Summary of Lecture 1

- Euler Theorem
- Algorithm for Euler Circuit.
- Basic techniques for designing algorithms:
  greedy, divide-and-conquer, dynamic programming

Week 2: Greedy Algorithms

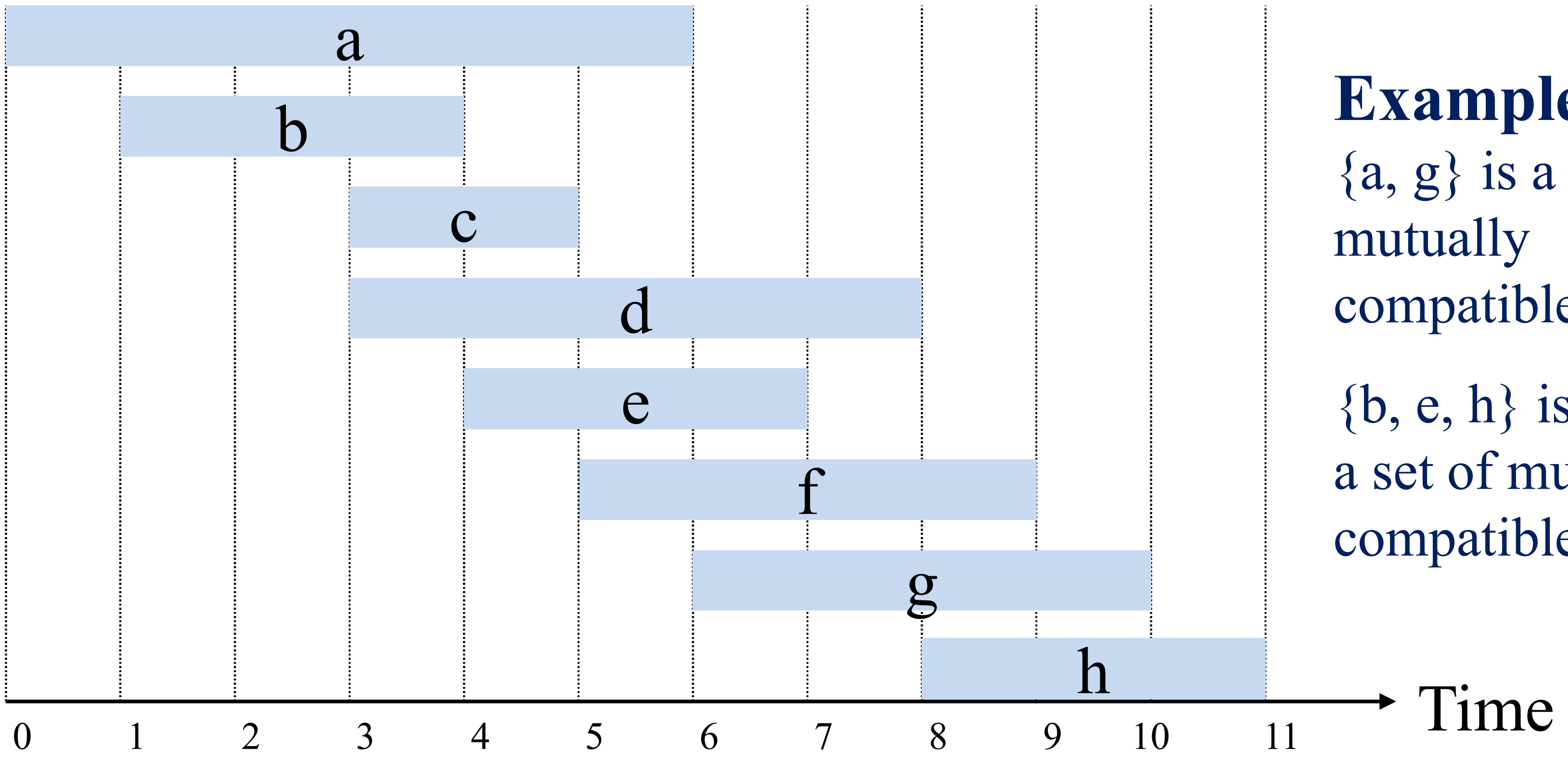
Greedy Algorithm

- A technique to solve problems:
  - always makes the locally best choice at the moment (local optimal).
  - Hopefully, a series of locally best choices will lead to a globally best solution.
- Greedy algorithms yield optimal solutions for many (but not all) problems.

Interval Scheduling

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.



Examples:

{a, g} is a set of mutually compatible jobs.

{b, e, h} is also a set of mutually compatible jobs.

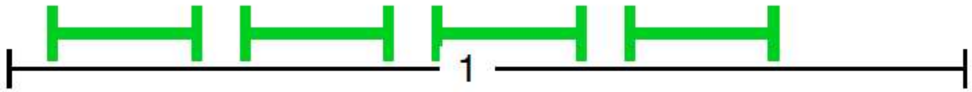
Ideas for Interval Scheduling

Greedy Algorithm: sort all the jobs in a list using a ‘greedy’ principle, and then choose it one by one

- What are possible rules for greedy sorting?
  - Choose the interval that starts earliest.
    - Rationale: start using the resource as soon as possible.
  - Choose the smallest interval.
    - Rationale: try to have lots of small jobs.
  - Choose the interval that overlaps (conflicts) with the fewest remaining intervals.
    - Rationale: keep our options open and eliminate as few intervals as possible.

Rules That Don't Work

1. Earliest start time

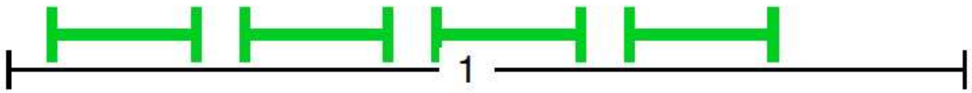


2. Shortest job

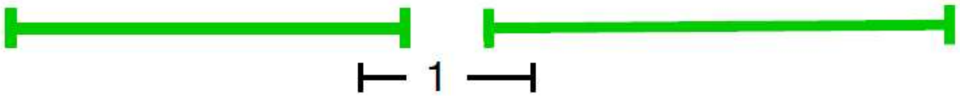
3. Fewest conflicts

Rules That Don't Work

1. Earliest start time



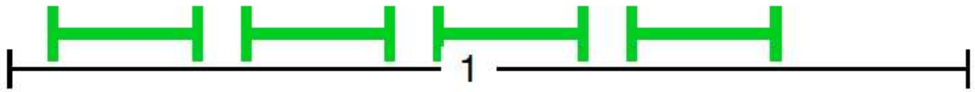
2. Shortest job



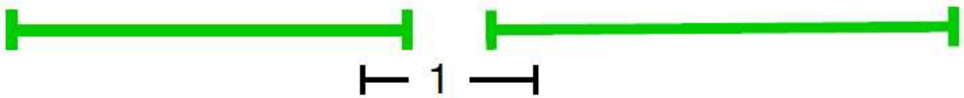
3. Fewest conflicts

Rules That Don't Work

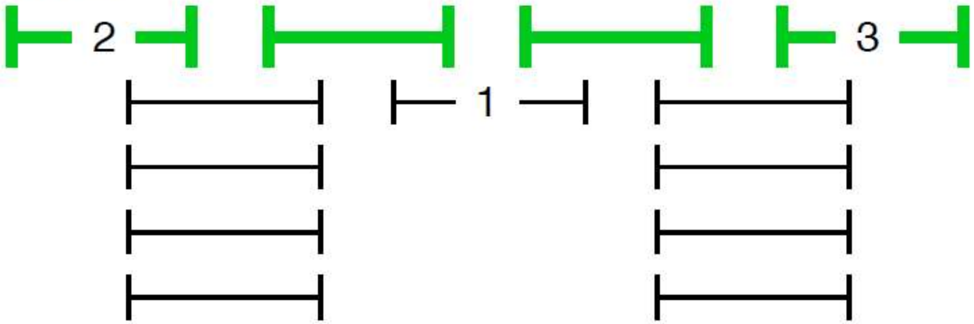
1. Earliest start time



2. Shortest job



3. Fewest conflicts



Example: sort based on finish time

- Jobs (s, f): (0, 10), (3, 4), (2, 8), (1, 5), (4, 5), (4, 8), (5, 6) (7,9).
- Sorting based on f_i:
  - (3, 4) (1, 5), (4, 5) (5, 6) (4,8) (2,8) (7, 9)(0,10).
- Selecting jobs:
  - (3, 4),
  - (4, 5),
  - (5, 6),
  - (7, 9),

