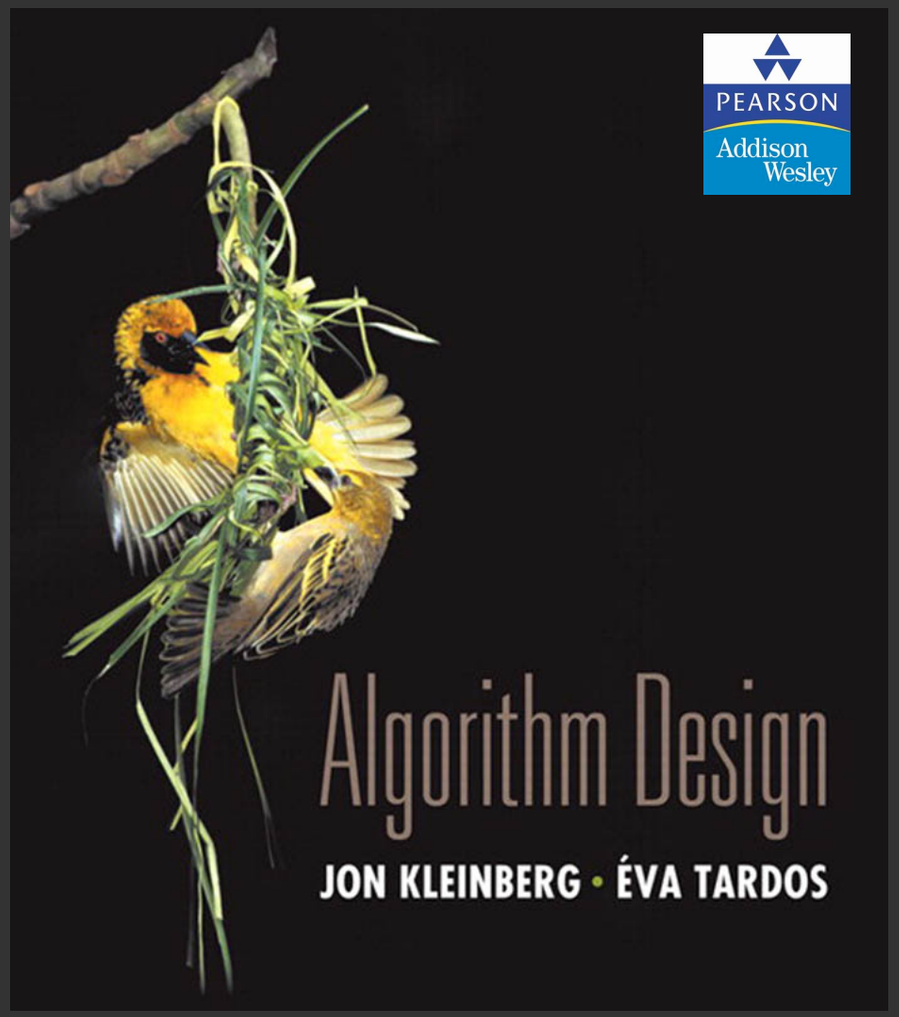


4. Greedy Algorithms I

- *coin changing*
- *interval scheduling*
- *interval partitioning*
- *scheduling to minimize lateness*

Lecture slides by Kevin Wayne

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

4. GREEDY ALGORITHMS I

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

Coin changing

Goal. Given U. S. currency denominations { 1, 5, 10, 25, 100 }, devise a method to pay amount to customer using fewest coins.

Ex. 34¢.

Cashier's algorithm. At each iteration, add coin of the largest value that does not take us past the amount to be paid.

Ex. $2.89.

Cashier's algorithm

At each iteration, add coin of the largest value that does not take us past the amount to be paid.

CASHIERS-ALGORITHM $(x, c_1, c_2, \ldots, c_n)$

SORT n coin denominations so that $0 < c_1 < c_2 < \ldots < c_n$.

$S \leftarrow \emptyset$. ← multiset of coins selected

WHILE $(x > 0)$

  $k \leftarrow$ largest coin denomination c_k such that $c_k \leq x$.

  IF (no such k)

    RETURN *"no solution."*

  ELSE

    $x \leftarrow x - c_k$.

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

RETURN S.

Greedy algorithms I: quiz 1

Is the cashier's algorithm optimal?

A. Yes, greedy algorithms are always optimal.

B. Yes, for any set of coin denominations $c_1 < c_2 < \ldots < c_n$ provided $c_1 = 1$.

C. Yes, because of special properties of U.S. coin denominations.

D. No.

Cashier's algorithm (for arbitrary coin denominations)

Q. Is cashier's algorithm optimal for any set of denominations?

A. No. Consider U.S. postage: 1, 10, 21, 34, 70, 100, 350, 1225, 1500.

- Cashier's algorithm: 140¢ = 100 + 34 + 1 + 1 + 1 + 1 + 1 + 1.
- Optimal: 140¢ = 70 + 70.

A. No. It may not even lead to a feasible solution if $c_1 > 1$: 7, 8, 9.

- Cashier's algorithm: 15¢ = 9 + ?.
- Optimal: 15¢ = 7 + 8.

Properties of any optimal solution (for U.S. coin denominations)

Property. Number of pennies ≤ 4.

Pf. Replace 5 pennies with 1 nickel.

Property. Number of nickels ≤ 1.

Property. Number of quarters ≤ 3.

Property. Number of nickels + number of dimes ≤ 2.

Pf.

- Recall: ≤ 1nickel.
- Replace 3 dimes and 0 nickels with 1 quarter and 1 nickel;
- Replace 2 dimes and 1 nickel with 1 quarter.

dollars (100¢)

quarters (25¢)

dimes (10¢)

nickels (5¢)

pennies (1¢)

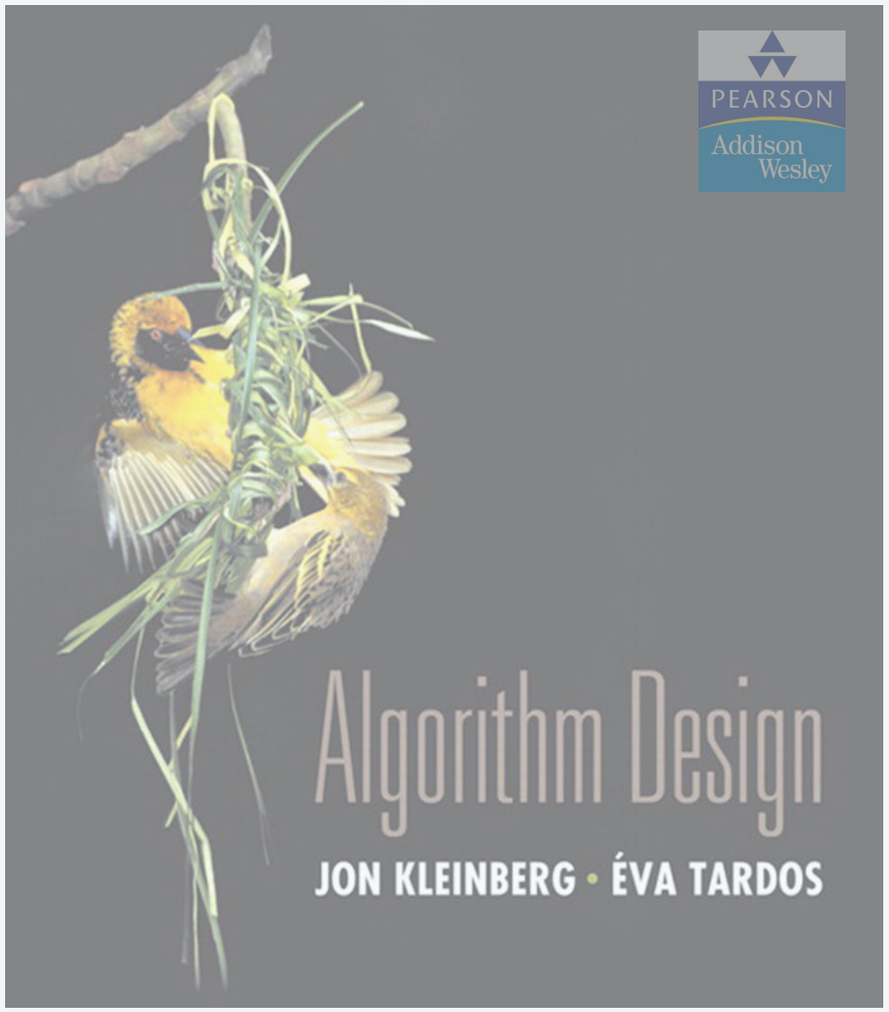
Optimality of cashier's algorithm (for U.S. coin denominations)

Theorem. Cashier's algorithm is optimal for U.S. coins { 1, 5, 10, 25, 100 }.

Pf. [by induction on amount to be paid x]

- Consider optimal way to change $c_k \le x < c_{k+1}$: greedy takes coin k.
- We claim that any optimal solution must take coin k.
  - if not, it needs enough coins of type $c_1, \ldots, c_{k-1}$ to add up to x
  - table below indicates no optimal solution can do this
- Problem reduces to coin-changing $x - c_k$ cents, which, by induction, is optimally solved by cashier's algorithm. ▪

k	c_k	all optimal solutions must satisfy	max value of coin denominations $c_1, c_2, \ldots, c_{k-1}$ in any optimal solution
1	1	$P \le 4$	–
2	5	$N \le 1$	4
3	10	$N + D \le 2$	4 + 5 = 9
4	25	$Q \le 3$	20 + 4 = 24
5	100	*no limit*	75 + 24 = 99



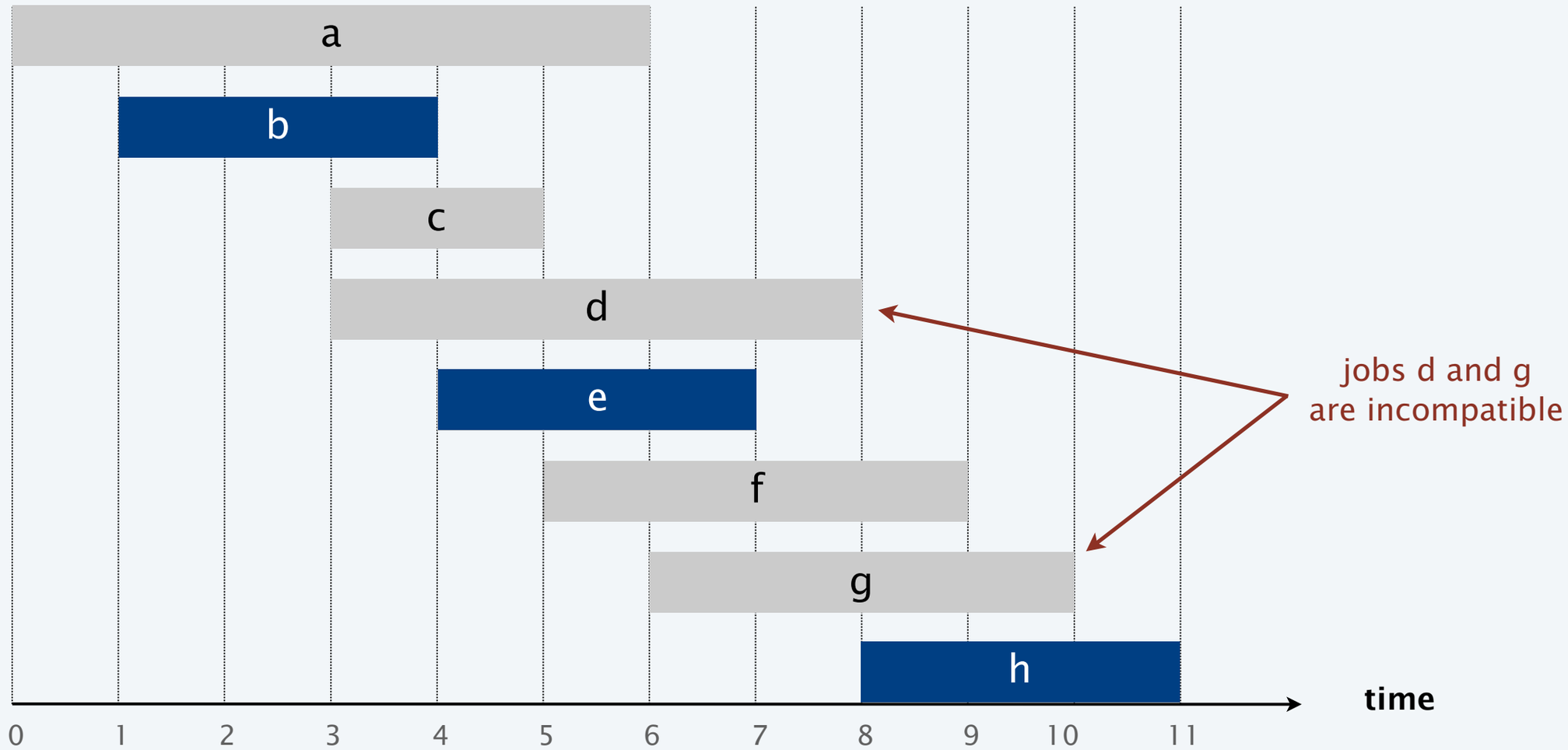
SECTION 4.1

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.



Greedy algorithms I: quiz 2

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 $(n, s_1, s_2, \ldots, s_n, f_1, f_2, \ldots, f_n)$

SORT jobs by finish times and renumber so that $f_1 \le f_2 \le \ldots \le f_n$.

$S \leftarrow \emptyset$. ⟵ 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.

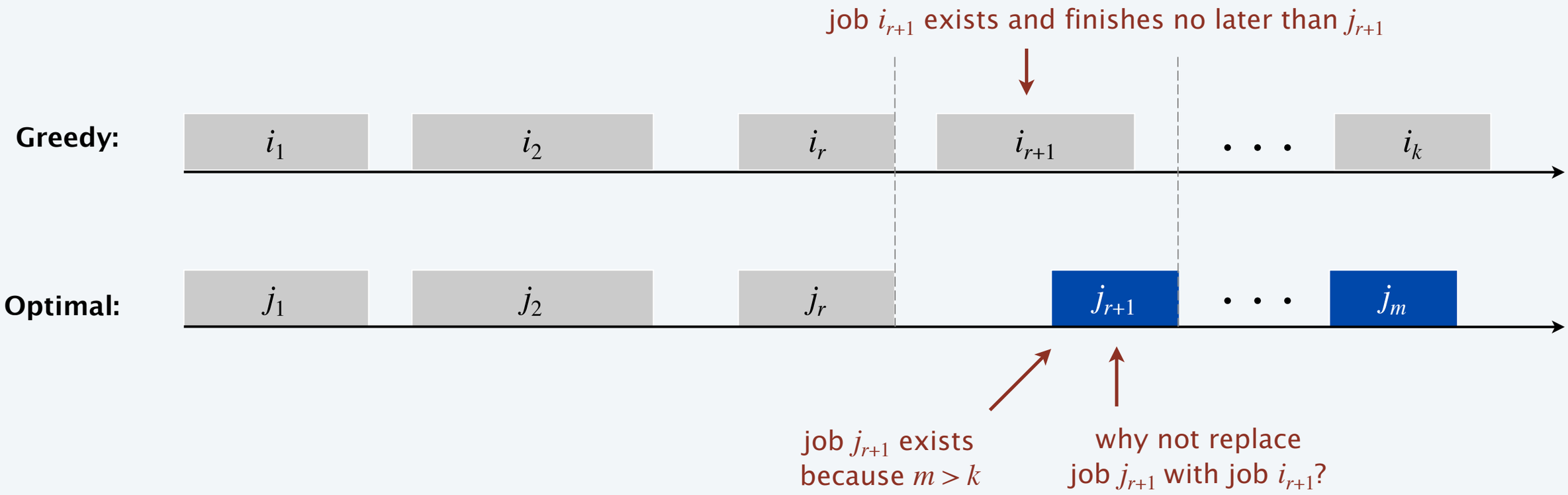
- Keep track of job j^* that was added last to S.
- Job j is compatible with S iff $s_j \ge f_{j^*}$.
- Sorting by finish times takes $O(n \log n)$ time.

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 $i_1, i_2, \ldots i_k$ denote set of jobs selected by greedy.
- Let $j_1, j_2, \ldots j_m$ denote set of jobs in an optimal solution with $i_1 = j_1, i_2 = j_2, \ldots, i_r = j_r$ for the largest possible value of r.

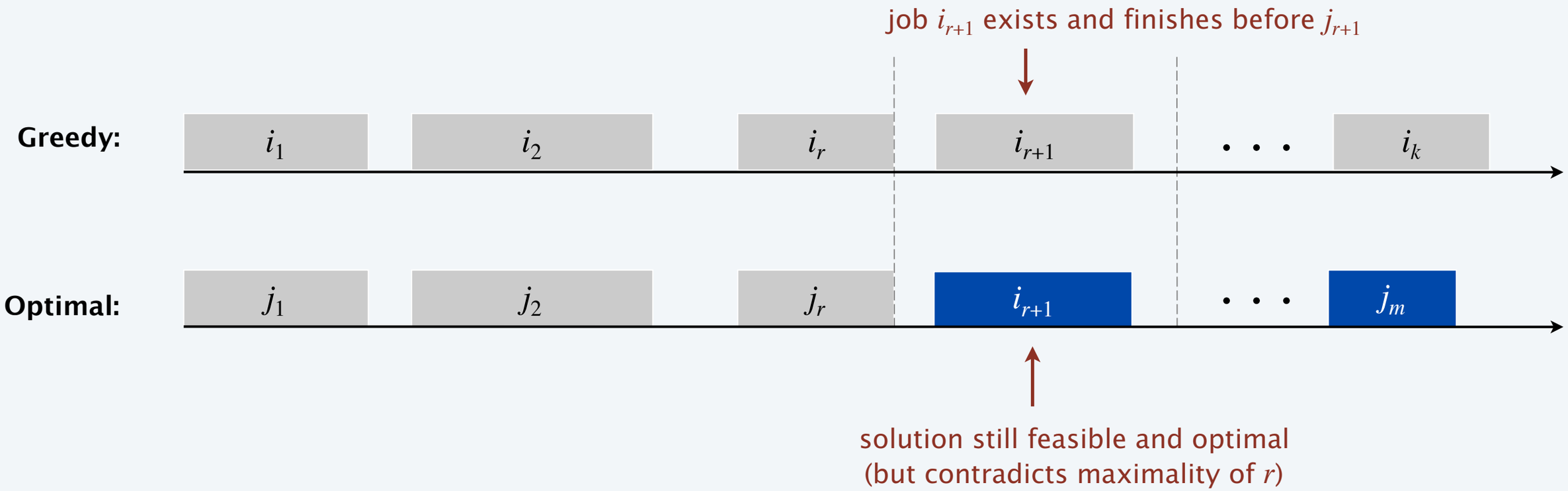


Interval scheduling: analysis of earliest-finish-time-first algorithm

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

Pf. [by contradiction]

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



Greedy algorithms I: quiz 3

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?

