Optimal Fixed Priority Scheduling with Deferred Pre-emption

Aditya Saripalli
(20173071)

Introduction

- A common misconception with the Fixed Priority Scheduling of sporadic tasks (on a single processor) is that a "A fully pre-emptive scheduling" is the best approach for schedulability.

- Fixed Priority scheduling is broadly classified into 3 categories
  - ✓ FPPS : Fixed Priority Pre-emptive Scheduling
  - ✓ FPNS : Fixed Priority Non-pre-emptive Scheduling
  - ✓ FPDS : Fixed Priority Scheduling with Deferred Pre-emption

- *FPPS* and *FPNS* are incomparable, because the tasksets that are schedulable under FPPS may not be schedulable under FPNS and vice-versa

- *FPDS* scheduling refer to a variety of different techniques by which pre-emptions may be deferred for some period after a high priority task becomes ready.

Introduction

- FPDS is superior algorithm to FPPS and FPNS and therefore can schedule any taskset that is schedulable under FPPS and FPNS.

- This paper analyse a form of FPDS where each task has a final non pre-emptive region.

- There are 2 key parameters with FPDS
  i. The priority assigned to each task
  ii. The length of each task's final non pre-emptive region

- This paper introduces and optimal algorithm (FNR-PA) for FPDS. It relies on finding the minimum final non pre-emptive region length for each task that ensures its schedulability.

- FNR-PA : Final Non-pre-emptive region and Priority Assignment

History

- Two different models of fixed priority scheduling with deferred pre-emption have been developed in the literature

- *Fixed Model*
  - The location of each non-pre-emptive region is statically determined prior to execution.
  - Pre-emption is only permitted at pre-defined locations in the code for each task – 'Pre-emption Points'.
  - Also know as *co-operative scheduling*.

- *Floating Model*
  - An upper bound is set for the longest non-pre-emptive region of each task.
  - The location of each non-pre-emptive may vary at run-time.

- In 2011, Bertogna derived a method of computing the optimal length of the final non-pre-emptive region of each task in order to maximize schedulability under FPDS for a given priority assignment

Assumptions

- A discrete time model is assumed, where all task parameters are assumed to be positive integers.
- Due to the integer time model considered in this paper, the discrete time granularity is “1” time unit.
- The minimum possible length of a non-pre-emptive region is “1” rather than 0, as we assume a discrete time model and tasks cannot be pre-empted during a processor clock cycle.
- The arrival times of the tasks are independent and unknown a-priori. Hence the tasks may share a common release time.
- Each task is released (becomes ready to execute), as soon as it arrives. The tasks do not voluntarily suspend themselves.
- Any resource access occurring within the final non-pre-emptive region of a task is properly nested, i.e. wholly included within that region, avoiding deadlocks.

System Model, Terminology & Notation

- "i" denotes priority (t_1 highest ... t_n lowest).
- *lp(i)* : Set of tasks with priorities < "i"
- *lep(i)* : Set of tasks with priorities ≤ "i"
- *hp(i)* : Set of tasks with priorities > "i"
- *hep(i)* : Set of tasks with priorities ≥ "i"

- **C_i** : Worst case execution time of task t_i
- **R_i** : Worst case response time of task t_i
- **T_i** : Task period
- **D_i** : Relative Deadline
- **U_i** : Utilisation of task "t_i" is **(C_i / T_i)**

- **F_i** : Each task is assumed to have a final non-pre-emptive region of length in the range [1, C_i]
- $\boldsymbol{B_l^i}$: The longest time a task t_l may continue executing while holding a resource needed by another task of lower priority t_i. [$t_l \in$ *lp(i)*]
- **B_i** : The longest time that the task ti can be blocked by the lower priority tasks.

System Model, Terminology & Notation

Types of tasksets

- *Implicit-deadline* : A taskset where for all tasks $D_i = T_i$ (deadline = period)
- *Constraint-deadline* : A taskset where for all tasks $D_i \leq T_i$ (deadline ≤ period)
- *Arbitrary-deadline* : A taskset where for all the tasks, deadlines are independent of their periods.

Definitions

- *Schedulability:* A taskset is said to be schedulable with respect to some scheduling algorithm, if all valid sequences of jobs that may be generated by the taskset can be scheduled by the algorithm without any missed deadlines.

- *Optimality:* A priority assignment policy "P" is said to be optimal with respect to some class of tasksets and some type of fixed priority scheduling algorithm - if there are no tasksets in the class that are schedulable under the scheduling algorithm using any other priority ordering policy, that are not also schedulable using the priority assignment determined by policy "P".

- A fixed priority scheduling algorithm A is said to dominate another fixed priority scheduling algorithm B if there are tasksets that can be scheduled under algorithm A, but cannot be scheduled under algorithm B, and all of the tasksets that are schedulable under algorithm B are also schedulable under algorithm A.
- If there are tasksets that are schedulable under algorithm A, but not under algorithm B, and vice-versa, then the two algorithms are said to be incomparable.
- If both algorithms can schedule precisely the same tasksets then they are said to be equivalent.

Definitions

- *Sustainability of a Scheduling Algorithm:* A scheduling algorithm is said to be sustainable with respect to a system model, if and only if schedulability of any taskset compliant with the model implies schedulability of the same taskset modified by:
    1. Decreasing execution times
    2. Increasing periods or inter-arrival times, and
    3. Increasing deadlines

- *Sustainability of a Schedulabiltiy Test:* A schedulability test is referred to as sustainable if the above changes {1, 2, 3} cannot result in a taskset that was previously deemed schedulable by the test becoming un-schedulable.

Summarizing Prior Work

- *Priority level-i active period :* A continuous period [t1,t2) during which tasks, of priority "i" or higher, that were released at the start of the active period at t_1 , or during the active period but strictly before its end at t_2 , are either executing or ready to execute.

- *Δ-Critical Instance (for a task t_i) :* Occurs when t_i is released simultaneously with all higher priority tasks, and subsequent releases of task t_i and higher priority tasks occur after the minimum permitted time intervals.

- Further a minimum possible amount of time Δ prior to this simultaneous release, a lower priority task t_k enters its final non-pre-emptive region or begins accessing a resource shared with a task of priority "i" or higher.

- Previous work on FPDS established that, the longest response time of a task t_i occurs for some job of that task within the priority level-i active period starting at Δ-Critical Instant.