Interval Scheduling: Greedy Algorithm

- Greedy algorithm. Consider jobs in increasing order of finishing 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 ← φ
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 $s_j \geq f_{j^*}$.

how to decide if job j is compatible with A

How to Prove Optimality

- How can we prove the schedule returned is optimal?
  - Let A be the schedule returned by this algorithm.
  - Let OPT be some optimal solution (there may be many optimal solutions!).

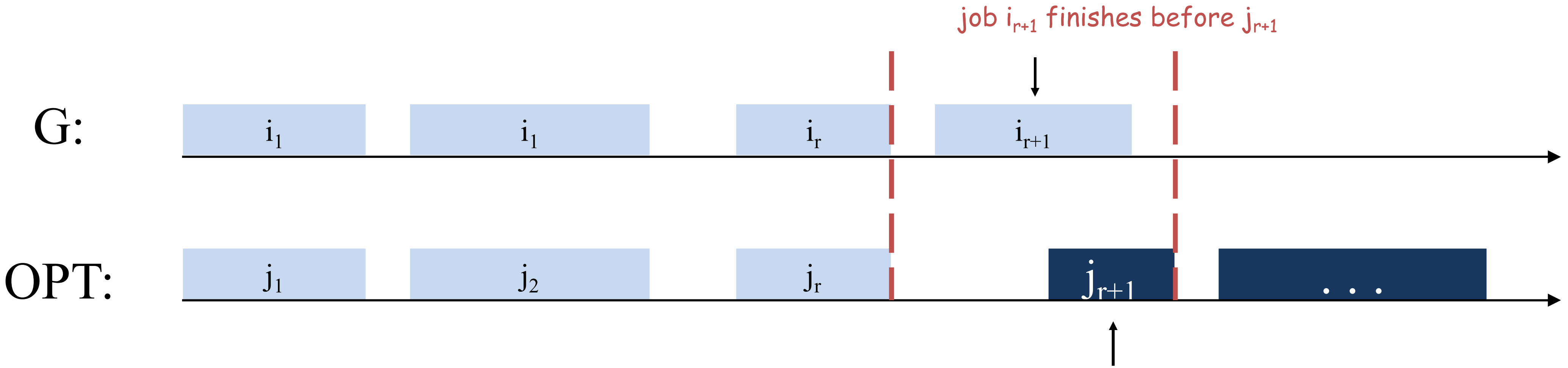
- Might be hard to show that A = OPT, instead we need only to show that |A| = |OPT| or equivalently A is one of the optimal solutions.

- **Note the distinction**: instead of proving directly that a choice of intervals A is the same as an optimal choice, we prove that it has the same number of intervals as an optimal. Therefore, it is optimal.

Interval Scheduling: Analysis

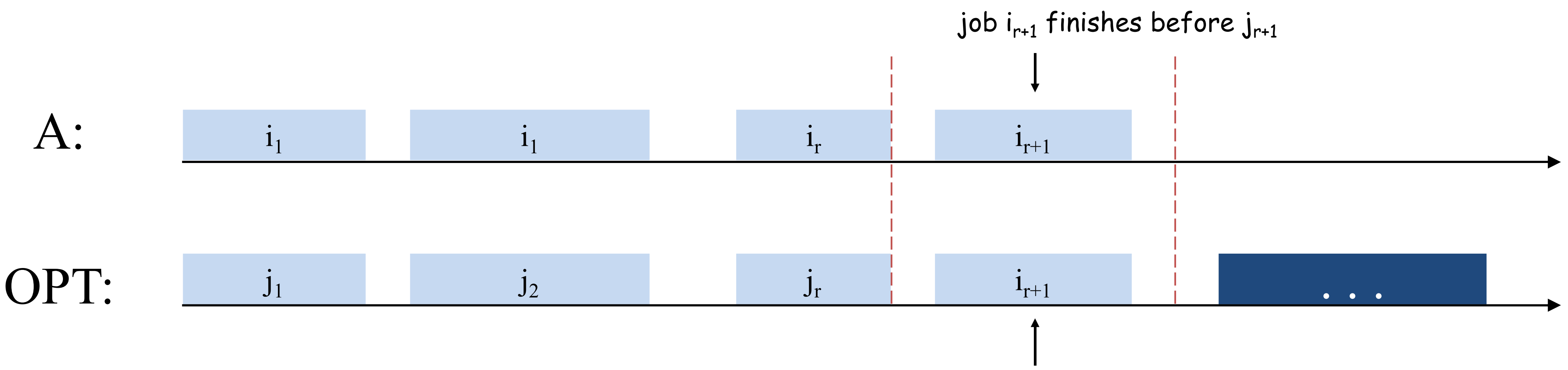
- Theorem. Greedy algorithm is optimal.
- Proof:
  - We compare the solution obtained from greedy algorithm with an optimal solution.
  - Let $G=i_1, i_2, \ldots, i_k$ denote the set of jobs selected by greedy.
  - Let Opt=$j_1, j_2, \ldots, j_n$ denote the set of jobs in the optimal solution.
    - The set of jobs are mutually compatible and the number of jobs is the largest.
  - Without loss of generality, we assume that

    $i_1=j_1, i_2=j_2, \ldots, i_r=j_r$ and $i_{r+1}\neq j_{r+1}$, where r could be 0, 1, 2, ….
  - Job i_{r+1} finishes before (or at the same time of) j_{r+1} due to our greedy algorithm.



Interval Scheduling: Analysis

- Theorem. Greedy algorithm is optimal.
- Pf.
  - Let A=i_1, i_2, ... i_k denote set of jobs selected by greedy.
  - Let Opt=j_1, j_2, ... j_m denote set of jobs in the optimal solution with $i_1 = j_1$, $i_2 = j_2$, ..., $i_r = j_r$ and $i_{r+1} \neq j_{r+1}$ (Note that r can be 0, 1, 2, ...)



Another Proof for Interval Scheduling:

Let: $G = i_1 i_2 \dots . i_m$ be the solution return by greedy

Let : $Opt = j_1 j_2 \dots . j_k$ be an optimal solution

Both are in the order of finish time.

1. If $i_1 \neq j_1$, since i_1 finishes the earliest, and thus not later than j_1, we can replace j_1 by i_1 in Opt. After this replacement, Opt is still optimal. Now

$$Opt = i_1 j_2 \dots j_k$$

2. If $i_2 \neq j_2$, since i_2 finishes earlier than j_2, we can replace j_2 by i_2 in Opt. After this replacement , Opt is still optimal. Now

$$Opt = i_1 i_2 j_3 \dots j_k$$

…….

