Online Resource Minimization

Nicole Megow
with Lin Chen and Kevin Schewior

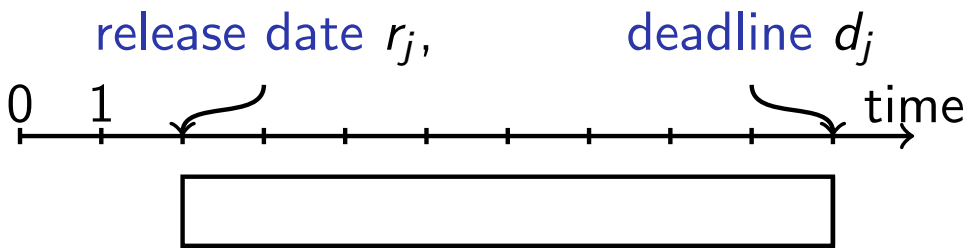
EURANDOM "Scheduling under Uncertainty"
Eindhoven, June 2015

The problem

- **Input**: set of jobs $J = \{1, \ldots, n\}$, where each job j is defined by

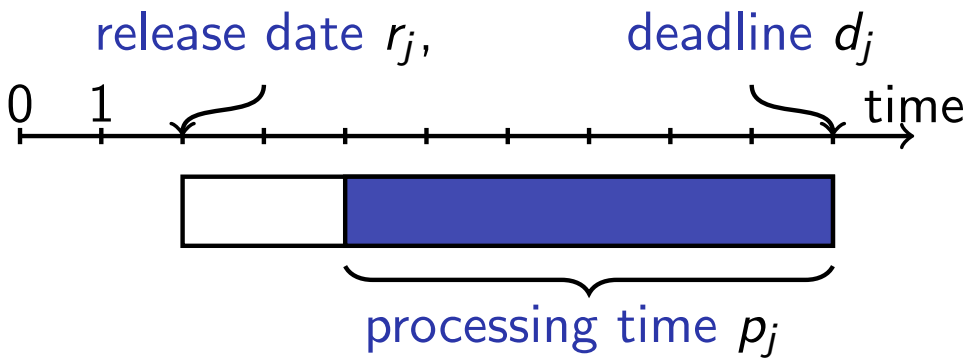
The problem

- **Input**: set of jobs $J = \{1, \ldots, n\}$, where each job j is defined by



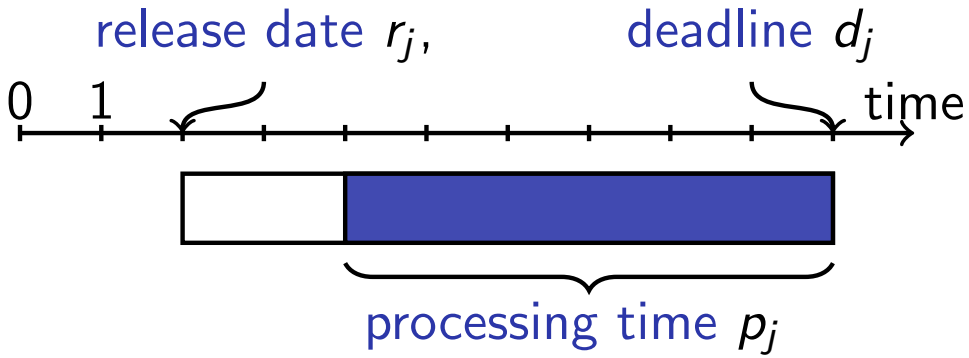
The problem

- **Input**: set of jobs $J = \{1, \ldots, n\}$, where each job j is defined by



The problem

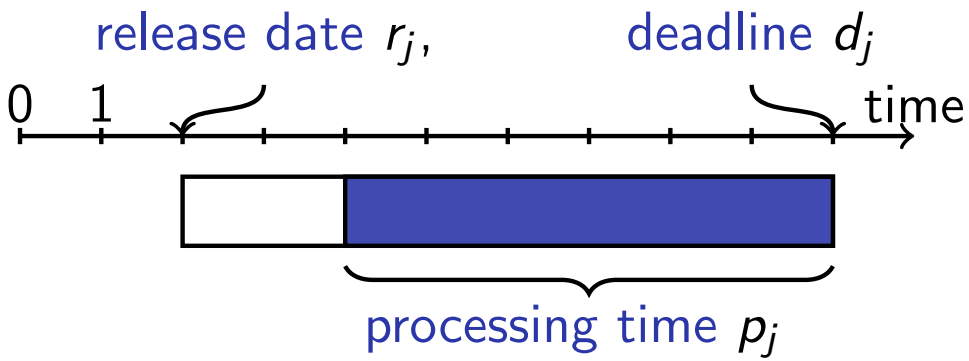
- **Input**: set of jobs $J = \{1, \ldots, n\}$, where each job j is defined by



- **Task**: Find a feasible schedule on a min. number of machines.

The problem

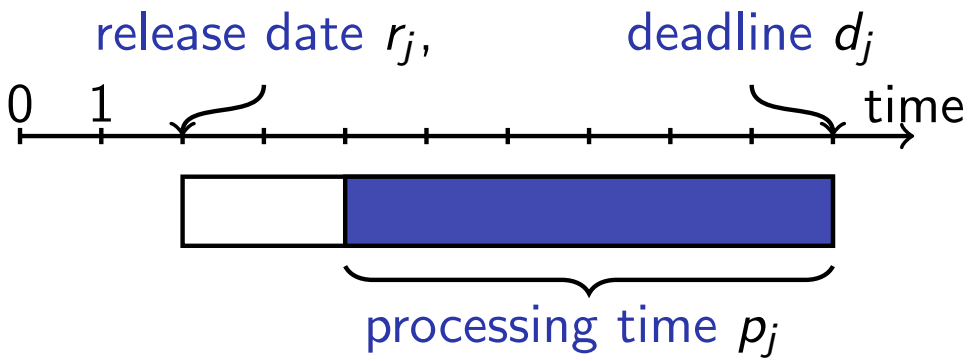
- **Input**: set of jobs $J = \{1, \ldots, n\}$, where each job j is defined by



- **Task**: Find a feasible schedule on a min. number of machines.
  - Each job j is processing for p_j time units within $[r_j, d_j]$.

The problem

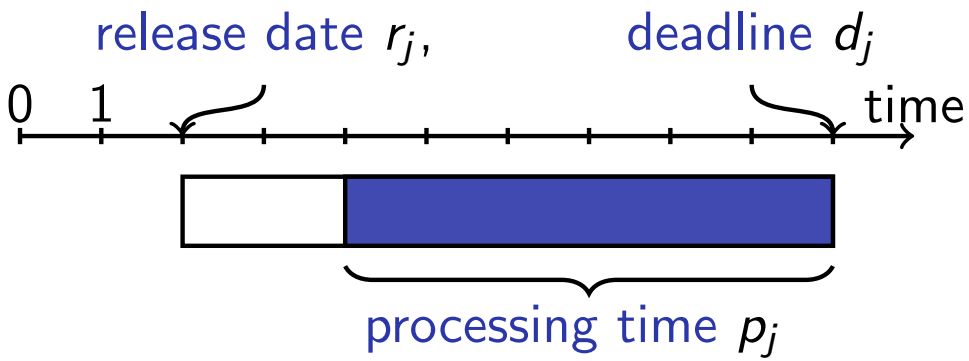
- **Input**: set of jobs $J = \{1, \ldots, n\}$, where each job j is defined by



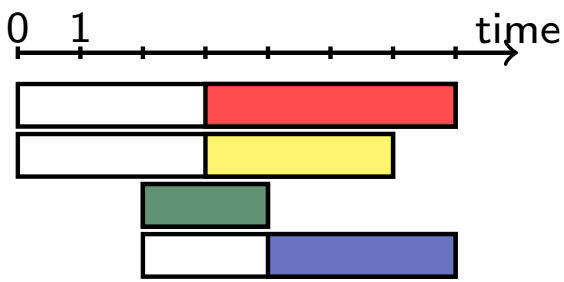
- **Task**: Find a feasible schedule on a min. number of machines.
  - Each job j is processing for p_j time units within $[r_j, d_j]$.
  - At any time, any job runs on at most one machine.

The problem

- **Input**: set of jobs $J = \{1, \ldots, n\}$, where each job j is defined by

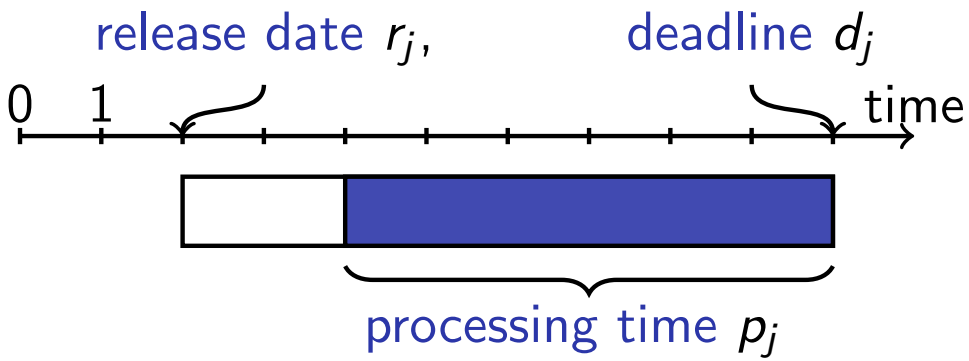


- **Task**: Find a feasible schedule on a min. number of machines.
  - Each job j is processing for p_j time units within $[r_j, d_j]$.
  - At any time, any job runs on at most one machine.

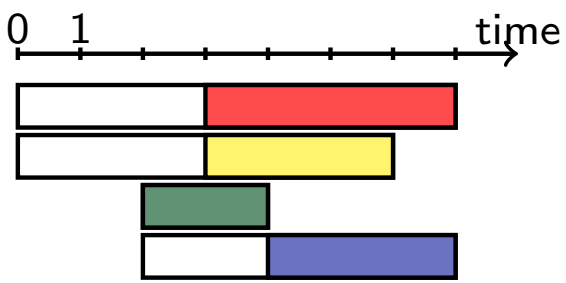


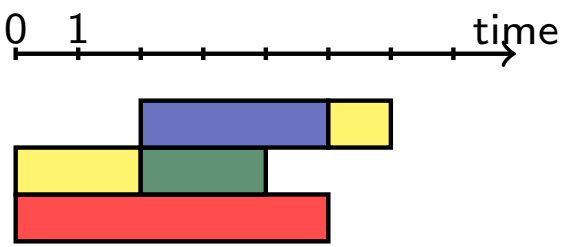
The problem

- **Input**: set of jobs $J = \{1, \ldots, n\}$, where each job j is defined by



- **Task**: Find a feasible schedule on a min. number of machines.
  - Each job j is processing for p_j time units within $[r_j, d_j]$.
  - At any time, any job runs on at most one machine.





The problem: offline

Offline problem: All jobs are known in advance.

The problem: offline

Offline problem: All jobs are known in advance.

- Preemptive problem:
  - Optimally solvable in polynomial time (LP or reduction to maximum flow problem) Horn (1974)

The problem: offline

Offline problem: All jobs are known in advance.

- Preemptive problem:
  - Optimally solvable in polynomial time (LP or reduction to maximum flow problem) Horn (1974)
- Non-preemptive problem: Chuzhoy et al. (FOCS 2004)
  - No polynomial-time algorithm unless P=NP.

The problem: offline

Offline problem: All jobs are known in advance.

- Preemptive problem:
  - Optimally solvable in polynomial time (LP or reduction to maximum flow problem) Horn (1974)
- Non-preemptive problem: Chuzhoy et al. (FOCS 2004)
  - No polynomial-time algorithm unless P=NP.
  - $\mathcal{O}(\mathrm{OPT}^2)$-approximation
  - $\mathcal{O}(\sqrt{\frac{\log n}{\log \log n}})$-approximation

The problem: offline

Offline problem: All jobs are known in advance.

- Preemptive problem:
  - Optimally solvable in polynomial time (LP or reduction to maximum flow problem) Horn (1974)
- Non-preemptive problem: Chuzhoy et al. (FOCS 2004)
  - No polynomial-time algorithm unless P=NP.
  - $\mathcal{O}(\mathrm{OPT}^2)$-approximation
  - $\mathcal{O}(\sqrt{\frac{\log n}{\log \log n}})$-approximation
  - Lower bounds?

The problem: online

Online problem: A job becomes known only at its release date.

The problem: online

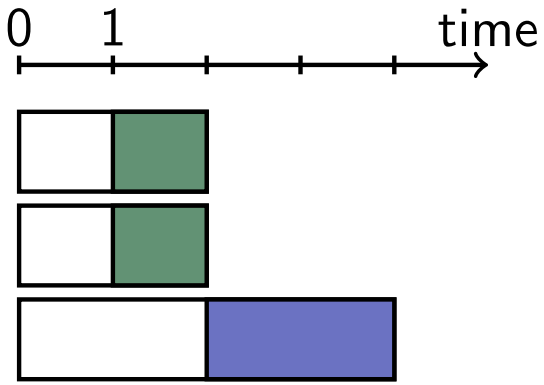
Online problem: A job becomes known only at its release date.

Slack versus processing time?

The problem: online

Online problem: A job becomes known only at its release date.

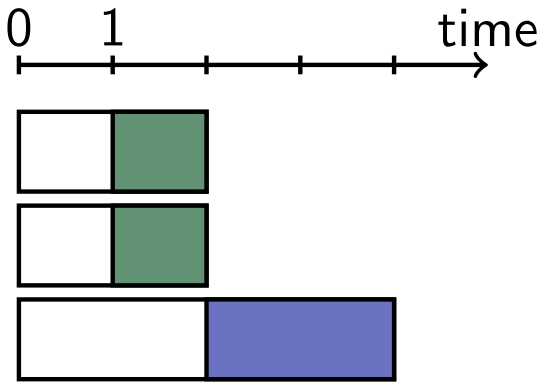
Slack versus processing time?



The problem: online

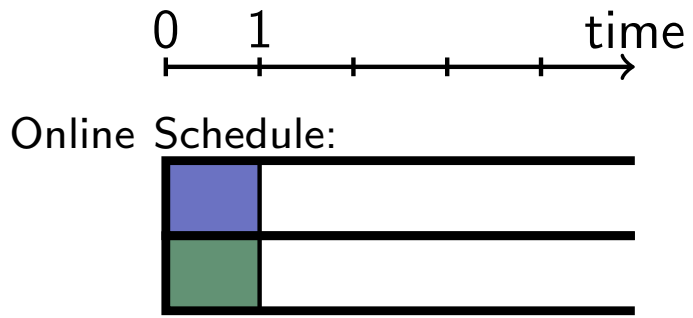
Online problem: A job becomes known only at its release date.

Slack versus processing time?



Case 1:

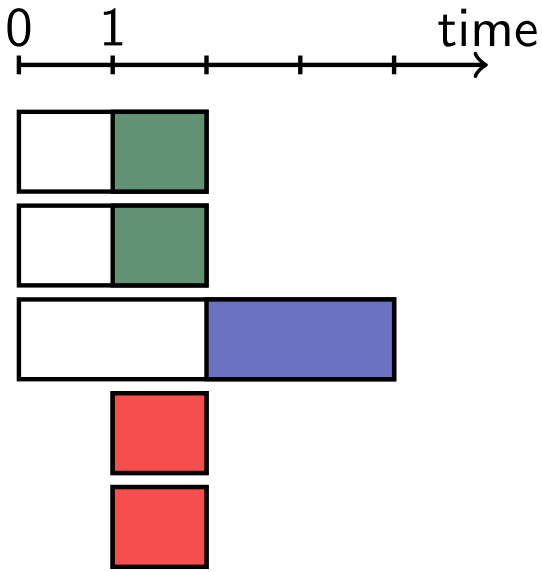
The blue job is scheduled to some extent in $[0, 1]$.



