Overview

- Today: this lecture introduces real-time scheduling theory

- **To learn more on real-time scheduling terminology**:
- **see basic concepts on "Hard Real-Time Computing Systems" book from G. Buttazzo** (useful chapters are in the Lab!)
- **see chapter 4 on "Hard Real-Time Computing Systems" book from G. Buttazzo**

- Basic tutorial at:

http://www.embedded.com/electronics-blogs/beginner-s-corner/4023927/Introduction-to-Rate-Monotonic-Scheduling#

So What is a Real-Time System?

- A **real-time system** is a system whose specification includes both logical and temporal correctness requirements.

  - ✓ *Logical Correctness*: produces correct outputs.
  - ✓ *Temporal Correctness*: produces outputs at the right time.

So What is a Real-Time System?

- A real-time system has different set of measure of merits:

	Real time systems	*Non-real time systems*
performance	Schedulable Utilization (schedulability)	*throughput*
responsiveness	Worst case response time of each tasks	*Average response time*
overload	*Stability (getting critical tasks done)*	*Fairness*

What does Real-Time mean?

- The word **time** means that the correctness of the system depends not only on the logical result of the computation but also on the time at which the results are produced.

- The word **real** indicates that the reaction of the system to external events must occur during their evolution. As a consequence, the system time (internal time) must be measured using the same time scale used for measuring the time in the controlled environment (external time).

[in chapter 1 of Buttazzo's book]

Real-Time systems

- Advances in computer hardware will not take care of the temporal requirements needed by a real-time system.

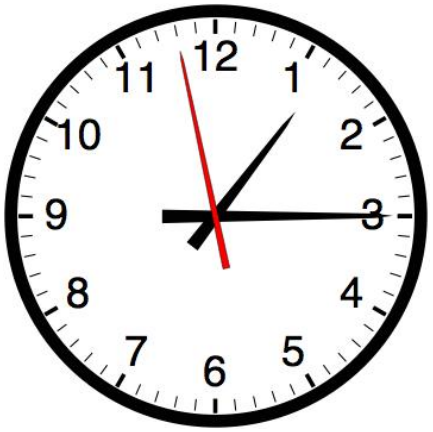
  - ✓ The old ‘‘buy a faster processor’’ argument does not work!
  - ✓ An old computer can be used to run a real-time application
  - ✓ A last generation pc with a general purpose operating system (windows, linux, etc.) can violate the temporal constraints of our real-time application.

- Rather than being fast, a real-time computing system should be predictable.

Are All Systems Real-Time systems?

- Question: is a payroll processing system a real-time system?
  - ✓ It has a time constraint: print the pay checks every two weeks
- Perhaps it is a real-time system in a definitional sense, but it doesn't pay us to view it as such.
- We are interested in systems for which it is not a priori obvious how to meet timing constraints.

Typical Real-Time Systems



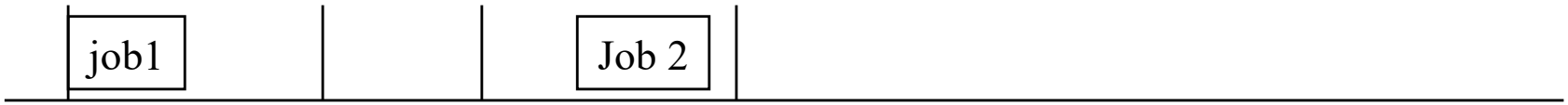
- Cell phones, digital cameras
- Avionic
- Radar Systems
- Factory Process control
- Sensing and Control
- Multi-media systems

- All of them have explicit timing requirements to meet.

Jobs and Tasks

- A job is a unit of computation, e.g.,
  - handling the press of a keyboard
  - or compute the control response in one instance of a control loop

- A task is a sequence of the same type of jobs, say, a control task or the keyboard handling task.



Periodic Task Model

- Periodic tasks are the "work horse" of real-time systems and they play a key role in real-time systems.

- **A task τ_i is said to be periodic if its inter-arrival time (period), p_i, is a constant**

- A periodic task, τ_i, is characterized by
  - Period, p_i
  - Release time, $r_{i,j}$. The default is $r_{i,j} = r_{i,j-1} + p_i$
  - Execution, C_i. The default is worst-case execution time
  - Relative deadline D_i. The default is equal to period
  - Phase ϕ_i : the starting time of the task, i.e., the first release time ($r_{i,1}$).