After a number of replacements:

$$Opt = i_1 i_2 \dots i_m$$

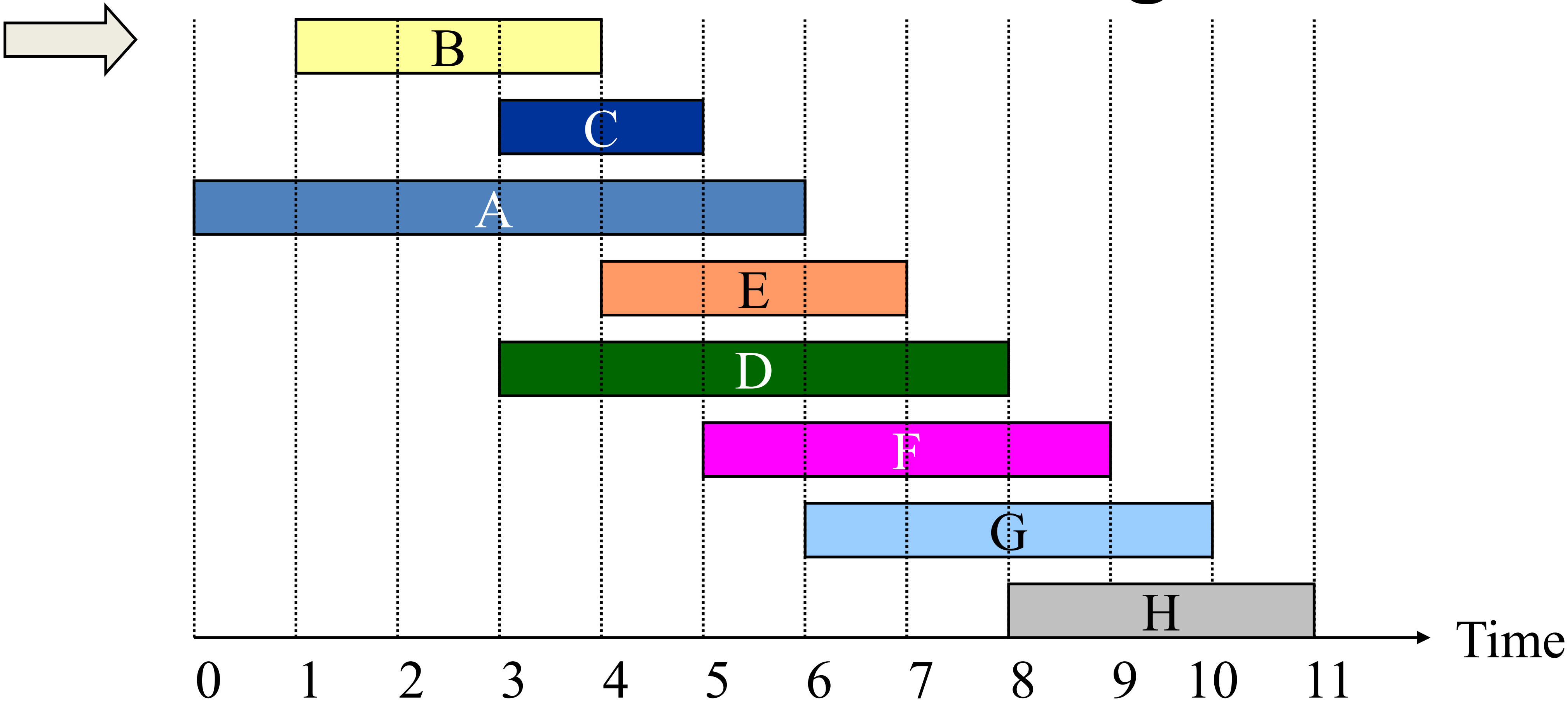
Or

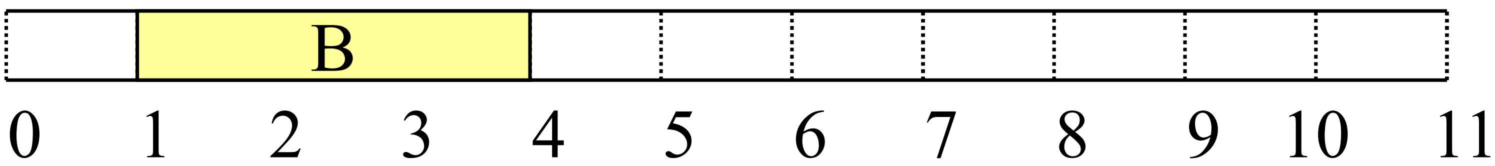
$$Opt = i_1 i_2 \dots i_m j_{m+1} \dots . j_k$$

The last case is impossible, since no job starts later than finish time of i_m.

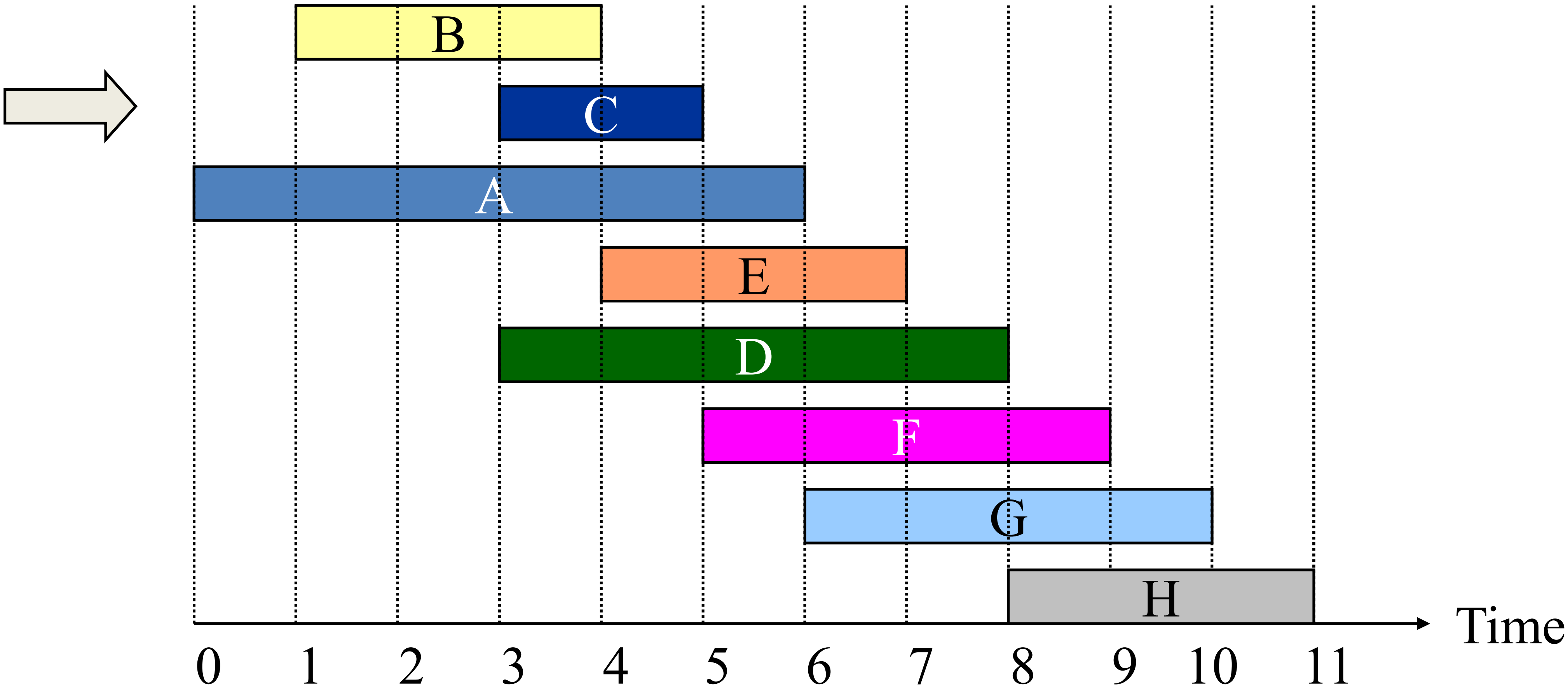
Therefore, G is optimal (it have the same number of jobs as Opt)