The problem: online

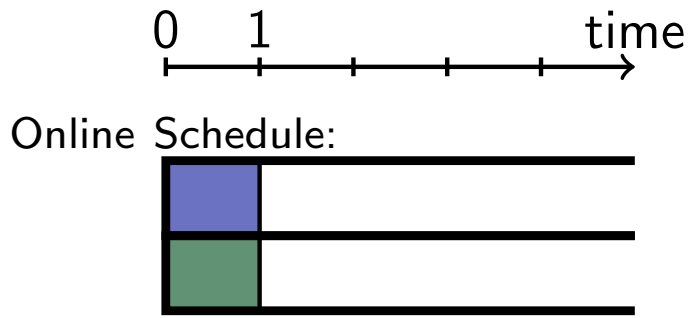
Online problem: A job becomes known only at its release date.

Slack versus processing time?



Case 1:

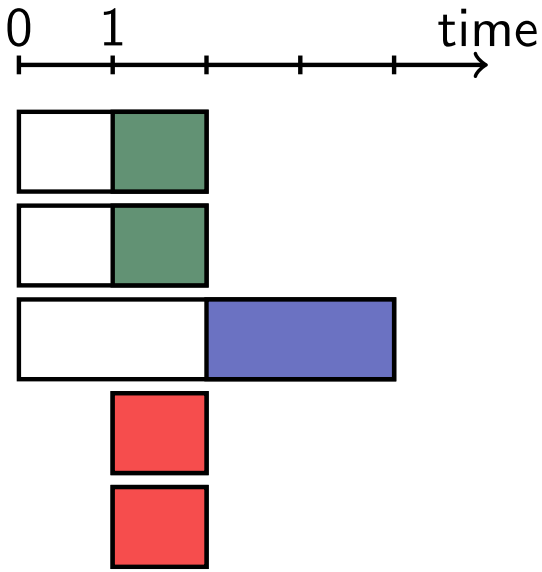
The blue job is scheduled to some extent in $[0, 1]$.



The problem: online

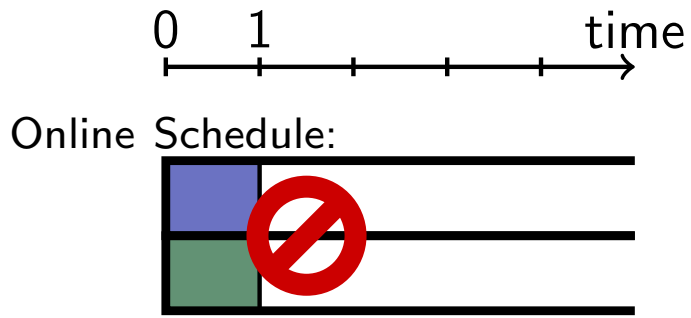
Online problem: A job becomes known only at its release date.

Slack versus processing time?



Case 1:

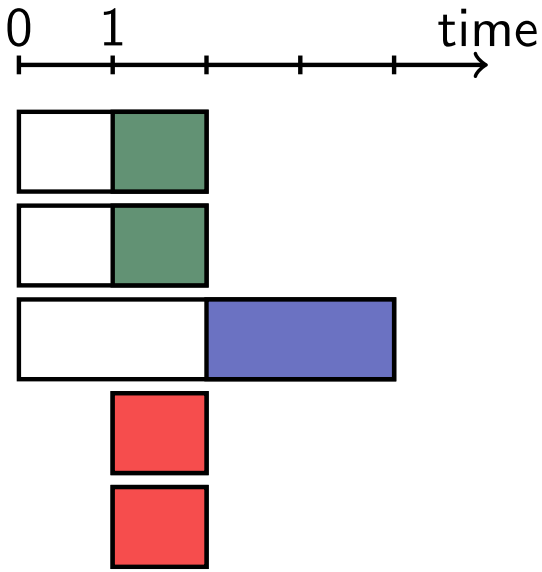
The blue job is scheduled to some extent in $[0, 1]$.



The problem: online

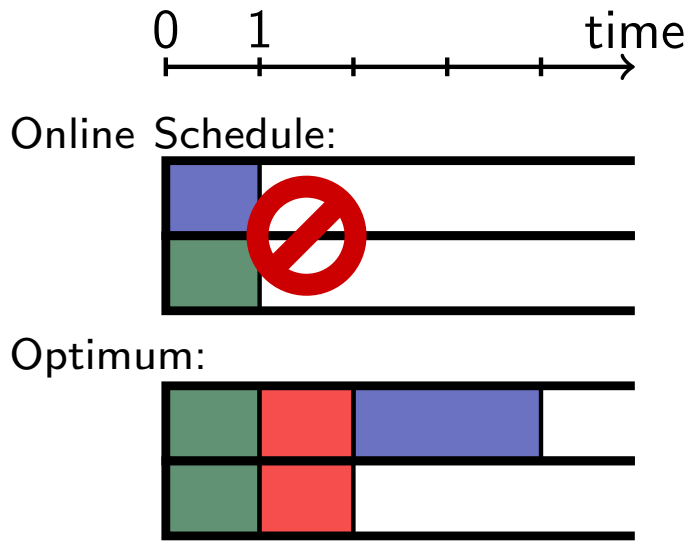
Online problem: A job becomes known only at its release date.

Slack versus processing time?



Case 1:

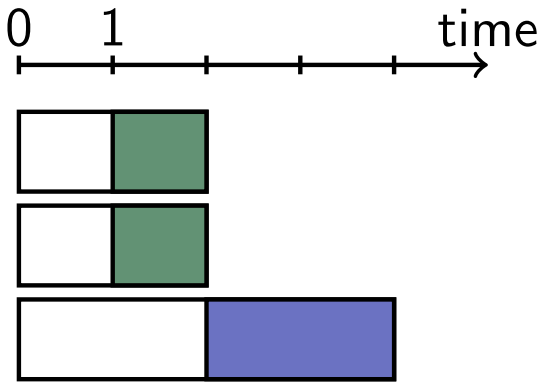
The blue job is scheduled to some extent in $[0, 1]$.



The problem: online

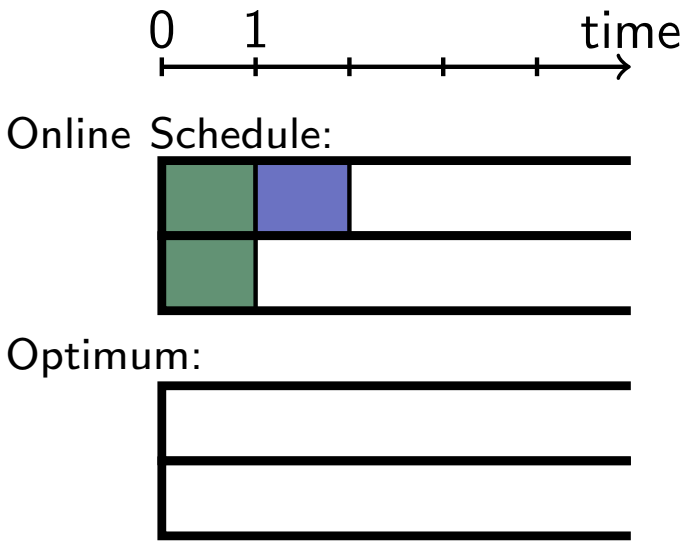
Online problem: A job becomes known only at its release date.

Slack versus processing time?



Case 2:

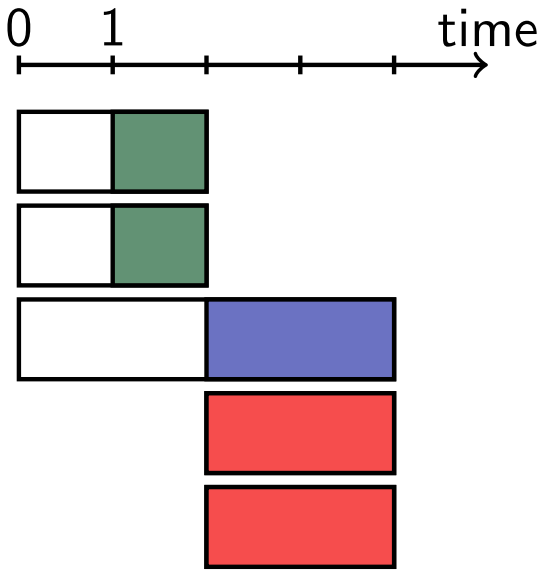
The blue job is not scheduled in $[0, 1]$, i.e., does not finish by 2.



The problem: online

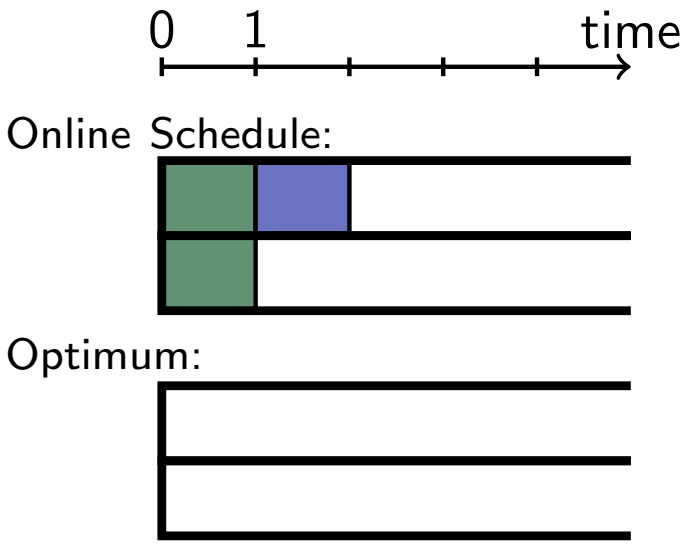
Online problem: A job becomes known only at its release date.

Slack versus processing time?



Case 2:

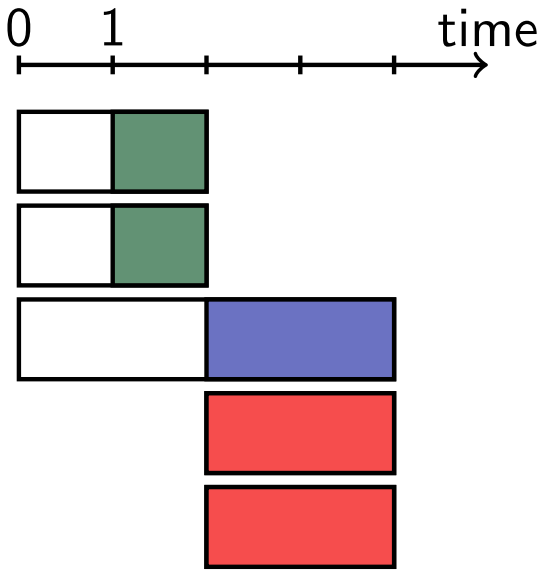
The blue job is not scheduled in $[0, 1]$, i.e., does not finish by 2.



The problem: online

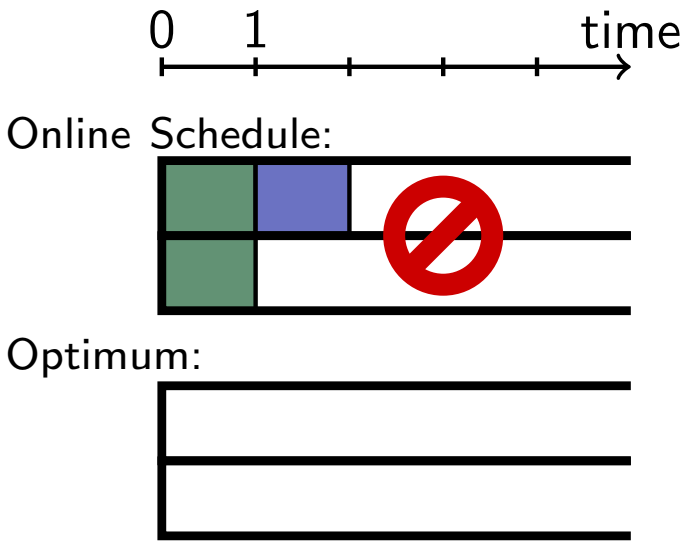
Online problem: A job becomes known only at its release date.

Slack versus processing time?



Case 2:

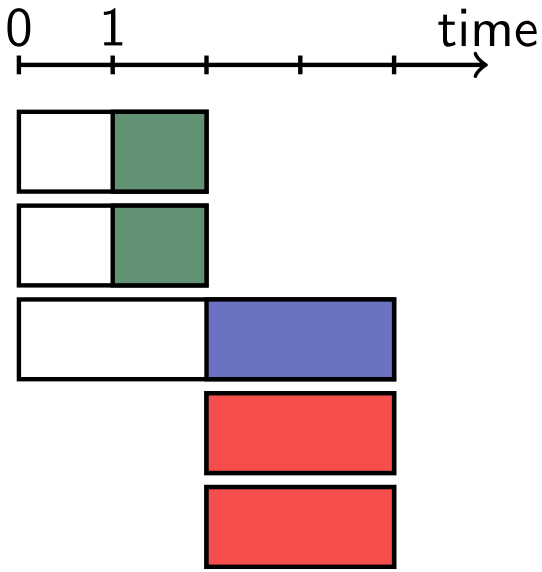
The blue job is not scheduled in $[0, 1]$, i.e., does not finish by 2.



The problem: online

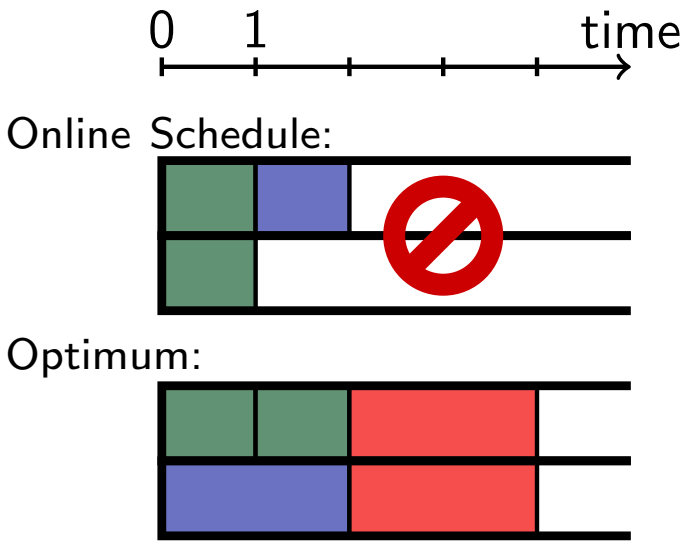
Online problem: A job becomes known only at its release date.

Slack versus processing time?



Case 2:

The blue job is not scheduled in $[0, 1]$, i.e., does not finish by 2.



The problem: online

Bad news: No (preemptive) online algorithm can guarantee to find always a feasible schedule on the minimum number of machines.

Dertousoz and Mok (TSE 1989)

The problem: online

Bad news: No (preemptive) online algorithm can guarantee to find always a feasible schedule on the minimum number of machines.

Dertousoz and Mok (TSE 1989)

Competitive analysis: An online algorithm ALG is c-competitive if

$$\mathrm{ALG}(I) \leq c \cdot m(I),$$

for all instances I with a feasible offline schedule on $m(I)$ machines.

Known results

- Non-preemptive problem is hopeless
  - Any online algorithm may need n machines. Saha (FSTTCS 2013)

Known results

- Non-preemptive problem is hopeless
  - Any online algorithm may need n machines. Saha (FSTTCS 2013)
- Preemptive problem is challenging
  - Best known algorithm (LLF) is $\mathcal{O}(\log p_{\max}/p_{\min})$-competitive.
  - General lower bound is $5/4$. Philips et al. (STOC 1996)

Known results

- Non-preemptive problem is hopeless
  - Any online algorithm may need n machines. Saha (FSTTCS 2013)
- Preemptive problem is challenging
  - Best known algorithm (LLF) is $\mathcal{O}(\log p_{\max}/p_{\min})$-competitive.
  - General lower bound is $5/4$. Philips et al. (STOC 1996)
  - $p_j = 1$: There is an optimal e-competitive online algorithm.
    Bansal, Kimbrel, Pruhs (J. ACM 2007)
    Devanur et al. (arxiv 2014)

Our results

- Preemptive online algorithm with competitive ratio $\mathcal{O}(m^2 \log m)$.

