Year 2 Review
Brussels, February 12th, 2010

Transversal Activity

Achievements and Perspectives :

Design for Predictability and Performance

leader : Bengt Jonsson

Uppsala University

High-Level Objectives

- technology and design techniques for achieving predictability of systems
  - especially on modern platforms
- trade-offs between performance and predictability

Predictability transverses all levels of abstraction in embedded systems design:

- Verification, modeling, compilation, OS, platforms.

Industrial Sectors

- safety-critical systems
  - transportation, power automation, medical systems, ...
  - Market of over $900 million in 2008 [int. ARC Advisory Group]
- sectors, where systems failure may lead to economic consequences
  - consumer electronics, telecom, ...

Partners

Modeling & Validation:

- IST – Austria (Tom Henzinger)
- INRIA – France (Alain Girault)
- Uppsala (Bengt Jonsson)
- PARADES – Italy
  (Alberto Sangiovanni–Vincentelli)

Code Generation & Timing analysis

- Dortmund (Peter Marwedel)
- Saarland (Reinhard Wilhelm)
- TU Vienna (Peter Puschner)

OS & Networks

- Cantabria
  (Michael Gonzalez–Harbour)
- SSSA (Giorgio Buttazzo)
- York (Alan Burns)

Hardware Platforms & MPSoC

- Bologna (Luca Benini)
- Braunschweig (Rolf Ernst)
  [affiliated]
- ETHZ – Zürich (Lothar Thiele)
- IMEC – Belgium
  (Stylianos Mamagkakis)
- Linköping (Petru Eles)

Building Excellence

- Most existing work is within one system level, e.g,:
  - Modeling and verification of timed component-based systems,
  - Timing analysis for programs
  - Compiler techniques for timing and memory predictability
  - OS Scheduling and resource management
  - Predictability in memory

Main Aim:

- Integrate research across different levels of abstraction

System Modeling and Validation

Checking Predictability and Robustness

- Y1: formalizations of predictability and robustness as determinism/continuity,
- Y2: algorithms for checking and synthesizing predictable and robust programs/circuits

Component-based design under RT constraints

(RT-CCM component model)

Standardization: Continuation of UML MARTE

- Participation to SysML standardization
- Dissemination of MARTE
  - Workshops
  - Methodology

Modeling & Validation:

- IST - Austria (Tom Henzinger)
- INRIA – France (Alain Girault)
- Uppsala (Bengt Jonsson)
- PARADES (Alberto Sangiovanni–Vincentelli)

Code Generation & Timing analysis

- Dortmund (Peter Marwedel)
- Saarland (Reinhard Wilhelm)
- TU Vienna (Peter Puschner)

OS & Networks

- Cantabria (Michael Gonzalez–Harbour)
- SSSA (Giorgio Buttazzo)
- York (Alan Burns)

Hardware Platforms & MPSoC

- Bologna (Luca Benini)
- Braunschweig (Rolf Ernst) [affiliated]
- ETHZ – Zürich (Lothar Thiele)
- IMEC – Belgium (Stylianos Mamagkakis)
- Linköping (Petru Eles)

Timing Analysis and Compiler Techniques

WCET Analysis for different features

- New cache replacement policies (FIFO)
- Pipeline representation
- Operating modes in software
- Dynamic memory allocation
- Context switches

Integration of WCET analysis and compilation

- Continuation of Y1 work
- Loop bound analysis
- WCET-aware optimizations

Resulting in the WCC compiler (the leading WCET-aware compiler).

Modeling & Validation:

- IST - Austria (Tom Henzinger)
- INRIA – France (Alain Girault)
- Uppsala (Bengt Jonsson)
- PARADES (Alberto Sangiovanni–Vincentelli)

Code Generation & Timing analysis

- Dortmund (Peter Marwedel)
- Saarland (Reinhard Wilhelm)
- TU Vienna (Peter Puschner)

OS & Networks

- Cantabria (Michael Gonzalez–Harbour)
- SSSA (Giorgio Buttazzo)
- York (Alan Burns)

Hardware Platforms & MPSoC

- Bologna (Luca Benini)
- Braunschweig (Rolf Ernst) [affiliated]
- ETHZ – Zürich (Lothar Thiele)
- IMEC – Belgium (Stylianos Mamagkakis)
- Linköping (Petru Eles)

Time Predictability on Multiprocessor systems

Scheduling Analysis and Model Checking for multicore platforms

- Cache isolation (cache coloring) techniques for tasks on muticores w. shared caches
- Optimal utilization bounds for multiprocessor scheduling
- Scheduling of tasks that access shared resources in a multicore system.
- Schedulability tests for task sets with shared resources (using priority ceiling)
- Combining scheduling and model checking techniques

Modeling & Validation:

- IST - Austria (Tom Henzinger)
- INRIA – France (Alain Girault)
- Uppsala (Bengt Jonsson)
- PARADES (Alberto Sangiovanni–Vincentelli)

Code Generation & Timing analysis

- Dortmund (Peter Marwedel)
- Saarland (Reinhard Wilhelm)
- TU Vienna (Peter Puschner)

OS & Networks

- Cantabria (Michael Gonzalez–Harbour)
- SSSA (Giorgio Buttazzo)
- York (Alan Burns)

Hardware Platforms & MPSoC

- Bologna (Luca Benini)
- Braunschweig (Rolf Ernst) [affiliated]
- ETHZ – Zürich (Lothar Thiele)
- IMEC – Belgium (Stylianos Mamagkakis)
- Linköping (Petru Eles)

Platforms and Architectures

Predictability for Fault-tolerant ES

- Combining HW (processor hardening) and SW (process re-execution) techniques, and associated analysis

Predictability for MPSoC Architectures

- New communication policies
- Design and Synthesis of bus controllers
  - Guaranteeing predictability *and* efficiency.

Power prediction algorithms

For distributed embedded systems

Modeling & Validation:

- IST - Austria (Tom Henzinger)
- INRIA – France (Alain Girault)
- Uppsala (Bengt Jonsson)
- PARADES (Alberto Sangiovanni–Vincentelli)

Code Generation & Timing analysis

- Dortmund (Peter Marwedel)
- Saarland (Reinhard Wilhelm)
- TU Vienna (Peter Puschner)

OS & Networks

- Cantabria (Michael Gonzalez–Harbour)
- SSSA (Giorgio Buttazzo)
- York (Alan Burns)

Hardware Platforms & MPSoC

- Bologna (Luca Benini)
- Braunschweig (Rolf Ernst) [affiliated]
- ETHZ – Zürich (Lothar Thiele)
- IMEC – Belgium (Stylianos Mamagkakis)
- Linköping (Petru Eles)

Time Predictive Hardware and Software

Time predictive language and arch.

- PRET-C, time-predictive programming language (C with Esterel-like constructs for parallel threads)

Code Generation Techniques

- for time-predictable program execution (eliminating timing anomalies)

Modeling & Validation:

- IST - Austria (Tom Henzinger)
- INRIA – France (Alain Girault)
- Uppsala (Bengt Jonsson)
- PARADES (Alberto Sangiovanni–Vincentelli)

Code Generation & Timing analysis

- Dortmund (Peter Marwedel)
- Saarland (Reinhard Wilhelm)
- TU Vienna (Peter Puschner)

OS & Networks

- Cantabria (Michael Gonzalez–Harbour)
- SSSA (Giorgio Buttazzo)
- York (Alan Burns)

Hardware Platforms & MPSoC

- Bologna (Luca Benini)
- Braunschweig (Rolf Ernst) [affiliated]
- ETHZ – Zürich (Lothar Thiele)
- IMEC – Belgium (Stylianos Mamagkakis)
- Linköping (Petru Eles)

Overall Assessment and Vision at Y0+2

- Many collaborations working very well
  - WCC Compiler, MPSoC architecture analysis, Definition and assessment of “predictability”, Predictable implementation for Model-based Design,
  - New developments
    - Scheduling on Multicore Platforms
    - Time-predictive HW and SW implementations
  - Global Event
- 10 joint publications, several mutual visits and joint projects

To be improved

- Wider transversal integration across levels of abstraction
  - E.g., Towards implementation of model-based design, timing aware compiler, predictable multicore architecture and operating system
- More solid definition of what is “predictability”

Scientific Highlights

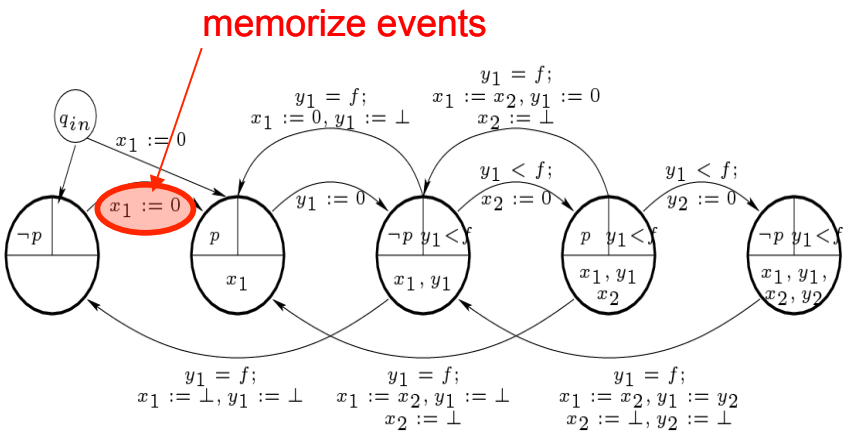
- Synthesis of Real –Time Controllers from MTL specifications
- Improved utilization bound for multiprocessor scheduling
- New results in WCET analysis

Synthesis of Predictable Real-time Controllers from Temporal Logic

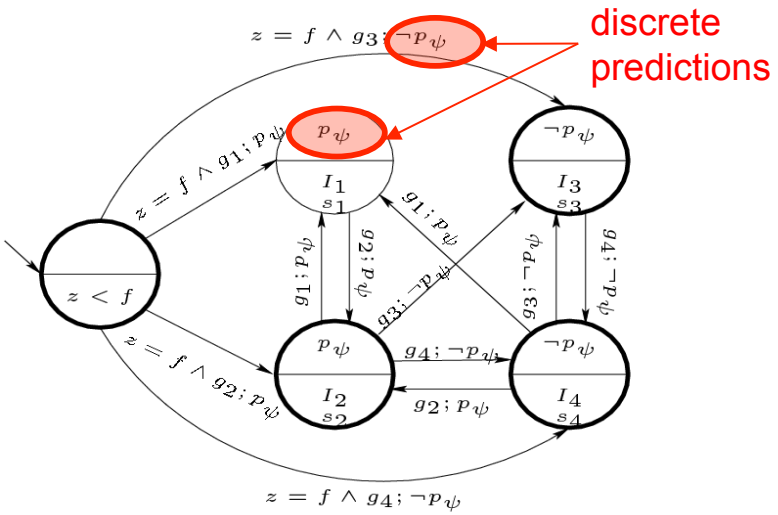
- **Objective:** automatic synthesis of **predictable** real-time controllers from high-level specifications
- **Metric Temporal Logic (MTL)** – real-time high-level specification language
  - Convert the MTL formula ϕ to a **deterministic** timed automaton (TA)
    - Synthesis from specifications given as deterministic timed automata is possible
- 2 sources of non-determinism in automata constructed from MTL formulas
  - **Unbounded** number of “events” in a bounded time interval
    - Automaton needs to remember them
  - **A-causal** semantics of MTL
    - Automaton needs to predict future values of inputs

From MTL to Deterministic Timed Automata

- Impose **bounded variability** of input signals
- Separate TA in two parts
  - **Proposition monitor (PM)** that observes changes in inputs and memorizes events with clocks
    - Deterministic TA by construction
    - Finite number of clock (bounded variability assumption)
  - **Dependent TA (DTA)** that handles a-causality of MTL
    - Generates **discrete** predictions regarding future events
    - Passive use of clocks reset by PM
    - DTA can be **determinized**



Proposition monitor

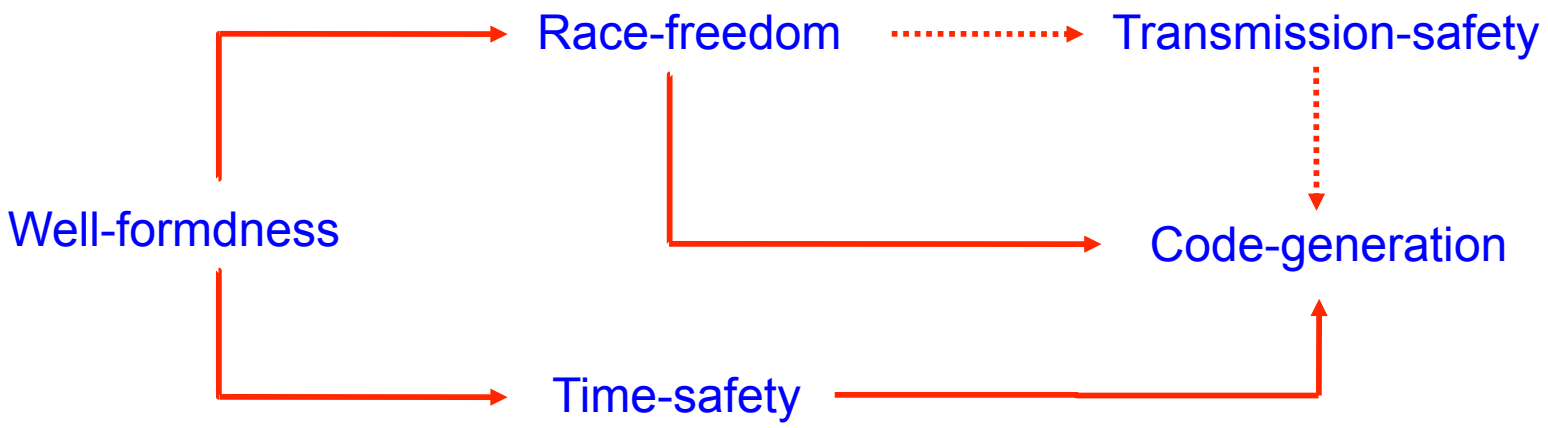


Dependent timed automaton

Distributed, Modular HTL

- Hierarchical Timing Language (HTL)
  - Real-time coordination language for distributed control systems
  - **Modular** syntax and semantics
  - **Time-determinism** is a key property of HTL programs
- **Modularity** = compositionality
  - HTL **compilation** is (quite) modular
  - HTL **distribution** is modular
- A system's I/O behavior is **time-deterministic** if, for all sequences of input values and times, the system always produces unique sequences of output values and times.
  - Time-deterministic = **predictable** behavior

HTL Compilation



- ***Well-formed***, ***race-free***, ***time-safe***, and ***transmission-safe*** HTL programs are **time-deterministic**
- **Modular checks** for well-formdness, race-freedom, time-safety and transmission-safety
  - Except for time-safety check at the top level

HTL Distribution

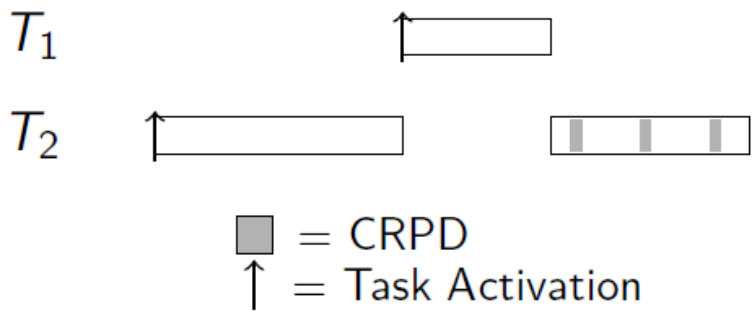
- **Transmission-safety** can be asserted by standard schedulability criteria for a variety of network platforms
  - (e.g. TDMA, FTT-CAN).
- **Time-safety** analysis and **code generation** can be done separately per host
- **Overall:** scalable distribution

Advances in WCET Determination

- Based on the Reineke-Metrics, the predictability of caches with FIFO replacement has been clarified.
- The powerset-domain for pipeline analysis war made more efficient by developing a compact, symbolic representation of sets of pipeline states using BDDs
- Making dynamic memory allocation timing predictable:
  - Predictable allocator
  - Transformation into static allocation
- Operating mode analysis
  - Identifying operating modes
- *Incorporation of context switches:* Theoretical work and prototype implementations

Cache Related Preemption Delay

- In case of preemption: preempting task might evict cache-content of preempted task
- *Cache Related Preemption Delay (CRPD):* cost of additional misses due to preemption



Determination of CRPD

- A memory block *m* at program point *P* is a useful cache block, if it
  - may be cached at *P,*
  - may be reused at point *Q* reached from *P* without being evicted on that path
- UCB analysis safely overapproximates context switch costs
- WCET analysis safely overapproximates execution time
- Very pessimistic results if combined

WCET Analysis vs. UCB Analysis

- WCET Analysis
  - uses *underapproximation* of cache-content (must)
  - only predicts cache-hit
- UCB Analysis
  - uses *overapproximation* of cache-content (may)
  - Predicts additional cache misses
- Some cache misses are counted twice

