

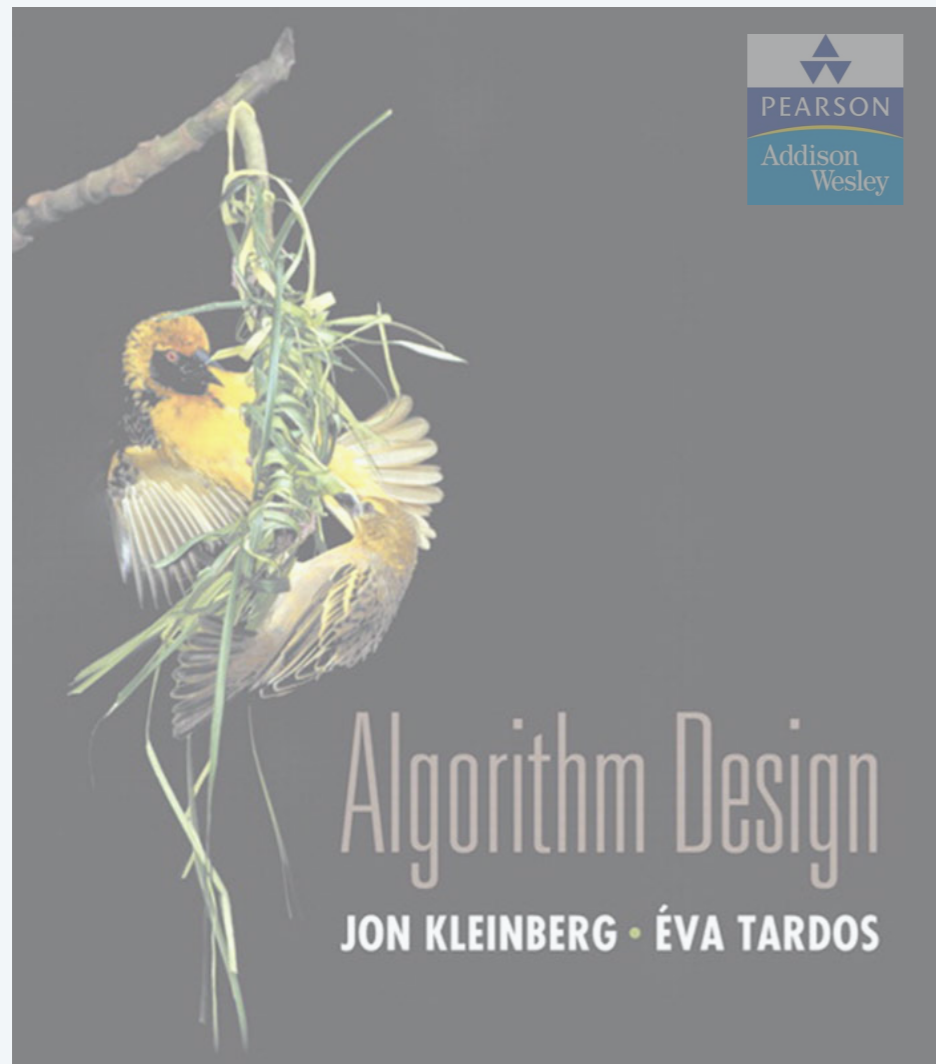
4. GREEDY ALGORITHMS I

- ▶ *coin changing*
- ▶ *interval scheduling*
- ▶ *interval partitioning*
- ▶ *scheduling to minimize lateness*
- ▶ *optimal caching*

