Lecture 8: Greedy Algorithms 1

CS 341: Algorithms

Thursday, Jan 31st 2019

Outline For Today

1. Introduction to Greedy Algorithms
2. Activity Selection
3. Job Scheduling 1
4. Job Scheduling 2

Outline For Today

1. Introduction to Greedy Algorithms
2. Activity Selection
3. Job Scheduling 1
4. Job Scheduling 2

Greedy Algorithms

- Algorithms that iteratively make
  - “short-sighted”, “locally optimum looking” decisions
  - hoping to output a good solution (hopefully optimum)
- Example: Coin Changing

Greedy vs Divide-And-Conquer Algorithms

Greedy	Divide and Conquer
easy to design	difficult to design

Greedy vs Divide-And-Conquer Algorithms

Greedy	Divide and Conquer
easy to design	difficult to design
easy to analyze run-time	difficult to analyze run-time

Greedy vs Divide-And-Conquer Algorithms

Greedy	Divide and Conquer
easy to design	difficult to design
easy to analyze run-time	difficult to analyze run-time
difficult to prove correctness	easy to prove correctness

Two Common Correctness Proof Techniques

- Call greedy's solution S_g, and let S be any other solution

1. "Greedy stays ahead"
   - Argue S_g is optimal/better than S *at each step*
   - Proof by induction
2. Exchange Arguments
   - Argue any *S can be transformed into S_g* step by step and *without getting worse along the way*

Warning: They're common but not applicable to every greedy algorithm!

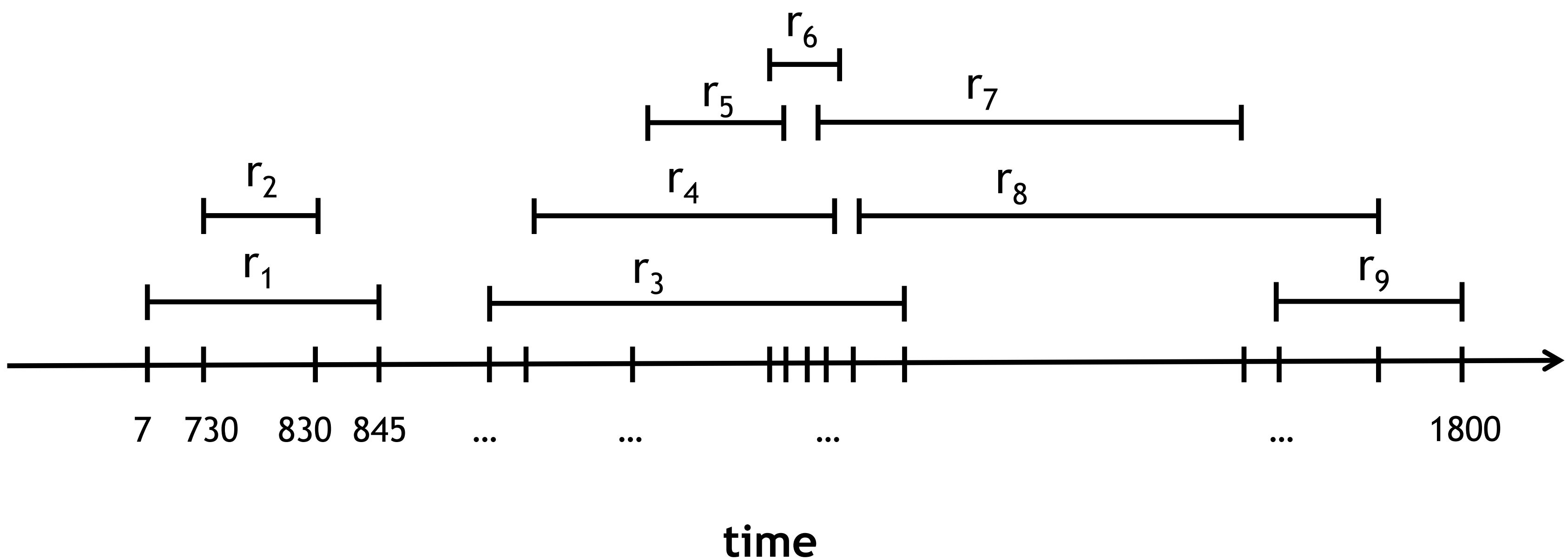
Outline For Today

1. Introduction to Greedy Algorithms
2. Activity Selection
3. Job Scheduling 1
4. Job Scheduling 2

Activity Selection

- Input: 1 resource (lecture room) & n requests (e.g. events)

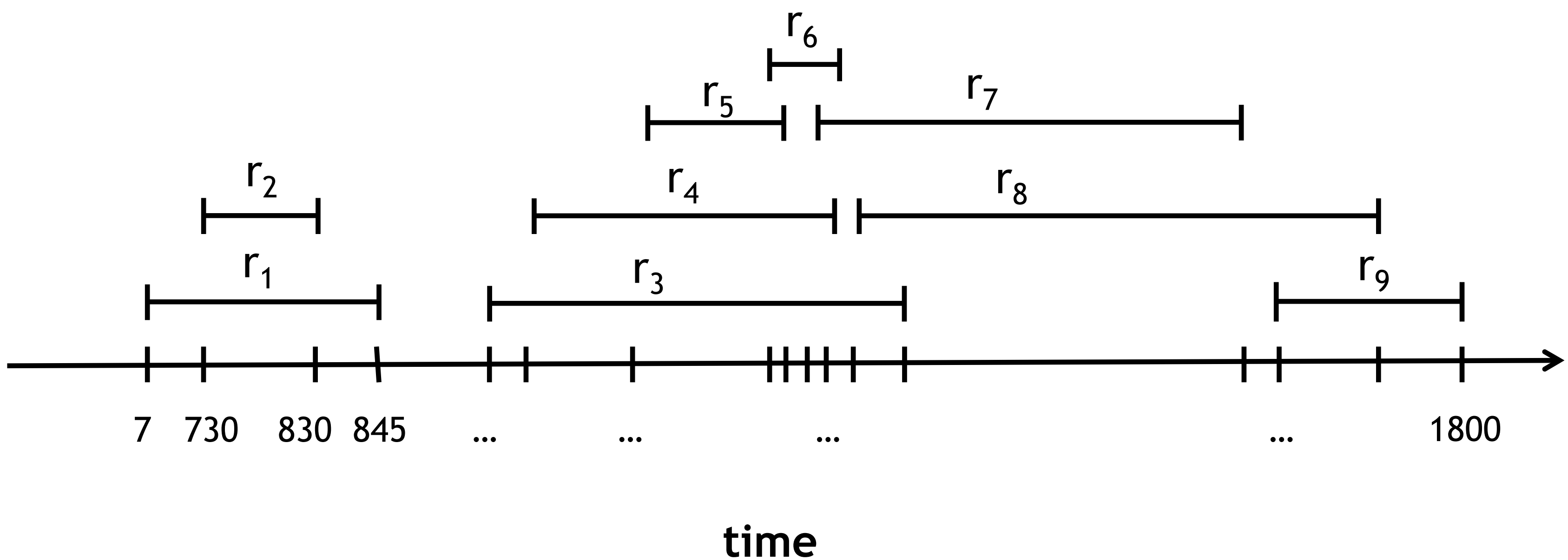
  where each request i has a start time s(i) and finish time f(i).



- Output: accept a maximum # requests that don't overlap each other
- I.e: Select a set S of requests s.t.

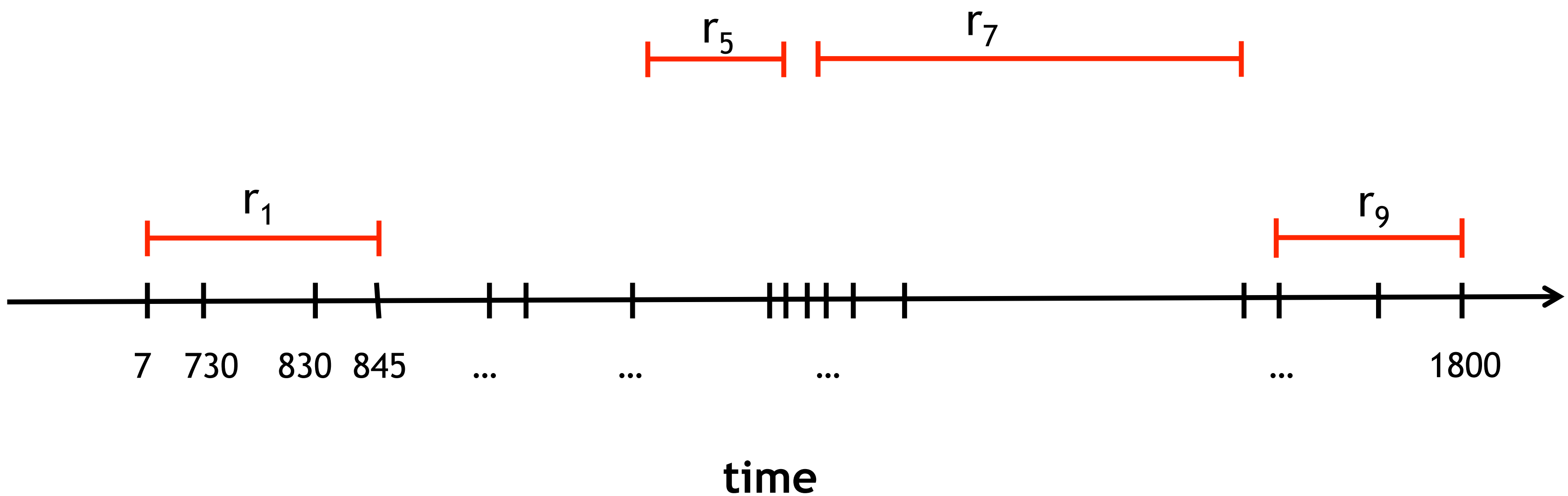
$$\forall (i, j) \text{ either } f(i) \le s(j) \text{ or } f(j) \le s(i)$$

Example (1)

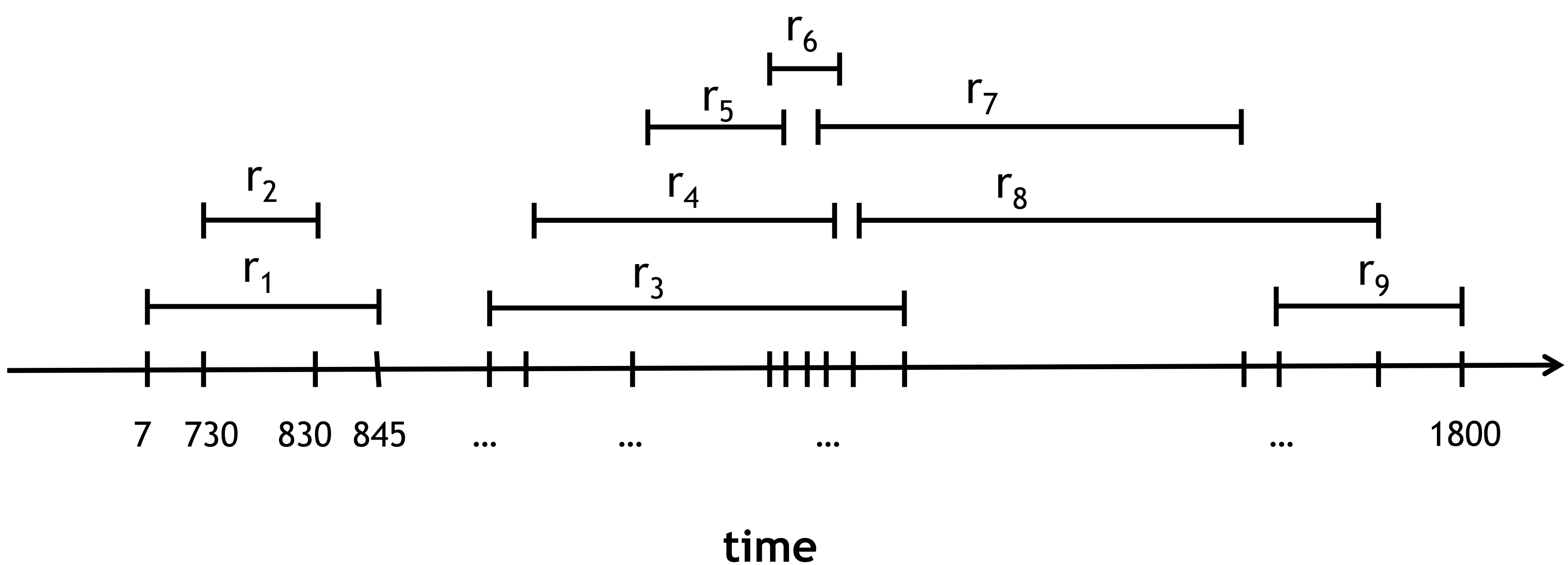


Example (1)

- $S_1=\{r_1, r_5, r_7, r_9\}$ selects 4 activities and is maximal

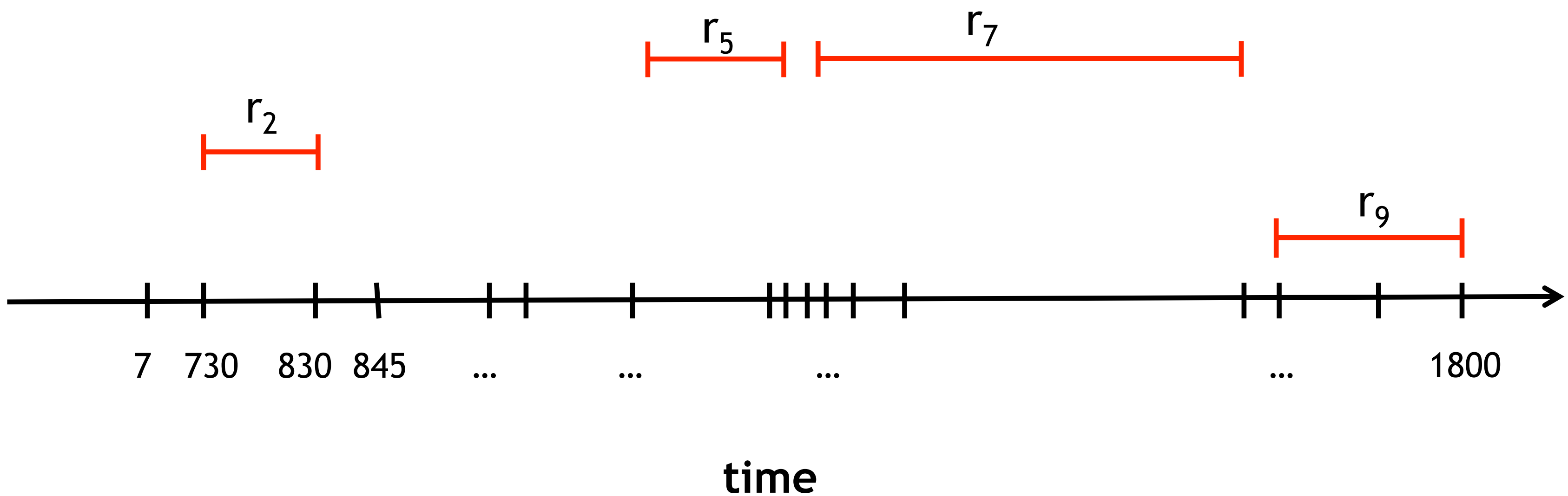


Example (2)

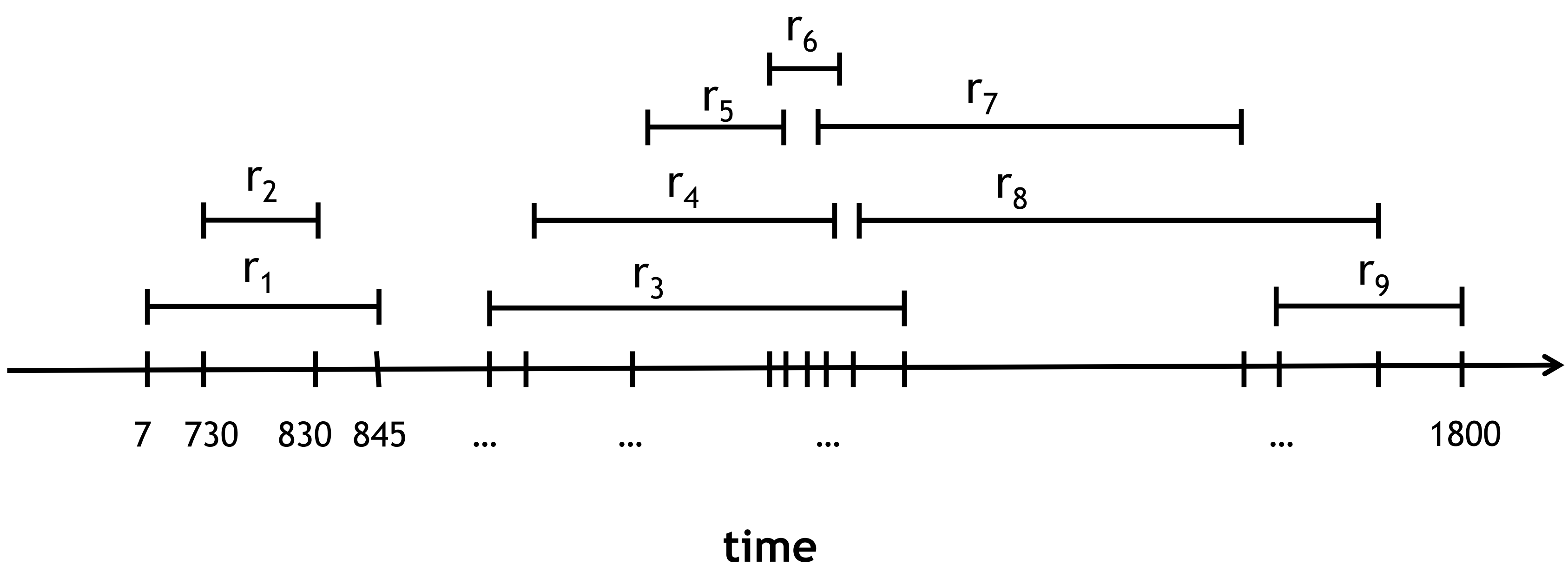


Example (2)

- S_2={r_2, r_5, r_7, r_9} also selects 4 activities and is maximal

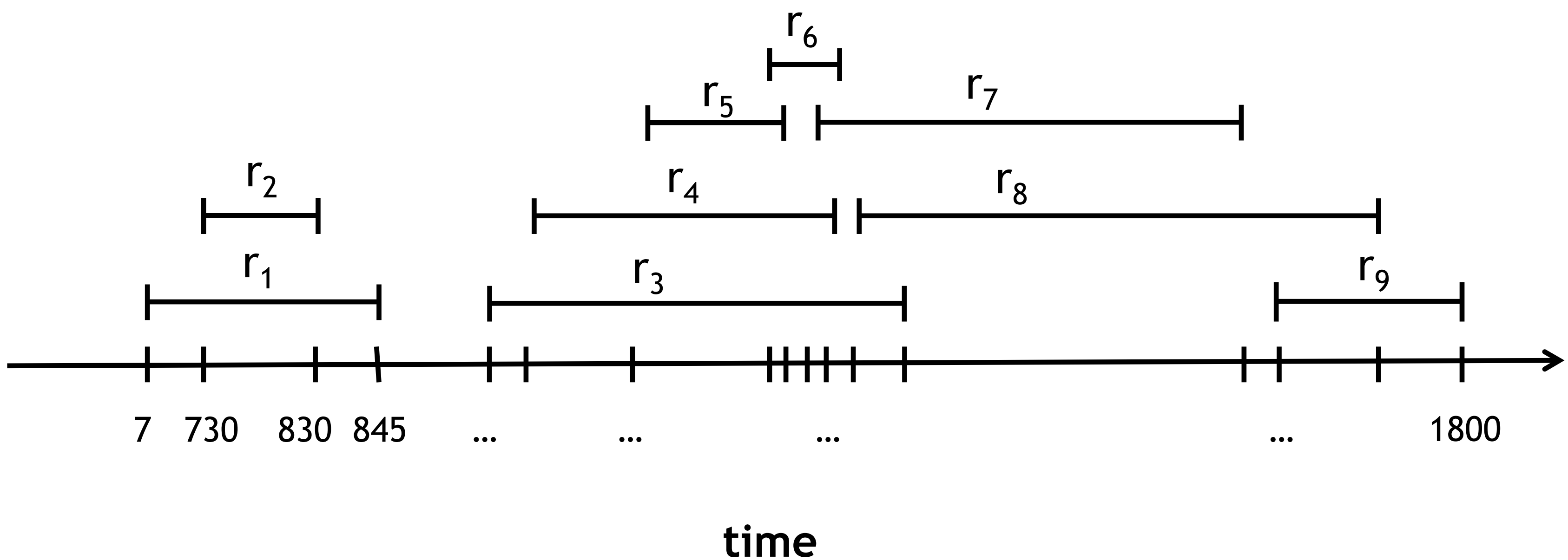


Check: Other Non-overlapping Sets Contain ≤ 3 Requests



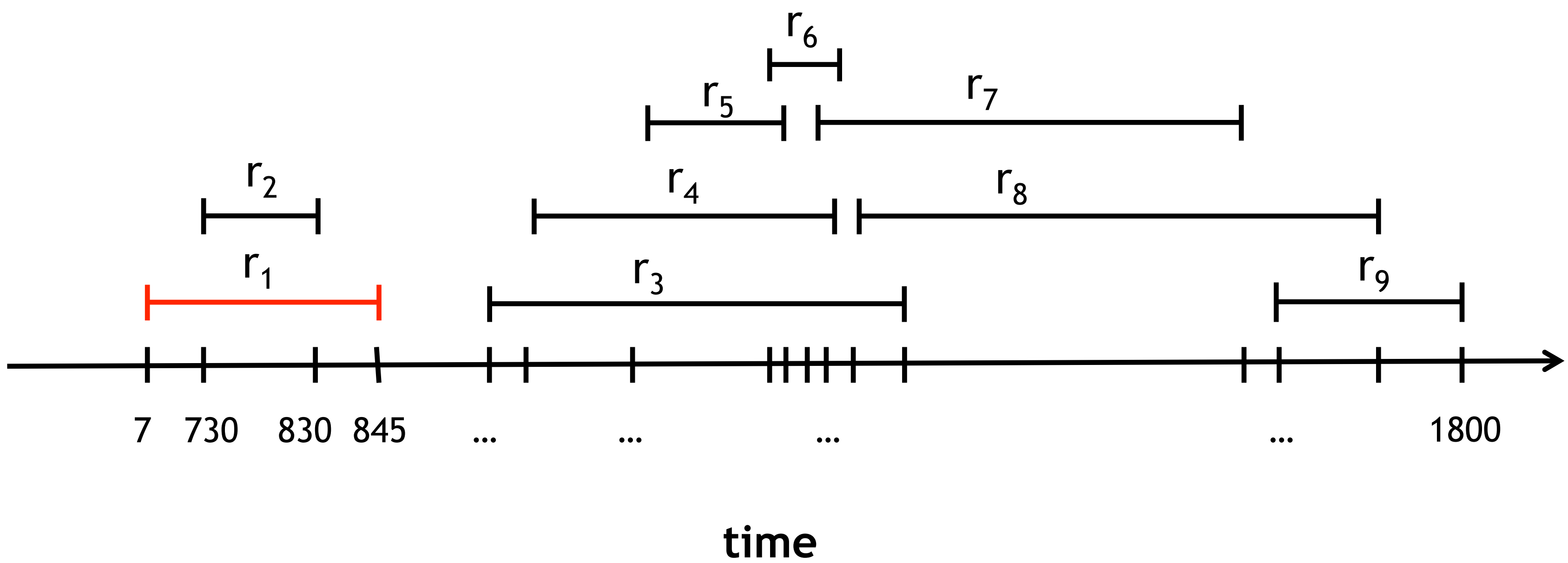
Possible Greedy Strategies (1)

- Earliest-Starting-Request: Pick the earliest starting request
- Remove any overlapping request
- Repeat until no requests left



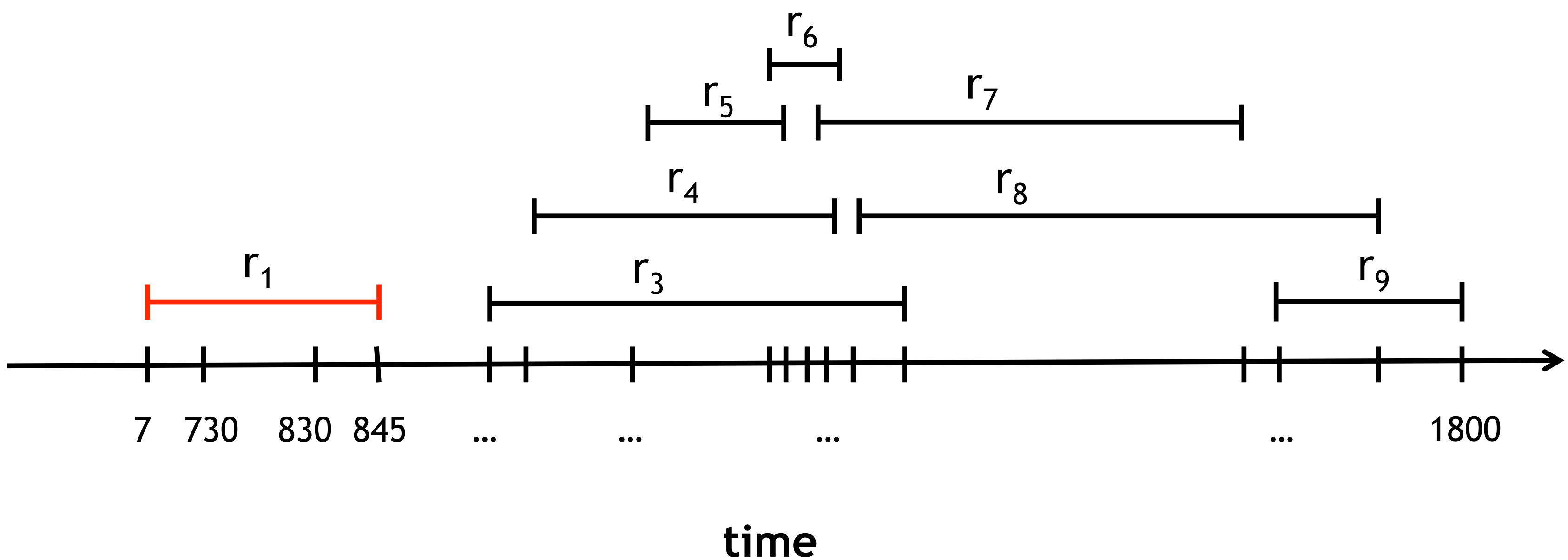
Possible Greedy Strategies (1)

