Optimal Fixed Priority Scheduling with Deferred Pre-emption

Rob Davis

Real-Time Systems Research Group,
Department of Computer Science,
University of York, York, UK.

rob.davis@york.ac.uk

Marko Bertogna

Algorithmic Research Group,
Department of Mathematics,
University of Modena, Italy

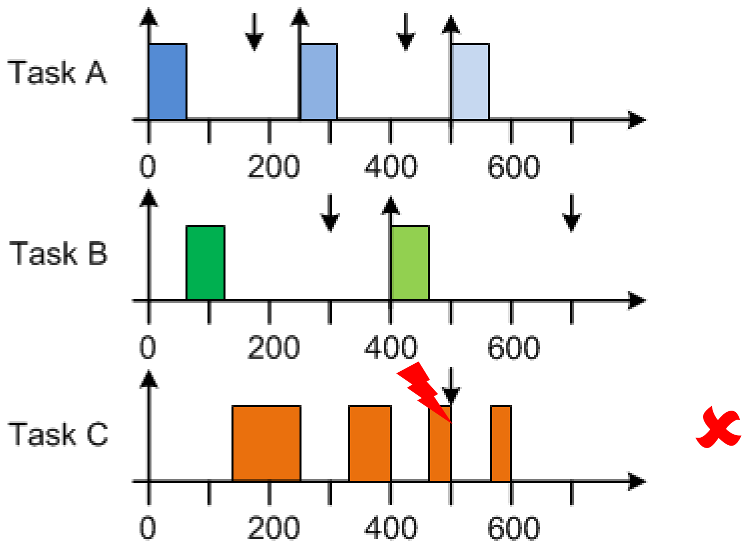
marko.bertogna@unimore.it

Types of Fixed Priority Scheduling

- Fixed Priority Scheduling
  - Tasks have unique priorities
  - At task release and completion, the highest priority ready task is chosen to execute
- Fixed Priority Pre-emptive Scheduling (FPPS)
  - Tasks execute at their initial priorities
  - The executing task can be pre-empted at any time when a higher priority task is released
- Fixed Priority Non-pre-emptive Scheduling (FPNS)
  - Once a task starts executing it is effectively given the highest priority and cannot be pre-empted
- Fixed Priority Scheduling with Deferred Pre-emption (FPDS)
  - Each task has a **final non-pre-emptive region** of execution Once it enters this region it is effectively given the highest priority and cannot be pre-empted

Comparison of FPPS, FPNS, FPDS

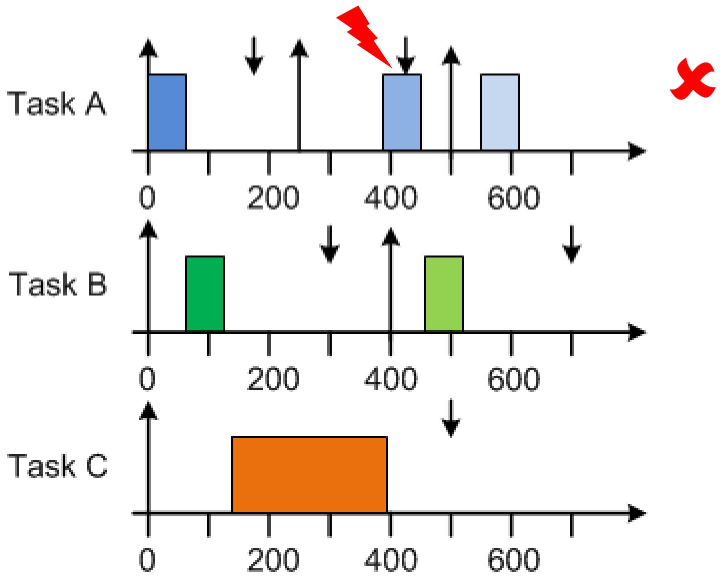
- Fixed Priority Pre-emptive Scheduling (FPPS)



- Minimal blocking of higher priority tasks
- Many pre-emptions
- Long response time for low priority task

Comparison of FPPS, FPNS, FPDS

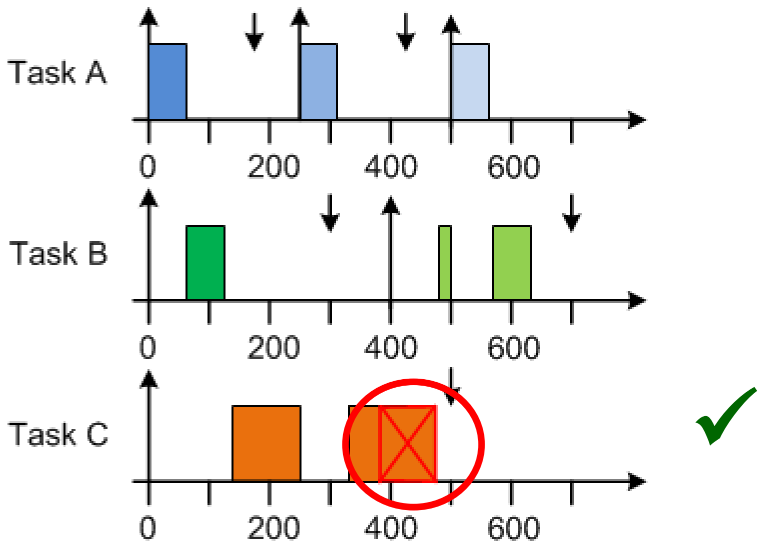
- Fixed Priority Non-pre-emptive Scheduling (FPNS)



