CS5800 – ALGORITHMS

MODULE 6. GREEDY ALGORITHMS - I

Lesson 2: Interval Scheduling

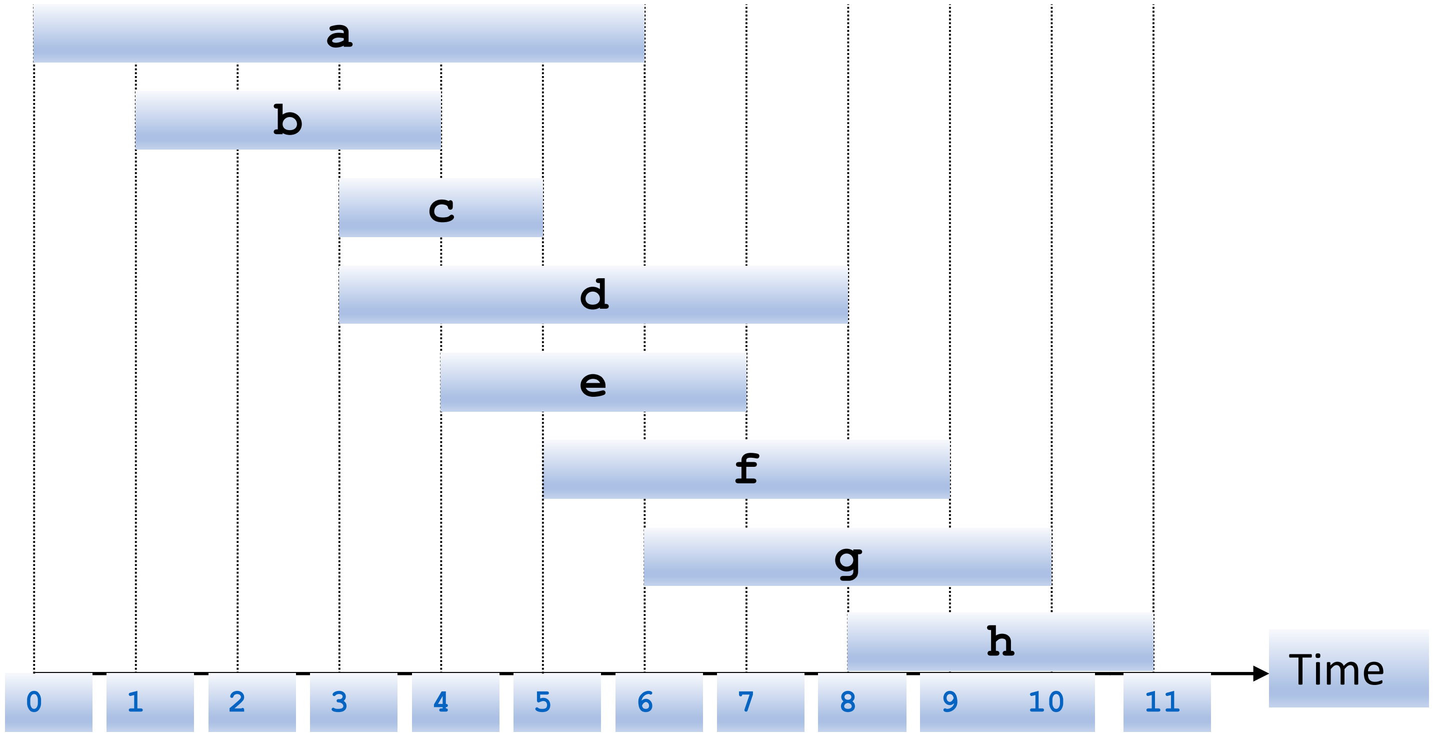
Ravi Sundaram

Topics

- Interval Scheduling
  - Problem
  - Naïve greedy
  - Algorithm
  - Demo
  - Proof
  - Efficiency
- Summary

Interval Scheduling - problem

- 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 Attempts

