

**BRNO UNIVERSITY OF TECHNOLOGY**  
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION  
DEPARTMENT OF CONTROL AND INSTRUMENTATION  
Kolejní 4, 616 00 Brno  
tel.: +420 5 4114 1113 fax: +420 5 4114 1123  
E-mail: [kucera@feec.vutbr.cz](mailto:kucera@feec.vutbr.cz) <http://taceo.eu>



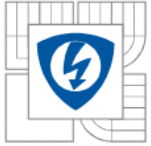
# Real-time Operating Systems

## Introduction

Pavel Kučera

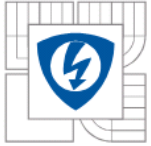
CAK, E112

[kucera@feec.vutbr.cz](mailto:kucera@feec.vutbr.cz)



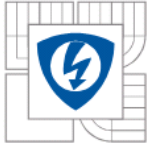
# Content

1. Super loop control
2. Multitasking in OS
3. What is RTOS
4. Basic Terms
5. Example
6. RTX – Clocks & Timing

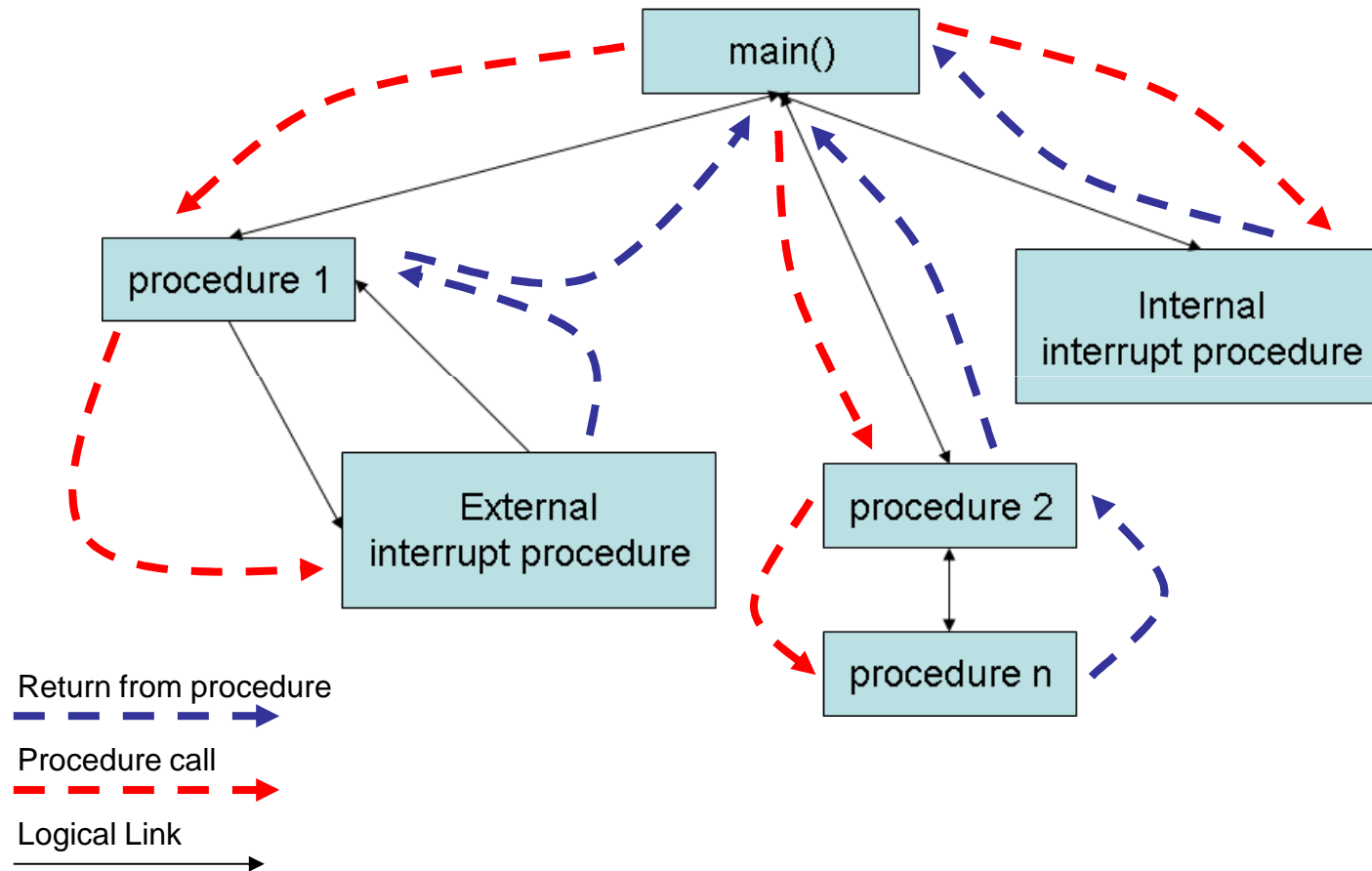


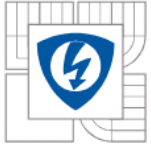
# References

- Web pages of the module: [www.taceo.eu/mrts](http://www.taceo.eu/mrts)
- Douglas, B., P. **Doing Hard Time**, Reading, Mass.: Addison-Wesley, 2000
- Li, Q., Yao, C. **Real-Time Concepts for Embedded Systems**, CMP Books, 2003
- Chowdary, V. P. **Simple Real-time Operating System: A Kernel Inside View for a Beginner**, Trafford Publishing, 2007