Our results

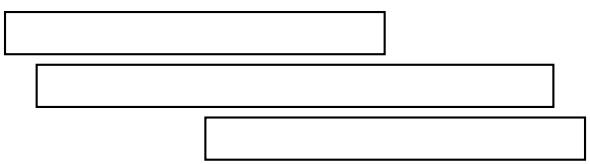
- Preemptive online algorithm with competitive ratio $\mathcal{O}(m^2 \log m)$.
- Two important special classes:
  - Agreeable instances: $O(1)$-competitive algorithm.
    $\longrightarrow$ even without preemption
  - Laminar instances: $O(\log m)$-competitive algorithm.

Our results

- Preemptive online algorithm with competitive ratio $\mathcal{O}(m^2 \log m)$.
- Two important special classes:
  - Agreeable instances: $O(1)$-competitive algorithm.
    $\longrightarrow$ even without preemption
  - Laminar instances: $O(\log m)$-competitive algorithm.

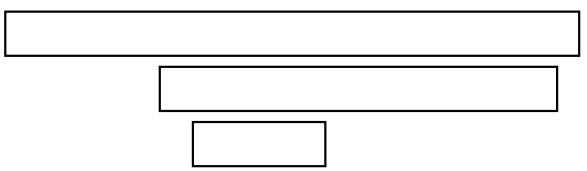
Agreeable instances

for any two jobs j, k:
$r_j < r_k$ implies $d_j \leq d_k$



Laminar instances

for any two intersecting time windows I_j, I_k: either $I_j \subseteq I_k$ or $I_k \subseteq I_j$



An observation

Does it help to know OPT $= m$?

An observation

Does it help to know OPT $= m$?

Theorem

At a loss of a factor 4 in the competitive ratio, we may assume that m is known.

An observation

Does it help to know OPT $= m$?

Theorem

At a loss of a factor 4 in the competitive ratio, we may assume that m is known.

Idea: Guess-and-Double.

(Preemptive) Earliest Deadline First

Algorithm $\text{EDF}_{m'}$: At any time schedule m available jobs with minimum deadline and preempt other jobs if necessary.

(Preemptive) Earliest Deadline First

Algorithm $\text{EDF}_{m'}$: At any time schedule m available jobs with minimum deadline and preempt other jobs if necessary.

Theorem [Philips et al., STOC 1997]

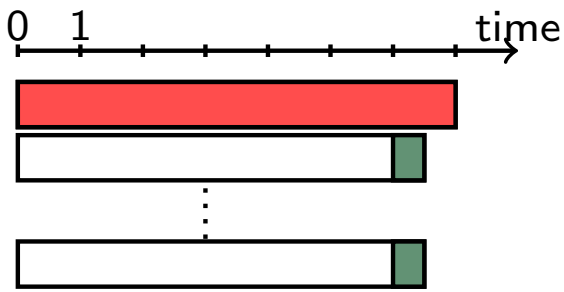
There are instances (for any $\text{OPT} \geq 2$) on which EDF_{n-1} fails.

(Preemptive) Earliest Deadline First

Algorithm $\text{EDF}_{m'}$: At any time schedule m available jobs with minimum deadline and preempt other jobs if necessary.

Theorem [Philips et al., STOC 1997]

There are instances (for any $\text{OPT} \geq 2$) on which EDF_{n-1} fails.

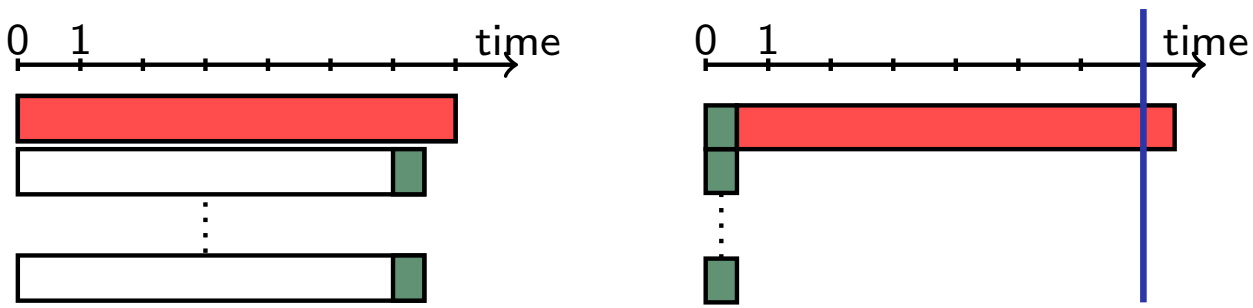


(Preemptive) Earliest Deadline First

Algorithm $\text{EDF}_{m'}$: At any time schedule m available jobs with minimum deadline and preempt other jobs if necessary.

Theorem [Philips et al., STOC 1997]

There are instances (for any $\text{OPT} \geq 2$) on which EDF_{n-1} fails.

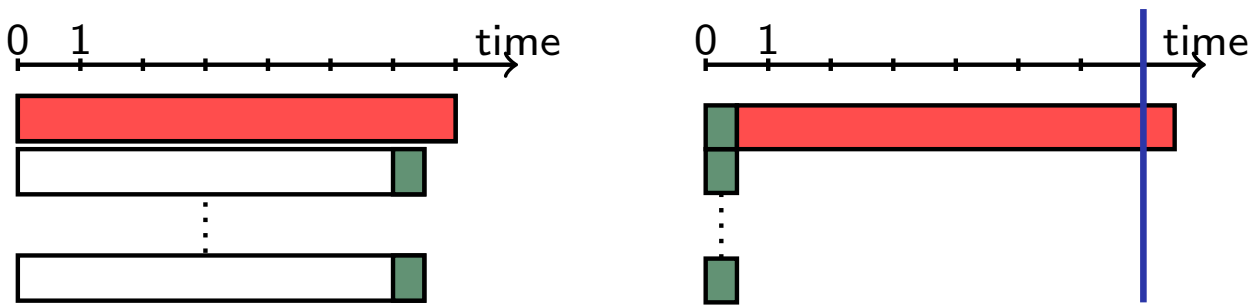


(Preemptive) Earliest Deadline First

Algorithm $\text{EDF}_{m'}$: At any time schedule m available jobs with minimum deadline and preempt other jobs if necessary.

Theorem [Philips et al., STOC 1997]

There are instances (for any OPT ≥ 2) on which EDF_{n-1} fails.



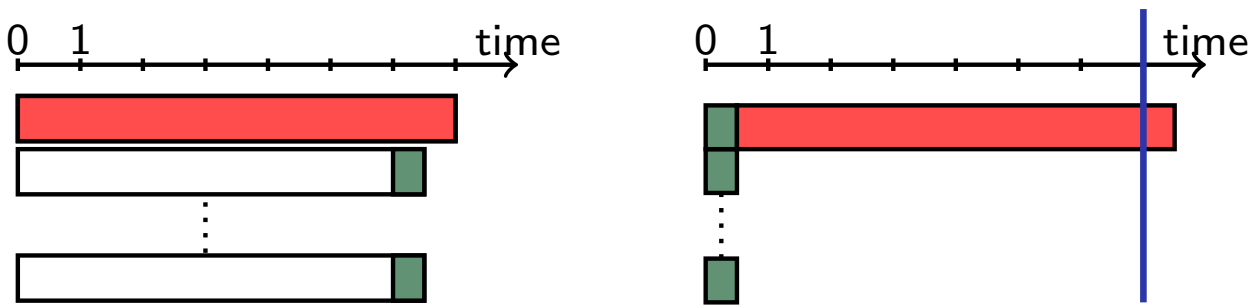
Def. Let $\alpha < 1$. Job j is α-tight if $p_j > \alpha(d_j - r_j)$ and α-loose othw.

(Preemptive) Earliest Deadline First

Algorithm $\text{EDF}_{m'}$: At any time schedule m available jobs with minimum deadline and preempt other jobs if necessary.

Theorem [Philips et al., STOC 1997]

There are instances (for any $\text{OPT} \geq 2$) on which EDF_{n-1} fails.



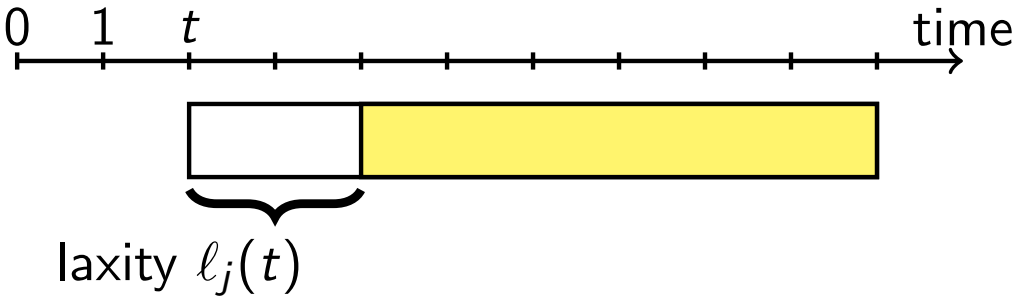
Def. Let $\alpha < 1$. Job j is α-tight if $p_j > \alpha(d_j - r_j)$ and α-loose othw.

Lemma

If every job is α-loose, then $\text{EDF}_\star$ is $\frac{1}{(1-\alpha)^2}$-competitive.

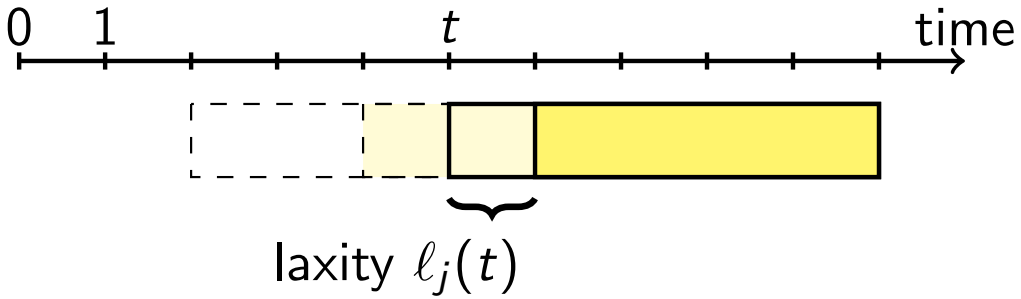
Least Laxity First (LLF)

Laxity of a job



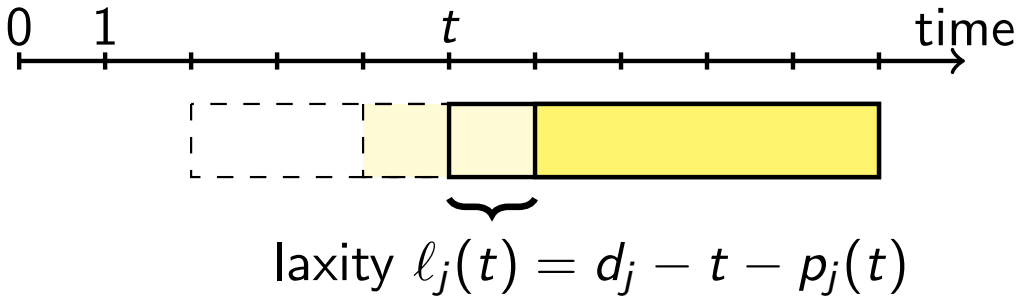
Least Laxity First (LLF)

Laxity of a job



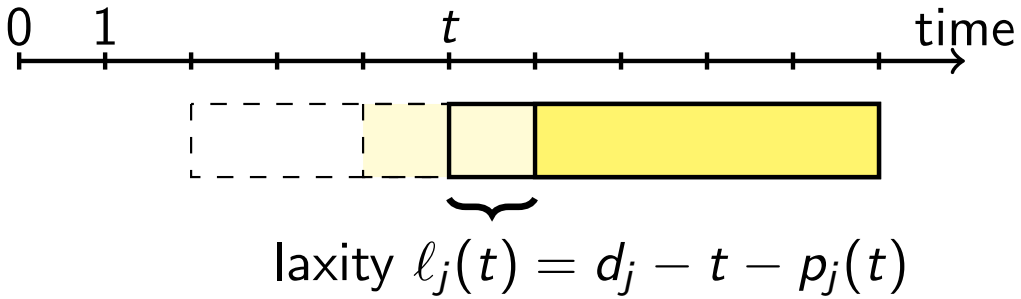
Least Laxity First (LLF)

Laxity of a job



Least Laxity First (LLF)

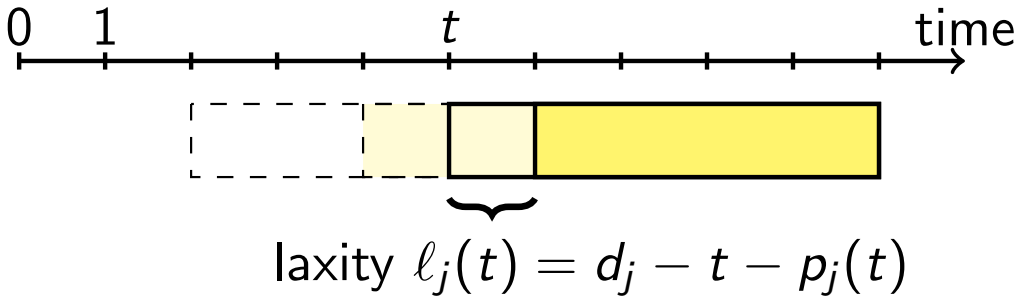
Laxity of a job



Algorithm Least Laxity First: At any time schedule m available jobs with minimum laxity and preempt other jobs if necessary.

Least Laxity First (LLF)

Laxity of a job



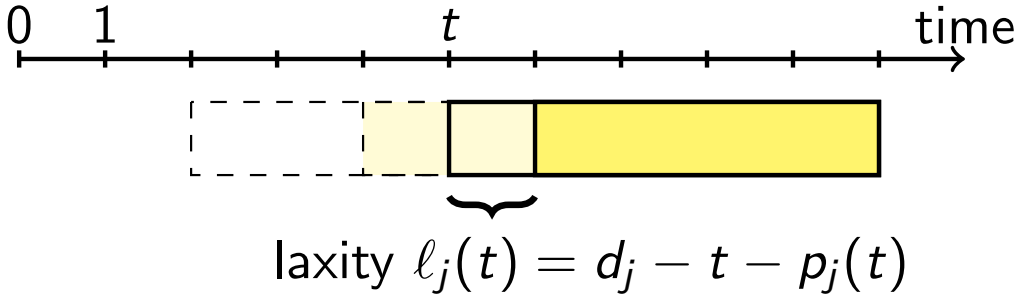
Algorithm Least Laxity First: At any time schedule m available jobs with minimum laxity and preempt other jobs if necessary.

Theorem [Philips et al., STOC 1997]

LLF may fail on any number of machines only dependent on OPT.

Least Laxity First (LLF)

Laxity of a job



Algorithm Least Laxity First: At any time schedule m available jobs with minimum laxity and preempt other jobs if necessary.

Theorem [Philips et al., STOC 1997]

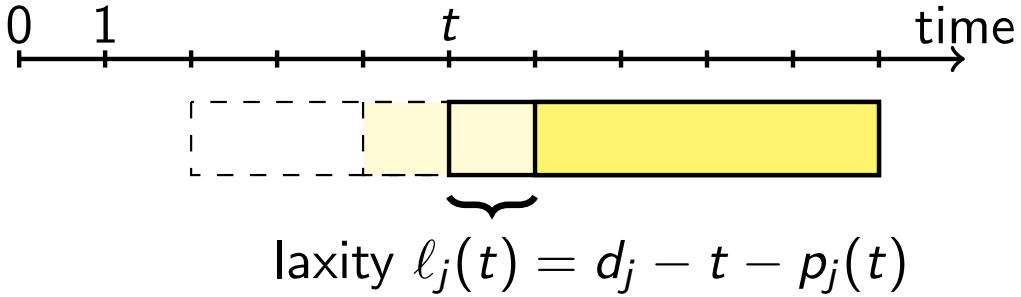
LLF may fail on any number of machines only dependent on OPT.