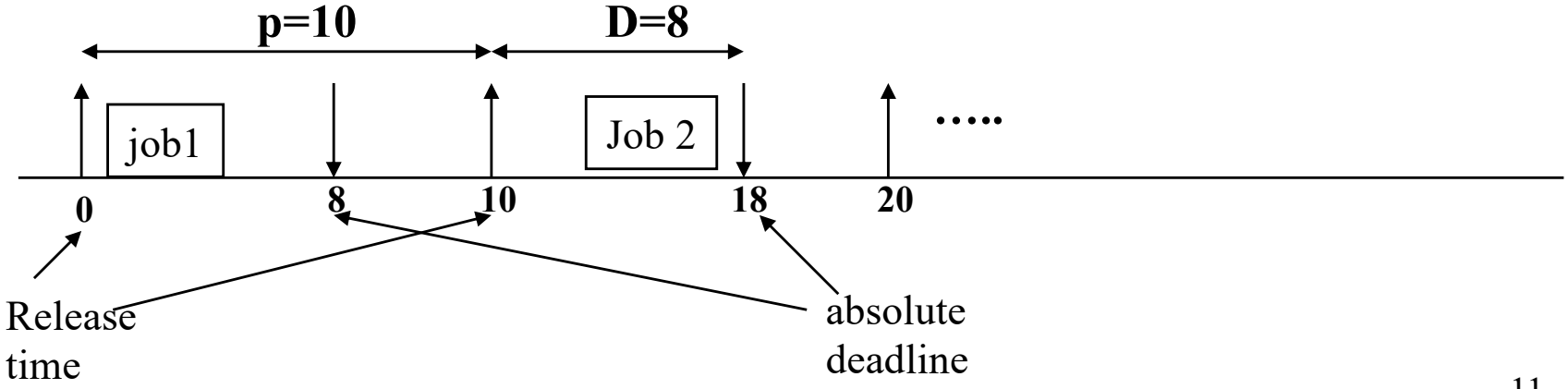
Periodic Tasks

- Periodic tasks are commonly used in embedded real time systems, e.g., a 10Hz task updates the display and a 20 Hz task samples the temperature.

