Algorithm Design and Analysis (H)

CS216

Instructor: Shan CHEN (陈杉)

chens3@sustech.edu.cn

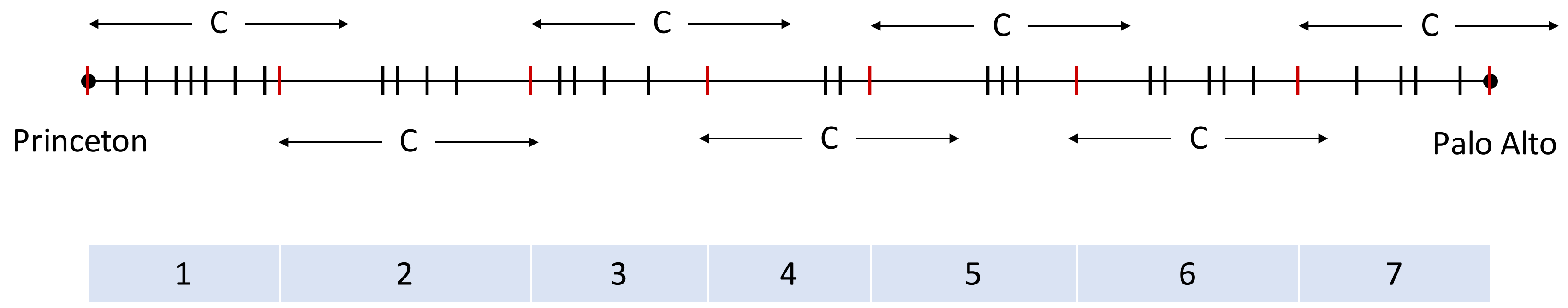
(slides edited from Prof. Shiqi Yu)

Greedy Algorithms

Example A: Selecting Breakpoints

Selecting Breakpoints

- **Selecting breakpoints.**
  - Road trip from Princeton to Palo Alto along fixed route.
  - Refueling stations at certain points $b_1, b_2, \ldots, b_n$ along the way.
  - Fuel capacity = C.
  - Goal: makes as few refueling stops as possible
- **Greedy algorithm.** Go as far as you can before refueling.



Selecting Breakpoints: Greedy Algorithm

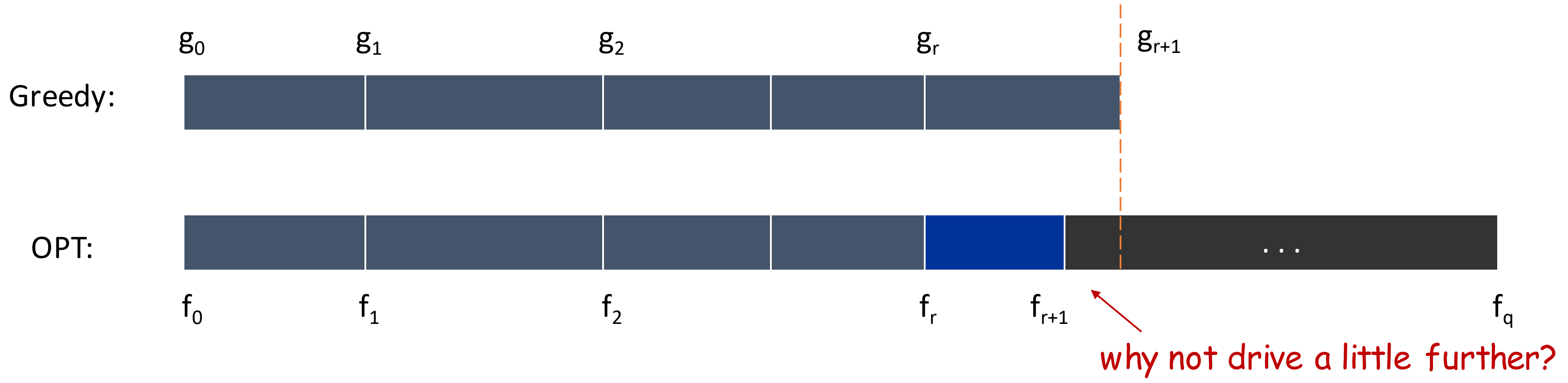
- **Truck driver's algorithm.**

```
Sort breakpoints so that: 0 = b0 < b1 < b2 < ... < bn = L
S ← {0}        ← selected breakpoints
x ← 0          ← current location
while (x ≠ bn)
   let p be largest integer such that bp ≤ x + C
   if (bp = x)
      return "no solution"
   x ← bp
   S ← S ∪ {p}
return S
```

- **Time complexity.** O(n log n)
  - ➢ Use binary search to find each breakpoint p.

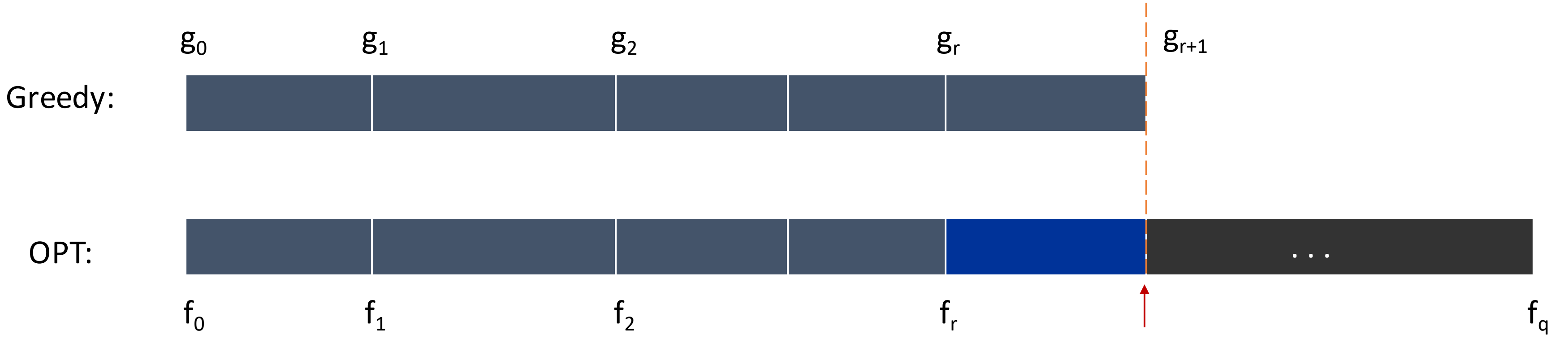
Selecting Breakpoints: Correctness

- **Theorem.** Greedy algorithm is optimal.
- Pf. (by contradiction)
  - Assume greedy is not optimal, and let's see what happens.
  - Let $0 = g_0 < g_1 < \ldots < g_p = L$ denote the set of breakpoints chosen by greedy.
  - Let $0 = f_0 < f_1 < \ldots < f_q = L$ denote the set of breakpoints in an optimal solution with $f_0 = g_0$, $f_1 = g_1$, ... , $f_r = g_r$ for largest possible value of r.
  - Note: $g_{r+1} > f_{r+1}$ by greedy choice of algorithm.



Selecting Breakpoints: Correctness

- **Theorem.** Greedy algorithm is optimal.
- Pf. (by contradiction)
  - Assume greedy is not optimal, and let's see what happens.
  - Let $0 = g_0 < g_1 < \ldots < g_p = L$ denote the set of breakpoints chosen by greedy.
  - Let $0 = f_0 < f_1 < \ldots < f_q = L$ denote the set of breakpoints in an optimal solution with $f_0 = g_0$, $f_1 = g_1$, ... , $f_r = g_r$ for largest possible value of r.
  - Note: $g_{r+1} > f_{r+1}$ by greedy choice of algorithm.



optimal solution that has r + 1 breakpoints in common, contradiction!

Example B:
Coin Changing

Coin Changing

- **Coin changing.** Given currency denominations: 1, 5, 10, 25, 100, devise a method to pay amount to customer using fewest number of coins.

- Ex: 34¢.

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

- Ex: $2.89.

Coin Changing: Greedy Algorithm

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

```
Sort coin denominations: c1 < c2 < … < cn.
S ← φ         ← selected coins
while (x ≠ 0) {
   let k be largest integer such that ck ≤ x
   if (k = 0)
      return "no solution found"
   x ← x - ck
   S ← S ∪ {k}
}
return S
```

- **Q.** Is the above greedy algorithm optimal?

Coin Changing: Properties of Optimal Solutions

- **Property.** Number of pennies $P \leq 4$.
- Pf. Replace 5 pennies with 1 nickel.
- **Property.** Number of nickels $N \leq 1$.
- Pf. Replace 2 nickels with 1 dime.
- **Property.** Number of quarters $Q \leq 3$.
- Pf. Replace 4 quarters with 1 dollar.
- **Property.** Number of nickels N + number of dimes $D \leq 2$.
- Pf. Recall: $N \leq 1$
  - Replace 3 dimes and 0 nickels with 1 quarter and 1 nickel.
  - Replace 2 dimes and 1 nickel with 1 quarter.

