L14

4102

3.15.2016

abhi shelat

- Scheduling
- Cache problem

Scheduling

	start	end
sy333	2	3.25
en162	1	4
ma123	3	4
cs4102	3.5	4.75
cs4402	4	5.25
cs6051	4.5	6
sy333	5	6.5
cs1011	7	8

problem statement

$(a_1, \ldots, a_n)$ ← n activities

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

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

find largest subset of activities C={a_i} such that
(compatible)

for any two $a_i, a_j \in C$ s.t. $i < j$

$f_i < s_j$

problem statement

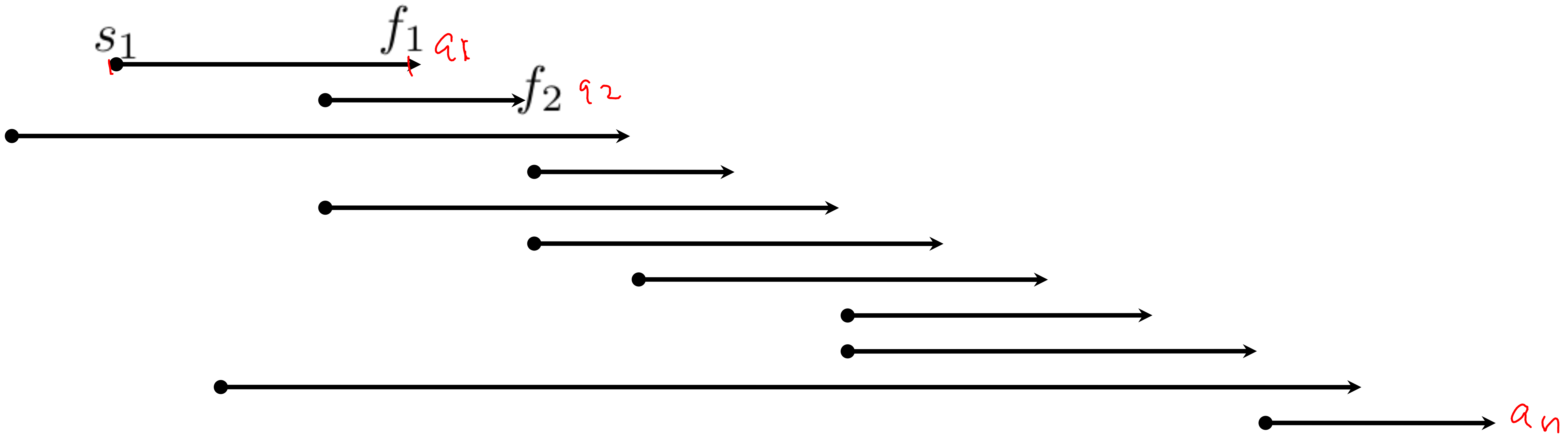
$$(a_1, \ldots, a_n)$$
$$(s_1, s_2, \ldots, s_n)$$
$$(f_1, f_2, \ldots, f_n)$$ (s0rted) $$s_i < f_i$$

find largest subset of activities C={a_i} such that
(compatible)

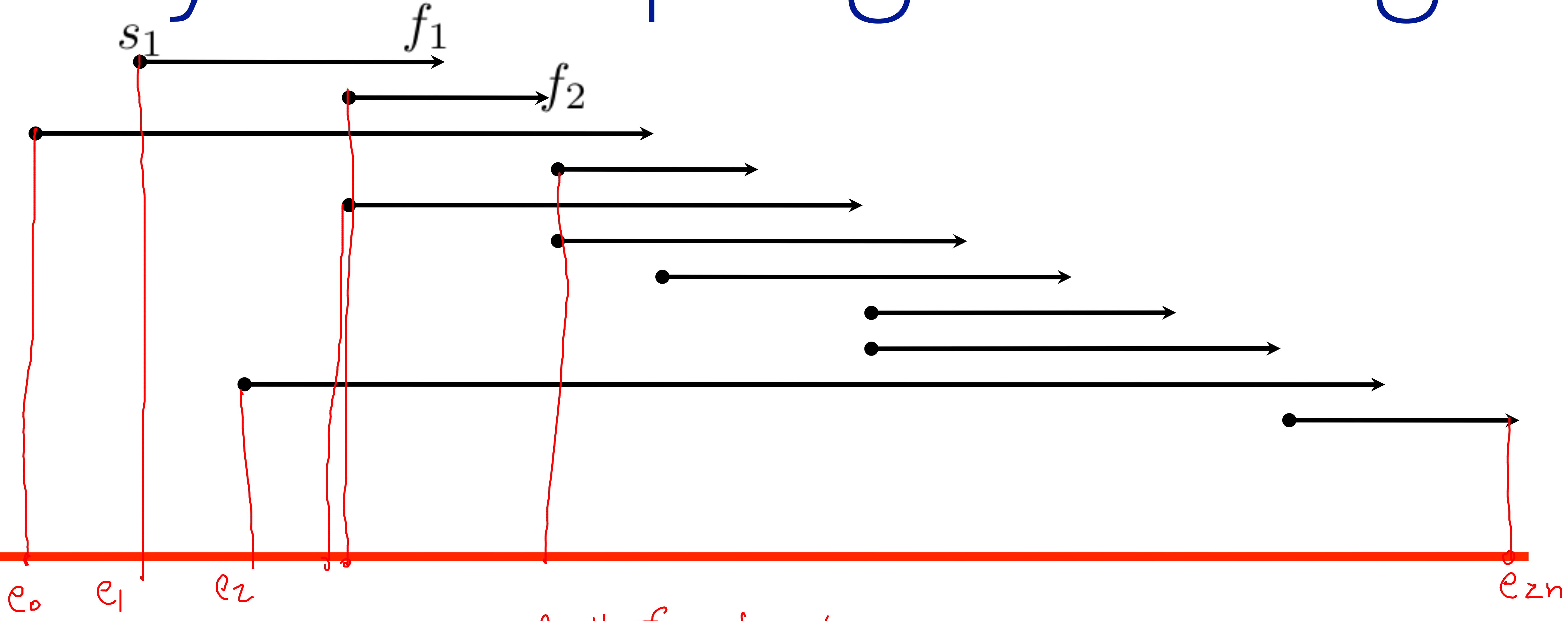
$$a_i, a_j \in C, i < j$$
$$f_i \leq s_j$$

problem statement

$$(a_1, \ldots, a_n)$$
$$(s_1, s_2, \ldots, s_n)$$
$$(f_1, f_2, \ldots, f_n) \text{ (sorted)} \quad s_i < f_i$$

