Methoden des Algorithmenentwurfs SS 2017 Kapitel 0: Einführung

Ulf-Peter Schroeder
https://sfb901.upb.de/sfb-901.html

Organisatorisches

Leitung:
- Dr. Ulf-Peter Schroeder (E-Mail: ups@upb.de)
- Sprechstunde: immer wenn ich in meinem Büro (F2.409) bin …

Modulinformation:
- Informatik Bachelor: Modul II.2.1 Modelle und Algorithmen (MuA)
- Winfo Bachelor: Modul Informatik im Kontext
- Mathematik Bachelor: Wahlpflichtveranstaltung
- Lehramt: Modul Algorithmen und Komplexität

Zeit, Ort, etc.:
- Mi 9-11 Uhr, F1.110, V2+Ü1/2, 4 ECTS Credits (Winfo 3 ECTS ?)

Webseite:
- https://cs.upb.de/ti/lehre/veranstaltungen/ss-2017/methoden-des-algorithmenentwurfs

Prüfungsanmeldung:
- via „Paul“ (ansonsten schriftlich in Ihrem Prüfungsamt)
- 1. Phase: 17.4.-17.5.2017 2. Phase: 4.9.-8.9.2017

Organisatorisches

Übungen:

- Übungsleitung: Robert Gmyr, M.Sc.
- Mi 11-13 Uhr (wöchentl.!), F1.110, ab 3. Mai
- Übungszettel:
  Ausgabe: wöchentlich auf der Webseite, ab 26. April
  Abgabe: keine!

Prüfung:

Mündliche Prüfung (ca. 30-45 Min.) am Ende des Semesters. 1. Prüfungszeitraum: 31.7. – 4.8.2017,

2. Prüfungszeitraum: 4.10. – 10.10.2017

Bei sinnvollem Vorrechnen einer Aufgabe in der Übung verbessert sich die Note um 0,33 Punkte. Maximal kann eine Person dieses zweimal Ausnutzen.

Inhaltsverzeichnis

Kapitel 1: Greedy Algorithms

Was ist die Greedy Methode?

- Algorithmus: Berechnet in endlicher Zeit zu einem Entscheidungs- oder Optimierungs-problem aus einer Eingabe eine Ausgabe respektive eine Lösung.
- Greedy-Algorithmus: Aufbau der Lösung in „kleinen“ Schritten. In jedem Schritt wird entsprechend eines Optimierungskriteriums eine „lokal optimale“ aber „irreversible“ Entscheidung getroffen.

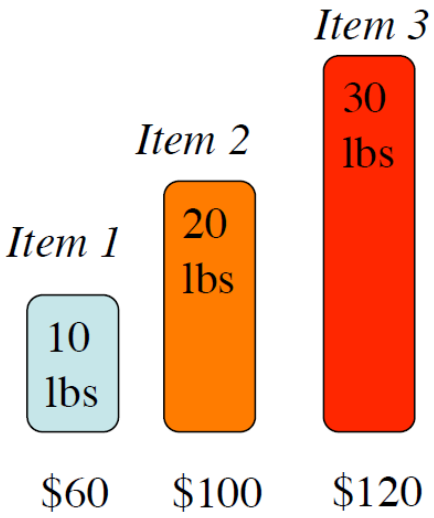
Greedy ist nicht immer optimal...

Problem 1.2.8 (0-1-KNAPSACK):

Input: *n items $S = \{1, \ldots, n\}$ of weight w_i and value v_i, $1 \leq i \leq n$. A knapsack able to carry weight W.*

Output: *A subset $I \subset S$, (or, equivalently, a vector $x \in \{0,1\}^n$ with $x_i = 1 \iff i \in I$) such that $\sum_{i \in I} w_i \leq W$ and $\sum_{i \in I} v_i \to \max$.*

This is called the 0-1 knapsack problem because each item must be taken in its entirety. If we use a greedy strategy to solve this problem, we would take the objects in order of their benefit/ weight value.

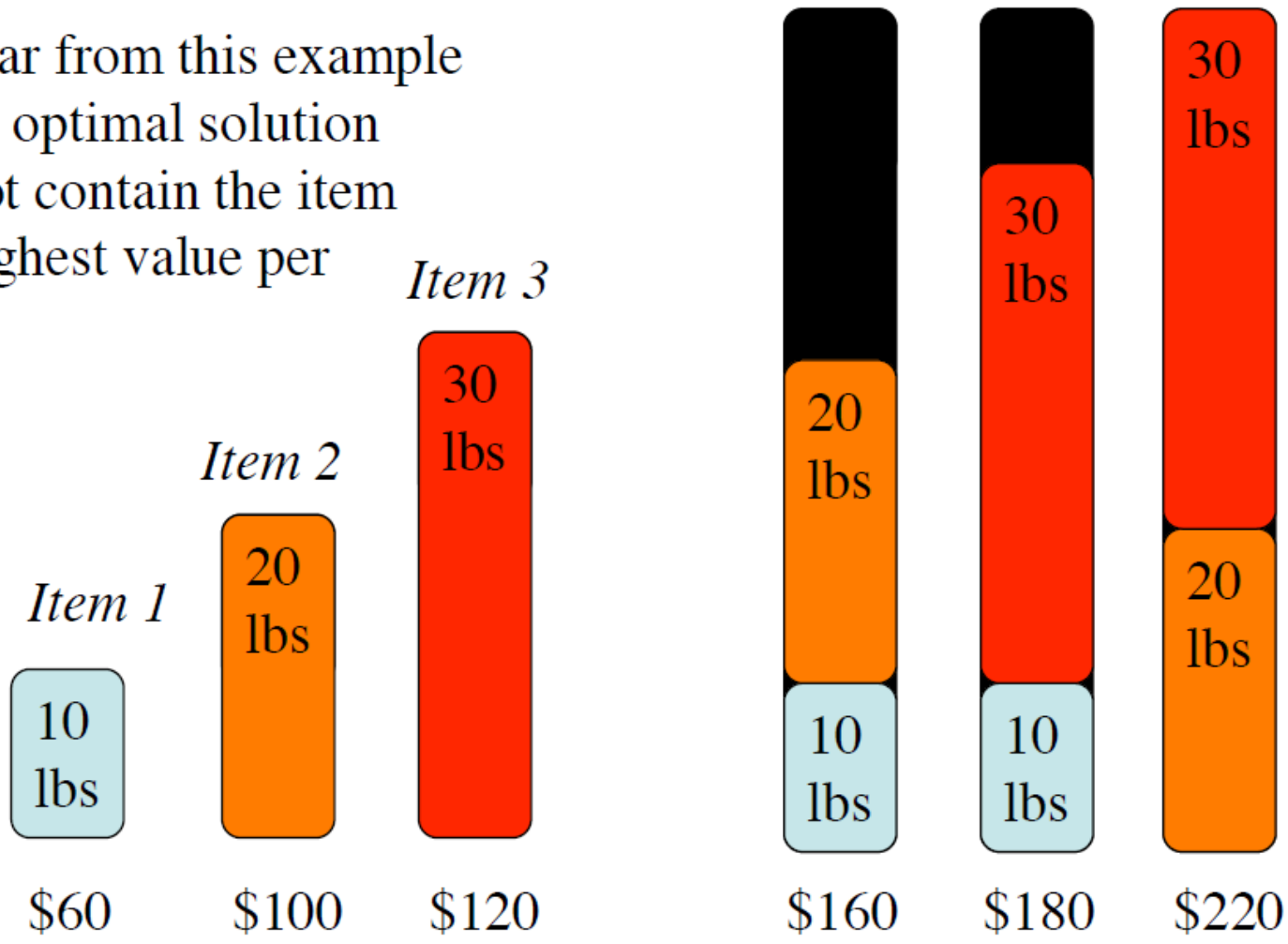


The cost per pound of:
Item 1 = \$60/10 lbs = \$6/lb
Item 2 = \$100/20 lb = \$5/lb
Item 3 = \$120/30 lb = \$4/lb

So how about if we add these items to our backpack in order of value? Suppose our backpack can only hold 50 lbs

Greedy ist nicht immer optimal...

It is clear from this example that the optimal solution does not contain the item with highest value per pound.