- Maximal blocking of higher priority tasks
- No pre-emptions
- Short response time for low priority task

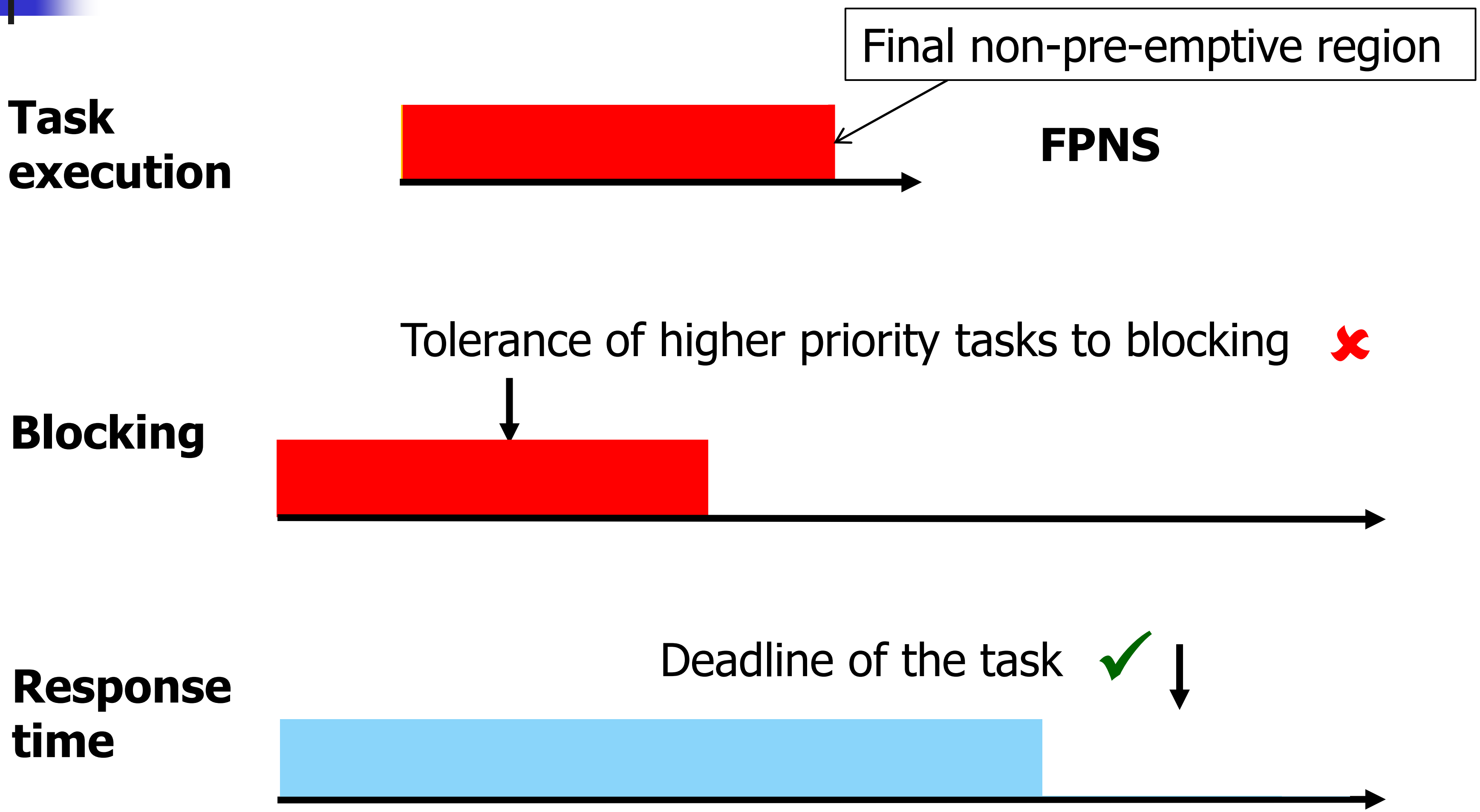
Comparison of FPPS, FPNS, FPDS

- Fixed Priority Scheduling with Deferred Pre-emption (FPDS)



