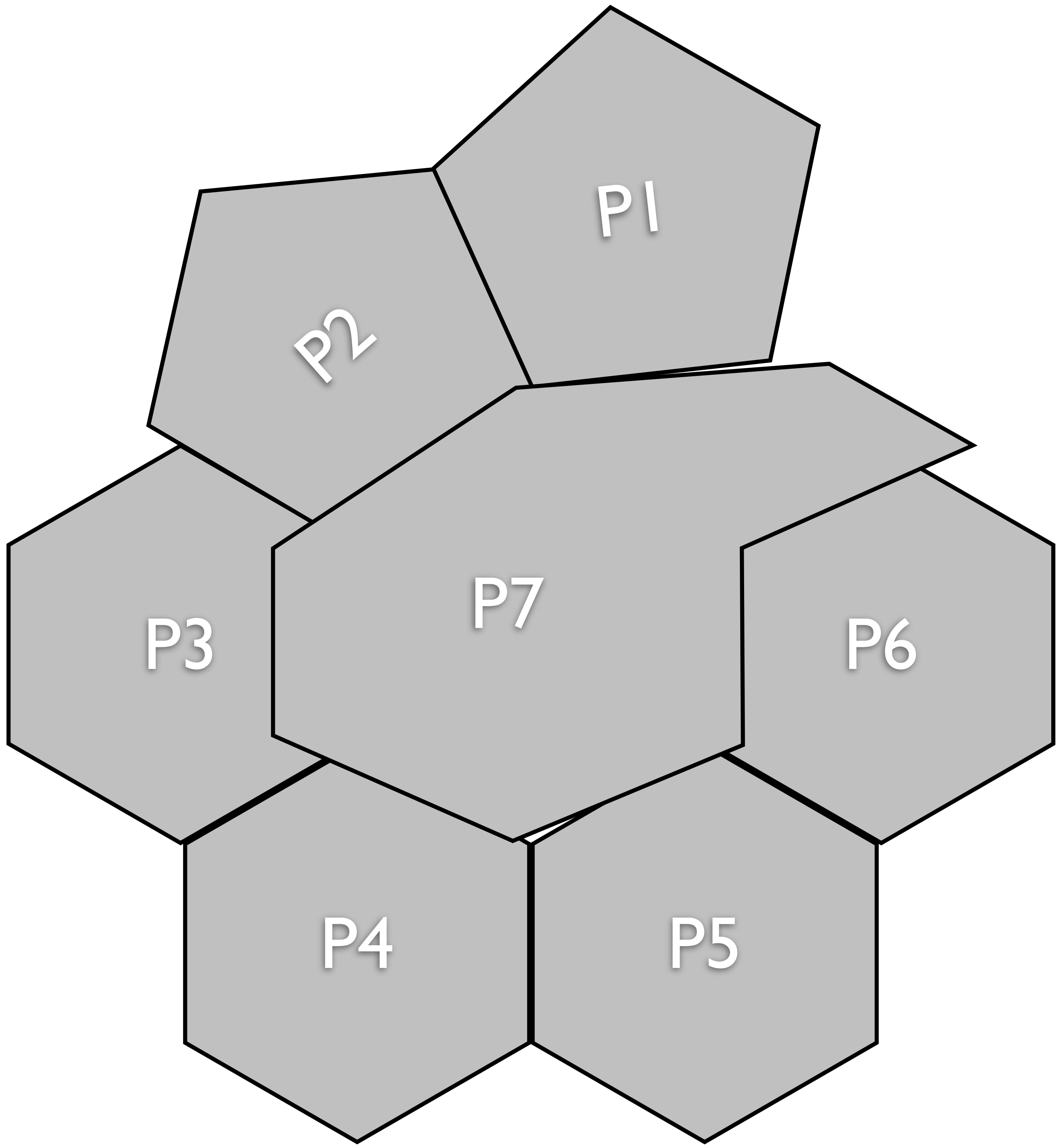
L13

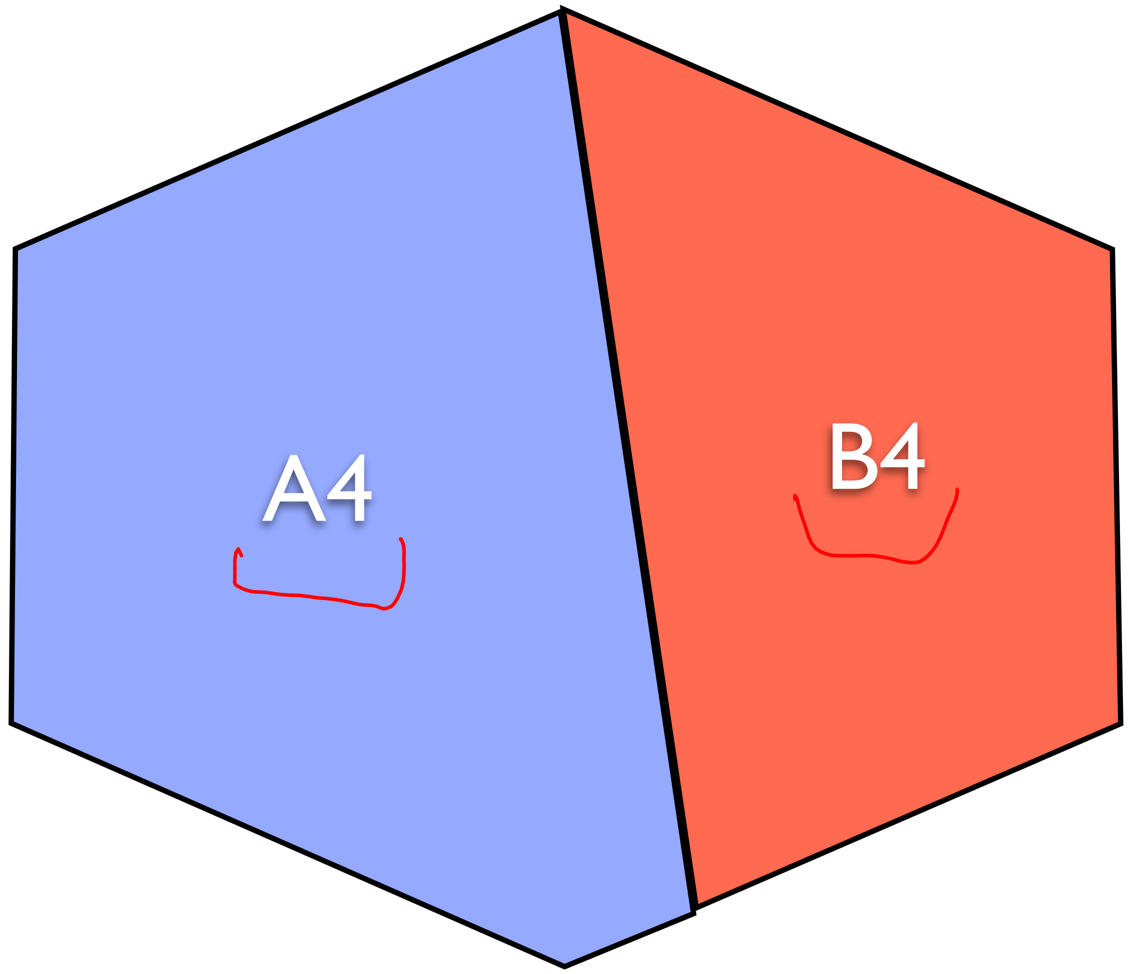
4102

10.8.2013

abhi shelat

Finish Gerry
Intro Greedy
Schedule
Caching


P1
P2
P7
P3
P6
P4
P5



$A_i + B_i = M$ ← # people in a district

A_i = # of people that vote for A in precinct i

GERRYMANDER PROBLEM

given: M, $A_1, A_2, \ldots A_n$ n is even

output: 2 districts D_1, D_2. $|D_1| = |D_2|$ i.e same # of precincts

$$A(D_1) > \frac{M \cdot n}{4}$$

$$A(D_2) > \frac{M \cdot n}{4}$$

i.e. party A has a majority in both districts.

GERRYMANDER PROBLEM

given: $m \quad A_1, A_2, \ldots, A_n$ n is even

$\uparrow$ # of people that vote for A in P_i

output: D_1, D_2

such that $|D_1| = |D_2|$

$$A(D_1) > \frac{mn}{4}$$

$$A(D_2) > \frac{mn}{4}$$

why ??

or "failure" if no such solution is possible

because $|D_1| = \frac{n}{2}$

$\Rightarrow$ # people in D_1

$\frac{Mn}{2}$

$\Rightarrow$ majority in D_1

$\geq \frac{Mn}{4}$

GERRYMANDER

imagine very last precinct and how it is assigned:

GERRYMANDER

$S_{j,k,x,y}$ = true or false variable

TRUE if $\exists$ an assignment of the first