- Earliest-Starting-Request: Pick the earliest starting request
- Remove any overlapping request
- Repeat until no requests left



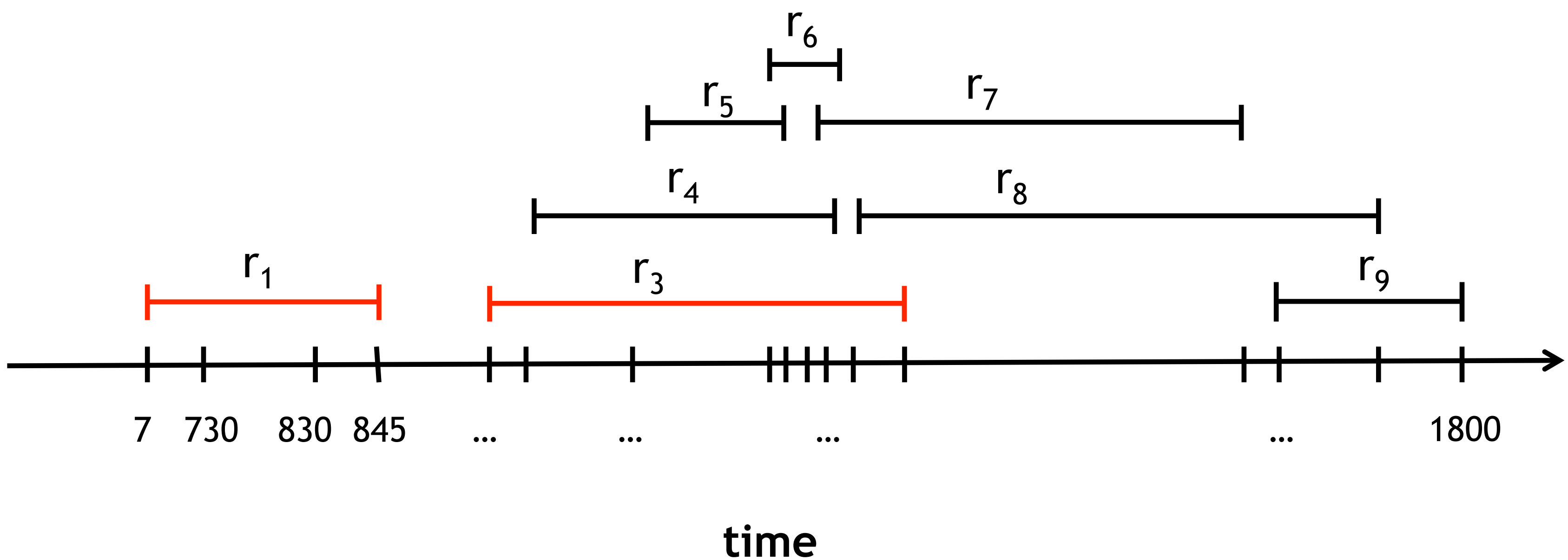
Possible Greedy Strategies (1)

- Earliest-Starting-Request: Pick the earliest starting request
- Remove any overlapping request
- Repeat until no requests left



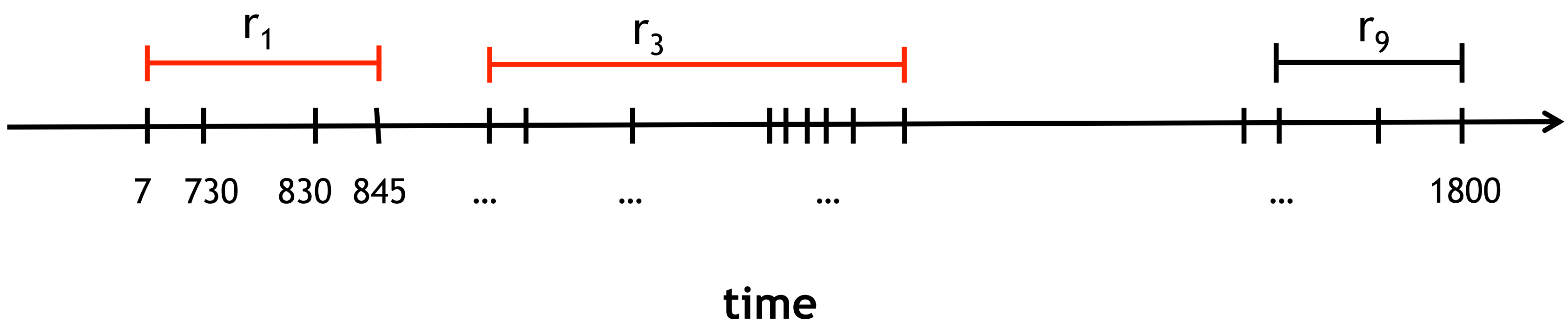
Possible Greedy Strategies (1)

- Earliest-Starting-Request: Pick the earliest starting request
- Remove any overlapping request
- Repeat until no requests left



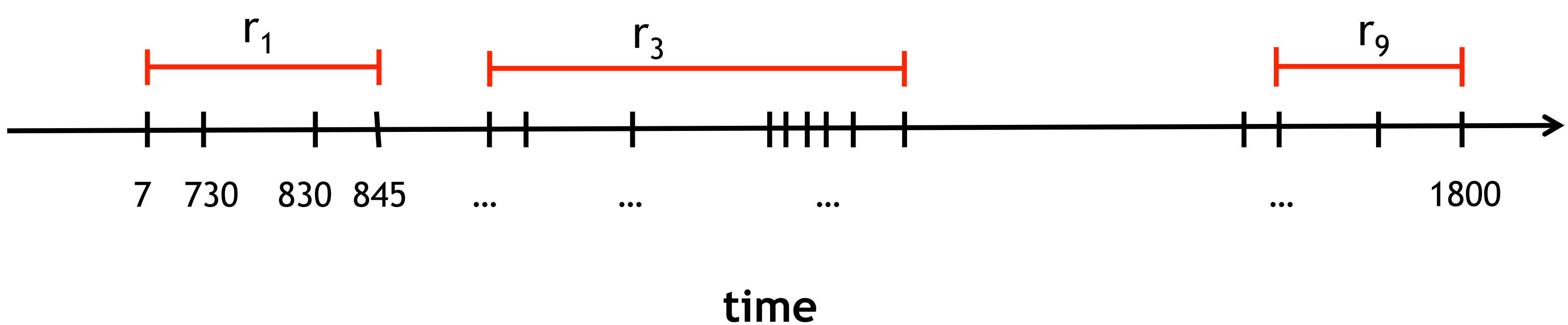
Possible Greedy Strategies (1)

- Earliest-Starting-Request: Pick the earliest starting request
- Remove any overlapping request
- Repeat until no requests left



Possible Greedy Strategies (1)

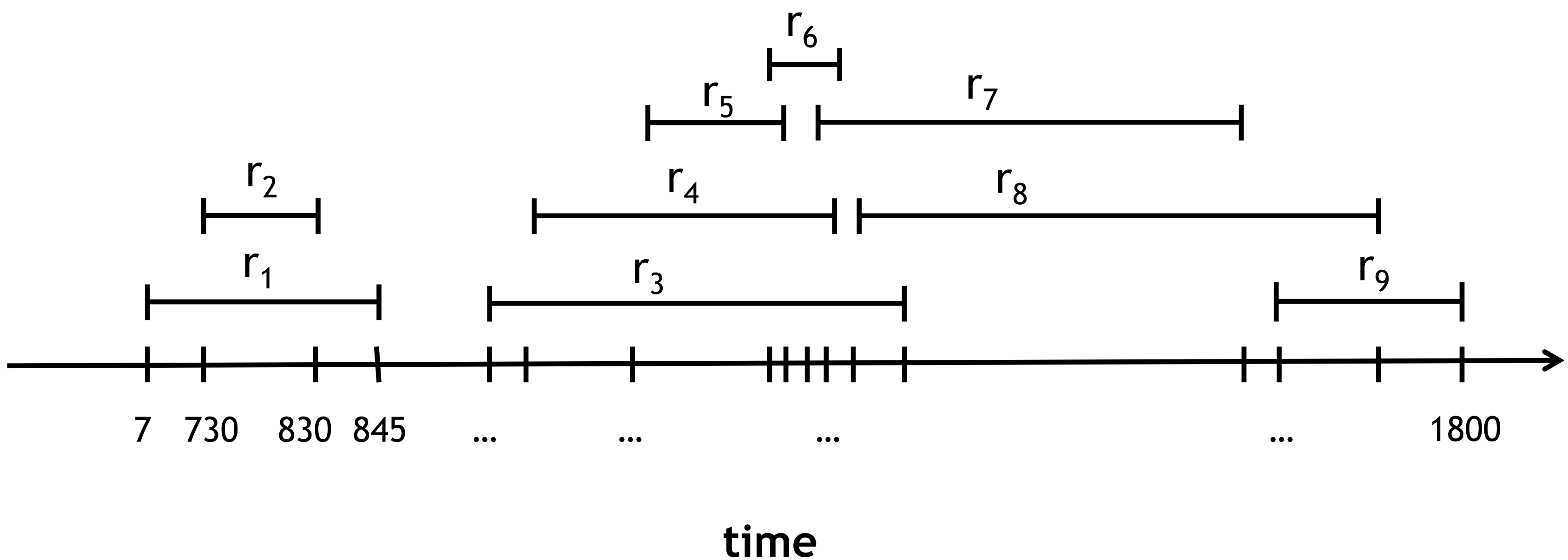
- Earliest-Starting-Request: Pick the earliest starting request
- Remove any overlapping request
- Repeat until no requests left



- Not optimal.
- Problem: earliest starting request can be very long. Worst-case: as long as the entire timeline.

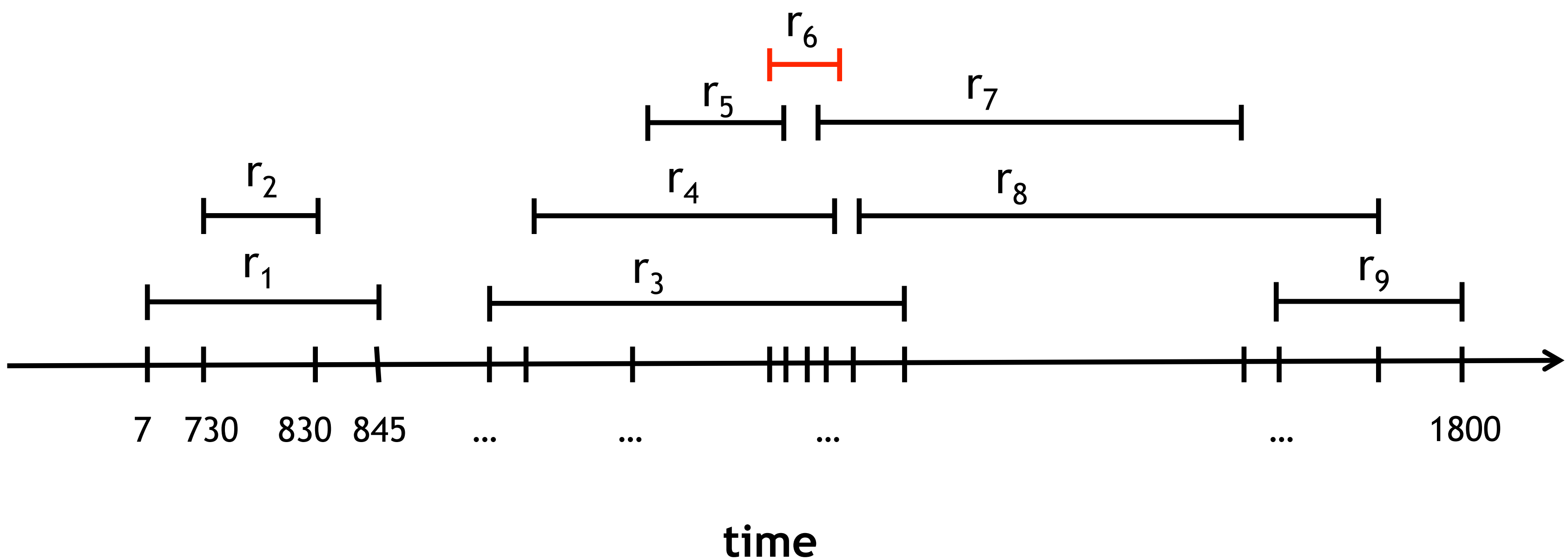
Possible Greedy Strategies (2)

- Shortest-Request: Pick the shortest request
- Remove any overlapping request
- Repeat until no requests left



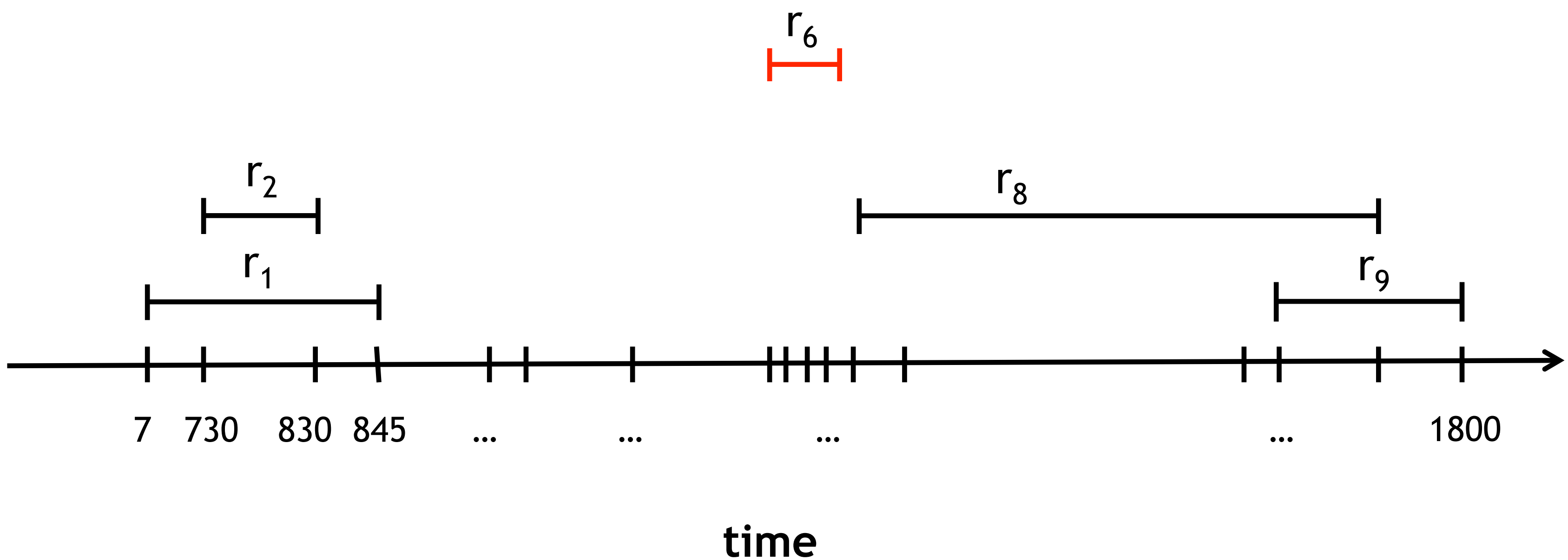
Possible Greedy Strategies (2)

- Shortest-Request: Pick the shortest request
- Remove any overlapping request
- Repeat until no requests left



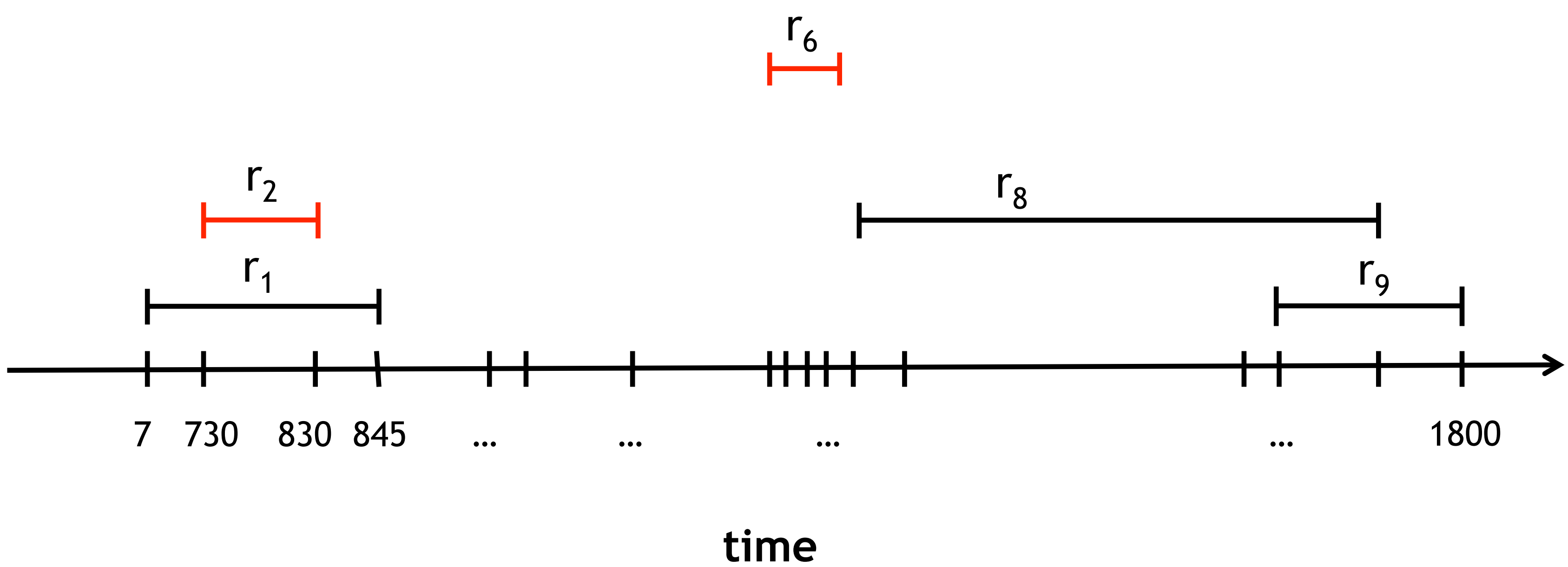
Possible Greedy Strategies (2)

- Shortest-Request: Pick the shortest request
- Remove any overlapping request
- Repeat until no requests left



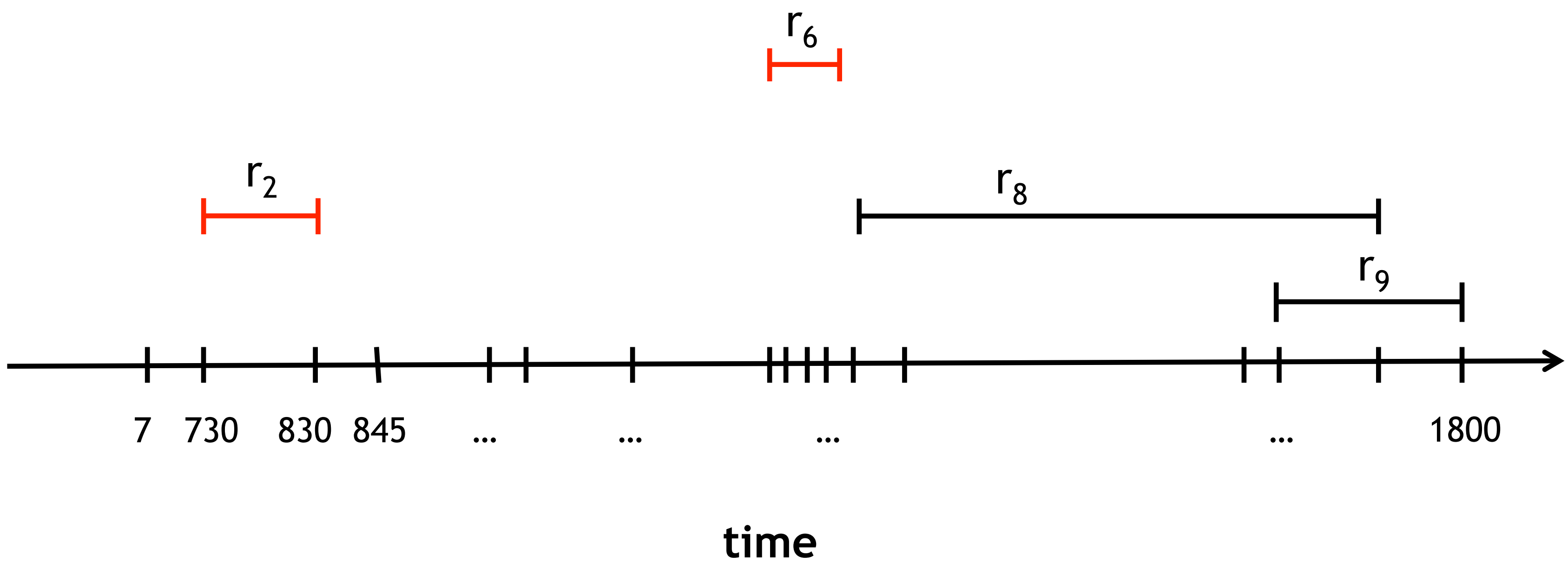
Possible Greedy Strategies (2)

- Shortest-Request: Pick the shortest request
- Remove any overlapping request
- Repeat until no requests left



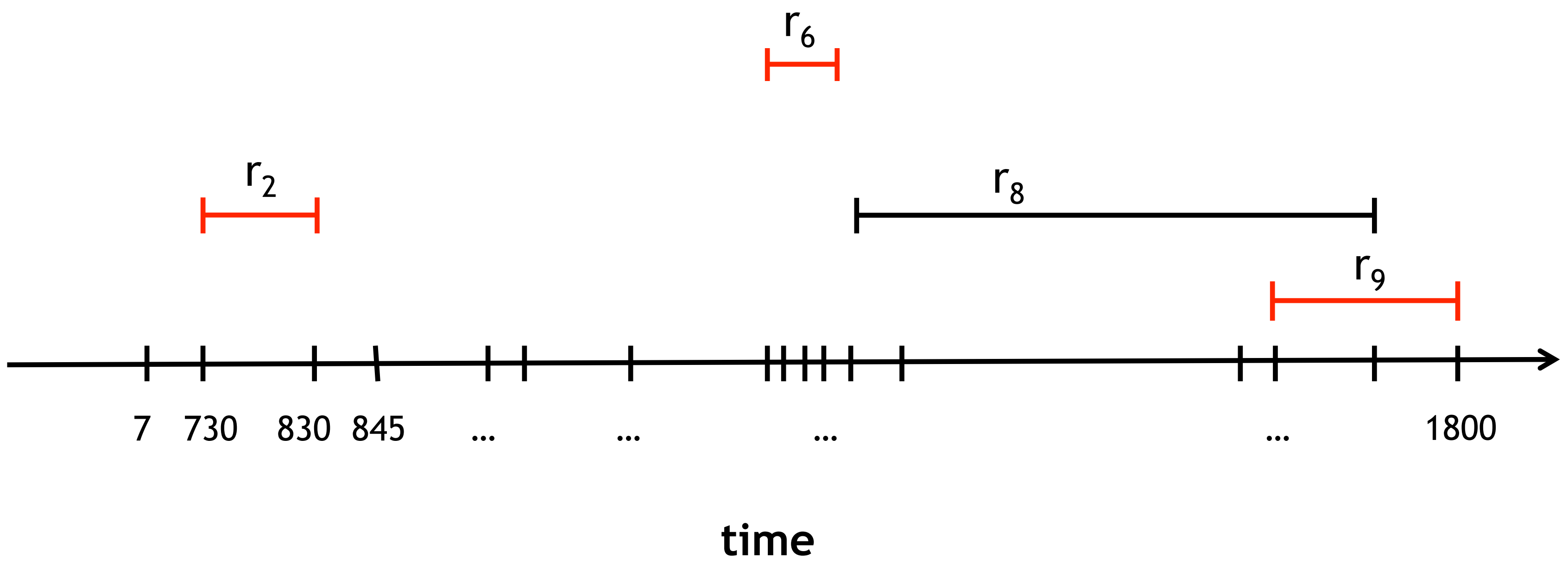
Possible Greedy Strategies (2)

- Shortest-Request: Pick the shortest request
- Remove any overlapping request
- Repeat until no requests left