Lecture slides by Kevin Wayne

Copyright © 2005 Pearson–Addison Wesley

<http://www.cs.princeton.edu/~wayne/kleinberg-tardos>



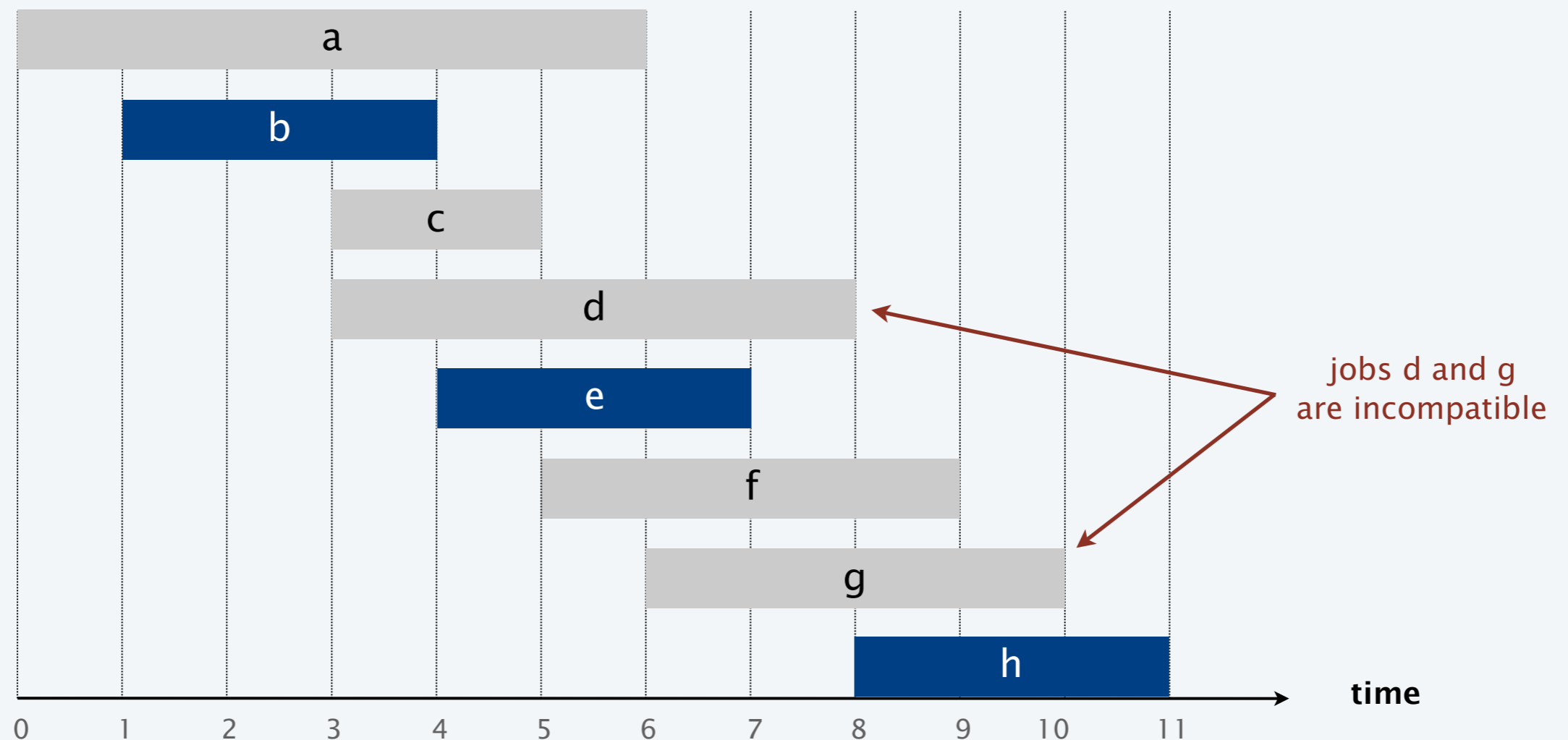
SECTION 4.1

4. GREEDY ALGORITHMS I

- ▶ *coin changing*
- ▶ ***interval scheduling***
- ▶ *interval partitioning*
- ▶ *scheduling to minimize lateness*
- ▶ *optimal caching*

Interval scheduling

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





Consider jobs in some order, taking each job provided it's compatible with the ones already taken. Which rule is optimal?

- A. [Earliest start time] Consider jobs in ascending order of s_j .
- B. [Earliest finish time] Consider jobs in ascending order of f_j .
- C. [Shortest interval] Consider jobs in ascending order of $f_j - s_j$.
- D. None of the above.

Interval scheduling: earliest-finish-time-first algorithm



EARLIEST-FINISH-TIME-FIRST $(n, s_1, s_2, \dots, s_n, f_1, f_2, \dots, f_n)$

SORT jobs by finish times and renumber so that $f_1 \leq f_2 \leq \dots \leq f_n$.

$S \leftarrow \emptyset$. \leftarrow set of jobs selected

FOR $j = 1$ **TO** n

IF (job j is compatible with S)

$S \leftarrow S \cup \{ j \}$.

RETURN S .

Proposition. Can implement earliest-finish-time first in $O(n \log n)$ time.

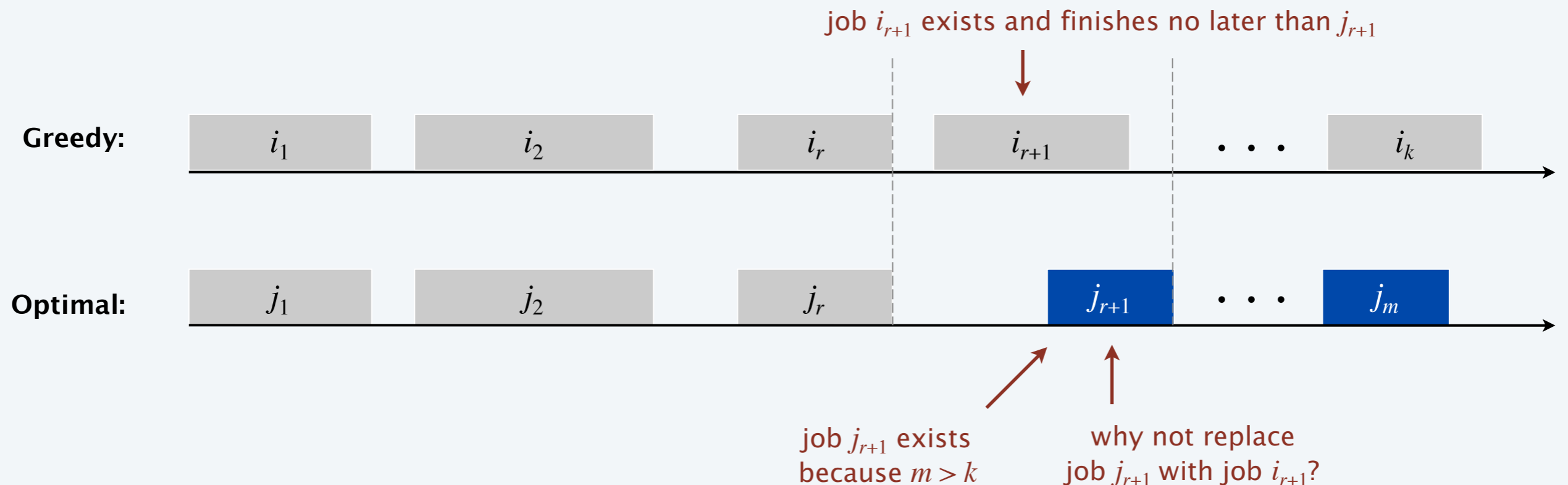
- Keep track of job j^* that was added last to S .
- Job j is compatible with S iff $s_j \geq f_{j^*}$.
- Sorting by finish times takes $O(n \log n)$ time.

