


Multiprocessor Real-Time Scheduling

Embedded System Software Design

Prof. Ya-Shu Chen
National Taiwan University of
Science and Technology

Outline

- Multiprocessor Real-Time Scheduling
- Global Scheduling *不能 partitioned 非 Global*
- Partitioned Scheduling
- Semi-partitioned Scheduling 

Multiprocessor Models

→ Global 每顆 CPU exe time - 一樣

- **Identical (Homogeneous):** All the processors have the same characteristics, i.e., the execution time of a job is independent on the processor it is executed. 相同（均質）：所有處理器具有相同的特性，即，作業的執行時間與執行的處理器無關。
- **Uniform:** Each processor has its own speed, i.e., the execution time of a job on a processor is proportional to the speed of the processor. 統一：每個處理器都有自己的速度，即，作業在處理器上的執行時間與處理器的速度成正比。
 - A faster processor always executes a job faster than slow processors do. – 速度較快的處理器總是比速度較慢的處理器更快地執行作業。
 - For example, multiprocessors with the same instruction set but with different supply voltages/frequencies. – 例如，具有相同指令集但電源電壓/頻率不同的多處理器。
- **Unrelated (Heterogeneous):** Each job has its own execution time on a specified processor partitioned power 不一樣
不相關（異構）：每個作業在指定處理器上都有自己的執行時間
 - A job might be executed faster on a processor, but other jobs might be slower on that processor. – 作業在處理器上的執行速度可能更快，但其他作業在該處理器上的執行速度可能會較慢。
 - For example, multiprocessors with different instruction sets.
– 例如，具有不同指令集的多處理器。

Scheduling Models

可以任意 process 執行

全局調度：

- 作業可以在任何處理器上執行。
- 系統維護全局就緒隊列。
- 在就緒隊列中執行M個優先級最高的作業，其中M是處理器數量。
- 它要求高昂的在線開銷。

- **Global Scheduling:**
 - A job may execute on any processor.
 - The system maintains a global ready queue.
 - Execute the M highest-priority jobs in the ready queue, where M is the number of processors.
 - It requires high on-line overhead.
- **Partitioned Scheduling:**
 - Each task is assigned on a dedicated processor.
 - Schedulability is done individually on each processor.
 - It requires no additional on-line overhead.
- **Semi-partitioned Scheduling:**
 - Adopt task partitioning first and reserve time slots (bandwidths) for tasks that allow migration.
 - It requires some on-line overhead.

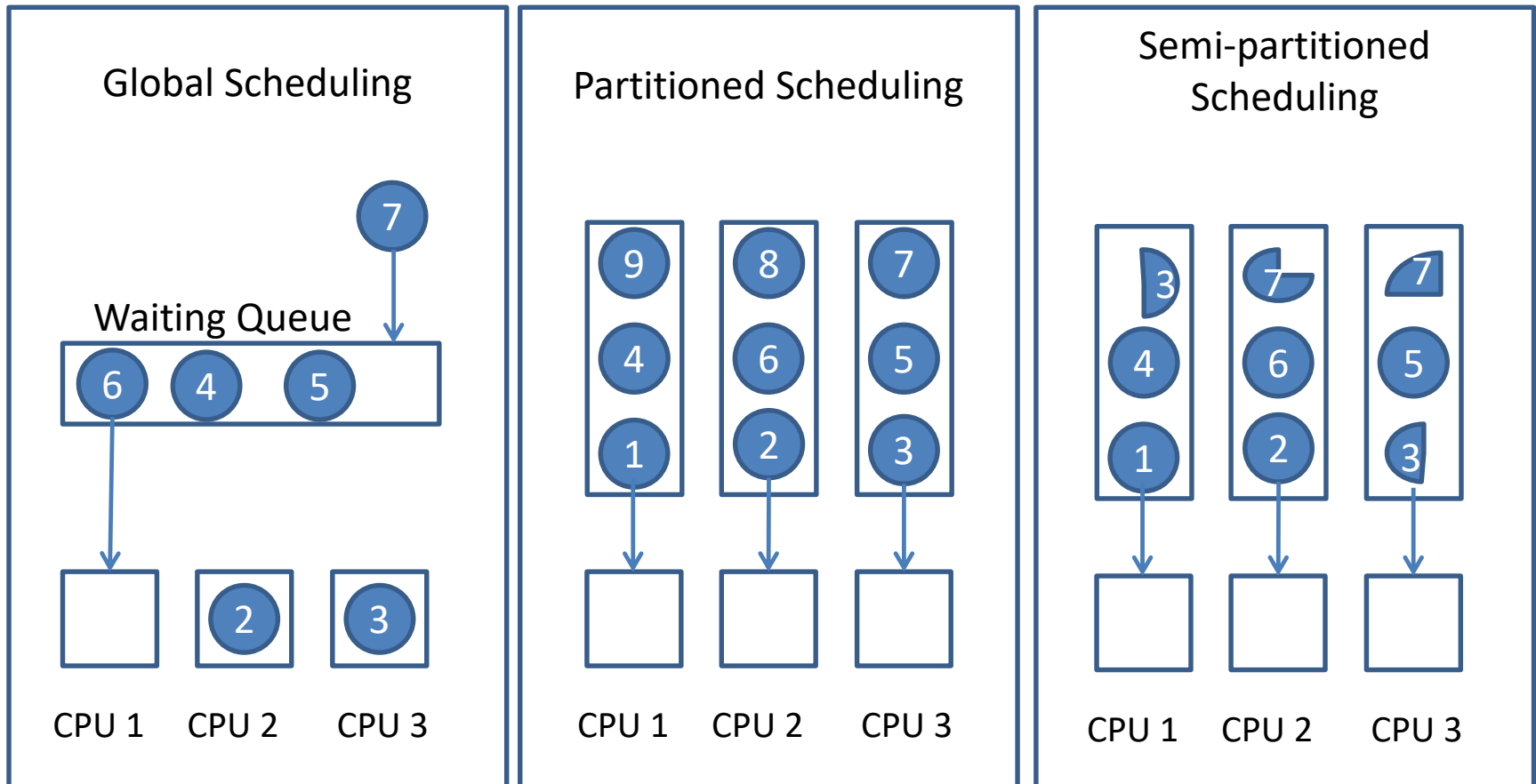
分區調度：

- 每個任務都在專用處理器上分配。
- 可調度性是在每個處理器上單獨完成的。
- 不需要額外的在線開銷。

半分區調度：

- 首先採用任務分區，並為允許遷移的任務保留時隙（帶寬）。
- 這需要一些在線開銷。

Scheduling Models



Global Scheduling

所有準備好的任務都保存在全局隊列中

- All ready tasks are kept in a global queue
- A job can be migrated to any processor. 作業可以遷移到任何處理器。
- Priority-based global scheduling: 基於優先級的全局調度：
 - 在全局隊列中的作業中，選擇M個優先級最高的作業以在M個處理器上執行。
 - 假定此處的任務遷移沒有任何開銷。
 - Among the jobs in the global queue, the M highest priority jobs are chosen to be executed on M processors.
 - Task migration here is assumed with no overhead.
- Global-EDF: When a job finishes or arrives to the global queue, the M jobs in the queue with the shortest absolute deadlines are chosen to be executed on M processors. 全局-EDF：當作業完成或到達全局隊列時，將選擇絕對期限最短的隊列中的M個作業在M個處理器上執行。
- Global-RM: When a job finishes or arrives to the global queue, the M jobs in the queue with the highest priorities are chosen to be executed on M processors.