- Superset of FPPS and FPNS
- Trade off between blocking effect on higher priority tasks and the response time of the task itself
- Fewer pre-emptions than FPPS
- Less blocking than FPNS

Blocking v. Response Time trade-off



System model

- Single processor
  - Fixed Priority Scheduling with Deferred Pre-emption (FPDS)

- Sporadic task model
  - Static set of n tasks. Each task τ_i has a unique priority i
    - C_i – Execution time (bound)
    - D_i – Relative deadline
    - T_i – Minimum inter-arrival time or period
    - F_i – Length of final non-pre-emptive region
  - Compute R_i worst-case response time to check if each task is schedulable

- FPDS subsumes FPPS and FPNS
  - F_i =1 equivalent to FPPS
  - $F_i = C_i$ equivalent to FPNS

Schedulability test for FPDS

- Worst-case response time for task τ_i occurs in the longest priority level-i active period starting at a Δ-critical instant

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

- Blocking: $B_i = \max_{\forall l \in lp(i)} (F_l - 1)$
- Number of jobs of task τ_i in the active period: $G_i = \left\lceil \frac{A_i}{T_i} \right\rceil$
- Start time of final non-pre-emptive region:

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

- Response time:

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

Unschedulable if

$$w_{i,g}^{m+1} + F_i - gT_i > D_i$$

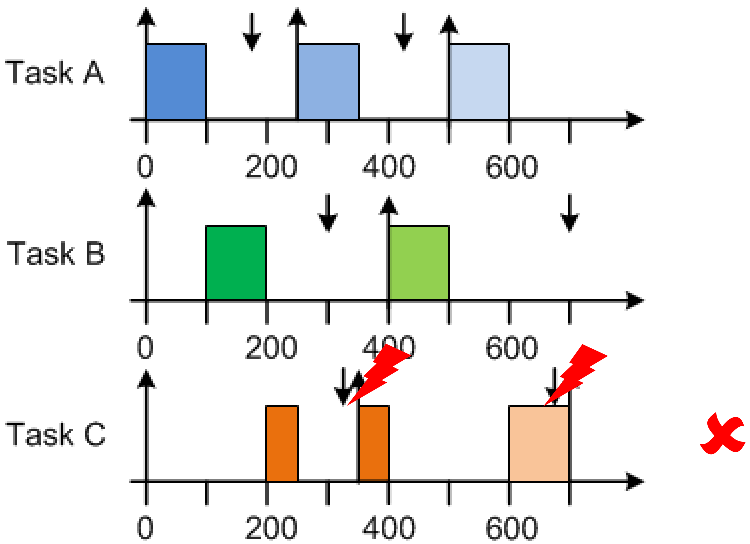
Schedulable if

$$R_i \le D_i$$

Example

Task	Execution Time	Deadline	Period
A	100	175	250
B	100	300	400
C	100	325	350

FPPS



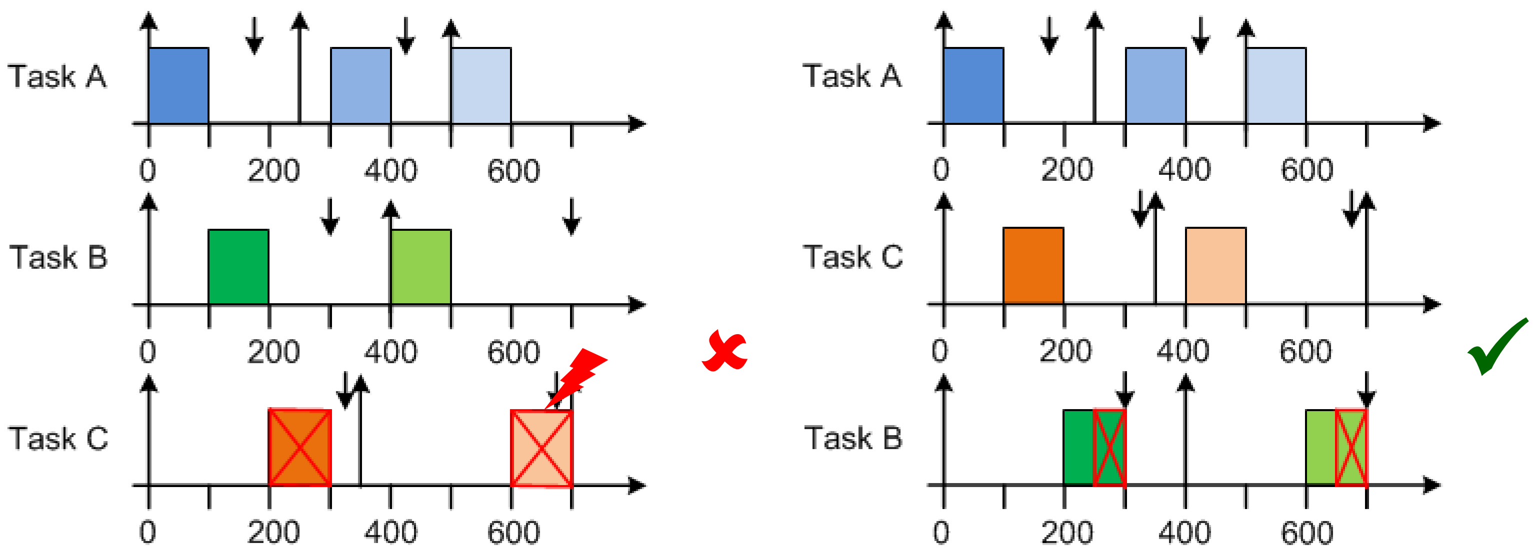
FPNS
Trivially not schedulable
100 + 100 > 175