Interval scheduling: analysis of earliest-finish-time-first algorithm

Theorem. The earliest-finish-time-first algorithm is optimal.

Pf. [by contradiction]

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

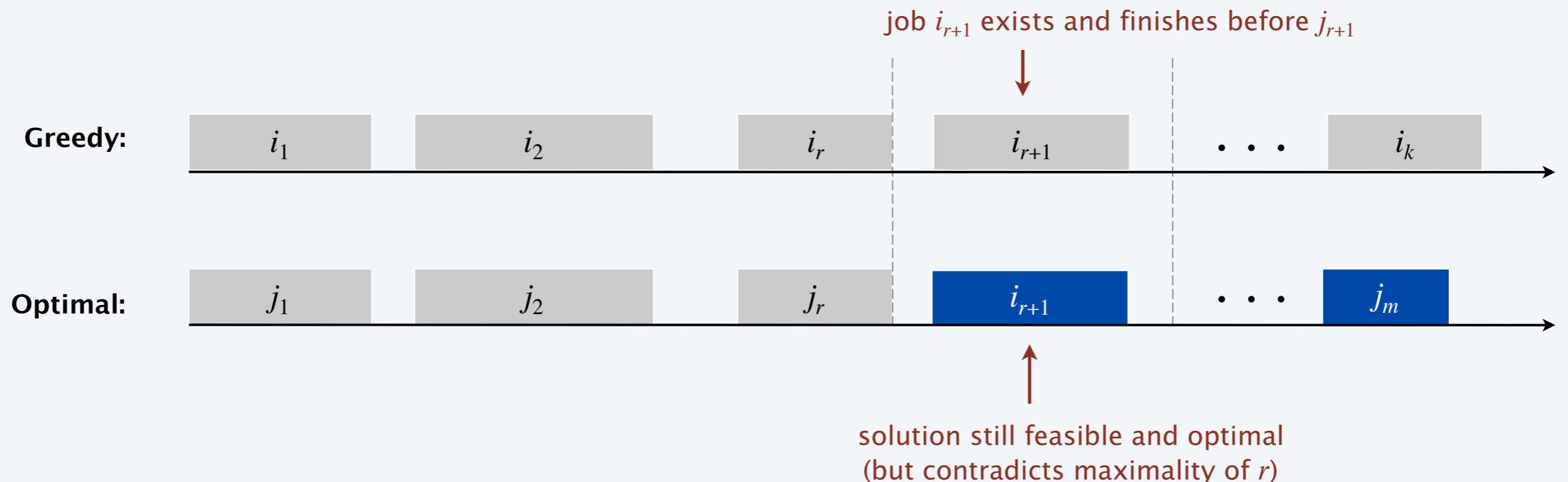


Interval scheduling: analysis of earliest-finish-time-first algorithm

Theorem. The earliest-finish-time-first algorithm is optimal.

Pf. [by contradiction]

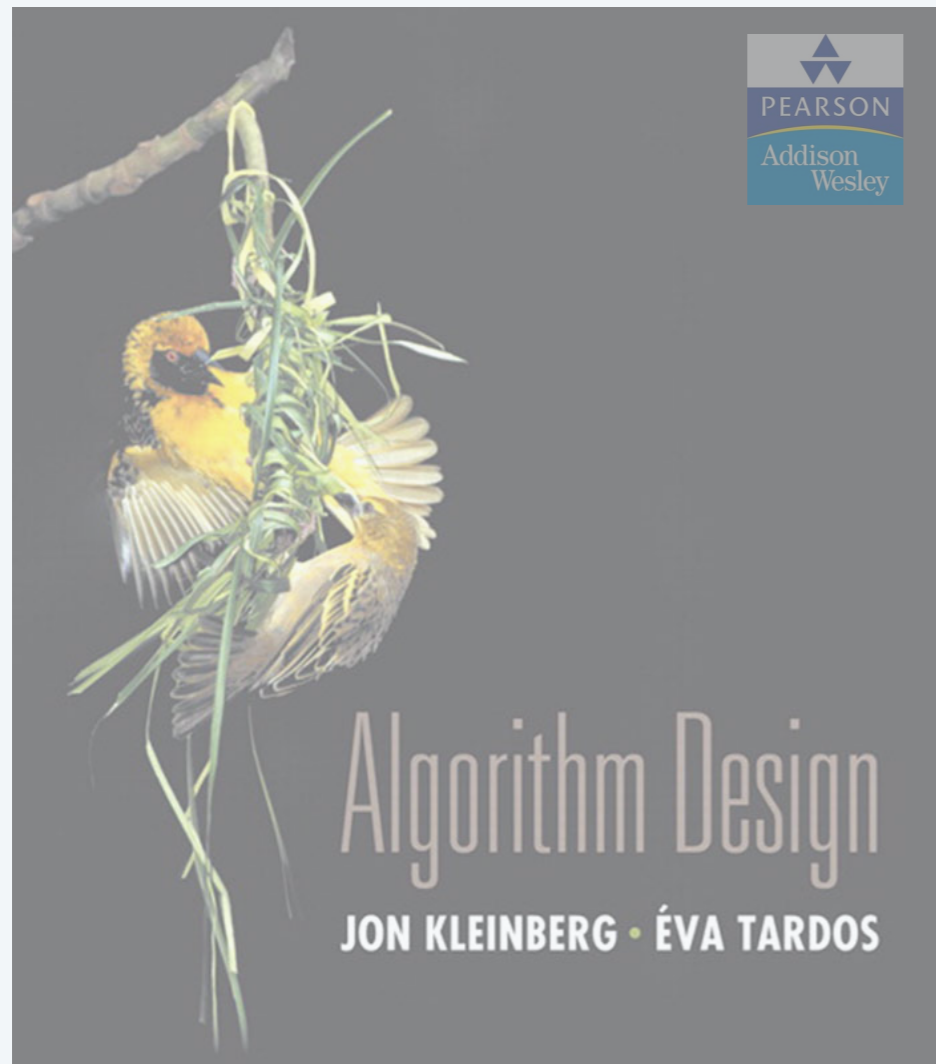
- Assume greedy is not optimal, and let's see what happens.
- Let i_1, i_2, \dots, i_k denote set of jobs selected by greedy.
- Let j_1, j_2, \dots, j_m denote set of jobs in an optimal solution with $i_1 = j_1, i_2 = j_2, \dots, i_r = j_r$ for the largest possible value of r .





