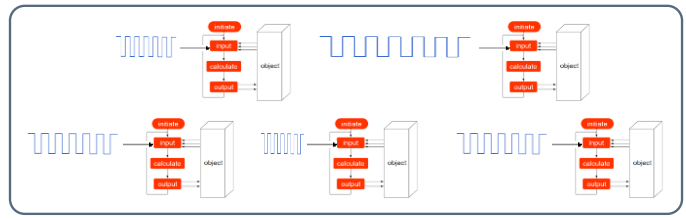
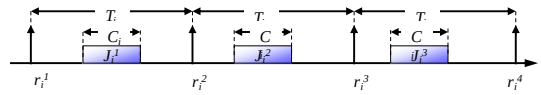
Real-time Systems



- N periodic tasks (of different rates/periods)



Utilization/workload: C_i/T_i

- How to schedule the jobs to avoid deadline miss?

On Single-processors

- Liu and Layland's Utilization Bound [1973]
  (the 19th most cited paper in computer science)

$$\sum_{\tau_i \in \tau} U_i \leq N(2^{1/N} - 1)$$

$$\Rightarrow \quad \text{the task set is schedulable}$$

number of tasks

- $N \to \infty, \quad N(2^{1/N} - 1) = 69.3\%$
- Scheduled by *RMS (Rate Monotonic Scheduling)*

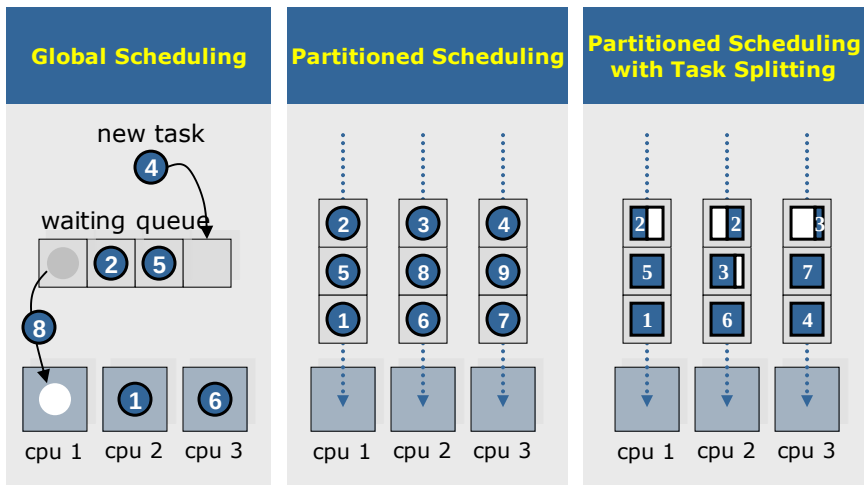
Quetion (since 1973)

- Find a multiprocessor scheduling algorithm that can achieve Liu and Layland's utilization bound

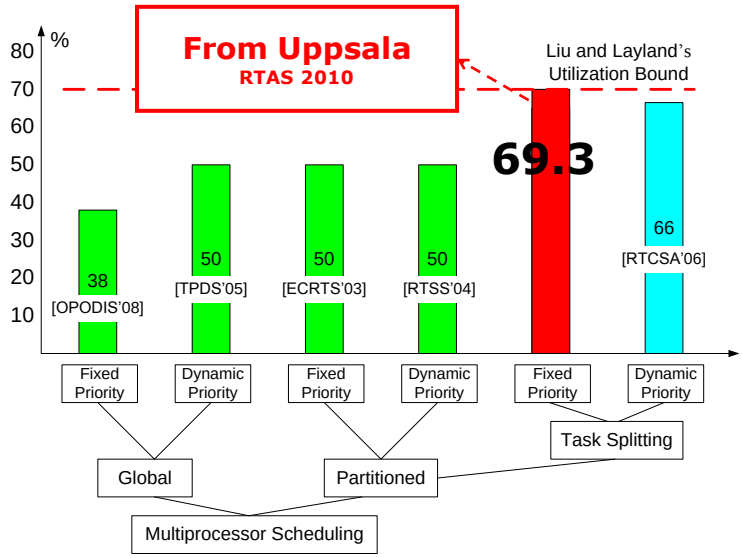
$$\frac{\sum C_i/T_i}{M} \leq N(2^{1/N} - 1) \Rightarrow \text{the task set is schedulable}$$

number of processors

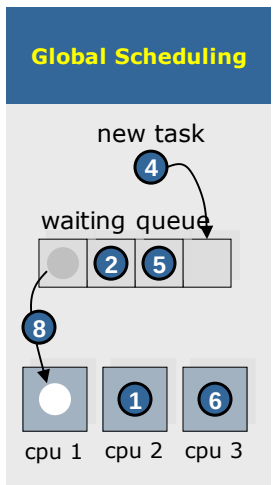
Multiprocessor Scheduling



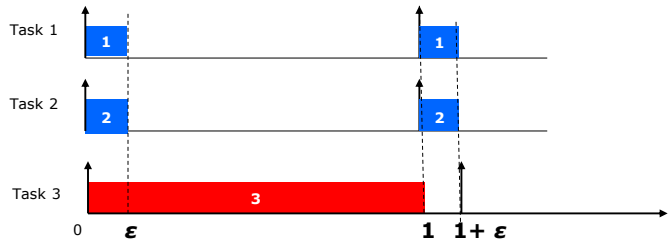
Best Known Results



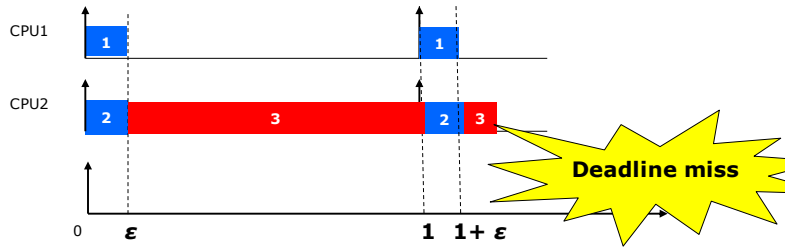
Problem with Global Scheduling



Global scheduling: Dhall's anomali



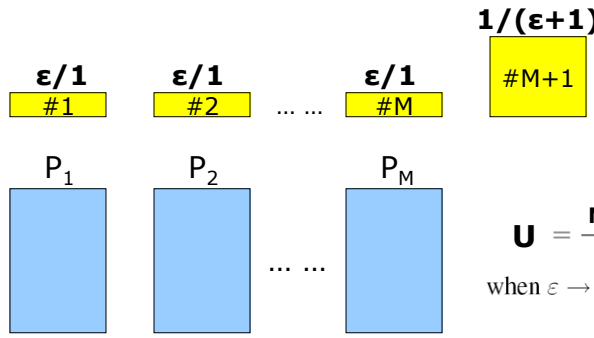
Global scheduling: Dhall's anomali



Schedule the 3 tasks on 2 CPUs using "RMS

Global scheduling: Dhall's anomali

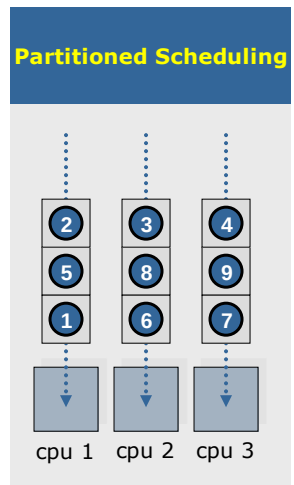
(M+1 tasks and M processors)



