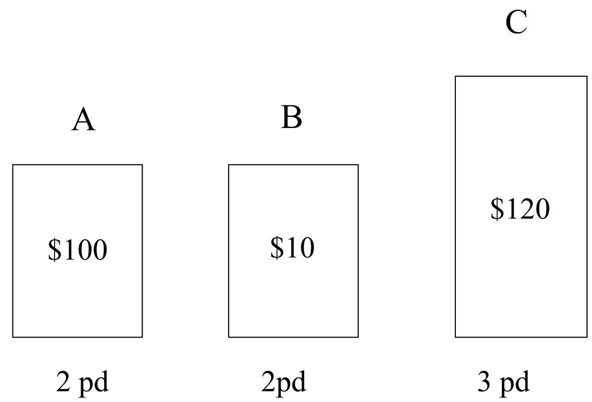
We already have seen an algorithmic technique in the previous chapter named divide and conquer

In this chapter, we introduce another such technique, called Greedy algorithms

A greedy algorithm, mostly designed for an optimization problem, starts with an empty solution and adds element to the subsolution based on a choice that looks best at this moment hoping that an optimal solution can be achieved in this way.

Most of the time, greedy solution fails to achieve optimality.

The Knapsack Problem...

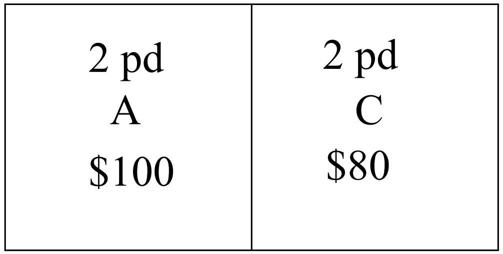


Capacity of knapsack: $K = 4$

Given items with weight and value and a knapsack with capacity K the objective is to put items inside the knapsack with maximum value.

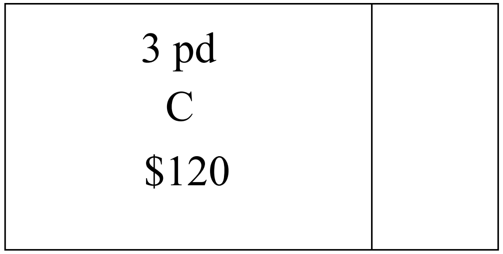
Fractional Knapsack Problem:
Can take a fraction of an item.

Solution:



0-1 Knapsack Problem:
Can only take or leave item. You can't take a fraction.

Solution:



The Fractional Knapsack Problem: Formal Definition

- Given K and a set of n items:

weight	w_1	w_2	$\ldots$	w_n
value	v_1	v_2	$\ldots$	v_n

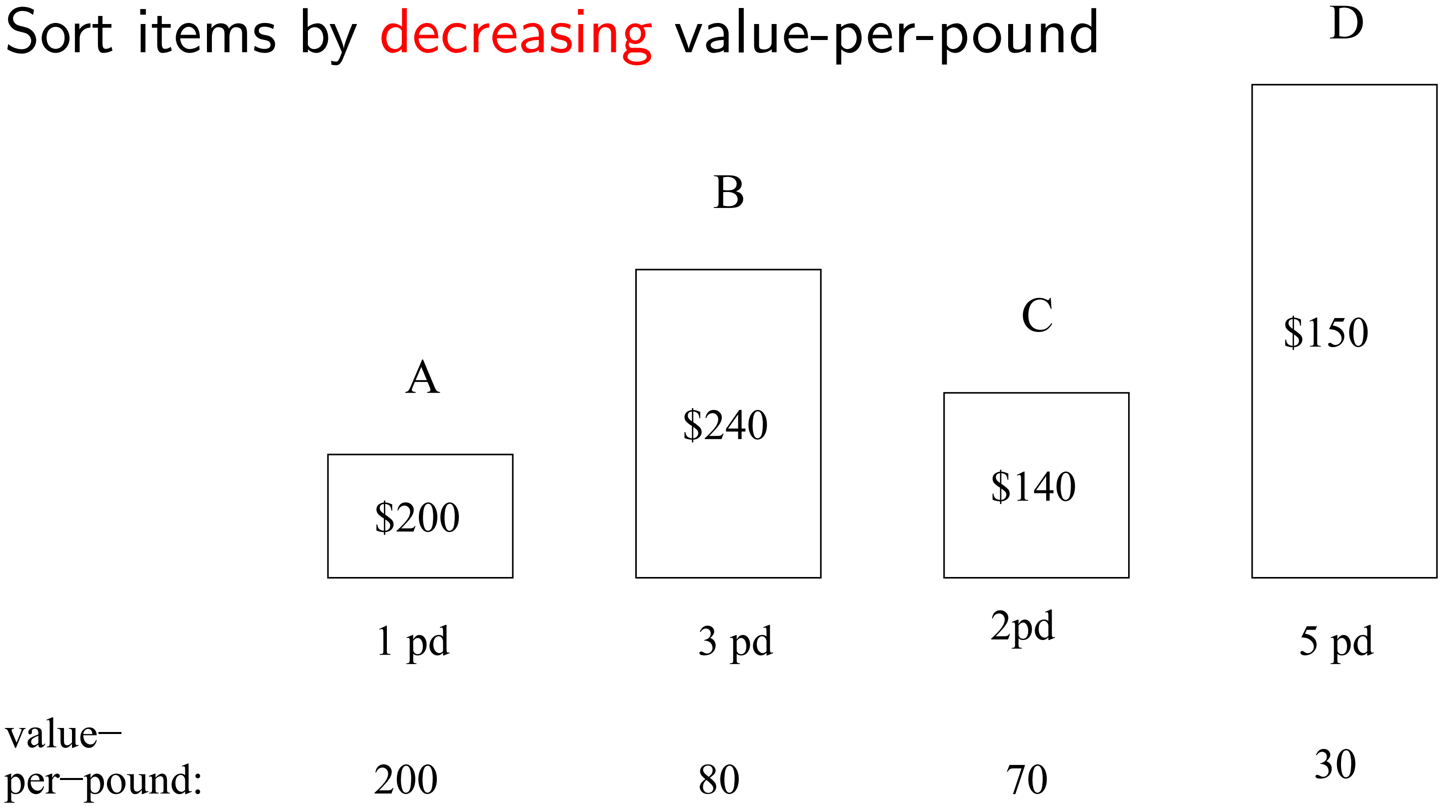
- Find: $0 \leq x_i \leq 1$, $i = 1, 2, \ldots, n$ such that

$$\sum_{i=1}^{n} x_i w_i \leq K$$

and the following is maximized:

$$\sum_{i=1}^{n} x_i v_i$$

Sort items by decreasing value-per-pound



If knapsack holds $K = 5$ pd, solution is:

1	pd	A
3	pd	B
1	pd	C

Greedy Solution for Fractional Knapsack

- Calculate the value-per-pound $\rho_i = \frac{v_i}{w_i}$ for $i = 1, 2, \ldots, n$.
- Sort the items by decreasing ρ_i.
  Let the sorted item sequence be $1, 2, \ldots, i, \ldots n$, and the corresponding value-per-pound and weight be ρ_i and w_i respectively.
- Let k be the current weight limit (Initially, $k = K$).
  In each iteration, we choose item i from the head of the unselected list.
  - If $k \geq w_i$, set $x_i = 1$ (we take item i), and reduce $k = k - w_i$, then consider the next unselected item.
  - If $k < w_i$, set $x_i = k/w_i$ (we take a fraction k/w_i of item i), Then the algorithm terminates.

