Lecture 15

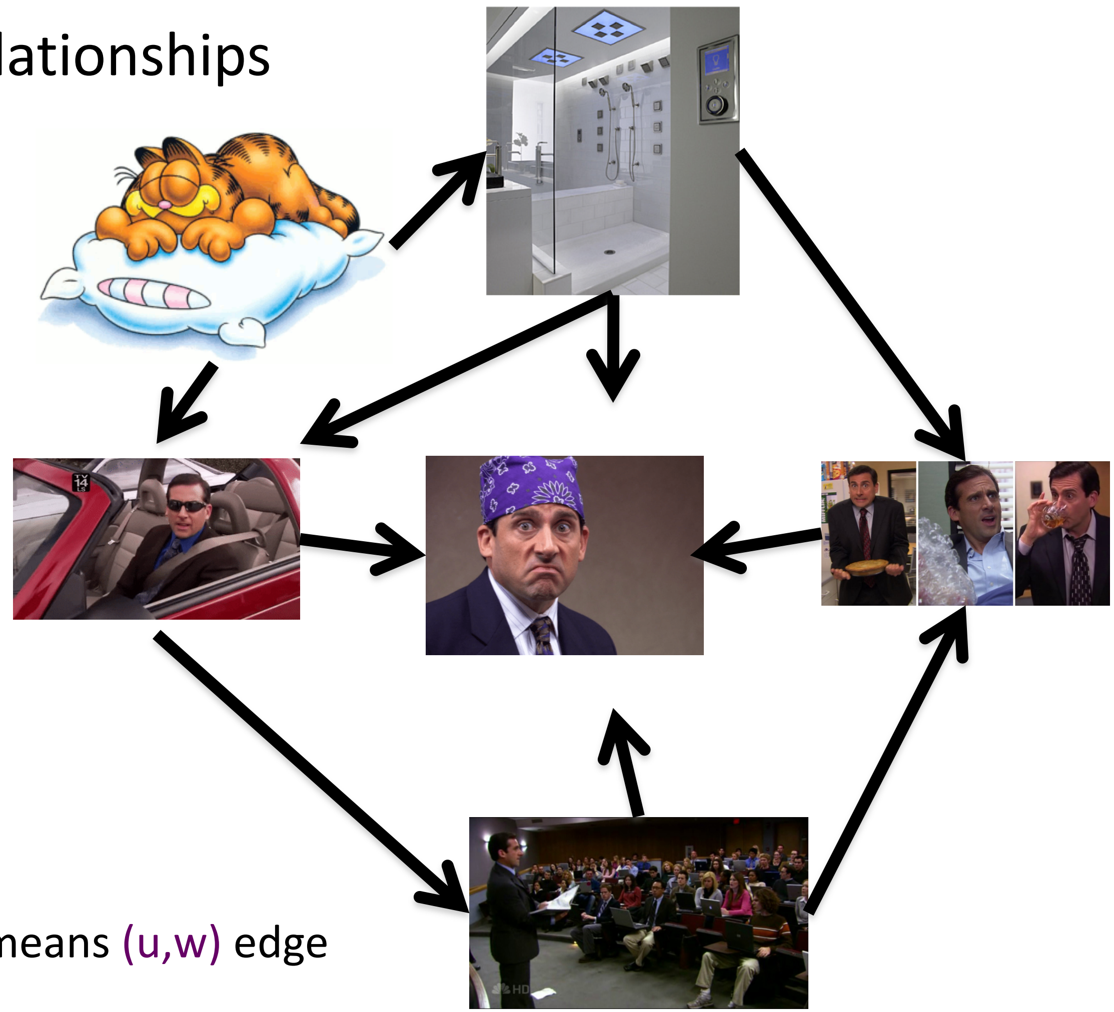
CSE 331

Mar 5, 2021

Directed graphs

Model asymmetric relationships

Precedence relationships

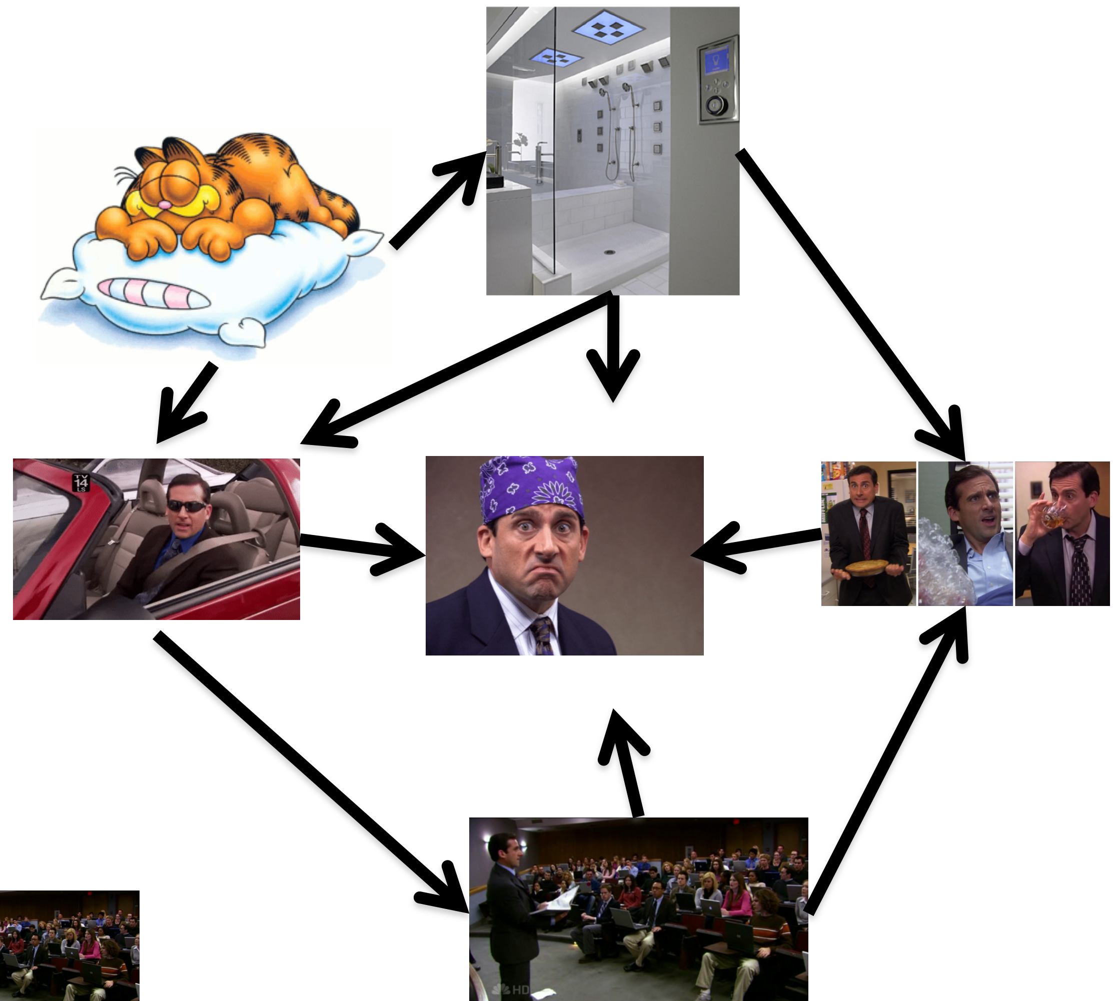


u needs to be done before w means (u,w) edge

Directed graphs

Adjacency matrix is not symmetric

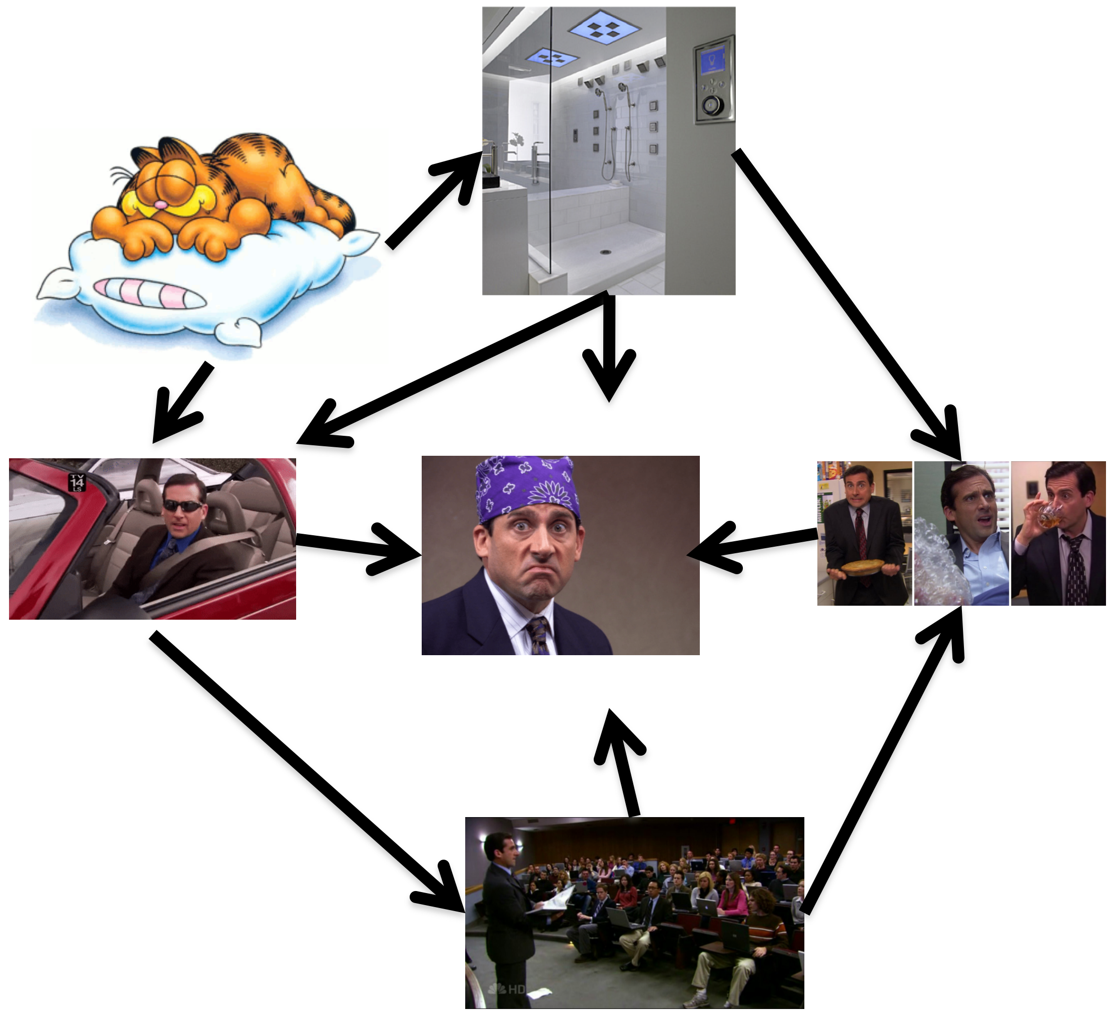
Each vertex has two lists in Adj. list rep.



