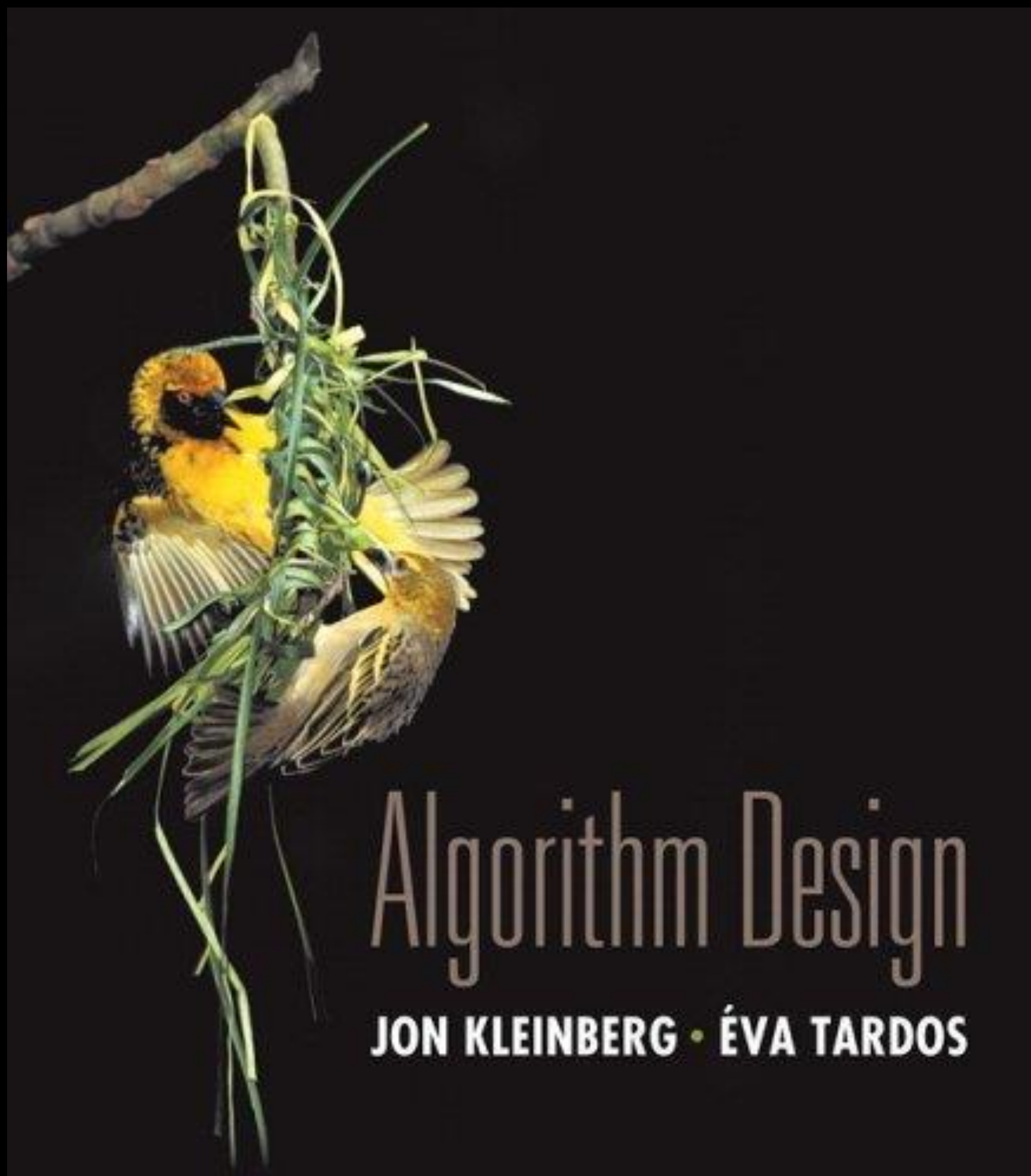


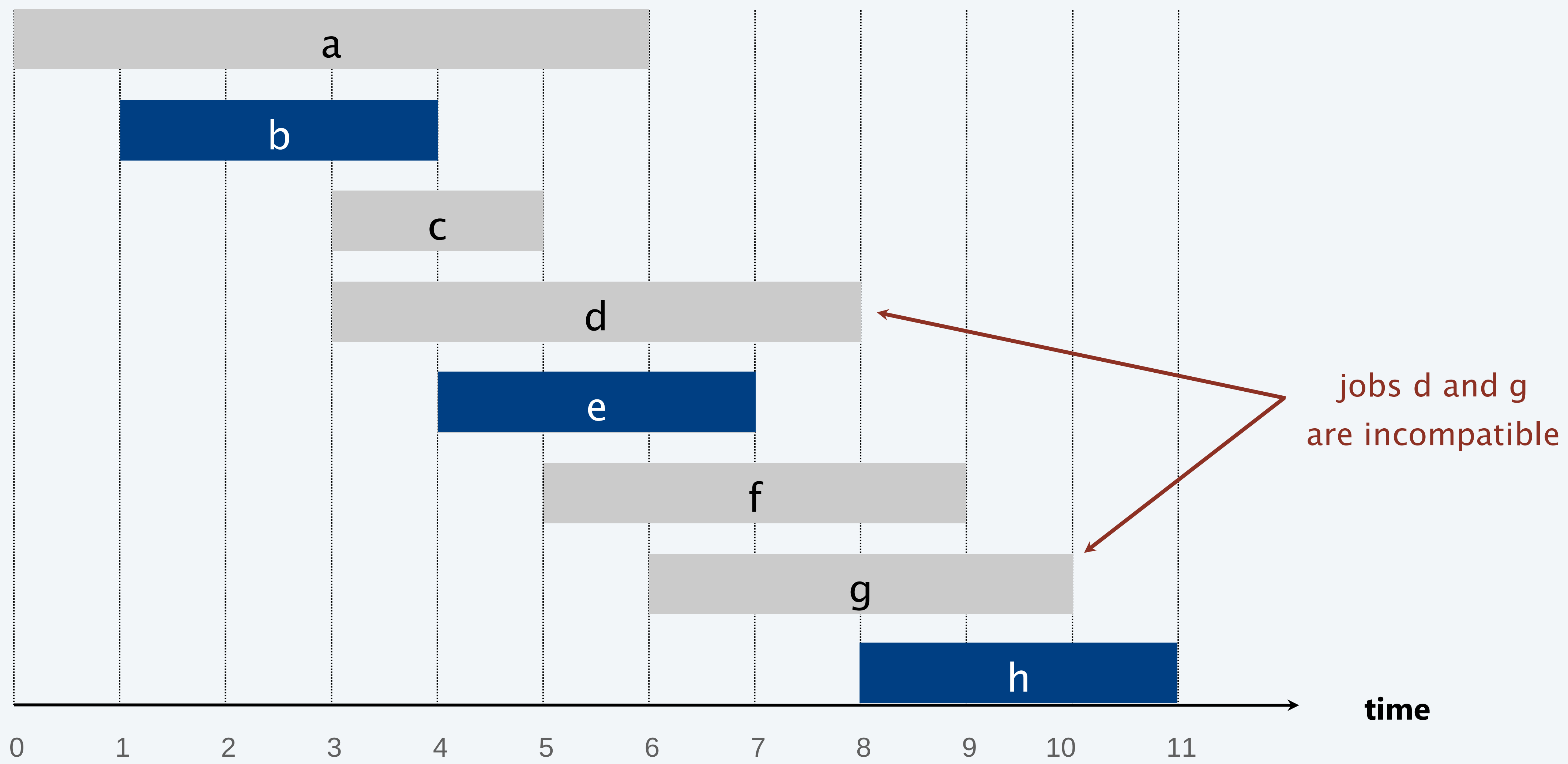
Chapter 4

Interval scheduling

Slides by Kevin Wayne.

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.



Interval scheduling

- Input:
  - A set of n intervals $I_1,\ldots,I_n$
  - interval I_i has starting time s_i and finish time f_i
- Feasible solution:
  - A subset S of the intervals that are mutually compatible, i.e. for each $I_i,I_j \in S$, I_i does not overlap with I_j
- Measure (to maximize):
  - number of scheduled intervals, i.e. cardinality of S

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: 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 $f_1 \le f_2 \le \ldots \le f_n$.

$S \leftarrow \varnothing$. ⟵ set of jobs selected

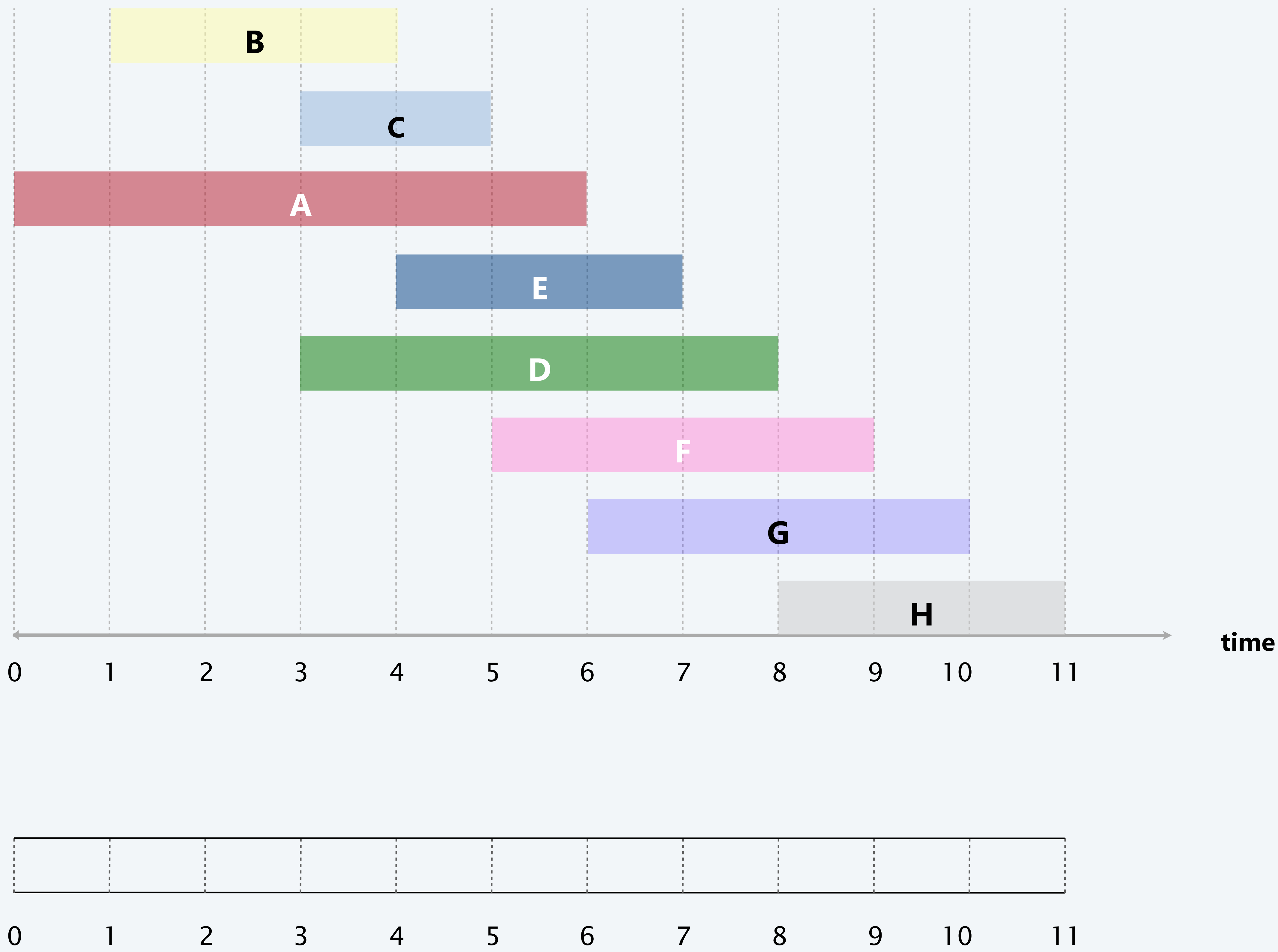
FOR $j = 1$ TO n

  IF (job j is compatible with S)

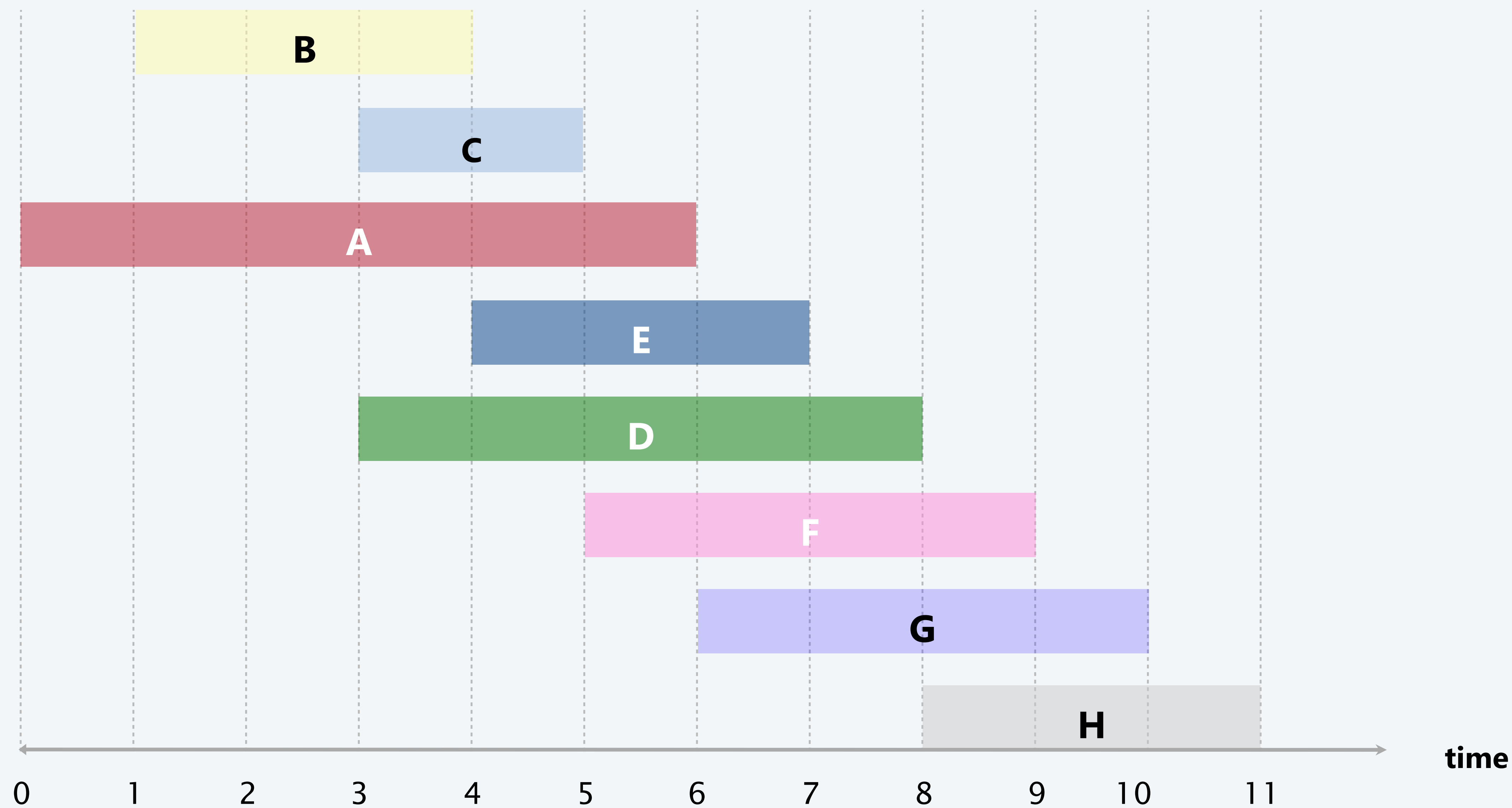
    $S \leftarrow S \cup \{ j \}$.

RETURN S.

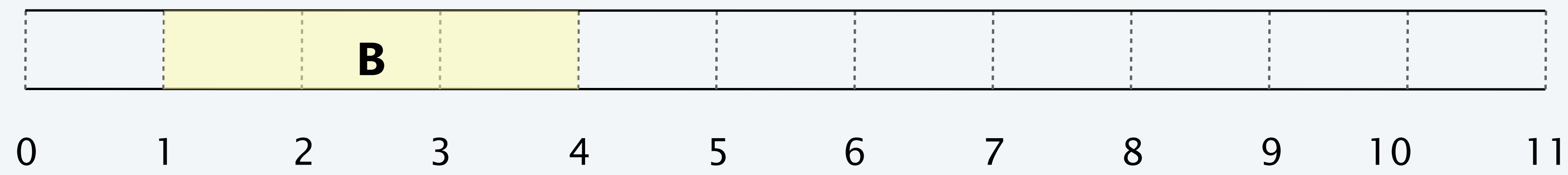
Earliest-finish-time-first algorithm demo



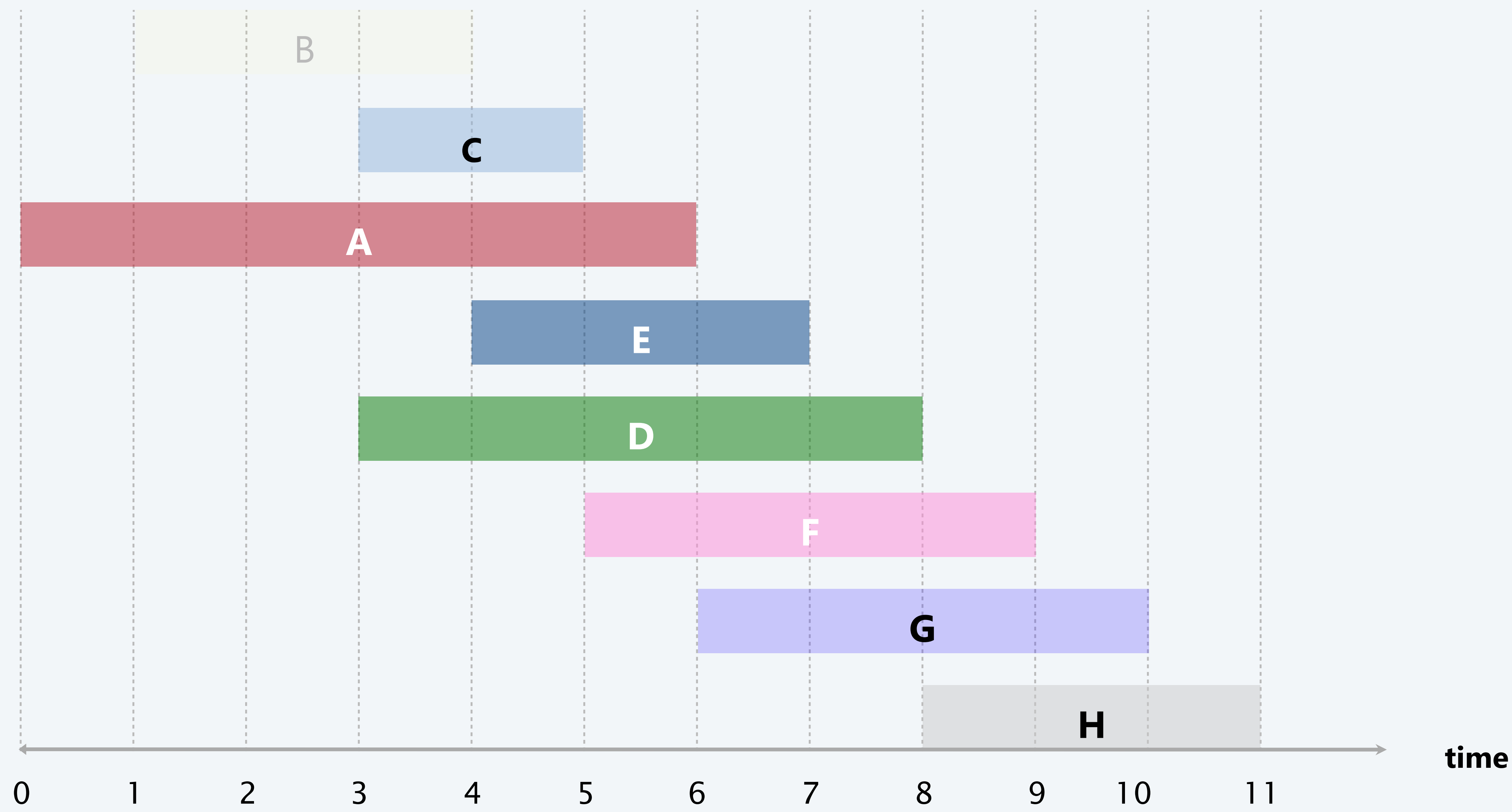
Earliest-finish-time-first algorithm demo

