Intro: Coin Changing

Coin Changing

Goal. Given currency denominations: 1, 5, 10, 25, 100, give change to customer using *fewest* number of coins.

Ex: 34¢.

Algorithm is "Greedy": One large coin better than two or more smaller ones

Cashier's algorithm. At each iteration, give the *largest* coin valued ≤ the amount to be paid.

Ex: $2.89.

Coin-Changing: Does Greedy Always Work?

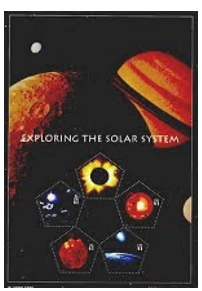
Observation. Greedy algorithm is sub-optimal for US postal denominations: 1, 10, 21, 34, 70, 100, 350, 1225, 1500.

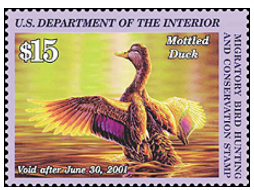
Counterexample. 140¢.

- Greedy: 100, 34, 1, 1, 1, 1, 1, 1.
- Optimal: 70, 70.

Algorithm is “Greedy”, but also short-sighted – attractive choice now may lead to dead ends later.

Correctness is key!





Outline & Goals

"Greedy Algorithms"

- what they are

Pros

- intuitive
- often simple
- often fast

Proof techniques

- stay ahead
- structural
- exchange arguments

Plan

Greed

Greeed

Greeeeeed

Greeeeeeeeeeed

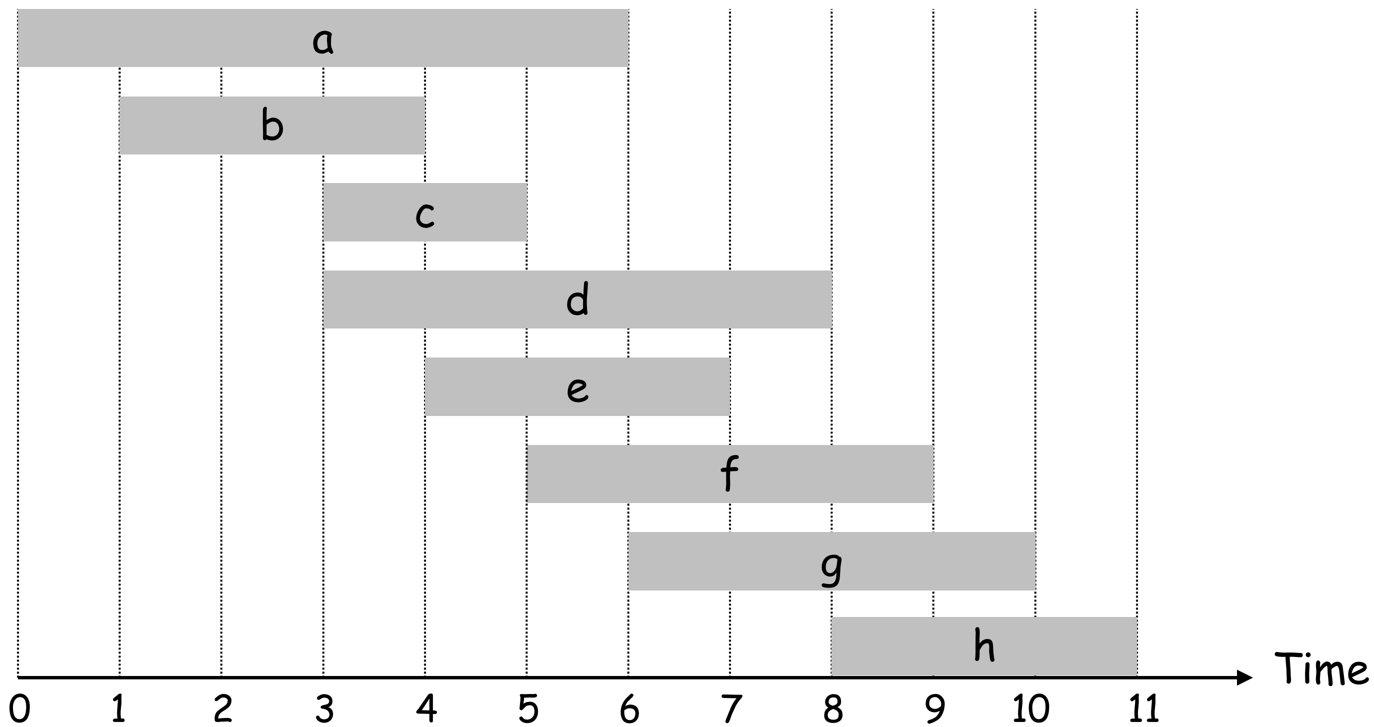
4.1 Interval Scheduling

Proof Technique 1: “greedy stays ahead”

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 order. Take each job provided it's compatible with the ones already taken.

- What order?
- Does that give best answer?
- Why or why not?
- Does it help to be greedy about order?

Interval Scheduling: Greedy Algorithms

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

[Earliest start time] Consider jobs in ascending order of start time s_j.

[Earliest finish time] Consider jobs in ascending order of finish time f_j.

[Shortest interval] Consider jobs in ascending order of interval length $f_j - s_j$.

[Fewest conflicts] For each job, count the number of conflicting jobs c_j. Schedule in ascending order of conflicts c_j.

Interval Scheduling: Greedy Algorithms

Greedy template. Consider jobs in some 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 Algorithms

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

~~[Earliest start time] Consider jobs in ascending order of start time s_j.~~

[Earliest finish time] Consider jobs in ascending order of finish time f_j.

~~[Shortest interval] Consider jobs in ascending order of interval length $f_j - s_j$.~~

~~[Fewest conflicts] For each job, count the number of conflicting jobs c_j. Schedule in ascending order of conflicts c_j.~~

Interval Scheduling: Greedy Algorithm

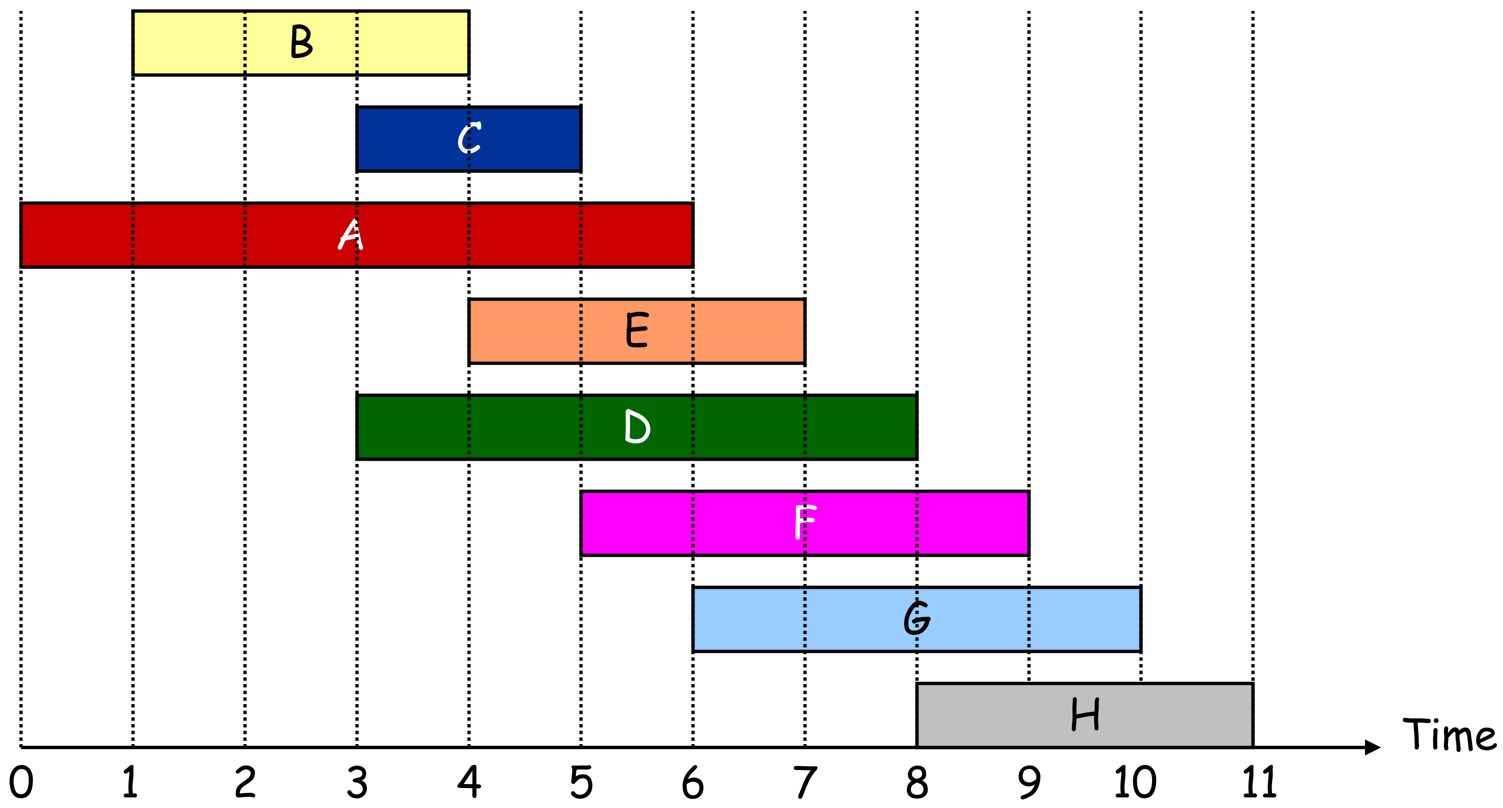
Greedy algorithm. Consider jobs in increasing order of finish time. Take each job provided it's compatible with the ones already taken.