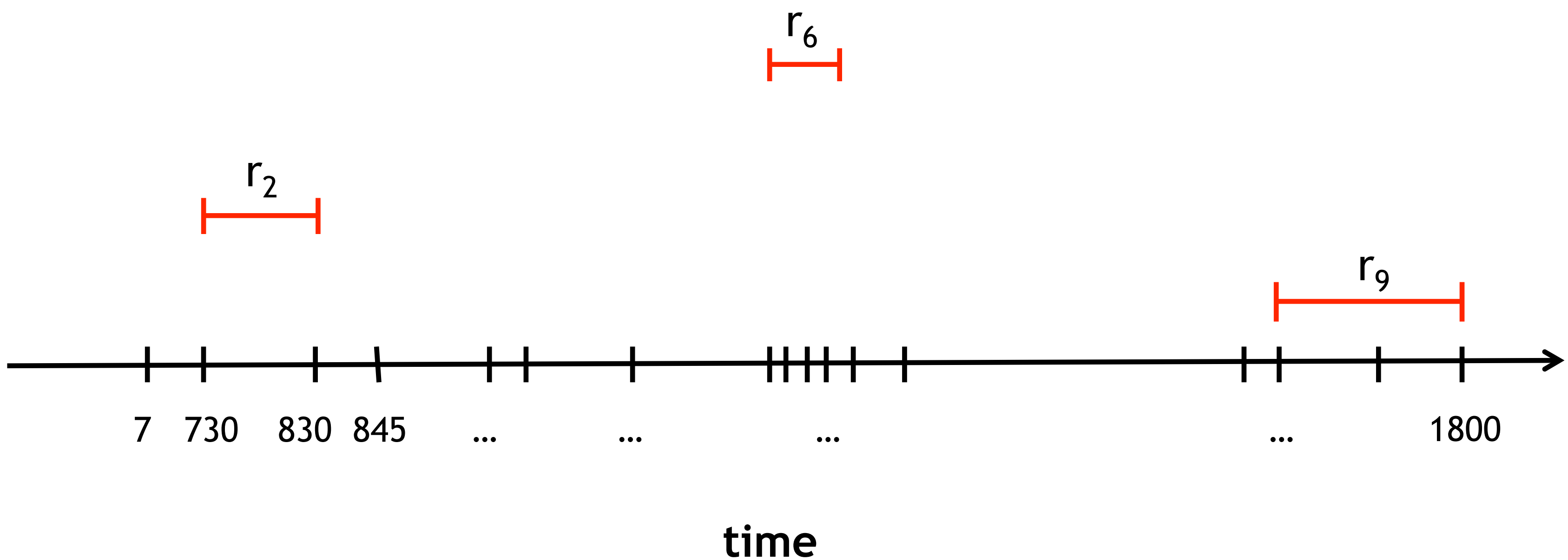
Possible Greedy Strategies (2)

- Shortest-Request: Pick the shortest request
- Remove any overlapping request
- Repeat until no requests left



Possible Greedy Strategies (2)

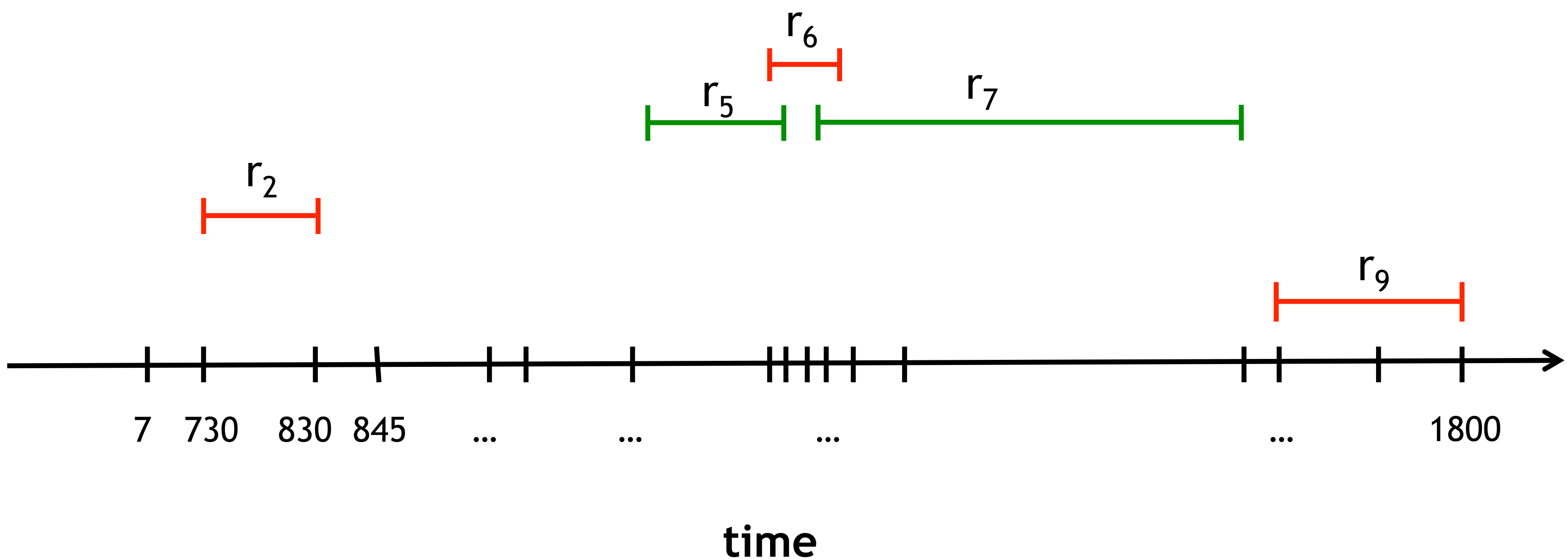
- Shortest-Request: Pick the shortest request
- Remove any overlapping request
- Repeat until no requests left



- Not optimal.
- Problem: can intersect two non-overlapping jobs that could have been accepted

Possible Greedy Strategies (2)

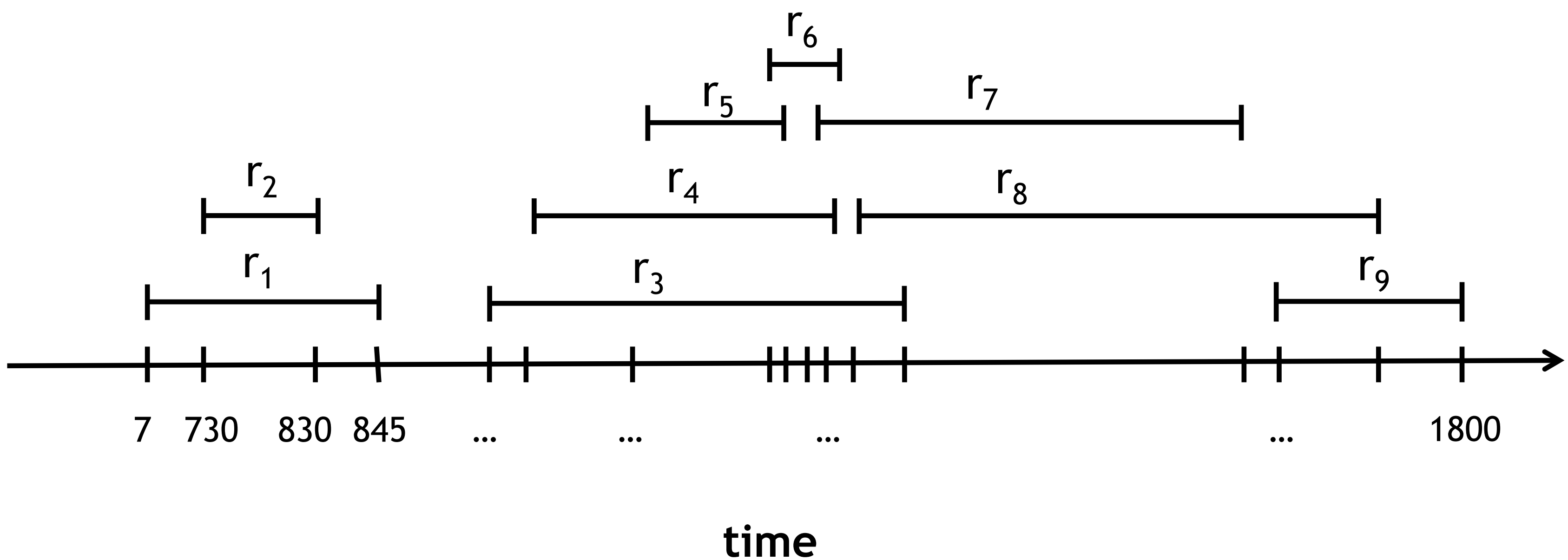
- Shortest-Request: Pick the shortest request
- Remove any overlapping request
- Repeat until no requests left



- Not optimal.
- Problem: can intersect two non-overlapping jobs that could have been accepted

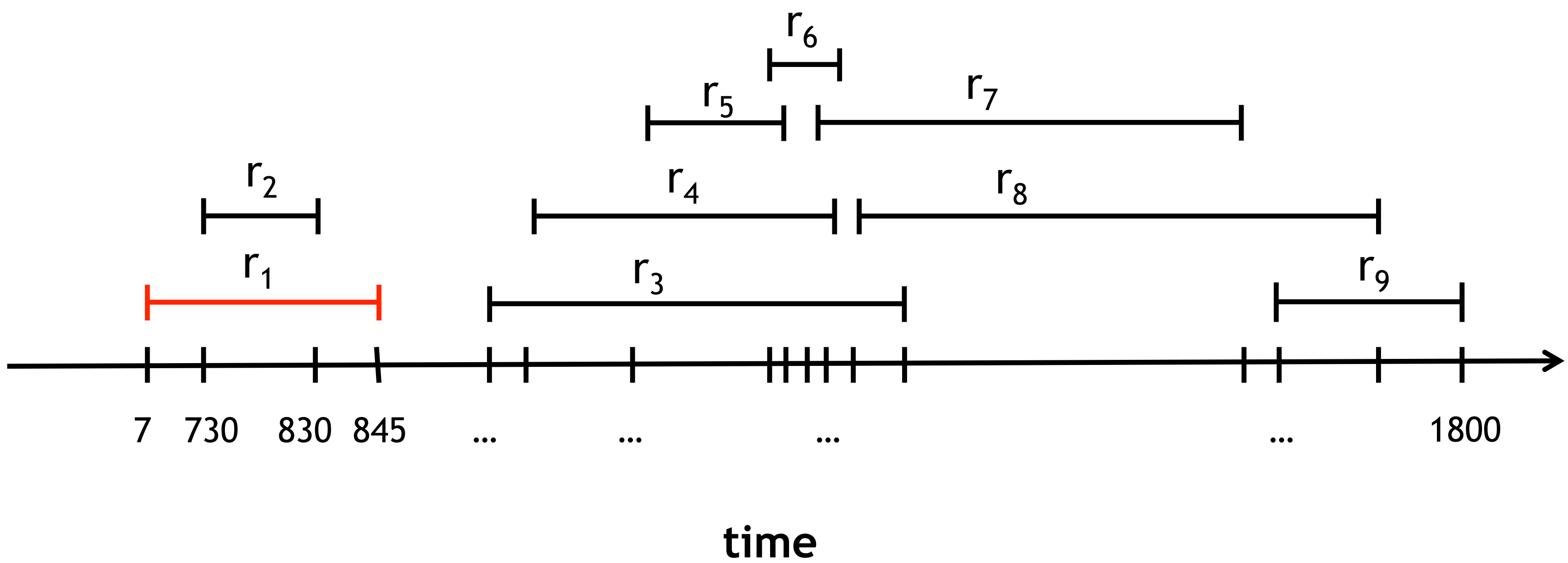
Possible Greedy Strategies (3)

- Pick Request-That-Overlaps-With-Min-Other-Remaining-Requests
- Remove any overlapping request
- Repeat until no requests left



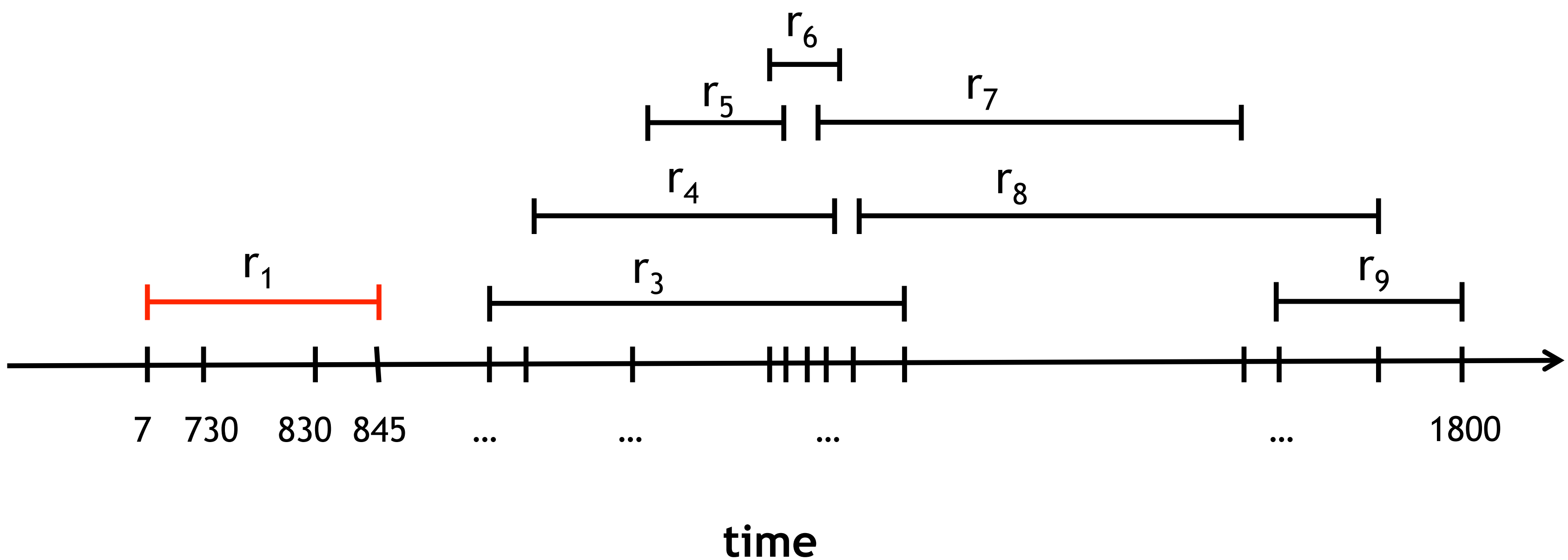
Possible Greedy Strategies (3)

- Pick Request-That-Overlaps-With-Min-Other-Remaining-Requests
- Remove any overlapping request
- Repeat until no requests left



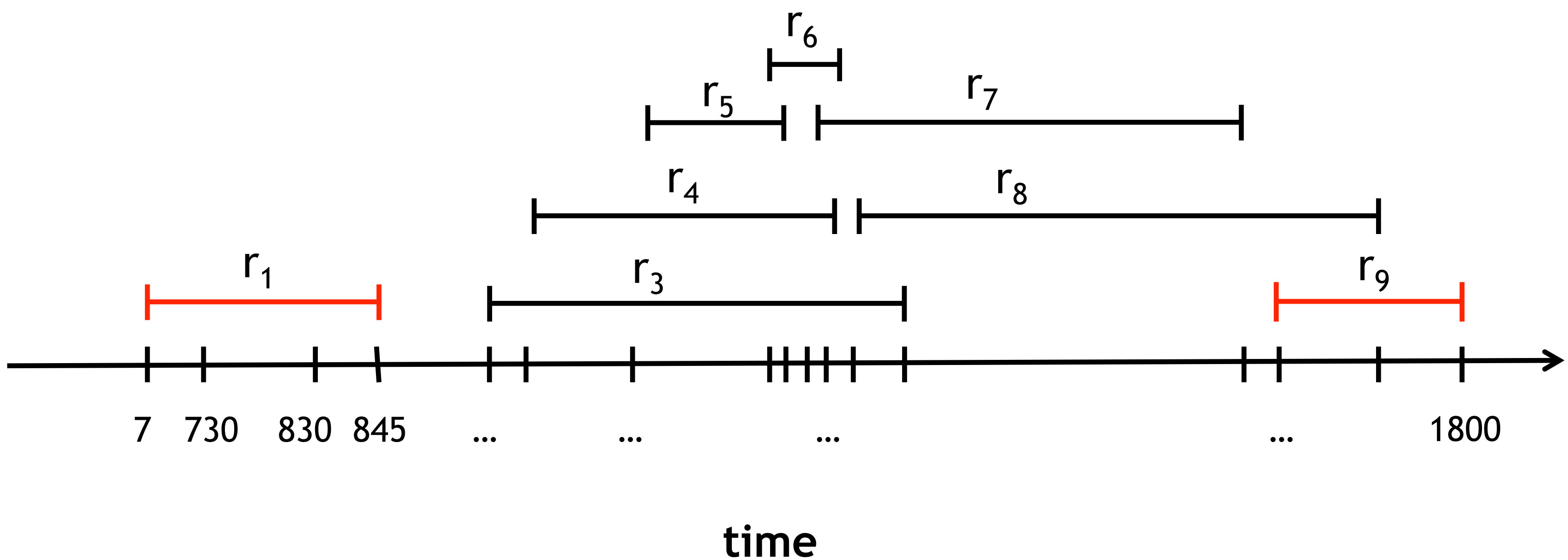
Possible Greedy Strategies (3)

- Pick Request-That-Overlaps-With-Min-Other-Remaining-Requests
- Remove any overlapping request
- Repeat until no requests left



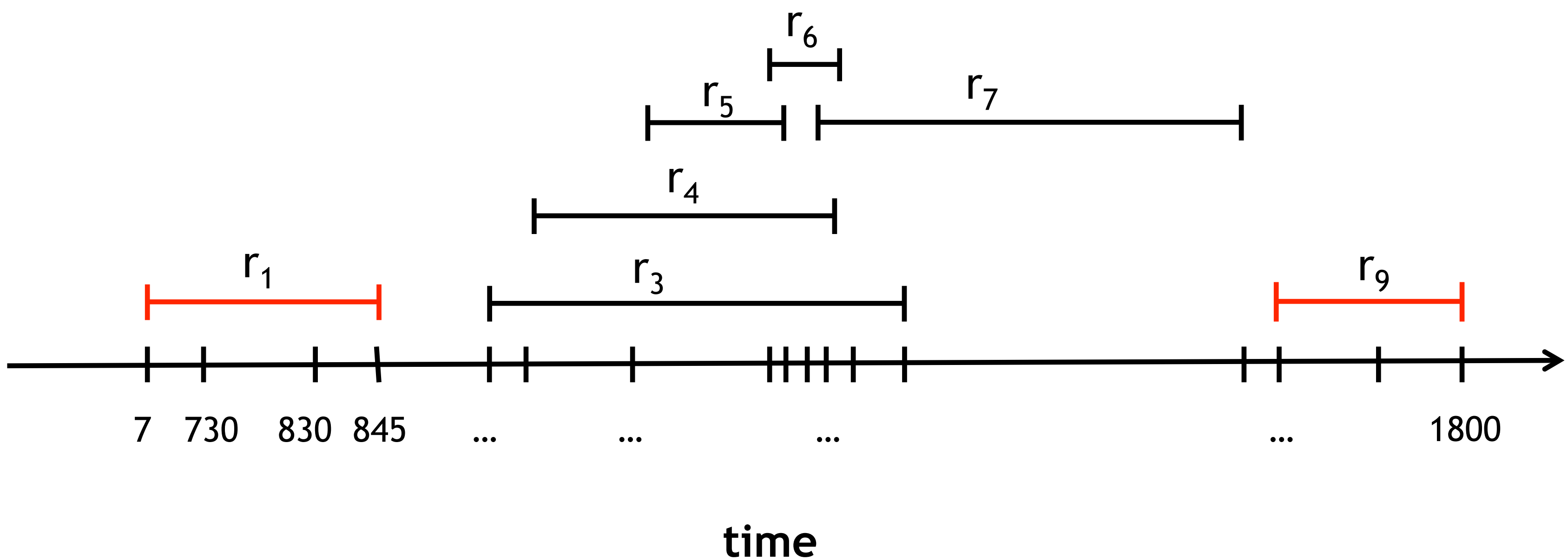
Possible Greedy Strategies (3)

- Pick Request-That-Overlaps-With-Min-Other-Remaining-Requests
- Remove any overlapping request
- Repeat until no requests left



Possible Greedy Strategies (3)

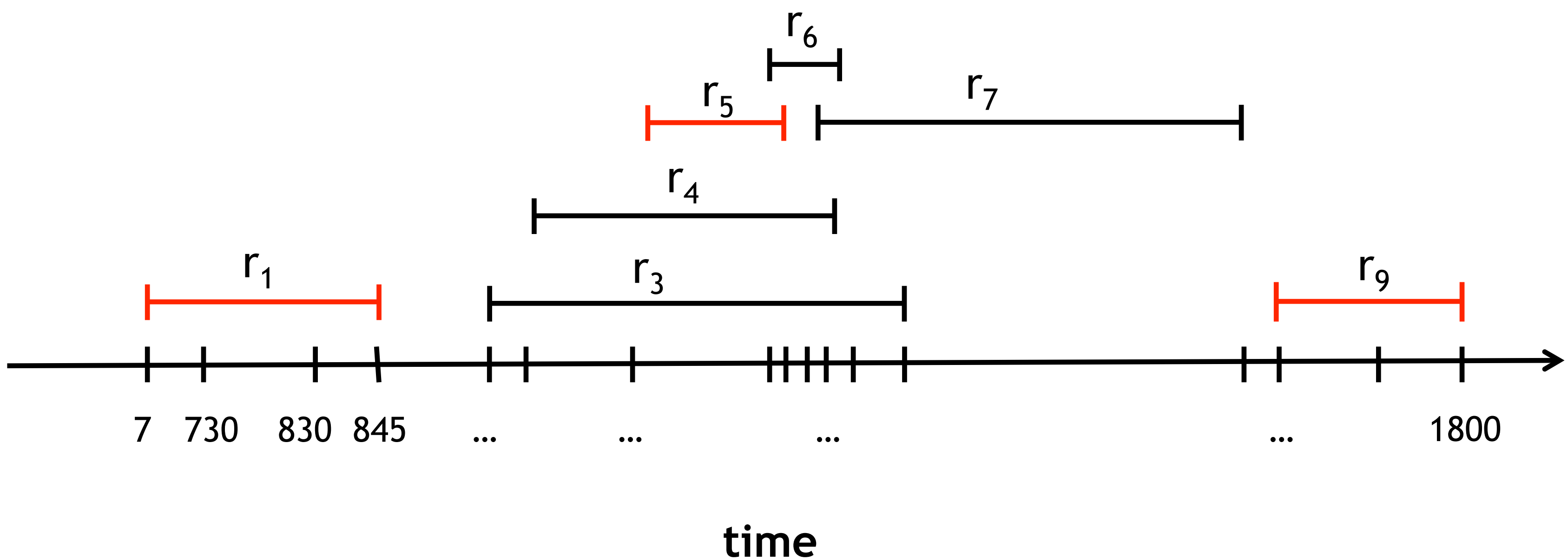
- Pick Request-That-Overlaps-With-Min-Other-Remaining-Requests
- Remove any overlapping request
- Repeat until no requests left



- Now: r_5, and r_7 overlap with 3 other requests
- r_3, r_4, and r_6 overlap with 4 other requests

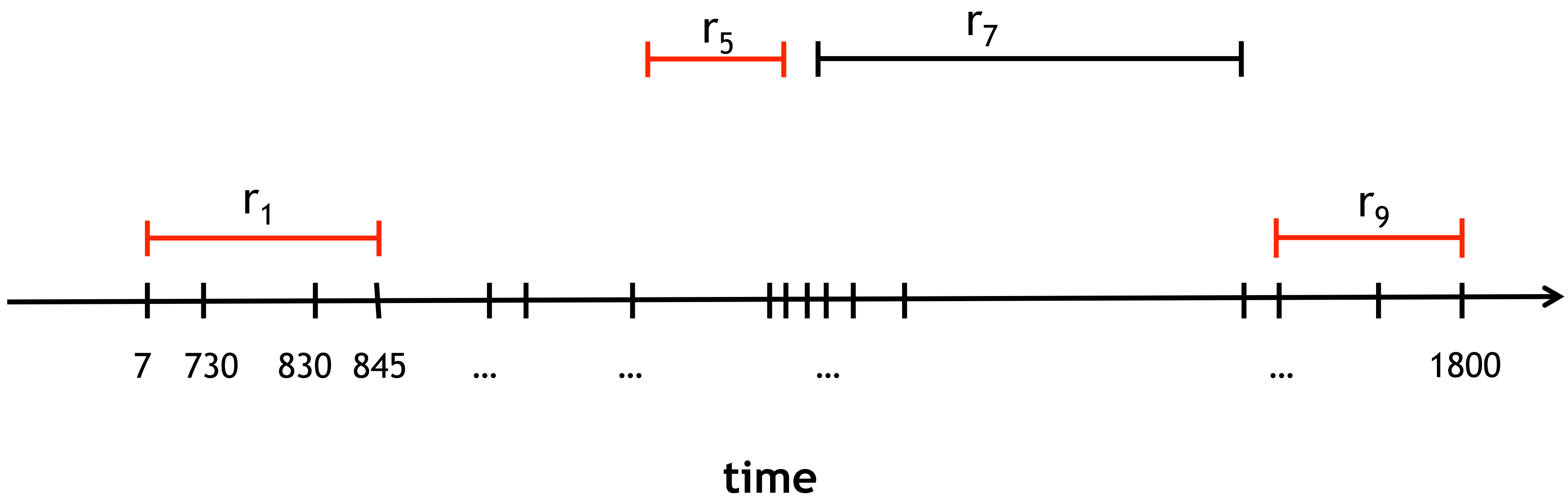
Possible Greedy Strategies (3)

- Pick Request-That-Overlaps-With-Min-Other-Remaining-Requests
- Remove any overlapping request
- Repeat until no requests left