Theorem

There are instances on which LLF fails on $o(n^{\frac{1}{3}})$ machines.

Least Laxity First (LLF)

Laxity of a job



Algorithm Least Laxity First: At any time schedule m available jobs with minimum laxity and preempt other jobs if necessary.

Theorem [Philips et al., STOC 1997]

LLF may fail on any number of machines only dependent on OPT.

Theorem

There are instances on which LLF fails on $o(n^{\frac{1}{3}})$ machines.

Lemma. LLF on α-loose jobs again $\frac{1}{(1-\alpha)^2}$-competitive.

Agreeable instances

The setting:

- Assume m is known.

Agreeable instances

The setting:

- Assume m is known.
- Assume all jobs are α-tight, i.e., $p_j > \alpha(d_j - r_j)$, for some $\alpha > 1$.

Agreeable instances

The setting:

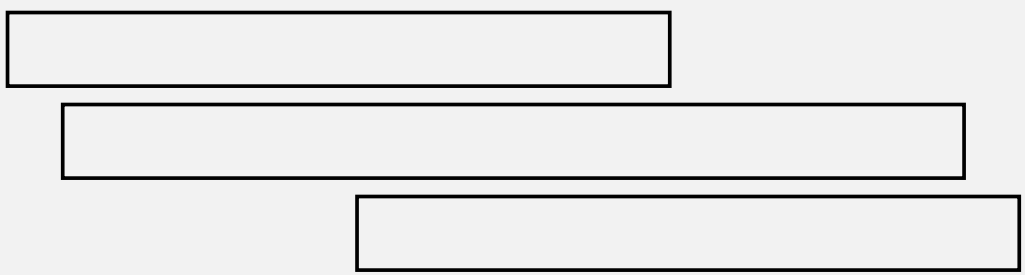
- Assume m is known.
- Assume all jobs are α-tight, i.e., $p_j > \alpha(d_j - r_j)$, for some $\alpha > 1$.
- Schedule α-loose jobs by EDF.
  $\longrightarrow$ EDF is non-preemptive on agreeable instances.

Agreeable instances

The setting:

- Assume m is known.
- Assume all jobs are α-tight, i.e., $p_j > \alpha(d_j - r_j)$, for some $\alpha > 1$.
- Schedule α-loose jobs by EDF.
  $\longrightarrow$ EDF is non-preemptive on agreeable instances.

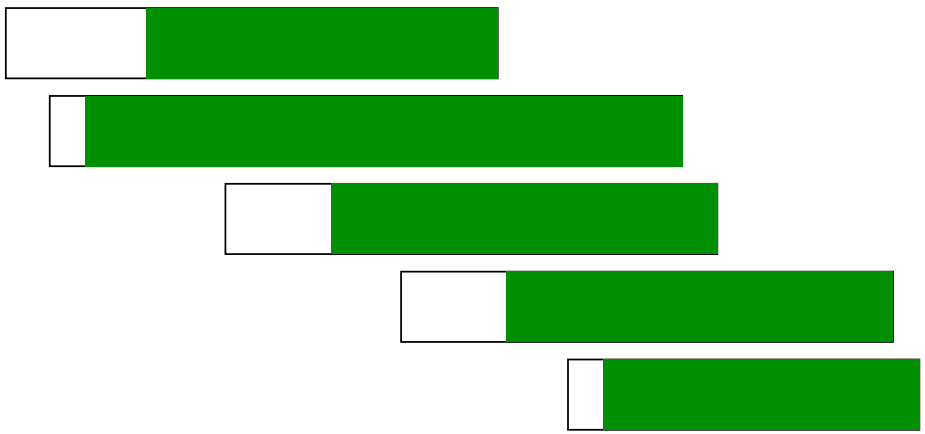
For any two jobs j, k: $r_j < r_k$ implies $d_j \leq d_k$.



Agreeable Instances

Algorithm MediumFit

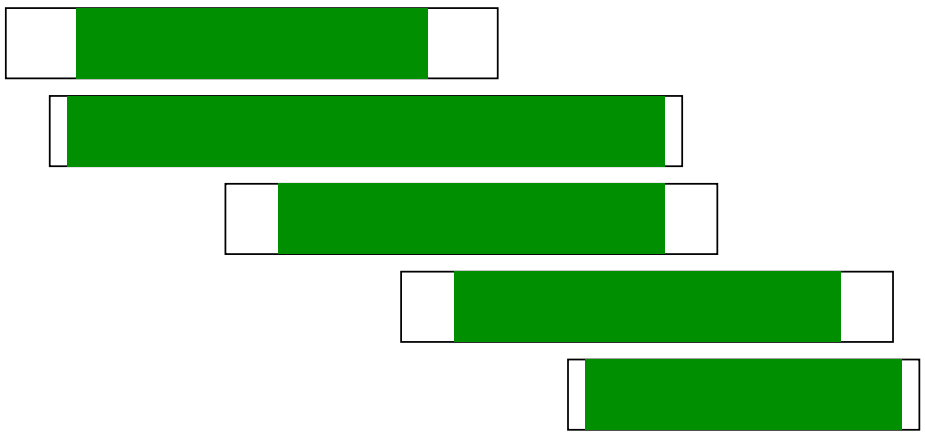
Schedule each job in the middle of its time window.



Agreeable Instances

Algorithm MediumFit

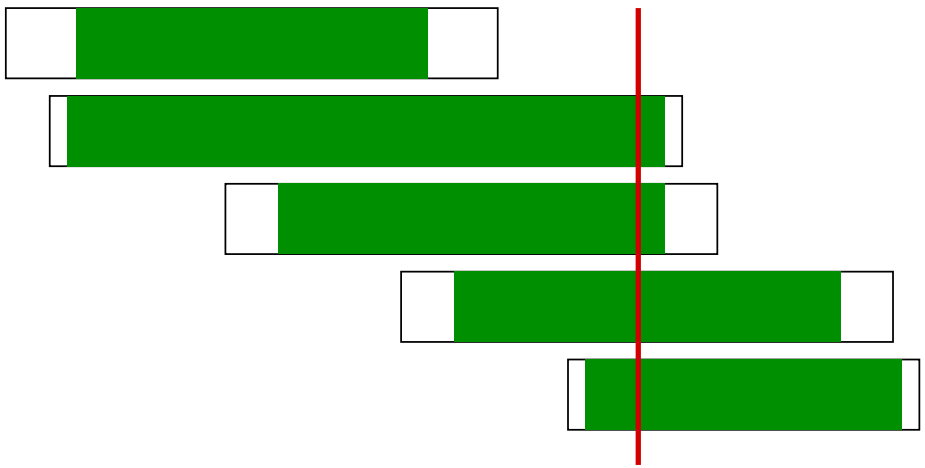
Schedule each job in the middle of its time window.



Agreeable Instances

Algorithm MediumFit

Schedule each job in the middle of its time window.

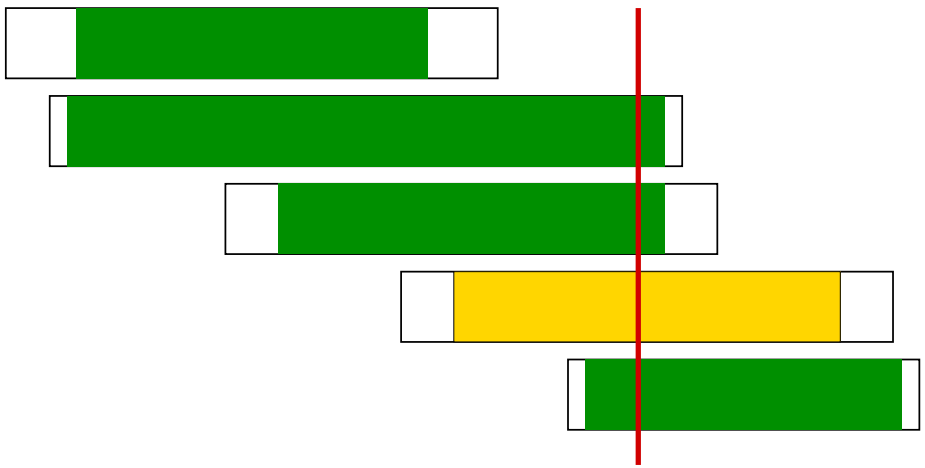


Lemma. At any time t, MediumFit is processing at most $\mathcal{O}(m)$ jobs.

Agreeable Instances

Algorithm MediumFit

Schedule each job in the middle of its time window.

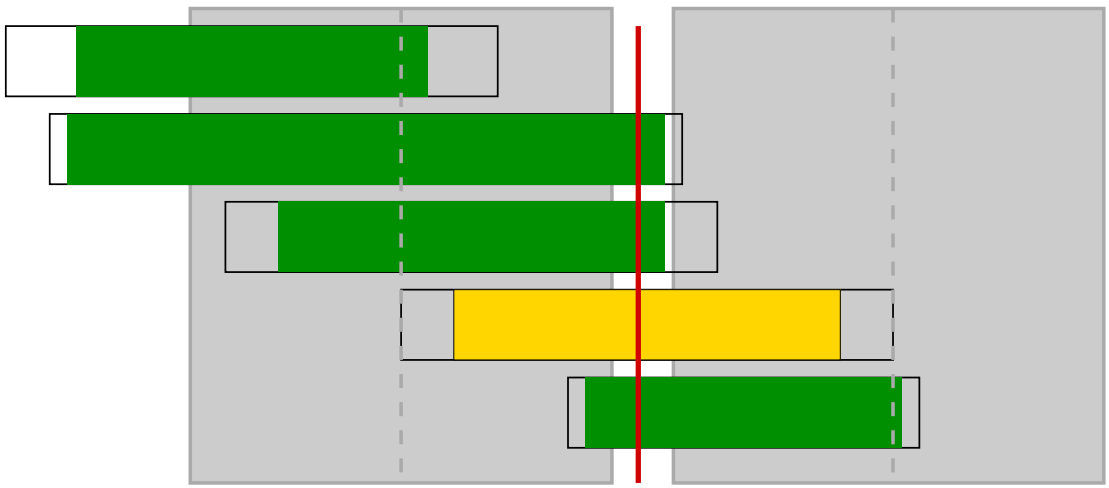


Lemma. At any time t, MediumFit is processing at most $\mathcal{O}(m)$ jobs.

Agreeable Instances

Algorithm MediumFit

Schedule each job in the middle of its time window.

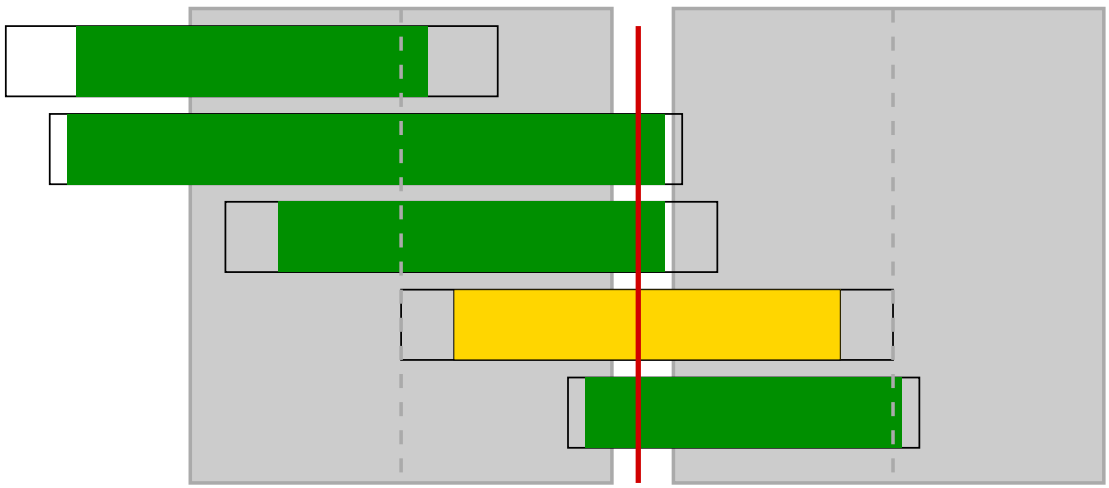


Lemma. At any time t, MediumFit is processing at most $\mathcal{O}(m)$ jobs.

Agreeable Instances

Algorithm MediumFit

Schedule each job in the middle of its time window.



Lemma. At any time t, MediumFit is processing at most $\mathcal{O}(m)$ jobs.

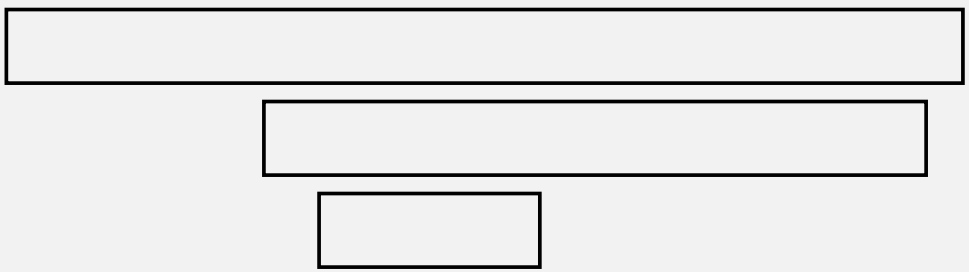
Theorem

EDF + MediumFit is a (non-)preemptive $\mathcal{O}(1)$-competitive algorithm.

Laminar instances

The setting:

For any two jobs j, k with intersecting time windows I_j, I_k:
either $I_j \subseteq I_k$ or $I_k \subseteq I_j$.



- Assume m is known.
- Schedule α-loose jobs by EDF.
- Assume all jobs are α-tight, i.e., $p_j > \alpha(d_j - r_j)$, for some $\alpha > 1$.

Laminar instances: High-level procedure

Open $m' = \mathcal{O}(m \log m)$ machines.

1. Assignment procedure
   - Upon arrival, a job is assigned to a machine and never migrates.
   - Jobs with larger time interval get assigned first (laminar jobs).
   - Fit full time interval into laxity of smallest already assigned job.
2. Scheduling procedure
   - Single machine scheduling: smallest job first (= EDF)

Laminar instances: Assignment procedure

Upon arrival of job j with time window $[r_j, d_j]$:

Laminar instances: Assignment procedure

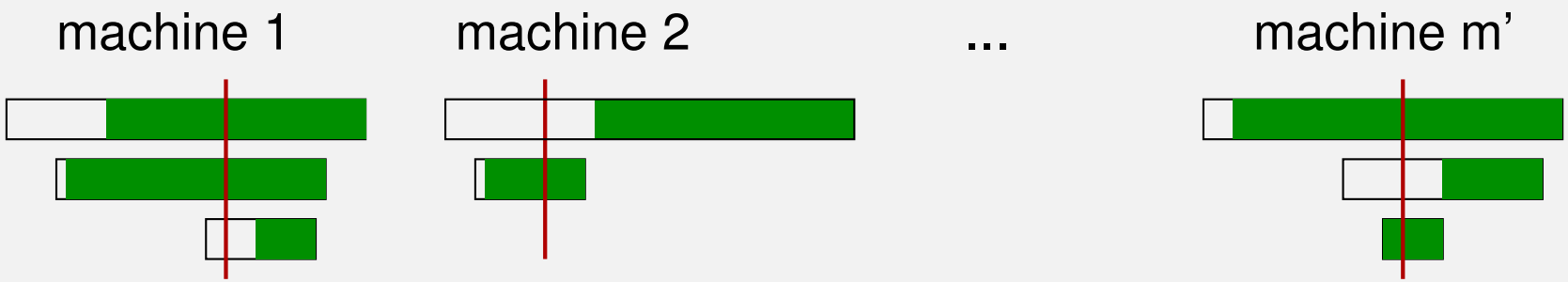
Upon arrival of job j with time window $[r_j, d_j]$:

- If there is a machine without any job covering r_j, then assign j.