Definitely-Cached UCBs

- A memory block *m* at program point *P* is a useful cache block, if it
  - must be cached at *P* and on the path to its reuse,
  - may be reused at point *Q* reached from *P*.
- UCB analysis possibly underapproximates context switch costs
- No cache miss counted twice
- Overapproximation (WCET) subsumes underapproximation (UCB)
- Tight and safe results if combined

CRPD Computation
Resilience of a UCB

- Not every UCB leads to an additional cache miss
- Depends on disturbance of preempting task
- Resilience: Amount of disturbance a UCB survives without preemption-induced cache misses
- Further reduction of CRPD bounds

Fixed-Priority Multiprocessor Scheduling with Liu & Layland's Utilization Bound

Nan Guan, Martin Stigge, Wang Yi
Uppsala University, Sweden

Liu and Layland's Utilization Bound

- Liu and Layland's utilization bound for single-processor scheduling [Liu1973]

(the 19th most cited paper in computer science)

$$\Theta(N) = N(2^{\frac{1}{N}} - 1)$$

: the number of tasks, N

$$N \to \infty, \quad \Theta(N) \doteq 69.3\%$$

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

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

Open Problem (for many years)

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

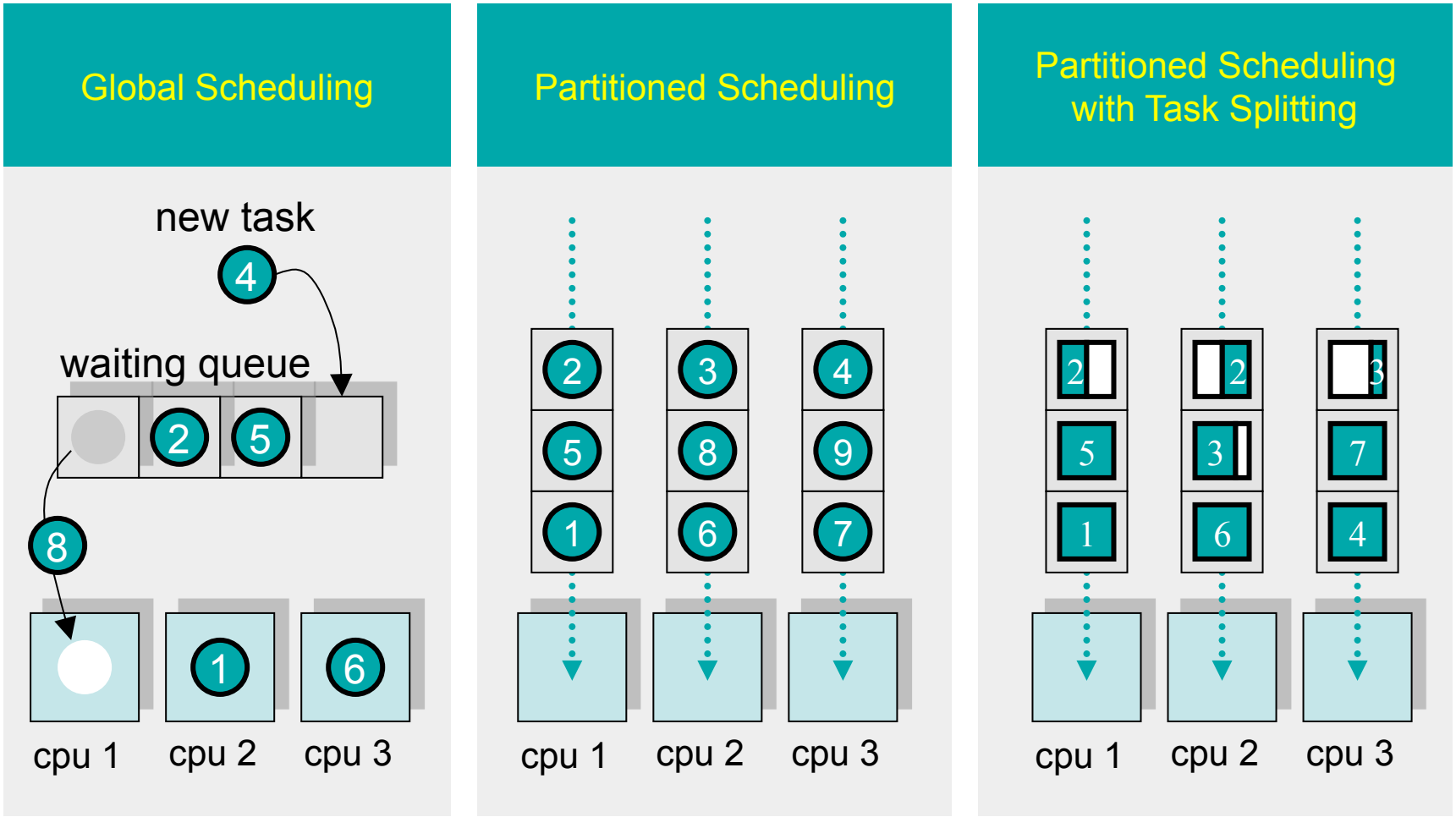
$$\frac{\sum C_i/T_i}{M} \leq N(2^{1/N} - 1)$$

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

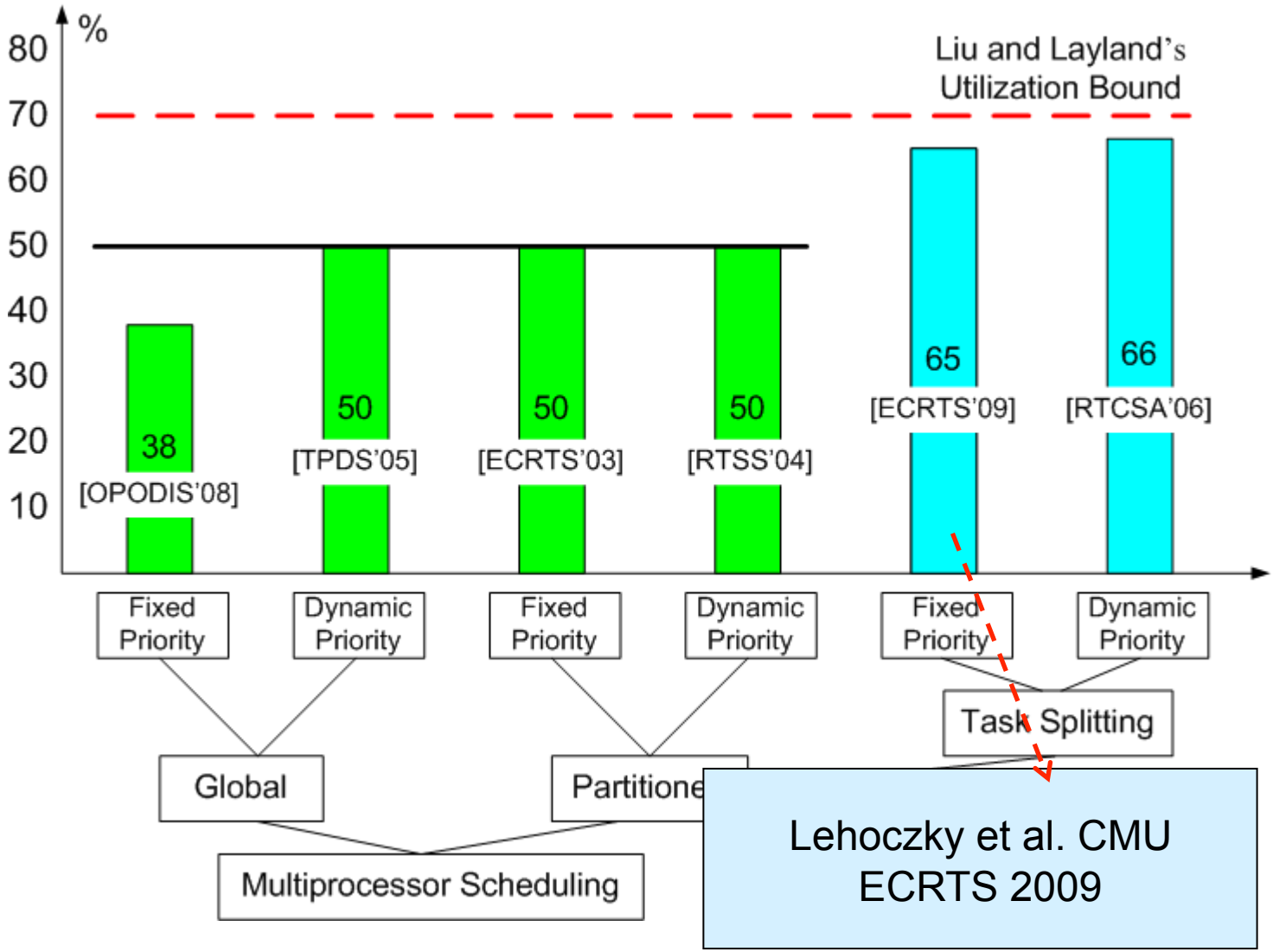
?

number of processors

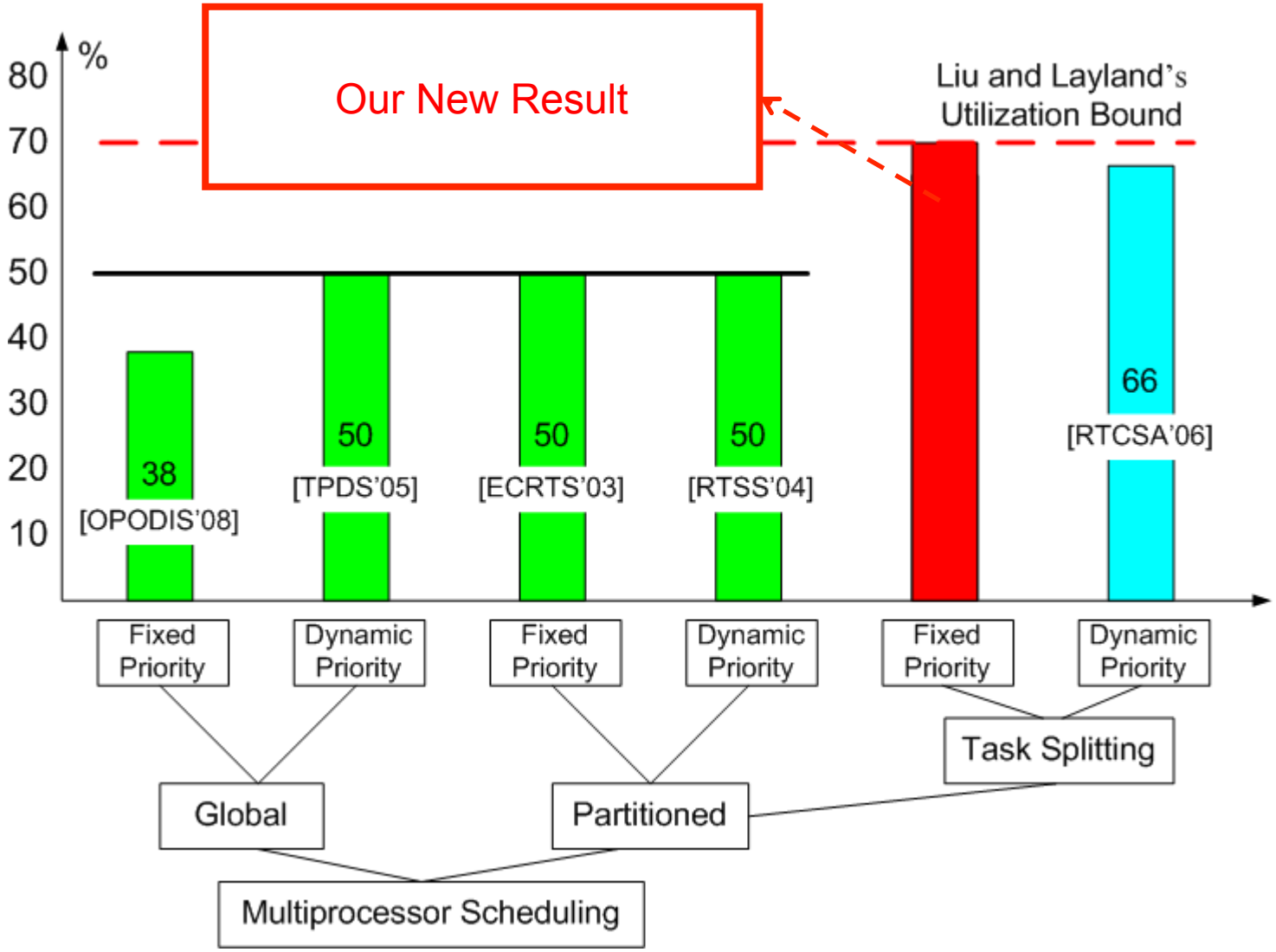
Multiprocessor Scheduling



Best Known Results

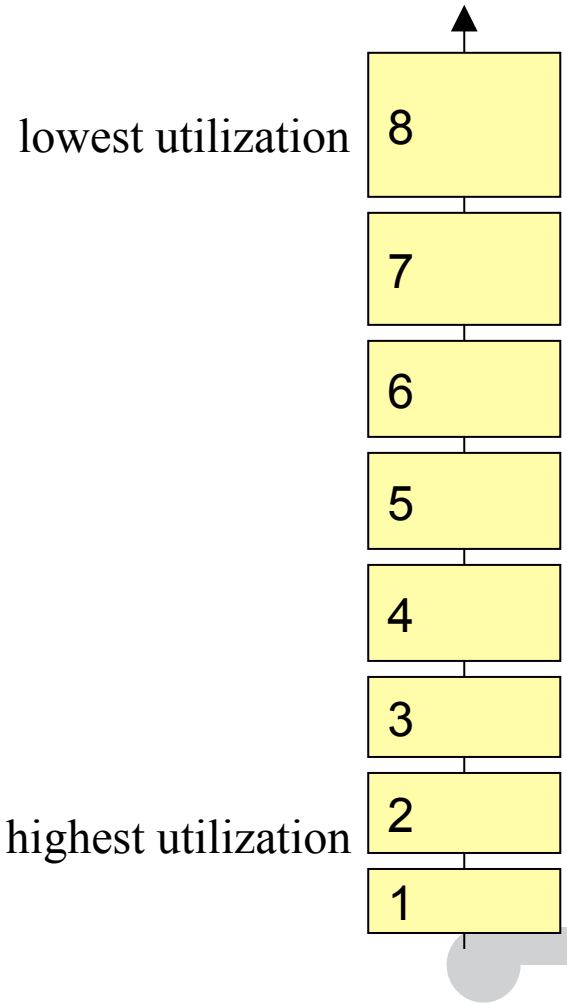


Best Known Results



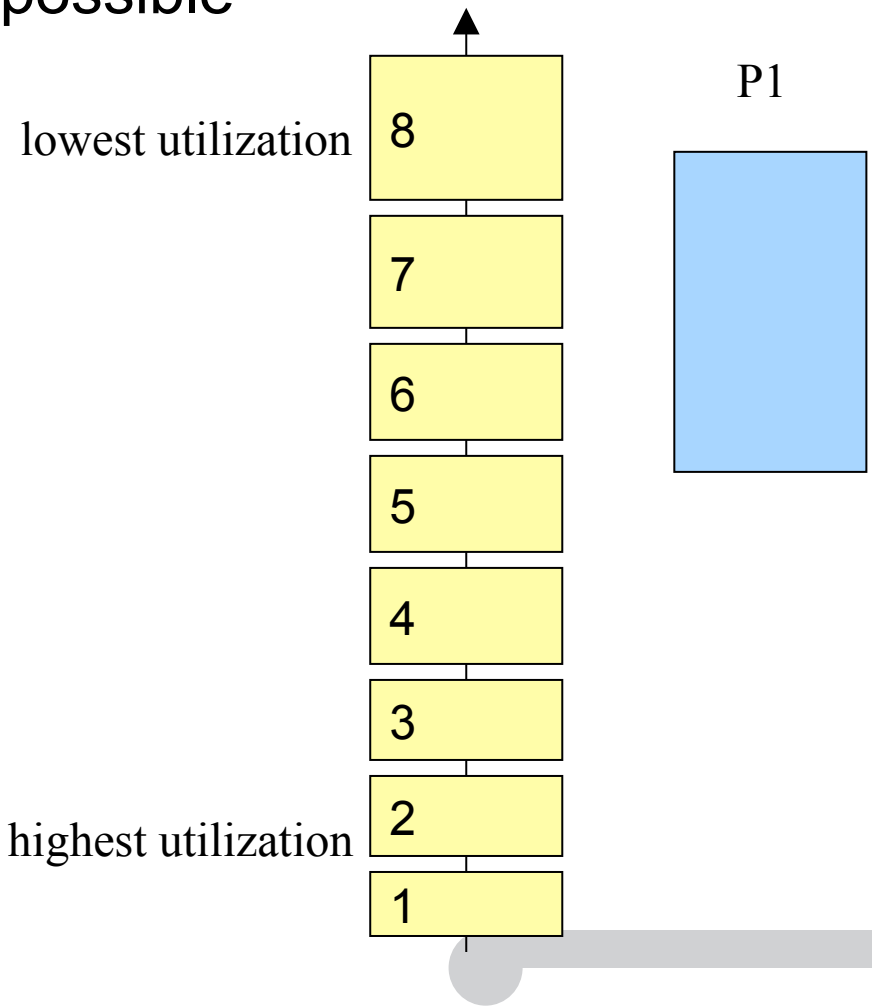
Lehoczky's Algorithm[ECRTS'09]

