Multiprocessor Real-Time Scheduling

Real-Time and Embedded Operating Systems

Prof. Li-Pin Chang
ESSLab@NYCU

Outline

- Multiprocessor Real-Time Scheduling
- Global Scheduling
- Partitioned Scheduling
- Semi-partitioned Scheduling

Multiprocessor Models

- Identical (Homogeneous) processors:
  - All processors are made of the same hardware
  - The execution time of jobs are independent of which processor they are executed on
- Uniform processors
  - All processors are made of the same hardware
  - Processes run at different speeds
  - A job runs faster on a processor of a higher clock rate
  - DVFS

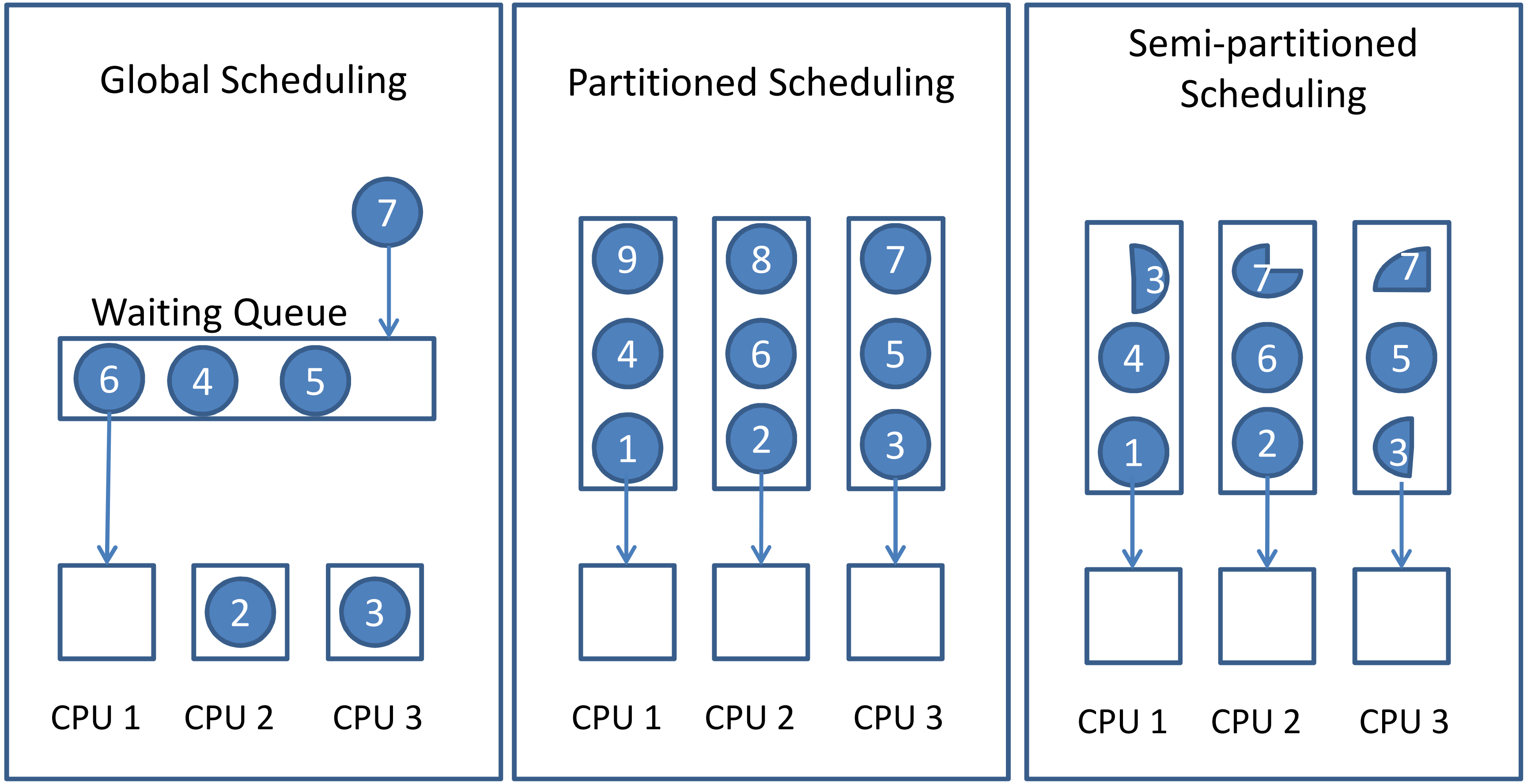
Multiprocessor Models

- Unrelated (Heterogeneous) processors
  - A job has different execution times on different processors
  - A processor may execute a job faster than other processors but execute another job slower than other processors
  - For example, multiprocessors with different instruction set architectures (ISAs)