Coin Changing: Analysis of Greedy Algorithm

- **Theorem.** Greedy is optimal for U.S. coinage: 1, 5, 10, 25, 100.
- Pf. (by induction on the 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 also take coin k, reducing x to $x - c_k$.
    - ✓ if not, it needs enough coins of type $c_1, \ldots, c_{k-1}$ to sum up to x
    - ✓ table below indicates no optimal solution can do this

k	c_k	All optimal solutions must satisfy	Max value of coins 1, 2, …, k-1 in any OPT
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

Coin Changing: Analysis of Greedy Algorithm

- **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.

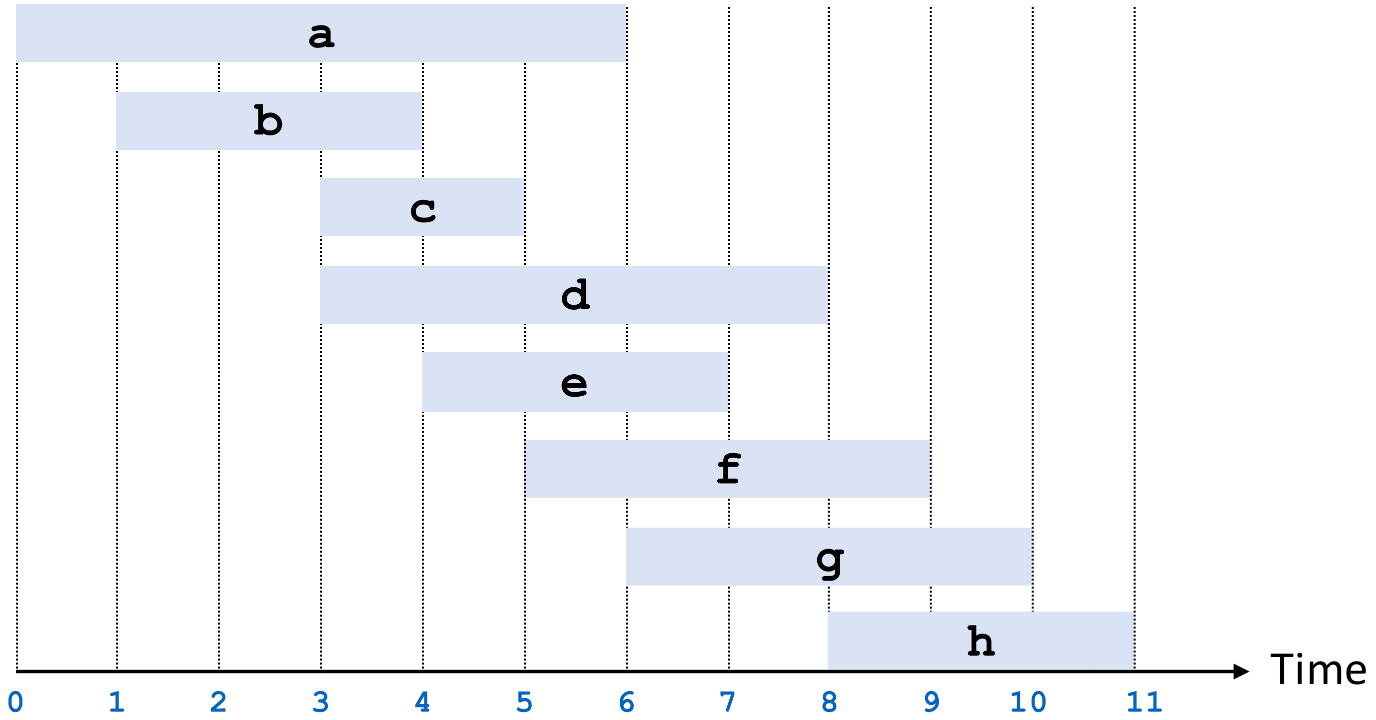
Greedy Algorithms

- Build up a solution in small steps.
- Choose a decision at each step myopically to optimize some underlying criterion.
- May not produce an optimal solution.
- But can yield locally optimal solutions that approximate a globally optimal solution in a reasonable amount of time.

1. 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.



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.

    Counterexample:

  - [Earliest finish time] Consider jobs in ascending order of f_j.
  - [Shortest interval] Consider jobs in ascending order of $f_j - s_j$.

    Counterexample:

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

    Counterexample:

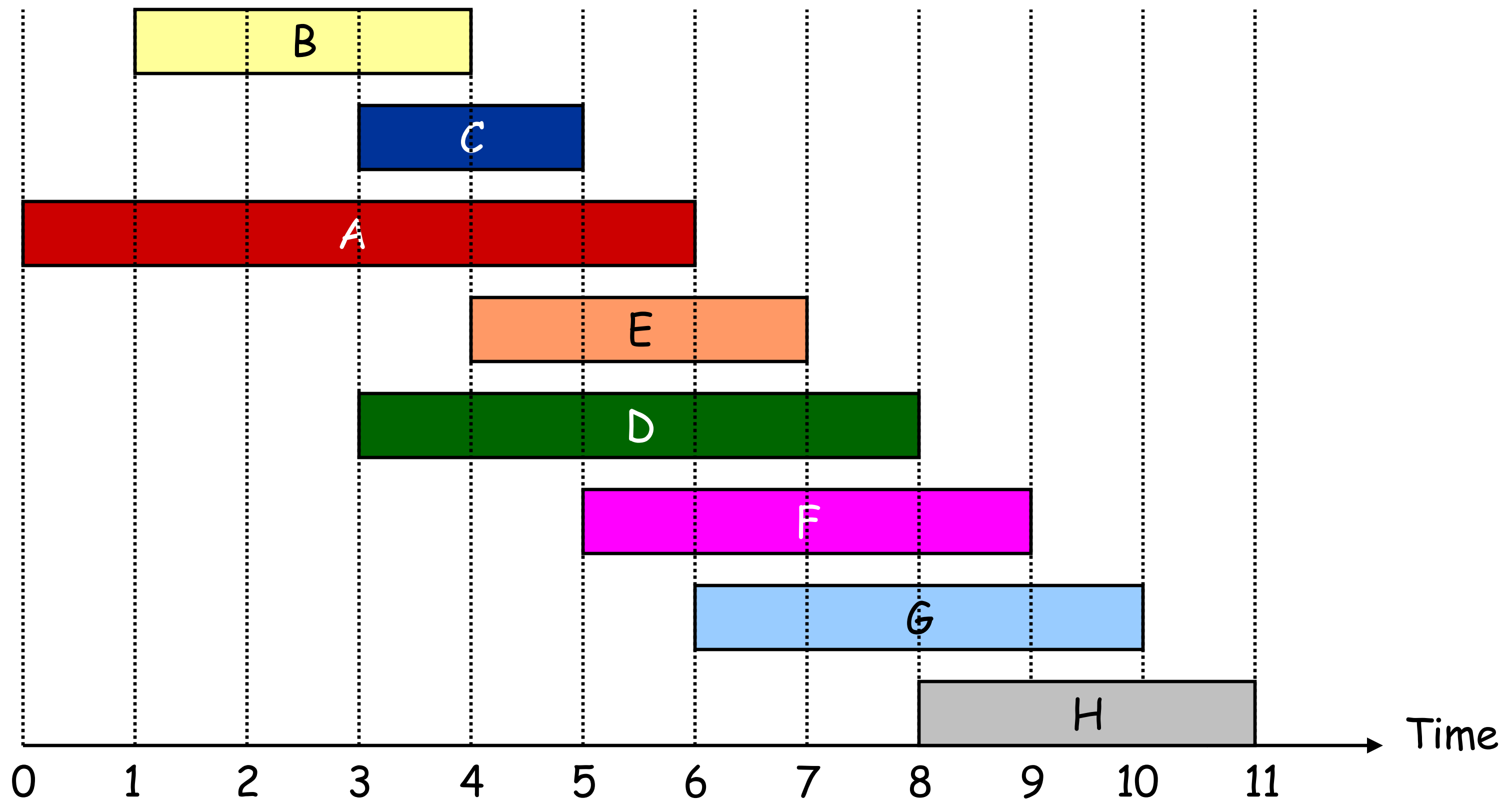
Interval Scheduling: Greedy Algorithm

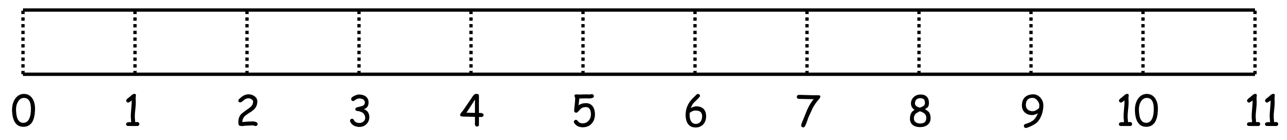
- **Greedy algorithm**. Consider jobs in increasing order of finish time. Take each job provided it's compatible with the ones already taken.

```
Sort jobs by finish times so that f₁ ≤ f₂ ≤ ... ≤ fₙ.
A ← ∅     ← selected jobs
for j = 1 to n {
   if (job j compatible with A)
      A ← A ∪ {j}
}
return A
```

