Greedy Algorithms

- The Greedy Algorithm Techniques
- Knapsack Problem
- Huffman Codes
- Scheduling

Scheduling Problems

There are many variations of the scheduling problem.

- Activity: Goal maximize the number of activities
- Machine/Task Scheduling: Goal minimize the number of machines needed to complete all Tasks with start/finish constraints.
- Job Scheduling: Goal minimize the total time it takes to complete all jobs on a set of machines.
- And more ….

An Activity Scheduling Problem

Input: A set of activities S = {a_1,…, a_n}

- Each activity has start time and a finish time: $a_i = (s_i, f_i)$
- Two activities are compatible if and only if their interval does not overlap

Output: a maximum-size subset of mutually compatible activities

The Activity Scheduling Problem

Here are a set of start and finish times

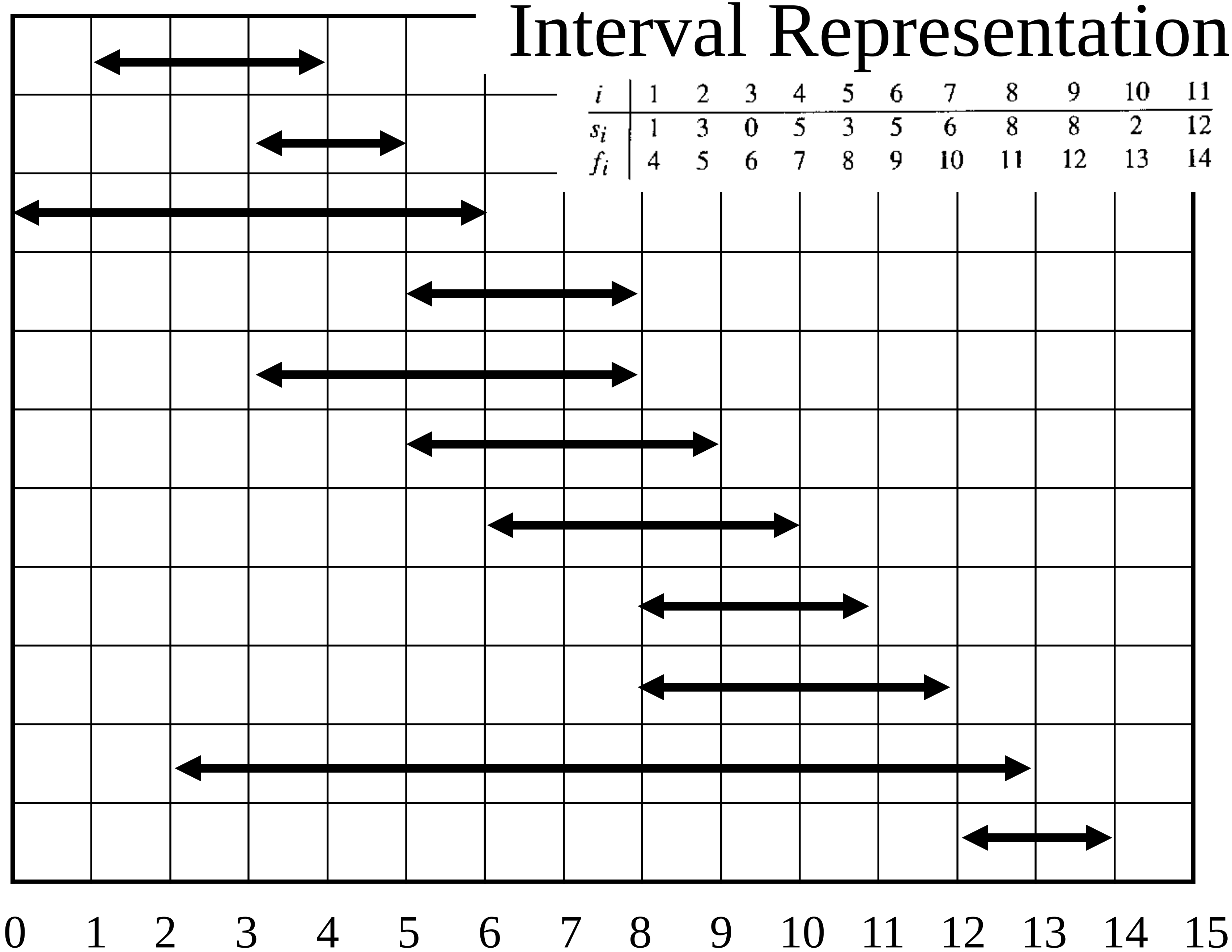
i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	9	10	11	12	13	14

What is the maximum number of activities that can be completed?

- $\{a_3, a_9, a_{11}\}$ can be completed
- But so can $\{a_1, a_4, a_8, a_{11}\}$ which is a larger set
- But it is not unique, consider $\{a_2, a_4, a_9, a_{11}\}$

Interval Representation

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	9	10	11	12	13	14



Activity Scheduling: Greedy Algorithms

Greedy. Consider activities in some natural order.
Take each activity provided it's compatible with the ones already taken.

- [Earliest start time] Consider activities in ascending order of s_j.
- [Earliest finish time] Consider activities in ascending order of f_j.
- [Shortest interval] Consider activities in ascending order of $f_j - s_j$.
- [Fewest conflicts] For each activity j, count the number of conflicting activities c_j. Schedule in ascending order of c_j.

