Lecture 17

CSE 331

Oct 5, 2018

Homework 5

Homework 5

Due by **11:59pm, Thursday, October 11, 2018**.

Make sure you follow all the homework policies.

All submissions should be done via Autolab.

Sample Problem

The Problem

Extend the topological ordering algorithm we saw in class so that, given an input directed graph G, it outputs one of two things: (a) a topological ordering, thus establishing that G is a DAG, or (b) a cycle in G, thus establishing that G is not a DAG.

The running time of your algorithm should be $O(m + n)$ for a directed graph with n nodes and m edges.

Click here for the Solution

Solutions to HW 4

End of the lecture

Quiz 1 on Monday

note

stop following

125 views

Quiz 1 on Monday, Oct 8

The first quiz will be from **8-8:10am in class** on **Monday, October 8**. We will have a 5 mins break after the quiz and the lecture will start at 8:15am.

We will hand out the quiz paper at 7:55am but you will **NOT** be allowed to open the quiz to see the actual questions till 8am. However, you can use those 5 minutes to go over the instructions and get yourself in the zone.

There will be two T/F with justification questions (like those in the sample mid term 1: @458.)
#pin

quiz1

edit · good note | 0

Updated 3 days ago by Mark Armstrong and Atri Rudra

Done with mid-term material

note

stop following

83 views

Linear time topological ordering

As I mentioned in the lecture today, one can actually implement TopOrd in $O(m+n)$ time instead of the $O(n^2)$ analysis that we did in the lecture today. The details are in the book or you can watch this video from last year.

BTW this is where the mid-term material ends: so all of Chapters 1, 2 and 3 in the textbook (except Chapter 1.2).
#pin

lectures

edit · good note | 0

Updated 1 day ago by Atri Rudra

Mid-term material

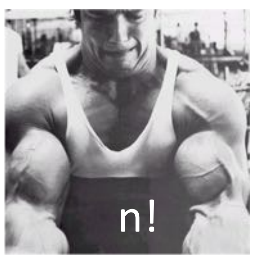
Everything we have covered so far (essentially Chaps 1-3 except Sec 1.2)

