time model



<u>Characterization of tasks</u>

- **C – Worst-case execution time (WCET)**
- **T – period (periodic)**
- **Φ –relative phase/offset = instant of 1st activation (periodic)**
- **mit – *minimum inter-arrival time* (sporadic)**
- **a_n – activation instant of the nth instance**
- **s_n – start of execution of the nth instance**
- **f_n – finishing time of the nth instance**
- **$c_n(t)$ – maximum remaining execution time at instant t of the nth instance**

Real-Time model

Tasks requirements:

- **Temporal** – time constraints on the **finishing instants** or for generating certain **output events**.
- **Precedence** – establish a certain **order of execution** among tasks.
- **Resource usage** – need for using **shared resources** (e.g. communication ports, buffers in shared memory, global variables, system peripherals…). May imply the use of **atomic operations** (which execution sequence cannot be interrupted)

Real-Time model

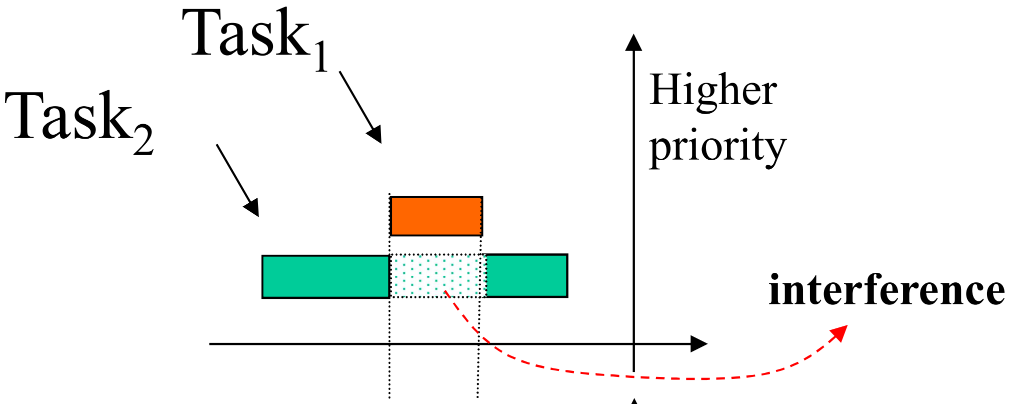
Preemption

- When a task can be **temporarily suspended** for the execution of another one with **higher priority**, it is **preemptable**.
- When a system makes use of the preemptability of tasks is it called **preemptive**.
- A set of tasks is said to admit **full preemption** when all its tasks can be preempted in any point of their execution (independent tasks)

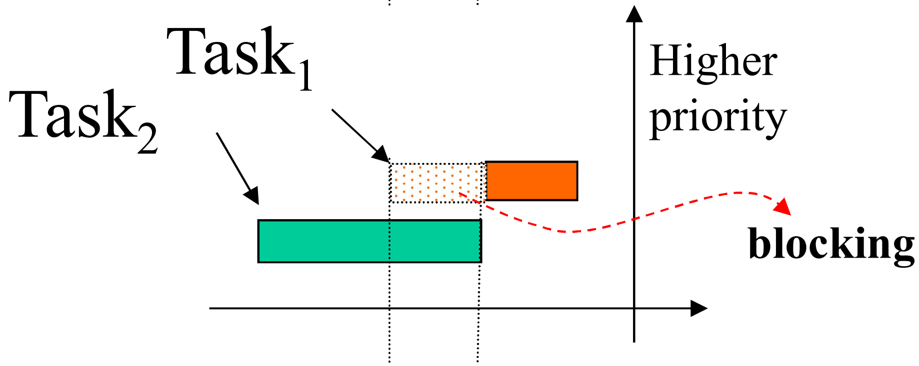
Note: the access to **shared resources** (thus tasks with **dependencies**) may impose restrictions on the level of preemptability of a task.