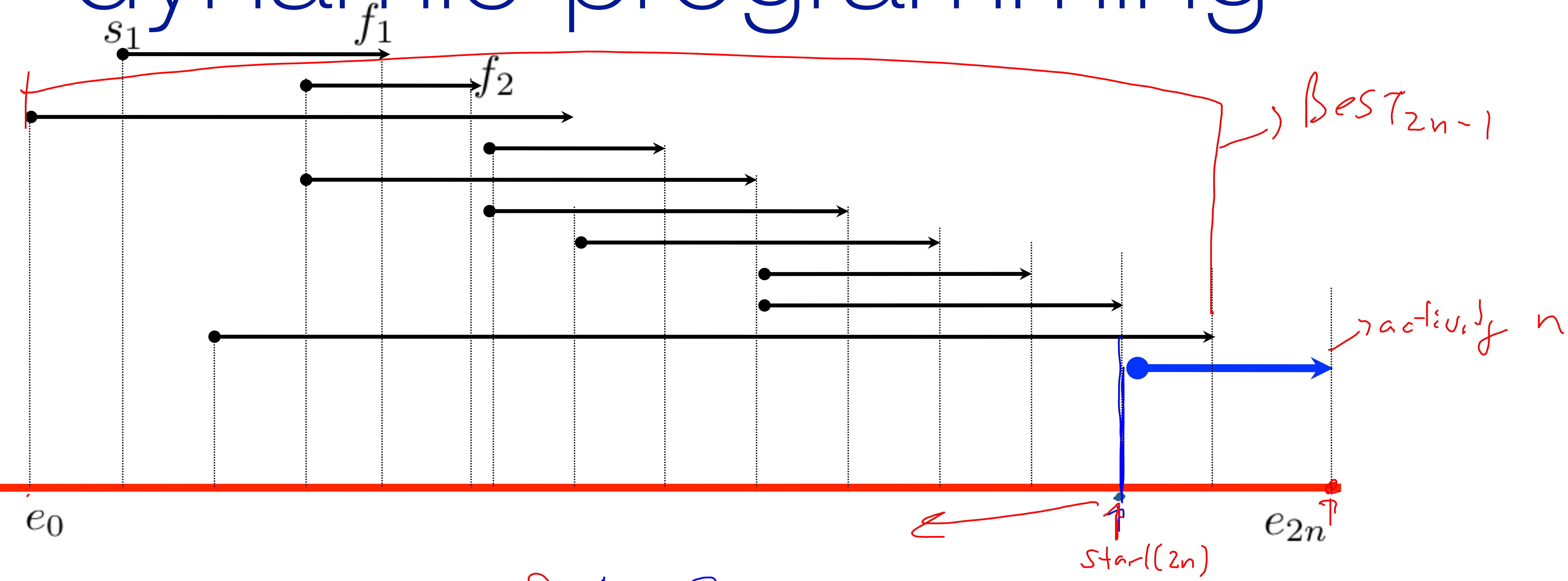


dynamic programming



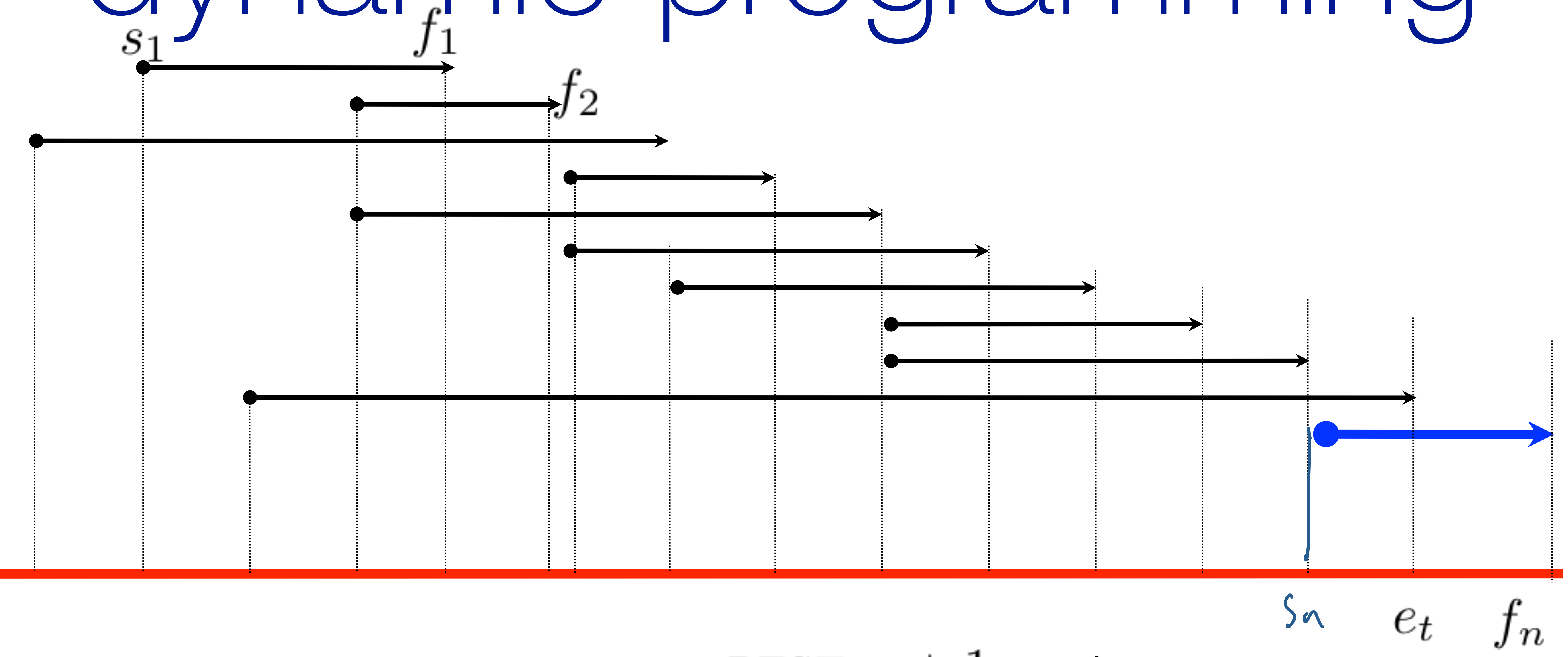
$Best_n$ = maximal # of activities that can occur before event n.

dynamic programming



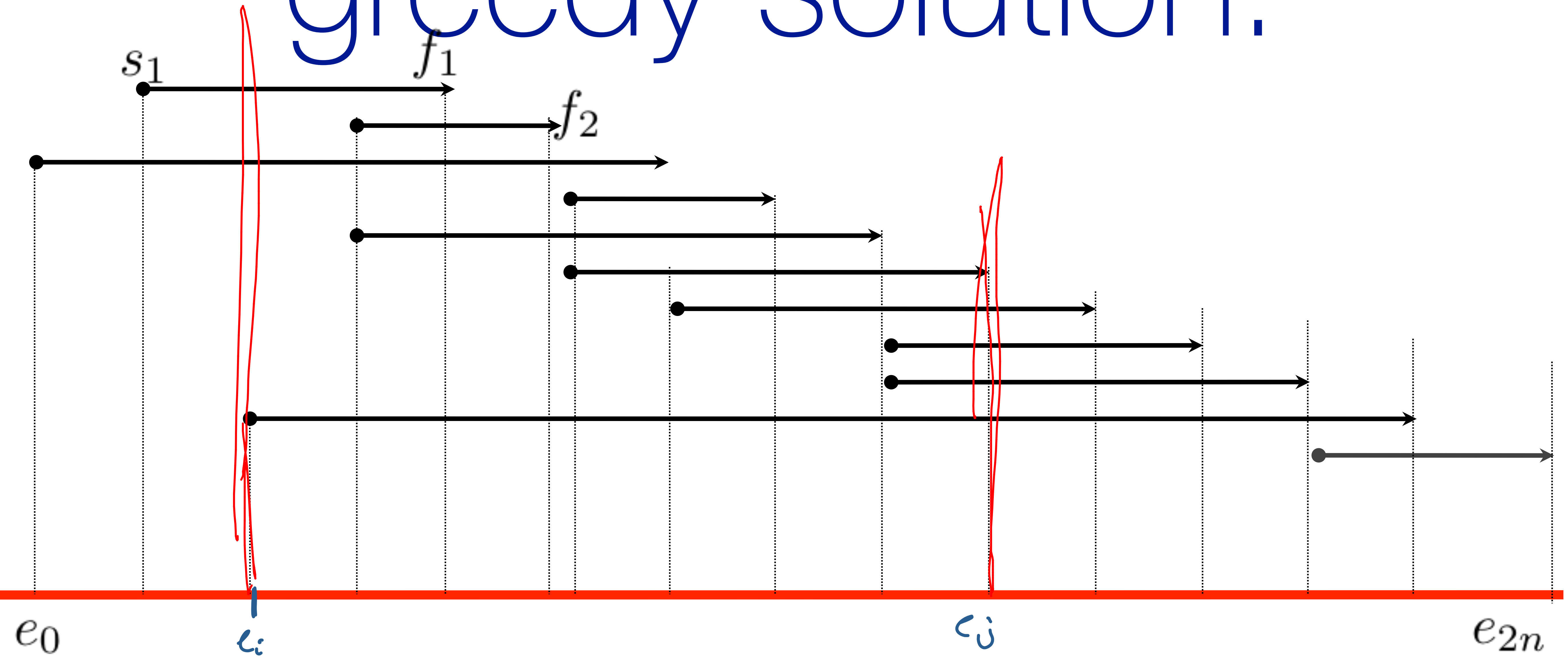
$$Best_{2n} = \max \begin{cases} 1 + Best_{start(2n)} \\ Best_{2n-1} \end{cases}$$

dynamic programming



$$\text{BEST}_{f_n} = \max \begin{cases} \text{BEST}_{s_n} + 1 & \text{in: } a_n \\ \text{BEST}_{e_t} & \text{out: } a_n \end{cases}$$

greedy solution:

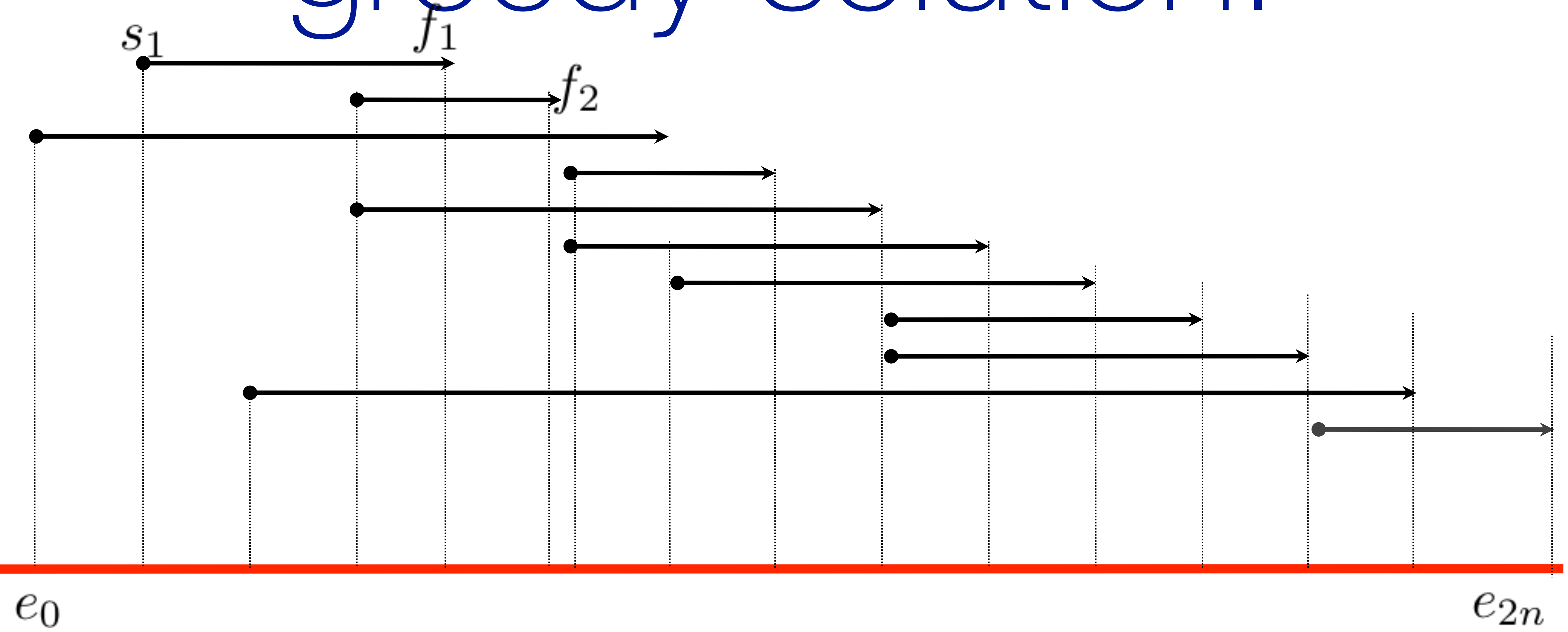


definition:

$\underbrace{\mathrm{SOLTN}_{i,j}}$ = maximal # of activities that occur between events i, j.

goal: $\mathrm{SOLTN}_{0,2n}$

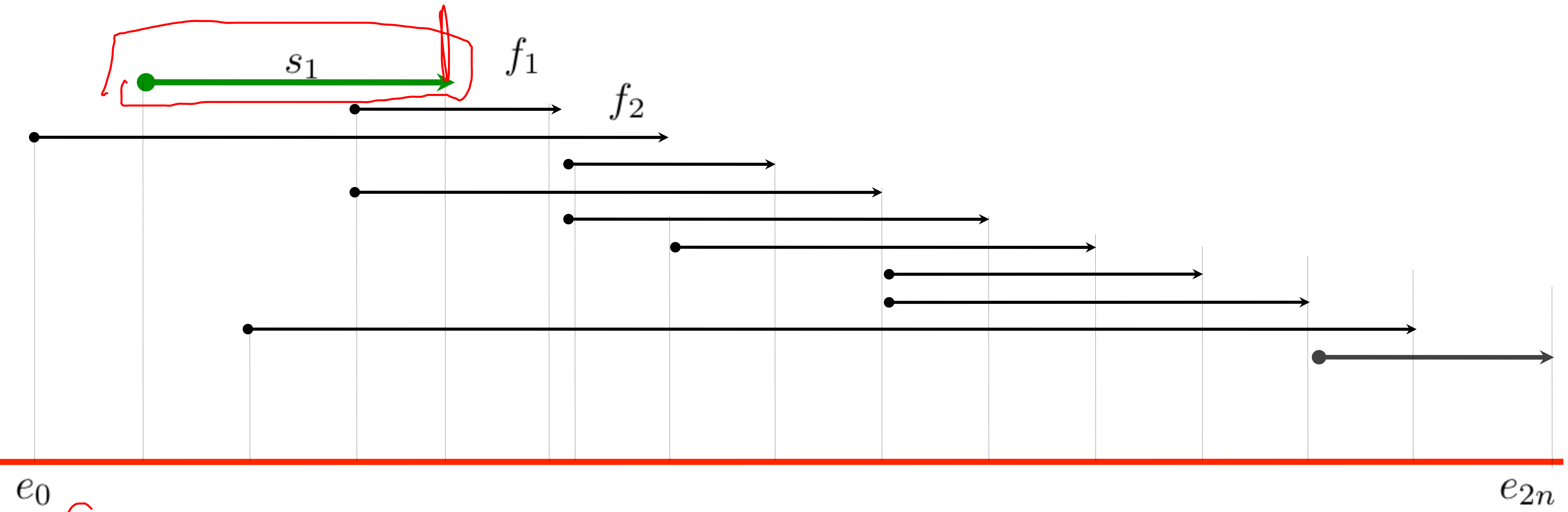
greedy solution:



$\mathrm{SOLTN}_{i,j}$

goal: $\mathrm{SOLTN}_{0,2n}$

greedy solution:



claim: the first action to finish in e[i,j] is always part of some $\textsc{soltn}_{i,j}$ → optimal solution for period [i,j]

⇒ a_1 is always part of some $\text{SOLUTN}_{0,2n}$

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

(Exchange arguments)

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

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

(2) Spse that $a^* \notin \text{SOLUTN}_{i,j}$. Let activity a be the first activity to finish in $\text{SOLUTN}_{i,j}$.

a^* finishes before a, i.e. $f_{a^*} \leq f_a$ by hypothesis.

