POSCAT Seminar 6 : Greedy Approach

yougatup @ POSCAT

Topic

- Topic today
  - Greedy Approach
    - Basic Concept
    - Interval Scheduling
    - Interval Partitioning
    - Fractional Knapsack
    - Huffman Encoding
    - Other Problems

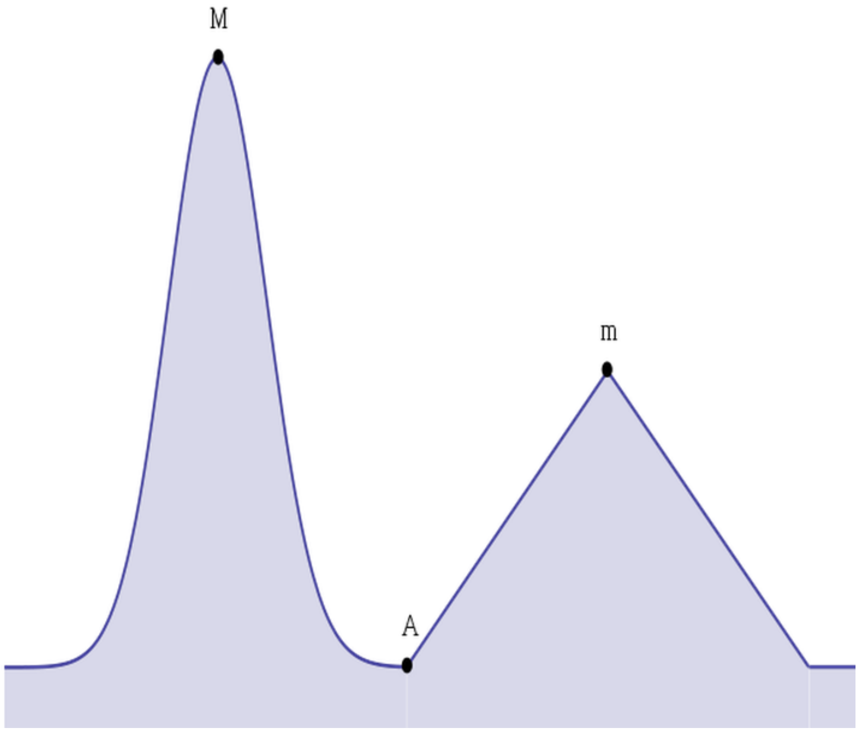
Poscat

Greedy Approach

- Problem Solving Paradigm
  - Follows the problem solving heuristic of making the **locally optimal choose** at each stage
  - You should **prove** that greedy approach guarantees **the global optimal**

- Very difficult normally
  - Finding a **solution** is very **difficult**, but
  - **Code** is very **simple** because it choose locally optimal usually

Greedy Approach



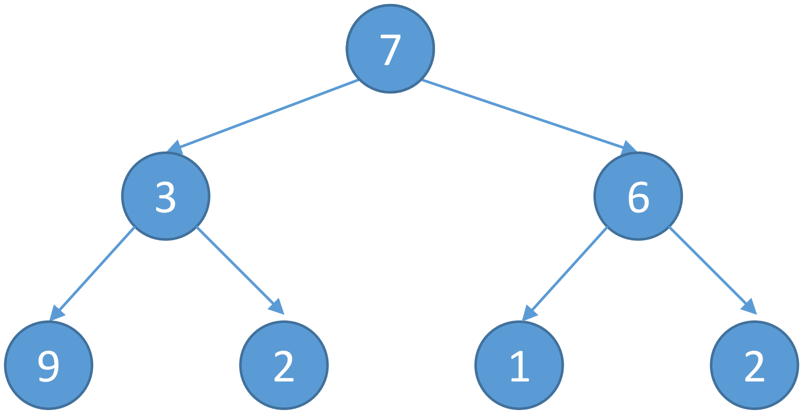
Starting at A, our greedy algorithm should find '**M**', not 'm'.

Therefore, we must **prove** that greedy algorithm always find 'M', although we always choose **locally optimal path**

Greedy Approach

- Problem

Find a path which has largest sum

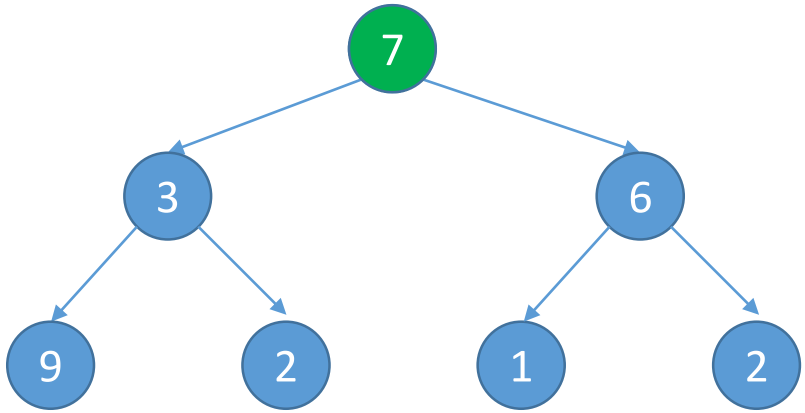


Greedy Approach

- Problem

Find a path which has largest sum

Greedy approach : Choose what appears to be the optimal

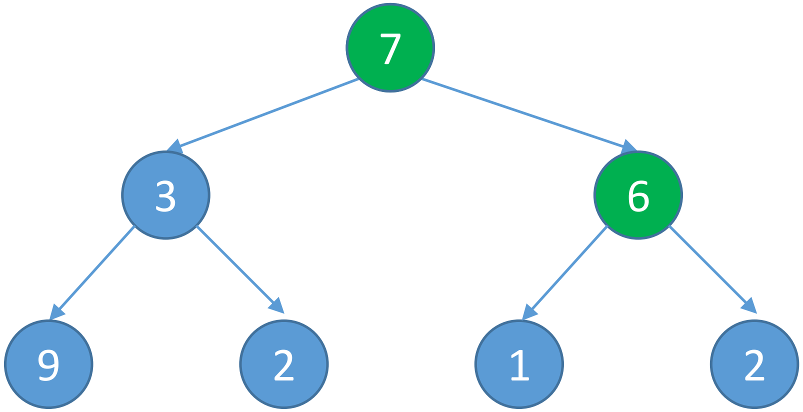


Greedy Approach

- Problem

Find a path which has largest sum

Greedy approach : Choose what appears to be the optimal

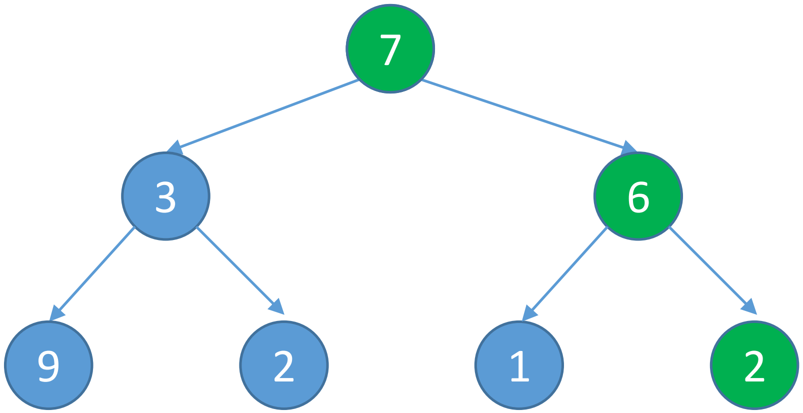


Greedy Approach

- Problem

Find a path which has largest sum

Greedy approach : Choose what appears to be the optimal



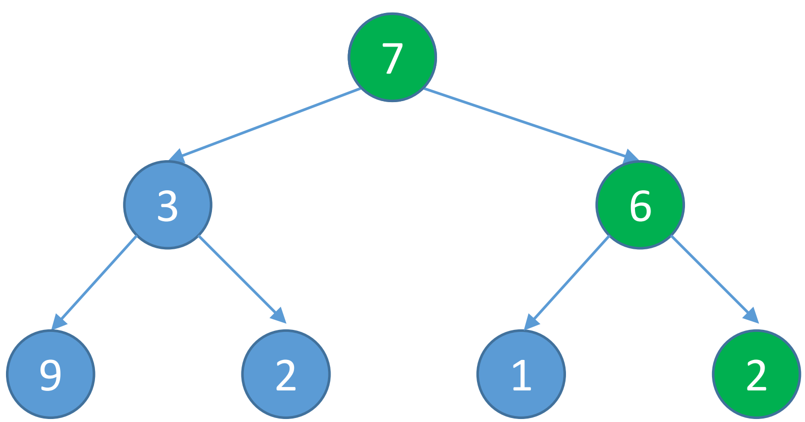
Greedy Approach

- Problem

Find a path which has largest sum

Greedy approach : Choose what appears to be the optimal

Is it our solution ?



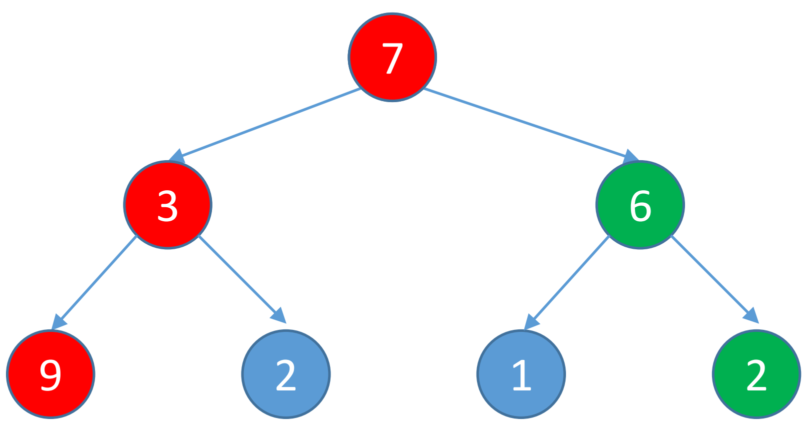
Greedy Approach

- Problem

Find a path which has largest sum

Greedy approach : Choose what appears to be the optimal

Is it our solution ?



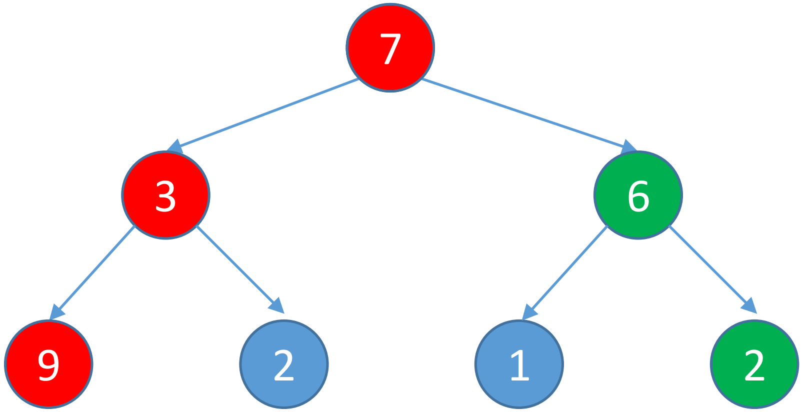
Greedy Approach

- Problem

Find a path which has largest sum

Greedy approach : Choose what appears to be the optimal

We **must prove** that our choice guarantees **global optimal value**

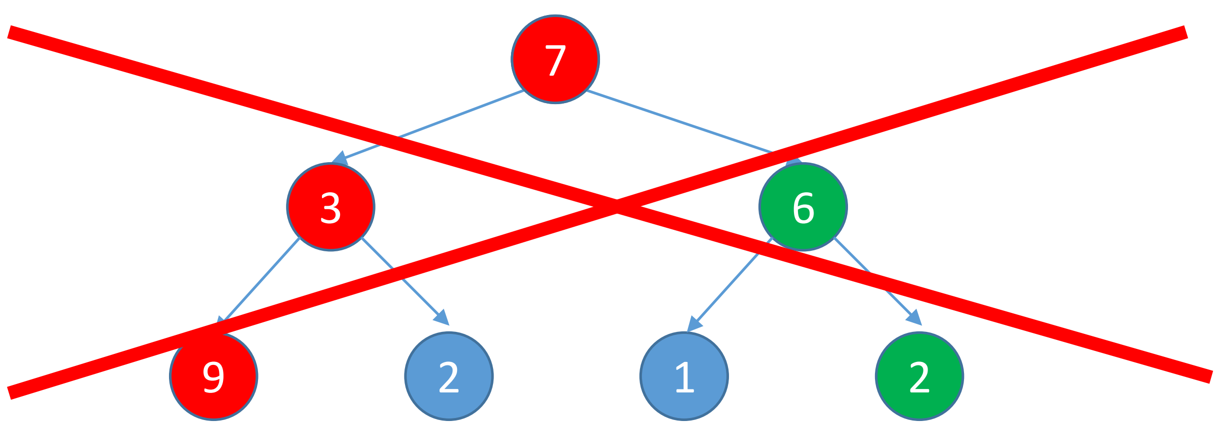


Greedy Approach

- Problem

Completely Wrong !

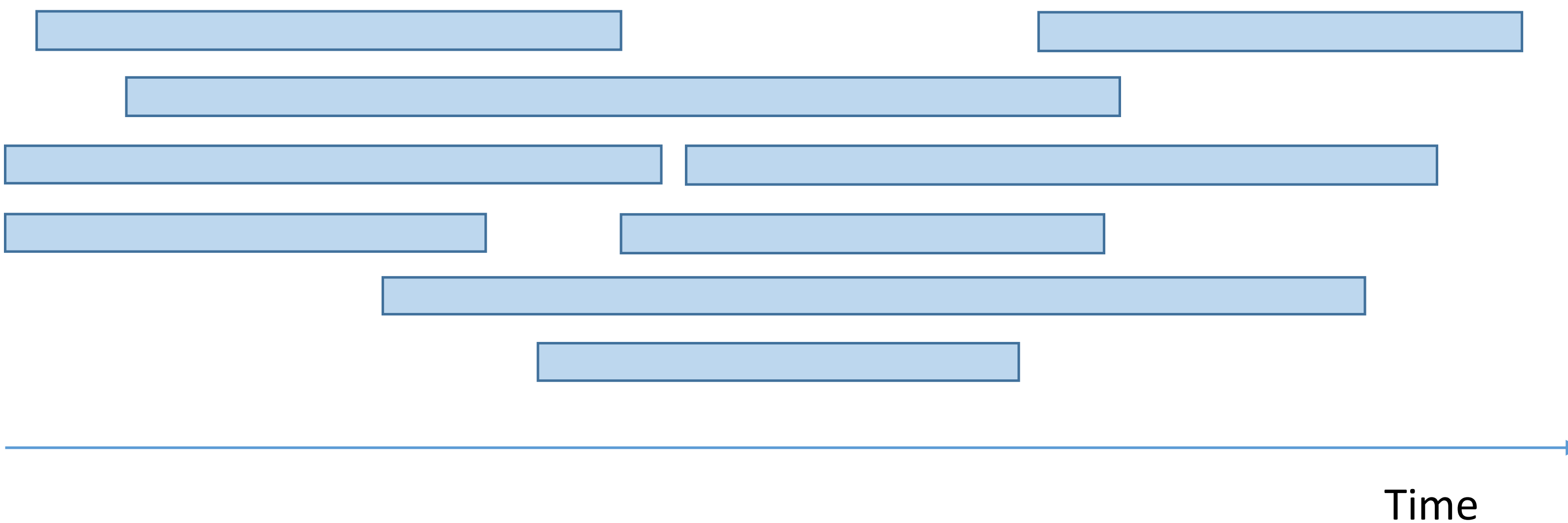
We **must prove** that our choice guarantees **global optimal value**



Poscat

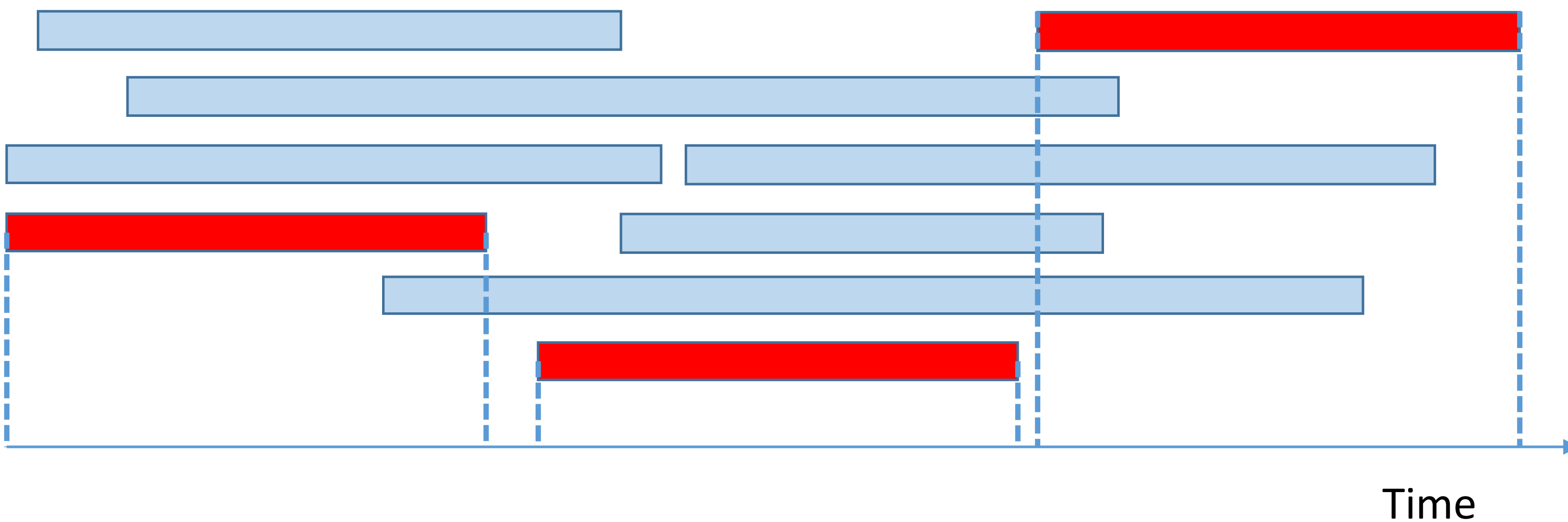
Interval Scheduling

- Problem
  - Job j starts at s_j and finishes at f_j
  - Two jobs **compatible** if they don't overlap
  - Find maximum subset of **mutually compatible jobs**



