طراحی سیستم‌های تعبیه‌شده
Embedded System Design

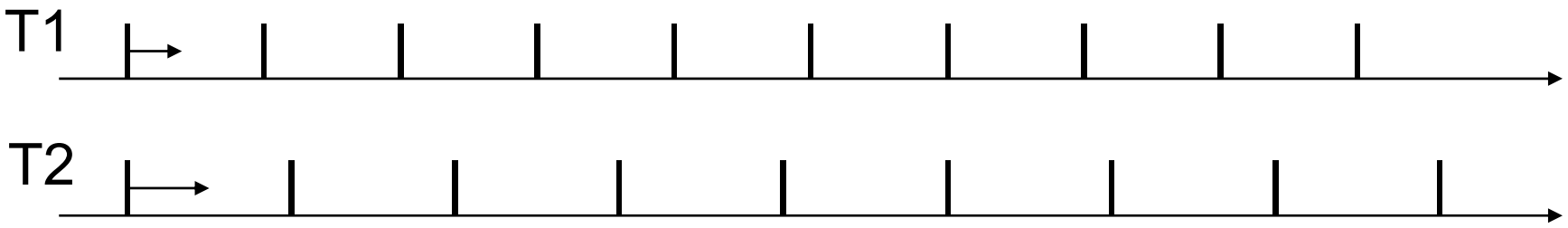
فصل چهارم ـ قسمت دوم

نرم‌افزار سیستم تعبیه‌شده
Embedded System Software

کاظم فولادی
دانشکده‌ی مهندسی برق و کامپیوتر
دانشگاه تهران

kazim@fouladi.ir

زمان‌بندی متناوب
Periodic scheduling



برای زمان‌بندی متناوب، بهترین کاری که می‌توانیم انجام دهیم، طراحی الگوریتمی است که همیشه یک زمان‌بندی را در صورت وجود پیدا می‌کند.

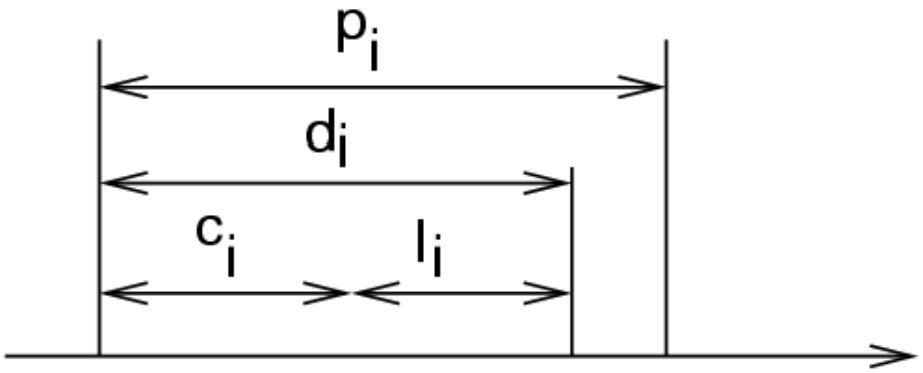
⇐ یک زمان‌بند **بهینه** تعریف می‌شود،

اگر و فقط اگر یک زمان‌بندی را در صورت وجود بیابد.

زمان‌بندی متناوب
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 $l_i = d_i - c_i$



سودمندی انبارشده
Accumulated utilization

Accumulated utilization:

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

$\mu \leq m$ شرط لازم برای قابلیت زمان‌بندی
(m با = تعداد پردازنده‌ها)

وظایف مستقل: زمان‌بندی با نرخ یکنوا
Independent tasks: Rate monotonic (RM) scheduling

معروف‌ترین تکنیک برای زمان‌بندی وظایف مستقل متناوب [Liu, 1973].

فرضیات:

- همه‌ی وظایف دارای مهلت زمانی، متناوب هستند.
- همه‌ی وظایف مستقل هستند.
- برای همه‌ی وظایف $d_i=p_i$
- c_i ثابت است و برای همه‌ی وظایف معلوم است.
- زمان لازم برای تعویض متن ناچیز است.
- برای یک پردازنده و n وظیفه، معادله‌ی زیر برای سودمندی انبارشده μ برقرار باشد:

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

زمان‌بندی با نرخ یکنوا ـ سیاست ـ
Rate monotonic (RM) scheduling - The policy -

سیاست RM:
اولویت یک وظیفه تابعی یکنوای نزولی از دوره‌ی تناوب آن است.
در هر زمان وظیفه‌ی دارای بالاترین اولویت بین آنهایی که برای اجرا آماده هستند، تخصیص داده می‌شود.

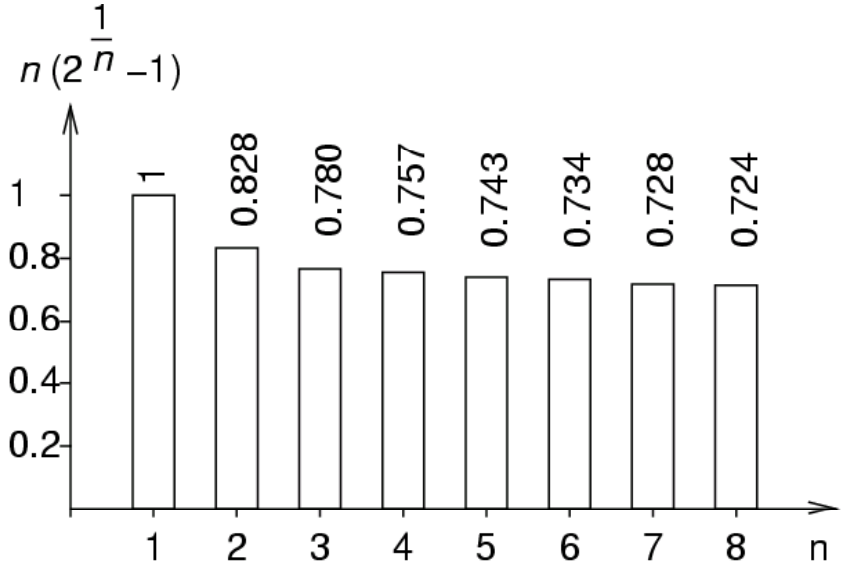
قضیه:
اگر همه‌ی فرضیات RM برقرار باشند، قابلیت زمان‌بندی تضمین می‌شود.

