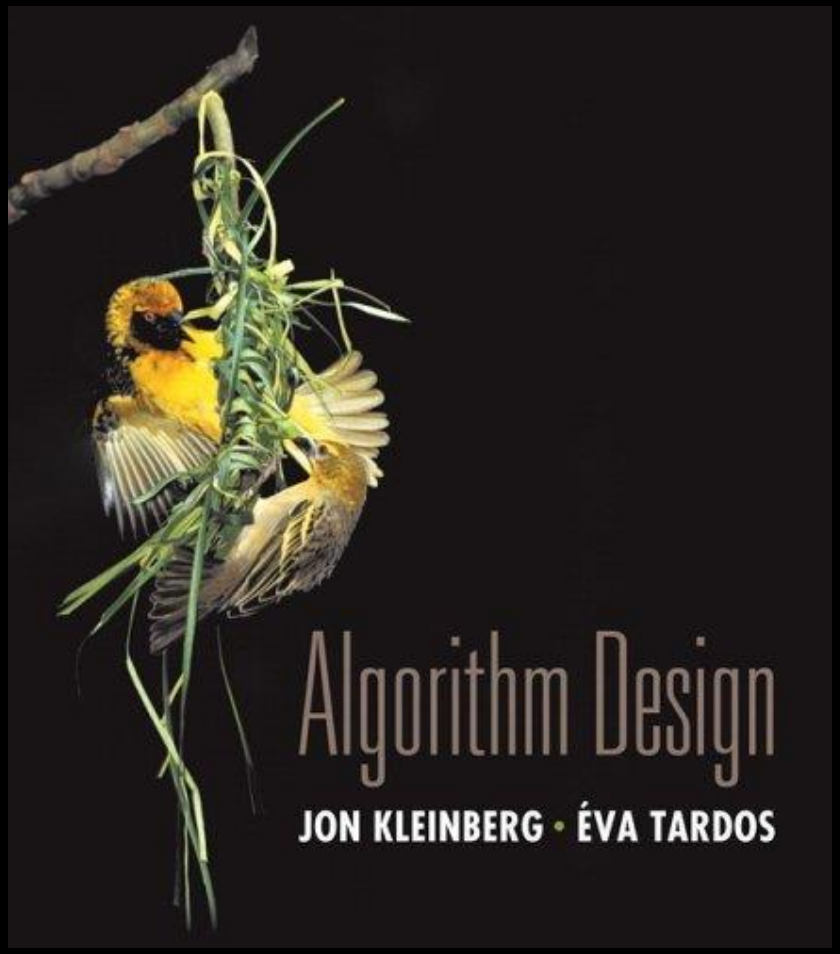


Chapter 4

Greedy Algorithms

Greedy Algorithms

General greedy algorithms. Consider the items in a specific order, makes "greedy" decisions

Pro: Typically very simple to implement and very efficient

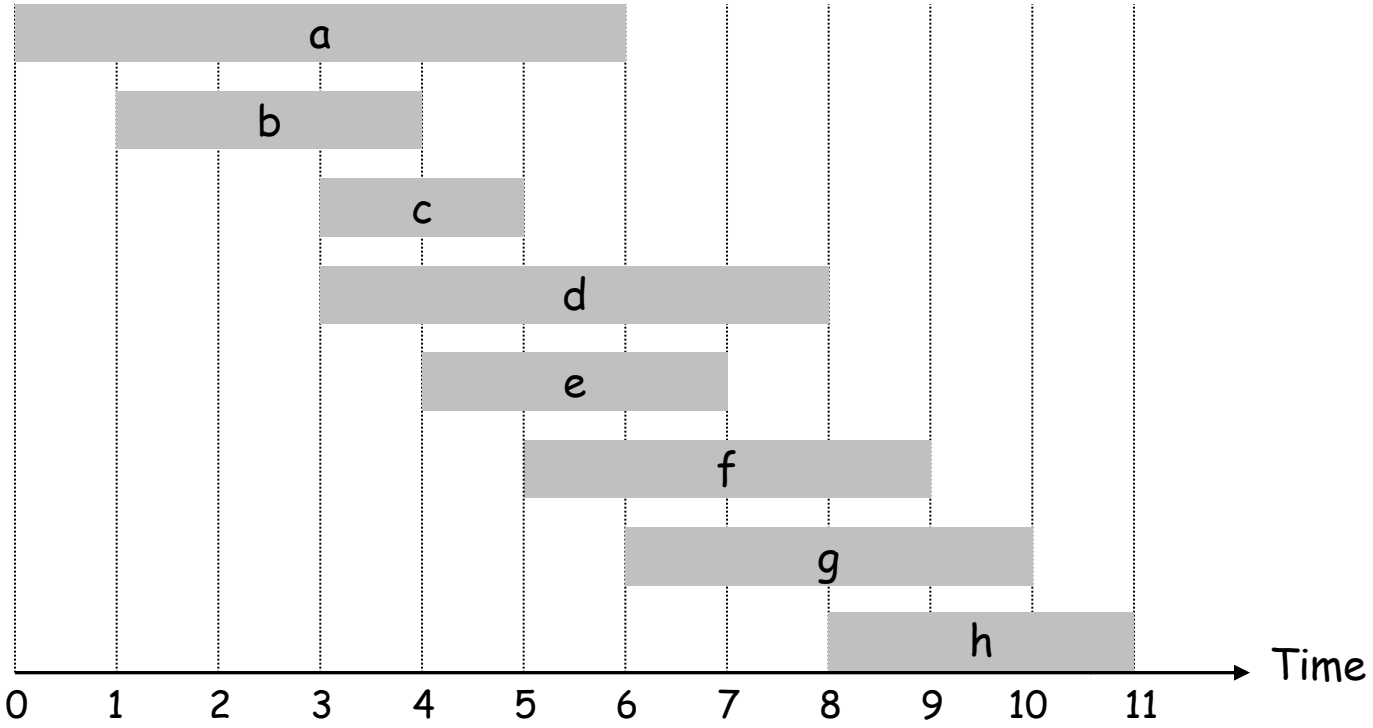
Con: Cannot solve every problem using a greedy algorithm, need to guarantee it works

4.1 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 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.

breaks earliest start time

breaks shortest interval

breaks 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 f_1 ≤ f_2 ≤ ... ≤ f_n.