Scheduling Models

- Global Scheduling:
  - A job can be dispatched to any processor
  - Job migrate among processors whenever necessary
  - A global ready queue
- Partitioned Scheduling:
  - Tasks are statically partitioned among processors
  - No task migration is allowed
  - Per-processor ready queues
- Semi-partitioned Scheduling:
  - Based on partitioned scheduling
  - Involves limited on-line job migration

Scheduling Models



Global Scheduling

- Here, a ready task/job means a task that can be executed
- It can be
  - A task waits in the ready queue
  - A task is being executed on a processor

Global Scheduling

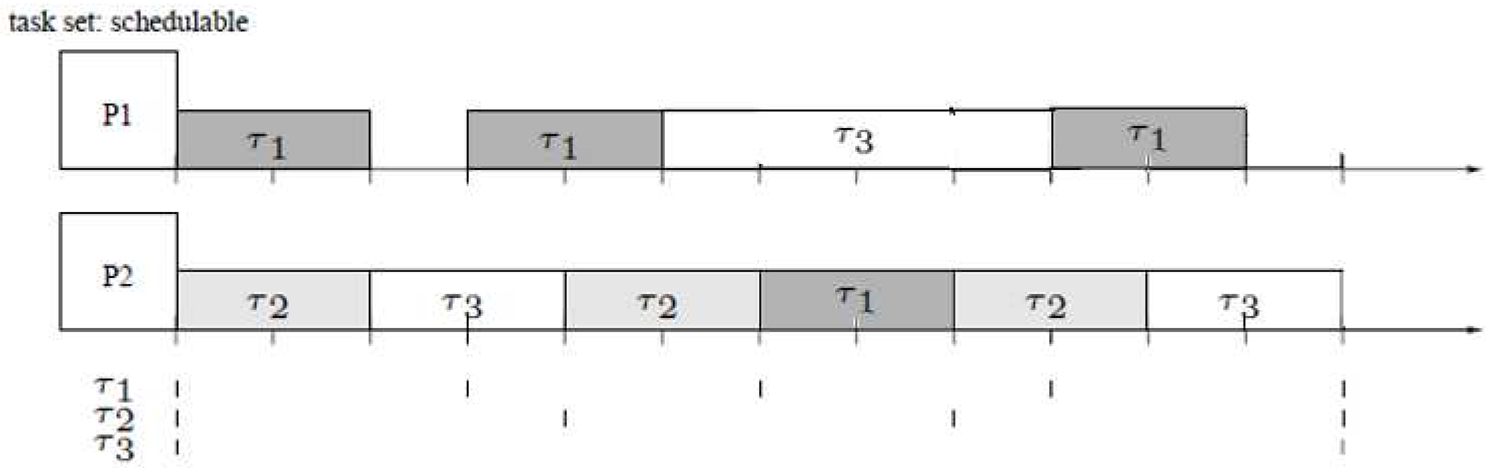
- All ready jobs are kept in a global queue, and a job can be migrated to any processor
- **Global-EDF**: When a job finishes or a new job arrives at the global queue, the M processor executes M ready jobs having the M shortest deadlines
- **Global-RM**: When a job finishes or a new job arrives at the global queue, the M processor executes M ready jobs having the M shortest periods

Global Scheduling

- The M processors always execute M ready jobs with the M earliest deadlines
- When a new job arrives
  1. If there is an idle processor, use it
  2. Otherwise, if it can preempt a running job, it preempts the running job having the farthest deadline
  - To avoid shuffling tasks on processors

Global EDF

- {t1=(2,3), t2=(2,4), t3=(8,12)}



Global Scheduling

- Advantages:
  - High processor utilization (if schedulable)
  - Unused processor time can easily be reclaimed during run-time for soft RT tasks
- Disadvantages:
  - Less intuitive! Many single-processor scheduling results cannot be extended to multiprocessor global scheduling
  - Adding processors, reducing task computation times, or "enhance" other system parameters can unexpectedly degrade task response!