Suppose that each job also has a positive weight and the goal is to find a maximum weight subset of mutually compatible intervals. Is the earliest–finish–time–first algorithm still optimal?

- A.** Yes, because greedy algorithms are always optimal.
- B.** Yes, because the same proof of correctness is valid.
- C.** No, because the same proof of correctness is no longer valid.
- D.** No, because you could assign a huge weight to a job that overlaps the job with the earliest finish time.



SECTION 4.1

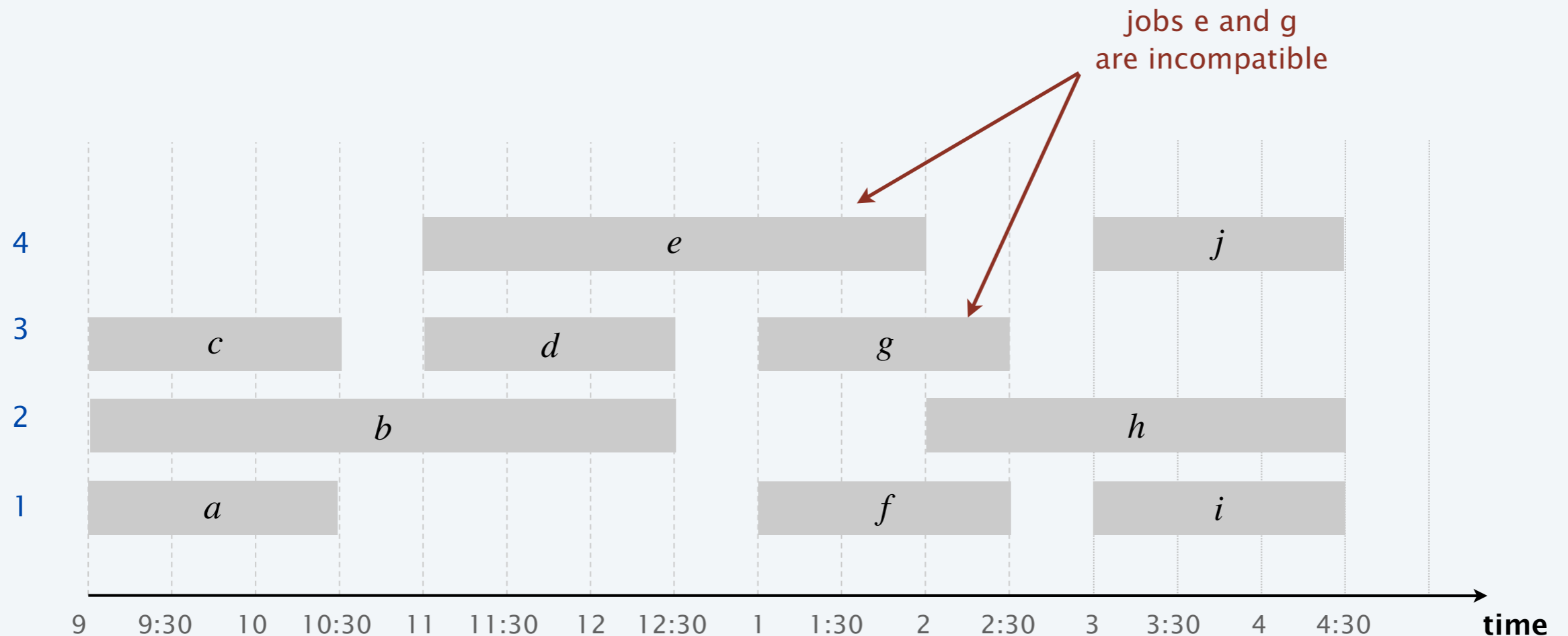
4. GREEDY ALGORITHMS I

- ▶ *coin changing*
- ▶ *interval scheduling*
- ▶ ***interval partitioning***
- ▶ *scheduling to minimize lateness*
- ▶ *optimal caching*

Interval partitioning

- Lecture j starts at s_j and finishes at f_j .
- Goal: find minimum number of classrooms to schedule all lectures so that no two lectures occur at the same time in the same room.