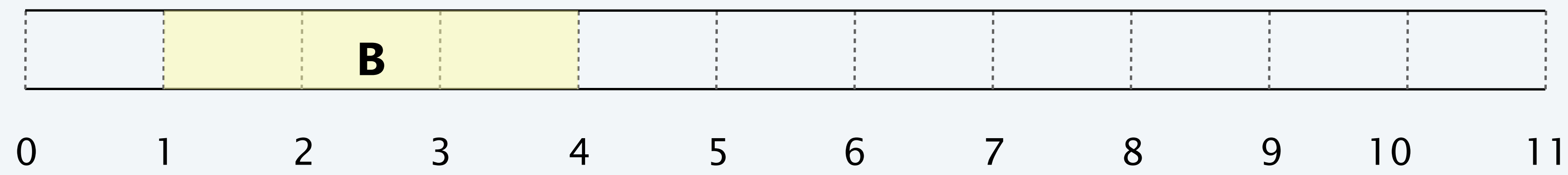


job B is compatible (add to schedule)

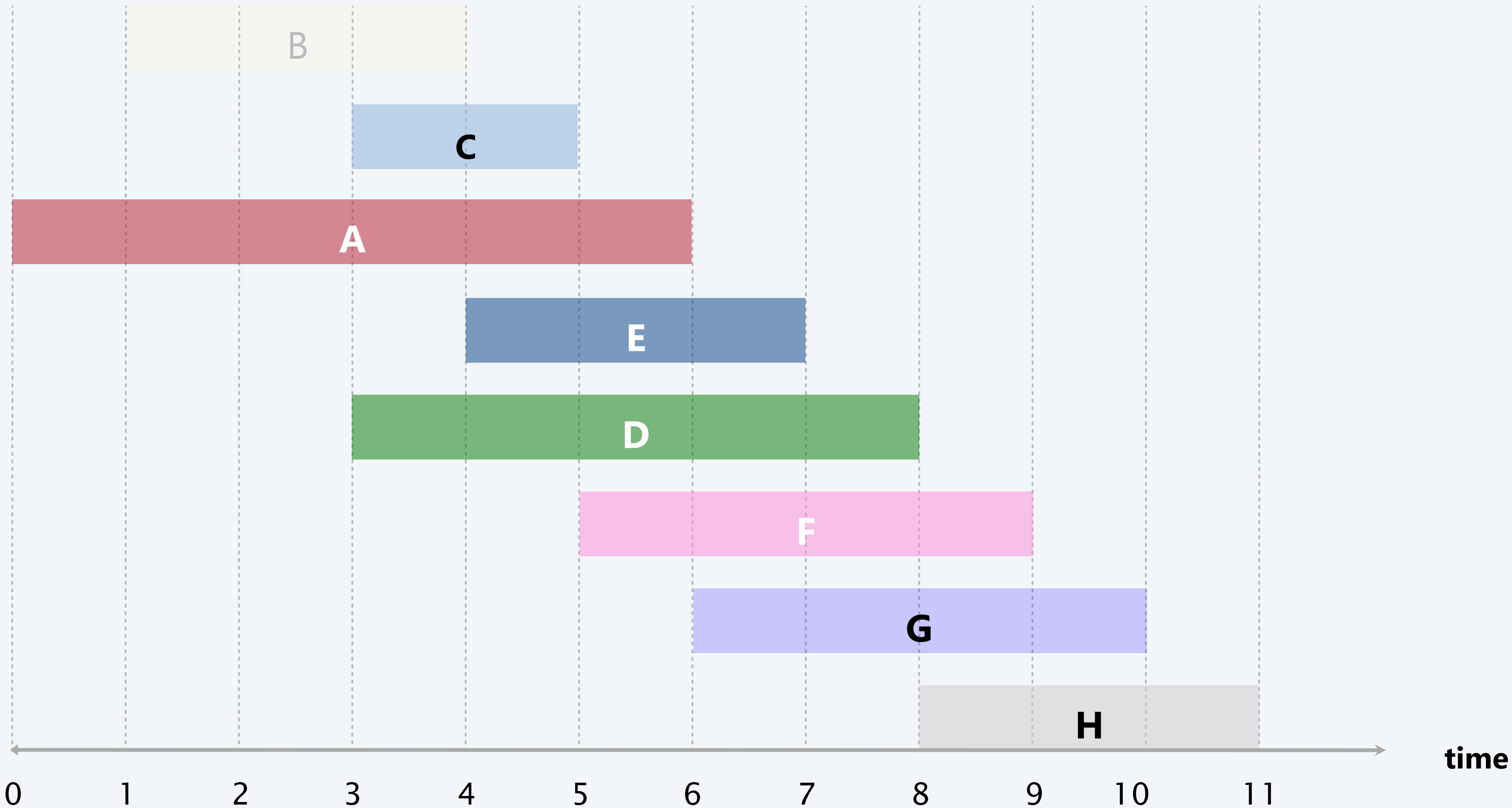


Earliest-finish-time-first algorithm demo

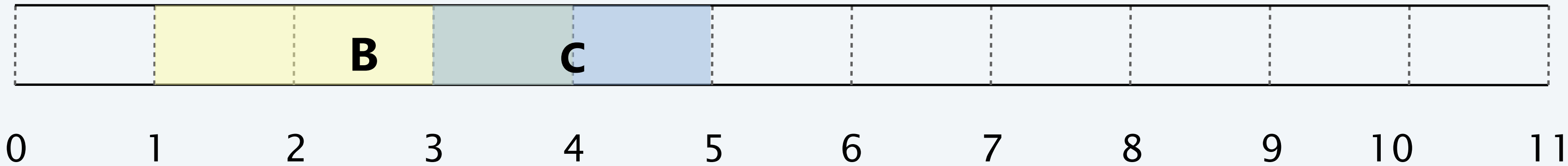




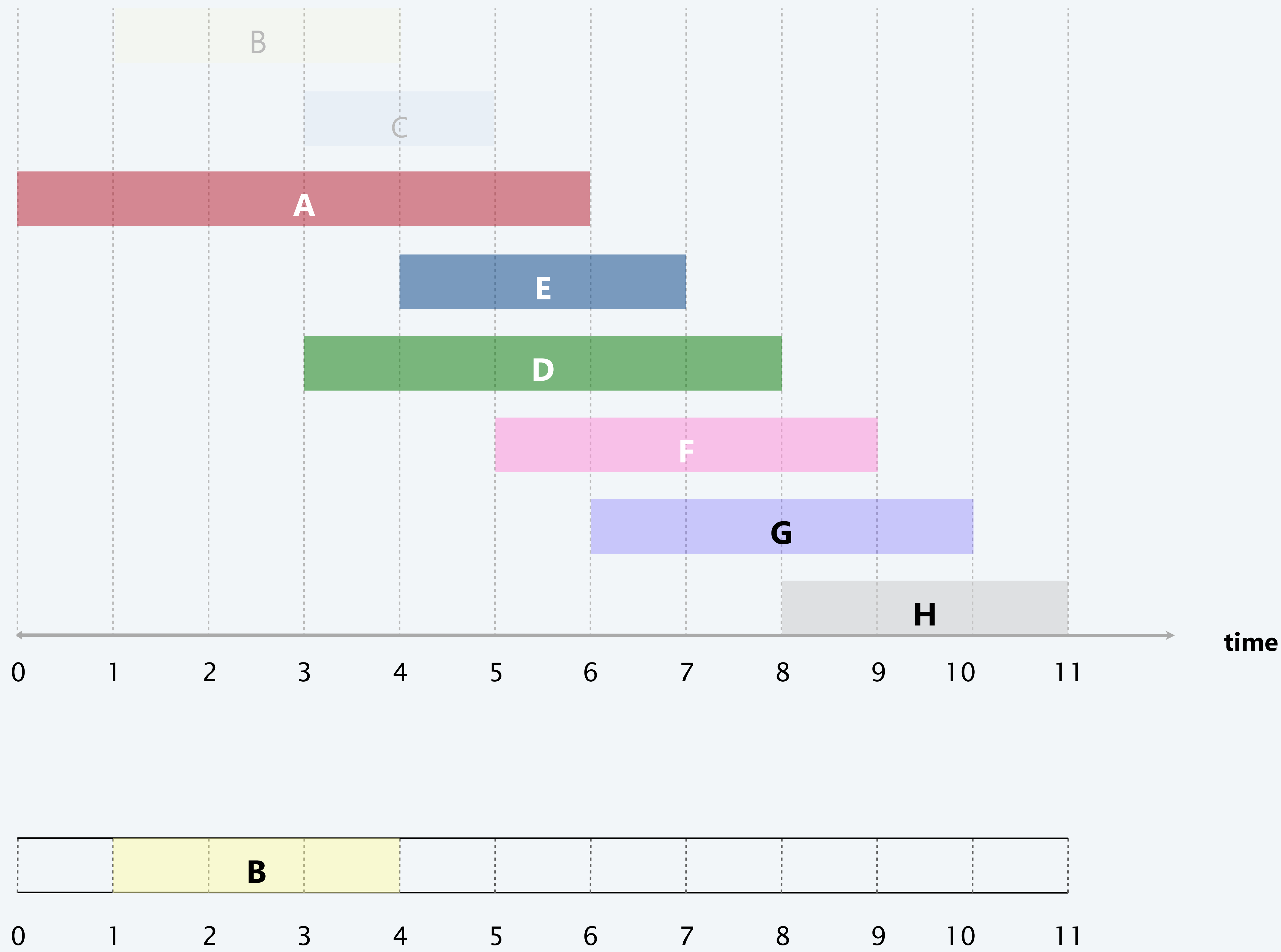
Earliest-finish-time-first algorithm demo



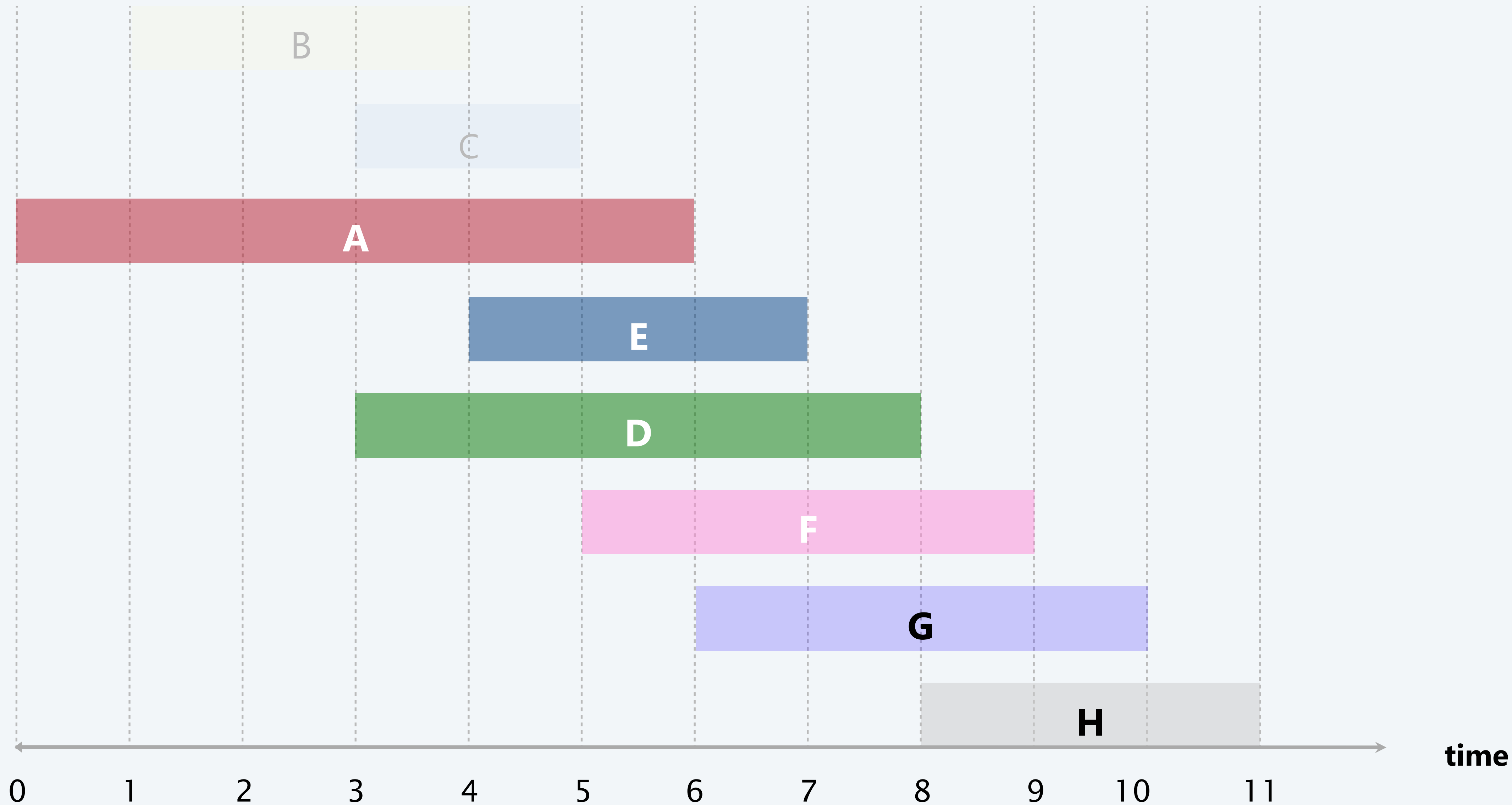
job C is incompatible (do not add to schedule)



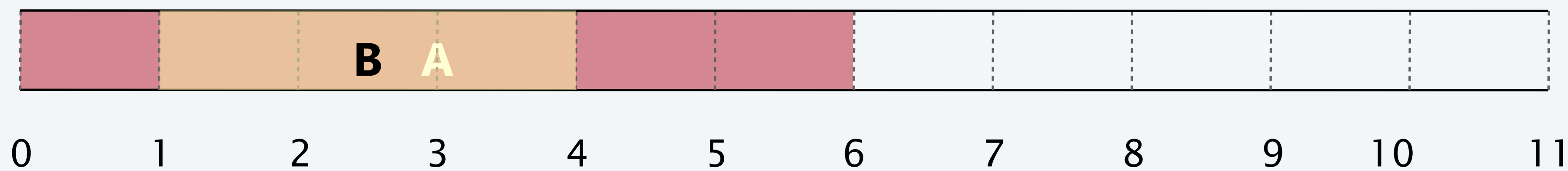
Earliest-finish-time-first algorithm demo



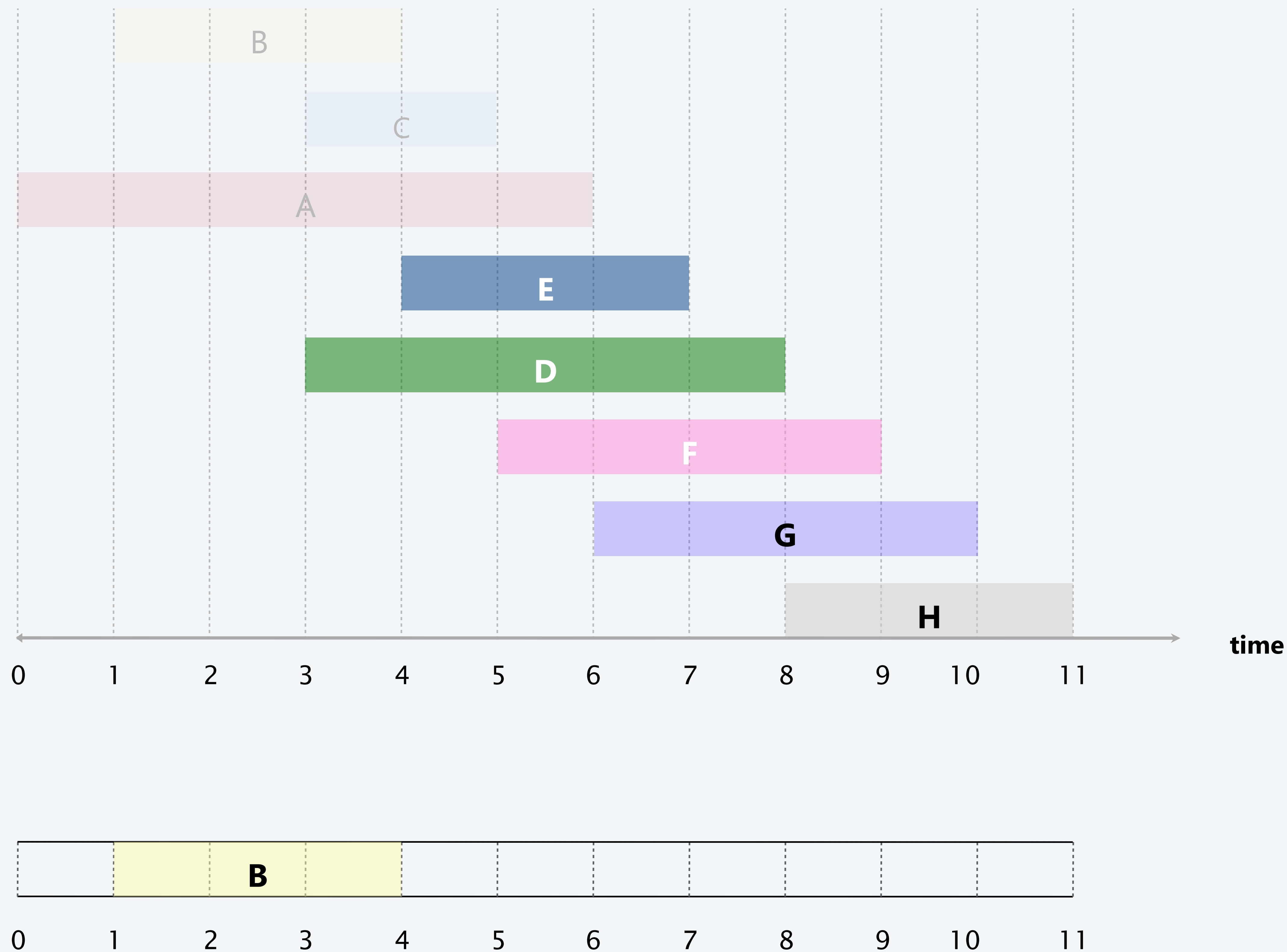
Earliest-finish-time-first algorithm demo