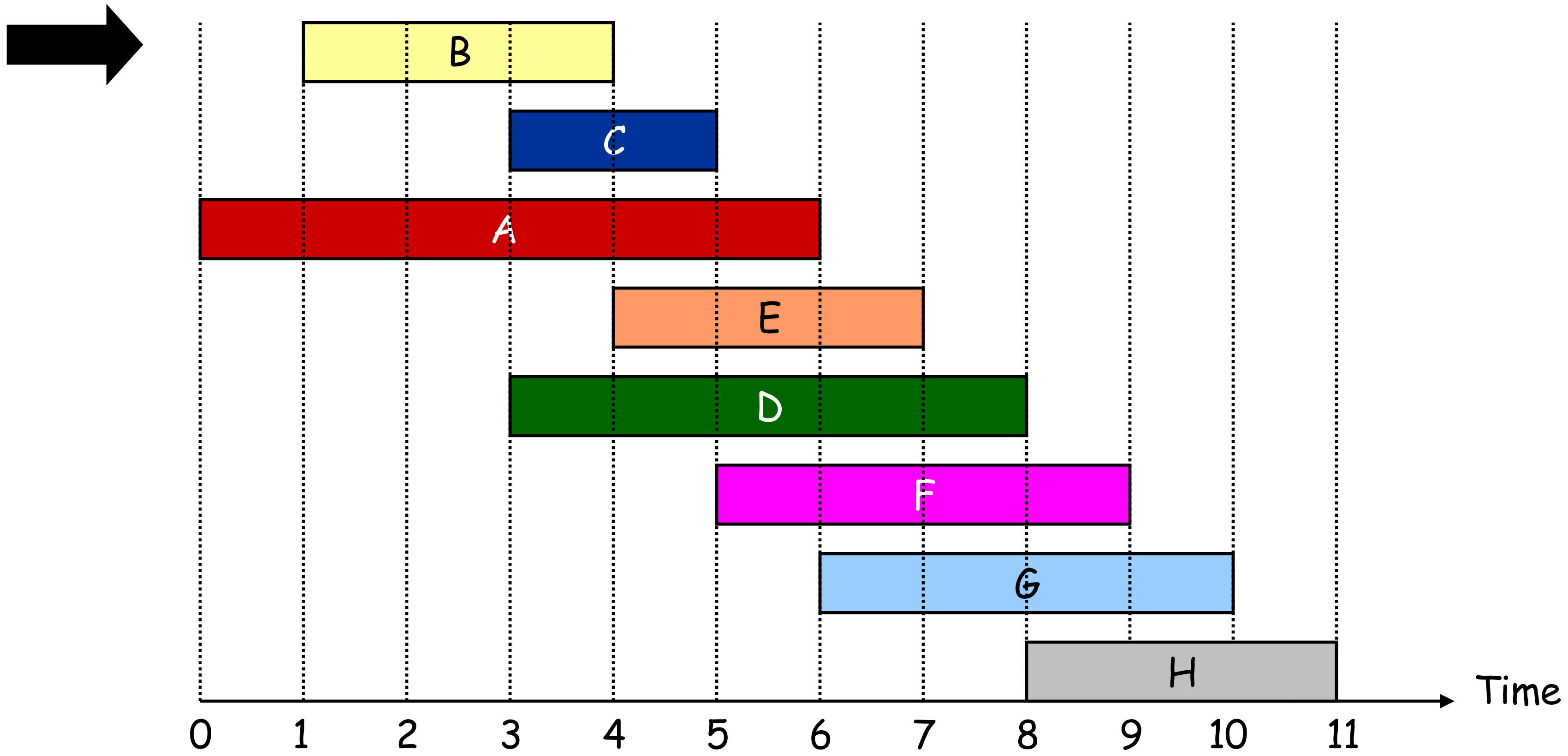
- **Time complexity.** O(n log n).
  - Remember job j* that was added last to A.
  - Job j is compatible with A if $s_j \geq f_{j*}$.