Global-RM：當作業完成或到達全局隊列時，將選擇隊列中優先級最高的M個作業在M個處理器上執行。

Global Scheduling

好處：

- Advantages:
 - 有效利用處理資源（如果可行）
 - 未使用的處理器時間可以在運行時輕鬆回收（硬RT和軟RT任務的混合以優化資源利用率）
 - Effective utilization of processing resources (if it works)
 - Unused processor time can easily be reclaimed at run-time (mixture of hard and soft RT tasks to optimize resource utilization)

↑ 有空閒時做

缺點：

- Disadvantages:
 - 在某些情況下，添加處理器並減少計算時間和其他參數實際上會降低最佳性能！
 - 由於嚴格的時間限制，資源利用不佳
 - 單處理器調度的結果很少可以使用
 - Adding processors and reducing computation times and other parameters can actually decrease optimal performance in some scenarios!
 - Poor resource utilization for hard timing constraints
 - Few results from single-processor scheduling can be used

global 在多顆 core 上跑，不能用 single 來分析

Schedule Anomaly

- **Anomaly 1**

high pro $\frac{C}{P}$ 減少 P 4→5

A decrease in processor demand from higher-priority tasks can *increase* the interference on a lower-priority task because of the change in the time when the tasks execute

由於任務執行時間的變化，較高優先級任務對處理器的需求減少會增加對較低優先級任務的干擾。

- **Anomaly 2**

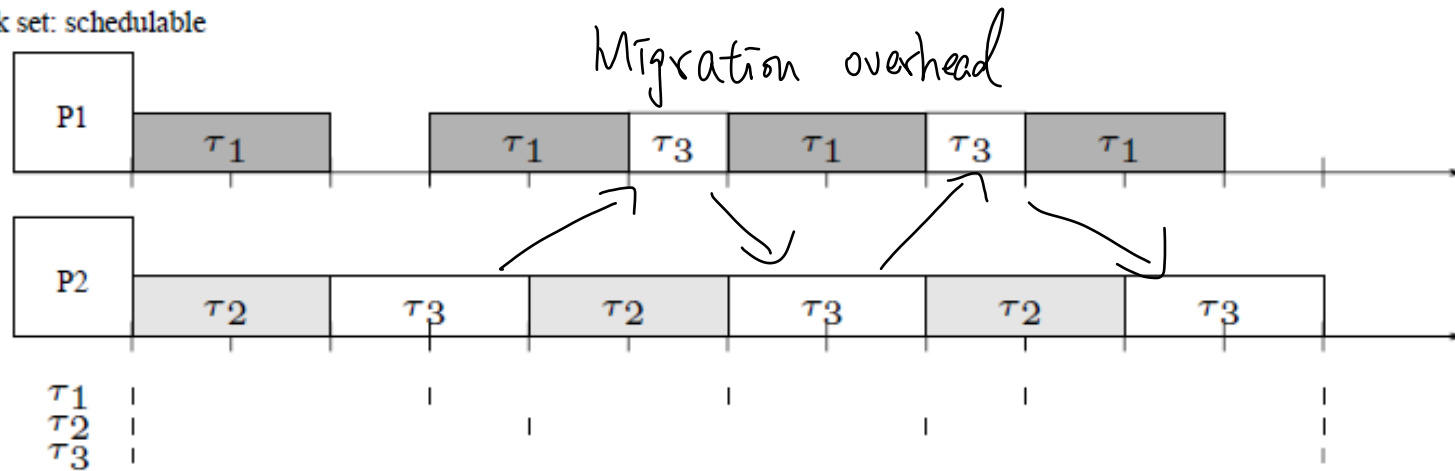
自己的processor需求減少 10→11

A decrease in processor demand of a task *negatively* affects the task itself because the change in the task arrival times make it suffer more interference

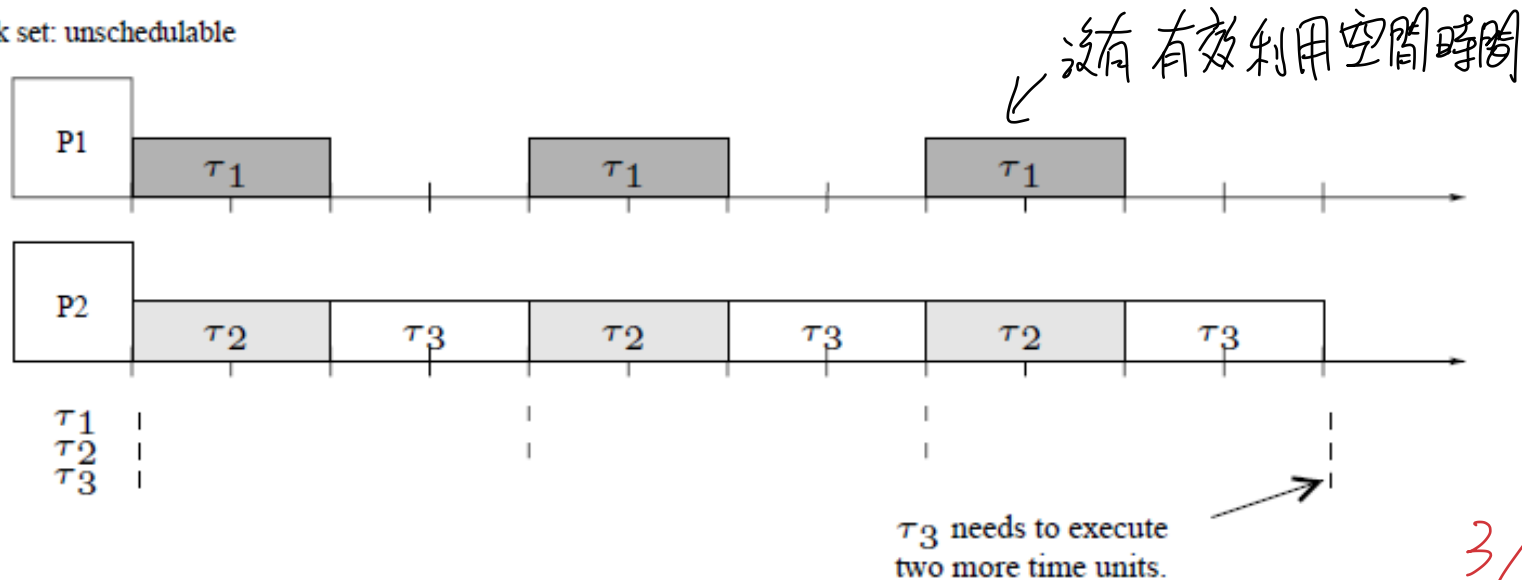
任務對處理器的需求減少會對任務本身產生負面影響，因為任務到達時間的變化使其受到更多干擾