…aber gelegentlich eben doch

Problem 1.2.9 (FRACTIONALKNAPSACK):

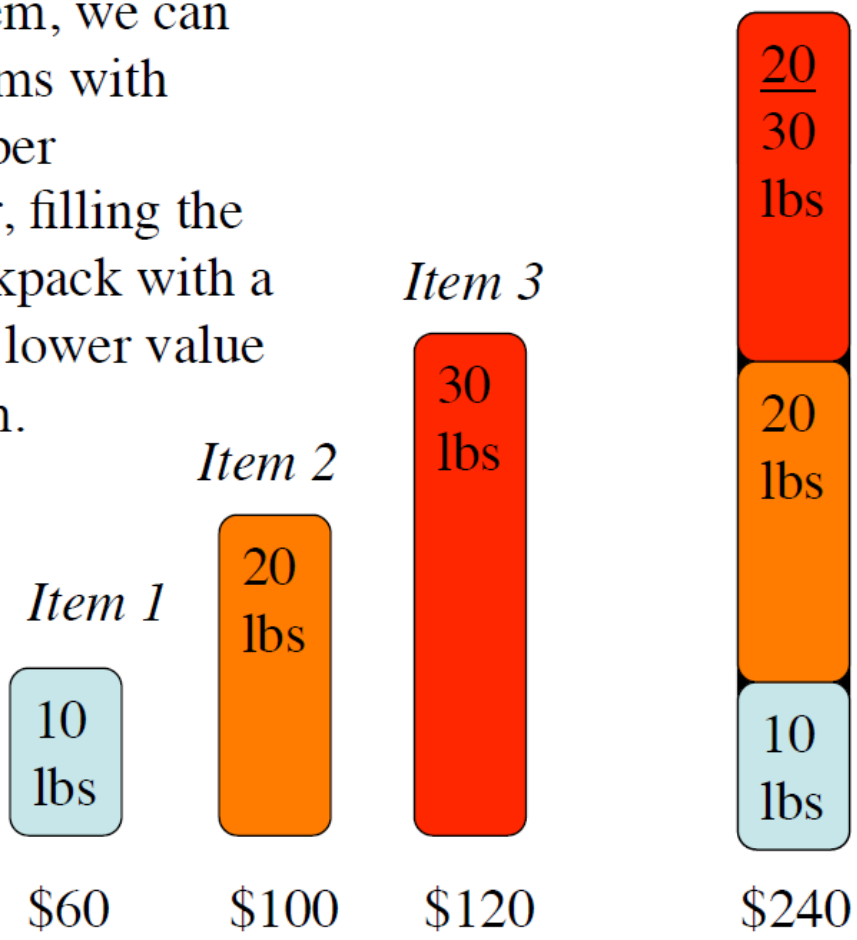
Input: *as for* 0-1-KNAPSACK.

Output: *A vector* $x \in [0,1]^n$ *such that* $\sum_{i \in S} x_i \cdot w_i \leq W$ *and* $\sum_{i \in S} x_i \cdot v_i \to \max$.

This is called the fractional knapsack problem because any fraction of each item can be used. Using a greedy strategy to solve this problem, we pack the items in order of their benefit/ weight value.

…aber gelegentlich eben doch

For this problem, we can include the items with highest value per pound in order, filling the rest of the backpack with a fraction of the lower value per pound item.



Notice that the overall benefit is higher than that achieved with the 0/1 version of the problem

Zwei fundamentale Fragen

- Wann kann ein Greedy Algorithmus zu einer optimalen Lösung für ein nichttriviales Problem führen?

- Wie kann man beweisen, dass ein Greedy Algorithmus eine optimale Lösung liefert?

Greedy Methode

- **Greedy as Algorithmic technique** can optimally solve an optimization problem,
  - if the **greedy algorithm stays ahead**. Show that after each step of the greedy algorithm, its solution is at least as good as any other algorithm's or
  - if we can find an **exchange argument**. Gradually transform any solution to the one found by the greedy algorithm without hurting its quality or
  - if the problem has the **greedy-choice property**. Prove that a globally-optimal solution can always be found by a series of local improvements from a starting configuration or
  - if we can define an adequate **Matroid** for the problem.

Kapitel 1: Greedy Algorithms

Inhalt:

- Interval Scheduling
  - Greedy stays ahead
- Scheduling to Minimize Lateness
  - Exchange Argument
- Theoretical Foundations of the Greedy Method
  - Matroid Theory

Activity Selection

Suppose we have some activities that wish to use a single resource. Each activity i has some starting time s_i and finishing time $f_i > s_i$, $i \in S$. activity i takes place in the time interval $[s_i, f_i)$, $i \in S$. Two activities i and j are called compatible iff $[s_i, f_i) \cap [s_j, f_j) = \emptyset$.

Problem 1.2.6 (ACTIVITYSELECTION):

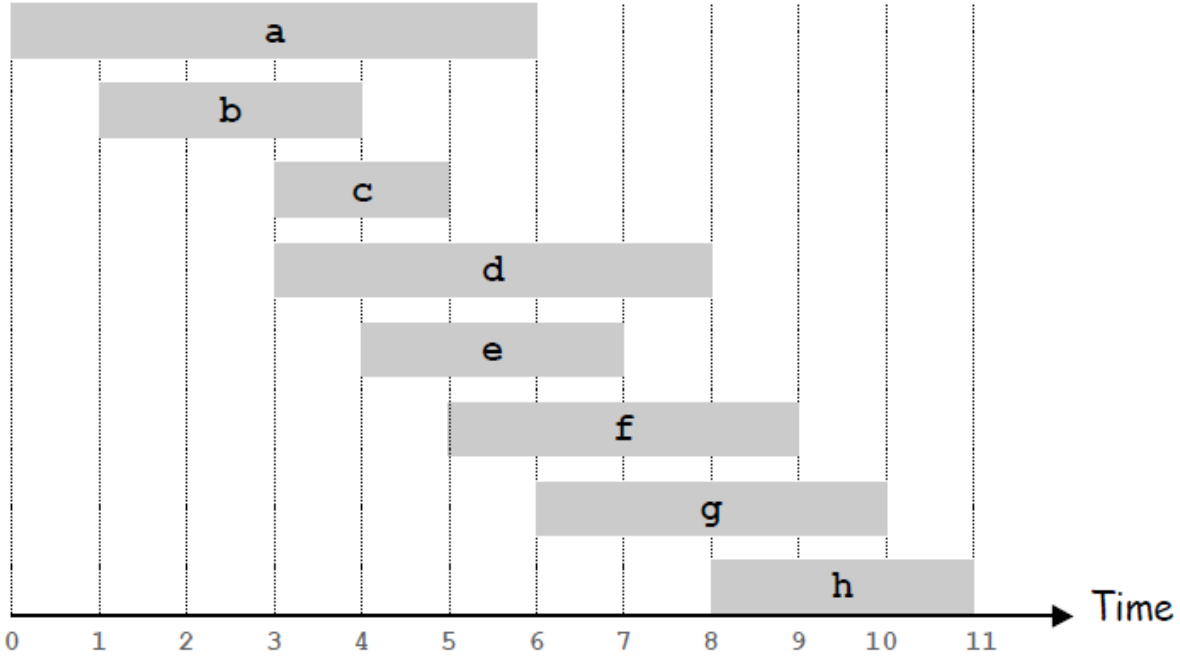
Input: *n activities $S = \{1, \ldots, n\}$ with starting times s_i and finishing times f_i, $i \in S$.*

Output: *A subset $A \subset S$ of maximum size where the elements of A are pairwise compatible.*

Interval Scheduling

Interval scheduling.

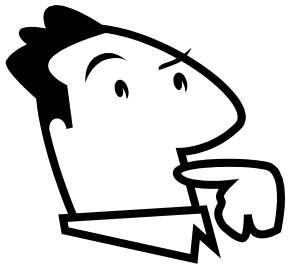
- Job j starts at s_j and finishes at f_j.
- Two jobs compatible if they don't overlap.
- Goal: find maximum subset of mutually compatible jobs.



Interval Scheduling: Greedy Algorithms

Greedy template. Consider jobs in some natural order.
Take each job provided it's compatible with the ones already taken.

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



Interval Scheduling: Greedy Algorithms

Greedy template. Consider jobs in some natural order.
Take each job provided it's compatible with the ones already taken.

counterexample for earliest start time

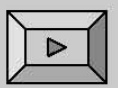
counterexample for shortest interval

counterexample for fewest conflicts

Interval Scheduling: Greedy Algorithm

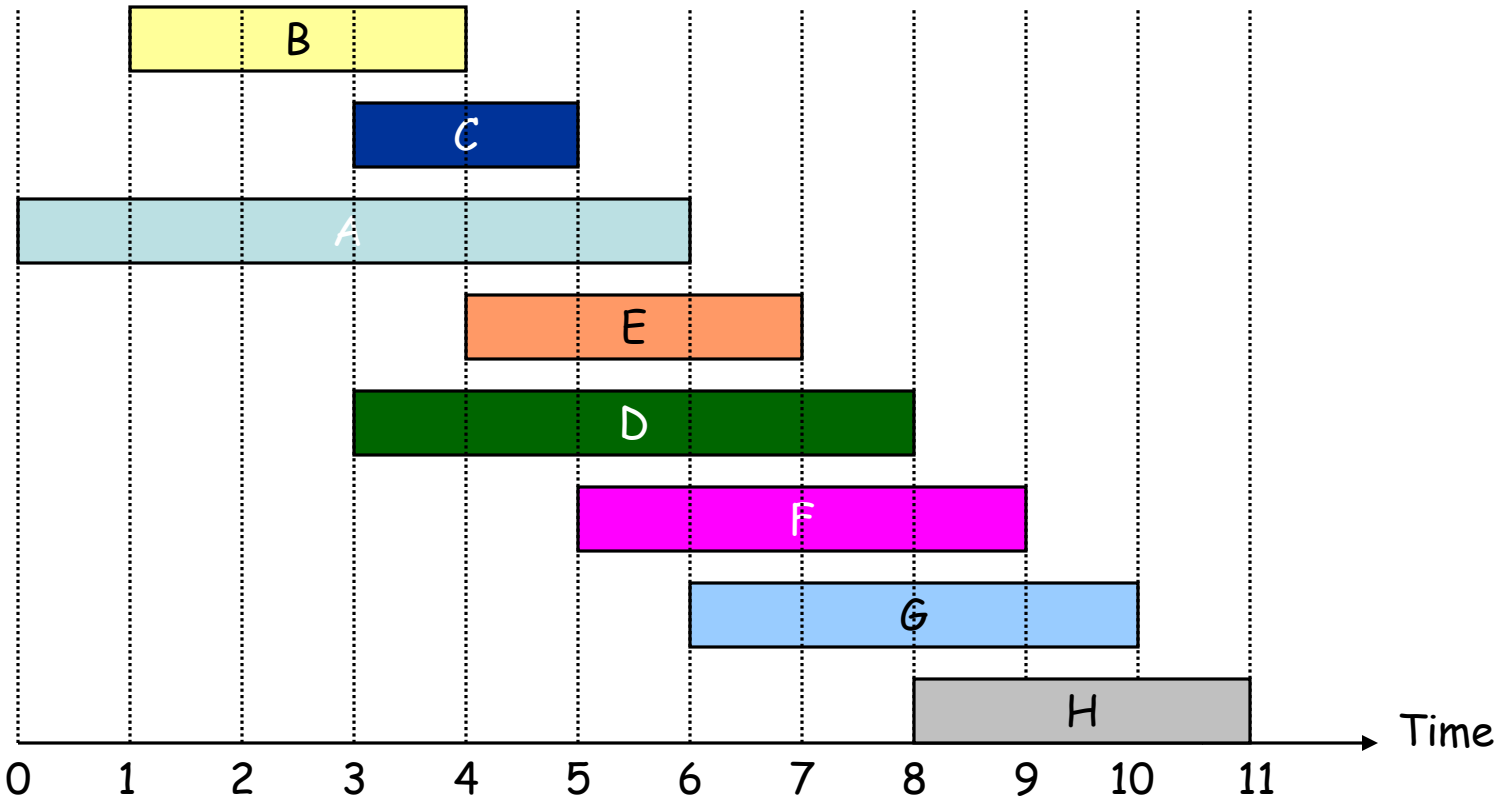
Greedy algorithm. Consider jobs in increasing order of finish time.
Take each job provided it's compatible with the ones already taken.

