Online scheduling of divisibles requests

Frédéric Vivien

13 & 27 novembre 2006

Outline

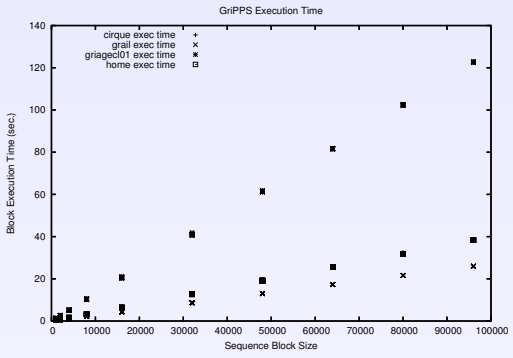
Outline

The problem

- An heterogeneous platform.
- A set of databanks
  - text files of several MB or GB;
  - each databank may be replicated;
  - a databank is not necessarily present on each processor.
- Some requests: comparing a motif with a databank
  - the requests are independent;
  - a very large number of requests;
  - motifs have different sizes.

Application: GriPPS from the Institut de Biologie et Chimie des Protéines

Analyzing the application : divisible jobs



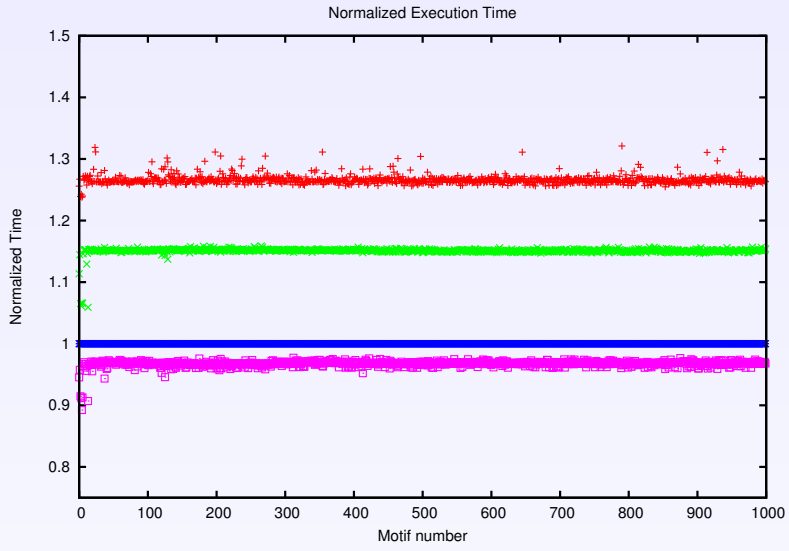
Experimental setup:

- Benchmark : a given set of motifs, compared with subsets of varying sizes of a databank.
- Each benchmark is executed 10 times.

Conclusions:

- A request execution time is linear in the size of the databank.
- Compact motif representation $\Rightarrow$ communication times are negligible. (another experiment)

Analyzing the application : uniform machines



- Determining the relative speeds of the processors
  - set of motifs individually compared on a set of machines
  - average on 40 iterations
  - average execution time compared to a reference machine

Notations

- Jobs J_1, ..., J_n
  Job J_j arrives in the system at date r_j.
  Job J_j has a weight (or a priority) w_j.

- Machines M_1, ..., M_m
  Machine M_i takes a time $c_{i,j}$ to process the job J_j.
  $c_{i,j}$ is infinite if job J_j needs a databank which is not present on machine M_i.

- Completion time of job J_j: C_j.
  Flow of job J_j : $F_j = C_j - r_j$ (time spent in the system)

Divisibility

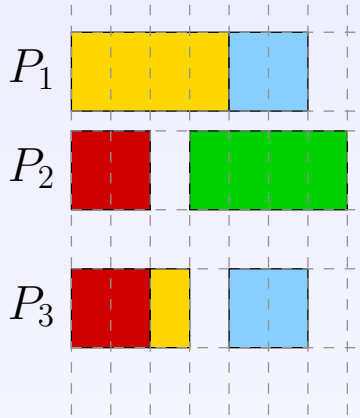
- Each job is inherently divisible: at any time, different processors can compare the motif against different subparts of the same databank.

- $\alpha_{i,j}$: fraction of job J_j processed by machine M_i:

$$\forall j, \sum_i \alpha_{i,j} = 1.$$

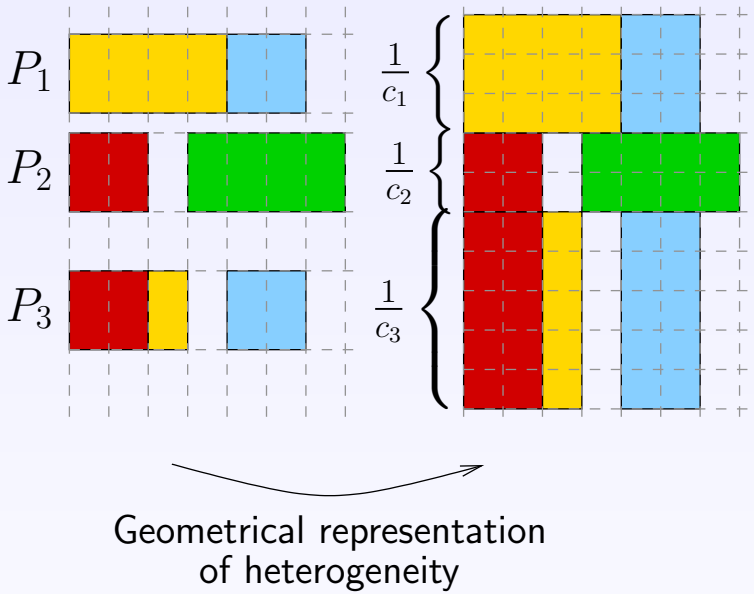
From divisible jobs to the uni-processor case

Geometric transformation of a uniform and divisible problem into a uni-processor problem.



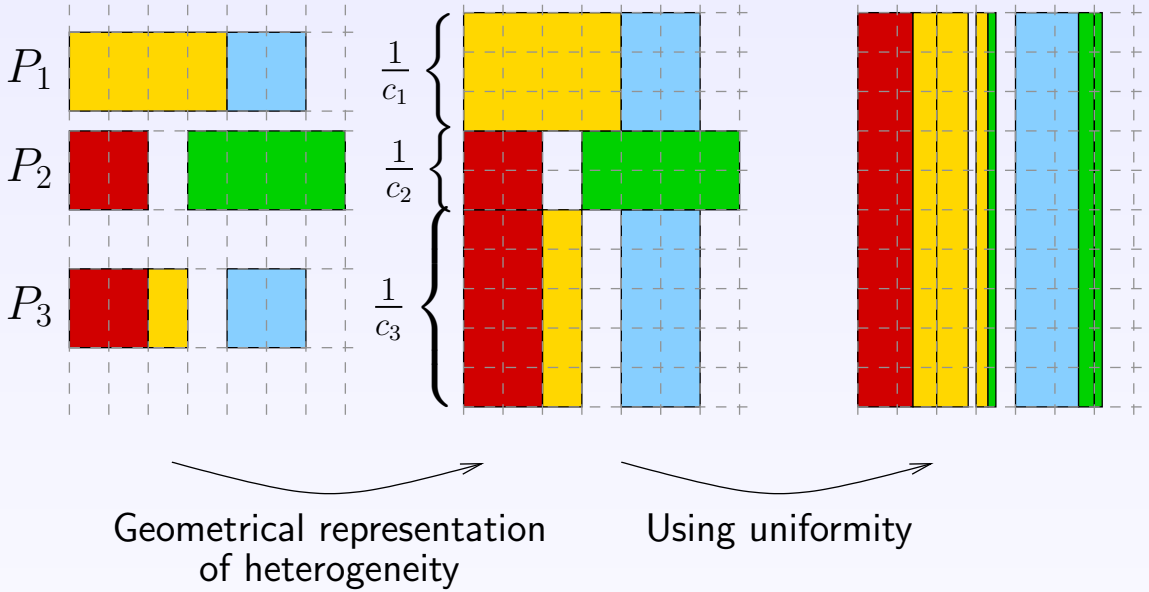
From divisible jobs to the uni-processor case

Geometric transformation of a uniform and divisible problem into a uni-processor problem.



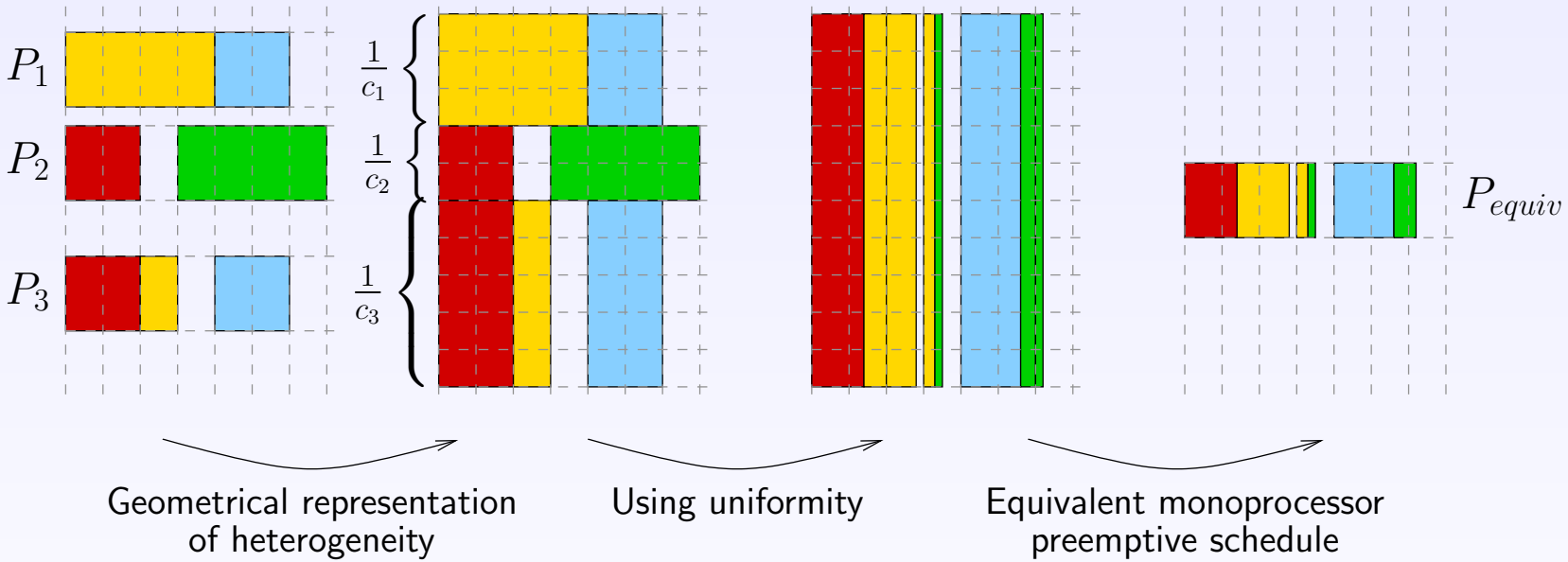
From divisible jobs to the uni-processor case

Geometric transformation of a uniform and divisible problem into a uni-processor problem.



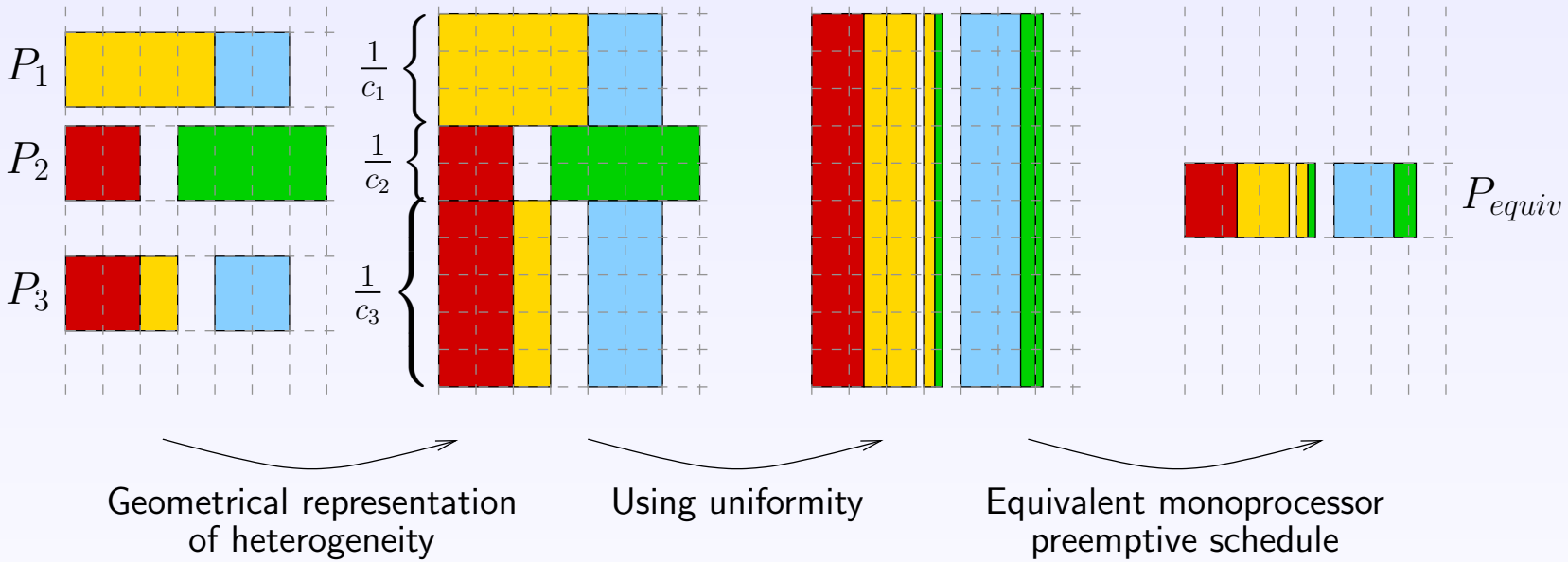
From divisible jobs to the uni-processor case

Geometric transformation of a uniform and divisible problem into a uni-processor problem.



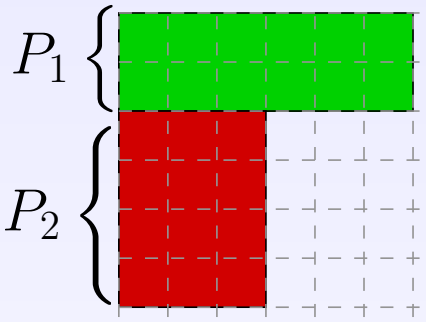
From divisible jobs to the uni-processor case

Geometric transformation of a uniform and divisible problem into a uni-processor problem.

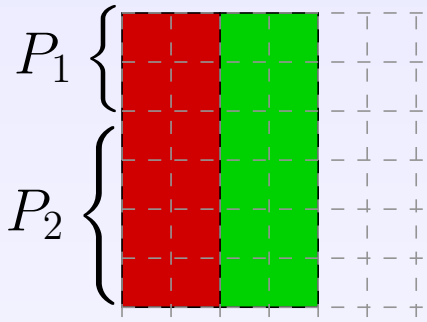


We can restrict the study to the uni-processor case...
... except that we are in the case of uni-processor with availability.

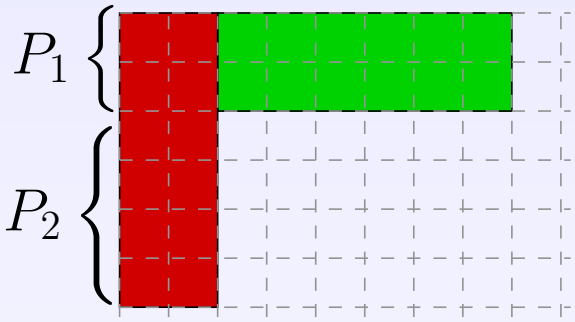
From the uni-processor case to divisible jobs



A: original schedule

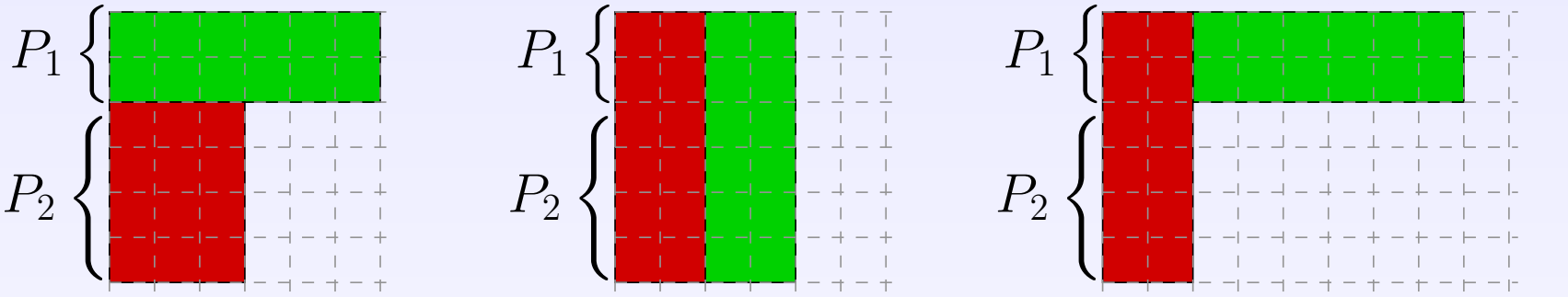


B: uniform machines



C: restricted availabilities

From the uni-processor case to divisible jobs



Simple rule to extend scheduling strategies defined for the uni-processor case:

1: **while** some processors are idle **do**
2: Select a job of highest priority and distribute it on available processors which are able to process it.

What should we optimize ?

- Makespan: $\max_j C_j$.
  Optimization of the machine usage.
  Arrival dates are not taken into account.

What should we optimize ?

- Makespan: $\max_j C_j$.
  Optimization of the machine usage.
  Arrival dates are not taken into account.
- Average flow or response time: $\sum_j (C_j - r_j)$.
  Optimization from the user point of view.

What should we optimize ?

- Makespan: $\max_j C_j$.
  Optimization of the machine usage.
  Arrival dates are not taken into account.
- Average flow or response time: $\sum_j (C_j - r_j)$.
  Optimization from the user point of view.
  Inconvenient: starvation.

What should we optimize ?

- Makespan: $\max_j C_j$.
  Optimization of the machine usage.
  Arrival dates are not taken into account.
- Average flow or response time: $\sum_j (C_j - r_j)$.
  Optimization from the user point of view.
  Inconvenient: starvation.
- Maximum flow or maximum response time: $\max_j (C_j - r_j)$.
  No starvation. Favor long jobs. Worst-case optimization.

What should we optimize ?

- Makespan: $\max_j C_j$.
  Optimization of the machine usage.
  Arrival dates are not taken into account.
- Average flow or response time: $\sum_j (C_j - r_j)$.
  Optimization from the user point of view.
  Inconvenient: starvation.
- Maximum flow or maximum response time: $\max_j (C_j - r_j)$.
  No starvation. Favor long jobs. Worst-case optimization.
- Maximum weighted flow: $\max_j w_j(C_j - r_j)$.
  Gives back some importance to short jobs.

What should we optimize ?

- Makespan: $\max_j C_j$.
  Optimization of the machine usage.
  Arrival dates are not taken into account.
- Average flow or response time: $\sum_j (C_j - r_j)$.
  Optimization from the user point of view.
  Inconvenient: starvation.
- Maximum flow or maximum response time: $\max_j (C_j - r_j)$.
  No starvation. Favor long jobs. Worst-case optimization.
- Maximum weighted flow: $\max_j w_j(C_j - r_j)$.
  Gives back some importance to short jobs.
  Particular case of the *stretch*:
  w_j=1/running time of the job on empty platform.

Evaluating the quality of an online schedule

An online algorithm has a competitive factor ρ if and only if:

Whatever the set of jobs T_1, ..., T_n:

Online schedule cost$(T_1, ..., T_N) \leq$
$\rho \times$ Optimal off-line schedule cost$(T_1, ..., T_N)$.

Stretch minimization

We consider minimizing the maximum or average stretch

Stretch minimization

We consider minimizing the maximum or average stretch

Theorem

Δ: ratio of the sizes of the largest and smallest job.
Consider any online algorithm whose competitive ratio for average stretch minimization satisfies $\rho(\Delta) < \Delta^2$.

There exists for this algorithm a sequence of jobs leading to starvation, and for which the maximum stretch can be as far as we want from the optimal maximum stretch.

The objectives are incompatible : demonstration (1)

- By contradiction, $\exists \Delta > 1$, $\exists \epsilon > 0$, $\exists$ algorithm $\mathcal{A}$ s.t.

$$\rho_{\mathcal{A}}(\Delta) < \Delta^2 - \epsilon.$$

The objectives are incompatible : demonstration (1)

- By contradiction, $\exists \Delta > 1$, $\exists \epsilon > 0$, $\exists$ algorithm $\mathcal{A}$ s.t.

$$\rho_{\mathcal{A}}(\Delta) < \Delta^2 - \epsilon.$$

- Adversary:

The objectives are incompatible : demonstration (1)

- By contradiction, $\exists \Delta > 1$, $\exists \epsilon > 0$, $\exists$ algorithm $\mathcal{A}$ s.t.

$$\rho_{\mathcal{A}}(\Delta) < \Delta^2 - \epsilon.$$

- Adversary:
  - Let $\alpha \in \mathbb{N}$ s.t. $\frac{1+\alpha\Delta}{1+\frac{\alpha}{\Delta}} > \Delta^2 - \frac{\epsilon}{2}$

The objectives are incompatible : demonstration (1)

- By contradiction, $\exists \Delta > 1$, $\exists \epsilon > 0$, $\exists$ algorithm $\mathcal{A}$ s.t.

$$\rho_{\mathcal{A}}(\Delta) < \Delta^2 - \epsilon.$$

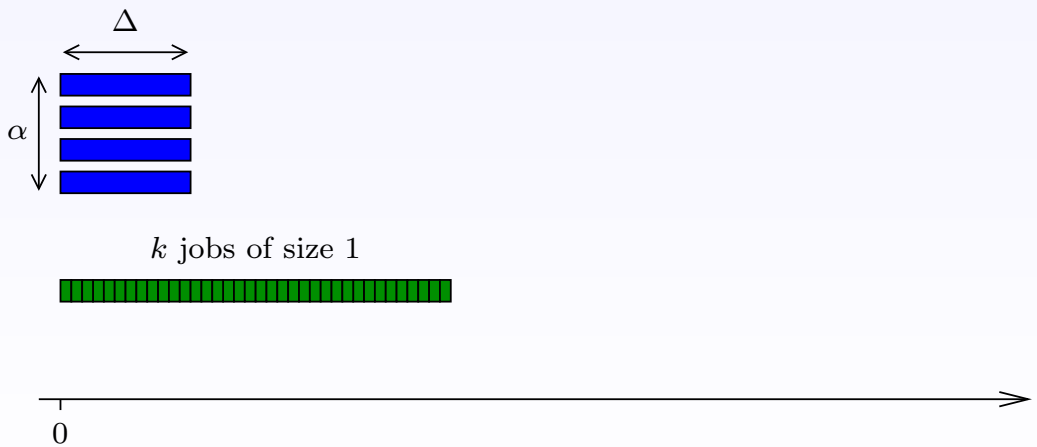
- Adversary:
  - Let $\alpha \in \mathbb{N}$ s.t. $\frac{1+\alpha\Delta}{1+\frac{\alpha}{\Delta}} > \Delta^2 - \frac{\epsilon}{2}$
  - At date 0 arrives α jobs of size Δ, J_1, ..., J_α.

The objectives are incompatible : demonstration (1)

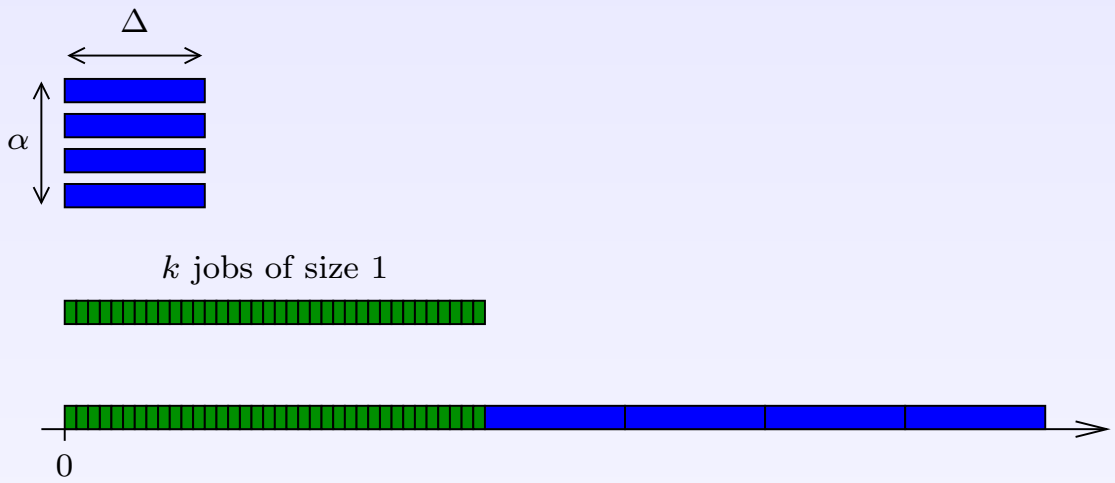
- By contradiction, $\exists \Delta > 1$, $\exists \epsilon > 0$, $\exists$ algorithm $\mathcal{A}$ s.t.

$$\rho_{\mathcal{A}}(\Delta) < \Delta^2 - \epsilon.$$

- Adversary:
  - Let $\alpha \in \mathbb{N}$ s.t. $\frac{1+\alpha\Delta}{1+\frac{\alpha}{\Delta}} > \Delta^2 - \frac{\epsilon}{2}$
  - At date 0 arrives α jobs of size Δ, J_1, ..., J_α.
  - $k \in \mathbb{N}$. $\forall t \in [0, k-1]$, job $J_{\alpha+t+1}$ of size 1 arrives at time t.



The objectives are incompatible : demonstration (2)



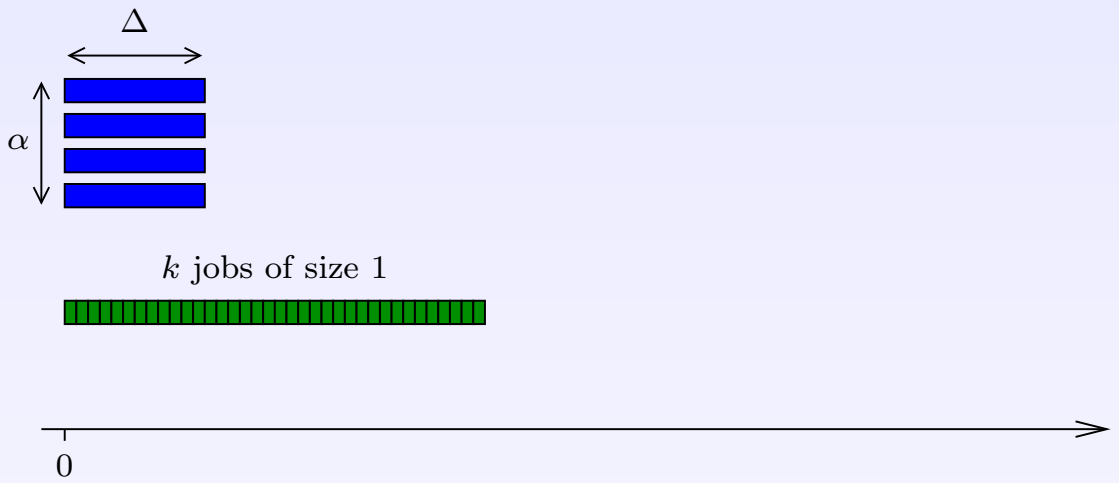
A possible schedule: each of the k unit jobs at its release date, and then the α Δ-units jobs.

$$\text{sum-stretch} = k\times 1+\frac{k+\Delta}{\Delta}+\ldots+\frac{k+\alpha\Delta}{\Delta} = \frac{\alpha(\alpha+1)}{2}+k\left(1+\frac{\alpha}{\Delta}\right).$$

$$\text{max-stretch} = \alpha+\frac{k}{\Delta}.$$

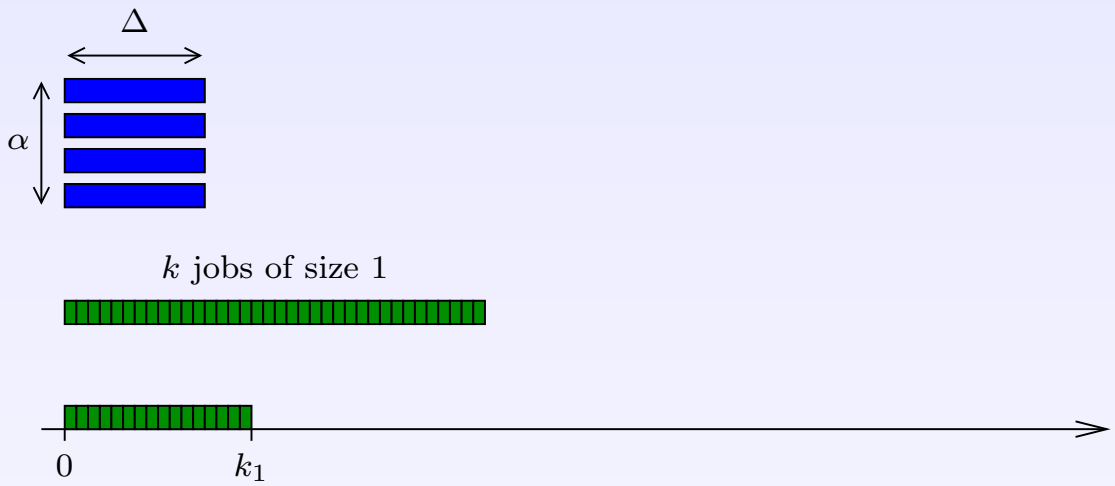
May not be optimal (just an upper-bound) and induces starvation.