Directed Acyclic Graph (DAG)

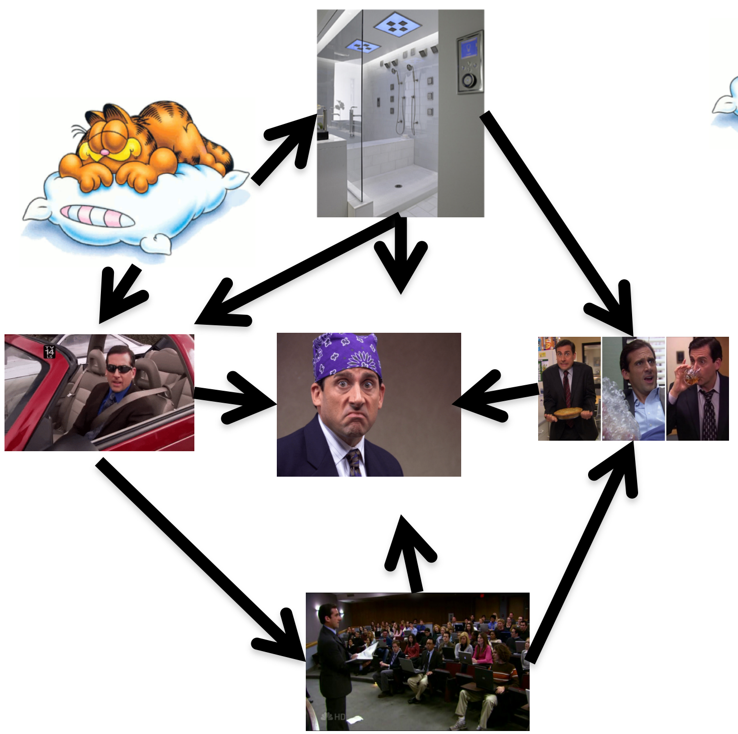
No directed cycles

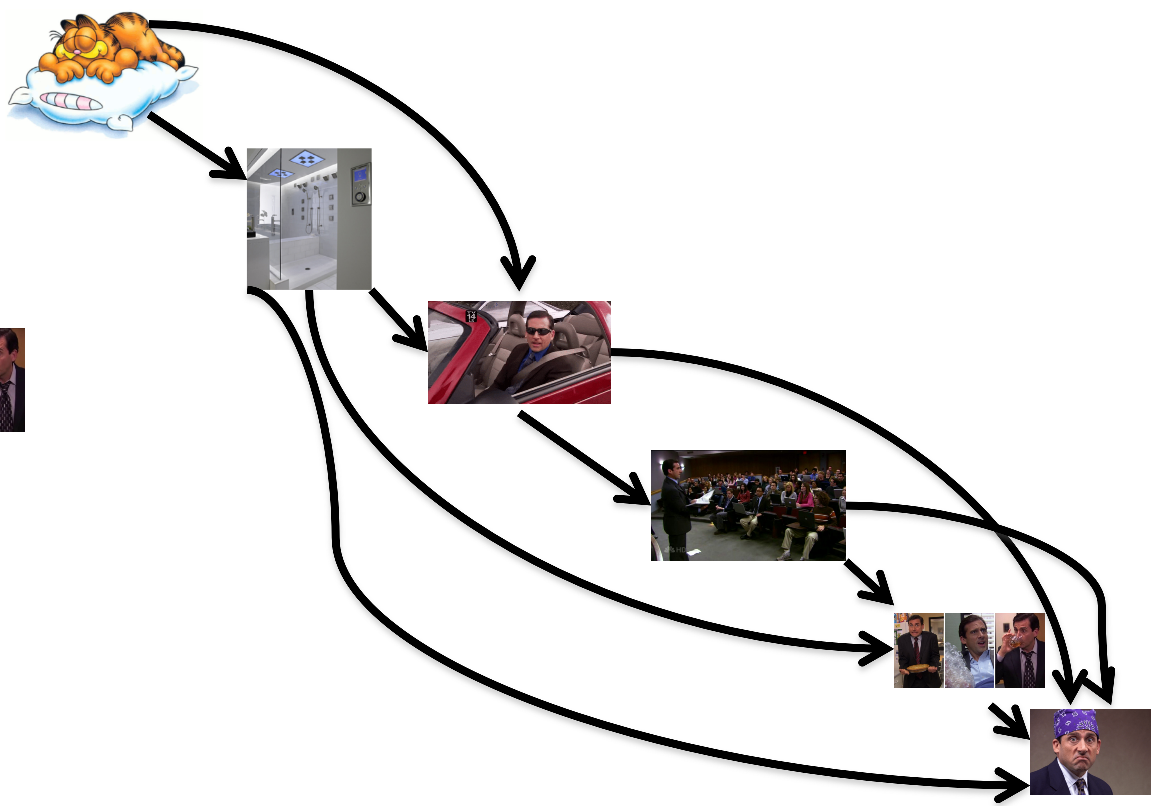
Precedence relationships are consistent



Topological Sorting of a DAG

Order the vertices so that all edges go "forward"





More details on Topological sort

Topological Ordering

This page collects material from previous incarnations of CSE 331 on topological ordering.