Running time: $O(n \log n)$.

Greedy Solution for Fractional Knapsack

- Observe that the algorithm may take a fraction of an item. This can only be the last selected item.
- We claim that the total value for this set of items is the optimal value.

Correctness

Given a set of n items $\{1, 2, ..., n\}$.

- Assume items sorted by per-pound values: $\rho_1 \geq \rho_2 \geq ... \geq \rho_n$.

Let the greedy solution be $G = \langle x_1, x_2, ..., x_k \rangle$

- x_i indicates fraction of item i taken (all $x_i = 1$, except possibly for $i = k$).

Consider any optimal solution $O = \langle y_1, y_2, ..., y_n \rangle$

- y_i indicates fraction of item i taken in O $\big($for all i, $0 \leq y_i \leq 1\big)$.
- Knapsack must be full in both G and O: $\sum_{i=1}^{n} x_i w_i = \sum_{i=1}^{n} y_i w_i = K$.

Consider the first item i where the two selections differ.

- By definition, solution G takes a greater amount of item i than solution O (because the greedy solution always takes as much as it can). Let $x = x_i - y_i$.

Correctness...

Consider the following new solution O' constructed from O:

- For $j < i$, keep $y'_j = y_j$.
- Set $y'_i = x_i$.
- In O, remove items of total weight xw_i from items $i + 1$ to n, resetting the y'_j appropriately.

  This is always doable because $\sum_{j=i}^{n} x_j = \sum_{j=i}^{n} y_j$

- The total value of solution O' is greater than or equal to the total value of solution O (why?)
- Since O is largest possible solution and value of O' cannot be smaller than that of O, O and O' must be equal.
- Thus solution O' is also optimal.

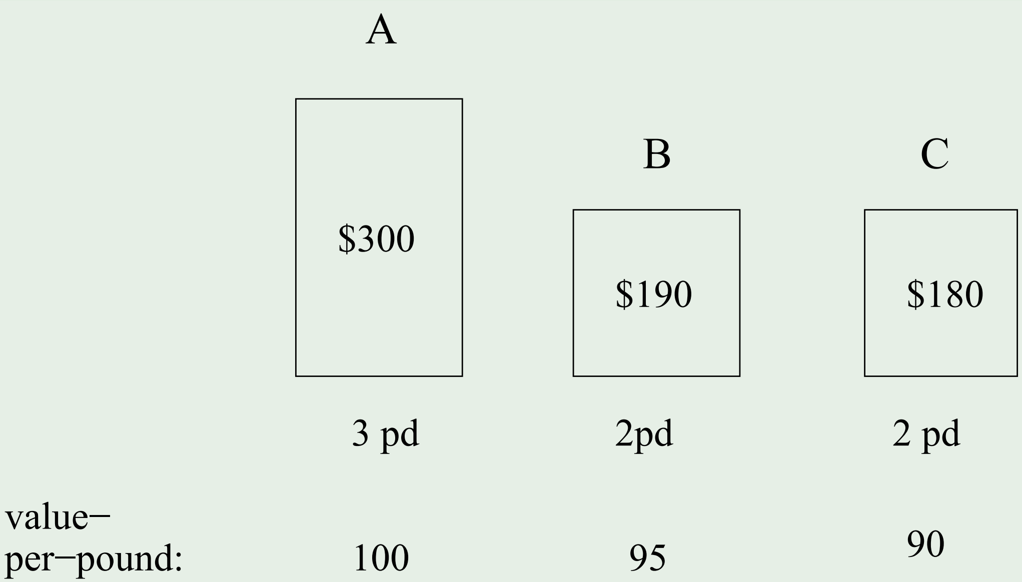
By repeating this process, we will eventually convert O into G, without changing the total value of the selection.

Therefore G is also optimal!

Greedy solution for 0-1 Knapsack Problem?

The 0-1 Knapsack Problem does not have a greedy solution!

Example



$K = 4$. Solution is item B + item C

Question

Suppose we tried to prove the greedy algorithm for 0-1 knapsack problem **does** construct an optimal solution. If we follow exactly the same argument as in the fractional knapsack problem where does the proof fail?

Greedy Scheduling

https://cs.pomona.edu/classes/cs140/

Scheduling (ignoring concurrency)

You have a shared resource

For example, a processor

You have many jobs that need to use the resource

Each job j has:

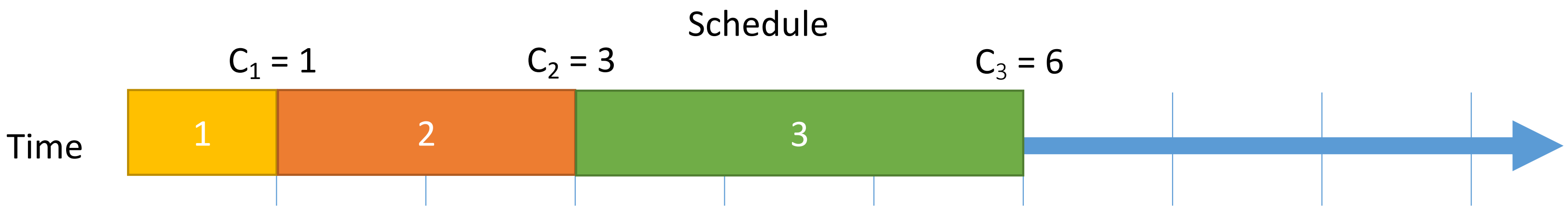
- A <u>Priority</u> P_j that stands for the job's importance
- A <u>Duration</u> D_j that stands for the length of time to run the job

In what sequence should we complete the jobs?

Scheduling (ignoring concurrency)

In what sequence should we complete the jobs?

- What is our criteria? What do we want to optimize?
- Let's start by looking at job j's **completion time** C_j
- Given three jobs: $D_1 = 1$, $D_2 = 2$, $D_3 = 3$
- What is the completion time for each if they are scheduled in order?



What is the completion time of Job 5?