The objectives are incompatible : demonstration (2)



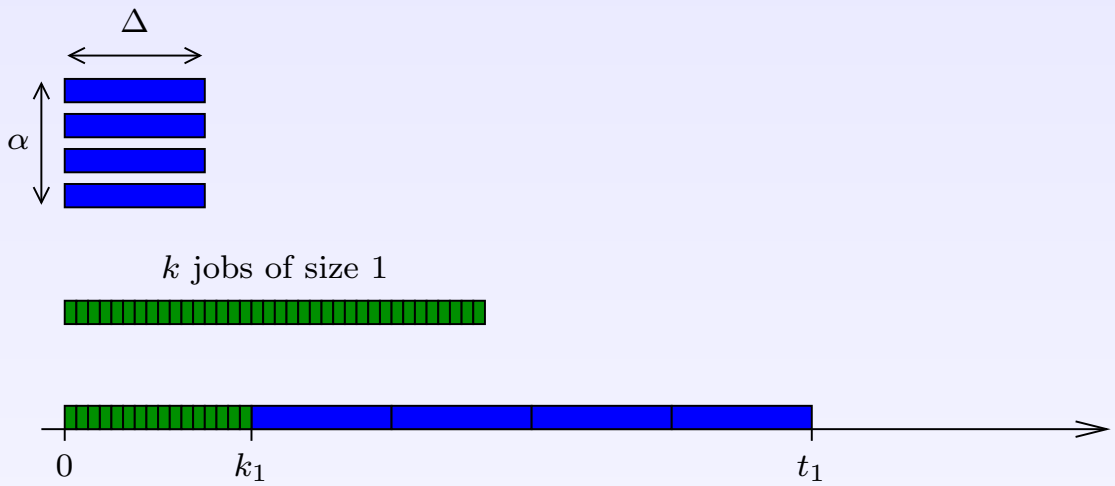
Otherwise, at a date $t_1 < k + \alpha\Delta$ the Δ size jobs are all completed.

The objectives are incompatible : demonstration (2)



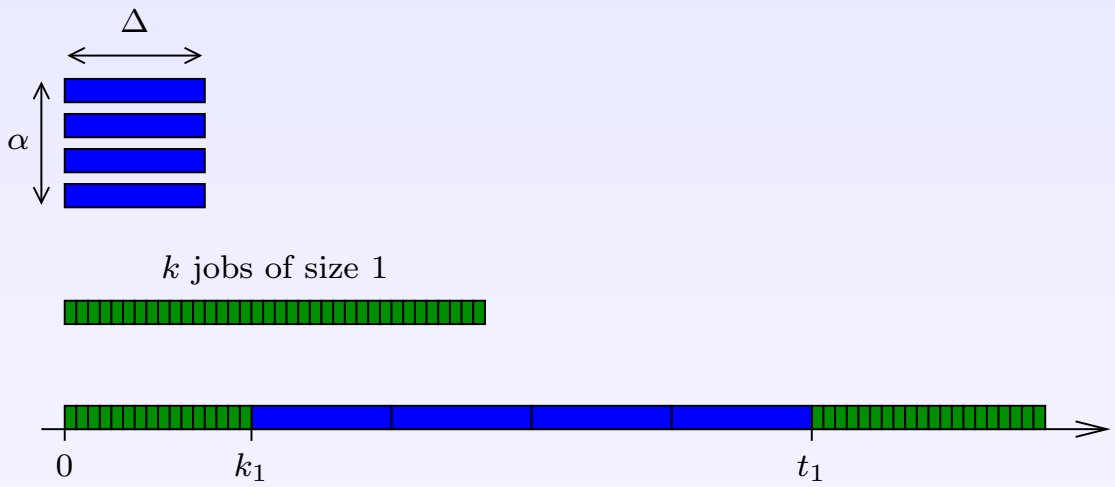
Otherwise, at a date $t_1 < k + \alpha\Delta$ the Δ size jobs are all completed. k_1 unit size jobs were completed before t_1.

The objectives are incompatible : demonstration (2)



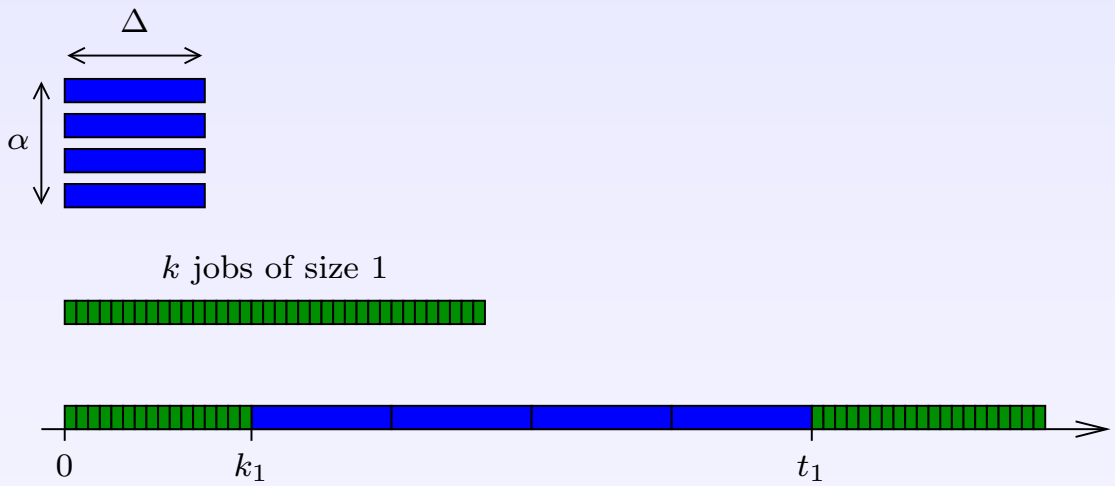
Otherwise, at a date $t_1 < k + \alpha\Delta$ the Δ size jobs are all completed. k_1 unit size jobs were completed before t_1.

The objectives are incompatible : demonstration (2)



Otherwise, at a date $t_1 < k + \alpha\Delta$ the Δ size jobs are all completed. k_1 unit size jobs were completed before t_1.

The objectives are incompatible : demonstration (2)



Otherwise, at a date $t_1 < k + \alpha\Delta$ the Δ size jobs are all completed.
k_1 unit size jobs were completed before t_1.
Best achievable sum-stretch:

$$k_1 \times 1 + \frac{k_1 + \Delta}{\Delta} + \ldots + \frac{k_1 + \alpha\Delta}{\Delta} + (k - k_1)(1 + \alpha\Delta) =$$
$$\left(\frac{\alpha(\alpha+1)}{2} + \frac{\alpha k_1}{\Delta}\right) + k_1 + (k - k_1)(1 + \alpha\Delta).$$

The objectives are incompatible : demonstration (3)

Hypothesis: $\mathcal{A}$ is $\rho_{\mathcal{A}}(\Delta)$-competitive

$$\left(\frac{\alpha(\alpha+1)}{2}+\frac{\alpha k_1}{\Delta}\right)+k_1+(k-k_1)(1+\alpha\Delta)$$
$$\leq \rho_{\mathcal{A}}(\Delta)\left(\frac{\alpha(\alpha+1)}{2}+k\left(1+\frac{\alpha}{\Delta}\right)\right) \quad \Leftrightarrow$$

$$-\alpha\Delta k_1+\frac{\alpha(\alpha+1)}{2}(1-\rho_{\mathcal{A}}(\Delta))+\frac{\alpha k_1}{\Delta}$$
$$\leq k\left(\rho_{\mathcal{A}}(\Delta)\left(1+\frac{\alpha}{\Delta}\right)-(1+\alpha\Delta)\right).$$

Must hold for any k, thus:

$$\left(\rho_{\mathcal{A}}(\Delta)\left(1+\frac{\alpha}{\Delta}\right)-(1+\alpha\Delta)\right)\geq 0 \Rightarrow \Delta^2-\epsilon > \frac{1+\alpha\Delta}{1+\frac{\alpha}{\Delta}}.$$

FIFO is Δ^2-competitive for sum-stretch

Theorem

First come, first served is:

- Δ^2*-competitive for the online minimization of sum-stretch,*
- Δ*-competitive for the online minimization of max-stretch,*
- Δ*-competitive for the online minimization of sum-flow, and*
- *optimal for the online minimization of max-flow.*

Outline

Minimizing the average or maximum flow

- Maximum flow is minimized by the policy *First come, first serve*
- Average flow is minimized by the *Shortest Remaining Processing Time* (SRPT) policy.

Minimizing the average stretch

- Problem complexity: open.

Minimizing the average stretch

- Problem complexity: open.
- Polynomial Time Approximation Schemes (PTAS).

Minimizing the average stretch

- Problem complexity: open.
- Polynomial Time Approximation Schemes (PTAS).
- The competitive ratio of any online algorithm is at least: 1.19485.

Minimizing the average stretch

- Problem complexity: open.
- Polynomial Time Approximation Schemes (PTAS).
- The competitive ratio of any online algorithm is at least: 1.19485.
- *Shortest Remaining Processing Time* is 2-competitive.

Minimizing the average stretch

- Problem complexity: open.
- Polynomial Time Approximation Schemes (PTAS).
- The competitive ratio of any online algorithm is at least: 1.19485.
- *Shortest Remaining Processing Time* is 2-competitive.
- Obvious improvement: *Shortest Weighted Remaining Processing Time*.
  At any time t, SWRPT schedules the job J_j minimizing $p_j \rho_t(j)$.
  SWRPT is *at best* 2-competitive.

Minimizing the average stretch

- Problem complexity: open.
- Polynomial Time Approximation Schemes (PTAS).
- The competitive ratio of any online algorithm is at least: 1.19485.
- *Shortest Remaining Processing Time* is 2-competitive.
- Obvious improvement: *Shortest Weighted Remaining Processing Time*.
  At any time t, SWRPT schedules the job J_j minimizing $p_j \rho_t(j)$.
  SWRPT is *at best* 2-competitive.

The off-line case looks difficult... but there exists a simple guaranteed online algorithm.

Outline

Off-line framework: the rules of the game

- We know in advance the release dates of jobs, and their size (*off-line* framework).
- We want to minimize the maximum weighted flow (for any weights).
- Jobs are divisible.

Existence of a schedule having a given max-stretch

Existence of a schedule of max-stretch $\mathcal{S}$:

For each job J_j, $\quad \frac{C_j - r_j}{p_j} \leq \mathcal{S}$

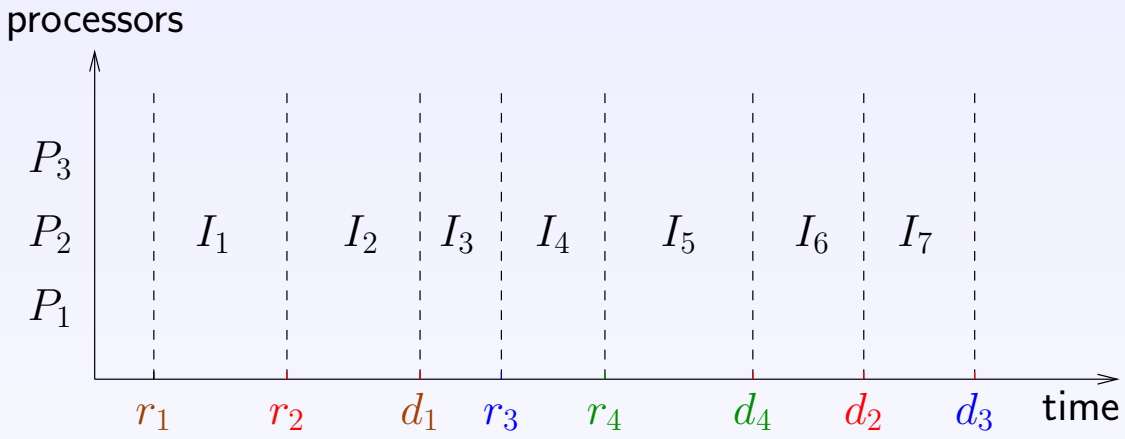
Equivalent to a scheduling problem with deadlines where $d_j(\mathcal{S}) = r_j + p_j \times \mathcal{S}$

Deadline scheduling: definition

- Each job J_j has a *deadline* d_j (and always a release/arrival date r_j).
- Question: is there a schedule which completes each job J_j during its existence interval $[r_j, d_j]$?

Deadline scheduling: notations

Set of release dates and deadlines: $\{r_1, ..., r_n, d_1, ..., d_n\}$.



These dates, when sorted, define a set of n_{int} intervals $I_1, ..., I_{n_{\text{int}}}$, with $1 \leq n_{\text{int}} \leq 2n - 1$.

$I_t = [\inf I_t, \sup I_t[$

$\alpha_{i,j}^{(t)}$: fraction of J_j processed by M_i during the interval I_t.

Deadline scheduling: solution

1. *Release dates*:

$$\forall i, \forall j, \forall t, \quad r_j \geq \sup I_t \Rightarrow \alpha_{i,j}^{(t)} = 0$$

Deadline scheduling: solution

1. *Release dates*:

$$\forall i, \forall j, \forall t, \quad r_j \geq \sup I_t \Rightarrow \alpha_{i,j}^{(t)} = 0$$

2. *Deadlines*:

$$\forall i, \forall j, \forall t, \quad d_j \leq \inf I_t \Rightarrow \alpha_{i,j}^{(t)} = 0$$

Deadline scheduling: solution

1. *Release dates*:

$$\forall i, \forall j, \forall t, \quad r_j \geq \sup I_t \Rightarrow \alpha_{i,j}^{(t)} = 0$$

2. *Deadlines*:

$$\forall i, \forall j, \forall t, \quad d_j \leq \inf I_t \Rightarrow \alpha_{i,j}^{(t)} = 0$$

3. *Resource constraints*:

$$\forall t, \forall i, \quad \sum_j \alpha_{i,j}^{(t)}.c_{i,j} \leq \sup I_t - \inf I_t$$

Deadline scheduling: solution

1. *Release dates*:

$$\forall i, \forall j, \forall t, \quad r_j \geq \sup I_t \Rightarrow \alpha_{i,j}^{(t)} = 0$$

2. *Deadlines*:

$$\forall i, \forall j, \forall t, \quad d_j \leq \inf I_t \Rightarrow \alpha_{i,j}^{(t)} = 0$$

3. *Resource constraints*:

$$\forall t, \forall i, \quad \sum_j \alpha_{i,j}^{(t)}.c_{i,j} \leq \sup I_t - \inf I_t$$

4. *Jobs' completion*:

$$\forall j, \quad \sum_t \sum_i \alpha_{i,j}^{(t)} = 1$$

Maximum weighted flow and deadlines

- Is there a schedule whose maximum weighted flow is no greater than $\mathcal{F}$?

Maximum weighted flow and deadlines

- Is there a schedule whose maximum weighted flow is no greater than $\mathcal{F}$?
- $\max_j w_j(C_j - r_j) \leq \mathcal{F} \quad \Leftrightarrow \quad \forall j, w_j(C_j - r_j) \leq \mathcal{F} \quad \Leftrightarrow$
  $\forall j, C_j \leq r_j + \mathcal{F}/w_j.$

Maximum weighted flow and deadlines

- Is there a schedule whose maximum weighted flow is no greater than $\mathcal{F}$?
- $\max_j w_j(C_j - r_j) \leq \mathcal{F} \quad \Leftrightarrow \quad \forall j, w_j(C_j - r_j) \leq \mathcal{F} \quad \Leftrightarrow$
  $\forall j, C_j \leq r_j + \mathcal{F}/w_j$.
- Equivalent to solving the scheduling problem with deadlines where $d_j(\mathcal{F}) = r_j + \mathcal{F}/w_j$ is job J_j's *deadline*.

Maximum weighted flow and deadlines

- Is there a schedule whose maximum weighted flow is no greater than $\mathcal{F}$?
- $\max_j w_j(C_j - r_j) \leq \mathcal{F} \quad \Leftrightarrow \quad \forall j, w_j(C_j - r_j) \leq \mathcal{F} \quad \Leftrightarrow$
  $\forall j, C_j \leq r_j + \mathcal{F}/w_j$.
- Equivalent to solving the scheduling problem with deadlines where $d_j(\mathcal{F}) = r_j + \mathcal{F}/w_j$ is job J_j's *deadline*.
- We only need to do a binary search over $\mathcal{F}$!

Maximum weighted flow and deadlines

- Is there a schedule whose maximum weighted flow is no greater than $\mathcal{F}$?
- $\max_j w_j(C_j - r_j) \leq \mathcal{F} \quad \Leftrightarrow \quad \forall j, w_j(C_j - r_j) \leq \mathcal{F} \quad \Leftrightarrow$
  $\forall j, C_j \leq r_j + \mathcal{F}/w_j$.
- Equivalent to solving the scheduling problem with deadlines where $d_j(\mathcal{F}) = r_j + \mathcal{F}/w_j$ is job J_j's *deadline*.
- We only need to do a binary search over $\mathcal{F}$!

  We would rather have an algorithm which terminates...

Solving on an interval of objectives (1)

- Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be two values, $\mathcal{F}_1 < \mathcal{F}_2$.

  We assume that the relative order of release dates and deadlines $r_1, \ldots, r_n, d_1(\mathcal{F}), \ldots, d_n(\mathcal{F})$, is independent of $\mathcal{F} \in]\mathcal{F}_1; \mathcal{F}_2[$.

Solving on an interval of objectives (1)

- Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be two values, $\mathcal{F}_1 < \mathcal{F}_2$.

  We assume that the relative order of release dates and deadlines $r_1, \ldots, r_n, d_1(\mathcal{F}), \ldots, d_n(\mathcal{F})$, is independent of $\mathcal{F} \in]\mathcal{F}_1; \mathcal{F}_2[$.

- We define from the set $r_1, \ldots, r_n, d_1(\mathcal{F}), \ldots, d_n(\mathcal{F})$ some intervals, as we previously did.

  The interval extremities are affine functions in $\mathcal{F}$.

Solving on an interval of objectives (1)

- Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be two values, $\mathcal{F}_1 < \mathcal{F}_2$.

  We assume that the relative order of release dates and deadlines $r_1, \ldots, r_n, d_1(\mathcal{F}), \ldots, d_n(\mathcal{F})$, is independent of $\mathcal{F} \in]\mathcal{F}_1; \mathcal{F}_2[$.

- We define from the set $r_1, \ldots, r_n, d_1(\mathcal{F}), \ldots, d_n(\mathcal{F})$ some intervals, as we previously did.

  The interval extremities are affine functions in $\mathcal{F}$.

- We adapt the previous system.

Solving on an interval of objectives (2)

$$\forall i, \forall j, \forall t, \quad r_j \geq \sup I_t(\mathcal{F}) \Rightarrow \alpha_{i,j}^{(t)} = 0$$

$$\forall i, \forall j, \forall t, \quad d_j(\mathcal{F}) \leq \inf I_t(\mathcal{F}) \Rightarrow \alpha_{i,j}^{(t)} = 0$$

$$\forall t, \forall i, \quad \sum_j \alpha_{i,j}^{(t)}.c_{i,j} \leq \sup I_t(\mathcal{F}) - \inf I_t(\mathcal{F})$$

$$\forall j, \quad \sum_t \sum_i \alpha_{i,j}^{(t)} = 1$$

Solving on an interval of objectives (2)

$$\forall i, \forall j, \forall t, \quad r_j \geq \sup I_t(\mathcal{F}) \Rightarrow \alpha_{i,j}^{(t)} = 0$$

$$\forall i, \forall j, \forall t, \quad d_j(\mathcal{F}) \leq \inf I_t(\mathcal{F}) \Rightarrow \alpha_{i,j}^{(t)} = 0$$

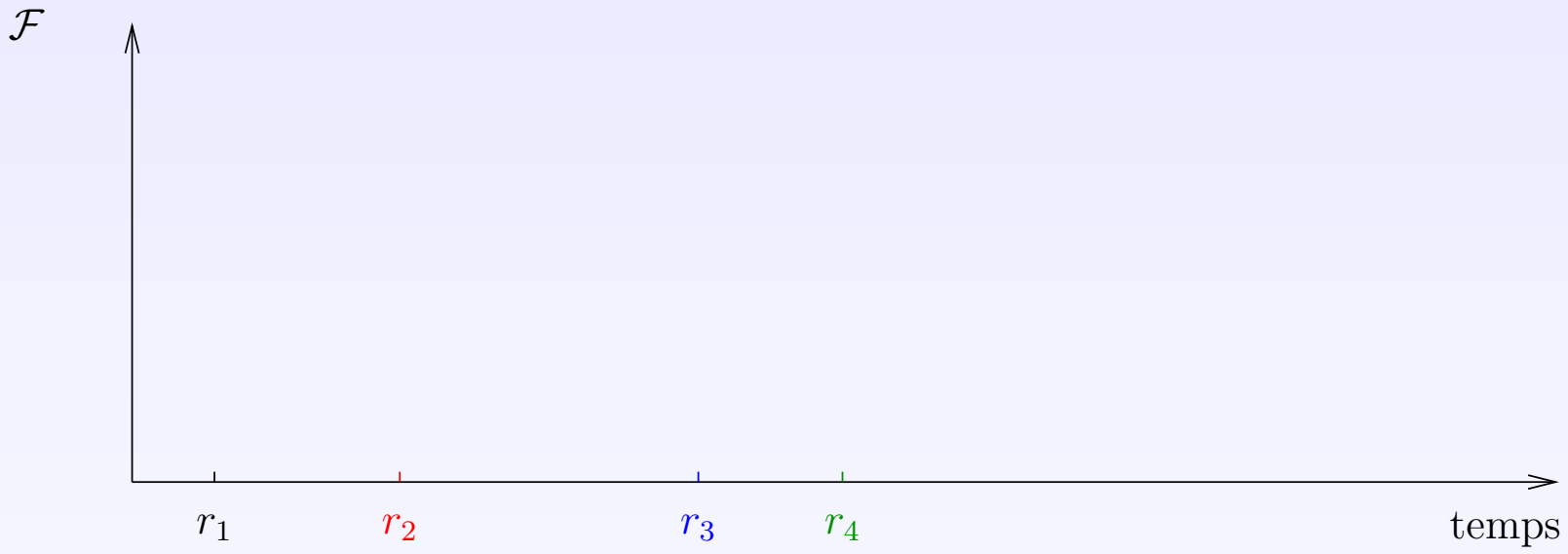
$$\forall t, \forall i, \quad \sum_j \alpha_{i,j}^{(t)}.c_{i,j} \leq \sup I_t(\mathcal{F}) - \inf I_t(\mathcal{F})$$

$$\forall j, \quad \sum_t \sum_i \alpha_{i,j}^{(t)} = 1$$

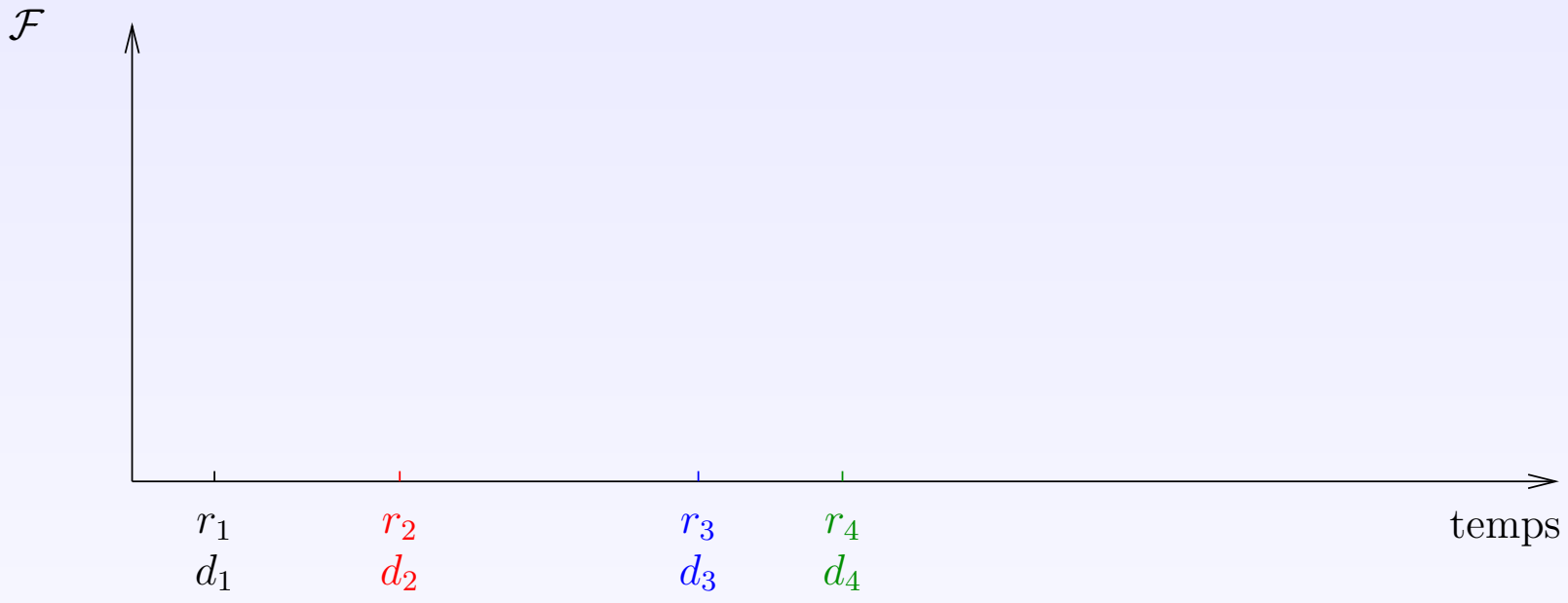
$$\mathcal{F}_1 \leq \mathcal{F} \leq \mathcal{F}_2$$