j precincts s.t.

$|D_1| = k$

$A(D_1) = x$

$A(D_2) = y$.

GERRYMANDER

$S_{j,k,x,y}$ = there is a split of first **j** precincts
in which |D1|=**k** and
x people in D1 vote A
y people in D2 vote A

$$S_{j,k,x,y} = S_{j-1,k-1,x-A_j,y} \quad \text{OR} \quad S_{j-1,k,x,y-A_j}$$

Brute force

$$\binom{n}{n/2} - \quad 2^{n/2}$$

$$S_{j,k,x,y} = S_{j-1,k-1,x-A_j,y} \vee S_{j-1,k,x,y-A_j}$$

GERRYMANDER(P,A,m)

$C_{j,k,x,y} = 1$ or 2

initialize array S[0,o,o,o]

for j=1 to n — $\Theta(n)$

for k=1 to n/2 — $\Theta(n)$

for x=1 to m·j — $\Theta(m \cdot n)$

for y=1 to m·j — $\Theta(m \cdot n)$

$\sim \Theta(n^4 \cdot m^2)$

$S_{j,k,x,y} = S_{j-1,k-1,x-A_j,y}$ OR $S_{j-1,k,x,y-A_j}$

Check if any $S_{n,n/2,x,y}$ is true for $x,y > \frac{m \cdot n}{4}$

$$S_{j,k,x,y} = S_{j-1,k-1,x-A_j,y} \;\vee\; S_{j-1,k,x,y-A_j}$$

GERRYMANDER(P,A,m)

```
initialize array S[0,o,o,o]
for j=1,...,n
    for k=1,...,n/2
      for x=0,...,jm
        for y=0,...,jm
              fill table according to equation
 search for true entry at S[n,n/2, >mn/4, >mn/4]
```