Laminar instances: Assignment procedure

Upon arrival of job j with time window $[r_j, d_j]$:

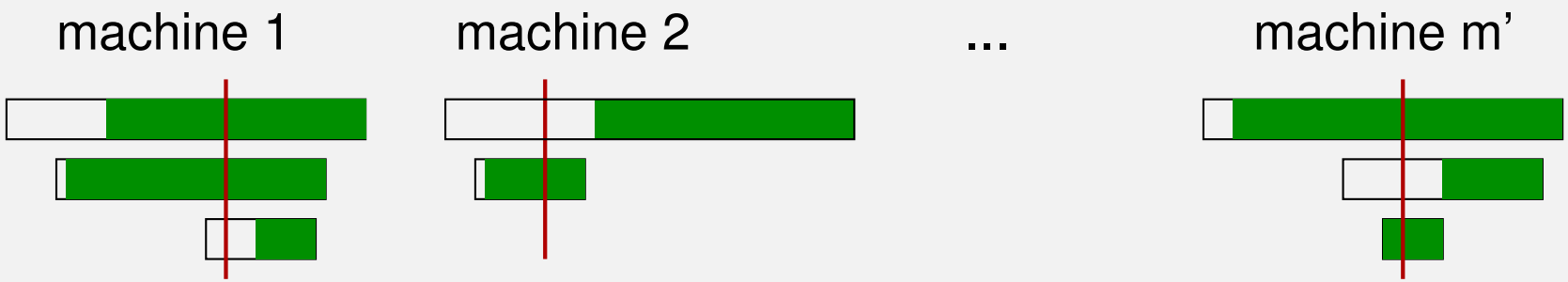
- If there is a machine without any job covering r_j, then assign j.
- Otherwise ...



Laminar instances: Assignment procedure

Upon arrival of job j with time window $[r_j, d_j]$:

- If there is a machine without any job covering r_j, then assign j.
- Otherwise ...

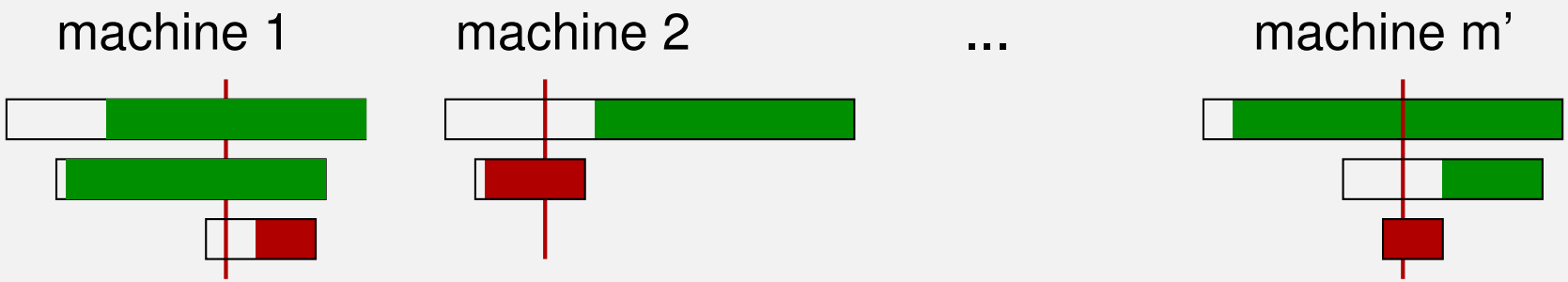


- On each machine consider inclusion minimal job.

Laminar instances: Assignment procedure

Upon arrival of job j with time window $[r_j, d_j]$:

- If there is a machine without any job covering r_j, then assign j.
- Otherwise ...

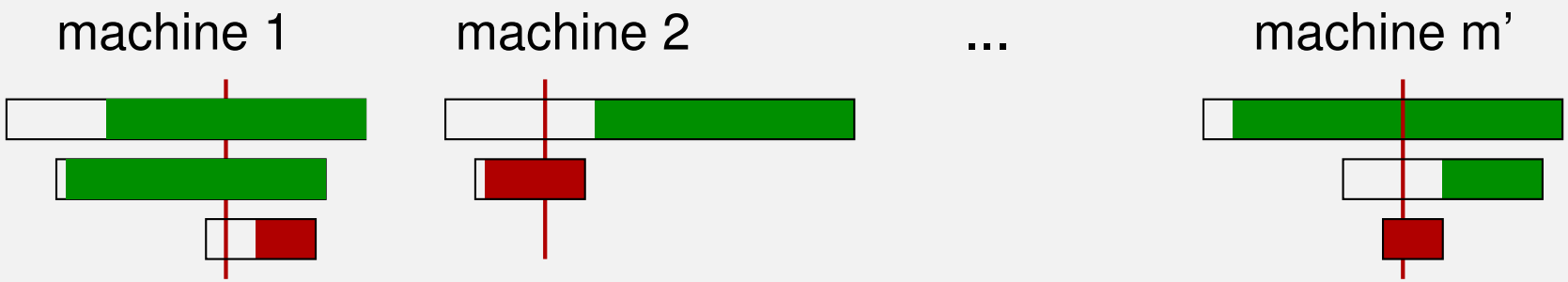


- On each machine consider inclusion minimal job.

Laminar instances: Assignment procedure

Upon arrival of job j with time window $[r_j, d_j]$:

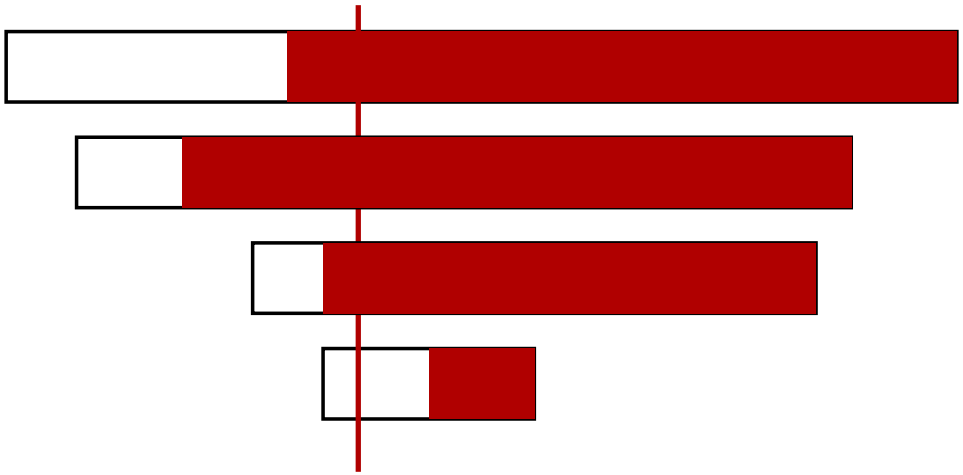
- If there is a machine without any job covering r_j, then assign j.
- Otherwise ...



- On each machine consider inclusion minimal job.
- These laminar jobs form an inclusion chain.

Laminar instances: Assignment procedure

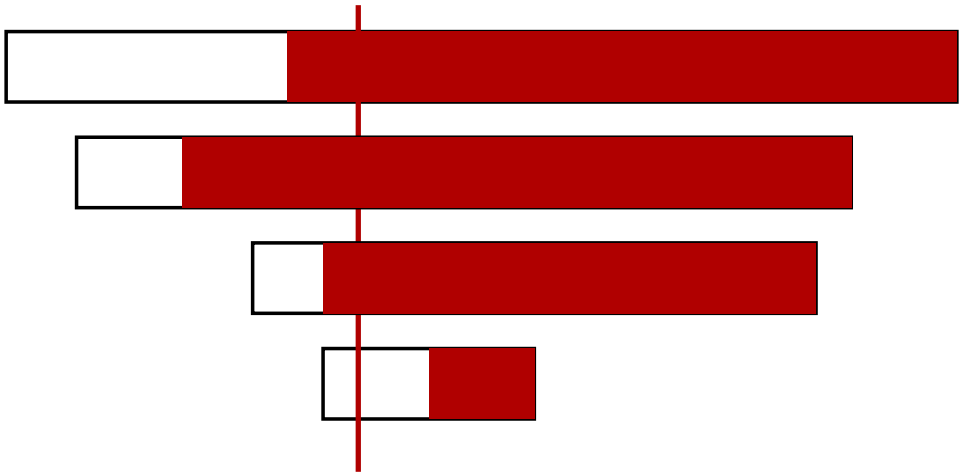
- Inclusion minimal jobs



Idea:

Laminar instances: Assignment procedure

- Inclusion minimal jobs

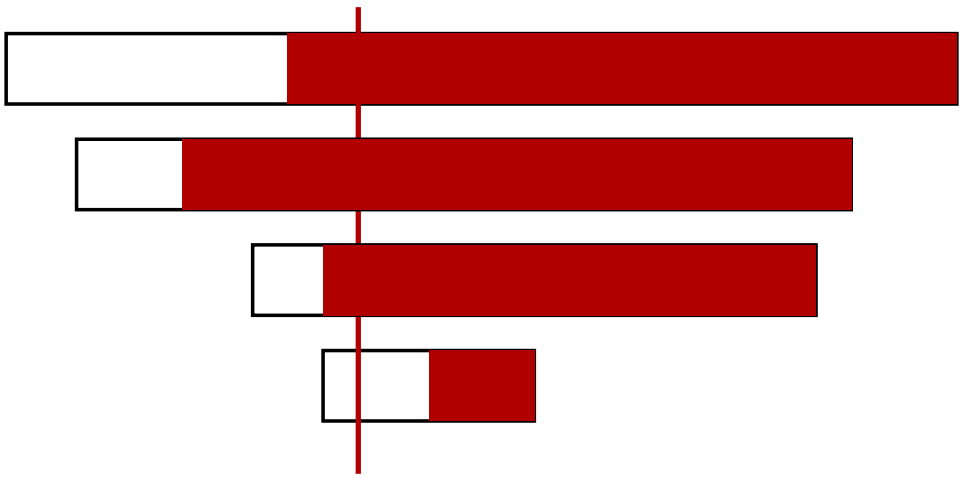


Idea:

- Select a job k such that j's time window fits into k's laxity.

Laminar instances: Assignment procedure

- Inclusion minimal jobs

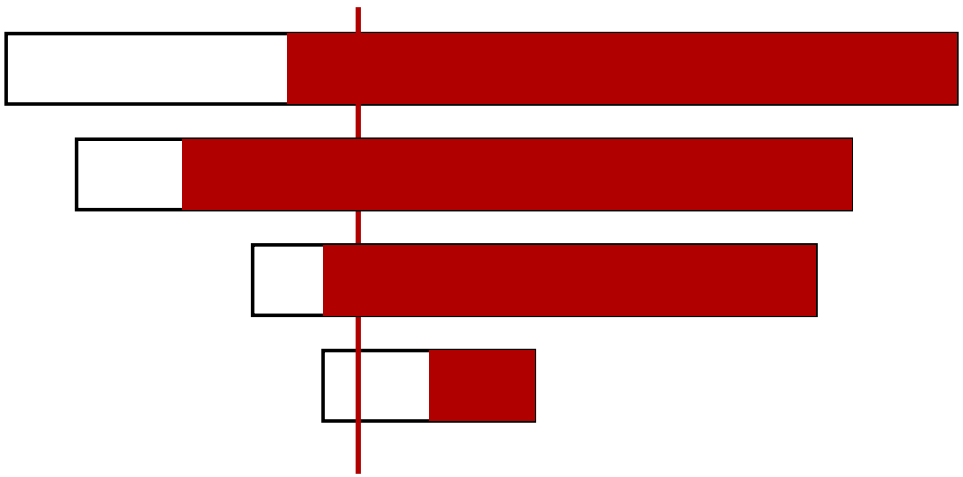


Idea:

- Select a job k such that j's time window fits into k's laxity.
- Keep track of previously assigned jobs.

Laminar instances: Assignment procedure

- Inclusion minimal jobs



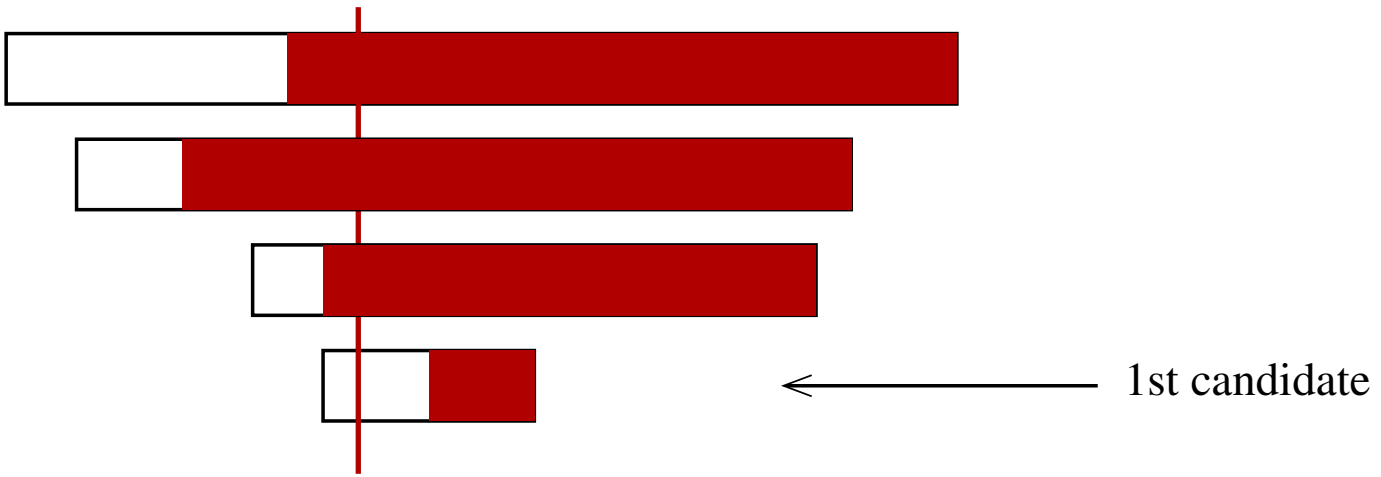
Idea:

- Select a job k such that j's time window fits into k's laxity.
- Keep track of previously assigned jobs.

Problem: greedy assignment fails.

Laminar instances: Assignment procedure

- Inclusion minimal jobs



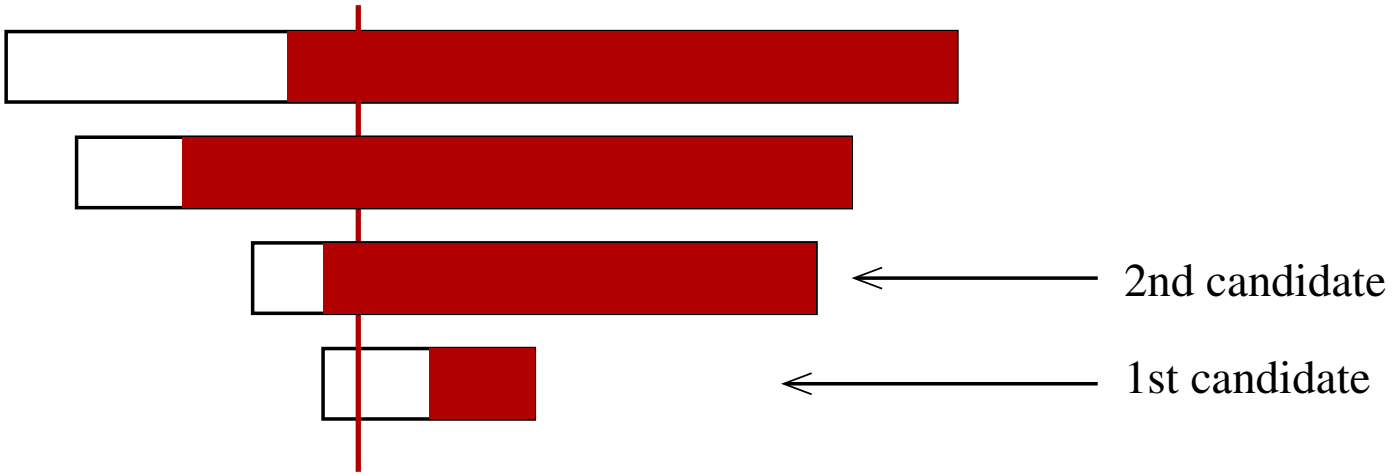
Idea:

- Select a job k such that j's time window fits into k's laxity.
- Keep track of previously assigned jobs.