سودمندی ماکزیمم برای قابلیت زمان‌بندی تضمین‌شده
Maximum utilization for guaranteed schedulability

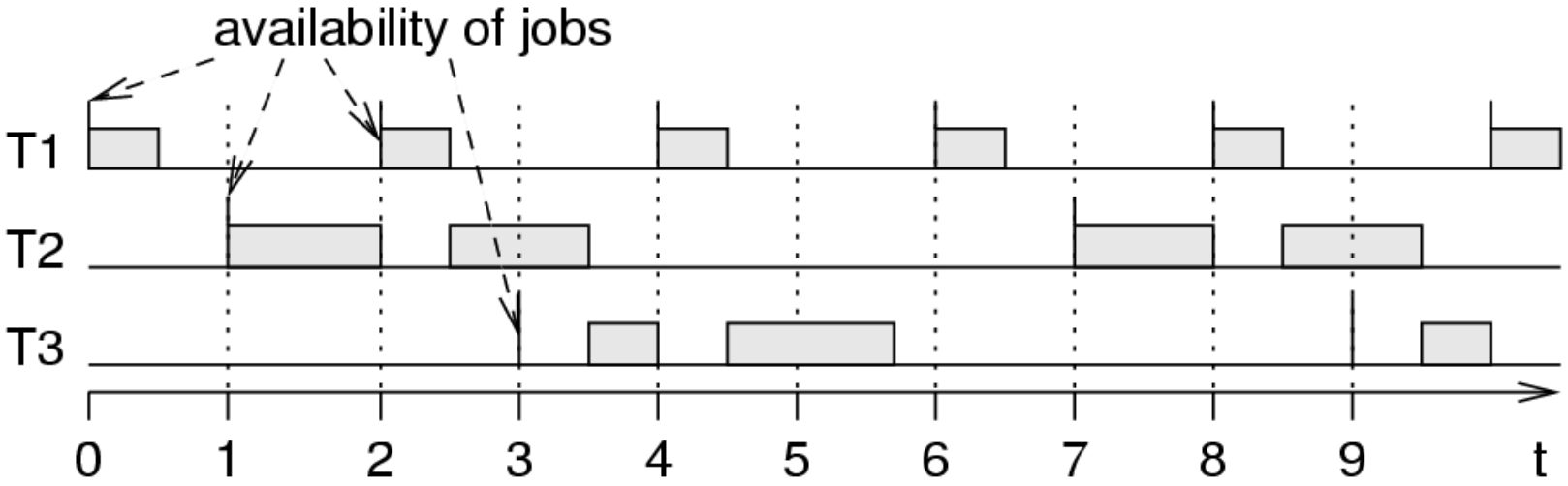
Maximum utilization as a function of the number of tasks:

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

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



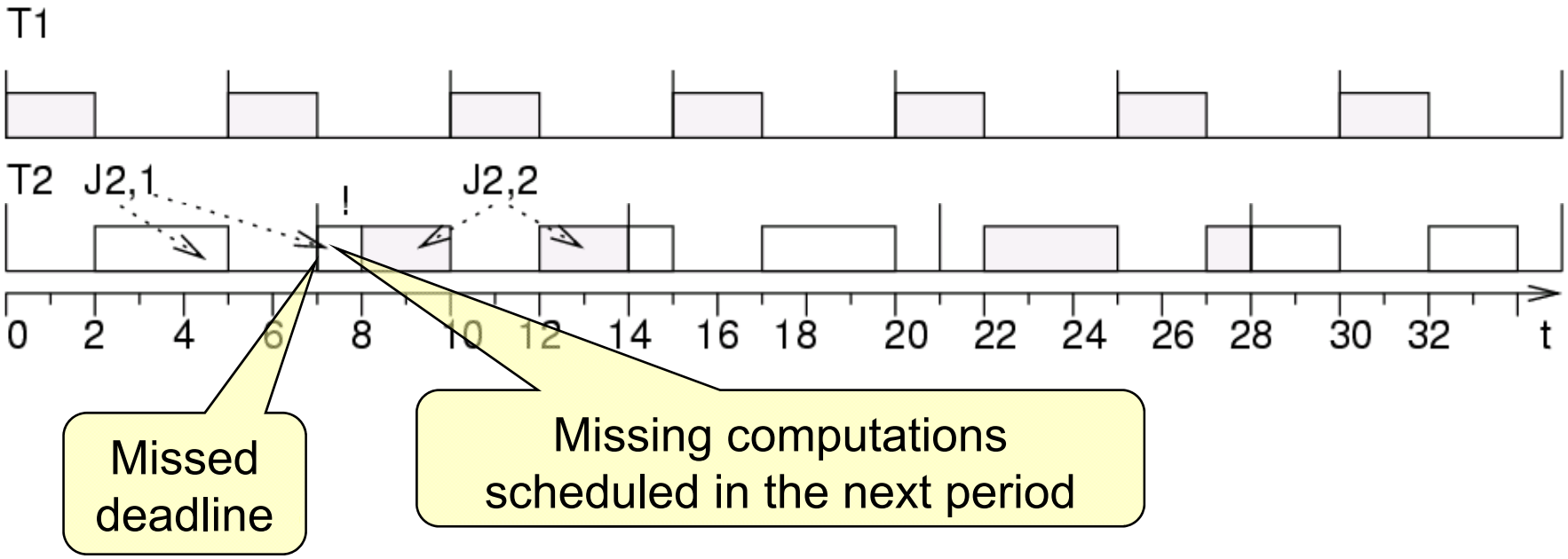
مثالی از زمان‌بندی تولیدشده با RM
Example of RM-generated schedule



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

حالت شکست زمان‌بندی RM
Case of failing RM scheduling

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



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.