Interval Scheduling

- Problem
  - Job j starts at s_j and finishes at f_j
  - Two jobs **compatible** if they don't overlap
  - Find maximum subset of **mutually compatible jobs**

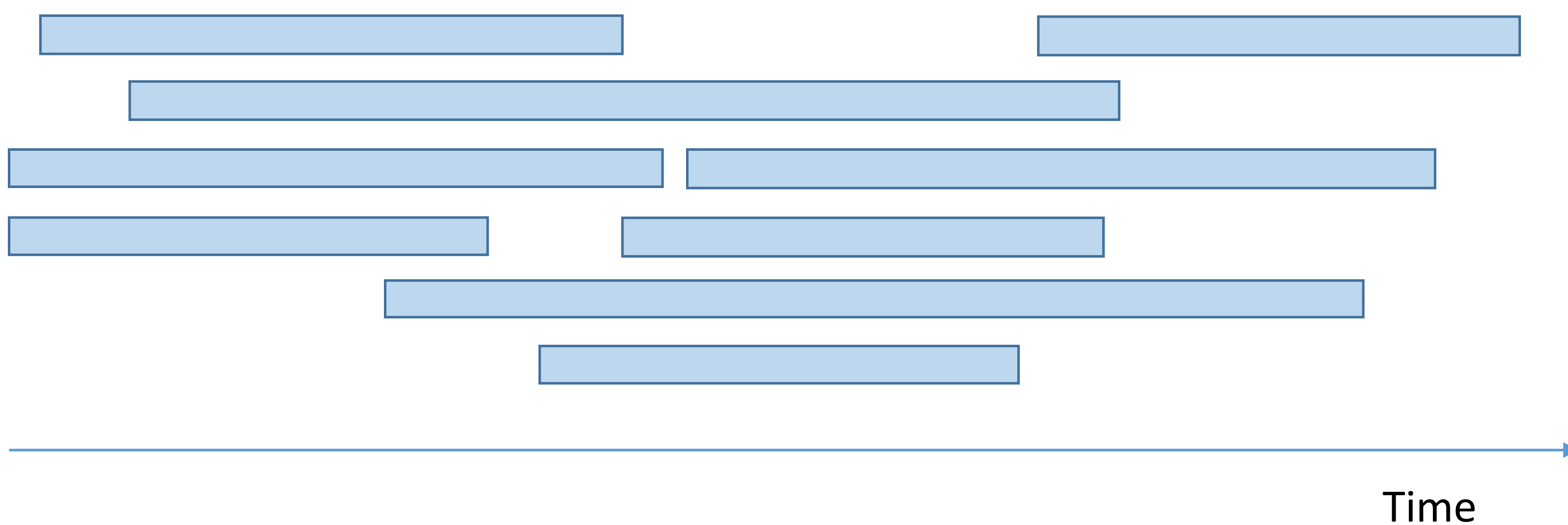


They don't overlap each other.

Interval Scheduling

- Problem

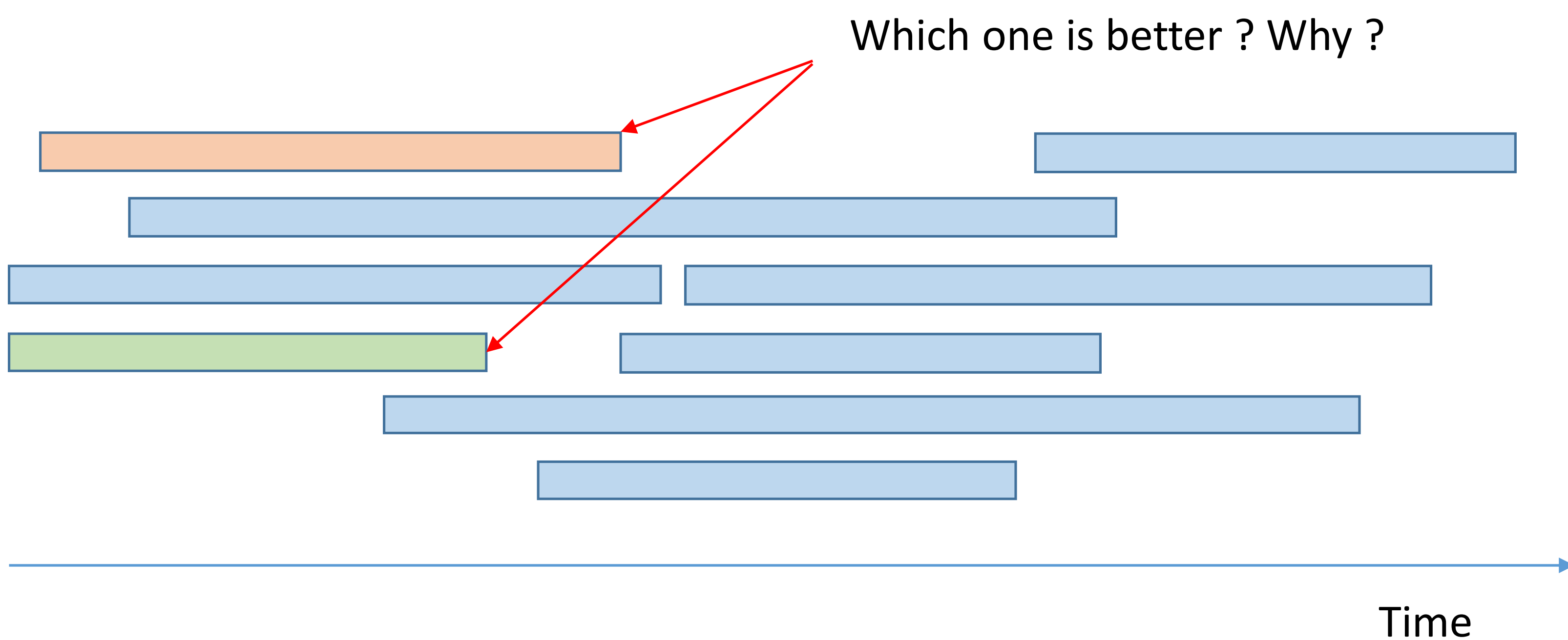
Idea : What have to be the first interval ?



Interval Scheduling

- Problem

Idea : What have to be the first interval ?

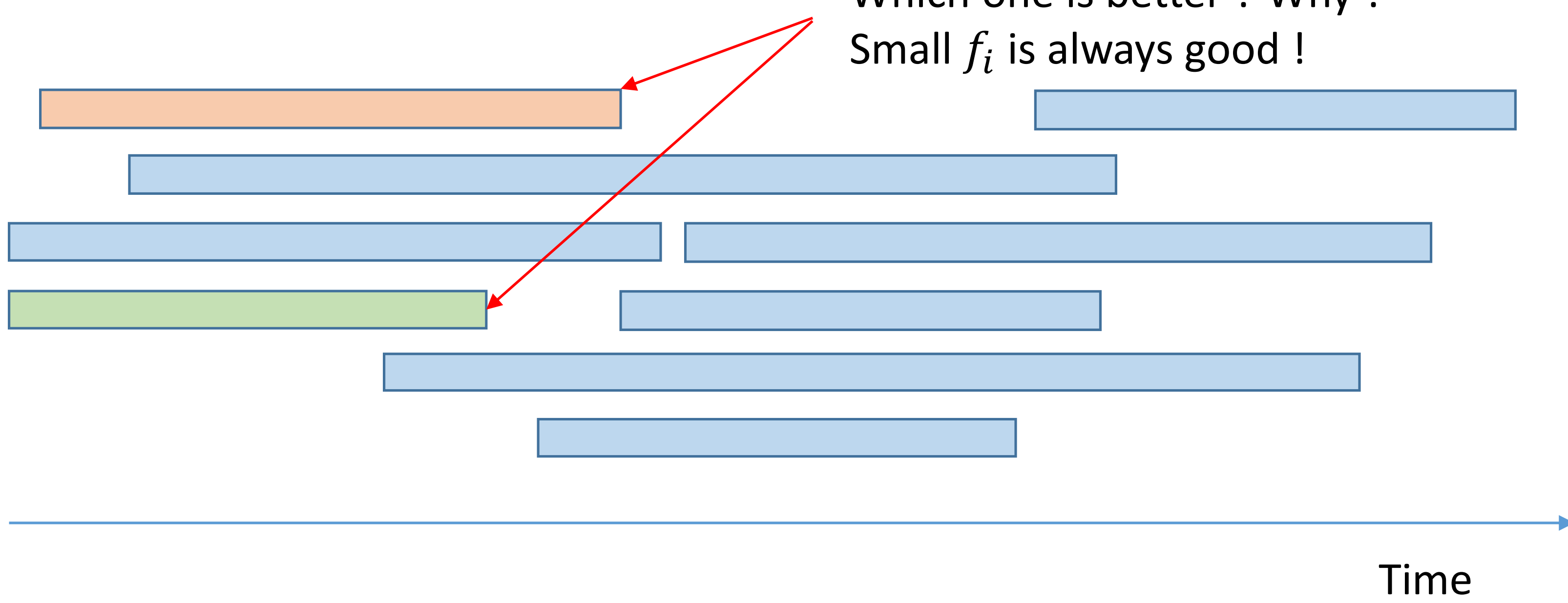


Interval Scheduling

- Problem

Idea : What have to be the first interval ?

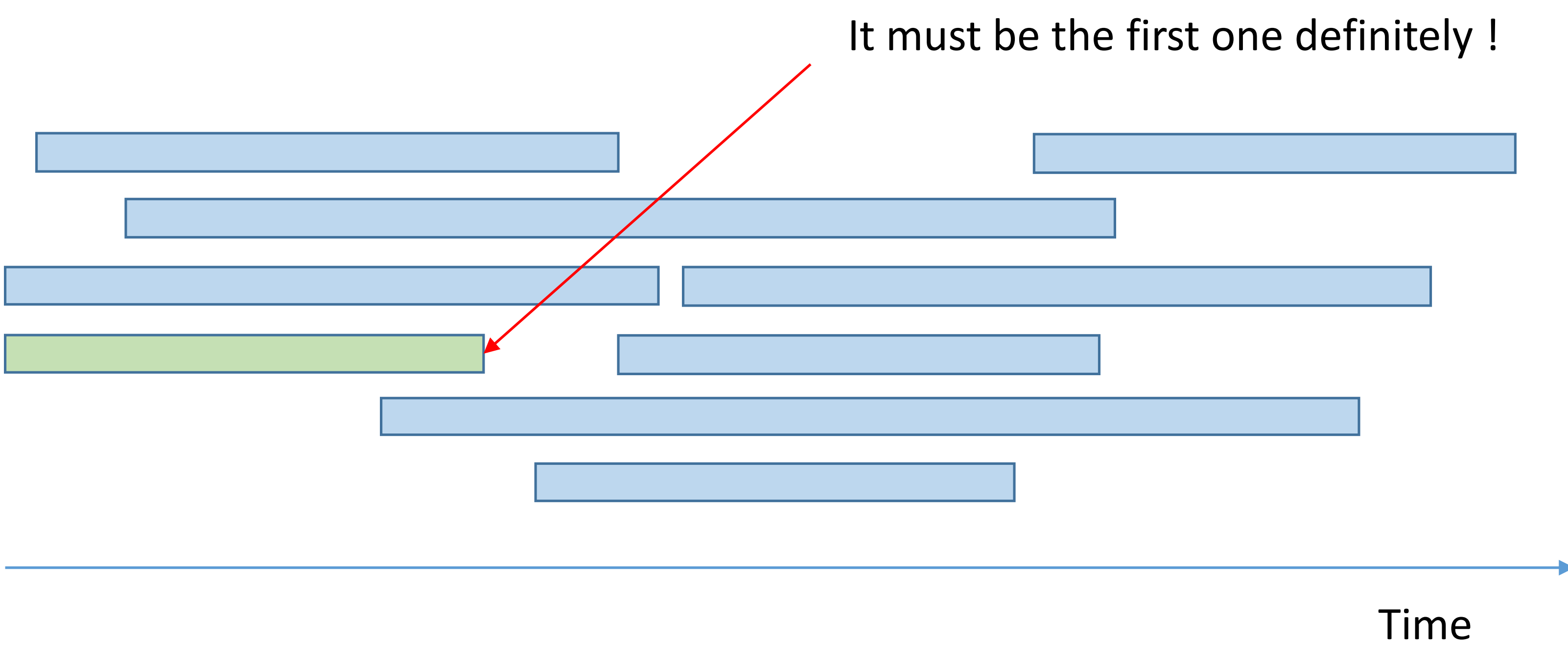
Which one is better ? Why ?
Small f_i is always good !



Interval Scheduling

- Problem

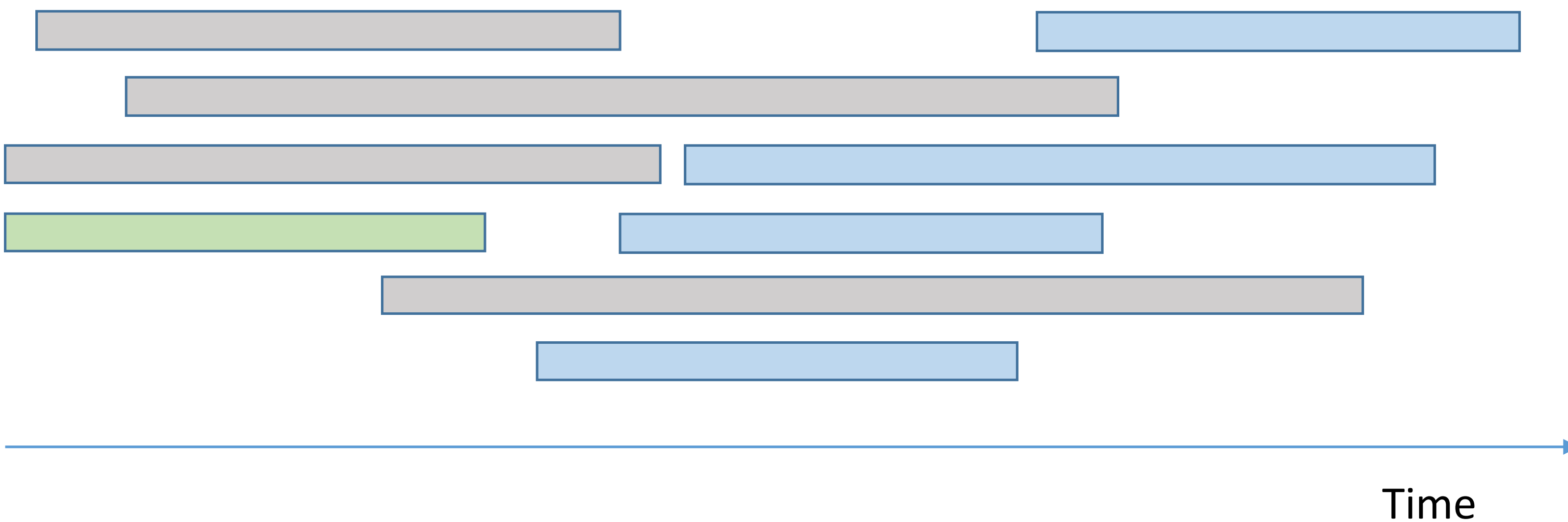
Idea : What have to be the first interval ?



Interval Scheduling

- Problem

Idea : What have to be the first interval ?