- sort all task in decreasing order of utilization



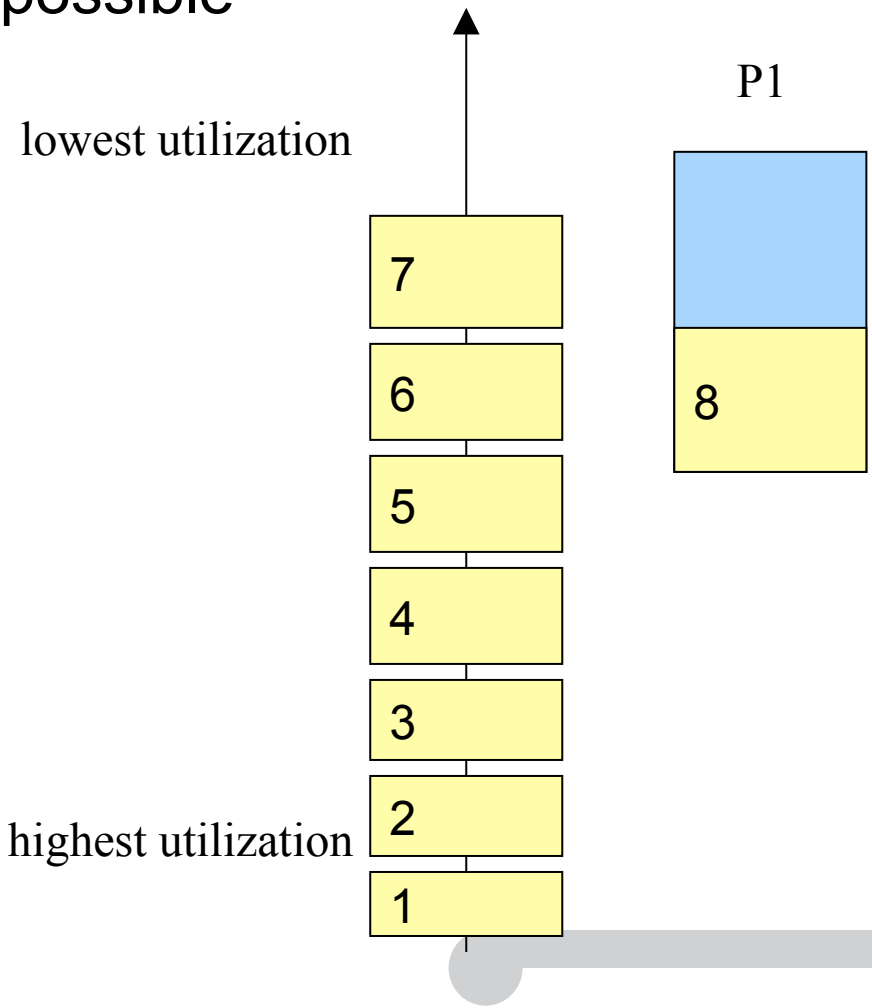
Lehoczky's Algorithm [ECRTS'09]

- pick up one processor, to assign as many tasks as possible



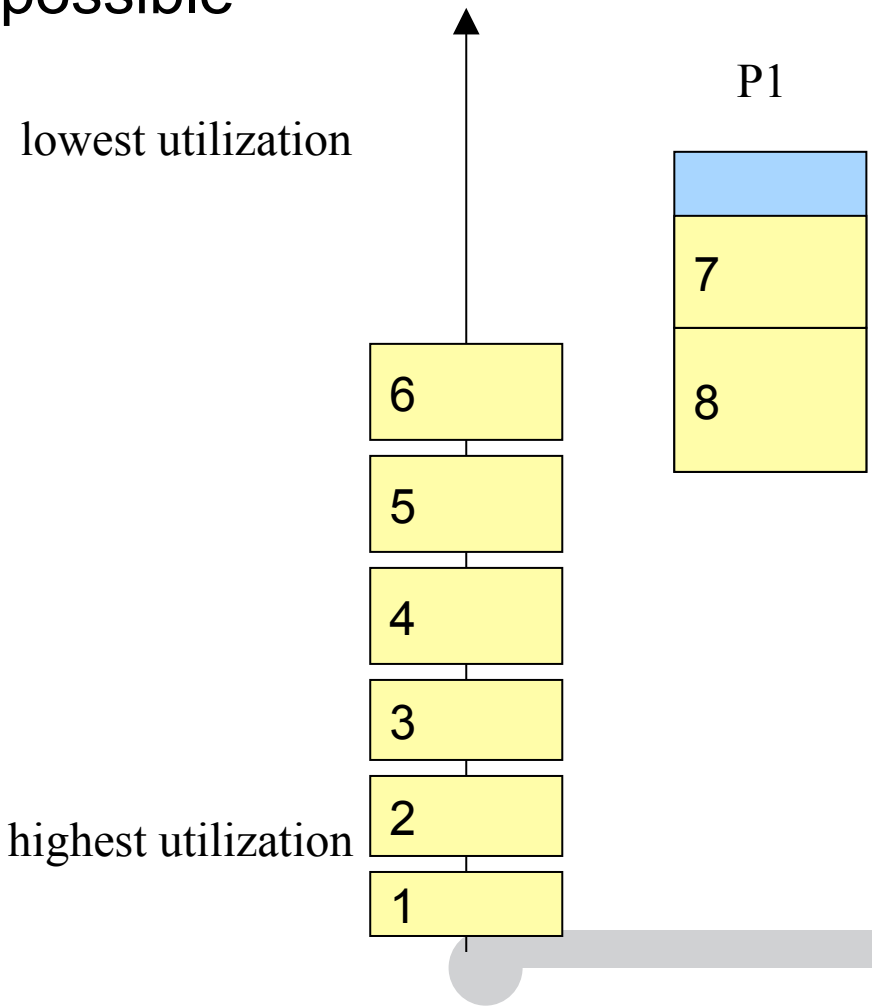
Lehoczky's Algorithm [ECRTS'09]

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



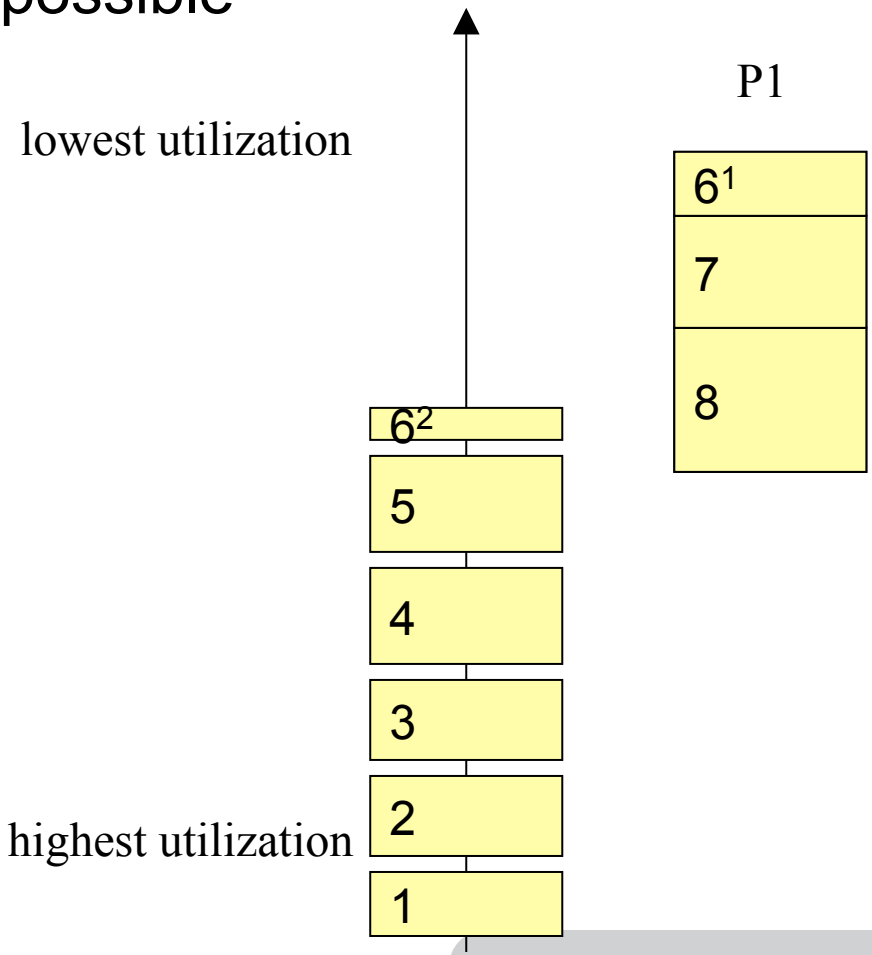
Lehoczky's Algorithm [ECRTS'09]

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



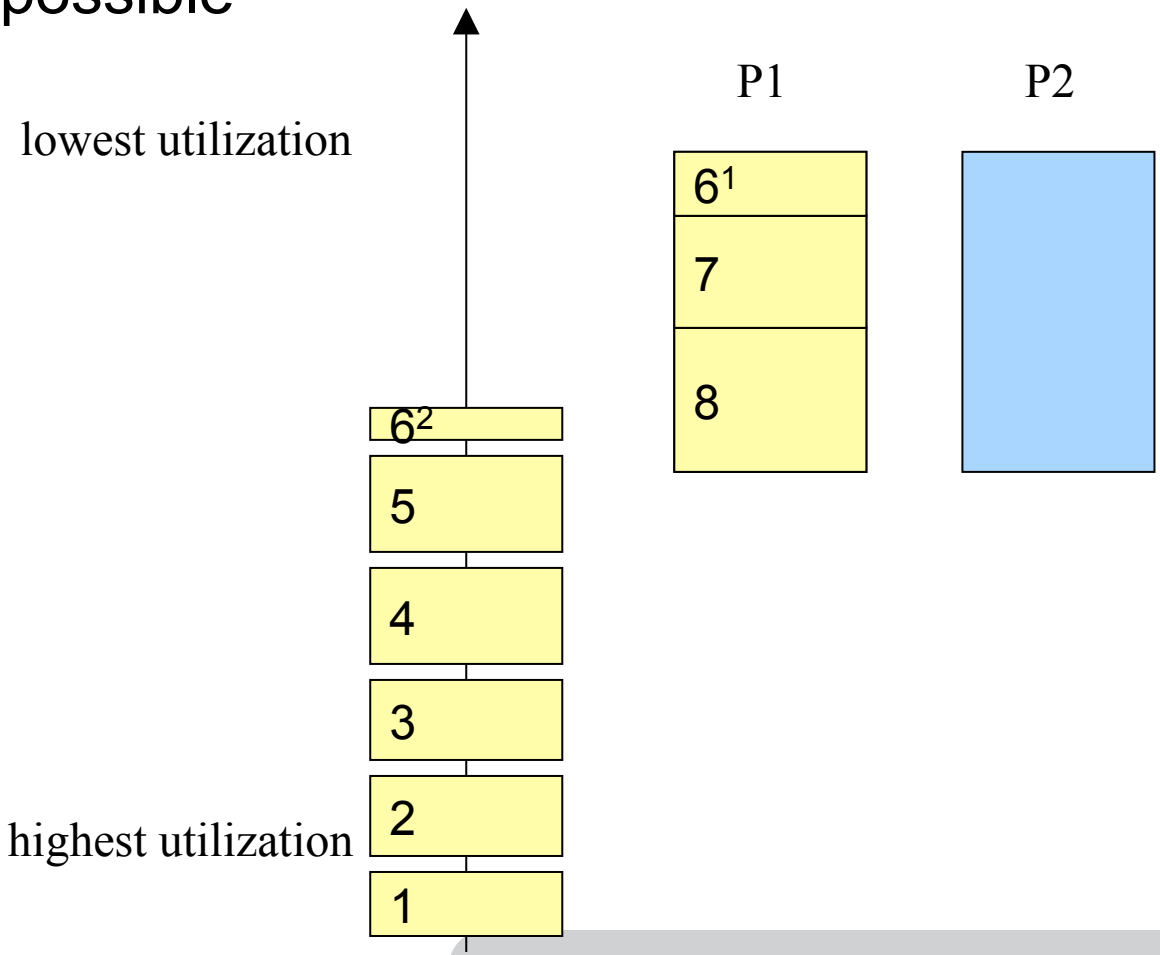
Lehoczky's Algorithm [ECRTS'09]

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



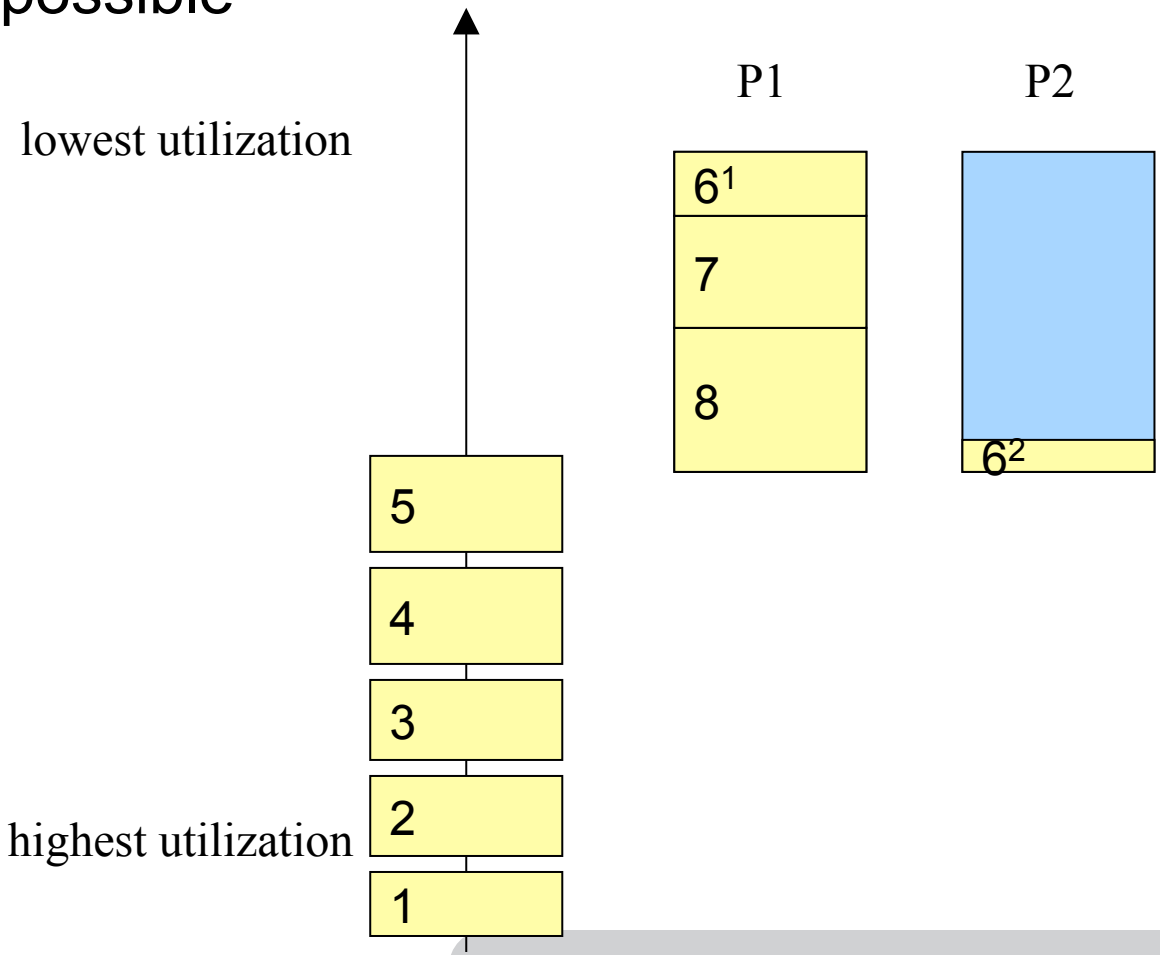
Lehoczky's Algorithm [ECRTS'09]