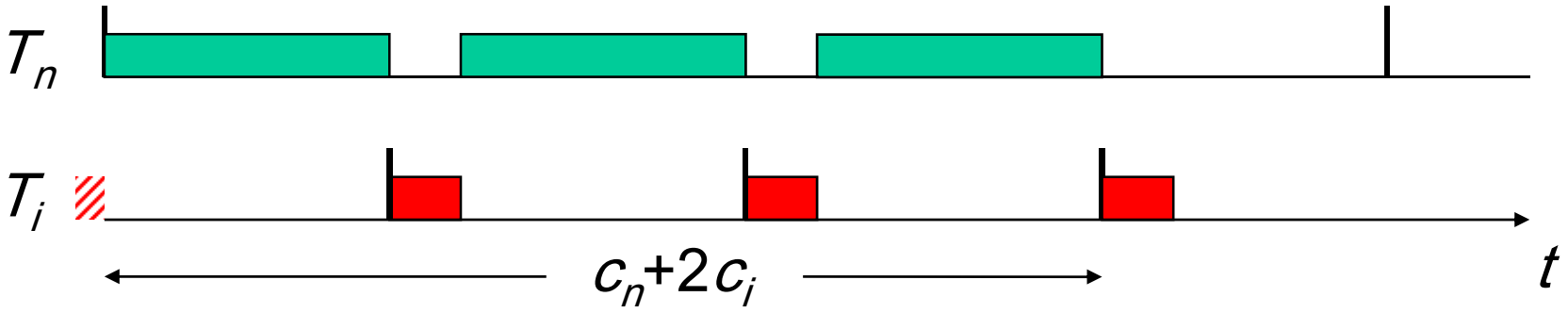
Lemma: For any task, the **critical instant** occurs if that task is simultaneously released with all higher priority tasks.

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

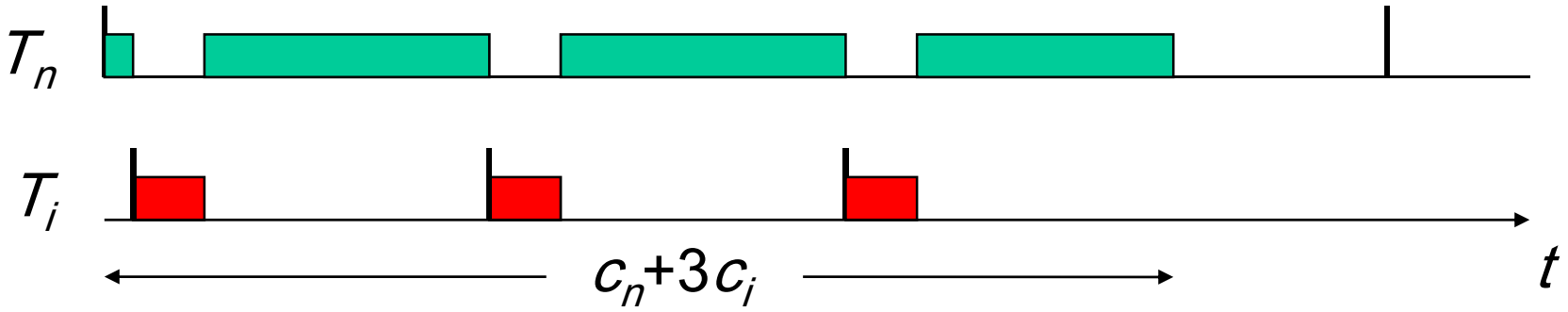
Source: G. Buttazzo, Hard Real-time Computing Systems, Kluwer, 2002

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 $i = 1, \ldots n-1$:

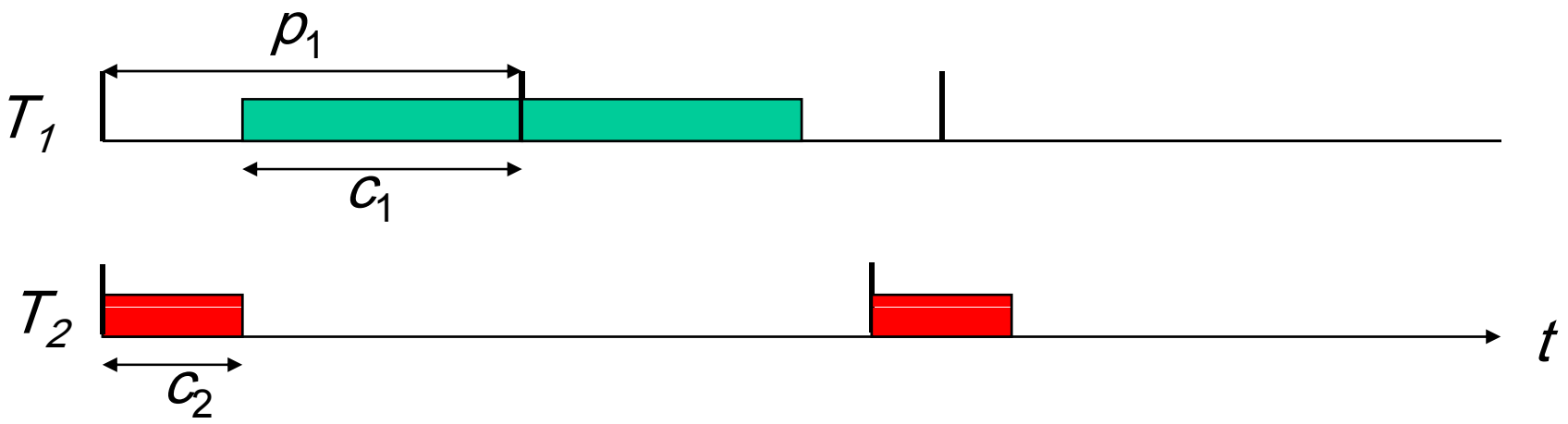
☞The worst case response time of a task occurs when it is released simultaneously with all higher-priority tasks. q.e.d.

