Greedy Algorithm

Greedy Algorithm

Scheduling

	Start	End
CIS 675	2	3.25
CIS 412	1	4
CIS 411	3	4
CIS 310	3.5	4.75
CIS 320	4	5.25
CIS 121	4.5	6
CIS 660	5	6.5
CIS 230	7	8

Problem Statement

$(a_1,\ldots.a_n)$

$(s_1,s_2,\ldots,s_n)$

$(f_1,f_2,\ldots,f_n)$ (sorted) $s_i < f_i$

Find largest subset of activities $C=\{a_i\}$ such that

For any two $a_i,a_j \in C$, such that $i < j$

$$\mathbf{f_i \leq s_j}$$

Problem Statement

$(a_1, \ldots a_n)$

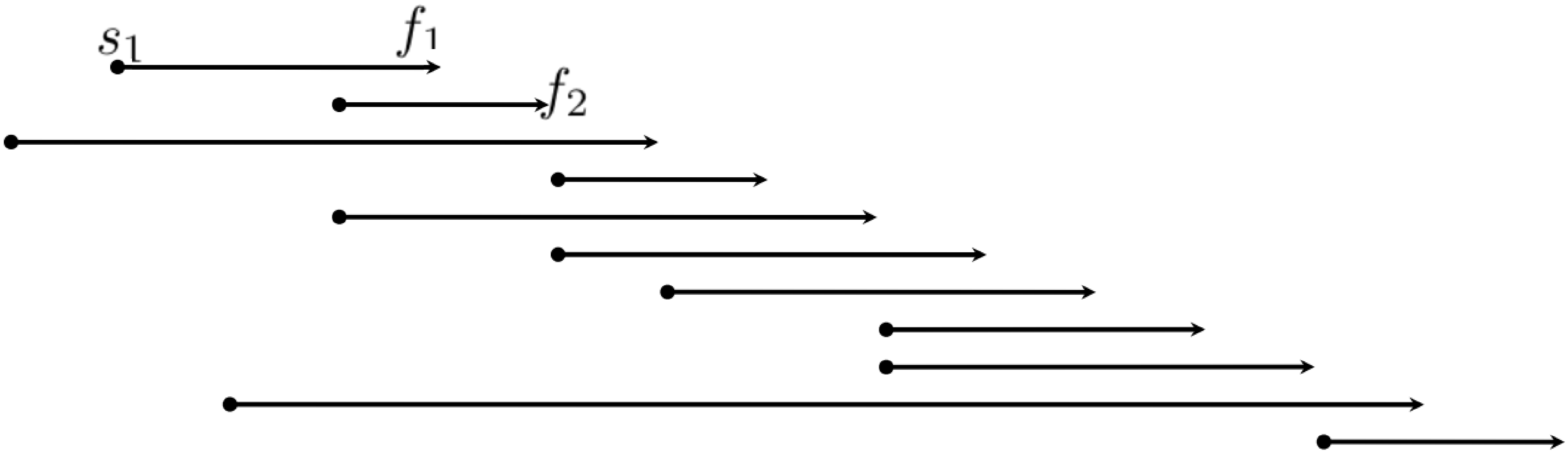
$(s_1, s_2, \ldots, s_n)$

$(f_1, f_2, \ldots, f_n)$ (sorted) $s_i < f_i$

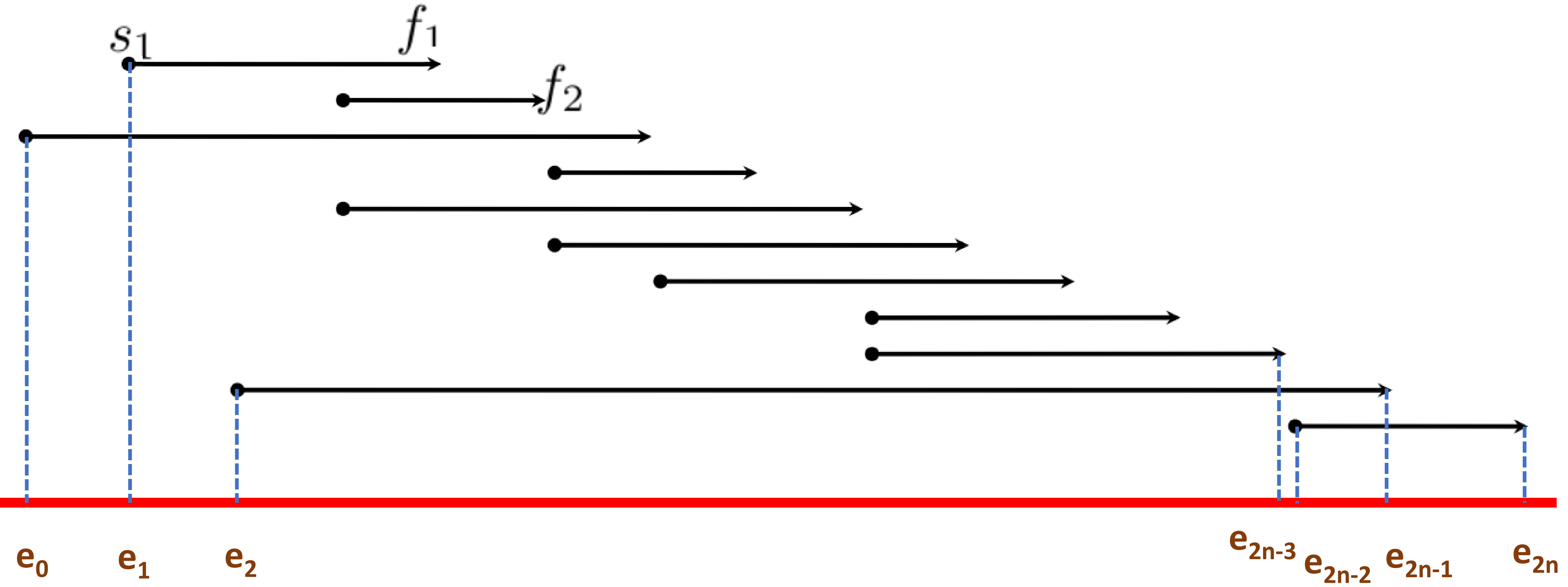
Find largest subset of activities $C = \{a_i\}$ such that:

