Greedy Algorithms
CSCI 532

Single Room Scheduling

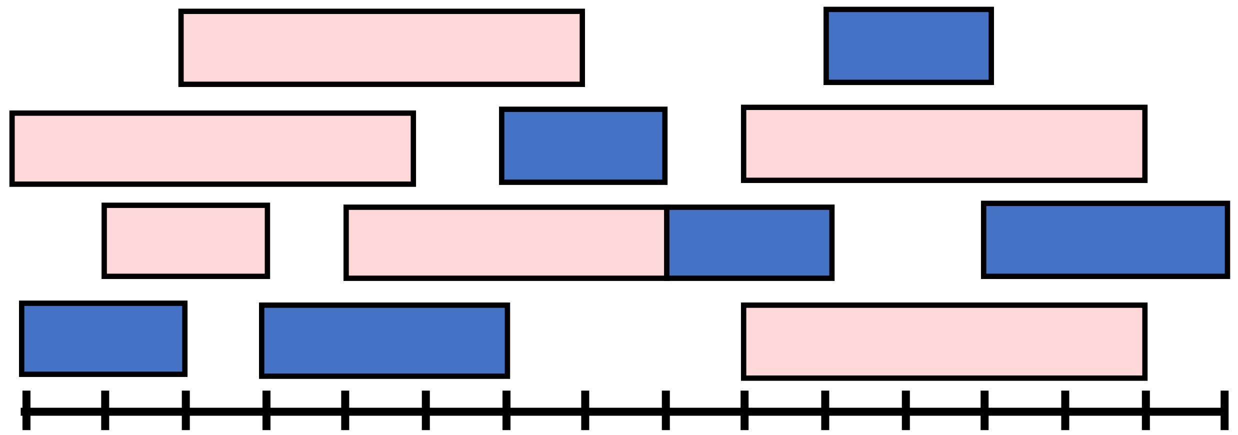
Input:

- $C = \{c_1, c_2, \dots, c_n\}$ – set of courses that need rooms.
- $c_i = [s_i, f_i)$ – start and finish times for each course.

Rules:

- c_i and c_j are compatible if $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap.

Goal: Select a maximum sized subset of compatible courses.



Greedy selection criteria?

Earliest compatible finish.

In each iteration, pick the course that ends earliest and is compatible with existing schedule.

Single Room Scheduling

Greedy decision: Select the next course with the earliest compatible finish time.

Proof of optimality: Let C be the set of courses, $S_{ALG} \subseteq C$ be the greedy algorithm's selection, and $S_{OPT} \subseteq C$ be an optimal selection, all sorted by increasing finish time.

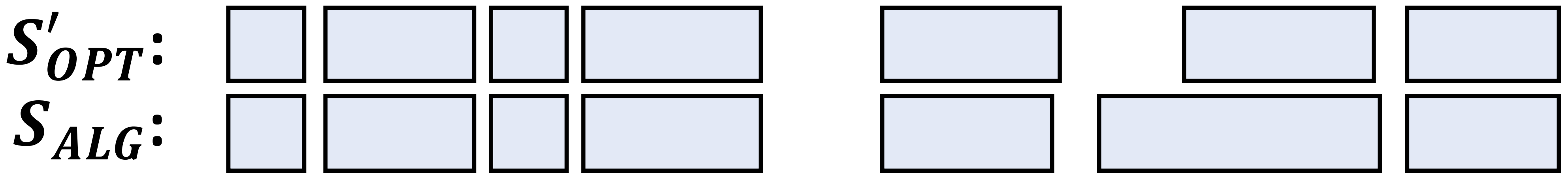
Suppose $S_{ALG}[i] = S_{OPT}[i]$, for all $i < k$ and $S_{ALG}[k] = c_i \neq c_j = S_{OPT}[k]$.

Create the revised schedule $S'_{OPT} = S_{OPT} \setminus \{c_j\} \cup \{c_i\}$. (I.e., Swap $S_{ALG}[k]$ for $S_{OPT}[k]$)

c_i is compatible with previous courses in S'_{OPT} since $S_{ALG}[i] = S_{OPT}[i] = S'_{OPT}[i]$, for all $i < k$

c_i is compatible with subsequent courses in S'_{OPT} since $f_i \leq f_j$. Otherwise, the greedy algorithm would have selected c_j instead of c_i.

So S'_{OPT} is a valid schedule with the same number of courses as S_{OPT}, so S'_{OPT} is also optimal.



Room Minimization

Input:

- $C = \{c_1, c_2, \dots, c_n\}$ – set of courses that need rooms.
- $c_i = [s_i, f_i)$ – start and finish times for each course.

Rules:

- c_i and c_j are compatible if $[s_i, f_i)$ and $\left[s_j, f_j\right)$ do not overlap.

Goal: Compatibly schedule all courses with the min number of rooms.

Room Minimization

Input:

- $C = \{c_1, c_2, \dots, c_n\}$ – set of courses that need rooms.
- $c_i = [s_i, f_i)$ – start and finish times for each course.

Rules:

- c_i and c_j are compatible if $[s_i, f_i)$ and $\left[s_j, f_j\right)$ do not overlap.

Goal: Compatibly schedule all courses with the min number of rooms.

Algorithm Idea?

Assign as much as possible to room 1,
then as much as possible to room 2,…

Optimal?

Room Minimization

Input:

- $C = \{c_1, c_2, \dots, c_n\}$ – set of courses that need rooms.
- $c_i = [s_i, f_i)$ – start and finish times for each course.

Rules:

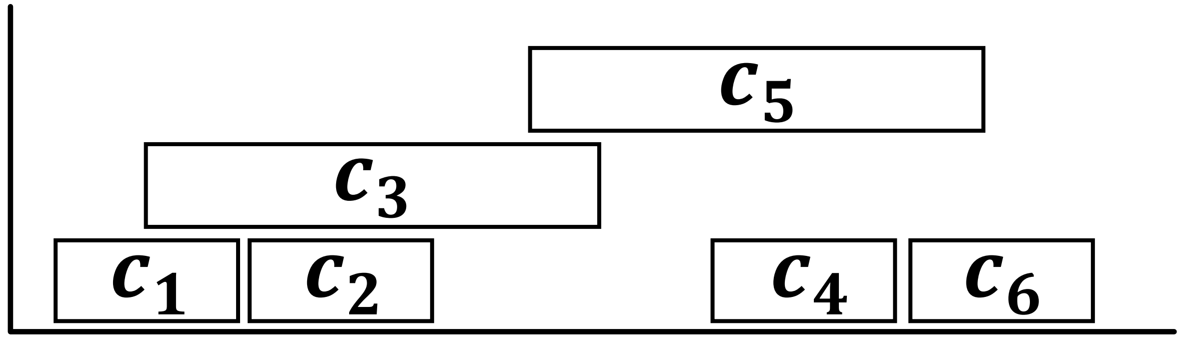
- c_i and c_j are compatible if $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap.

Goal: Compatibly schedule all courses with the min number of rooms.

Algorithm Idea?

Assign as much as possible to room 1, then as much as possible to room 2,…

Optimal? No!



Room Minimization

Input:

- $C = \{c_1, c_2, \dots, c_n\}$ – set of courses that need rooms.
- $c_i = [s_i, f_i)$ – start and finish times for each course.

Rules:

- c_i and c_j are compatible if $[s_i, f_i)$ and $\left[s_j, f_j\right)$ do not overlap.

Goal: Compatibly schedule all courses with the min number of rooms.

Algorithm Idea?

Given C, what is the smallest number of rooms possible?

Room Minimization

Input:

- $C = \{c_1, c_2, \dots, c_n\}$ – set of courses that need rooms.
- $c_i = [s_i, f_i)$ – start and finish times for each course.

Rules:

- c_i and c_j are compatible if $[s_i, f_i)$ and $\left[s_j, f_j\right)$ do not overlap.

Goal: Compatibly schedule all courses with the min number of rooms.

Algorithm Idea?

Given C, what is the smallest number of rooms possible?

The number of concurrent courses.

Room Minimization

Input:

- $C = \{c_1, c_2, \dots, c_n\}$ – set of courses that need rooms.
- $c_i = [s_i, f_i)$ – start and finish times for each course.

Rules:

- c_i and c_j are compatible if $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap.

Goal: Compatibly schedule all courses with the min number of rooms.

Algorithm Idea?

Given C, what is the smallest number of rooms possible?

The number of concurrent courses.

What if we made a schedule where # rooms = # concurrent courses?

Room Minimization

Input:

- $C = \{c_1, c_2, \ldots, c_n\}$ – set of courses that need rooms.
- $c_i = [s_i, f_i)$ – start and finish times for each course.

Rules:

- c_i and c_j are compatible if $[s_i, f_i)$ and $\left[s_j, f_j\right)$ do not overlap.

Goal: Compatibly schedule all courses with the min number of rooms.

Algorithm Idea?

Given C, what is the smallest number of rooms possible?

The number of concurrent courses.

What if we made a schedule where # rooms = # concurrent courses?

It must be optimal.

Room Minimization

Input:

- $C = \{c_1, c_2, \ldots, c_n\}$ – set of courses that need rooms.
- $c_i = [s_i, f_i)$ – start and finish times for each course.

Rules:

- c_i and c_j are compatible if $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap.

Goal: Compatibly schedule all courses with the min number of rooms.

Algorithm Idea?

Occupied Rooms

Unoccupied Rooms $\leftarrow$ But previously occupied

Room Minimization

Input:

- $C = \{c_1, c_2, \dots, c_n\}$ – set of courses that need rooms.
- $c_i = [s_i, f_i)$ – start and finish times for each course.

Rules:

- c_i and c_j are compatible if $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap.

Goal: Compatibly schedule all courses with the min number of rooms.

At each time:

Algorithm Idea?

Occupied Rooms

Unoccupied Rooms ← But previously occupied

Room Minimization

Input:

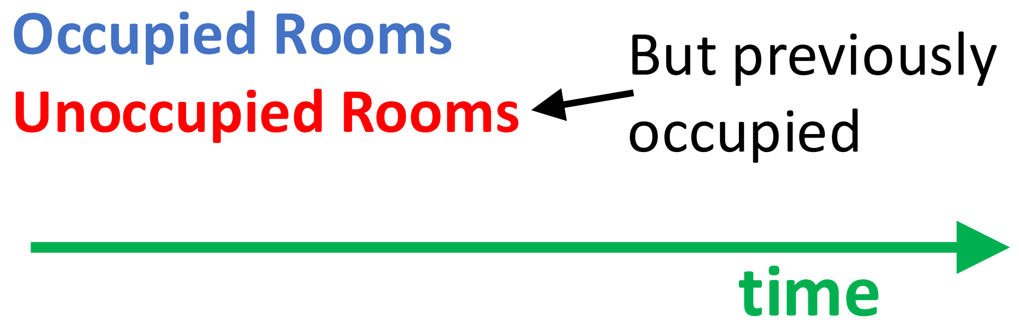
- $C = \{c_1, c_2, \dots, c_n\}$ – set of courses that need rooms.
- $c_i = [s_i, f_i)$ – start and finish times for each course.

Rules:

- c_i and c_j are compatible if $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap.

Goal: Compatibly schedule all courses with the min number of rooms.

Algorithm Idea?



