CS 580
ALGORITHM DESIGN AND ANALYSIS

Greedy Algorithms 2:
Scheduling (cf. KT 4.1 & 4.2)

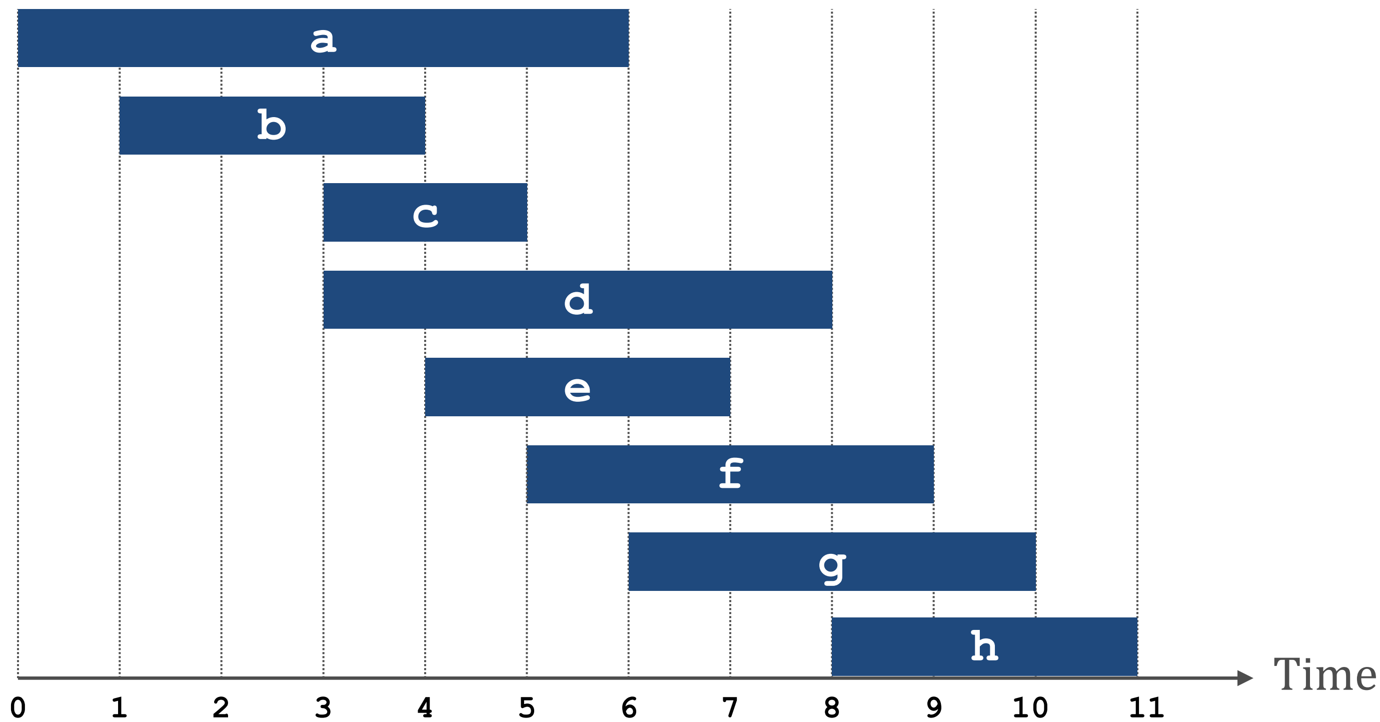
Vassilis Zikas

SO FAR

- Shortest paths
  - Dijkstra
  - Bellman-Ford
- Today:
  - Greedy algorithms that don’t have to do with graphs!
  - Interval scheduling (4.1 in KT)
  - Scheduling to minimize lateness (4.2 in KT)

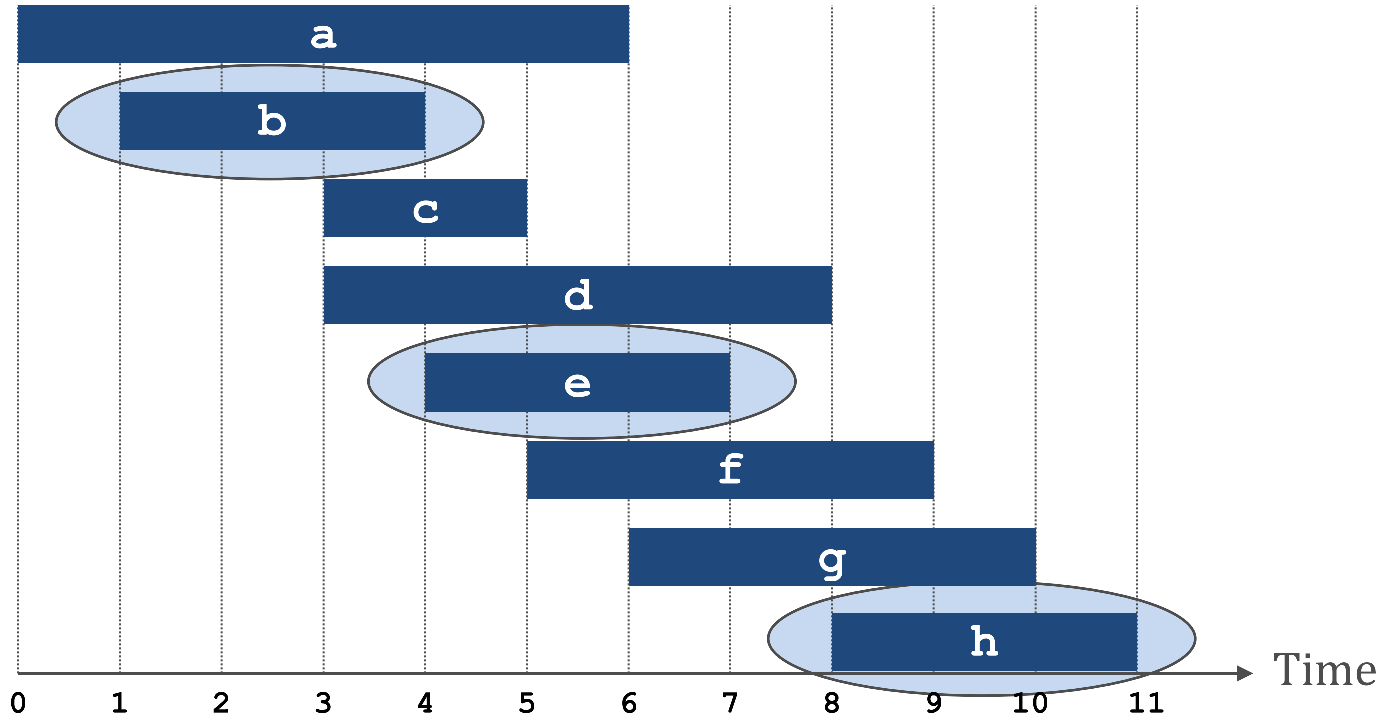
INTERVAL SCHEDULING

- There's an incoming set of jobs $\{1, \ldots, n\}$
- The ith job corresponds to an interval $[s_i, t_i]$
- Two jobs are compatible if they don't overlap
- Goal: find maximum subset of compatible jobs



