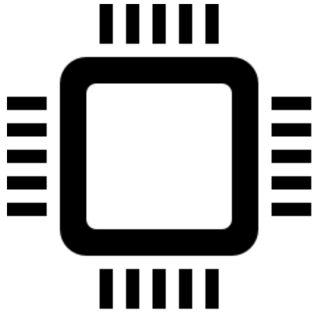
Interval Scheduling

Interval Scheduling

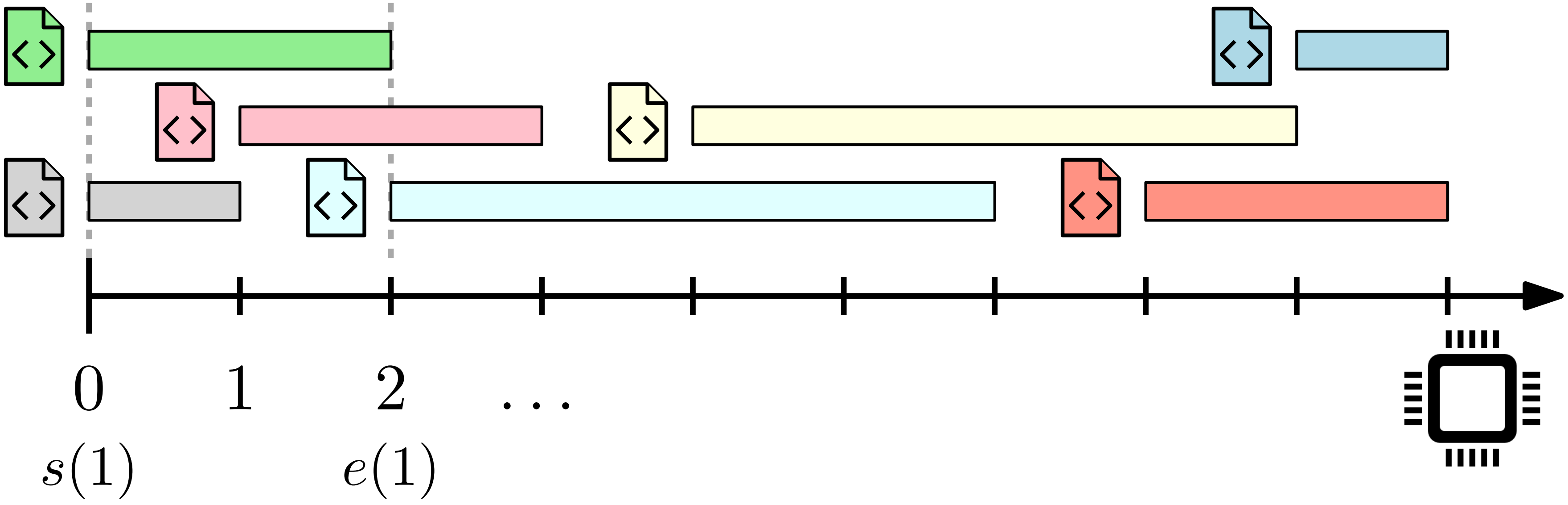
You need to compute a non-preemptive schedule on a supercomputer.

- There are n jobs indexed by $1, \ldots, n$ submitted for execution.
- Each job i has a desired start time $s(i)$ and a completion time $e(i) > s(i)$.
- Two jobs i and j are *compatible* if the intervals $[s(i), e(i))$ and $[s(j), e(j))$ are disjoint.

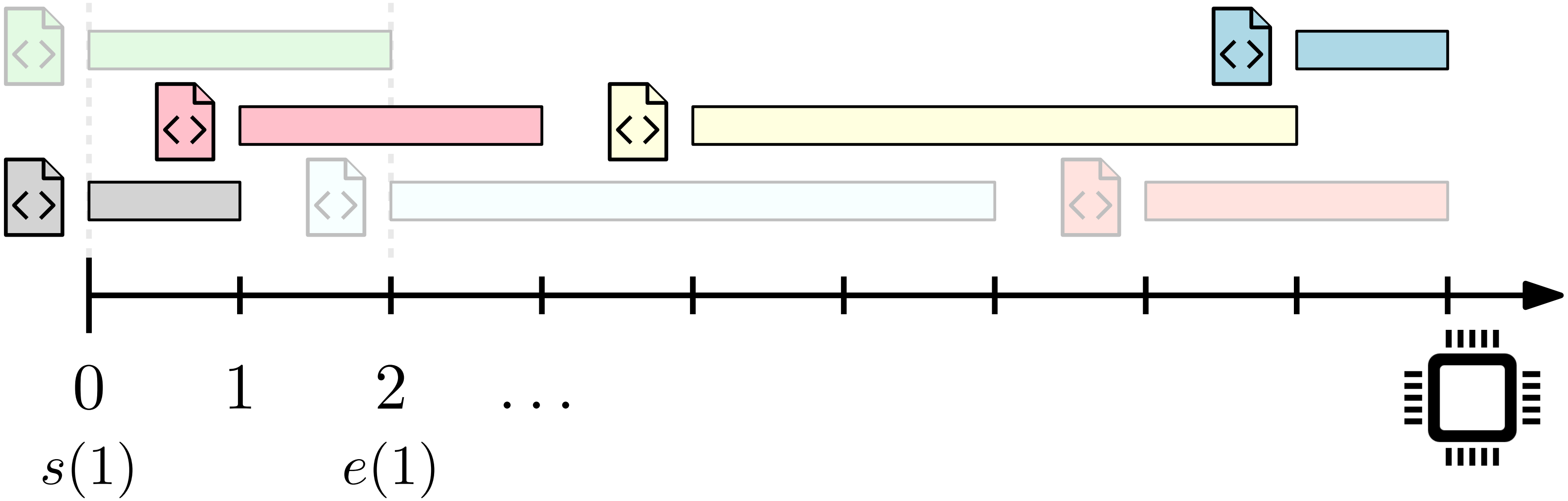
Goal: Find a subset of mutually compatible jobs of maximum cardinality.

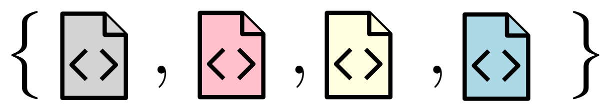


Example



Example



Optimal solution: 

Greedy template:

- Start with an empty set of jobs $R = \emptyset$.
- Examine jobs in some order.
  - When job i is examined: add i to R if it is compatible with all jobs j already in R.
- Finally, return R.

Greedy template:

- Start with an empty set of jobs $R = \emptyset$.
- Examine jobs in some order.
  - When job i is examined: add i to R if it is compatible with all jobs j already in R.
- Finally, return R.

Key question:
In what order should we process the jobs?

Some Possibilities:

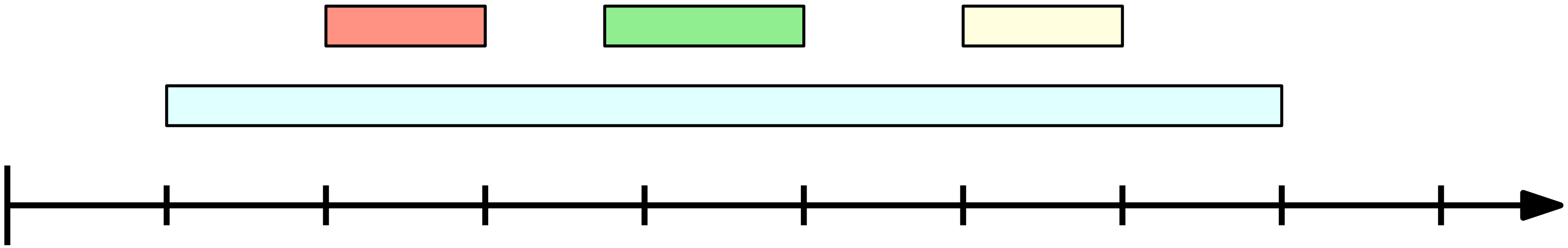
- **Earliest Start Time**: Increasing order of $s(i)$.

- **Earliest Finish Time**: Increasing order of $e(i)$.

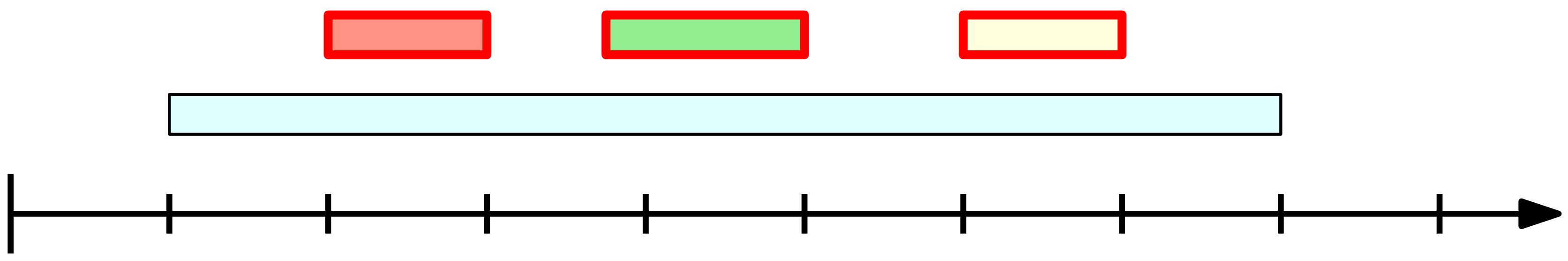
- **Shortest Interval**: Increasing order of $e(i) - s(i)$.

- **Fewest Conflicts**: Increasing order w.r.t. the number of conflicting jobs.