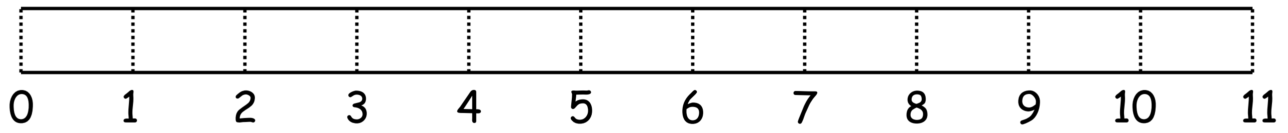
```
Sort jobs by finish times so that f1 ≤ f2 ≤ ... ≤ fn.

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

Implementation. O(n log n).

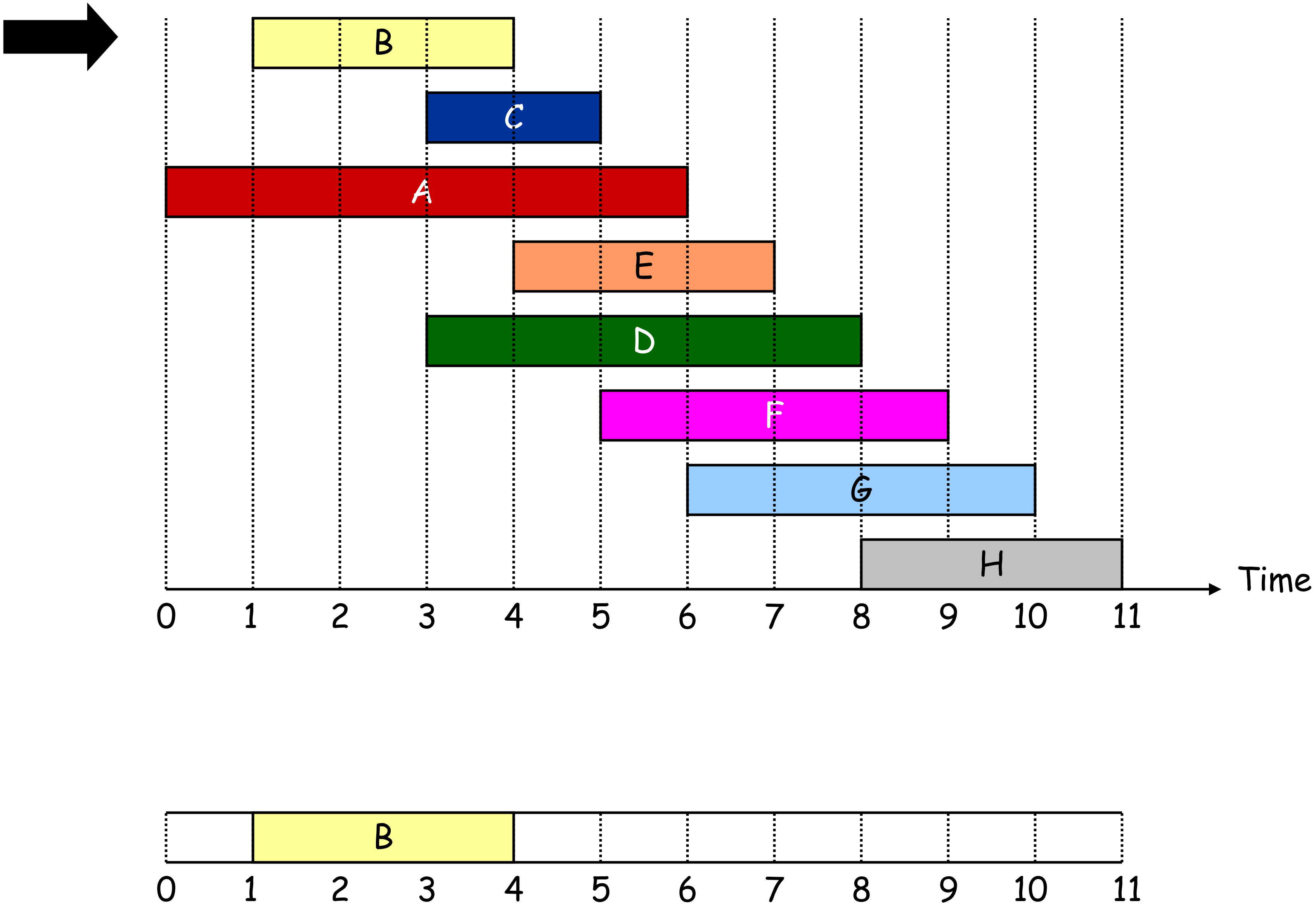
- Remember job j* that was added last to A.
- Job j is compatible with A if $s_j \geq f_{j^*}$.

Interval Scheduling

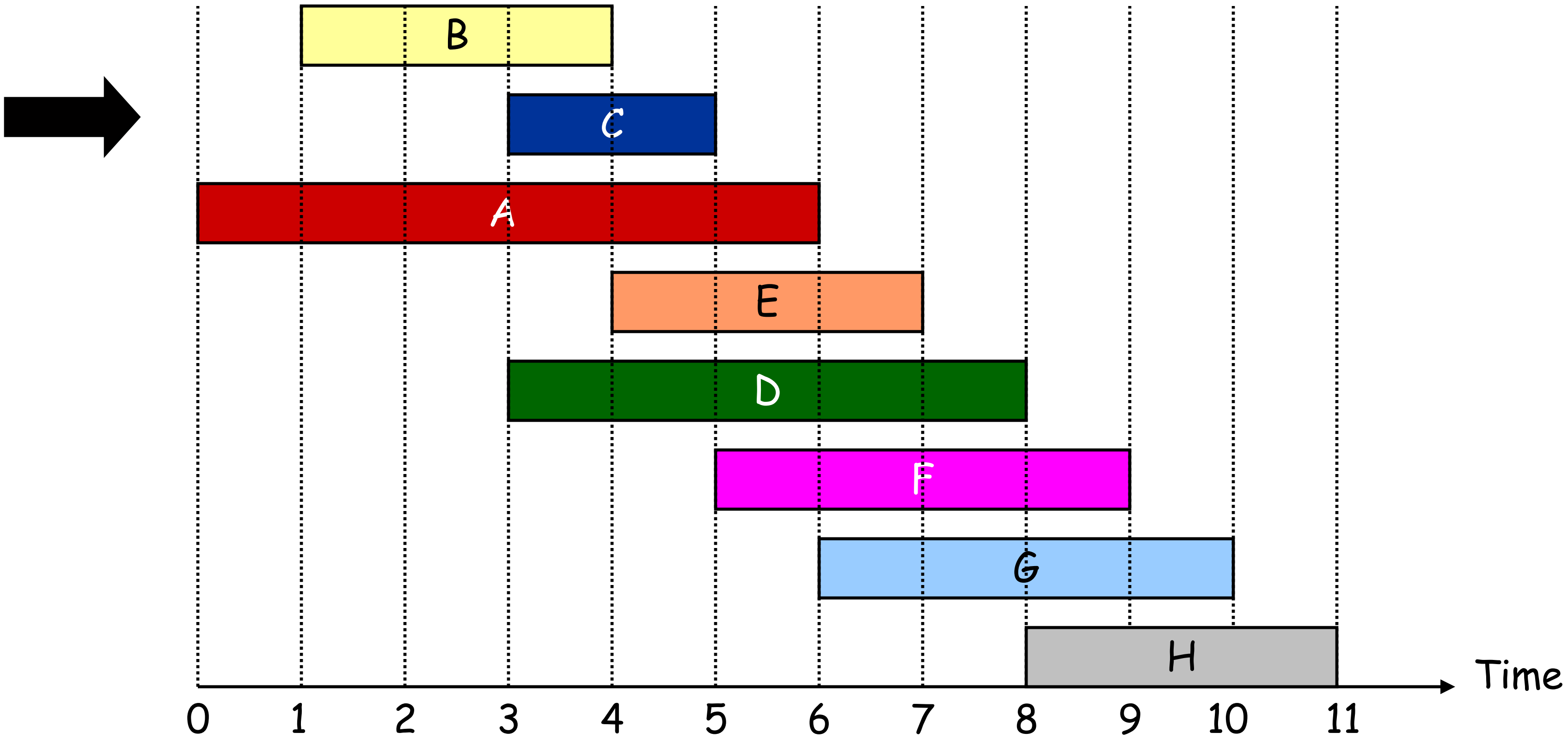


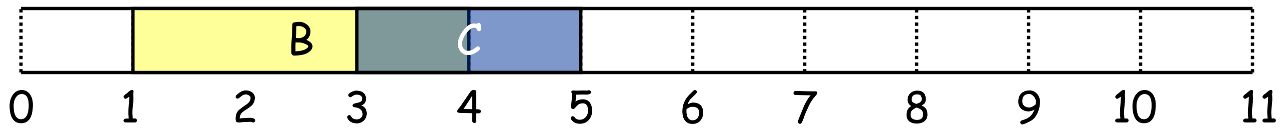


Interval Scheduling

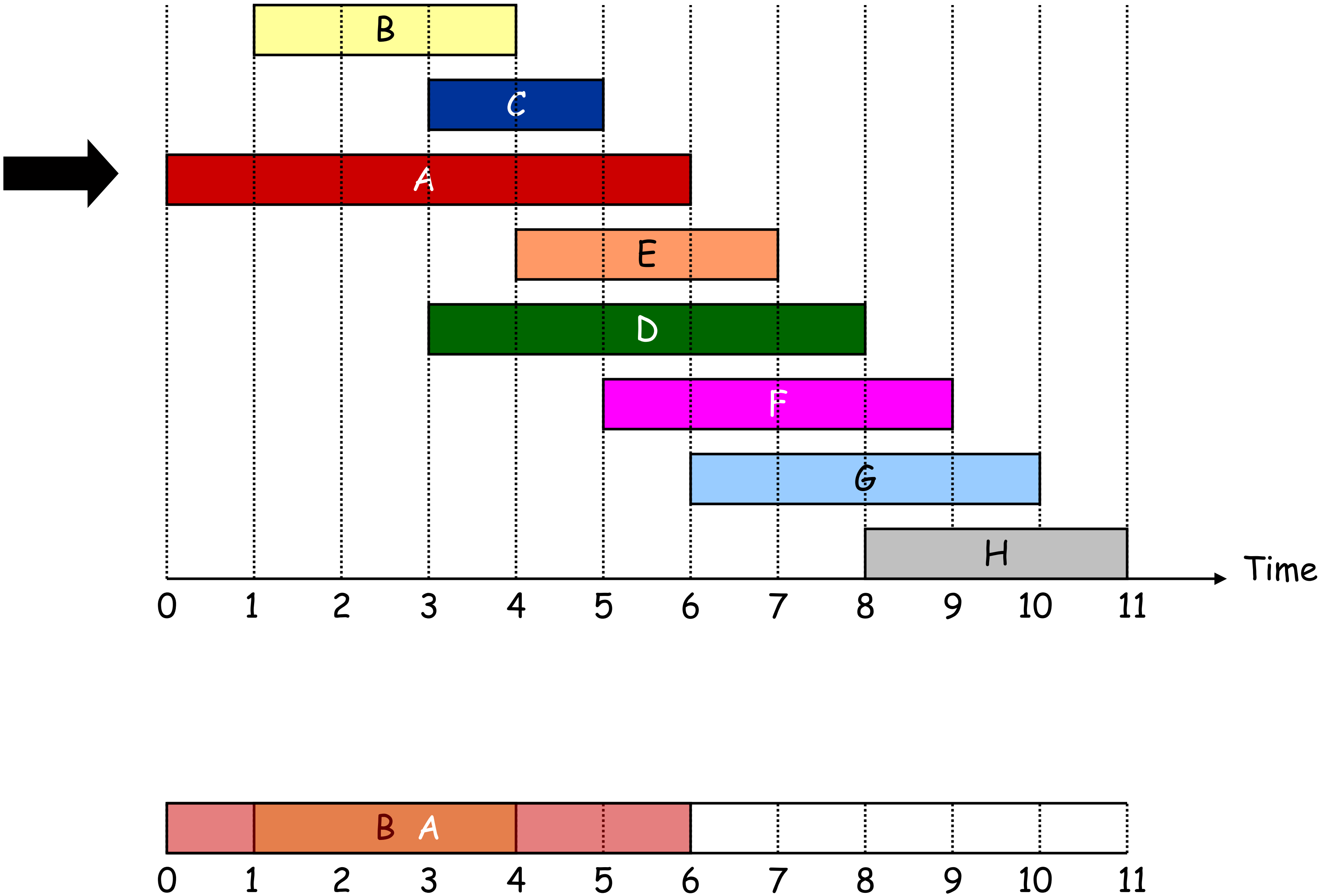


Interval Scheduling

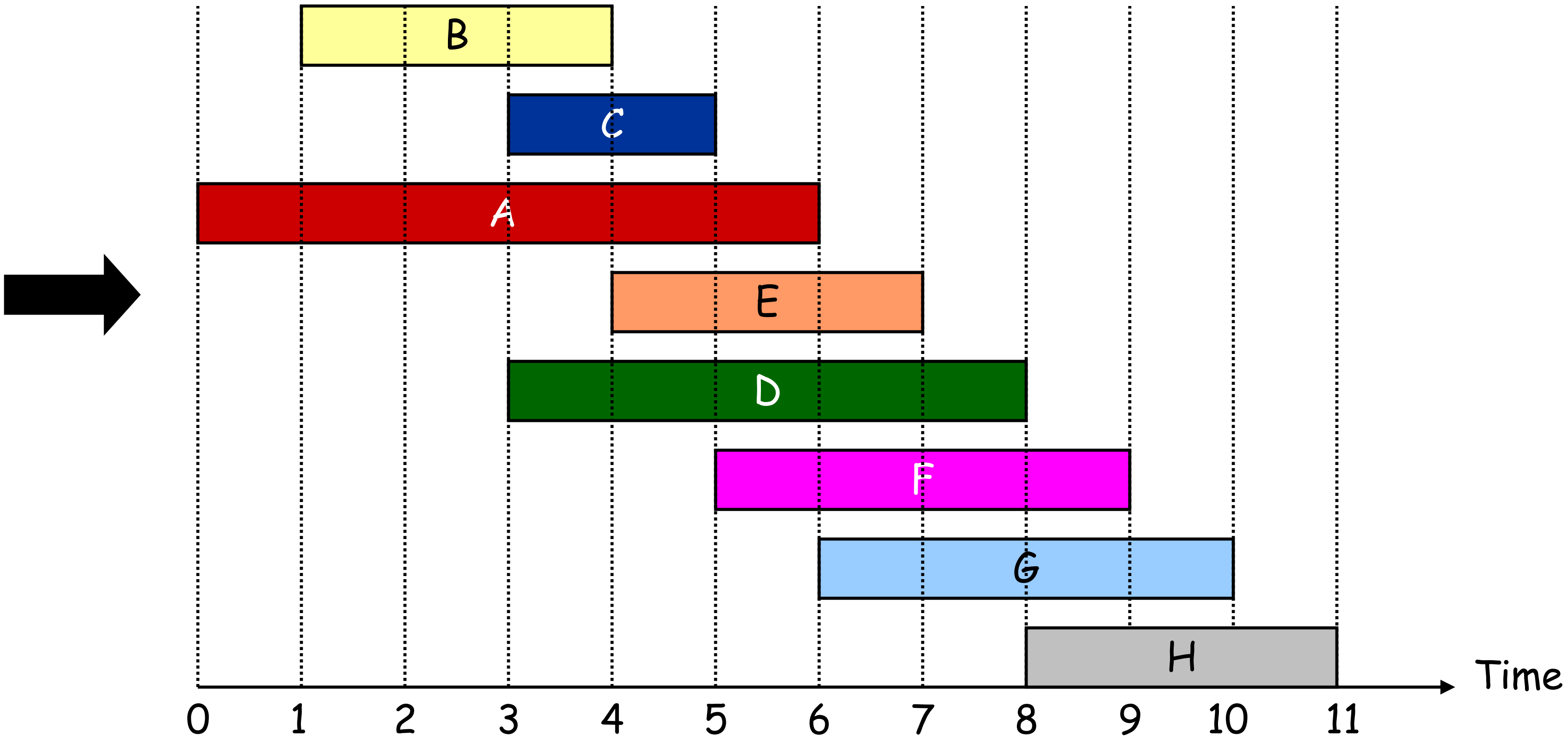


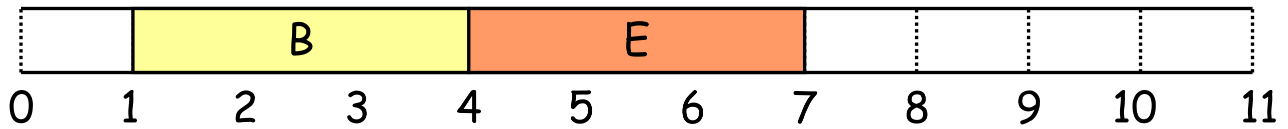


Interval Scheduling

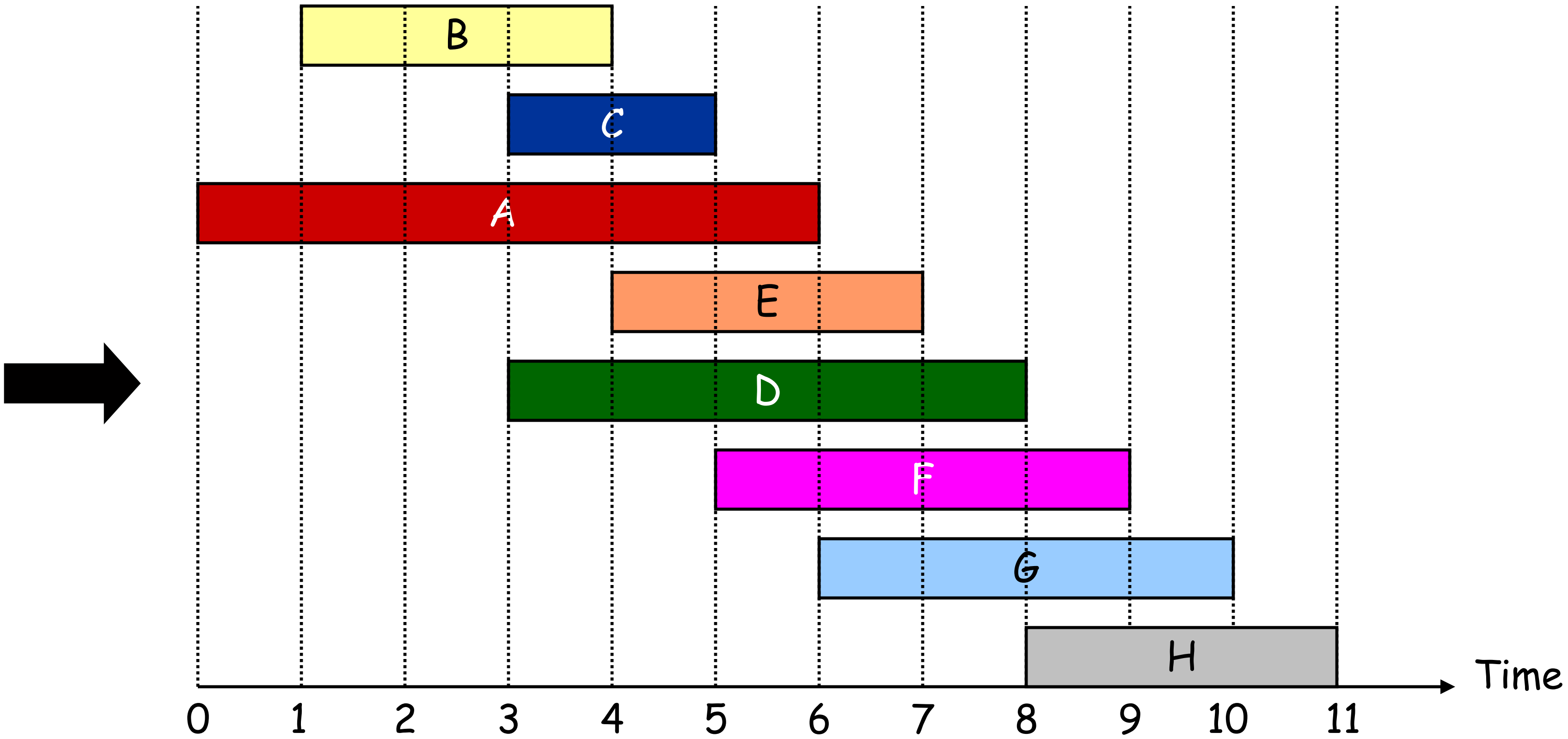


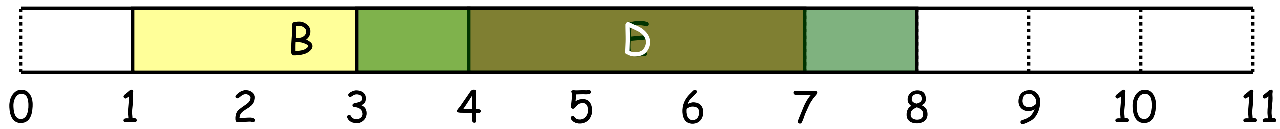
Interval Scheduling



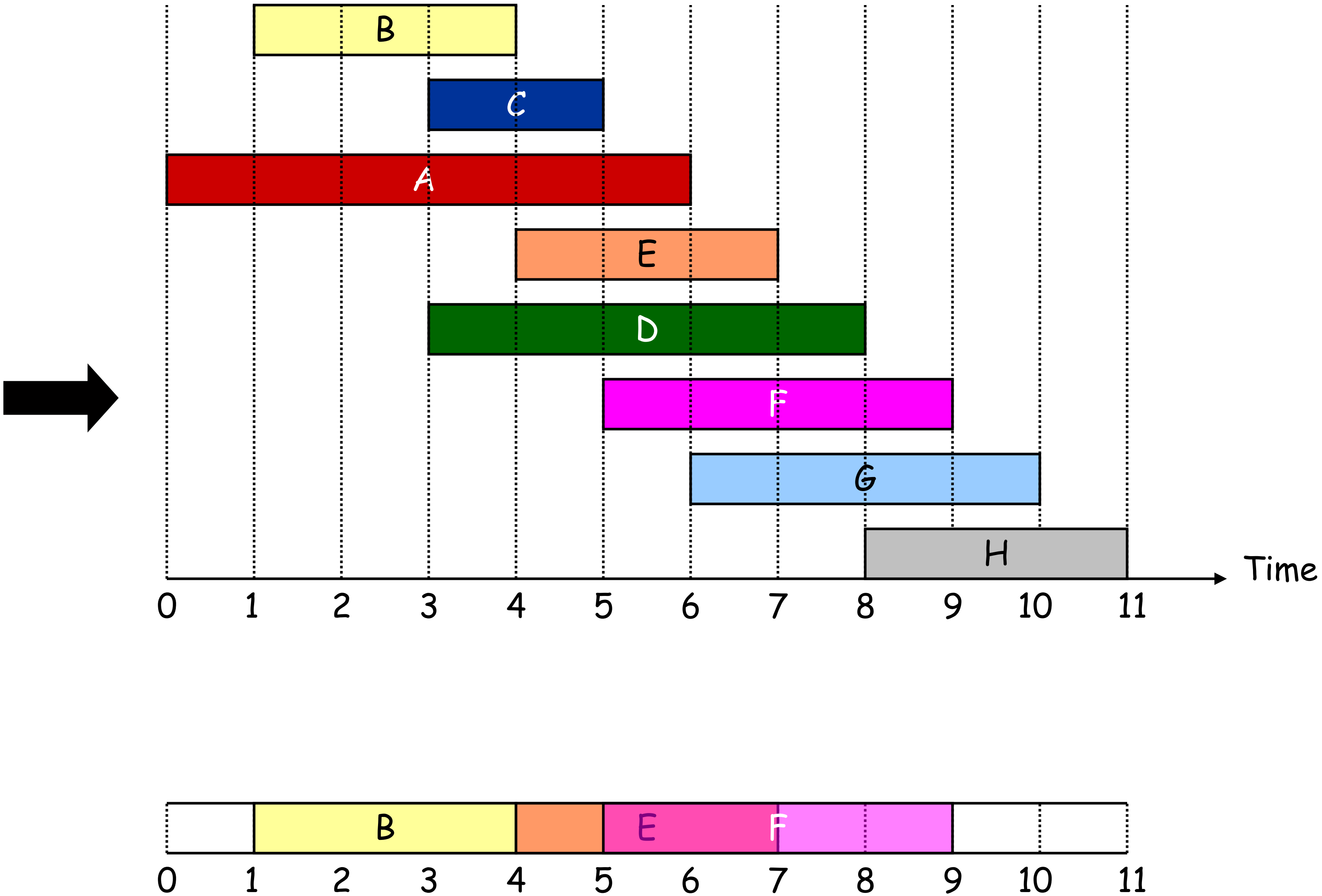


Interval Scheduling

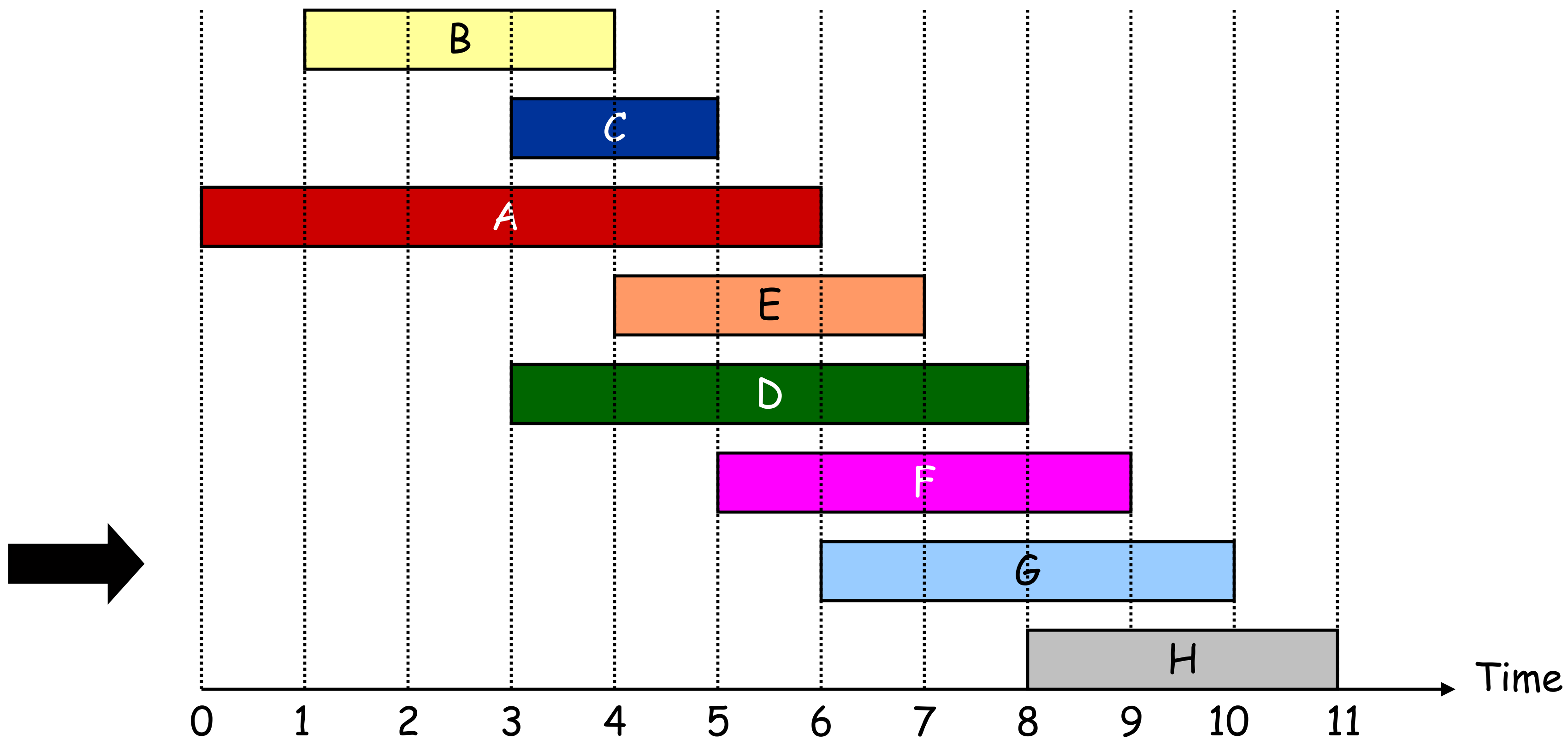


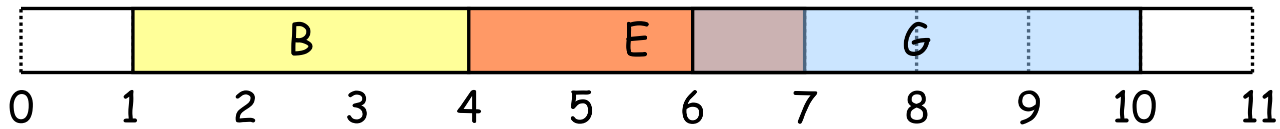


Interval Scheduling

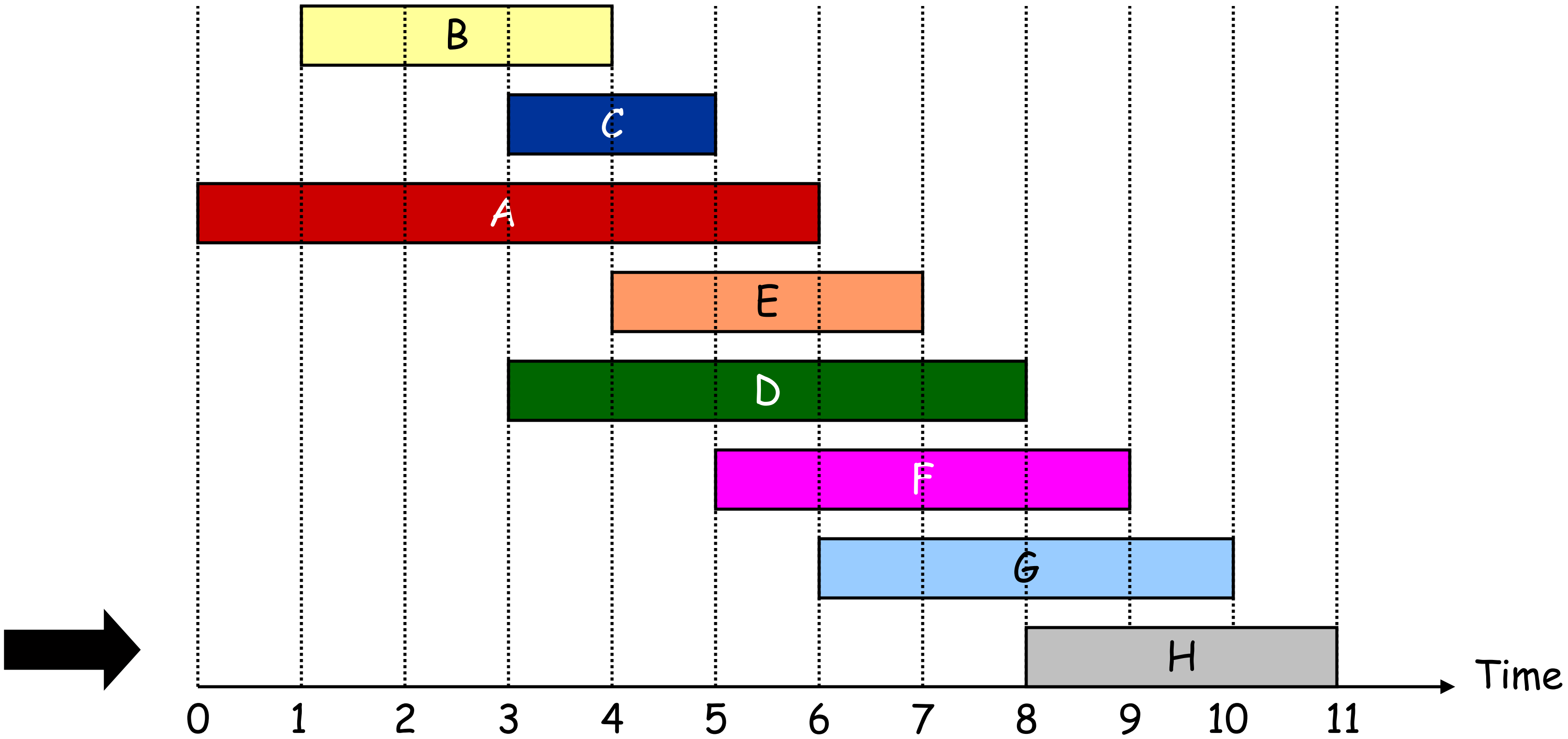


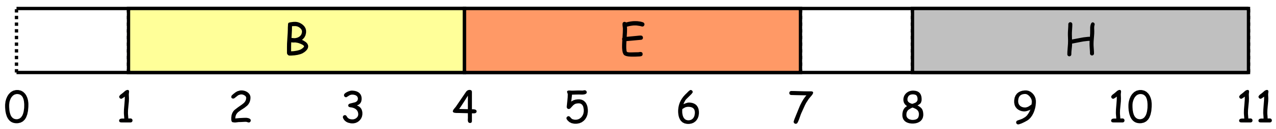
Interval Scheduling





Interval Scheduling