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



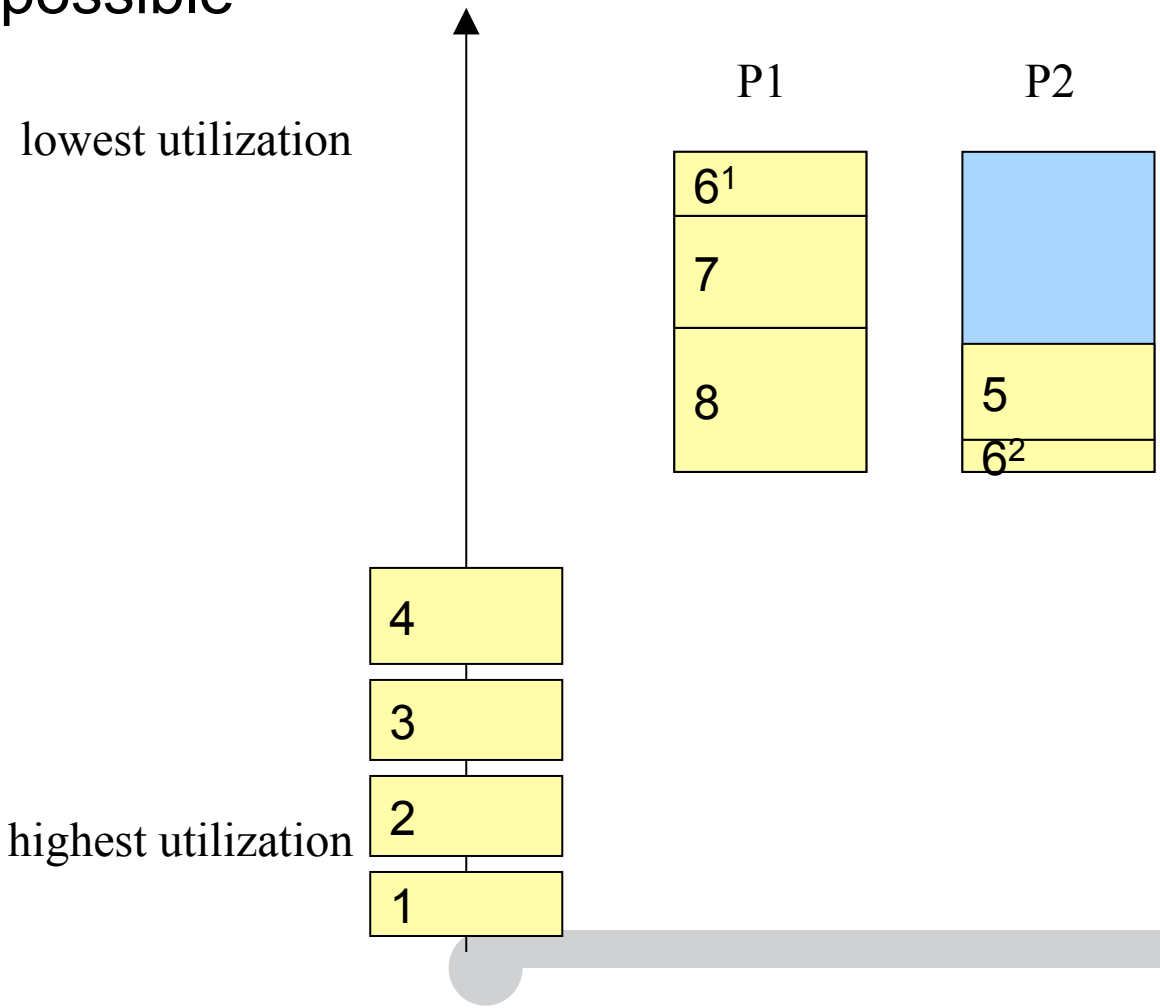
Lehoczky's Algorithm [ECRTS'09]

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



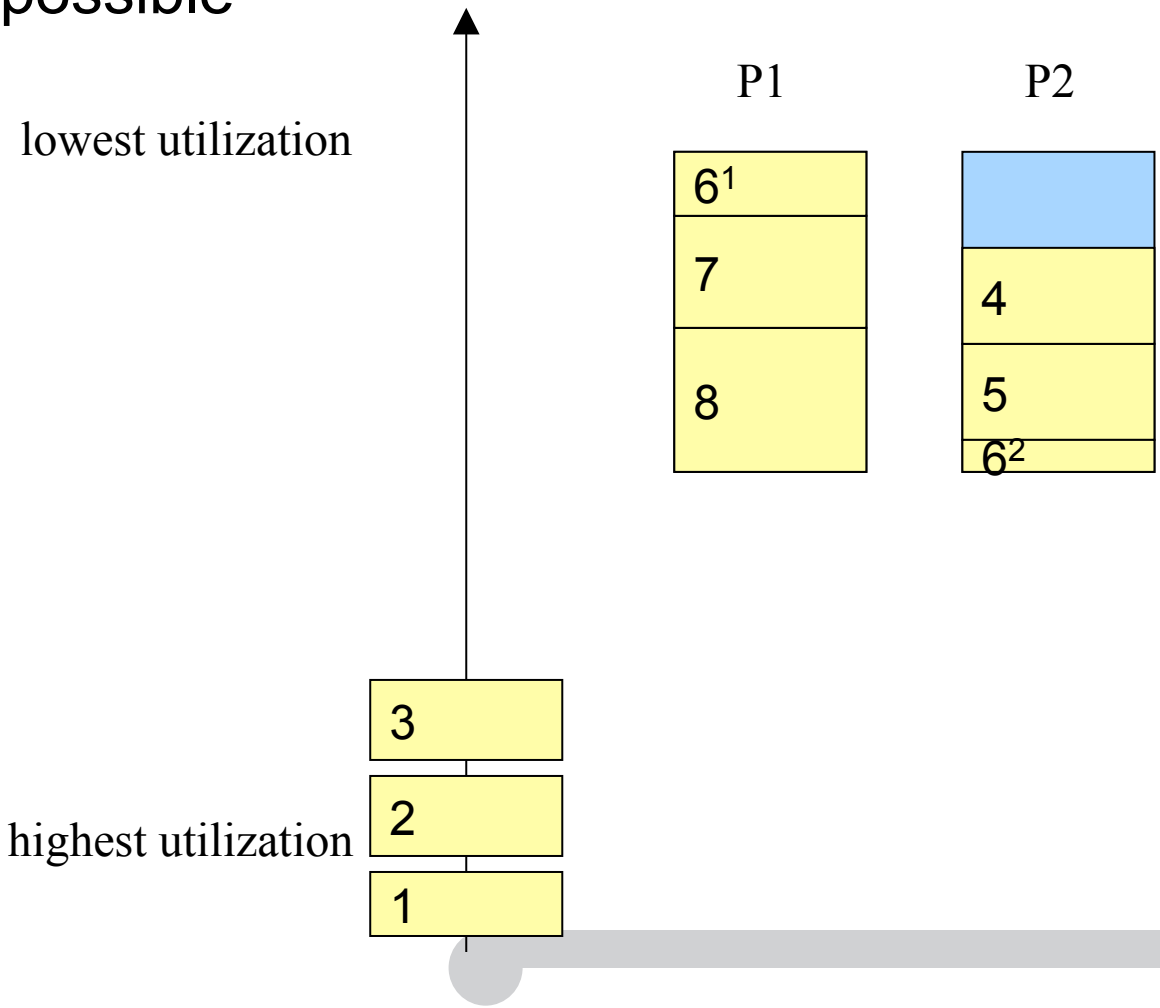
Lehoczky's Algorithm [ECRTS'09]

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



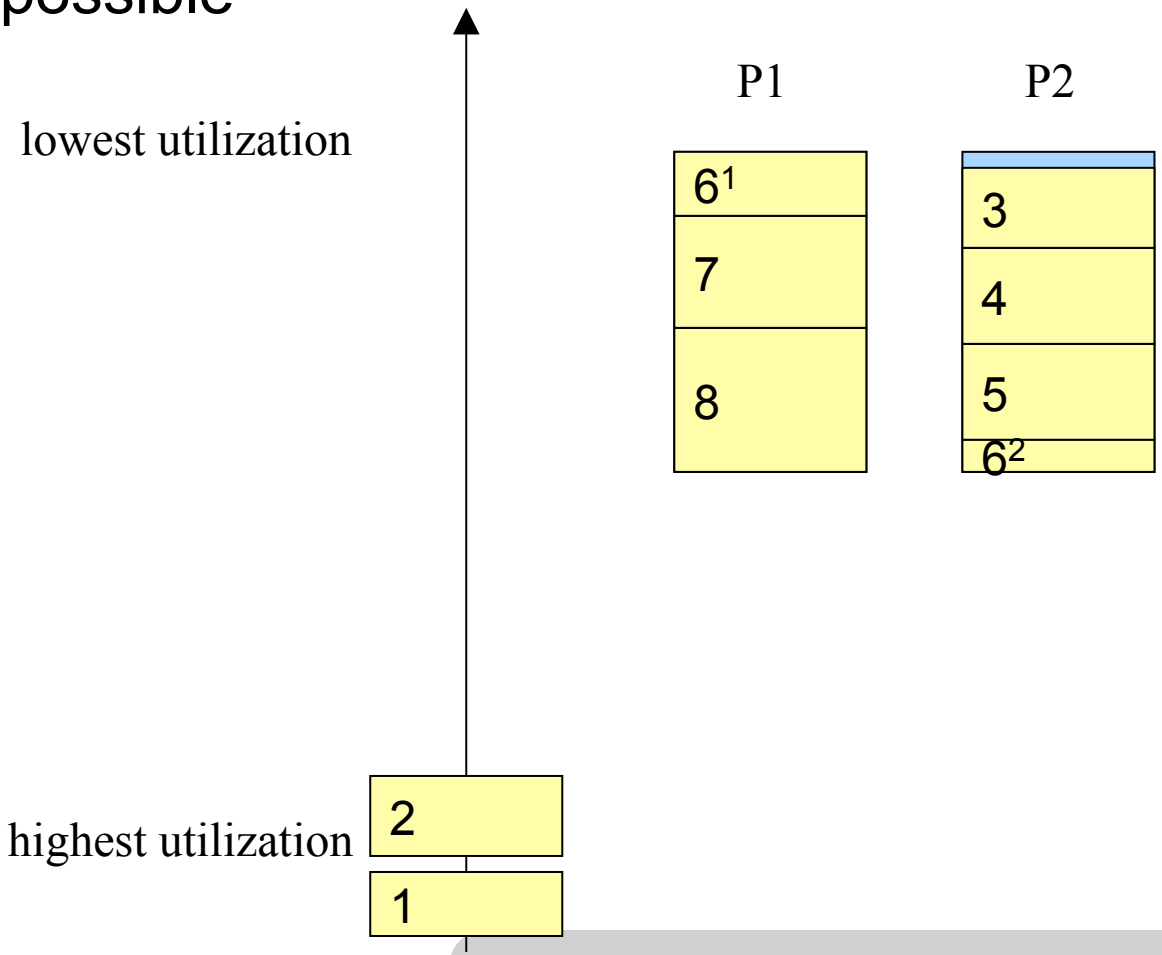
Lehoczky's Algorithm [ECRTS'09]

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



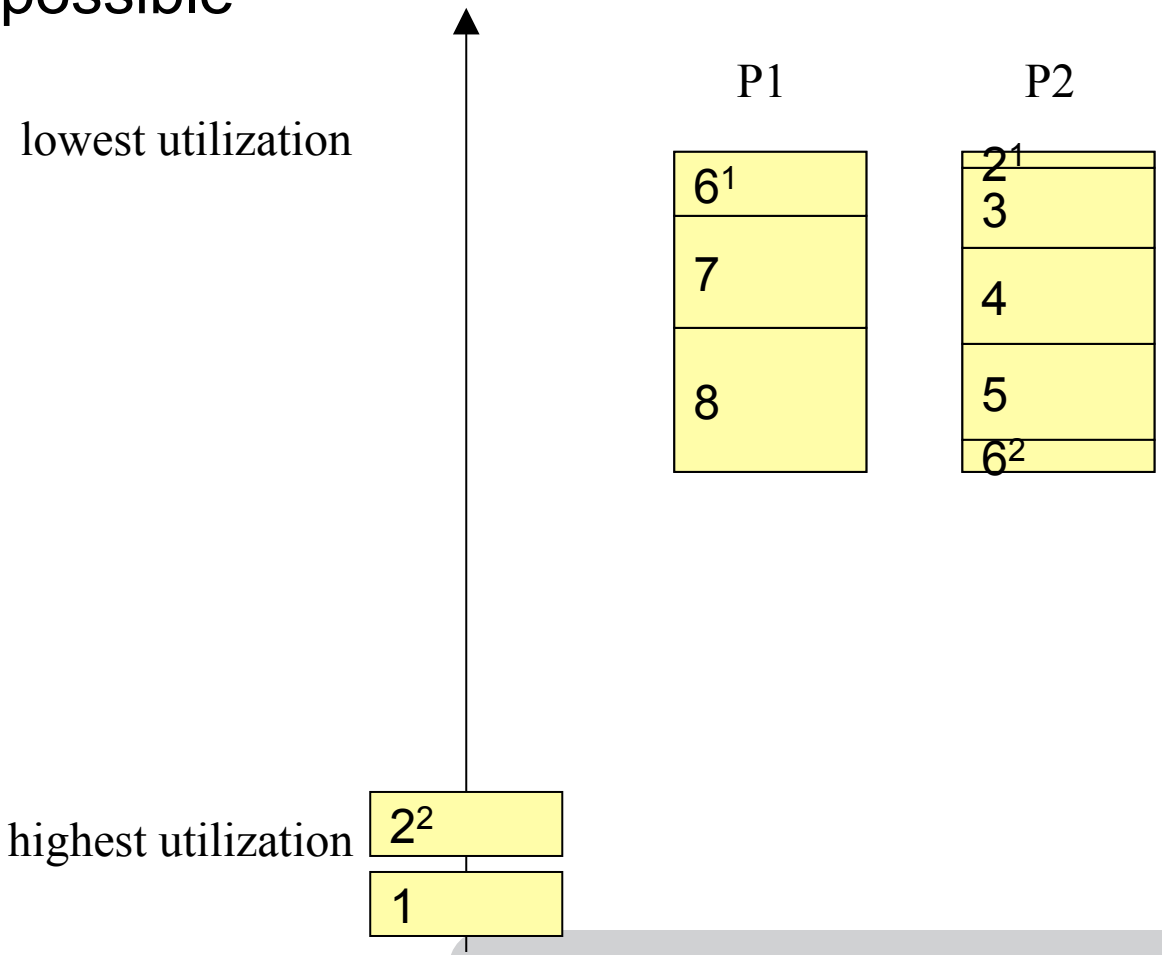
Lehoczky's Algorithm [ECRTS'09]

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



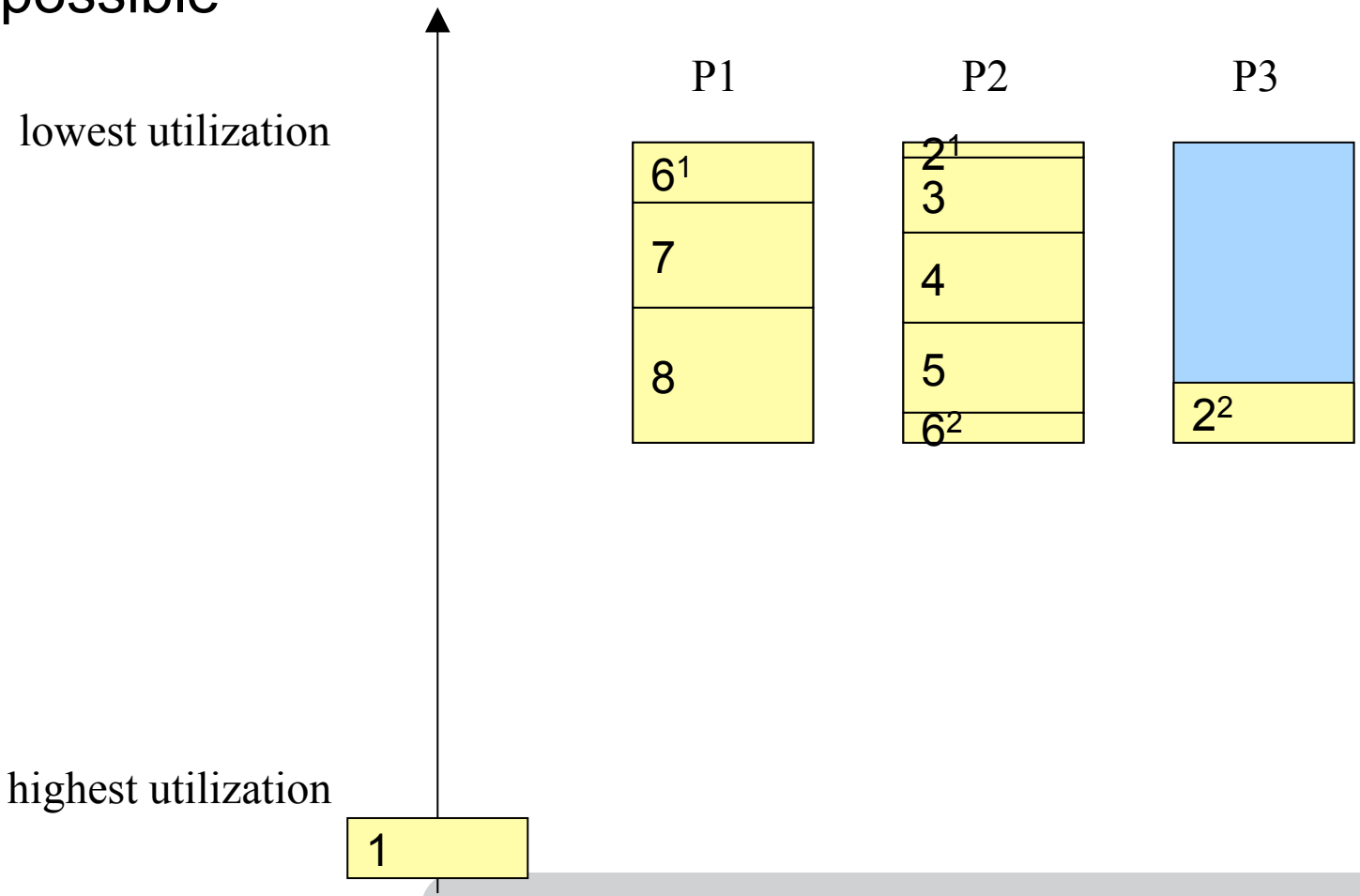
Lehoczky's Algorithm [ECRTS'09]