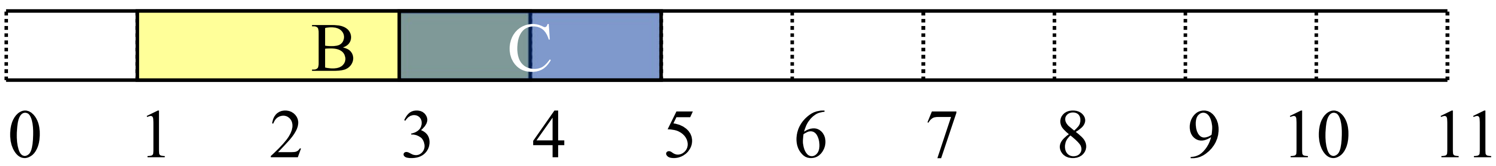
Interval Scheduling



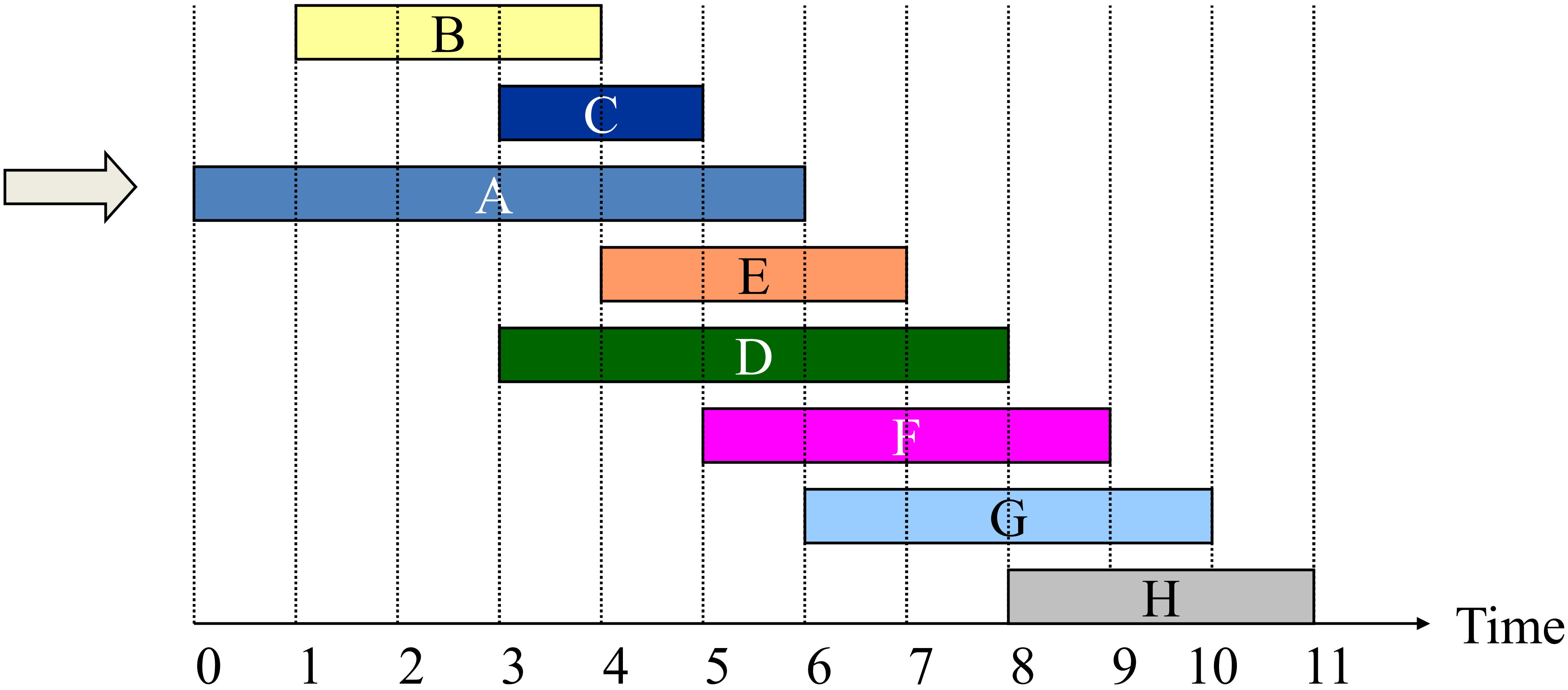


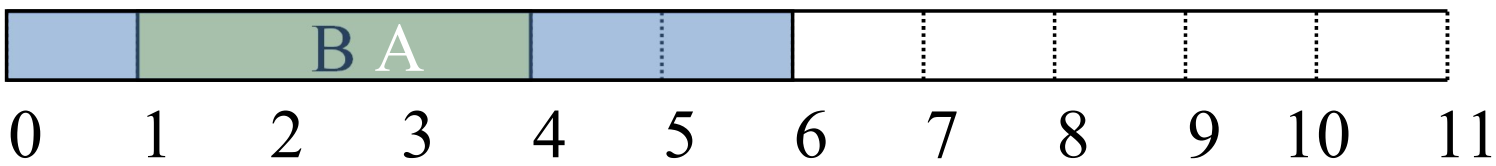
Interval Scheduling



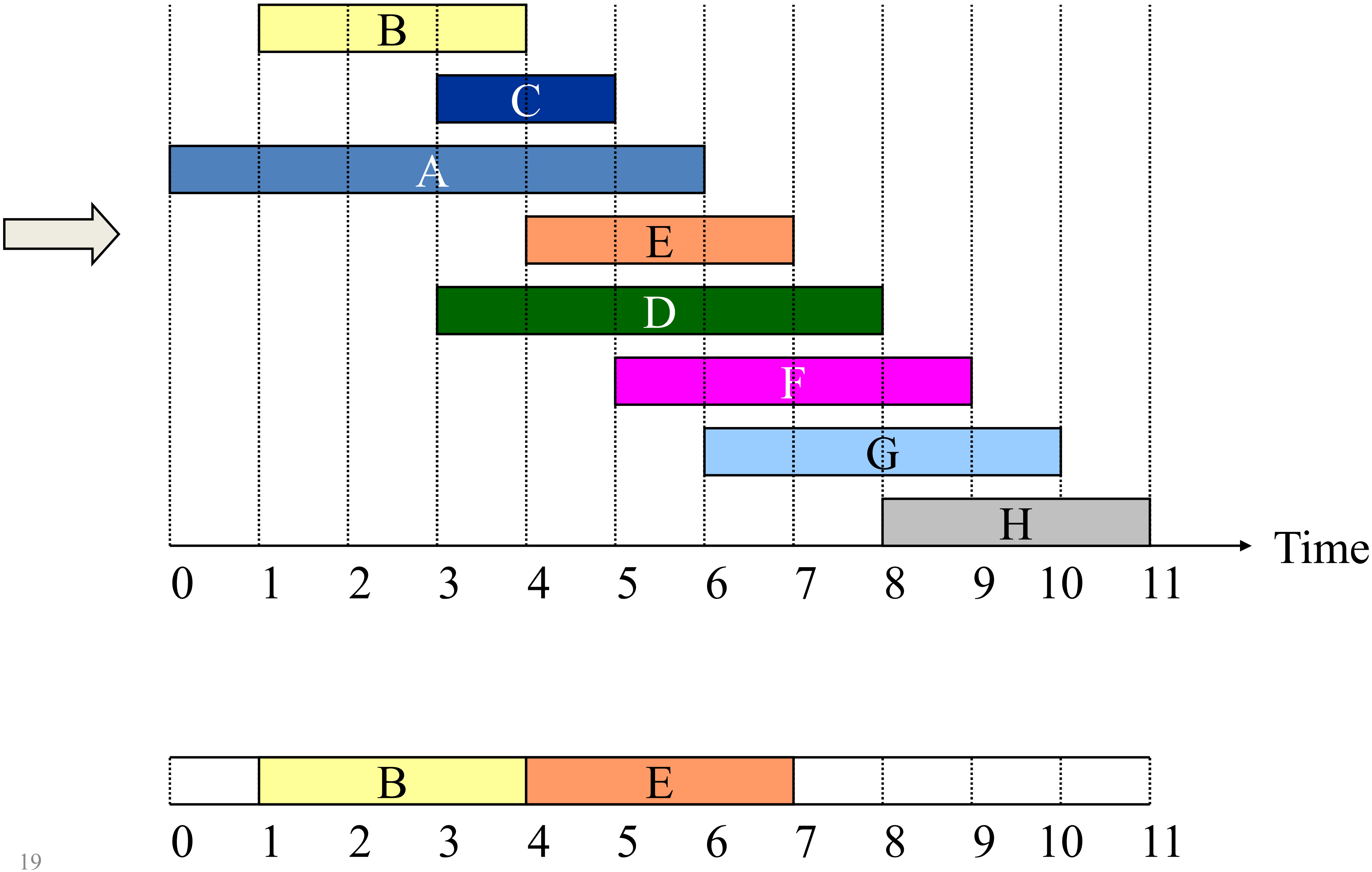