Greedy Algorithms are not always Optimal

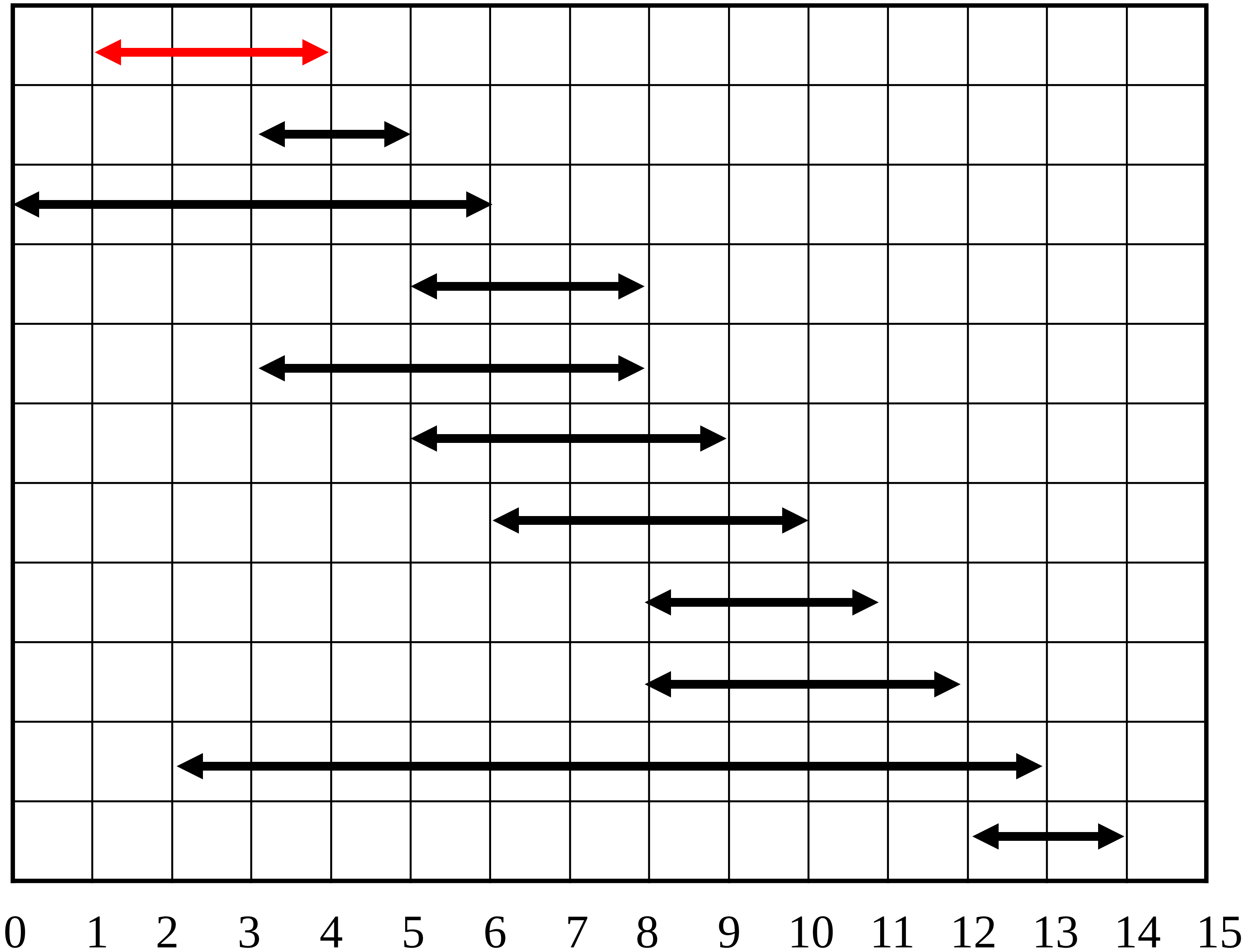
Counterexample for earliest start time

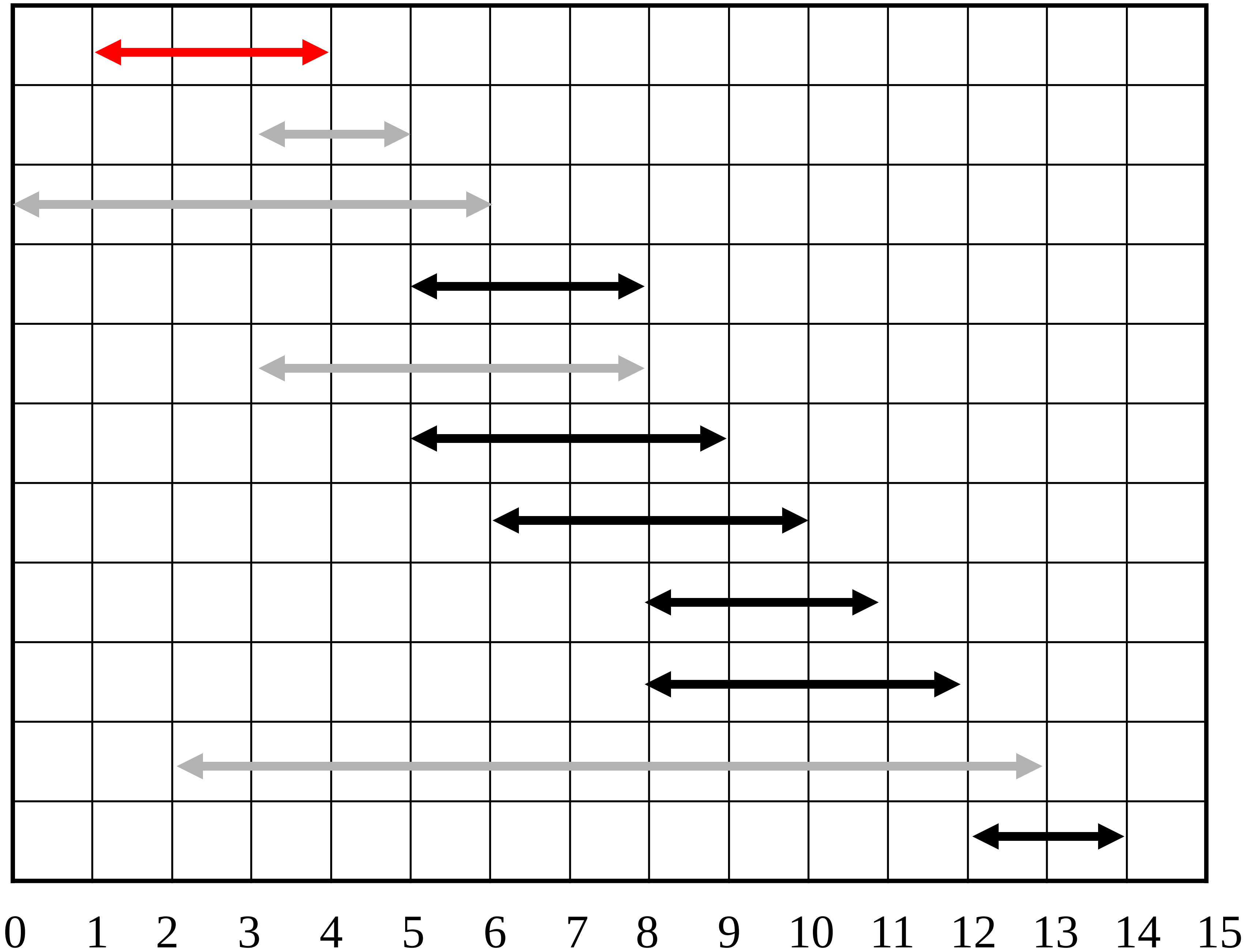
Counterexample for shortest interval

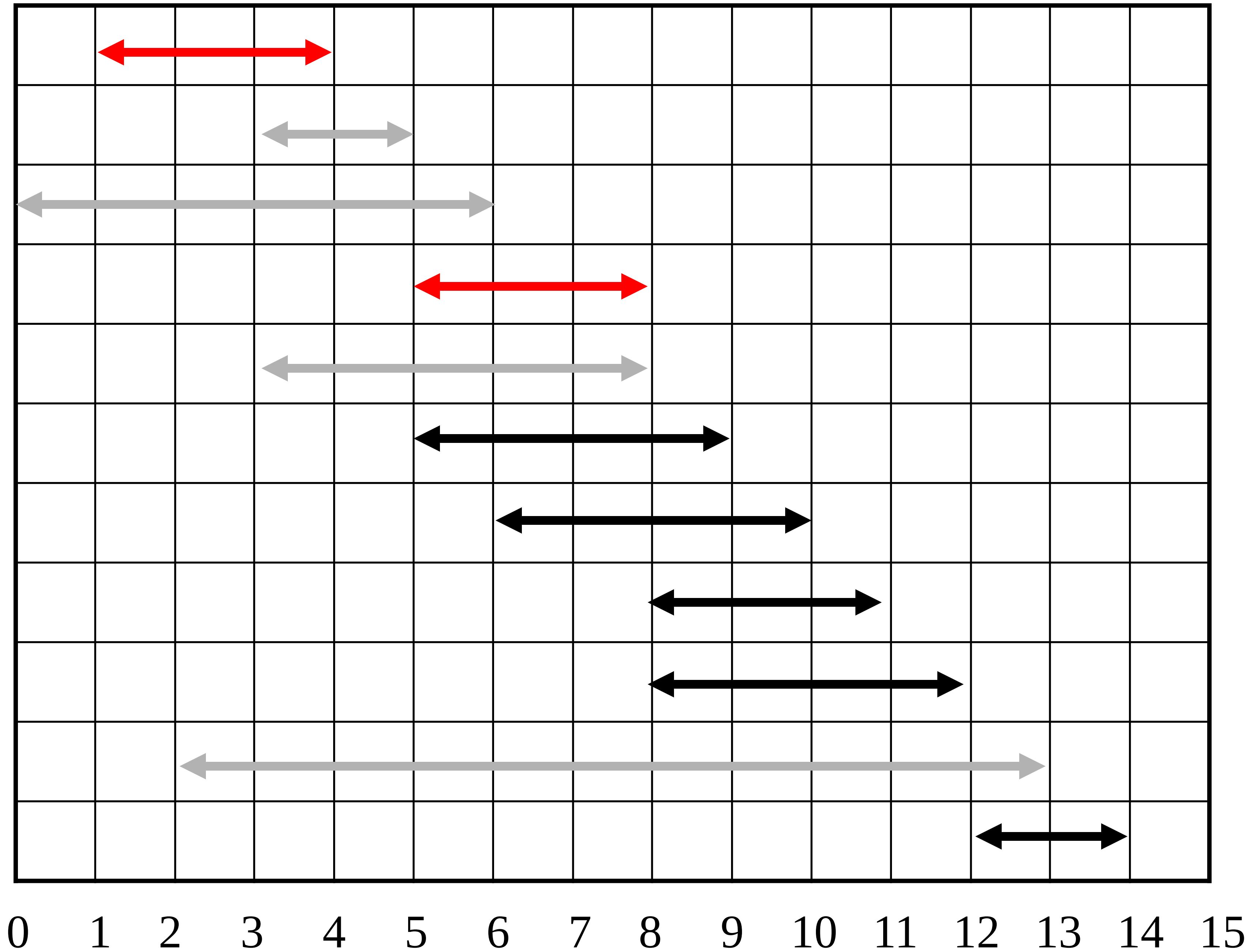