For FPPS deadline monotonic is the optimal priority assignment

Example

Task	Execution Time	Deadline	Period
A	100	175	250
B	100	300	400
C	100	325	350

FPDS



- Shows:
  - FPDS strictly dominates both FPPS and FPNS (not equivalent)
  - Deadline Monotonic is not an optimal priority assignment for FPDS
- Use Audsley's Optimal Priority Assignment algorithm when FNR lengths are known

Optimal FPDS

- Problem #1: Final Non-pre-emptive Region length Problem (*FNR Problem*)
  - For a taskset complying with the task model with some known priority order X, find a value for the length F_i of the FNR of each task such that the taskset is schedulable under FPDS

An **optimal** FNR length assignment algorithm can schedule any system for which there exists a schedulable FNR length assignment

Optimal FPDS

- Solution to Problem #1: Final Non-pre-emptive Region length Problem (*FNR Problem*)
  - The minimum FNR length F_i such that task τ_i is schedulable at priority i is a monotonically non-decreasing function of the blocking factor B_i due to tasks at lower priorities
  - The blocking factor at higher priorities is a monotonically non-decreasing function of F_i

FNR Algorithm

```
for each priority level i, lowest first {
     determine the smallest value for the
     final non-pre-emptive region length such
     that the task at priority i is schedulable.
     Set the length of the final non-pre-emptive
     region to that value
}
```

Minimises both the final non-pre-emptive region length and the blocking factor at every priority level

Optimal FPDS

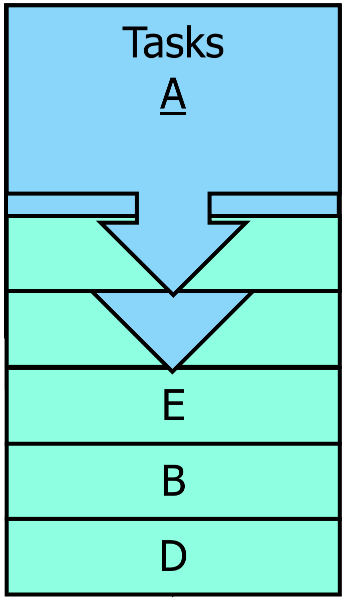
- Problem #2: Final Non-pre-emptive Region length and Priority Assignment Problem
(*FNR-PA Problem*)
  - For a taskset complying with the task model, find both (i) a priority assignment, and (ii) a value for the length of the final non-pre-emptive region of each task that makes the taskset schedulable under FPDS.

An **optimal** FNR length and priority assignment algorithm can schedule any system for which there exists a schedulable priority and FNR length assignment

FNR-PA Algorithm

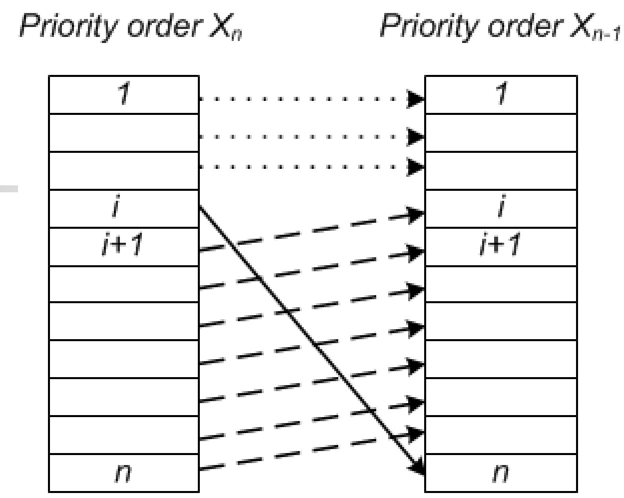
- Solution to Problem #2: Final Non-pre-emptive Region length and Priority Assignment Problem (*FNR-PA Problem*)

```
for each priority level i, lowest first {
    for each unassigned task τ {
        determine minimum final non-pre-emptable region length
        (if any) that makes the task schedulable at priority i
        assuming that all unassigned tasks have higher priorities
    }
    if no tasks are schedulable at priority i {
        return unschedulable
    }
    else {
        assign the schedulable task with the shortest final non-
        pre-emptive region at priority i to priority i
    }
}
return schedulable
```