Interval Scheduling

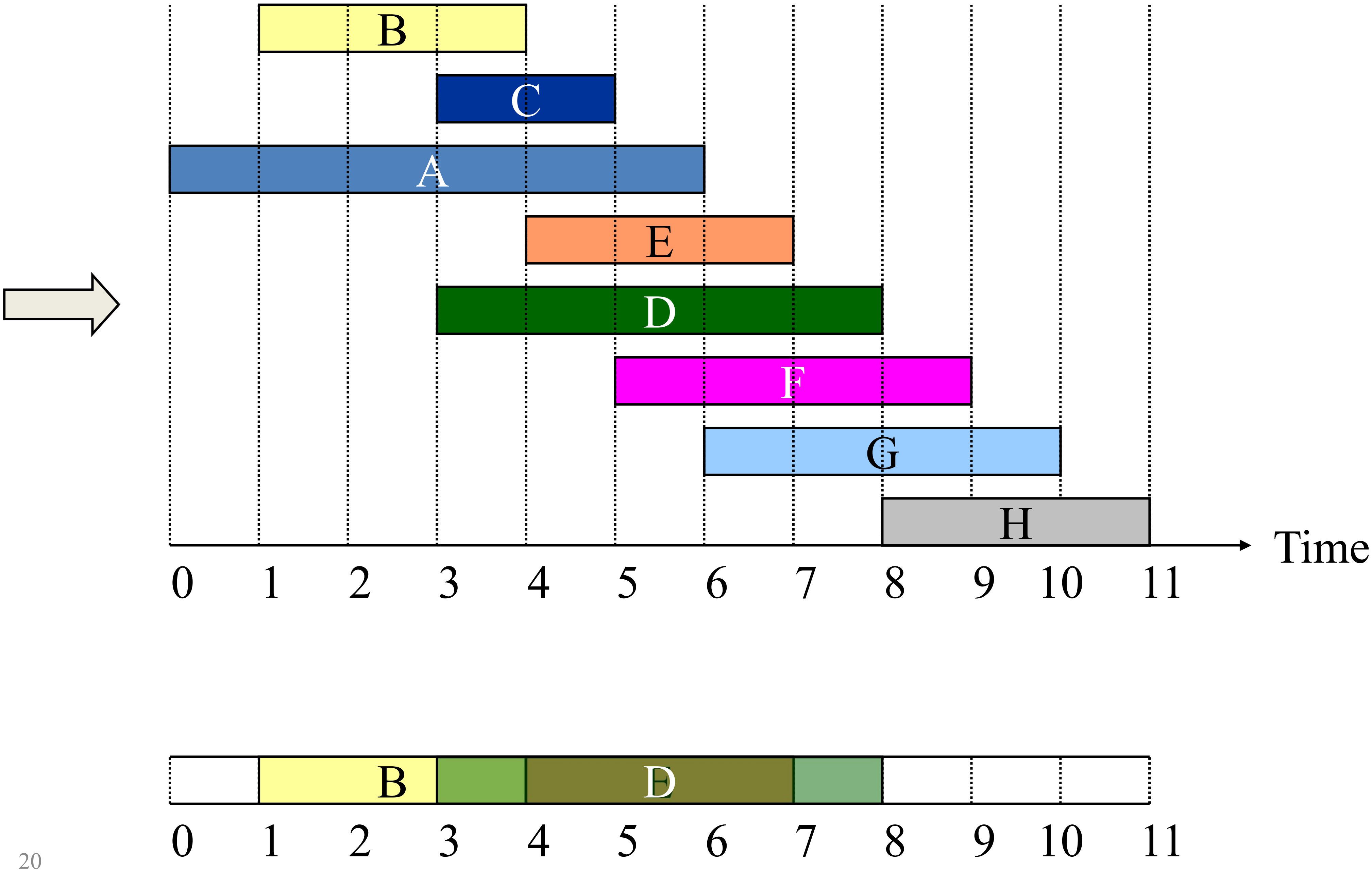




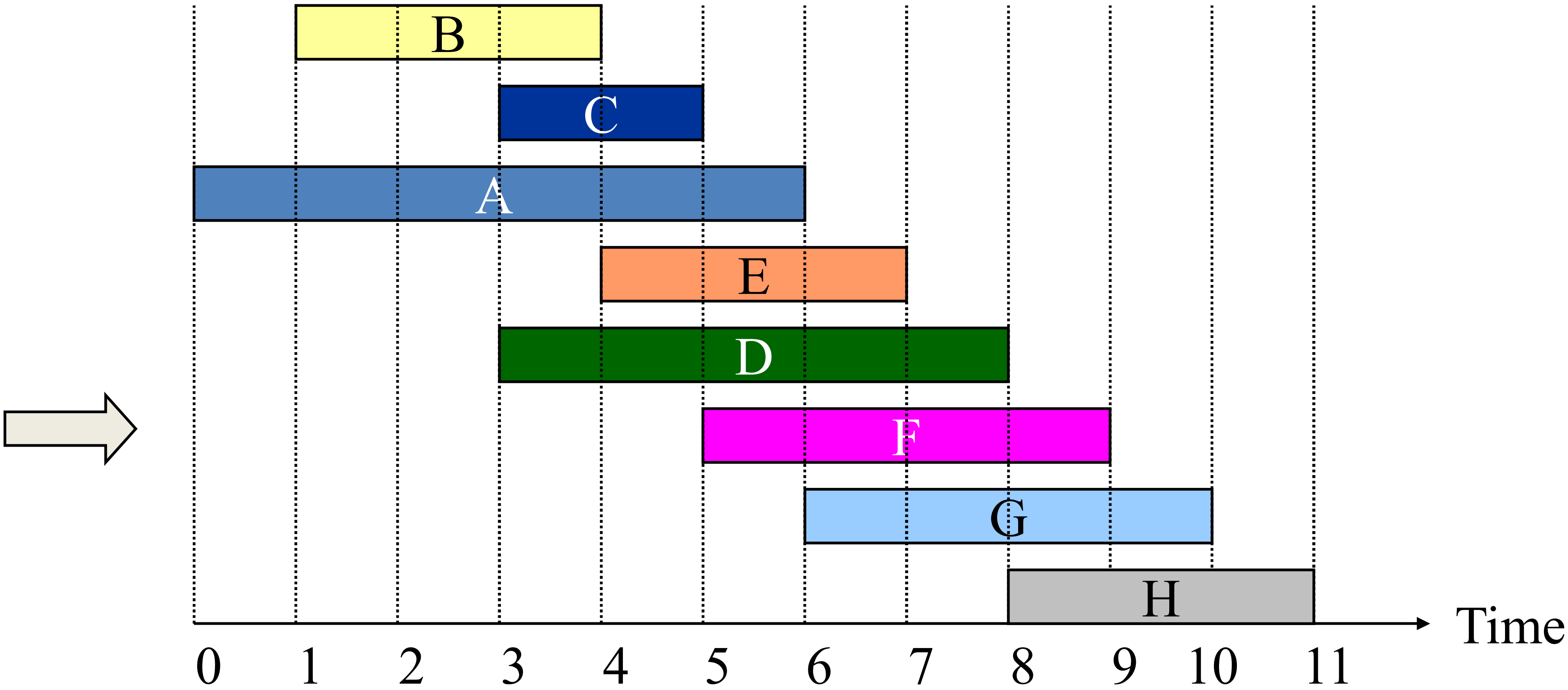
Interval Scheduling

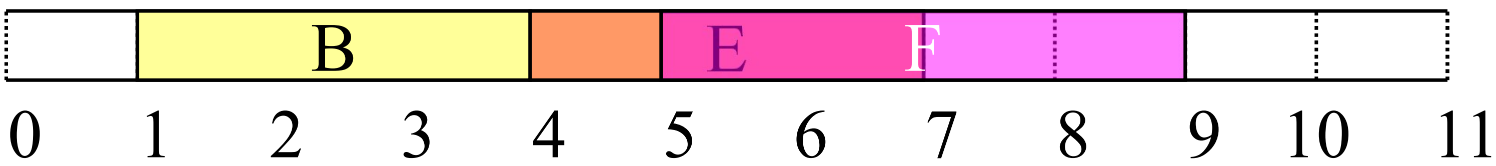


Interval Scheduling

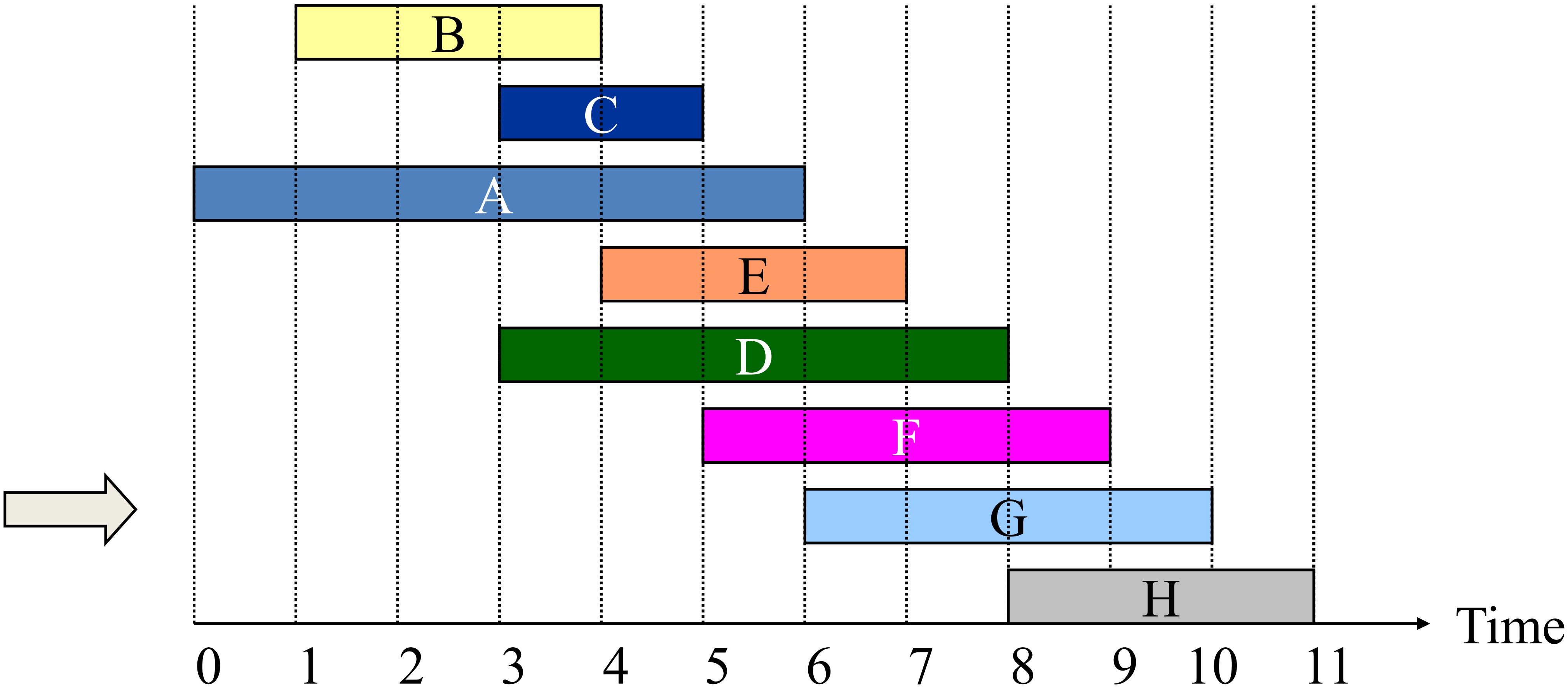