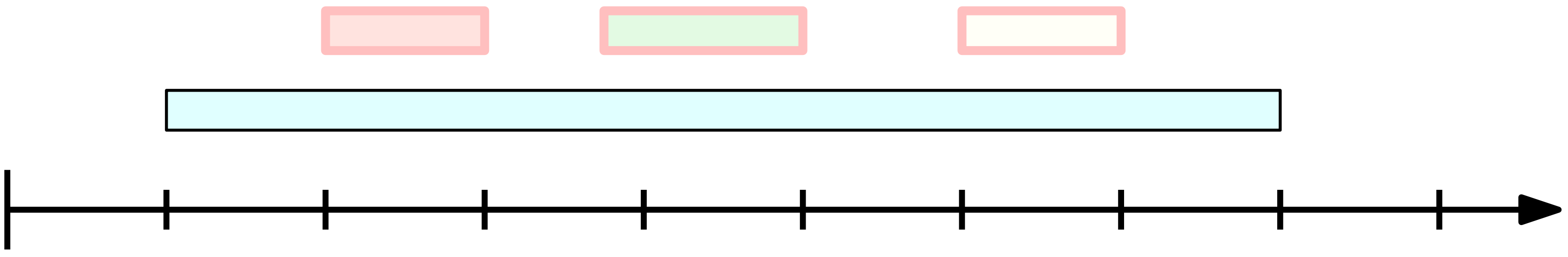
Earliest Start Time



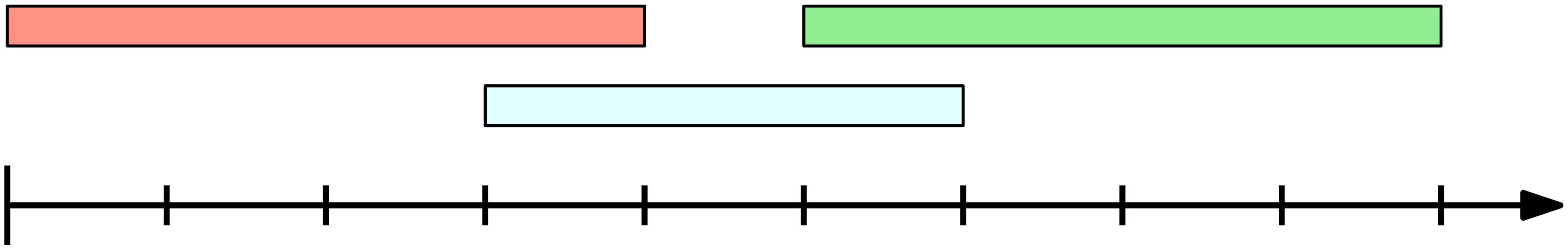
Earliest Start Time



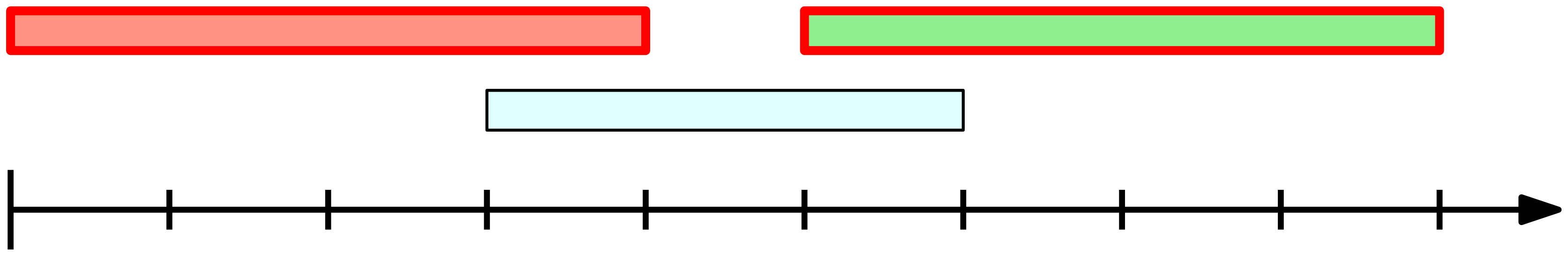
Earliest Start Time



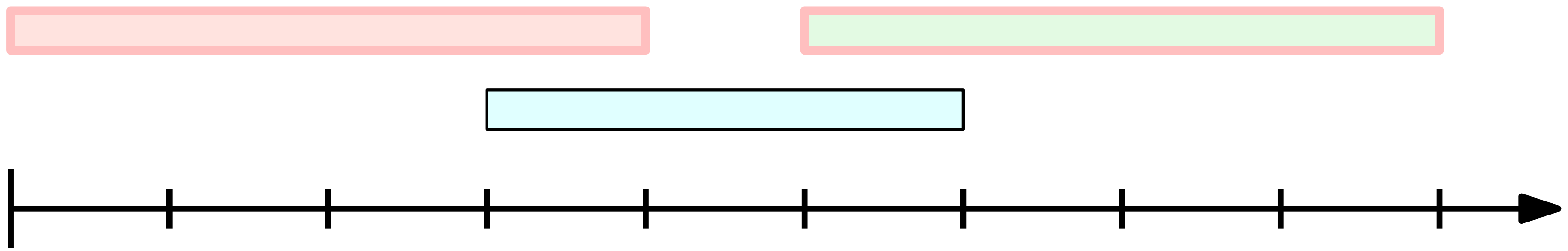
Shortest Interval



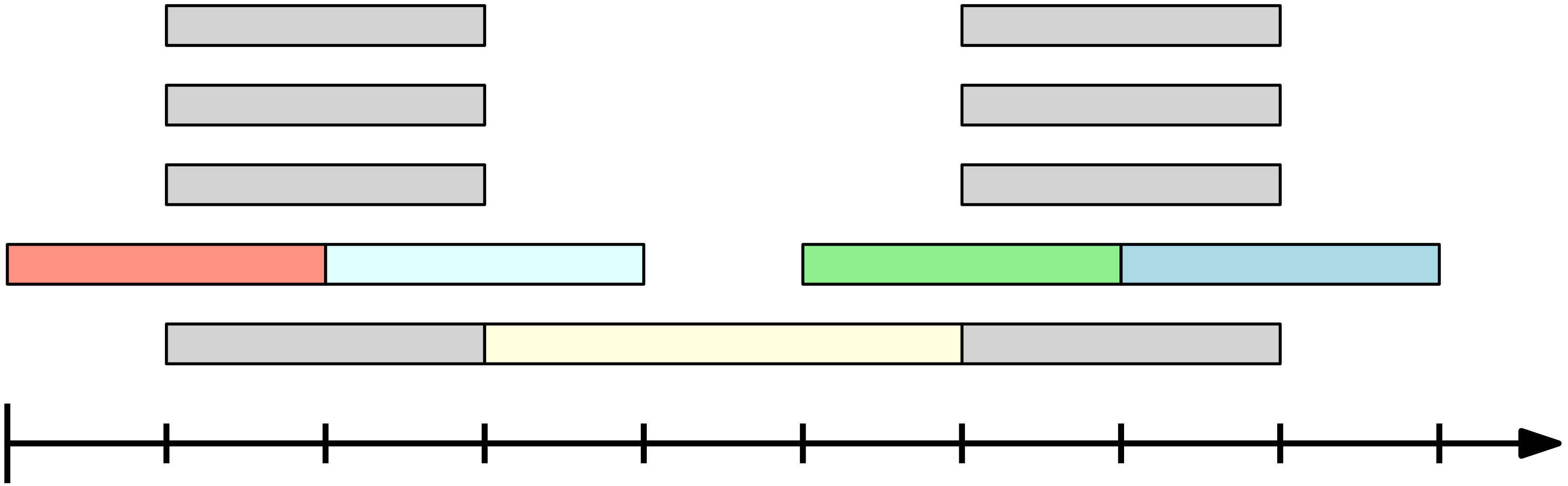
Shortest Interval



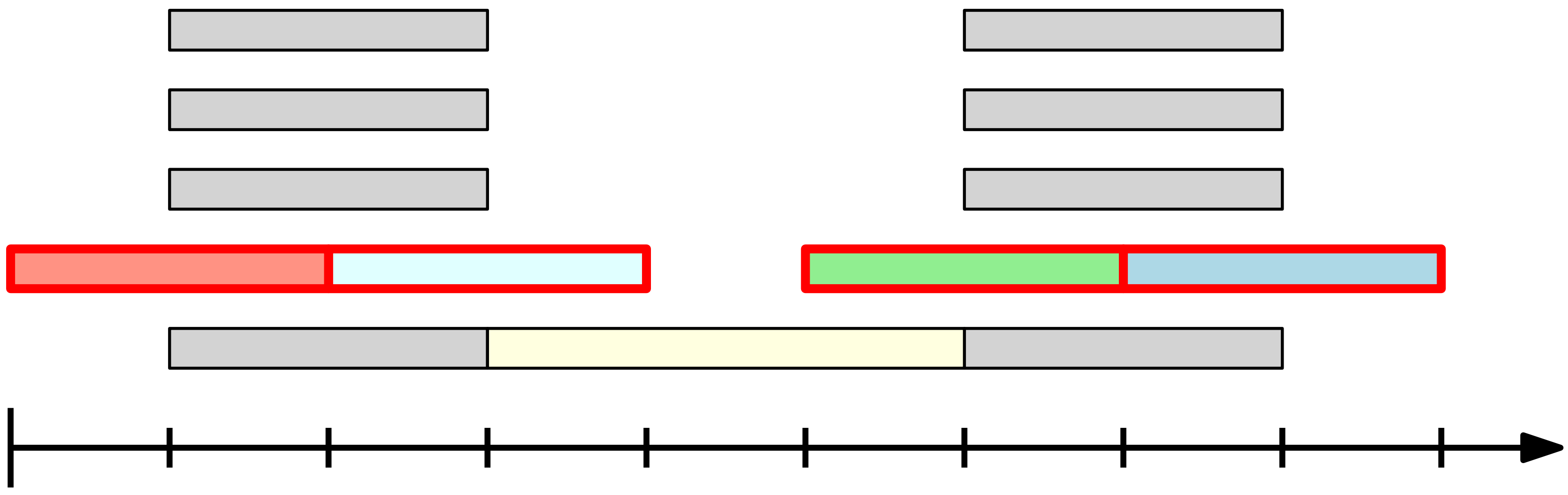
Shortest Interval



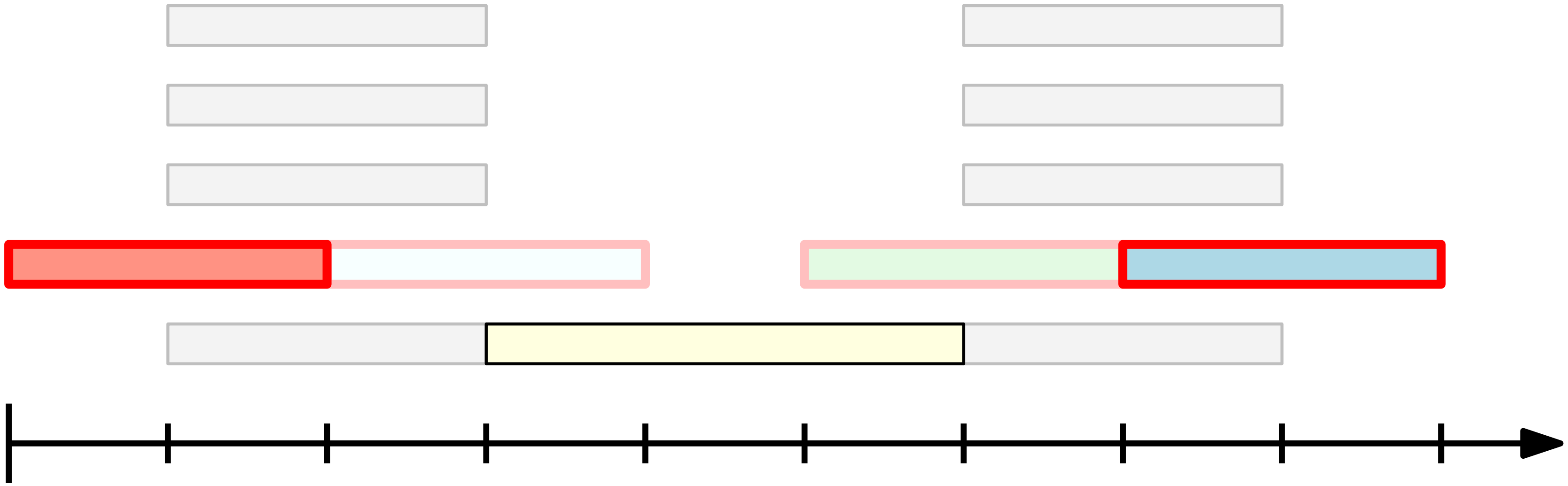
Fewest Conflicts



Fewest Conflicts



Fewest Conflicts



Some Possibilities:

- **Earliest Start Time**: Increasing order of $s(i)$.

- **Earliest Finish Time**: Increasing order of $e(i)$.

- **Shortest Interval**: Increasing order of $e(i) - s(i)$.

- **Fewest Conflicts**: Increasing order w.r.t. the number of conflicting jobs.

Earliest Finish Time

- Let $\mathcal{J} = \{1 \ldots, n\}$ be the set of jobs in input.
- $R \leftarrow \emptyset$
- While $\mathcal{J}$ is not empty:
  - Find a job $i \in \mathcal{J}$ minimizing $e(i)$.
  - Add i to R
  - Remove from $\mathcal{J}$ all jobs $j \in \mathcal{J}$ that are not compatible with i (including i itself).
- Return R

Observation: R is always a set of mutually compatible jobs.

EFT: Proof of Correctness

Let R^* be an optimal set of jobs.

Let $i_1, i_2, \ldots, i_m$ (resp. $i_1^*, i_2^*, \ldots, i_\ell^*$) be the indices of the jobs in R (resp. R^*), sorted w.r.t. $e(\cdot)$.

We want to prove $m = |R| \geq |R^*| = \ell$.

Claim: For $k = 1, \ldots, \ell$, index i_k exists and $e(i_k) \leq e(i_k^*)$.

EFT: Proof of Correctness

Let R^* be an optimal set of jobs.

Let $i_1, i_2, \ldots, i_m$ (resp. $i_1^*, i_2^*, \ldots, i_\ell^*$) be the indices of the jobs in R (resp. R^*), sorted w.r.t. $e(\cdot)$.

We want to prove $m = |R| \geq |R^*| = \ell$.

Claim: For $k = 1, \ldots, \ell$, index i_k exists and $e(i_k) \leq e(i_k^*)$.

Base case ($k = 1$):

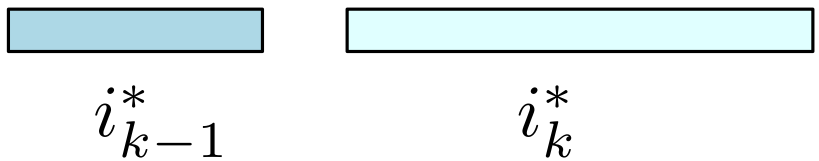
- Since $n \geq 1$, $\mathcal{J}$ is not empty before the first iteration, and i_1 exists.
- By the choice of i_1: $e(i_1) \leq \min_{j=1,\ldots,n} e(j) \leq e(i_1^*)$

EFT: Proof of Correctness

Claim: For $k = 1, \ldots, \ell$, index i_k exists and $e(i_k) \leq e(i_k^*)$.

Induction step $(k > 1)$:

- i_k^* is compatible with i_{k-1}^*, thus $e(i_{k-1}^*) \leq s(i_k^*)$

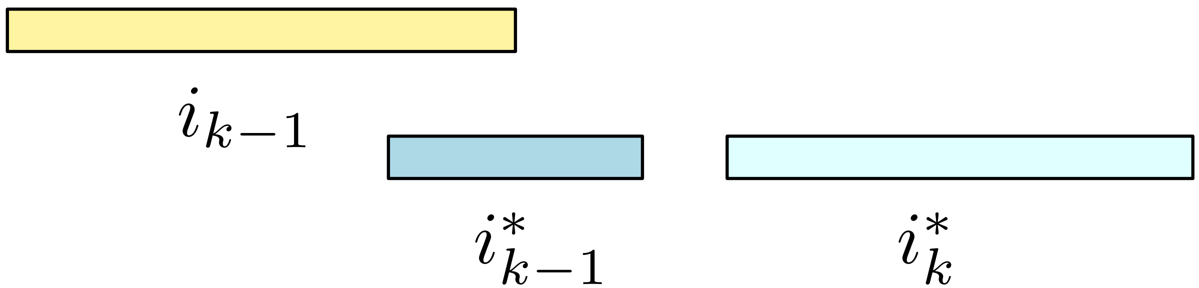


EFT: Proof of Correctness

Claim: For $k = 1, \ldots, \ell$, index i_k exists and $e(i_k) \leq e(i_k^*)$.

Induction step $(k > 1)$:

- i_k^* is compatible with i_{k-1}^*, thus $e(i_{k-1}^*) \leq s(i_k^*)$
- by induction hypothesis $e(i_{k-1}) \leq e(i_{k-1}^*)$

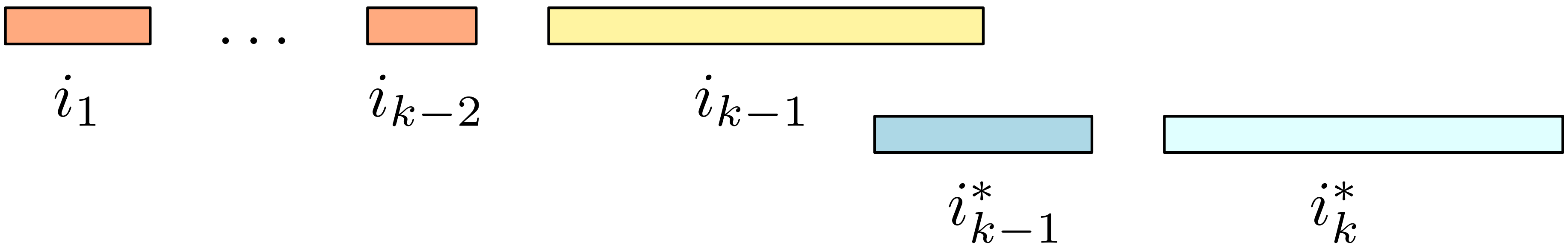


EFT: Proof of Correctness

Claim: For $k = 1, \ldots, \ell$, index i_k exists and $e(i_k) \leq e(i_k^*)$.

Induction step $(k > 1)$:

- i_k^* is compatible with i_{k-1}^*, thus $e(i_{k-1}^*) \leq s(i_k^*)$
- by induction hypothesis $e(i_{k-1}) \leq e(i_{k-1}^*)$
- Thefore, at the beginning ot the k-th iteration, $i_k^* \in \mathcal{J}$ since it is compatible with $i_1, \ldots, i_{k-1}$

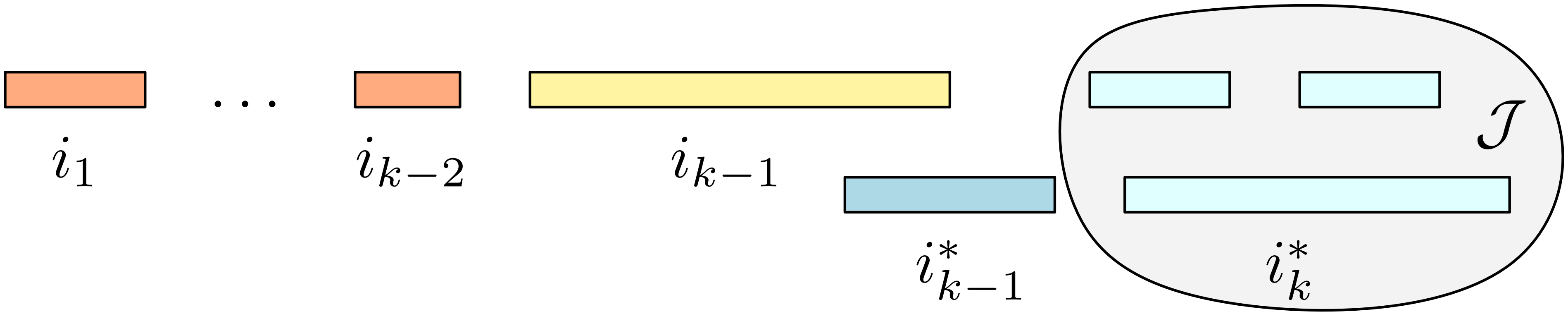


EFT: Proof of Correctness

Claim: For $k = 1, \ldots, \ell$, index i_k exists and $e(i_k) \le e(i_k^*)$.

Induction step $(k > 1)$:

- i_k^* is compatible with i_{k-1}^*, thus $e(i_{k-1}^*) \le s(i_k^*)$
- by induction hypothesis $e(i_{k-1}) \le e(i_{k-1}^*)$
- Thefore, at the beginning ot the k-th iteration, $i_k^* \in \mathcal{J}$ since it is compatible with $i_1, \ldots, i_{k-1}$
- $\mathcal{J} \neq \emptyset \implies \exists i_k$
- By the greedy choice: $e(i_k) = \min_{j \in \mathcal{J}} e(j) \le e(i_k^*)$. $\square$



EFT: Proof of Correctness

Claim: For $k = 1, \ldots, \ell$, index i_k exists and $e(i_k) \leq e(i_k^*)$.

Trick/Technique: Greedy Stays Ahead

At each step, the solution produced by greedy is not worse than the one produced by any other algorithm.

Implementing EFT

- Naive implementation: $O(n^2)$ time.

A better implementation:

- $\langle i_1, \ldots, i_n \rangle \leftarrow$ sort $\{1, \ldots, n\}$ w.r.t. $e(\cdot)$.
- Let $R = \emptyset$ be the current (partial) solution.
- Let $f = 0$ be the current finish time.
- For $j = 1, \ldots, n$:
    - If $s(i_j) \geq f$:
        - $R \leftarrow R \cup \{i_j\}$
        - $f \leftarrow e(i_j)$
- Return R

Implementing EFT

- Naive implementation: $O(n^2)$ time.