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



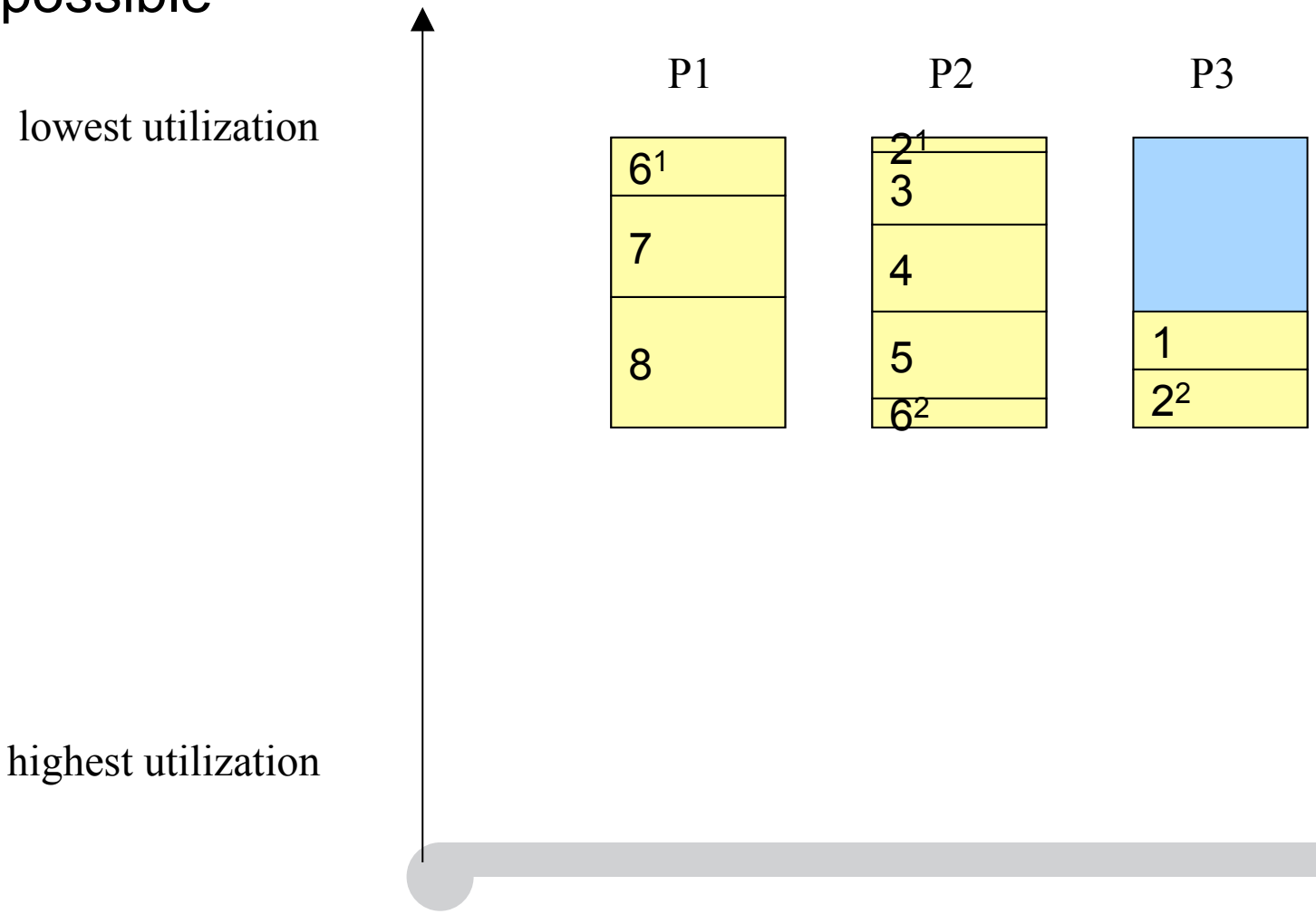
Lehoczky's Algorithm [ECRTS'09]

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



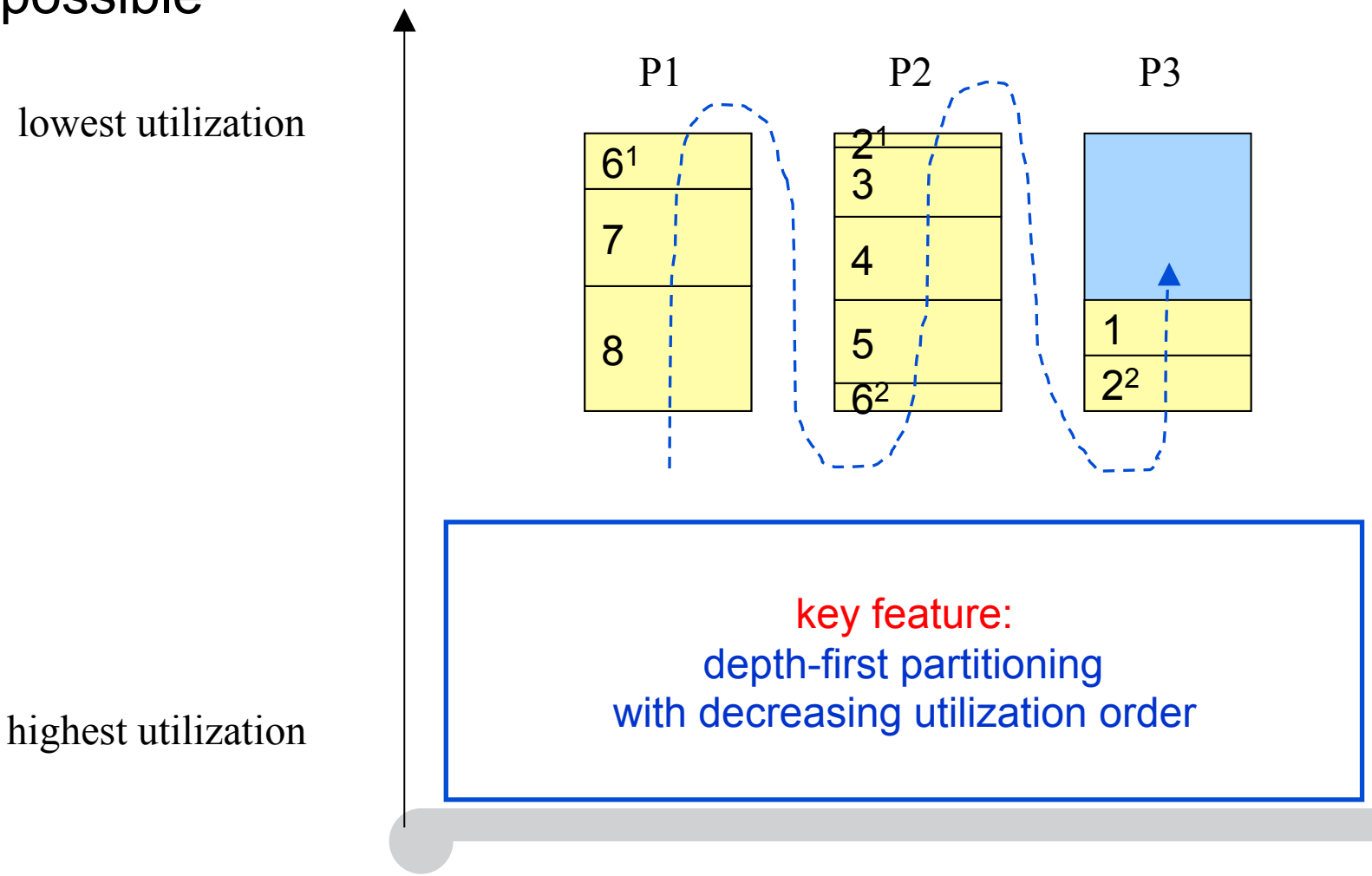
Lehoczky's Algorithm [ECRTS'09]

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



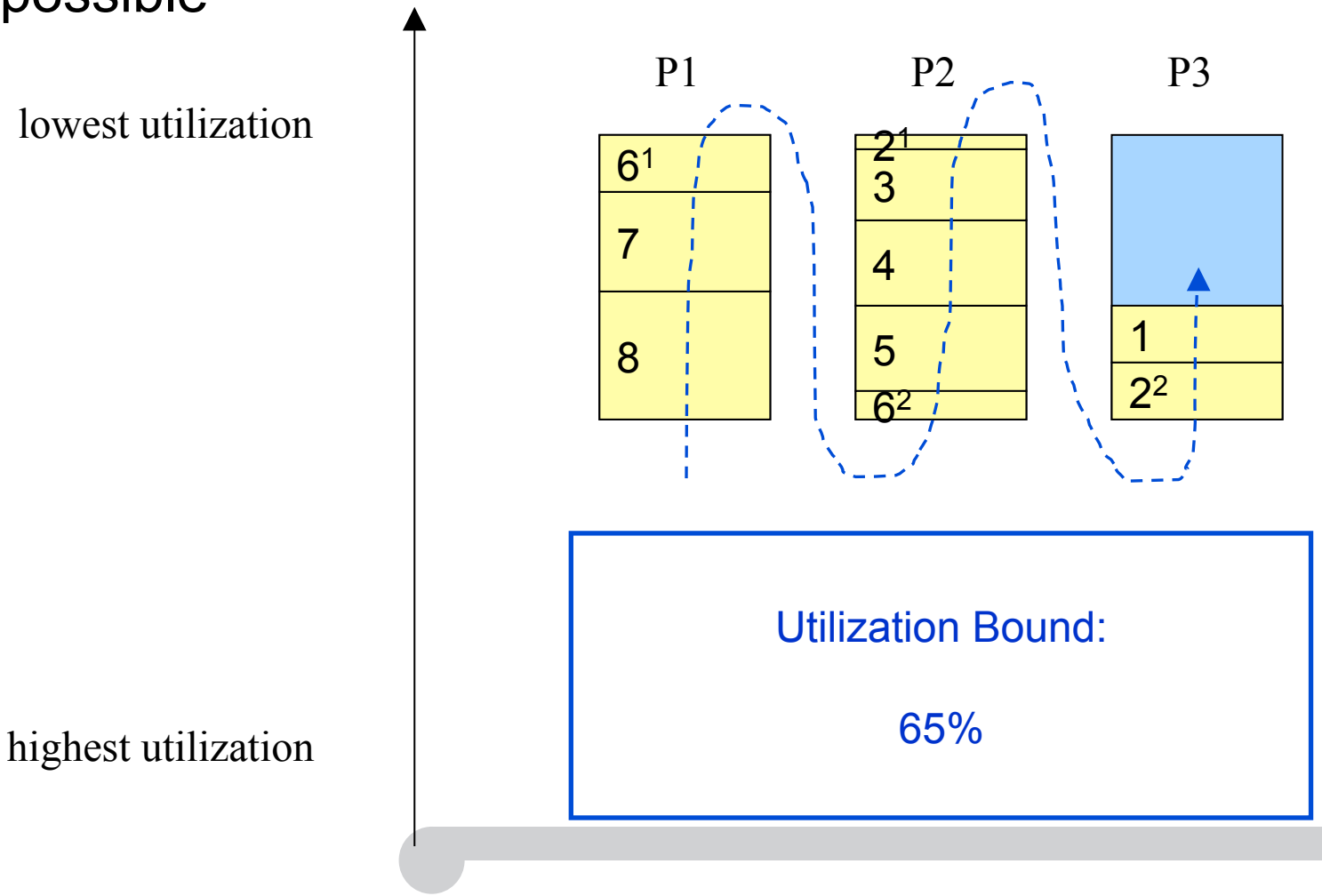
Lehoczky's Algorithm [ECRTS'09]

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



Lehoczky's Algorithm [ECRTS'09]