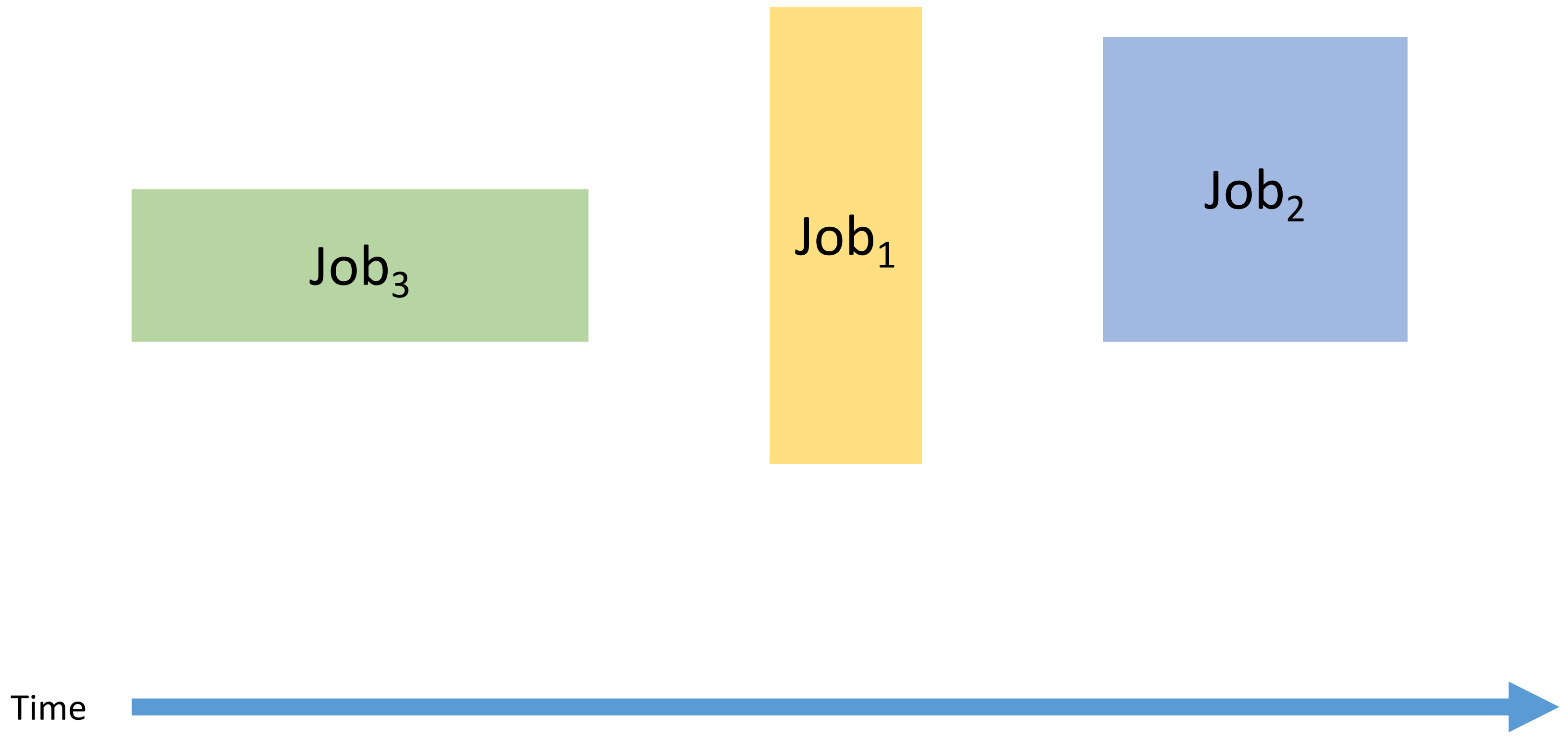
Scheduling

Optimization objective: minimize the weighted sum of completion times

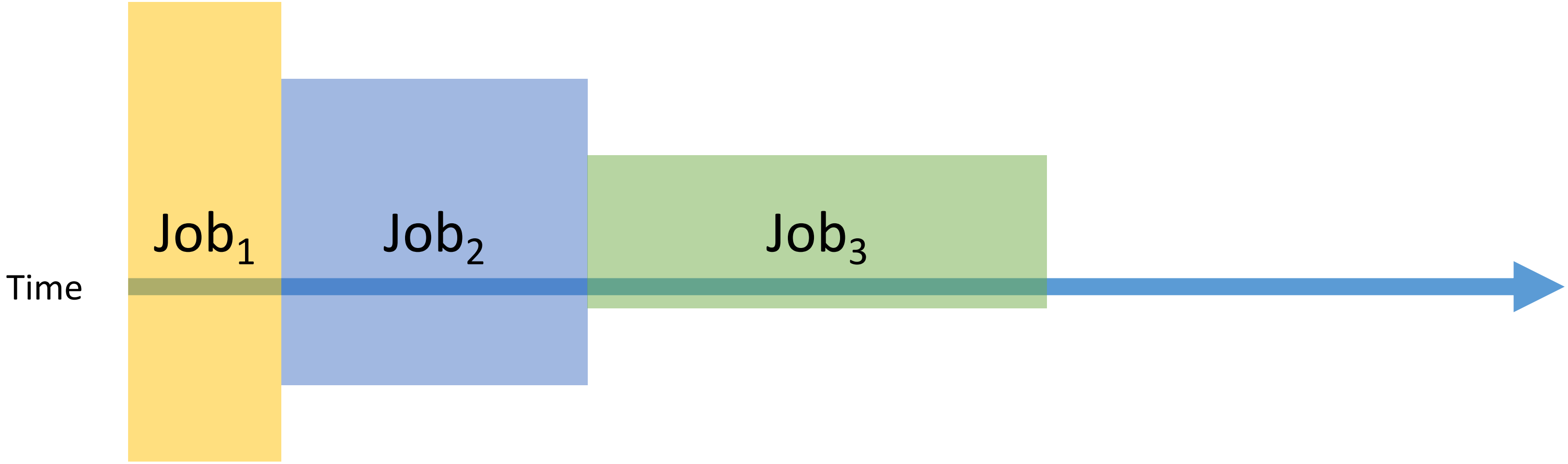
$$\mathrm{S}_{\mathrm{cost}} = \min[\sum_{j=1}^{n} P_j C_j]$$

What is the weighted sum of completion times if we schedule the following jobs in order?

Job	J_1	J_2	J_3
Duration	$D_1 = 1$	$D_2 = 2$	$D_3 = 3$
Priority	$P_1 = 3$	$P_2 = 2$	$P_3 = 1$


Job$_3$
Job$_1$
Job$_2$
Time

Exercise Question 1, 2, and 3



Scheduling

Calculate the weighted sum of completion times for the following jobs if they are scheduled in the order: 1, 2, 3.

Job	J_1	J_2	J_3
Duration	$D_1 = 1$	$D_2 = 2$	$D_3 = 3$
Priority	$P_1 = 3$	$P_2 = 2$	$P_3 = 1$
Completion			
Weight			

Weighted sum of completion times: ?