A better implementation:

- $\langle i_1, \ldots, i_n \rangle \leftarrow$ sort $\{1, \ldots, n\}$ w.r.t. $e(\cdot)$. $O(n \log n)$
- Let $R = \emptyset$ be the current (partial) solution.
- Let $f = 0$ be the current finish time.
- For $j = 1, \ldots, n$:
  - If $s(i_j) \geq f$:
    - $R \leftarrow R \cup \{i_j\}$
    - $f \leftarrow e(i_j)$

  (For loop: $O(n)$)
- Return R

Time complexity: $O(n \log n)$

Implementing EFT

```
struct job { int id; int start; int end; };
std::vector<job> jobs;

//[...] Read jobs

std::sort(jobs.begin(), jobs.end(), [](const job &j1, const job &j2)
                                    { return j1.end < j2.end; })

int f = 0;
std::vector<int> schedule;
for(const job &j : jobs)
{
    if(j.start >= f)
    {
        schedule.push_back(j.id);
        f = j.end;
    }
}

//schedule contains an optimal set of jobs
```

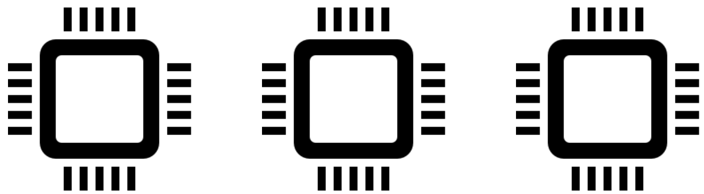
Interval Partitioning

Interval Partitioning

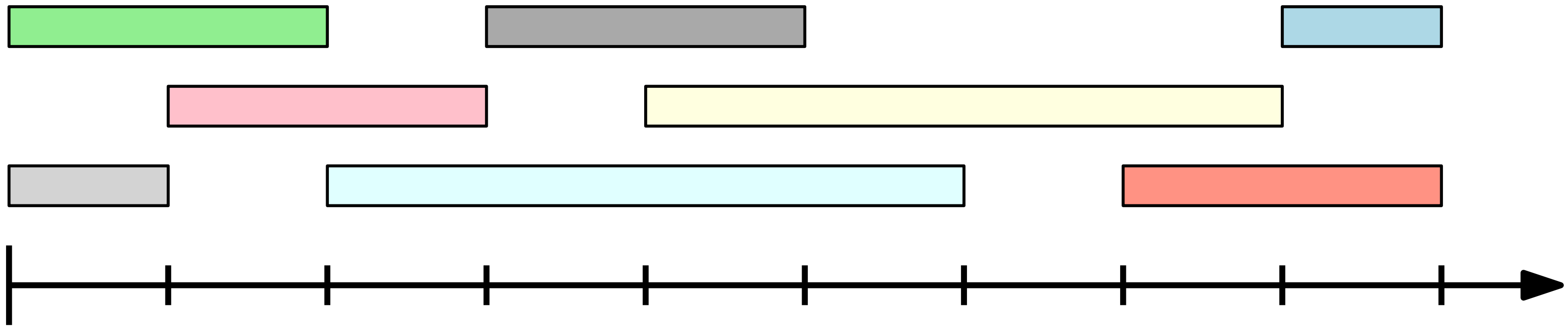
- There are n jobs indexed by $1, \ldots, n$.
- Each job i has a start time $s(i)$ and a completion time $e(i) > s(i)$.
- Two jobs i and j are *compatible* if the intervals $[s(i), e(i))$ and $[s(j), e(j))$ are disjoint.
- **All jobs must be executed**, but you can use k processors.
- Jobs scheduled on the same processor must be mutually compatible.

Goal: Minimize k.

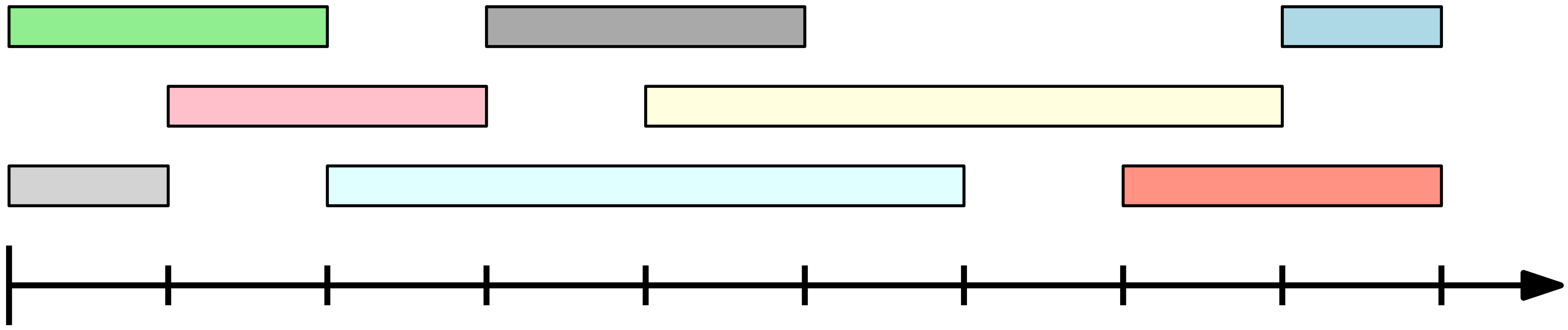
(and return the k corresponding schedules)



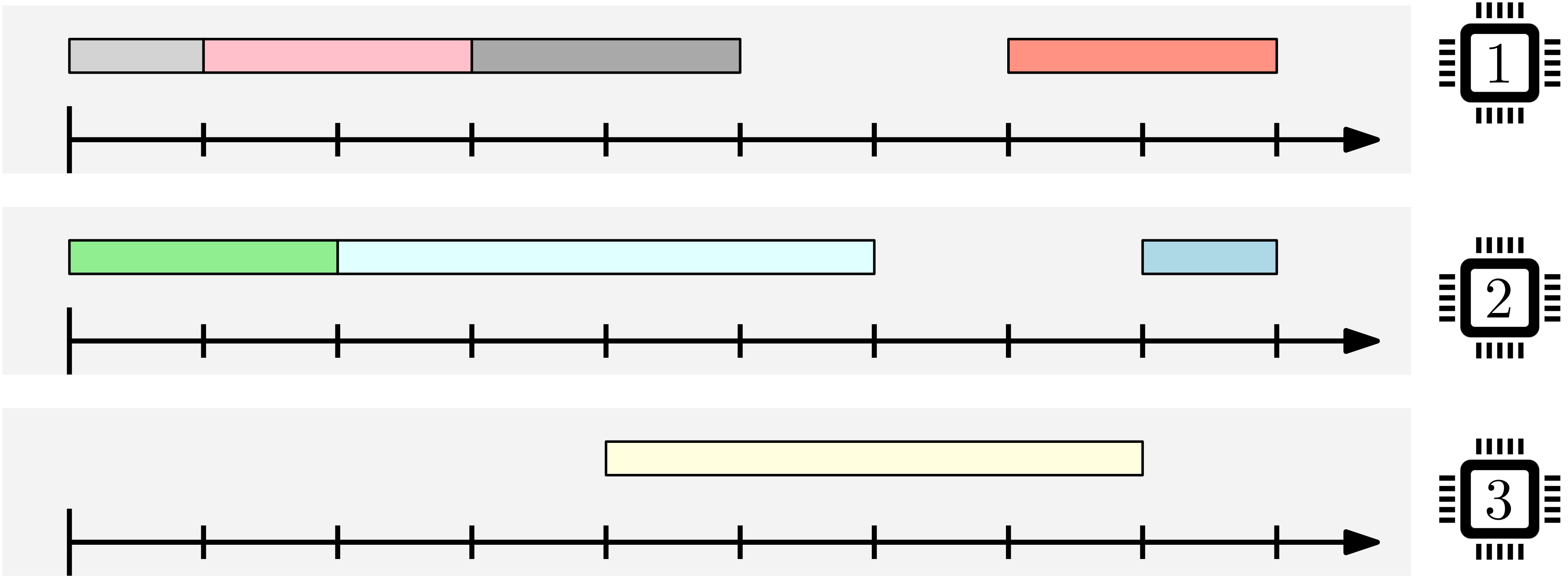
Example



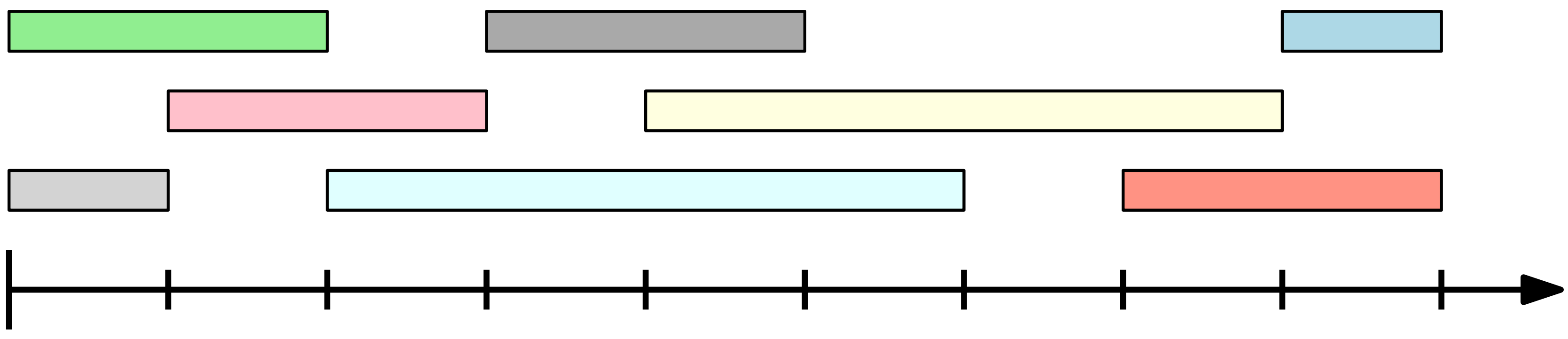
Example



$$k = 3$$

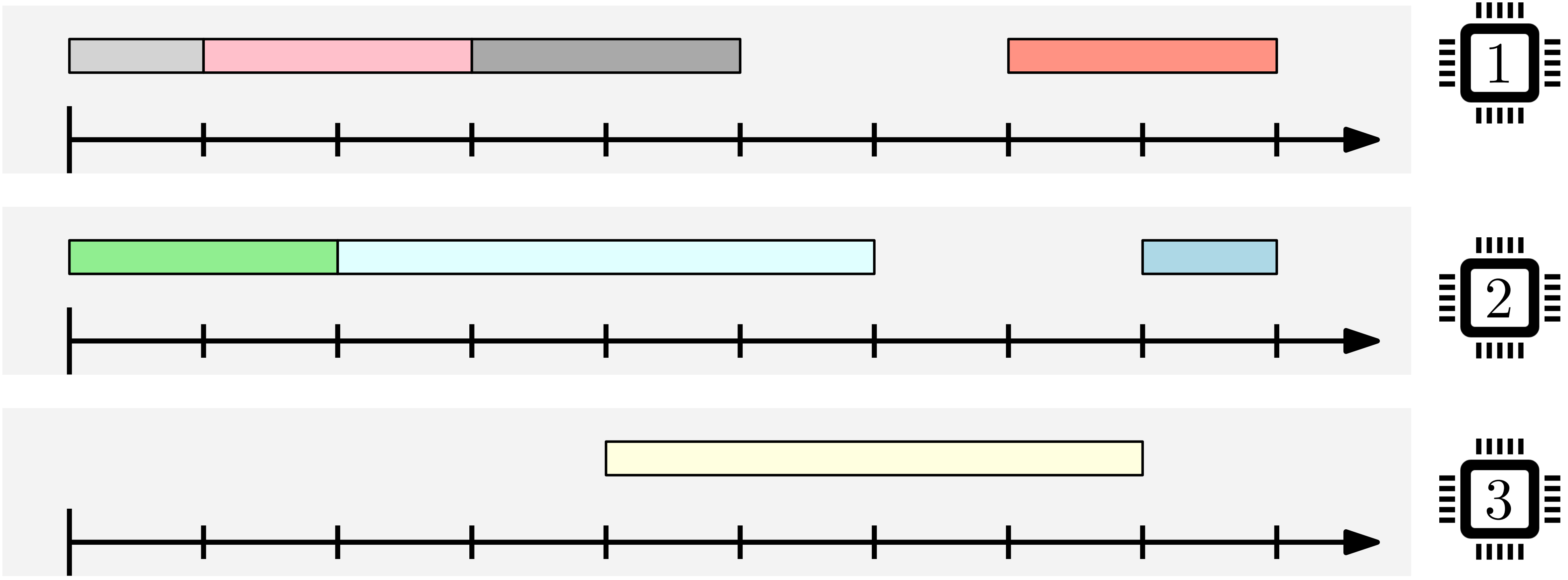


Example

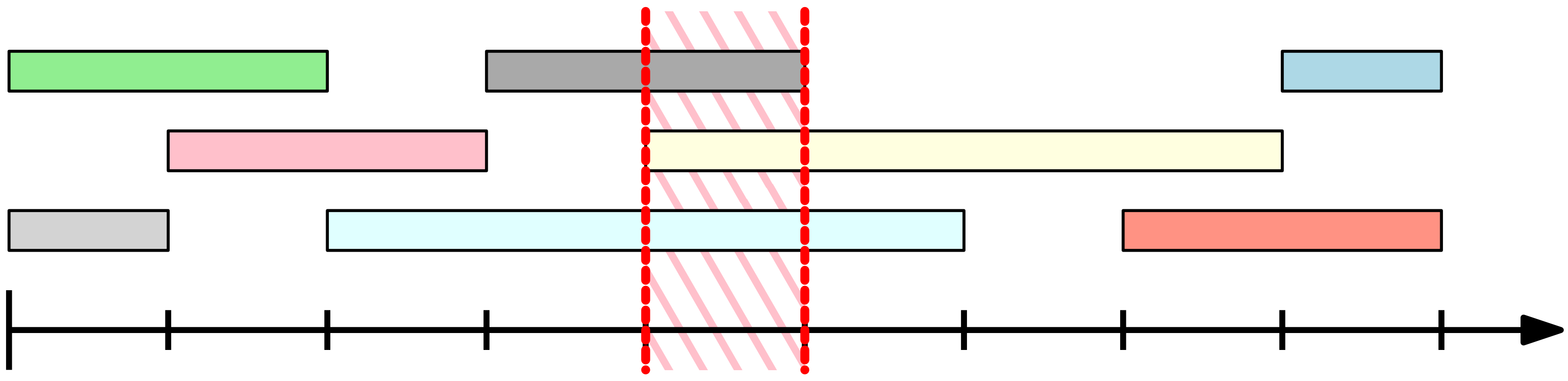


$k = 3$

Is $k = 3$ optimal?

