Advanced System Software
（先端システムソフトウェア）

#7 (2018/10/25)

CSC.T431, 2018-3Q
Mon/Thu 9:00-10:30, W832

Instructor: Takuo Watanabe（渡部卓雄）
Department of Computer Science

e-mail: takuo∅c.titech.ac.jp
http://www.psg.c.titech.ac.jp/~takuo/
ext: 3690, office: W8E-805

About vTaskDelete

- In the slide used at the last lecture, I wrote:
  - "If a task is no longer required, it should be stopped and deleted explicitly from outside of it." (p. 7)
- The above is not correct for the current version of FreeRTOS. You may use NULL as the argument of vTaskDelete to safely finish the task by itself.
- Note that the IDLE tasks are responsible for freeing the kernel memory used for the deleted tasks. So your code should not cause IDLE tasks to be starved.

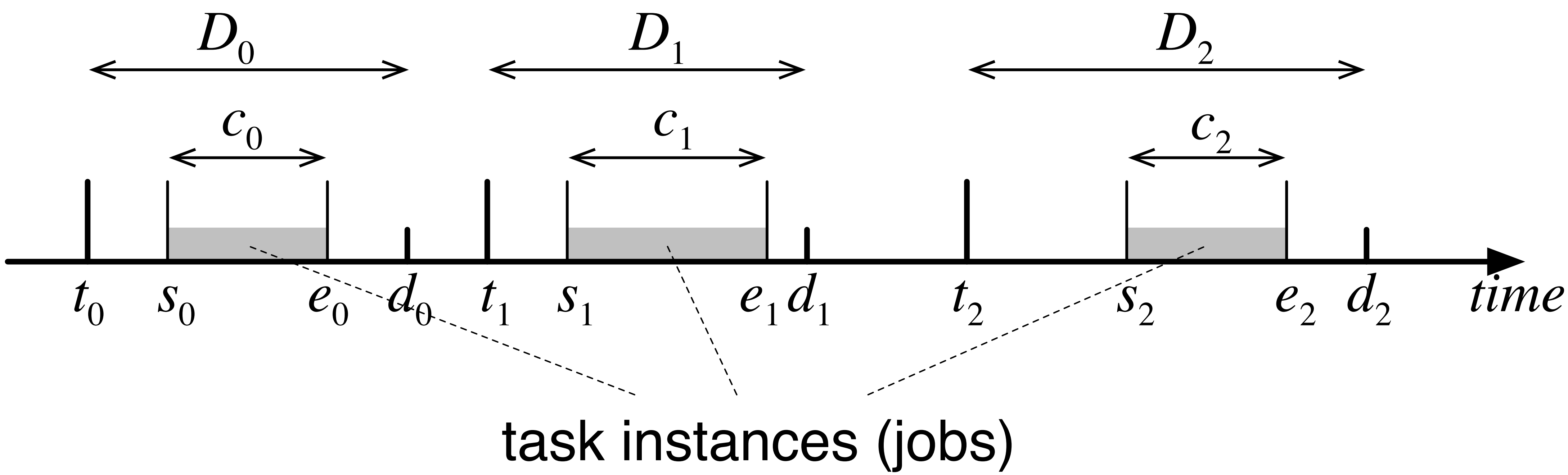
Technical Requirements on Your Projects

- Each project should use at least one M5Stack (or other) peripherals
  - LCD, Buttons, Speaker, Motion Sensor
  - Temperature/Humidity Sensor (DHT12), etc.
- It is desirable that your project uses at least one wireless capabilities (If your team has more than one devices, this is required).
  - WiFi, Bluetooth, BLE
- Your code can be written in any languages, but should contain the direct use of at least one FreeRTOS features (in C/C++), such as:
  - Multitasking, Inter-task synch/comm, etc.

Agenda

- Real-Time Task Scheduling Algorithm

Real-Time Task

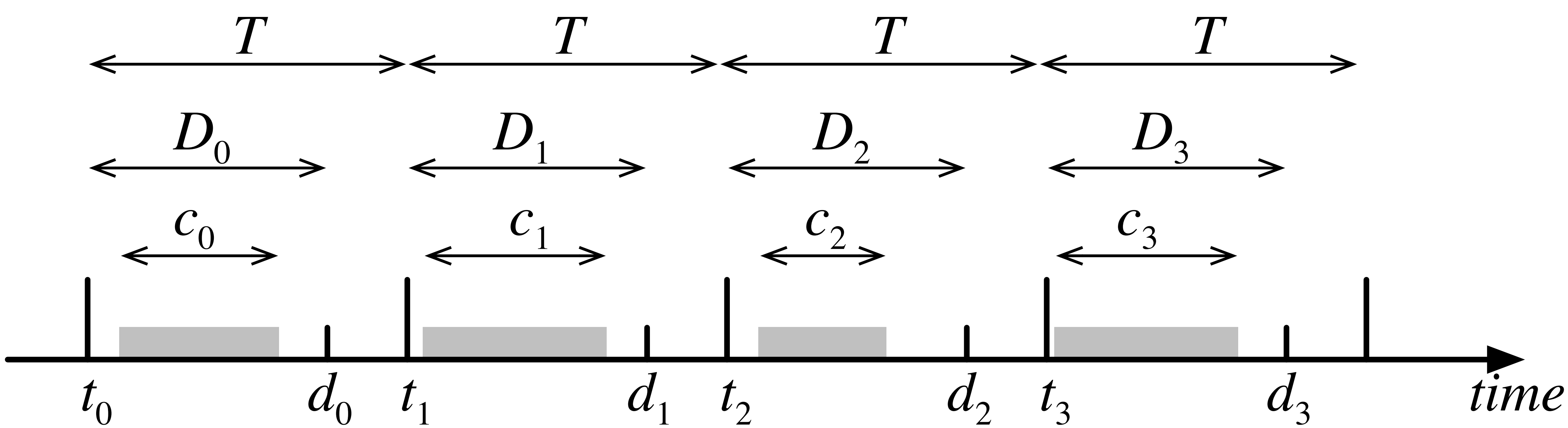


- t_i release (arrival) time of i-th task instance that corresponds to i-th event
- s_i start time of i-th task instance
- e_i end time of i-th task instance
- d_i absolute deadline of i-th task instance
- D_i relative deadline of i-th task instance ($D_i = d_i - t_i$)
- c_i (worst case) execution time of i-th task instance ($c_i \leq D_i$)

Real-time Tasks

- Periodic Task
  - A task consists of a sequence of similar (or identical) jobs that are arrived at a constant rate.
  - e.g. sensor value acquisition, playing videos
- Aperiodic Task
  - A task consists of a sequence of jobs that are arrived at irregular intervals.
  - e.g. user activities
- Sporadic Task
  - An aperiodic task characterized by a minimum inter-arrival time between consecutive activities.
  - e.g. network packets

Periodic Task (1)



T period ($T = t_{i+1} - t_i$)

ϕ phase ($= t_0$)

t_i release time of i-th task instance ($t_i = \phi + iT$)

