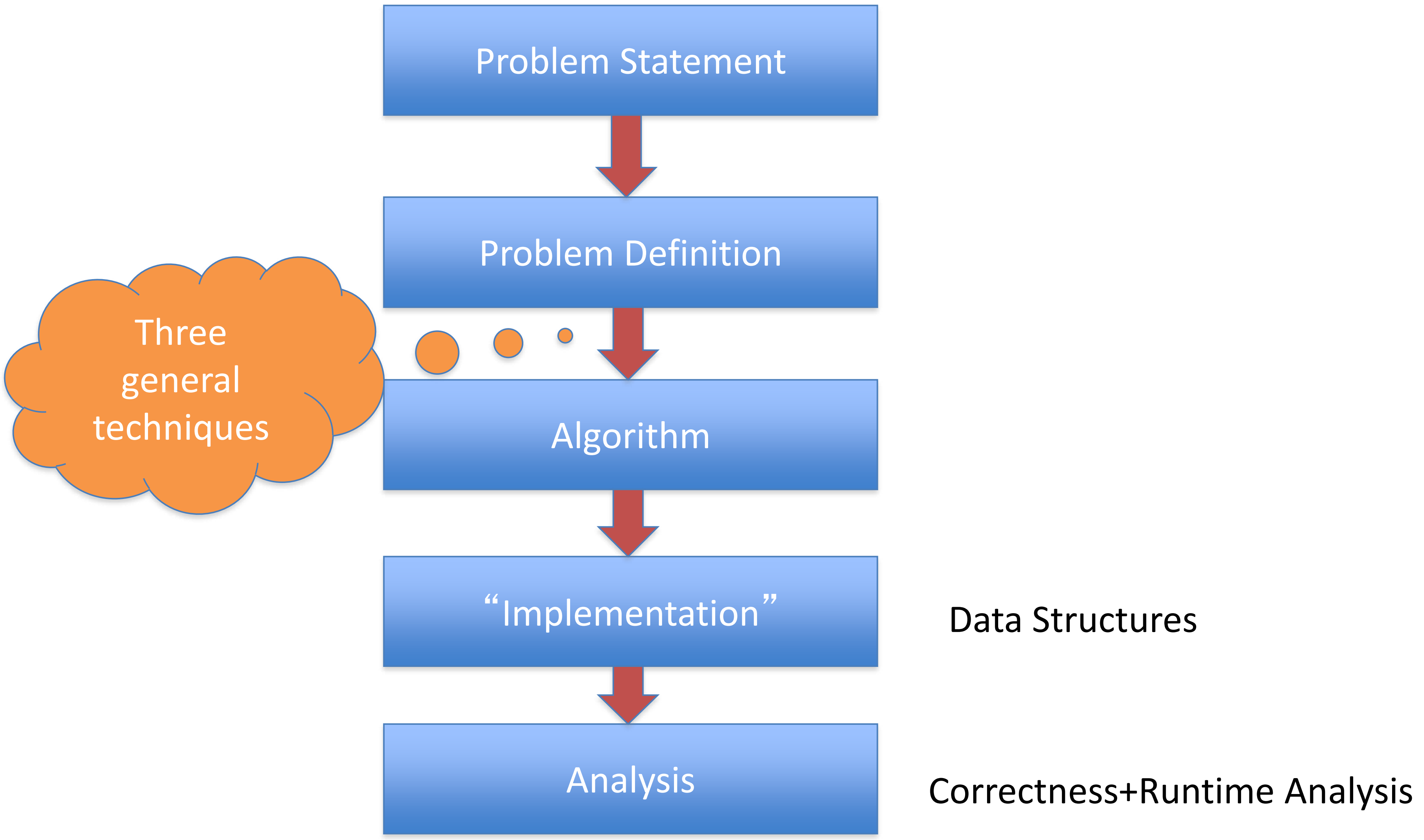
Lecture 29

CSE 331

High level view of CSE 331



Greedy Algorithms

Natural algorithms

Reduced exponential running time to polynomial

Divide and Conquer

Recursive algorithmic paradigm

Reduced large polynomial time to smaller polynomial time

A new algorithmic technique

Dynamic Programming

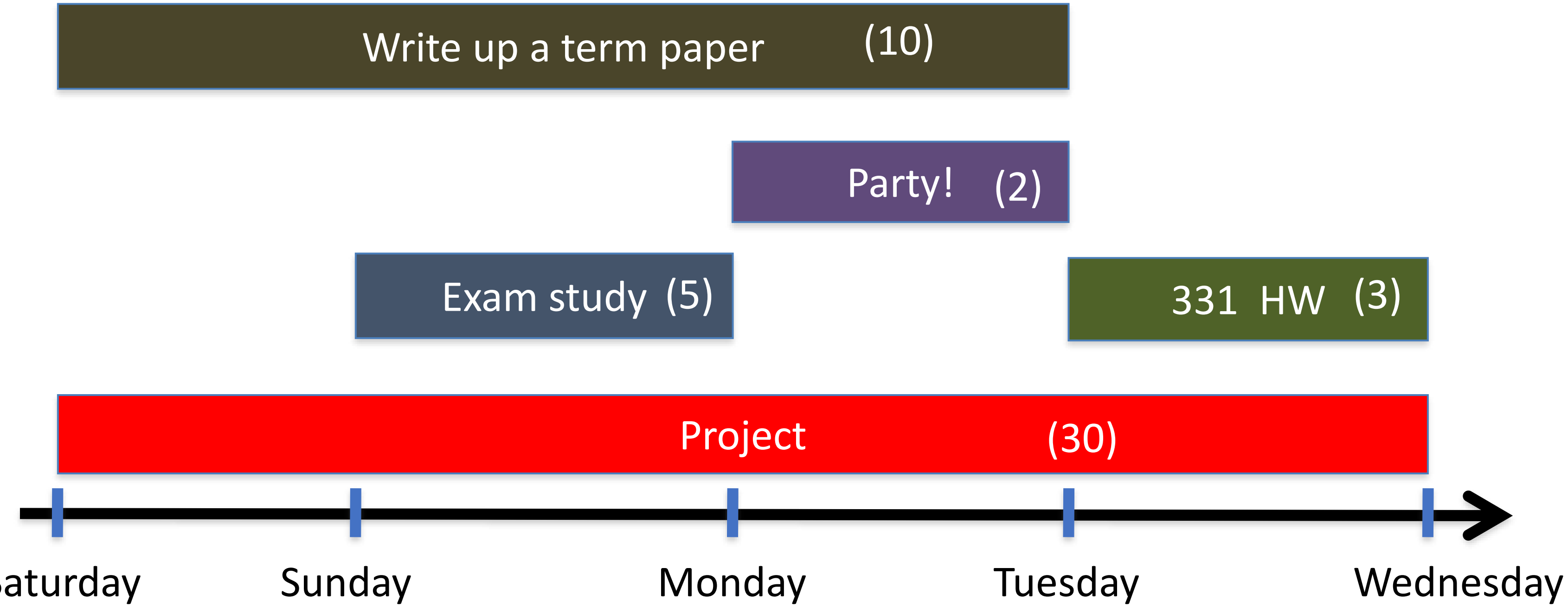
Dynamic programming vs. Divide & Conquer

