Scheduling Algorithms for Multiprocessor

Paulo Baltarejo Sousa

24/09/2010

Agenda

- Part I: Scheduling Algorithms for Multiprocessor in a Hard Real-Time Environment
- Part II: Scheduling Algorithms for Multiprocessor Systems
  - Global
  - Partitioned
  - Semi-partitioned

Part I: Scheduling Algorithms for Multiprocessor in a Hard Real-Time Environment

C. L. Liu

1969

Notation and Assumptions

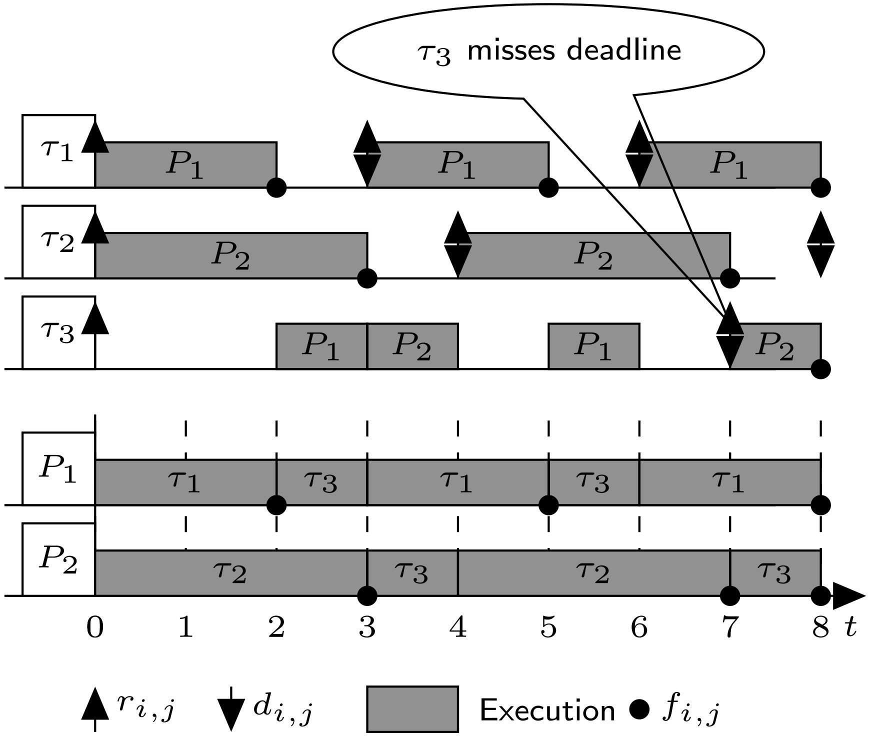
- A task set (τ) is composed by n tasks ($\tau = \{\tau_1 \cdots \tau_n\}$):
- Each task is independent and is characterized by three-tuple (C_i,T_i,D_i), where:
  - C - Execution time
  - T - Period
  - D - Deadline
- The system is composed by m processors and preemption is allowed.
- A task is said to be *background task* if it is allowed to execute on any processor.
- A *non-background task* executes on a dedicated processor.
- Background computation time on m processors is the non-overlapping processor time on these m processors available to execute background tasks.

Period-driven

- Period-driven scheduling algorithm (shorter period, the higher priority) is not optimun for multiprocessor systems.

	C	T	$U = C/T$
τ_1	2.0	3.0	0.62
τ_2	3.0	4.0	0.75
τ_3	4.0	7.0	0.57

$$U_s = \frac{1}{m} \sum_{i}^{n} U_i$$
$$= 0.97$$



- task τ_1 executes on processor P_1 (non-background task).
- task τ_2 executes on processor P_2 (non-background task).
- task τ_3 executes on processor P_1 and P_2 (background task).

Theorem 1: definition

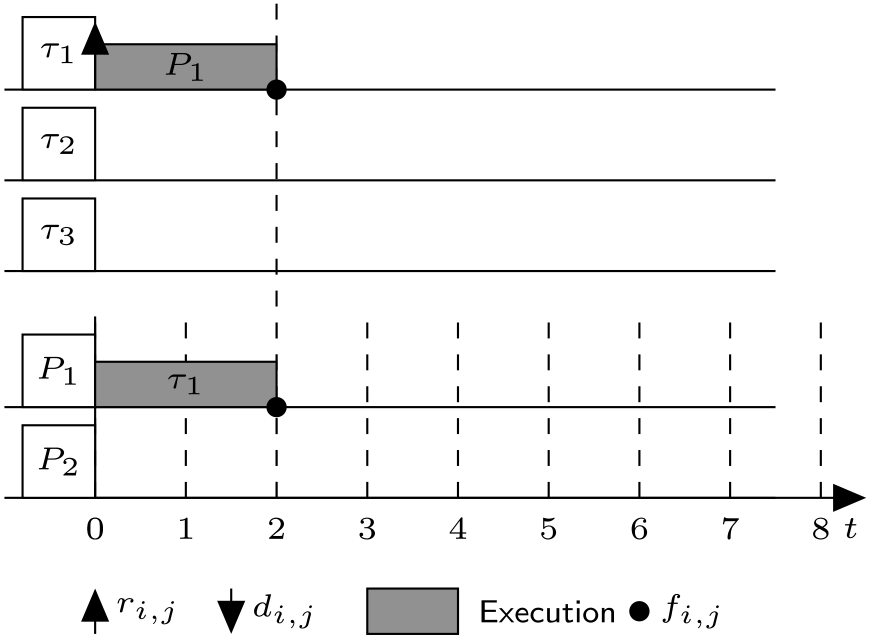
- *Theorem 1: A lower bound δ_{m+1} to the value of C_{m+1} such that the period-driven scheduling algorithm is feasible for $C_{m+1} \leq \delta_{m+1}$*
- Given the values T_1, T_2, ..., T_m and T_{m+1} and C_1, C_2, ..., C_m, theorem 1 gives a lower bound to the value of C_{m+1}, such that the period-driven scheduling algorithm is feasible.
- Consider the following task set (composed by three tasks) to be scheduled on a system composed by $m = 2$ processors.
- Which is the value of C_3?

	C	T
τ_1	2.0	3.0
τ_2	3.0	4.0
τ_3	?	7.0

Theorem 1: concepts : $g_j(t)$

- $g_j(t)$ gives a lower bound to the background computation time on processors P_1, P_2, $\cdots$, P_j, within any contiguous t time units.