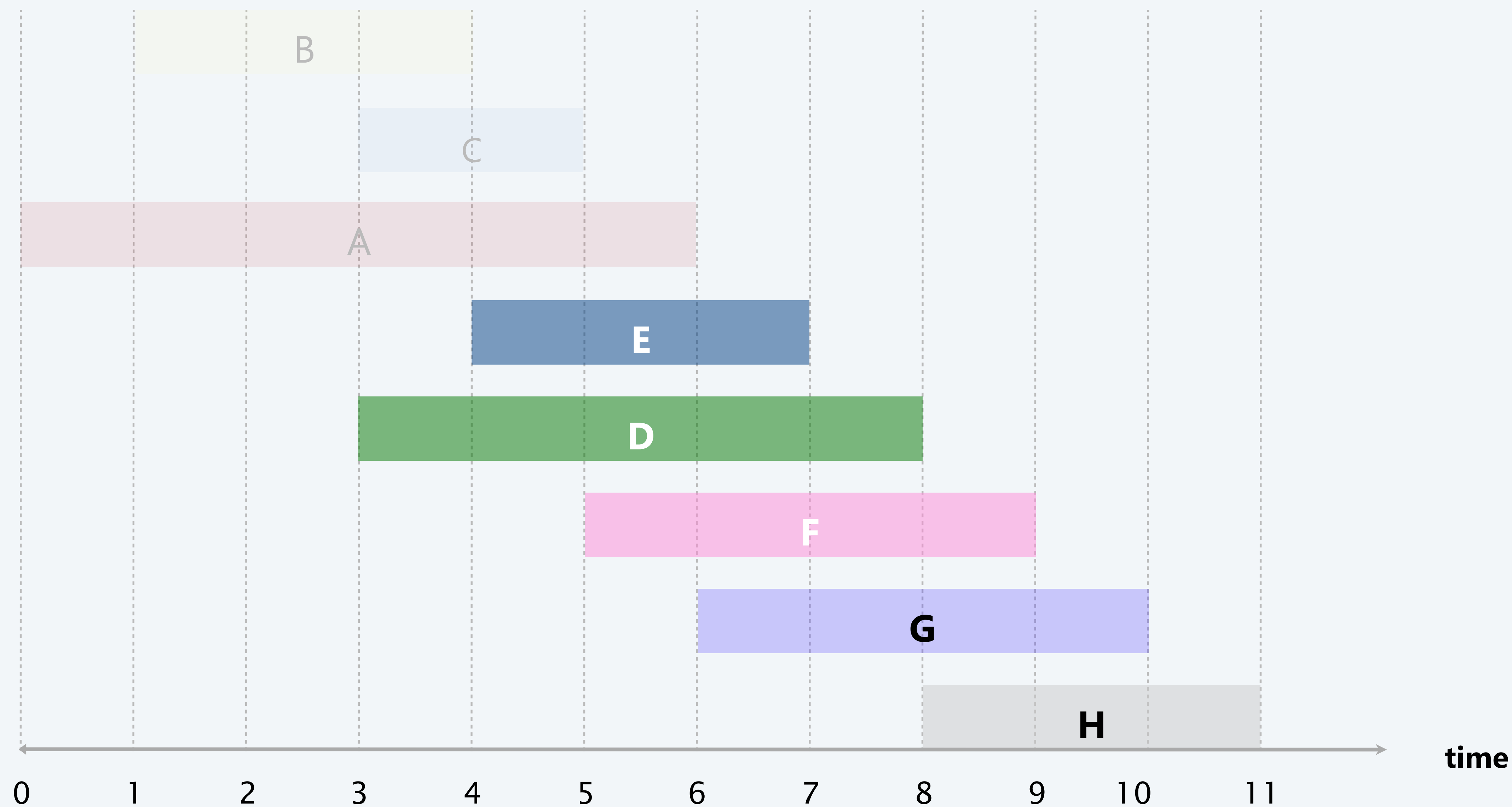
job A is incompatible (do not add to schedule)



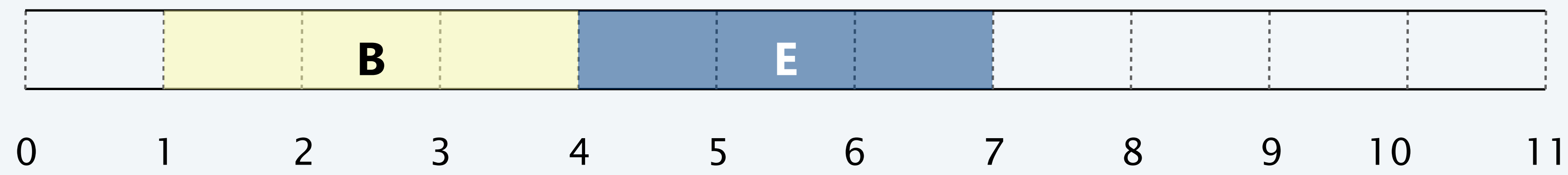
Earliest-finish-time-first algorithm demo



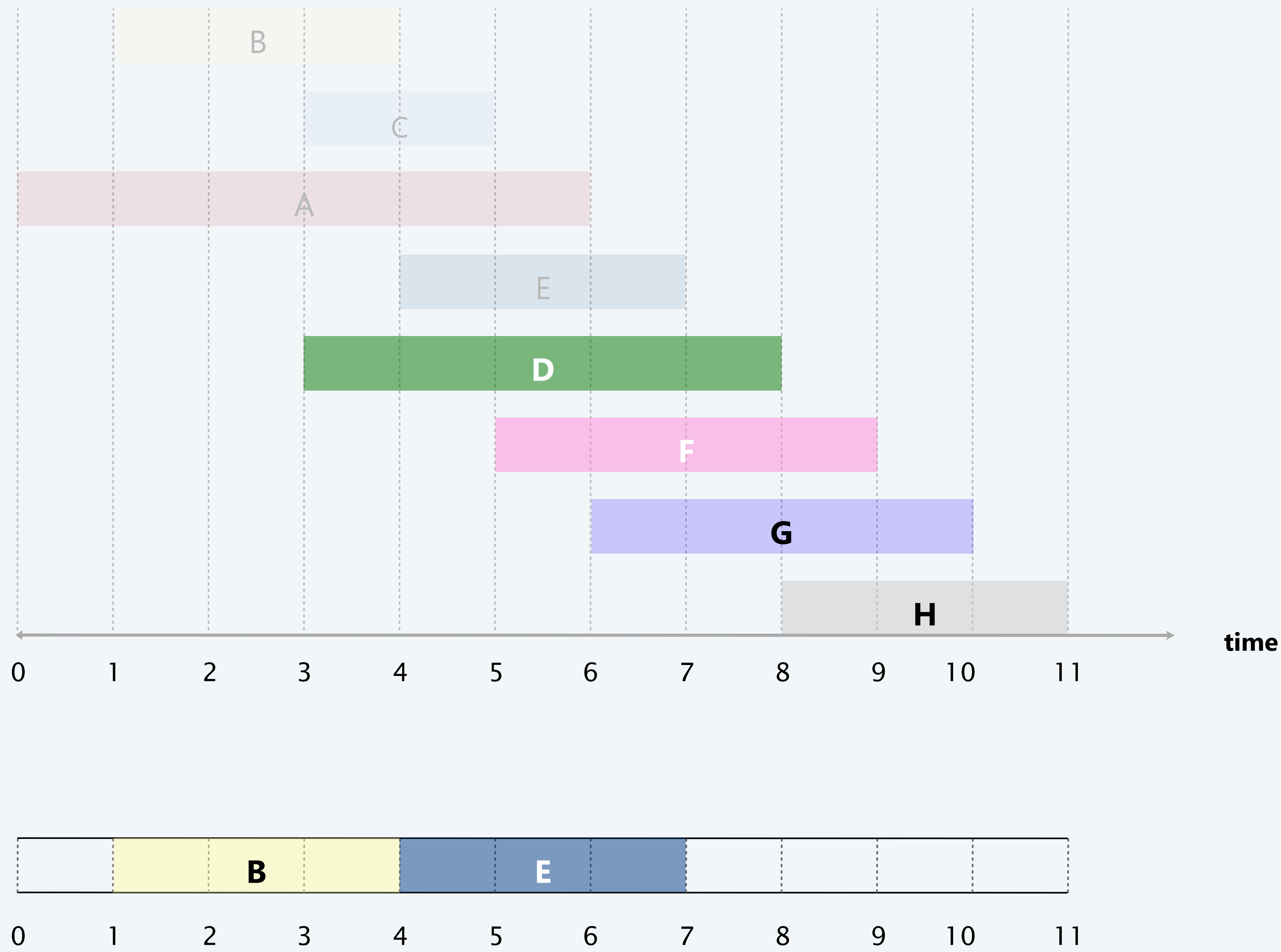
Earliest-finish-time-first algorithm demo



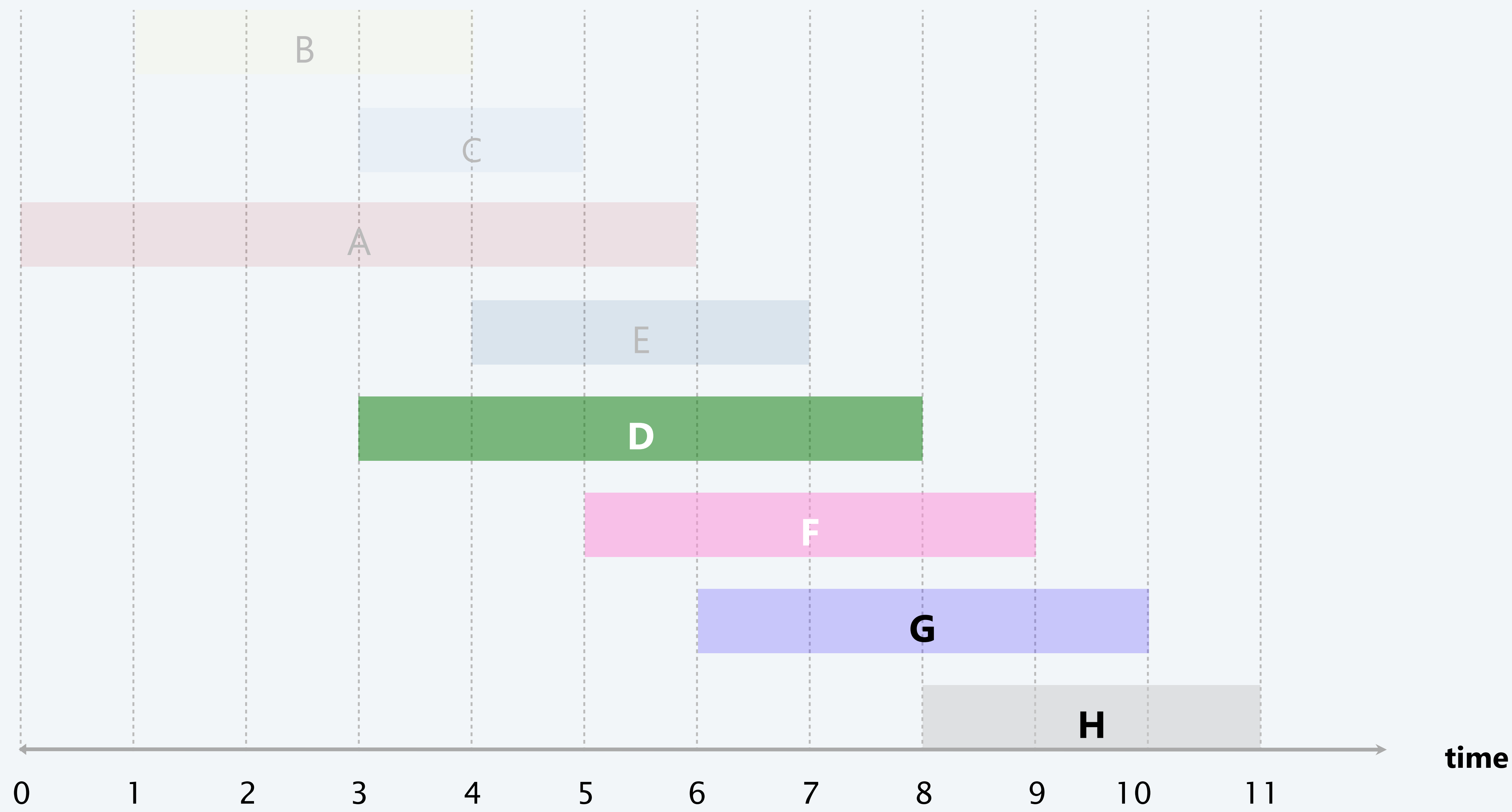
job E is compatible (add to schedule)



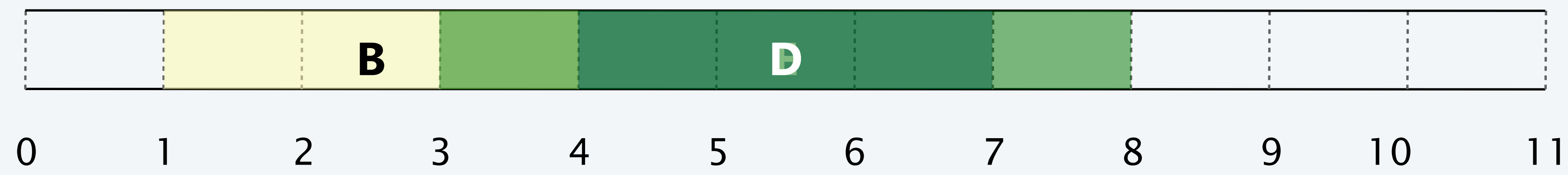
Earliest-finish-time-first algorithm demo



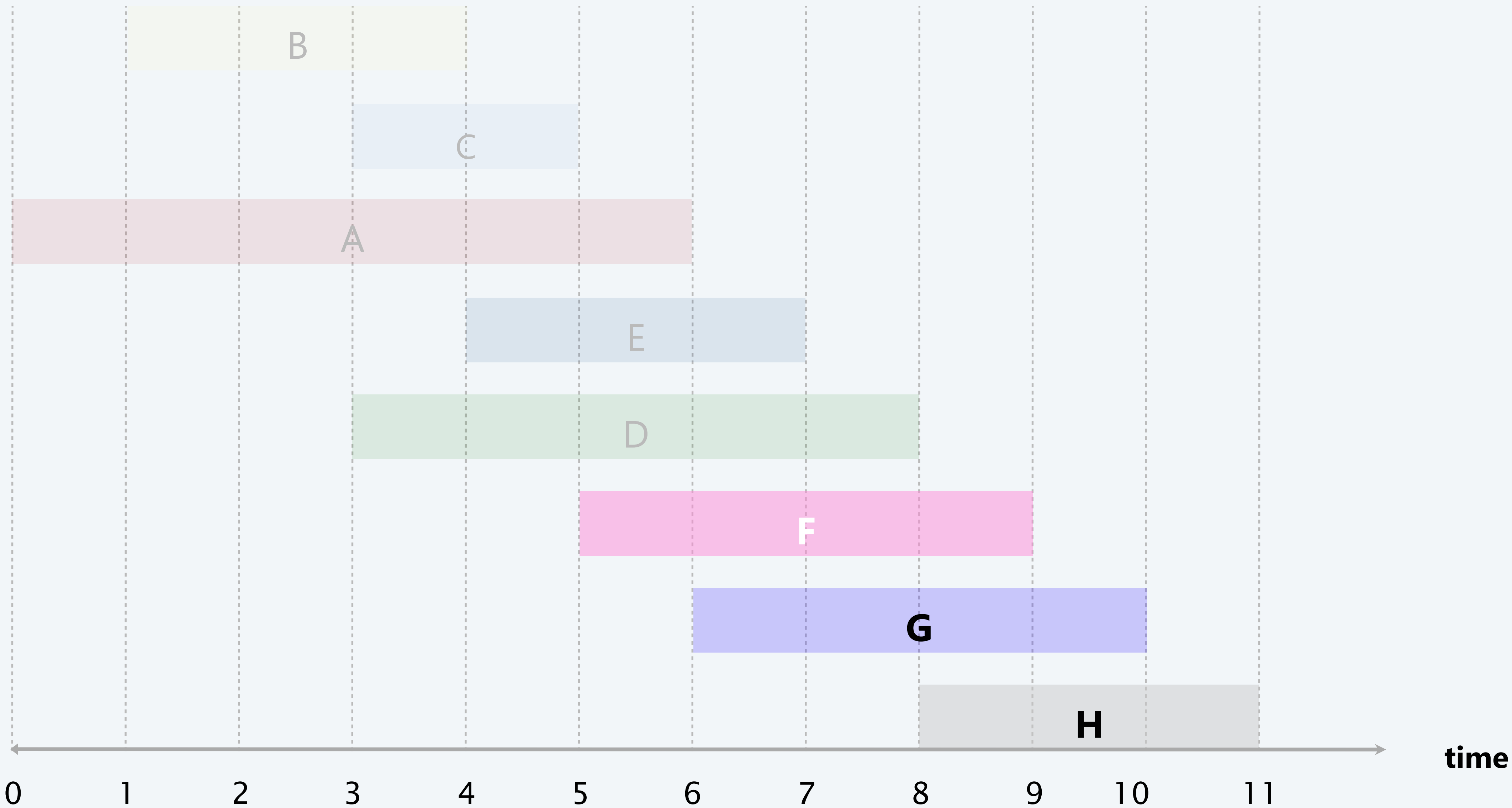
Earliest-finish-time-first algorithm demo

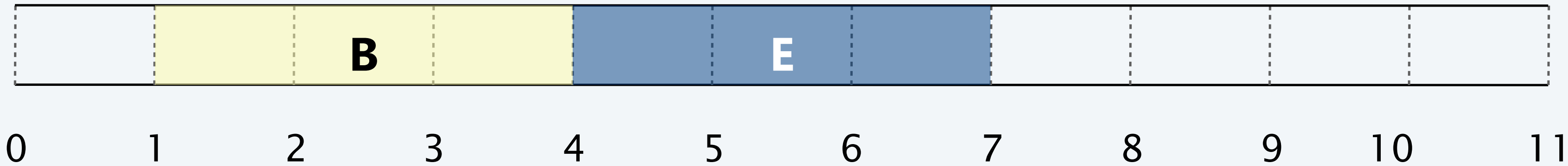


job D is incompatible (do not add to schedule)