Summarizing Prior Work

- The worst-case length of a *priority level-i active period* $\boldsymbol{A_i}$ is given by the minimum solution to the following fixed-point iteration:

$$A_i^{m+1} = B_i + \sum_{\forall j \in hep(i)} \left\lceil \frac{A_i^m}{T_j} \right\rceil C_j \qquad (1)$$

- Iteration starts with $A_i^0 = C_i$ and ends when $A_i^{m+1} = A_i^m$
- The Blocking period $\boldsymbol{B_i}$ can be computed as follows:

$$B_i = \max\left(B_i^{RES}, B_i^{FNR}\right) \qquad (2)$$

$$B_i^{RES} = \max_{\forall\, l \in lp(i)} \left(B_l^i - 1\right) \qquad B_i^{FNR} = \max_{\forall\, l \in lp(i)} \left(F_l - 1\right)$$

Summarizing Prior Work

- The number of jobs Gi of the task ti in the priority level-I active period is given by: $G_i = \left\lceil \frac{A_i}{T_i} \right\rceil$ (3)

- The start time $W_{i,g}$ of the final non-pre-emptive region of job **g** of task $\mathbf{t_i}$ measured with respect to the start of the *Δ-critical instant* is given by the minimum solution to the following fixed-point iteration:

$$w_{i,g}^{m+1} = (B_i + (g+1)C_i - F_i) + \sum_{\forall\, i \in hp(i)} \left(\left\lfloor \frac{w_{i,g}^{m}}{T_j} \right\rfloor + 1 \right) C_j$$

{ *g=0 is the first job* }

- Iteration starts with $w_{i,g}^{0} = B_i + (g+1)C_i - F_i$

  and ends with $w_{i,g}^{m+1} = w_{i,g}^{m}$, in which case we have $W_{i,g} = w_{i,g}^{m+1}$

  or ends when $w_{i,g}^{m+1} + F_i - gT_i > D_i$, in which case job **g** and task $\mathbf{t_i}$ are *unschedulable*

Summarizing Prior Work

Thus the worst-case response time of the task $\mathbf{t_i}$ can be computed as:

$$R_i = \max_{\forall\, g=0,1,2\ldots G_i-1} \left(W_{i,g} + F_i - gT_i\right)$$

- The task $\mathbf{t_i}$ is *schedulable* when $R_i \leq D_i$

Problem Descriptions

Problem-1

- *Final Non-pre-emptive Region length (**FNR**) Problem*
  - For a given taskset complying with the task model described above and a *given priority ordering X*, find a value for the *length of the final non-pre-emptive region* of each task such that the taskset is schedulable under FPDS.

Problem-2

- *Final Non-pre-emptive Region length & Priority Assignment (**FNR-PA**) Problem*
  - For a given taskset complying with the task model described above, find both
  - A *priority assignment*, and
  - A value for the *length of the final non-pre-emptive region* of each task that makes the taskset schedulable under FPDS.

An Example

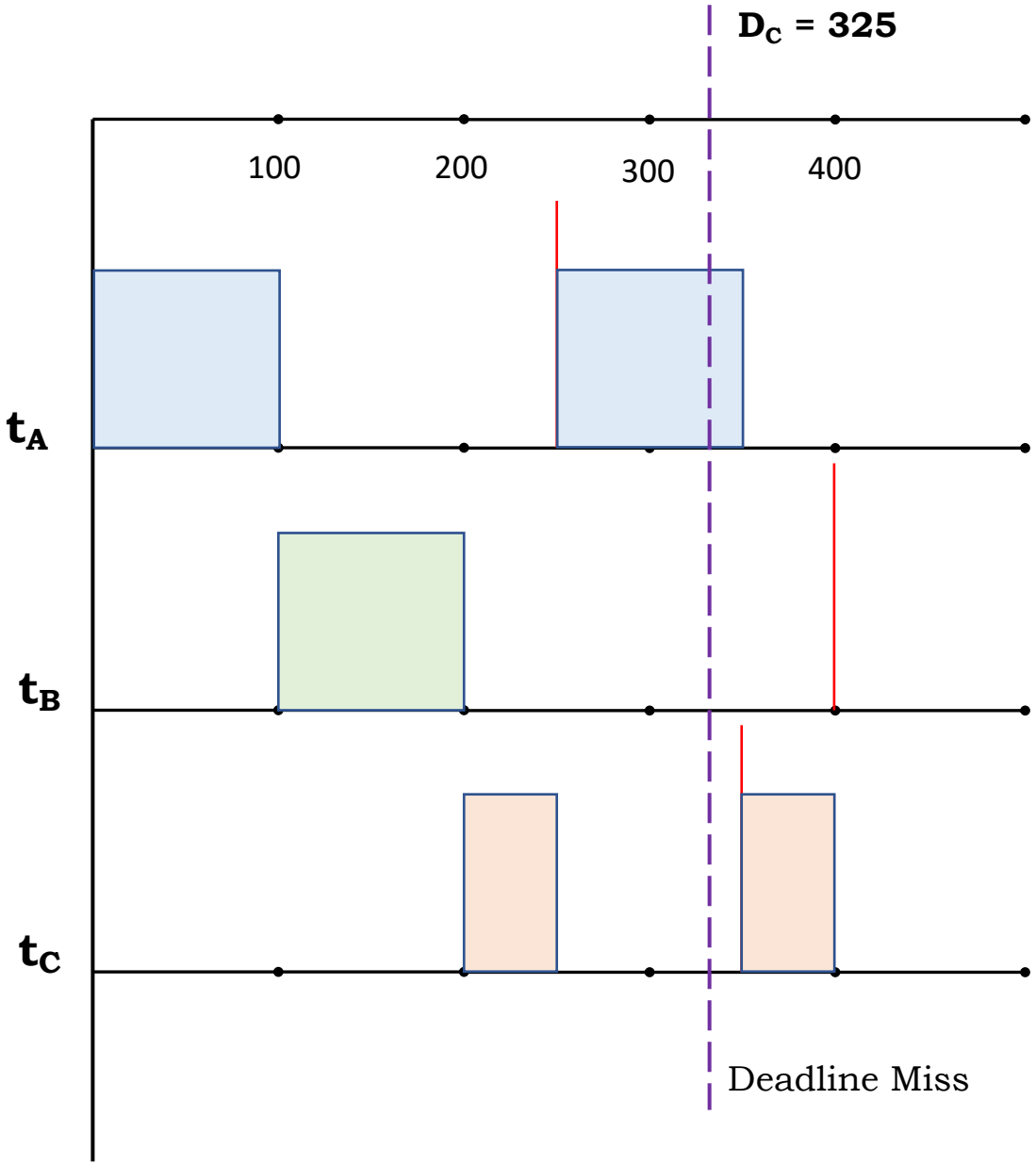
- Let us consider an example to show the effectiveness of FPDS scheduling algorithm over other fixed-priority scheduling algorithms
- Consider the following taskset:

Task	Period	Execution Time	Deadline
t_A	250	100	175
t_B	400	100	300
t_C	350	100	325