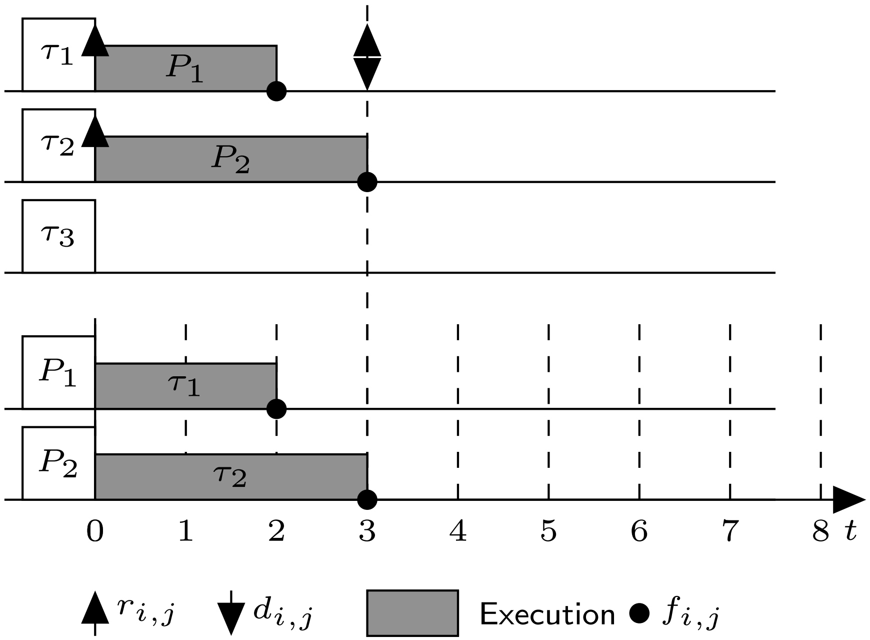
$$g_1(C_1) = g_1(2) \\ = 0;$$



Theorem 1: concepts : $g_j(t)$

- $g_j(t)$ gives a lower bound to the background computation time on processors P_1, P_2,$\cdots$, P_j, within any contiguous t time units.

$$g_1(C_1) = g_1(2) = 0$$
$$g_1(C_2) = g_1(3) = 1$$
$$g_2(C_2) = g_2(3) = 1$$



Theorem 1: concepts : δ_i

- $\delta_i, i = 1, 2, \cdots, m$ is a lower bound to the background computation time on processors P_1, P_2,$\cdots$, P_i, within each cycle of task τ_i.

$$\delta_1 = T_1 - C_1$$

$$\delta_i = T_i - C_i + \max(g_1(C_i), g_2(C_i), \cdots, g_{i-1}(C_i)),$$

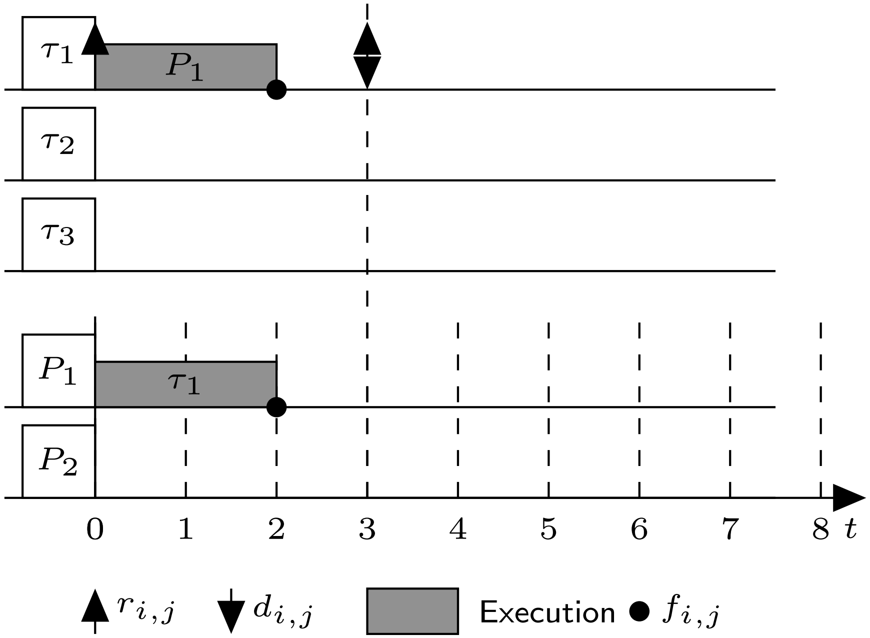
$$i = 2, \cdots, m$$

$$\delta_{m+1} = \max(g_1(T_{m+1}), g_2(T_{m+1}), \cdots, g_m(T_{m+1}))$$

Theorem 1: concepts : δ_i

- $\delta_i, i = 1, 2, \cdots, m$ is a lower bound to the background computation time on processors P_1, P_2,$\cdots$, P_i within each cycle of task τ_i.

$$\begin{aligned}\delta_1 &= T_1 - C_1 \\ &= 3 - 2 \\ &= 1\end{aligned}$$

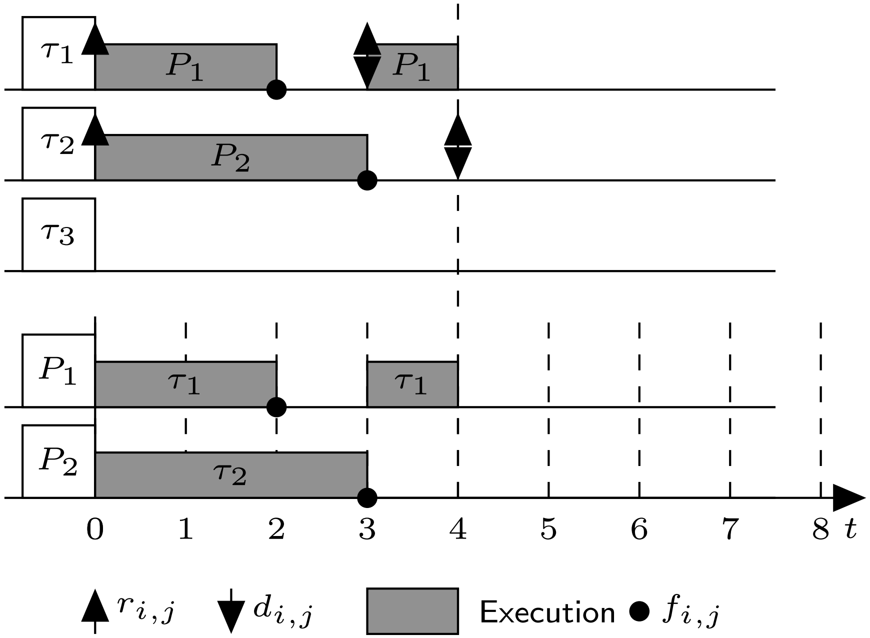


Theorem 1: concepts : δ_i

- $\delta_i, i = 1, 2, \cdots, m$ is a lower bound to the background computation time on processors P_1, P_2,$\cdots$, P_i within each cycle of task τ_i.