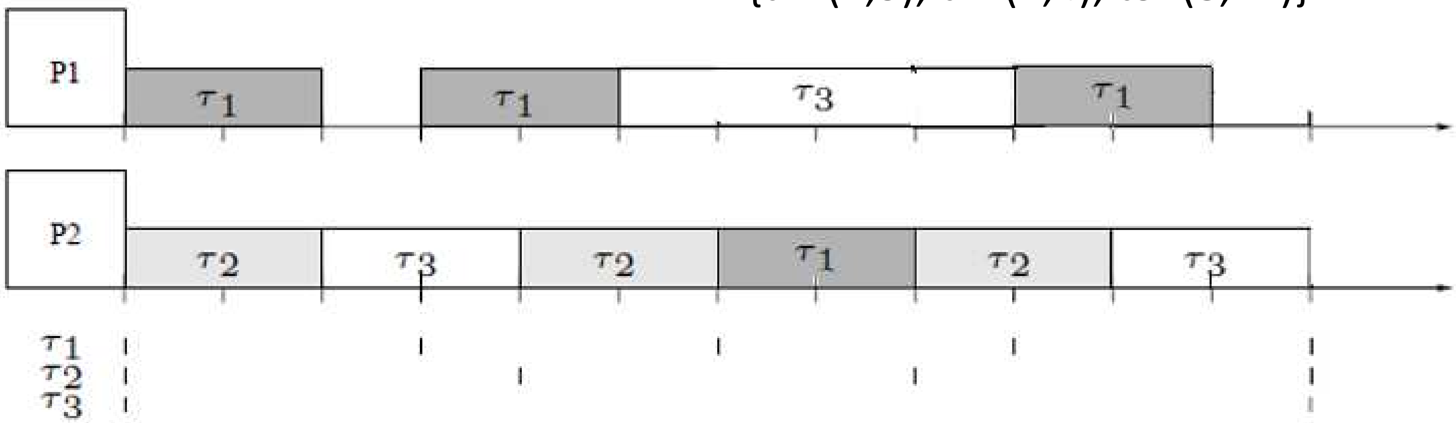
Scheduling Anomaly

- Increasing the period of a task may negatively impact on the response time of another task

Anomaly 1: Relaxing Task Period

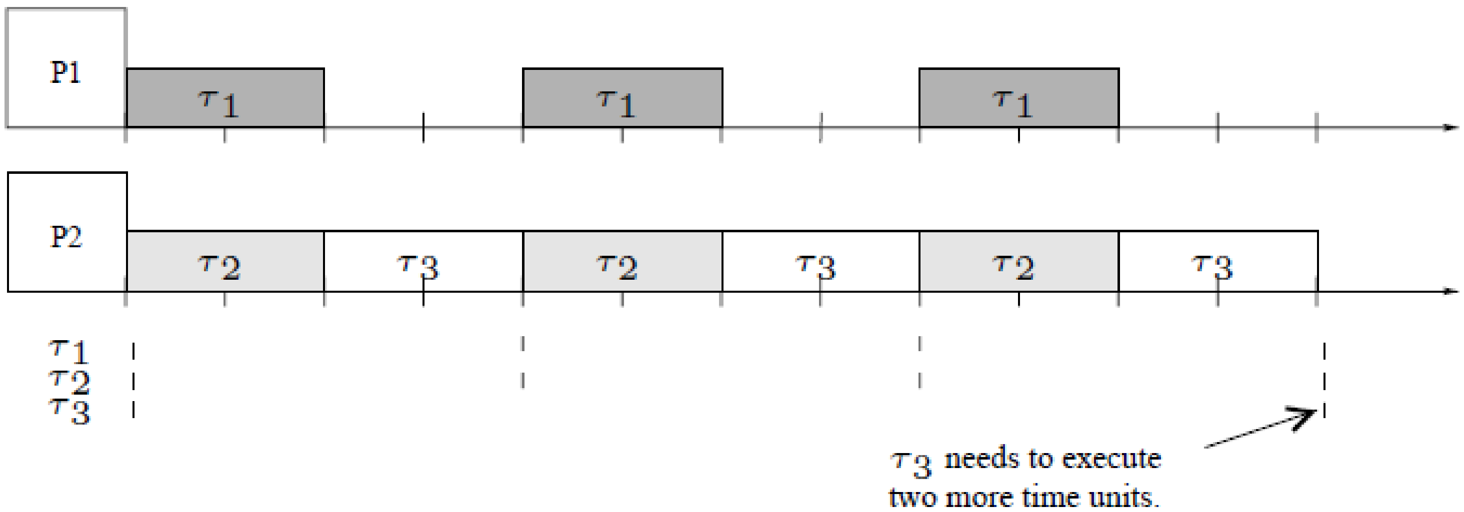
task set: schedulable

{t1=(2,3), t2=(2,4), t3=(8,12)}



task set: unschedulable

{t1=(2,4), t2=(2,4), t3=(8,12)}



Anomaly 2: Dhall’s Effect

- Dhall’s Effect of global scheduling

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

- T3 is not schedulable by global EDF/RM
  – but is schedulable if T1 and T2 share the same processor

Schedulability Test

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

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

where t_k is the task of the largest utilization C_k/T_k

Weakness of Global Scheduling

- Scheduling Anomaly
- Migration overhead
  - Cache re-population (cold start)
  - Pipeline stall