$$\min \mathcal{F}$$

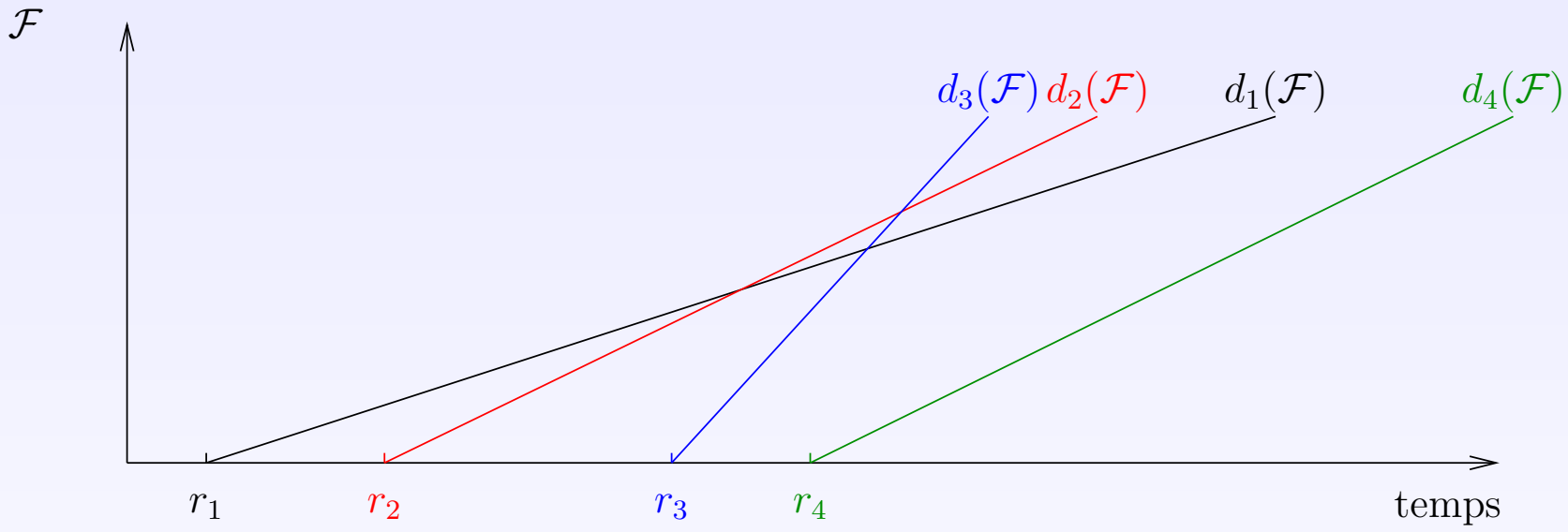
Relative ordering of the epochal times



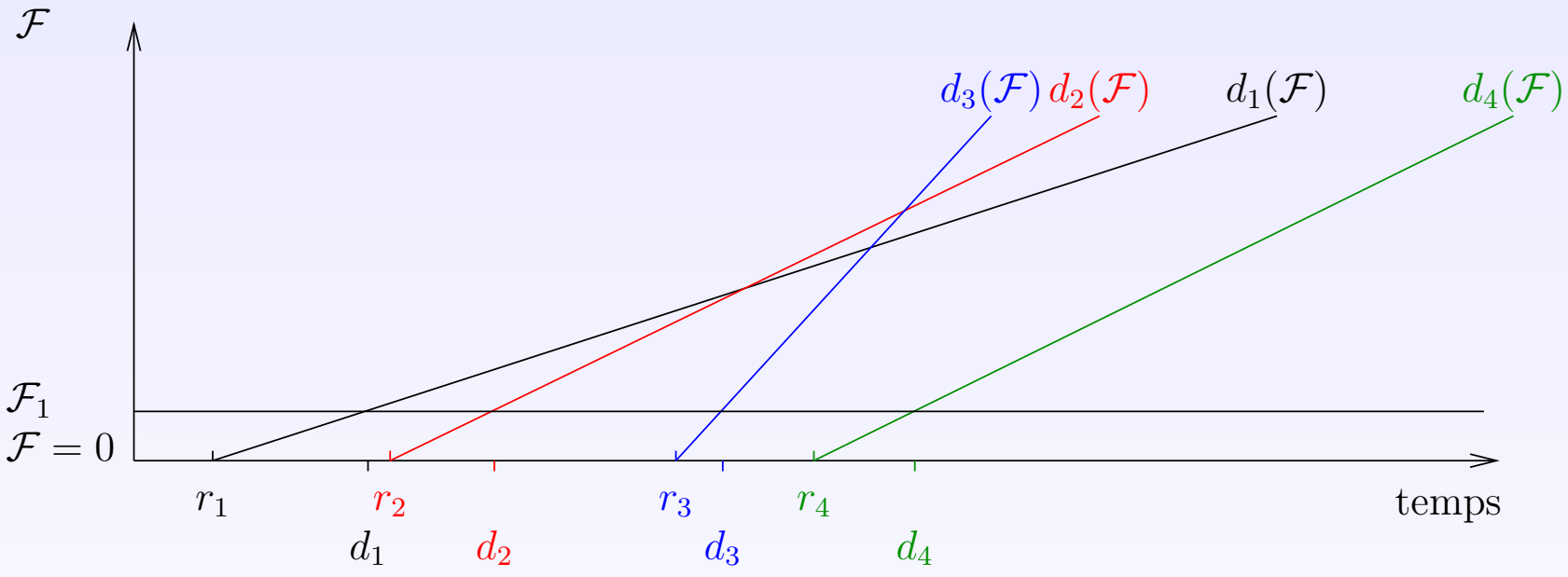
Relative ordering of the epochal times



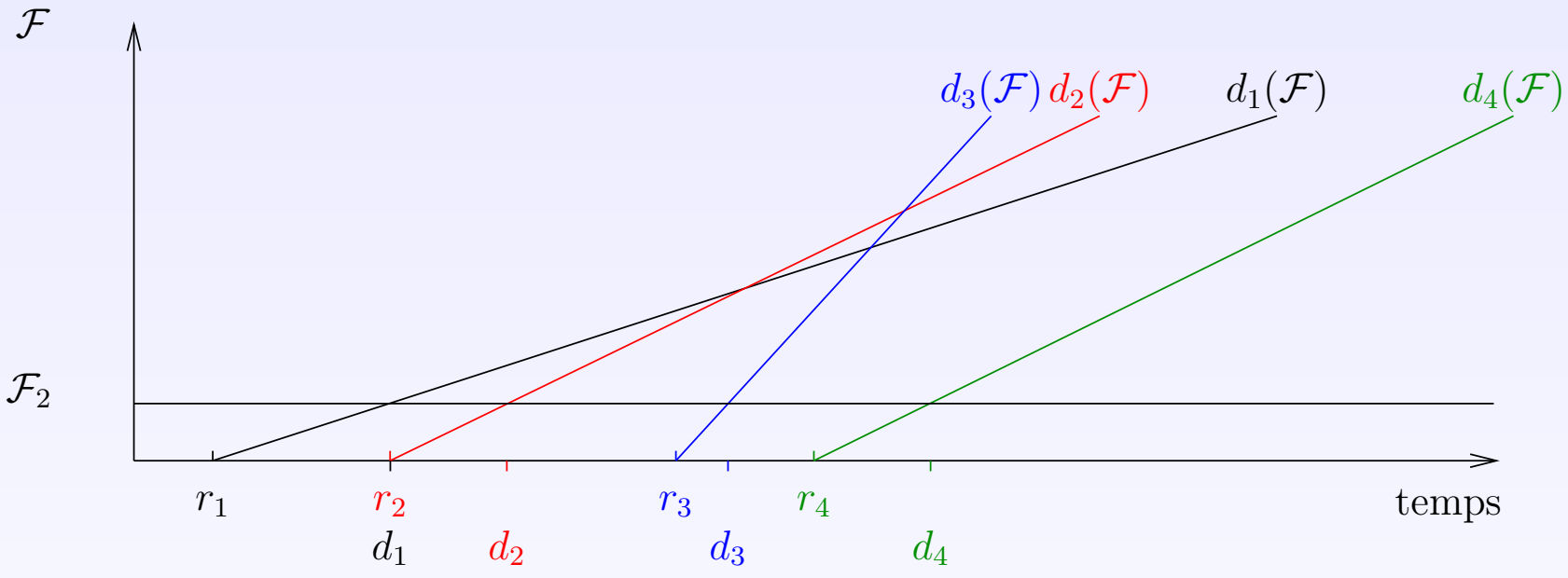
Relative ordering of the epochal times



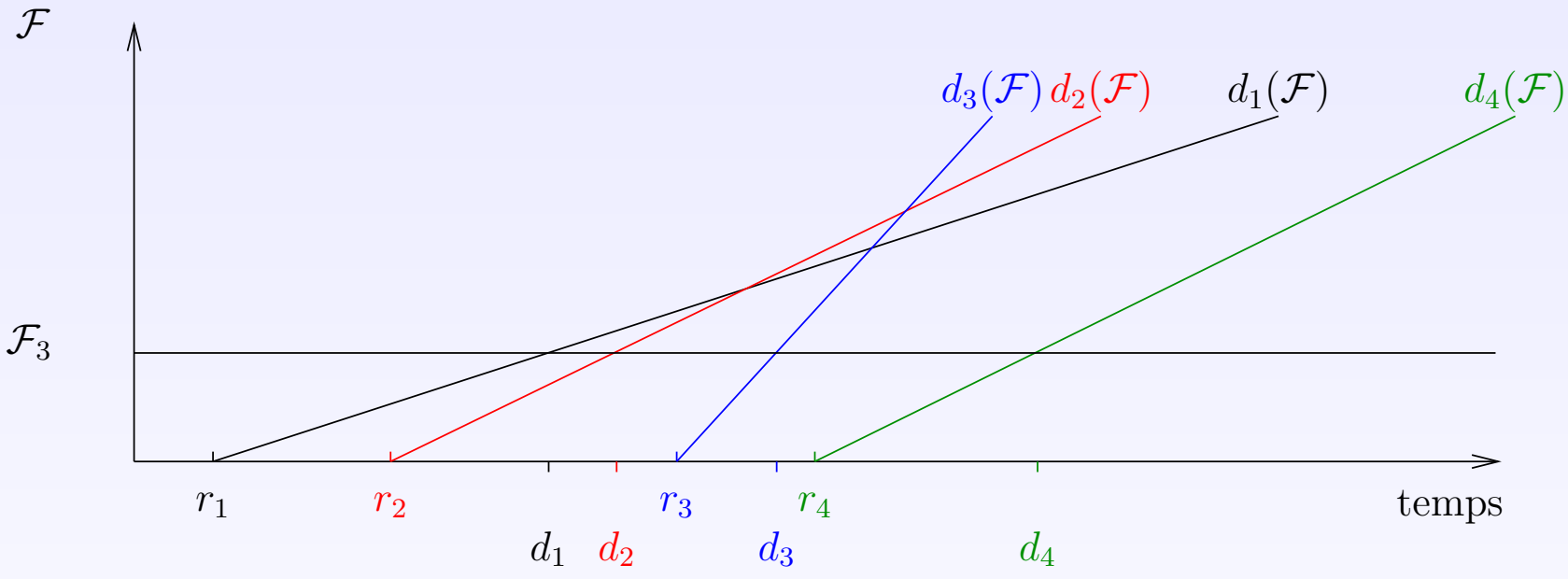
Relative ordering of the epochal times



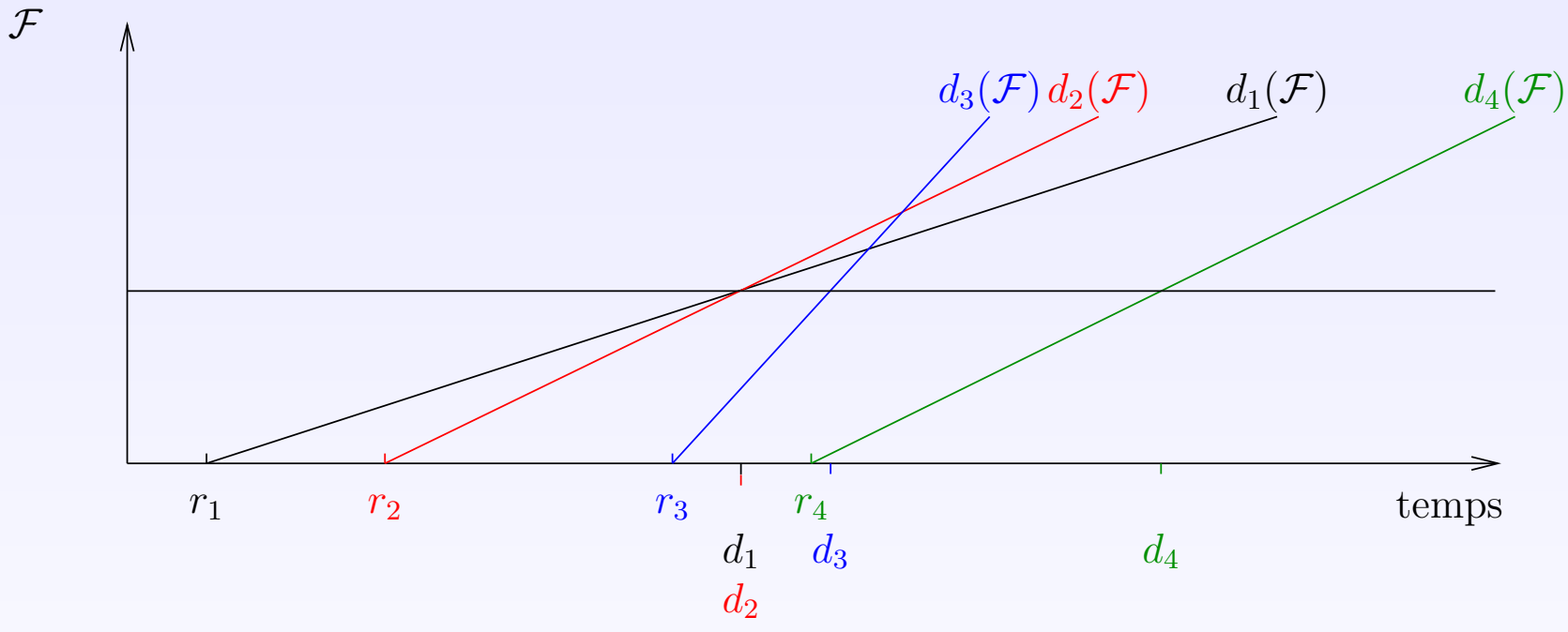
Relative ordering of the epochal times



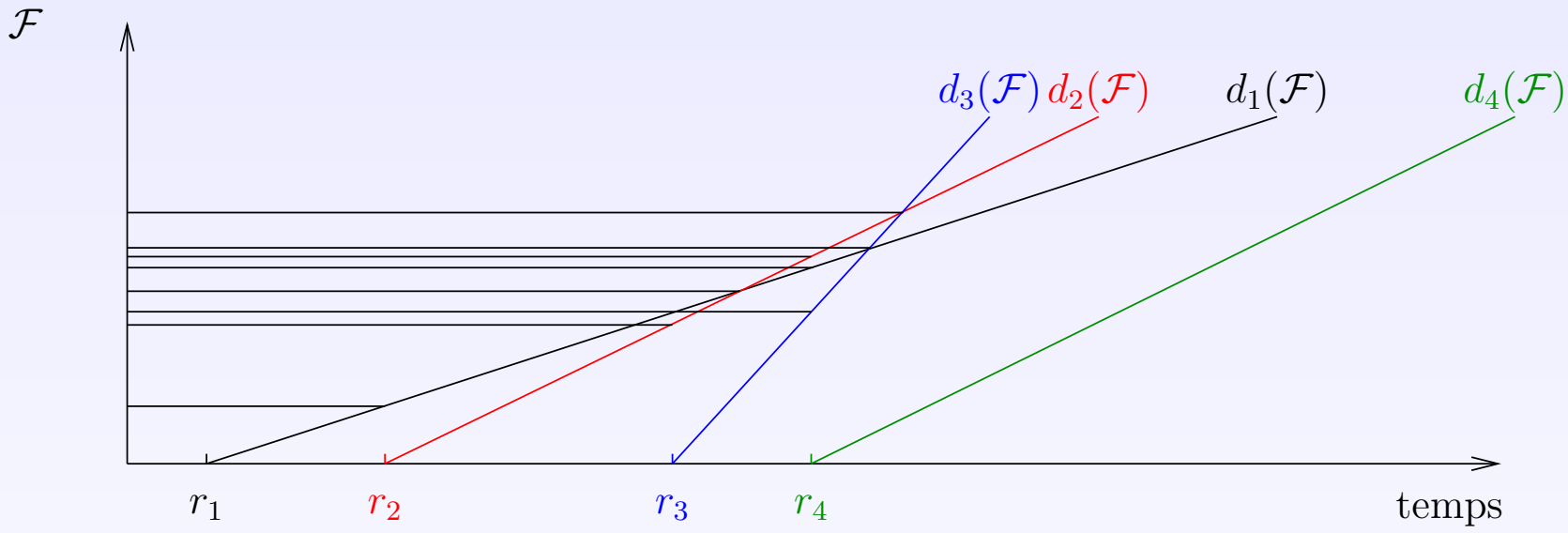
Relative ordering of the epochal times



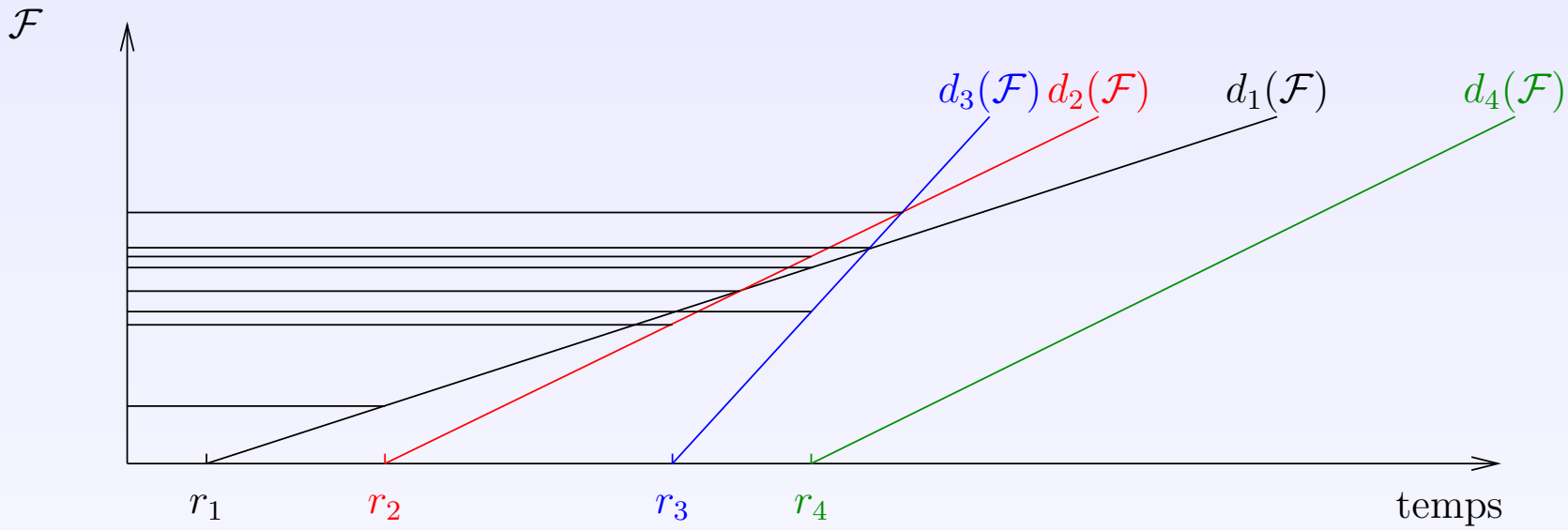
Relative ordering of the epochal times



Relative ordering of the epochal times



Relative ordering of the epochal times



There are at most $n_{\mathsf{q}} \leq n^2 - n$ possible intersections.

Resolution

- We compute the $n_{\mathsf{q}} \leq n^2 - n$ possible intersections.

Resolution

- We compute the $n_{\mathsf{q}} \leq n^2 - n$ possible intersections.
- Let $\mathcal{F}_1, \mathcal{F}_2, ..., \mathcal{F}_{n_{\mathsf{q}}}$ be the peculiar values of the objective.

  We perform a binary search on the set of the peculiar values $\mathcal{F}_i$, each time searching if there is a solution in the interval $[\mathcal{F}_i, \mathcal{F}_{i+1}]$.

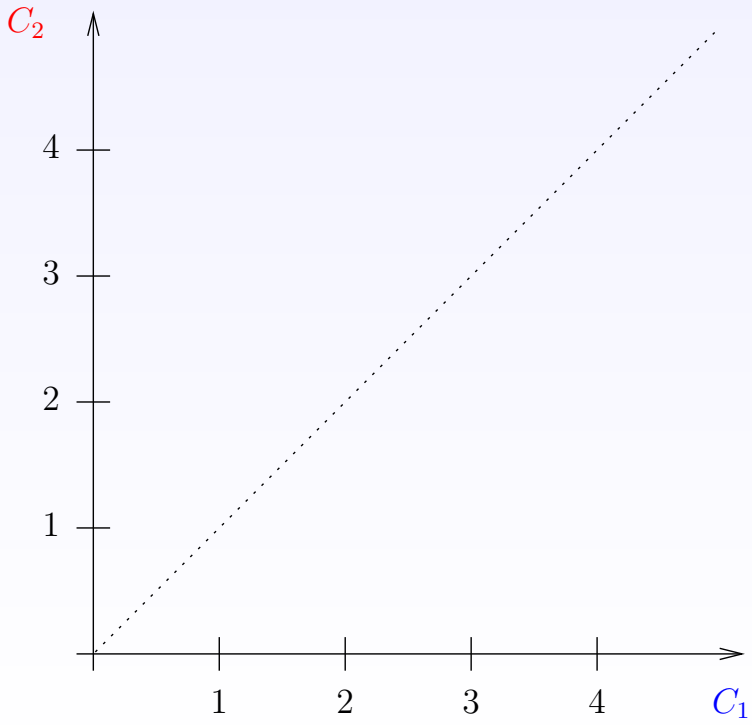
Resolution

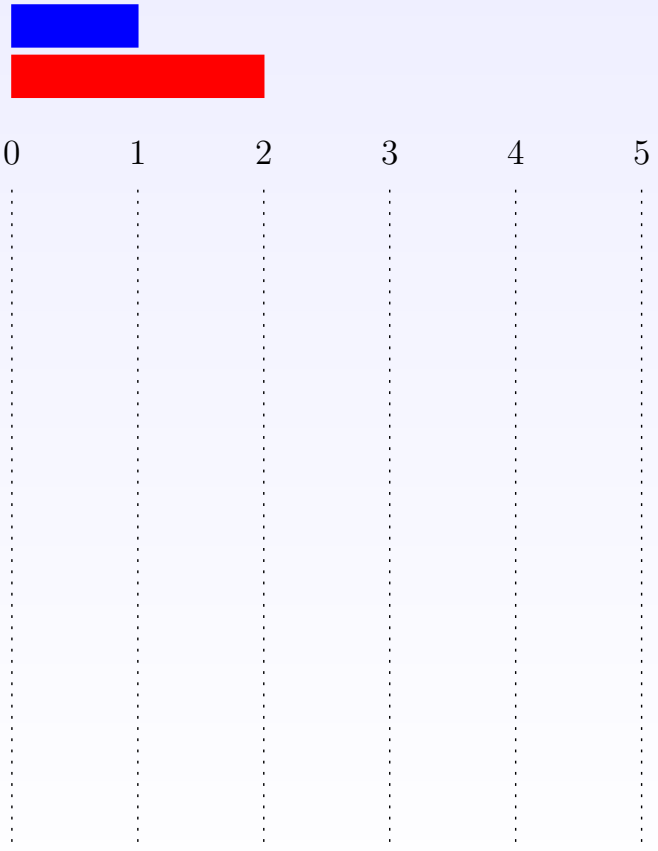
- We compute the $n_{\mathsf{q}} \leq n^2 - n$ possible intersections.
- Let $\mathcal{F}_1, \mathcal{F}_2, ..., \mathcal{F}_{n_{\mathsf{q}}}$ be the peculiar values of the objective.

  We perform a binary search on the set of the peculiar values $\mathcal{F}_i$, each time searching if there is a solution in the interval $[\mathcal{F}_i, \mathcal{F}_{i+1}]$.
- Polynomial-time algorithm.

Pareto optimality

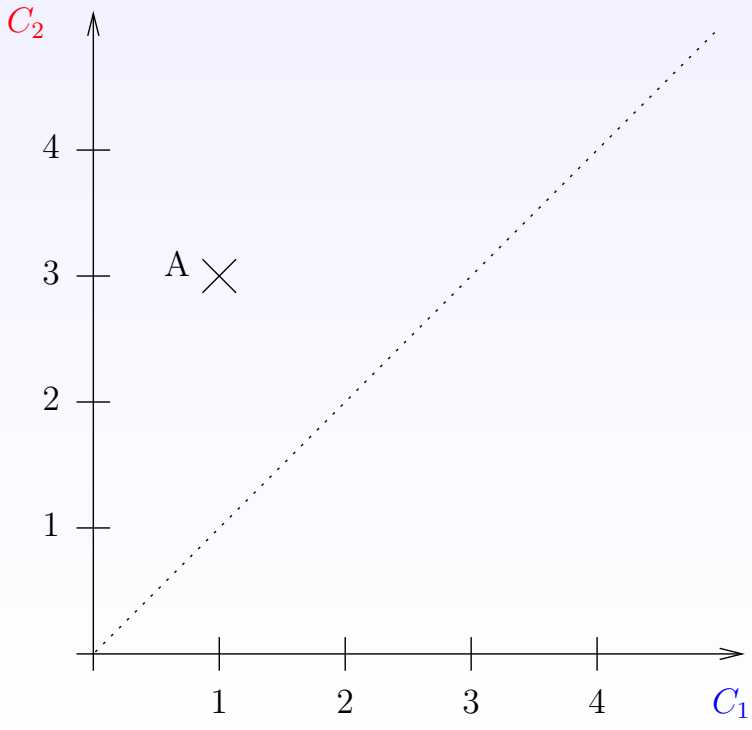
The cost (here the completion time) of a user can not be improved without degrading the cost of another user.

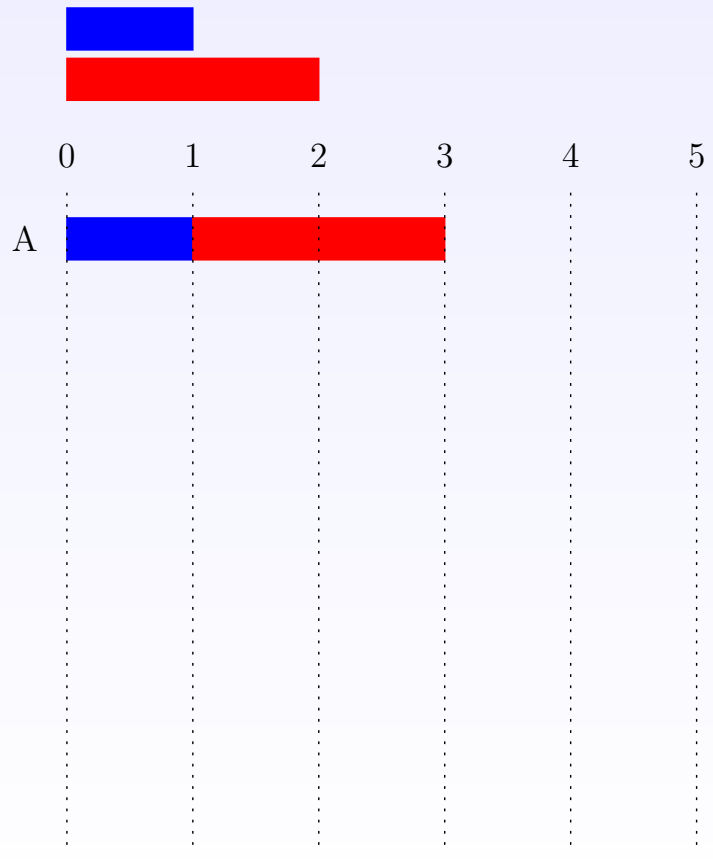




Pareto optimality

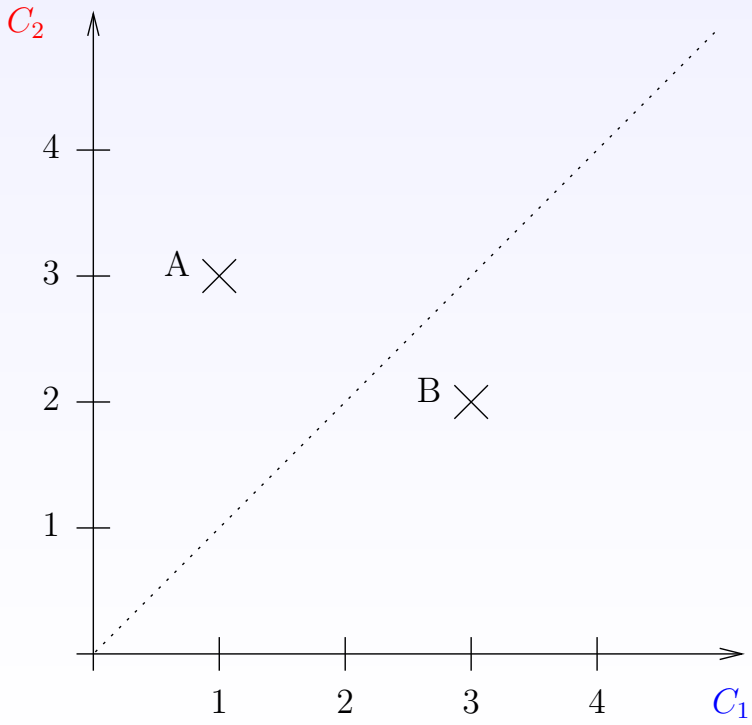
The cost (here the completion time) of a user can not be improved without degrading the cost of another user.

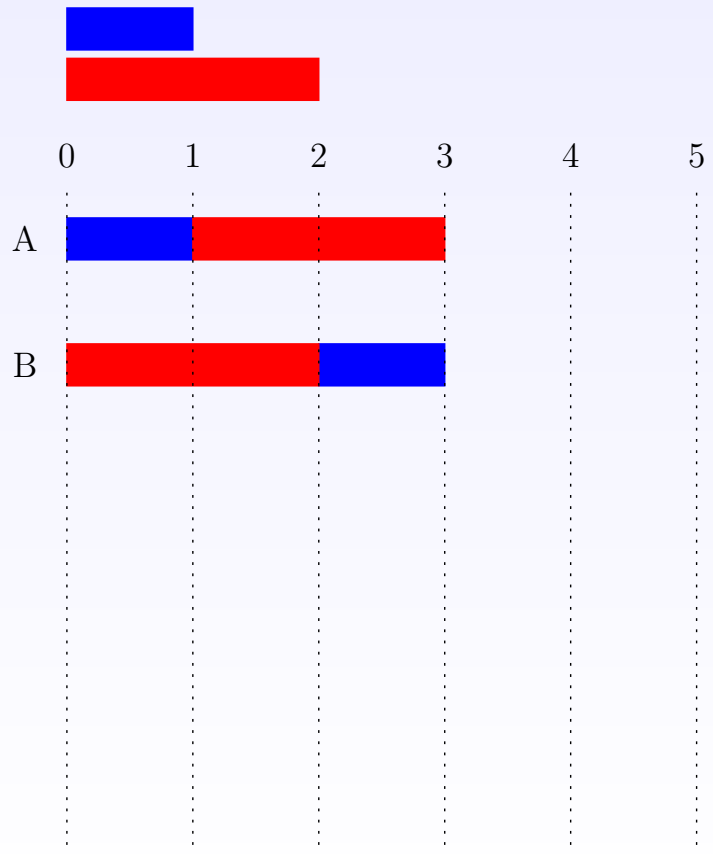




Pareto optimality

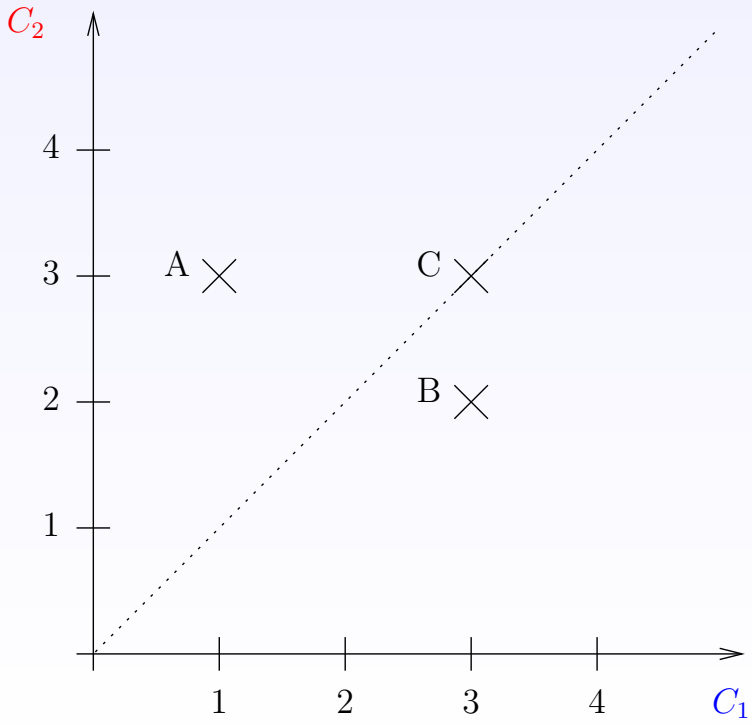
The cost (here the completion time) of a user can not be improved without degrading the cost of another user.

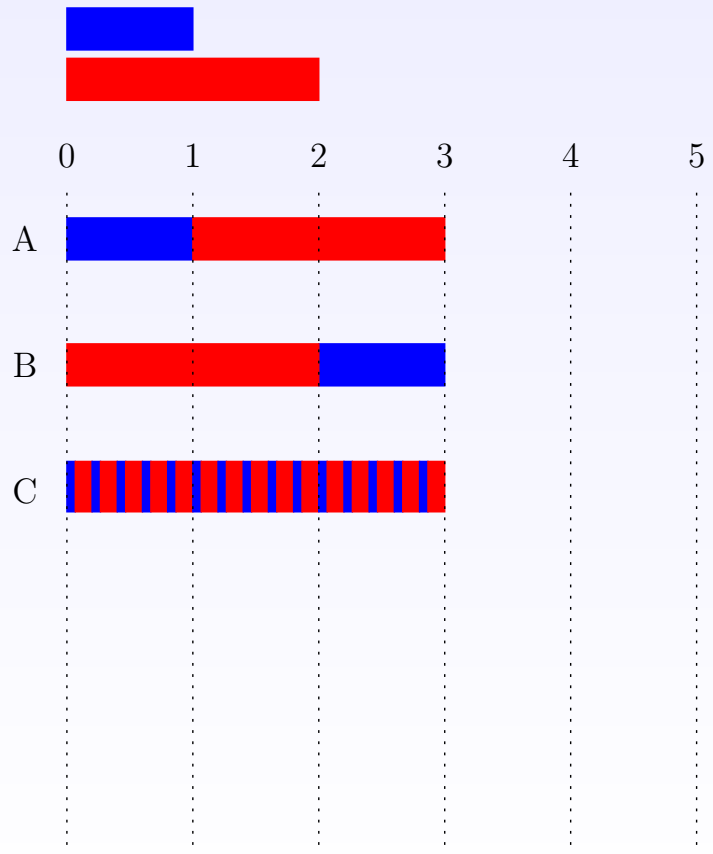




Pareto optimality

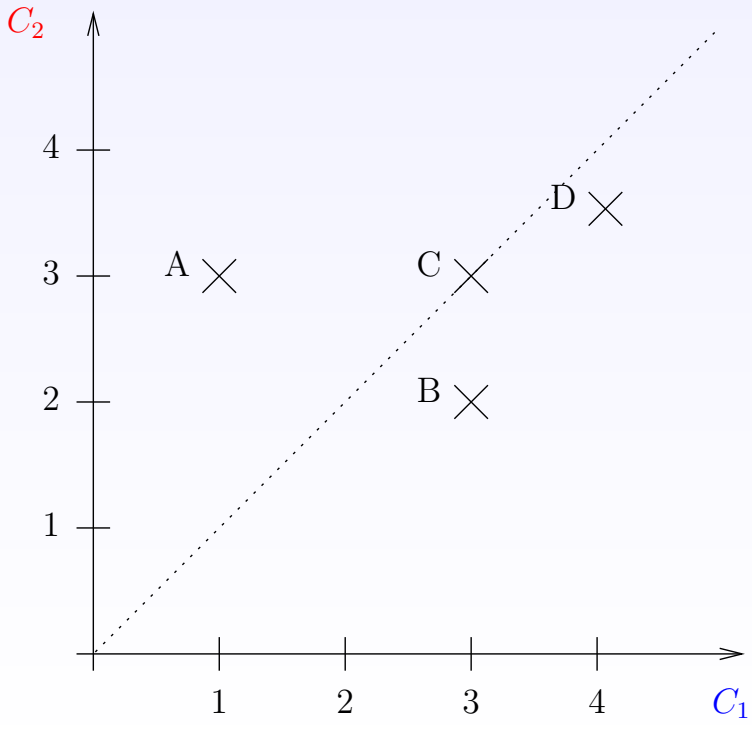
The cost (here the completion time) of a user can not be improved without degrading the cost of another user.