Complexity $n(n+1)/2$ x determining task schedulability and minimum FNR length

Proof of Optimality



- Assume some priority order *X* exists that is schedulable with some set of FNR lengths

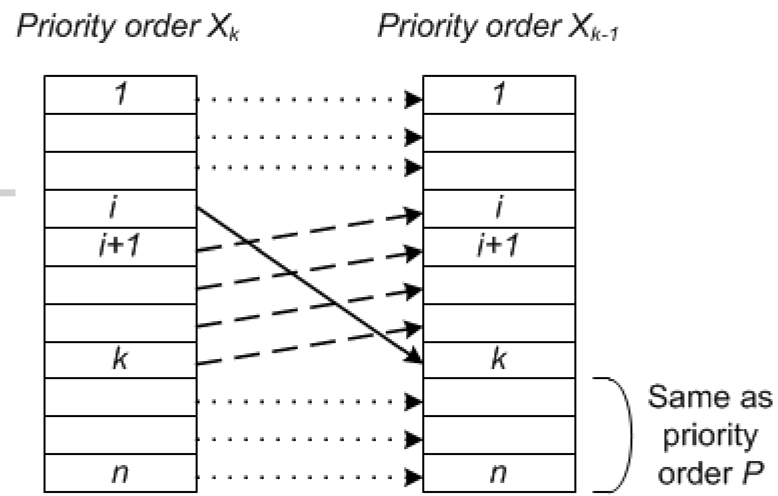
  Transform *X* into the priority order *P* constructed, along with a set of FNR lengths, by the Optimal FNR-PA Algorithm without loss of schedulability

  Do this in *n* steps

- First step
  - Select the task in X_n that is at priority *n* in *P*
  - Shift the task (from priority *i*) to priority *n*
  - Set the FNR length for task τ_n in X_{n-1} to the smallest possible value such that it is schedulable (FNR algorithm).
    - This is the same as the value determined by the optimal FNR-PA algorithm (same set of hp tasks)
    - No greater than the value for the task at priority *n* in X_n otherwise the optimal FNR-PA algorithm would have chosen that task instead

- Show X_{n-1} is schedulable
  - Tasks at higher priority than *i* in X_n – no increase in blocking
  - Tasks at priorities *i*+1 to *n* in X_n – shifted up in priority hence remain schedulable
  - Task τ_n must be schedulable at the lowest priority in X_{n-1} – as it was chosen by the FNR-PA algorithm (and there must be such a task e.g. task at priority *n* in X_n)

Proof of Optimality



- Intermediate steps
  - Select the task in X_k that is at priority k in P
  - Shift the task (from priority i) to priority k - note i is never lower than k due to the lowest priority tasks being the same in both orderings
  - Set the FNR length for task τ_k in X_{k-1} to the smallest possible value such that it is schedulable (FNR algorithm).
    - This value is the same as the value determined by the optimal FNR-PA algorithm (same set of hp tasks, and same set of lp tasks with the same FNR lengths)
    - This value is no greater than that for the task at priority k in X_k otherwise the Optimal FNR-PA algorithm would have chosen that task instead
- Show X_{k-1} is schedulable
  - Tasks at higher priority than i – no increase in blocking
  - Tasks at priorities i+1 to k-1 – are shifted up in priority hence remain schedulable
  - Task τ_k at priority k in X_{n-1} – was chosen by the FNR-PA algorithm, so must be schedulable
  - Task at lower priorities – have the same set of hp tasks and unchanged FNR lengths so remain schedulable

Optimal FPDS

- FNR-PA algorithm
  - **Optimality:** Determines a schedulable priority ordering and set of final non-pre-emptive region lengths whenever such a combination exists.

    Proof – see the paper

    Provides Optimal Fixed Priority Scheduling with Deferred Pre-emption

  - Has the side-effect of **minimising blocking** due to FNRs at **every priority level**

  - Also works when tasks share resources according to Stack Resource Policy (provided there is proper nesting) or have other non-pre-emptive regions – may constrain the permitted length of FNRs

FNR length calculation

- Algorithms presented rely on being able to find the minimum final non-pre-emptive region length such that a task is schedule (if it is schedulable for any FNR)
- Simple option is Binary Search
  - Requires multiple single task schedulability tests
- Analytical method given in the paper
  - Pseudo-polynomial in complexity - same as a single task schedulability test

- FNR-PA algorithm using the analytical method
  - Needs the equivalent of $n(n+1)/2$ task schedulability tests to determine an optimal priority <u>and</u> final non-pre-emptive region length assignment
  - Compares to a search space of $n!\prod_{\forall i} C_i$