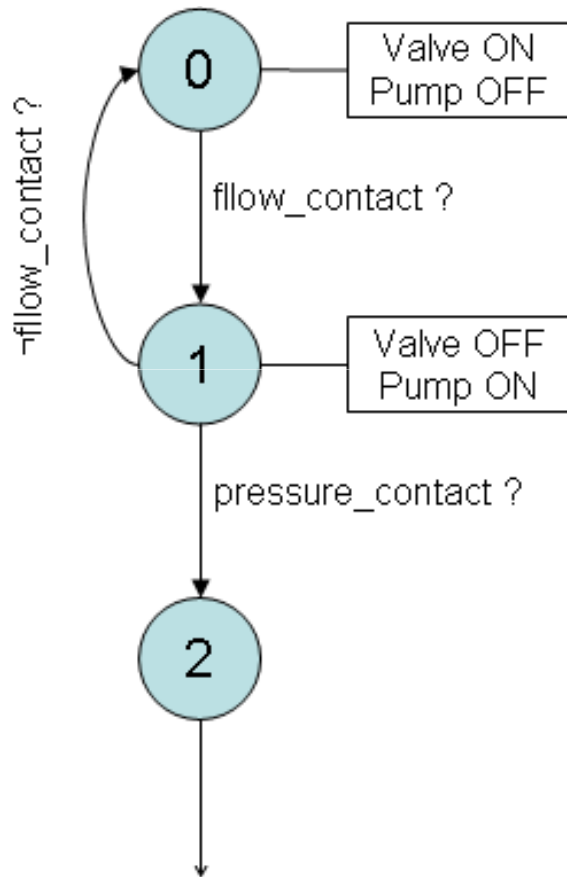


# Super loop Control





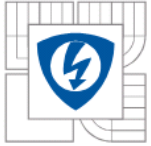
# Super loop Control



*state diagram*

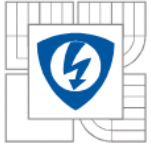
```
switch (actual_state) {  
    case 0: ActionInState0(); ConditionsInState0(); break;  
    case 1: ActionInState1(); ConditionsInState1(); break;  
    case 2: ActionInState2(); ConditionsInState2(); break;  
    .  
    .  
    case n: ActionInStateN(); ConditionsInStateN(); break;  
    default: DefaultAction(); DefaultConditions(); break;  
}  
  
ActionInState0() {  
    Valve = 1;  
    Pump = 0;  
}  
  
ConditionsInState0() {  
    if (flow_contact) actual_state = 1;  
}  
  
ActionInState1() {  
    Valve = 0;  
    Pump = 1;  
}  
  
ConditionsInState1() {  
    if (flow_contact) actual_state = 1;  
    if (!flow_contact) actual_state = 0;  
    if (pressure_contact) actual_state = 2;  
}
```

*synthesis in C*



# Super loop Control

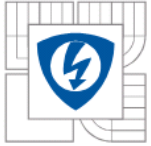
- An intuitive solution
  - Operating System is not necessary
  - Rapid implementation of simple tasks
- 
- + Simplicity of the system
  - + Speed of the system
  - Difficult to achieve Real-time behaviour of the system
  - Modification usually leads to completely new design



# Multitasking

## Motivation for multitasking system:

- Decomposition of the complex process to the simple task leads to simplification of the procedures and effortless debugging of the system.
- Failure of the single task must not necessarily leads to failing of the entire system (safety and fault-tolerant systems).
- Tasks can be prioritized and competed for restricted computer sources (CPU, memory, HW ...).
- Multitasking with the system of priorities and deterministic kernel scheduler are necessary conditions (but not sufficient) for real-time control of the process.



# Nonpreemptive Multitasking

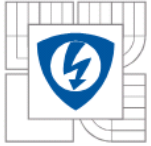
Nonpreemptive multitasking is based on a cooperation between running tasks.

The operating system never switch a context from a running task to another task.

Schedulers is either based on statically scheduled plan (FIFO, Cyclic executive buffer) or on a cooperation between the tasks (priority, FCFS).

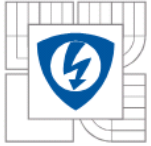
Nonpreemptive operating system has minimal or none capability to preempt, or interrupt, and later resume, other tasks in the system.

In extreme cases running task can block an activity of the operating system itself (Windows 3.x, Windows 95 for 16.bit applications, MAC OS, ...).

