Global and Partitioned Scheduling

CPEN 432 Real-Time System Design

Arpan Gujarati
University of British Columbia

Partitioned Scheduling

Reasonable Algorithms

- **Reasonable allocation (RA):** An algorithm that fails to allocate a task to a multiprocessor platform only when the task does not fit into any processor on the platform

- When a task is considered for assignment, to which processor does it get assigned?
    - **First-fit (FF):** the processors are considered ordered in some manner and the task is assigned to the first processor on which it fits
    - **Worst-fit (WF):** the task is assigned to the processor with the maximum remaining capacity
    - **Best-fit (BF):** the task is assigned to the processor with the minimum remaining capacity exceeding its own utilization (i.e., on which it fits)

- In what order are the tasks considered for assignment?
    - **Decreasing (D):** tasks are considered in non-increasing order of their utilizations
    - **Increasing (I):** tasks are considered in non-decreasing order of their utilizations
    - **Unordered (ε):** tasks are considered in arbitrary order (i.e., tasks need not be sorted prior to allocation)

- Nine different heuristics: **{FF, WF, BF} x {D, I, ε}**, i.e., FFD, FFI, FF, WFD, WFI, WF, BFD, BFI, BF

Utilization Bounds

- Let α denote an upper bound on the per-task utilization, and $\beta = \lfloor 1/\alpha \rfloor$
- For any reasonable allocation algorithm, its utilization bound U_b is bounded as follows: $m - (m-1)\alpha \leq U_b \leq (\beta m + 1)/(\beta + 1)$
- **WF and WFI:** $U_b = m - (m-1)\alpha$
- **FF, FFI, FFD, BF, BFI, BFD, and WFD:** $U_b = (\beta m + 1)/(\beta + 1)$
- What if α is unknown?

Reasonable Algorithms

- Nine different heuristics: **{FF, WF, BF} x {D, I, ε}**
- Each can be implemented extremely efficiently
    - Sorting n tasks: $O(n \log n)$
    - Choosing a fit for any given task: $O(m)$
- From Multiprocessor Scheduling for Real-Time Systems (Baruah et al., 2015)
    - *"… **it seems reasonable to actually run the partitioning algorithm, rather than computing the utilization of the task system and comparing against the algorithm's (sufficient, not exact) utilization bound** … from the perspective of actually implementing a real-time system using partitioned scheduling, there is **no particular significance to using a utilization bound formula rather than actually trying out the algorithms**. Rather, the major benefit to determining these bounds arises from the insight such bounds may provide regarding the efficacy of the algorithm."* [Emphasis added]

Speedup Factors

- Consider partitioning algorithms $\mathscr{A}_{optimal}$ (optimal) and $\mathscr{A}_{heuristic}$ (approximate)

- Speedup factor of $\mathscr{A}_{heuristic}$
  - The smallest number f such that any task system that can be partitioned by $\mathscr{A}_{optimal}$ upon a particular platform can be partitioned by $\mathscr{A}_{heuristic}$ upon a platform in which each processor is f times faster

- Nine different heuristics: **{FF, WF, BF} x {D, I, ε}**
  - $f_{FFD,WFD,BFD} = \frac{4}{3} - \frac{1}{3m}, f_{WF,WFI} = 2 - \frac{2}{m}, f_{FF,FFI,BF,BFI} = 2 - \frac{2}{m+1}$

- More expressive than utilization bounds
  - As observed in practice, the speedup factors show that when tasks are considered in non-increasing order of their utilizations (i.e., FFD, WFD, BFD), partitioning is easier

- Question: What is the speedup factor of algorithm $\mathscr{A}'_{optimal}$ that is also an optimal algorithm?