Both design recursive algorithms

Dynamic programming is smarter about solving recursive sub-problems

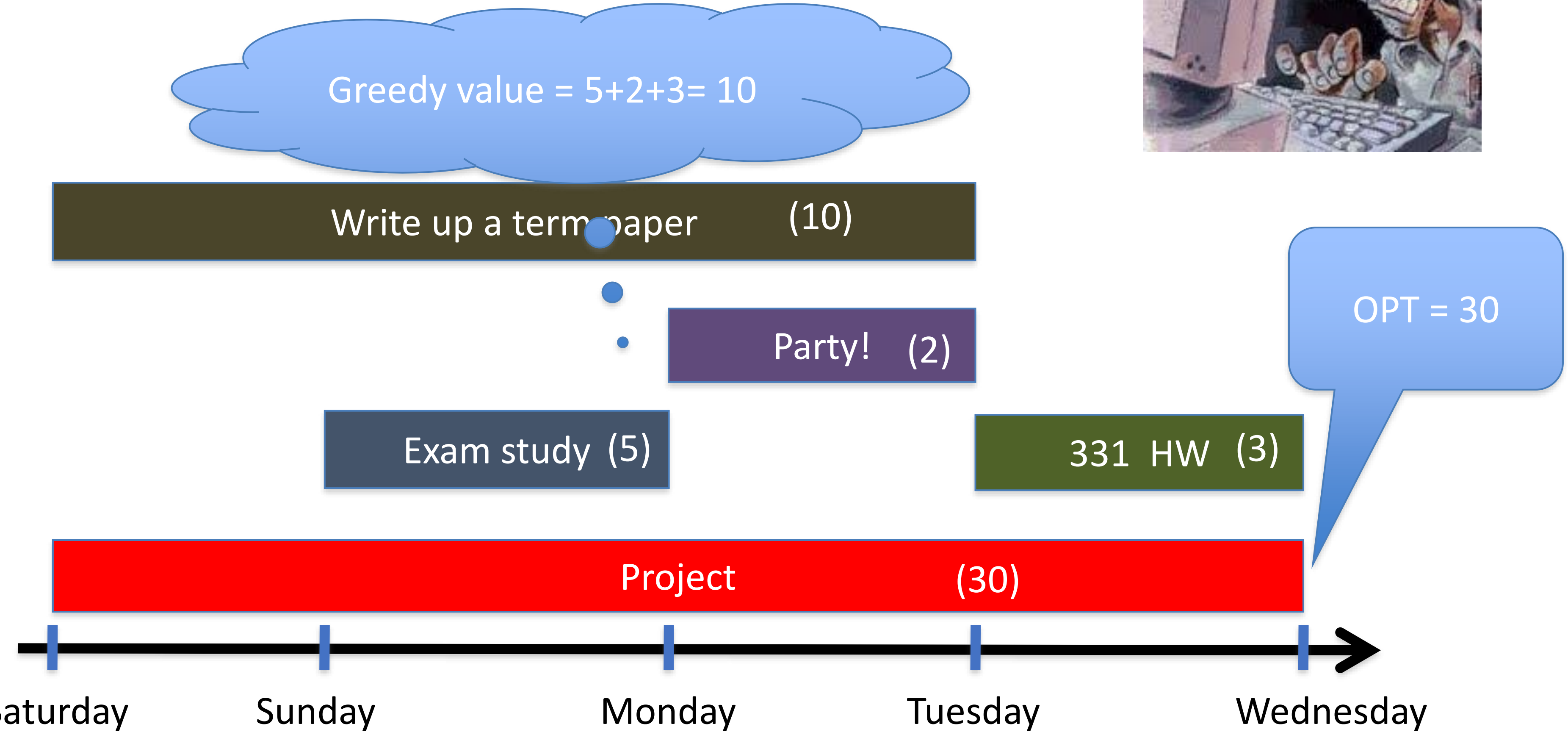
End of Semester blues

Can only do one thing at any day: what is the optimal schedule to obtain maximum value?



Previous Greedy algorithm

Order by end time and pick jobs greedily



Today’s agenda

Formal definition of the problem

Start designing a recursive algorithm for the problem

Weighted Interval Scheduling

Input: n jobs/intervals. Interval i is triple (s_i, f_i, v_i)