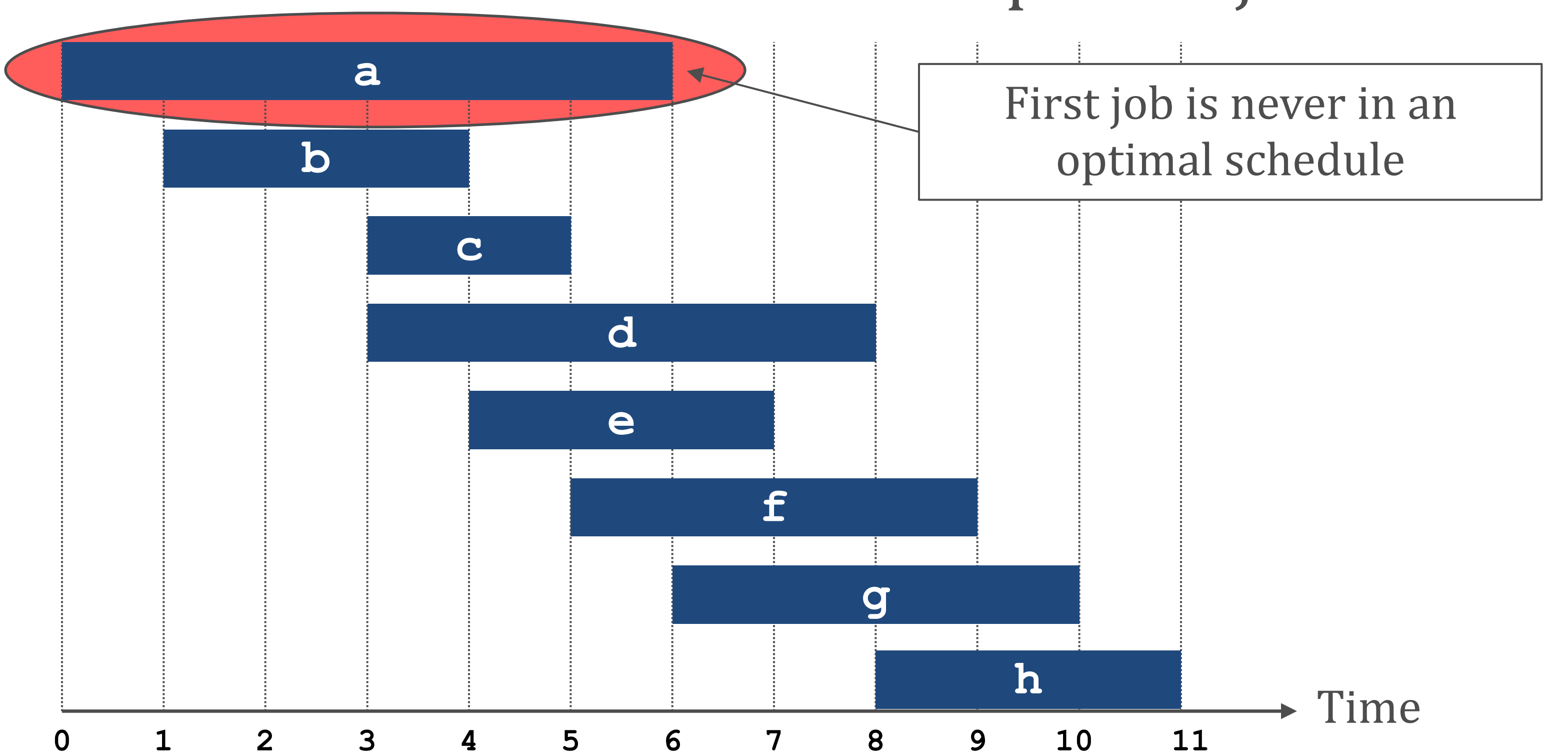
INTERVAL SCHEDULING

- There's an incoming set of jobs $\{1, \dots, n\}$
- The ith job corresponds to an interval $[s_i, t_i]$
- Two jobs are compatible if they don't overlap
- Goal: find maximum subset of compatible jobs



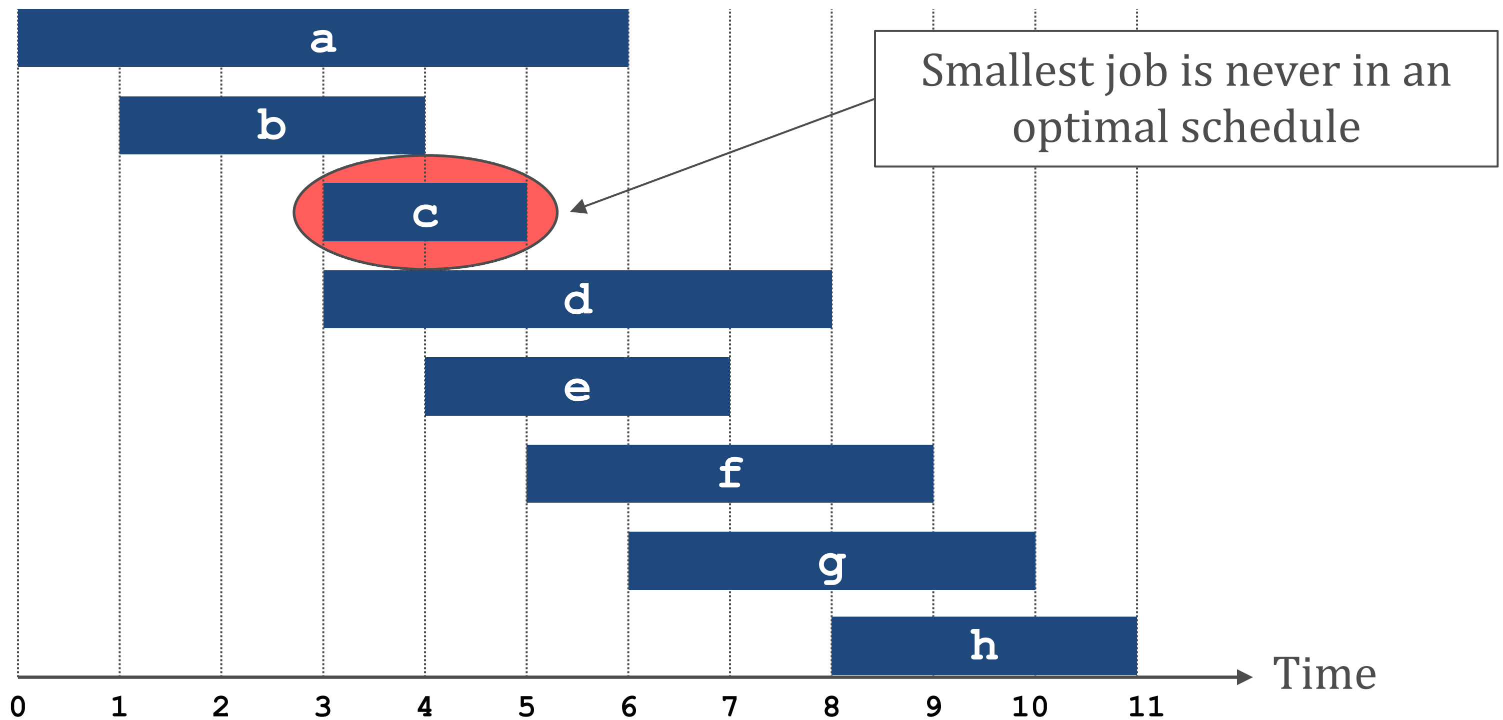
INTERVAL SCHEDULING

- There's an incoming set of jobs $\{1, \dots, n\}$
- The ith job corresponds to an interval $[s_i, t_i]$
- Two jobs are compatible if they don't overlap
- Goal: find maximum subset of compatible jobs



