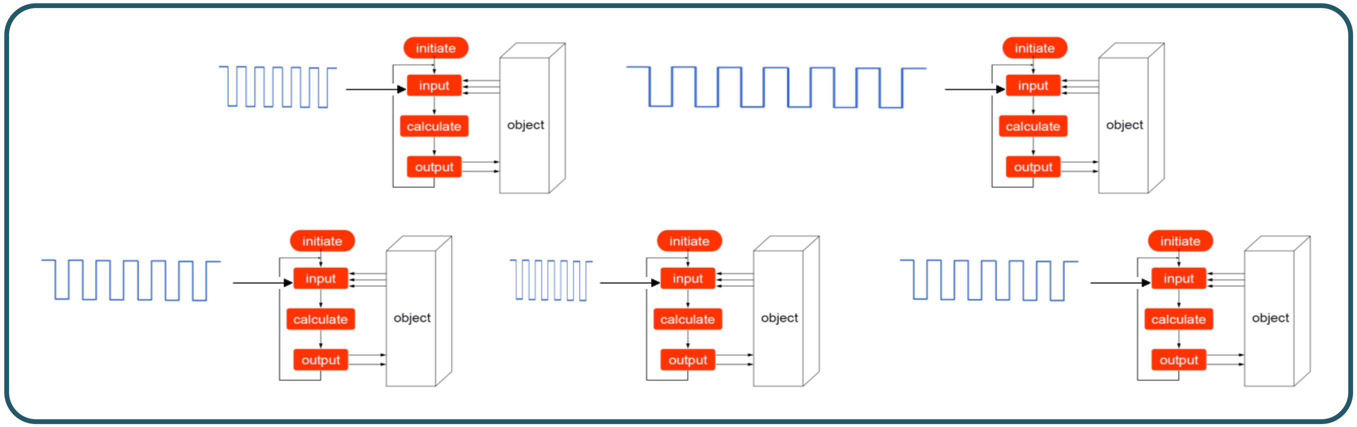
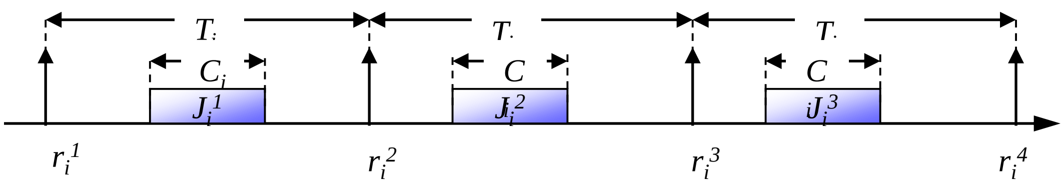
Multiprocessor scheduling of infinite tasks

- Task models as uniprocessor scheduling
  - Periodic tasks
  - Sporadic tasks
- Partition-based/global scheduling
  - Static partitioning of tasks
  - Dynamic partitioning av jobs

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

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

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

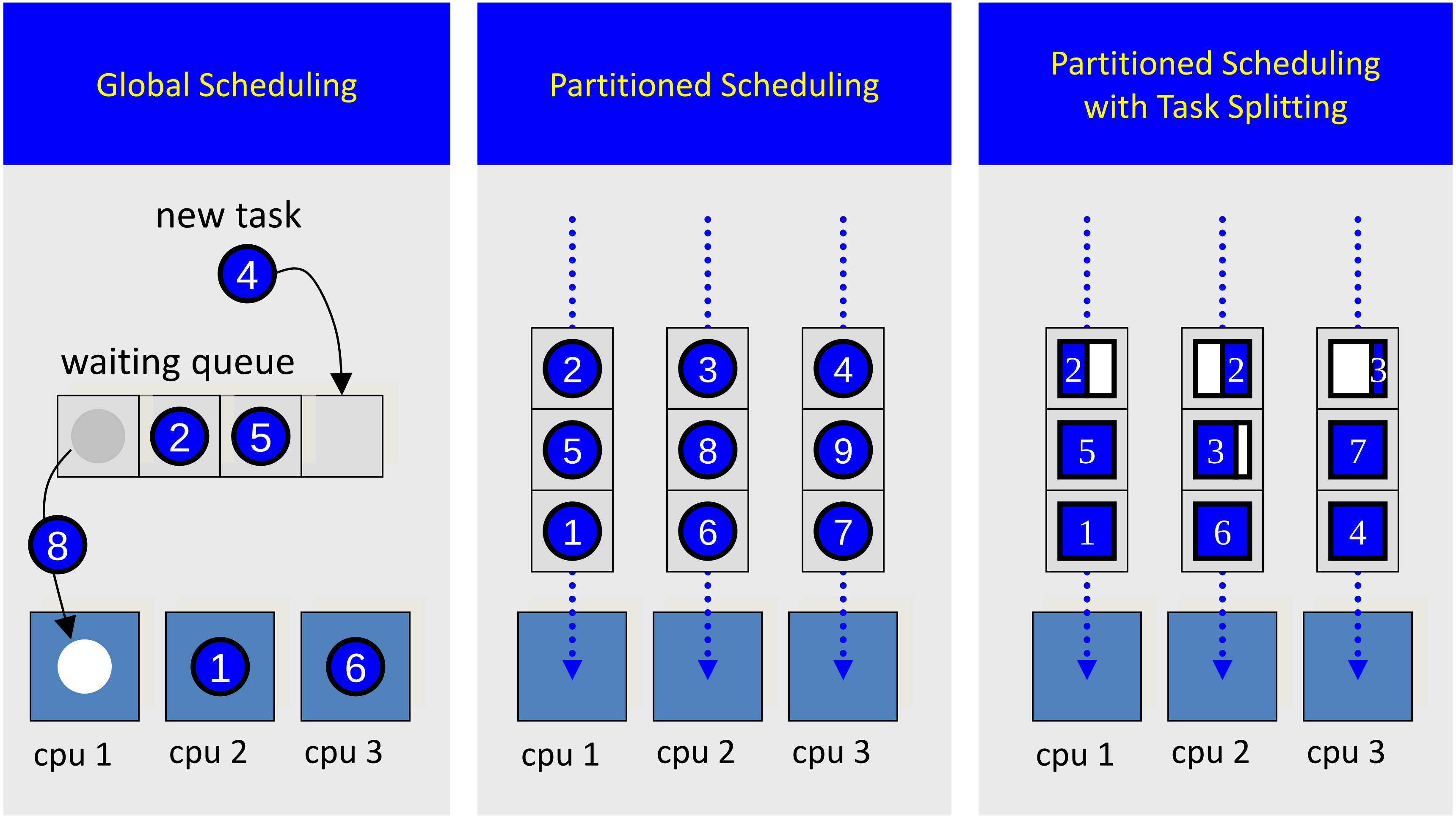
number of processors

Multiprocessor scheduling of sequential tasks

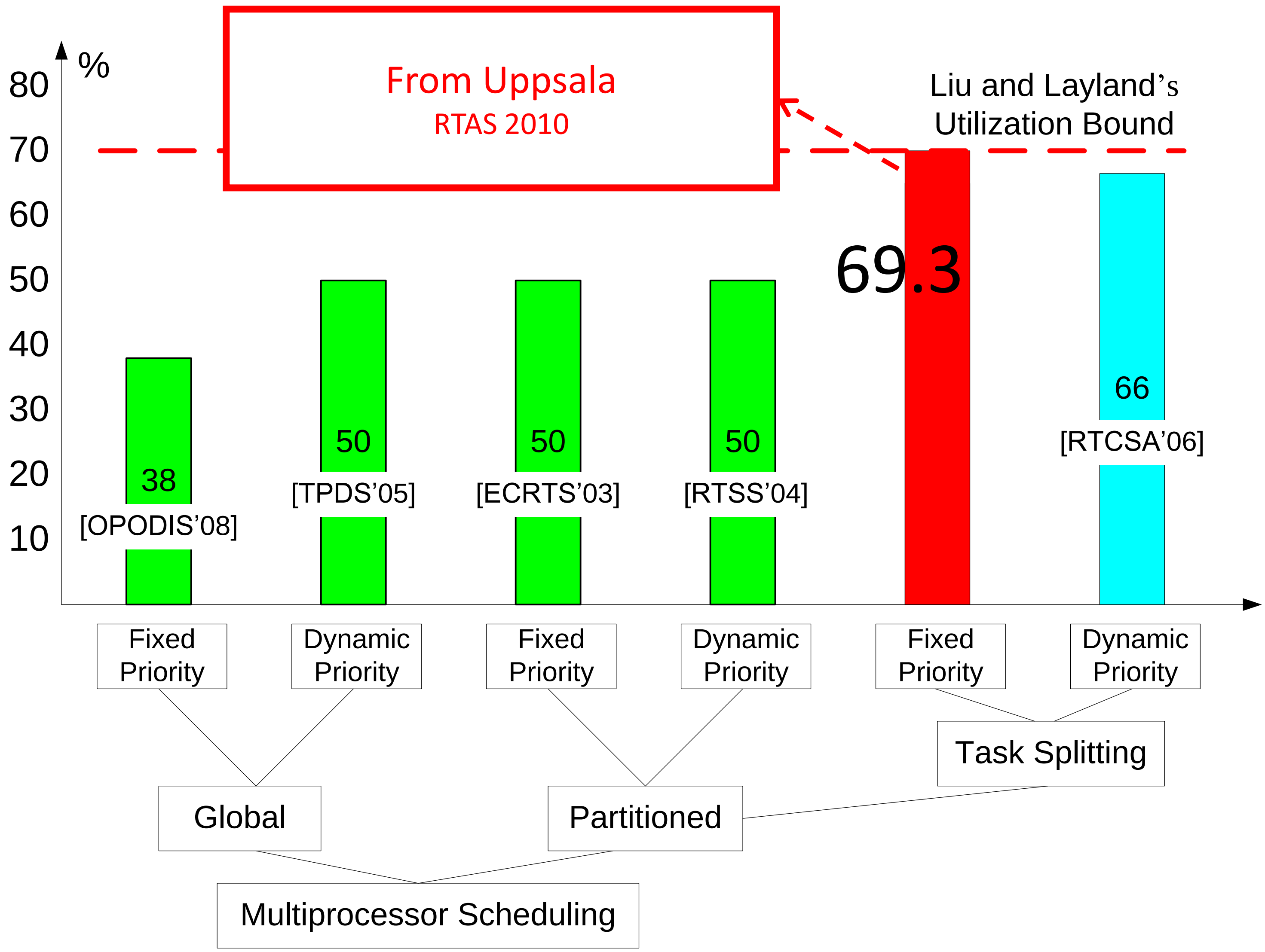
– static and dynamic task assignment

- Global scheduling
  - Dynamic task assignment
    - Any instance of any task may execute on any processor
    - Task migration
- Partitioned scheduling
  - Static task assignment
    - Each task may only execute on a fixed processor
    - No task migration
- Semi-partitioned scheduling
  - Static task assignment
    - Each instance (or part of it) of a task is assigned to a fixed processor
    - task instance or part of it may migrate

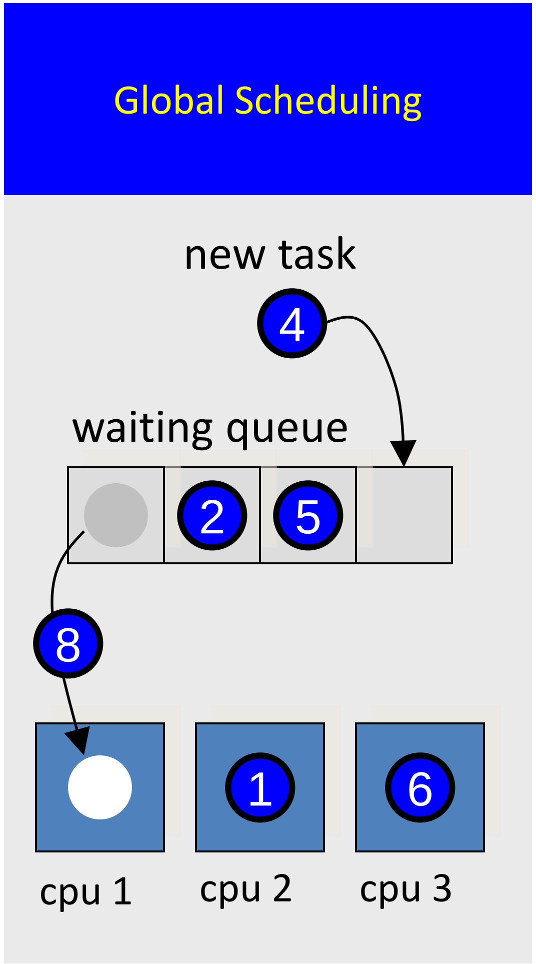
Multiprocessor Scheduling of sequential tasks



Best Known Results



Global Scheduling



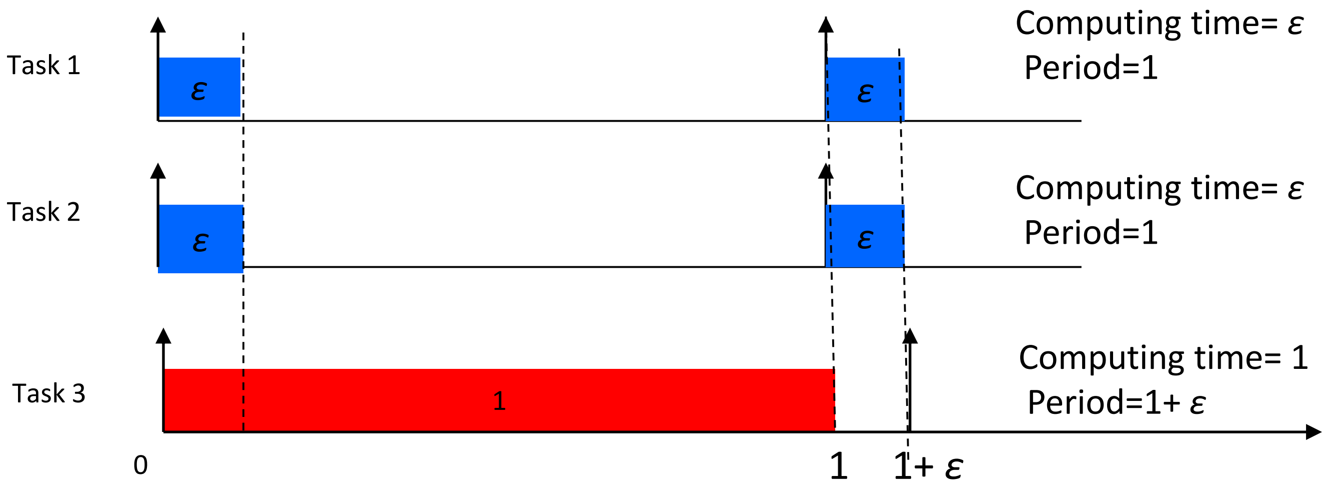
- All ready tasks are kept in a global queue
- When selected for execution, a task can be assigned to any processor, even after being preempted (task migration to another processor!)
  - Task migration may add overheads

Global scheduling Algorithms

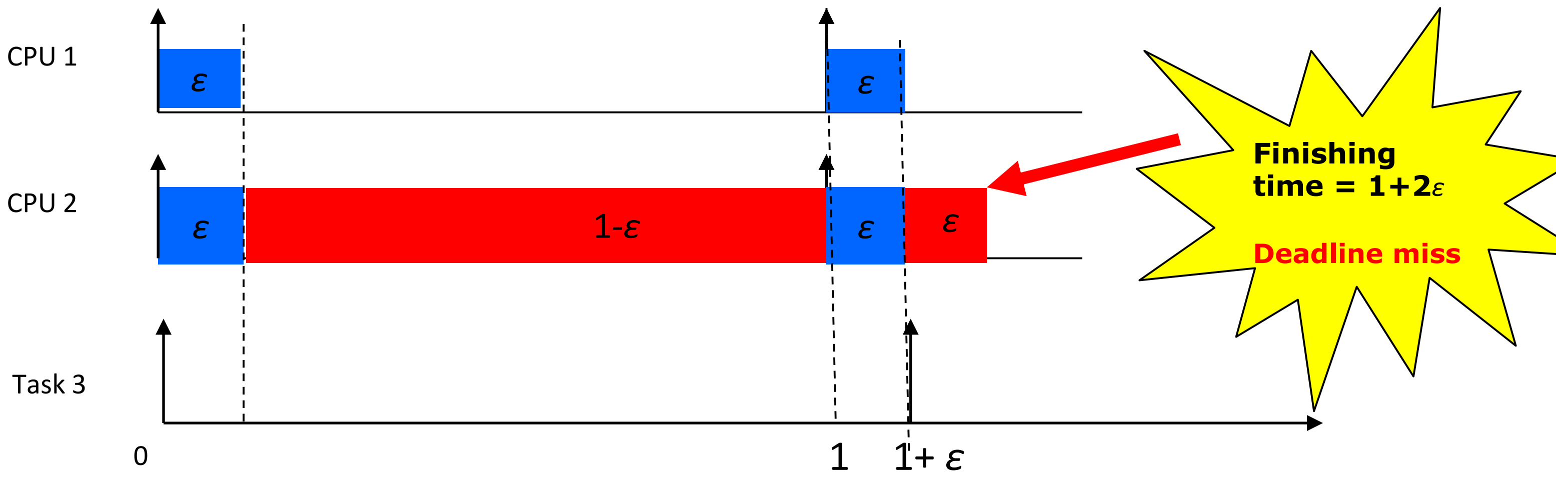
Any algorithm for single processor scheduling may work, but schedulability analysis is non-trivial

- EDF (optimal for single processor)
  - Unfortunately EDF is not optimal for multiprocessors
  - No exact, but only sufficient schedulability test known
- Fixed Priority Scheduling e.g. RM
  - Difficult to find the optimal priority order
  - Difficult to check the schedulability
  - Suffer from the Dhal's anomali

Global scheduling: Dhall's anomali

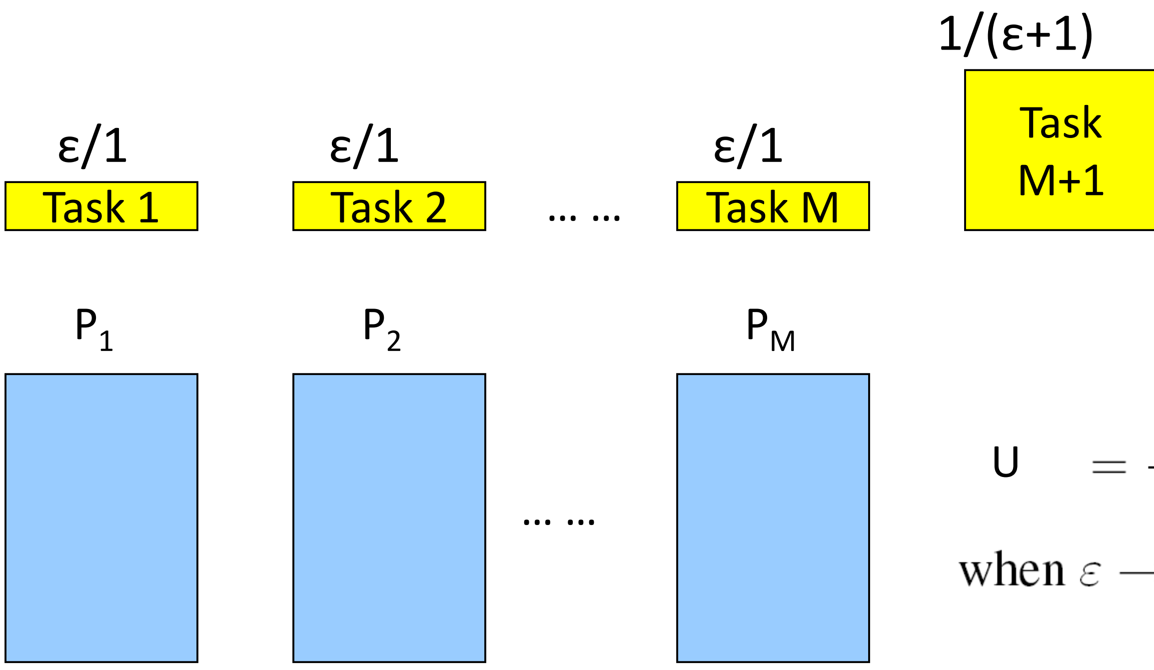


Global scheduling: Dhall's anomali



Global scheduling: Dhall's anomali

(M+1 tasks and M processors)



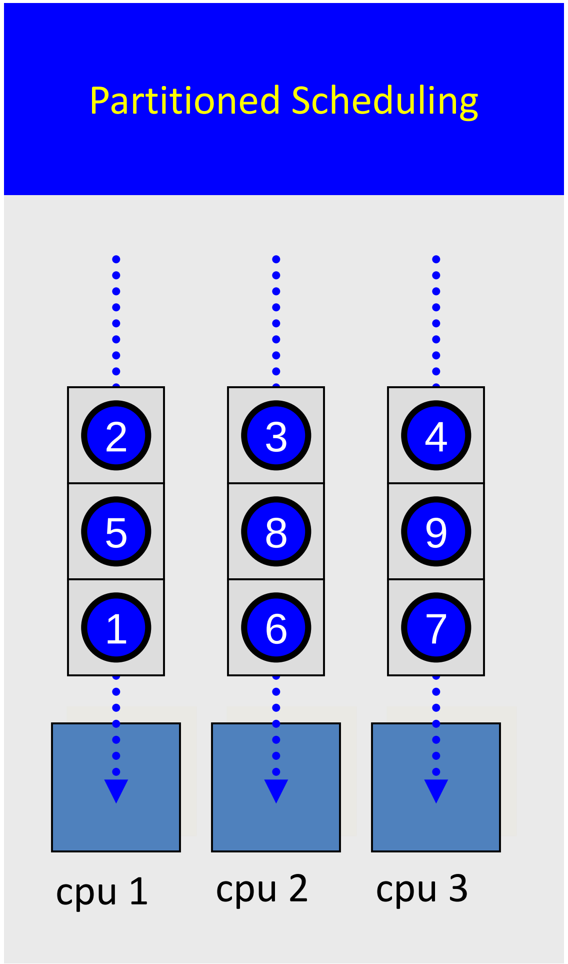
$$U = \frac{M^*\varepsilon + 1/(1+\varepsilon)}{M} \rightarrow 0$$

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

Global Scheduling: + and -

- Advantages:
  - Supported by most multiprocessor operating systems
  - 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)
- Disadvantages:
  - Few results on schedulability analysis from single-processor apply here
  - No “optimal” algorithms known except idealized assumption (Pfair sch)
  - Poor resource utilization for hard timing constraints
    - No more than 50% resource utilization can be guaranteed for hard RT tasks
  - Scheduling anomalies

Partition-Based Scheduling



Partitioned scheduling

- Two steps:
  - Determine a mapping of tasks to processors
  - Perform run-time scheduling

Bin-packing algorithms

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

Example: 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

Example: Partitioned with RMS

Rate-Monotonic-First-Fit (RMFF): [Dhall and Liu, 1978]

- First, sort the tasks in the order of increasing periods.
- Task Assignment
  - All tasks are assigned in the First Fit manner starting from *the task with highest priority*
  - A task can be assigned to a processor if all the tasks assigned to the processor are RM-schedulable i.e.
    - the total utilization of tasks assigned on that processor is bounded by $n(2^{1/n}-1)$ where n is the number of tasks assigned.