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

Implementation.

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

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 f_1 ≤ f_2 ≤ ... ≤ f_n.

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

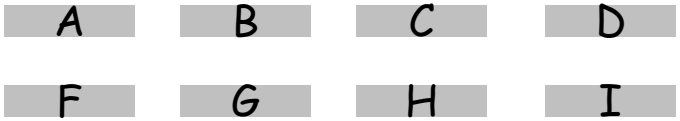
Implementation. $O(n \log n)$.

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

Interval Scheduling: Analysis

Need to argue that greedy returns optimal solution

Optimal Solutions. There may be more than one optimal solutions for the same instance



2x2x2x2 = 16 optimal solutions

We will look at optimal solution that is most similar to the greedy solution…

jobs selected by both solutions

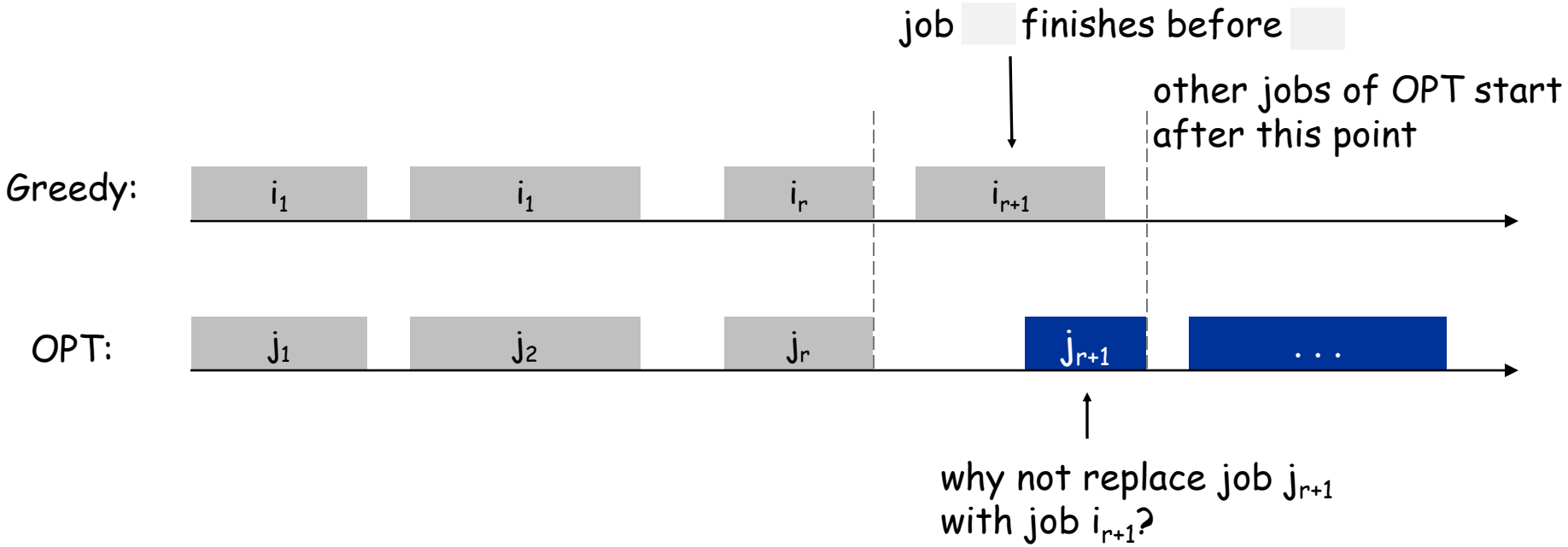
… and show that solutions are actually the same

Interval Scheduling: Analysis

Theorem. Greedy on finish time 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 the set of jobs of the optimal solution which is the most similar to greedy's solution

