Multiprocessor Scheduling I: Partitioned Scheduling

Prof. Dr. Jian-Jia Chen

LS 12, TU Dortmund

24, June, 2014

Outline

Multiprocessor Models

- Identical (Homogeneous): All the processors have the same characteristics, i.e., the execution time of a job is independent on the processor it is executed.
- Uniform: Each processor has its own speed, i.e., the execution time of a job on a processor is proportional to the speed of the processor.
  - A faster processor always executes a job faster than slow processors do.
  - For example, multiprocessors with the same instruction set but with different supply voltages/frequencies.
- Unrelated (Heterogeneous): Each job has its own execution time on a specified processor
  - A job might be executed faster on a processor, but other jobs might be slower on that processor.
  - For example, multiprocessors with different instruction sets.

Scheduling Models

- Partitioned Scheduling:
  - Each task is assigned on a dedicated processor.
  - Schedulability is done individually on each processor.
  - It requires no additional on-line overhead.
- Global Scheduling:
  - A job may execute on any processor.
  - The system maintains a global ready queue.
  - Execute the M highest-priority jobs in the ready queue, where M is the number of processors.
  - It requires high on-line overhead.
- Semi-Partitioned Scheduling:
  - Adopt task partitioning first and reserve time slots (bandwidths) for tasks that allow migration.
  - It requires some on-line overhead.

Course Material (This Week)

- Ronald L. Graham: Bounds for certain multiprocessing anomalies. in Bell System Technical Journal (1966).
- Ronald L. Graham: Bounds on Multiprocessing Timing Anomalies. SIAM Journal of Applied Mathematics 17(2): 416-429 (1969)
- Dorit S. Hochbaum, David B. Shmoys: Using dual approximation algorithms for scheduling problems theoretical and practical results. J. ACM 34(1): 144-162 (1987) (in textbook Approximation Algorithms by Vijay Vazirani, Chapter 10, not covered)
- Sanjoy K. Baruah, Nathan Fisher: The Partitioned Multiprocessor Scheduling of Sporadic Task Systems. RTSS 2005: 321-329

Recap Some Terms

Graham's Scheduling Algorithm Classification

- Classification: $a|b|c$
  - a: machine environment
    (e.g., uniprocessor, multiprocessor, distributed, ...)
  - b: task and resource characteristics
    (e.g., preemptive, independent, synchronous, ...)
  - c: performance metric and objectives
    (e.g., $L_{\max}$, sum of finish times, ...)
- Makespan problem:
  - $M||C_{\max}$
  - The course material mainly comes from the traditional makespan scheduling in the context of *minimizing the maximum utilization*. (The paper by Baruah and Fisher in RTSS 2005 is an exception.)
  - Imagine that the goal is to minimize the maximum utilization after task partitioning.
  - We will only talk about EDF scheduling for partitioned scheduling.

Bin Packing Problem

- Given a bin size b, and a set of items with individual sizes, the objective is to assign each item to a bin without violating the bin size constraint such that the number of allocated bins is minimized.

Outline

Problem Definition

Partitioned Scheduling

Given a set $\mathbf{T}$ of tasks with implicit deadlines, i.e., $\forall \tau_i \in \mathbf{T}$, $T_i = D_i$, the objective is to decide a feasible task assignment onto M processors such that all the tasks meet their timing constraints, where C_{im} is the execution time of task τ_i on processor m.

- For identical multiprocessors: $C_i = C_{i1} = C_{i2} = \cdots = C_{iM}$.
- For uniform multiprocessors: each processor m has a speed s_m, in which $C_{im}s_m$ is a constant.
- For unrelated multiprocessors: C_{im} is an independent variable.

Hardness and Approximation of Partitioned Scheduling

$\mathcal{NP}$-complete

Deciding whether there exists a feasible task assignment is $\mathcal{NP}$-complete in the strong sense.

Proof

Reduced from the makespan or the bin packing problem.

Hardness and Approximation of Partitioned Scheduling

$\mathcal{NP}$-complete

Deciding whether there exists a feasible task assignment is $\mathcal{NP}$-complete in the strong sense.

Proof

Reduced from the makespan or the bin packing problem.

- Approximations are possible, but what do we approximate when only binary decisions (Yes or No) have to be made?
  - Deadline relaxation: requires modifications of task specification
  - Period relaxation: requires modifications of task specification
  - Resource augmentation by speeding up: requires a faster platform
  - Resource augmentation by allocating more processors: requires a better platform

Approximation Algorithms

An algorithm $\mathcal{A}$ is called an η-approximation algorithm (for a minimization problem) if it guarantees to derive a feasible solution for any input instance I with at most η times of the objective function of an optimal solution. That is,

$$\mathcal{A}(I) \leq \eta OPT(I),$$

where $OPT(I)$ is the objective function of an optimal solution.

Largest-Utilization-First (LUF)

Input: $\mathbf{T}, M$;

1: re-index (sort) tasks such that $\frac{C_i}{T_i} \geq \frac{C_j}{T_j}$ for $i < j$;
2: $\mathbf{T}_i \leftarrow \emptyset, U_i \leftarrow 0, \forall m = 1, 2, \ldots, M$;
3: **for** $i = 1$ to N, where $N = |\mathbf{T}|$ **do**
4: find m^* with the minimum utilization, i.e., $U_{m^*} = \min_{m \leq M} U_m$;
5: **if** $U_{m^*} + \frac{C_i}{T_i} > 1$ **then**
6: return "The task assignment fails";
7: **else**
8: assign task τ_i onto processor m^*, where $U_{m^*} \leftarrow U_{m^*} + \frac{C_i}{T_i}, \mathbf{T}_i \leftarrow \mathbf{T}_i \cup \{\tau_i\}$;
9: return feasible task assignment $\mathbf{T}_1, \mathbf{T}_2, \ldots, \mathbf{T}_M$;

Largest-Utilization-First (LUF)

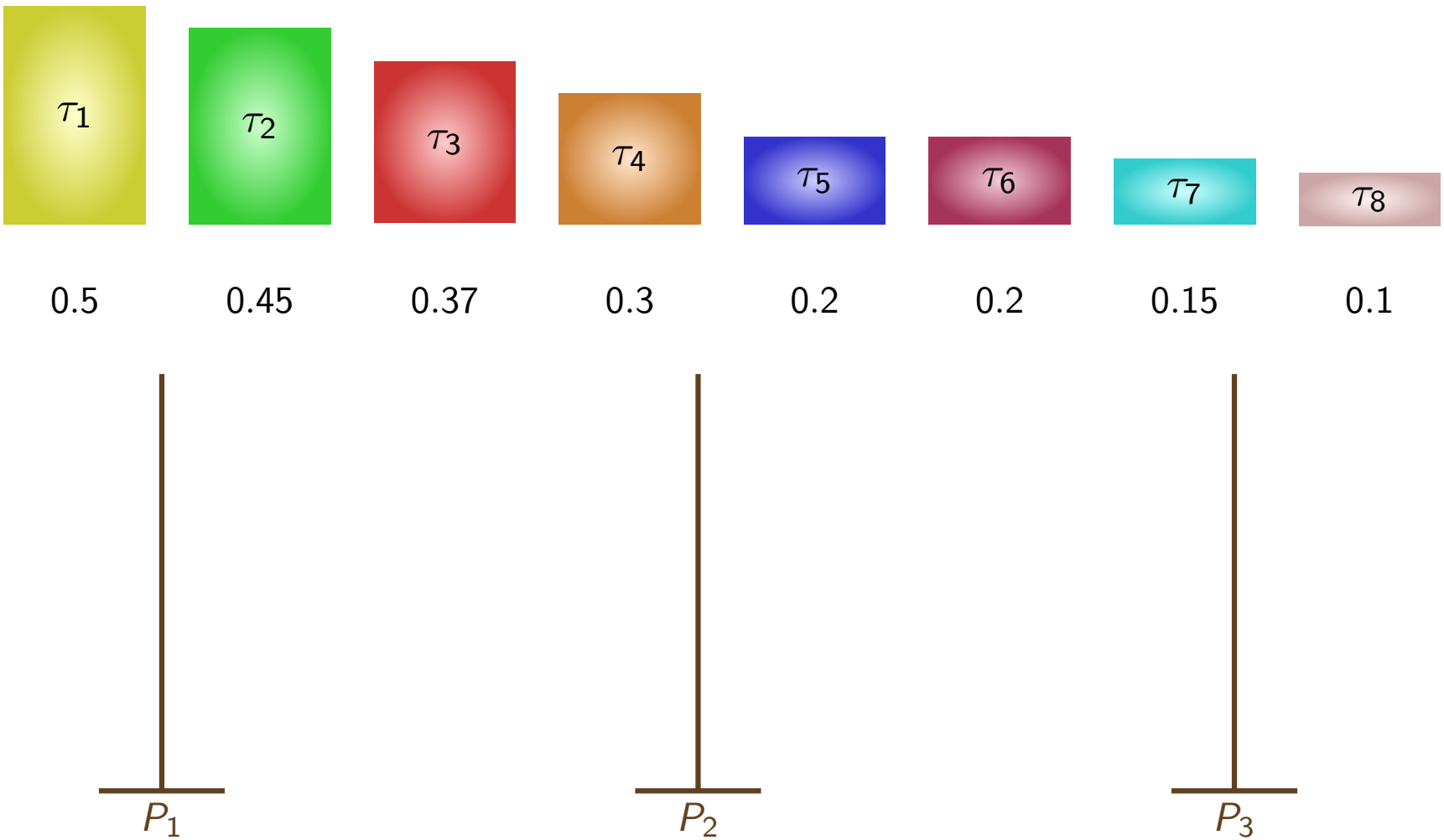
Input: $\mathbf{T}, M$;

1: re-index (sort) tasks such that $\frac{C_i}{T_i} \geq \frac{C_j}{T_j}$ for $i < j$;
2: $\mathbf{T}_i \leftarrow \emptyset, U_i \leftarrow 0, \forall m = 1, 2, \ldots, M$;
3: **for** $i = 1$ to N, where $N = |\mathbf{T}|$ **do**
4: find m^* with the minimum utilization, i.e., $U_{m^*} = \min_{m \leq M} U_m$;
5: **if** $U_{m^*} + \frac{C_i}{T_i} > 1$ **then**
6: return "The task assignment fails";
7: **else**
8: assign task τ_i onto processor m^*, where $U_{m^*} \leftarrow U_{m^*} + \frac{C_i}{T_i}, \mathbf{T}_i \leftarrow \mathbf{T}_i \cup \{\tau_i\}$;
9: return feasible task assignment $\mathbf{T}_1, \mathbf{T}_2, \ldots, \mathbf{T}_M$;