$$\delta_1 = T_1 - C_1 = 1$$

$$\begin{aligned}\delta_2 &= T_2 - C_2 + g_1(C_2)\\ &= 4 - 3 + 1\\ &= 2\end{aligned}$$



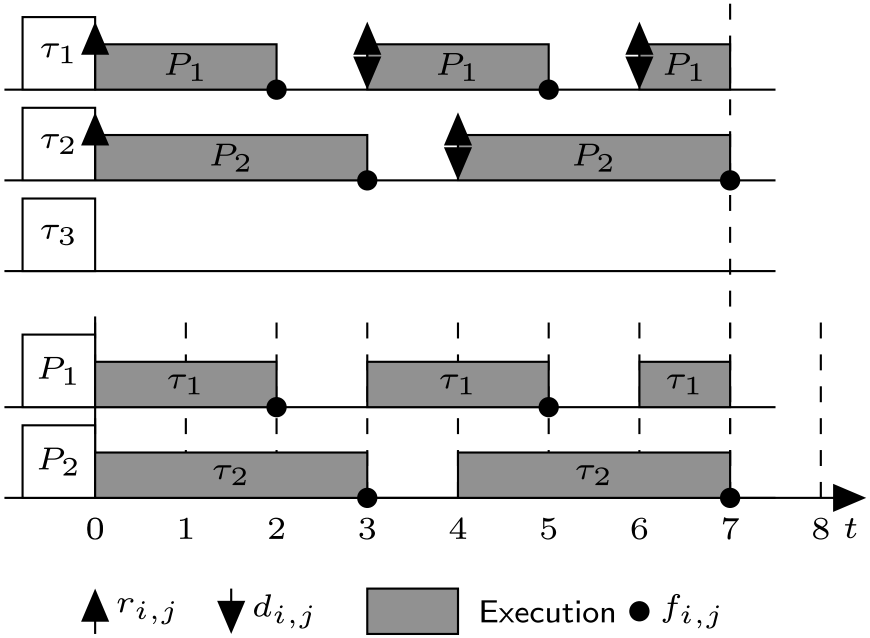
Theorem 1: concepts : δ_i

- $\delta_i, i = 1, 2, \cdots, m$ is a lower bound to the background computation time on processors P_1, P_2,$\cdots$, P_i within each cycle of task τ_i.

$$\delta_1 = 1$$

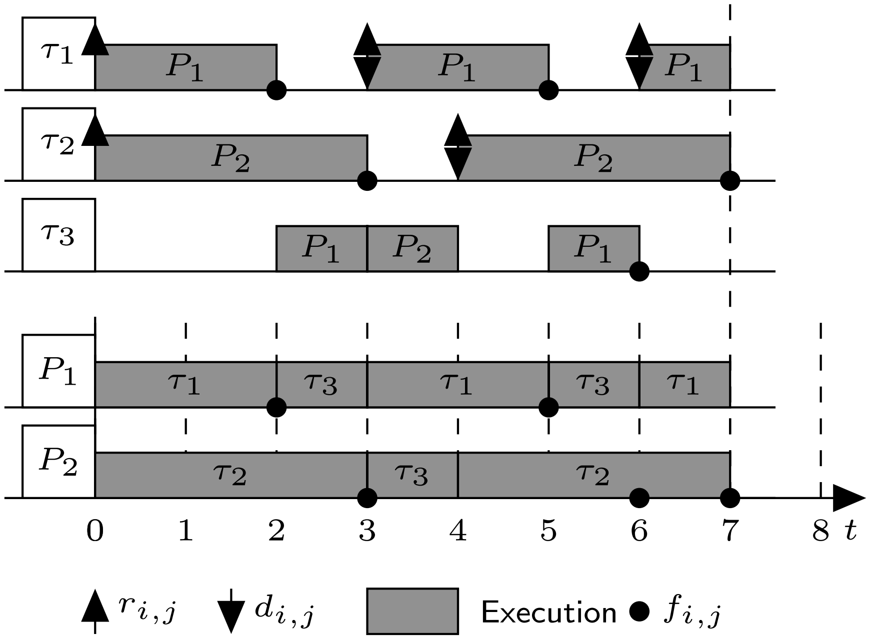
$$\delta_2 = 2$$

$$\delta_3 = max(g_1(T_3), g_2(T_3)) = max(2, 3) = 3$$



Theorem 1: δ_{m+1}

- With $C_3 = 3$, task set (τ) is schedulable.



Theorem 1: general case

$$\delta_{m+2} = \left(\left\lfloor \frac{T_{m+2}}{T_{m+1}} \right\rfloor - 1 \right) (\delta_{m+1} - C_{m+1})$$

$$\delta_{m+3} = \left(\left\lfloor \frac{T_{m+3}}{T_{m+2}} \right\rfloor - 1 \right) (\delta_{m+2} - C_{m+2})$$

$$\cdots$$

Conclusions

- It is 4 pages paper (more precisely 3.5 pages, with two big tables)
- Focus is on period-driven and also on deadline-driven scheduling algorithms for multiprocessor systems.
- The content is not very clear and mathematical formulation is the same for both types of scheduling algorithms, based on time t, T_i and C_i, but the results are different (using the same task set).
- Main contribution: *Few of the results obtained for a single processor generalize directly to the multiple processor case... bringing in additional processors adds a new dimension to the scheduling problem*.

Part II:
Scheduling Algorithm for Multiprocessor Systems

Scheduling Algorithm for Multiprocessors

- Multiprocessor scheduling algorithms are categorized as:
  - **Global** scheduling algorithms store tasks in one global queue, shared by all processors. At any moment, the m highest-priority tasks among those are selected for execution on the m processors.
  - **Partitioned** scheduling algorithms part the task set such that all tasks in a partition are assigned to the same processor.
  - **Semi-partitioned** or task-splitting scheduling algorithms; some tasks are assigned to specific processors, as partitioned, and the other tasks may migrate between processors, like global.

Task Set

- Consider a preemptive system composed by three ($m = 3$) identical processors (P_1, P_2 and P_3) and a synchronous periodic task set composed by four ($n = 4$) independent tasks (τ_1,...,τ_4) with implicit deadlines ($D_i = T_i$).