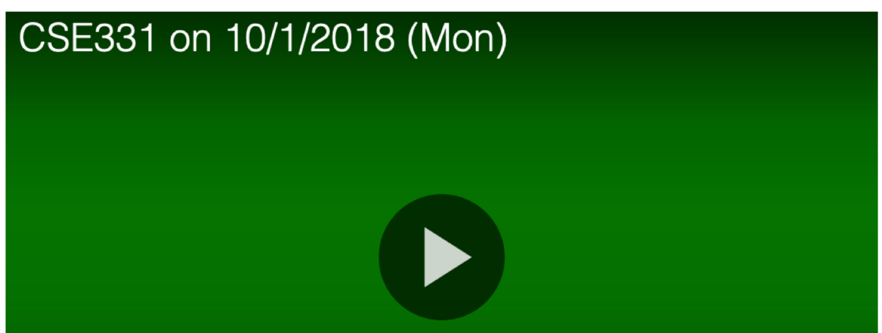
Where does the textbook talk about this?

`Section 3.6` in the textbook has the lowdown on topological ordering.

Fall 2018 material

First lecture

Here is the lecture video:



Main Steps in Algorithm Design

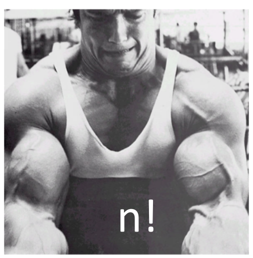
Problem Statement

NRMP
National Resident Matching Program

↓

Problem Definition

↓



Algorithm

↓

"Implementation"