Real-Time model

- **Execution with preemption**



- **Execution without preemption**



Real-Time model

The temporal requirements can be of diverse types:

- ***Deadline*** – Upper bound to the finish instant of a task.
- **Window** – Lower and upper bounds to the finish instant of a task.
- **Synchronism** – Upper bound to the interval between two output events (there are types, too).
- **Distance** – Upper bound to the interval between the finishing time of two consecutive jobs

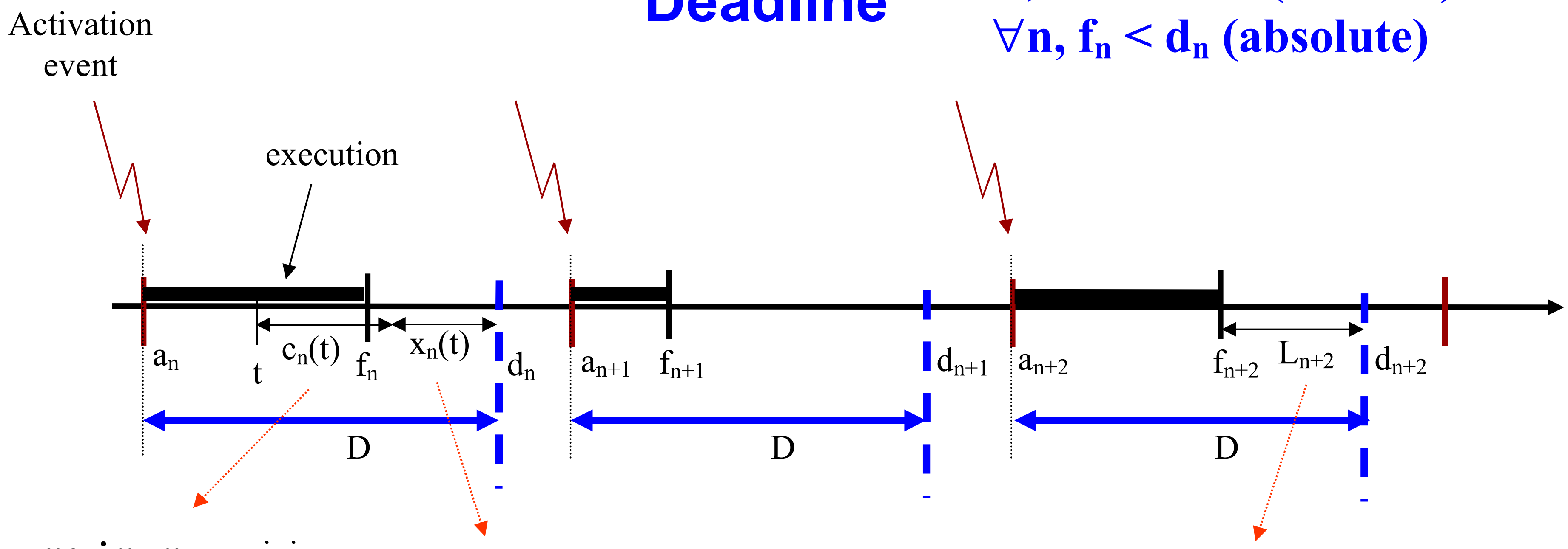
The ***deadline*** type is the most common!