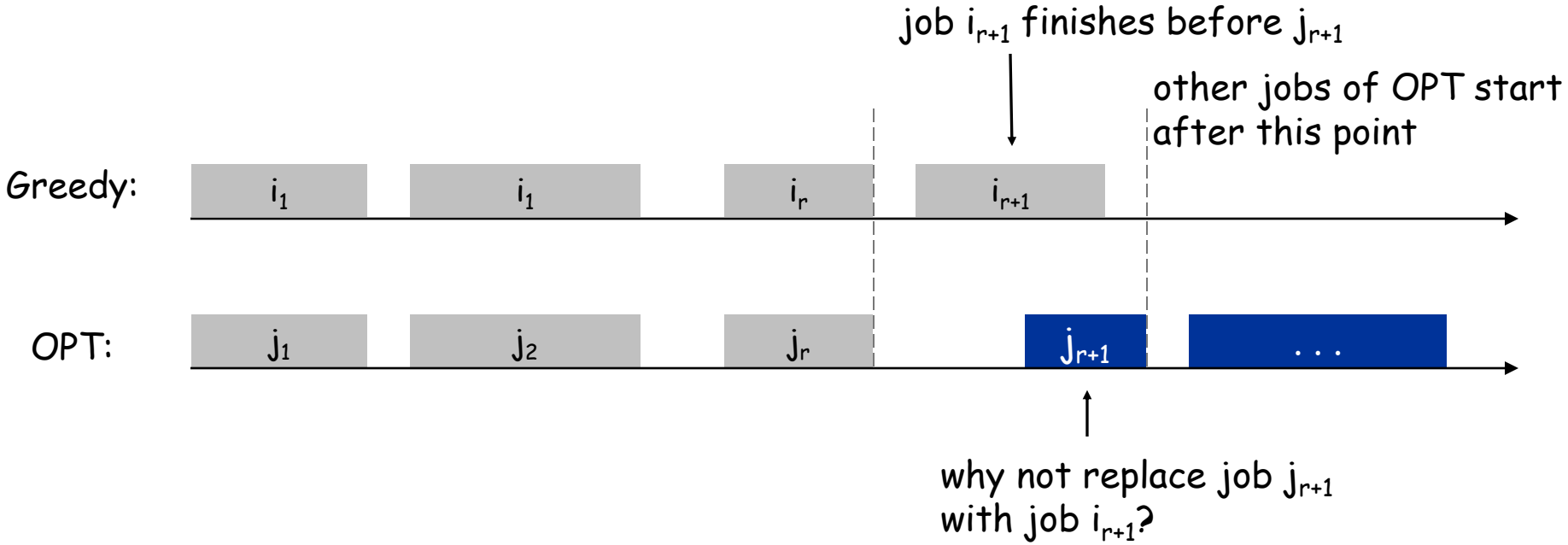


Interval Scheduling: Analysis

Theorem. Greedy on finish time 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 the set of jobs of the optimal solution which is the most similar to greedy's solution

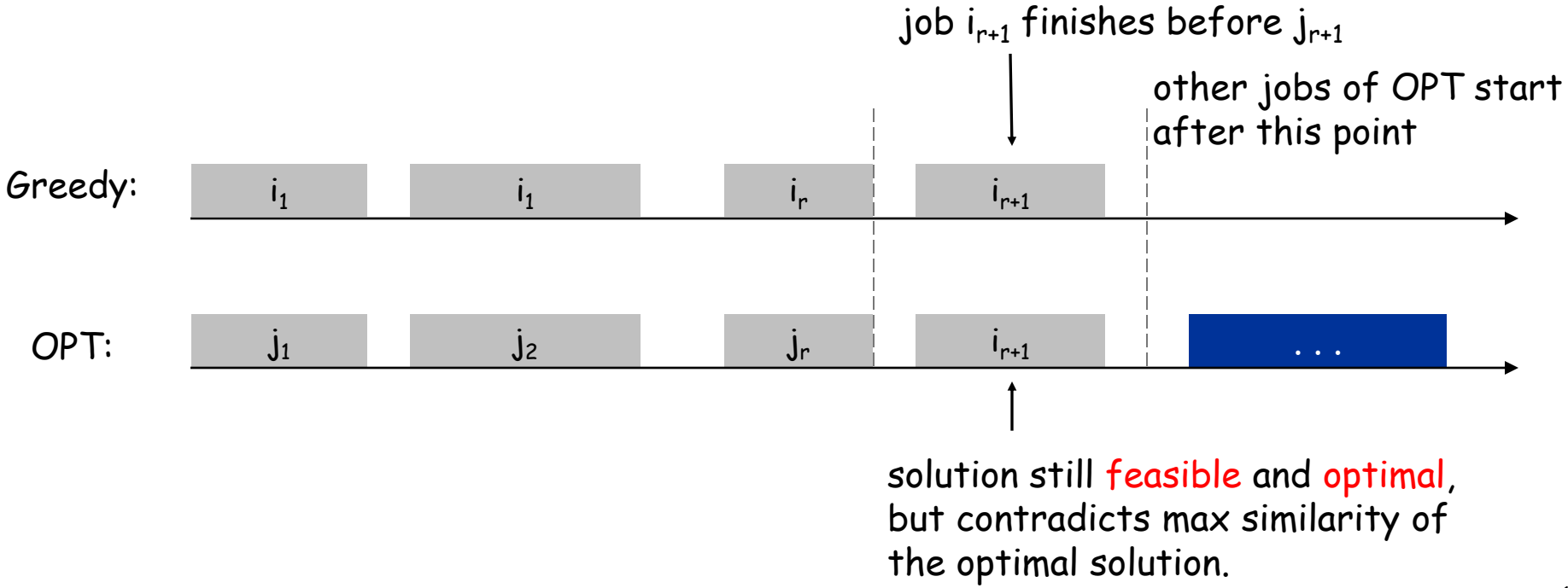


Interval Scheduling: Analysis

Theorem. Greedy on finish time 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 the set of jobs of the optimal solution which is the most similar to greedy's solution