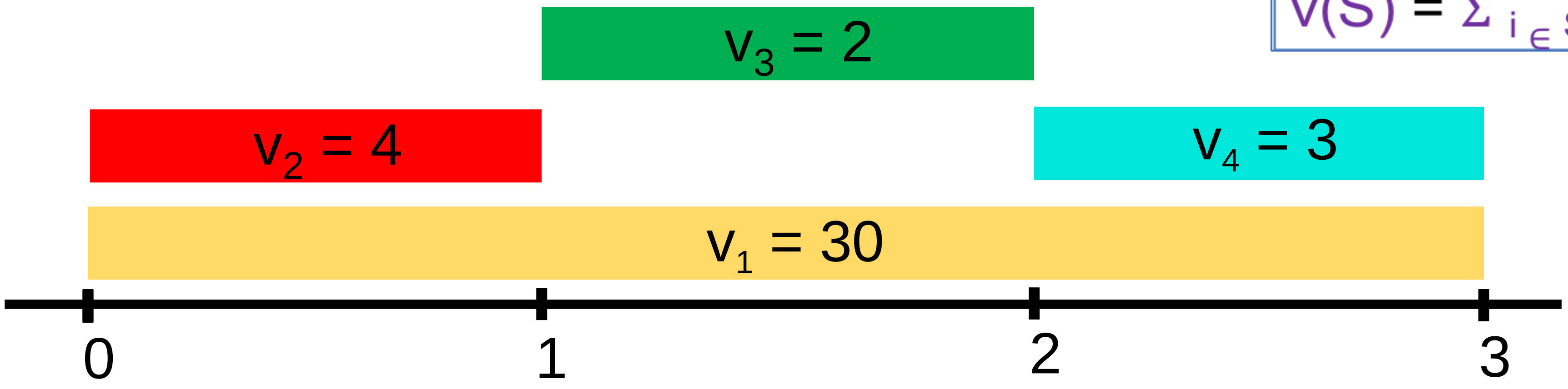
start time

finish time

value

Output: A valid schedule $S \subseteq [n]$ that maximizes $v(S)$

$v(S) = \Sigma_{i \in S} v_i$



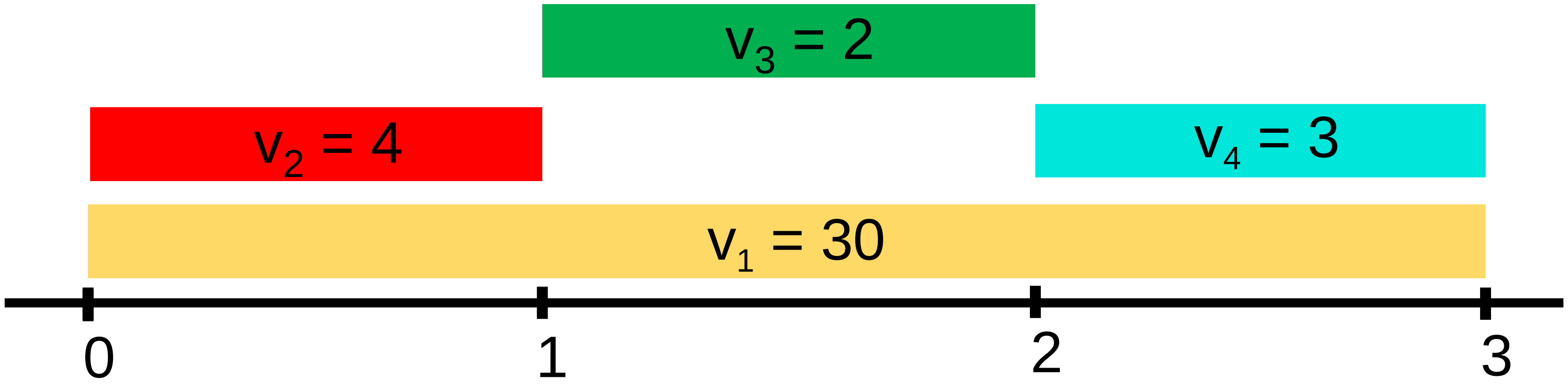
Previous Greedy Algorithm

R = original set of jobs

S = ϕ

While R is not empty
    Choose i in R where f_i is the smallest
    Add i to S