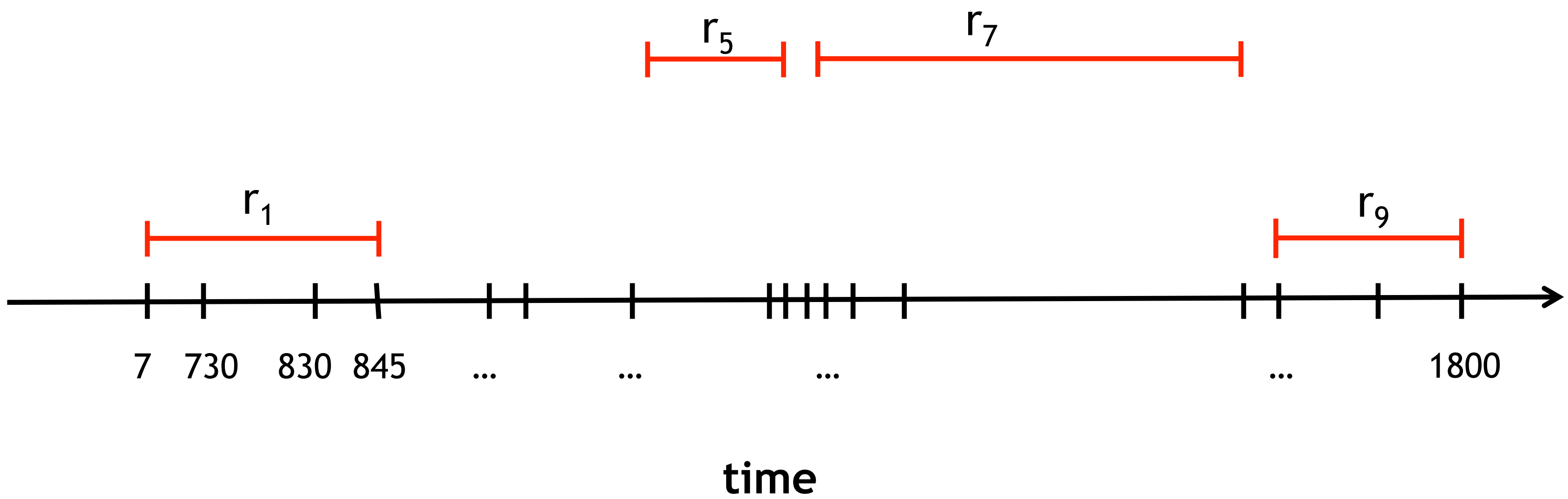
Possible Greedy Strategies (3)

- Pick Request-That-Overlaps-With-Min-Other-Remaining-Requests
- Remove any overlapping request
- Repeat until no requests left



Possible Greedy Strategies (3)

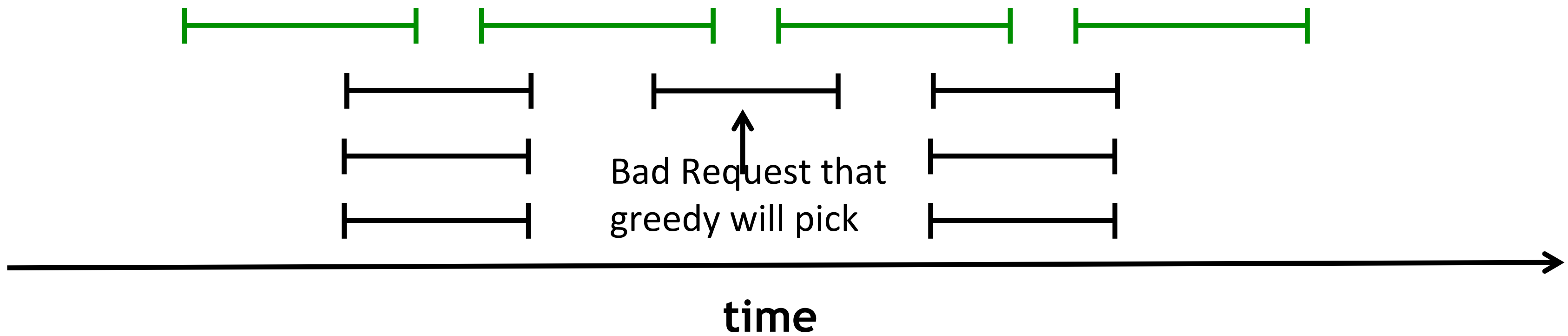
- Pick Request-That-Overlaps-With-Min-Other-Remaining-Requests
- Remove any overlapping request
- Repeat until no requests left



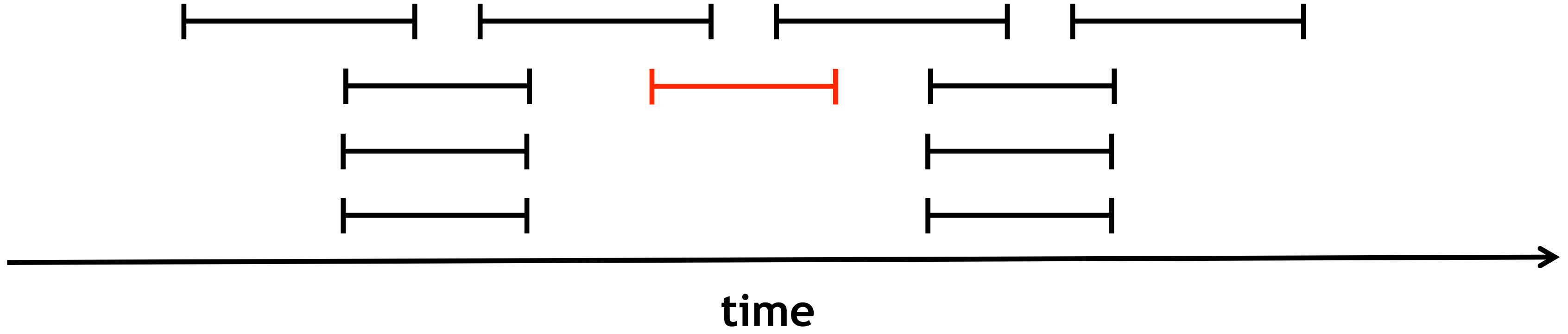
- Seems to work.
- But it actually won't always return the optimum one.

Counter Example For Strategy 3

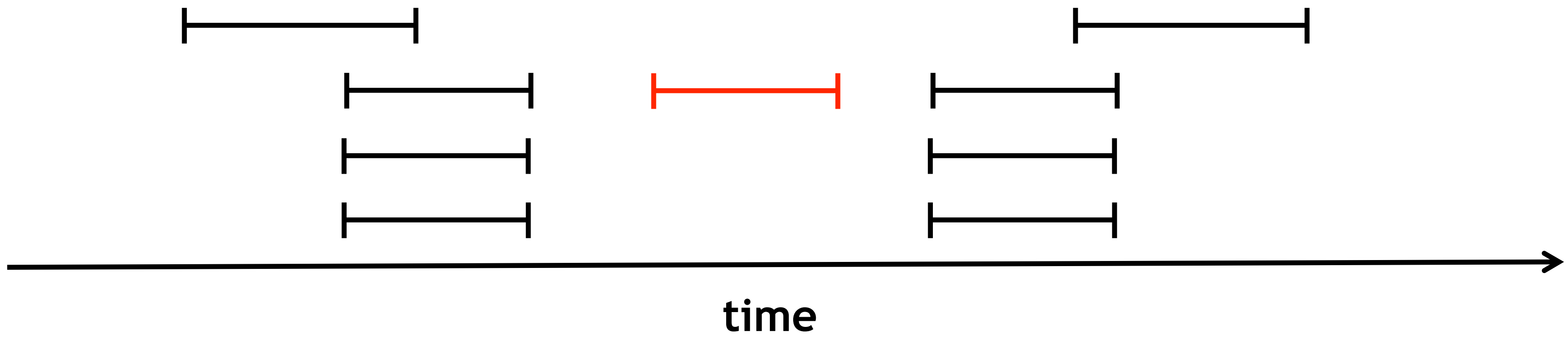
- Method for designing a counter example:
- Plant an optimal solution
- Plant a bad request that strategy 3 will pick in the first selection.
- Now: Make sure greedy picks the bad request
  - Bad request intersects with 2 other requests
  - Make sure every other request intersects with 3 or more.



Counter Example For Strategy 3

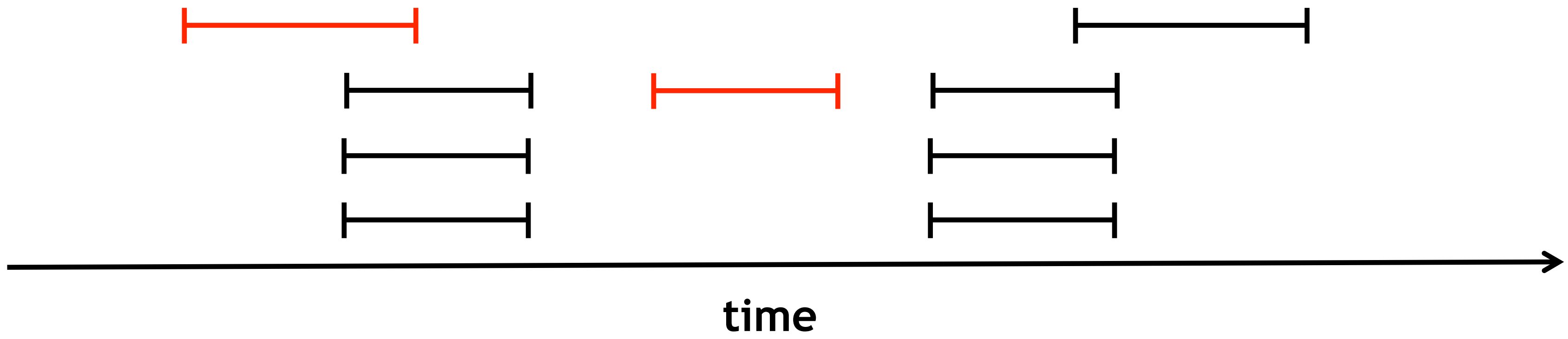


Counter Example For Strategy 3

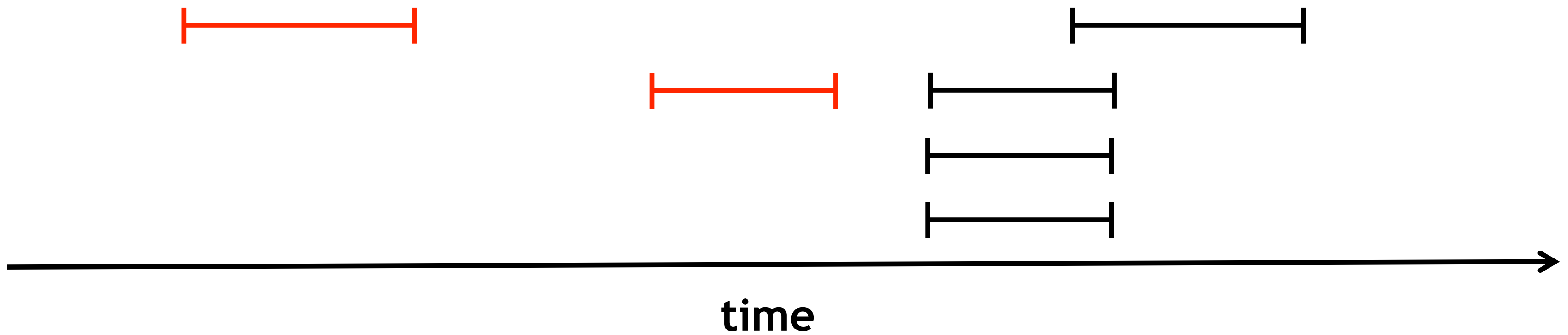


- Now every remaining request intersects 3 other, so can pick any

Counter Example For Strategy 3

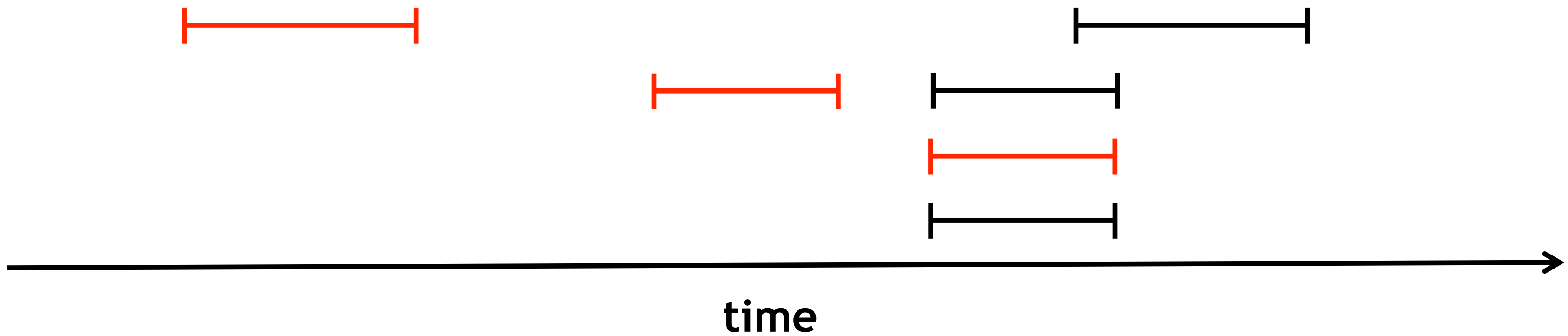


Counter Example For Strategy 3



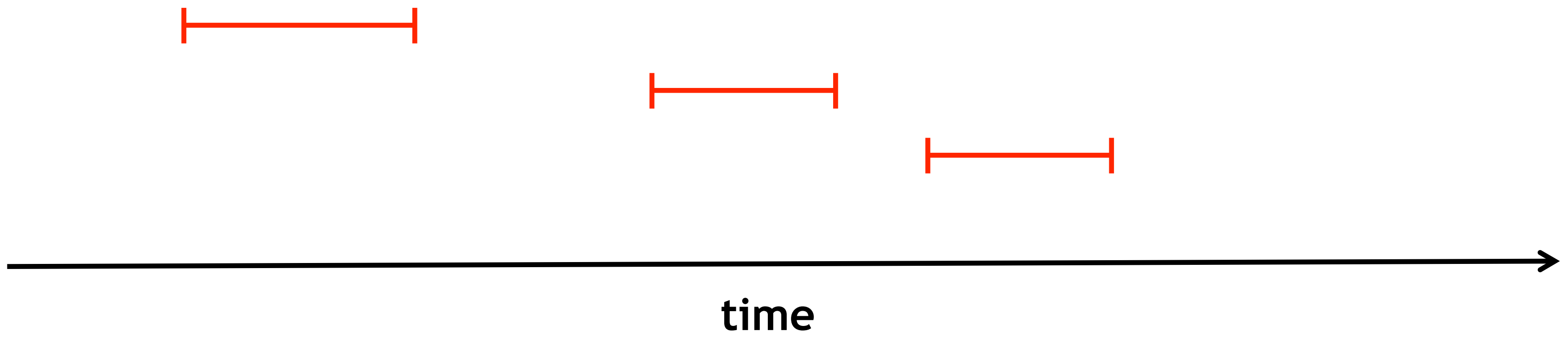
- Similarly: every remaining request intersects 3 other

Counter Example For Strategy 3



◆ Similarly: every remaining request intersects 3 other

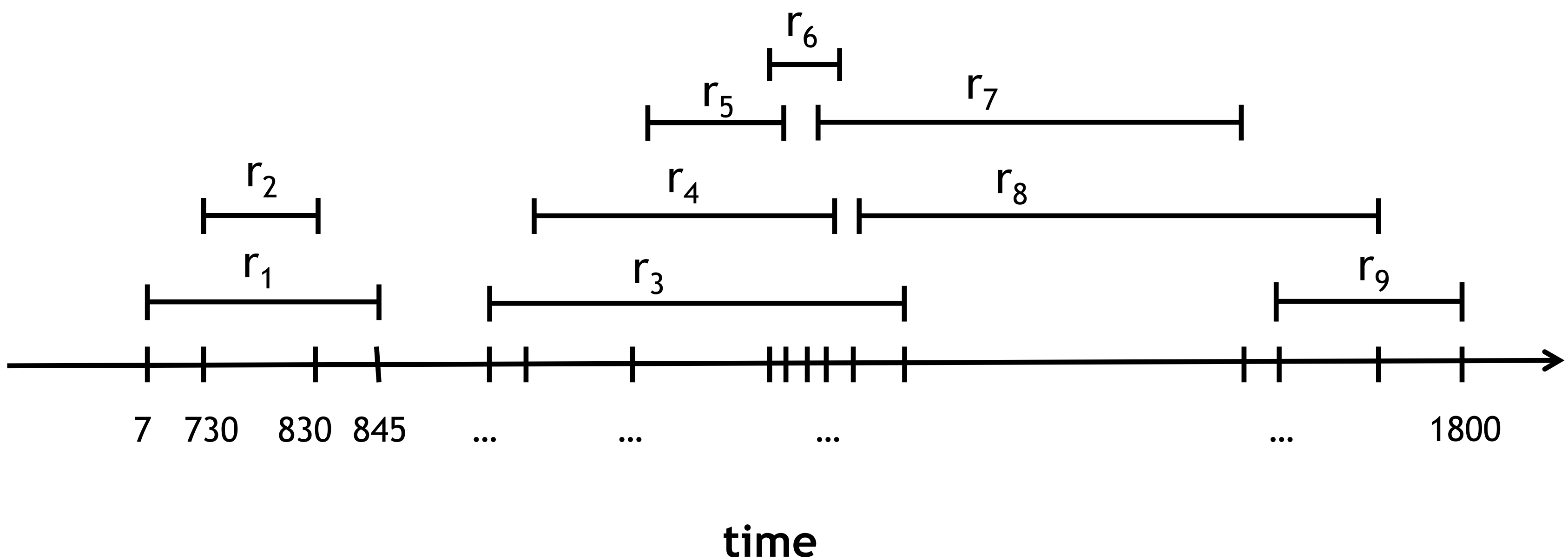
Counter Example For Strategy 3



- Not optimal

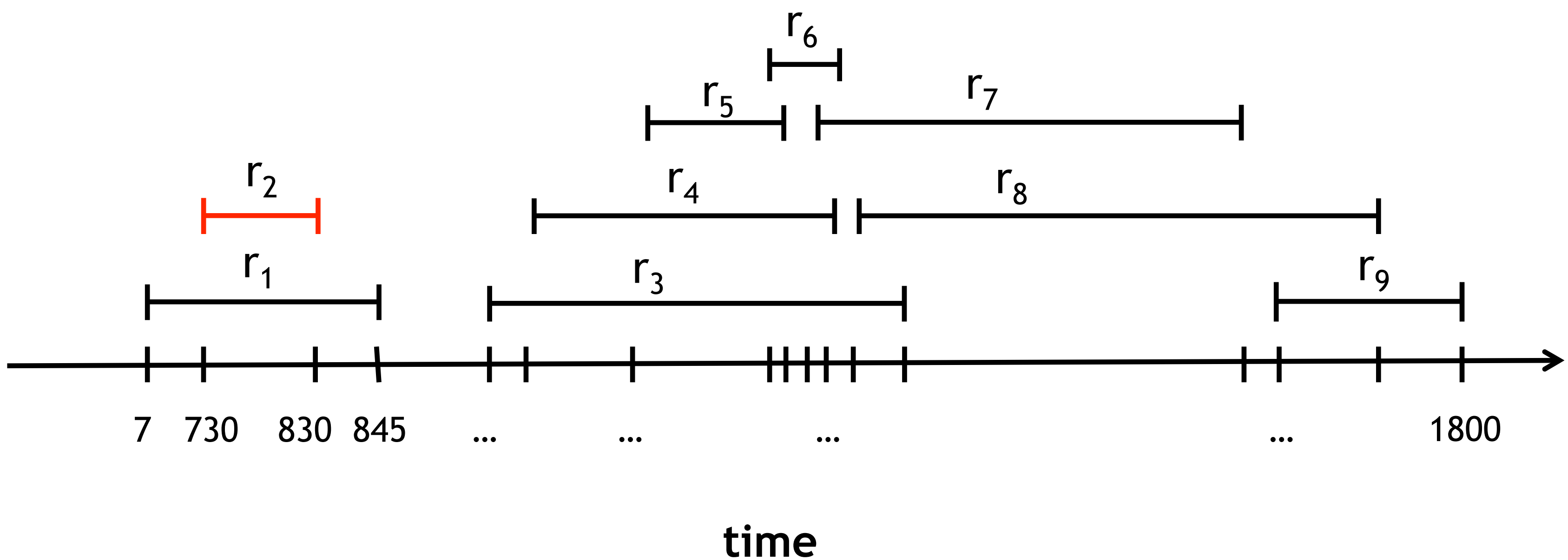
Possible Greedy Strategies (4)

- Earliest-Finishing-Request: Pick the earliest finishing request
- Remove any overlapping request
- Repeat until no requests left



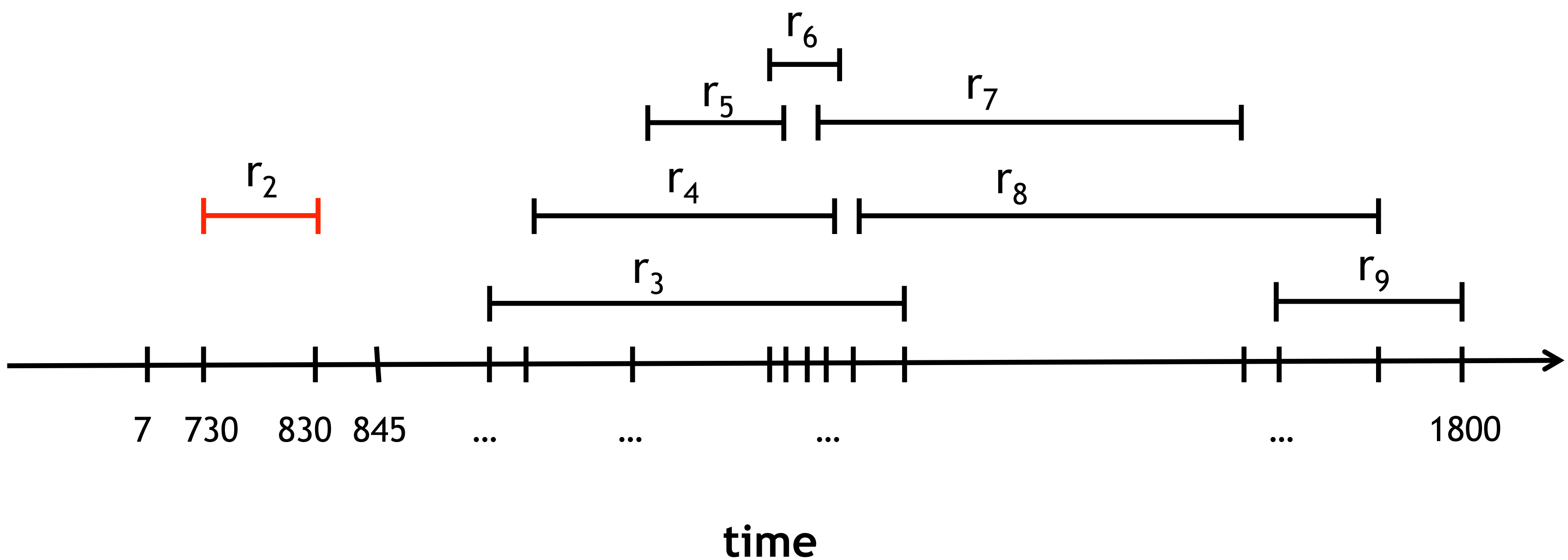
Possible Greedy Strategies (4)

- Earliest-Finishing-Request: Pick the earliest finishing request
- Remove any overlapping request
- Repeat until no requests left



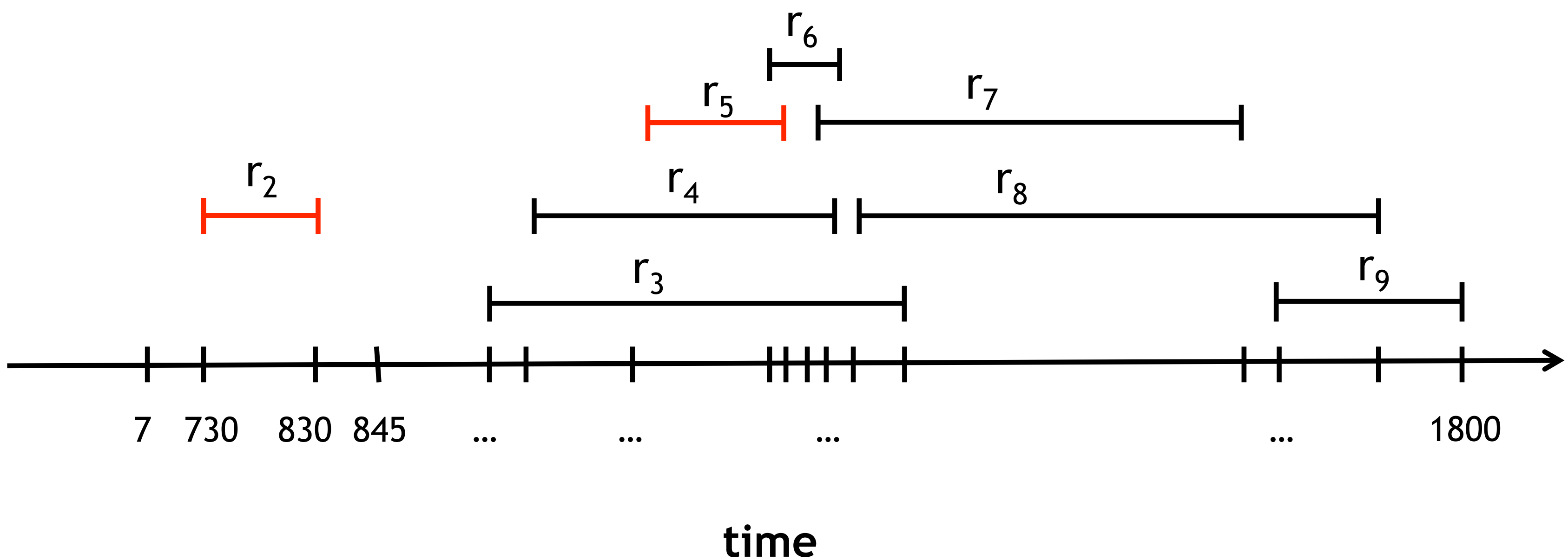
Possible Greedy Strategies (4)