Partitioned Scheduling

- Two steps:
  - Partitioning tasks among processors
  - Scheduling tasks on each processor
- Example: Partitioned scheduling with EDF
  - Assign tasks to the processors such that no processor's capacity exceeds 100%
  - Schedule tasks on each processor using EDF

Partitioned Scheduling

- Advantages:
  - Most techniques for single-processor scheduling are applicable here
- Partitioning of tasks can be automated
  - Solving a bin-packing problem
- Disadvantages:
  - Cannot reclaim unused processor time
  - May have very low utilization, bounded by 50%
    - Worst case of bin-packing heuristic

Task Partitioning Problem

Given a set of tasks with arbitrary deadlines, the objective is to find a feasible task assignment onto M processors such that all the tasks meet their timing constraints

Bin Packing Problem

Optimization version

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

Decision version

The same as above, but asking whether all the objects can be packed into B bins of the same capacity.

The decision version of Bin Packing is known to be NP-complete, which can be reduced (transformed) to an instance of partitioned scheduling.

Bin-Packing versus Partitioned Scheduling

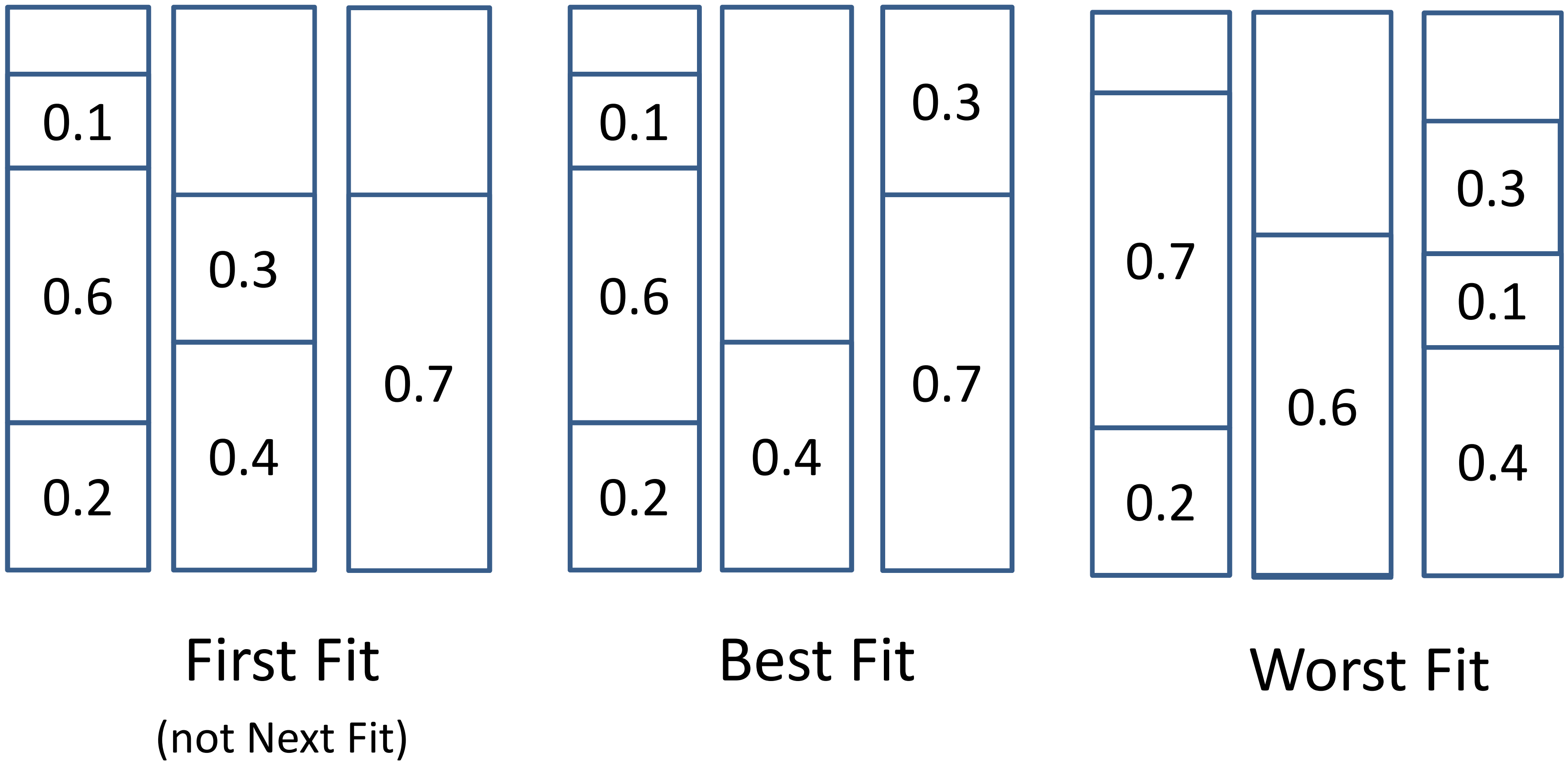
- Bin Packing: packing objects of varying sizes in boxes (”bins”) with the objective of minimizing number of used boxes.
  - Solutions (Heuristics): First Fit, etc.
- Application to multiprocessor systems:
  - Bins are represented by processors and objects by tasks
  - The decision whether a processor is ”full” or not is derived from a utilization-based schedulability test