Greedy Scheduling

Our process for creating a greedy scheduling algorithm

1. Look at some special cases for our problem
2. Describe some possible greedy criteria
3. Compare our greedy criteria
4. Select the “best” one
5. Prove correctness if possible

Greedy Scheduling

Goal: devise a greedy algorithm to minimize the weighted sum of completion times

Why are we approaching this problem with a greedy algorithm?

- It’s a pretty easy way to start.
- Compare the approach we go through in these slides with a Divide and Conquer approach

1. What are some special cases to consider?

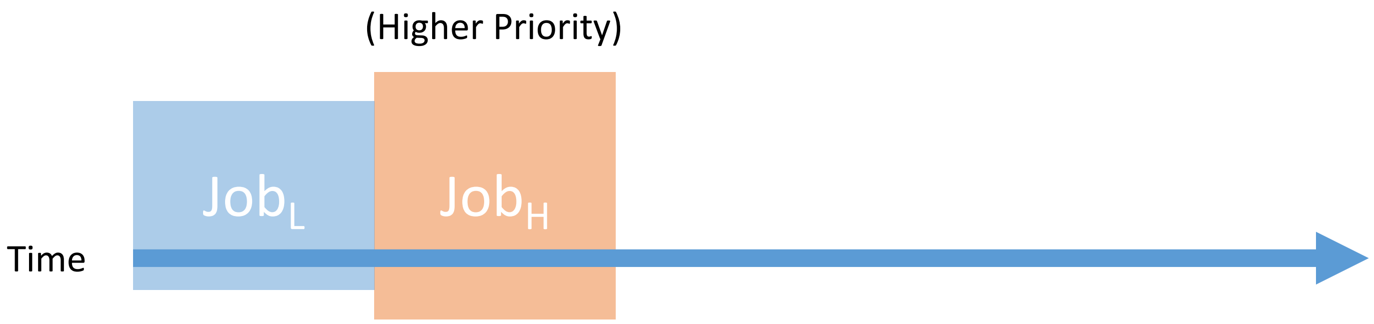
Consider <u>two</u> jobs with equal durations (D)

- These jobs have different priorities (P_H and P_L)
- **Do we schedule the lower or higher priority job first?**

1. What are some special cases to consider?

Consider <u>two</u> jobs with equal durations (D)

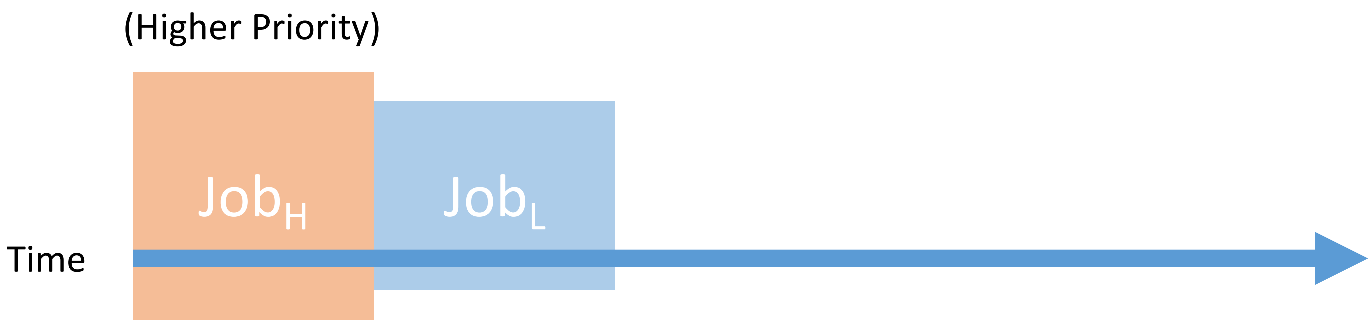
- These jobs have different priorities (P_H and P_L)
- **Do we schedule the lower or higher priority job first?**



1. What are some special cases to consider?

Consider <u>two</u> jobs with equal durations (D)

- These jobs have different priorities (P_H and P_L)
- **Do we schedule the lower or higher priority job first?**



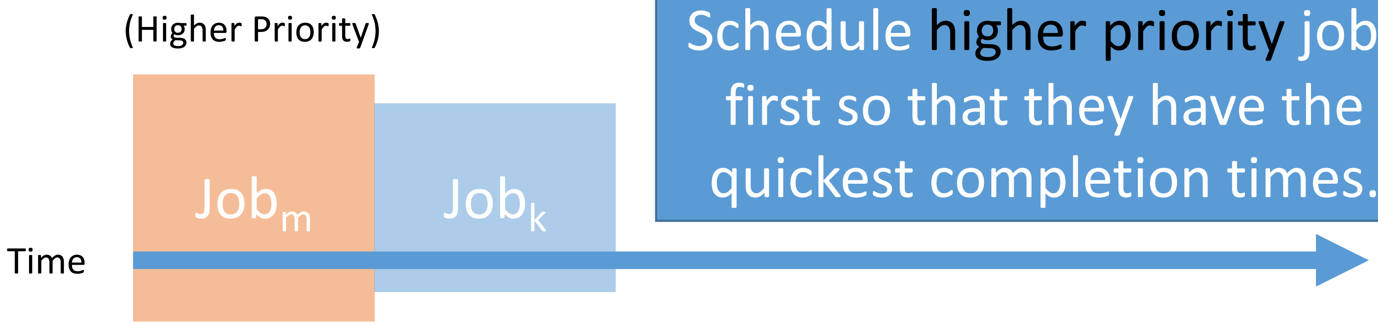
Schedule with Lower Priority First

Schedule with Higher Priority First

1. What are some special cases to consider?

Consider <u>two</u> jobs with equal durations (D)

- These jobs have different priorities (P_H and P_L)
- **Do we schedule the lower or higher priority job first?**