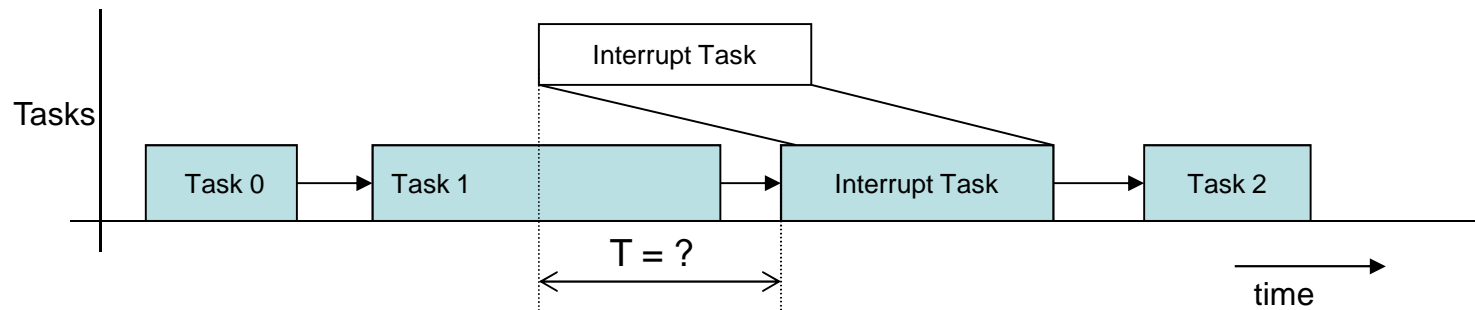


# Nonpreemptive Multitasking

- + Simple implementation of the system.
- + Low demands to CPU and memory.
- + Efficient switching between tasks (preemption is not often).
- Behaviour of the system can be nondeterministic in time.
- It is difficult to determine time response for many running tasks (~100) .
- Real-time behavior of the system requires careful design of the application.



# Nonpreemptive Multitasking

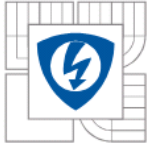


*Task 0* has finished and the scheduler has planned *Task 1*.

In the middle of the running *Task 1* the *Interrupt Task* has occurred.

Service routine of this interrupt can not be serviced as CPU executes *Task 1*.

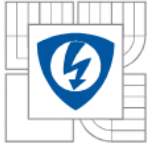
Time  $T$  is a duration between an occurrence of the *Interrupt Task* request and its execution. This time can not be predicted at all circumstances.



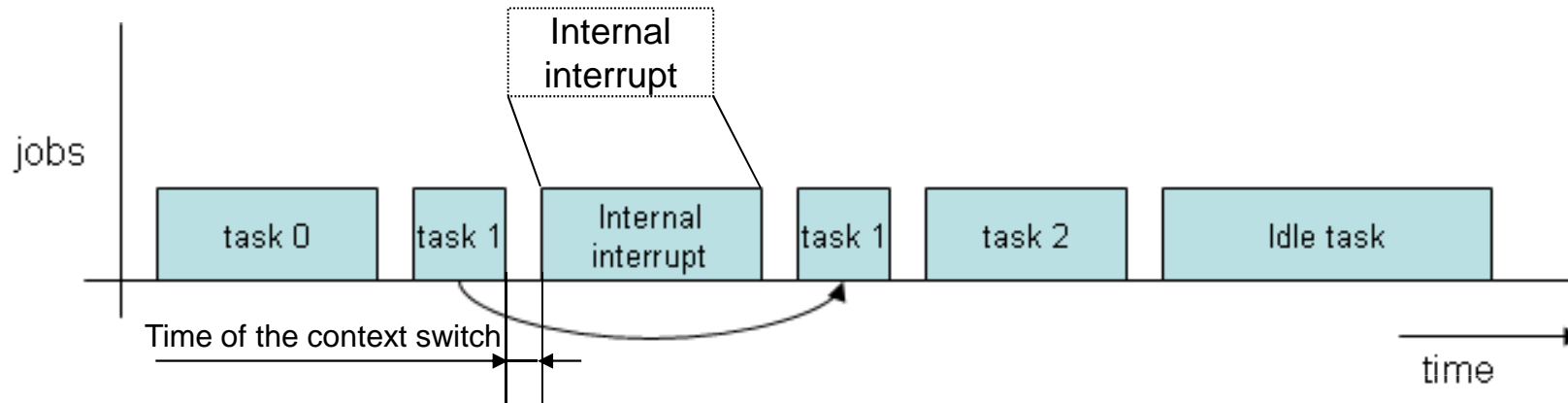
# Nonpreemptive Multitasking

Operating system that is based on nonpreemptive multitasking simplifies implementation of the complex systems. It has many similarities with super loop control.

