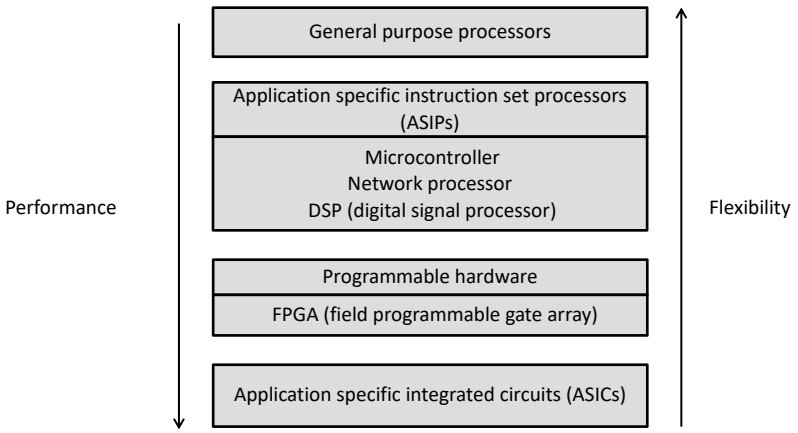
# COP 4610

Operating System Principles

## Embedded (Real-Time) Systems

1

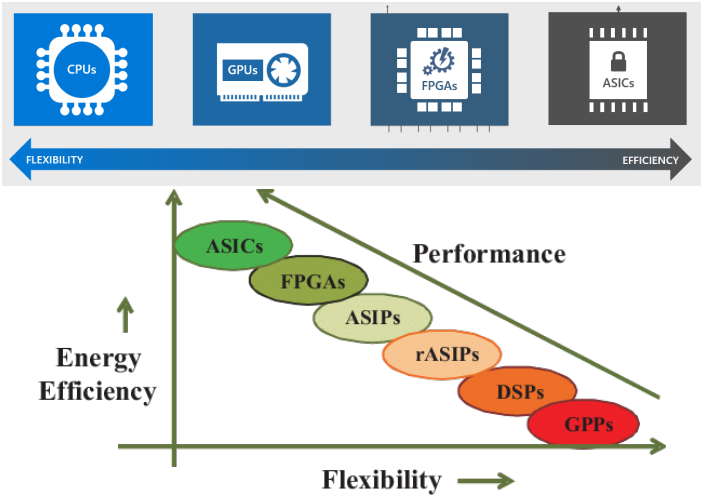
## Implementation Types






2

# Implementation Types






3

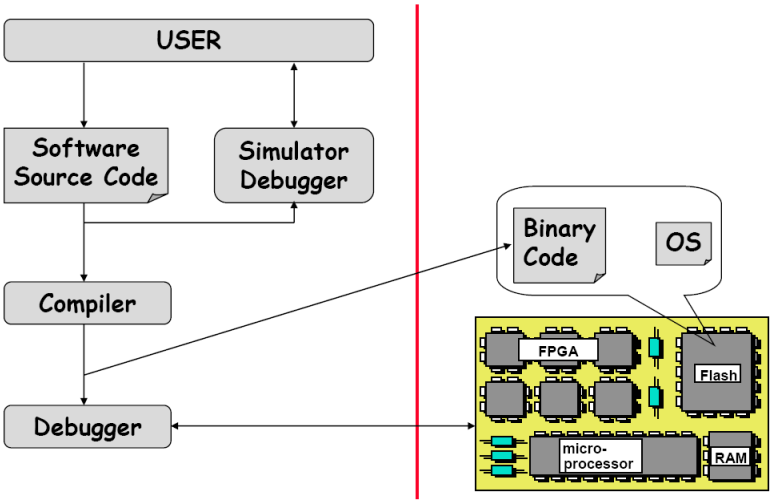
# Processor Spectrum

- General purpose microprocessor
  - Workstations, PCs, high-performance applications, …
- Microcontroller (MCU)
  - Embedded systems, control systems, …
- Digital signal processors
  - Signal processing applications, …
- Special purpose (co-)processor
  - Security or image processing applications, ...
- FPGA/ASIC-based processors
  - Very dedicated scenarios (engine control, cell phones, cryptocurrency mining, …)