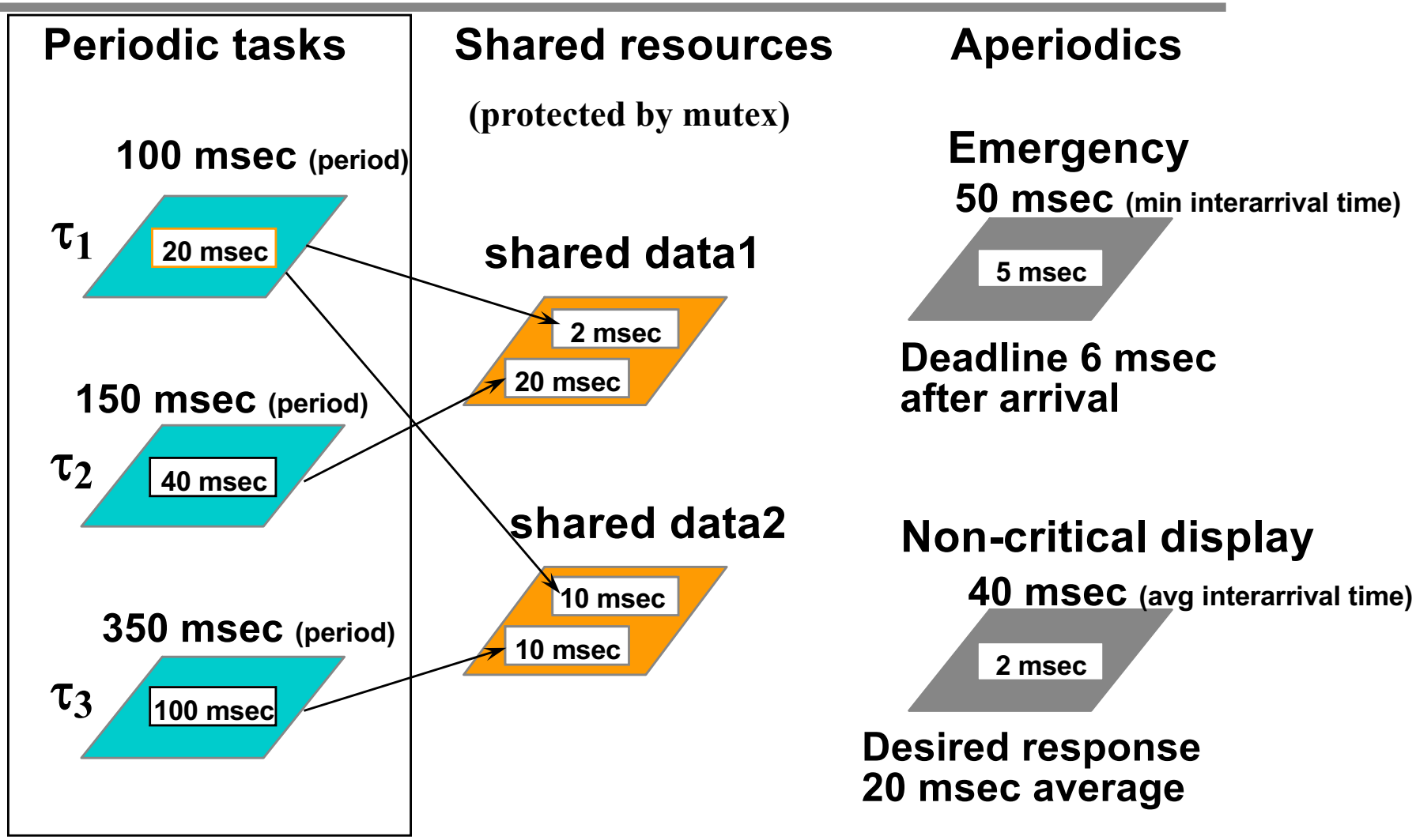
- Rate monotonic scheduling (RMS) algorithm is commonly used in practice. RMS assigns high priorities to tasks with higher rates, e.g., 20 Hz task should be given higher priority than 10 Hz tasks.
  - If every instance of a periodic task has the same priority, it is called a fixed priority scheduling method
  - commercial RTOS support only fixed priority scheduling
  - RMS is an optimal fixed priority scheduling method
  - The timing behavior of a real time system scheduled by RMS, can be fully analyzed and predicted by Rate Monotonic Analysis (RMA).
  - we will study the design and implementation of periodic tasks both on dsPic board and Linux OS

Release Time and Deadlines

- Release time is the instant at which the job becomes ready to execute
- The common form of deadlines are absolute deadlines where deadlines are specified in, well, absolute times. Train and airlines schedules have absolute deadlines.
- Normally, relative deadlines are related to the release time. For example, a relative deadline D=8 msec after the release time.
- The default absolute deadline of a task is the end of period. By convention, we will refer to an absolute deadline as "d", and a relative deadline as "D".



A Sample Problem



Goal: guarantee that no real-time deadline is missed!!!

Real-time scheduling algorithms

- Jobs can be scheduled according to the following scheduling algorithms:

  - **Rate Monotonic (RM)**: the faster the rate, the higher is the priority. All the jobs in a task have the same priority and hence the name "fixed priority" algorithm.

  - **Earliest Deadline First (EDF)**: the job with the earliest deadline has the highest priority. Jobs in a task have different priorities and hence the name, "dynamic priority" algorithm.

Priority and Criticality - 1

- Priority: priority is the order we execute ready jobs.
- Criticality (Importance): represents the penalty if a task misses a deadline (one of its jobs misses a deadline).

- Quiz: Which task should have higher priority?
- Task 1: The most import task in the system: if it does not get done, serious consequences will occur
- Task 2: A mp3 player: if it does not get done in time, the played song will have a glitch

- If it is feasible, we would like to meet the real-time deadlines of both tasks!

Utilization and Schedulability

- A task's utilization (of the CPU) is the fraction of time for which it uses the CPU (over a long interval of time).

- A periodic task's utilization U_i (of CPU) is the ratio between its execution time and period: $U_i = C_i/p_i$

- Given a set of periodic tasks, the total CPU's utilization is equal to the sum of periodic tasks' utilization:

$$U = \sum_i \frac{C_i}{p_i}$$

- Schedulability bound of a scheduling algorithm is the percentage of CPU utilization at or below which a set of periodic tasks can always meet their deadlines. You may think of it as a standard benchmark for the effectiveness of a scheduling algorithm.