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



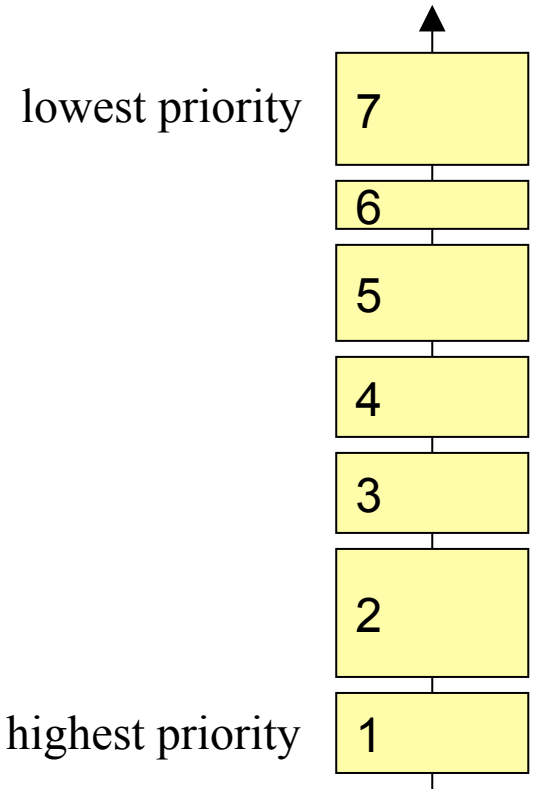
Our Algorithm

width-first partitioning

with increasing priority order

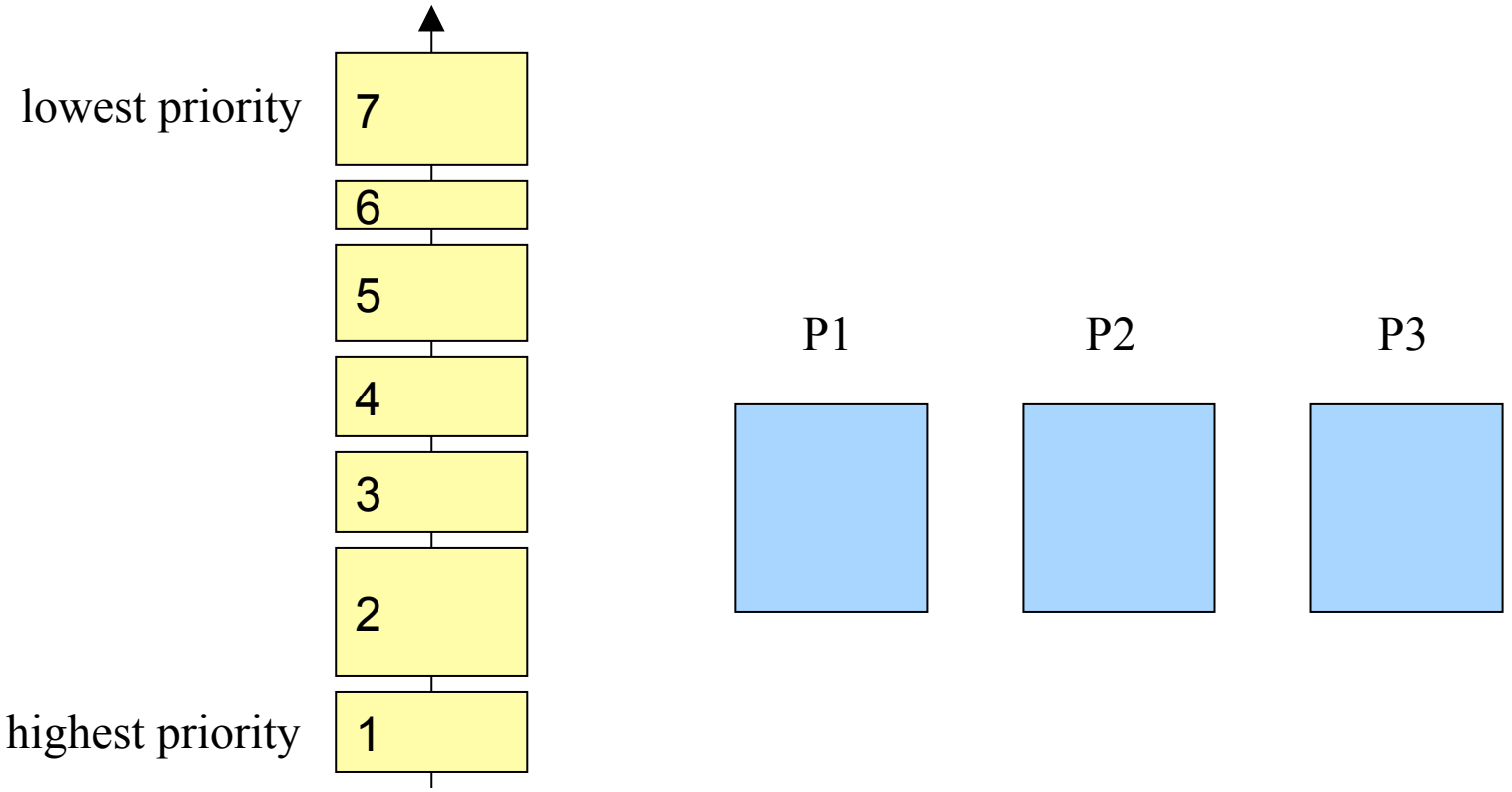
Our Algorithm

- sort all tasks in increasing priority order



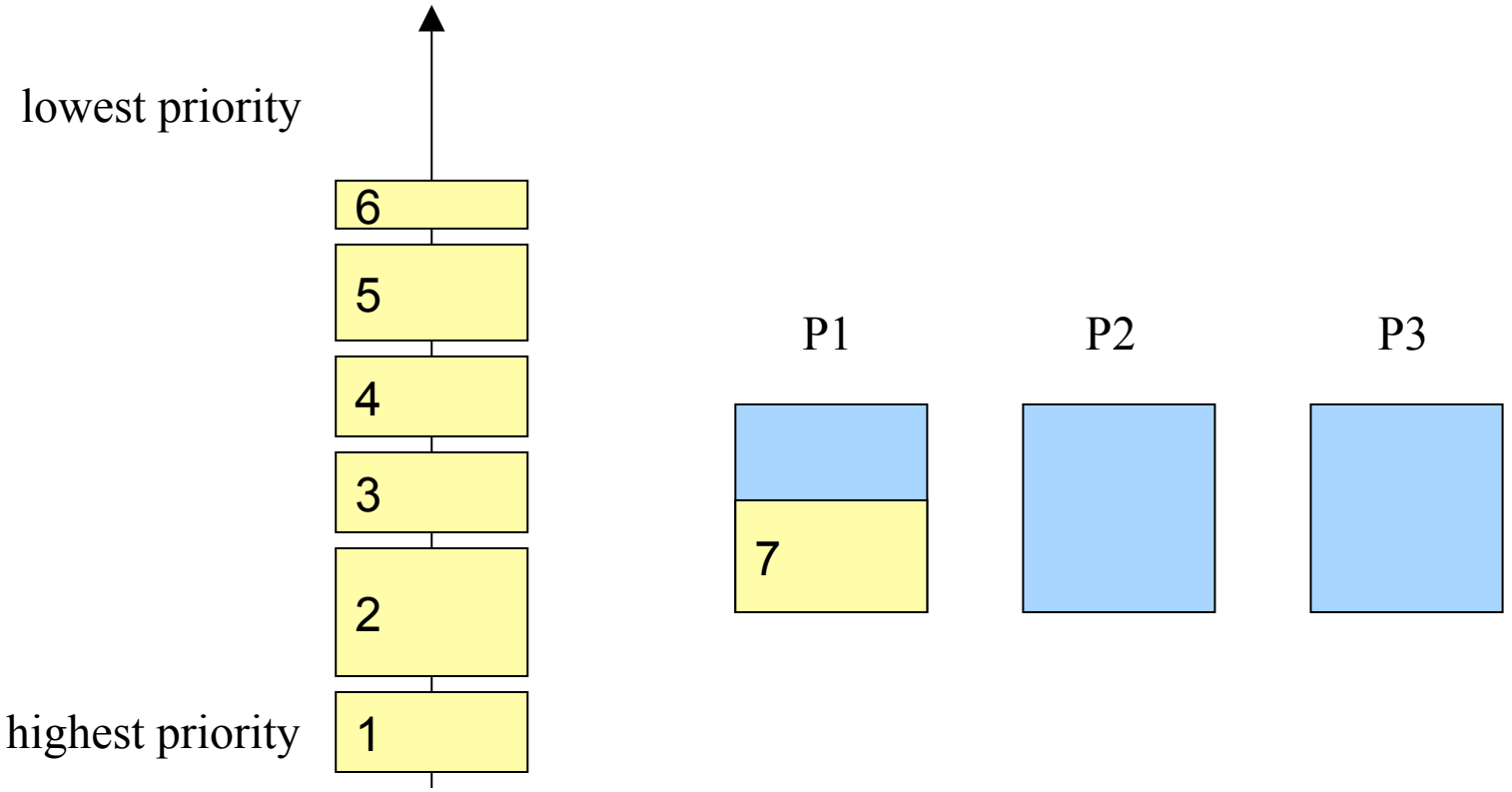
Our Algorithm

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



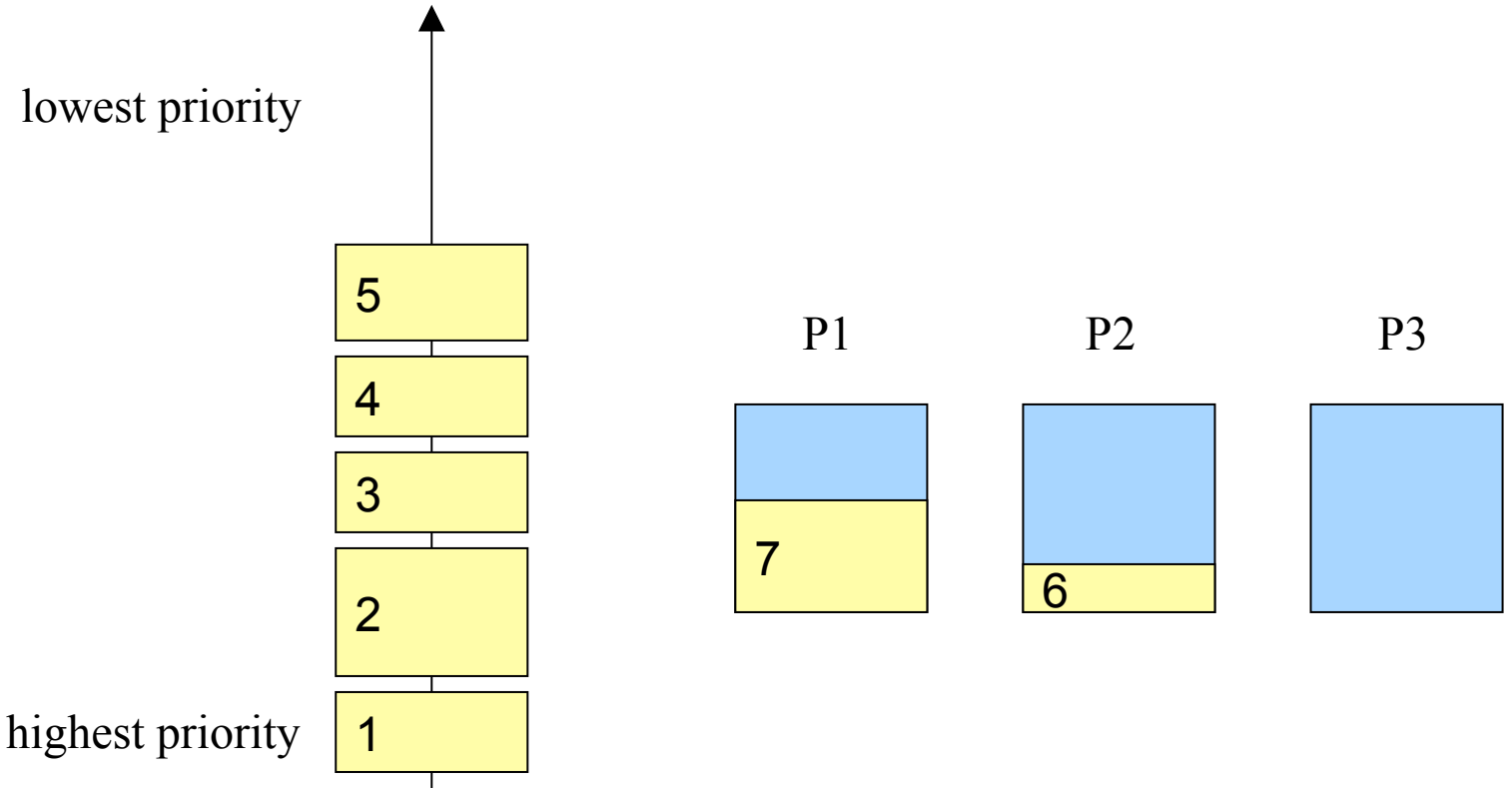
Our Algorithm

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



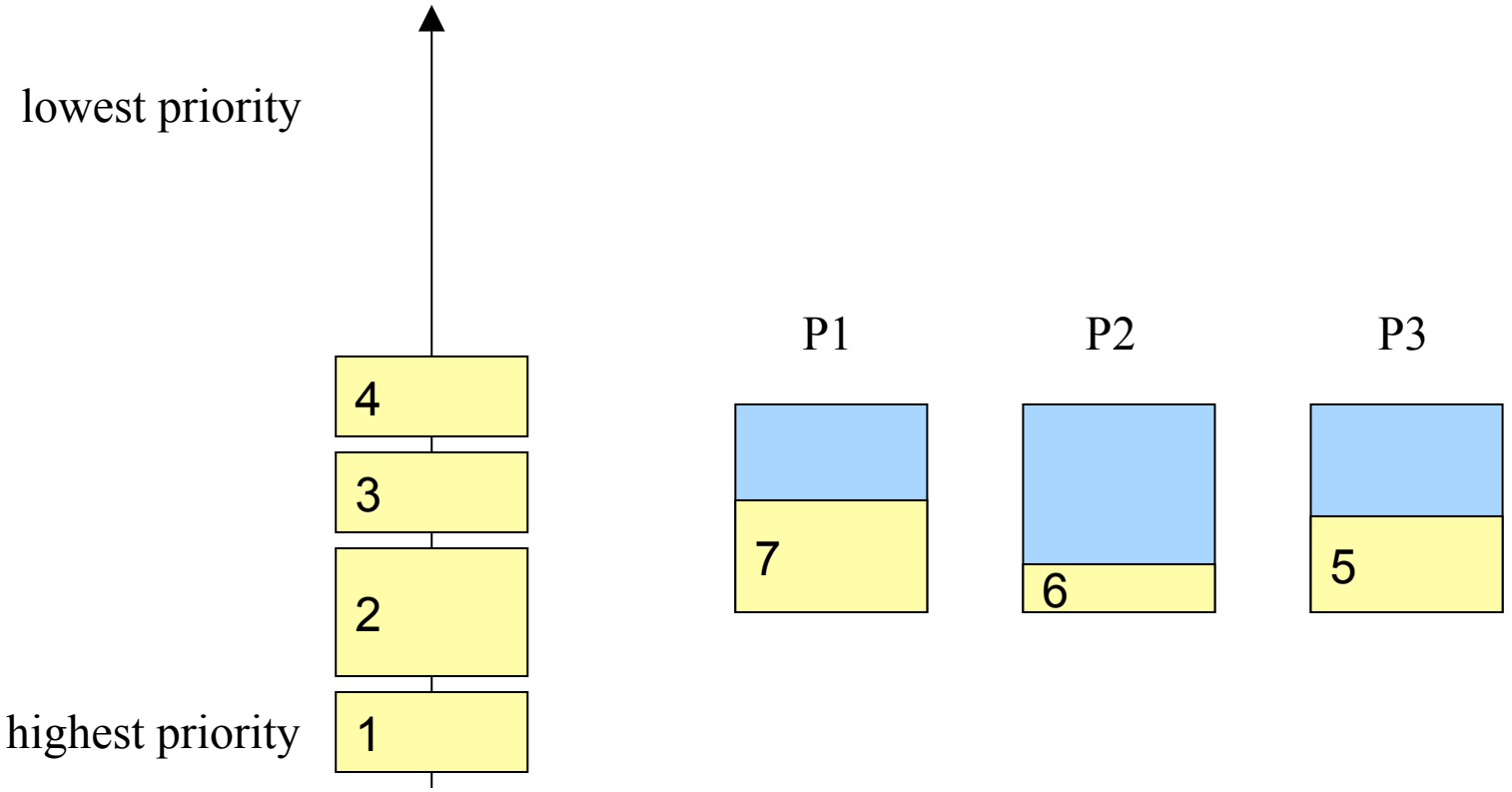
Our Algorithm

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



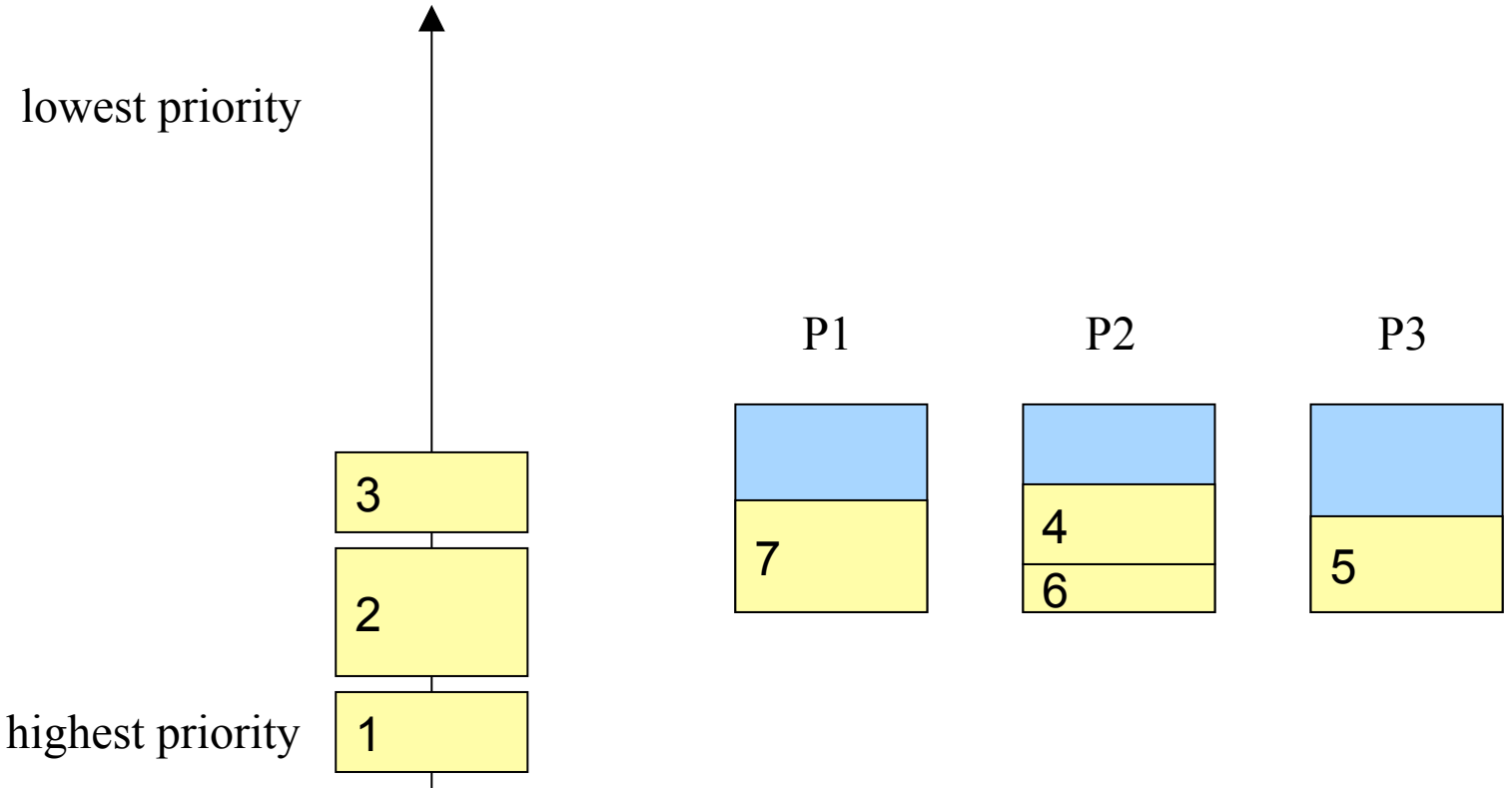
Our Algorithm

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



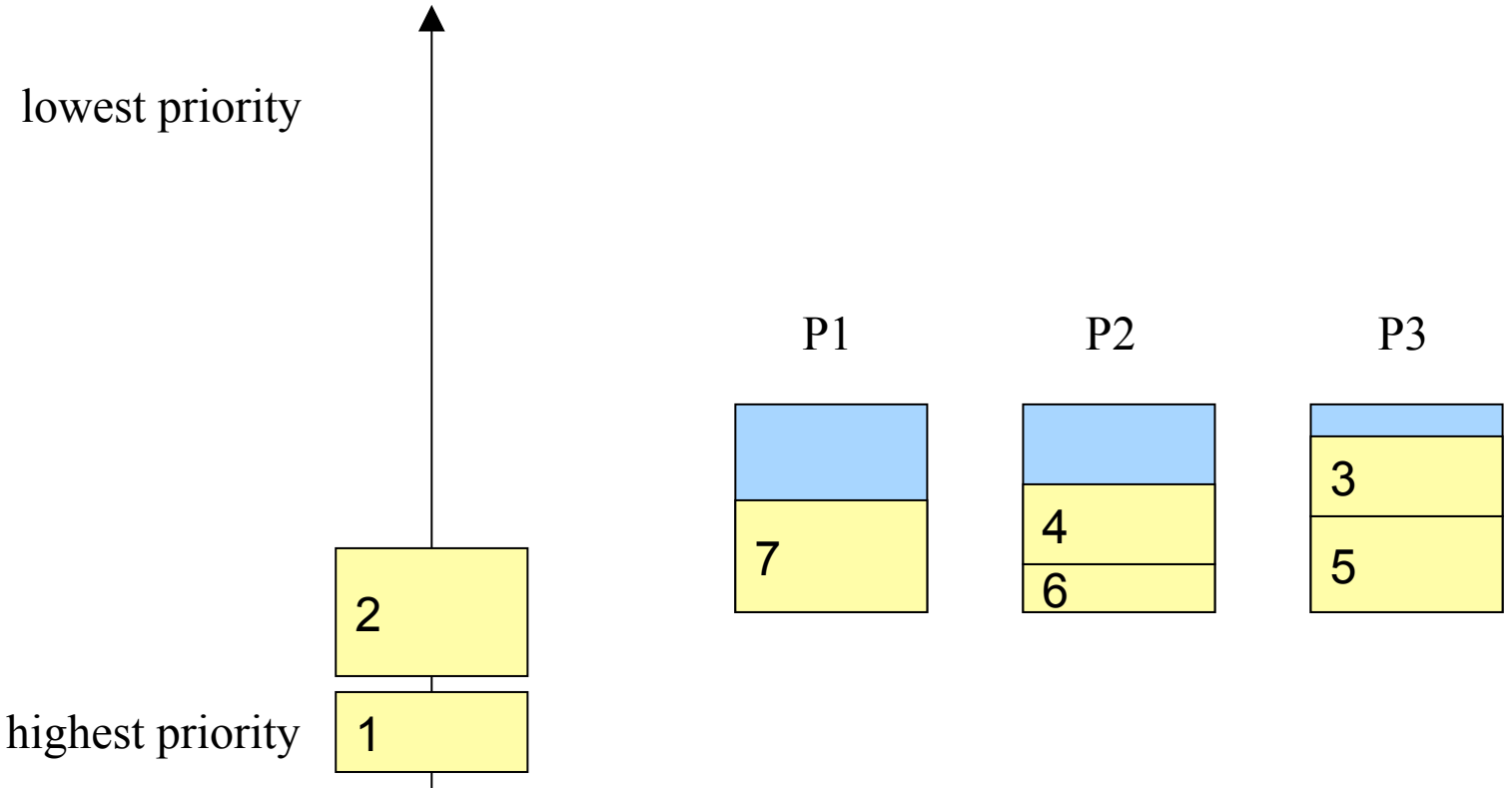
Our Algorithm

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



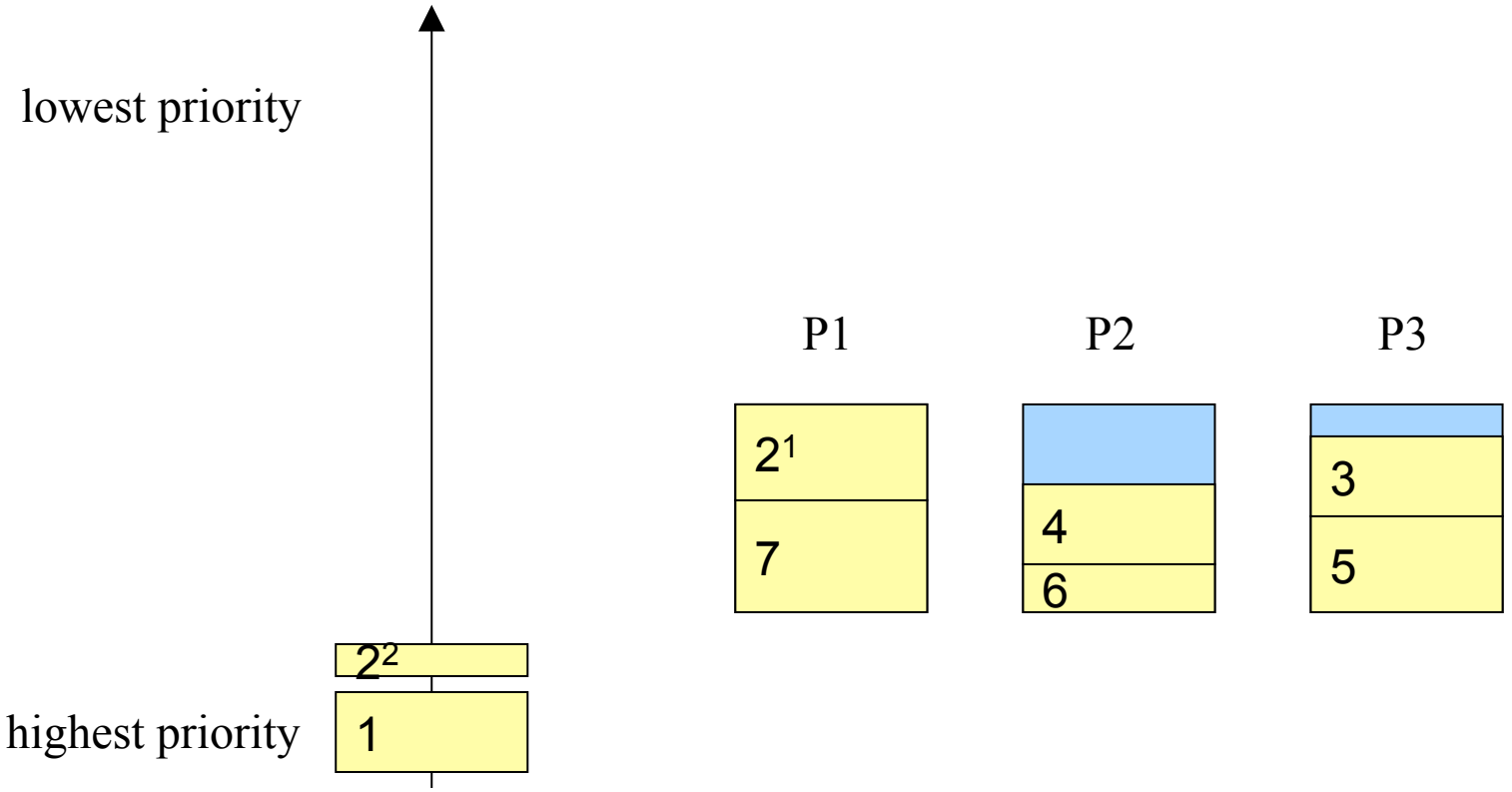
Our Algorithm

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



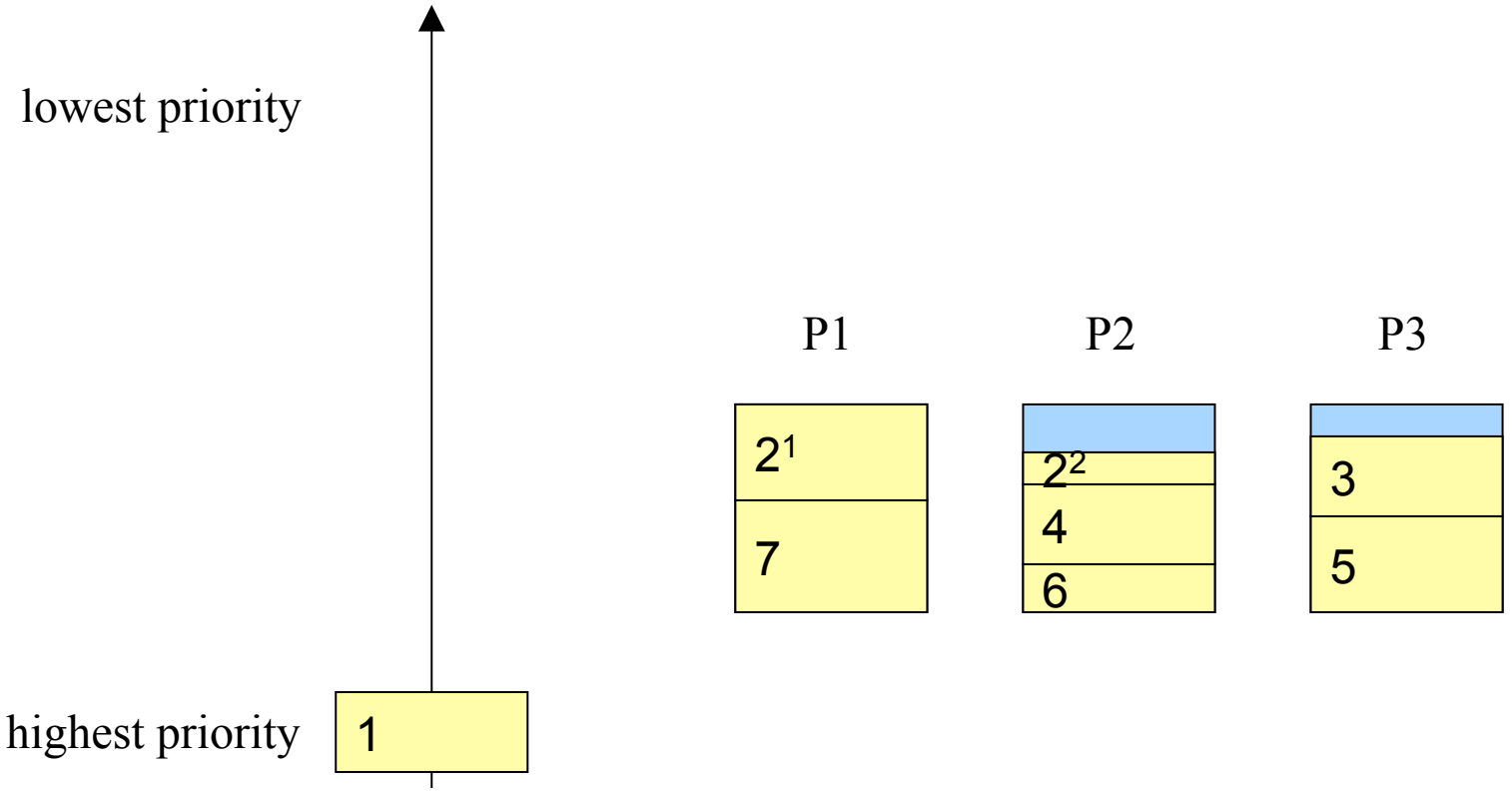
Our Algorithm

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



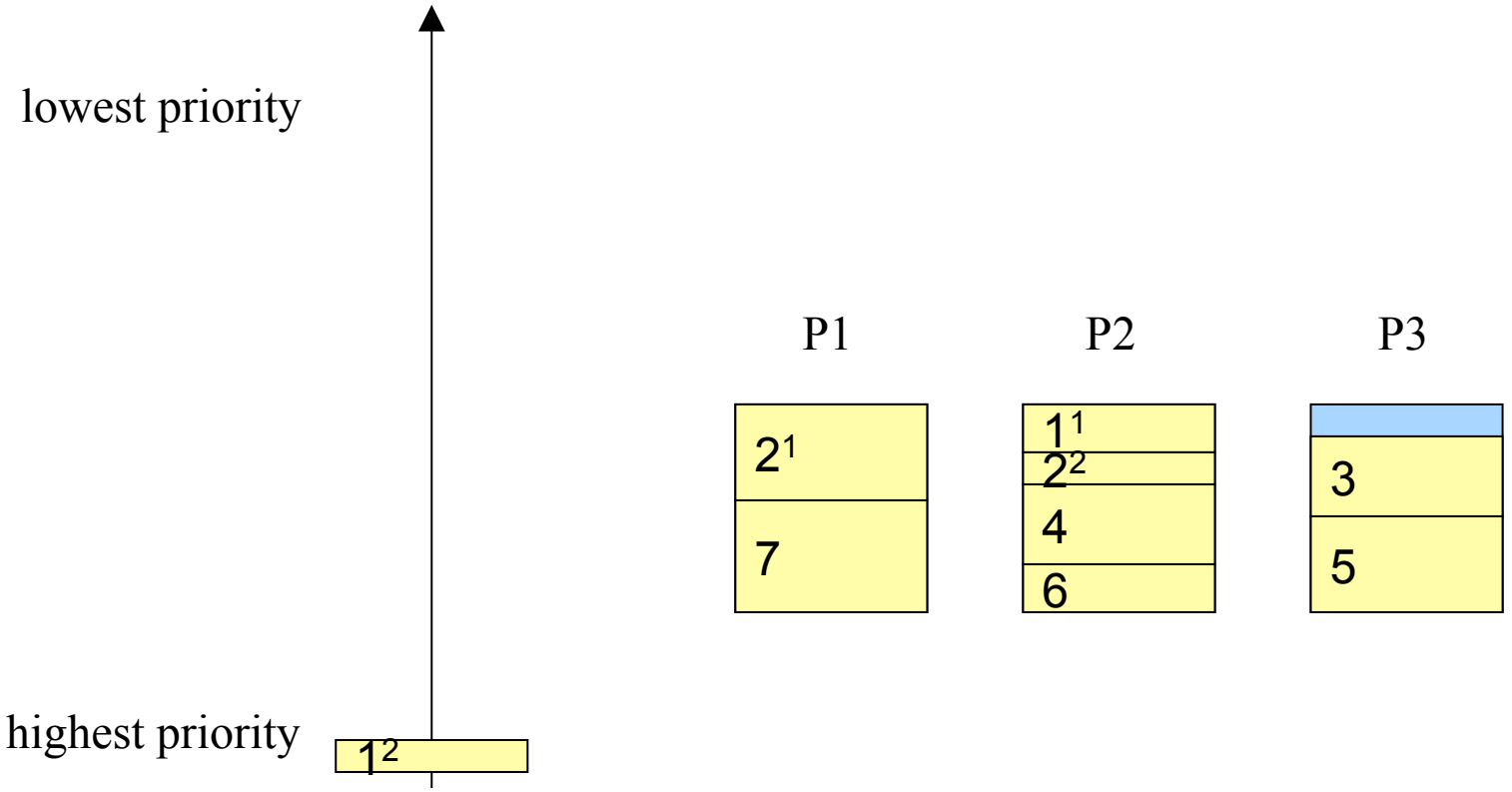
Our Algorithm

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



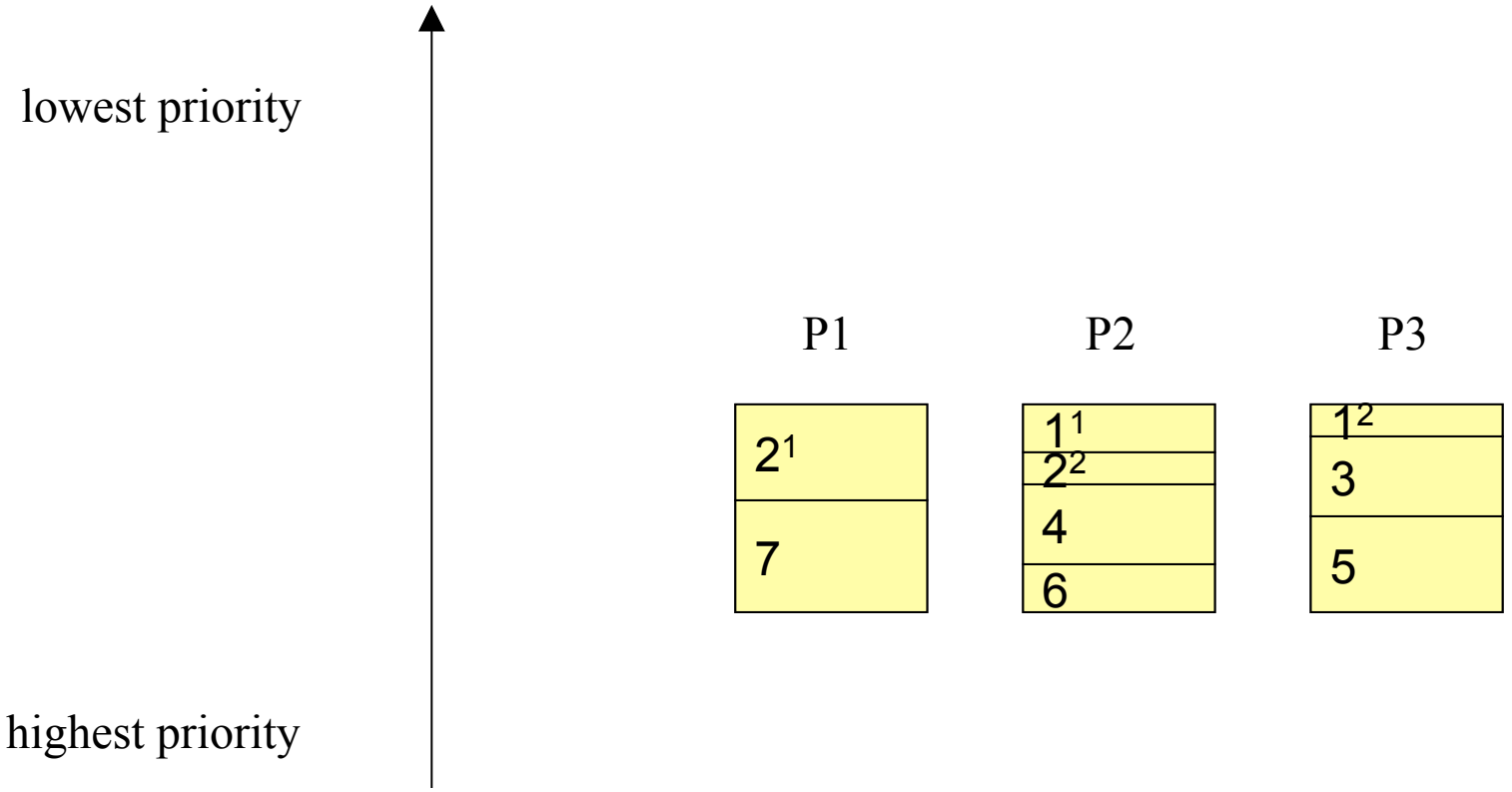
Our Algorithm

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



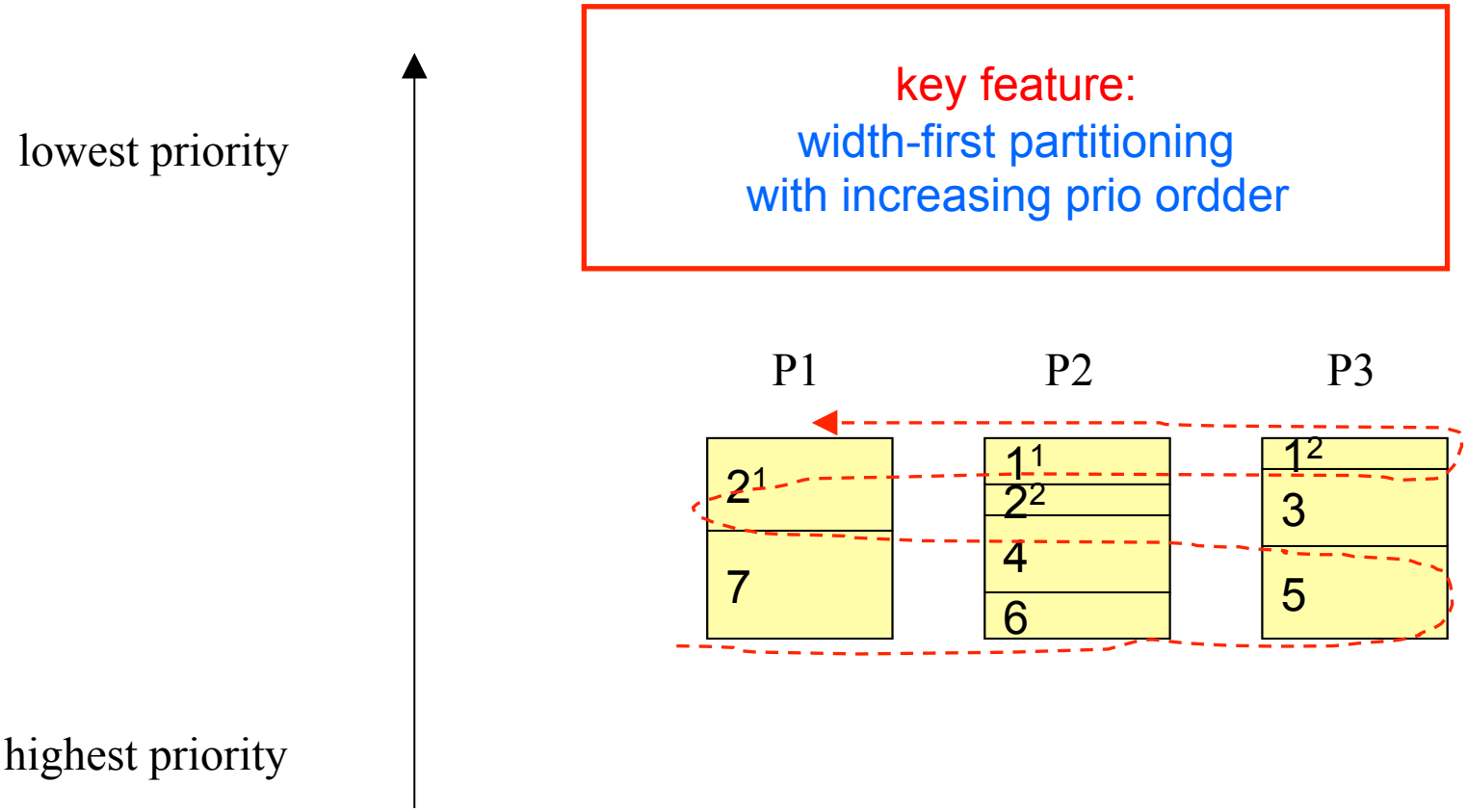
Our Algorithm

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



Our Algorithm

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

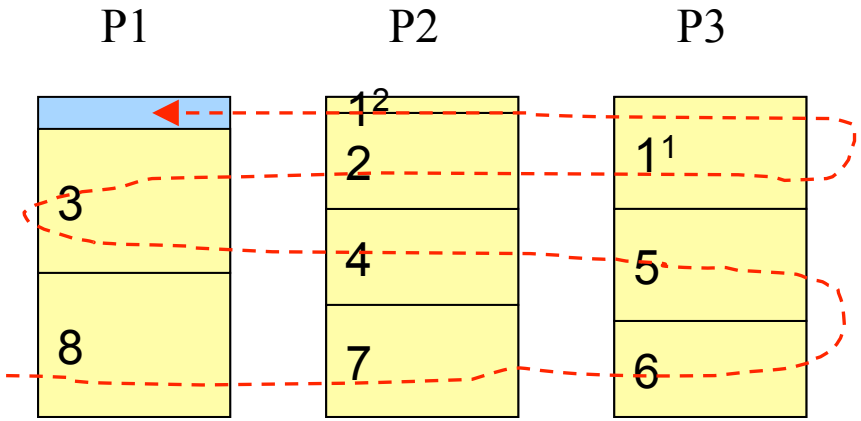


Comparison

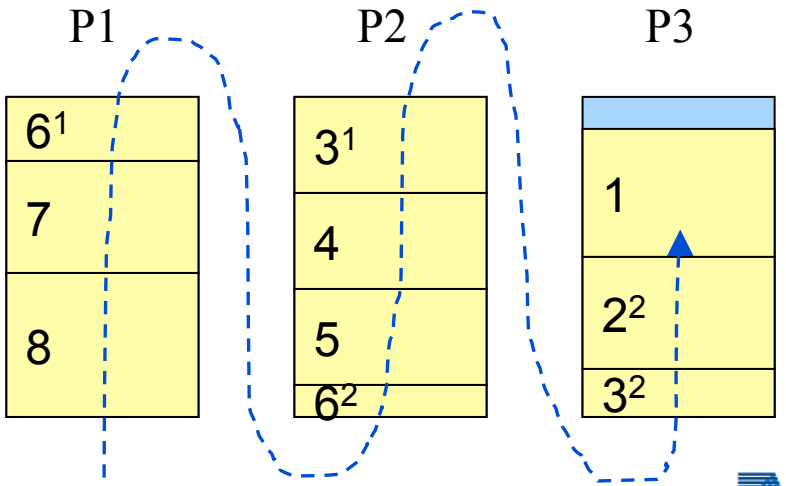