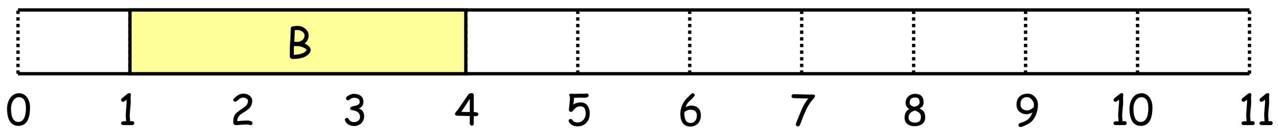
Interval Scheduling



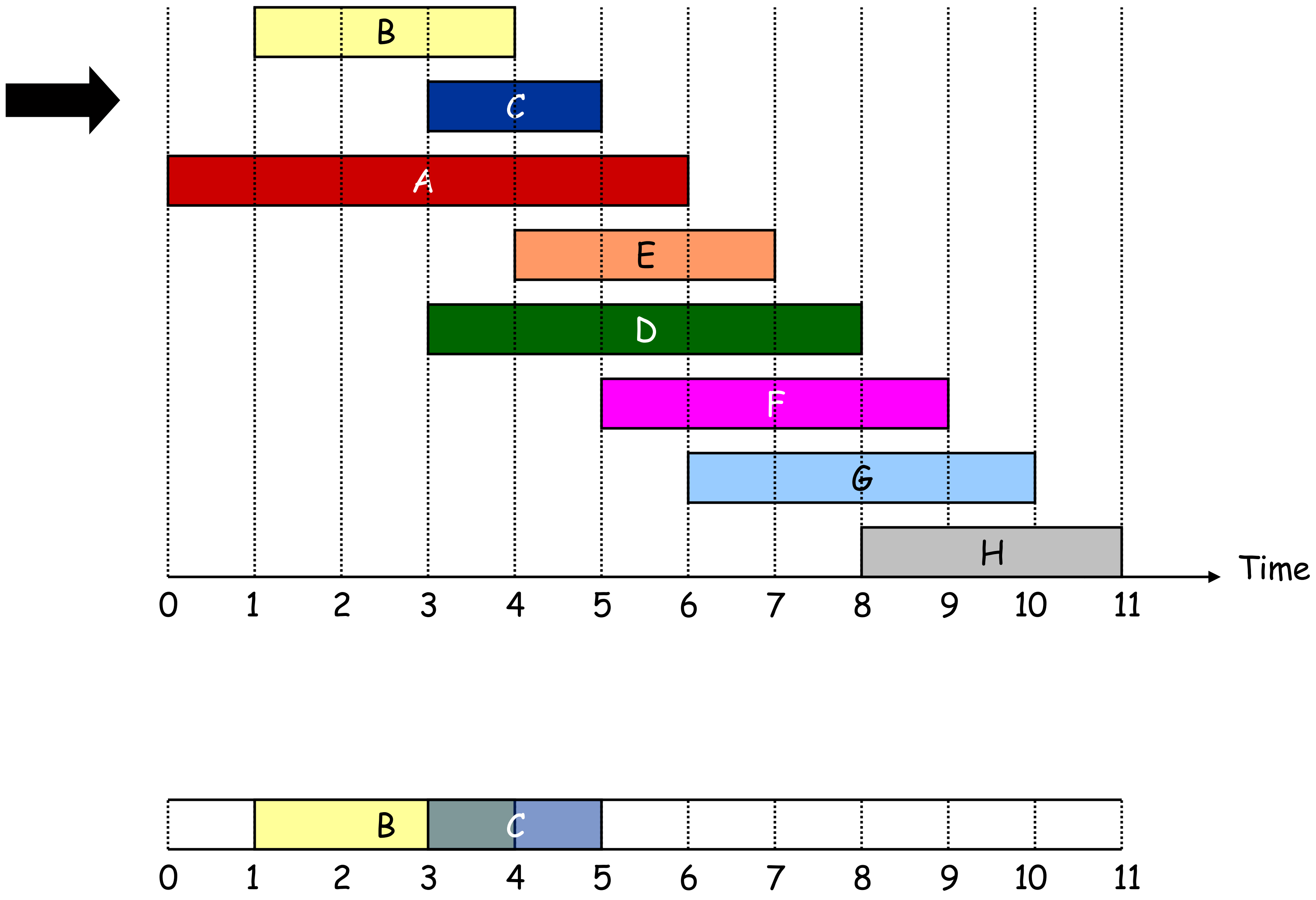


Interval Scheduling

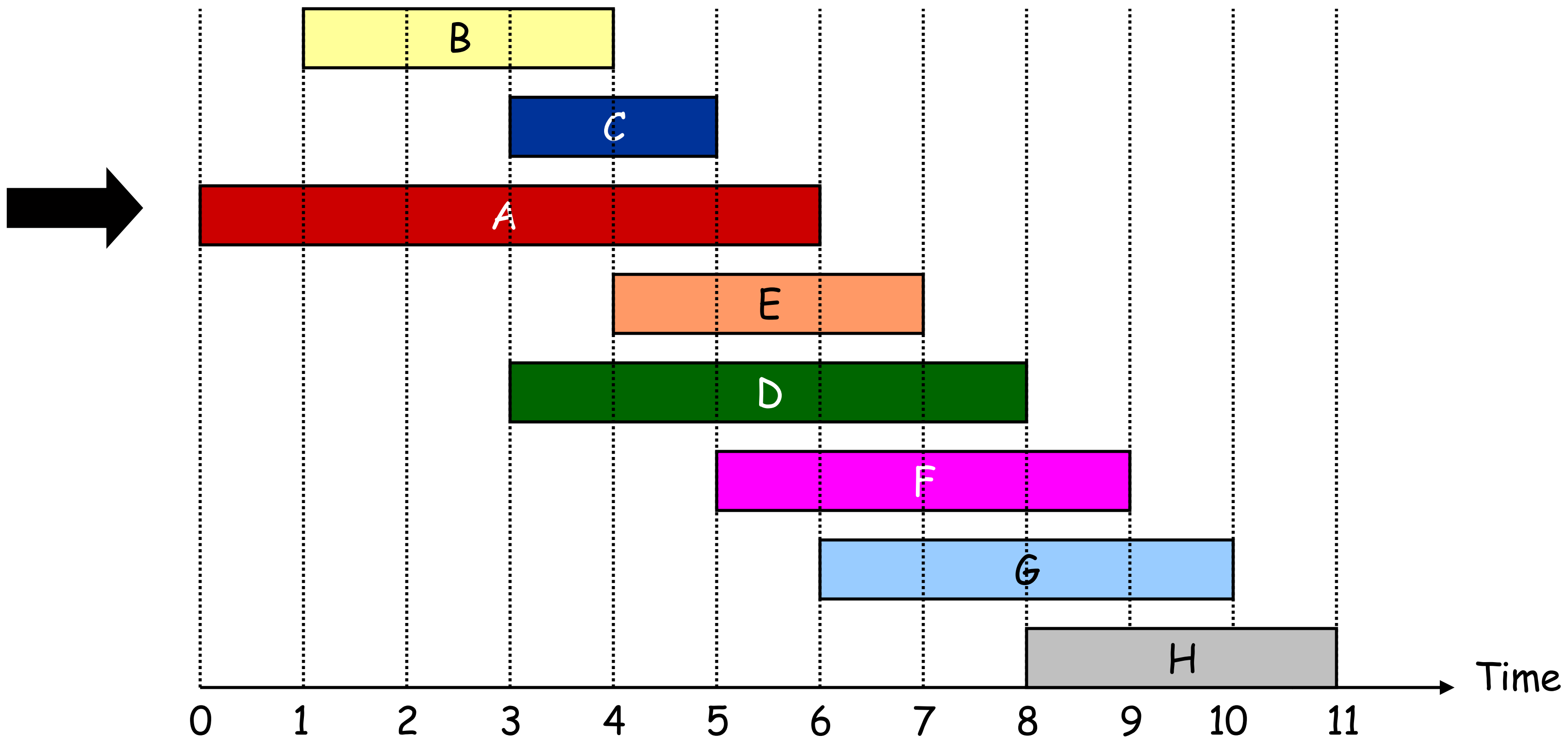


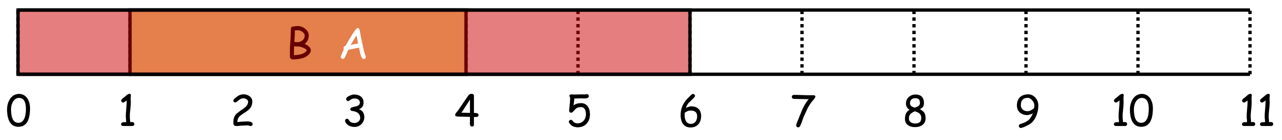


Interval Scheduling

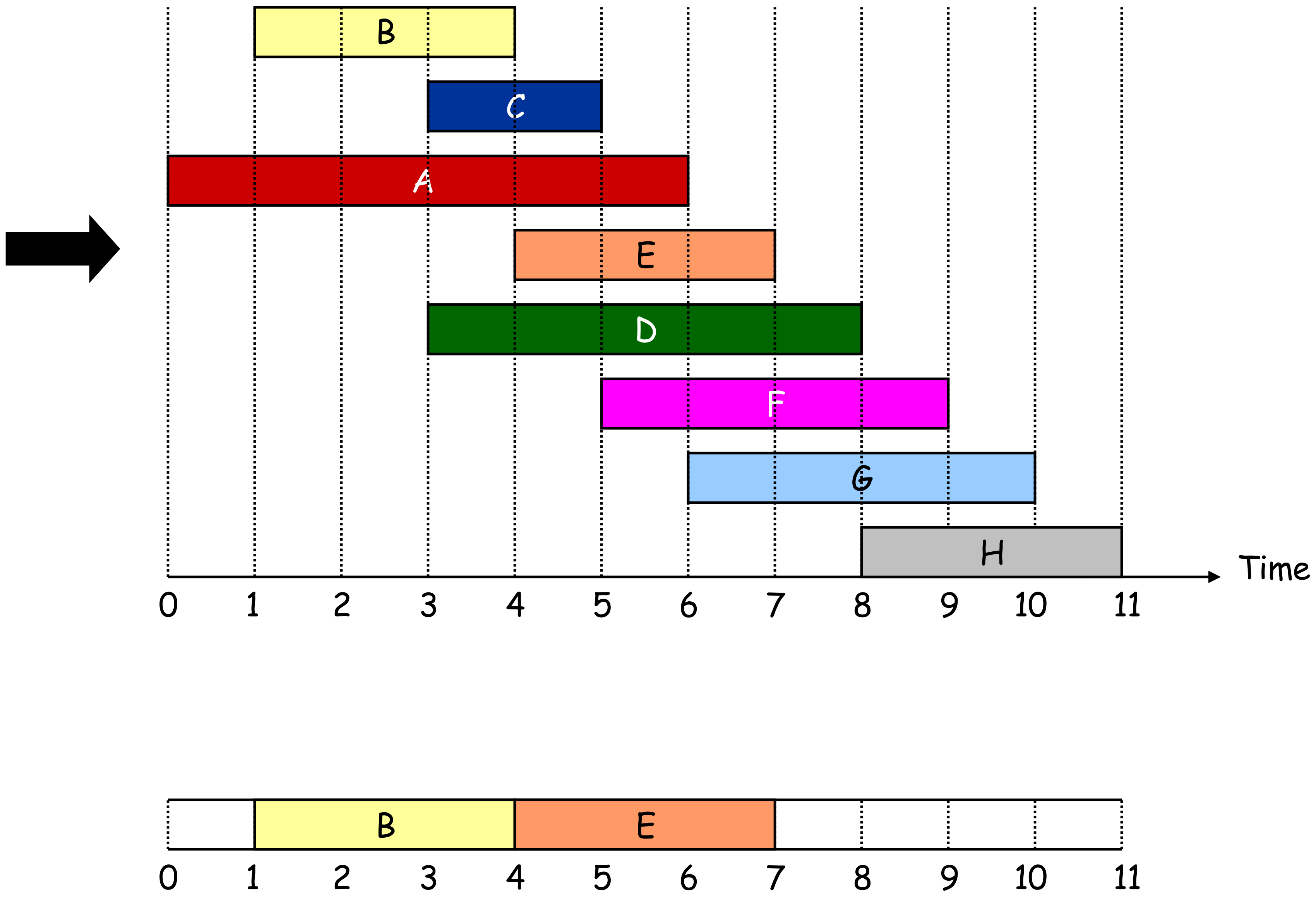


Interval Scheduling

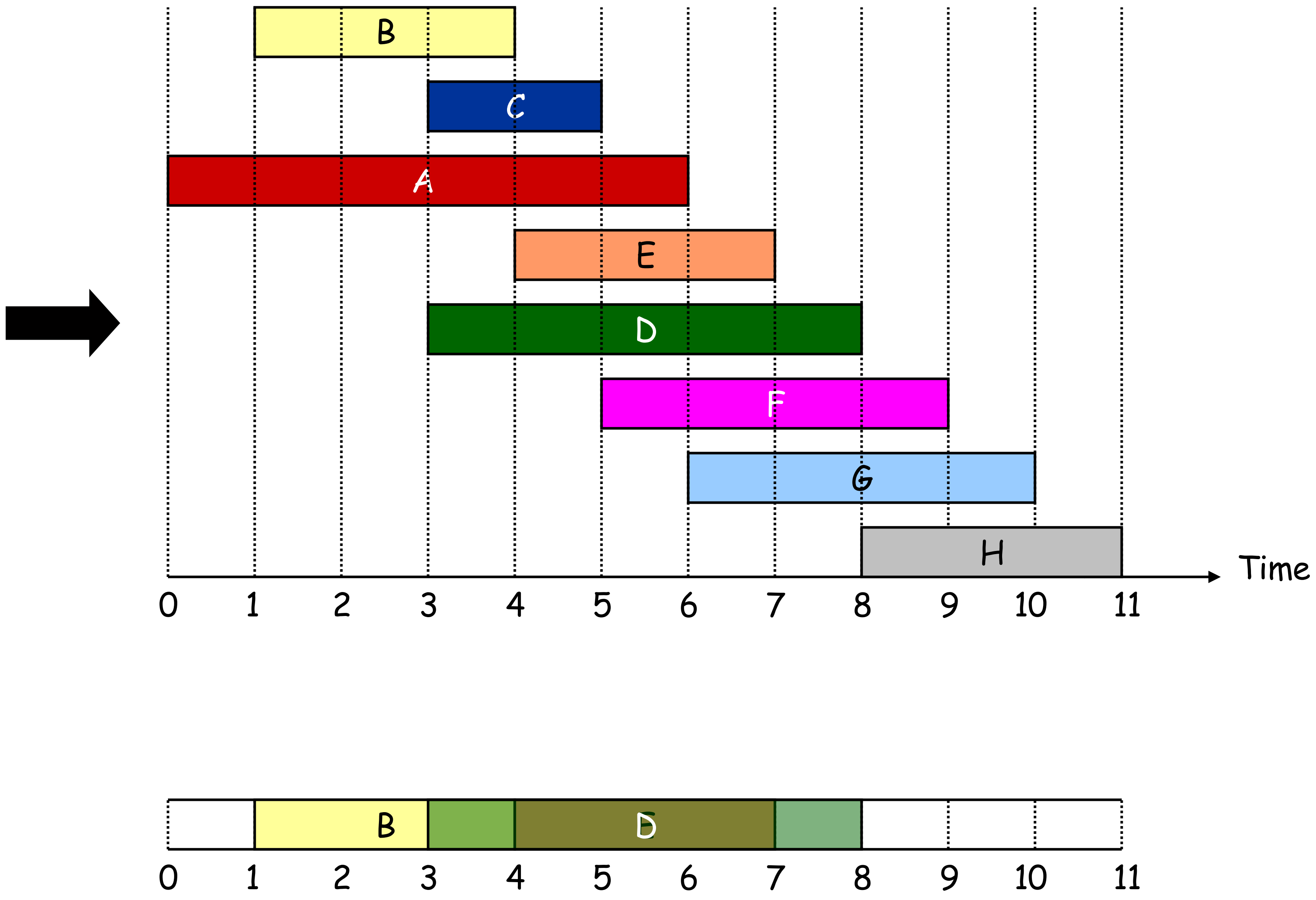




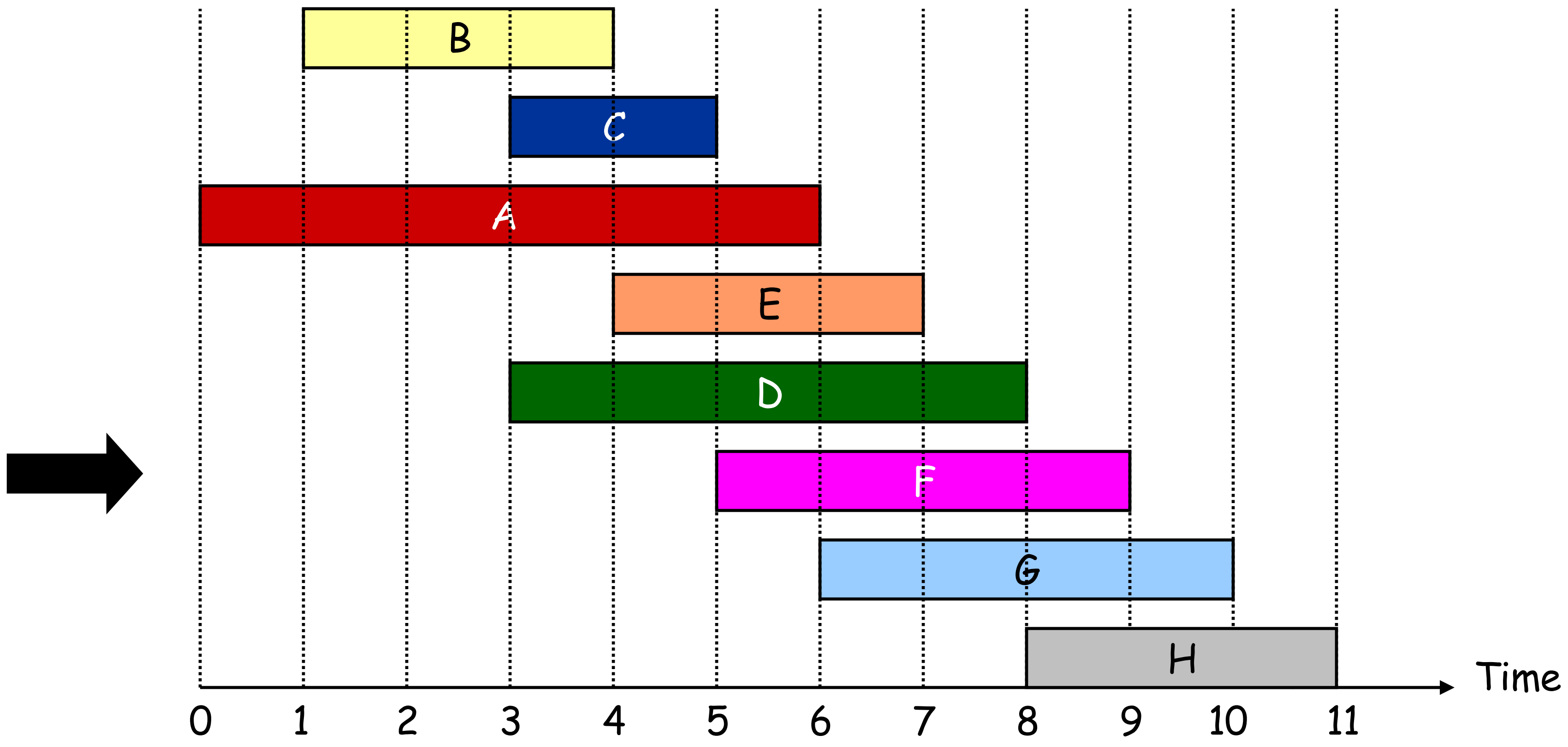