For any two $a_i, a_j \in C$, such that $i < j$

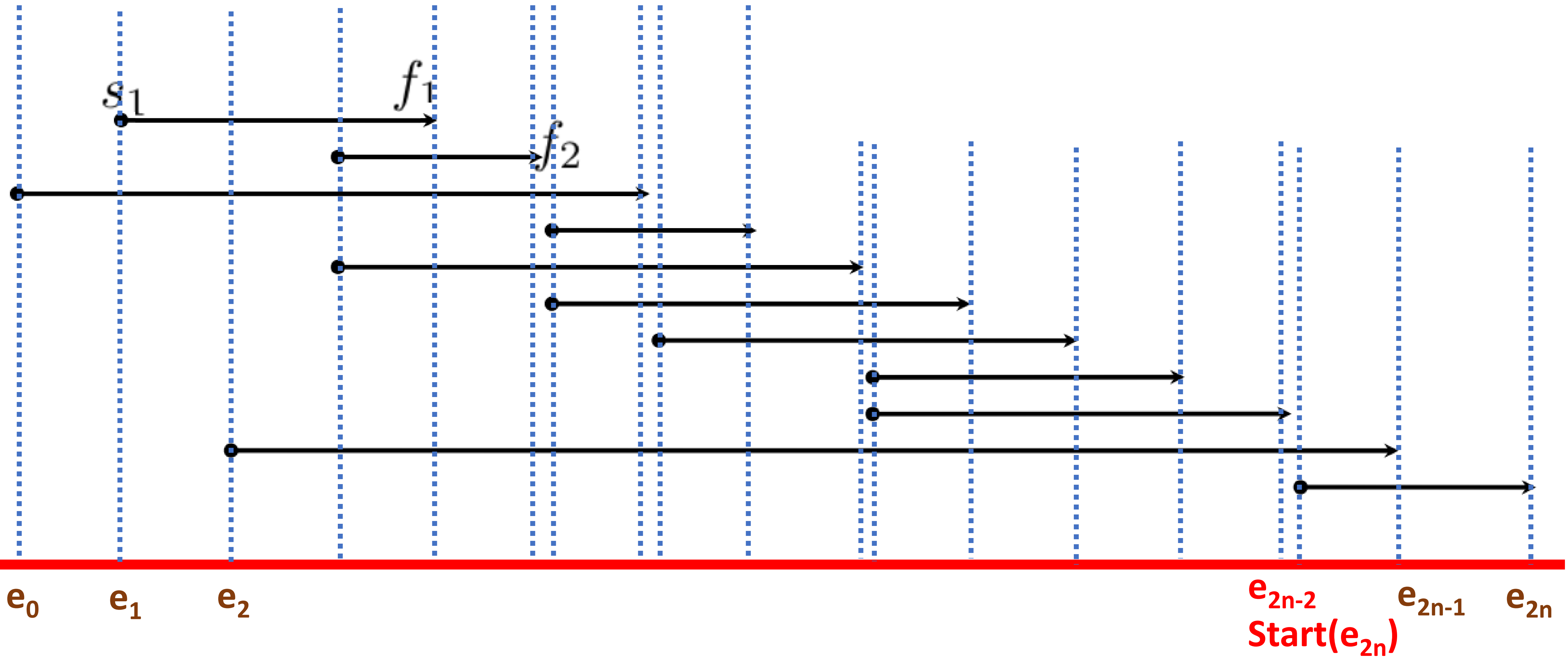
$$f_i \leq s_j$$



DP Solution

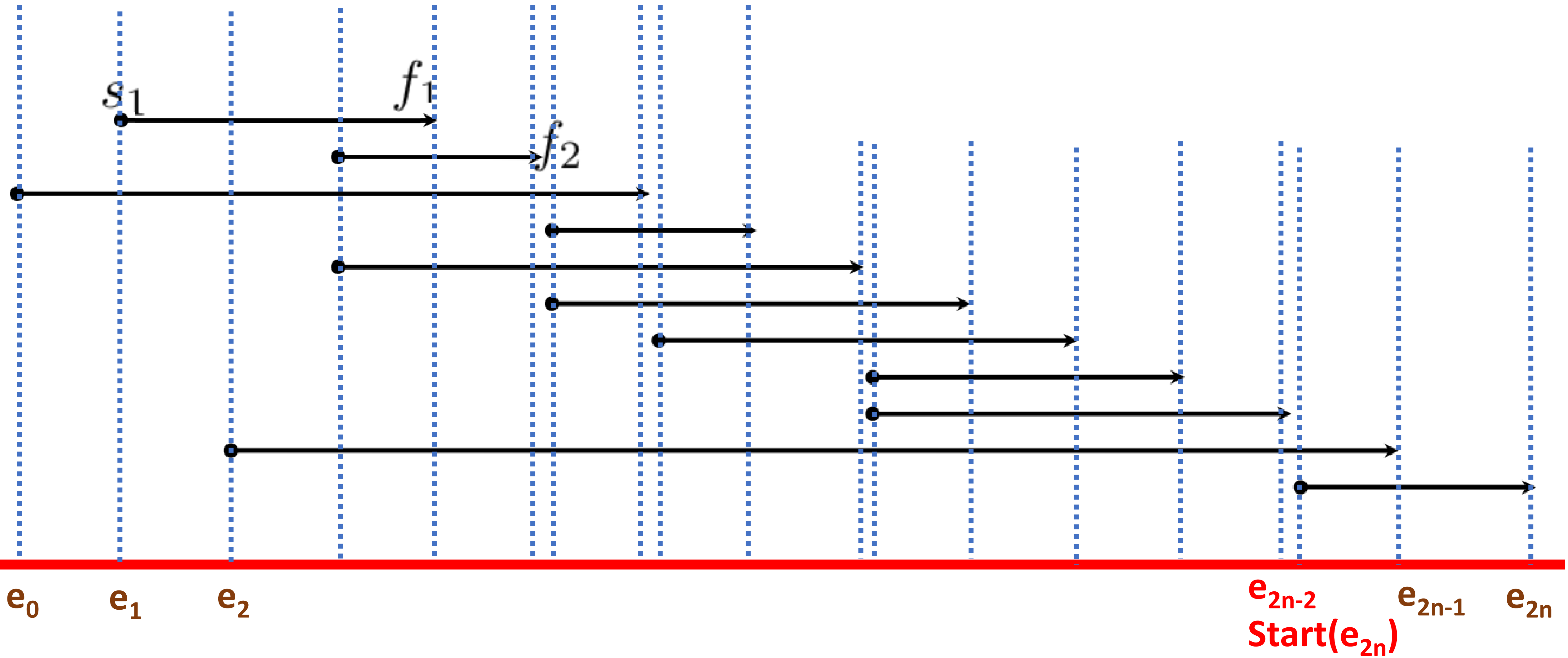


DP Solution



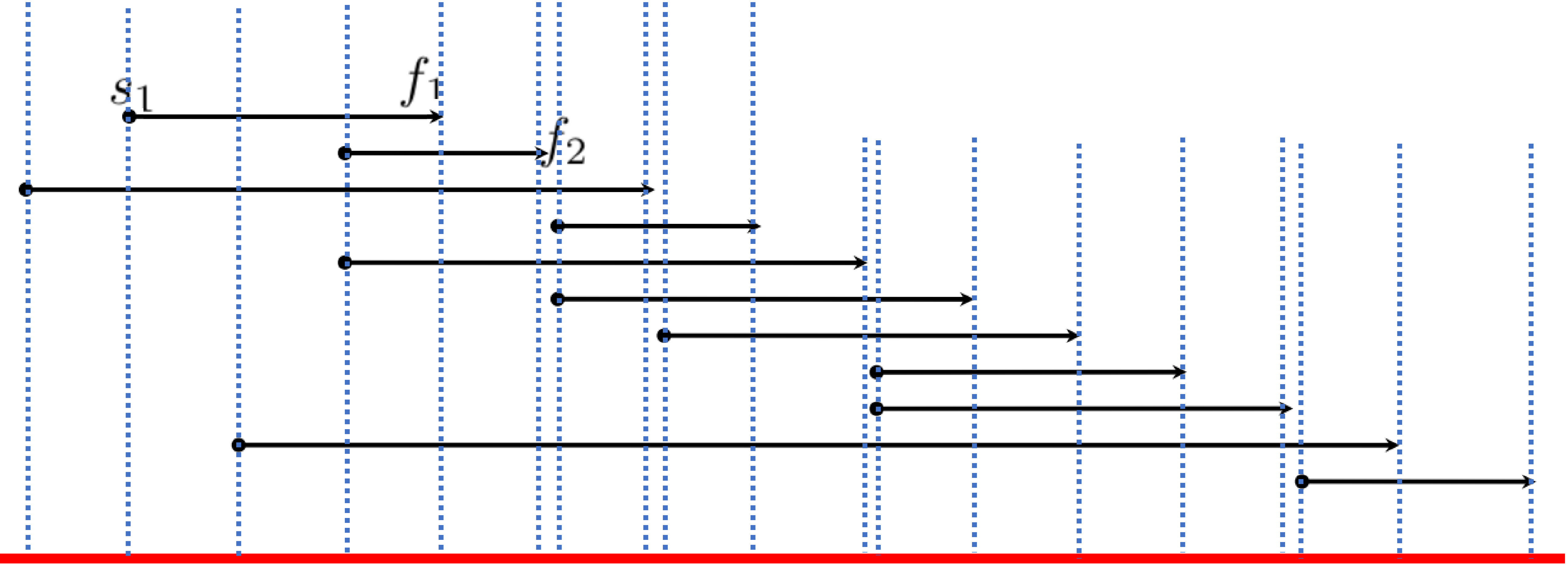
Best$_n$= maximal number of activities that can occur before event n

DP Solution



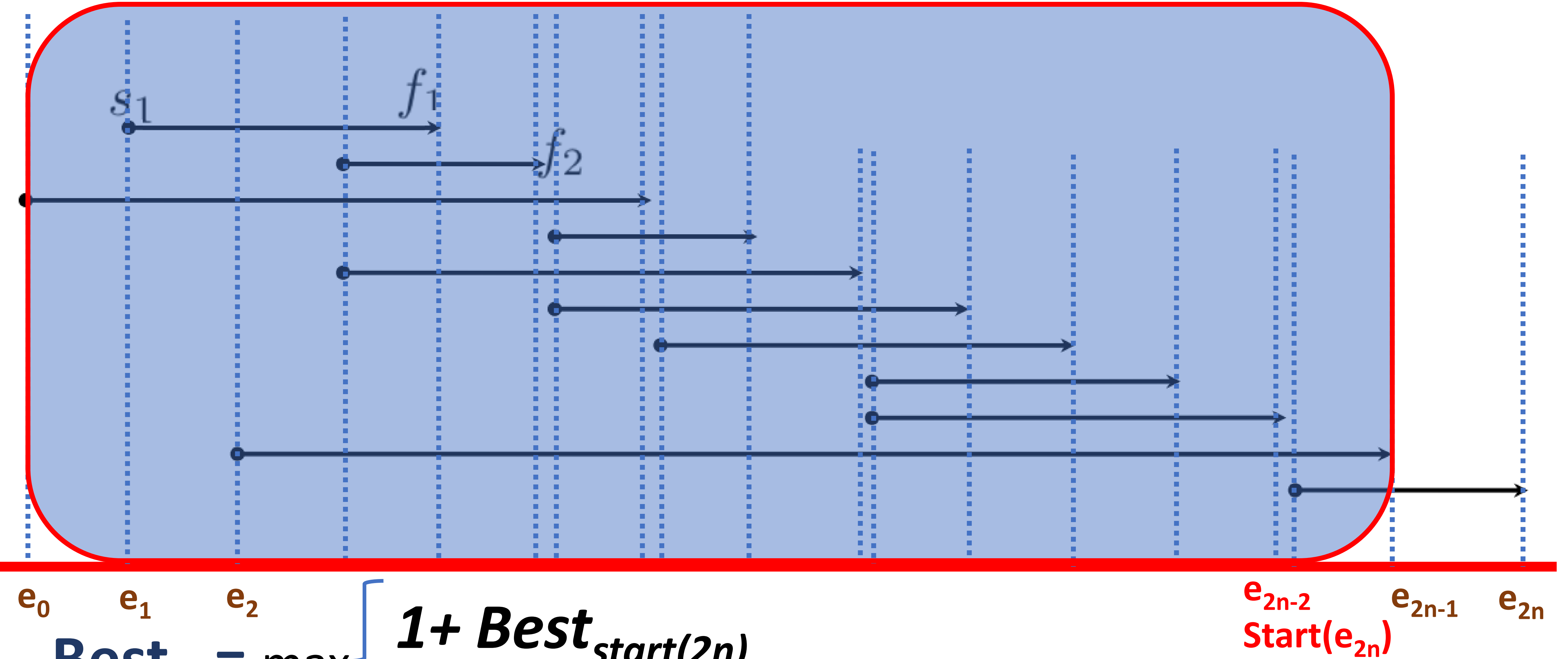
$Best_{2n}$= maximal number of activities that can occur before event 2n

DP Solution



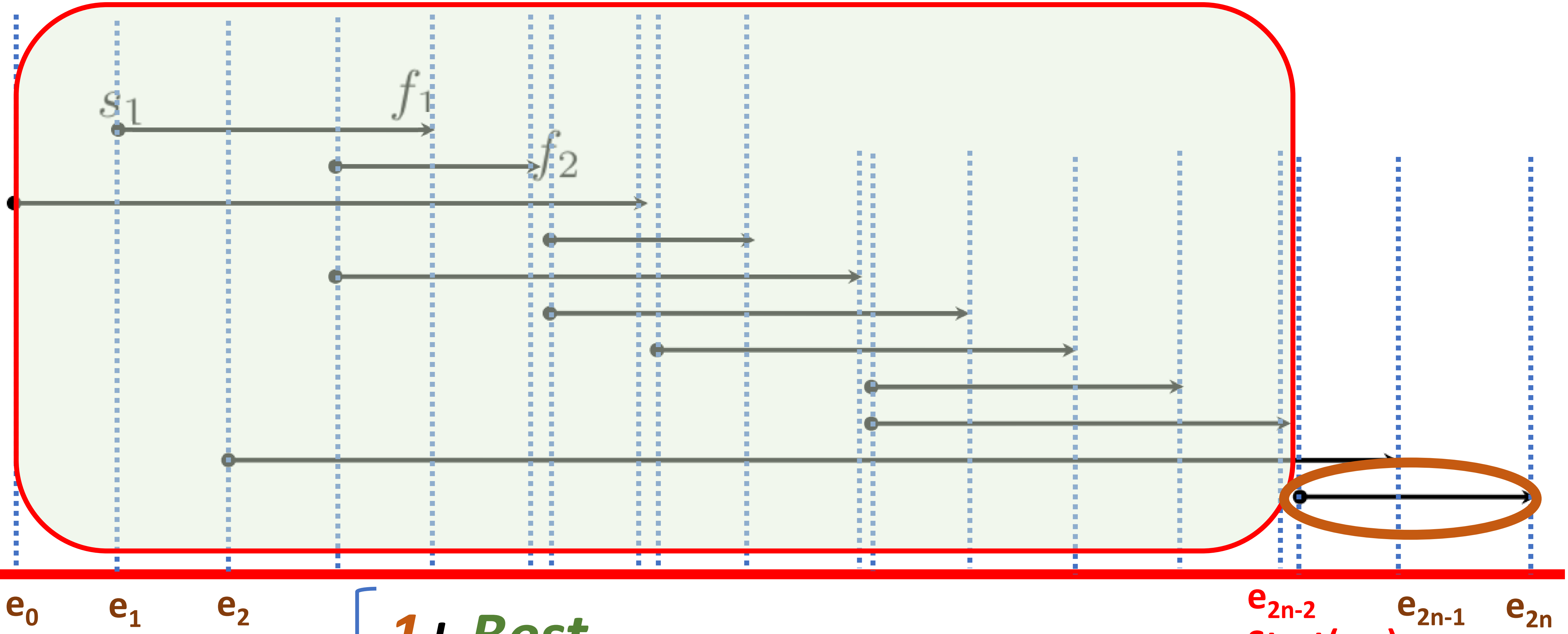
$$Best_{2n} = \max \begin{cases} 1 + Best_{start(2n)} \\ Best_{2n-1} \end{cases}$$