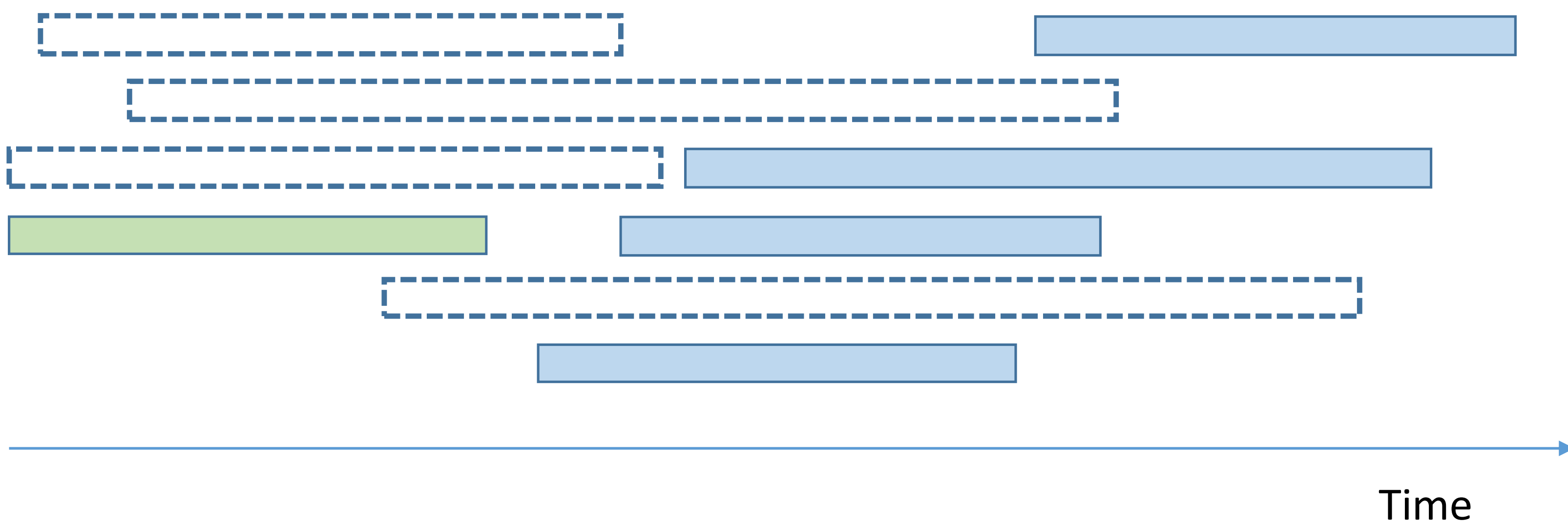


We can't choose grey interval → remove it

Interval Scheduling

- Problem

Idea : What have to be the first interval ?

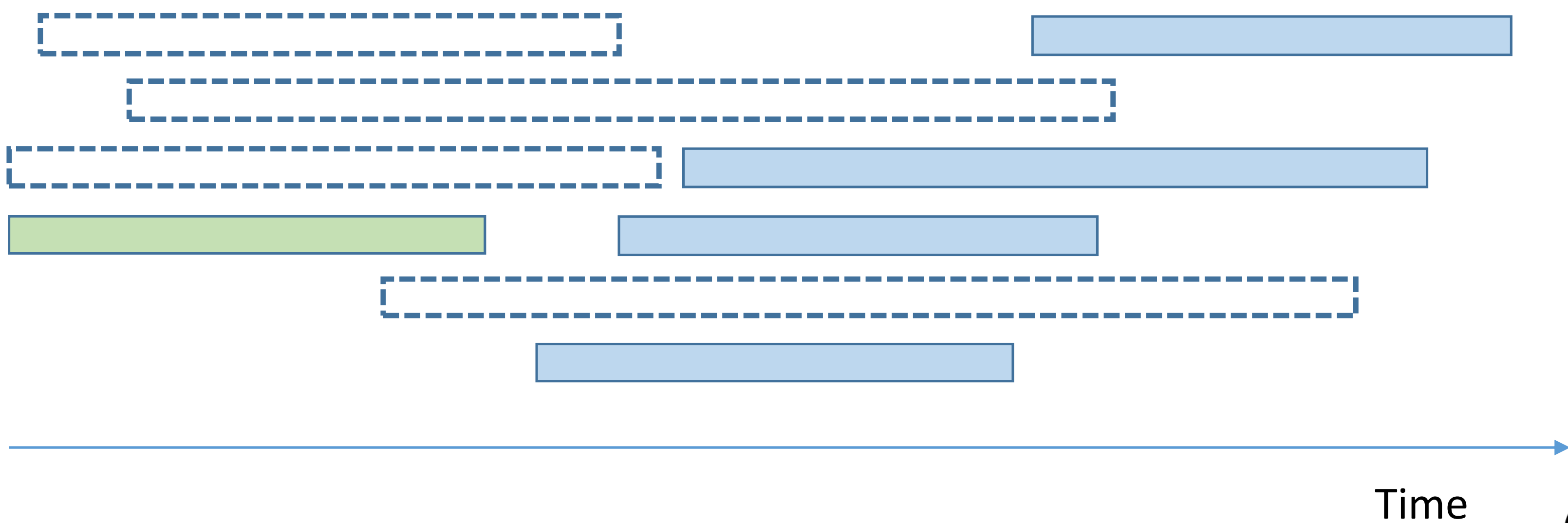


We can't choose grey interval → remove it

Interval Scheduling

- Problem

Idea : What have to be the first interval ?

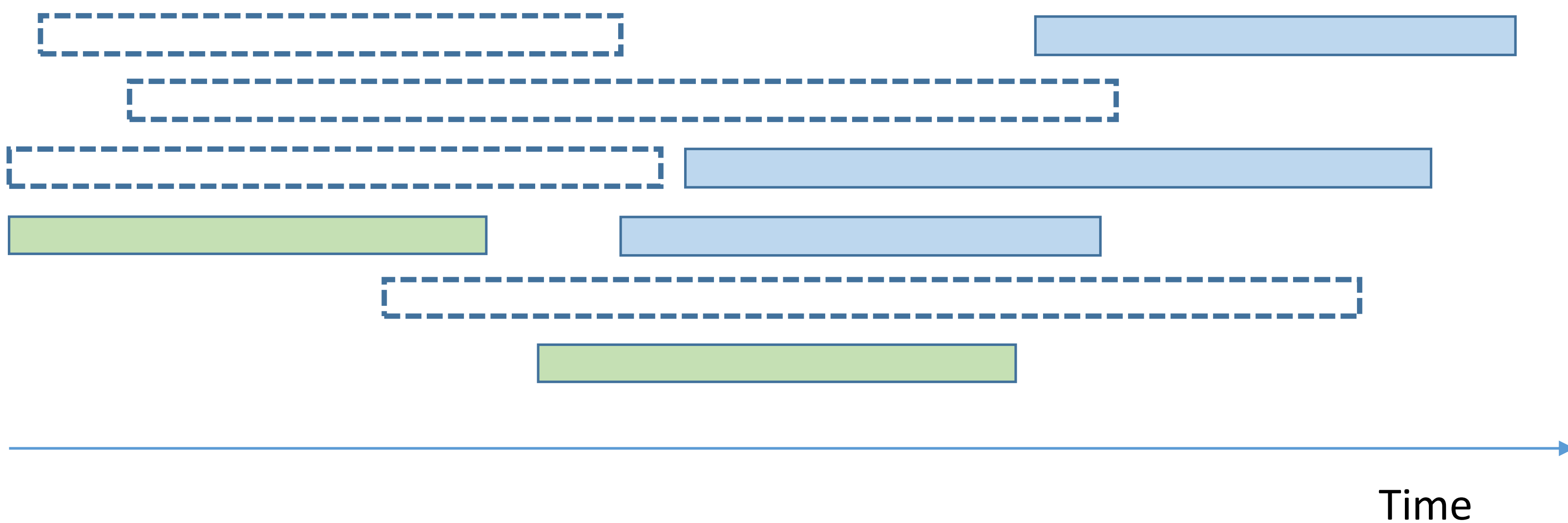


Which one is better ?

Interval Scheduling

- Problem

Idea : What have to be the first interval ?

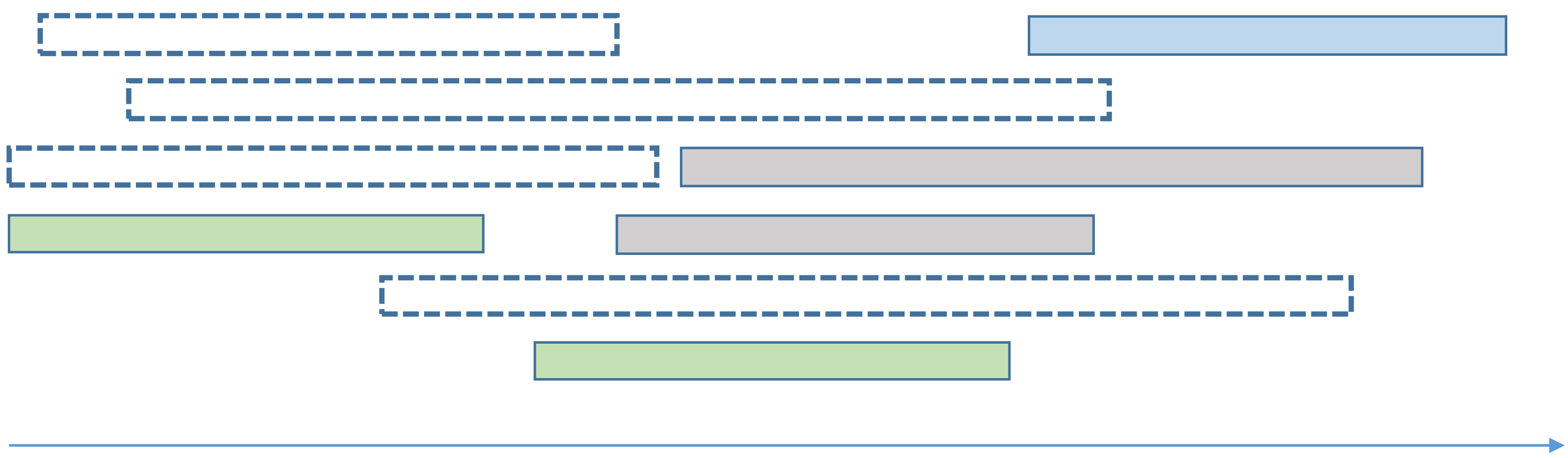


Which one is better ?

Interval Scheduling

- Problem

Idea : What have to be the first interval ?

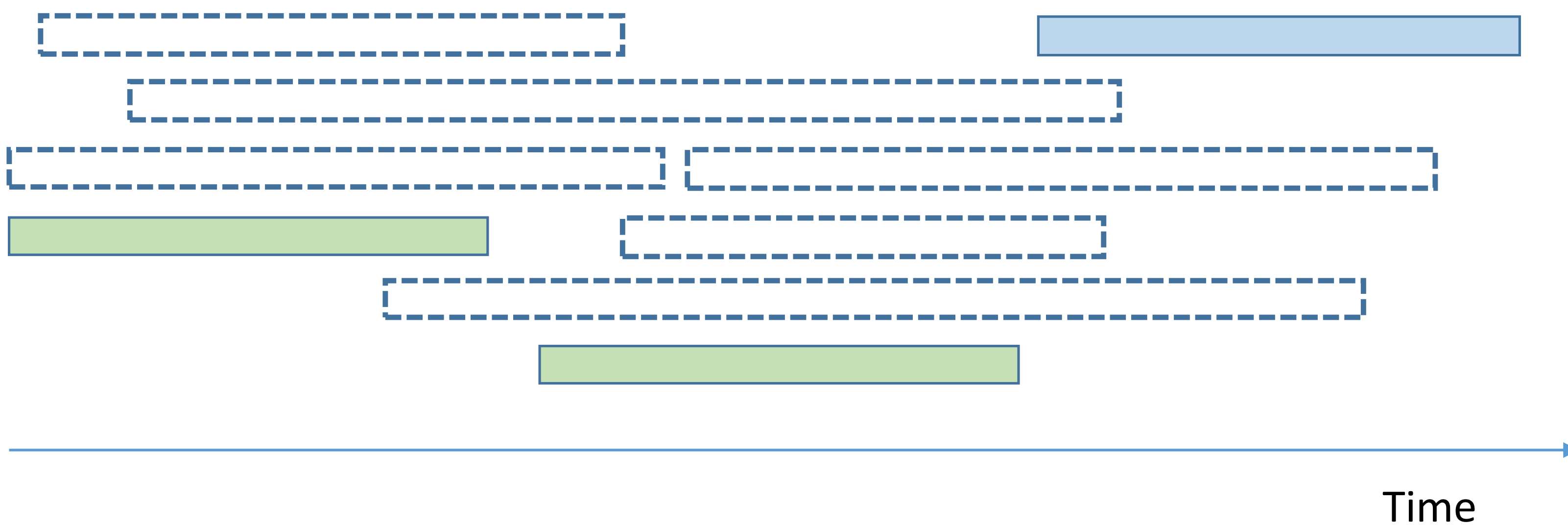


Which one is better ?

Interval Scheduling

- Problem

Idea : What have to be the first interval ?

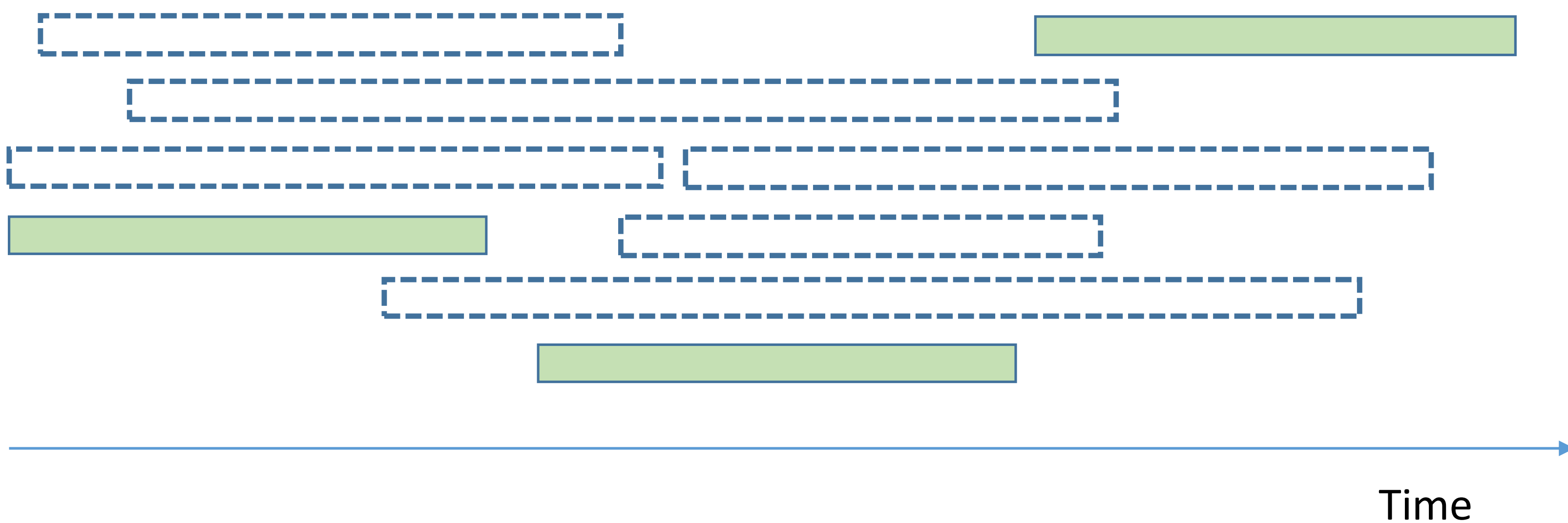


Which one is better ?

Interval Scheduling

- Problem

Idea : What have to be the first interval ?

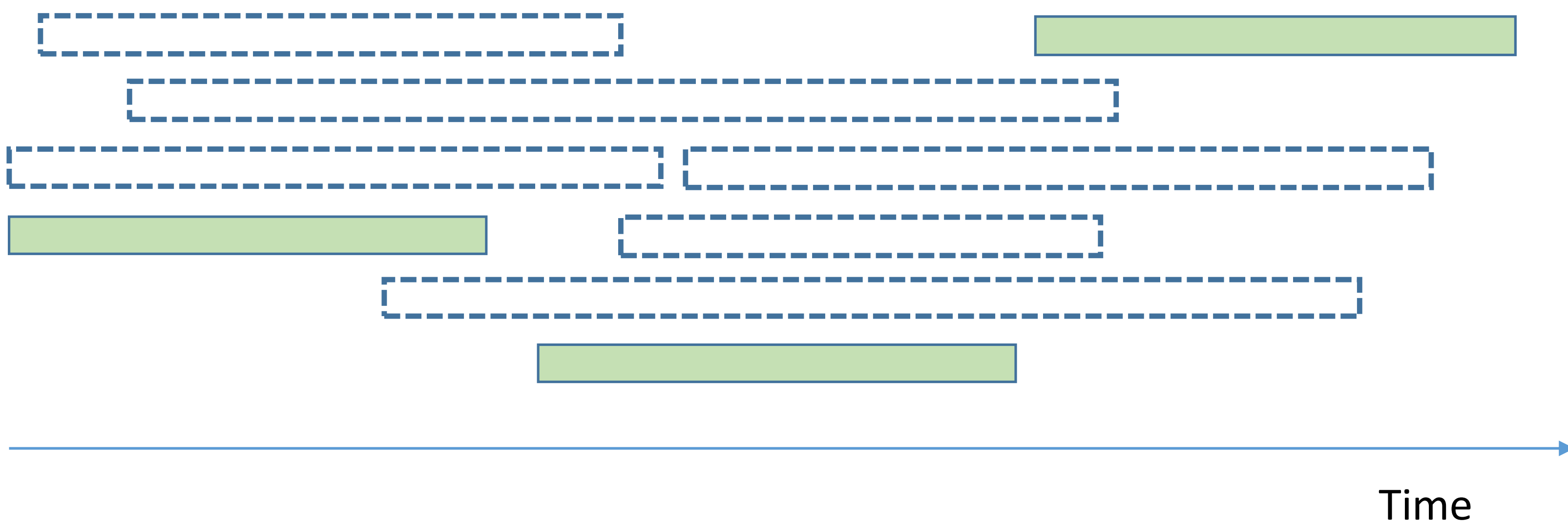


Which one is better ?

Interval Scheduling

- Problem

Idea : What have to be the first interval ?