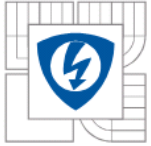
- + Simplicity of the Real-time System
- + Fast running with low overhead
- + Simple modification of the system (add/remove task)
- Real-time behavior is difficult to achieve for complex tasks



# Preemptive Multitasking

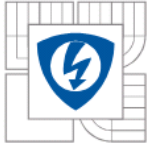


- + Simple modifications of the system
- + Real-time response is available
- Complexity of the RTOS
- Overhead of the RTOS



# Preemptive Multitasking

- Operating system that is based on preemptive multitasking is necessarily not real-time operating system. It can be only if the deterministic kernel scheduler is used.
- An application that is realized in RTOS is necessarily not running in real-time. An implementation of the task **must** by always adapted to the capabilities of the RTOS and target platform (HW).



# Motivation

**1982, Therac-25**, a compounding of process design, and implementation failures, software defect caused massive radiation killing 3 people.

source: Levenson, Nancy. *Safeware*, Reading, Mass.:Addison-Wesley, 1995

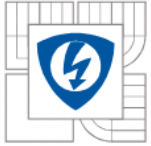
**1991, Patriot Failed to Launch an interceptor**

Patriot computer had 24-bit precision, so it chopped 0.0001% off timing values. System fell behind by 0.0034 sec per hour.

System had been running for 100 hours losing 0.3433 seconds.

Range gate affected cumulatively by timing error – looked in the wrong place about 687 metres. 28 casualties, more than 90 injured.

source: <http://shelley.toich.net/projects/CS201/patriot.html>



# Motivation

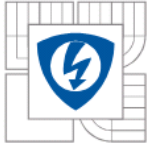
## **2008, Casting steel**

Inoperability of the real-time diagnostic system of the continuous casting process caused 20 mil. EUR financial lost in 2008.

## ***2009, D.C. Metro Red Line Crash***

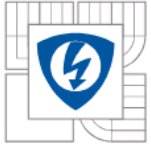
A Red Line Metrorail train crashed into a stationary train between Ft. Totten and Takoma stations. Nine people died and more than 70 people were injured. A train control system that should have prevented this deadly Metro crash failed in a test conducted by federal investigators.

source: [www.washingtonpost.com](http://www.washingtonpost.com)



# What are RTOS?

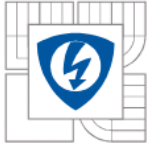
- Assembler, C, 8-bit 4MHz CPU, 2K ROM, 256B RAM for fridges, washers, microwave ovens, cardiac stimulators, gasoline direct injection ...
- C, C++, Ada, 64-bit 3GHz CPU, MB RAM, TB HDD for distributed or centralized control of plants, aircrafts, power plants ...
- Often without a keyboard, monitor, or human machine interface (HMI)
- Must operate for days/months/years/ without interruption of a failure (Fail-Safe, Fail-Stop, Fault-Tolerant)
- Must operate under difficult environment (-40...+85°C, humidity 0...100%, vibration, high altitude ...)