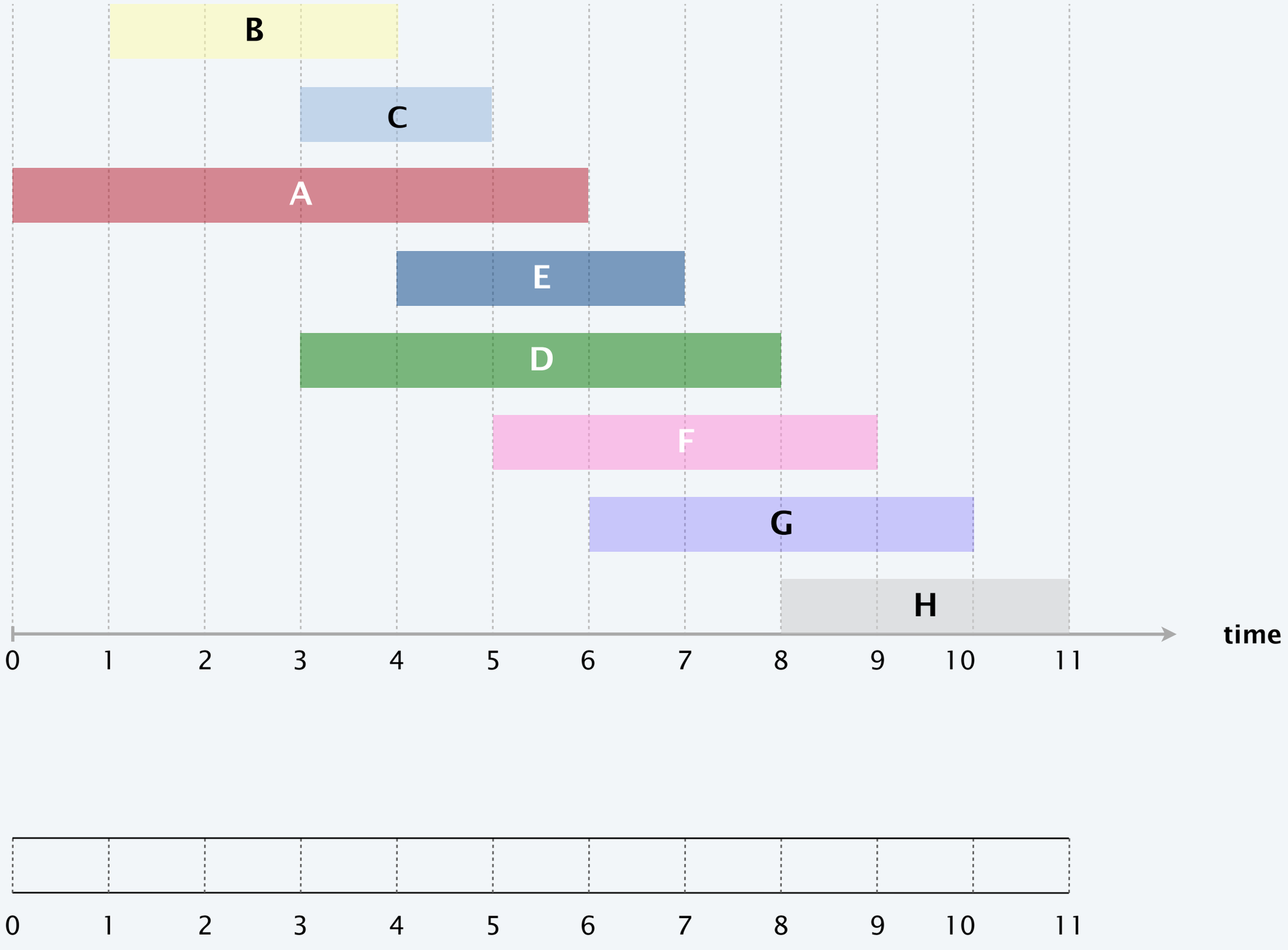
A. Yes, because greedy algorithms are always optimal.

B. Yes, because the same proof of correctness is valid.

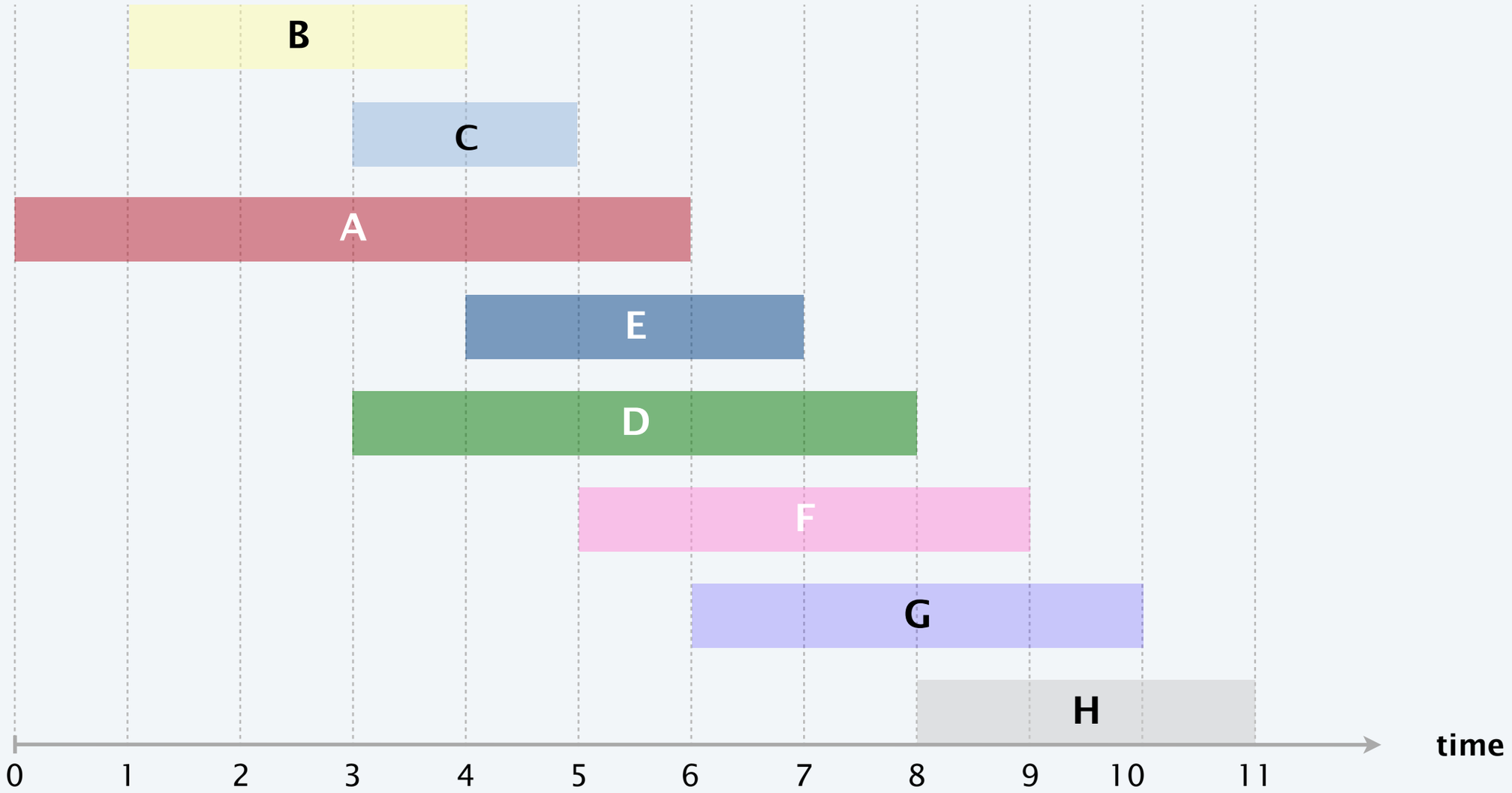
C. No, because the same proof of correctness is no longer valid.

D. No, because you could assign a huge weight to a job that overlaps the job with the earliest finish time.

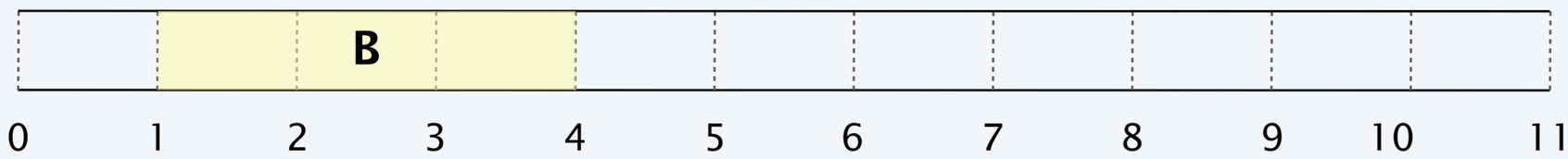
Earliest-finish-time-first algorithm demo



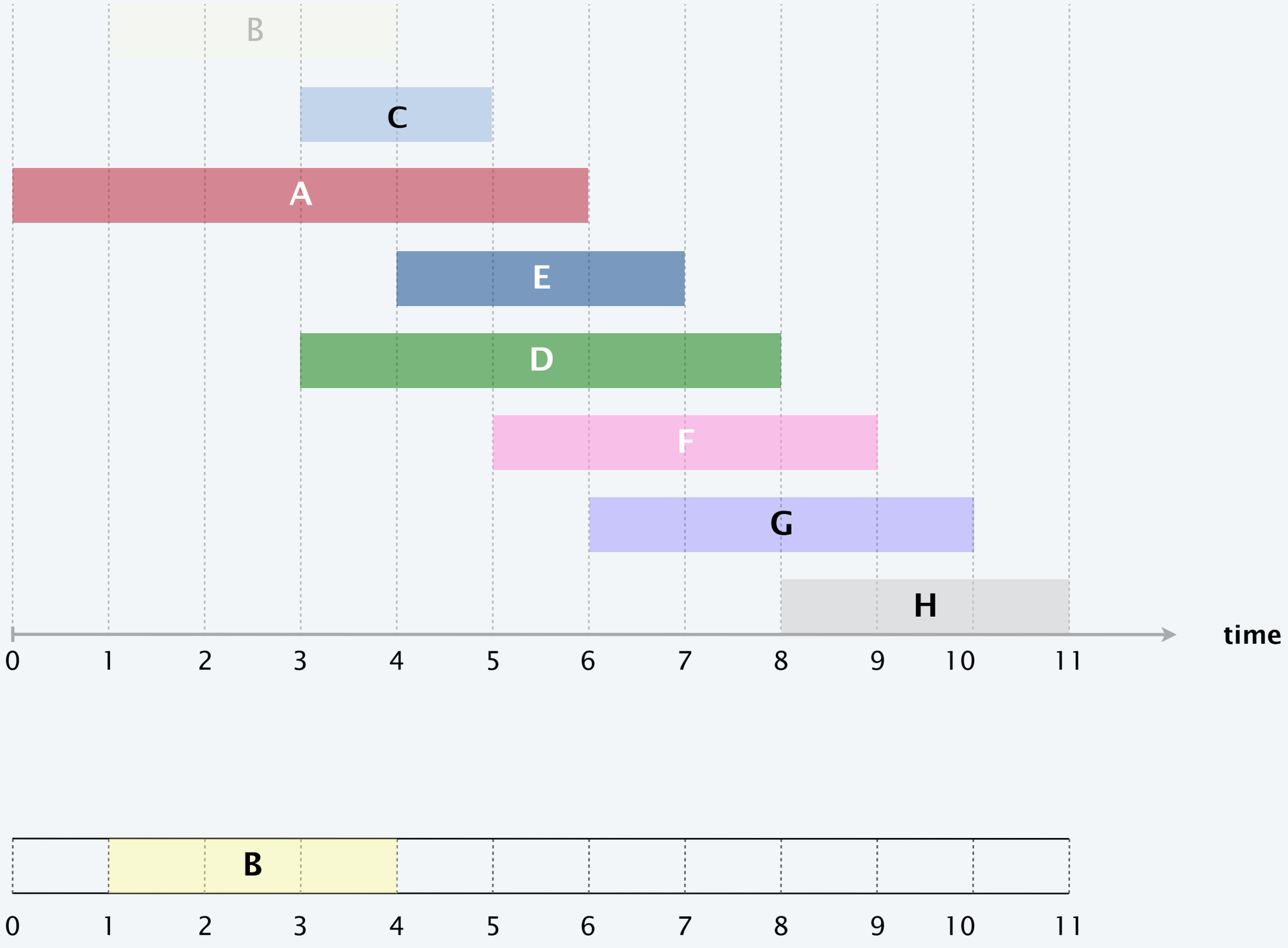
Earliest-finish-time-first algorithm demo



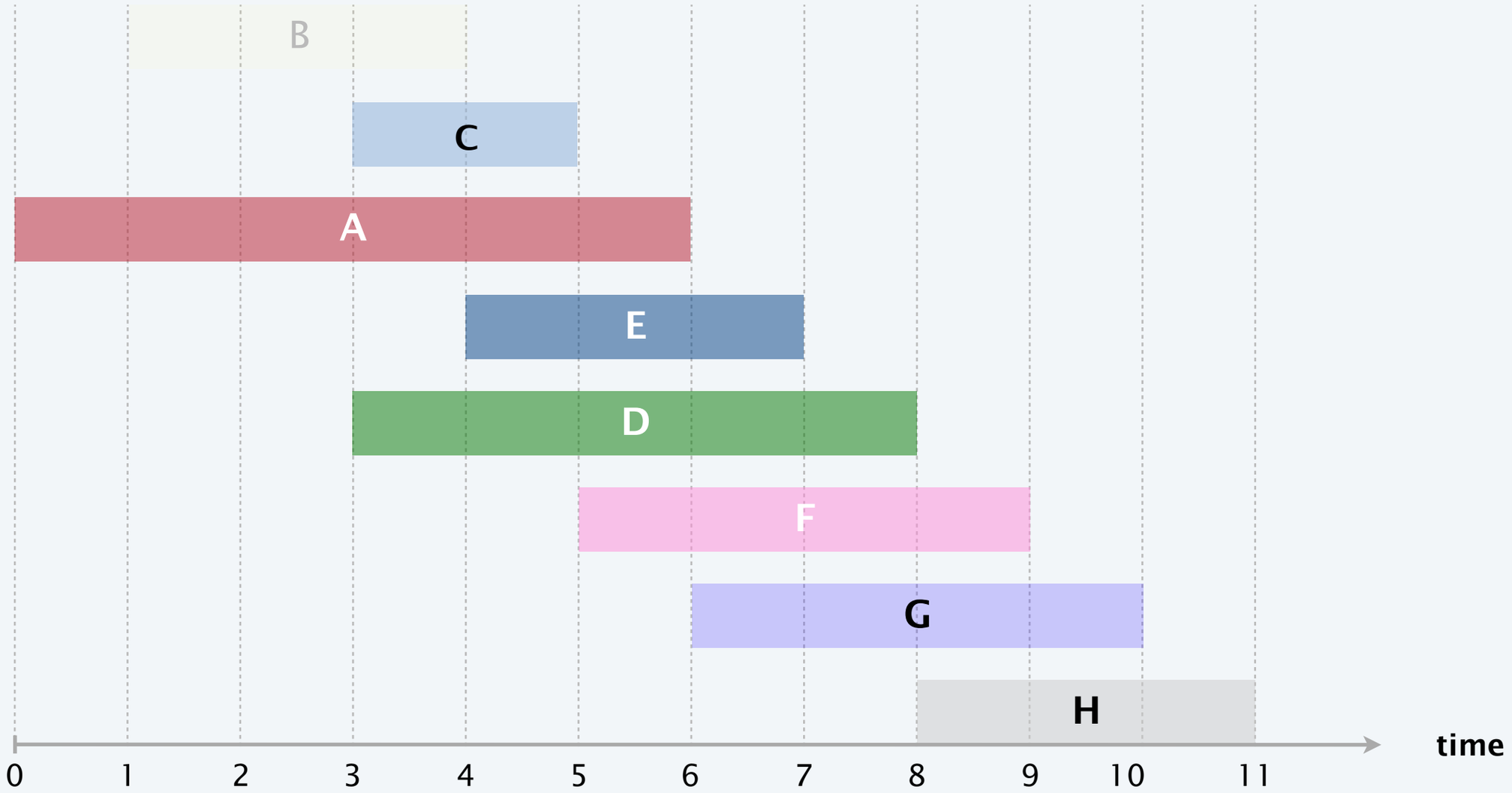
job B is compatible (add to schedule)



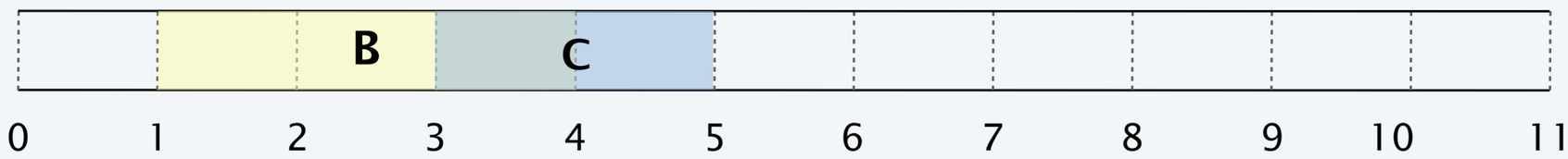
Earliest-finish-time-first algorithm demo



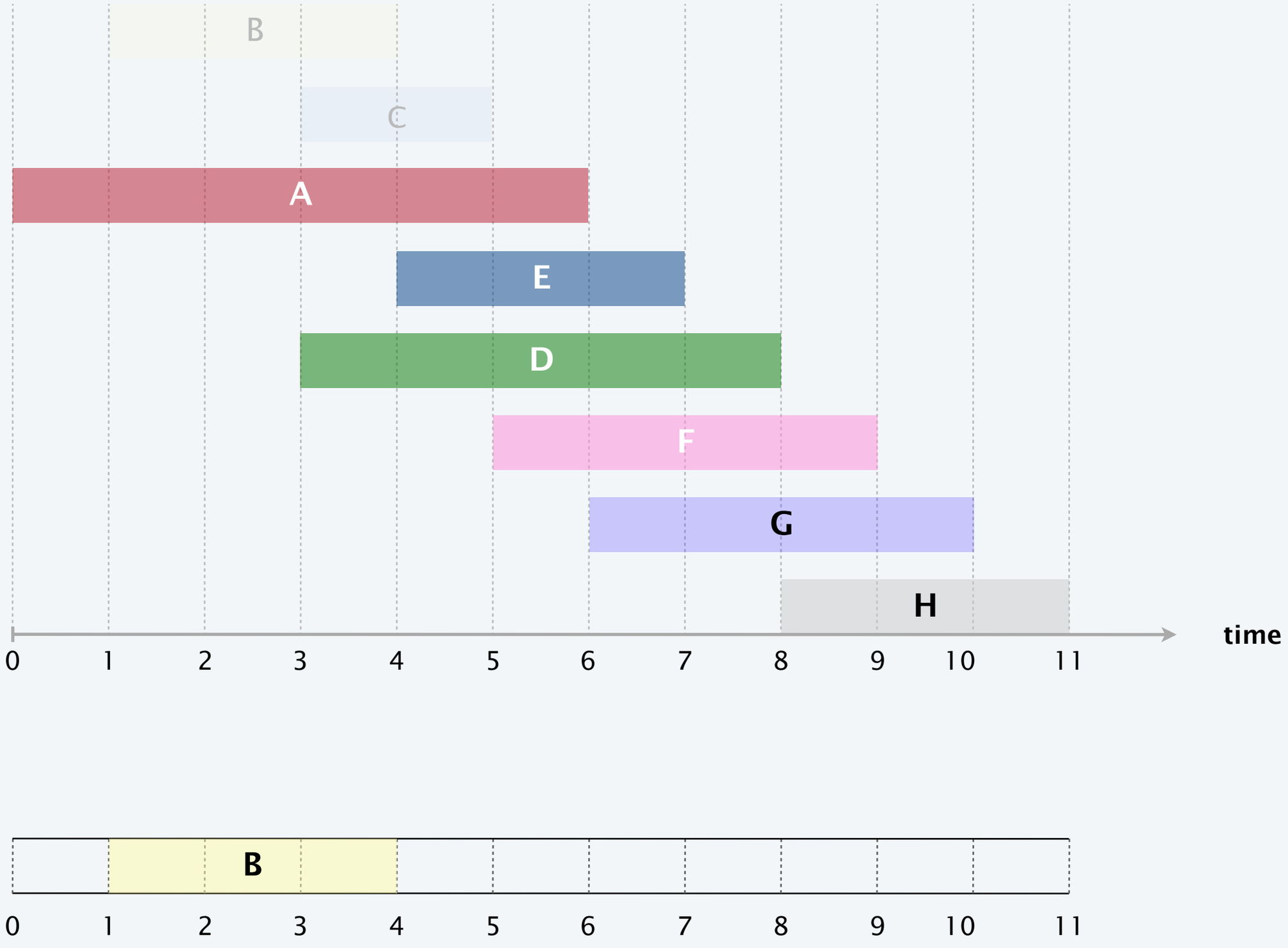
Earliest-finish-time-first algorithm demo