Done ! You should know why greedy algorithm guarantees the global optimal

Interval Scheduling

- Problem

Prove that this algorithm is optimal

Done ! You should know why greedy algorithm guarantees the global optimal

Interval Scheduling

- Problem

Prove that this algorithm is optimal

Suppose that there is a solution $s_1 \le f_1 \le s_2 \le f_2 \le \ \dots \le s_k \le f_k$,
and we found $s_1' \le f_1' \le s_2' \le f_2' \le \ \dots \le s_{opt}' \le f_{opt}{}'$

Done ! You should know why greedy algorithm guarantees the global optimal

Interval Scheduling

- Problem

Prove that this algorithm is optimal

Suppose that there is a solution $s_1 \leq f_1 \leq s_2 \leq f_2 \leq \ \dots \leq s_k \leq f_k$,
and we found $s_1' \leq f_1' \leq s_2' \leq f_2' \leq \ \dots \leq s_{opt}' \leq f_{opt}'$

We select the interval whose finish time is minimal.

$$\therefore \ f_1' \leq f_1.$$

Done ! You should know why greedy algorithm guarantees the global optimal

Interval Scheduling

- Problem

Prove that this algorithm is optimal

Suppose that there is a solution $s_1 \le f_1 \le s_2 \le f_2 \le \ \dots \le s_k \le f_k$,
and we found $s_1' \le f_1' \le s_2' \le f_2' \le \ \dots \le s_{opt}' \le f_{opt}'$

Also, we choose the second interval whose finish time is smallest among intervals compatible with the first one.

$$\therefore f_2' \le f_2$$

Done ! You should know why greedy algorithm guarantees the global optimal

Interval Scheduling

- Problem

Prove that this algorithm is optimal

Suppose that there is a solution $s_1 \le f_1 \le s_2 \le f_2 \le \ \ldots \le s_k \le f_k$,
and we found $s_1' \le f_1' \le s_2' \le f_2' \le \ \ldots \le s_{opt}' \le f_{opt}{}'$

Like this, we can always guarantee that $f_k \le f_k'$. $\therefore\ opt \ge k$

Also, $opt \le k$ because k is solution

Done ! You should know why greedy algorithm guarantees the global optimal

Interval Scheduling

- Problem

Prove that this algorithm is optimal

Suppose that there is a solution $s_1 \leq f_1 \leq s_2 \leq f_2 \leq \ \dots \leq s_k \leq f_k,$
and we found $s_1' \leq f_1' \leq s_2' \leq f_2' \leq \ \dots \leq s_{opt}' \leq f_{opt}{}'$

Like this, we can always guarantee that $f_k \leq f_k'. \therefore \ opt \geq k$

Also, $opt \leq k$ because k is solution

Therefore, $opt = k.$

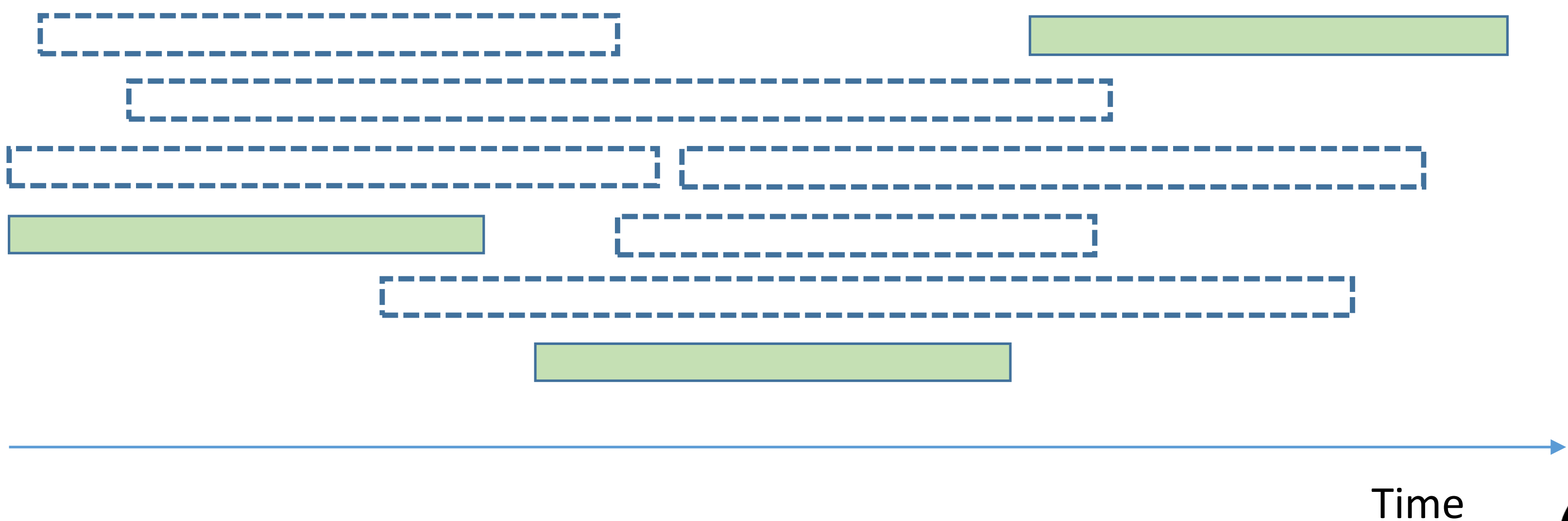
Done ! You should know why greedy algorithm guarantees the global optimal

Poscat

Interval Scheduling

- Problem

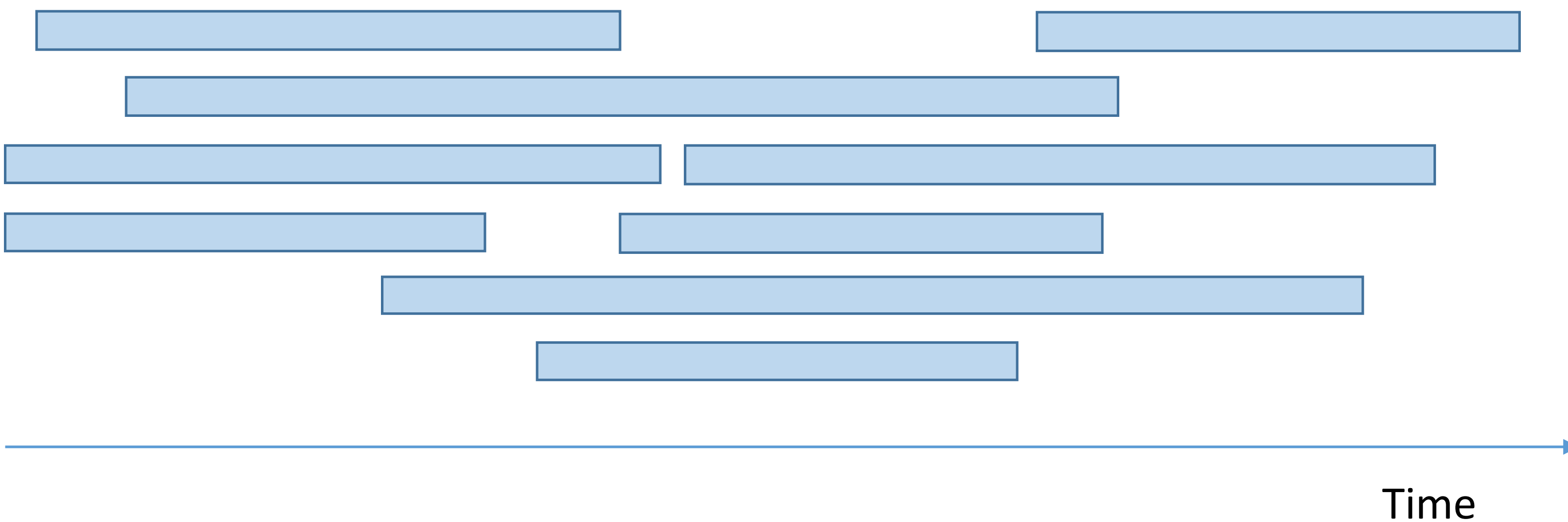
Idea : What have to be the first interval ?



Time complexity : **O(n log n)** for sorting

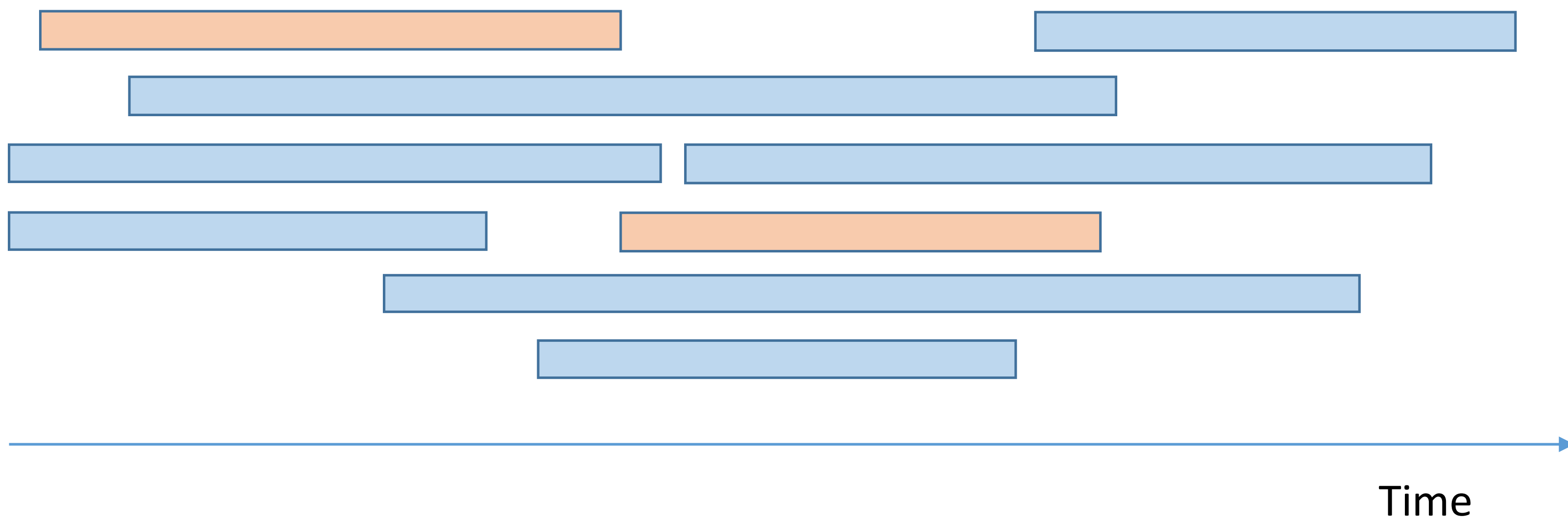
Interval Partitioning

- Problem
  - Lecture j starts at s_j and finishes at f_j
  - Find minimum number of classrooms to schedule all lectures so that no two occur at the same time in the same room



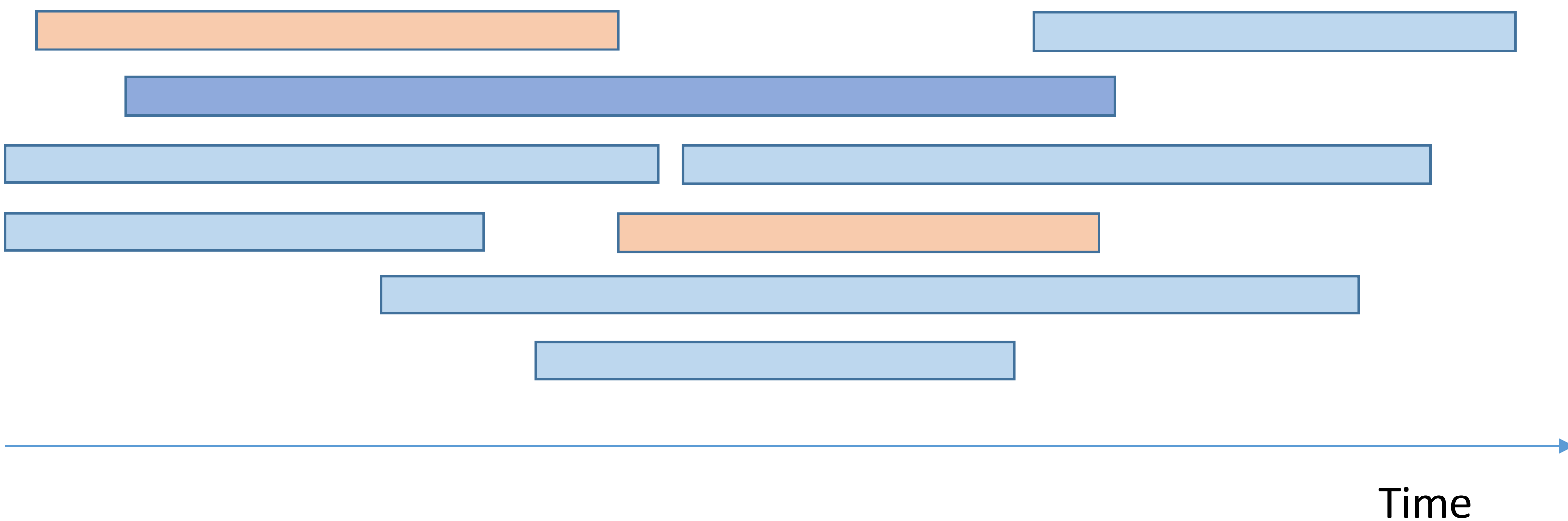
Interval Partitioning

- Problem
  - Lecture j starts at s_j and finishes at f_j
  - Find minimum number of classrooms to schedule all lectures so that no two occur at the same time in the same room



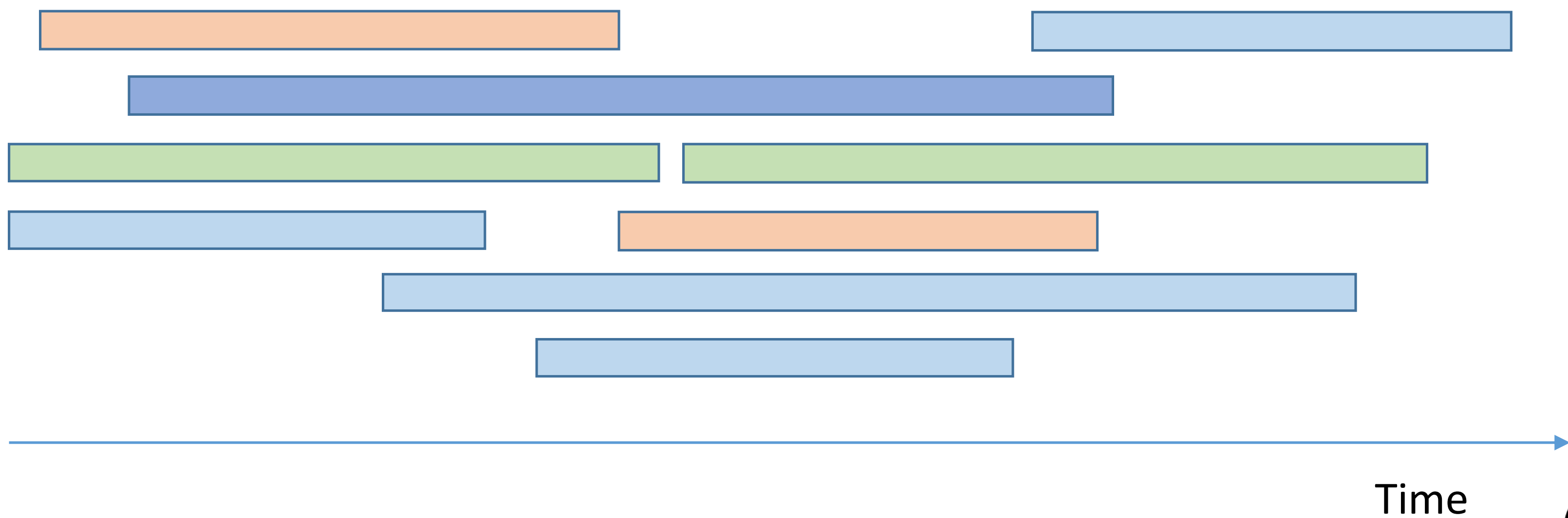
Interval Partitioning

- Problem
  - Lecture j starts at s_j and finishes at f_j
  - Find minimum number of classrooms to schedule all lectures so that no two occur at the same time in the same room