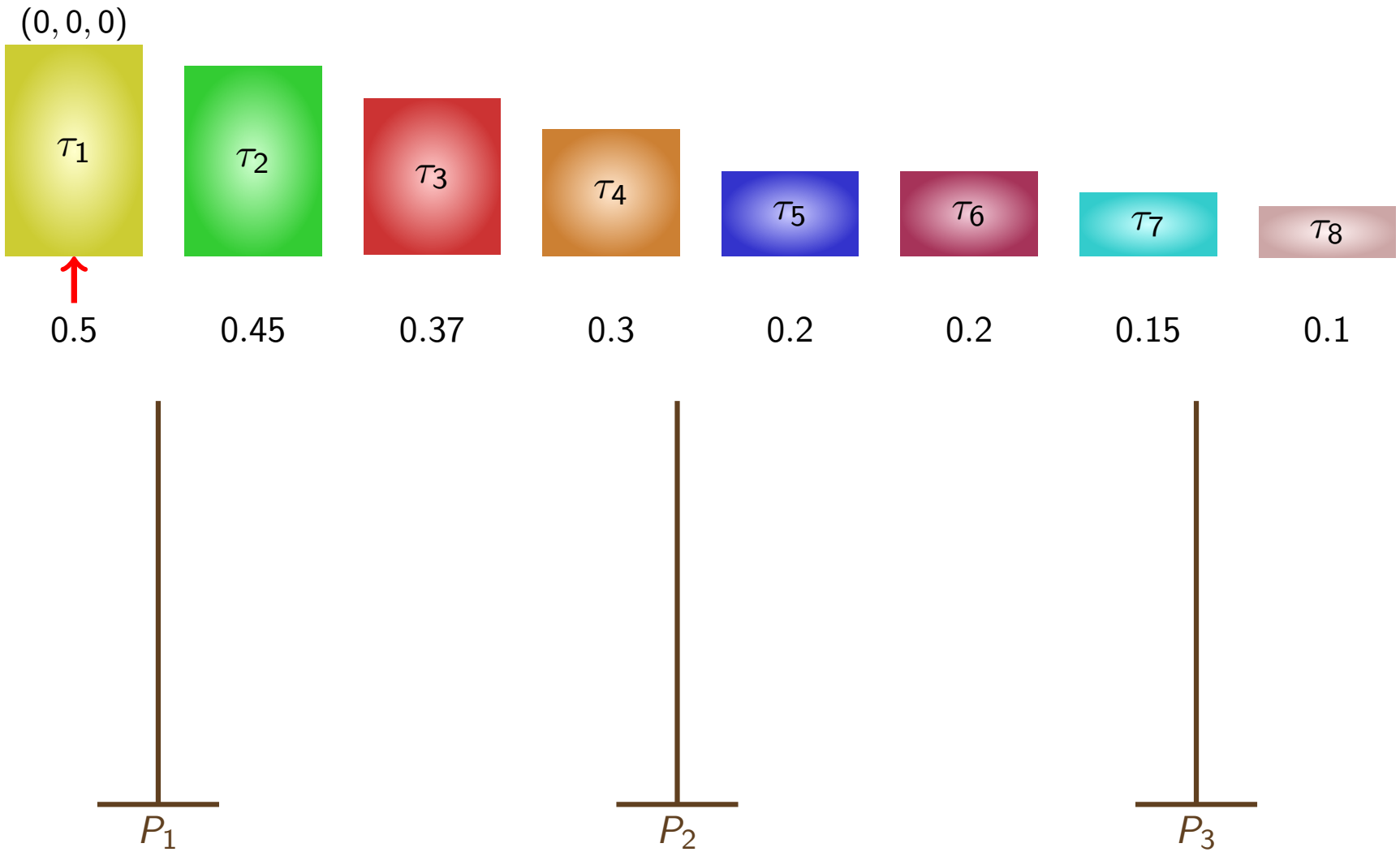
Properties

- The time complexity is $O((N + M) \log(N + M))$
- If the task assignment is derived, the resulting solution is feasible.