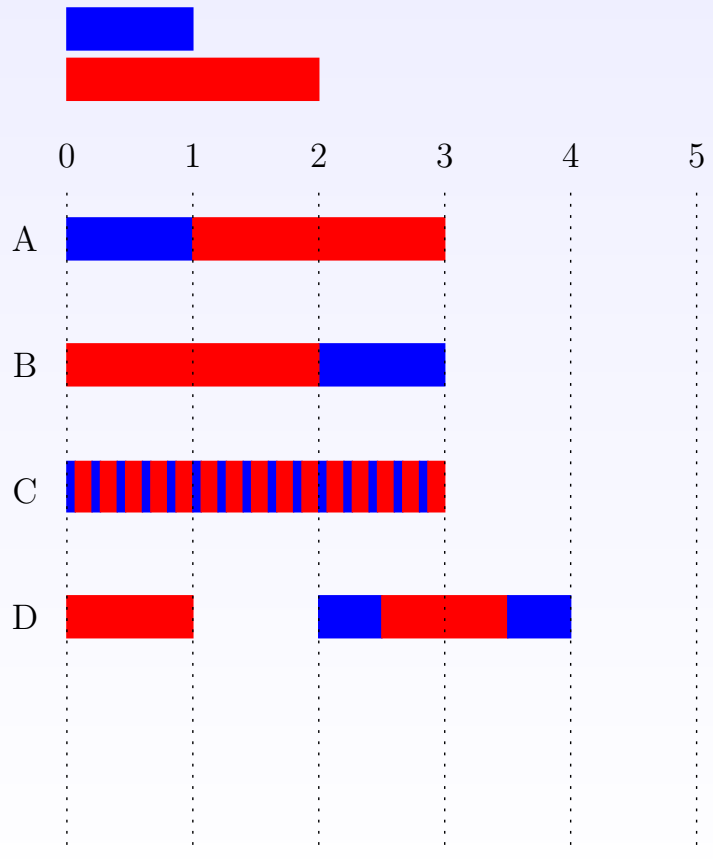




Pareto optimality

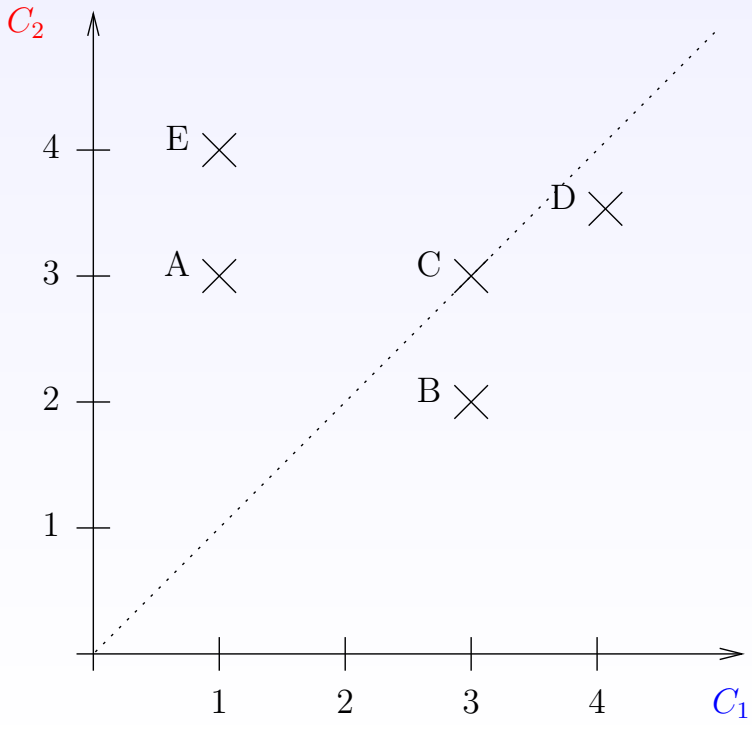
The cost (here the completion time) of a user can not be improved without degrading the cost of another user.

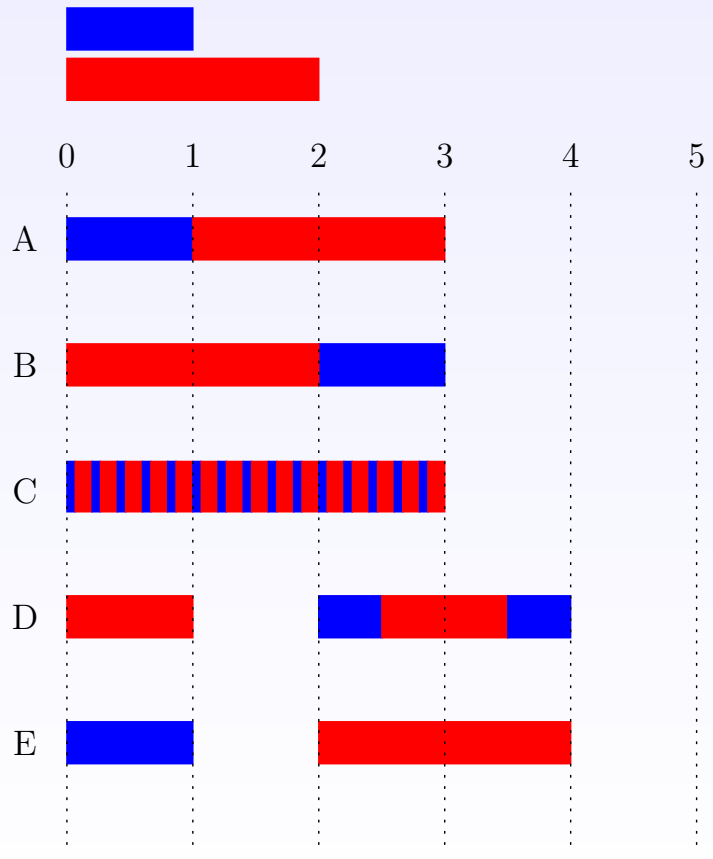




Pareto optimality

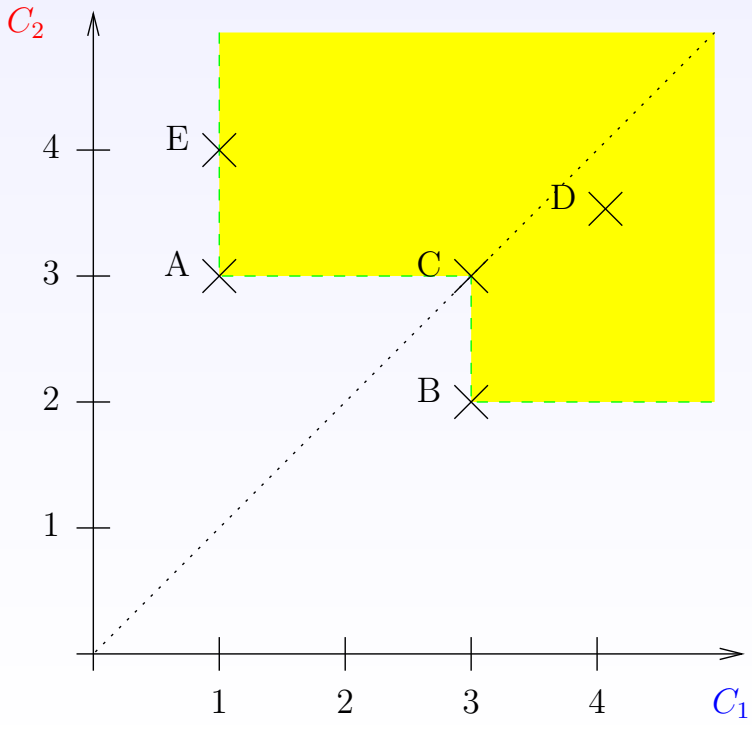
The cost (here the completion time) of a user can not be improved without degrading the cost of another user.

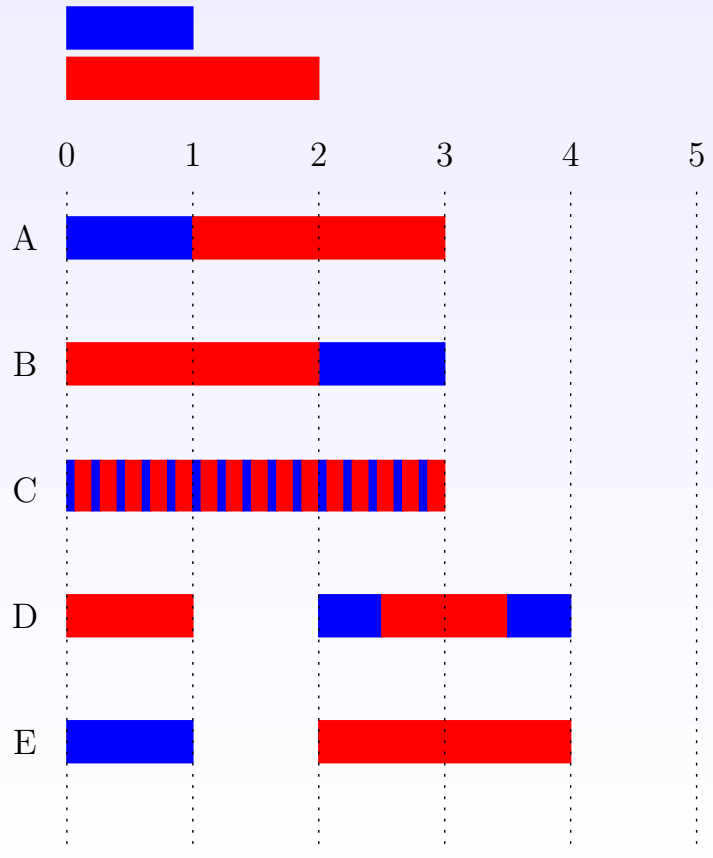




Pareto optimality

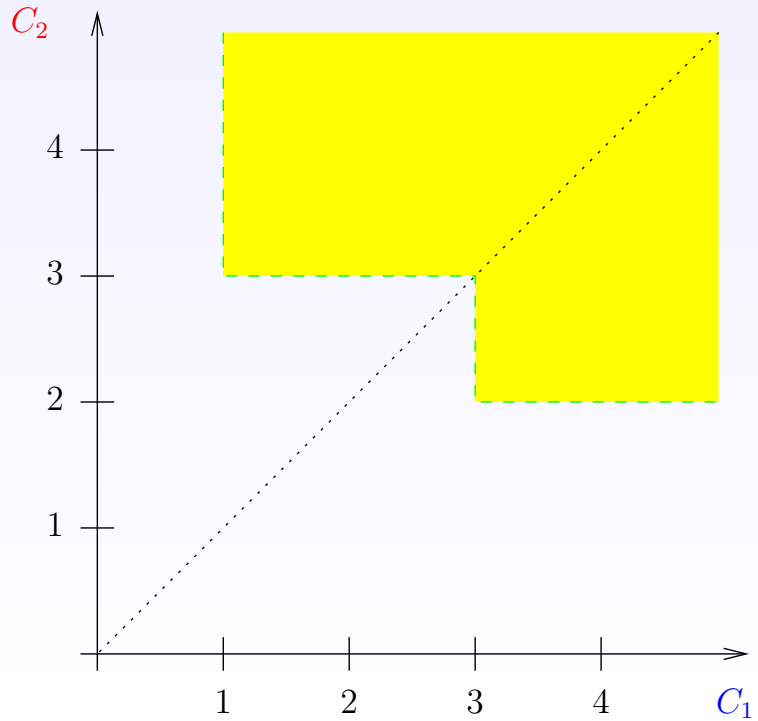
The cost (here the completion time) of a user can not be improved without degrading the cost of another user.





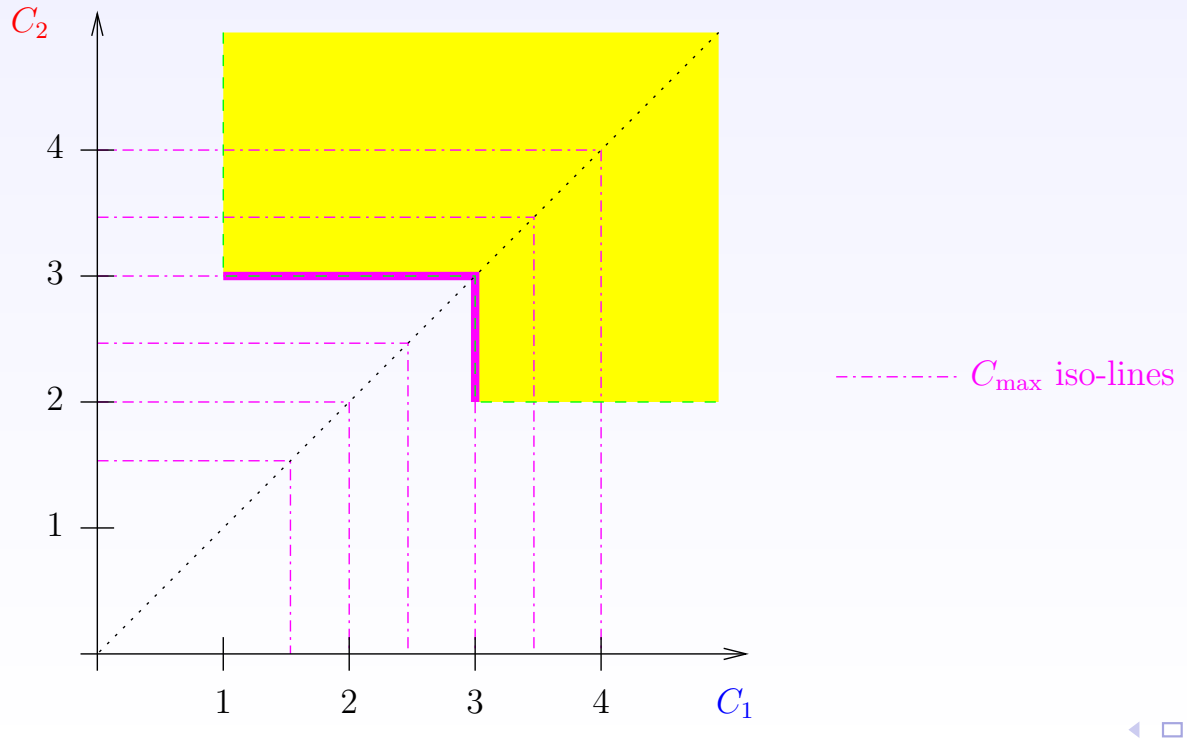
Pareto optimality

The cost (here the completion time) of a user can not be improved without degrading the cost of another user.



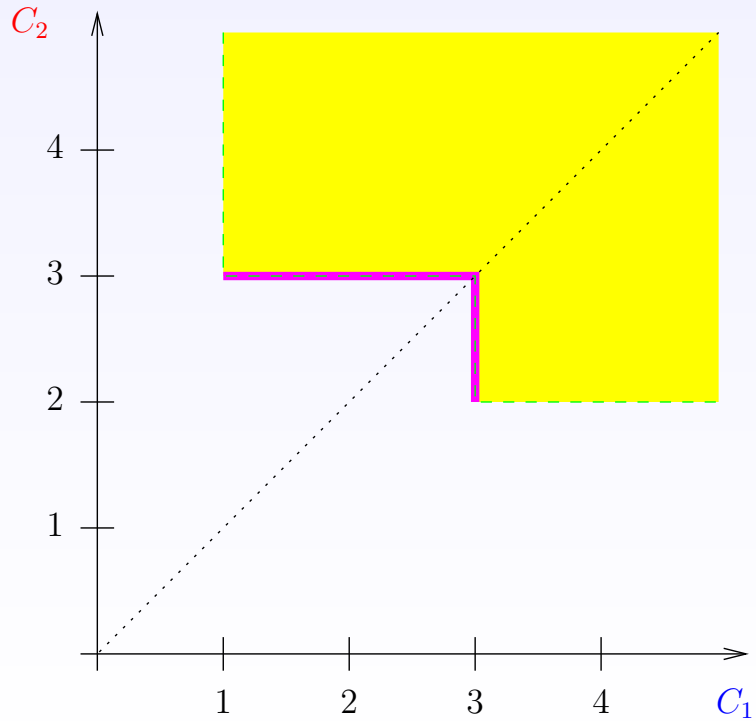
Pareto optimality

The cost (here the completion time) of a user can not be improved without degrading the cost of another user.



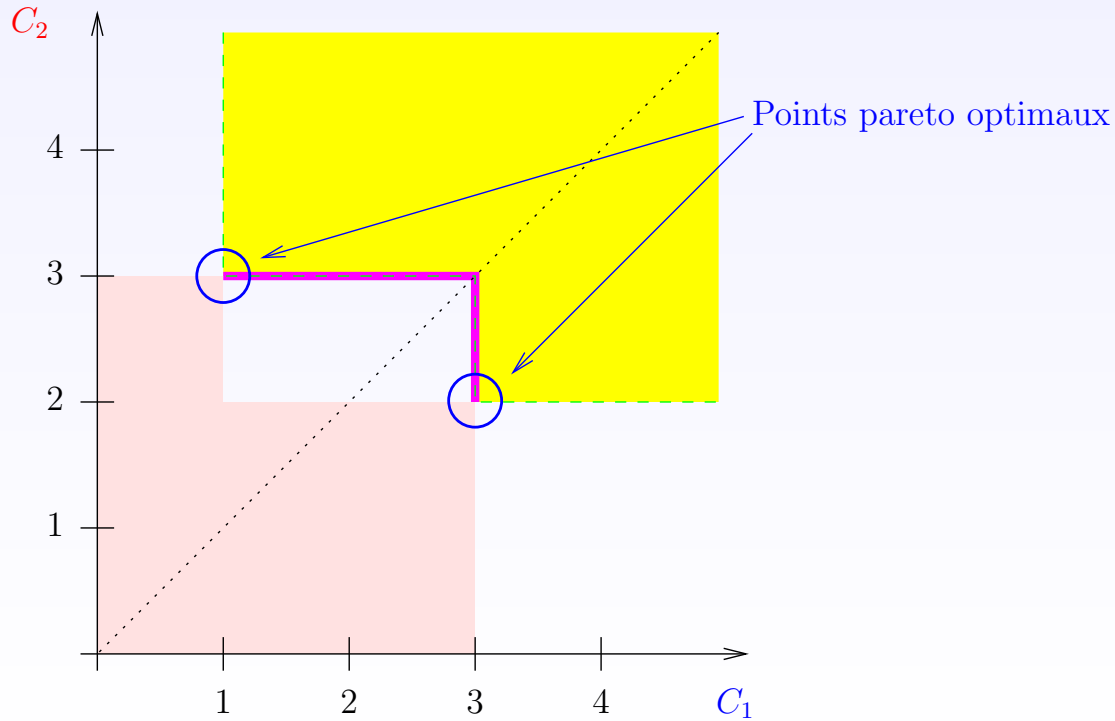
Pareto optimality

The cost (here the completion time) of a user can not be improved without degrading the cost of another user.



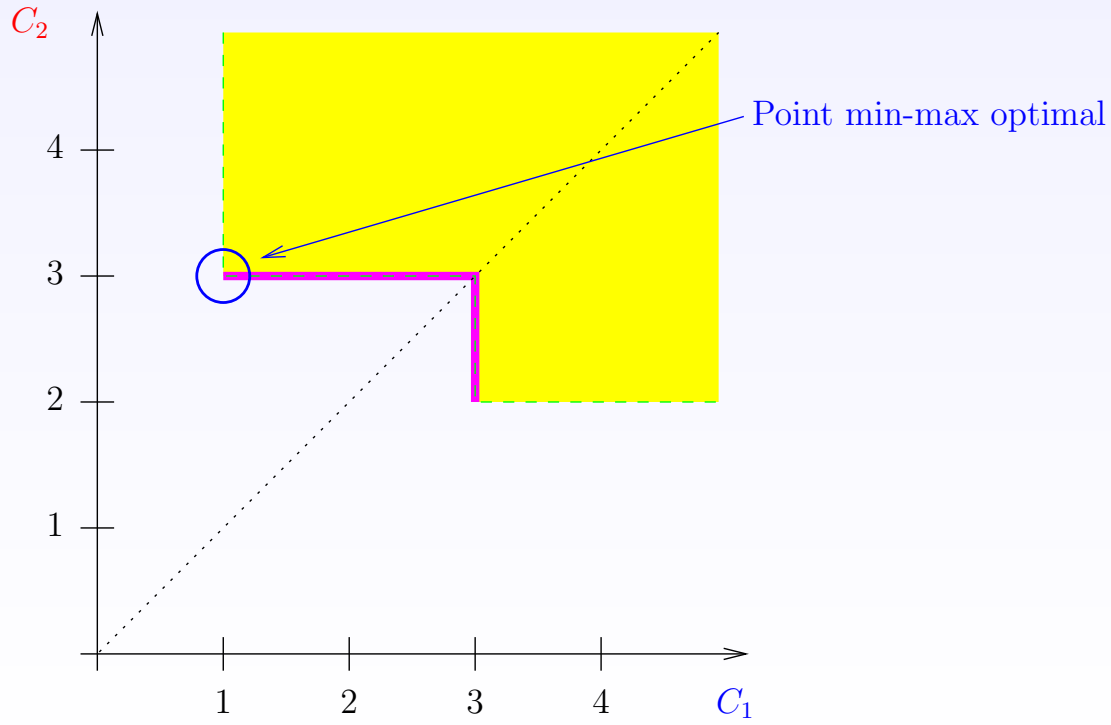
Pareto optimality

The cost (here the completion time) of a user can not be improved without degrading the cost of another user.



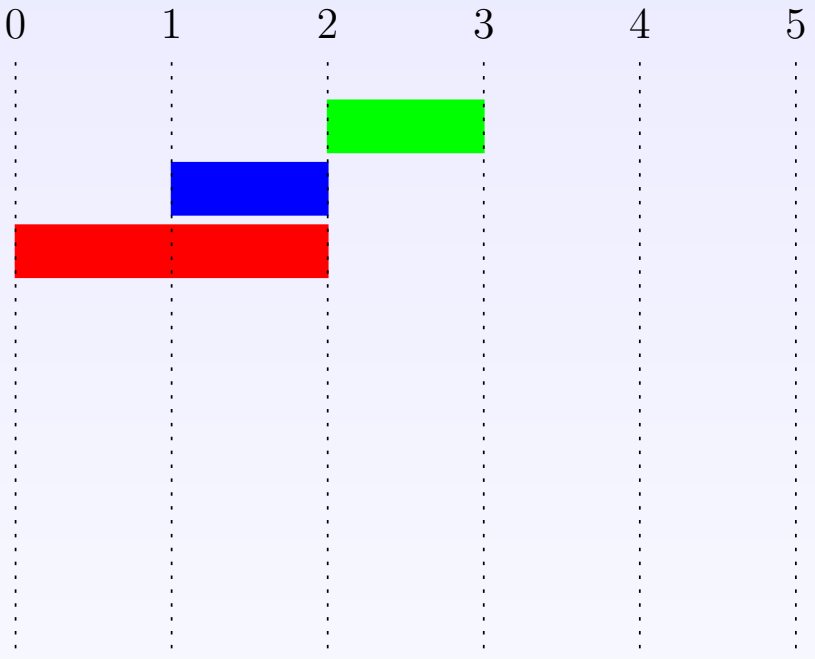
Pareto optimality

The cost (here the completion time) of a user can not be improved without degrading the cost of another user.

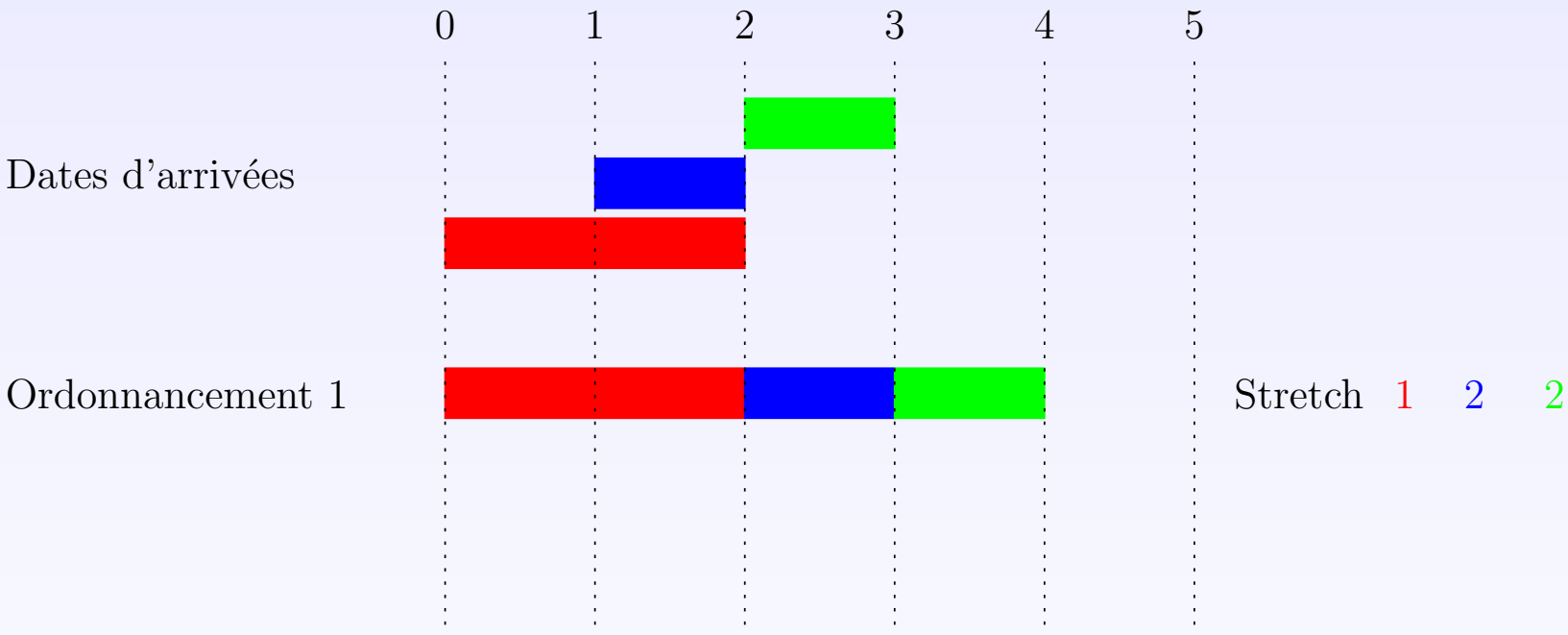


The Pareto max-stretch (1)

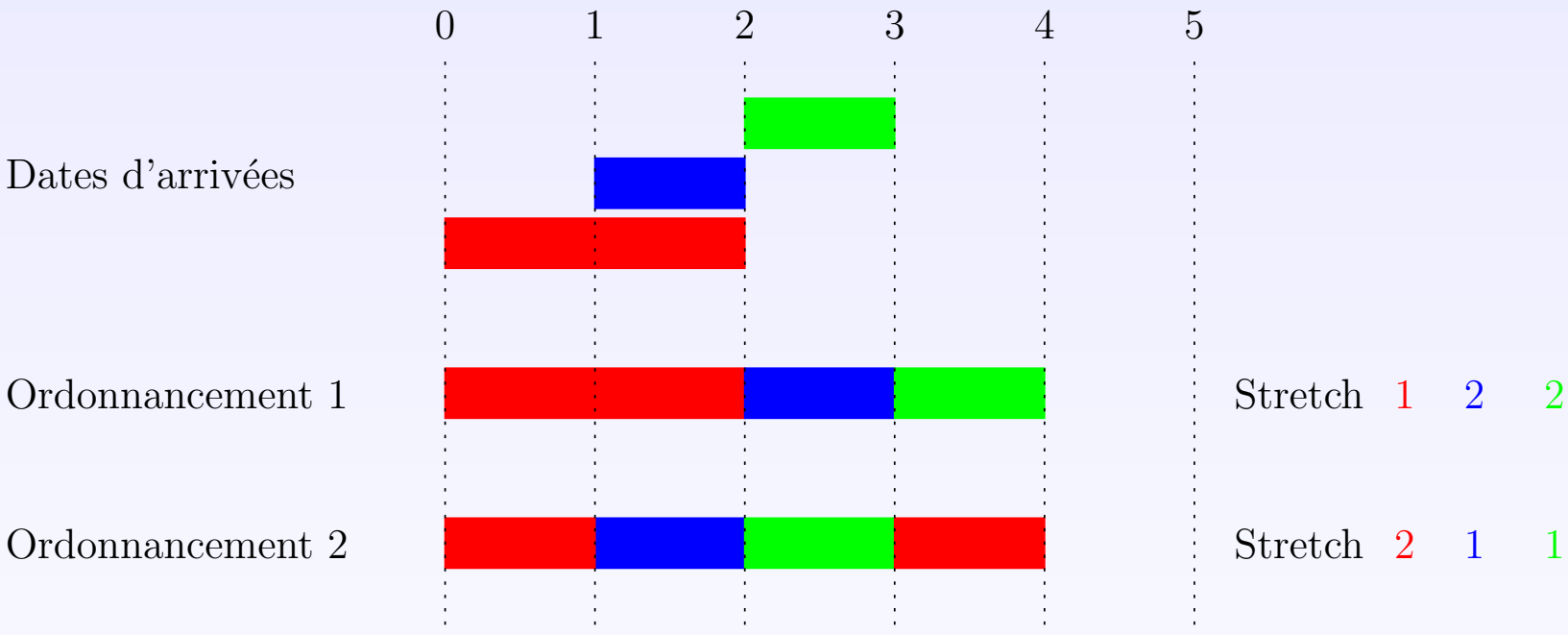
Dates d'arrivées



The Pareto max-stretch (1)