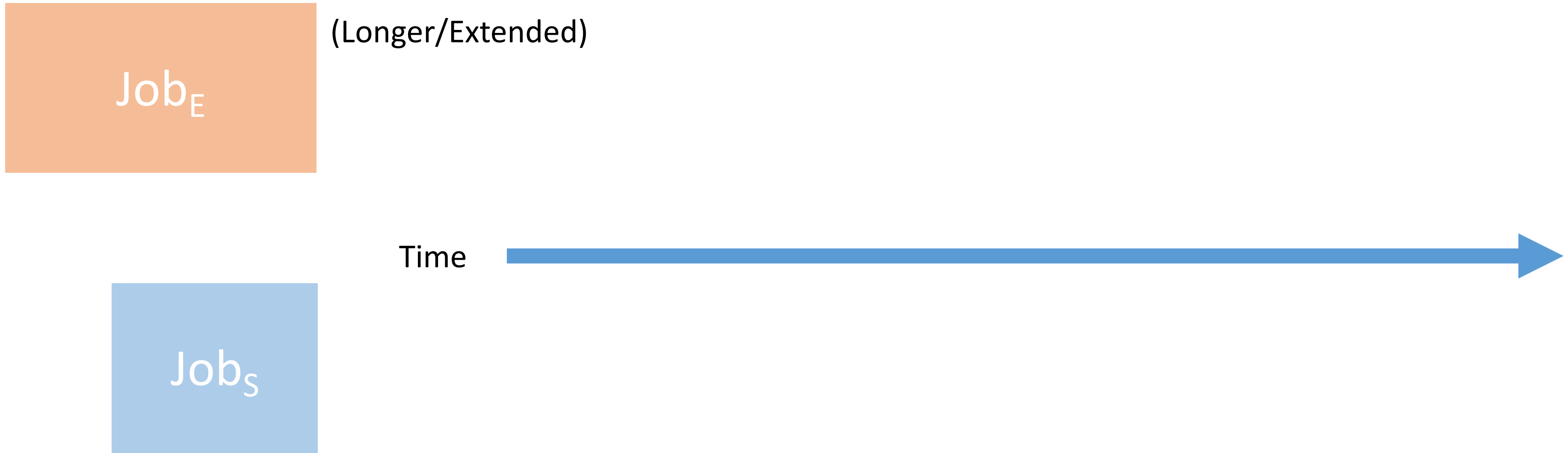


Schedule higher priority jobs first so that they have the quickest completion times.

1. What are some special cases to consider?

Consider two jobs with equal priorities (P)

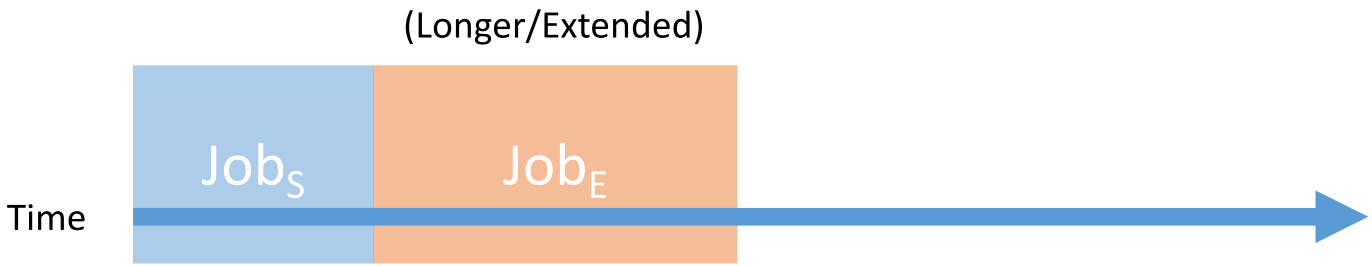
- These jobs have different durations (D_E and D_S)
- **Do we schedule the shorter or longer (Extended) job first?**



1. What are some special cases to consider?

Consider two jobs with equal priorities (P)

- These jobs have different durations (D_E and D_S)
- **Do we schedule the shorter or longer (Extended) job first?**



1. What are some special cases to consider?

Consider <u>two</u> jobs with equal priorities (P)

- These jobs have different durations (D_E and D_S)
- **Do we schedule the shorter or longer (Extended) job first?**

Schedule with Shorter Job First

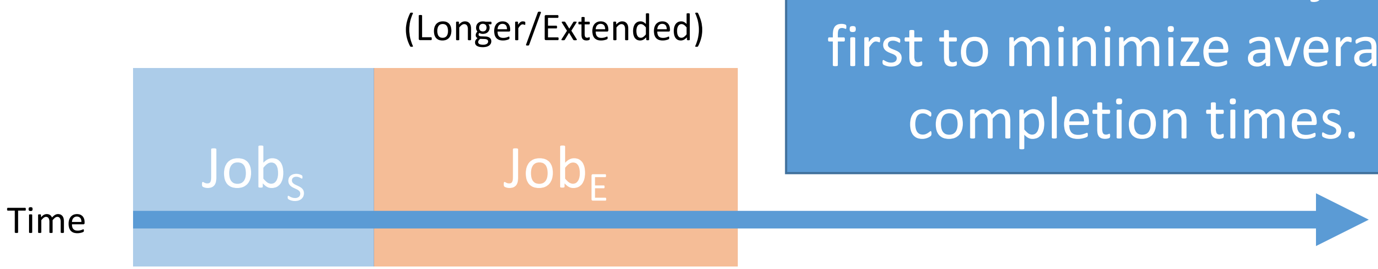
Schedule with Longer Job First

1. What are some special cases to consider?

Consider <u>two</u> jobs with equal priorities (P)

- These jobs have different durations (D_E and D_S)
- **Do we schedule the shorter or longer (Extended) job first?**

Schedule shortest jobs first to minimize average completion times.



2. Describe some possible greedy criteria

What do we do when in the more general case:

1. Schedule highest priority first
2. Schedule shortest duration first

$P_i > P_j$ *and* $D_i > D_j$ (job i has higher priority and longer duration)

What are some simple scoring functions that *aggregate* our criteria?

We want a function for which jobs with a bigger score are scheduled first:
- Score increases for higher priorities
- Score increases for shorter times

1. Greedy Criterion 1: $P_i - D_i$ (take the difference)

2. Greedy Criterion 2: ${P_i}/{D_i}$ (take the ratio)

3. Compare our greedy criteria

- Jobs will be ordered from biggest to smallest value

Job with same duration	Difference Metric ($P_i - D_i$)	Ratio Metric (P_i / D_i)
Job 1: P=2, D=1		
Job 2: P=5, D=1		

Which job should be scheduled first?

3. Compare our greedy criteria

- Jobs will be ordered from biggest to smallest value

	Job with same duration	Difference Metric $(P_i - D_i)$	Ratio Metric (P_i/D_i)
	Job 1: P=2, D=1	1	2
Highest priority	Job 2: P=5, D=1	4	5
	Total weighted sum		

3. Compare our greedy criteria

- Jobs will be ordered from biggest to smallest value

Job with same duration	Difference Metric $(P_i - D_i)$	Ratio Metric (P_i / D_i)
Job 1: P=2, D=1	1	2
Job 2: P=5, D=1	4	5
Total weighted sum	5*1 + 2*2 = 9	5*1 + 2*2 = 9

Highest priority

Same Result

Job with same priority	Difference Metric $(P_i - D_i)$	Ratio Metric (P_i / D_i)
Job 1: P=1, D=3		
Job 2: P=1, D=4		
Total weighted sum		

Which job should be scheduled first?

