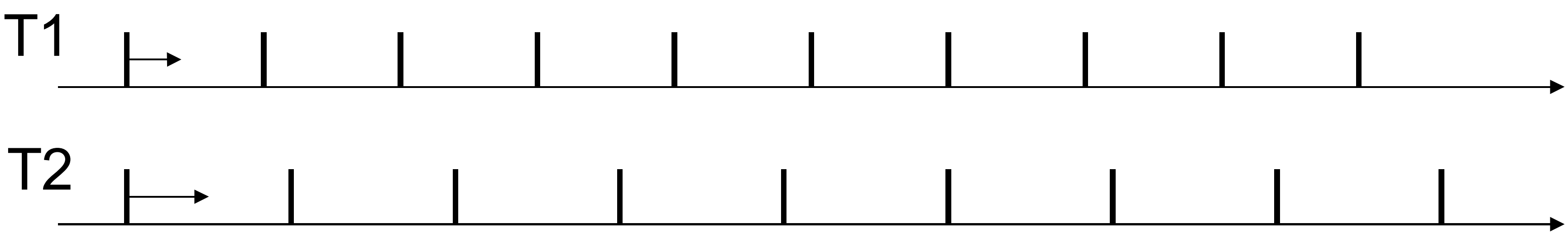
Periodic scheduling

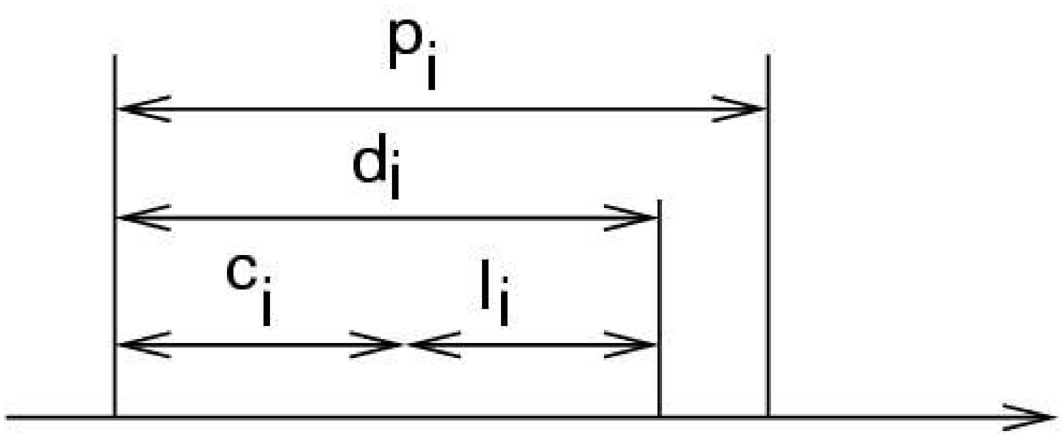


For periodic scheduling, the best that we can do is to design an algorithm which will always find a schedule if one exists.
☞ A scheduler is defined to be **optimal** iff it will find a schedule if one exists.

Periodic scheduling

Let

- p_i be the period of task T_i,
- c_i be the execution time of T_i,
- d_i be the deadline interval, that is, the time between a job of T_i becoming available and the time after which the same job T_i has to finish execution.
- l_i be the **laxity** or **slack**, defined as $l_i = d_i - c_i$



Accumulated utilization

Accumulated utilization:

$$\mu = \sum_{i=1}^{n} \frac{c_i}{p_i}$$

Necessary condition for schedulability (with m=number of processors):

$$\mu \leq m$$

Independent tasks: Rate monotonic (RM) scheduling

Most well-known technique for scheduling independent periodic tasks [Liu, 1973].

Assumptions:

- All tasks that have hard deadlines are periodic.
- All tasks are independent.
- $d_i = p_i$, for all tasks.
- c_i is constant and is known for all tasks.
- The time required for context switching is negligible.
- For a single processor and for n tasks, the following equation holds for the accumulated utilization μ:

$$\mu = \sum_{i=1}^{n} \frac{c_i}{p_i} \leq n(2^{1/n} - 1)$$

Rate monotonic (RM) scheduling - The policy -

RM policy: The priority of a task is a monotonically decreasing function of its period.
At any time, a highest priority task among all those that are ready for execution is allocated.

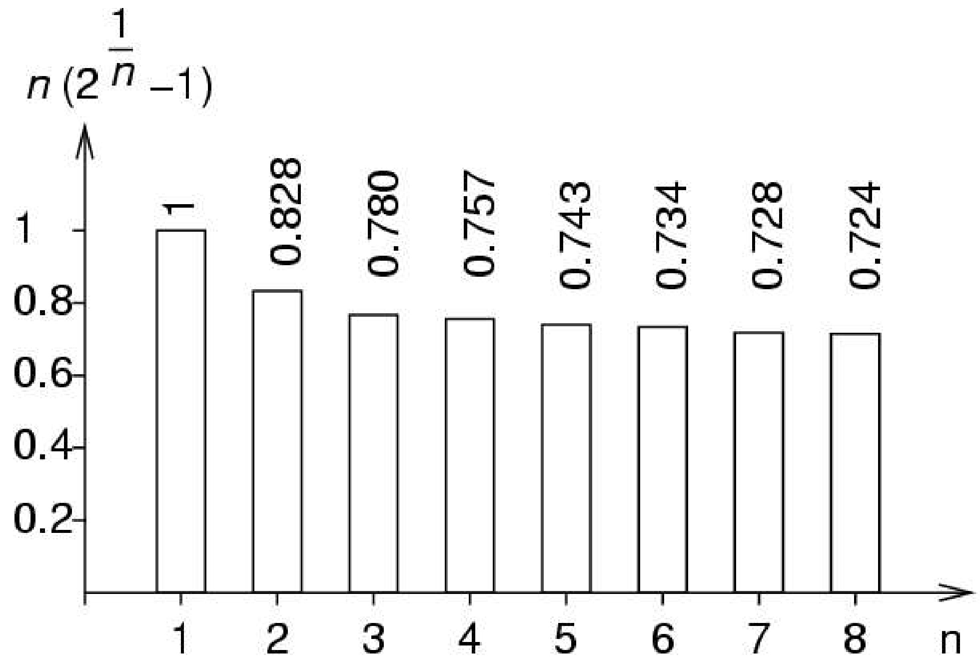
Theorem: If all RM assumptions are met, schedulability is guaranteed.