Real-Time model

Deadline

$\forall n,\ f_n - a_n < D$ **(relative)**

$\forall n,\ f_n < d_n$ **(absolute)**



$c_n(t)$ = **maximum** remaining execution time at instant t

$x_n(t)$ = **minimum** task slack

$x_n(t) = d_n - t - c_n(t)$

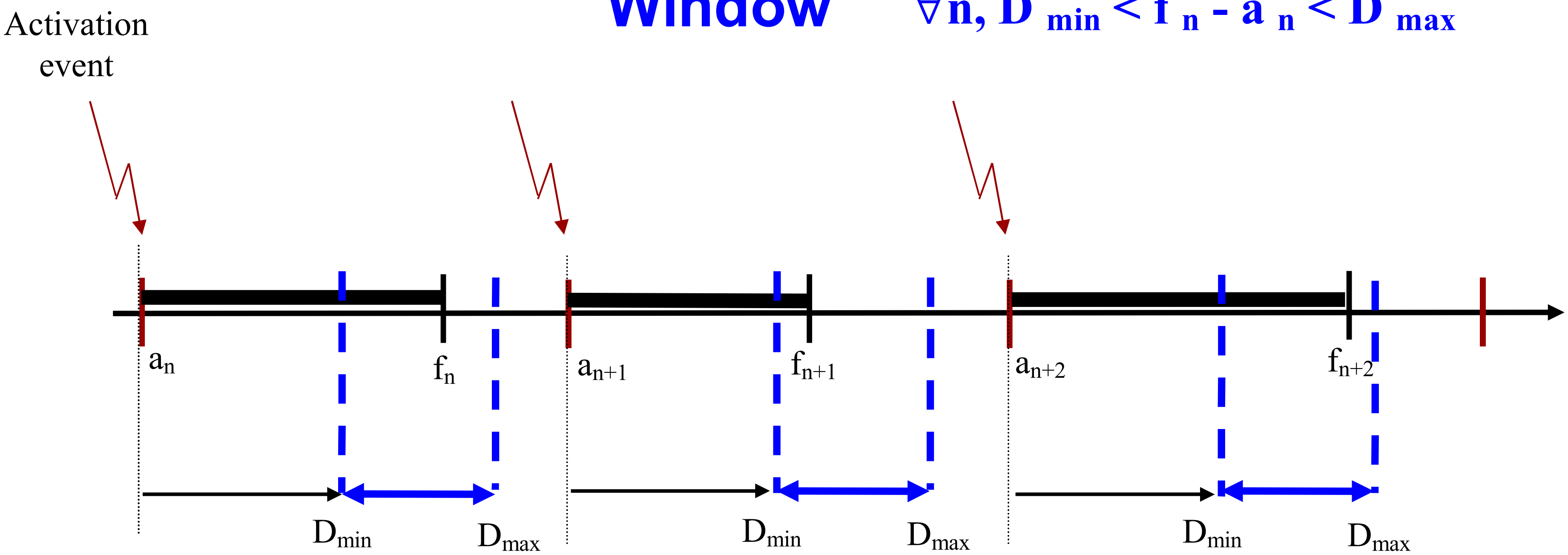
$L_n = f_n - d_n$ = delay of nth activation