DP Solution



$$\text{Best}_{2n} = \max \begin{cases} 1 + \text{Best}_{start(2n)} \\ \text{Best}_{2n-1} \end{cases}$$

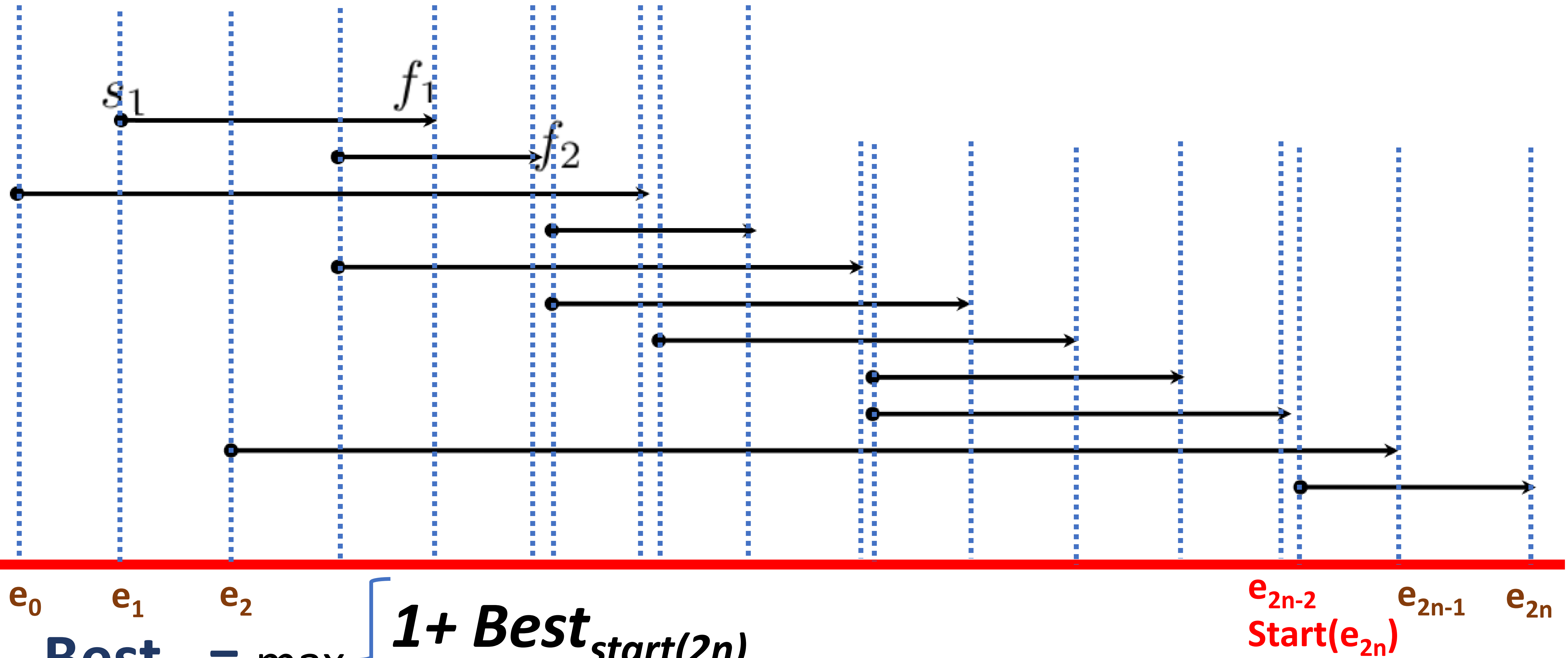
DP Solution



Start(e_{2n})

$$\mathbf{Best}_{2n} = \max \begin{cases} 1 + \mathit{Best}_{start(2n)} \\ \mathit{Best}_{2n-1} \end{cases}$$

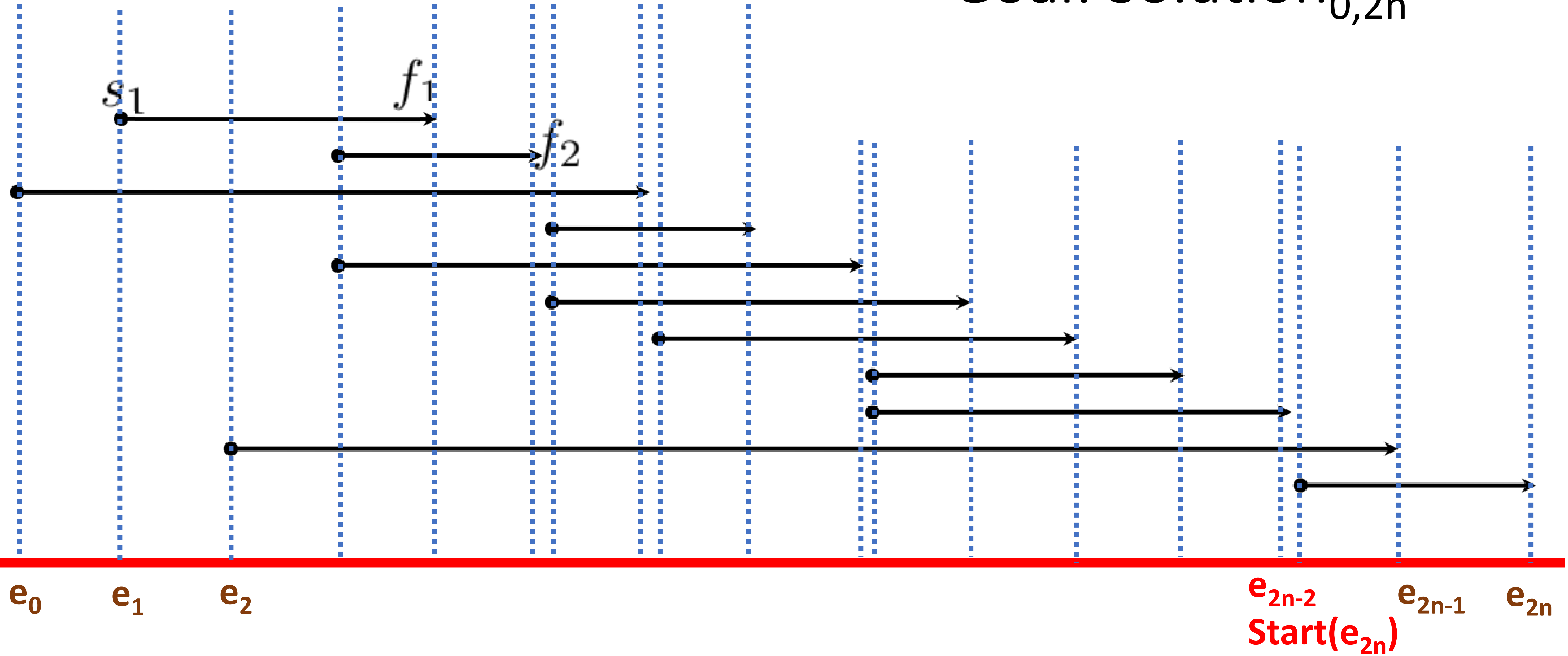
DP Solution



$$\mathbf{Best}_{2n} = \max \begin{cases} 1 + Best_{start(2n)} \\ Best_{2n-1} \end{cases}$$