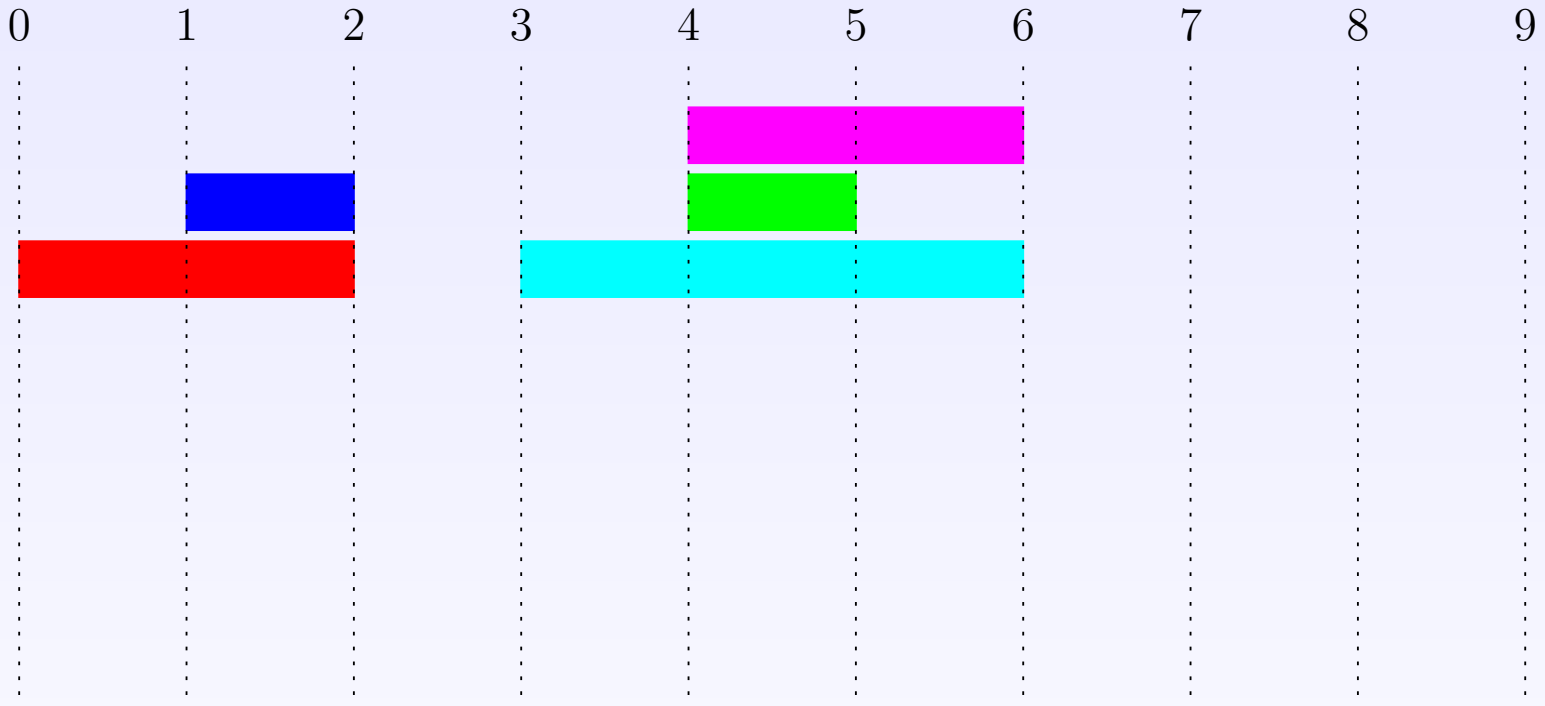
The Pareto max-stretch (1)



Schedule 1 and Schedule 2 are Pareto optimal

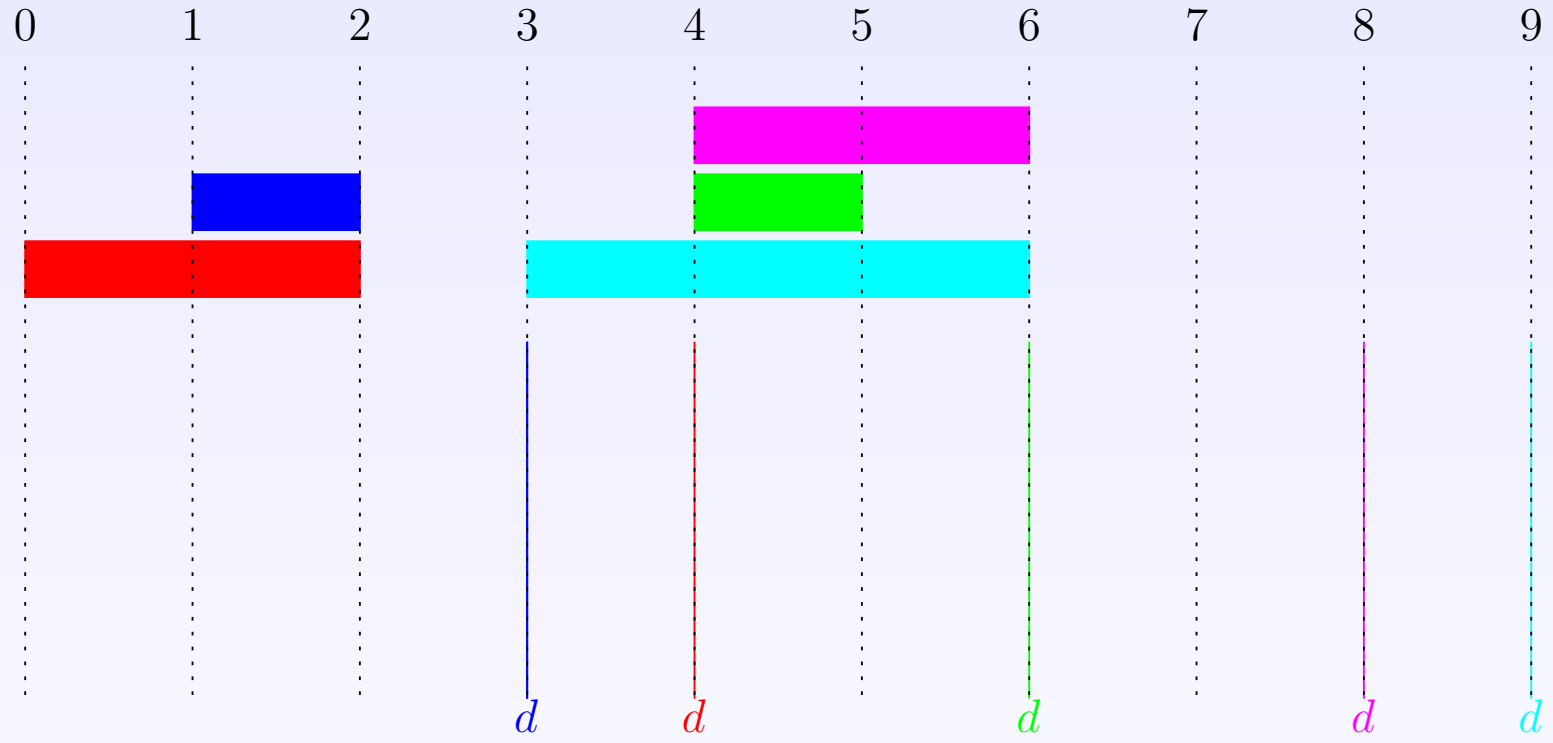
Schedule 2 is the min-max solution

The Pareto max-stretch (2)



Computation of the optimal max-stretch: 2.

The Pareto max-stretch (2)

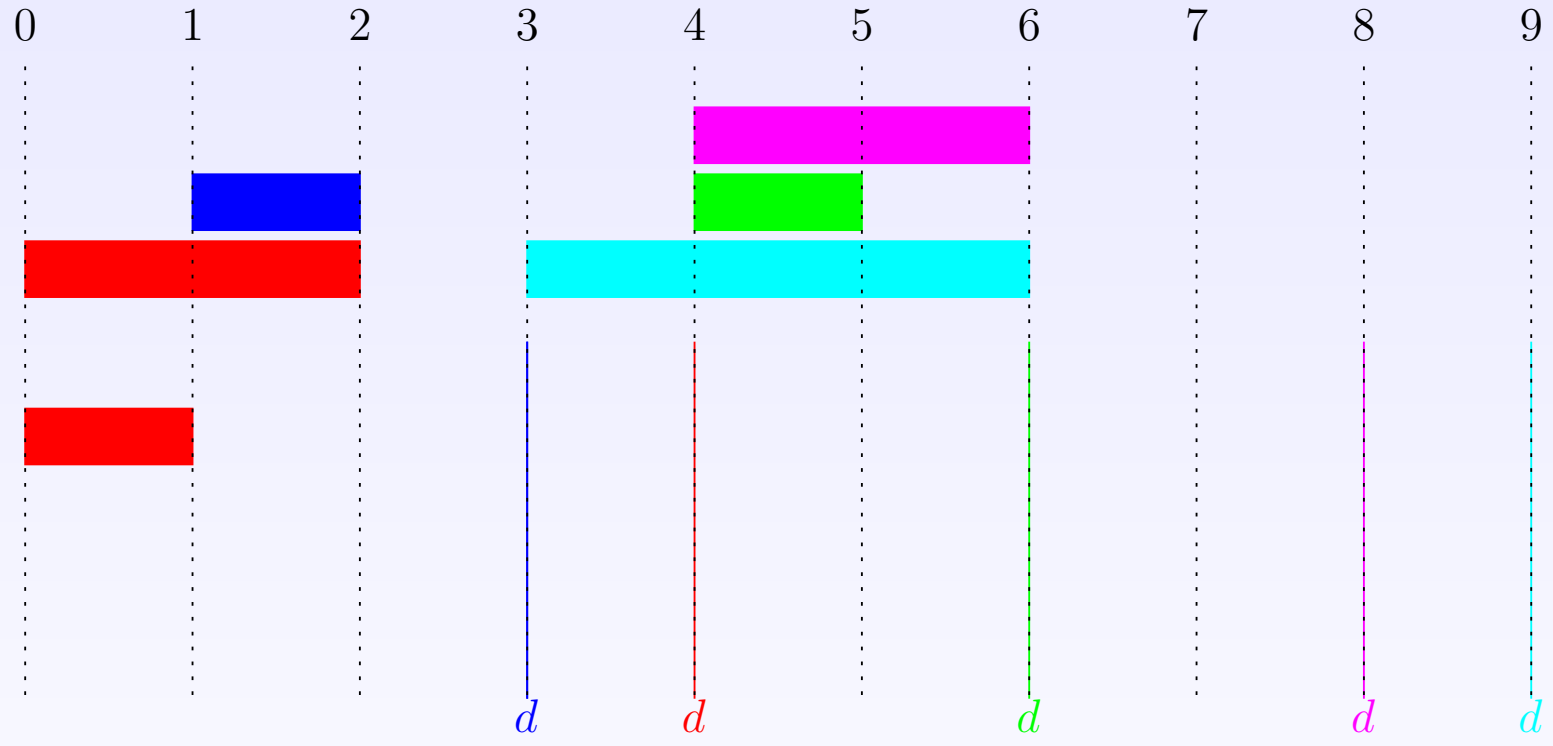


Computation of the optimal max-stretch: 2.

Defining a deadline per job.

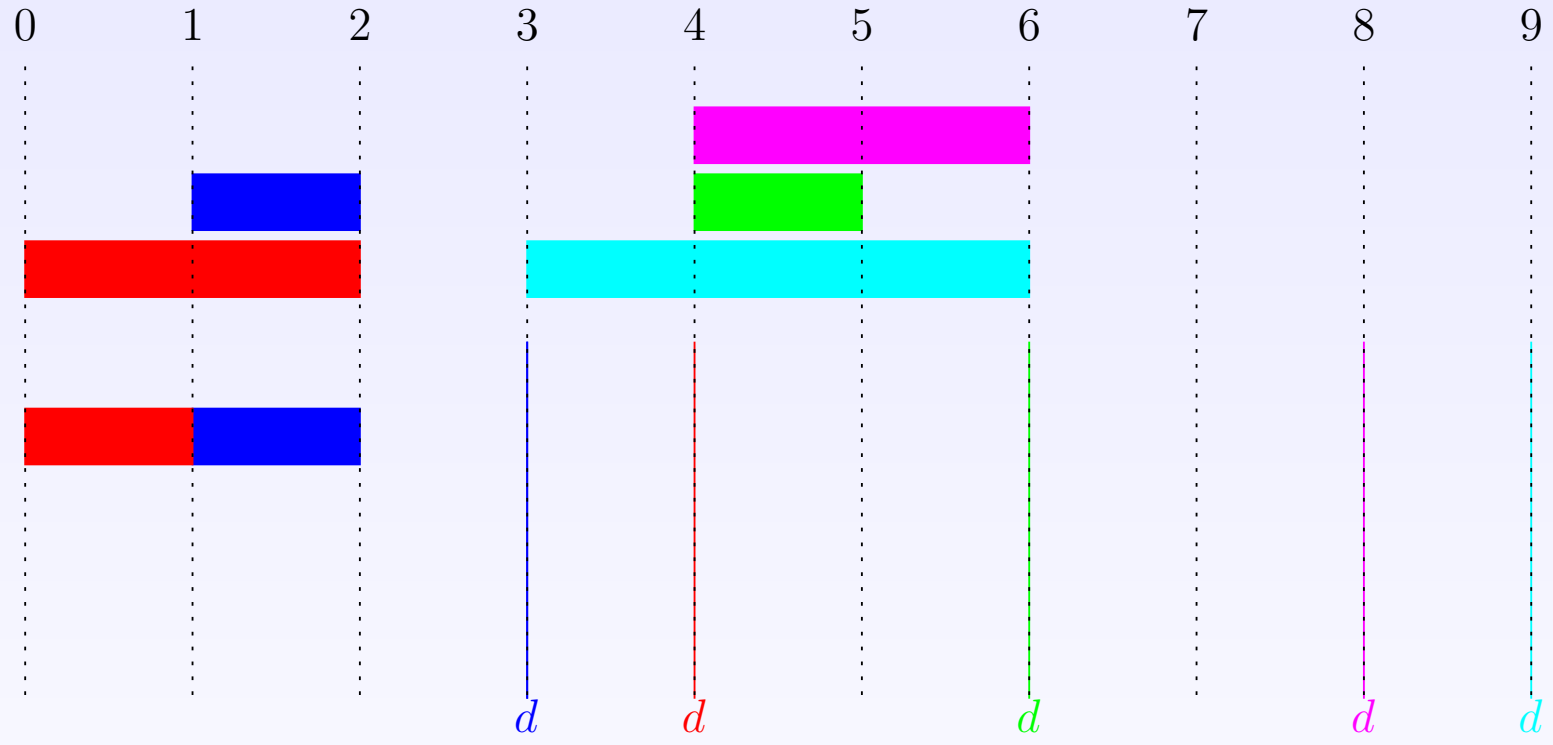
Jobs are scheduled *Earliest deadline first*.

The Pareto max-stretch (2)



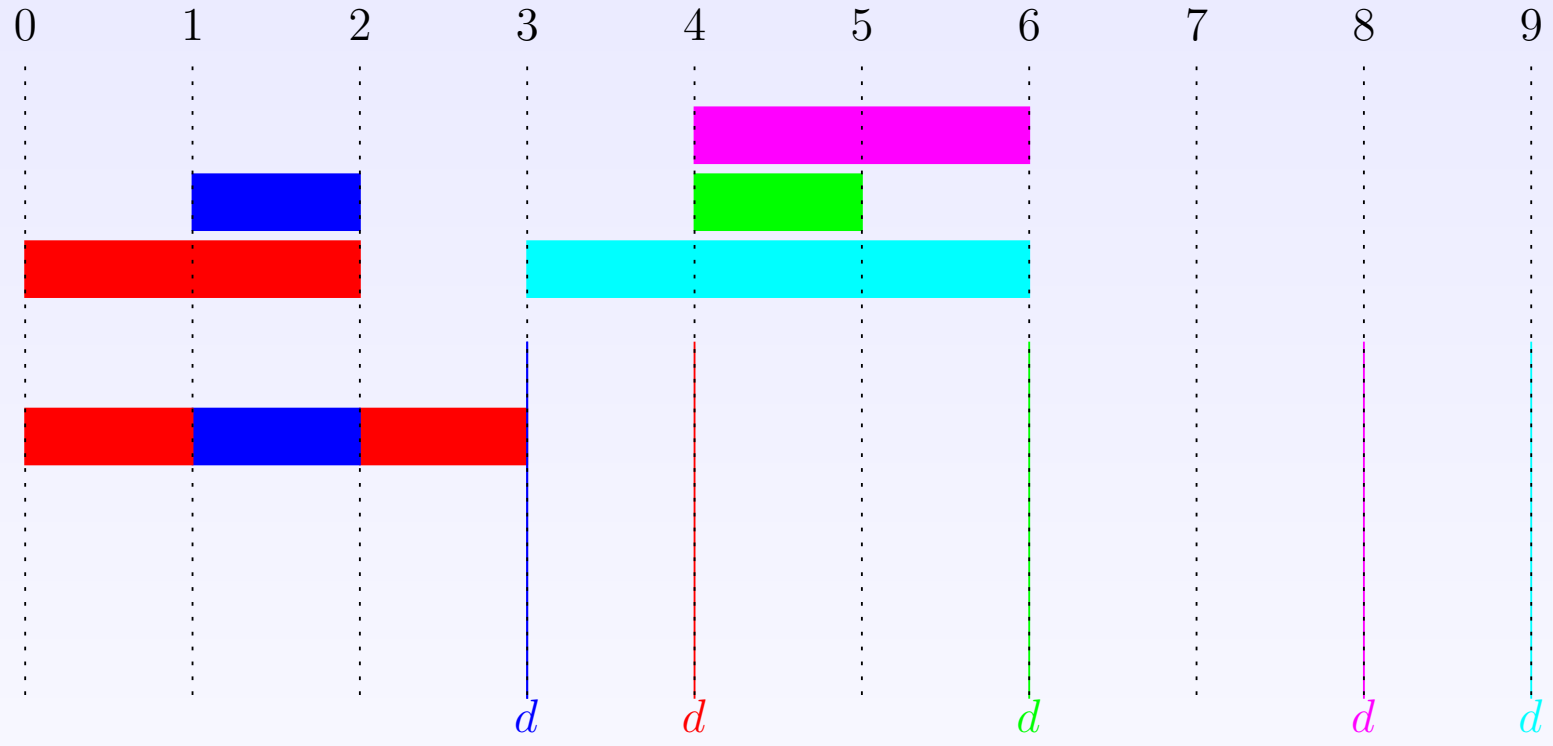
Jobs are scheduled *Earliest deadline first*.

The Pareto max-stretch (2)



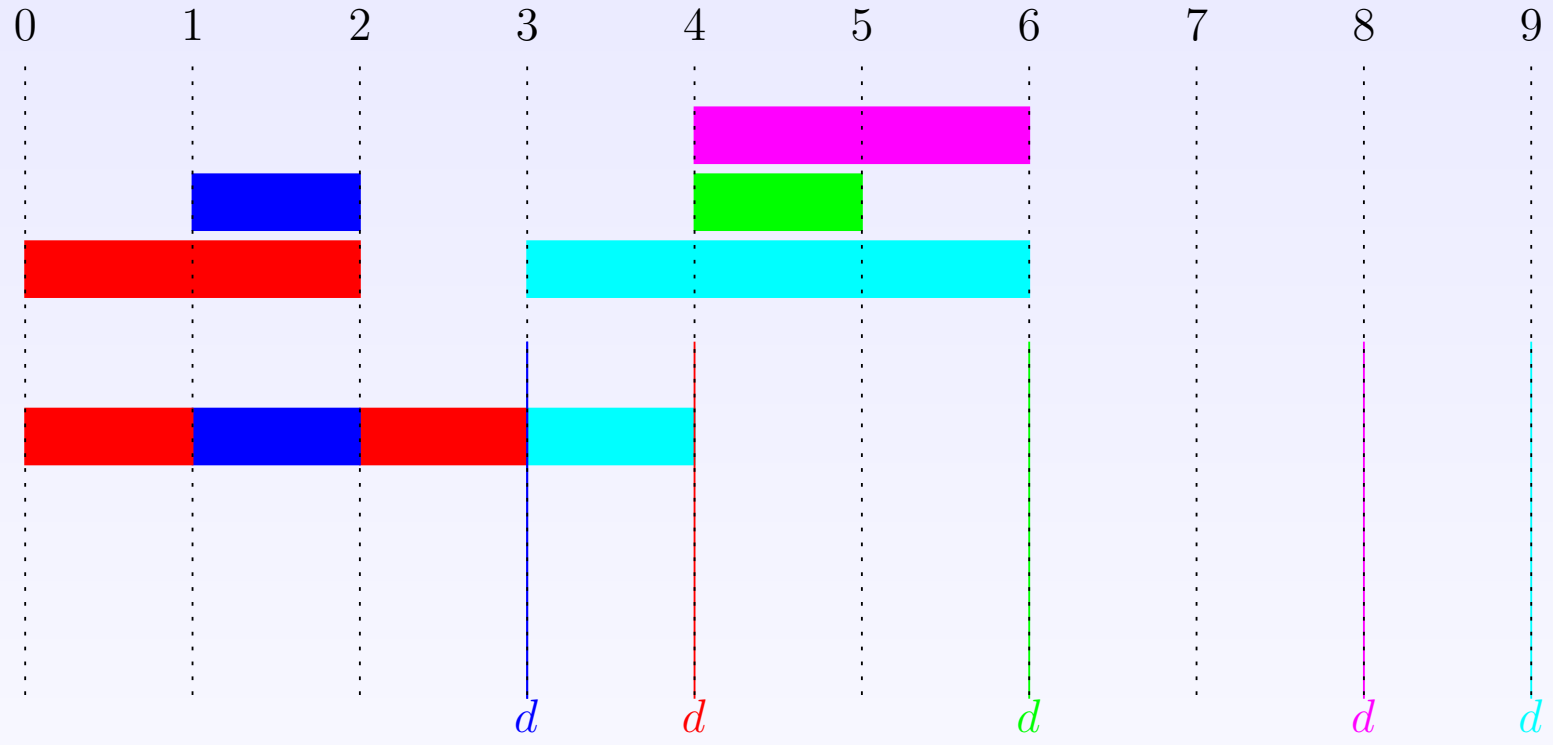
Jobs are scheduled *Earliest deadline first*.

The Pareto max-stretch (2)



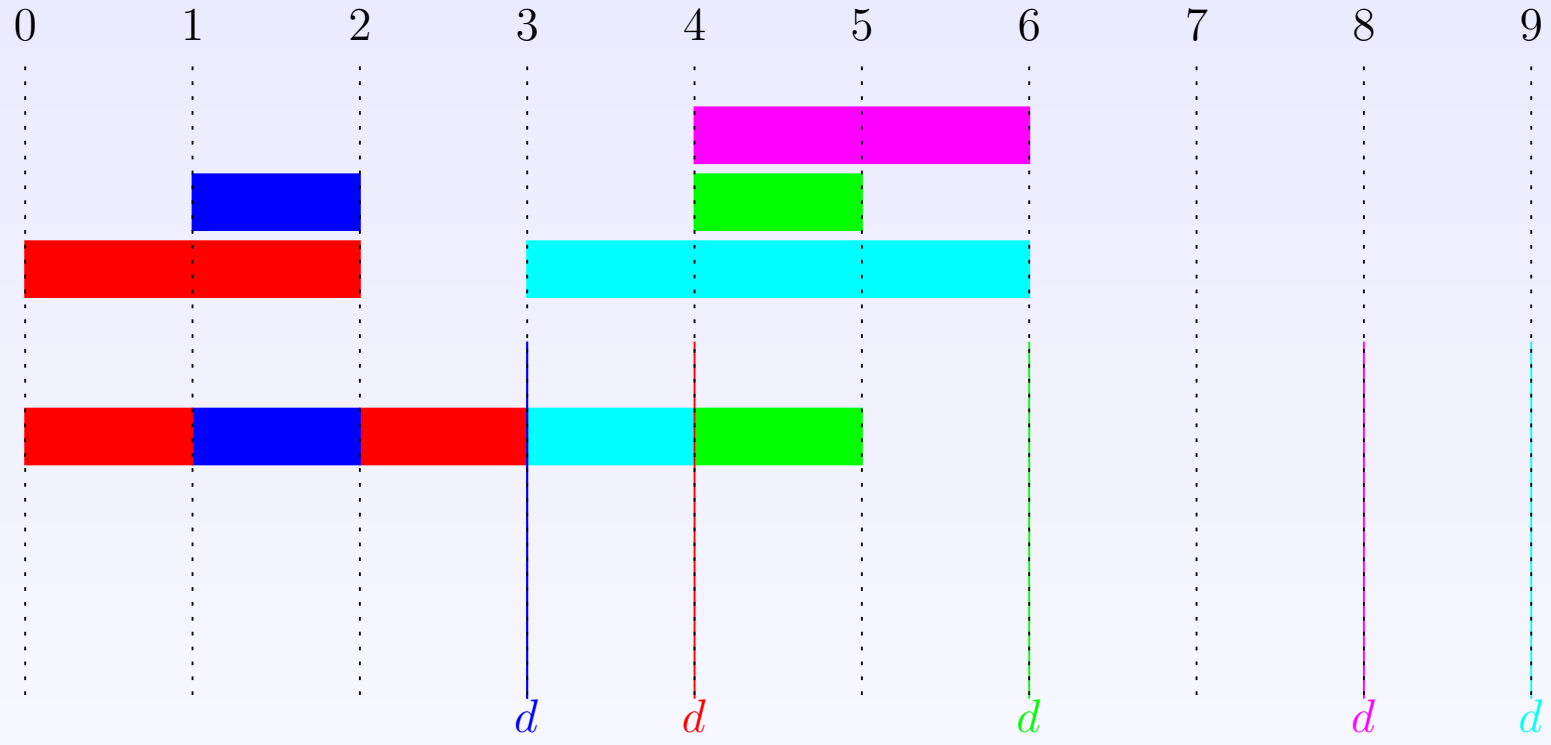
Jobs are scheduled *Earliest deadline first*.

The Pareto max-stretch (2)



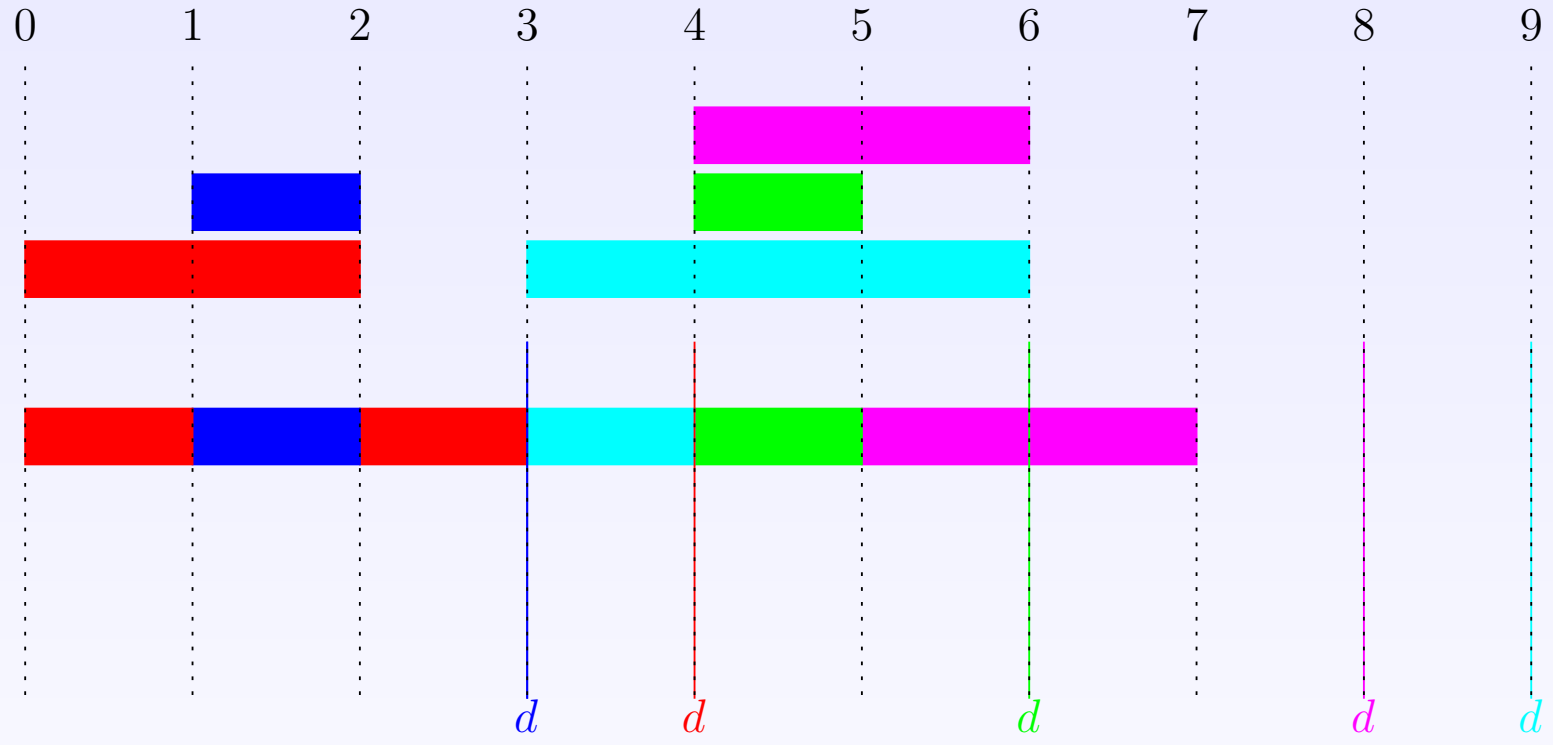
Jobs are scheduled *Earliest deadline first*.

The Pareto max-stretch (2)



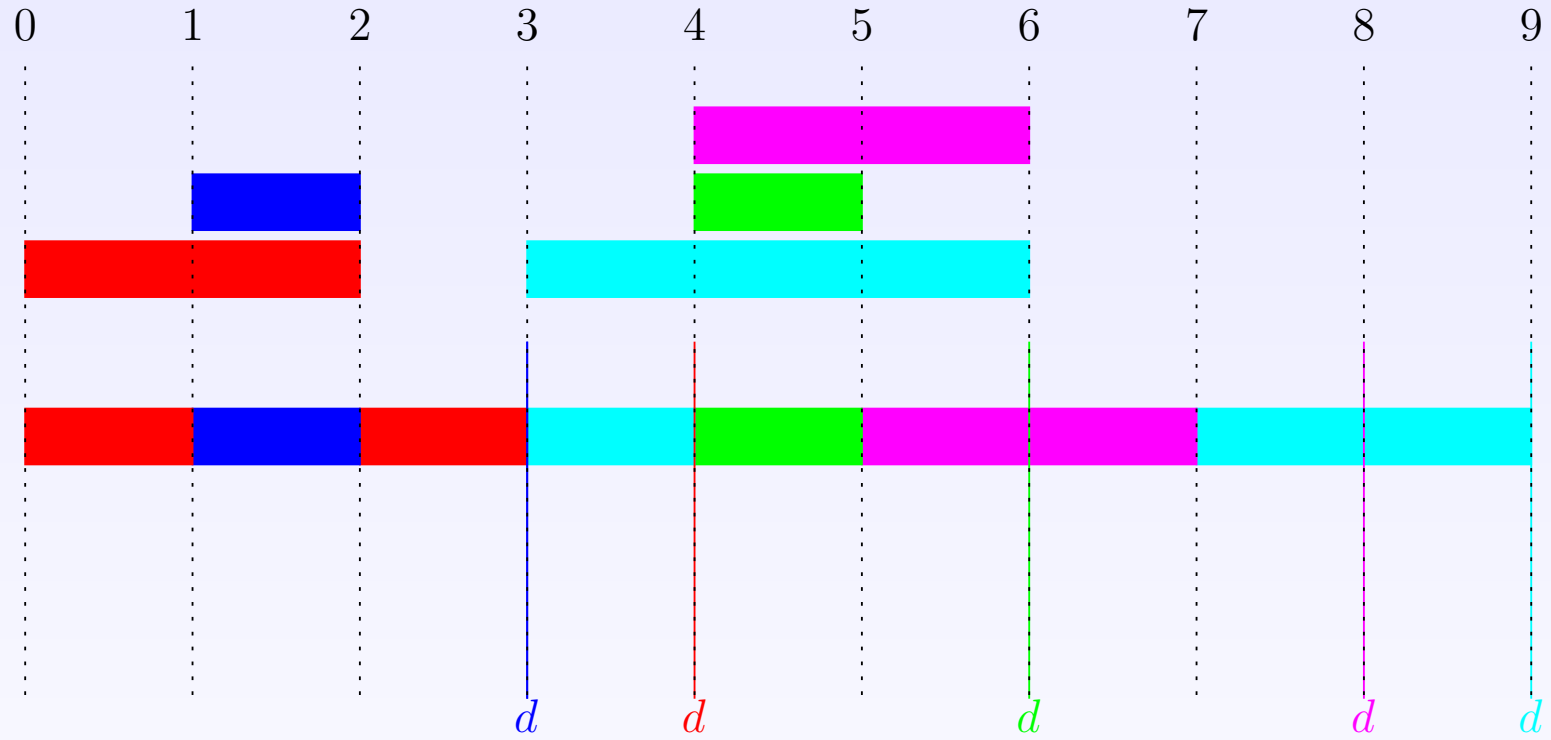
Jobs are scheduled *Earliest deadline first*.