    Remove all requests that conflict with i from R

Return S* = S



Perhaps be greedy differently?

R = original set of jobs

$S = \phi$

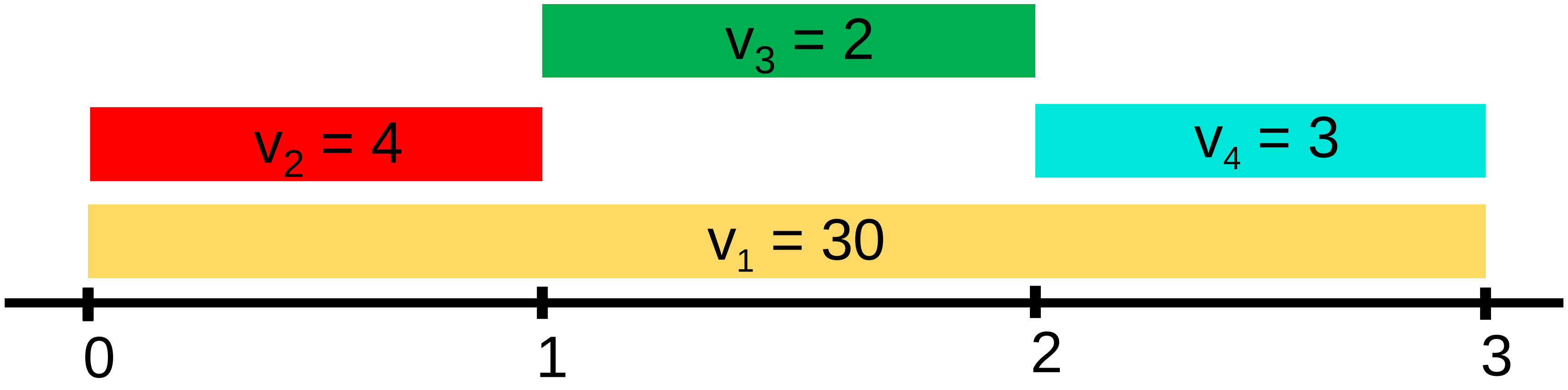
While R is not empty

  Choose i in R where $v_i/(f_i - s_i)$ is the largest

  Add i to S

  Remove all requests that conflict with i from R

Return $S^* = S$



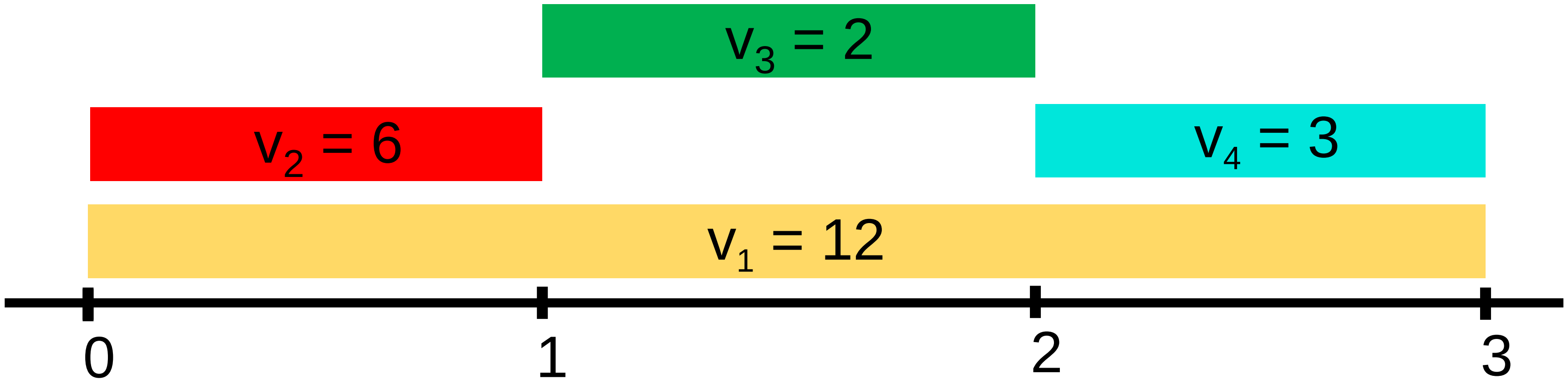
Can this work?