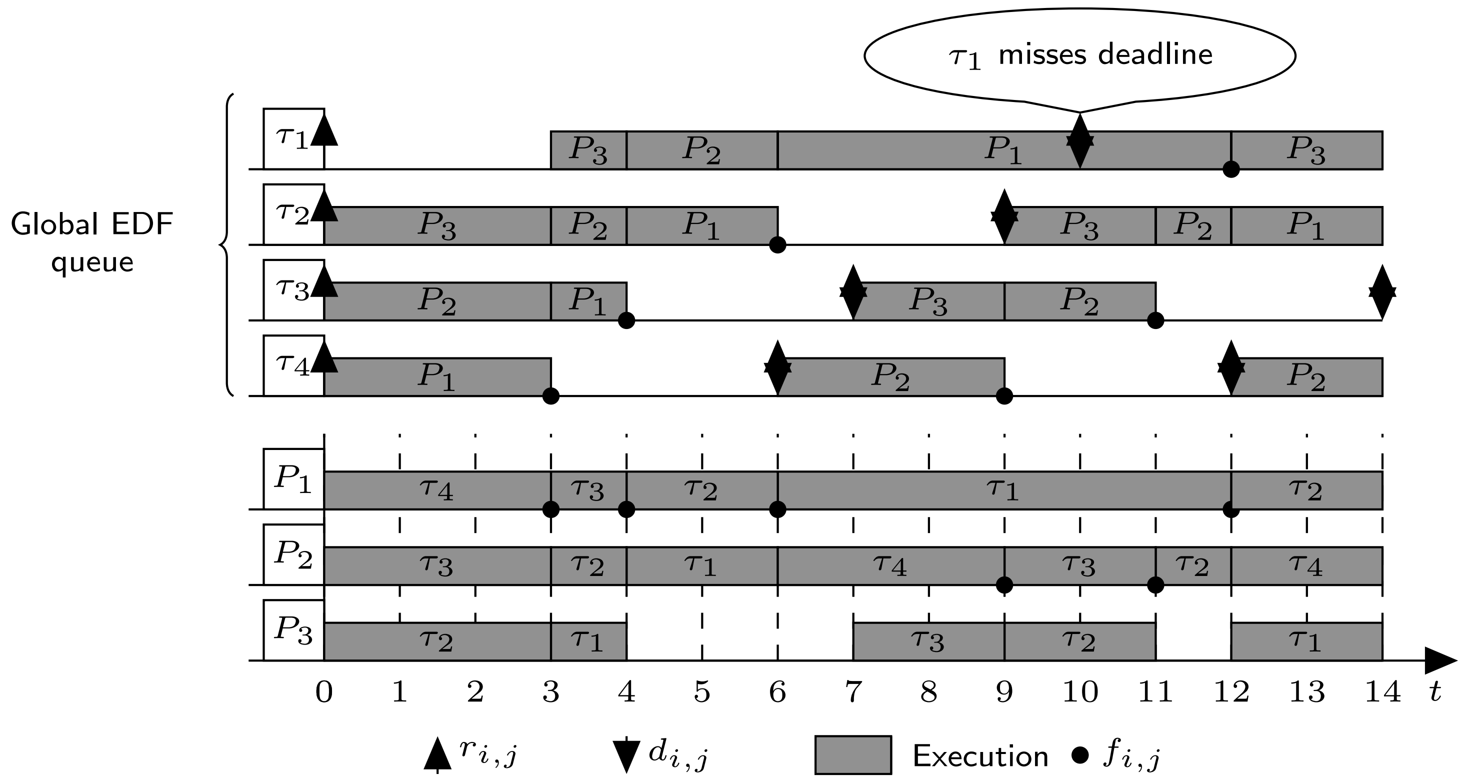
Task	C	T	U
τ_1	9	10	0.900
τ_2	6	9	0.667
τ_3	4	7	0.571
τ_4	3	6	0.500

$$U_s = \frac{1}{m} \sum_{i=1}^{4} U_i = 0.879.$$

Global

Global EDF

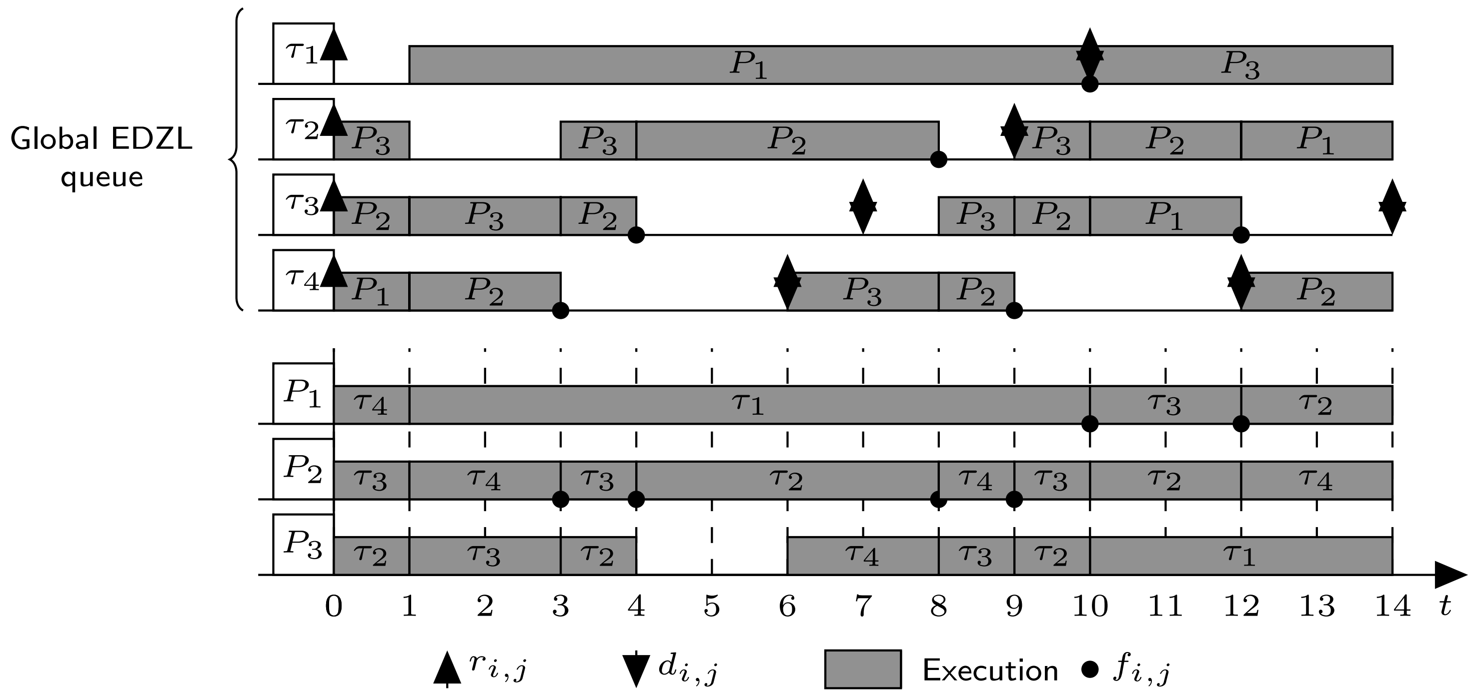
- Under global EDF scheduling policy, all tasks are stored into a global queue sorted by the absolute deadline and at each time t the m highest priority tasks ready to be executed, executes on m processors.