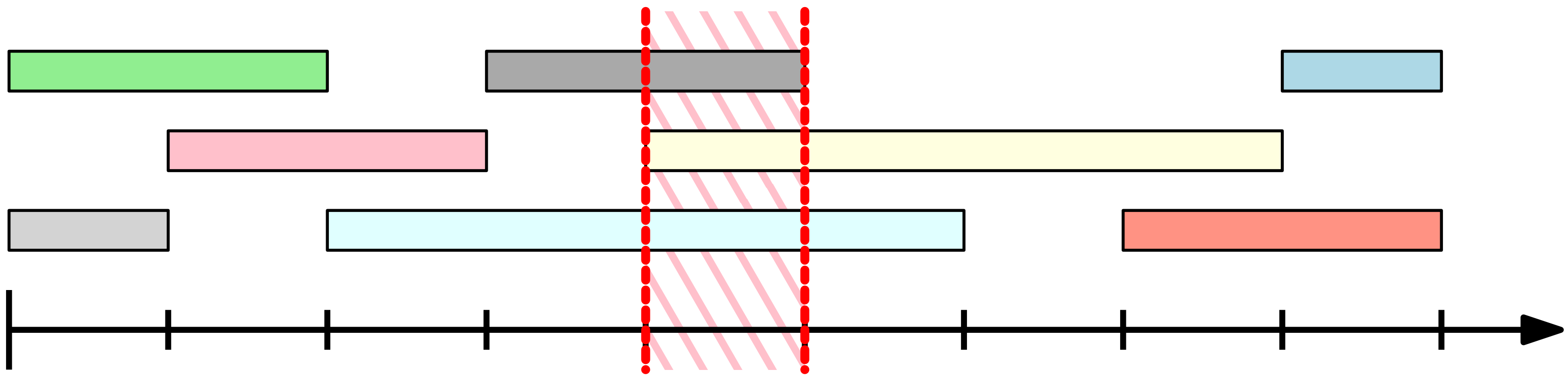


Is $k = 3$ optimal?



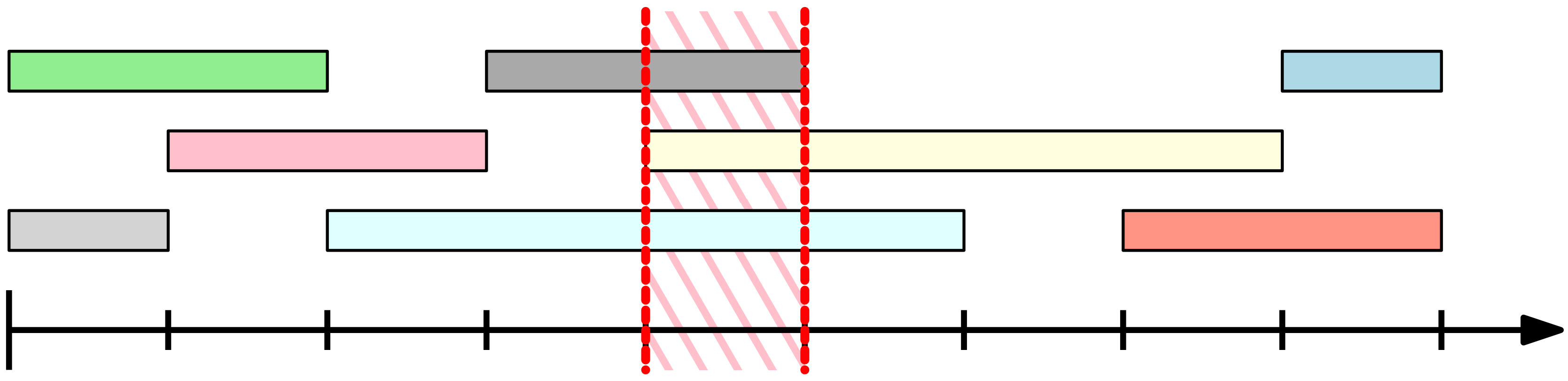
- **Observation:** There are 3 jobs that must be executed simultaneously.
- 3 is a lower bound to the optimal solution k^*.

Is $k = 3$ optimal?



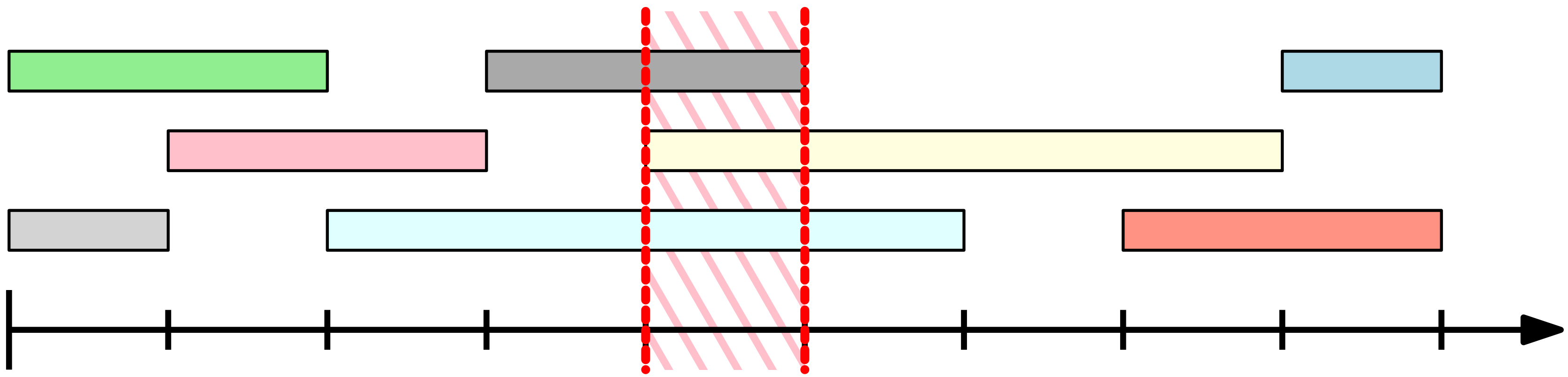
- **Observation:** There are 3 jobs that must be executed simultaneously.
- 3 is a lower bound to the optimal solution k^*.
- **Definition:** The *depth* D of a set of intervals is the maximum number of intervals $[s(i), e(i))$ that contain any single point.

Is $k = 3$ optimal?



- **Observation:** There are 3 jobs that must be executed simultaneously.
- 3 is a lower bound to the optimal solution k^*.
- **Definition:** The *depth* D of a set of intervals is the maximum number of intervals $[s(i), e(i))$ that contain any single point.
- **Observation:** $k^* \geq D$.

Is $k = 3$ optimal?



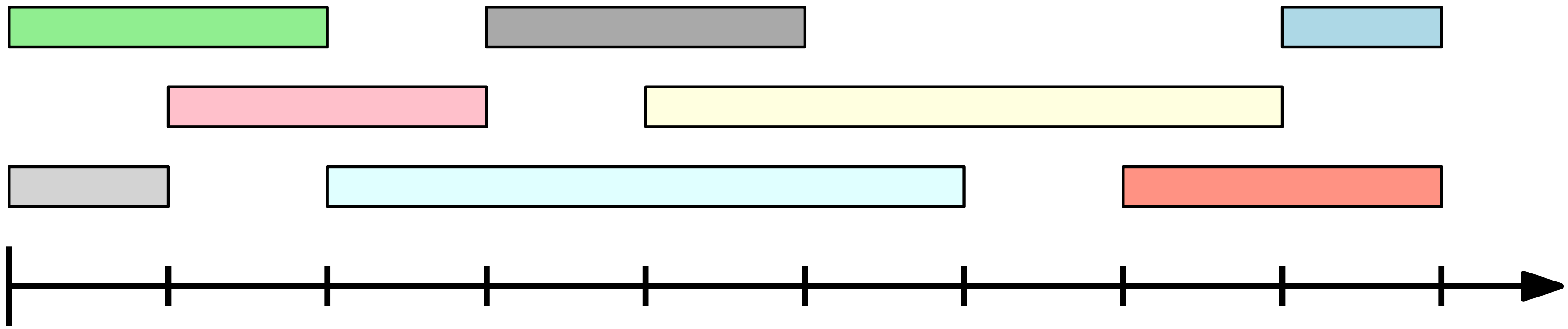
- **Observation:** There are 3 jobs that must be executed simultaneously.
- 3 is a lower bound to the optimal solution k^*.
- **Definition:** The *depth* D of a set of intervals is the maximum number of intervals $[s(i), e(i))$ that contain any single point.
- **Observation:** $k^* \geq D$.

Is $k^* \leq D$?

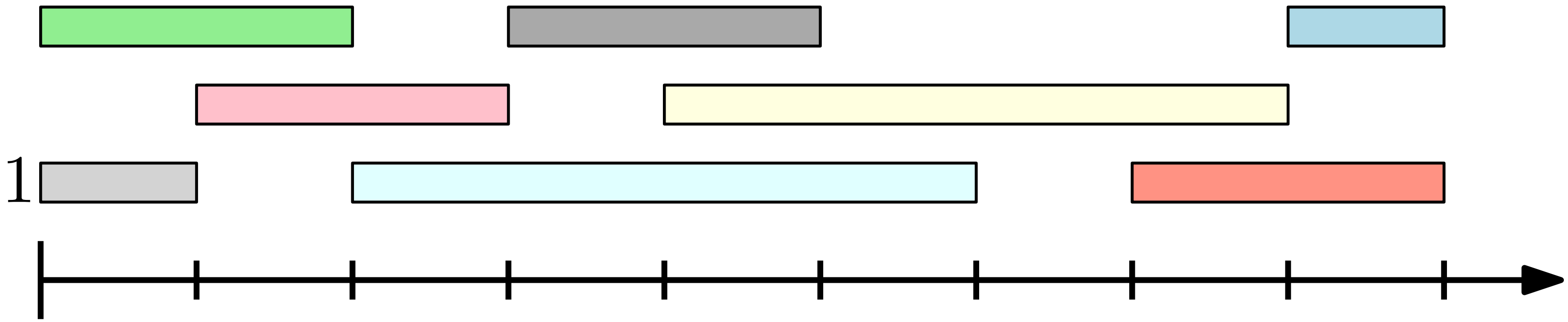
A greedy algorithm

- Assume that $\mathcal{J} = \{1, \ldots, n\}$ is sorted w.r.t. $s(\cdot)$.
- Each job $j \in \mathcal{J}$ will get a label $\ell(j) \in \mathbb{N}^+$.
- For $j = 1 \ldots, n$:
  - $C_j \leftarrow$ set of jobs in $1, \ldots, j-1$ that conflict with j.
  - $\ell(j) \leftarrow$ smallest positive integer not in $\{\ell(i) : i \in C_j\}$
- $k \leftarrow \max_{j=1,\ldots,n} \ell(j)$.
- Return a solution on k processors. The jobs assigned to the h-th processor are those in $\{i : \ell(i) = h\}$.