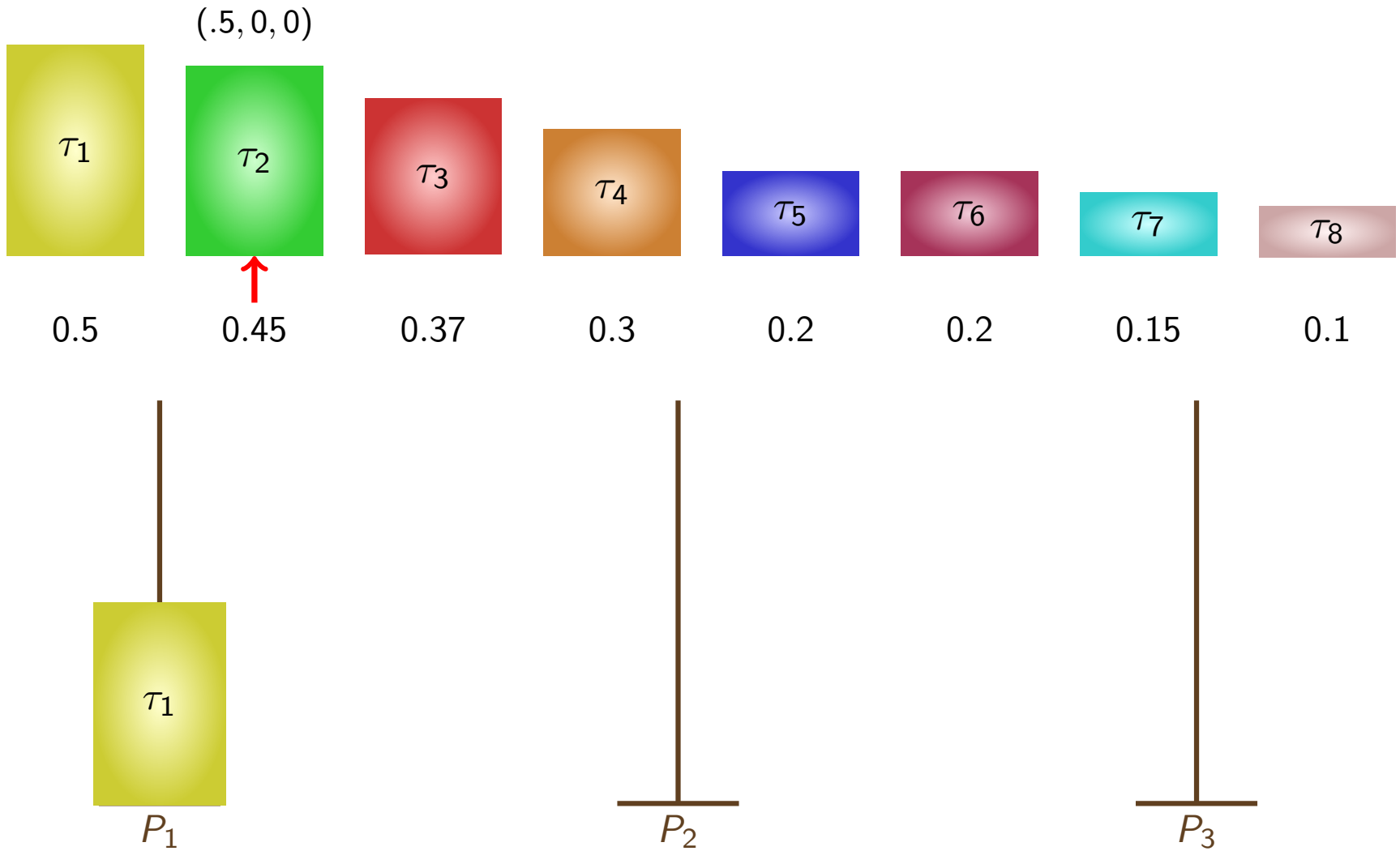
Algorithm LUF



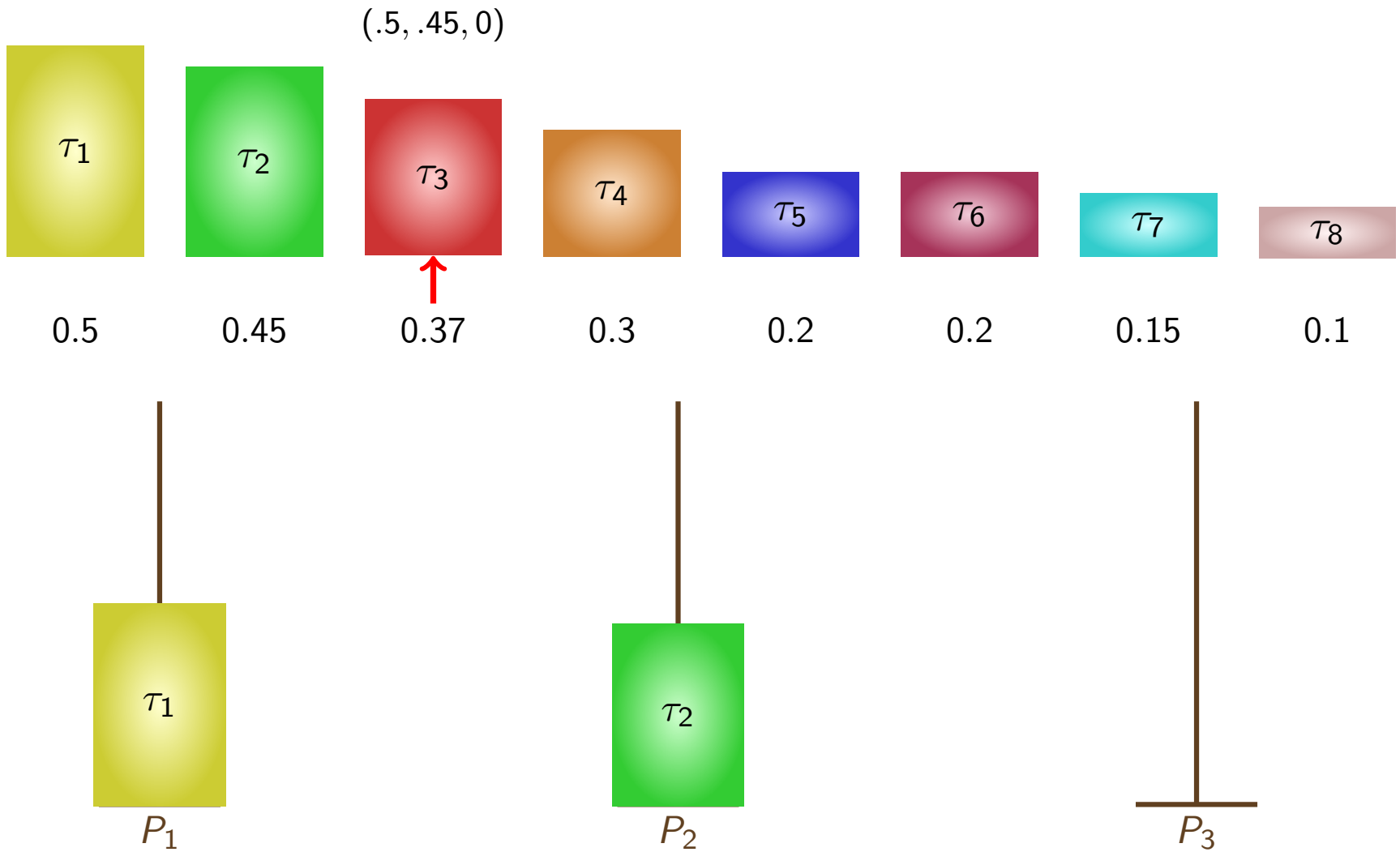
Algorithm LUF



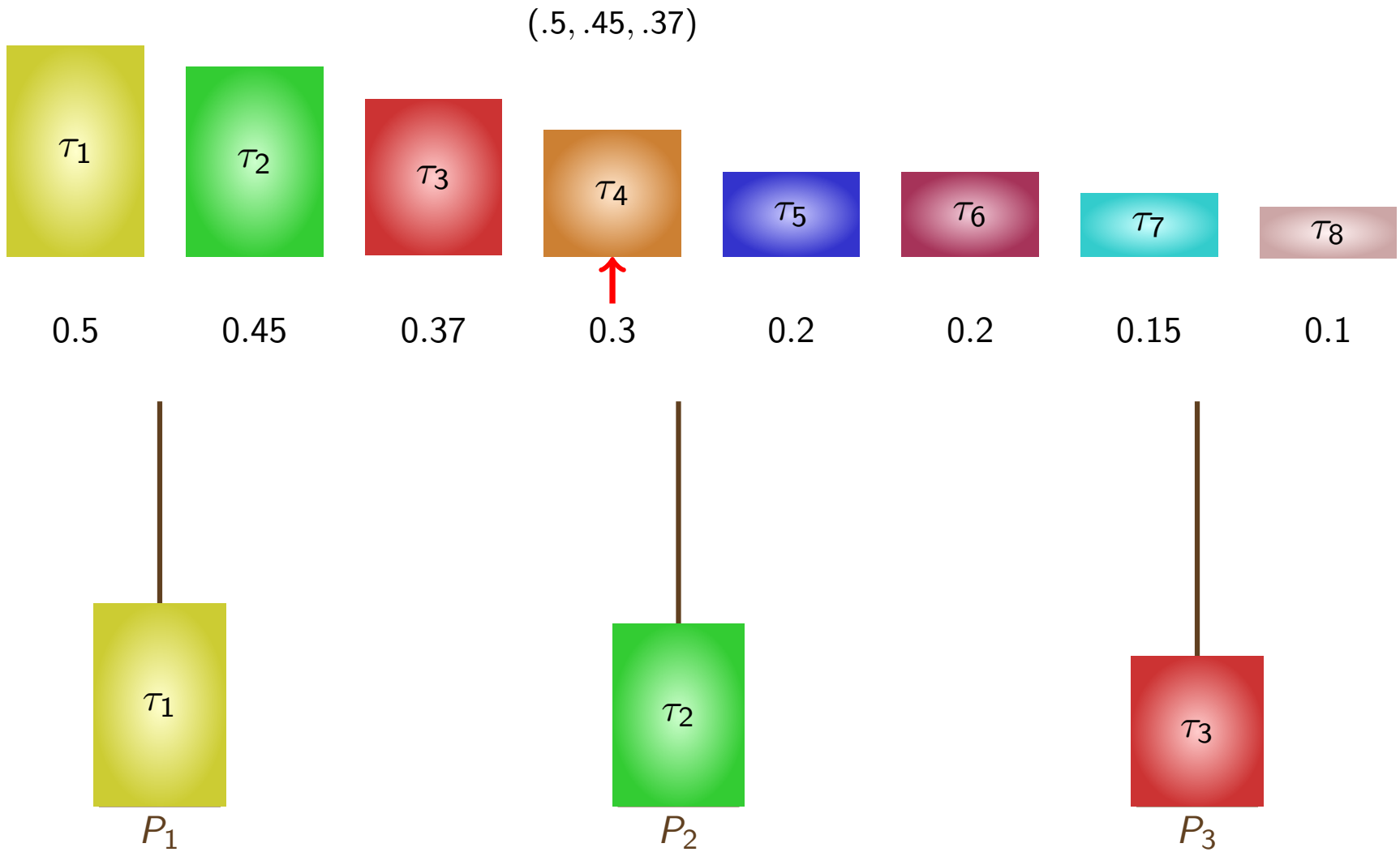
Algorithm LUF



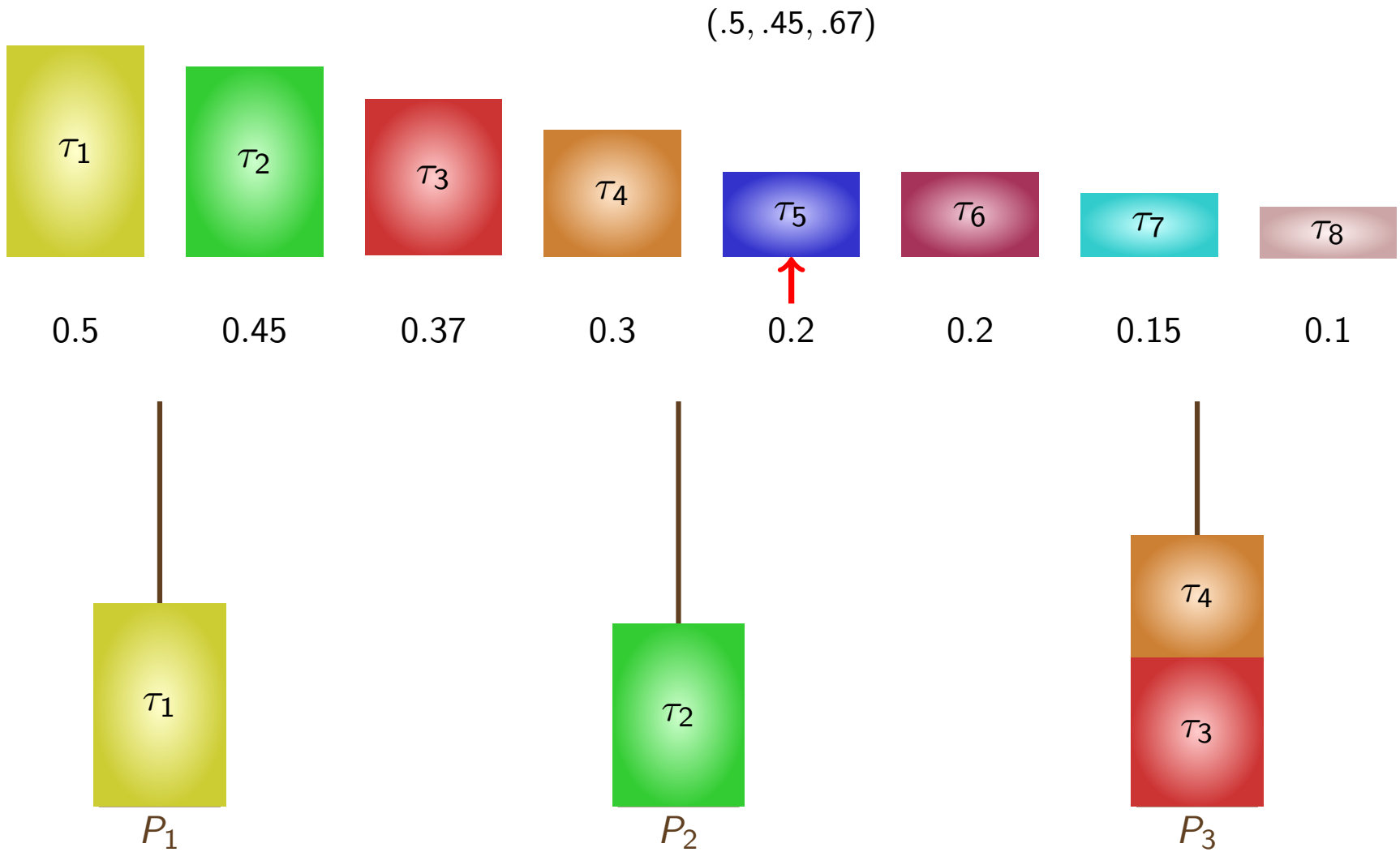
Algorithm LUF



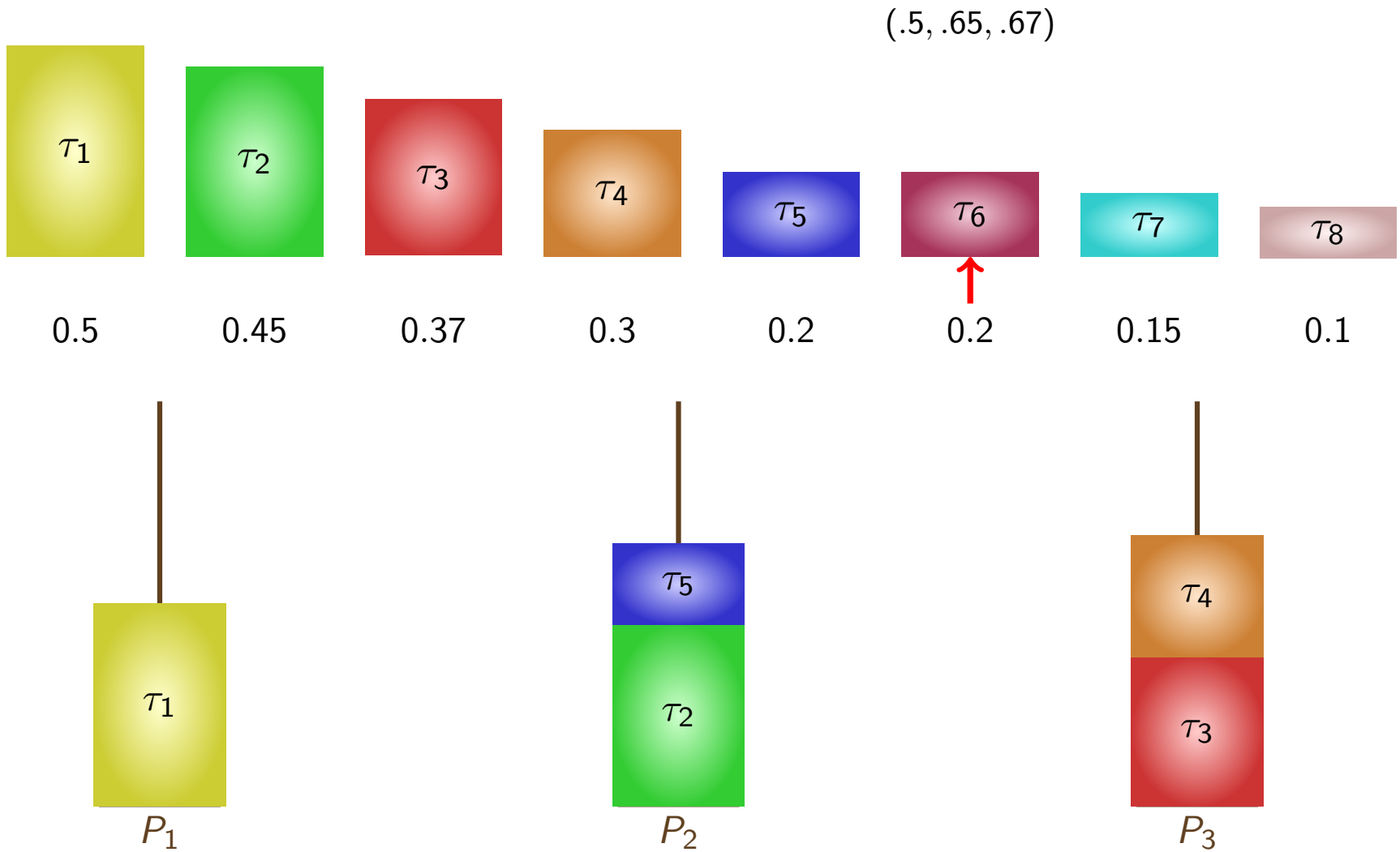
Algorithm LUF



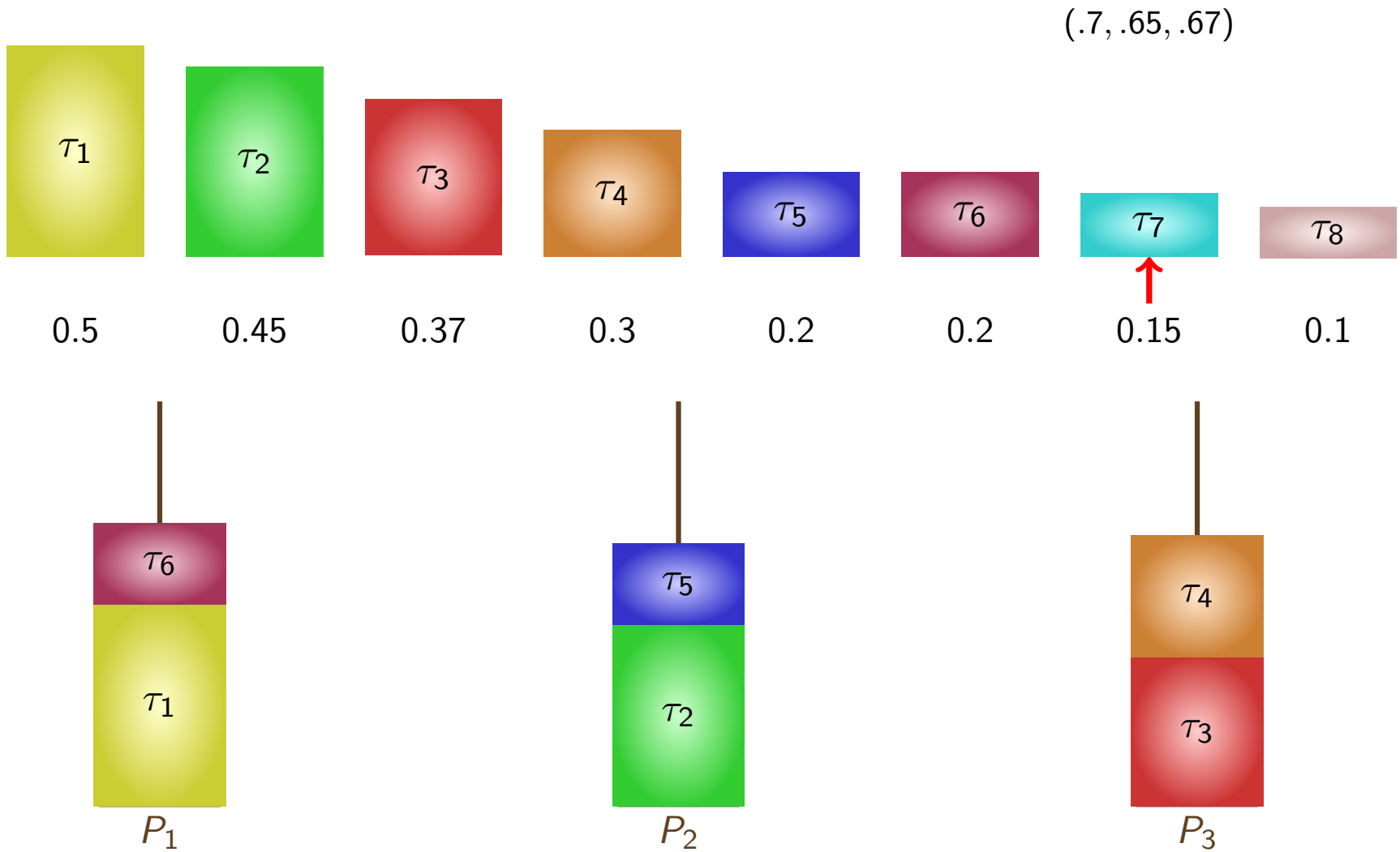
Algorithm LUF



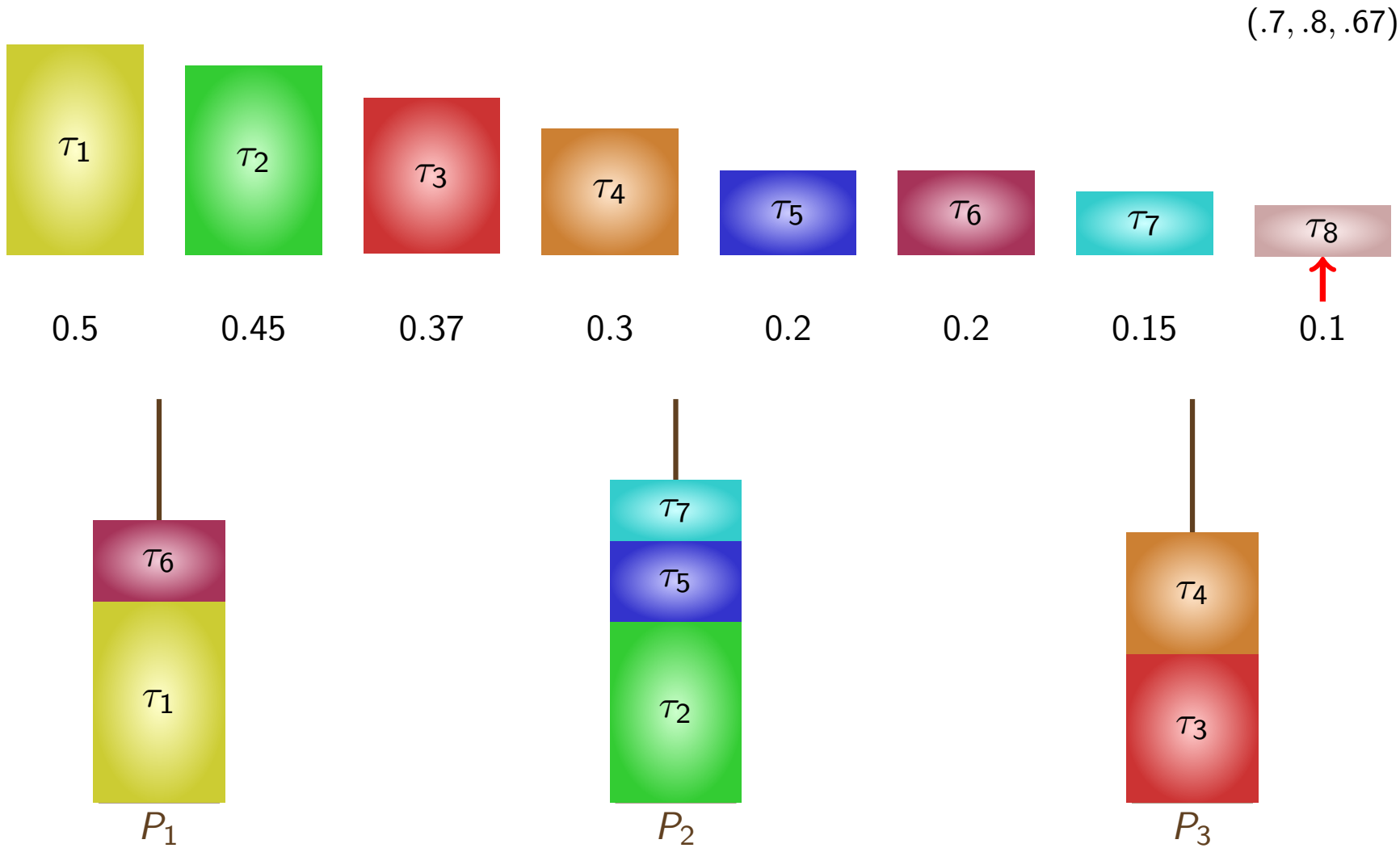
Algorithm LUF



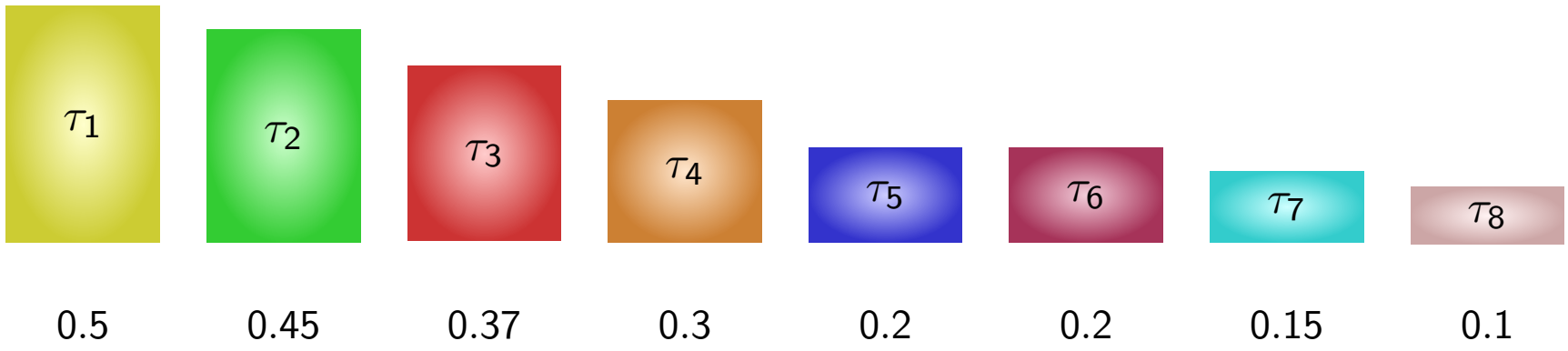
Algorithm LUF

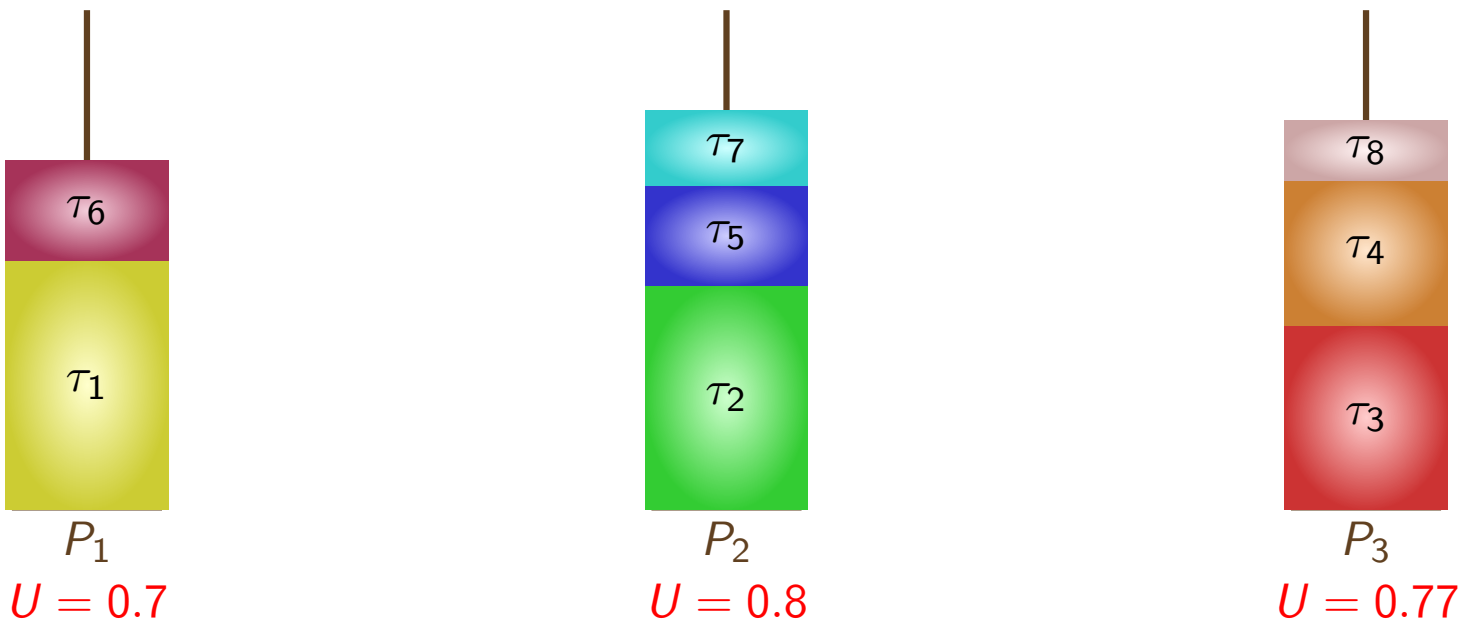


Algorithm LUF



Algorithm LUF





Optimality of Algorithm LUF

Theorem

If an optimal assignment for minimizing the maximal utilization results in at most two tasks on any processor, LUF is optimal.

Proof

Please take the proof as an exercise.

What Happens if Algorithm LUF Fails?

Assume that there exists a feasible task partition on M processors (for providing the analysis of resource augmentation).

- Suppose that Algorithm LUF fails when assigning task τ_j and U_m for $m = 1, 2, \ldots, M$ is the utilization of processor m before assigning τ_j.
- Let U_{opt} be the utilization of the optimal assignment for minimizing the maximal utilization for tasks $\{\tau_1, \tau_2, \ldots, \tau_j\}$.
- By definition, $1 \geq U_{opt} \geq \sum_{i=1}^{j} \frac{C_i/T_i}{M}$.
- $\frac{C_j}{T_j} \leq \frac{1}{3} U_{opt}$: otherwise, there will be at most two tasks on any processors in the optimal solution. $\Rightarrow$ this contradicts the assumption that Algorithm LUF fails as it is optimal.
- Since $U_{m^*} \leq U_m$, we know that $U_{m^*} \leq \sum_{m=1}^{M} \frac{U_m}{M} = \sum_{i=1}^{j-1} \frac{C_i/T_i}{M}$.
- Therefore,