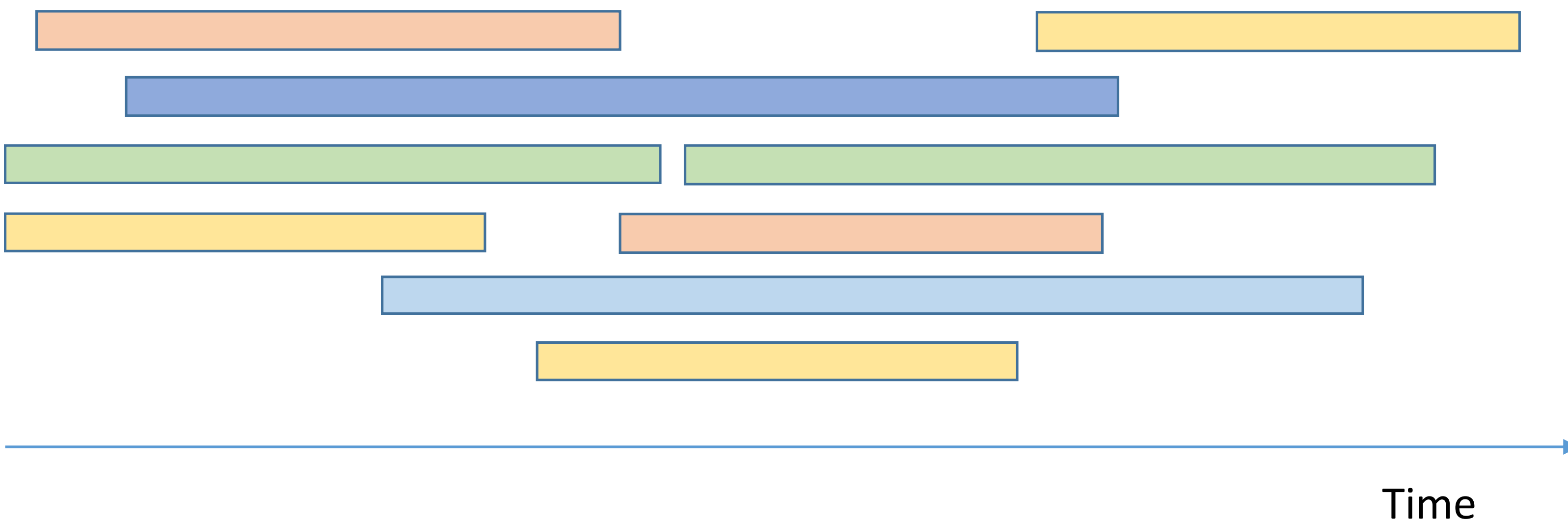
Interval Partitioning

- Problem
  - Lecture j starts at s_j and finishes at f_j
  - Find minimum number of classrooms to schedule all lectures so that no two occur at the same time in the same room



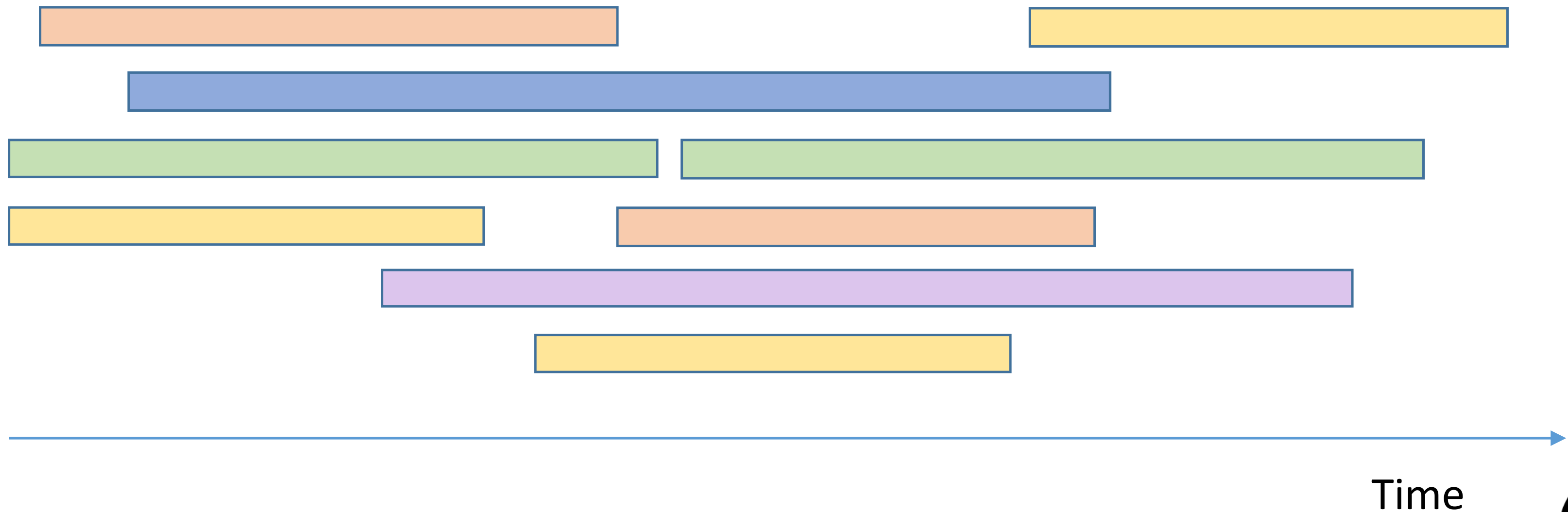
Interval Partitioning

- Problem
  - Lecture j starts at s_j and finishes at f_j
  - Find minimum number of classrooms to schedule all lectures so that no two occur at the same time in the same room



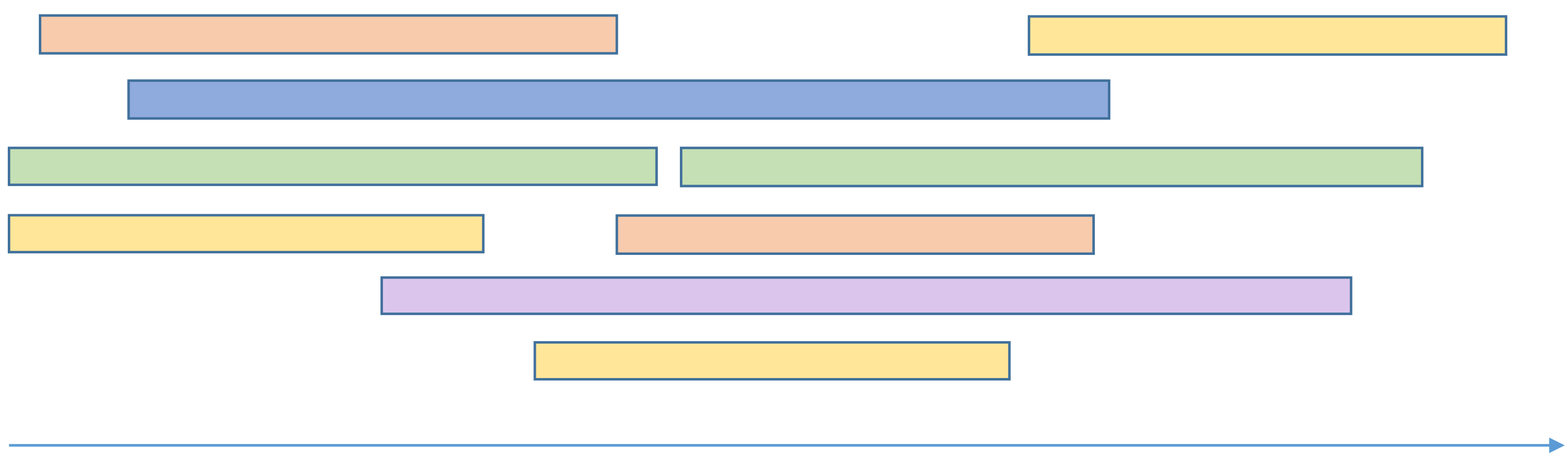
Interval Partitioning

- Problem
  - Lecture j starts at s_j and finishes at f_j
  - Find minimum number of classrooms to schedule all lectures so that no two occur at the same time in the same room



Interval Partitioning

- Problem
  - Lecture j starts at s_j and finishes at f_j
  - Find minimum number of classrooms to schedule all lectures so that no two occur at the same time in the same room



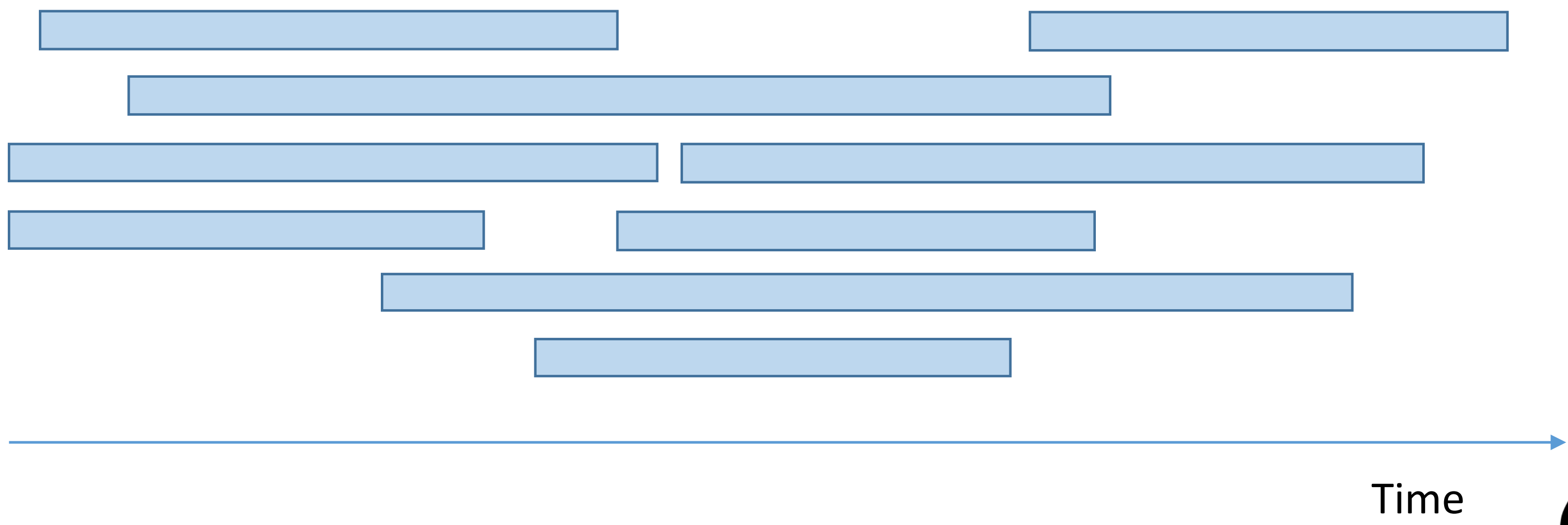
Time

We need 5 classrooms !

Interval Partitioning

- Problem

Idea : What is the minimum number of classrooms ?



Interval Partitioning

- Problem

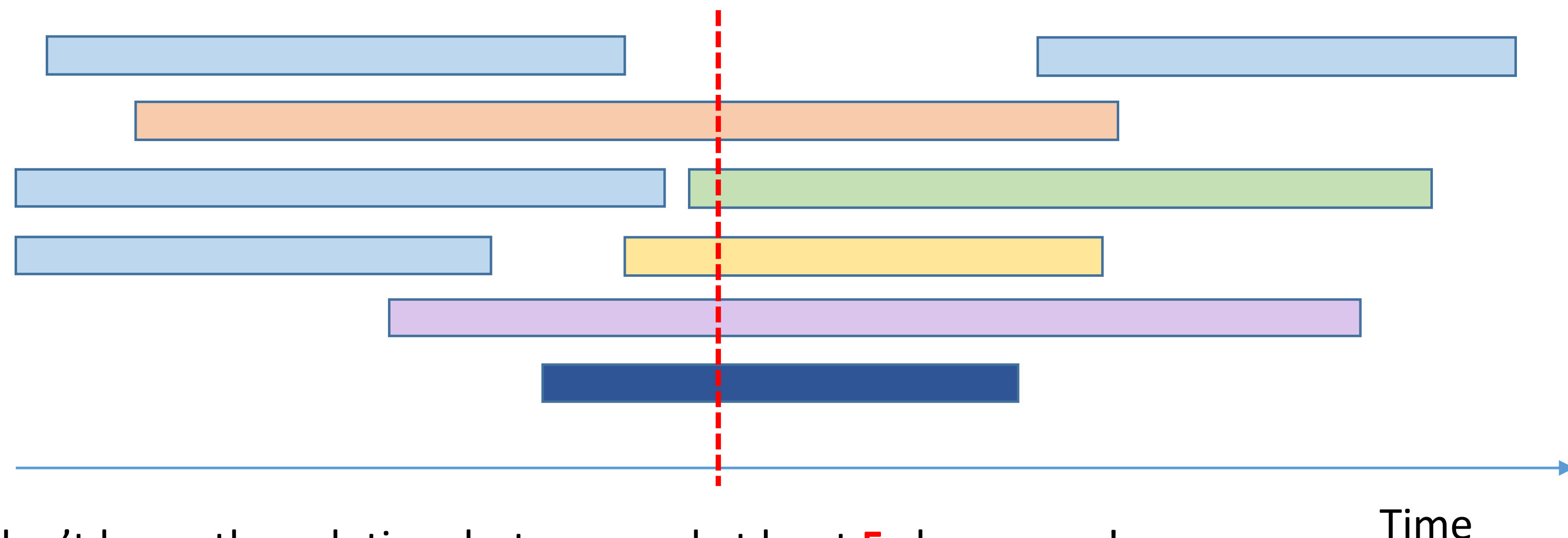
Idea : What is the minimum number of classrooms ?

We don’t know the solution, but we need at least 4 classrooms!
→ Is it sufficient ?

Interval Partitioning

- Problem

Idea : What is the minimum number of classrooms ?



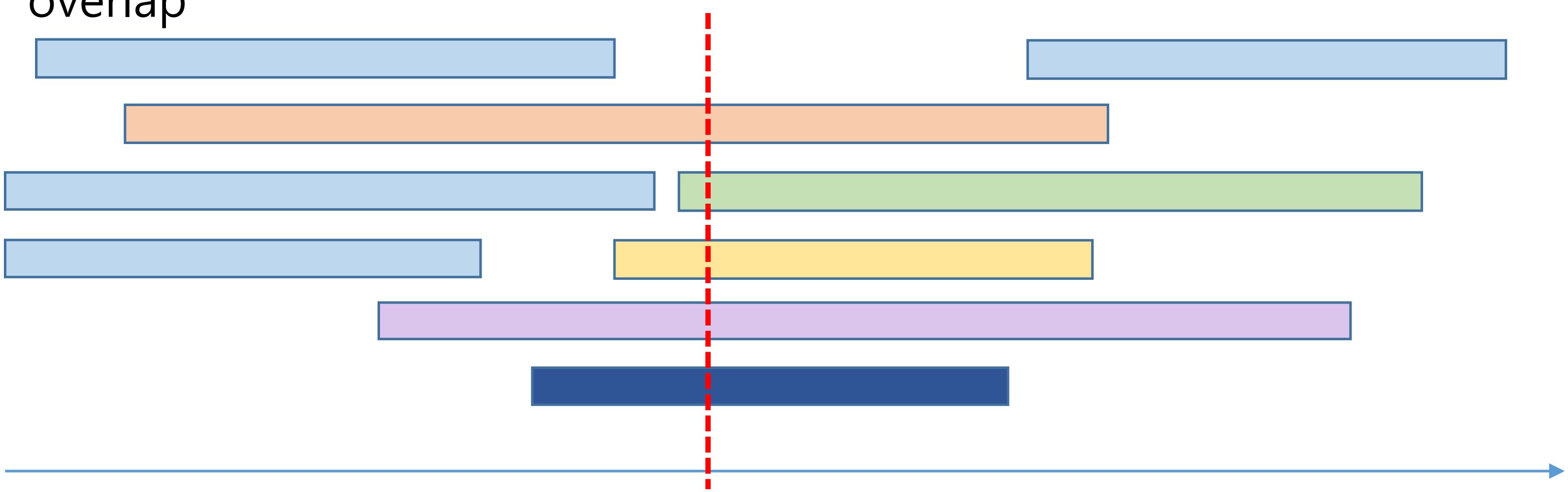
We don't know the solution, but we need at least **5** classrooms!

Interval Partitioning

- Problem

Idea : What is the minimum number of classrooms ?

∴ We need classrooms **at least** as many as the maximum number of overlap



Time

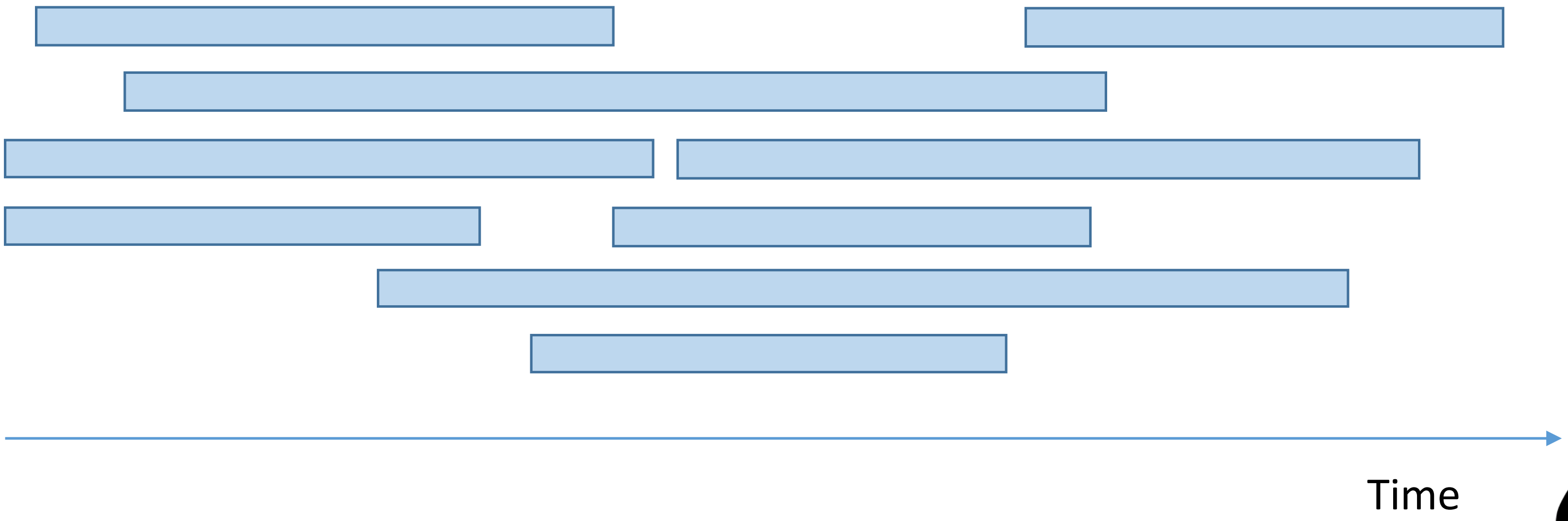
We don't know the solution, but we need at least **5** classrooms!