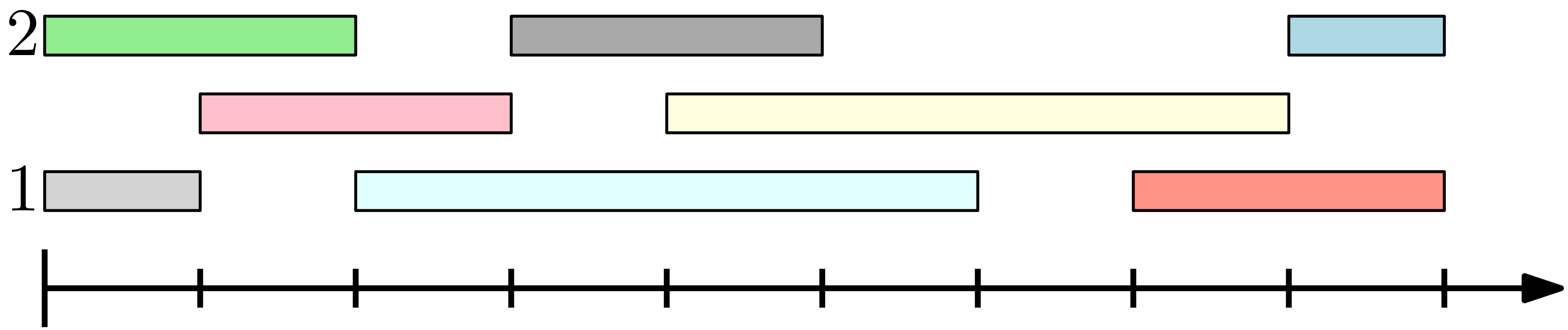
A greedy algorithm



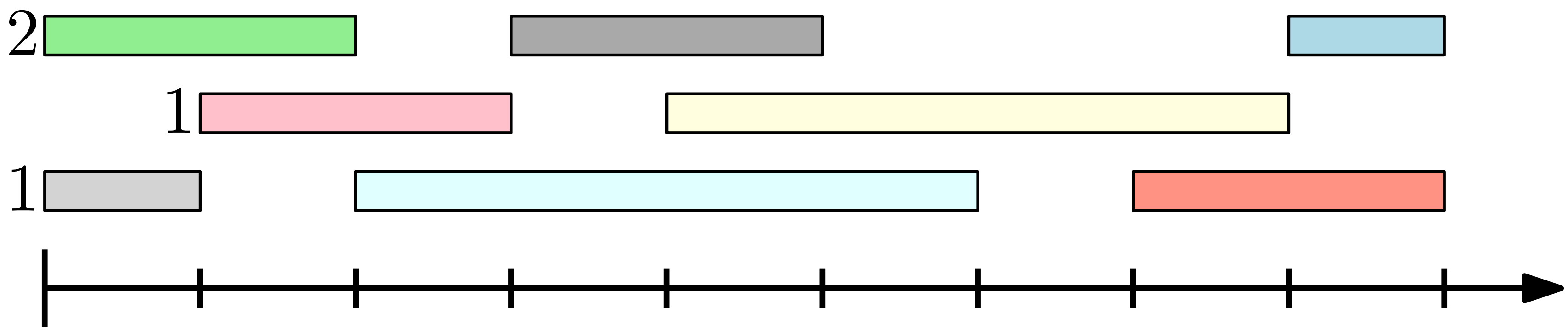
A greedy algorithm



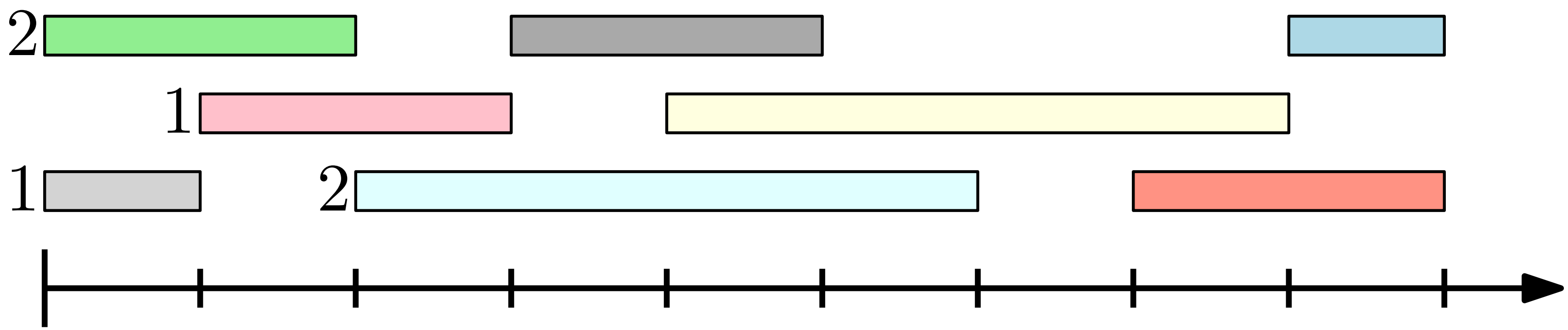
A greedy algorithm



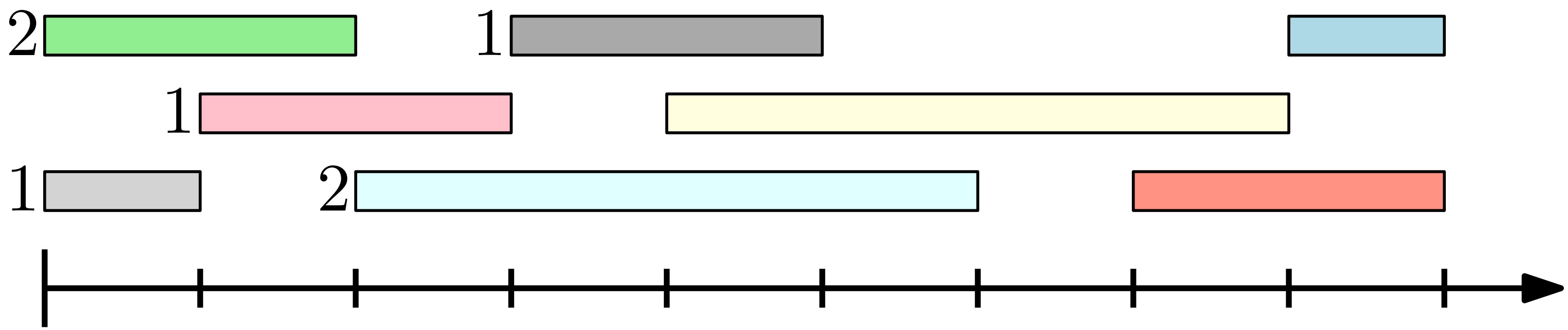
A greedy algorithm



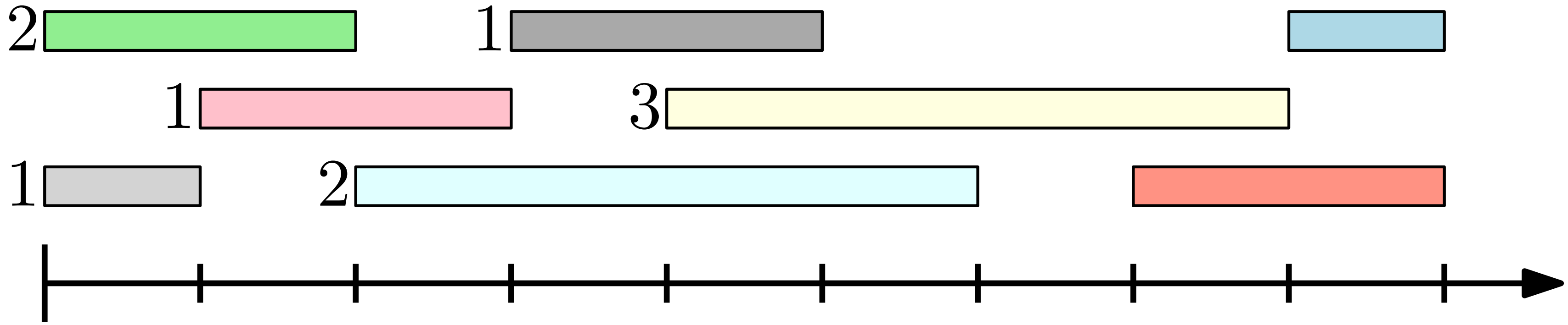
A greedy algorithm



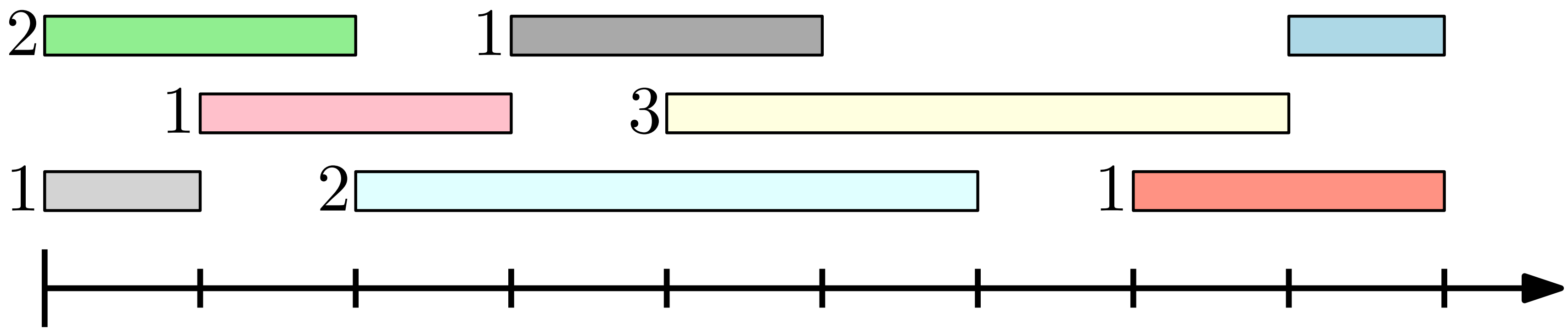
A greedy algorithm



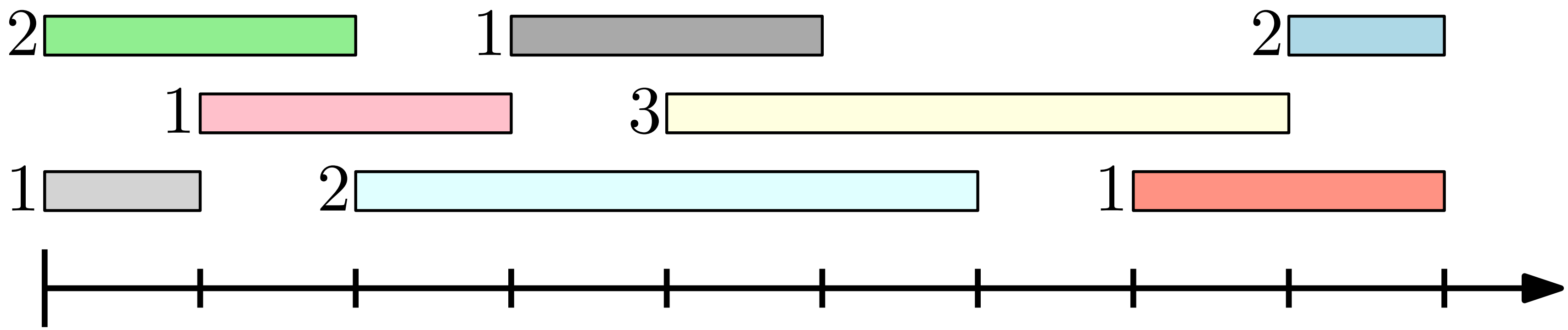
A greedy algorithm



A greedy algorithm



A greedy algorithm



Analysis

- **Observation:** No pair of overlapping intervals can get the same label $\implies$ all schedules consist of mutually compatible jobs.

Analysis

- **Observation:** No pair of overlapping intervals can get the same label $\implies$ all schedules consist of mutually compatible jobs.

- **Claim:** $k \le D$.
  - Let j be a job for which $\ell(j) = k$.
  - By the choice of $\ell(j)$: $1, \ldots, k-1 \in \{\ell(i) : i \in C_j\}$
  - For all $i \in C_j$, $e(i) > s(j)$, i.e., $s(j) \in [s(i), e(i))$.
  - $s(j)$ belongs to at least k intervals $\implies D \ge k$ $\square$

Analysis

- **Observation:** No pair of overlapping intervals can get the same label $\implies$ all schedules consist of mutually compatible jobs.

- **Claim:** $k \le D$.
  - Let j be a job for which $\ell(j) = k$.
  - By the choice of $\ell(j)$: $1, \ldots, k-1 \in \{\ell(i) : i \in C_j\}$
  - For all $i \in C_j$, $e(i) > s(j)$, i.e., $s(j) \in [s(i), e(i))$.
  - $s(j)$ belongs to at least k intervals $\implies D \ge k$ □

$$k^* \le k \le D$$

Analysis

- **Observation:** No pair of overlapping intervals can get the same label $\implies$ all schedules consist of mutually compatible jobs.

- **Claim:** $k \leq D$.
  - Let j be a job for which $\ell(j) = k$.
  - By the choice of $\ell(j)$: $1, \ldots, k-1 \in \{\ell(i) : i \in C_j\}$
  - For all $i \in C_j$, $e(i) > s(j)$, i.e., $s(j) \in [s(i), e(i))$.
  - $s(j)$ belongs to at least k intervals $\implies D \geq k$ $\square$

$$k^* \leq k \leq D$$

$$D \leq k^*$$

Analysis

- **Observation:** No pair of overlapping intervals can get the same label $\implies$ all schedules consist of mutually compatible jobs.

- **Claim:** $k \leq D$.
  - Let j be a job for which $\ell(j) = k$.
  - By the choice of $\ell(j)$: $1, \ldots, k-1 \in \{\ell(i) : i \in C_j\}$
  - For all $i \in C_j$, $e(i) > s(j)$, i.e., $s(j) \in [s(i), e(i))$.
  - $s(j)$ belongs to at least k intervals $\implies D \geq k$ $\square$

$$\left.\begin{array}{r} k^* \leq k \leq D \\ D \leq k^* \end{array}\right\} \implies k = k^* = D$$

Analysis

- **Observation:** $k^* \geq D$.
- **Claim:** $k \leq D$.

Trick/Technique: Finding Structural Properties

Find a structural property that implies optimality. (e.g., a lower bound to the measure of an optimal solution). Prove that greedy returns a solution with that property.

A possible implementation

- Every starting time $s(j)$ or finish time $e(j)$ of a job j is an *event* $\langle s(j), j \rangle$ or $\langle e(j), j \rangle$.

  $O(n \log n)$

- Create a sorted list of events. (break ties in favor of ending events)
- $k \leftarrow 0$ (number distinct labels)
- Mantain a min-heap H. (stores unused labels in $\{1, \ldots, k\}$)

A possible implementation

- Every starting time $s(j)$ or finish time $e(j)$ of a job j is an *event* $\langle s(j), j \rangle$ or $\langle e(j), j \rangle$. $O(n \log n)$
- Create a sorted list of events. (break ties in favor of ending events)
- $k \leftarrow 0$ (number distinct labels)
- Mantain a min-heap H. (stores unused labels in $\{1, \ldots, k\}$)
- For each event $\langle t, j \rangle$: $O(n)$
  - If $t = s(j)$
    - If H is empty, increment k and set $\ell(j) \leftarrow k$
    - Otherwise $\ell(j) \leftarrow$ pop from H $O(\log k)$
  - Otherwise $(t = e(j))$:
    - Push $\ell(j)$ into H. $O(\log k)$

A possible implementation

```
struct job { int id; int start; int end; };
std::vector<job> jobs;

//[...] Read jobs

std::vector<std::tuple<int, bool, int>> events;
for(const job &j : jobs)
{
    //Use second entry for tie breaking (false<true)
    events.push_back( std::make_tuple(j.start, true, j.id) );
    events.push_back( std::make_tuple(j.end, false, j.id) );
}

std::sort(events.begin(), events.end());
```

A possible implementation

```
int k=0;
std::vector<int> H; //A min-heap of available labels
std::vector<int> labels(jobs.size()); //Labels assigned to jobs
for(const auto &event : events)
{
    if(std::get<1>(event)) //Start event
    {
        if(H.empty())
            labels[std::get<2>(event)] = ++k;
        else
        {
            std::pop_heap(H.begin(), H.end(), std::greater<int>());
            labels[std::get<2>(event)] = H.back();
            H.pop_back();
        }
    }
    else //End event
    {
        H.push_back(labels[std::get<2>(event)]);
        std::push_heap(H.begin(),  H.end(), std::greater<int>());
    }
}
//labels[i] contains the label of job i
```

Minimizing Lateness

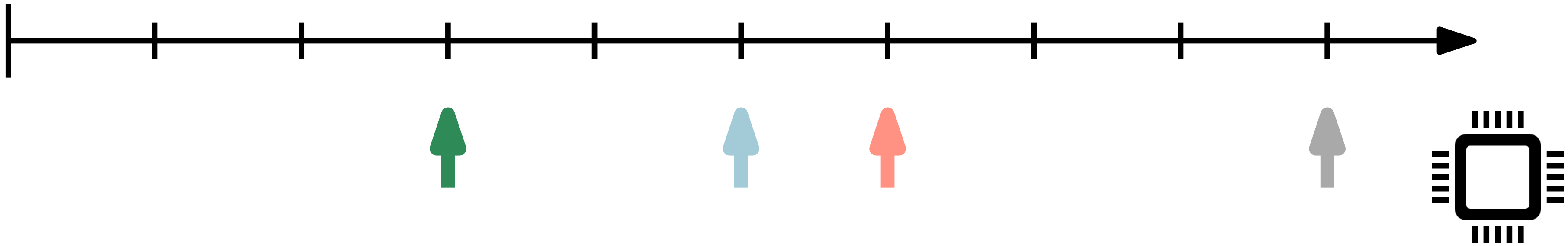
Minimizing Lateness

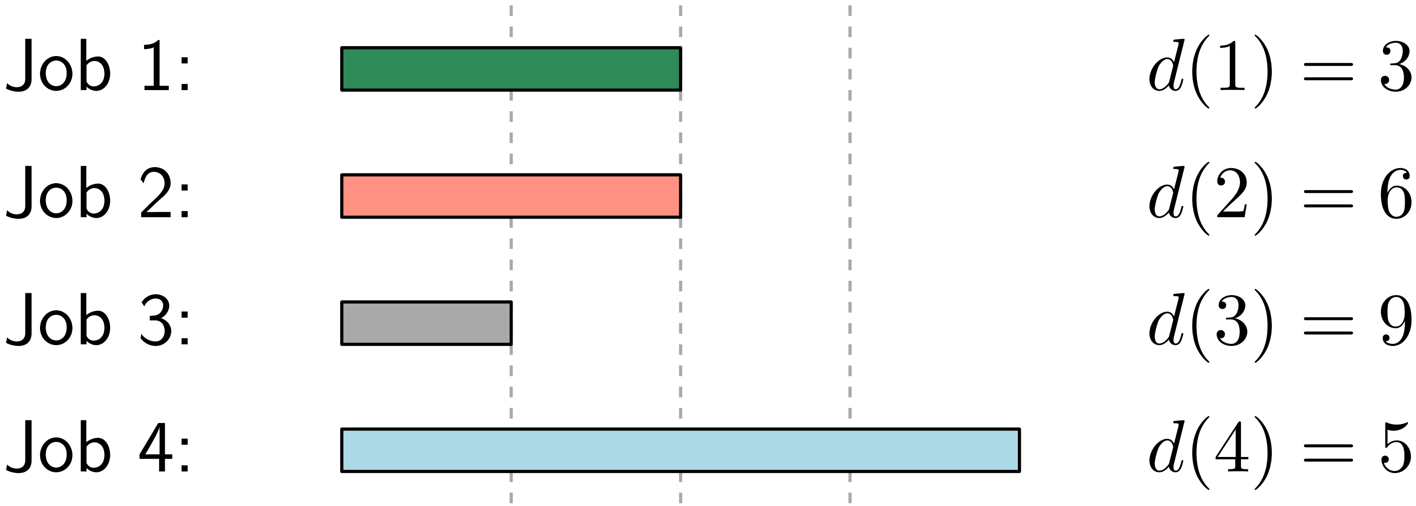
- There are n jobs indexed by $1, \ldots, n$.
- Each job i has a length $t(i)$ and a distinct deadline $d(i)$.
- All jobs have to be scheduled on a single processor (one at a time).
- If job i completes by time $f_i \leq d(i)$ its *lateness* ℓ_i is 0. Otherwise $\ell_i = f_i - d(i)$.

