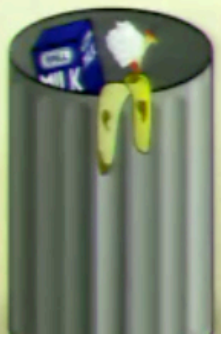
Greedy Algorithms

- The simplest class of algorithms are **greedy algorithms**.
  - They make a *locally optimal* (or *myopic*) choice at each step.
- Greedy algorithms have several nice properties:
  - *Fast.*
  - *Simple.*
  - *Easy to Code.*
- Unfortunately, they are also usually rubbish.



So for this topic the goal is to understand:

- *The basic greedy algorithmic techniques.*
- *On which problems these techniques work.*

Task Scheduling

- A firm receives **job orders** from n customers.
  - The firm can do only <u>one</u> task at a time.
  - The job of customer i will take the firm t_i units of *time* to complete.
  - Each customer i wants their job completed as *early as possible* (and will pay accordingly).
- Assume the firm processes the jobs in the order $\{1, 2, 3, \ldots, n\}$.
  - Then the **waiting time** of customer i will be: $w_\ell = \sum_{i=1}^{\ell} t_i$
- So to maximize profits the objective of the firm is to *minimize the sum of the waiting times*:

$$\sum_{\ell=1}^{n} w_\ell = \sum_{\ell=1}^{n} \sum_{i=1}^{\ell} t_i$$

Greedy Task Scheduling

- Here is a **greedy algorithm** that solves this problem optimally.

Greedy Task Scheduling Algorithm

Sort the jobs by length $t_1 \leq t_2 \leq \cdots \leq t_n$

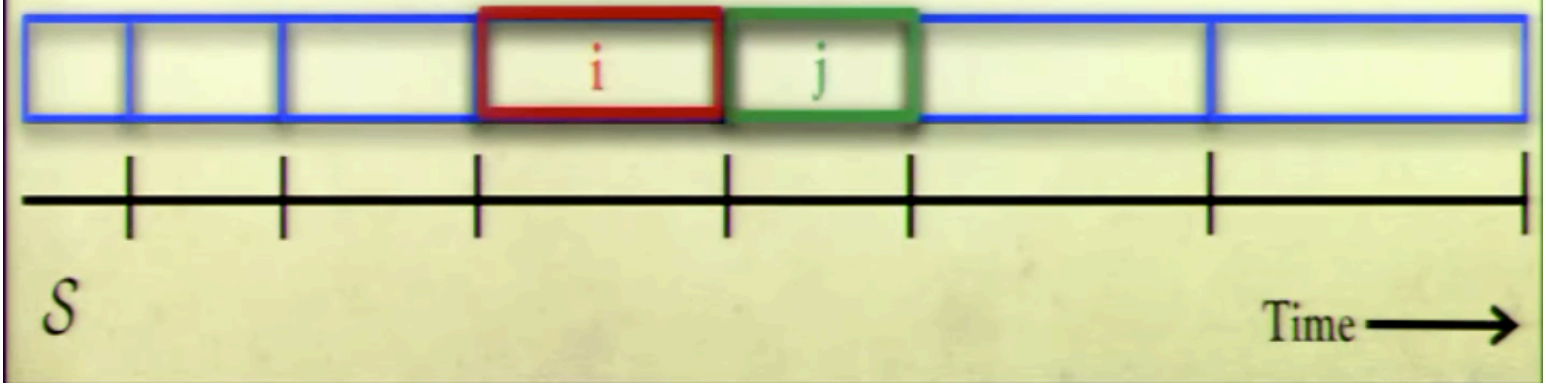
Schedule the jobs in the order $\{1, 2, 3, \ldots, n\}$

An Exchange Argument

Theorem. The greedy algorithm outputs an *optimal* schedule.