$$\frac{C_j}{T_j} + U_{m^*} \leq \frac{C_j}{T_j}(1 - \frac{1}{M}) + \sum_{i=1}^{j} \frac{C_i/T_i}{M} \leq \left(\frac{4}{3} - \frac{1}{3M}\right) U_{opt} \leq \left(\frac{4}{3} - \frac{1}{3M}\right).$$

Algorithm LUF^*: Resource Augmentation

Input: $\mathbf{T}, M$;

1: re-index (sort) tasks such that $\frac{C_i}{T_i} \geq \frac{C_j}{T_j}$ for $i < j$;
2: $\mathbf{T}_i \leftarrow \emptyset, U_i \leftarrow 0, \forall m = 1, 2, \ldots, M$;
3: **for** $i = 1$ to N, where $N = |\mathbf{T}|$ **do**
4: find the processor m^* with the minimum task utilization, i.e., $U_{m^*} = \min_m U_m$;
5: assign task τ_i onto processor m^*, where $U_i \leftarrow U_i + \frac{C_i}{T_i}, \mathbf{T}_i \leftarrow \mathbf{T}_i \cup \{\tau_i\}$;
6: return task assignment $\mathbf{T}_1, \mathbf{T}_2, \ldots, \mathbf{T}_M$ with a speedup factor $\max\{1, U_m\}$;

Properties of LUF^*

Theorem

If the input task set can be feasibly scheduled, Algorithm LUF^* derives a feasible task assignment/schedule by running the processors at a speedup factor by at most $\frac{4}{3} - \frac{1}{3M}$.

Theorem

If the input task set cannot be feasibly scheduled by Algorithm LUF, the task set is not schedulable by running at a slow-down factor $\frac{1}{\frac{4}{3} - \frac{1}{3M}}$.

Algorithm LUF^+: Resource Augmentation on Processors

Input: $\mathbf{T}$;

1: re-index (sort) tasks such that $\frac{C_i}{T_i} \geq \frac{C_j}{T_j}$ for $i < j$;
2: $\mathbf{T}_1 \leftarrow \emptyset, U_1 \leftarrow 0, \hat{M} \leftarrow 1$;
3: **for** $i = 1$ to N, where $N = |\mathbf{T}|$ **do**
4: find a processor m^* with $U_{m^*} + \frac{C_i}{T_i} \leq 1$;
5: **if** no such a processor exists **then**
6: $\hat{M} \leftarrow \hat{M} + 1, \mathbf{T}_{\hat{M}} \leftarrow \emptyset, U_{\hat{M}} \leftarrow 0$;