      (One may also use the Precise test to get a better assignment!)
  - Add a new processor if needed for the RM-test.

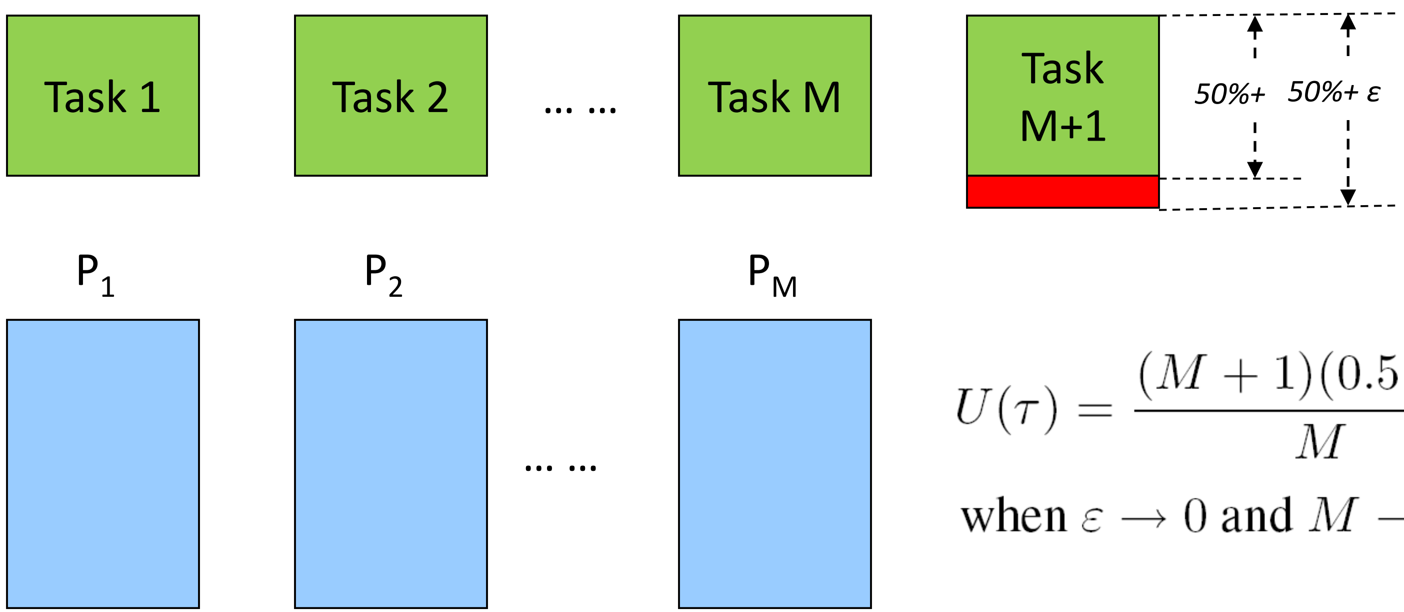
Partitioned Scheduling:
Utilization bounded by 50%

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

$$\sum C_i/T_i \le 1$$

necessary condition to guarantee schedulability

Utilization = 50% for task 1 … M and 50 *%+ ε* for task M+1



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

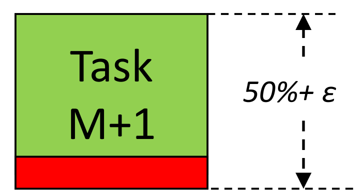
when $\varepsilon \to 0$ and $M \to +\infty$

Partitioned Scheduling:
Utilization bounded by 50%

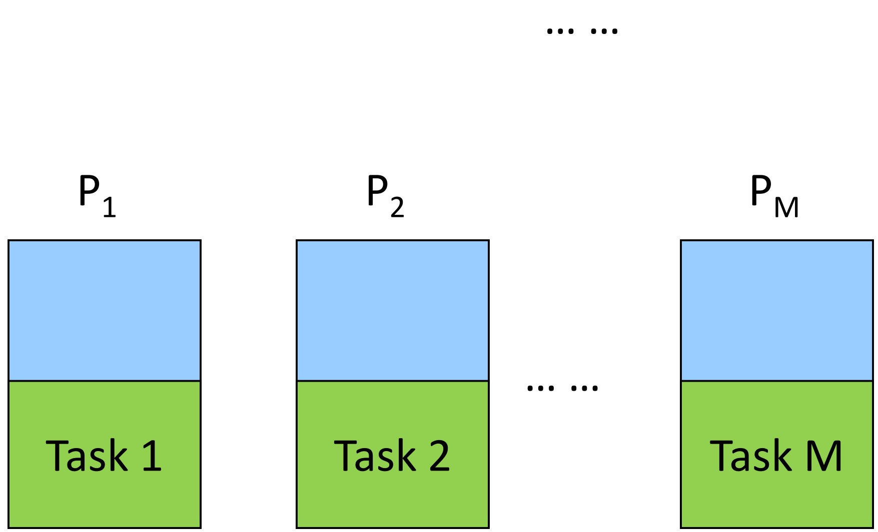
- The Partitioning Problem is similar to Bin-packing Problem (NP-hard)
- Limited Resource Usage, 50%

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

necessary condition to guarantee schedulability



No processor can run Task M+1 without being overloaded



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

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

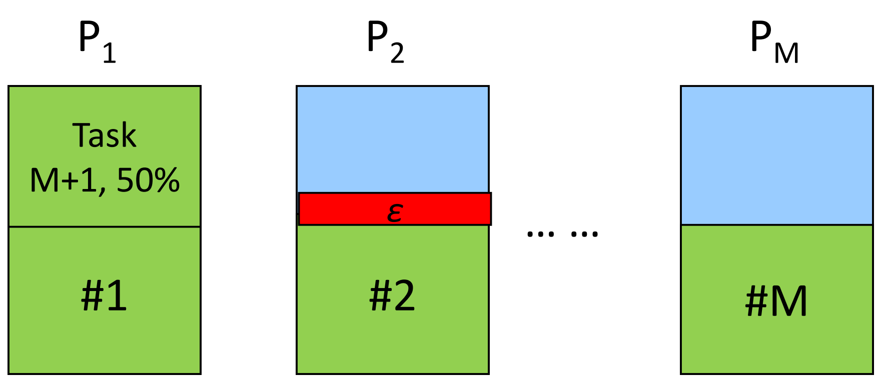
Partitioned Scheduling: with job spliting

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

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

necessary condition to guarantee schedulability

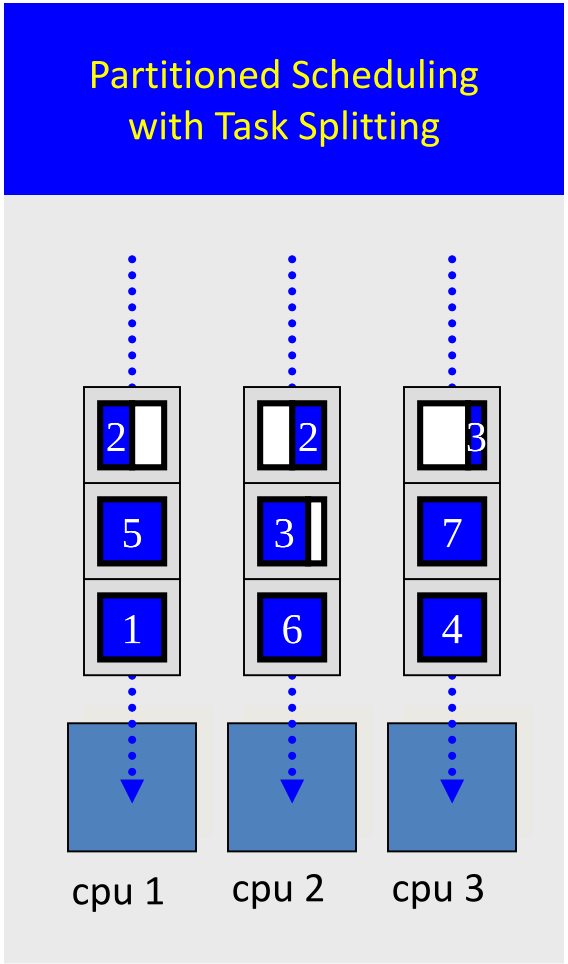
… …



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

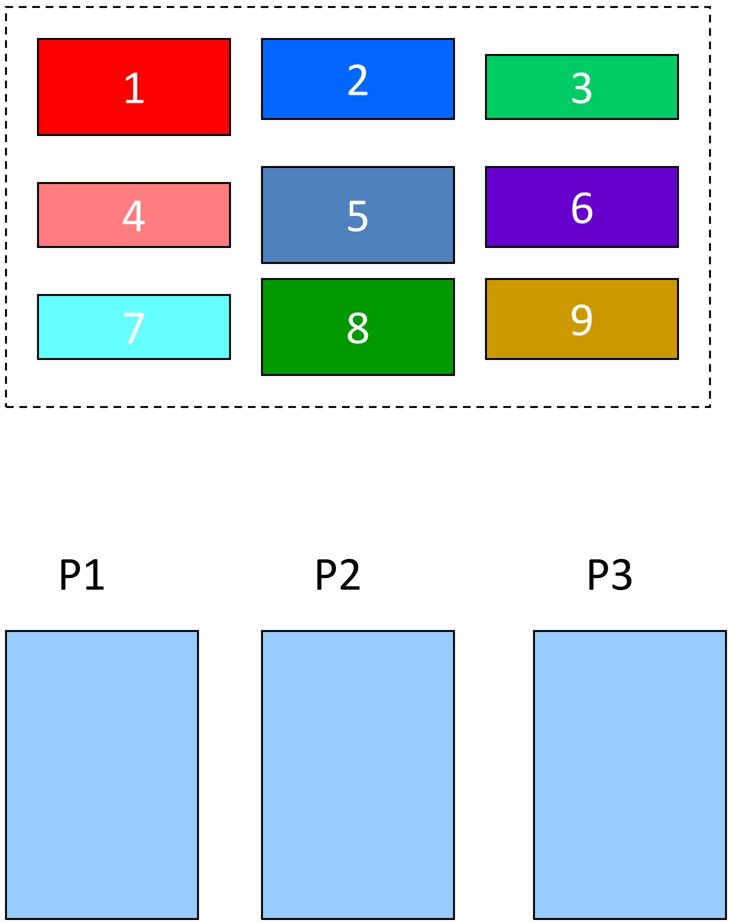
when $\varepsilon \rightarrow 0$ and $M \rightarrow +\infty$

Partition-Based Scheduling with Task-Splitting



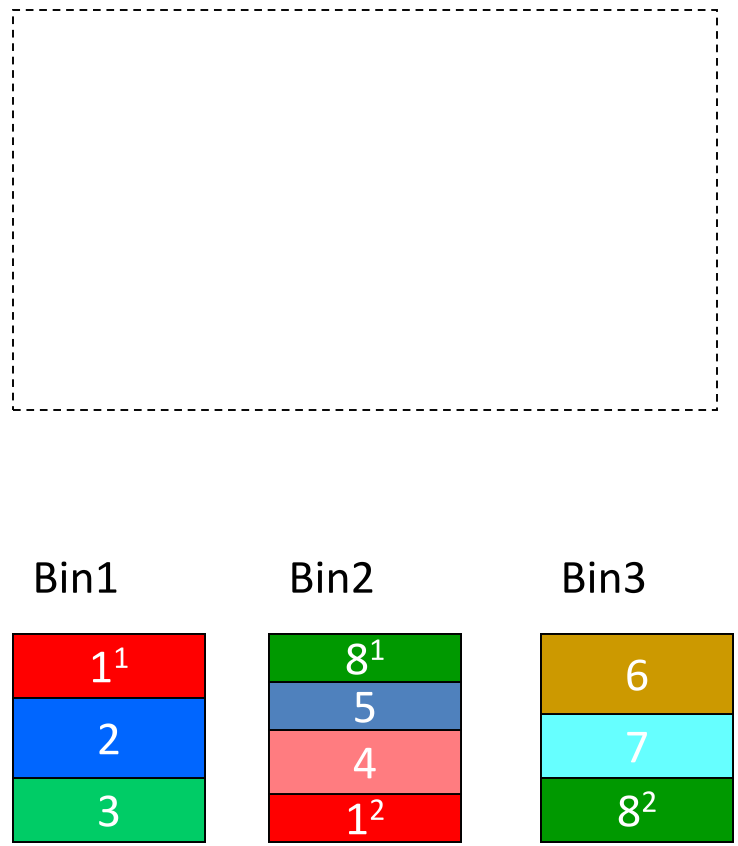
Partitioned Scheduling

- Partitioning



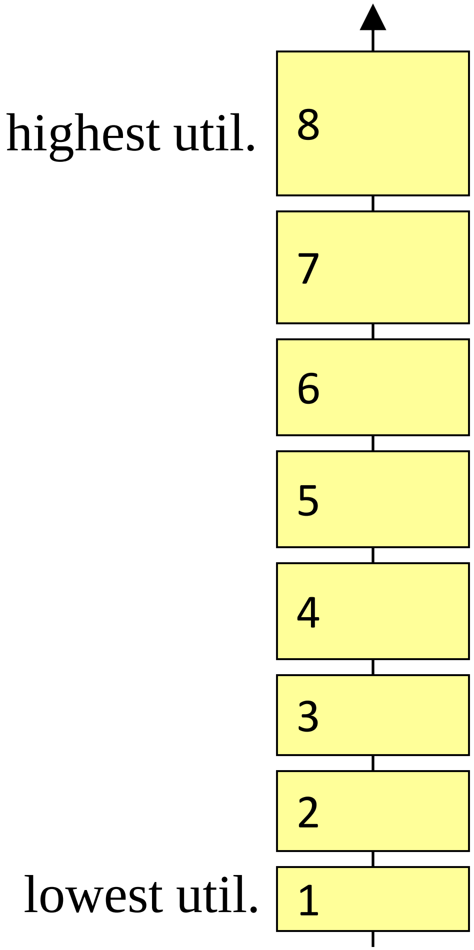
Bin-Packing with Item Splitting

- Resource can be “fully” (better) utilized



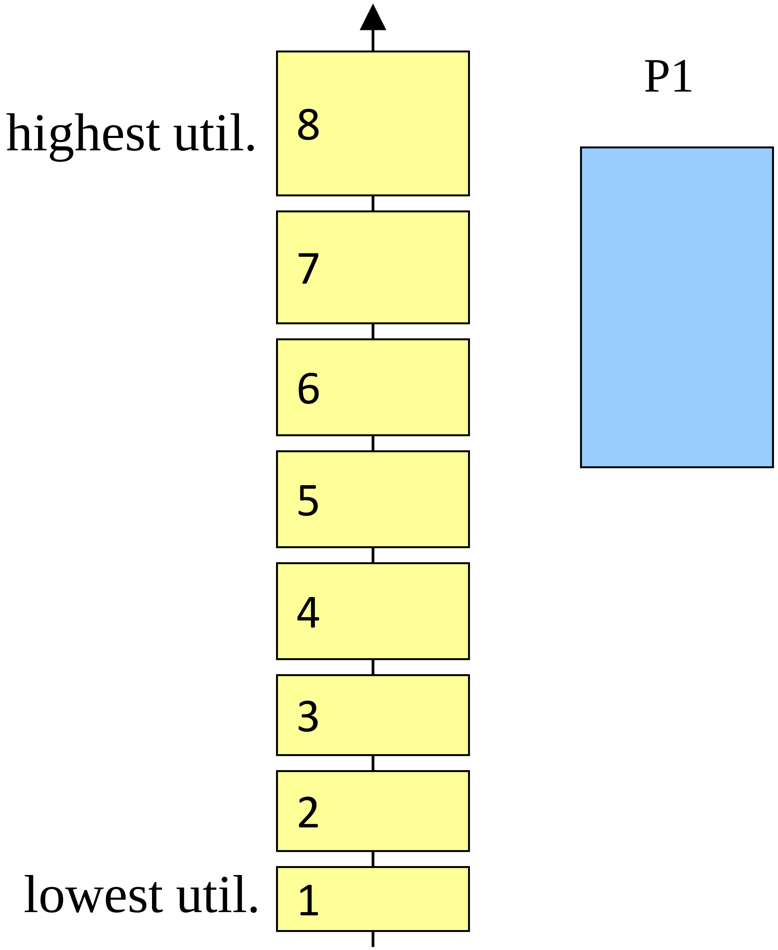
Lakshmanan's Algorithm [ECRTS'09]

- Sort all tasks in decreasing order of utilization



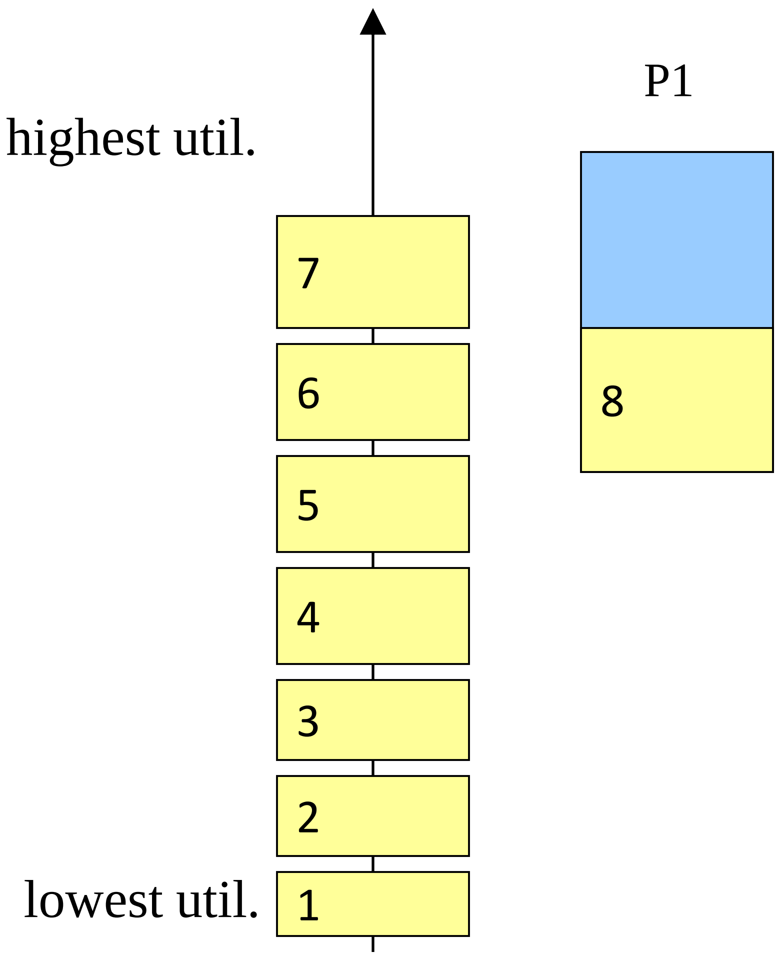
Lakshmanan's Algorithm [ECRTS'09]