☞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:



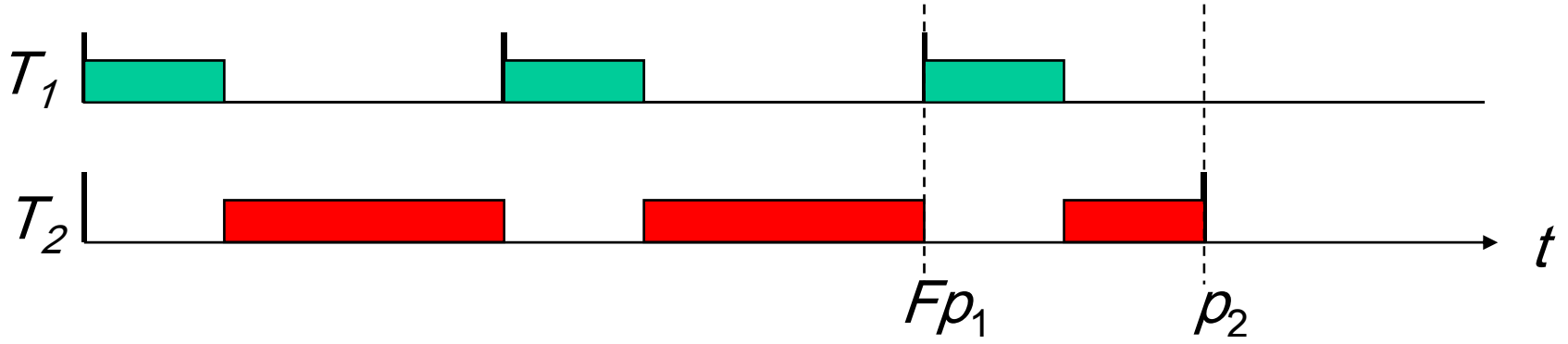
Schedule is feasible if $c_1 + c_2 \leqq p_1$ (1)

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 $c_1+c_2 \le p_1$ (1)

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

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

Since F ≥ 1: $Fc_1+c_2 \le Fc_1+Fc_2 \le Fp_1$

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

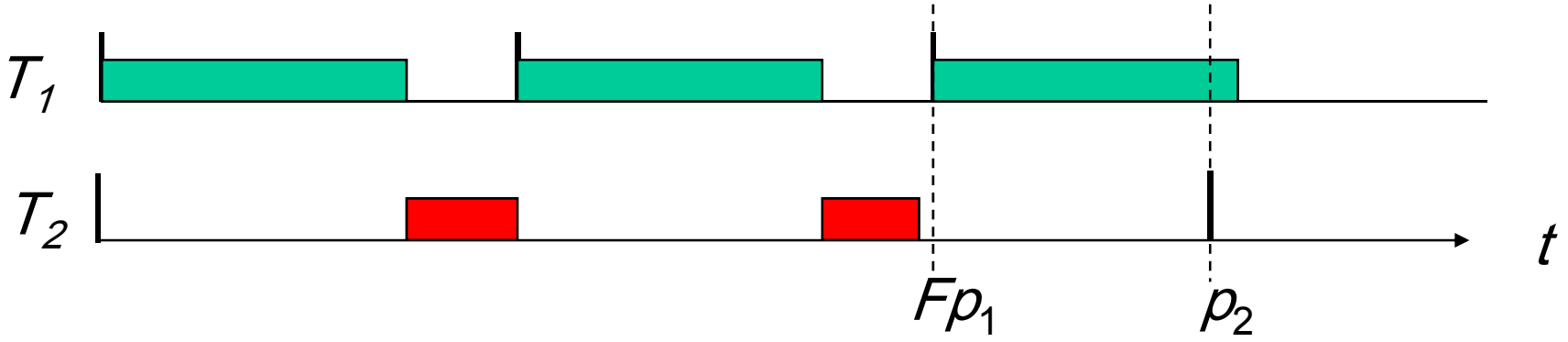
Since $c_1 \le p_2 - Fp_1$: $(F+1)c_1+c_2 \le Fp_1 + c_1 \le p_2$

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 $F c_1 + c_2 \le F p_1$ (3)

$c_1 + c_2 \le p_1$ (1)

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

Since $F \ge 1$: $F c_1 + c_2 \le F c_1 + F c_2 \le F p_1$

☞ Same statement as for case 1.

Calculation of the least upper utilization bound

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

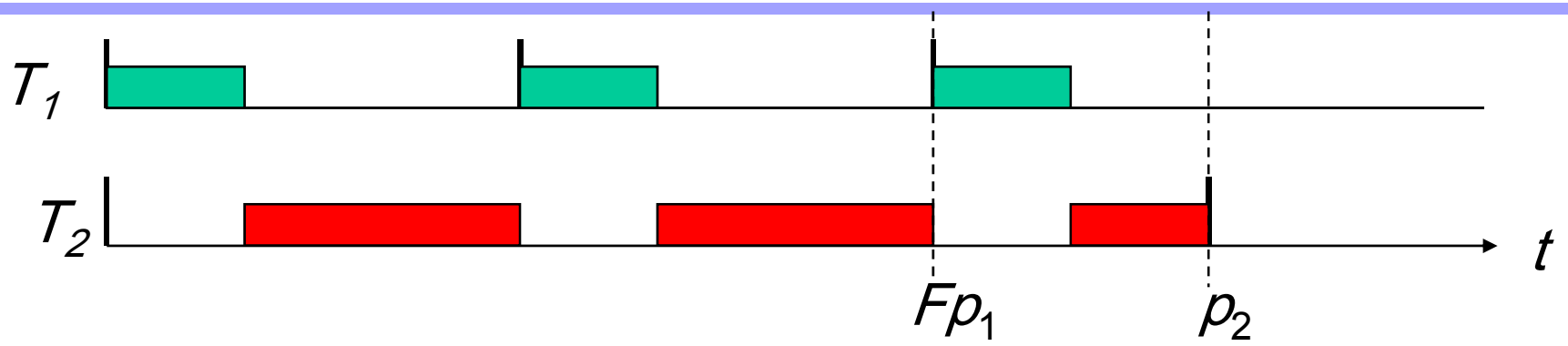
Proof procedure: compute least upper bound U_{lup} as follows