4

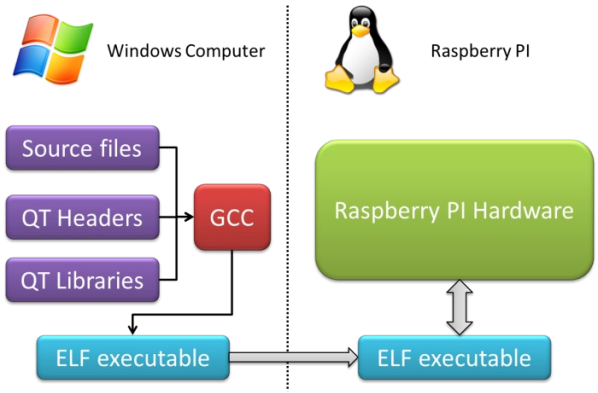
# Cross Compilation






5

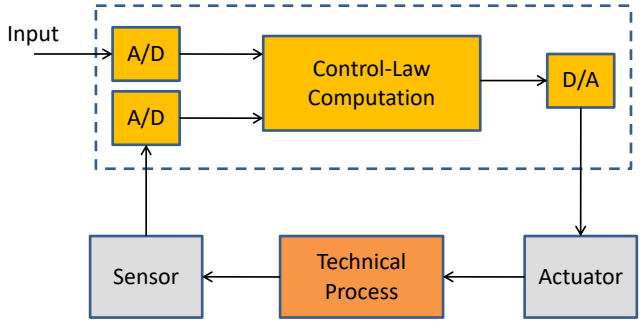
# Example






6

# Generic Real-Time Control System

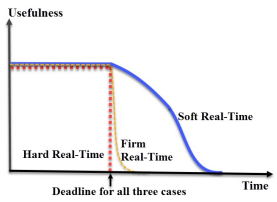





7

# Real-Time Systems

- A real-time system (RTS) is a system that must satisfy explicit (bounded) response-time constraints or risk severe consequences, including failure.
  - Soft RTS: Miss deadline(s)-> performance degraded
  - Hard RTS: Miss deadline(s)-> system failure
  - Firm RTS: Miss deadline(s)-> some tolerance




Who decides deadlines?



8

# Real-Time Systems

| System | Real-Time Classification | Explanation |
|---|---|---|
| Avionics weapons delivery system in which pressing a button launches an air-to-air missile | Hard | Missing the deadline to launch the missile within a specified time after pressing the button may cause the target to be missed, which will result in a catastrophe |
| Navigation controller for an autonomous weed-killer robot | Firm | Missing a few navigation deadlines causes the robot to veer out from a planned path and damage some crops |
| Console hockey game | Soft | Missing even several deadlines will only degrade performance |





# Schedule

- Given a set of tasks *J = {J1, J2,...}* :
  - A schedule is an assignment of tasks to the processor, such that each task is executed until completion.
  - A schedule can be defined as an integer step function σ : *R → {0,1,2,..,n}* where σ(*t)* is the task executed at time *t;* if σ(*t) = 0* then the processor is called idle.
  - If σ(*t)* changes its value at some time, then the processor performs a context switch**.**
  - Each interval, where σ (*t)* is constant, is called a time slice**.**
  - A preemptive schedule is a schedule where the running task can be arbitrarily suspended at any time, to assign the CPU to another task according to a predefined scheduling policy.

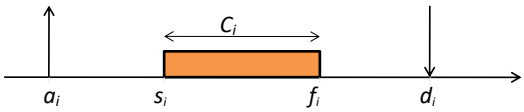
# Schedule & Timing

- A schedule is said to be feasible, if all tasks can be completed according to a set of specified constraints.
- A set of tasks is said to be schedulable with algorithm A, if A can produce a feasible schedule.
- Arrival time $a_i$ or request or release time $r_i$ is the time at which a task becomes ready for execution.
- Computation time $C_i$ is the time necessary to the processor for executing the task without interruption.
- Deadline $d_i$ is the time by which a task should be completed.
- Start time $s_i$ is the time by which a task starts its execution.
- Finishing time $f_i$ is the time by which a task finishes its execution.