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



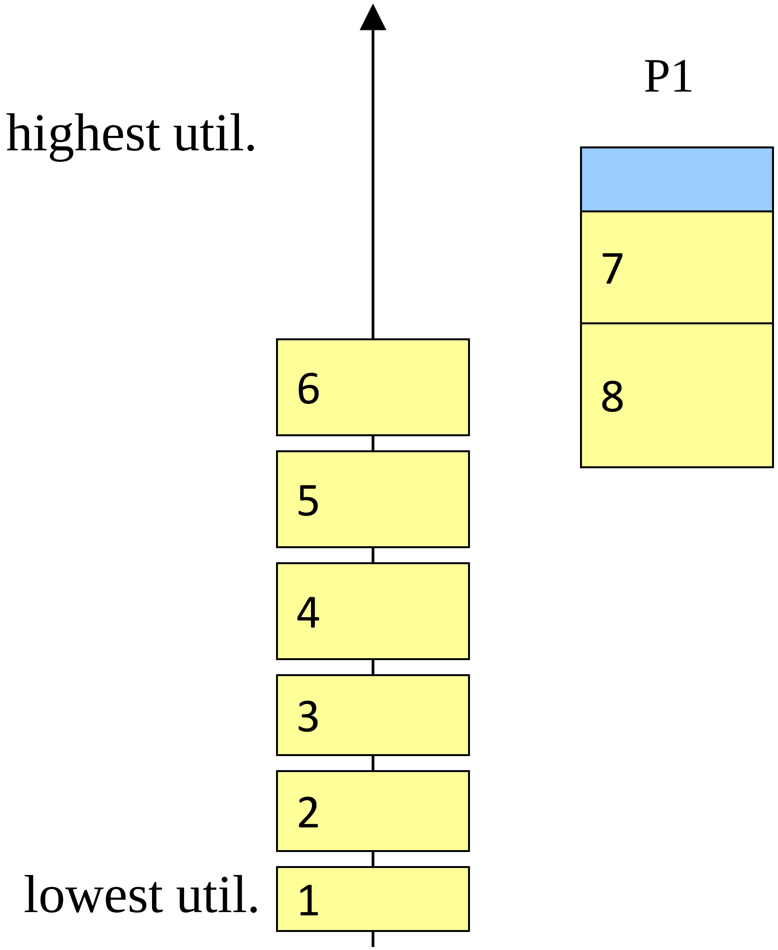
Lakshmanan's Algorithm [ECRTS'09]

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



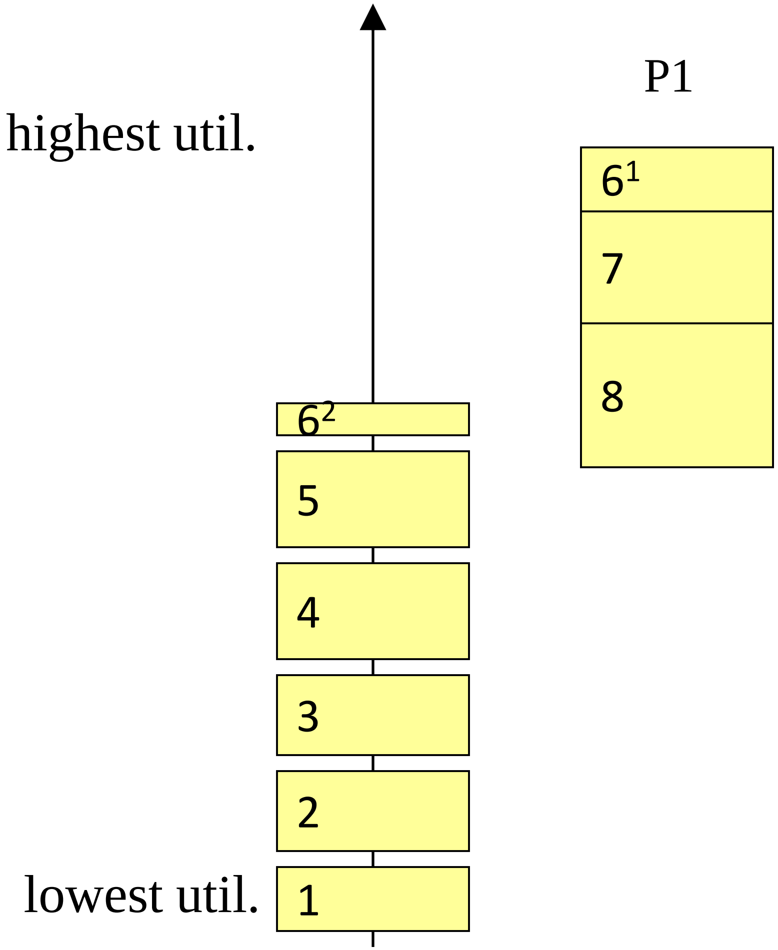
Lakshmanan's Algorithm [ECRTS'09]

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



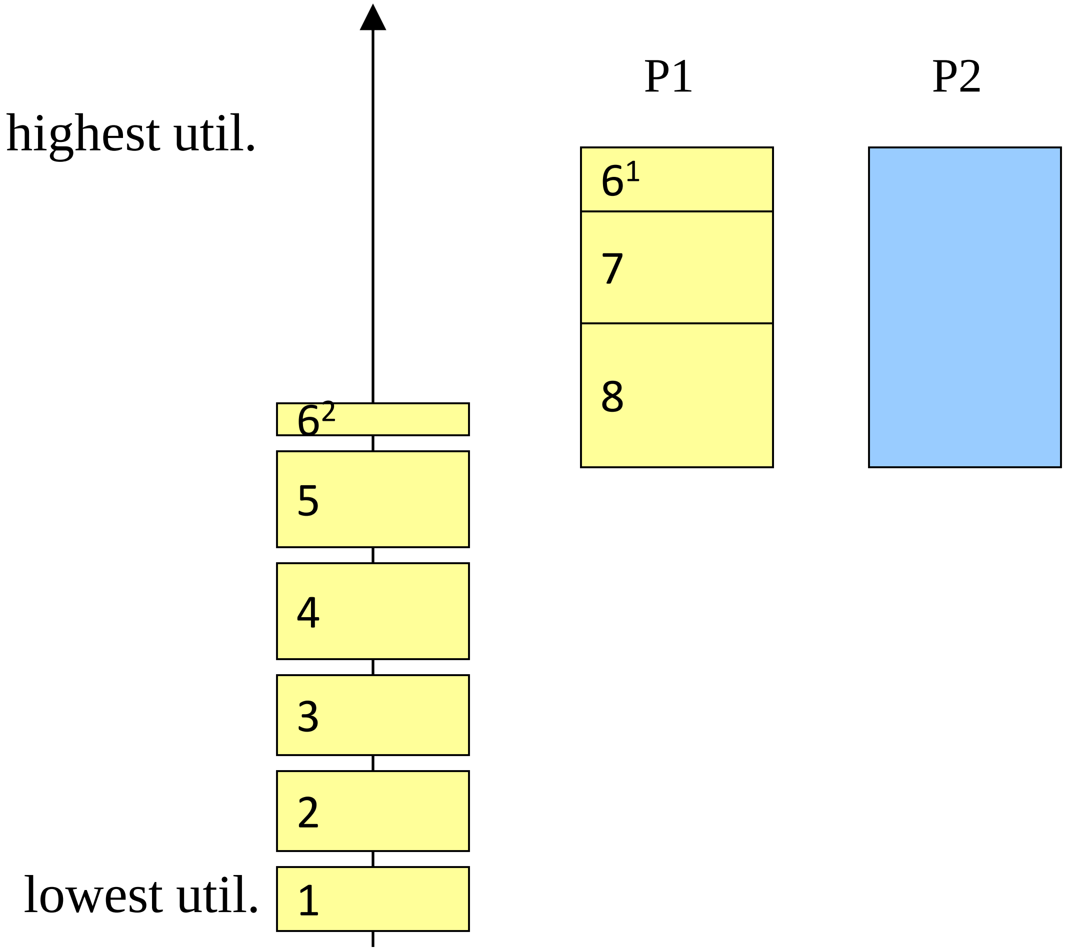
Lakshmanan's Algorithm [ECRTS'09]

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



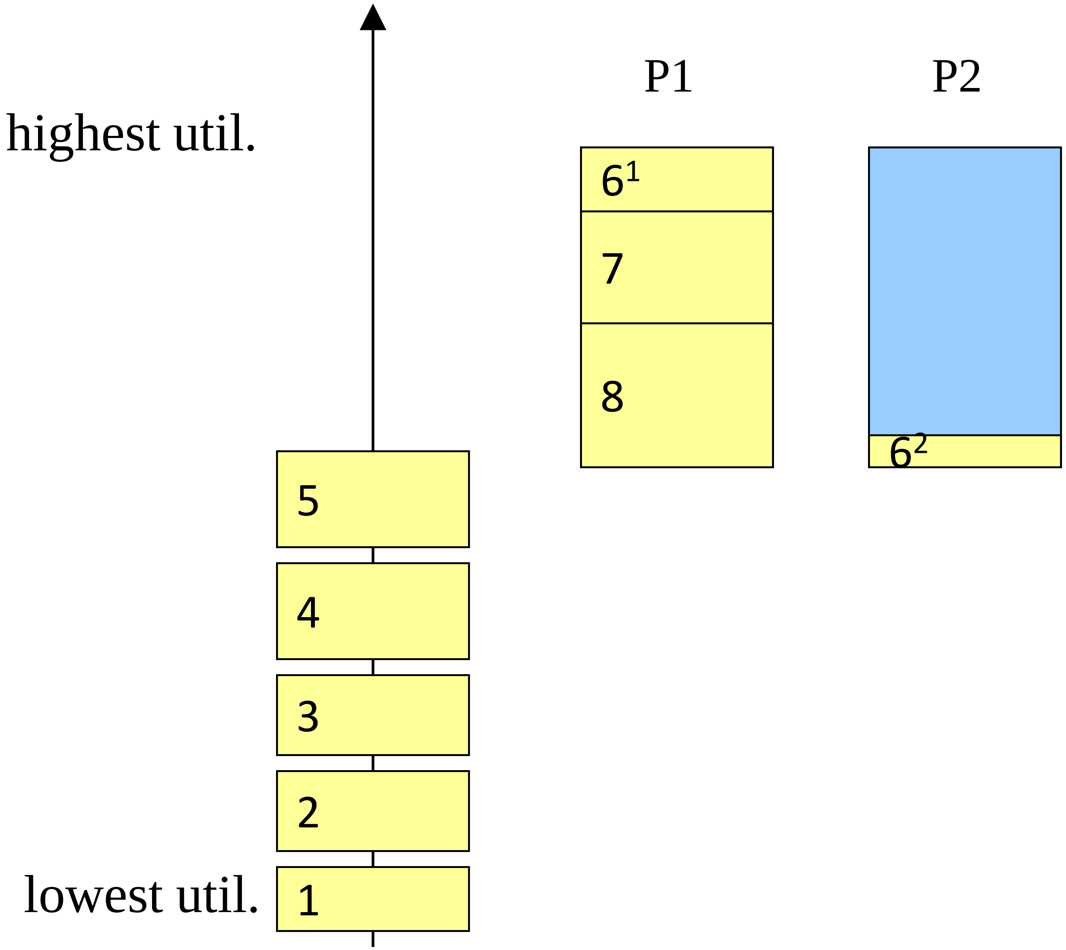
Lakshmanan's Algorithm [ECRTS'09]

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



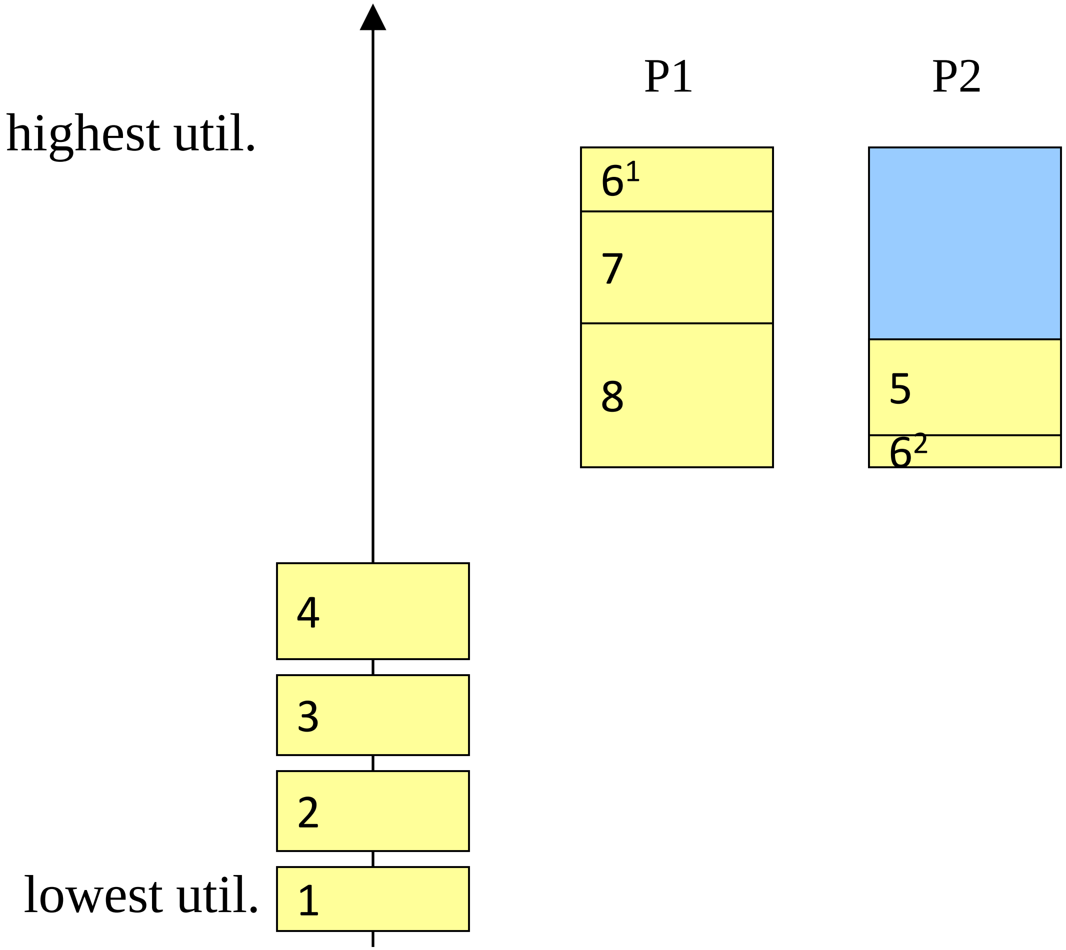
Lakshmanan's Algorithm [ECRTS'09]

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



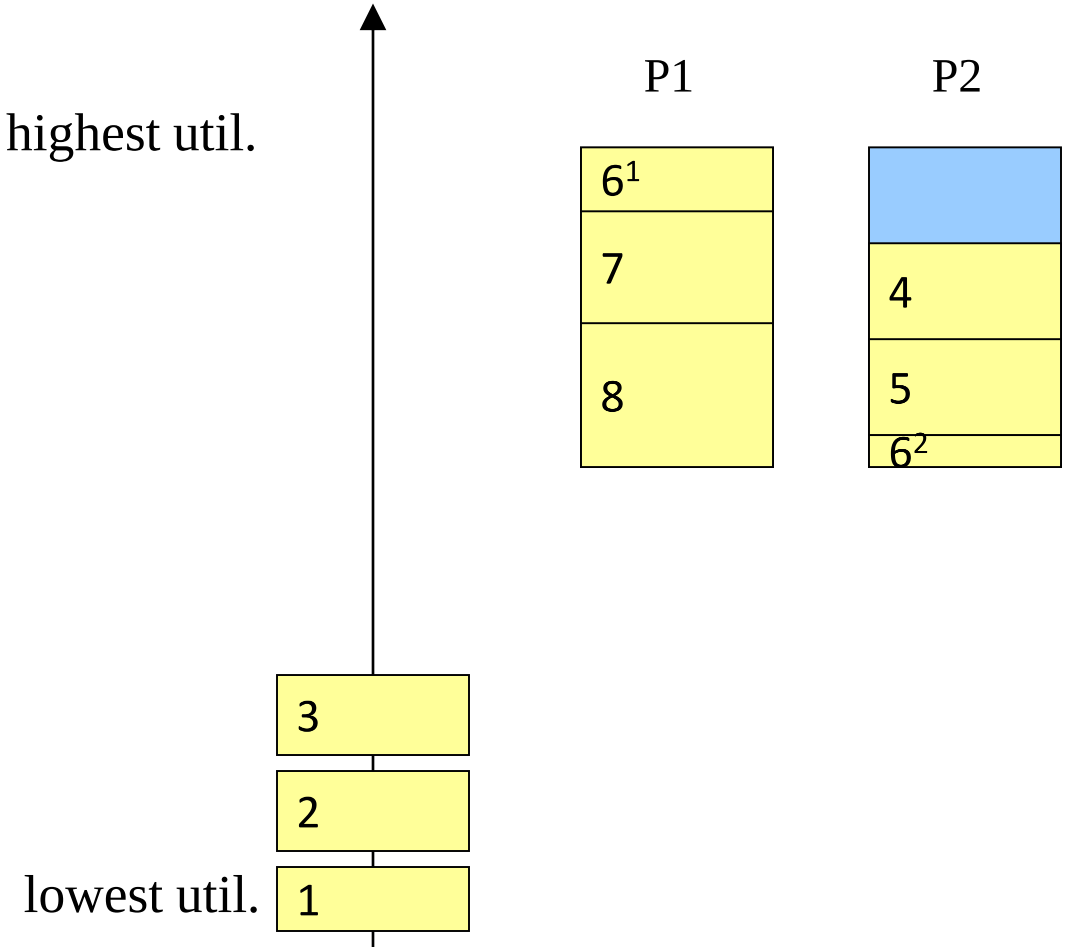
Lakshmanan's Algorithm [ECRTS'09]

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



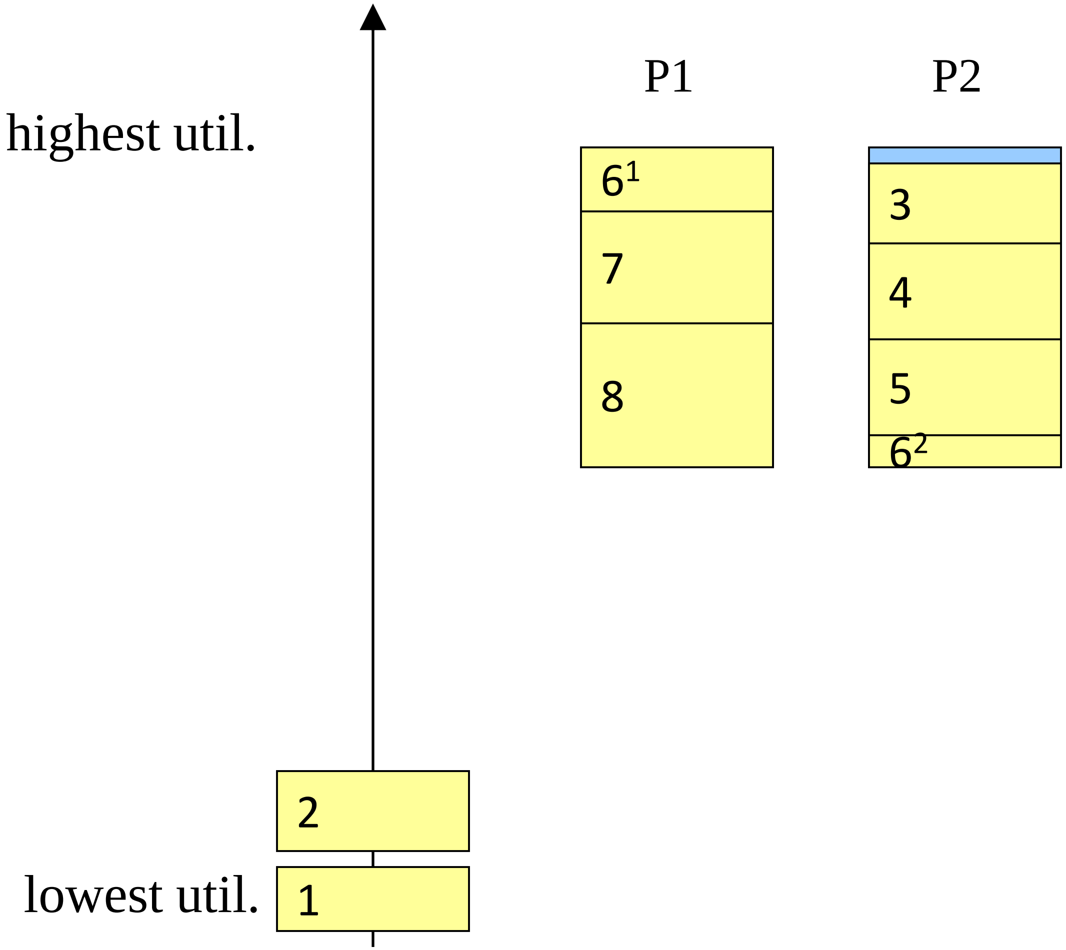
Lakshmanan's Algorithm [ECRTS'09]