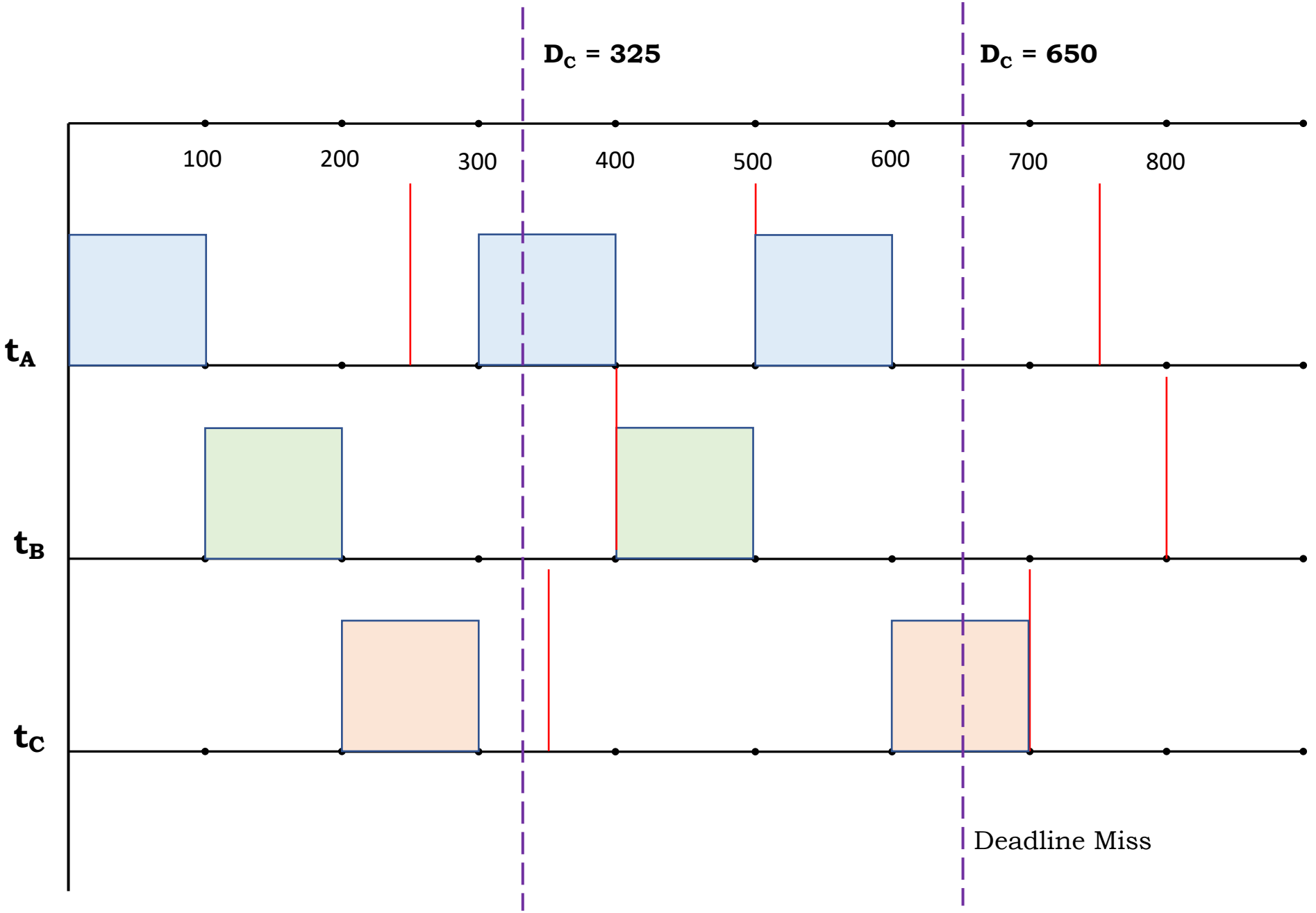
- We can see that Task tA should be given the highest priority, irrespective of whether we use FPPS or FPNS or FPDS.
- Considering deadline monotonic scheduling we get the priority order as:

$$P_A > P_B > P_C$$

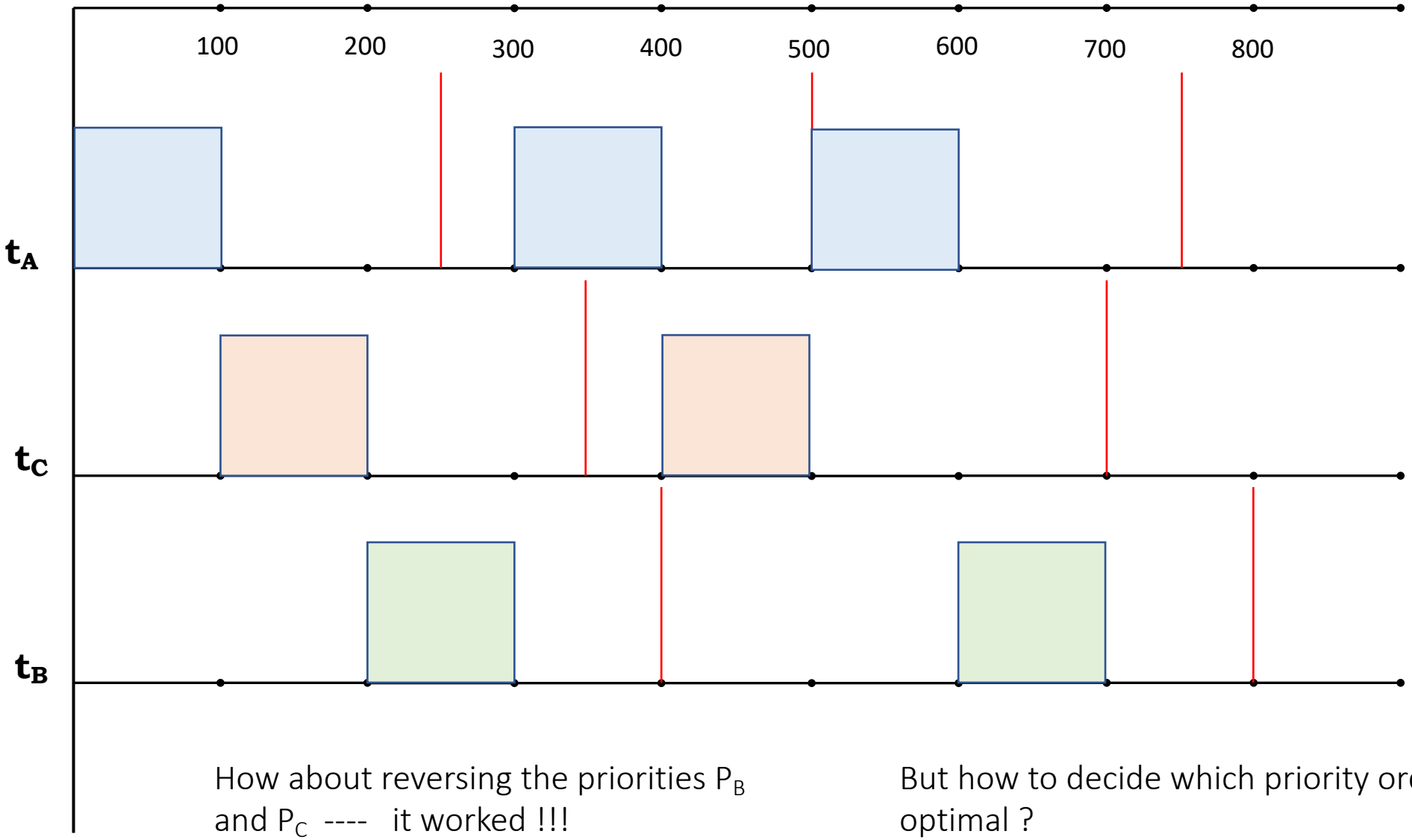
FPPS



FPNS


D_C = 325
D_C = 650
100
200
300
400
500
600
700
800
t_A
t_B
t_C
Deadline Miss

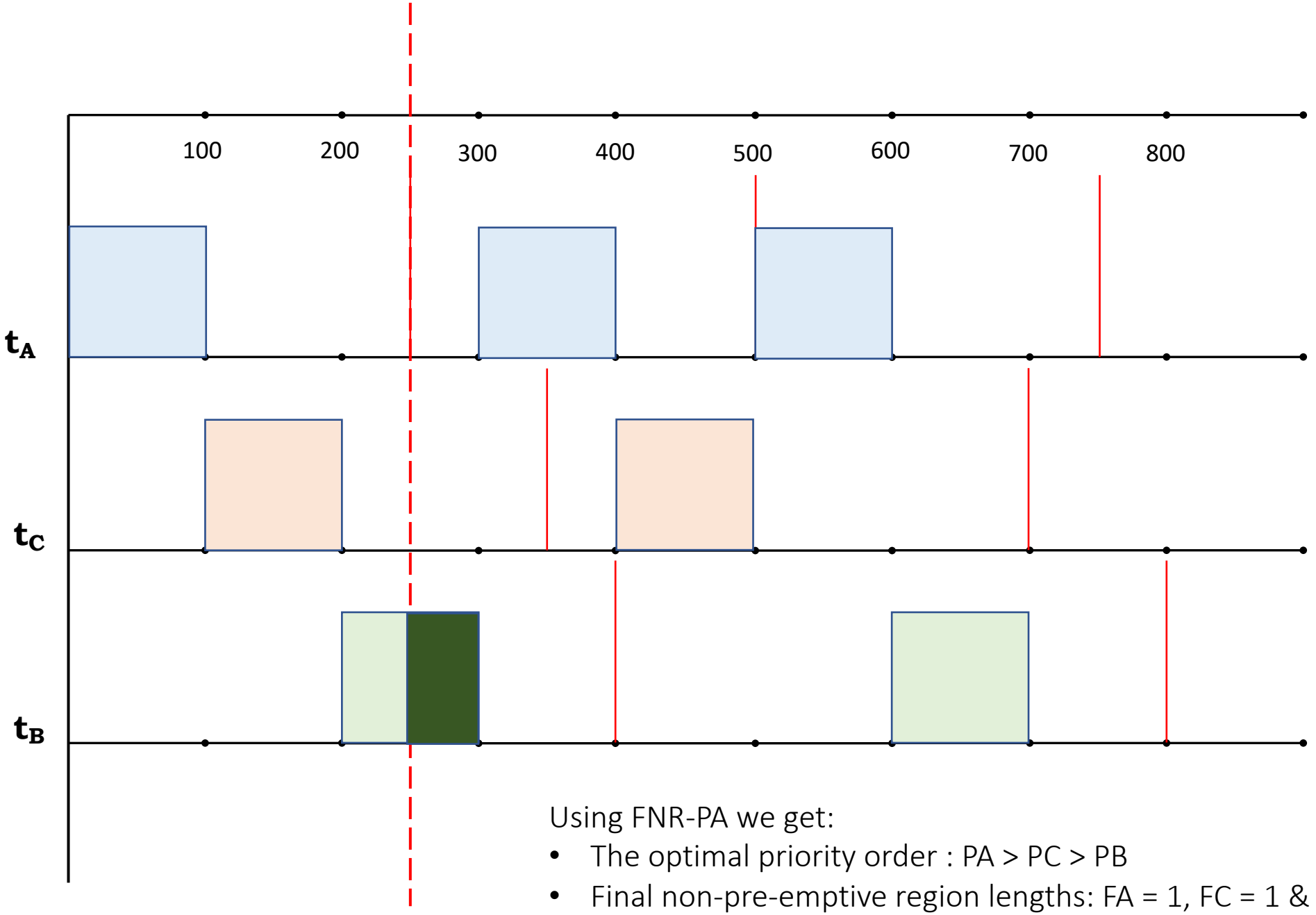
FPNS



How about reversing the priorities P_B and P_C ---- it worked !!!

But how to decide which priority order is optimal ?

FPDS



Using FNR-PA we get:

- The optimal priority order : PA > PC > PB
- Final non-pre-emptive region lengths: FA = 1, FC = 1 & FB = 51

Mathematical Approach

$$A_i^{m+1} = B_i + \sum_{\forall j \in hep(i)} \left\lceil \frac{A_i^m}{T_j} \right\rceil C_j$$

$$B_i = \max\left(B_i^{RES}, B_i^{FNR}\right)$$

$$B_i^{RES} = \max_{\forall\, l \in lp(i)} \left(B_l^i - 1\right) \;\&\; B_i^{FNR} = \max_{\forall\, l \in lp(i)} (F_l - 1)$$

Based on the computations shown we get the convergence at:

$$A_3^6 = A_3^5 = 700$$

Thus the worst-case length of priority level-3 active period is:

$$\boldsymbol{A_3 = 700}$$

Initially $A_i^0 = C_i$ which gives us $A_3^0 = C_3 = C_c = 100$

The blocking period $B_3 = \max(0,0) = 0$

Now From the iterative result we have:

$$A_3^1 = 0 + \sum_{j=1,2,3} \left\lceil \frac{A_3^0}{T_j} \right\rceil C_j$$

$$A_3^1 = 0 + \left\lceil \frac{A_3^0}{T_1} \right\rceil C_1 + \left\lceil \frac{A_3^0}{T_2} \right\rceil C_2 + \left\lceil \frac{A_3^0}{T_3} \right\rceil C_3 = 100 + 100 + 100 = 300$$

$$A_3^2 = 0 + \left\lceil \frac{A_3^1}{T_1} \right\rceil C_1 + \left\lceil \frac{A_3^1}{T_2} \right\rceil C_2 + \left\lceil \frac{A_3^1}{T_3} \right\rceil C_3 = 200 + 100 + 100 = 400$$

$$A_3^3 = 0 + \left\lceil \frac{A_3^2}{T_1} \right\rceil C_1 + \left\lceil \frac{A_3^2}{T_2} \right\rceil C_2 + \left\lceil \frac{A_3^2}{T_3} \right\rceil C_3 = 200 + 100 + 200 = 300$$

$$A_3^4 = 0 + \left\lceil \frac{A_3^3}{T_1} \right\rceil C_1 + \left\lceil \frac{A_3^3}{T_2} \right\rceil C_2 + \left\lceil \frac{A_3^3}{T_3} \right\rceil C_3 = 200 + 200 + 200 = 600$$

$$A_3^5 = 0 + \left\lceil \frac{A_3^5}{T_1} \right\rceil C_1 + \left\lceil \frac{A_3^5}{T_2} \right\rceil C_2 + \left\lceil \frac{A_3^5}{T_3} \right\rceil C_3 = 300 + 200 + 200 = 700$$

$$A_3^6 = 0 + \left\lceil \frac{A_3^5}{T_1} \right\rceil C_1 + \left\lceil \frac{A_3^5}{T_2} \right\rceil C_2 + \left\lceil \frac{A_3^5}{T_3} \right\rceil C_3 = 300 + 200 + 200 = \mathbf{700}$$