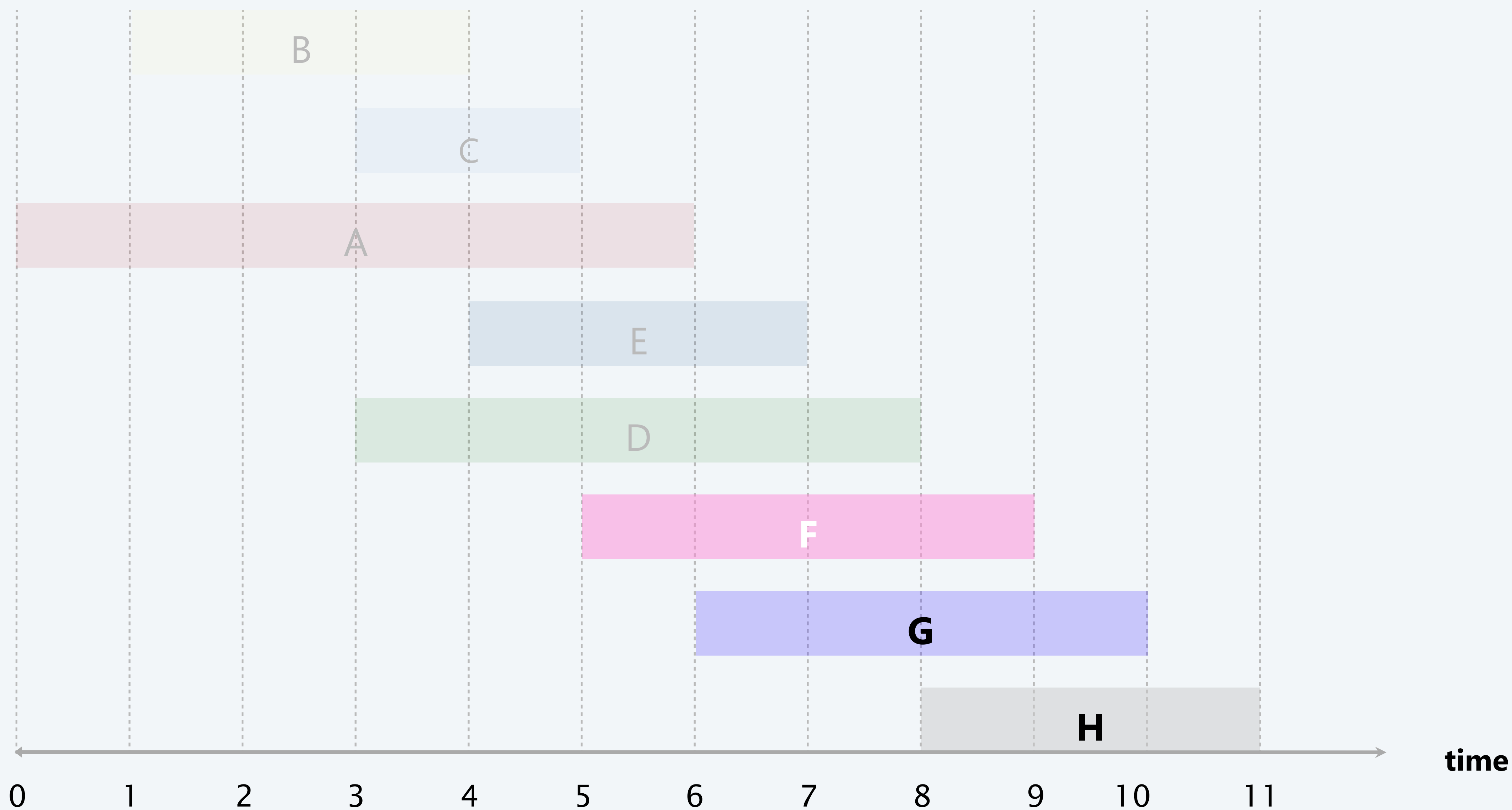


Earliest-finish-time-first algorithm demo

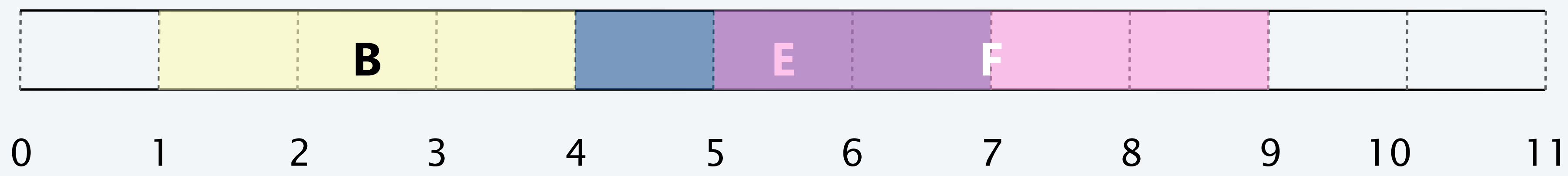




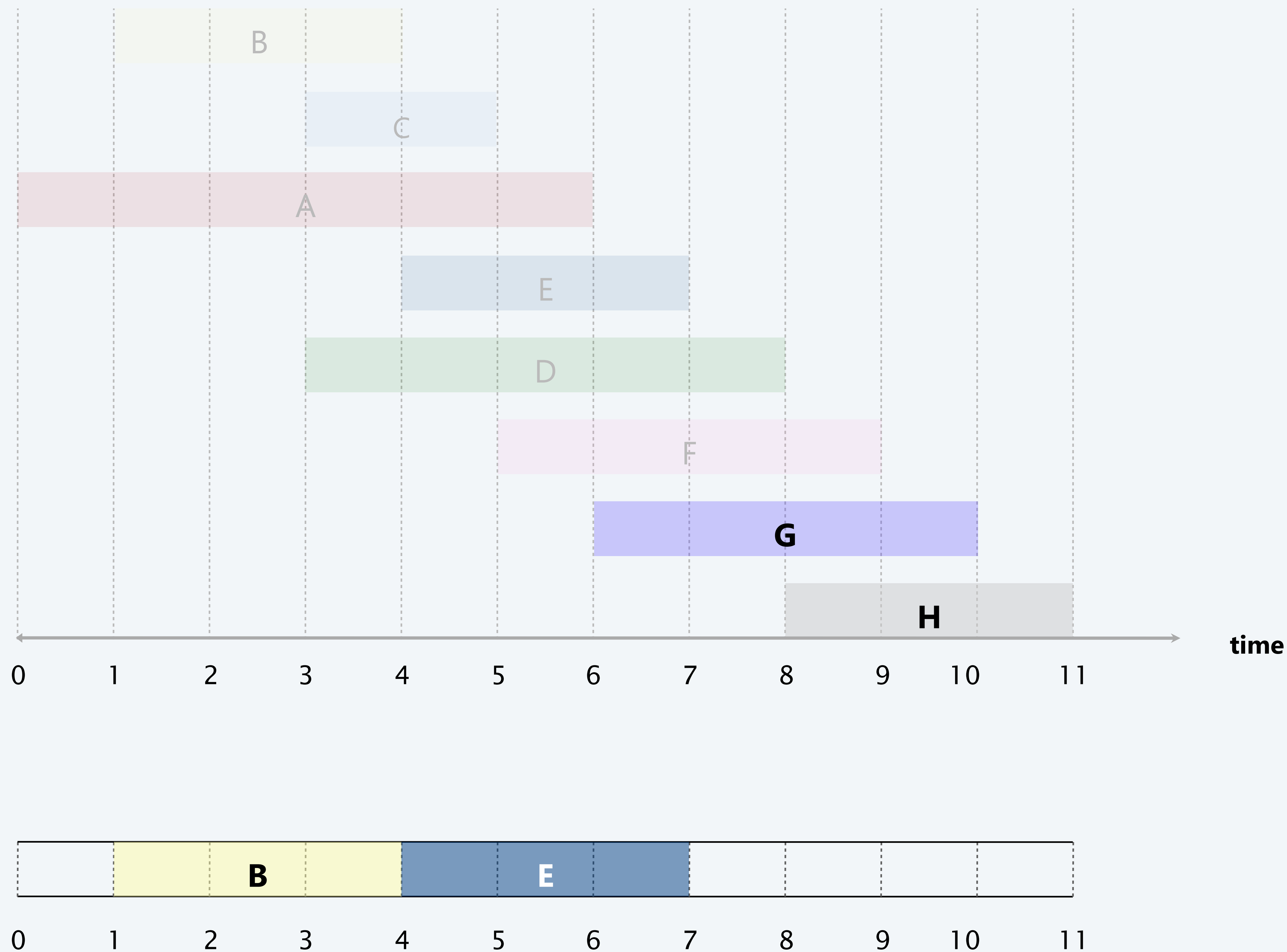
Earliest-finish-time-first algorithm demo



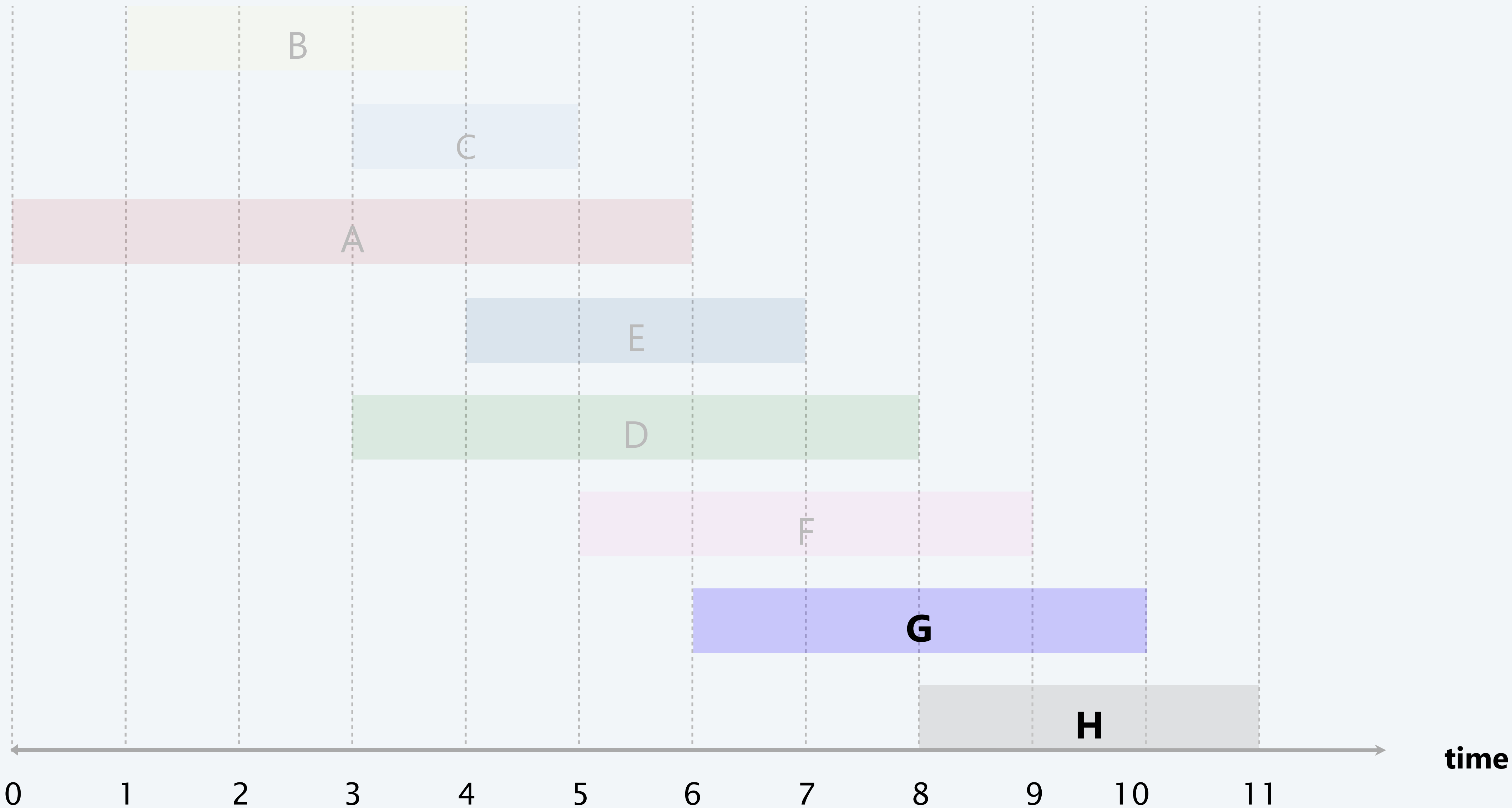
job F is incompatible (do not add to schedule)



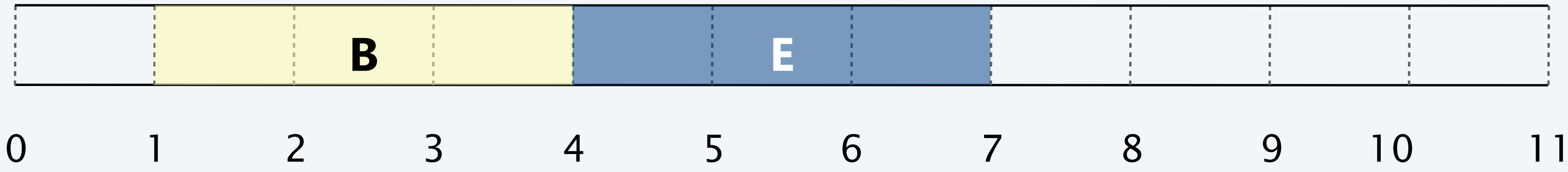
Earliest-finish-time-first algorithm demo



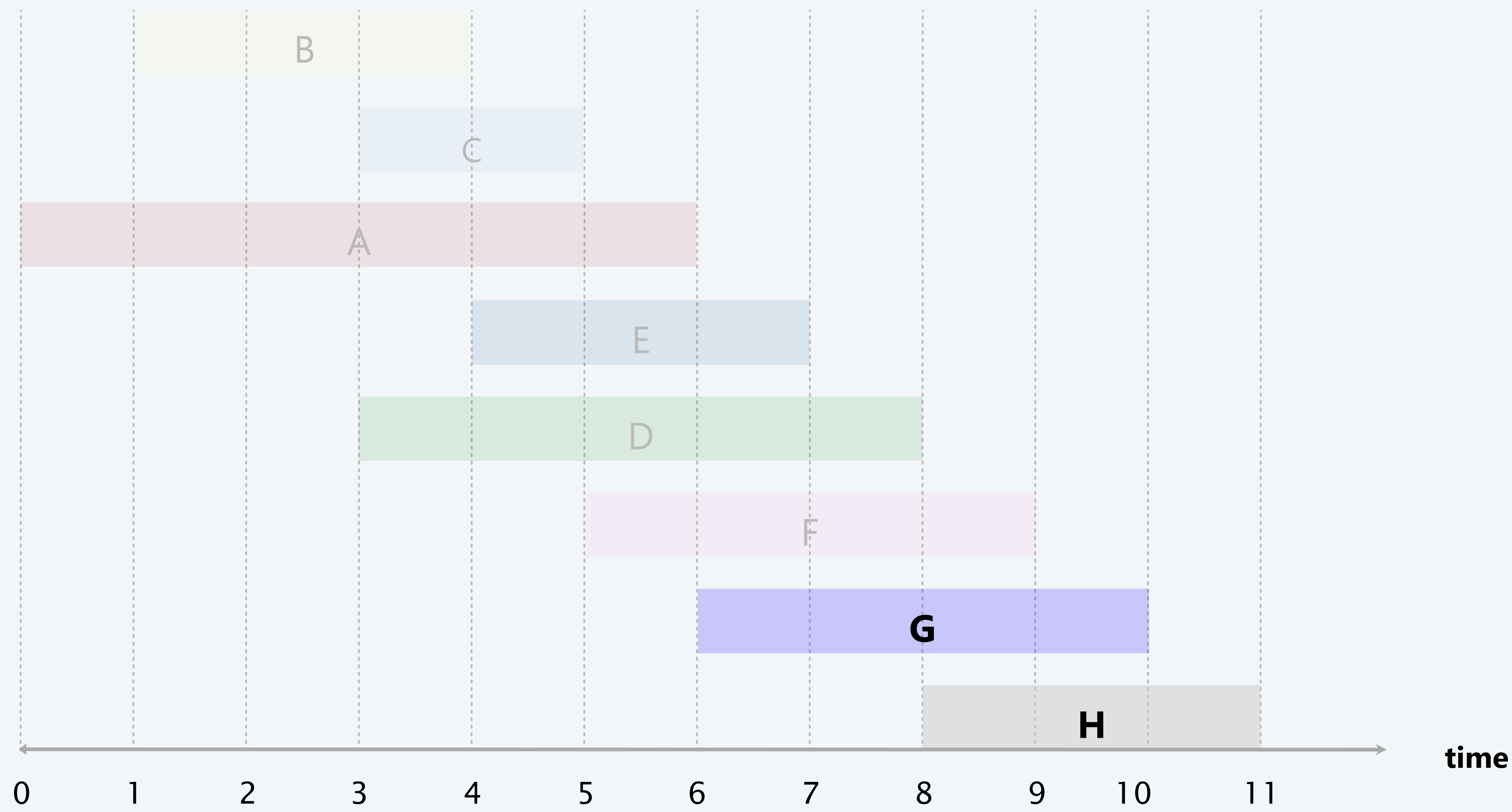
Earliest-finish-time-first algorithm demo



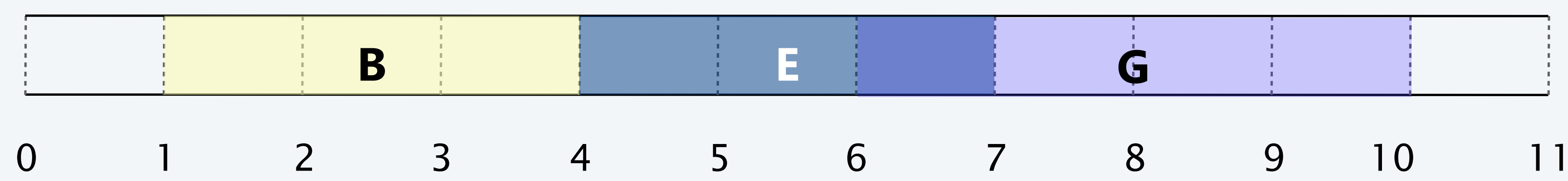
job G is incompatible (do not add to schedule)



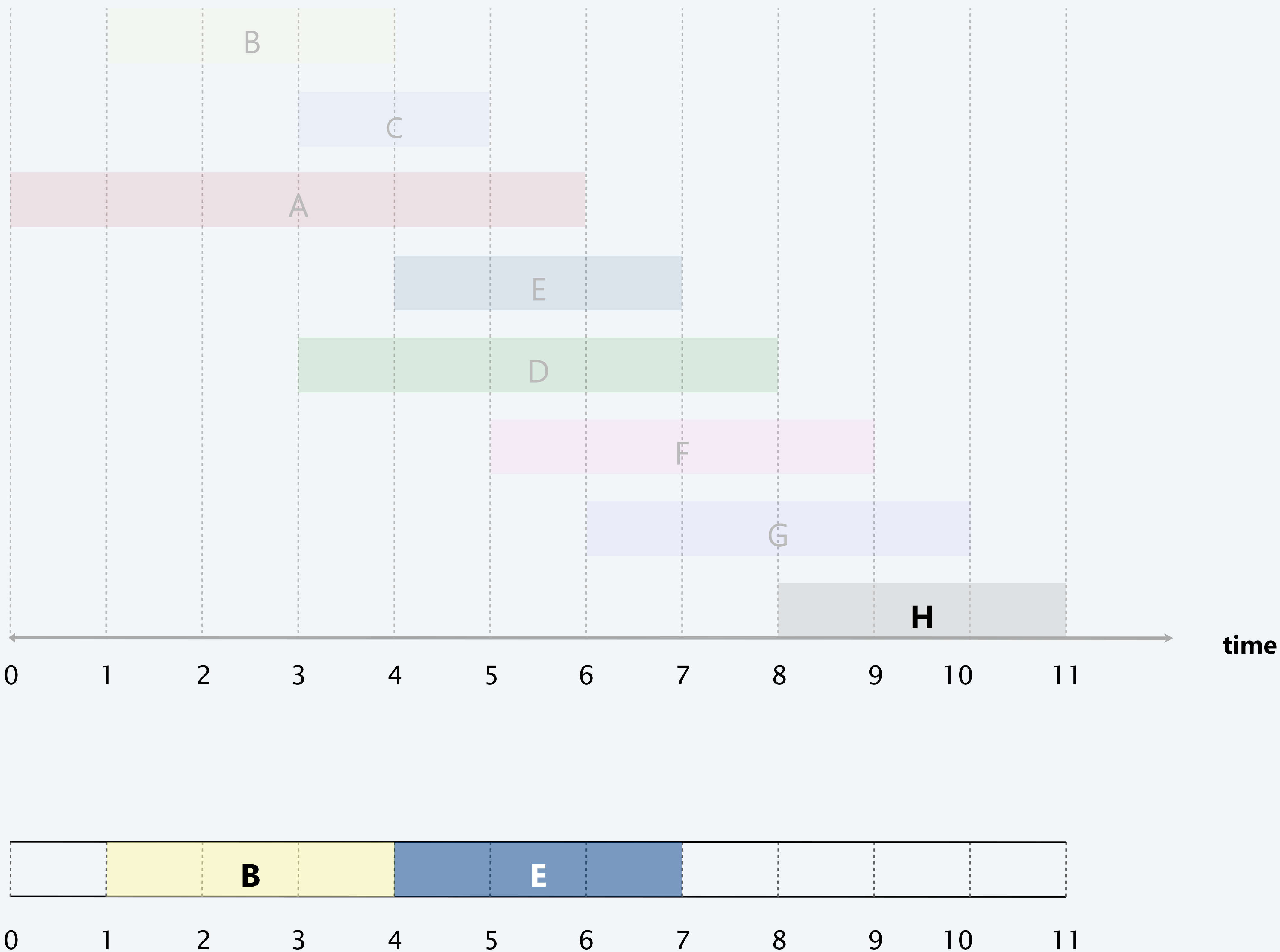
Earliest-finish-time-first algorithm demo