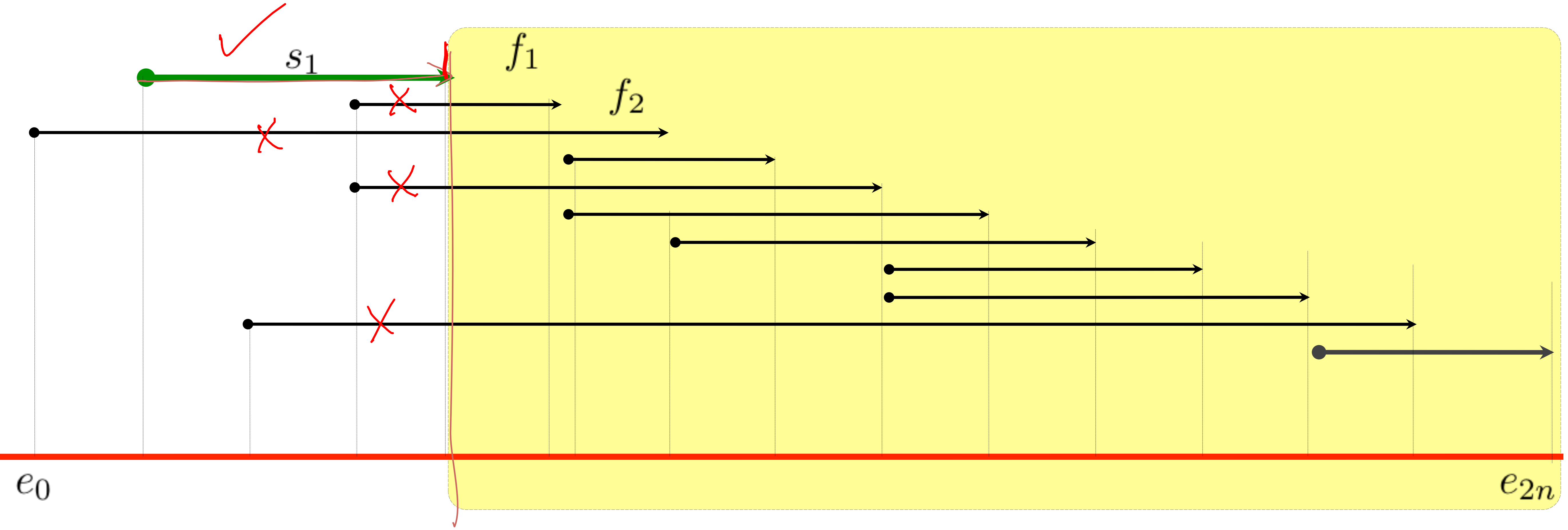
So therefore,

$$S = \text{SOLUTN}_{i,j} - \{a\} + \{a^*\}$$

then $|S| = |\text{SOLUTN}_{i,j}|$ and so S is optimal too.

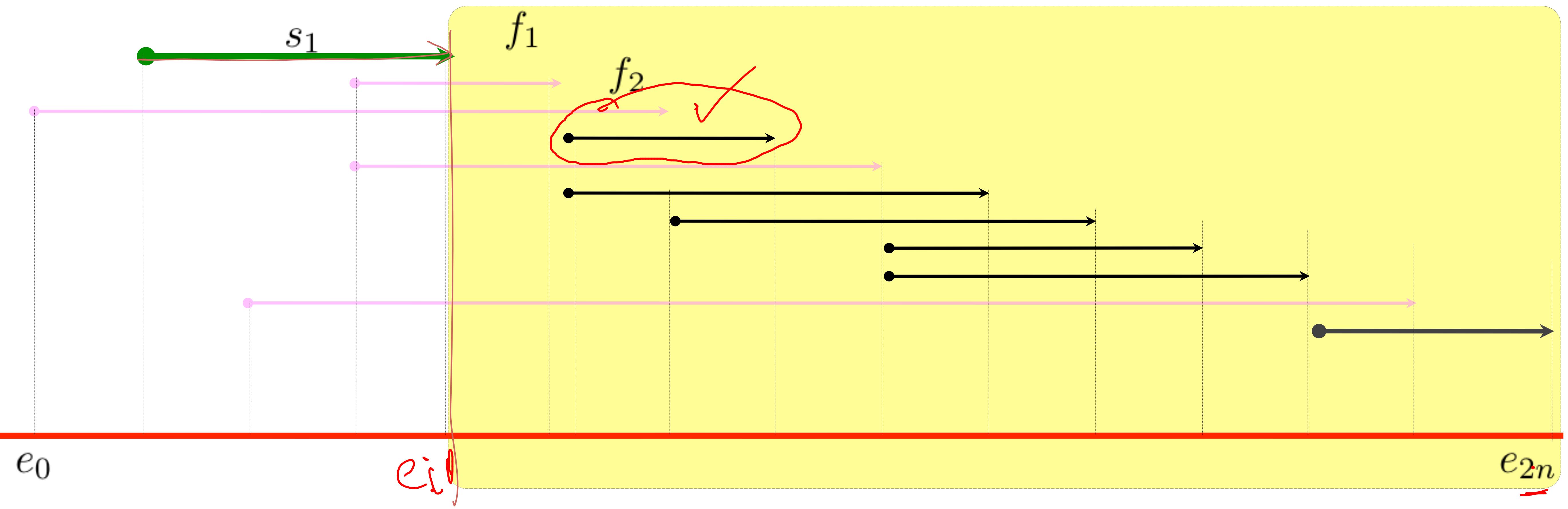
Therefore the lemma follows.

greedy solution:



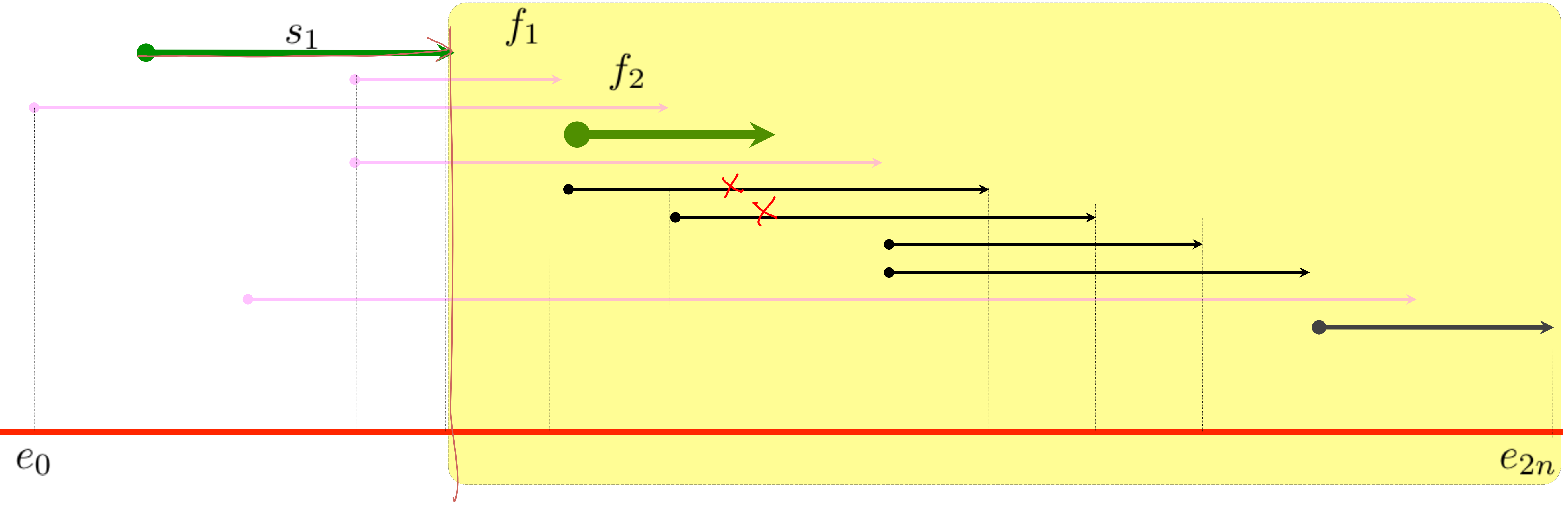
algorithm: find first event to finish. add to solution.
remove conflicting events.
continue.

greedy solution:



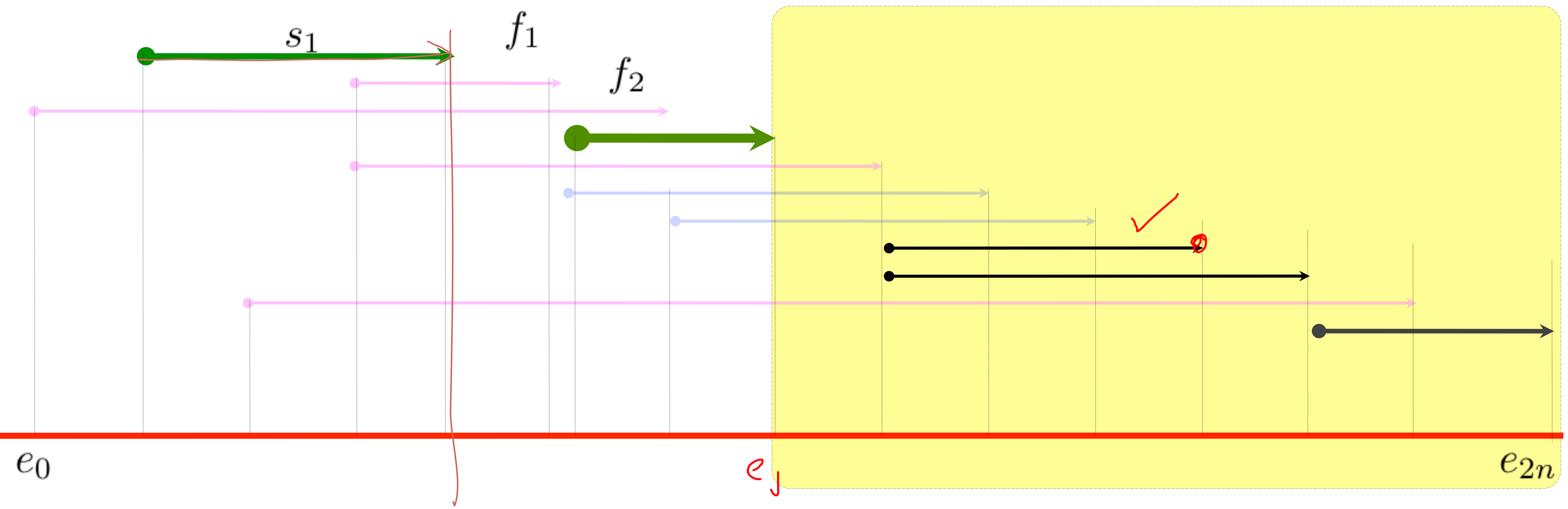
algorithm: find first event to finish. add to solution.
remove conflicting events.
continue.

greedy solution:



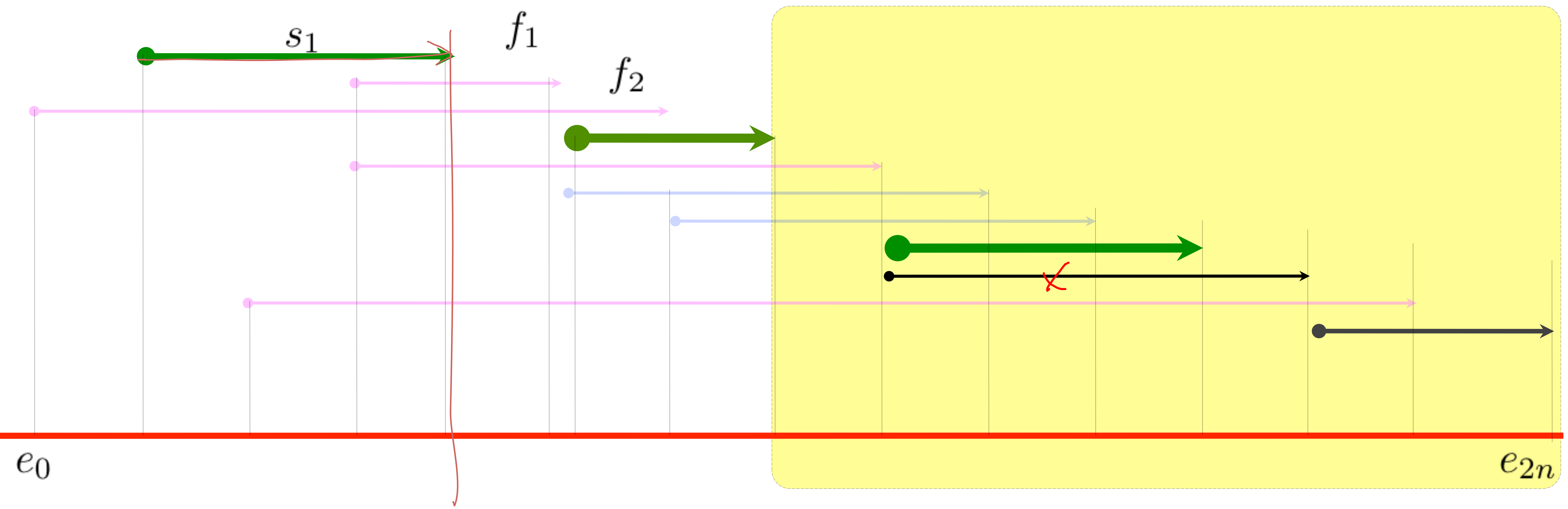
algorithm: find first event to finish. add to solution.
remove conflicting events.
continue.

greedy solution:



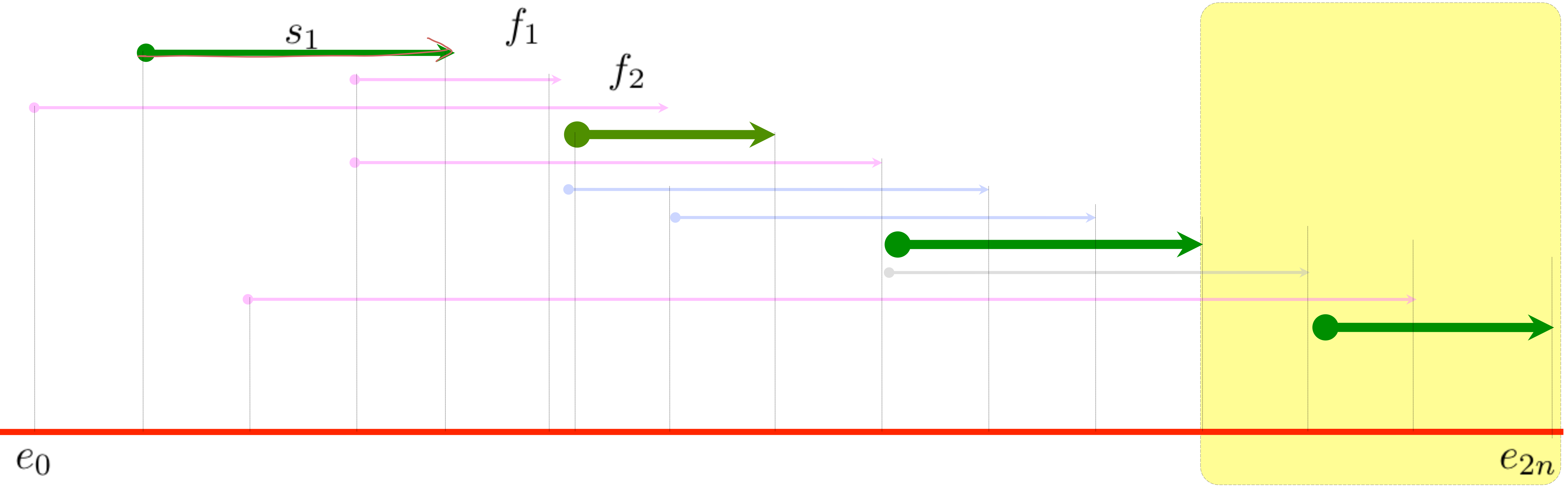
algorithm: find first event to finish. add to solution.
remove conflicting events.
continue.

greedy solution:



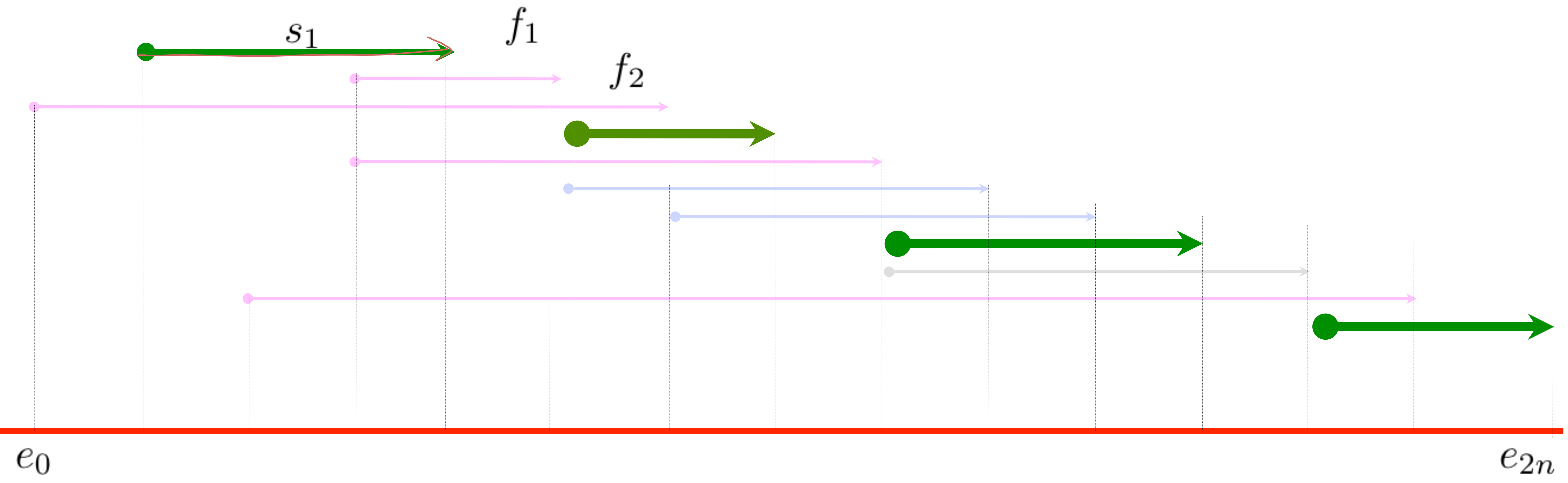
algorithm: find first event to finish. add to solution.
remove conflicting events.
continue.

greedy solution:



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

greedy solution:



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

running time

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

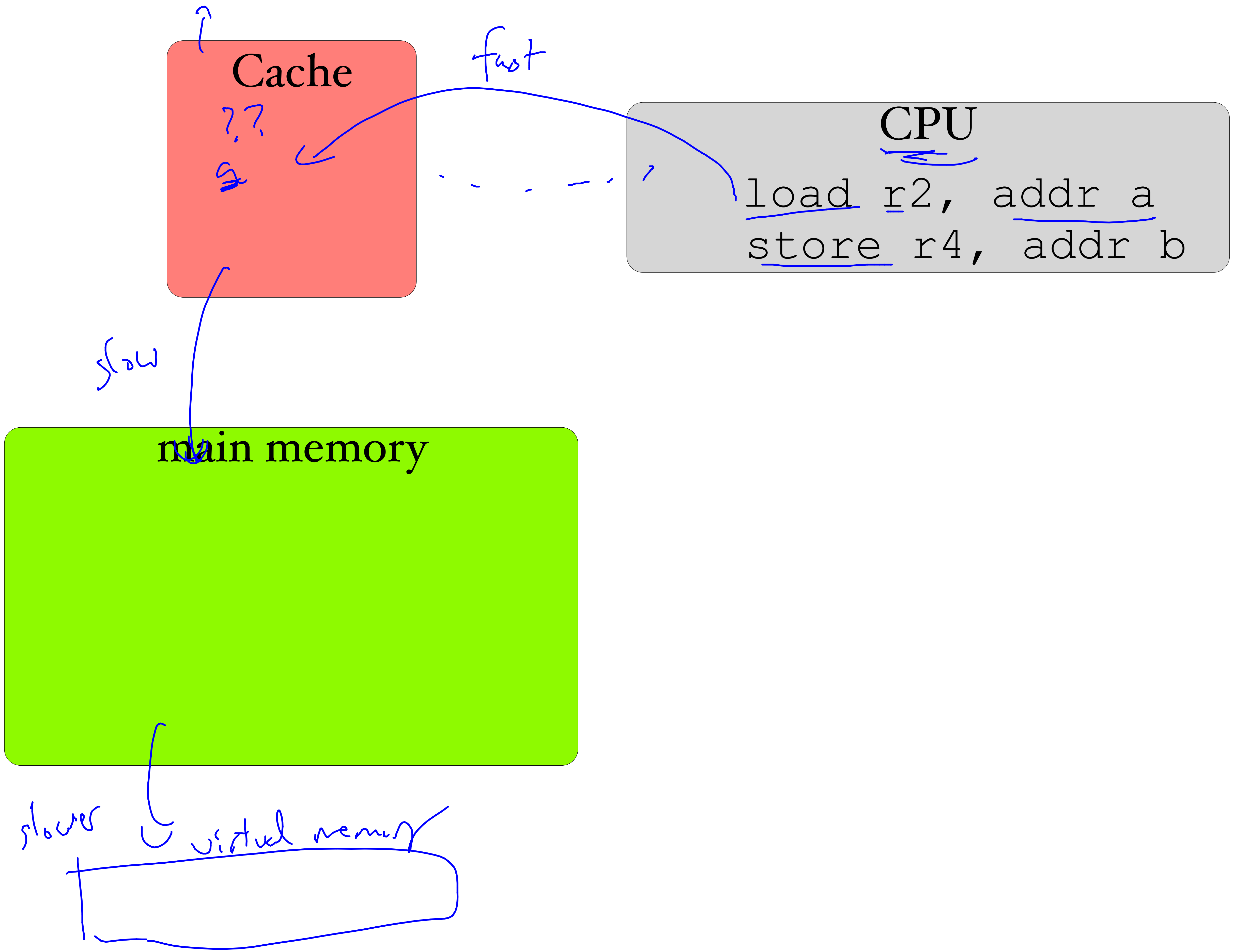
$\Theta(n)$

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

caching

cache hit

small cache
very fast



question:

(1) How can we manage a cache in order to minimize # of cache misses.

(*) simplify by assuming that we know all memory accesses beforehand

(*) cache is fully associative, line size = 1.

problem statement

input: K - cache size , $d_1, d_2, d_3, \ldots, d_n$ memory access pattern.

output: schedule for the cache

cache is

problem statement

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

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

cache is fully associative, line size is 1

contrast with reality

1. caches are not fully associative

2. ~~#~~!!! the manager <u>does not know</u> the future access pattern

Belady evict rule

"evict the item from the cache that is accessed farthest in the future"

→ Using the ff rule results in an optimal schedule.

example

cache

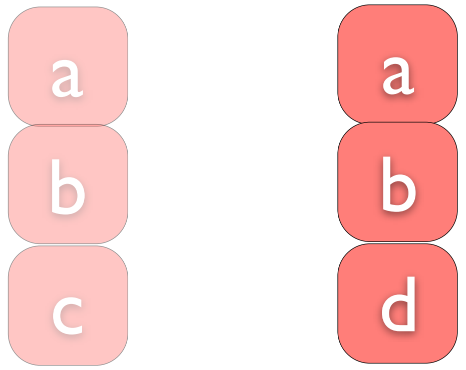
a
b
c

a b c d a d e a d b a e c e a ← memory access pattern

"evict c for d."