Interval Scheduling: Correctness

Theorem. Greedy algorithm is optimal.

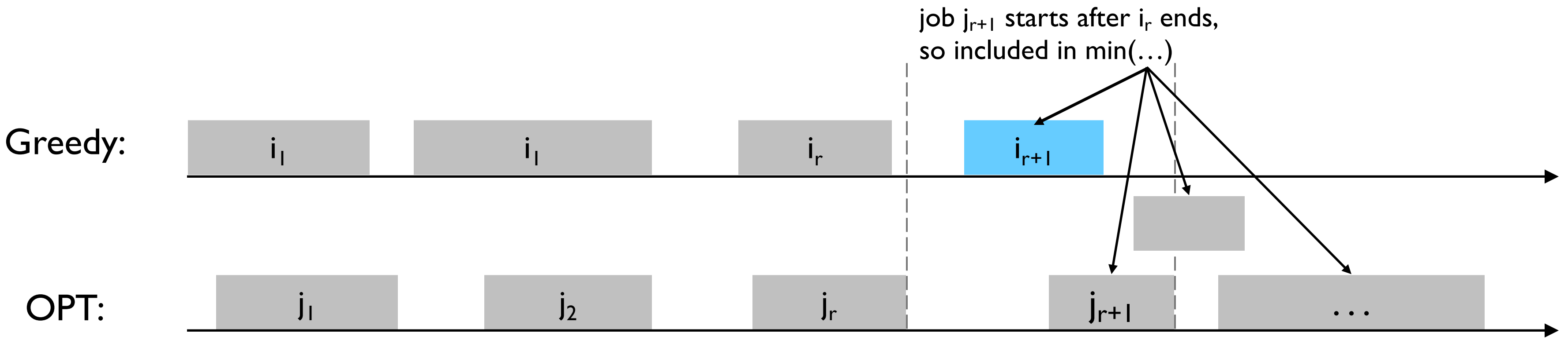
Pf. ("greedy stays ahead")

Let $i_1, i_2, \ldots i_k$ be jobs picked by greedy, $j_1, j_2, \ldots j_m$ those in some optimal solution

Show $f(i_r) \leq f(j_r)$ by induction on r.

- Basis: i_1 chosen to have min finish time, so $f(i_1) \leq f(j_1)$
- Ind: $f(i_r) \leq f(j_r) \leq s(j_{r+1})$, so j_{r+1} is among the candidates considered by greedy when it picked i_{r+1}, & it picks min finish, so $f(i_{r+1}) \leq f(j_{r+1})$

Similarly, $k \geq m$, else j_{k+1} is among (nonempty) set of candidates for i_{k+1}



4.1 Interval Partitioning

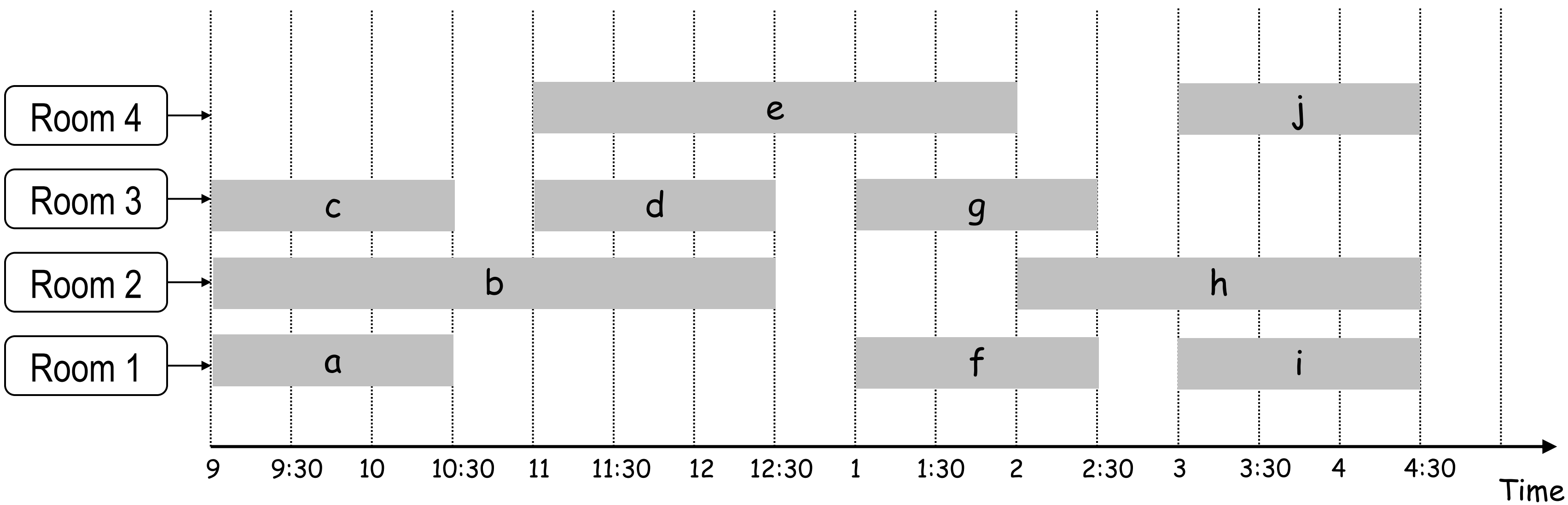
Proof Technique 2: “Structural”

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 occur at the same time in the same room.

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



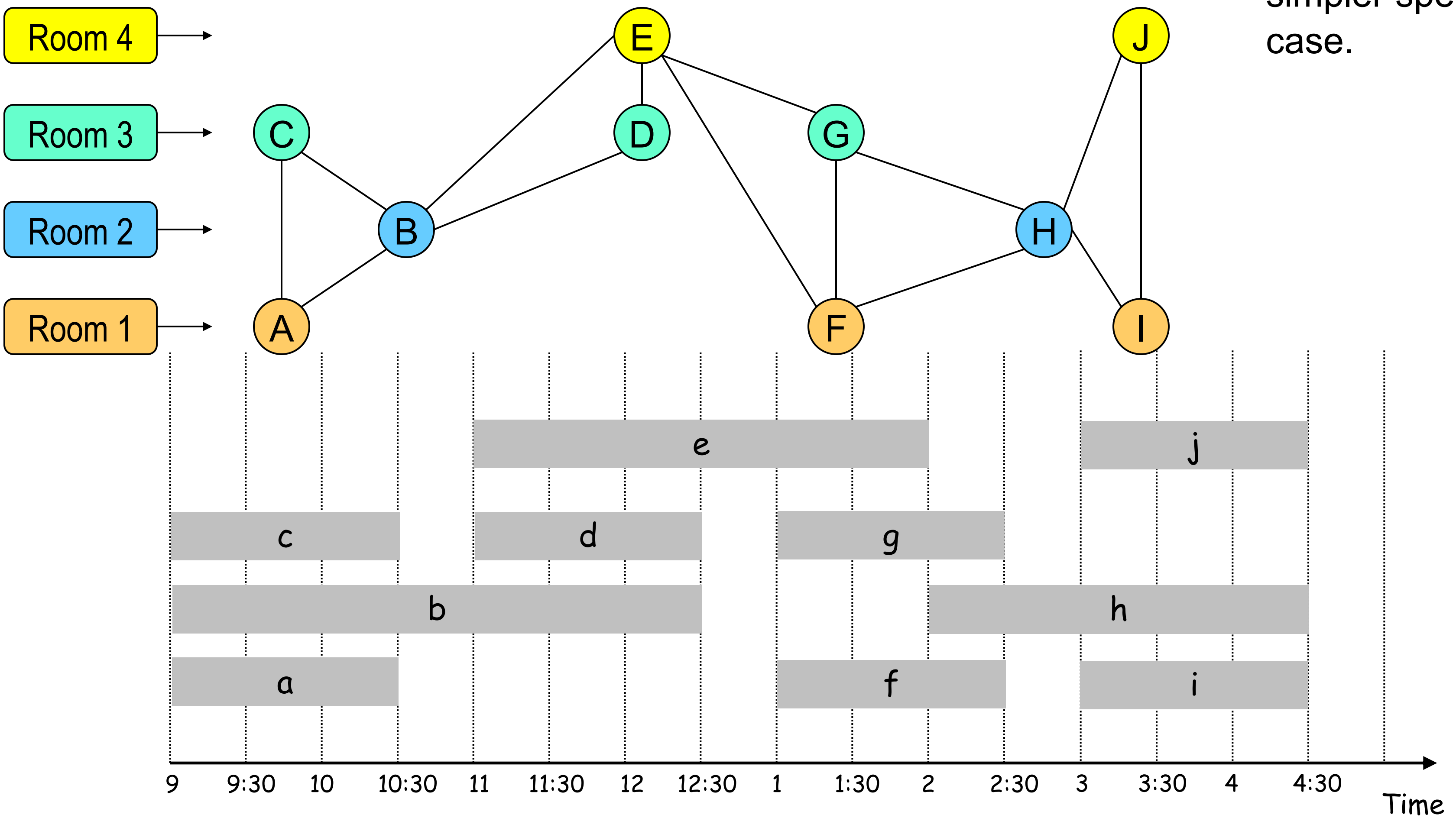
Interval Partitioning as Interval Graph Coloring

Vertices = classes;

edges = conflicting class pairs;