Anomaly 1

task set: schedulable



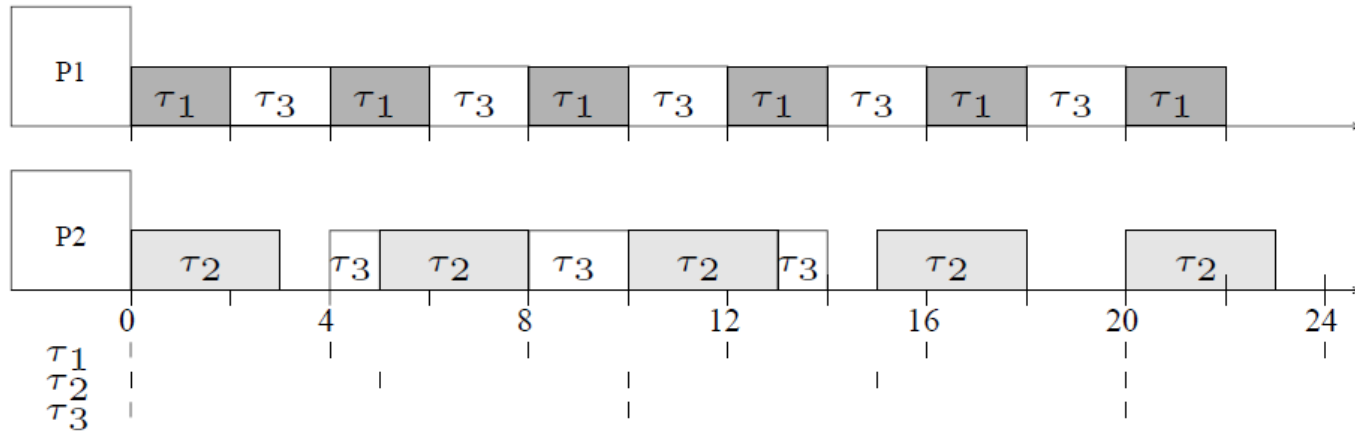
task set: unschedulable



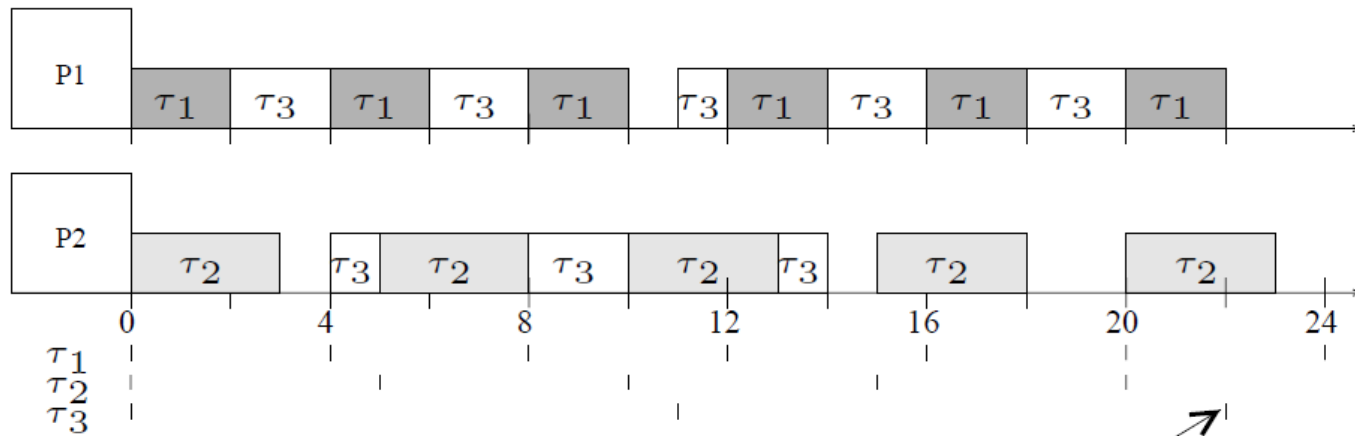
3/24

Anomaly 2

task set: schedulable



task set: unschedulable



τ_3 needs to execute
one more time unit.

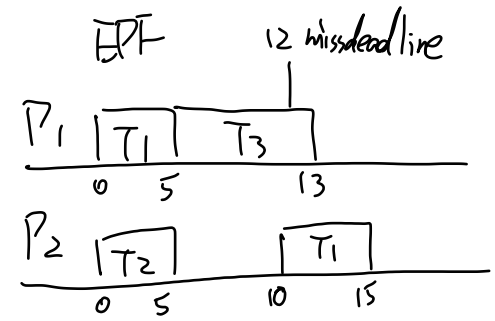
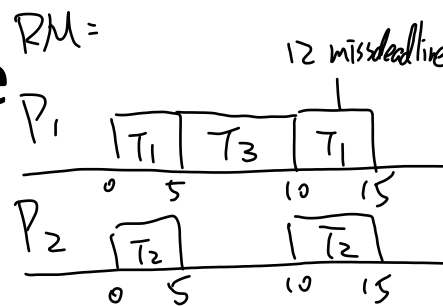
Dhall effect

Dhall效應：對於Global-EDF或Global-RM，可調度性分析的最小上限為1。.

- Dhall effect : For Global-EDF or Global-RM, the least upper bound for schedulability analysis is at most 1.
- On 2 processors:

Task	T	D	C	U
T1	10	10	5	0.5
T2	10	10	5	0.5
T3	12	12	8	0.67

- T3 is not schedulable



Schedulability Test

可以通過使用搶占式全局EDF在M個處理器上調度具有隱式截止日期的一組週期性任務 t_1, t_2, \dots, t_N

- A set of periodic tasks t_1, t_2, \dots, t_N with implicit deadlines is schedulable on M processors by using preemptive Global EDF scheduling if

$$\sum_{i=1}^N \frac{C_i}{T_i} \leq M(1 - \frac{C_k}{T_k}) + \frac{C_k}{T_k}$$

放在指定位子 (此 core 只能做這一個工作)

任意位子

利用率最高的任務在哪裡

where t_k is the task with the largest utilization

$$C_k/T_k \quad \frac{5}{10} + \frac{5}{10} + \frac{8}{12} \leq M(1 - \frac{8}{12}) + \frac{8}{12} \quad M \geq 3$$

全局調度的弱點

Weakness of Global Scheduling

- Migration overhead
 - 遷移開銷
- Schedule Anomaly
 - 日程異常

分區調度

Partitioned Scheduling

- Two steps:
 - 兩步：
 - 確定任務到處理器的映射
 - 執行運行時單處理器調度
 - Determine a mapping of tasks to processors
 - Perform run-time single-processor scheduling
- Partitioned with EDF
 - Assign tasks to the processors such that no processor's capacity is exceeded (utilization bounded by 1.0)
 - Schedule each processor using EDF

用EDF分區

- 將任務分配給處理器，以確保不超過處理器的容量（利用率以1.0為界）
- 使用EDF安排每個處理器

裝箱問題

Bin-packing Problem

給定箱尺寸 V 和要包裝物品的尺寸的清單 $a_1 \dots a_n$ ，找到整數 B 和 B 分區 $S_1 \cup \dots \cup S_B$ 為 $[1, \dots, n]$ ，使得對於所有 $h = 1 \dots B$ ， $\sum_{i \in S_h} a_i \leq V$ 。如果解決方案的 B 最小，則它是最佳的。

Given a bin size V and a list a_1, \dots, a_n of sizes of the items to pack, find an integer B and a B -partition $S_1 \cup \dots \cup S_B$ of $\{1, \dots, n\}$ such that $\sum_{i \in S_k} a_i \leq V$, for all $k = 1, \dots, B$.
A solution is optimal if it has minimal B .

The problem is NP-complete !!