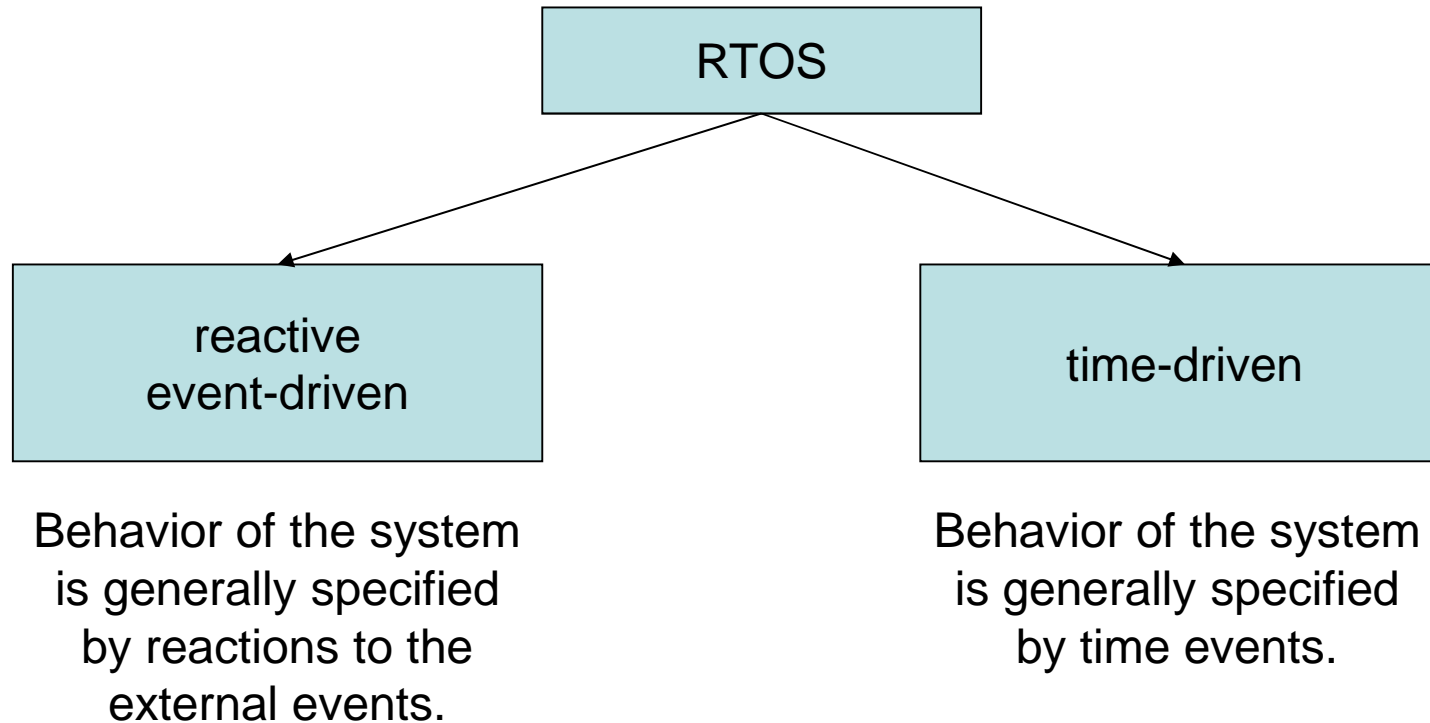
# Definition of RTOS

**RTOS** is such operating system that is capable to perform the calculations and react to the events in defined time intervals called deadlines.

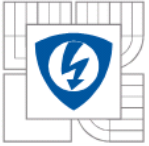
**RTOS** is such operating system that ensures that every correct information is at the correct time at the right place.



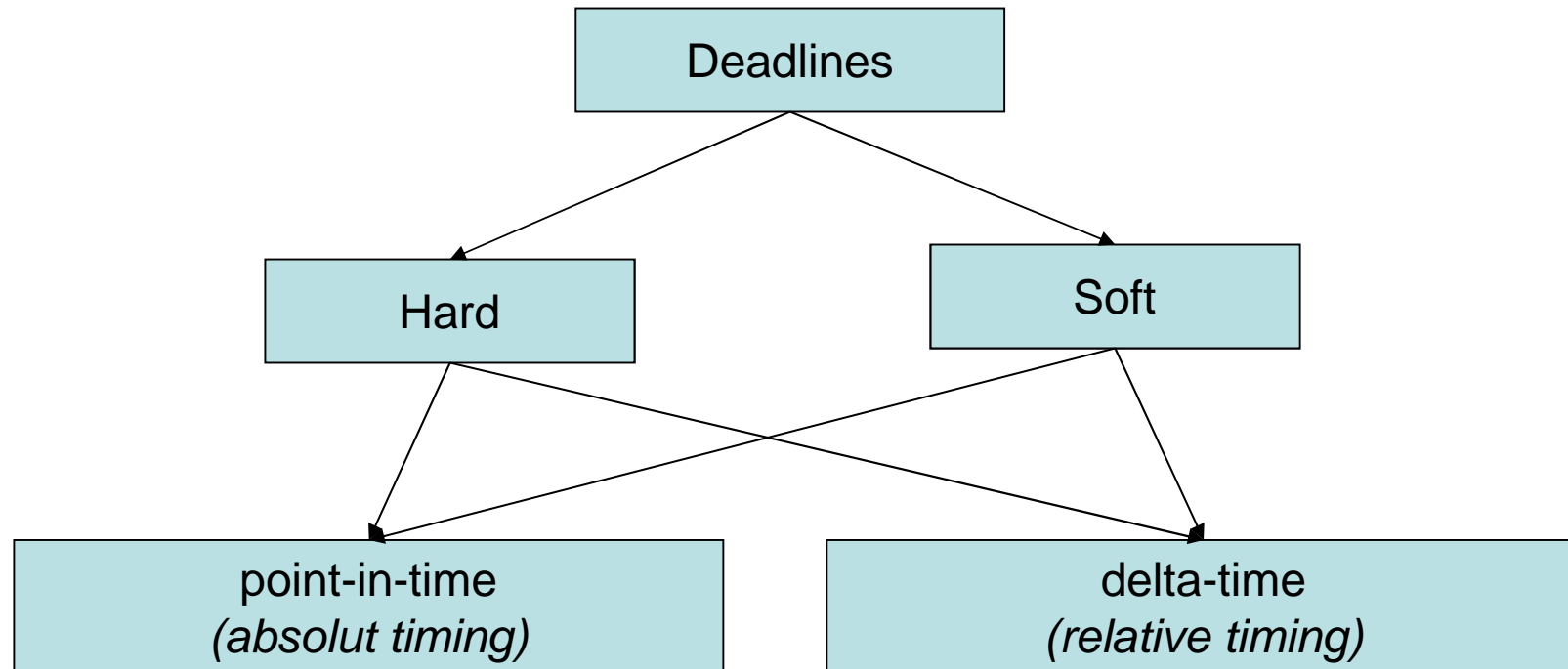
# Definition of RTOS

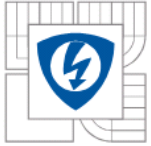


Real systems usually require both.