different colors = different assigned rooms

Note: graph coloring is very hard in general, but graphs corresponding to interval intersections are a much simpler special case.

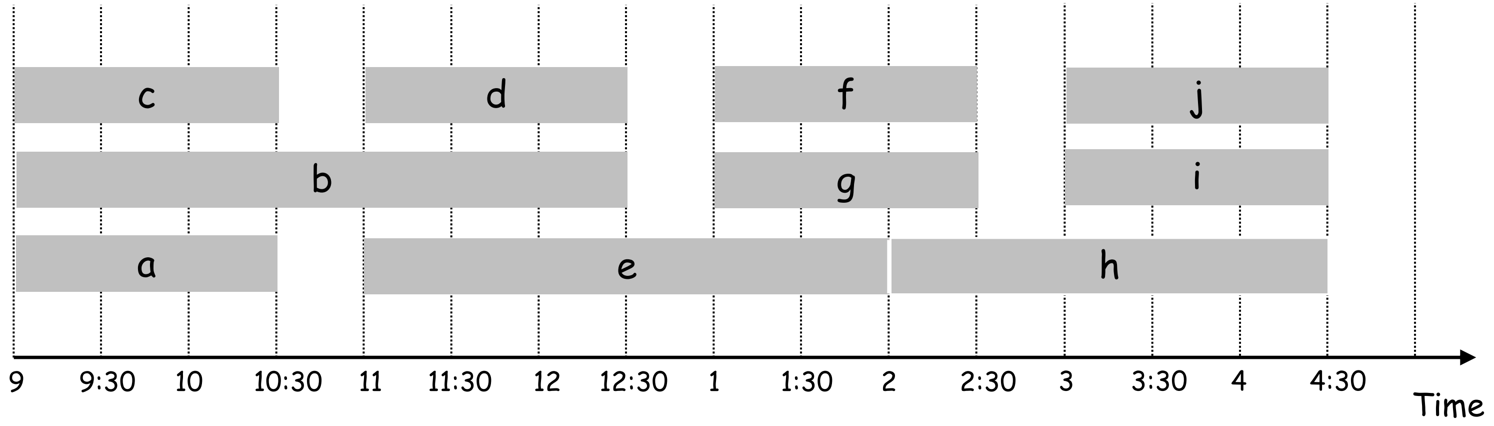


Interval Partitioning

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 occur at the same time in the same room.

Ex: This schedule uses only 3.



Interval Scheduling: Greedy Algorithms

Greedy template. Consider lectures in some order. If next lecture fits in the schedule we have, add it to one of the classrooms, otherwise open a new classroom.

[Earliest start time] Consider lectures in ascending order of start time s_j.

[Earliest finish time] Consider lectures in ascending order of finish time f_j.

[Shortest interval] Consider lectures in ascending order of interval length $f_j - s_j$.

[Fewest conflicts] For each lecture, count the number of conflicting lectures c_j. Schedule in ascending order of conflicts c_j.

Interval Scheduling: Greedy Algorithms

counterexample for earliest finish time