The Pareto max-stretch (2)



Jobs are scheduled *Earliest deadline first*.

The Pareto max-stretch (2)

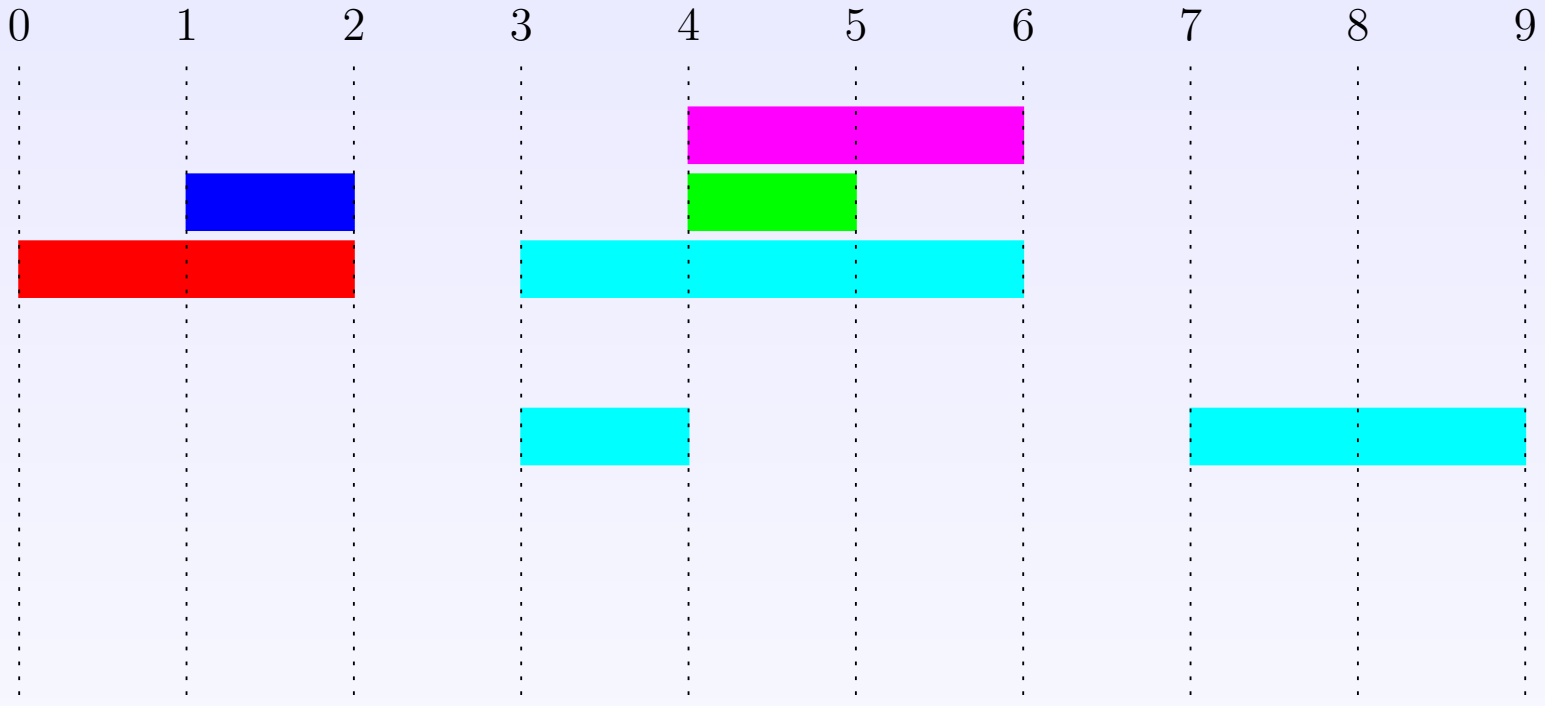


Jobs are scheduled *Earliest deadline first*.

If completion time = deadline, whatever the schedule, the stretch of this job is equal to the maximum stretch.

We set the jobs that cannot be optimized, and we call recursively the process.

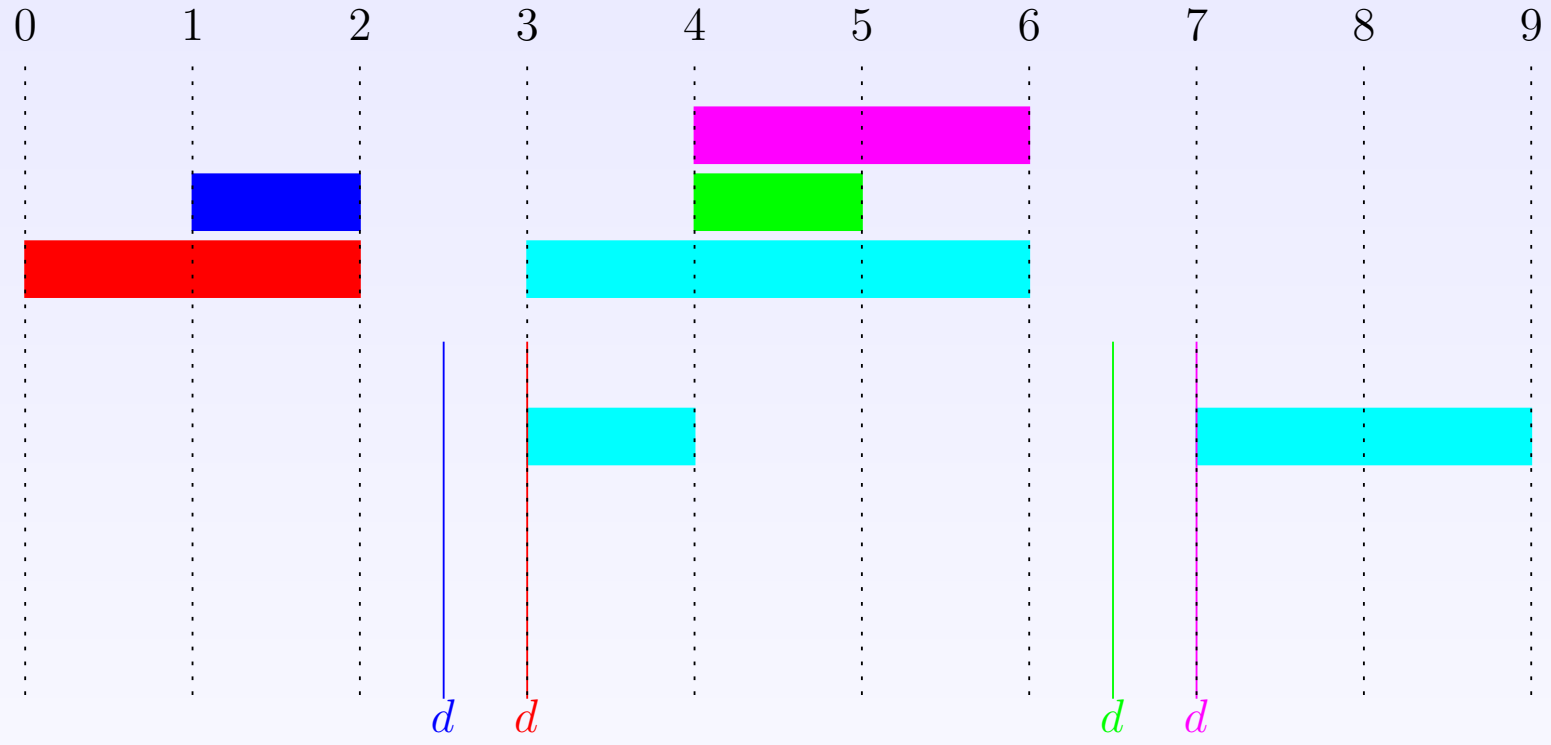
The Pareto max-stretch (2)



We set the jobs that cannot be optimized, and we call recursively the process.

Max-stretch of remaining jobs : 1,5.

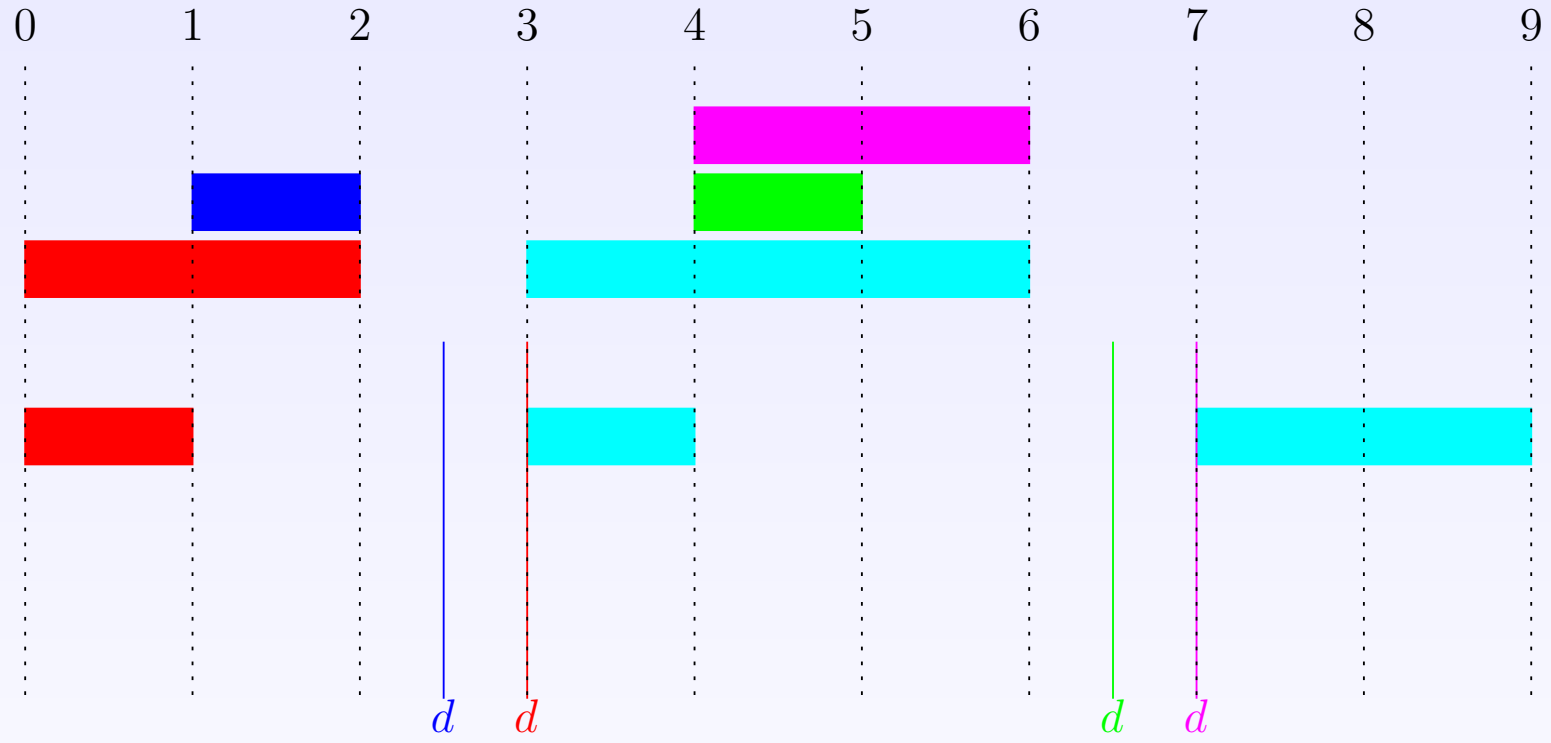
The Pareto max-stretch (2)



We set the jobs that cannot be optimized, and we call recursively the process.

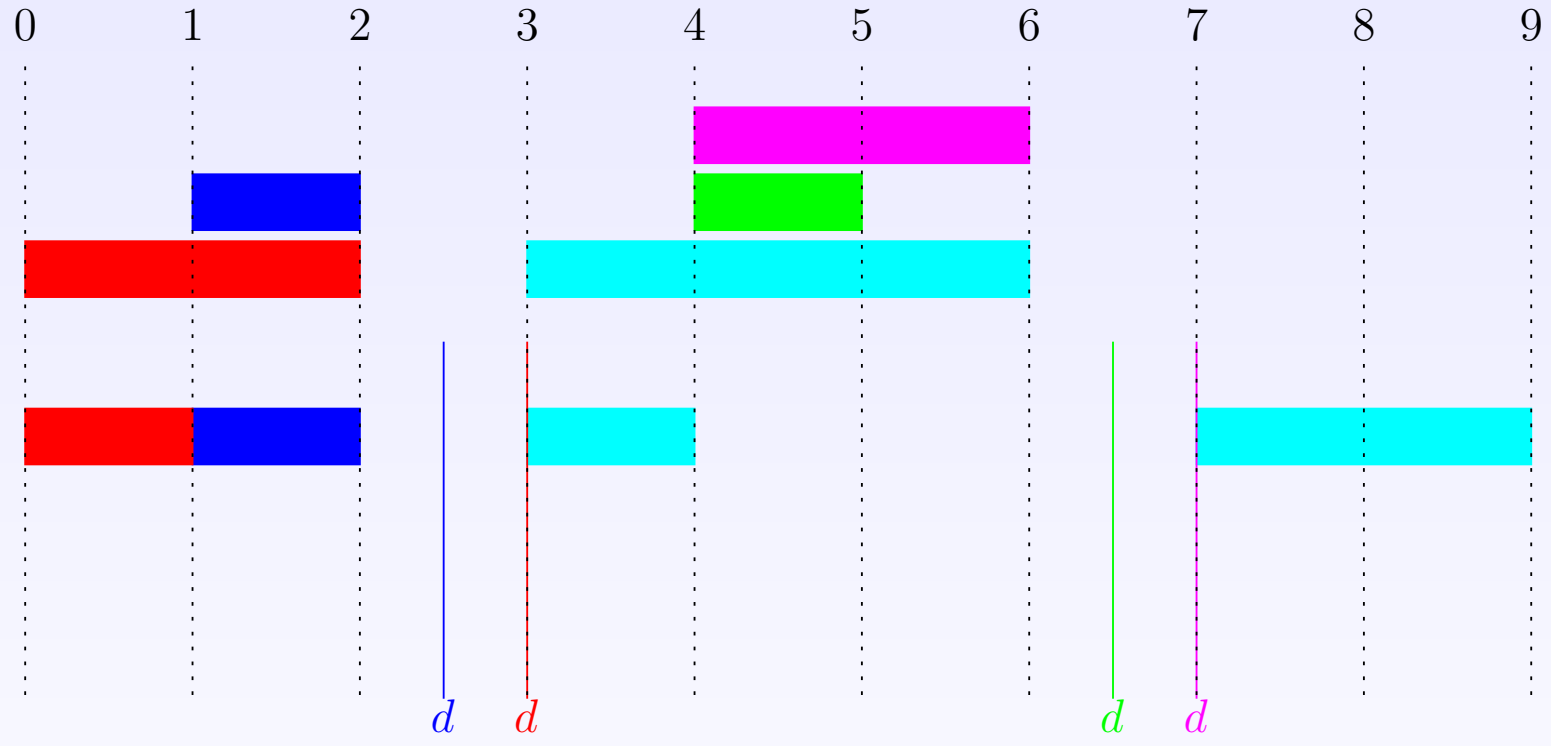
Max-stretch of remaining jobs : 1,5.

The Pareto max-stretch (2)



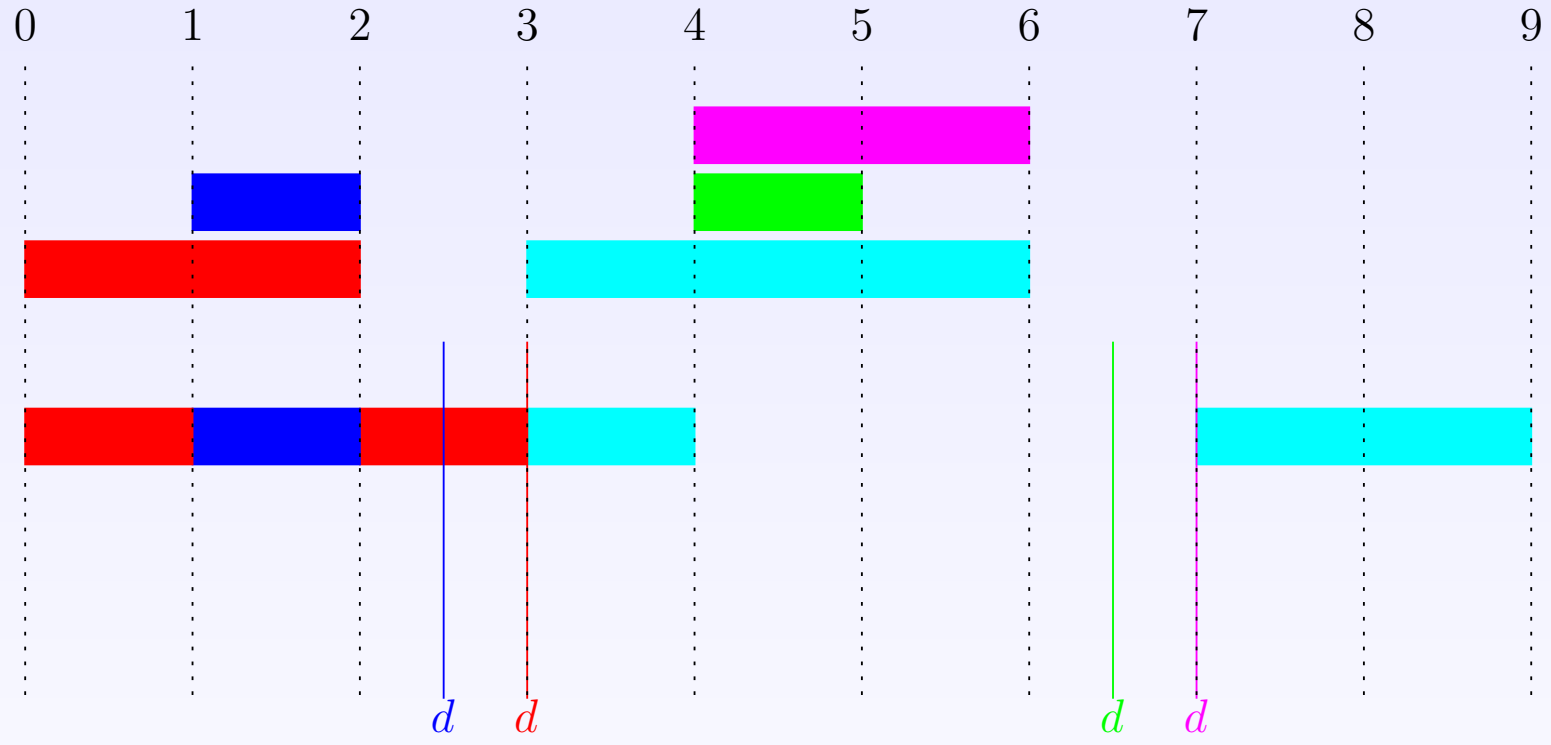
Jobs are scheduled *Earliest deadline first*.

The Pareto max-stretch (2)



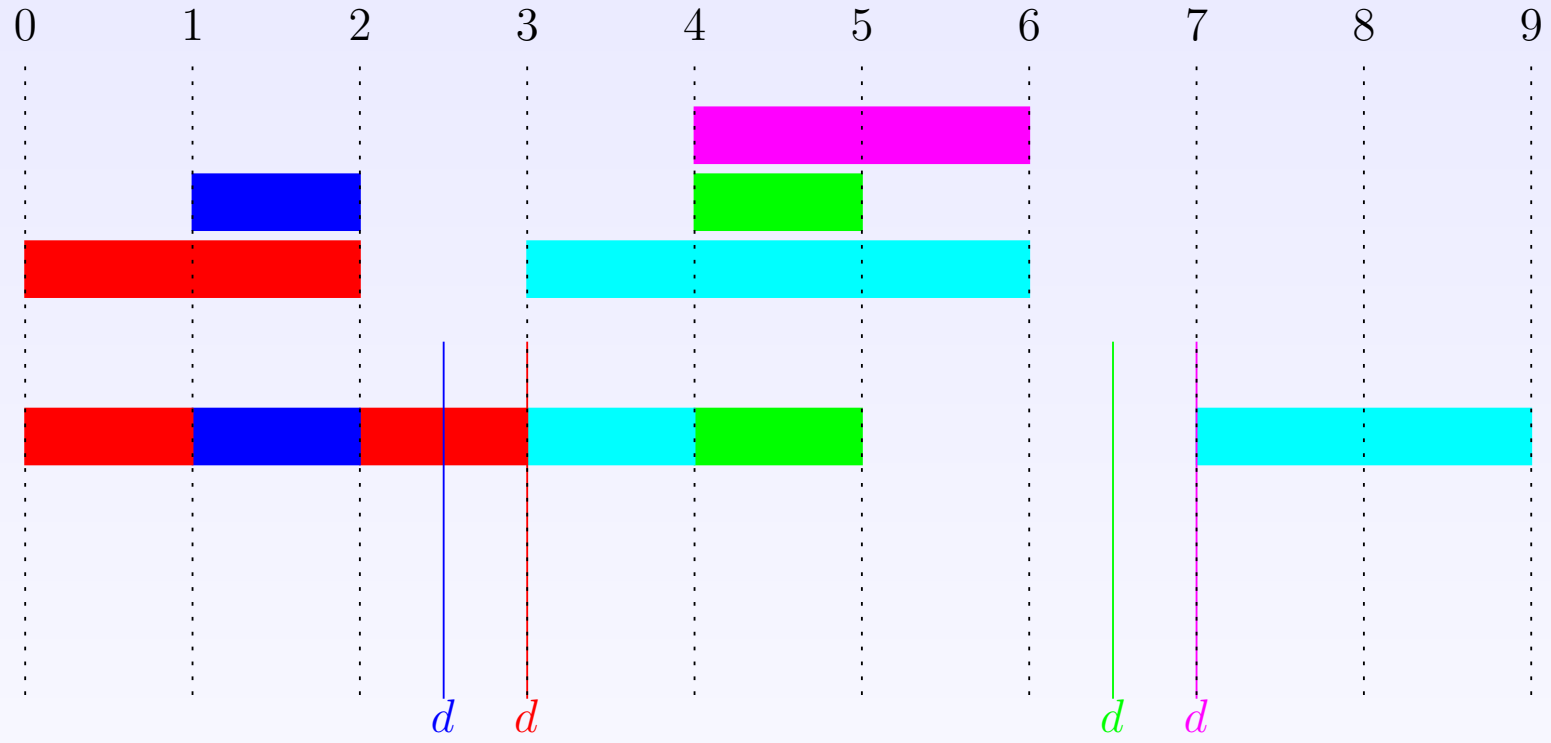
Jobs are scheduled *Earliest deadline first*.

The Pareto max-stretch (2)



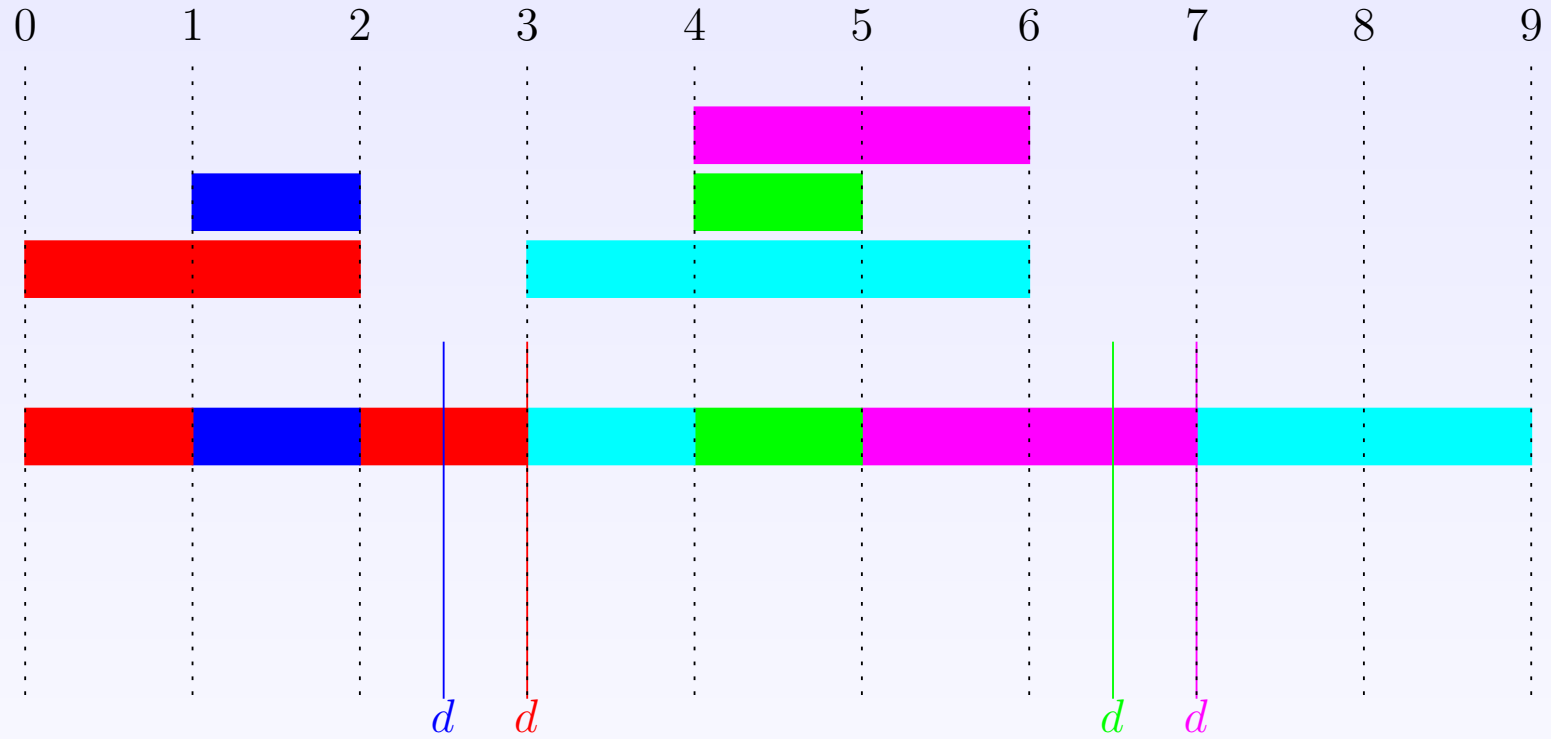
Jobs are scheduled *Earliest deadline first*.

The Pareto max-stretch (2)



Jobs are scheduled *Earliest deadline first*.

The Pareto max-stretch (2)

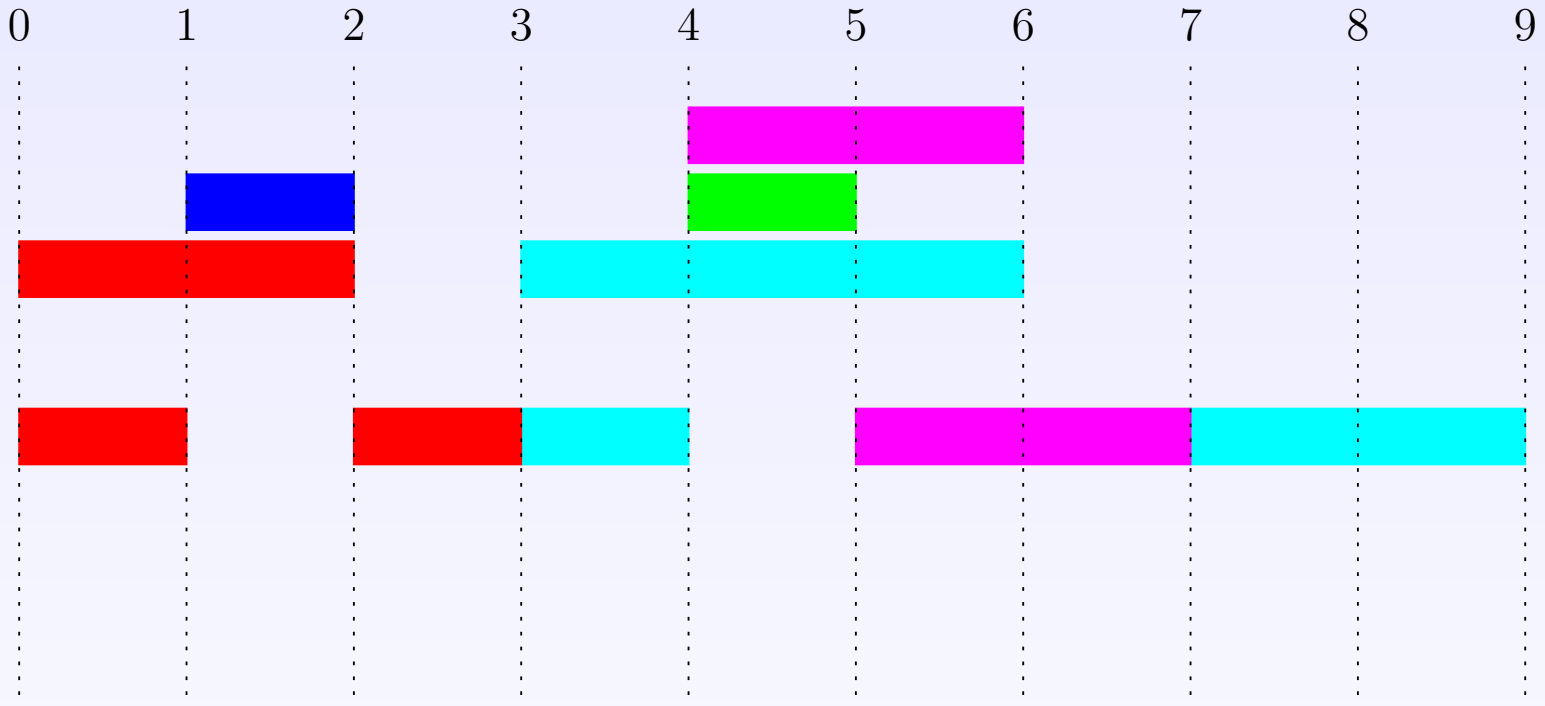


Jobs are scheduled *Earliest deadline first*.