$$\mathbf{U} = \frac{\mathbf{M*\varepsilon + 1/(1+\varepsilon)}}{\mathbf{M}} \rightarrow \mathbf{0}$$

when $\varepsilon \rightarrow 0$ and $M \rightarrow +\infty$

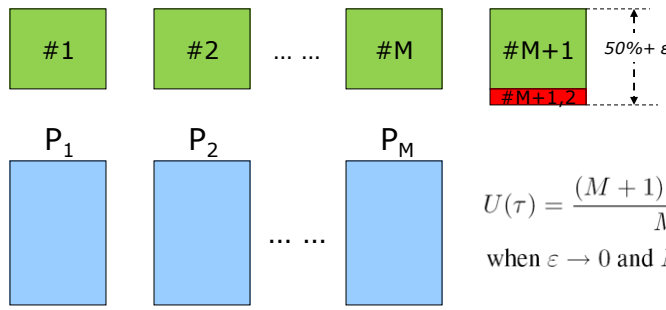
Problem with Partitioned Scheduling



Partitioned Scheduling: Utilization bounded by 50%

- The Partitioning Problem is similar to Bin-packing Problem (NP-hard)
- Limited Resource Usage, 50%

$\sum C_i/T_i \leq 1$
necessary condition to guarantee schedulability

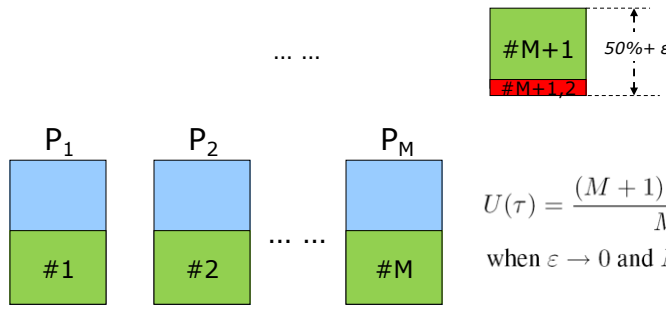


$$U(\tau) = \frac{(M+1)(0.5+\varepsilon)}{M} \rightarrow 0.5$$
when $\varepsilon \rightarrow 0$ and $M \rightarrow +\infty$

Partitioned Scheduling: Utilization bounded by 50%

- The Partitioning Problem is similar to Bin-packing Problem (NP-hard)
- Limited Resource Usage

$\sum C_i/T_i \leq 1$
necessary condition to guarantee schedulability



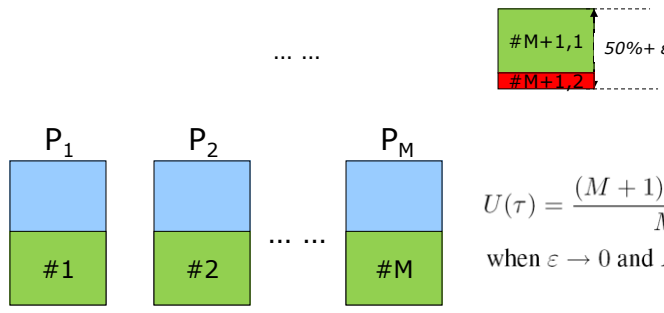
$$U(\tau) = \frac{(M+1)(0.5+\varepsilon)}{M} \rightarrow 0.5$$
when $\varepsilon \rightarrow 0$ and $M \rightarrow +\infty$

Partitioned Scheduling: Utilization bounded by 50%

- The Partitioning Problem is similar to Bin-packing Problem (NP-hard)
- Limited Resource Usage

$$\sum C_i/T_i \leq 1$$

necessary condition to guarantee schedulability



$$U(\tau) = \frac{(M+1)(0.5+\varepsilon)}{M} \rightarrow 0.5$$

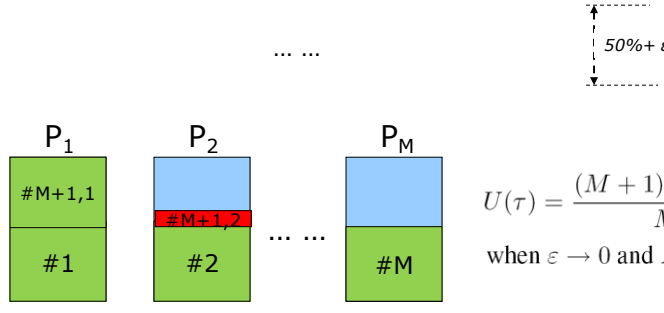
when $\varepsilon \rightarrow 0$ and $M \rightarrow +\infty$

Partitioned Scheduling: with job spliting

- The Partitioning Problem is similar to Bin-packing Problem (NP-hard)
- Limited Resource Usage

$$\sum C_i/T_i \leq 1$$

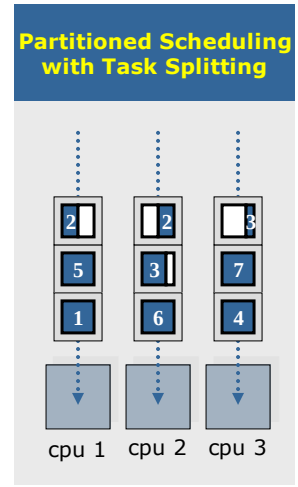
necessary condition to guarantee schedulability



$$U(\tau) = \frac{(M+1)(0.5+\varepsilon)}{M} \rightarrow 0.5$$

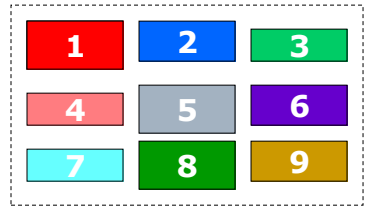
when $\varepsilon \rightarrow 0$ and $M \rightarrow +\infty$