Poscat

Interval Partitioning

- Problem

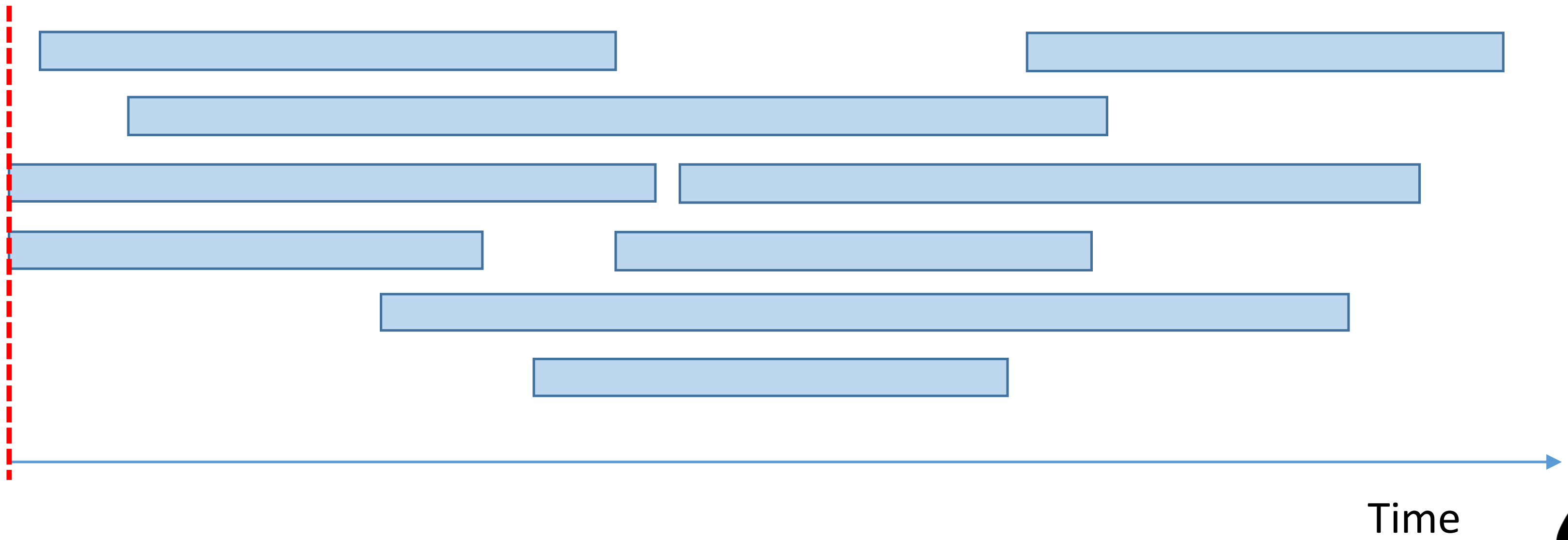
Is it sufficient ?



Interval Partitioning

- Problem

Is it sufficient ? Yes ! Consider this algorithm

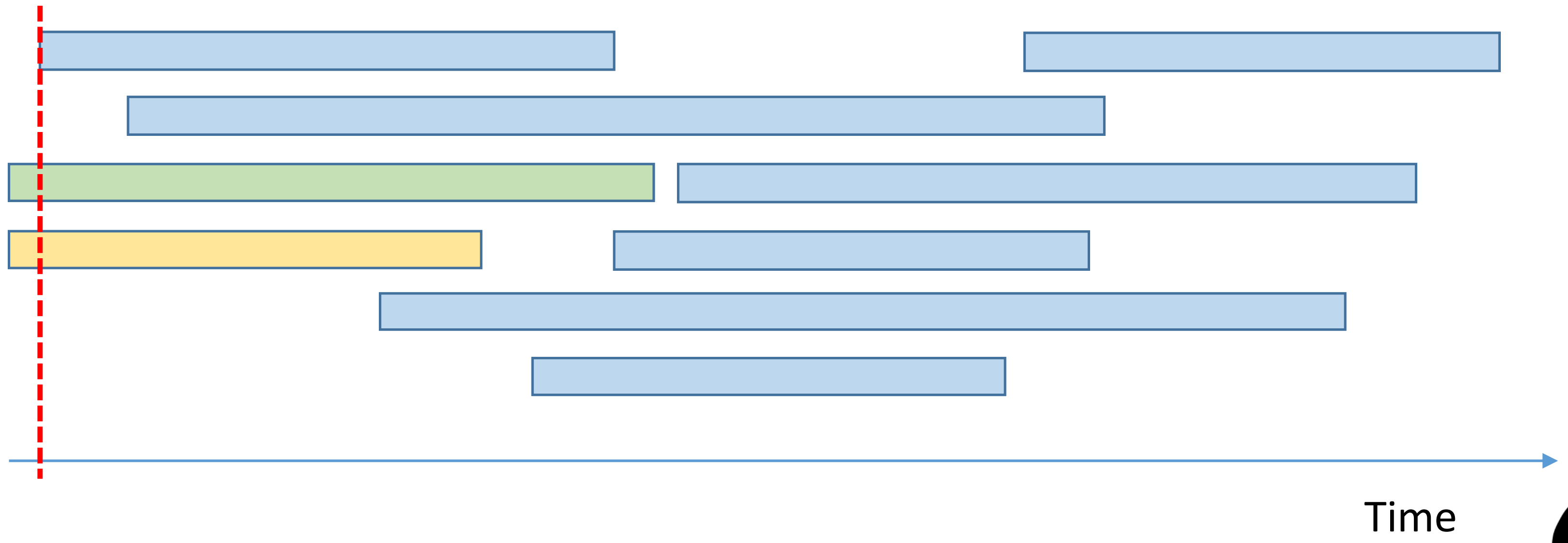


Poscat

Interval Partitioning

- Problem

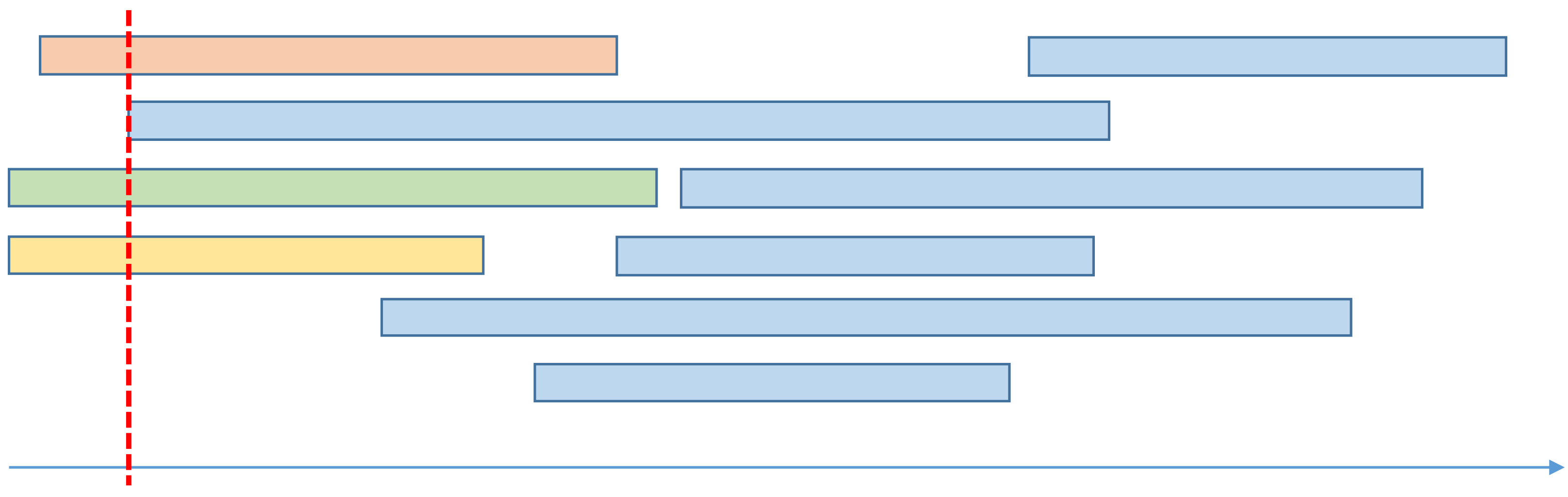
Is it sufficient ? Yes ! Consider this algorithm



Interval Partitioning

- Problem

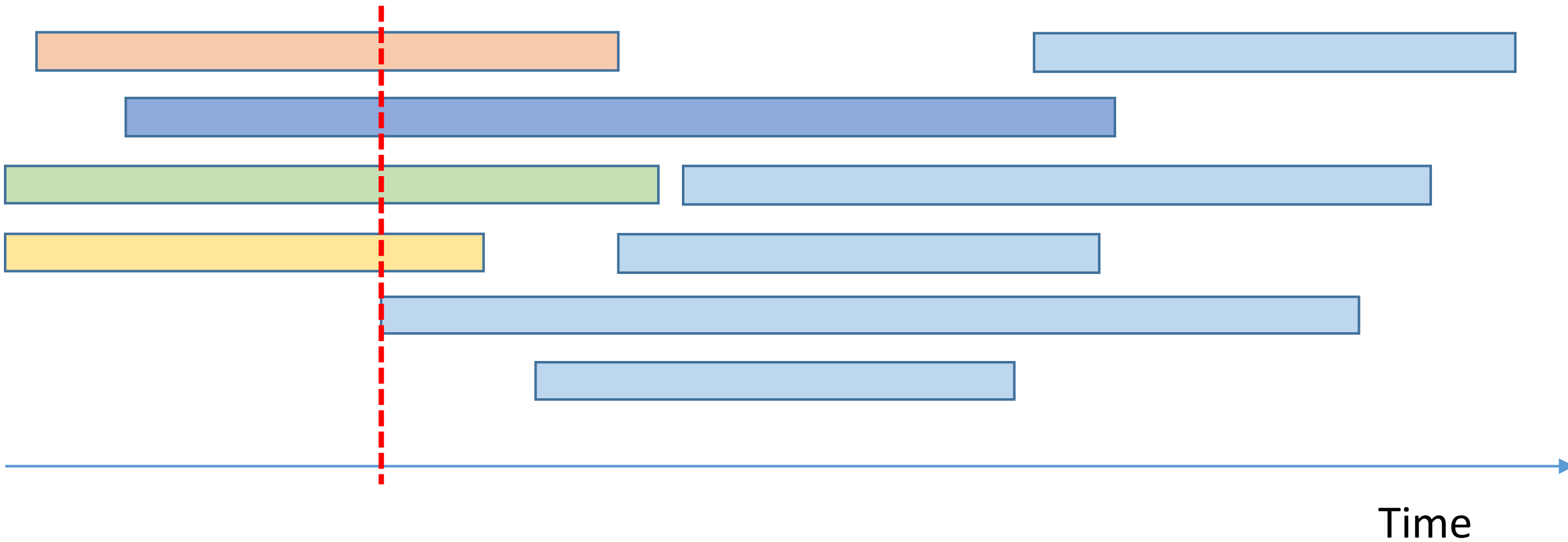
Is it sufficient ? Yes ! Consider this algorithm



Interval Partitioning

- Problem

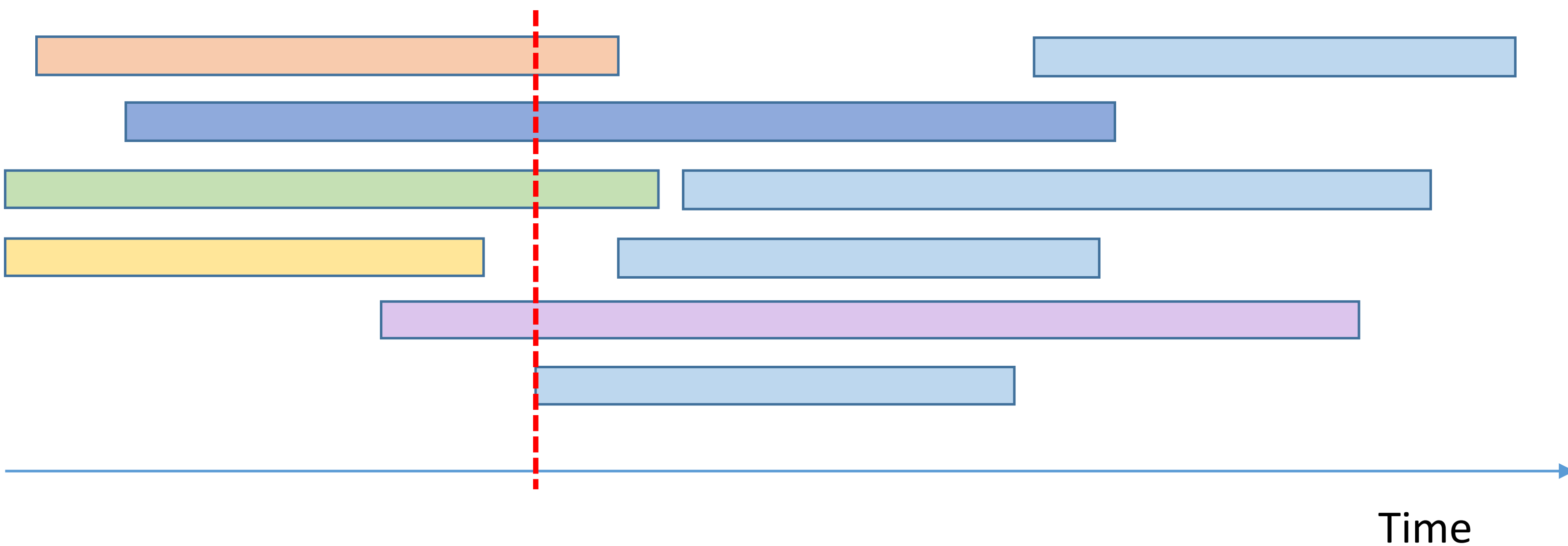
Is it sufficient ? Yes ! Consider this algorithm



Interval Partitioning

- Problem

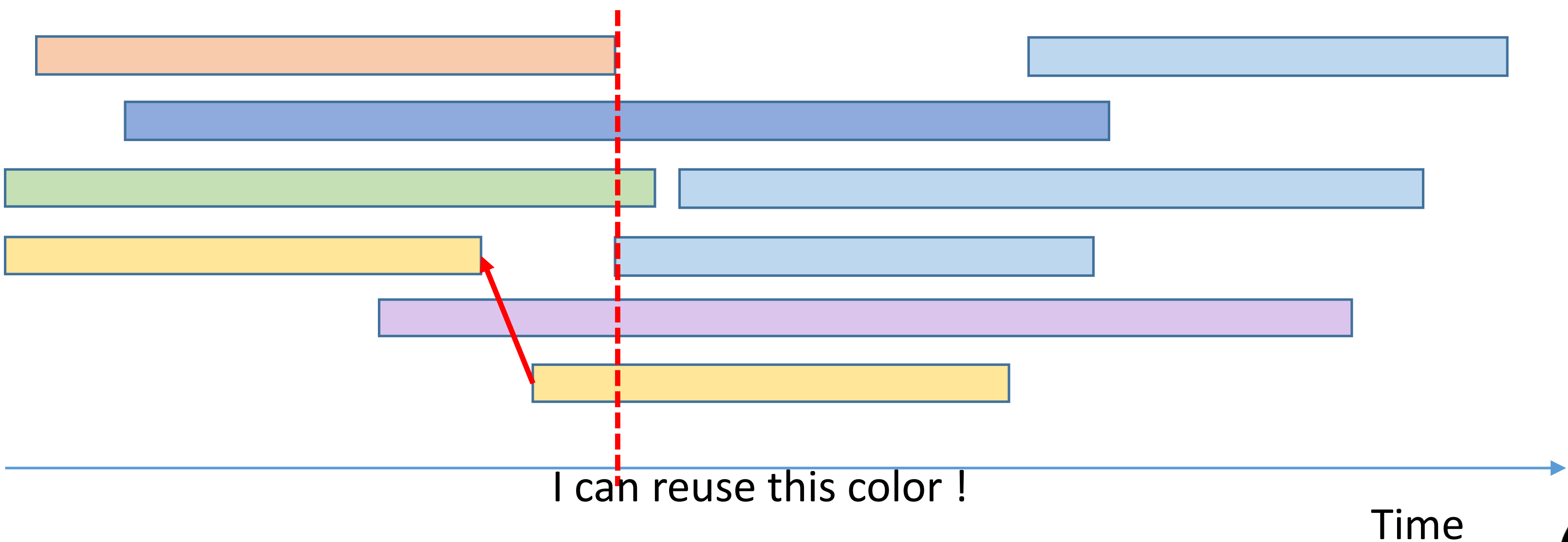
Is it sufficient ? Yes ! Consider this algorithm