Maximum utilization for guaranteed schedulability

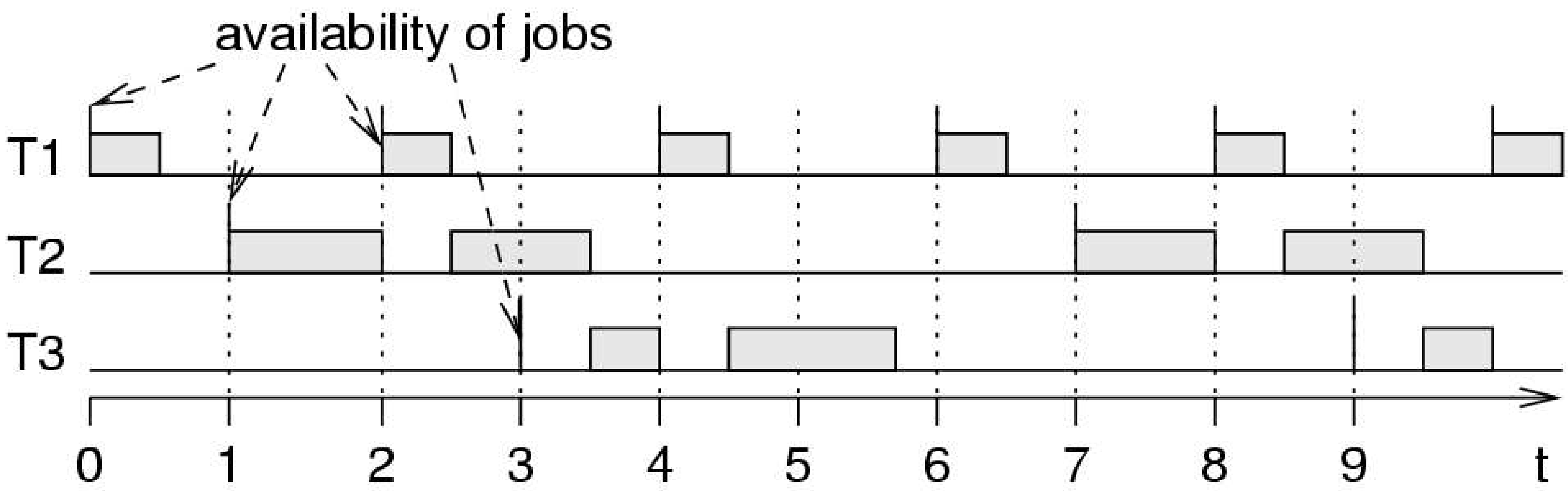
Maximum utilization as a function of the number of tasks:

$$\mu = \sum_{i=1}^{n} \frac{c_i}{p_i} \le n(2^{1/n} - 1)$$

$$\lim_{n \to \infty}(n(2^{1/n} - 1) = \ln(2)$$



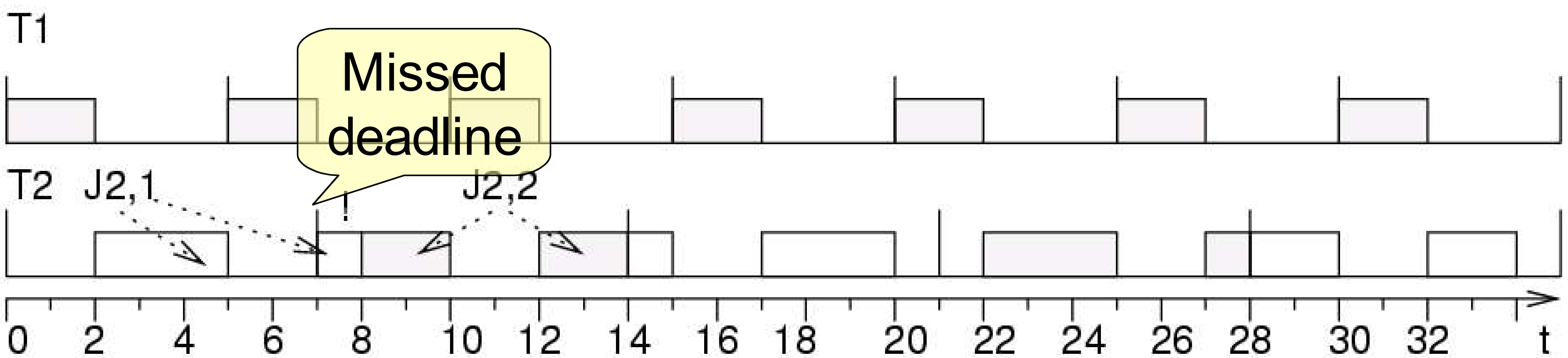
Example of RM-generated schedule



T1 preempts T2 and T3.
T2 and T3 do not preempt each other.

Case of failing RM scheduling

Task 1: period 5, execution time 2
Task 2: period 7, execution time 4
$\mu = 2/5 + 4/7 = 34/35 \approx 0.97$
$2(2^{1/2}-1) \approx 0.828$



Missing computations scheduled in the next period

More in-depth:

Proof of RM optimality

Definition: A **critical instant** of a task is the time at which the release of a task will produce the largest response time.

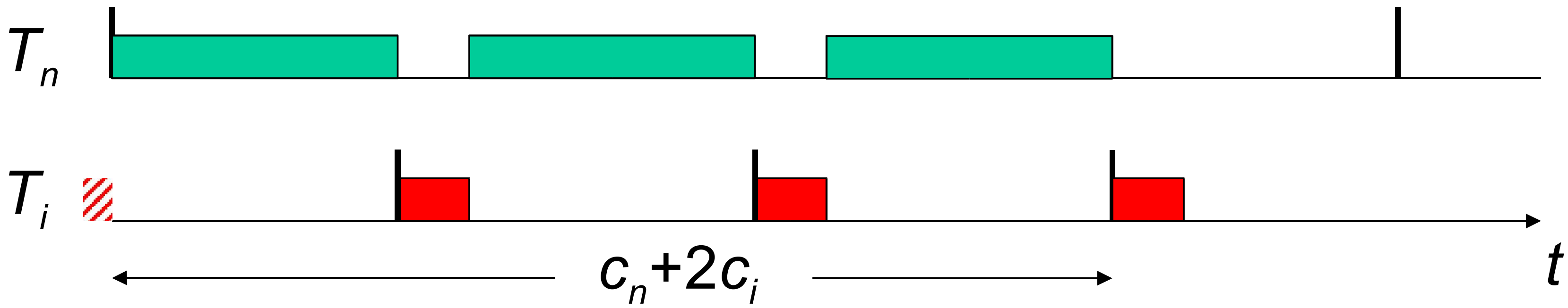
Lemma: For any task, the **critical instant** occurs if that task is simultaneously released with all higher priority tasks.