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



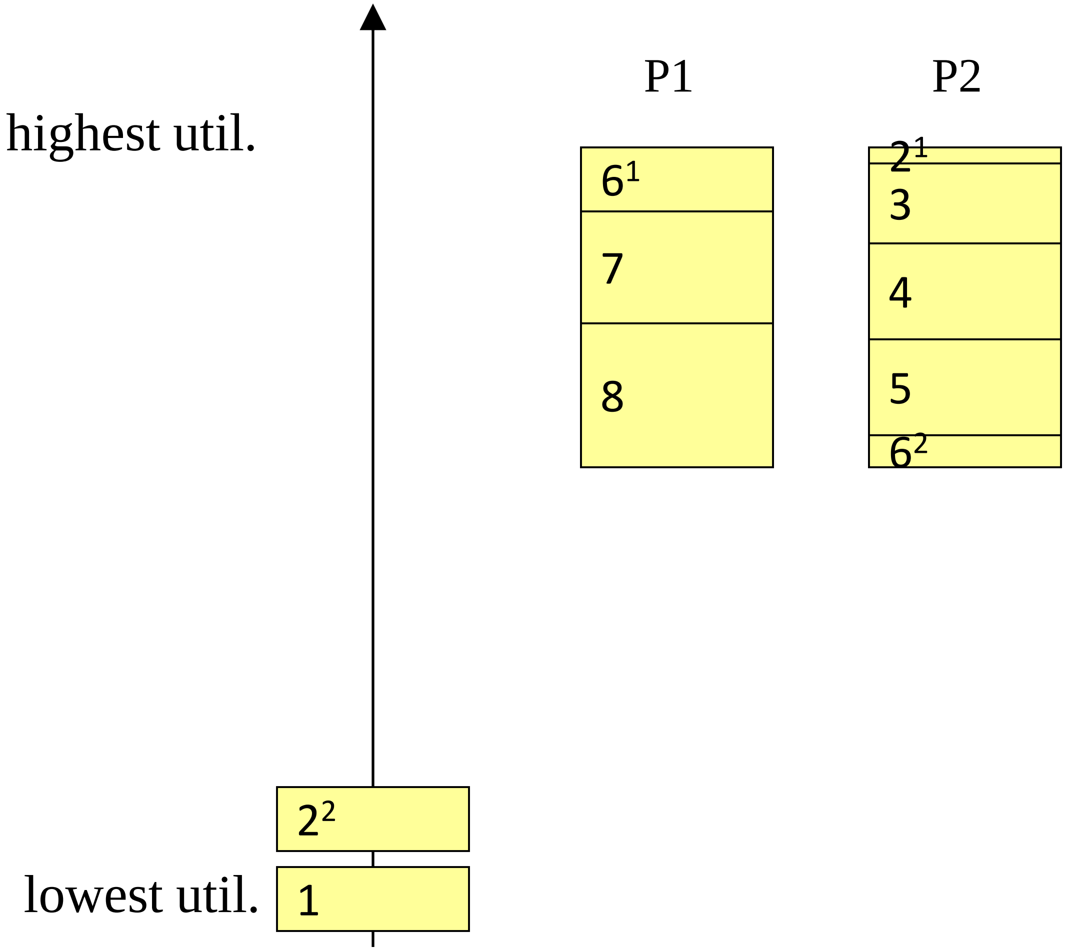
Lakshmanan's Algorithm [ECRTS'09]

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



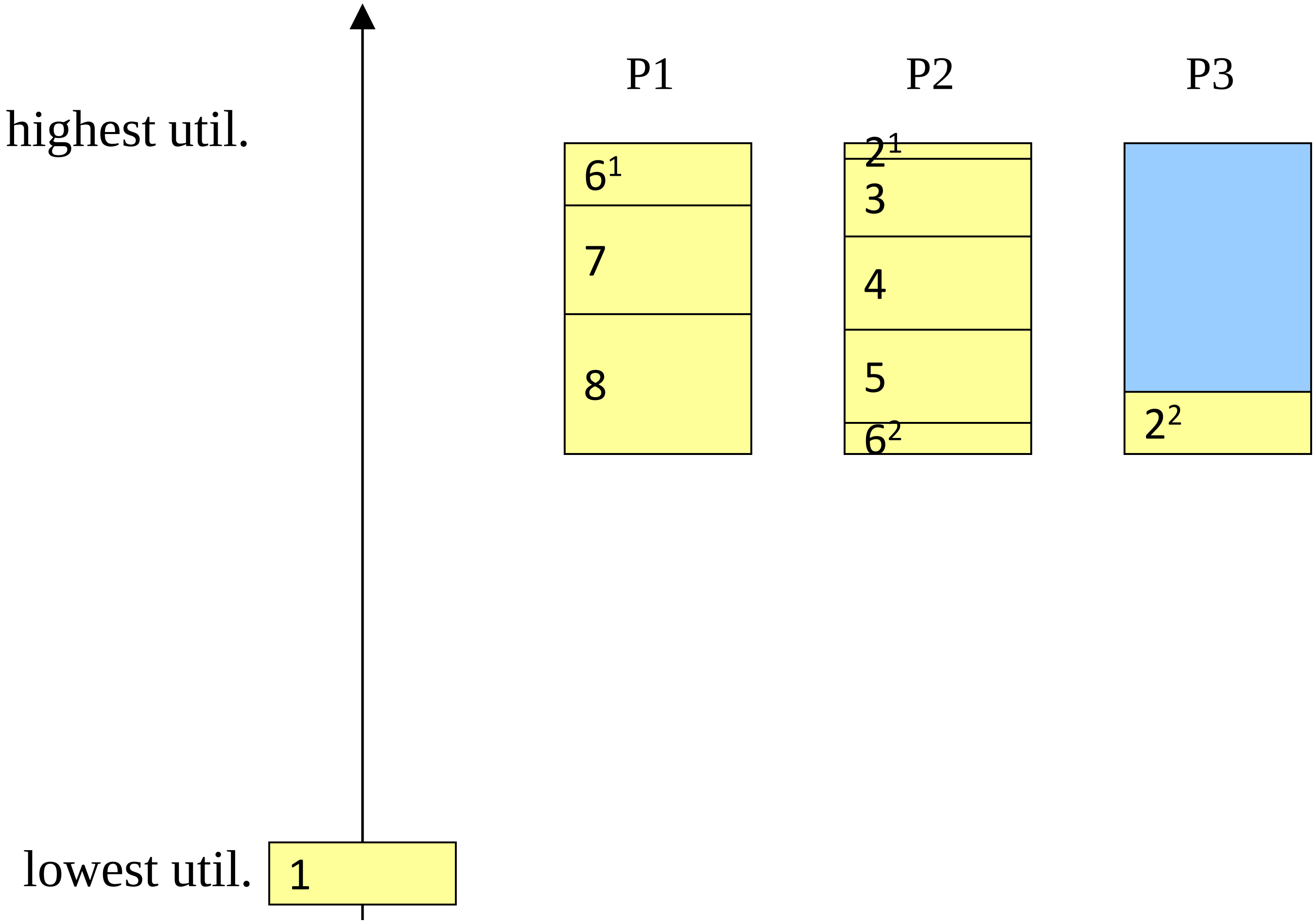
Lakshmanan's Algorithm [ECRTS'09]

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



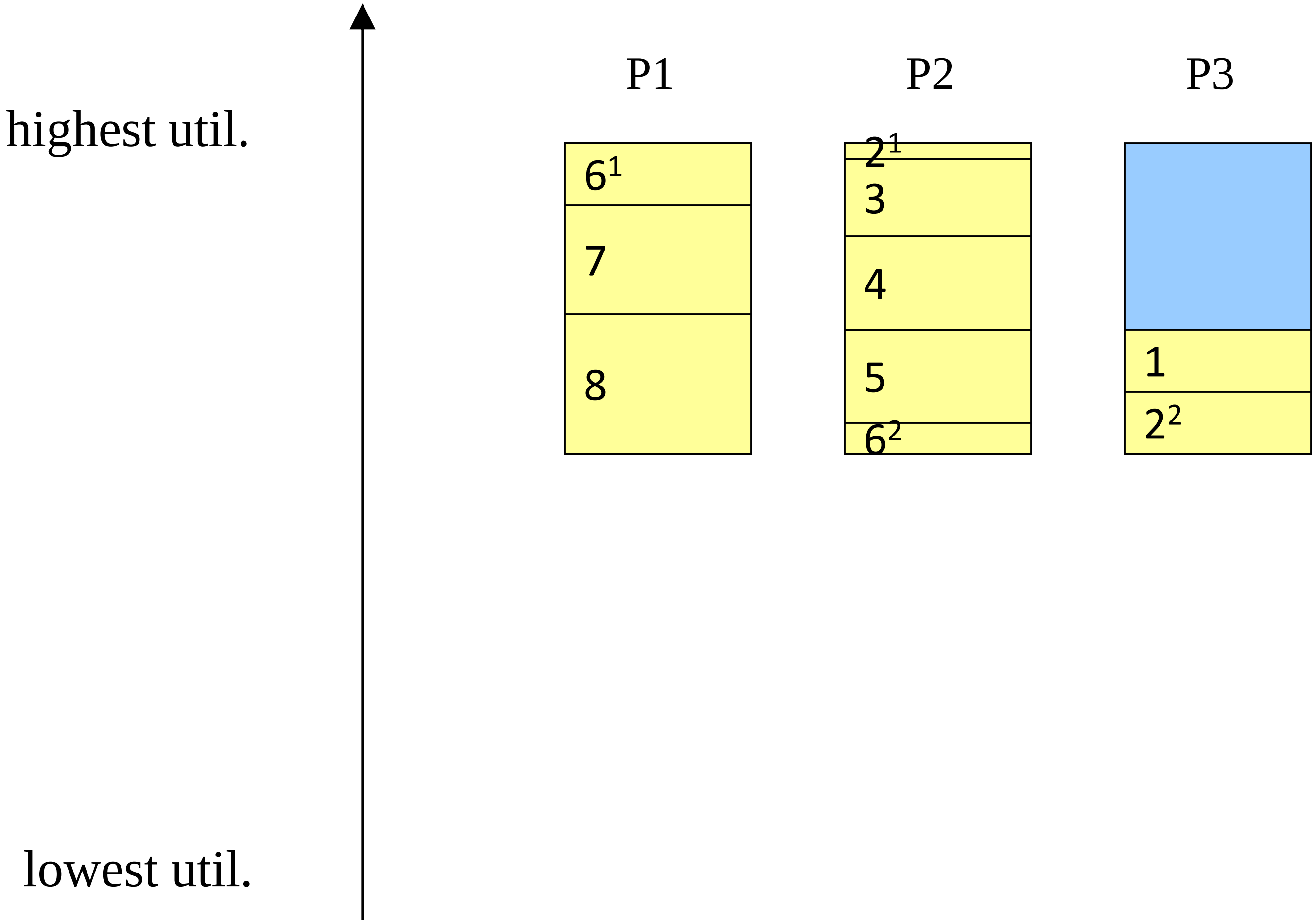
Lakshmanan's Algorithm [ECRTS'09]

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



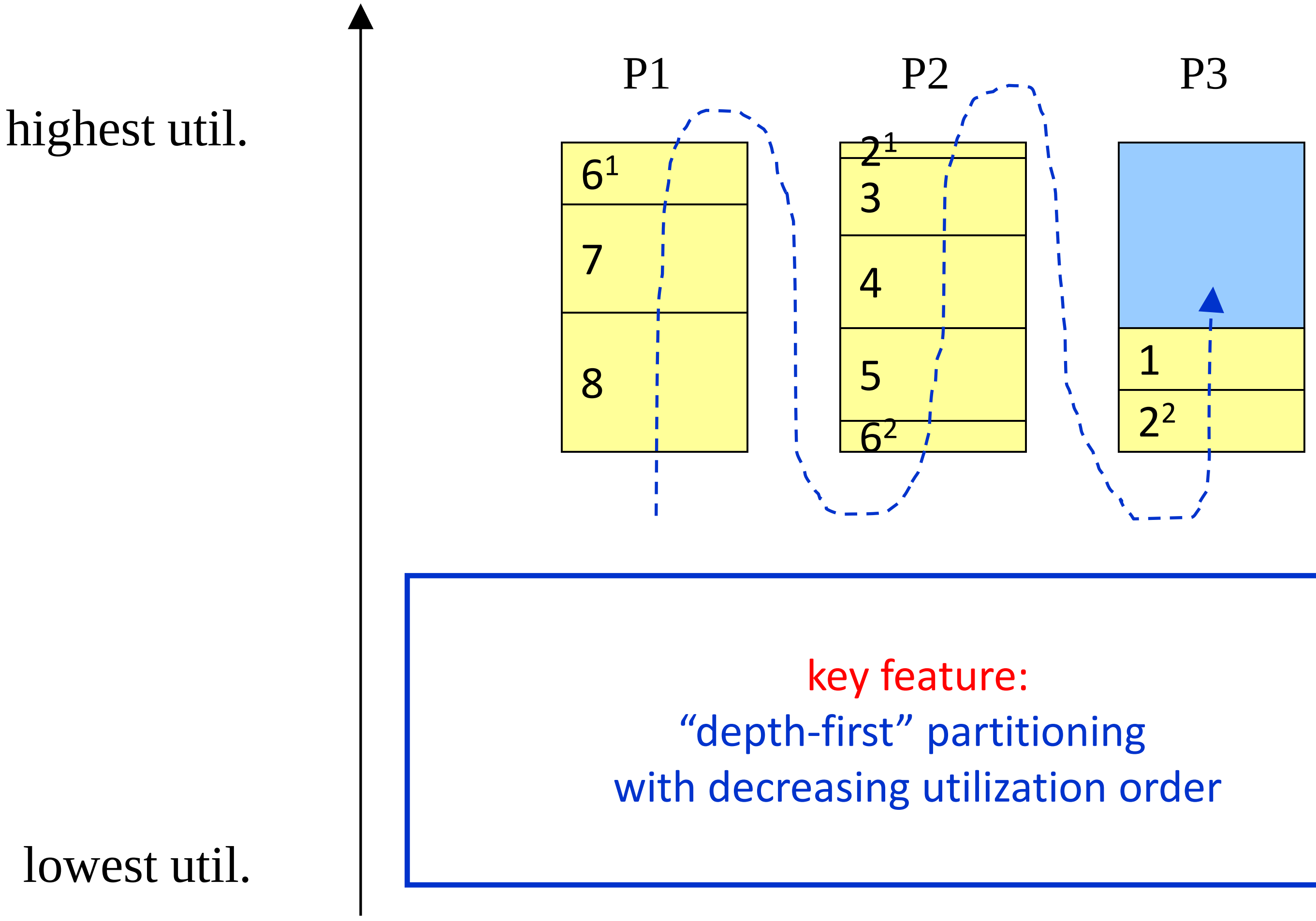
Lakshmanan's Algorithm [ECRTS'09]

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



Lakshmanan's Algorithm [ECRTS'09]

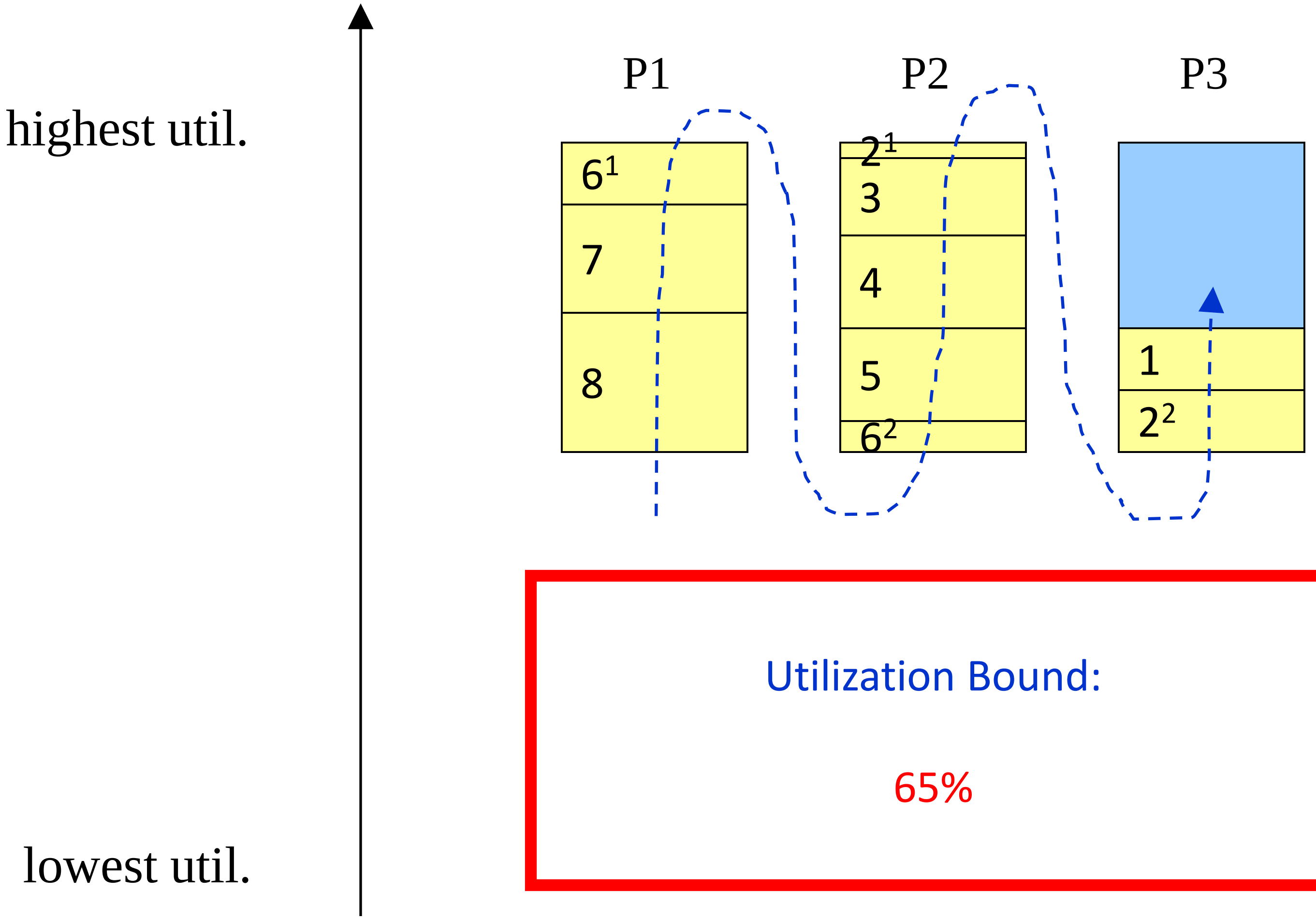
- Pick up one processor, and assign as many tasks as possible



key feature:
"depth-first" partitioning
with decreasing utilization order

Lakshmanan's Algorithm [ECRTS'09]

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



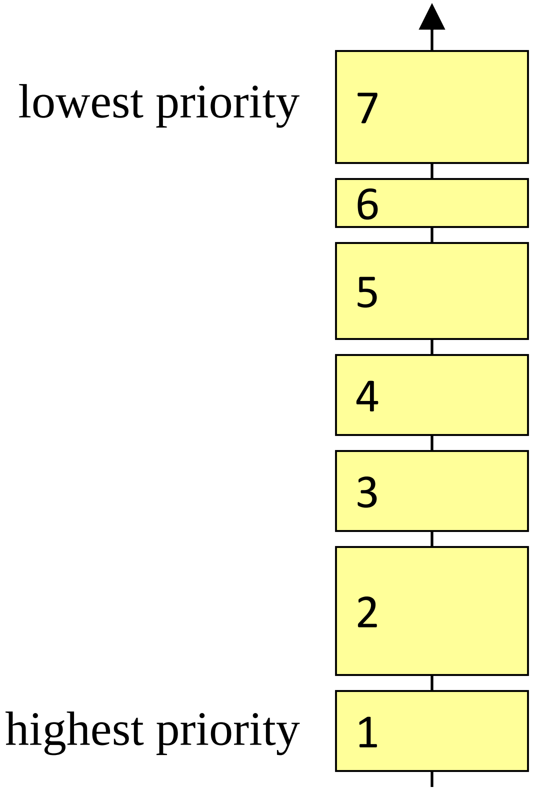
Utilization Bound:

65%

Breadth-First Partitioning Algorithms

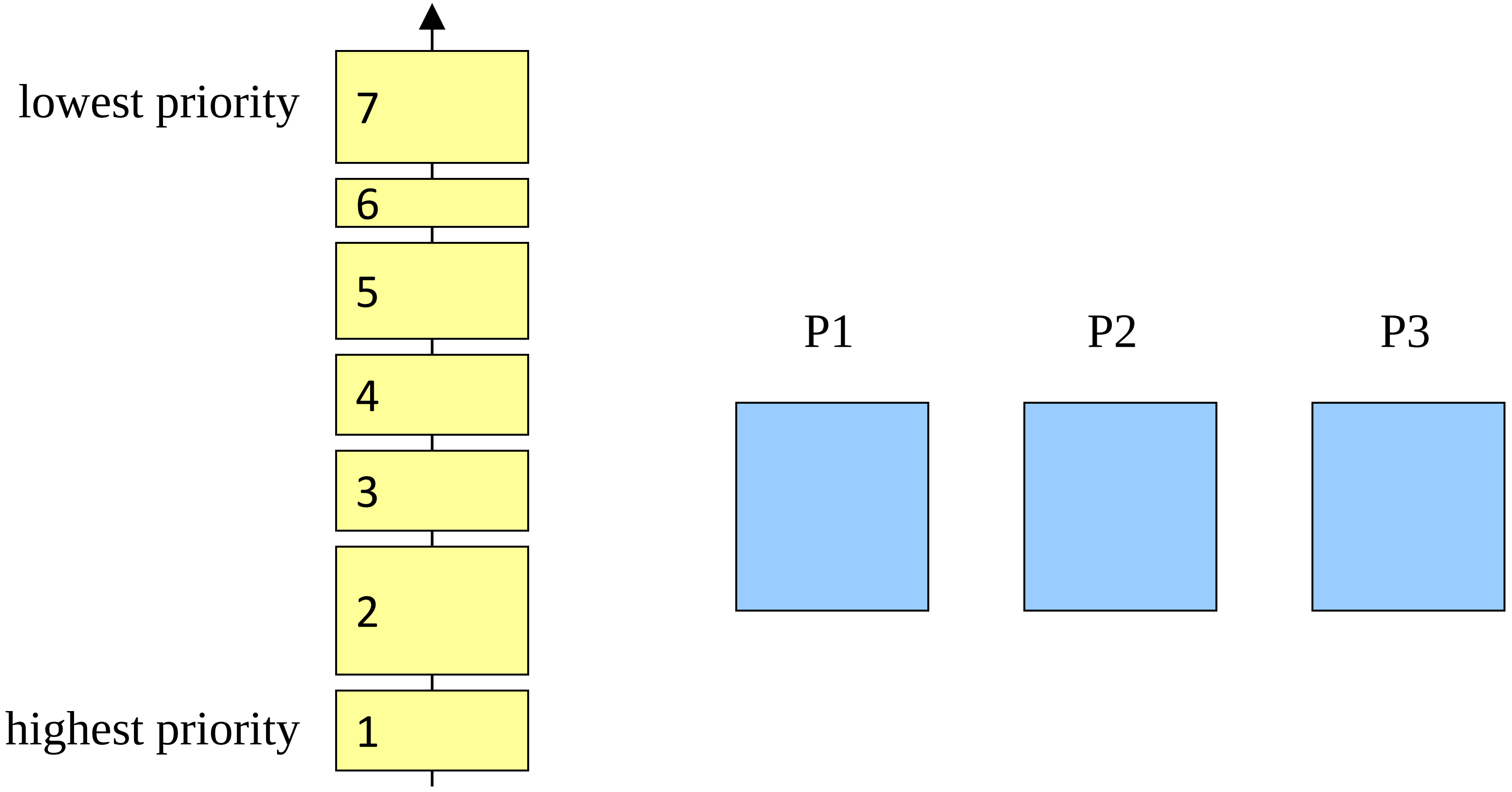
[RTAS 2010]