3. Compare our greedy criteria

- Jobs will be ordered from biggest to smallest value

Highest priority

Job with same duration	Difference Metric ($P_i - D_i$)	Ratio Metric (P_i / D_i)
Job 1: P=2, D=1	1	2
Job 2: P=5, D=1	4	5
Total weighted sum	5*1 + 2*2 = 9	5*1 + 2*2 = 9

Same Result

Shortest time

Job with same priority	Difference Metric ($P_i - D_i$)	Ratio Metric (P_i / D_i)
Job 1: P=1, D=3	-2	1/3
Job 2: P=1, D=4	-3	1/4
Total weighted sum		

Which job should be scheduled first?

3. Compare our greedy criteria

- Jobs will be ordered from biggest to smallest value

Job with same duration	Difference Metric $(P_i - D_i)$	Ratio Metric (P_i / D_i)
Job 1: P=2, D=1	1	2
Job 2: P=5, D=1	4	5
Total weighted sum	5*1 + 2*2 = 9	5*1 + 2*2 = 9

Highest priority

Same Result

Job with same priority	Difference Metric $(P_i - D_i)$	Ratio Metric (P_i / D_i)
Job 1: P=1, D=3	-2	1/3
Job 2: P=1, D=4	-3	1/4
Total weighted sum	1*3 + 1*7 = 10	1*3 + 1*7 = 10

Shortest time

Same Result

Which job should be scheduled first?

3. Compare our greedy criteria

- Let’s try to get them to disagree.
- Why does it matter if they don’t produce the same result?
- One scoring metric must be better than the other

- Apply the two greedy algorithms and calculate their weighted sum of completion times

Exercise Question 4

3. Compare our greedy criteria

- Jobs will be ordered from biggest to smallest metric value

Job	Difference Metric $(P_i - D_i)$	Ratio Metric (P_i / D_i)
Job 1: P=3, D=5		
Job 2: P=1, D=2		
Total weighted sum		

3. Compare our greedy criteria

- Jobs will be ordered from biggest to smallest metric value

Job	Difference Metric $(P_i - D_i)$	Ratio Metric (P_i / D_i)
Job 1: P=3, D=5	-2	3/5
Job 2: P=1, D=2	-1	1/2
Total weighted sum		

Which job goes first?

3. Compare our greedy criteria

- Jobs will be ordered from biggest to smallest metric value

Job	Difference Metric ($P_i - D_i$)	Ratio Metric (P_i/D_i)
Job 1: P=3, D=5	-2	3/5
Job 2: P=1, D=2	-1	1/2
Total weighted sum		

Which job goes first?

What is the priority sum?

4. Select the "best" one

- Jobs will be ordered from biggest to smallest metric value

Job	Difference Metric ($P_i - D_i$)	Ratio Metric (P_i / D_i)
Job 1: P=3, D=5	-2	3/5
Job 2: P=1, D=2	-1	1/2
Total weighted sum	1*2 + 3*7 = 23	3*5 + 1*7 = 22

Which job goes first?

What is the priority sum?

Which criteria is better?

5. Prove correctness if possible

Is criteria 2 optimal?

- We don’t know yet.

Claim: Criteria 2 is optimal for minimizing the weighted sum of completion times.

- <u>We’re going to prove this using an exchange argument!</u>

Exchange Arguments

- Consider your greedy solution, G
- Consider an alternative solution, A
  - A can be anything that is not G
  - Create A by changing G in some way
- Compare these solutions
  - Show that turning A into G makes A get better

Proof

- Assume that we have no ties (all P_i / D_i are distinct numbers)
- Fix an arbitrary input with n jobs
- Let's perform a proof using an exchange argument contradiction

Let G = the greedy schedule and A = the (alternative) optimal schedule

- Let's **assume** that A must be better than G (assume greedy is not optimal)
- To perform the contradiction, we must show that G is better than A, thus contradicting the purported optimality of A

Proof

Let G = the greedy schedule and A = the optimal schedule

- Assume that: $P_1/D_1 > P_2/D_2 > \ldots > P_n/D_n$
- We can just rename all jobs after we calculate their scores…
- Thus, G is just job 1 followed by job 2 etc. (1, 2, ..., n)

Job ID	Weight	Length	Ratio	Reorder
1	1	4	0.3	4
2	8	6	1.3	3
3	6	1	6.0	1
4	1	5	0.2	5
5	1	9	0.1	6
6	7	3	2.3	2

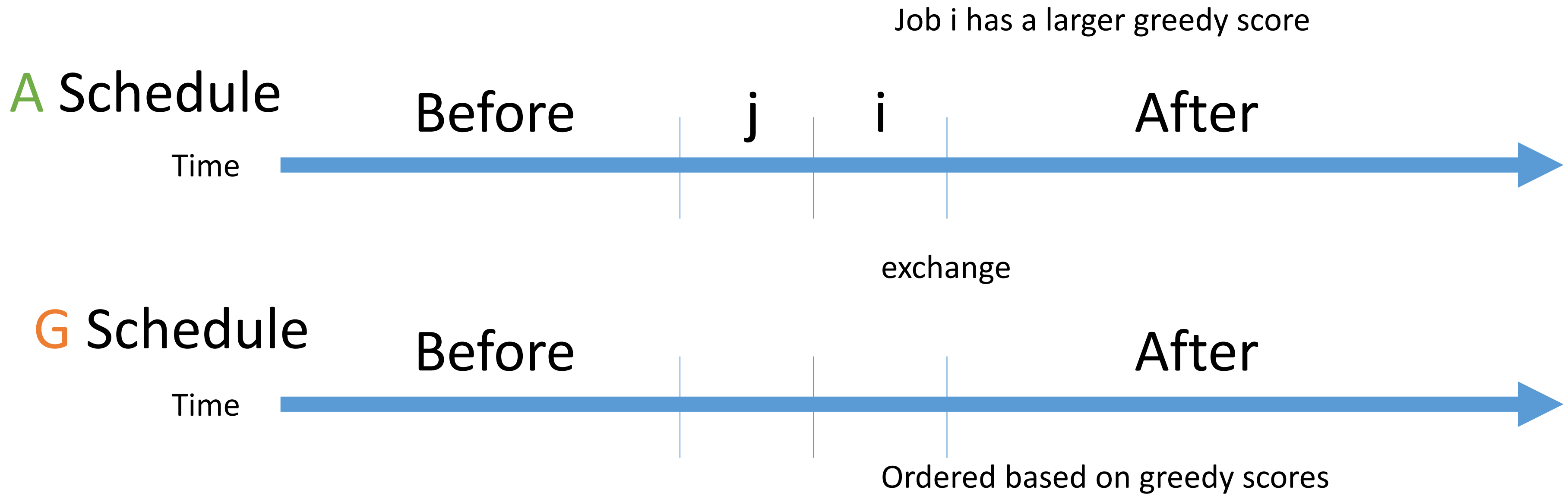
Proof

Let G = the greedy schedule and A = the optimal schedule

- Assume that: $P_1/D_1 > P_2/D_2 > \ldots > P_n/D_n$
- We can just rename all jobs after we calculate their scores…
- Thus, G is just job 1 followed by job 2 etc. (1, 2, ..., n)
- For A there must be at least two jobs that are "out of order"
  - Specifically, jobs i and j where i is scheduled after j, but $S_i > S_j$ (for example, Job_5 after Job_6)
- The greedy schedule is the only schedule where the jobs are in order

