# OUTLINE

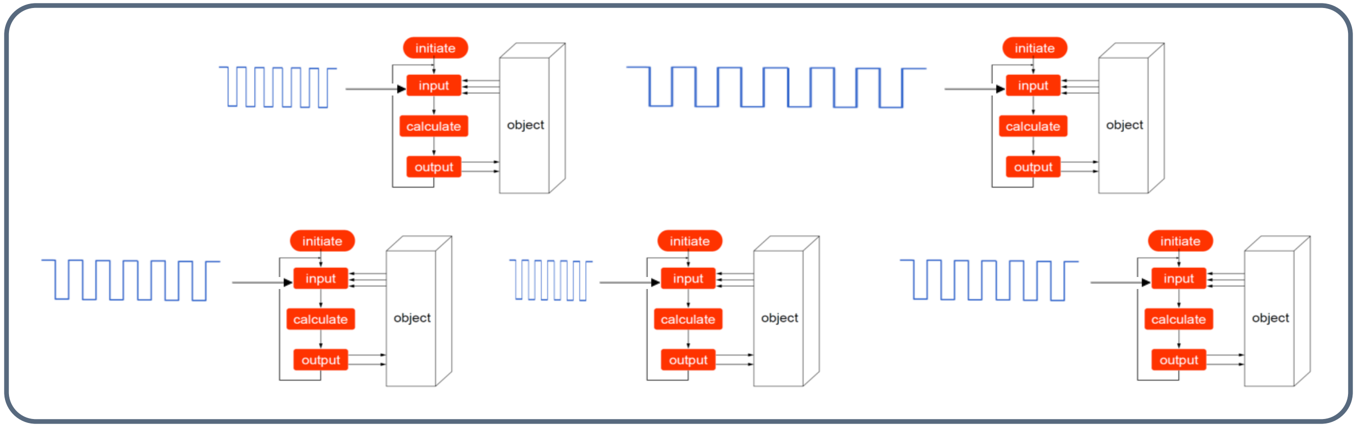
- **Multicore Challenges**
  - Why and what are multicores?
  - What we are doing in Uppsala: CoDeR-MP
  - The timing analysis problem
- **Possible Solutions – Partition/Isolation**
  - Dealing with Shared Caches [EMSOFT 2009]
  - Dealing with Bus Interference [RTSS 2010]
  - Dealing with Core Sharing [RTAS 2010]

# Dealing with Core Sharing:
# Fixed-Priority Multiprocessor Scheduling

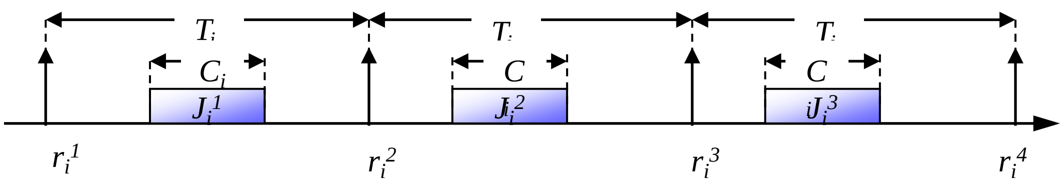
Joint work with
Nan Guan, Martin Stigge and Yu Ge

Northeastern University, China
Uppsala University, Sweden

# 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)

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

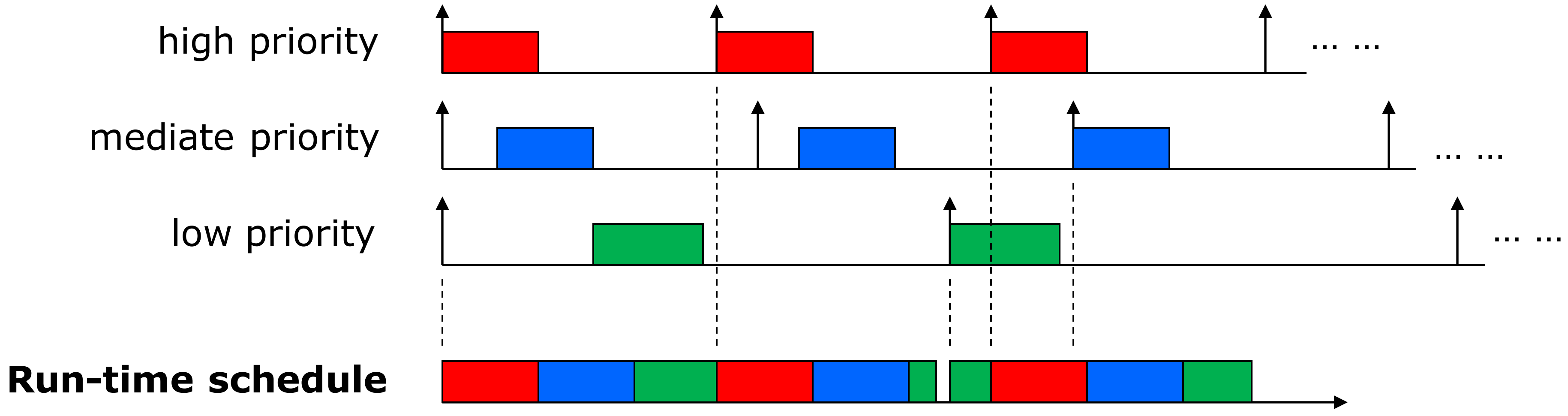
$$\Rightarrow \quad \text{the task set is schedulable}$$

number of tasks

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

# Rate Monotonic Scheduling

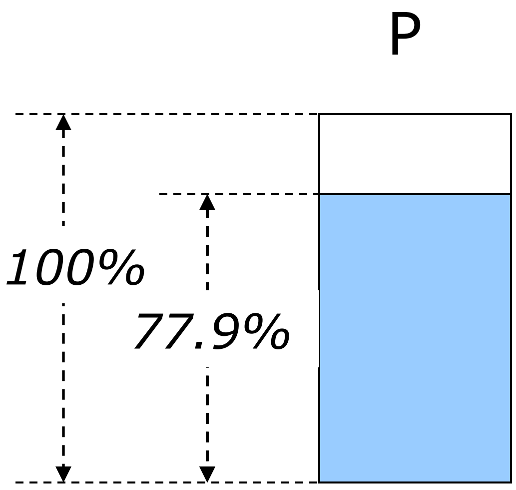
- Priority assignment: shorter period → higher prio.
- Run-time schedule: the highest priority first




- **How to check whether all deadlines are met?**

# Liu and Layland's Utilization Bound

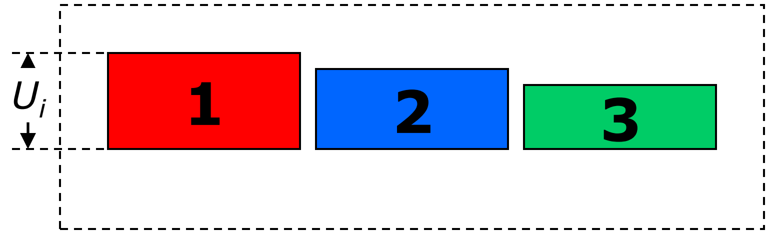
- Schedulability Analysis




*Liu and Layland's bound:*

$$3 \times (2^{1/3} - 1) = 77.9\%$$

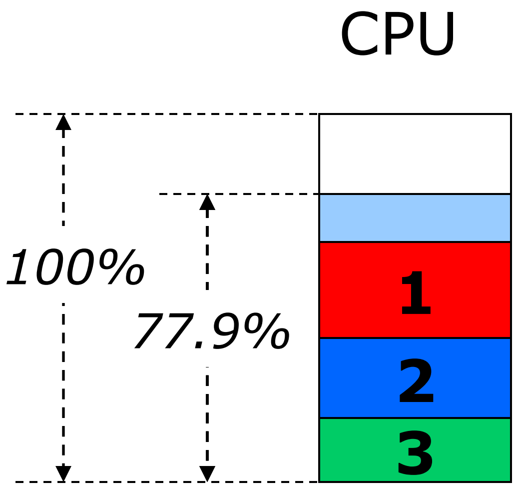
Schedulable?




$U_i = C_i / T_i$

# Liu and Layland’s Utilization Bound

- Schedulability Analysis




Yes, schedulable!

*Liu and Layland’s bound:*

$$3 \times (2^{1/3} - 1) = 77.9\%$$

# Multiprocessor (multicore) Scheduling

- Significantly more difficult:
  - Timing anomalies
  - Hard to identify the worst-case scenario
  - Bin-packing/NP-hard problems
  - Multiple resources e.g. caches, bandwidth
  - … …

# Open Problem (since 1973)

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

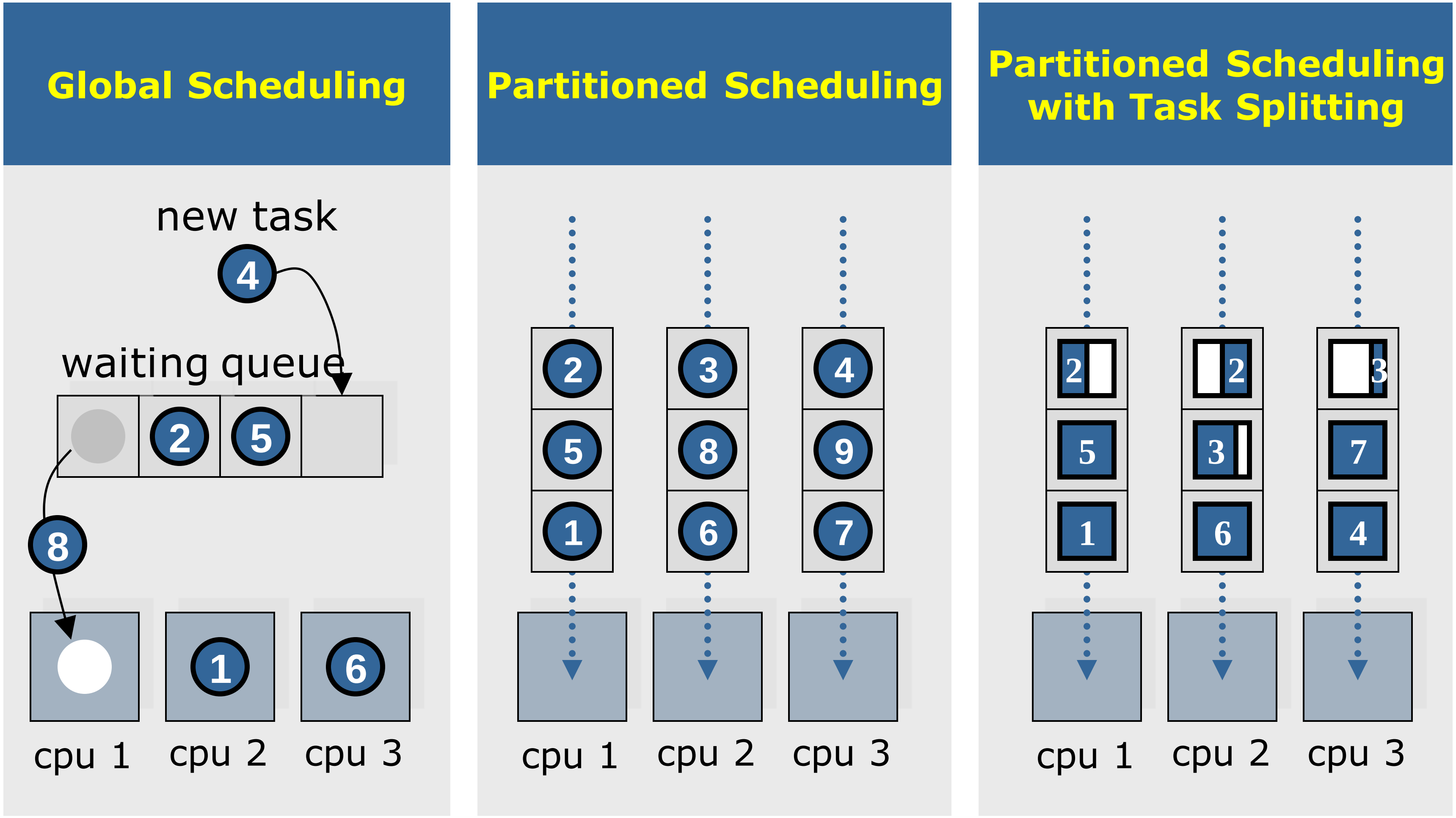
$$\frac{\sum C_i/T_i}{M} \leq N(2^{1/N} - 1)$$

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

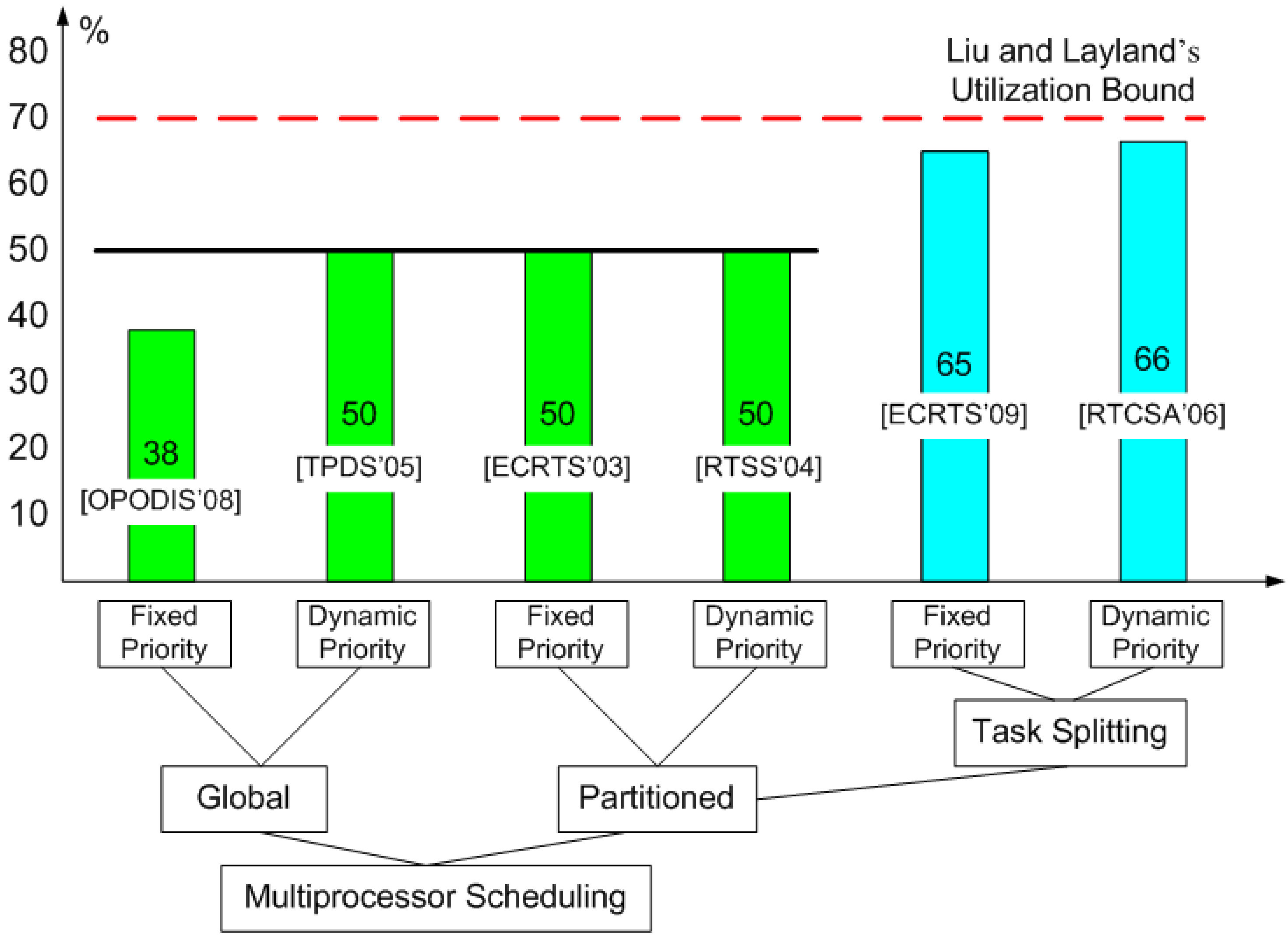
?

number of processors

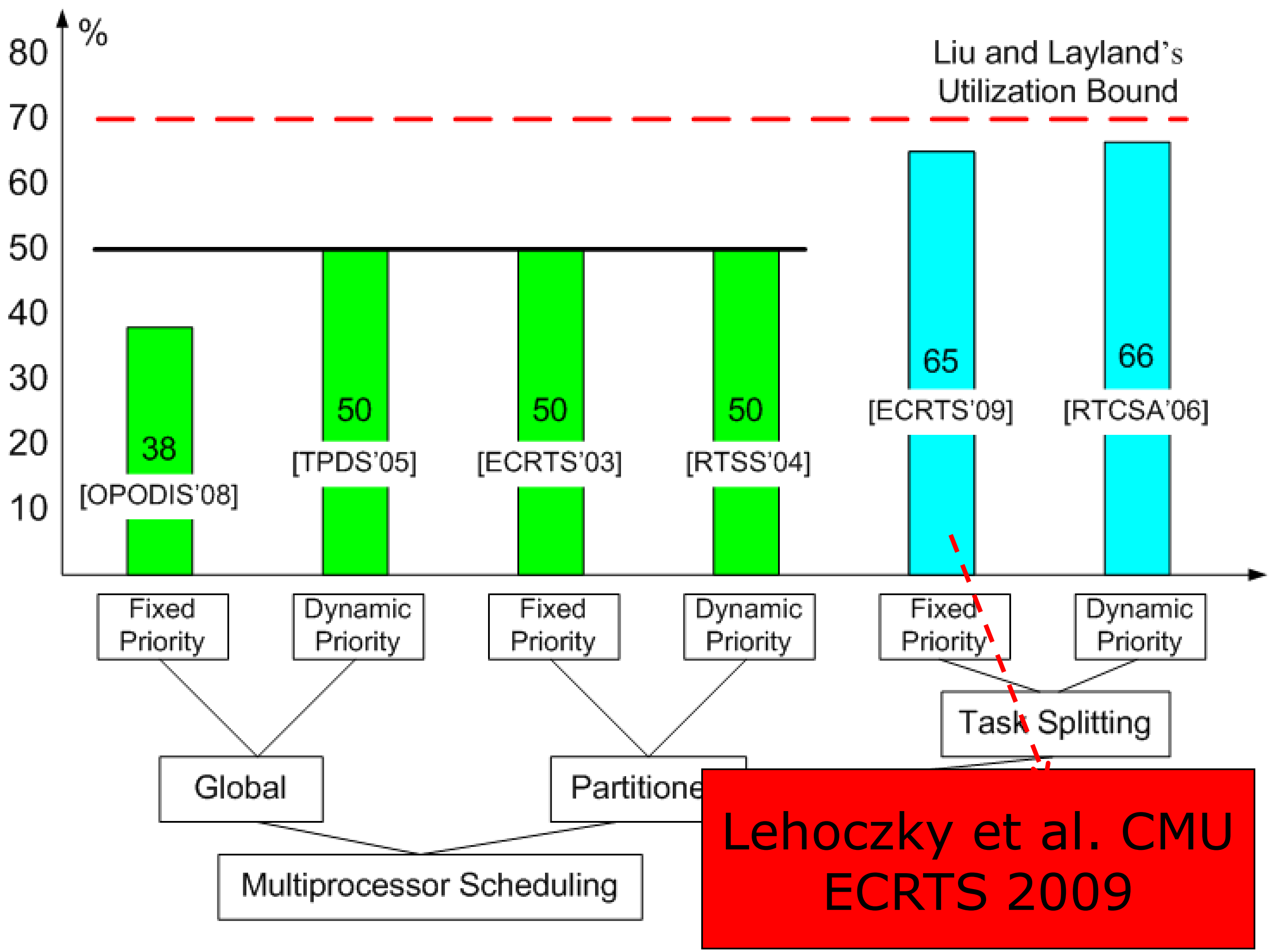
# Multiprocessor Scheduling



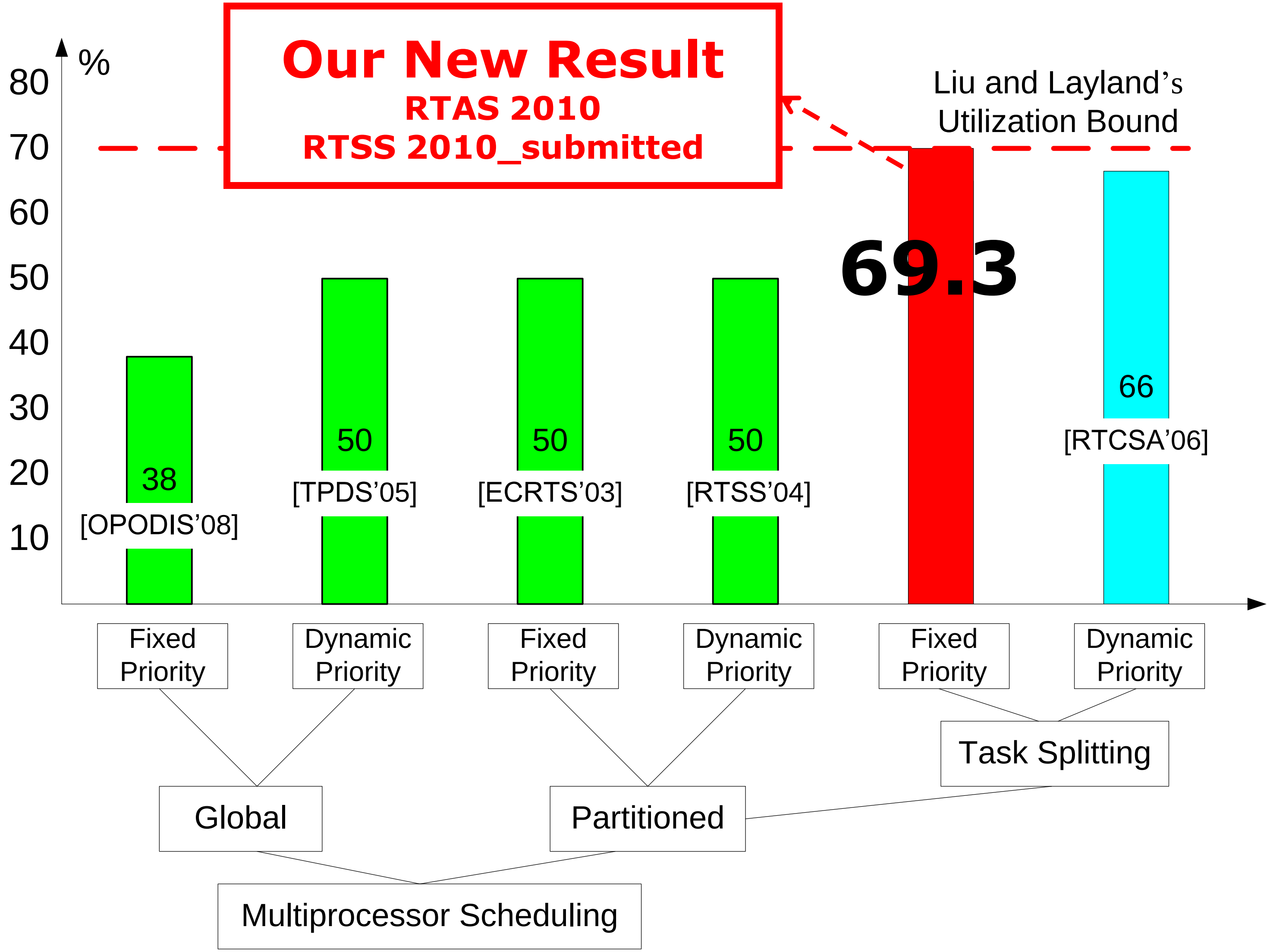
# Best Known Results (before 2010)



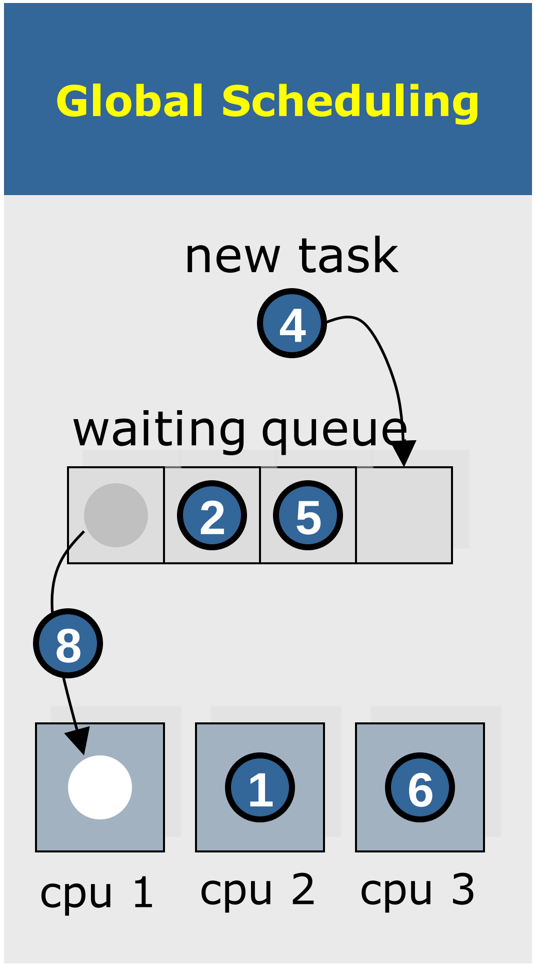
# Best Known Results (before 2010)



# Best Known Results

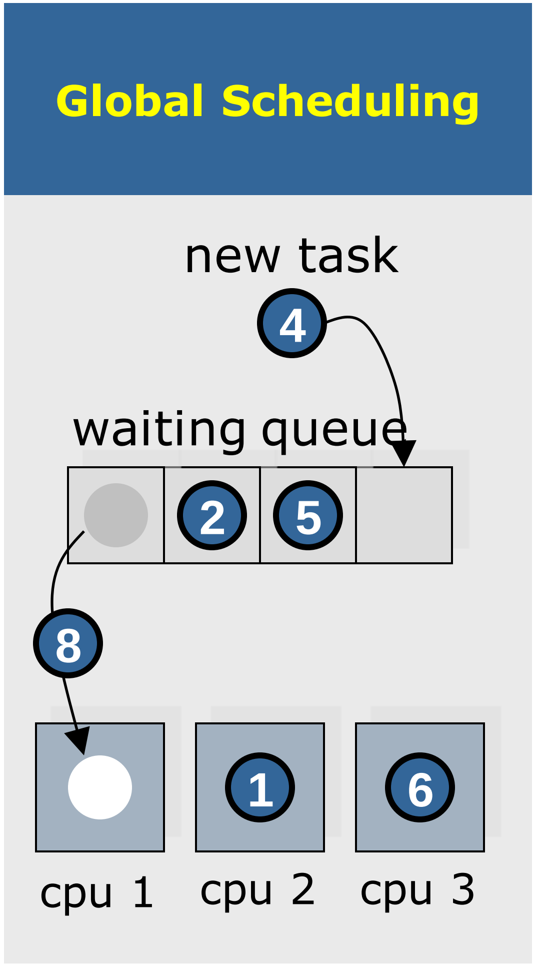


# Multiprocessor Scheduling




**Would fixed-priority scheduling e.g. "RMS" work?**

# Multiprocessor Scheduling

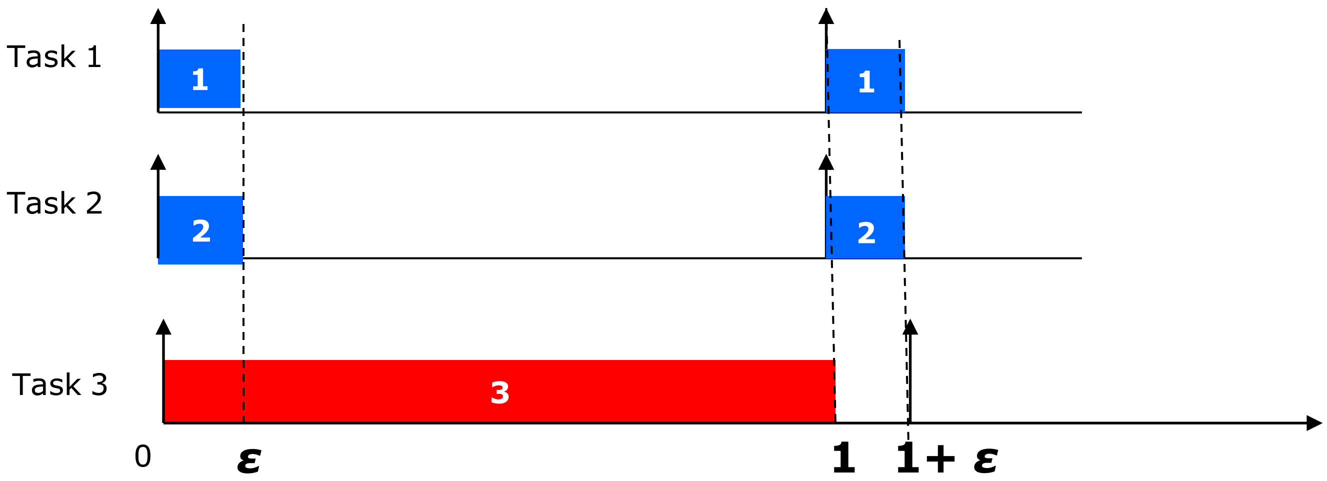



**Would fixed-priority scheduling e.g. "RMS" work?**

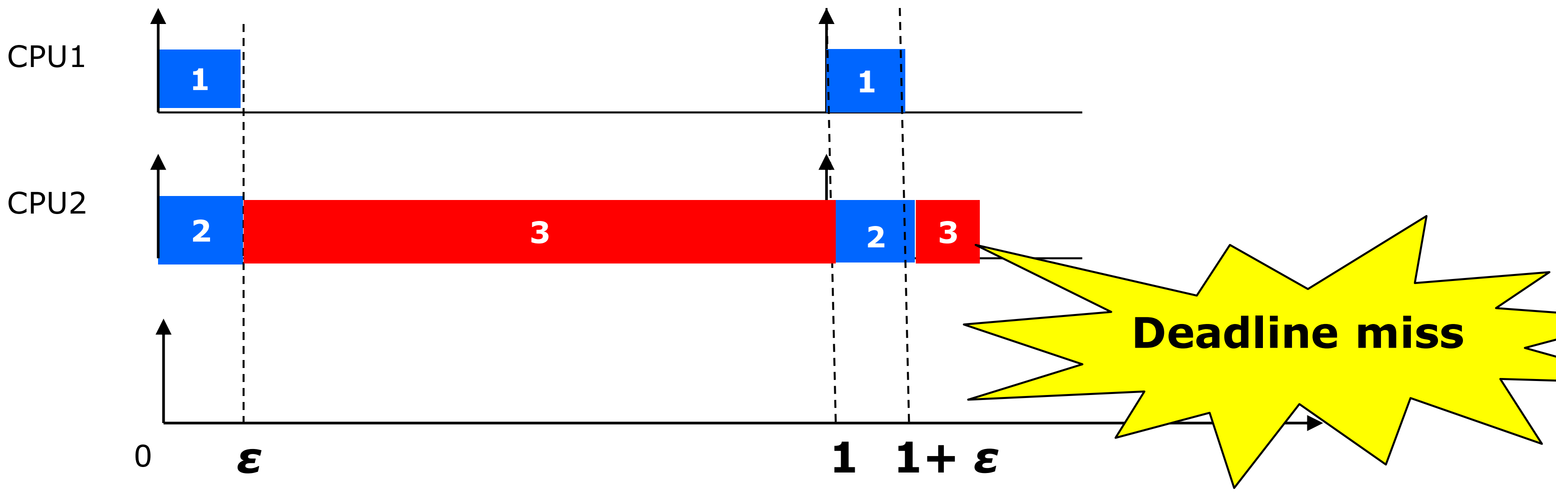
**Unfortunately "RMS" suffers from the Dhall's anomali**