Mathematical Approach

$$w_{i,g}^{m+1} = (B_i + (g+1)C_i - F_i) + \sum_{\forall\, i \in hp(i)} \left(\left\lfloor \frac{w_{i,g}^{m}}{T_j} \right\rfloor + 1 \right) C_j$$

$$B_i = \max\left(B_i^{RES}, B_i^{FNR}\right)$$

$$B_i^{RES} = \max_{\forall\, l \in lp(i)} \left(B_l^i - 1\right) \;\&\; B_i^{FNR} = \max_{\forall\, l \in lp(i)} (F_l - 1)$$

The start time of final non-pre-emptive regions of task $\boldsymbol{t_C}$ is:

$$300 - x = \mathbf{249}$$

Thus the minimum lengths of final non-pre-emptive region for all the tasks is:

$$\boldsymbol{F_A = 1},\ \boldsymbol{F_B = 51}\ \textit{and}\ \boldsymbol{F_C = 1}$$

The blocking period $B_3 = \max(0,0) = 0$ and $C_3 = 100$
Also $w_{3,g}^0 = B_3 + (g+1)C_3 - F_3$ and let $F_3 = F_B = x$
Consider the first job of the task $t_C \Longrightarrow g = 0$

$$w_{3,0}^1 = (100 - x) + \sum_{j=1,2} \left(\left\lfloor \frac{w_{3,0}^0}{T_j} \right\rfloor + 1 \right) C_j$$

$$w_{3,0}^1 = (100 - x) + \left(\left\lfloor \frac{w_{3,0}^0}{T_1} \right\rfloor + 1 \right) C_1 + \left(\left\lfloor \frac{w_{3,0}^0}{T_2} \right\rfloor + 1 \right) C_2$$

$$= (100 - x) + \left(\left\lfloor \frac{100-x}{250} \right\rfloor + 1 \right) 100 + \left(\left\lfloor \frac{100-x}{400} \right\rfloor + 1 \right) 100$$

$$= (100 - x) + 100 + 100 = (300 - x)$$

$$w_{3,0}^2 = (100 - x) + \left(\left\lfloor \frac{w_{3,0}^1}{T_1} \right\rfloor + 1 \right) C_1 + \left(\left\lfloor \frac{w_{3,0}^1}{T_2} \right\rfloor + 1 \right) C_2$$

$$= (100 - x) + \left(\left\lfloor \frac{300-x}{250} \right\rfloor + 1 \right) 100 + \left(\left\lfloor \frac{300-x}{400} \right\rfloor + 1 \right) 100$$

$$= (100 - x) + 100 + 100 = (\mathbf{300 - x})$$

For the above convergence to happen we need:

$$\left\lfloor \frac{300-x}{250} \right\rfloor \to 0 \Rightarrow\ 300 - x < 250\ \Rightarrow\ x > 50\ \Rightarrow\ \boldsymbol{x_{min} = F_B = 51}$$

Similarly we get $F_A = 1$ & $F_C = 1$

Algorithms

FNR

```
for each priority level k, lowest first
{
  # determine the smallest value for the
    final non-pre-emptive region length F(k)
    such that the task at priority k is
    schedulable.

  # Set the length of the final non-pre-
    emptive region of the task to this value.
}
```

FNR-PA

```
for each priority level k, lowest first
{
  for each unassigned task
  {
    # determine the smallest value for the final non-pre-emptive
      region length F(k) such that task 't' is schedulable at
      priority k, assuming all other unassigned tasks have higher
      priorities.
    # record as task 'Z' the unassigned task with the minimum value
      for the length of its final non-pre-emptive region F(k).
  }
  if no tasks are schedulable at priority k
  {
    return unschedulable
  }
  else
  {
    # assign priority k to task 'Z'
    # use the value of F(k) as the length of its final non-pre-
      emptive region.
  }
}
return schedulable
```

FNR-PA Outline

- Let the algorithm start with an arbitrary priority order of X_n and let the iteration count be k
- It iteratively transforms the priority order as : $X_n \rightarrow X_{n-1} \rightarrow \ldots \rightarrow X_1$
- On k^{th} iteration the FNR-PA algorithm examines all the tasks of priority $\geq k$ in X_k.
- Of those tasks it selects the one that is schedulable at priority k, with the minimum FNR length, and assigns it to that priority level.
- Finally we arrive at the optimal priority order $P = X_1$

FNR-PA Transformation

- Find the priority level i in X_k of the task that the FNR-PA algorithm selects.
- We refer to this task as t_k (since the FNR-PA algorithm will assign it to priority level k)
- As the tasks of lower priority than k are the same in both X_k and P, the priority level $i \geq k$.