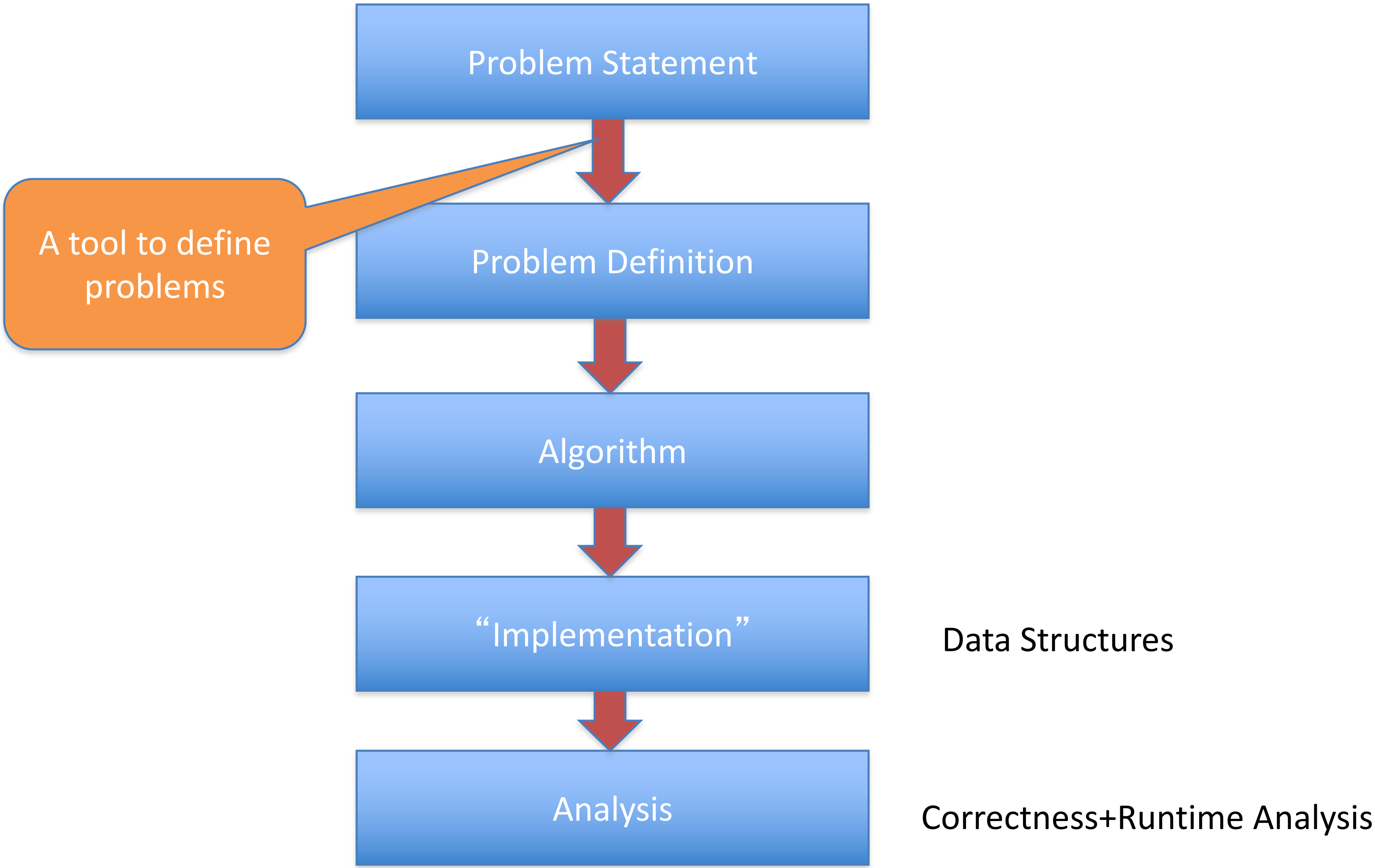
Data Structures

↓

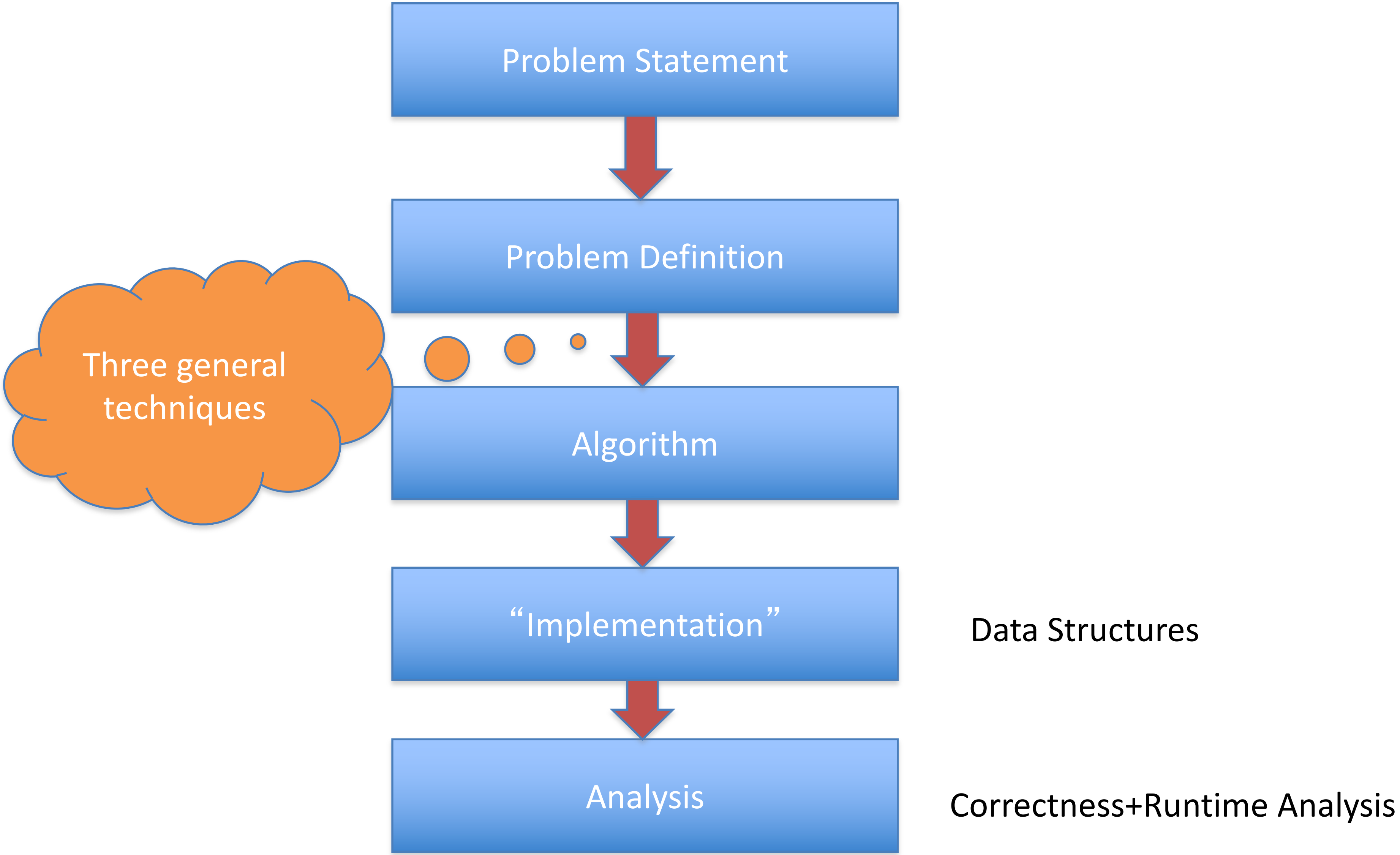
Analysis

Correctness+Runtime Analysis

Where do graphs fit in?