A PTAS for Partitioning

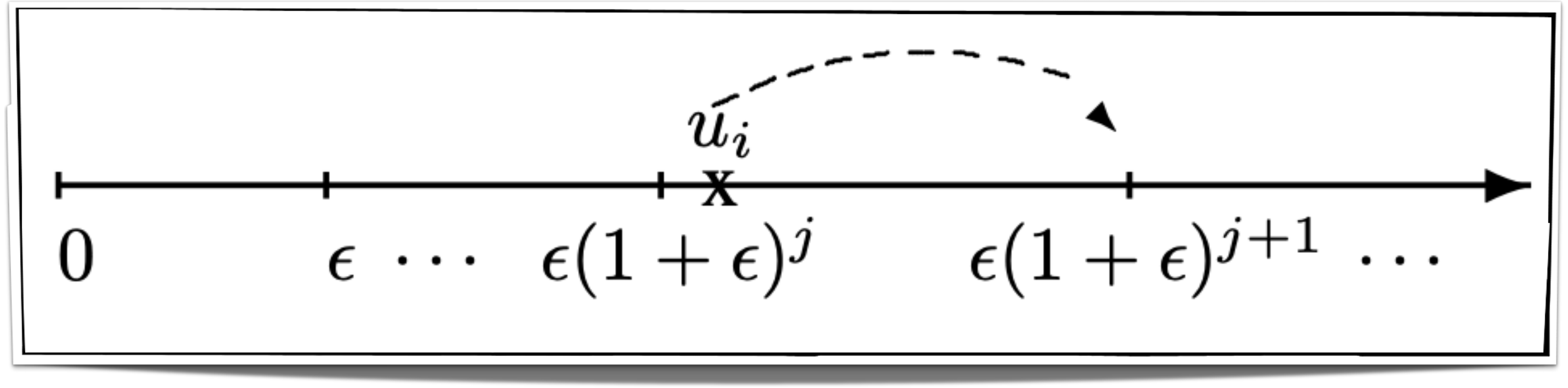
- Optimal partitioning is NP-hard
- Common heuristics have a speedup factor of 4/3 or 2 as $m \to \infty$
- Is there a Polynomial-Time Approximation Scheme (PTAS) that can achieve a speedup factor of $1 + \epsilon$, for any positive constant ϵ?
  - If yes, we can partition a task set to any desired degree of accuracy in polynomial time
  - Of course, we will need faster processors in order to run the tasks :-)
- Next few slides
  - PTAS for partitioning proposed by Hochbaum and Shmoys (1987)
  - Implementation by Chattopadhyay and Baruah (2011)

Key Ideas [1/5]

- Choosing ϵ
  - PTAS requires processors that are $1 + \epsilon$ times faster
    - We cannot provide faster processors
    - But we can assume that $\mathscr{A}_{optimal}$ has only $\left(\frac{1}{1+\epsilon}\right)^{th}$ of each processor available
    - Thus, if $\mathscr{A}_{optimal}$ can partition a task set on m processors with an available utilization of $(1/1 + \epsilon)$ on each processor, then $\mathscr{A}_{PTAS}$ can partition the task set on m processors with full utilization available
  - Suppose we are willing to tolerate a loss of up to 10% of the processor utilization
    - Thus, $U_{loss} = 1 - \frac{1}{1+\epsilon} = \frac{\epsilon}{1+\epsilon} = 0.1$ (i.e. 10%), which yields $\epsilon = \frac{1}{9}$

Key Ideas [2/5]

- Bucketing utilization values
  - The utilization of a task u_i may vary anywhere from 0 to 1, i.e., infinitely many possibilities
  - Instead, we consider only a finite number of points $V(\epsilon) = (v_0, v_1, v_2, v_3, \ldots)$ as valid utilizations
    - Where $v_j = \epsilon \times (1 + \epsilon)^j \leq 1$
  - Any utilization $v_j < u_i < v_{j+1}$ is inflated to the next valid utilization v_{j+1}



  - For example, $V(\epsilon = 0.3) = \{0.3, 0.39, 0.507, 0.6591, 0.8568\}$

Key Ideas [3/5]