- QUIZ: What is the obvious limit on U?

Real-time scheduling algorithms

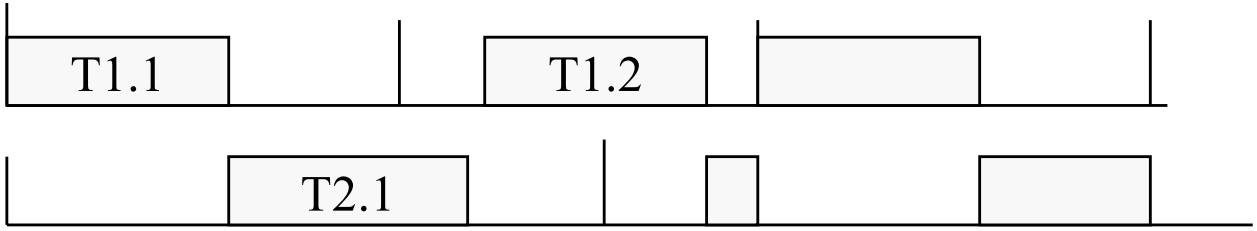
- Scheduling algorithms need to be simple: cannot use many processor cycles

- Static vs. dynamic priorities
  - Static priority: All jobs of a task have the same priority
  - Dynamic priority: Different jobs of the same task may have different priorities

- Examples
  - Rate Monotonic Scheduling [RM]: Tasks with smaller periods are assigned higher priorities (static priority)
  - Earliest Deadline First [EDF]: Jobs are prioritized based on absolute deadlines (dynamic priority)

Dynamic vs "Static" Priority Scheduling in Theory

- *An instance of a task is called a job. "Static" priority assigns a (base) priority to all the jobs in a task. Dynamic priority scheduling adjusts priorities in each task job by job.*



- Quiz: what type of scheduling algorithm is used to schedule these two tasks?

- An optimal dynamic scheduling algorithm is the earlier deadline first (EDF) algorithm. Jobs closer to deadlines will have higher priority. With independent periodic tasks, all tasks will meet their deadlines as long as the processor utilization is less than or equal to 1.

- An optimal "static" scheduling algorithm is the rate monotonic scheduling (RMS) algorithm. For a periodic task, the higher the rate (frequency) the higher the priority.