Experimental Evaluation

- Performance comparison of
  - **FPDS (OPT)** – Optimal FPDS
  - **FPDS (DM)** – assumes Deadline Monotonic Priority Order (not optimal)
  - **FPPS** – with DMPO (which is optimal for FPPS)
  - **FPNS** – with optimal priority assignment using Audsley's algorithm
  - **FPTS** – Fixed Priority Pre-emption Threshold scheduling with optimal threshold assignment and DMPO

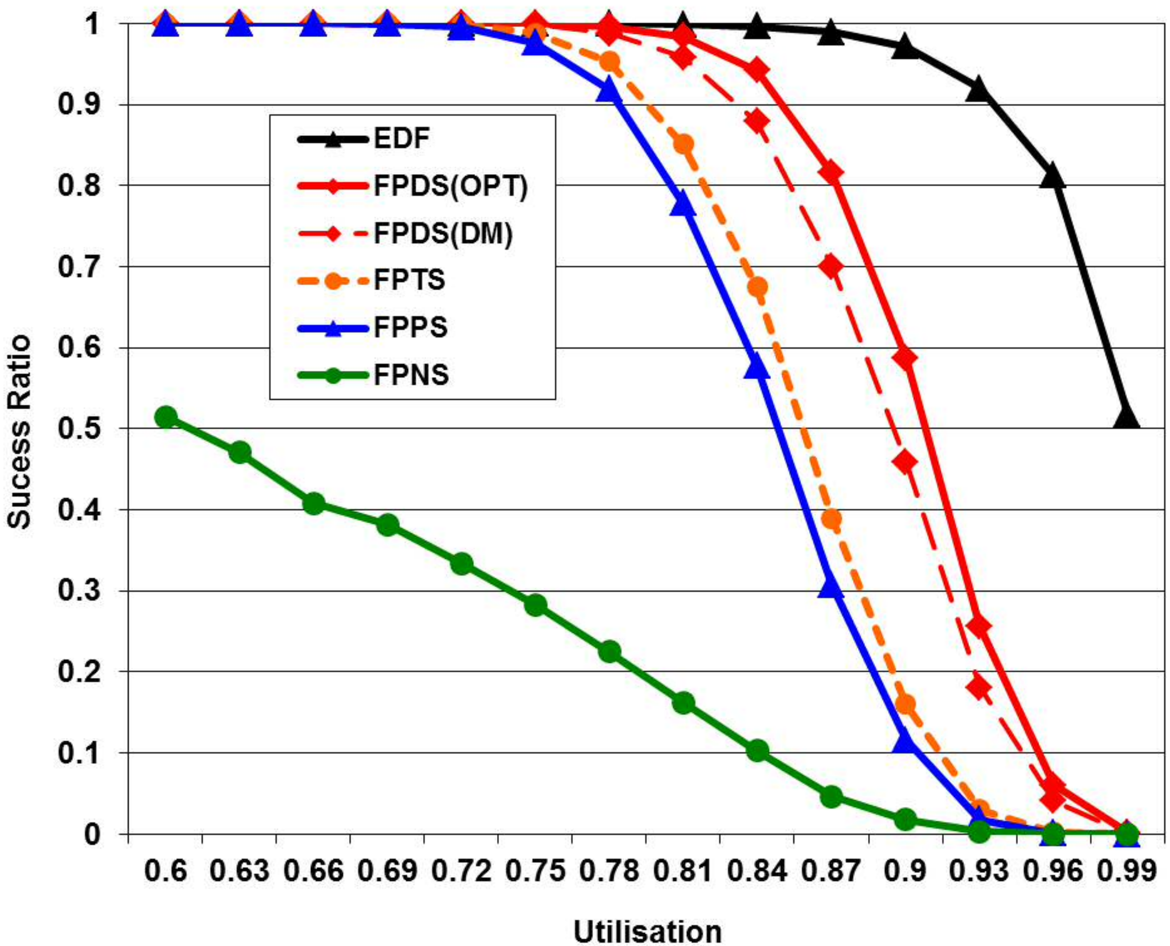
  and

  - **EDF** (pre-emptive) as a benchmark as this is the optimal single processor scheduling algorithm

Experimental Evaluation

- Parameter generation for tasks
  - Utilisation values generated via UUnifast
  - Task periods – log-uniform distribution with a ratio of 10^r between max and min periods (default $r = 1$)
  - Execution times based on the utilisation and period values selected
  - Independent tasks – so no constraints on FNR lengths
  - Deadlines were either *implicit* or *constrained* and chosen according to a uniform distribution in the range $[C_i + \alpha(T_i - C_i), T_i]$ (default $\alpha = 0.5$)
- Taskset generation
  - Default taskset cardinality was $n = 10$
  - Total utilisation values from 0.03 to 0.99
  - 5000 tasksets generated for each utilisation value