- Enumerate all maximal single-processor configurations
  - Each configuration identifies a vector $\langle x_1, x_2, \ldots, x_{|V(\epsilon)|} \rangle$
    - Where x_i identifies the number of tasks with utilization $V(\epsilon)[i]$
    - Such that $x_1 v_1 + x_2 v_2 + \ldots x_{|V(\epsilon)|} v_{|V(\epsilon)|} \leq 1$
    - I.e., each configuration identifies a set of tasks that can be scheduled on a uniprocessor
  - The configuration is maximal if no other task can be further added
    - I.e., no x_i can be incremented without violating the above inequality

Config. ID	0.3000	0.3900	0.5070	0.6591	0.8568
1	3	0	0	0	0
2	2	1	0	0	0
3	1	0	1	0	0
4	1	0	0	1	0
5	0	2	0	0	0
6	0	1	1	0	0
7	0	0	0	0	1

Just seven maximal single-processor configurations for $\epsilon = 0.3$

Key Ideas [4/5]

- Enumerate all maximal multi-processor configurations

0.3	0.39	0.507	0.6591	0.8568	**Single-proc. ID's**
3	2	1	2	0	[4 4 5 3]
3	4	2	0	0	[6 6 5 1]
0	3	3	0	1	[6 6 6 7]
4	1	1	1	1	[7 4 3 2]
4	0	1	3	0	[4 4 4 3]

Example configurations for $m = 4$ and $\epsilon = 0.3$ (out of 140)

Config. ID	0.3000	0.3900	0.5070	0.6591	0.8568
1	3	0	0	0	0
2	2	1	0	0	0
3	1	0	1	0	0
4	1	0	0	1	0
5	0	2	0	0	0
6	0	1	1	0	0
7	0	0	0	0	1

Just seven maximal single-processor configurations for $\epsilon = 0.3$

Key Ideas [5/5]

$$\frac{1}{5}, \frac{1}{5}, \frac{1}{3}, \frac{7}{20}, \frac{9}{25}, \frac{2}{5}, \frac{1}{2}, \frac{1}{2}, \frac{3}{4}$$

Example task set

- Task assignment
  - Step 1: Round up task utilizations to values in $V(\epsilon = 0.3) = \{0.3, 0.39, 0.507, 0.6591, 0.8568\}$
  - Step 2: Ignore "small" tasks with utilization less than $\epsilon/1 + \epsilon$
  - Step 3: For the remaining "large" tasks, identify a matching configuration from the multi-processor lookup table
  - Step 4: Assign these "large" tasks to appropriate processors based on the chosen configuration
  - Step 5: Assign each "small" task to any processor upon which it fits

0.3	0.39	0.507	0.6591	0.8568	**Single-proc. ID's**
3	2	1	2	0	[4 4 5 3]
3	4	2	0	0	[6 6 5 1]
0	3	3	0	1	[6 6 6 7]
4	1	1	1	1	[7 4 3 2]
4	0	1	3	0	[4 4 4 3]

Example configurations for $m = 4$ and $\epsilon = 0.3$ (out of 140)

Config. ID	0.3000	0.3900	0.5070	0.6591	0.8568
1	3	0	0	0	0
2	2	1	0	0	0
3	1	0	1	0	0
4	1	0	0	1	0
5	0	2	0	0	0
6	0	1	1	0	0
7	0	0	0	0	1

Just seven maximal single-processor configurations for $\epsilon = 0.3$

Global Scheduling

Global vs. Partitioned: Pros & Cons

- **Global** scheduling
  - ✓ Automatic load balancing
  - ✓ Lower avg. response time
  - ✓ Simpler implementation
  - ✓ *Optimal* schedulers exist
  - ✓ More efficient reclaiming
  - ✗ Migration costs
  - ✗ Inter-core synchronization
  - ✗ Loss of cache affinity
  - ✗ Weak scheduling framework

- **Partitioned** scheduling
  - ✓ Supported by automotive industry (e.g., AUTOSAR)
  - ✓ No migrations
  - ✓ Isolation between cores
  - ✓ Mature scheduling framework
  - ✗ Cannot exploit unused capacity
  - ✗ Rescheduling not convenient
  - ✗ NP-hard allocation