Partitioning Algorithms

- First-Fit: choose the fitting processor of the smallest index
- Best-Fit: choose the fitting processor of the maximal utilization
- Worst-Fit: choose the fitting processor of the minimal utilization

Partitioned Example

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



Schedulability Test

Lopez [3] proves that the worst-case achievable utilization for EDF scheduling and FF allocation (EDF-FF) takes the value

If all the tasks have an utilization factor C/T under a value α, where m is the number of processors

$$\boxed{U_{wc}^{EDF-FF}(m,\beta) = \frac{\beta m + 1}{\beta + 1}} \quad \text{where} \quad \beta = \lfloor 1/\alpha \rfloor$$

Weakness of Partitioned Scheduling

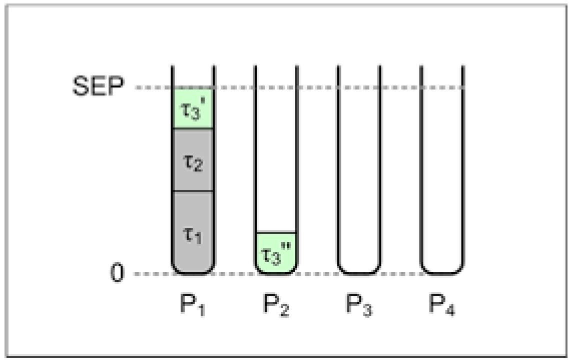
- Binding a task to a processor makes the problem NP-hard and cause pessimistic results
- Example: Suppose that there are M processors and $M + 1$ tasks with the same period T and the (worst-case) execution times of all these $M + 1$ tasks are $T/2 + e$ with $e > 0$
  - With partitioned scheduling, it is not schedulable
  - Is it possible to divide a task between two processors?

Semi-partitioned Scheduling

- Based on First Fit
- Adding tasks to a processor until the processor is fully loaded
- Partitioning the next task into p1 and p2 and completely fill the current processor with p1
- Adding p2 to the next processor

Semi-partitioned EDF

- Assignment phase
  - Applying first-fit algorithm, by taking SEP as the upper bound of utilization on a processor.
  - If a task does not fit, split this task into two subtasks, one is assigned on the current processor and the other is assigned to the next processor