- Sort all tasks in increasing priority order



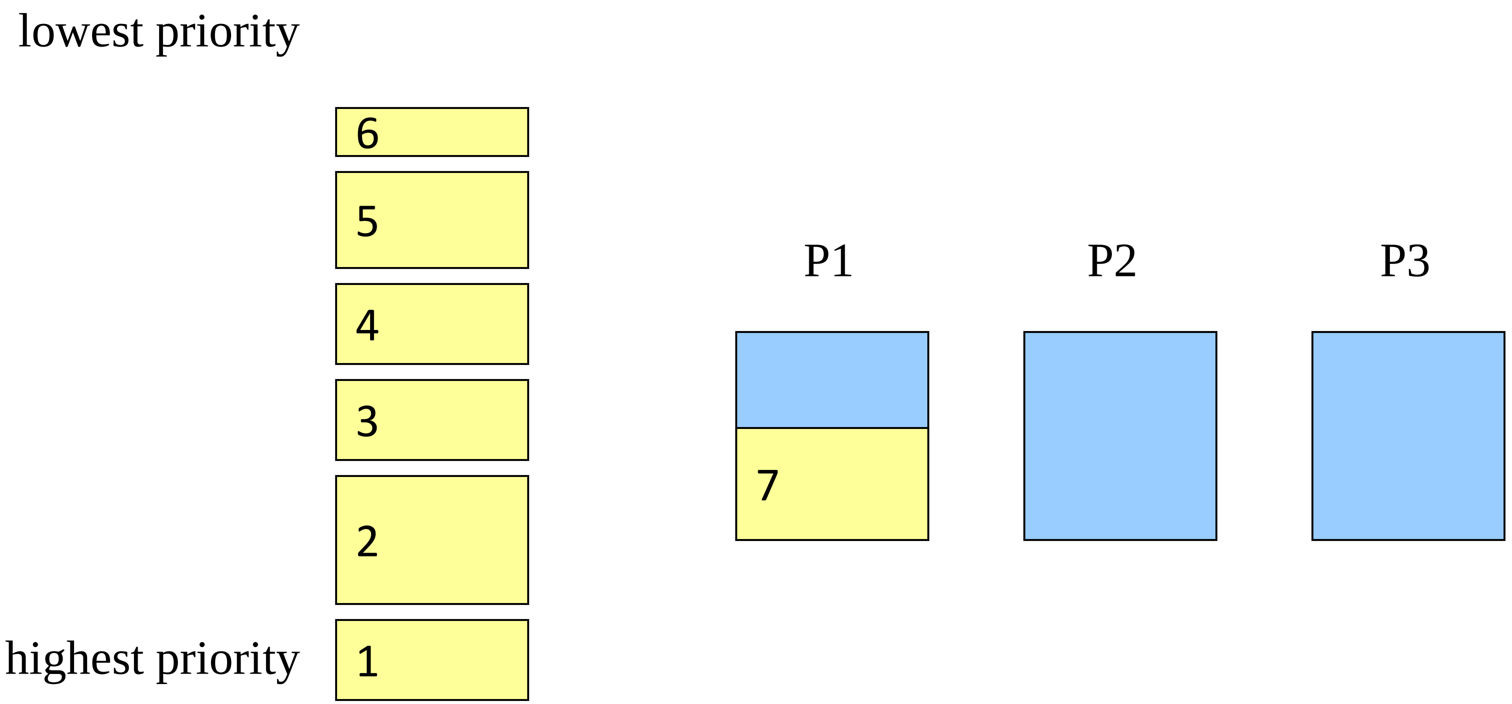
Breadth-First Partitioning Algorithms

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



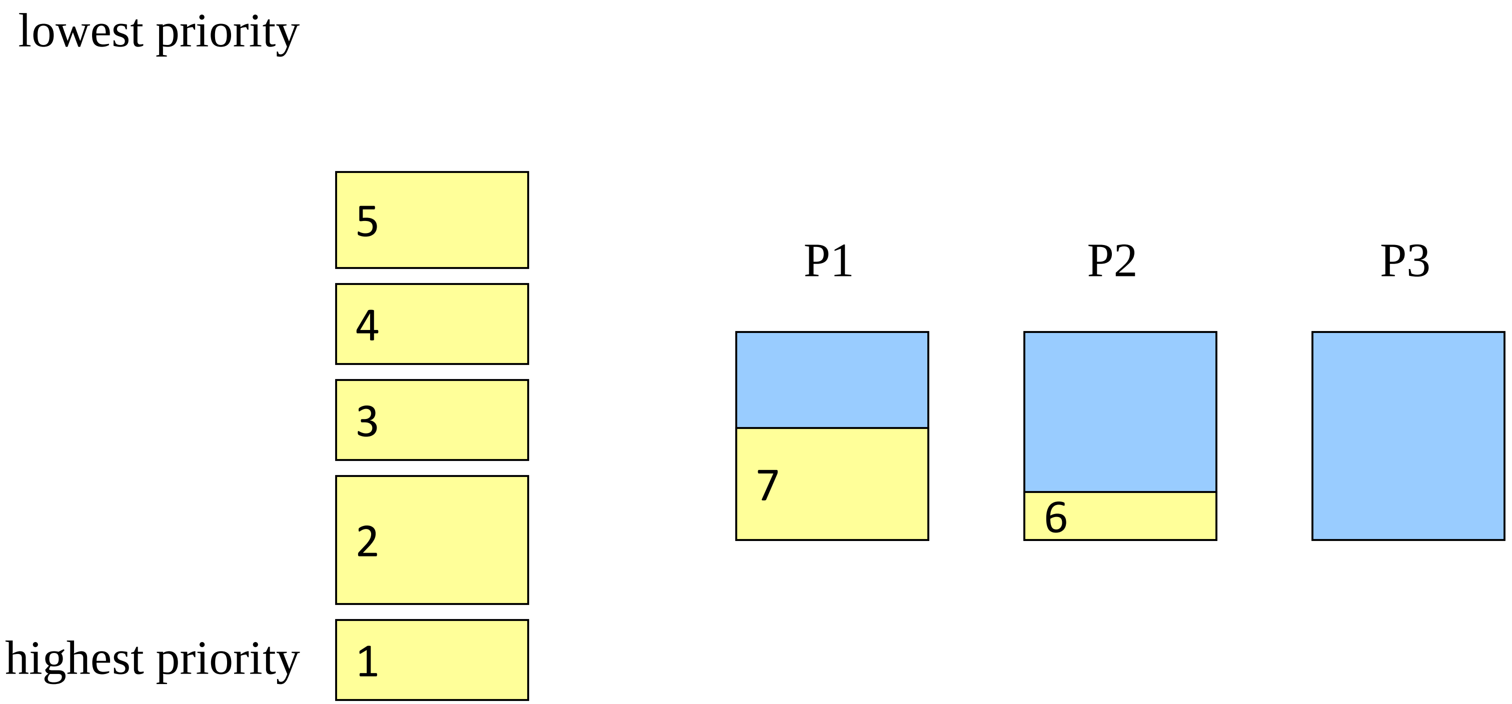
Breadth-First Partitioning Algorithms

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



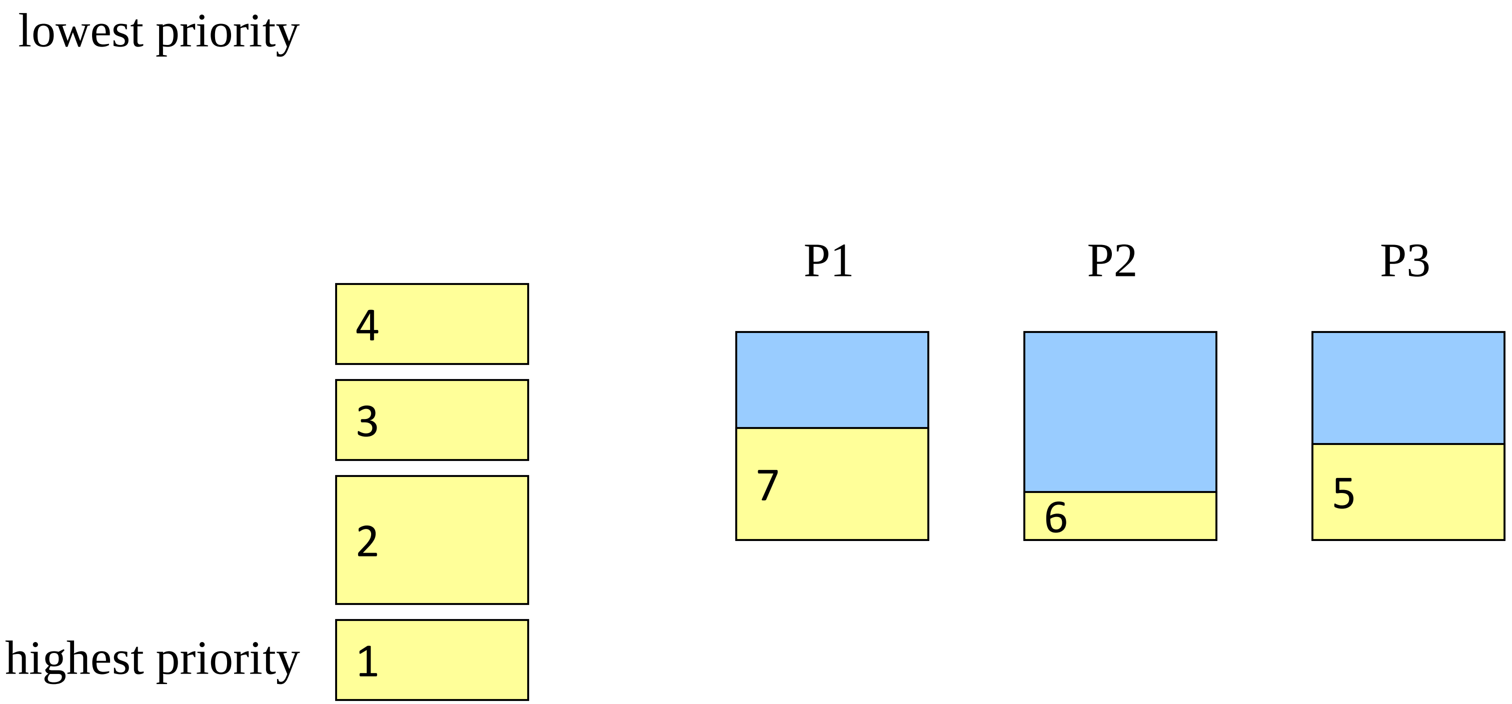
Breadth-First Partitioning Algorithms

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



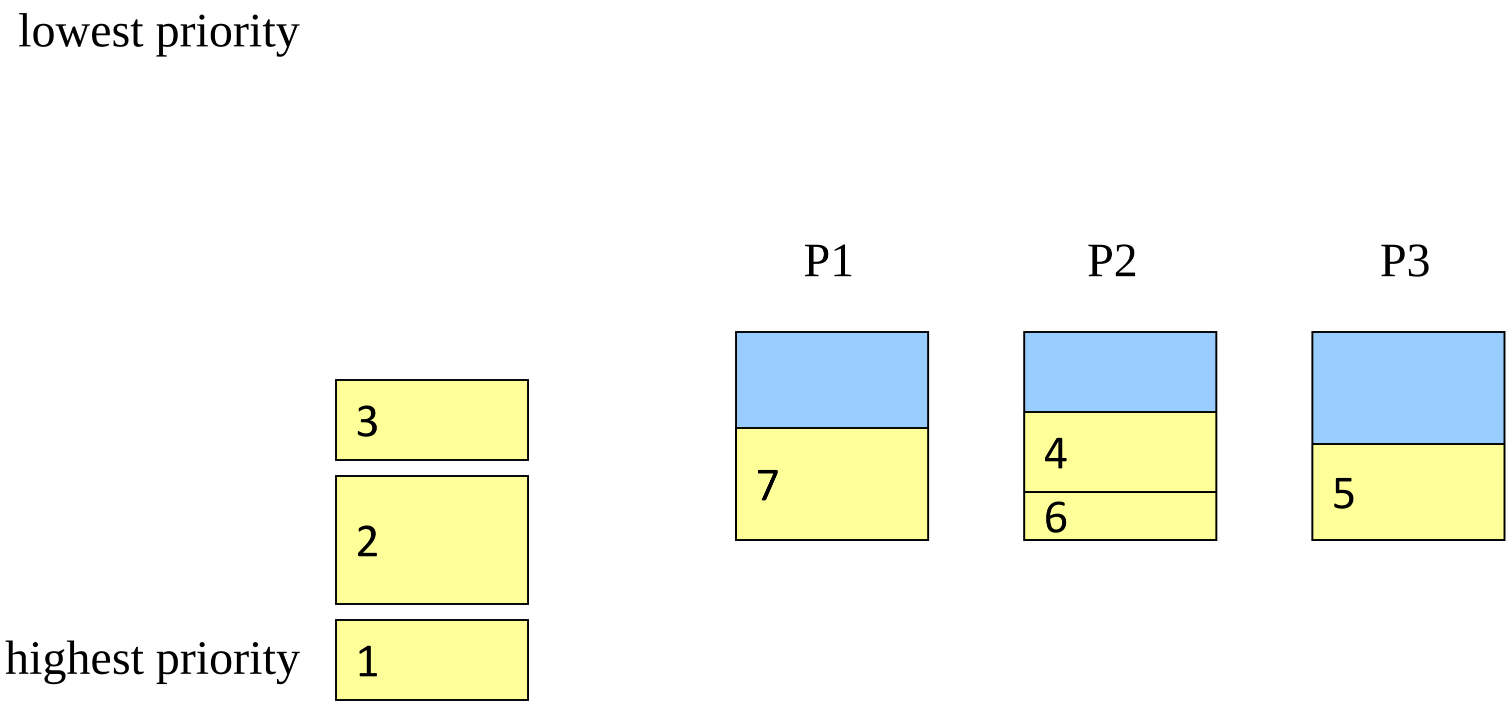
Breadth-First Partitioning Algorithms

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



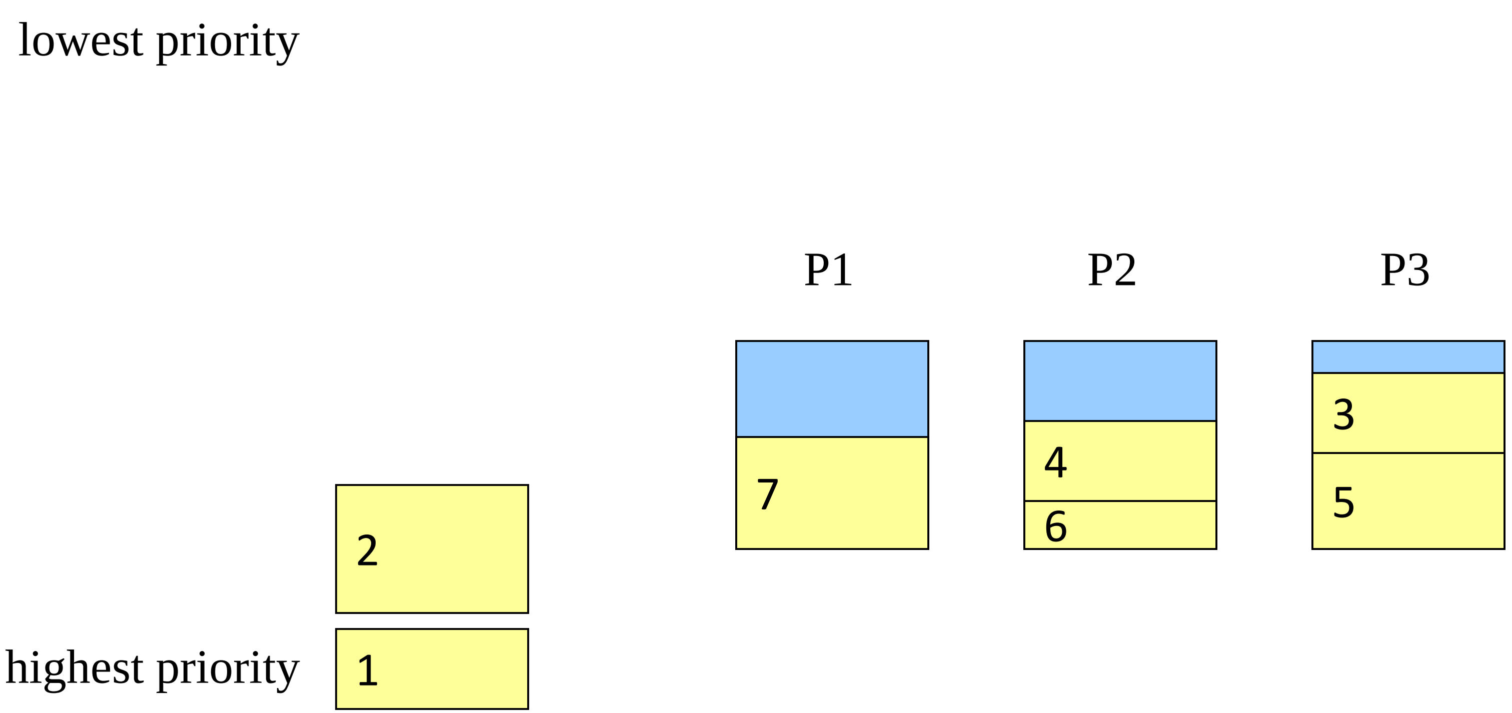
Breadth-First Partitioning Algorithms

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



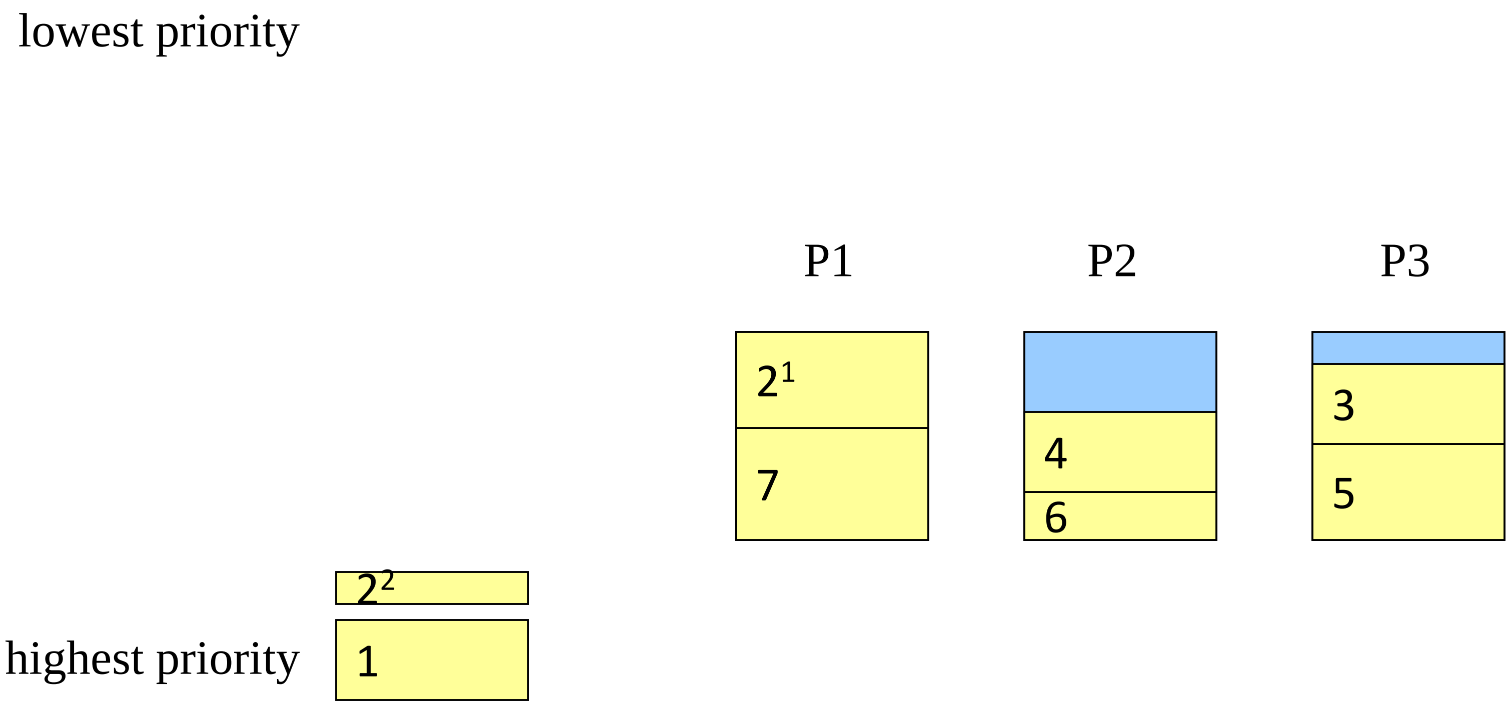
Breadth-First Partitioning Algorithms

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



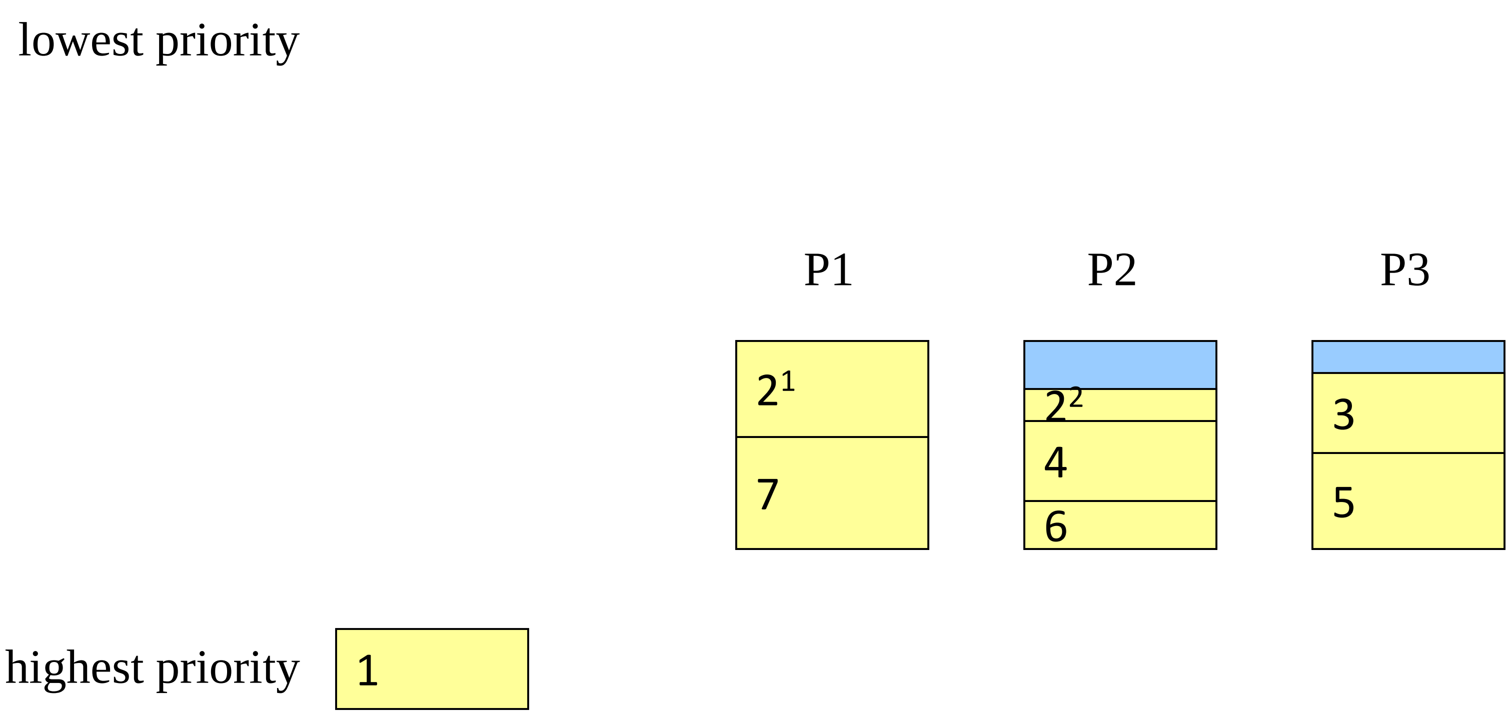
Breadth-First Partitioning Algorithms

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



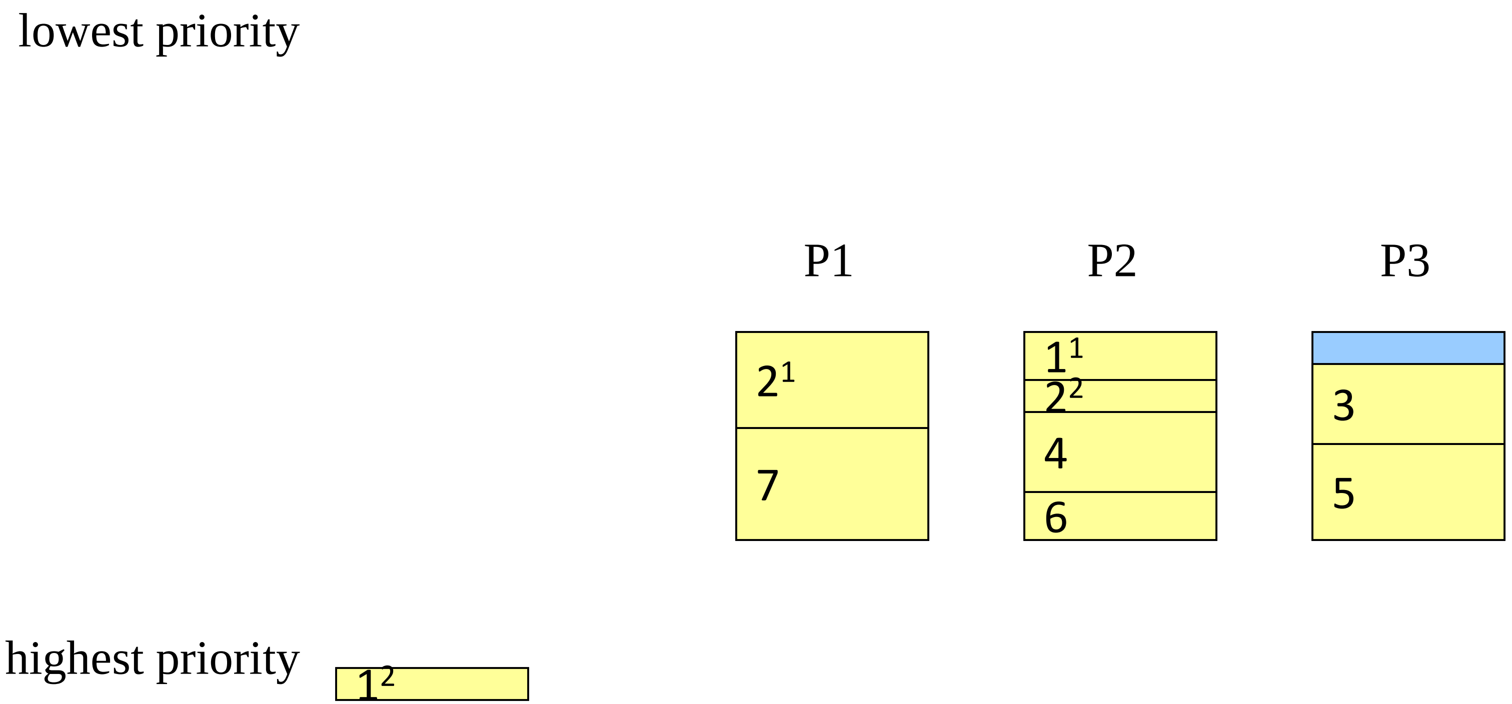
Breadth-First Partitioning Algorithms

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



Breadth-First Partitioning Algorithms

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

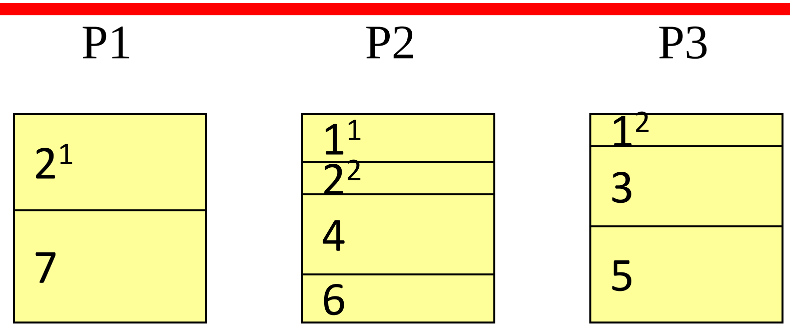


Breadth-First Partitioning Algorithms

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

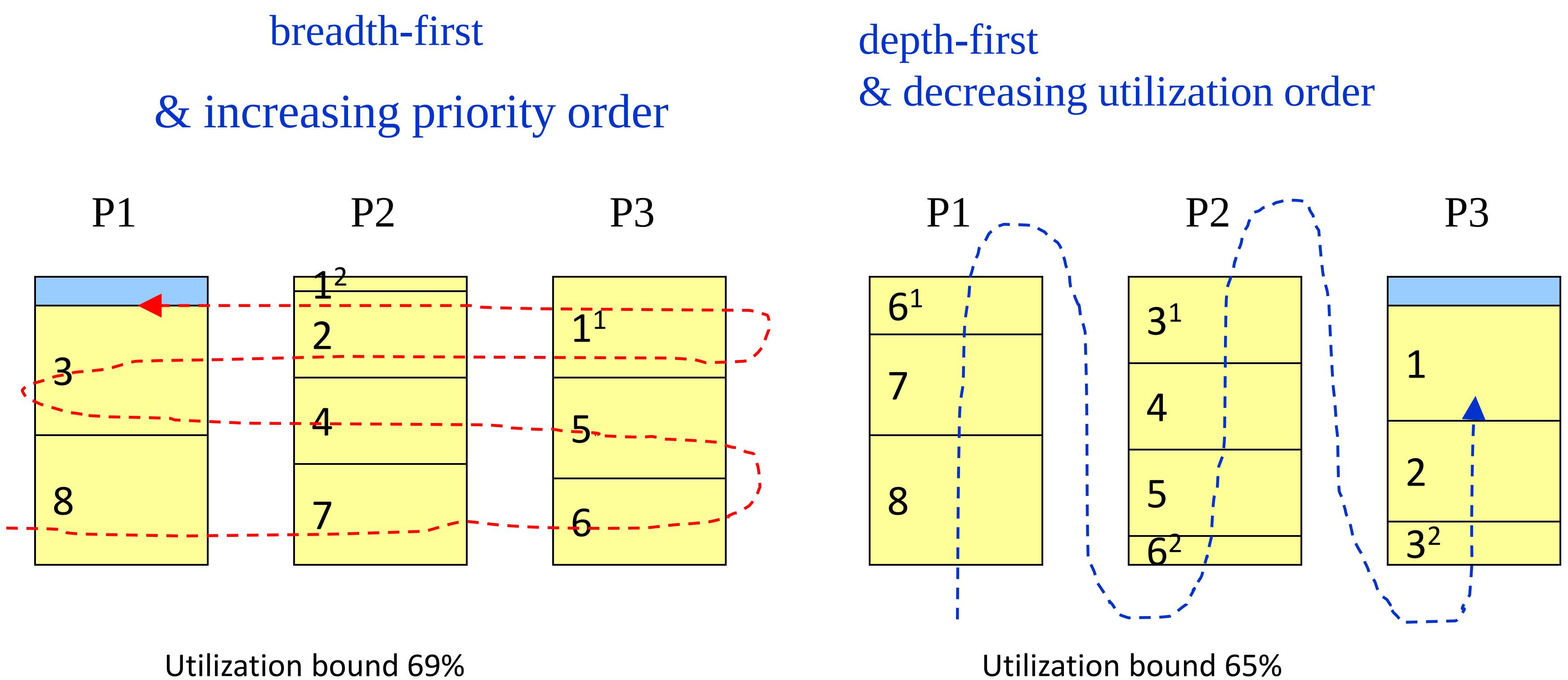
Utilization Bound:

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

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

we have

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

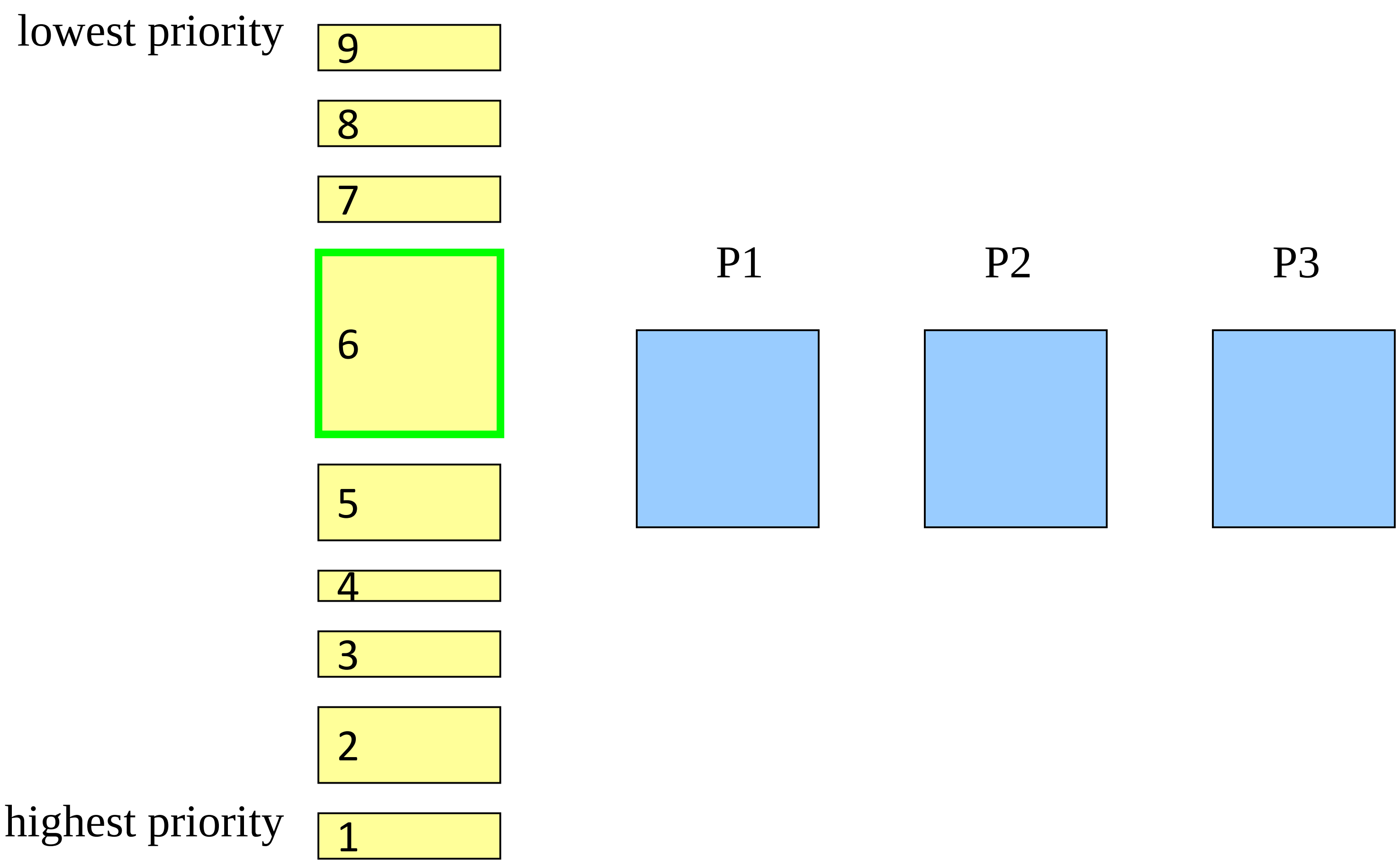
$$\Rightarrow \quad \text{the task set is schedulable}$$