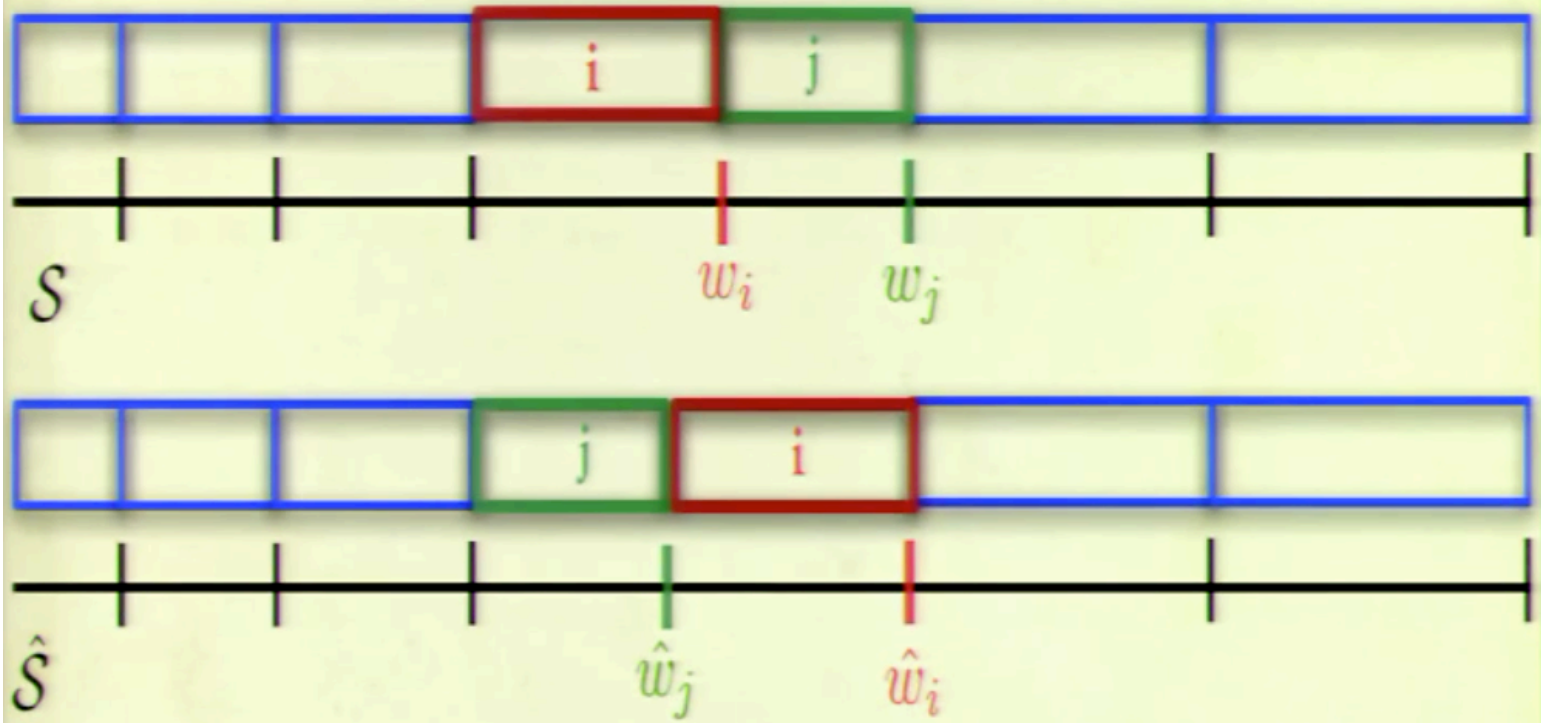
Proof.

- Let the greedy algorithm schedule in the order $\{1, 2, \ldots, n\}$
- Assume there is a better schedule $\mathcal{S}$
- Then there must be a pair of jobs i and j such that:
  1) Job i is scheduled immediately **before** job j by schedule $\mathcal{S}$
  2) Job i is **longer** than job j: $t_i > t_j$



Proof [cont.]

- But then **exchanging** the order of jobs i and j gives a better schedule $\hat{S}$.