At each time:

1. When course ends, move its **occupied room** to **unoccupied rooms**.

Room Minimization

Input:

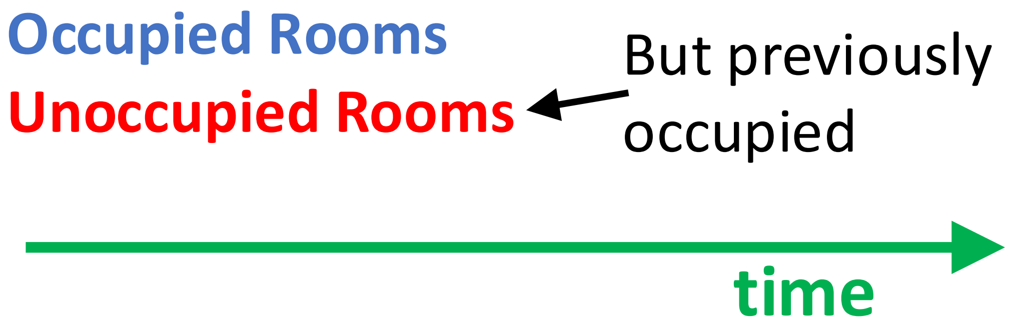
- $C = \{c_1, c_2, \ldots, c_n\}$ – set of courses that need rooms.
- $c_i = [s_i, f_i)$ – start and finish times for each course.

Rules:

- c_i and c_j are compatible if $[s_i, f_i)$ and $\left[s_j, f_j\right)$ do not overlap.

Goal: Compatibly schedule all courses with the min number of rooms.

Algorithm Idea?



At each time:

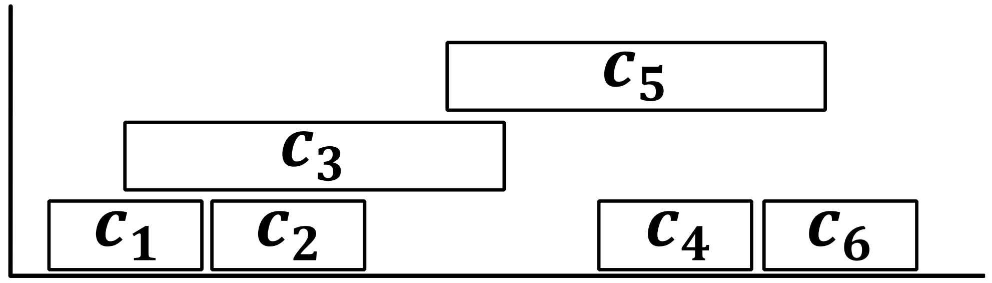
1. When course ends, move its **occupied room** to **unoccupied rooms**.
2. When course starts, **select unoccupied room**. If none exists, select a new (never used) room.

```
room_minimization(courses C)
    F = B = S = ∅
    room_num = 0
    foreach timeslot t
        foreach c in C
            if c.finish == t
                F.add(B.getBookedRoom(S.getroom(c)))
        foreach c in C
            if c.start == t
                if F.isEmpty()
                    room_num += 1
                    F.add(room_num)
                room = F.getFreeRoom()
                S.schedule(c, room)
                B.add(room)
    return S
```

F = Unoccupied Rooms

B = Occupied Rooms

S = course -> room Map

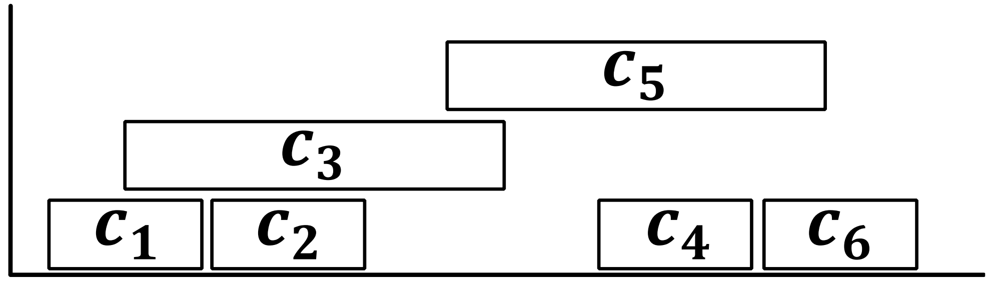


```
room_minimization(courses C)
    F = B = S = Ø
    room_num = 0
    foreach timeslot t
        foreach c in C
            if c.finish == t
                F.add(B.getBookedRoom(S.getroom(c)))
        foreach c in C
            if c.start == t
                if F.isEmpty()
                    room_num += 1
                    F.add(room_num)
                room = F.getFreeRoom()
                S.schedule(c, room)
                B.add(room)
    return S
```

F = Unoccupied Rooms

B = Occupied Rooms

S = course -> room Map



$F = \{\}$

$B = \{\}$

$S = \{\}$

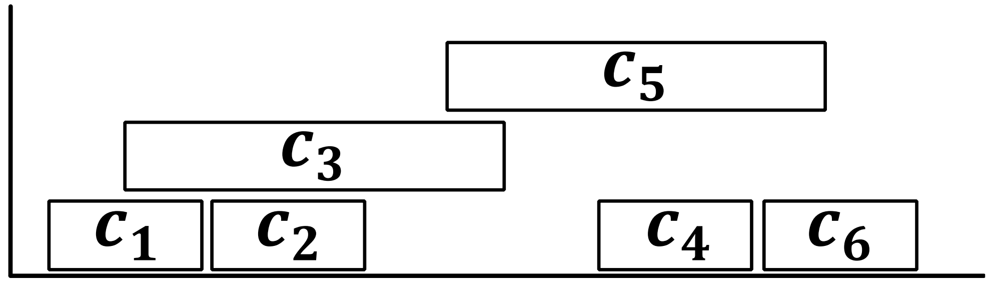
```
room_minimization(courses C)
    F = B = S = ∅
    room_num = 0
    foreach timeslot t
        foreach c in C
            if c.finish == t
                F.add(B.getBookedRoom(S.getroom(c)))
        foreach c in C
            if c.start == t
                if F.isEmpty()
                    room_num += 1
                    F.add(room_num)
                room = F.getFreeRoom()
                S.schedule(c, room)
                B.add(room)
    return S
```

Go through one timeslot at a time

F = Unoccupied Rooms

B = Occupied Rooms

S = course -> room Map



$F = \{\}$

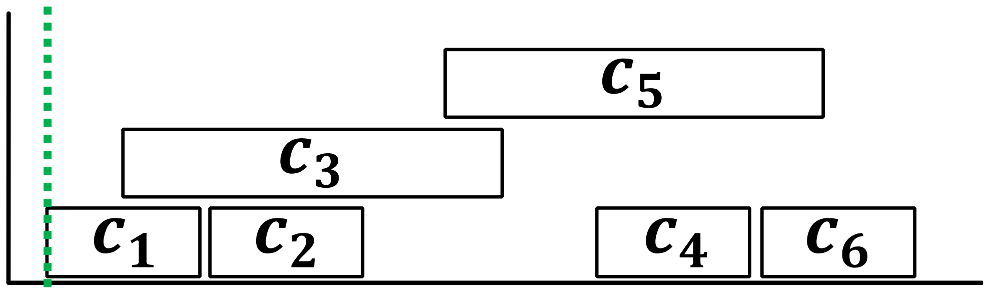
$B = \{\}$

$S = \{\}$

```
room_minimization(courses C)
    F = B = S = ∅
    room_num = 0
    foreach timeslot t
        foreach c in C
            if c.finish == t
                F.add(B.getBookedRoom(S.getroom(c)))
        foreach c in C
            if c.start == t
                if F.isEmpty()
                    room_num += 1
                    F.add(room_num)
                room = F.getFreeRoom()
                S.schedule(c, room)
                B.add(room)
    return S
```

Go through one timeslot at a time

F = Unoccupied Rooms
B = Occupied Rooms
S = course -> room Map



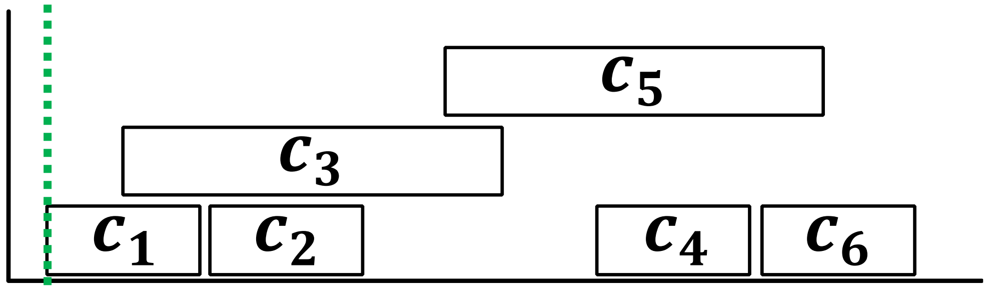
$F = \{\}$
$B = \{\}$
$S = \{\}$

```
room_minimization(courses C)
    F = B = S = Ø
    room_num = 0
    foreach timeslot t
        foreach c in C
            if c.finish == t
                F.add(B.getBookedRoom(S.getroom(c)))
        foreach c in C
            if c.start == t
                if F.isEmpty()
                    room_num += 1
                    F.add(room_num)
                room = F.getFreeRoom()
                S.schedule(c, room)
                B.add(room)
    return S
```

F = Unoccupied Rooms

B = Occupied Rooms

S = course -> room Map



$F = \{\}$

$B = \{\}$

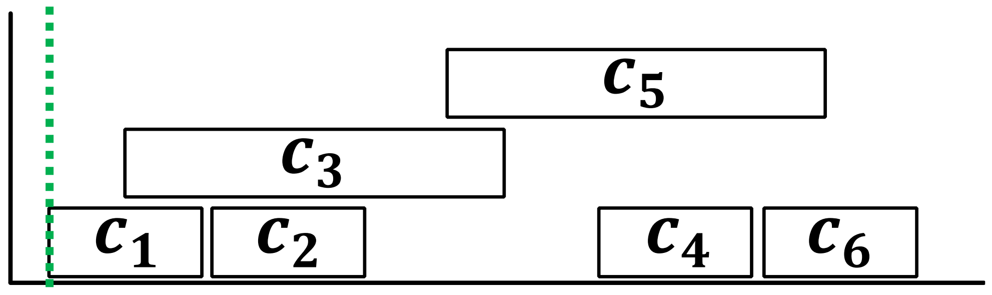
$S = \{\}$

```
room_minimization(courses C)
    F = B = S = ∅
    room_num = 0
    foreach timeslot t
        foreach c in C
            if c.finish == t
                F.add(B.getBookedRoom(S.getroom(c)))
        foreach c in C
            if c.start == t
                if F.isEmpty()
                    room_num += 1
                    F.add(room_num)
                room = F.getFreeRoom()
                S.schedule(c, room)
                B.add(room)
    return S
```

F = Unoccupied Rooms

B = Occupied Rooms

S = course -> room Map



$F = \{\}$

$B = \{\}$

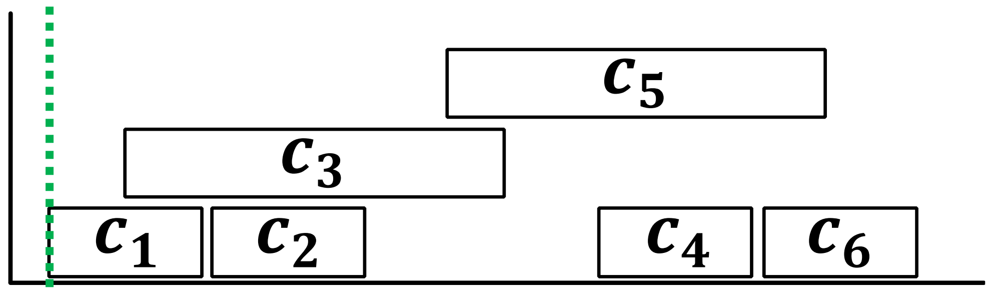
$S = \{\}$

```
room_minimization(courses C)
    F = B = S = ∅
    room_num = 0
    foreach timeslot t
        foreach c in C
            if c.finish == t
                F.add(B.getBookedRoom(S.getroom(c)))
        foreach c in C
            if c.start == t
                if F.isEmpty()
                    room_num += 1
                    F.add(room_num)
                room = F.getFreeRoom()
                S.schedule(c, room)
                B.add(room)
    return S
```