Greedy Solution

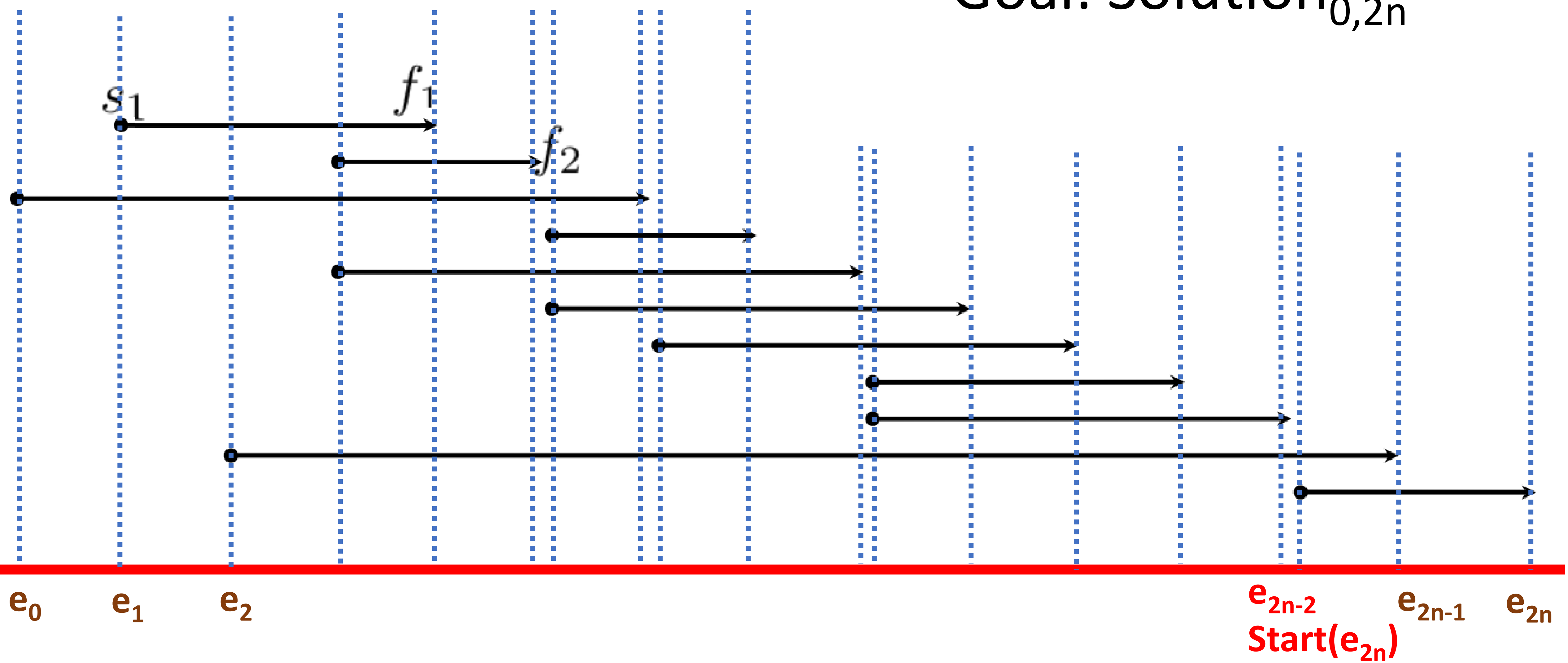
Goal: $\text{Solution}_{0,2n}$



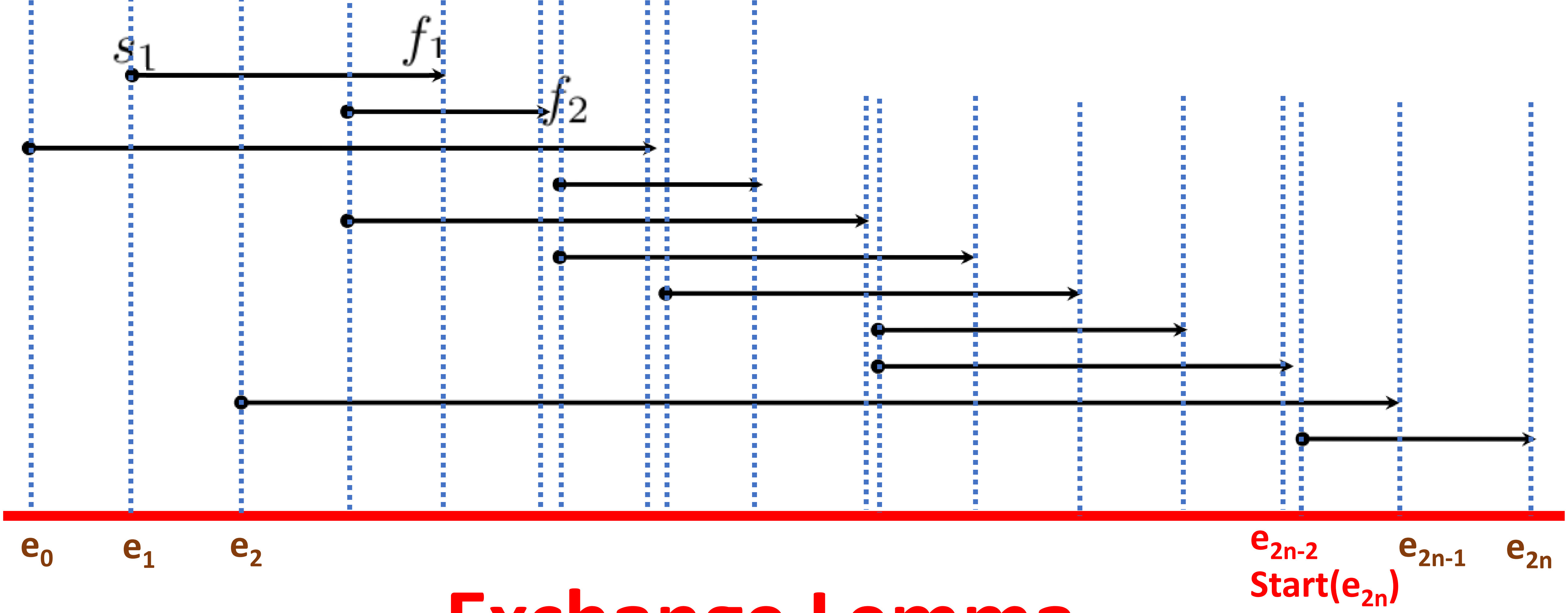
$\text{Solution}_{i,j}$ = maximum number of activities that occur between events i,j

Greedy Solution

Goal: $Solution_{0,2n}$



$Solution_{i,j}$ = maximum number of activities that occur between events i,j

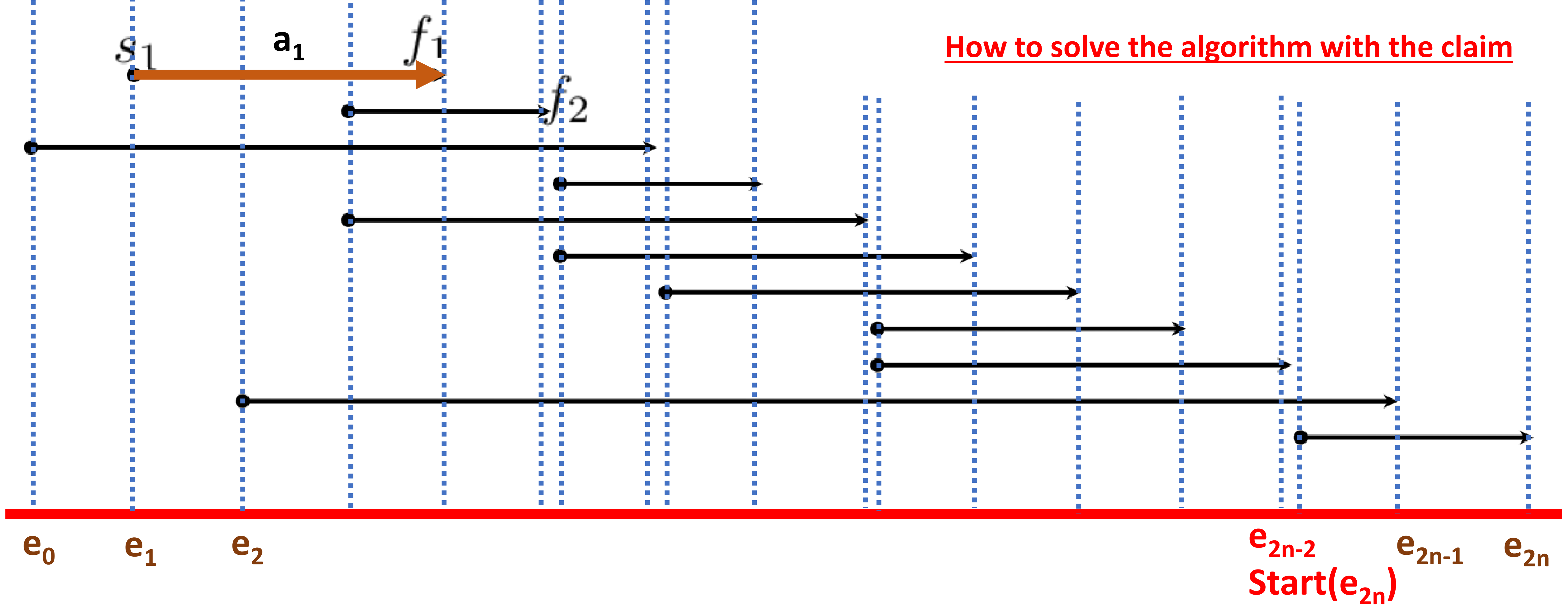


Exchange Lemma

Claim:

the first action to finish in e[i,j] is always part of some optimal Solution$_{i,j}$

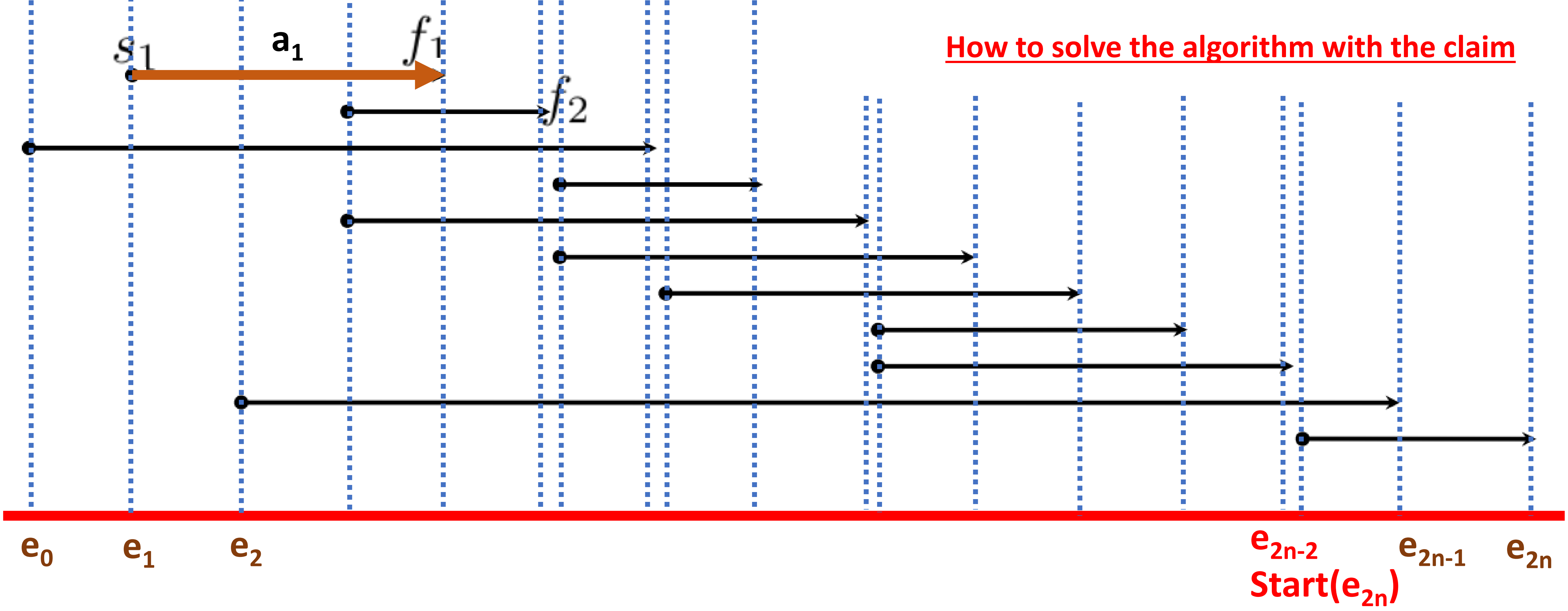
How to solve the algorithm with the claim