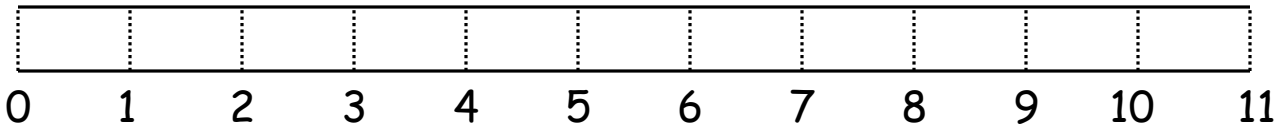
```
Sort jobs by finish times so that f_1 ≤ f_2 ≤ ... ≤ f_n.

A ← φ                          (set of jobs selected)
for j = 1 to n {
   if (job j compatible with A)
      A ← A ∪ {j}
}
return A
```

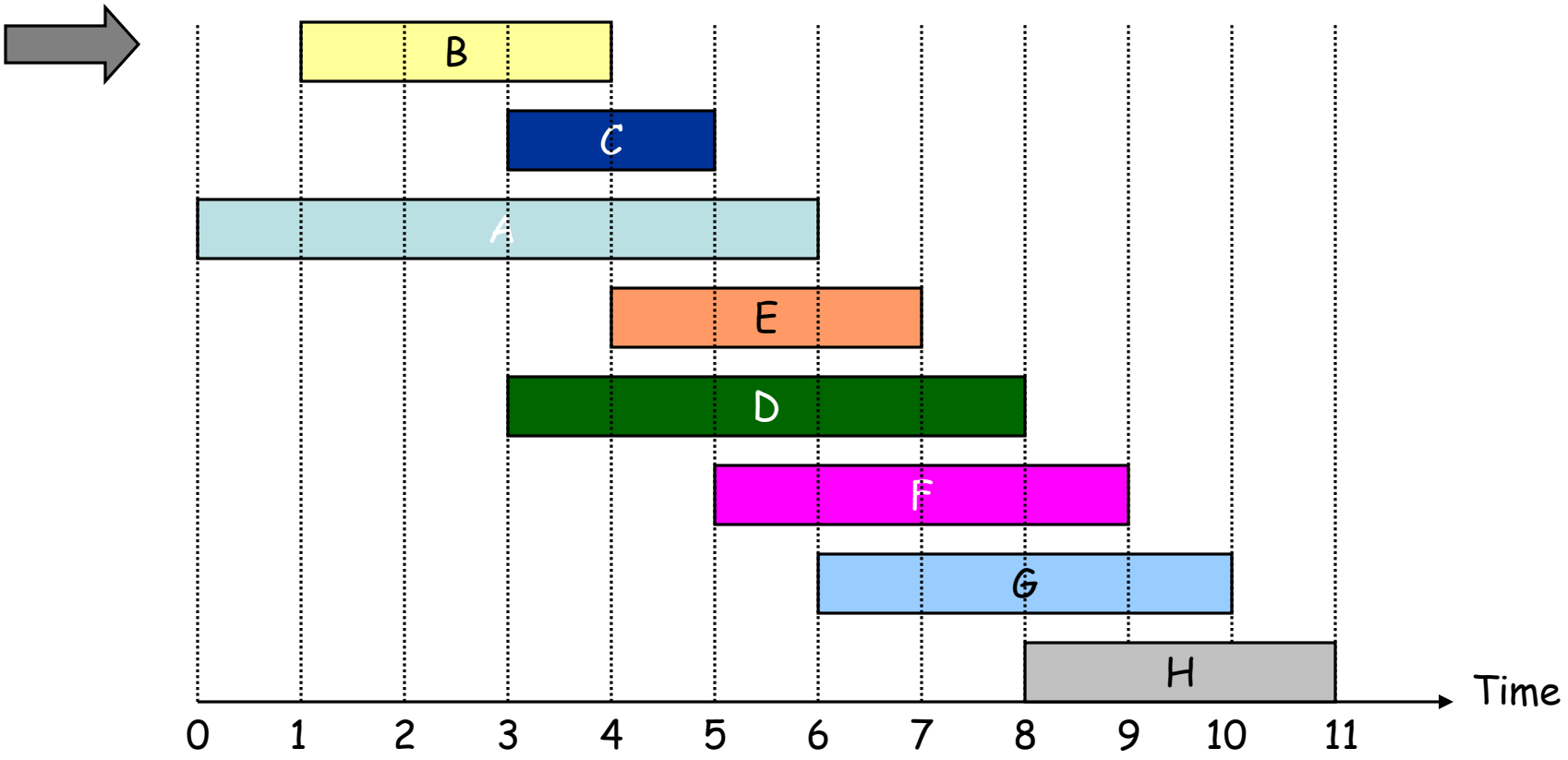


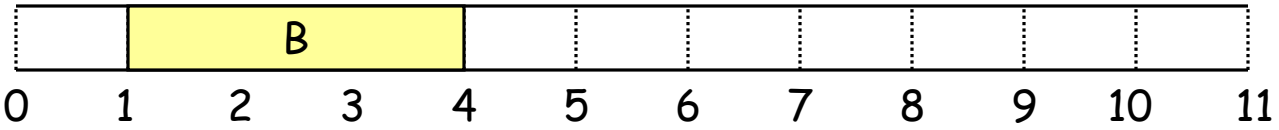
Implementation. O(n log n).

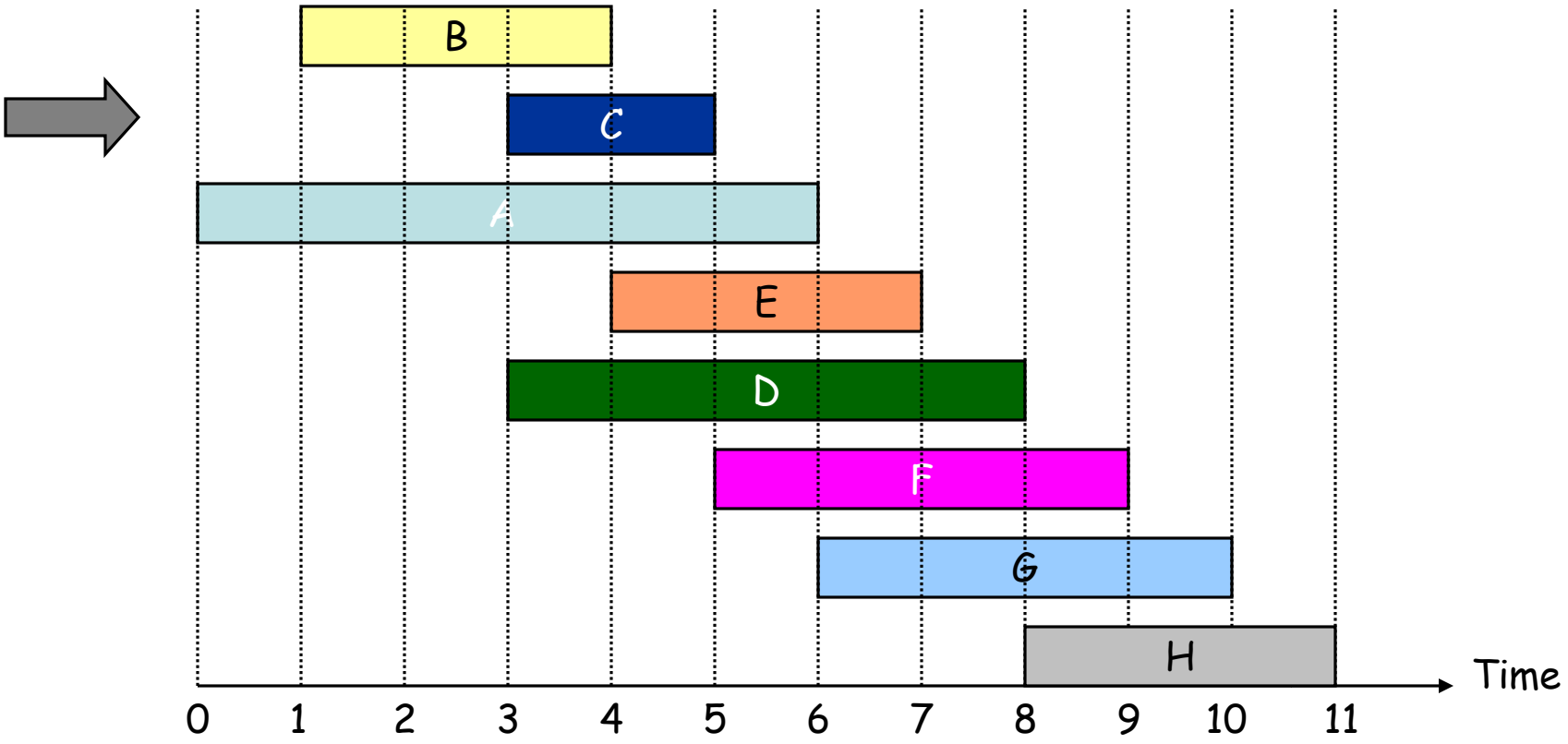
- Remember job j* that was added last to A.
- Job j is compatible with A if $s_j \geq f_{j^*}$.