F = Unoccupied Rooms

B = Occupied Rooms

S = course -> room Map



$F = \{1\}$

$B = \{\}$

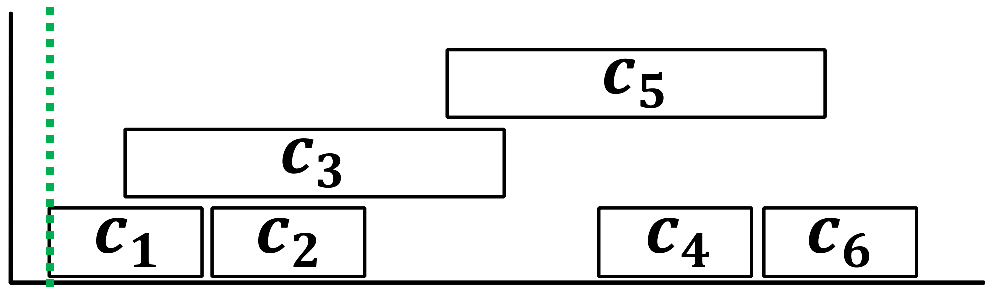
$S = \{\}$

```
room_minimization(courses C)
    F = B = S = ∅
    room_num = 0
    foreach timeslot t
        foreach c in C
            if c.finish == t
                F.add(B.getBookedRoom(S.getroom(c)))
        foreach c in C
            if c.start == t
                if F.isEmpty()
                    room_num += 1
                    F.add(room_num)
                room = F.getFreeRoom()
                S.schedule(c, room)
                B.add(room)
    return S
```

F = Unoccupied Rooms

B = Occupied Rooms

S = course -> room Map



$F = \{\}$

$B = \{1\}$

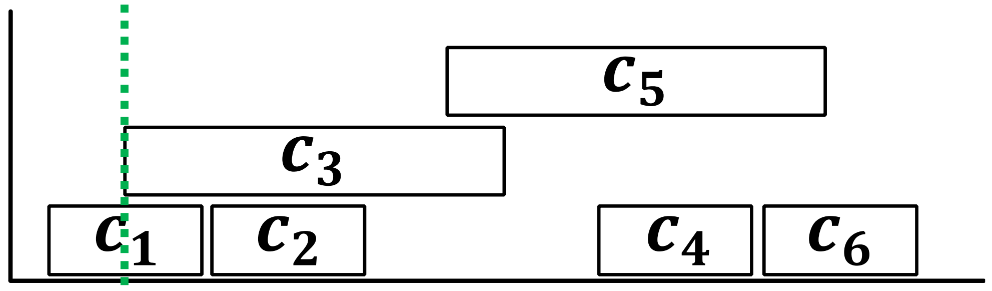
$S = \{c_1 \rightarrow 1\}$

```
room_minimization(courses C)
    F = B = S = Ø
    room_num = 0
    foreach timeslot t
        foreach c in C
            if c.finish == t
                F.add(B.getBookedRoom(S.getroom(c)))
        foreach c in C
            if c.start == t
                if F.isEmpty()
                    room_num += 1
                    F.add(room_num)
                room = F.getFreeRoom()
                S.schedule(c, room)
                B.add(room)
    return S
```

F = Unoccupied Rooms

B = Occupied Rooms

S = course -> room Map



$F = \{\}$

$B = \{1\}$

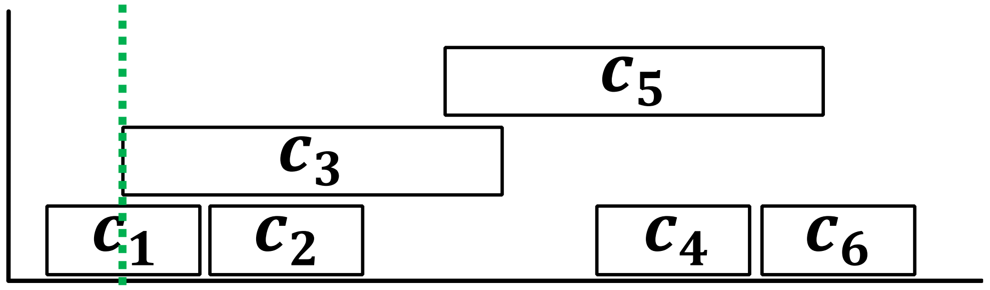
$S = \{c_1 \rightarrow 1\}$

```
room_minimization(courses C)
    F = B = S = ∅
    room_num = 0
    foreach timeslot t
        foreach c in C
            if c.finish == t
                F.add(B.getBookedRoom(S.getroom(c)))
        foreach c in C
            if c.start == t
                if F.isEmpty()
                    room_num += 1
                    F.add(room_num)
                room = F.getFreeRoom()
                S.schedule(c, room)
                B.add(room)
    return S
```

F = Unoccupied Rooms

B = Occupied Rooms

S = course -> room Map



$F = \{\}$

$B = \{1\}$

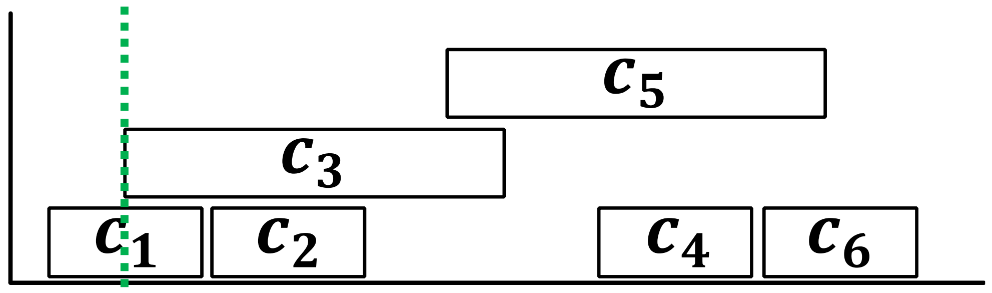
$S = \{c_1 \rightarrow 1\}$

```
room_minimization(courses C)
    F = B = S = ∅
    room_num = 0
    foreach timeslot t
        foreach c in C
            if c.finish == t
                F.add(B.getBookedRoom(S.getroom(c)))
        foreach c in C
            if c.start == t
                if F.isEmpty()
                    room_num += 1
                    F.add(room_num)
                room = F.getFreeRoom()
                S.schedule(c, room)
                B.add(room)
    return S
```

F = Unoccupied Rooms

B = Occupied Rooms

S = course -> room Map



$F = \{\}$

$B = \{1\}$

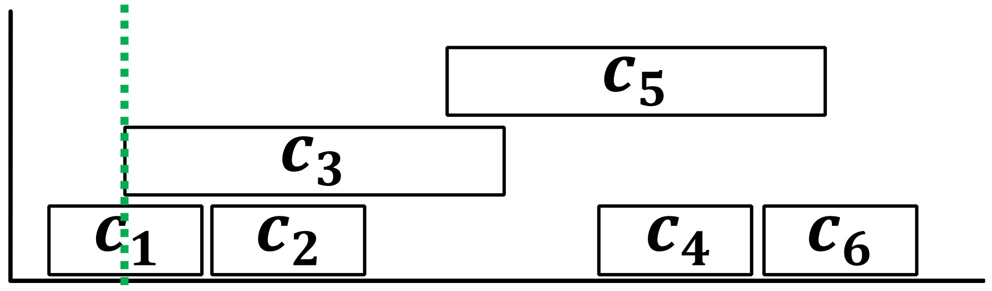
$S = \{c_1 \to 1\}$

```
room_minimization(courses C)
    F = B = S = ∅
    room_num = 0
    foreach timeslot t
        foreach c in C
            if c.finish == t
                F.add(B.getBookedRoom(S.getroom(c)))
        foreach c in C
            if c.start == t
                if F.isEmpty()
                    room_num += 1
                    F.add(room_num)
                room = F.getFreeRoom()
                S.schedule(c, room)
                B.add(room)
    return S
```

F = Unoccupied Rooms

B = Occupied Rooms

S = course -> room Map



$F = \{2\}$

$B = \{1\}$

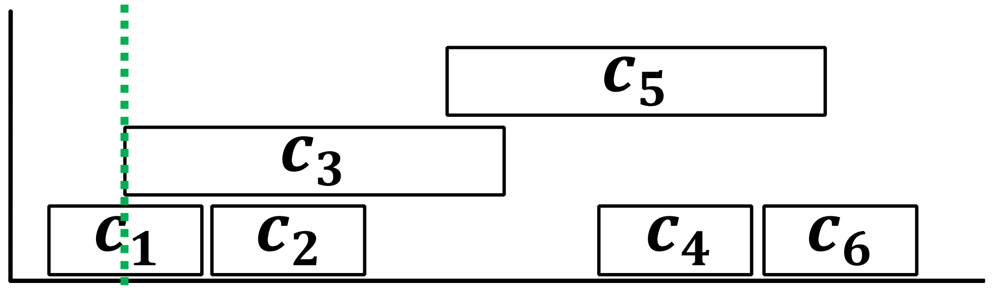
$S = \{c_1 \to 1\}$

```
room_minimization(courses C)
    F = B = S = ∅
    room_num = 0
    foreach timeslot t
        foreach c in C
            if c.finish == t
                F.add(B.getBookedRoom(S.getroom(c)))
        foreach c in C
            if c.start == t
                if F.isEmpty()
                    room_num += 1
                    F.add(room_num)
                room = F.getFreeRoom()
                S.schedule(c, room)
                B.add(room)
    return S
```

F = Unoccupied Rooms

B = Occupied Rooms

S = course -> room Map



$F = \{\}$

$B = \{1,2\}$

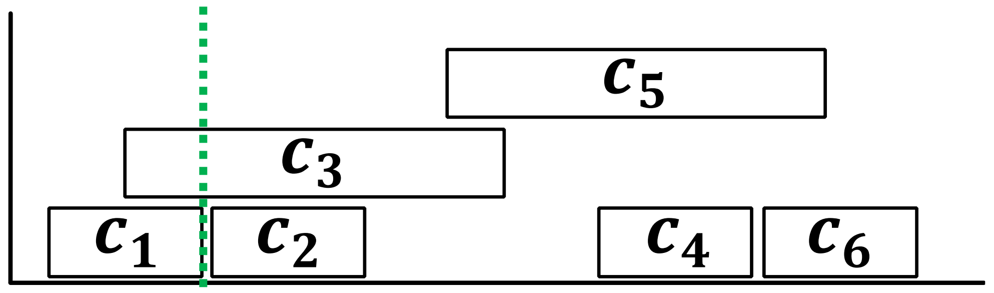
$S = \{c_1 \rightarrow 1, c_3 \rightarrow 2\}$

```
room_minimization(courses C)
    F = B = S = ∅
    room_num = 0
    foreach timeslot t
        foreach c in C
            if c.finish == t
                F.add(B.getBookedRoom(S.getroom(c)))
        foreach c in C
            if c.start == t
                if F.isEmpty()
                    room_num += 1
                    F.add(room_num)
                room = F.getFreeRoom()
                S.schedule(c, room)
                B.add(room)
    return S
```