Semi-partitioned EDF

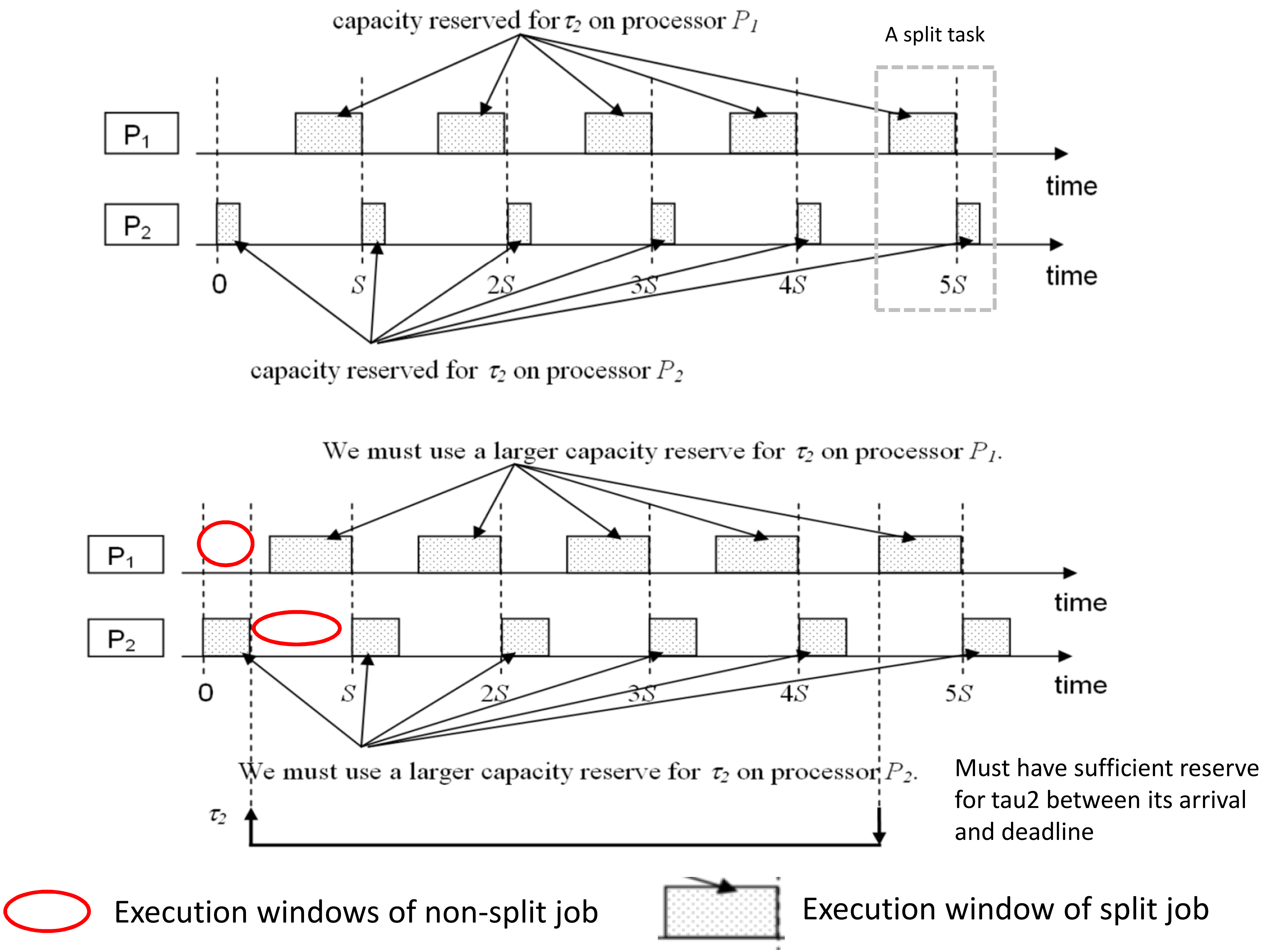
- We can assign all the tasks t_i with U_i > SEP on a dedicated processor. So, we only consider tasks with U_i no larger SEP.

$m \leftarrow 1$, $U_m \leftarrow 0$;
for $i = 1$ to N, where $N = |\mathbf{T}|$ **do**
    if $\frac{C_i}{T_i} + U_m \leq SEP$ **then**
        assign task τ_i on processor m;
        $U_m \leftarrow U_m + \frac{C_i}{T_i}$;
    else
        assign task τ_i on processor m with $lo\_split(\tau_i)$ set to $SEP - U_m$ and on processor $m + 1$ with $high\_split(\tau_i)$ set to $\frac{C_i}{T_i} - (SEP - U_m)$;
        $m \leftarrow m + 1$ and $U_m \leftarrow \frac{C_i}{T_i} - (SEP - U_m)$;

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

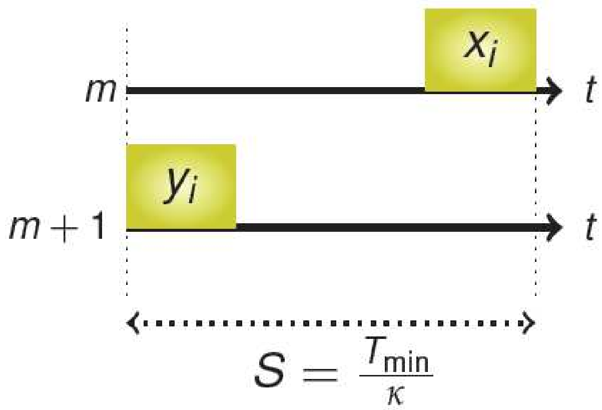
Semi-partitioned EDF

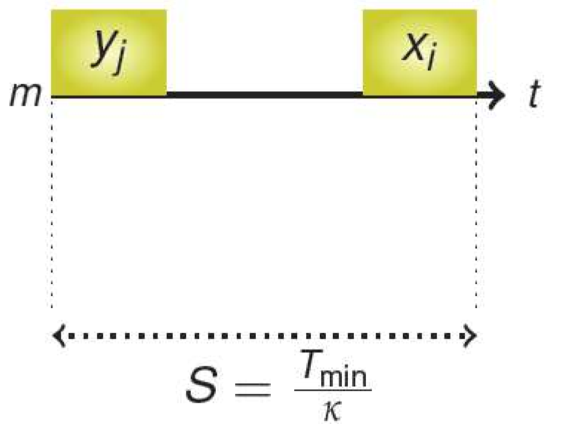
- Execution phase
  - T_{min} is the minimum period among all the tasks
  - By a user-designed parameter *k*, we divide time into slots with length $S = T_{min}/k$
  - Execution of a split task is only possible in the reserved time window in the time slot
  - The rest of the time: scheduling tasks on each individual processor using EDF