- 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 $c_2 = p_2 - c_1 (F+1)$

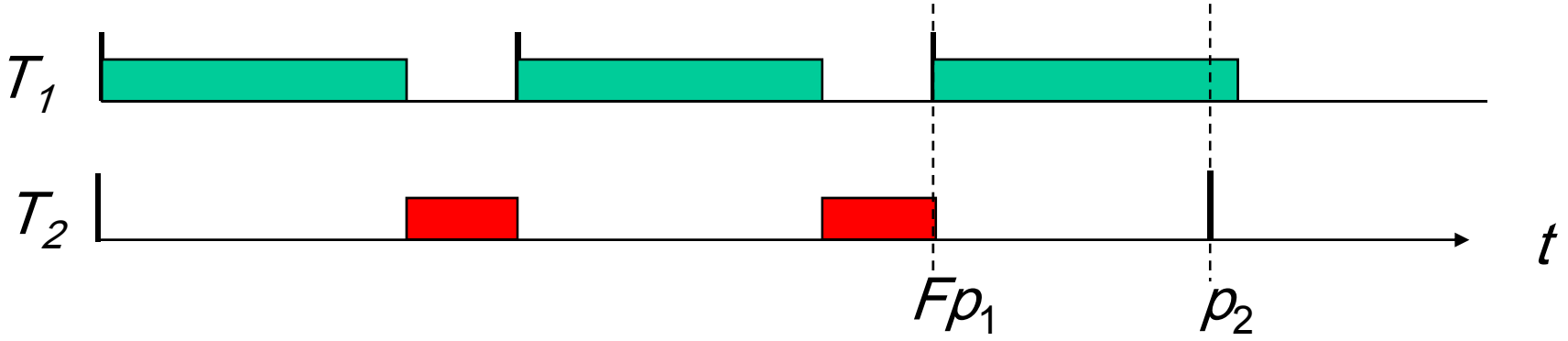
Corresponding upper bound is

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

{ } 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:

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

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

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

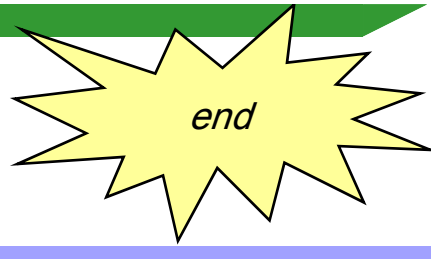
$$= 1 - \frac{G(1 - G)}{F + G}$$

Since $0 \le G < 1$: $G(1-G) \ge 0$ → U_{ub} increasing in F →

Minimum of U_{ub} for min(F): $F=1$ →

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

Proving the RM theorem for n=2



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

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

ویژگی‌های زمان‌بندی RM
Properties of RM scheduling

- بر اساس اثبات، بدیهی است که اگر $p_2=F\,p_1$ هیچ ظرفیت بیکاری لازم نخواهد بود. در حالت کلی: اگر دوره‌ی تناوب همه‌ی وظایف مضربی از دوره‌ی تناوب پراولویت‌ترین وظیفه باشد، لازم نیست، یعنی، قابلیت زمان‌بندی در این صورت نیز تضمین می‌شود اگر $\mu \leq 1$.
- زمان‌بندی RM مبتنی بر اولویت‌ها **ایستا** است. این به زمان‌بندی RM اجازه می‌دهد که در سیستم عامل‌های استاندارد، مانند ویندوز NT، استفاده شود.
- انواع گوناگونی از زمان‌بندی RM وجود دارد.
- در زمینه‌ی زمان‌بندی RM اثبات‌های رسمی فراوانی وجود دارد.

EDF

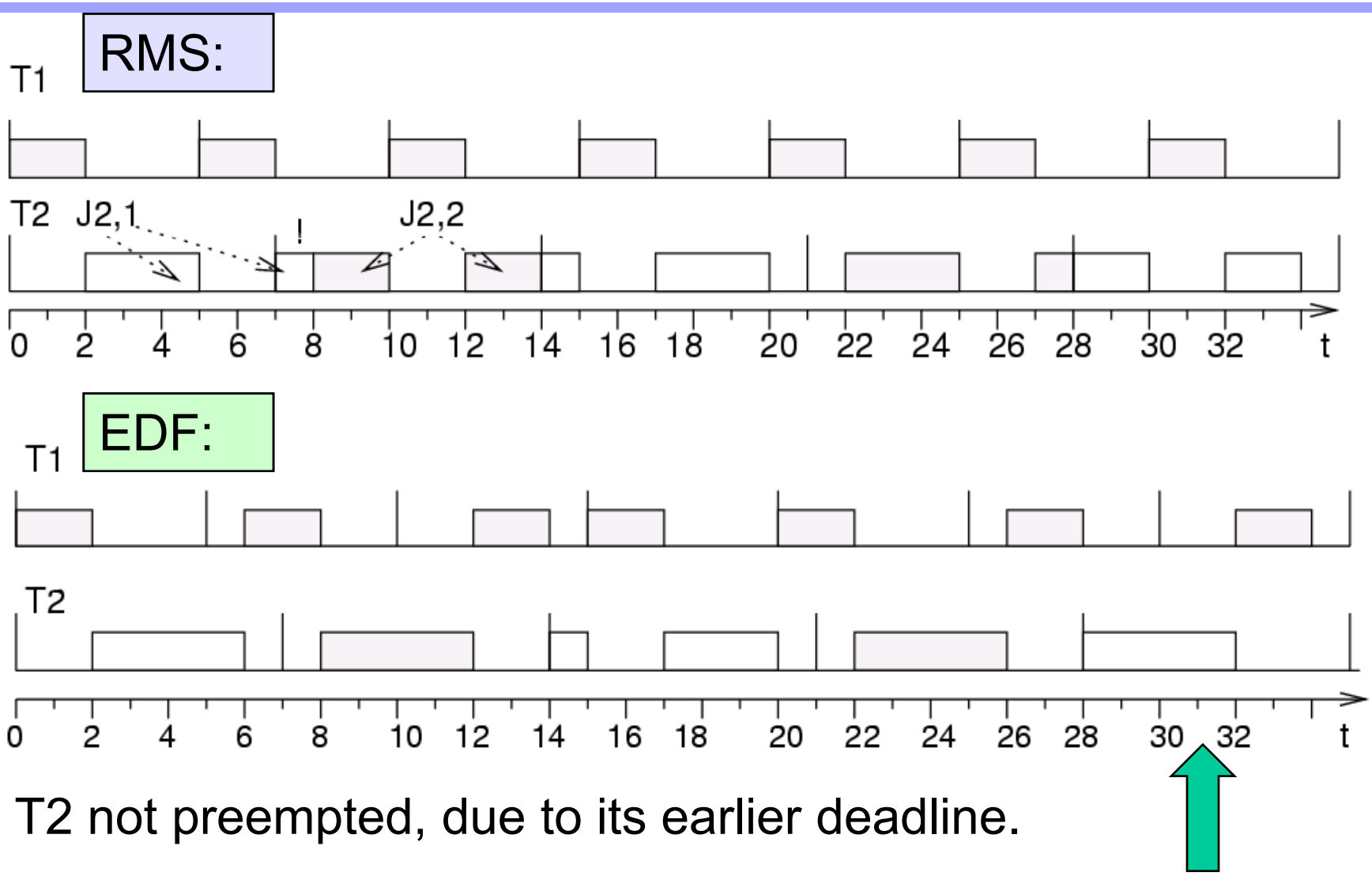
EDF نیز می‌تواند برای زمان‌بندی متناوب به کار گرفته شود.