Scheduling

A New technique: Greedy

	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

$$(a_1, \ldots, a_n)$$

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

$$(f_1, f_2, \ldots, f_n) \qquad \text{(SORTED)} s_i < f_i$$

(COMPATIBLE)

FIND LARGEST SUBSET OF ACTIVITIES C={a_i} SUCH THAT

$a_i, a_j \in C \qquad i < j$

$f_i \le s_j$

PROBLEM STATEMENT

$(a_1, \ldots, a_n)$

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

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

(COMPATIBLE)

FIND LARGEST SUBSET OF ACTIVITIES C={a_i} SUCH THAT

$$a_i, a_j \in C, i < j$$

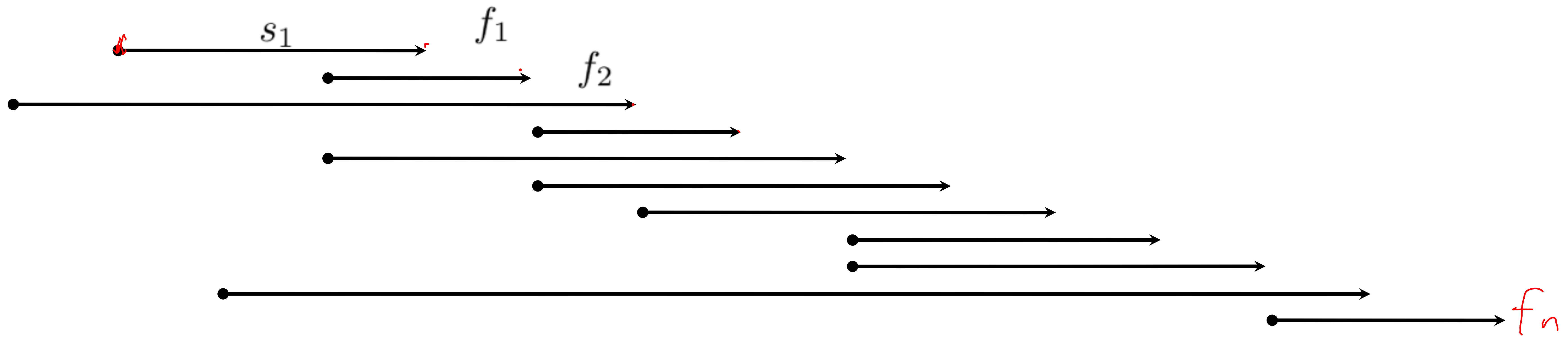
$$f_i \leq s_j$$

PROBLEM STATEMENT

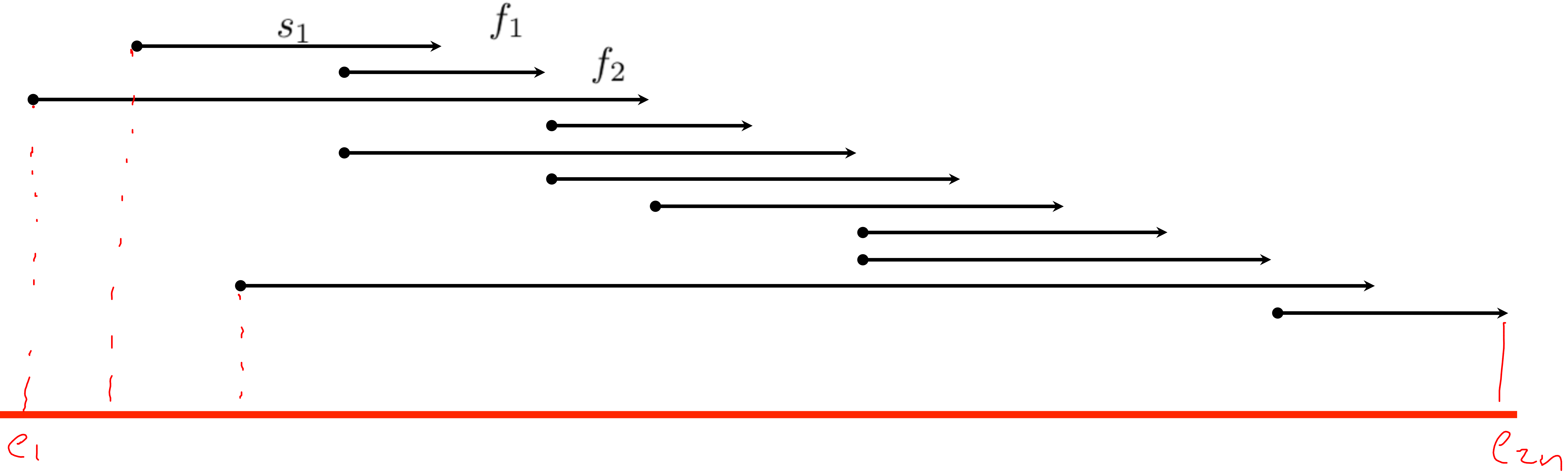
$(a_1, \ldots, a_n)$

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

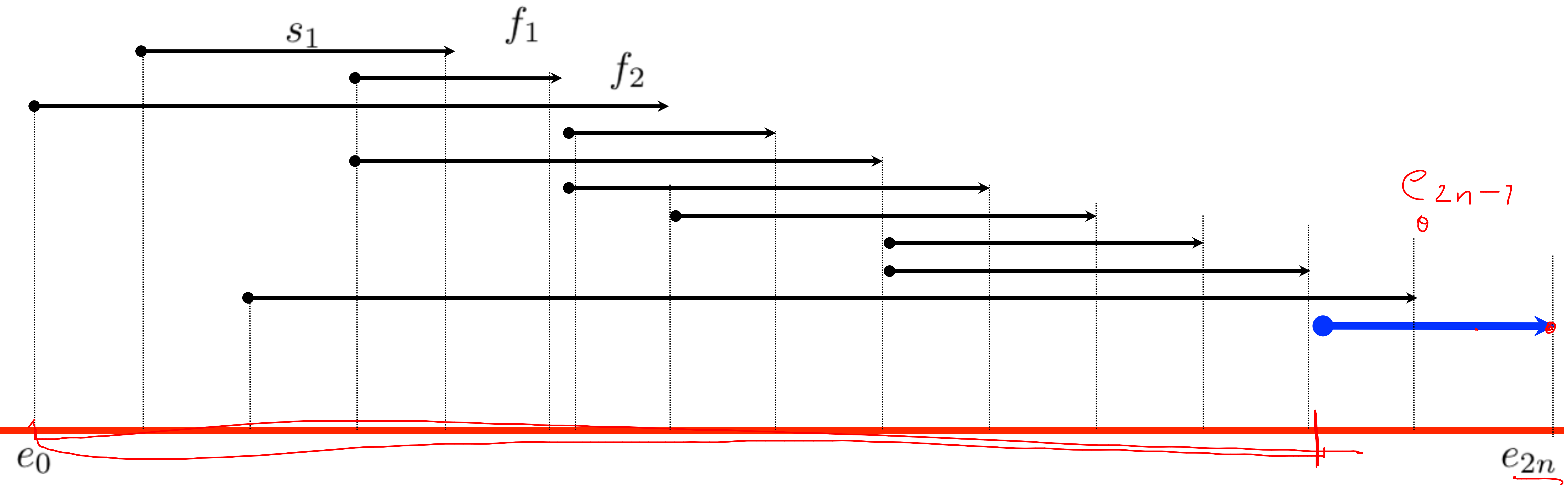
$(f_1, f_2, \ldots, f_n)$ (SORTED) $s_i < f_i$



DYNAMIC PROGRAMMING



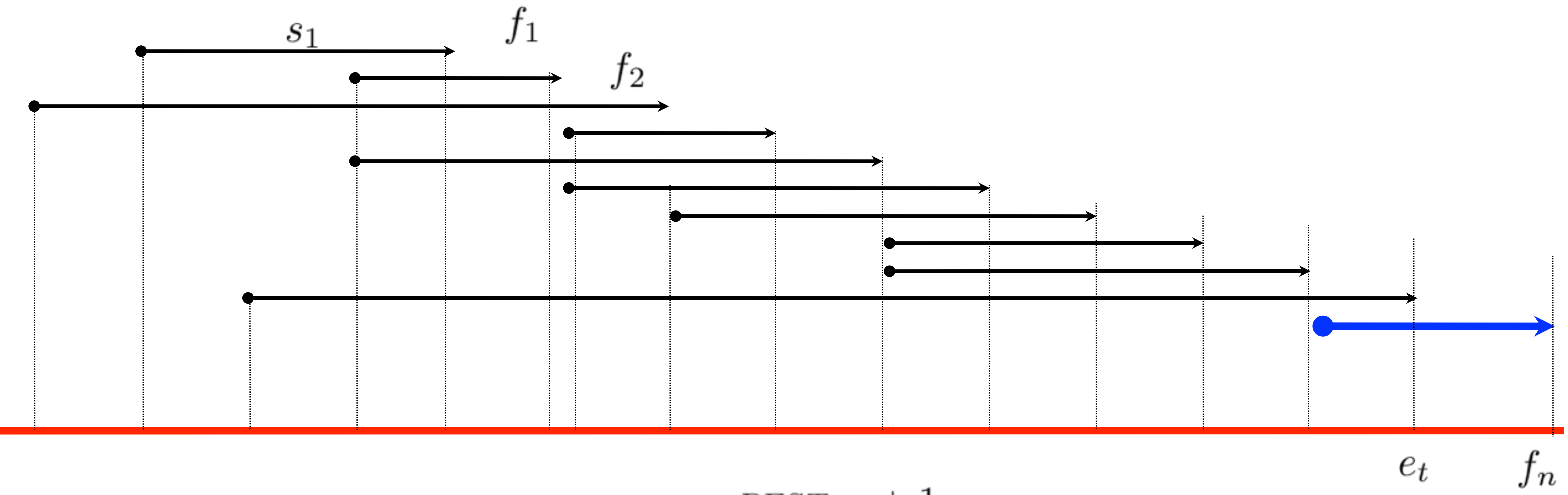
DYNAMIC PROGRAMMING



$Best_m$: most number of classes that can be scheduled between event 0 and event m.