***Utilization may be "0%"***

# Dhall's anomali



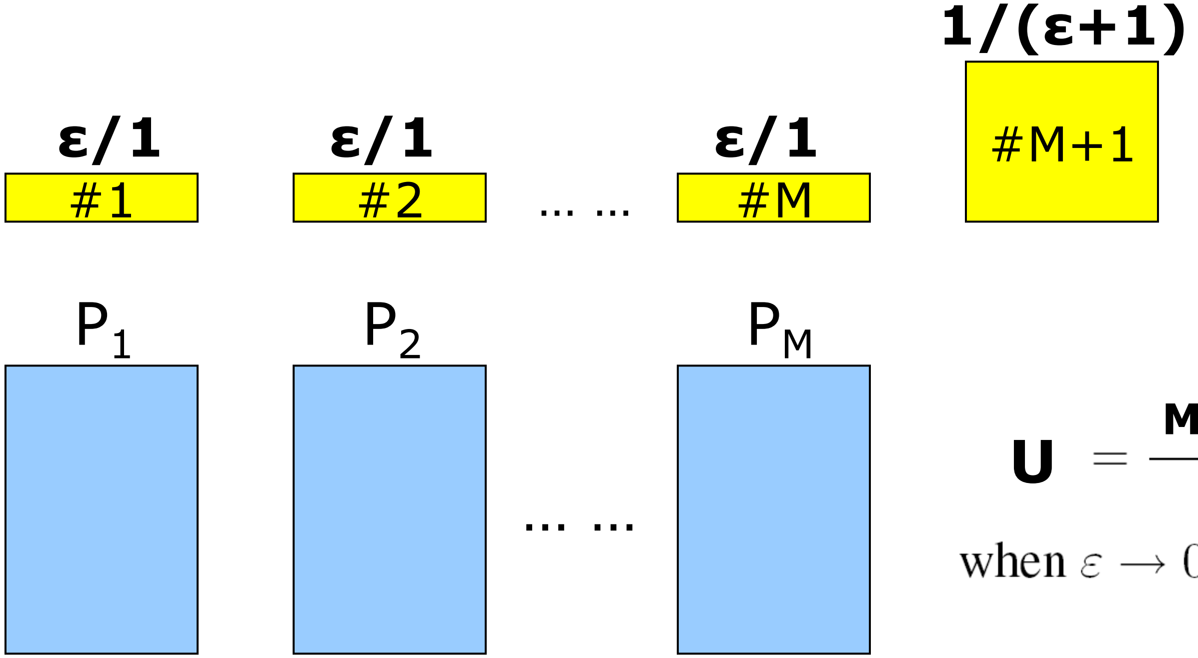
# Dhall’s anomali




**Schedule the 3 tasks on 2 CPUs using “RMS**

# Dhall's anomali

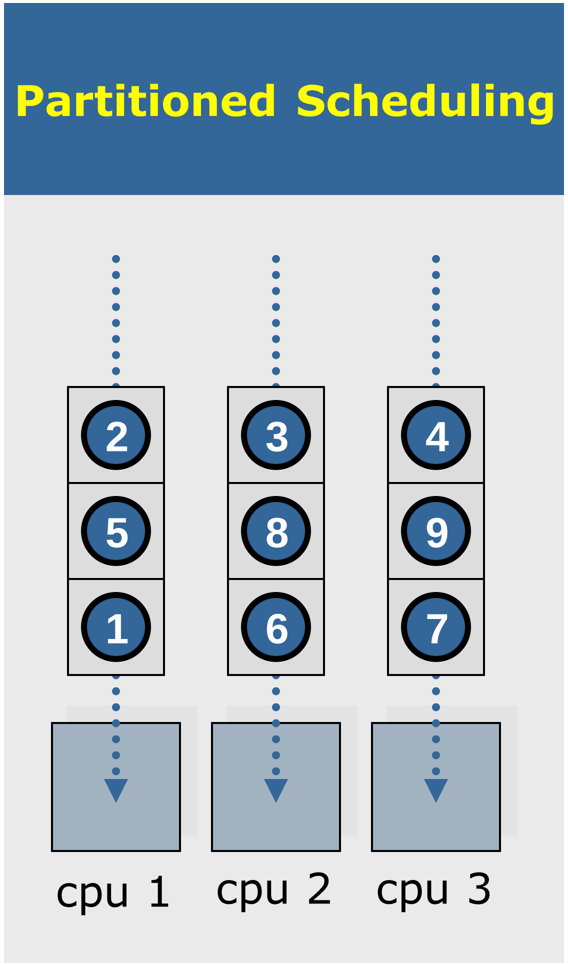
**(M+1 tasks and M processors)**



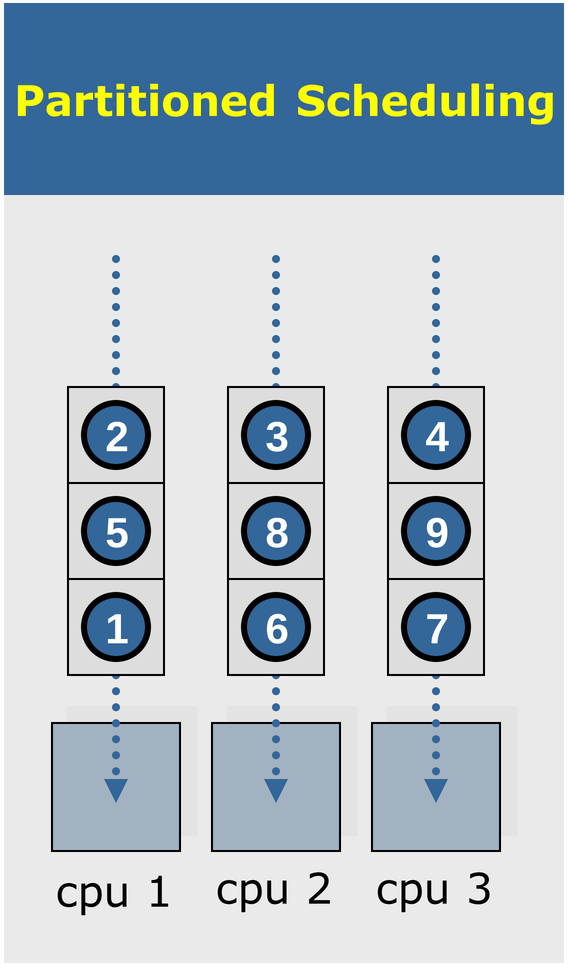

$$\mathbf{U} = \frac{\mathbf{M*\varepsilon + 1/(1+\varepsilon)}}{\mathbf{M}} \rightarrow \mathbf{0}$$

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

# Multiprocessor Scheduling



# Multiprocessor Scheduling




**Resource utilization may be limited to 50%**

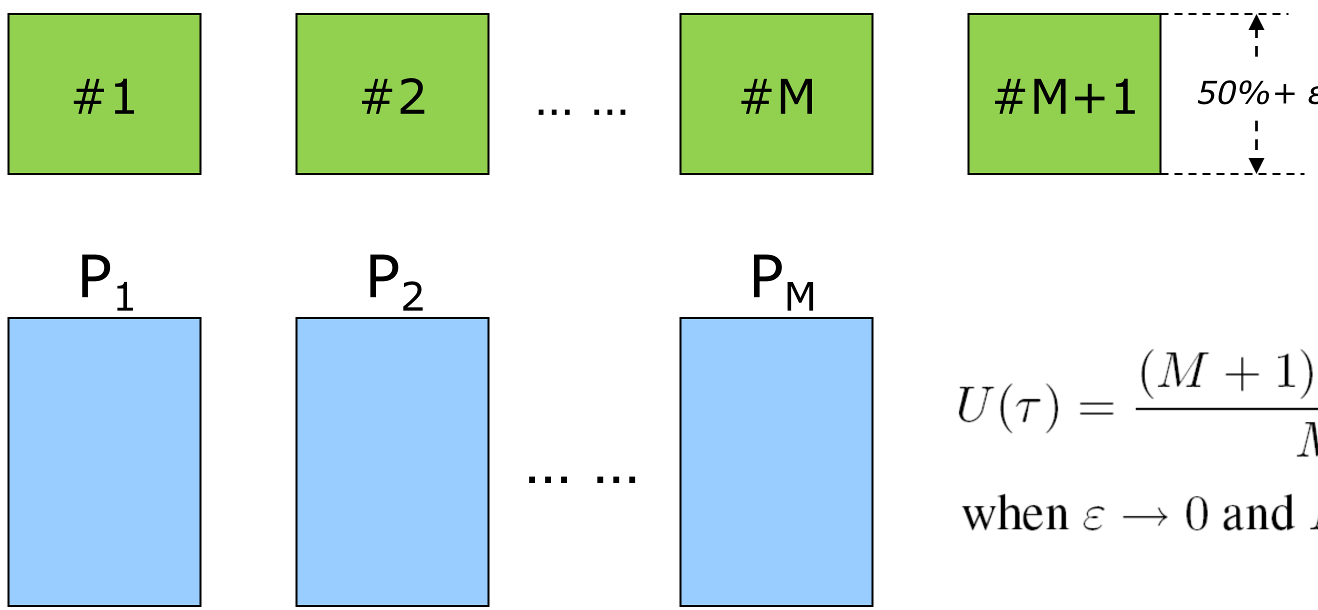
# Partitioned Scheduling

- 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




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

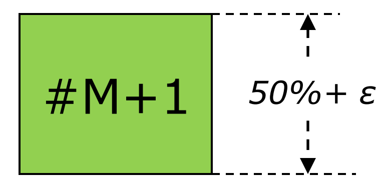
when $\varepsilon \longrightarrow 0$ and $M \longrightarrow +\infty$

# Partitioned Scheduling

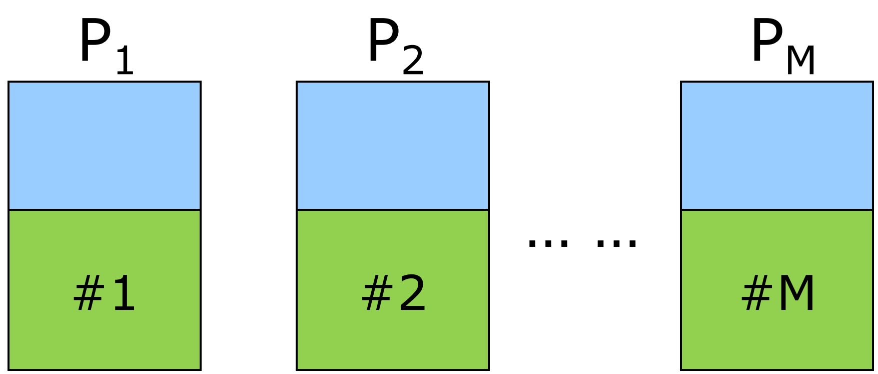
- The Partitioning Problem is similar to Bin-packing Problem (NP-hard)
- Limited Resource Usage

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

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

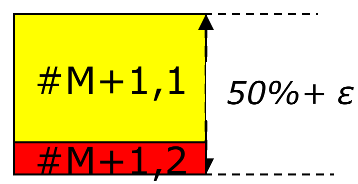
- The Partitioning Problem is similar to Bin-packing Problem (NP-hard)

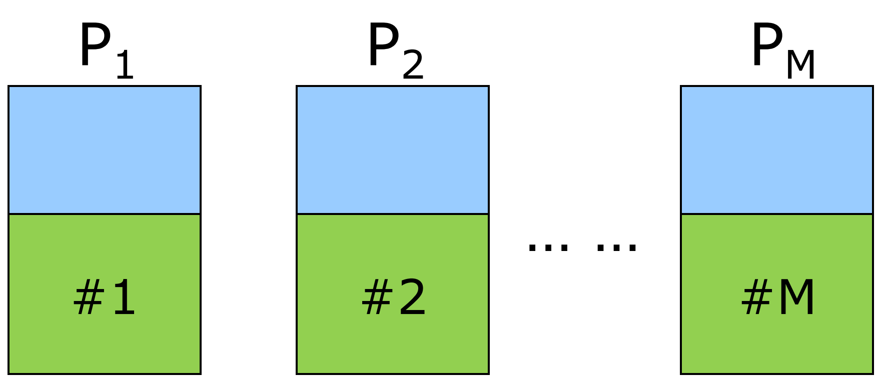
- Limited Resource Usage

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

necessary condition to guarantee schedulability

… …




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

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

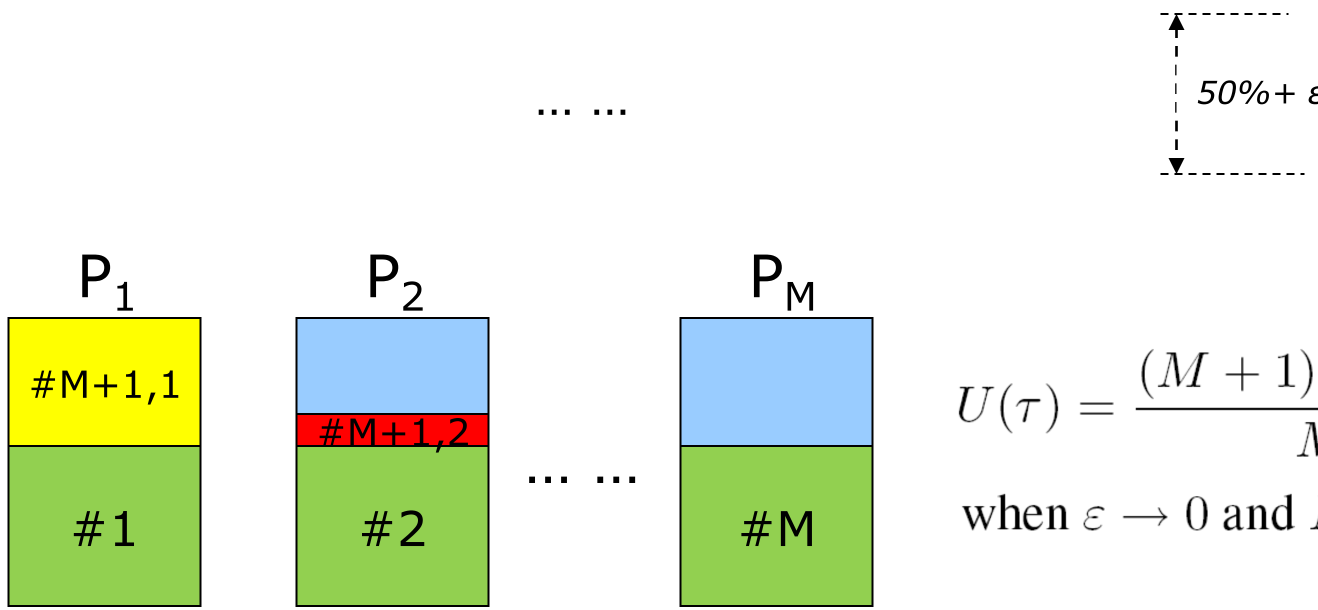
# Partitioned Scheduling

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

- Limited Resource Usage

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

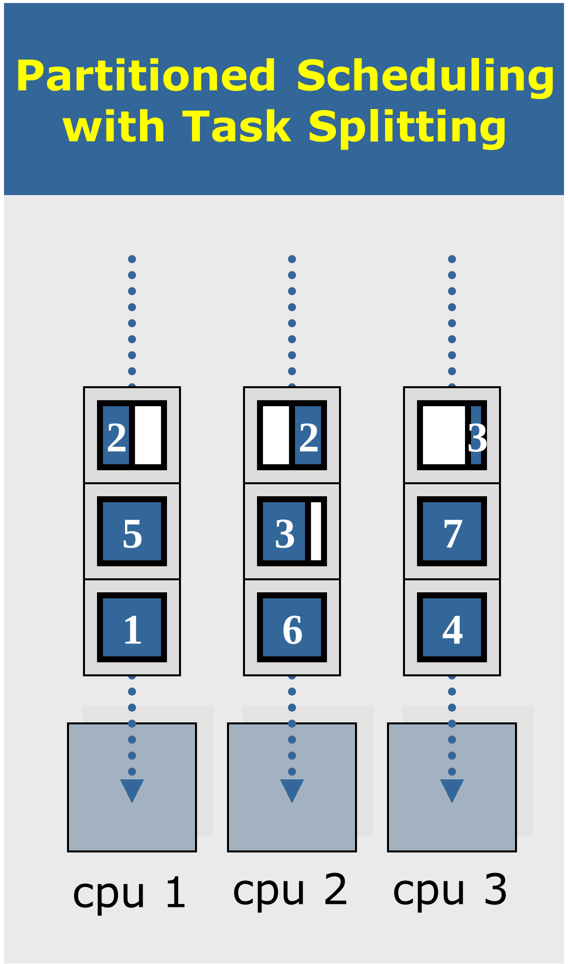
necessary condition to guarantee schedulability




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

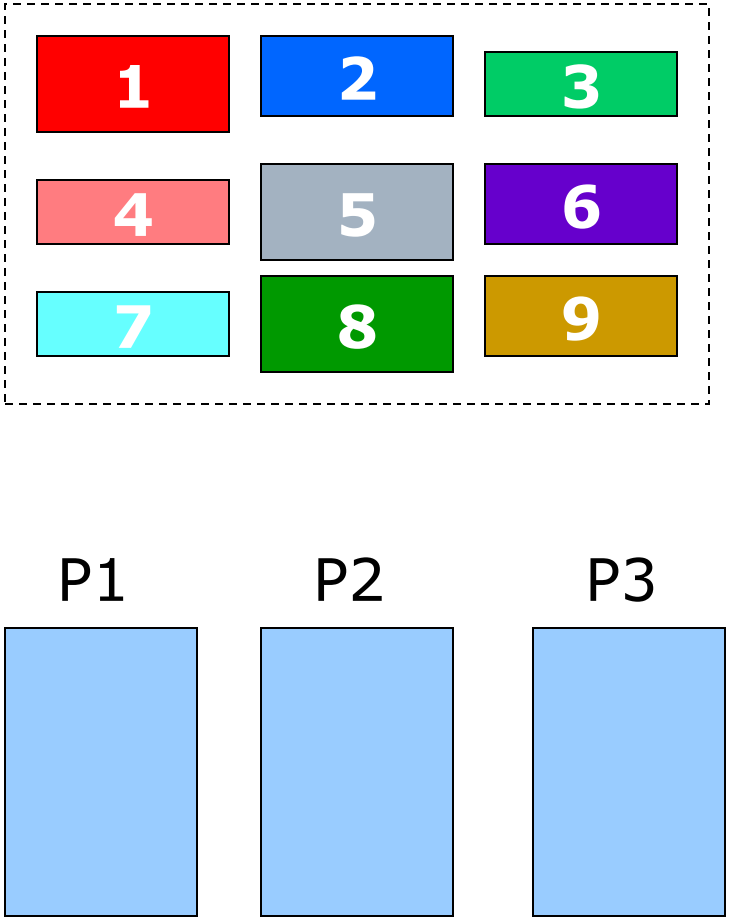
when $\varepsilon \rightarrow 0$ and $M \rightarrow +\infty$

# Multiprocessor Scheduling



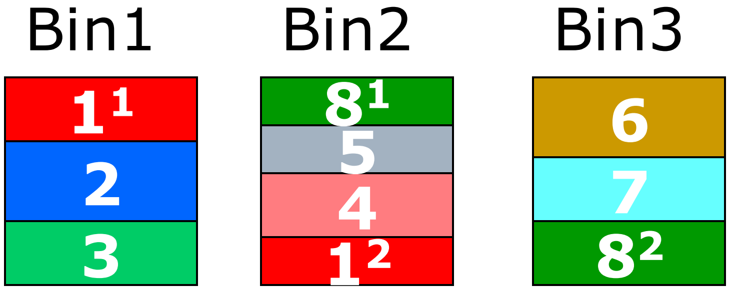
# Partitioned Scheduling

- Partitioning



# Bin-Packing with Item Splitting

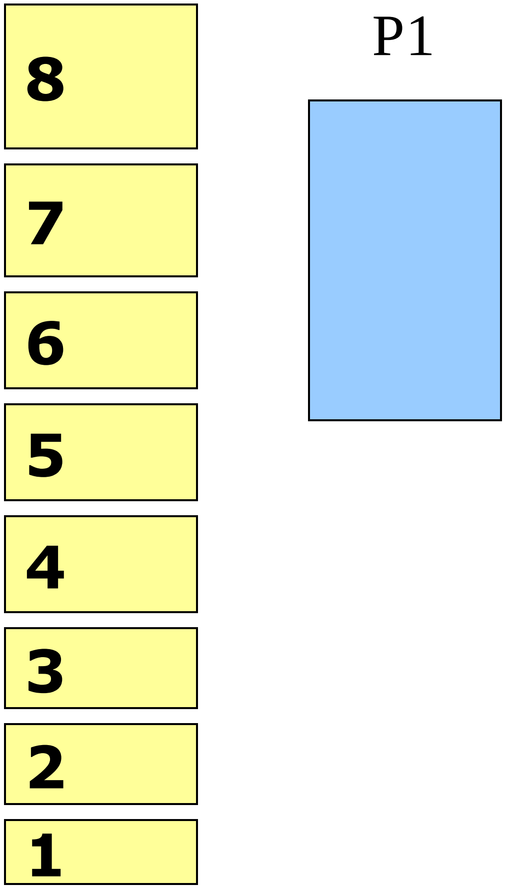
- Resource can be “fully” (better) utilized



# Previous Algorithms

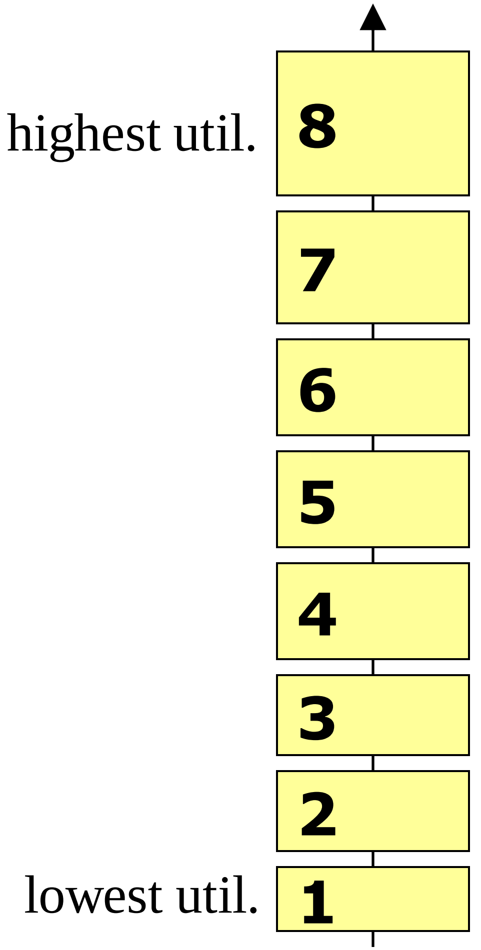
[Kato et al. IPDPS'08] [Kato et al. RTAS'09] [Lakshmanan et al. ECRTS'09]

- Sort the tasks in some order e.g. utilization or priority order
- Select a processor, and assign as many tasks as possible



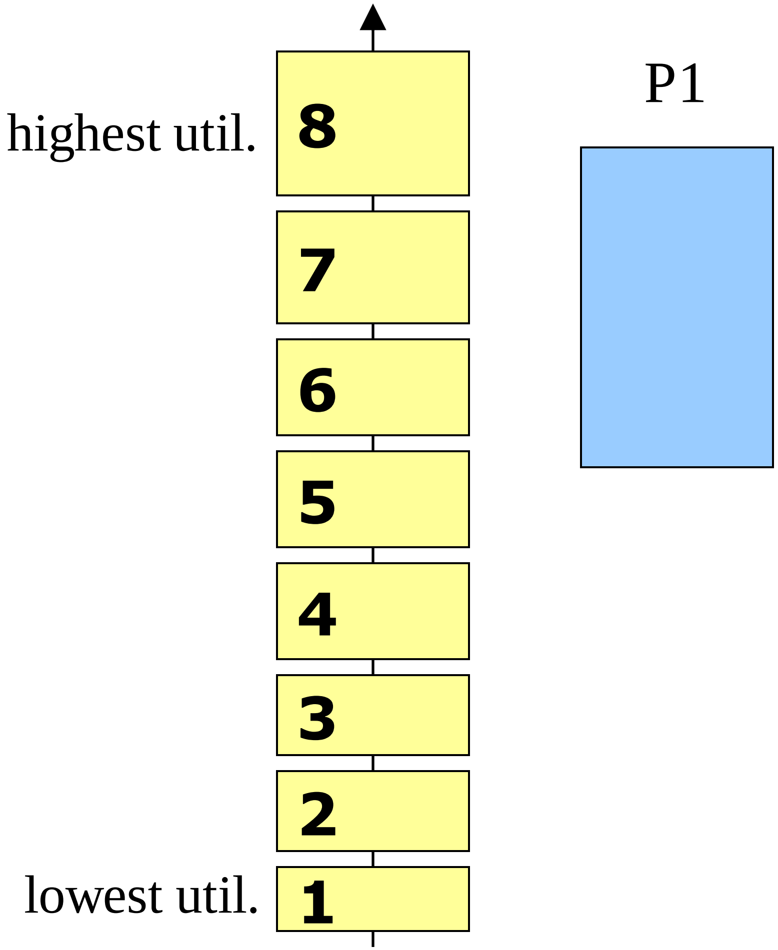
# Lakshmanan’s Algorithm [ECRTS’09]

- Sort all tasks in decreasing order of utilization



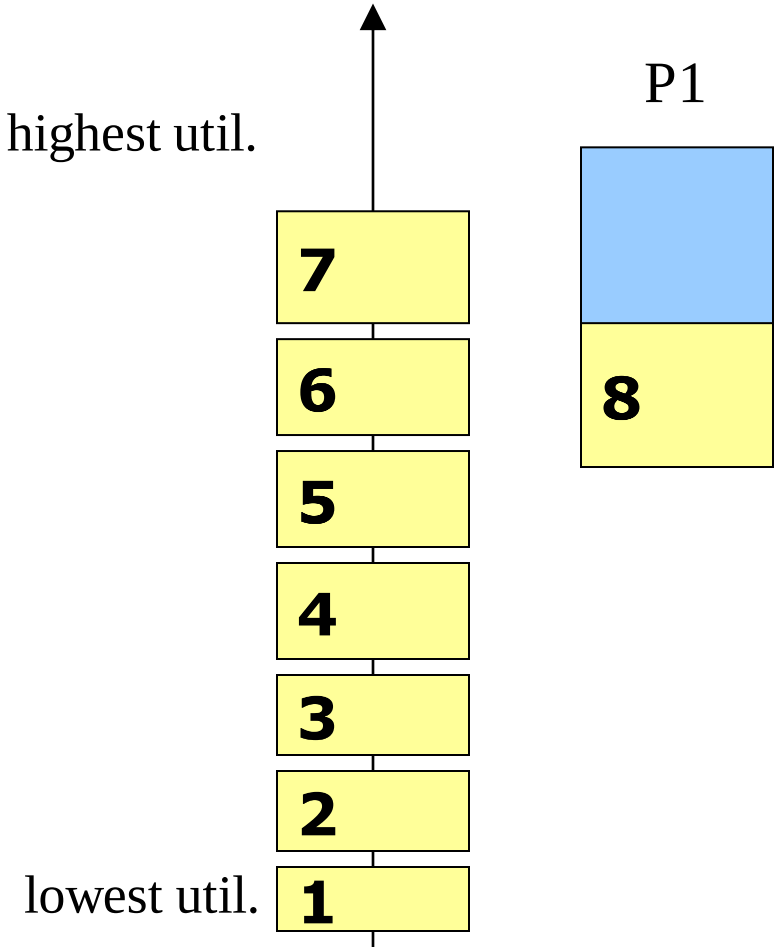
# Lakshmanan's Algorithm [ECRTS'09]

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



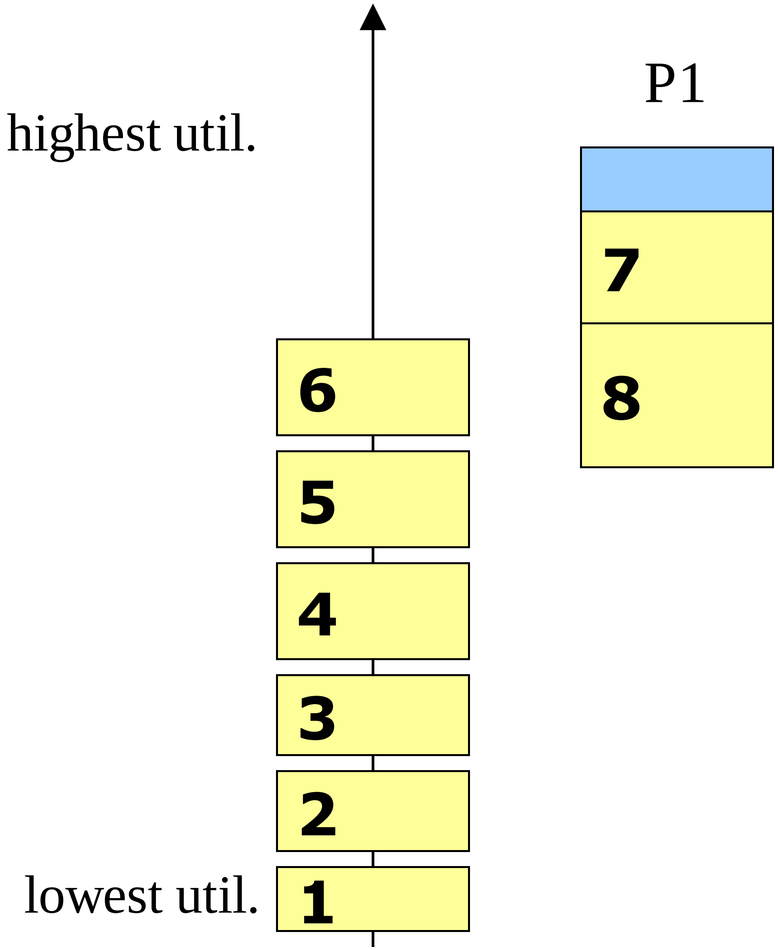
# Lakshmanan's Algorithm [ECRTS'09]

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



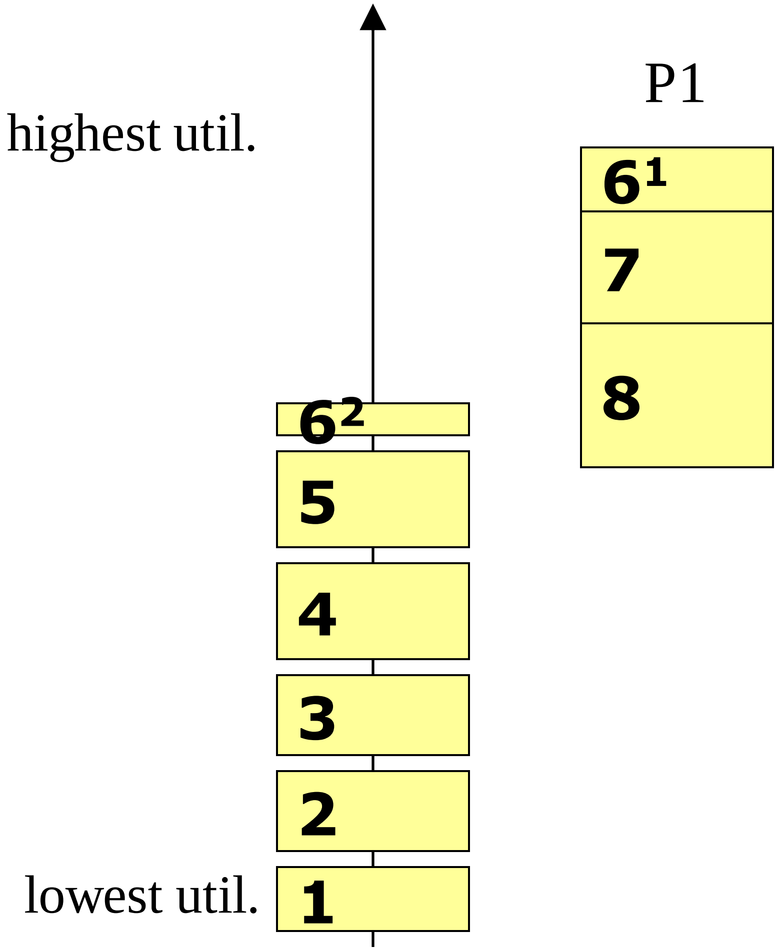
# Lakshmanan's Algorithm [ECRTS'09]