- Earliest-Finishing-Request: Pick the earliest finishing request
- Remove any overlapping request
- Repeat until no requests left



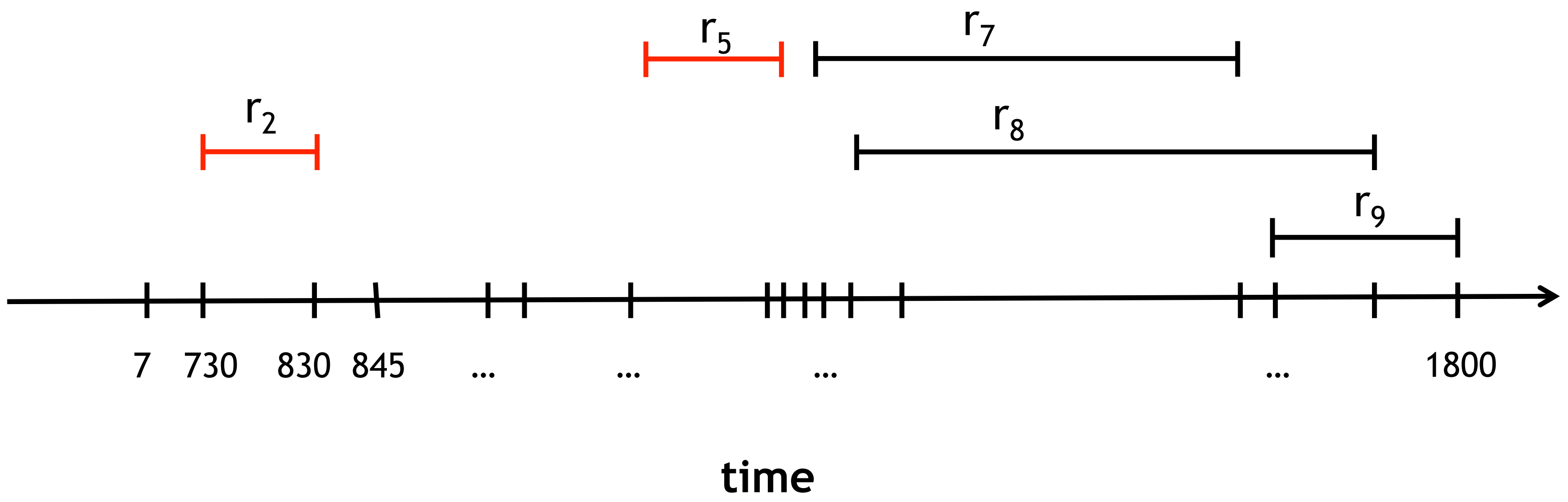
Possible Greedy Strategies (4)

- Earliest-Finishing-Request: Pick the earliest finishing request
- Remove any overlapping request
- Repeat until no requests left



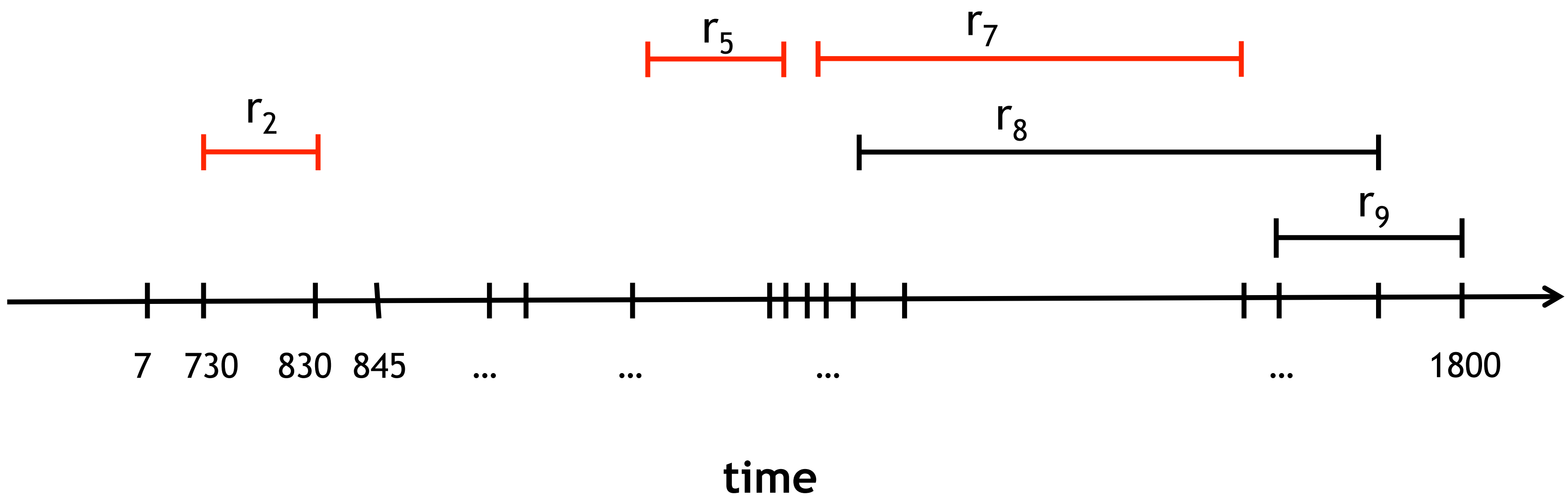
Possible Greedy Strategies (4)

- Earliest-Finishing-Request: Pick the earliest finishing request
- Remove any overlapping request
- Repeat until no requests left



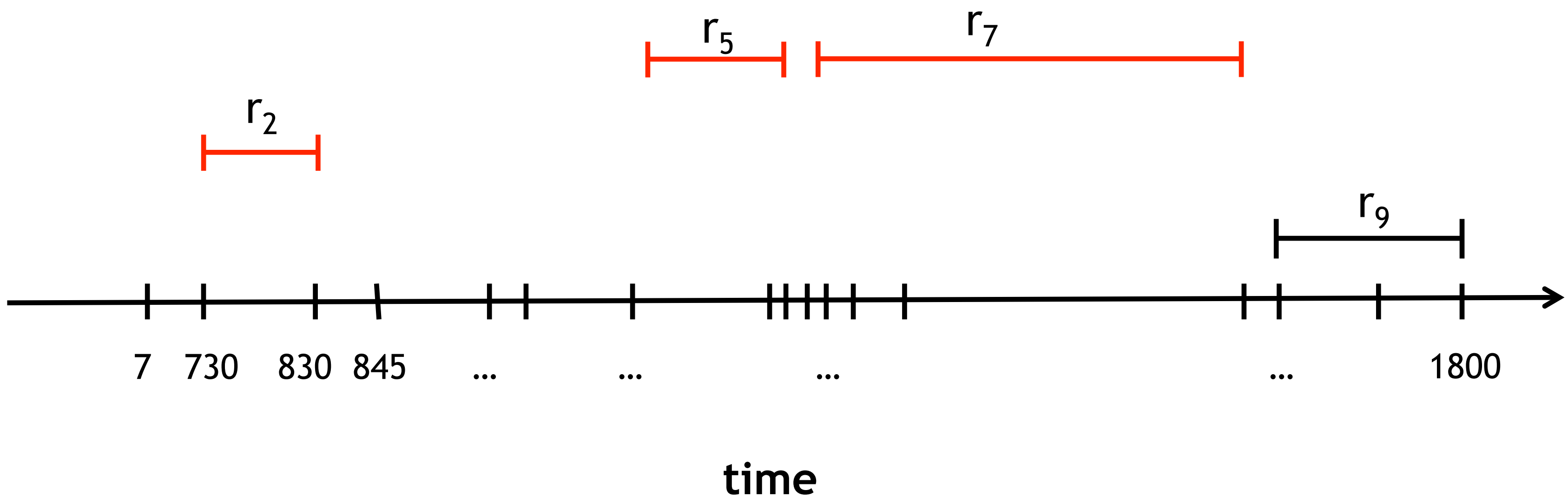
Possible Greedy Strategies (4)

- Earliest-Finishing-Request: Pick the earliest finishing request
- Remove any overlapping request
- Repeat until no requests left



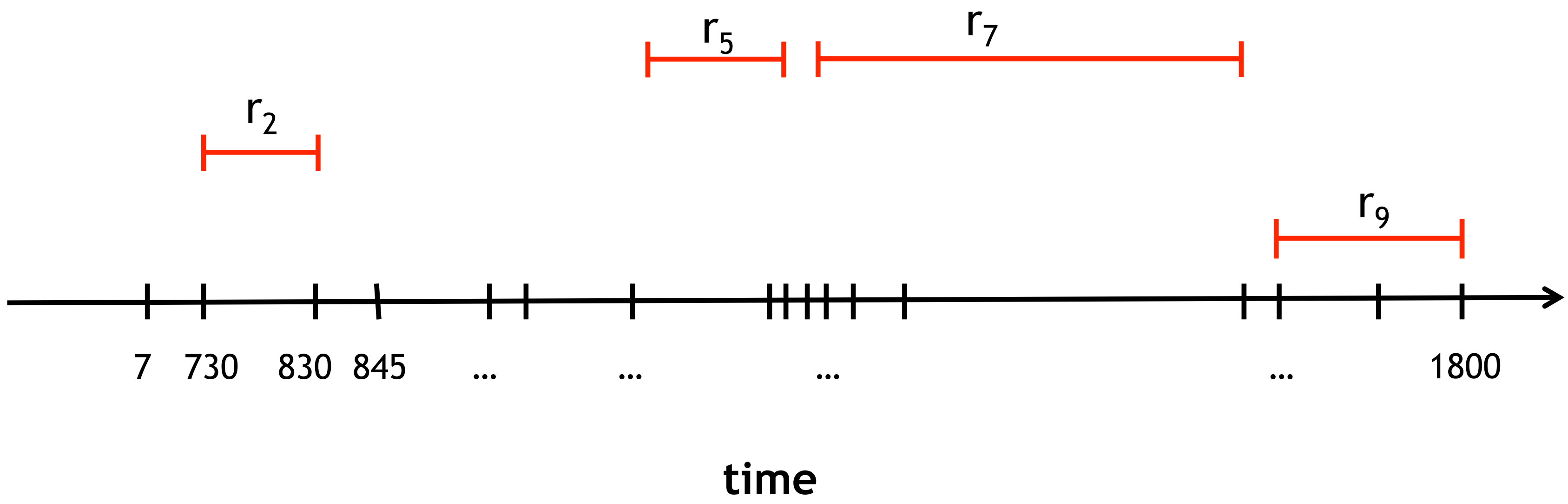
Possible Greedy Strategies (4)

- Earliest-Finishing-Request: Pick the earliest finishing request
- Remove any overlapping request
- Repeat until no requests left



Possible Greedy Strategies (4)

- Earliest-Finishing-Request: Pick the earliest finishing request
- Remove any overlapping request
- Repeat until no requests left

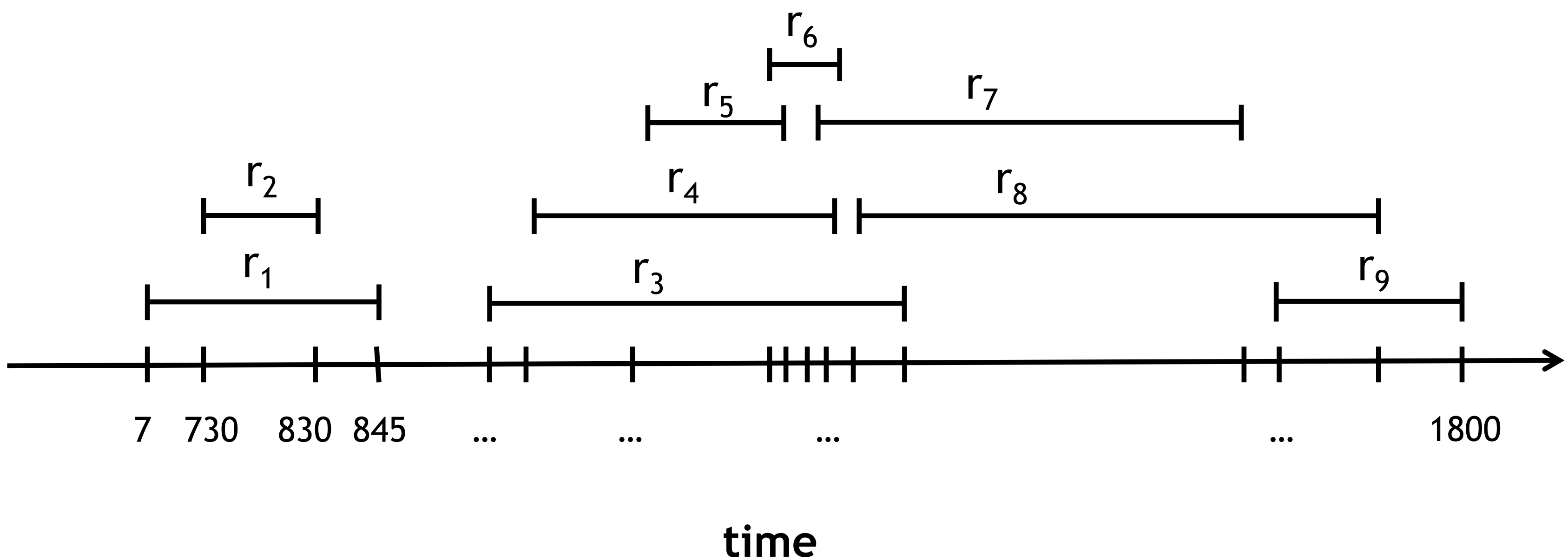


- Looks Optimal

What's the intuition?

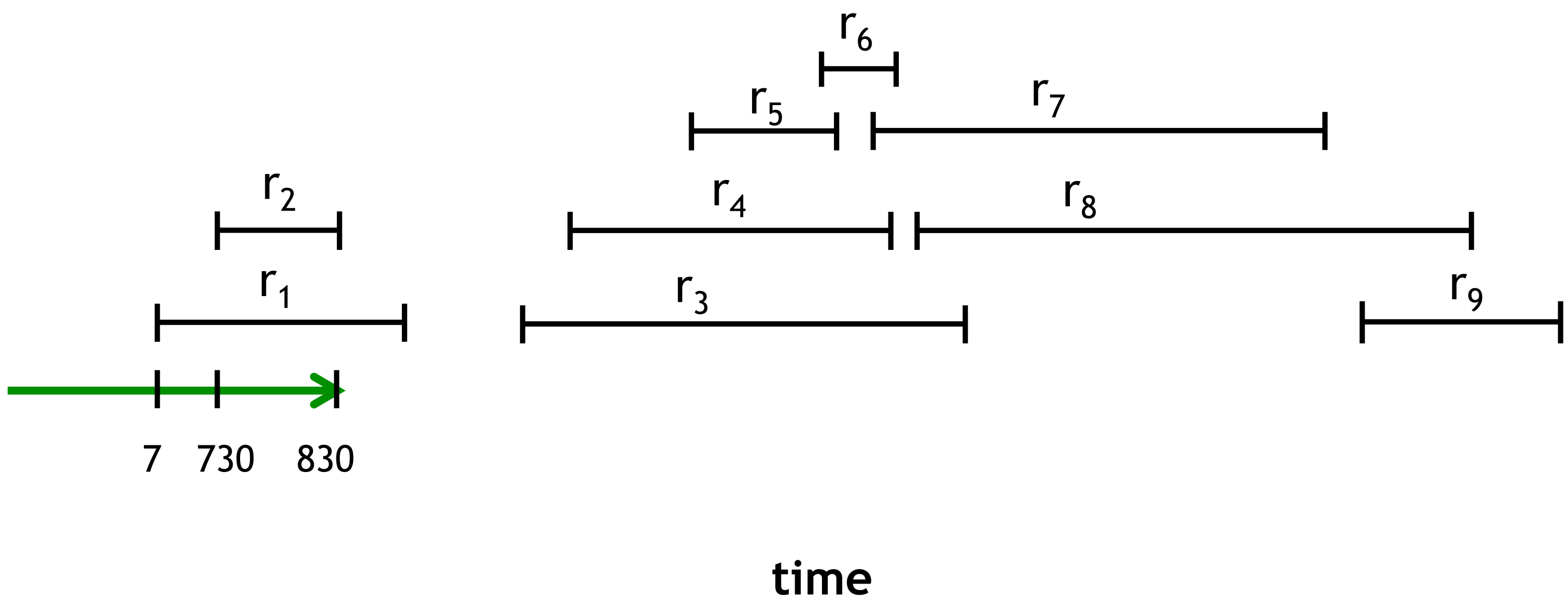
Intuition

- Q: What's the earliest time at which 1 request can be "fulfilled", i.e. accepted and executed?



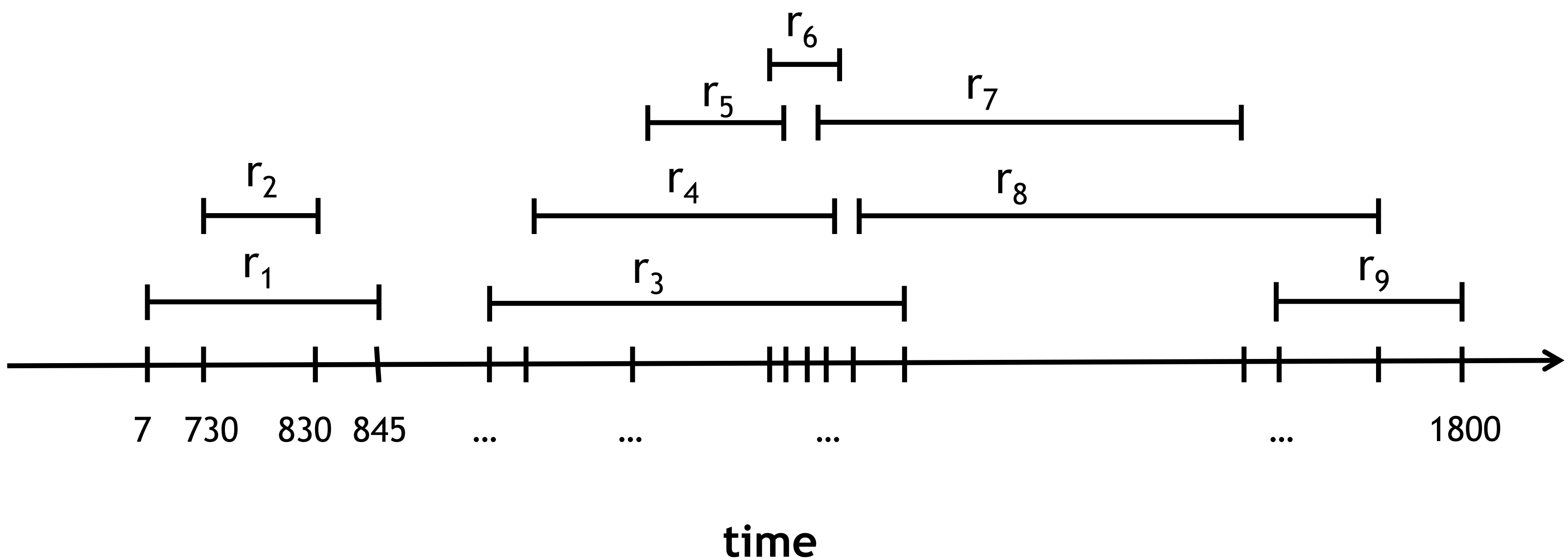
Intuition

- Q: What's the earliest time at which 1 request can be "fulfilled", i.e. accepted and executed?
- A: Earliest finishing time of any job (e.g. 8:30am)



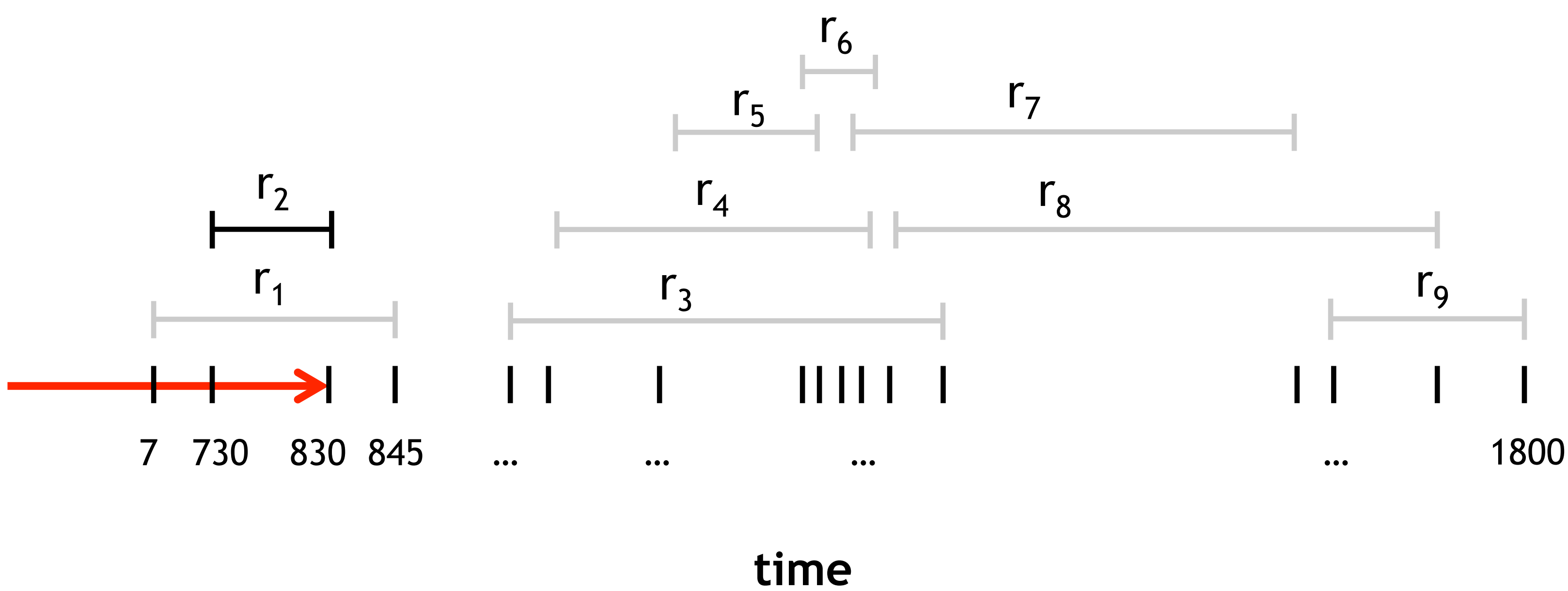
Intuition

- Q: What's the earliest time at which 2 requests can be fulfilled?