Problem: greedy assignment fails.

Laminar instances: Assignment procedure

- Inclusion minimal jobs



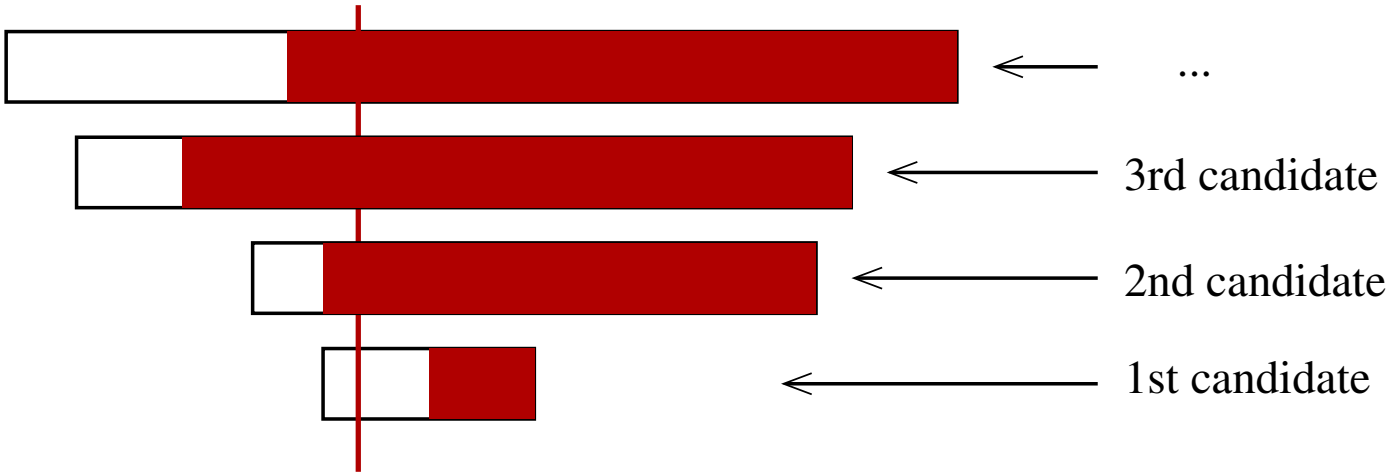
Idea:

- Select a job k such that j's time window fits into k's laxity.
- Keep track of previously assigned jobs.

Problem: greedy assignment fails.

Laminar instances: Assignment procedure

- Inclusion minimal jobs



Idea:

- Select a job k such that j's time window fits into k's laxity.
- Keep track of previously assigned jobs.

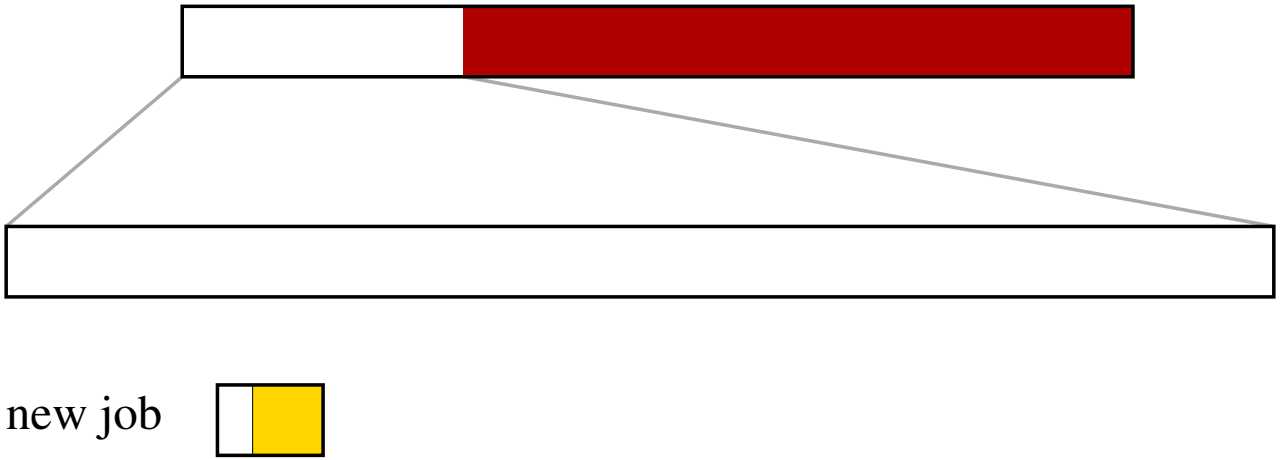
Problem: greedy assignment fails.

Laminar instances: Assignment & Scheduling

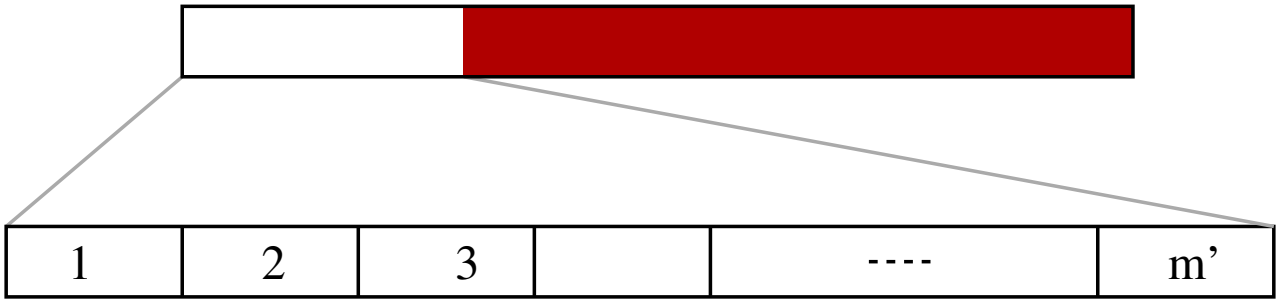
Laminar instances: Assignment & Scheduling

new job 

Laminar instances: Assignment & Scheduling

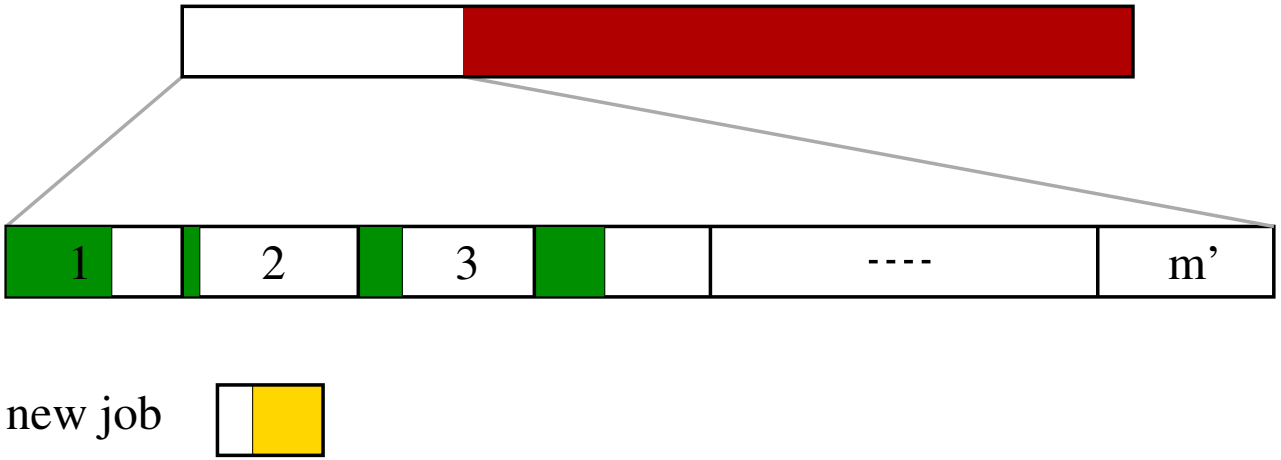


Laminar instances: Assignment & Scheduling

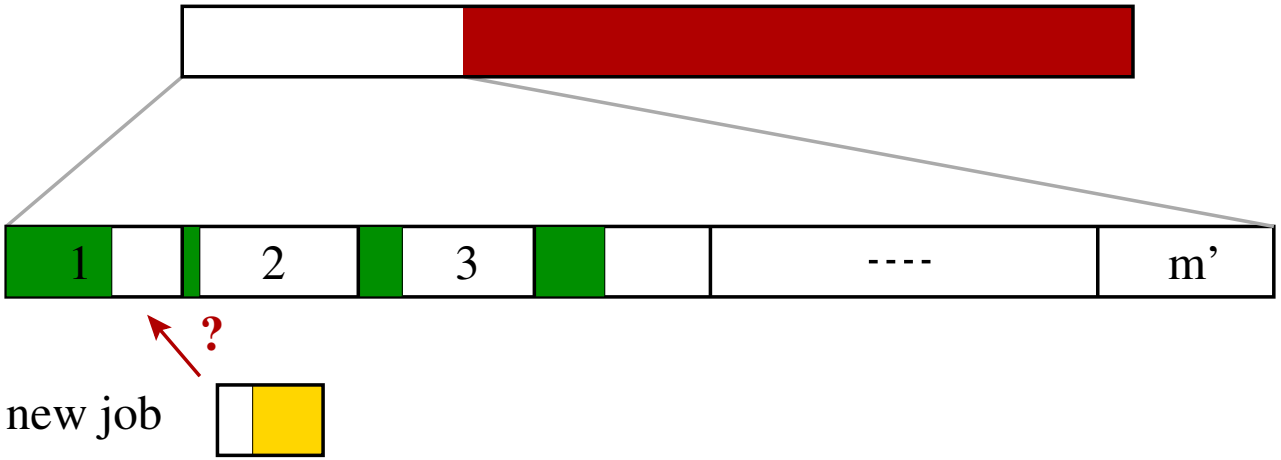


new job 

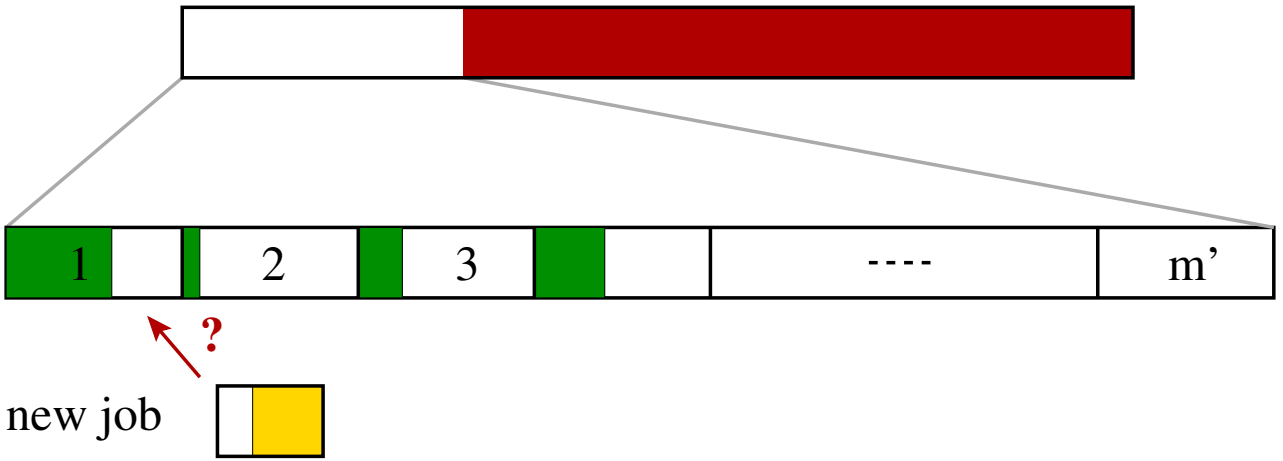
Laminar instances: Assignment & Scheduling



Laminar instances: Assignment & Scheduling



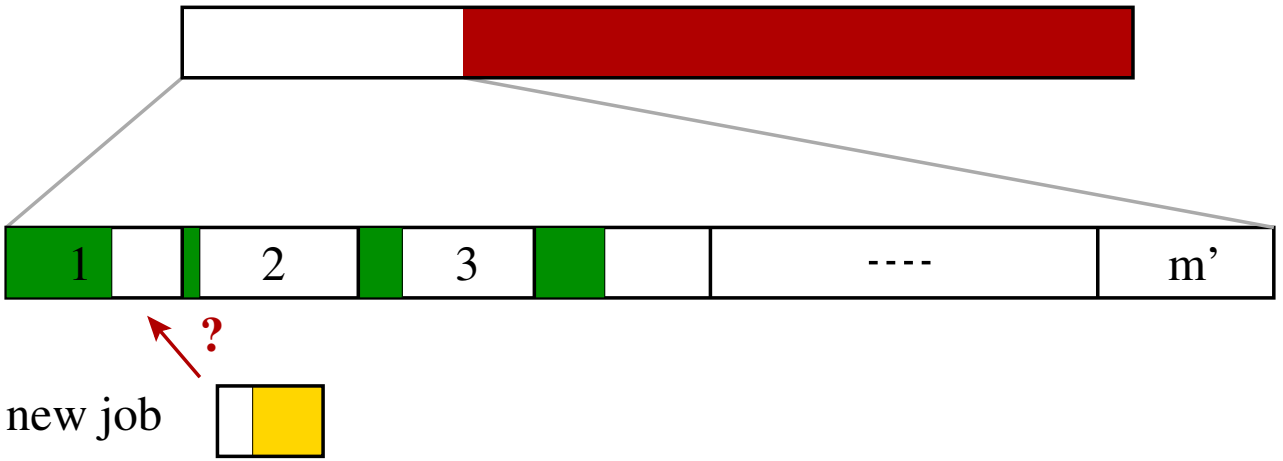
Laminar instances: Assignment & Scheduling



Assignment policy

For $i = 1, \ldots, m'$, if the *i*-th bin of the *i*-th candidate has sufficient capacity to accommodate the new job, then assign it and stop.

Laminar instances: Assignment & Scheduling



Assignment policy

For $i = 1, \ldots, m'$, if the *i*-th bin of the *i*-th candidate has sufficient capacity to accommodate the new job, then assign it and stop.

Scheduling policy

On any machine and at any time, schedule the inclusion minimal job.