R = original set of jobs

S = ϕ

While R is not empty
    Choose i in R where $v_i/(f_i - s_i)$ is the largest
    Add i to S
    Remove all requests that conflict with i from R

Return S* = S



Avoiding the greedy rabbit hole

https://www.writerightwords.com/down-the-rabbit-hole/

Provably IMPOSSIBLE for a large class of greedy algos

There are no known greedy algorithm to solve this problem

Perhaps a divide & conquer algo?

Divide the problem in 2 or more many EQUAL SIZED INDEPENDENT problems

Recursively solve the sub-problems

Patchup the SOLUTIONS to the sub-problems

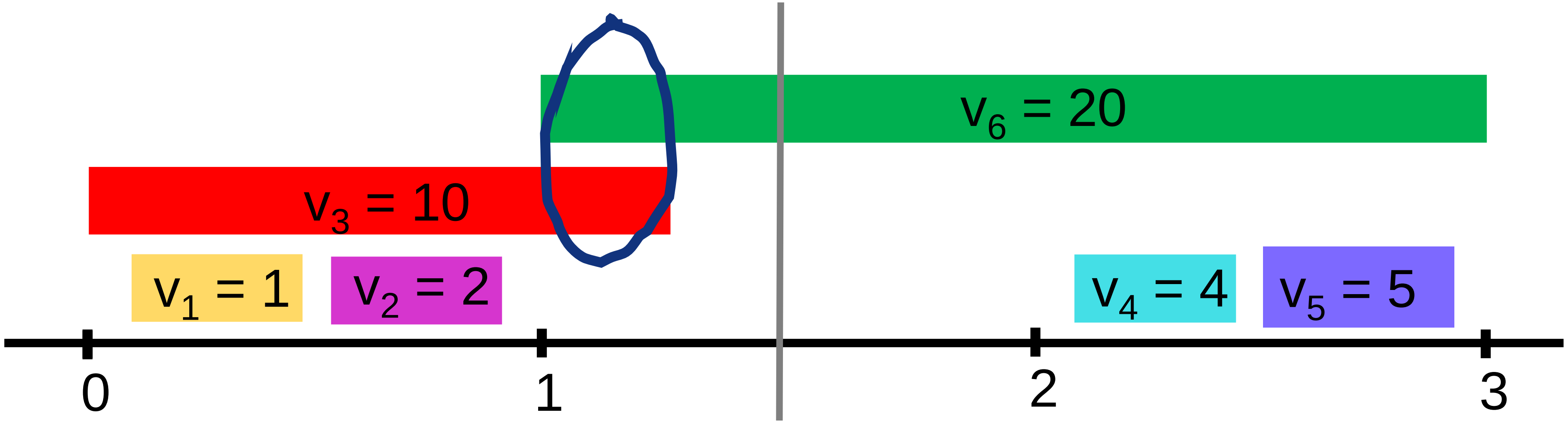
Perhaps a divide & conquer algo?

```
RecurWeightedInt([n])

    if n = 1 return the only interval

    L = first n/2 intervals
    R = last n/2 intervals

    S_L = RecurWeightedInt(L)

    S_R = RecurWeightedInt(R)

    PatchUp(S_L, S_R)
```

Would this general scheme work?

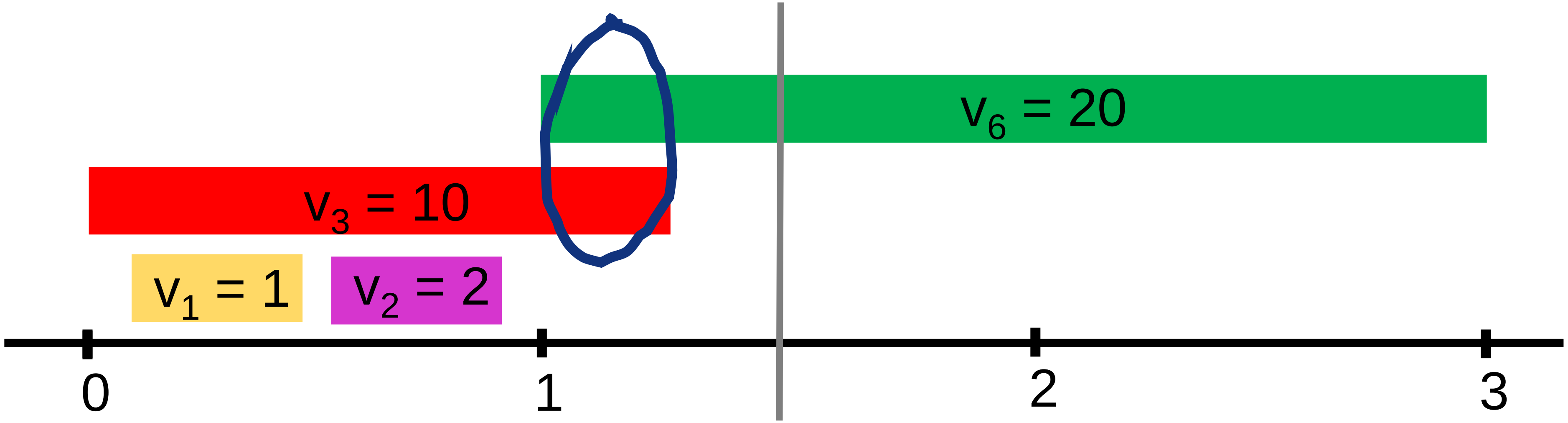
Divide the problem in 2 or more many EQUAL SIZED INDEPENDENT problems

Sub-problems NOT independent!