11

# Schedule & Timing



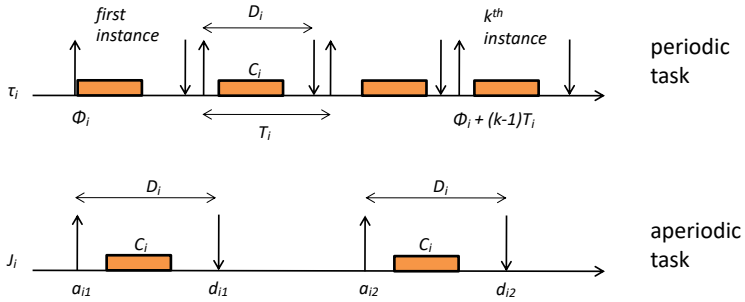

- Using the above definitions, we have $d_i \geq a_i + C_i$
- Lateness $L_i = f_i - d_i$ represents the delay of a task completion with respect to its deadline; note that if a task completes before the deadline, its lateness is negative.
- Tardiness or exceeding time $E_i = max(0, L_i)$ is the time a task stays active after its deadline.
- Laxity or slack time $X_i = d_i - a_i - C_i$ is the maximum time a task can be delayed after its activation to still complete by its deadline.



12

# Schedule & Timing

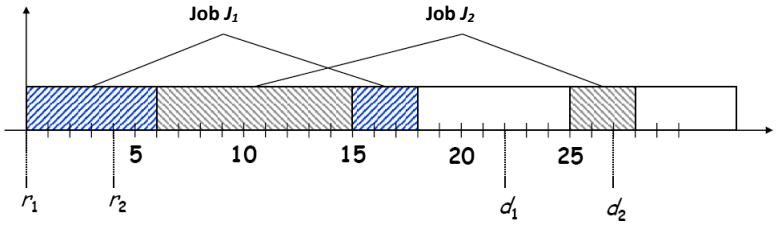
- Periodic task $\tau_i$: infinite sequence of identical activities, called instances or jobs, that are regularly activated at a constant rate with period $T_i$. The activation time of the first instance is called phase $\Phi_i$.








# Example




- Computation times: $C_1 = 9$, $C_2 = 12$
- Start times: $s_1 = 0$, $s_2 = 6$
- Finishing times: $f_1 = 18$, $f_2 = 28$
- Lateness: $L_1 = -4$, $L_2 = 1$
- Tardiness: $E_1 = 0$, $E_2 = 1$
- Laxity: $X_1 = 13$, $X_2 = 11$

# Worst-Case Execution Time (WCET)

- Analyze and instrument the task
- Analyze the compiler
- Analyze the operating system
- Analyze the hardware

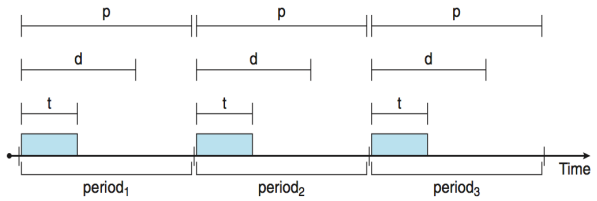
- **Analytical Approach**: all sub-problems are solved analytically.
- **Pragmatic Approach**: investigate and instrument the source program to generate test cases that are biased towards the maximum execution time. Execute the test cases on the target system.





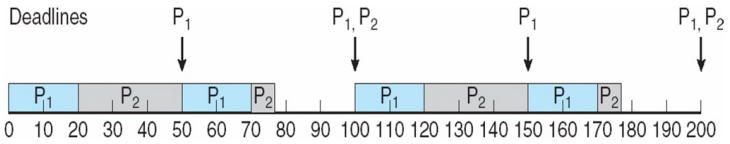
# Priority-Based Scheduling

- Many real-time processes are **periodic**, i.e., they require CPU at constant intervals
  - Has processing time *t*, deadline *d,* period *p*
  - $0 \le t \le d \le p$
  - **Rate** of periodic task is $1/p$



# Rate Montonic Scheduling

- **Priority = the inverse of its period**
- Shorter periods = higher priority
- P1: period of 50 (c=20); P2: period of 100 (c=35)
- $P_1$ is assigned a higher priority than $P_2$

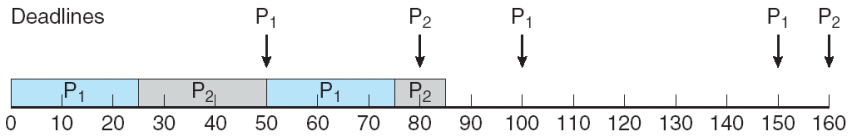






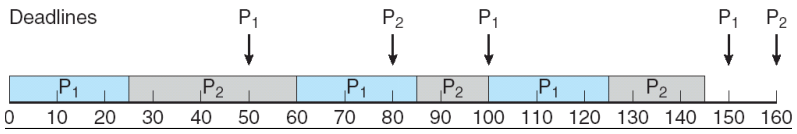

# Missed Deadlines with Rate Monotonic Scheduling

(p, c), p=d
P1(50, 25)
P2(80, 35)