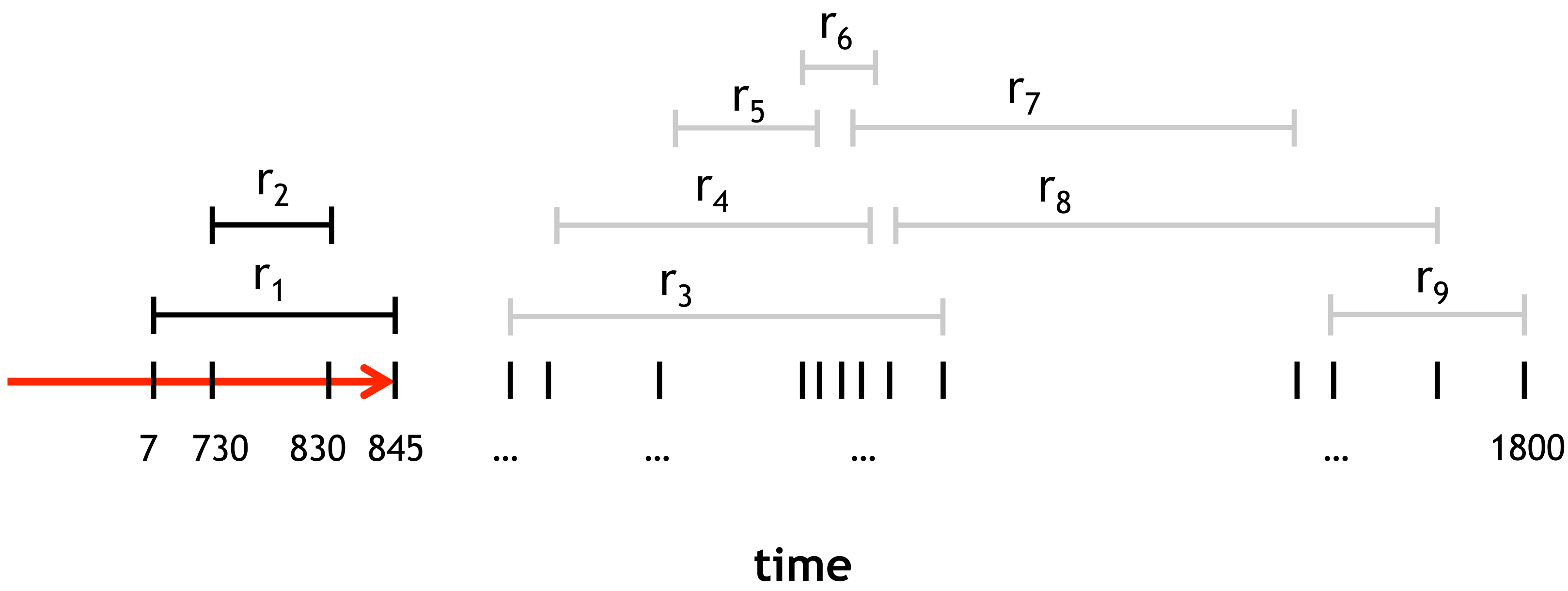
Intuition

- Q: What's the earliest time at which 2 requests can be fulfilled?
- Q2: Can we fulfill 2 requests by 8:30am?
- A: No



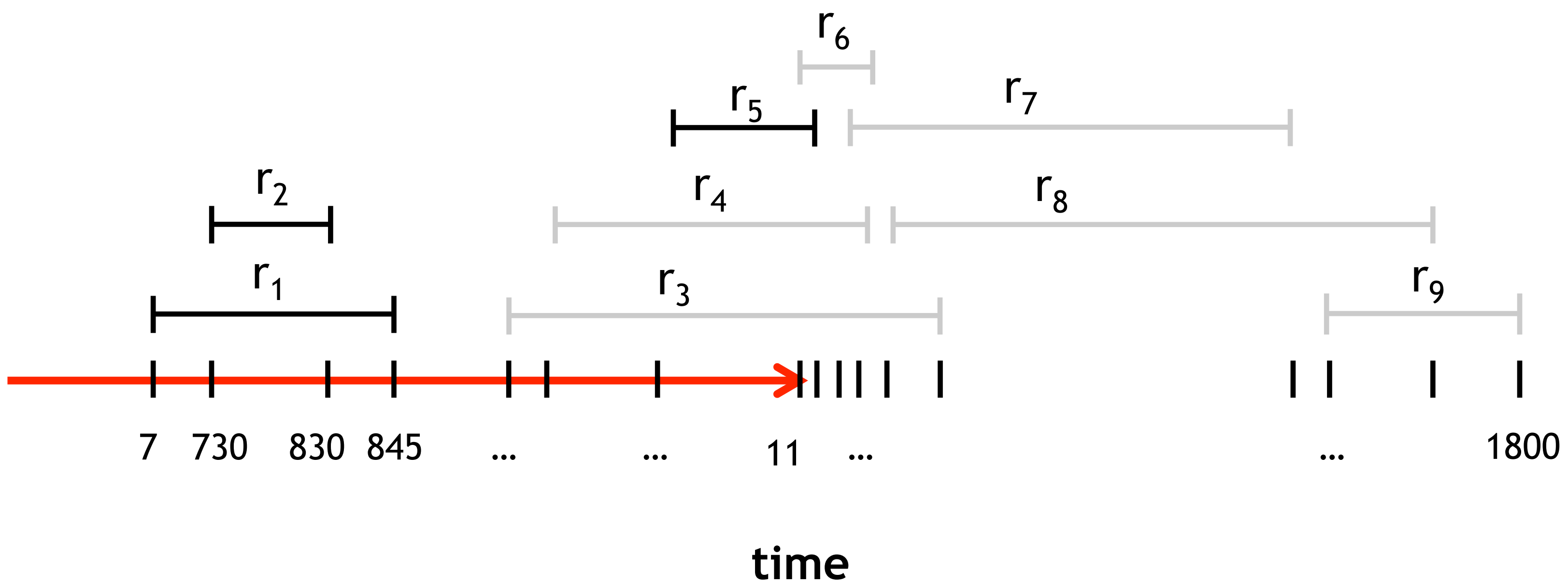
Intuition

- Q: What's the earliest time at which 2 requests can be fulfilled?
- Q2: Can we accept 2 requests by 8:45am?
- A: No



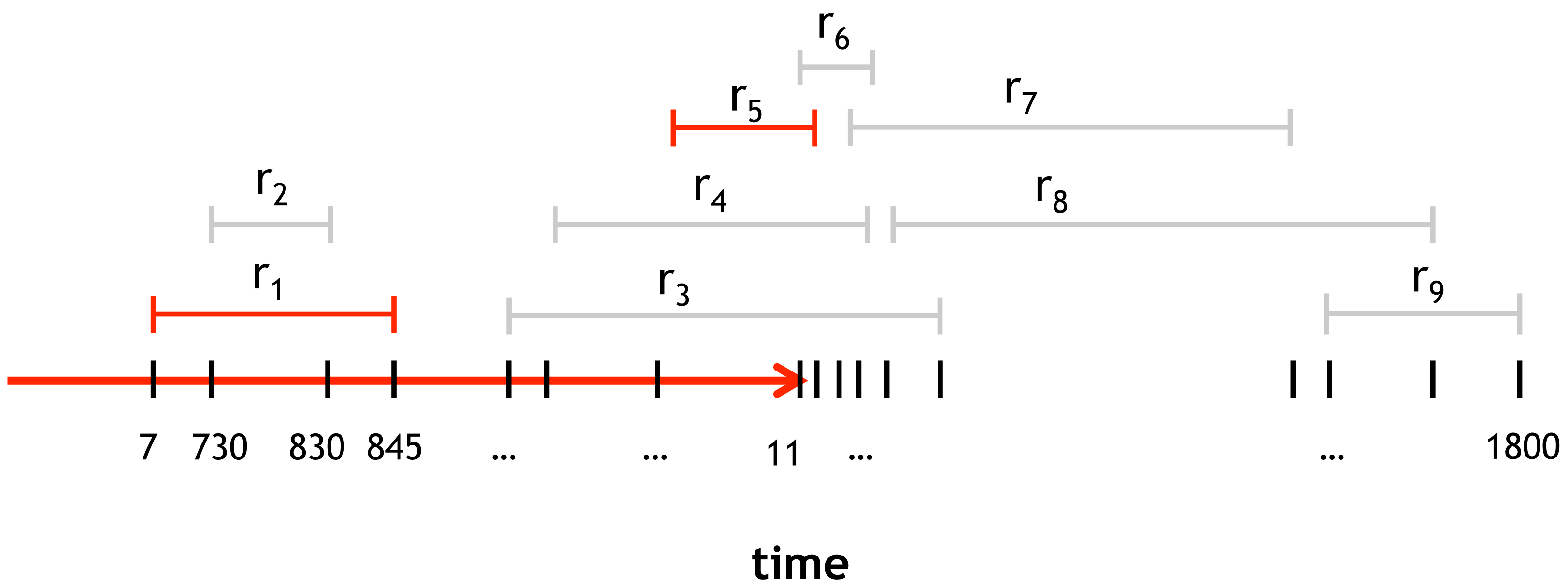
Intuition

- Q: What's the earliest time at which 2 requests can be fulfilled?
- Q2: Can we accept 2 requests by r_5's finishing time (say 11am)?
- A: Yes: $\{r_1, r_5\}$ or $\{r_2, r_5\}$



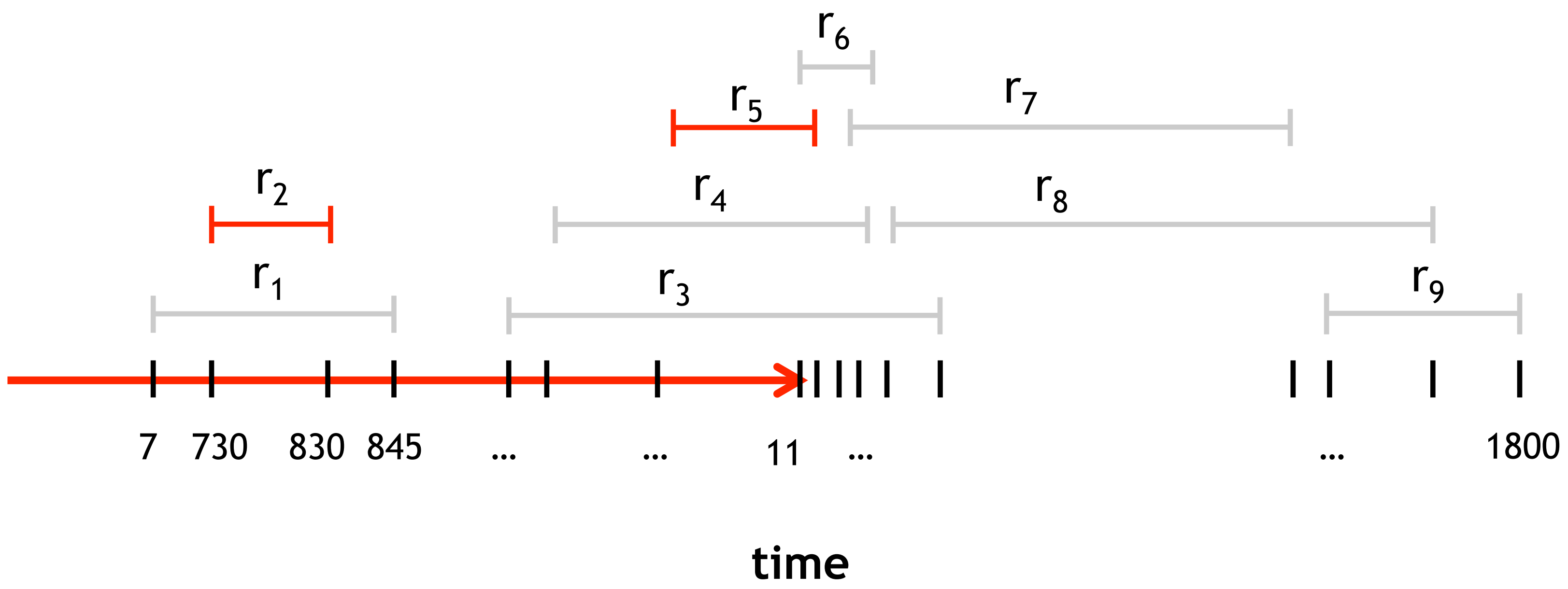
Intuition

- Q: What's the earliest time at which 2 requests can be fulfilled?
- Q2: Can we accept 2 requests by r_5's finishing time (say 11am)?
- A: Yes: $\{r_1, r_5\}$ or $\{r_2, r_5\}$



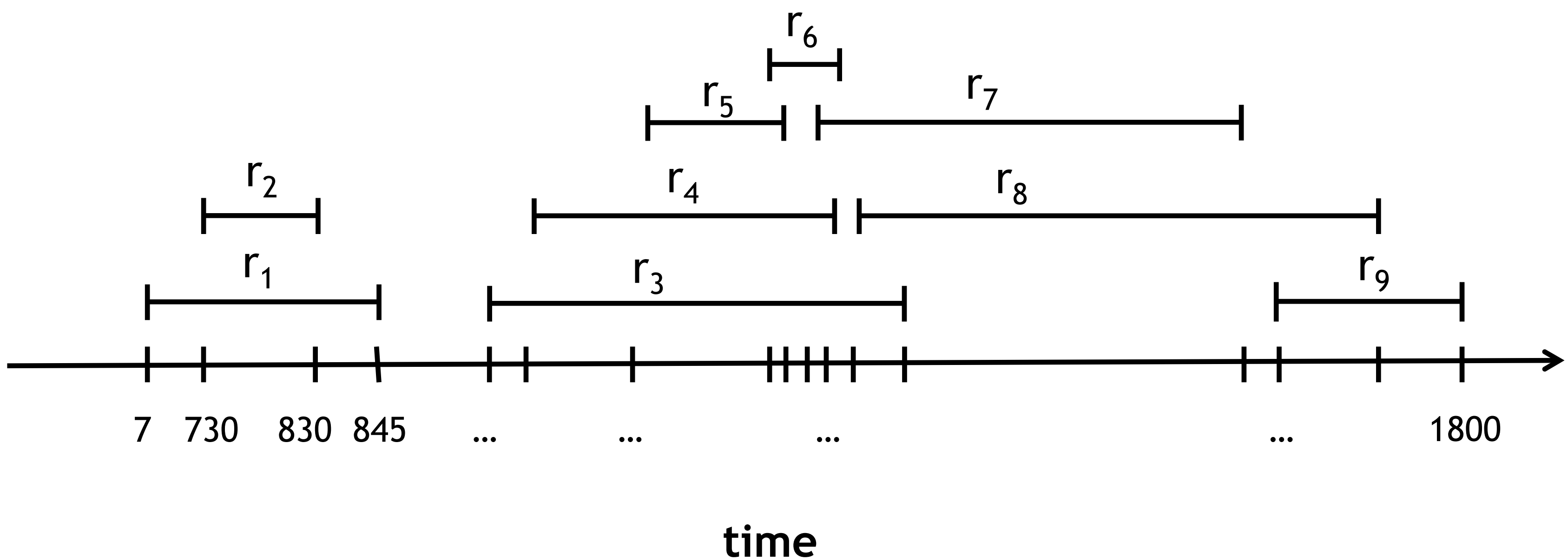
Intuition

- Q: What's the earliest time at which 2 requests can be fulfilled?
- Q2: Can we accept 2 requests by r_5's finishing time (say 11am)?
- A: Yes: $\{r_1, r_5\}$ or $\{r_2, r_5\}$



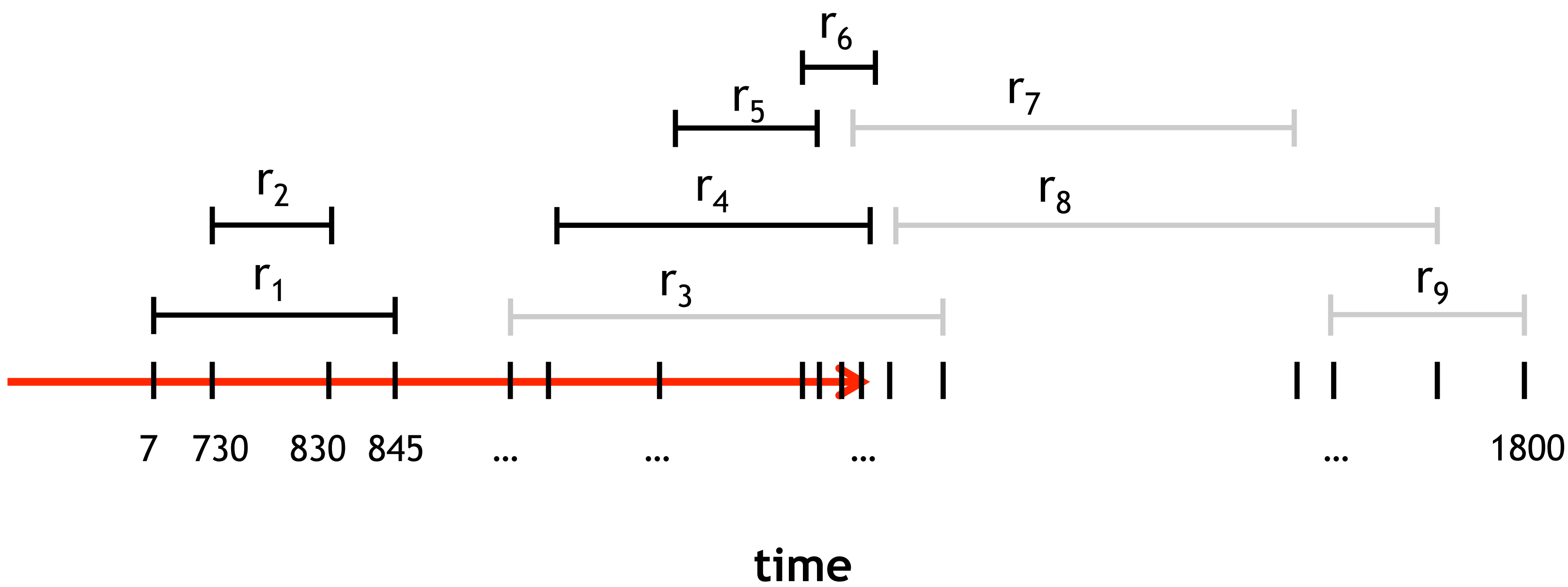
Intuition

- Q: What's the earliest time at which 3 requests can be fulfilled?



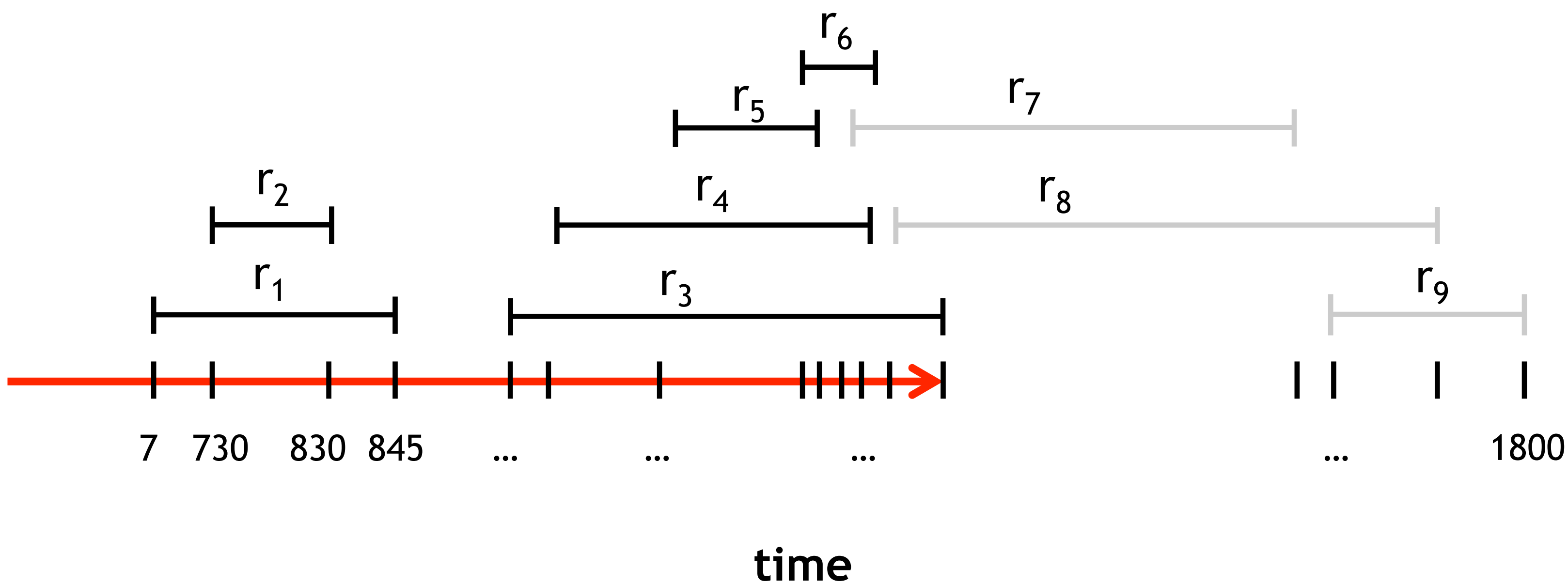
Intuition

- Q: What's the earliest time at which 3 requests can be fulfilled?
- Q2: Can we fulfill 3 requests by r_4 and r_6's finishing time?
- A: No



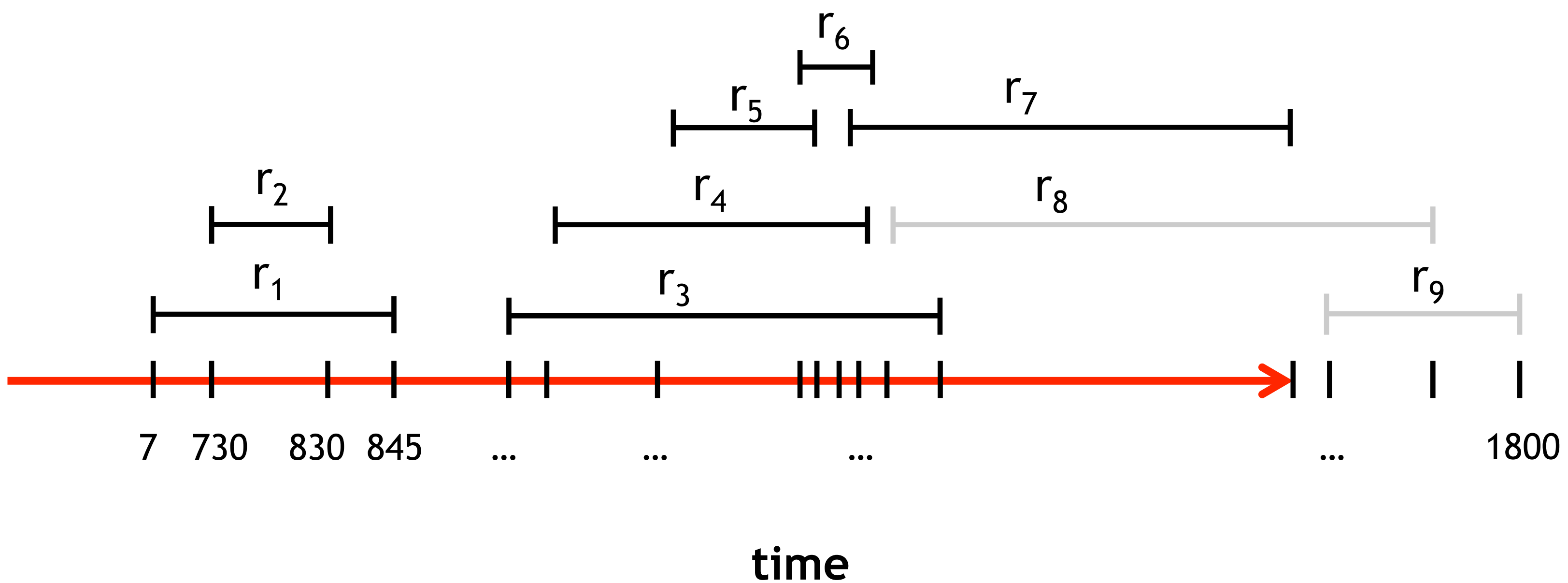
Intuition

- Q: What's the earliest time at which 3 requests can be fulfilled?
- Q2: Can we fulfill 3 requests by r_3's finishing time?
- A: No



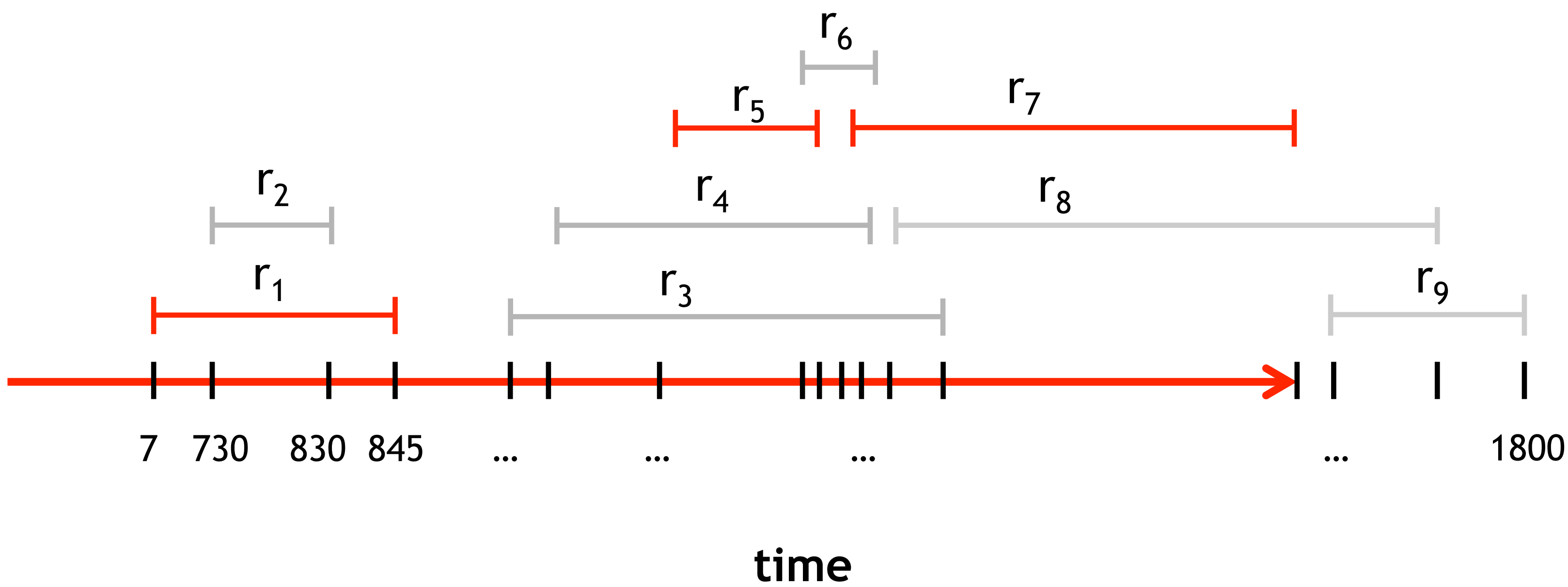
Intuition

- Q: What's the earliest time at which 3 requests can be fulfilled?
- Q2: Can we fulfill 3 requests by r_7's finishing time?
- A: Yes = $\{r_1, r_5, r_7\}$ or $\{r_2, r_5, r_7\}$



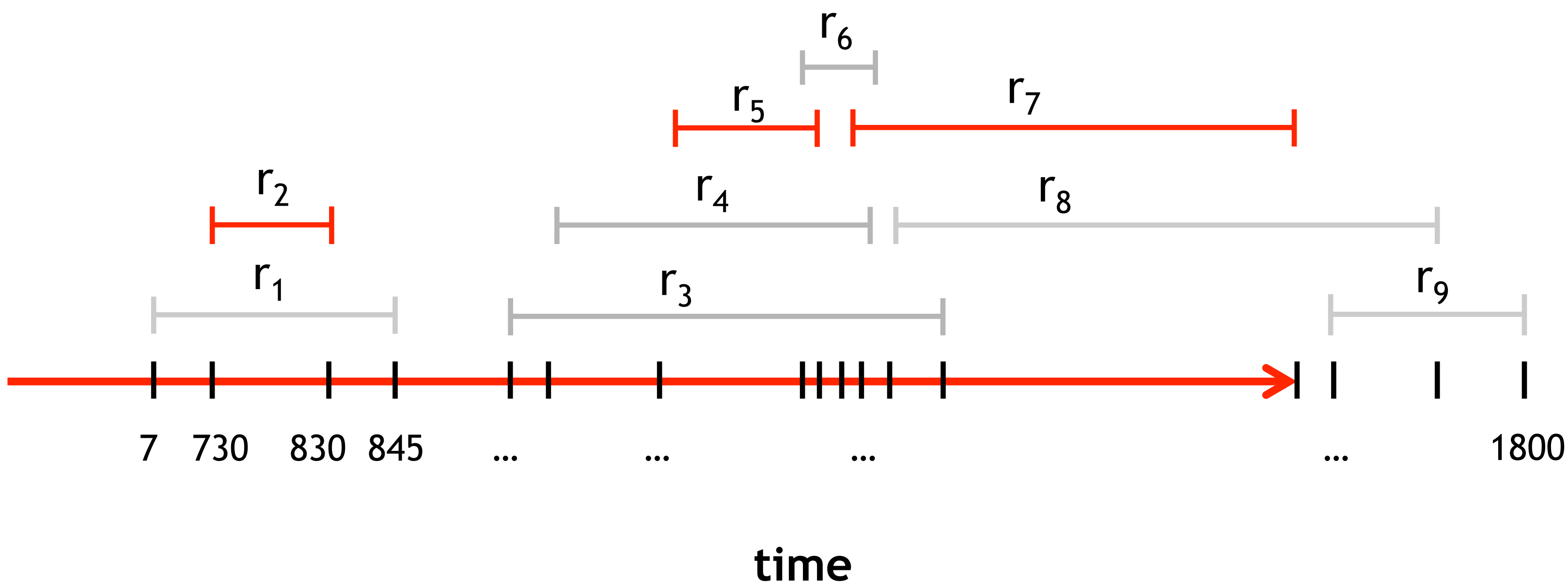
Intuition

- Q: What's the earliest time at which 3 requests can be fulfilled?
- Q2: Can we fulfill 3 requests by r_7's finishing time?
- A: Yes = $\{r_1, r_5, r_7\}$ or $\{r_2, r_5, r_7\}$



Intuition

- Q: What's the earliest time at which 3 requests can be fulfilled?
- Q2: Can we fulfill 3 requests by r_7's finishing time?
- A: Yes = $\{r_1, r_5, r_7\}$ or $\{r_2, r_5, r_7\}$



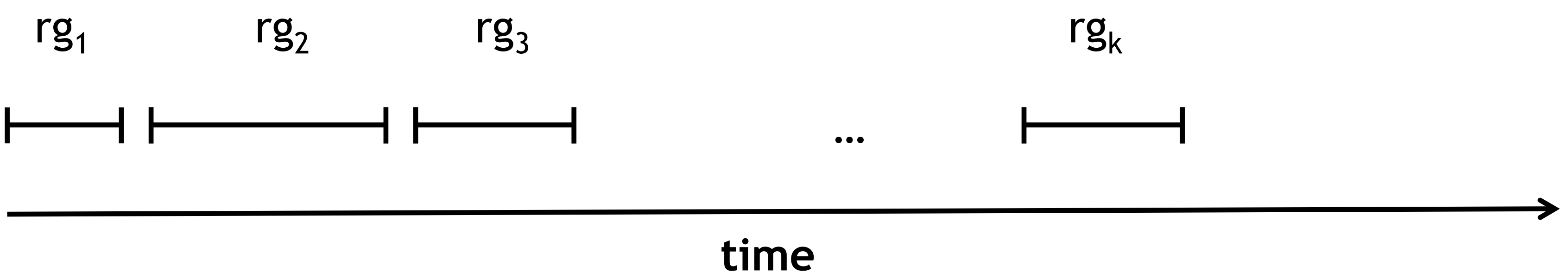
Possible Claim: Earliest time that we can fulfill i jobs is the finishing time of the i-th job that earliest-ft greedy picks. Looks like "Greedy is Staying Ahead".