capacity reserved for τ_2 on processor P_1
A split task
P_1
P_2
time
time
0
S
$2S$
$3S$
$4S$
$5S$
capacity reserved for τ_2 on processor P_2
We must use a larger capacity reserve for τ_2 on processor P_1.
P_1
P_2
time
time
0
S
$2S$
$3S$
$4S$
$5S$
We must use a larger capacity reserve for τ_2 on processor P_2.
τ_2
Must have sufficient reserve for tau2 between its arrival and deadline
Execution windows of non-split job
Execution window of split job

Semi-partitioned EDF

- 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 .

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.

Two Split Tasks on a Processor

- For split tasks to be schedulable, the following sufficient conditions have to be satisfied
  - lo_split(t_i) + f + high_split(t_i) + f <= 1 for any split task t_i .
  - lo_split(t_j) + f + high_split(t_i) + f <= 1 when t_i and t_j are assigned on the same processor.
- Therefore, the "magic value" 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

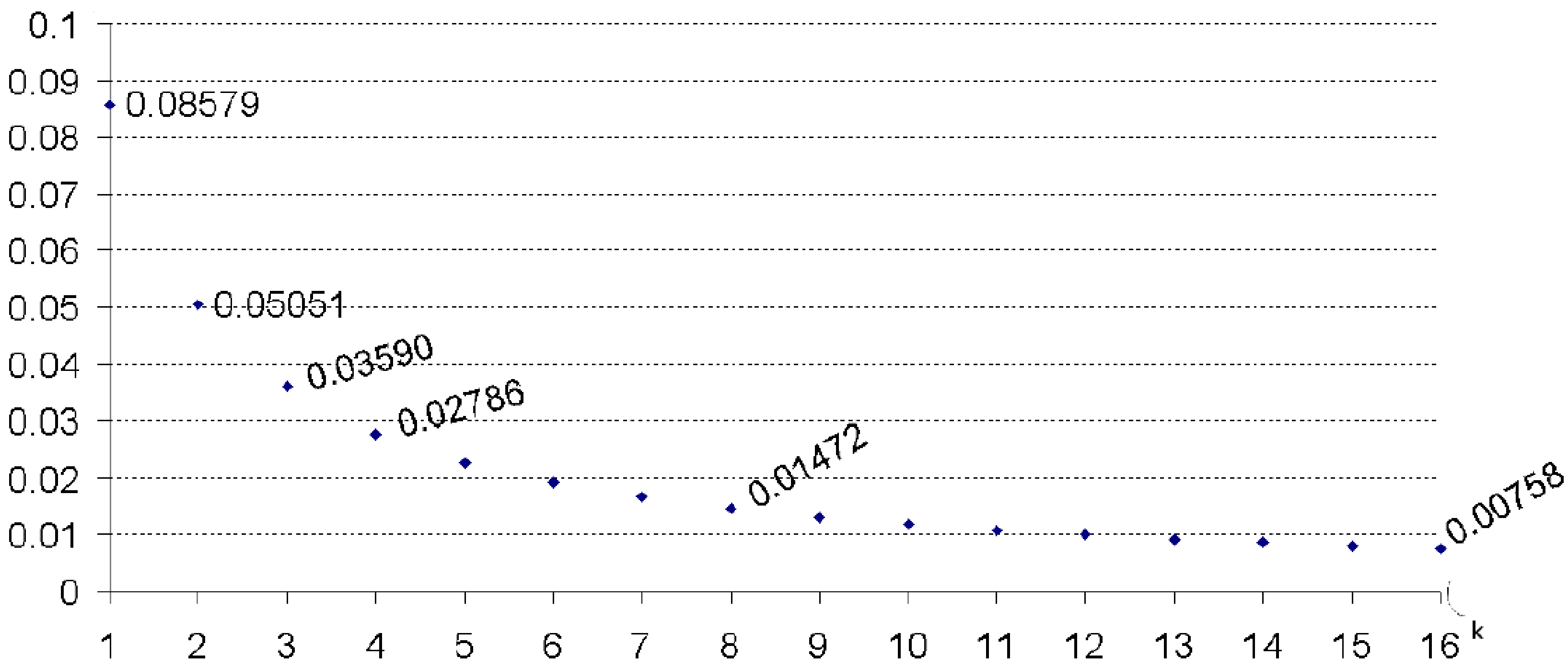
$$SEP \leq 1 - 4f \leq 1 - 4(\sqrt[2]{\kappa(\kappa+1)} - \kappa).$$