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

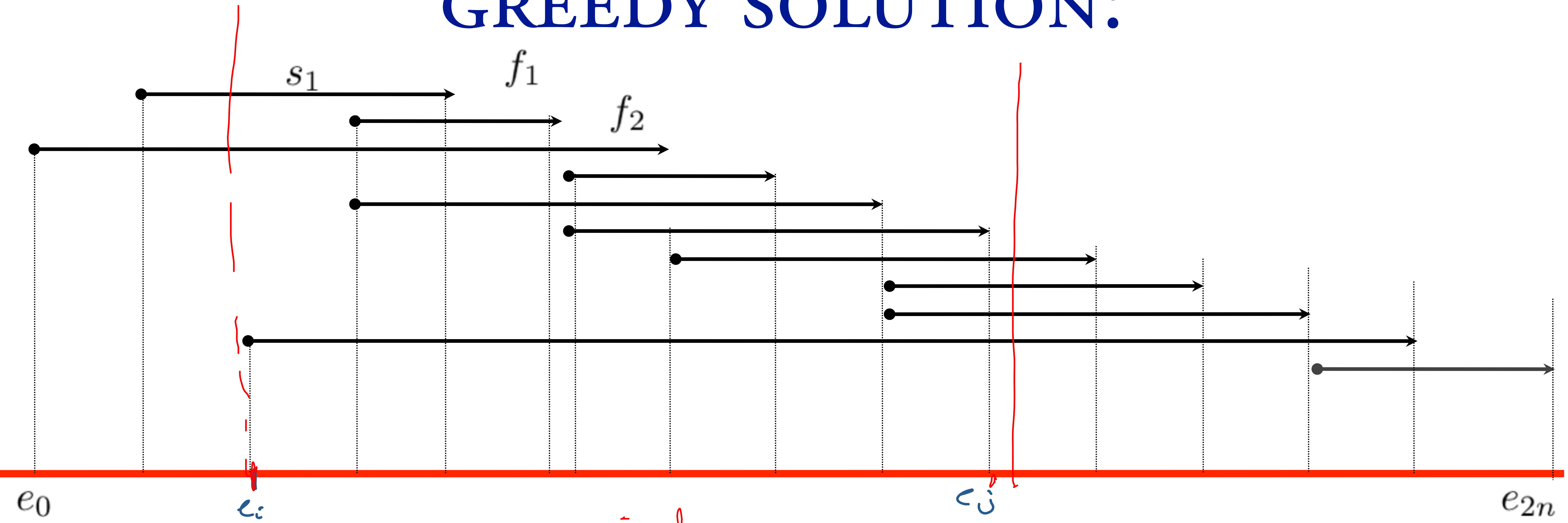
DYNAMIC PROGRAMMING



$\text{BEST}_{f_n} =$ MAX $\text{BEST}_{s_n} + 1$ a_n IN:

BEST_{e_t} a_n OUT:

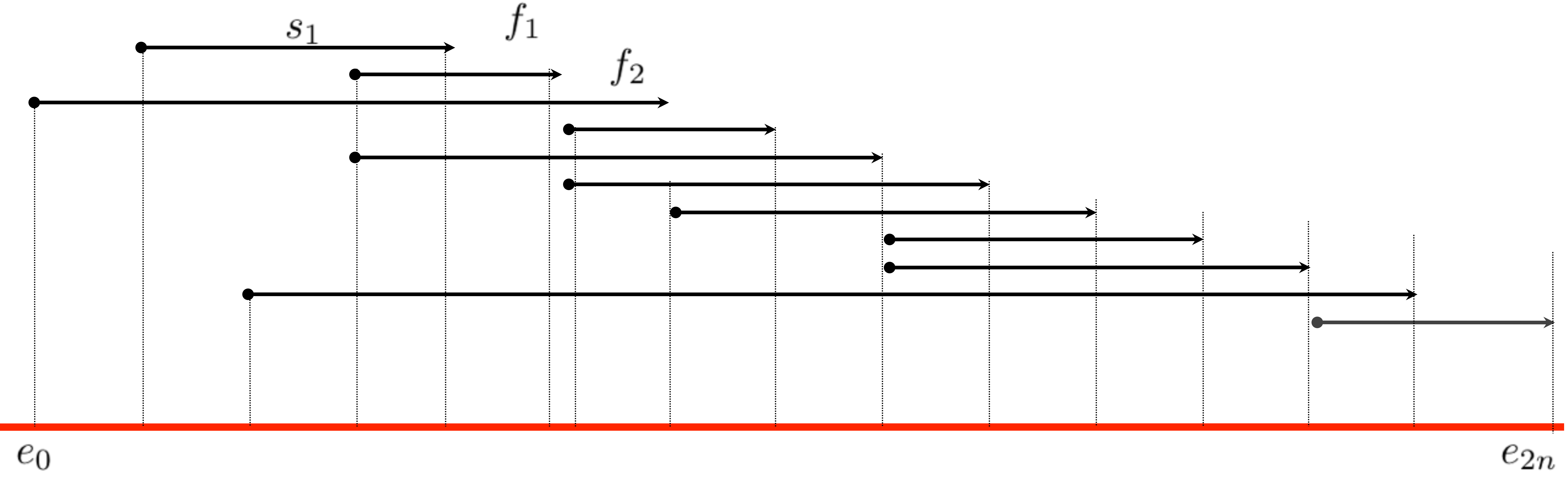
GREEDY SOLUTION:



DEFINITION:

$\text{SOLTN}_{i,j}$ → maximal set of activities for period i, j

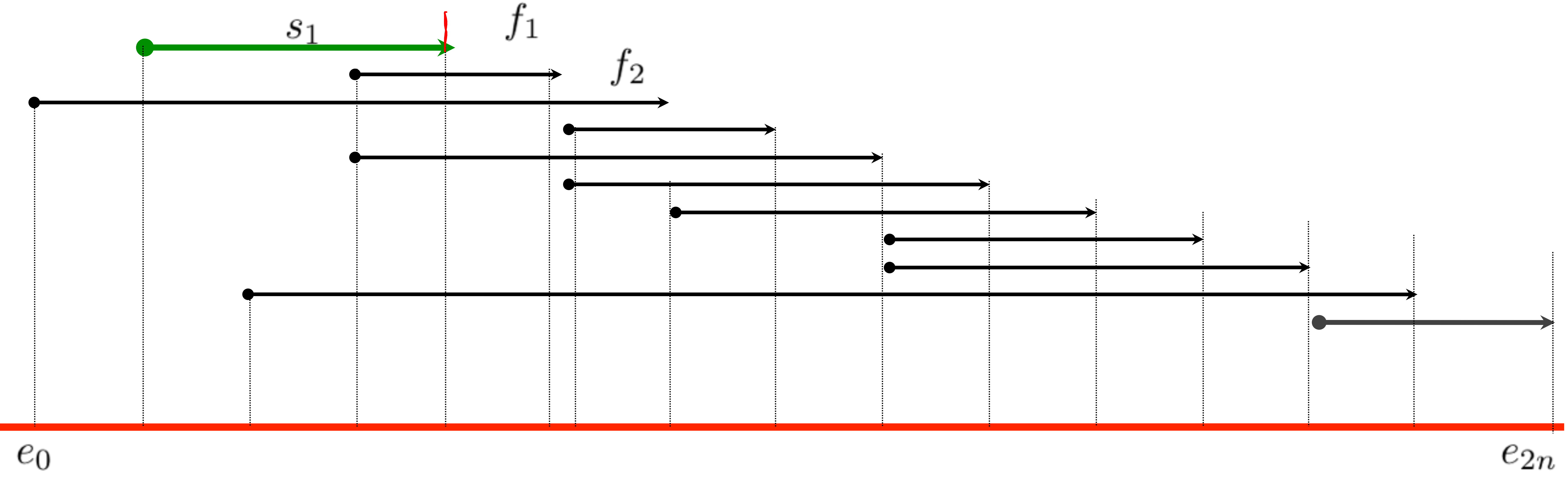
GREEDY SOLUTION:



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

GOAL: $\text{SOLTN}_{0,2n}$

GREEDY SOLUTION:



CLAIM: THE FIRST ACTION TO FINISH IN `e[i,j]` IS ALWAYS PART OF SOME $\text{SOLTN}_{i,j}$

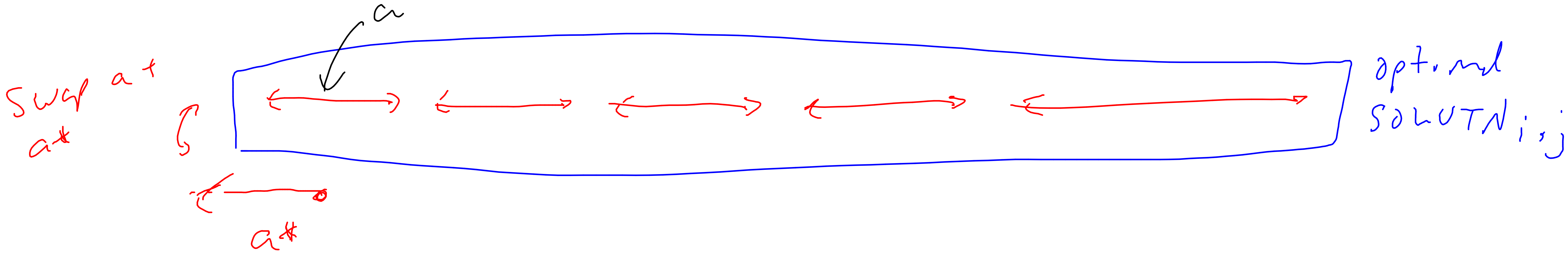
"first-to-finish is always part of the solution"

CLAIM: THE FIRST ACTION TO FINISH IN `e[i,j]` IS ALWAYS PART OF SOME $\text{SOLTN}_{i,j}$

EXCHANGE ARGUMENT

PROOF: Consider the optimal $\text{SOLUTN}_{i,j}$. Let a^* to be the f-to-f in period $[i,j]$. Suppose that $a^* \notin \text{SOLUTN}_{i,j}$.