INTERVAL SCHEDULING

- There's an incoming set of jobs $\{1, \ldots, n\}$
- The ith job corresponds to an interval $[s_i, t_i]$
- Two jobs are compatible if they don't overlap
- Goal: find maximum subset of compatible jobs



INTERVAL SCHEDULING GREEDY TEMPLATES

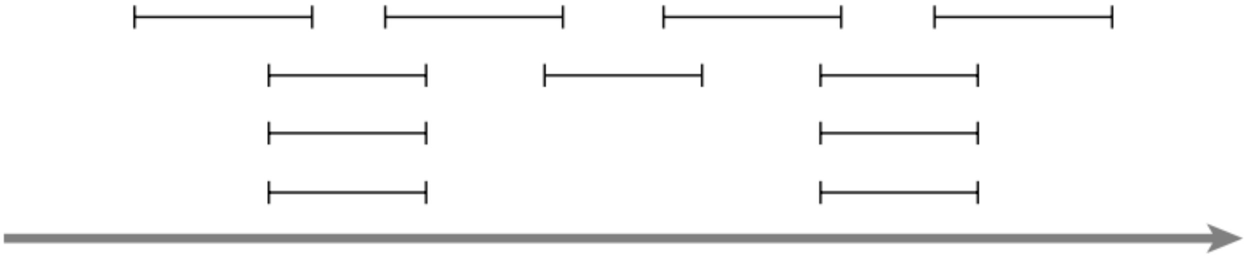
- Earliest start time?
- Earliest finish time?
- Pick smallest size first?
- Minimize number of overlaps?

INTERVAL SCHEDULING GREEDY TEMPLATES

- Earliest start time?
- Earliest finish time?
- Pick smallest size first?
- Minimize number of overlaps?

INTERVAL SCHEDULING GREEDY TEMPLATES

- Earliest start time?
- Earliest finish time?
- Pick smallest size first?
- Minimize number of overlaps?



INTERVAL SCHEDULING: EARLIEST FINISH TIME?

- Consider algorithms in increasing order of finish time t_i
  - Take a job if compatible with the ones picked so far

1. Sort by finish time to get $t_1 \leq t_2 \leq \cdots \leq t_n$
2. $A \leftarrow \emptyset$.   //Set of jobs selected so far
3. For $j = 1, \ldots, n$:
   - If j compatible with A: $A = A \cup \{j\}$

INTERVAL SCHEDULING

- Running time $O(nlogn)$
  - Sorting takes $O(nlogn)$
  - Checking if a job is compatible equivalent to $s_i \geq f_{j^*}$, the last job added: $O(n)$

1. Sort by finish time to get $t_1 \leq t_2 \leq \cdots \leq t_n$
2. $A \leftarrow \emptyset$. //Set of jobs selected so far
3. For $j = 1, \ldots, n$:
   - If j compatible with A: $A = A \cup \{j\}$

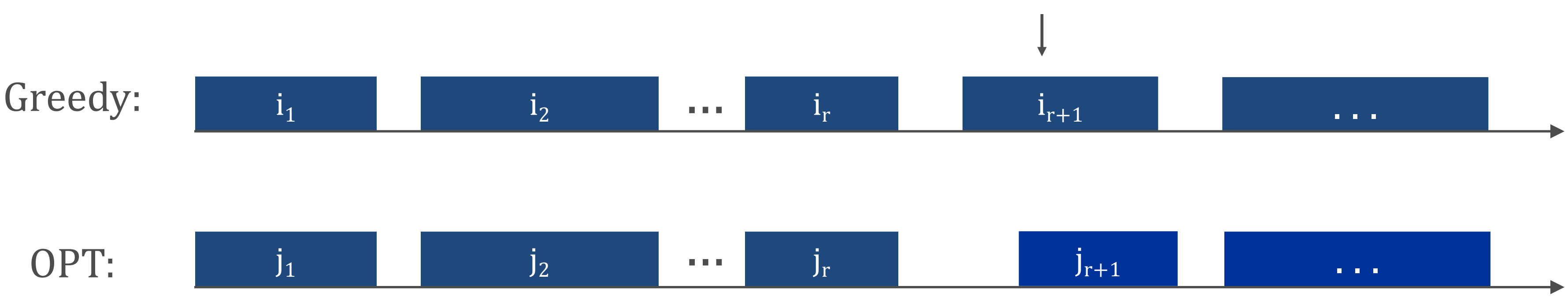
INTERVAL SCHEDULING

- Theorem: The greedy algorithm is optimal

Proof

- Assume greedy is not optimal
- Let $i_1, \dots, i_k$ be the jobs selected by greedy
- Let $j_1, \dots, j_m$ be the optimal set of jobs
    - $m > k$

By definition, i_{r+1} finishes before j_{r+1}



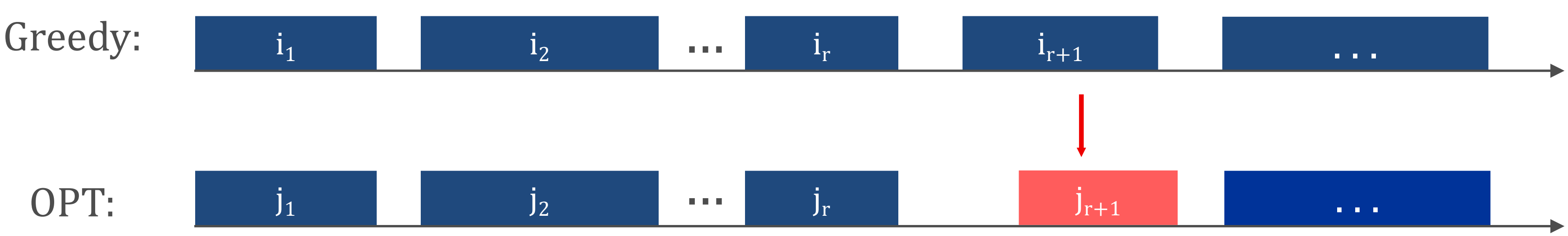
INTERVAL SCHEDULING

- Theorem: The greedy algorithm is optimal

Proof

- Assume greedy is not optimal
- Let $i_1, \dots, i_k$ be the jobs selected by greedy
- Let $j_1, \dots, j_m$ be the optimal set of jobs
  - $m > k$

Replacing j_{r+1} with i_{r+1} won't break feasibility



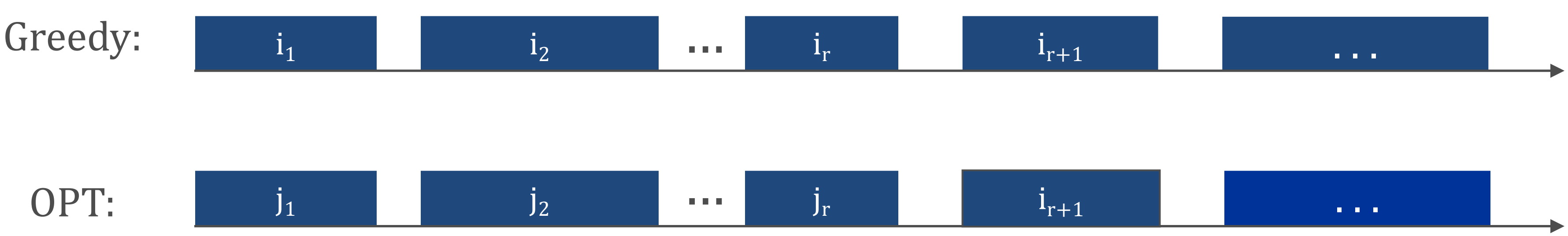
INTERVAL SCHEDULING

- Theorem: The greedy algorithm is optimal

Proof

- Assume greedy is not optimal
- Let $i_1, \ldots, i_k$ be the jobs selected by greedy
- Let $j_1, \ldots, j_m$ be the optimal set of jobs
  - $m > k$