See piazza post on how to prepare for the mid-terms

Main Steps in Algorithm Design

Problem Statement → Problem Definition → Algorithm → "Implementation" → Analysis

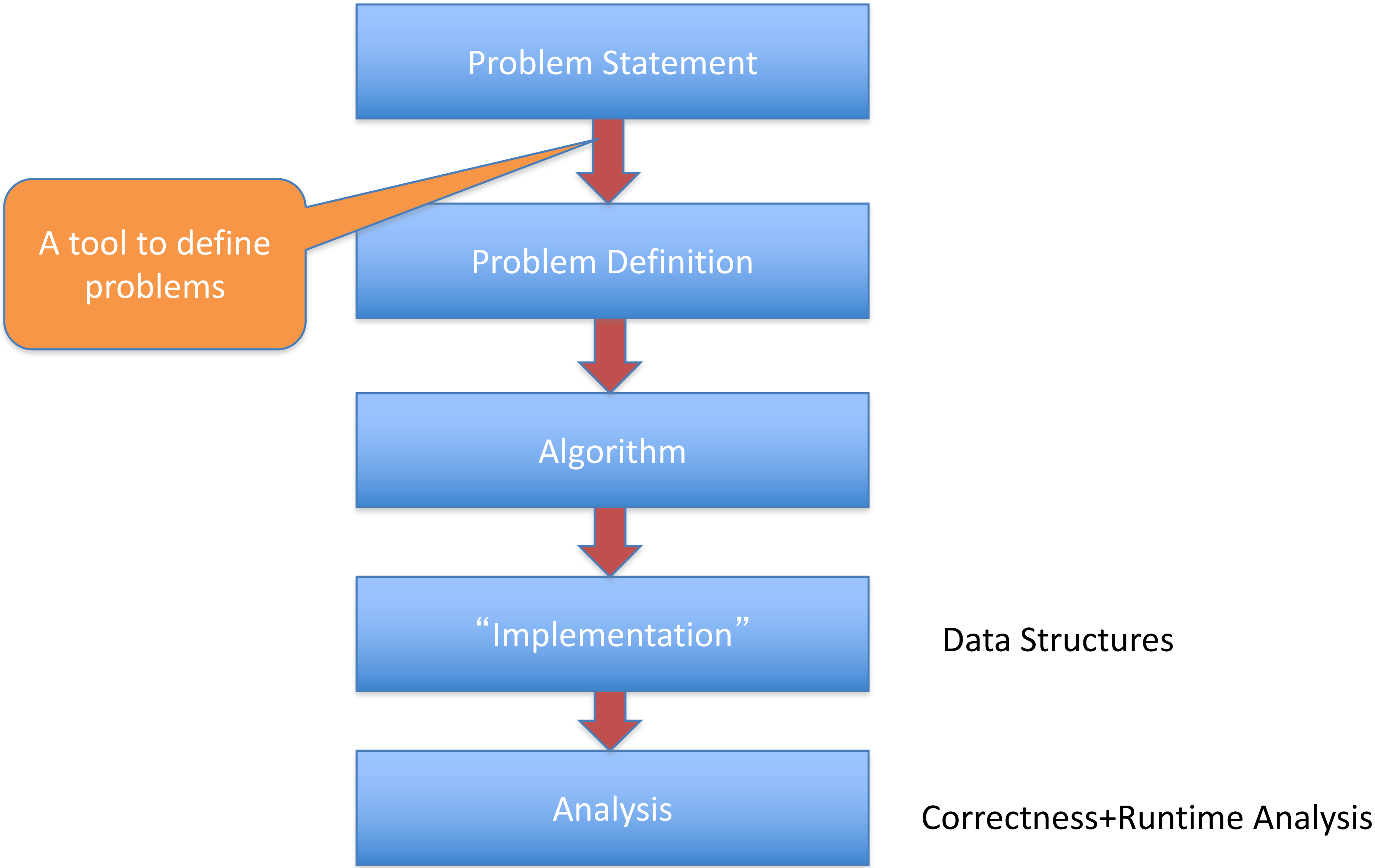
NRMP
National Resident Matching Program



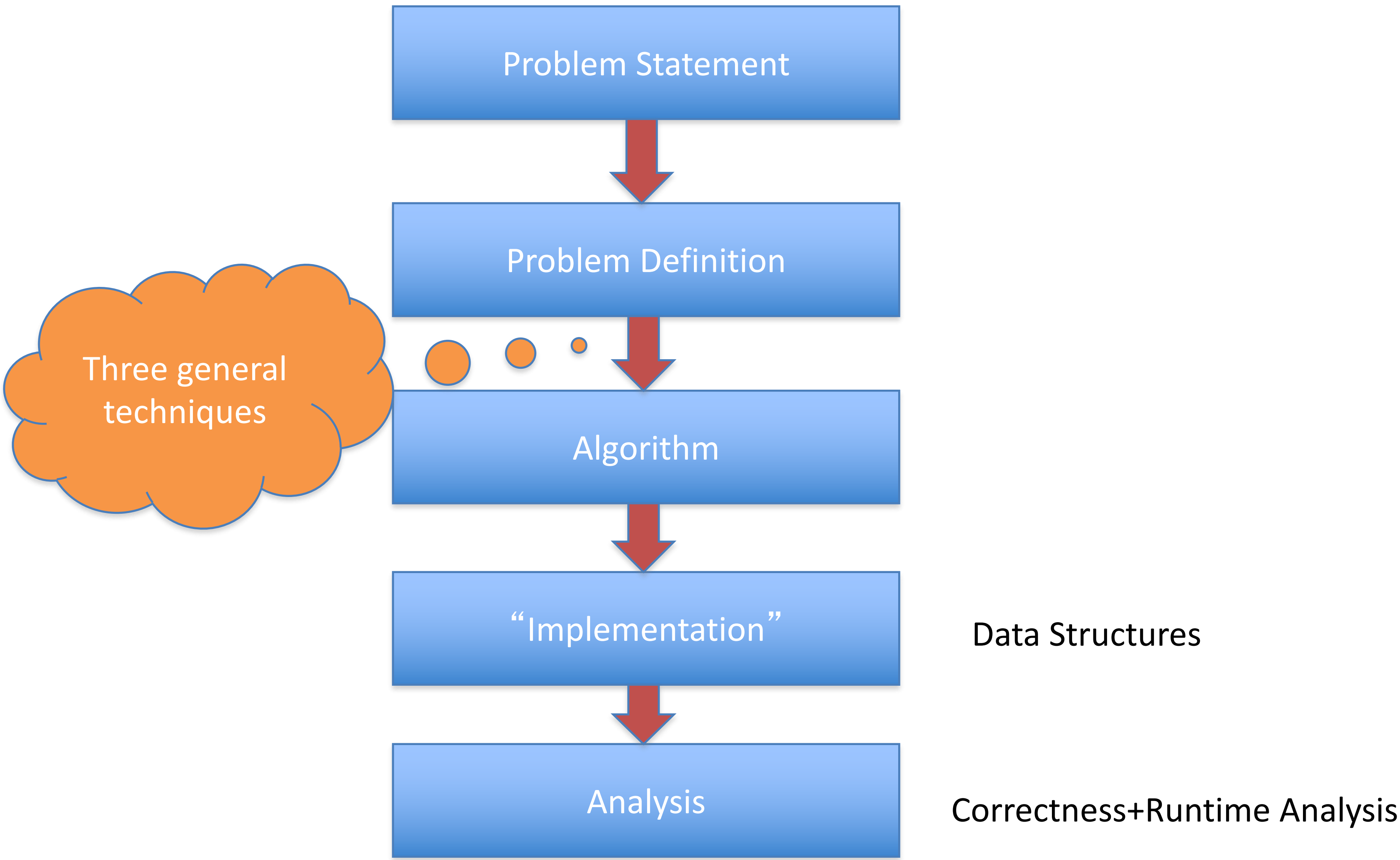
Data Structures

Correctness+Runtime Analysis

Where do graphs fit in?