G vs A

(jobs i and j where i is scheduled after j, but P_i/D_i > P_j/D_j)

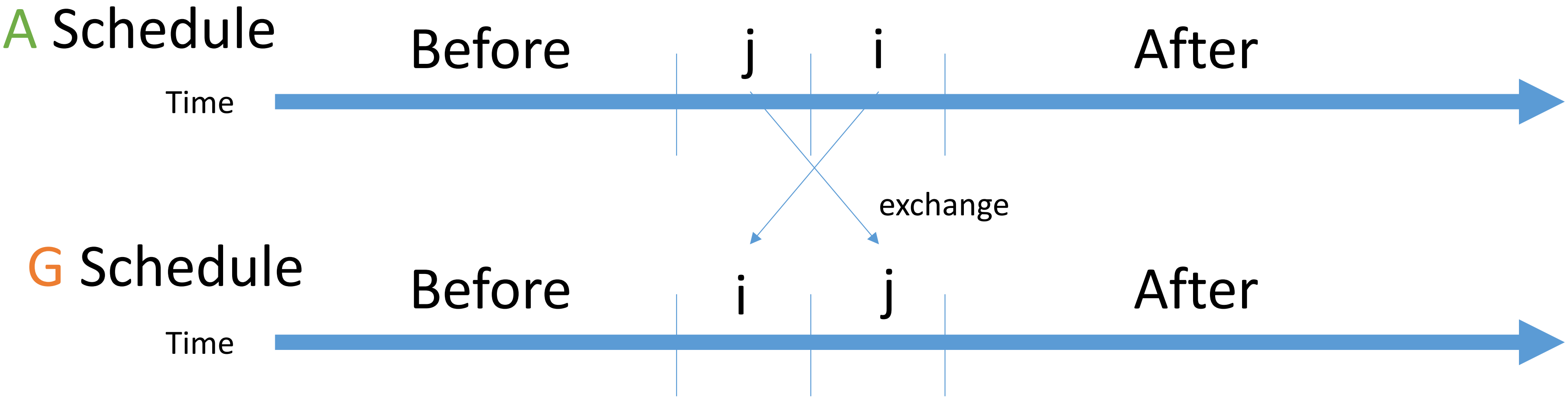


For example, i=7 and j=8

G vs A

(jobs i and j where i is scheduled after j, but $P_i/D_i > P_j/D_j$)

Job i has a larger greedy score



Ordered based on greedy scores

How does the exchange affect the completion time for:

1. Jobs other than i and j?
2. Job i
3. Job j

For example, i=7 and j=8

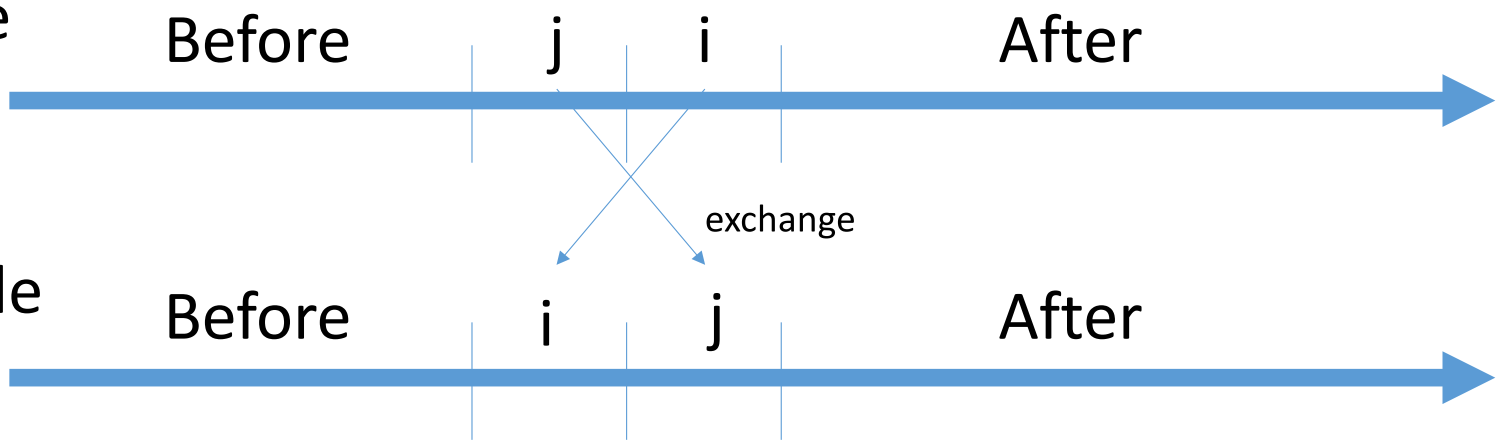
G vs A

(jobs i and j where i is scheduled after j, but $P_i/D_i > P_j/D_j$)

Job i has a larger greedy score

A Schedule

Time



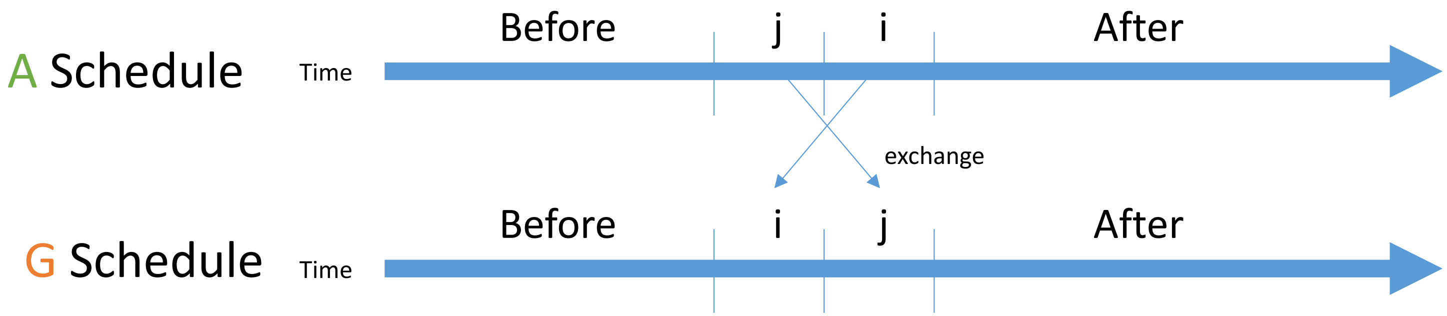
G Schedule

Time

Ordered based on greedy scores

What is the weighted sum of completion times for each schedule?