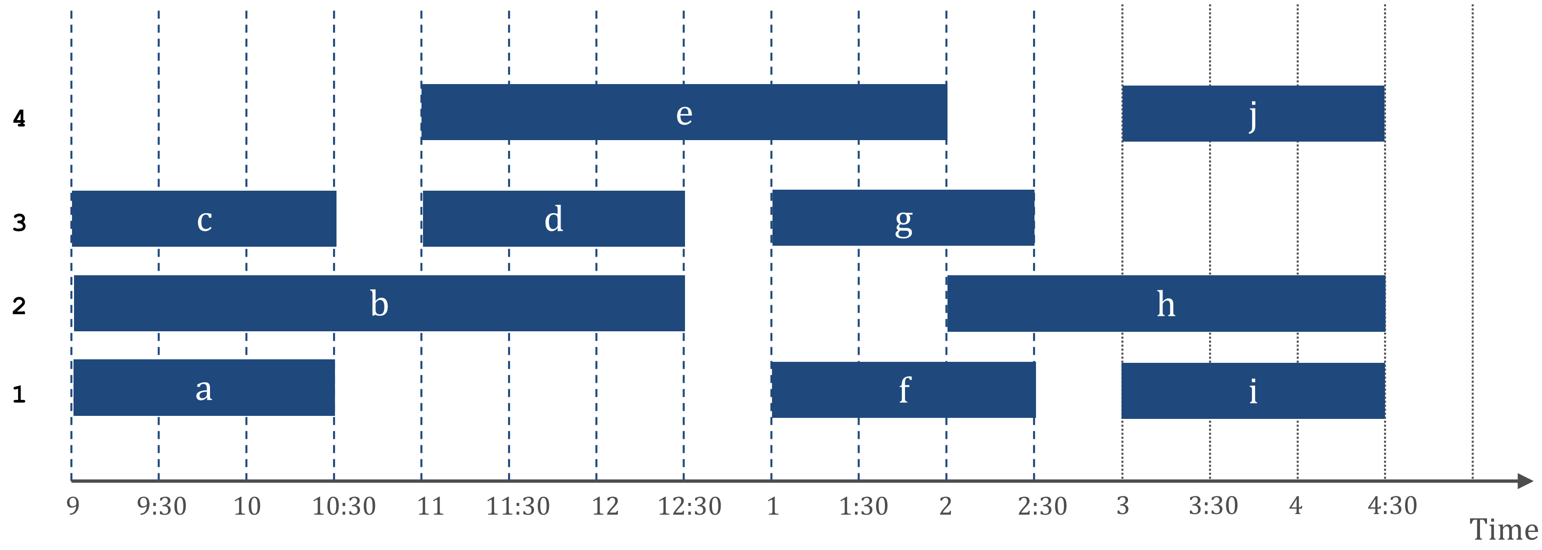
Size of OPT didn't change either
Contradiction: r was supposed to be maximal



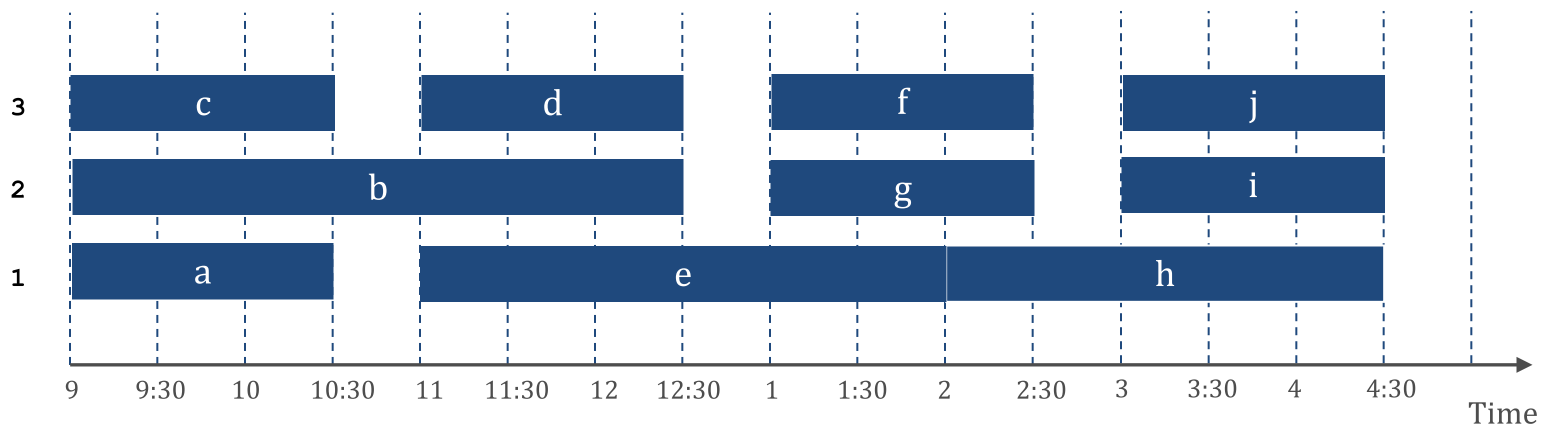
INTERVAL PARTITIONING (SCHEDULING ALL INTERVALS)

- 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

INTERVAL PARTITIONING (SCHEDULING ALL INTERVALS)

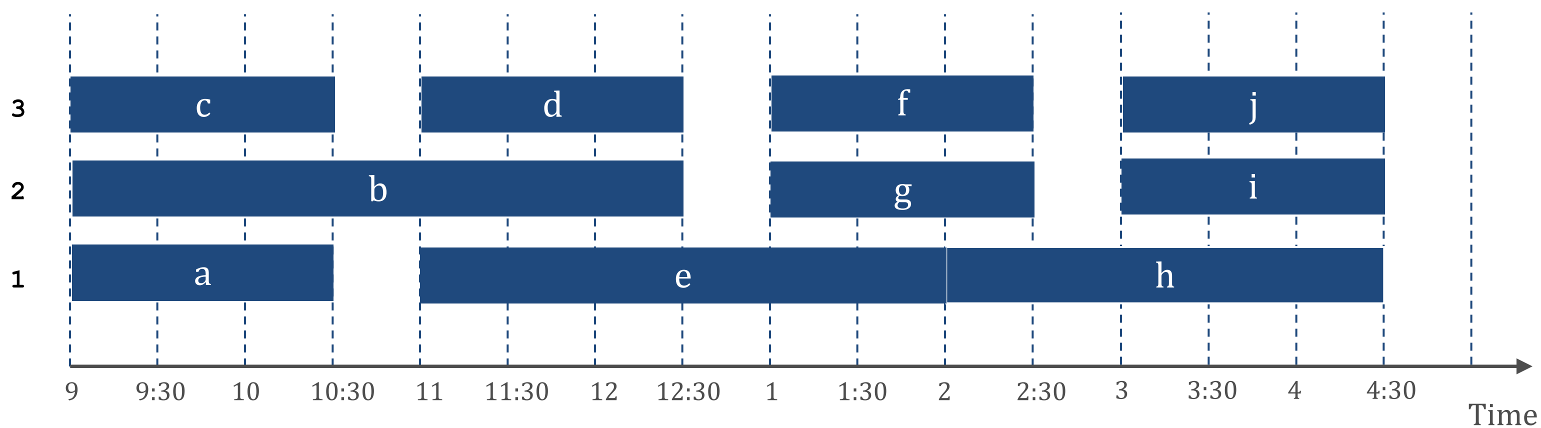


INTERVAL PARTITIONING (SCHEDULING ALL INTERVALS)