Rest of the course



Greedy algorithms

Build the final solution piece by piece

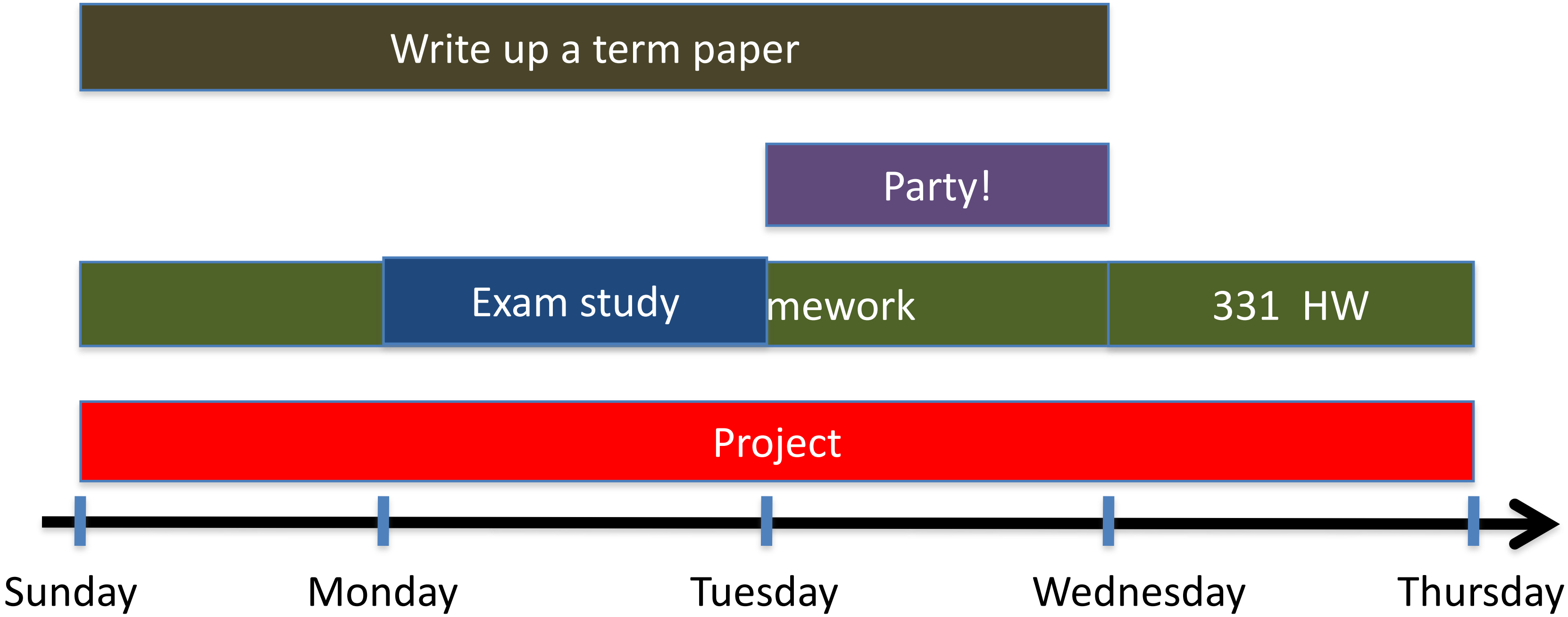
Being short sighted on each piece

Never undo a decision

Know when you see it

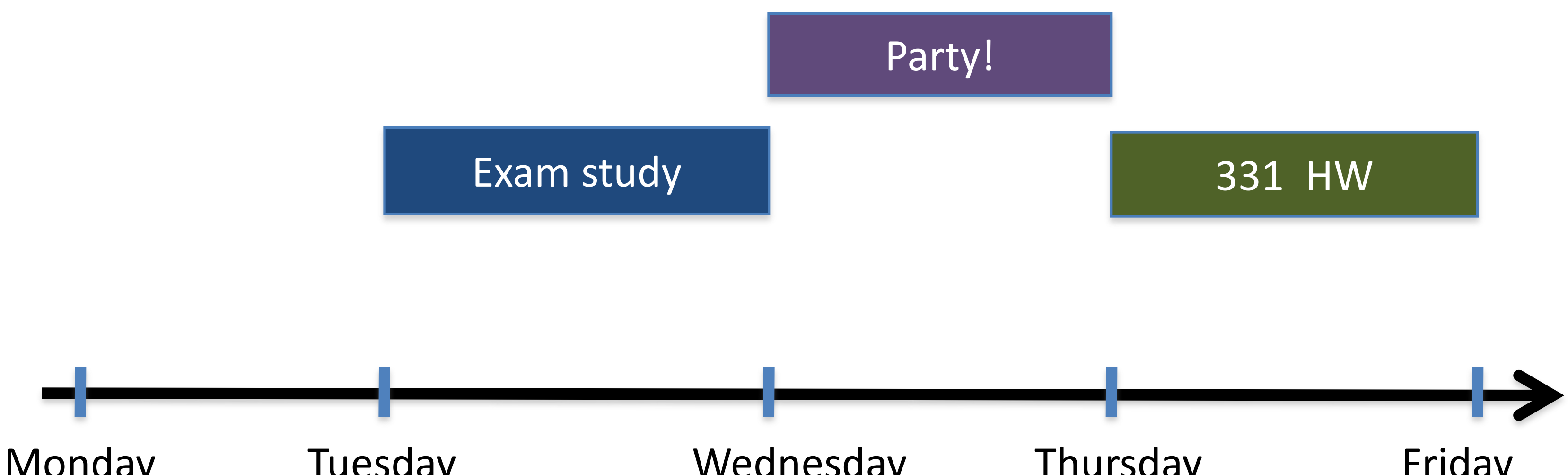
End of Semester blues

Can only do one thing at any day: what is the maximum number of tasks that you can do?



The optimal solution

Arrange tasks in some order and iteratively pick non-overlapping tasks



Interval Scheduling Problem

Input: n intervals $[s(i), f(i))$ for $1 \leq i \leq n$

$\{ s(i), \ldots, f(i)-1 \}$

Output: A *schedule* S of the n intervals

No two intervals in S conflict

$|S|$ is maximized

Algorithm with examples

Interval Scheduling via examples

In which we derive an algorithm that solves the Interval Scheduling problem via a sequence of examples.

The problem

In these notes we will solve the following problem:

Interval Scheduling Problem

Input: An input of n intervals $[s(i), f(i))$, or in other words, $\{s(i), \ldots, f(i)-1\}$ for $1 \le i \le n$ where i represents the intervals, $s(i)$ represents the start time, and $f(i)$ represents the finish time.

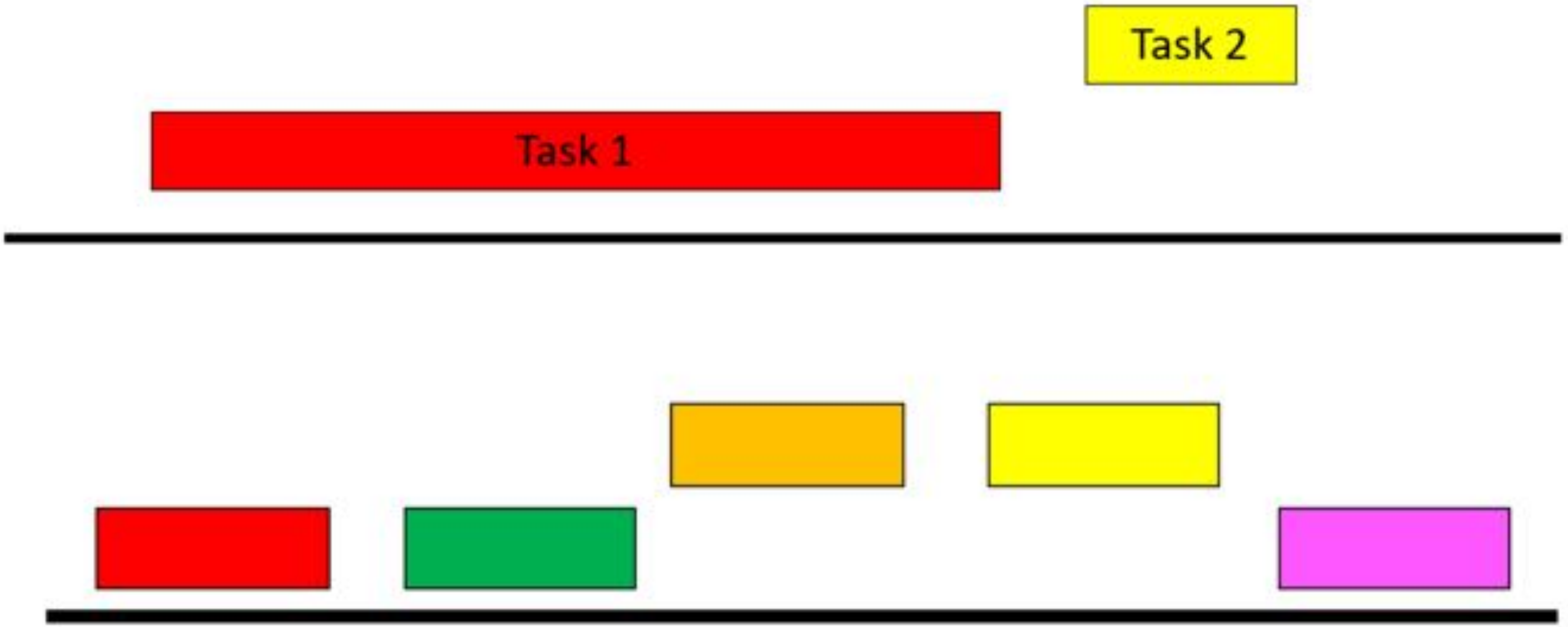
Output: A schedule S of n intervals where no two intervals in S conflict, and the total number of intervals in S is maximized.