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

Solution for Heavy Tasks

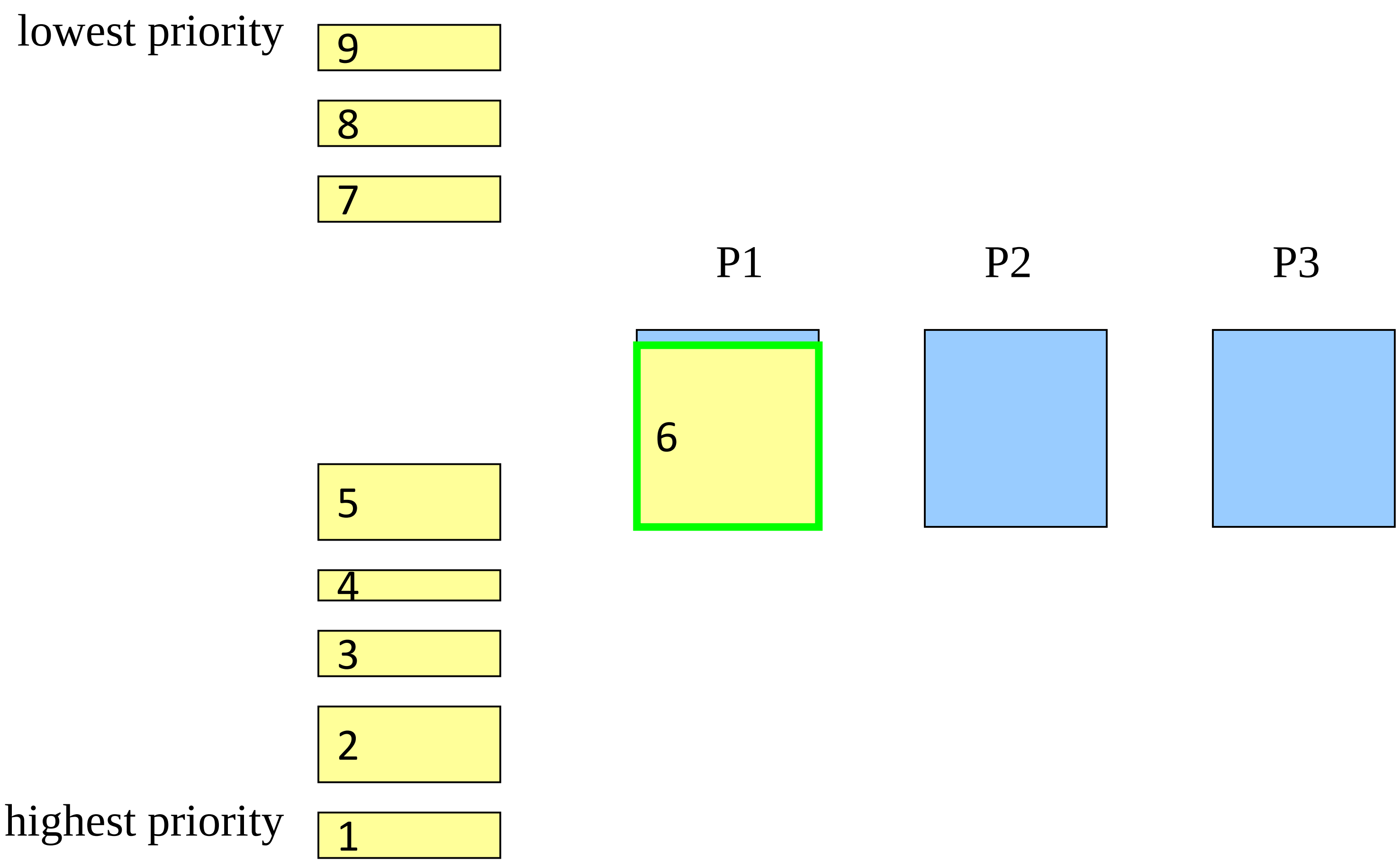
whose utilization larger than 0.41

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



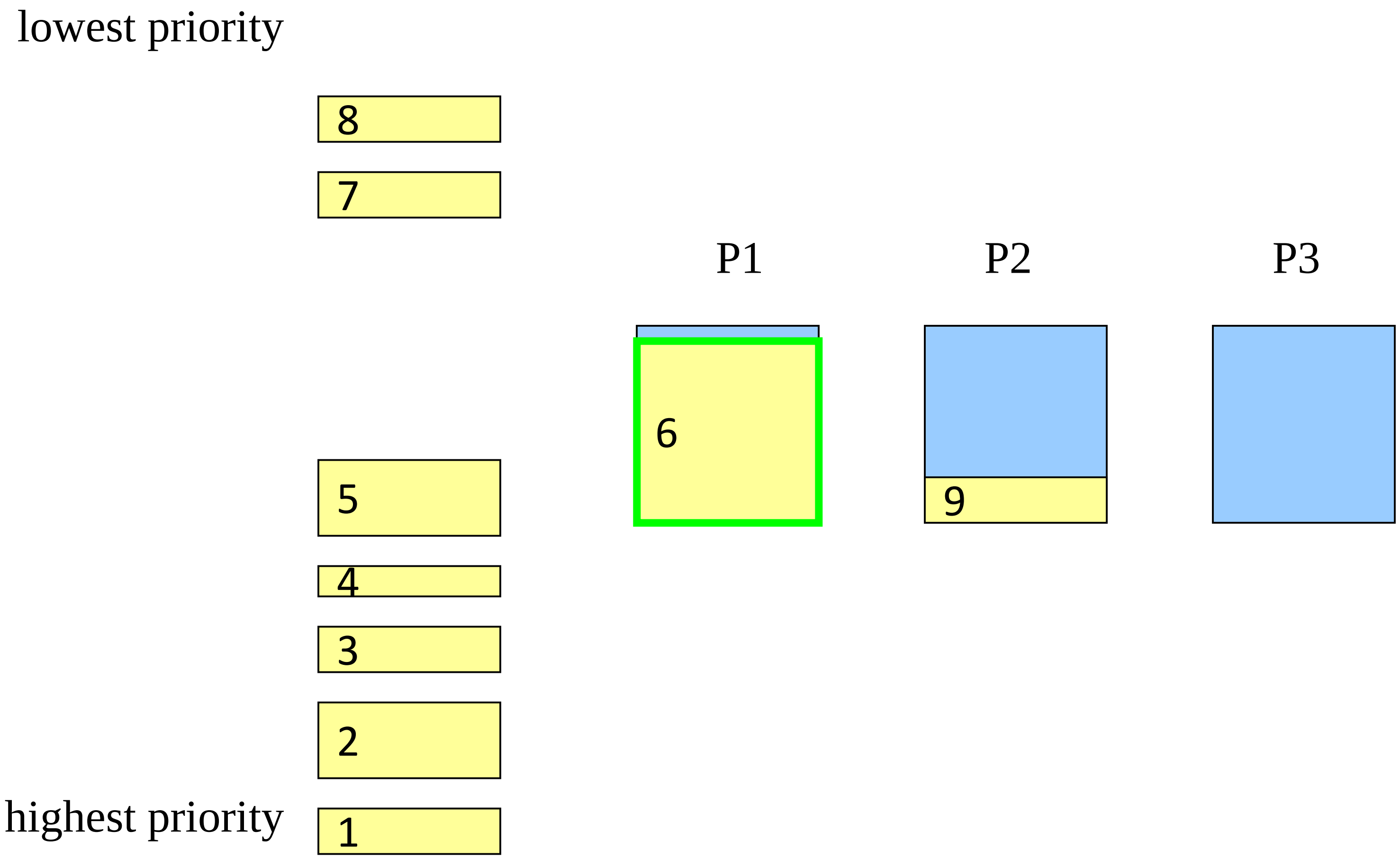
Solution for Heavy Tasks

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



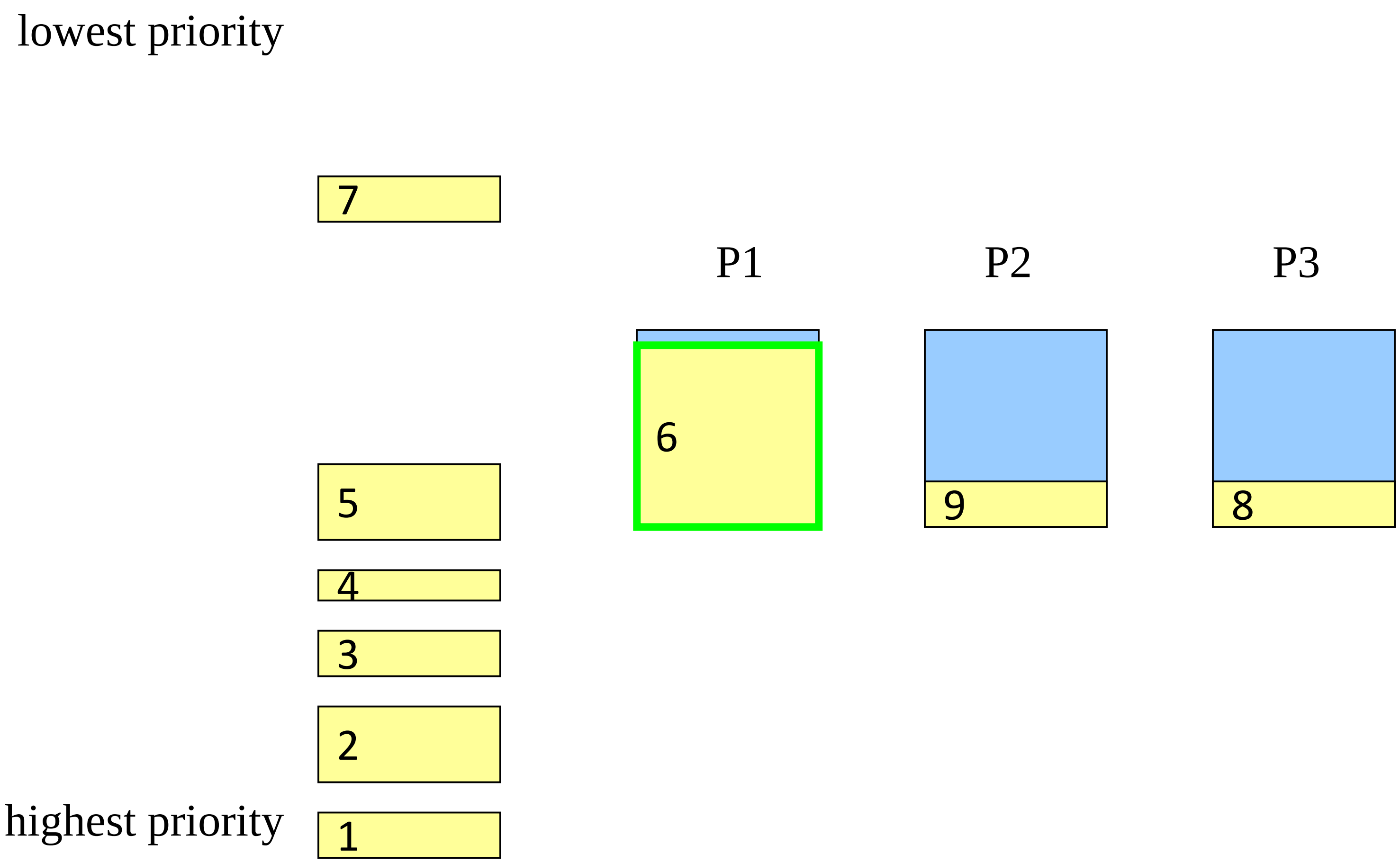
Solution for Heavy Tasks

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



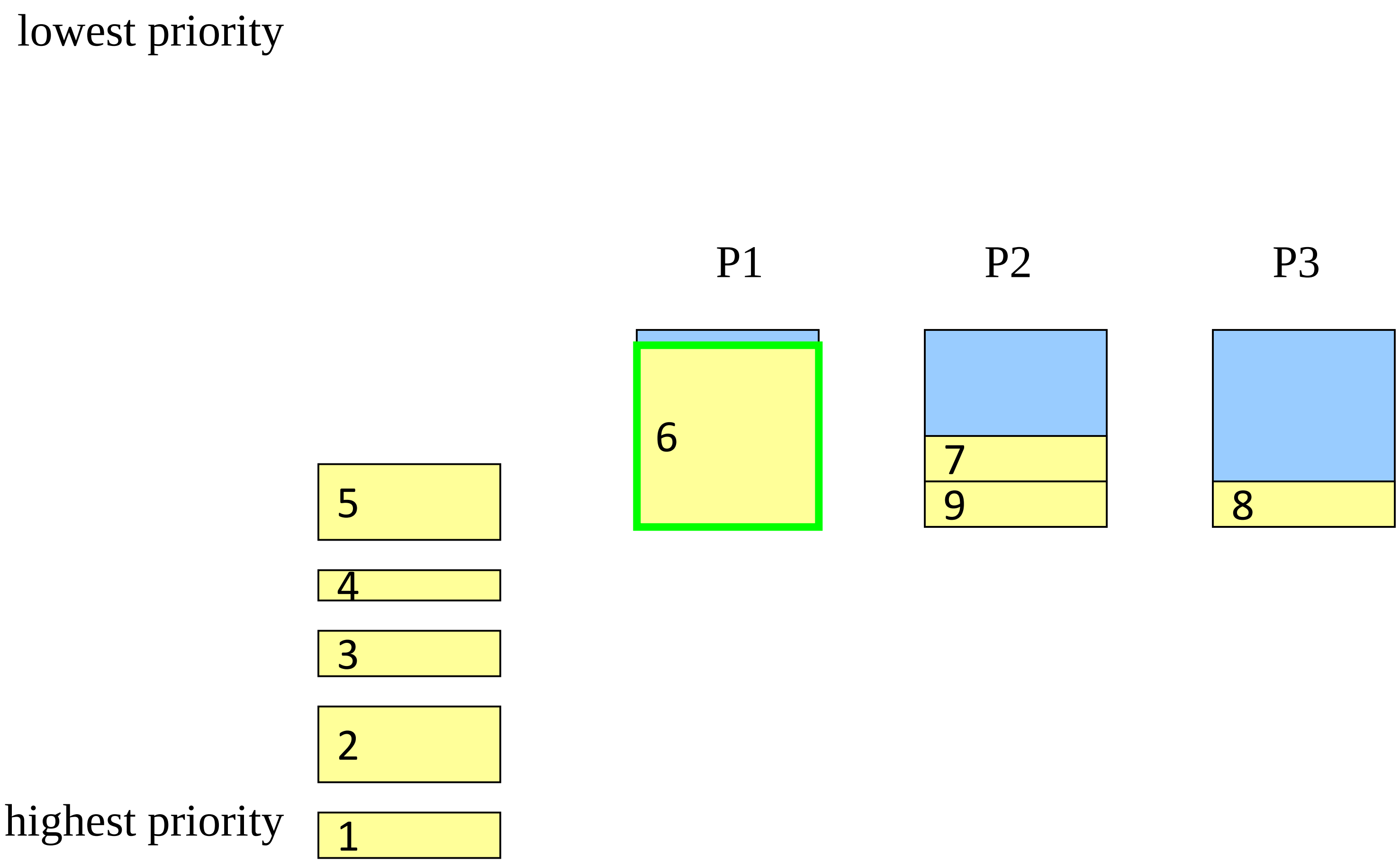
Solution for Heavy Tasks

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



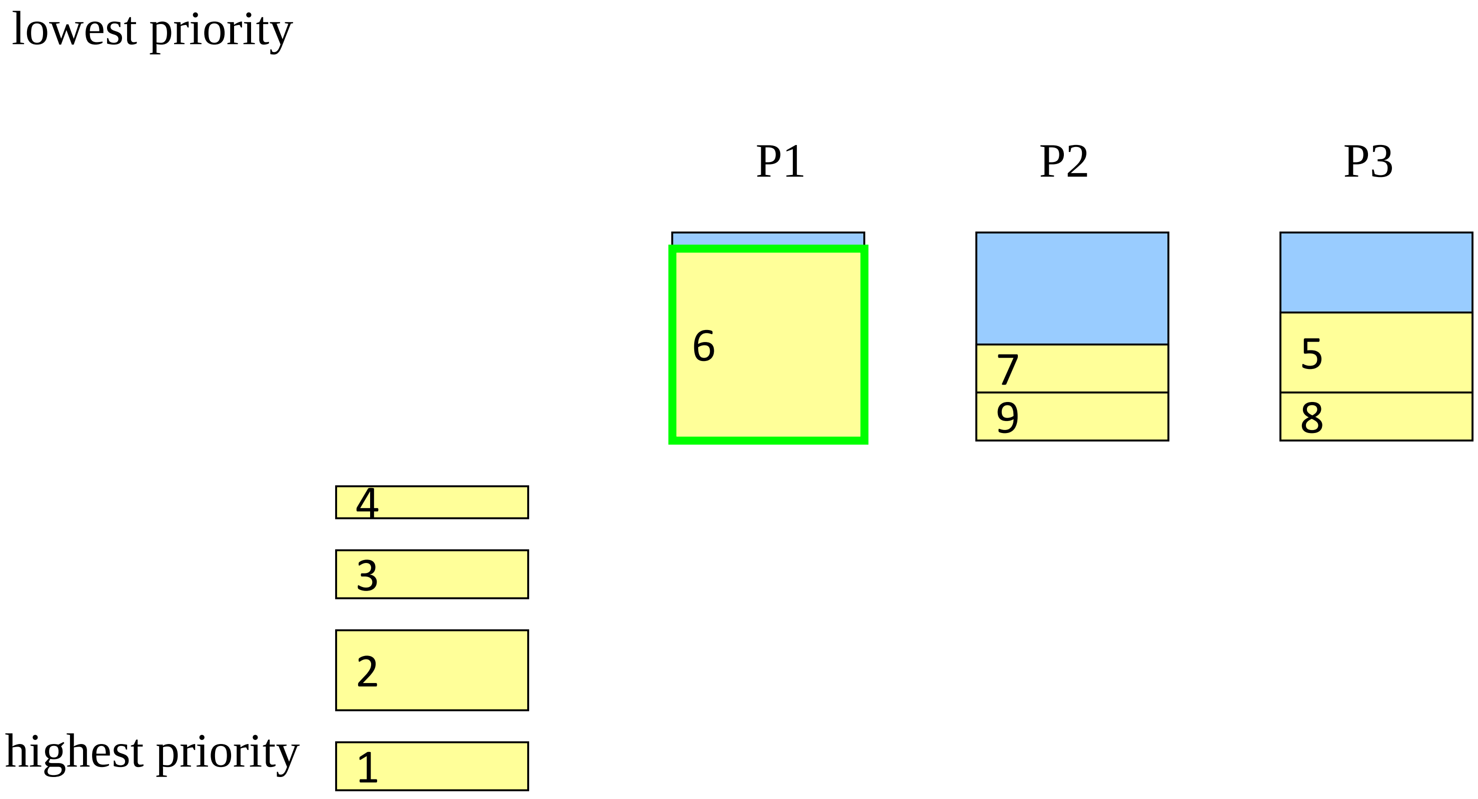
Solution for Heavy Tasks

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



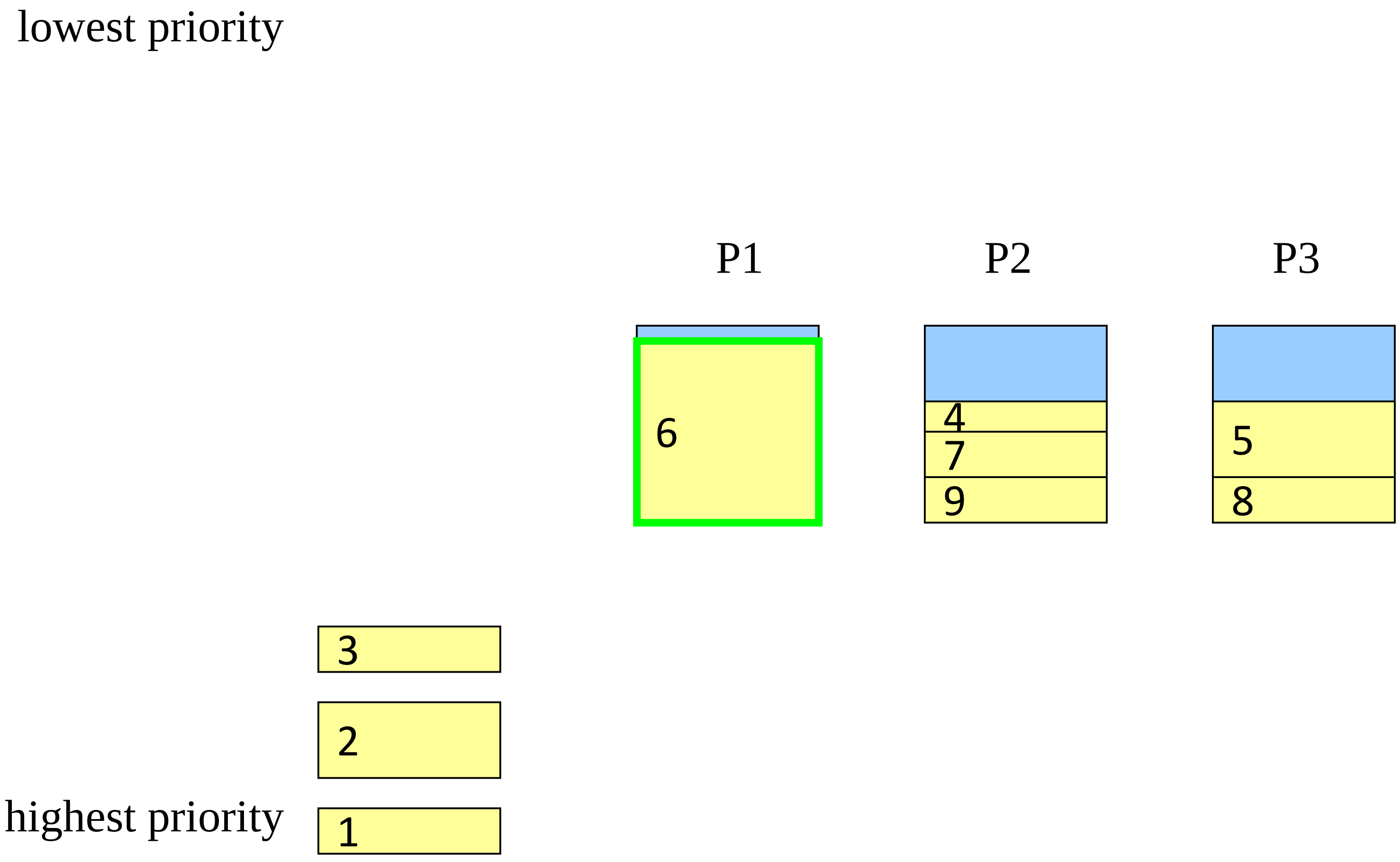
Solution for Heavy Tasks

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



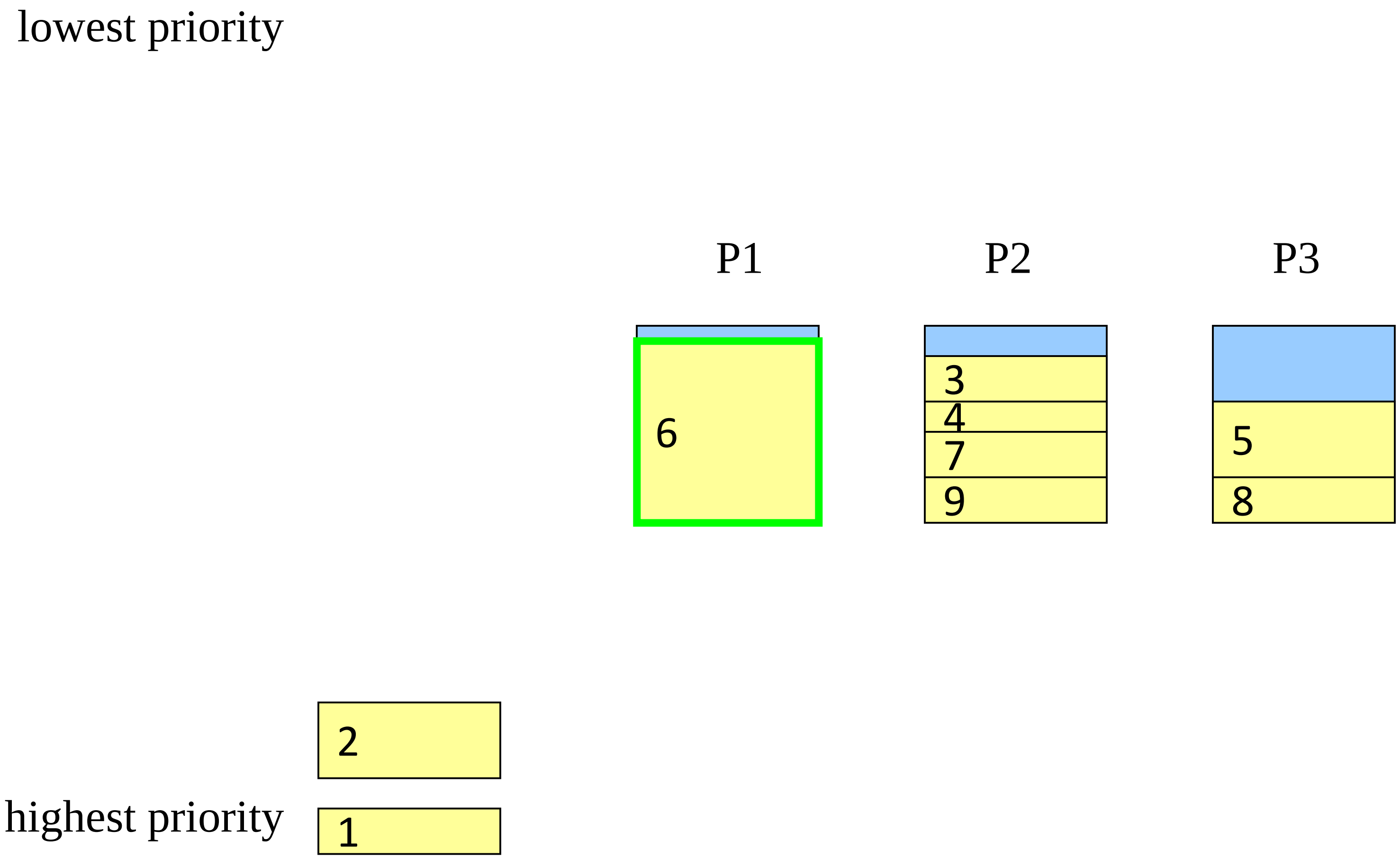
Solution for Heavy Tasks

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



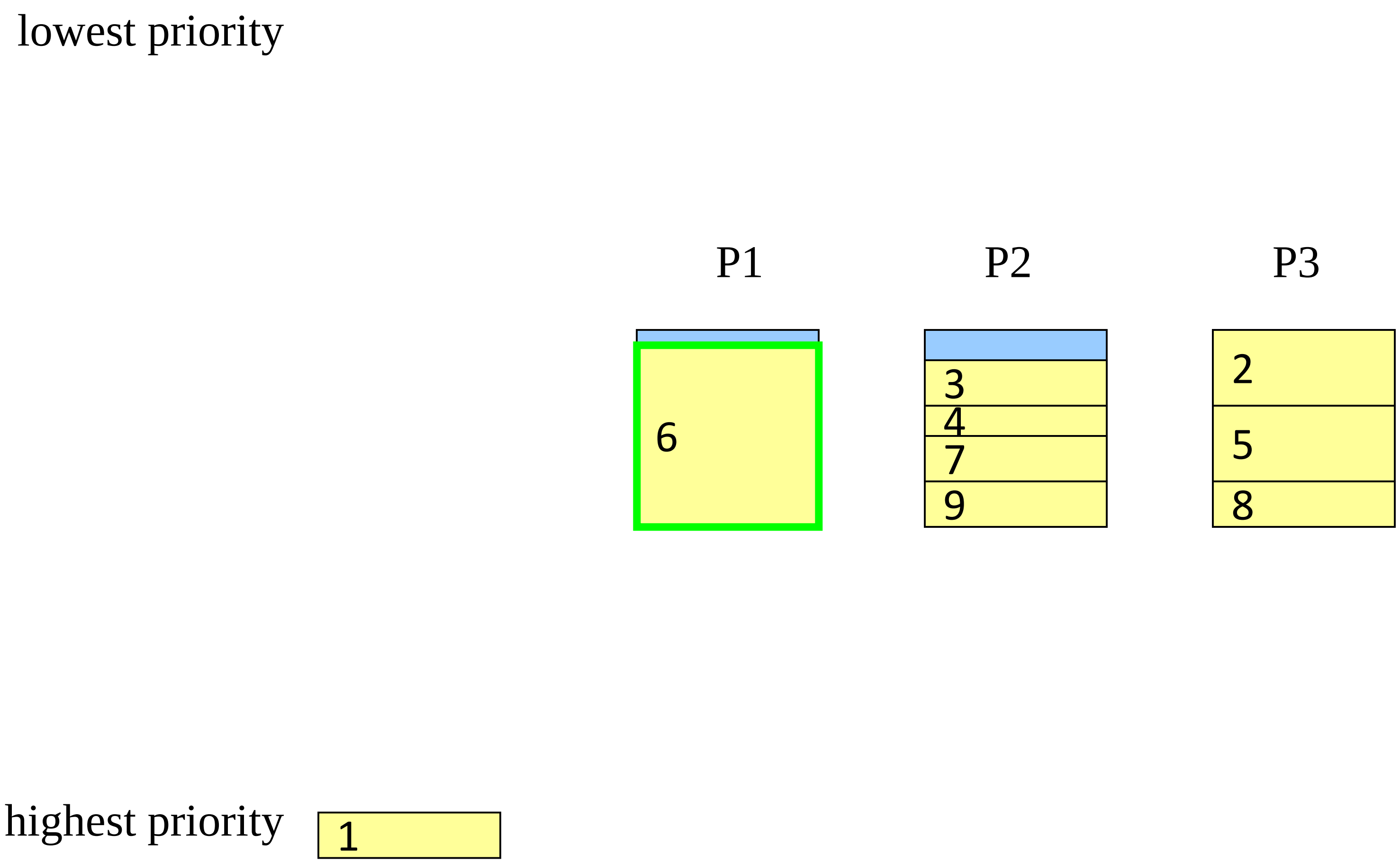
Solution for Heavy Tasks

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