The Dhall Effect

- Consider an implicit-deadline sporadic task system of (m + 1) tasks to be scheduled upon an m-processor platform
  - Tasks $T_1, \ldots, T_m$ have parameters $(e_i = 1, P_i = P)$
  - Task T_{m+1} has parameters $e_{m+1} = P_{m+1} = P + 1$
- $U = m\left(\frac{1}{P}\right) + \frac{P+1}{P+1} = \frac{m}{P} + 1 \qquad \left[\lim_{P\uparrow\infty} U = 1\right]$
- Is this task set schedulable with Global EDF if all tasks are released simultaneously?
- What happens if we increase the number of processors?

This task system is not EDF-schedulable despite having a utilization close to 1

The utilization bound of global EDF is **very poor**: it is arbitrarily close to one regardless of the number of processors.

The Dhall Effect

- Consider an implicit-deadline sporadic task system of (m + 1) tasks to be scheduled upon an m-processor platform
  - Tasks $T_1, \ldots, T_m$ have parameters $(e_i = 1, P_i = P)$
  - Task T_{m+1} has parameters $e_{m+1} = P_{m+1} = P + 1$
- $U = m\left(\frac{1}{P}\right) + \frac{P+1}{P+1} = \frac{m}{P} + 1 \qquad \left[\lim_{P\uparrow\infty} U = 1\right]$
- Is this task set schedulable with Global EDF if all tasks are released simultaneously?
- What happens if we increase the number of processors?

> This task system is not EDF-schedulable despite having a utilization close to 1
>
> The utilization bound of global EDF is **very poor**: it is arbitrarily close to one regardless of the number of processors.

- **Question:** Is this task set partitioned-schedulable?
  - **Answer:** Yes! for example, when $m < P$, we need only two processors!

The Dhall Effect

- Dhall's Effect shows the limitation of global EDF and RM: both utilization bounds tend to 1, independently of the value of m.
- Researchers lost interest in global scheduling for ~25 years, since late 1990s.
- Such a limitation is related to EDF and RM, not to global scheduling in general

Global Scheduling: Negative Results

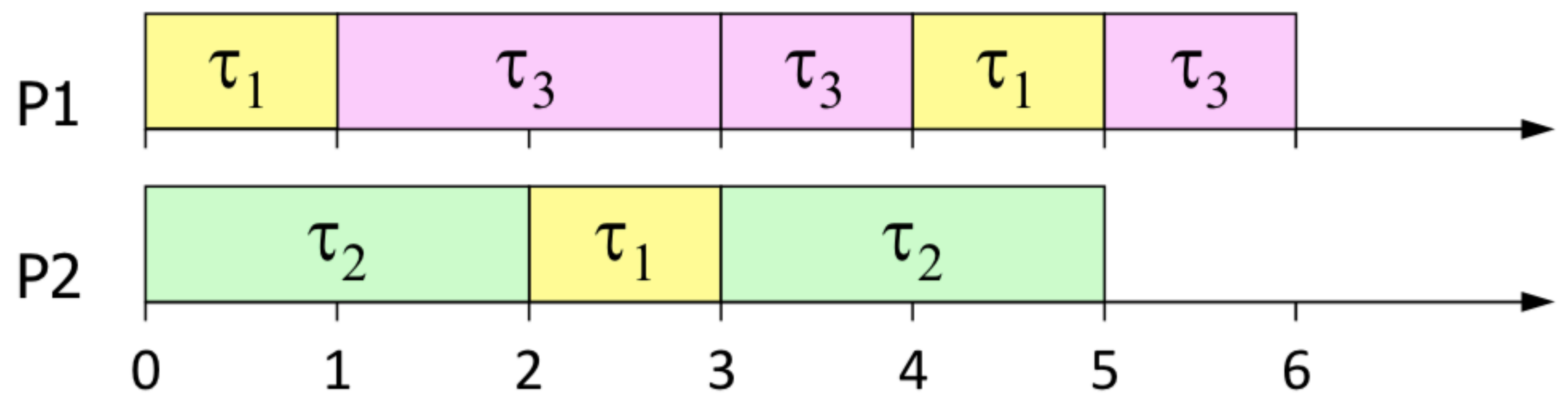
- Weak theoretical framework
  - Unknown critical instant
  - Global EDF is not optimal
  - Any global job-fixed (or task-dynamic) priority scheduler is not optimal
  - Optimality only for implicit deadlines
  - Many sufficient tests (most of them incomparable)