Interval Scheduling

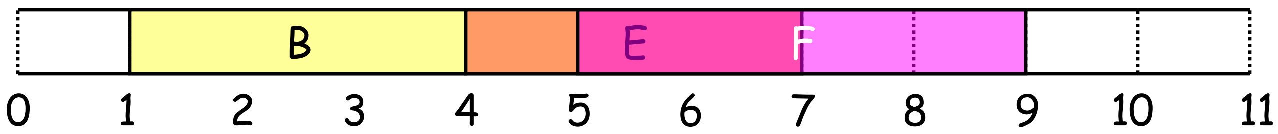


Interval Scheduling

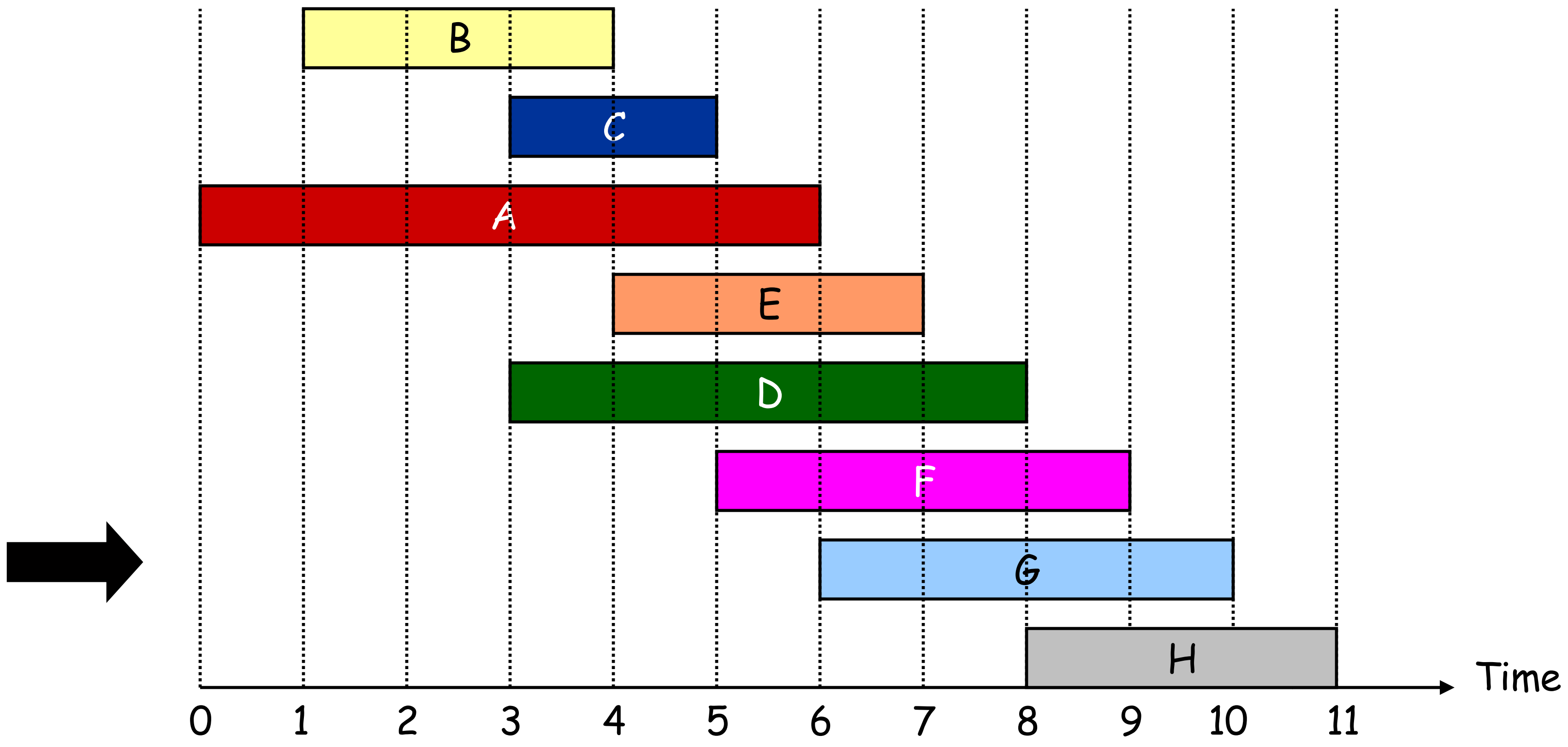


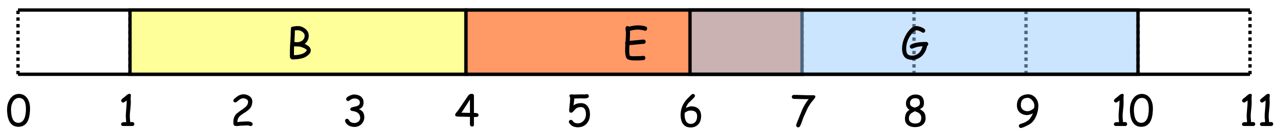
Interval Scheduling



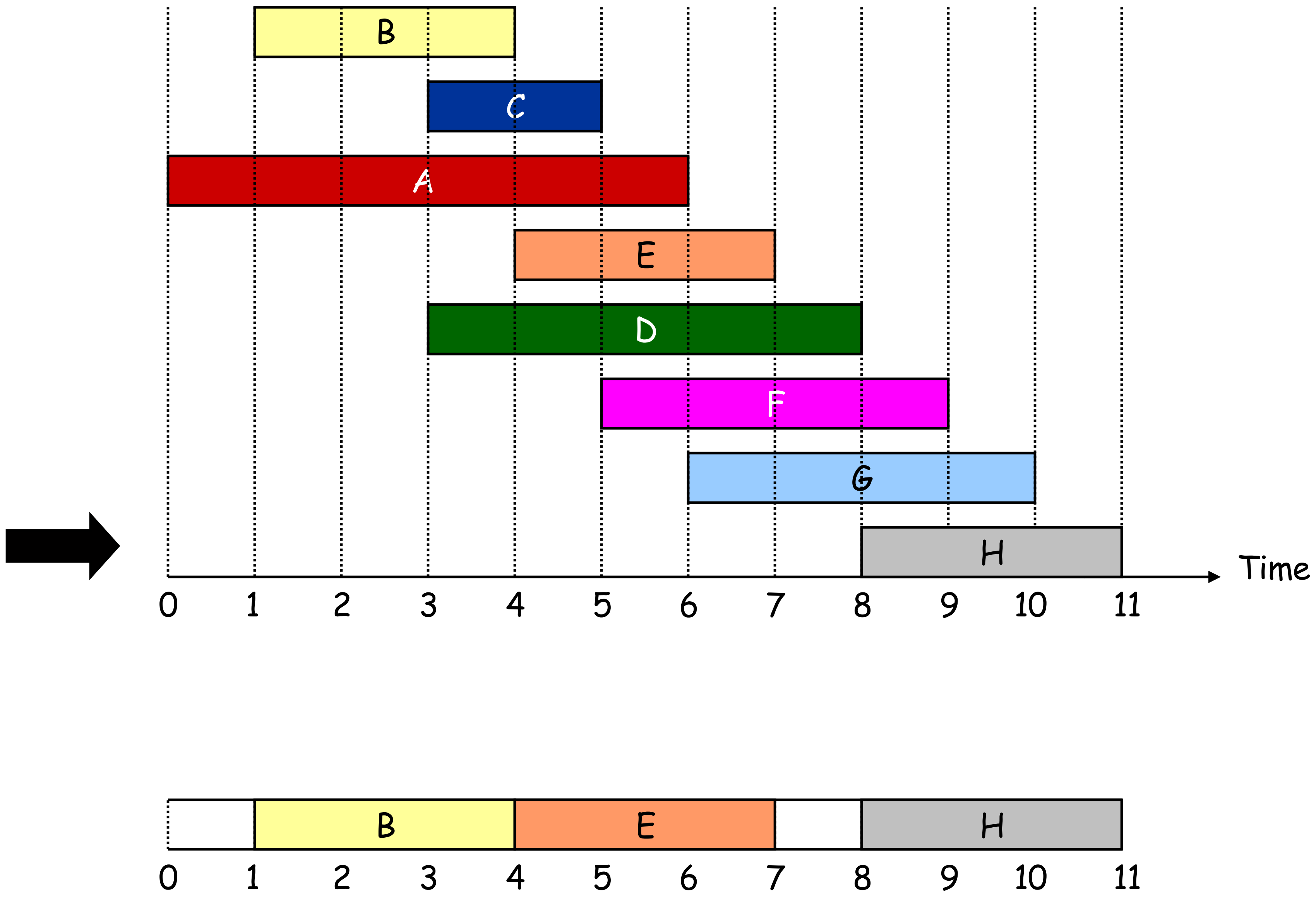


Interval Scheduling



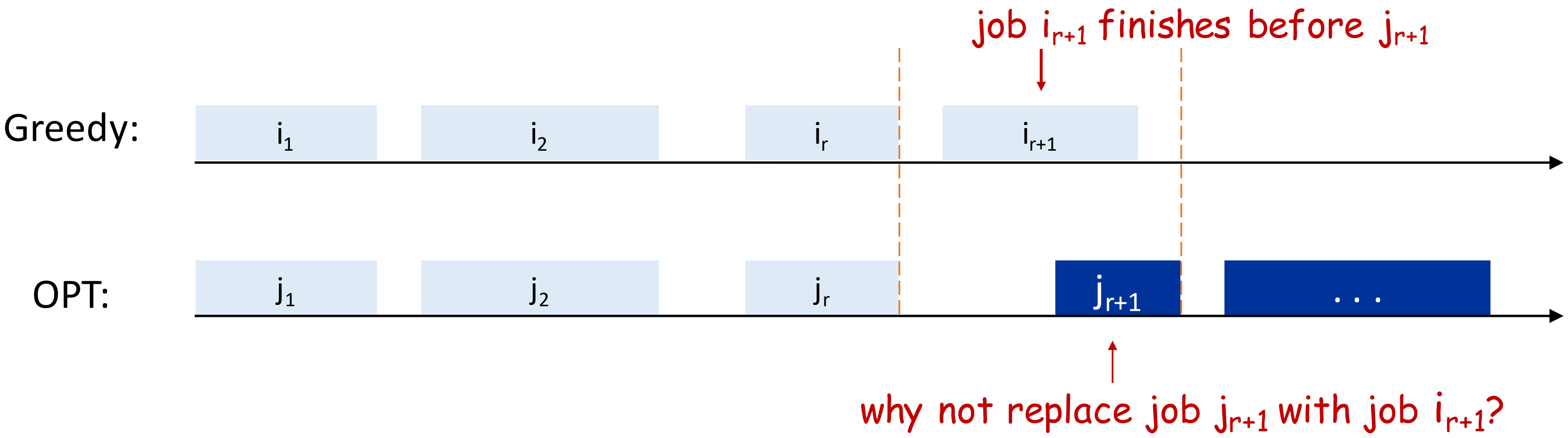


Interval Scheduling



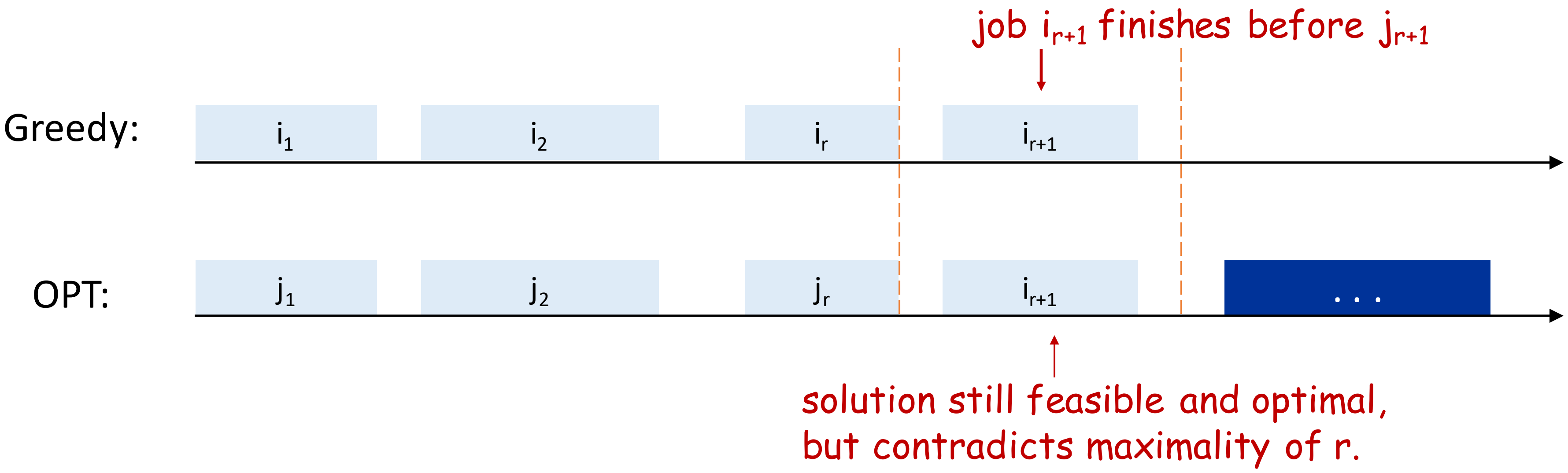
Interval Scheduling: Analysis

- **Theorem.** Greedy algorithm is optimal.
- Pf. (by contradiction)
  - Assume greedy is not optimal, and let's see what happens.
  - Let $\{i_1, i_2, \ldots i_n\}$ denote the set of jobs selected by greedy.
  - Let $\{j_1, j_2, \ldots j_m\}$ denote the set of jobs in the optimal solution with the largest possible value of r such that $i_1 = j_1$, $i_2 = j_2$, ..., $i_r = j_r$.