- 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

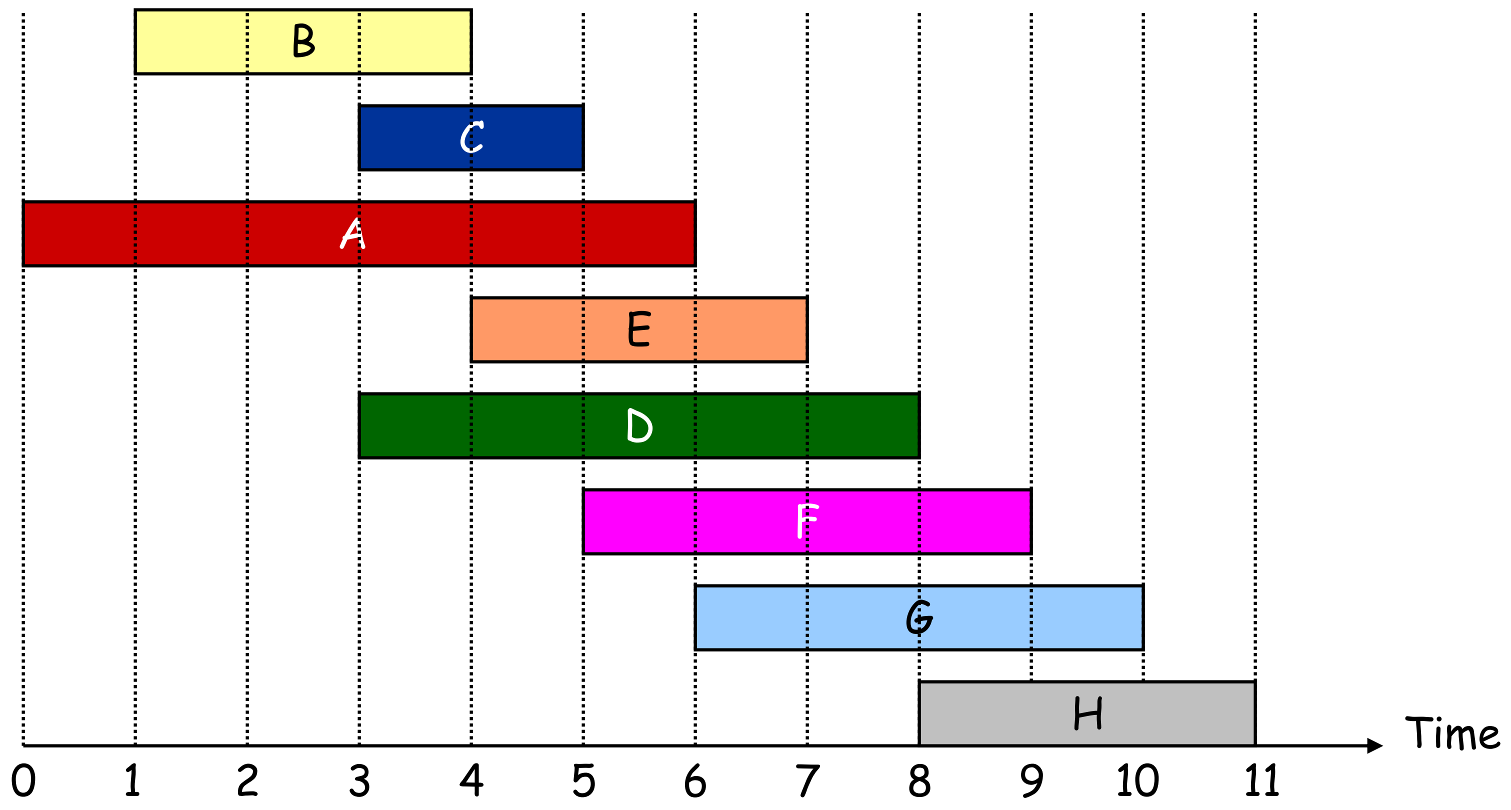
counterexample for fewest conflicts

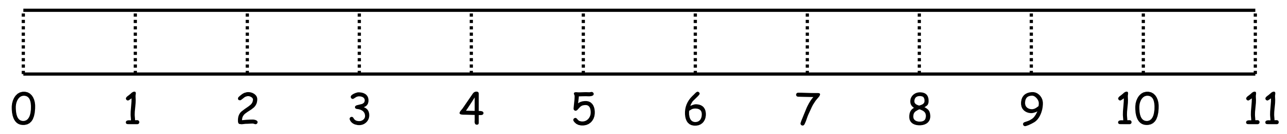
Interval Scheduling: Greedy Algorithm

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

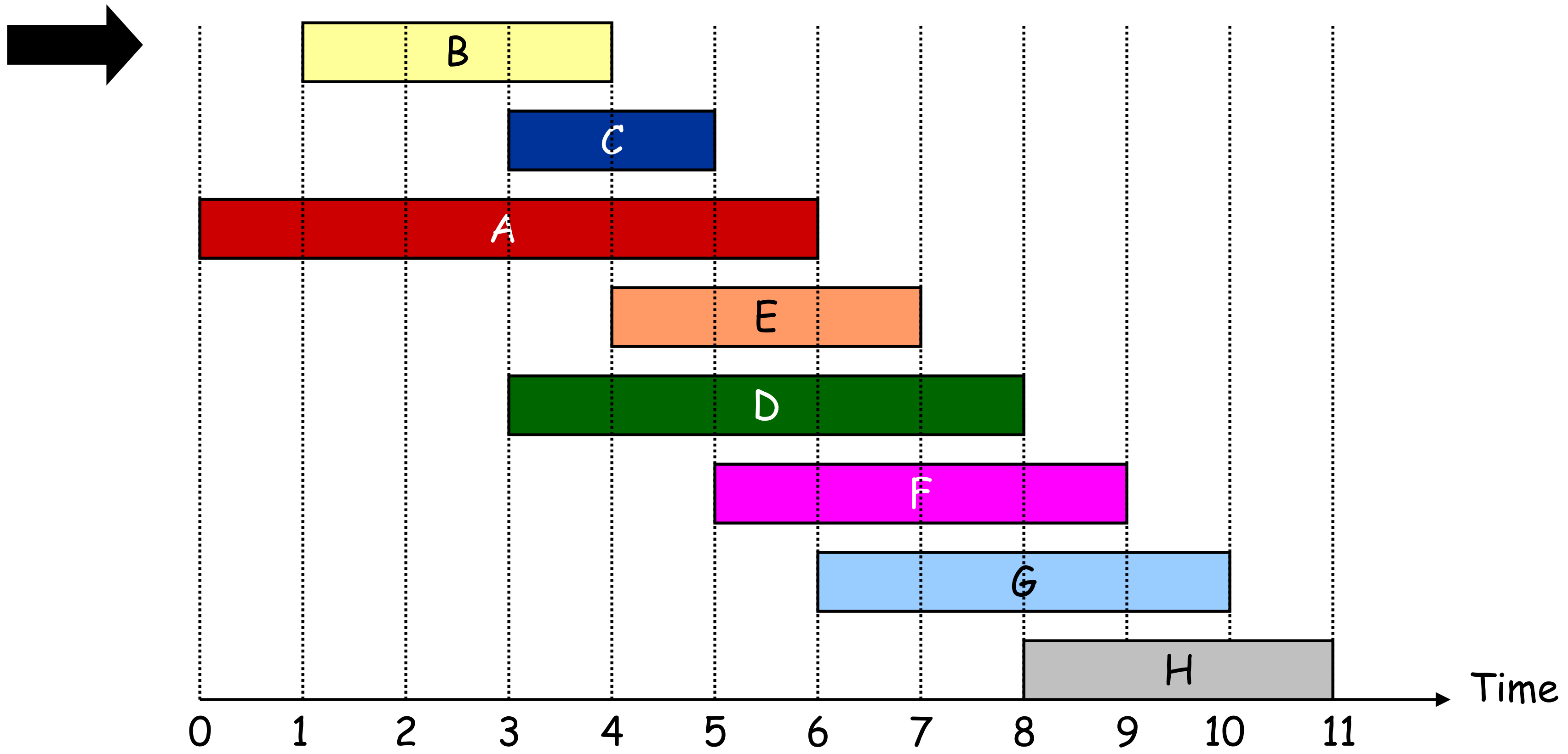
```
Sort jobs by finish times so that f₁ ≤ f₂ ≤ ... ≤ fₙ.