Sample Input and Output

Input:

Example 1

No intervals overlap



Algorithm?

No intervals overlap

R: set of requests

Set S to be the empty set

While R is not empty

Choose i in R

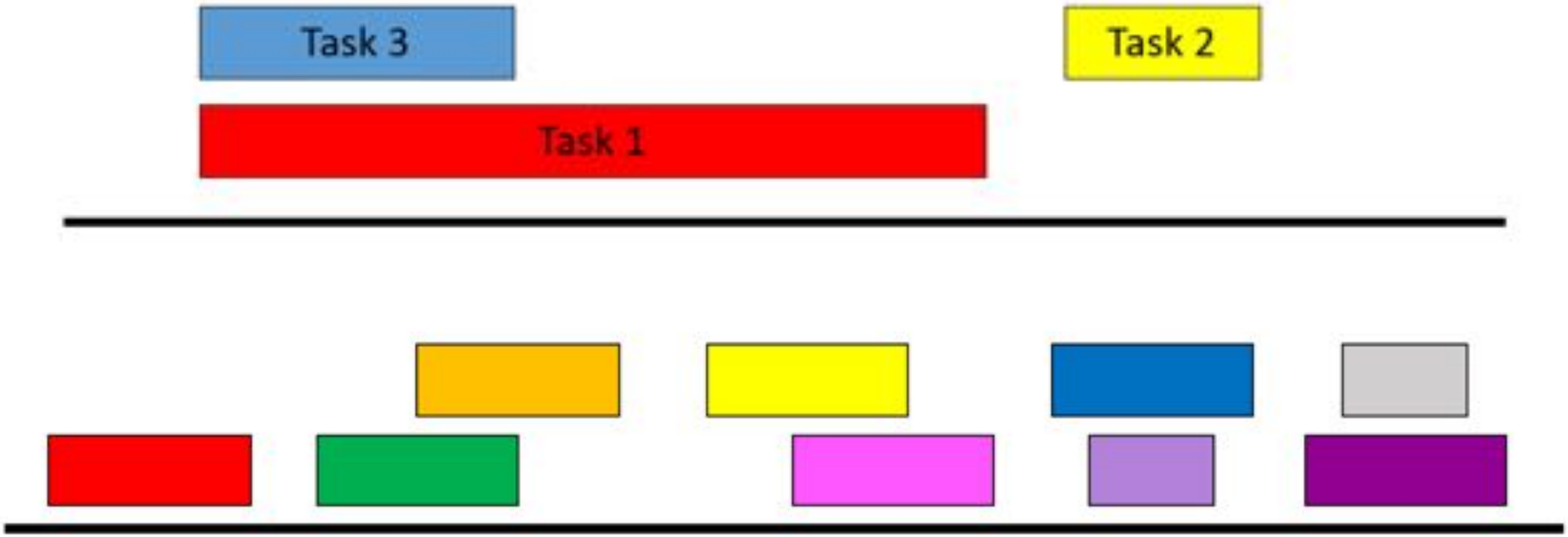
Add i to S

Remove i from R

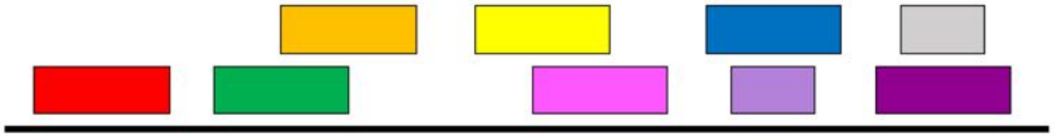
Return $S^*= S$

Example 2

At most one overlap



Algorithm?