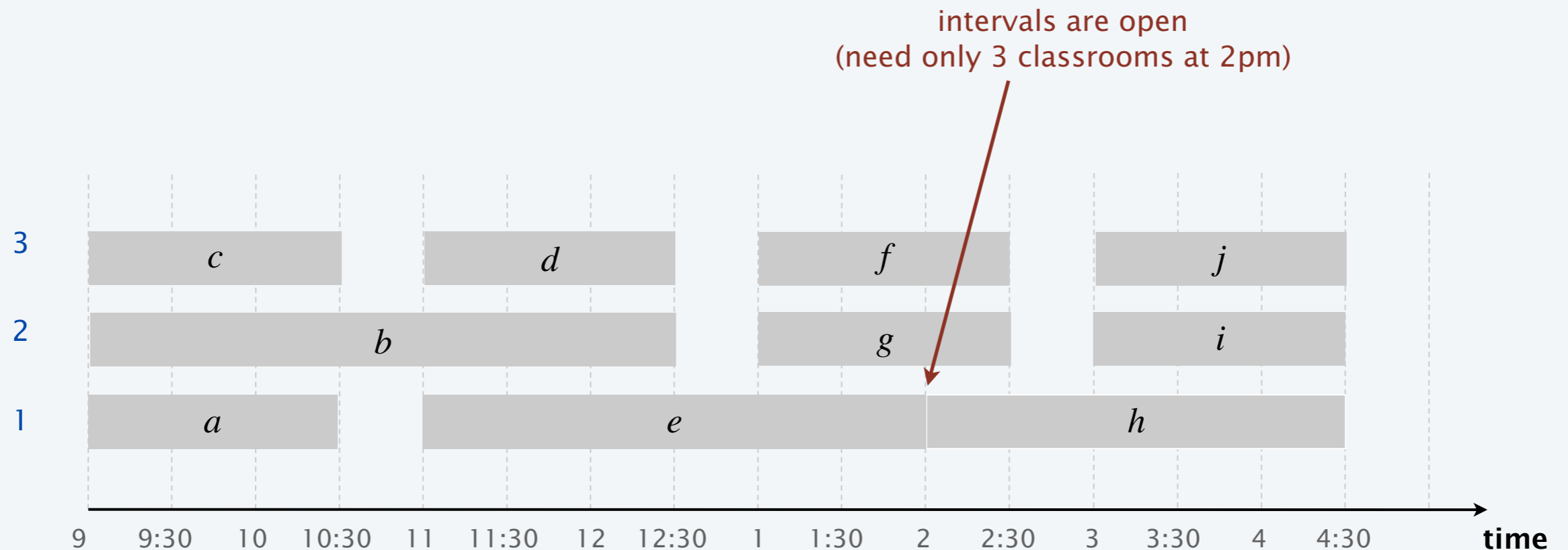
Ex. This schedule uses 4 classrooms to schedule 10 lectures.



Interval partitioning

- Lecture j starts at s_j and finishes at f_j .
- Goal: find minimum number of classrooms to schedule all lectures so that no two lectures occur at the same time in the same room.

Ex. This schedule uses **3** classrooms to schedule 10 lectures.





Consider lectures in some order, assigning each lecture to first available classroom (opening a new classroom if none is available). Which rule is optimal?

- A. [Earliest start time] Consider lectures in ascending order of s_j .
- B. [Earliest finish time] Consider lectures in ascending order of f_j .
- C. [Shortest interval] Consider lectures in ascending order of $f_j - s_j$.
- D. None of the above.

Interval partitioning: earliest-start-time-first algorithm



EARLIEST-START-TIME-FIRST ($n, s_1, s_2, \dots, s_n, f_1, f_2, \dots, f_n$)

SORT lectures by start times and renumber so that $s_1 \leq s_2 \leq \dots \leq s_n$.

$d \leftarrow 0$. \longleftarrow number of allocated classrooms

FOR $j = 1$ **TO** n

IF (lecture j is compatible with some classroom)

 Schedule lecture j in any such classroom k .

ELSE

 Allocate a new classroom $d + 1$.

 Schedule lecture j in classroom $d + 1$.

$d \leftarrow d + 1$.

RETURN schedule.

Interval partitioning: earliest-start-time-first algorithm

Proposition. The earliest-start-time-first algorithm can be implemented in $O(n \log n)$ time.

Pf.

- Sorting by start times takes $O(n \log n)$ time.
- Store classrooms in a **priority queue** (key = finish time of its last lecture).
 - to allocate a new classroom, INSERT classroom onto priority queue.
 - to schedule lecture j in classroom k , INCREASE-KEY of classroom k to f_j .
 - to determine whether lecture j is compatible with any classroom, compare s_j to FIND-MIN
- Total # of priority queue operations is $O(n)$; each takes $O(\log n)$ time. ■

Remark. This implementation chooses a classroom k whose finish time of its last lecture is the **earliest**.

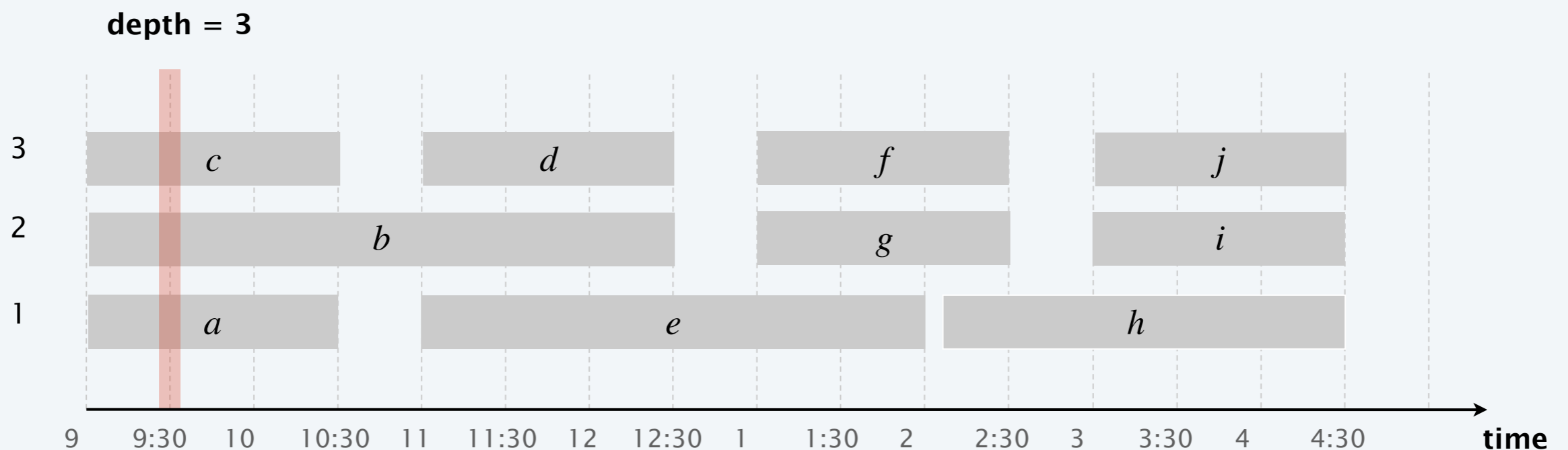
Interval partitioning: lower bound on optimal solution