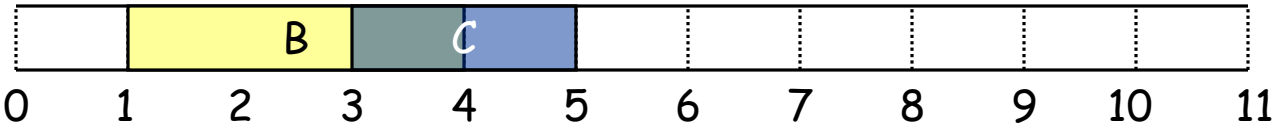

B
C
A
E
D
F
G
H
Time
0
1
2
3
4
5
6
7
8
9
10
11

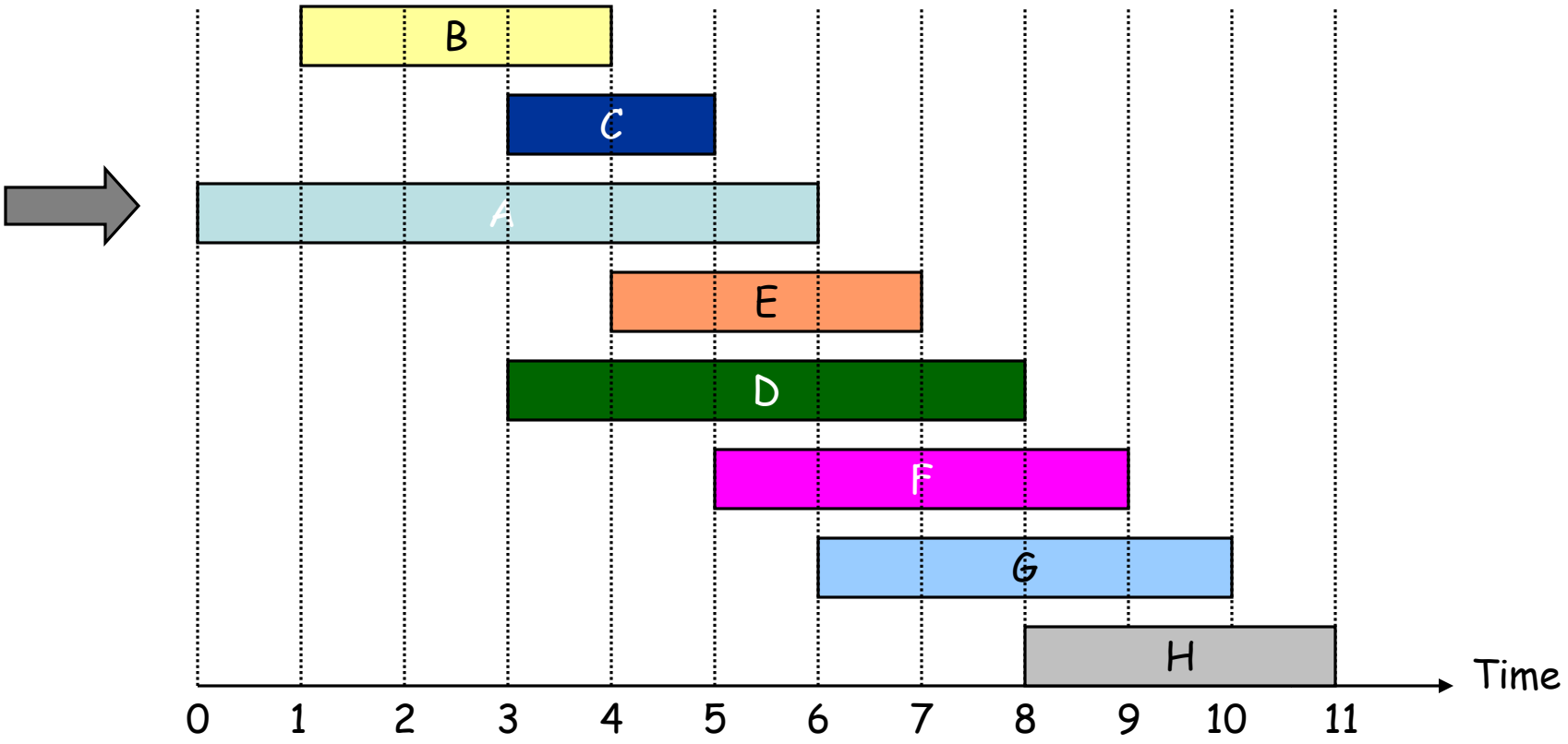

0
1
2
3
4
5
6
7
8
9
10
11

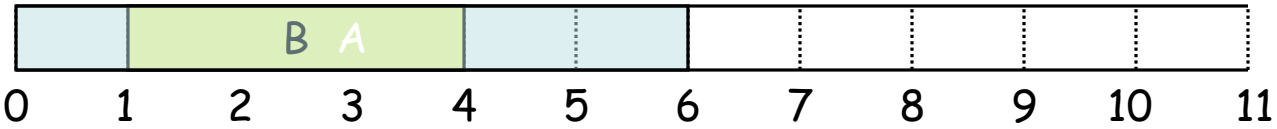

B
C
A
E
D
F
G
H
Time
0
1
2
3
4
5
6
7
8
9
10
11

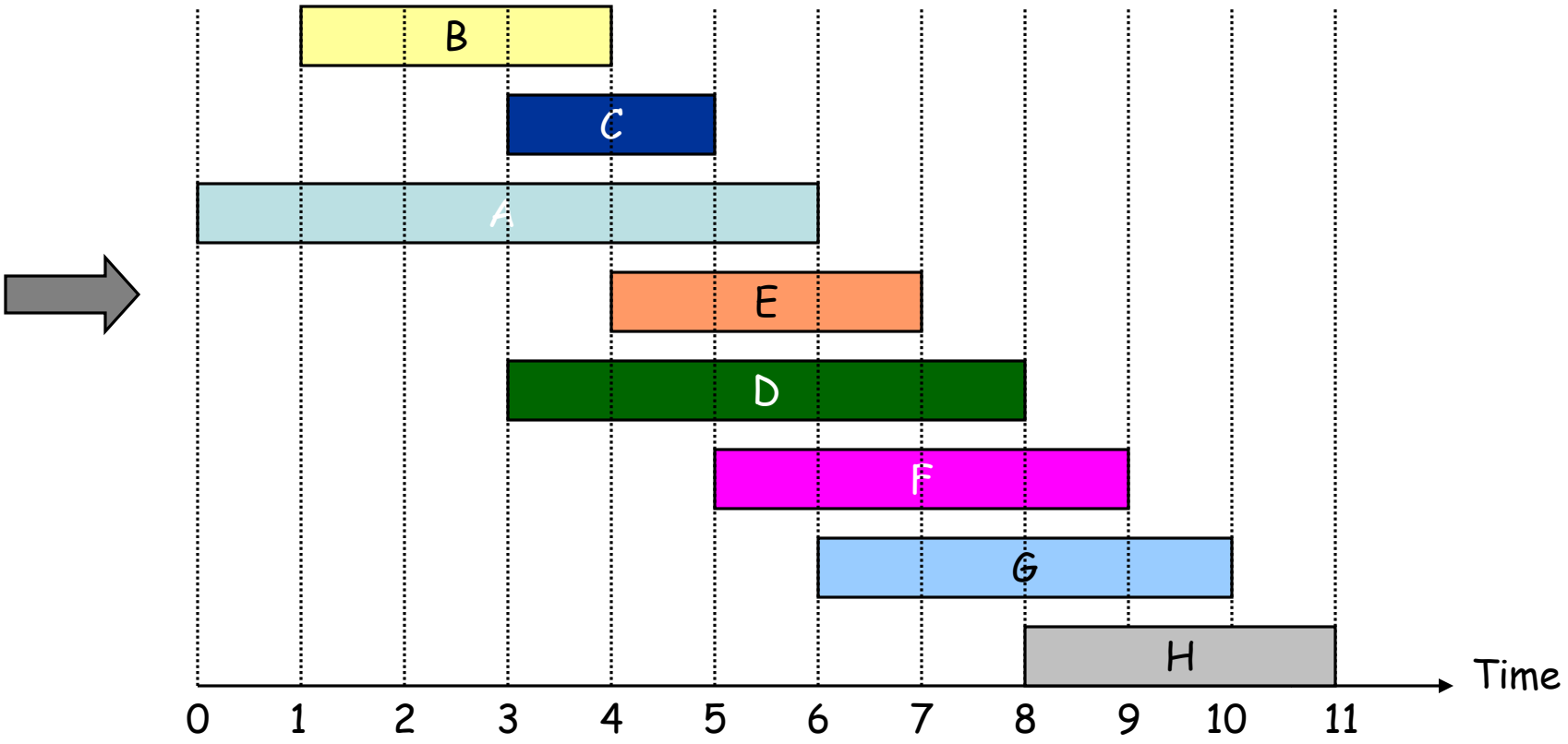

B
0
1
2
3
4
5
6
7
8
9
10
11

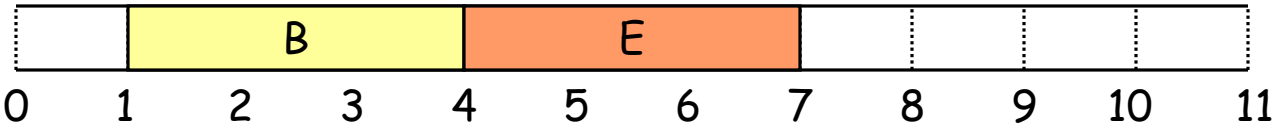

B
C
A
E
D
F
G
H
Time
0
1
2
3
4
5
6
7
8
9
10
11

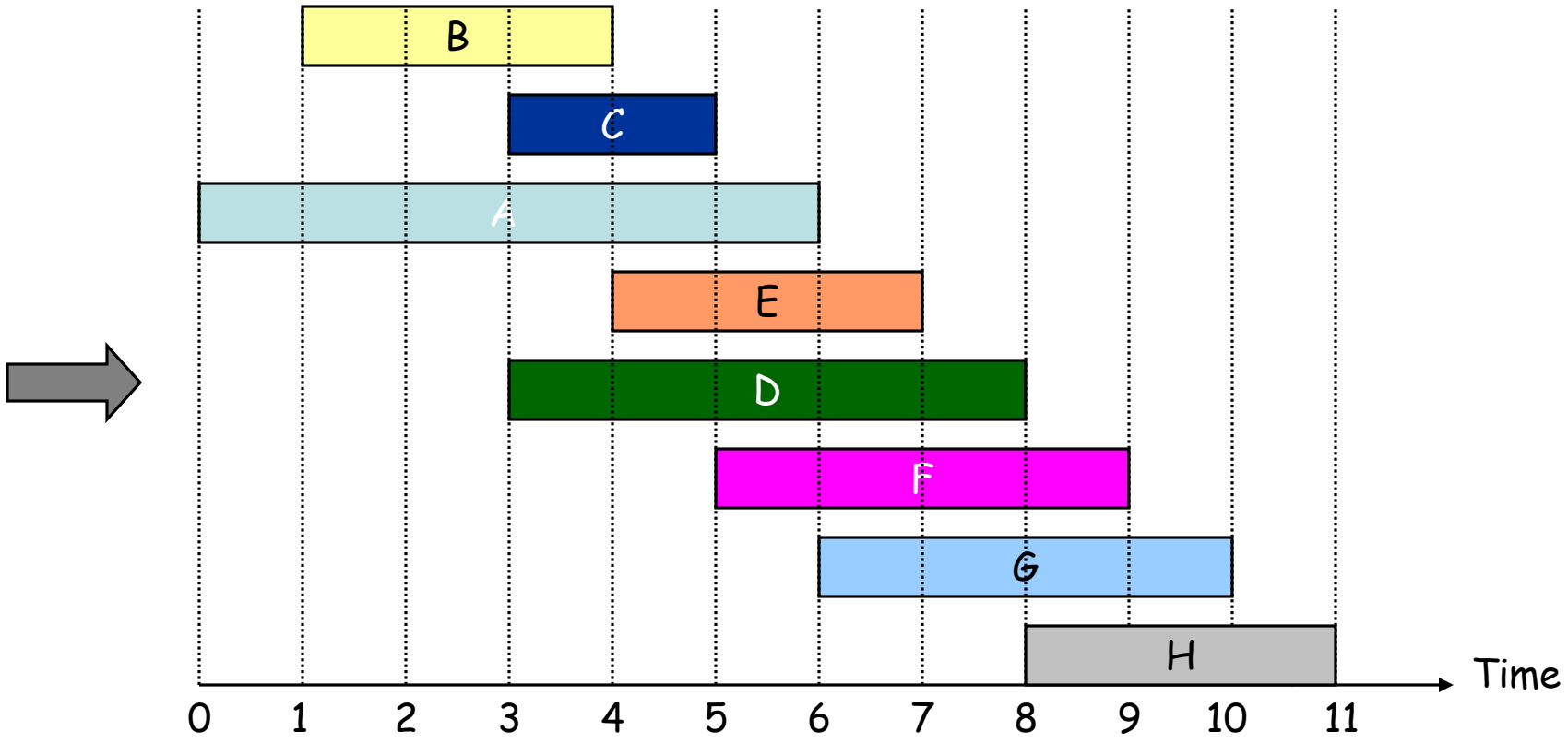

B
C
0
1
2
3
4
5
6
7
8
9
10
11

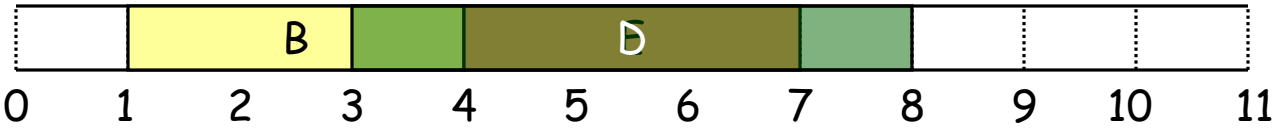

B
C
A
E
D
F