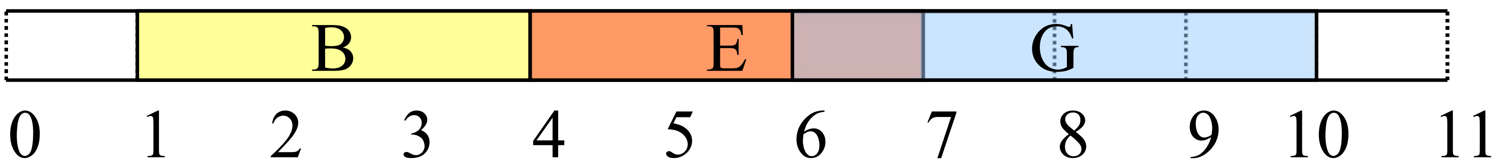
Interval Scheduling



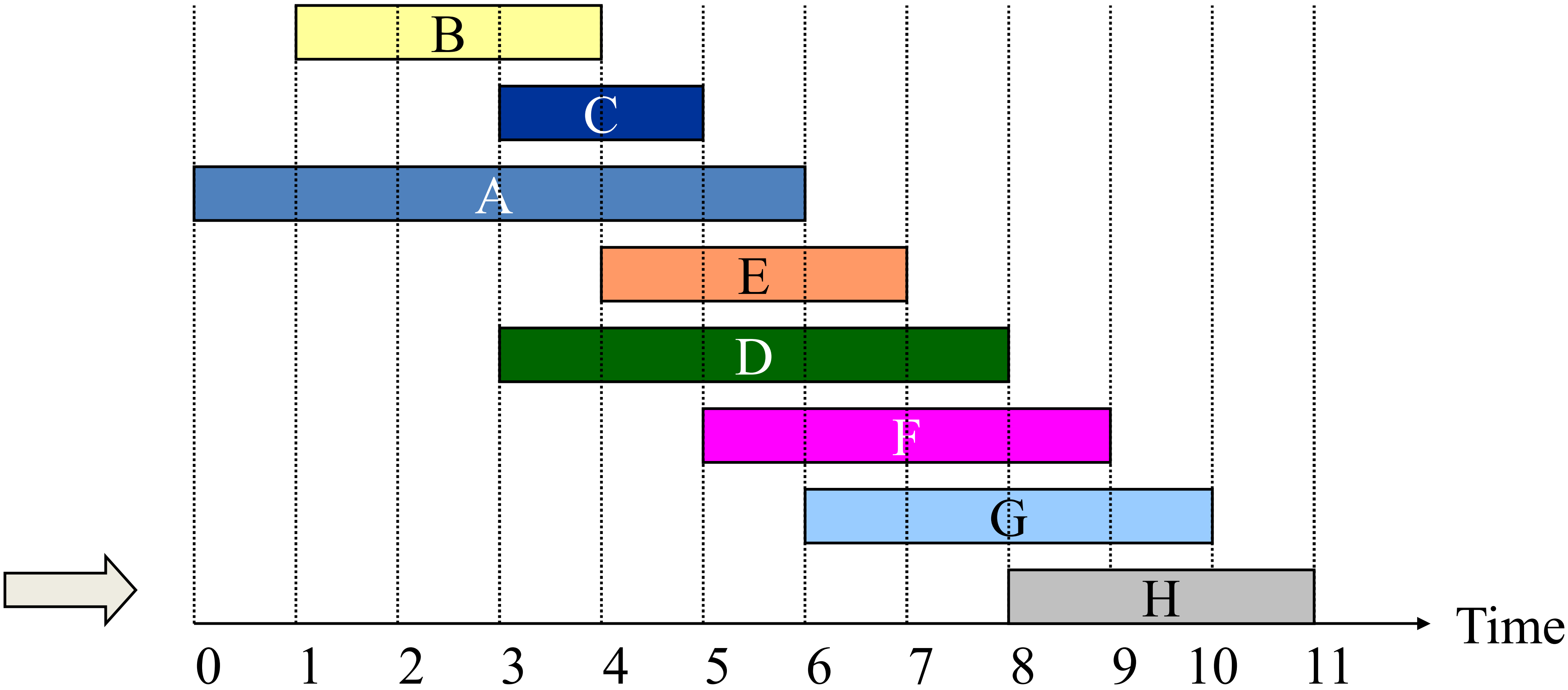


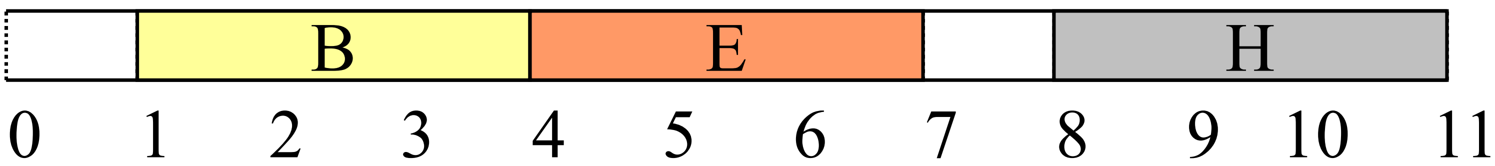
Interval Scheduling





Interval Scheduling