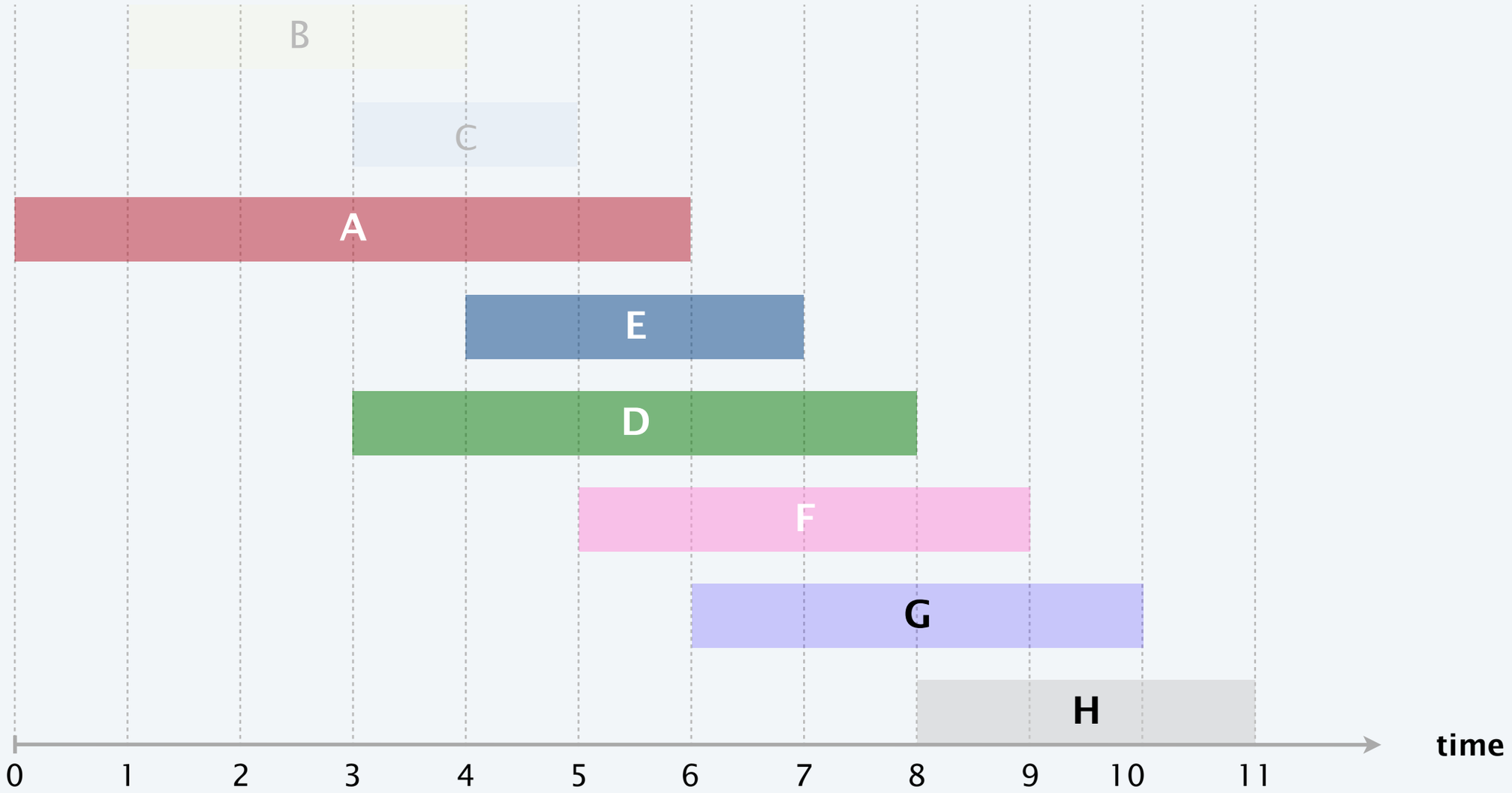
job C is incompatible (do not add to schedule)



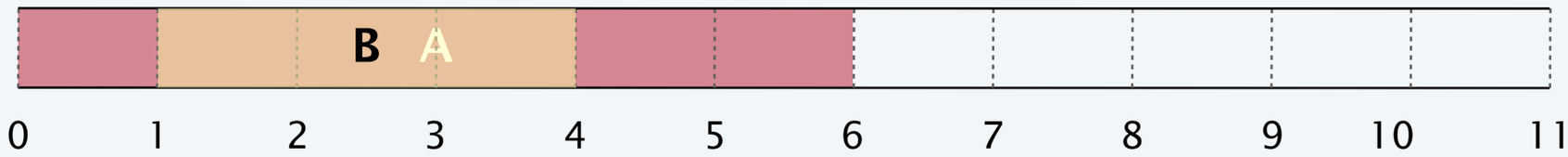
Earliest-finish-time-first algorithm demo



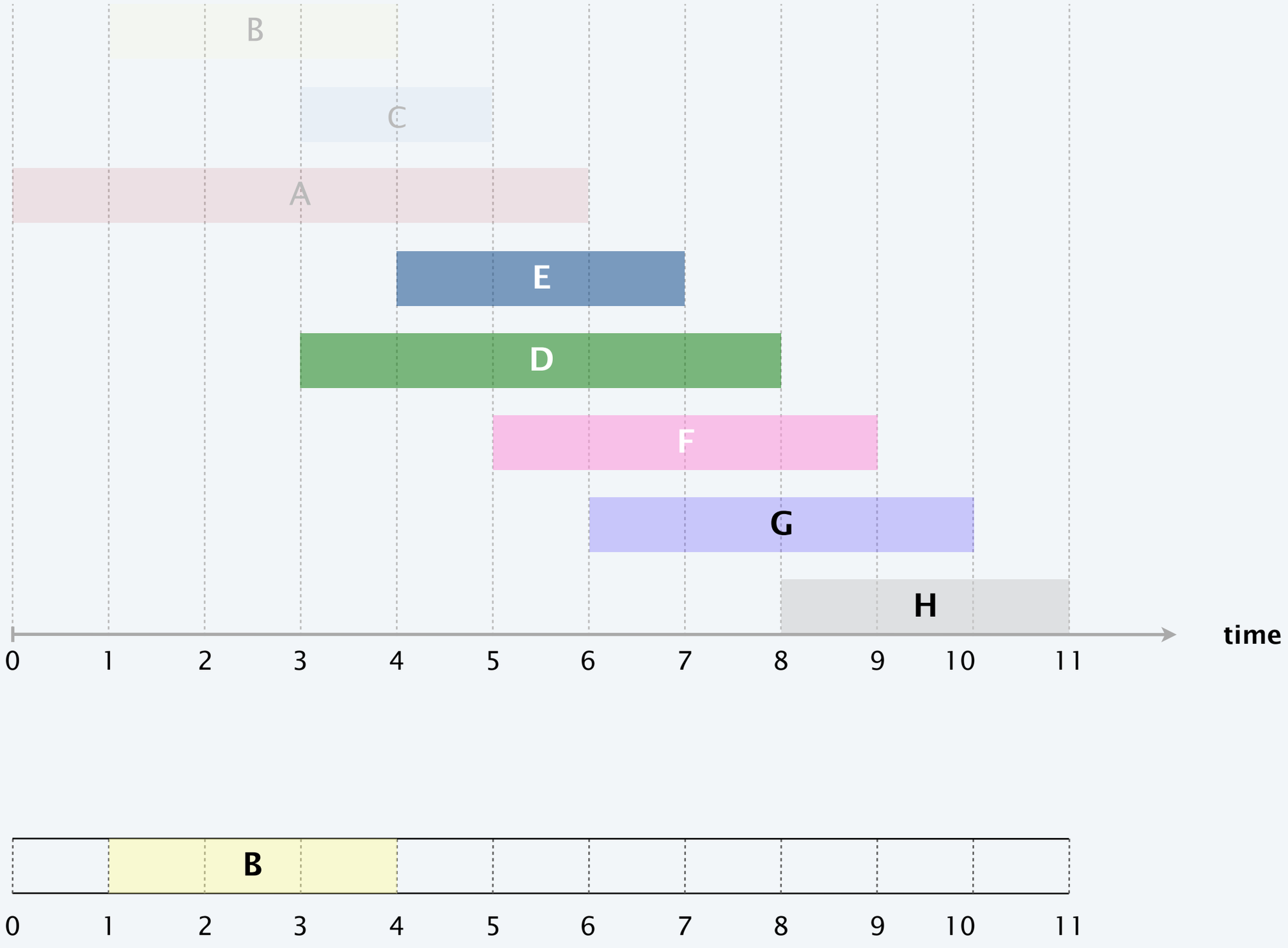
Earliest-finish-time-first algorithm demo



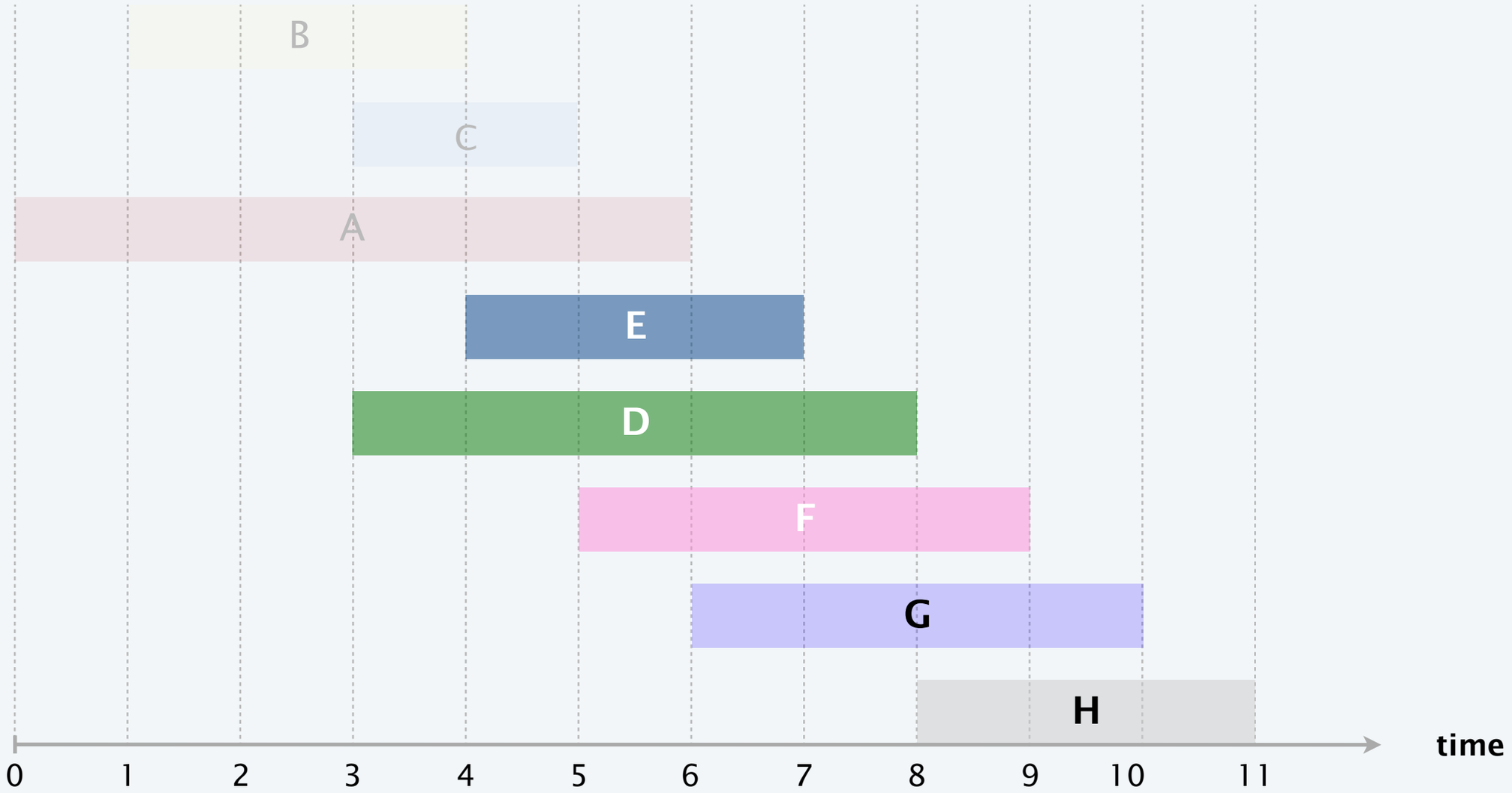
job A is incompatible (do not add to schedule)



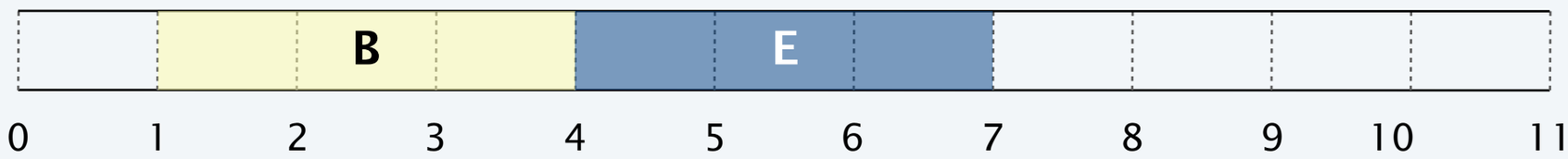
Earliest-finish-time-first algorithm demo



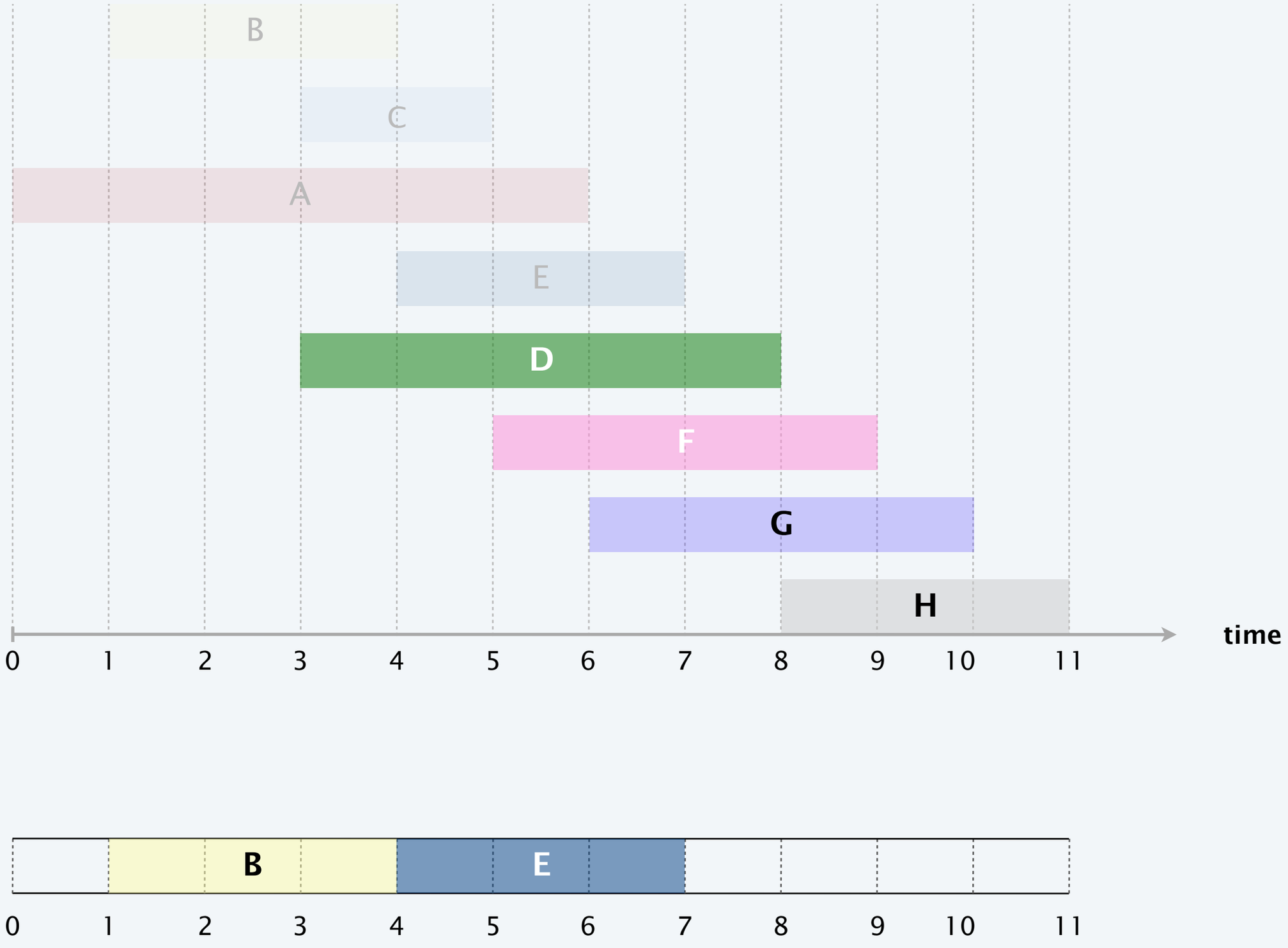
Earliest-finish-time-first algorithm demo



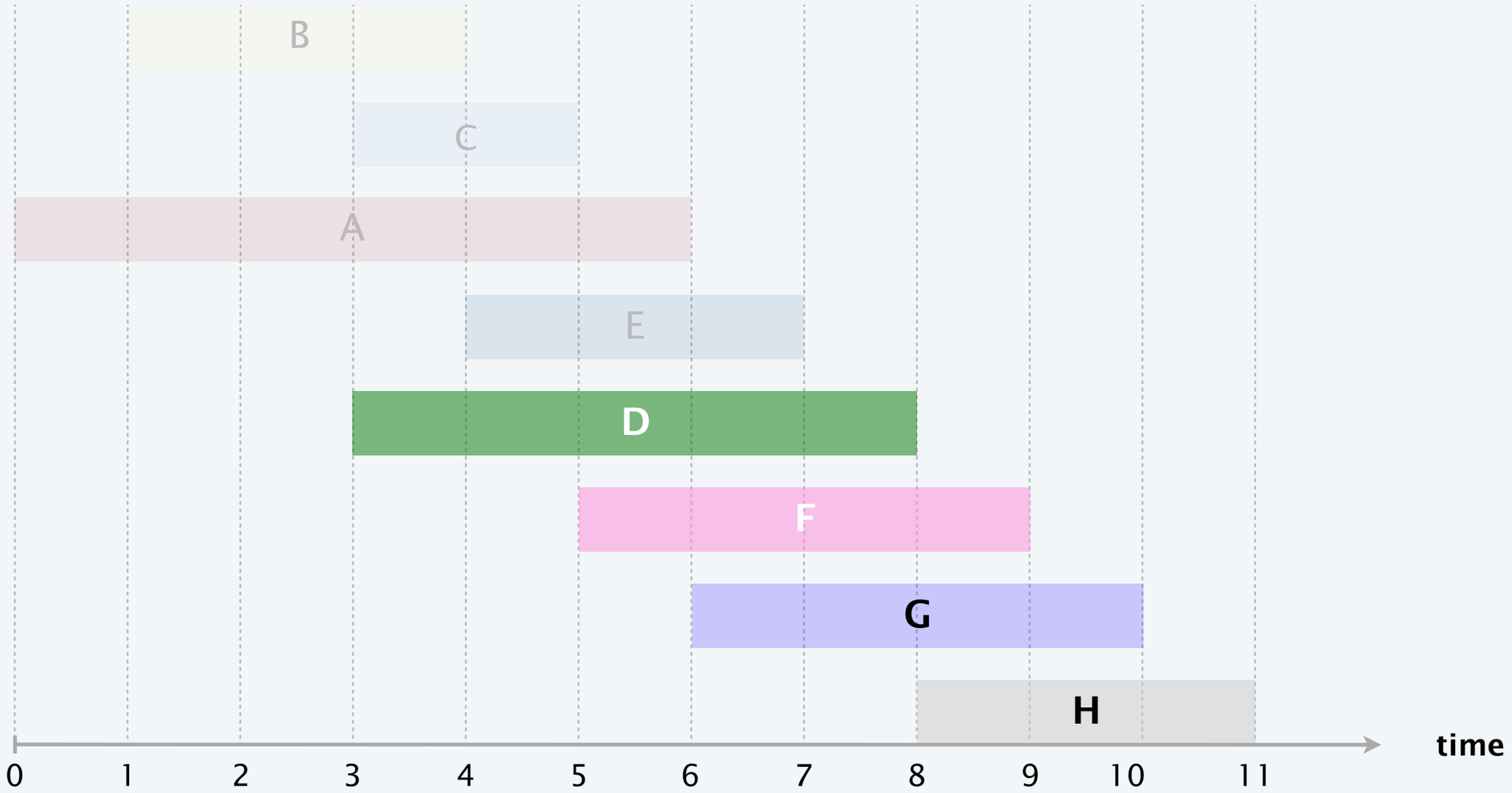
job E is compatible (add to schedule)



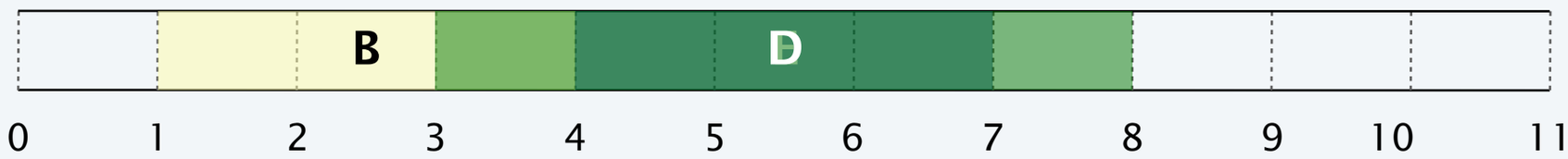
Earliest-finish-time-first algorithm demo