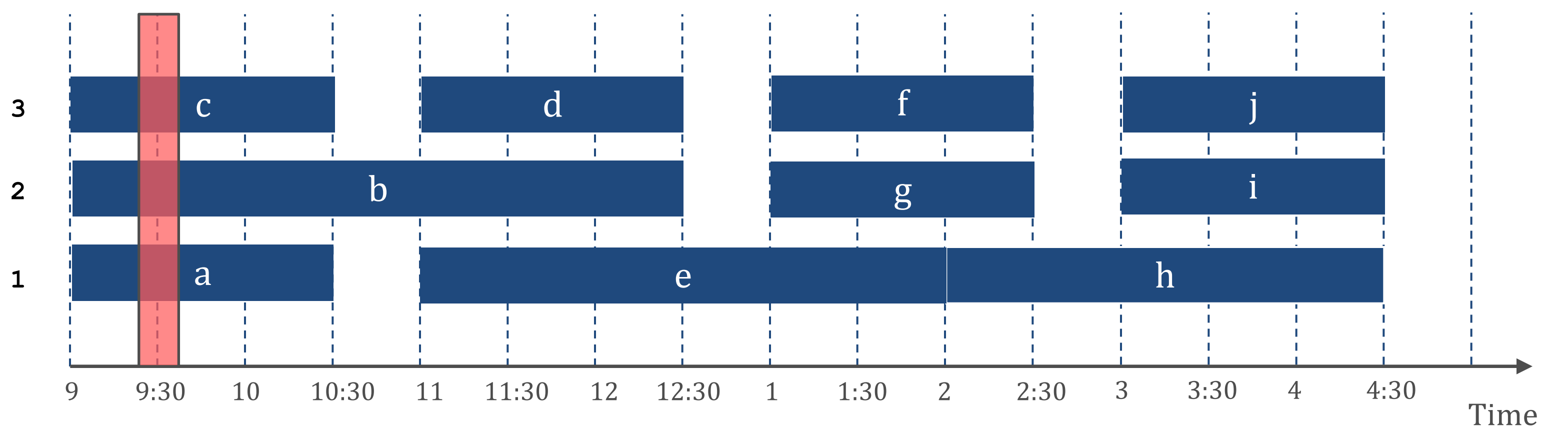
INTERVAL PARTITIONING (SCHEDULING ALL INTERVALS)

- Perhaps possible with two?
  - No: a, b and c overlap, so we need at least 3
- Definition: The depth of a set of intervals is the maximum number that pass over any single point in time



INTERVAL PARTITIONING (SCHEDULING ALL INTERVALS)

- Perhaps possible with two?
  - No: a, b and c overlap, so we need at least 3
- Definition: The depth of a set of intervals is the maximum number that pass over any single point in time



INTERVAL PARTITIONING

- Question: does there exist a schedule equal to the depth?
  - If the answer is yes, then this is clearly optimal

INTERVAL PARTITIONING

- Greedy algorithm: consider lectures in increasing starting time
  - assign to any compatible classroom (including a new one)

1. Sort intervals by start time to get $s_1 \leq \cdots \leq s_n$
2. For $j = 1$ to n:
   - If lecture j is compatible with some classroom k, schedule j in k
   - Otherwise, schedule j in a new classroom

INTERVAL PARTITIONING

- Implementation:
  - $O(n \log(n))$ to sort
  - At each time, need to figure out if a lecture is compatible
    - Use priority queue to remember end times of classrooms!

INTERVAL PARTITIONING

Theorem: Greedy is optimal

<u>Proof</u>

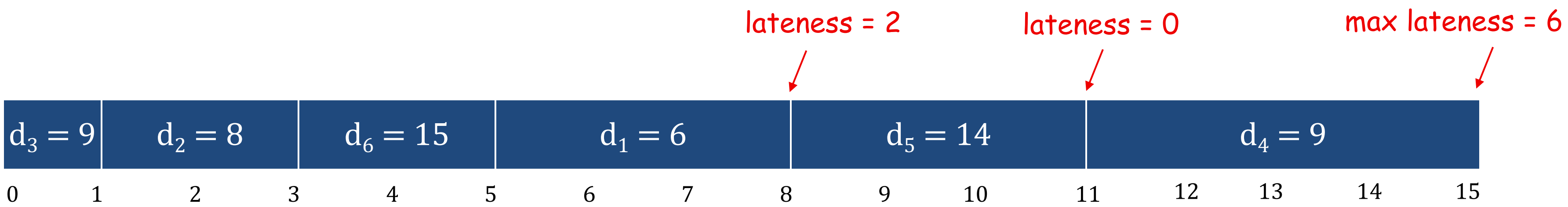
- Let d be the number of classrooms greedy needs
- Classroom d opened because we needed to schedule some job j that was incompatible with all $d - 1$ previous classrooms
- These d jobs (the $d - 1$ plus j) <u>all</u> **end** after s_j
- Since we sorted by start time, <u>all</u> these jobs also **start** before (or at) s_j
- Therefore, there exists a set of d jobs that overlap at s_j → need at least d classrooms

SCHEDULING TO MINIMIZE LATENESS

- Problem:
  - We have a single resource and need to process n jobs
  - We can process only one job at a time
  - Job j requires t_j units of time and expires at time d_j
    - If j starts at s_j it finishes at $f_j = s_j + t_j$
  - Lateness: $\ell_j = \max\{0, f_j - d_j\}$
  - Goal: minimize maximum latency $\mathrm{L} = \max_j \ell_j$

SCHEDULING TO MINIMIZE LATENESS

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



SCHEDULING TO MINIMIZE LATENESS

- Greedy template: Consider jobs in *some* order
  - Shortest processing time first
    - Sort by t_j (ascending)
  - Smallest slack first
    - Sort by $d_j - t_j$ (ascending)

SCHEDULING TO MINIMIZE LATENESS

- Greedy template: Consider jobs in *some* order
  - Shortest processing time first
    - Sort by t_j (ascending)
  - Smallest slack first
    - Sort by $d_j - t_j$ (ascending)

	1	2
t_j	1	10
d_j	100	10

Processing time ≠ Urgency
Deadline can be very far in the future

SCHEDULING TO MINIMIZE LATENESS

- Greedy template: Consider jobs in *some* order
  - Shortest processing time first
    - Sort by t_j (ascending)
  - Smallest slack first
    - Sort by $d_j - t_j$ (ascending)

	1	2
t_j	1	10
d_j	2	10

Deadline - Processing time ≠ Urgency
Ignores small deadlines that take little time