Key Claim: Greedy Stays Ahead

Let rg_1, rg_2, …, rg_k be the k request that earliest-ft-greedy picks.

Key Claim: Earliest time that we can fullfill i requests is the $f(rg_i)$.

(i.e. finishing time of the i-th request that earliest-ft-greedy picks)



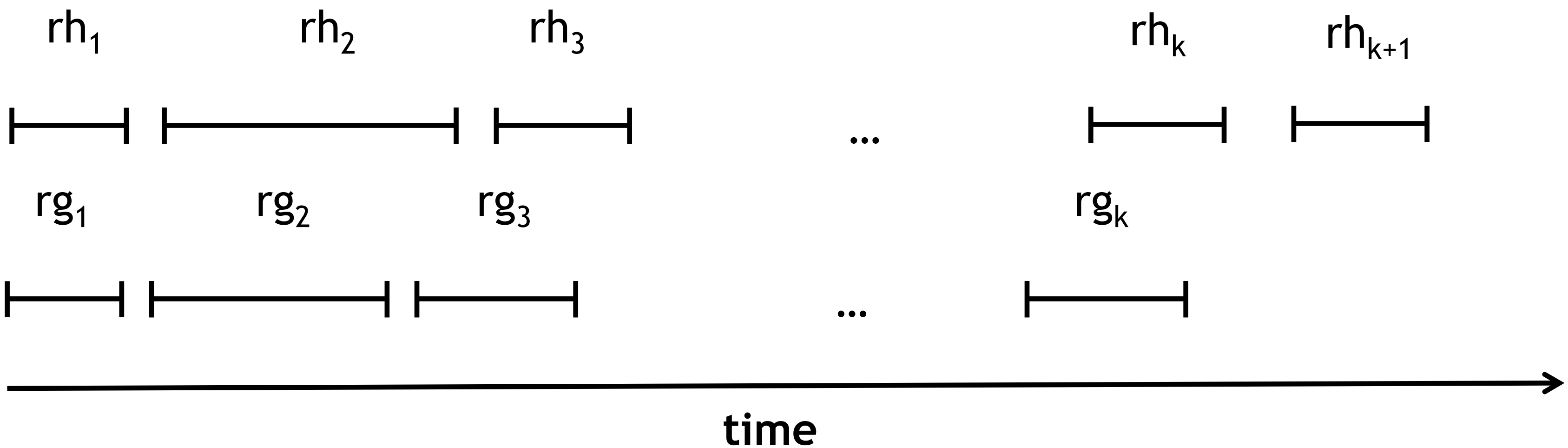
Claim of Optimality: If Key Claim is true, then earliest-ft-greedy is optimal!

Why?

Proof of Optimality

Suppose there is another selection that accepts ≥ k+1 requests.

Call those requests rh_1 …, rh_{k+1}, … for “request-hypothetical”



By Key Claim, $f(rh_k) \geq f(rg_k)$.

Which means $s(rh_{k+1})$ and $f(rh_{k+1})$ are both $\geq f(rg_k)$.

But such a request k+1 cannot exist b/c earliest-ft-greedy would not have stopped at k and have picked it as k+1st request as well.

Proof of Key Claim

Upshot: Just a repetition of the proof of optimality!

Let rg_1, rg_2, …, rg_k be the k request that earliest-ft-greedy picks.

Key Claim: Earliest time that we can fullfill i requests is $f(rg_i)$.

Proof by Induction on i:

Base Case i=1: Holds b/c $f(rg_1)$ is the earliest finishing time of all requests.

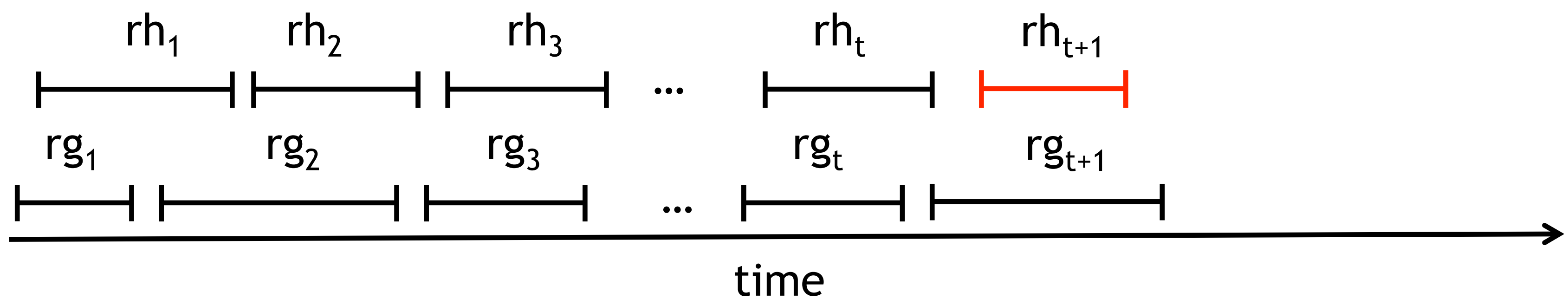
Inductive Hypothesis. Suppose claim holds for i=t. (Now show holds for i=t+1).

Suppose for purpose of contradiction another schedule rh_1, rh_2, …, rh_t, rh_{t+1} is s.t.

$$f(rh_{t+1}) < f(rg_{t+1}).$$

We know that $f(rg_t) \le f(rh_t) \le s(rh_{t+1}) \le f(rh_{t+1}) \le f(rg_{t+1})$

This is a contradiction b/c then rh_{t+1} does not overlap with rg_1, …, rg_t and by the greedy criterion earliest-ft-greedy would pick rh_{t+1} over rg_{t+1} but didn't.



Full Algorithm

```
procedure earliestFTGreedy(Array R of size n):
   sort(R by finishing times)
   Sg = { R[0] }; // insert the first job
   for (i = 1; i < n; i++):
      if (R[i].start < Sg.last.finish) {
         i++;
      } else {
         Sg.insert(R[i]);
      }
   }
   return Sg
```

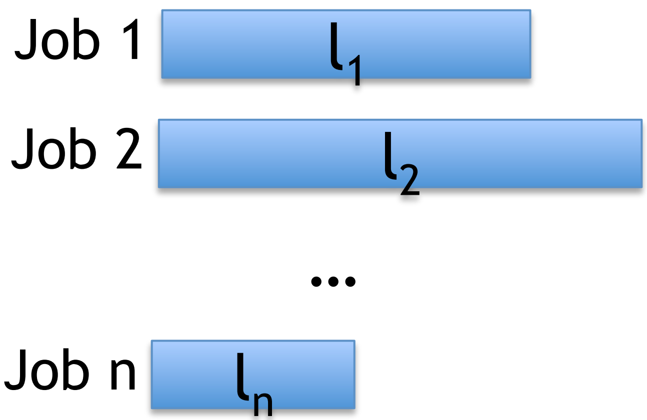
Runtime: O(nlog(n))

Outline For Today

1. Introduction to Greedy Algorithms
2. Activity Selection
3. Job Scheduling 1
4. Job Scheduling 2

Scheduling Problem 1

- **Input:** A set of n jobs J. Each job_i has length l_i



- **Output:** A schedule of the jobs on a processor

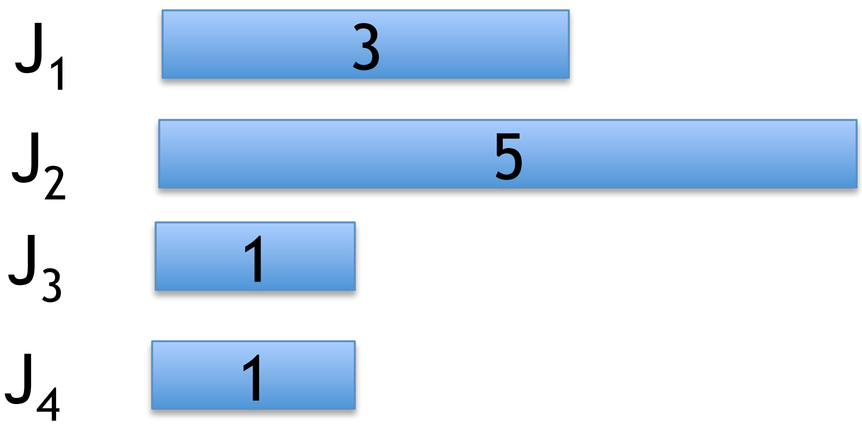
s.t: $\sum_{i=1}^{n} C_i \leftarrow$ completion time of job i

is minimum over all possible n! schedules.

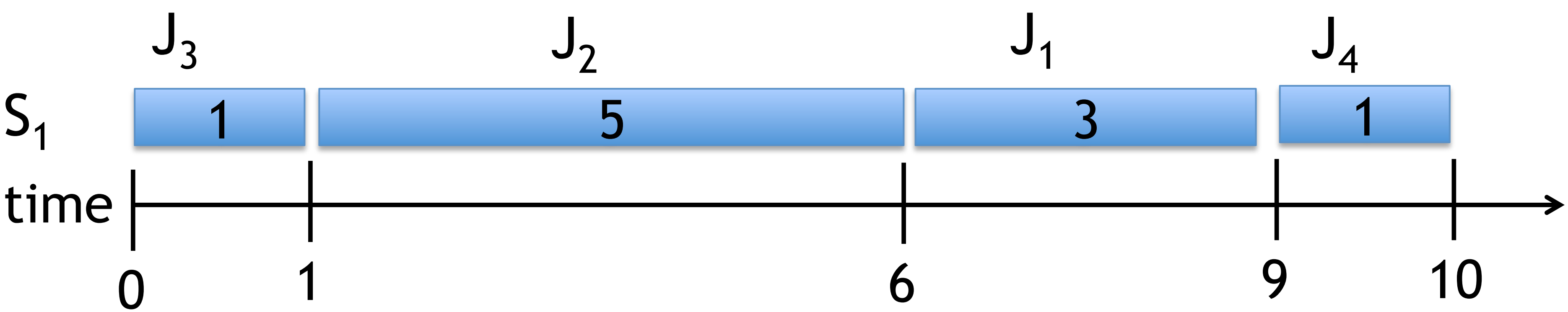
Completion Time of Job i

- **Definition:** time when job_i finishes

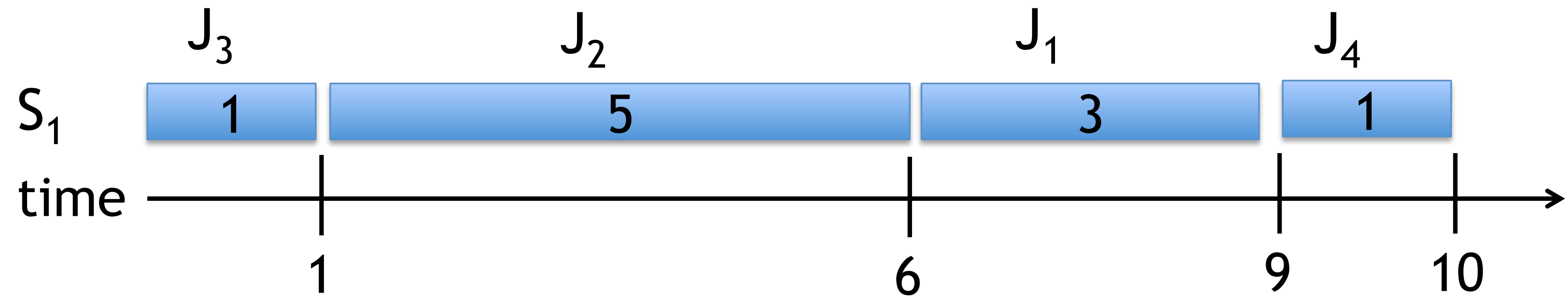
  i.e., sum of scheduled job lengths up to and including job_i



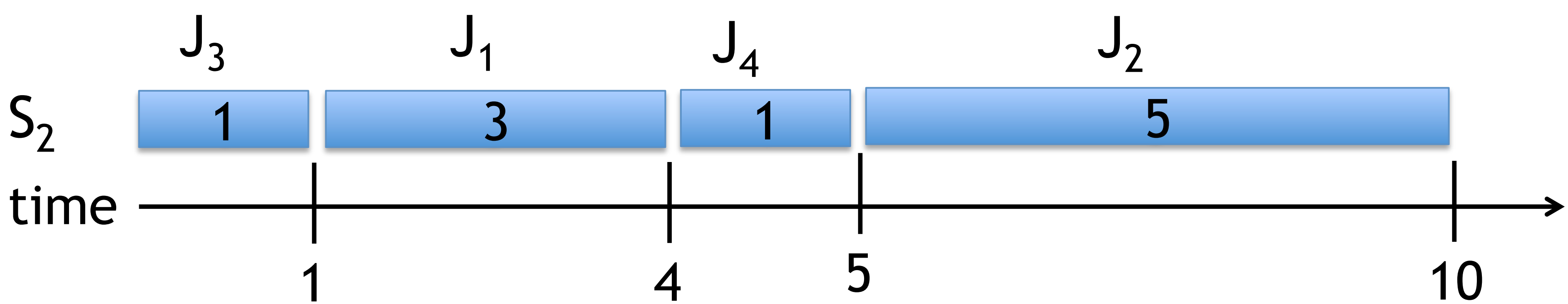
Total Cost of S_1:
1 + 6 + 9 + 10 = 26



Another Example Schedule



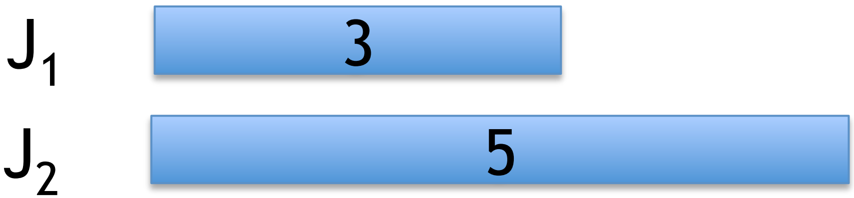
Total Cost of S_1: 1 + 6 + 9 + 10 = 26



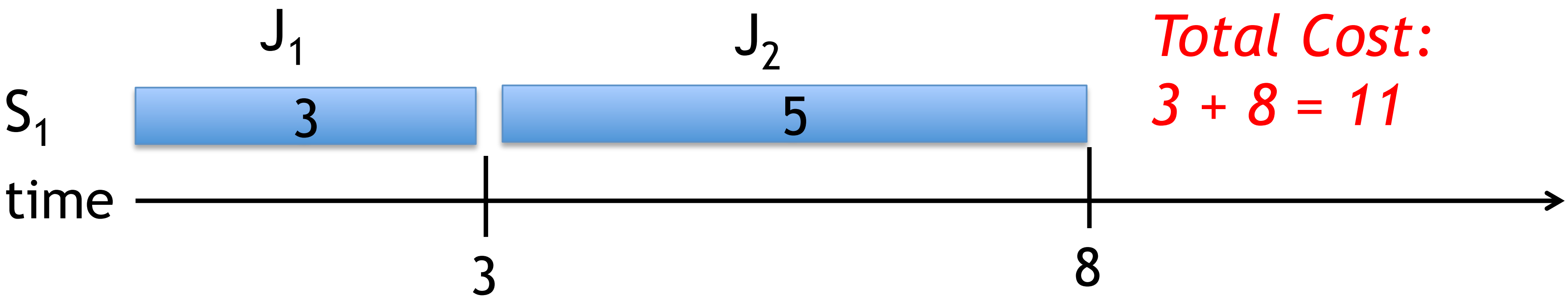
Total Cost of S_2:1 + 4 + 5 + 10 = 20

Goal is to find the min cost schedule!

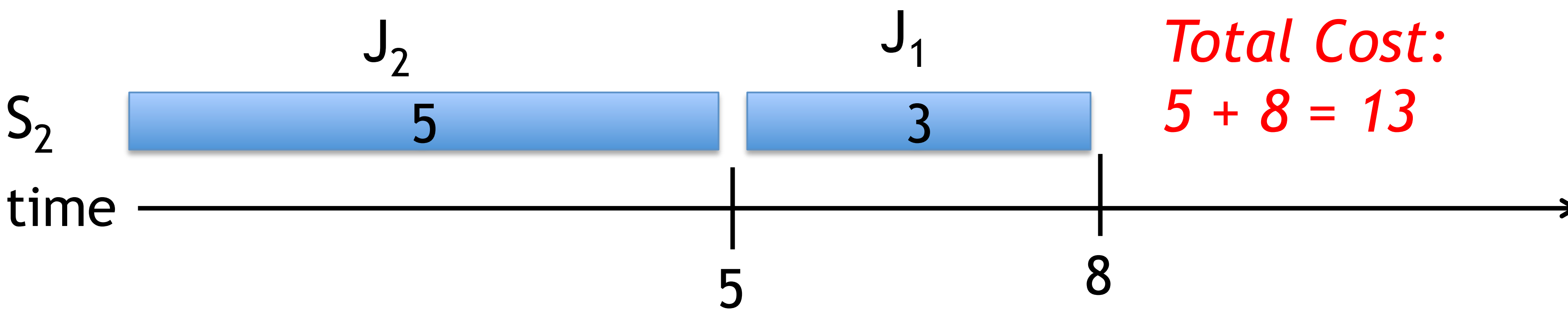
Let’s Start Simple



What are all possible schedules?



Total Cost:
3 + 8 = 11



Total Cost:
5 + 8 = 13

Why Put One Job In Front of Another?

- Observation:

  Shorter jobs have less impact on the completion times of future jobs

Greedy Scheduling Algorithm

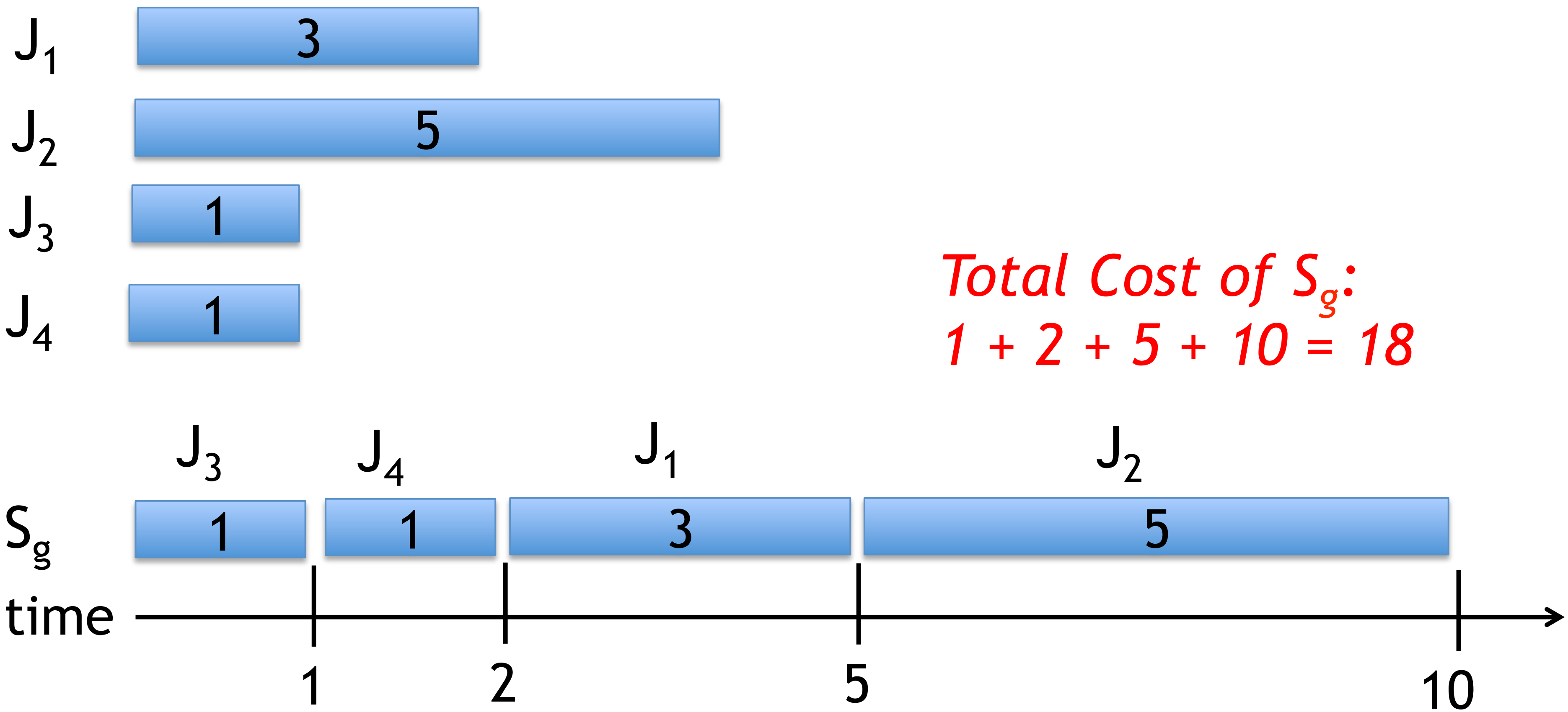
Schedule jobs by increasing lengths

```
procedure greedySchedule(Array J of size n):
    return sort(J)
```

Run-time O(nlog(n))!