For example, i=7 and j=8

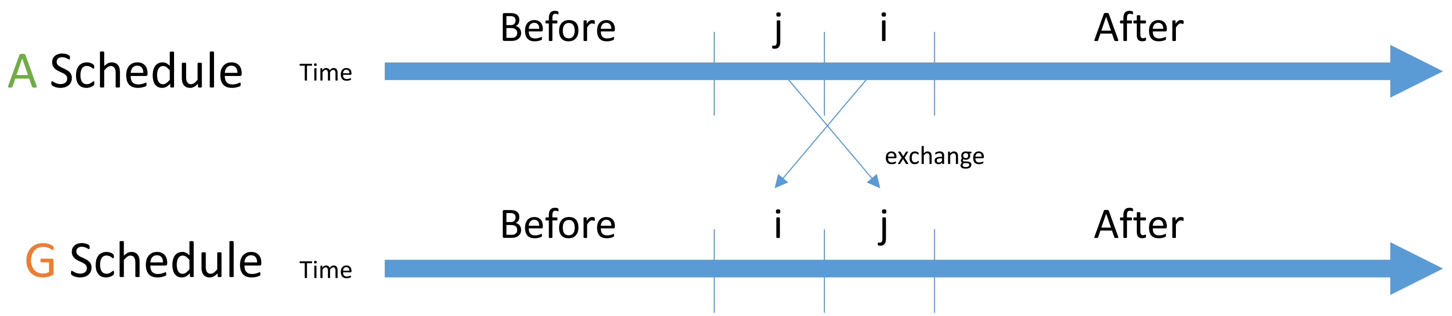



```
Cost(A) = Cost(Before) + Pj * (Tb + Dj) + Pi * (Tb + Dj + Di) + Cost(After)
Cost(G) = Cost(Before) + Pi * (Tb + Di) + Pj * (Tb + Di + Dj) + Cost(After)
```

```
Cost(A) < Cost(G)
```

Implied by optimality of A

```
Cost(Before) + Pj * (Tb + Dj) + Pi * (Tb + Dj + Di) + Cost(After)
    < Cost(Before) + Pi * (Tb + Di) + Pj * (Tb + Di + Dj) + Cost(After)
```



$$\mathrm{Cost}(A) = \mathrm{Cost}(\mathrm{Before}) + Pj * (Tb + Dj) + Pi * (Tb + Dj + Di) + \mathrm{Cost}(\mathrm{After})$$
$$\mathrm{Cost}(G) = \mathrm{Cost}(\mathrm{Before}) + Pi * (Tb + Di) + Pj * (Tb + Di + Dj) + \mathrm{Cost}(\mathrm{After})$$

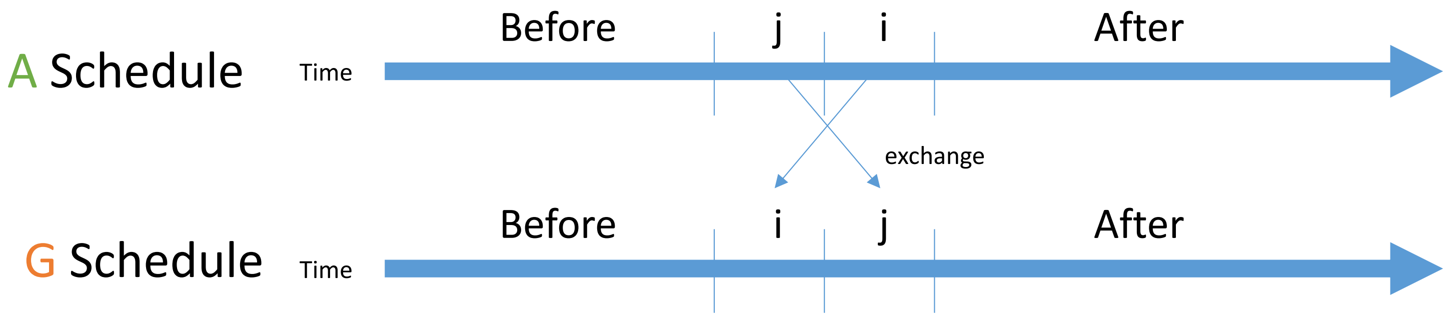
$$\mathrm{Cost}(A) < \mathrm{Cost}(G)$$

Implied by optimality of A

$$\mathrm{Cost}(\mathrm{Before}) + Pj * (Tb + Dj) + Pi * (Tb + Dj + Di) + \mathrm{Cost}(\mathrm{After})$$
$$< \mathrm{Cost}(\mathrm{Before}) + Pi * (Tb + Di) + Pj * (Tb + Di + Dj) + \mathrm{Cost}(\mathrm{After})$$

$$Pj * (Tb + Dj) + Pi * (Tb + Dj + Di)$$
$$< Pi * (Tb + Di) + Pj * (Tb + Di + Dj)$$

$$Pj*Tb + Pj*Dj + Pi*Tb + Pi*Dj + Pi*Di$$
$$< Pi*Tb + Pi*Di + Pj*Tb + Pj*Di + Pj*Dj$$



$$\mathrm{Cost}(A) = \mathrm{Cost}(Before) + Pj * (Tb + Dj) + Pi * (Tb + Dj + Di) + \mathrm{Cost}(After)$$
$$\mathrm{Cost}(G) = \mathrm{Cost}(Before) + Pi * (Tb + Di) + Pj * (Tb + Di + Dj) + \mathrm{Cost}(After)$$

$$\mathrm{Cost}(A) < \mathrm{Cost}(G)$$

Implied by optimality of A

$$\mathrm{Cost}(Before) + Pj * (Tb + Dj) + Pi * (Tb + Dj + Di) + \mathrm{Cost}(After)$$
$$< \mathrm{Cost}(Before) + Pi * (Tb + Di) + Pj * (Tb + Di + Dj) + \mathrm{Cost}(After)$$

$$Pj * (Tb + Dj) + Pi * (Tb + Dj + Di)$$
$$< Pi * (Tb + Di) + Pj * (Tb + Di + Dj)$$

$$Pj*Tb + Pj*Dj + Pi*Tb + Pi*Dj + Pi*Di$$
$$< Pi*Tb + Pi*Di + Pj*Tb + Pj*Di + Pj*Dj$$

$$Pi*Dj < Pj*Di$$

$$Pi/Di < Pj/Dj$$

Contradiction to how they were ordered by our greedy criteria

Summary of Greedy Scheduling

- Given n jobs, each with a priority and a duration
- Give each job a score based on their ratio of priority to duration
- Schedule jobs in decreasing order of their score
- This gives us an optimal schedule

- What do we do if we're given more jobs while these are running?
- Any issues with this scheme?
  - Some jobs might always be postponed.