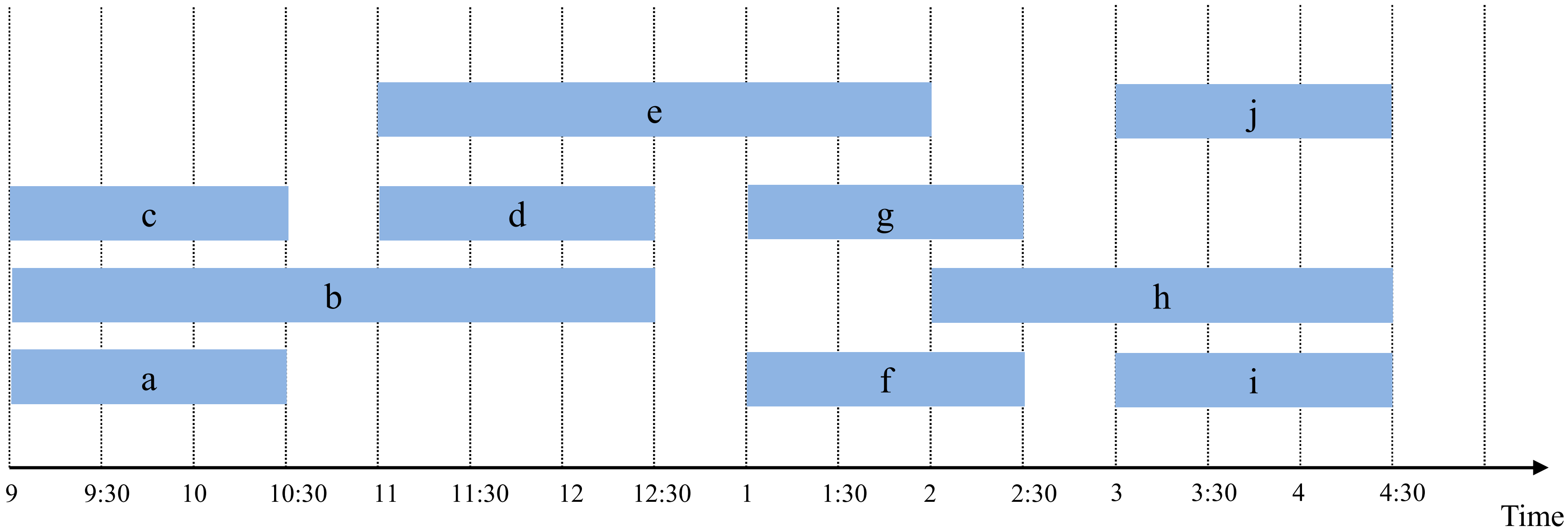


Interval Partitioning

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.

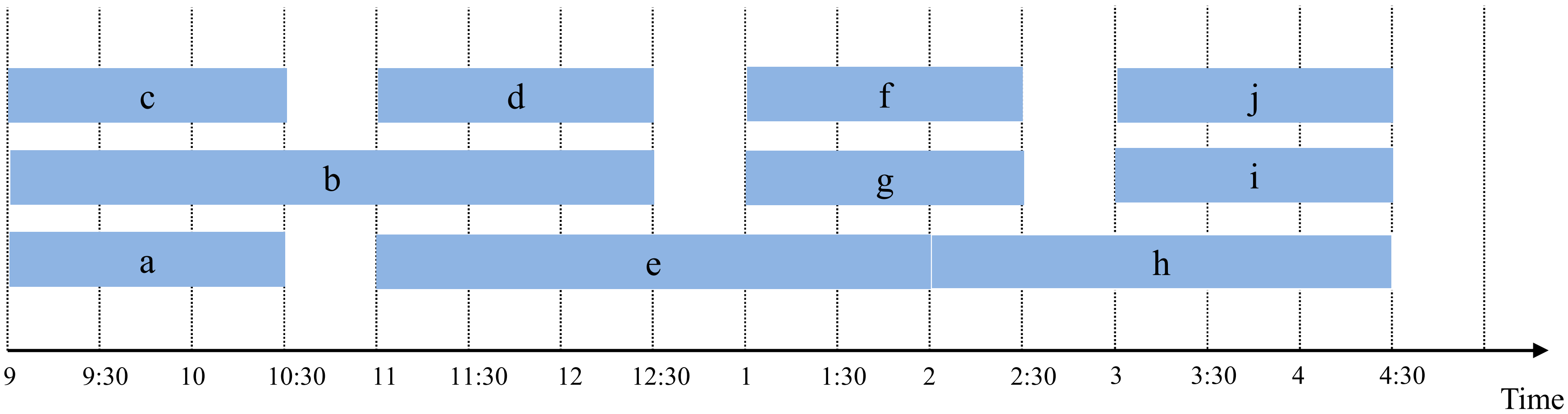
- Ex: This schedule uses 4 classrooms to schedule 10 lectures.