Global vs. Partitioned

- There are tasks that are schedulable only with a **global** scheduler!

Example:

	C_i	T_i
τ_1	1	2
τ_2	2	3
τ_3	2	3

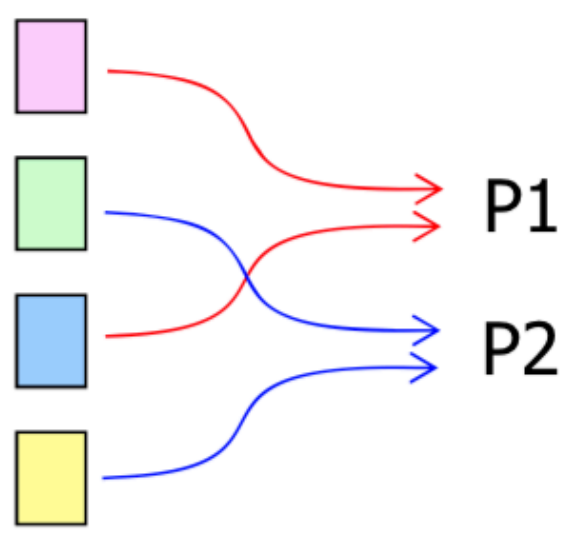


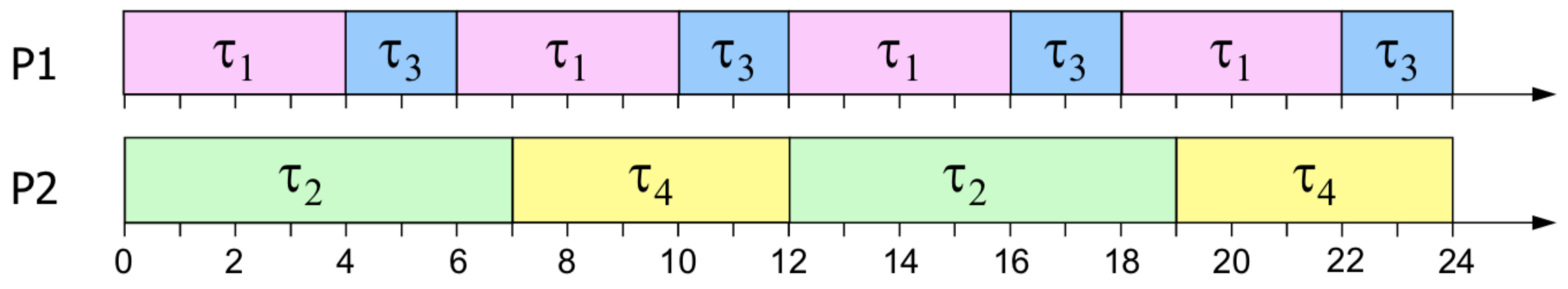
Global vs. Partitioned

- But there are also task sets that are schedulable only with a **partitioned** scheduler

Example:

	C_i	T_i
τ_1	4	6
τ_2	7	12
τ_3	4	12
τ_4	10	24





All 4! = 24 global priority assignments lead to deadline miss.

Global vs. Partitioned

- Example of an unfeasible global schedule with $\pi_1 > \pi_2 > \pi_3 > \pi_4$

	C_i	T_i
τ_1	4	6
τ_2	7	12
τ_3	4	12
τ_4	10	24