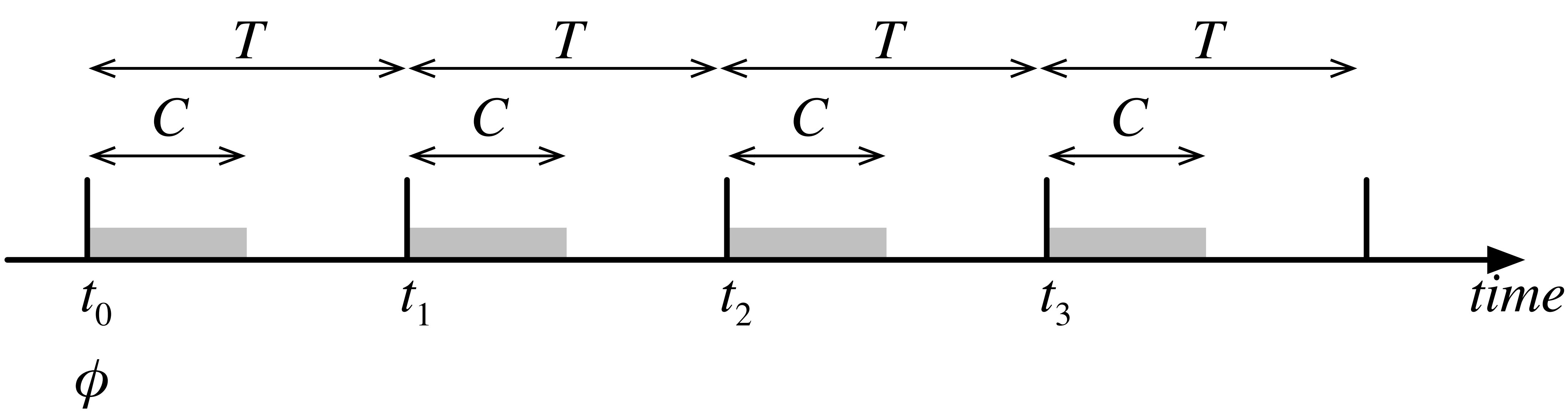
d_i absolute deadline of i-th task instance

D_i relative deadline of i-th task instance ($D_i = d_i - t_i$)

c_i (worst case) execution time of i-th task instance ($c_i \leq D_i$)

Periodic Task (2)

- To make things simpler, we assume that
  - $\forall i \in \boldsymbol{N}.\ D_i = T$,
  - $\forall i \in \boldsymbol{N}.\ s_i = t_i$, and
  - $\forall i \in \boldsymbol{N}.\ c_i = C$.
- Thus we can describe a periodic task τ as a triple (T, C, ϕ).



Schedule of Tasks

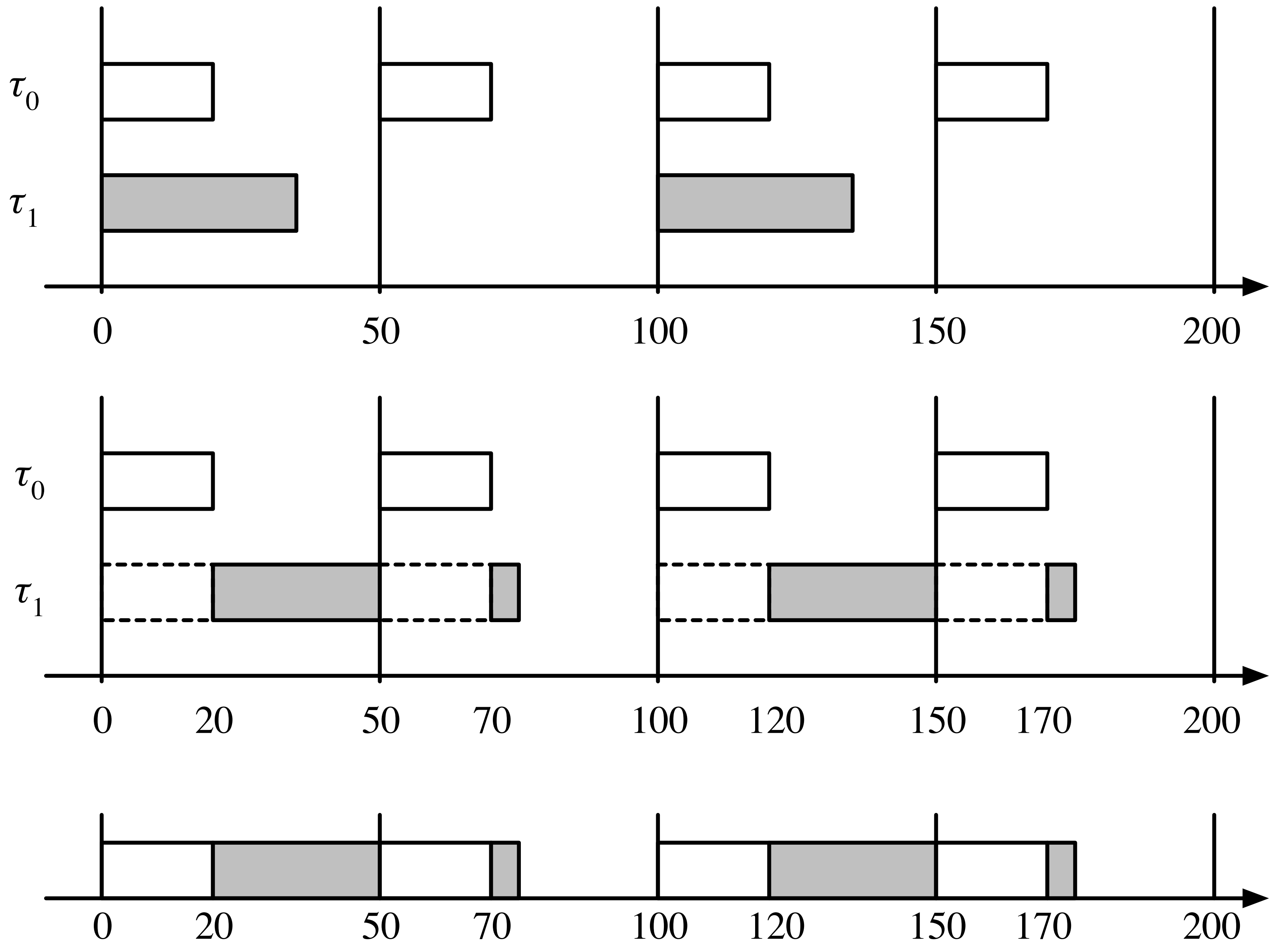
- A *schedule* is an assignment of tasks to processors.
  - Here, we only consider uniprocessor machines.
- A schedule is *feasible* if all tasks (task instances) can be completed according a set of specified constraints (e.g., deadlines).
- A set of tasks is *schedulable* if there exists at least one feasible schedule.

Scheduling Algorithm

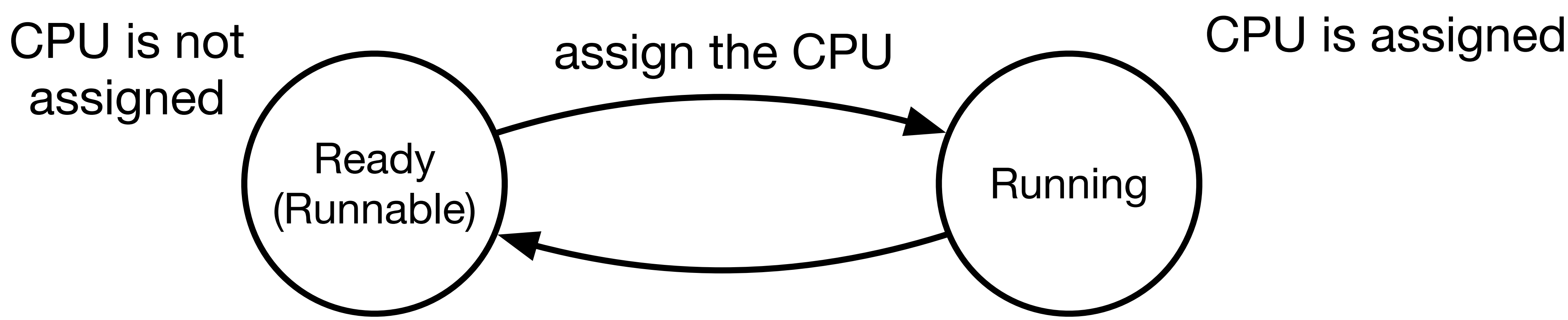
- A *scheduling algorithm* generates a schedule from a set of tasks.
- A scheduling algorithm is *preemptive* if it generates a schedule in which any running task can be arbitrary suspended at any time, to assign the CPU to another task.
- A scheduling algorithm is *static* if its scheduling decision is based on fixed parameters, assigned to tasks before their activation.