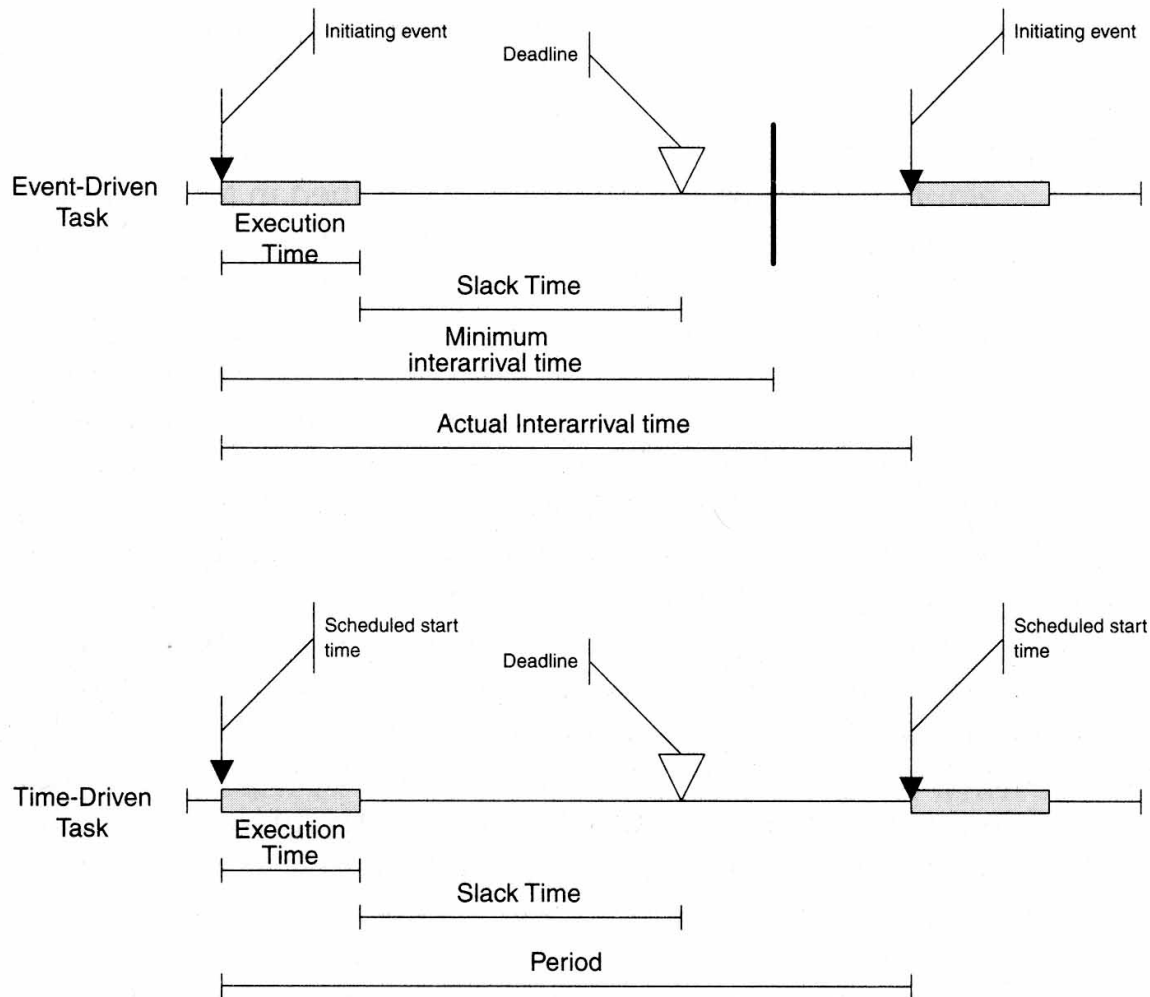


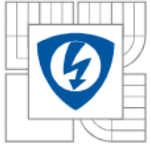
# Deadlines





# Deadlines





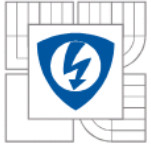
# Deadlines

## Hard deadlines

such deadlines that must be unconditionally fulfilled at all times.

If hard deadline is not fulfilled, the entire system can fail. It may leads to financial lost, injuring people or casualties.

Data in hard deadlines systems can be potentially dangerous if deadline is not fulfilled and the system must not work with it.



# Deadlines

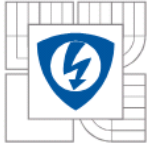
## Soft deadlines

such deadlines that must not be unconditionally fulfilled at all times. Usually it is enough to fulfill the deadline in some average time or between an average interval.

Data in soft deadlines systems can be used even if deadline is not fulfilled.