7: $m^* \leftarrow \hat{M}$;
8: assign task τ_i onto processor m^*, where $U_i \leftarrow U_i + \frac{C_i}{T_i}, \mathbf{T}_i \leftarrow \mathbf{T}_i \cup \{\tau_i\}$;
9: return task assignment $\mathbf{T}_1, \mathbf{T}_2, \ldots, \mathbf{T}_{\hat{M}}$;

Algorithm LUF^+: Resource Augmentation on Processors

Input: $\mathbf{T}$;

1: re-index (sort) tasks such that $\frac{C_i}{T_i} \geq \frac{C_j}{T_j}$ for $i < j$;
2: $\mathbf{T}_1 \leftarrow \emptyset, U_1 \leftarrow 0, \hat{M} \leftarrow 1$;
3: **for** $i = 1$ to N, where $N = |\mathbf{T}|$ **do**
4: find a processor m^* with $U_{m^*} + \frac{C_i}{T_i} \leq 1$;
5: **if** no such a processor exists **then**
6: $\hat{M} \leftarrow \hat{M} + 1, \mathbf{T}_{\hat{M}} \leftarrow \emptyset, U_{\hat{M}} \leftarrow 0$;
7: $m^* \leftarrow \hat{M}$;
8: assign task τ_i onto processor m^*, where $U_i \leftarrow U_i + \frac{C_i}{T_i}, \mathbf{T}_i \leftarrow \mathbf{T}_i \cup \{\tau_i\}$;
9: return task assignment $\mathbf{T}_1, \mathbf{T}_2, \ldots, \mathbf{T}_{\hat{M}}$;

Properties

- The time complexity is $O(N \log N)$ or $O(N^2)$, depending on the fitting approaches.
- The resulting solution is feasible on $\hat{M}$ processors.

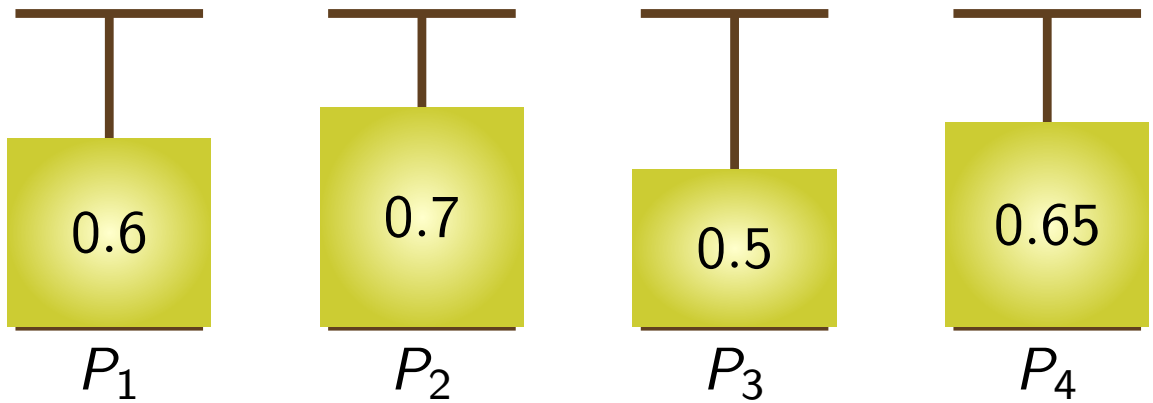
Different Fitting Approaches

4: find a processor m^* with $U_{m^*} + \frac{C_i}{T_i} \leq 1$;

Fitting Strategies

- First-Fit: choose the feasible one with the smallest index
- Last-Fit: choose the feasible one with the largest index
- Best-Fit: choose the feasible one with the maximal utilization
- Worst-Fit: choose the feasible one with the minimal utilization

Suppose that we want to assign a task with utilization equal to 0.1.