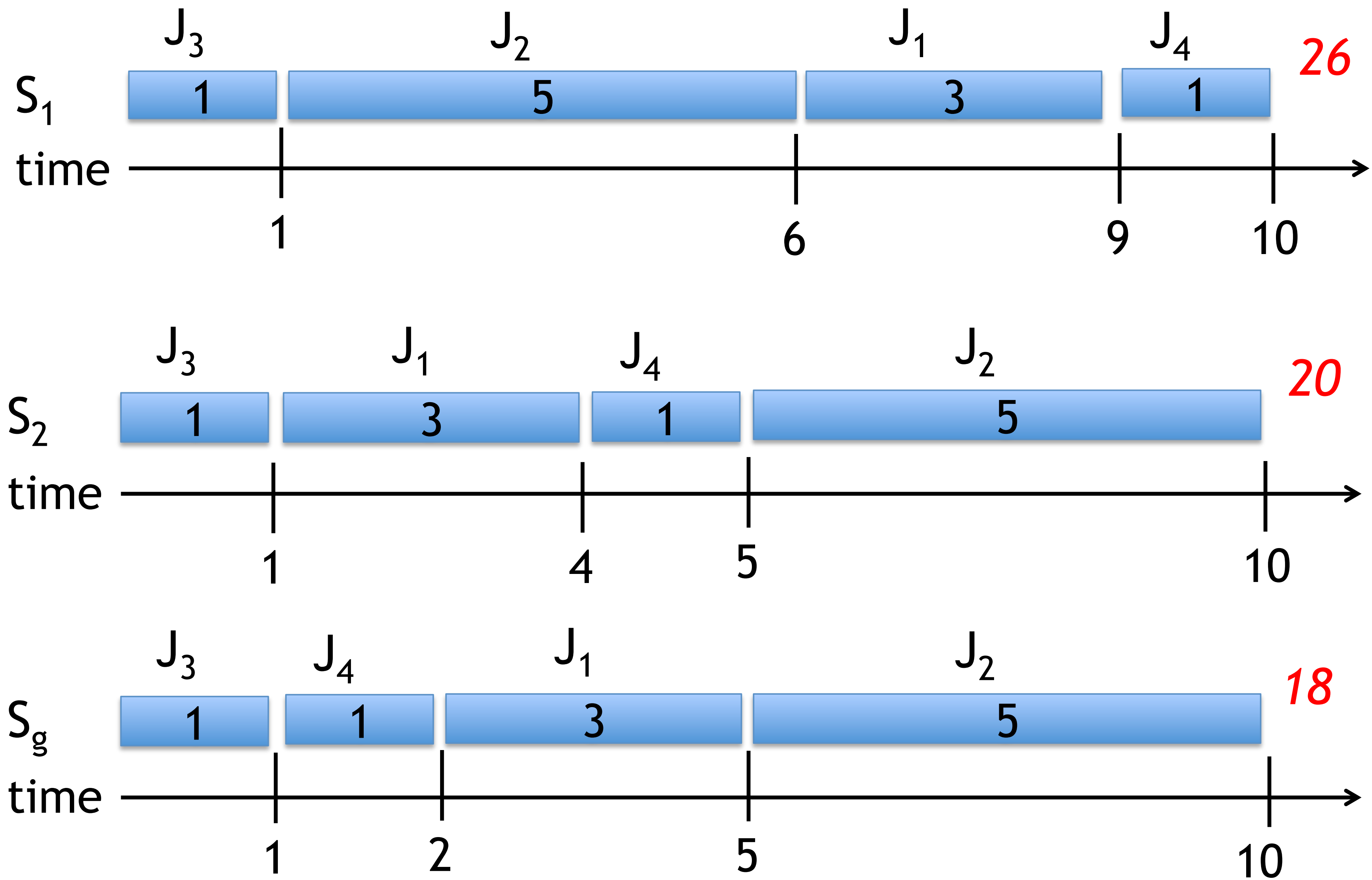
Greedy Algorithm 1

Ex:



Total Cost of S_g:
1 + 2 + 5 + 10 = 18

Comparing S_g to Previous Schedules



Proof of Correctness (1)

- "Greedy stays ahead" proof:
  - Induct on the cost of the first k jobs executed
  - Argue S_g beats everyone else at each step
- Let S[i]: the ith job that a schedule S executes
  - E.g., $S_g[1]$ is the first job S_g executes
- Let Cost(S, i): be the sum of the costs of the first i jobs that schedule S executes.
  - E.g., Cost(S_g, 3) is the sum of completion times $S_g[1]$, $S_g[2]$, $S_g[3]$: $S_g[1] + (S_g[1]+S_g[2]) + (S_g[1]+S_g[2]+S_g[3])$
- Goal: Argue $\forall S$, $Cost(S_g, n) \le Cost(S, n)$ by inducting on i

Proof of Correctness (2)

- Base Case: $\forall S$, $Cost(S_g, 1) = S_g[1] \le Cost(S, 1)$ since $S_g[1]$ is the shortest length job
- Inductive Hypothesis: $Cost(S_g, k\text{-}1) \le Cost(S, k\text{-}1)$

$$Cost(S_g, k) = Cost(S_g, k-1) + \sum_{i=1}^{k} length(S_g[i])$$

$$\le \qquad\qquad \le \qquad\qquad \le$$

$$Cost(S, k) = Cost(S, k-1) + \sum_{i=1}^{k} length(S[i])$$

By inductive hypothesis

By greedy criterion of S_g

QED

Outline For Today

1. Introduction to Greedy Algorithms
2. Activity Selection
3. Job Scheduling 1
4. Job Scheduling 2

Scheduling Problem 2

- **Input:** Now each job_i has length l_i AND weight w_i

Job 1 l_1, w_1

Job 2 l_2, w_2

...

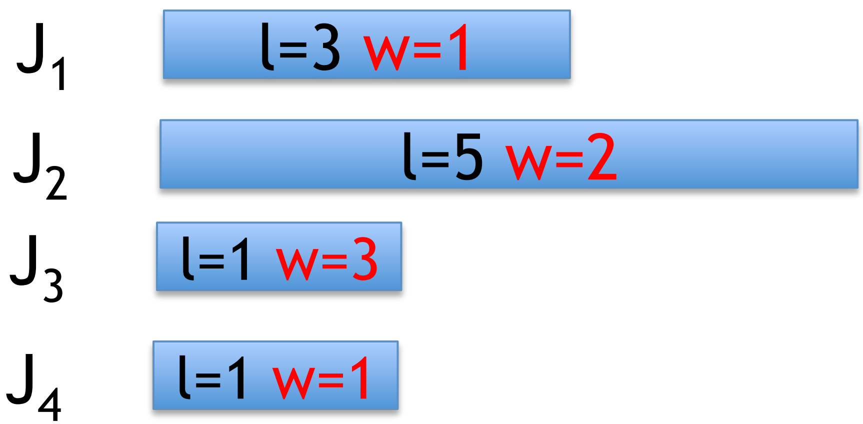
Job n l_n, w_n

- **Output:** A schedule of the jobs on a processor

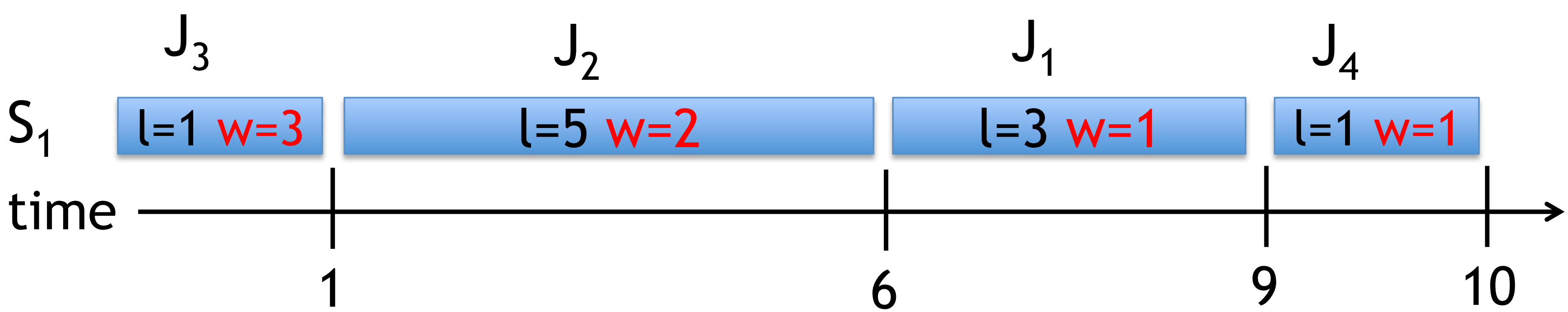
s.t: $\sum_{i=1}^{n} w_i C_i$ ← weighted completion time of job i

is minimum over all possible n! schedules.

Example Schedule And Cost



Total Cost of S_1:
$3*1 + 2*6 + 1*9 + 1*10 = 34$



Q1: What To Do When Weights Are Same?

Same As Un-weighted Case → Shorter lengths first

Previous Greedy Algorithm is Optimal

Q2: What To Do When Lengths Are Same?

J_1 l=3 w=2

J_2 l=3 w=3

J_3 l=3 w=1

J_4 l=3 w=4

minimize: $\sum_{i=1}^{n} w_i C_i$

Higher weights first

Q3: What To Do With Mixed L & W?

- Say J_1 is shorter and also has less weight than J_2?

J_1 l=3 w=1

J_2 l=5 w=2

minimize: $$\sum_{i=1}^{n} w_i C_i$$

- Unclear:
  - Intuition for Q1 says J1 should come first
  - Intuition for Q2 says J2 should come first

Ideal Scenario: Combine l and w into a single score that combines both intuitions and we could order by that score

Possible Combined Scores?

- The combined score $f(l_i, w_i)$ should satisfy:
  1. If weights same → shorter lengths get smaller scores
  2. If lengths same → larger weights get smaller scores
- Guess 1: $f_1(l_i, w_i) = l_i - w_i$
- Guess 2: $f_2(l_i, w_i) = l_i / w_i$

Is either one correct?

Let's First Try To Eliminate One Guess

J_1 $l=3$ $w=1$

J_2 $l=5$ $w=2$

	$f_1 = l_i - w_i$	$f_2 = l_i/w_i$
J_1 ($l=3$, $w=1$)		
J_2 ($l=5$, $w=2$)		
SCHEDULE		
TOTAL COST		

Let's First Try To Eliminate One Guess

J_1 l=3 w=1

J_2 l=5 w=2

	$f_1 = l_i - w_i$	$f_2 = l_i/w_i$
J_1 (l=3, w=1)	2	
J_2 (l=5, w=2)		
SCHEDULE		
TOTAL COST		

Let's First Try To Eliminate One Guess

J_1 l=3 w=1

J_2 l=5 w=2

	$f_1 = l_i - w_i$	$f_2 = l_i/w_i$
J_1 (l=3, w=1)	2	
J_2 (l=5, w=2)	3	
SCHEDULE		
TOTAL COST		

Let's First Try To Eliminate One Guess

J_1 l=3 w=1

J_2 l=5 w=2

	$f_1 = l_i - w_i$	$f_2 = l_i/w_i$
J_1 (l=3, w=1)	2	
J_2 (l=5, w=2)	3	
SCHEDULE	$J_1::J_2$	
TOTAL COST		

Let's First Try To Eliminate One Guess

J_1 l=3 w=1

J_2 l=5 w=2

	$f_1 = l_i - w_i$	$f_2 = l_i/w_i$
J_1 (l=3, w=1)	2	
J_2 (l=5, w=2)	3	
SCHEDULE	J_1::J_2	
TOTAL COST	1*3 + 2*8 = 19	

Let's First Try To Eliminate One Guess

J_1 $l=3$ $w=1$

J_2 $l=5$ $w=2$

	$f_1 = l_i - w_i$	$f_2 = l_i/w_i$
J_1 ($l=3$, $w=1$)	2	3
J_2 ($l=5$, $w=2$)	3	
SCHEDULE	$J_1::J_2$	
TOTAL COST	1*3 + 2*8 = 19	

Let's First Try To Eliminate One Guess

J_1 | l=3 w=1

J_2 | l=5 w=2

	$f_1 = l_i - w_i$	$f_2 = l_i/w_i$
J_1 (l=3, w=1)	2	3
J_2 (l=5, w=2)	3	2.5
SCHEDULE	$J_1::J_2$	
TOTAL COST	1*3 + 2*8 = 19	

Let’s First Try To Eliminate One Guess

J_1 l=3 w=1

J_2 l=5 w=2

	$f_1 = l_i - w_i$	$f_2 = l_i/w_i$
J_1 (l=3, w=1)	2	3
J_2 (l=5, w=2)	3	2.5
SCHEDULE	$J_1::J_2$	$J_2::J_1$
TOTAL COST	1*3 + 2*8 = 19	

Let's First Try To Eliminate One Guess

J_1 l=3 w=1

J_2 l=5 w=2

	$f_1 = l_i - w_i$	$f_2 = l_i/w_i$
J_1 (l=3, w=1)	2	3
J_2 (l=5, w=2)	3	2.5
SCHEDULE	$J_1::J_2$	$J_2::J_1$
TOTAL COST	1*3 + 2*8 = 19	2*5 + 1*8 = 18

Guess 1 is certainly not optimal.

Is Guess 2 optimal?

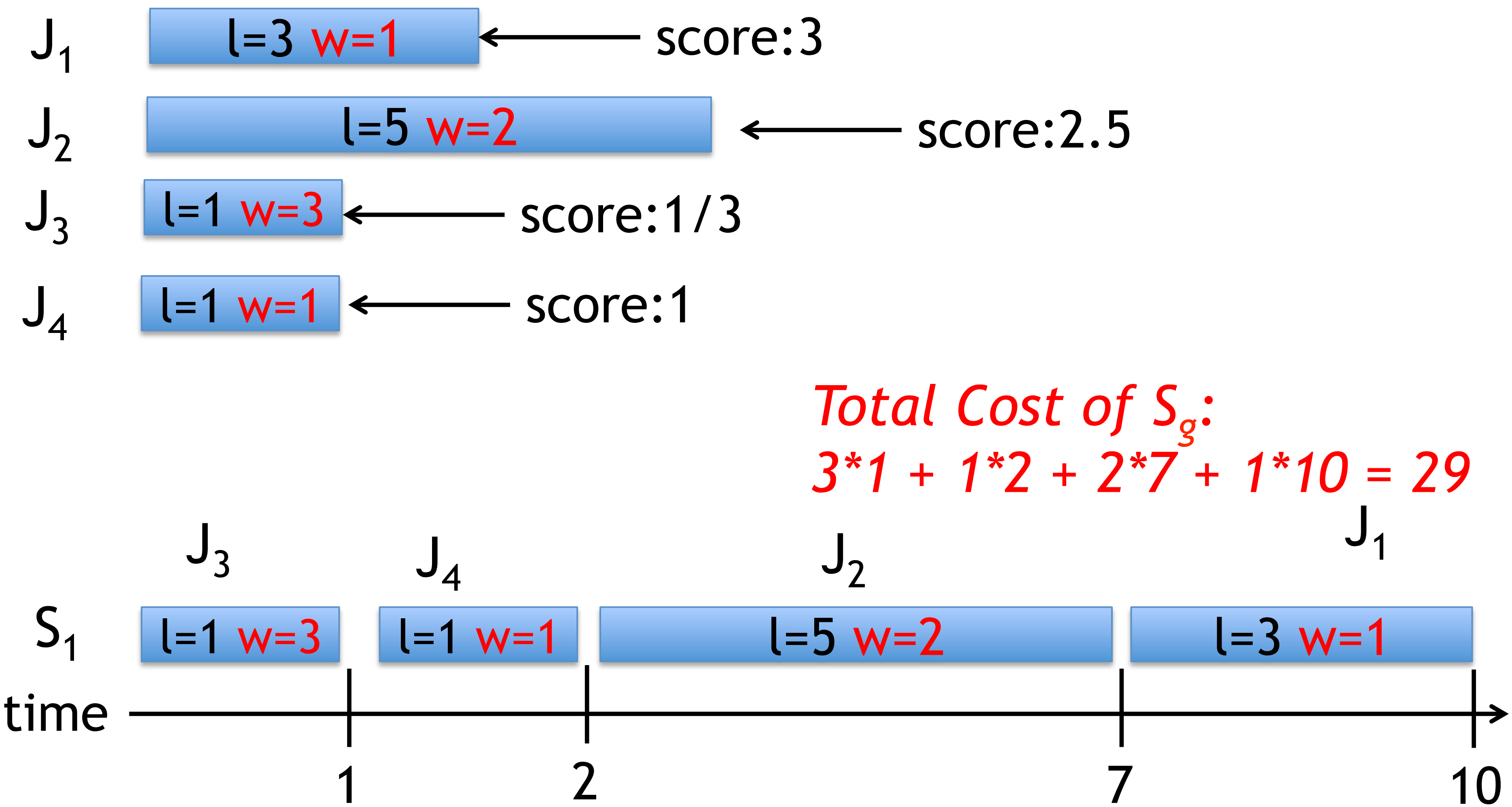
Greedy Weighted Scheduling Algorithm

Schedule jobs by increasing l_i/w_i scores

```
procedure greedySchedule(Array J of size n):
    return sort(J by li/wi scores)
```

Run-time O(nlog(n))!

Greedy Weighted Scheduling Algorithm



Proof of Correctness (1)

- By Exchange Argument:
  - Argue *any S can be transformed into S_g* step by step and *without getting worse along the way*
- Let's rename jobs so that S_g schedules jobs in order:

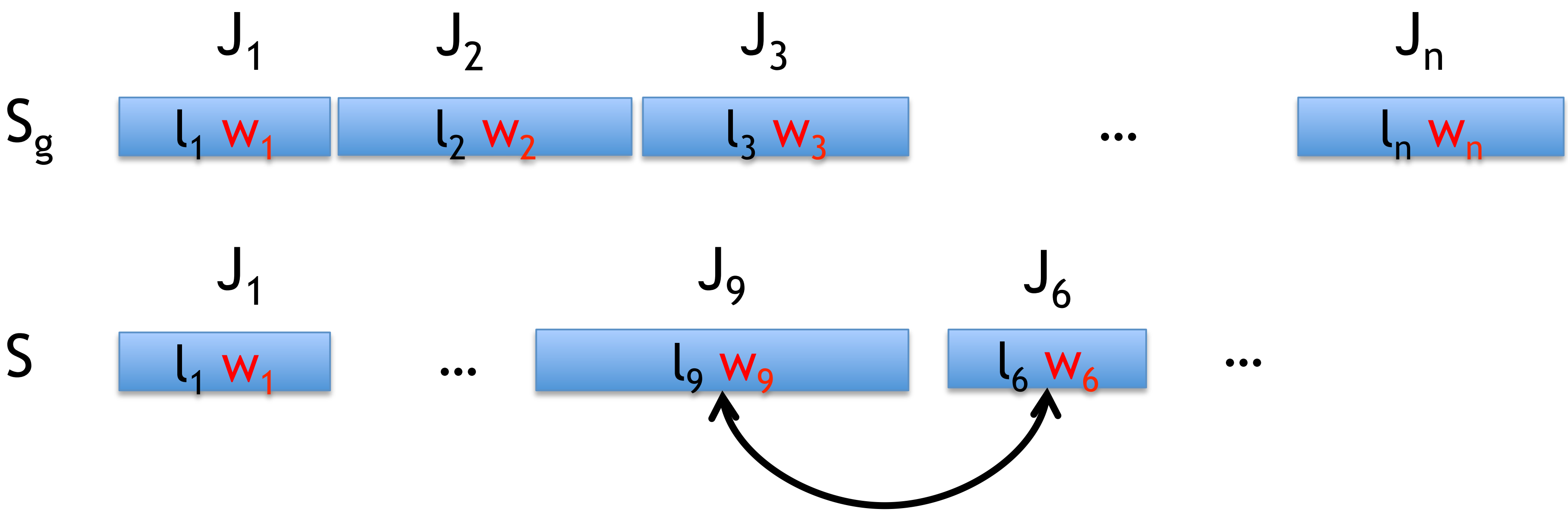
$$J_1, J_2, \ldots, J_n$$

i.e., J_1 happens to be the job with smallest l/w ratio

- Therefore $S_g = J_1::J_2::\ldots::J_n$

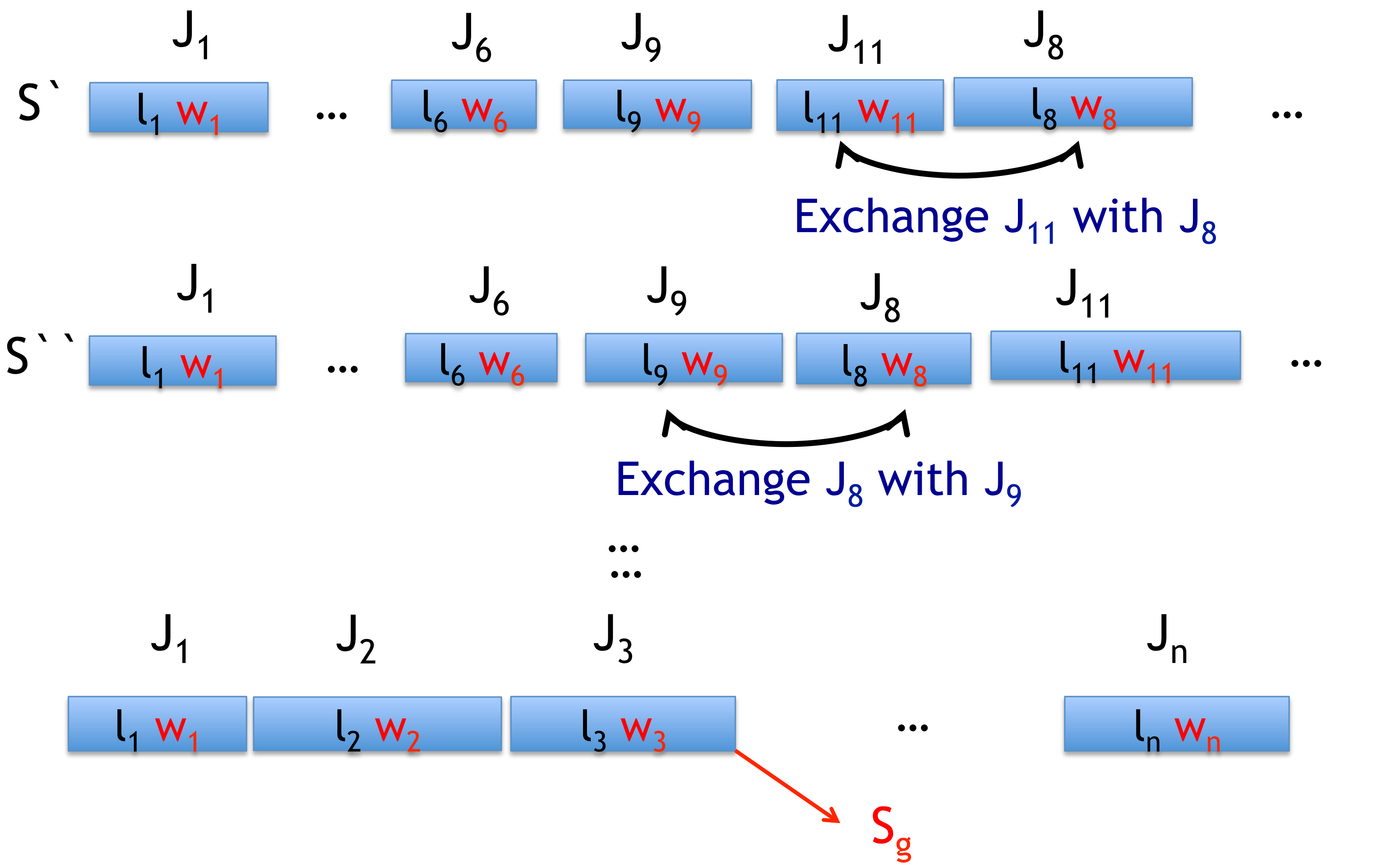
Proof of Correctness (2)

- Consider any other schedule $S \neq S_g$
- Claim: In $S=J_{s1}::J_{s2}::\ldots::J_{sn}$ there is a job k, right after a job i where k < i.



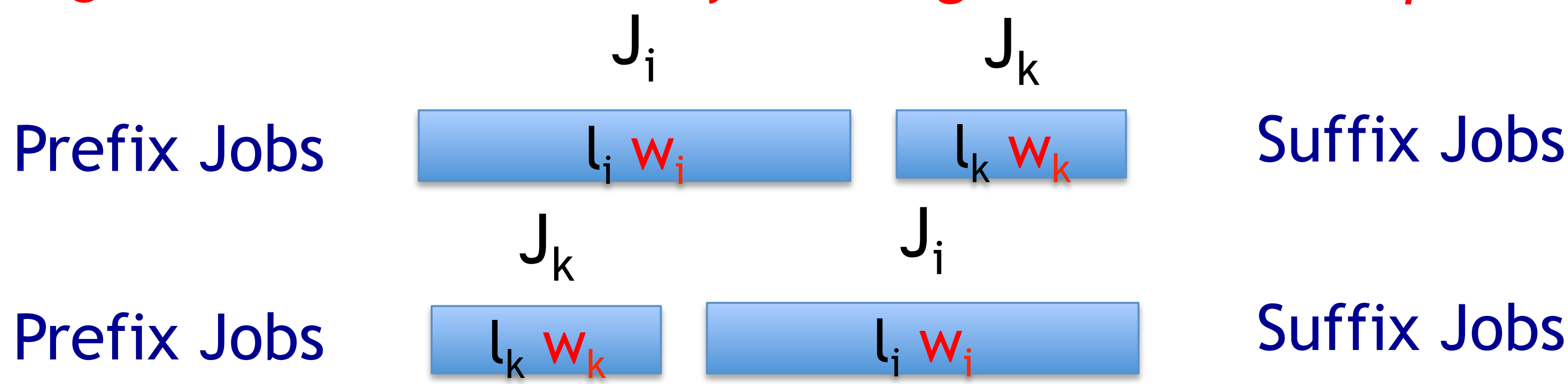
Exchange J_9 with J_6

Proof of Correctness (3)



Completing the Proof

- Recall Claim: In $S = J_{s1}::J_{s2}::\ldots::J_{sn}$ there is a job k, right after a job i where k < i.
- *Q: How does the cost of S change when we swap i and j?*



$$\sum_{i=1}^{n} w_i C_i$$

$w_i * l_k$

$w_k * l_i$

By renaming of jobs and k < i => $l_k/w_k \leq l_i/w_i$ => $w_i l_k \leq w_k l_i$

QED