問題是NP完全！

Bin-packing to Multiprocessor Scheduling

該問題涉及將不同尺寸的物體包裝在箱子（“箱”）中，目的是使用過的箱子的數量最小化。

- The problem concerns packing objects of varying sizes in boxes (“bins”) with the objective of minimizing number of used boxes.
 - Solutions (Heuristics): First Fit –解決方案（啟發式）：最適合應用於多處理器系統：
- Application to multiprocessor systems:
 - Bins are represented by processors and objects by tasks. –容器由處理器表示，容器由任務表示。
–處理器是否“已滿”的決定是基於基於利用率的可調度性測試得出的。
 - The decision whether a processor is “full” or not is derived from a utilization-based schedulability test.

Partitioned Scheduling

- **Advantages:**
 - 優點：
 - 大多數用於單處理器調度的技術在這裡也適用
 - Most techniques for single-processor scheduling are also applicable here
 - **Partitioning of tasks can be automated**
 - Solving a bin-packing algorithm
 - 任務分區可以自動化
 - 解決垃圾箱打包算法
 - **Disadvantages:**
 - Cannot exploit/share all unused processor time
 - May have very low utilization, bounded by 50%
- 缺點：
- 無法利用/共享所有未使用的處理器時間
 - 利用率可能很低，大約為50%

Partitioned Scheduling Problem

給定一組具有任意期限的任務，目標是確定對 M 個處理器的可行任務分配，以使所有任務均滿足其時序約束，其中 C_i 是任務 t_i 在任何處理器 m 上的執行時間。

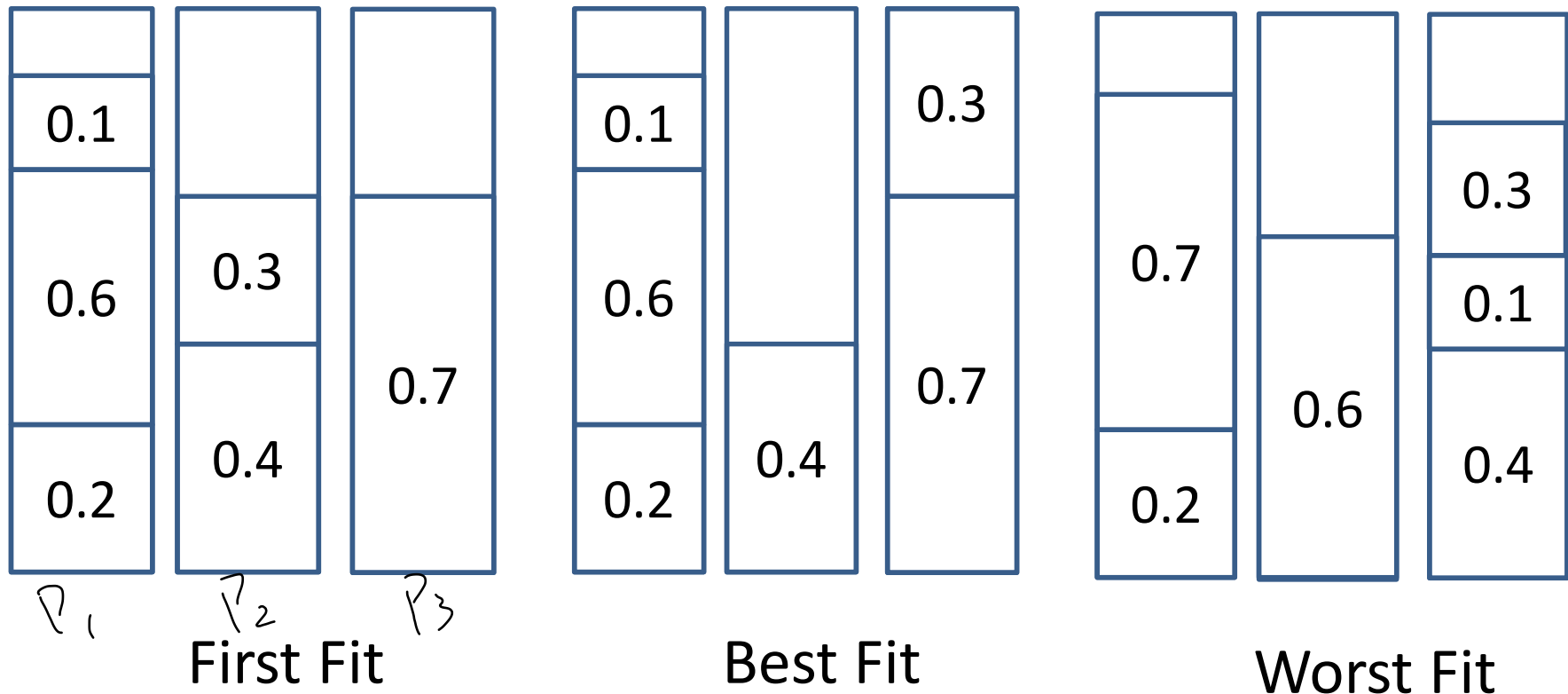
Given a set of tasks with arbitrary deadlines, the objective is to decide a feasible task assignment onto M processors such that all the tasks meet their timing constraints, where C_i is the execution time of task t_i on any processor m .

Partitioned Algorithm

- ^{index min 失敗 (最簡單的)} First-Fit: choose the one with the smallest index
- Best-Fit: choose the one with the maximal utilization ^{空間少的失敗 (盡量放滿, 容易追加)}
- Worst-Fit: choose the one with the minimal utilization ^{空間多的失敗 (平均放, 但不好追加)}

Partitioned Example

- 0.2 -> 0.6 -> 0.4 -> 0.7 -> 0.1 -> 0.3



EDF with First Fit

$n=3$ $U(0.5, 0.5, 0.67)$

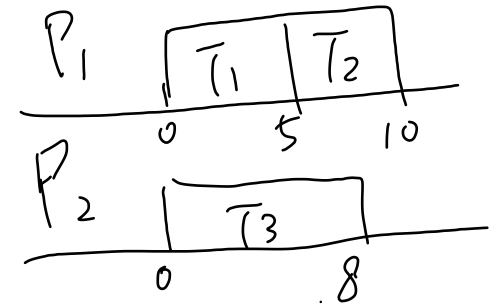
Input: A task set $\tau = \{\tau_1, \tau_2, \dots, \tau_n\}$ and a set of processors $\{p_1, \dots, p_m\}$
Output: j ; number of processors required.

```

1.   $i := 1; j := 1; k_q = 0; (\forall_q)$ 
2.  while ( $i \leq n$ ) do
3.       $q := 1;$ 
4.      while ( $(U_q + u_i) > 1$ ) do
5.           $q := q + 1; /* \text{increase the processor index} */$ 
6.           $U_q := U_q + u_i; k_q := k_q + 1;$ 
7.          if ( $q > j$ ) then
8.               $j := q;$ 
9.           $i := i + 1;$ 
10. return ( $j$ );
11. end

```

EDF \rightarrow RM
 $U_q + u_i > 1$ \rightarrow $1 - \frac{1}{2} - \frac{1}{2}$



Schedulability Test

Lopez [3]證明，在EDF調度和FF分配（EDF-FF）的最壞情況下可達到的利用率為
Lopez [3] proves that the **worst-case achievable utilization** for
EDF scheduling and FF allocation (EDF-FF) takes the value