G
H
Time
0
1
2
3
4
5
6
7
8
9
10
11

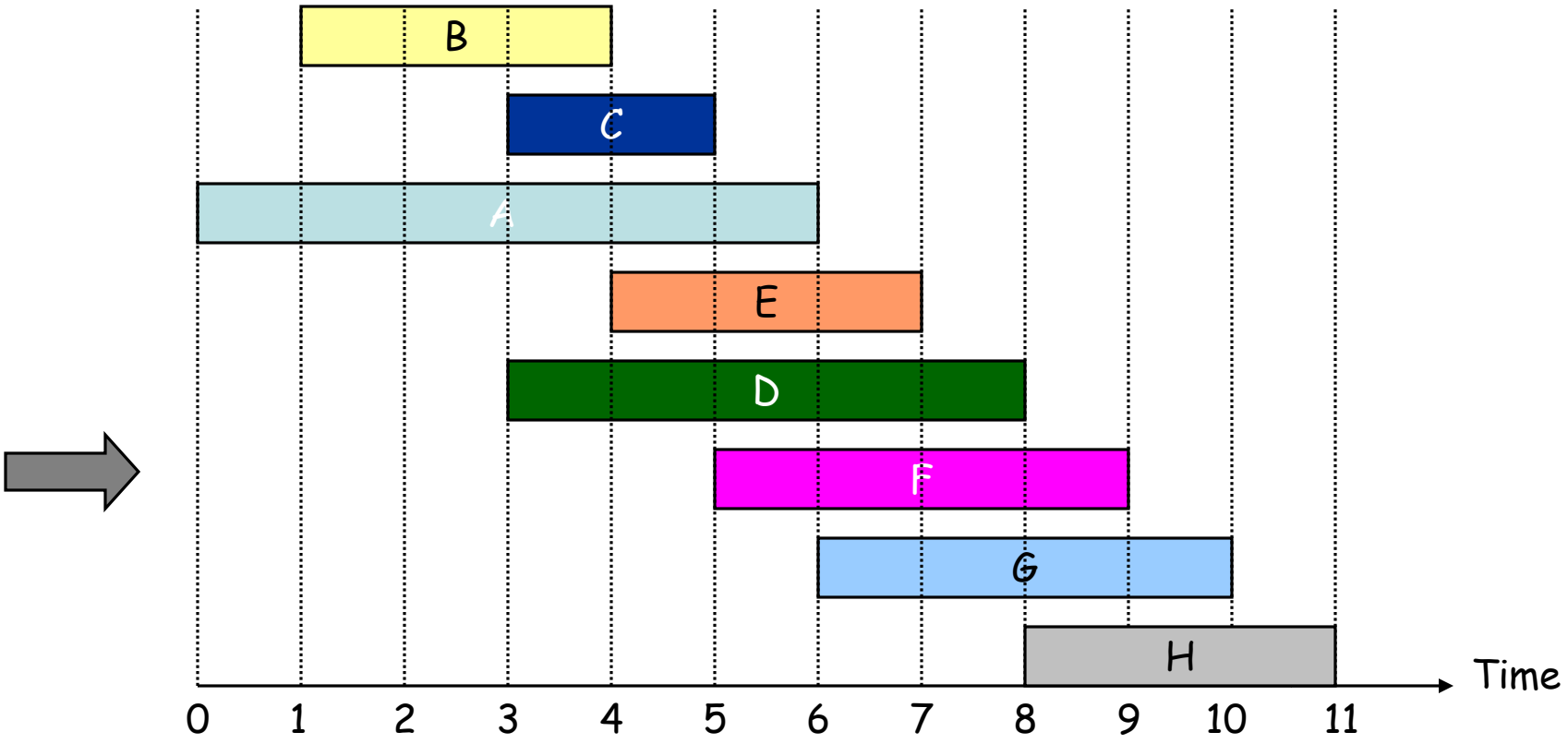

B
A
0
1
2
3
4
5
6
7
8
9
10
11

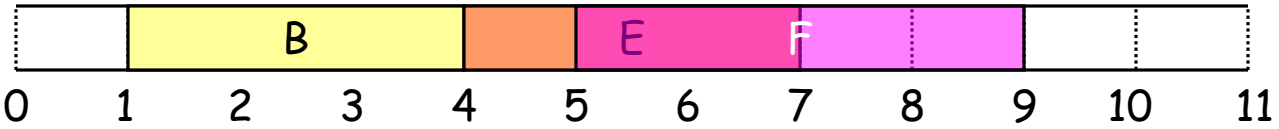

B
C
A
E
D
F
G
H
Time
0
1
2
3
4
5
6
7
8
9
10
11

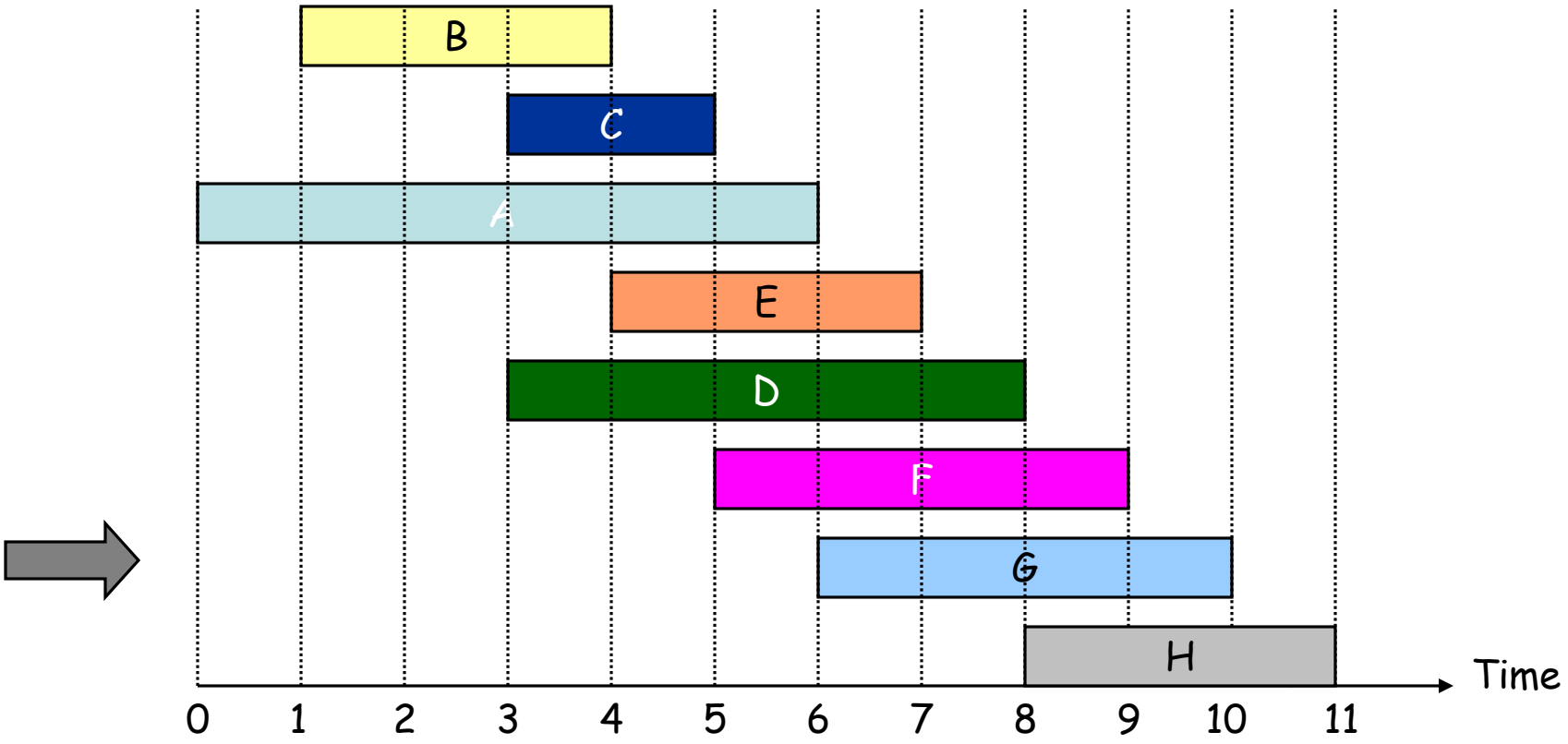

B
E
0
1
2
3
4
5
6
7
8
9
10
11

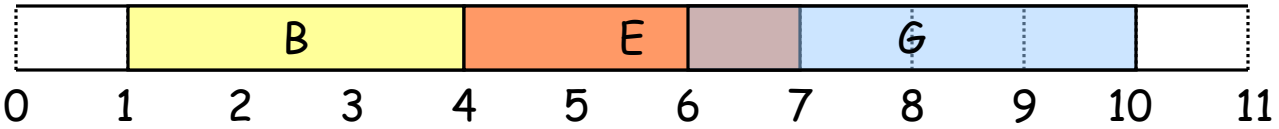

B
C
A
E
D
F
G
H
Time
0
1
2
3
4
5
6
7
8
9
10
11

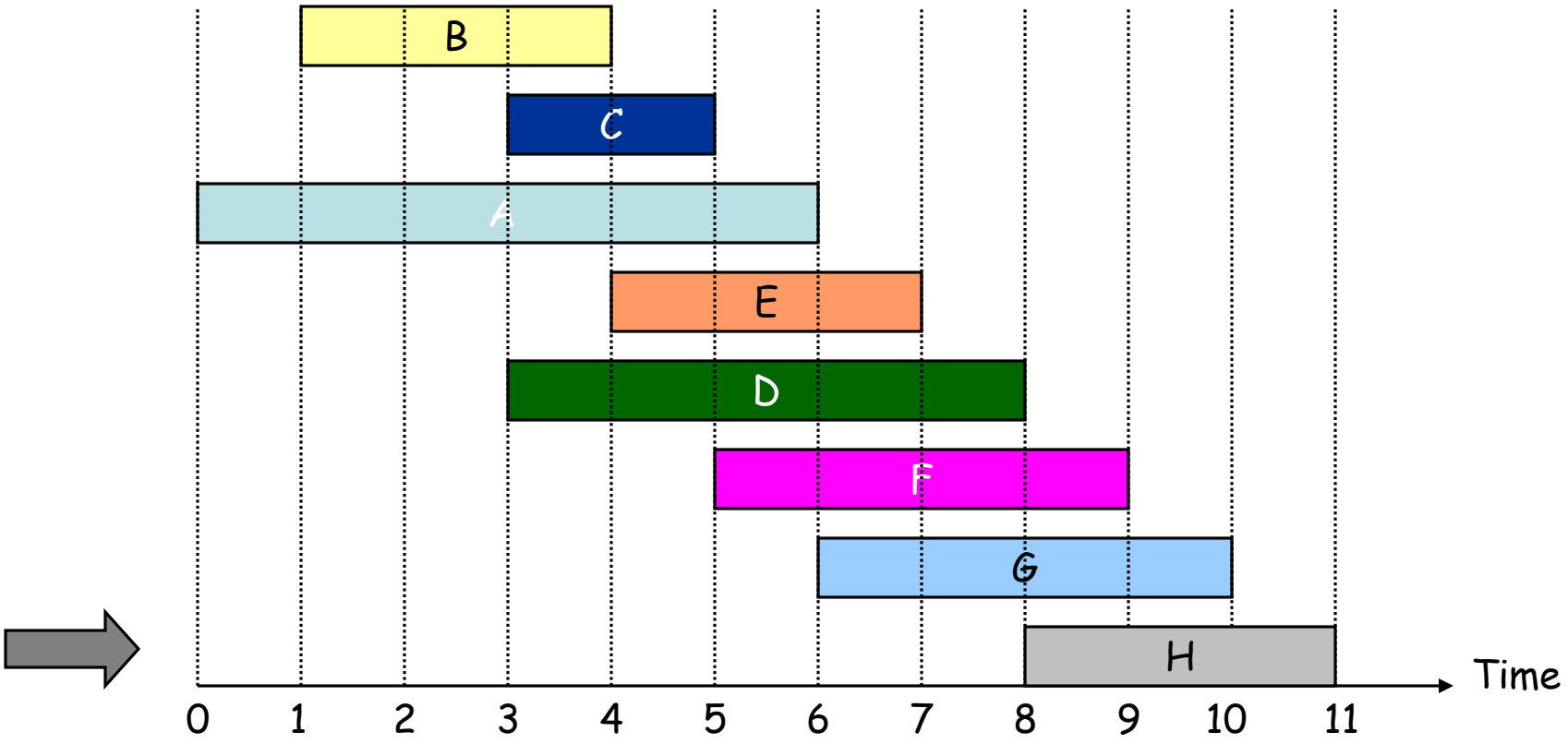

B
D
0
1
2
3
4
5
6
7
8
9
10
11

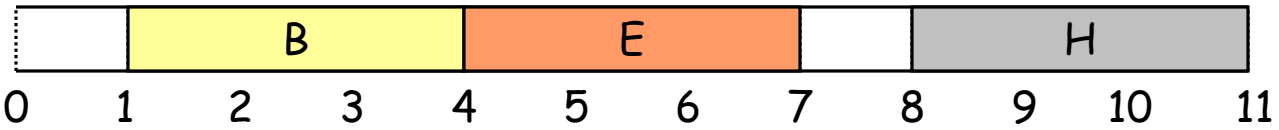

B
C
A
E
D
F
G
H
Time
0
1
2
3
4
5
6
7
8
9
10
11


B
E
F
0
1