If completion time = deadline, whatever the schedule, the stretch of this job is equal to the maximum stretch.

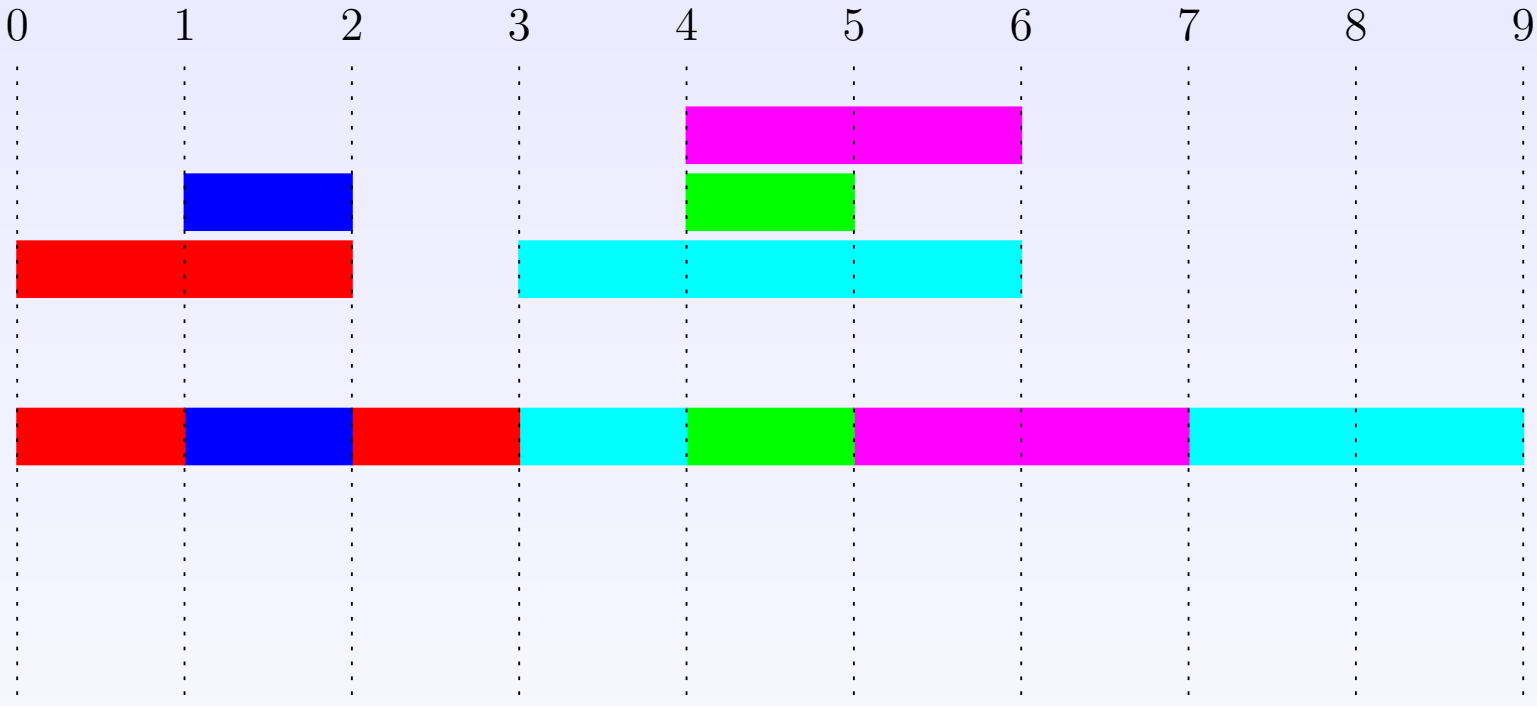
We set the jobs that cannot be optimized, and we call recursively the process.

The Pareto max-stretch (2)



We set the jobs that cannot be optimized, and we call recursively the process.

The Pareto max-stretch (2)



Outline

Rules of the game

- A job characteristics are only known at the time the job arrives in the system (i.e., at the release date).
- We want to minimize the maximum weighted flow (for any weights).
- Jobs are divisible.

Evaluating the quality of an online schedule

An online algorithm has a competitive factor ρ if and only if:

Whatever the set of jobs T_1, ..., T_n:

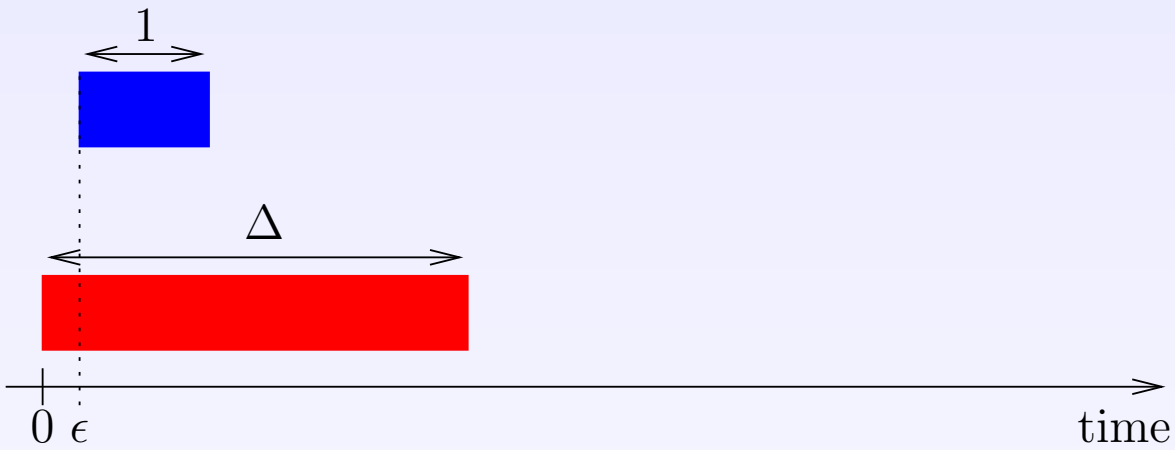
Online schedule cost$(T_1, ..., T_N) \leq$
$\rho \times$ Optimal off-line schedule cost$(T_1, ..., T_N)$.

FIFO competitiveness (1)

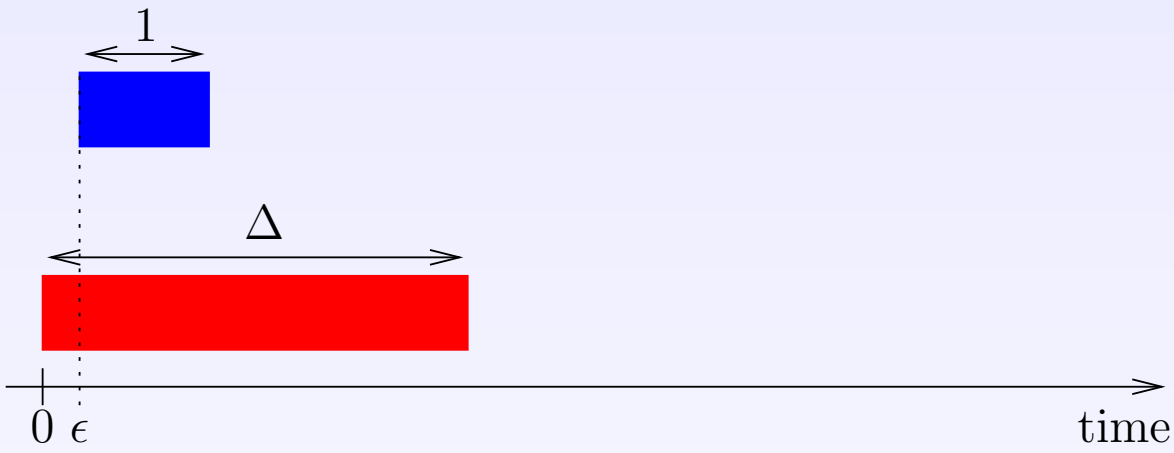
Theorem

FIFO is Δ competitive for maximum stretch minimization.

FIFO competitiveness (2): at bestΔ-competitive



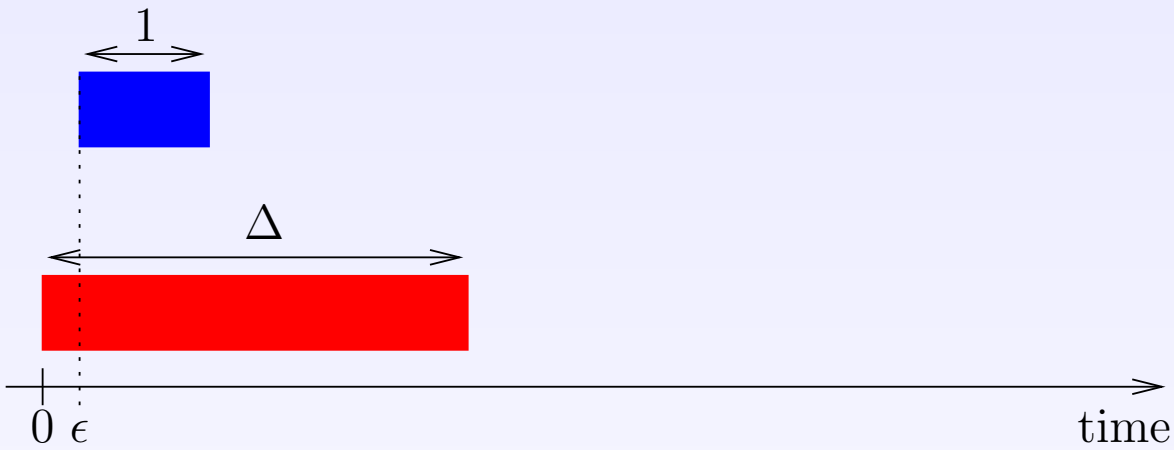
FIFO competitiveness (2): at bestΔ-competitive



FIFO

Max-stretch $= 1 + \Delta - \epsilon$

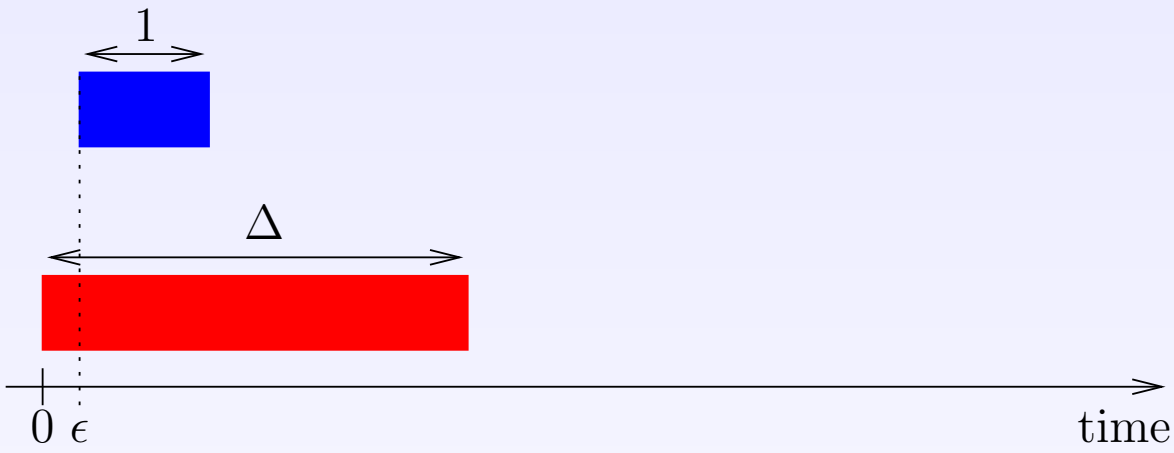
FIFO competitiveness (2): at bestΔ-competitive

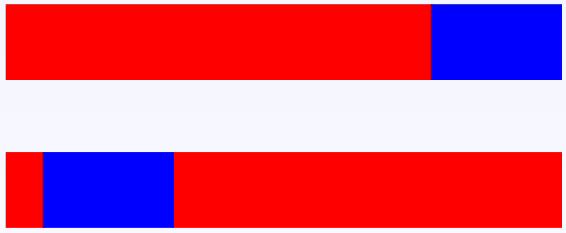


FIFO Max-stretch $= 1 + \Delta - \epsilon$

Optimal Max-stretch $= \frac{1+\Delta}{\Delta}$

FIFO competitiveness (2): at bestΔ-competitive





FIFO Max-stretch $= 1 + \Delta - \epsilon$

Optimal Max-stretch $= \frac{1+\Delta}{\Delta}$

Competitive ratio: $\frac{1+\Delta-\epsilon}{\frac{1+\Delta}{\Delta}} = \Delta \frac{1+\Delta-\epsilon}{1+\Delta} = \Delta - \epsilon \frac{\Delta}{1+\Delta} \geq \Delta - \epsilon.$

FIFO competitiveness (3): at worst Δ competitive

- An instance J_1, ..., J_n.
  Θ^*: an optimal schedule for max-stretch.
  C_j: completion time of J_j under FIFO (C_j^* under Θ^*).
  $\mathcal{S}_j$: stretch of J_j under FIFO ($\mathcal{S}_j^*$ under Θ^*).

FIFO competitiveness (3): at worst Δ competitive

- An instance J_1, ..., J_n.
  Θ^*: an optimal schedule for max-stretch.
  C_j: completion time of J_j under FIFO (C_j^* under Θ^*).
  $\mathcal{S}_j$: stretch of J_j under FIFO ($\mathcal{S}_j^*$ under Θ^*).
- Any job J_l s.t. $\mathcal{S}_l > \mathcal{S}_l^*$.

FIFO competitiveness (3): at worst Δ competitive

- An instance J_1, ..., J_n.
  Θ^*: an optimal schedule for max-stretch.
  C_j: completion time of J_j under FIFO (C_j^* under Θ^*).
  $\mathcal{S}_j$: stretch of J_j under FIFO ($\mathcal{S}_j^*$ under Θ^*).
- Any job J_l s.t. $\mathcal{S}_l > \mathcal{S}_l^*$.
  t last time before C_l s.t. the processor was idle under FIFO.
  t is the release date r_i of some job J_i.

FIFO competitiveness (3): at worst Δ competitive

- An instance J_1, ..., J_n.
  Θ^*: an optimal schedule for max-stretch.
  C_j: completion time of J_j under FIFO (C_j^* under Θ^*).
  $\mathcal{S}_j$: stretch of J_j under FIFO ($\mathcal{S}_j^*$ under Θ^*).
- Any job J_l s.t. $\mathcal{S}_l > \mathcal{S}_l^*$.
  t last time before C_l s.t. the processor was idle under FIFO.
  t is the release date r_i of some job J_i.
  During the time interval $[r_i, C_l]$, FIFO exactly executes J_i, J_{i+1}, ..., J_{l-1}, J_l.

FIFO competitiveness (3): at worst Δ competitive

- An instance J_1, ..., J_n.
  Θ^*: an optimal schedule for max-stretch.
  C_j: completion time of J_j under FIFO (C_j^* under Θ^*).
  $\mathcal{S}_j$: stretch of J_j under FIFO ($\mathcal{S}_j^*$ under Θ^*).
- Any job J_l s.t. $\mathcal{S}_l > \mathcal{S}_l^*$.
  t last time before C_l s.t. the processor was idle under FIFO.
  t is the release date r_i of some job J_i.
  During the time interval $[r_i, C_l]$, FIFO exactly executes J_i, J_{i+1}, ..., J_{l-1}, J_l.
  As $C_l^* < C_l$, there is a job J_k, $i \leq k \leq l-1$ s.t. $C_k^* \geq C_l$.
  Then:

$$\max_j \mathcal{S}_j^* \geq \mathcal{S}_k^* = \frac{C_k^* - r_k}{c_k} \geq \frac{C_l - r_l}{c_k} = \frac{C_l - r_l}{c_l}\frac{c_l}{c_k} \geq \mathcal{S}_l \times \frac{1}{\Delta}$$

$$\forall l, \mathcal{S}_l > \mathcal{S}_l^* \quad \Rightarrow \quad \mathcal{S}^* \geq \mathcal{S}_l \times \frac{1}{\Delta}.$$

Bound on the competitive ratio

Theorem

On one processor, any online scheduling algorithm with preemption minimizing the max-stretch has a competitive ratio greater than $\frac{1}{2}\Delta^{\sqrt{2}-1}$*, if the system receives at least jobs of three different sizes, and if* Δ *is the ratio between the size of the largest and the smallest job.*

Bound on the competitive ratio

Theorem

On one processor, any online scheduling algorithm with preemption minimizing the max-stretch has a competitive ratio greater than $\frac{1}{2}\Delta^{\sqrt{2}-1}$*, if the system receives at least jobs of three different sizes, and if* Δ *is the ratio between the size of the largest and the smallest job.*

Proof principle: by contradiction we assume that there exists an algorithm and we build a sequence of jobs and a scenario to make the algorithm fail.

The adversary

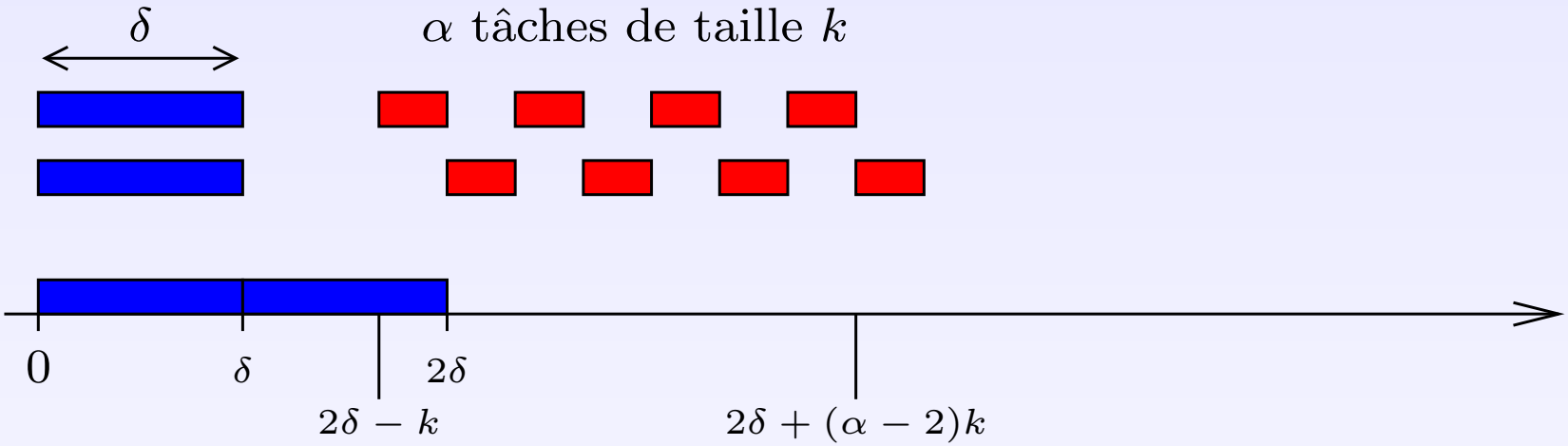
The adversary

The adversary

Achievable stretch: $\dfrac{2\delta - 0}{\delta} = 2.$

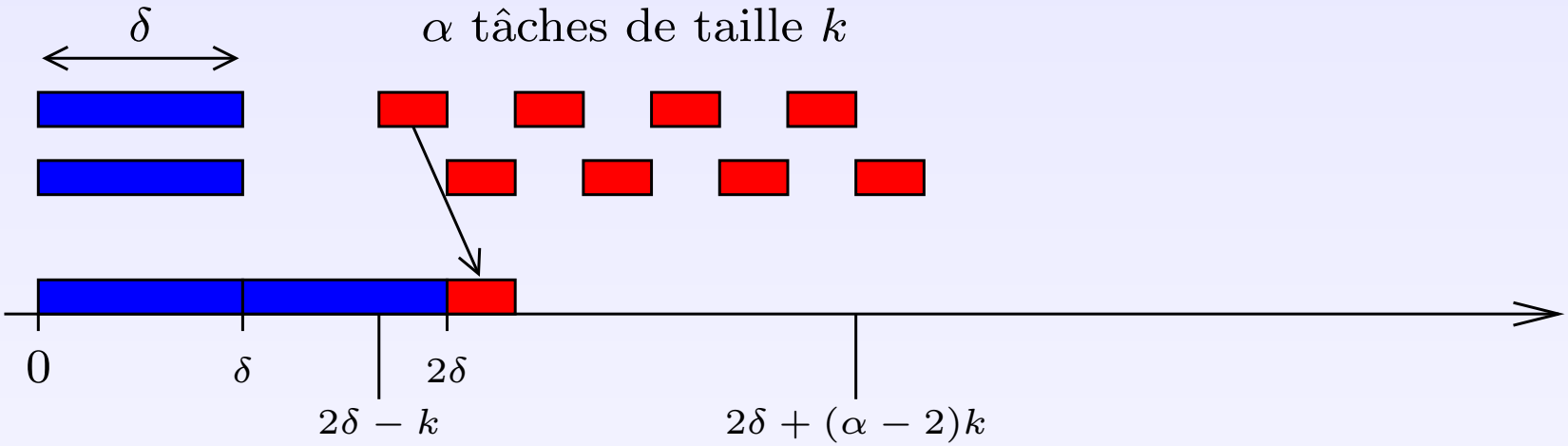
The adversary

The adversary



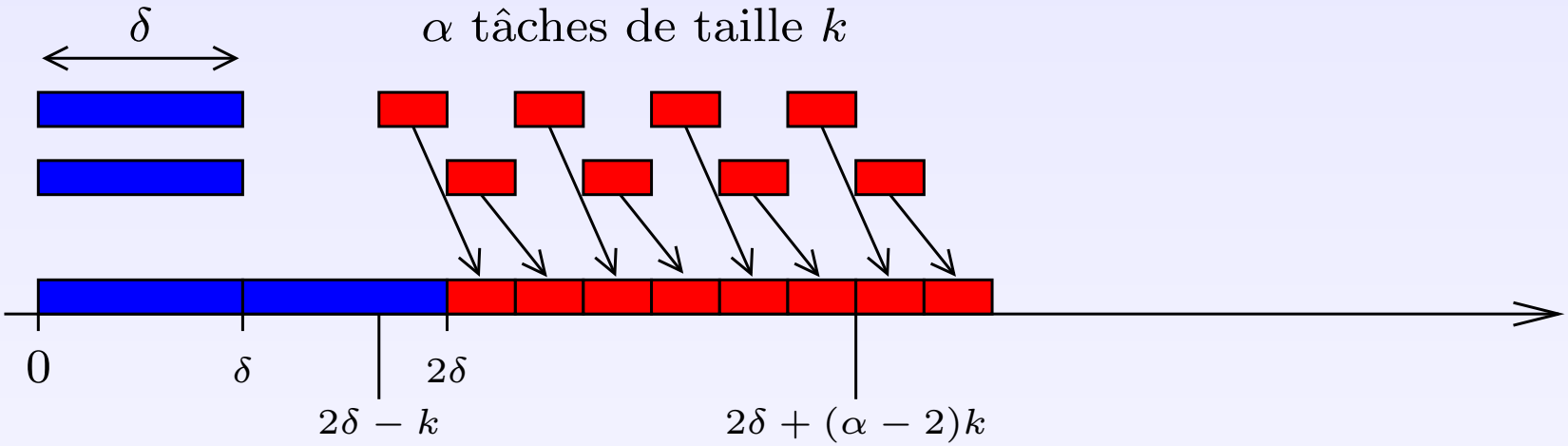
The job T_{2+j} arrives at time $2\delta + (j-2)k$.

The adversary



The job T_{2+j} arrives at time $2\delta + (j-2)k$.

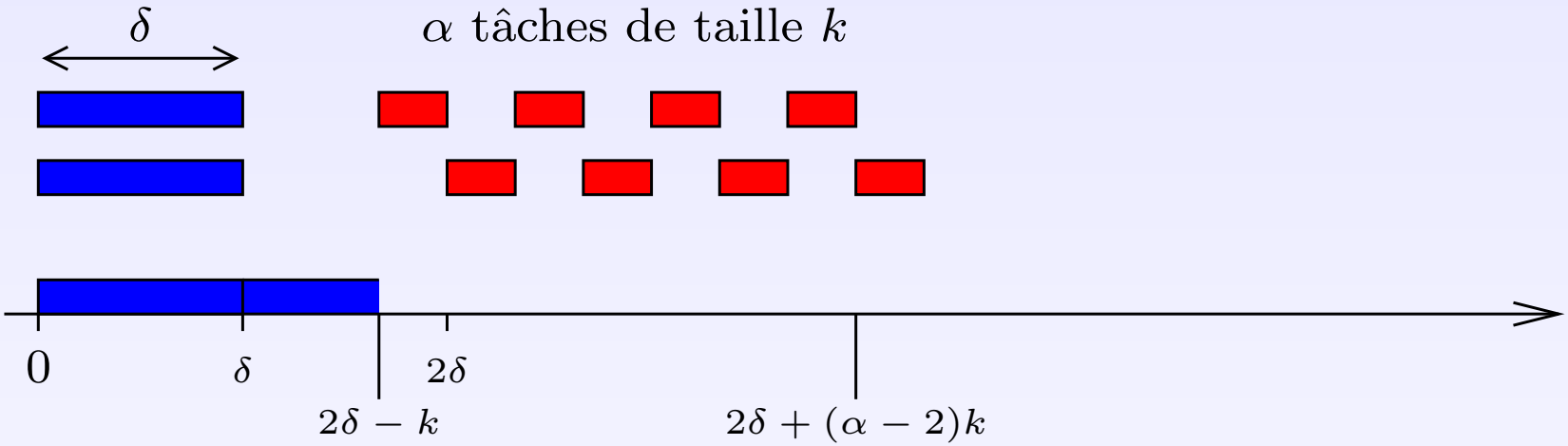
The adversary



The job T_{2+j} arrives at time $2\delta + (j-2)k$.

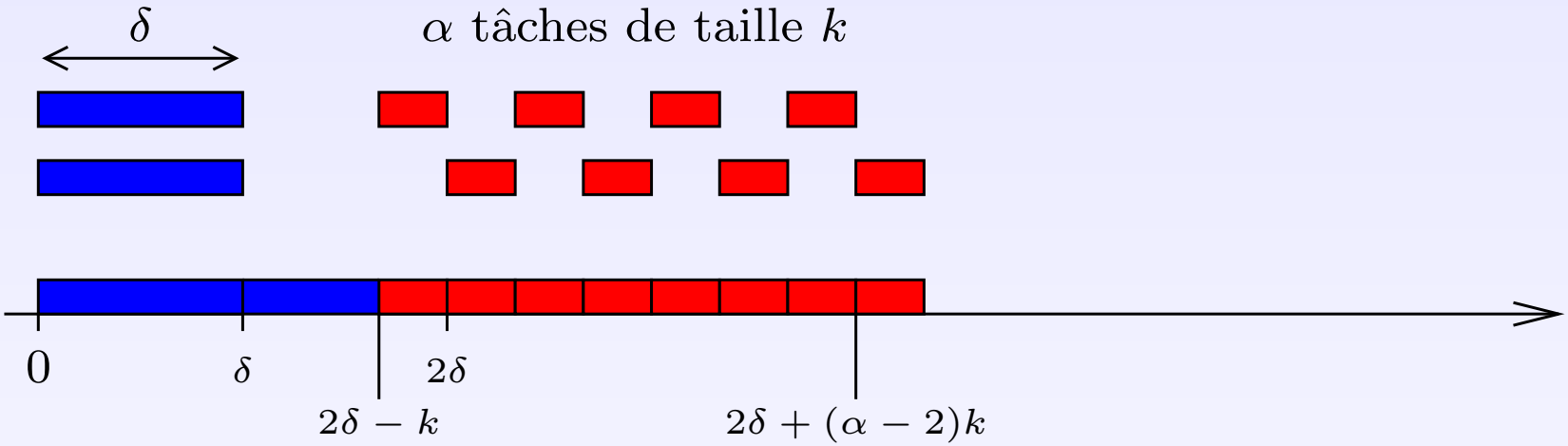
Achievable stretch: $\dfrac{(2\delta + jk) - (2\delta + (j-2)k)}{k} = 2.$

The adversary



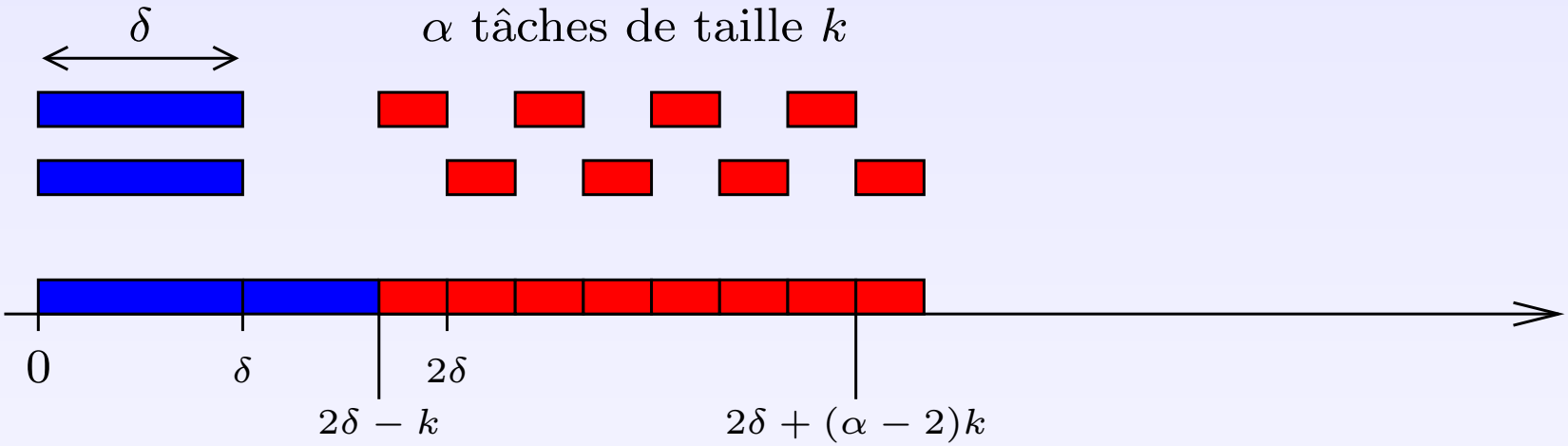
In practice: we do not know what happens after $2\delta - k$.