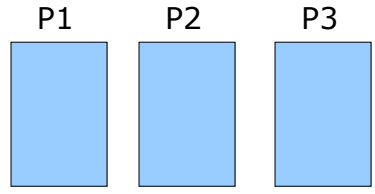
Fixed Priority Multiprocessor Scheduling with task partition and job spliting



Partitioned Scheduling

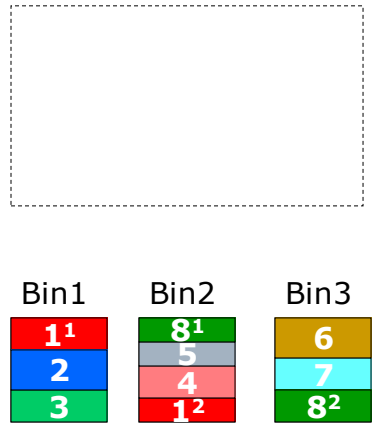
- Partitioning





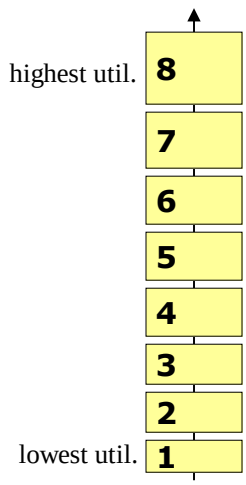
Bin-Packing with Item Splitting

- Resource can be "fully" (better) utilized



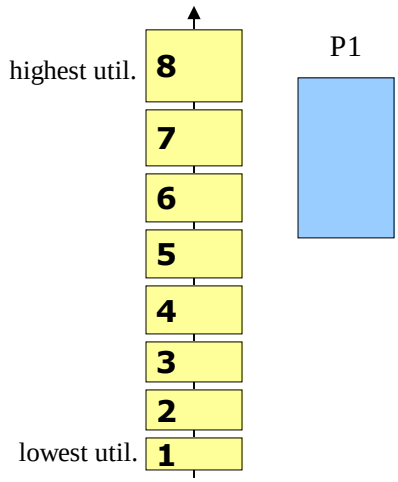
Lakshmanan's Algorithm [ECRTS'09]

- Sort all tasks in decreasing order of utilization



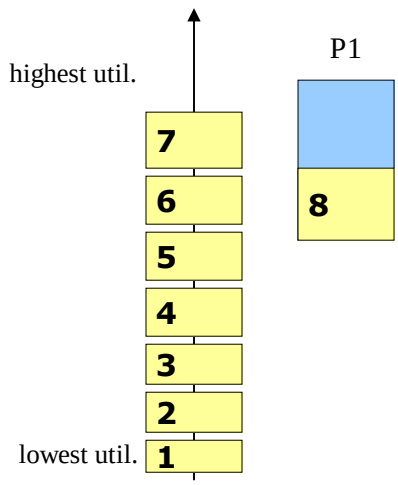
Lakshmanan's Algorithm [ECRTS'09]

- Pick up one processor, and assign as many tasks as possible



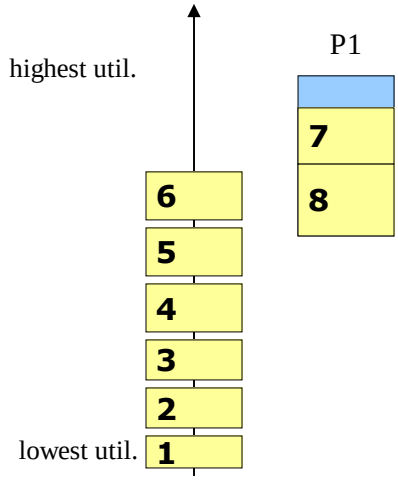
Lakshmanan's Algorithm [ECRTS'09]

- Pick up one processor, and assign as many tasks as possible



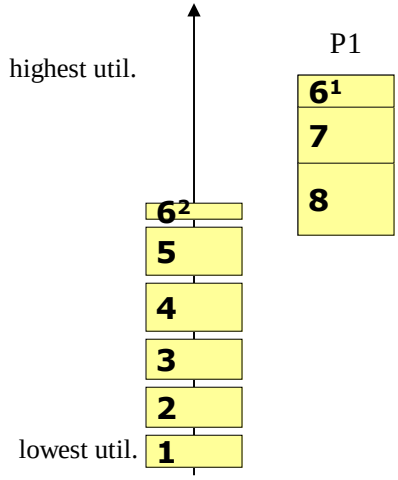
Lakshmanan's Algorithm [ECRTS'09]

- Pick up one processor, and assign as many tasks as possible



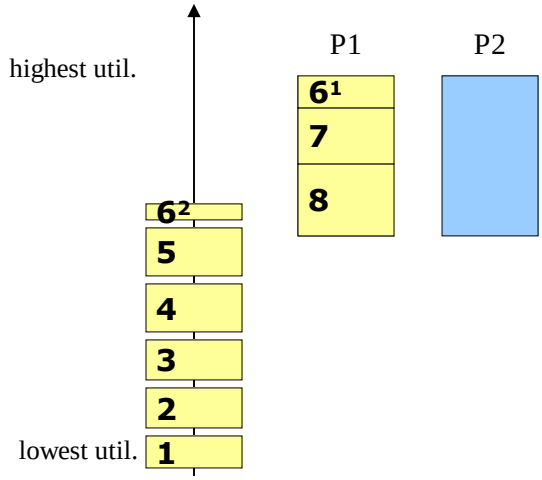
Lakshmanan's Algorithm [ECRTS'09]

- Pick up one processor, and assign as many tasks as possible



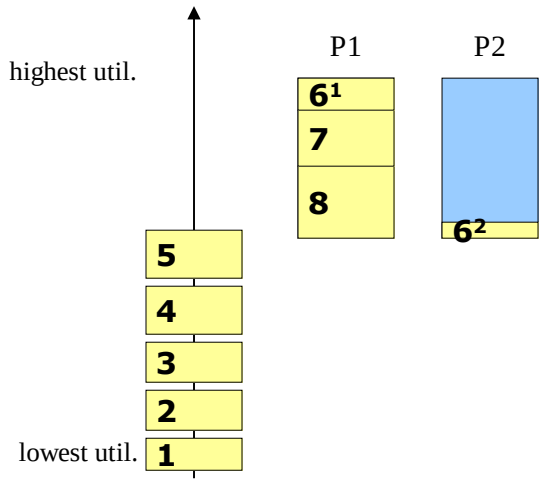
Lakshmanan's Algorithm [ECRTS'09]