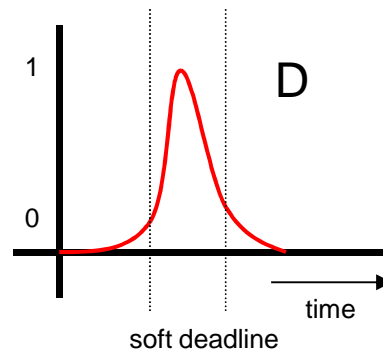
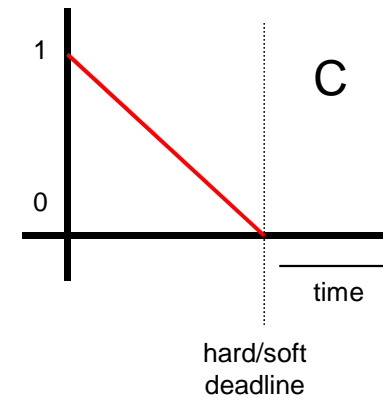
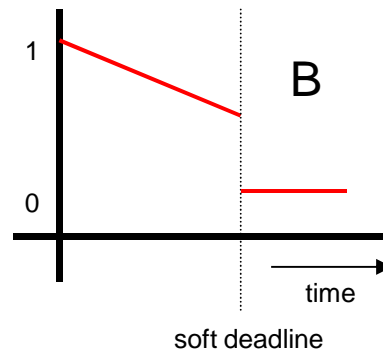
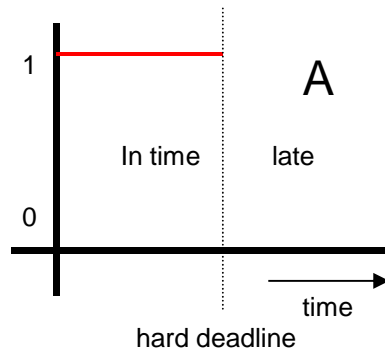
Hard real-time X Soft real-time

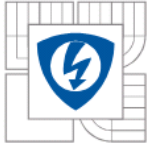
Real systems usually deal with both deadlines.



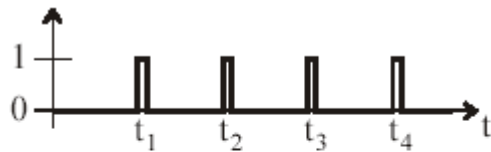
# Time/Utility Function

An ability of a system to utilize an information can be described by time/utility function.





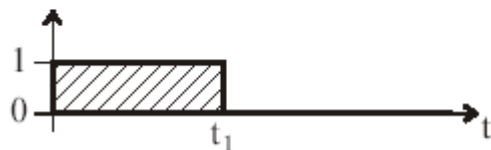
# Timing is systems



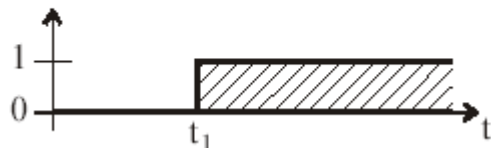
Action must be performed exactly in defined time points (usually periodic)  $t_1, t_2, \dots$ , (sampling, synchronization, communication ...).



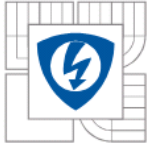
Action must be performed in defined intervals (external/internal interrupts).