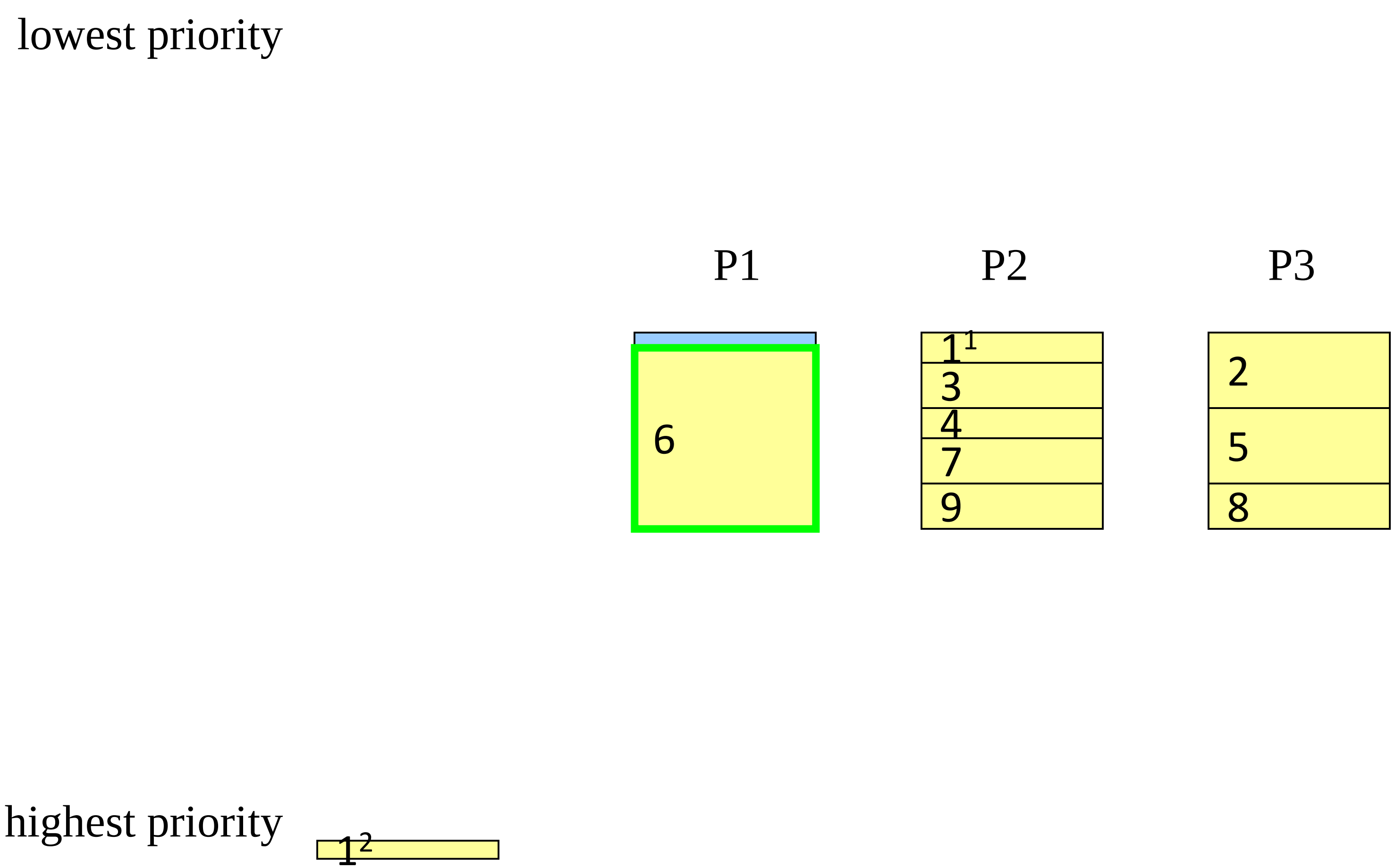
Solution for Heavy Tasks

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



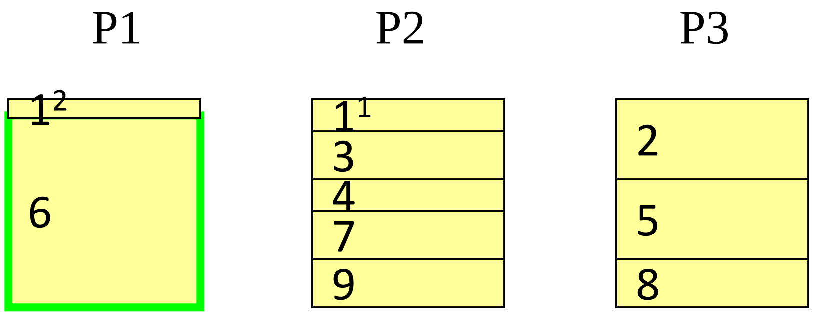
Solution for Heavy Tasks

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



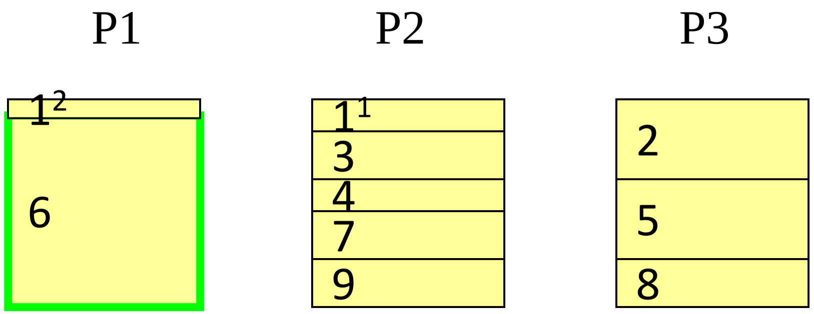
Solution for Heavy Tasks

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



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

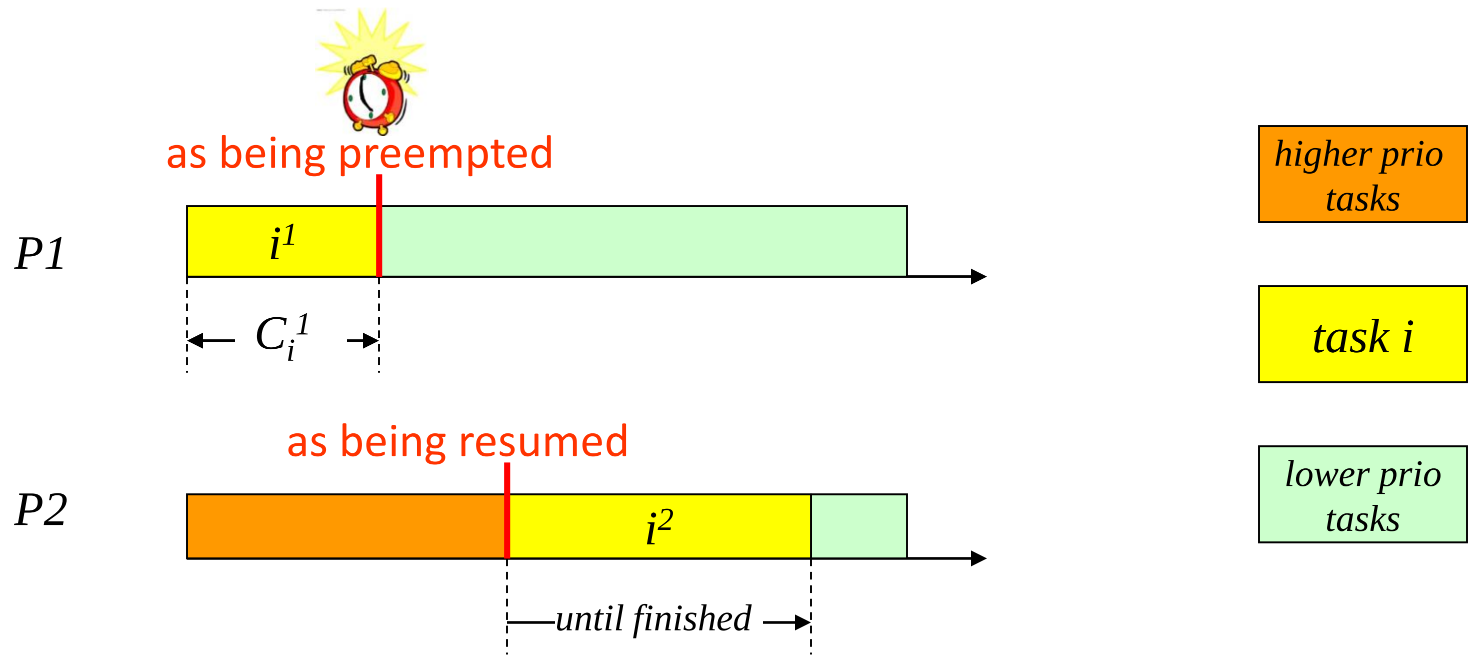
$$\frac{\sum C_i/T_i}{M} \leq N(2^{1/N} - 1)$$

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

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

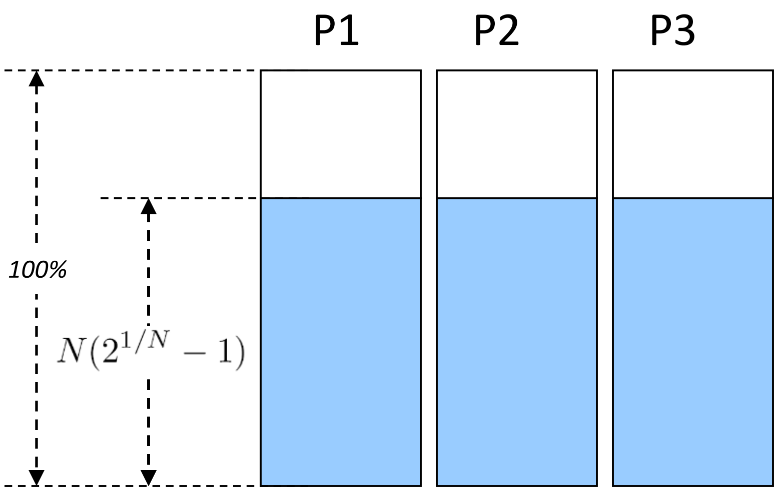
Implementation

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

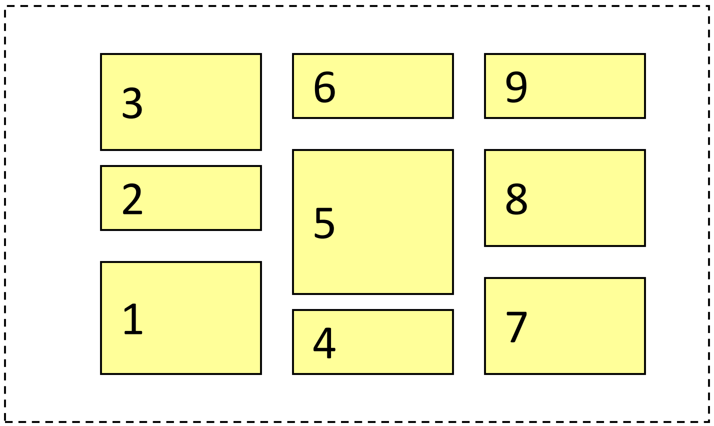


Further Improvement

Partioning using response time analysis

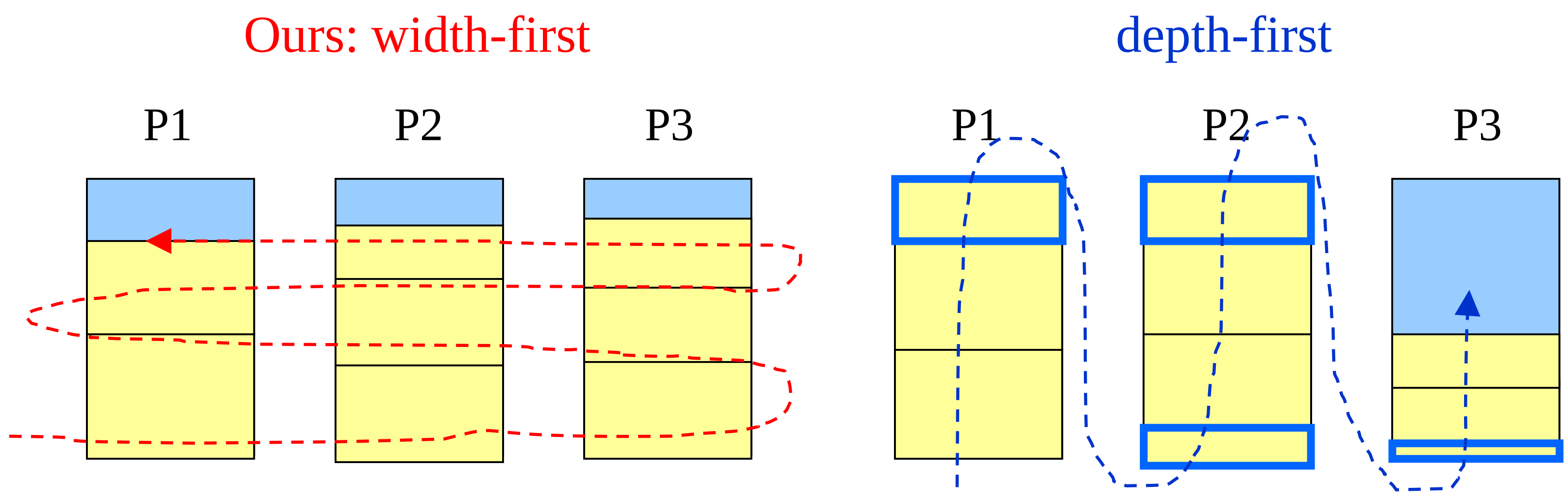


Schedulable?



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



Partitioned scheduling with Task Splitting: + &-

- High resource utilization
- Higher overhead (due to task migrations/preemptions)