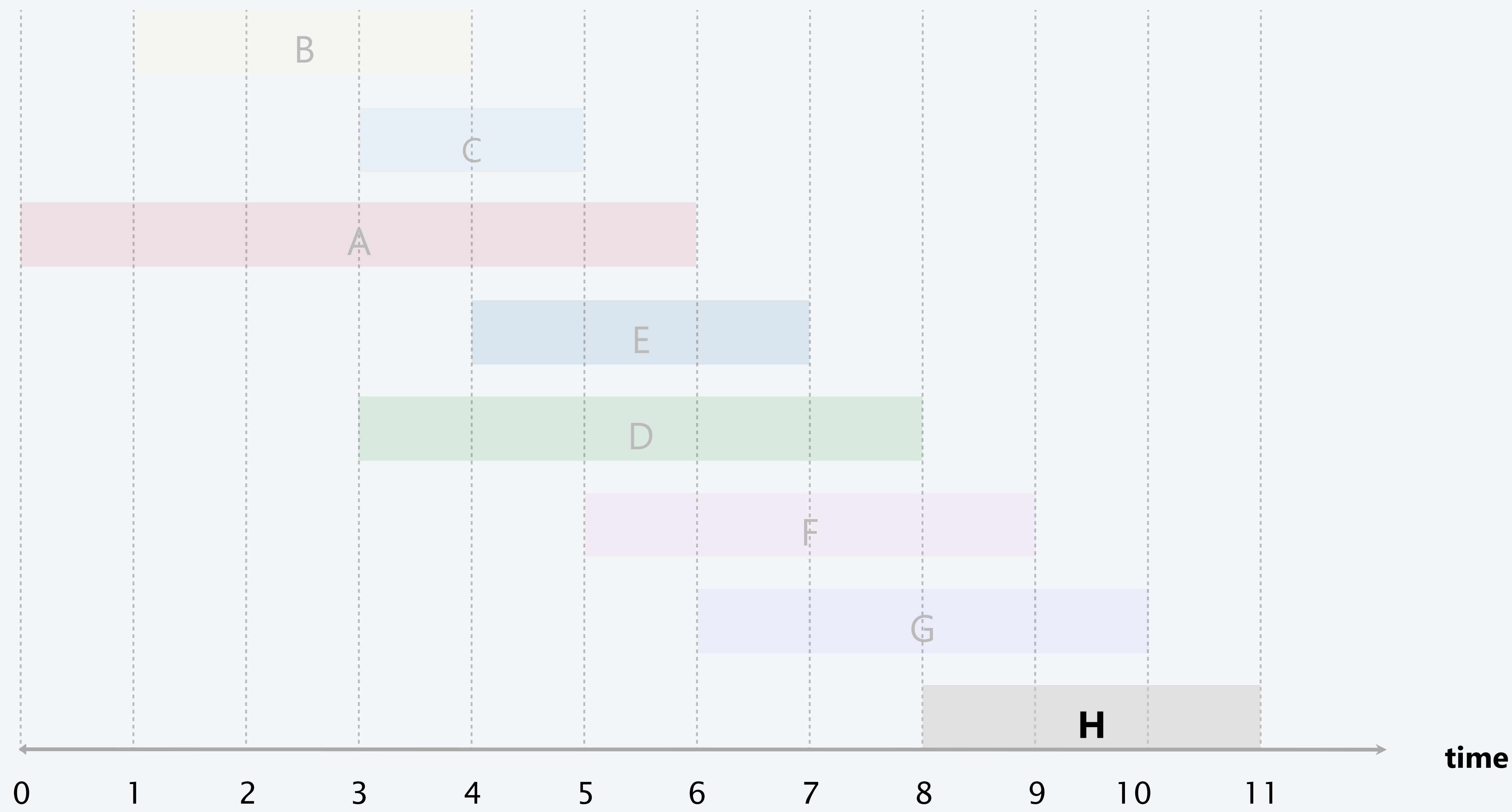
job G is incompatible (do not add to schedule)



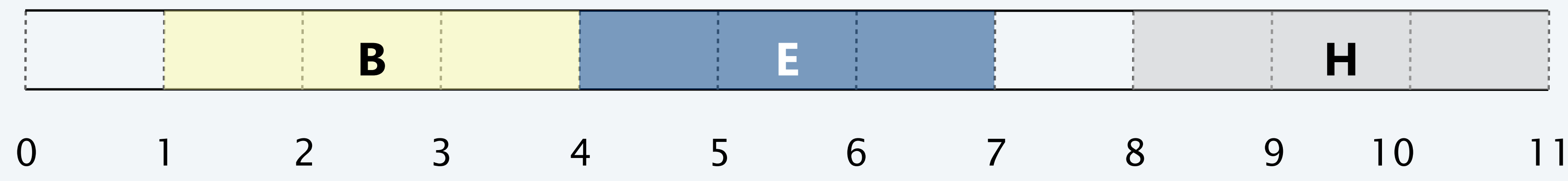
Earliest-finish-time-first algorithm demo



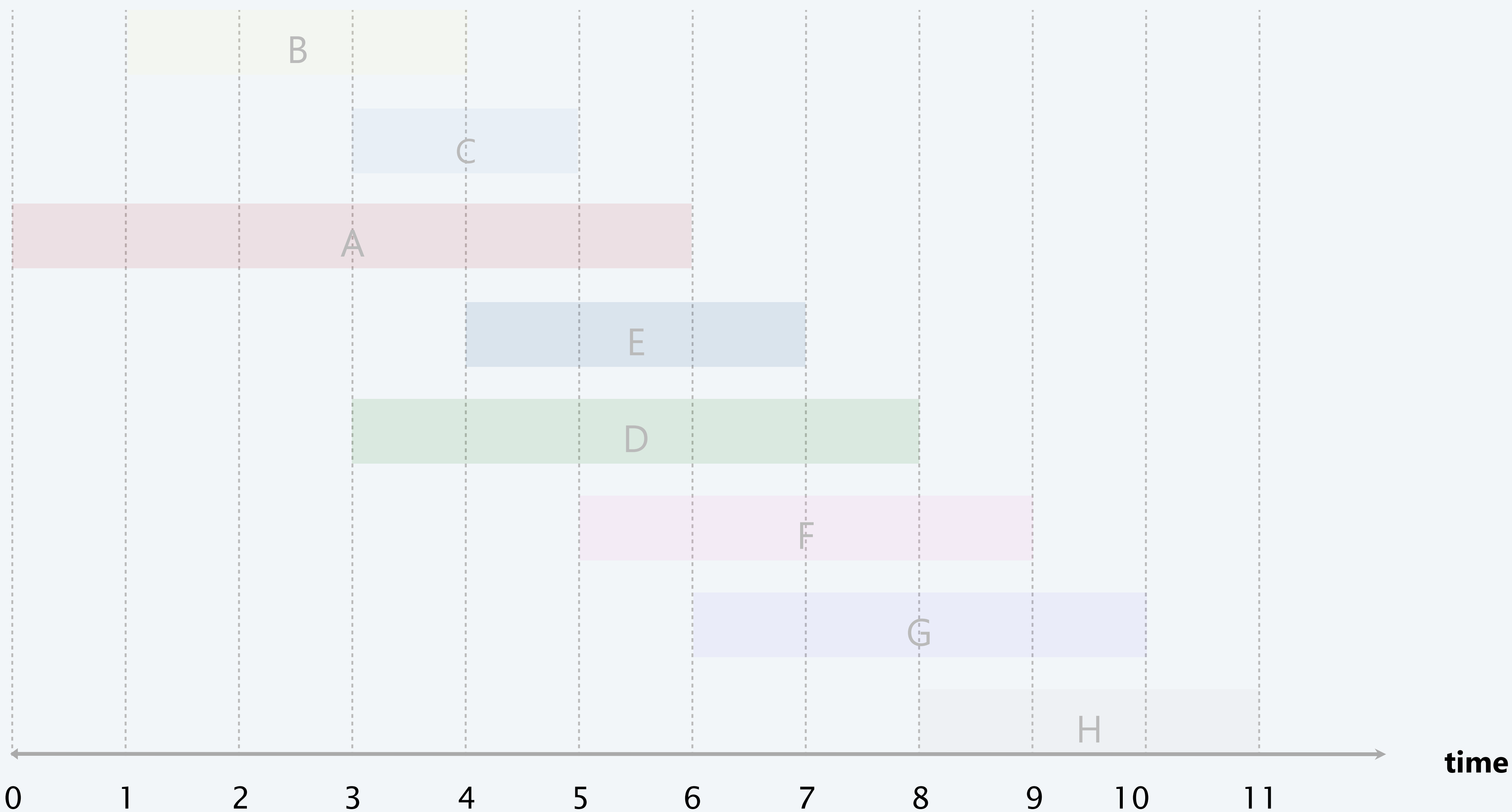
Earliest-finish-time-first algorithm demo

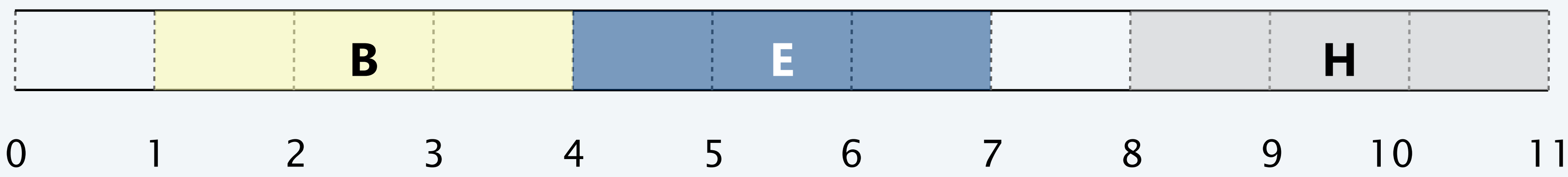


job H is compatible (add to schedule)

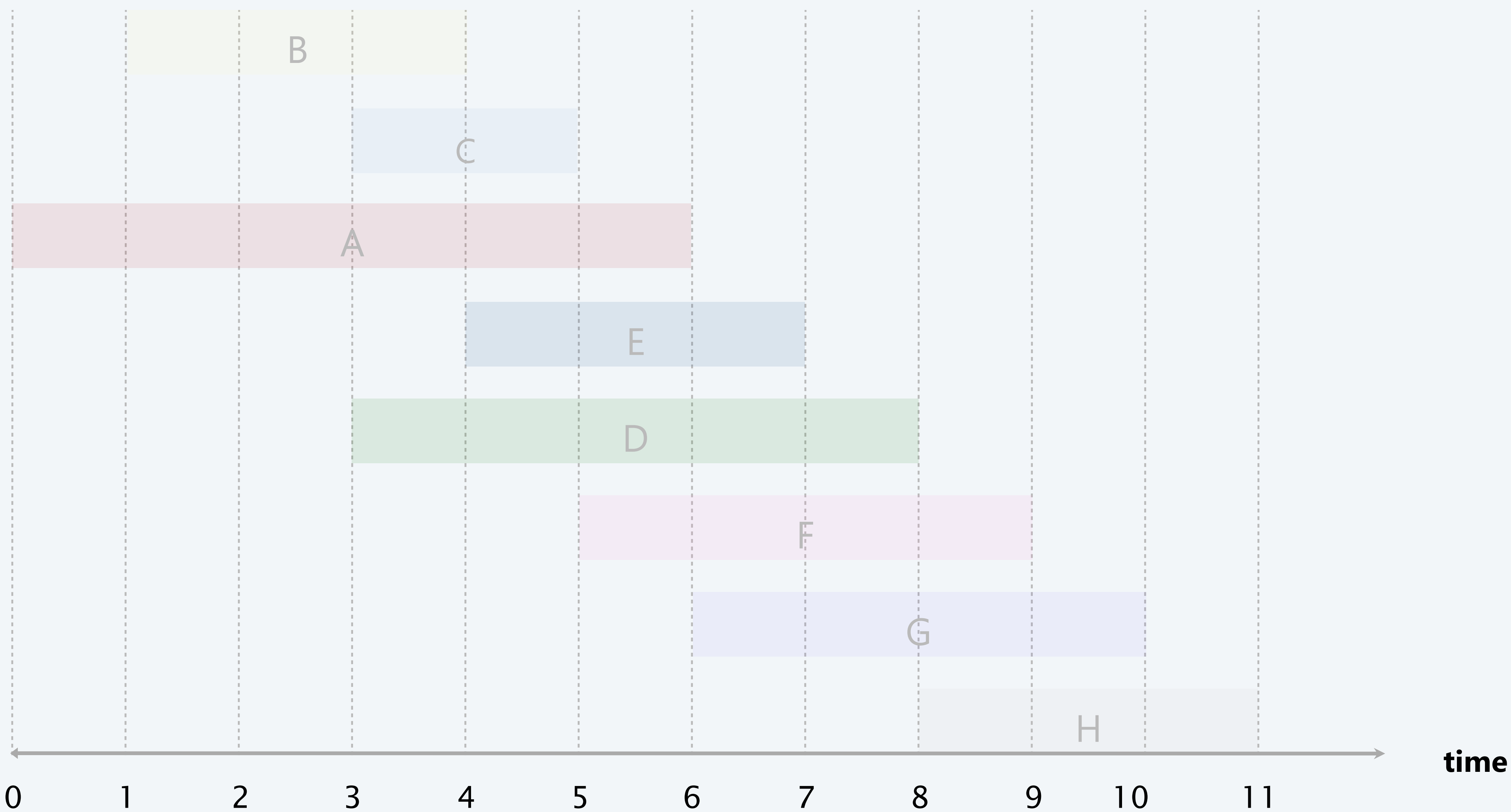


Earliest-finish-time-first algorithm demo

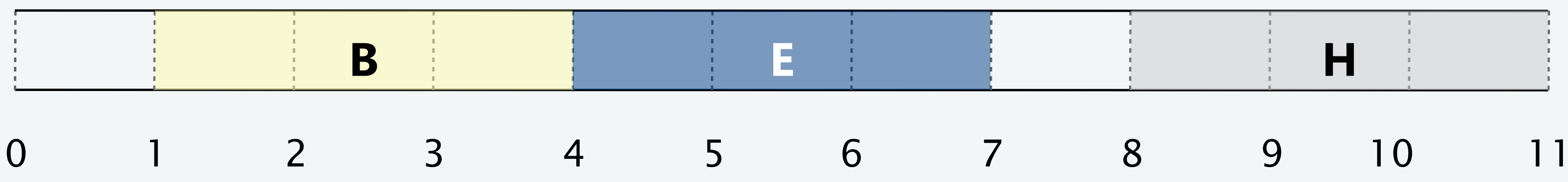




Earliest-finish-time-first algorithm demo



done (an optimal set of jobs)



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 $f_1 \leq f_2 \leq \ldots \leq f_n$.

$S \leftarrow \varnothing$. ⟵ 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.

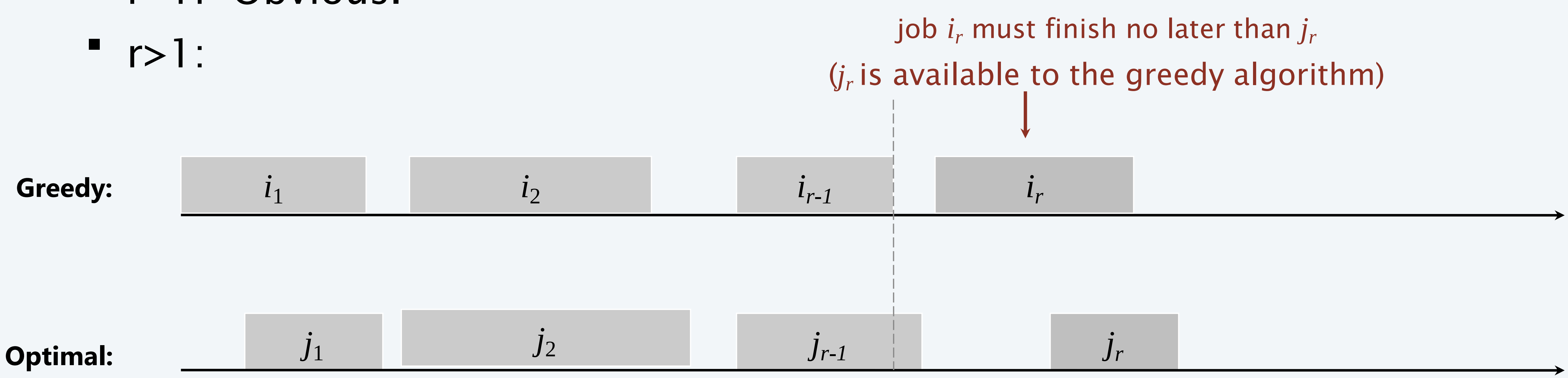
Interval scheduling: analysis of earliest-finish-time-first algorithm

Let i_1, i_2, ... i_k be set of jobs selected by greedy (ordered by finish times).
Let j_1, j_2, ... j_m be set of jobs in an optimal solution (ordered by finish times)

denote by $f(i_r)$ the finish time of job i_r .