如果所有任務的利用率 C/T 都在 α 值以下，則 m 是處理器數
If all the tasks have an utilization factor C/T under a value α , where
 m is the number of processors

$$U_{wc}^{EDF-FF}(m, \beta) = \frac{\beta m + 1}{\beta + 1}$$

where $\beta = \lfloor 1/\alpha \rfloor$

$$T_1 = \frac{5}{10} \quad \beta = 2 \quad U = \frac{4+1}{2+1} = \frac{5}{3}$$

$$T_2 = \frac{5}{10} \quad \beta = 2 \quad U = \frac{5}{3}$$

$$T_3 = \frac{8}{12} \quad \beta = 1 \quad U = \frac{2+1}{2} = \frac{3}{2}$$

$$\alpha \nmid \max\left(\frac{C}{T}\right)$$

需求約束功能

如果計算資源足夠,則拿來使用

非週期性工作

Demand Bound Function

將需求約束函數 $dbf(\tau_i, t)$ 定義為

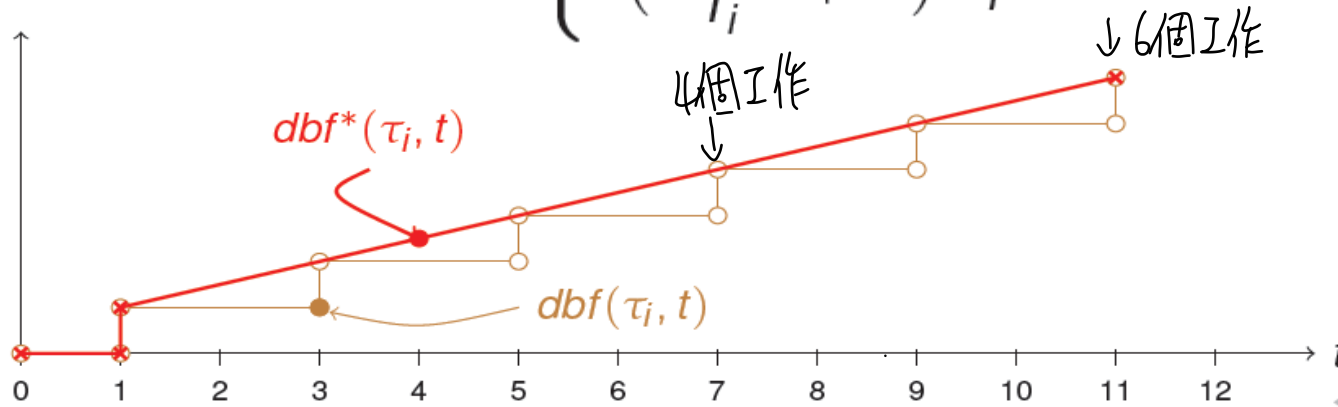
T : 週期 D : Deadline

- Define demand bound function $dbf(\tau_i, t)$ as
給一個 t (任意時間, low pro Deadline), τ_i 需要多少執行單位 (High pro 來殺?)
 $dbf(\tau_i, t) = \max \left\{ 0, \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor \right\} C_i = \max \left\{ 0, \left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1 \right\} C_i.$

我們需要近似來執行多項式時間可調度性測試

- We need approximation to enforce polynomial-time schedulability test

$$dbf^*(\tau_i, t) = \begin{cases} 0 & \text{if } t < D_i \\ (\frac{t - D_i}{T_i} + 1) C_i & \text{otherwise.} \end{cases}$$



Deadline Monotonic Partition

Input: T, M ;

- 1: re-index (sort) tasks such that $D_i \leq D_j$ for $i < j$;
- 2: $T_i \leftarrow \emptyset, U_i \leftarrow 0, \forall m = 1, 2, \dots, M$;
- 3: **for** $i = 1$ to N , where $N = |T|$ **do**
- 4: **for** $m = 1$ to M **do**
- 5: **if** $\frac{C_i}{T_i} + \sum_{\tau_j \in T_m} \frac{C_j}{T_j} \leq 1$ and $\underbrace{C_i + \sum_{\tau_j \in T_m} dbf^*(\tau_j, D_i)}_{\text{自己執行時間} + \text{deadline 比我短的所花時間}} \leq D_i$ **then** 我可以提供的時間
↓
- 6: assign task τ_i onto processor m and $T_m \leftarrow T_m \cup \{\tau_i\}$;
- 7: break;
- 8: **if** τ_i is not assigned **then**
- 9: return "The task assignment fails";
- 10: return feasible task assignment T_1, T_2, \dots, T_M ;

可計劃性檢驗

Schedulability Test

定理4任何零星的任務系統 τ 都可以由Algorithm PARTITION在 m 個單位容量的處理器上成功調度，對於任何

Theorem 4 *Any sporadic task system τ is successfully scheduled by Algorithm PARTITION on m unit-capacity processors, for any*

$$m \geq \left(\frac{2\delta_{\text{sum}} - \delta_{\text{max}}}{1 - \delta_{\text{max}}} + \frac{u_{\text{sum}} - u_{\text{max}}}{1 - u_{\text{max}}} \right) \quad (14)$$

$$\begin{aligned} \delta_{\text{max}} &\stackrel{\text{def}}{=} \max_{i=1}^n (e_i/d_i) & u_{\text{max}} &\stackrel{\text{def}}{=} \max_{i=1}^n (u_i) \\ \delta_{\text{sum}} &\stackrel{\text{def}}{=} \max_{t>0} \left(\frac{\sum_{j=1}^n \text{DBF}(\tau_j, t)}{t} \right) & u_{\text{sum}} &\stackrel{\text{def}}{=} \sum_{j=1}^n u_j \end{aligned}$$

Weakness of Partitioned Scheduling

- Restricting a task on a processor reduces the schedulability 在處理器上限制任務會降低可調度性
- Restricting a task on a processor makes the problem NP-hard 限制處理器上的任務使問題變得困難
- Example: Suppose that there are M processors and $M + 1$ tasks with the same period T and the (worst-case) execution times of all these $M + 1$ tasks are $T/2 + e$ with $e > 0$
 - With partitioned scheduling, it is not schedulable

示例：假設有 M 個處理器和 $M + 1$ 個任務具有相同的周期 T ，並且所有這 $M + 1$ 個任務的（最壞情況）執行時間是 $T/2 + e$ ，其中 $e > 0$

–使用分區調度時，它是不可調度的

ex: $U_1=0.5$ | $U_2=0.5$ | $U_3=0.5$

 U_1 U_2 U_3 排不進去

Semi-partitioned Scheduling

首先將任務劃分到處理器中

- Tasks are first partitioned into processor.

為了降低利用率，我們再次選擇具有最低任務利用率的處理器

- To reduce the utilization, we again pick the processor with the minimum task utilization

如果任務無法放入所選擇的處理器中，我們將不得不將其拆分為多個（兩個甚至更多個）部分。

- If a task cannot fit into the picked processor, we will have to split it into multiple (two or even more) parts.

- If t_i is split and assigned to a processor m and the utilization on processor m after assigning t_i is at most $U(\text{scheduler}, N)$, then t_i is so far schedulable.