- Observe the *waiting time* of every other job remains the **same**:

$$\hat{w}_k = w_k \quad \forall k \neq i, j$$

- But clearly: $\hat{w}_i + \hat{w}_j < w_i + w_j$

- This contradicts the assumption that S was an optimal schedule. □

The Running Time

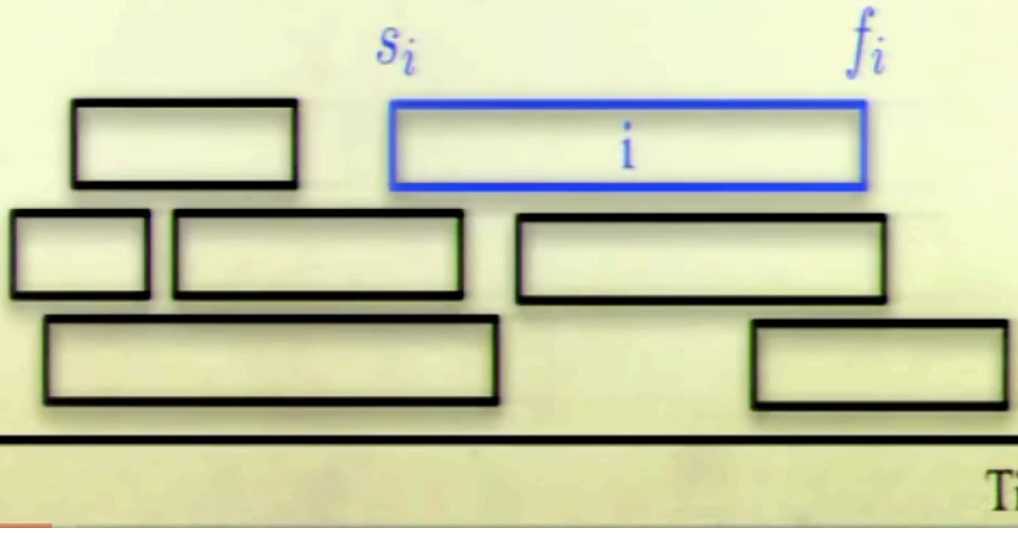
- How long does the greedy task scheduling algorithm take?
  - We need only sort n time lengths.

$$\Longrightarrow \quad \text{Runtime} = O(n \cdot \log n)$$

- So this algorithm is **efficient**.

Class Scheduling

- There is one classroom.
- There is a set $I = \{1, 2, \ldots, n\}$ of classes that request the room.
  - Class i has a start time s_i and a finish time f_i.
- The objective is to book as many classes into the room as possible.
  - However, we **cannot** book two classes that need to use the room at exactly the *same time*.
- This problem is often called the **Interval Selection Problem**, as we can plot the classes as intervals between their start and finish times.



Greedy Scheduling Algorithms

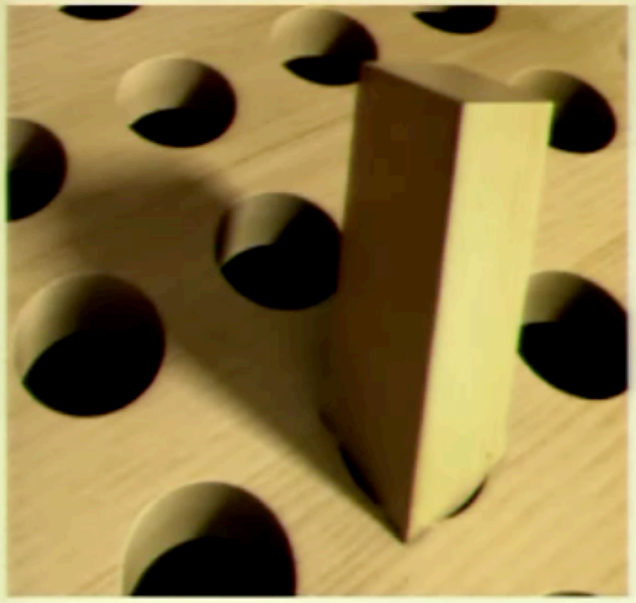
- But which greedy algorithm? *There are many possibilities:*

First-Start. Select the class that starts earliest, and iterate on the remaining classes that do not conflict with this selection.

Shortest-Duration. Select the class of shortest duration, and iterate on the remaining classes that do not conflict with this selection.

Minimum-Conflict. Select the class that conficts with the fewest number of classes, and then iterate...