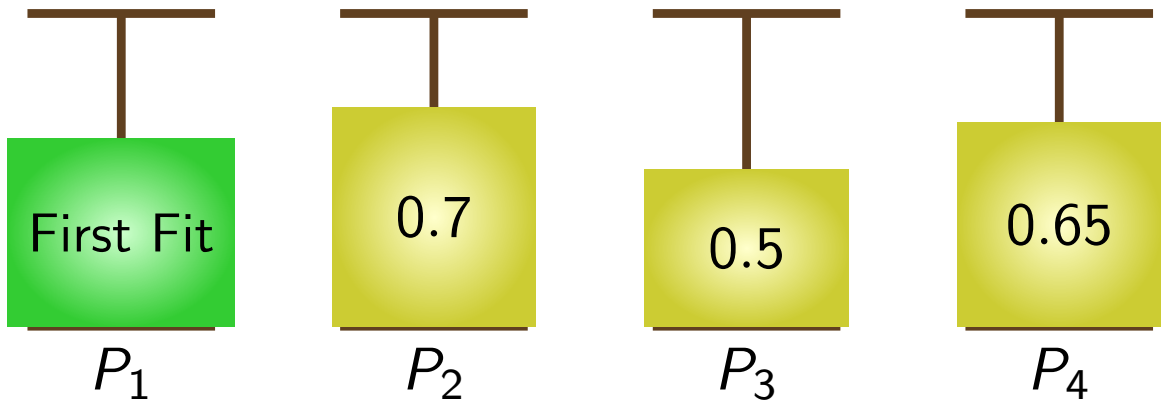
Different Fitting Approaches

4: find a processor m^* with $U_{m^*} + \frac{C_i}{T_i} \leq 1$;

Fitting Strategies

- First-Fit: choose the feasible one with the smallest index
- Last-Fit: choose the feasible one with the largest index
- Best-Fit: choose the feasible one with the maximal utilization
- Worst-Fit: choose the feasible one with the minimal utilization

Suppose that we want to assign a task with utilization equal to 0.1.



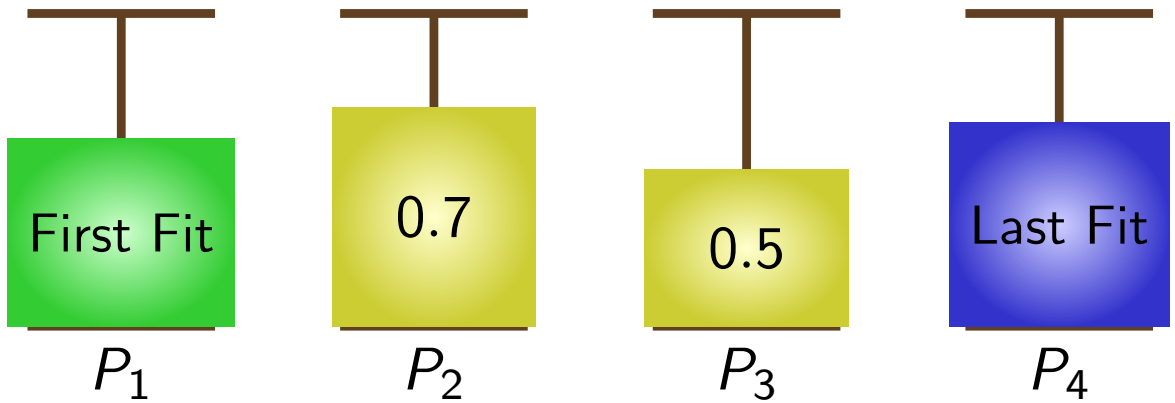
Different Fitting Approaches

4: find a processor m^* with $U_{m^*} + \frac{C_i}{T_i} \leq 1$;

Fitting Strategies

- First-Fit: choose the feasible one with the smallest index
- Last-Fit: choose the feasible one with the largest index
- Best-Fit: choose the feasible one with the maximal utilization
- Worst-Fit: choose the feasible one with the minimal utilization

Suppose that we want to assign a task with utilization equal to 0.1.



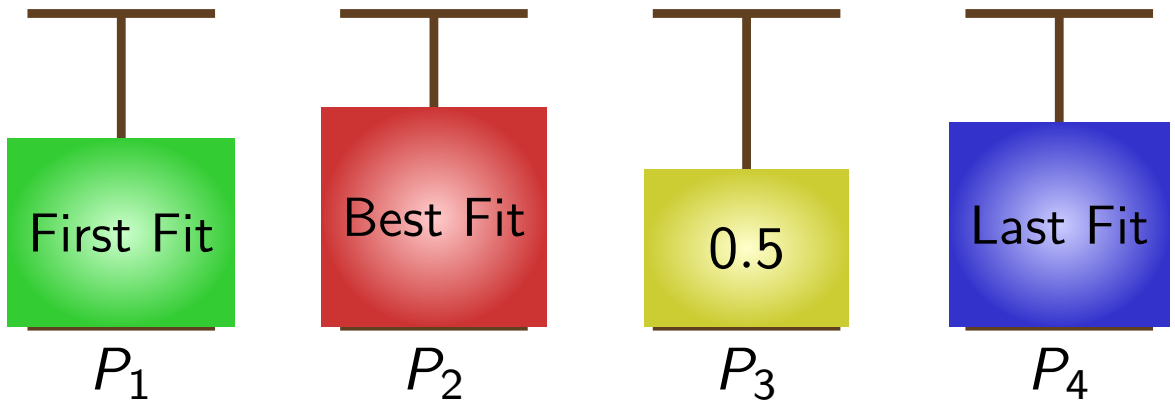
Different Fitting Approaches

4: find a processor m^* with $U_{m^*} + \frac{C_i}{T_i} \leq 1$;

Fitting Strategies

- First-Fit: choose the feasible one with the smallest index
- Last-Fit: choose the feasible one with the largest index
- Best-Fit: choose the feasible one with the maximal utilization
- Worst-Fit: choose the feasible one with the minimal utilization

Suppose that we want to assign a task with utilization equal to 0.1.



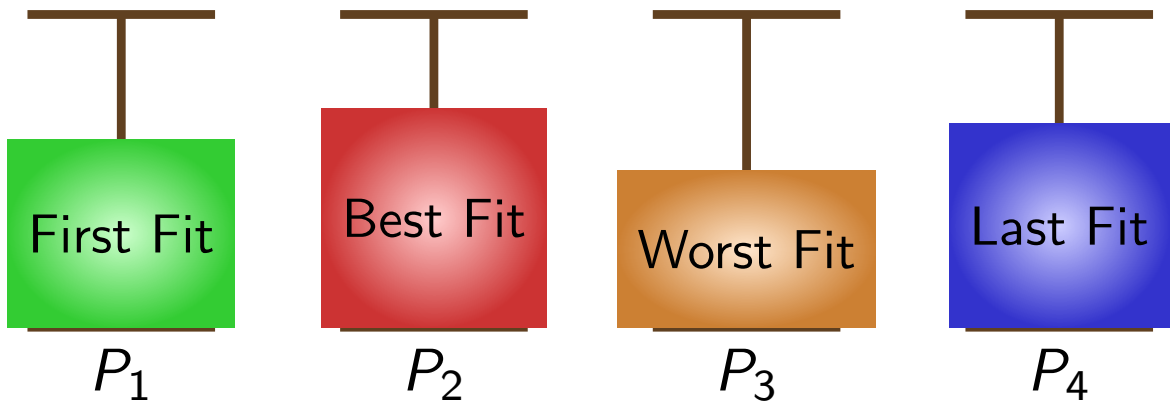
Different Fitting Approaches

4: find a processor m^* with $U_{m^*} + \frac{C_i}{T_i} \leq 1$;

Fitting Strategies

- First-Fit: choose the feasible one with the smallest index
- Last-Fit: choose the feasible one with the largest index
- Best-Fit: choose the feasible one with the maximal utilization
- Worst-Fit: choose the feasible one with the minimal utilization

Suppose that we want to assign a task with utilization equal to 0.1.



Algorithm LUF^+: How Many Processors?

- Suppose that the processor used by Algorithm LUF^+ is $\hat{M} \geq 2$.
- Let m^* be the processor with the minimum utilization.
- By the fitting algorithm, we know that $U_m + U_{m^*} > 1$ and $U_m \geq U_{m^*}$ for all the other processors ms.
- If $U_{m^*} \leq 0.5$, by $U_m > 1 - U_{m^*}$, we know that

$$\sum_{\tau_i \in \mathbf{T}} \frac{C_i}{T_i} \geq U_{m^*} + \sum_{m=1, m \neq m^*}^{\hat{M}} U_m \geq \hat{M} - 1 - (\hat{M} - 2) U_{m^*} = (\hat{M} - 2)(1 - U_{m^*}) + 1 \geq \frac{\hat{M}}{2}.$$

- If $U_{m^*} > 0.5$, by $U_m \geq U_{m^*}$, we know that

$$\sum_{\tau_i \in \mathbf{T}} \frac{C_i}{T_i} \geq U_{m^*} + \sum_{m=1, m \neq m^*}^{\hat{M}} U_m \geq \frac{\hat{M}}{2}.$$

Theorem

Algorithm LUF^+ is a 2-approximation algorithm.

Algorithm *LUF* Revisit: Upper and Lower Bounds

- Let m^* be the processor with the maximum utilization among M processors.
- Let τ_k be the last task that is added to m^*.
- Therefore, we know that $U_m \geq U_{m^*} - \frac{C_k}{T_k}$ for any other processor m.
- The lower bound of the maximum utilization to map $\mathbf{T}$ on M processor is

$$LB \geq \max\left\{\max_{\tau_i \in \mathbf{T}} \frac{C_i}{T_i}, \frac{\sum_{\tau_i \in \mathbf{T}} \frac{C_i}{T_i}}{M}\right\}$$

$$\Rightarrow U_{m^*} \leq U_m + \frac{C_k}{T_k} \leq \frac{\sum_{m=1}^{M} U_m}{M} + \frac{C_k}{T_k} = \frac{\sum_{\tau_i \in \mathbf{T}} \frac{C_i}{T_i}}{M} + \frac{C_k}{T_k} \leq 2LB.$$

- Therefore, we reach that

$$\max\left\{\max_{\tau_i \in \mathbf{T}} \frac{C_i}{T_i}, \frac{\sum_{\tau_i \in \mathbf{T}} \frac{C_i}{T_i}}{M}\right\} = LB \leq U_{m^*} \leq 2LB.$$

Outline

Definition and Theorem Recalled

For a task set, we say that the task set is with

- *constrained deadline* when the relative deadline D_i is no more than the period T_i, i.e., $D_i \leq T_i$, for every task τ_i, or
- *arbitrary deadline* when the relative deadline D_i could be larger than the period T_i for some task τ_i.