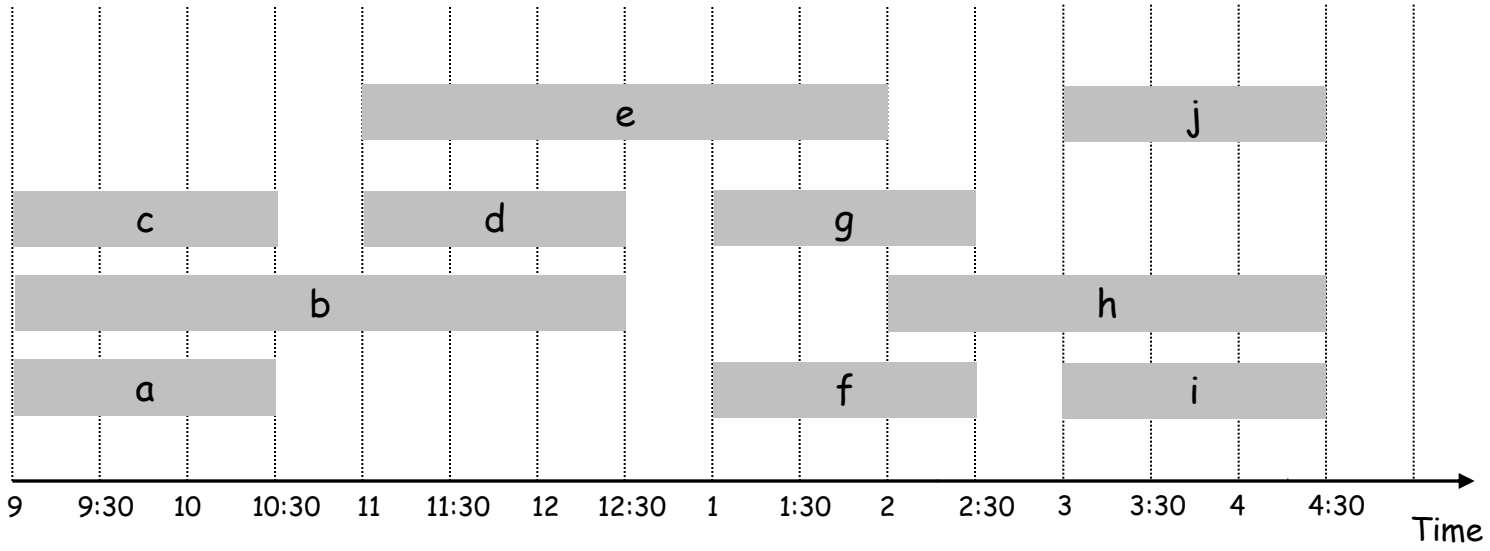
4.1 Interval Partitioning

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.
- (ignore overlap at start/finish)

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

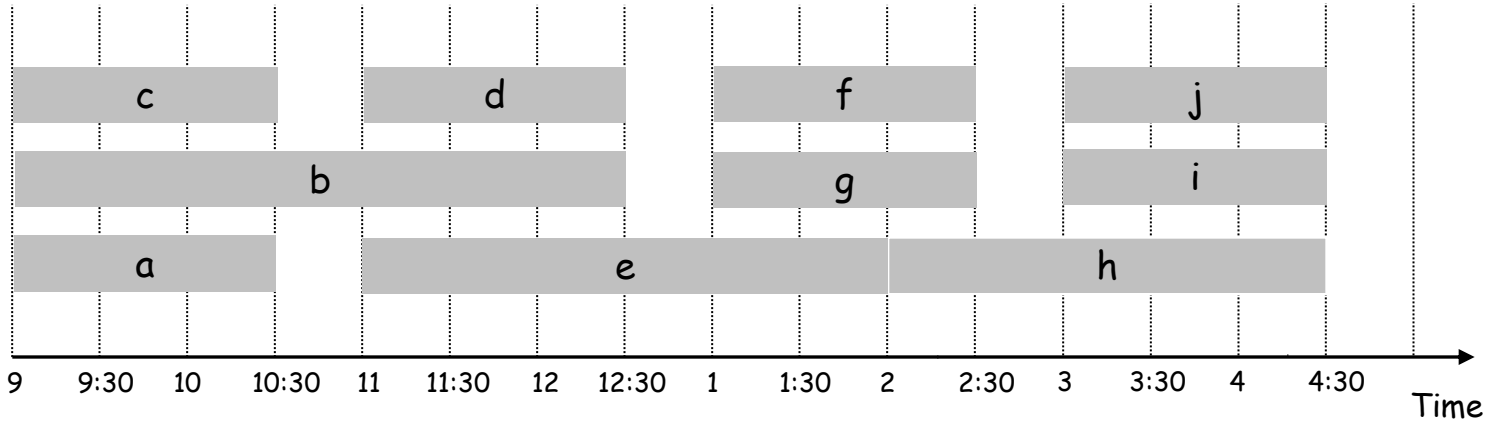


Interval Partitioning

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.
- (ignore overlap at start/finish)

Ex: This schedule uses only 3.



Interval Partitioning: Lower Bound on Optimal Solution

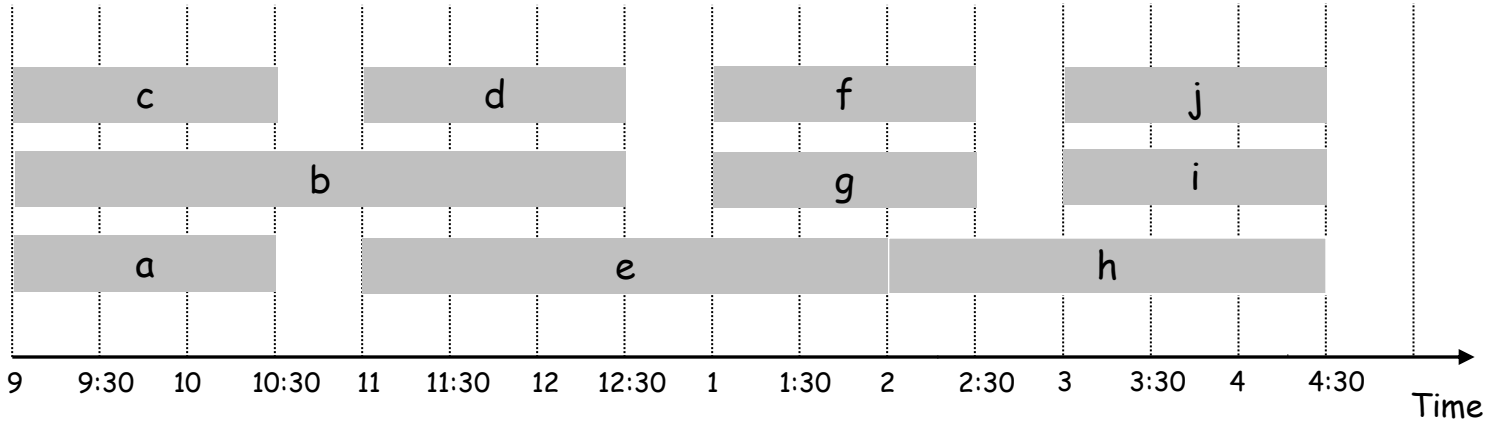
Def. The depth is defined as the maximum number of lectures that overlap at one given time