At most one overlap

R: set of requests

Set S to be the empty set

While R is not empty

Choose i in R

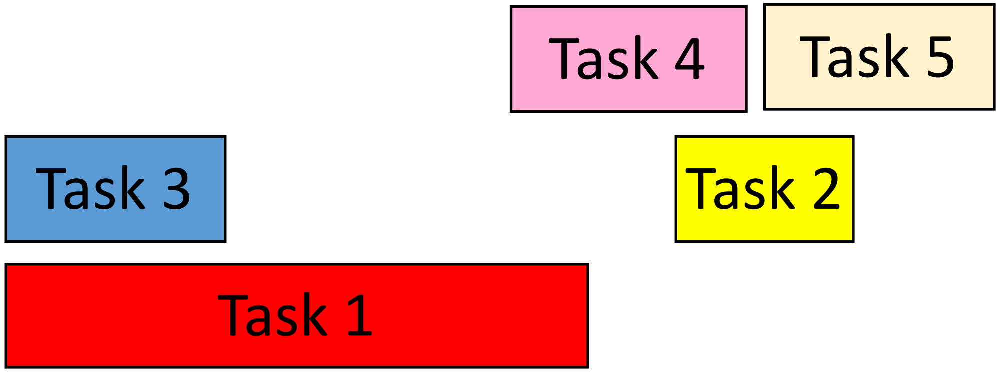
Add i to S

Remove all tasks that conflict with i from R

Return $S^* = S$

Example 3

More than one conflict



Set S to be the empty set

While R is not empty

Choose i in R

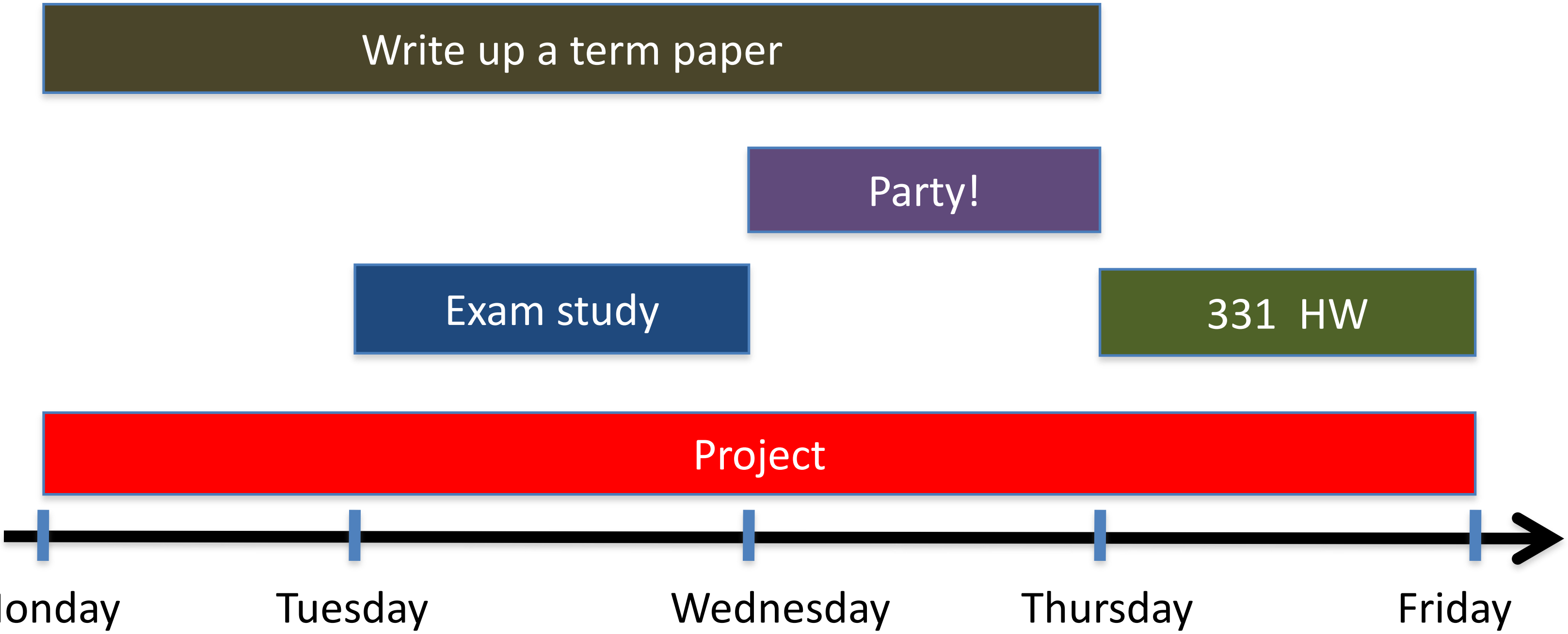
Add i to S

Remove all tasks that conflict with i from R

Return $S^* = S$

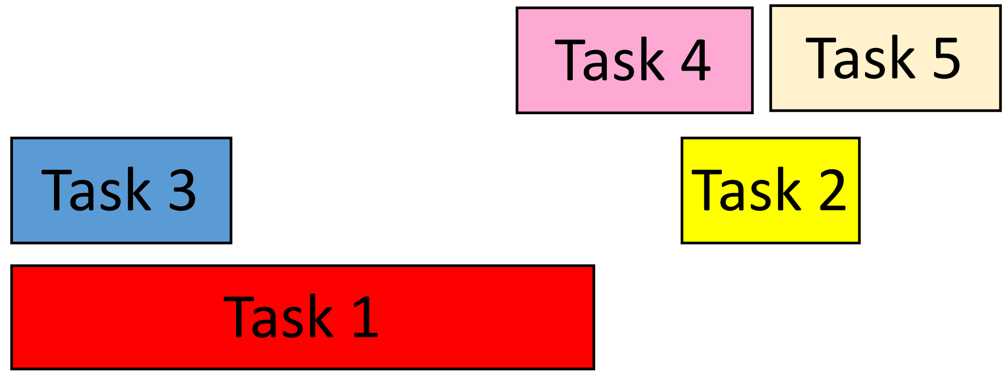
Greedily solve your blues!

Arrange tasks in some order and iteratively pick non-overlapping tasks



Making it more formal

More than one conflict



Set S to be the empty set

While R is not empty

    Choose i in R that minimizes v(i)

    Add i to S

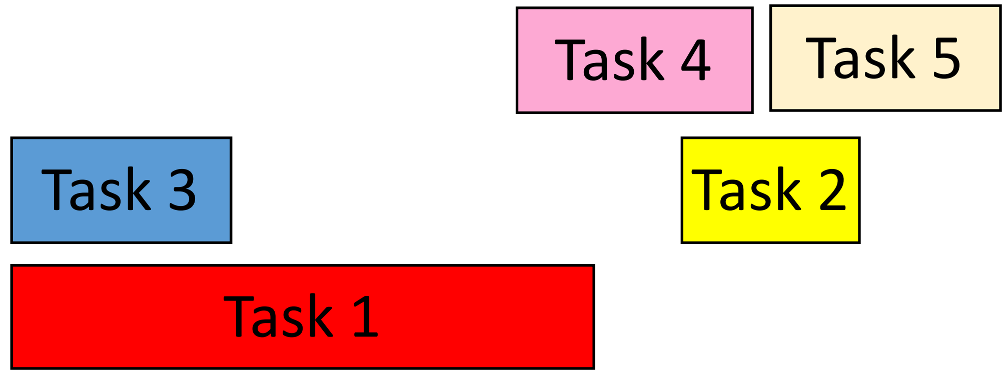
    Remove all tasks that conflict with i from R

Return S^*= S

Associate a value v(i) with task i

What is a good choice for v(i)?