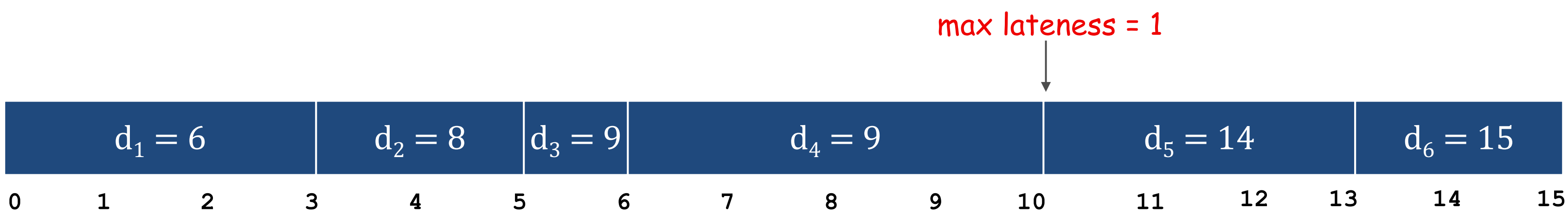
SCHEDULING TO MINIMIZE LATENESS

- Greedy algorithm: Earliest deadline first
  - Initially seems like a bad idea to not take processing time into account…

1. Sort jobs so that $d_1 \leq d_2 \leq \cdots \leq d_n$
2. t=0
3. For $j = 1 \ldots n$:
   - Assign job j to interval $[t, t + t_j]$
   - $t = t + t_j$

SCHEDULING TO MINIMIZE LATENESS

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

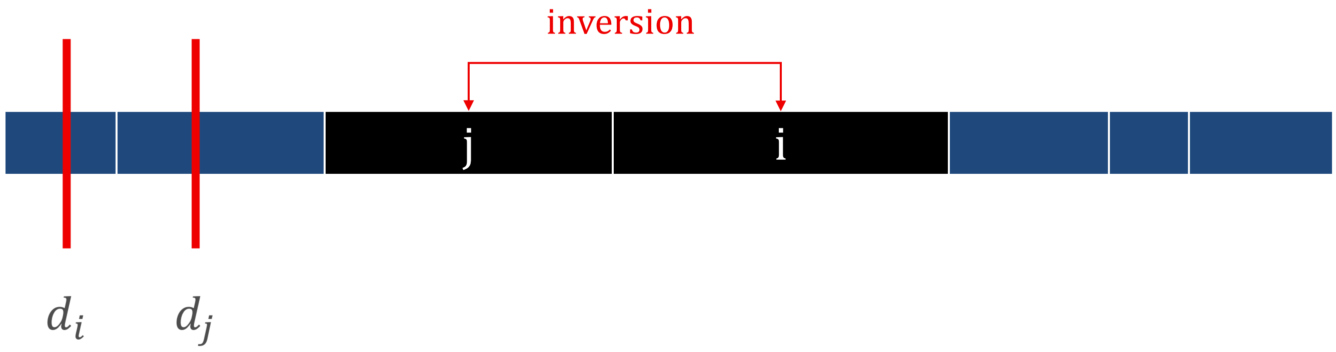


SCHEDULING TO MINIMIZE LATENESS

- **Observation**: There exists an optimal schedule with no idle time
  - Proof:
  - Assume the optimal schedule has some time interval $[t, t']$ where nothing is scheduled
  - For all jobs that start after t, move their start time earlier by $t' - t$
  - Still feasible, and latencies only decreased
- **Observation**: Greedy has no idle time

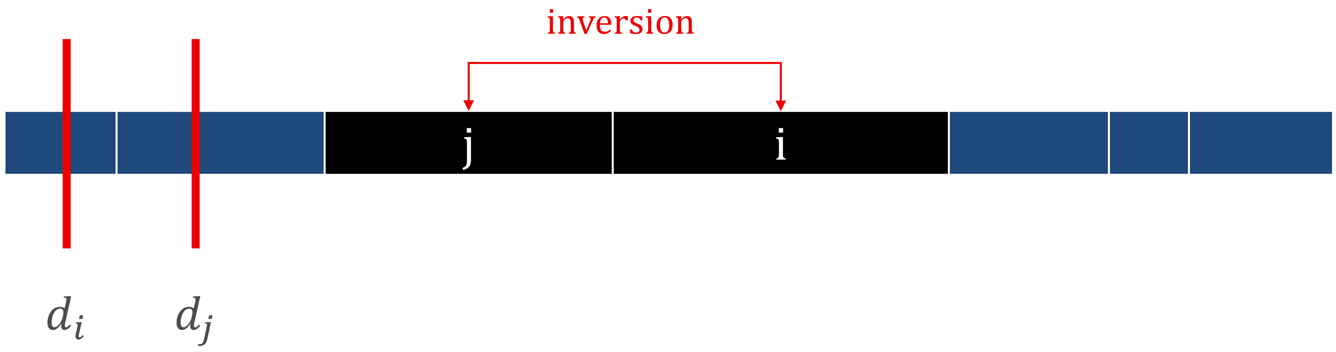
SCHEDULING TO MINIMIZE LATENESS

- **Definition**: Given a schedule S, an inversion is a pair of jobs i and j such that $d_i < d_j$ but j is scheduled before i
  - By definition, greedy has no inversions