Counterexample for fewest conflicts

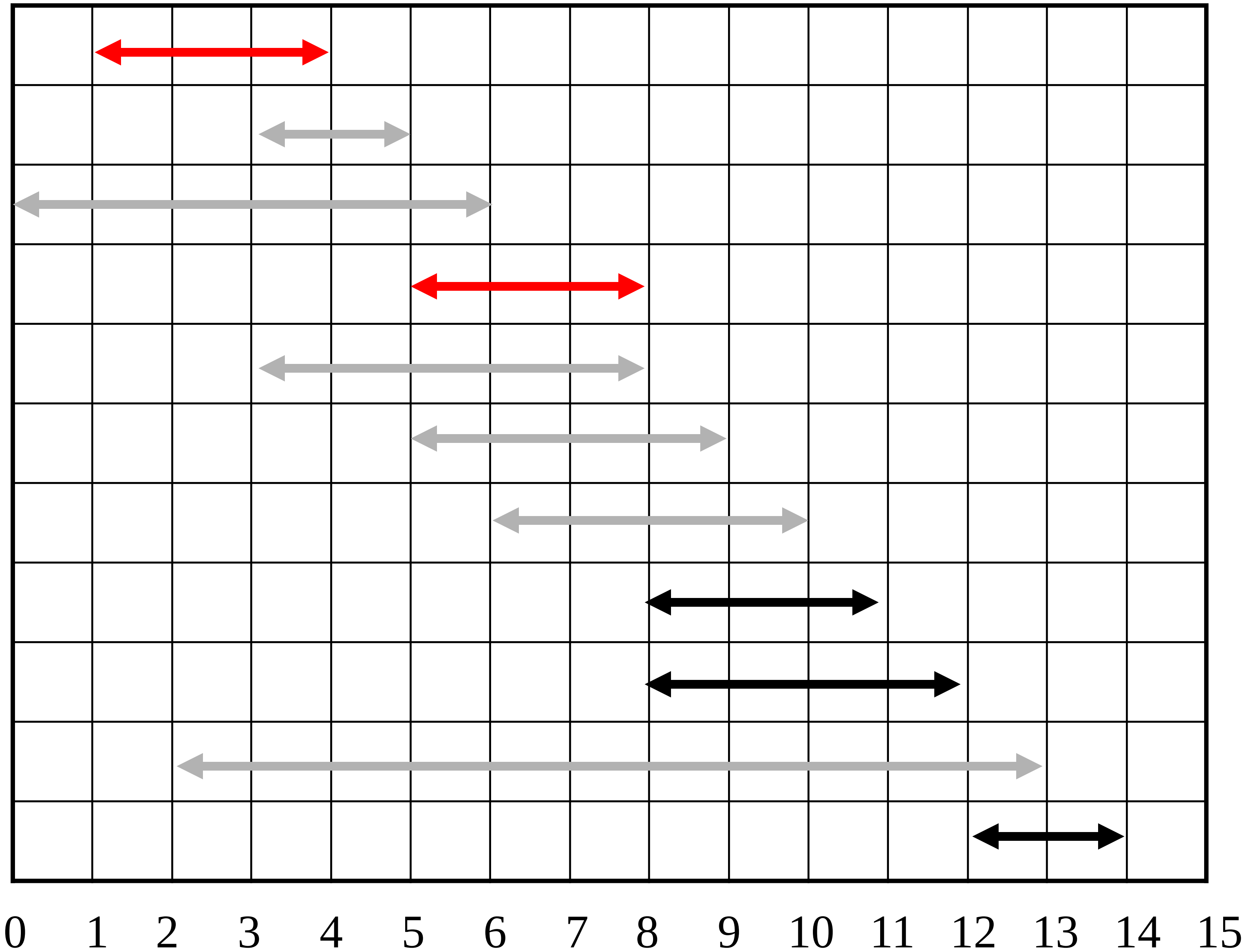
Earliest Finish Greedy Strategy

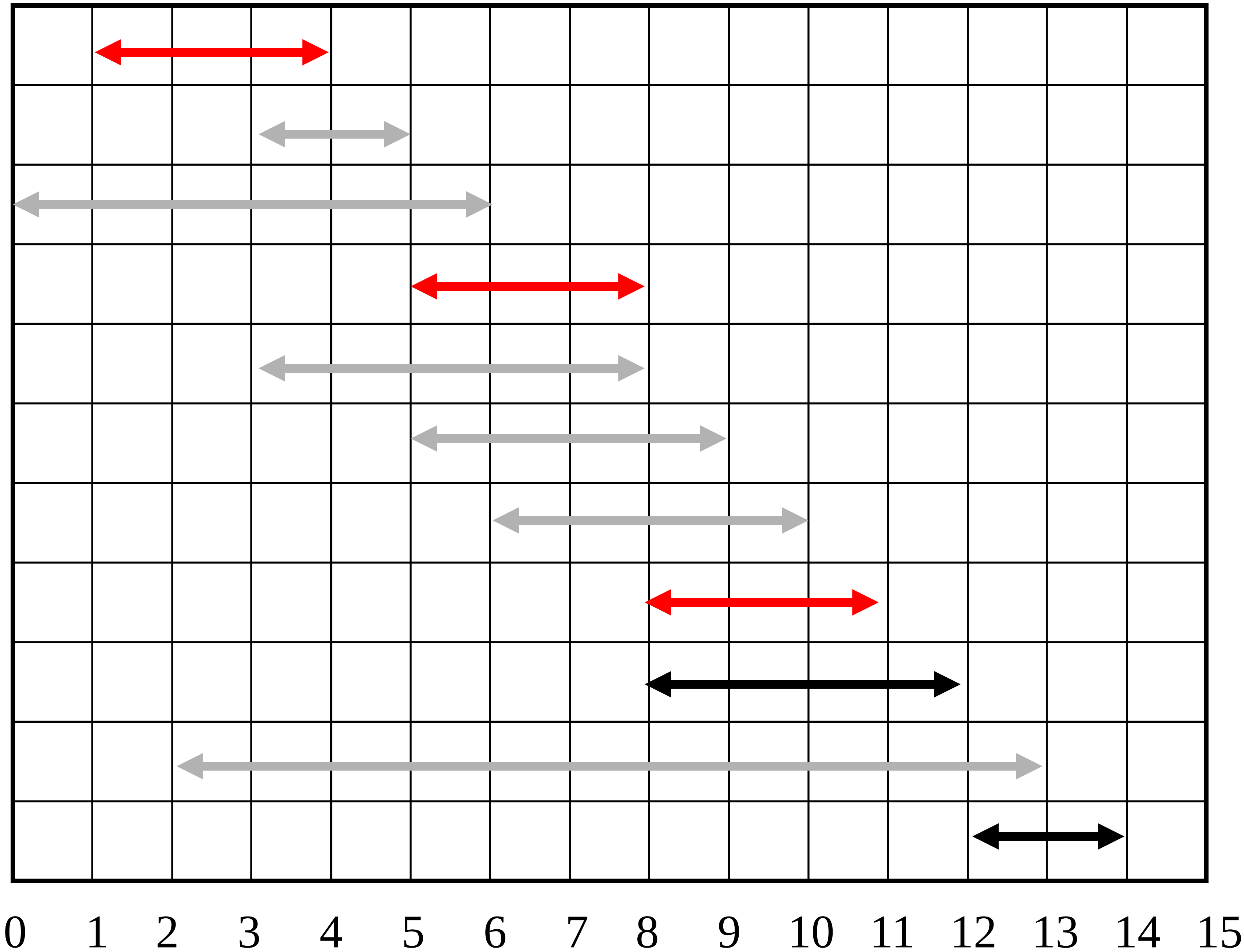
- Select the activity with the earliest finish
- Eliminate the activities that could not be scheduled
- Repeat!
- Greedy in the sense that it leaves as much opportunity as possible for the remaining activities to be scheduled
- The greedy choice is the one that maximizes the amount of unscheduled time remaining