example

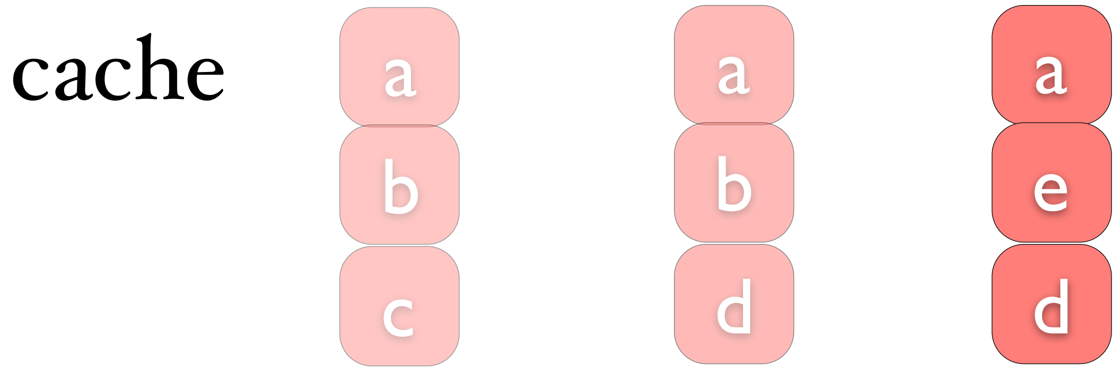
cache



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

“evict b for e”

example

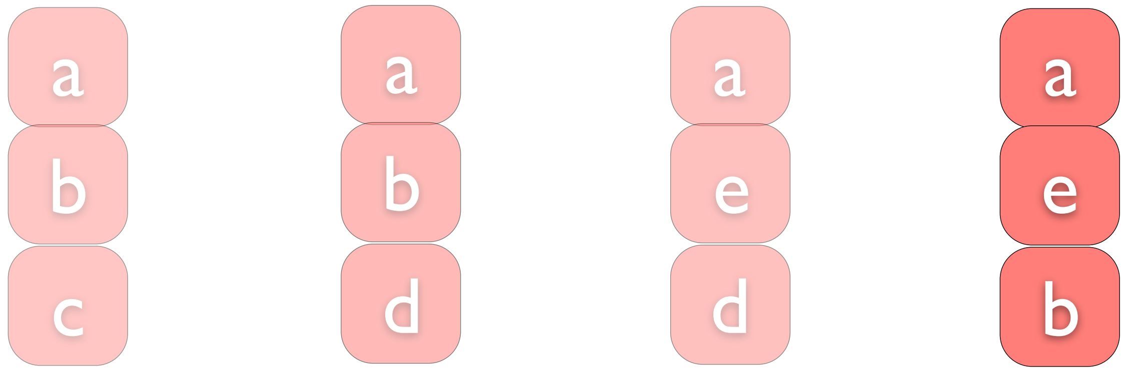


a b c d a d e a d b a e c e a

" evict d for b "

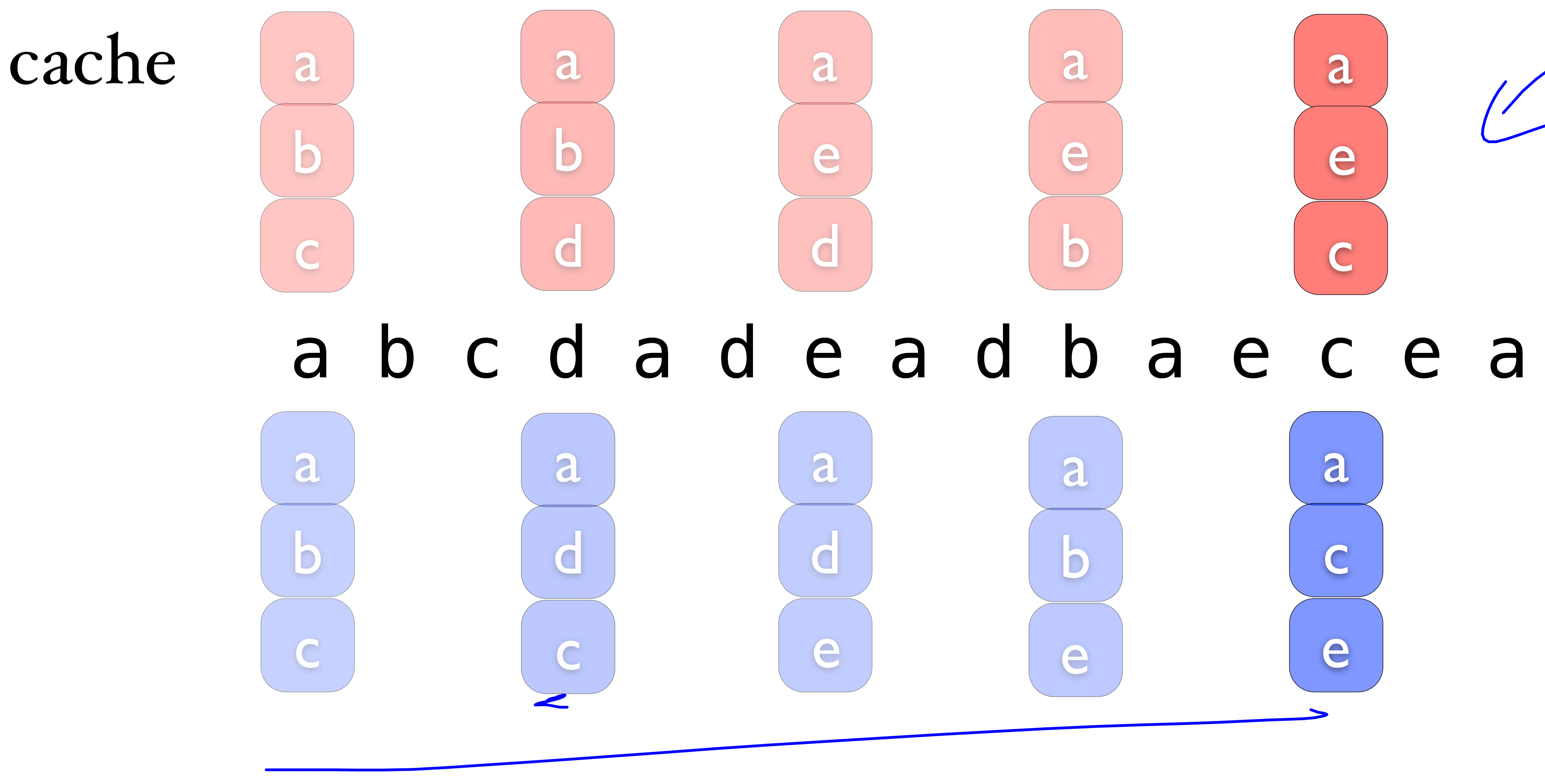
example

cache



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

example



Belady eviction rule.

4 cache misses

another schedule.

Not FF

4 cache misses.

Surprising theorem

- Belady eviction rules leads to an optimal schedule.

schedule

Schedule for access pattern $d_1, d_2, \ldots, d_n$: either the 'NOP' operation or the 'evict x for y' operation @ each step 1,…n.

- At step i, element di must be in the cache.

lazy

Reduced schedule: Schedule for which the operation "Evict x for y" only occurs at a step i if y = di.

Exchange lemma

Belady schedule.

Let reduced schedule S agree with schedule S_{ff} for the first j operations.

There exists a reduced schedule S' that agrees with S_{ff} on the first $j+1$ operations and

$$\#\text{misses}(S') \le \#\text{misses}(S).$$

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.

optimal schedule.

$$\boxed{S^* = S_0} \xrightarrow{\text{lemma}} \underline{S_1} \xrightarrow{\text{lemma}} S_2 \longrightarrow \cdots \longrightarrow \cdots \longrightarrow S_{ff}$$

$\#\text{misses}(S^*)$ $\qquad\qquad$ $= \#\text{misses}(S_{ff})$

S_1 agrees with S_{ff} on 1 operation

$$\#\text{misses}(S_1) \leq \#\text{misses}(S^*) = \#\text{misses}(S_2)$$

Proof of Lemma

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.