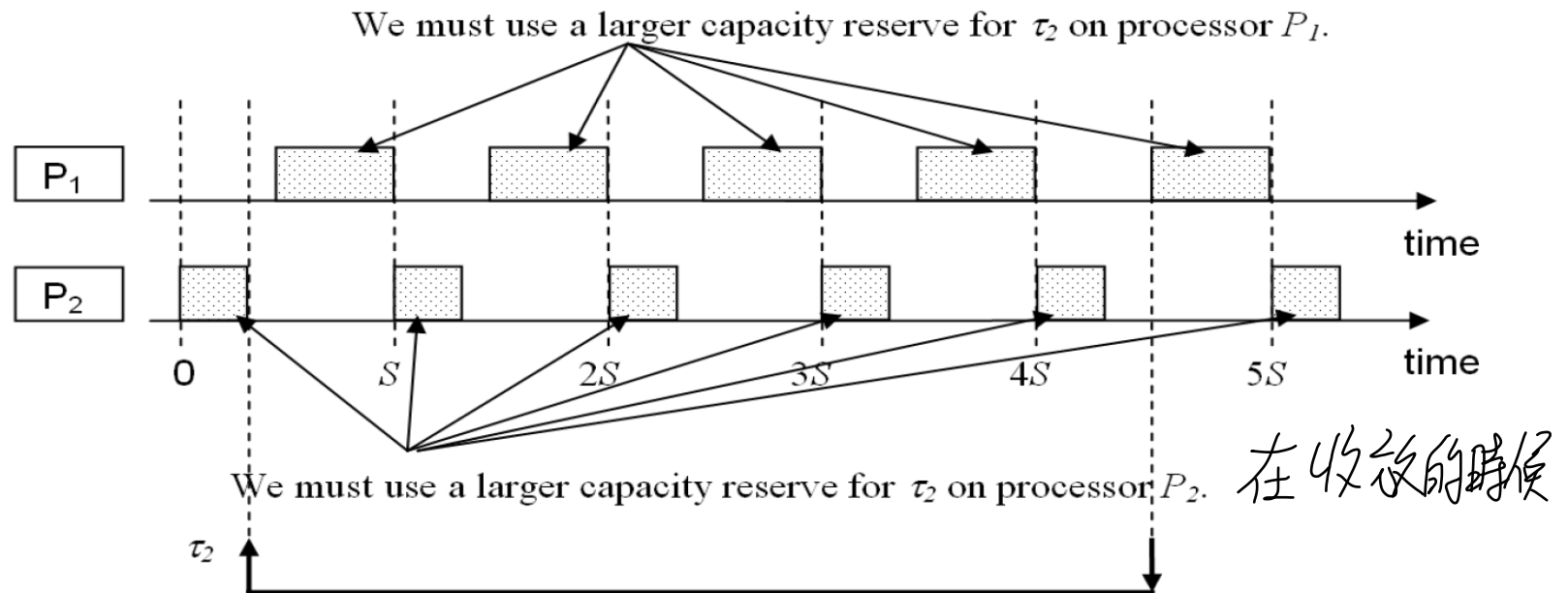
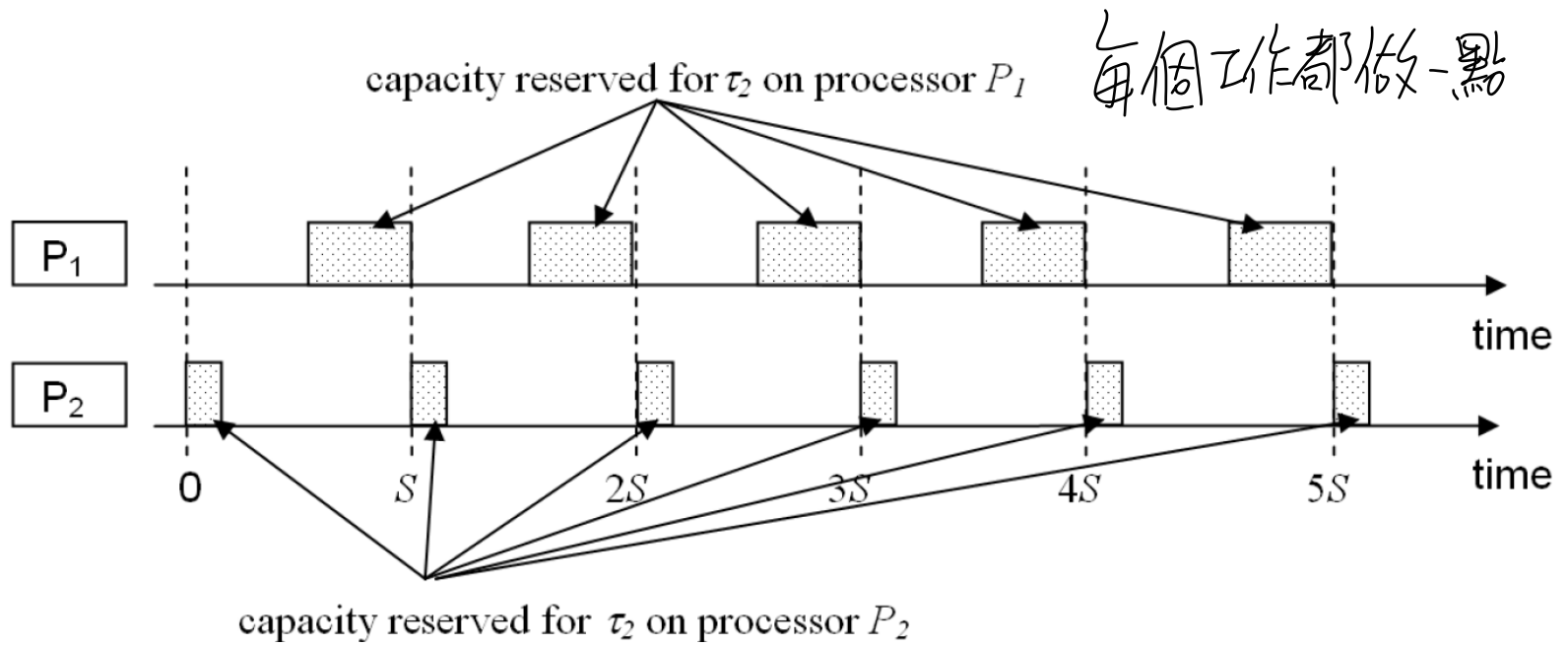
如果將 t_i 拆分並分配給處理器 m ，並且分配 t_i 後處理器 m 上的利用率最多為 U