- maximal number of task splitting

  both are *M-1*

ours: width-first
(increasing priority order)



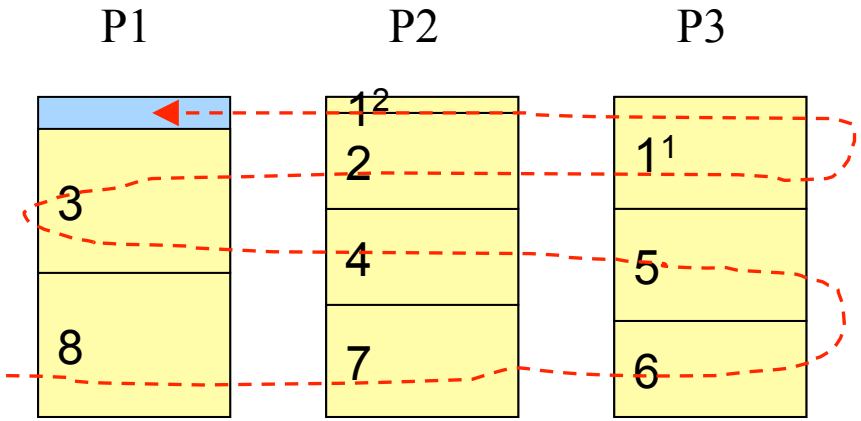
Lehoczky's: depth-first



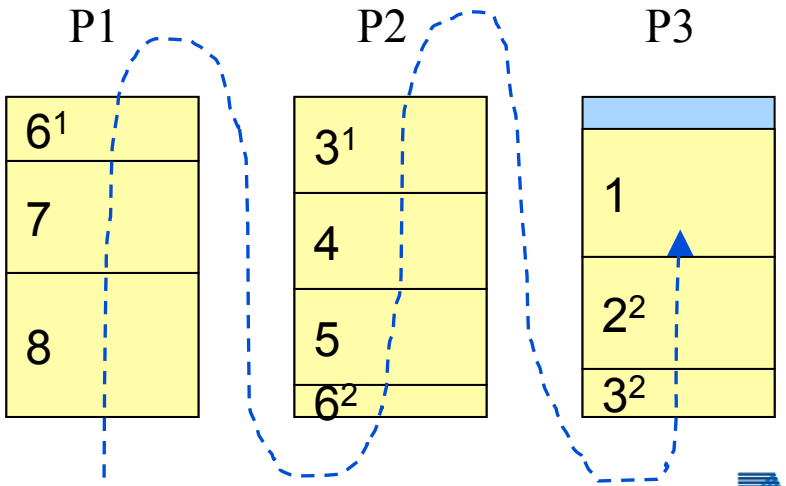
Comparison

- why is our algorithm better?

Ours: width-first
(increasing priority order)



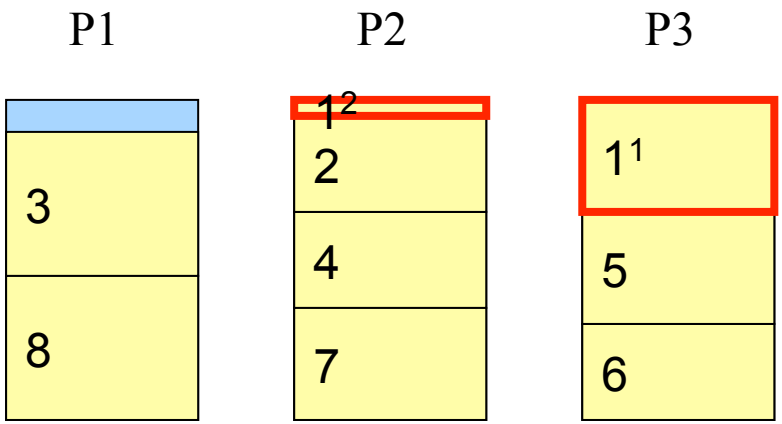
Lehoczky's: depth-first
(decreasing utilization order)



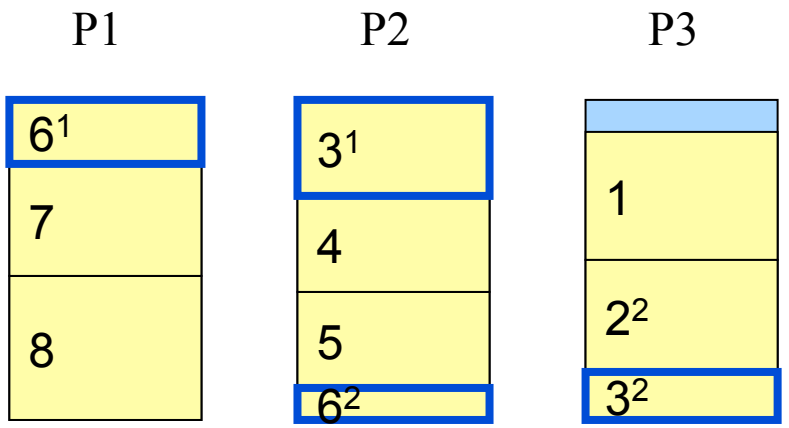
Comparison

key point:
by our algorithm, split tasks generally
have high priorities on each processor

Ours: width-first (increasing priority order)

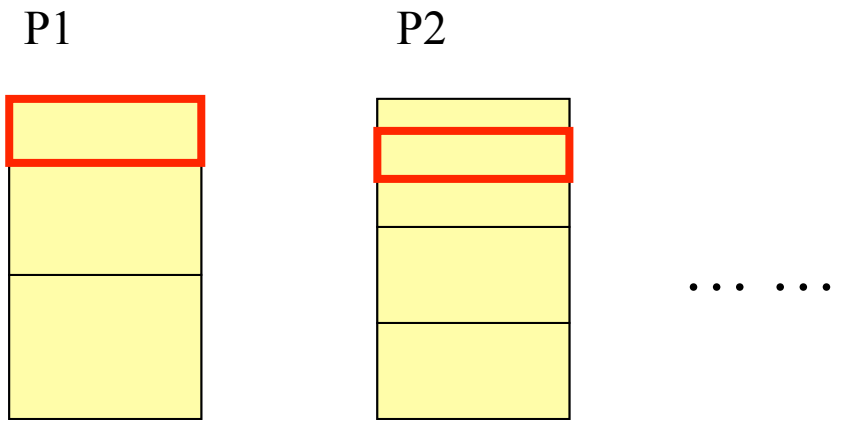


Lehoczky's: depth-first (decreasing utilization order)



Our Algorithm

- intuition
  - high priority tasks have better chance to meet its deadline
  - an extreme scenario:
    - can meet their deadlines anyway
    - no “utilization increase”



Workshops and Meetings

- **Workshop on Reconciliating Performance and Predictability**
  ESWEEK , Grenoble, France – October, 2009
- **Workshop on Software & Compilers for Embedded Systems (SCOPES) 2009**
  Nice, France – April 23-24, 2009
- **Meeting on Static WCET Analysis of multi-process and Multi-processor systems**
  Saarbrücken, Germany – 25th of September, 2009
- **Several Technical Meetings within Predator**
  e.g., *Pisa, Italy – 23th June, 2009,*
- Several short visits for collaboration

KeyNotes and Invited Talks

- **ACES 2009** (Peter Marwedel, Dortmund)
- **RTCSA 2009** (Lothar Thiele, ETHZ)
- **SCOPES 2.009** (Reinhard Wilhelm, USAAR)