More than one conflict



Set S to be the empty set

While R is not empty

Choose i in R that minimizes v(i)

Add i to S

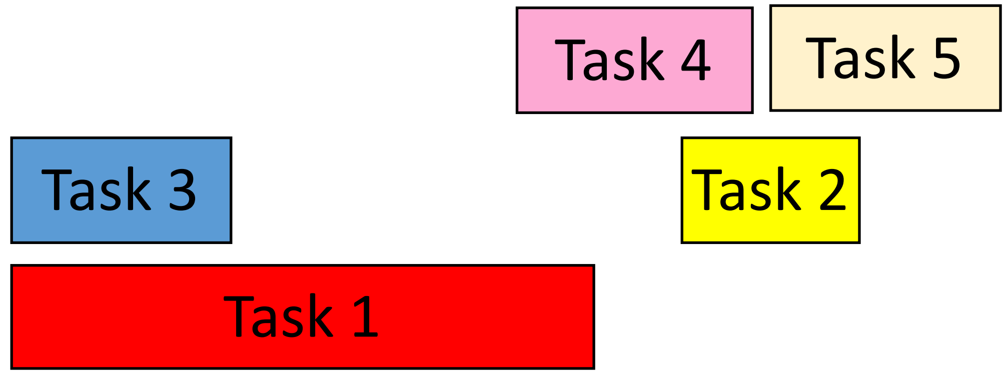
Remove all tasks that conflict with i from R

Return $S^* = S$

Associate a value v(i) with task i

$v(i) = f(i) - s(i)$

Smallest duration first



Set S to be the empty set

While R is not empty

  Choose i in R that minimizes $f(i) - s(i)$

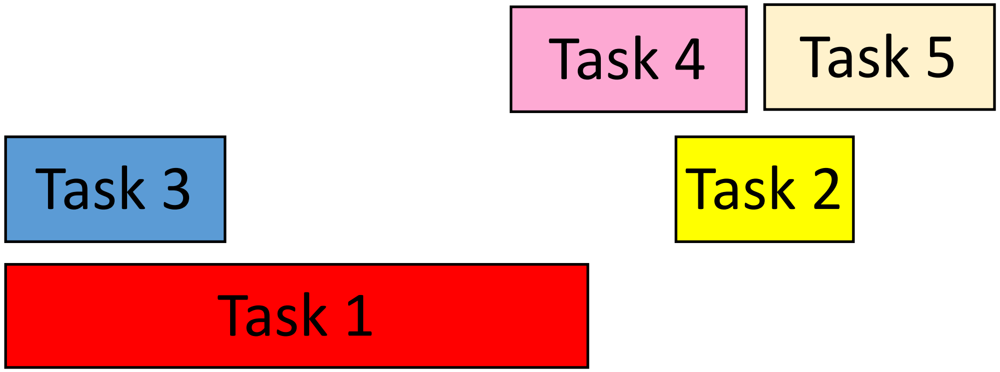
  Add i to S

  Remove all tasks that conflict with i from R

Return $S^* = S$

$v(i) = s(i)$

Earliest time first?



Set S to be the empty set

While R is not empty

  Choose i in R that minimizes s(i)

  Add i to S

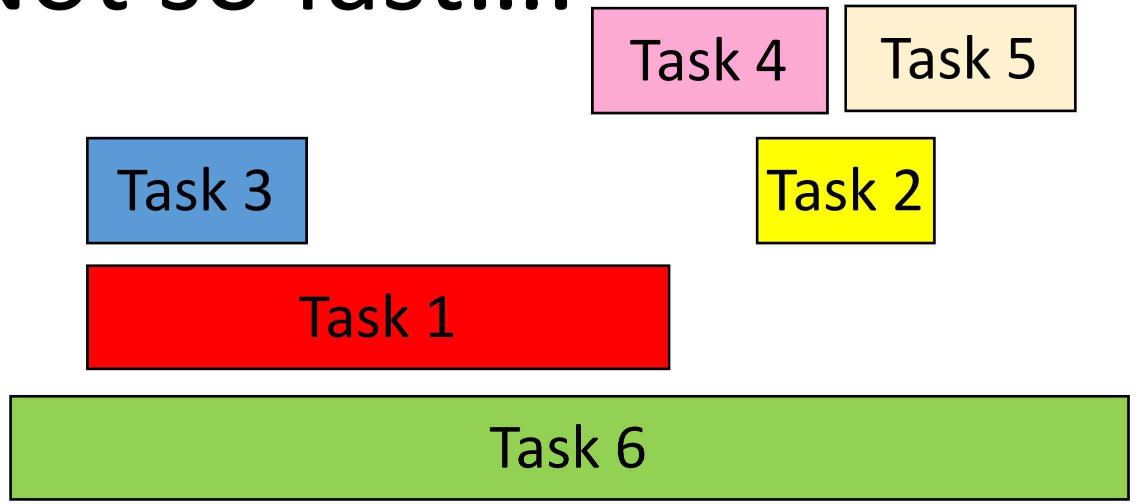
  Remove all tasks that conflict with i from R

Return $S^* = S$

So are we done?

Not so fast….

Earliest time first?