Interval Scheduling: Analysis

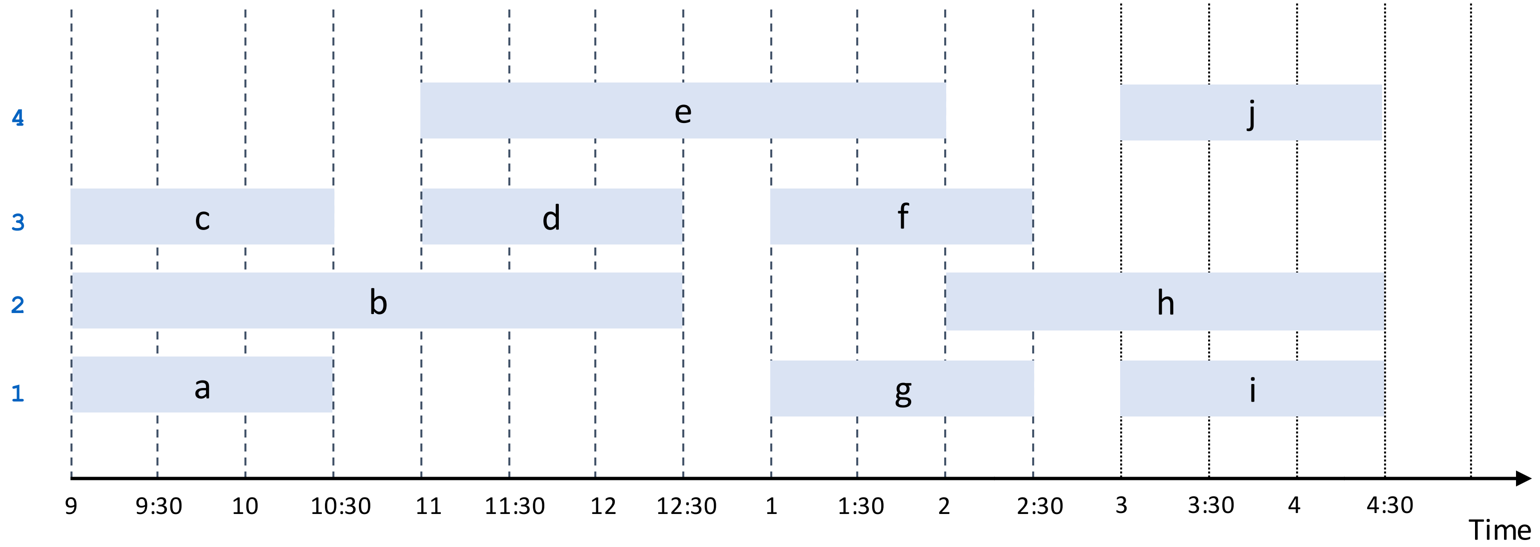
- **Theorem.** Greedy algorithm is optimal.
- Pf. (by contradiction)
  - Assume greedy is not optimal, and let's see what happens.
  - Let $\{i_1, i_2, \ldots i_n\}$ denote the set of jobs selected by greedy.
  - Let $\{j_1, j_2, \ldots j_m\}$ denote the set of jobs in the optimal solution with the largest possible value of r such that $i_1 = j_1$, $i_2 = j_2$, ..., $i_r = j_r$.



2. Interval Partitioning

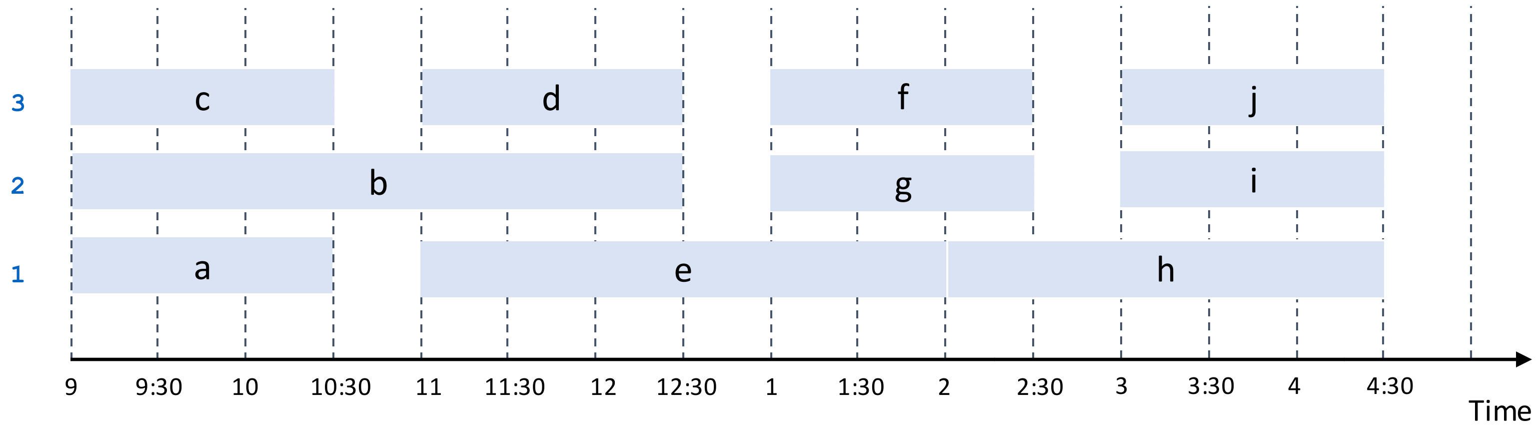
Interval Partitioning

- **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 classrooms.

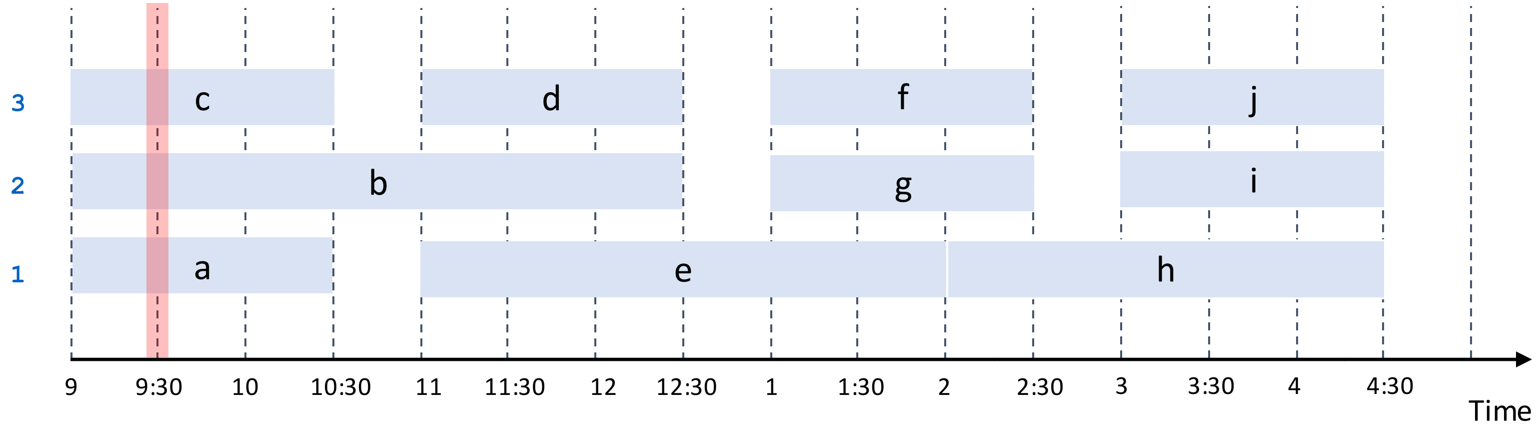


Interval Partitioning: Lower Bound on Optimal Solution

- **Def.** The depth of a set of open intervals is the maximum number of intervals that contain any given time point.
- **Key observation.** Number of classrooms needed $\geq$ depth.
- **Ex:** Depth of schedule below = 3 -> schedule below is optimal.

e.g., 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.

```
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
}
```

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

Interval Partitioning: Greedy Analysis

- **Observation.** Greedy algorithm never schedules two incompatible lectures in the same classroom.
- **Theorem.** Greedy algorithm is optimal.
- Pf. Let d = number of classrooms that the greedy algorithm allocates.
  - Classroom d is opened because the greedy algorithm needed to schedule a lecture, say j, that is incompatible with all d - 1 other classrooms.
  - The d - 1 last lectures in those d - 1 classrooms each finish after s_j.
  - Since the greedy algorithm sorted lectures by starting time, all those d - 1 incompatible lectures start no later than s_j.
  - Thus, we have d lectures overlapping at time $s_j + \varepsilon$ (i.e., right after s_j).
  - Key observation: all schedules must use ≥ d classrooms. ▪