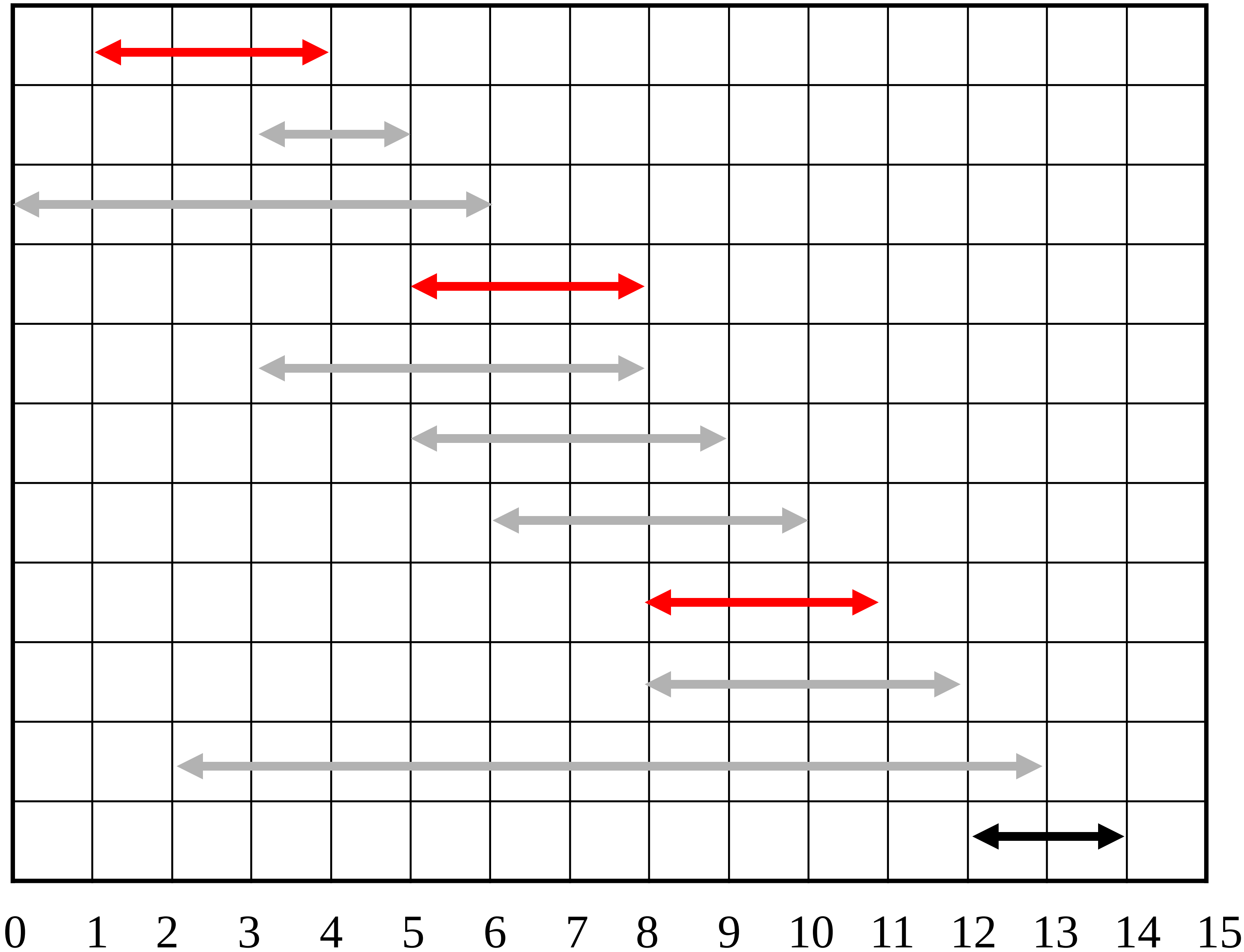

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

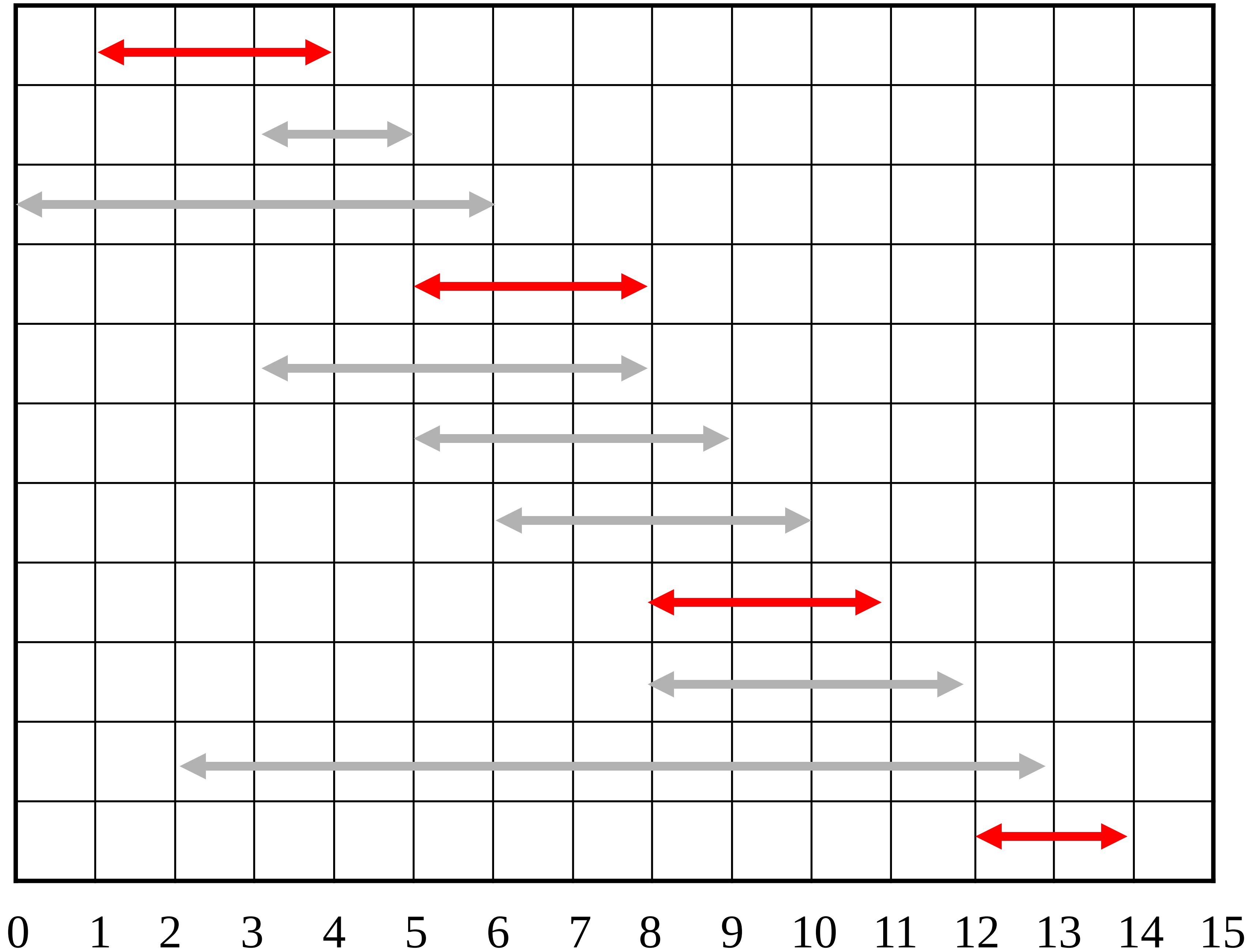

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Assuming activities are sorted by finish time

```
GREEDY-ACTIVITY-SELECTOR(s, f)
n ← length[s]
A ← {a_1}
i ← 1
for m ← 2 to n
        do if s_m ≥ f_i
                then A ← A ∪ {a_m}
                        i ← m
return A
```

Why this Algorithm is Optimal?

We will show that this algorithm uses the following properties

- The problem has the optimal substructure property
- The algorithm satisfies the greedy-choice property

Thus, it is Optimal

Greedy-Choice Property

- Show there is an optimal solution that begins with a greedy choice (with activity 1, which as the earliest finish time)
- Suppose $A \subseteq S$ in an optimal solution
  - Order the activities in A by finish time. The first activity in A is k
    - If k = 1, the schedule A begins with a greedy choice
    - If $k \neq 1$, show that there is an optimal solution B to S that begins with the greedy choice, activity 1
  - Let $B = A - \{k\} \cup \{1\}$
    - $f_1 \le f_k \rightarrow$ activities in B are disjoint (compatible)
    - B has the same number of activities as A
    - Thus, B is optimal

Greedy-Choice Property

- A globally optimal solution can be arrived at by making a locally optimal (greedy) choice
  - Make whatever choice seems best at the moment and then solve the sub-problem arising after the choice is made
  - The choice made by a greedy algorithm may depend on choices so far, but it cannot depend on any future choices or on the solutions to sub-problems
- Of course, we must prove that a greedy choice at each step yields a globally optimal solution