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.

```
Sort intervals by starting time so that s_1 ≤ s_2 ≤ ... ≤ s_n.
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
}
```

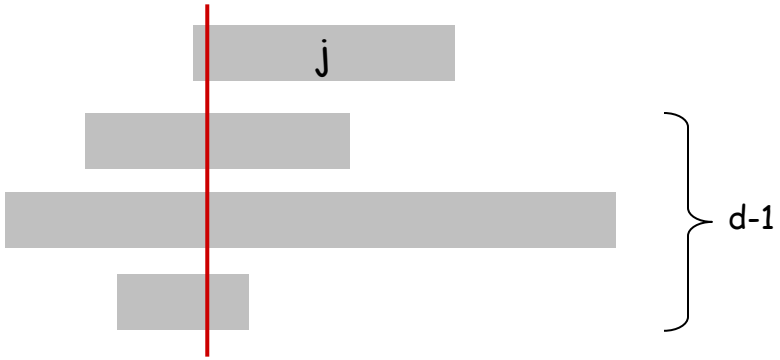
Interval Partitioning: Greedy Analysis

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 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 $s_j + \varepsilon$.
- Key observation $\Rightarrow$ all schedules use $\geq$ d classrooms. ▪



Interval Partitioning: Greedy Algorithm

Implementation. $O(nd)$.

- For each classroom k, maintain the finish time of the last job added.
- Keep the classrooms in a list

[if (lecture j is compatible with some classroom k)]

Compare start time of j with the finish time of each class: O(d) time

[schedule lecture j in classroom k]

Replace the current finish time of class k to f_j: O(1) time

[allocate a new classroom] : O(1) time

[schedule lecture j in the new classroom]

Insert a new class in the list of classes with finish time f_j: O(1) time

Interval Partitioning: Greedy Algorithm

Implementation. $O(n \log d)$.

- For each classroom k, maintain the finish time of the last job added.
- Keep the classrooms in a heap

[if (lecture j is compatible with some classroom k)]

Compare start time of j with the finish time of the class at the root of the heap

[schedule lecture j in classroom k]

Change the priority of the root of heap to f_j

Update the heap : O(log d) time

[allocate a new classroom]

[schedule lecture j in the new classroom]

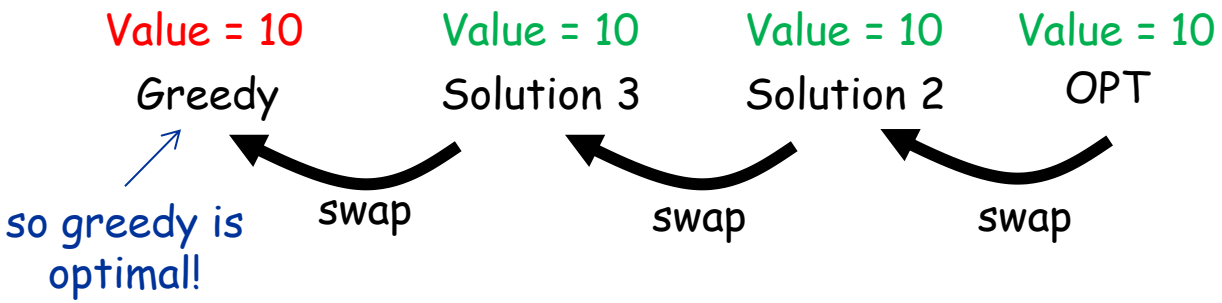
Insert a new class at the heap with priority f_j: O(log d)

Greedy Algorithms

The hard part of greedy algorithms is to argue it returns the optimal solution

A very comon strategy for analysis is the swap:

- Replace one action of OPT for one action of Greedy, and show value does not get worse
- Repeating we bring OPT closer and closer until we reach Greedy



This is how we analyzed the Interval Scheduling problem

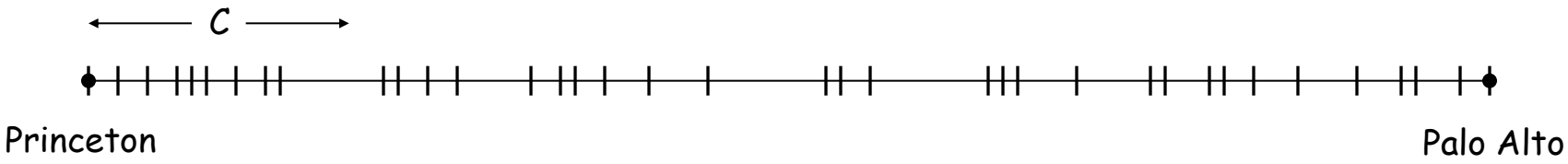
Let's see a few more examples

Example: Selecting fueling points

Selecting fueling points

Selecting fueling points.

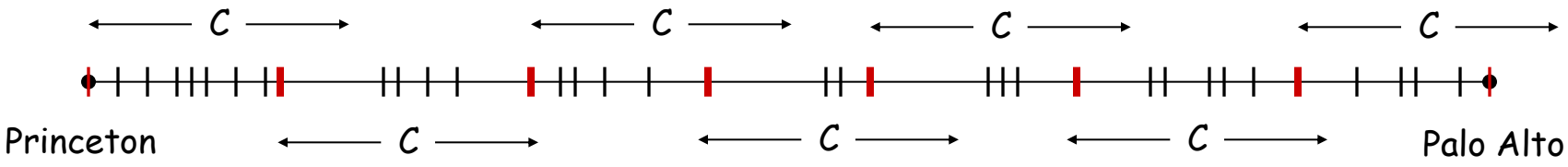
- Road trip from Princeton to Palo Alto along fixed route.
- There are refueling stations at certain points along the way.
- Fuel capacity = C.
- Goal: Choose smallest number of refueling stops that make the trip possible



Selecting fueling points

Selecting fueling points.