Proof: Let $T=\{T_1, \ldots, T_n\}$: periodic tasks with $\forall i$: $p_i \leqq p_{i+1}$.

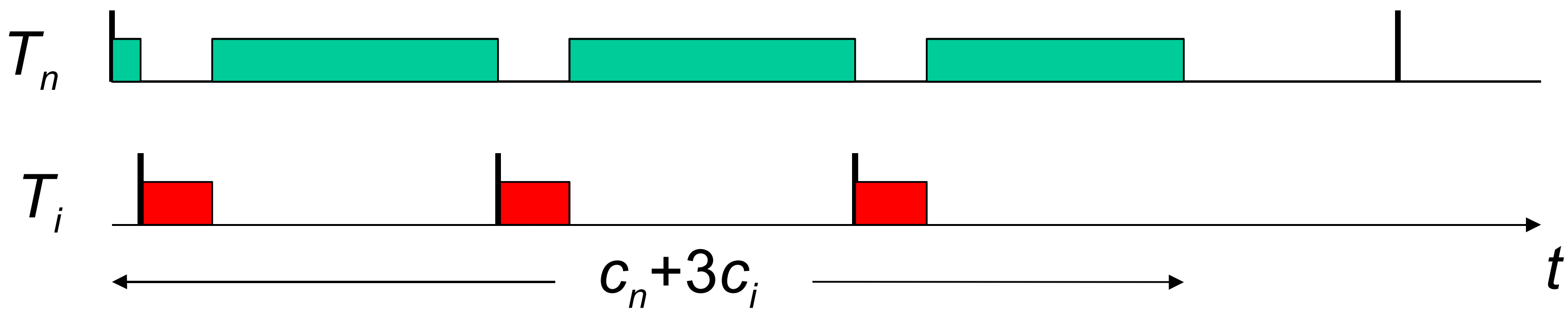
Source: G. Buttazzo, Hard Real-time Computing Systems, Kluwer, 2002

Critical instances (1)

Response time of T_n is delayed by tasks T_i of higher priority:



Delay may increase if T_i starts earlier



Maximum delay achieved if T_n and T_i start simultaneously.

Critical instances (2)

Repeating the argument for all $i = 1, \ldots n-1$:

☞The worst case response time of a task occurs when it is released simultaneously with all higher-priority tasks. q.e.d.

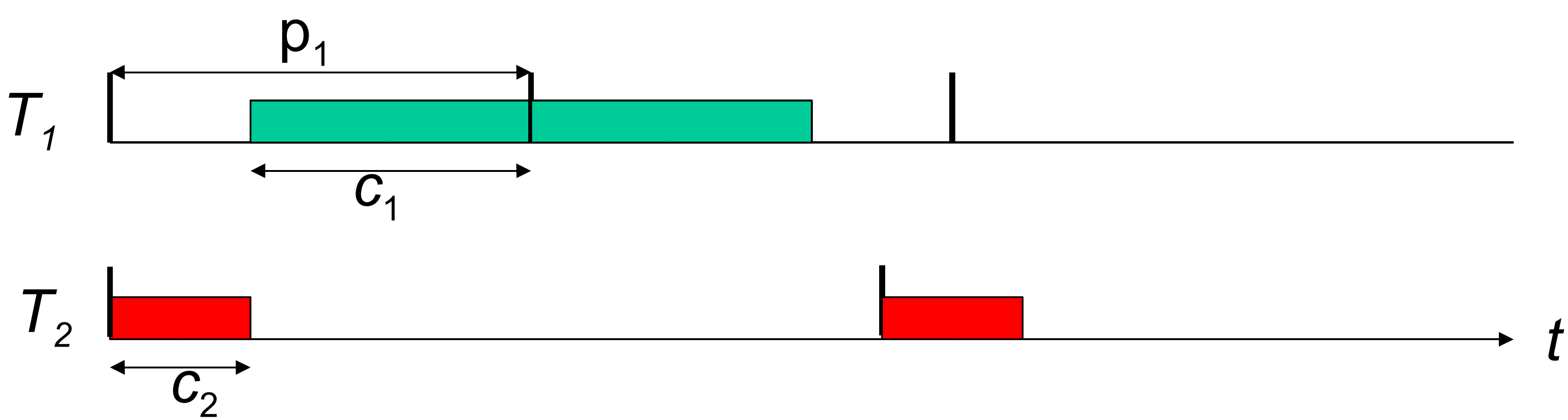
☞Schedulability is checked at the critical instants.

☞If all tasks of a task set are schedulable at their critical instants, they are schedulable at all release times.

Proof of the RM theorem

Let $T=\{T_1, T_2\}$ with $p_1 < p_2$.

Assume RM is **not** used → prio(T_2) is highest:



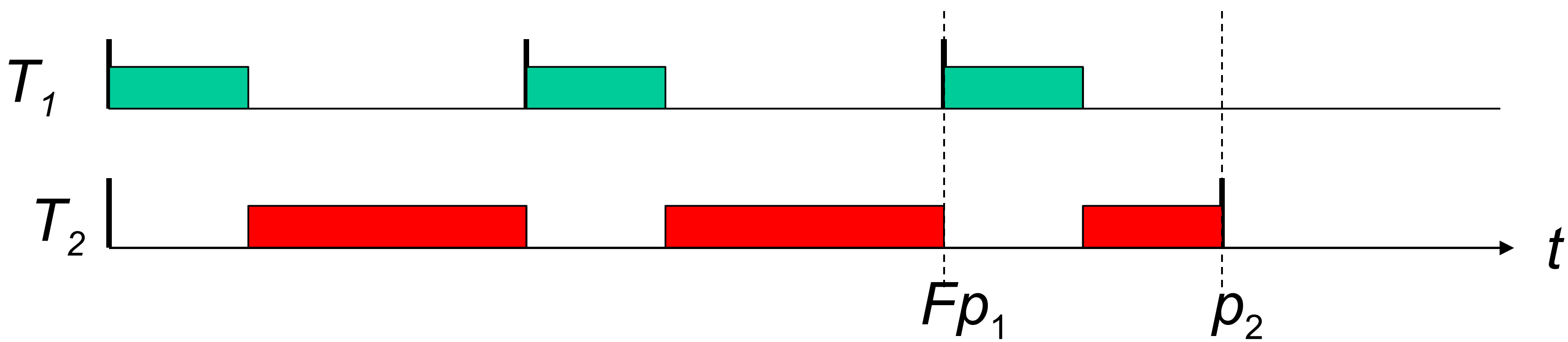
Schedule is feasible if $c_1+c_2 \leqq p_1$ (1)