Claim:

the first action to finish in e[i,j] is always part of some optimal $\text{Solution}_{i,j}$

a_1 : will always be part of at least one optimal solution



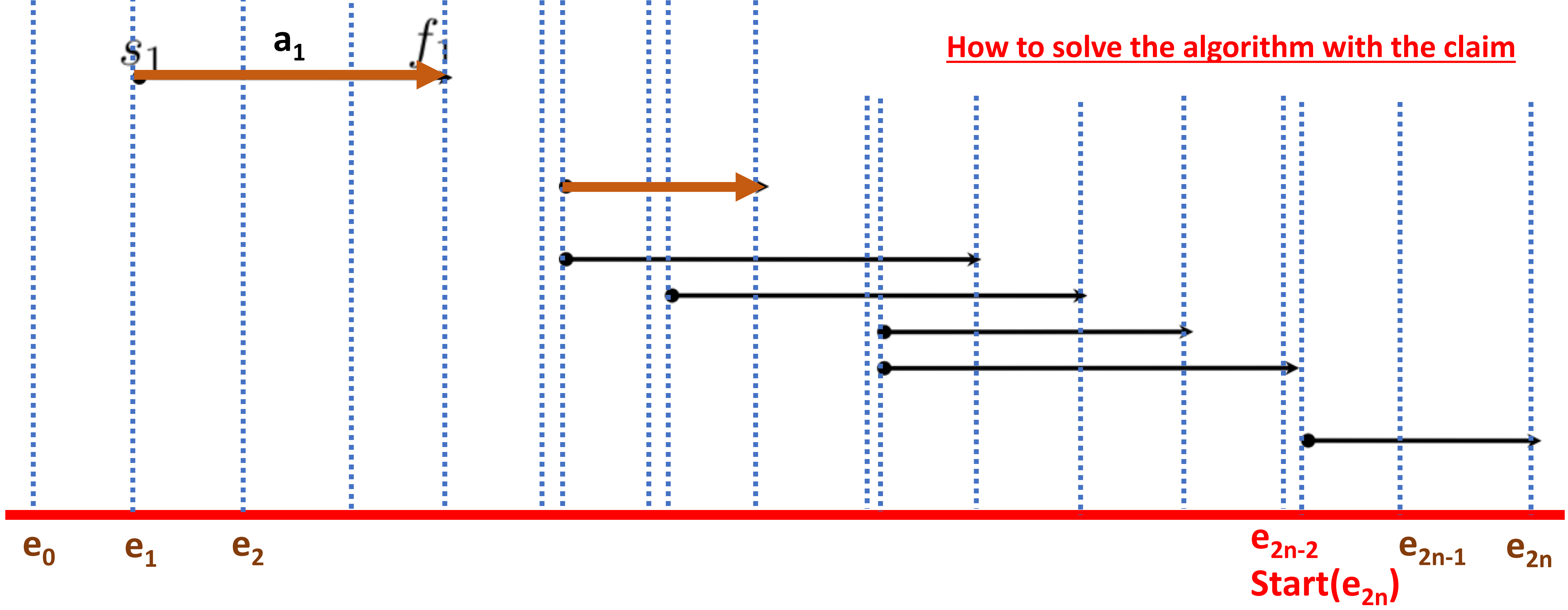
How to solve the algorithm with the claim

Algorithm: find first event to finish. add to solution. remove conflicting events. continue.

Claim:

the first action to finish in e[i,j] is always part of some optimal Solution$_{i,j}$

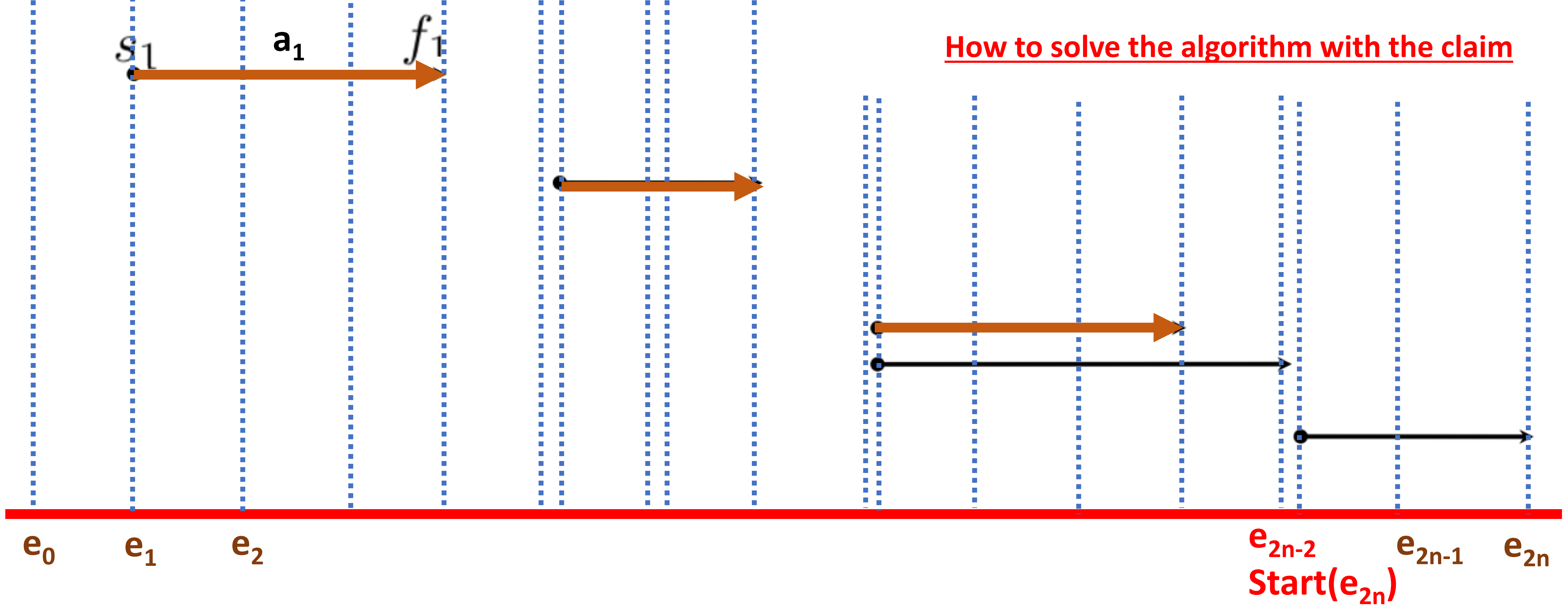
How to solve the algorithm with the claim



Algorithm: find first event to finish. add to solution. remove conflicting events. continue.

Claim:

the first action to finish in e[i,j] is always part of some optimal $\text{Solution}_{i,j}$

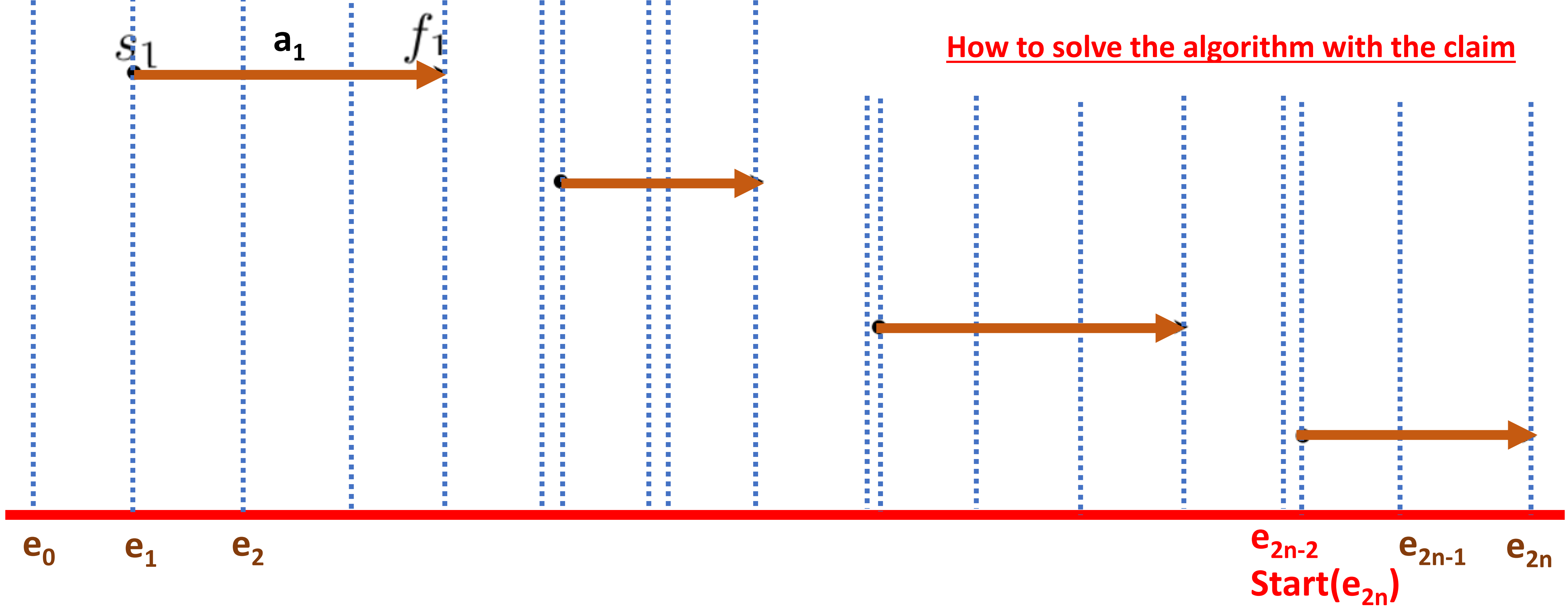


Algorithm: find first event to finish. add to solution. remove conflicting events. continue.

Claim:

the first action to finish in e[i,j] is always part of some optimal $\text{Solution}_{i,j}$

How to solve the algorithm with the claim

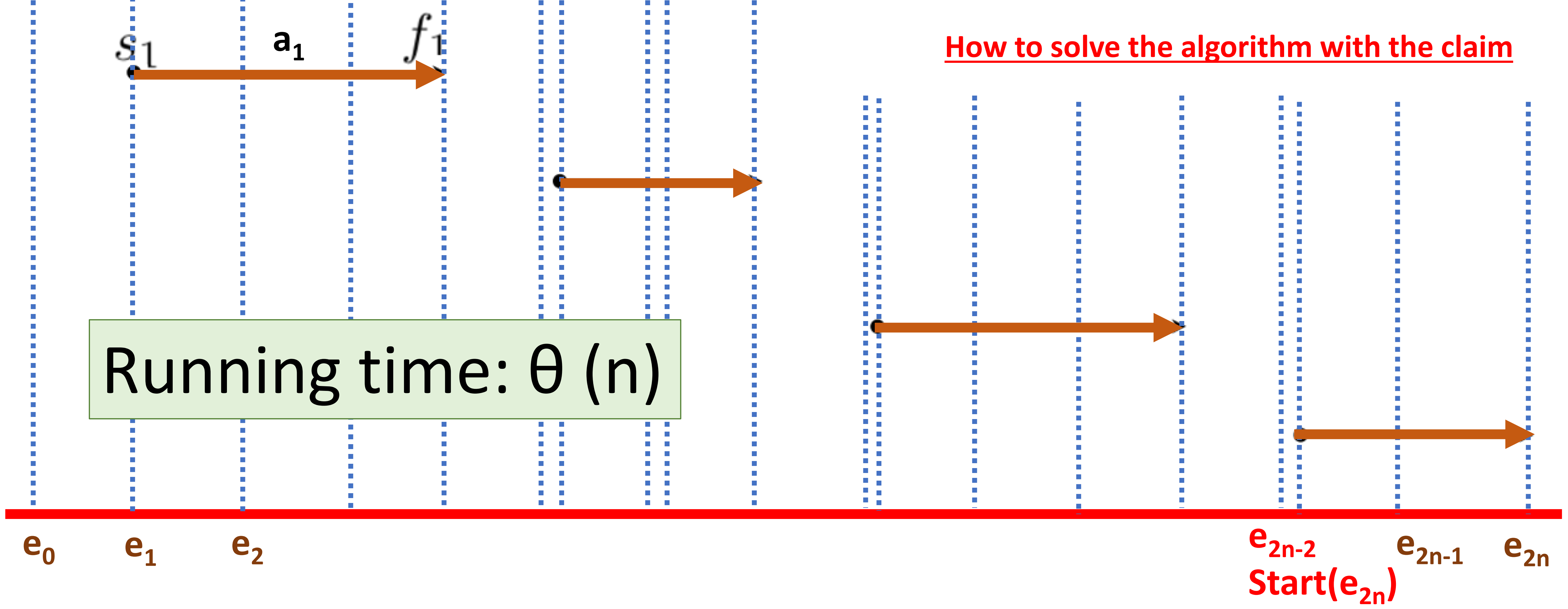


Algorithm: find first event to finish. add to solution. remove conflicting events. continue.