- Pick up one processor, and assign as many tasks as possible



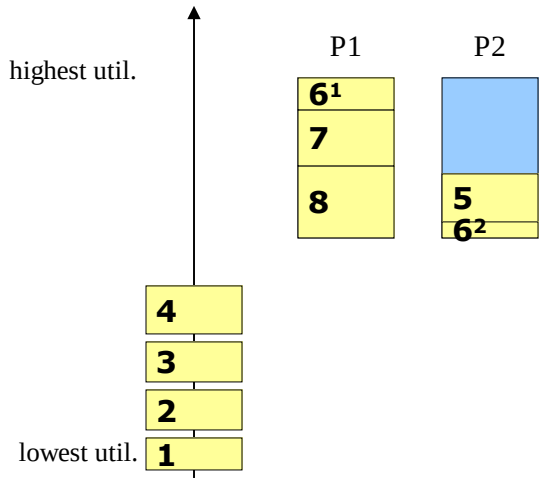
Lakshmanan's Algorithm [ECRTS'09]

- Pick up one processor, and assign as many tasks as possible



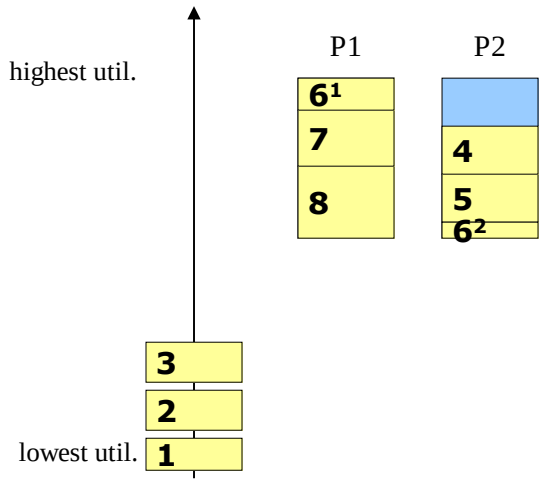
Lakshmanan's Algorithm [ECRTS'09]

- Pick up one processor, and assign as many tasks as possible



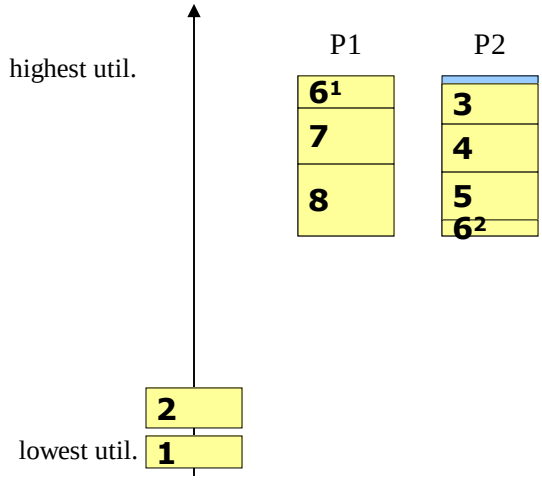
Lakshmanan's Algorithm [ECRTS'09]

- Pick up one processor, and assign as many tasks as possible



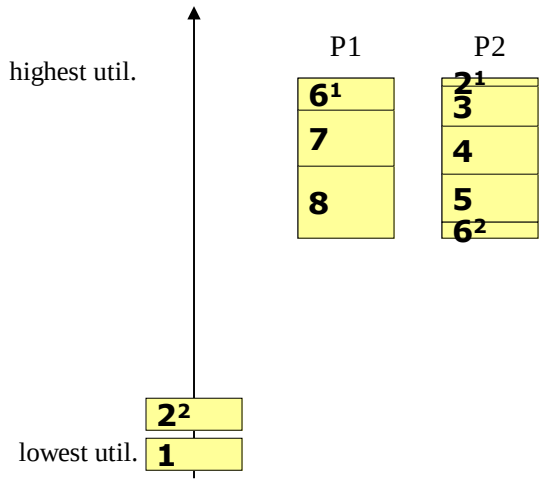
Lakshmanan's Algorithm [ECRTS'09]

- Pick up one processor, and assign as many tasks as possible



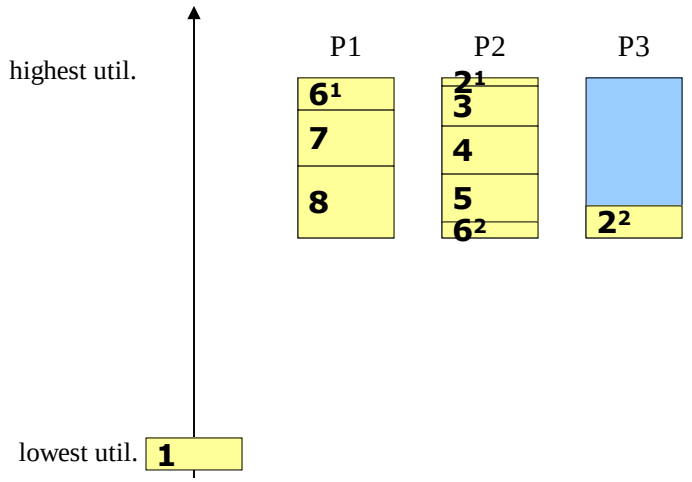
Lakshmanan's Algorithm [ECRTS'09]

- Pick up one processor, and assign as many tasks as possible



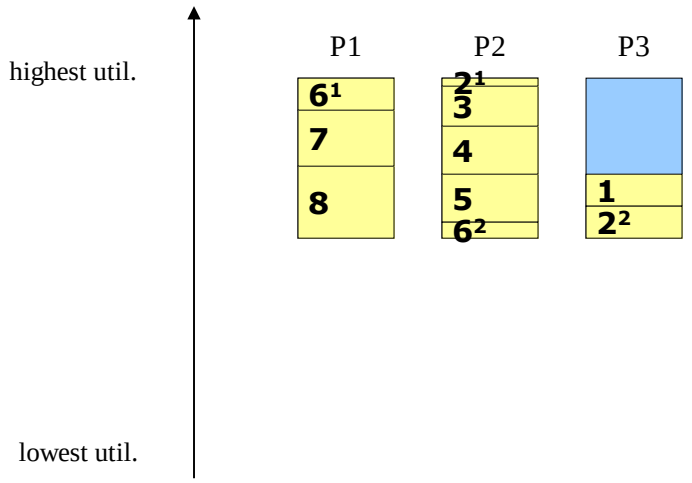
Lakshmanan's Algorithm [ECRTS'09]