Success ratio



Constrained deadlines

Taskset cardinality n = 10

Period range 10^r (r = 1)

Deadlines in range

$$[C_i + \alpha(T_i - C_i), T_i]$$

with α = 0.5

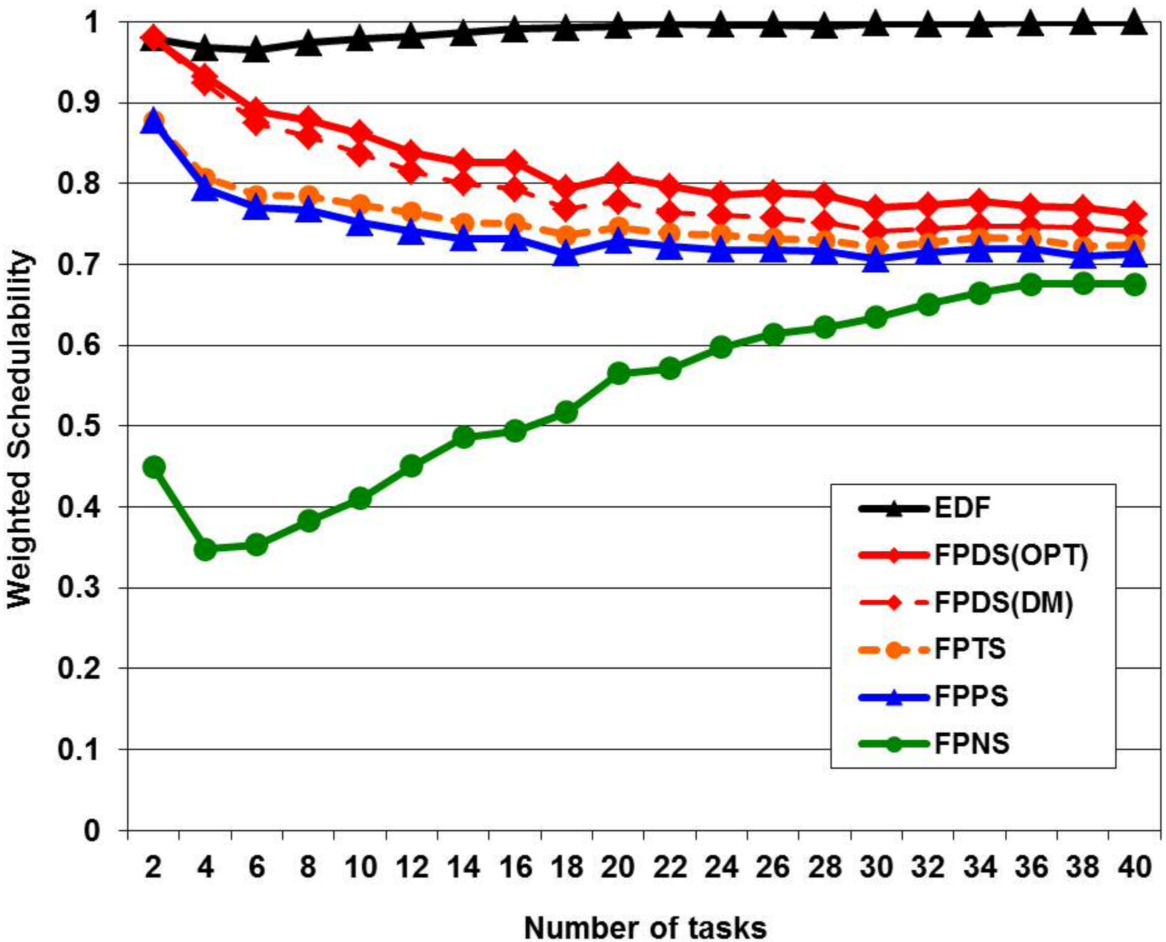
Other comparisons

- Weighted schedulability
  - Enables overall comparisons when varying a specific parameter (not just utilisation)
  - Combines results from all of a set of equally spaced utilisation levels
  - Weighted schedulability:

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

  - Collapses all data on a success ratio plot for a given algorithm, into a single point on a weighted schedulability graph