2
3
4
5
6
7
8
9
10
11


B
C
A
E
D
F
G
H
Time
0
1
2
3
4
5
6
7
8
9
10
11


B
E
G
0
1
2
3
4
5
6
7
8
9
10
11


B
C
A
E
D
F
G
H
Time
0
1
2
3
4
5
6
7
8
9
10
11

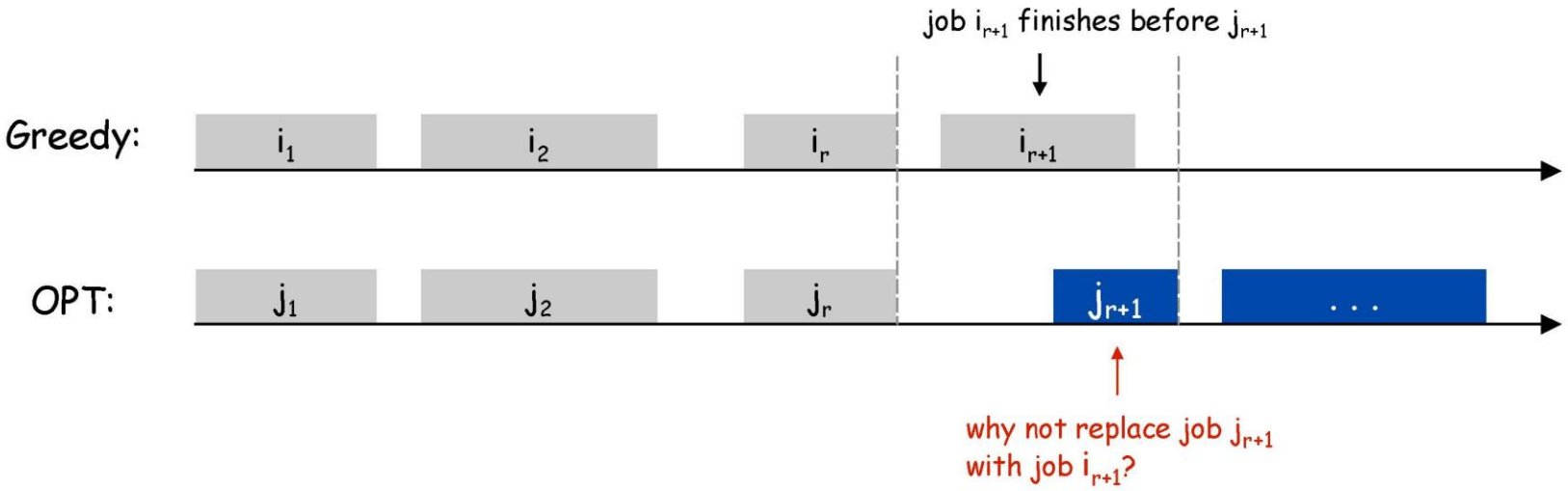

B
E
H
0
1
2
3
4
5
6
7
8
9
10
11

Interval Scheduling: Analysis

Theorem. Greedy algorithm is optimal.

Pf. (by contradiction)

- Assume greedy is not optimal, and let's see what happens.
- Let $i_1, i_2, \ldots i_k$ denote set of jobs selected by greedy.
- Let $j_1, j_2, \ldots j_m$ denote set of jobs in the optimal solution with $i_1 = j_1, i_2 = j_2, \ldots, i_r = j_r$ for the largest possible value of r.