Define $F=\lfloor p_2/p_1 \rfloor$: # of periods of T_1 fully contained in T_2

Case 1: $c_1 \leq p_2 - Fp_1$

Assume RM is used → prio(T_1) is highest:

Case 1*: $c_1 \leq p_2 - Fp_1$
(c_1 small enough to be finished before 2nd instance of T_2)



Schedulable if $(F+1)\ c_1 + c_2 \leq p_2$ (2)

* Typos in [Buttazzo 2002]: < and ≤ mixed up]

Proof of the RM theorem (3)

Not RM: schedule is feasible if $c_1+c_2 \leq p_1$ (1)

RM: schedulable if $(F+1)\ c_1 + c_2 \leq p_2$ (2)

From (1): $Fc_1+Fc_2 \leq Fp_1$

Since $F \geq 1$: $Fc_1+c_2 \leq Fc_1+Fc_2 \leq Fp_1$

Adding c_1: $(F+1)c_1+c_2 \leq Fp_1 +c_1$

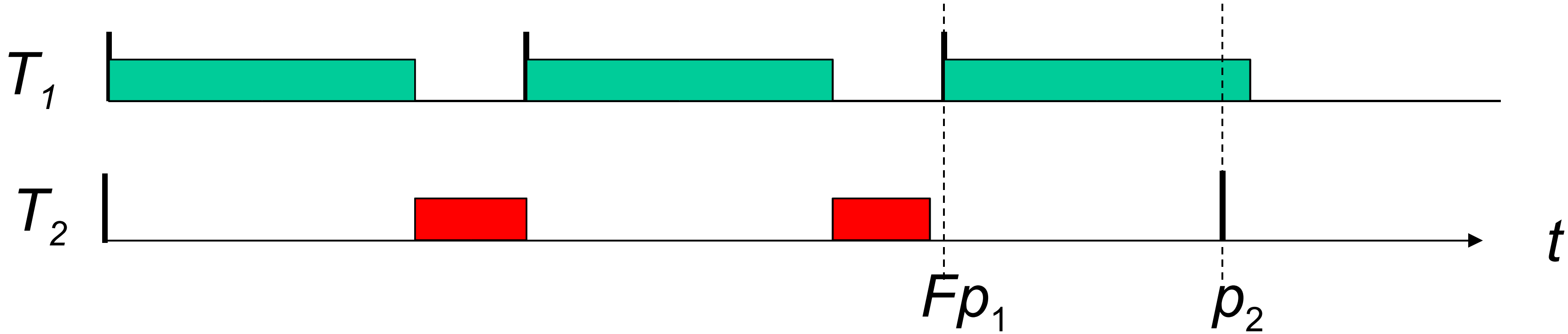
Since $c_1 \leq p_2 - Fp_1$: $(F+1)c_1+c_2 \leq Fp_1 +c_1 \leq p_2$

Hence: if (1) holds, (2) holds as well

☞ For case 1: Given tasks T_1 and T_2 with $p_1 < p_2$, then if the schedule is feasible by an arbitrary (but fixed) priority assignment, it is also feasible by RM.

Case 2: $c_1 > p_2 - Fp_1$

Case 2: $c_1 > p_2 - Fp_1$
(c_1 large enough not to finish before 2nd instance of T_2)



Schedulable if $$F\,c_1 + c_2 \le F\,p_1 \quad (3)$$

$$c_1 + c_2 \le p_1 \quad (1)$$

Multiplying (1) by F yields $$F\,c_1 + F\,c_2 \le F\,p_1$$

Since F ≥ 1: $$F\,c_1 + c_2 \le F\,c_1 + Fc_2 \le F\,p_1$$

☞ Same statement as for case 1.

Calculation of the least upper utilization bound

Let $T=\{T_1, T_2\}$ with $p_1 < p_2$.

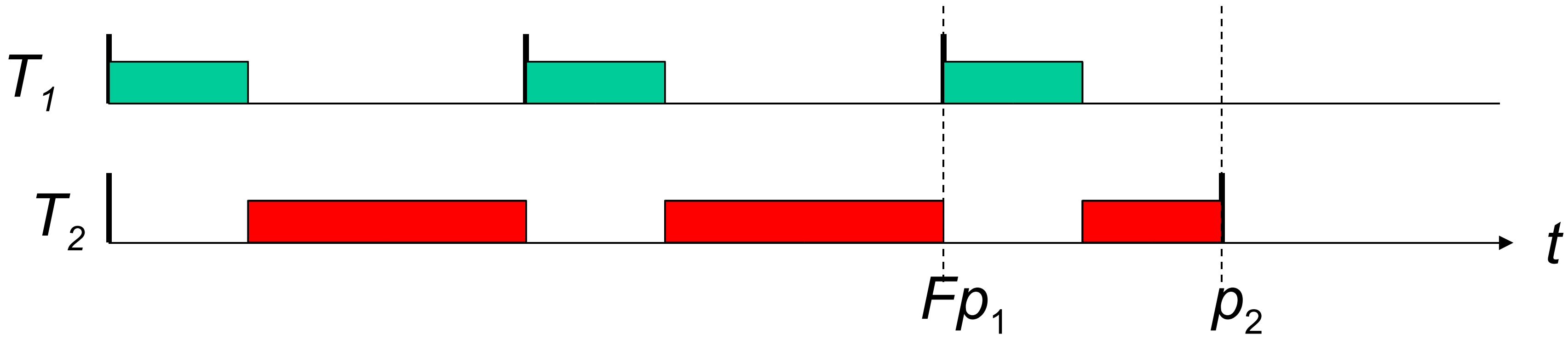
Proof procedure: compute least upper bound U_{lup} as follows

- Assign priorities according to RM
- Compute upper bound U_{up} by setting computation times to fully utilize processor
- Minimize upper bound with respect to other task parameters

As before: $F=\lfloor p_2/p_1 \rfloor$

c_2 adjusted to fully utilize processor.