Goal: Find a schedule S minimizing the maximum lateness

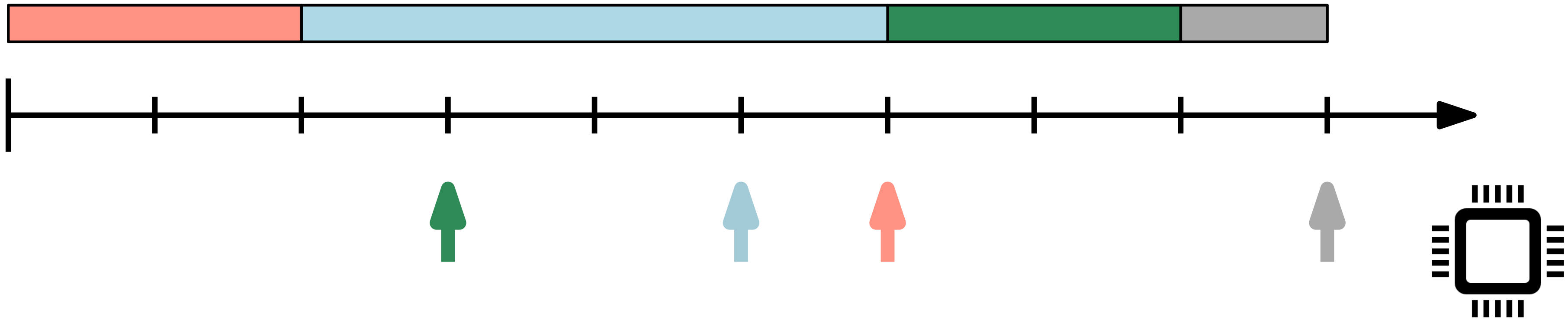
$$L(S) = \max_{i=1,\ldots,n} \max\{0, f_i - d(i)\}.$$