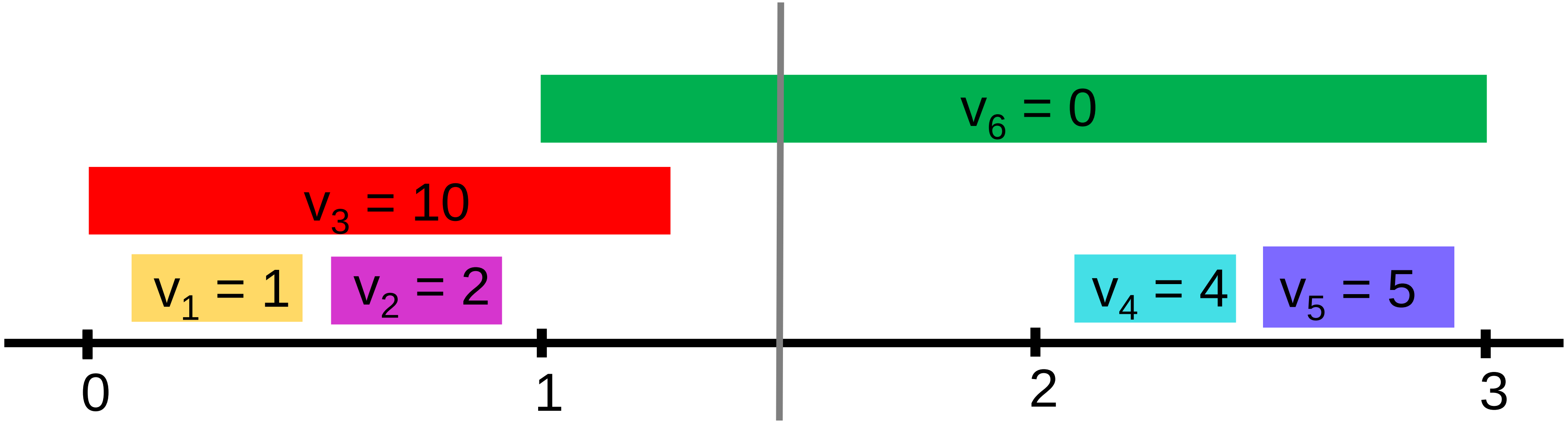


Perhaps patchup can help?

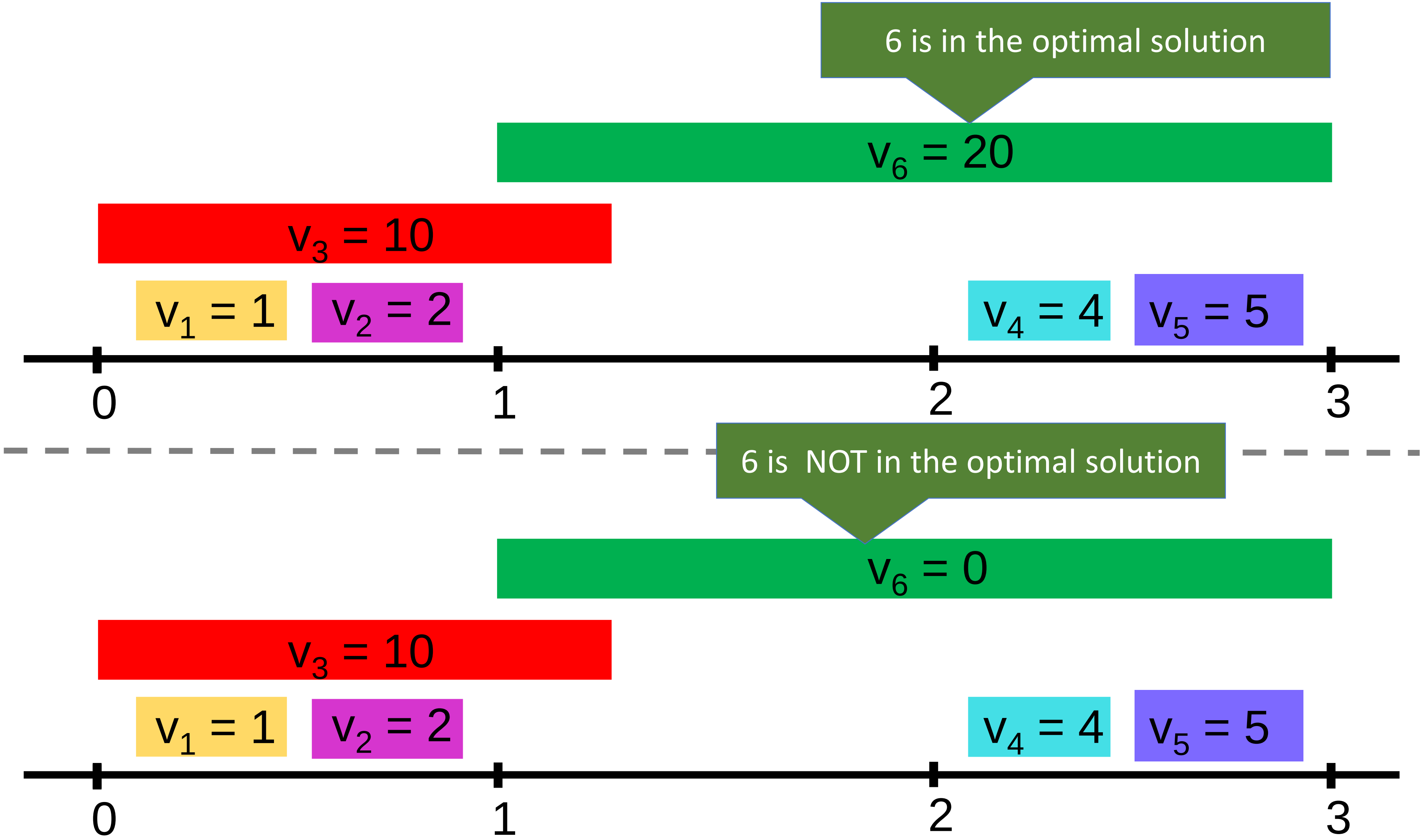
Patchup the SOLUTIONS to the sub-problems