Def. The **depth** of a set of open intervals is the maximum number of intervals that contain any given point.

Key observation. Number of classrooms needed \geq depth.

Q. Does minimum number of classrooms needed always equal depth?

A. Yes! Moreover, earliest-start-time-first algorithm finds a schedule whose number of classrooms equals the depth.



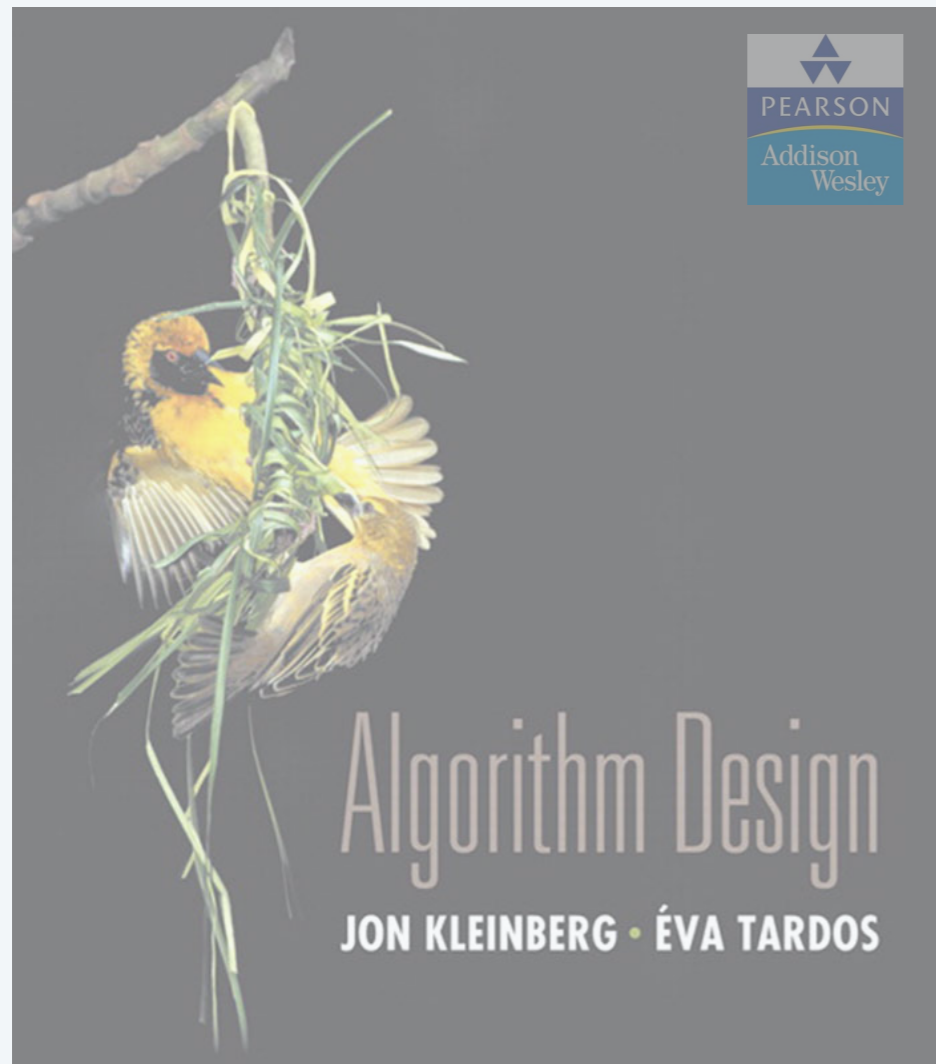
Interval partitioning: analysis of earliest-start-time-first algorithm

Observation. The earliest-start-time first algorithm never schedules two incompatible lectures in the same classroom.

Theorem. Earliest-start-time-first algorithm is optimal.

Pf.

- Let d = number of classrooms that the algorithm allocates.
- Classroom d is opened because we needed to schedule a lecture, say j , that is incompatible with a lecture in each of $d - 1$ other classrooms.
- Thus, these d lectures each end after s_j .
- Since we sorted by start time, each of these incompatible lectures start no later than s_j .
- Thus, we have d lectures overlapping at time $s_j + \epsilon$.
- Key observation \Rightarrow all schedules use $\geq d$ classrooms. ■