Lemma. For every r=1,2...,k, we have $f(i_r) \le f(j_r)$.
Pf. [by induction]

- r=1: Obvious.
- r>1:

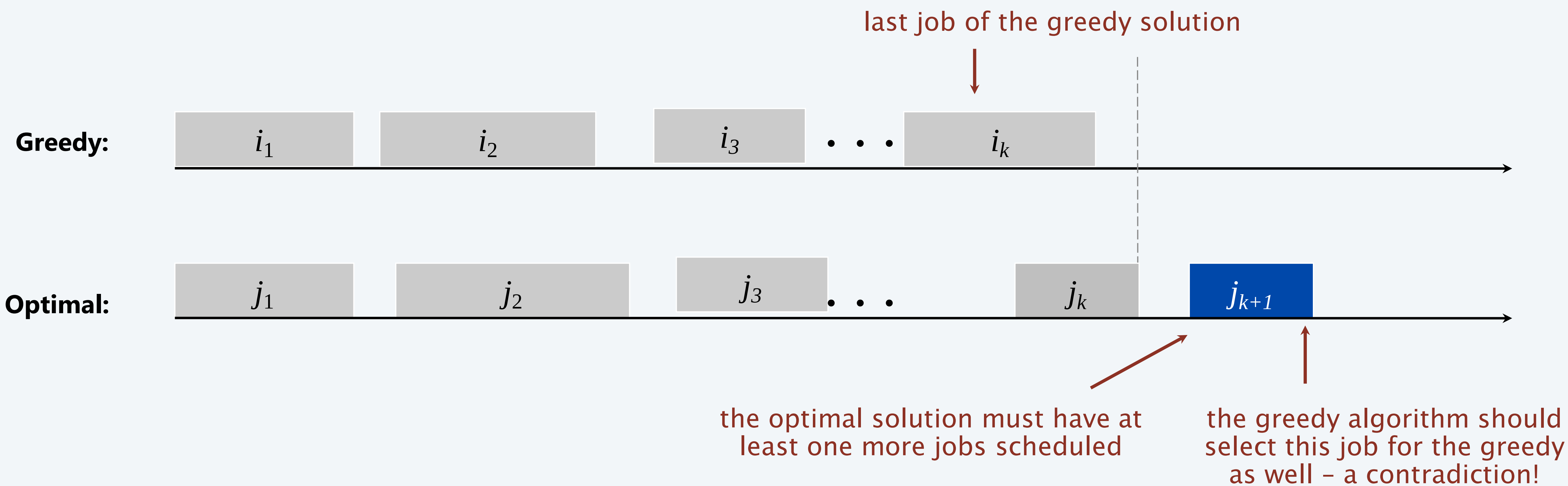


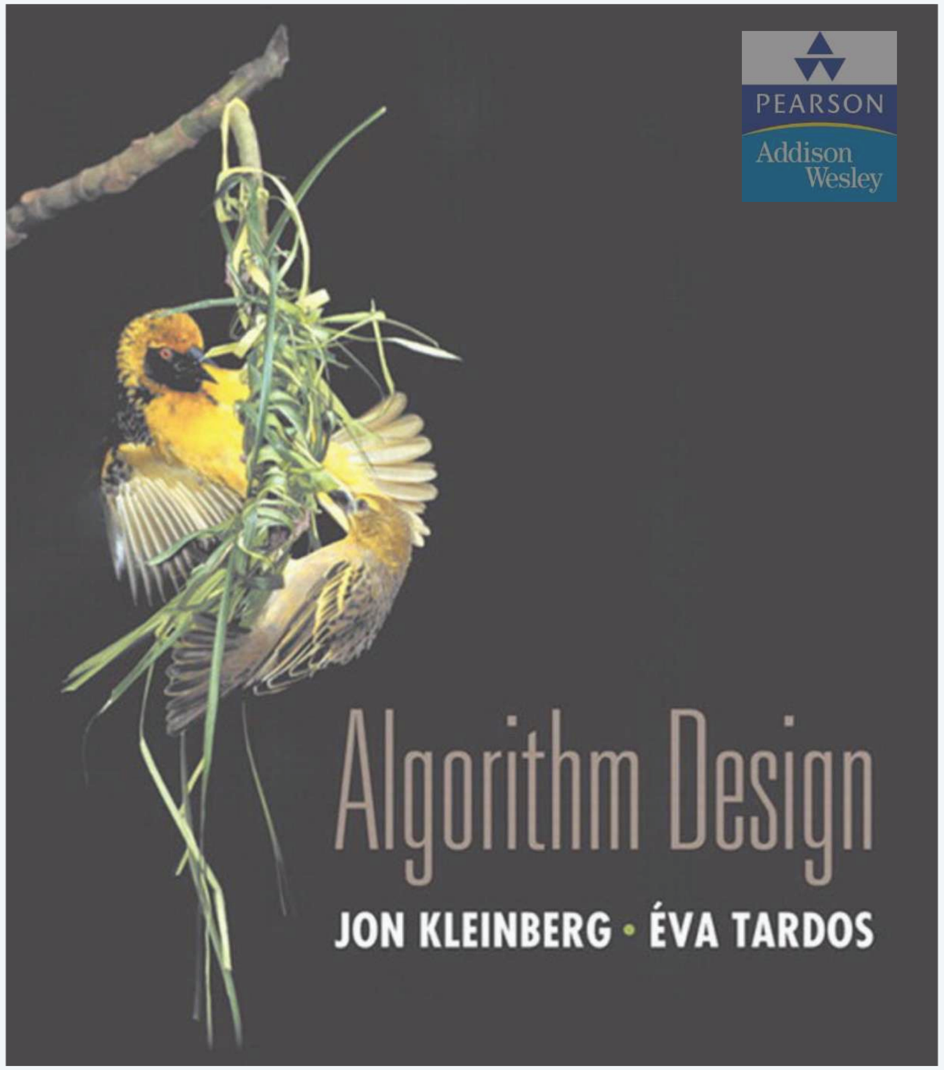
Interval scheduling: analysis of earliest-finish-time-first algorithm

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

Pf. [by contradiction]

- Let i_1, i_2, ... i_k be set of jobs selected by greedy (ordered by finish times).
- Let j_1, j_2, ... j_m be set of jobs in an optimal solution (ordered by finish times)
- Assume greedy is not optimal
- then $m>k$





SECTION 4.1

4. GREEDY ALGORITHMS I

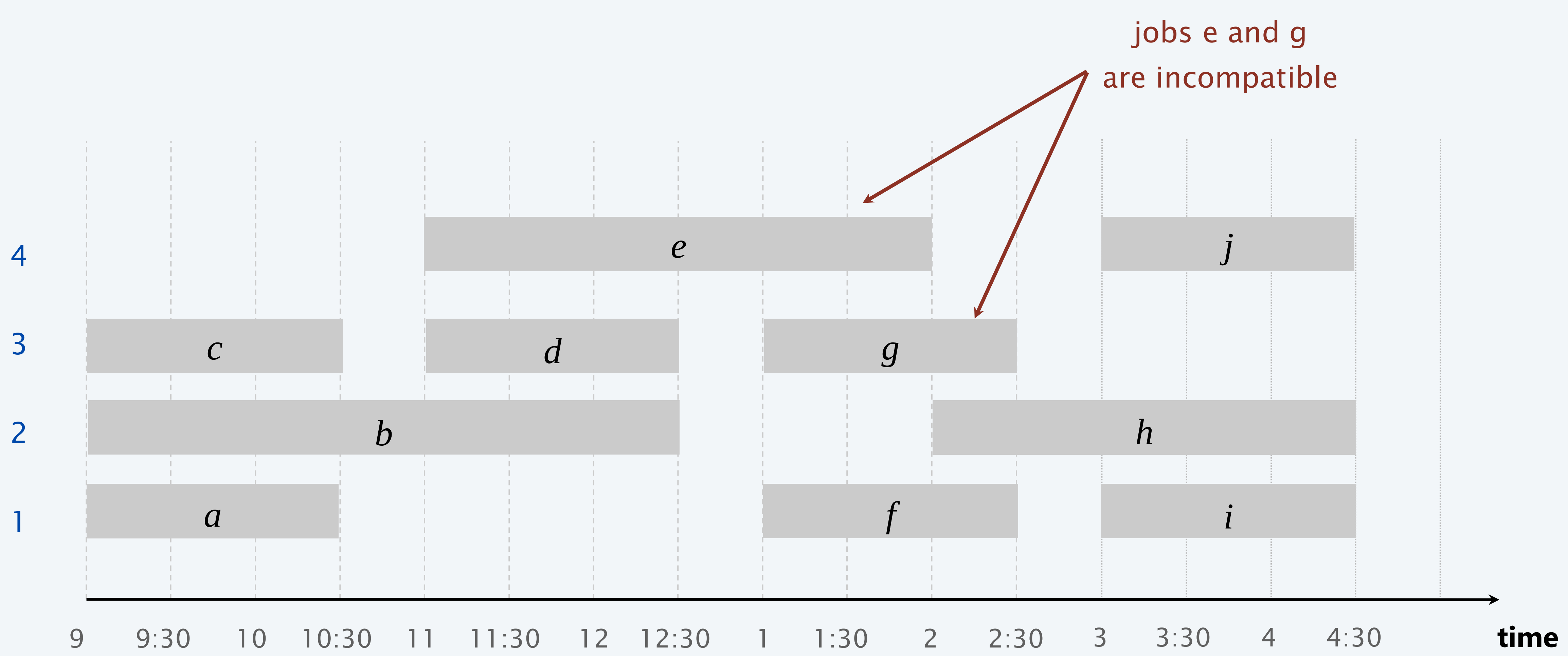
A related problem:

- *interval partitioning*

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.

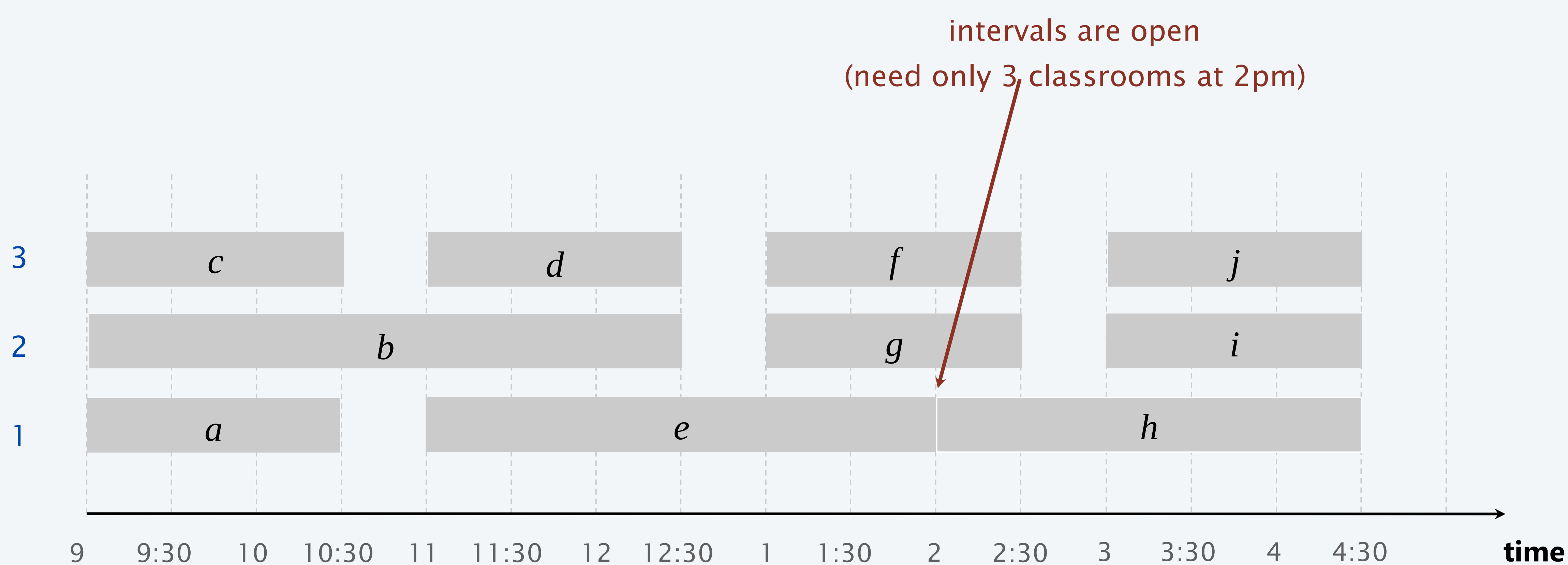
Ex. This schedule uses 4 classrooms to schedule 10 lectures.



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.



Interval partitioning

- Input:
  - A set of n intervals $I_1,\ldots,I_n$
  - interval I_i has starting time s_i and finish time f_i
- Feasible solution:
  - A partition of the intervals into subsets (called classrooms) $C_1,\ldots,C_d$ such that each C_i contains mutually compatible intervals
- Measure (to minimize):
  - number of classrooms, i.e. d

Interval partitioning: greedy algorithms

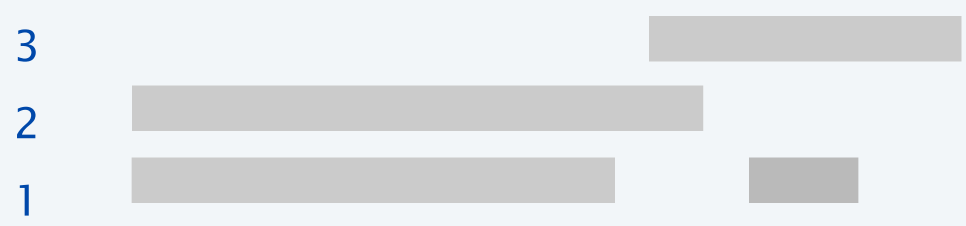
Greedy template. Consider lectures in some natural order. Assign each lecture to an available classroom (which one?); allocate a new classroom if none are available.

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

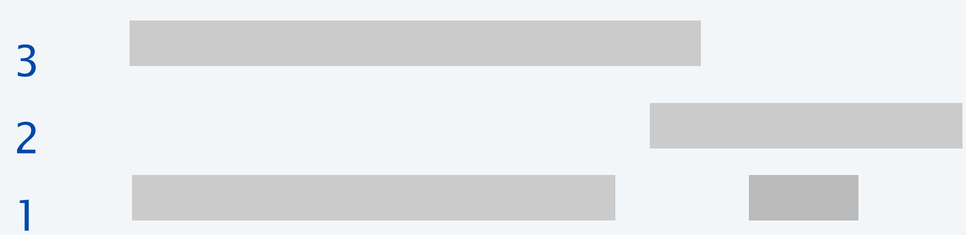
Interval partitioning: greedy algorithms

Greedy template. Consider lectures in some natural order. Assign each lecture to an available classroom (which one?); allocate a new classroom if none are available.

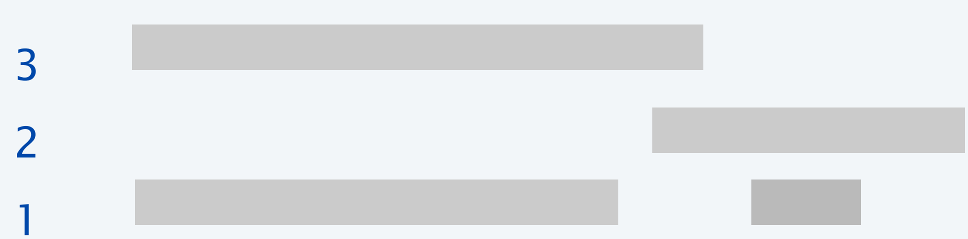
counterexample for earliest finish time



counterexample for shortest interval



counterexample for fewest conflicts



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 $d + 1$.

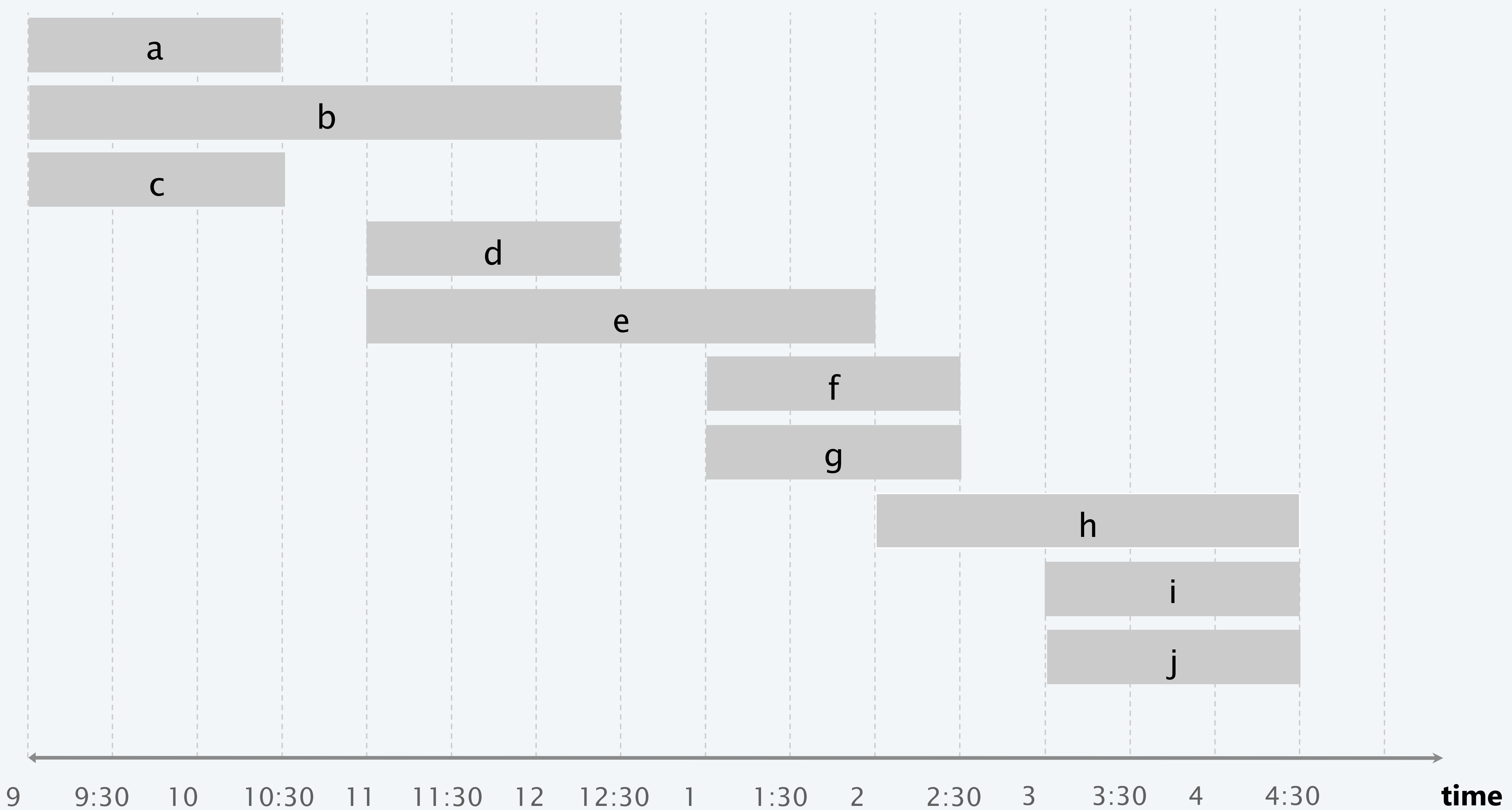
    $d \leftarrow d + 1$.

RETURN schedule.

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.