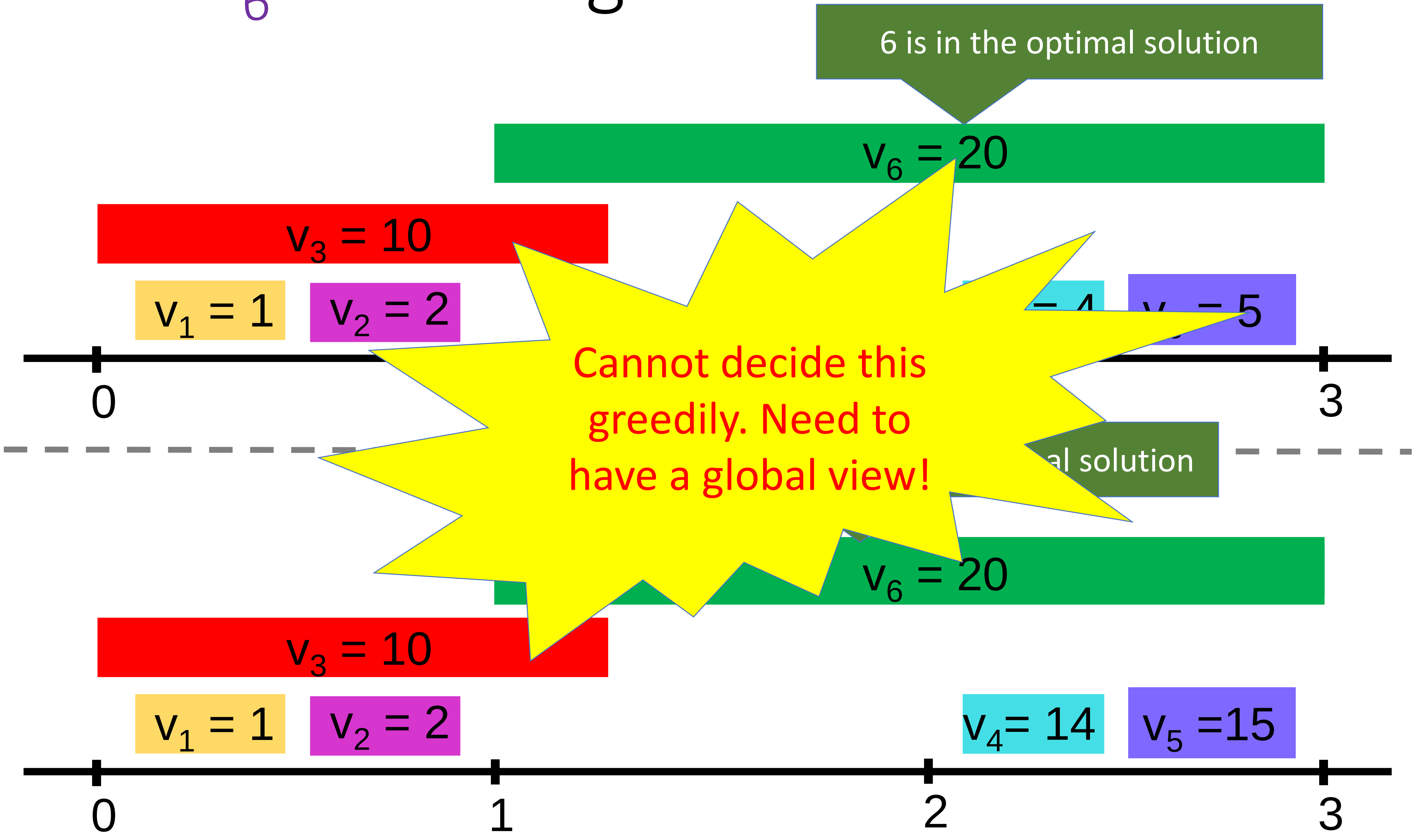
Sometimes patchup NOT needed!