counterexample for shortest interval

counterexample for fewest conflicts

Interval Scheduling: Greedy Algorithms

Greedy template. Consider lectures in some order. If next lecture fits in the schedule we have, add it to one of the classrooms, otherwise open a new classroom.

[Earliest start time] Consider lectures in ascending order of start time s_j.

~~[Earliest finish time] Consider lectures in ascending order of finish time f_j.~~

~~[Shortest interval] Consider lectures in ascending order of interval length $f_j - s_j$.~~

~~[Fewest conflicts] For each lecture, count the number of conflicting lectures c_j. Schedule in ascending order of conflicts c_j.~~

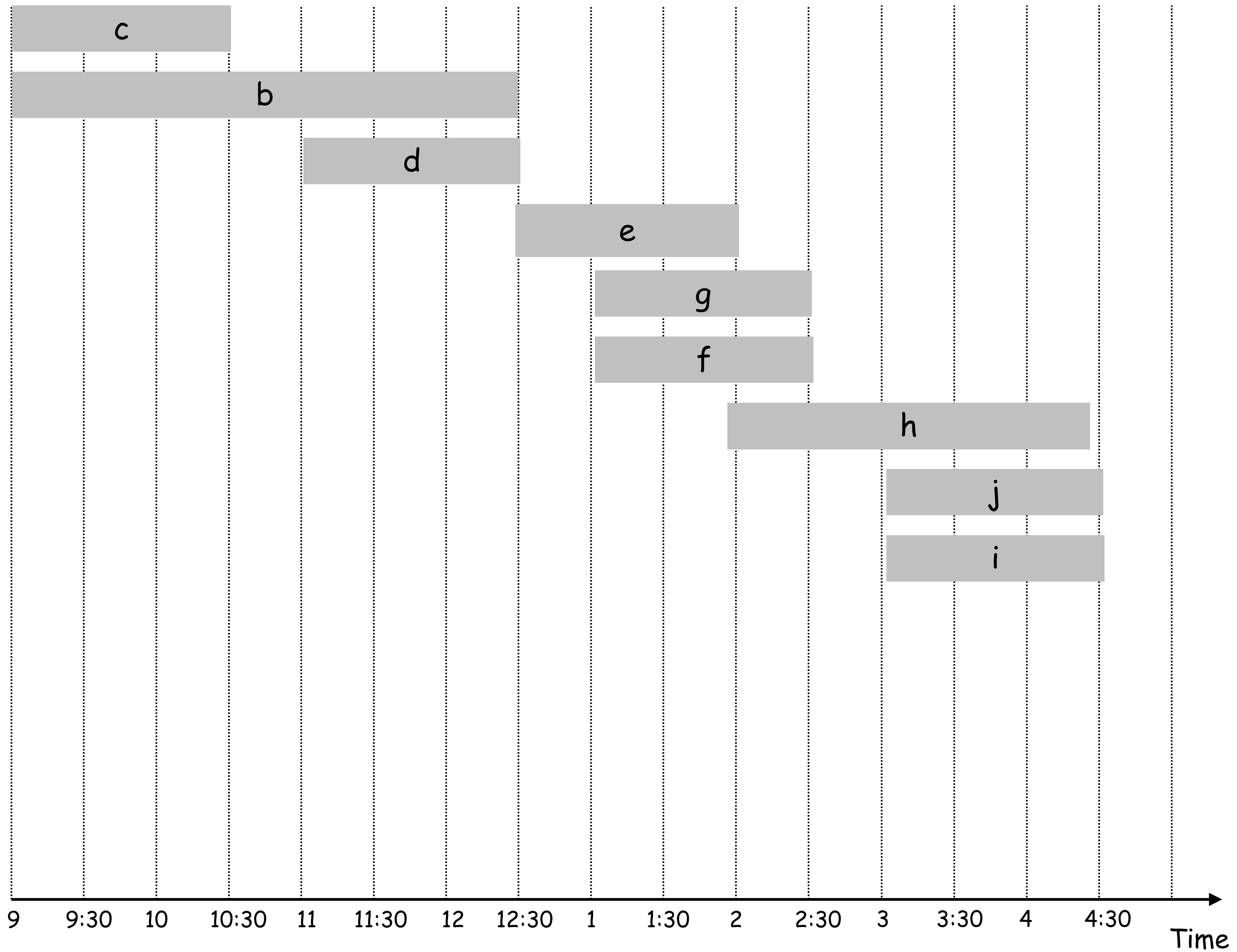
Interval Partitioning: Greedy Algorithm

Greedy algorithm. Consider lectures in increasing order of start time: assign lecture to any compatible classroom.

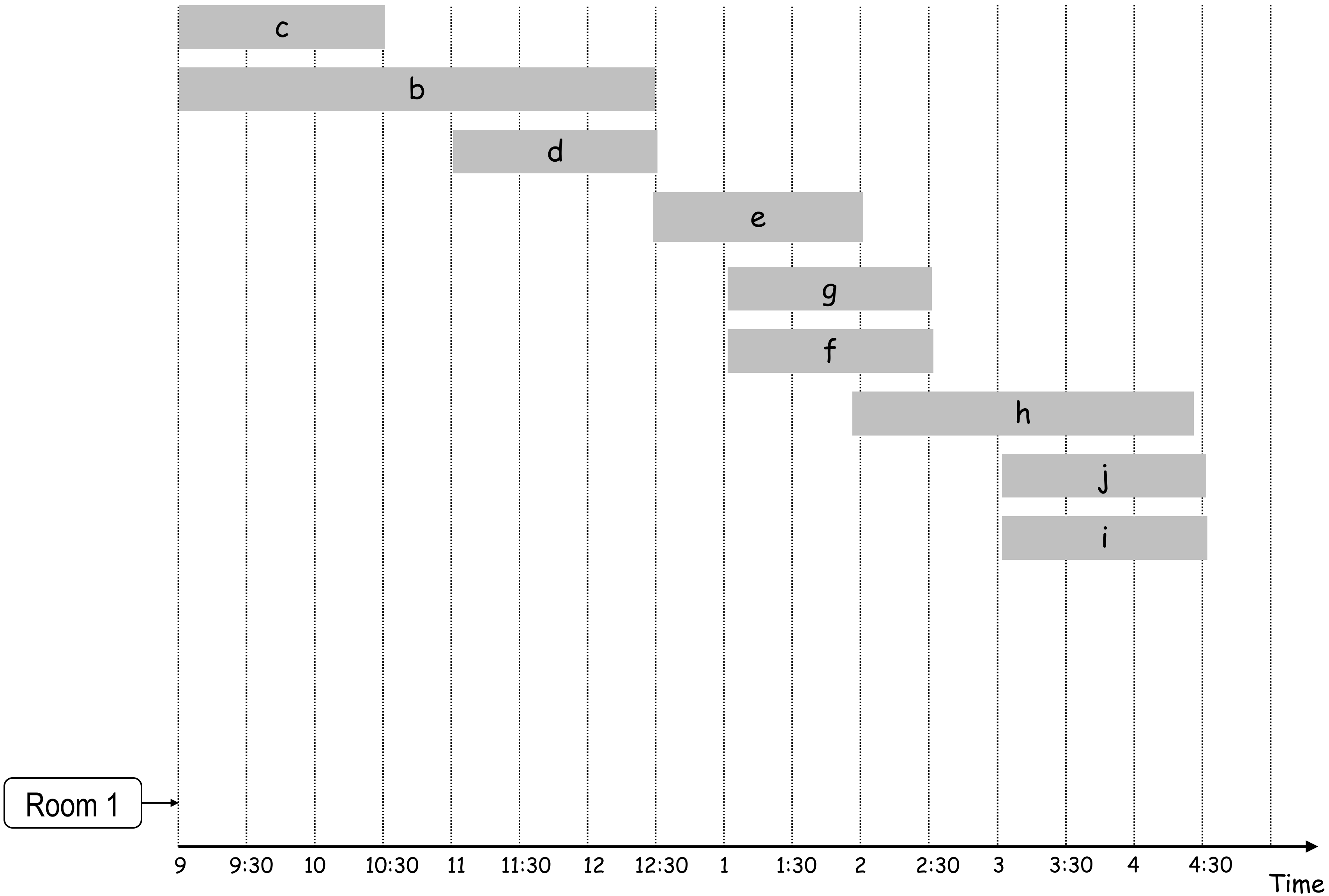
```
Sort intervals by starting time so that s1 ≤ s2 ≤ ... ≤ sn.
d= 0       ← number of allocated classrooms