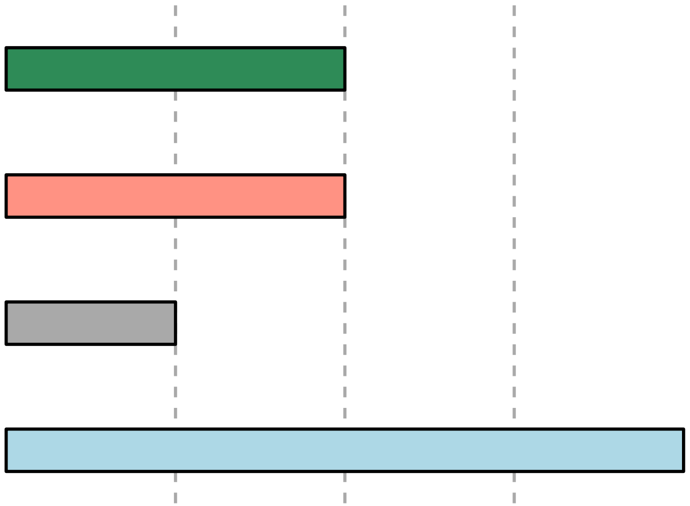
Example





Example





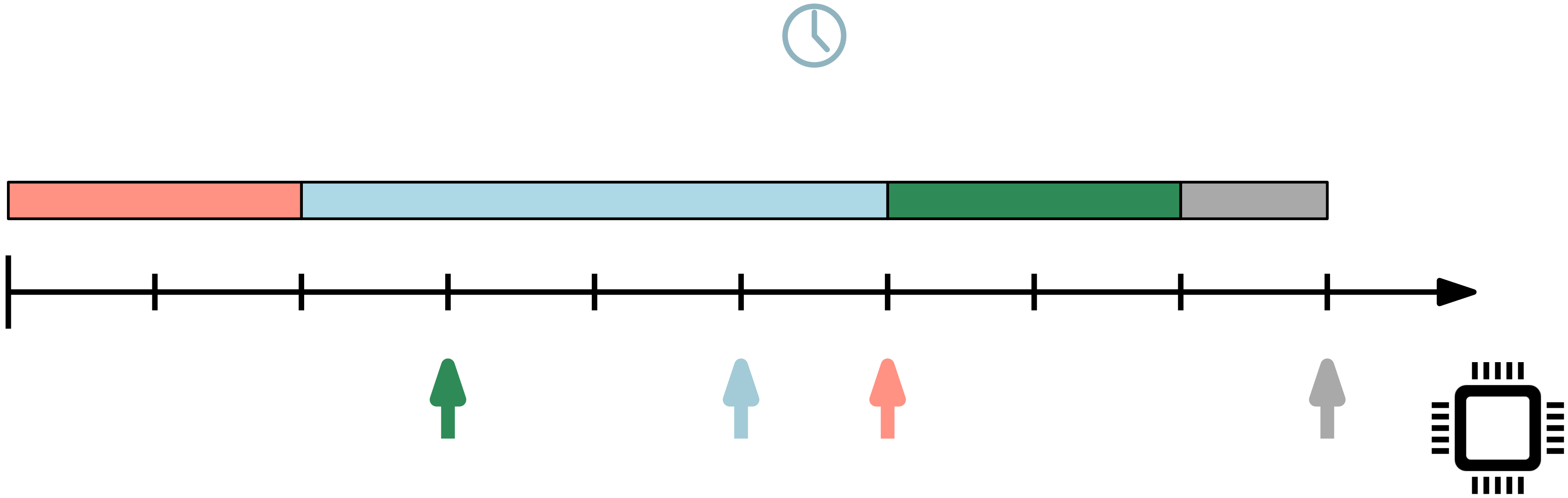
Job 1: $d(1) = 3$

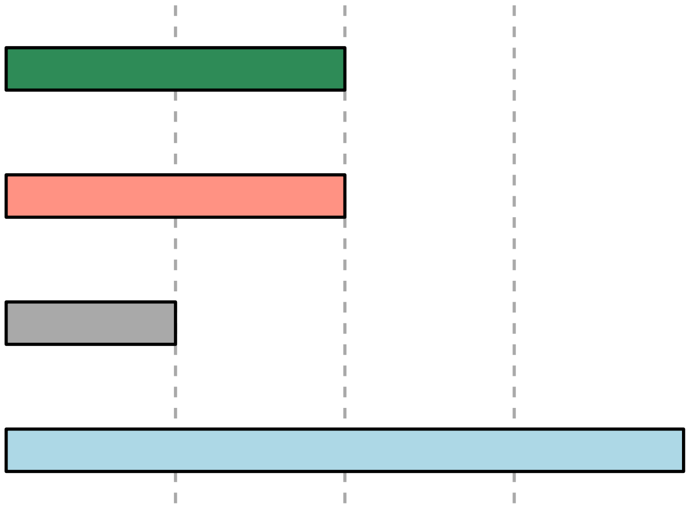
Job 2: $d(2) = 6$

Job 3: $d(3) = 9$

Job 4: $d(4) = 5$

Example





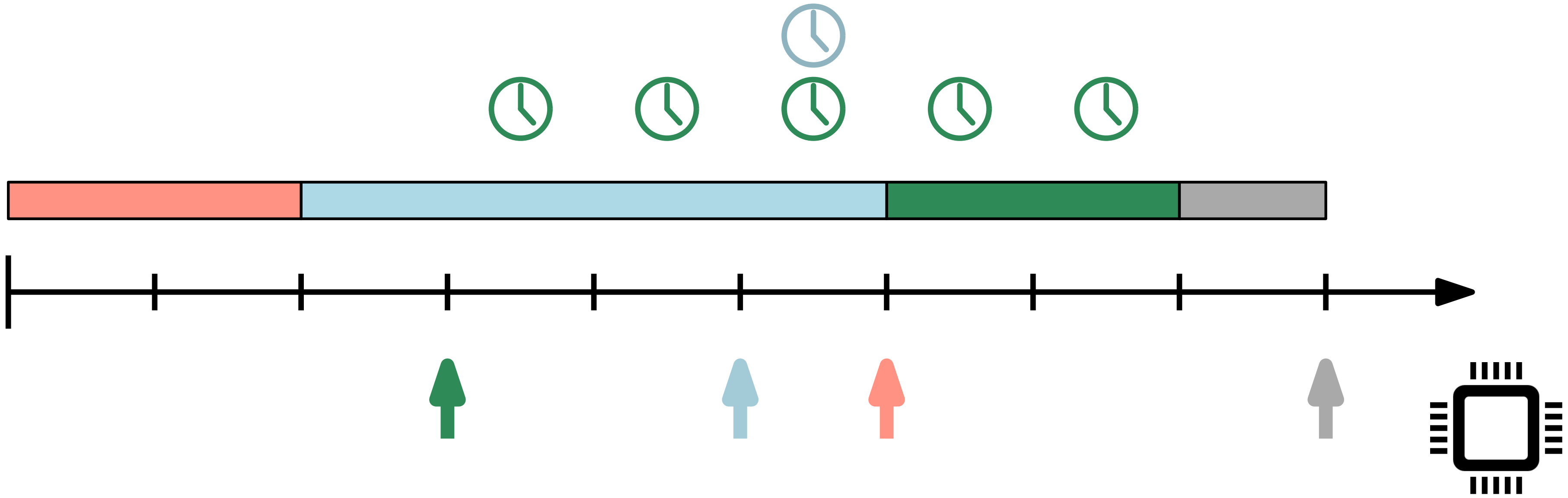
Job 1: $d(1) = 3$

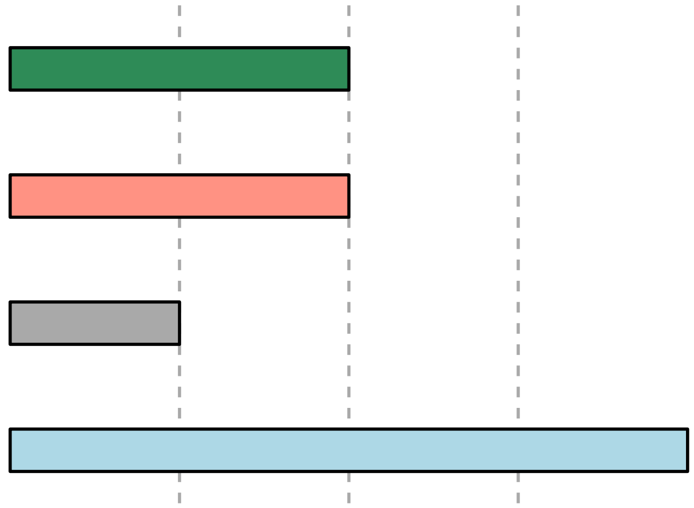
Job 2: $d(2) = 6$

Job 3: $d(3) = 9$

Job 4: $d(4) = 5$

Example





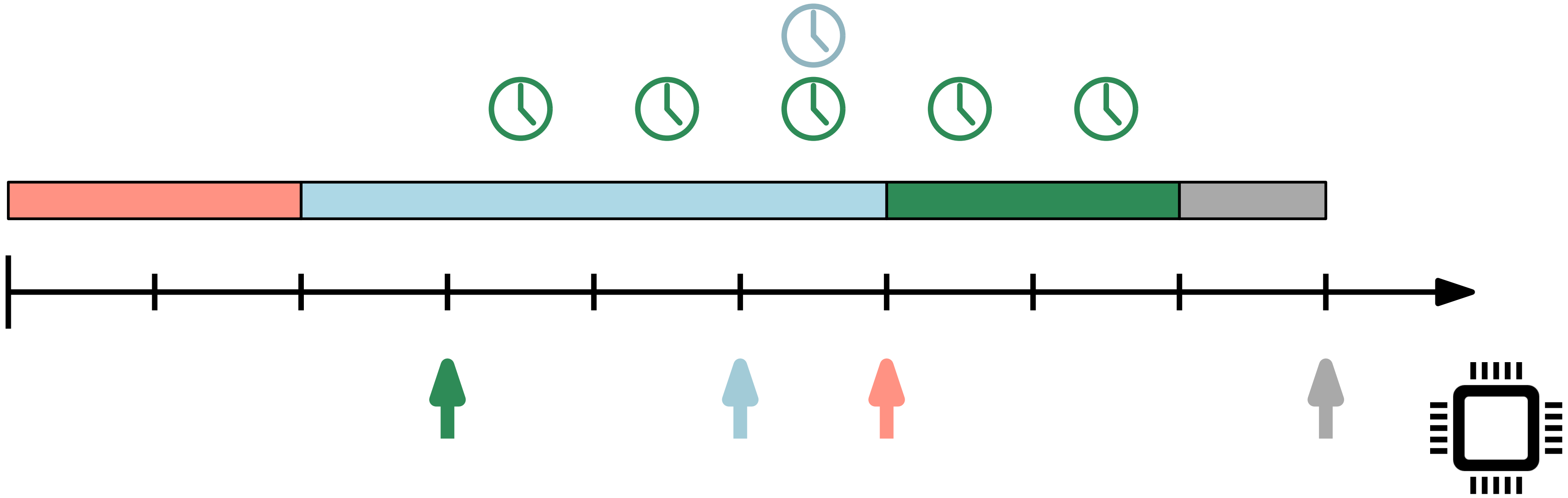
Job 1: $d(1) = 3$

Job 2: $d(2) = 6$

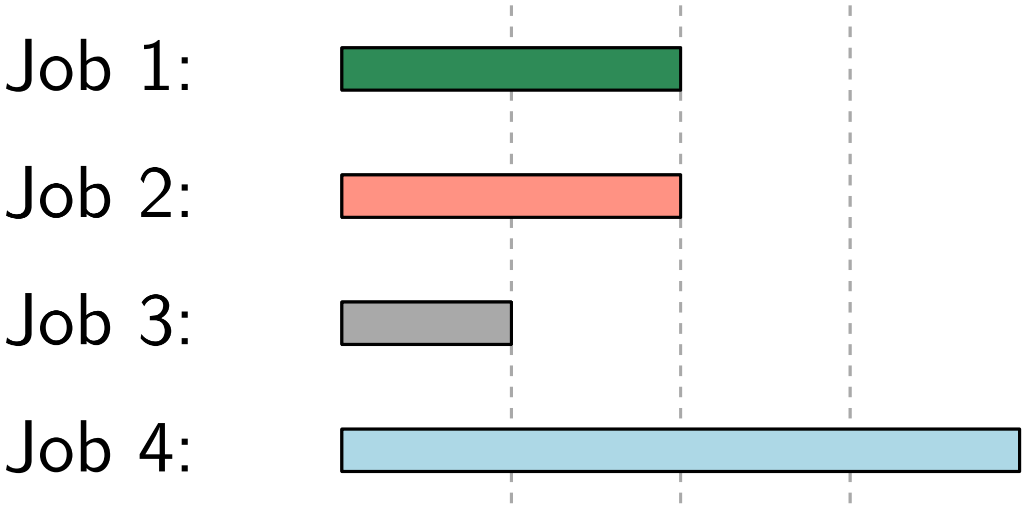
Job 3: $d(3) = 9$

Job 4: $d(4) = 5$

Example



Maximum Lateness: 5



$d(1) = 3$

$d(2) = 6$

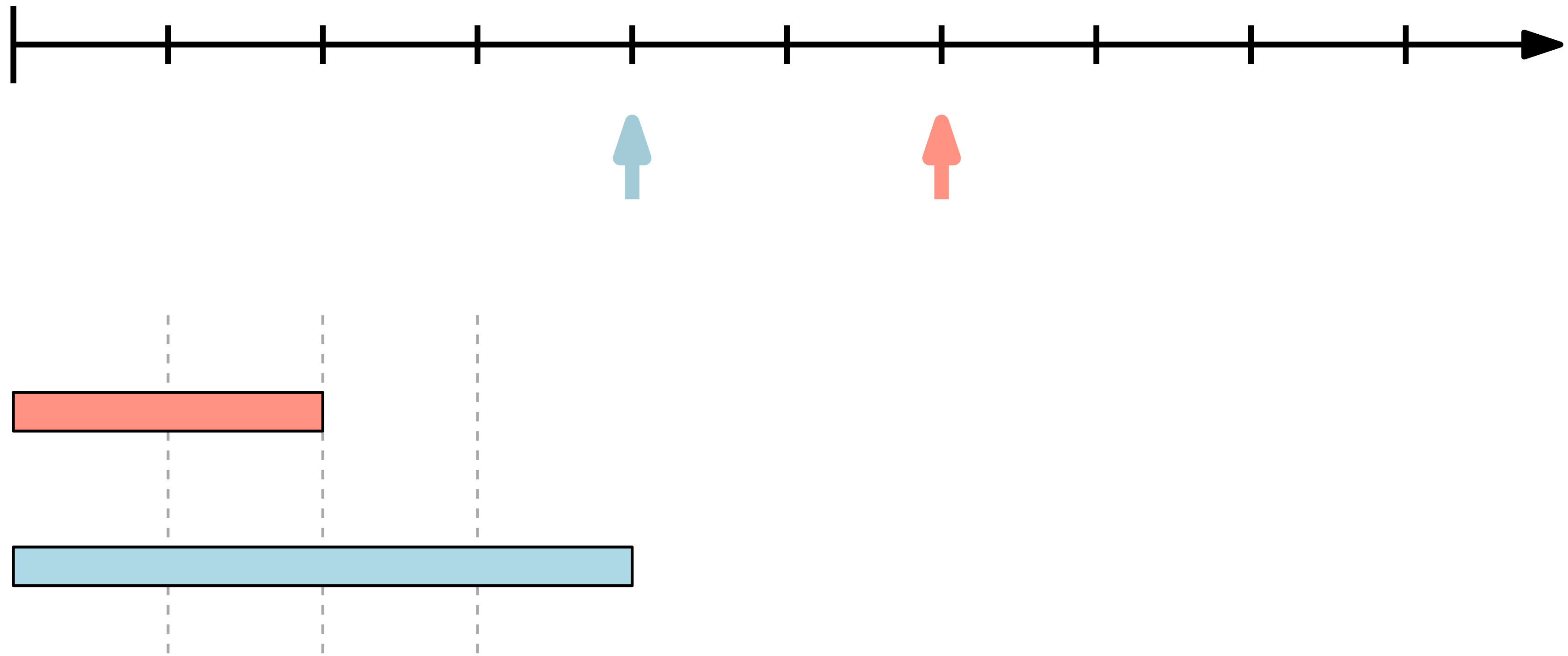
$d(3) = 9$

$d(4) = 5$

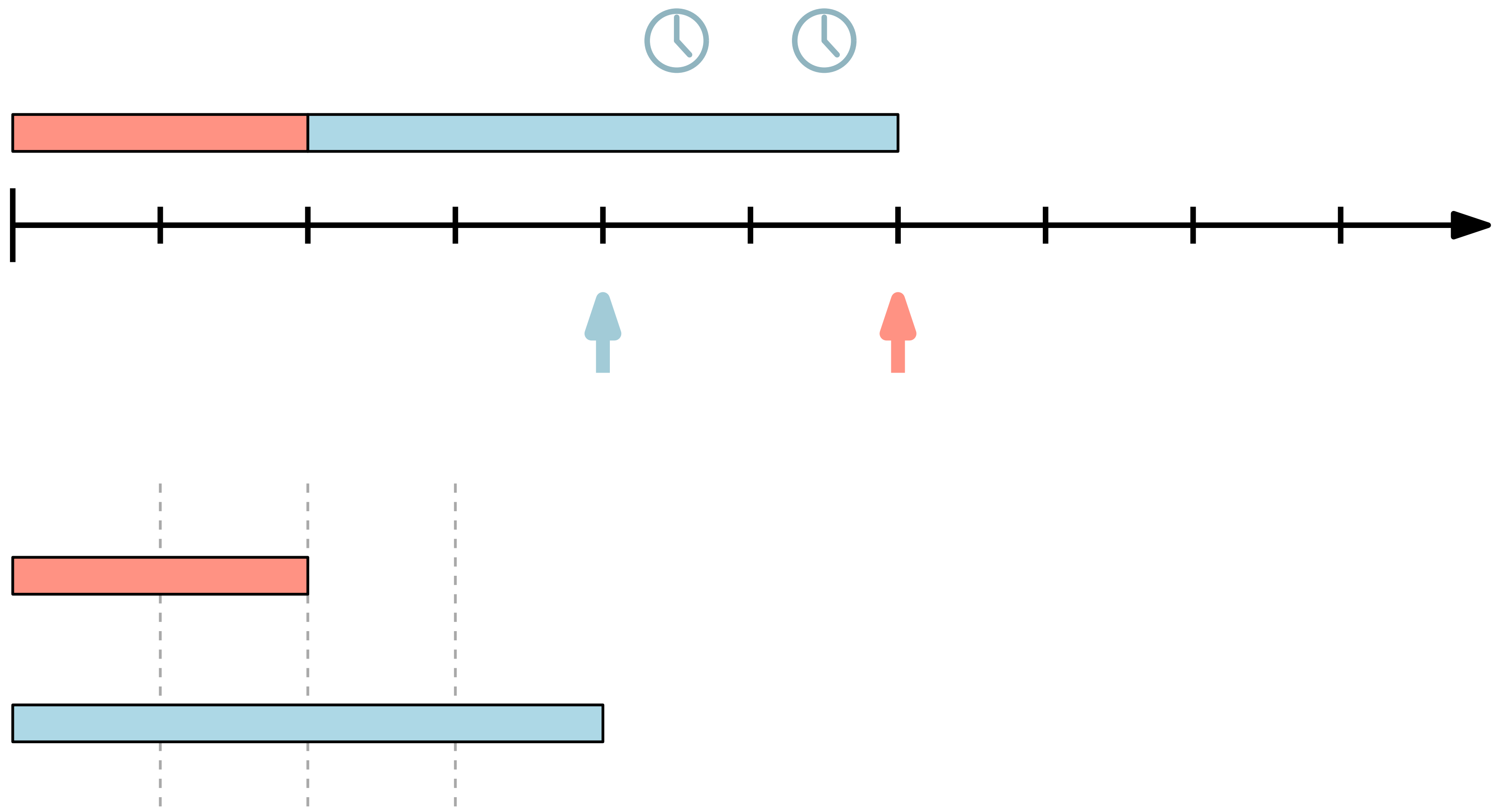
Which order for the jobs?

- **Shortest Job First**: Increasing order of $t(i)$.
- **Shortest Slack Time First:** Increasing order of $d(i) - t(i)$.
- **Earliest Deadline First:** Increasing order of $d(i)$.