Case 1: $c_1 \leq p_2 - Fp_1$



Largest possible value of c_2 is $c_2 = p_2 - c_1\,(F+1)$

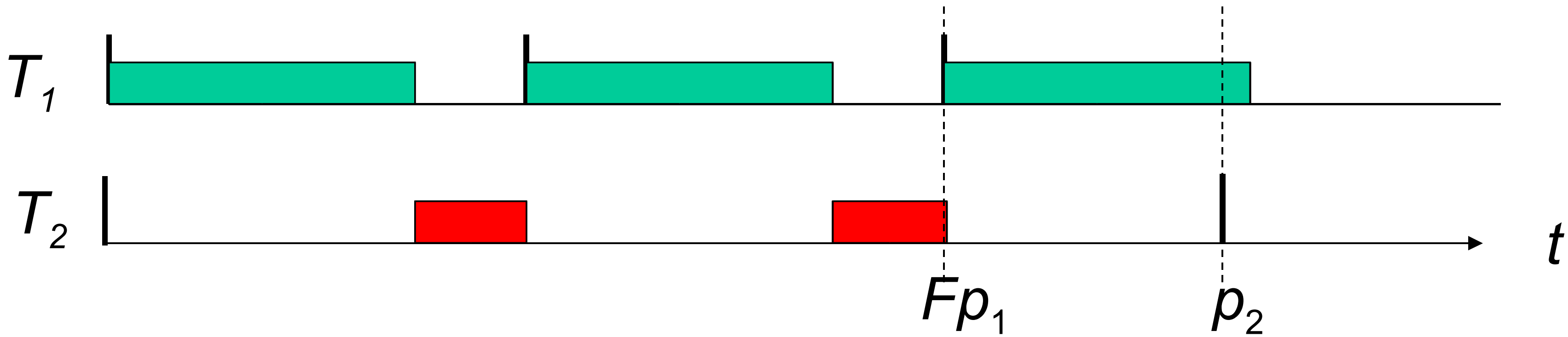
Corresponding upper bound is

$$U_{ub} = \frac{c_1}{p_1} + \frac{c_2}{p_2} = \frac{c_1}{p_1} + \frac{p_2 - c_1\,(F+1)}{p_2} = 1 + \frac{c_1}{p_1} - \frac{c_1\,(F+1)}{p_2} = 1 + \frac{c_1}{p_2}\left\{\frac{p_2}{p_1} - (F+1)\right\}$$

{ } is <0 → U_{ub} monotonically decreasing in c_1

Minimum occurs for $c_1 = p_2 - Fp_1$

Case 2: $c_1 \geq p_2 - Fp_1$



Largest possible value of c_2 is $c_2 = (p_1 - c_1)F$

Corresponding upper bound is:

$$U_{ub} = \frac{c_1}{p_1} + \frac{c_2}{p_2} = \frac{c_1}{p_1} + \frac{(p_1 - c_1)\,F}{p_2} = \frac{p_1}{p_2}F + \frac{c_1}{p_1} - \frac{c_1}{p_2}F = \frac{p_1}{p_2}F + \frac{c_1}{p_2}\left\{\frac{p_2}{p_1} - F\right\}$$

{ } is ≥ 0 → U_{ub} monotonically increasing in c_1 (independent of c_1 if {}=0)

Minimum occurs for $c_1 = p_2 - Fp_{1,}$ as before.

Utilization as a function of $G=p_2/p_1-F$

For minimum value of c_1:

$$U_{ub} = \frac{p_1}{p_2}F + \frac{c_1}{p_2}\left(\frac{p_2}{p_1} - F\right) = \frac{p_1}{p_2}F + \frac{(p_2 - p_1F)}{p_2}\left(\frac{p_2}{p_1} - F\right) = \frac{p_1}{p_2}\left\{F + \left(\frac{p_2}{p_1} - F\right)\left(\frac{p_2}{p_1} - F\right)\right.$$

Let $G = \frac{p_2}{p_1} - F; \qquad \Rightarrow$

$$U_{ub} = \frac{p_1}{p_2}\left(F + G^2\right) = \frac{\left(F + G^2\right)}{p_2 / p_1} = \frac{\left(F + G^2\right)}{\left(p_2 / p_1 - F\right) + F} = \frac{\left(F + G^2\right)}{F + G} = \frac{(F + G) - (G - G^2)}{F + G}$$

$$= 1 - \frac{G(1 - G)}{F + G}$$

Since $0 \leq G < 1$: $\quad G(1-G) \geq 0 \quad \rightarrow \quad U_{ub}$ increasing in F $\rightarrow$

Minimum of U_{ub} for min(F): $\quad F=1$ $\rightarrow$

$$U_{ub} = \frac{1 + G^2}{1 + G}$$

Proving the RM theorem for *n*=2

end

$$U_{ub} = \frac{1+G^2}{1+G}$$

Using derivative to find minimum of U_{ub} :

$$\frac{dU_{ub}}{dG} = \frac{2G(1+G)-(1+G^2)}{(1+G)^2} = \frac{G^2+2G-1}{(1+G)^2} = 0$$

$$G_1 = -1-\sqrt{2}; \qquad G_2 = -1+\sqrt{2};$$

Considering only G_2, since $0 \le G < 1$:

$$U_{lub} = \frac{1+(\sqrt{2}-1)^2}{1+(\sqrt{2}-1)} = \frac{4-2\sqrt{2}}{\sqrt{2}} = 2(\sqrt{2}-1) = 2(2^{\frac{1}{2}}-1) \cong 0.83$$

This proves the RM theorem for the special case of *n*=2

Properties of RM scheduling

- From the proof, it is obvious that no idle capacity is needed if $p_2 = F\ p_1$. In general: not required if the period of all tasks is a multiple of the period of the highest priority task, that is, schedulability is then also guaranteed if $\mu \leq 1$.
- RM scheduling is based on **static** priorities. This allows RM scheduling to be used in standard OS, such as Windows NT.
- A huge number of variations of RM scheduling exists.
- In the context of RM scheduling, many formal proofs exist.