Earliest Deadline First until Zero Laxity (EDZL)

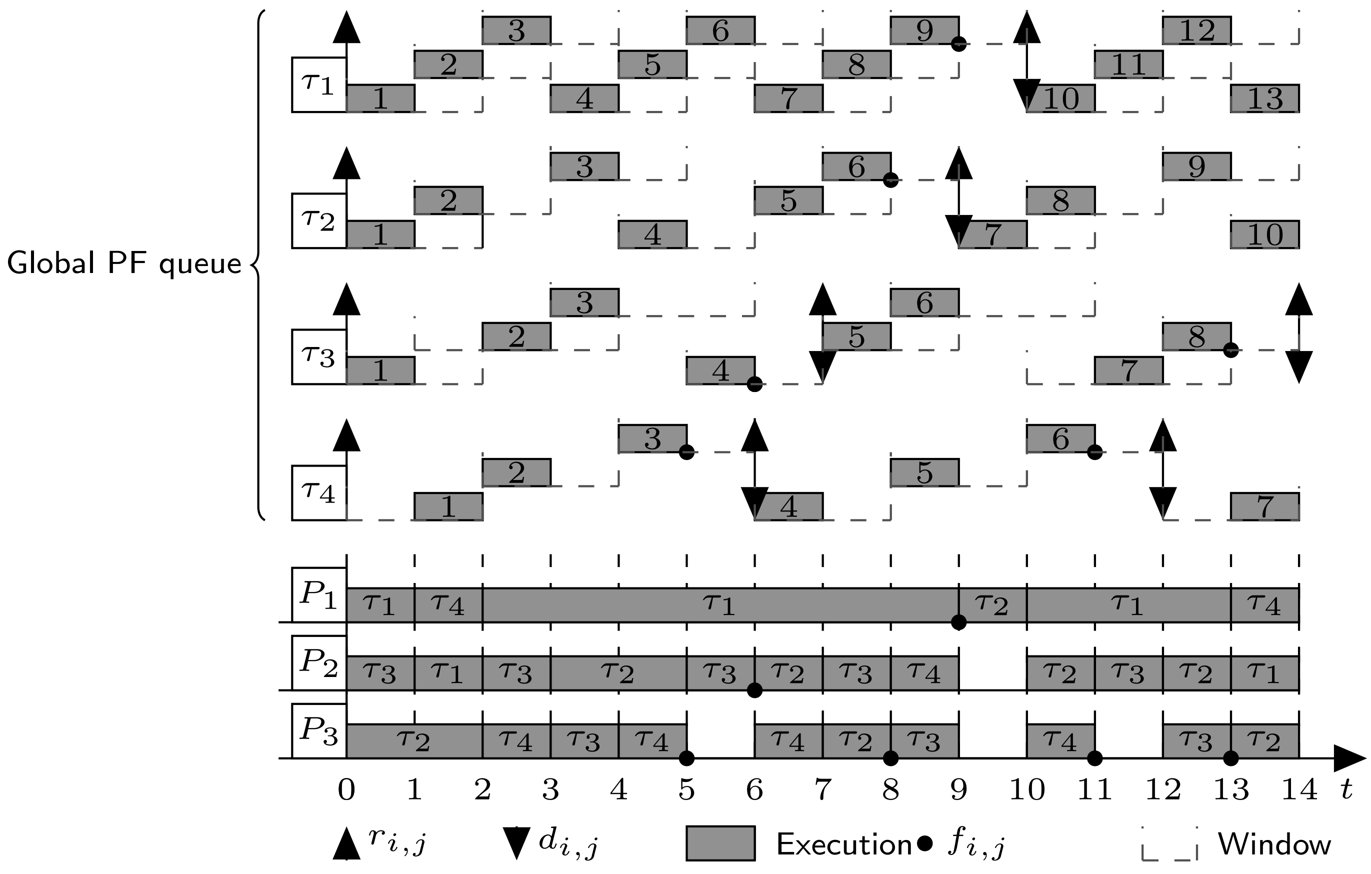
- EDZL for multiprocessor systems is a global scheduling algorithm that combines the features of two uniprocessor scheduling algorithms: EDF and LLF. LLF scheduling algorithm is a scheduling algorithm that assigns higher priority to a task with the least laxity.

- The laxity of a task at time t is defined as the difference between the deadline and the amount of execution time remaining to be complete.



Pfair scheduling algorithms

- The main idea of the pfair scheduling algorithms is to provide a proportionate progress according to the task utilization. For that, pfair breaks each task in an infinite sequence of quantum-length `subtasks` and each subtask has a `pseudo-release` and a `pseudo-deadline`.



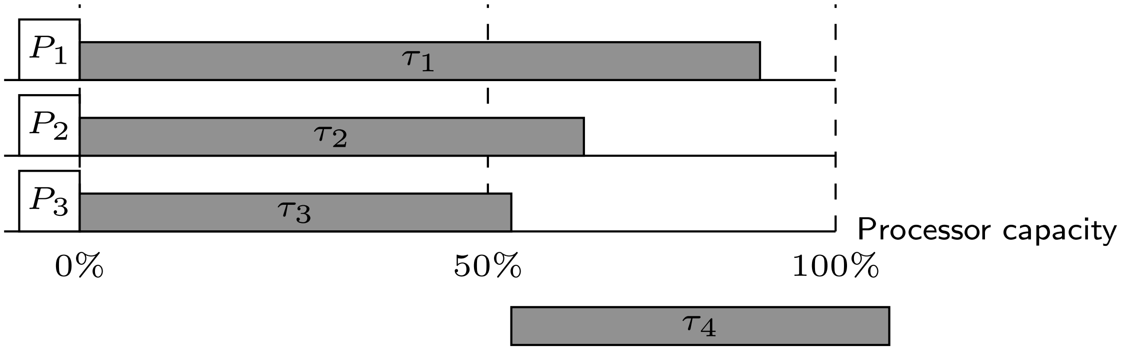
Partitioned

Bin-packing

- The partitioned scheduling algorithms are composed by two algorithms: offline task assigning algorithm and the online dispatching algorithm.
- Assigning tasks to processors is a bin-packing problem, which is known to be a NP-hard problem.
- The main goal of bin-packing is to pack a collection of items with different sizes into the minimum number of fixed-size bins such that the total weight, volume, etc. does not exceed some maximum value.
- In the context of real-time scheduling algorithm, each item is a task τ_i that composed the task set (τ), the size of each item is the utilization of task (U_i), each bin is a processor (P_i) and the size of each bin is the capacity of processor.
- There are several heuristics for these kind of problems, examples of those heuristics are Next-fit (NF), First-Fit (FF) and Best-Fit (BF).