- Pick up one processor, and assign as many tasks as possible



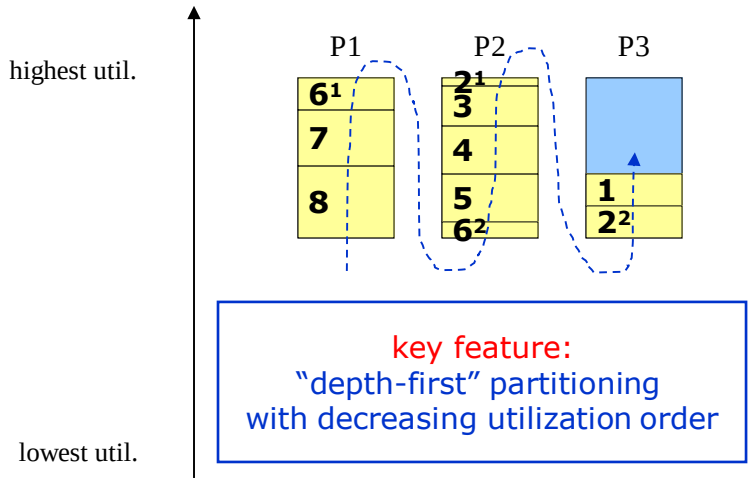
Lakshmanan's Algorithm [ECRTS'09]

- Pick up one processor, and assign as many tasks as possible



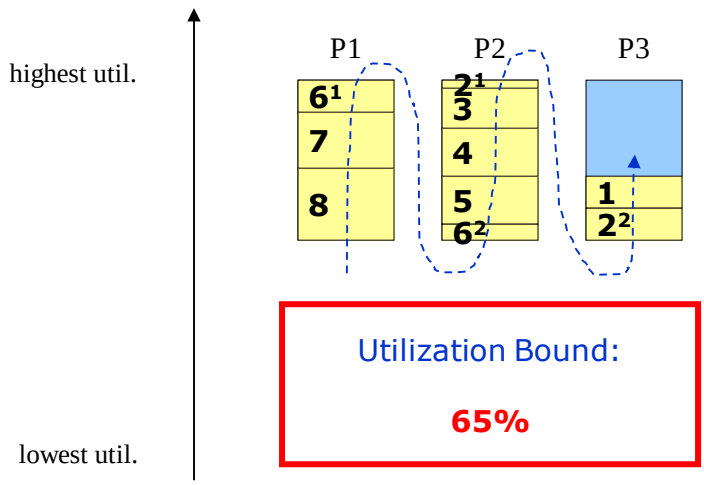
Lakshmanan's Algorithm [ECRTS'09]

- Pick up one processor, and assign as many tasks as possible



Lakshmanan's Algorithm [ECRTS'09]

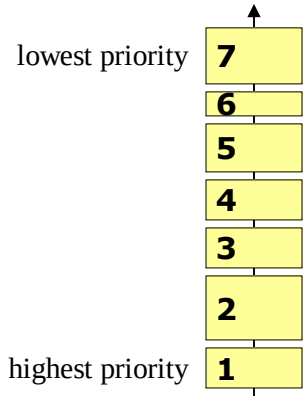
- Pick up one processor, and assign as many tasks as possible



Breadth-First Partitioning Algorithms

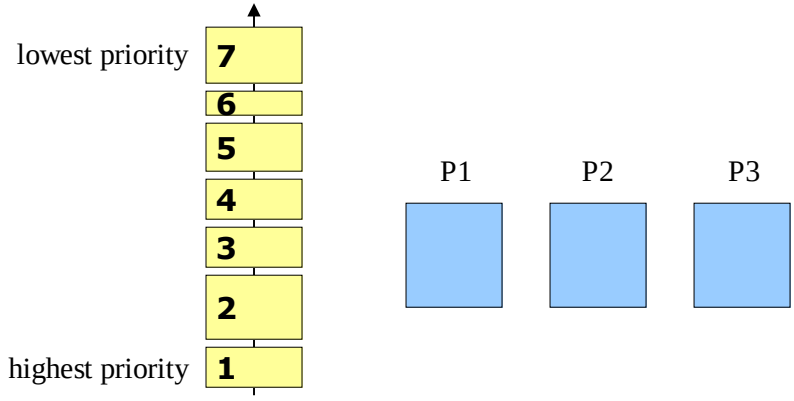
[RTAS 2010]