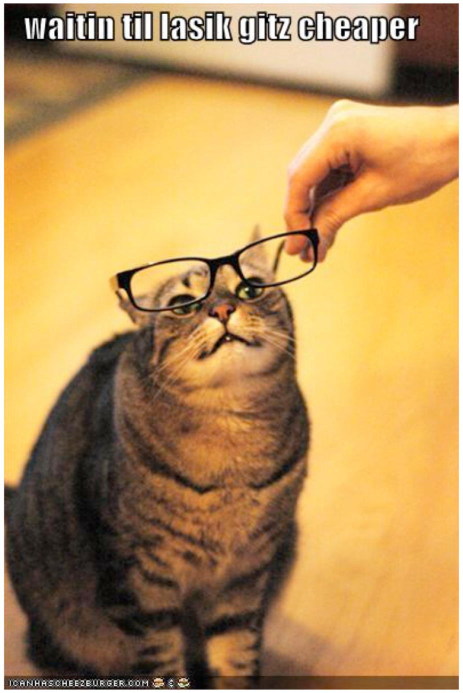


Rest of the course*



Greedy algorithms

Build the final solution piece by piece



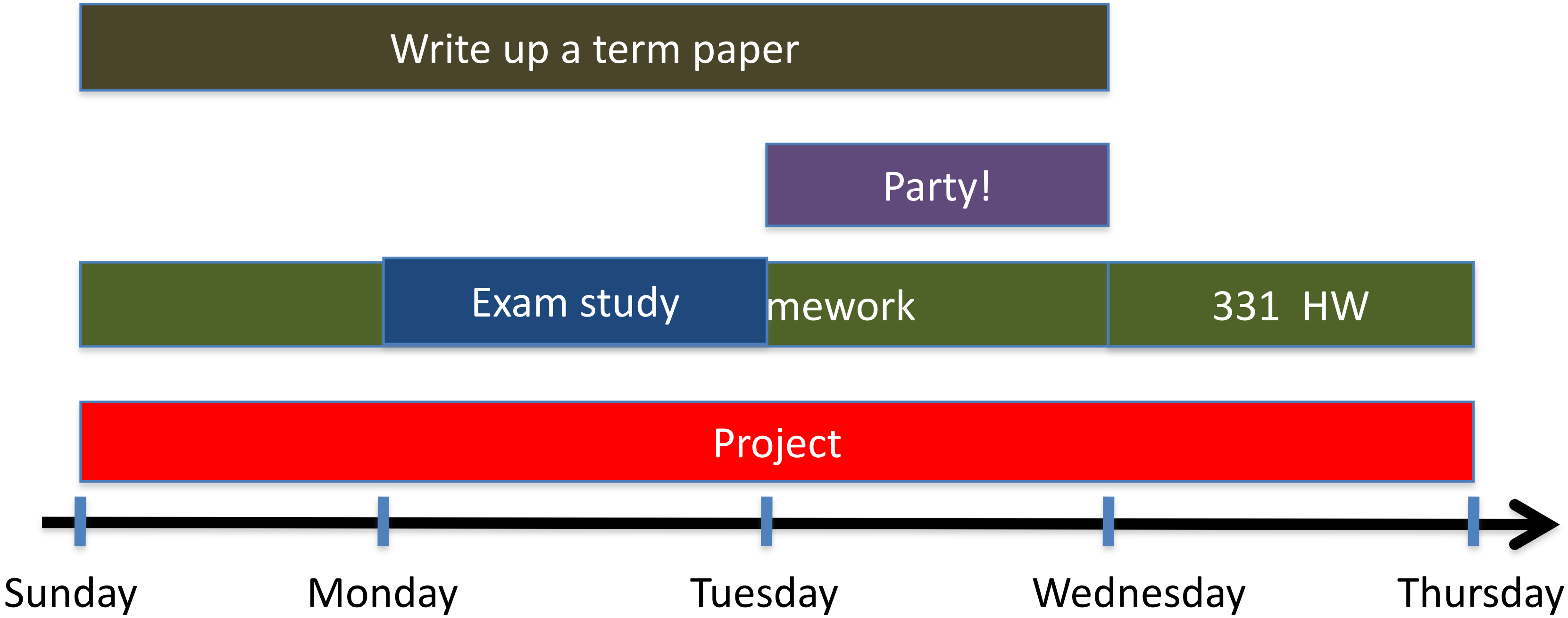
Being short sighted on each piece

Never undo a decision

Know when you see it

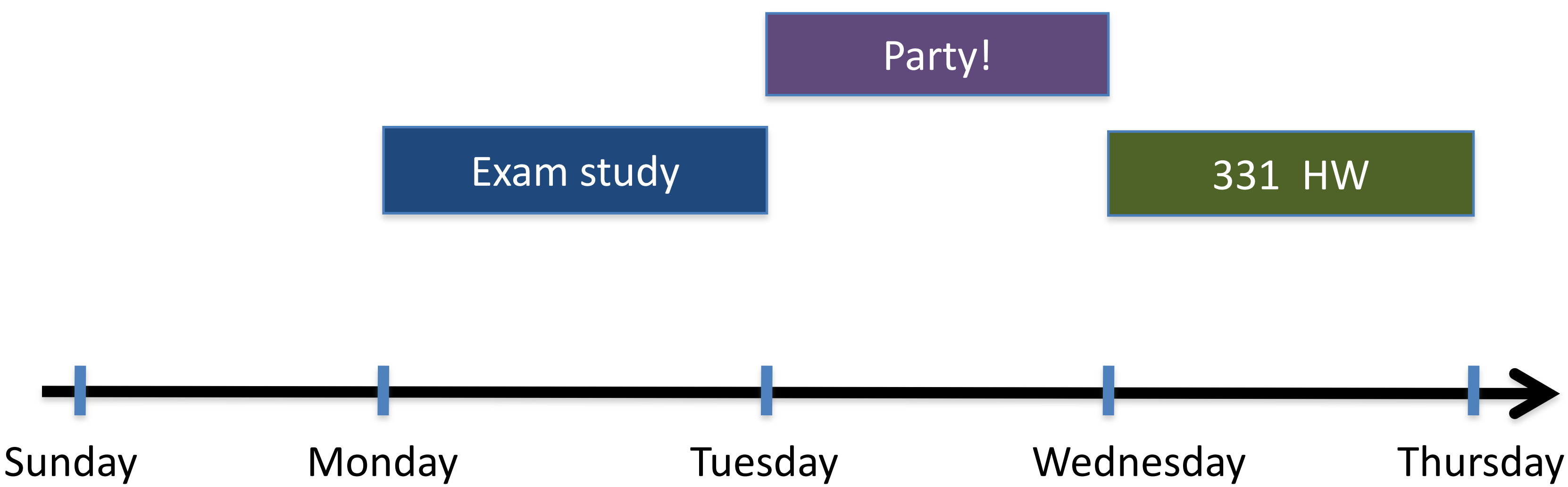
End of Semester blues

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



The optimal solution

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



Interval Scheduling Problem

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

{ s(i), … ,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 $1 \leq i \leq n$ where i represents the intervals, $s(i)$ represents the start time, and $f(i)$ represents the finish time.

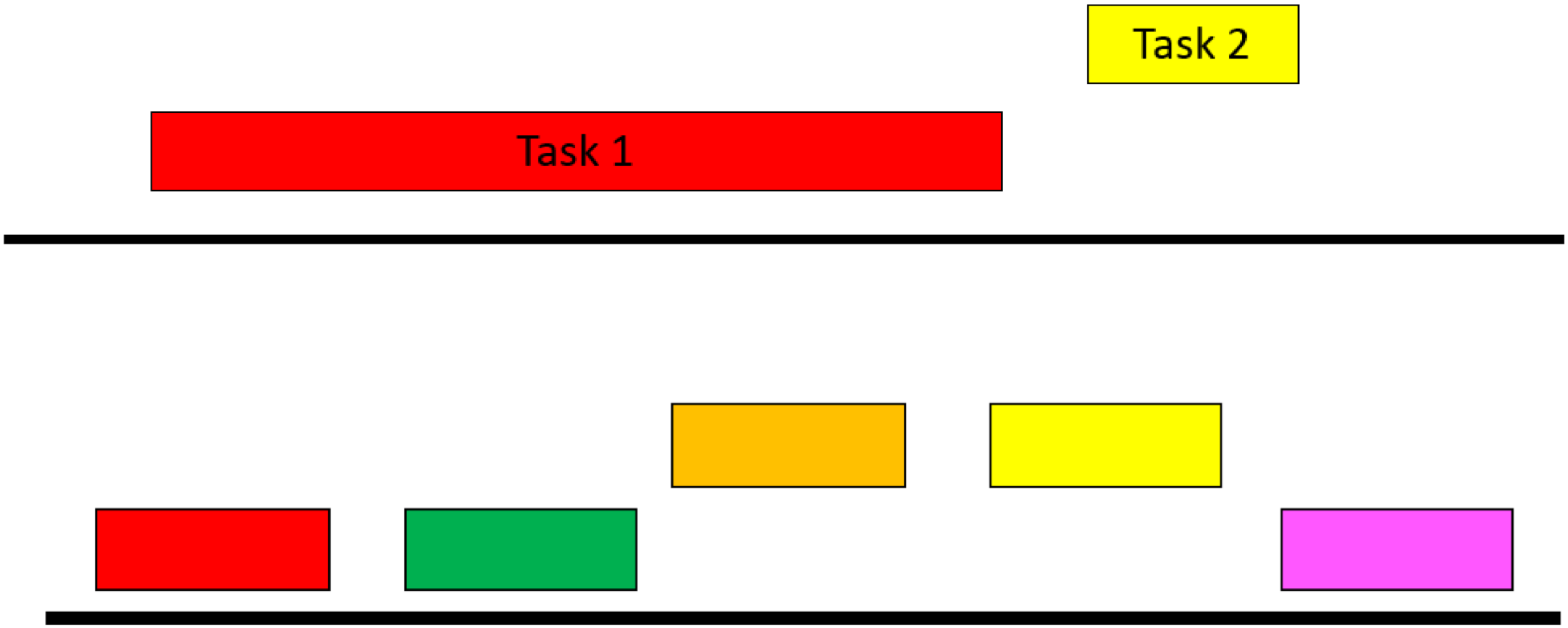
Output: A schedule S of n intervals where no two intervals in S conflict, and the total number of intervals in S is maximized.