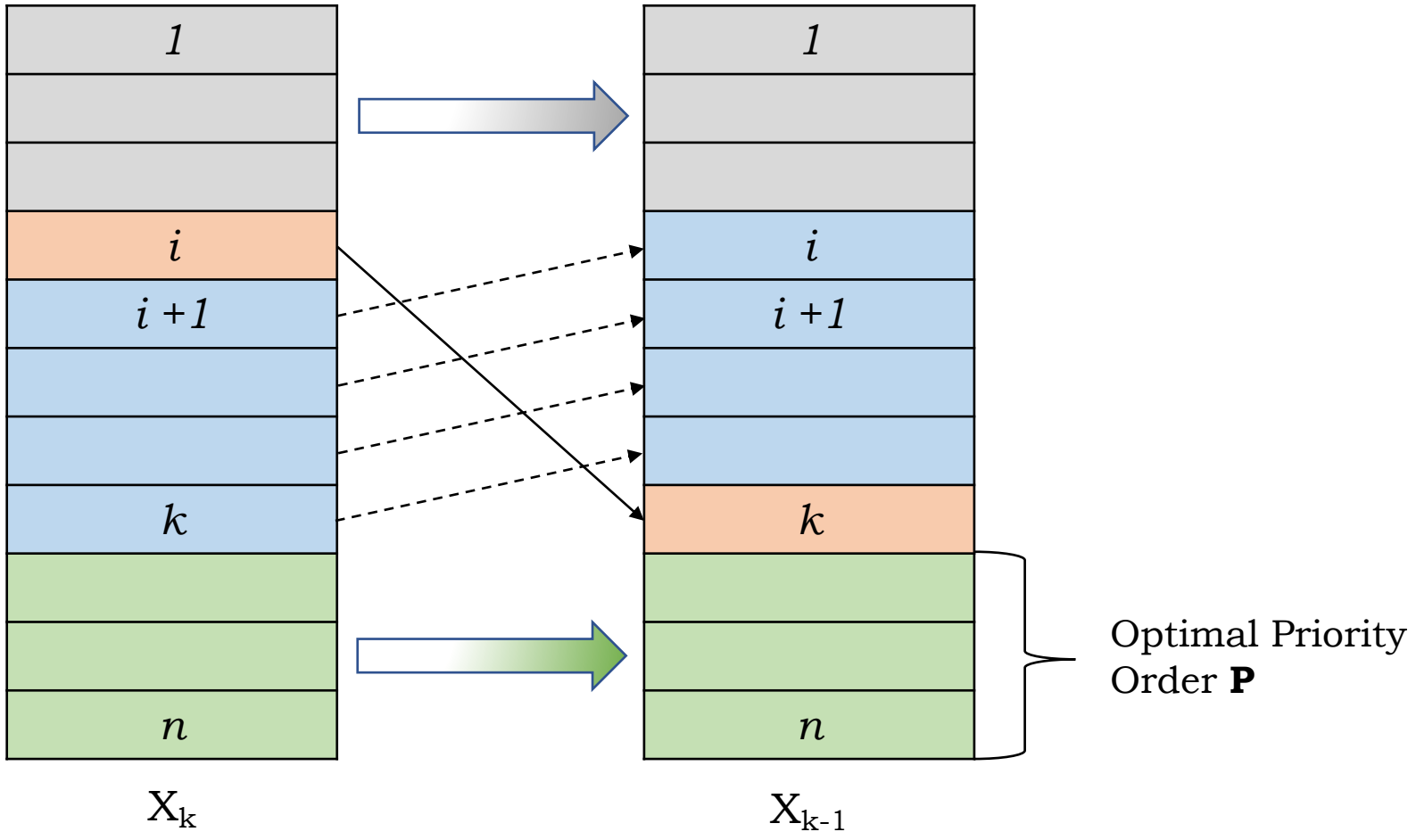
Case-1: *Task t_k is at priority k in both P and X_k*

- In this case no transformation is required on this iteration
- X_{k-1} is same as X_k

Case-2: *Task t_k is at higher priority i in X_k*

- In this case the task t_k is moved down in priority from priority level i to priority level k
- All the tasks from priority level $i+1$ to k are all moved up one priority level

FNR-PA Transformation



Greedily building the Optimal Priority Order

FNR-PA Transformation

$t_l \in lp(k, X_k)$ and **$t_h \in hp(k, X_{k-1})$**

- These tasks are assigned same priorities in X_k and X_{k-1}
- Thus these tasks remain schedulable with unchanged FNR length as per priority ordering k.

t_k

- Task t_k is at higher priority level i in X_k and at lower priority level k in X_{k-1}.
- As per the FNR-PA algorithm task t_k is schedulable at priority k and tolerates the final non-preemptive region length at priority k of any of the tasks in $hep(k\ X_k)$.

$t_m \in hp(k, X_{k-1}) \cap lep(i, X_{k-1})$

- These tasks maintain the same priority order but are shifted up one priority level in X_{k-1}.
- In priority order X_k, t_m is subjected to interference from t_k which is at higher priority, whereas in X_{k-1}, t_m is subjected to blocking from t_k which is at lower priority.
- However the blocking factor from t_k at the lower priority level in X_{k-1} cannot exceed its execution time C_k and the reduction in interference is also at least C_k.
- Thus task t_m remains schedulable under priority order X_{k-1} with its final non-pre-emptive region length set as per priority ordering X_k.

FNR Length Calculation

- The FNR and FNR-PA algorithms presented in the previous section need to compute the minimum final non-pre-emptive region length (FNR Length) for each task.

- One way is to find the length using a binary search.

- However there is a much efficient analytical method that can be used instead and reduce the overall complexity of the algorithms.

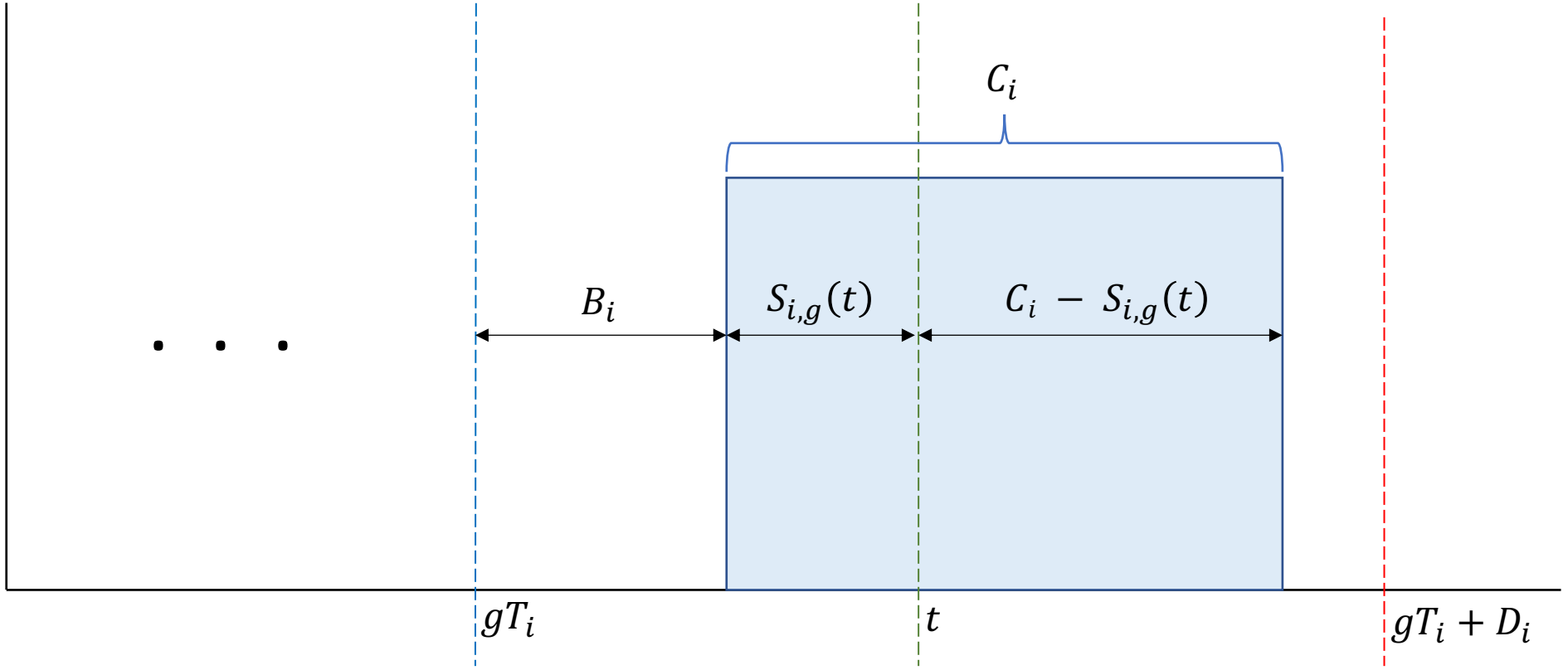
- To determine the minimum final non-pre-emptive region length $F(i)$, we effectively need to compute the largest amount of execution that the task can complete pre-emptively without missing its deadline.