$L_n \le 0 \rightarrow$ timely termination

$L_n > 0 \rightarrow$ late termination

Real-Time model

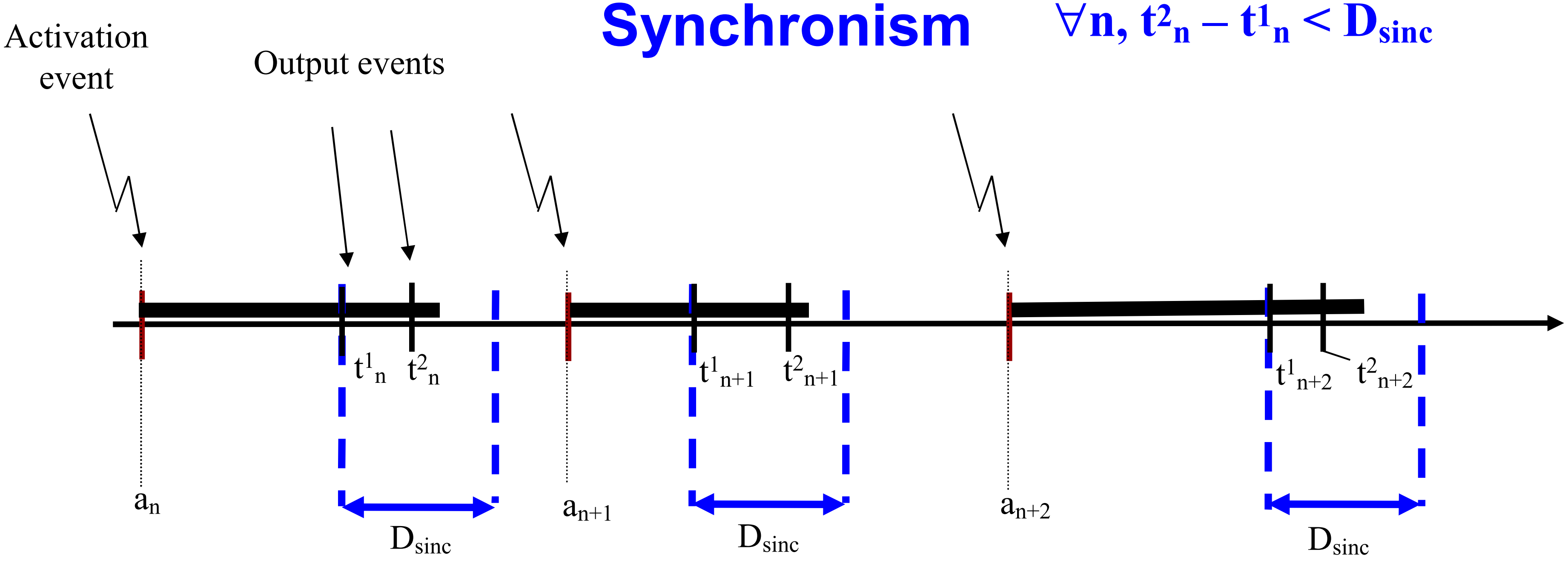
Window $\forall n, D_{min} < f_n - a_n < D_{max}$