Claim:

the first action to finish in e[i,j] is always part of some optimal Solution$_{i,j}$

How to solve the algorithm with the claim



Algorithm: find first event to finish. add to solution. remove conflicting events. continue.

Claim:

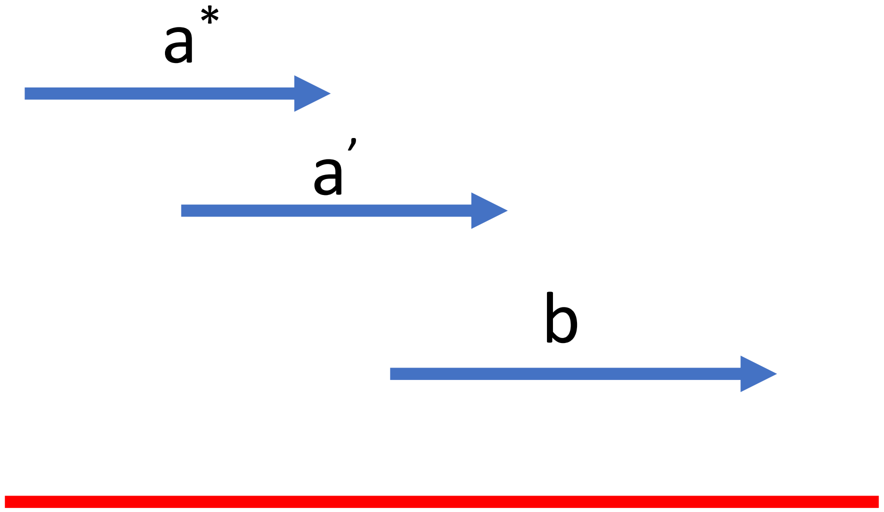
the first action to finish in e[i,j] is always part of some optimal $\text{Solution}_{i,j}$

Claim: the first action to finish in e[i,j] is always part of some optimal $\text{Solution}_{i,j}$

Proof: Consider $\text{SOLUTION}_{i,j}$ and let A* be the first to activity to finish in [i,j]

1. If a* ϵ $\text{SOLUTION}_{i,j}$ then the claim follows.

2. Suppose that a* is not in $\text{SOLUTION}_{i,j}$. Let activity a’ be the first activity to finish the solution.

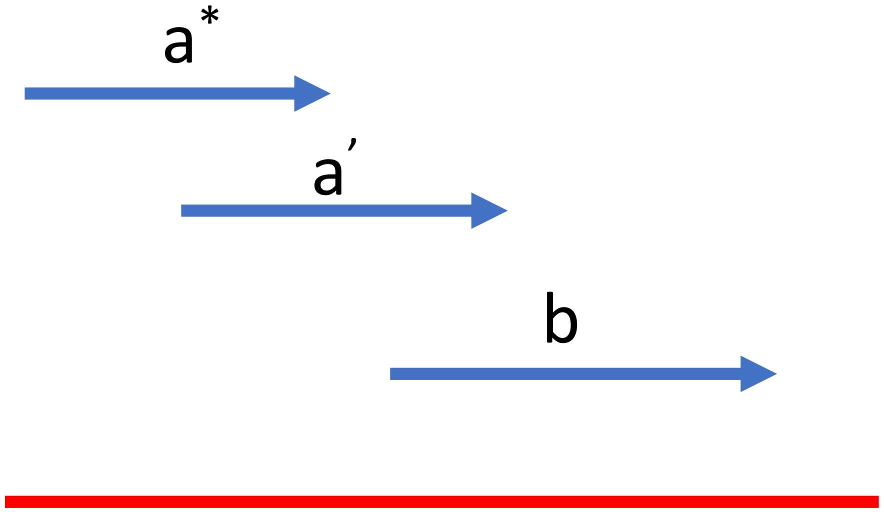


Claim: the first action to finish in e[i,j] is always part of some optimal $\text{Solution}_{i,j}$

Proof: Consider $\text{SOLUTION}_{i,j}$ and let a* be the first to activity to finish in [i,j]

1. If $a^* \in \text{SOLUTION}_{i,j}$ then the claim follows.

2. Suppose that a* is not in $\text{SOLUTION}_{i,j}$. Let activity a' be the first activity to finish the solution.



a* finishes before a', $f_{a*} < f_{a'}$

So therefore,
$S = \text{SOLUTION}_{i,j} - \{a'\} + \{a^*\}$

Then, $S = \text{SOLUTION}_{i,j}$

So, S is optimal too. Therefore lemma follows.

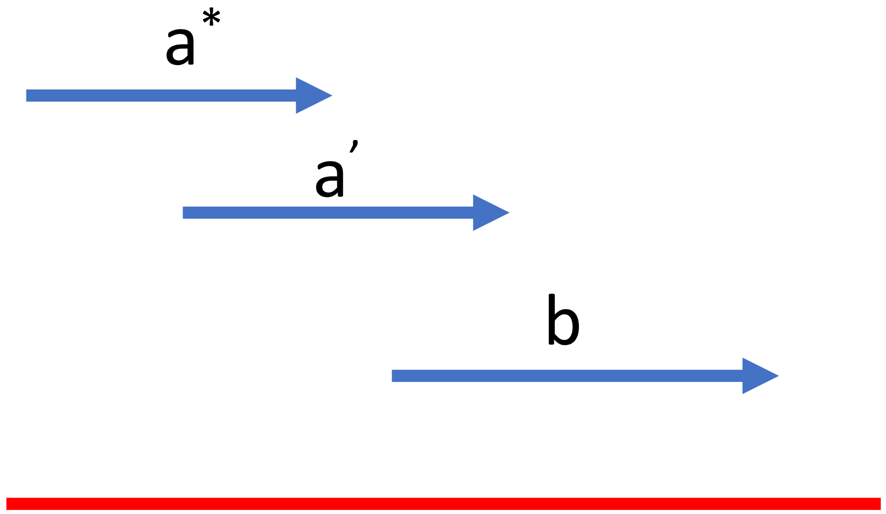
Claim: the first action to finish in e[i,j] is always part of some optimal Solution$_{i,j}$

Proof: Consider SOLUTION$_{i,j}$ and let a* be the first to activity to finish in [i,j]

1. If a* ϵ SOLUTION$_{i,j}$ then the claim follows.

2. Suppose that a* is not in SOLUTION$_{i,j}$. Let activity a’ be the first activity to finish the solution.

a* finishes before a’, $f_{a*} < f_{a'}$



So therefore,
S = SOLUTION$_{i,j}$ – {a’} + {a*}

Then, S = SOLUTION$_{i,j}$

So, S is optimal too. Therefore lemma follows.

Caching

Cache

Main Memory

Virtual Memory

CPU

load r2, addr a

store r4, addr b

Question of problem:

1. How can we manage a cache in order to minimize number of cache misses
2. Simplify the assumption that we know all memory accesses beforehand
3. Cache is fully associative, meaning, you know which data is in which cache address in $\theta(1)$ time.

problem statement

input: K, the size of the cache
$d_1, d_2, \ldots, d_m$ memory accesses

output: schedule for that cache that minimizes # of cache misses while satisfying requests

cache is fully associative, line size is 1

Belady evict rule

Evict the item from the cache that is accessed farthest in the future.