Preemptive Scheduling

$$\tau_0 = (50, 20, 0), \tau_1 = (100, 35, 0)$$



Task (Process/Thread) States



- Running
  - The CPU is running the task.
  - In a uniprocessor system, only a single task can be in this state at a time.
- Ready
  - The task is prepared to execute when given the opportunity

Scheduling Periodic Tasks

- Let Γ be a set of n periodic tasks.

$$
\begin{aligned}
\Gamma &= \{\tau_0, \tau_1, \tau_2, \ldots, \tau_{n-1}\} \\
\tau_i &= (T_i, C_i, \phi_i) \qquad (0 \leq i < n)
\end{aligned}
$$

- We assume the following:
  - Number of tasks in Γ is fixed; *i.e.*, no runtime task creations/deletions occur.
  - All tasks in Γ are independent; *i.e.*, there are no precedence relations and no resource constraints.
  - No tasks can suspend itself; *e.g.*, I/O operations.
  - No kernel overheads.

Rate Monotonic (RM) Scheduling Algorithm

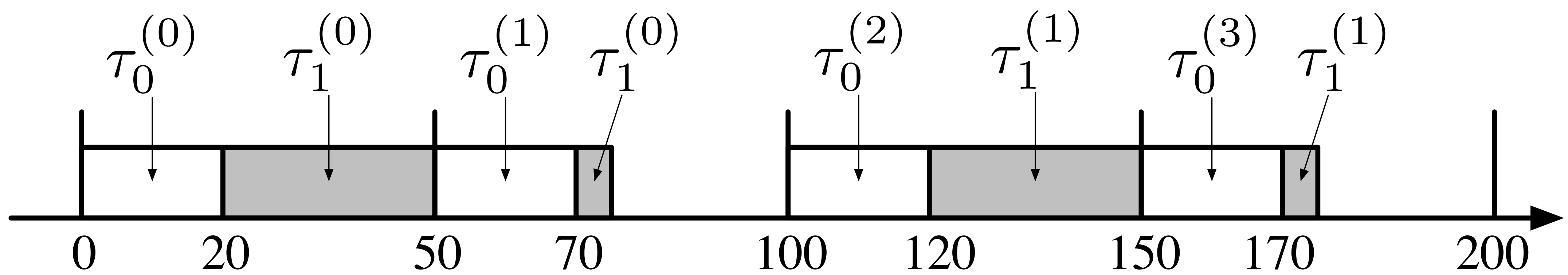
- A preemptive and static scheduling algorithm that assigns the priorities to tasks according to the task periods.
  - $T_i < T_j \Rightarrow p(\tau_i) > p(\tau_j)$.
    - $p(\tau)$: priority of τ

- RM algorithm is optimal in the sense that if a set of periodical tasks Γ cannot be scheduled by this algorithm, it cannot be scheduled by any algorithms that assigns task priorities statically.

Example 1

tao0 with higher priority

$$\Gamma = \{\tau_0,\ \tau_1\},\ \tau_0 = (50,\ 20,\ 0),\ \tau_1 = (100,\ 35,\ 0)$$

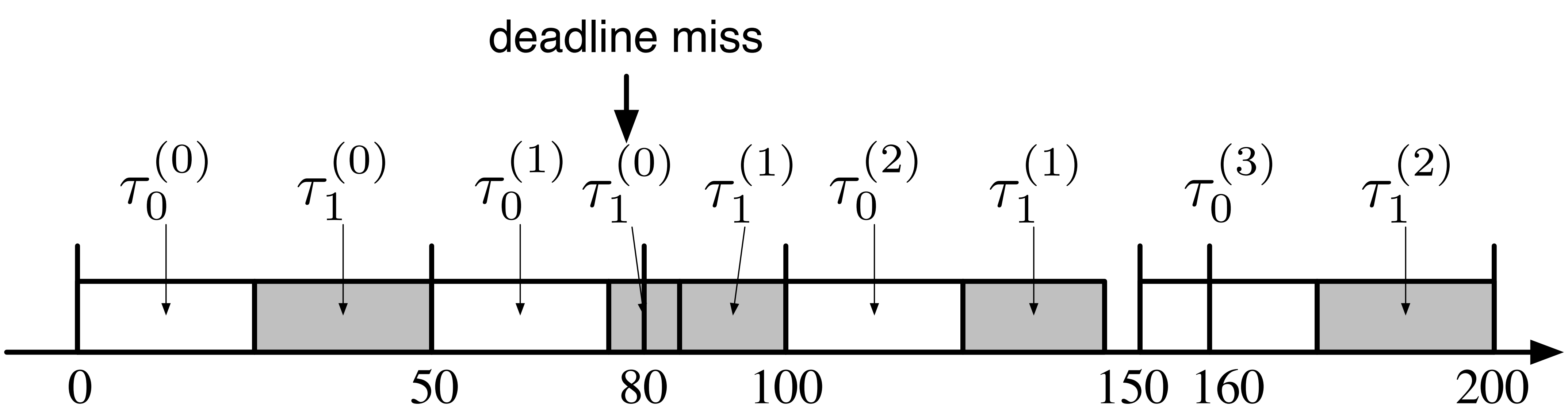


Γ is schedulable with RM algorithm.

$\tau_i^{(k)}$ is the k-th task instance of task τ_i.

Example 2

$$\Gamma = \{\tau_0,\ \tau_1\},\ \tau_0 = (50,\ 25,\ 0),\ \tau_1 = (80,\ 35,\ 0)$$



Γ is not feasible (deadline misses occur) with RM algorithm.

Processor Utilization Factor (1)

- Given a set $\Gamma = \{ \tau_0, \tau_1, ..., \tau_{n\text{-}1} \}$ of n periodical tasks ($\tau_i = (T_i, C_i, \phi_i)$), the *processor utilization factor* U of Γ is the fraction of processor time spent in the execution of Γ.

$$U = \sum_{i=0}^{n-1} \frac{C_i}{T_i}$$

the least upper bound of the utilization factor is the minimum of the utilization factors over all sets of tasks that fully utilize the processor. This means that for all task sets whose utilization factor is below this bound, there exists a priority assignment that is feasible.

- Theorem 1: Γ is not schedulable if $U > 1$.
- Theorem 2: Γ is schedulable with RM if

$$U \leq n(2^{\frac{1}{n}} - 1).$$

Processor Utilization Factor (2)

- $n(2^{1/n}-1)$

n	1	2	3	4	5	6	7	8	9	10	∞
$n(2^{1/n}-1)$	1	0.828	0.78	0.757	0.743	0.735	0.729	0.724	0.721	0.718	0.693

- Thus any Γ is schedulable with RM if $U < 0.693$.

if u>this value also possibly schedulable

On the other hand any task set with utilization factor greater than this bound can only achieve a feasible schedule if the task periods and execution times are specially related to each other.