Real-Time model

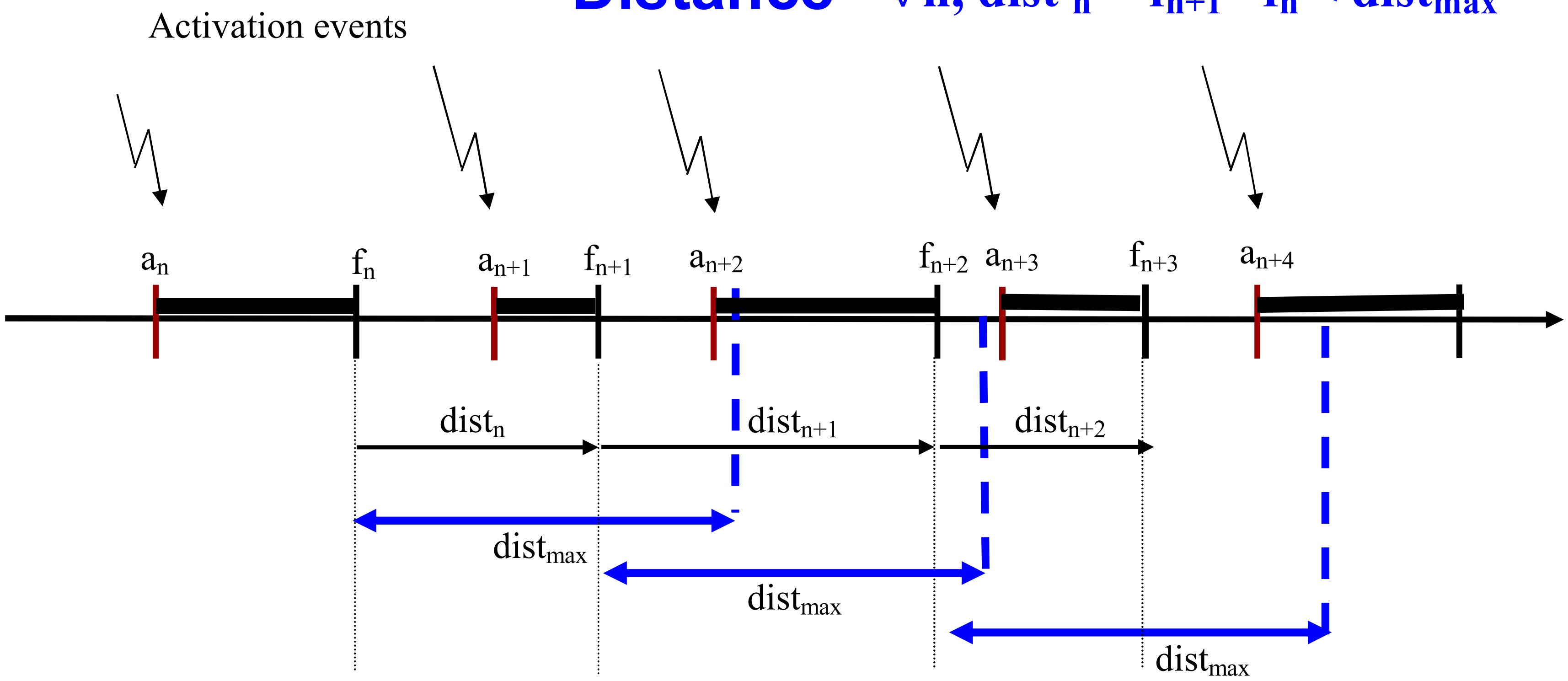
Synchronism

$\forall n,\ t^2_n - t^1_n < D_{sinc}$



Real-Time model

Distance $\forall n, \text{dist}_n = f_{n+1} - f_n < \text{dist}_{max}$



Real-Time model

Examples of tasks characterization:

- **Periodic:** $\tau_i = \tau_i\ (C_i, \Phi_i, T_i, D_i)$

  $\tau_1 = \tau_1\ (2,5,10,10)$ $\tau_2 = \tau_2\ (3,10,20,20)$

- **Sporadic:** Similar to periodic but with *mit_i* in the place of T_i and Φ_i is not typically used (could mean a minimal interval until the first activation).

  $\tau_i = \tau_i\ (C_i, mit_i, D_i)$

  $\tau_1 = \tau_1\ (2,5,5)$ $\tau_2 = \tau_2\ (3,10,7)$

Logic and temporal control

Logic control

Program control flow, i.e., effective sequence of operations to be executed – **fundamental to determine C (WCET)**

Temporal control

Control of the **execution instants** of the program operations (e.g., activations, verification of time constraints,...)

Temporal control

Triggering tasks (or functions)

By time (*time-triggered*)

The execution of tasks (or functions) is triggered by a control signal based on the progression of time (e.g., a periodic interrupt).

By events (*event-triggered*)

The execution of tasks (or functions) is triggered by an asynchronous control signal generated upon a change in the system state (e.g., by an external interrupt).

Temporal control

Activating tasks by Time

time-triggered (TT) systems

- Typically used in automatic control (sampling of continuous variables).
- There is a common time base (allows establishing offset relationships or relative phasing)
- CPU utilization is constant even when there are no variations in the system state.
  - **Well defined worst-case situation**

Temporal control

Activating tasks by Events

event-triggered (ET) systems

- Typically used in monitoring sporadic conditions in the system state (e.g., alarms or asynchronous service requests).
- CPU utilization (and not only CPU!) is variable according to the frequency of event occurrence.
    - **Poorly defined worst-case situation**
      either *probabilistic arguments* are used
      or the *maximum event rate must be bounded*

Temporal control

Example: Consider the following task sets and determine the worst-case response time of each task

- TT $\{\tau_i = \tau_i\ (C_i=1,\ \Phi_i=i,\ T_i=5,\ D_i=T_i\quad i=1..5)\}$

- ET $\{\tau_i = \tau_i\ (C_i=1,\ (\Phi_i=0),\ mit_i=5,\ D_i=mit_i\quad i=1..5)\}$

Moreover, determine the average and maximum **CPU utilization**
**** utilization = sum$_{i=1..N}$(execution time/activation period) ****
for both cases, considering that on average the ET tasks are activated once every 100 time units.