Example

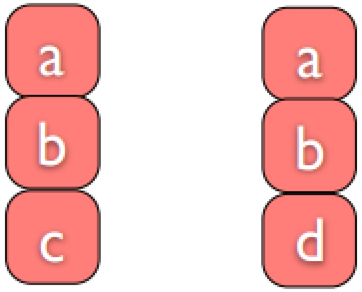
cache

a
b
c

a b c d a d e a d b a e c e a

Example

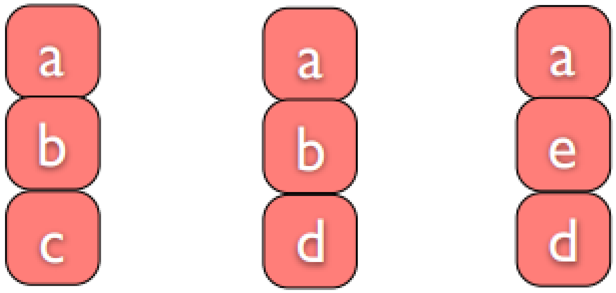
cache



a b c d a d e a d b a e c e a

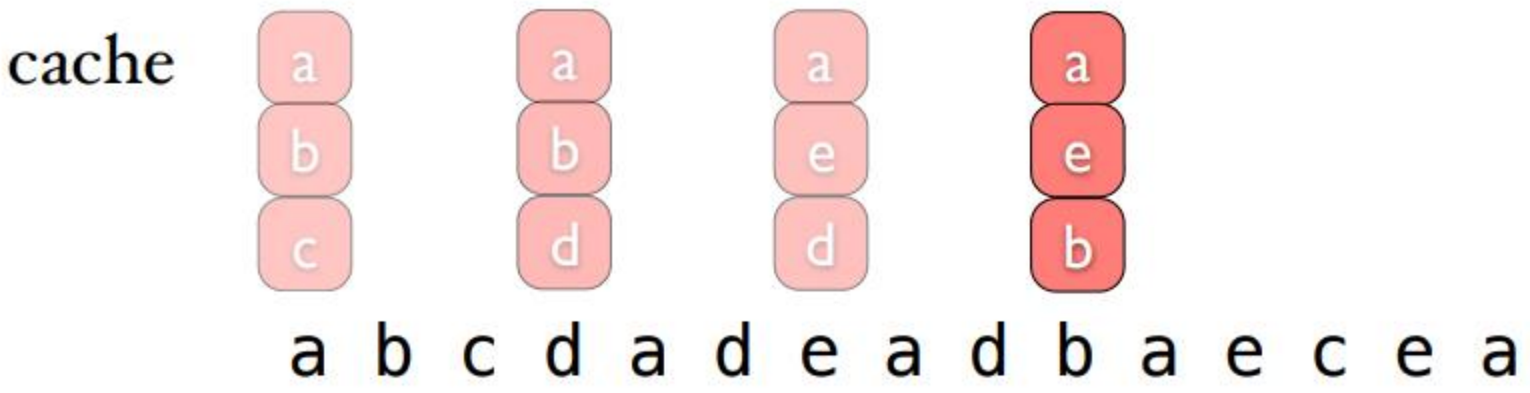
Example

cache

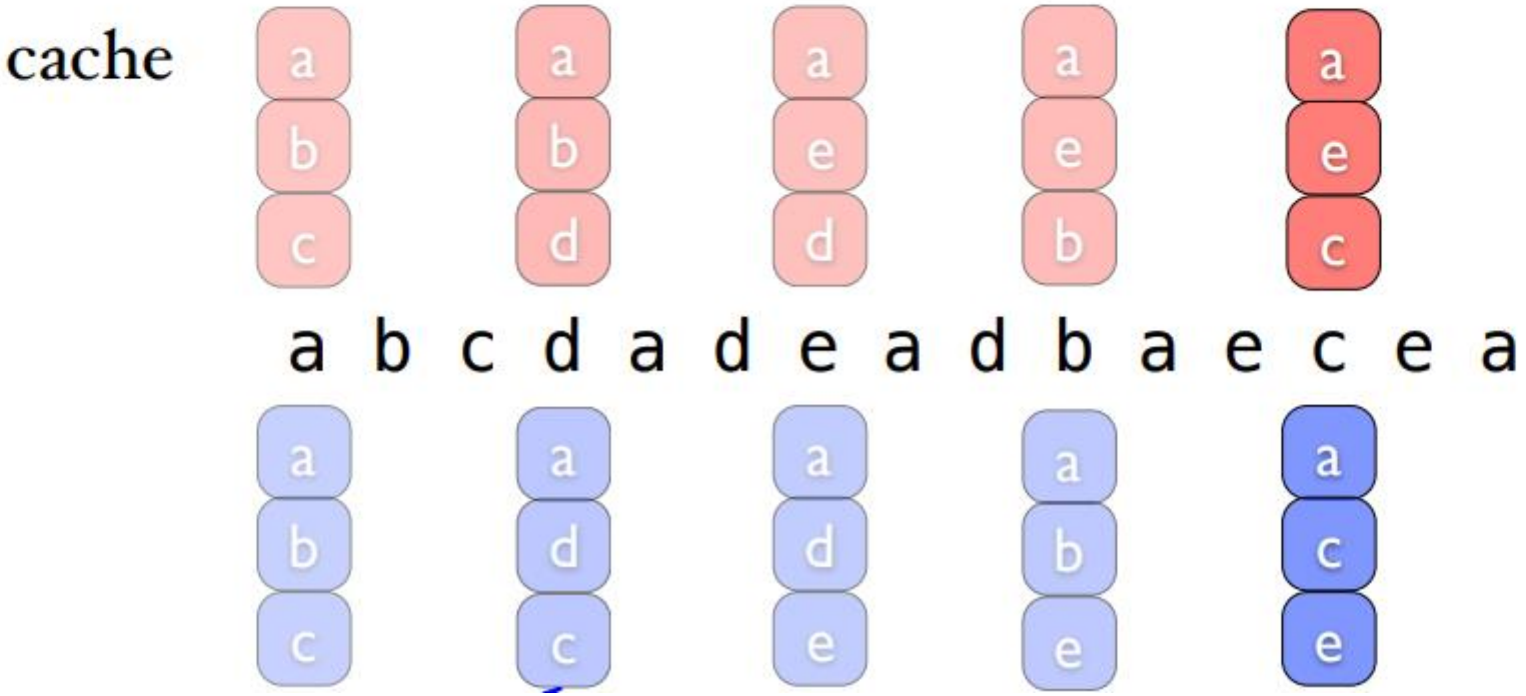


a b c d a d e a d b a e c e a

Example



Example



cache
a
b
c
a
b
d
a
e
d
a
e
b
a
e
c
a b c d a d e a d b a e c e a
a
b
c
a
d
c
a
d
e
a
b
e
a
c
e

Belady evict rule

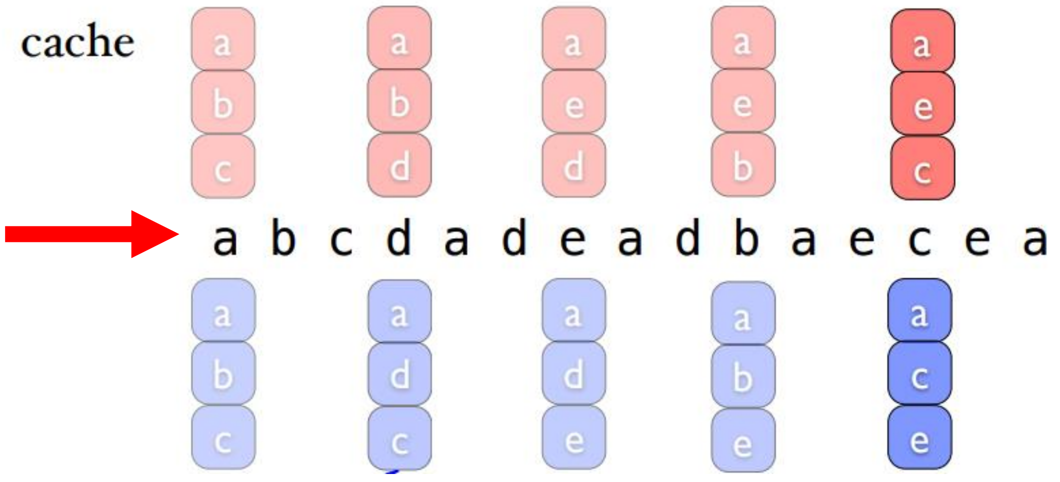
Evict the item from the cache that is accessed $\underbrace{\text{farthest in the future.}}_{S_{ff}}$

Gives optimal solution

How to proof it?

Which lemma we would need?

Some definition



Schedule for access pattern $d_1, d_2, \ldots\ldots d_n$:

Either the 'nop' operation or 'evict x for y' operation — **Each step**

**** at step i, element d_i must be in the cache

Reduce Lazy Schedule:

Schedule for which of the operation 'evict x for y'

Only occurs at a step i, if $y=d_i$

Exchange Lemma:

Let S be a reduced schedule that agrees with S_{ff} on the first j items. There exists a reduced schedule S' that agrees with S_{ff} on the first j+1 items and has the same or fewer #misses as S.

Number of misses (S') ≤ Number of misses (S)

Exchange Lemma:

Let S be a reduced schedule that agrees with S_{ff} on the first j items. There exists a reduced schedule S' that agrees with S_{ff} on the first j+1 items and has the same or fewer #misses as S.

Number of misses (S') ≤ Number of misses (S)

Optimal schedule

S_0^*

S_{ff}

S_0^* agrees with S_{ff} on first i=0

Exchange Lemma:

Let S be a reduced schedule that agrees with S_{ff} on the first j items. There exists a reduced schedule S' that agrees with S_{ff} on the first j+1 items and has the same or fewer #misses as S.

Number of misses (S') ≤ Number of misses (S)

Optimal schedule

S_0^* S_1 S_{ff}

S_1 agrees with S_{ff} on first i=1

S_0^* agrees with S_{ff} on first i=0

 # misses (S_1) ≤ # misses (S_1^*)

Exchange Lemma:

Let S be a reduced schedule that agrees with S_{ff} on the first j items. There exists a reduced schedule S' that agrees with S_{ff} on the first j+1 items and has the same or fewer #misses as S.

Number of misses (S') ≤ Number of misses (S)

Optimal schedule

S_0^* S_1 S_2 S_{ff}

S_2 agrees with S_{ff} on first i=2

S_1 agrees with S_{ff} on first i=1

S_0^* agrees with S_{ff} on first i=0

 # misses (S_2) ≤ # misses (S_2^*)

Exchange Lemma:

Let S be a reduced schedule that agrees with S_{ff} on the first j items. There exists a reduced schedule S' that agrees with S_{ff} on the first j+1 items and has the same or fewer #misses as S.

Number of misses (S') ≤ Number of misses (S)

Optimal schedule

$S_0{}^*$ S_1 S_2 S_3 S_{ff-1} S_{ff}

S_2 agrees with S_{ff} on first i=2

S_1 agrees with S_{ff} on first i=1

$S_0{}^*$ agrees with S_{ff} on first i=0

 # misses (S_2) ≤ # misses ($S_2{}^*$)

Exchange Lemma: **Proof**

Let S be a reduced schedule that agrees with S_{ff} on the first j items. There exists a reduced schedule S' that agrees with S_{ff} on the first j+1 items and has the same or fewer #misses as S.

Proof: Since **S** agrees with $\mathbf{S_{ff}}$ on the first **j** operations, then the state of the cache at operation ***j+1*** will be the same. Let d Be the addresses accessed at the operation ***j+1***.

State of the cache after j operations under the two schedules

S

$\mathbf{S_{ff}}$

Proof of lemma

State of the cache after j operations under the two schedules

S

S_{ff}

Easy Case 1: **d** $\in$ cache. **Then S’ = S Because both S and S_{ff} issue ‘nop’ operation**

Easy Case 1: **d** $\notin$ cache. **But, both S, S’, S_{ff} ‘evict e for d’ or ‘evict f for d’**

Proof of lemma

State of the cache after j operations under the two schedules

Easy Case 1: **d** $\in$ cache. **Then S’ = S Because both S and S_{ff} issue ‘nop’ operation**

Easy Case 1: **d** $\notin$ cache. **But, both S, S’, S_{ff} ‘evict e for d’ or ‘evict f for d’**

Exchange Lemma:

Let S be a reduced schedule that agrees with S_{ff} on the first j items. There exists a reduced schedule S’ that agrees with S_{ff} on the first j+1 items and has the same or fewer #misses as S.