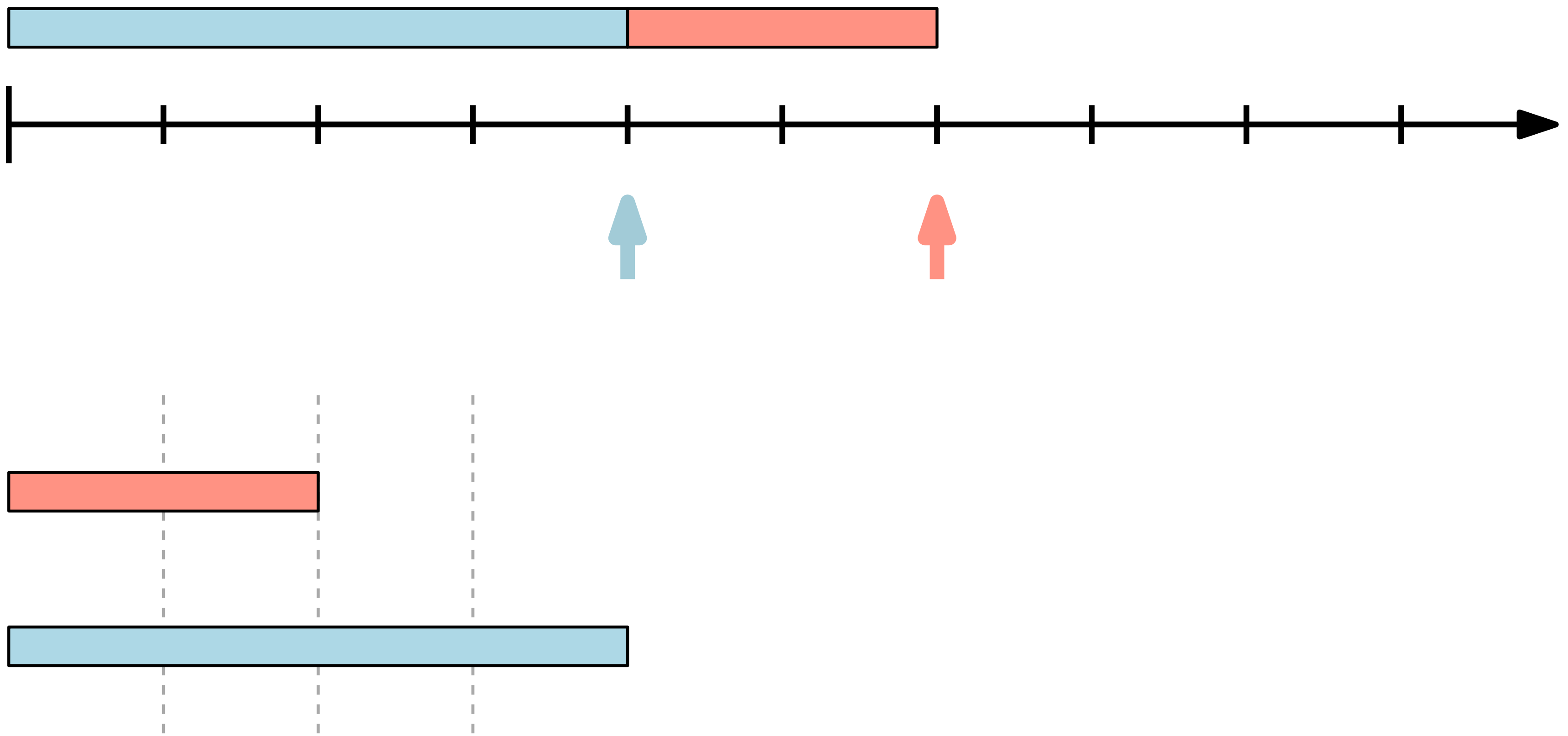
Shortest Job First



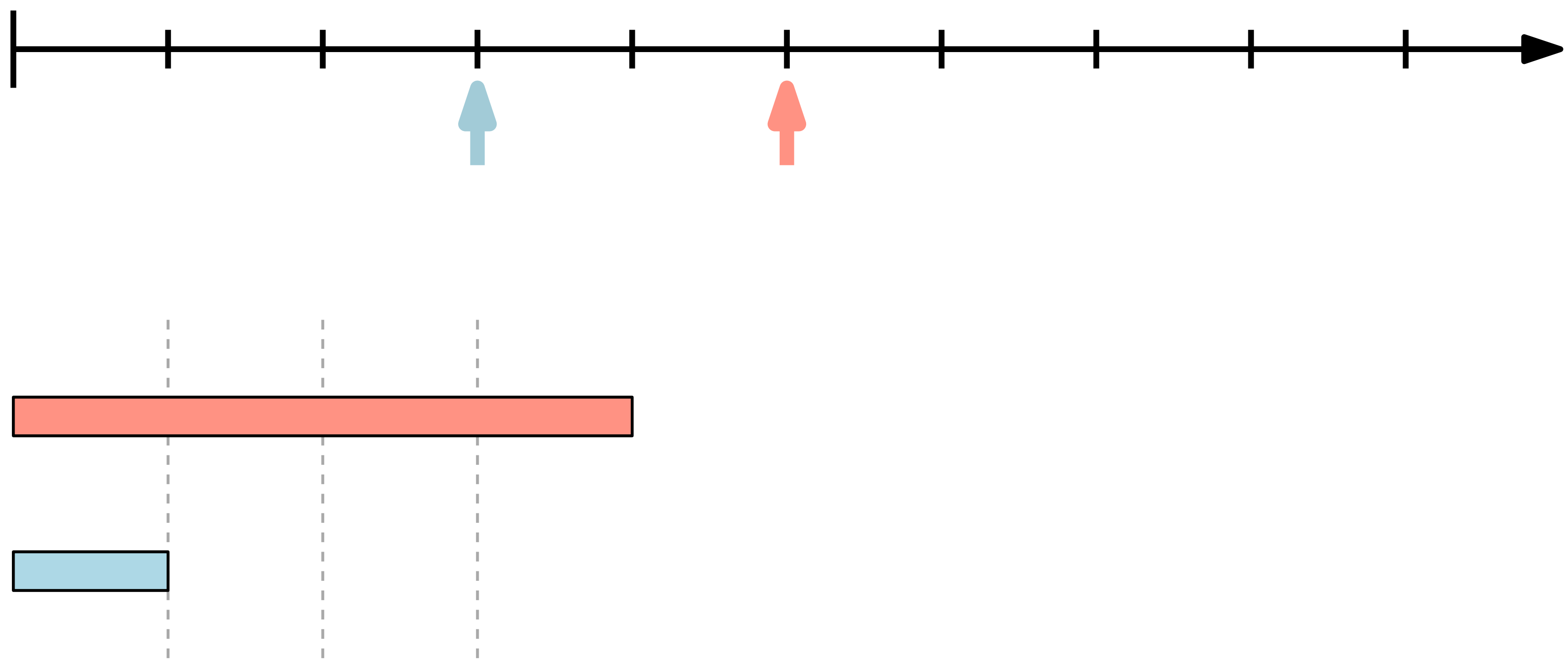
Shortest Job First



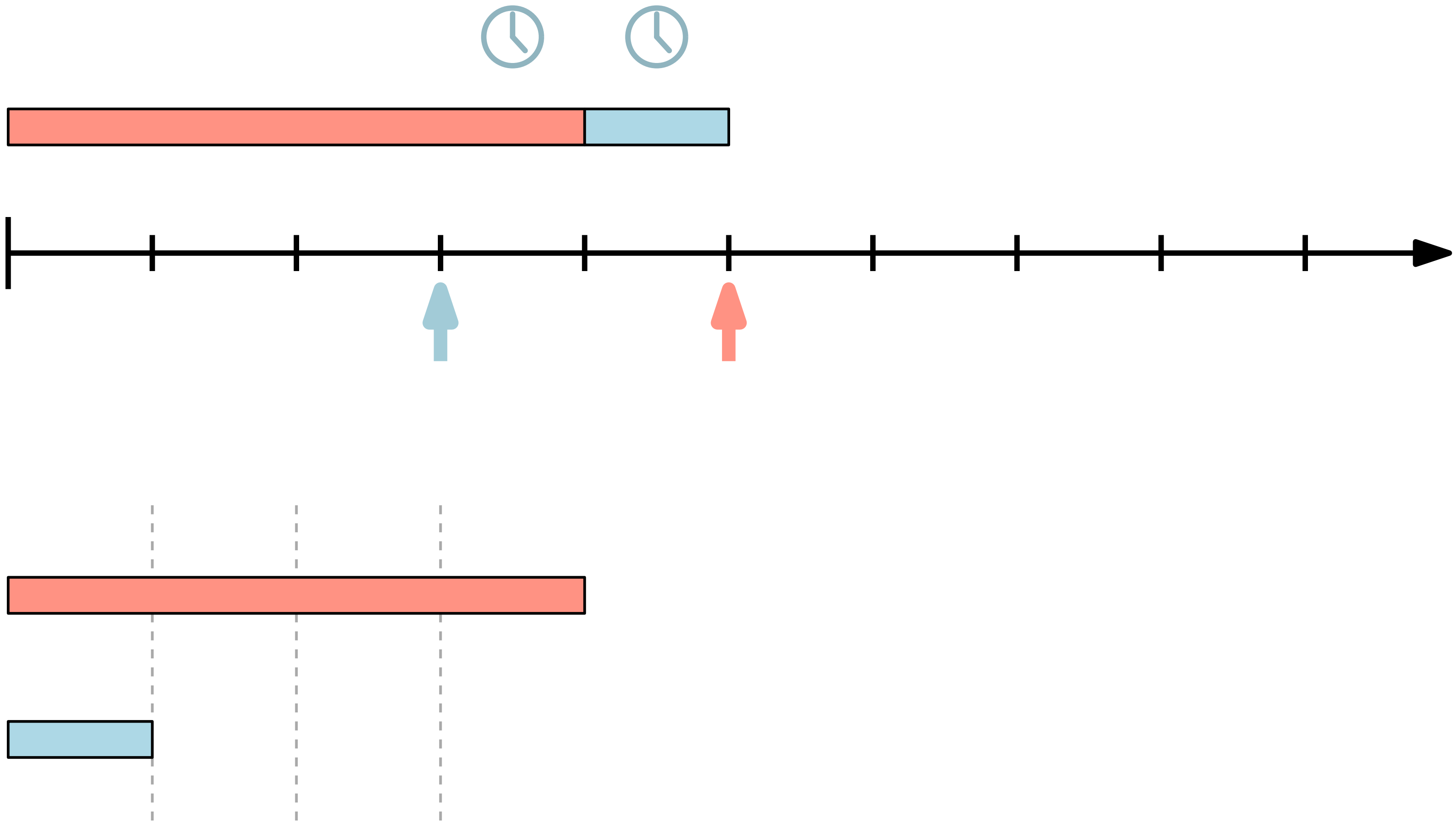
Shortest Job First



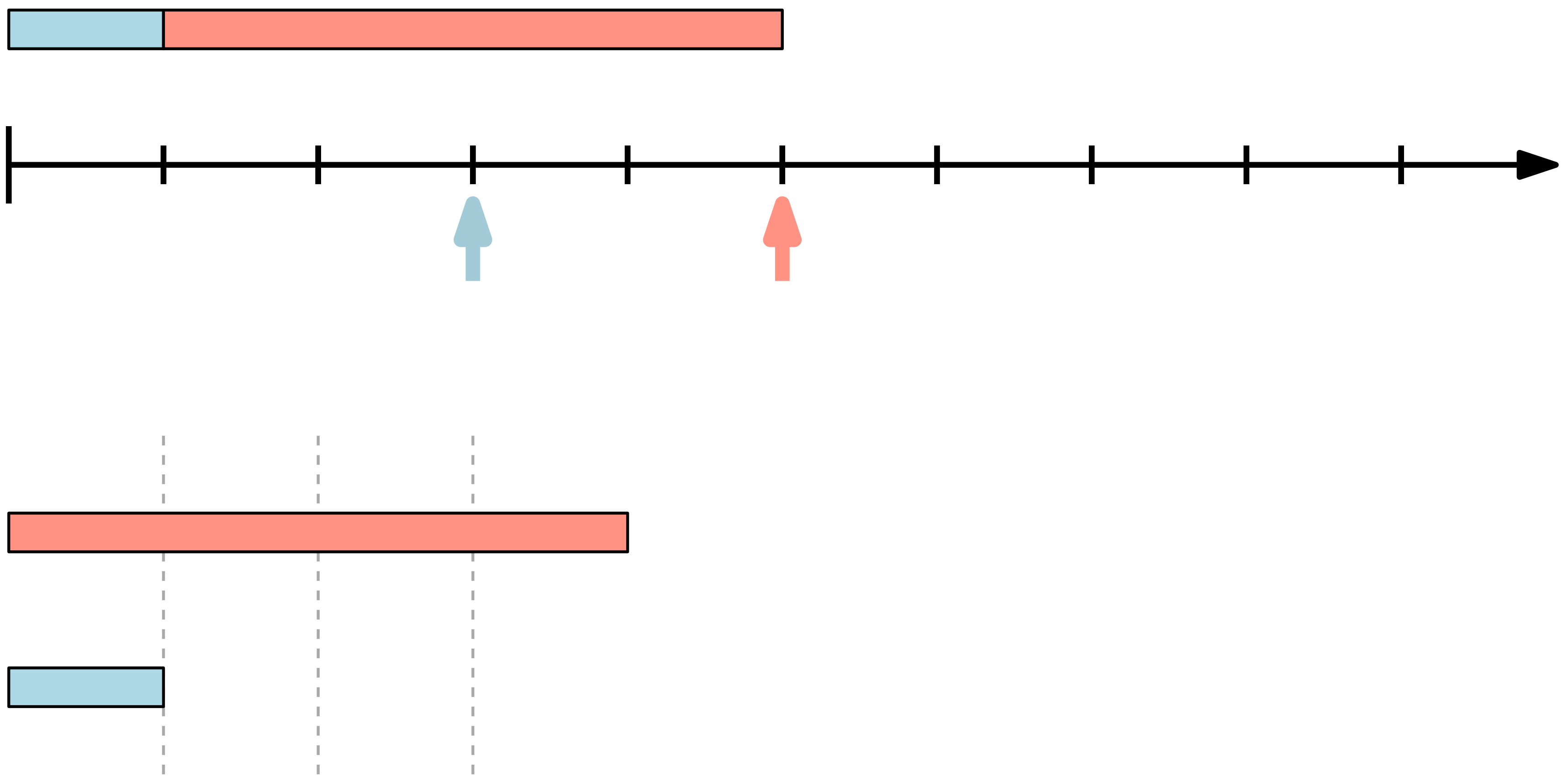
Shortest Slack Time First



Shortest Slack Time First



Shortest Slack Time First



Which order for the jobs?

- ~~**Shortest Job First:** Increasing order of $t(i)$.~~
- ~~**Shortest Slack Time First:** Increasing order of $d(i) - t(i)$.~~
- **Earliest Deadline First:** Increasing order of $d(i)$.

Earliest Deadline First

The algorithm:

- $\langle j_1, \ldots, j_n \rangle \leftarrow$ sort jobs w.r.t. $d(\cdot)$.
- For $i = 1 \ldots, n$
  - Schedule j_i at time $\sum_{k=1}^{i-1} t(k)$

Earliest Deadline First

The algorithm:

- $\langle j_1, \ldots, j_n \rangle \leftarrow$ sort jobs w.r.t. $d(\cdot)$.
- For $i = 1 \ldots, n$
    - Schedule j_i at time $\sum_{k=1}^{i-1} t(k)$

Proof of correctness:

- **Observation:** The greedy schedule has no idle time.

Earliest Deadline First

The algorithm:

- $\langle j_1, \ldots, j_n \rangle \leftarrow$ sort jobs w.r.t. $d(\cdot)$.
- For $i = 1 \ldots, n$
  - Schedule j_i at time $\sum_{k=1}^{i-1} t(k)$

Proof of correctness:

- **Observation:** The greedy schedule has no idle time.

- **Definition:** An inversion of a schedule S is a pair of jobs (i, j) such that job i is scheduled before job j but $d(i) > d(j)$.

- **Observation:** The greedy schedule has no inversion.

EDF - Proof of Correctness

- **Observation:** The greedy schedule has no idle time and no inversions.
- **Claim:** All schedules with no idle time and no inversions are identical.

EDF - Proof of Correctness

- **Observation:** The greedy schedule has no idle time and no inversions.
- **Claim:** All schedules with no idle time and no inversions are identical.
- **It suffices to show:** There exists an optimal schedule with no idle time and no inversions.

EDF - Proof of Correctness

Claim: For every optimal schedule S^* there is an optimal schedule S with no idle time and the same number of inversions as S^*.

EDF - Proof of Correctness

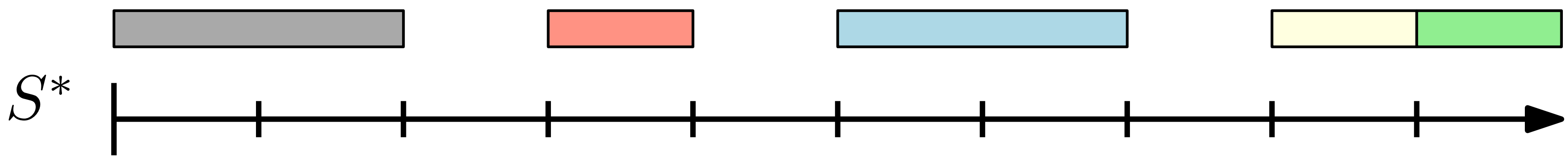
Claim: For every optimal schedule S^* there is an optimal schedule S with no idle time and the same number of inversions as S^*.

Proof: Let $j_1, \ldots, j_n$ be the sequence of jobs of S^*. Let f_k^* and ℓ_k^* be the finish time and lateness of job k according to S^*, respectively.

Consider the schedule S that excecutes $j_1, \ldots, j_n$ (in order) with no idle time.

Notice that $f_i = \sum_{k=1}^{i} t(j_k) \leq f_i^*$ and hence $\ell_i \leq \ell_i^*$.

S is feasible and has the same inversions as S^*. □



EDF - Proof of Correctness

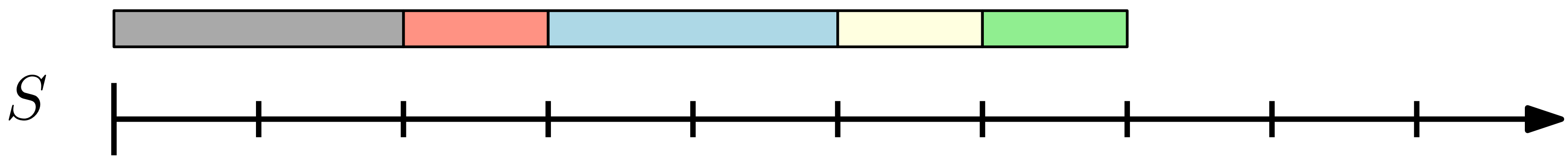
Claim: For every optimal schedule S^* there is an optimal schedule S with no idle time and the same number of inversions as S^*.

Proof: Let $j_1, \ldots, j_n$ be the sequence of jobs of S^*. Let f_k^* and ℓ_k^* be the finish time and lateness of job k according to S^*, respectively.

Consider the schedule S that excecutes $j_1, \ldots, j_n$ (in order) with no idle time.

Notice that $f_i = \sum_{k=1}^{i} t(j_k) \leq f_i^*$ and hence $\ell_i \leq \ell_i^*$.

S is feasible and has the same inversions as S^*. $\square$



EDF - Proof of Correctness

- **Observation:** The greedy schedule has no idle time and no inversions.
- **Claim:** All schedules with no idle time and no inversions are identical.
- **It suffices to show:** There exists an optimal schedule with no idle time and no inversions.

DONE

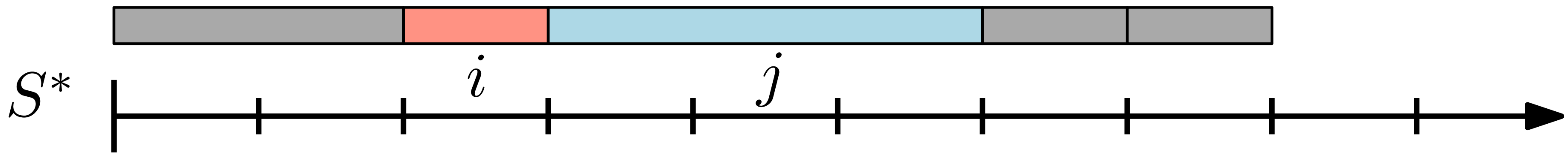
EDF - Proof of Correctness

Claim: Let S^* be an optimal schedule with no idle time and at least 1 inversion. There is an optimal schedule S with no idle time and less inversions than S^*.

EDF - Proof of Correctness

Claim: Let S^* be an optimal schedule with no idle time and at least 1 inversion. There is an optimal schedule S with no idle time and less inversions than S^*.

Proof (sketch): S^* must also contain an inversion (i, j) such that no job is scheduled between i and j.

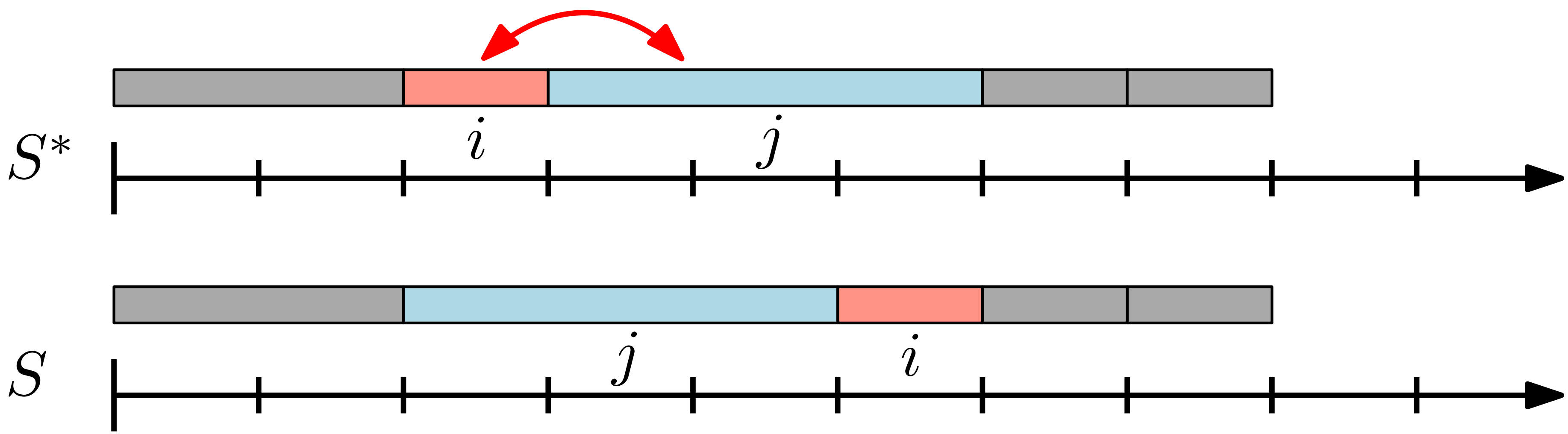


EDF - Proof of Correctness

Claim: Let S^* be an optimal schedule with no idle time and at least 1 inversion. There is an optimal schedule S with no idle time and less inversions than S^*.

Proof (sketch): S^* must also contain an inversion (i, j) such that no job is scheduled between i and j.

Consider the schedule S obtained by swapping job i with job j.



$f_j < f_j^* \leq d(j) + \ell_j^*$

$f_i = f_j^* \leq d(j) + \ell_j^* < d(i) + \ell_j^*$

□

EDF - Proof of Correctness

Claim: Let S^* be an optimal schedule with no idle time and at least 1 inversion. There is an optimal schedule S with no idle time and less inversions than S^*.

- Pick any optimal schedule S^*
- Initially S^* can have at most $\binom{n}{2}$ inversions.
- Iteratively apply the claim until no inversions are left.
- We have obtained an optimal schedule with no idle time and no inversions.

EDF - Proof of Correctness

Claim: Let S^* be an optimal schedule with no idle time and at least 1 inversion. There is an optimal schedule S with no idle time and less inversions than S^*.

- Pick any optimal schedule S^*
- Initially S^* can have at most $\binom{n}{2}$ inversions.
- Iteratively apply the claim until no inversions are left.
- We have obtained an optimal schedule with no idle time and no inversions.

This is exactly the greedy schedule!

EDF - Proof of Correctness

Claim: Let S^* be an optimal schedule with no idle time and at least 1 inversion. There is an optimal schedule S with no idle time and less inversions than S^*.

- Pick any optimal schedule S^*
- Initially S^* can have at most $\binom{n}{2}$ inversions.
- Iteratively apply the claim until no inversions are left.

Trick/Technique: Exchange Argument

Iteratively transform the optimal solution into the greedy solution without worsening its quality.

Recap

Trick/Technique: Greedy Stays Ahead

At each step, the solution produced by greedy is not worse than the one produced by any other algorithm.

Trick/Technique: Greedy Stays Ahead

At each step, the solution produced by greedy is not worse than the one produced by any other algorithm.

Trick/Technique: Finding Structural Properties

Find a structural property that implies optimality. (e.g., a lower bound to the measure of an optimal solution). Prove that greedy returns a solution with that property.

Trick/Technique: Greedy Stays Ahead

At each step, the solution produced by greedy is not worse than the one produced by any other algorithm.

Trick/Technique: Finding Structural Properties

Find a structural property that implies optimality. (e.g., a lower bound to the measure of an optimal solution). Prove that greedy returns a solution with that property.

Trick/Technique: Exchange Argument

Iteratively transform the optimal solution into the greedy solution without worsening its quality.