Interval Partitioning

- 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.

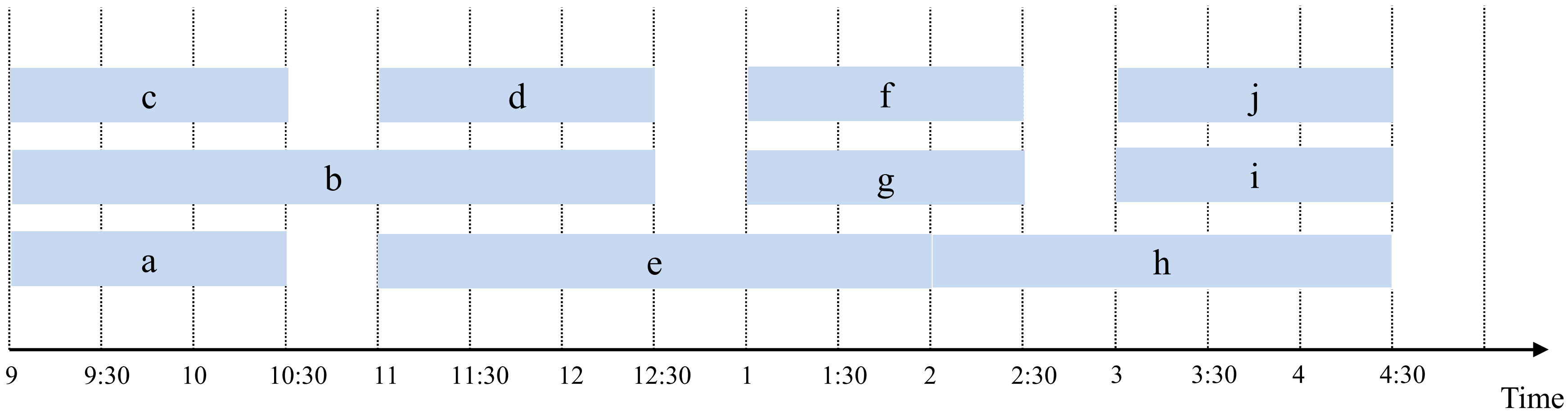


Interval Partitioning: Lower Bound on Optimal Solution

- Def. The depth of a set of open intervals is the maximum number of overlapped lectures during the whole period.
- **Key observation**. Number of classrooms needed $\geq$ depth.
- **Ex**: Depth of schedule below = 3 $\Rightarrow$ schedule below is optimal.

a, b, c all contain 9:30

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



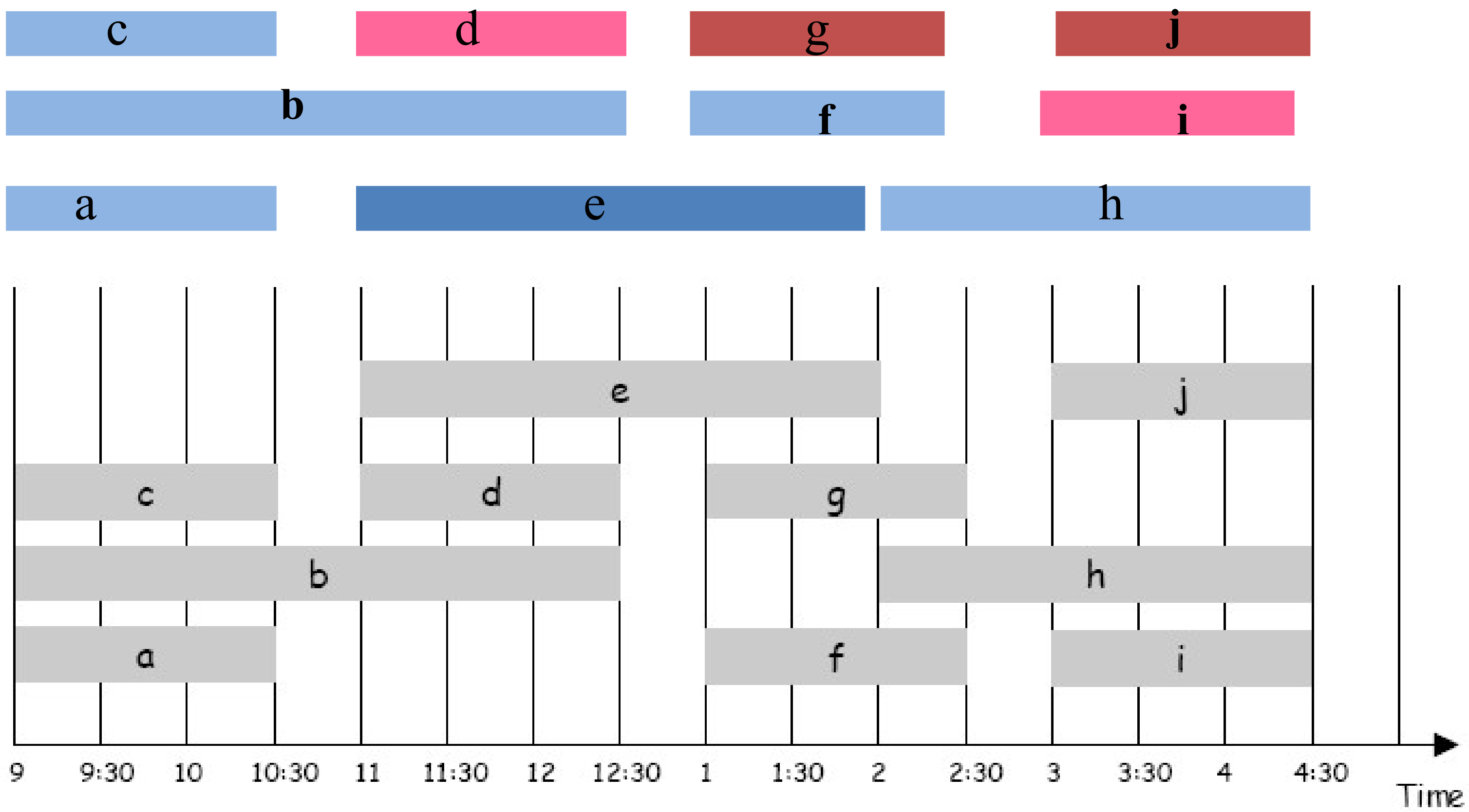
Interval Partitioning: Greedy Algorithm

- Greedy algorithm. Consider lectures in increasing order of start time: assign lecture to any compatible classroom (Don't open any new classroom unless necessary).

```
Sort intervals by starting time so that s1 ≤ s2 ≤ ... ≤ sn.
d ← 0  ← number of allocated classrooms
for j = 1 to n {
   if (lecture j is compatible with some classroom k)
      schedule lecture j in classroom k
   else
      allocate a new classroom d + 1
      schedule lecture j in classroom d + 1
      d ← d + 1
}
```

- Implementation. O(n log n).
  - For each classroom k, maintain the finish time of the last job added.
  - Keep the classrooms in a priority queue.

Greedy Algorithm:



Interval Partitioning: Greedy Analysis

- **Observation**. Greedy algorithm never schedules two incompatible lectures in the same classroom (its solution is always feasible).
- **Theorem**. Greedy algorithm is optimal.
- Pf.
  - 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 other 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 .
  - $d \leq depth$
  - Key observation $\Rightarrow$ all schedules need to use $\geq$ depth classrooms.

Greedy Analysis Strategies

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

- **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.