Proof of lemma

State of the cache after j operations under the two schedules

S ‘evict e for d’

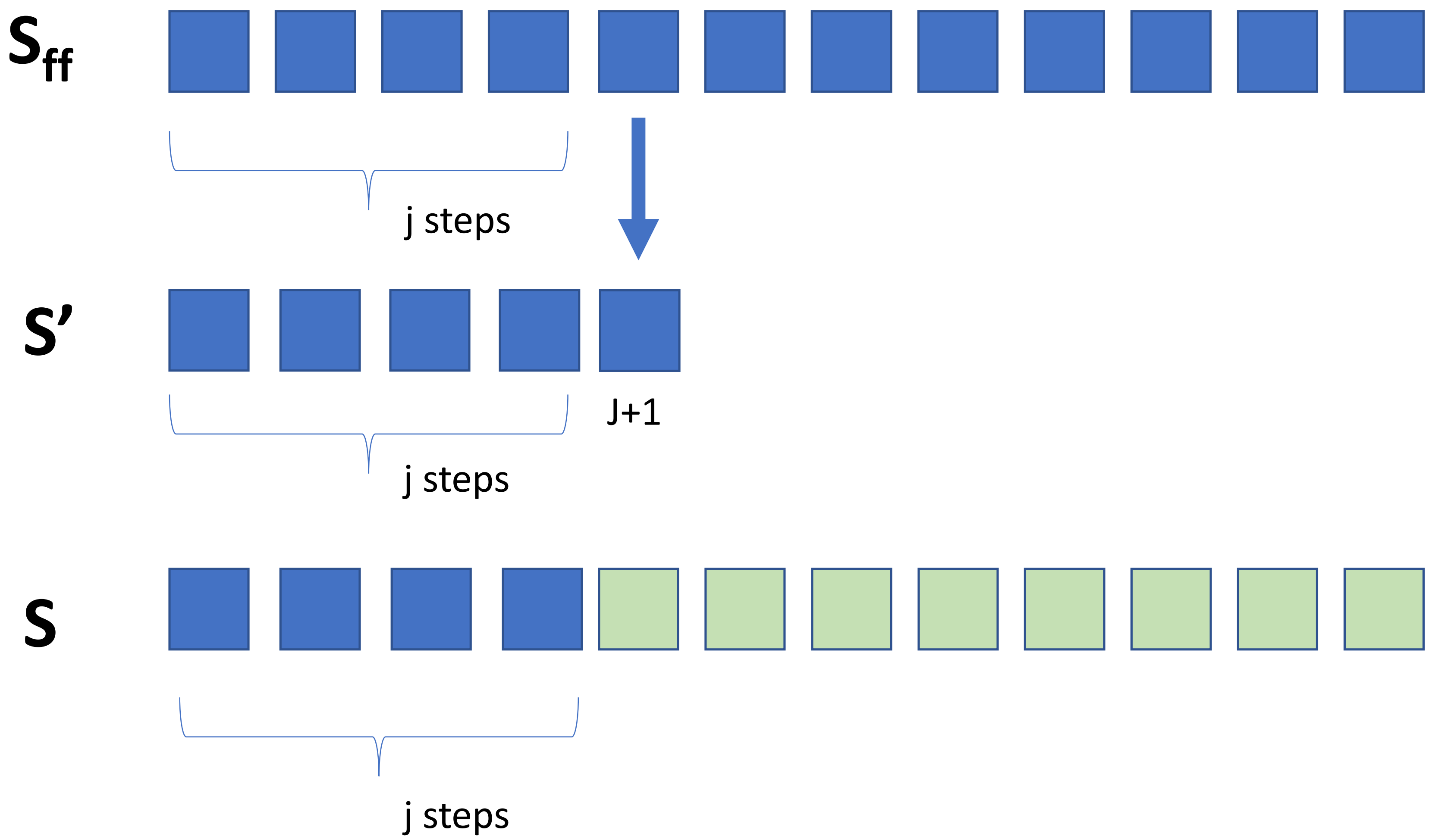
S_{ff} ‘evict f for d’

Case 3: d $\notin$ cache.

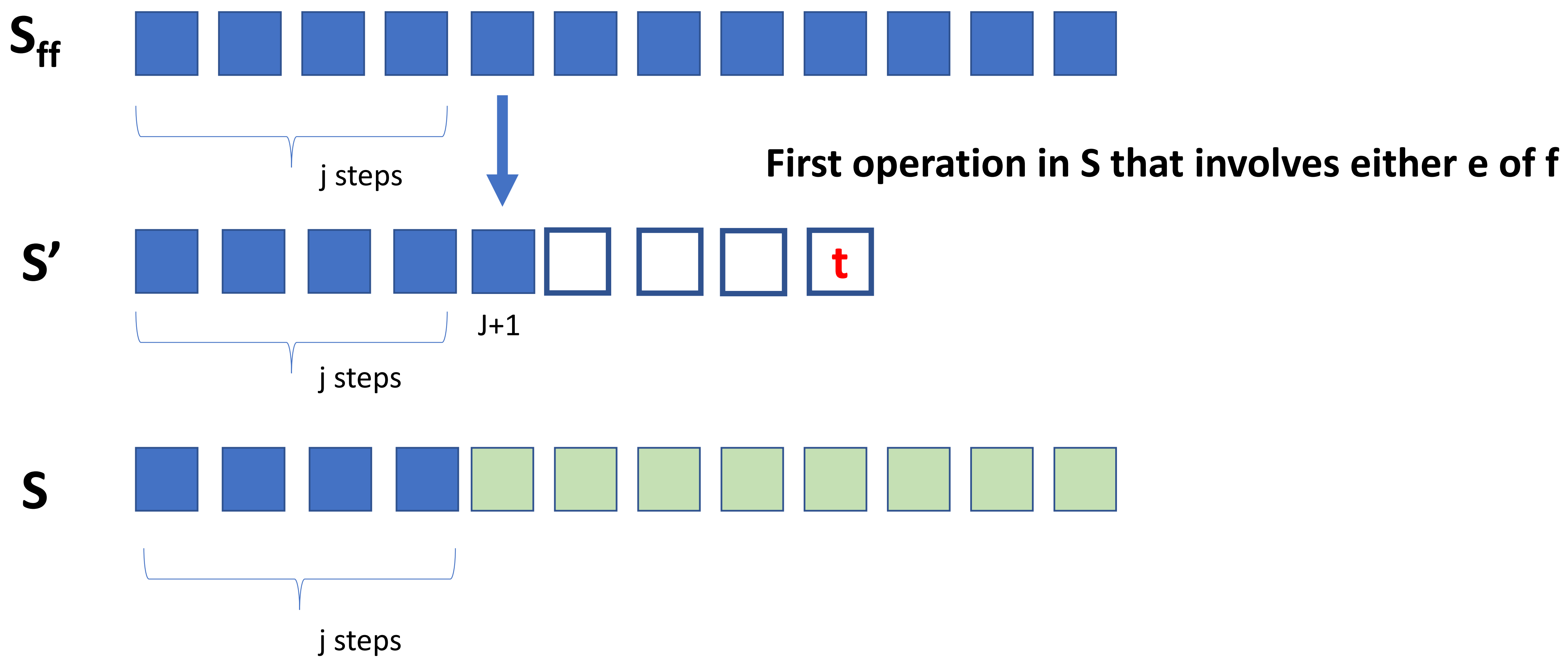
We need to construct a schedule S’ that satisfy:

1. agrees with Sff on j+1 step

2. has same number of misses as S, up to j+1 step.



S_{ff}
j steps
S'
J+1
j steps
S
j steps



S_{ff}
j steps
First operation in S that involves either e of f
S'
t
J+1
j steps
S
j steps

Let access t be the first operation in S after j+1 that involves either e or f

Either, t=e

t=f

t is something else

Let access t be the first operation in S after j+1 that involves either e or f

Either, t=e

t=f

t is something else

Let access t be the first operation in S after j+1 that involves either e or f

Either, t=e

S must load e. ‘Evict x for e’

S’ can issue the operation:
‘Evict x for f’

As a result S’ and S will have the same state of
The cache and same # of misses.

Let access t be the first operation in S after j+1 that involves either e or f

Either, t=f

Can not happen!!!! Why?

S_{ff} uses the farthest in the future rule. So, e would have been evicted in S_{ff}, not f.

Let access t be the first operation in S after j+1 that involves either e or f

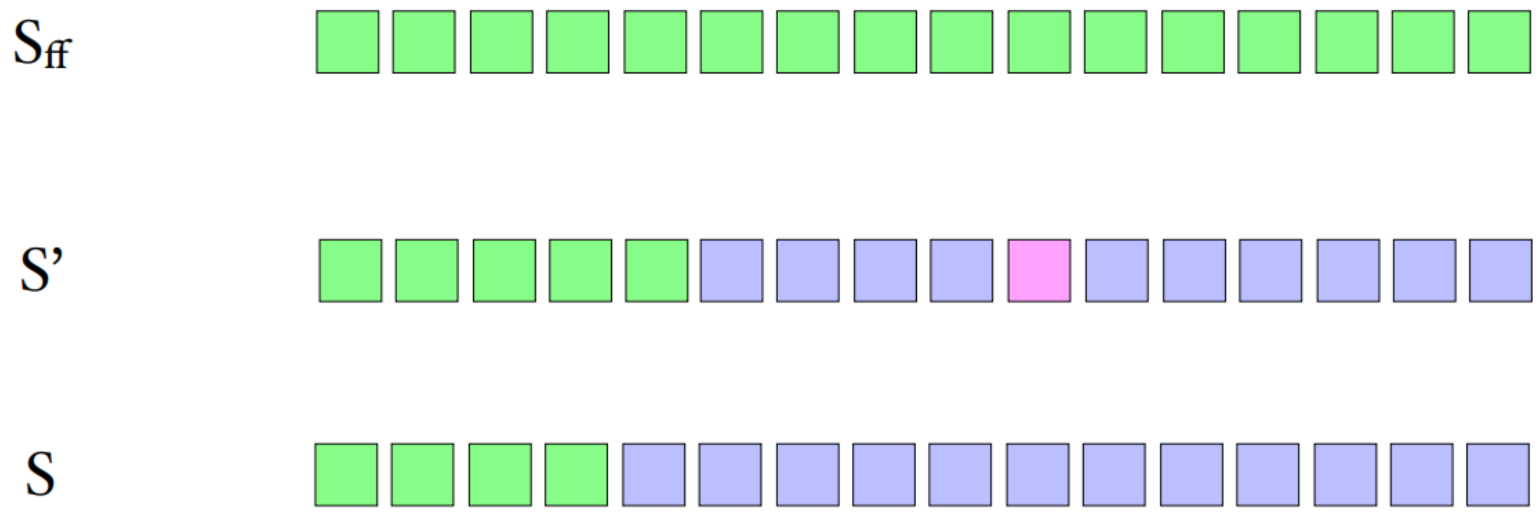
‘Evict f for t’

‘Evict e for t’

Either, t= t is something else

Same state of cache. Same # of misses!!!!

So what we have shown?



Let S be a reduced sched that agrees with S_{ff} on the first j items. There exists a reduced sched **S'** that agrees with S_{ff} on the first j+1 items and has the same or fewer #misses as S.