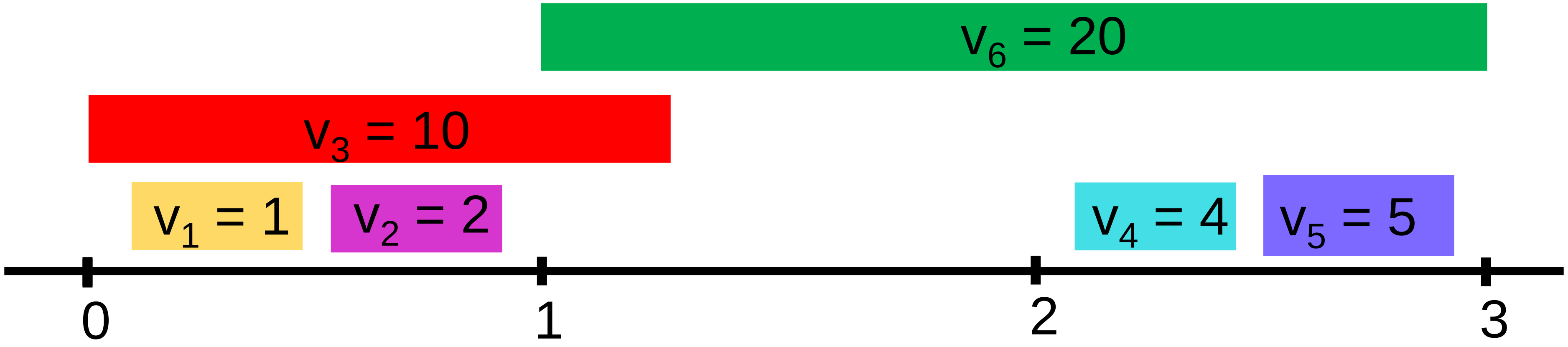
Check for two cases?



Check if v_6 is the largest value?

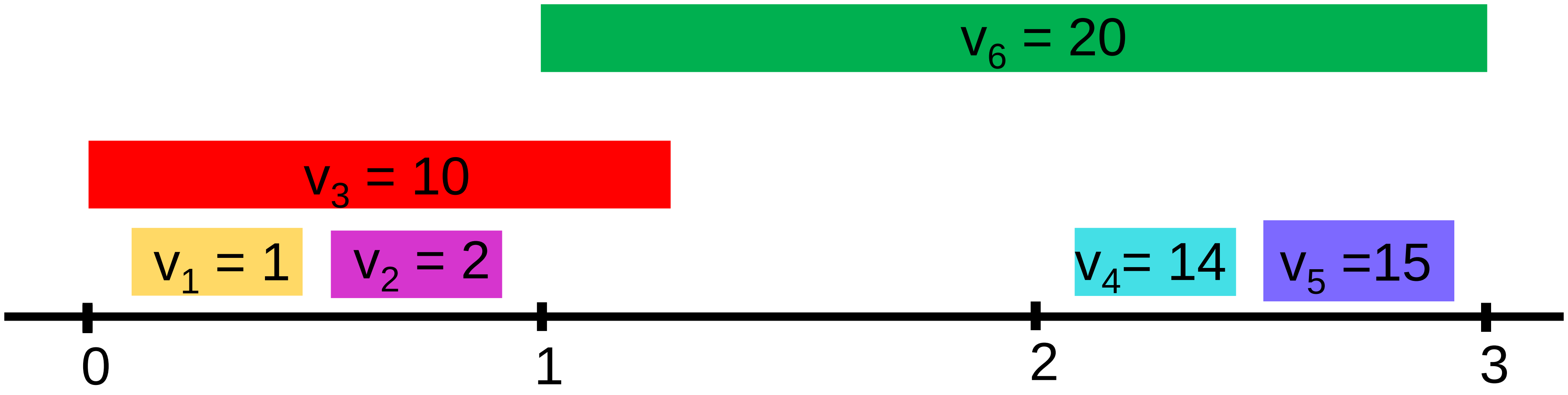


Check out both options!



Case 1: 6 is in the optimal solution

6 is not in optimal solution



So what sub-problems?

Divide the problem in 2 or more many ~~EQUAL SIZED~~ ~~INDEPENDENT~~ problems