Proof: Since S agrees with S_{ff} on the first j operations, then the state of the cache @ operation $j+1$ will be the same. Let d be the address accessed @ this operation $j+1$.

Proof of lemma

operation j+1 accesses d.

State of the cache after J operations under the two schedules.

easy case 1 Spse $d \in$ cache. Then $S' = S$ b/c both S and S_{ff} issue "NOP."

$\Rightarrow$ S' agrees with S_{ff} and has #misses(S)

easy case 2

Spse $d \notin$ cache, but both S and S_{ff} "evict e for d"

Same arguments apply from case I.

Proof of lemma

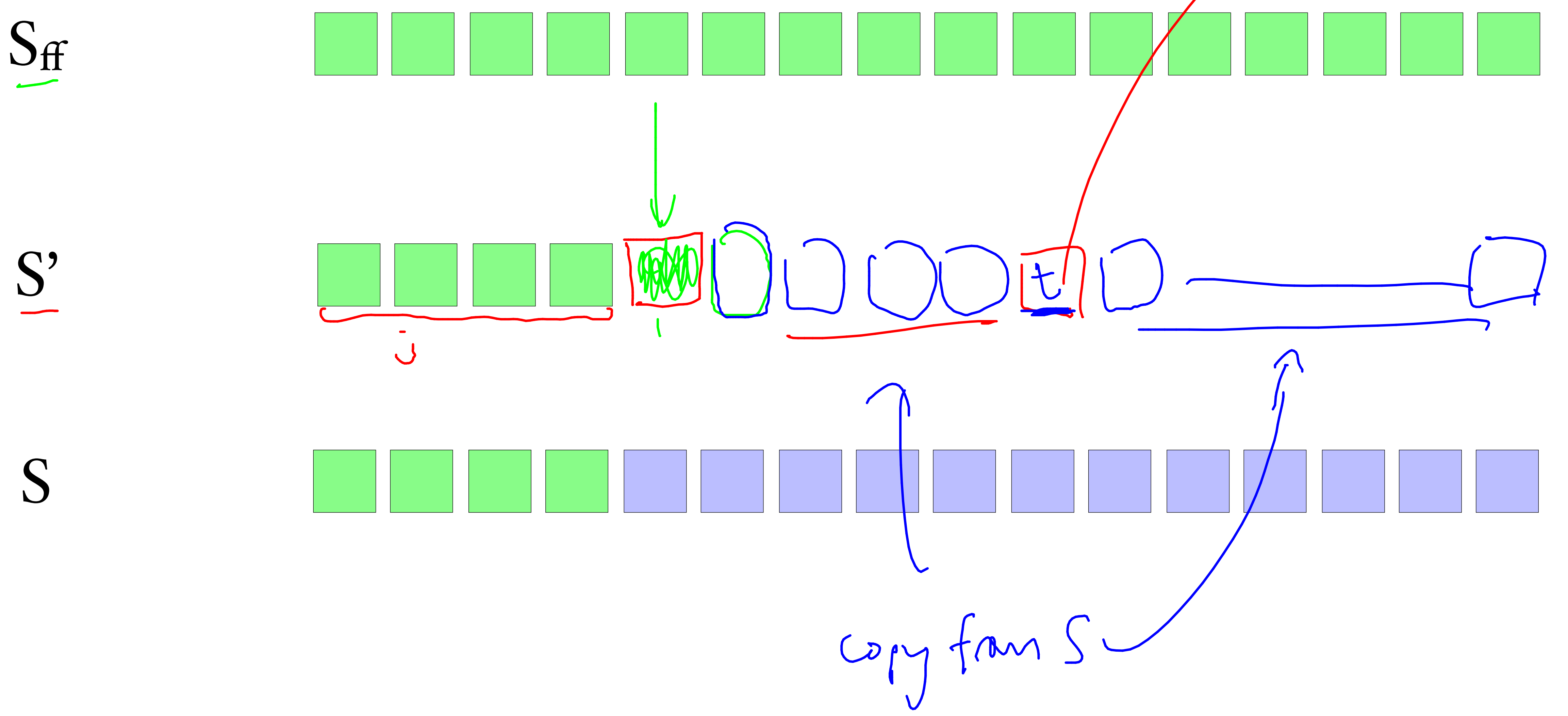
case 3 $d \notin$ cache, S evicts e. S_{ff} evicts f.

We need to construct a schedule $\underline{S'}$ that satisfies

1. agrees w/ S_{ff} on $j+1$
2. has same # of misses as S

Timeline

t → first operation in S that involves either e or f.



Proof of lemma

Let access t be the first operation in S after $j+1$ that involves either $\underline{e \text{ or } f}$.

Either $\underline{t = e}$

$\underline{t = f}$

$\underline{t = g \neq e \neq f}$

Proof of lemma

S | x | d | f

S’ | e | d

what if g=e ? S must load e.

"Evict x for e"

b/c this is the first operation after j+1 that involve e or f

S' can issue the operation

"evict x for f"

as a result, S' and S will have the same state of the cache and therefore the same # of misses.

Return Reduce(S')

Proof of lemma

what if g=f ?

<u>Cannot happen.</u> Because S_{ff} uses the "farthest in the future" rule, so f cannot be accessed before e.

Proof of lemma

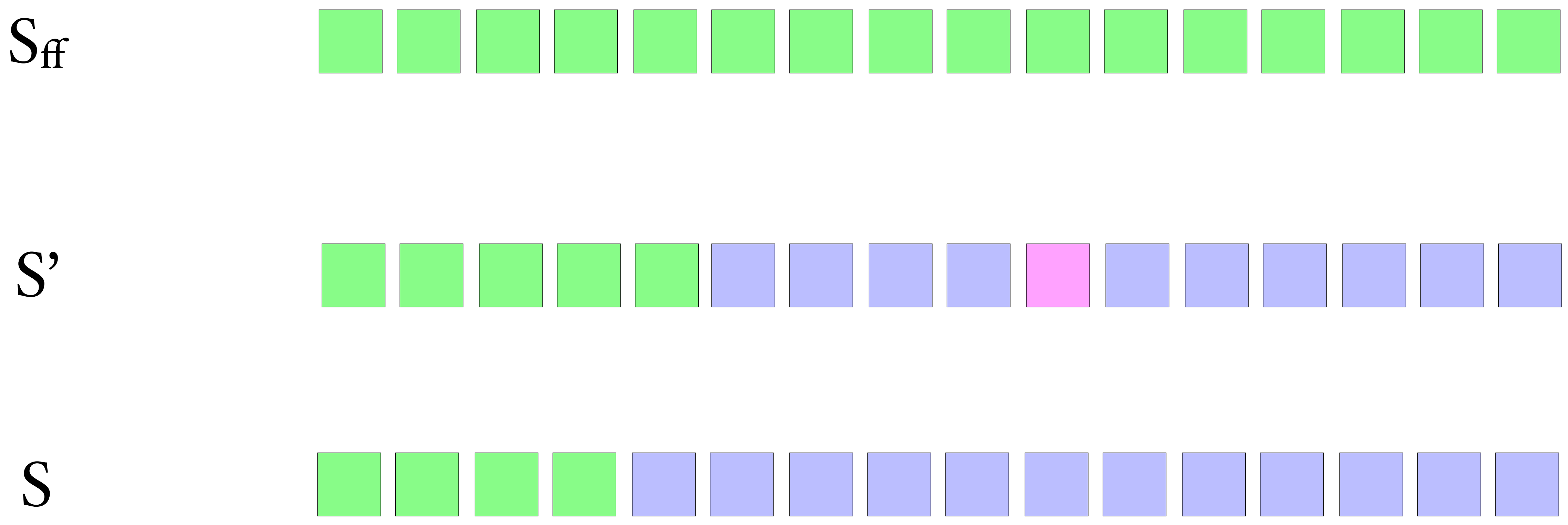
what if ~~g~~ f is neither e nor f ?

"evict f for t"

"evict e for t"

caches will be the same again.

What have we 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.

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.

S^*

S_{ff}