Partitioned

- The partitioned scheduling algorithms assign statically tasks to the processor and those are scheduled on each processor using an uniprocessor scheduling algorithm, like, for instance, RM or EDF.
- Assuming that the assignment algorithm work as the FF bin-packing that assigns tasks one by one to the lowest-indexed processor where each fits, then, tasks τ_1 (with $U_1 = 0.900$), τ_2 (with $U_2 = 0.667$) and τ_3 (with $U_3 = 0.571$) are assigned to processors P_1, P_2 and P_3, respectively. Consequently, task τ_4 (with $U_4 = 0.500$) cannot be assigned to any processor, because none of them have capacity enough to encompass this task.

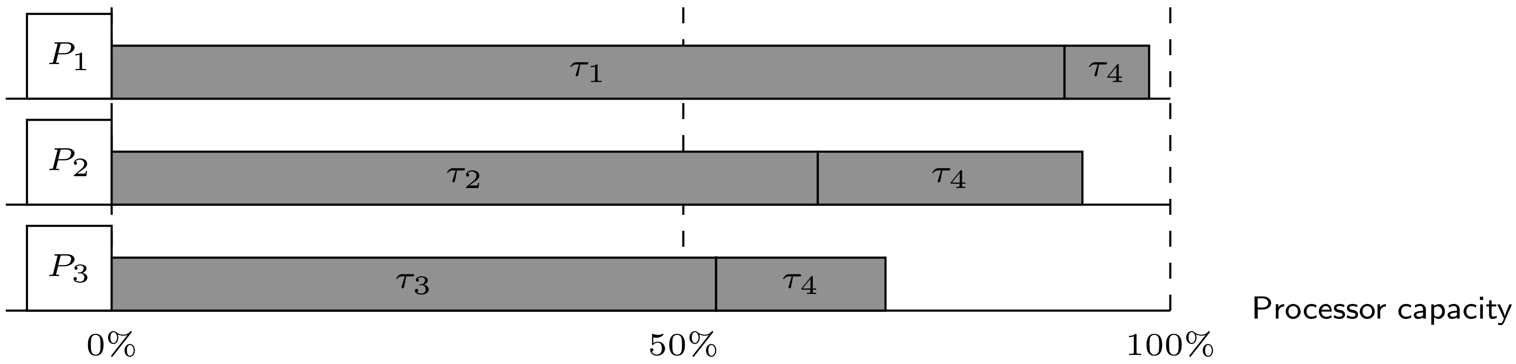


Semi-partitioned

EDF-Window-constraint Migration (EDF-WM)(I)

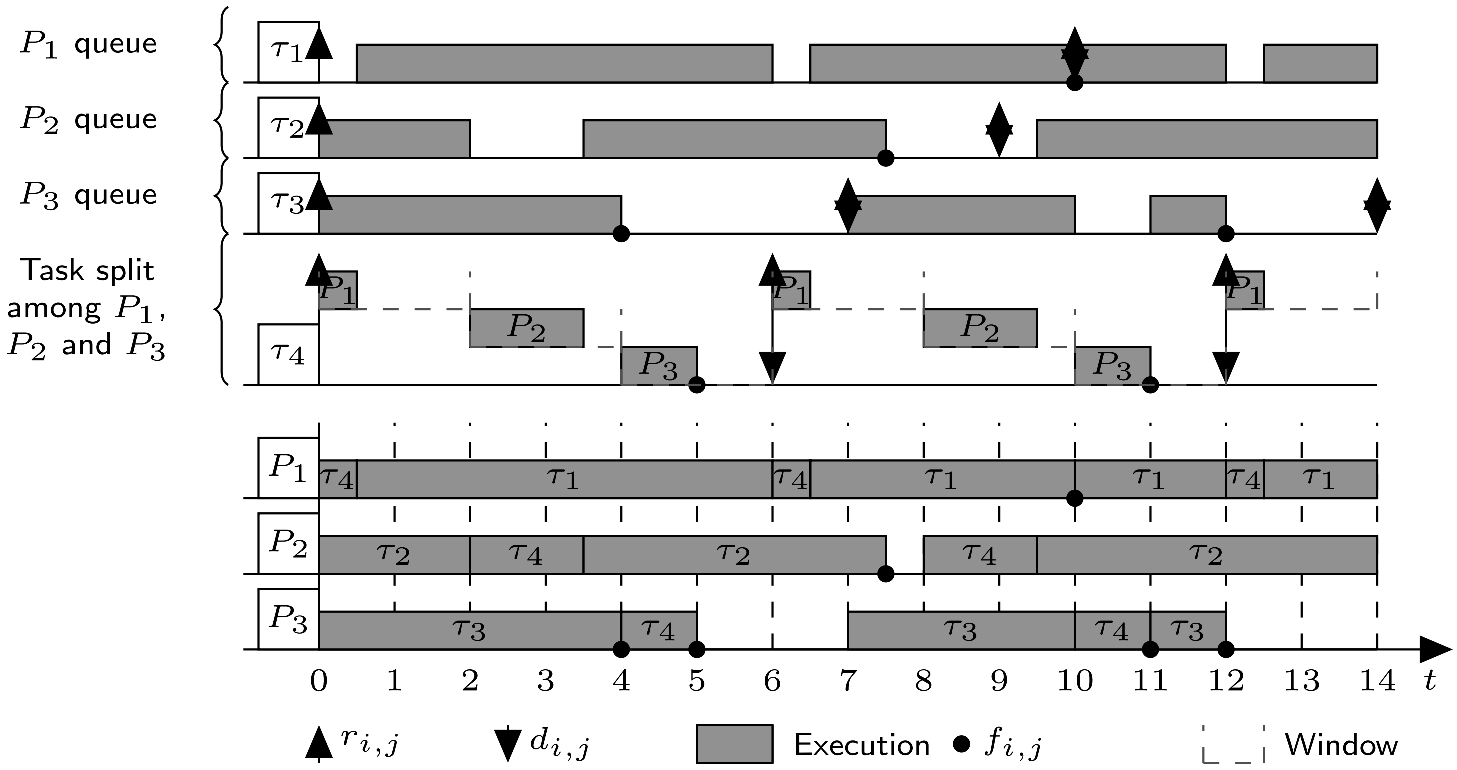
- Each task is assigned to an individual processor using FF bin-packing heuristic. A task is split, only when no individual processor has remaining capacity enough to encompass that task.