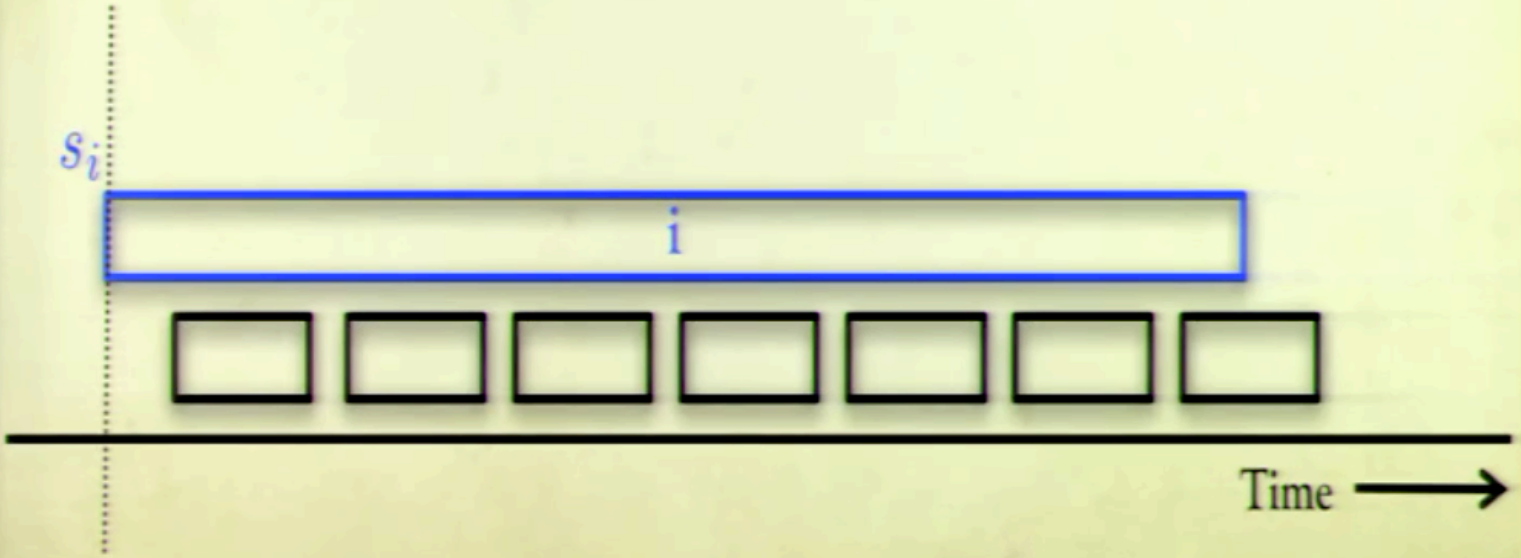
Last-Start. Select the class that starts last, and then iterate...

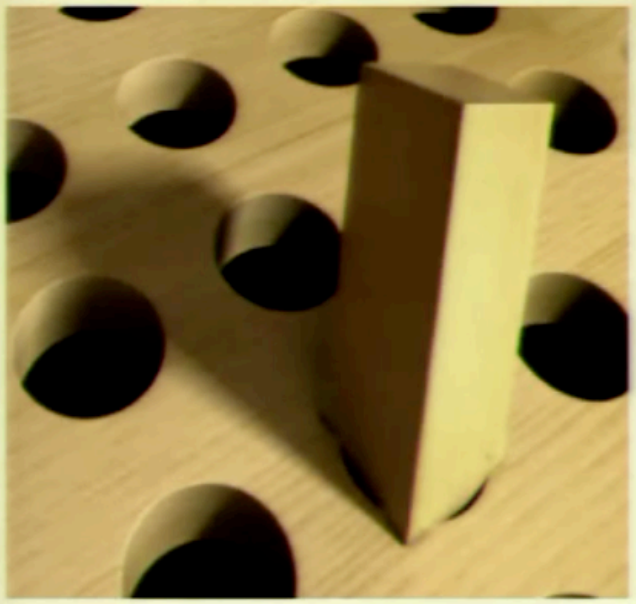


First-Start

- Do these any of these algorithms even work?