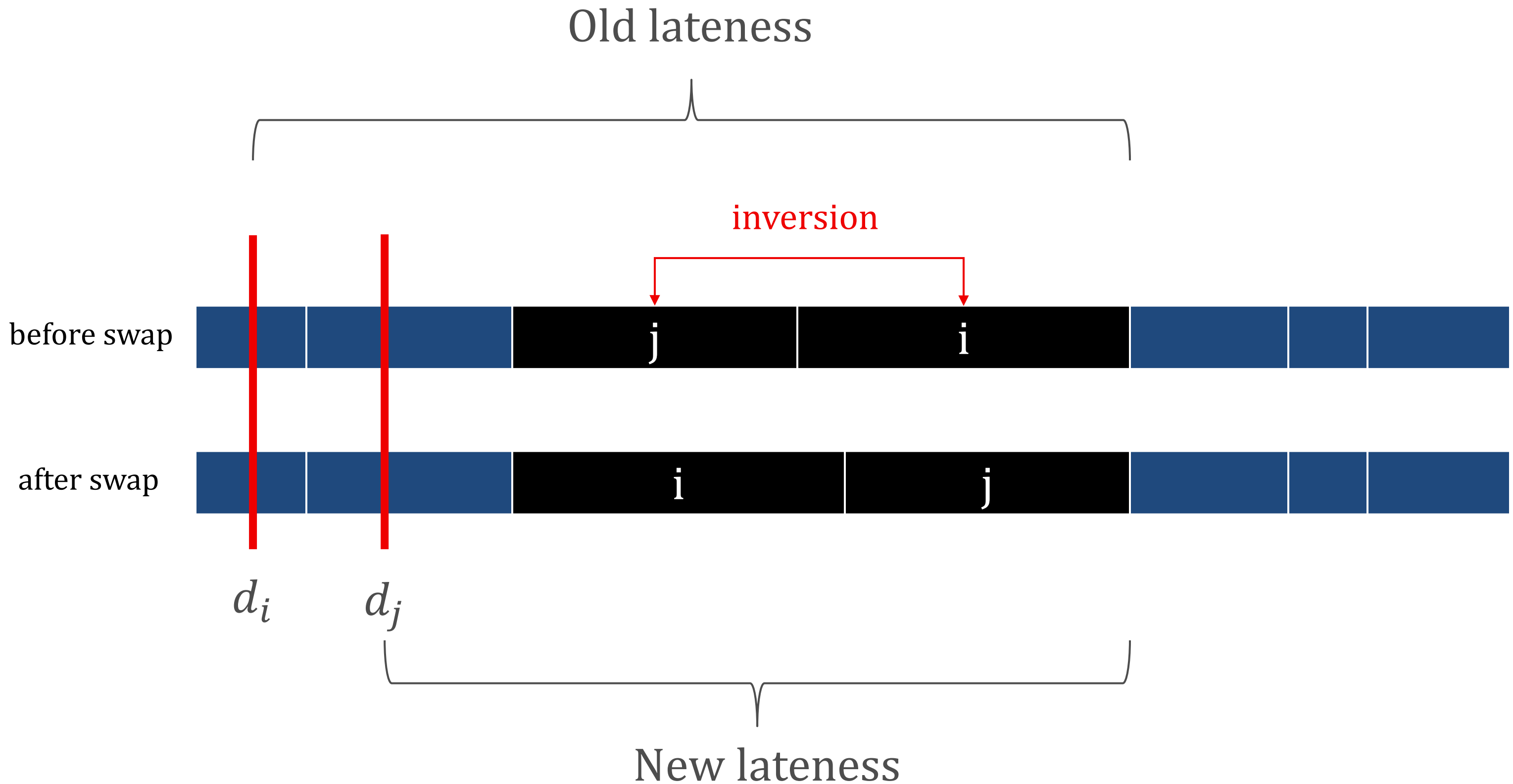
SCHEDULING TO MINIMIZE LATENESS

- Observation: If an inversion exists, then a "consecutive" inversion exists
  - Start from the left, until you find the first violation