EDF

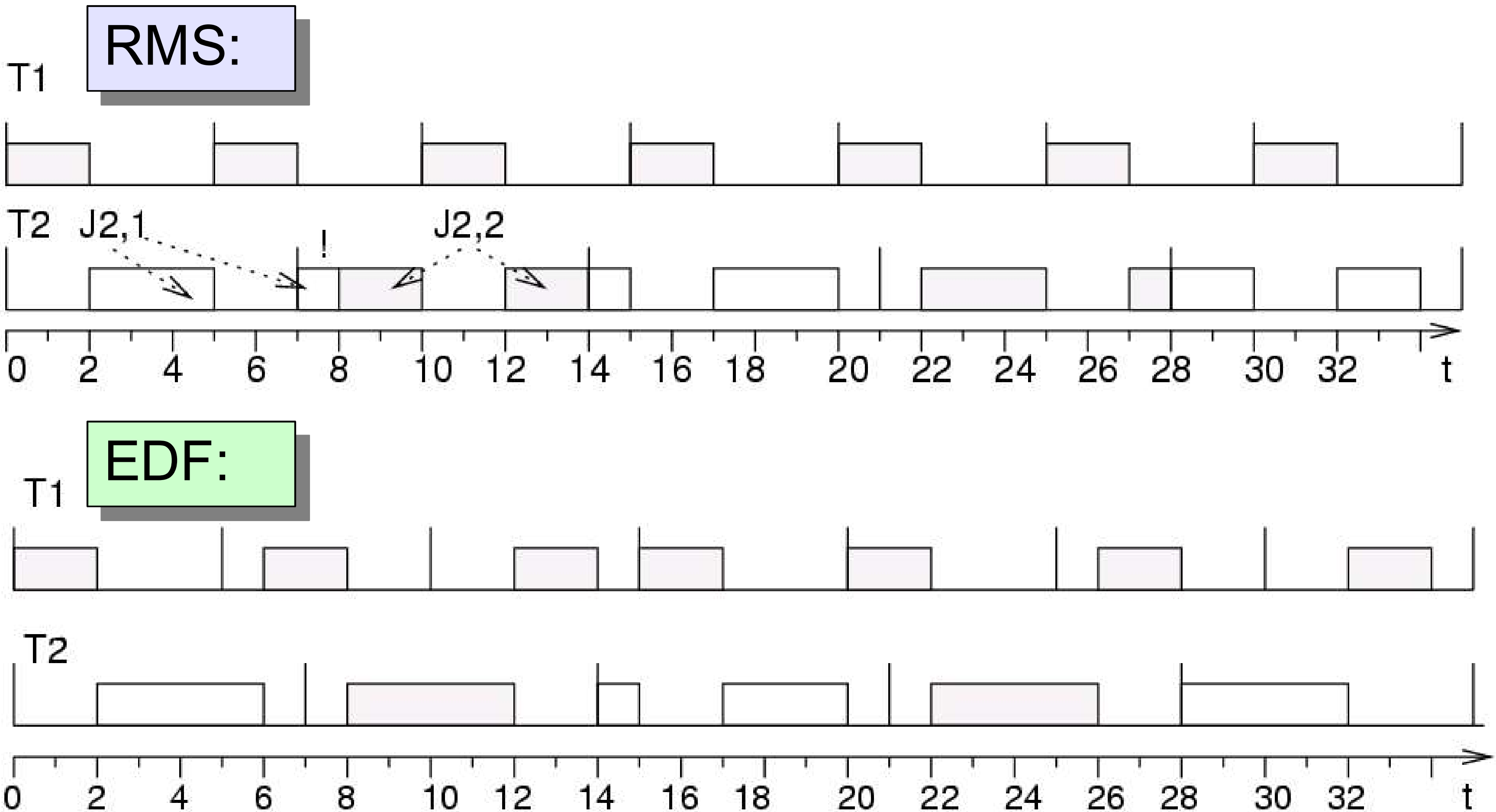
EDF can also be applied to periodic scheduling.

EDF optimal for every period

☞ optimal for periodic scheduling

☞ EDF must be able to schedule the example in which RMS failed.

Comparison EDF/RMS



T2 not preempted, due to its earlier deadline.

EDF: Properties

EDF requires dynamic priorities

☞ EDF cannot be used with a standard operating system just providing static priorities.

Dependent tasks

The problem of deciding whether or not a schedule exists for a set of dependent tasks and a given deadline is NP-complete in general [Garey/Johnson].

Strategies:

2. Add resources, so that scheduling becomes easier
3. Split problem into static and dynamic part so that only a minimum of decisions need to be taken at run-time.

Sporadic tasks

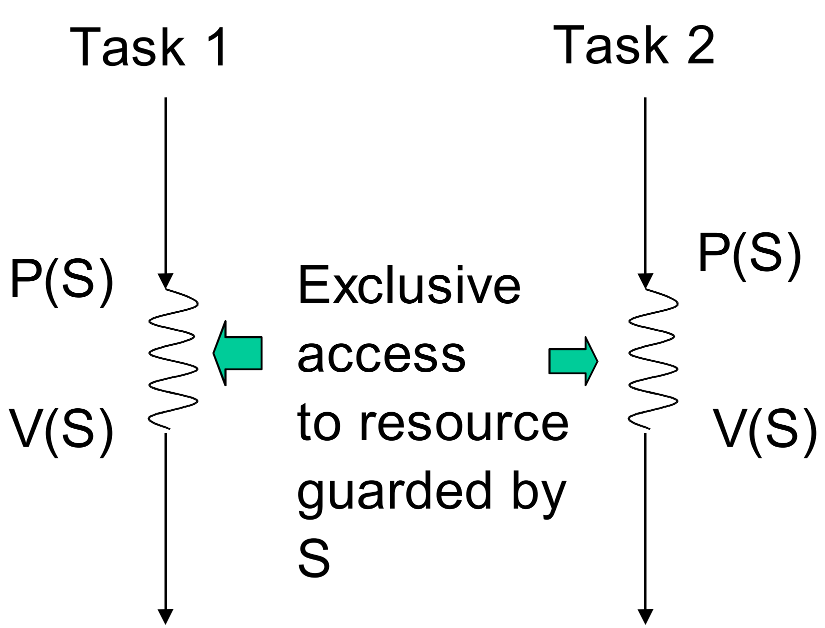
If sporadic tasks were connected to interrupts, the execution time of other tasks would become very unpredictable.

☞ Introduction of a sporadic task server, periodically checking for ready sporadic tasks;

☞ Sporadic tasks are essentially turned into periodic tasks.

Resource access protocols

Critical sections: sections of code at which exclusive access to some resource must be guaranteed. Can be guaranteed with semaphores S.