Interval Partitioning

- Problem

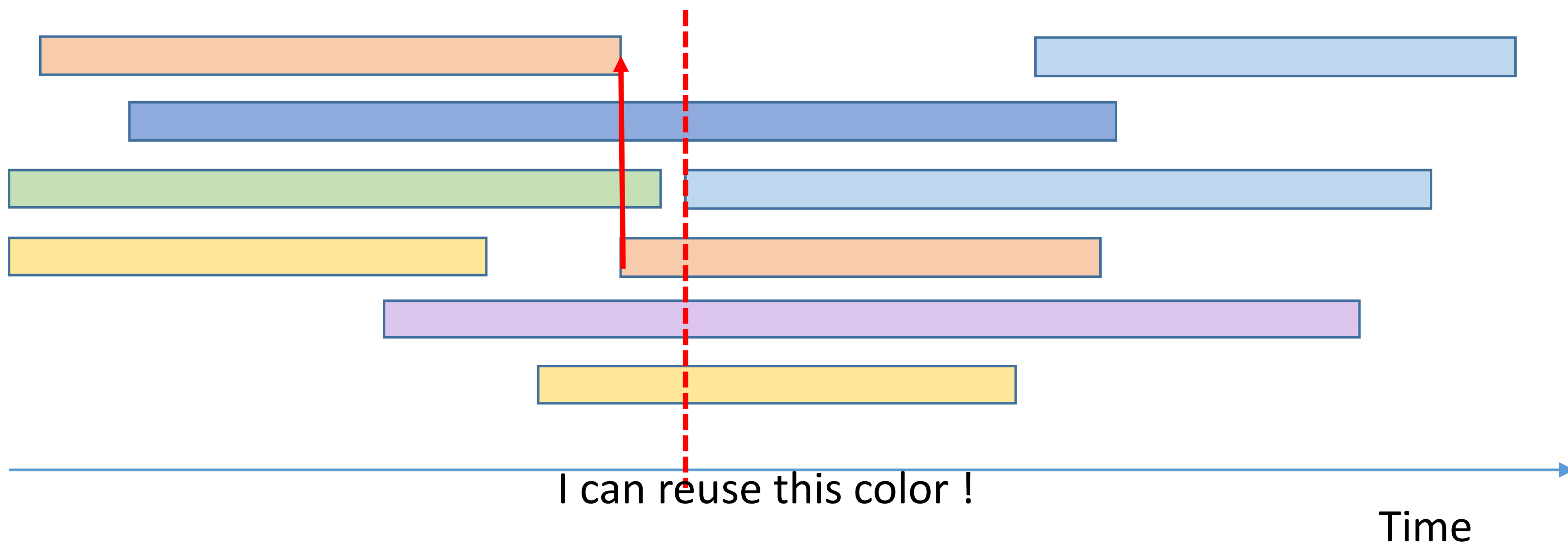
Is it sufficient ? Yes ! Consider this algorithm



Interval Partitioning

- Problem

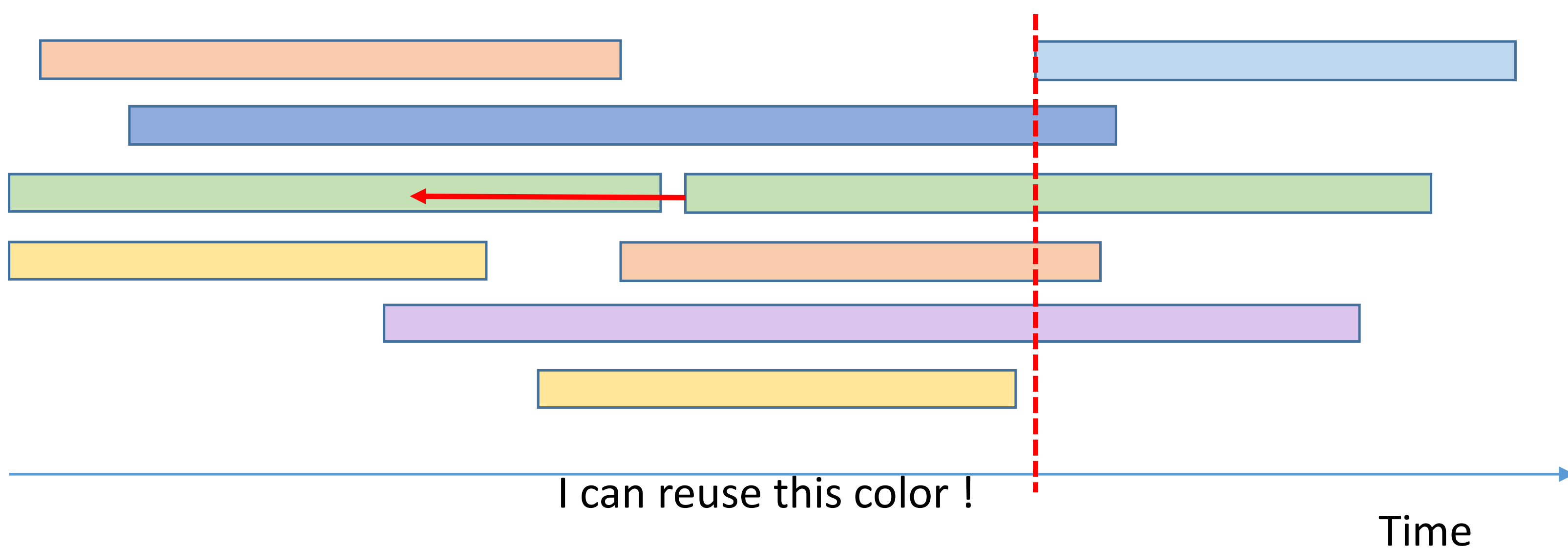
Is it sufficient ? Yes ! Consider this algorithm



Interval Partitioning

- Problem

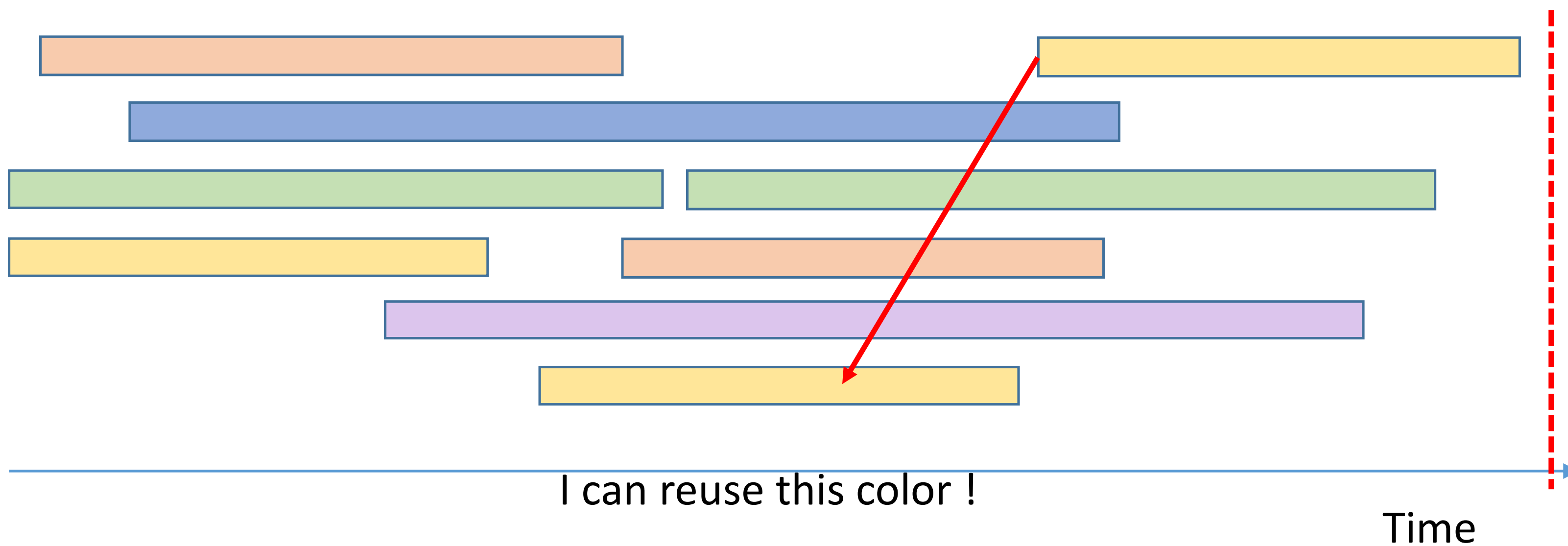
Is it sufficient ? Yes ! Consider this algorithm



Interval Partitioning

- Problem

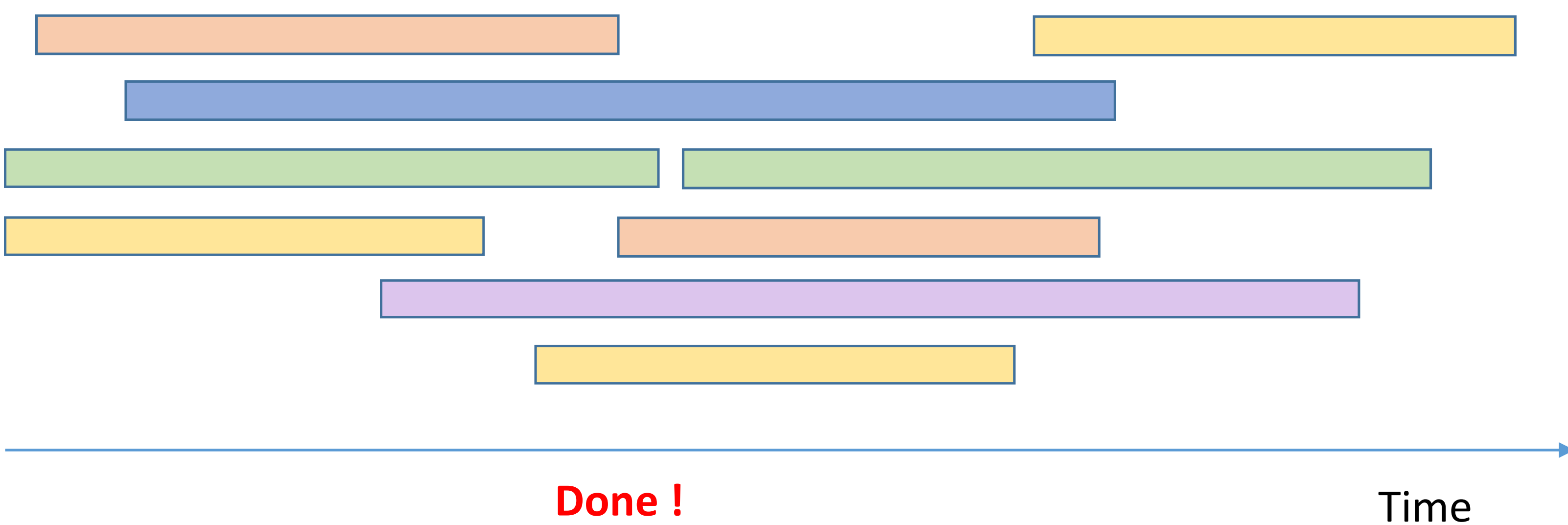
Is it sufficient ? Yes ! Consider this algorithm



Interval Partitioning

- Problem

Is it sufficient ? Yes ! Consider this algorithm



Interval Partitioning

function interval-partition

sort intervals by starting time so that $s_1 \leq s_2 \leq \cdots \leq s_n$.
$d = 0$.
for $j = 1$ to n **do**
  if lecture j is compatible with some classroom k **then**
    schedule lecture j in classroom k
  else
    allocate a new classroom $d + 1$
    schedule lecture j in classroom $d + 1$
    $d = d + 1$
  end if
end for

Prove that this algorithm needs classrooms no more than the maximum number of overlap

Poscat

Interval Partitioning

function interval-partition

sort intervals by starting time so that $s_1 \leq s_2 \leq \cdots \leq s_n$.
$d = 0$.
for $j = 1$ to n **do**
  if lecture j is compatible with some classroom k **then**
    schedule lecture j in classroom k
  else
    allocate a new classroom $d + 1$
    schedule lecture j in classroom $d + 1$
    $d = d + 1$
  end if
end for

Prove that this algorithm needs classrooms no more than the maximum number of overlap

Let M be maximum number of overlap.
Suppose that our algorithm needs $M + 1$ classrooms

Interval Partitioning

function interval-partition

sort intervals by starting time so that $s_1 \leq s_2 \leq \cdots \leq s_n$.
$d = 0$.
for $j = 1$ to n **do**
    if lecture j is compatible with some classroom k **then**
        schedule lecture j in classroom k
    else
        allocate a new classroom $d + 1$
        schedule lecture j in classroom $d + 1$
        $d = d + 1$
    end if
end for

Prove that this algorithm needs classrooms no more than the maximum number of overlap

Consider the situation that we need last classrooms when we consider lecture j.
In other words, we need 1 more classroom although we already have M classrooms

Poscat

Interval Partitioning

function interval-partition

sort intervals by starting time so that $s_1 \leq s_2 \leq \cdots \leq s_n$.
$d = 0$.
for $j = 1$ to n **do**
  if lecture j is compatible with some classroom k **then**
    schedule lecture j in classroom k
  else
    allocate a new classroom $d + 1$
    schedule lecture j in classroom $d + 1$
    $d = d + 1$
  end if
end for

Prove that this algorithm needs classrooms no more than the maximum number of overlap

To allocate another classroom, we have to go to "else" part
It means that lecture j have no compatible classroom

Interval Partitioning

function interval-partition

sort intervals by starting time so that $s_1 \leq s_2 \leq \cdots \leq s_n$.
$d = 0$.
for $j = 1$ to n **do**
  if lecture j is compatible with some classroom k **then**
    schedule lecture j in classroom k
  else
    allocate a new classroom $d + 1$
    schedule lecture j in classroom $d + 1$
    $d = d + 1$
  end if
end for

Implementation: $O(n \log n)$.
For each classroom k, maintain the finish time of the last job added.
Keep the classrooms in a priority queue.

Prove that this algorithm needs classrooms no more than the maximum number of overlap

To allocate another classroom, we have to go to "else" part
It means that lecture j have no compatible classroom
Therefore, the maximum number of overlap have to be $M + 1$. Contradiction.

Interval Partitioning

function interval-partition

sort intervals by starting time so that $s_1 \leq s_2 \leq \cdots \leq s_n$.
$d = 0$.
for $j = 1$ to n **do**
  if lecture j is compatible with some classroom k **then**
    schedule lecture j in classroom k
  else
    allocate a new classroom $d + 1$
    schedule lecture j in classroom $d + 1$
    $d = d + 1$
  end if
end for

Implementation ?

Interval Partitioning

```
function interval-partition
sort intervals by starting time so that s1 ≤ s2 ≤ ··· ≤ sn.
d = 0.
for j = 1 to n do
   if lecture j is compatible with some classroom k then
      schedule lecture j in classroom k
   else
      allocate a new classroom d + 1
      schedule lecture j in classroom d + 1
      d = d + 1
   end if
end for
```

Implementation: $O(n \log n)$.
For each classroom k, maintain the finish time of the last job added.
Keep the classrooms in a priority queue.

Implementation ?

By using priority queue, we can make **$O(n \log n)$** algorithm

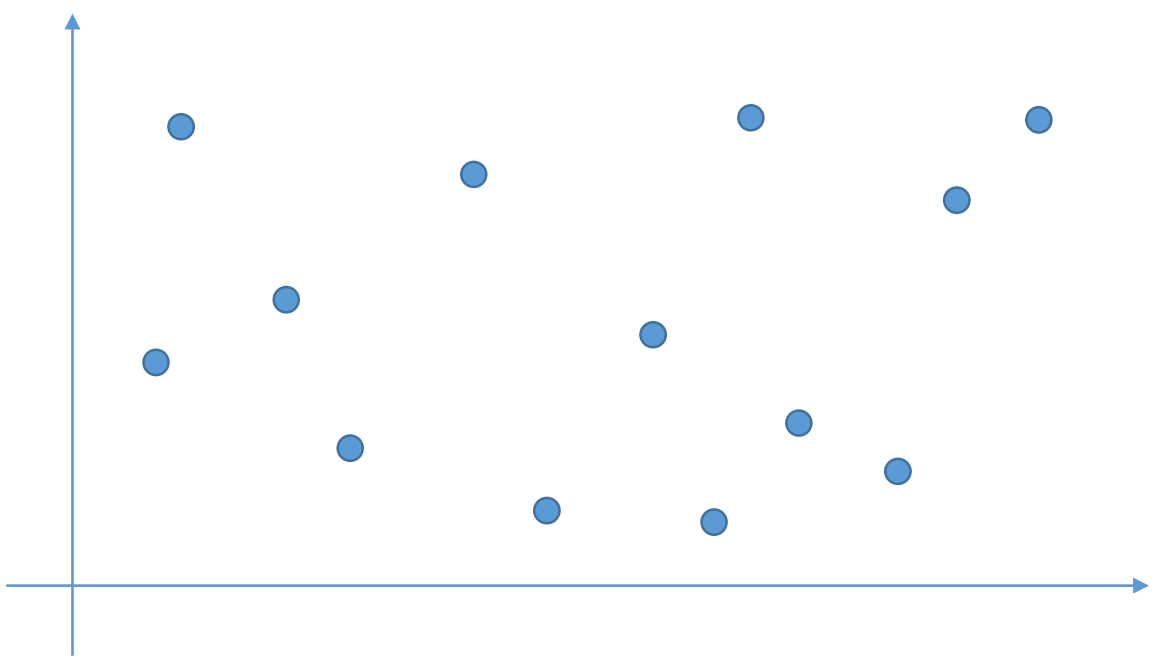
Fractional Knapsack

- Easy, just think about it
  - I'll provide this problem today

Other problems

- Problem

Given N points, find a maximum value of m such that $m = |\frac{\Delta y}{\Delta x}|$