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



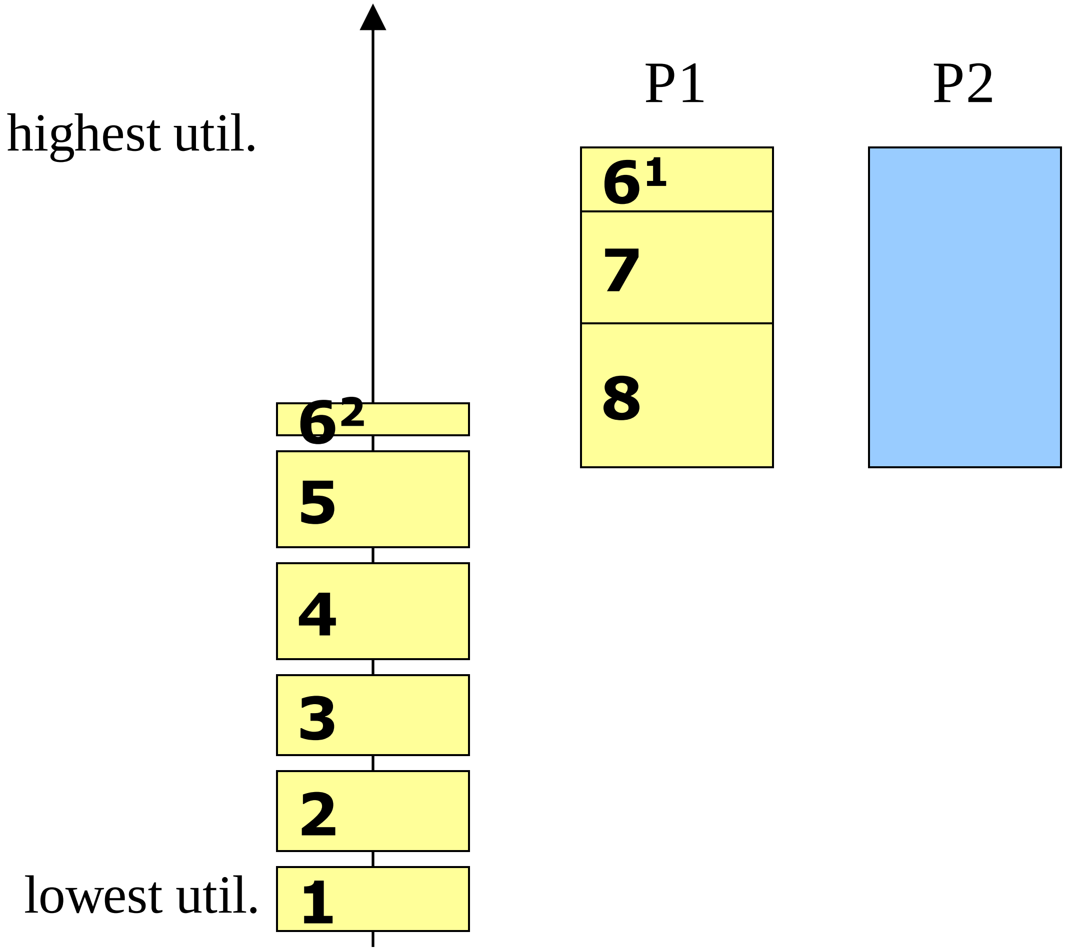
# Lakshmanan's Algorithm [ECRTS'09]

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



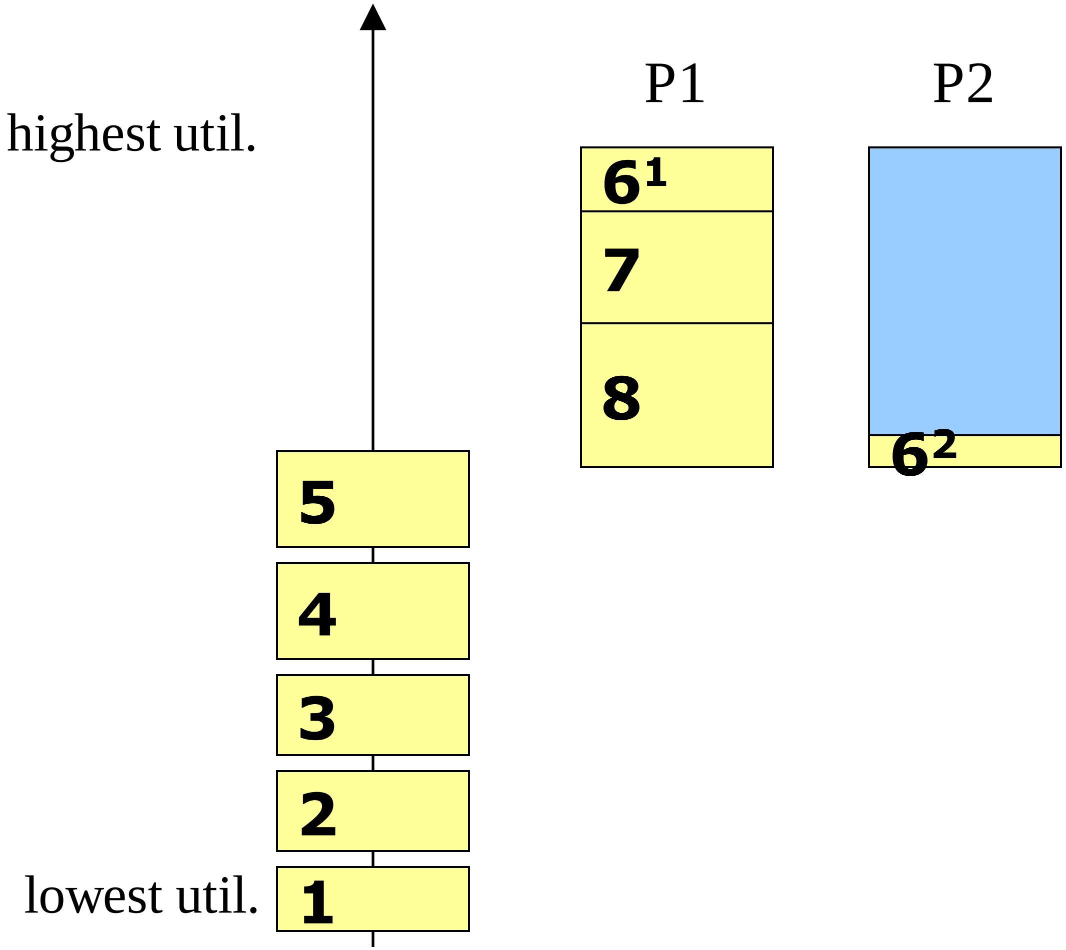
# Lakshmanan’s Algorithm [ECRTS’09]

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



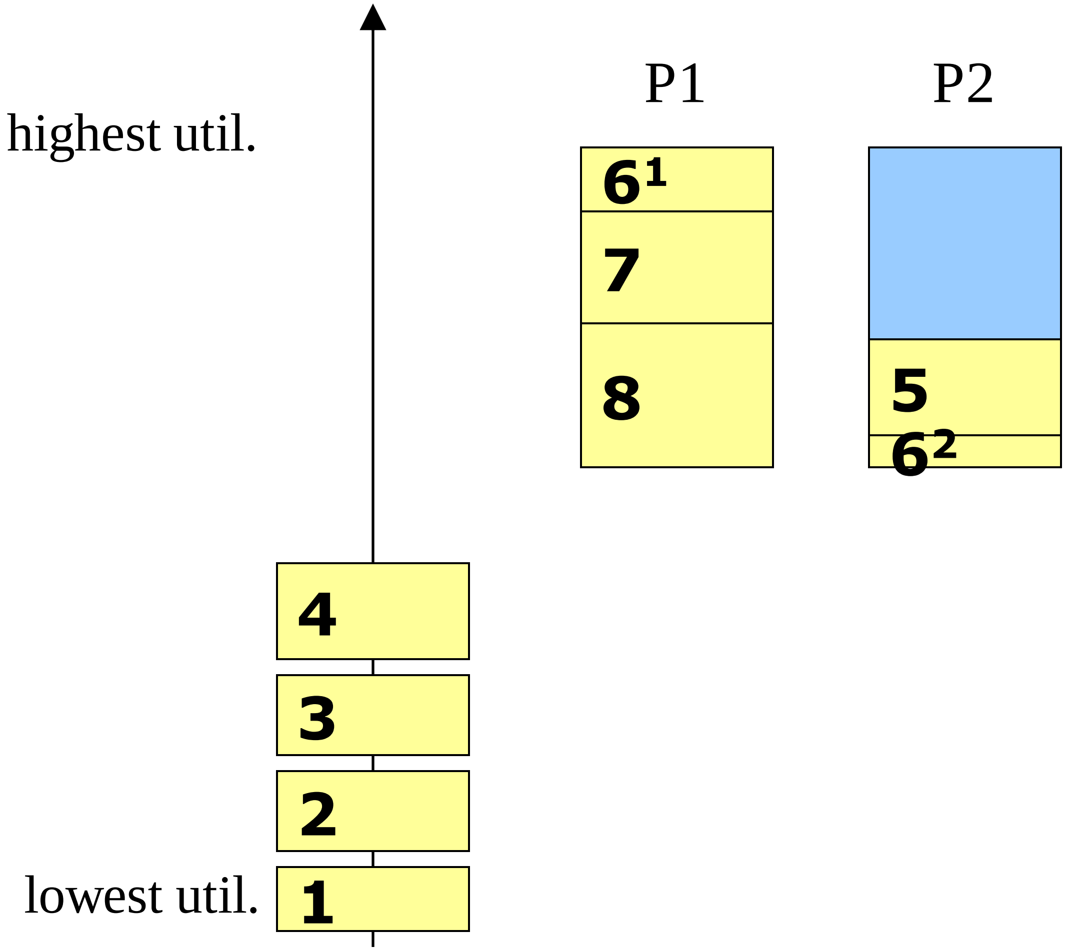
# Lakshmanan's Algorithm [ECRTS'09]

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



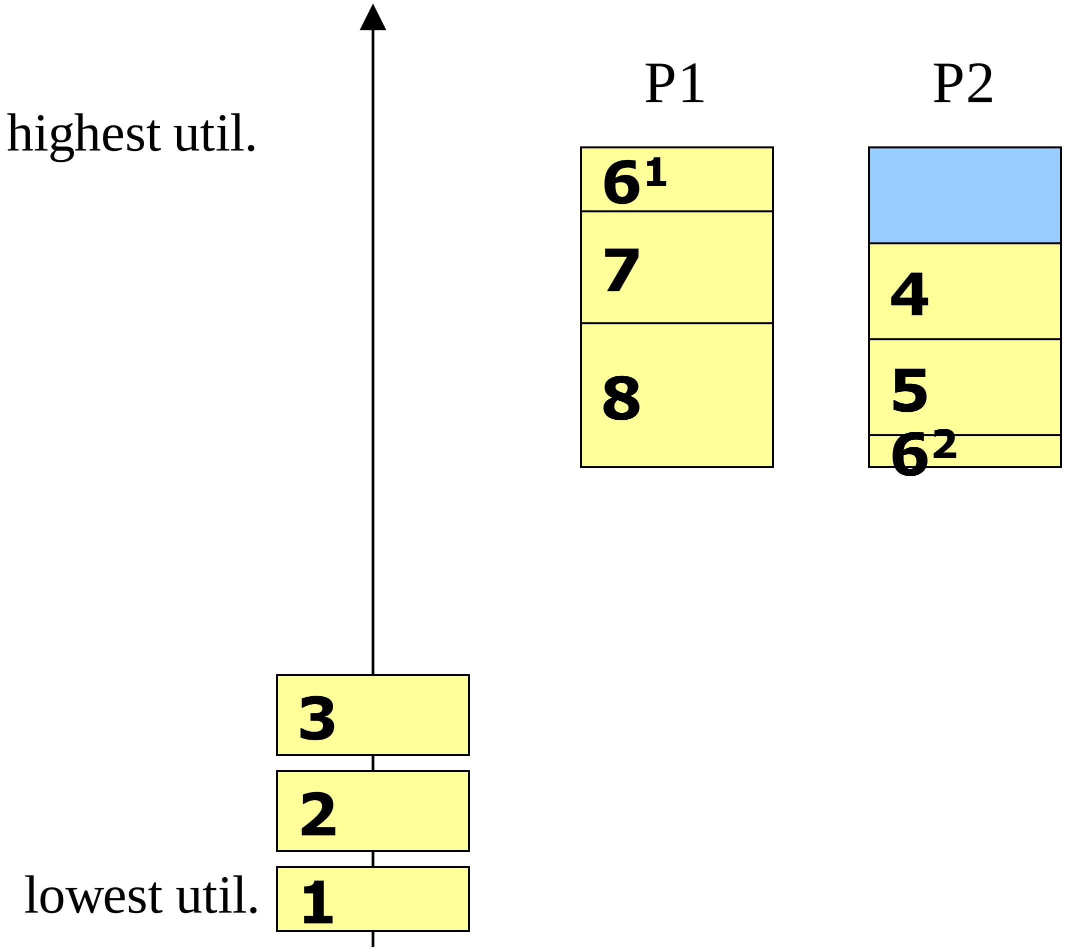
# Lakshmanan's Algorithm [ECRTS'09]

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



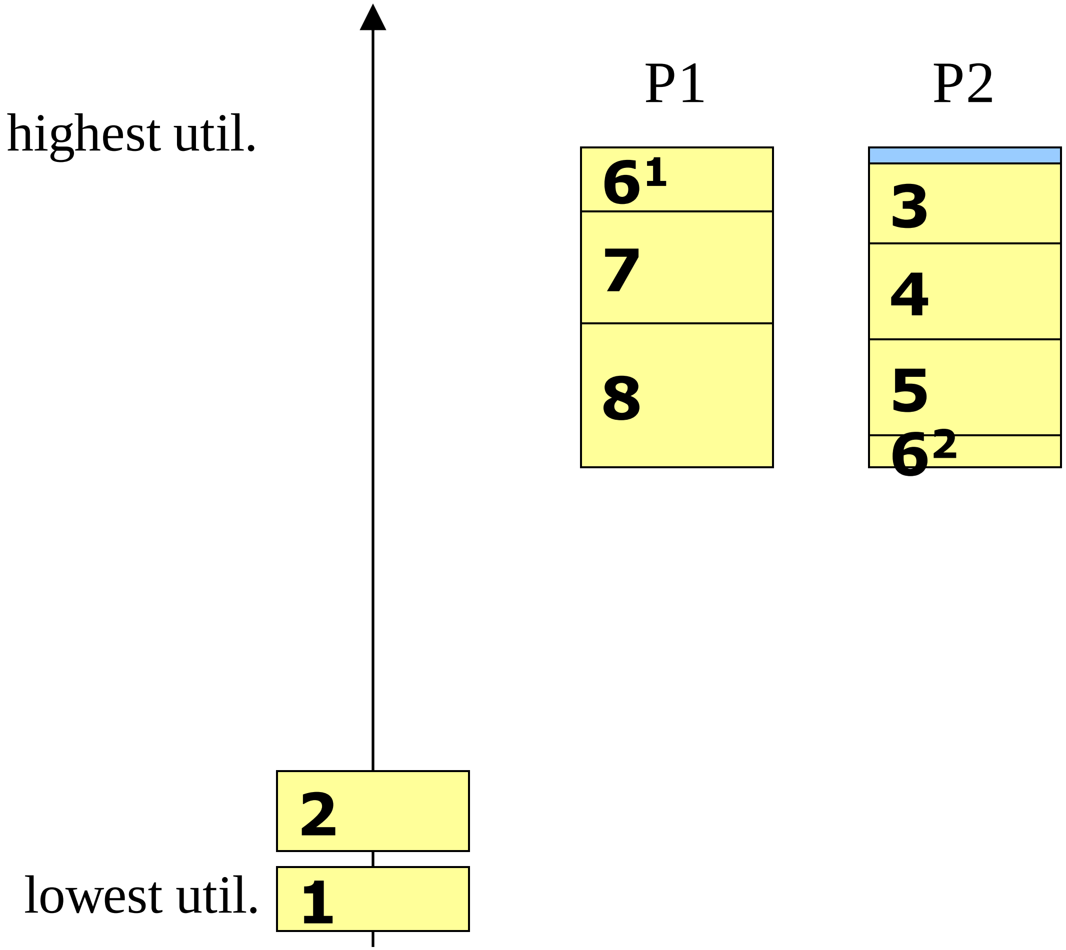
# Lakshmanan’s Algorithm [ECRTS’09]

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



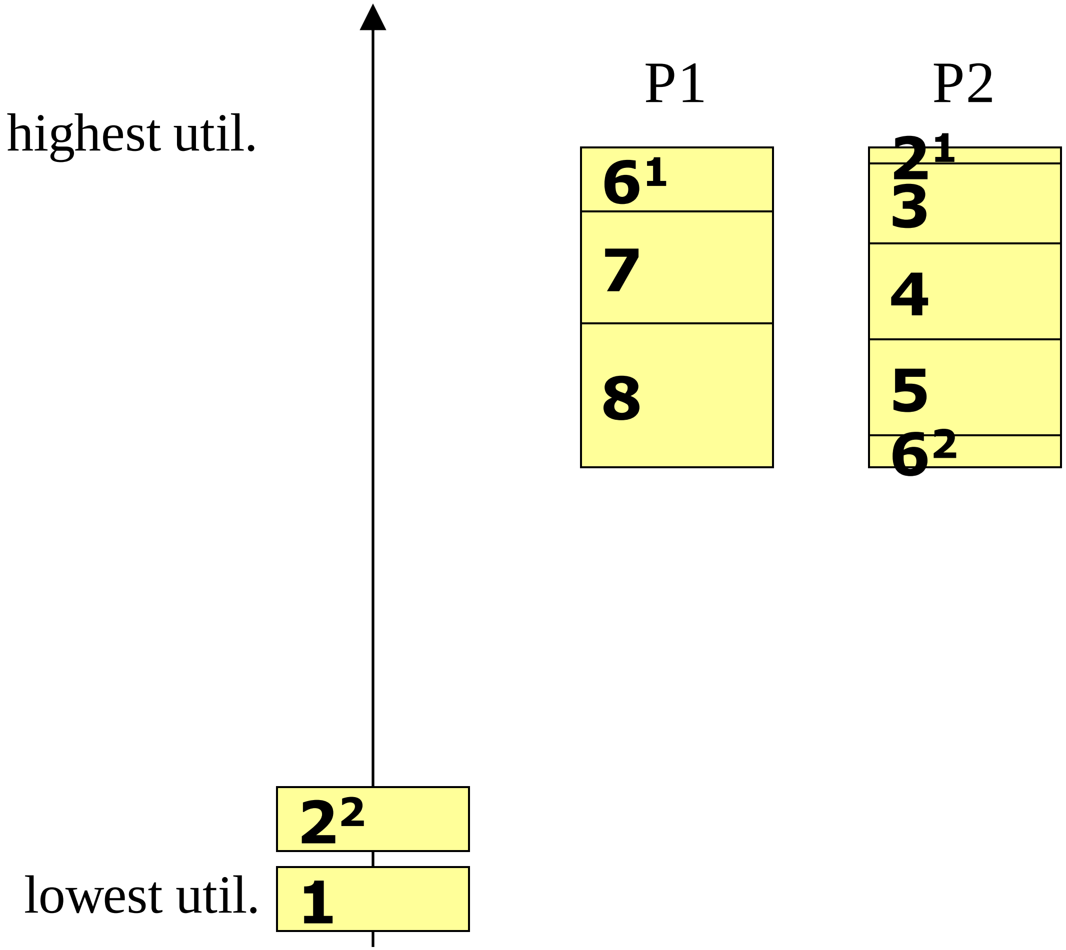
# Lakshmanan's Algorithm [ECRTS'09]

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



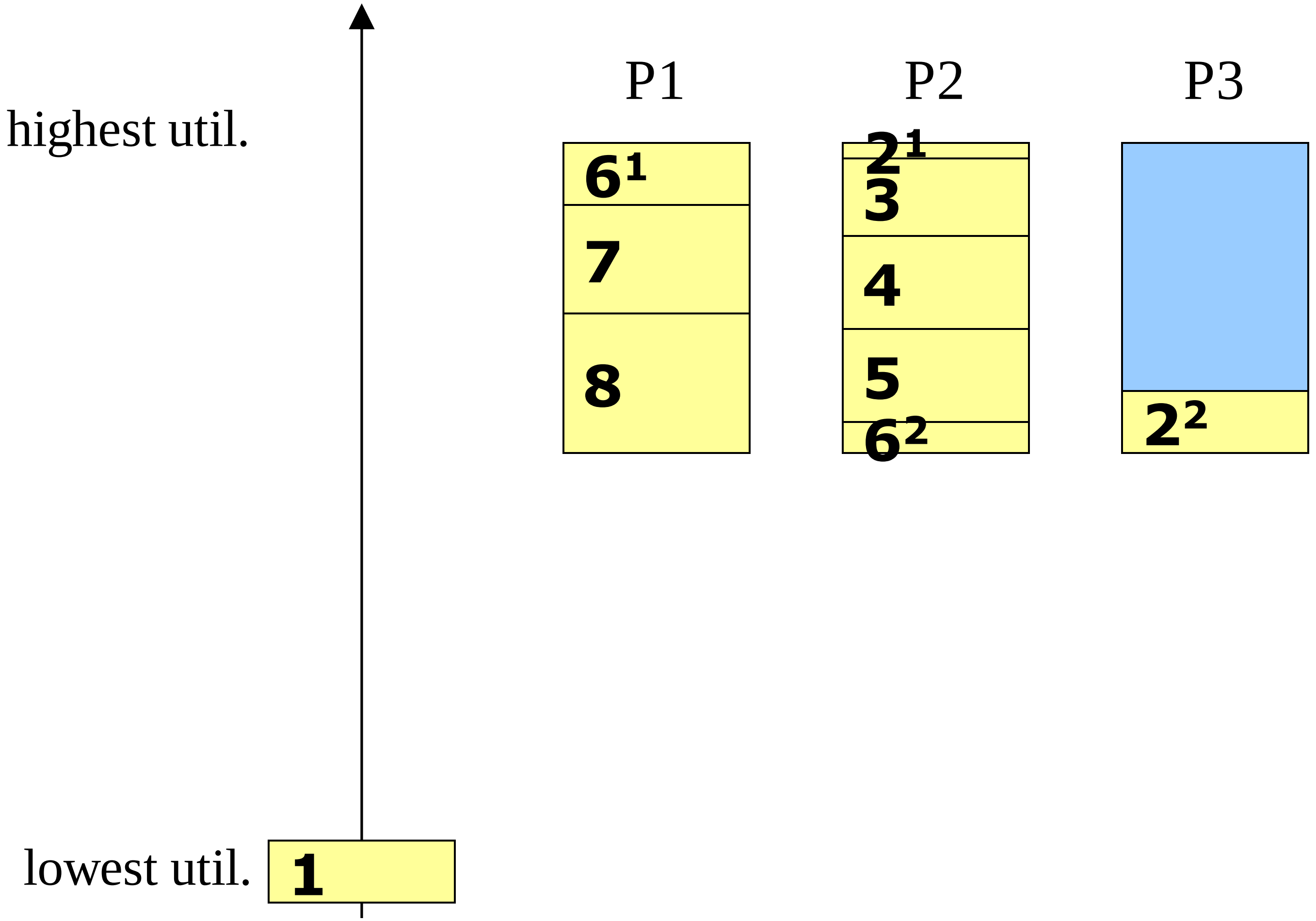
# Lakshmanan's Algorithm [ECRTS'09]

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



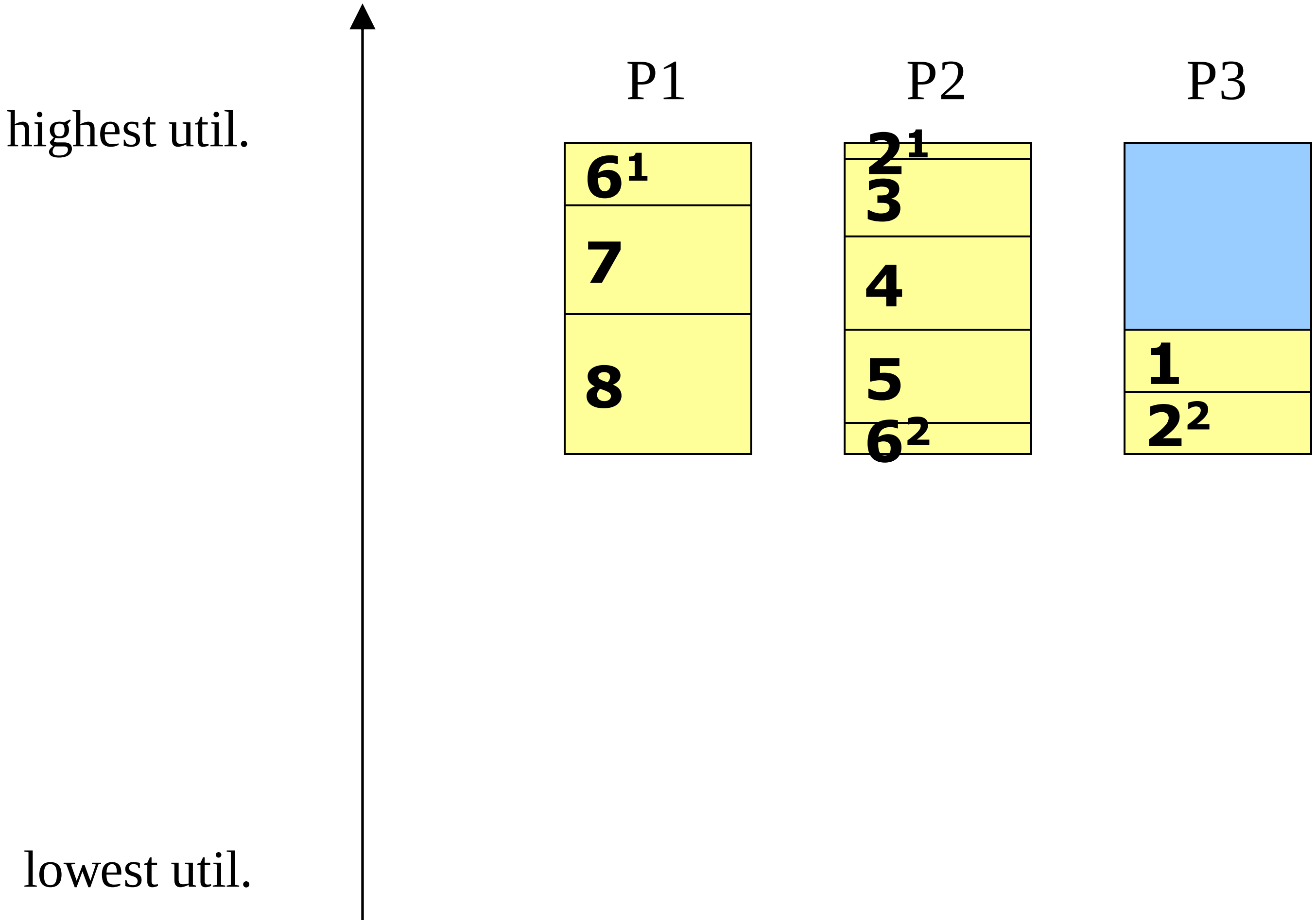
# Lakshmanan's Algorithm [ECRTS'09]

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



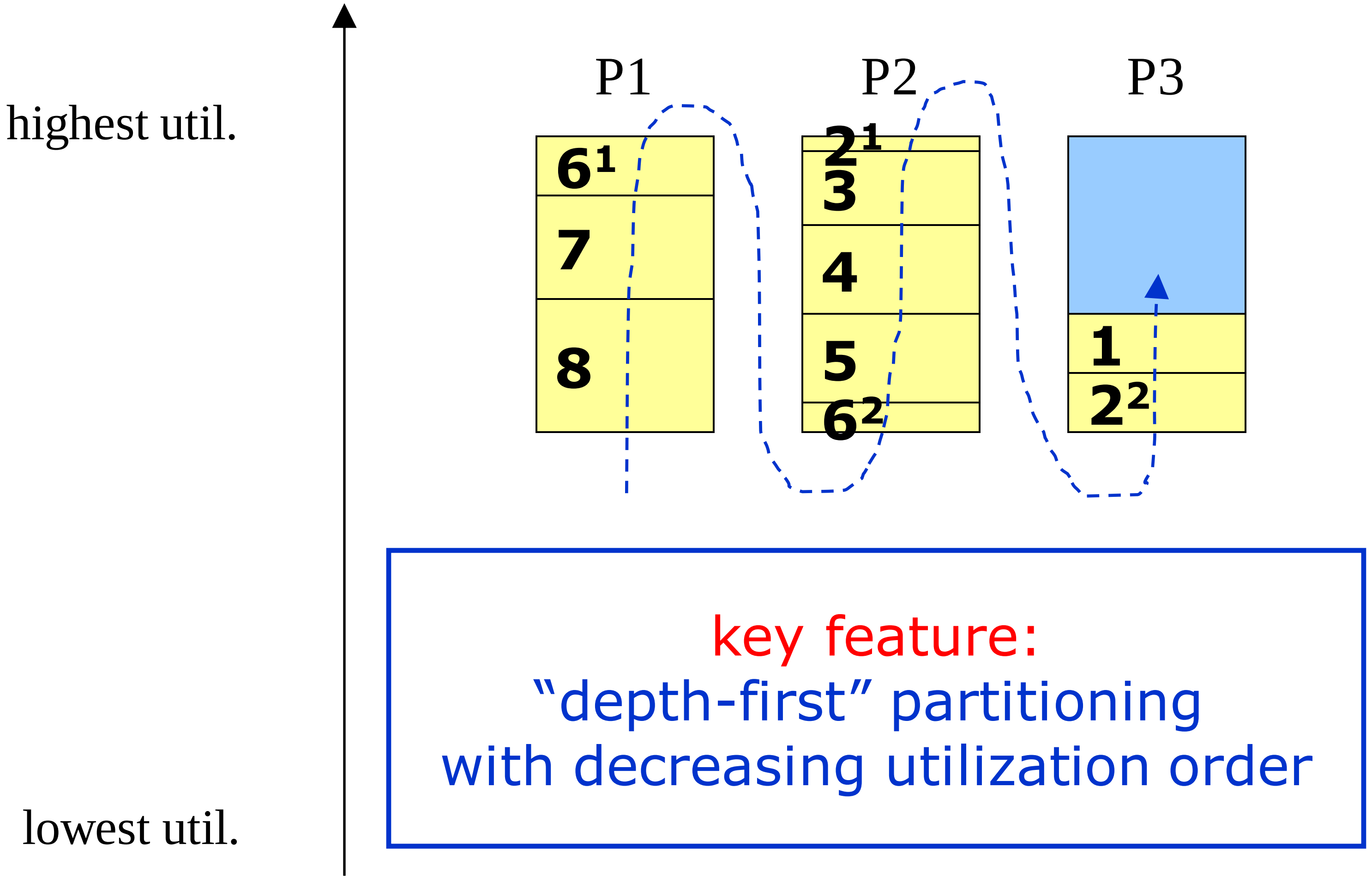
# Lakshmanan's Algorithm [ECRTS'09]