- Sort all tasks in increasing priority order



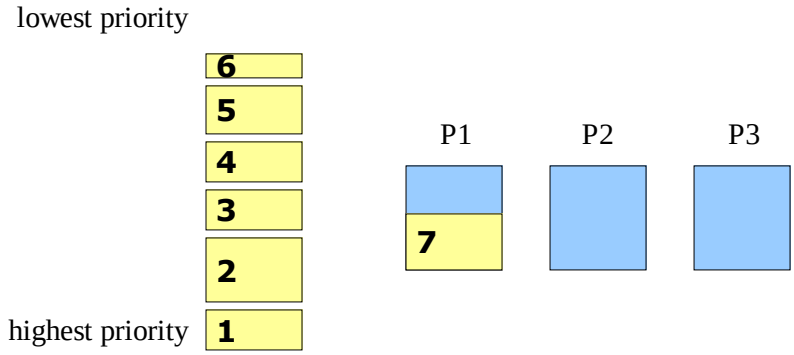
Breadth-First Partitioning Algorithms

- Select the processor on which the assigned utilization is the lowest



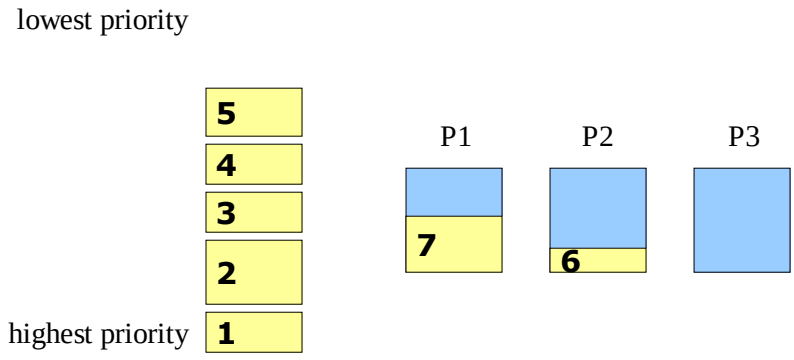
Breadth-First Partitioning Algorithms

- Select the processor on which the assigned utilization is the lowest



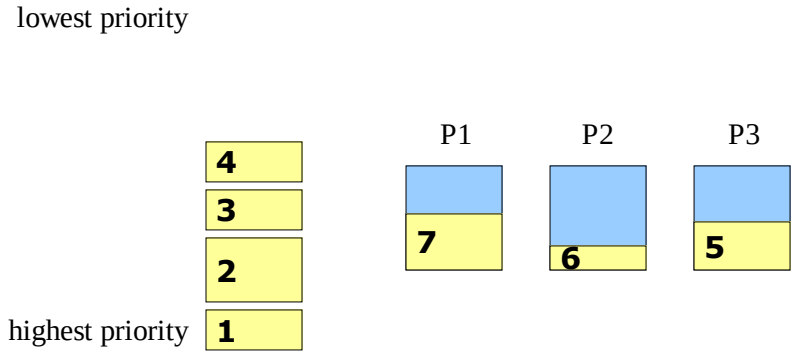
Breadth-First Partitioning Algorithms

- Select the processor on which the assigned utilization is the lowest



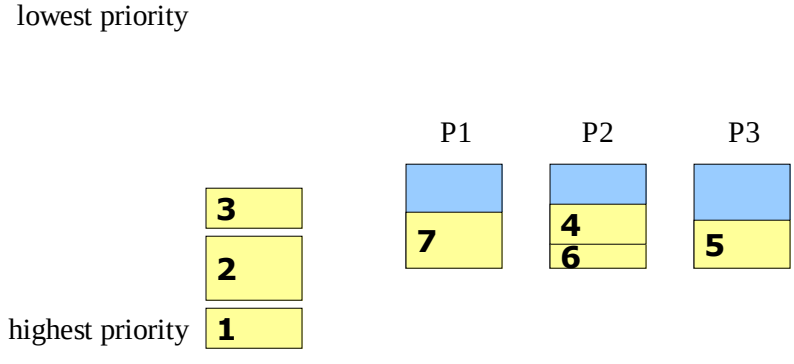
Breadth-First Partitioning Algorithms

- Select the processor on which the assigned utilization is the lowest



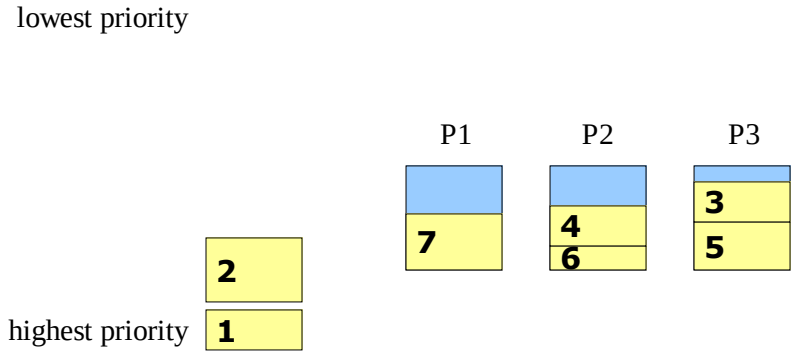
Breadth-First Partitioning Algorithms

- Select the processor on which the assigned utilization is the lowest



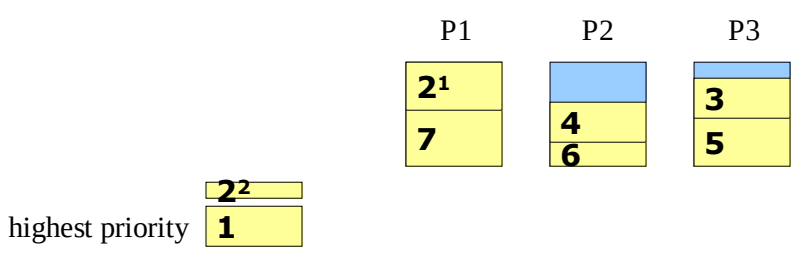
Breadth-First Partitioning Algorithms

- Select the processor on which the assigned utilization is the lowest



Breadth-First Partitioning Algorithms

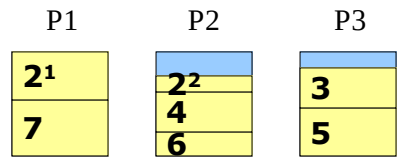
- Select the processor on which the assigned utilization is the lowest



Breadth-First Partitioning Algorithms

- Select the processor on which the assigned utilization is the lowest

lowest priority

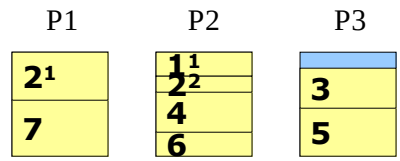


highest priority 1

Breadth-First Partitioning Algorithms

- Select the processor on which the assigned utilization is the lowest

lowest priority



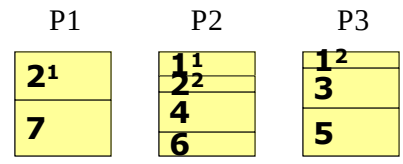
highest priority 1²

Breadth-First Partitioning Algorithms

- Select the processor on which the assigned utilization is the lowest

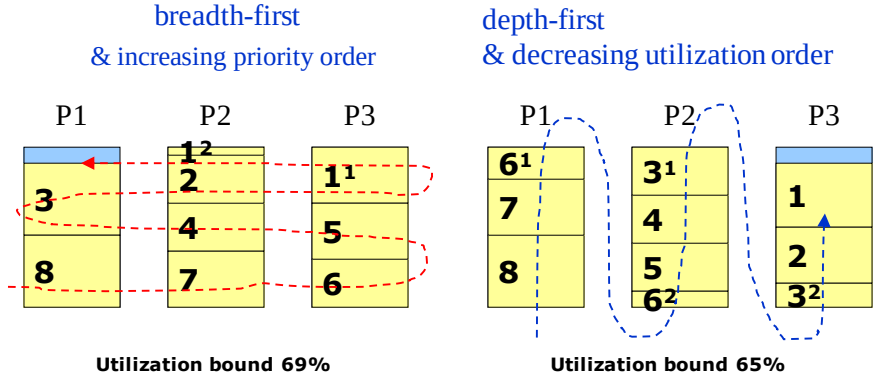
Utilization Bound:

69%



Comparison

Why is the breadth algorithm better?