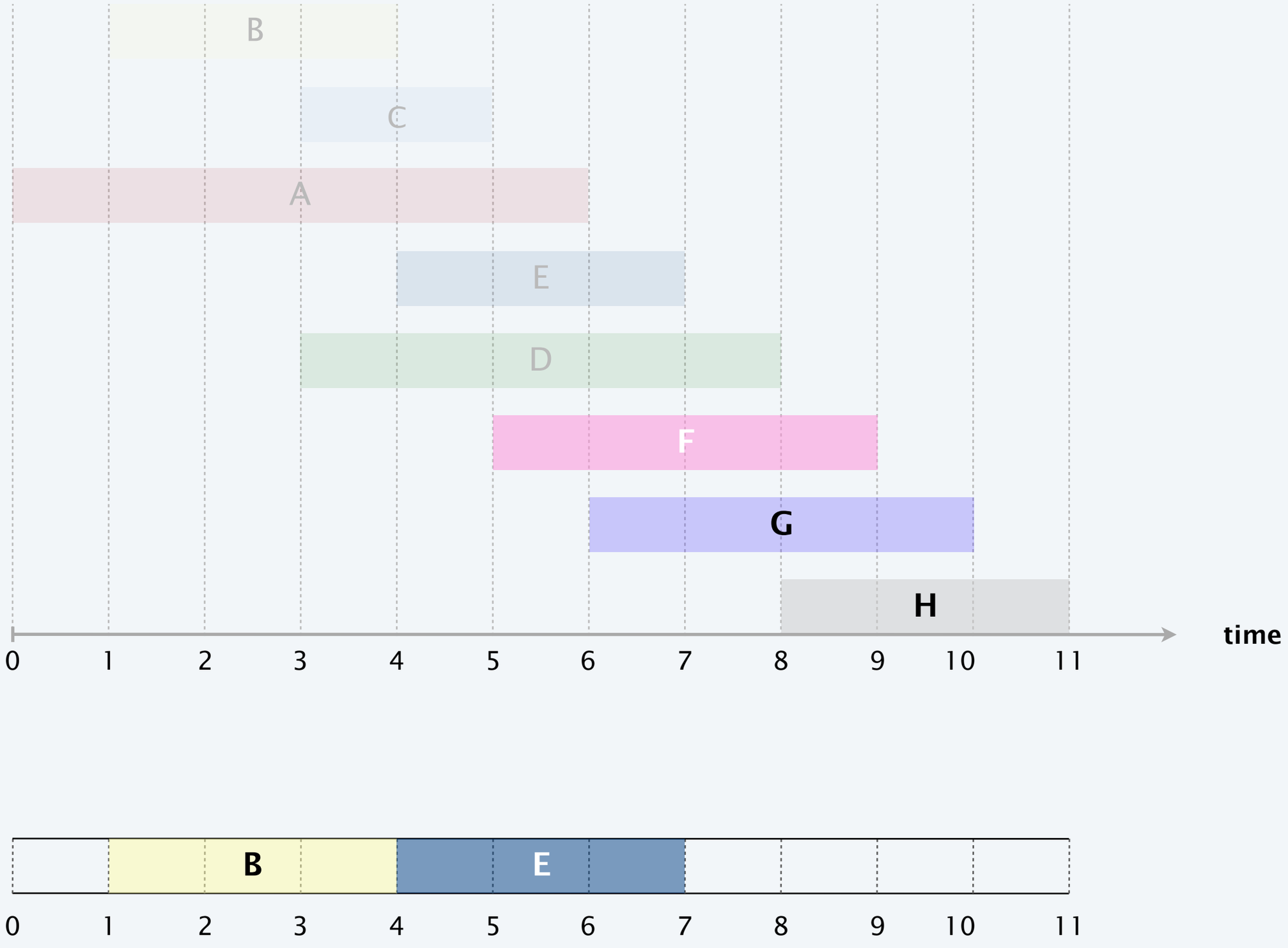
Earliest-finish-time-first algorithm demo



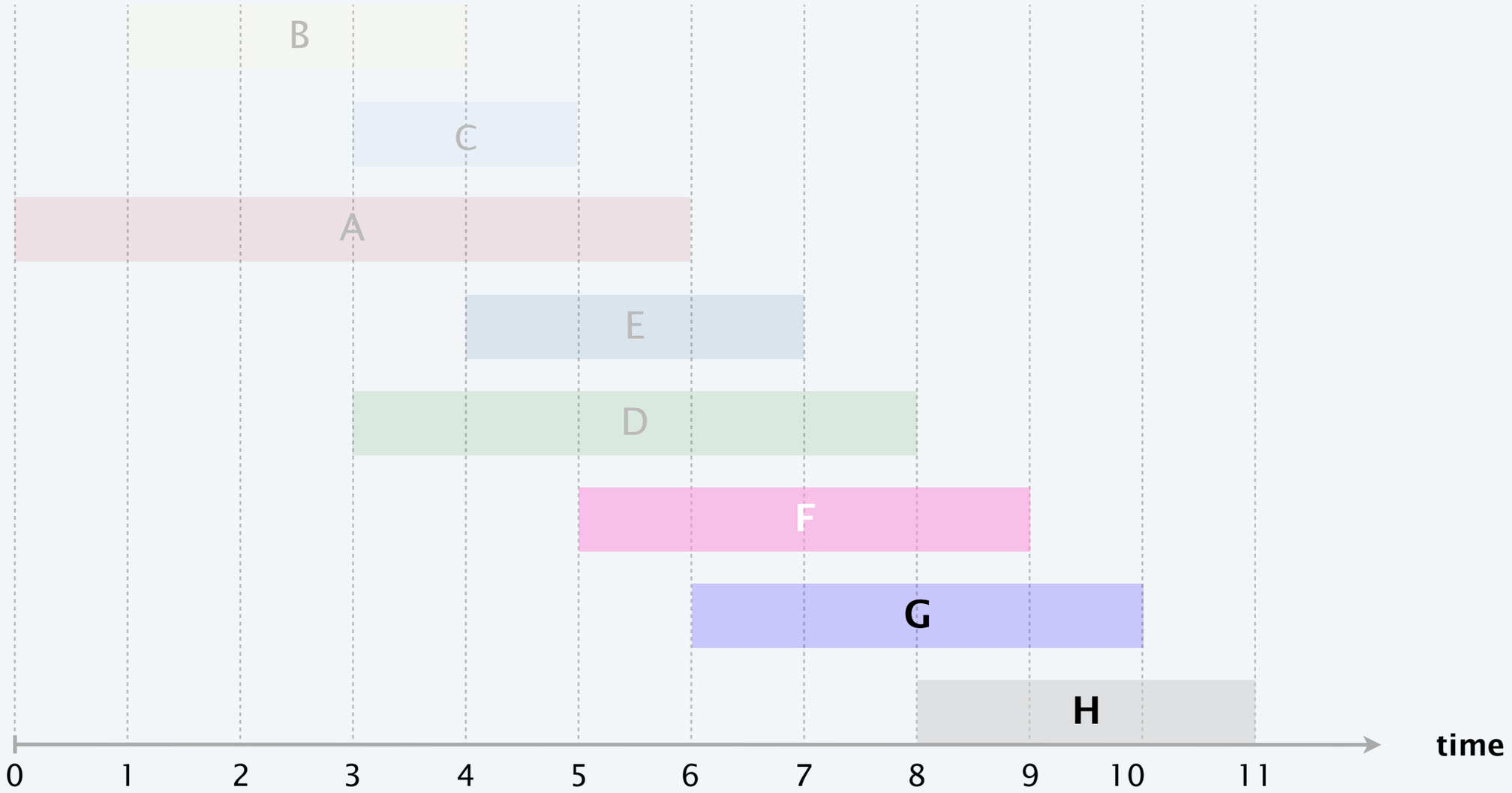
job D is incompatible (do not add to schedule)



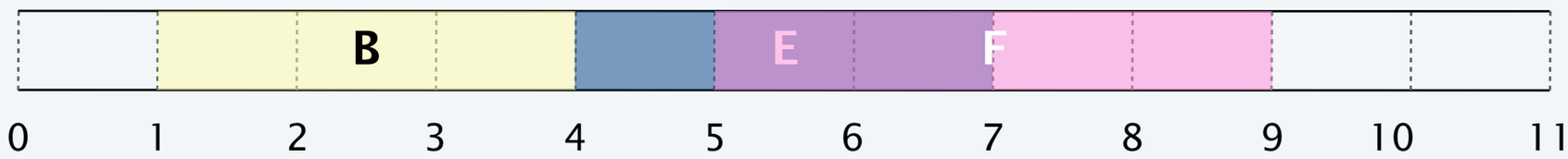
Earliest-finish-time-first algorithm demo



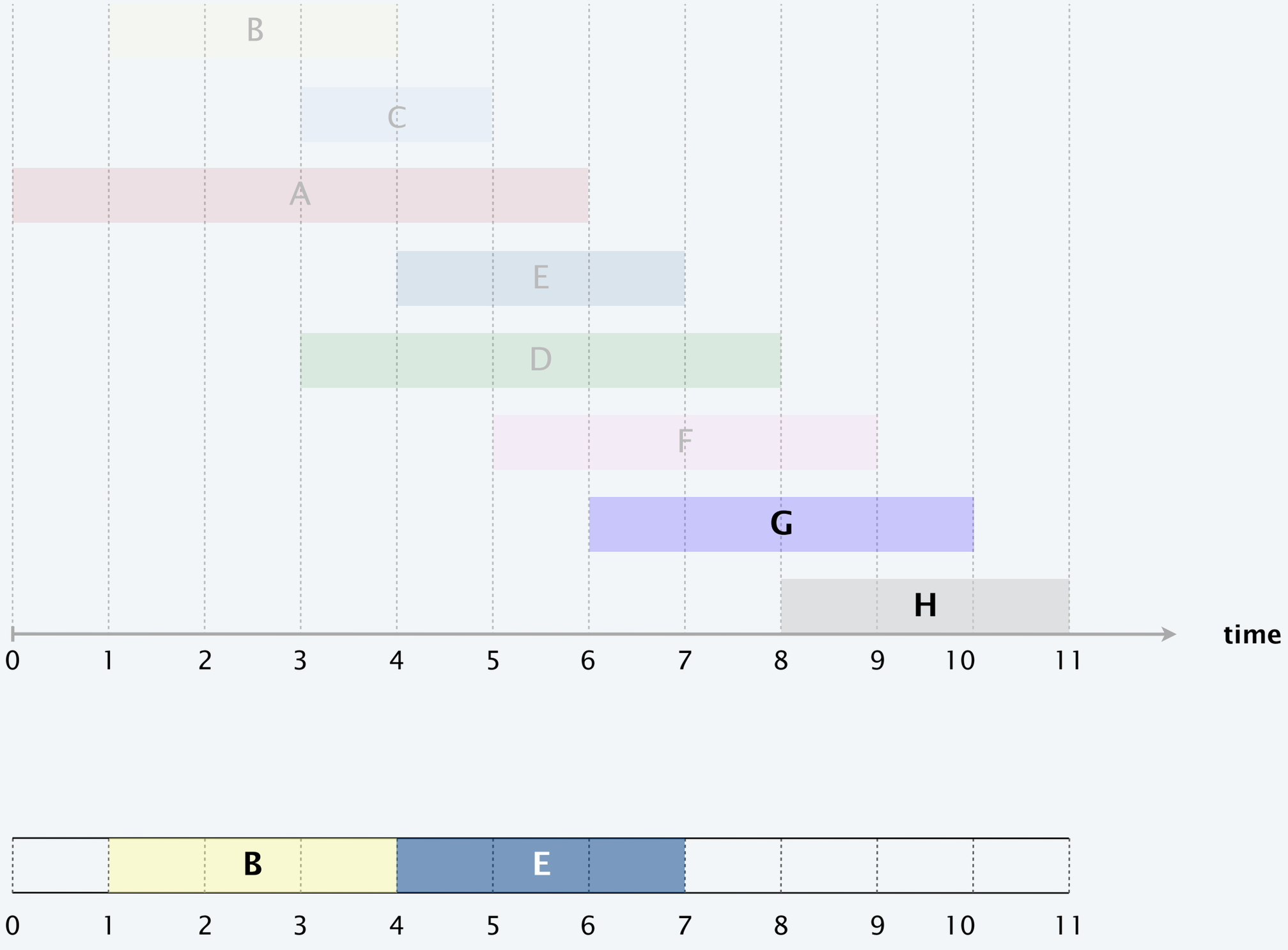
Earliest-finish-time-first algorithm demo



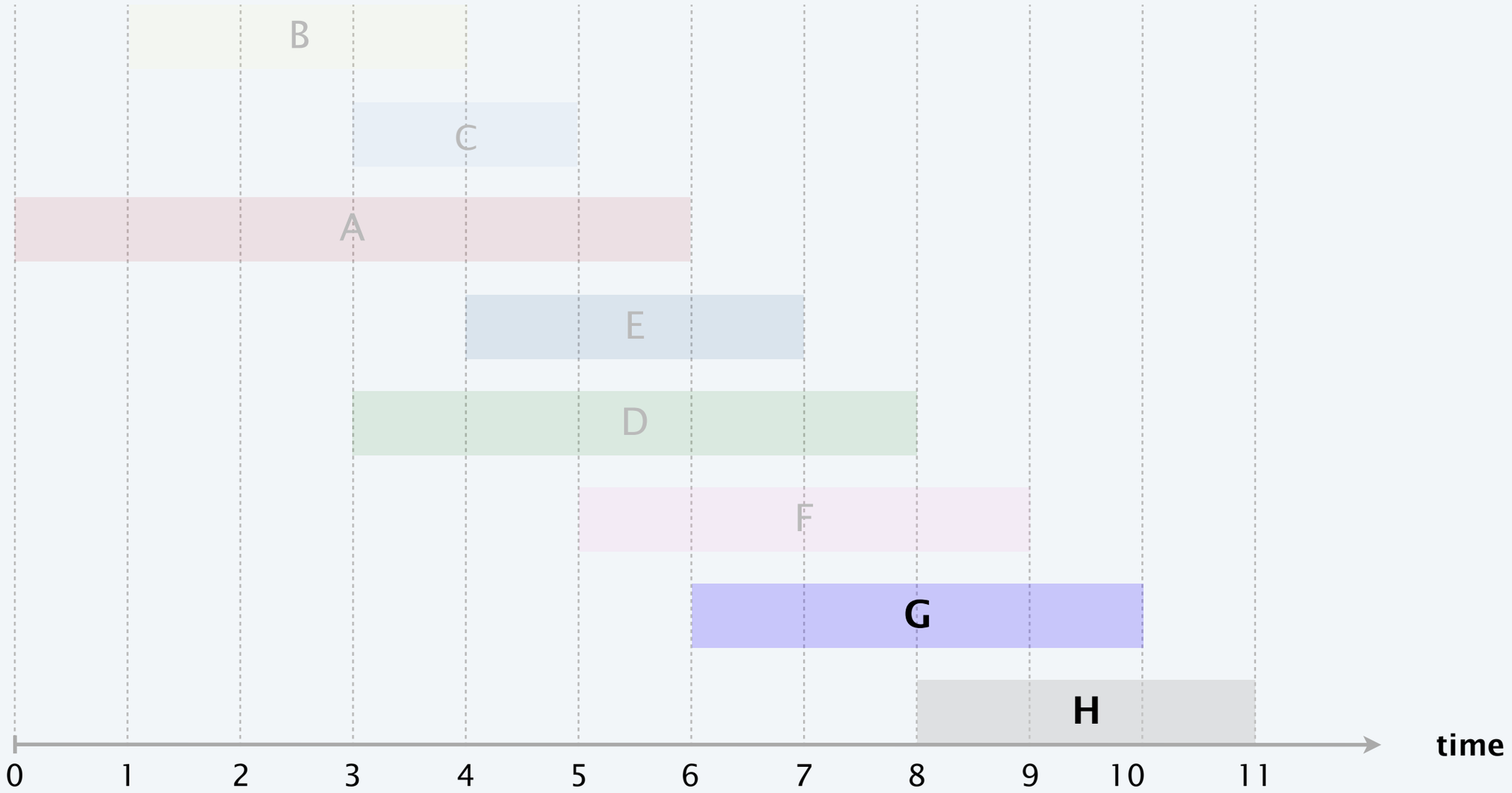
job F is incompatible (do not add to schedule)



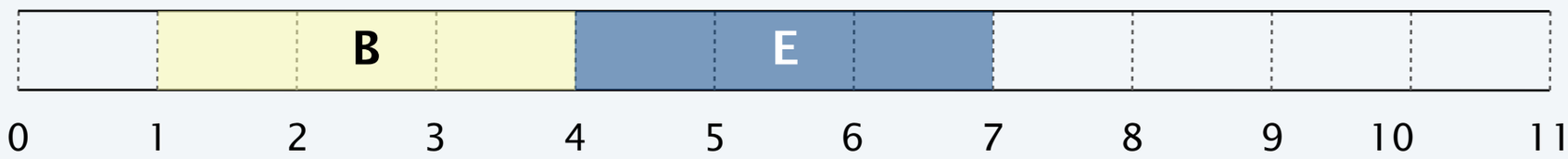
Earliest-finish-time-first algorithm demo



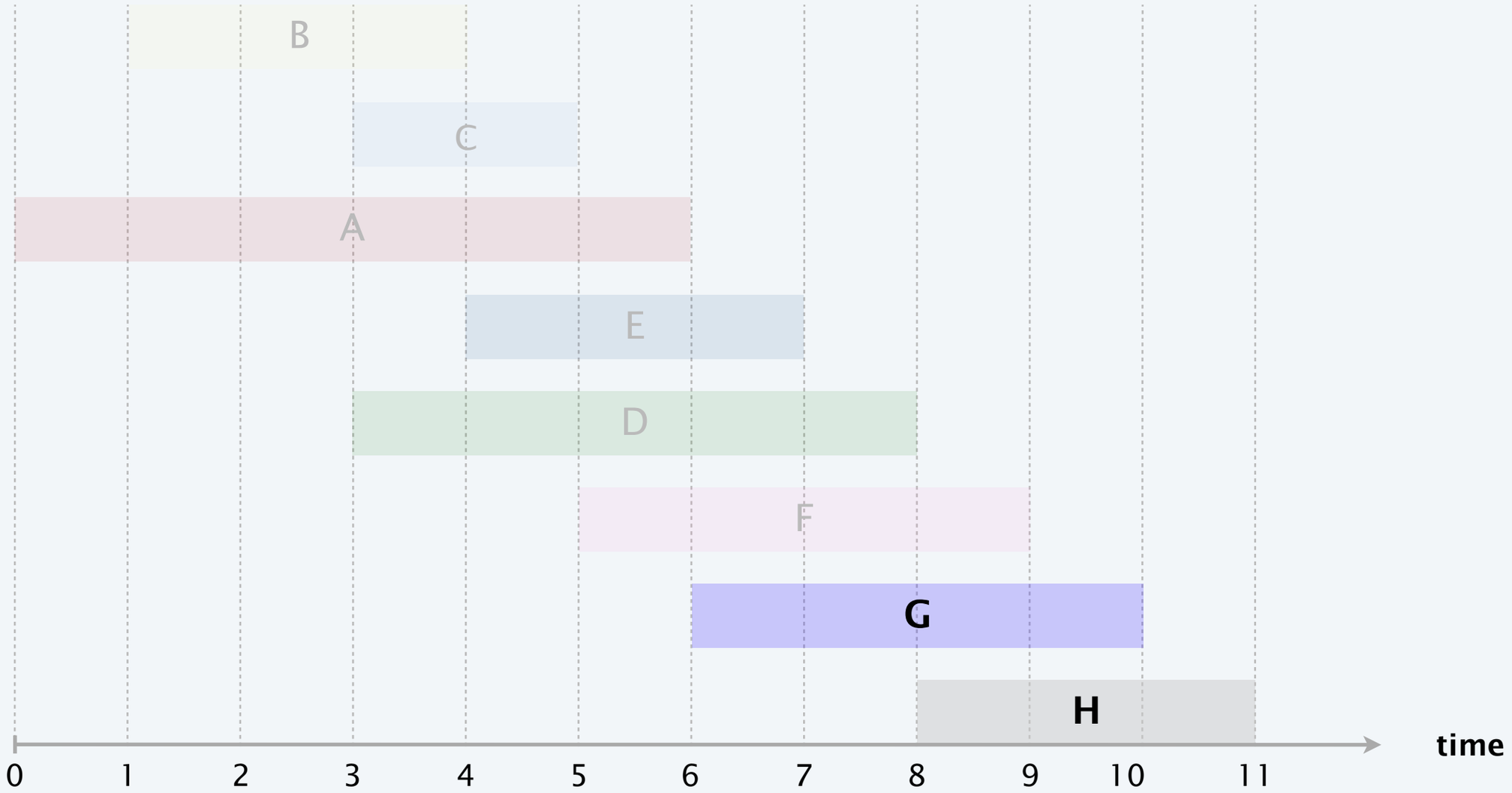
Earliest-finish-time-first algorithm demo