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

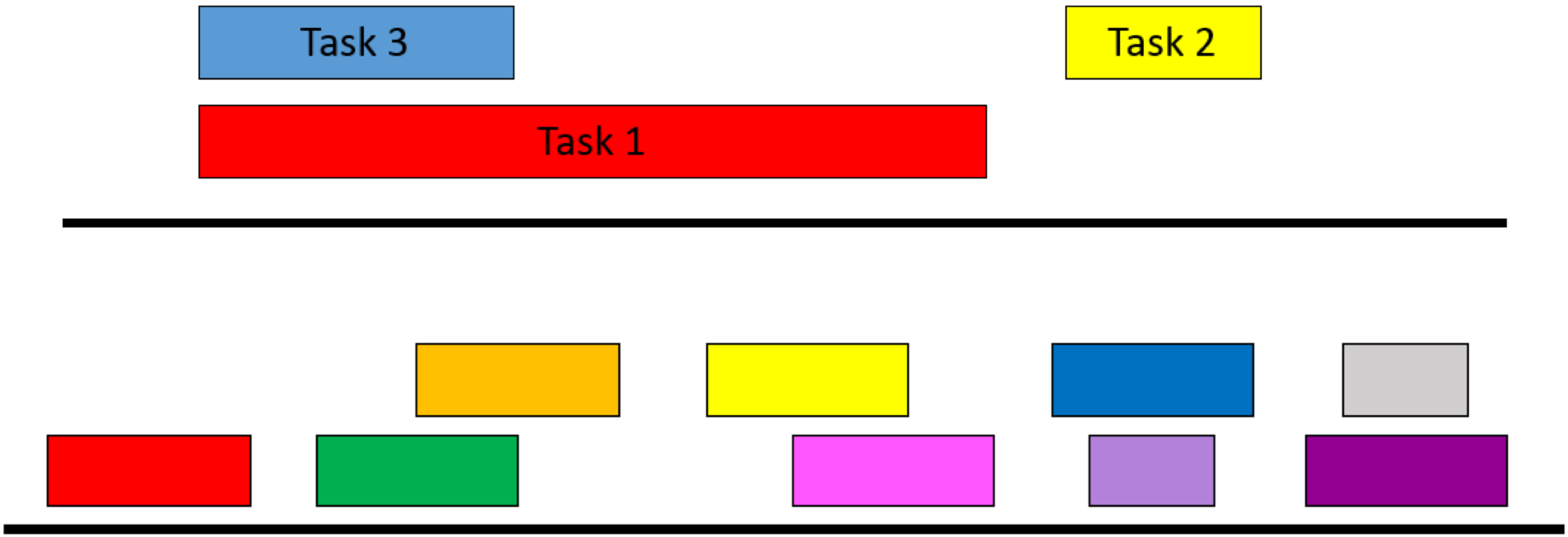
Add i to S

Remove i from R

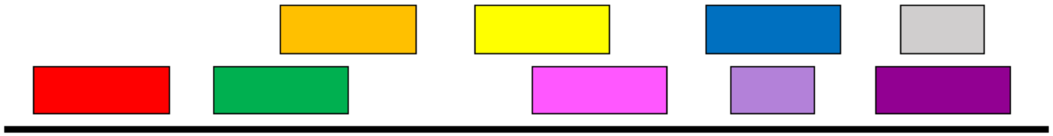
Return S^*= S

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

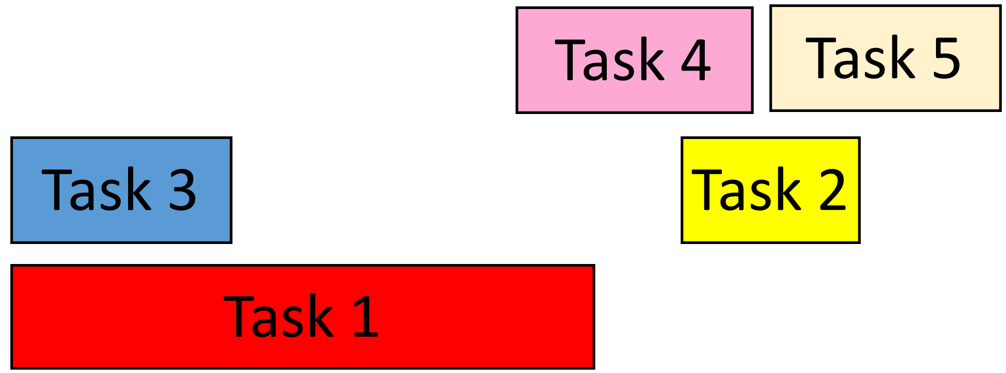
Add i to S

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

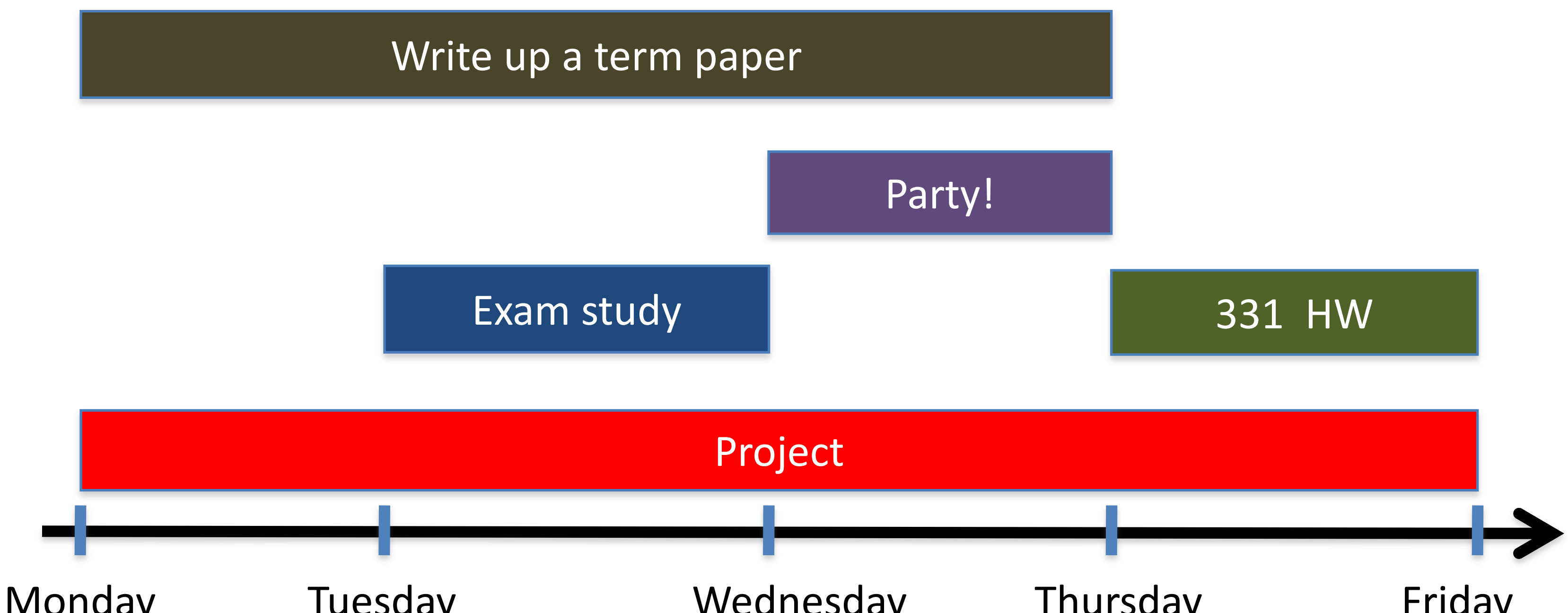
Add i to S

Remove all tasks that conflict with i from R

Return $S^* = S$

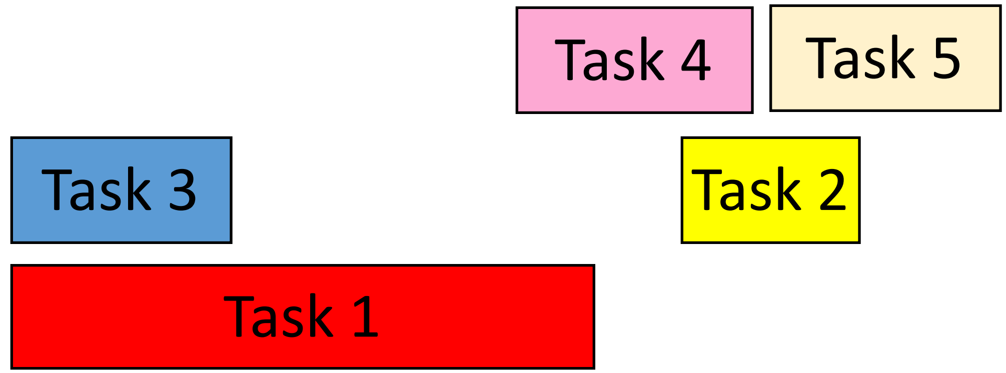
Greedily solve your blues!

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

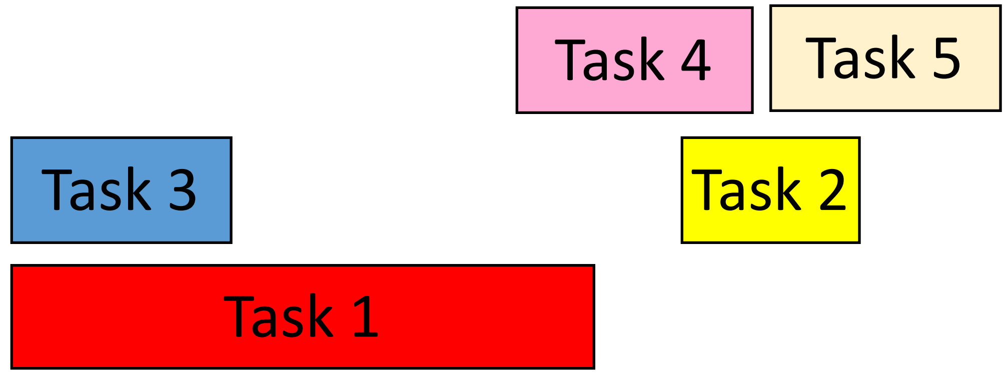
  Remove all tasks that conflict with i from R

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

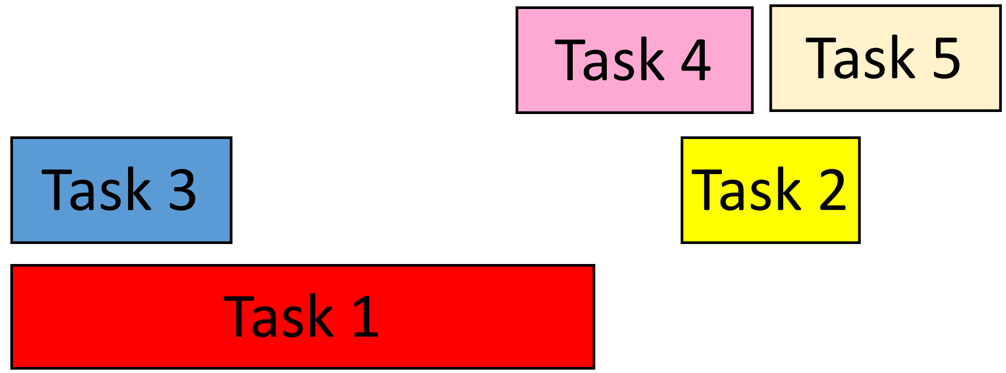
Remove all tasks that conflict with i from R