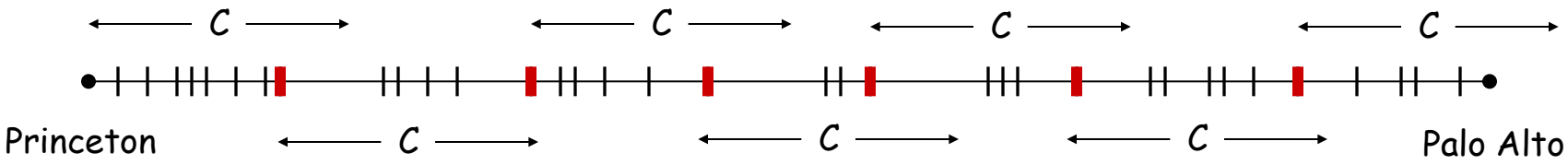
- Road trip from Princeton to Palo Alto along fixed route.
- There are refueling stations at certain points along the way.
- Fuel capacity = C.
- Goal: Choose smallest number of refueling stops that make the trip possible



Task: Design a greedy algorithm for this problem, and give a brief argument for why it returns an optimal solution

Selecting fueling points

Solution: Go as far as you can before refueling.



To argue it returns an optimal solution, think "where could it go wrong/where is another algorithm smater"?

Maybe another algorithm could stop before it is required

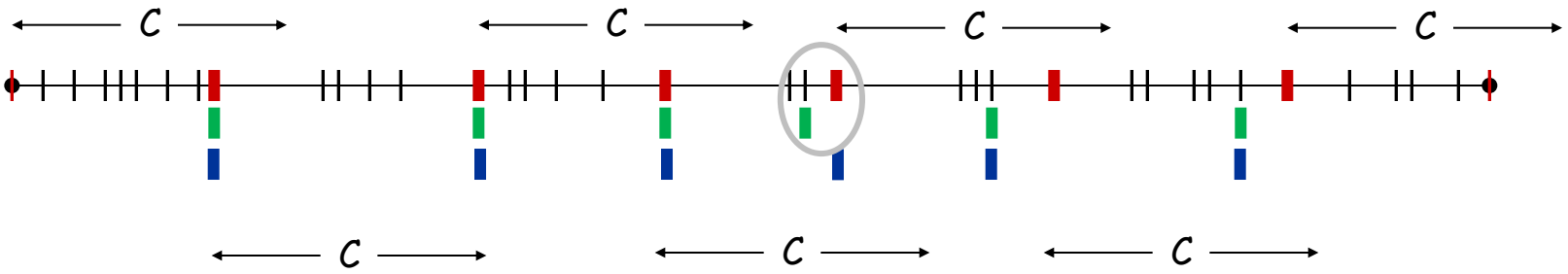
But that does not seem to help...

Selecting Breakpoints: Correctness

Suppose these are the **greedy** solution and the **OPT** solution

Q: Is there a swap that gives a valid solution of same cost as OPT but brings OPT closer to greedy?

A: Swap first stop where they disagree (swapped solution in **blue**)



Reapeating such swap shows that greedy has same cost as OPT

Theorem. Greedy algorithm is optimal.

Scheduling to Minimize Lateness

Scheduling to Minimizing Lateness

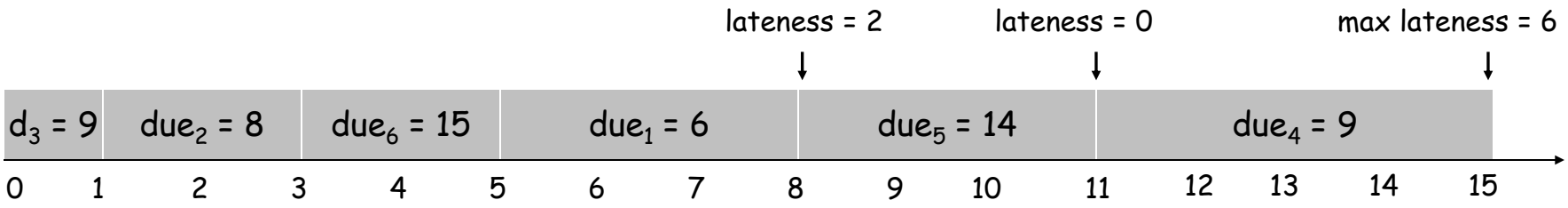
Minimizing lateness problem.

- Machine processes one job at a time.
- Job j requires $time_j$ units of processing time and is due at time due_j.
- If j starts at time $start_j$, it finishes at time $finish_j = start_j + time_j$.
- Lateness: $late_j = 0$ if finished before deadline, otherwise $late_j = due_j - finish_j$
- Goal: schedule all jobs to minimize maximum lateness = max $late_j$.

Aplication: Project management

Ex:

	1	2	3	4	5	6
$time_j$	3	2	1	4	3	2
due_j	6	8	9	9	14	15



[Try to find optimal solution for this example]

Minimizing Lateness: Greedy Algorithms

Greedy template. Process jobs in some order.

- [Shortest processing time first] Consider jobs in ascending order of processing time $time_j$.

- [Earliest deadline first] Consider jobs in ascending order of deadline due_j.

- [Smallest slack] Consider jobs in ascending order of slack $due_j - time_j$.

Minimizing Lateness: Greedy Algorithms

Greedy template. Process jobs in some order.

- [Shortest processing time first] Consider jobs in ascending order of processing time $time_j$.

	1	2
t_j	1	10
d_j	100	10

counterexample

- [Smallest slack] Consider jobs in ascending order of slack $due_j - time_j$.