for j = 1 to n {
   if (lect j is compatible with some classroom k, 1≤k≤d)
      schedule lecture j in classroom k
   else
      allocate a new classroom d + 1
      schedule lecture j in classroom d + 1
      d = d + 1
}
```

Implementation? Run-time?
Exercises

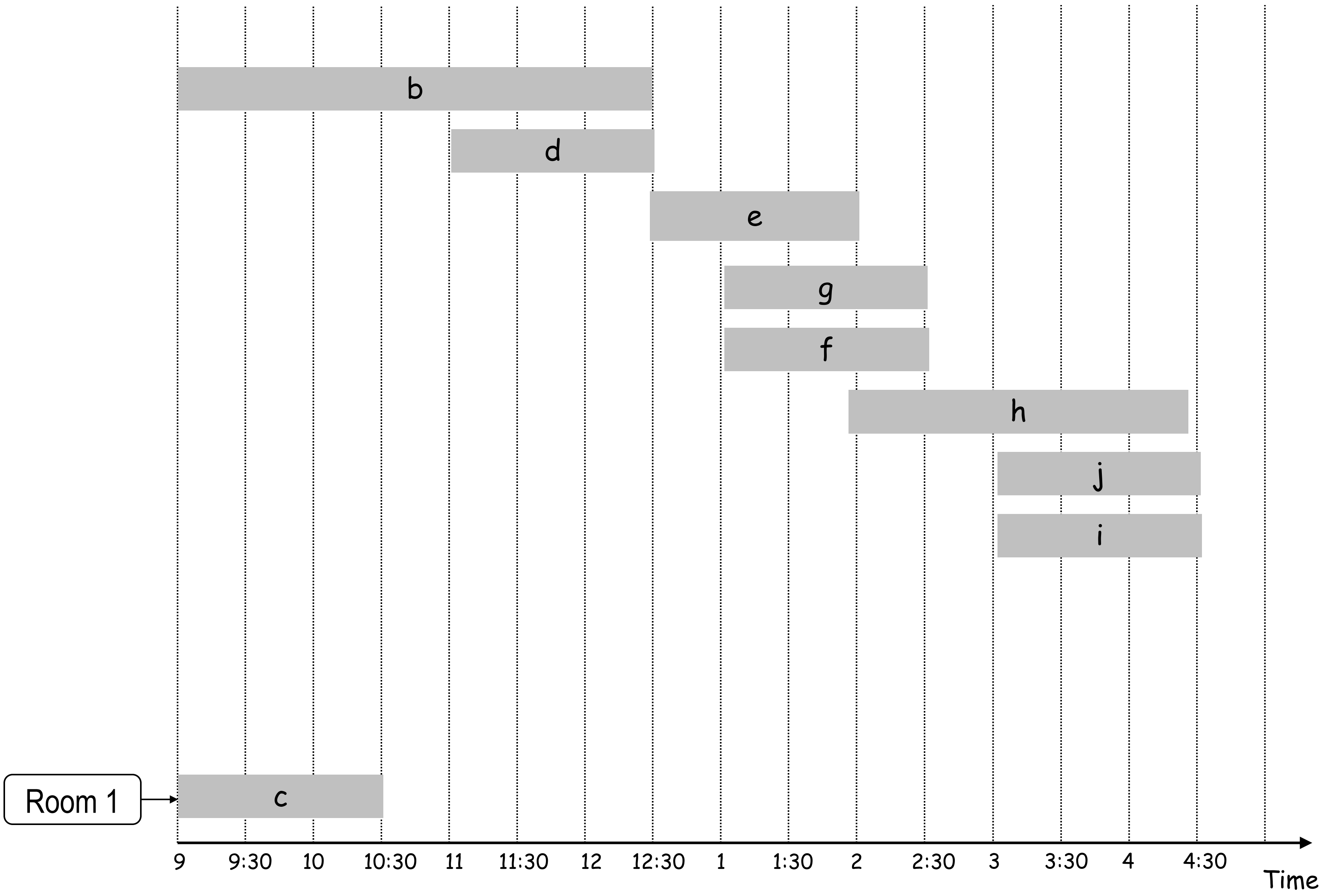
Interval Partitioning



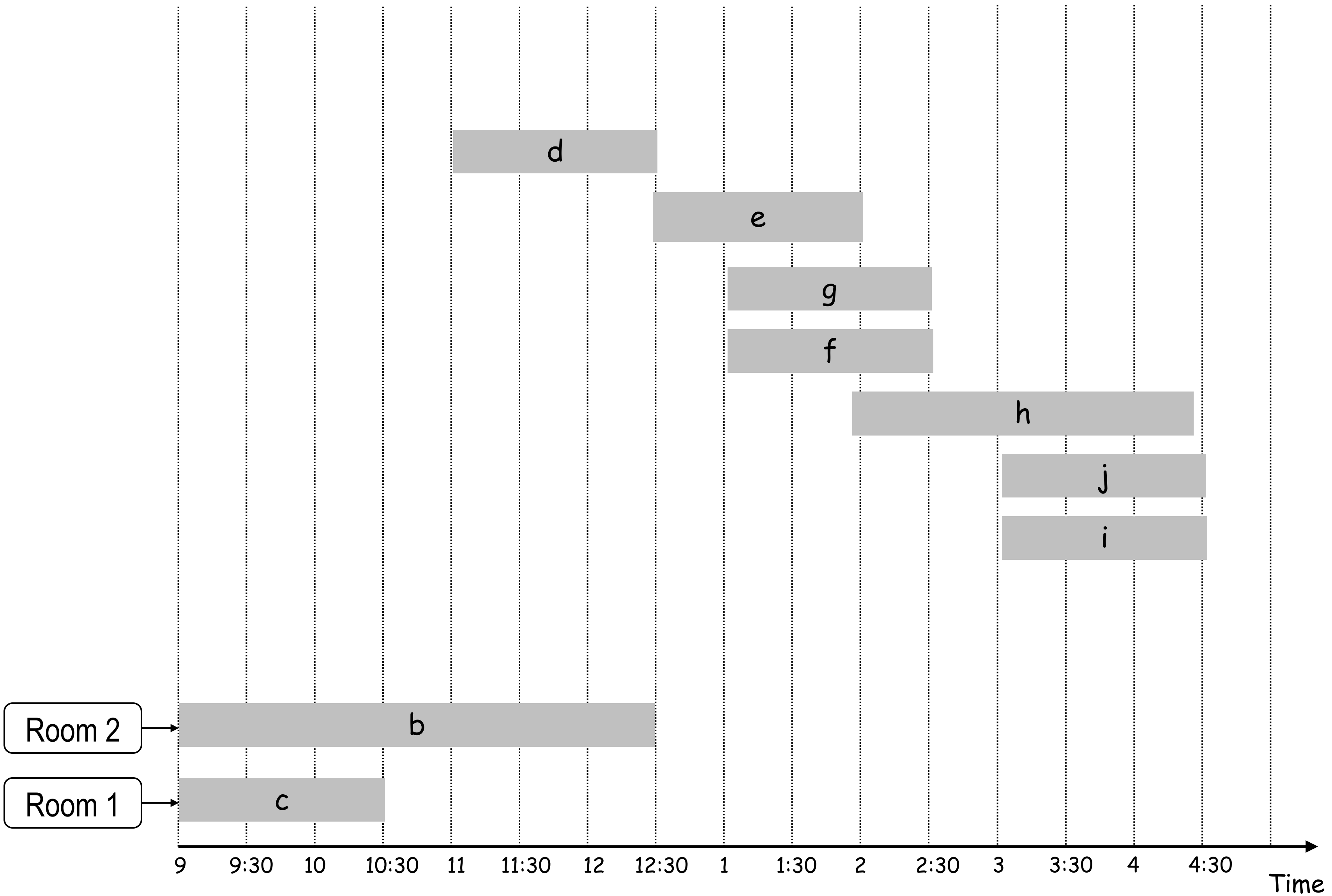
Interval Partitioning



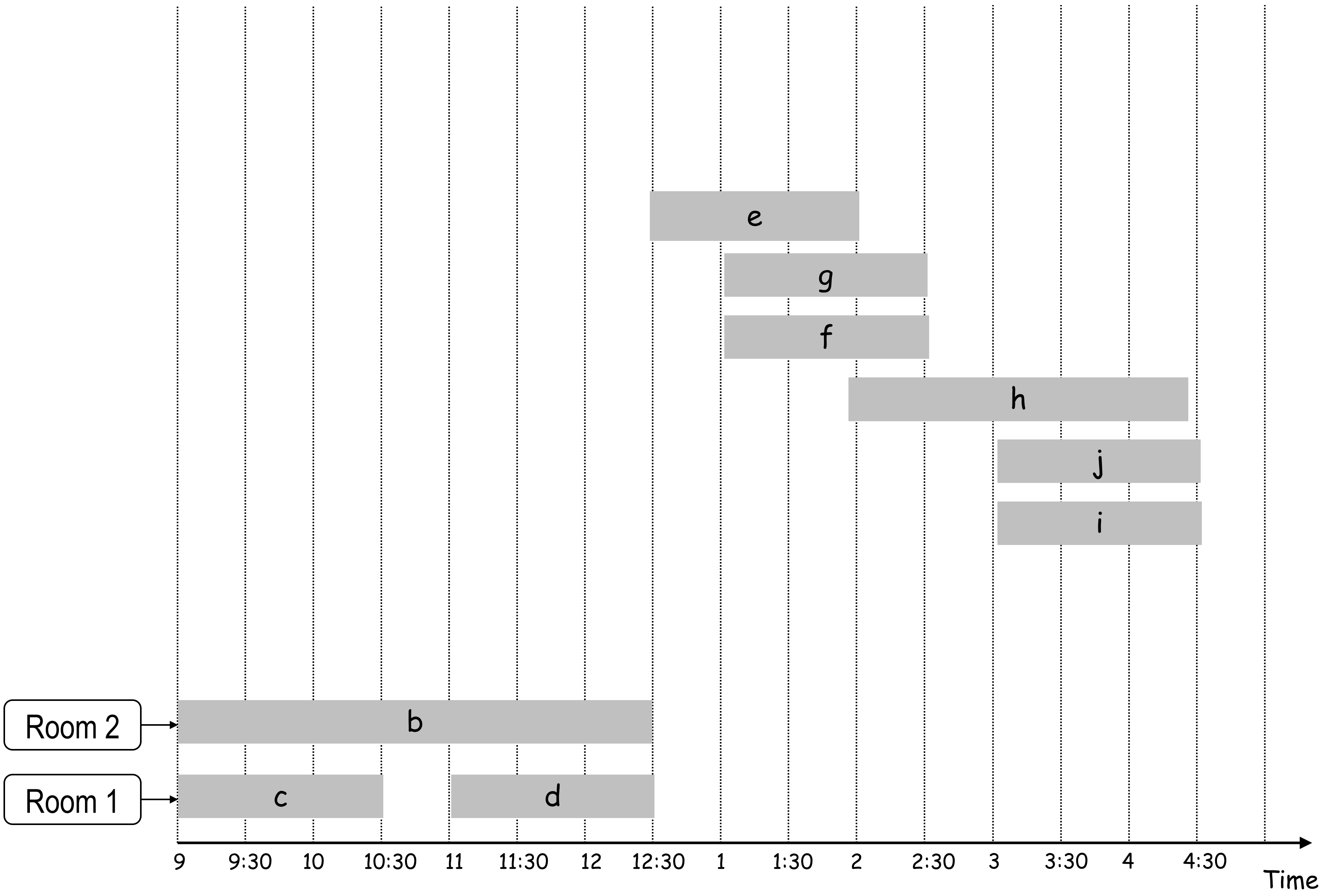
Interval Partitioning



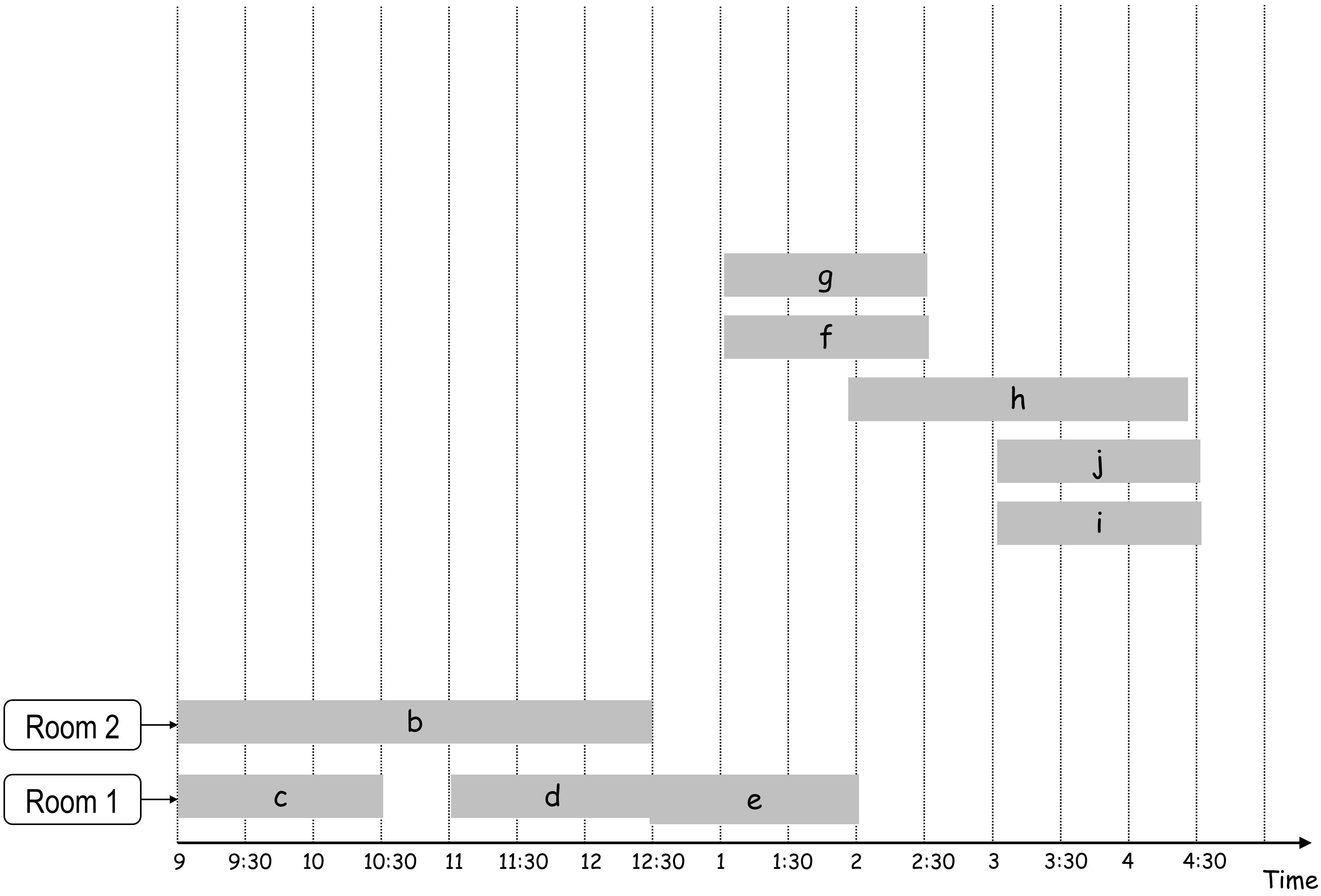
Interval Partitioning



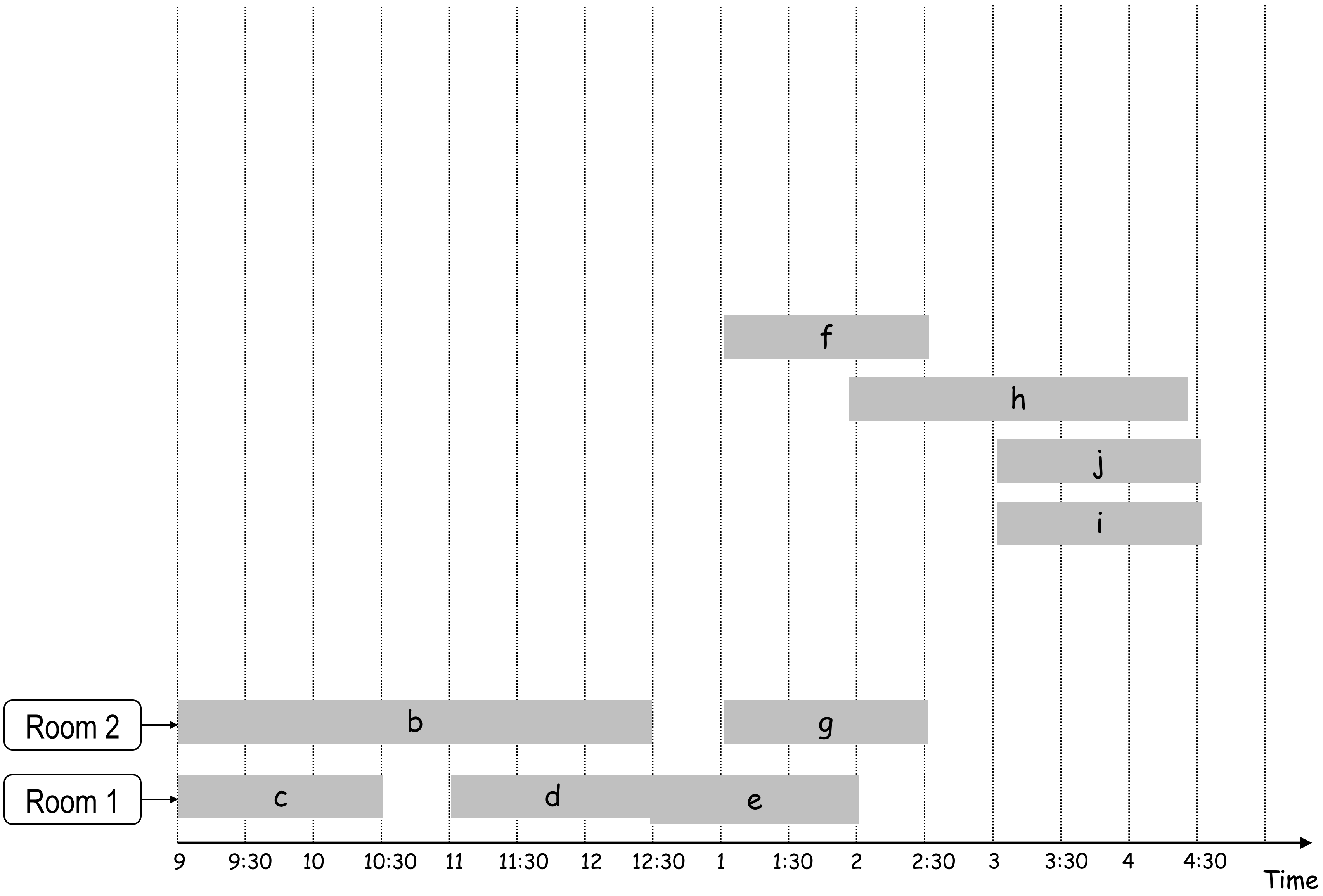
Interval Partitioning



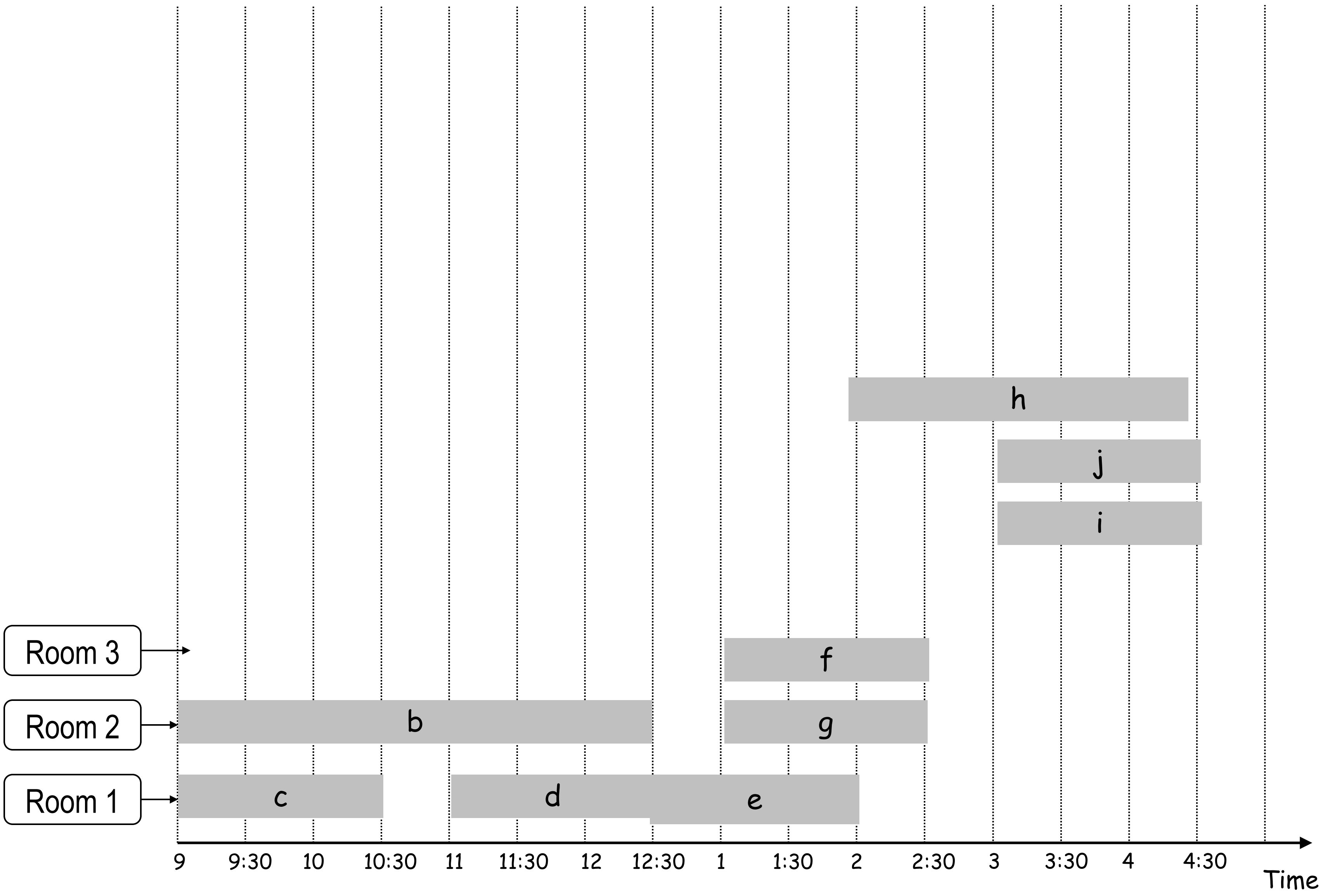