F = Unoccupied Rooms

B = Occupied Rooms

S = course -> room Map



$F = \{\}$

$B = \{1,2\}$

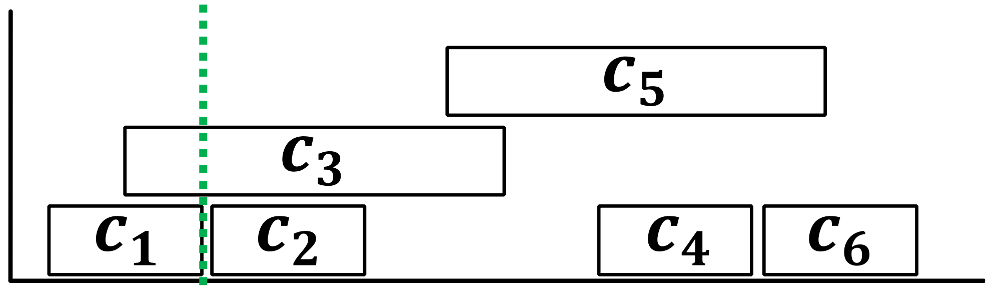
$S = \{c_1 \rightarrow 1, c_3 \rightarrow 2\}$

```
room_minimization(courses C)
    F = B = S = ∅
    room_num = 0
    foreach timeslot t
        foreach c in C
            if c.finish == t
                F.add(B.getBookedRoom(S.getroom(c)))
        foreach c in C
            if c.start == t
                if F.isEmpty()
                    room_num += 1
                    F.add(room_num)
                room = F.getFreeRoom()
                S.schedule(c, room)
                B.add(room)
    return S
```

F = Unoccupied Rooms

B = Occupied Rooms

S = course -> room Map



$F = \{\}$

$B = \{1,2\}$

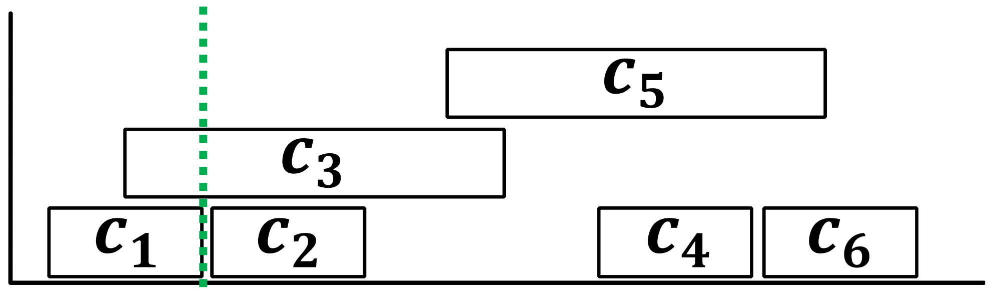
$S = \{c_1 \rightarrow 1, c_3 \rightarrow 2\}$

```
room_minimization(courses C)
    F = B = S = ∅
    room_num = 0
    foreach timeslot t
        foreach c in C
            if c.finish == t
                F.add(B.getBookedRoom(S.getroom(c)))
        foreach c in C
            if c.start == t
                if F.isEmpty()
                    room_num += 1
                    F.add(room_num)
                room = F.getFreeRoom()
                S.schedule(c, room)
                B.add(room)
    return S
```

F = Unoccupied Rooms

B = Occupied Rooms

S = course -> room Map



$F = \{1\}$

$B = \{2\}$

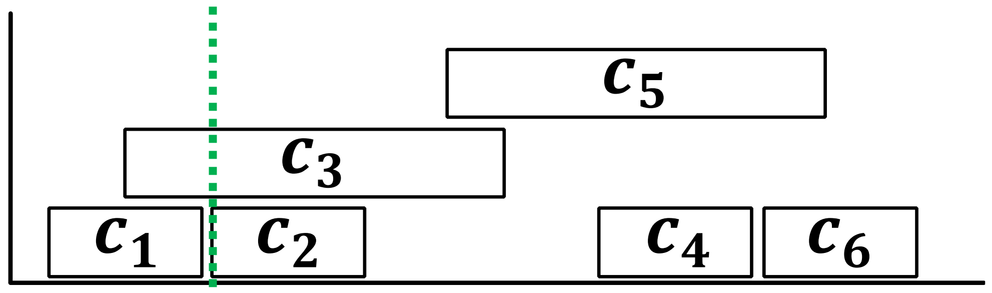
$S = \{c_1 \to 1, c_3 \to 2\}$

```
room_minimization(courses C)
    F = B = S = ∅
    room_num = 0
    foreach timeslot t
        foreach c in C
            if c.finish == t
                F.add(B.getBookedRoom(S.getroom(c)))
        foreach c in C
            if c.start == t
                if F.isEmpty()
                    room_num += 1
                    F.add(room_num)
                room = F.getFreeRoom()
                S.schedule(c, room)
                B.add(room)
    return S
```

F = Unoccupied Rooms

B = Occupied Rooms

S = course -> room Map



$F = \{1\}$

$B = \{2\}$

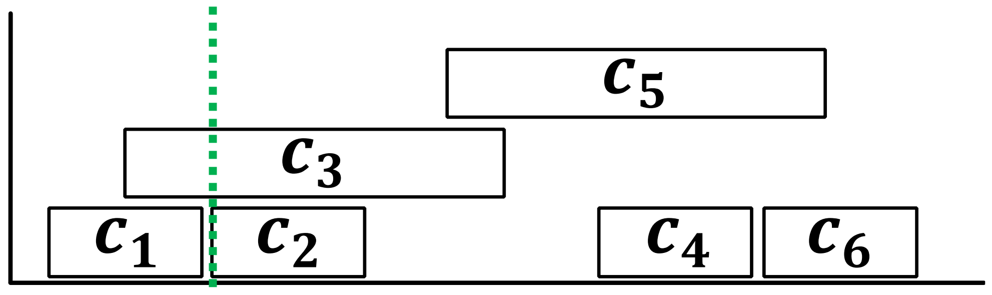
$S = \{c_1 \to 1, c_3 \to 2\}$

```
room_minimization(courses C)
    F = B = S = Ø
    room_num = 0
    foreach timeslot t
        foreach c in C
            if c.finish == t
                F.add(B.getBookedRoom(S.getroom(c)))
        foreach c in C
            if c.start == t
                if F.isEmpty()
                    room_num += 1
                    F.add(room_num)
                room = F.getFreeRoom()
                S.schedule(c, room)
                B.add(room)
    return S
```

F = Unoccupied Rooms

B = Occupied Rooms

S = course -> room Map



$F = \{\}$

$B = \{2,1\}$

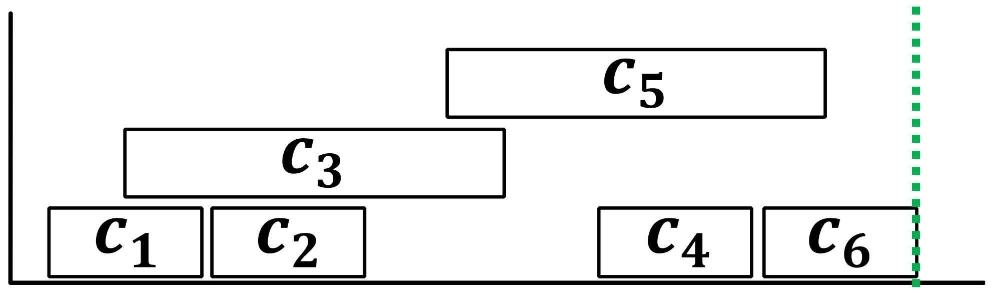
$S = \{c_1 \to 1, c_3 \to 2, c_2 \to 1\}$

```
room_minimization(courses C)
    F = B = S = ∅
    room_num = 0
    foreach timeslot t
        foreach c in C
            if c.finish == t
                F.add(B.getBookedRoom(S.getroom(c)))
        foreach c in C
            if c.start == t
                if F.isEmpty()
                    room_num += 1
                    F.add(room_num)
                room = F.getFreeRoom()
                S.schedule(c, room)
                B.add(room)
    return S
```

F = Unoccupied Rooms

B = Occupied Rooms

S = course -> room Map



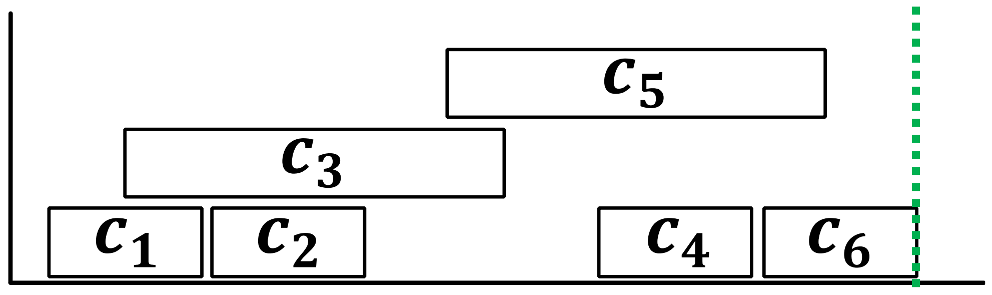
$F = \{1,2\}$

$B = \{\}$

$S = \{c_1 \rightarrow 1, c_3 \rightarrow 2, c_2 \rightarrow 1, \dots\}$

```
room_minimization(courses C)
    F = B = S = ∅
    room_num = 0
    foreach timeslot t
        foreach c in C
            if c.finish == t
                F.add(B.getBookedRoom(S.getroom(c)))
        foreach c in C
            if c.start == t
                if F.isEmpty()
                    room_num += 1
                    F.add(room_num)
                room = F.getFreeRoom()
                S.schedule(c, room)
                B.add(room)
    return S
```

Valid?



$F = \{1,2\}$

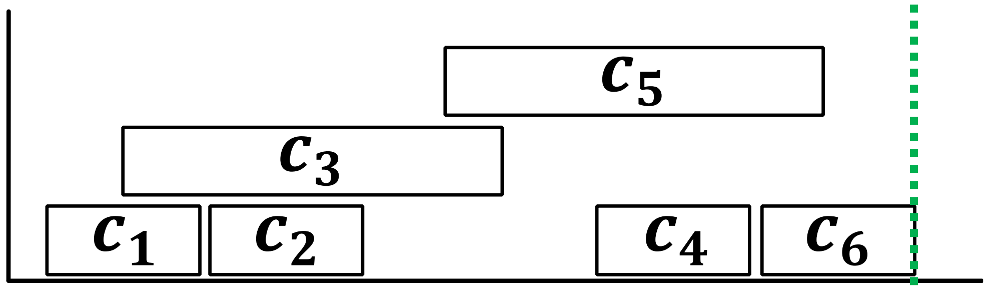
$B = \{\}$

$S = \{c_1 \to 1, c_3 \to 2, c_2 \to 1, \ldots\}$

```
room_minimization(courses C)
    F = B = S = ∅
    room_num = 0
    foreach timeslot t
        foreach c in C
            if c.finish == t
                F.add(B.getBookedRoom(S.getroom(c)))
        foreach c in C
            if c.start == t
                if F.isEmpty()
                    room_num += 1
                    F.add(room_num)
                room = F.getFreeRoom()
                S.schedule(c, room)
                B.add(room)
    return S
```

Valid?

Yes. Each course is given a room and no rooms are concurrently occupied.