$(\text{scheduler}, N)$ ，則到目前為止 t_i 是可調度的。

Semi-partitioned EDF

T_{\min} 是所有任務中的最短時間。

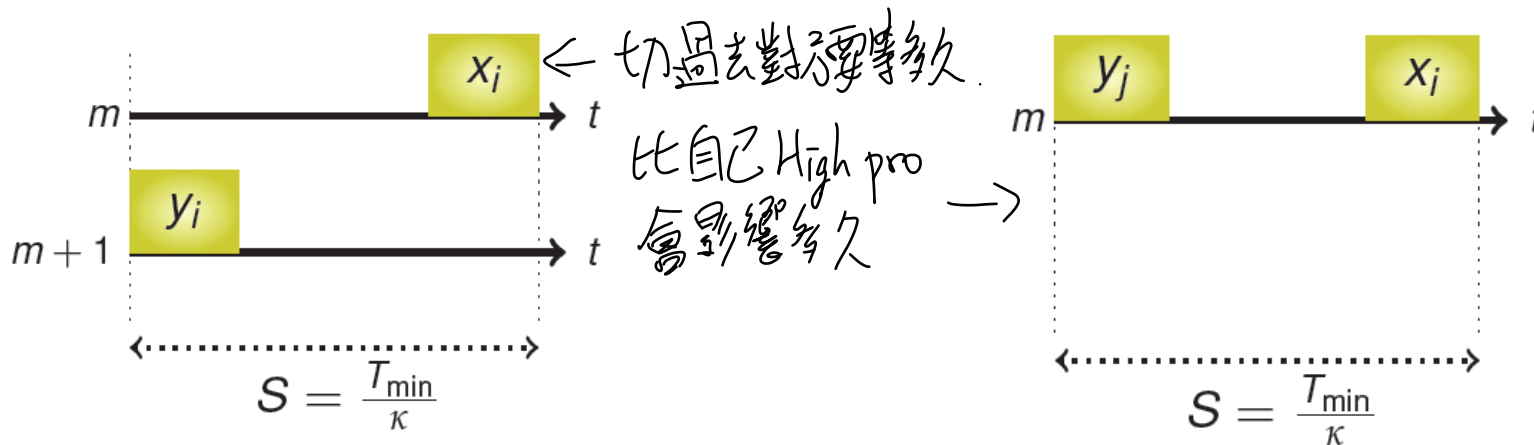
- T_{\min} is the minimum period among all the tasks.
- By a user-designed parameter k , we divide time into slots with length $S = T_{\min}/k$. 通過用戶設計的參數 k ，我們將時間劃分為長度為 $S = T_{\min} / k$ 的時隙。
- We can use the first-fit approach by splitting a task into 2 subtasks, in which one is executed on processor m and the other is executed on processor $m + 1$. 我們可以通過將一個任務分解為兩個子任務來使用“優先擬合”方法，其中一個在處理器 m 上執行，另一個在處理器 $m + 1$ 上執行。
- Execution of a split task is only possible in the reserved time window in the time slot. 僅在時隙的保留時間窗口中才能執行拆分任
- Applying first-fit algorithm, by taking SEP as the upper bound of utilization on a processor. 通過採用SEP作為處理器上的利用率上限，應用“最適合”算法。
- If a task does not fit, split this task into two subtasks and allocate a new processor, one is assigned on the processor under consideration, and the other is assigned on the newly allocated processor. 如果某個任務不適合，則將該任務分成兩個子任務並分配一個新的處理器，其中一個在正在考慮的處理器上分配，另一個在新分配的處理器上分配。



Semi-partitioned EDF

對於每個時隙，我們將保留兩個部分。

- For each time slot, we will reserve two parts.



If a task t_i is split, the task can be served only within these two pre-defined time slots with length x_i and y_i . 如果任務 t_i 被分割，則只能在這兩個長度為 x_i 和 y_i 的預定義時隙內提供服務。

處理器可以承載兩個拆分任務 t_i 和 t_j 。 t_i 在時隙的開始處提供，而 t_j 在結束時間處提供。A processor can host two split tasks, t_i and t_j . t_i is served at the beginning of the time slot, and t_j is served at the end.

The schedule is EDF, but if a split task instance is in the ready queue, it is executed in the reserved time region. 計劃為EDF，但如果拆分任務實例在就緒隊列中，則在保留的時間範圍內執行該計劃。

Semi-partitioned EDF

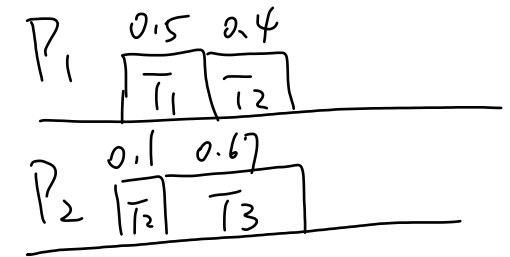
我們可以在專用處理器上使用 $U_i > \text{SEP}$ 分配所有任務 t_i 。因此，我們只考慮 U_i 不大於 SEP 的任務。

- We can assign all the tasks t_i with $U_i > \text{SEP}$ on a dedicated processor. So, we only consider tasks with U_i no larger SEP .

```

1:  $m \leftarrow 1, U_m \leftarrow 0;$ 
2: for  $i = 1$  to  $N$ , where  $N = |T|$  do
3:   if  $\frac{C_i}{T_i} + U_m \leq \text{SEP}$  then
4:     assign task  $\tau_i$  on processor  $m$ ;
5:      $U_m \leftarrow U_m + \frac{C_i}{T_i};$ 
6:   else
7:     assign task  $\tau_i$  on processor  $m$  with  $lo\_split(\tau_i)$  set to  $\text{SEP} - U_m$  and on
       processor  $m + 1$  with  $high\_split(\tau_i)$  set to  $\frac{C_i}{T_i} - (\text{SEP} - U_m);$ 
8:      $m \leftarrow m + 1$  and  $U_m \leftarrow \frac{C_i}{T_i} - (\text{SEP} - U_m);$ 

```



When executing, the reservation to serve t_i is to set x_i to $S \times (f + lo_split(t_i))$ and y_i to $S \times (f + high_split(t_i))$. SEP is set as a constant.

為什麼不能排到100%
 { 有執行順序的問題
 Highpro

3/31

執行時，為 t_i 服務的預留是將 x_i 設置為 $S \times (f + lo_split(t_i))$ ，將 y_i 設置為 $S \times (f + high_split(t_i))$ 。 SEP 設置為常數。

Two Split Tasks on a Processor

為了使拆分任務可調度，必須滿足以下充分條件

- For split tasks to be schedulable, the following sufficient conditions have to be satisfied
 - $lo_split(t_i) + f + high_split(t_i) + f \leq 1$ for any split task t_i . 切兩塊, 剩下的
 - $lo_split(t_j) + f + high_split(t_i) + f \leq 1$ when t_i and t_j are assigned on the same processor. 被幾個工作 preempt.
- Therefore, the “magic value” SEP 因此, “神奇的價值” SEP