Proof of Theorem 2

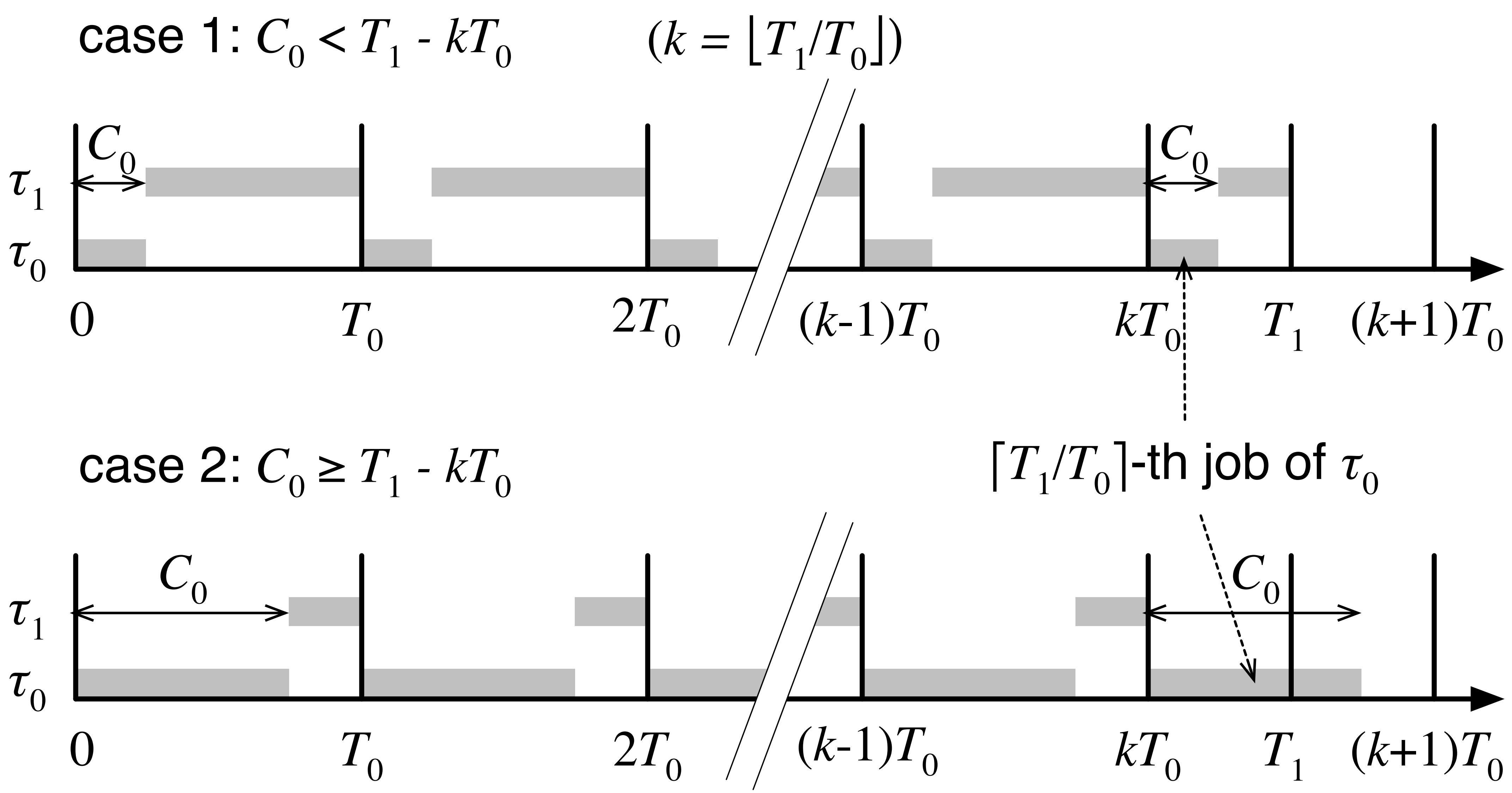
- The proof is divided into two cases:
  - $n = 2$
  - $n > 2$
- Proof Outline
  - Consider the case where the CPU usage is maximal. fully utilize:ddl is at the next starting time. slightly delay of the execution time caused infeasible schedule
  - Minimize U for the case.
  - Show that the minimum U is $n(2^{1/n} - 1)$.

Proof of Theorem 2 ($n = 2$)

task starts at the same time tao0 higher priority

- Let $\Gamma = \{ \tau_0, \tau_1 \}$ where $\phi_0 = \phi_1 = 0$ and $T_0 < T_1$.
- Before the first deadline of τ_1 (at T_1), Jobs (= task instances) of τ_0 are released $\lceil T_1/T_0 \rceil$ times.
- There are two cases whether the $\lceil T_1/T_0 \rceil$-th job of τ_0 can complete before T_1 (case 1) or not (case 2).
  - case 1: $C_0 < T_1 - \lfloor T_1/T_0 \rfloor T_0$
  - case 2: $C_0 \geq T_1 - \lfloor T_1/T_0 \rfloor T_0$
    - note: $\lfloor T_1/T_0 \rfloor T_0$ is the release time of $\lceil T_1/T_0 \rceil$-th job of τ_0.

Cases 1 and 2


case 1: $C_0 < T_1 - kT_0$
$(k = \lfloor T_1/T_0 \rfloor)$
C_0
τ_1
τ_0
0
T_0
$2T_0$
$(k\text{-}1)T_0$
kT_0
T_1
$(k+1)T_0$
$\lceil T_1/T_0 \rceil$-th job of τ_0
case 2: $C_0 \geq T_1 - kT_0$

- case 1: $C_0 < T_1 - \lfloor T_1/T_0 \rfloor T_0$
  - The maximum time $C_1^{\max}$ that τ_1 can use the processor can be represented as

    $$C_1^{\max} = T_1 - \lceil T_1/T_0 \rceil C_0.$$

  - In this case we have

    $$U = 1 + (1/T_0 - \lceil T_1/T_0 \rceil / T_1) C_0$$

    that is monotonically decreasing with C_0 ($\because T_1/T_0 \leq \lceil T_1/T_0 \rceil$).
- case 2: $C_0 \geq T_1 - \lfloor T_1/T_0 \rfloor T_0$
  - The maximum time that τ_1 can use the processor can be represented as

    $$C_1^{\max} = \lfloor T_1/T_0 \rfloor (T_0 - C_0).$$

  - In this case we have

    $$U = \lfloor T_1/T_0 \rfloor T_0/T_1 + (1/T_0 - \lfloor T_1/T_0 \rfloor / T_1) C_0$$

    that is monotonically increasing with C_0 ($\because T_1/T_0 \geq \lfloor T_1/T_0 \rfloor$).

- U becomes the minimum at the border of the two cases, i.e.,

  $$C_0 = T_1 - \lfloor T_1/T_0 \rfloor T_0.$$

- In this case we have

  $$U = 1 - (\lceil T_1/T_0 \rceil - T_1/T_0)(T_1/T_0 - \lfloor T_1/T_0 \rfloor)T_0/T_1.$$

- This can be written as

  $$U = 1 - f(1 - f)/(I + f)$$

  where $\underbrace{I}_{\text{integrated part}} = \lfloor T_1/T_0 \rfloor$ and $\underbrace{f}_{\text{fractional part}} = T_1/T_0 - \lfloor T_1/T_0 \rfloor$.

- Note: I and f correspond to the integral part and fractional part of T_1/T_0 respectively. We can change I and f independently.

- $I \geq 1$ ($\because T_0 < T_1$) and U is monotonically increasing with I. Thus U becomes the minimum when $I = 1$.
- Now we have $U = 1 - f(1 - f)/(1 + f)$. Under the restriction of $0 \leq f < 1$, we can minimize U when $f = 2^{1/2} - 1$.
- Then we can obtain the minimum of U as $U = 2(2^{1/2} - 1)$.

prove the minimum of U by differentiation

$C_0 < T_1 - T_0 x$

$C_0 \geq T_1 - T_0 x$

easy to get along

$$\frac{f(1-f) - (1-2f)(1+f)}{(1+f)^2}$$

$$U = 1 - \frac{f-f^2}{1+f}$$

$$\frac{dU}{df} = 0 - \frac{f-f^2}{1+f}$$

$$\frac{f-f^2-(1+f-2f-2f^2)}{(1+f)^2} = \frac{\cancel{f} - \cancel{f^2} - 1 - \cancel{f} + 2f + 2f^2}{(1+f)^2}$$

$$\boxed{= \frac{f^2+2f-1}{(1+f)^2}}$$

$$\frac{dU}{df} = -(f-f^2)\cdot(1+f)^{-1}$$

$$= -[(1-2f)\cdot(1+f)^{-1} + (f-f^2)\cdot(-1)(1+f)^{-2}]$$

$$= -\left[\frac{1-2f}{1+f} - \frac{f-f^2}{(1+f)^2}\right] = \frac{f-f^2}{(1+f)^2} - \frac{1-2f}{1+f} = \frac{f(1-f)}{(1+f)^2} - \frac{1-2f}{1+f}$$

Why U is minimized when $f=2^{1/2}-1$?