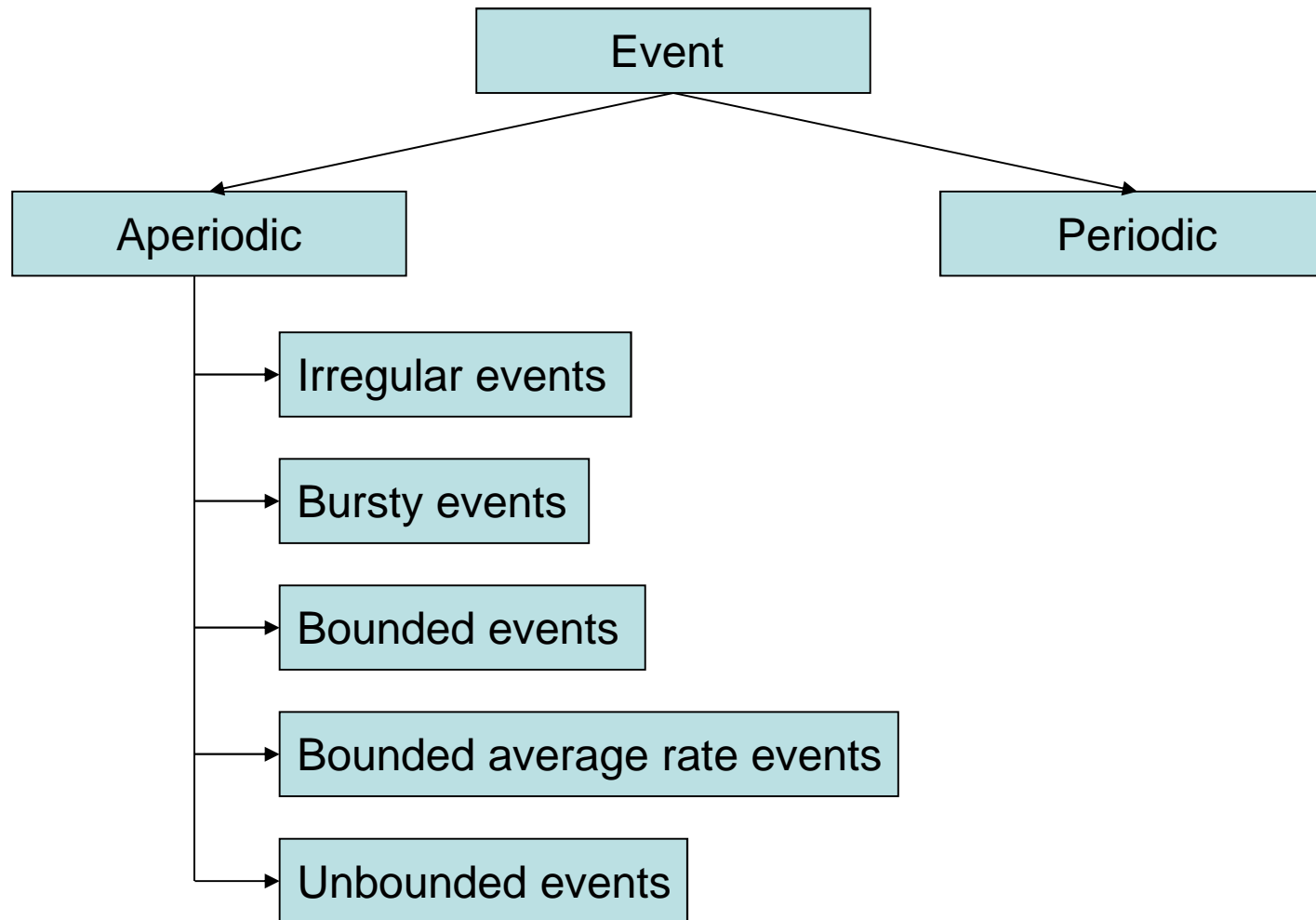
Action must be performed at the defined latest time point  $t_1$  (data processing, access to bus ...).

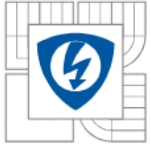


Action must not be performed earlier than the defined time point  $t_1$ .

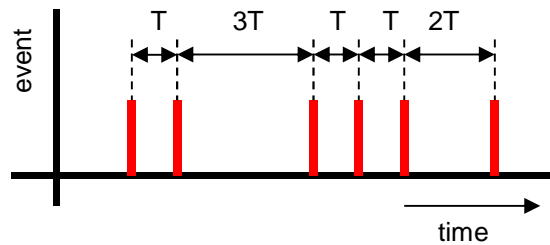


# Event Patterns

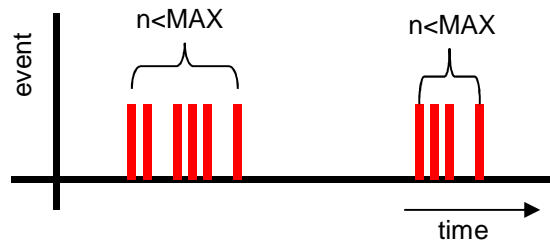




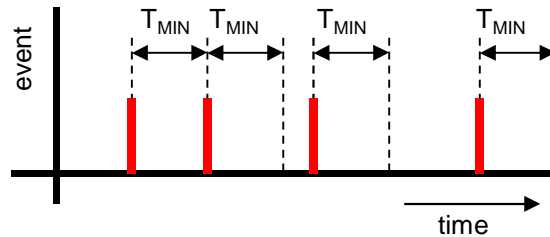
# Event Patterns



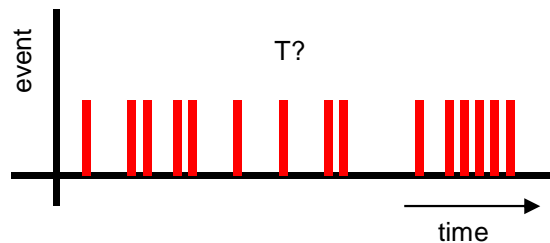
**Irregular** – a series of arrival times.



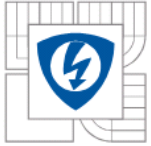
**Bursty** – arrival of events is correlated in time.



**Bounded** – arrival of events has a minimal distance.



**Unbounded** – arrival times of events are drawn as a renewal process from an underlying probability density function.



# Basic Terms

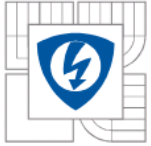
**Task** – a logic sequence of actions that are executed independently to the other actions. Tasks are usually. But not necessarily, divided into threads.

**Thread** – a primitive execution block for which separated program counter, stack and memory is allocated.

**Process** – an environment where tasks and their thread are executed.

**Kernel** – the core of the RTOS. Its main function is to deterministic planning of the threads execution.

**Context switch** – execution of one thread is preempted by the other thread.



# Relationship between Task-Thread-Process

process: MACHINE DIAGNOSTIC