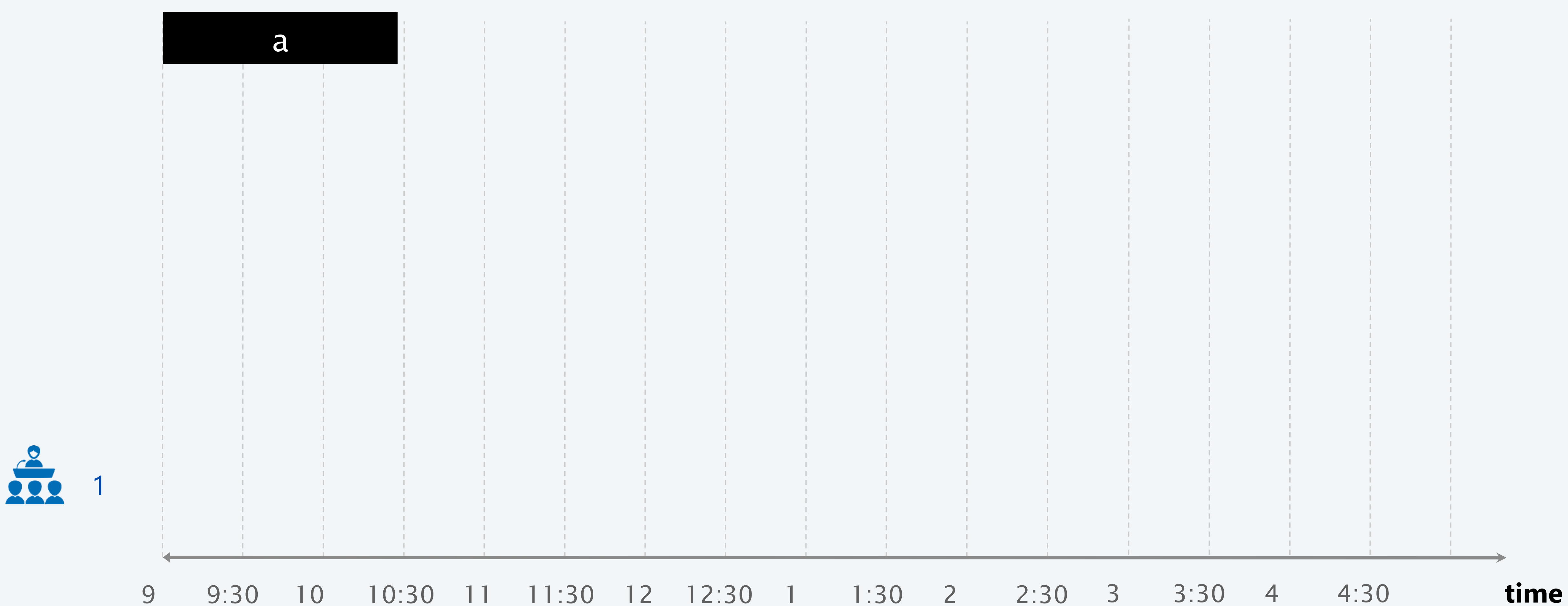


Earliest-start-time-first algorithm demo

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

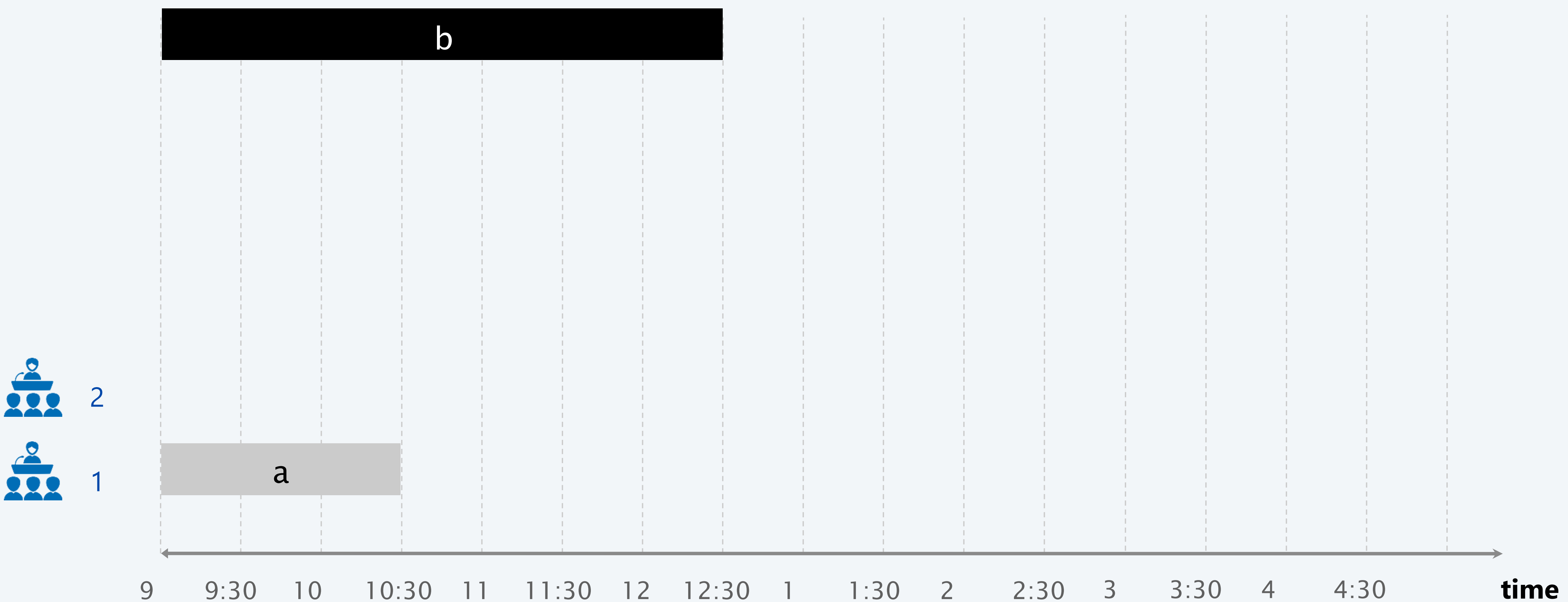


Earliest-start-time-first algorithm demo

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

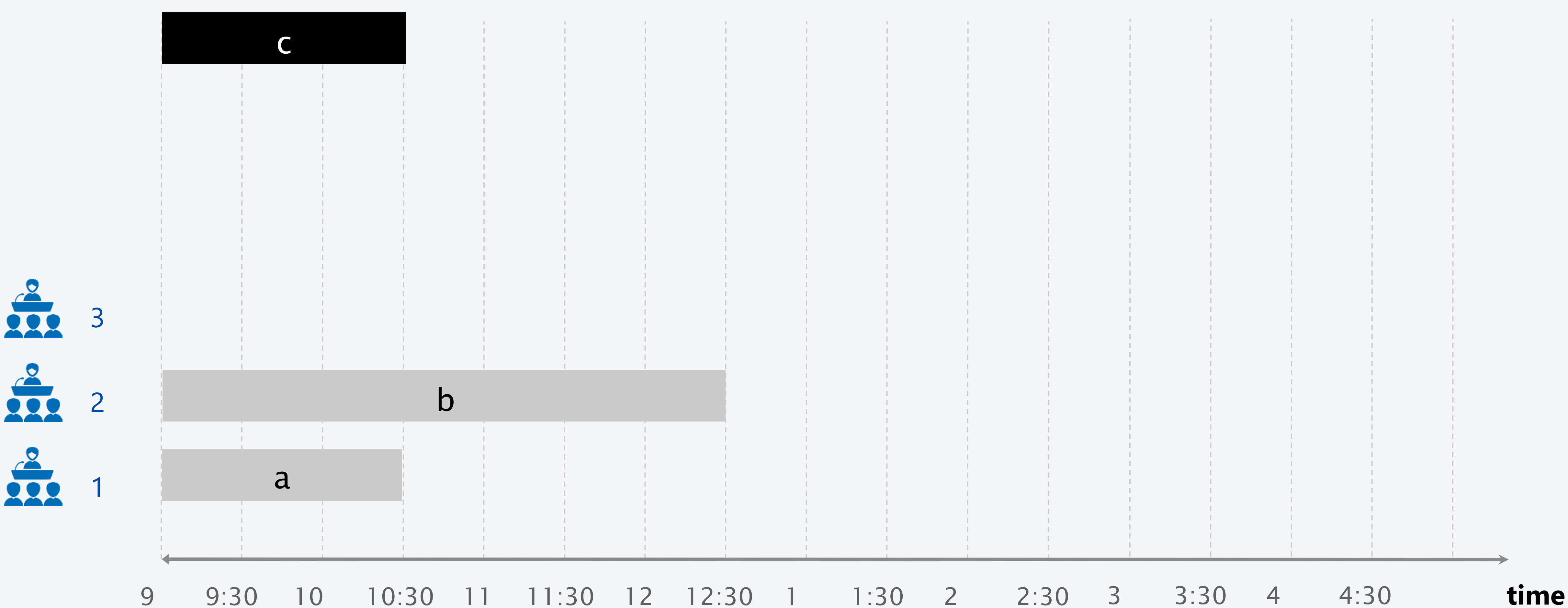


Earliest-start-time-first algorithm demo

Consider lectures in order of start time:

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

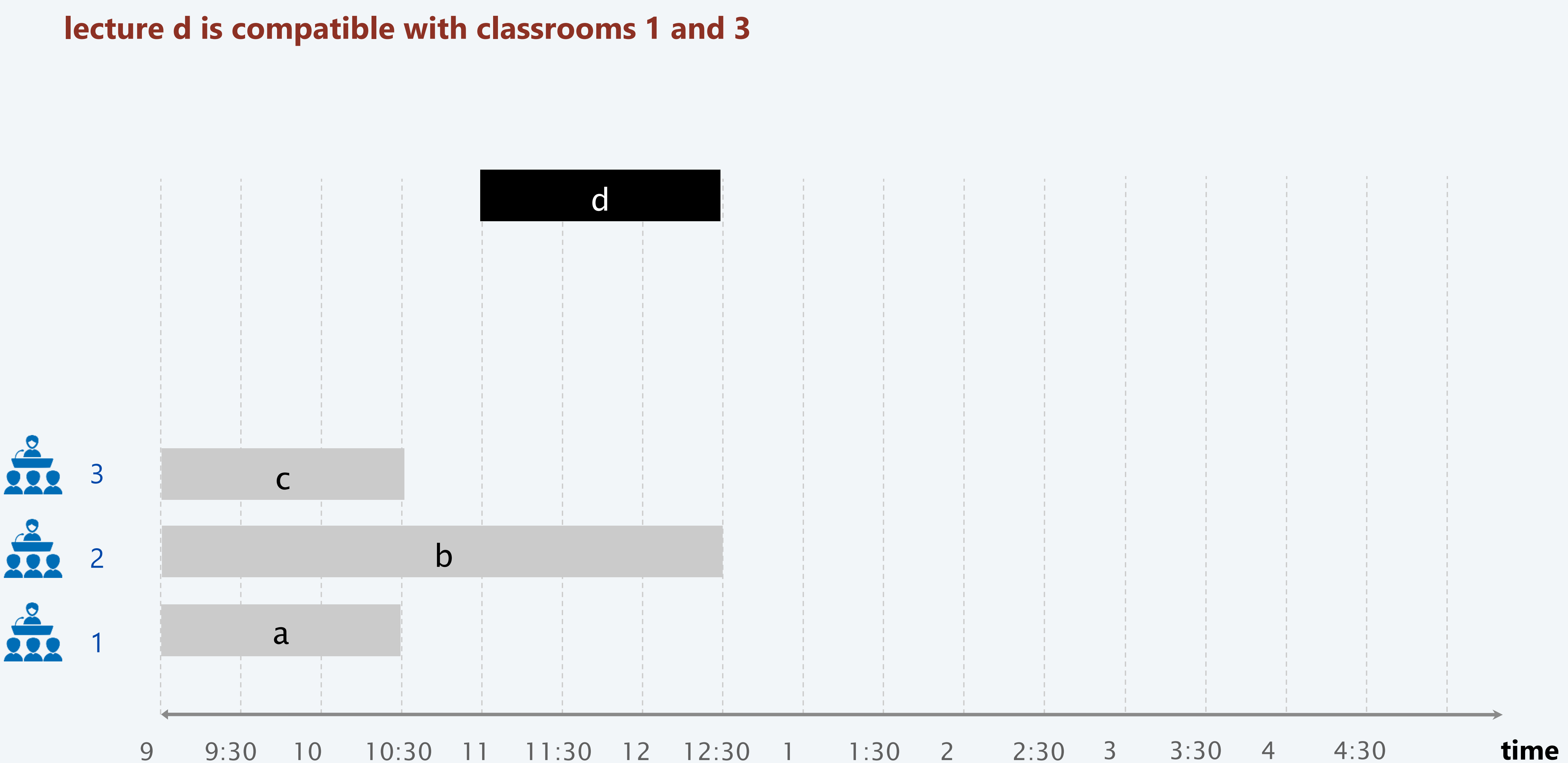
no compatible classroom: open up a new classroom and assign lecture to it



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.