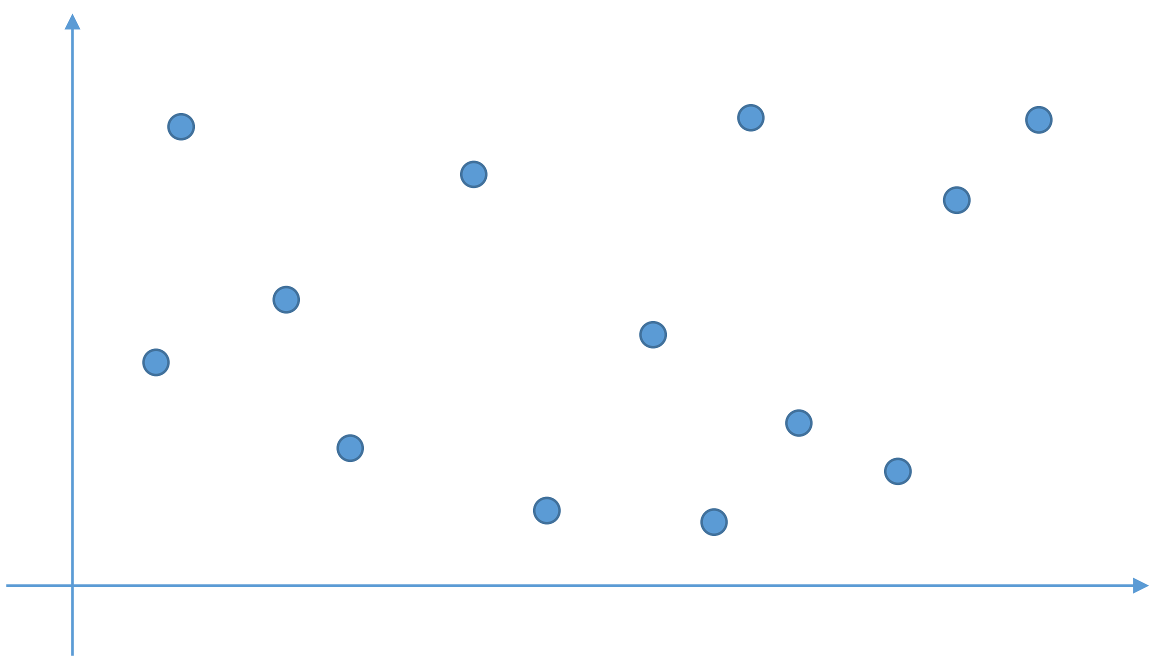


Other problems

- Problem

Given N points, find a maximum value of m such that $m = |\frac{\Delta y}{\Delta x}|$

Naïve approach → O(n^2)

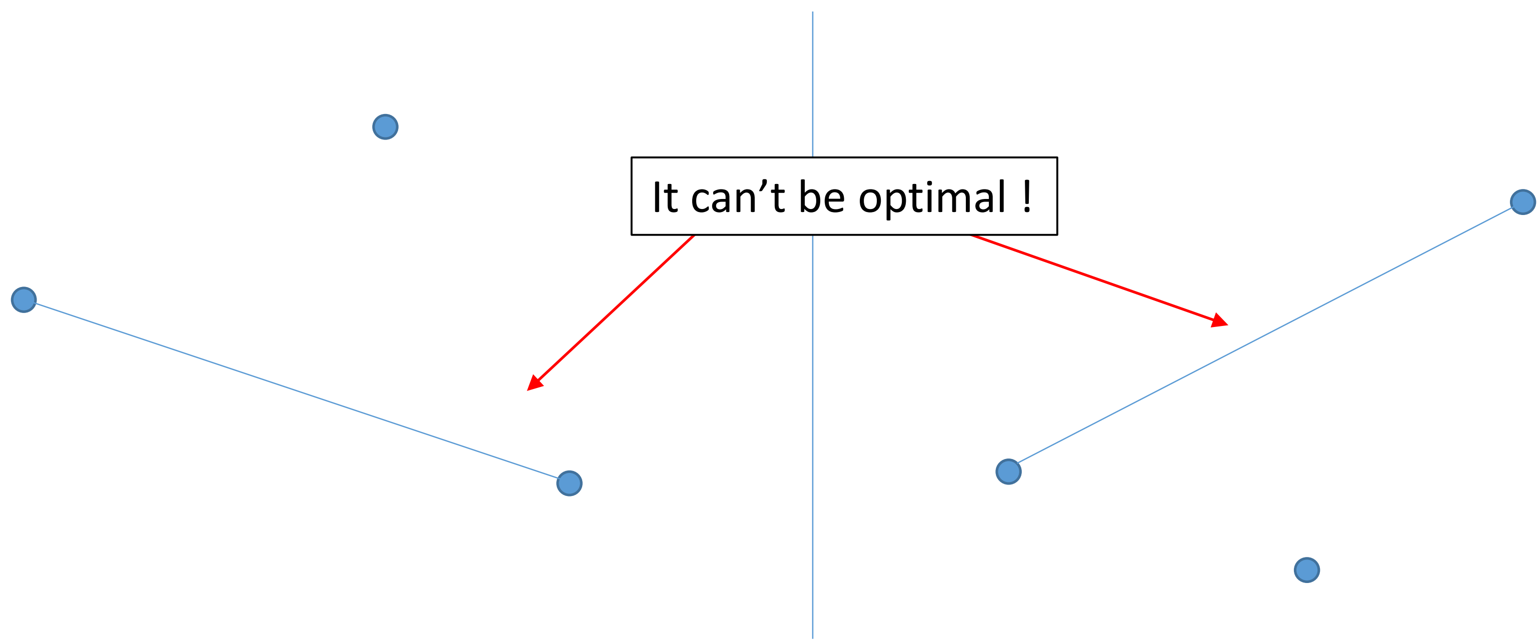


Other problems

- Problem

Given N points, find a maximum value of m such that $m = |\frac{\Delta y}{\Delta x}|$

We can prove that two points have to be adjacent



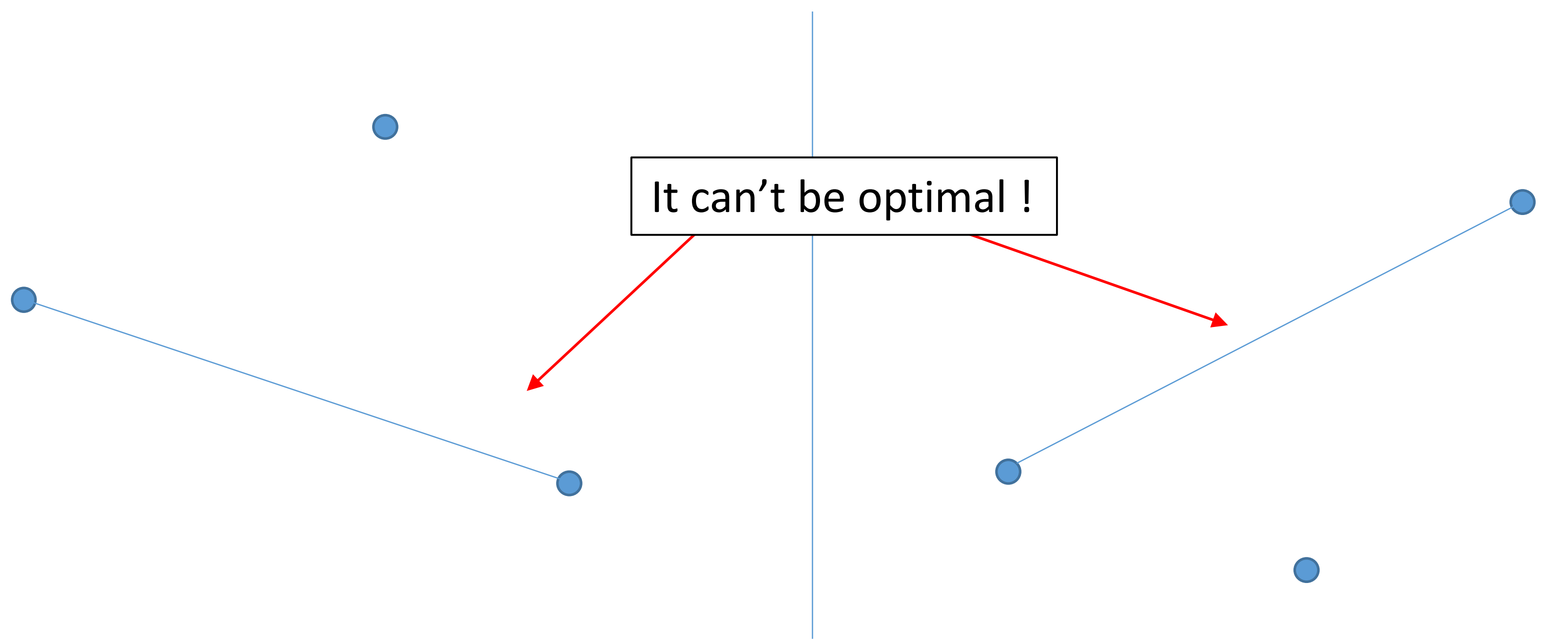
Poscat

Other problems

- Problem

Given N points, find a maximum value of m such that $m = |\frac{\Delta y}{\Delta x}|$

We can prove that two points have to be adjacent → O($n \log n$)



Other problems

- Problem

 You have 2N cards, and each card have two numbers on both sides.

There are 2N locations to put your card on the table. For each location, it has specific sign which is changed alternatively (+ or -).

Find the maximum value of the result of your calculation.

Locations : + [ ] - [ ] + [ ] - [ ] + [ ] - [ ] = ?

Cards (front/back)						
front	1	9	-9	7	4	2
back	2	-1	-8	4	-3	-1

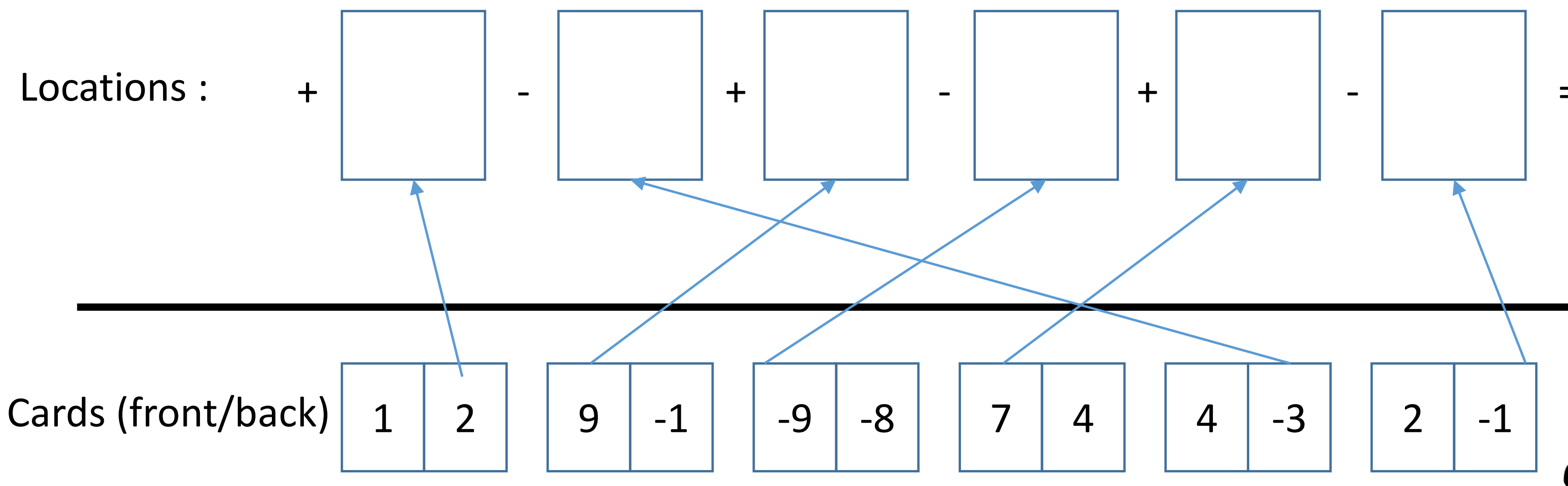
Other problems

- ## Problem

You have 2N cards, and each card have two numbers on both sides.

There are 2N locations to put your card on the table. For each location, it has specific sign which is changed alternatively (+ or -).

Find the maximum value of the result of your calculation.



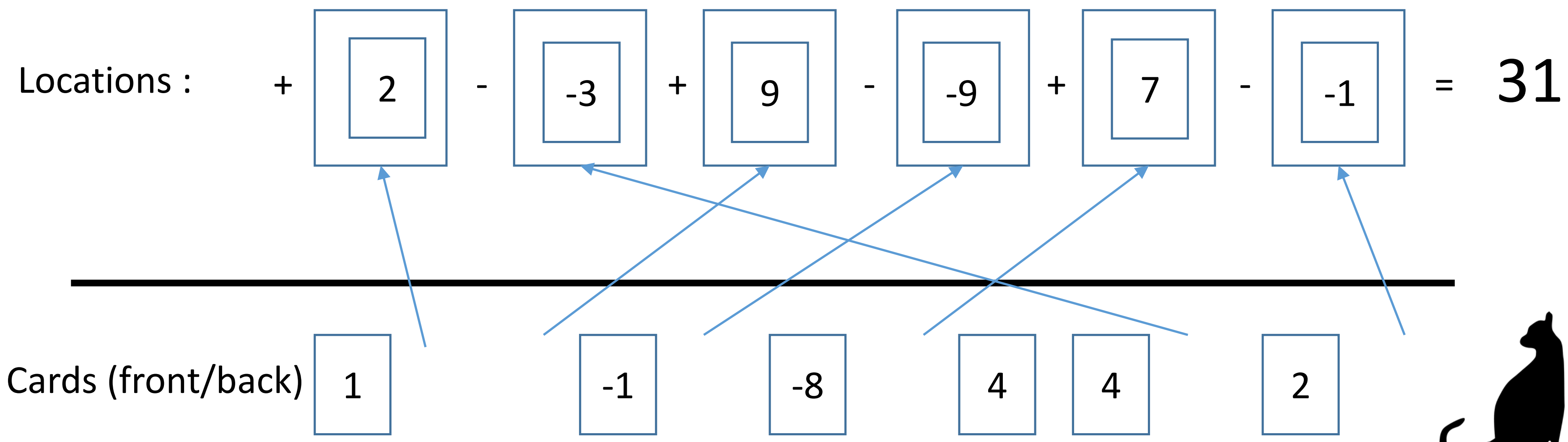
Other problems

- Problem

You have 2N cards, and each card have two numbers on both sides.

There are 2N locations to put your card on the table. For each location, it has specific sign which is changed alternatively (+ or -).

Find the maximum value of the result of your calculation.



Other problems

- Solution

  Let a_i be the value of bigger side of card i.

  b_i be the value of smaller side of card i.

  If card i which is located with (+) should have the value a_i

  Otherwise, it should have the value b_i

Other problems

- Solution

Let a_i be the value of bigger side of card i.
b_i be the value of smaller side of card i.

If card i which is located with (+) should have the value a_i
Otherwise, it should have the value b_i

Let S = { i | card i is located in (+) side }
Let S' = { i | card i is located in (-) side }

Other problems

- Solution

Let a_i be the value of bigger side of card i.

b_i be the value of smaller side of card i.

Then our result is represented by

$$\sum_{i \in S} a_i - \sum_{j \in S'} b_j$$

Poscat

Other problems

- Solution

  Let a_i be the value of bigger side of card i.

  b_i be the value of smaller side of card i.

  Then our result is represented by

$$\sum_{i \in S} a_i - \sum_{j \in S'} b_j = \sum_{i \in S} a_i - \sum_{j \in S'} b_j + \left(\sum_{i \in S} b_i - \sum_{i \in S} b_i \right)$$

Other problems

- Solution

Let a_i be the value of bigger side of card i.

b_i be the value of smaller side of card i.

Then our result is represented by

$$\sum_{i \in S} a_i + \sum_{j \in S'} b_j = \sum_{i \in S} a_i - \sum_{j \in S'} b_j + \left(\sum_{i \in S} b_i - \sum_{i \in S} b_i \right)$$

$$\sum_{i \in S} a_i + \sum_{i \in S} b_i - \left(\sum_{j \in S'} b_j + \sum_{i \in S} b_i \right)$$

Other problems

- Solution

Let a_i be the value of bigger side of card i.

b_i be the value of smaller side of card i.

Then our result is represented by

$$\sum_{i \in S} a_i + \sum_{j \in S'} b_j = \sum_{i \in S} a_i - \sum_{j \in S'} b_j + \left(\sum_{i \in S} b_i - \sum_{i \in S} b_i \right)$$

$$\sum_{i \in S} (a_i + b_i) - (Sum\ of\ b)$$

Other problems

- Solution

Let a_i be the value of bigger side of card i.

b_i be the value of smaller side of card i.

$$\sum_{i \in S} (a_i + b_i) - (Sum\ of\ b)$$

(Sum of b) is constant because b is the smaller value of each card.

Other problems

- Solution

Let a_i be the value of bigger side of card i.

b_i be the value of smaller side of card i.

$$\sum_{i \in S} (a_i + b_i) - (Sum\ of\ b)$$

(Sum of b) is constant because b is the smaller value of each card.

Therefore, we have to choose i by considering the value of $a + b$

It means that a card whose $a + b$ value is larger should be located in (+)

Other problems

- Solution

1. Sort cards with respect to the value of a + b
2. N cards whose value is larger should be located in (+)
3. Remaining N cards should be located in (-)

$$O(n \log n)$$