- The execution of task τ_4 on processors P_1, P_2 and P_3 cannot violate the timimg requeriments of the already assigned tasks.



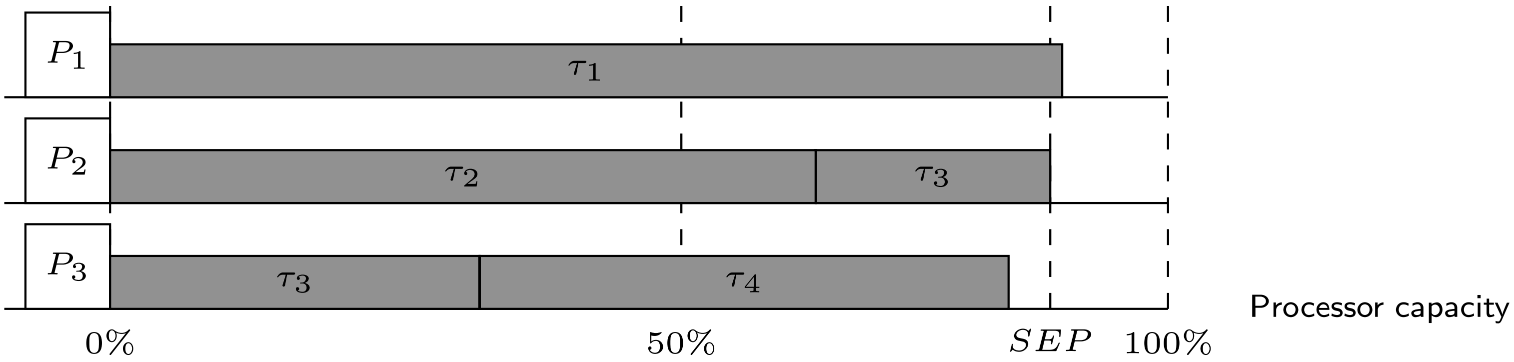
EDF-Window-constraint Migration (EDF-WM)(II)

- The online dispatching algorithm schedules tasks on each processor under EDF scheduling algorithm.



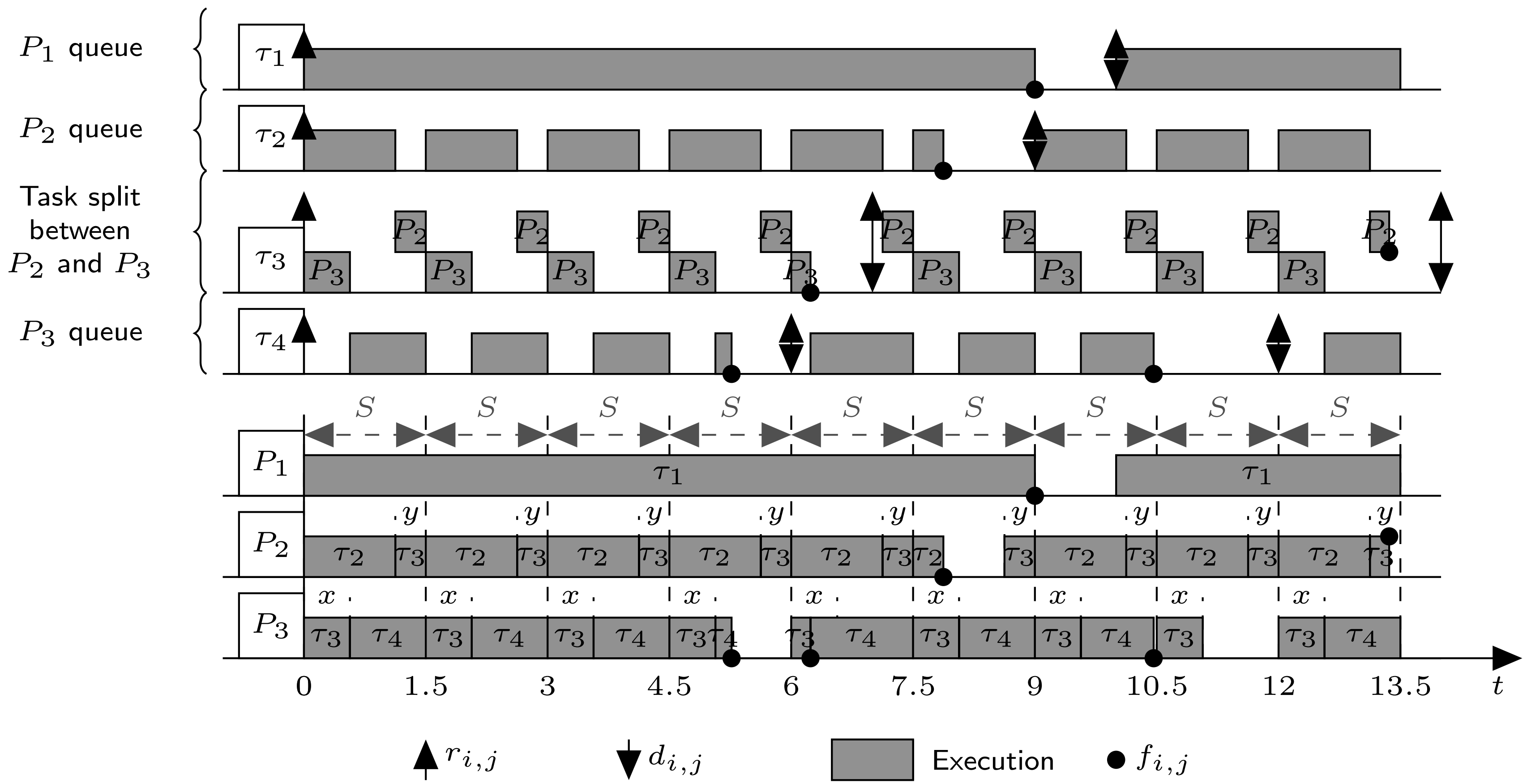
Sporadic Multiprocessor Scheduling (SMS) (I)

- The SMS algorithm divides time into slots.
- A task whose utilization exceed SEP is assigned to a dedicated processor.
- Task splitting is performed whenever a task causes the utilization of the processor to exceed SEP.