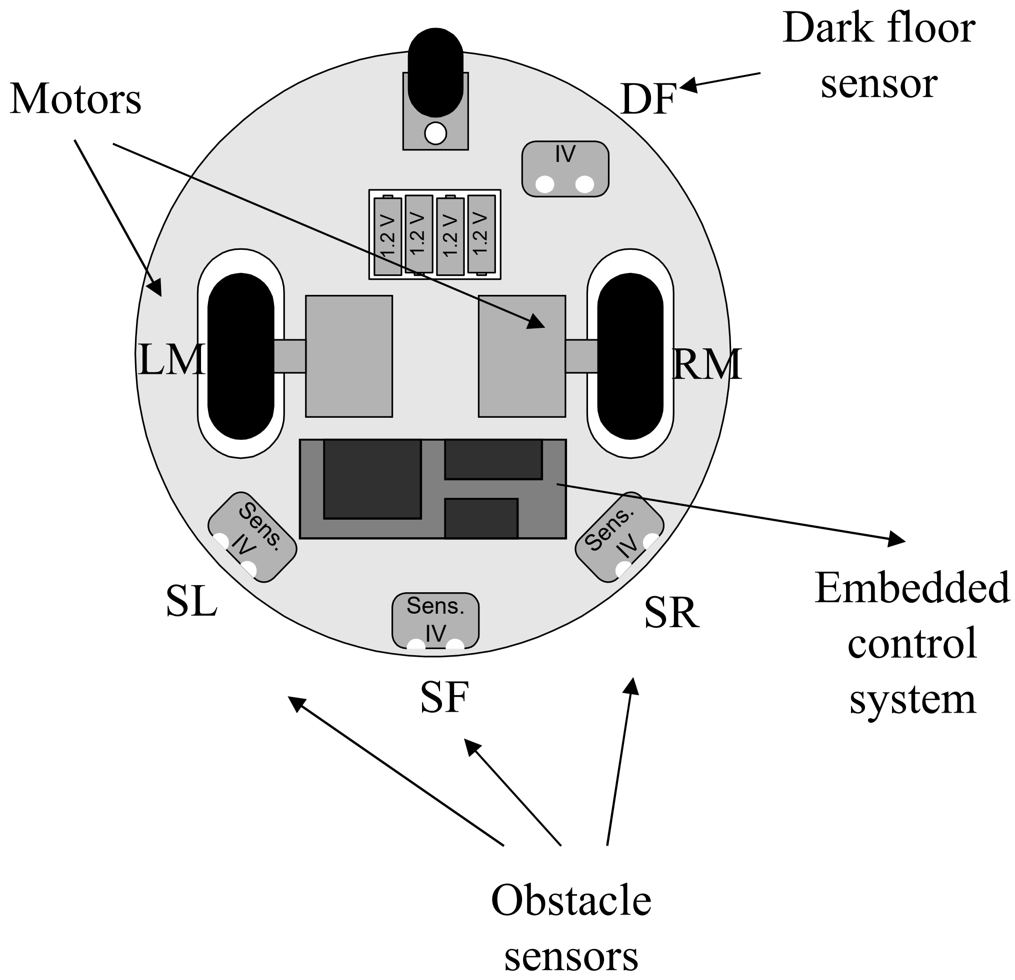
Modeling a simple mobile robot

Functional requirements:

- **Differential** motion control
- If **dark floor**, then **stop and finish**
- Else if **wall**, then **follow wall**
- Else if **obstacle**, then **go around** or **move back**
- Else **move straight**

Non-functional aspects:

- Max speed – **50 cm/s;** time to stop – **0.3s**
- Obst. sensors range – **5-25 cm**
- Dark floor sensor – **within 1s**