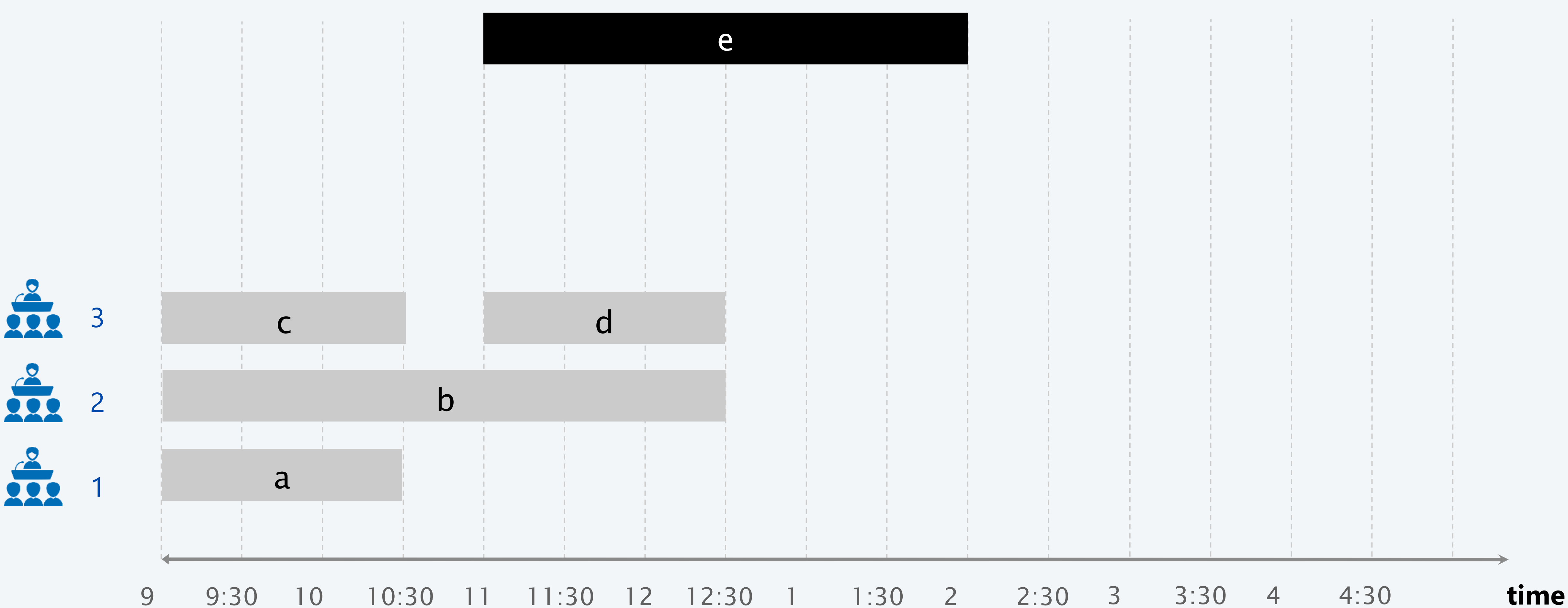


Earliest-start-time-first algorithm demo

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

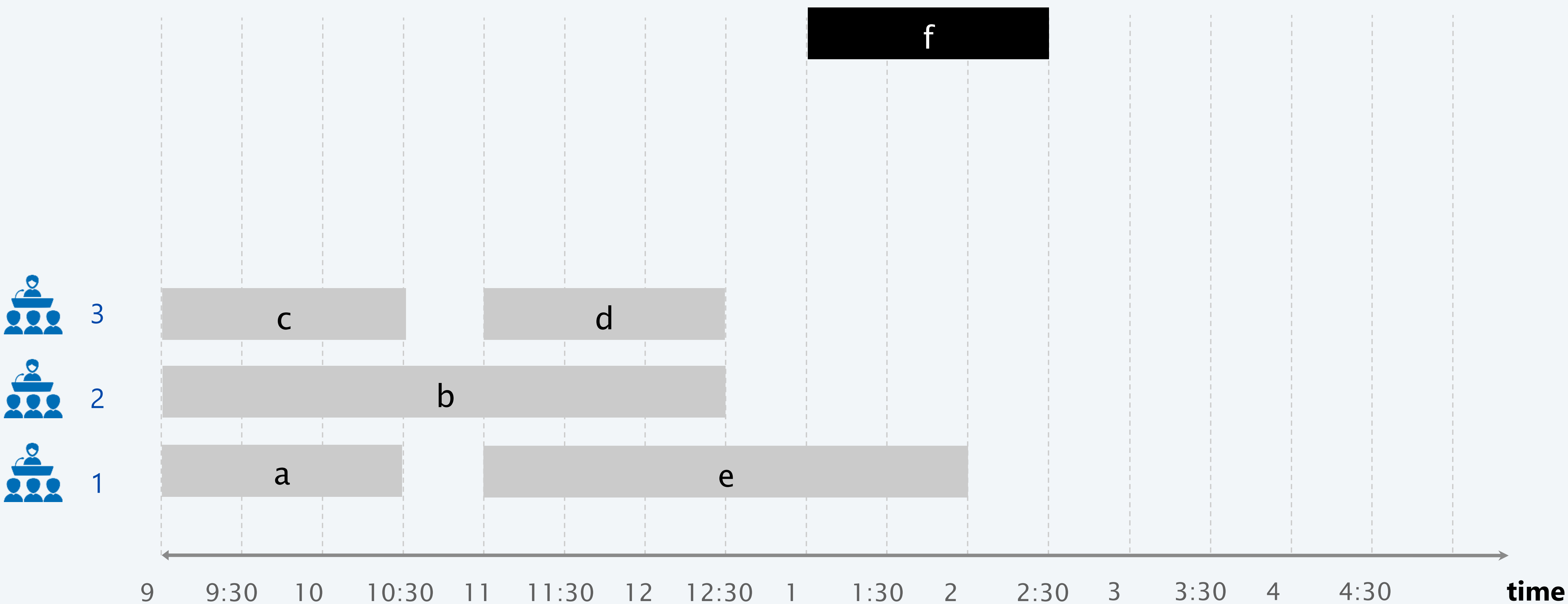


Earliest-start-time-first algorithm demo

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

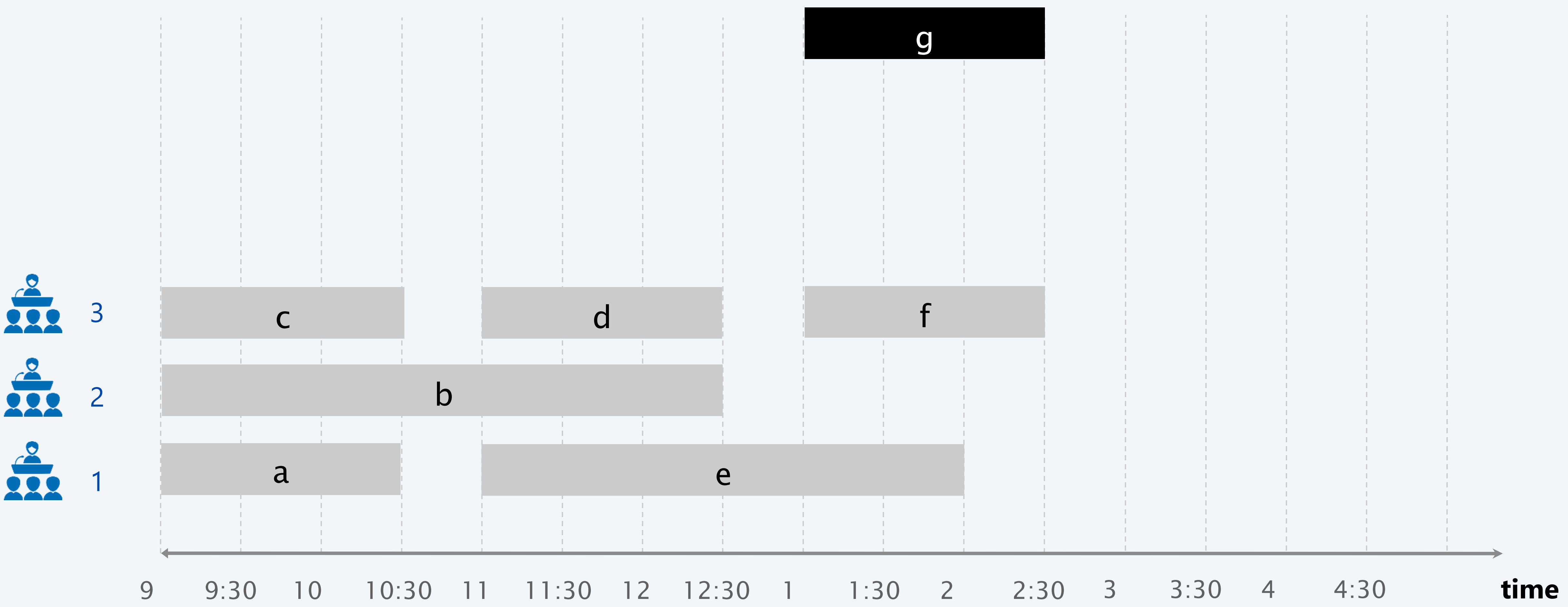


Earliest-start-time-first algorithm demo

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

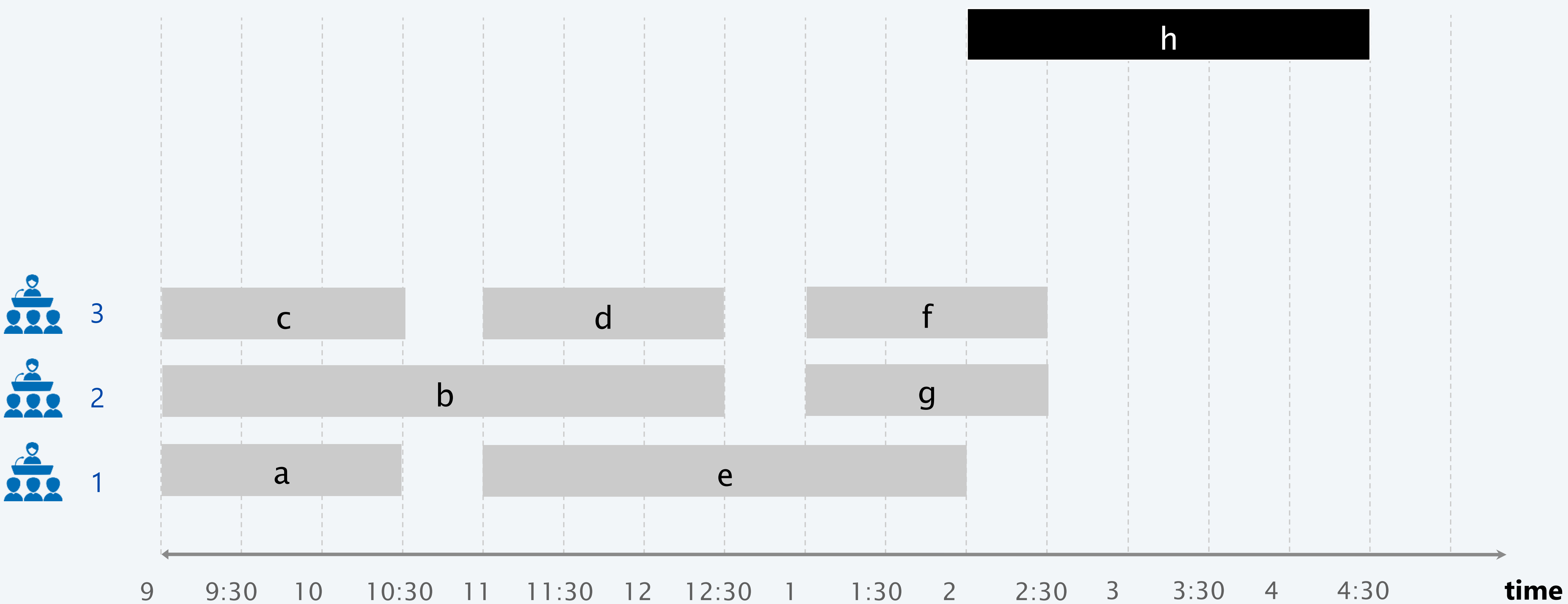


Earliest-start-time-first algorithm demo

Consider lectures in order of start time:

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

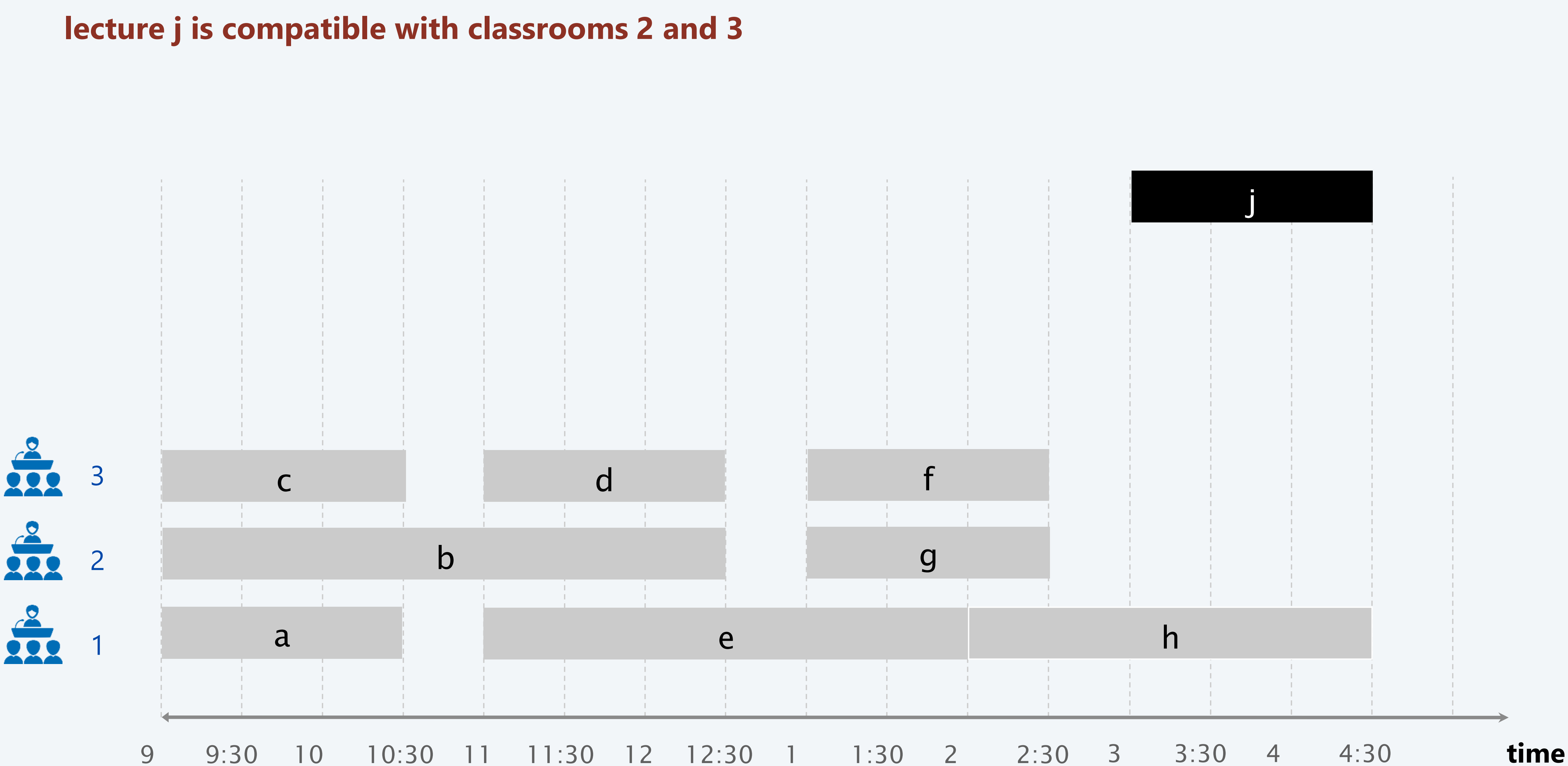
lecture h is compatible with classroom 1



Earliest-start-time-first algorithm demo

Consider lectures in order of start time:

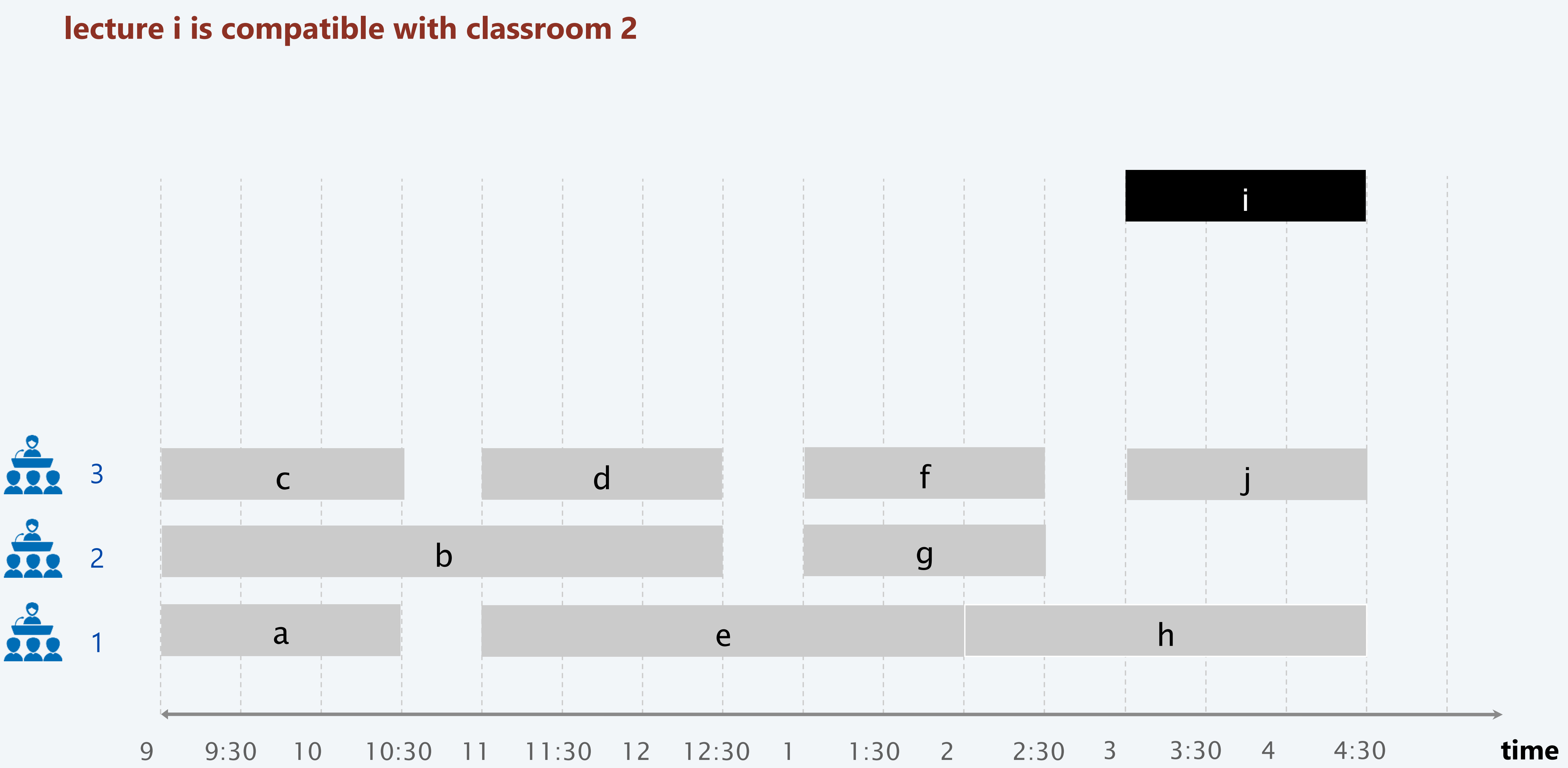
- Assign next lecture to any compatible classroom (if one exists).
- Otherwise, open up a new classroom.



Earliest-start-time-first algorithm demo

Consider lectures in order of start time:

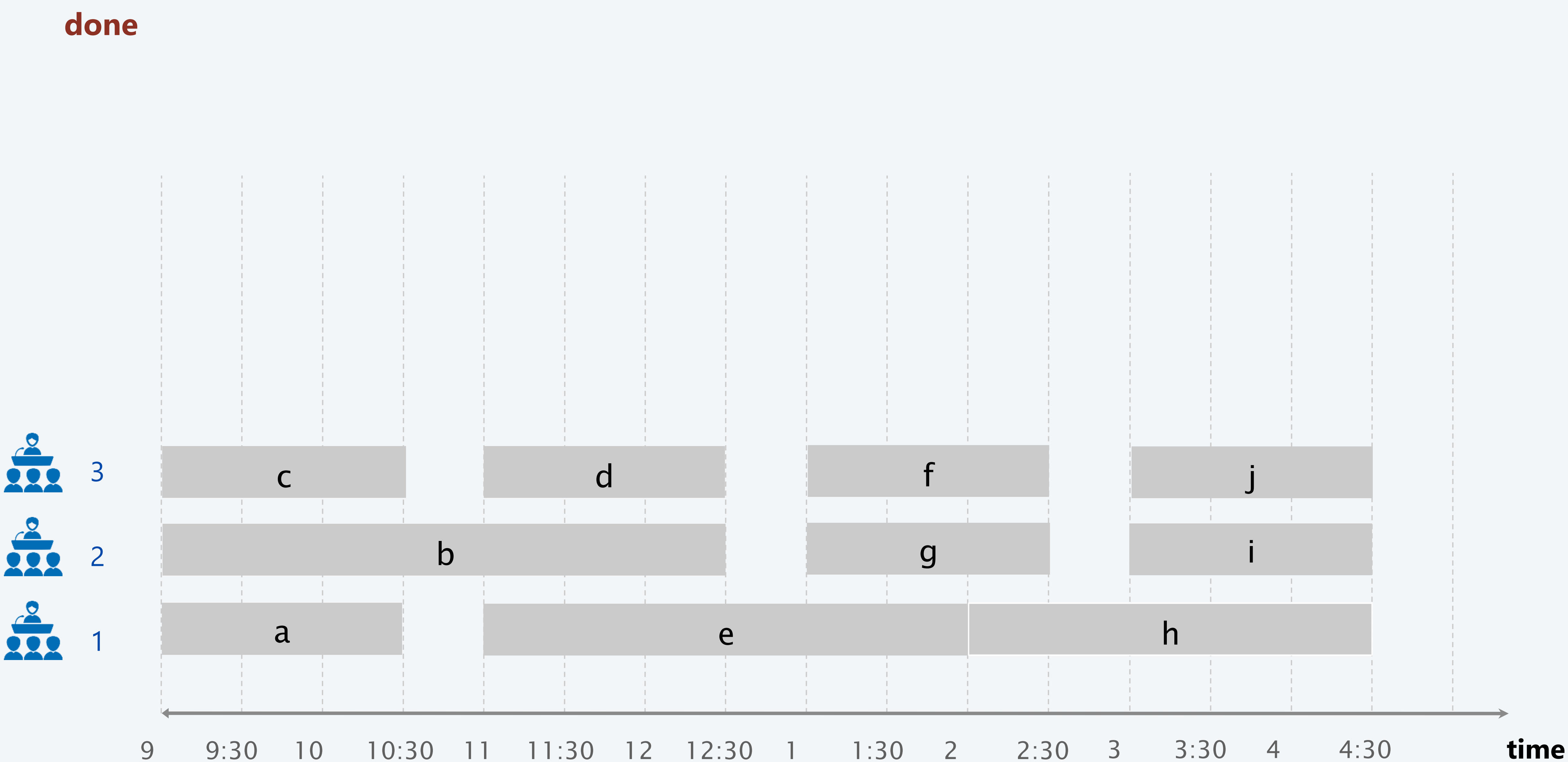
- Assign next lecture to any compatible classroom (if one exists).
- Otherwise, open up a new classroom.



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.



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. ▪

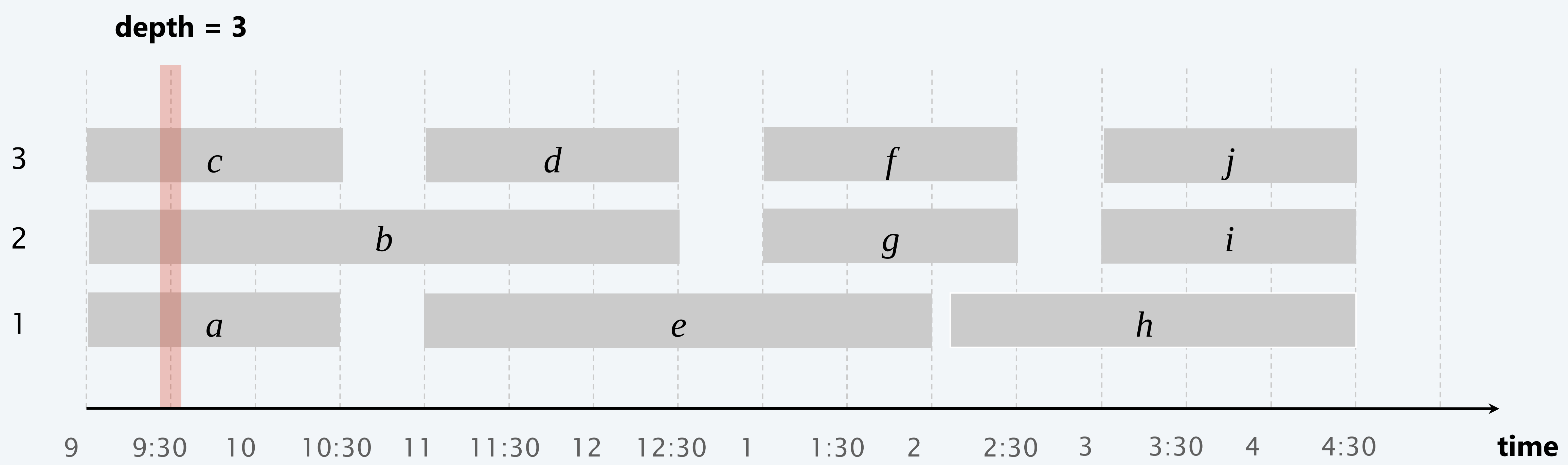
Remark. This implementation chooses a classroom k whose finish time of its last lecture is the earliest.

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. ▪