EDF برای هر دوره‌ی تناوب بهینه است.

⇐ برای زمان‌بندی متناوب بهینه است.

⇐ EDF باید قادر باشد مثال‌هایی که در آنها RM شکست می‌خورد را زمان‌بندی کند.

مقایسه‌ی زمان‌بندی‌های EDF و RM
Comparison EDF/RMS



T2 not preempted, due to its earlier deadline.

EDF: ویژگی‌ها
EDF: Properties

EDF به اولویت‌های پویا نیاز دارد

⇐ EDF نمی‌تواند با سیستم عامل‌های استاندارد که تنها اولویت‌های ایستا را در نظر می‌گیرند، استفاده شود.

وظایف وابسته
Dependent tasks

مساله‌ی تصمیم‌گیری در مورد اینکه آیا برای مجموعه‌ای از وظایف مستقل و مهلت داده شده یک زمان‌بندی وجود دارد یا خیر، در حالت کلی NP-complete است [Garey/Johnson].

راهبردها:

۱. اضافه کردن منابع، به گونه‌ای که زمان‌بندی ساده شود شود.

۲. تقسیم مساله به بخش‌های ایستا و پویا تا تنها حداقل تصمیمات لازم باشد در زمان اجرا گرفته شود.

وظایف گاه و بیگاه
Sporadic tasks

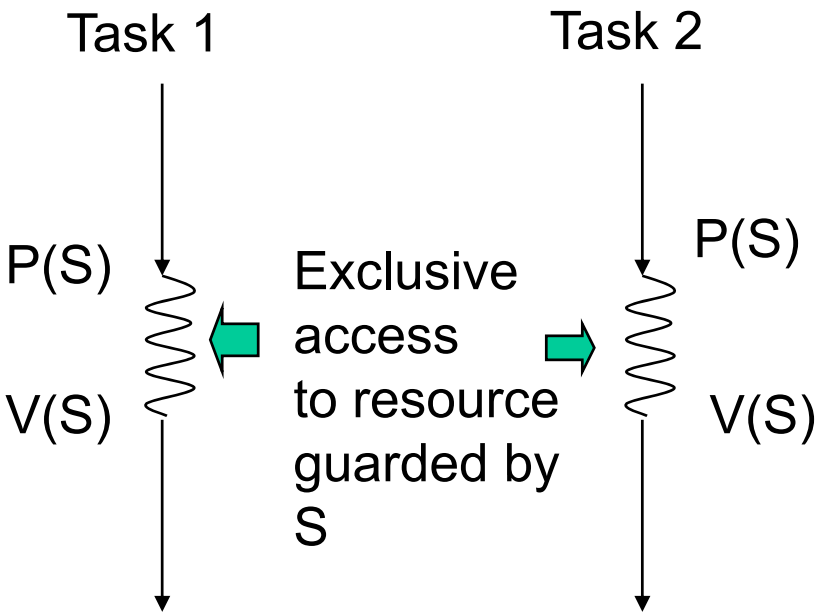
اگر وظایف گاه و بیگاه به وقفه‌ها متصل شوند، زمان اجرای سایر وظایف بسیار غیر قابل پیش‌بینی می‌شود.

⇐ وارد کردن یک سرور وظایف گاه و بیگاه،
بررسی متناوب برای وظَایف گاه و بیگاه آماده.

⇐ وظایف گاه و بیگاه اساساً به وظایف متناوب تبدیل می‌شوند.

قراردادهای دسترسی به منبع
Resource access protocols

نواحی بحرانی (critical sections): بخش‌هایی از کد که در آنها دسترسی انحصاری به برخی از منابع باید تضمین شود.
می‌تواند با سمافور S تضمین شود.



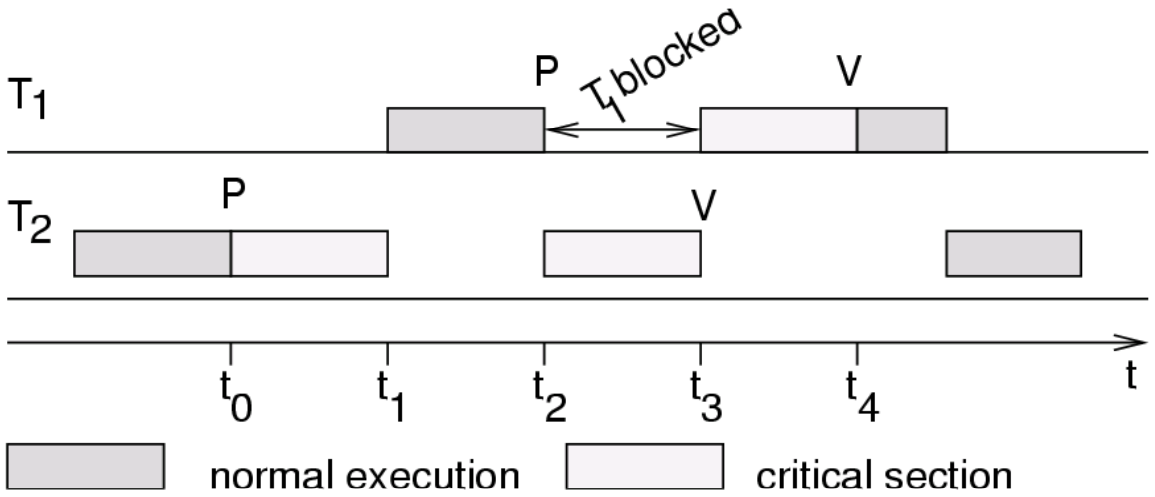
P(S) سمافور را بررسی می‌کند تا ببیند که آیا منبع موجود است یا خیر. اگر بود، S را به "used" مقداردهی می‌کند.
عملیات وقفه ناپذیر! اگر نبود، وظیفه‌ی فراخوانی کننده‌ی آن باید صبر کند.