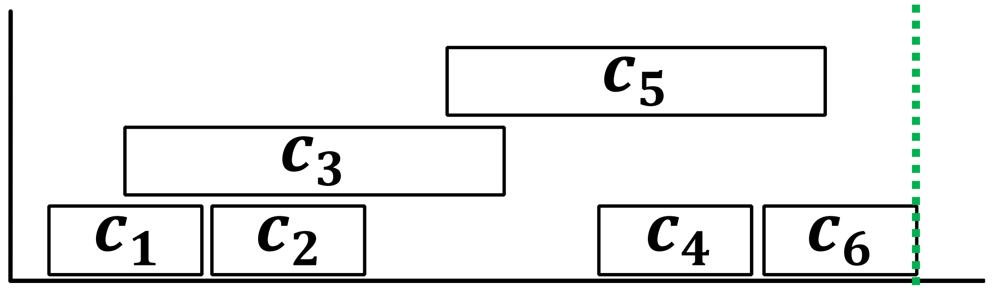
$F = \{1,2\}$

$B = \{\}$

$S = \{c_1 \to 1, c_3 \to 2, c_2 \to 1, \ldots\}$

```
room_minimization(courses C)
    F = B = S = ∅
    room_num = 0
    foreach timeslot t
        foreach c in C
            if c.finish == t
                F.add(B.getBookedRoom(S.getroom(c)))
        foreach c in C
            if c.start == t
                if F.isEmpty()
                    room_num += 1
                    F.add(room_num)
                room = F.getFreeRoom()
                S.schedule(c, room)
                B.add(room)
    return S
```

Running Time?



$F = \{1,2\}$

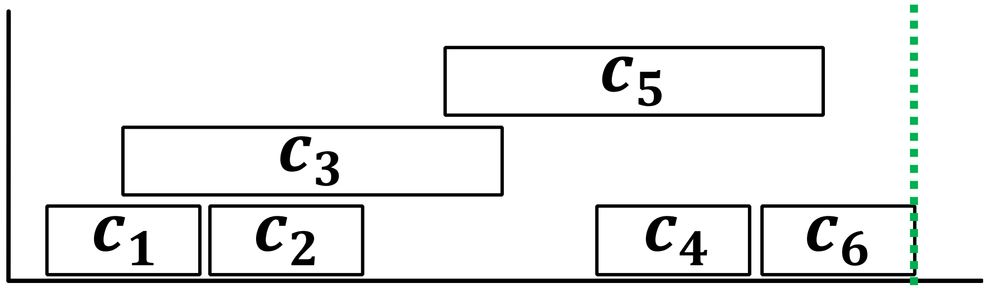
$B = \{\}$

$S = \{c_1 \to 1, c_3 \to 2, c_2 \to 1, \dots\}$

```
room_minimization(courses C)
    F = B = S = ∅
    room_num = 0
    foreach timeslot t
        foreach c in C
            if c.finish == t
                F.add(B.getBookedRoom(S.getroom(c)))
        foreach c in C
            if c.start == t
                if F.isEmpty()
                    room_num += 1
                    F.add(room_num)
                room = F.getFreeRoom()
                S.schedule(c, room)
                B.add(room)
    return S
```

Running Time?

$O(|C|^2)$ – Don't need to go through all timeslots. Just start/finish for each activity.



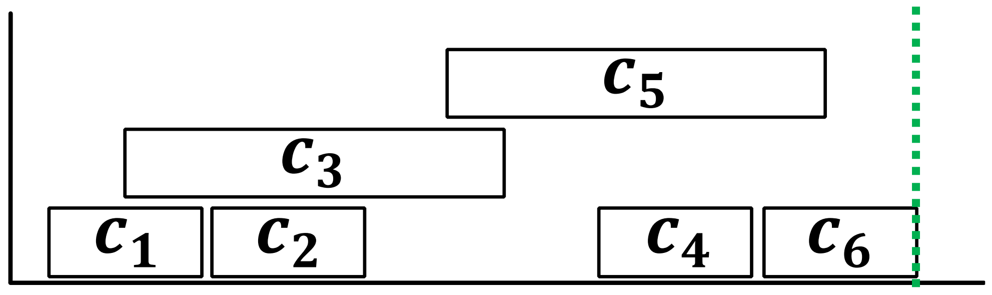
$F = \{1,2\}$

$B = \{\}$

$S = \{c_1 \rightarrow 1, c_3 \rightarrow 2, c_2 \rightarrow 1, \ldots\}$

```
room_minimization(courses C)
    F = B = S = Ø
    room_num = 0
    foreach timeslot t
        foreach c in C
            if c.finish == t
                F.add(B.getBookedRoom(S.getroom(c)))
        foreach c in C
            if c.start == t
                if F.isEmpty()
                    room_num += 1
                    F.add(room_num)
                room = F.getFreeRoom()
                S.schedule(c, room)
                B.add(room)
    return S
```

Optimal?



$F = \{1,2\}$

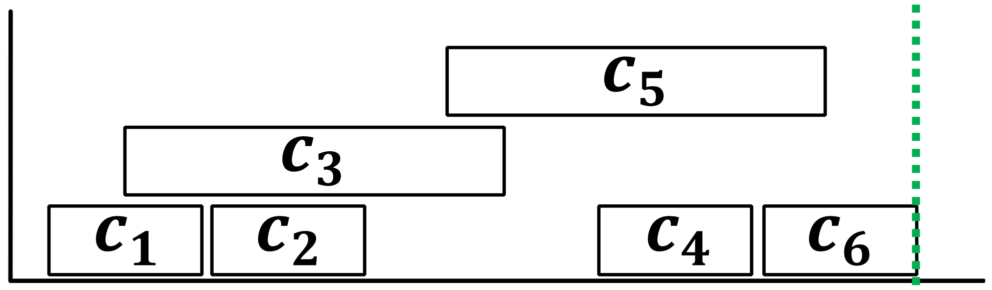
$B = \{\}$

$S = \{c_1 \rightarrow 1, c_3 \rightarrow 2, c_2 \rightarrow 1, \ldots\}$

```
room_minimization(courses C)
    F = B = S = ∅
    room_num = 0
    foreach timeslot t
        foreach c in C
            if c.finish == t
                F.add(B.getBookedRoom(S.getroom(c)))
        foreach c in C
            if c.start == t
                if F.isEmpty()
                    room_num += 1
                    F.add(room_num)
                room = F.getFreeRoom()
                S.schedule(c, room)
                B.add(room)
    return S
```

Optimal?

# rooms needed ≥ # concurrent courses.



$F = \{1,2\}$

$B = \{\}$

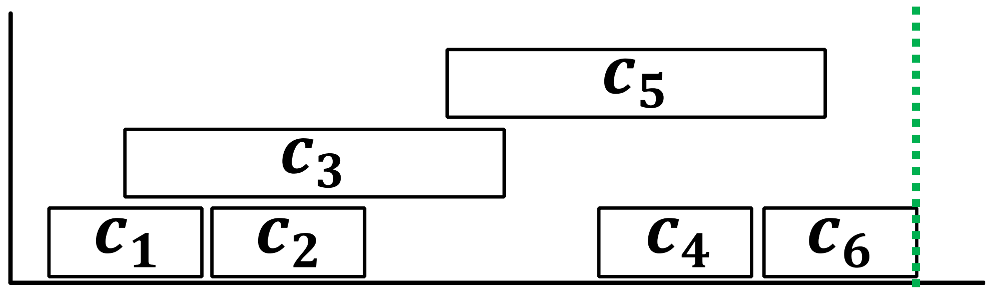
$S = \{c_1 \rightarrow 1, c_3 \rightarrow 2, c_2 \rightarrow 1, \ldots\}$

```
room_minimization(courses C)
    F = B = S = ∅
    room_num = 0
    foreach timeslot t
        foreach c in C
            if c.finish == t
                F.add(B.getBookedRoom(S.getroom(c)))
        foreach c in C
            if c.start == t
                if F.isEmpty()
                    room_num += 1
                    F.add(room_num)
                room = F.getFreeRoom()
                S.schedule(c, room)
                B.add(room)
    return S
```

Optimal?

# rooms needed ≥ # concurrent courses.

New room added ⇔ all other rooms in use



$F = \{1,2\}$

$B = \{\}$

$S = \{c_1 \to 1, c_3 \to 2, c_2 \to 1, \ldots\}$

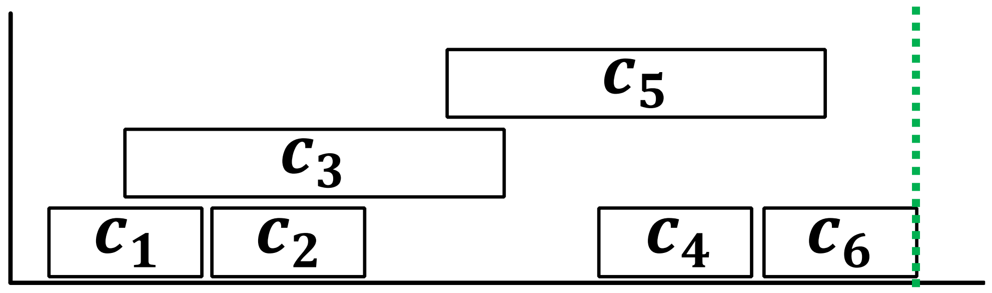
```
room_minimization(courses C)
    F = B = S = ∅
    room_num = 0
    foreach timeslot t
        foreach c in C
            if c.finish == t
                F.add(B.getBookedRoom(S.getroom(c)))
        foreach c in C
            if c.start == t
                if F.isEmpty()
                    room_num += 1
                    F.add(room_num)
                room = F.getFreeRoom()
                S.schedule(c, room)
                B.add(room)
    return S
```

Optimal?

# rooms needed ≥ # concurrent courses.

New room added ⇔ all other rooms in use

(i.e., # rooms used = max[# concurrent courses])



$F = \{1,2\}$

$B = \{\}$

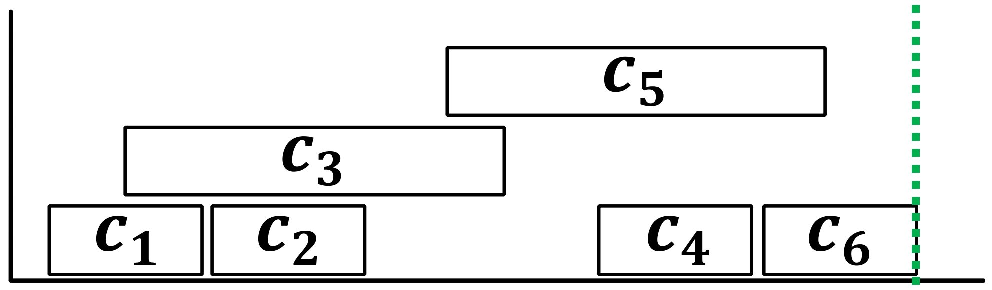
$S = \{c_1 \rightarrow 1, c_3 \rightarrow 2, c_2 \rightarrow 1, \ldots\}$

```
room_minimization(courses C)
    F = B = S = ∅
    room_num = 0
    foreach timeslot t
        foreach c in C
            if c.finish == t
                F.add(B.getBookedRoom(S.getroom(c)))
        foreach c in C
            if c.start == t
                if F.isEmpty()
                    room_num += 1
                    F.add(room_num)
                room = F.getFreeRoom()
                S.schedule(c, room)
                B.add(room)
    return S
```

Optimal?

# rooms needed ≥ # concurrent courses.
New room added ⇔ all other rooms in use
(i.e., # rooms used = max[# concurrent courses])

So, yes.



$F = \{1,2\}$

$B = \{\}$

$S = \{c_1 \to 1, c_3 \to 2, c_2 \to 1, \ldots\}$

Client Scheduling

Suppose you are a plumber and you have a list of clients that want help.

Client Scheduling

- d_i: deadline for client i.
- t_i: time required for client i.

Suppose you are a plumber and you have a list of clients that want help.

Each client i has a deadline d_i of when they need help by and an amount of time t_i they will need help for.