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

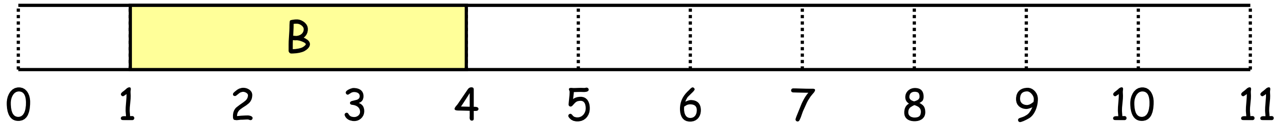
Interval Scheduling



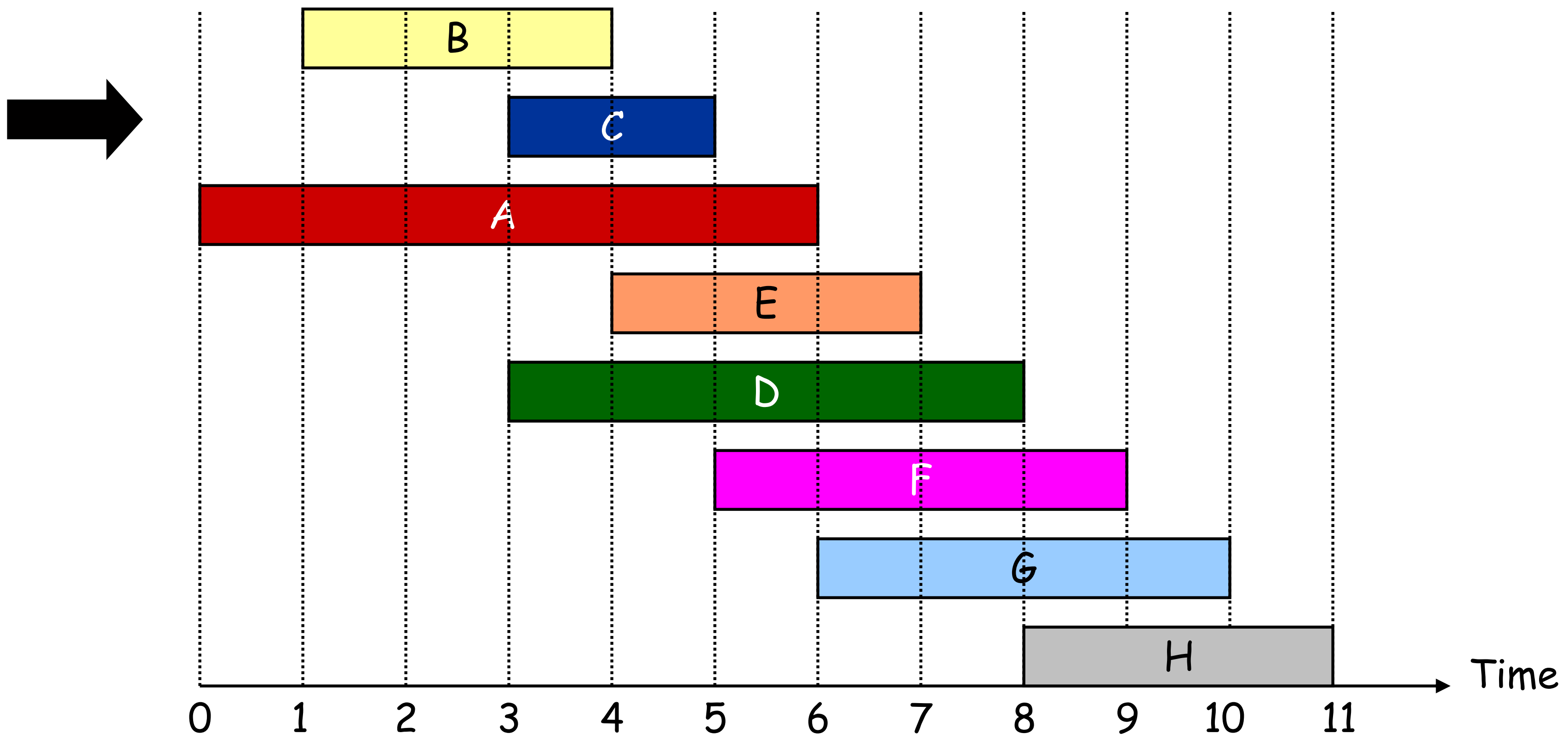


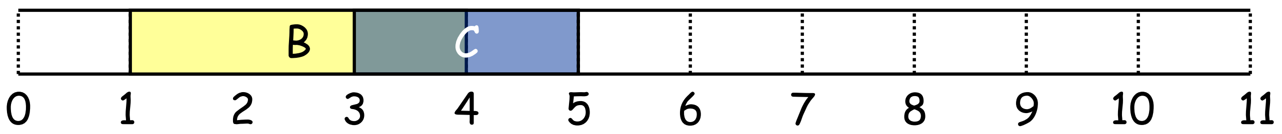
Interval Scheduling



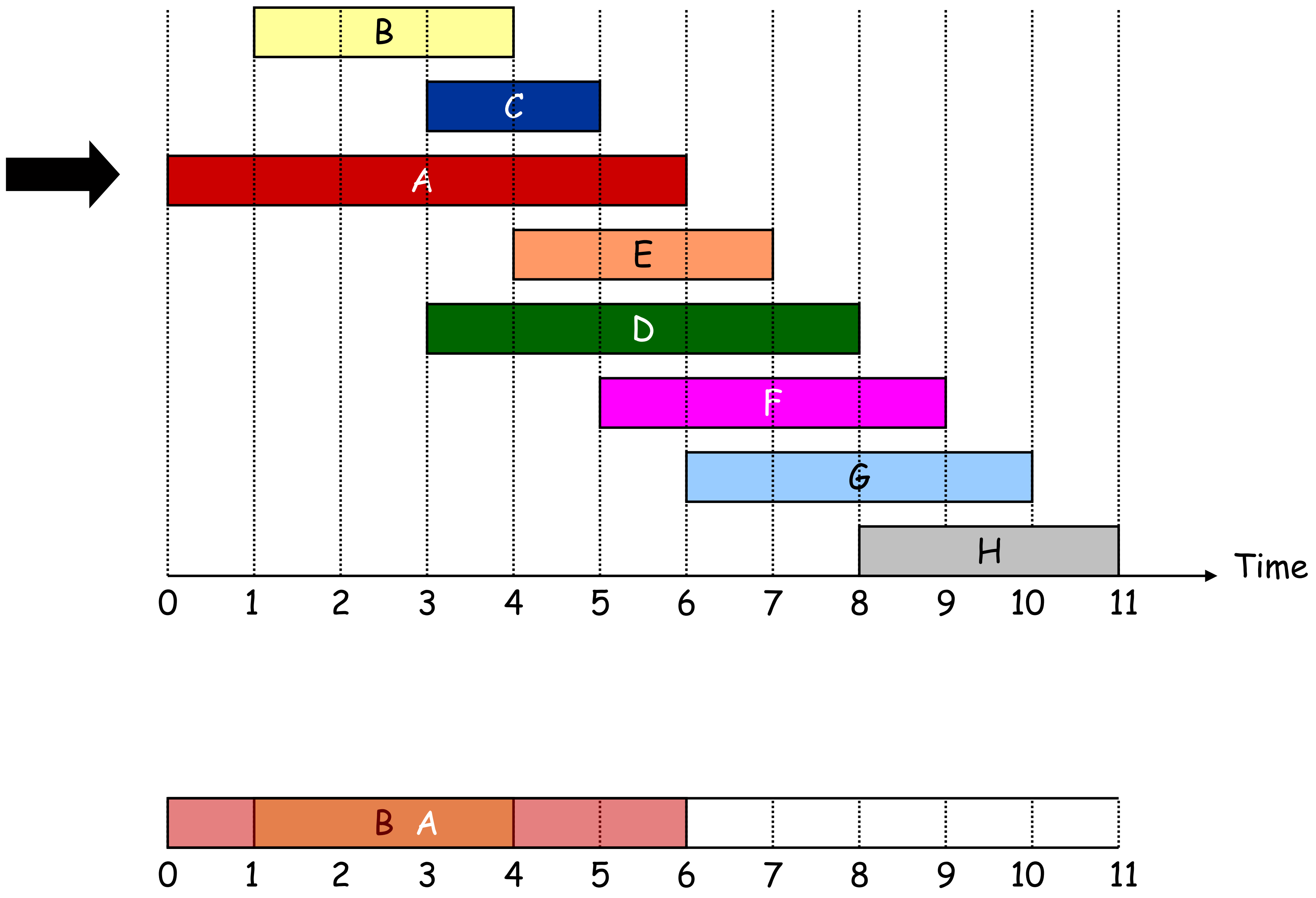


Interval Scheduling

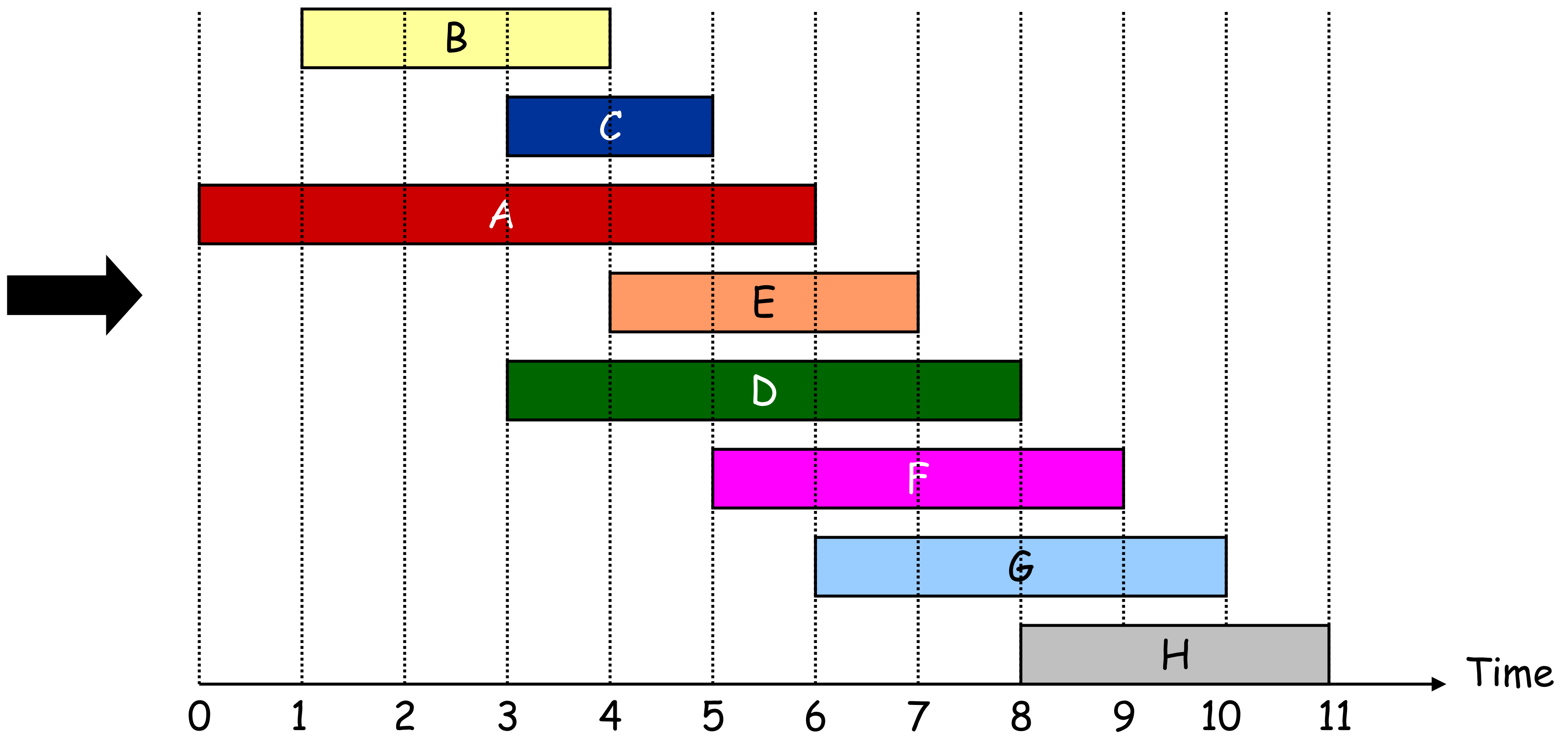