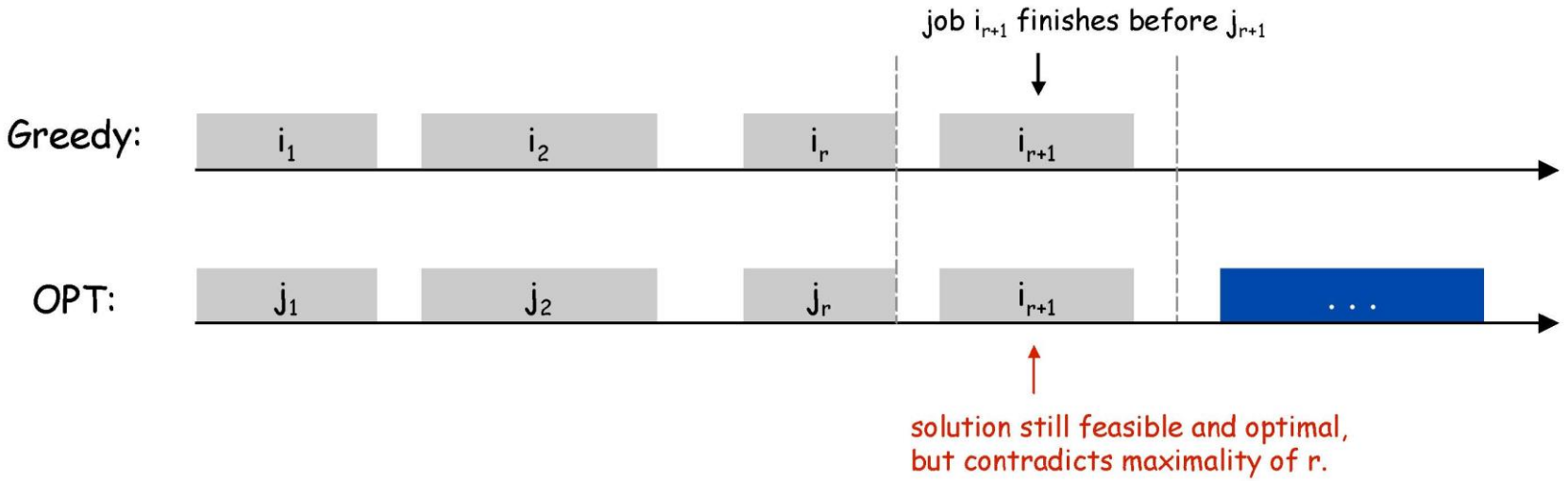


Interval Scheduling: Analysis

Theorem. Greedy algorithm is optimal.

Pf. (by contradiction)

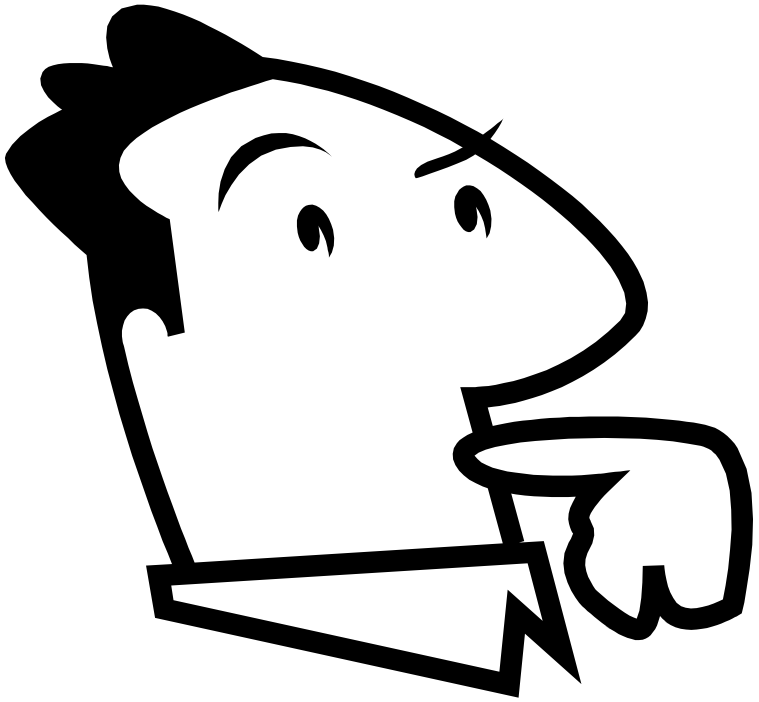
- Assume greedy is not optimal, and let's see what happens.
- Let $i_1, i_2, \ldots i_k$ denote set of jobs selected by greedy.
- Let $j_1, j_2, \ldots j_m$ denote set of jobs in the optimal solution with $i_1 = j_1, i_2 = j_2, \ldots, i_r = j_r$ for the largest possible value of r.