Client	$\boldsymbol{d_i}$	$\boldsymbol{t_i}$
1	5	3
2	6	4
3	8	3

Client Scheduling

- d_i: deadline for client i.
- t_i: time required for client i.

Client	$\boldsymbol{d_i}$	$\boldsymbol{t_i}$
1	5	3
2	6	4
3	8	3

Suppose you are a plumber and you have a list of clients that want help.

Each client i has a deadline d_i of when they need help by and an amount of time t_i they will need help for.

You cannot help multiple clients at the same time and cannot pause helping one to help another.

Client Scheduling

- d_i: deadline for client i.
- t_i: time required for client i.

Client	$\boldsymbol{d_i}$	$\boldsymbol{t_i}$
1	5	3
2	6	4
3	8	3

Suppose you are a plumber and you have a list of clients that want help.

Each client i has a deadline d_i of when they need help by and an amount of time t_i they will need help for.

You cannot help multiple clients at the same time and cannot pause helping one to help another.

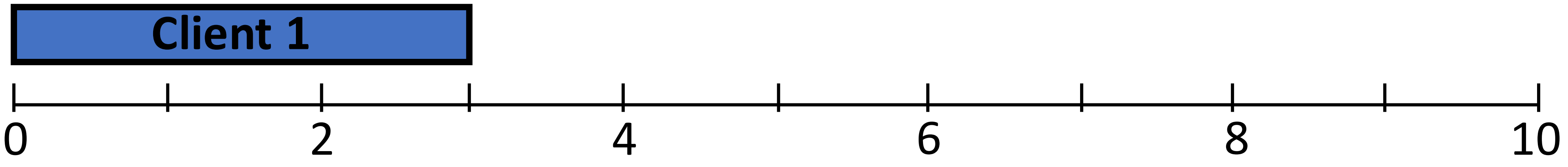
You need to help all clients, even if it goes over their deadline (though they will be angry then).

Client Scheduling

- d_i: deadline for client i.
- t_i: time required for client i.
- $s(i)$: start time for client i.

Client	$\boldsymbol{d_i}$	$\boldsymbol{t_i}$
1	5	3
2	6	4
3	8	3

Schedule:



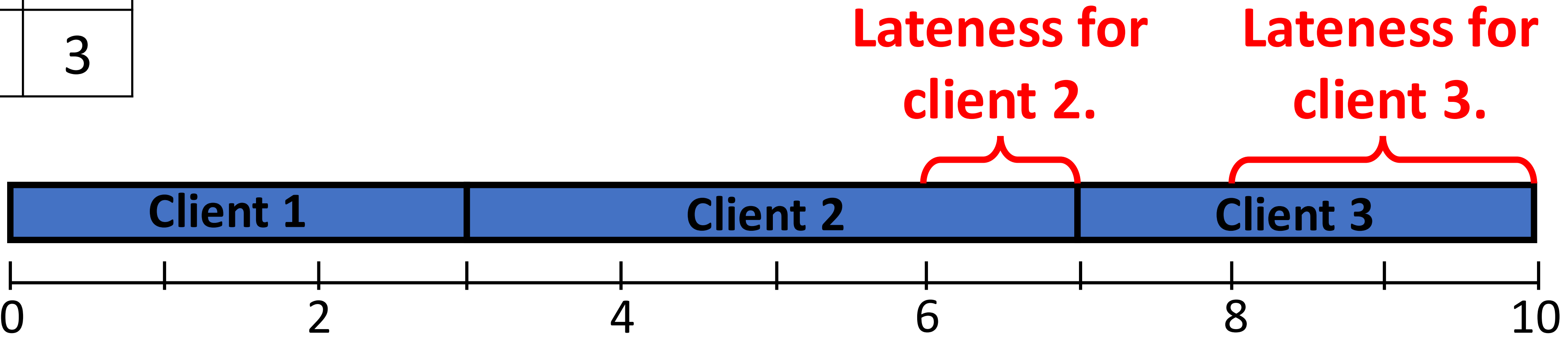
Client Scheduling

- d_i: deadline for client i.
- t_i: time required for client i.
- $s(i)$: start time for client i.
- $f(i) = s(i) + t_i$: finish time for client i.
- $l_i = f(i) - d_i$: lateness for client i.

Client	d_i	t_i
1	5	3
2	6	4
3	8	3

The lateness of a client is the amount of past their deadline their request took.

Schedule:



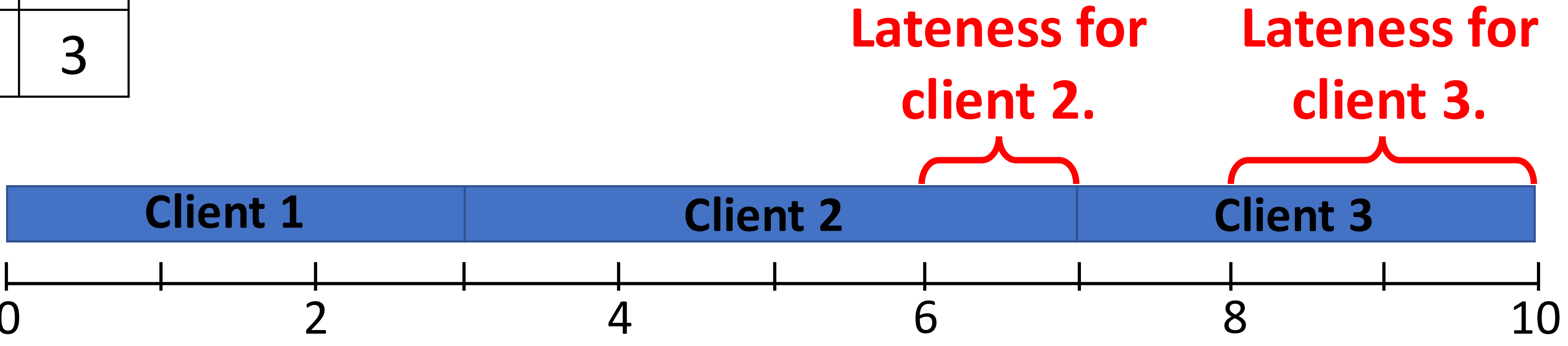
Client Scheduling

- d_i: deadline for client i.
- t_i: time required for client i.
- $s(i)$: start time for client i.
- $f(i) = s(i) + t_i$: finish time for client i.
- $l_i = f(i) - d_i$: lateness for client i.
- $L = \max_i l_i$: maximum lateness.

Client	d_i	t_i
1	5	3
2	6	4
3	8	3

We want a schedule that minimizes the lateness of the latest client.

Schedule:



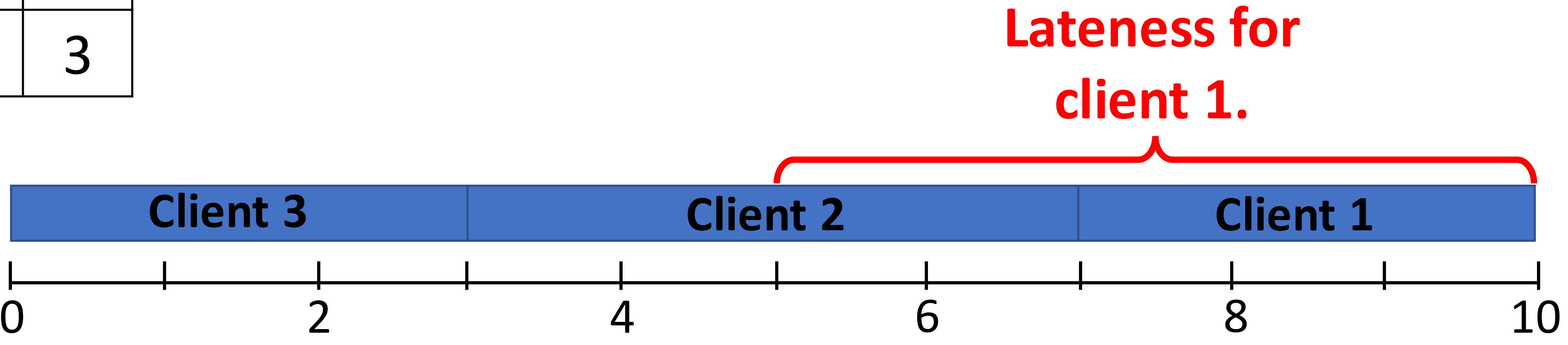
Client Scheduling

- d_i: deadline for client i.
- t_i: time required for client i.
- $s(i)$: start time for client i.
- $f(i) = s(i) + t_i$: finish time for client i.
- $l_i = f(i) - d_i$: lateness for client i.
- $L = \max_i l_i$: maximum lateness.

Client	d_i	t_i
1	5	3
2	6	4
3	8	3

We want a schedule that minimizes the lateness of the latest client.

Schedule:



Greedy Decision

What are some possible Greedy decisions?

Greedy Decision

What are some possible Greedy decisions?

- Smallest t_i first.
- Smallest slack time $(d_i - t_i)$ first.
- Earliest d_i first.

Greedy Decision

What are some possible Greedy decisions?

- Smallest t_i first.

- Smallest slack time $(d_i - t_i)$ first.

- Earliest d_i first.

How to decide which to use?

Greedy Decision

What are some possible Greedy decisions?

- Smallest t_i first.

- Smallest slack time $(d_i - t_i)$ first.

- Earliest d_i first.

How to decide which to use?
Hunt for counterexamples.

Greedy Decision

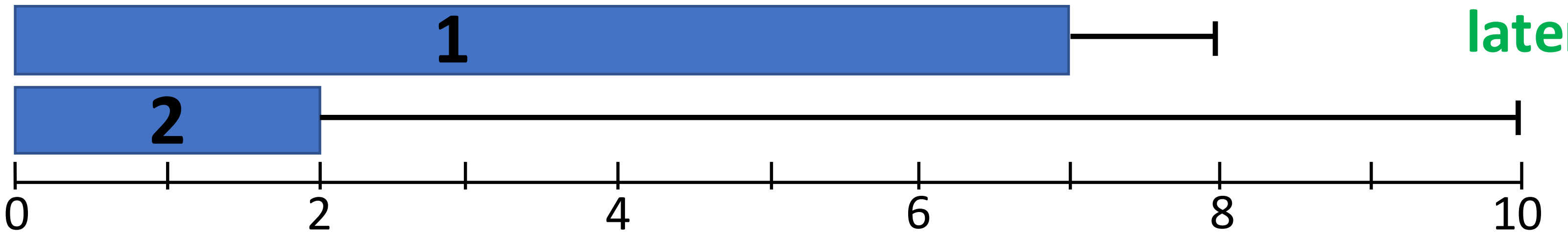
What are some possible Greedy decisions?

Algorithm: 2 then 1, lateness of 1.

Optimal: 1 then 2, lateness of 0.

- Smallest t_i first.

Client	d_i	t_i
1	8	7
2	10	2



- Smallest slack time $(d_i - t_i)$ first.

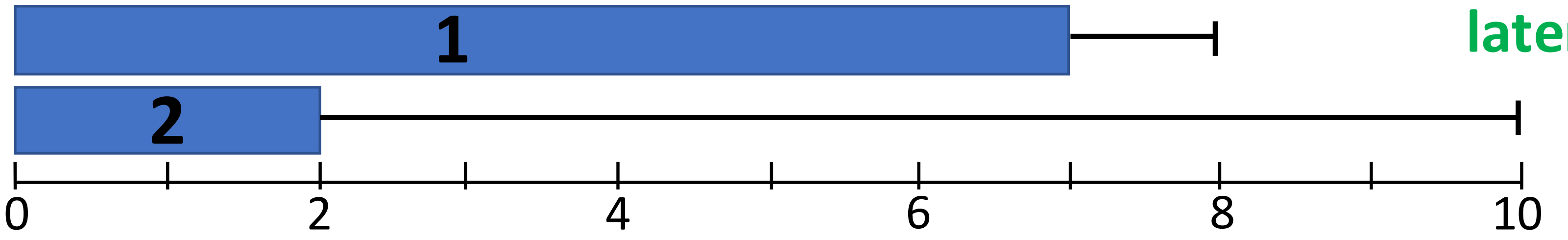
- Earliest d_i first.