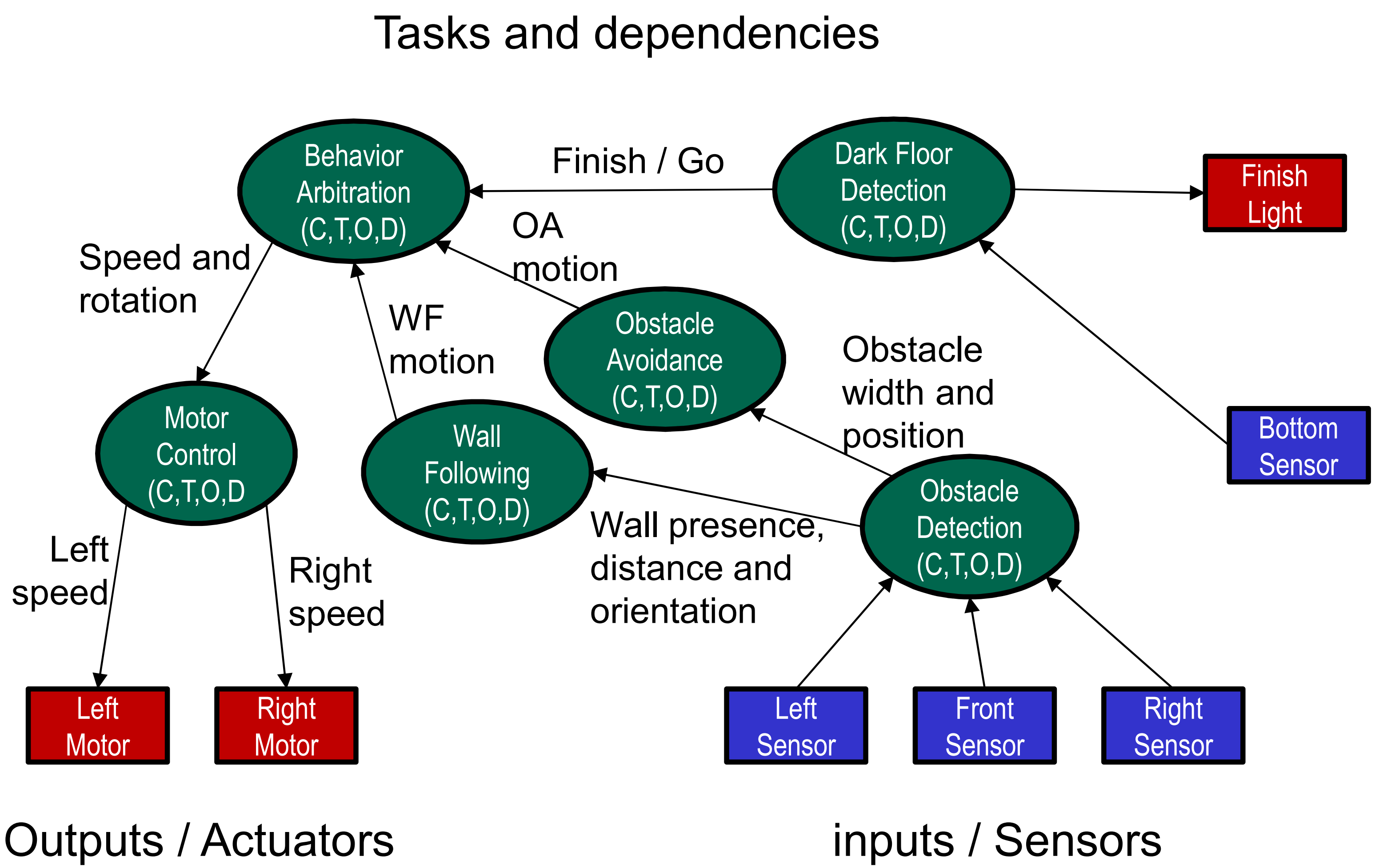


A procedure-based SW modeling approach

RT entities:
sensors, motors, light

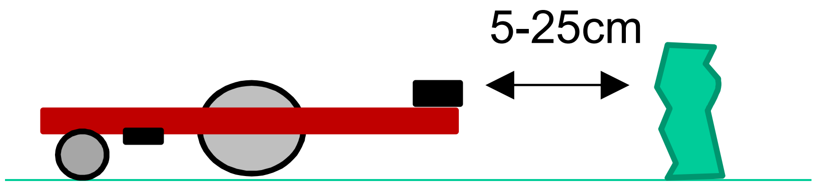
Inter-task communication:
Global variables
(good to represent them explicitly in the model as shared data items)

Periodic (TT) approach:
C – execution time
T – period
O – offset
D – deadline

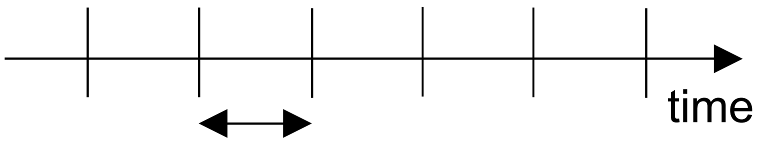


Time-constraints

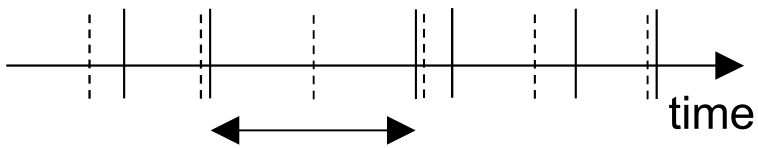
Obstacle detection



20cm range, @ 50cm/s → 0.4s window to detect the obstacle
0.3s for stopping → actual time for detecting the obstacle 0.1s
Obstacle sampling must guarantee no more than 0.1s between samples



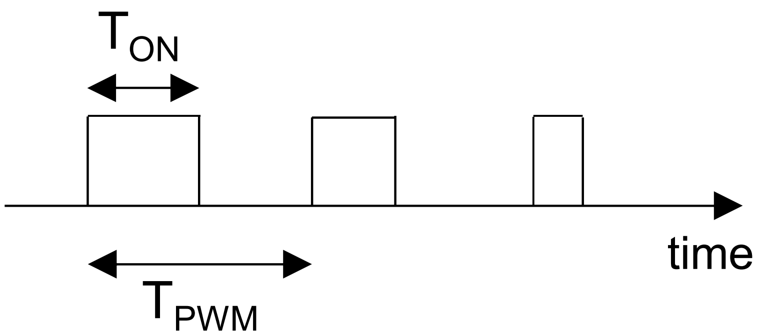
Jitter-free sampling → max T = 0.1s



100% jitter → max interval between samples can duplicate → max T = 0.1s/2 = 50ms

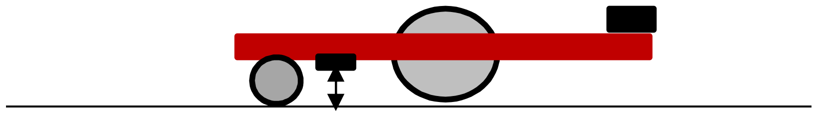
Time-constraints

Motor control (PWM)



Max T_{PWM} → 1/10 trise (~1s) → max T_{PWM}=100ms
T_{ON} = f(v), v → desired speed

Dark floor detection



Reliable detection → several consecutive detections + hysteresis (say 5 samples)
1s / 5samples → max T = 200ms

Behavior tasks

Depends on desired reactivity
Normally equal to Obstacle detection

Wrapping up

- Computing models (**real-time model**)
- Real-time tasks: periodic, sporadic and aperiodic
- Temporal constraints: **deadline**, window, synchronism and distance
- **Logic control** and **temporal control**
- **Event-triggered** versus **time-triggered** tasks
- **Example model** and time constraints from a simple case study