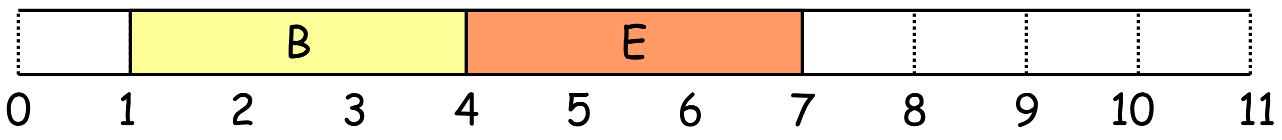


Interval Scheduling

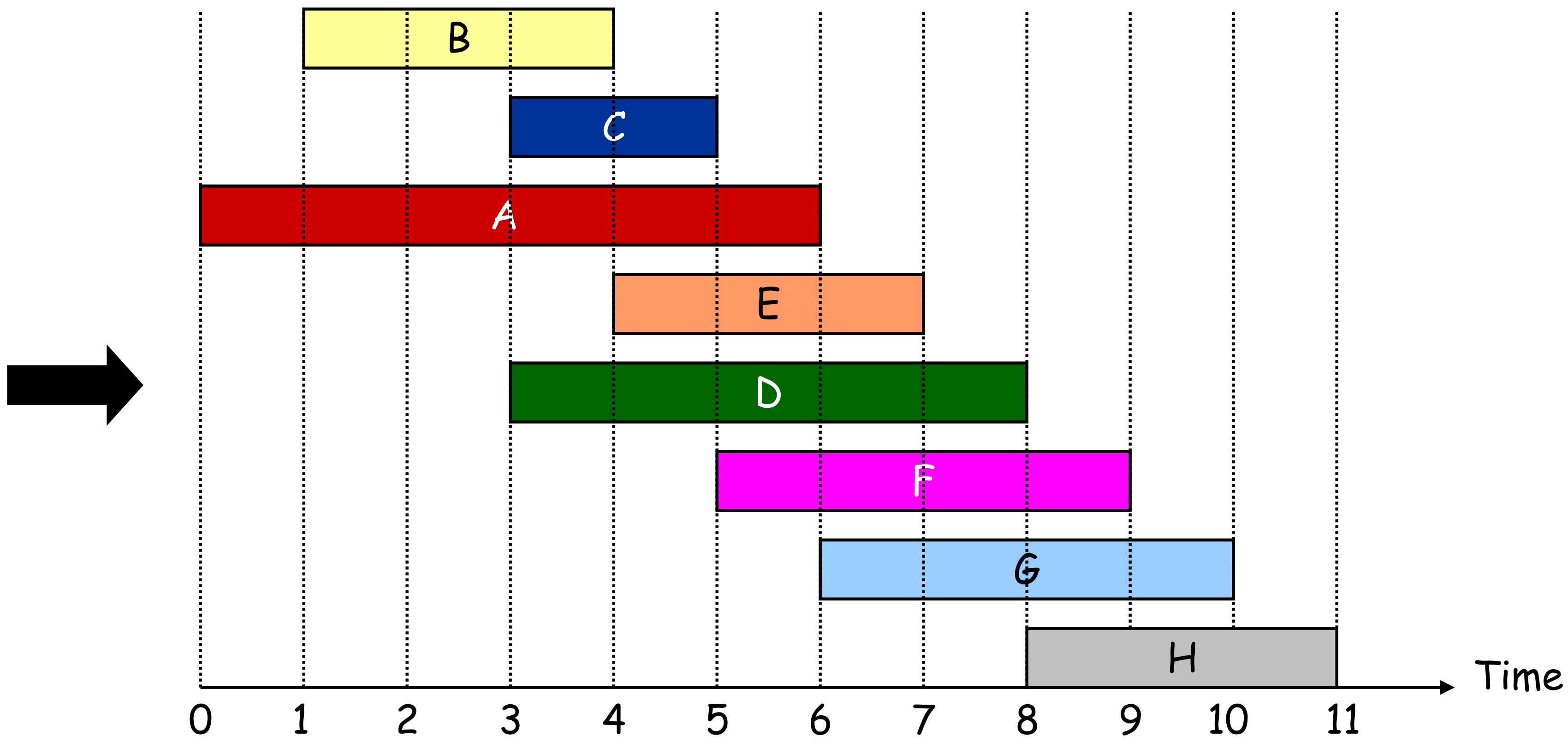


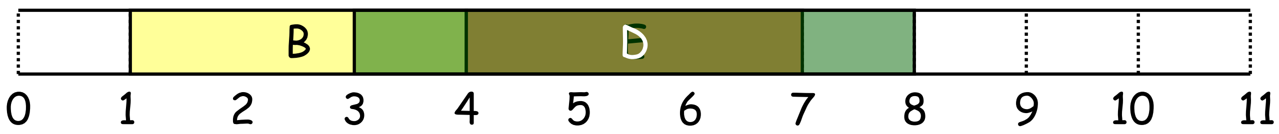
Interval Scheduling



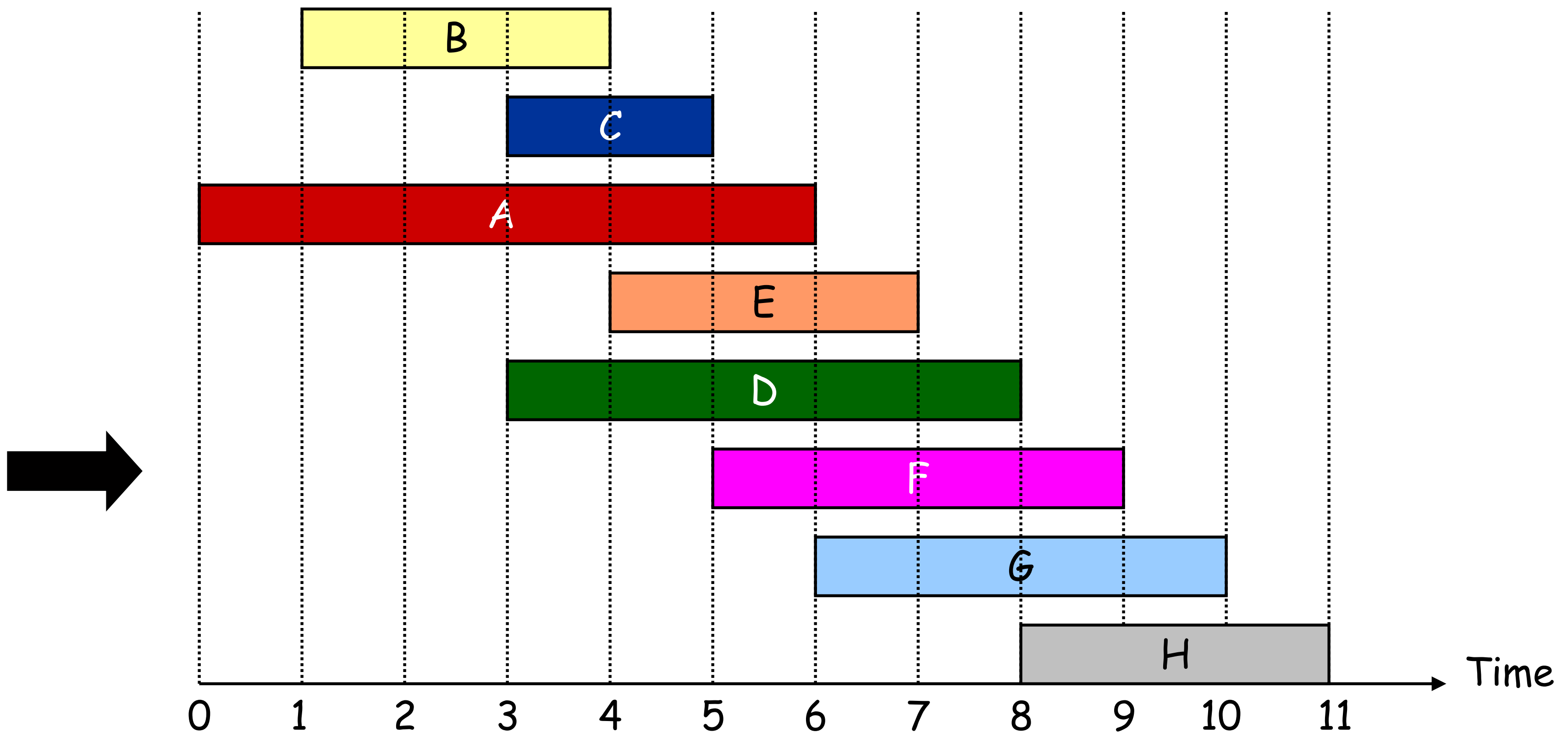


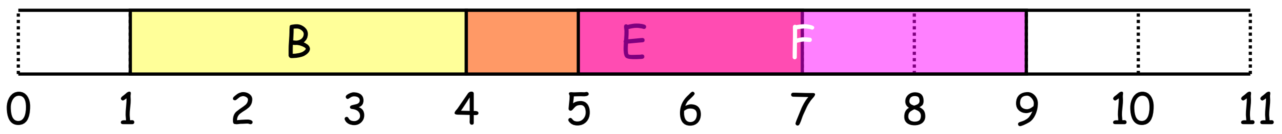
Interval Scheduling



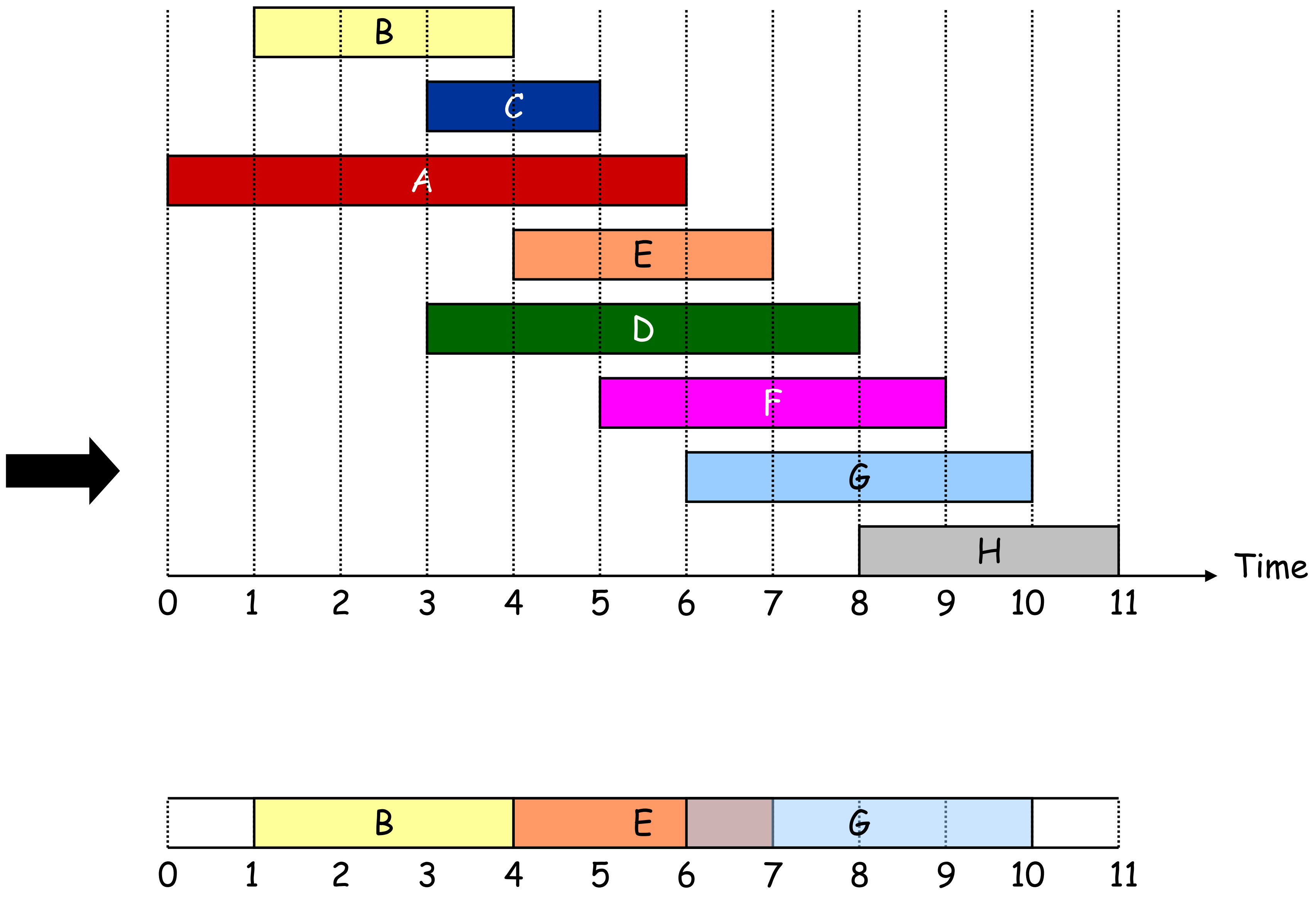


Interval Scheduling