- Differentiate U with respect to f.

$$U' = \frac{dU}{df} = \frac{f^2 + 2f - 1}{(1+f)^2}, \quad U'' = \frac{d^2U}{df^2} = \frac{4}{(1+f)^3}$$

- $U=1$ when $f=0$ or 1 and $U''>0$ under $0 \leq f < 1$. Thus U is convex downward on $[0,1]$. $U'=0$ iff $f^2+2f-1=0$. Under $0 \leq f < 1$, we have $f=2^{1/2}-1$.

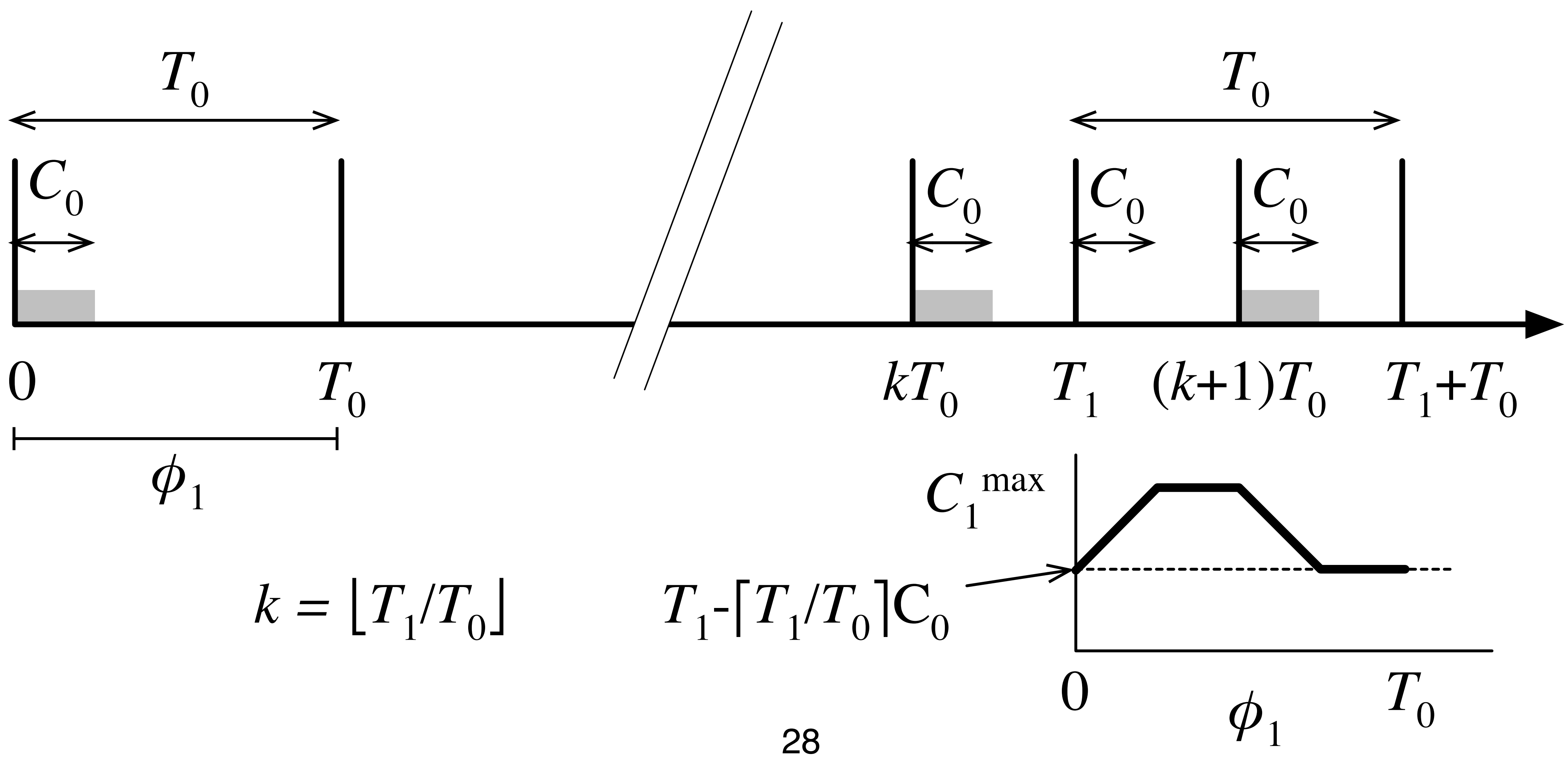
Question 1

- Is Γ schedulable with RM when $U > n(2^{1/n} - 1)$?
- Yes. $U \leq n(2^{1/n} - 1)$ is a sufficient condition of the schedulability of Γ. We can find a set of tasks whose processor utilization factor is greater than $n(2^{1/n} - 1)$.
  - Quiz: Try to find such a set when $n = 2$.

Question 2

- What happens if $\phi_0 = \phi_1 = 0$ does not hold?
- Suppose that $\phi_0 = 0$ and $0 \leq \phi_1 < T_0$ (wlog).
- Consider the following cases:
  - case 1: $C_0 < T_1 - \lfloor T_1/T_0 \rfloor T_0$
    - (a) $T_1 + C_0 < (\lfloor T_1/T_0 \rfloor + 1)T_0$
    - (b) $T_1 + C_0 \geq (\lfloor T_1/T_0 \rfloor + 1)T_0$
  - case 2: $C_0 \geq T_1 - \lfloor T_1/T_0 \rfloor T_0$
    - (a) $T_1 + C_0 < (\lfloor T_1/T_0 \rfloor + 1)T_0$
    - (b) $T_1 + C_0 \geq (\lfloor T_1/T_0 \rfloor + 1)T_0$

- case 1(a): ($k = \lfloor T_1/T_0 \rfloor$)
  - $0 \le \phi_1 < C_0 \Rightarrow C_1^{\max} = T_1 - \lceil T_1/T_0 \rceil C_0 + \phi_1$
  - $C_0 \le \phi_1 < (k+1)T_0 - T_1 \Rightarrow C_1^{\max} = T_1 - \lceil T_1/T_0 \rceil C_0 + C_0$
  - $(k+1)T_0 - T_1 \le \phi_1 < (k+1)T_0 - T_1 + C_0 \Rightarrow$
    $C_1^{\max} = \lceil T_1/T_0 \rceil (T_0 - C_0) + C_0 - \phi_1$
  - $(k+1)T_0 - T_1 + C_0 \le \phi_1 < T_0 \Rightarrow$
    $C_1^{\max} = T_1 - \lceil T_1/T_0 \rceil C_0 + C_0$



- Thus for case 1(a), the minimum of C_1^{max} is $T_1 - \lceil T_1/T_0 \rceil C_0$.
- By applying similar arguments for other cases, we have the minimums of C_1^{max} as follows.
  - $C_1^{max} = T_1 - \lceil T_1/T_0 \rceil C_0$ (case 1),
  - $C_1^{max} = \lfloor T_1/T_0 \rfloor T_0 - \lfloor T_1/T_0 \rfloor C_0$ (case 2).
- These minimums are the same as the values obtained when $\phi_1 = 0$.

- Q. Does the above argument cover the case when $T_1/T_0 = \lceil T_1/T_0 \rceil = \lfloor T_1/T_0 \rfloor$?
- A. Yes. The case 2(a) covers that.

Critical Instant

- Consider that we have n tasks. The worst case response times (WCRTs) are obtained when the all tasks are released simultaneously.
- This result implies that if we can have a feasible schedule of n tasks when $\phi_0 = \phi_1 = ... = \phi_{n\text{-}1} = 0$, the set of tasks are schedulable.

Priority Assignment

- Let $\Gamma = \{ \tau_0, \tau_1 \}$ where $\phi_0 = \phi_1 = 0$ and $T_0 < T_1$.
- Consider the following two scheduling policies.
  - A_0: $p(\tau_0) > p(\tau_1)$
  - A_1: $p(\tau_0) < p(\tau_1)$
- Theorem 3: If Γ is schedulable with A_1, then it is schedulable with A_0.