Weighted schedulability: Varying taskset cardinality



Constrained deadlines

Variable taskset cardinality

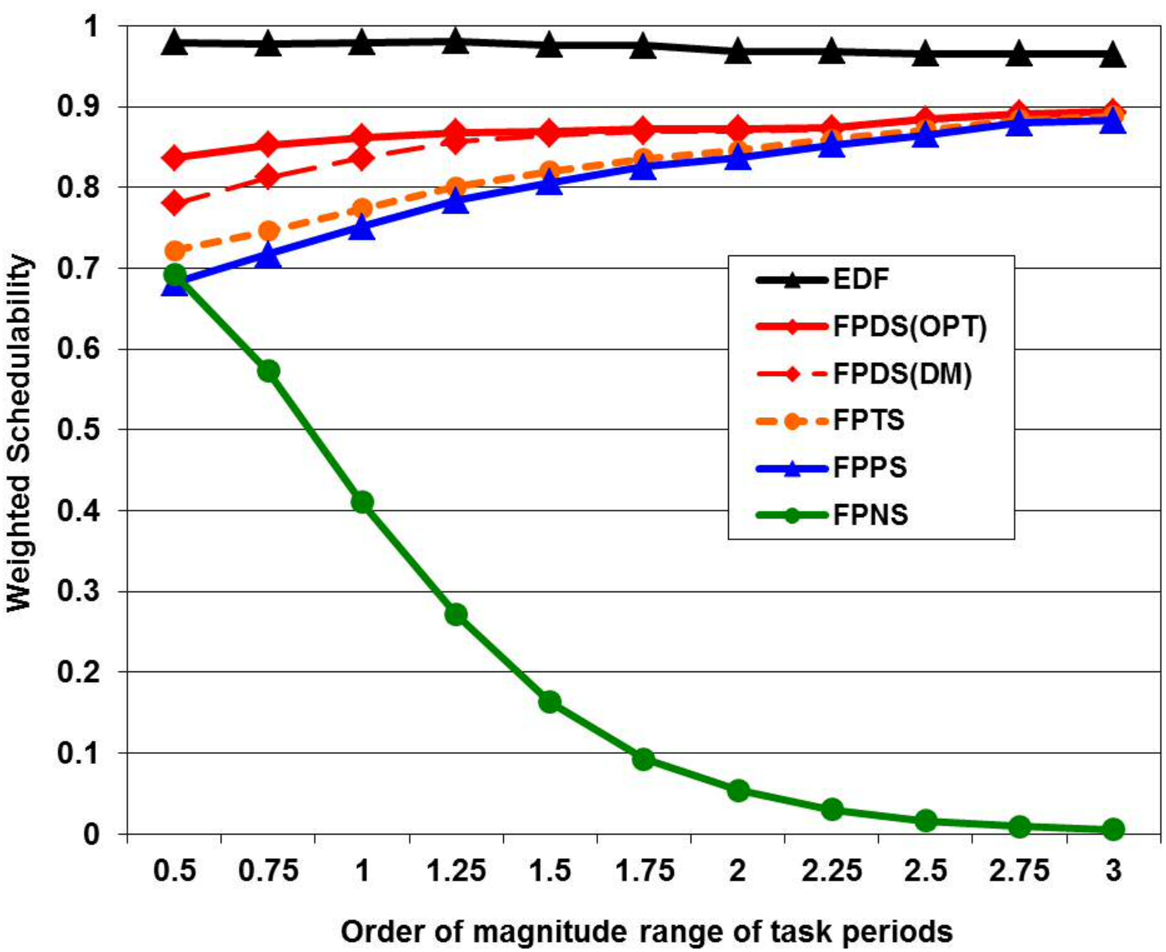
Period range 10^r (r = 1)

Deadlines in range

$$[C_i + \alpha(T_i - C_i), T_i]$$

with α = 0.5

Weighted schedulability: Varying range of task periods



Constrained deadlines

Taskset cardinality n = 10

Variable range of periods

Deadlines in the range

$$[C_i + \alpha(T_i - C_i), T_i]$$

with α = 0.5

Summary and conclusions

- Main contribution:
  - **Optimal Fixed Priority Scheduling with Deferred Pre-emption**
  - Can find the priorities and final non-pre-emptive region lengths to obtain a schedulable system whenever such parameters exist

Optimal FNR-PA Algorithm

```
for each priority level i, lowest first {
    for each unassigned task τ {
        determine minimum final non-pre-emptable region length
        (if any) that makes the task schedulable at priority i
        assuming that all unassigned tasks have higher priorities
    }
    if no tasks are schedulable at priority i {
        return unschedulable
    }
    else {
        assign the schedulable task with the shortest final non-
        pre-emptive region at priority i to priority i
    }
}
return schedulable
```

Minimises blocking at EVERY priority level

Compatible with SRP for resource locking

Complexity O(n^2) search space

$$n!\prod_{\forall i} C_i$$

Applications and Future work

- Applications
  - Automotive systems: tasks composed of 50-300 sequential functions each of which can be a non-pre-emptive region
  - FNR-PA algorithm can be used to determine optimal priority assignments and final non-pre-emptive region lengths, subject to constraints (granularity due to sequential functions)
- Future work
  - Integration with:
    - Pre-emption costs, and Cache Related Pre-emption Delays
    - Requirements for robustness – must not end up with systems that are only just schedulable

Questions?

Optimal Fixed Priority Scheduling with Deferred Pre-emption

Rob Davis and Marko Bertogna

RTSS 2012
San Juan, Puerto Rico

End