V(S) به "unused" مقداردهی می‌شود و وظیفه‌ی منتظر (در صورت وجود) آغاز می‌شود.

معکوس‌سازی اولویت
Priority inversion

اولویت T_1 فرض می‌شود که باید بزرگتر از اولویت T_2 باشد.

اگر T_2 در ابتدا (زمان t_0) در خواست دسترسی انحصاری داشته باشد، T_1 باید صبر کند تا T_2 منبع را آزاد کند (زمان t_3)، بنابراین اولویت‌ها معکوس می‌شود.



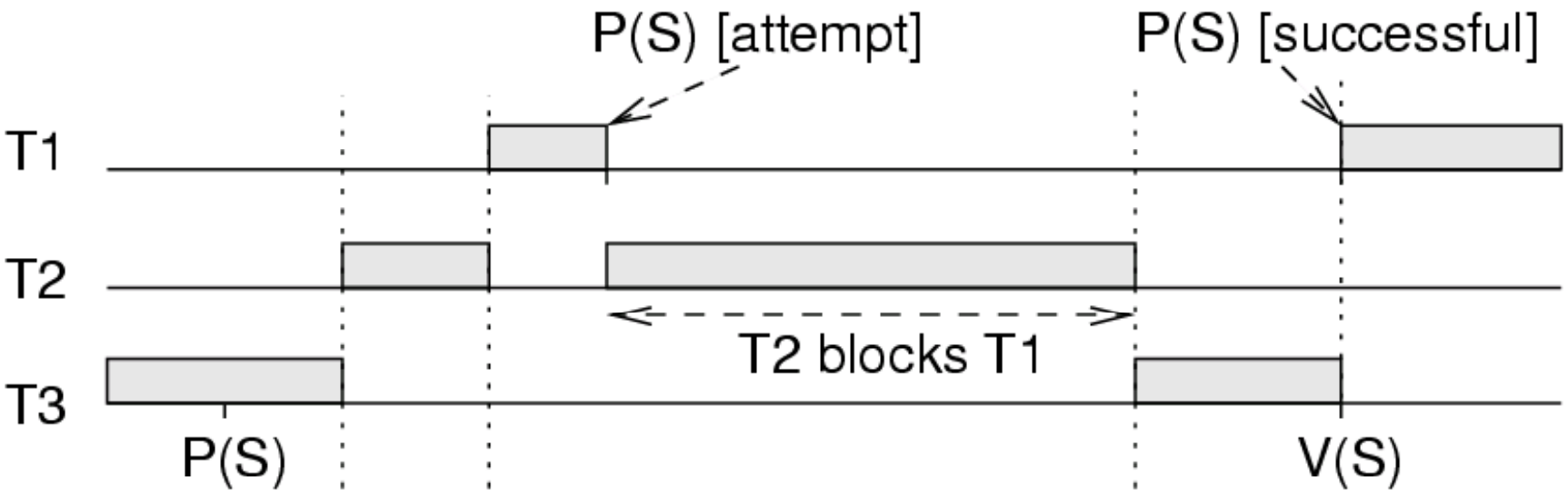
در این مثال مدت معکوس‌سازی با طول ناحیه‌ی بحرانی T_2 محدود شده است.

مدت معکوس‌سازی اولویت برای بیش از ۲ وظیفه می‌توان از طول هر ناحیه‌ی بحرانی تجاوز کند
Duration of priority inversion with >2 tasks can exceed the length of any critical section

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

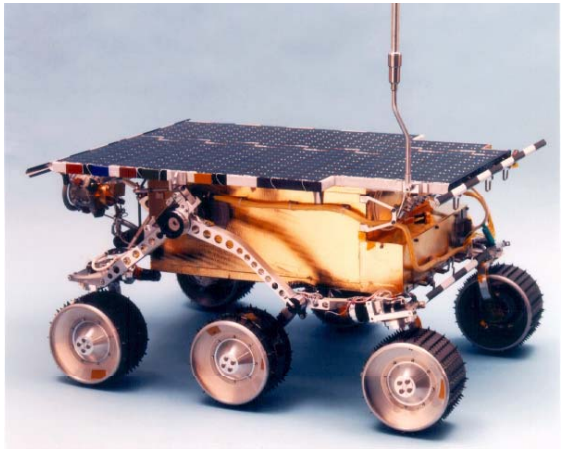
T2 preempts T3:

T2 can prevent T3 from releasing the resource.



مساله‌ی مسیریاب مریخ
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."