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



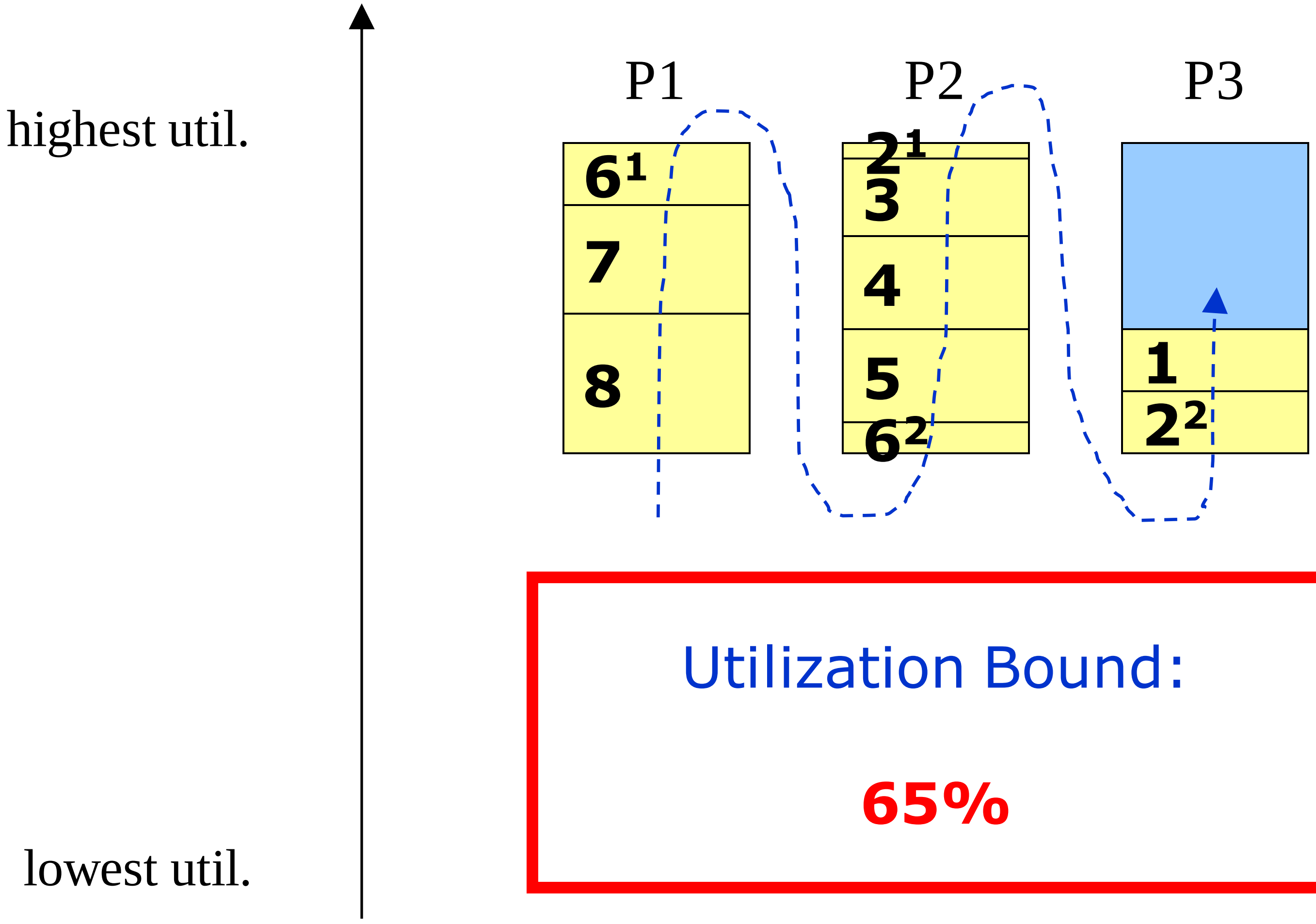
# Lakshmanan's Algorithm [ECRTS'09]

- 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




Utilization Bound:

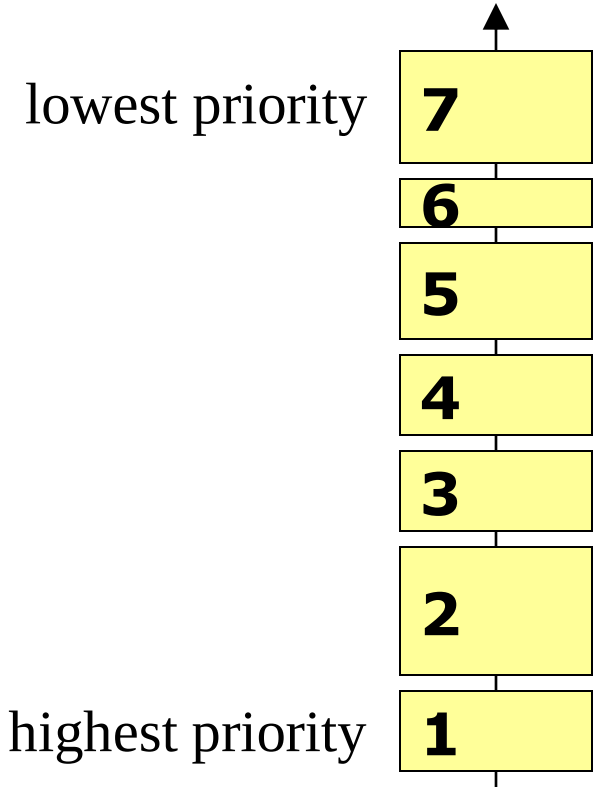
**65%**

# Our Algorithm
[RTAS10]

"width-first" partitioning
with increasing priority order

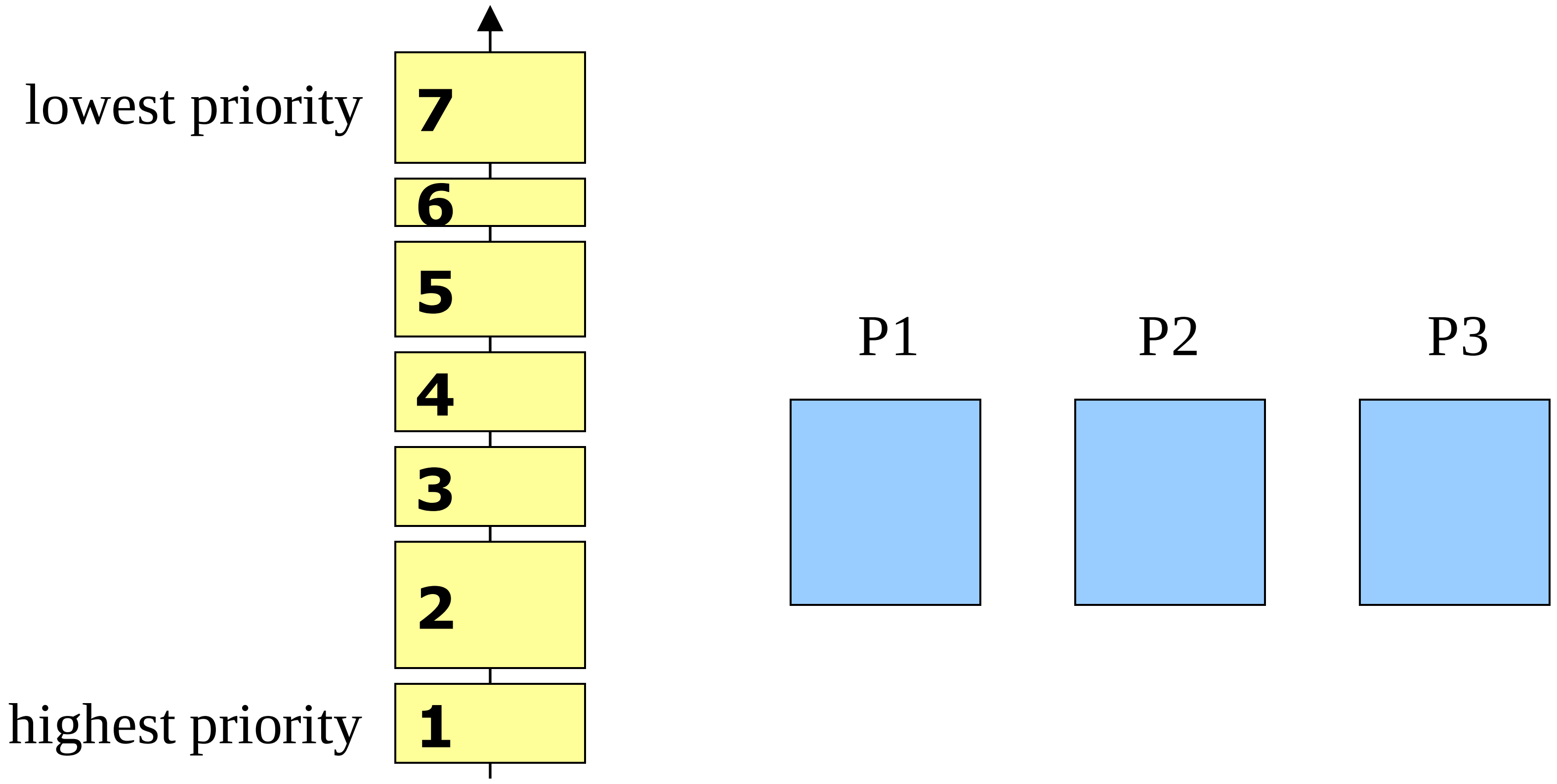
# Our Algorithm

- Sort all tasks in increasing priority order



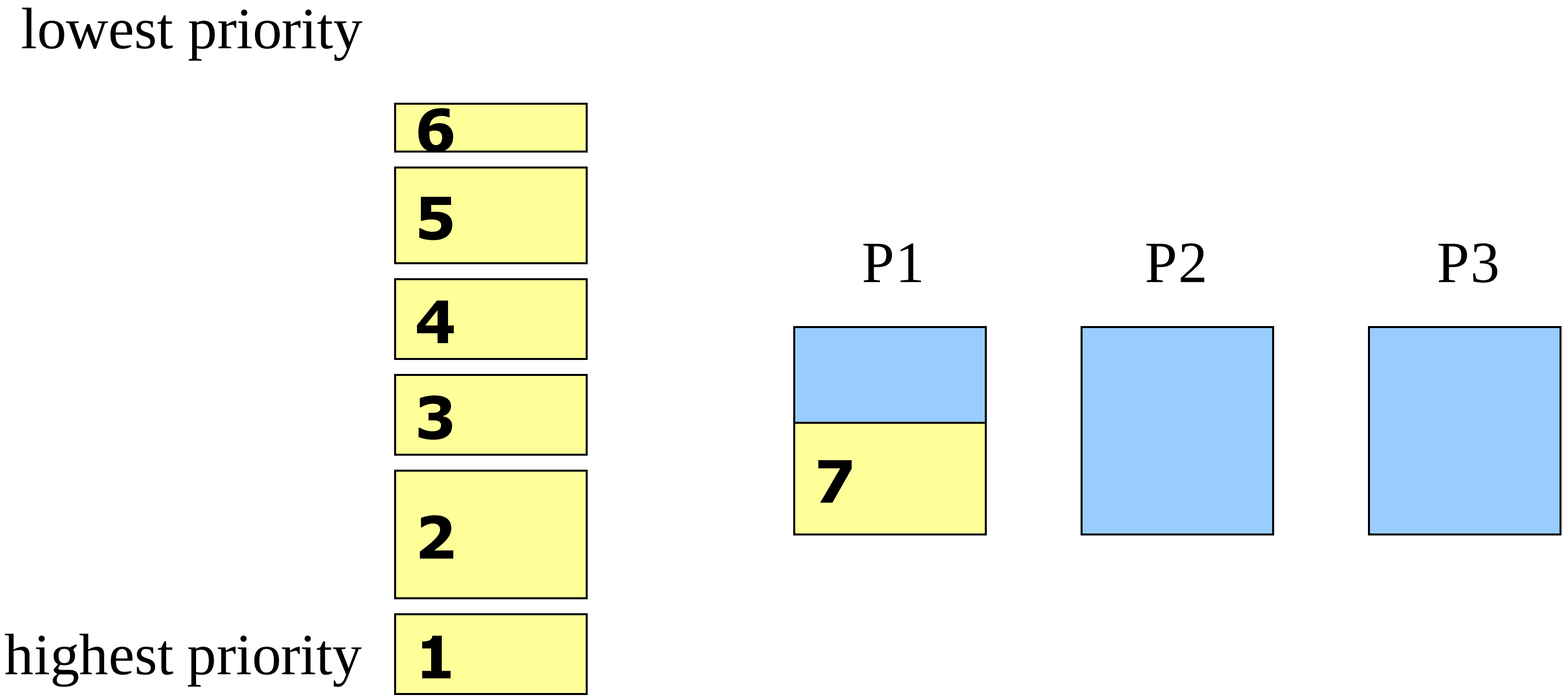
# Our Algorithm

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



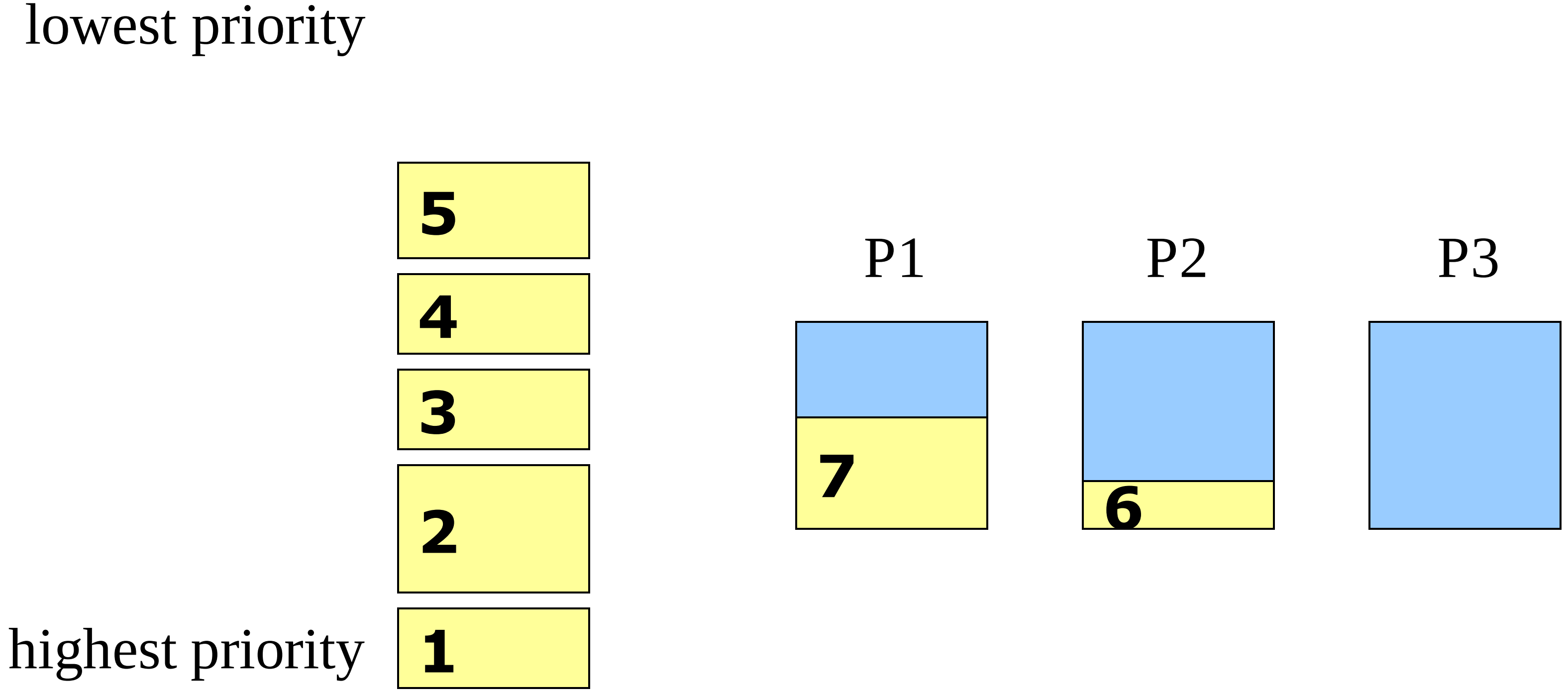
# Our Algorithm

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



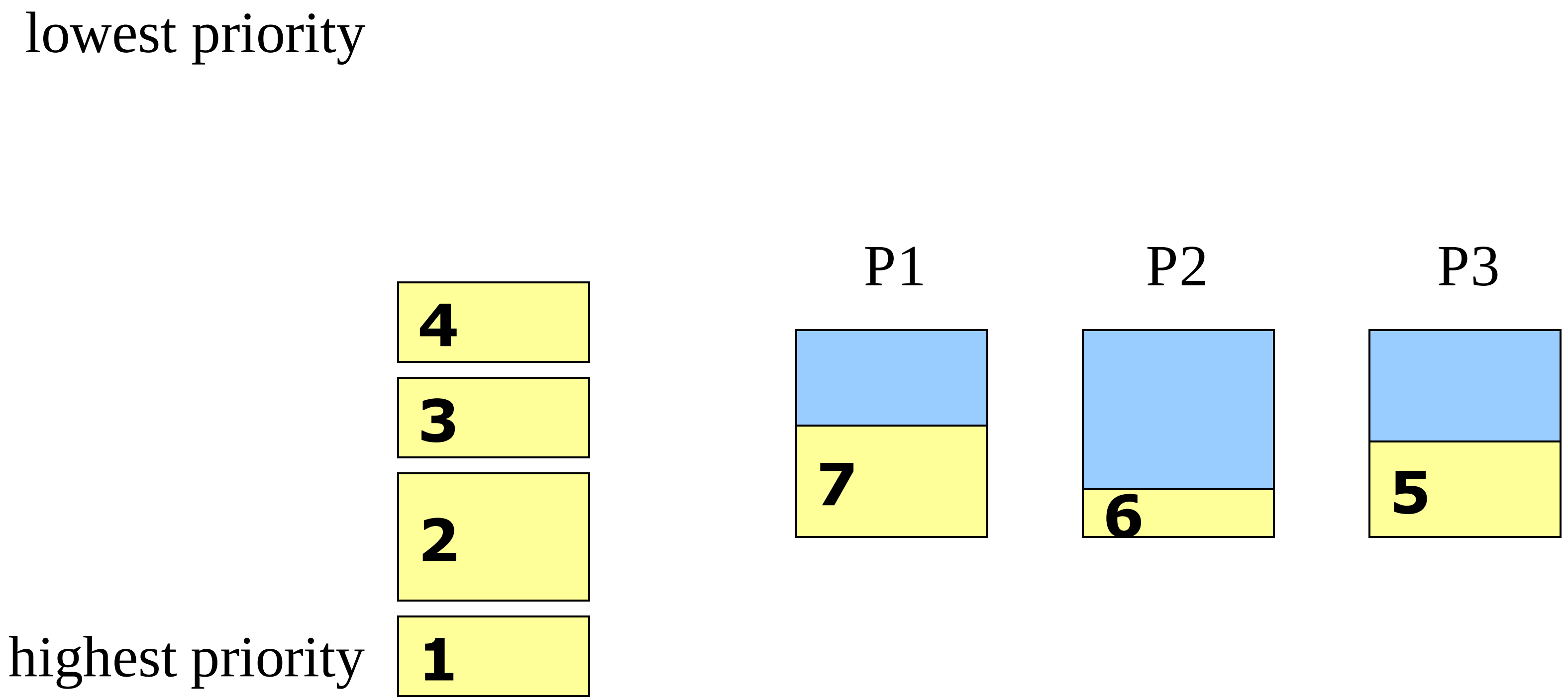
# Our Algorithm

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



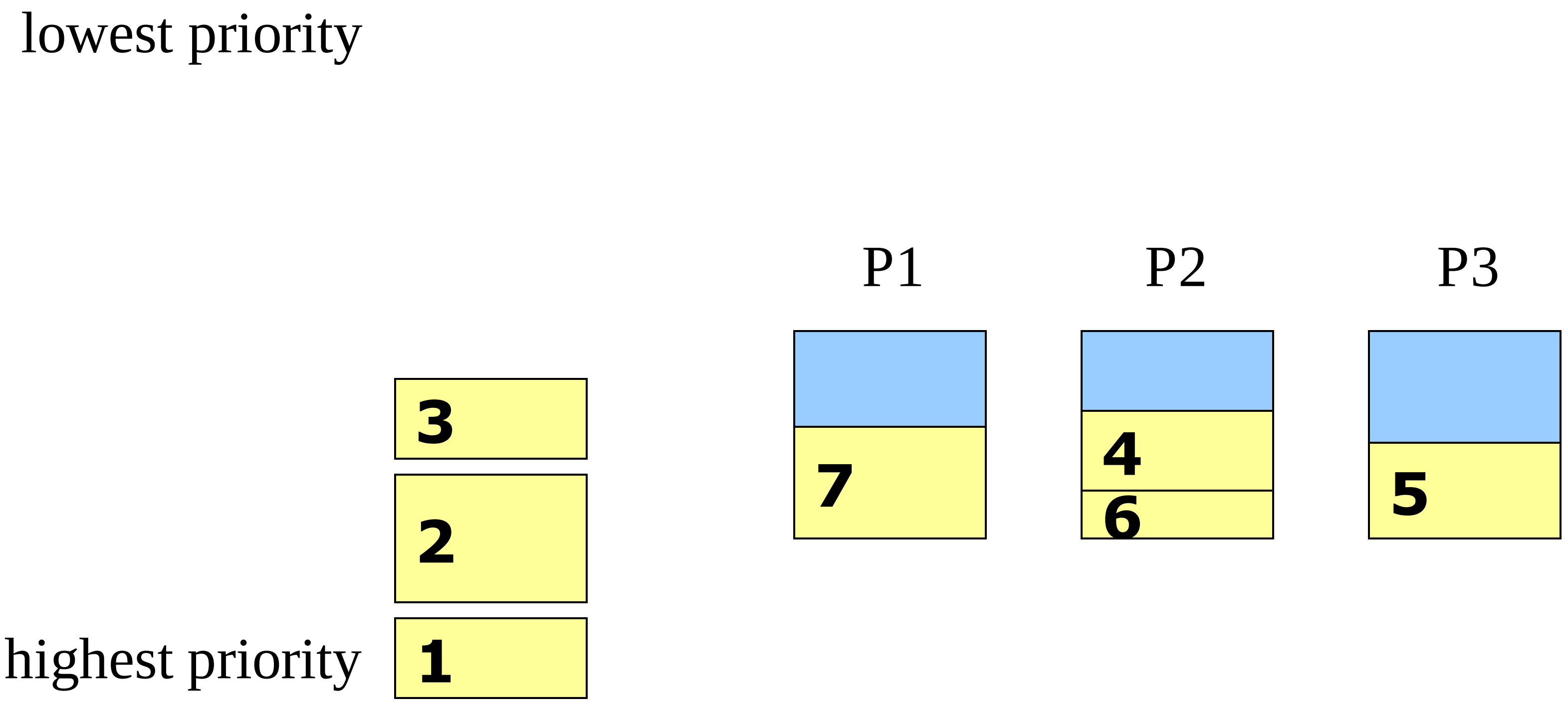
# Our Algorithm

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



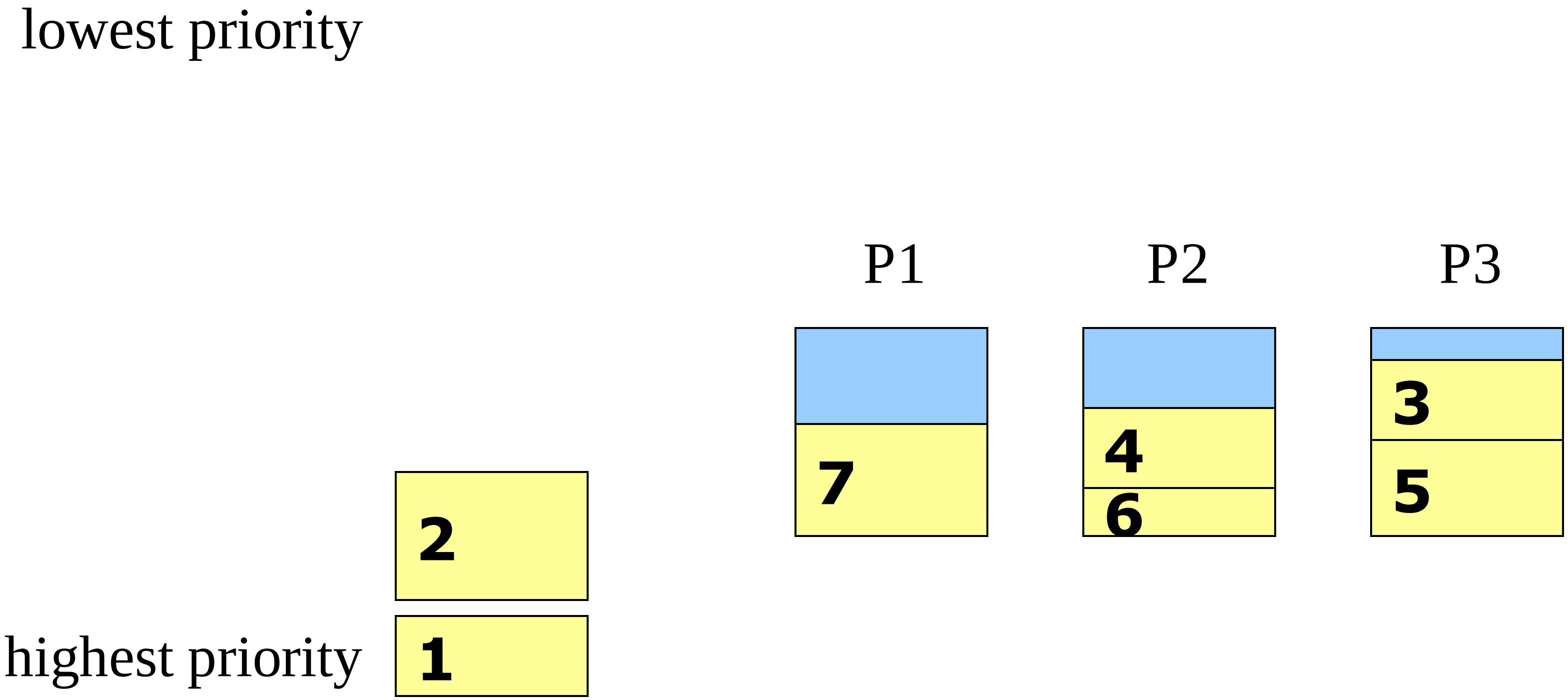
# Our Algorithm

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



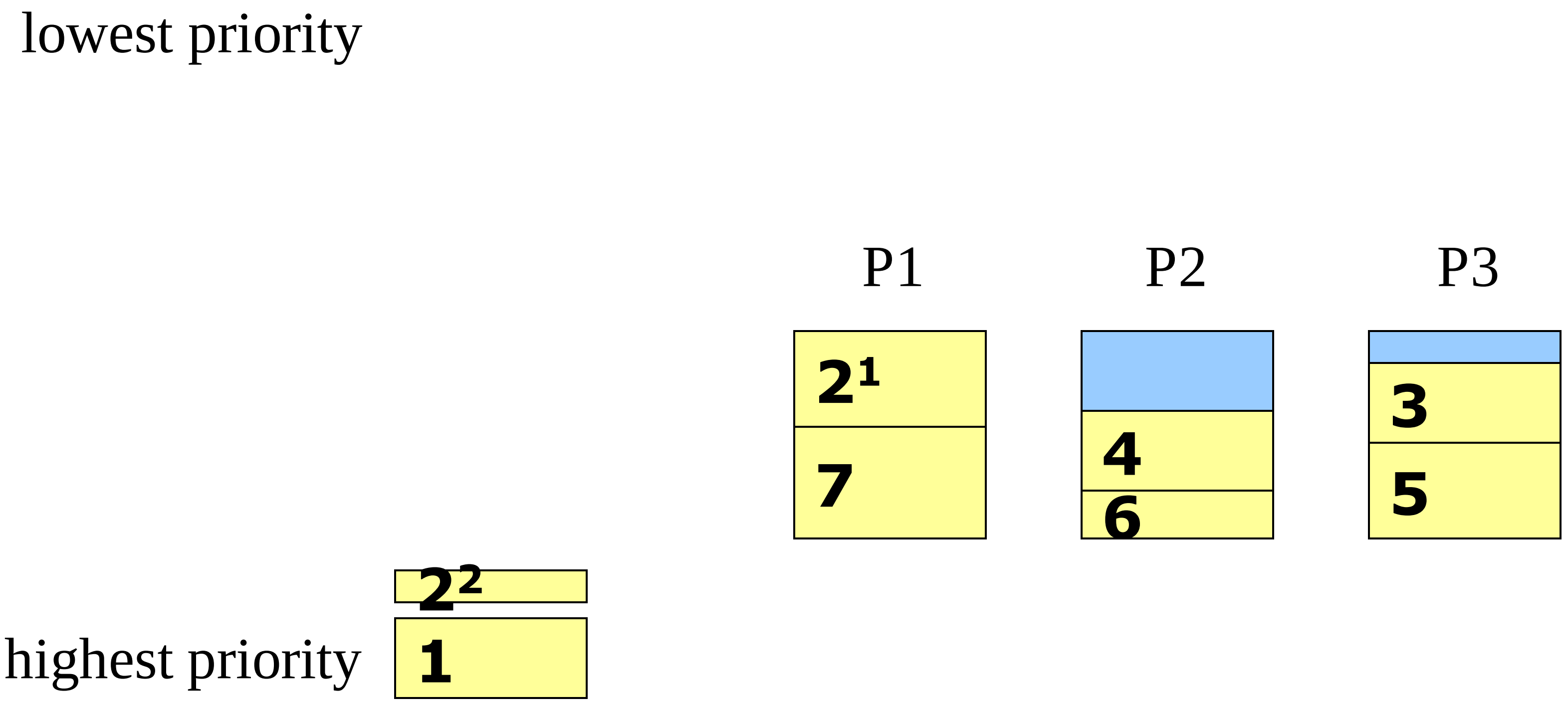
# Our Algorithm

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



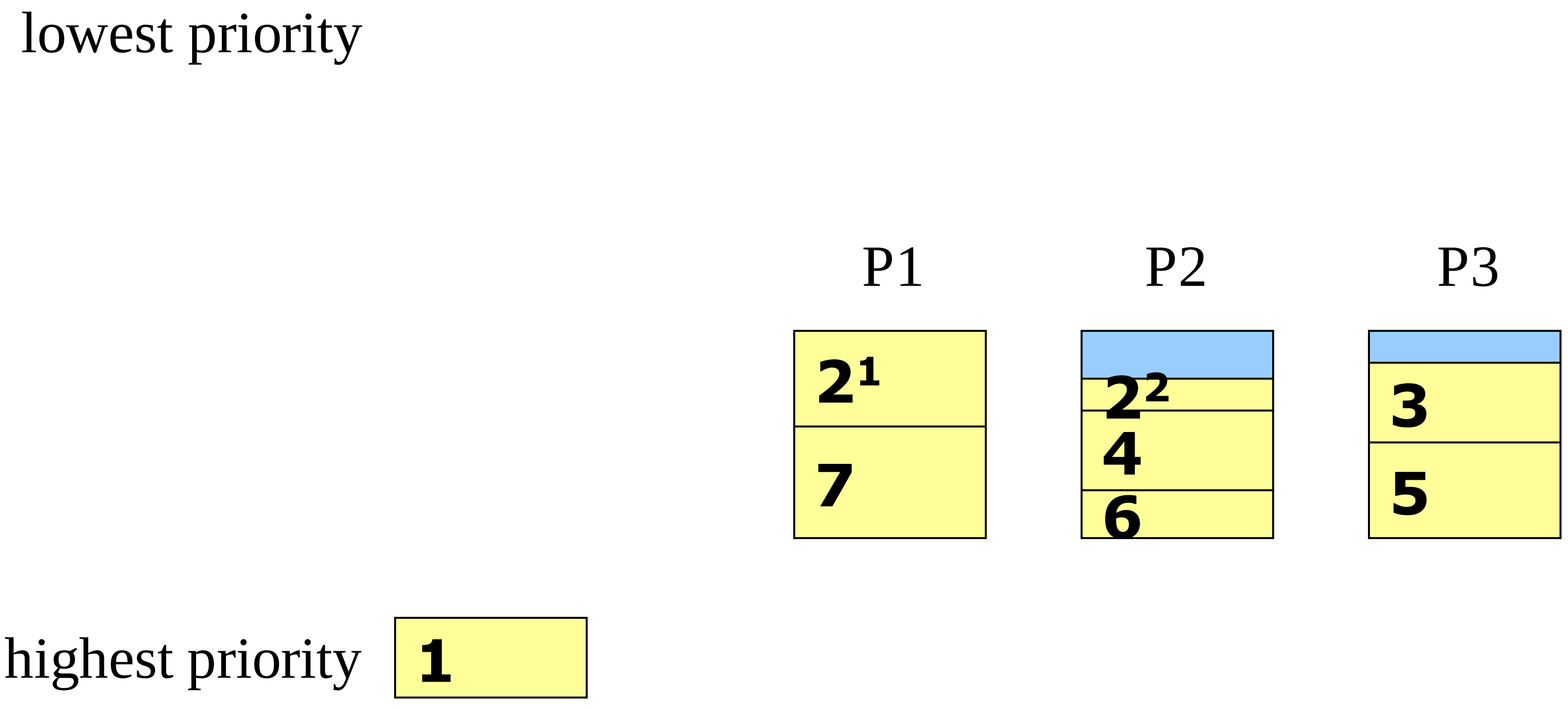
# Our Algorithm

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



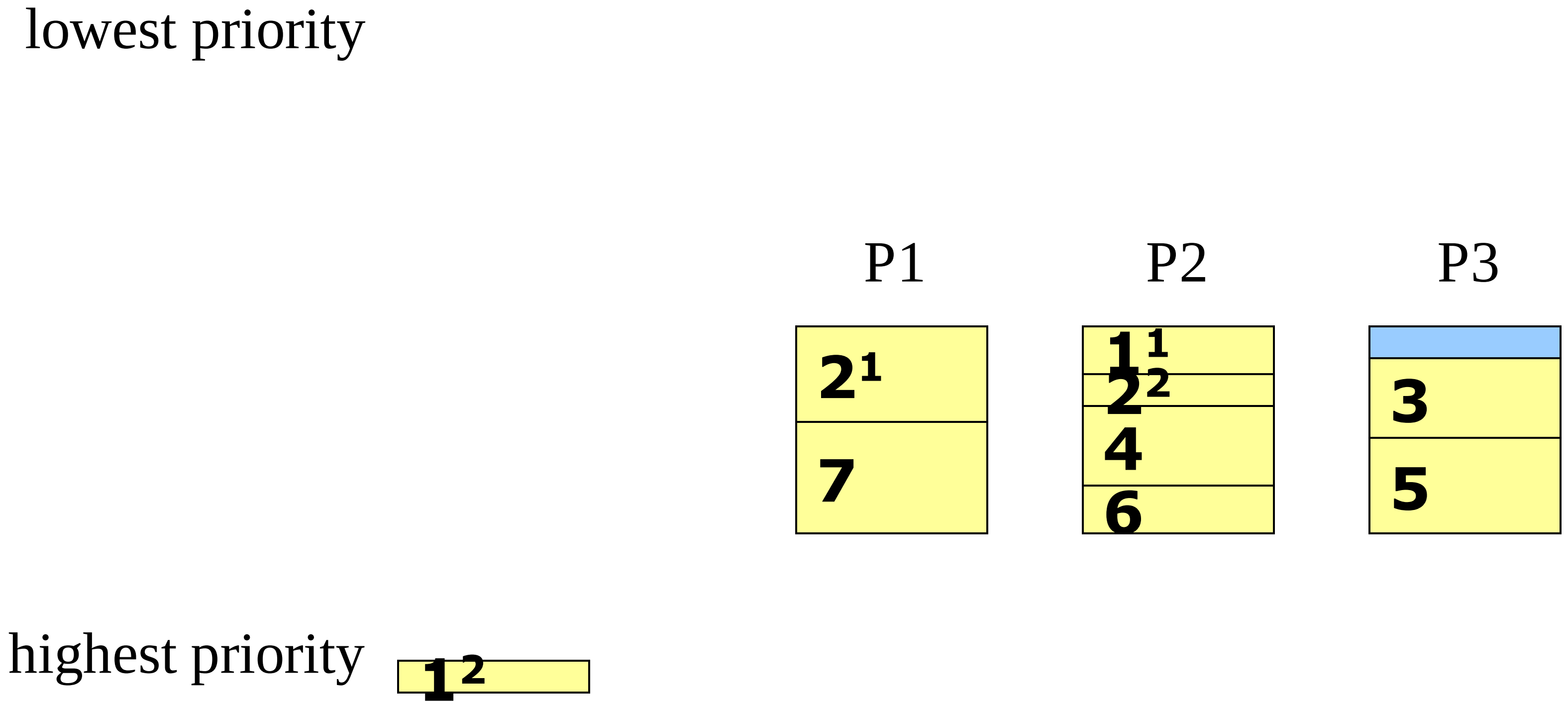
# Our Algorithm

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



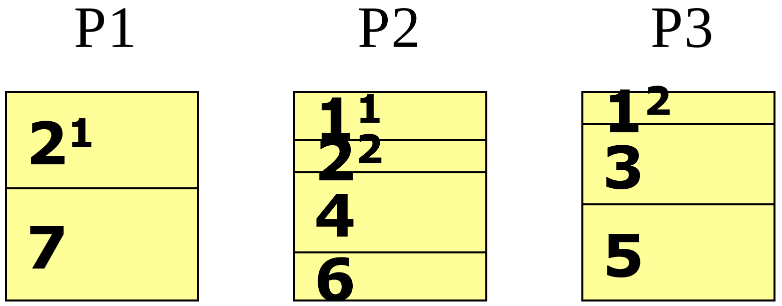
# Our Algorithm

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



# Our Algorithm

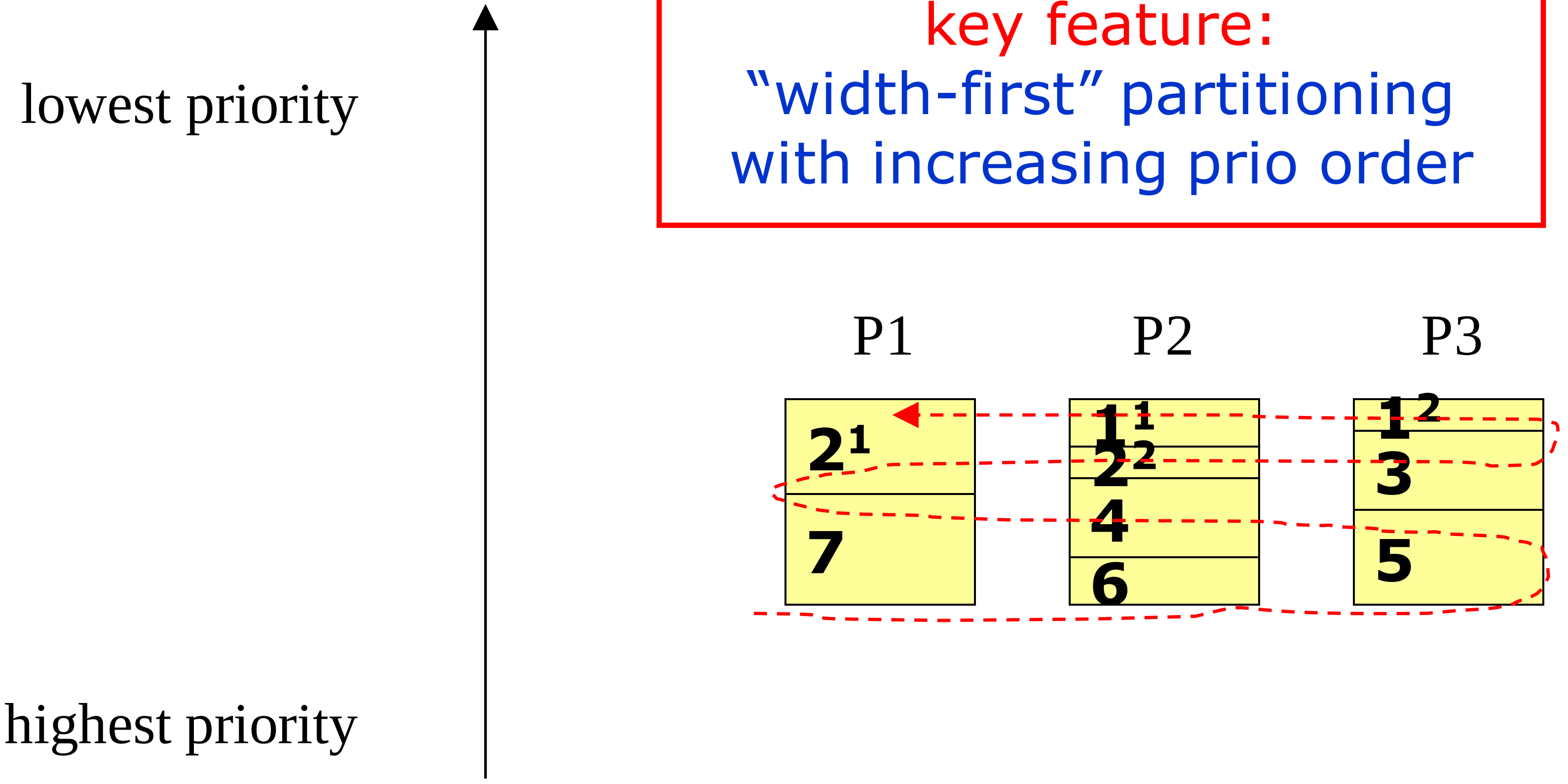
- Select the processor on which the assigned utilization is the lowest



# Our Algorithm

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

key feature:
"width-first" partitioning
with increasing prio order

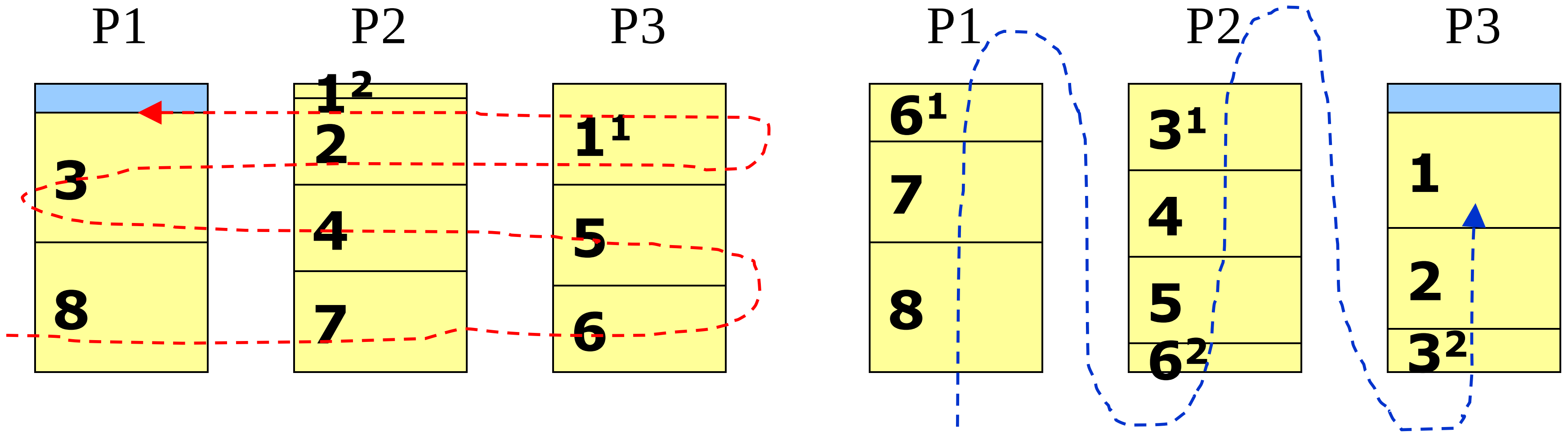


# Comparison

Why is our algorithm better?

Ours: width-first & increasing priority order

Previous: depth-first & decreasing utilization order

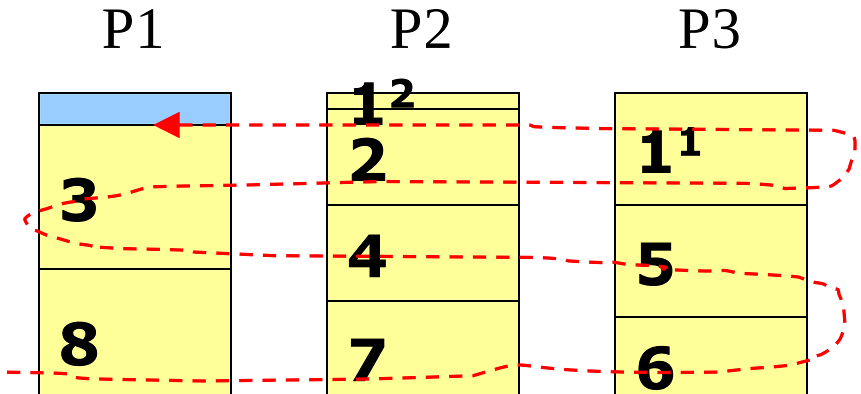


# Comparison

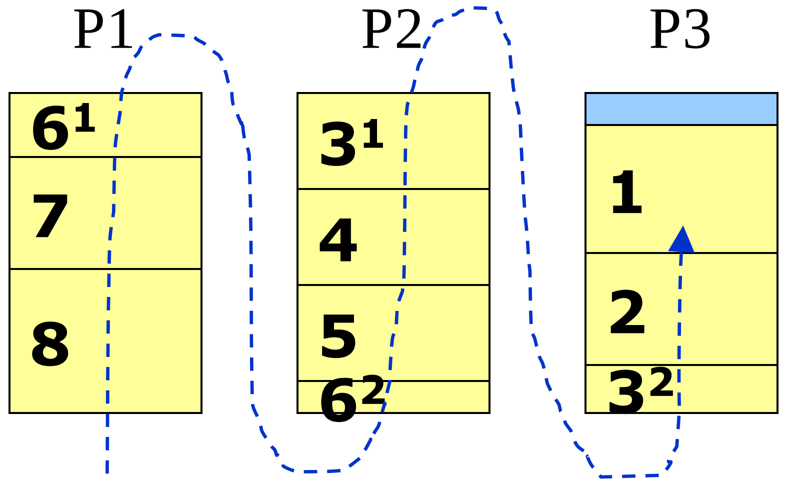
**Why is our algorithm better?**

By our algorithm split tasks generally have higher priorities

Ours: width-first
& increasing priority order

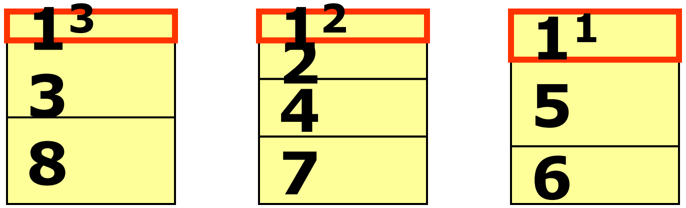



Previous: depth-first
& decreasing utilization order



# Split Task

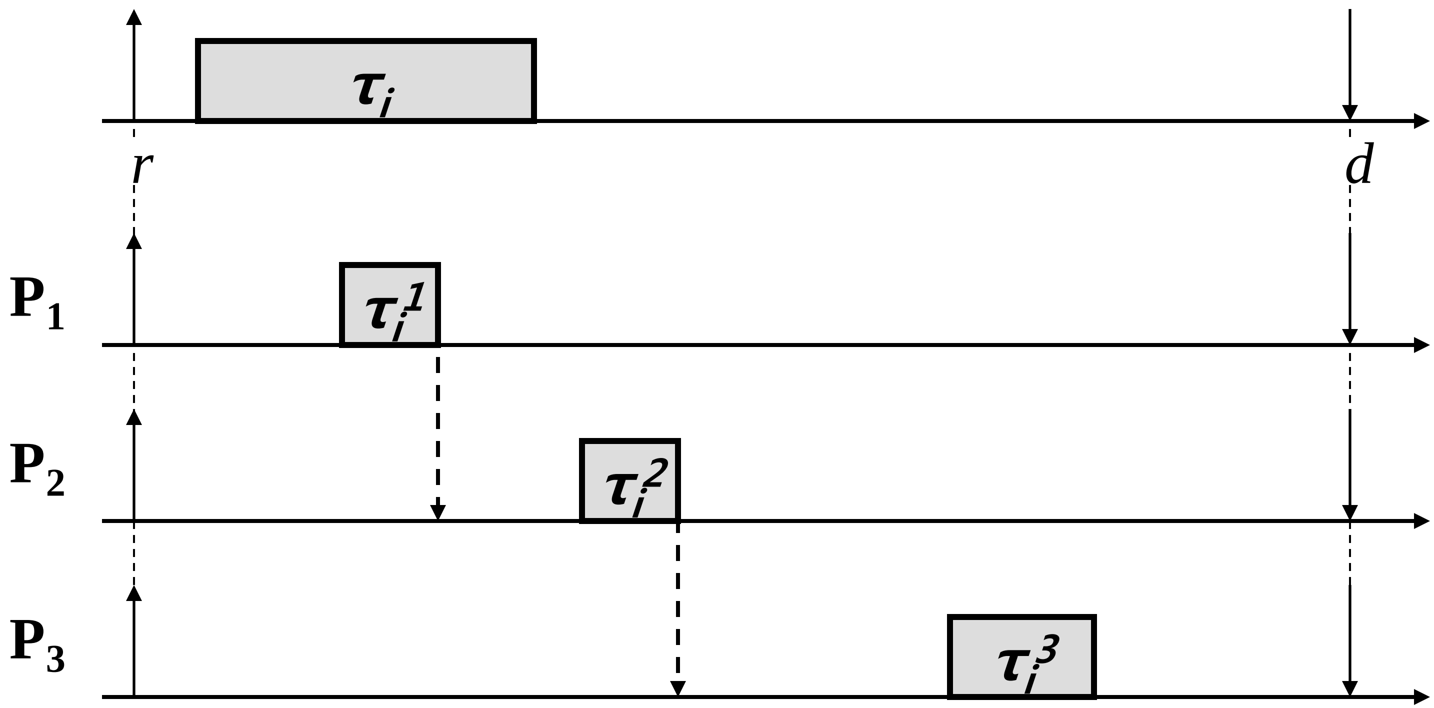
- Consider an extreme scenario:
  - suppose each subtask has the highest priority
  - schedulable anyway, we do not need to worry about their deadlines




- The difficult case is when the tail task is not on the top
  - the key point is to ensure the tail task is schedulable

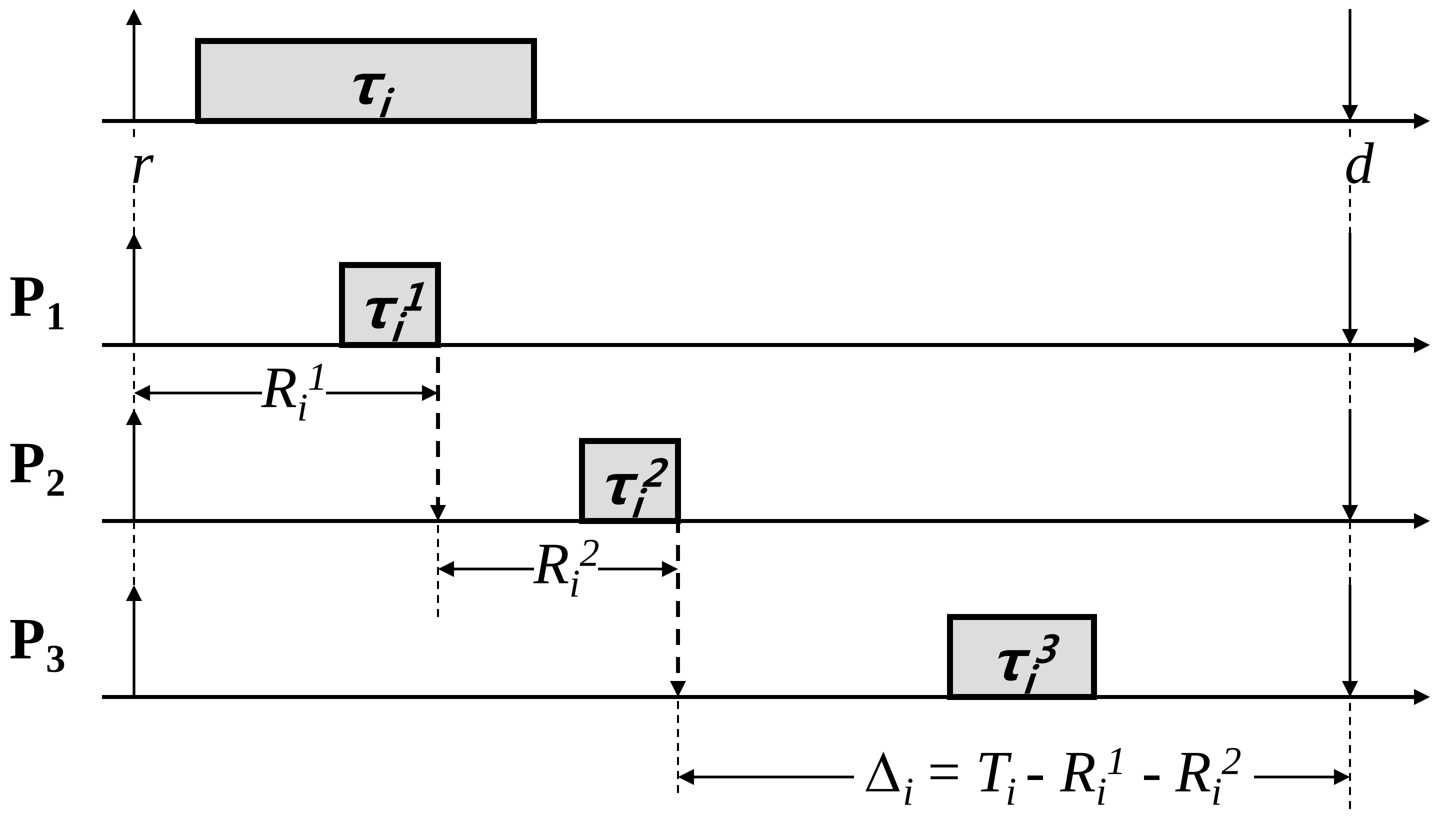
# Split Task

- Subtasks should execute in the correct order



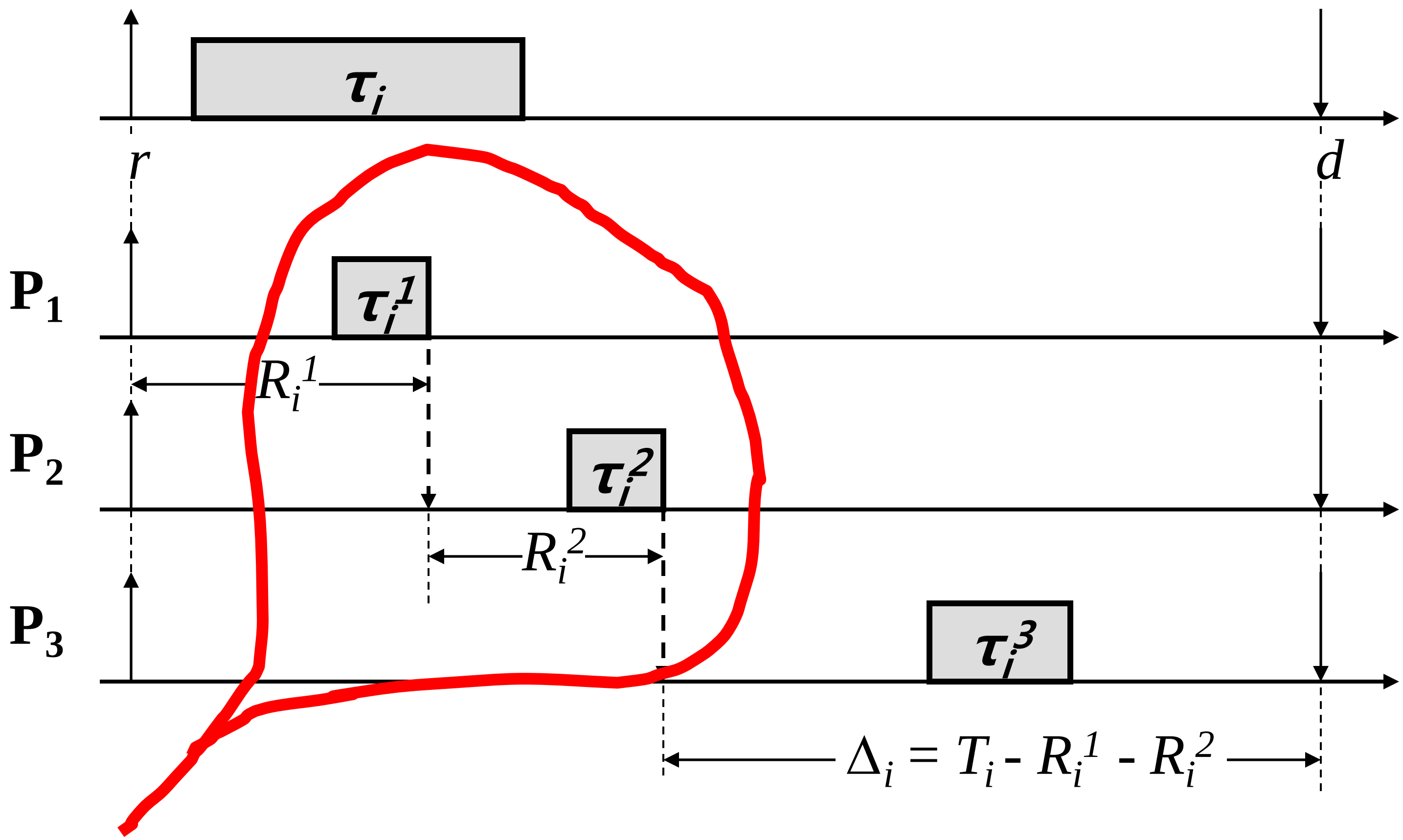
# Split Task

- Subtasks get "shorter deadlines"



# Split Task

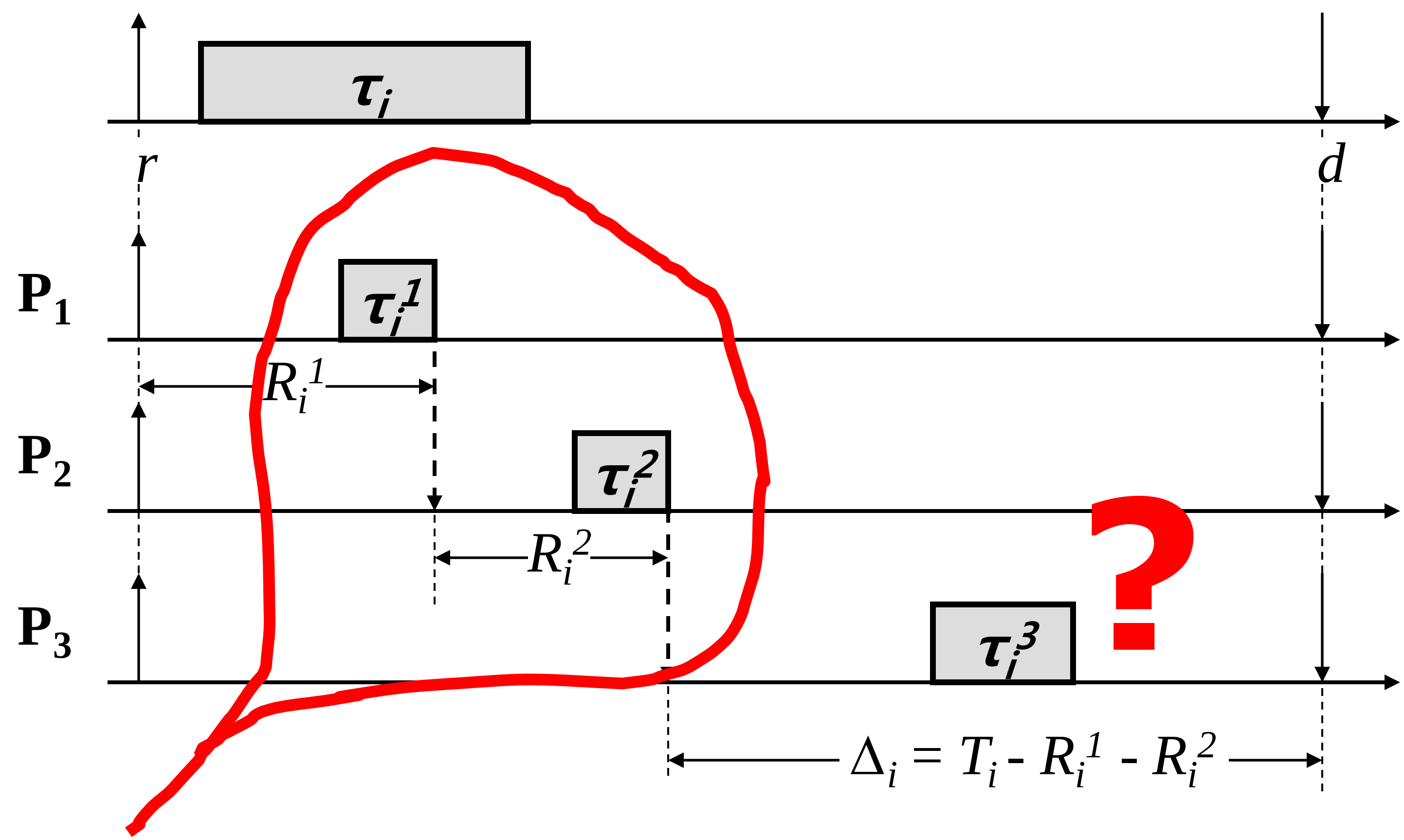
- Subtasks should execute in the correct order




**These two are on the top: no problem with schedulability**

# Split Task

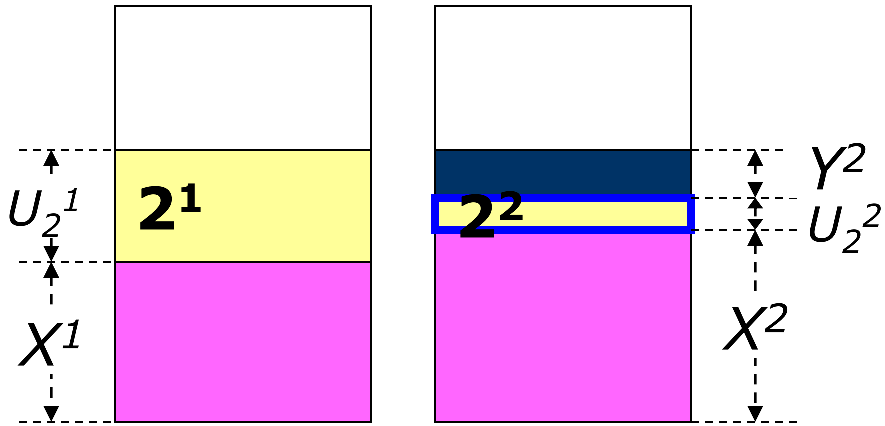
- Subtasks should execute in the correct order




These two are on the top: no problem with schedulability

# Why the tail task is schedulable?

The typical case: two CPUs and task 2 is split to two sub-tasks




As we always select the CPU with the lowest load assigned, we know

$$Y^2 + U_2^2 <= U_2^1$$

$$\Downarrow$$

$$Y^2 <= U_2^1 - U_2^2$$

That is, the "blocking factor" for the tail task is bounded.