SECTION 4.2

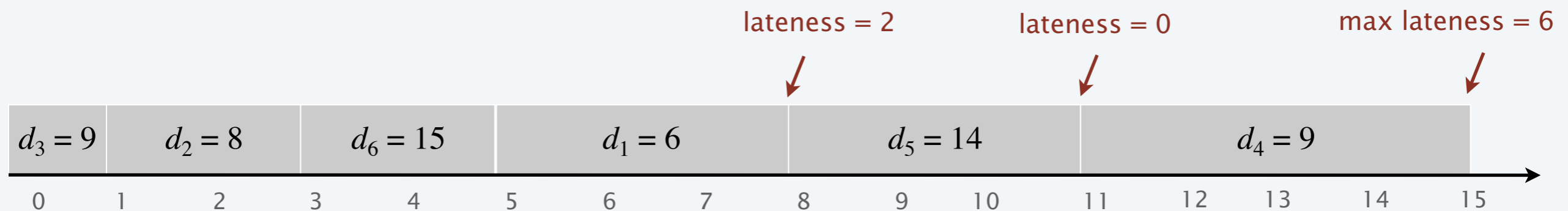
4. GREEDY ALGORITHMS I

- ▶ *coin changing*
- ▶ *interval scheduling*
- ▶ *interval partitioning*
- ▶ ***scheduling to minimize lateness***
- ▶ *optimal caching*

Scheduling to minimizing lateness

- 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 $L = \max_j \ell_j$.

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





Schedule jobs according to some natural order. Which order minimizes the maximum lateness?

- A.** [shortest processing time] Ascending order of processing time t_j .
- B.** [earliest deadline first] Ascending order of deadline d_j .
- C.** [smallest slack] Ascending order of slack: $d_j - t_j$.
- D.** None of the above.

Minimizing lateness: earliest deadline first

EARLIEST-DEADLINE-FIRST ($n, t_1, t_2, \dots, t_n, d_1, d_2, \dots, d_n$)

SORT jobs by due times and renumber so that $d_1 \leq d_2 \leq \dots \leq d_n$.

$t \leftarrow 0$.

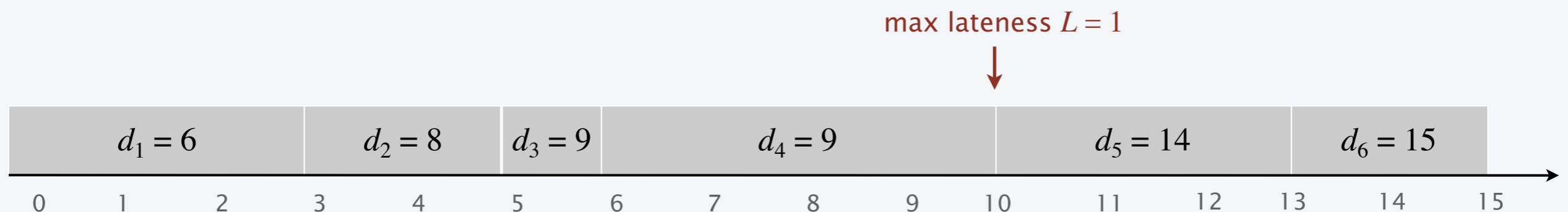
FOR $j = 1$ **TO** n

 Assign job j to interval $[t, t + t_j]$.

$s_j \leftarrow t$; $f_j \leftarrow t + t_j$.

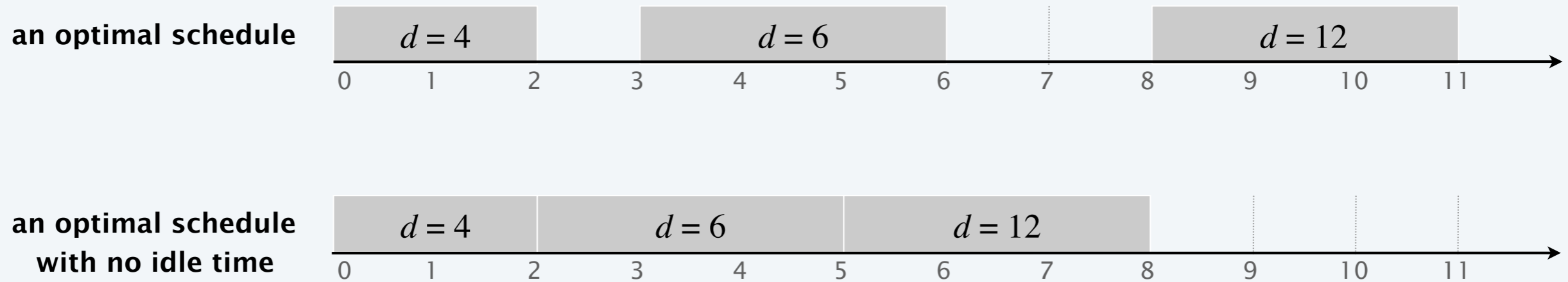
$t \leftarrow t + t_j$.

RETURN intervals $[s_1, f_1], [s_2, f_2], \dots, [s_n, f_n]$.



Minimizing lateness: no idle time

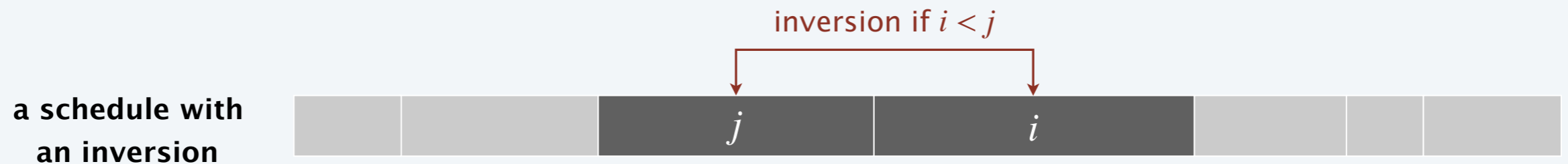
Observation 1. There exists an optimal schedule with no **idle time**.