Theorem

A task set $\mathbf{T}$ of independent, preemptable, periodic tasks can be feasibly scheduled (under EDF) on one processor if and only if

$$\forall t \geq 0, \sum_{i=1}^{n} \max\left\{0, \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor\right\} C_i \leq t.$$

Problem Definition

Partitioned Scheduling

Given a set $\mathbf{T}$ of tasks with arbitrary deadlines, the objective is to decide a feasible task assignment onto M processors such that all the tasks meet their timing constraints, where C_i is the execution time of task τ_i on any processor m.

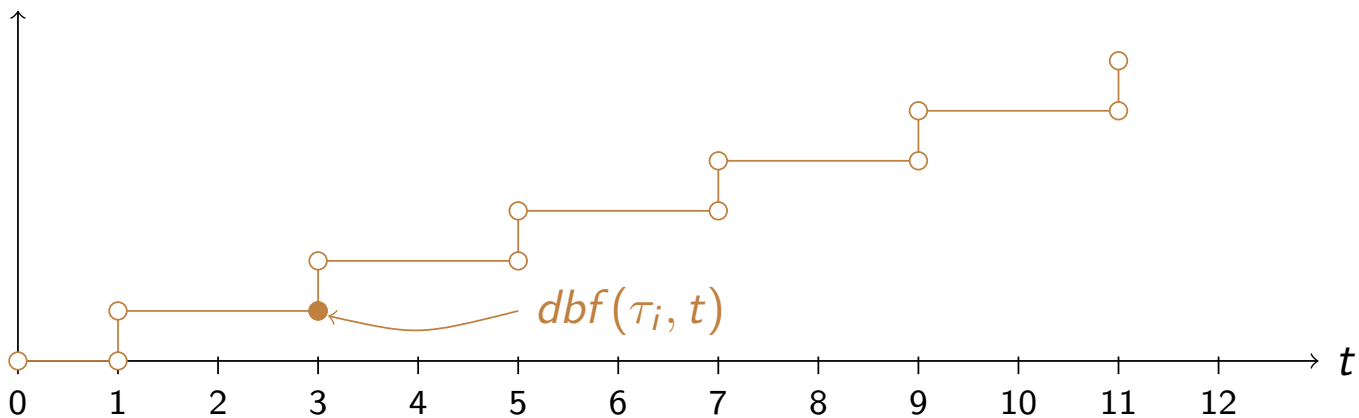
Demand Bound Function Revisit for EDF

Define demand bound function $dbf(\tau_i, t)$ as

$$dbf(\tau_i, t) = \max\left\{0, \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor\right\} C_i = \max\left\{0, \left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1\right\} C_i.$$

We need approximation to enforce *polynomial-time* schedulability test.

$$dbf^*(\tau_i, t) = \begin{cases} 0 & \text{if } t < D_i \\ (\frac{t - D_i}{T_i} + 1) C_i & \text{otherwise.} \end{cases}$$



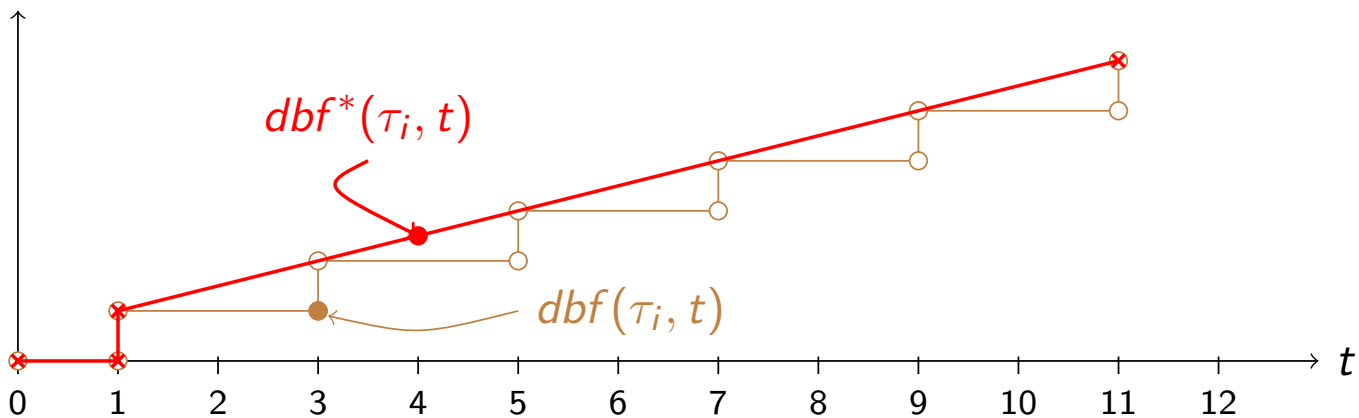
Demand Bound Function Revisit for EDF

Define demand bound function $dbf(\tau_i, t)$ as

$$dbf(\tau_i, t) = \max\left\{0, \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor\right\} C_i = \max\left\{0, \left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1\right\} C_i.$$

We need approximation to enforce *polynomial-time* schedulability test.

$$dbf^*(\tau_i, t) = \begin{cases} 0 & \text{if } t < D_i \\ (\frac{t - D_i}{T_i} + 1) C_i & \text{otherwise.} \end{cases}$$



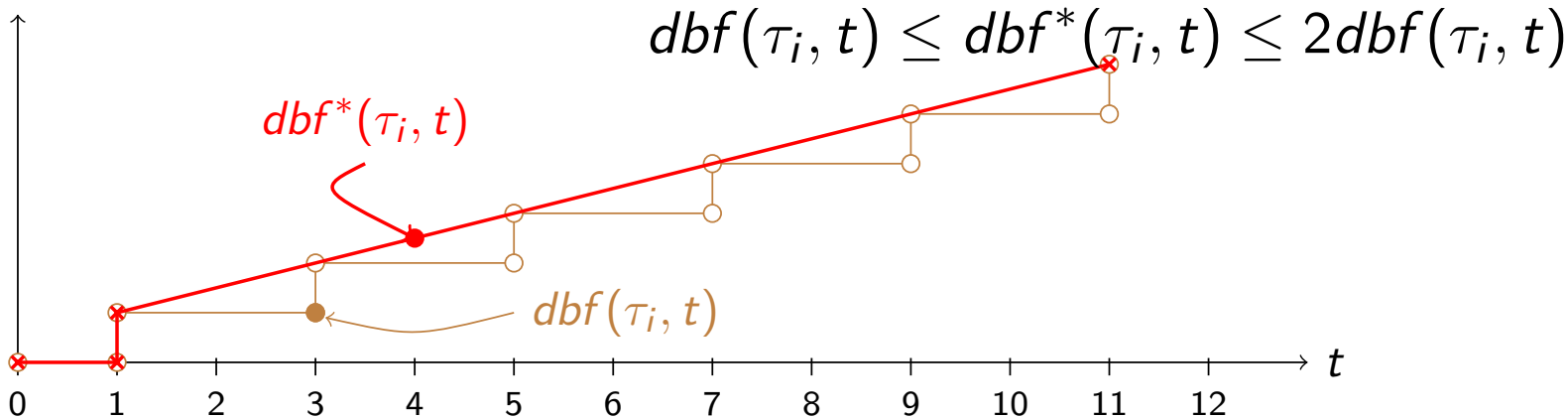
Demand Bound Function Revisit for EDF

Define demand bound function $dbf(\tau_i, t)$ as

$$dbf(\tau_i, t) = \max\left\{0, \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor\right\} C_i = \max\left\{0, \left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1\right\} C_i.$$

We need approximation to enforce *polynomial-time* schedulability test.

$$dbf^*(\tau_i, t) = \begin{cases} 0 & \text{if } t < D_i \\ (\frac{t - D_i}{T_i} + 1) C_i & \text{otherwise.} \end{cases}$$



Deadline-Monotonic Partition (Shortest Relative-Deadline First)

Input: $\mathbf{T}, M$;

1: re-index (sort) tasks such that $D_i \leq D_j$ for $i < j$;
2: $\mathbf{T}_i \leftarrow \emptyset, U_i \leftarrow 0, \forall m = 1, 2, \ldots, M$;
3: **for** $i = 1$ to N, where $N = |\mathbf{T}|$ **do**

10: return feasible task assignment $\mathbf{T}_1, \mathbf{T}_2, \ldots, \mathbf{T}_M$;

Deadline-Monotonic Partition (Shortest Relative-Deadline First)

Input: $\mathbf{T}, M$;

re-index (sort) tasks such that $D_i \leq D_j$ for $i < j$;
$\mathbf{T}_i \leftarrow \emptyset, U_i \leftarrow 0, \forall m = 1, 2, \ldots, M$;
for $i = 1$ to N, where $N = |\mathbf{T}|$ **do**
  for $m = 1$ to M **do**
    if $\frac{C_i}{T_i} + \sum_{\tau_j \in \mathbf{T}_m} \frac{C_j}{T_j} \leq 1$ and
    $C_i + \sum_{\tau_j \in \mathbf{T}_m} dbf^*(\tau_j, D_i) \leq D_i$ **then**
      assign task τ_i onto processor m and $\mathbf{T}_m \leftarrow \mathbf{T}_m \cup \{\tau_i\}$;
      break;

return feasible task assignment $\mathbf{T}_1, \mathbf{T}_2, \ldots, \mathbf{T}_M$;

Deadline-Monotonic Partition (Shortest Relative-Deadline First)

Input: $\mathbf{T}, M$;

re-index (sort) tasks such that $D_i \leq D_j$ for $i < j$;
$\mathbf{T}_i \leftarrow \emptyset, U_i \leftarrow 0, \forall m = 1, 2, \ldots, M$;
for $i = 1$ to N, where $N = |\mathbf{T}|$ **do**
  for $m = 1$ to M **do**
    if $\frac{C_i}{T_i} + \sum_{\tau_j \in \mathbf{T}_m} \frac{C_j}{T_j} \leq 1$ and
    $C_i + \sum_{\tau_j \in \mathbf{T}_m} dbf^*(\tau_j, D_i) \leq D_i$ **then**
      assign task τ_i onto processor m and $\mathbf{T}_m \leftarrow \mathbf{T}_m \cup \{\tau_i\}$;
      break;
  if τ_i is not assigned **then**
    return "The task assignment fails";
return feasible task assignment $\mathbf{T}_1, \mathbf{T}_2, \ldots, \mathbf{T}_M$;

Feasibility

Theorem

If the tasks are partitioned successfully by using Deadline-Monotonic Partition, EDF can feasibly schedule the tasks.