Set S to be the empty set

While R is not empty

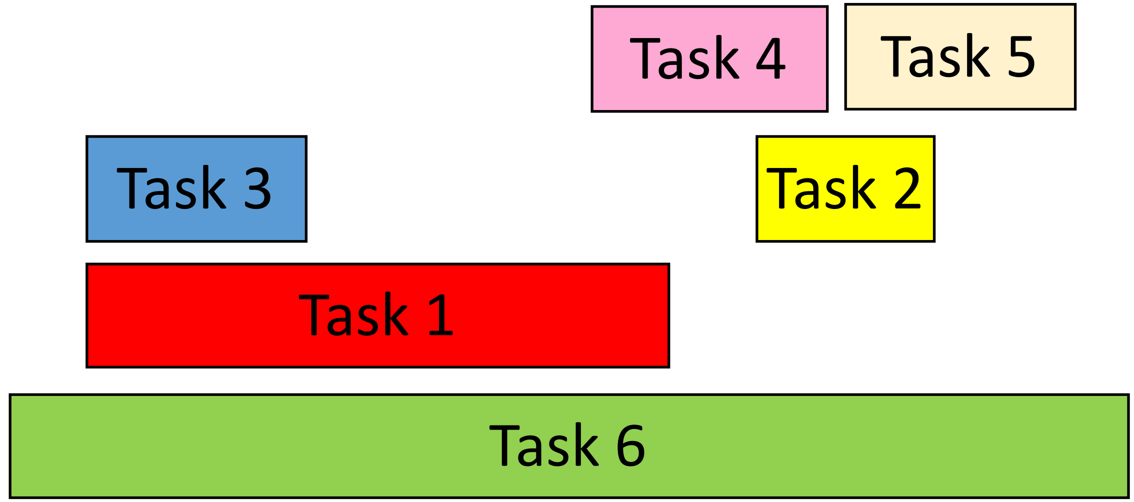
Choose i in R that minimizes s(i)

Add i to S

Remove all tasks that conflict with i from R

Return $S^* = S$

Pick job with minimum conflicts



Set S to be the empty set

While R is not empty

  Choose i in R that has smallest number of conflicts

  Add i to S

  Remove all tasks that conflict with i from R

Return $S^* = S$

So are we done?

Nope (but harder to show)

Set S to be the empty set

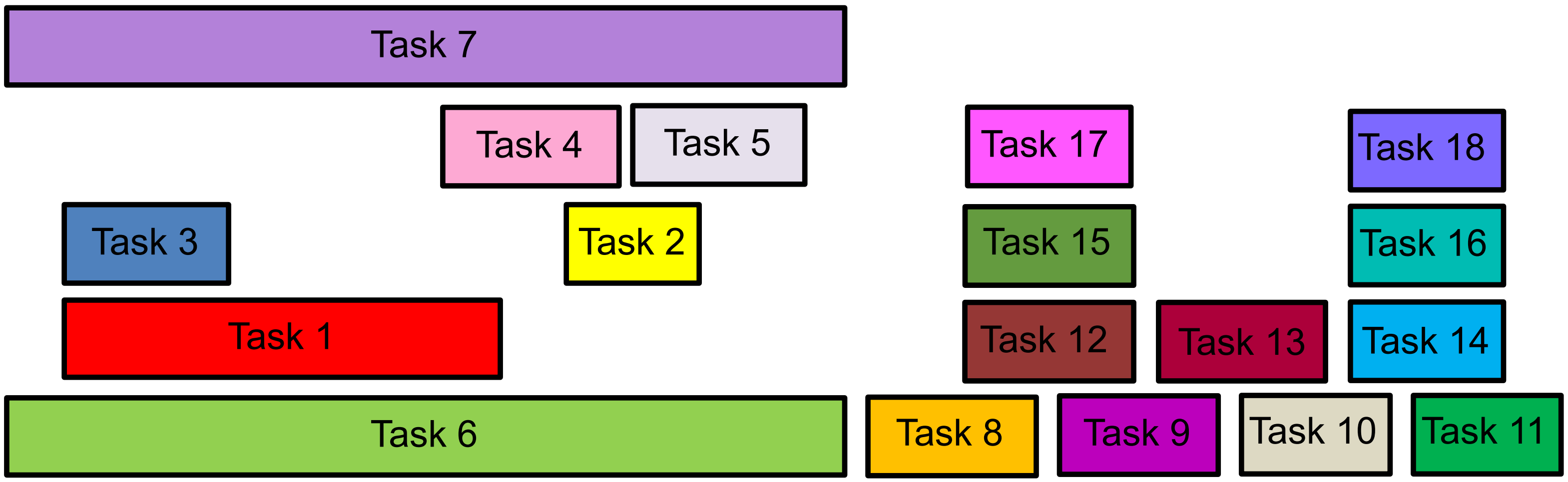
While R is not empty

Choose i in R that has smallest number of conflicts

Add i to S

Remove all tasks that conflict with i from R

Return $S^* = S$



Set S to be the empty set

While R is not empty

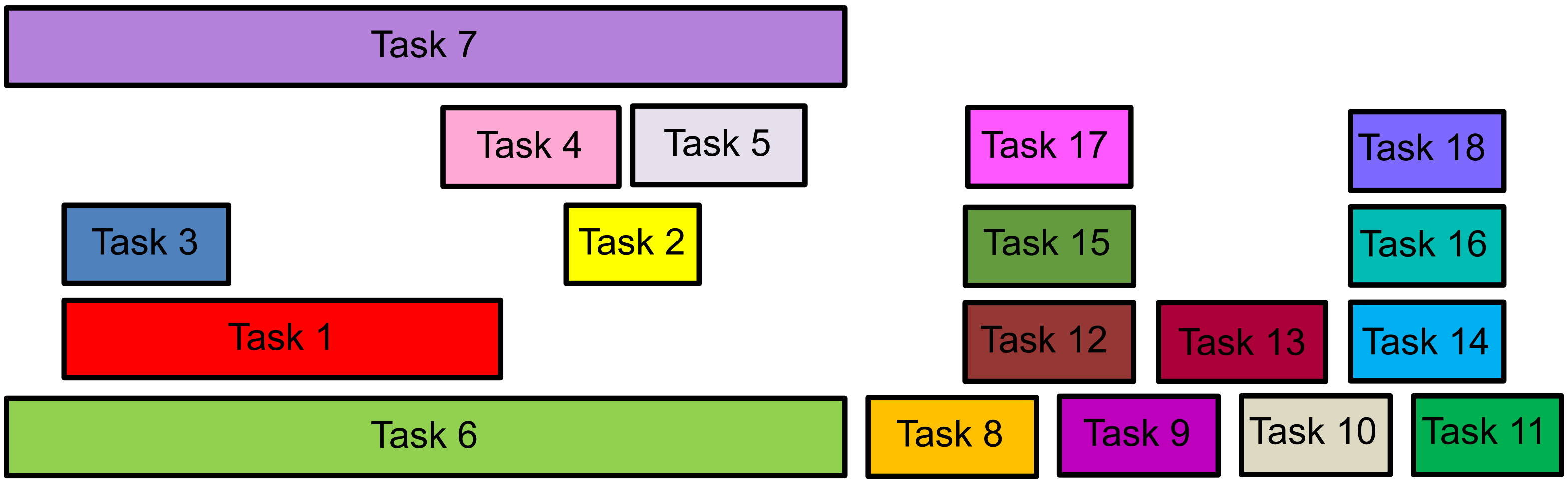
Choose i in R that has smallest number of conflicts

Add i to S

Remove all tasks that conflict with i from R

Return $S^* = S$

Algorithm?