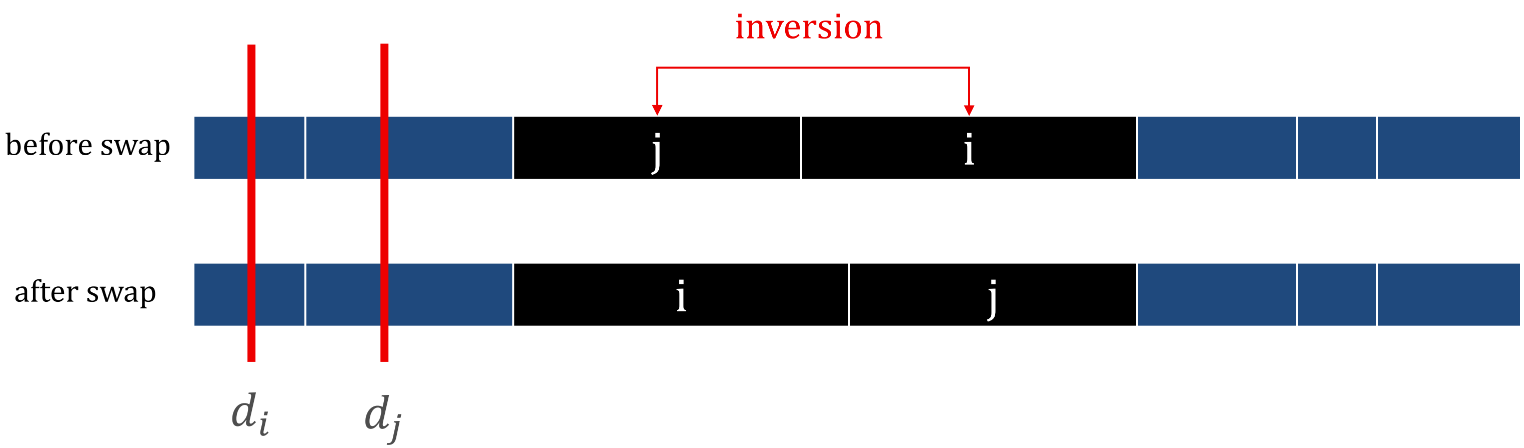
SCHEDULING TO MINIMIZE LATENESS

- What happens if we swap?
  - Everyone other than *i* and *j* remain unchanged



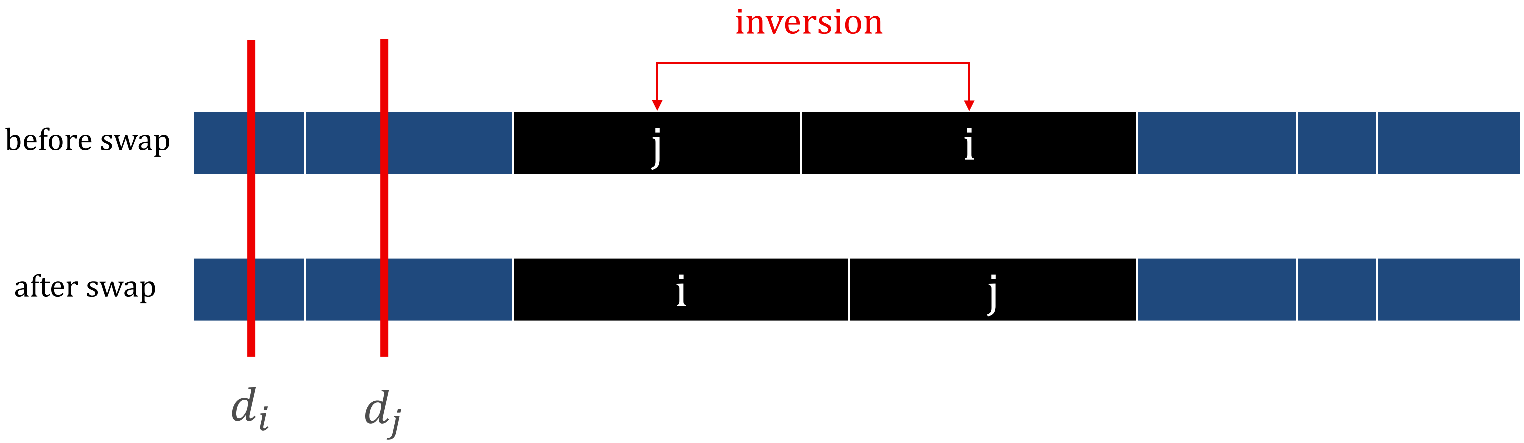
SCHEDULING TO MINIMIZE LATENESS

- What happens if we swap?
  - ℓ'_j



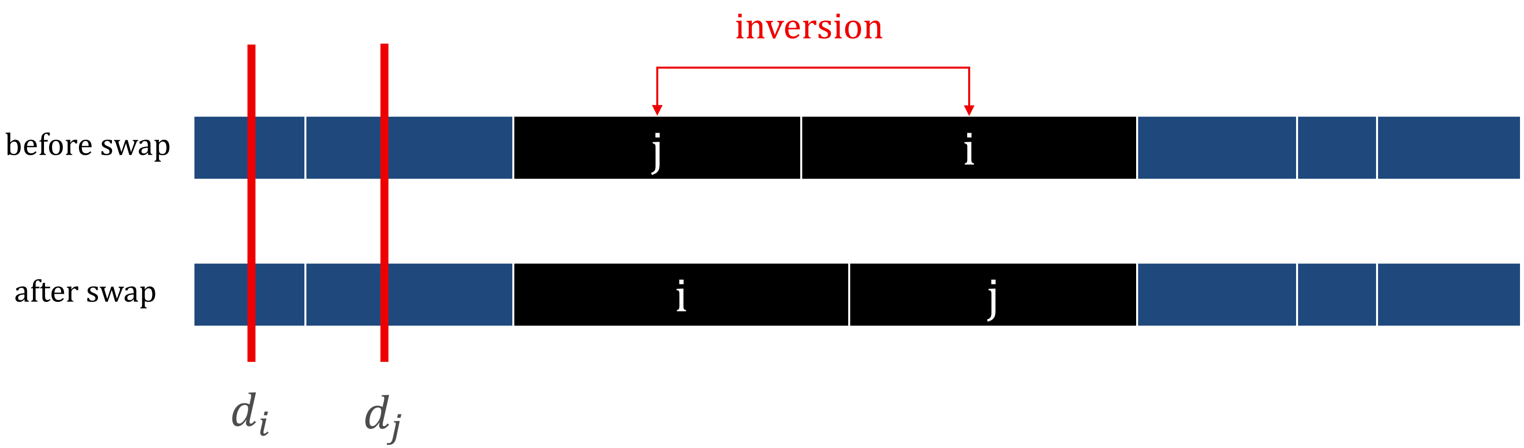
SCHEDULING TO MINIMIZE LATENESS

- What happens if we swap?
  - $\ell'_j = f'_j - d_j$



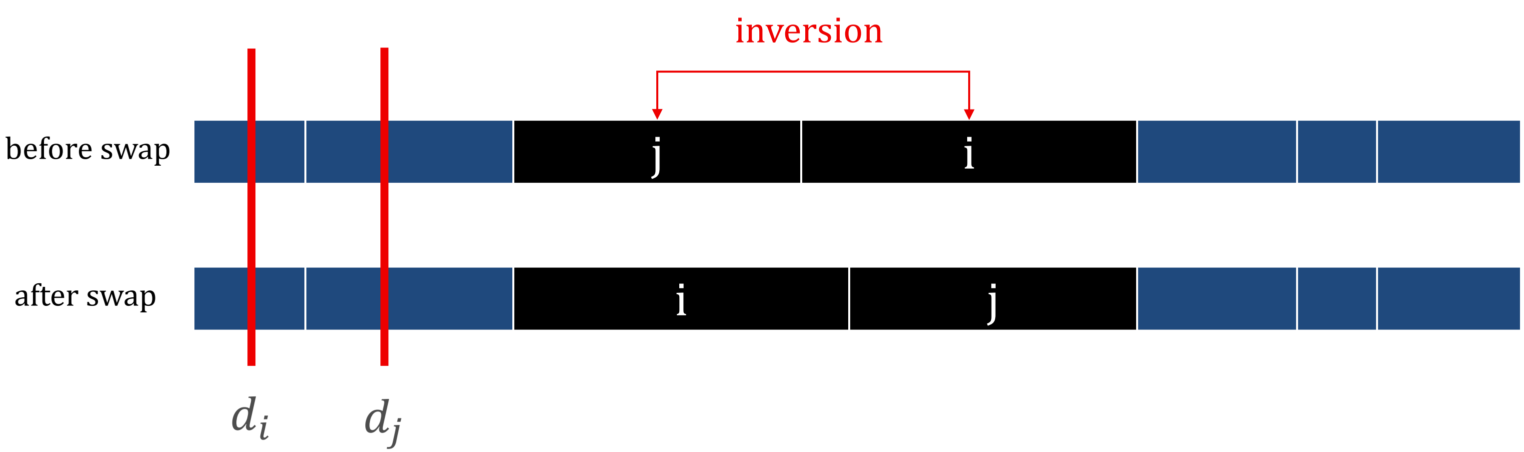
SCHEDULING TO MINIMIZE LATENESS

- What happens if we swap?
  - $\ell'_j = f'_j - d_j = f_i - d_j$



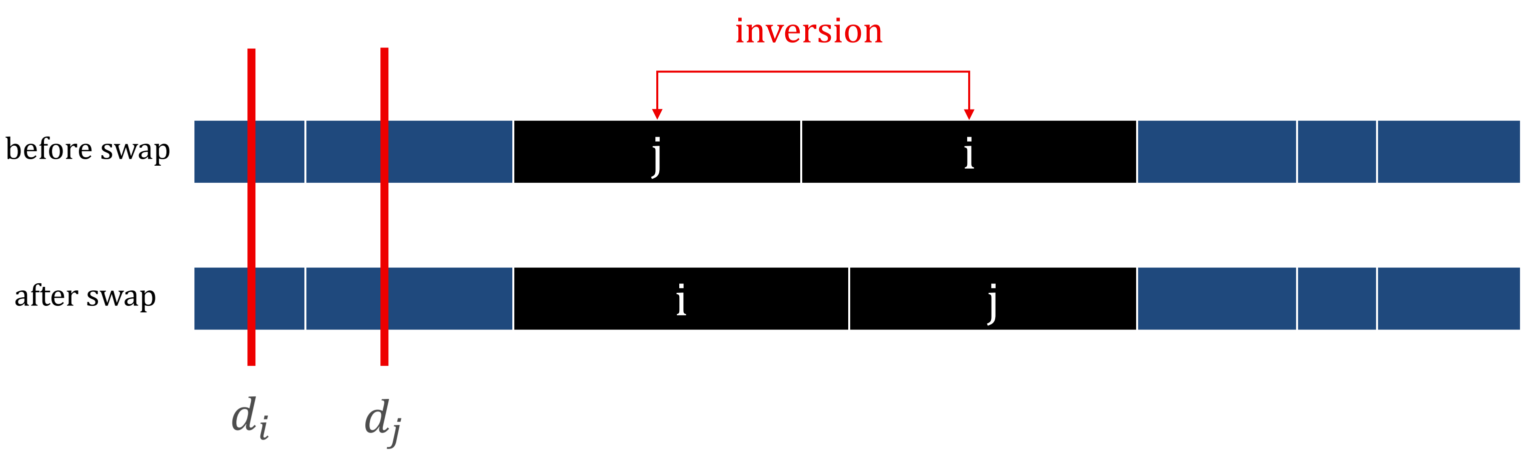
SCHEDULING TO MINIMIZE LATENESS

- What happens if we swap?
  - $\ell'_j = f'_j - d_j = f_i - d_j < f_i - d_i$



SCHEDULING TO MINIMIZE LATENESS

- What happens if we swap?
  - $\ell'_j = f'_j - d_j = f_i - d_j < f_i - d_i = \ell_i$



SCHEDULING TO MINIMIZE LATENESS

- Theorem: Greedy is optimal
- Proof:
- Let S^* be an optimal schedule (strictly) better than greedy, with the fewest number of inversions
- Wlog S^* has no idle times
- If S^* has no inversions, then $S^* = S$
- If S^* has an inversion, let (i, j) be an adjacent inversion: swap it!
  - Maximum latency didn't decrease (otherwise S^* is not optimal)
  - But, number of inversions decreased! Contradiction

SUMMARY

- Interval scheduling
- Interval Partioning
- Scheduling to minimize latency

- Meta lesson: similar (same?) analysis
    - Gradually transform the solution of any other algorithm to one of greedy, without changing the performance