Observation 2. The earliest-deadline-first 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 is scheduled before i .



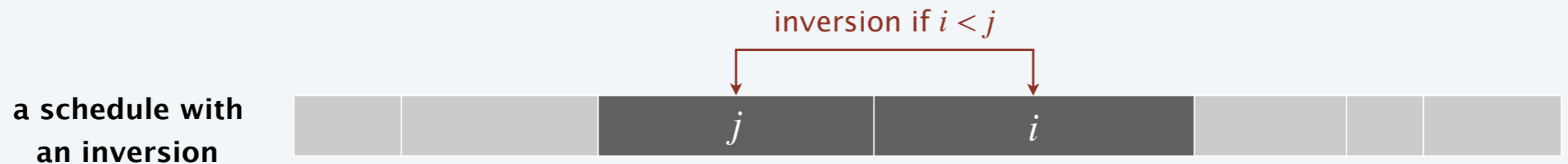
recall: we assume the jobs are numbered so that $d_1 \leq d_2 \leq \dots \leq d_n$

Observation 3. The earliest-deadline-first schedule is the unique idle-free schedule with no inversions.



Minimizing lateness: inversions

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

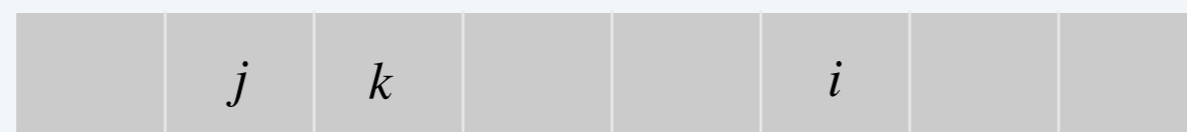


recall: we assume the jobs are numbered so that $d_1 \leq d_2 \leq \dots \leq d_n$

Observation 4. If an idle-free schedule has an inversion, then it has an adjacent inversion.

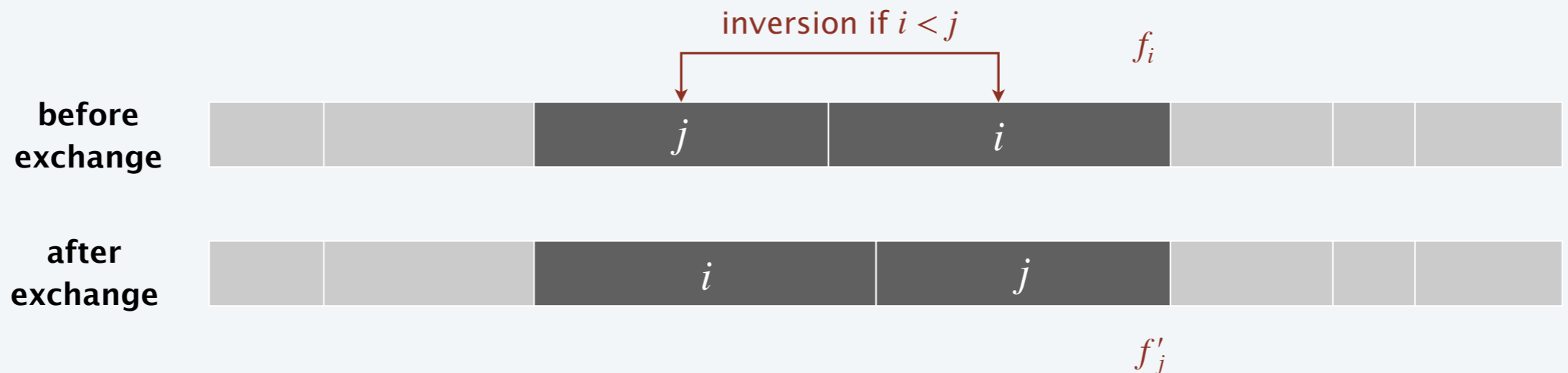
Pf. ← two inverted jobs scheduled consecutively

- Let $i-j$ be a closest inversion.
- Let k be element immediately to the right of j .
- Case 1. [$j > k$] Then $j-k$ is an adjacent inversion.
- Case 2. [$j < k$] Then $i-k$ is a closer inversion since $i < j < k$. ✖



Minimizing lateness: inversions

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



Key claim. Exchanging two adjacent, inverted jobs i and j reduces the number of inversions by 1 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, $\ell'_j = f'_j - d_j \leftarrow \text{definition}$
 $= f_i - d_j \leftarrow j \text{ now finishes at time } f_i$
 $\leq f_i - d_i \leftarrow i < j \Rightarrow d_i \leq d_j$
 $\leq \ell_i \leftarrow \text{definition}$


Minimizing lateness: analysis of earliest-deadline-first algorithm

Theorem. The earliest-deadline-first schedule S is optimal.

Pf. [by contradiction]

Define S^* to be an optimal schedule with the fewest inversions.

optimal schedule can
have inversions



- Can assume S^* has no idle time. ← Observation 1
- Case 1. [S^* has no inversions] Then $S = S^*$. ← Observation 3
- Case 2. [S^* has an inversion]
 - let $i-j$ be an adjacent inversion ← Observation 4
 - exchanging jobs i and j decreases the number of inversions by 1 without increasing the max lateness ← key claim
 - contradicts “fewest inversions” part of the definition of S^* ※

Greedy analysis strategies

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.

Structural. Discover a simple “structural” bound asserting that every possible solution must have a certain value. Then show that your algorithm always achieves this bound.

Exchange argument. Gradually transform any solution to the one found by the greedy algorithm without hurting its quality.

Other greedy algorithms. Gale–Shapley, Kruskal, Prim, Dijkstra, Huffman, ...