- Note: This theorem justifies that RM is optimal.

- Proof:
  - It is easy to see that
    - Γ is schedulable w/ $A_0 \Leftrightarrow \lfloor T_1/T_0 \rfloor C_0 + C_1 \le T_1$ and
    - Γ is schedulable w/ $A_1 \Leftrightarrow C_0 + C_1 \le T_0$.
  - Thus, the theorem is equivalent to
    $C_0 + C_1 \le T_0 \Rightarrow \lfloor T_1/T_0 \rfloor C_0 + C_1 \le T_1$.
    - $C_0 + C_1 \le T_0$
      $\Rightarrow C_1 \le T_0 - C_0$
      $\Rightarrow C_1 \le \lfloor T_1/T_0 \rfloor T_0 - \lfloor T_1/T_0 \rfloor C_0 \quad (\because \lfloor T_1/T_0 \rfloor \ge 1)$
      $\Rightarrow C_1 \le \lfloor T_1/T_0 \rfloor (T_0/T_1) T_1 - \lfloor T_1/T_0 \rfloor C_0$
      $\Rightarrow C_1 \le T_1 - \lfloor T_1/T_0 \rfloor C_0 \quad (\because \lfloor T_1/T_0 \rfloor \le T_1/T_0)$
      $\Rightarrow \lfloor T_1/T_0 \rfloor C_0 + C_1 \le T_1$.

Theorem 2 (n > 2)

- A set $\Gamma = \{ \tau_0, \tau_1, ..., \tau_{n\text{-}1} \}$ of periodical tasks is schedulable with RM if

$$U \leq n(2^{\frac{1}{n}} - 1)$$

where $\tau_i = (T_i, C_i, \phi_i)$ for $0 \leq i < n$ and

$$U = \sum_{i=0}^{n-1} \frac{C_i}{T_i}.$$

Lemma 3 [Liu&Layland 1973]

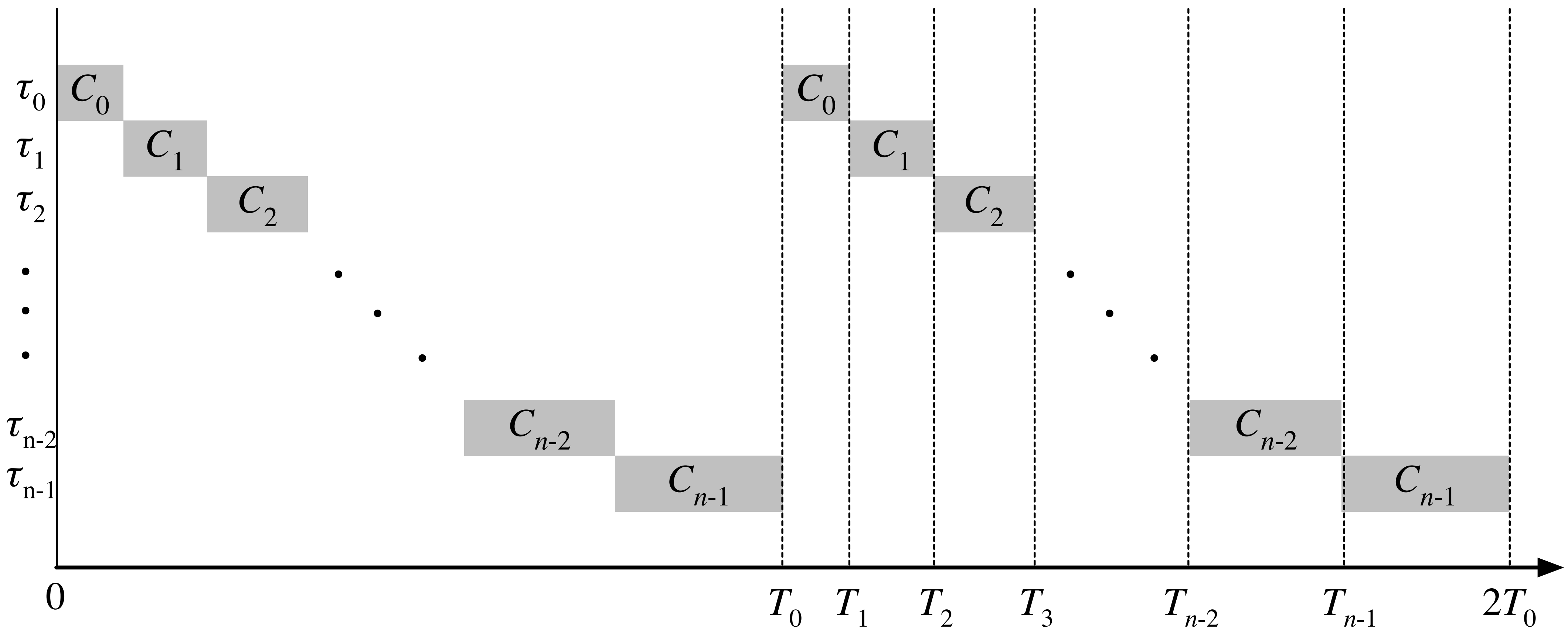
We assume that $T_0 < T_1 < T_2 < \ldots < T_{n\text{-}1}$ and $\phi_i = 0$ $(0 \leq i < n)$. To minimize the processor utilization factor with full CPU utilization (= 100% of its execution time is spent in the tasks) under RM, the following relationships should hold.

it is said that these tasks fully utilize the processor if the priority assignment is feasible and an increase in the run time of any of the tasks in the set will make the priority assignment infeasible

$$T_{n-1} < 2T_0 \tag{A}$$

$$C_i = T_{i+1} - T_i \quad (0 \leq i < n-1) \tag{B}$$

$$C_{n-1} = 2T_0 - T_{n-1} \tag{C}$$



τ_0
τ_1
τ_2
τ_{n-2}
τ_{n-1}
C_0
C_1
C_2
C_{n-2}
C_{n-1}
C_0
C_1
C_2
C_{n-2}
C_{n-1}
0
T_0
T_1
T_2
T_3
T_{n-2}
T_{n-1}
$2T_0$

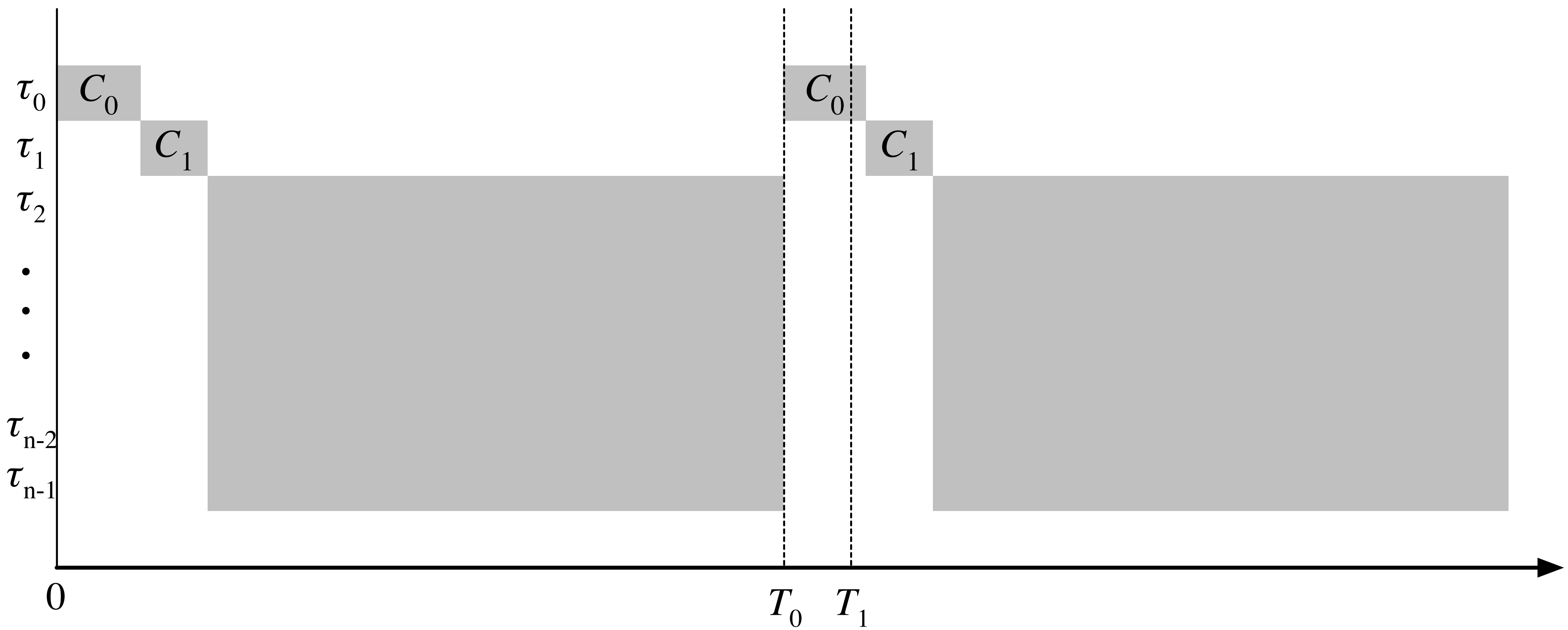
Proof of Lemma 3 (1/4)