# Earliest Deadline First Scheduling (EDF)

- Priorities are assigned according to **deadlines**:
  - the earlier the deadline, the higher the priority
  - the later the deadline, the lower the priority








# Preemption

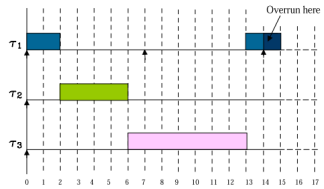



Figure 1: Priorities without preemption

Assume all tasks are released at time 0.

| | *Priority* | *Period* | *Computation time* |
|---|---|---|---|
| $\tau_1$ | 1 | 7 | 2 |
| $\tau_2$ | 2 | 16 | 4 |
| $\tau_3$ | 3 | 31 | 7 |

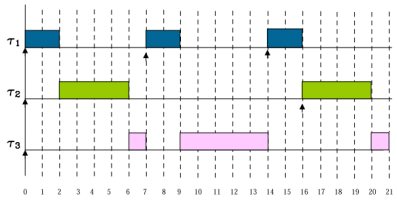



Figure 2: Priorities with preemption

# Utilization

- **Execution time of a task** - time it takes for a task to run to completion
- **Period of a task** - how often a task is being called to execute; can generally assume tasks are periodic although this may not be the case in real-world systems
- **CPU utilization** - the percentage of time that the processor is being used to execute a specific scheduled task

$$U = \frac{e_i}{P_i}$$

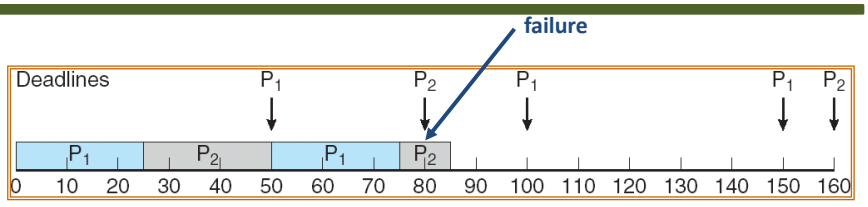
  - where $e_i$ is the execution time of task i, and $P_i$ is its period
- **Total CPU utilization** - the summation of the utilization of all tasks to be scheduled

$$\sum_{i=1}^{n} \frac{e_i}{P_i}$$





# Utilization: RMS




Process $P_1$: service time = 25, period = 50, deadline = 50
Process $P_2$: service time = 35, period = 80, deadline = 80

**RMS is guaranteed to work if**

$N$ = number of processes

**sufficient** condition

$$u = \sum_{i=1}^{N} \frac{t_i}{P_i} \le N\left(\sqrt[N]{2} - 1\right);$$

$$\lim_{N \to \infty} N\left(\sqrt[N]{2} - 1\right) = \ln 2 \approx 0.693147$$

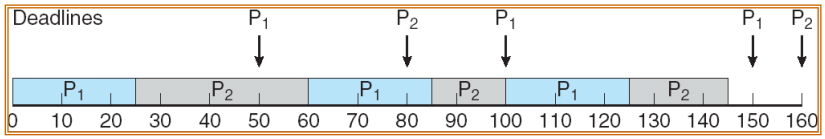
| $N$ | $N\left(\sqrt[N]{2} - 1\right)$ |
|---|---|
| 2 | 0,828427 |
| 3 | 0,779763 |
| 4 | 0,756828 |
| 5 | 0,743491 |
| 10 | 0,717734 |
| 20 | 0,705298 |

# Utilization: EDF

- EDF: shorter absolute deadline → Higher priority
- Utilization bound $U_b = 1$
- $U_b$ is necessary and sufficient

Process $P_1$: service time = 25, period = 50, deadline = 50

Process $P_2$: service time = 35, period = 80, deadline = 80








# EDF vs RMS

- RMS
  - RMS may not guarantee schedulability even when U < 1
  - Low overhead: priorities do not change for a fixed task set
- EDF
  - EDF guarantee schedulability as long as U <= 1
  - High overhead: task priorities may change dynamically