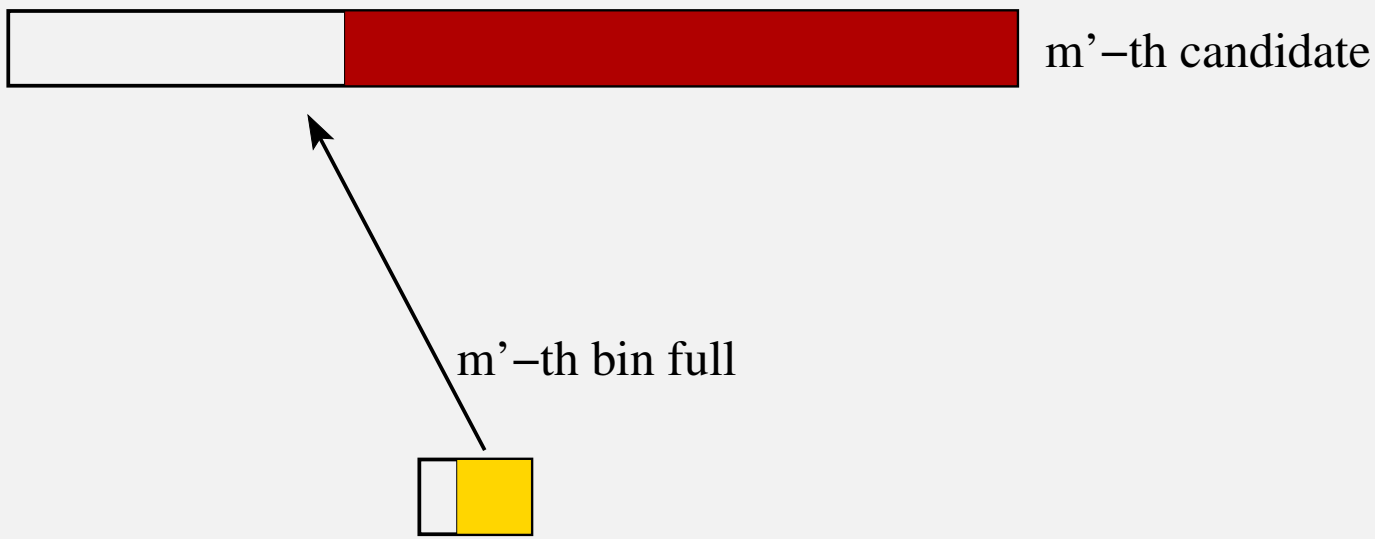
Laminar instances: Analysis sketch

To show: for $m' = \mathcal{O}(m \log m)$ the assignment never fails.

Laminar instances: Analysis sketch

To show: for $m' = \mathcal{O}(m \log m)$ the assignment never fails.

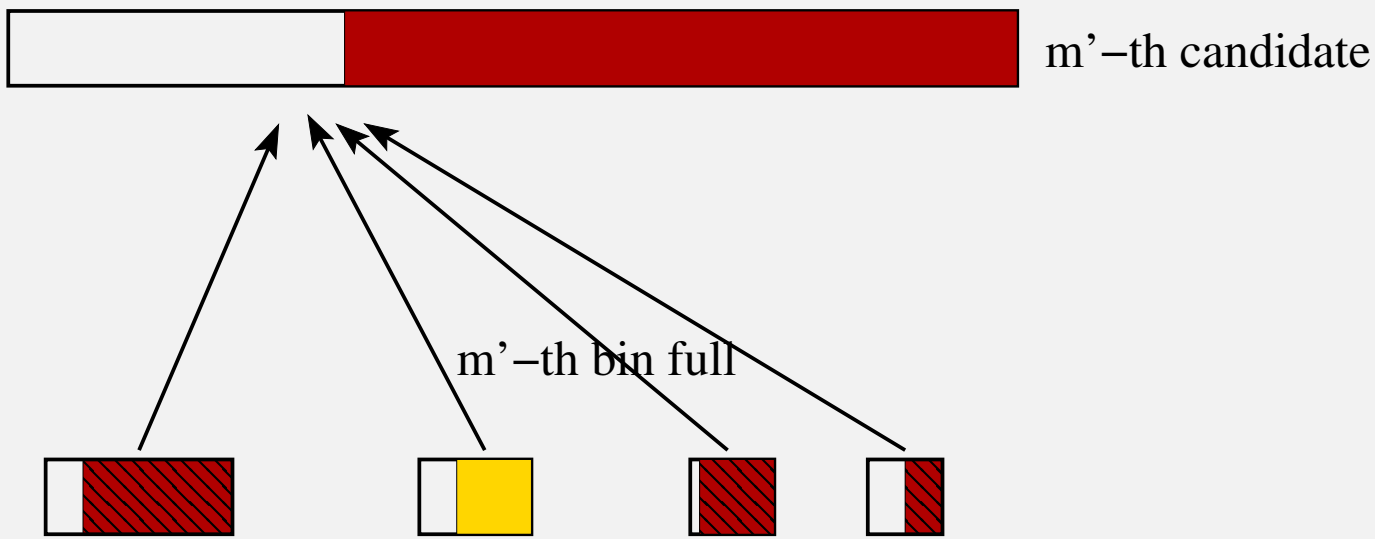
Suppose some job j does not find a bin of sufficient capacity.



Laminar instances: Analysis sketch

To show: for $m' = \mathcal{O}(m \log m)$ the assignment never fails.

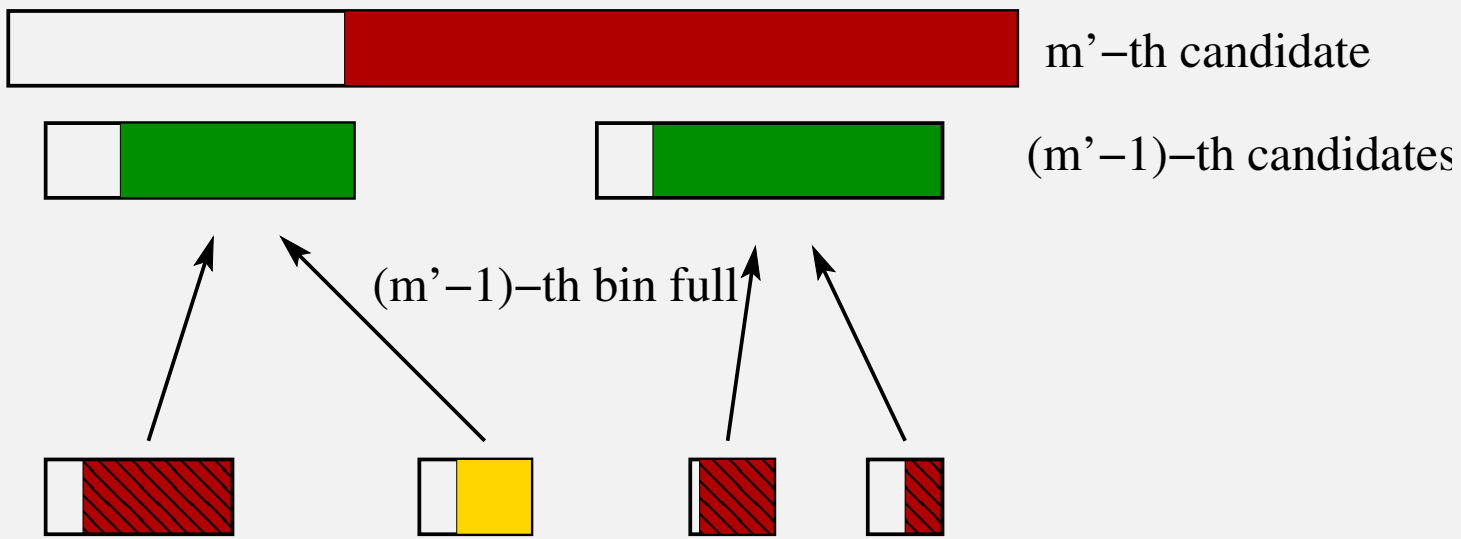
Suppose some job j does not find a bin of sufficient capacity.



Laminar instances: Analysis sketch

To show: for $m' = \mathcal{O}(m \log m)$ the assignment never fails.

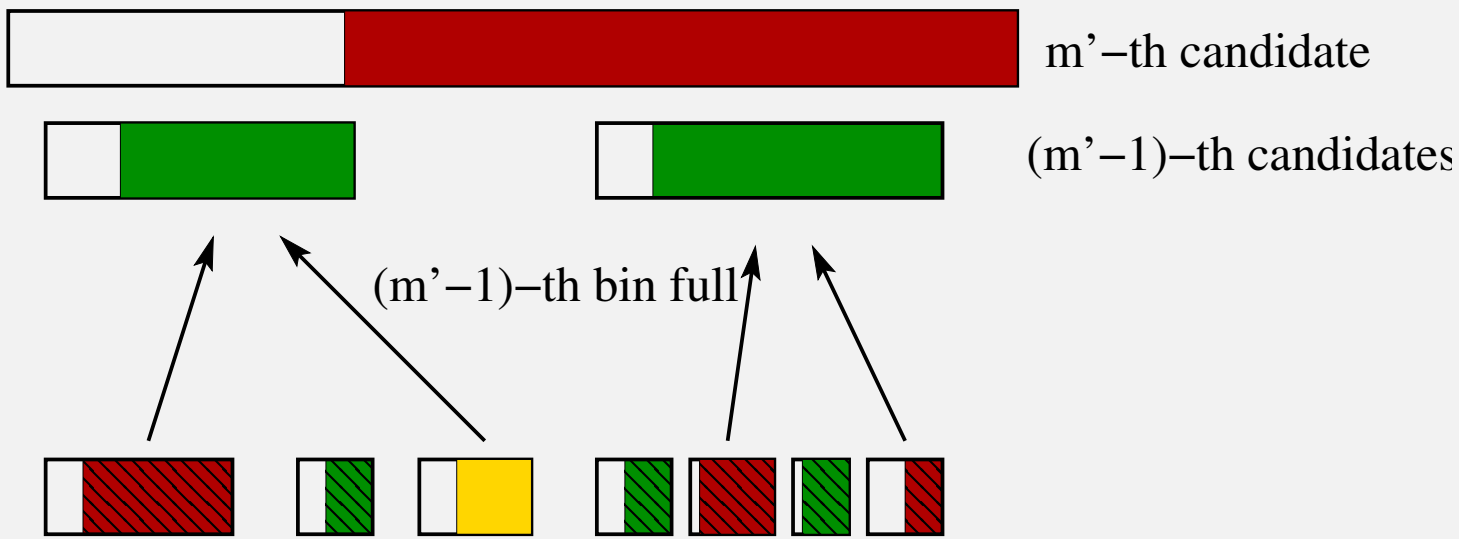
Suppose some job j does not find a bin of sufficient capacity.



Laminar instances: Analysis sketch

To show: for $m' = \mathcal{O}(m \log m)$ the assignment never fails.

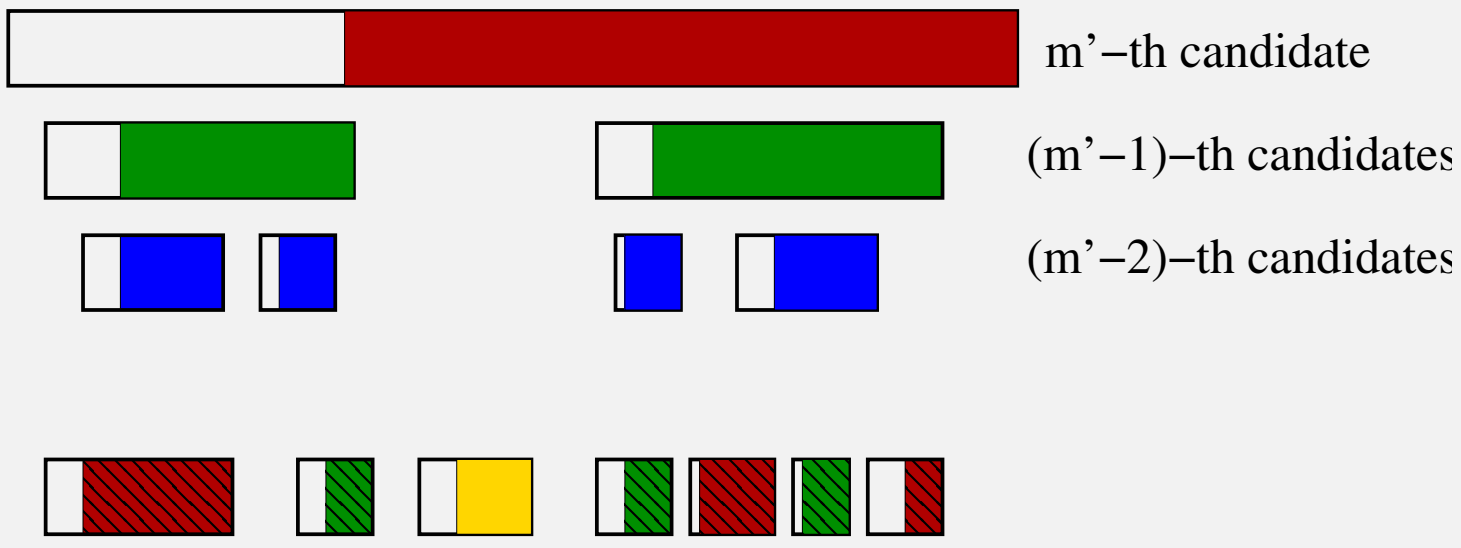
Suppose some job j does not find a bin of sufficient capacity.



Laminar instances: Analysis sketch

To show: for $m' = \mathcal{O}(m \log m)$ the assignment never fails.

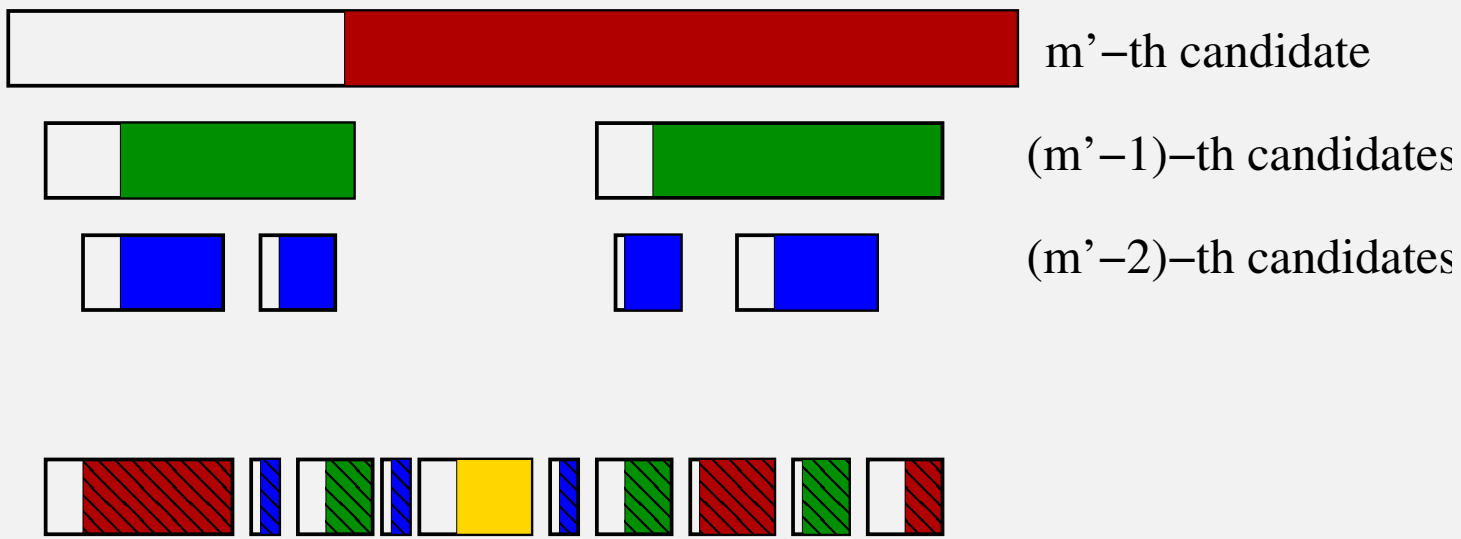
Suppose some job j does not find a bin of sufficient capacity.



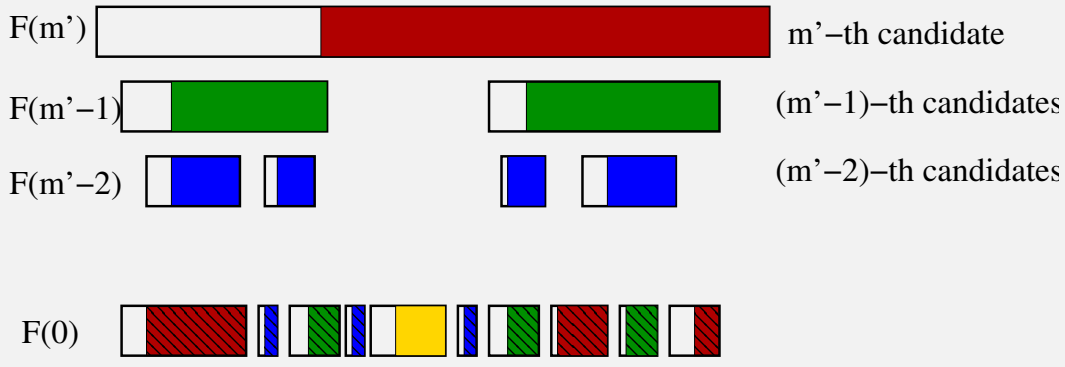
Laminar instances: Analysis sketch

To show: for $m' = \mathcal{O}(m \log m)$ the assignment never fails.