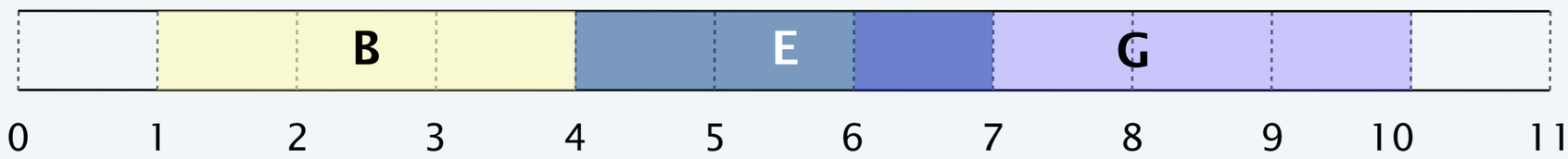
job G is incompatible (do not add to schedule)



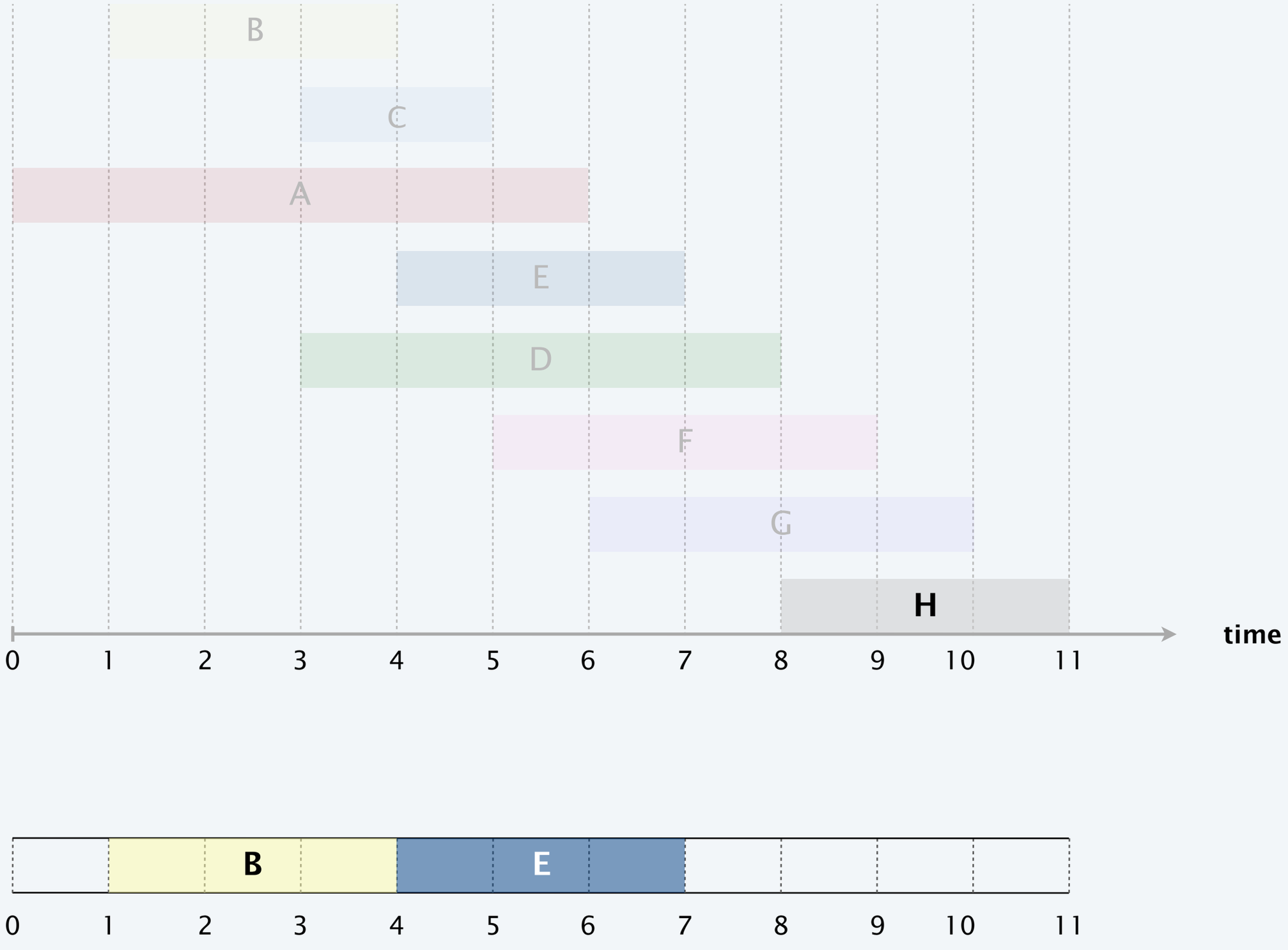
Earliest-finish-time-first algorithm demo



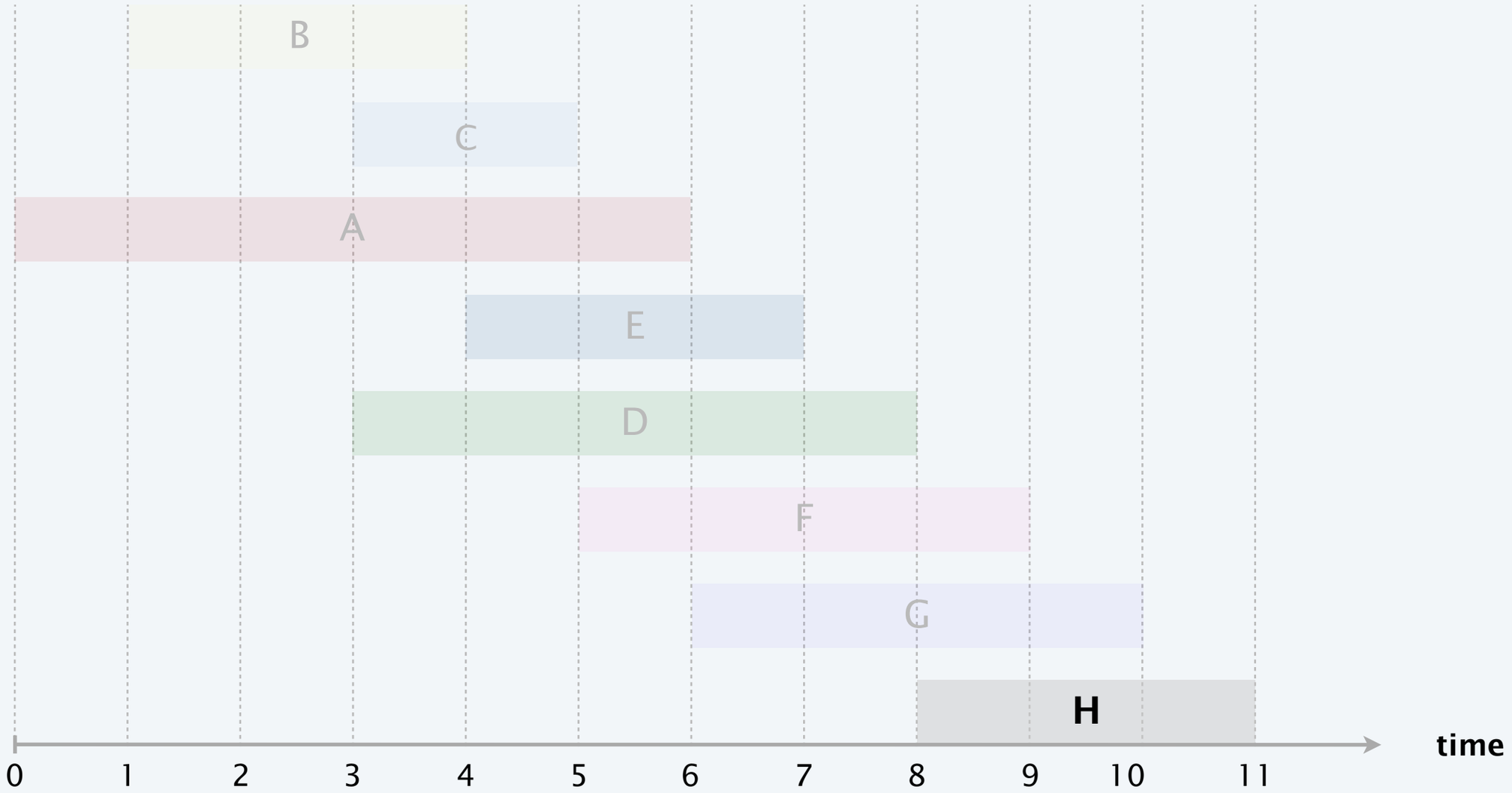
job G is incompatible (do not add to schedule)



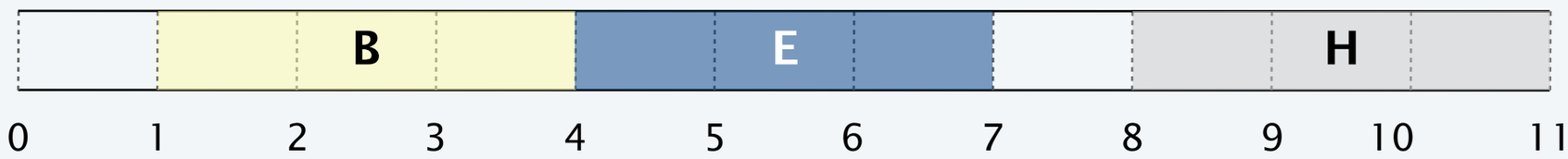
Earliest-finish-time-first algorithm demo



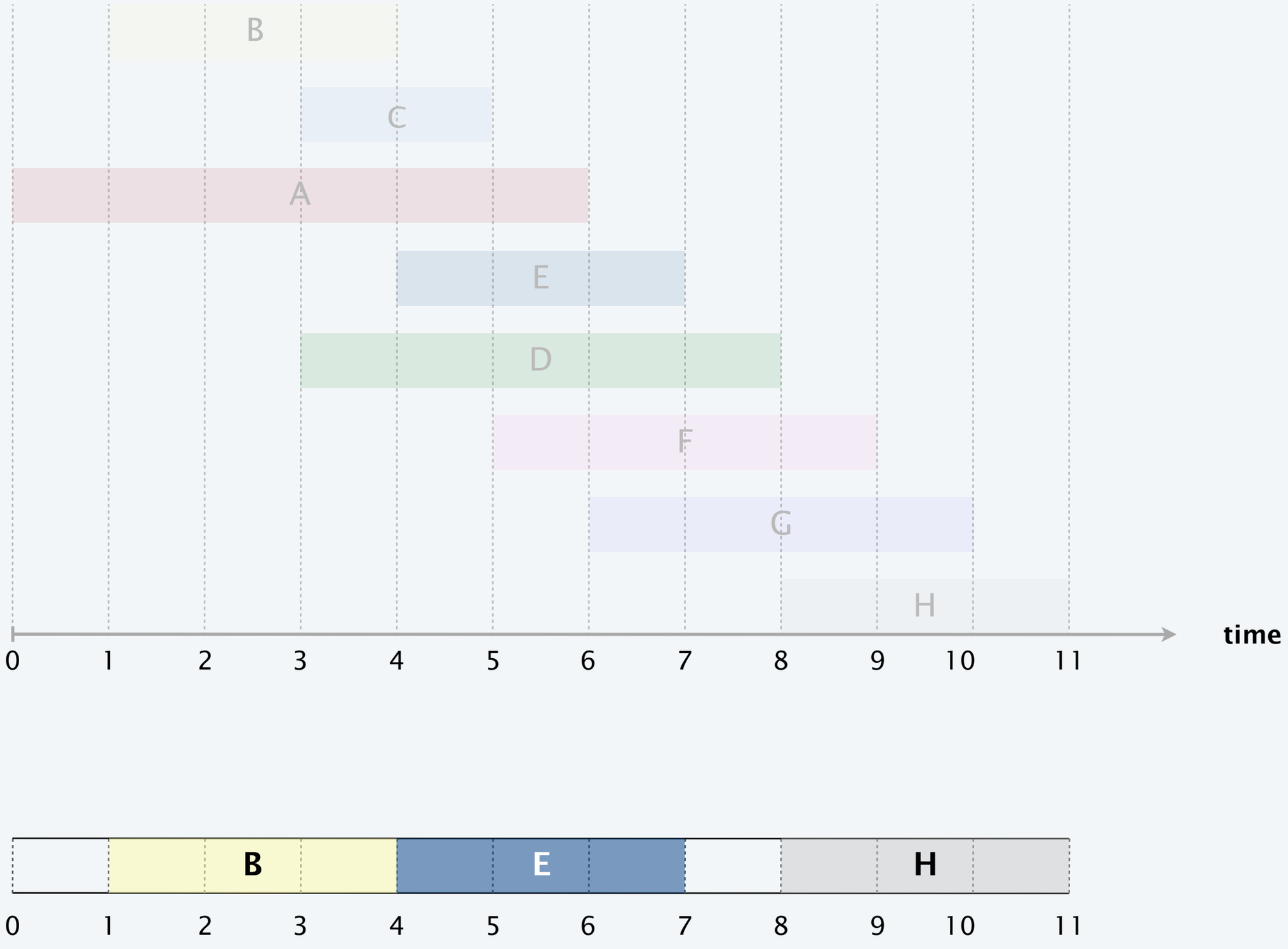
Earliest-finish-time-first algorithm demo



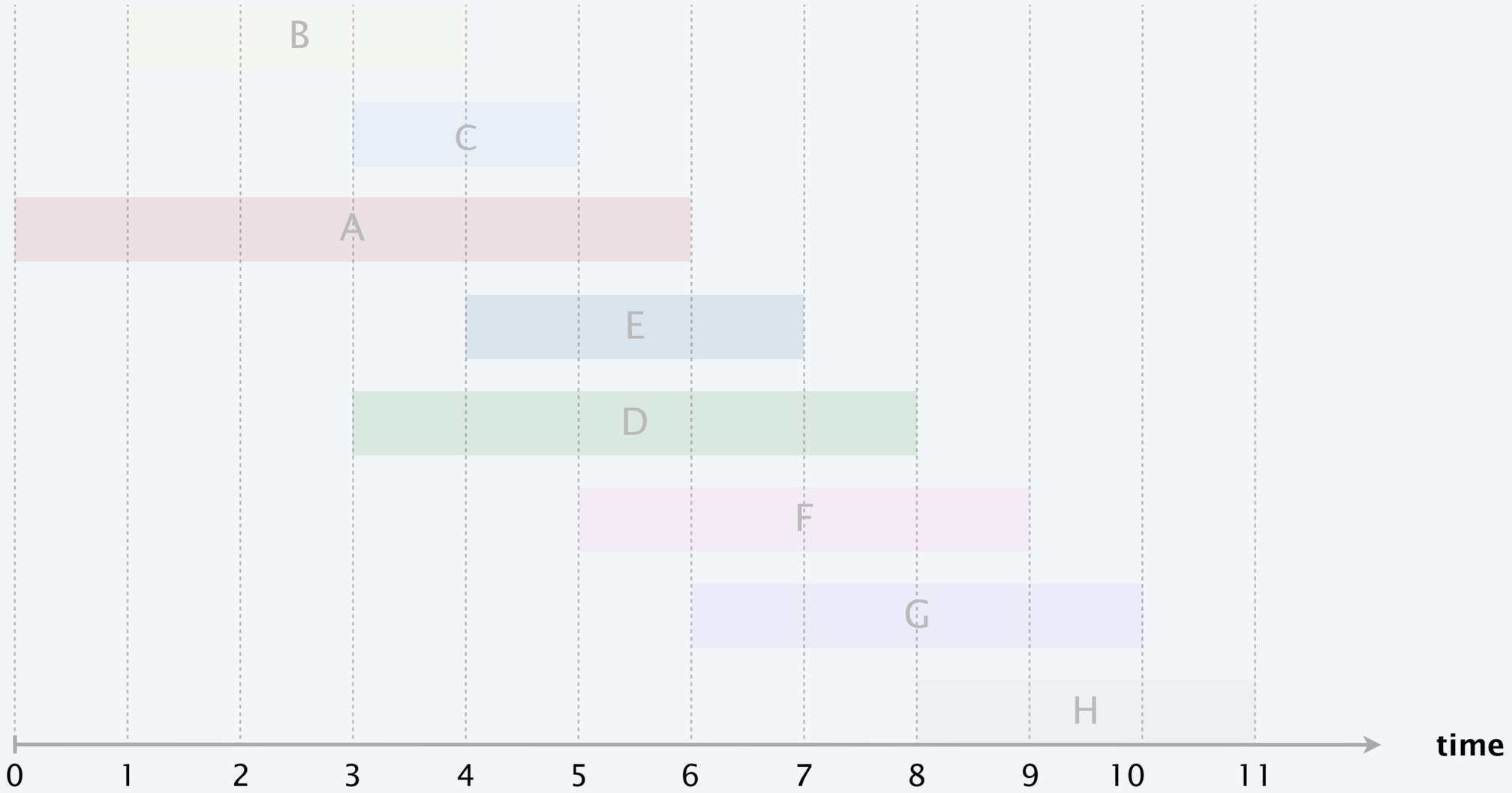
job H is compatible (add to schedule)



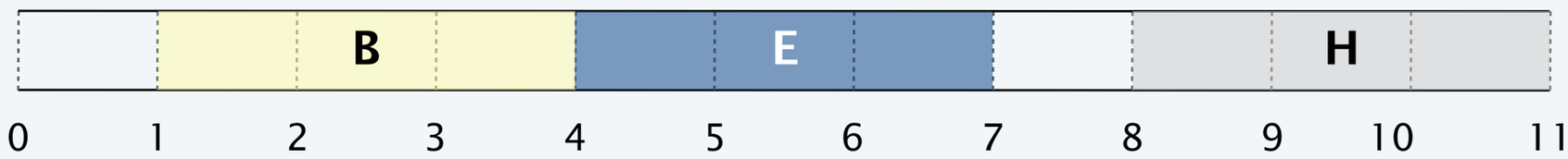
Earliest-finish-time-first algorithm demo

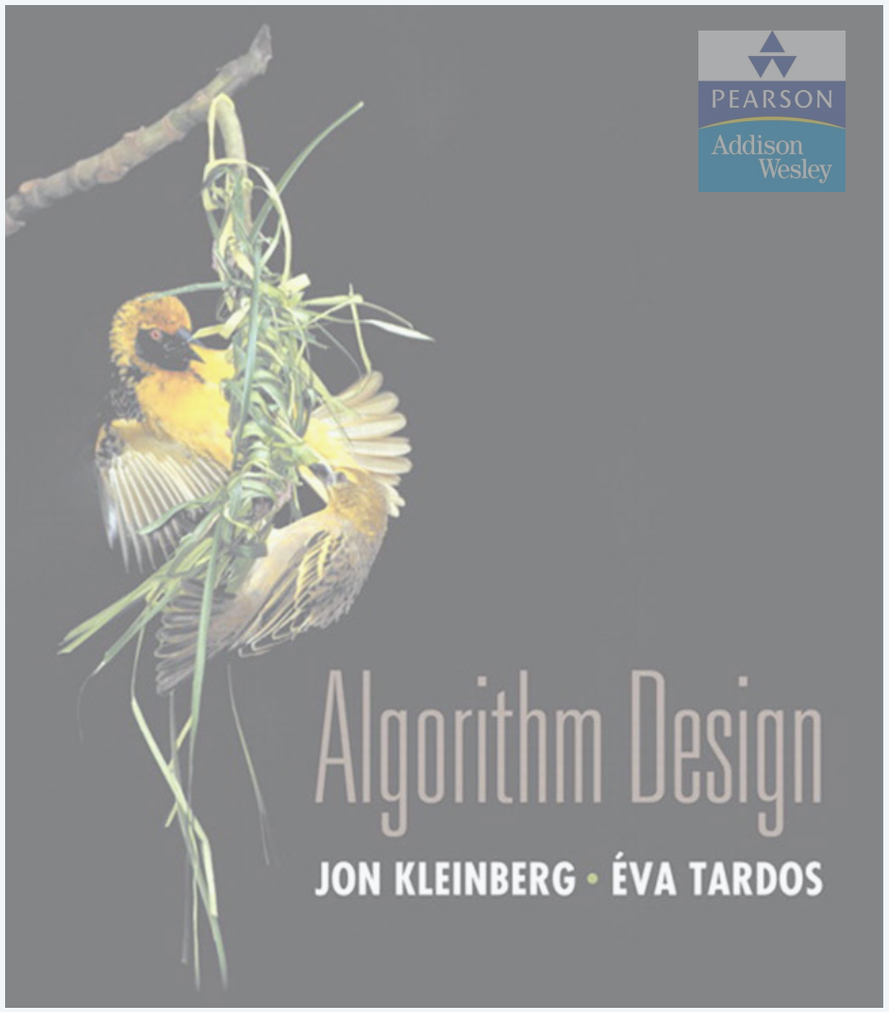


Earliest-finish-time-first algorithm demo



done (an optimal set of jobs)