Set S to be the empty set

While R is not empty

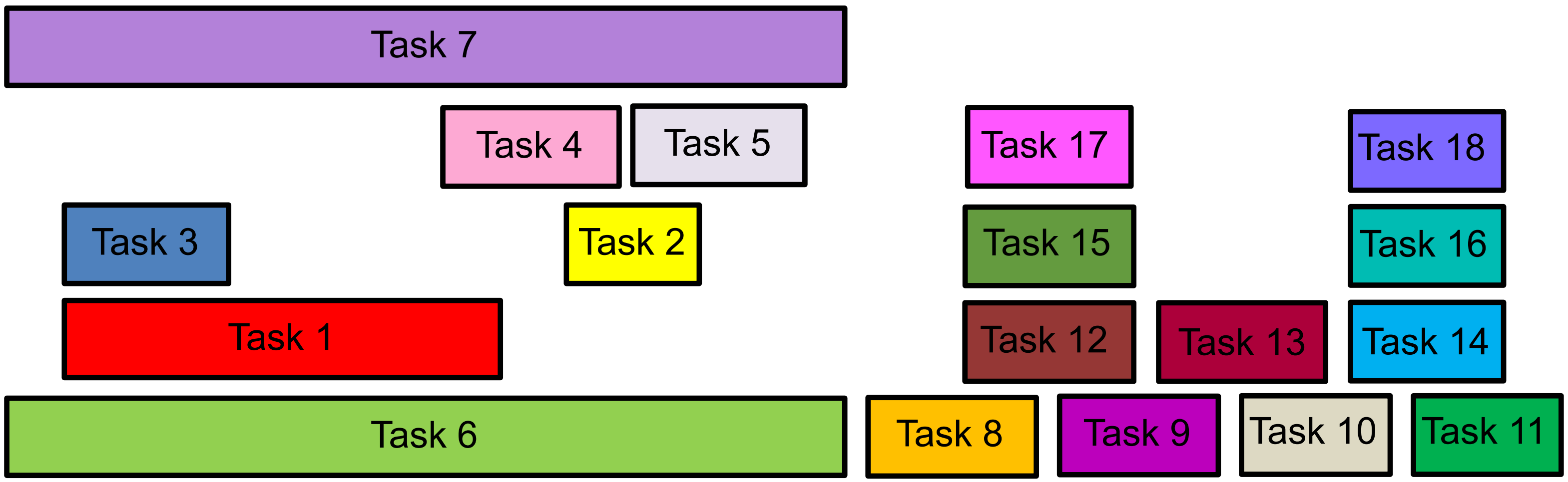
Choose i in R that minimizes v(i)

Add i to S

Remove all tasks that conflict with i from R

Return $S^* = S$

Earliest finish time first



Set S to be the empty set

While R is not empty

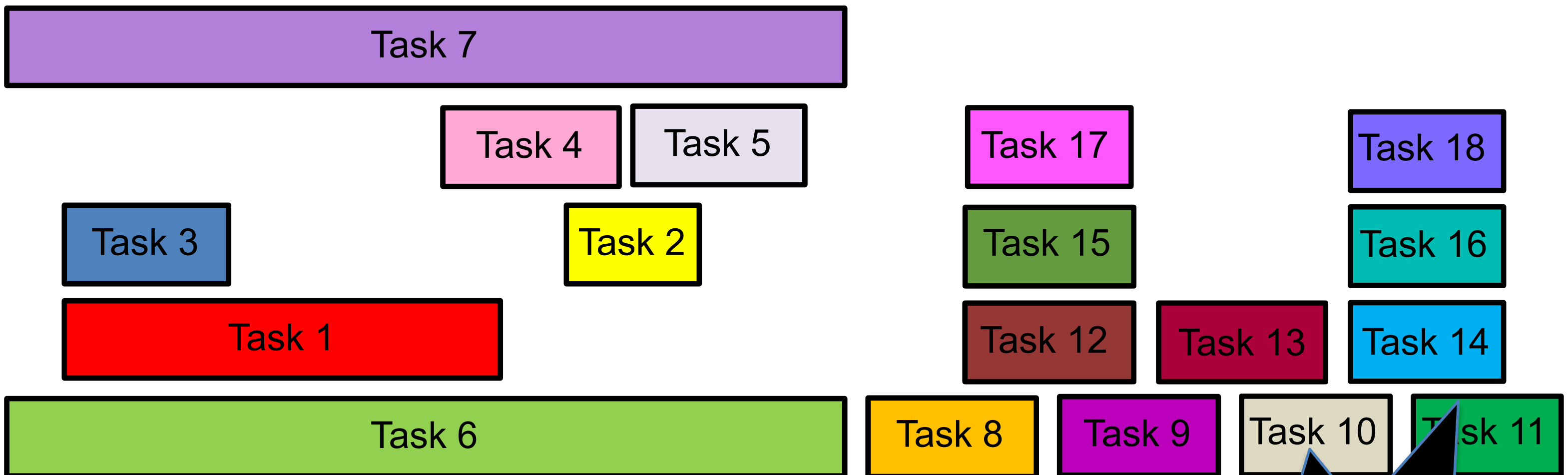
Choose i in R that minimizes f(i)

Add i to S

Remove all tasks that conflict with i from R

Return $S^* = S$

Find a counter-example?



Set S to be the empty set

While R is not empty

Choose i in R that minimizes f(i)

Add i to S

Remove all tasks that conflict with i from

Return S* = S

It works!

Questions?

Today’s agenda

Prove the correctness of the algorithm

Final Algorithm

R: set of requests

Set S to be the empty set

While R is not empty

Choose i in R with the earliest finish time

Add i to S

Remove all requests that conflict with i from R

Return $S^* = S$