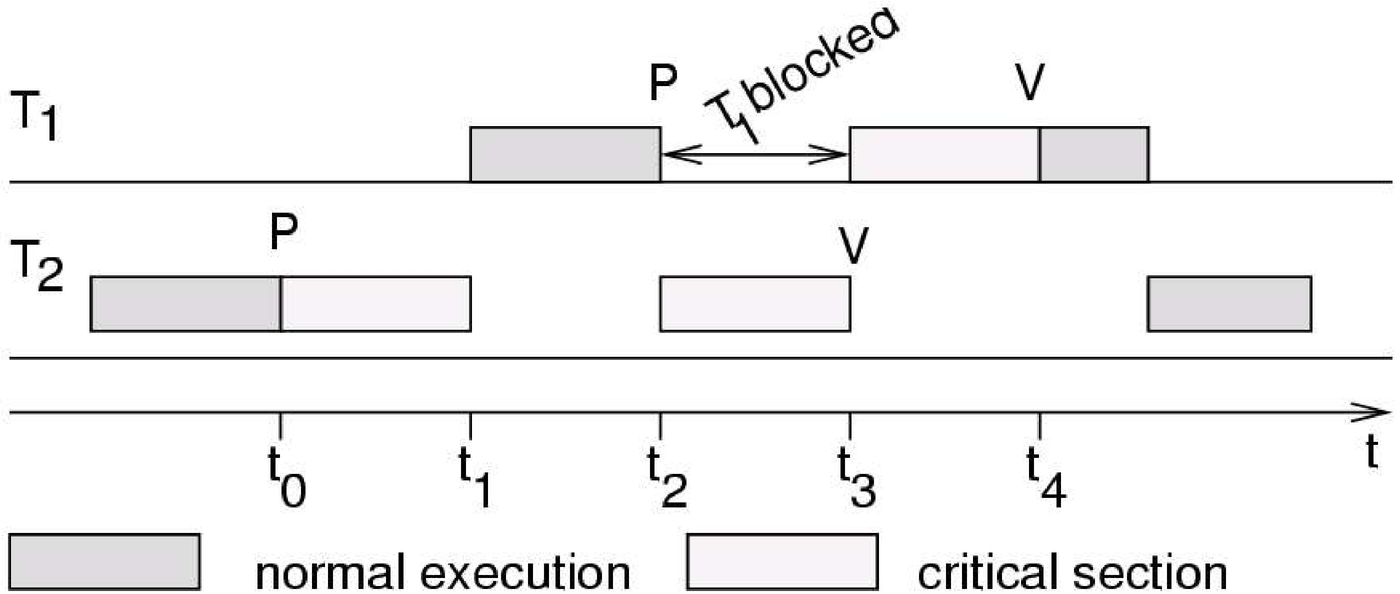
P(S) checks semaphore to see if resource is available and if yes, sets S to „used". Uninterruptable operations! If no, calling task has to wait.

V(S): sets S to „unused" and starts sleeping task (if any).

Priority inversion

Priority T_1 assumed to be > than priority of T_2.

If T_2 requests exclusive access first (at t_0), T_1 has to wait until T_2 releases the resource (time t_3), thus inverting the priority:



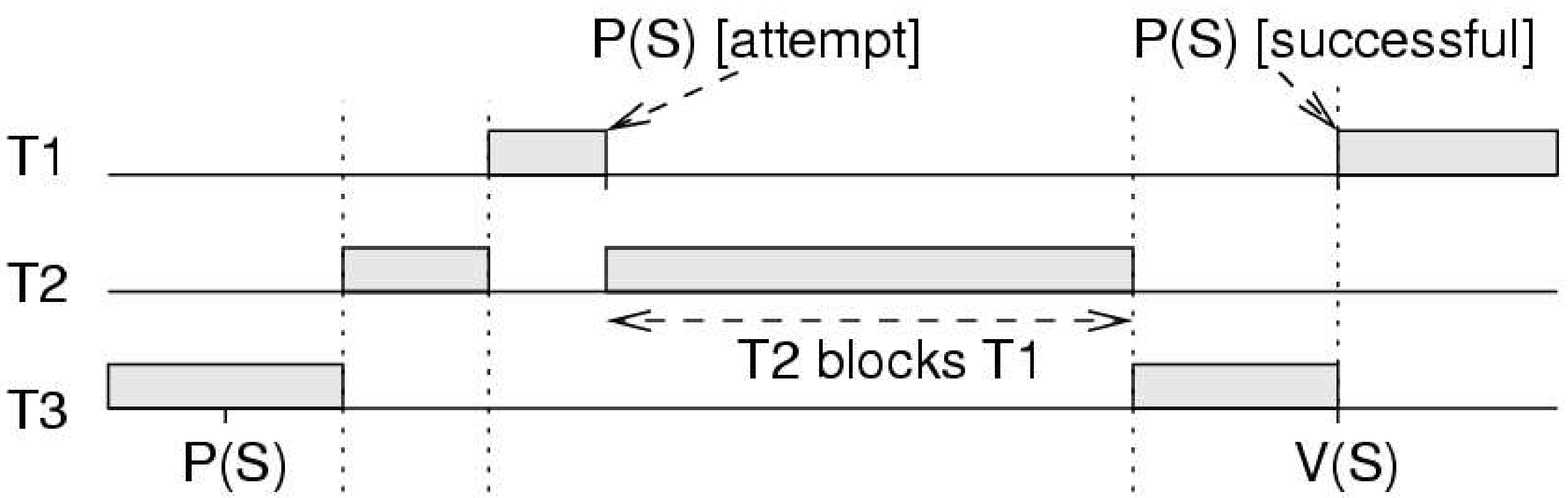
In this example:
duration of inversion bounded by length of critical section of T_2.

Duration of priority inversion with >2 tasks can exceed the length of any critical section

Priority of T1 > priority of T2 > priority of T3.

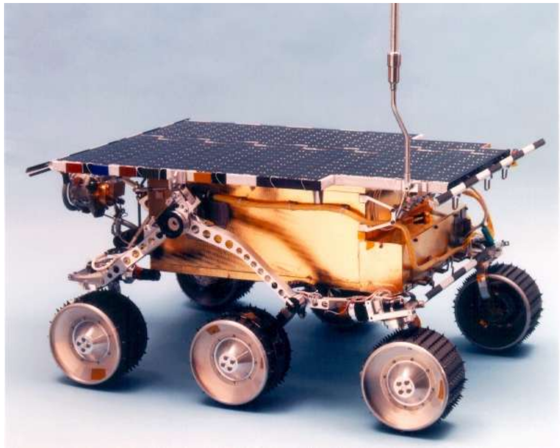
T2 preempts T3:

T2 can prevent T3 from releasing the resource.



The MARS Pathfinder problem (1)

"But a few days into the mission, not long after Pathfinder started gathering meteorological data, the spacecraft began experiencing total system resets, each resulting in losses of data. The press reported these failures in terms such as "software glitches" and "the computer was trying to do too many things at once"." …



The MARS Pathfinder problem (2)

“VxWorks provides preemptive priority scheduling of threads. Tasks on the Pathfinder spacecraft were executed as threads with priorities that were assigned in the usual manner reflecting the relative urgency of these tasks.”

“Pathfinder contained an "information bus", which you can think of as a shared memory area used for passing information between different components of the spacecraft.”

- A bus management task ran frequently with high priority to move certain kinds of data in and out of the information bus. Access to the bus was synchronized with mutual exclusion locks (mutexes).”

The MARS Pathfinder problem (3)

- The meteorological data gathering task ran as an infrequent, low priority thread, … When publishing its data, it would acquire a mutex, do writes to the bus, and release the mutex. ..
- The spacecraft also contained a communications task that ran with medium priority."