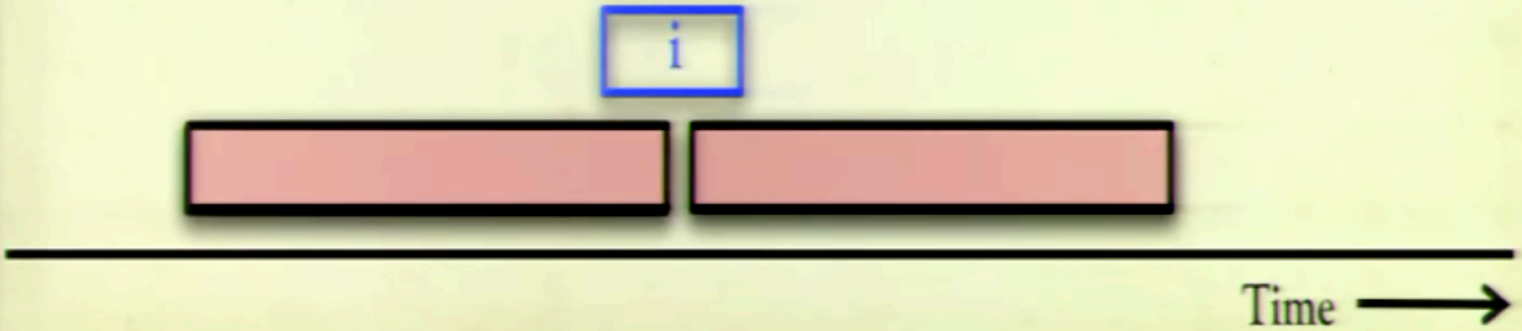
First-Start. Select the class that starts earliest, and iterate on the remaining classes that do not conflict with this selection.

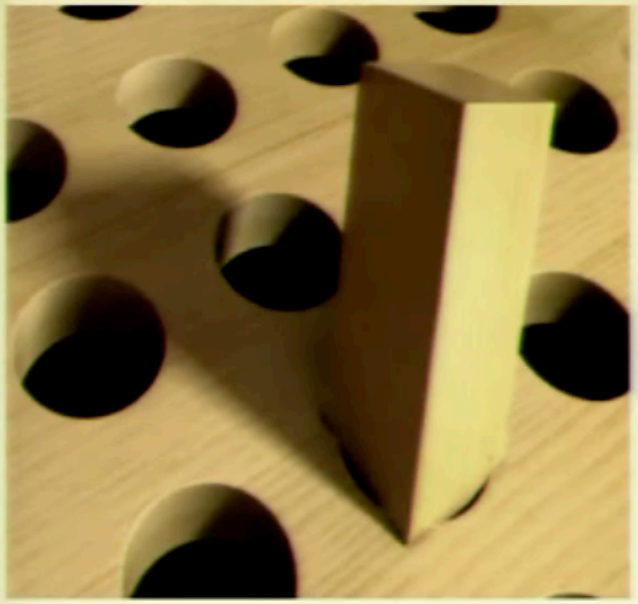




Shortest-Duration

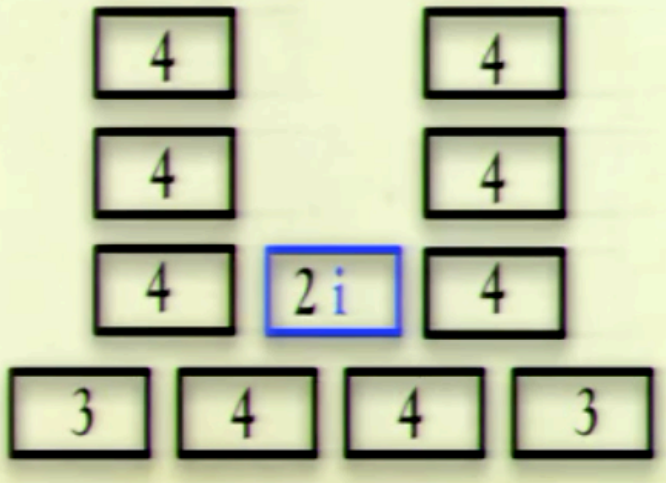
Shortest-Duration. Select the class of shortest duration, and iterate on the remaining classes that do not conflict with this selection.