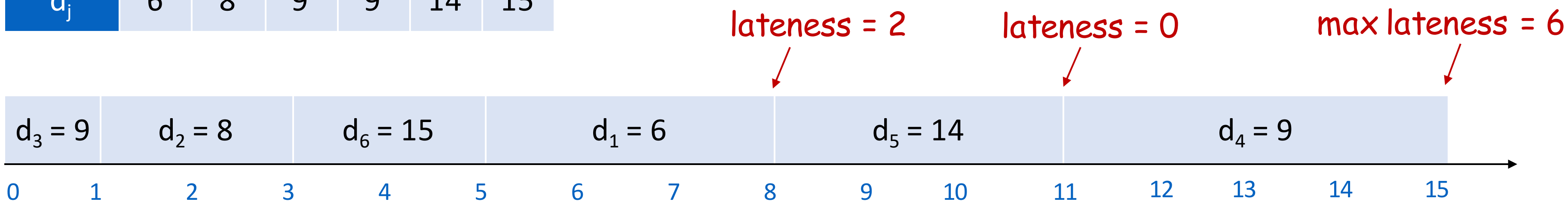
3. Scheduling to Minimize Lateness

Scheduling to Minimizing Lateness

- **Minimizing lateness problem.**
  - 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 $L = \max \ell_j$.

- **Ex:**

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15



Minimizing Lateness: Greedy Algorithms

- **Greedy template.** Consider jobs in some order.
  - [Shortest processing time first] Consider jobs in ascending order of processing time t_j.

	1	2
t_j	1	10
d_j	100	10

counterexample

  - [Earliest deadline first] Consider jobs in ascending order of deadline d_j.
  - [Smallest slack] Consider jobs in ascending order of slack $d_j - t_j$.

	1	2
t_j	1	10
d_j	2	10

counterexample

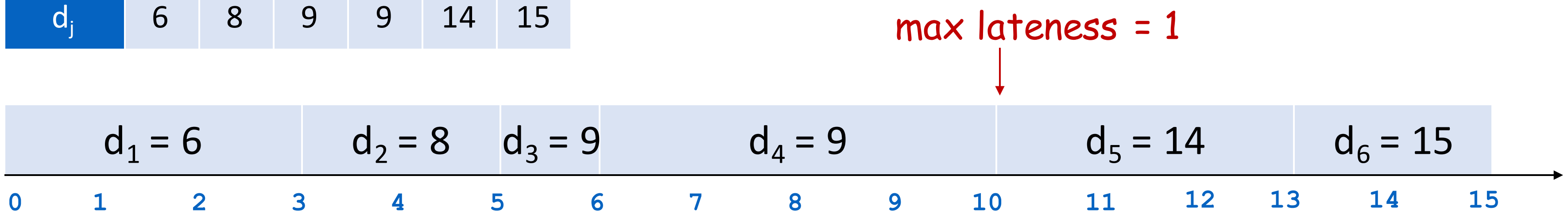
Minimizing Lateness: Greedy Algorithm

- **Greedy algorithm.** Earliest deadline first.

```
Sort n jobs by deadline so that d1 ≤ d2 ≤ ... ≤ dn
t ← 0        ← current start time
for j = 1 to n
   Assign job j to interval [t, t + tj]
   sj ← t, fj ← t + tj
   t ← t + tj
output intervals [sj, fj]
```

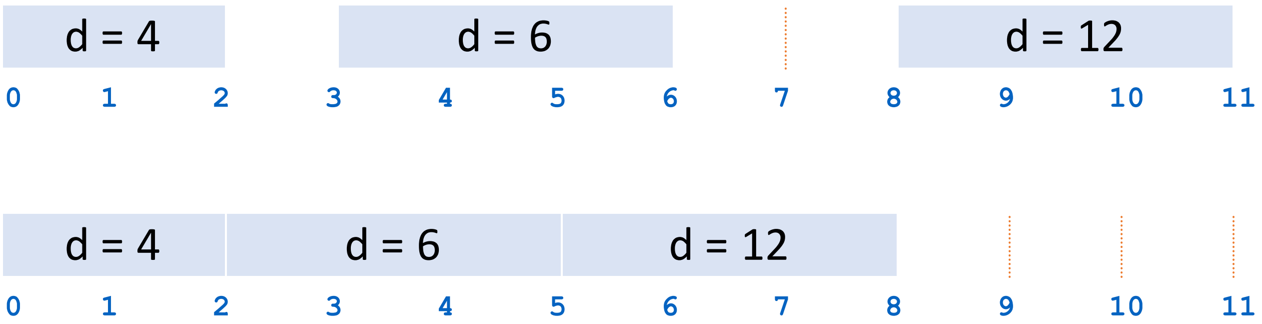
- **Ex:**

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15



Minimizing Lateness: No Idle Time

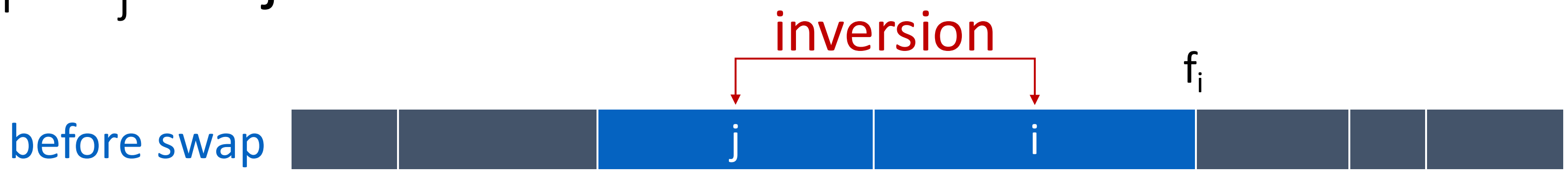
- **Observation.** There exists an optimal schedule with no idle time.



- **Observation.** The greedy 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: $d_i < d_j$ but j scheduled before i.

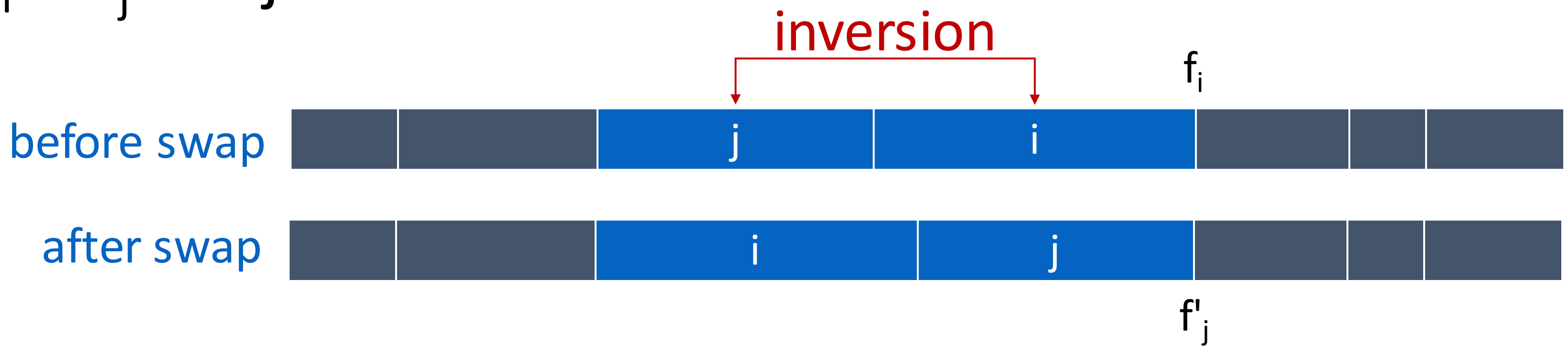


[as before, we assume jobs are numbered such that $d_1 \le d_2 \le \ldots \le d_n$]

- **Observation.** Greedy schedule has no inversions.
- **Observation.** If a schedule (with no idle time) has an inversion, it has one with a pair of inverted jobs scheduled consecutively.

Minimizing Lateness: Inversions

- **Def.** Given a schedule S, an inversion is a pair of jobs i and j such that: $d_i < d_j$ but j scheduled before i.



- **Claim.** Swapping two consecutive inverted jobs reduces the number of inversions by one 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 $k \neq i, j$
  - $\ell'_i \leq \ell_i$
  - $\ell'_j = \max\{ 0, f'_j - d_j \} = \max\{ 0, f_i - d_j \} \leq \max\{0, f_i - d_i \} = \ell_i$

Minimizing Lateness: Greedy Analysis

- **Theorem.** Greedy schedule S is optimal.
- Pf. Define S* to be an optimal schedule that has the fewest number of inversions, and let's see what happens.
  - Can assume S* has no idle time.
  - If S* has no inversions, then S = S*.
  - If S* has an inversion, let job pair (i, j) be an adjacent inversion.
    - ✓swapping i and j does not increase the maximum lateness and strictly decreases the number of inversions
    - ✓this contradicts 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.** GS, Kruskal, Prim, Dijkstra, Huffman, …