Importance of the scheduling algorithm

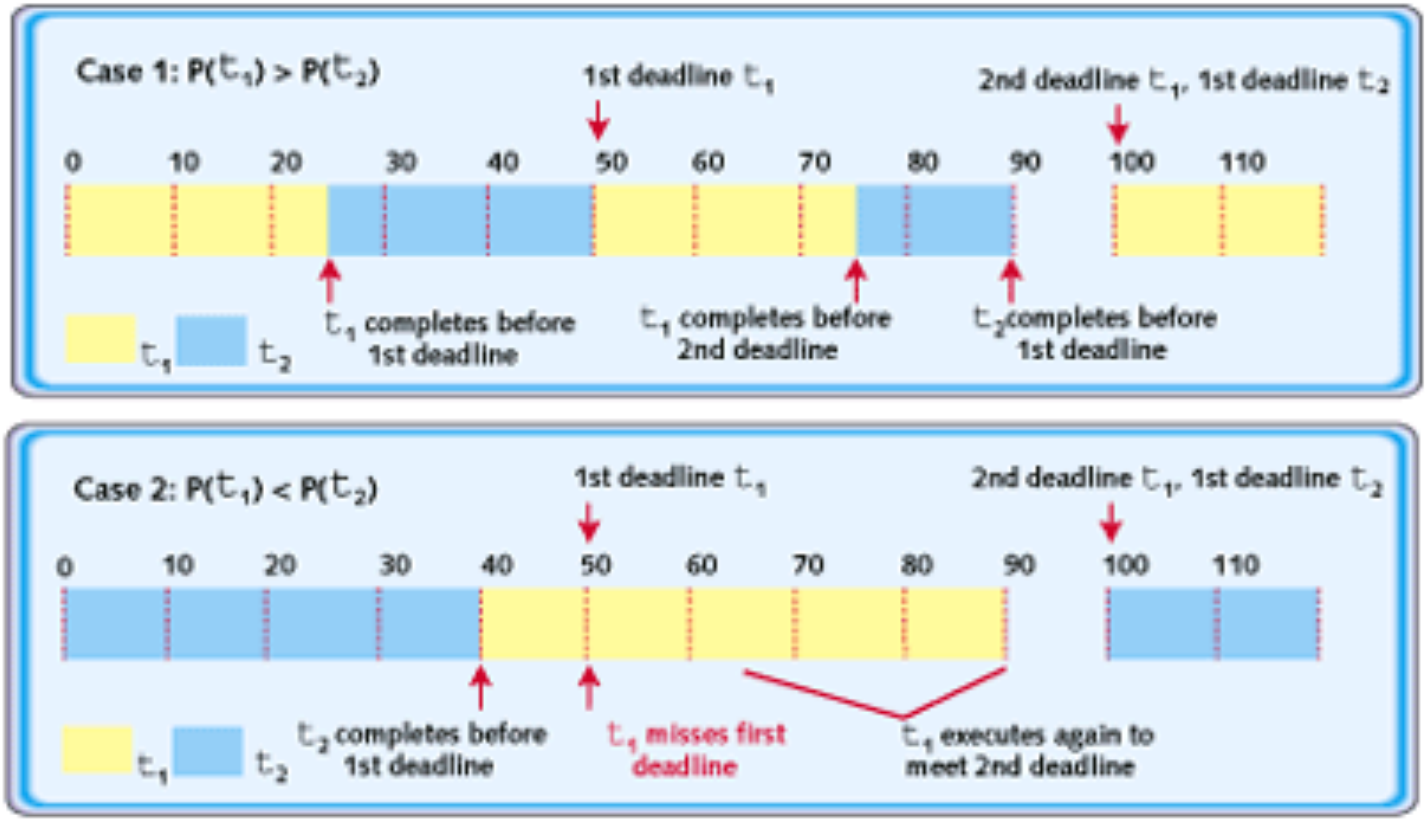
- To demonstrate the importance of a scheduling algorithm, consider a system with only two tasks, T1 and T2. Assume these are both periodic tasks with periods p1 and p2, and each has a deadline that is the beginning of its next cycle.

- Task 1 has p1 = 50 ms, and a worst-case execution time of C1 = 25 ms. Task 2 has p2 = 100 ms and C2 = 40 ms. Note that the utilization, Ui, of task i is Ci/Ti. Thus U1 = 50% and U2 = 40%. This means total requested utilization U = U1 + U2 = 90%. It seems logical that if utilization is less than 100%, there should be enough available CPU time to execute both tasks.

- Let's consider a static priority scheduling algorithm. With two tasks, there are only two possibilities:
  - Case 1: Priority(T1) > Priority(T2)
  - Case 2: Priority(T1) < Priority(T2)

- The two cases are shown in next figure. In Case 1, both tasks meet their respective deadlines. In Case 2, however, Task 1 misses a deadline, despite 10% idle time. **This illustrates the importance of priority assignment.**

Importance of the scheduling algorithm



Both possible outcomes for static-priority scheduling with two tasks (T1=50, C1=25, T2=100, C2=40)

Importance of the scheduling algorithm

- It is theoretically possible for a set of tasks to require just 70% CPU utilization in sum and still not meet all their deadlines. For example, consider the case shown in the Figure below. The only change is that both the period and execution time of Task 2 have been lowered. Based on RM, Task 1 is assigned higher priority. Despite only 90% utilization, Task 2 misses its first deadline. Reversing priorities would not have improved the situation.

Some task sets aren't schedulable (T1=50, C1=25, T2=75, C2=30)

Exercise: try to use EDF and check if the first deadlines are met!

Key scheduling results

- For periodic tasks with relative deadlines equal to their periods:

- Rate monotonic priority assignment is the optimal static priority assignment
  - No other static priority assignment can do better
  - Yet, it cannot achieve 100% CPU utilization

- Earliest deadline first scheduling is the optimal dynamic priority policy
  - EDF can achieve 100% CPU utilization

What is GRMS for?

As the hardware becomes much faster and the RTOS market is gaining wide spread usage, the key required skill is no longer the ability of writing assembly programs that execute on bare custom hardware.