$$SEP \leq 1 - 2f \leq 1 - 2(\sqrt[2]{\kappa(\kappa + 1)} - \kappa).$$

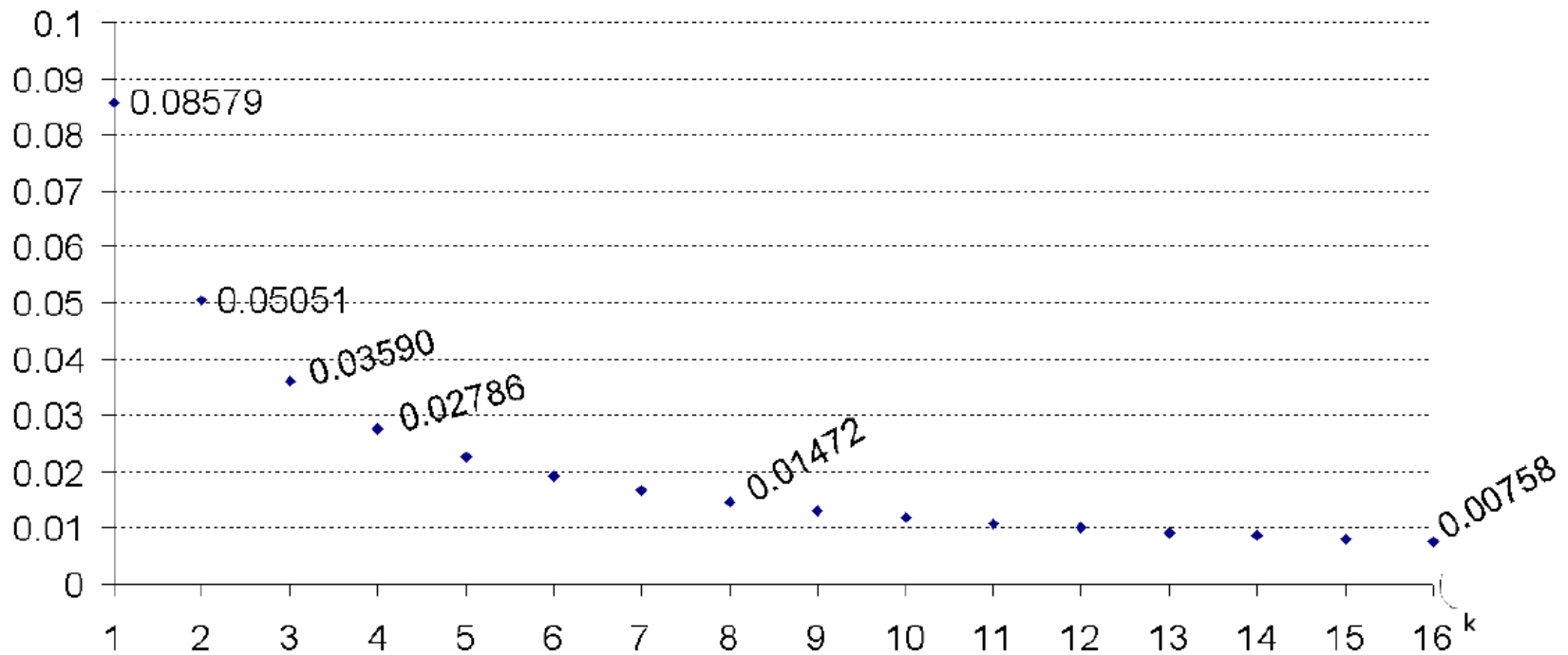
- However, we still have to guarantee the schedulability of the non-split tasks. It can be shown that the sufficient condition is 但是, 我們仍然必須保證非拆分任務的可調度性。可以證明充分條件是

$$SEP \leq 1 - 4f \leq 1 - 4(\sqrt[2]{\kappa(\kappa + 1)} - \kappa).$$

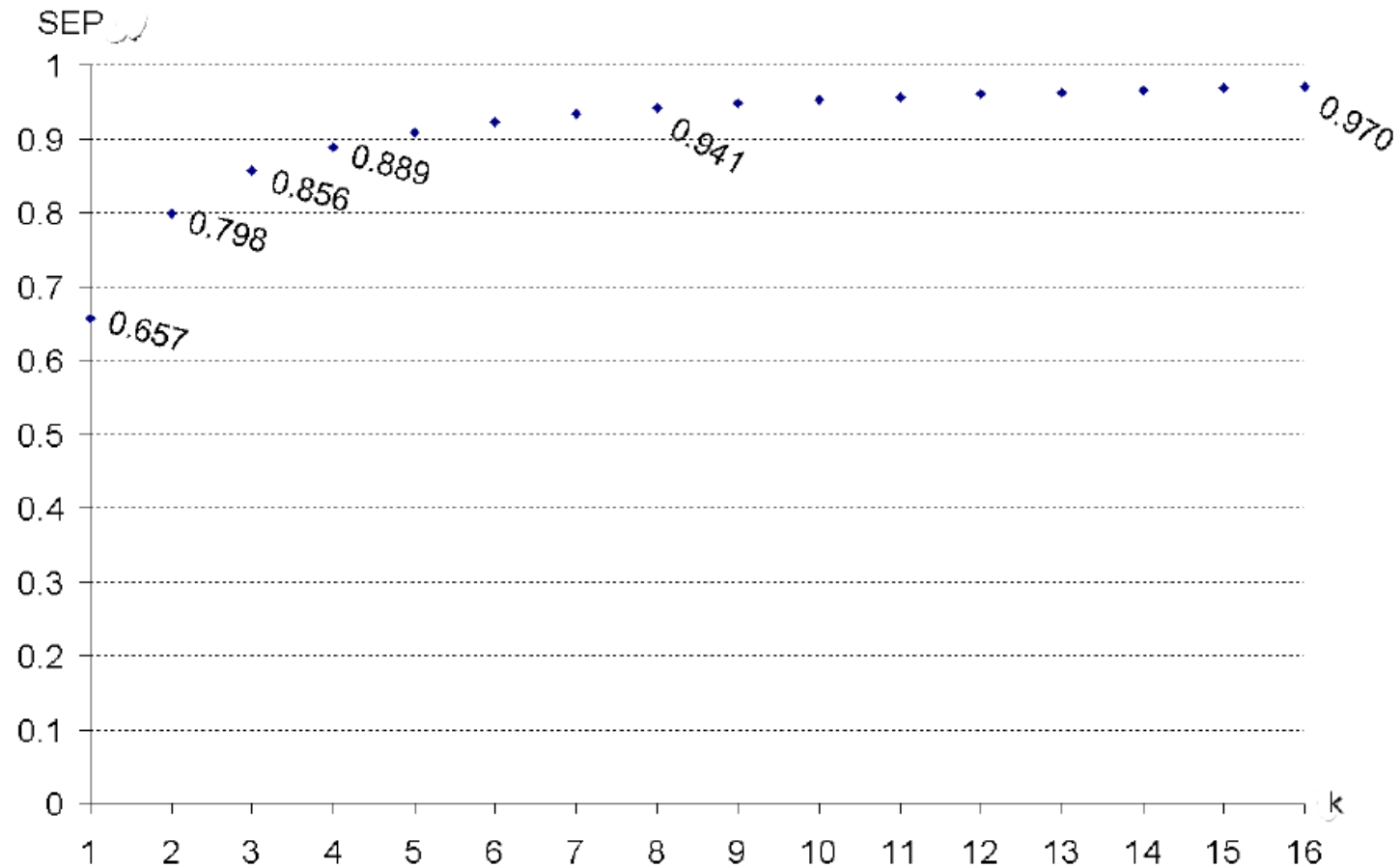
Schedulability Test

By taking SEP as $1 - 4(\sqrt[2]{\kappa(\kappa + 1)} - \kappa)$ and $f = \sqrt[2]{\kappa(\kappa + 1)} - \kappa$, the above algorithm guarantees to derive feasible schedule if $\sum_{\tau_i \in \mathbf{T}} \frac{C_i}{T_i} \leq M' \cdot SEP$ and $\frac{C_i}{T_i} \leq SEP$ for all tasks τ_i .

Magic Values: f



Magic Values: SEP



Reference

- Multiprocessor Real-Time Scheduling
 - Dr. Jian-Jia Chen: Multiprocessor Scheduling. Karlsruhe Institute of Technology (KIT): 2011-2012
- Global Scheduling
 - Sanjoy K. Baruah: Techniques for Multiprocessor Global Schedulability Analysis. RTSS 2007: 119-128
- Partitioned Scheduling
 - Sanjoy K. Baruah, Nathan Fisher: The Partitioned Multiprocessor Scheduling of Sporadic Task Systems. RTSS 2005: 321-329
- Semi-partitioned Scheduling
 - Björn Andersson, Konstantinos Bletsas: Sporadic Multiprocessor Scheduling with Few Preemptions. ECRTS 2008: 243-252

See You Next Week

4/17