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 $f(i) - s(i)$

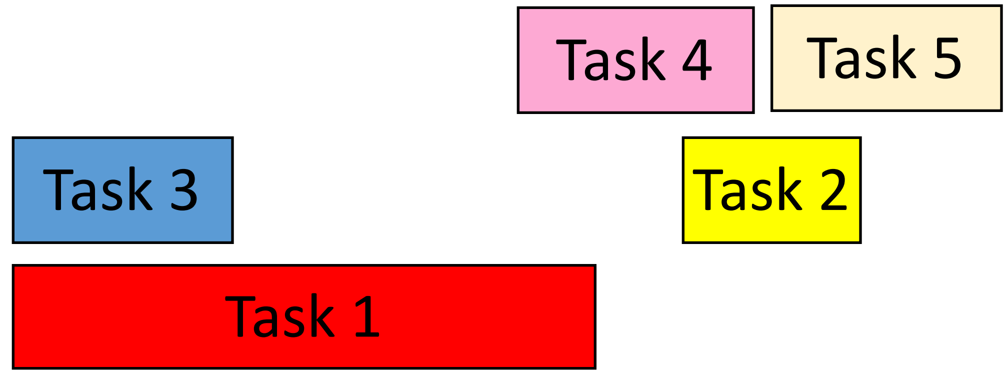
    Add i to S

    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 $s(i)$

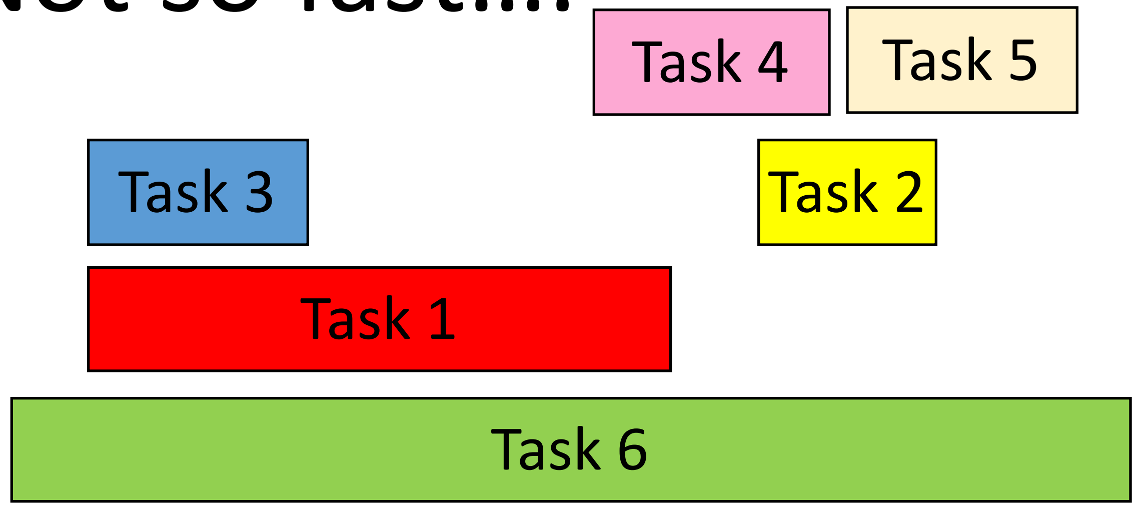
Add i to S

Remove all tasks that conflict with i from R

Return $S^* = S$

Not so fast….

Earliest time first?



Set S to be the empty set

While R is not empty

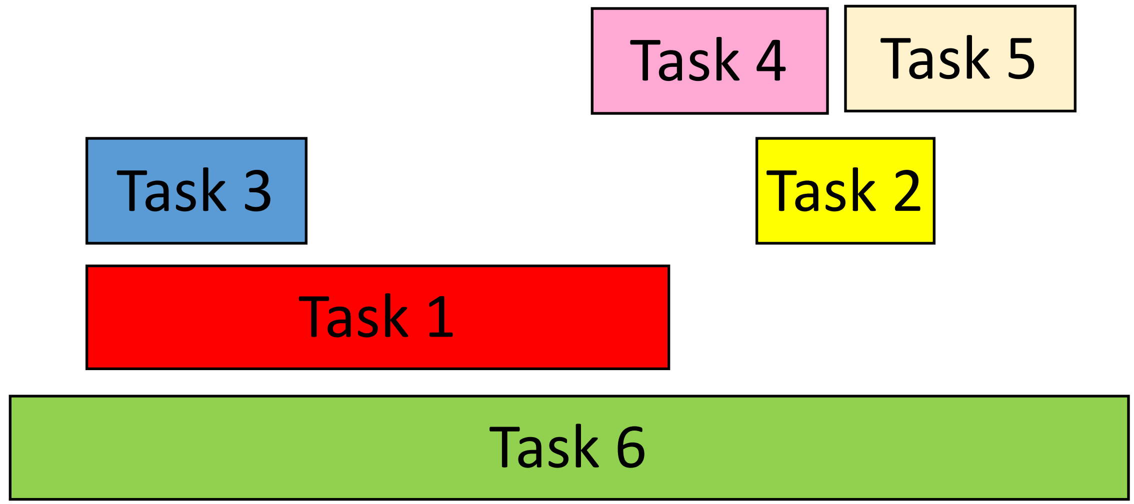
Choose i in R that minimizes s(i)