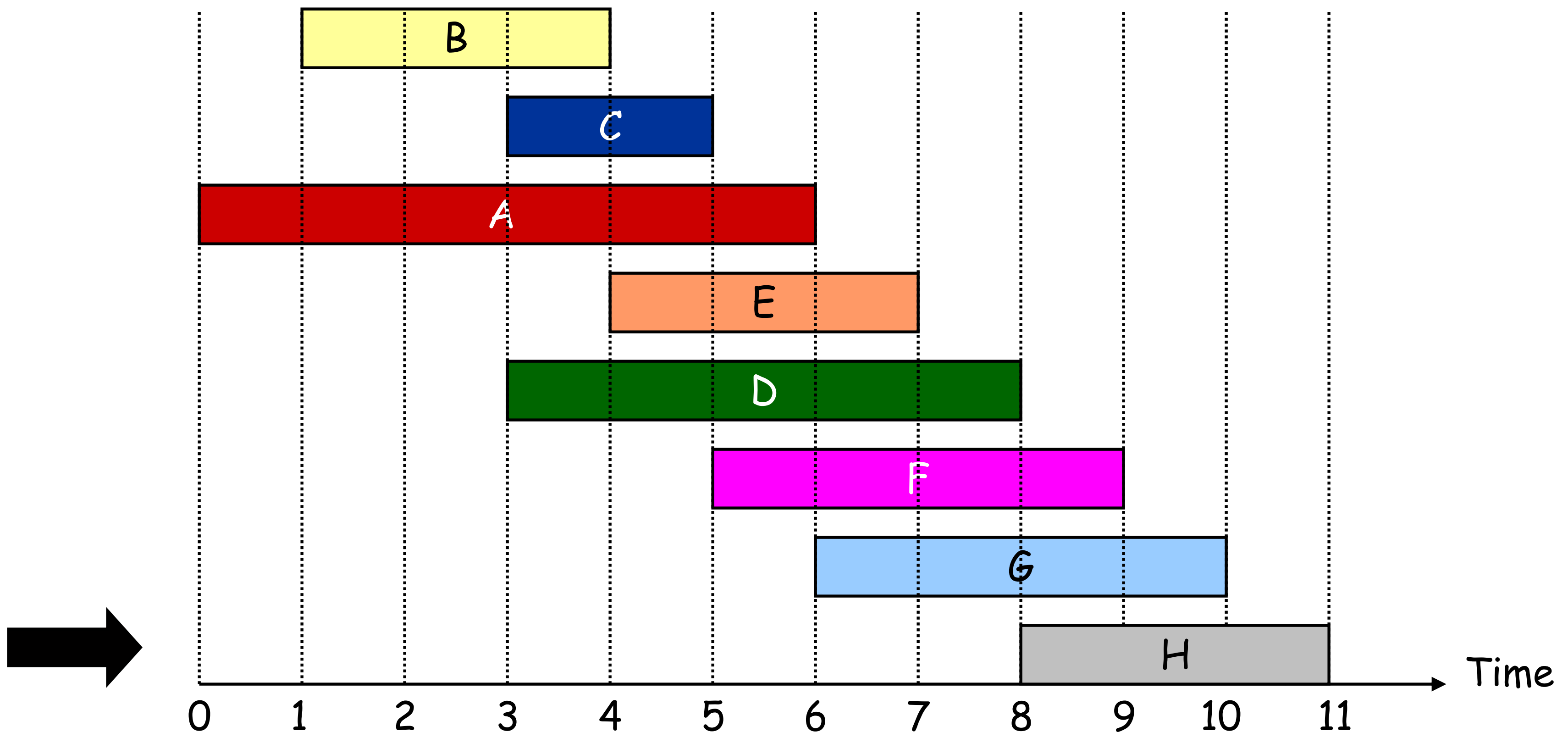


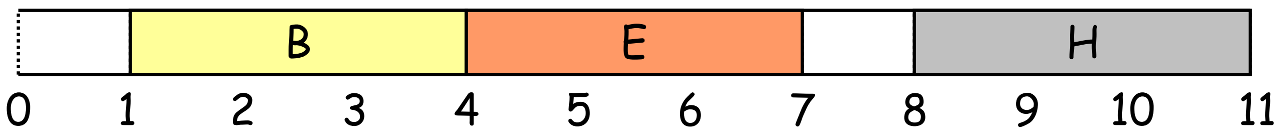


Interval Scheduling



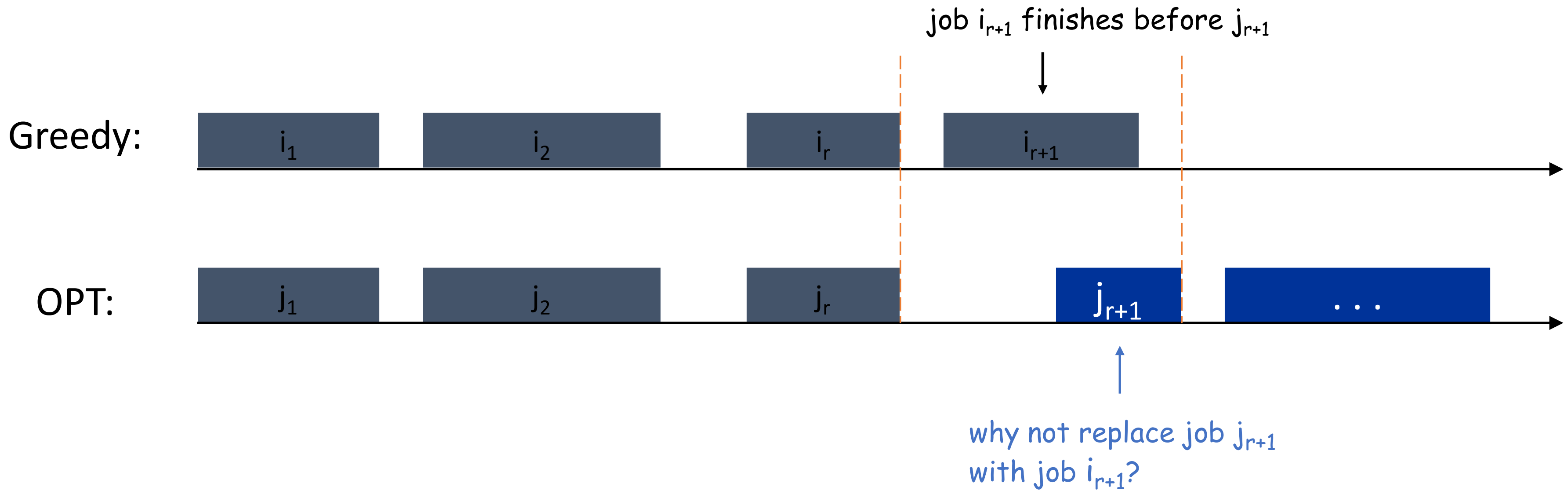
Interval Scheduling





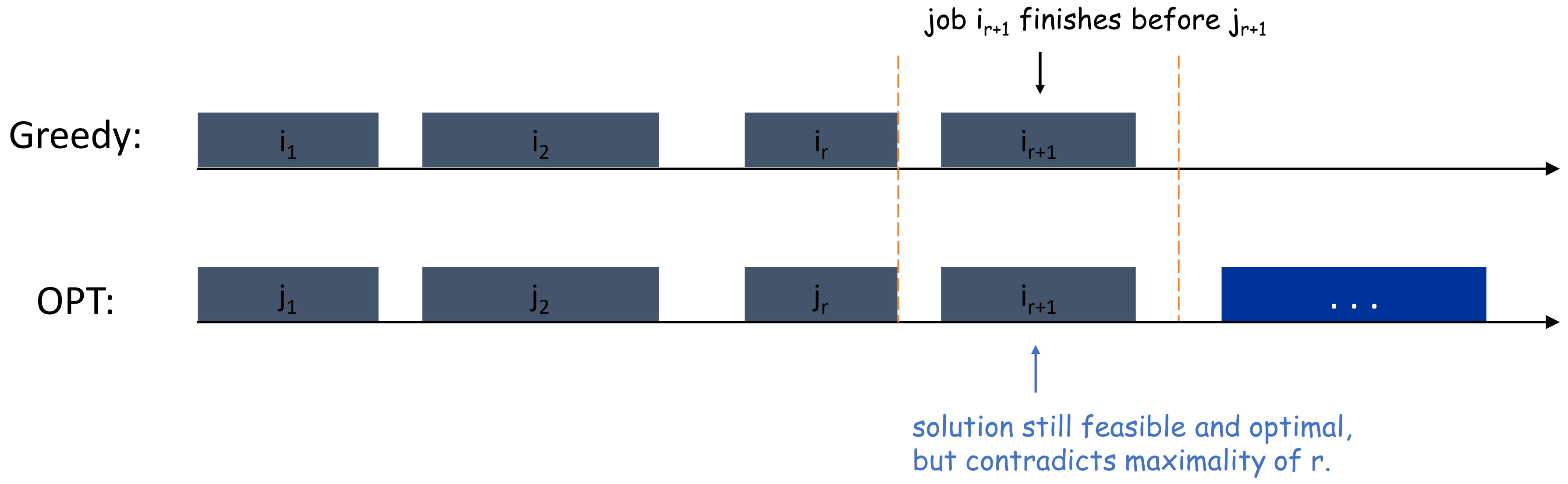
Interval Scheduling: Proof

- Theorem. Greedy algorithm is optimal.
- Pf. (by contradiction) greedy-stays-ahead approach
  - 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 the 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: Proof

- Theorem. Greedy 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 the optimal solution with $i_1 = j_1, i_2 = j_2, \ldots, i_r = j_r$ for the largest possible value of r.



Implementation

- Finding the next earliest finishing time of remaining
  - $O(n^2)$.
- Sorting
  - Sort all the requests by finishing time — O(n log n)
  - Iterate through the sorted array taking the next legal request — O(n)
  - O(nlogn)
- Using min-heap based priority queue
  - Build the heap using finish time keys — O(n)
  - While the heap has elements do n times
    - FindMin — O(1)
    - If the request doesn't conflict with the most recent scheduled request add to the final list of requests — O(1)
    - Delete root — O(log n)
- O(nlogn)

Summary

- Scheduling problems are often amenable to greedy approach
- But there many be many greedy choices and it is important to select the right one
- Main Takeaway: Greedy-stays-ahead is a useful proof approach