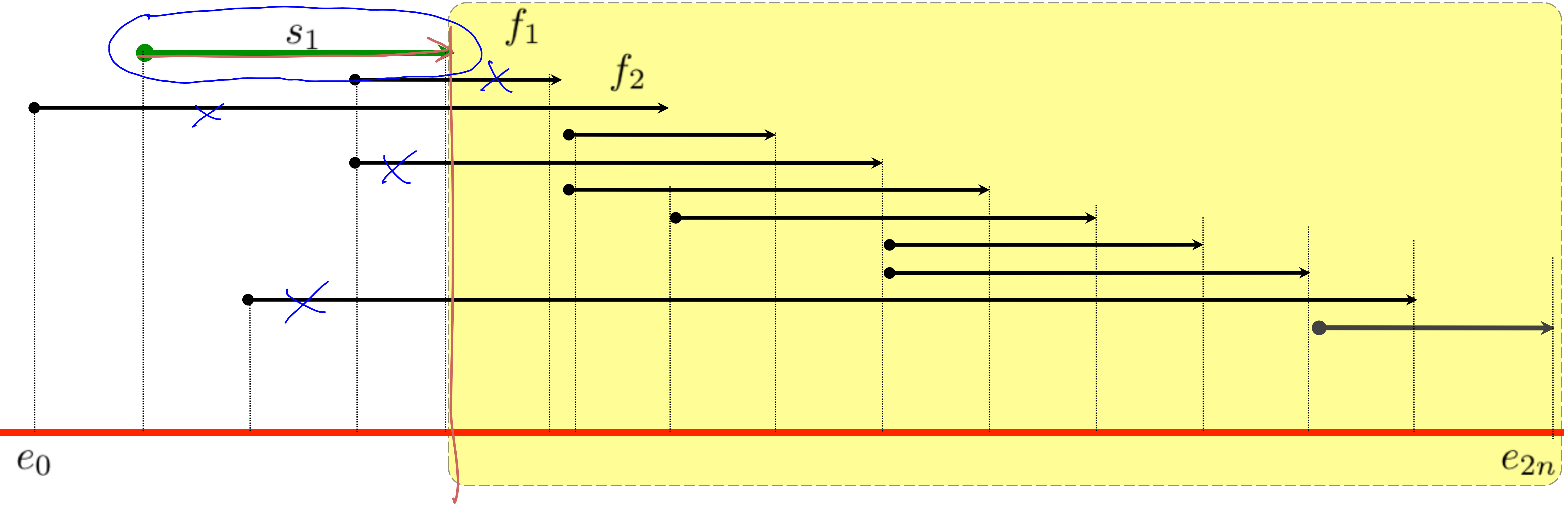
Let a be the first activity in $\text{SOLUTN}_{i,j}$



$\Rightarrow a^*$ is f-to-f, so $f(a^*) < f(a)$

$\Rightarrow \text{SOLUTN}_{i,j} - \{a\} \cup \{a^*\}$ is ALSO optimal.

GREEDY SOLUTION:

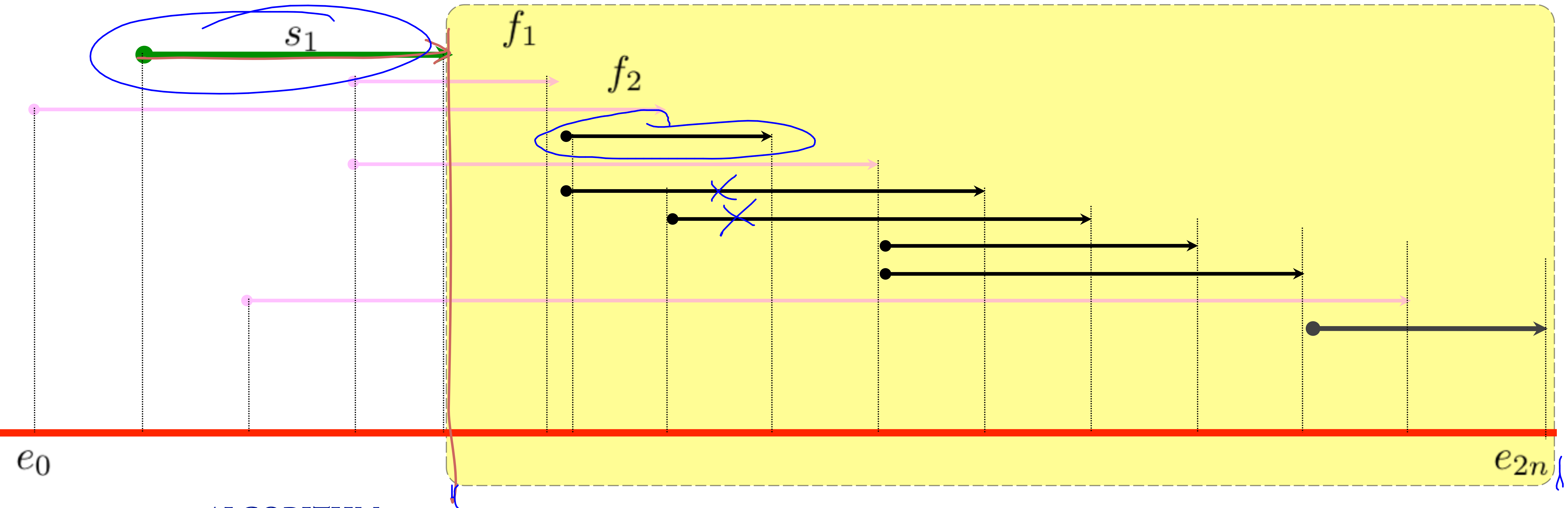


ALGORITHM: FIND FIRST EVENT TO FINISH. ADD TO SOLUTION.

REMOVE CONFLICTING EVENTS.

CONTINUE.