Interval Partitioning



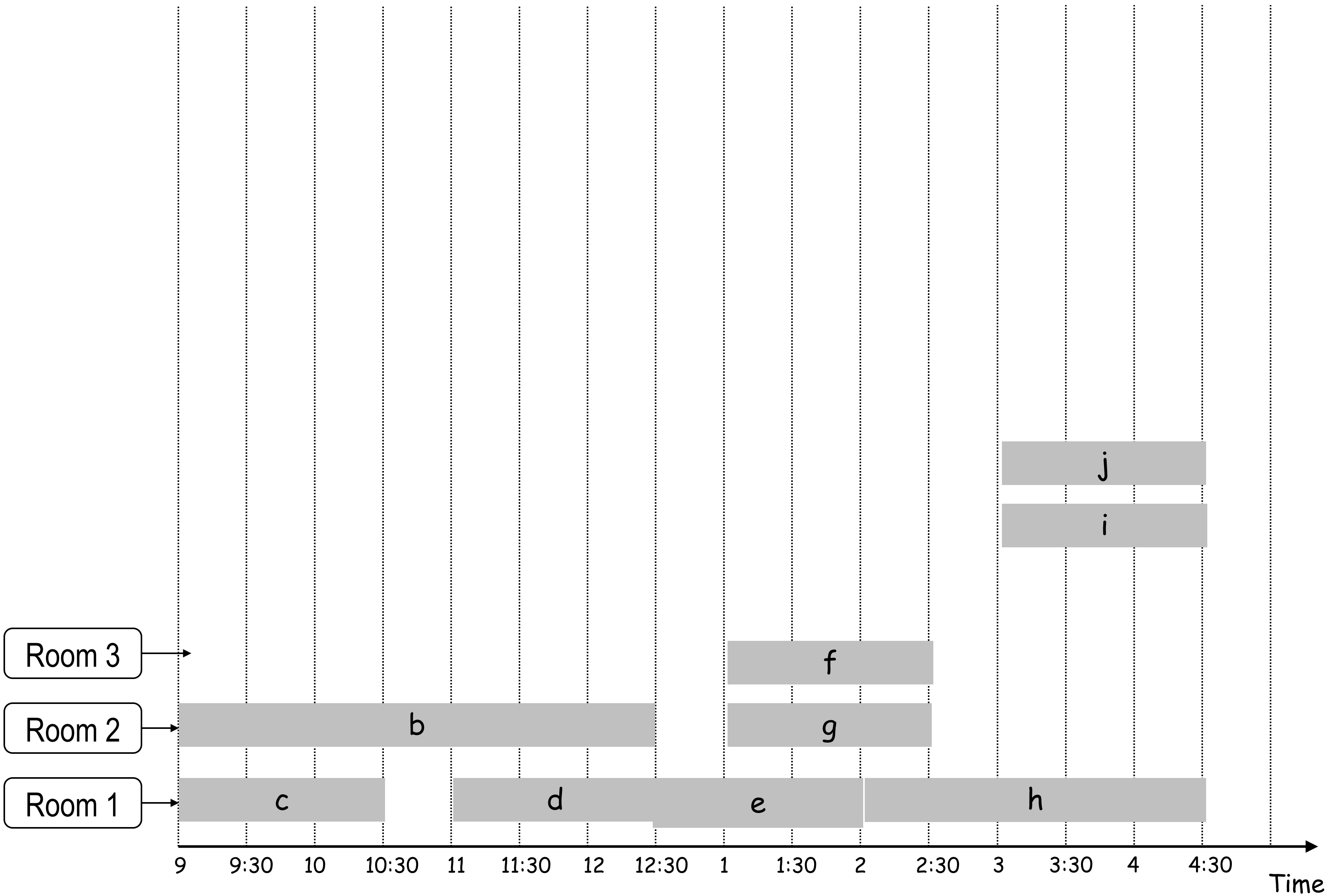
Interval Partitioning



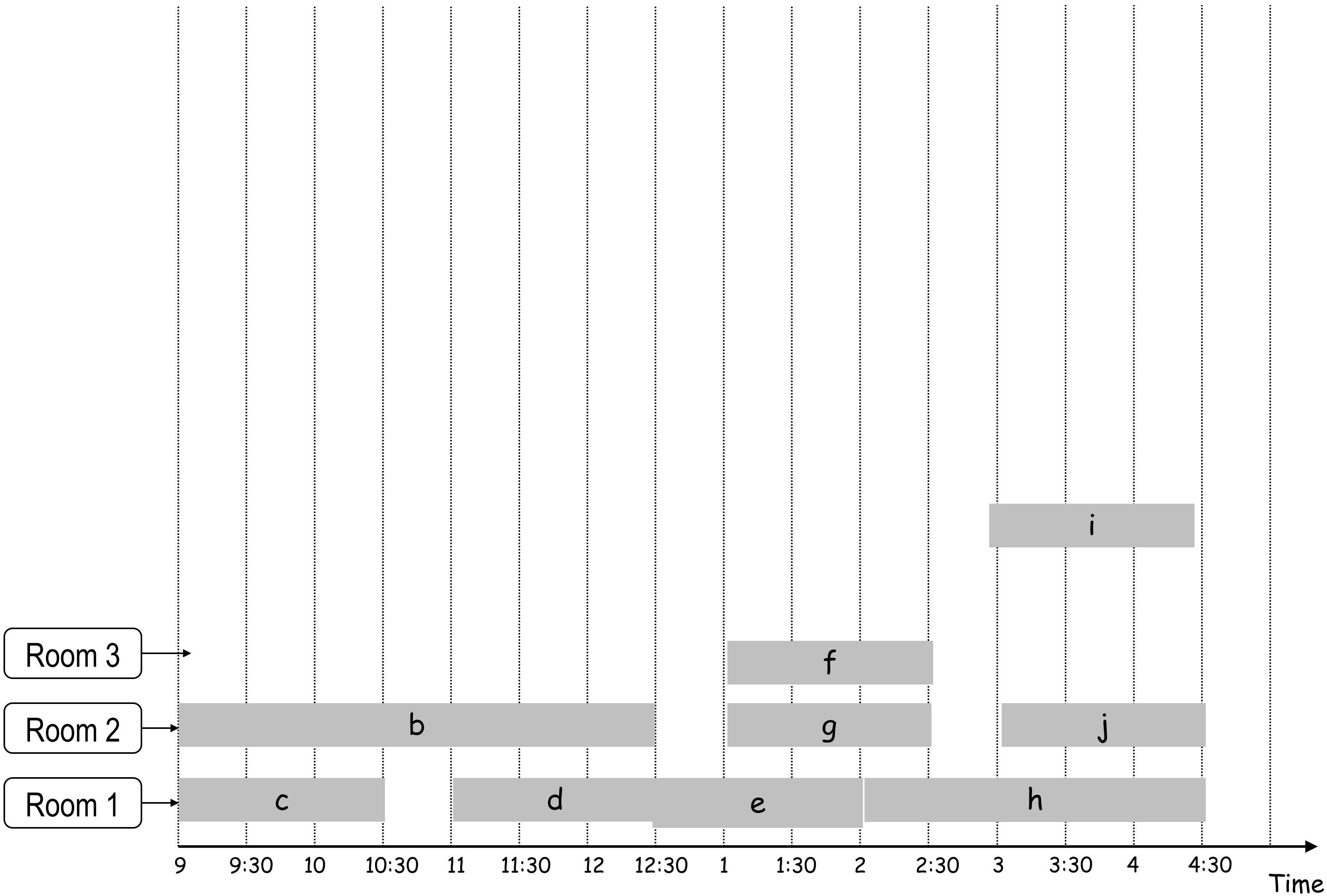
Interval Partitioning



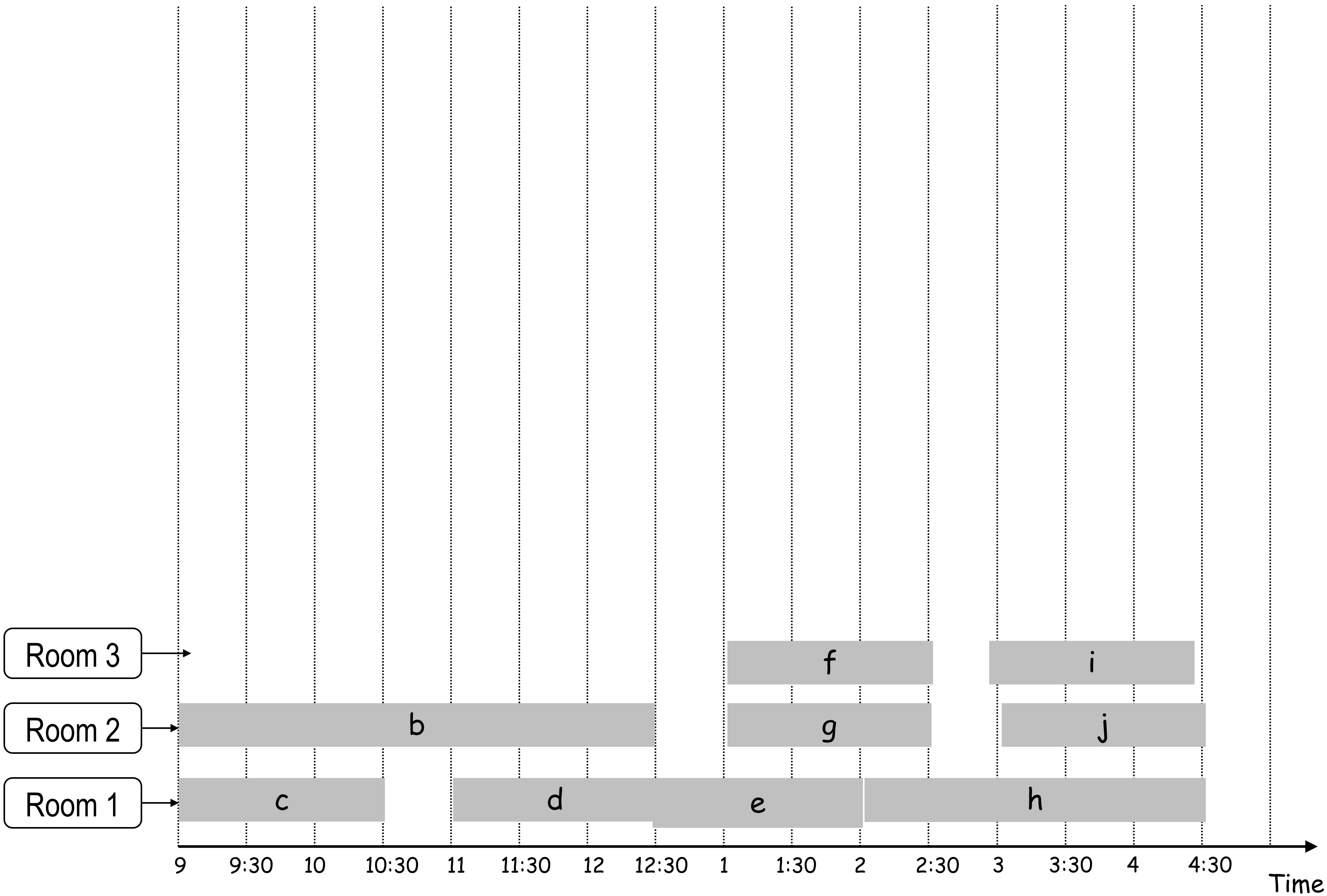
Interval Partitioning



Interval Partitioning



Interval Partitioning



Interval Partitioning: A "Structural" Lower Bound on Optimal Solution

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

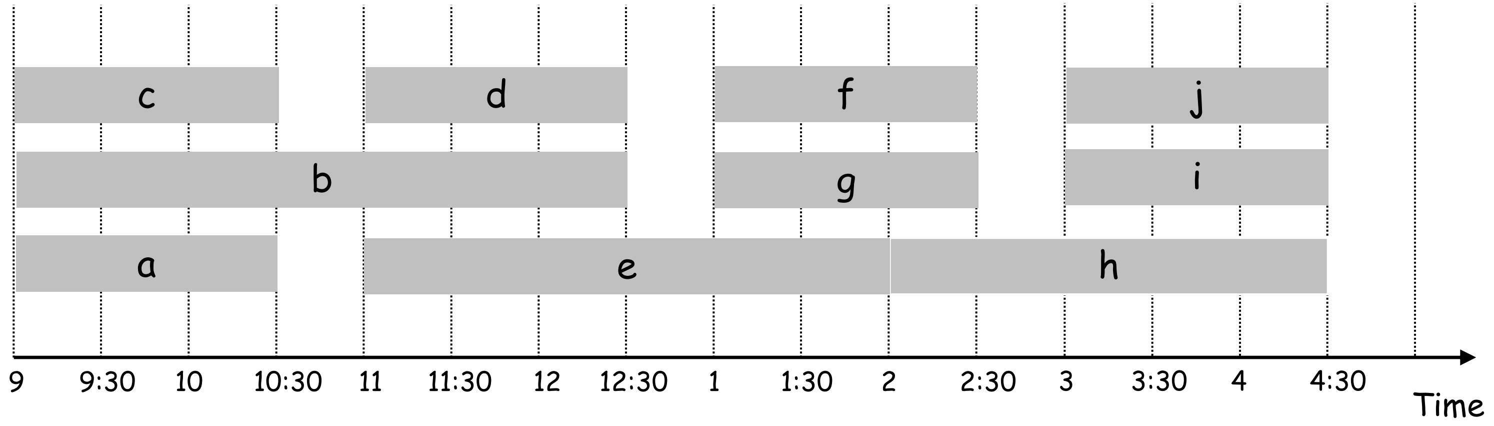
↑ no collisions at ends

Key observation. Number of classrooms needed ≥ depth.

Ex: Depth of schedule below = 3 ⇒ schedule below is optimal.

↑ a, b, c all contain 9:30

Q. Does there always exist a schedule equal to depth of intervals?



Interval Partitioning: Greedy Analysis

Theorem. Greedy algorithm is optimal.

Pf (exploit structural property).

- Let d = number of classrooms that the greedy algorithm allocates.
- Classroom d is opened because we needed to schedule a job, say j, that is incompatible with all d-1 previously used classrooms.
- Since we sorted by start time, all these incompatibilities are caused by lectures that start no later than s_j.
- Thus, we have d lectures overlapping at time s_j , i.e. depth ≥ d
- "Key observation" all schedules use ≥ depth classrooms, so d = depth and greedy is optimal ▪