Proof

During the iteration, in which τ_i is decided to be assigned onto processor m, suppose that $\mathbf{T}'_m$ is the task set $\mathbf{T}_m$ before assigning τ_i to processor m.

- Let's prove by contradiction by assuming that τ_i misses its deadline, in which the response time $t_f > D_i$.
- Therefore, there exists $t_f > D_i$ such that $\sum_{\tau_j \in \mathbf{T}_m} dbf(\tau_j, t_f) > t_f$.
- By definition, $dbf^*(\tau_j, t) \geq dbf(\tau_j, t) \Rightarrow \sum_{\tau_j \in \mathbf{T}_m} dbf^*(\tau_j, t_f) > t_f$.
- By Deadline Monotonic Partition, $D_j \leq D_i$ for any $\tau_j \in \mathbf{T}'_m$, and

$$dbf^*(\tau_j, t_f) = C_j(1 + \frac{t_f - D_j}{T_j}) = dbf^*(\tau_j, D_i) + (t_f - D_i)\frac{C_j}{T_j}$$

Proof (cont.)

Proof

When τ_i is assigned on processor m, it must be $\frac{C_i}{T_i} + \sum_{\tau_j \in \mathbf{T}'_m} \frac{C_j}{T_j} \leq 1$ and $C_i + \sum_{\tau_j \in \mathbf{T}'_m} dbf^*(\tau_j, D_i) \leq D_i$.

$$t_f < \sum_{\tau_j \in \mathbf{T}_m} dbf^*(\tau_j, t_f) = \sum_{\tau_j \in \mathbf{T}_m} dbf^*(\tau_j, D_i) + (t_f - D_i)\frac{C_j}{T_j}$$

$$= \left(C_i + \sum_{\tau_j \in \mathbf{T}'_m} dbf^*(\tau_j, D_i) \right) + (t_f - D_i) \left(\frac{C_i}{T_i} + \sum_{\tau_j \in \mathbf{T}'_m} \frac{C_j}{T_j} \right)$$

$$\leq D_i + (t_f - D_i) \left(\frac{C_i}{T_i} + \sum_{\tau_j \in \mathbf{T}'_m} \frac{C_j}{T_j} \right)$$

$$\Rightarrow t_f - D_i < (t_f - D_i) \left(\frac{C_i}{T_i} + \sum_{\tau_j \in \mathbf{T}'_m} \frac{C_j}{T_j} \right) \Rightarrow 1 < \left(\frac{C_i}{T_i} + \sum_{\tau_j \in \mathbf{T}'_m} \frac{C_j}{T_j} \right)$$

When Is Deadline-Monotonic Partition Always Successful?

Define that

- δ_{max} is the maximum density, i.e., $\max_{\tau_i \in \mathbf{T}} \frac{C_i}{D_i}$.
- μ_{max} is the maximum utilization, i.e., $\max_{\tau_i \in \mathbf{T}} \frac{C_i}{T_i}$.
- δ_{max} and μ_{max} are both less than or equal to 1.
- δ_{sum} is the maximum total dbf density, i.e., $\max_{t>0} \sum_{\tau_i \in \mathbf{T}} \frac{dbf(\tau_i, t)}{t}$.
- μ_{sum} is the total utilization, i.e., $\sum_{\tau_i \in \mathbf{T}} \frac{C_i}{T_i}$.

Theorem

Any sporadic task set is successfully scheduled by Deadline-Monotonic Partition on M identical processors, when

$$M \geq \frac{2\delta_{\text{sum}} - \delta_{\text{max}}}{1 - \delta_{\text{max}}} + \frac{\mu_{\text{sum}} - \mu_{\text{max}}}{1 - \mu_{\text{max}}}.$$

Proof of Schedulability

We prove by contradiction. Suppose that τ_i is not able to be assigned on any processor by Deadline-Monotonic Partition.

- Either $\frac{C_i}{T_i} + \sum_{\tau_j \in \mathbf{T}'_m} \frac{C_j}{T_j} > 1$ or $C_i + \sum_{\tau_j \in \mathbf{T}'_m} dbf^*(\tau_j, D_i) > D_i$.
- Suppose that among M processors, there are M_1 processors, denoted by set $\mathbf{M}_1$, with $C_i + \sum_{\tau_j \in \mathbf{T}'_m} dbf^*(\tau_j, D_i) > D_i$.
- Therefore, by summing the inequalities together, we have

$$M_1 C_i + \sum_{\tau_j \in \mathbf{T}'_m \text{ and } m \in \mathbf{M}_1} dbf^*(\tau_j, D_i) > M_1 D_i.$$

$$\Rightarrow \sum_{\tau_j \in \mathbf{T} \setminus \{\tau_i\}} 2dbf(\tau_j, D_i) > M_1(D_i - C_i)$$

$$\Rightarrow \sum_{\tau_j \in \mathbf{T}} 2dbf(\tau_j, D_i) > M_1(D_i - C_i) + C_i \Rightarrow 2\delta_{\text{sum}} > M_1 \frac{D_i - C_i}{D_i} + \frac{C_i}{D_i}$$

- By above, we know that $M_1 < \frac{2\delta_{\text{sum}} - \frac{C_i}{D_i}}{1 - \frac{C_i}{D_i}}$.

Proof of Schedulability

- $M_2 = M - M_1$ processors are purely with $\frac{C_i}{T_i} + \sum_{\tau_j \in \mathbf{T}'_m} \frac{C_j}{T_j} > 1$.
- Similarly, $M_2 < \frac{\mu\text{sum} - \frac{C_i}{T_i}}{1 - \frac{C_i}{T_i}}$.
- Now, it is time to show that $M < \frac{2\delta\text{sum} - \delta_{\max}}{1 - \delta_{\max}} + \frac{\mu\text{sum} - \mu_{\max}}{1 - \mu_{\max}}$.
- Let's now consider 4 cases:
  - $\delta_{\text{sum}} \leq \frac{1}{2}$ and $\mu_{\text{sum}} \leq 1$: Both conditions in Step 5 will be successful. (prove by yourself.)
  - $\delta_{\text{sum}} \leq \frac{1}{2}$ and $\mu_{\text{sum}} > 1$: logically impossible, as by definition, $\delta_{\text{sum}} \geq \mu_{\text{sum}}$.
  - $\delta_{\text{sum}} > \frac{1}{2}$ and $\mu_{\text{sum}} \leq 1$: M_2 is always 0, and $M = M_1 < \frac{2\delta\text{sum} - \frac{C_i}{D_i}}{1 - \frac{C_i}{D_i}} \leq \frac{2\delta\text{sum} - \delta_{\max}}{1 - \delta_{\max}}$.
  - $\delta_{\text{sum}} > \frac{1}{2}$ and $\mu_{\text{sum}} > 1$: Clearly, $M = M_1 + M_2 < \frac{2\delta\text{sum} - \delta_{\max}}{1 - \delta_{\max}} + \frac{\mu\text{sum} - \mu_{\max}}{1 - \mu_{\max}}$.

Resource Augmentation

Theorem

If a sporadic task set is schedulable on M identical processors, Deadline-Monotonic Partition will successfully partition this task set upon a platform comprised of M processors that are each $(4 - \frac{2}{M})$ times as fast as the original.

Proof

The necessary condition to be schedulable on M identical processors is

$$\delta_{\max} \leq 1, \mu_{\max} \leq 1, \mu_{\text{sum}} \leq M, \text{ and } \delta_{\text{sum}} \leq M.$$

If speeding up by $\frac{1}{\gamma}$ satisfies $M \geq \frac{2\delta_{\text{sum}} - \delta_{\max}}{1-\delta_{\max}} + \frac{\mu_{\text{sum}} - \mu_{\max}}{1-\mu_{\max}}$, it is feasible by applying Deadline-Monotonic Partition. Therefore,

$$\frac{2\delta_{\text{sum}} - \delta_{\max}}{1 - \delta_{\max}} + \frac{\mu_{\text{sum}} - \mu_{\max}}{1 - \mu_{\max}} \leq \frac{2M\gamma - \gamma}{1 - \gamma} + \frac{M\gamma - \gamma}{1 - \gamma} = \frac{3M\gamma - 2\gamma}{1 - \gamma}$$

If $M \geq \frac{3M\gamma - 2\gamma}{1-\gamma}$, the schedulability is enforced. $\Rightarrow 4 - \frac{2}{M} \leq \frac{1}{\gamma}$.

Deadline-Monotonic Partition for Constrained Deadlines

Theorem

Any sporadic task set with constrained deadlines is successfully scheduled by Deadline-Monotonic Partition on M identical processors, when

$$M \geq \frac{2\delta_{\text{sum}} - \delta_{\max}}{1 - \delta_{\max}}.$$

Theorem

If a sporadic task set with constrained deadlines is schedulable on M identical processors, Deadline-Monotonic Partition will successfully partition this task set upon a platform comprised of M processors that are each $(3 - \frac{1}{M})$ times as fast as the original.

Main difference: Condition $\frac{C_i}{T_i} + \sum_{\tau_j \in \mathbf{T}'_m} \frac{C_j}{T_j} \leq 1$ in Step 5 is always enforced when $C_i + \sum_{\tau_j \in \mathbf{T}_m} dbf^*(\tau_j, D_i) \leq D_i$ holds. ($\Rightarrow M_2$ is always 0).

Feasibility

Theorem

[Fisher and Baruah, RTSS 2005] The resource augmentation factor for Deadline-Monotonic Partition has

- a $4 - \frac{2}{M}$ resource augmentation factor for tasks with arbitrary deadlines
- a $3 - \frac{1}{M}$ resource augmentation factor for tasks with constrained deadlines

Feasibility

Theorem

[Fisher and Baruah, RTSS 2005] The resource augmentation factor for Deadline-Monotonic Partition has

- a $4 - \frac{2}{M}$ resource augmentation factor for tasks with arbitrary deadlines
- a $3 - \frac{1}{M}$ resource augmentation factor for tasks with constrained deadlines

Theorem

[Chen and Chakraborty, RTSS 2011] The Deadline-Monotonic Partition has

- a $3 - \frac{1}{M}$ resource augmentation factor for tasks with arbitrary deadlines
- a $\frac{3e-1}{e} - \frac{1}{M} \approx 2.6322 - \frac{1}{M}$ resource augmentation factor for tasks with constrained deadlines