# Theorem

For a task set in which each task $\tau_i$ satisfies

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

we have

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

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

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

# Theorem

For a task set in which each task $\tau_i$ satisfies

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

get rid of this constraint

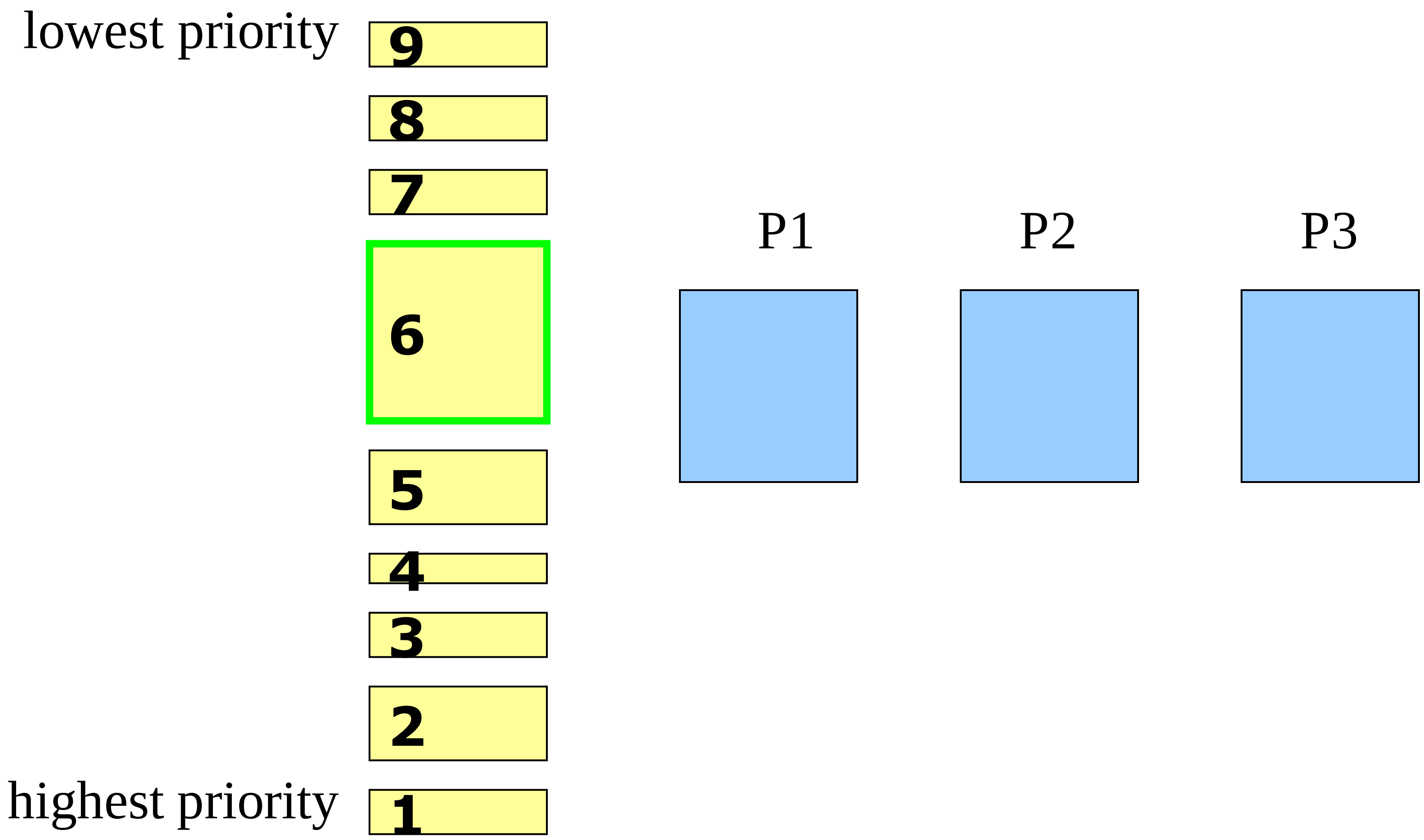
we have

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

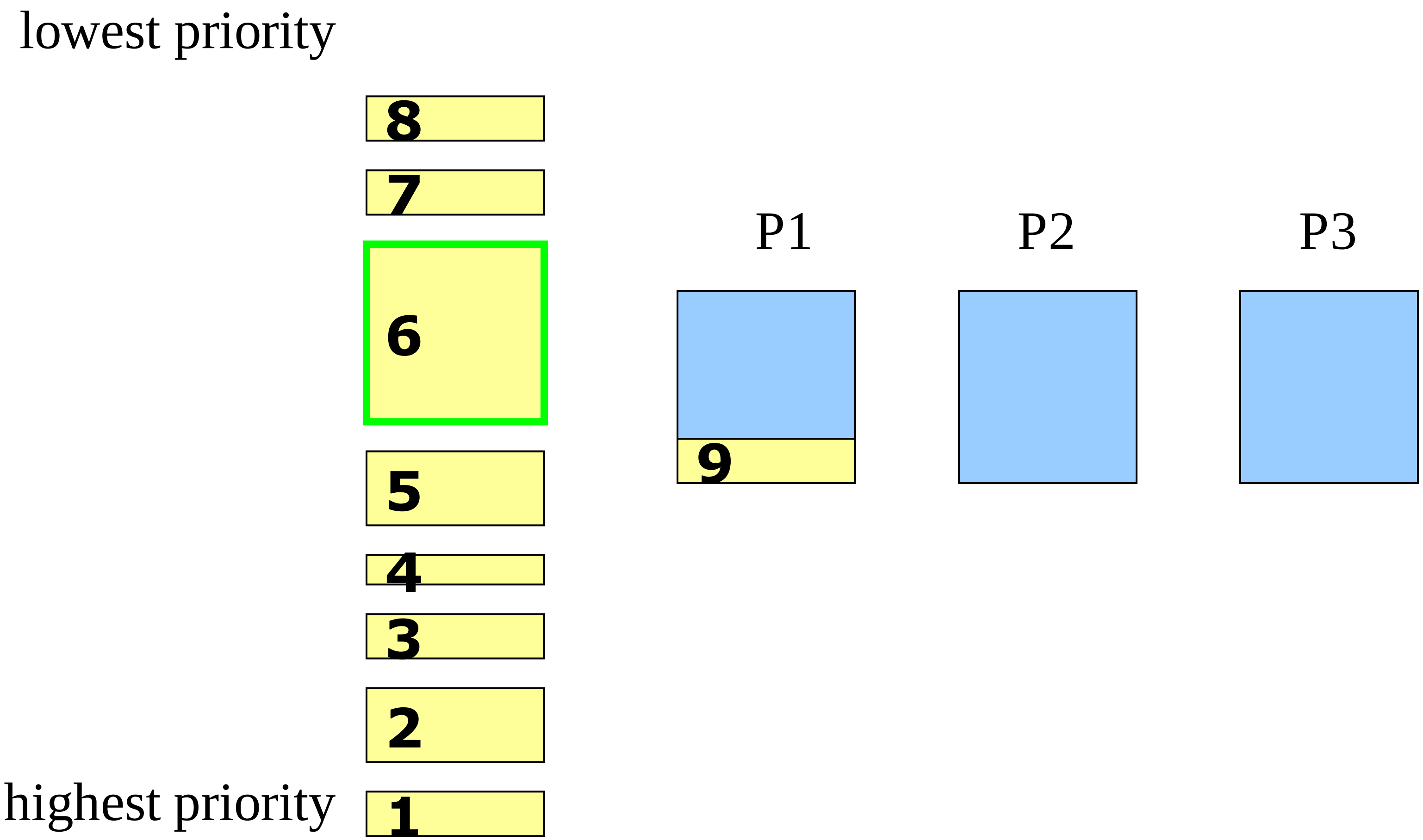
$$\Rightarrow \quad \text{the task set is schedulable}$$