- We assume that $T_{n-1} < 2T_0$.   (A)
- Let $C_0, \ldots, C_{n-1}$ be the execution times that fully utilize the processor and minimize the processor utilization factor. Let U be the (minimized) processor utilization factor.
- First, we show that $C_0 = T_1 - T_0$ using proof by contradiction. We will show that we can construct $U' < U$ if $C_0 \neq T_1 - T_0$.

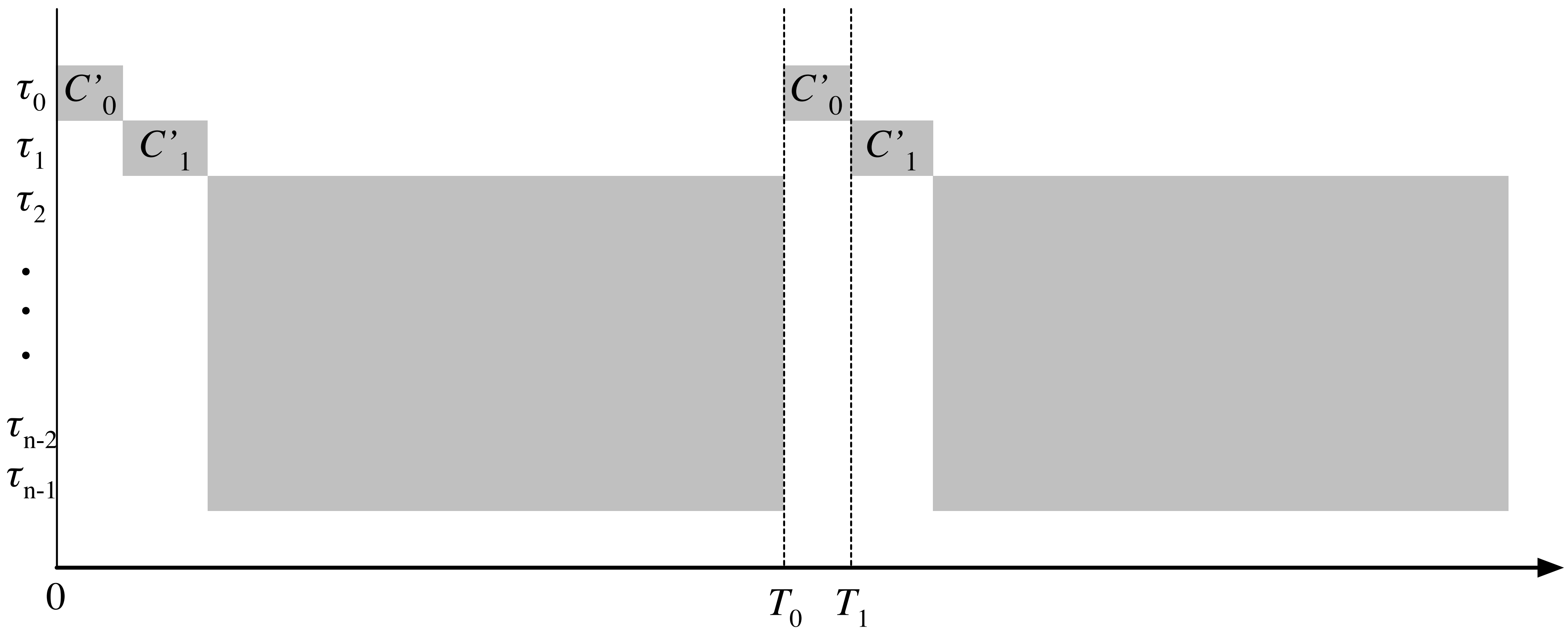
Proof of Lemma 3 (2/4)

- Suppose that $C_0 > T_1 - T_0$. This can be written as $C_0 = T_1 - T_0 + \varDelta$ ($\varDelta > 0$).
- Define $C'_0, ..., C'_{n\text{-}1}$ as follows.
  - $C'_0 = T_1 - T_0$
  - $C'_1 = C_1 + \varDelta$
  - $C'_i = C_i$ $(2 \leq i < n)$
- $C'_0, ..., C'_{n\text{-}1}$ also fully utilize the processor.
- Let U' be the corresponding utilization factor. We have $U - U' = \varDelta / T_0 - \varDelta / T_1 > 0$. This contradicts the minimality of U.
- Thus $C_0 \leq T_1 - T_0$.

$$C_0 = T_1 - T_0 + \Delta$$



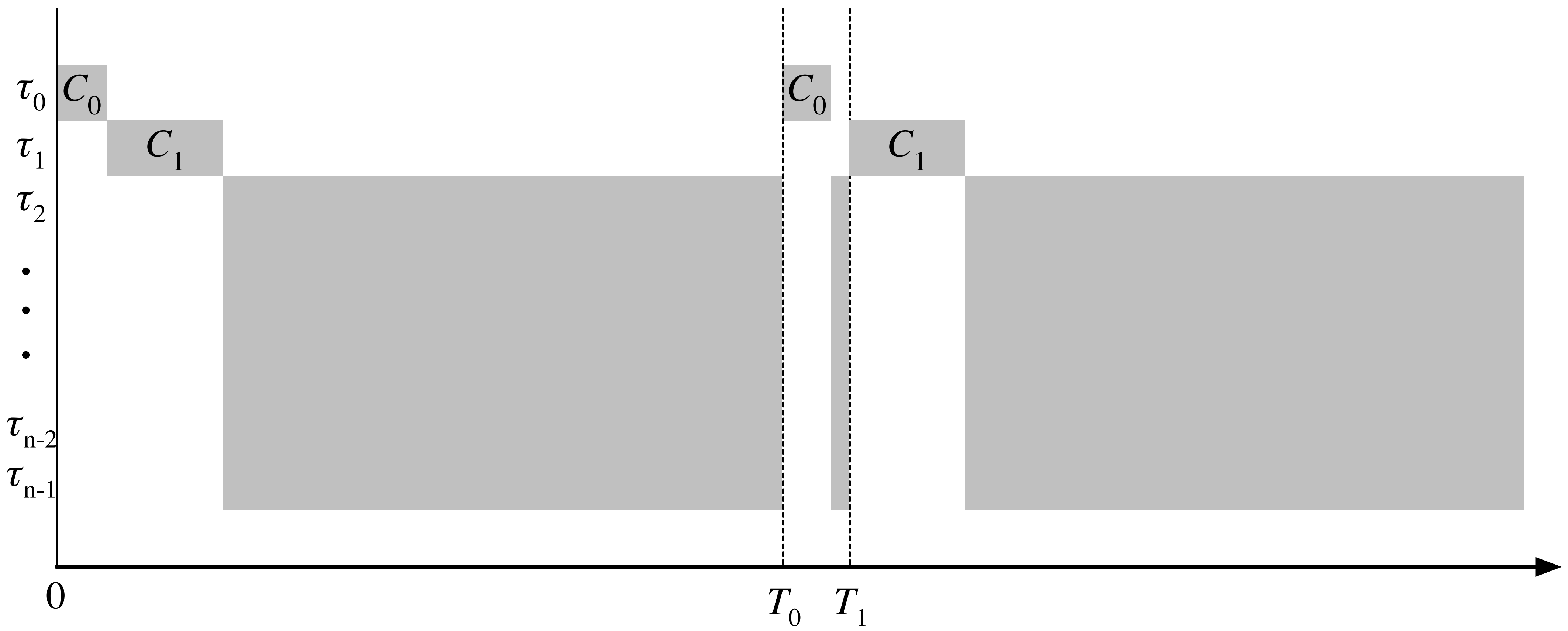
$$C'_0 = C_0 - \Delta \;\; = T_1 - T_0$$
$$C'_1 = C_1 + \Delta$$