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

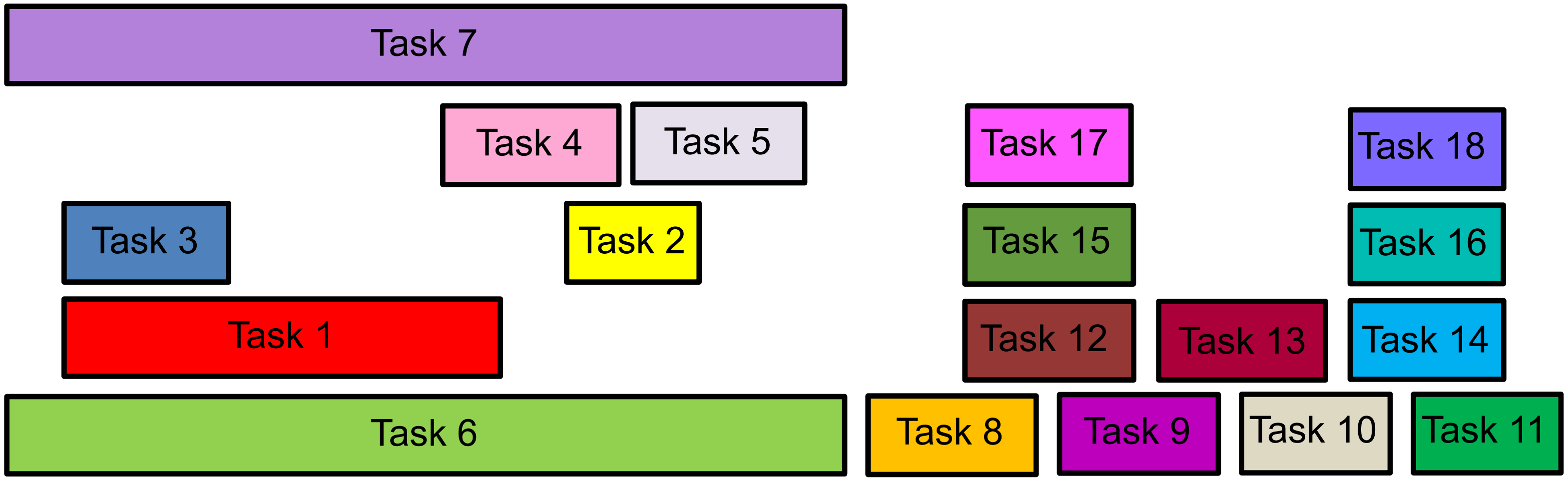
While R is not empty

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

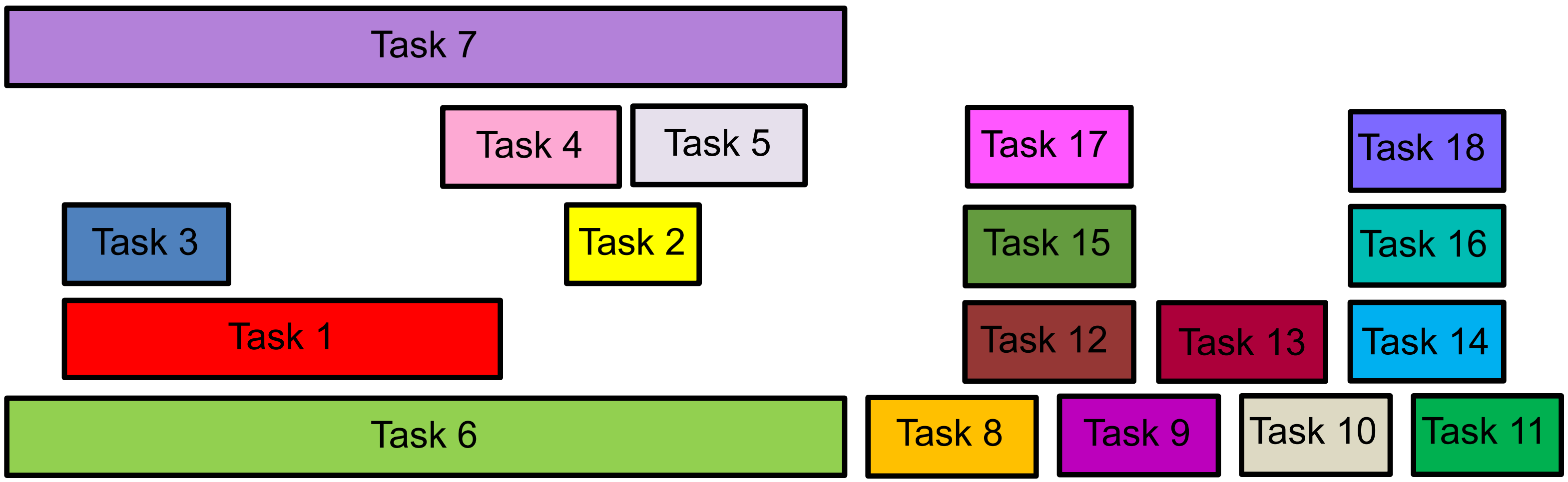
Choose i in R that has smallest number of conflicts

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

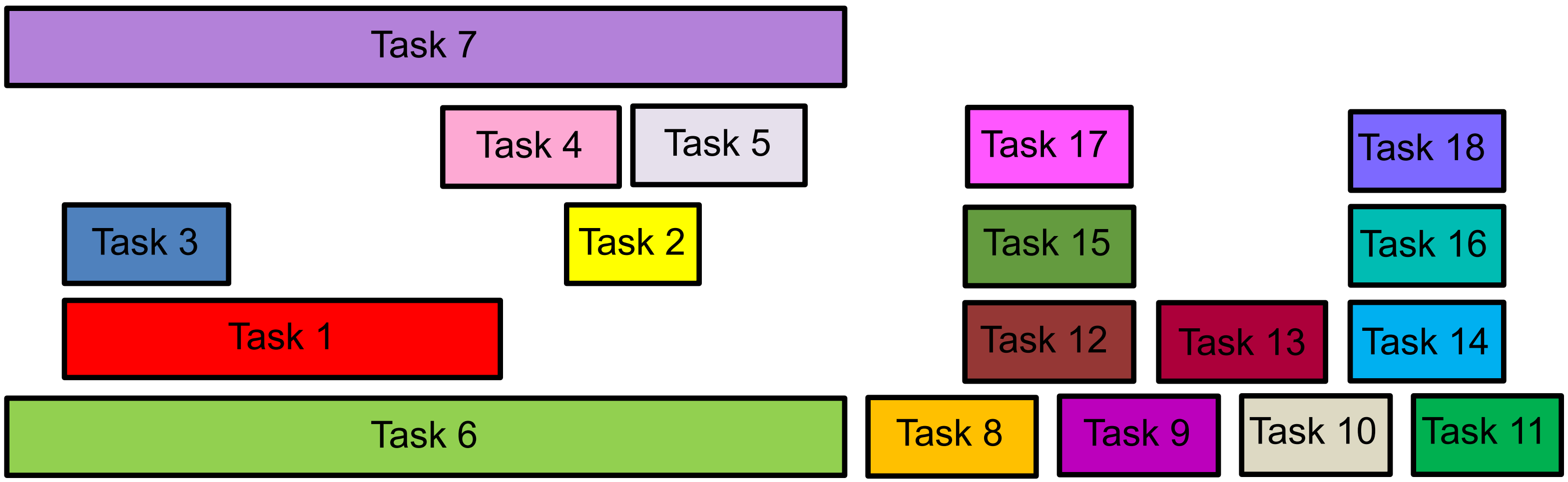
    Choose i in R that minimizes v(i)

    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

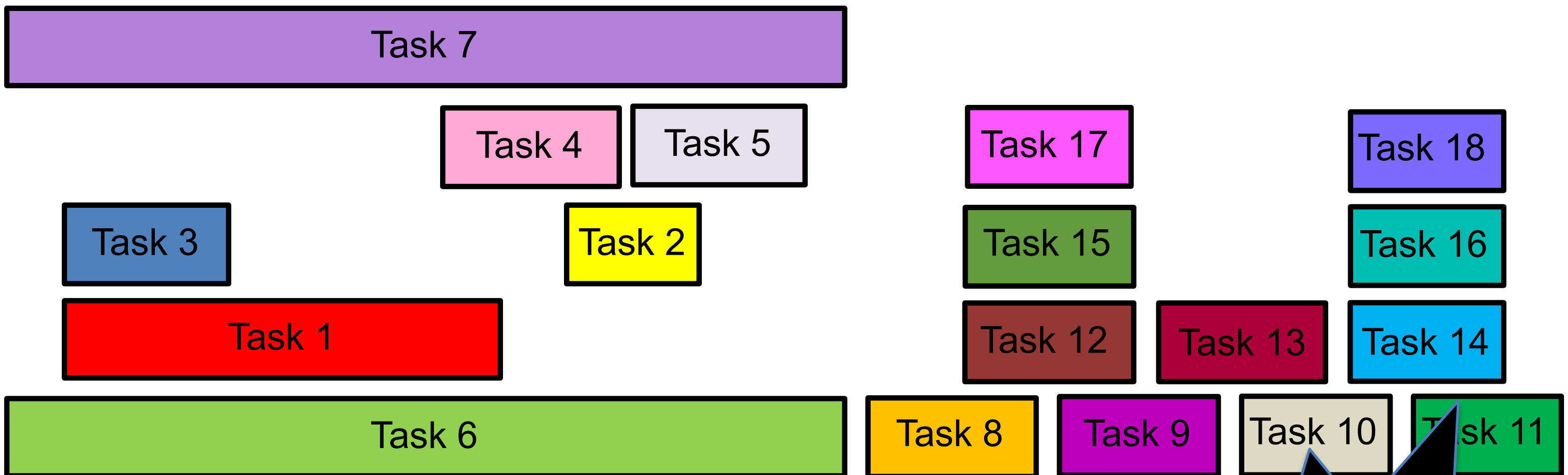
Choose i in R that minimizes f(i)

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 $S^* = S$