SECTION 4.1

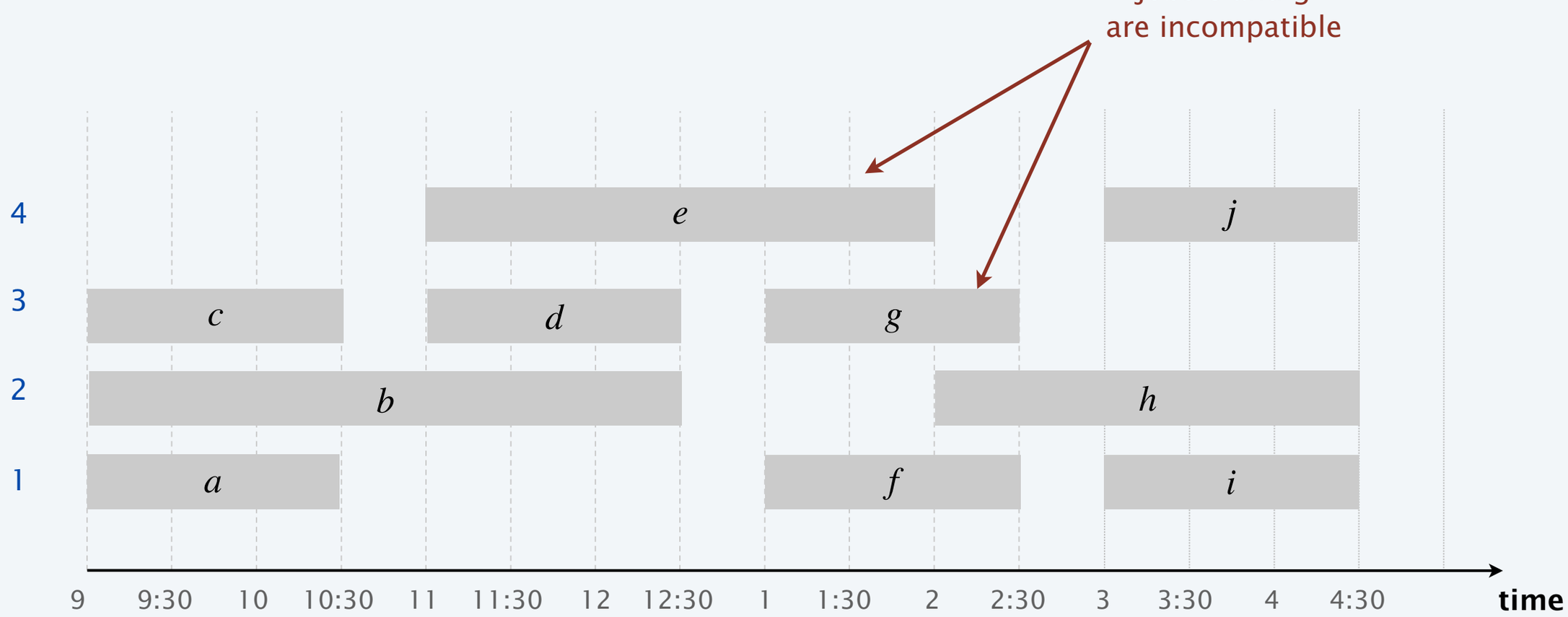
4. GREEDY ALGORITHMS I

- *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.

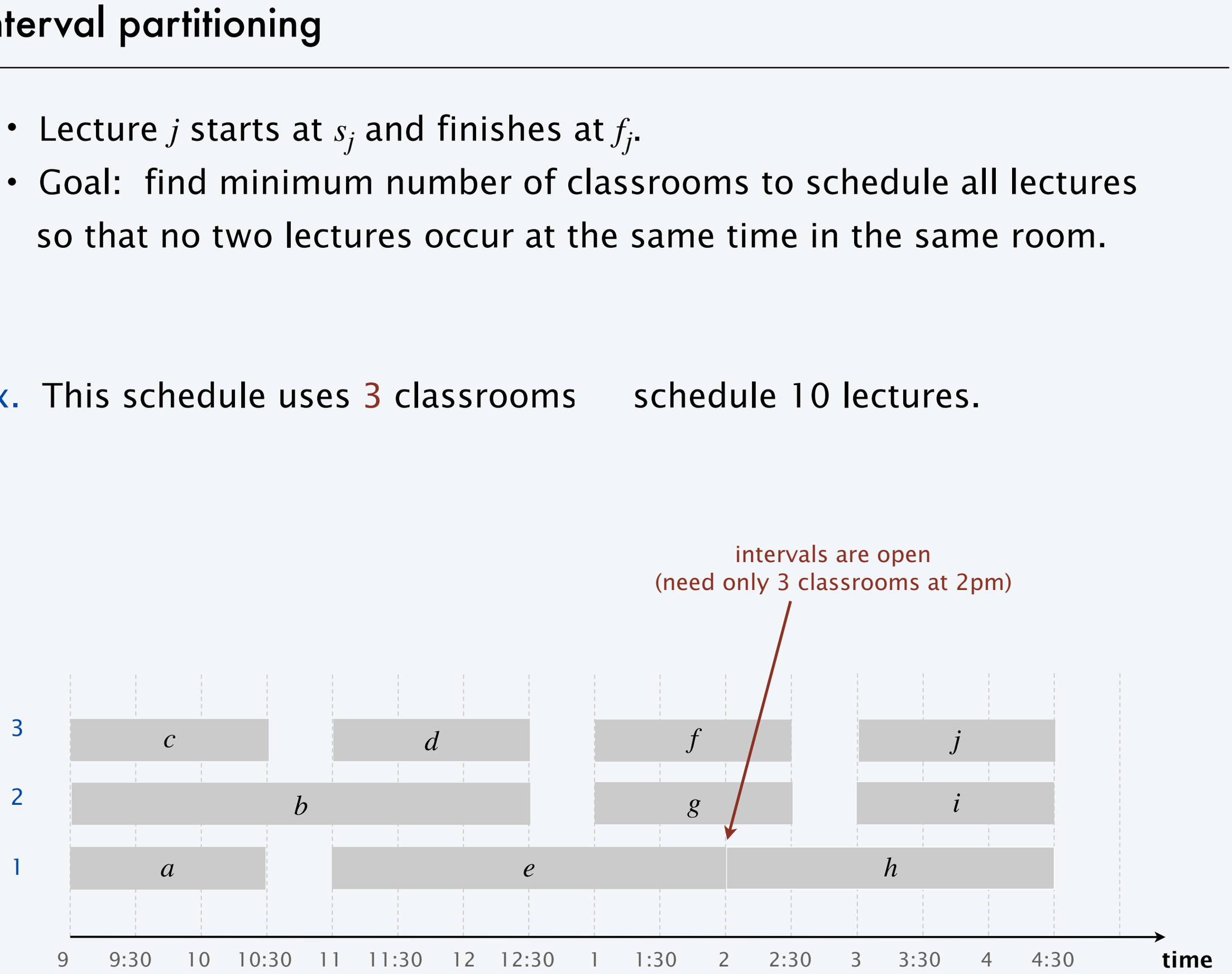
Ex. This schedule uses 4 classrooms to schedule 10 lectures.



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.



Greedy algorithms I: quiz 4

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 $(n, s_1, s_2, \ldots, s_n, f_1, f_2, \ldots, f_n)$

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

$d \leftarrow 0$. ← 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.

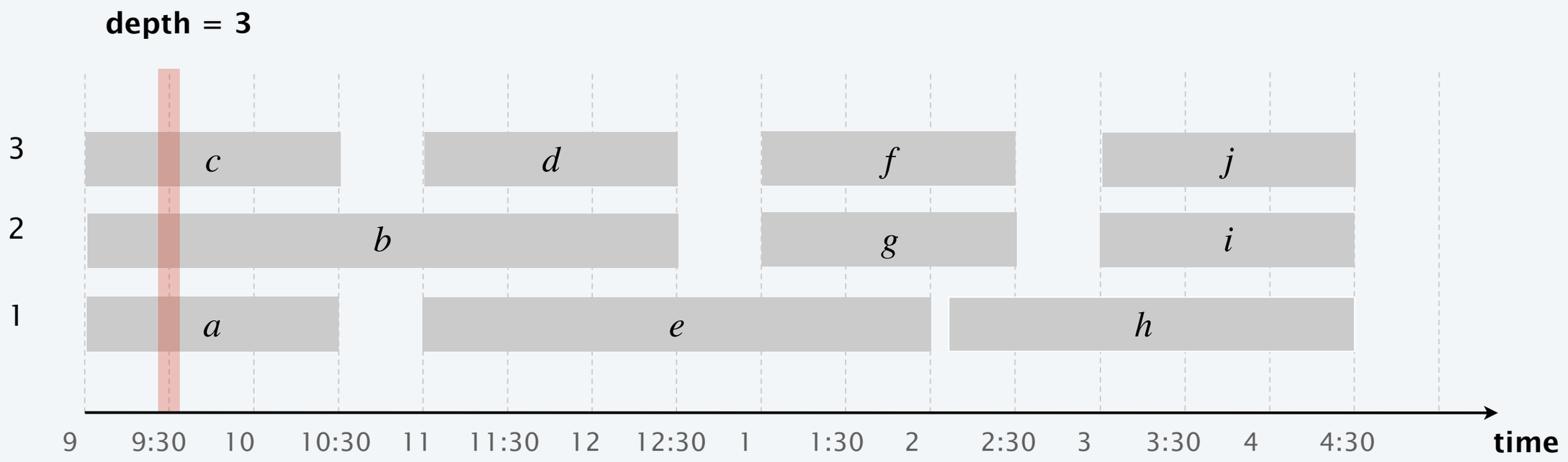
Interval partitioning: lower bound on optimal solution

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.

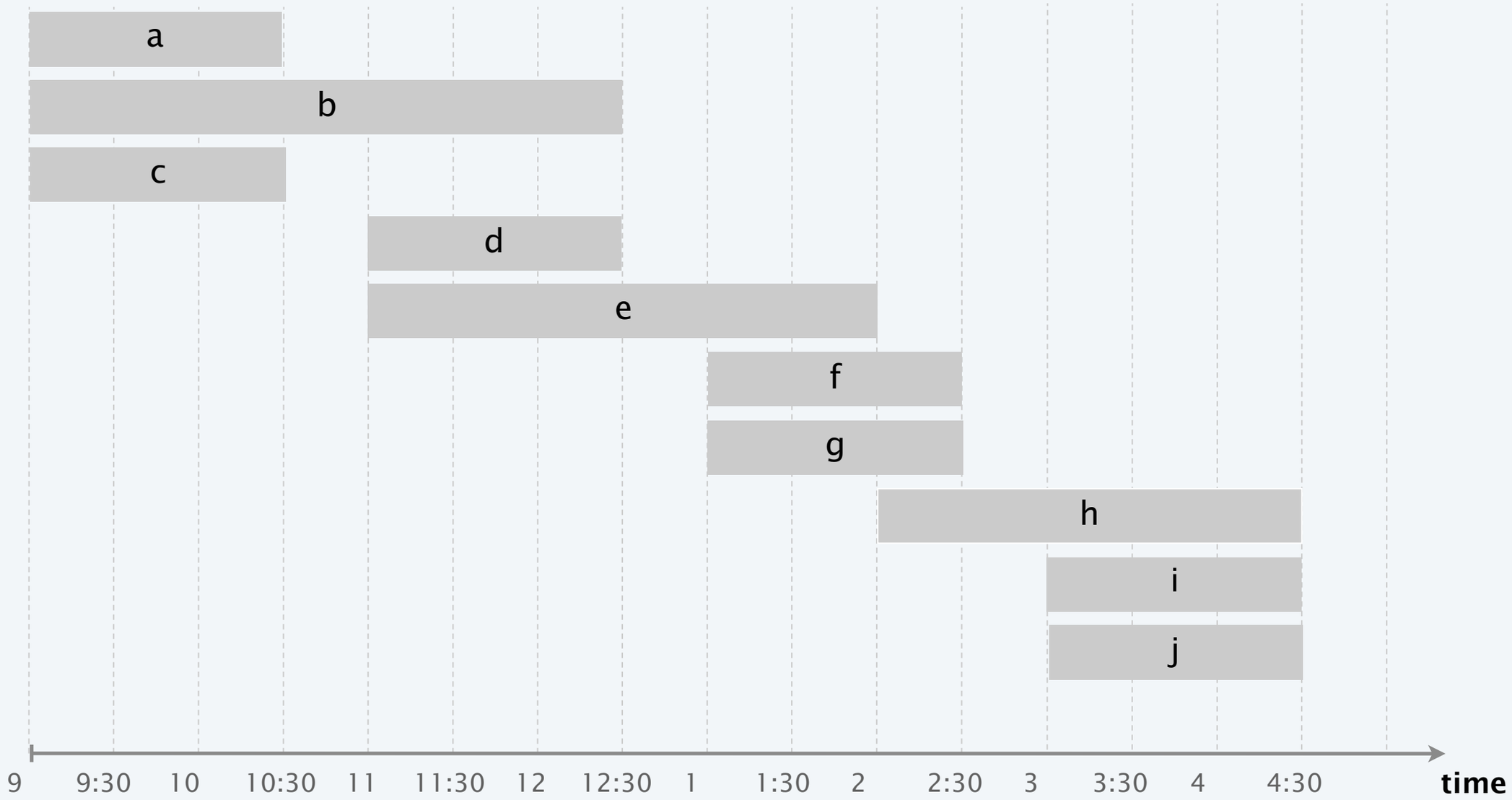
Pf.

- 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 $s_j + \varepsilon$.
- Key observation $\Rightarrow$ all schedules use $\geq d$ classrooms. ▪

Earliest-start-time-first algorithm demo

Consider lectures in order of start time:

- Assign next lecture to any compatible classroom (if one exists).
- Otherwise, open up a new classroom.

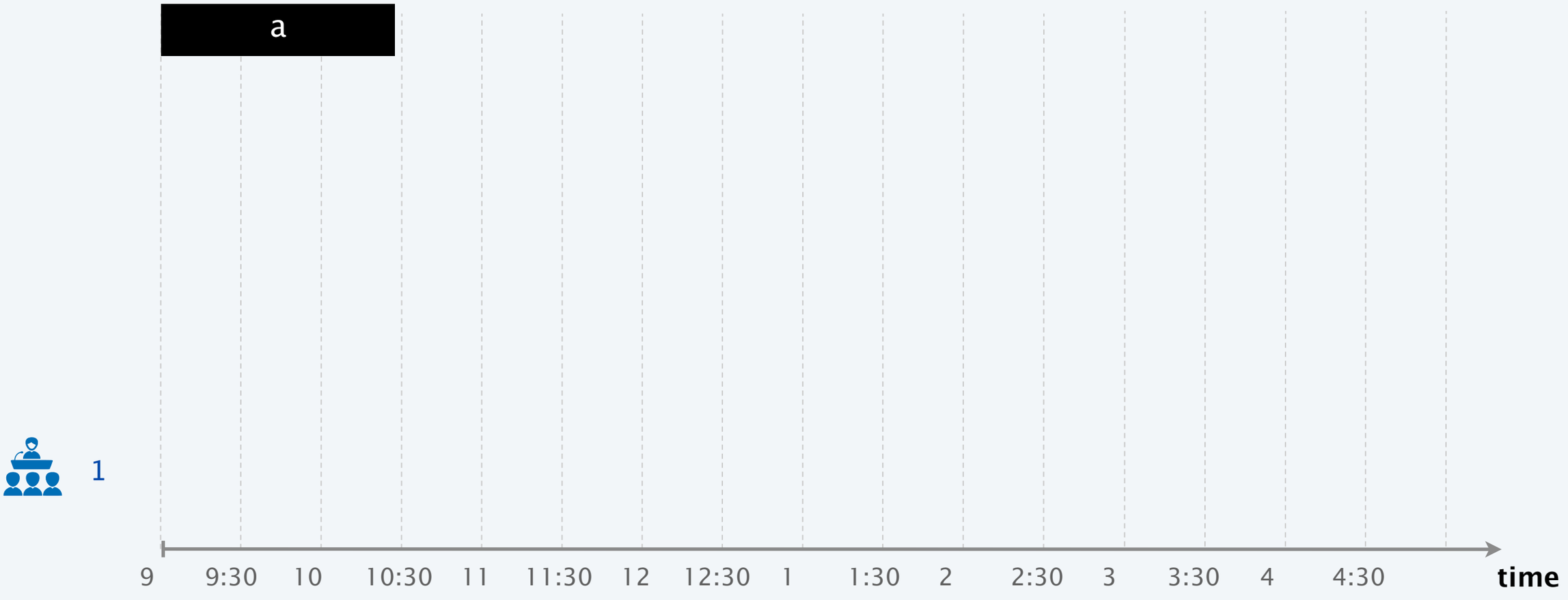


Earliest-start-time-first algorithm demo

Consider lectures in order of start time:

- Assign next lecture to any compatible classroom (if one exists).
- Otherwise, open up a new classroom.

no compatible classroom: open up a new classroom and assign lecture to it

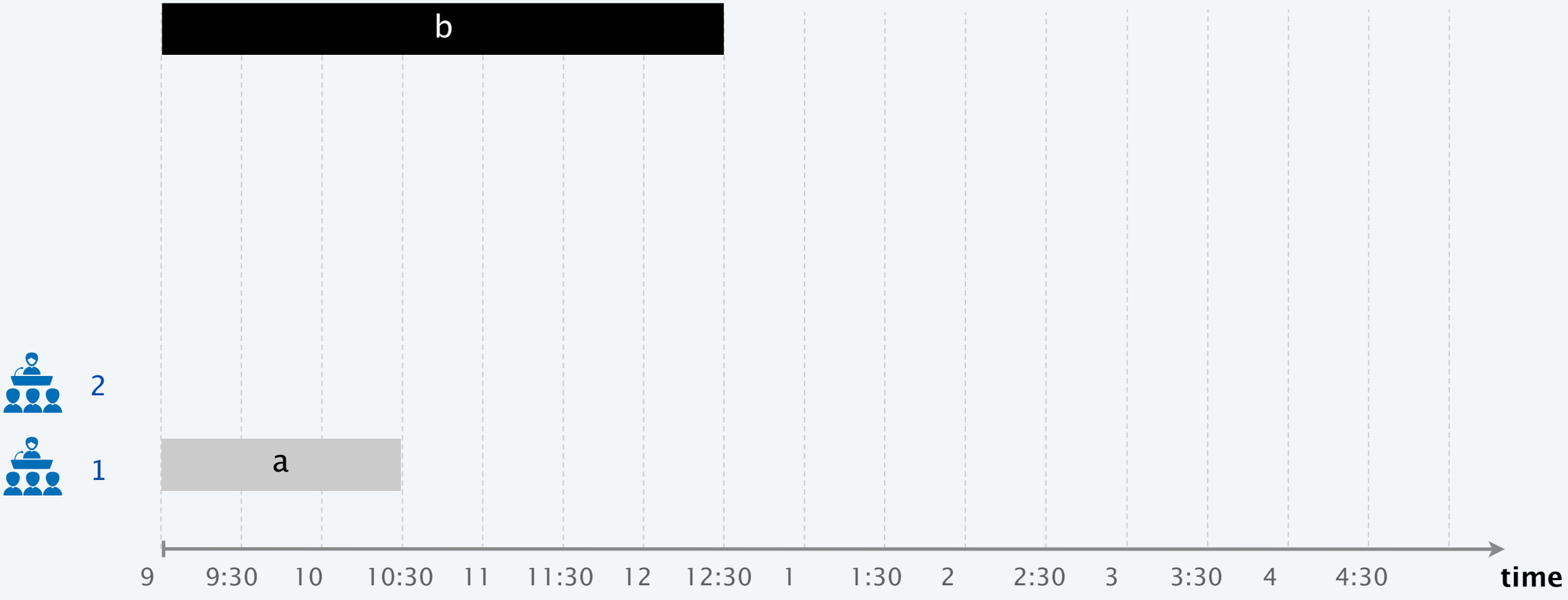


Earliest-start-time-first algorithm demo

Consider lectures in order of start time:

- Assign next lecture to any compatible classroom (if one exists).
- Otherwise, open up a new classroom.