☞

High priority: retrieval of data from shared memory
Medium priority: communications task
Low priority: thread collecting meteorological data

The MARS Pathfinder problem (4)

"Most of the time this combination worked fine. However, very infrequently it was possible for an interrupt to occur that caused the (medium priority) communications task to be scheduled during the short interval while the (high priority) information bus thread was blocked waiting for the (low priority) meteorological data thread. In this case, the long-running communications task, having higher priority than the meteorological task, would prevent it from running, consequently preventing the blocked information bus task from running. After some time had passed, a watchdog timer would go off, notice that the data bus task had not been executed for some time, conclude that something had gone drastically wrong, and initiate a total system reset. This scenario is a classic case of priority inversion."

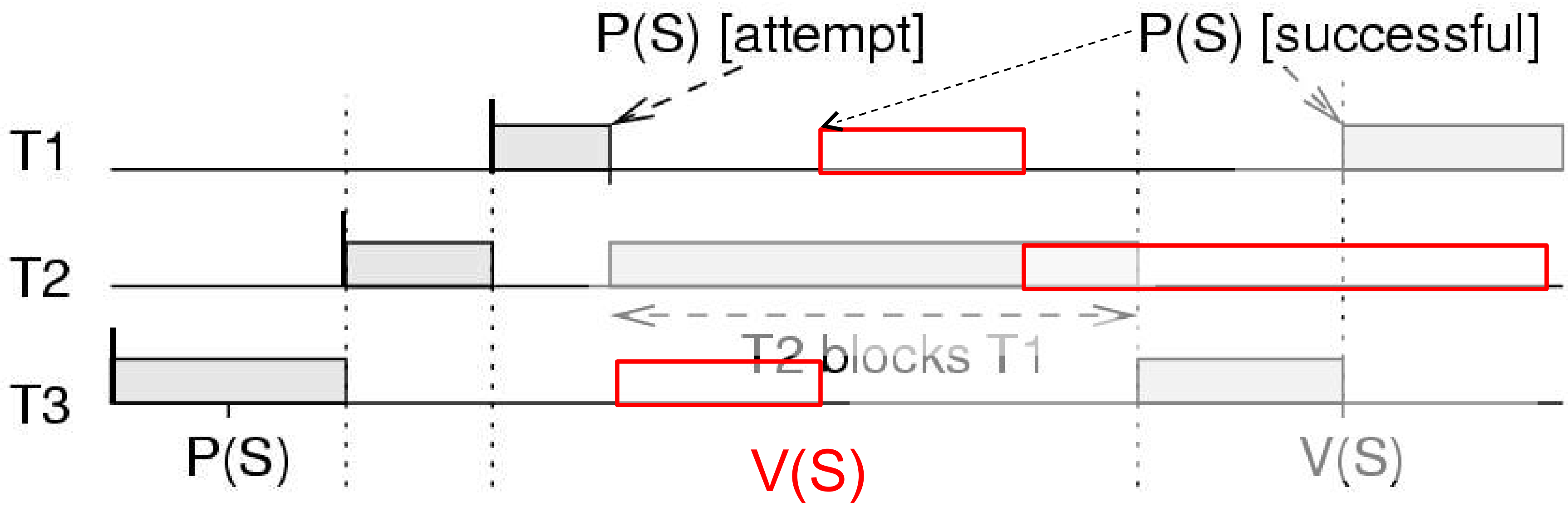
Coping with priority inversion: the priority inheritance protocol

- Tasks are scheduled according to their active priorities. Tasks with the same priorities are scheduled FCFS.
- If task T1 executes P(S) & exclusive access granted to T2: T1 will become blocked.
  If priority(T2) < priority(T1): T2 inherits the priority of T1.
  ☞ T2 resumes.
  Rule: tasks inherit the highest priority of tasks blocked by it.
- When T2 executes V(S), its priority is decreased to the highest priority of the tasks blocked by it.
  If no other task blocked by T2: priority(T2):= original value.
  Highest priority task so far blocked on S is resumed.
- Transitive: if T2 blocks T1 and T1 blocks T0, then T2 inherits the priority of T0.

Example

How would priority inheritance affect our example with 3 tasks?

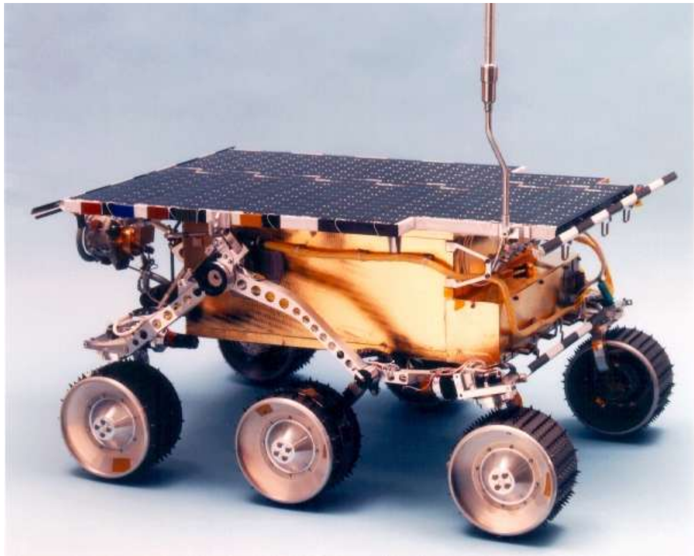
T3 inherits the priority of T1 and T3 resumes.



Priority inversion on Mars

Priority inheritance also solved the Mars Pathfinder problem: the VxWorks operating system used in the pathfinder implements a flag for the calls to mutex primitives. This flag allows priority inheritance to be set to "on". When the software was shipped, it was set to "off".

The problem on Mars was corrected by using the debugging facilities of VxWorks to change the flag to "on", while the Pathfinder was already on the Mars [Jones, 1997].



Remarks on priority inheritance protocol

Possible large number of tasks with high priority.

Possible deadlocks.

Ongoing debate about problems with the protocol:

> Victor Yodaiken: Against Priority Inheritance,
> http://www.fsmlabs.com/articles/inherit/inherit.html

Finds application in ADA: During *rendez-vous*, task priority is set to the maximum.

More sophisticated protocol: priority ceiling protocol.

Summary

Periodic scheduling

- Rate monotonic scheduling
  - Proof of the utilization bound for n=2.
- EDF
- Dependent and sporadic tasks (briefly)

Resource access protocols

- Priority inversion
  - The Mars pathfinder example
- Priority inheritance
  - The Mars pathfinder example