Was haben wir gezeigt ?

If one measures the Greedy Algorithm's progress in a step-by-step fashion, one sees that it does better than any other algorithm at each step ("greedy stays ahead"); it then follows that it produces an optimal solution!

Fragen?



Kapitel 1: Greedy Algorithms

Inhalt:

- Intervall Scheduling
  - Greedy stays ahead
- Scheduling to Minimize Lateness
  - Exchange Argument
- Theoretical Foundations of the Greedy Method
  - Matroid Theory

Scheduling to Minimizing Lateness

Minimizing lateness problem.

- Single resource processes one job at a time.
- Job j requires t_j units of processing time and is due at time d_j.
- If j starts at time s_j, it finishes at time $f_j = s_j + t_j$.
- Lateness: $\ell_j = \max \{ 0, f_j - d_j \}$.
- Goal: schedule all jobs to minimize maximum lateness $L = \max \ell_j$.

Ex:

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15

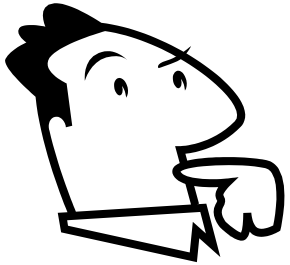
Minimizing Lateness: Greedy Algorithms

Greedy template. Consider jobs in some order.

- [Shortest processing time first] Consider jobs in ascending order of processing time t_j.

- [Earliest deadline first] Consider jobs in ascending order of deadline d_j.

- [Smallest slack] Consider jobs in ascending order of slack $d_j - t_j$.



Minimizing Lateness: Greedy Algorithms

Greedy template. Consider jobs in some order.

- [Shortest processing time first] Consider jobs in ascending order of processing time t_j.

	1	2
t_j	1	10
d_j	100	10

counterexample

- [Smallest slack] Consider jobs in ascending order of slack $d_j - t_j$.

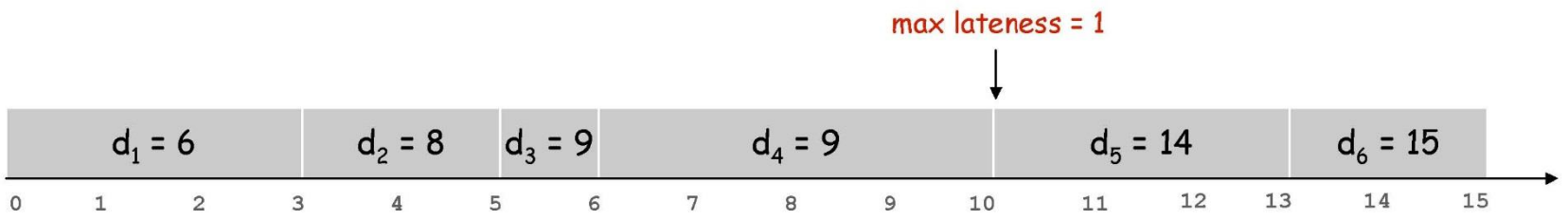
	1	2
t_j	1	10
d_j	2	10

counterexample

Minimizing Lateness: Greedy Algorithm

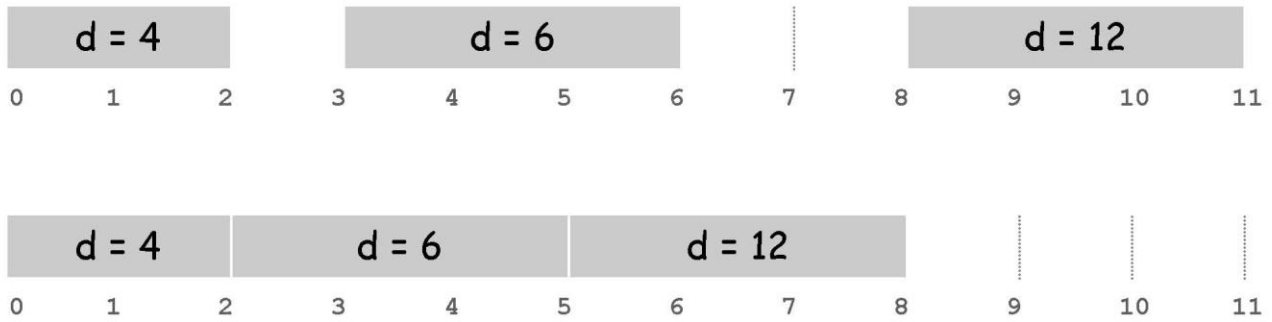
Greedy algorithm. Earliest deadline first.

```
Sort n jobs by deadline so that d_1 ≤ d_2 ≤ … ≤ d_n

t ← 0
for j = 1 to n
  // Assign job j to interval [t, t + t_j]
    s_j ← t, f_j ← t + t_j
    t ← t + t_j
output intervals [s_j, f_j]
```



Minimizing Lateness: No Idle Time

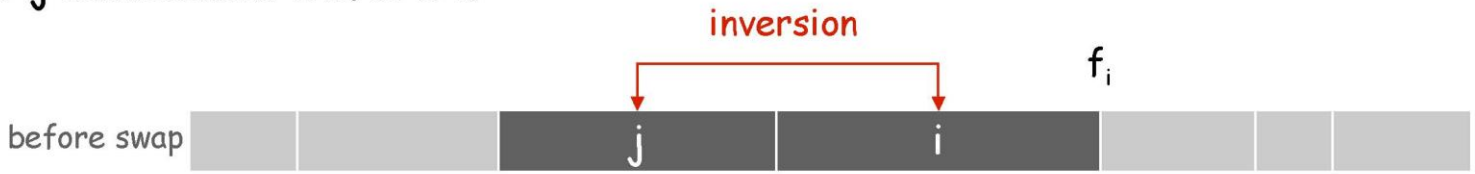
Observation. There exists an optimal schedule with no idle time.



Observation. The greedy schedule has no idle time.

Minimizing Lateness: Inversions

Def. Given a schedule S, an inversion is a pair of jobs i and j such that: i < j but j scheduled before i.



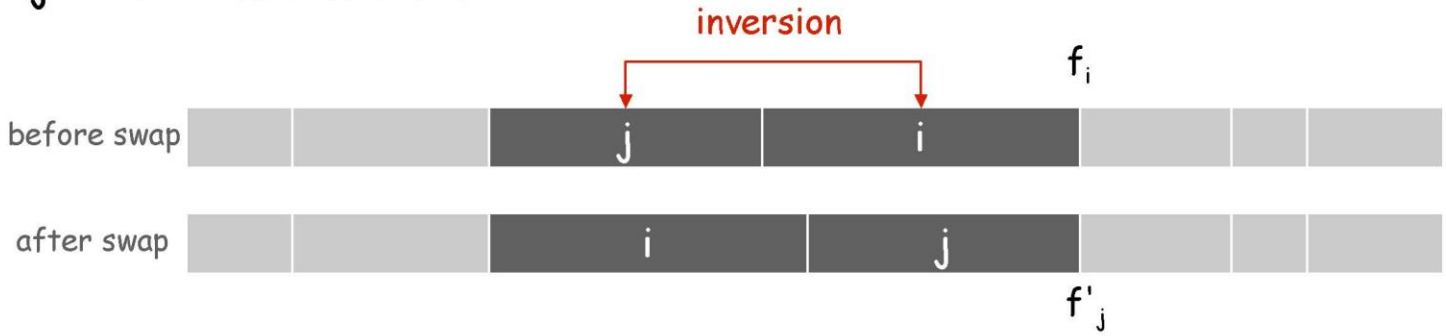
[as before, we assume jobs are numbered so that $d_1 \leq d_2 \leq \ldots \leq d_n$]

Observation. Greedy schedule has no inversions.

Observation. If a schedule (with no idle time) has an inversion, it has one with a pair of inverted jobs scheduled consecutively.

Minimizing Lateness: Inversions

Def. Given a schedule S, an **inversion** is a pair of jobs i and j such that: i < j but j scheduled before i.



Claim. Swapping two consecutive, inverted jobs reduces the number of inversions by one and does not increase the max lateness.

Pf. Let ℓ be the lateness before the swap, and let ℓ' be it afterwards.

- $\ell'_k = \ell_k$ for all $k \neq i, j$
- $\ell'_i \leq \ell_i$
- If job j is late:

$$
\begin{aligned}
\ell'_j &= f'_j - d_j && \text{(definition)} \\
&= f_i - d_j && (j \text{ finishes at time } f_i) \\
&\leq f_i - d_i && (i < j) \\
&\leq \ell_i && \text{(definition)}
\end{aligned}
$$

Minimizing Lateness: Analysis of Greedy Algorithm

Theorem. Greedy schedule S is optimal.

Pf. Define S* to be an optimal schedule that has the fewest number of inversions, and let's see what happens.

- Can assume S* has no idle time.
- If S* has no inversions, then S = S*.
- If S* has an inversion, let i-j be an adjacent inversion.
  - swapping i and j does not increase the maximum lateness and strictly decreases the number of inversions
  - this contradicts definition of S* ▪

Was haben wir gezeigt ?

Any possible solution to the problem can be transformed (Exchange Property) into the solution found by the Greedy Algorithm without hurting its quality; it then follows that the greedy algorithm must have found a solution that is at least as good as any other solution!

Fragen?