Fact for "Light" Tasks
whose utilization less than 0.41

For a task set in which each task τ_i satisfies

$$U_i \leq \frac{\Theta(N)}{1+\Theta(N)}$$

we have

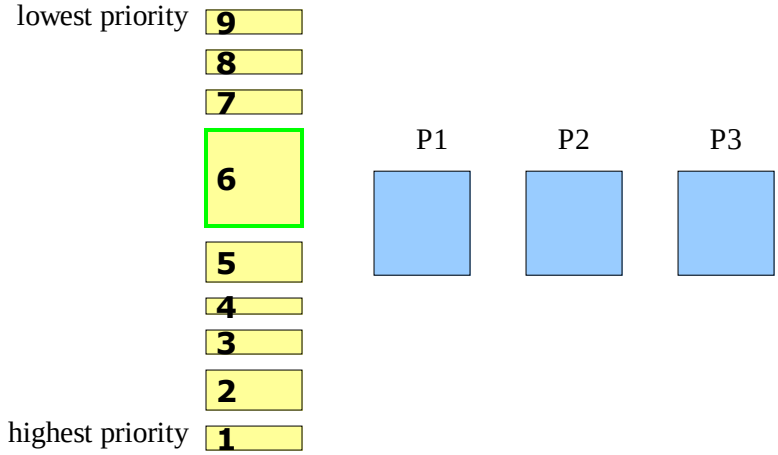
$$\frac{\sum C_i/T_i}{M} \leq N(2^{1/N}-1)$$

$\Rightarrow$ the task set is schedulable

$$\Theta(N) = N(2^{\frac{1}{N}}-1) \qquad N \to \infty, \ \frac{\Theta(N)}{1+\Theta(N)} \doteq 0.41$$

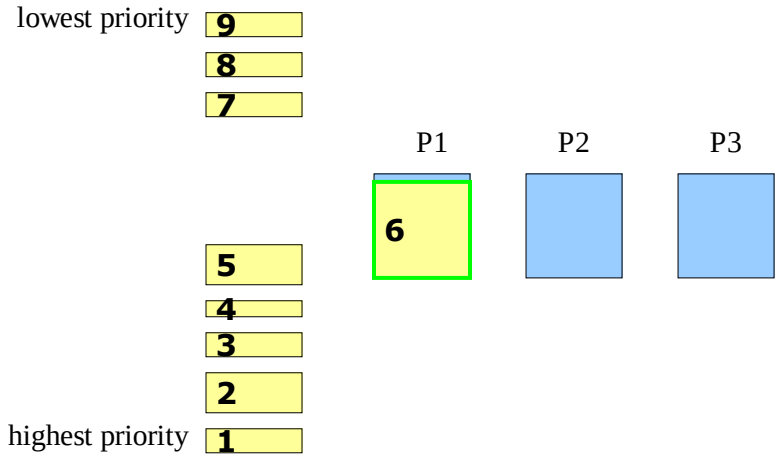
Solution for Heavy Tasks
whose utilization larger than 0.41

- Pre-assigning the heavy tasks (that may have low priorities)



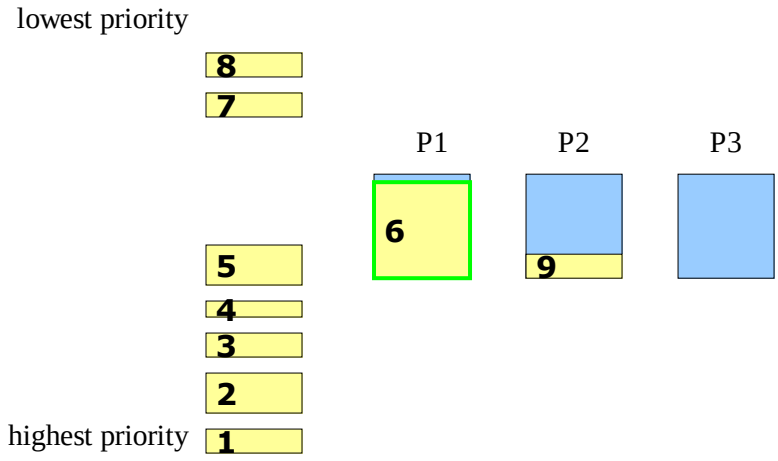
Solution for Heavy Tasks

- Pre-assigning the heavy tasks (that may have low priorities)



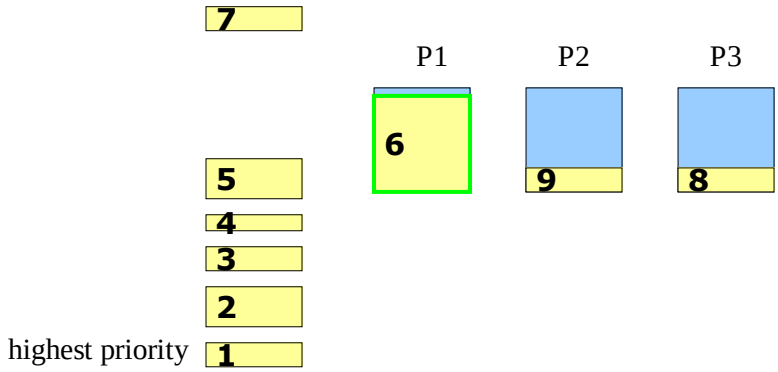
Solution for Heavy Tasks

- Pre-assigning the heavy tasks (that may have low priorities)



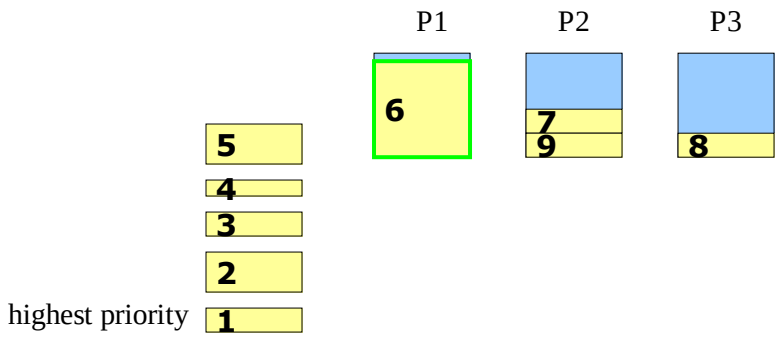
Solution for Heavy Tasks

- Pre-assigning the heavy tasks (that may have low priorities)