no compatible classroom: open up a new classroom and assign lecture to it

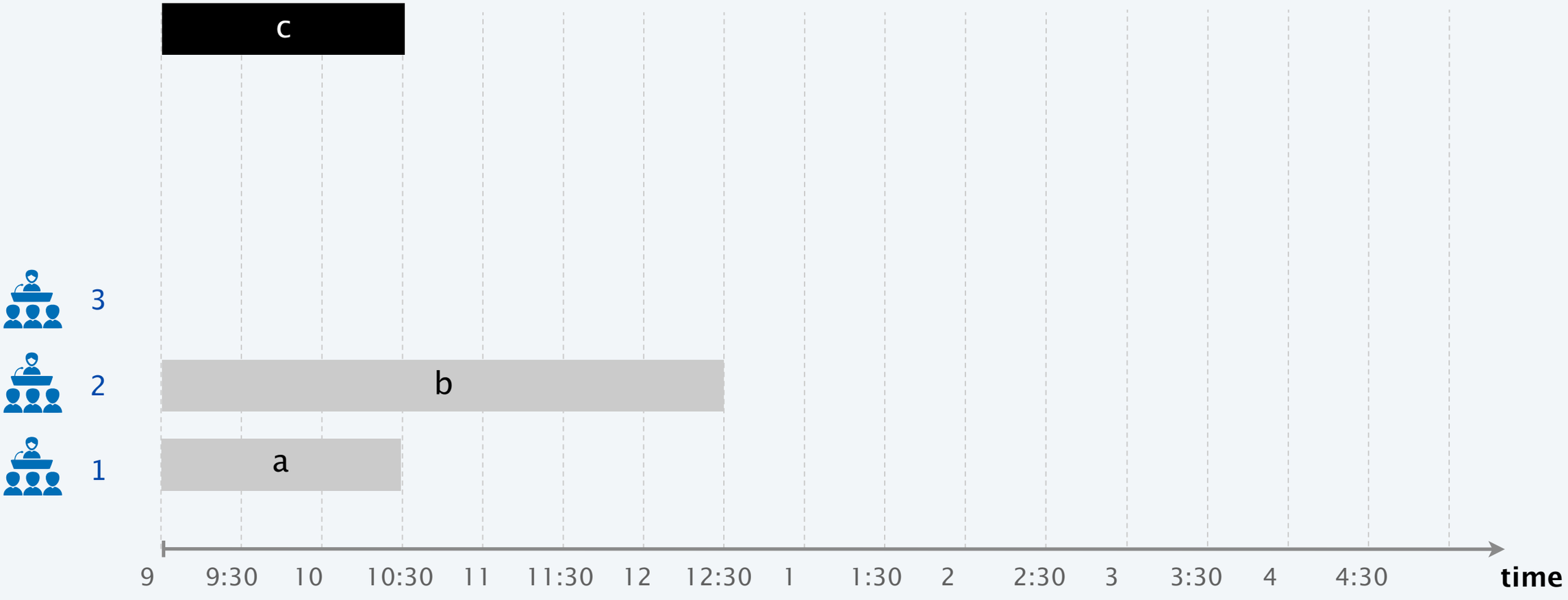


Earliest-start-time-first algorithm demo

Consider lectures in order of start time:

- Assign next lecture to any compatible classroom (if one exists).
- Otherwise, open up a new classroom.

no compatible classroom: open up a new classroom and assign lecture to it

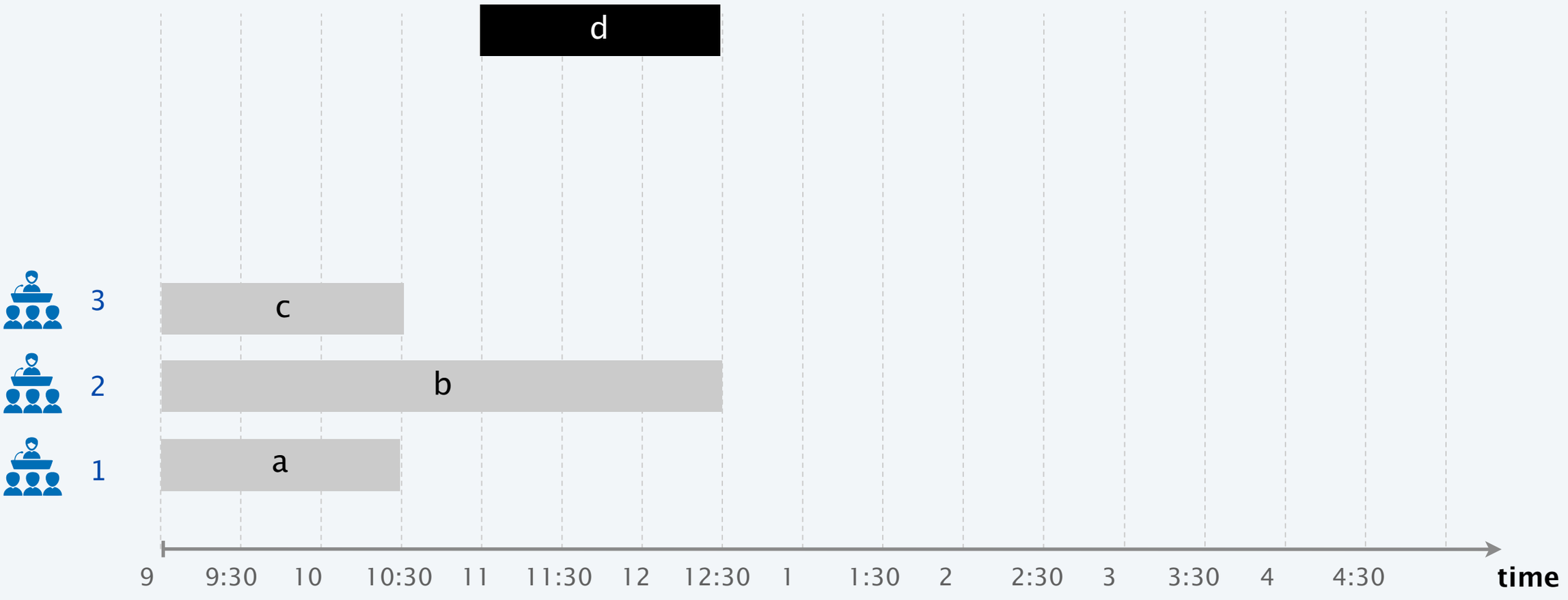


Earliest-start-time-first algorithm demo

Consider lectures in order of start time:

- Assign next lecture to any compatible classroom (if one exists).
- Otherwise, open up a new classroom.

lecture d is compatible with classrooms 1 and 3

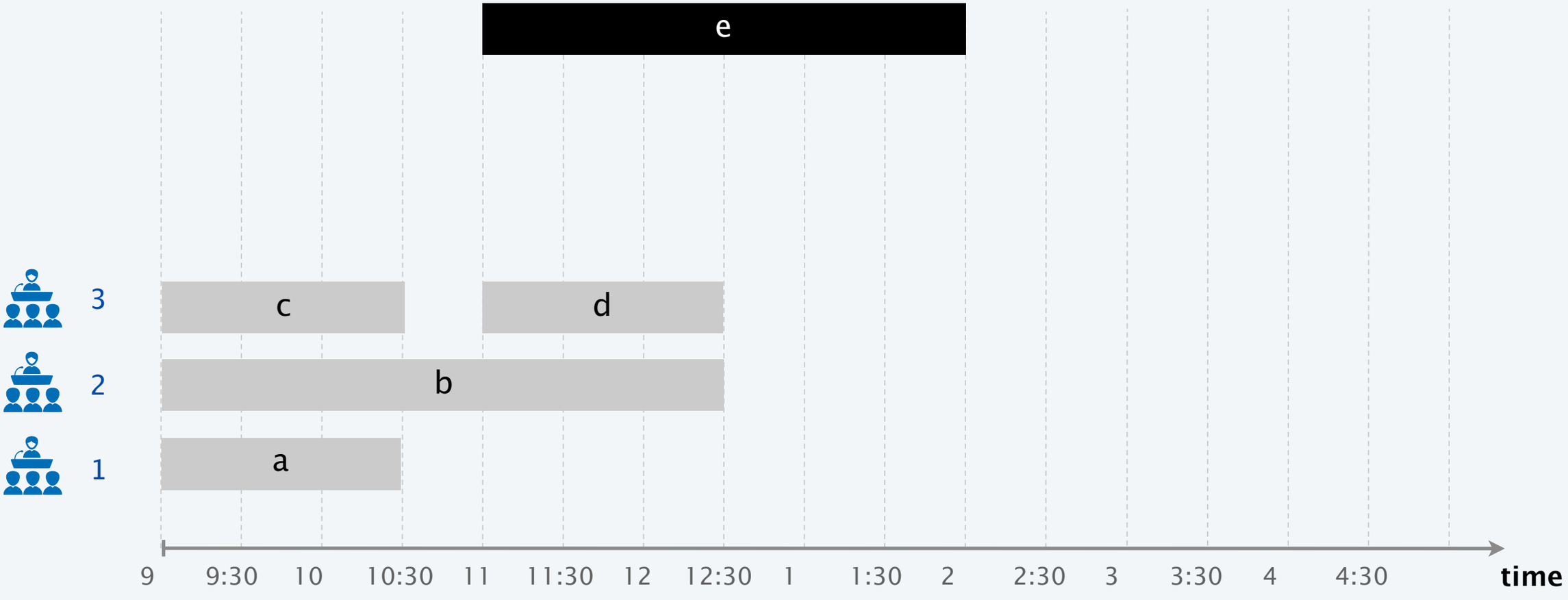


Earliest-start-time-first algorithm demo

Consider lectures in order of start time:

- Assign next lecture to any compatible classroom (if one exists).
- Otherwise, open up a new classroom.

lecture e is compatible with classroom 1

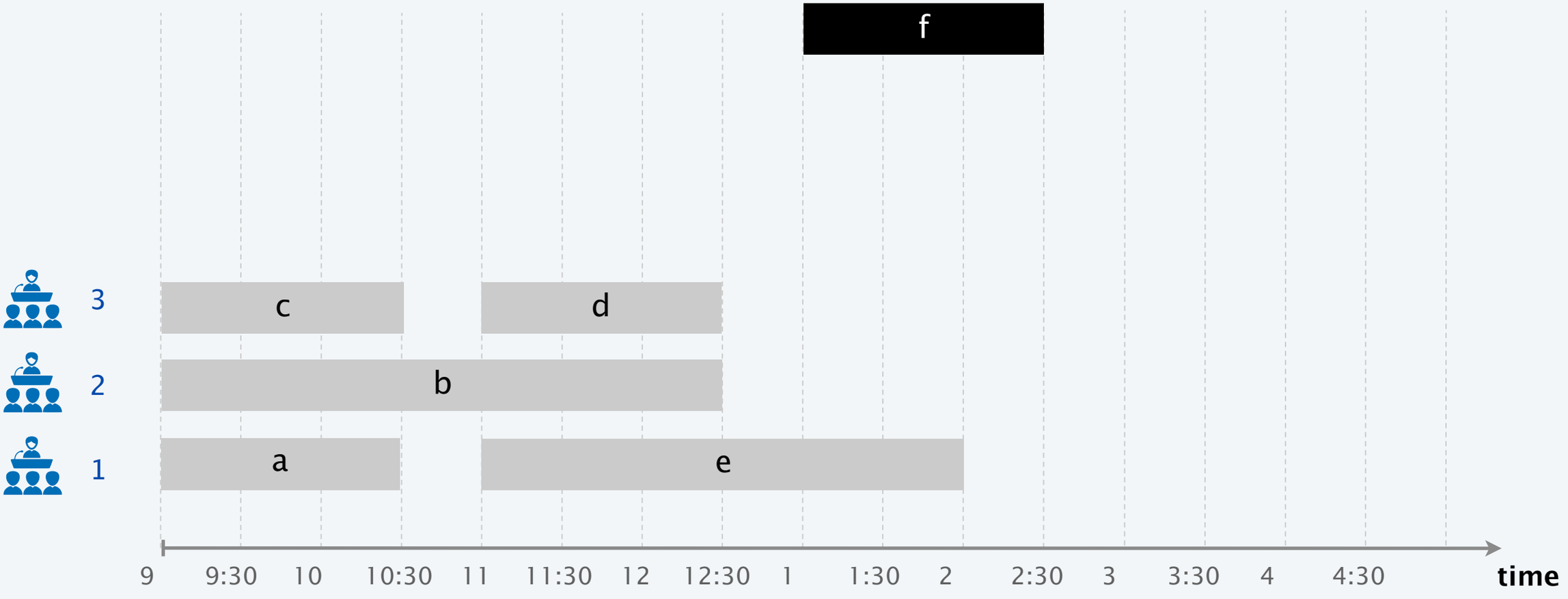


Earliest-start-time-first algorithm demo

Consider lectures in order of start time:

- Assign next lecture to any compatible classroom (if one exists).
- Otherwise, open up a new classroom.

lecture f is compatible with classroom 2 and 3

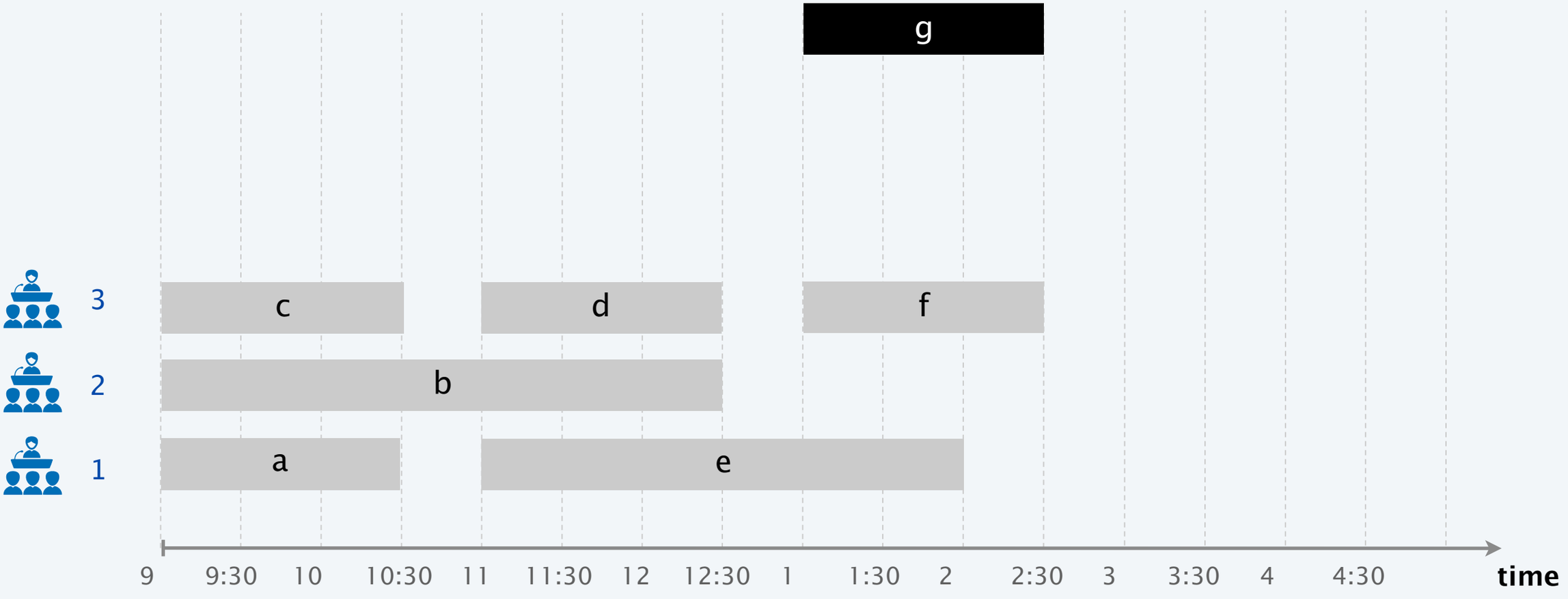


Earliest-start-time-first algorithm demo

Consider lectures in order of start time:

- Assign next lecture to any compatible classroom (if one exists).
- Otherwise, open up a new classroom.

lecture g is compatible with classroom 2

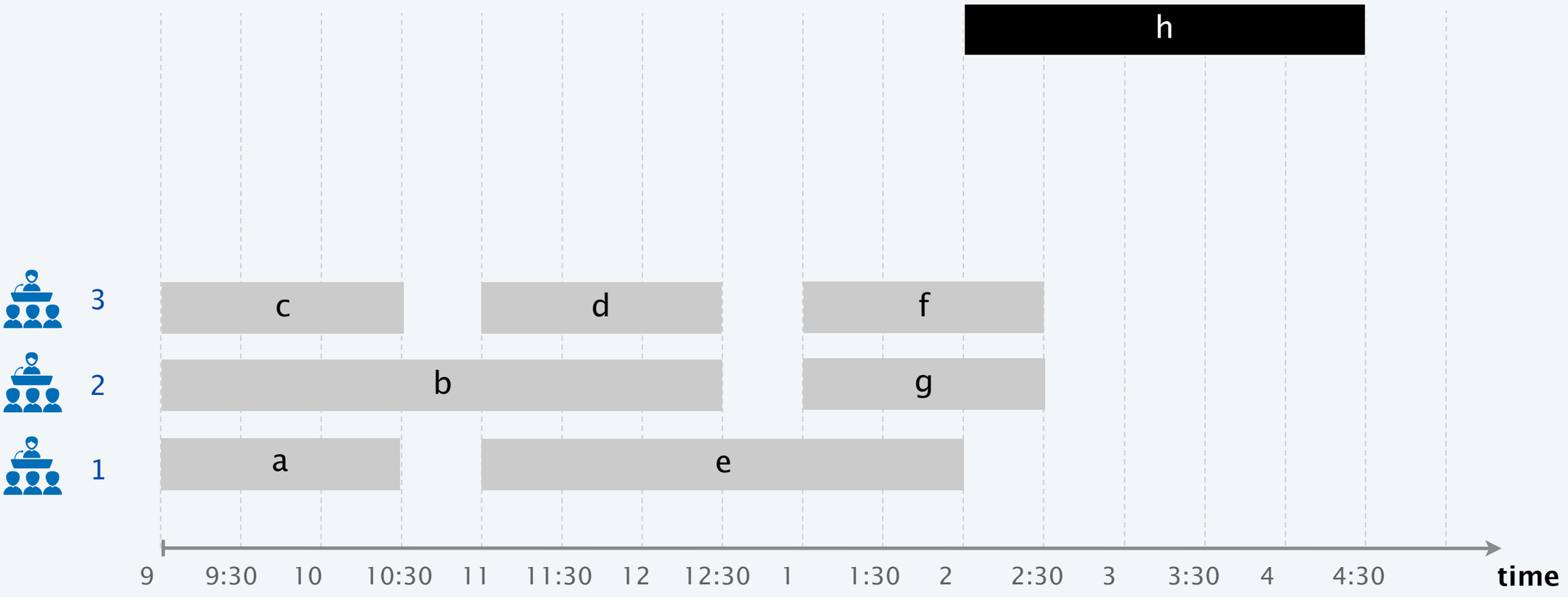


Earliest-start-time-first algorithm demo

Consider lectures in order of start time:

- Assign next lecture to any compatible classroom (if one exists).
- Otherwise, open up a new classroom.

lecture h is compatible with classroom 1

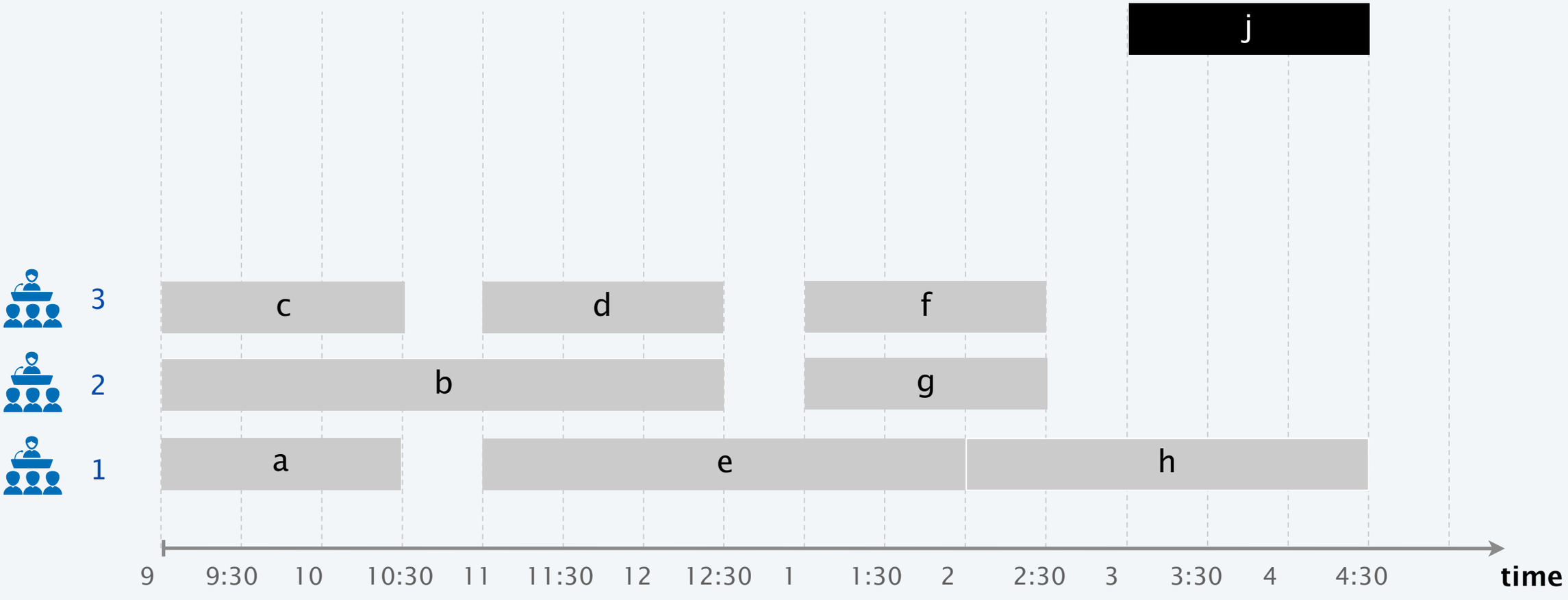


Earliest-start-time-first algorithm demo

Consider lectures in order of start time:

- Assign next lecture to any compatible classroom (if one exists).
- Otherwise, open up a new classroom.

lecture j is compatible with classrooms 2 and 3

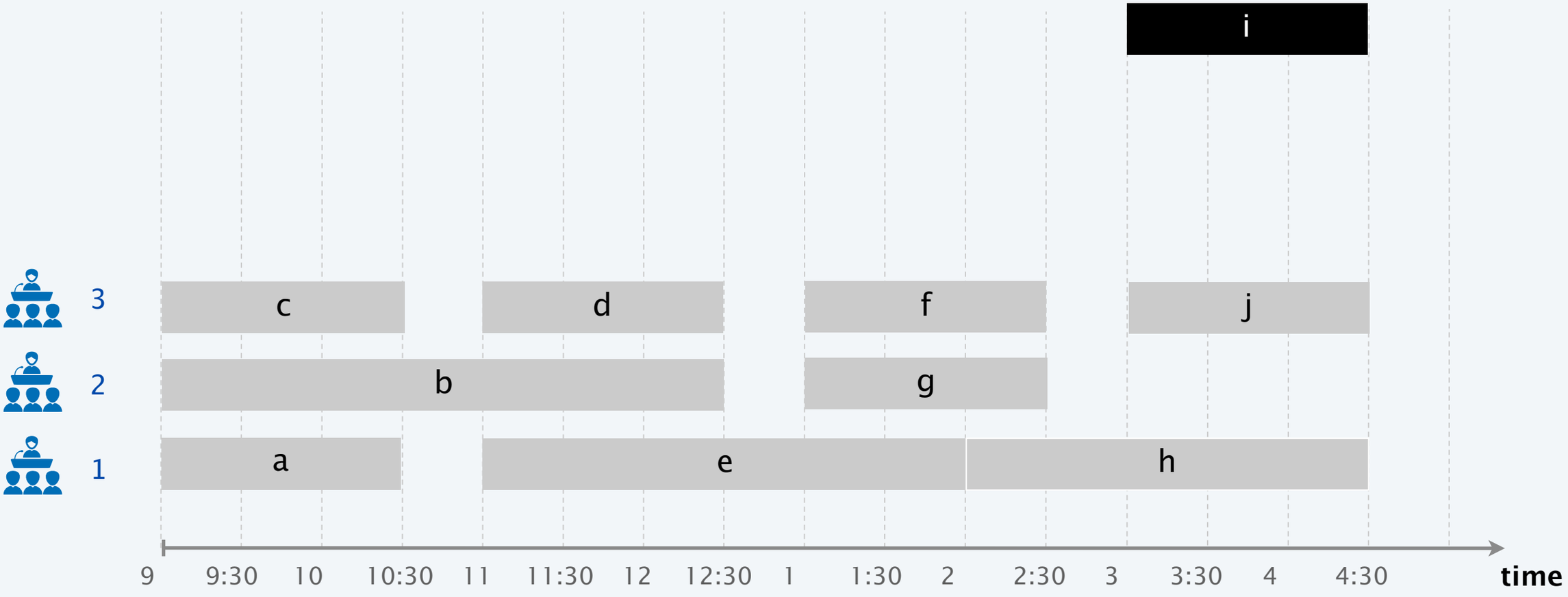


Earliest-start-time-first algorithm demo

Consider lectures in order of start time:

- Assign next lecture to any compatible classroom (if one exists).
- Otherwise, open up a new classroom.

lecture i is compatible with classroom 2

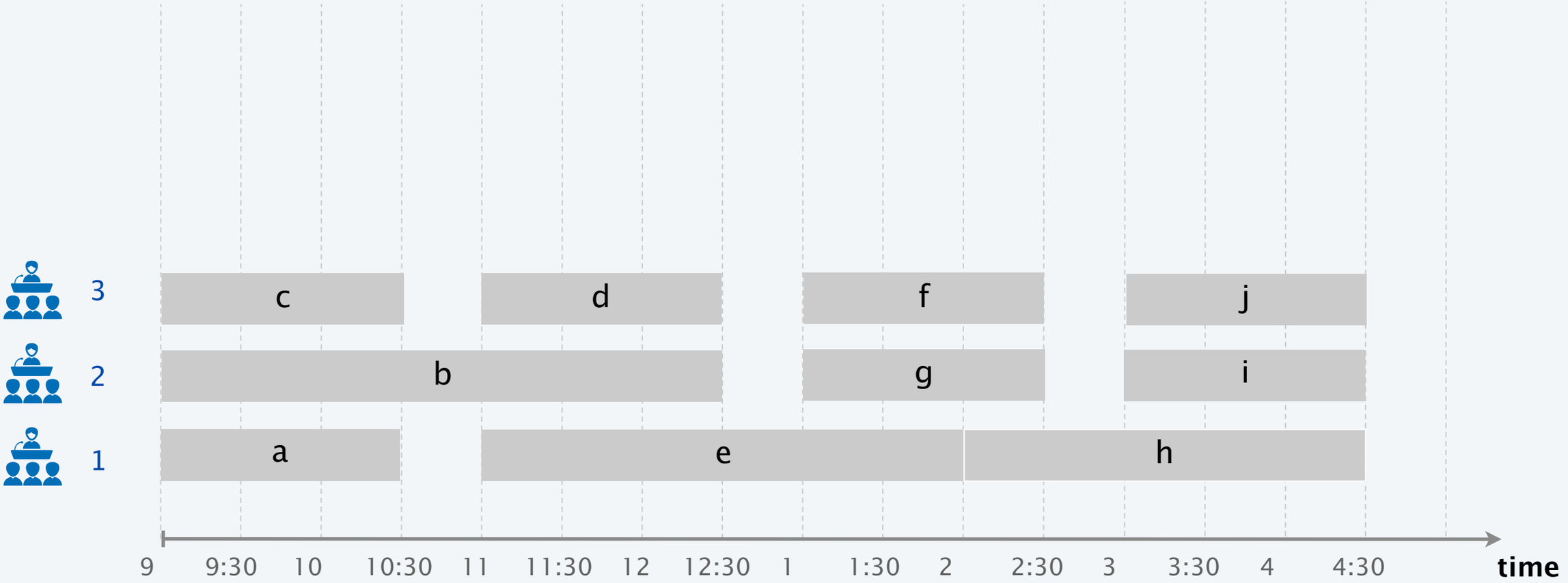


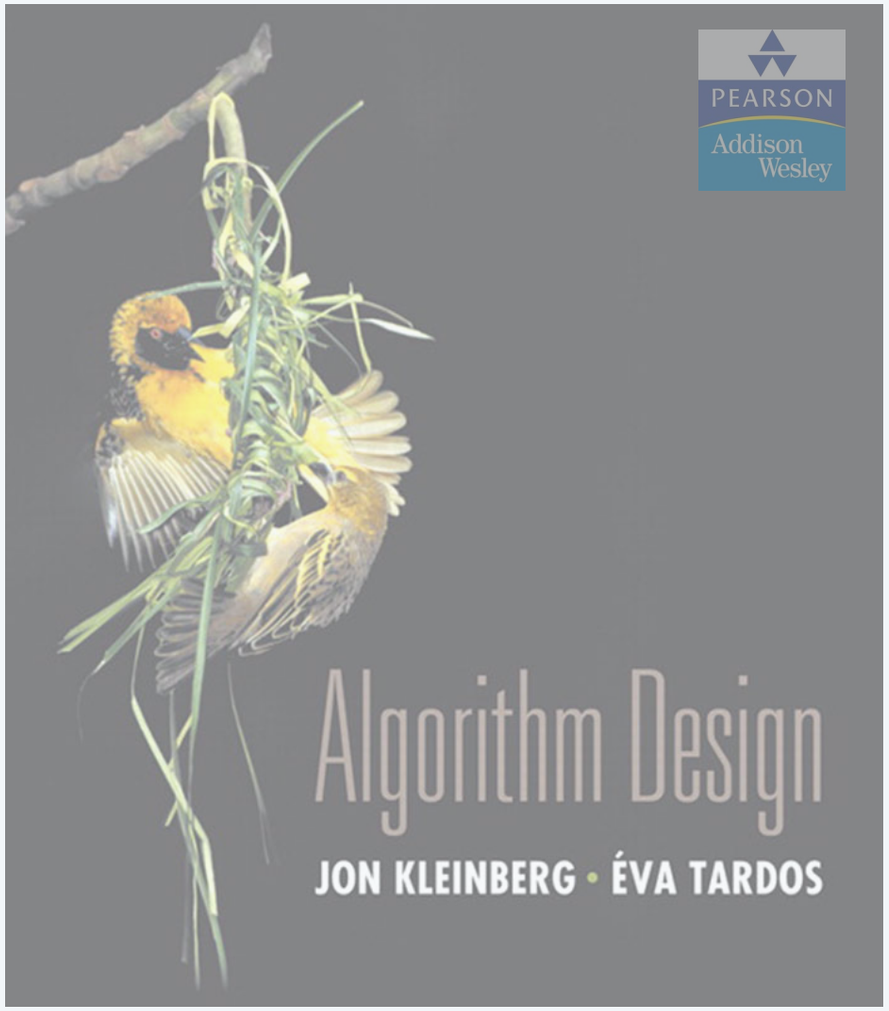
Earliest-start-time-first algorithm demo

Consider lectures in order of start time:

- Assign next lecture to any compatible classroom (if one exists).
- Otherwise, open up a new classroom.

done





4. GREEDY ALGORITHMS I

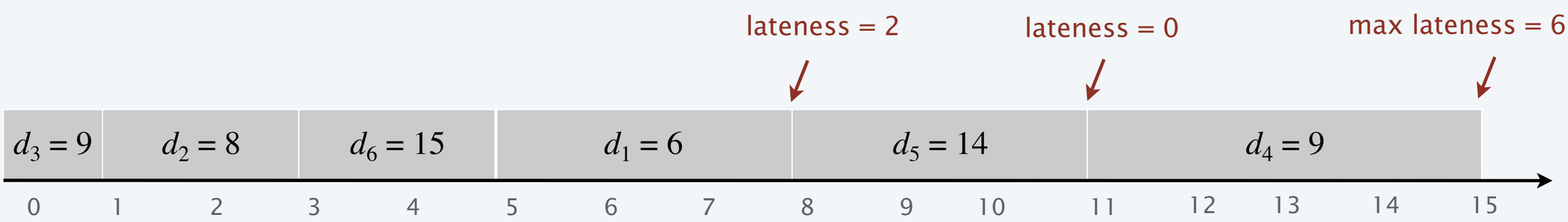
- *coin changing*
- *interval scheduling*
- *interval partitioning*
- *scheduling to minimize lateness*
- *optimal caching*

SECTION 4.2

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



Greedy algorithms I: quiz 5

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 $(n, t_1, t_2, \ldots, t_n, d_1, d_2, \ldots, d_n)$

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

$t \leftarrow 0$.

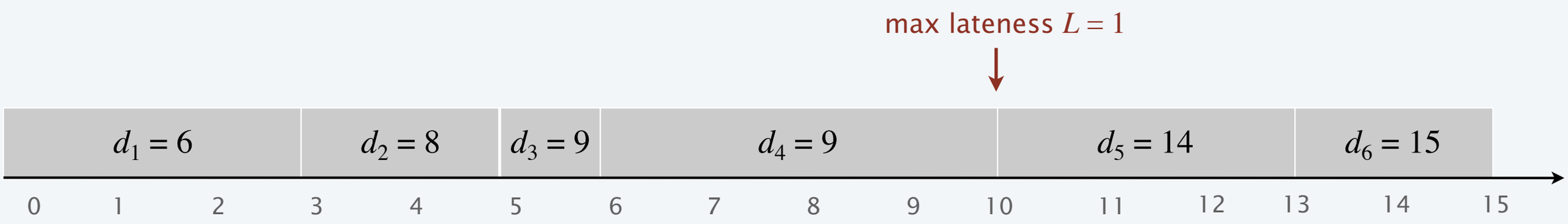
FOR $j = 1$ TO n

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

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

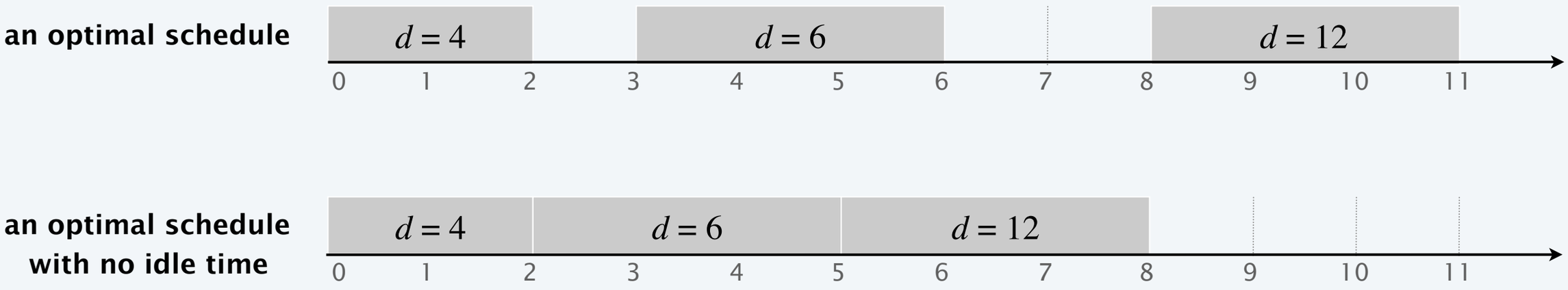
  $t \leftarrow t + t_j$.

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



Minimizing lateness: no idle time

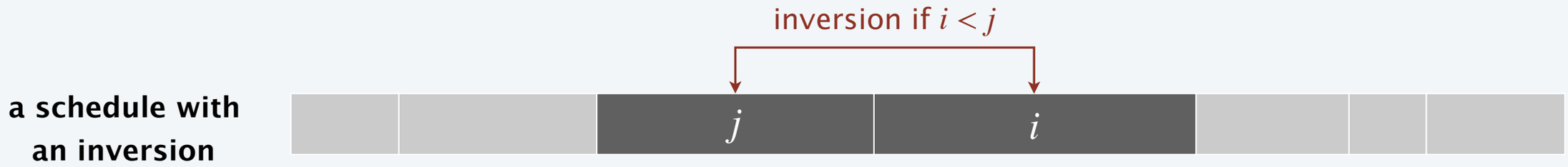
Observation 1. There exists an optimal schedule with no idle time.



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.



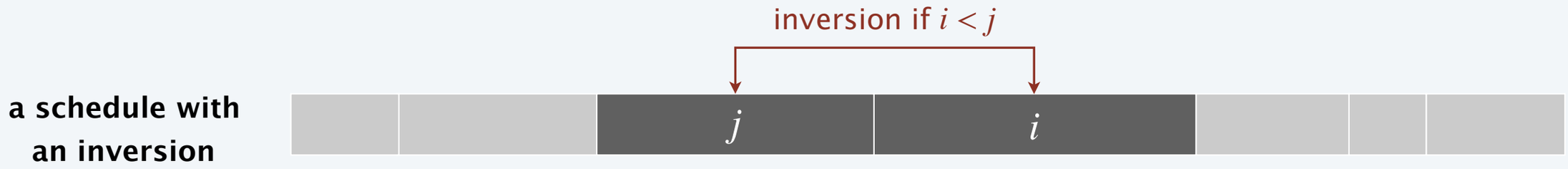
recall: we assume the jobs are numbered so that $d_1 \leq d_2 \leq \ldots \leq d_n$

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

1	2	3	4	5	6	…	n

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 $d_1 \le d_2 \le \ldots \le d_n$

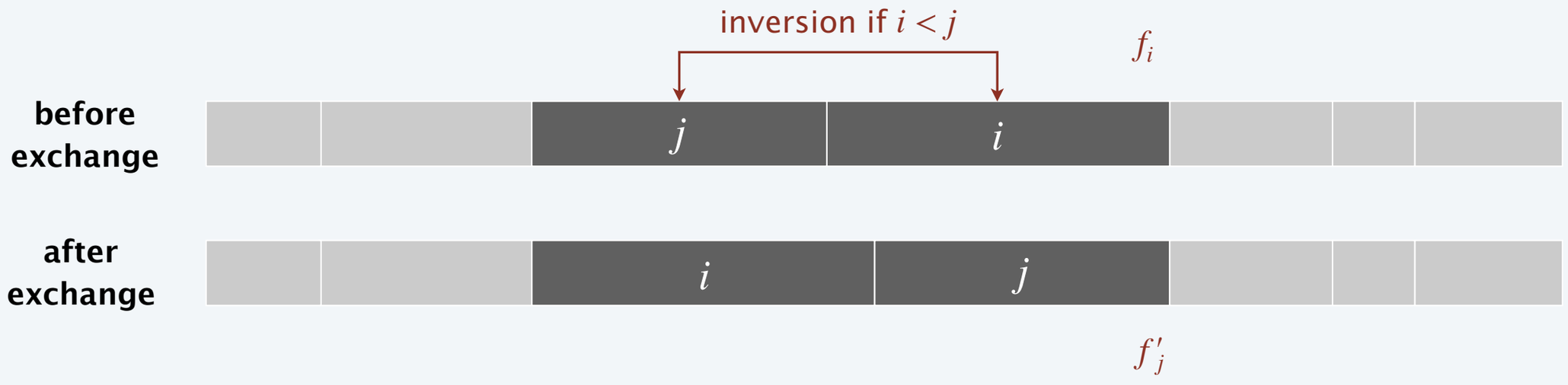
Observation 4. If an idle-free schedule has an inversion, then it has an adjacent inversion.

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,

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

Minimizing lateness: analysis of earliest-deadline-first algorithm

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

optimal schedule can have inversions

Pf. [by contradiction]

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

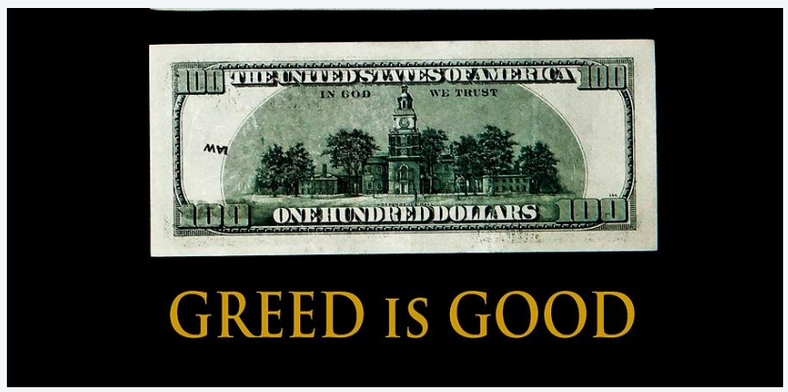
Greedy analysis strategies

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, ...



GOOGLE'S FOO.BAR CHALLENGE

Quantum antimatter fuel comes in small pellets, which is convenient since the many moving parts of the LAMBCHOP each need to be fed fuel one pellet at a time. However, minions dump pellets in bulk into the fuel intake. You need to figure out the most efficient way to sort and shift the pellets down to a single pellet at a time.

The fuel control mechanisms have three operations:

- Add 1 fuel pellet
- Remove 1 fuel pellet
- Divide the entire group of fuel pellets by 2 (due to the destructive energy released when a quantum antimatter pellet is cut in half, the safety controls will only allow this to happen if there is an even number of pellets)

Write a function called answer(n) which takes a positive integer n as a string and returns the minimum number of operations needed to transform the number of pellets to 1.

$$29 \rightarrow 28 \rightarrow 14 \rightarrow 7 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1$$