Elements of Greedy Strategy

- An greedy algorithm makes a sequence of choices, each of the choices that seems best at the moment is chosen
  - NOT always produce an optimal solution
- Two ingredients that are exhibited by most problems that lend themselves to a greedy strategy
  - Greedy-choice property
  - Optimal substructure

Optimal Substructures

A problem exhibits optimal substructure if an optimal solution to the problem contains within it optimal solutions to sub-problems

Optimal Substructures

Once the greedy choice of activity 1 is made, the problem reduces to finding an optimal solution for the activity-selection problem over those activities in S that are compatible with activity 1

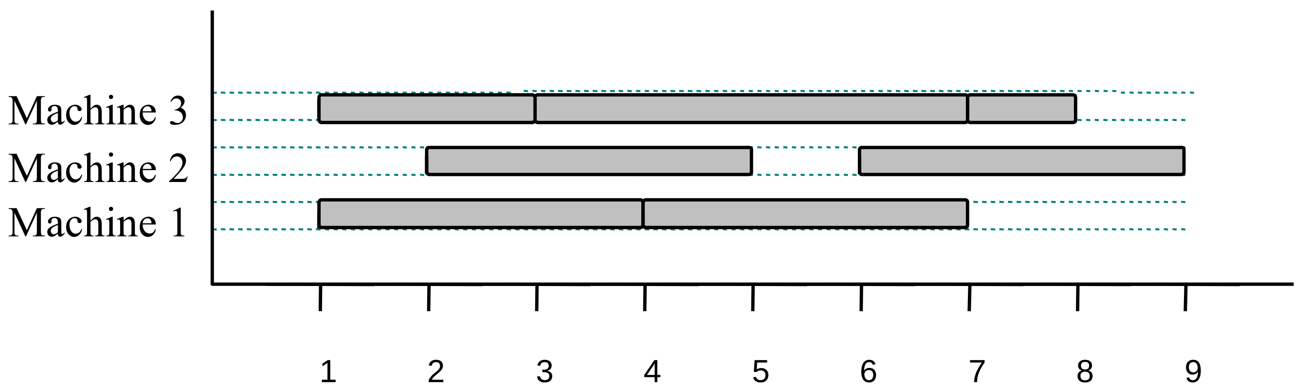
- If A is optimal to S, then $A' = A - \{1\}$ is optimal to $S' = \{i \in S: s_i \geq f_1\}$
- If we could find a solution B' to S' with more activities than A', adding activity 1 to B' would yield a solution B to S with more activities than A ➔ contradicting the optimality of A

After each greedy choice is made, we are left with an optimization problem of the same form as the original problem

- By induction on the number of choices made, making the greedy choice at every step produces an optimal solution

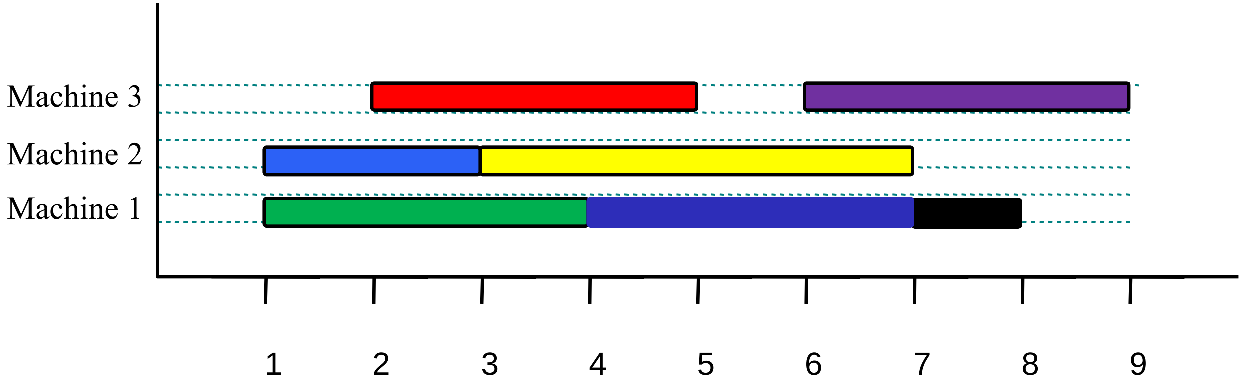
Machine Scheduling with start times

- Given: a set T of n tasks, each having:
  - A start time, s_i
  - A finish time, f_i (where $s_i < f_i$)
- Goal: Perform all the tasks using a minimum number of "machines."



Example

- Given: a set T of n=7 tasks, each having:
  - A start time, s_i
  - A finish time, f_i (where $s_i < f_i$)
  - [1,4], **[1,3]**, [2,5], [3,7], **[4,7]**, [6,9], [7,8] (ordered by start)
- Goal: Perform all tasks on min. number of machines



Machine Scheduling Algorithm

- **Greedy choice**: consider tasks by their start time and use as few machines as possible with this order.
  - Run time: $\Theta(n \log n)$.
- **Correctness:** Suppose there is a better schedule.
  - We can use k-1 machines
  - The algorithm uses k
  - Let i be first task scheduled on machine k
  - Task i must conflict with k-1 other tasks
  - K mutually conflict tasks
  - But that means there is no non-conflicting schedule using k-1 machines

Algorithm TaskSchedule(*T*)