FNR Length Calculation

- From equations (1) and (3) we can compute the length of priority level-i active period (A_i), and then we can get the number of jobs in A_i for the task t_i.

- For each job g of task t_i in A_i, we need to compute the minimum final non-pre-emptive region required to guarantee the schedulability of that job.

- For that, we consider the minimum amount of execution $S_{i,g}(t)$ that task t_i is certain to be able to perform between the start of its g^{th} job and some arbitrary time $\boldsymbol{t}$ prior to that job's deadline $gT_i + D_i$.



$$S_{i,g}(t) = t - B_i - gC_i - \sum_{\forall\, j \in hp(i)} \left(\left\lfloor \frac{t}{T_j} \right\rfloor + 1 \right) C_j$$

Remaining Execution Time: $C_i - S_{i,g}(t)$

Time to reach Deadline: $gT_i + D_i - t$

FNR Length Calculation

- The function $S_{i,g}(t)$ is a piecewise linear function with local maxima corresponding to the release times of higher priority tasks minus one-time unit.

- Let for each job $g = 0,1,2 \ldots G_i - 1$, $P_{i,g}$ be the set of points corresponding to the local maxima of $S_{i,g}(t)$ for the interval $t \in [gT_i, gT_i + D_i - 1]$

$$P_{i,g} = \left(\forall_{j \in hp(i)} \{hT_j - 1\} \in [gT_i, gT_i + D_i - 1]\right) \cup \{gT_i + D_i - 1\}$$

- To find the maximum $S_{i,g}(t)$ it is sufficient to check the schedulable points in $P_{i,g}$

$$S_{i,g} = \max_{t \in P_{i,g}} \{S_{i,g}(t) \mid C_i - S_{i,g}(t) \leq gT_i + D_i - t \ \wedge \ S_{i,g}(t) \geq 0\}$$

- The first inequality guarantees that the job has sufficient time to complete before deadline and the second inequality guarantees that the final non-pre-emptive region can start executing at or before time t.

FNR Length Calculation

$$S_{i,g} = \max_{t \in P_{i,g}} \left\{ S_{i,g}(t) \middle| C_i - S_{i,g}(t) \leq gT_i + D_i - t \ \wedge \ S_{i,g}(t) \geq 0 \right\}$$

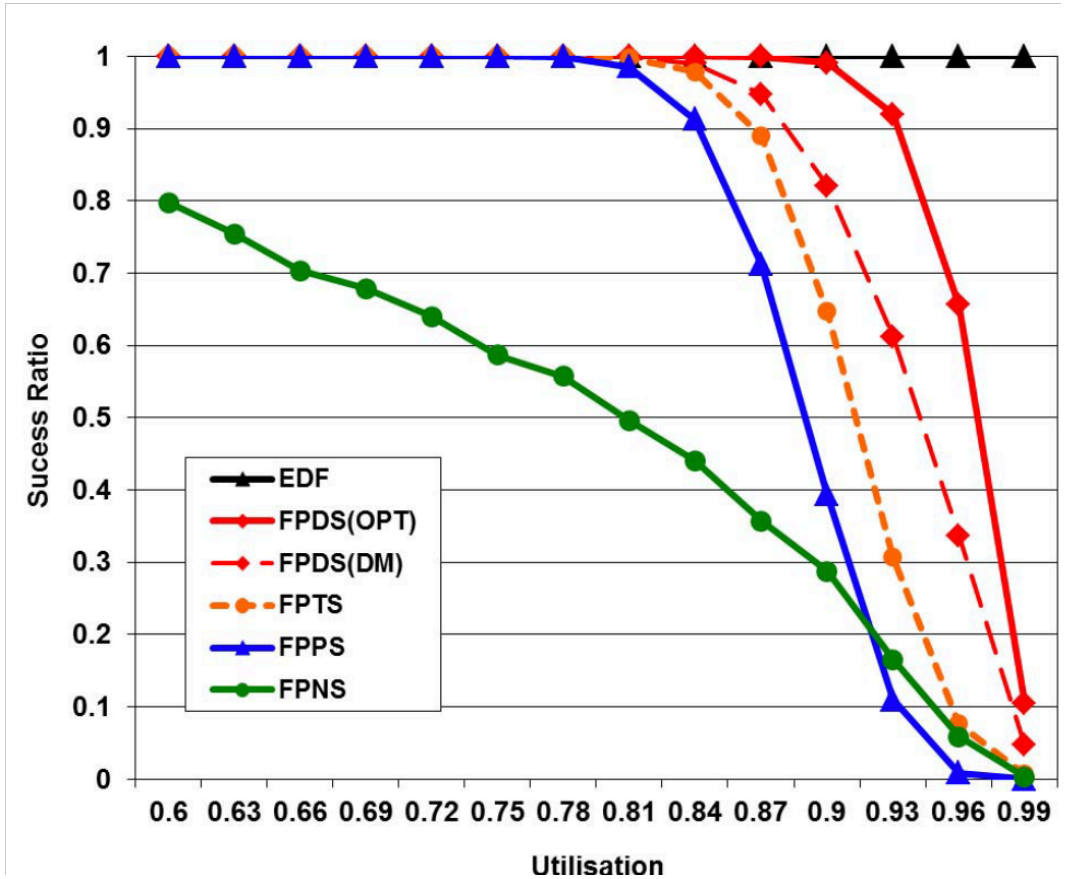
- The first inequality guarantees that the job has sufficient time to complete before deadline and the second inequality guarantees that the final non-pre-emptive region can start executing at or before time t.
- If no point in $P_{i,g}$ satisfies both inequalities, then the job is not schedulable and $S_{i,g}$ is set to a negative value.
- The minimum final non-pre-emptive region of the g^{th} job of the task t_i is given by:

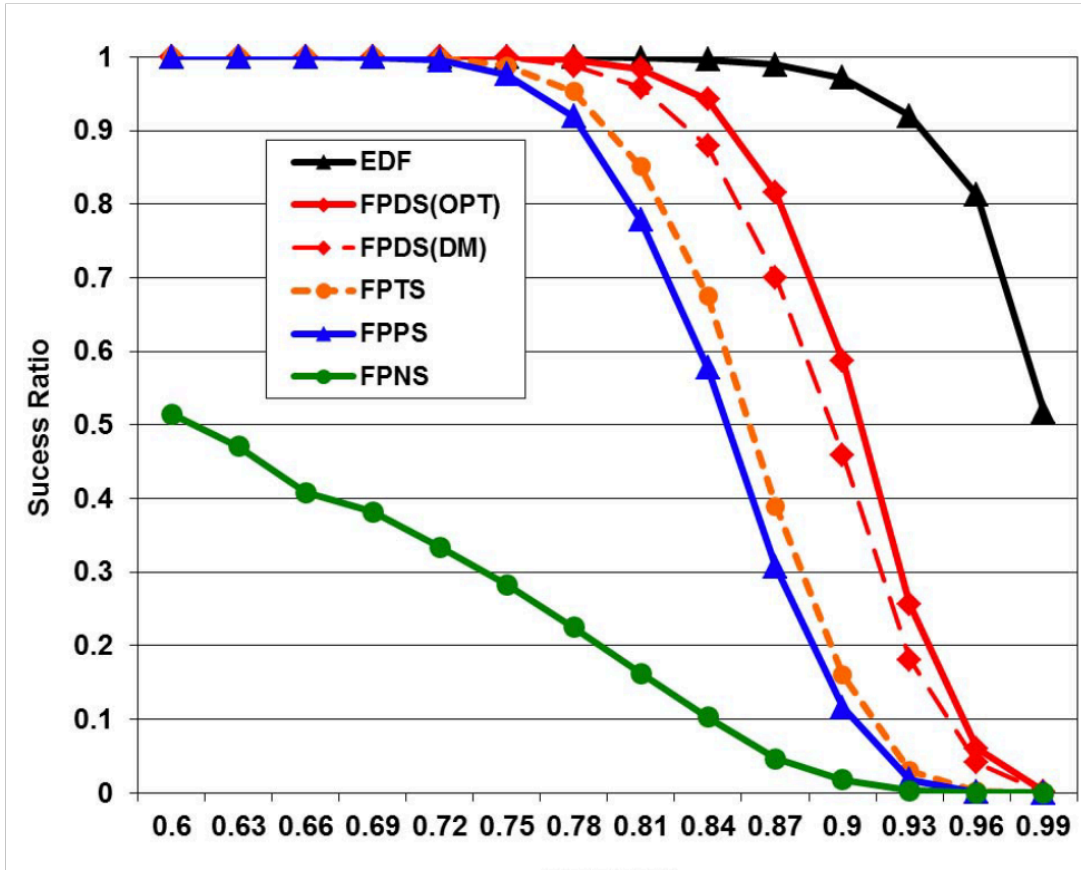
$$F(i,g) = max\left(C_i - S_{i,g}, 1\right)$$

- Taking the maximum over all the jobs of task t_i in the priority level-i active period determines the minimum final non-preemptive region for task t_i.

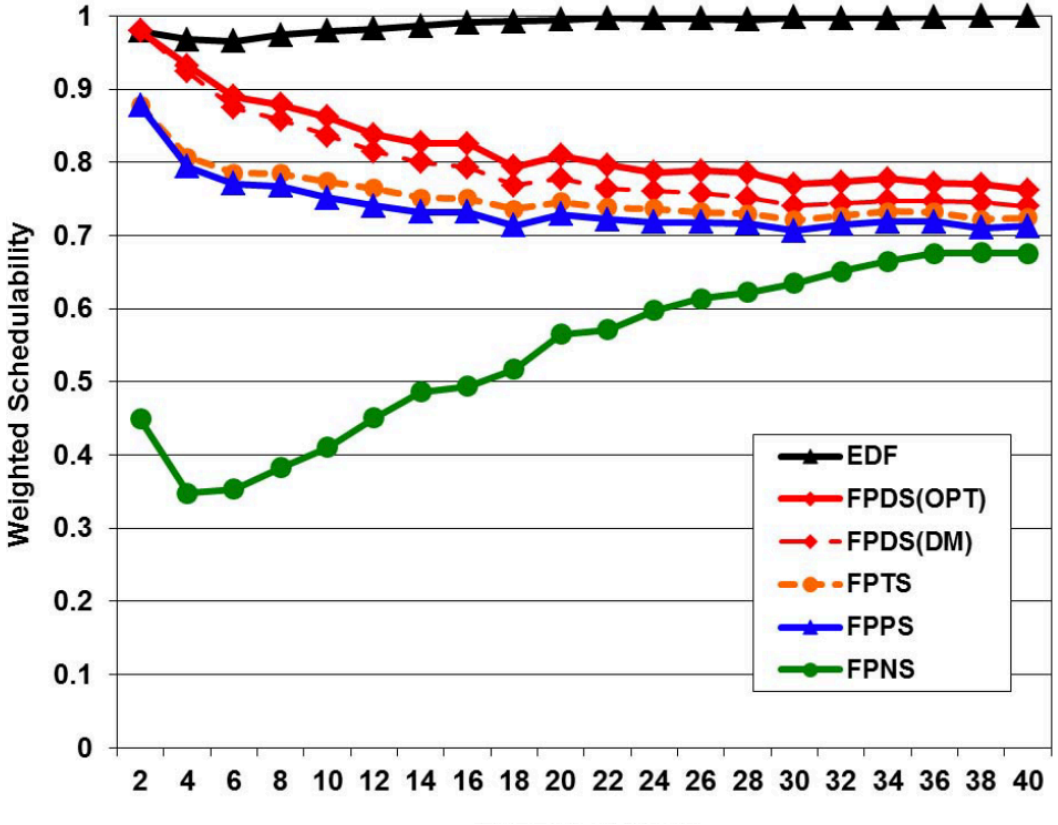
$$F(i) = \max_{g=0,1,2..G_i-1} \{F(i,g)\}$$

Experimental Evaluations





Experimental Evaluations



Weigthed schedulability measure

$$Z_y(p) = \sum_{\forall t} \frac{S_y(t).U(t)}{U(t)}$$

Back up

$$A_i^{m+1} = B_i + \sum_{\forall j \in hep(i)} \left\lceil \frac{A_i^m}{T_j} \right\rceil C_j$$

Worst case length of priority level-i active period

$$B_i = \max\left(B_i^{RES}, B_i^{FNR}\right)$$

$$w_{i,g}^{m+1} = (B_i + (g+1)C_i - F_i) + \sum_{\forall i \in hp(i)} \left(\left\lfloor \frac{w_{i,g}^m}{T_j} \right\rfloor + 1 \right)$$

Start of priority level-i active period

$$R_i = \max\left(W_{i,g}^{NP} + F_i - gT_i\right) \qquad (\forall\, g = 0,1,2 \ldots G_i - 1)$$