The skills in demand now become the development of large and sophisticated multi-threaded real time applications.

GRMS is a practical theory that will enable you to design and analyze multi-threaded real time applications.

GRMS in The Real World

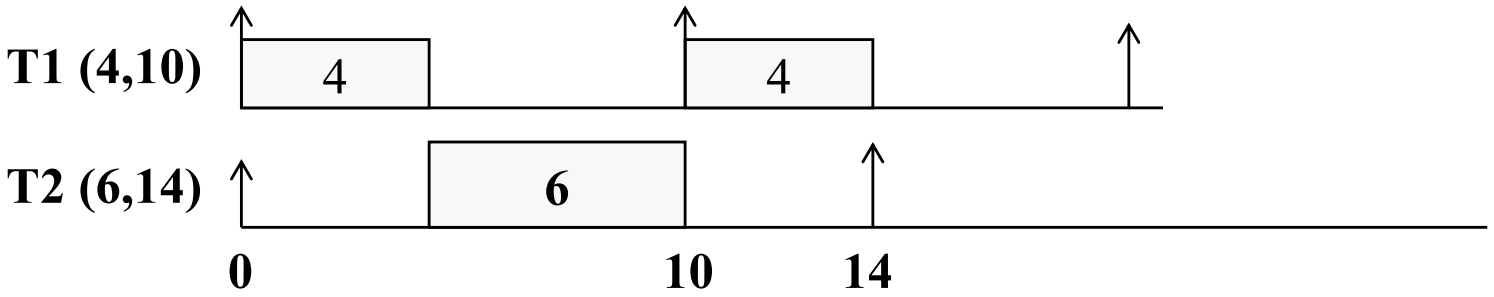
"A major payoff...System designers can use this theory to predict whether task deadlines will be met long before the costly implementation phase of a project begins. It also eases the process of making modifications to application software." DoD *1991 Software Technology Strategy*. pp. 8-15.

"Through the development of [Generalized] Rate Monotonic Scheduling, we now have a system that will allow [Space Station] Freedom's computers to budget their time, to choose between a variety of tasks, and decide not only which one to do first but how much time to spend in the process."
--- Aaron Cohen, former deputy administrator of NASA, "Charting The Future: Challenges and Promises Ahead of Space Exploration", October, 28, 1992, p. 3.

It also provides the theoretical foundation to solve MARS Pathfinder's reset problem while it was on MARS. http://catless.ncl.ac.uk/Risks/19.49.html

GRMS is supported by almost all the commercially available RTOS

RMS: Less Than 100% Utilization but not Schedulable



- **In this example, 2 tasks are scheduled under RMS, an optimal static priority method**
  4/10 + 6/14 = 0.83

- **The task set is schedulable but if we try to increase the computation time of task T1, the task set becomes unschedulable in spite of the fact that total utilization is 83%!**

- **To achieve 100% utilization when using fixed priorities, assign periods so that all tasks are harmonic. This means that for each task, its period is an exact multiple of every other task that has a shorter period.**

- **For example, a three-task set whose periods are 10, 20, and 40, respectively, is harmonic, and preferred over a task set with periods 10, 20, and 50**

The L&L Bound

- A set of n periodic tasks is schedulable if:

$$\frac{c_1}{p_1} + \frac{c_2}{p_2} + \ldots + \frac{c_n}{p_n} \leq n\left(2^{1/n} - 1\right)$$

- U(1) = 1.0 U(4) = 0.756 U(7) = 0.728
- U(2) = 0.828 U(5) = 0.743 U(8) = 0.724
- U(3) = 0.779 U(6) = 0.734 U(9) = 0.720

- For harmonic task sets, the utilization bound is U(n)=1.00 for all n. Otherwise, for large n, the bound converges to *ln 2* ~ 0.69.

- The L&L bound for rate monotonic algorithm is one of the most significant results in real-time scheduling theory. It allows to check the schedulability of a group of tasks with a single test! It is a sufficient condition; hence, it is inconclusive if it fails!

Sample Problem: Applying UB Test

	C	P	U
Task τ1:	**20**	**100**	**0.200**
Task τ2:	**40**	**150**	**0.267**
Task τ3:	**100**	**350**	**0.286**

- Are all the tasks schedulable?
- What if we double the execution time of task τ1?

Sample Problem: draw the schedule by using RM and EDF