Schedulability Test

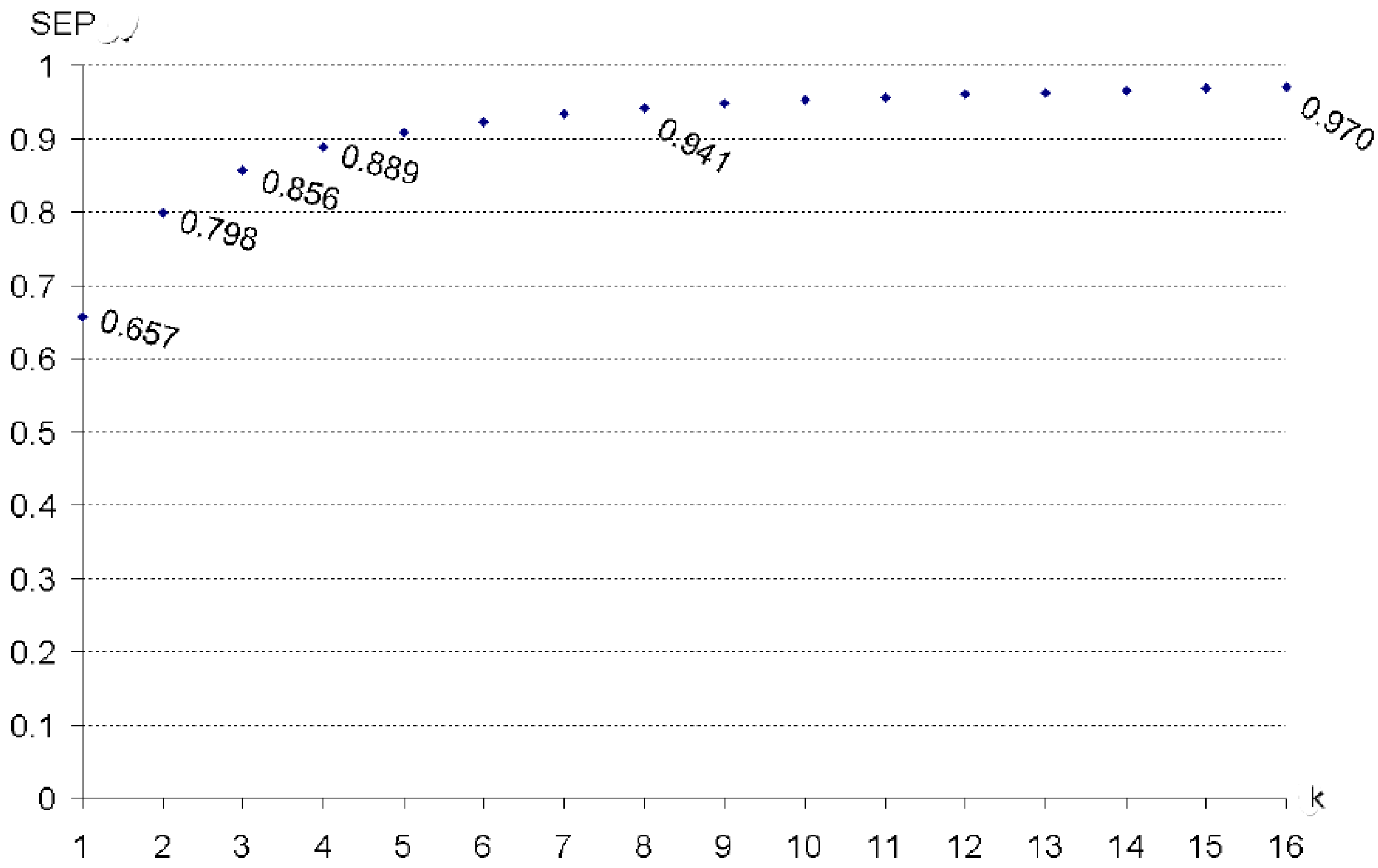
By taking SEP as $1 - 4(\sqrt[2]{\kappa(\kappa+1)} - \kappa)$ and $f = \sqrt[2]{\kappa(\kappa+1)} - \kappa$, the above algorithm guarantees to derive feasible schedule if $\sum_{\tau_i \in \mathbf{T}} \frac{C_i}{T_i} \leq M' \cdot SEP$ and $\frac{C_i}{T_i} \leq SEP$ for all tasks τ_i.

M' = the # of processors serving tasks whose individual utilization <= SEP

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

Credit

- This slice set is based on materials provided by Prof. Ya-Shu Chen (NTUST)