Greedy Decision

What are some possible Greedy decisions?

- Smallest t_i first.

Client	d_i	t_i
1	8	7
2	10	2

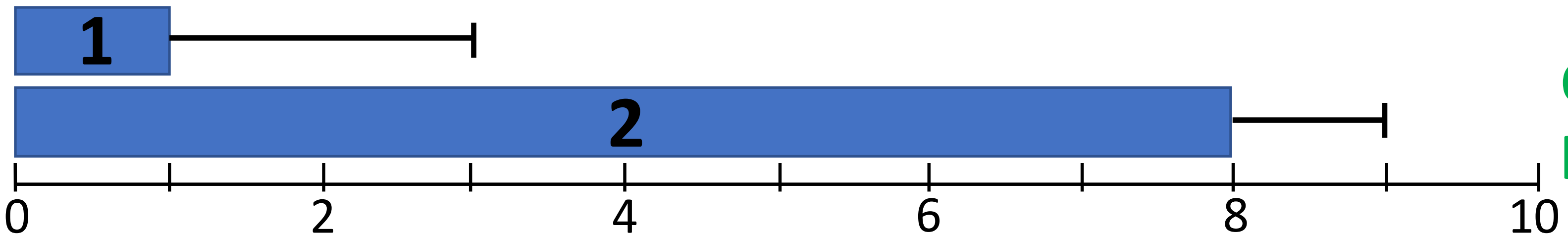


Algorithm: 2 then 1, lateness of 1.

Optimal: 1 then 2, lateness of 0.

- Smallest slack time $(d_i - t_i)$ first.

Client	d_i	t_i
1	3	1
2	9	8



Algorithm: 2 then 1, lateness of 6.

Optimal: 1 then 2, lateness of 0.

- Earliest d_i first.

Earliest Deadline First Algorithm

1. Order clients by increasing deadline.
2. Rename so that $d_1 \leq \cdots \leq d_n$.
3. Let $s(1) = 0$. (implies that $f(1) = t_1$)
4. For each subsequent (in order) client c,

   $s(c) = f(c-1)$ and $f(c) = s(c) + t_c$.

Earliest Deadline First Algorithm

Theorem: The maximum lateness given by a schedule from the EDF algorithm is optimal.

Plan of attack:

Earliest Deadline First Algorithm

Theorem: The maximum lateness given by a schedule from the EDF algorithm is optimal.

Plan of attack: Consider an optimal schedule, modify it in such a way that optimality is preserved until it is the same as our schedule.

How could our schedule differ from optimal?

Earliest Deadline First Algorithm

Theorem: The maximum lateness given by a schedule from the EDF algorithm is optimal.

Plan of attack: Consider an optimal schedule, modify it in such a way that optimality is preserved until it is the same as our schedule.

How could our schedule differ from optimal?

1. Gaps in schedules.
2. Clients out of order.

Earliest Deadline First Algorithm

1. Order clients by increasing deadline.
2. Rename so that $d_1 \leq \cdots \leq d_n$.
3. Let $s(1) = 0$. (implies that $f(1) = t_1$)
4. For each subsequent (in order) client c,

   $s(c) = f(c-1)$ and $f(c) = s(c) + t_c$.

Could there be gaps in our schedule?

Earliest Deadline First Algorithm

1. Order clients by increasing deadline.
2. Rename so that $d_1 \leq \cdots \leq d_n$.
3. Let $s(1) = 0$. (implies that $f(1) = t_1$)
4. For each subsequent (in order) client c,

   $s(c) = f(c-1)$ and $f(c) = s(c) + t_c$.

Could there be gaps in our schedule?

No – as soon as one client is finished, $f(c)$, the next client starts, $s(c + 1) = f(c)$.

Earliest Deadline First Algorithm

Lemma: An optimal schedule exists that has no gaps between clients.

Proof: ?

Earliest Deadline First Algorithm

Lemma: An optimal schedule exists that has no gaps between clients.

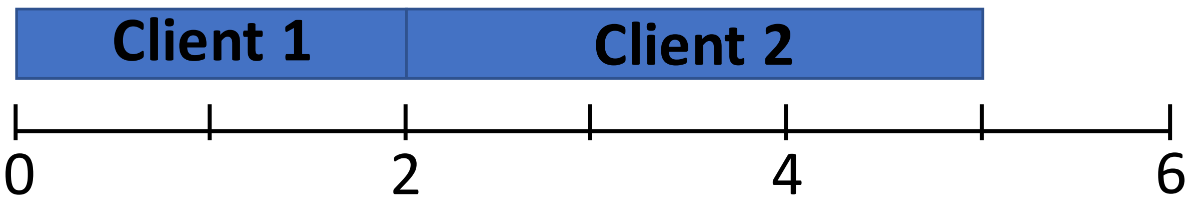
Proof: Since all clients are available to start at the same time, they can be scheduled one after the other without gaps. If an optimal schedule exists with gaps, those gaps can be removed by shifting clients forward without increasing the maximum lateness.

Earliest Deadline First Algorithm

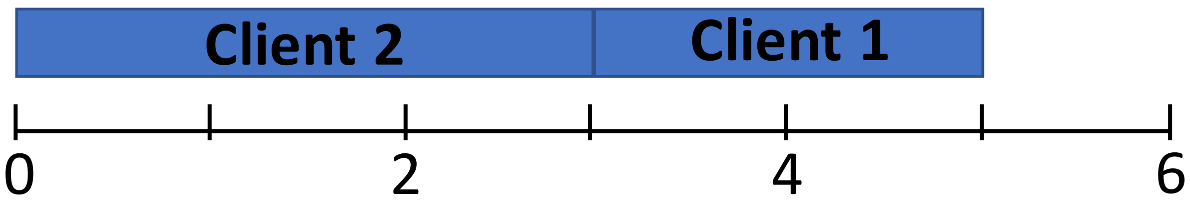
Definition: A schedule has an *inversion* if some client is scheduled before a client with an earlier deadline.

Client 1:	Client 2:
$d_1 = 5$	$d_2 = 6$
$t_1 = 2$	$t_2 = 3$

No Inversion:



Inversion:



Earliest Deadline First Algorithm

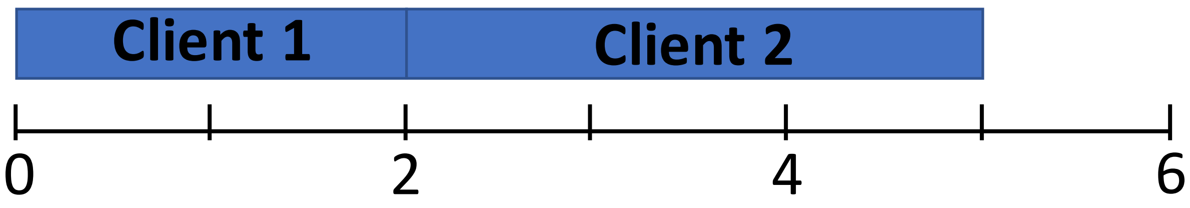
<u>Definition:</u> A schedule has an *inversion* if some client is scheduled before a client with an earlier deadline.

Client 1:
d_1 = 5
t_1 = 2

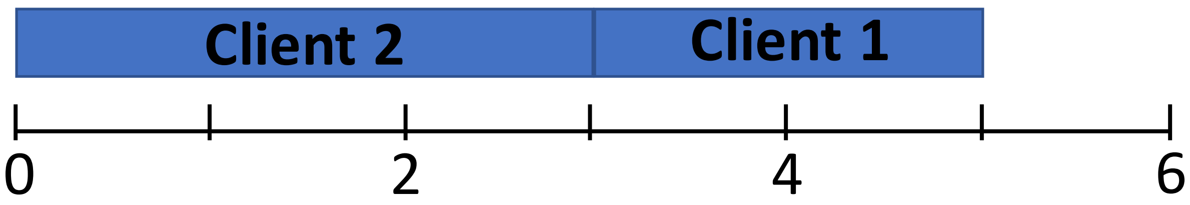
Client 2:
d_2 = 6
t_2 = 3

Does our schedule have any inversions?

No Inversion:



Inversion:



Earliest Deadline First Algorithm

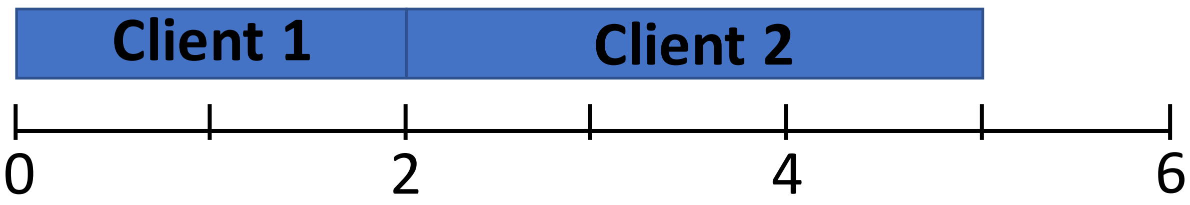
Definition: A schedule has an *inversion* if some client is scheduled before a client with an earlier deadline.

Client 1:	Client 2:
d_1 = 5	d_2 = 6
t_1 = 2	t_2 = 3

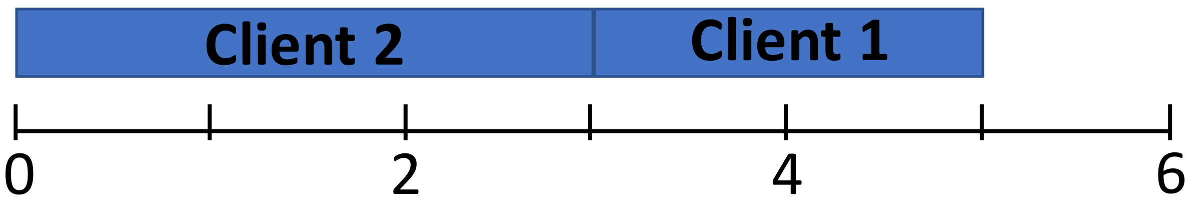
Does our schedule have any inversions?

No – The algorithm schedules client i before client j if $d_i \leq d_j$

No Inversion:



Inversion:



Earliest Deadline First Algorithm

Lemma: An optimal schedule exists that has no inversions.

Proof:

Earliest Deadline First Algorithm

Lemma: An optimal schedule exists that has no inversions.