Minimum-Conflict

Minimum-Conflict. Select the class that conficts with the fewest number of classes, and iterate on the classes that do not conflict with this selection.



Last-Start

Last-Start. Select the class that starts last, and iterate on the classes that do not conflict with this selection.

- It is not easy to find a **bad example** for this greedy algorithm.
- Because there are no bad examples! It ouputs an *optimal* schedule.
- This algorithm is *symmetric* to the following greedy algorithm:

First-Finish. Select the class that finishes first, and iterate on the classes that do not conflict with this selection.

- So let's prove the First-Finish greedy algorithm does indeed work...

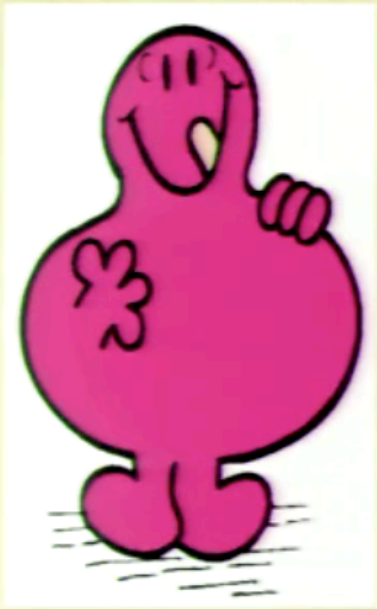
First-Finish Greedy Scheduling

`FirstFinish(I)`

Let Class 1 be the class with the earliest finish time.

Let X be the set of classes that clash with class 1.

Output $\{1\} \cup$ `FirstFinish(I \ X)`



First-Finish Greedy Scheduling (Revisted)