Suppose some job j does not find a bin of sufficient capacity.

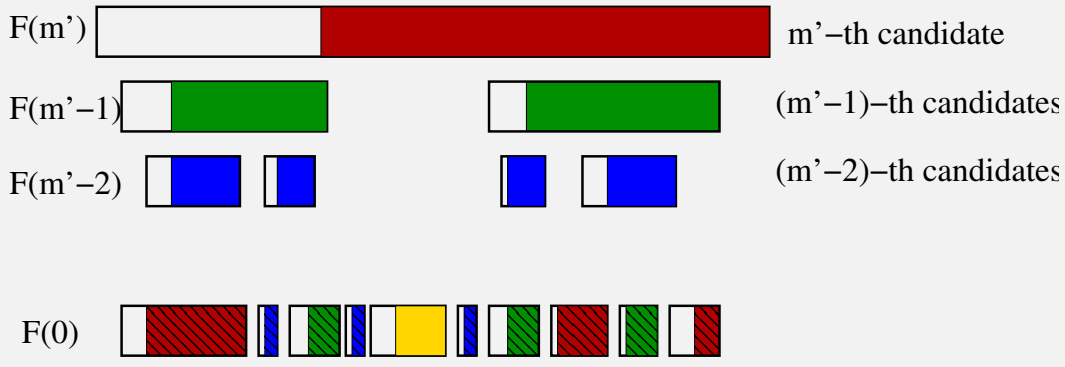


Laminar instances: Analysis sketch



Some properties:

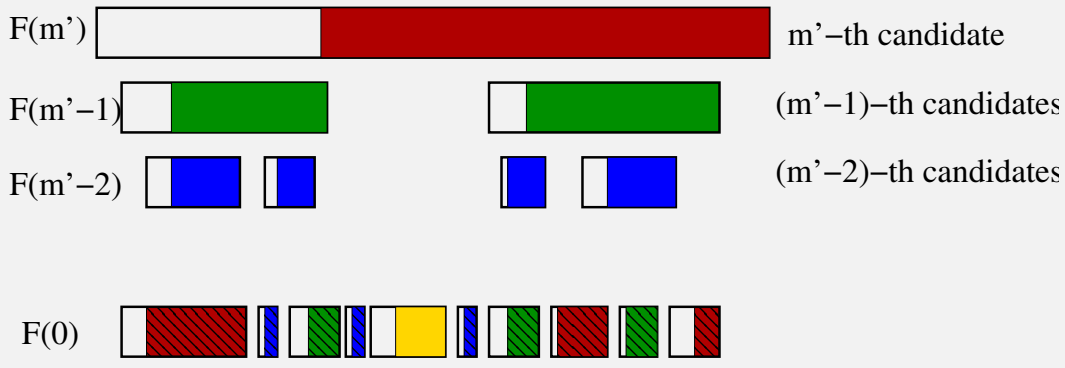
Laminar instances: Analysis sketch



Some properties:

- Sets $F(0), F(1), \ldots, F(m')$ are pairwise disjoint.

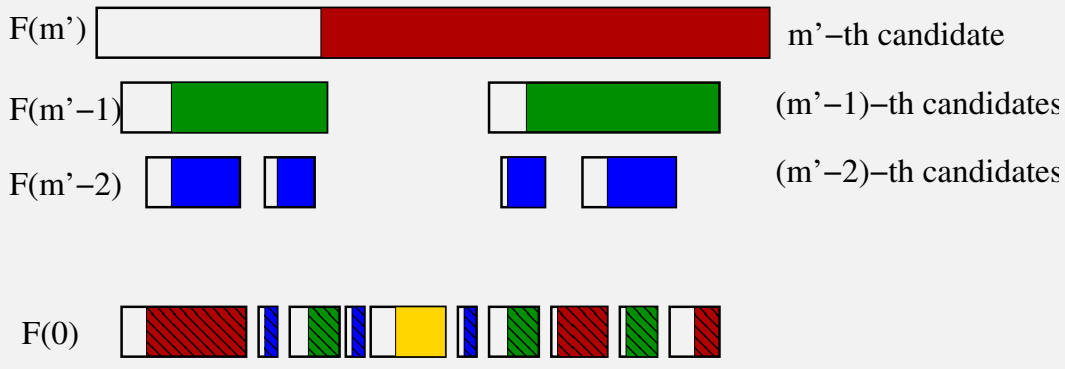
Laminar instances: Analysis sketch



Some properties:

- Sets $F(0), F(1), \ldots, F(m')$ are pairwise disjoint.
- $F(0)$ cover at least an $1/m'$-fraction of $F(i)$'s total laxity

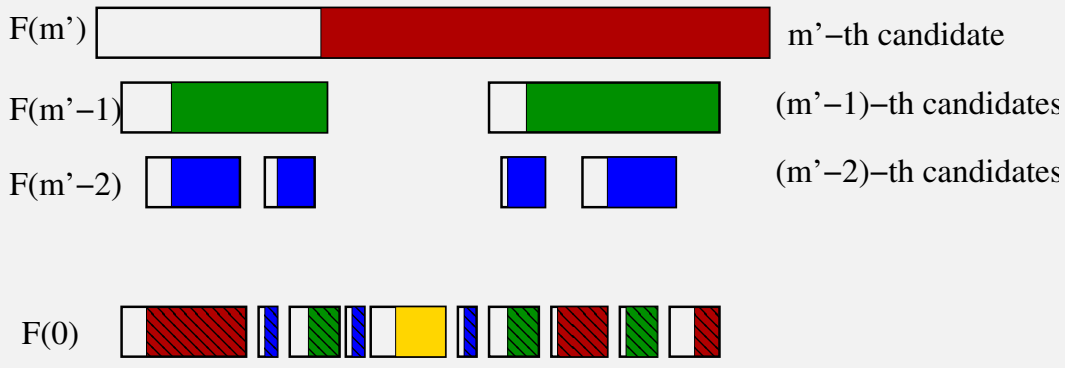
Laminar instances: Analysis sketch



Some properties:

- Sets $F(0), F(1), \ldots, F(m')$ are pairwise disjoint.
- $F(0)$ cover at least an $1/m'$-fraction of $F(i)$'s total laxity
- $F(i)$'s have geometrically increasing total length:$|F(i + \mathcal{O}(m))| \geq 2|F(i)|$.

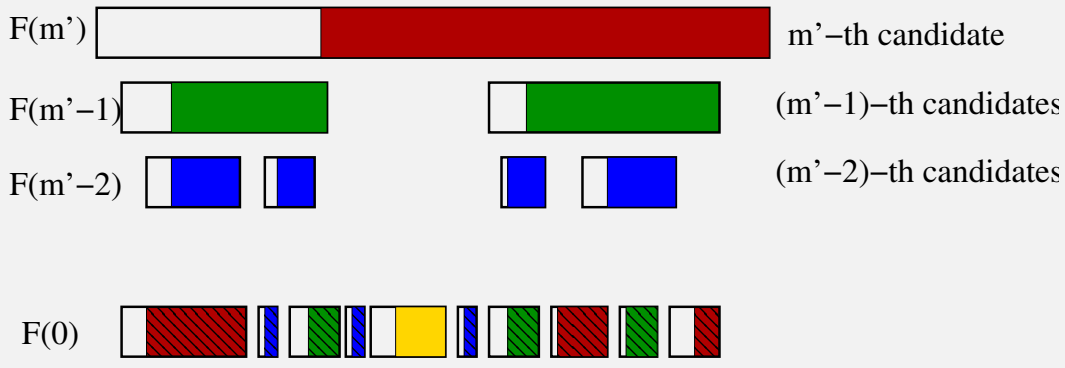
Laminar instances: Analysis sketch



Some properties:

- Sets $F(0), F(1), \ldots, F(m')$ are pairwise disjoint.
- $F(0)$ cover at least an $1/m'$-fraction of $F(i)$'s total laxity
- $F(i)$'s have geometrically increasing total length:$|F(i + \mathcal{O}(m))| \geq 2|F(i)|$.
- For $m' = \mathcal{O}(m \log m)$ contradiction to feasibility on m machines.

Laminar instances: Analysis sketch



Some properties:

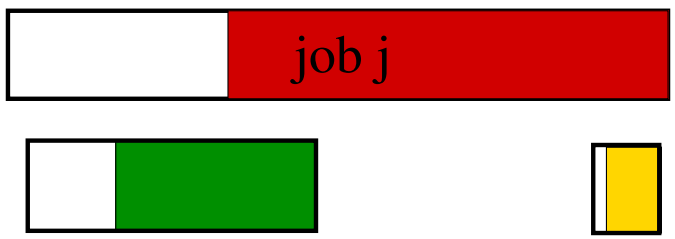
- Sets $F(0), F(1), \ldots, F(m')$ are pairwise disjoint.
- $F(0)$ cover at least an $1/m'$-fraction of $F(i)$'s total laxity
- $F(i)$'s have geometrically increasing total length:$|F(i + \mathcal{O}(m))| \geq 2|F(i)|$.
- For $m' = \mathcal{O}(m \log m)$ contradiction to feasibility on m machines.

Theorem

The algorithm is $\mathcal{O}(\log m)$-competitive for laminar instances.

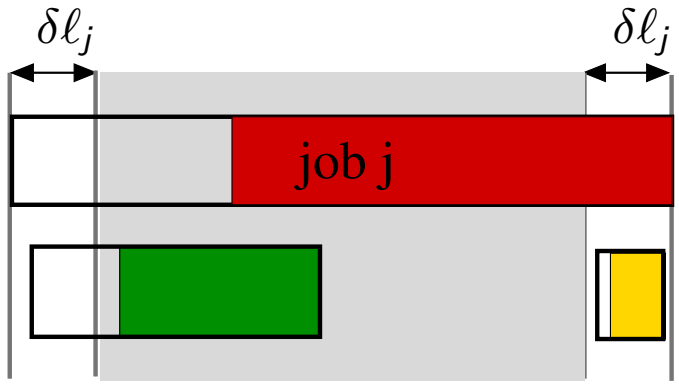
Outlook on a generalization

No clear separation into laminar or agreeable subinstances possible.



Outlook on a generalization

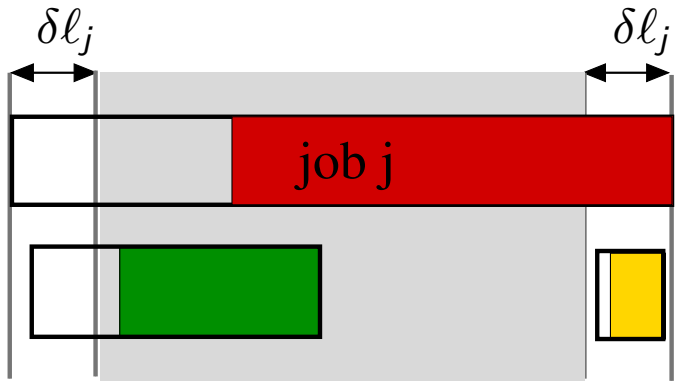
No clear separation into laminar or agreeable subinstances possible.



Job j δ-dominates j' if the time window of j' intersects with $[r_j + \delta\ell_j, d_j - \delta\ell_j]$.

Outlook on a generalization

No clear separation into laminar or agreeable subinstances possible.



Job j δ-dominates j' if the time window of j' intersects with $[r_j + \delta\ell_j, d_j - \delta\ell_j]$.

Lemma. There are at most $\mathcal{O}(m)$ jobs that dominate the same job but do not dominate each other.

Outlook on a generalization

1. Assignment procedure
   - Jobs are assigned to $g = \mathcal{O}(m^2 \log m)$ groups.

2. Scheduling procedure
   - Schedule each group on $\mathcal{O}(m)$ machines.
   - At any time t schedule inclusion minimal jobs among those whose "relevant" time window contains t.

Outlook on a generalization

1. Assignment procedure
   - Jobs are assigned to $g = \mathcal{O}(m^2 \log m)$ groups.
   - When assigning a job j, consider only δ-dominating jobs and run procedure similar to laminar case

2. Scheduling procedure
   - Schedule each group on $\mathcal{O}(m)$ machines.
   - At any time t schedule inclusion minimal jobs among those whose "relevant" time window contains t.

Outlook on a generalization

1. Assignment procedure
   - Jobs are assigned to $g = \mathcal{O}(m^2 \log m)$ groups.
   - When assigning a job j, consider only δ-dominating jobs and run procedure similar to laminar case
     - prioritize by least laxity
     - choose an inclusion chain of $\Theta(g/m)$ candidates
2. Scheduling procedure
   - Schedule each group on $\mathcal{O}(m)$ machines.
   - At any time t schedule inclusion minimal jobs among those whose "relevant" time window contains t.

Outlook on a generalization

1. Assignment procedure
   - Jobs are assigned to $g = \mathcal{O}(m^2 \log m)$ groups.
   - When assigning a job j, consider only δ-dominating jobs and run procedure similar to laminar case
     - prioritize by least laxity
     - choose an inclusion chain of $\Theta(g/m)$ candidates
2. Scheduling procedure
   - Schedule each group on $\mathcal{O}(m)$ machines.
   - At any time t schedule inclusion minimal jobs among those whose "relevant" time window contains t.

Theorem

There is a preemptive $\mathcal{O}(m^2 \log m)$-competitive algorithm.

Summary & Open questions

Summary

- General $\mathcal{O}(m^2 \log m)$-competitive preemptive algorithm.
  - Agreeable instances: $\mathcal{O}(1)$-competitive (even non-preemptive)
  - Laminar instances: $\mathcal{O}(\log m)$-competitive

Summary & Open questions

Summary

- General $\mathcal{O}(m^2 \log m)$-competitive preemptive algorithm.
  - Agreeable instances: $\mathcal{O}(1)$-competitive (even non-preemptive)
  - Laminar instances: $\mathcal{O}(\log m)$-competitive

Open questions

- Does there exist a constant-competitive online algorithm?

Summary & Open questions

Summary

- General $\mathcal{O}(m^2 \log m)$-competitive preemptive algorithm.
  - Agreeable instances: $\mathcal{O}(1)$-competitive (even non-preemptive)
  - Laminar instances: $\mathcal{O}(\log m)$-competitive

Open questions

- Does there exist a constant-competitive online algorithm?
- Decrease the gap between $5/4$ and $\mathcal{O}(m^2 \log m)$.

Summary & Open questions

Summary

- General $\mathcal{O}(m^2 \log m)$-competitive preemptive algorithm.
  - Agreeable instances: $\mathcal{O}(1)$-competitive (even non-preemptive)
  - Laminar instances: $\mathcal{O}(\log m)$-competitive

Open questions

- Does there exist a constant-competitive online algorithm?
- Decrease the gap between $5/4$ and $\mathcal{O}(m^2 \log m)$.
- Tradeoff between machines and additional speed?

Summary & Open questions

Summary

- General $\mathcal{O}(m^2 \log m)$-competitive preemptive algorithm.
  - Agreeable instances: $\mathcal{O}(1)$-competitive (even non-preemptive)
  - Laminar instances: $\mathcal{O}(\log m)$-competitive

Open questions

- Does there exist a constant-competitive online algorithm?
- Decrease the gap between $5/4$ and $\mathcal{O}(m^2 \log m)$.
- Tradeoff between machines and additional speed?

- Stochastics: Random arrival model? Stochastic processing times?