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

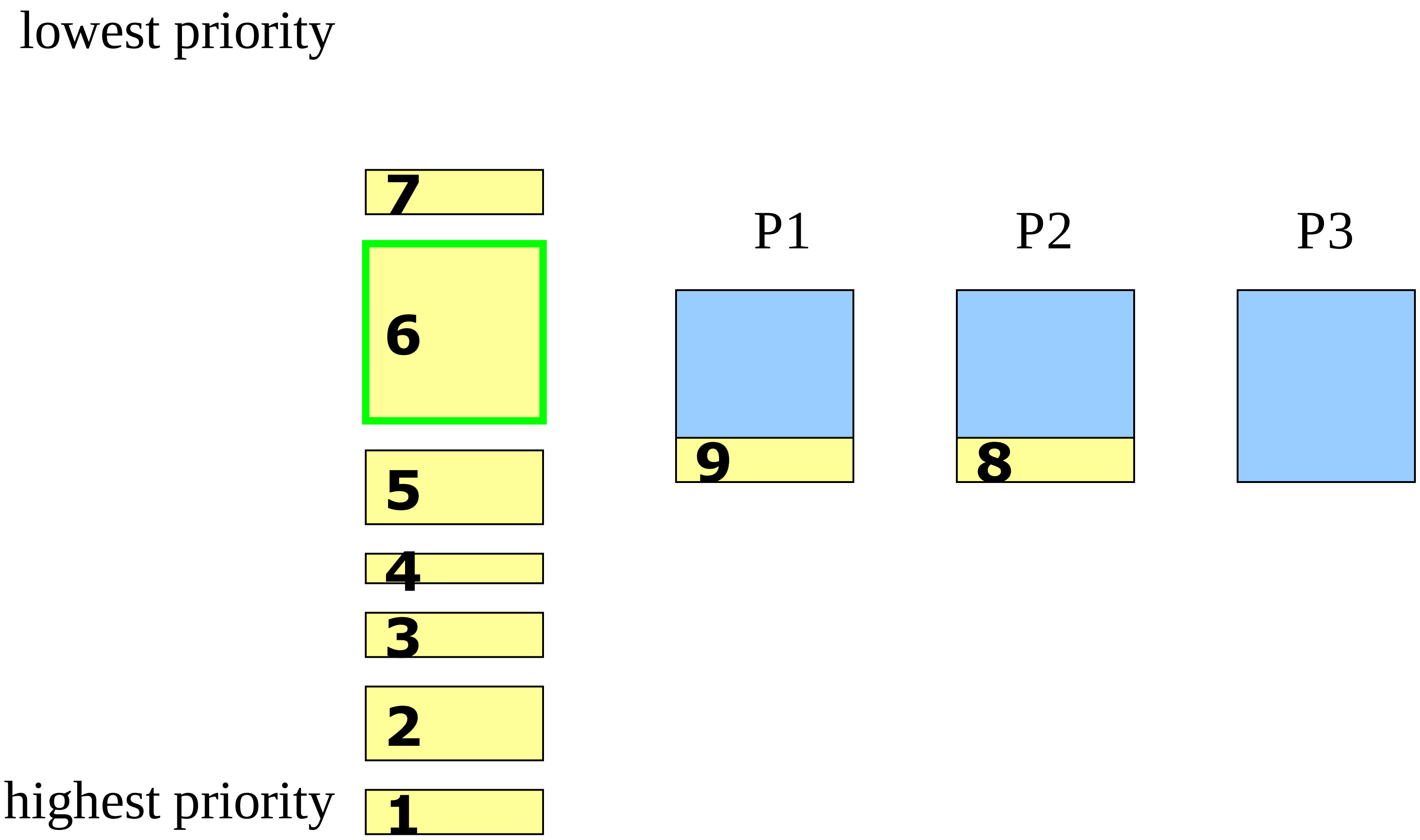
# Problem of Heavy Tasks



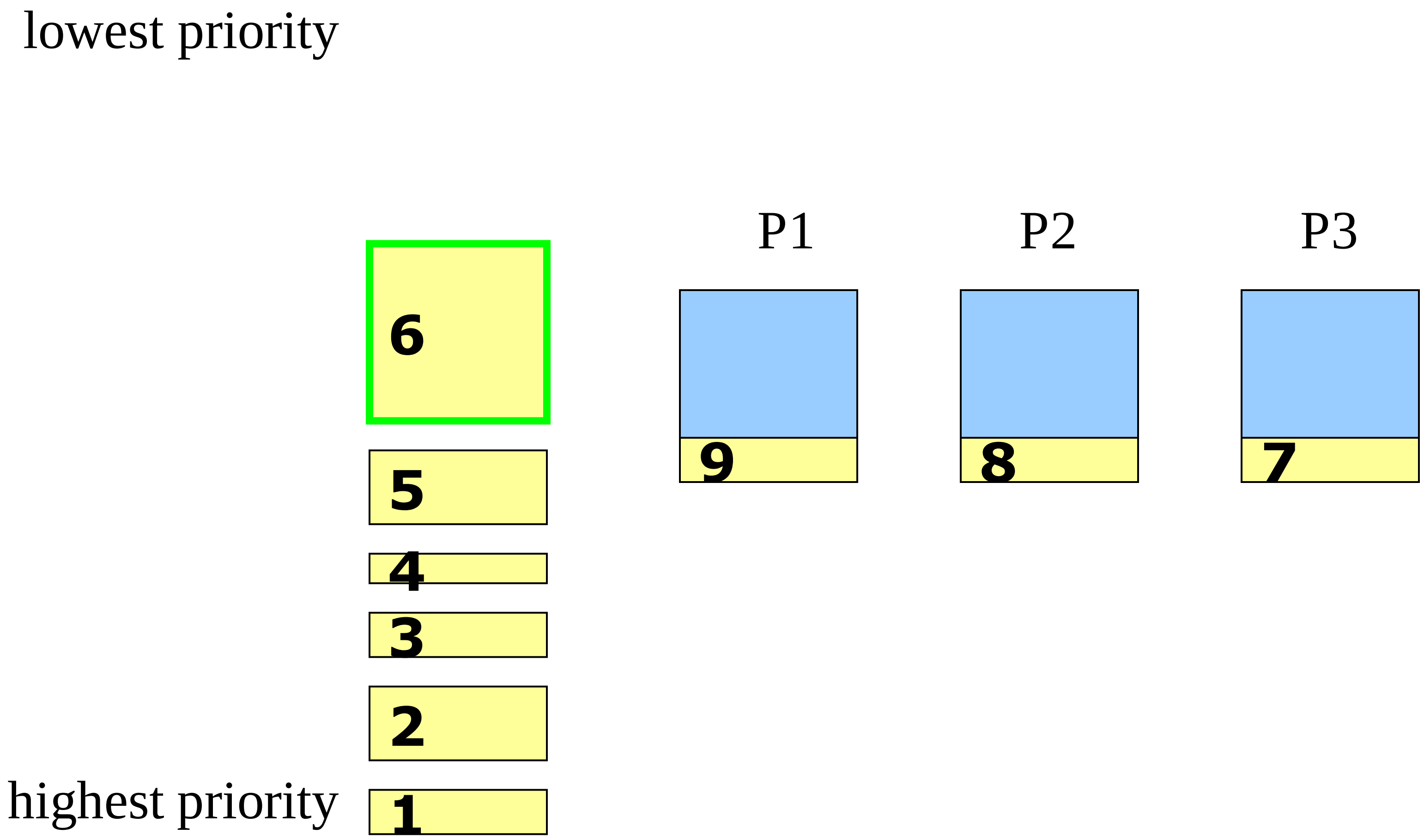
# Problem of Heavy Tasks



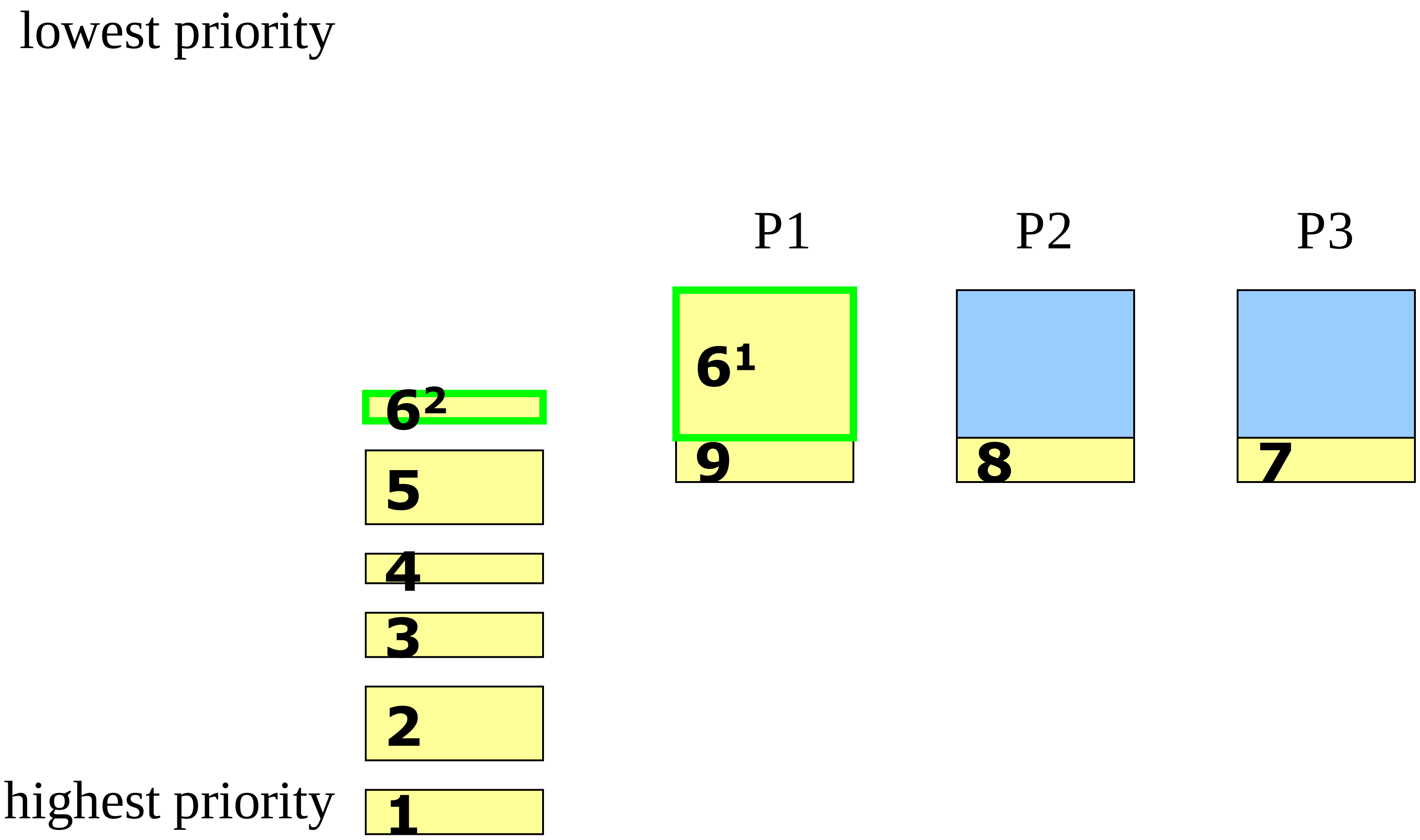
# Problem of Heavy Tasks



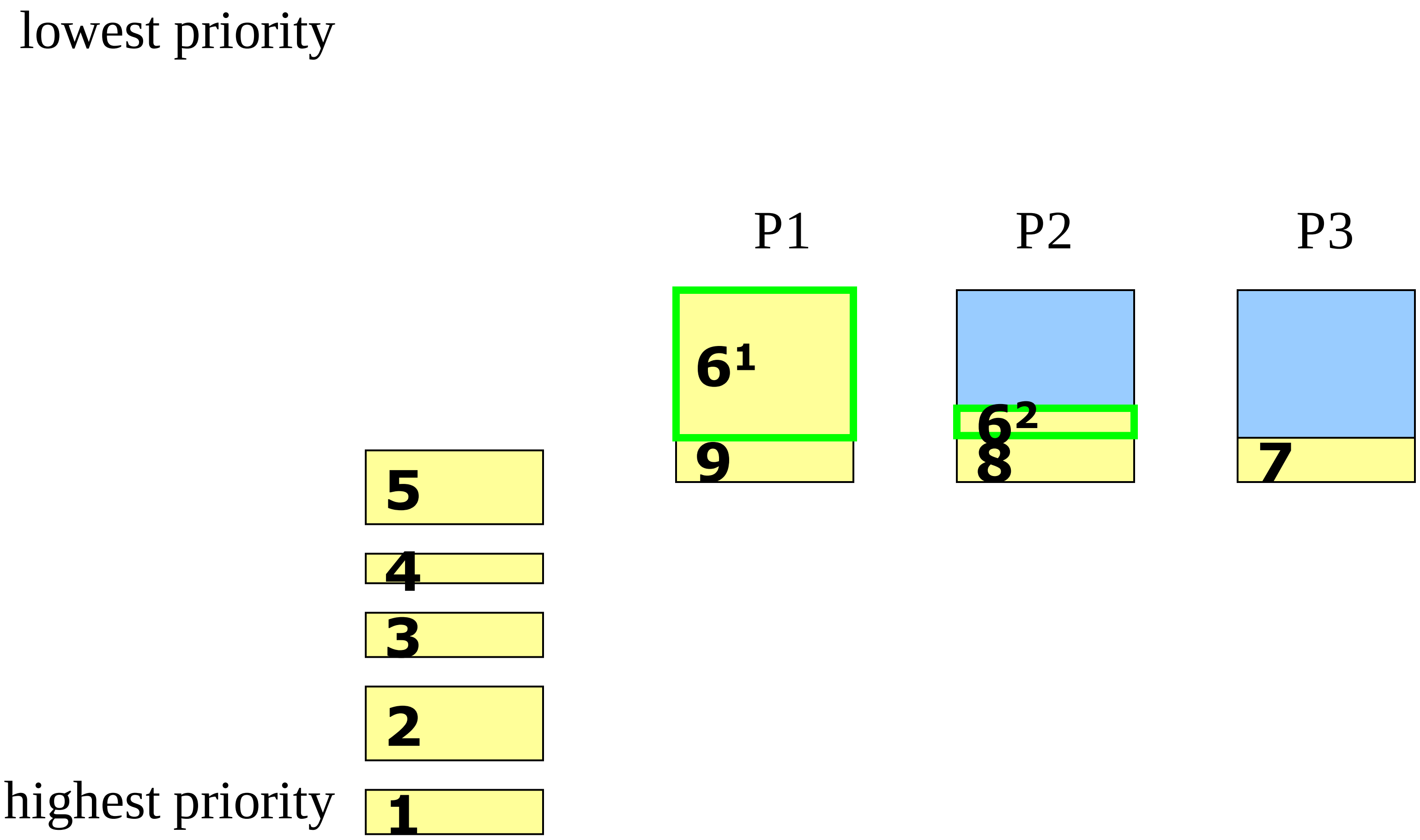
# Problem of Heavy Tasks



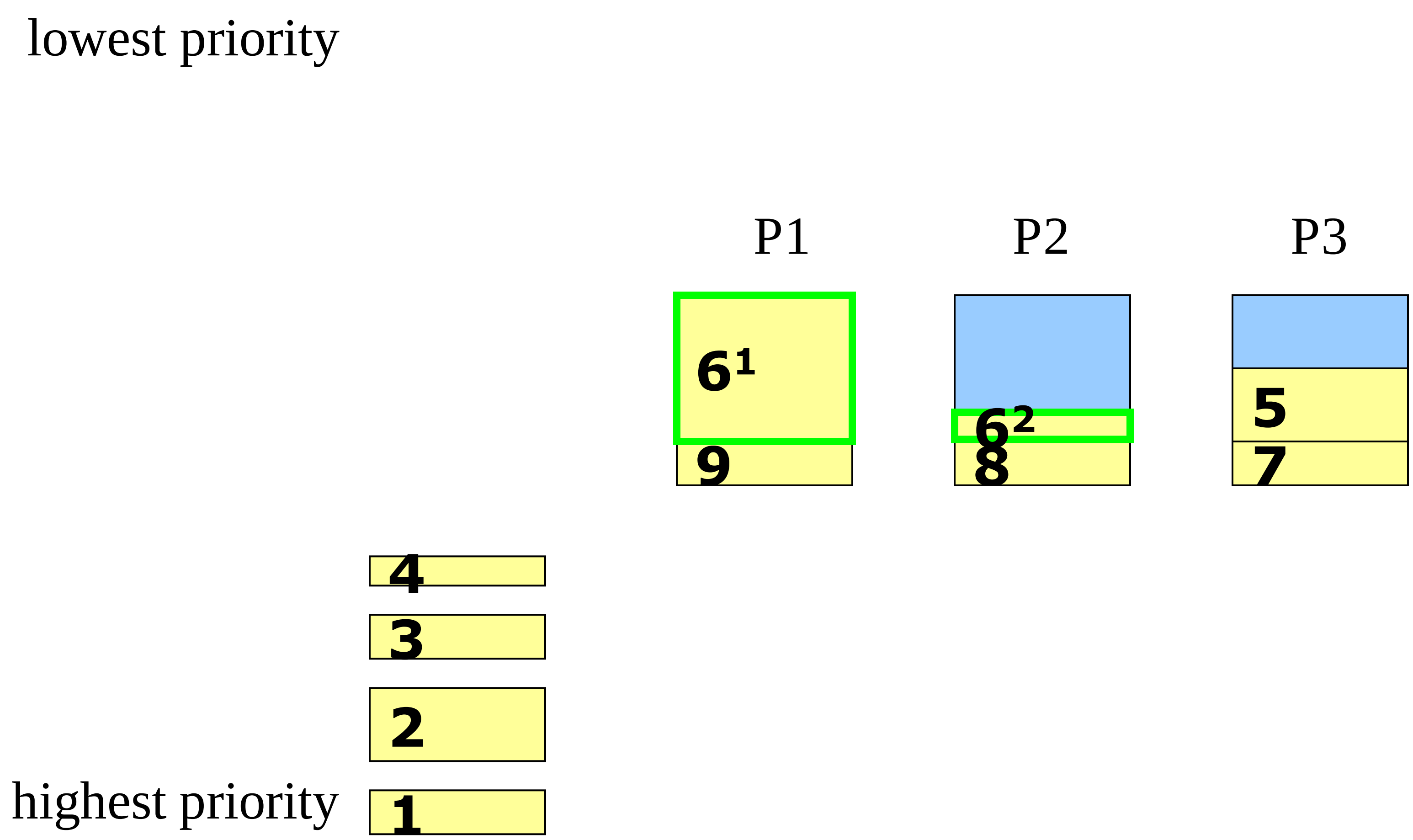
# Problem of Heavy Tasks



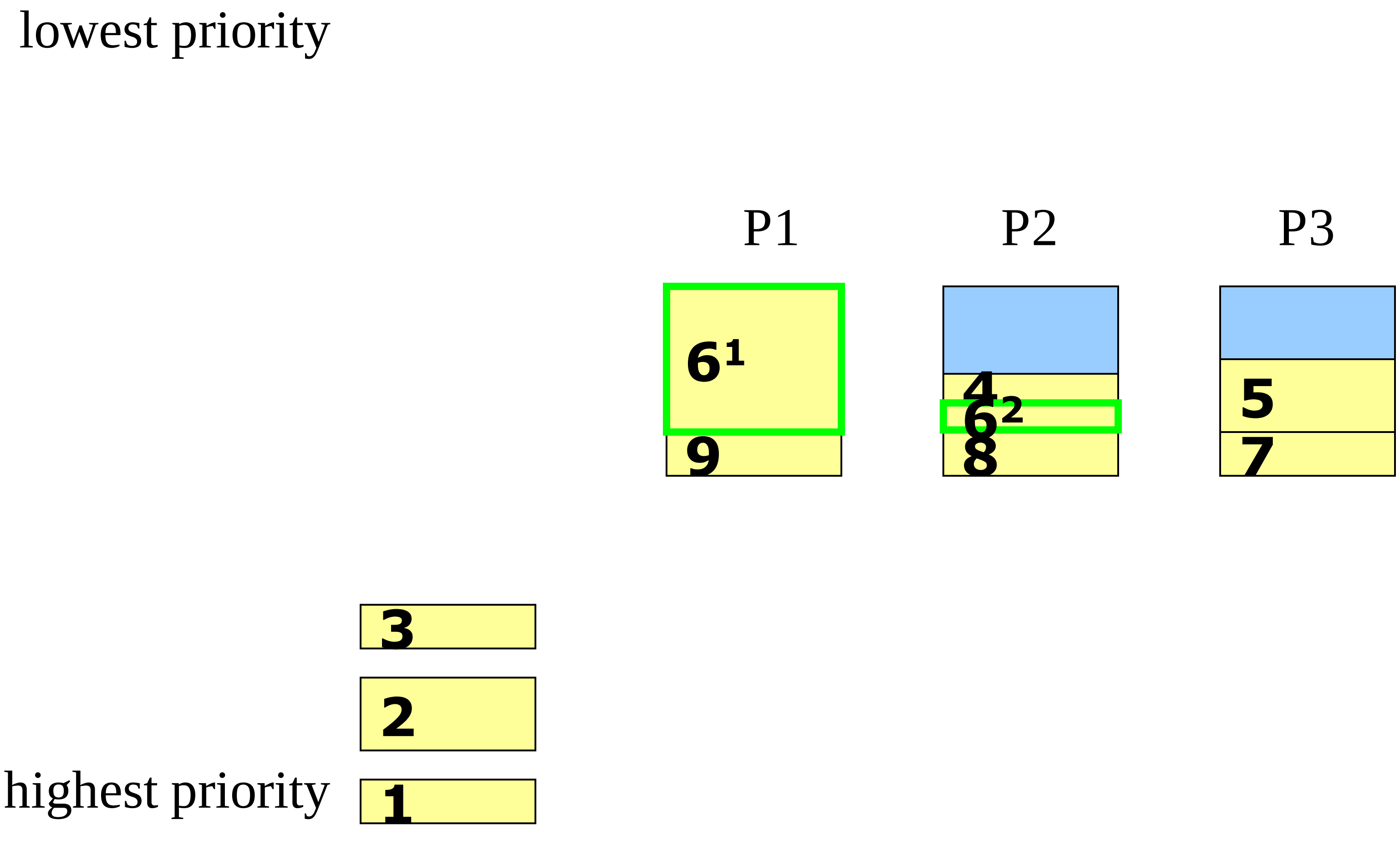
# Problem of Heavy Tasks



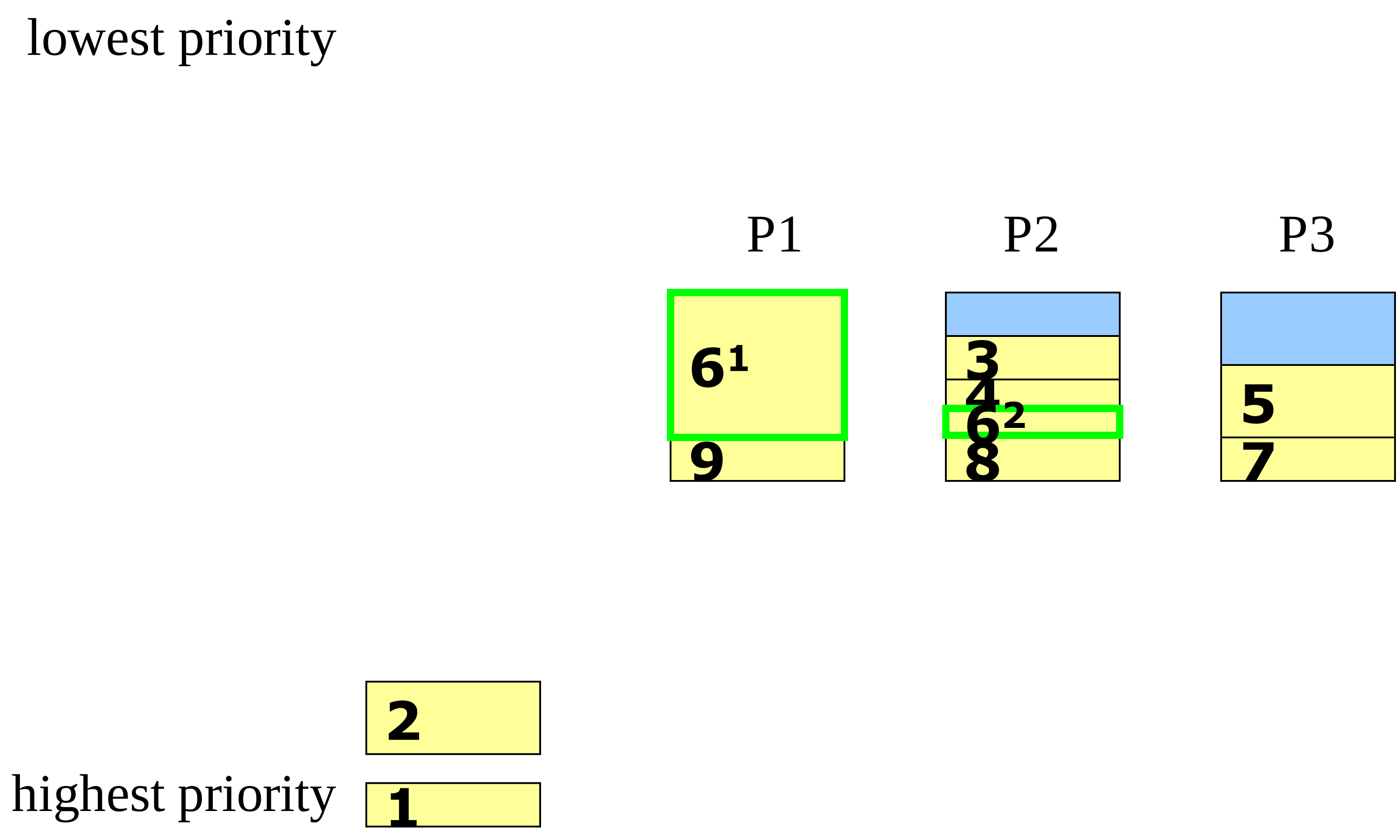
# Problem of Heavy Tasks



# Problem of Heavy Tasks

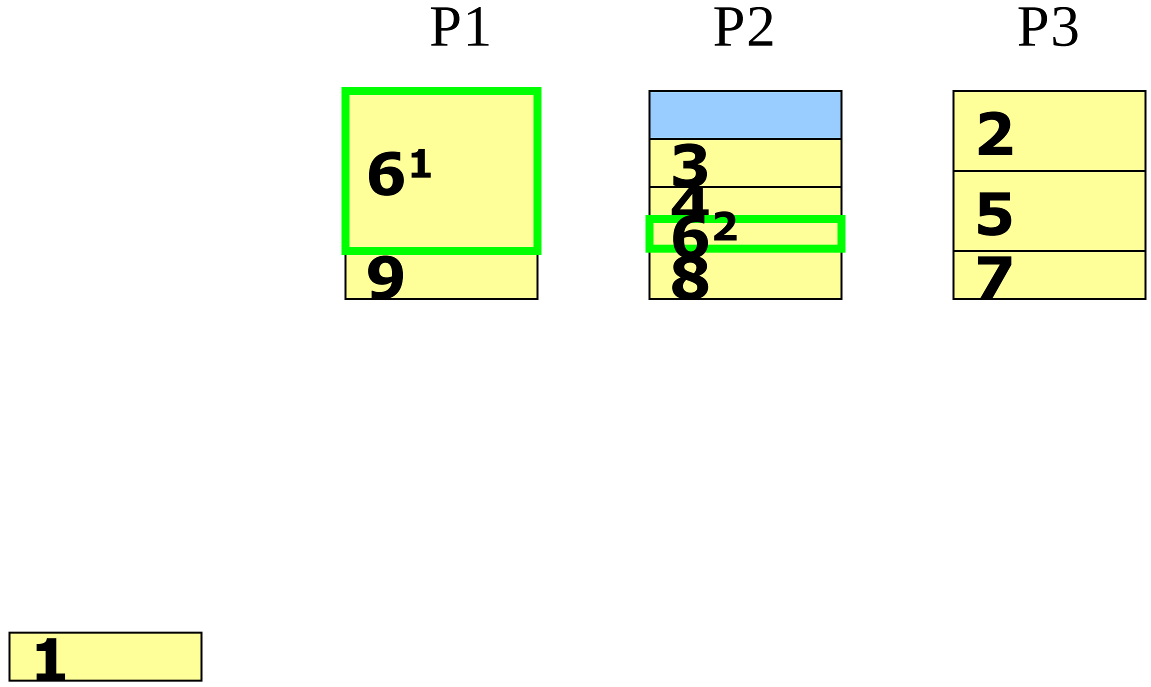


# Problem of Heavy Tasks



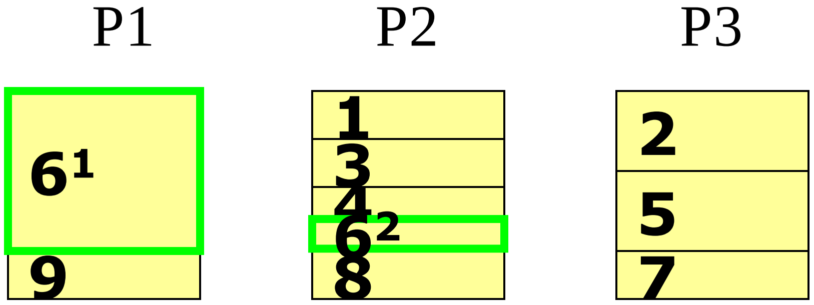
# Problem of Heavy Tasks

lowest priority



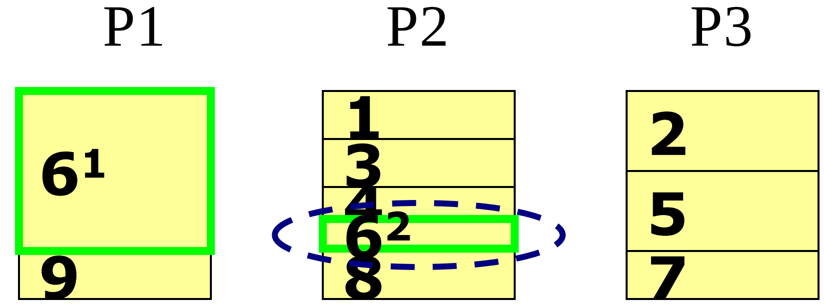

highest priority

# Problem of Heavy Tasks



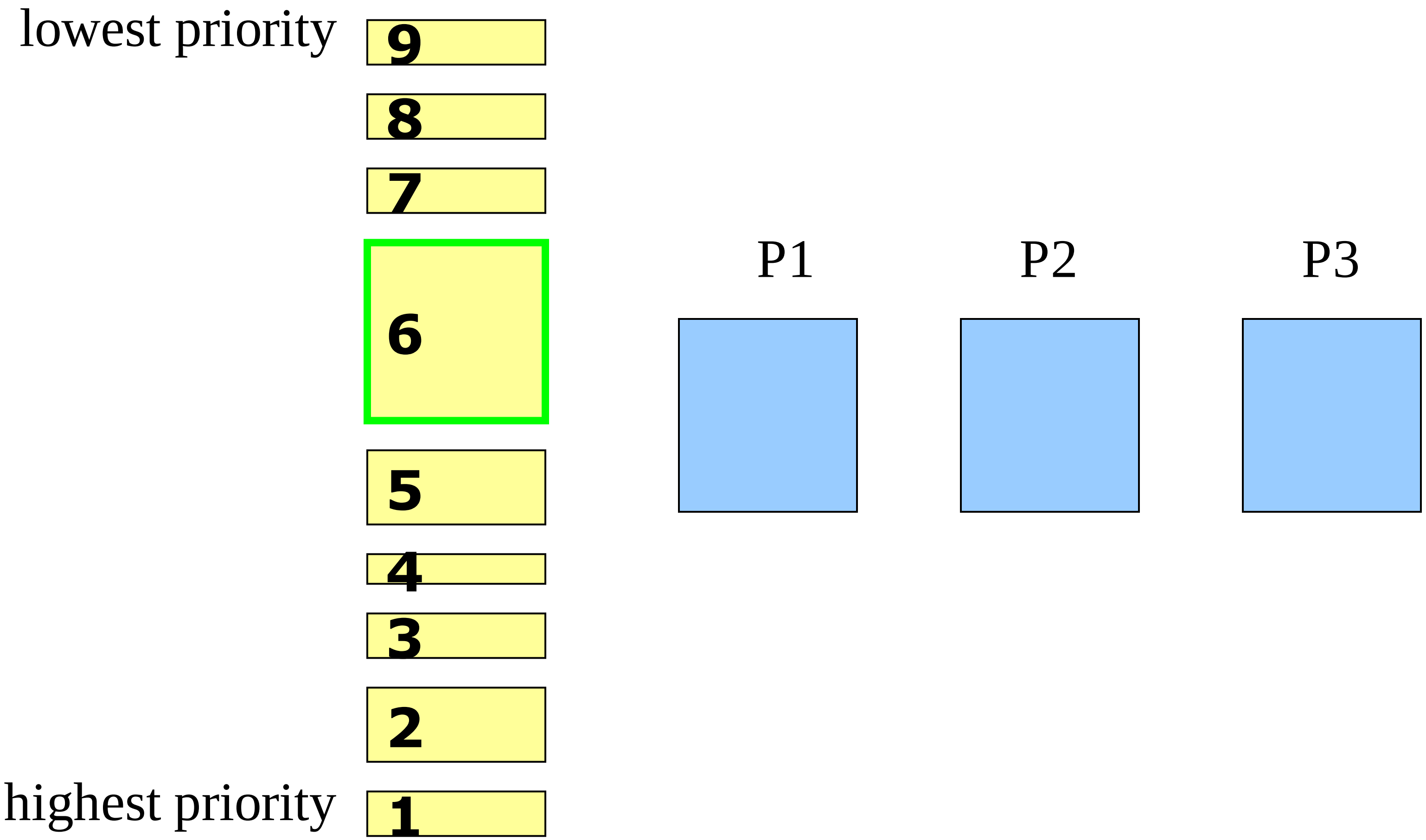
# Problem of Heavy Tasks

the heavy tasks' tail task
may have too low priority level



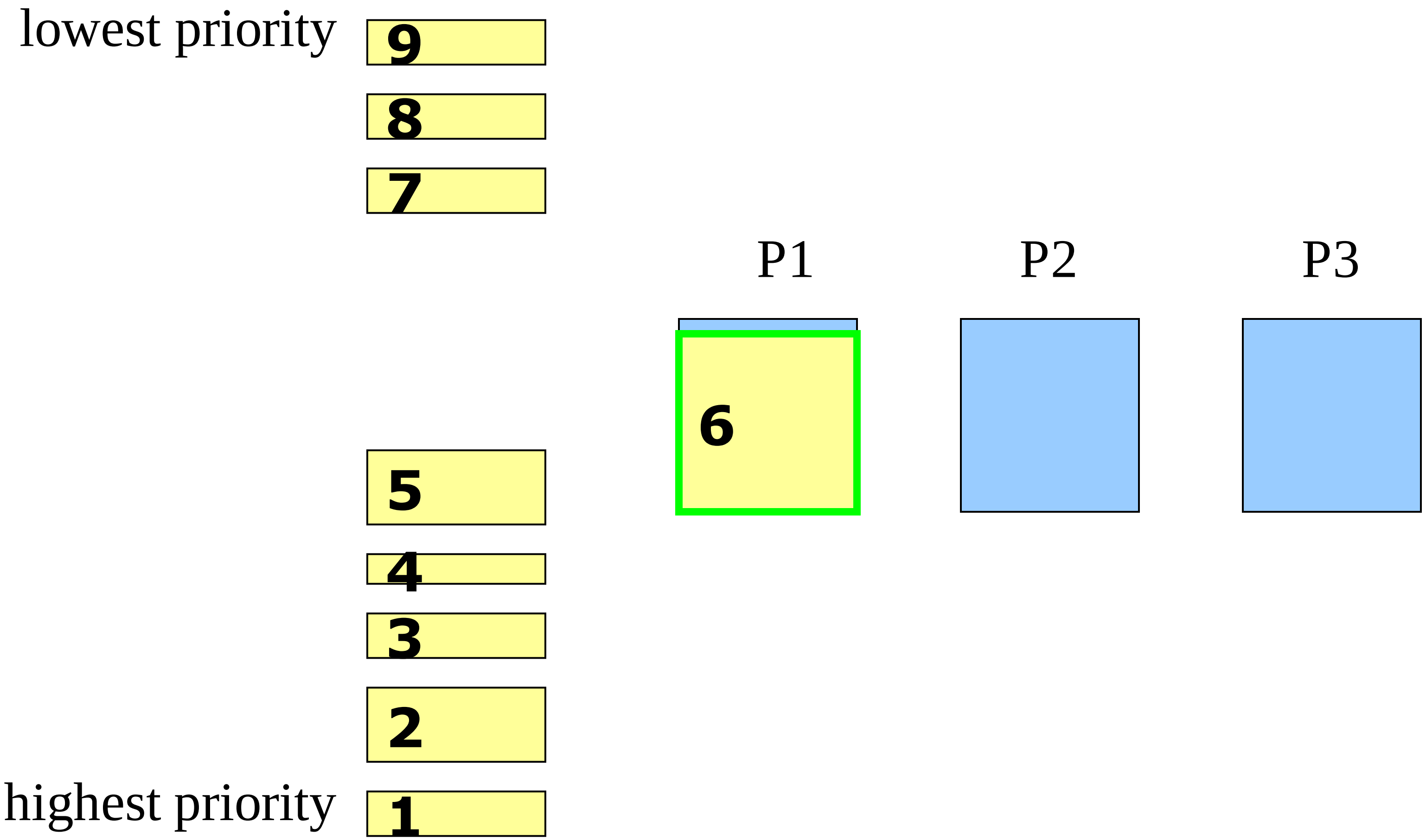
# Solution for Heavy Tasks

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



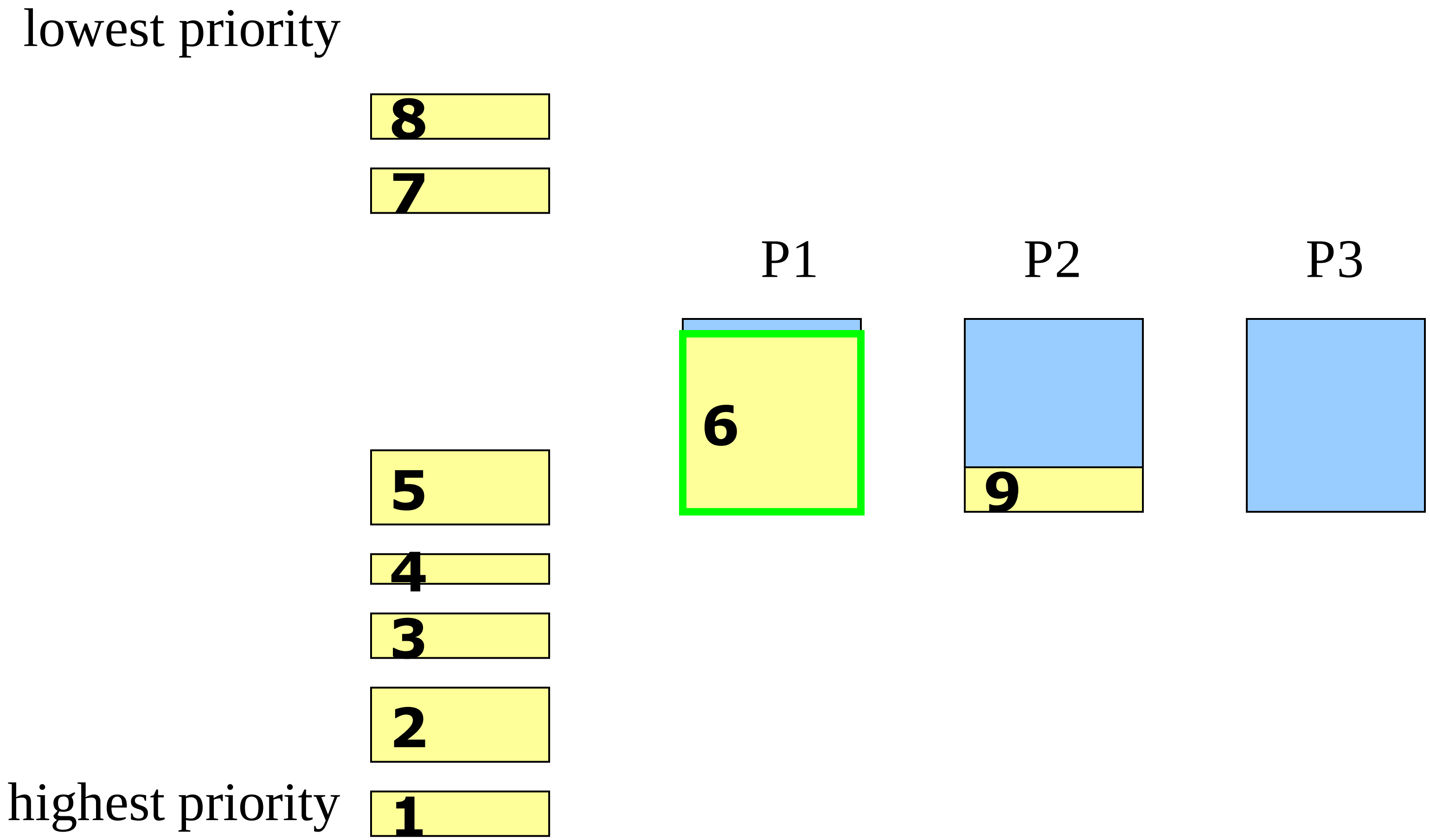
# Solution for Heavy Tasks

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



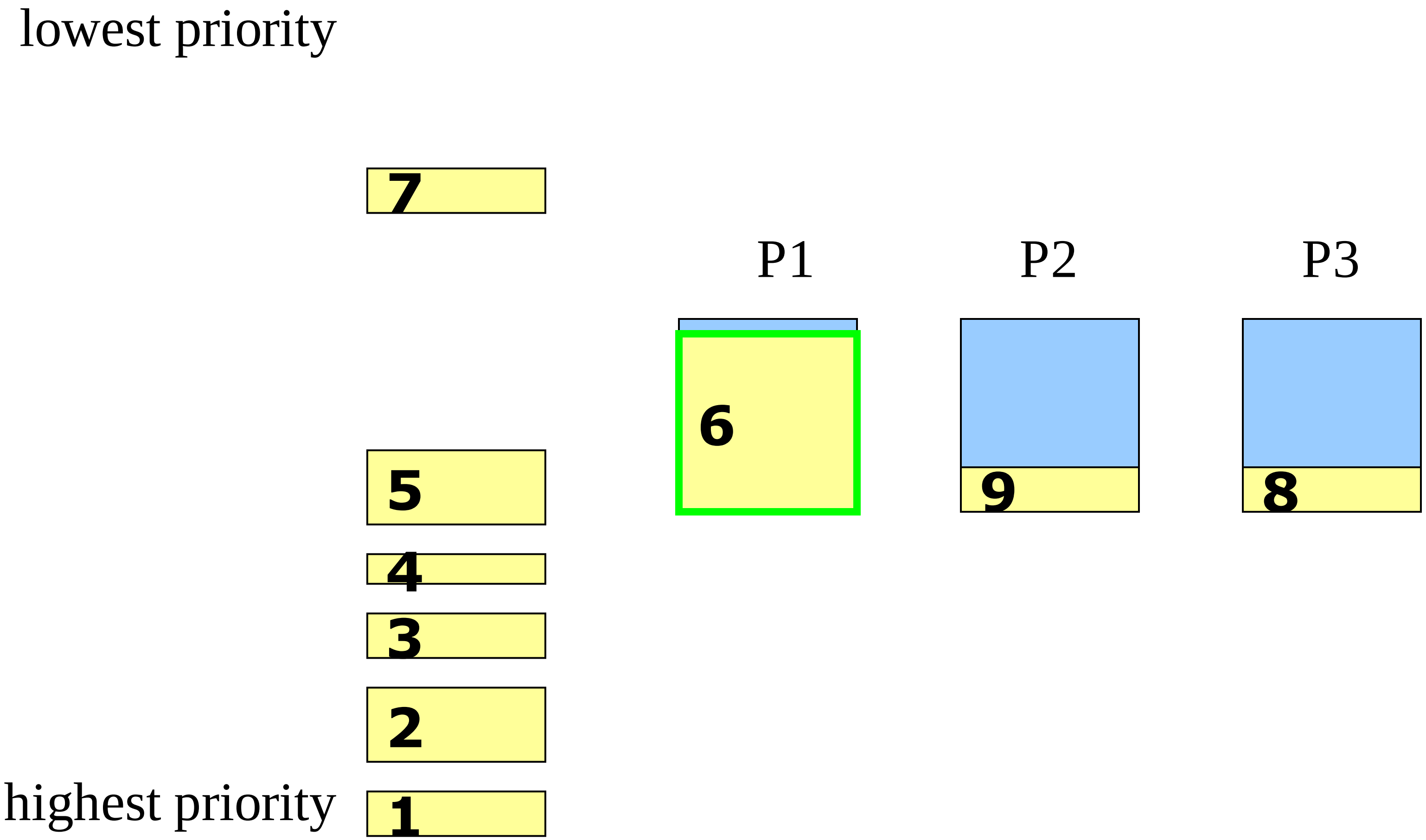
# Solution for Heavy Tasks

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



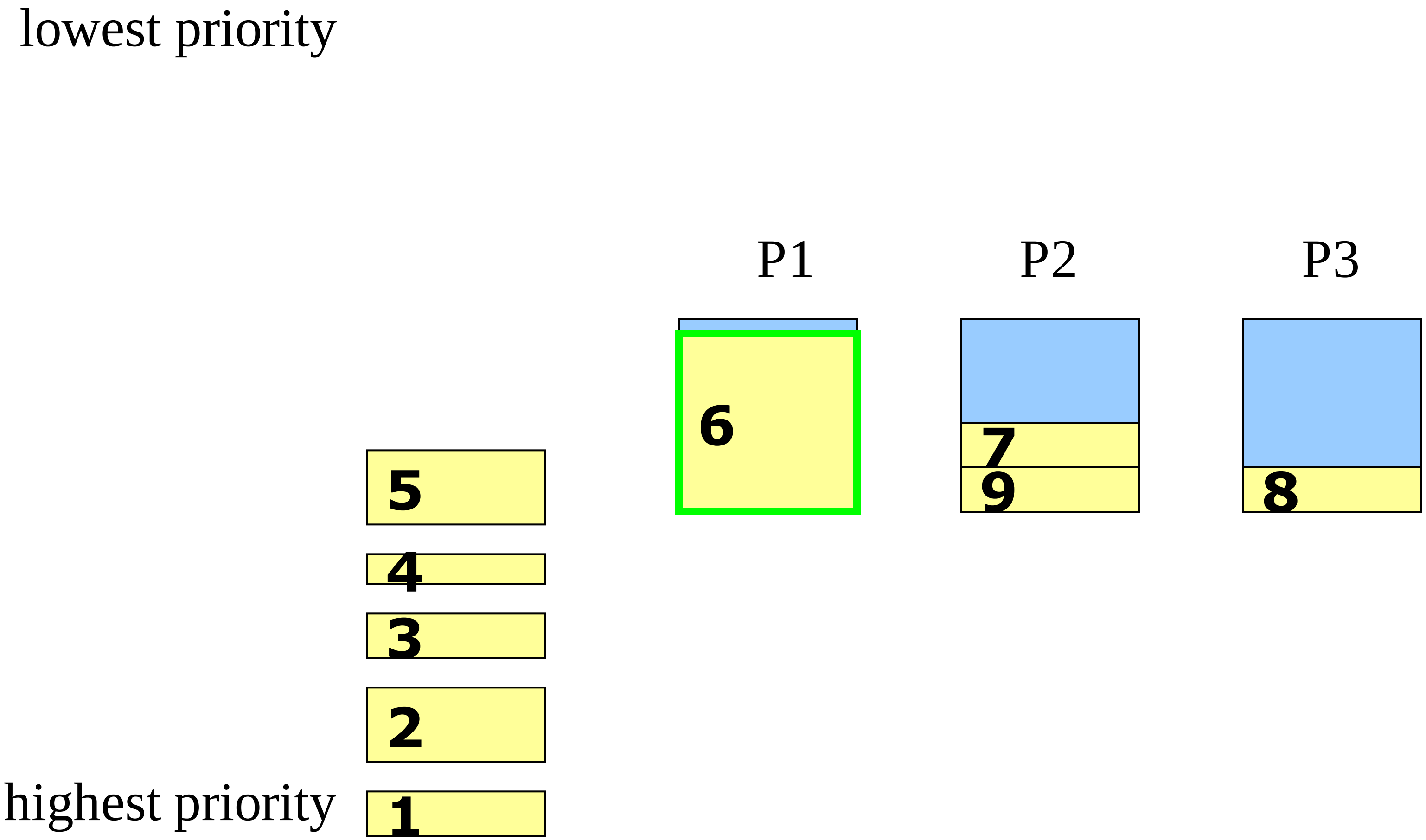
# Solution for Heavy Tasks

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



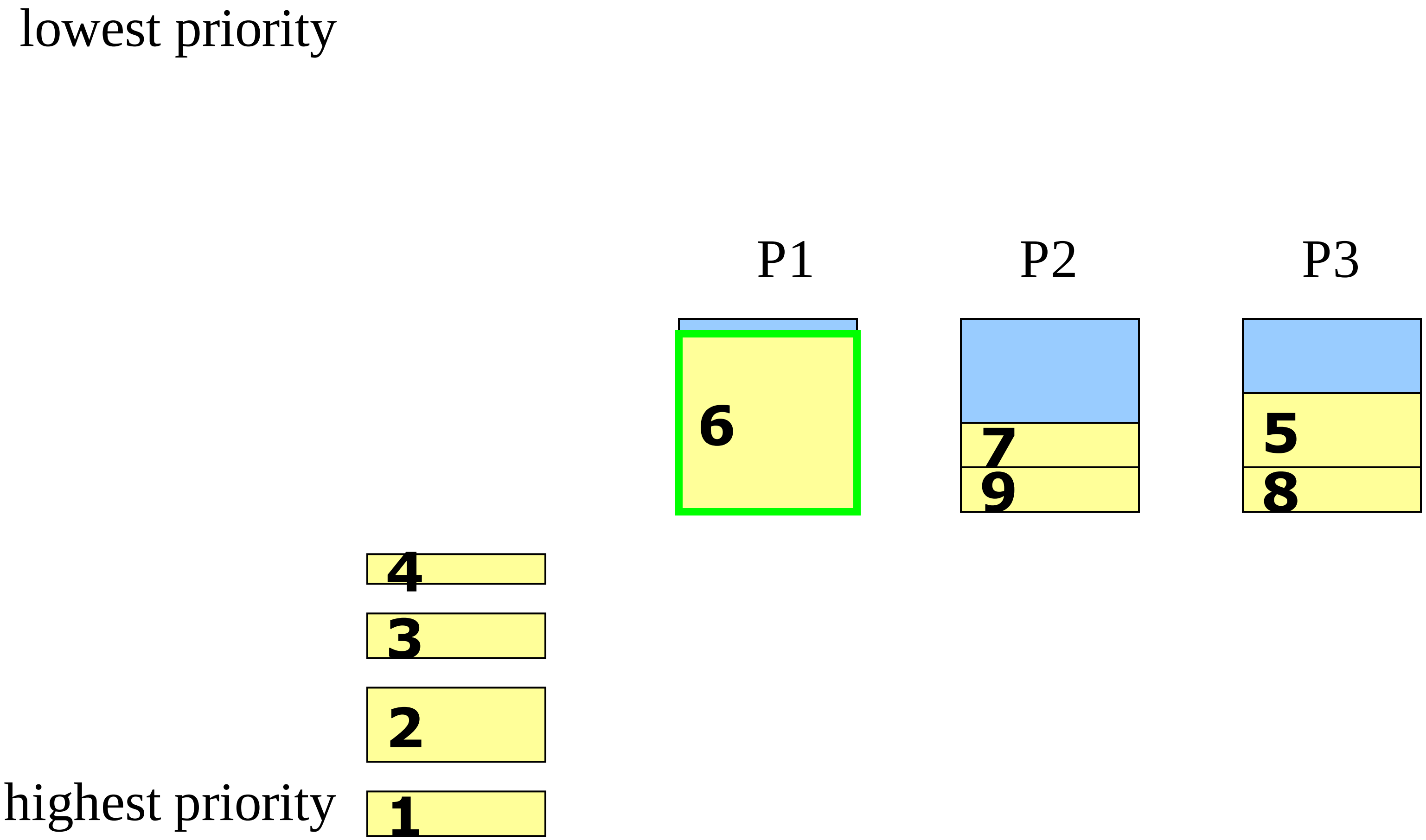
# Solution for Heavy Tasks

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



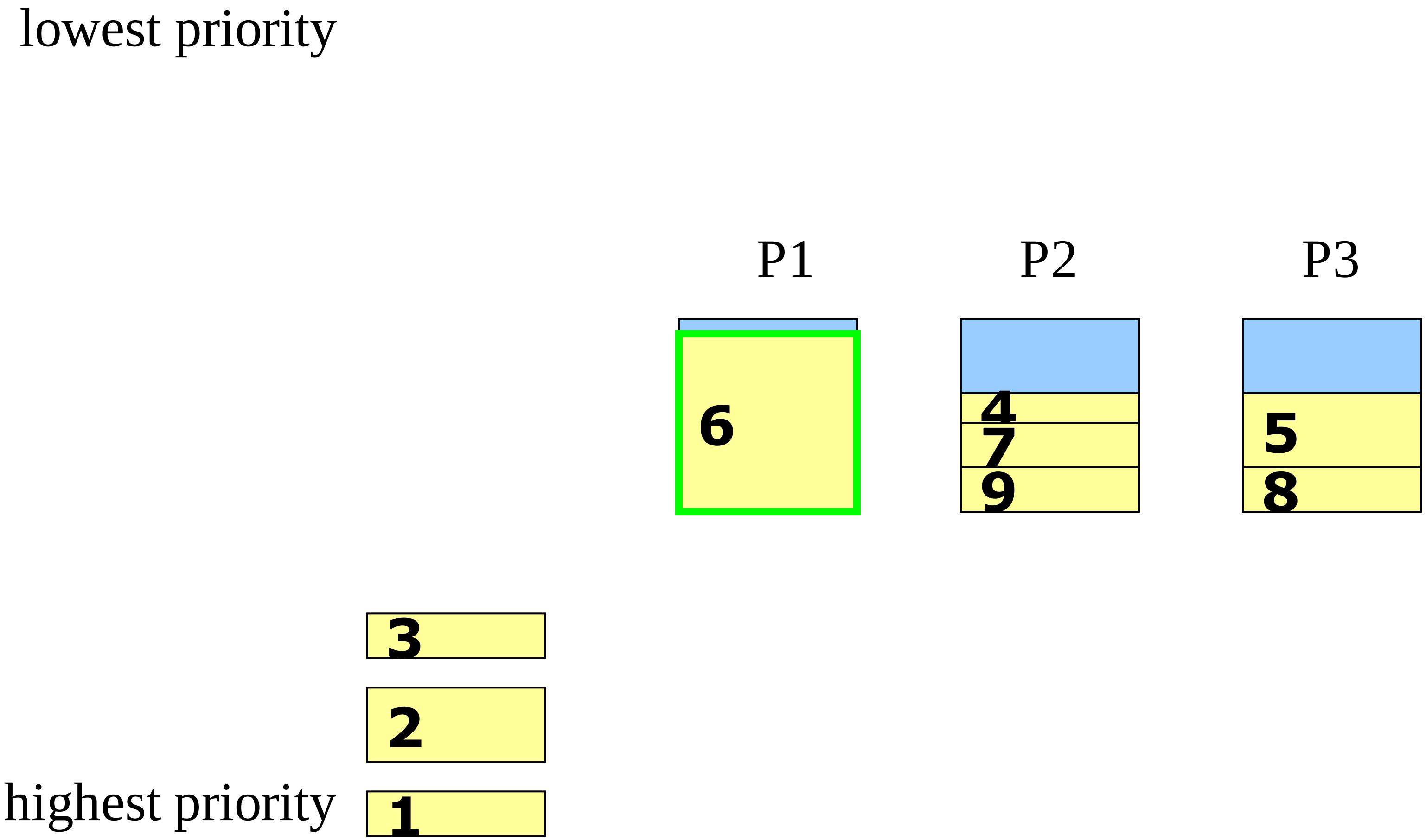
# Solution for Heavy Tasks

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



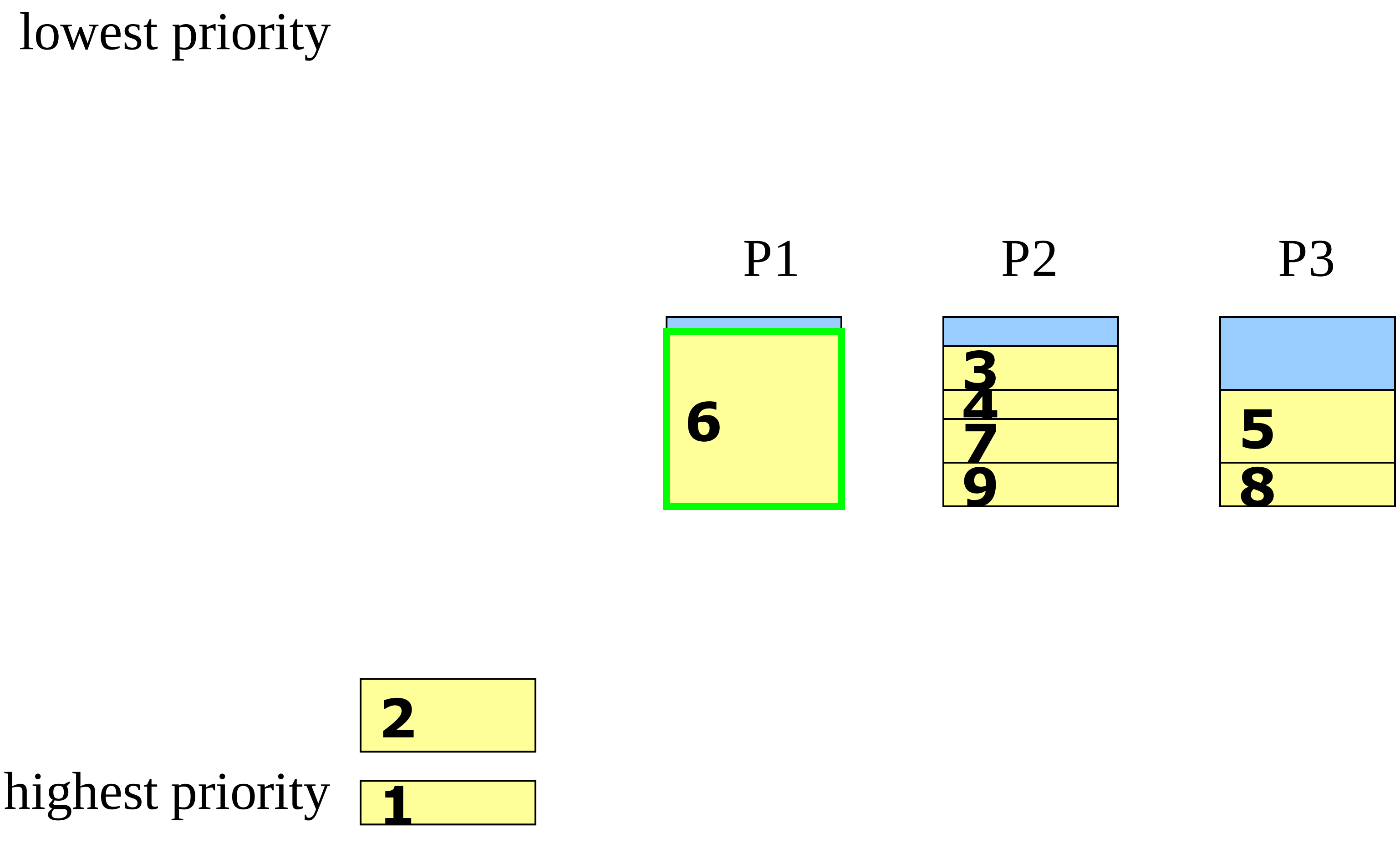
# Solution for Heavy Tasks

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



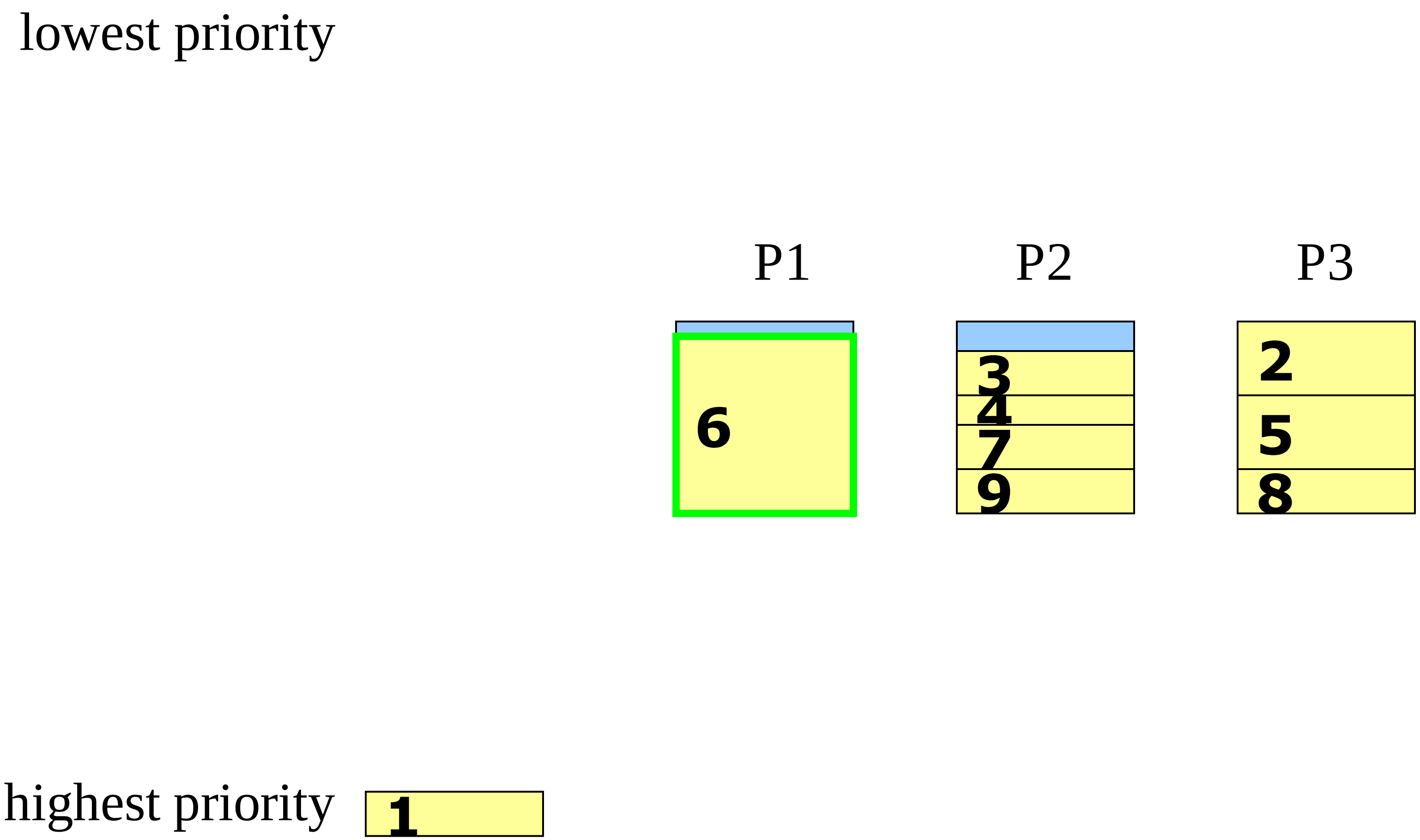
# Solution for Heavy Tasks

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



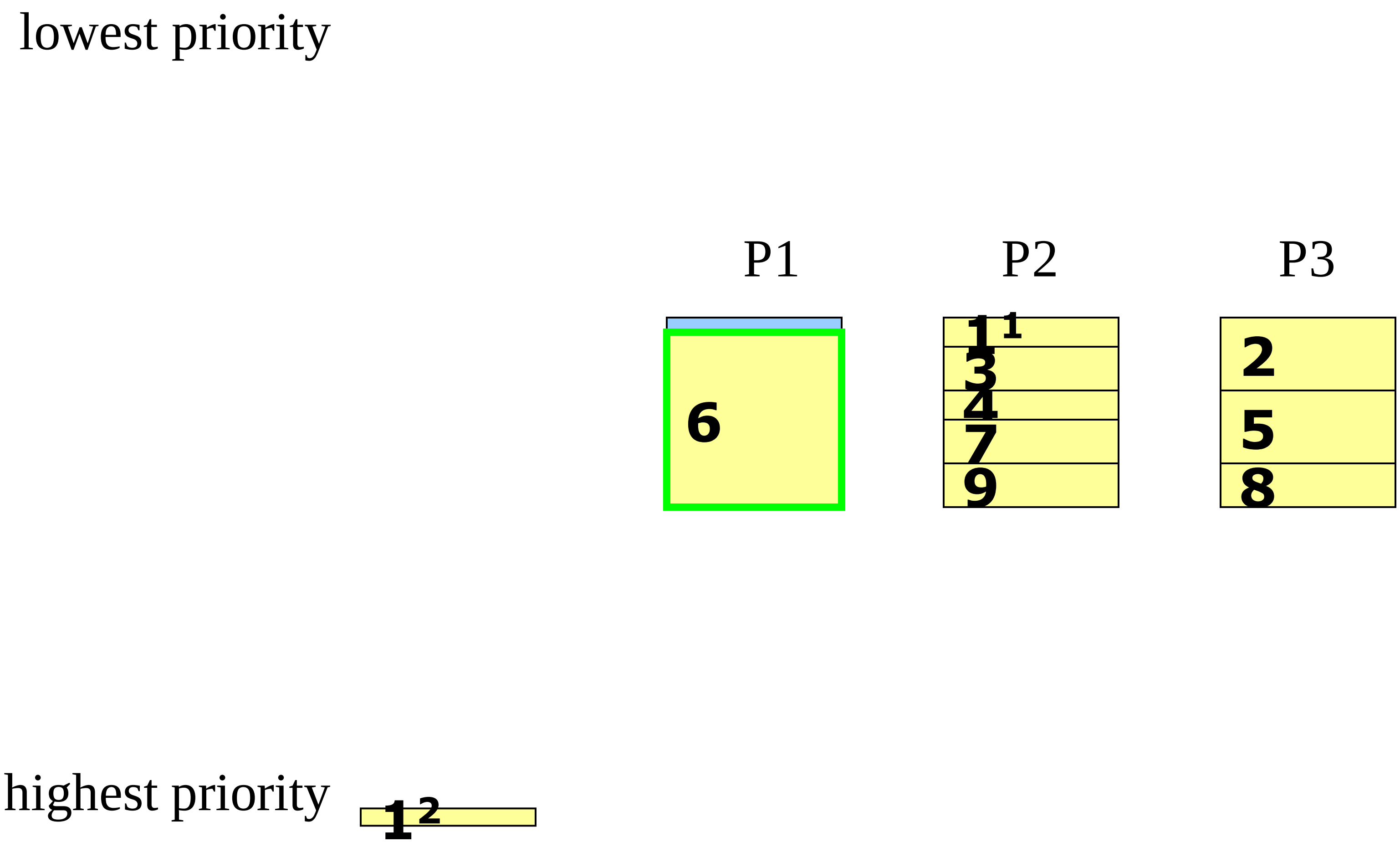
# Solution for Heavy Tasks

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



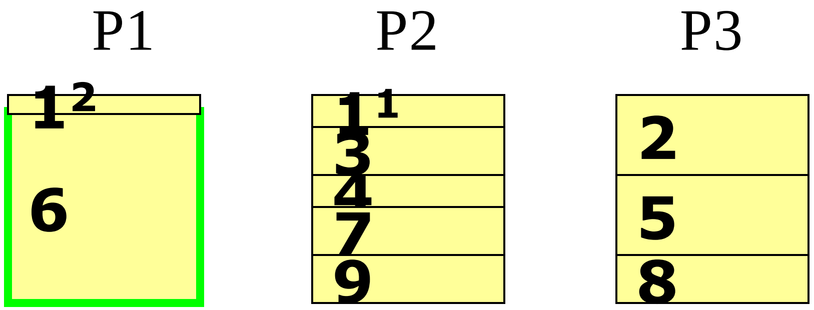
# Solution for Heavy Tasks

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



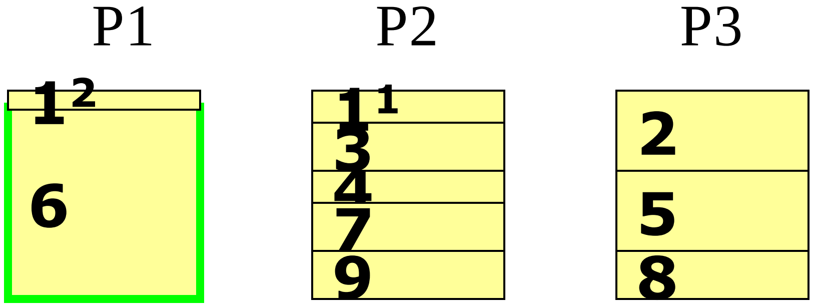
# Solution for Heavy Tasks

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



# Solution for Heavy Tasks

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




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