	1	2
t_j	1	10
d_j	2	10

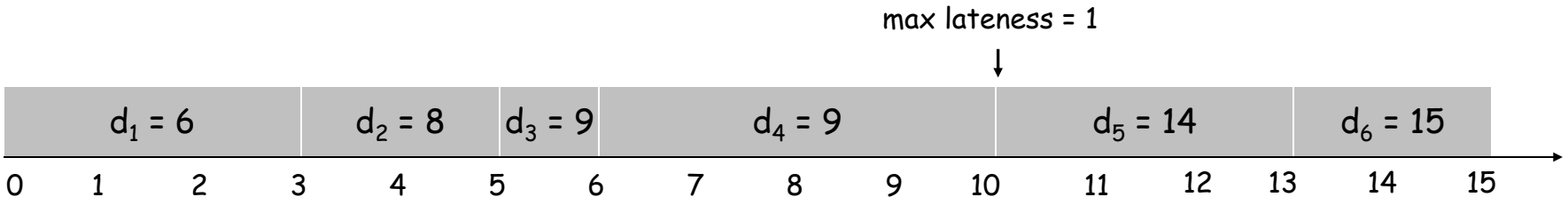
counterexample

Minimizing Lateness: Greedy Algorithm

Greedy algorithm. Earliest deadline first.

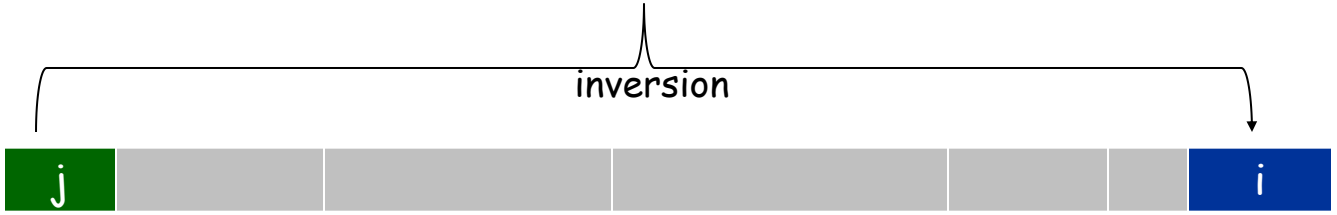
```
Sort n jobs by deadline so that d_1 ≤ d_2 ≤ … ≤ d_n

for j = 1 to n
   Assign job j to next available time period
```



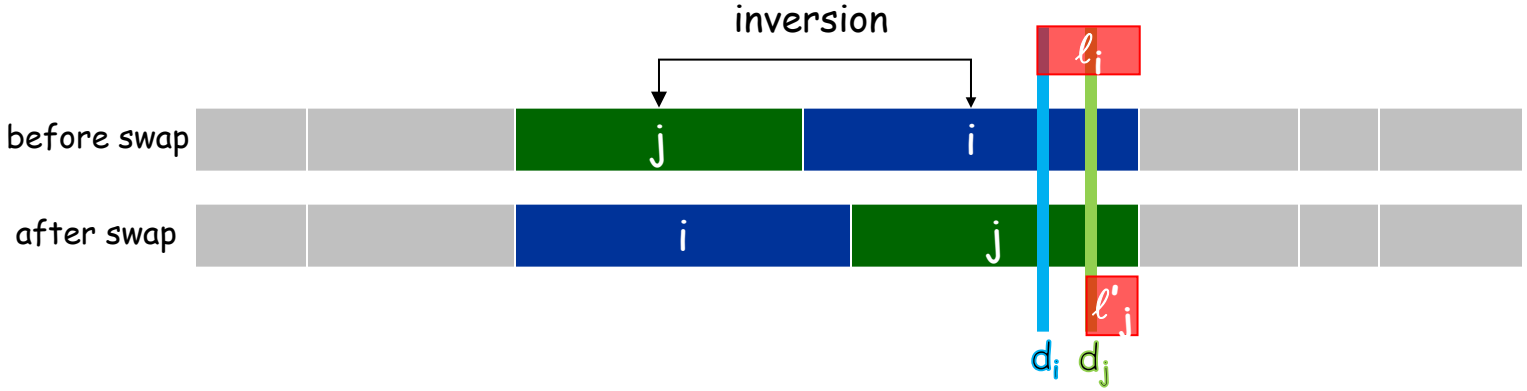
Minimizing Lateness: Idea of analysis

Def. An inversion in schedule S is a pair of jobs i and j such that: $d_i < d_j$ but j scheduled before i.



Observation 1. Greedy schedule has no inversions.

Observation 2. Swapping two adjacent, inverted jobs does not increase the max lateness

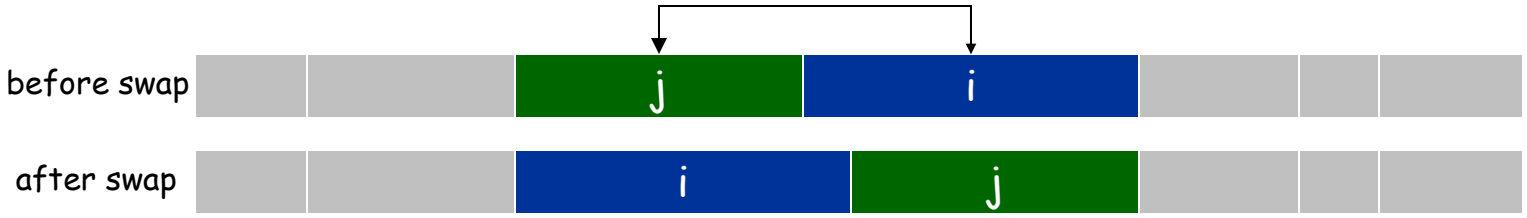


Minimizing Lateness: Idea of analysis

Swaps: Take OPT, keep swapping inverted jobs, brings it closer to greedy without increasing lateness

Repeating this we get a solution without any inversion (even non-adjacent, why?)