Input: set ***T*** of tasks w/ start time s_i and finish time f_i

Output: non-conflicting schedule with minimum number of machines

$m \leftarrow 0$ {no. of machines}

while ***T is not empty***

remove task i w/ smallest s_i

if ***there's a machine j for*** **i then**

schedule i on machine j

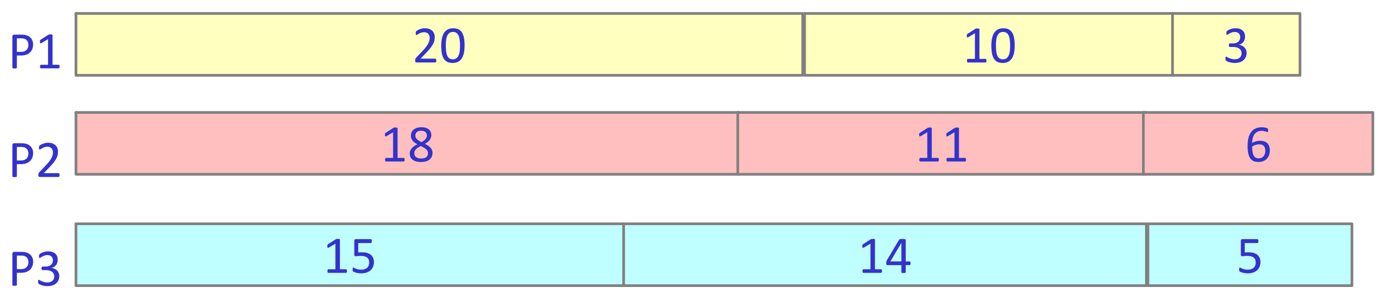
else

$m \leftarrow m + 1$

schedule i on machine m

Job Scheduling Problem

- There is no specified start times only durations.
- You have to run nine jobs, with running times of 3, 5, 6, 10, 11, 14, 15, 18, and 20 minutes
- You have three processors on which you can run these jobs
- You decide to do the longest-running jobs first, on whatever processor is available

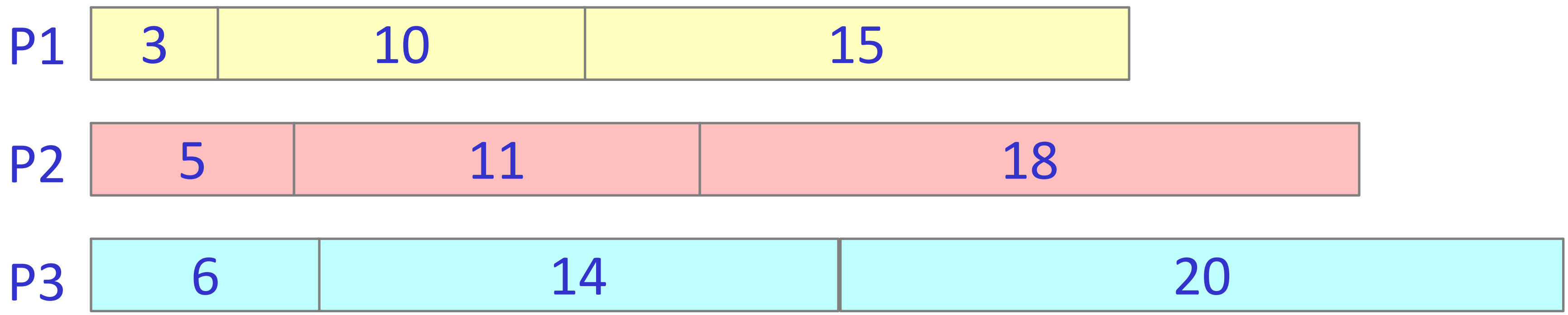


Time to completion: $18 + 11 + 6 = 35$ minutes

This solution isn't bad, but we might be able to do better

Another approach

- What would be the result if you ran the *shortest* job first?
- Again, the running times are 3, 5, 6, 10, 11, 14, 15, 18, and 20 minutes

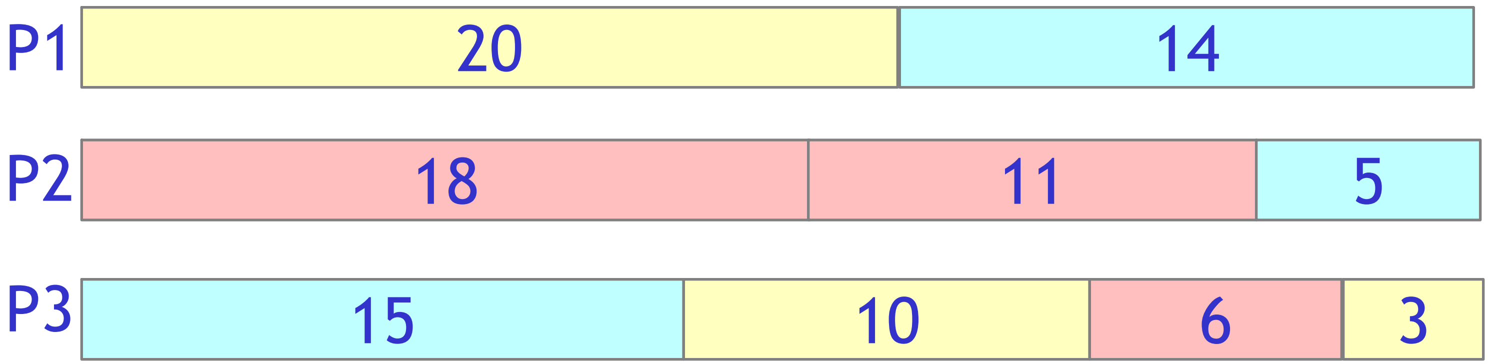


That wasn't such a good idea; time to completion is now
$6 + 14 + 20 = 40$ minutes

Note, however, that the greedy algorithm itself is fast

- All we had to do at each stage was pick the minimum or maximum

An optimum solution



- This solution is clearly optimal (why?)
- Clearly, there are other optimal solutions (why?)
- How do we find such a solution?
  - One way: Try all possible assignments of jobs to processors
  - Unfortunately, this approach can take exponential time