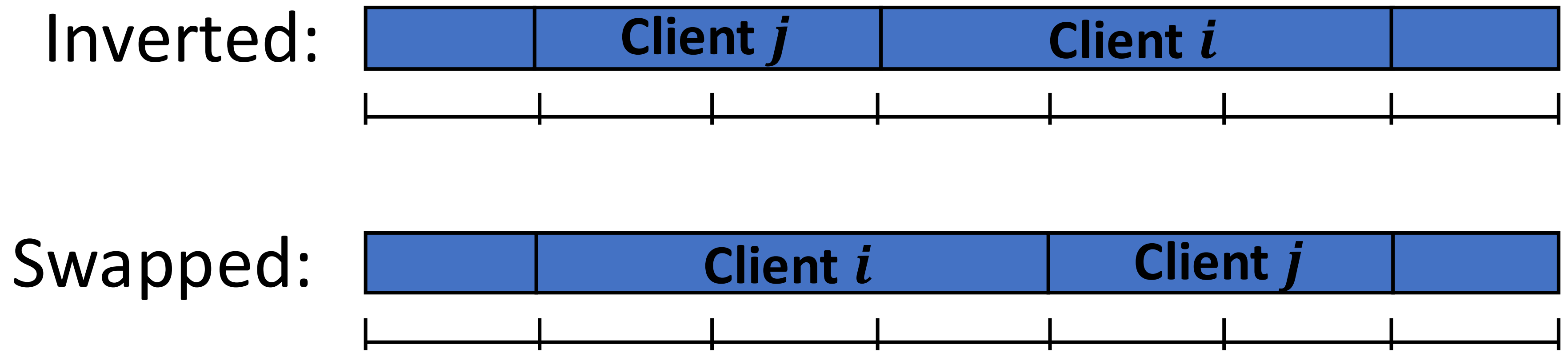
Proof:

1. Any inversion results from two consecutive inverted clients.
2. Swapping an inversion reduces the number of inversions.
3. Swapping an inversion does not increase the maximum lateness of the schedule.

Earliest Deadline First Algorithm

3. Swapping an inversion does not increase the maximum lateness of the schedule.

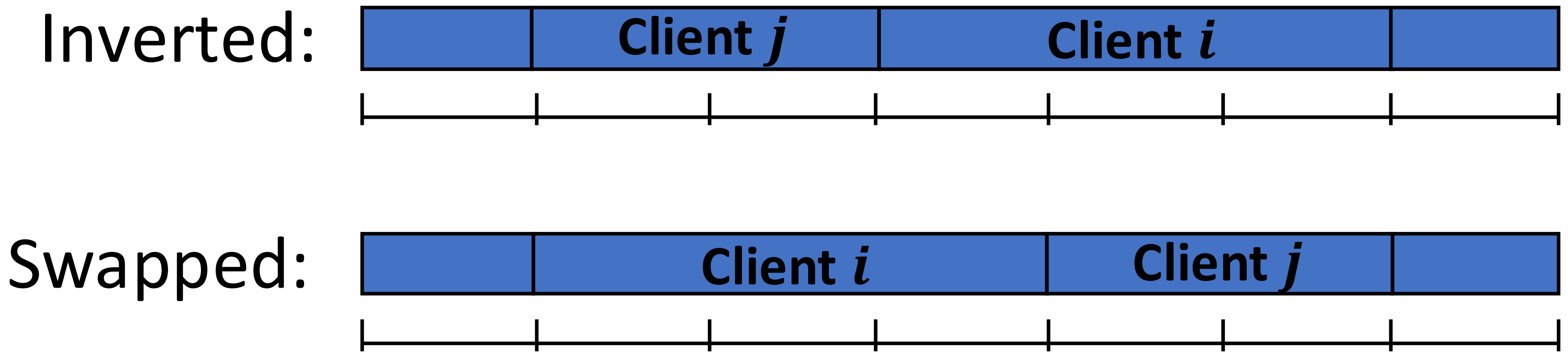
Proof: Whose lateness does swapping consecutive clients affect?



Earliest Deadline First Algorithm

3. Swapping an inversion does not increase the maximum lateness of the schedule.

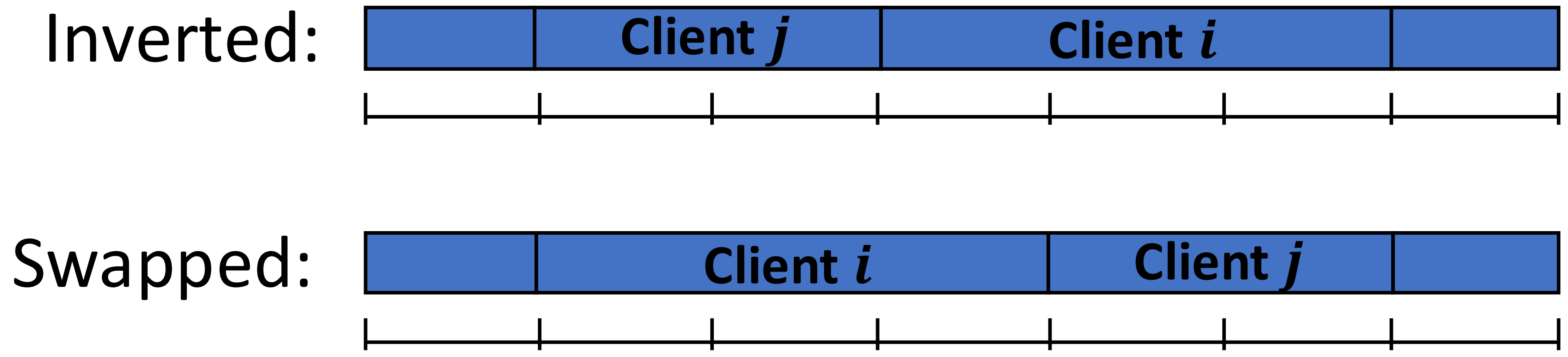
Proof: The lateness is less for the client swapped earlier, later for the client swapped later, and the same for all other clients (because inverted $f(i)$ = swapped $f(j)$).



Earliest Deadline First Algorithm

3. Swapping an inversion does not increase the maximum lateness of the schedule.

Proof: The only client we need to consider is the one swapped to be later (because all other clients remain the same or have smaller lateness).

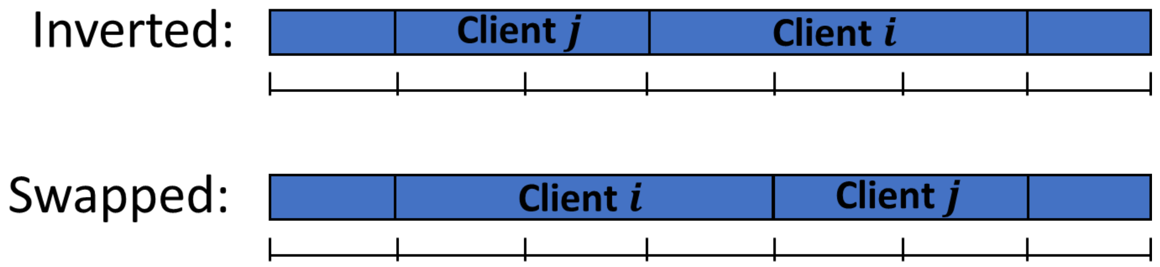


Earliest Deadline First Algorithm

3. Swapping an inversion does not increase the maximum lateness of the schedule.

Proof: The only client we need to consider is the one swapped to be later (because all other clients remain the same or have smaller lateness).

Let $s(i)$, $f(i)$, l_i be for inverted schedule.
Let $s'(i)$, $f'(i)$, l_i' be for swapped schedule.



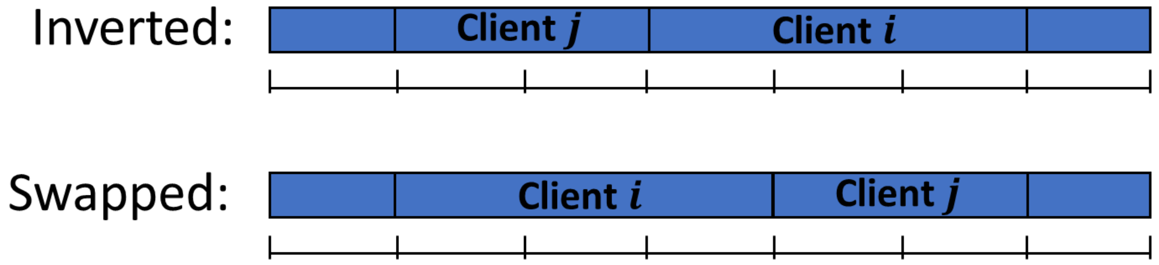
Earliest Deadline First Algorithm

3. Swapping an inversion does not increase the maximum lateness of the schedule.

Proof: The only client we need to consider is the one swapped to be later (because all other clients remain the same or have smaller lateness).

Let $s(i)$, $f(i)$, l_i be for inverted schedule.
Let $s'(i)$, $f'(i)$, l_i' be for swapped schedule.

$$l_j' \;?\; l_i$$



Remember:
- $l_j = f(j) - d_j$
- $d_i \leq d_j$

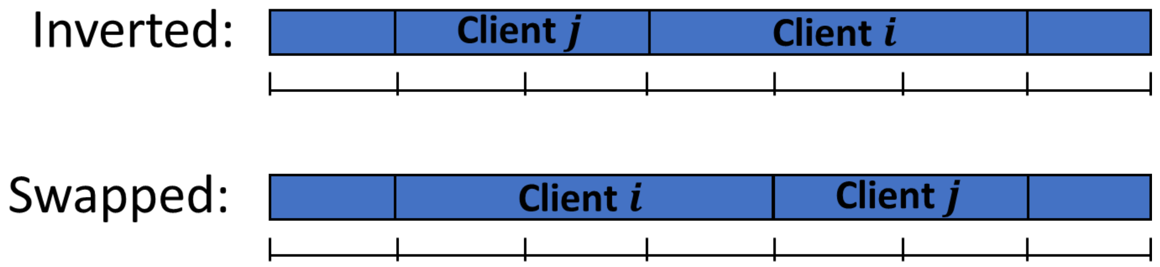
Earliest Deadline First Algorithm

3. Swapping an inversion does not increase the maximum lateness of the schedule.

<u>Proof:</u> The only client we need to consider is the one swapped to be later (because all other clients remain the same or have smaller lateness).

Let $s(i)$, $f(i)$, l_i be for inverted schedule.
Let $s'(i)$, $f'(i)$, $l_i{}'$ be for swapped schedule.

$$l_j{}' = f'(j) - d_j = f(i) - d_j \leq f(i) - d_i = l_i$$



Remember:
- $l_j = f(j) - d_j$
- $d_i \leq d_j$

Earliest Deadline First Algorithm

3. Swapping an inversion does not increase the maximum lateness of the schedule.

<u>Proof:</u> The only client we need to consider is the one swapped to be later (because all other clients remain the same or have smaller lateness).

Let $s(i)$, $f(i)$, l_i be for inverted schedule.
Let $s'(i)$, $f'(i)$, $l_i{}'$ be for swapped schedule.

$$l_j{}' = f'(j) - d_j = f(i) - d_j \leq f(i) - d_i = l_i$$

$\Rightarrow$ The maximum lateness was not increased!

Earliest Deadline First Algorithm

Theorem: The maximum lateness given by a schedule from the EDF algorithm is optimal.

Proof: The EDF schedule can only differ from an optimal schedule by the order of clients with identical deadlines (since both have no gaps or inversions).

Does the ordering of these clients lead to different maximal lateness?

Earliest Deadline First Algorithm

Theorem: The maximum lateness given by a schedule from the EDF algorithm is optimal.

Proof: The EDF schedule can only differ from an optimal schedule by the order of clients with identical deadlines (since both have no gaps or inversions).

Clients with identical deadlines d are all scheduled consecutively. The client with the largest lateness ($l_i = f(i) - d$) is the one with the latest finish time (regardless of order).

Earliest Deadline First Algorithm

Theorem: The maximum lateness given by a schedule from the EDF algorithm is optimal.

Proof: The EDF schedule can only differ from an optimal schedule by the order of clients with identical deadlines (since both have no gaps or inversions).

Clients with identical deadlines d are all scheduled consecutively. The client with the largest lateness ($l_i = f(i) - d$) is the one with the latest finish time (regardless of order).

Thus, all schedules with no inversions or gaps have the same maximal lateness.

Therefore, the EDF schedule is optimal.