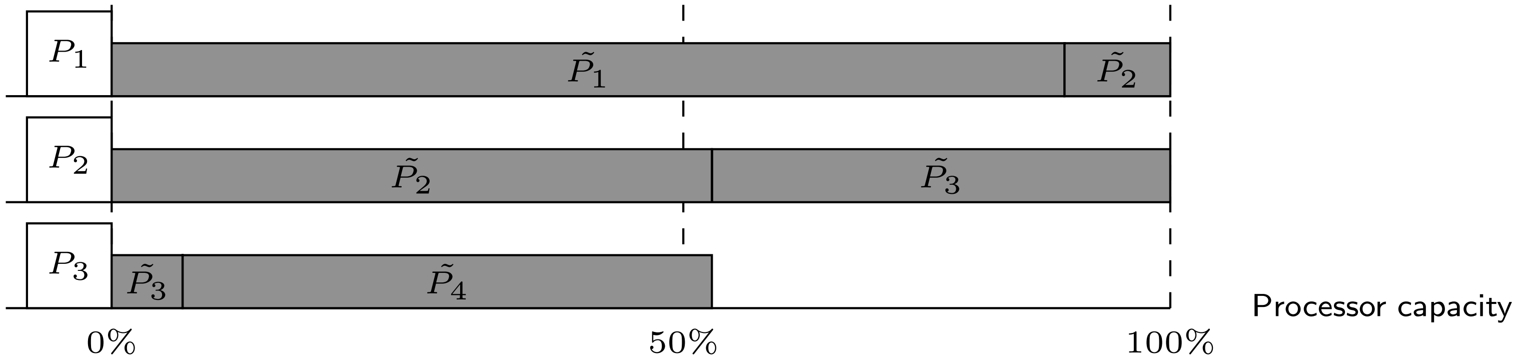
Sporadic Multiprocessor Scheduling (SMS) (II)

- heavy tasks execute on a dedicated processor.
- `split task` execute on `reserves`.
- The non-split tasks are scheduled under EDF scheduling algorithm.



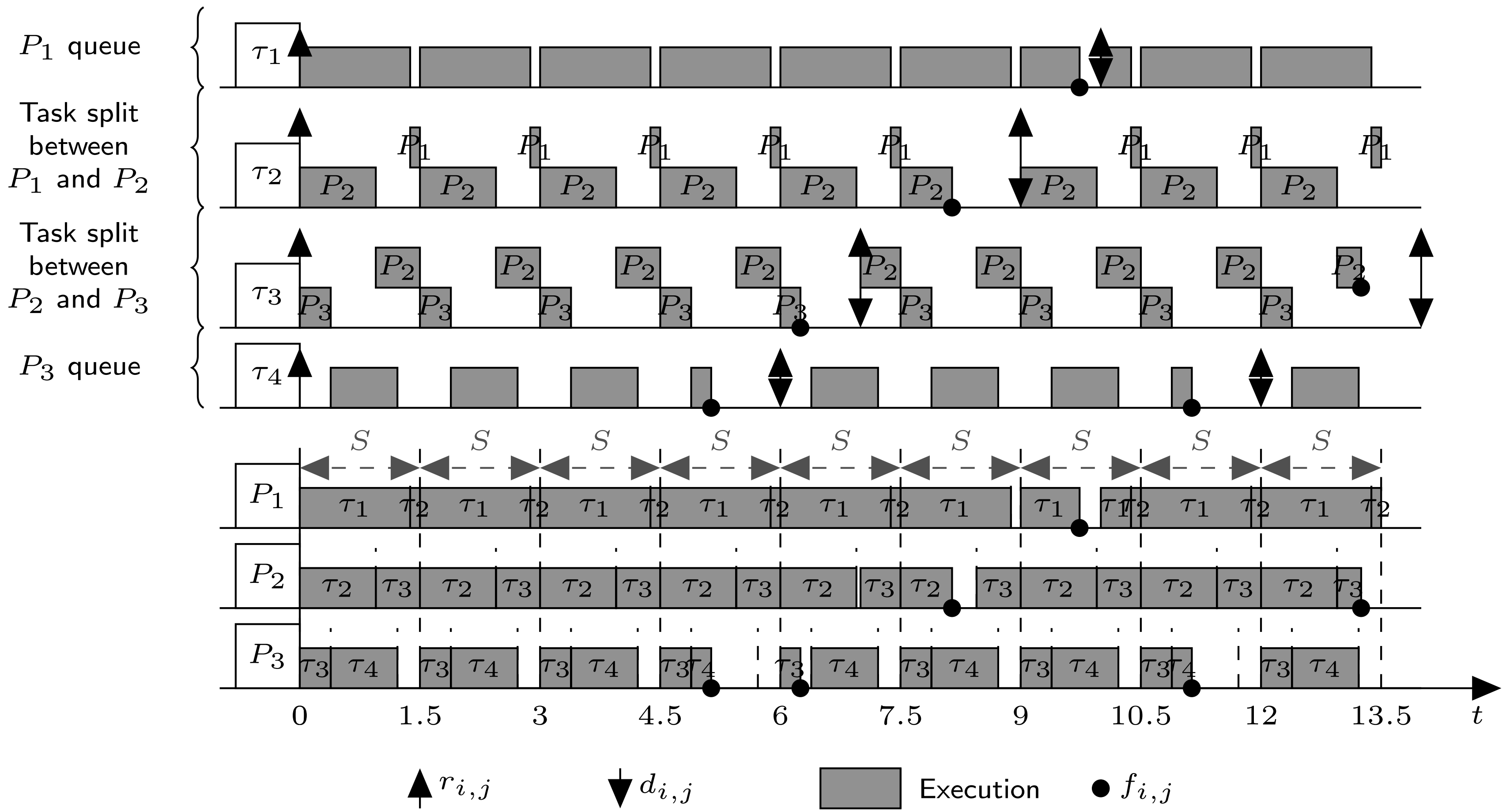
Notional Processor Scheduling - Fractional capacity (NPS-F) (I)

- NPS-F uses an approach based on `bins`. To each bin is assigned one or more tasks and there is one to one relation between each bin and each `notional processor`.
- Then, notional processor schedules tasks of each bin under EDF scheduling policy.
- In turn, all notional processors are implemented upon the m physical processors (P_1 to P_m) by the means of `reserves`.



Notional Processor Scheduling - Fractional capacity (NPS-F) (II)

- The dispatching algorithm is very simple, tasks are only allowed to execute within their reserves, that is, within reserves of the notional processors.



Conclusions (I)

- Global
  - \+ High utilization
  - \- Higher number of migrations (Cache misses)
  - \- Complex dispatcher
  - \- Shared queue (implies the use of synchronization mechanisms)

- Partitioned
  - \+ No migrations
  - \+ Simple dispatcher
  - \+ No need the use of synchronization mechanisms
  - \+ Lower number of preemptions
  - \- Low utilization
  - \- The offline assign algorithm

Conclusions (II)

- Semi-Partitioned: tries to get the advantages of the global and the partitioned
  - limited migrations
  - Simple dispatcher
  - No need the use of synchronization mechanisms
  - High utilization
  - Lower number of preemptions

Questions

Thank you for your attention!