First-Finish Scheduling Algorithm

Sort the classes by finish times $f_1 \leq f_2 \leq \cdots \leq f_n$

Initialize the set of selected classes: $S = \emptyset$

Initialize the set of undecided class requests: $I = \{1, 2, \ldots, n\}$

While $I \neq \emptyset$

Let i be the lowest index class in I

Set $S \leftarrow S \cup \{i\}$ and $I \leftarrow I \setminus \{i\}$

For each class $j \in I$

If $s_j < f_i$ **then** set $I \leftarrow I \setminus \{j\}$

Output S

First-Finish Scheduling Algorithm

Sort the classes by finish times $f_1 \leq f_2 \leq \cdots \leq f_n$

Initialize the set of selected classes: $S = \emptyset$

Initialize the set of open class requests: $I = \{1, 2, \ldots, n\}$

While $I \neq \emptyset$

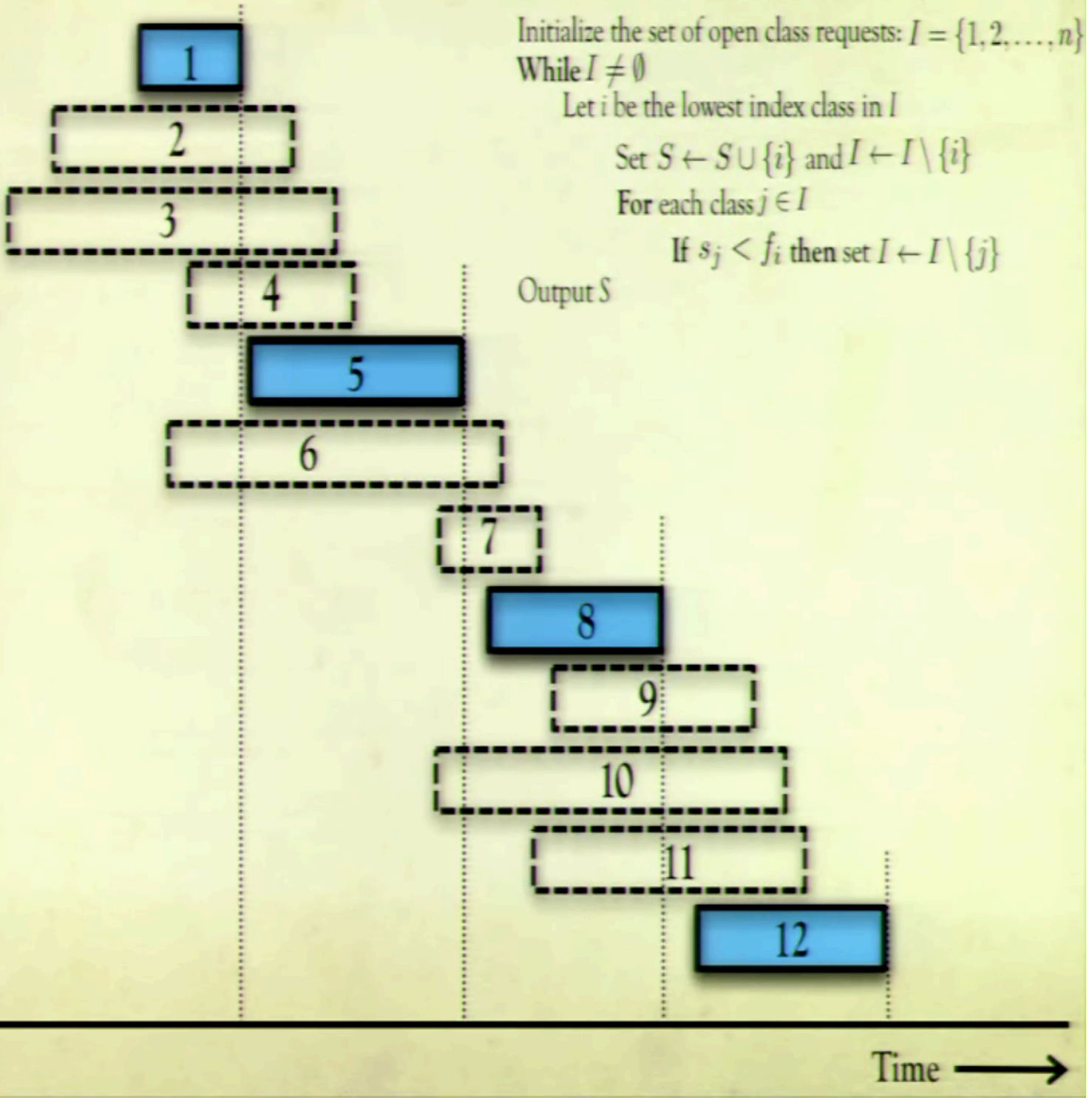
Let i be the lowest index class in I

Set $S \leftarrow S \cup \{i\}$ and $I \leftarrow I \setminus \{i\}$

For each class $j \in I$

If $s_j < f_i$ then set $I \leftarrow I \setminus \{j\}$

Output S

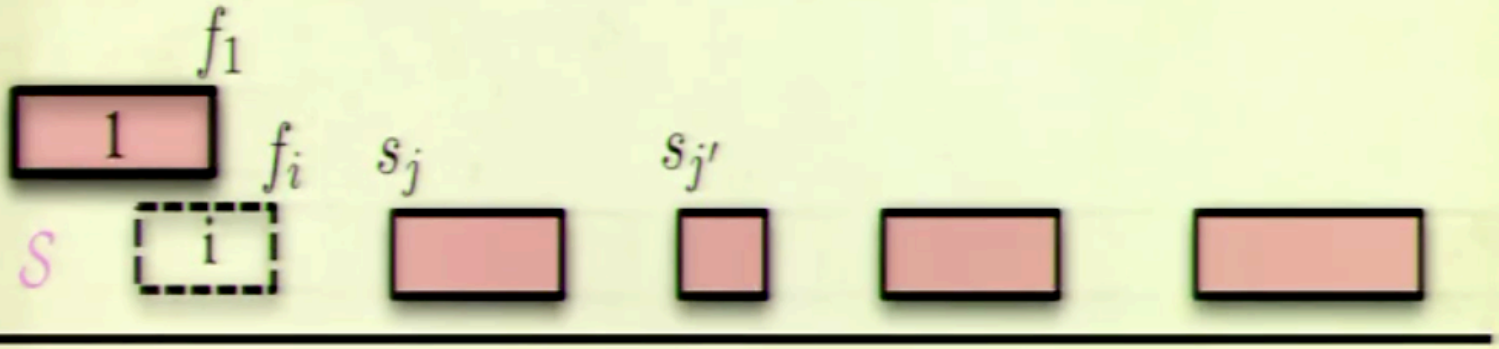


Another Exchange Argument

Lemma. There is an optimal schedule that selects Class 1.