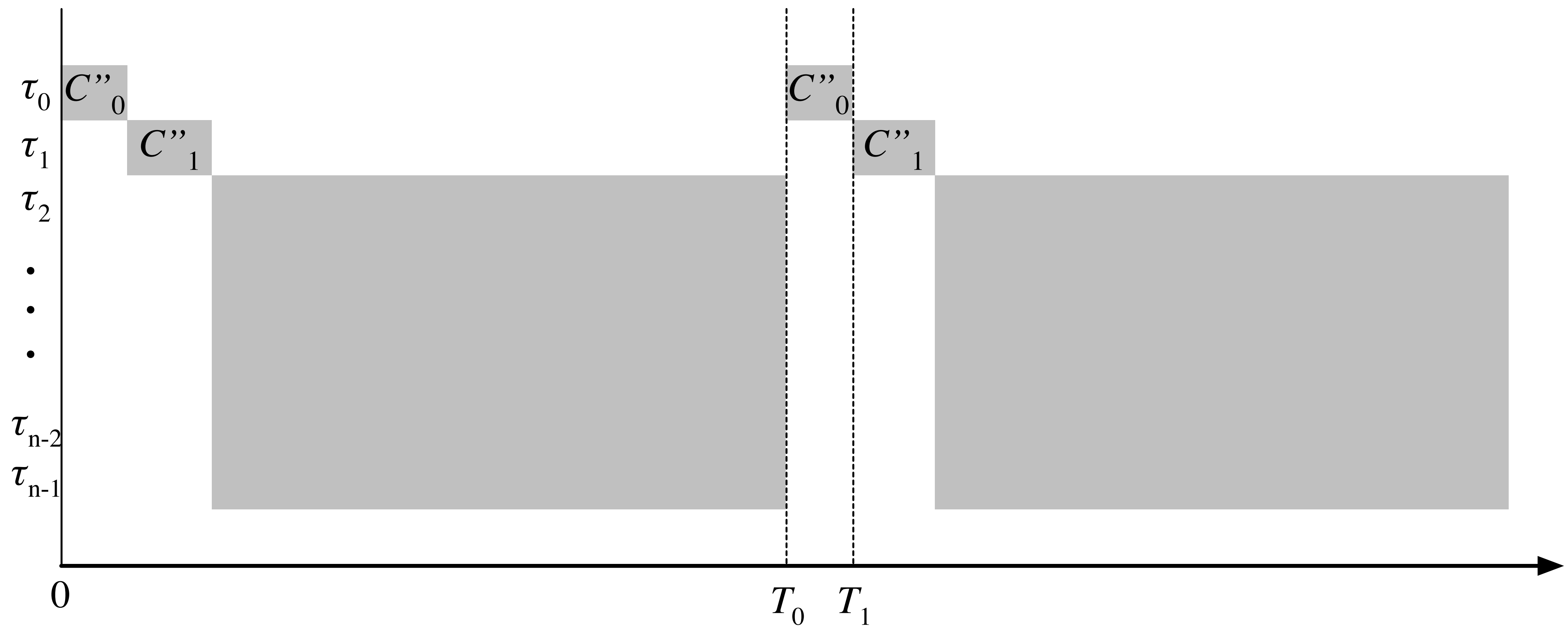
Proof of Lemma 3 (3/4)

- Suppose that $C_0 < T_1 - T_0$. This can be written as $C_0 = T_1 - T_0 - \Delta$ ($\Delta > 0$).
- Define $C''_0, \ldots, C''_{n-1}$ as follows.
  - $C''_0 = T_1 - T_0$
  - $C''_1 = C_1 - 2\Delta$
  - $C''_i = C_i$ $(2 \leq i < n)$
- $C''_0, \ldots, C''_{n-1}$ also fully utilize the processor.
- Let U'' be the corresponding utilization factor. We have $U - U'' = -\Delta / T_0 + 2\Delta / T_1 > 0$ ($\because$ A). This contradicts the minimality of U.
- Thus $C_0 = T_1 - T_0$.

$$C_0 = T_1 - T_0 - \Delta$$



$$C''_0 = C_0 + \Delta = T_1 - T_0$$
$$C''_1 = C_1 - 2\Delta$$



Proof of Lemma 3 (4/4)

- By the similar arguments, we have

$$C_1 = T_2 - T_1$$
$$C_2 = T_3 - T_2$$
$$\vdots$$
$$C_{n-2} = T_{n-1} - T_{n-2}.$$

- We can also show the following.

$$C_{n-1} = 2T_0 - T_{n-1}. \qquad \text{(C)}$$

Proof of Theorem 2 (n > 2) (1/2)

- In the case of Lemma 3, we have

$$U = \sum_{i=0}^{n-2} \frac{T_{i+1} - T_i}{T_i} + \frac{2T_0 - T_{n-1}}{T_{n-1}}.$$

- By using $R_i = T_{i+1} / T_i$, we can express U as

$$U = \sum_{i=0}^{n-2} R_i + \frac{2}{R_0 R_1 \cdots R_{n-2}} - 1.$$

- From this we have

$$\frac{\partial U}{\partial R_i} = 1 - \frac{2}{R_i (R_0 R_1 \cdots R_{n-2})} \qquad (0 \leq i < n-1)$$

Proof of Theorem 2 (n > 2) (2/2)

- $\partial U/\partial R_i = 0$ if $R_i(R_0 R_1 \ldots R_{n-2}) = 2$ $(0 \le i < n-1)$. By multiplying these equations, we obtain

$$R_0 R_1 \cdots R_{n-2} = 2^{\frac{n-1}{n}}.$$

- Thus U can be minimized when $R_i = 2^{1/n}$. By using this, we have

$$U = n(2^{\frac{1}{n}} - 1).$$

Reference

1. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment
    - C. L. Liu and James W. Layland
    - Journal of the ACM, 20(1), pp. 46-61, 1973.
    - http://dx.doi.org/10.1145/321738.321743
2. The Deferrable Server Algorithm for Enhanced Aperiodic Responsiveness in Hard Real-Time Environments
    - Jay K. Strosnider and John P. Lehoczky and Lui Sha
    - IEEE Transactions on Computers, 44(1), pp. 73-91, 1995.
    - http://dx.doi.org/10.1109/12.368008

Assignment #1

1. Can you give an RM-schedulable set of tasks whose processor utilization factor is greater than $n(2^{1/n} - 1)$?
2. Can the proof used in the general case of Theorem 2 $(n > 2)$ be used to prove the $n = 2$ case? just sufficient condition
3. Show (C) of Lemma 3. c1 to c(n-2)
4. (Optional) show (A) of Lemma 3.
   - See Reference 2.

- Submit your answer via OCW-i by Nov. 9.
   - This assignment is NOT team-based. Everyone who want to take a credit should submit.