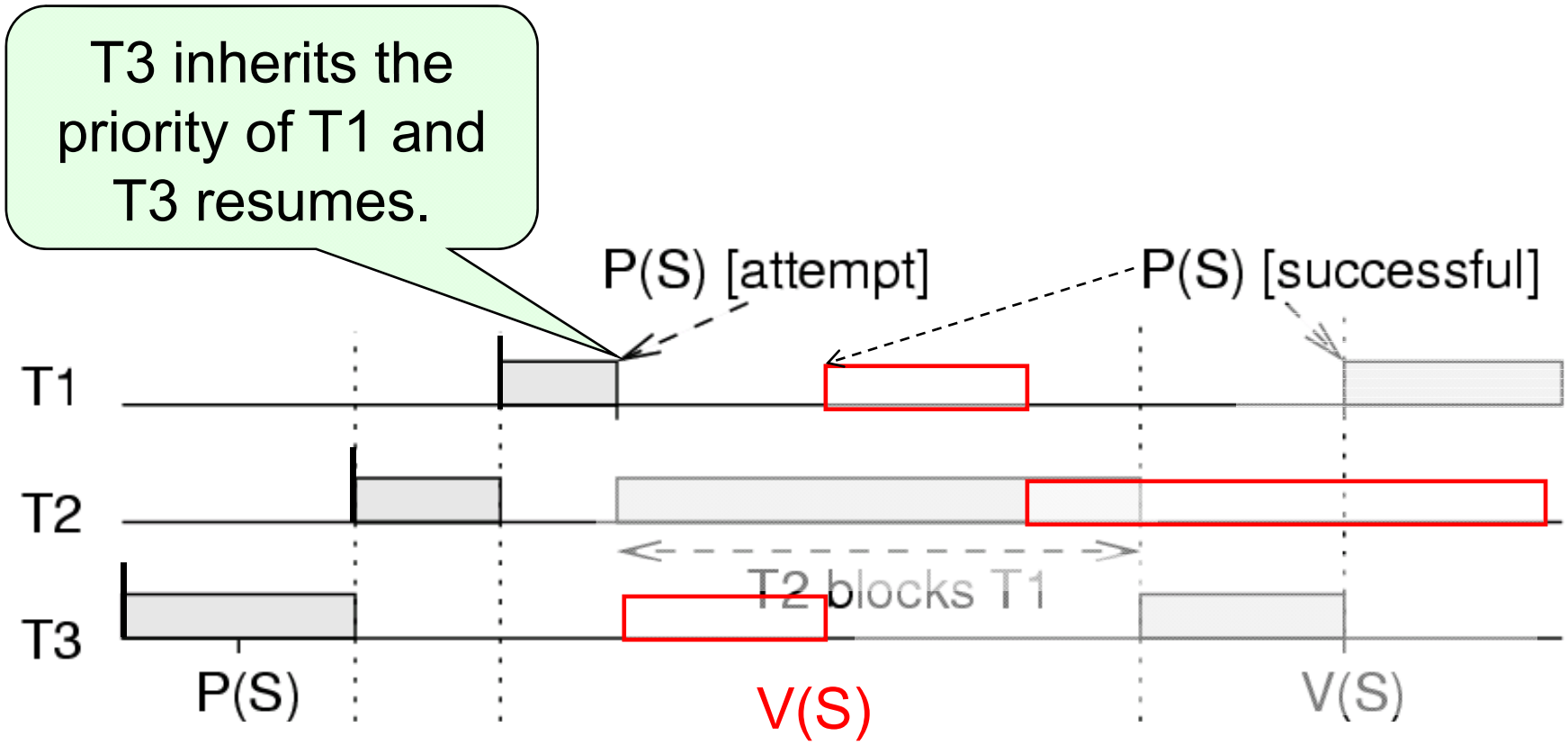
قرارداد وراثت اولویت

Coping with priority inversion: the priority inheritance protocol

- وظایف بر اساس اولویت‌های فعالشان زمان‌بندی می‌شوند. وظایف دارای اولویت‌های یکسان به صورت FCFS زمان‌بندی می‌شوند.
- اگر T1 دستور P(S) را اجرا کند و دسترسی انحصاری به T2 داده شده باشد: T1 مسدود می‌شود.
  اگر اولویت T2 کمتر از اولویت T1 باشد: T2 اولویت T1 را به ارث می‌برد ⇐ T2 ادامه می‌یابد.
  قاعده: وظایف بالاترین اولویت وظایفی که توسط آنها مسدود شده اند را به ارث می‌برد.
- وقتی T2 دستور V(S) را اجرا می‌کند، اولویت آن به بالاترین اولویت وظایف مسدود شده توسط آن کاهش می‌یابد.
  اگر هیچ وظیفه‌ای توسط T2 مسدود نشده بود: اولویت T2 می‌شود: مقدار اصلی. وظیفه‌ی دارای بالاترین اولویت تاکنون بر روی S مسدود شده است، از سر گرفته می‌شود.
- تراگذری: لگر T2، T1 را مسدود کند و T1، T0 را مسدود کند، آن‌گاه T2 اولویت T0 را به ارث می‌برد.

Example

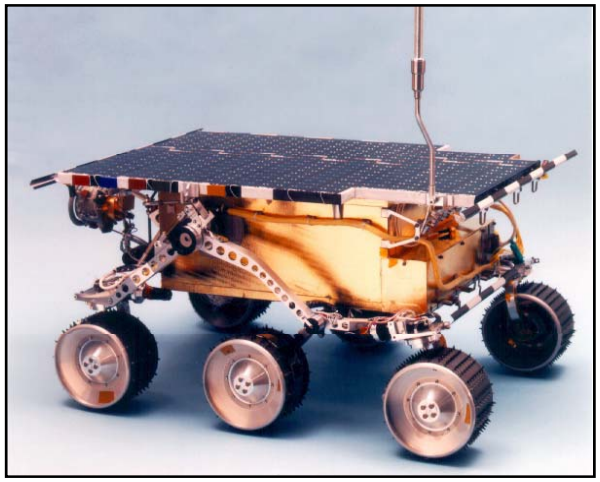
How would priority inheritance affect our example with 3 tasks?



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].



توضیحاتی در مورد قرارداد وراثت اولویت
Remarks on priority inheritance protocol

امکان تعدادی زیادی وظیفه با اولویت بالا.

امکان بن‌بست.

بحث‌های پی در پی در مورد مشکلات این قرارداد:

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

دارای کاربردهایی در ADA: در حین قرار ملاقات، اولویت وظیفه به ماکزیمم آن مقداردهی می‌شود.

قرارداد پیشرفته‌تر: قرارداد سقف اولویت (priority ceiling protocol)

خلاصه

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