- **WCET Analysis workshop (**Petru Eles, Linköping)
- **2009 Int. Conf. on Comp. Sci. and Eng.** (Wang Yi, UU)
- **Workshop on Emb. Comm., Braunschweig** (Lothar Thiele, ETHZ)
- **Ershov Memorial Conf. 2009** (Lothar Thiele, ETHZ)
- **PUMA Workshop** Reinhard Wilhelm, USAAR)
- **Anniversary of Hasso-Plattner Inst.**(Reinhard Wilhelm, USAAR)
- **Tag der Informatik, Aachen** (Reinhard Wilhelm, USAAR)

Summer Schools

- **ACACES 2009, (**Peter Puschner, Vienna) *WCET Analysis: Problems, Methods and Time-Predictable Architectures*
- **ARTIST Summer School 2009** (several speakers),
- **COMES Autumn School** Lugano, Switzerland (Lothar Thiele, ETHZ)

Tools and Platforms

- **AiT**, the leading tool for computing WCETs [AbsInt, Dortmund, Saarland]
- **WCC,** the WCET aware Compiler [AbsInt, Dortmund, Saarland]
- **MAST,** Modeling and Analysis Suite for Real-Time Applications [Cantabria]
- **MPA toolbox,** analysis of distributed embedded real-time systems, based on the real-time calculus [ETHZ]
- **MPARM,** virtual SoC platform, written in SystemC, to model system HW and SW [Bologna]
- **UPPAAL,** leading tool for precise automata-based analysis of timed systems [Uppsala, Aalborg]

Plans for Y3

Continued transversal integration

- Principles for definition of predictability multi-core architectures
- Technology for designing predictable software: language constructs, use of architectural features, operating system services, minimization of interference, handling parallel platforms.
- Continued work on Predictable HW and SW designs
- Fault-tolerance

Global event

Whitepaper