Solution for Heavy Tasks

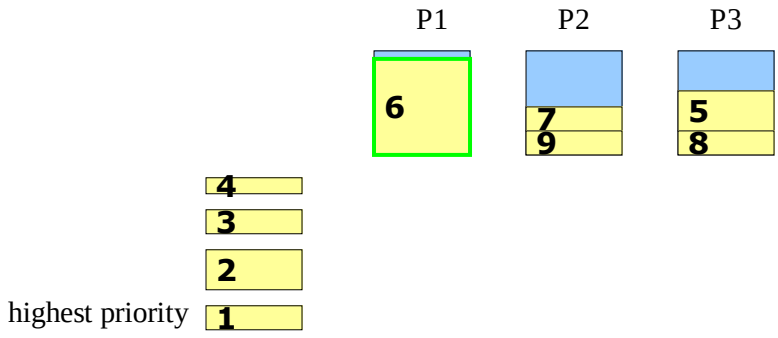
- Pre-assigning the heavy tasks (that may have low priorities)



Solution for Heavy Tasks

- Pre-assigning the heavy tasks (that may have low priorities)

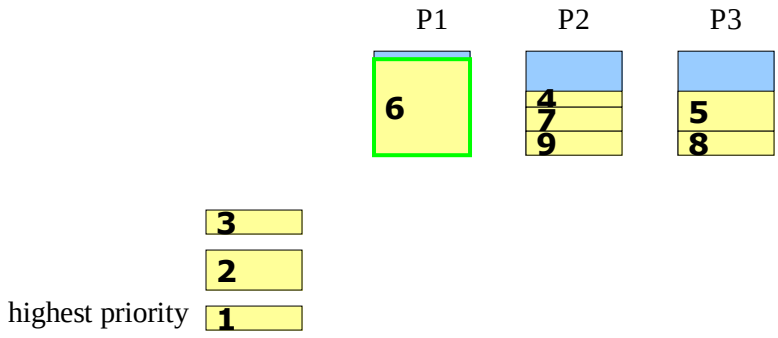
lowest priority



Solution for Heavy Tasks

- Pre-assigning the heavy tasks (that may have low priorities)

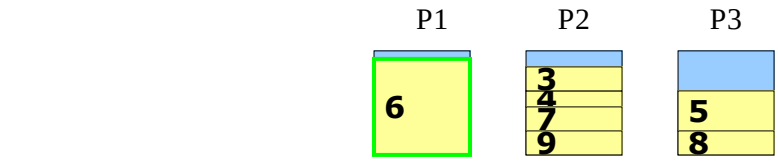
lowest priority

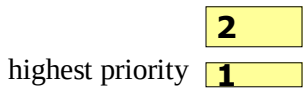


Solution for Heavy Tasks

- Pre-assigning the heavy tasks (that may have low priorities)

lowest priority

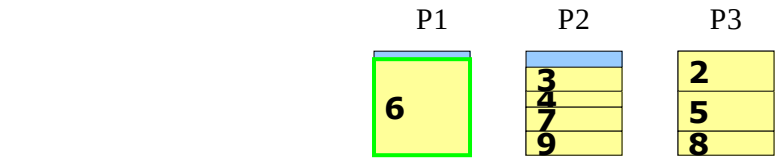




Solution for Heavy Tasks

- Pre-assigning the heavy tasks (that may have low priorities)

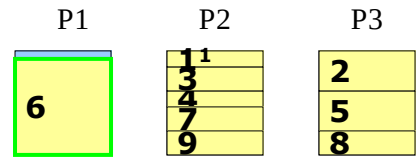
lowest priority

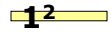


Solution for Heavy Tasks

- Pre-assigning the heavy tasks (that may have low priorities)

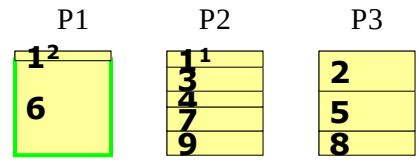
lowest priority



highest priority 

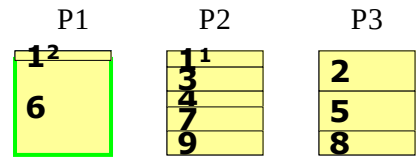
Solution for Heavy Tasks

- Pre-assigning the heavy tasks (that may have low priorities)



Solution for Heavy Tasks

- Pre-assigning the heavy tasks (that may have low priorities)



avoid to split heavy tasks
(that may have low priorities)

FACT

- By introducing the pre-assignment mechanism, we have

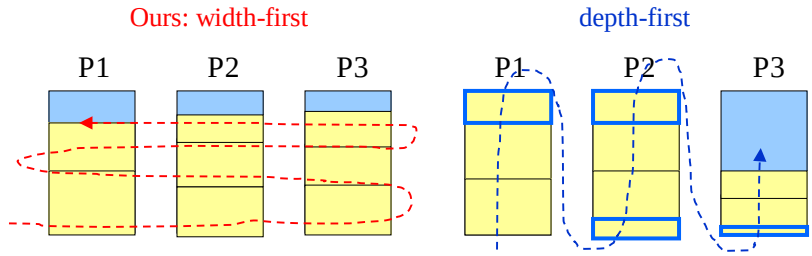
$$\frac{\sum C_i/T_i}{M} \leq N(2^{1/N} - 1)$$

$\Rightarrow$ the task set is schedulable

Liu and Layland's utilization bound for all task sets!

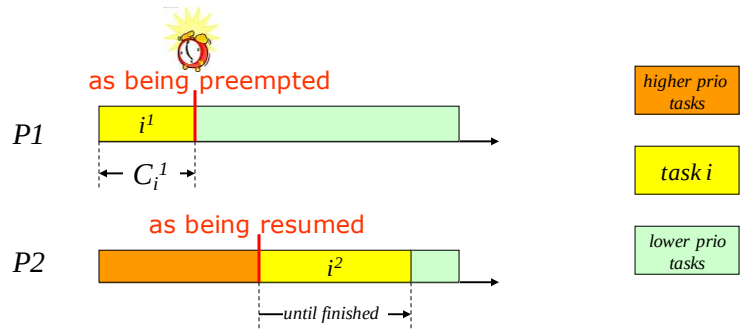
Overhead

- In both previous algorithms and ours
  - The number of task splitting is at most *M-1*
    - task splitting -> extra "migration/preemption"
  - Our algorithm on average has less task splitting



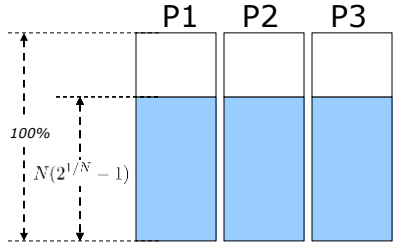
Implementation

- Easy!
  - One timer for each split task
  - Implemented as "task migration"



Further Improvement

Partioning using response time analysis



Schedulable?