Proof.

- Recall the classes are indexed such that $f_1 \leq f_2 \leq \cdots \leq f_n$
- Take an *optimal schedule* $\mathcal{S}$ and assume $1 \notin \mathcal{S}$
- Let i be the lowest index class selected in $\mathcal{S}$
- We claim $\mathcal{S} \setminus \{i\} \cup \{1\}$ is a feasible allocation of **maximum size.**



- This follows as $f_i \leq s_j$ for any $j \in \mathcal{S} \setminus \{i\}$.
- But $f_1 \leq f_i \leq s_j$ so Class 1 does not conflict with any Class in $\mathcal{S} \setminus \{i\}$

$\implies \mathcal{S} \setminus \{i\} \cup \{1\}$ is feasible with cardinality equal to $|\mathcal{S}|$. □

`FirstFinish(I)`
Let Class *1* be the class with the earliest finish time.
Let *X* be the set of classes that clash with class *1*.
Output $\{1\} \cup$ `FirstFinish(I \ X)`

The Optimality of First-Finish

Theorem. The first-finish algorithm outputs an *optimal* schedule.

Proof.

- By *induction* on the cardinality of the optimal solution $|\texttt{opt}(I)|$

Base Case:

- Let $|\texttt{opt}(I)| = 1$
- Then First-Finish outputs $\{1\}$ and this is an optimal solution.

Induction Hypothesis:

- If $|\texttt{opt}(I)| = k$ then First-Finish outputs an optimal solution.

`FirstFinish(I)`
Let Class *1* be the class with the earliest finish time.
Let *X* be the set of classes that clash with class *1*.
Output $\{1\} \cup$ `FirstFinish(I \ X)`

Proof [cont.]

Induction Step:

- Let $|\texttt{opt}(I)| = k+1$
- First-Finish outputs $\{1\} \cup \texttt{FirstFinish(I \ K)}$
- But, by the lemma, $\{1\} \in \mathcal{S}^*$ for some optimal solution $\mathcal{S}^*$

  $\Longrightarrow$ Thus $\mathcal{S}^* \setminus \{1\}$ is an optimal solution for the sub-problem $I \setminus X$

  $\Longrightarrow |\texttt{opt}(I \setminus X)| = k$
- By the induction hypothesis, First-Finish gives an optimal solution for the sub-problem $I \setminus X$

  $\Longrightarrow |\texttt{FirstFinish(I \ X)}| = k$

  $\Longrightarrow |\{1\} \cup \texttt{FirstFinish(I \ X)}| = k+1$
- Thus First-Finish outputs an optimal solution. □

`FirstFinish(I)`
Let Class *1* be the class with the earliest finish time.
Let X be the set of classes that clash with class *1*.
Output $\{1\} \cup$ `FirstFinish(I \ X)`

The Running Time

- How long does the first-finish scheduling algorithm take?
  - There are at most n iterations.
  - It takes $O(n)$ time to find class that finishes earliest in each iteration.

  $\Longrightarrow$ Runtime $= O(n^2)$

- But this is a very *crude* analysis.
- With a more subtle implementation and analysis we can obtain a running time of $O(n \cdot \log n)$.

Practical Aspects

- Scheduling problem are *ubiquitous* in practice.
- However, the scheduling problems seen today should be viewed as toy examples used to introduce the topic of greedy algorithms.
- For example, for more realistic applications of interval scheduling we would want to allow for:
  - *Multiple Rooms.*
  - *Weighted Intervals.*
  - *Online Booking Requests, etc.*