The adversary



We want to forbid this case (each size-k job being executed at its release date.

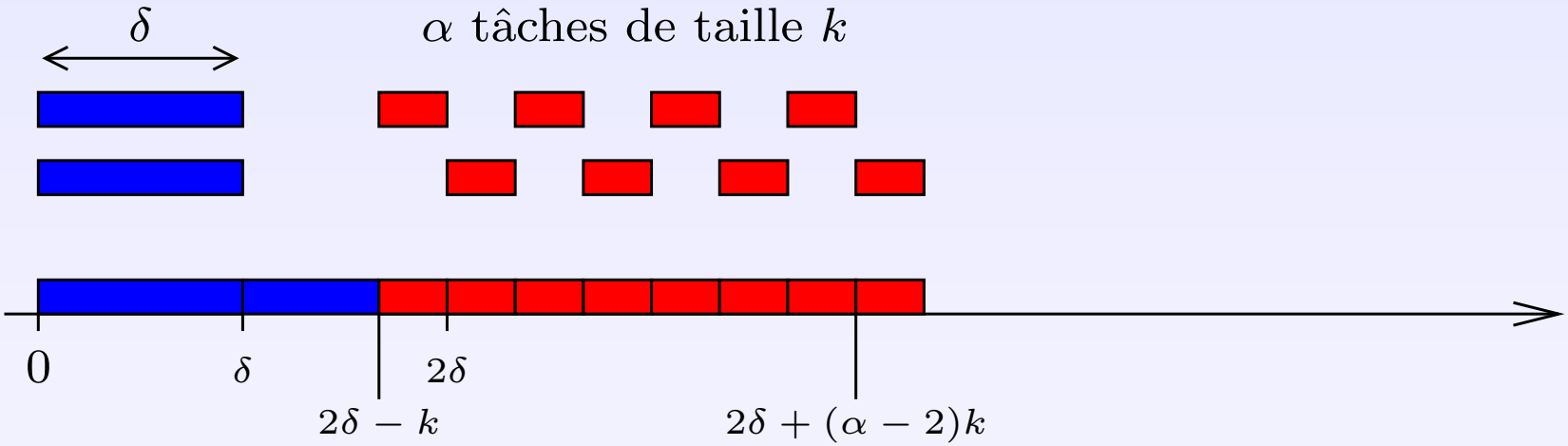
The adversary



We want to forbid this case (each size-k job being executed at its release date.

The algorithm being $\frac{1}{2}\Delta^{\sqrt{2}-1}$-competitive, T_1 and T_2 must be completed at the latest at time: $2 \cdot \frac{1}{2}\Delta^{\sqrt{2}-1} \cdot \delta = 2 \cdot \frac{1}{2}\left(\frac{\delta}{k}\right)^{\sqrt{2}-1} \cdot \delta$

The adversary

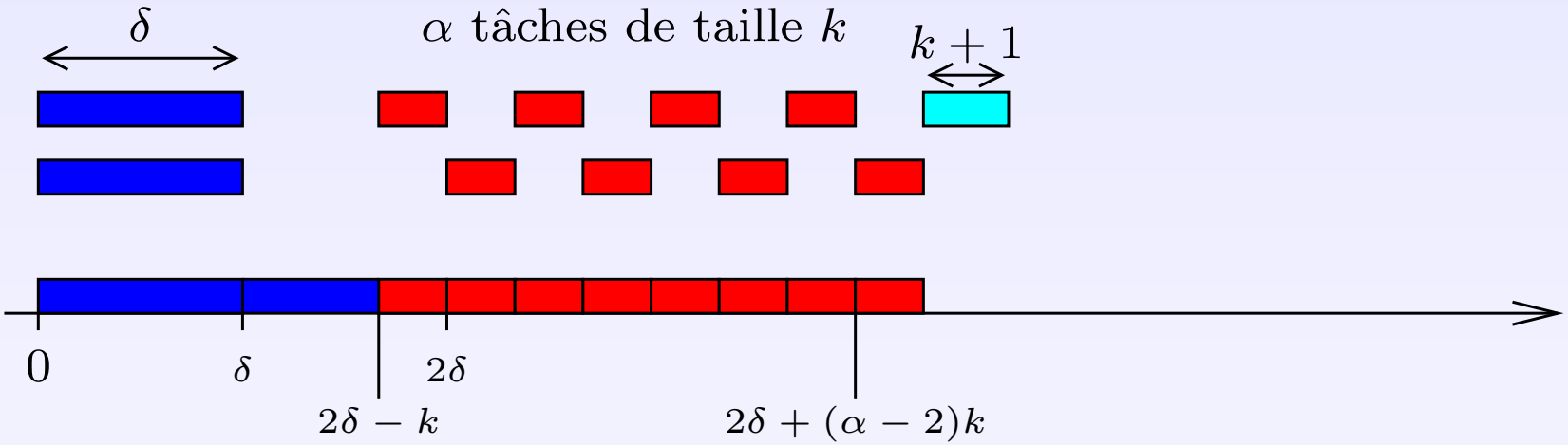


We want to forbid this case (each size-k job being executed at its release date.

The algorithm being $\frac{1}{2}\Delta^{\sqrt{2}-1}$-competitive, T_1 and T_2 must be completed at the latest at time: $2 \cdot \frac{1}{2}\Delta^{\sqrt{2}-1} \cdot \delta = 2 \cdot \frac{1}{2}\left(\frac{\delta}{k}\right)^{\sqrt{2}-1} \cdot \delta$

We let $\alpha = \lceil 1 + k - \frac{2\delta}{k} \rceil$ and then $2\delta + (\alpha - 1)k \geq 2 \cdot \frac{1}{2}\left(\frac{\delta}{k}\right)^{\sqrt{2}-1} \cdot \delta$.

The adversary

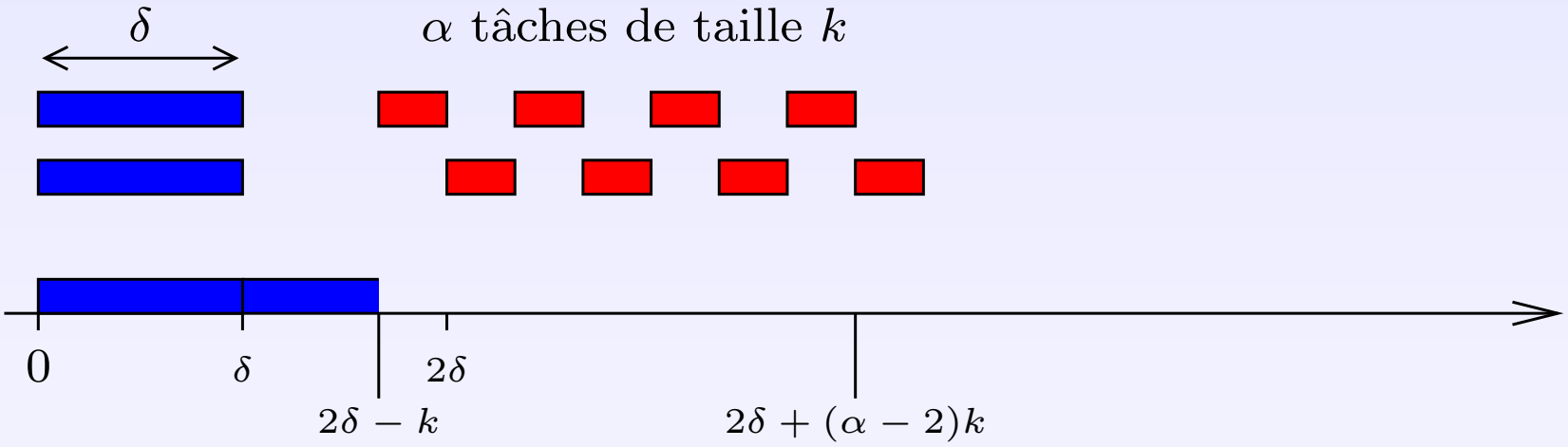


We want to forbid this case (each size-k job being executed at its release date.

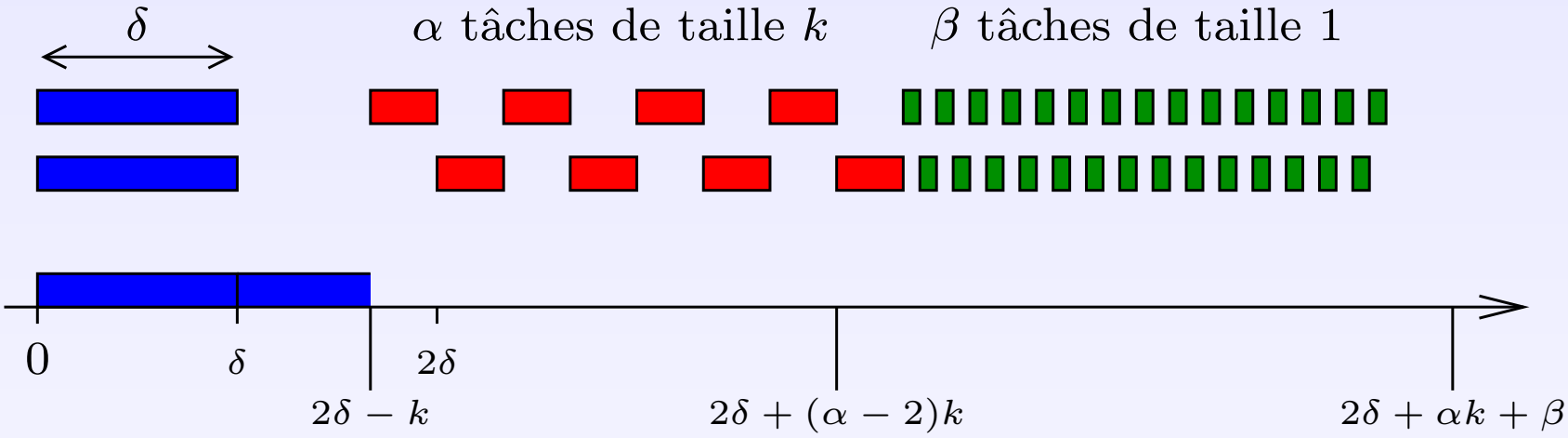
The algorithm being $\frac{1}{2}\Delta^{\sqrt{2}-1}$-competitive, T_1 and T_2 must be completed at the latest at time: $2 \cdot \frac{1}{2}\Delta^{\sqrt{2}-1} \cdot \delta = 2 \cdot \frac{1}{2}\left(\frac{\delta}{k}\right)^{\sqrt{2}-1} \cdot \delta$

We let $\alpha = \lceil 1 + k - \frac{2\delta}{k} \rceil$ and then $2\delta + (\alpha - 1)k \geq 2 \cdot \frac{1}{2}\left(\frac{\delta}{k}\right)^{\sqrt{2}-1} \cdot \delta$.

The adversary

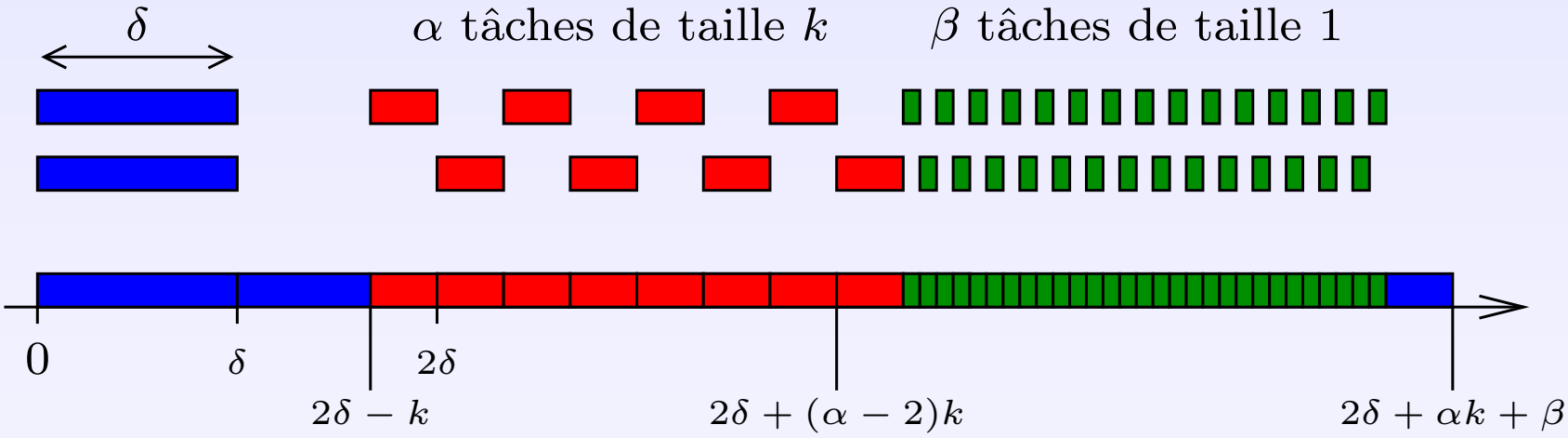


The adversary



The job $T_{2+\alpha+j}$ arrives at time $2\delta + (\alpha - 1)k + (j - 1)$.

The adversary



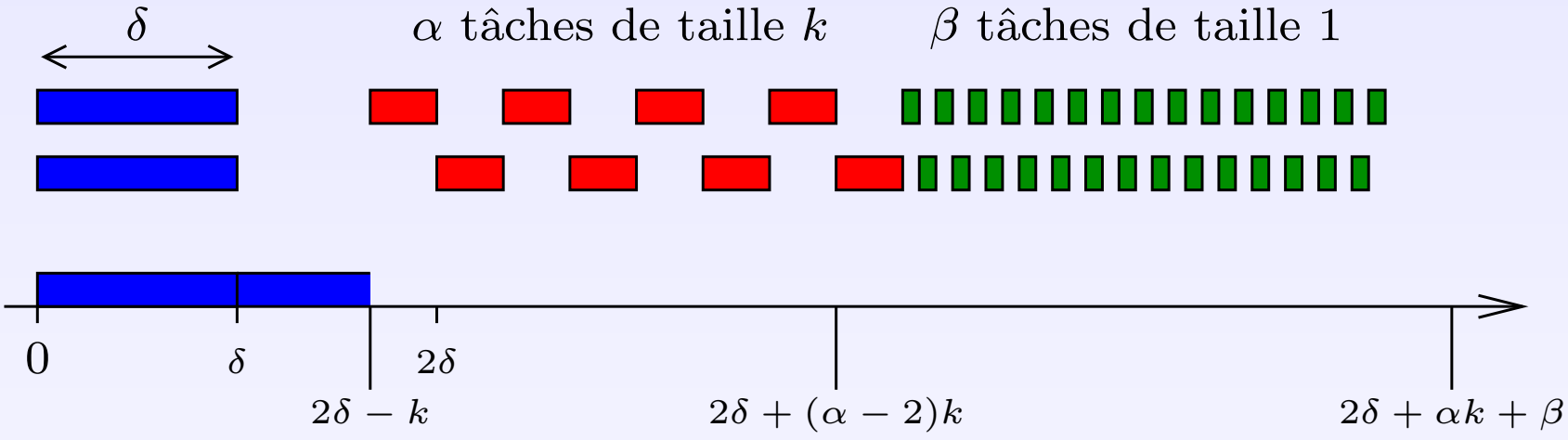
Achievable stretch (off-line)

Stretch of each job of size k or 1 : 1.

Stretch of T_1 or T_2: $\dfrac{2\delta + \alpha k + \beta}{\delta}$

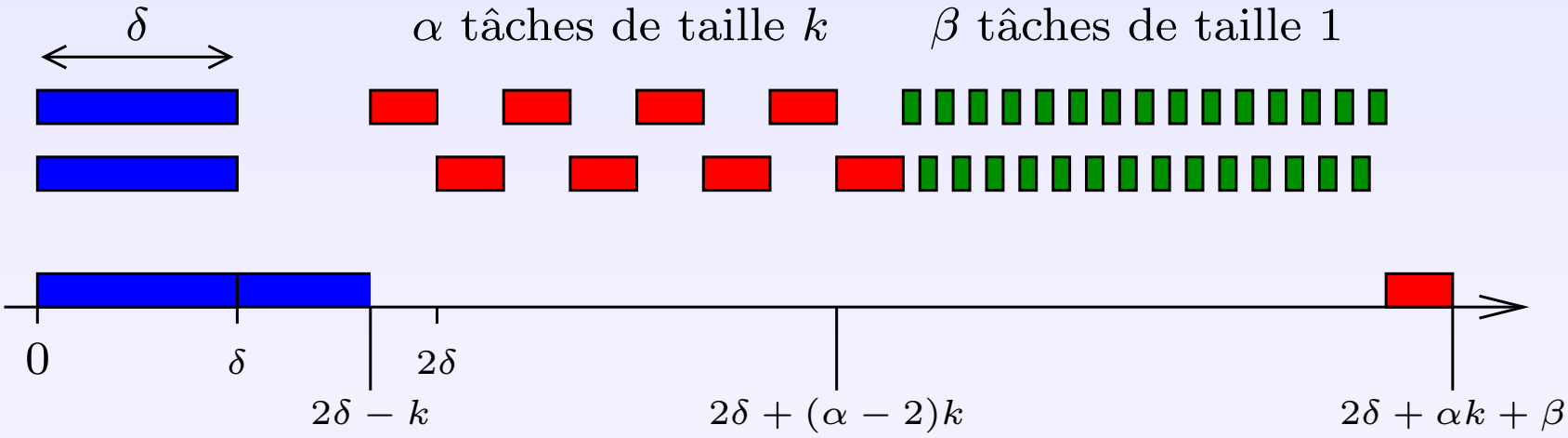
Optimal stretch $\leq \dfrac{2\delta + \alpha k + \beta}{\delta}$

The adversary



Achievable stretch (online)

The adversary

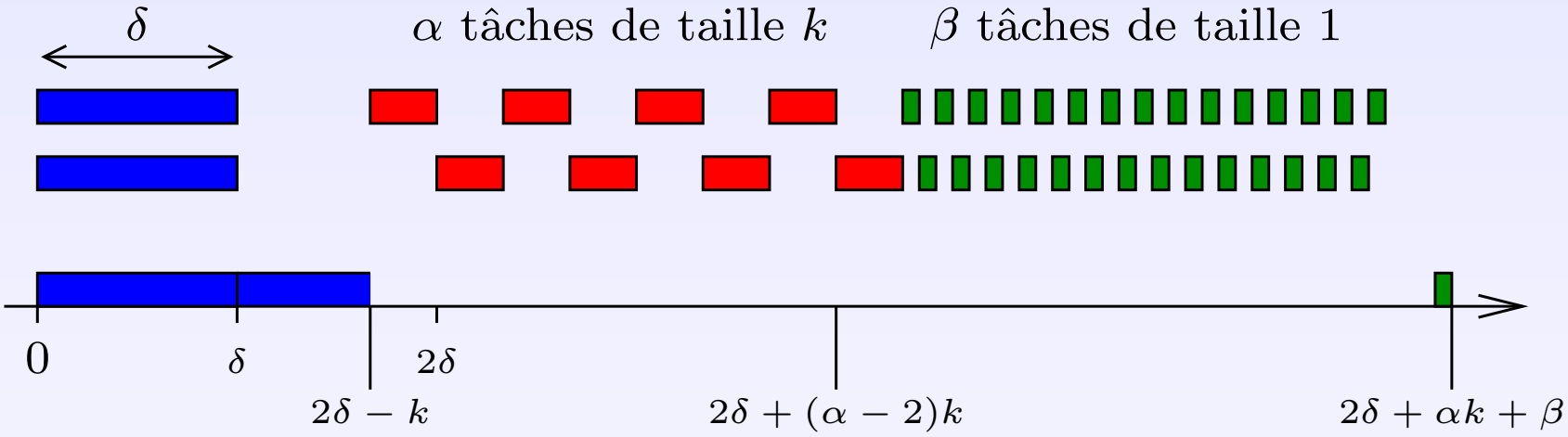


Achievable stretch (online)

The last completed job is of size k.

$$\text{Stretch} \geq \frac{(2\delta + \alpha k + \beta) - (2\delta + (\alpha - 2)k)}{k} = 2 + \frac{\beta}{k}.$$

The adversary

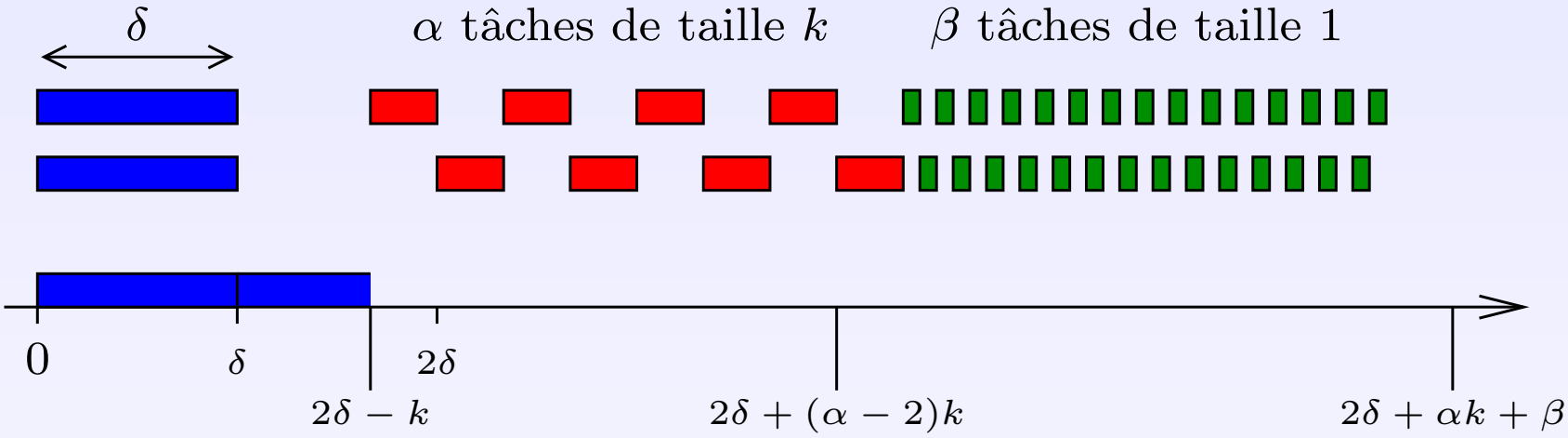


Achievable stretch (online)

The last completed job is of size 1.

$$\text{Stretch} \geq \frac{(2\delta + \alpha k + \beta) - (2\delta + (\alpha - 1)k + (\beta - 1))}{1} = k + 1.$$

The adversary



Achievable stretch (online)

Stretch $\geq \min\left\{2 + \frac{\beta}{k}, k + 1\right\}$

We let: $\beta = \lceil k(k-1) \rceil$

Then: stretch $\geq k + 1$.

The adversary: summing things up

$$\alpha = \left\lceil 1 + k - \frac{2\delta}{k} \right\rceil$$

$$\beta = \lceil k(k-1) \rceil$$

Optimal stretch $\leq \dfrac{2\delta + \alpha k + \beta}{\delta}$

Achieved stretch $\geq k + 1$.

The adversary: summing things up

$$\alpha = \left\lceil 1 + k - \frac{2\delta}{k} \right\rceil$$

$$\beta = \lceil k(k-1) \rceil$$

Optimal stretch $\leq \dfrac{2\delta + \alpha k + \beta}{\delta}$

Achieved stretch $\geq k + 1$.

We let $k = \delta^{2-\sqrt{2}}$

The adversary: summing things up

$$\alpha = \left\lceil 1 + k - \frac{2\delta}{k} \right\rceil$$

$$\beta = \lceil k(k-1) \rceil$$

Optimal stretch $\leq \dfrac{2\delta + \alpha k + \beta}{\delta}$

Achieved stretch $\geq k + 1$.

We let $k = \delta^{2-\sqrt{2}}$

Therefore $k + 1 > \left(\dfrac{1}{2}\delta^{\sqrt{2}-1}\right)\left(\dfrac{2\delta + \alpha k + \beta}{\delta}\right)$

Existing approximation algorithms

Two greedy approximation algorithms which are $\sqrt{\Delta}$-competitive:

Existing approximation algorithms

Two greedy approximation algorithms which are $\sqrt{\Delta}$-competitive:

1. Bender, Muthukrishnan, and Rajaraman (2002)
   For each job J_j, we define a pseudo-stretch $\widehat{\mathcal{S}}_j(t)$:

$$\widehat{\mathcal{S}}_j(t) = \begin{cases} \frac{t-r_j}{\sqrt{\Delta}} & \text{si } 1 \leq p_j \leq \sqrt{\Delta}, \\ \frac{t-r_j}{\Delta} & \text{si } \sqrt{\Delta} < p_j \leq \Delta. \end{cases}$$

   The jobs are scheduled by non increasing pseudo-stretch.

Existing approximation algorithms

Two greedy approximation algorithms which are $\sqrt{\Delta}$-competitive:

1. Bender, Muthukrishnan, and Rajaraman (2002)
   For each job J_j, we define a pseudo-stretch $\widehat{\mathcal{S}}_j(t)$:

$$\widehat{\mathcal{S}}_j(t) = \begin{cases} \frac{t-r_j}{\sqrt{\Delta}} & \text{si } 1 \leq p_j \leq \sqrt{\Delta}, \\ \frac{t-r_j}{\Delta} & \text{si } \sqrt{\Delta} < p_j \leq \Delta. \end{cases}$$

   The jobs are scheduled by non increasing pseudo-stretch.

2. Bender, Chahrabarti, and Muthukrishnan (1998).
   Each time a job arrives:
   - Compute the off-line max-stretch $\mathcal{S}$.
   - Jobs are scheduled *earliest deadline first* with the deadlines defined by $\sqrt{\Delta} \times \mathcal{S}$.

   Problem : only tries to optimize the most constraining jobs.

A non guaranteed heuristic

Each time a job arrives:

1. Preempt the running job (if any).
2. Compute the best achievable max-stretch, $\mathcal{S}$, taking into account the already taken decisions.
3. With the deadlines and time intervals defined by the max-stretch $\mathcal{S}$, solve:

$$\text{MINIMIZE} \quad \sum_{j=1}^{n} w_j \left(\left(\sum_t \left(\sum_{i=1}^{m} \alpha_{i,j}^{(t)} \right) \frac{\sup I_t(\mathcal{S}) + \inf I_t(\mathcal{S})}{2} \right) r_j \right) \quad \text{, WHILE}$$

$$\begin{cases} \text{(0a)} & \forall i, \forall j, \forall t, \quad r_j \geq \sup I_t(\mathcal{S}) \Rightarrow \alpha_{i,j}^{(t)} = 0 \\ \text{(0b)} & \forall i, \forall j, \forall t, \quad d_j(\mathcal{S}) \leq \inf I_t(\mathcal{S}) \Rightarrow \alpha_{i,j}^{(t)} = 0 \\ \text{(0c)} & \forall t, \forall i, \quad \sum_j \alpha_{i,j}^{(t)}.c_{i,j} \leq \sup I_t(\mathcal{S}) - \inf I_t(\mathcal{S}) \\ \text{(0d)} & \forall j, \quad \sum_t \sum_i \alpha_{i,j}^{(t)} = 1 \end{cases}$$

(Pseudo-approximation of a rational relaxation of sum-stretch.)

No guarantee !

Conclusion

Minimizing the average stretch

- Off-line case: looks difficult.
- Online case: rather easy.

Minimizing the max-stretch

- Off-line case: in polynomial time.
- Online: very difficult.

and in practice ?

Outline

Simulation parameters

- **platforms** containing 3, 10, or 20 homogeneous clusters of 10 processors;
- **applications** with 3, 10, or 20 distinct databases;
- **availability of the databases** of 30%, 60%, or 90% each;
- **workload** of 0.75, 1.0, 1.25, 1.5, 2.0, or 3.0.

Simulation results

	Max-stretch			Sum-stretch		
	Mean	SD	Max	Mean	SD	Max
OFFLINE:	1.0000	0.0000	1.0000	1.3570	0.2658	2.4006
OFFLINEPARETO:	1.0000	0.0000	1.0000	1.2487	0.2301	3.0291
ONLINE:	1.0030	0.0129	1.1847	1.0386	0.0392	1.3554
ONLINE-EDF:	1.0030	0.0129	1.1847	1.0378	0.0384	1.3609
ONLINE-EGDF:	1.0295	0.0598	1.5476	1.0020	0.0054	1.0962
SWRPT:	1.0321	0.0644	1.6702	1.0003	0.0015	1.0363
SRPT:	1.0548	0.1022	1.9885	1.0043	0.0075	1.0894
SPT:	1.0483	0.0932	1.7630	1.0019	0.0050	1.0876
BENDER98:	1.0377	0.0922	2.1521	1.0027	0.0070	1.0737
BENDER02:	2.7554	2.5492	21.4543	1.1786	0.2882	4.9793
MCT:	35.1396	23.0246	145.9341	45.6123	33.4706	148.0354
FCFS-DIV:	4.6077	6.1197	56.5935	1.3342	0.7059	12.9306
RAND:	4.1106	6.1035	62.2086	1.2149	0.4912	10.8549

Table: Aggregate statistics over all 162 platform/application configurations.

Simulation results (on one processor)

	Max-stretch			Sum-stretch		
	Mean	SD	Max	Mean	SD	Max
OFFLINE	1.0000	0.0000	1.0000	1.0413	0.0593	1.6735
ONLINE	1.0016	0.0149	1.6344	1.0549	0.0893	1.8134
SWRPT	1.1316	0.2071	3.1643	1.0001	0.0009	1.0398
SRPT	1.1242	0.2003	3.0753	1.0139	0.0212	1.2576
SPT	1.1961	0.2667	3.9752	1.0229	0.0296	1.3573
BENDER98	1.1200	0.1766	2.5428	1.0194	0.0279	1.4466
BENDER02	3.5422	2.4870	21.4819	2.9872	1.9599	15.0019
MCT	8.7762	9.1900	80.7465	6.8979	7.7409	88.2449
RAND	11.3059	11.1981	125.3726	5.8227	6.3942	68.0009

Table: Aggregate statistics for a single machine for all application configurations.

Outline

Conclusion

Minimizing the average stretch

- Off-line case: looks difficult.
- Online case: rather easy.

Minimizing the max-stretch

- Off-line case: in polynomial time.
- Online: very difficult.

In practice

- The approximation algorithms are out.
- SWRPT and our online heuristics are very efficient.
- SWRPT can induce starvation.
- Sum-stretch does not seem to be a pertinent metrics.

More generally...

1. Study of the (theoretical) off-line problem as the online solution will always be at best as good as the off-line.

2. Comparison of online and off-line solutions to quantify the quality of the online solutions.

3. Transposing results from a model to another one (from divisible loads to the model with preemption, back and forth).