# Theorem

- By introducing the pre-assignment mechanism, we have

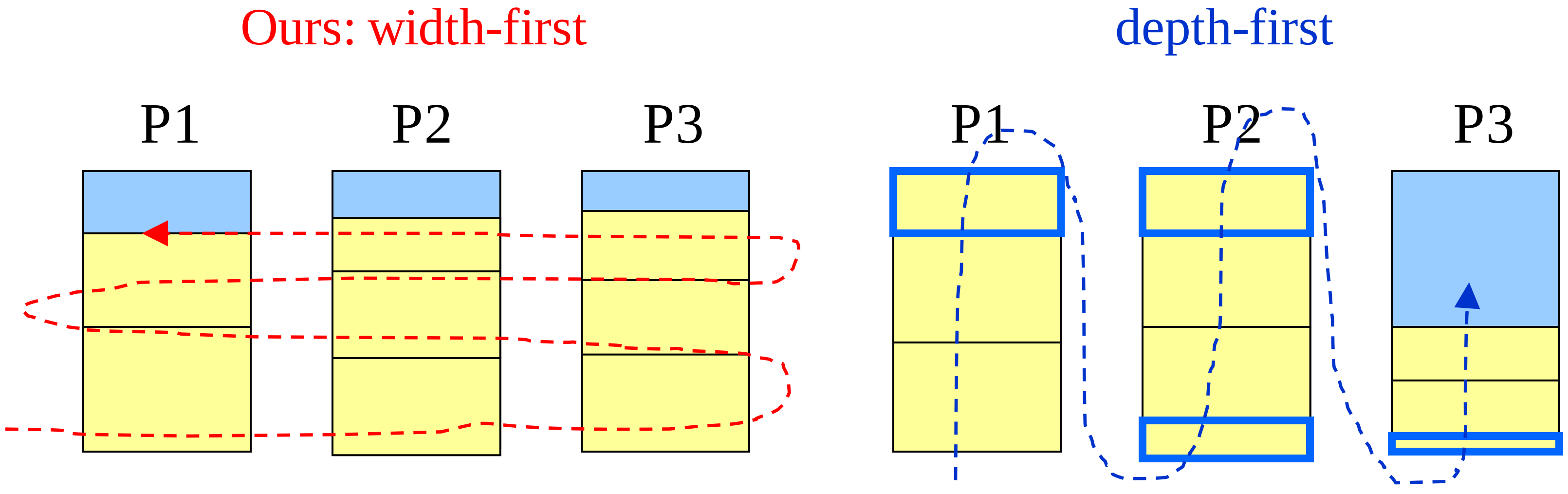
$$\frac{\sum C_i/T_i}{M} \leq N(2^{1/N} - 1)$$

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

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

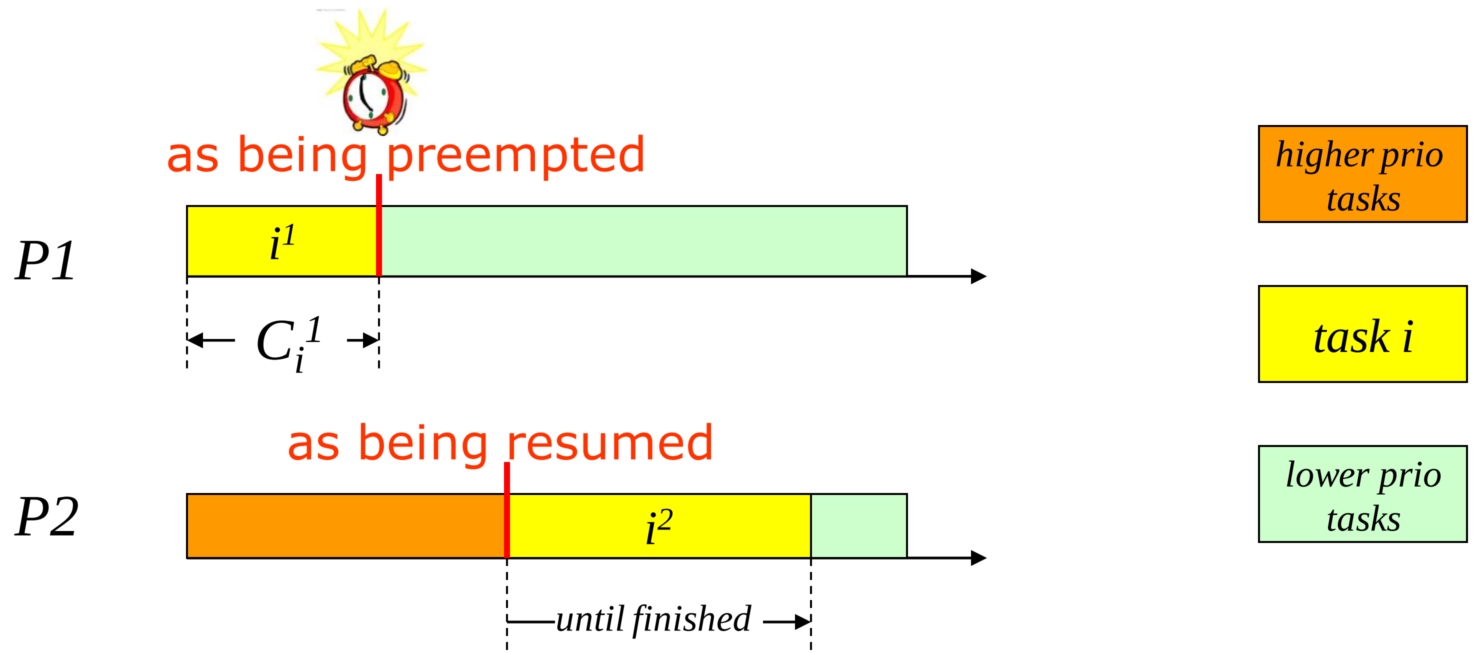
# Overhead

- 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

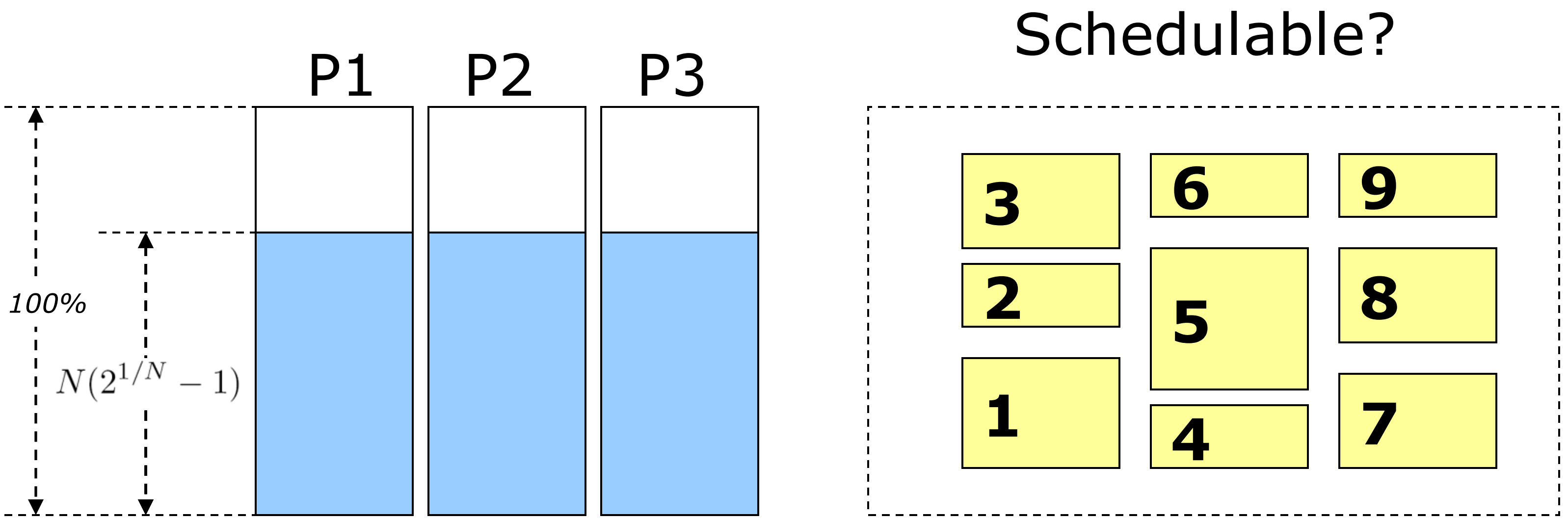


# Implementation

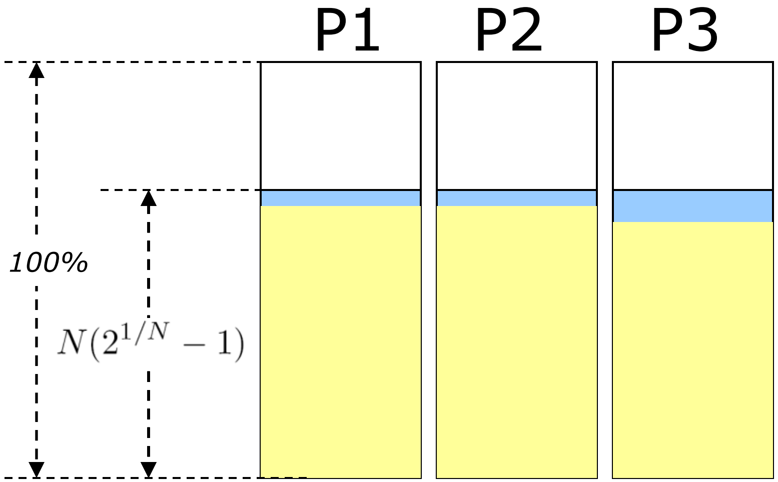
- Easy!
  - One timer for each split task
  - Implemented as "task migration"



# Further Improvement



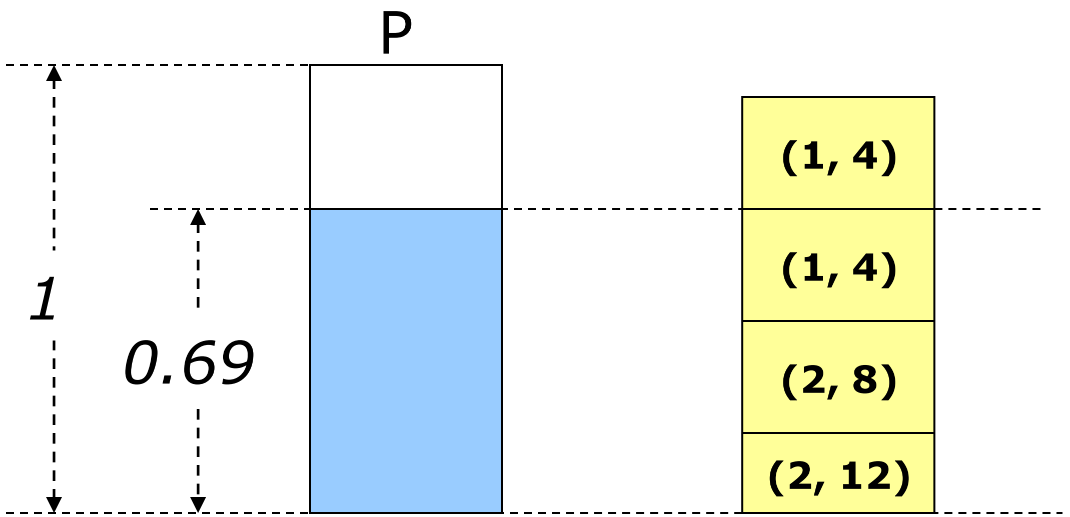
# Uisng Liu and Layland's Utilization Bound




Yes, schedulable by our algorithm

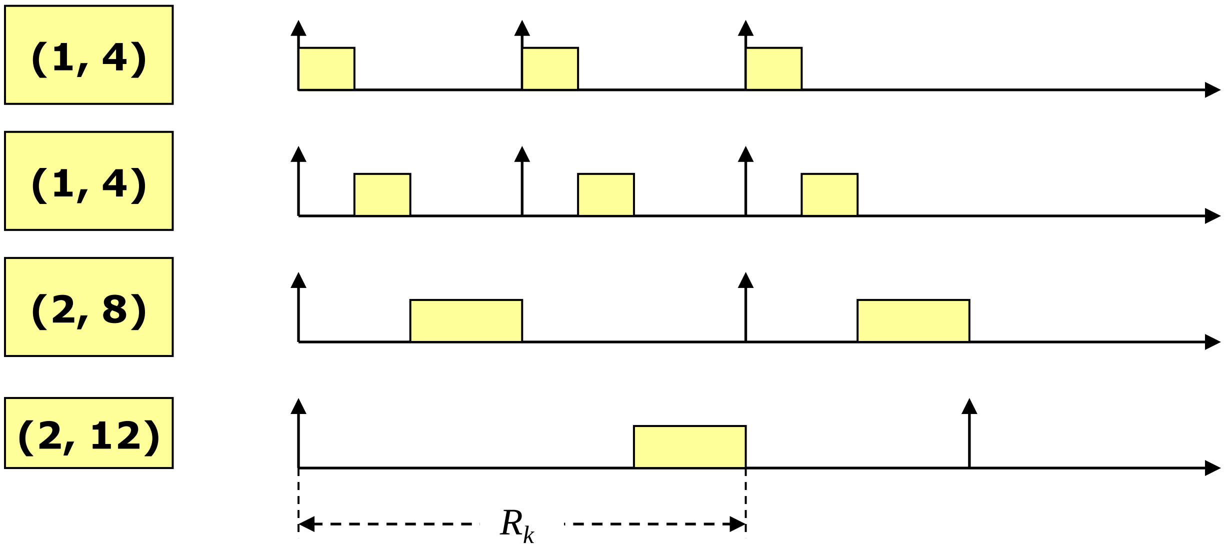
# Utilization Bound is Pessimistic

- The Liu and Layland utilization bound is sufficient but not necessary
- many task sets are actually schedulable even if the total utilization is larger than the bound



# Exact Analysis

- Exact Analysis: Response Time Analysis [Lehoczky_89]
  - pseudo-polynomial



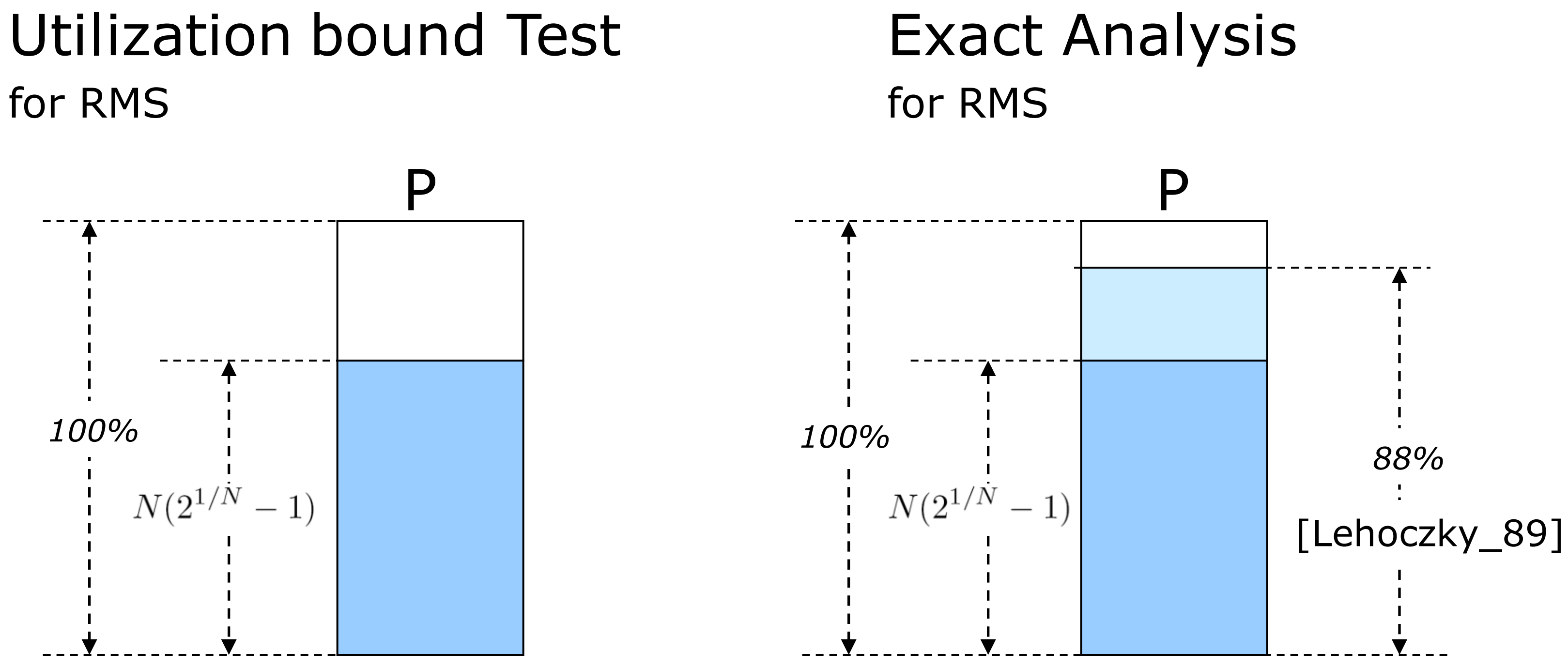

$$R_k = \sum_{T_i < T_k} \left\lceil \frac{R_k}{T_i} \right\rceil C_i + C_k$$

task $k$ is schedulable iff

$R_k <= T_k$

# Utilization Bound v.s. Exact Analysis

- On single processors

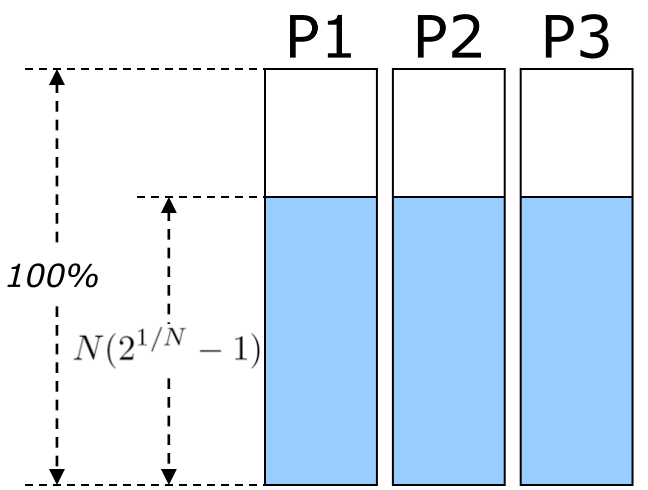


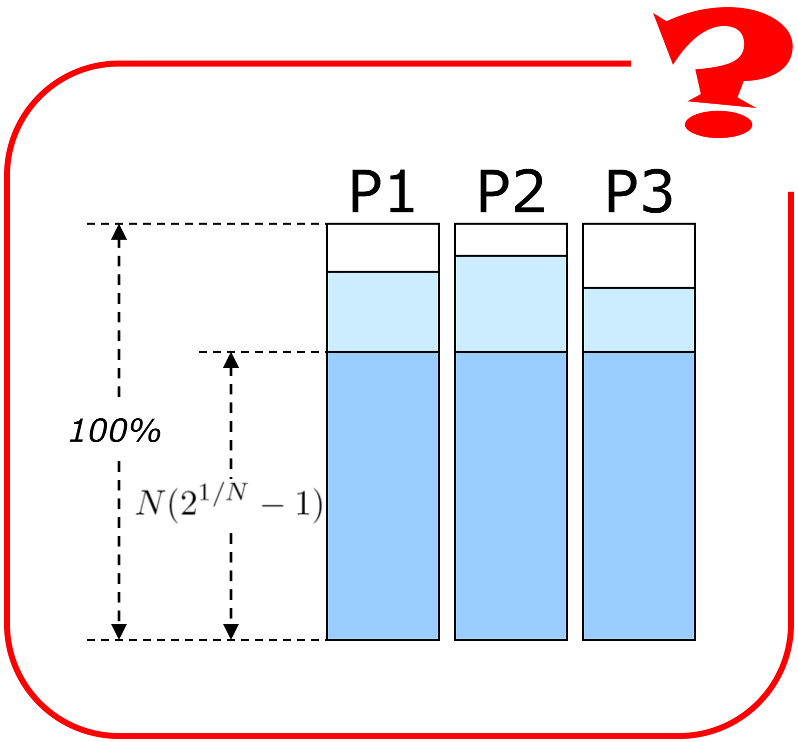
# On Multiprocessors

- Can we do something similar on multiprocessors?

Utilization bound Test

the algorithm introduced above



# Beyond Layland & Liu's Bound [RTSS 2010, rejected!]

- Our RTAS10 algorithm:
  - Increasing RMS priority order & worst-fit partitioning
  - Utilization test to determine the maximal load for each processor
  - The maximal load for each processor bounded by 69.3% $N(2^{\frac{1}{N}} - 1)$

- Improved algorithm:
  - Employ Response Time Analysis to determine the maximal workload on each processor
  - more flexible behavior (more difficult to prove …)
  - Same utilization bound for the worst case, but
  - Much better average performance (by simulation)

**I believe this is "the best algorithm" one can hope for "fixed-prioritiy multiprocessor scheduling"**

# Conclusions

- The (multicore) Timing Problem is challenging
  - Difficult to guarantee Real-Time
  - and Difficult to analyze/predict

- Solutions: Partition & Isolation
  - Shared caches: coloring/partition
  - Memory bus/bandwidth: TDMA, ?
  - Processor cores: partition-based scheduling

Thanks!