Reach solution where all items are in order of deadline => greedy solution (essentially, forget about ties)



Theorem. The Greedy algorithm in increasing order of due date is optimal

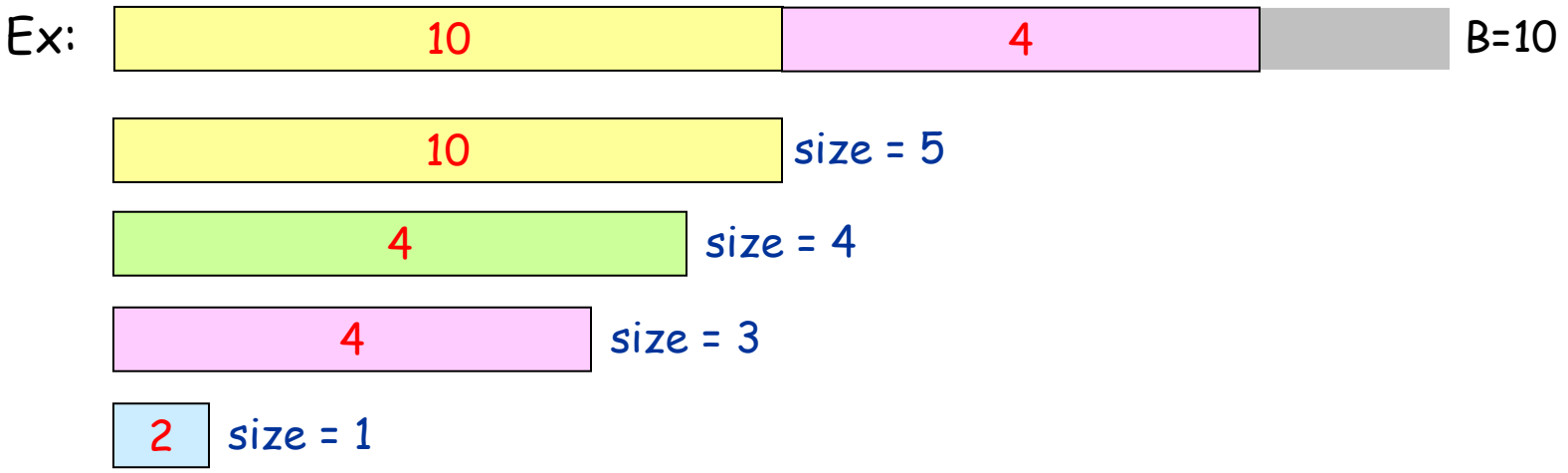
Fractional Knapsack Problem

Fractional Knapsack Problem

Knapsack Problem:

- You have a backpack of size B and several items
- Each item i has a size $size_i$ and value $value_i$
- Goal: Select items to put in the backpack that maximizes total value

Application: Project selection. (B is budget, size is cost, value is revenue)

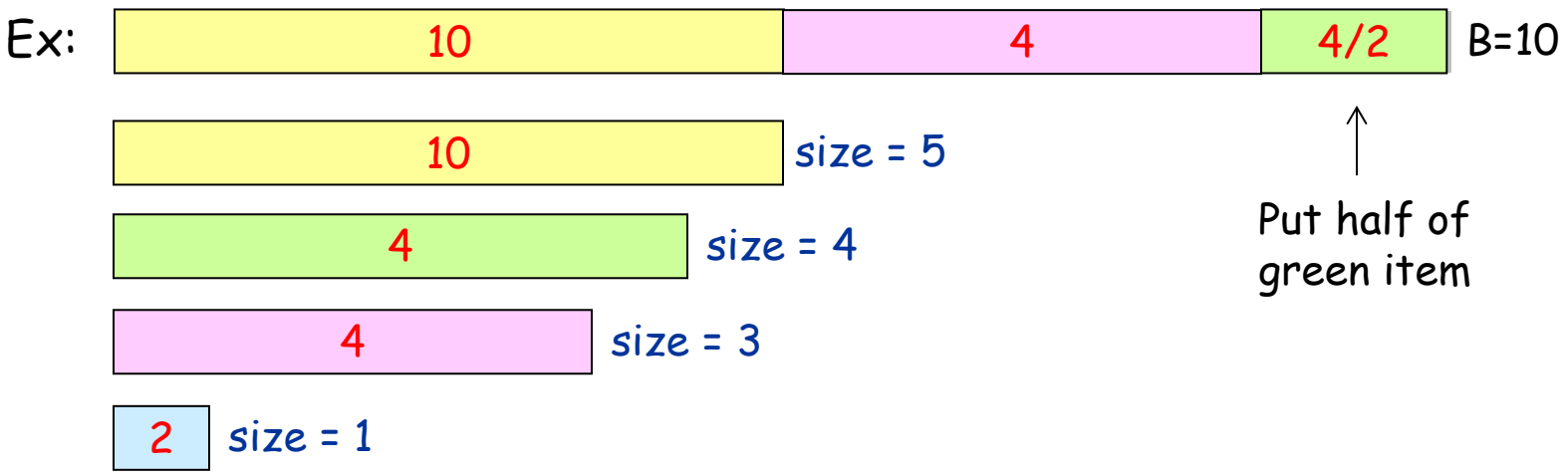


This is a harder problem, we will see in dynamic programming

Fractional Knapsack Problem

Fractional Knapsack Problem:

- You have a backpack of size B and several items
- Each item i has a size $size_i$ and value $value_i$
- Goal: Select items to put in the backpack that maximizes total value, can select just a fraction of an item (e.g. 10%)



Q: Can you see how to improve this solution?

A: Replace $\frac{1}{4}$ of green item by the blue item

Fractional Knapsack Problem

Task: Give a greedy algorithm for the Fractional Knapsack Problem
[Dantzig '57]

Greedy algorithm:

- sort items in decreasing order of value/size
- Scan items in this order, add as much of the item as you can to the backpack

Why it woks? Swap argument, based on what happened in previous example

If has item different from greedy => has lower value/size

- Swap a little bit of this item with some item in greedy => get closer to greedy solution and does not worsen the value

Repeat to get greedy solution

Theorem: Greedy by value/size is optimal.