GREEDY SOLUTION:

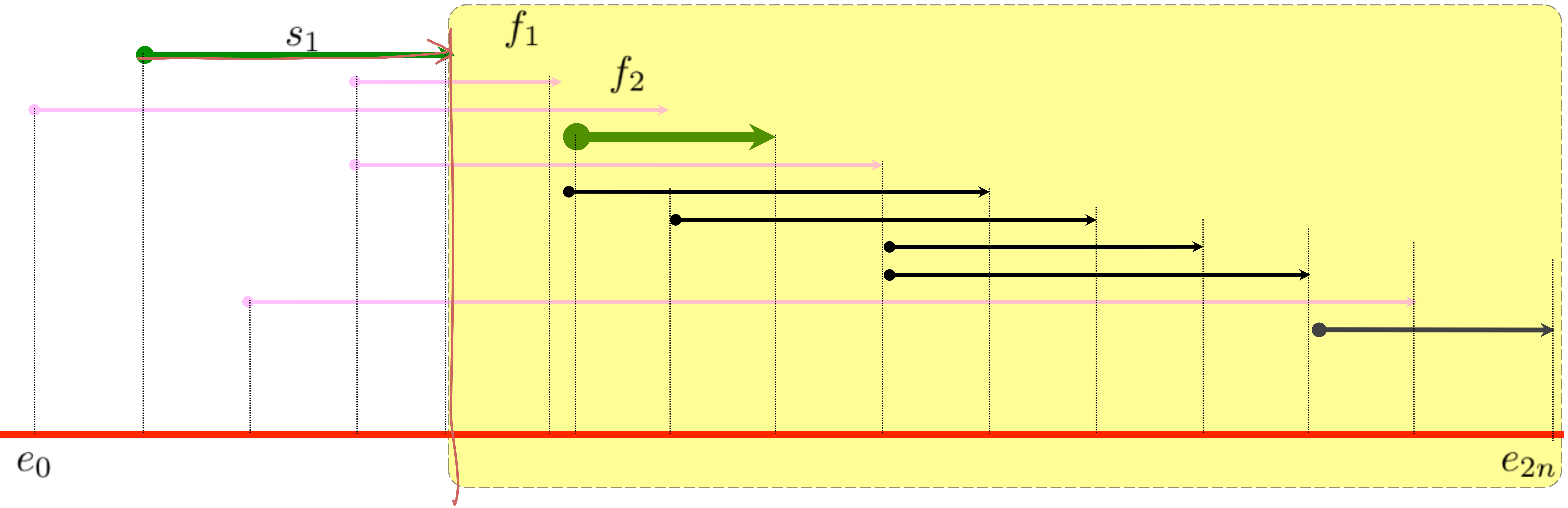


ALGORITHM: FIND FIRST EVENT TO FINISH. ADD TO SOLUTION.

REMOVE CONFLICTING EVENTS.

CONTINUE.

GREEDY SOLUTION:

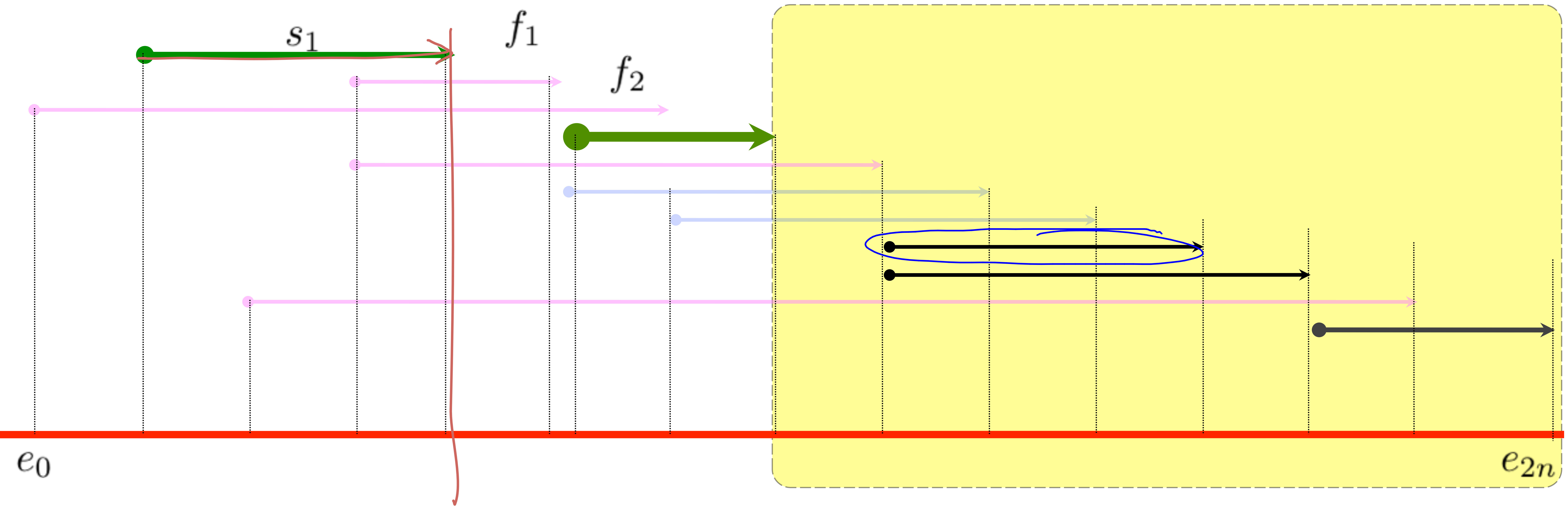


ALGORITHM: FIND FIRST EVENT TO FINISH. ADD TO SOLUTION.

REMOVE CONFLICTING EVENTS.

CONTINUE.

GREEDY SOLUTION:

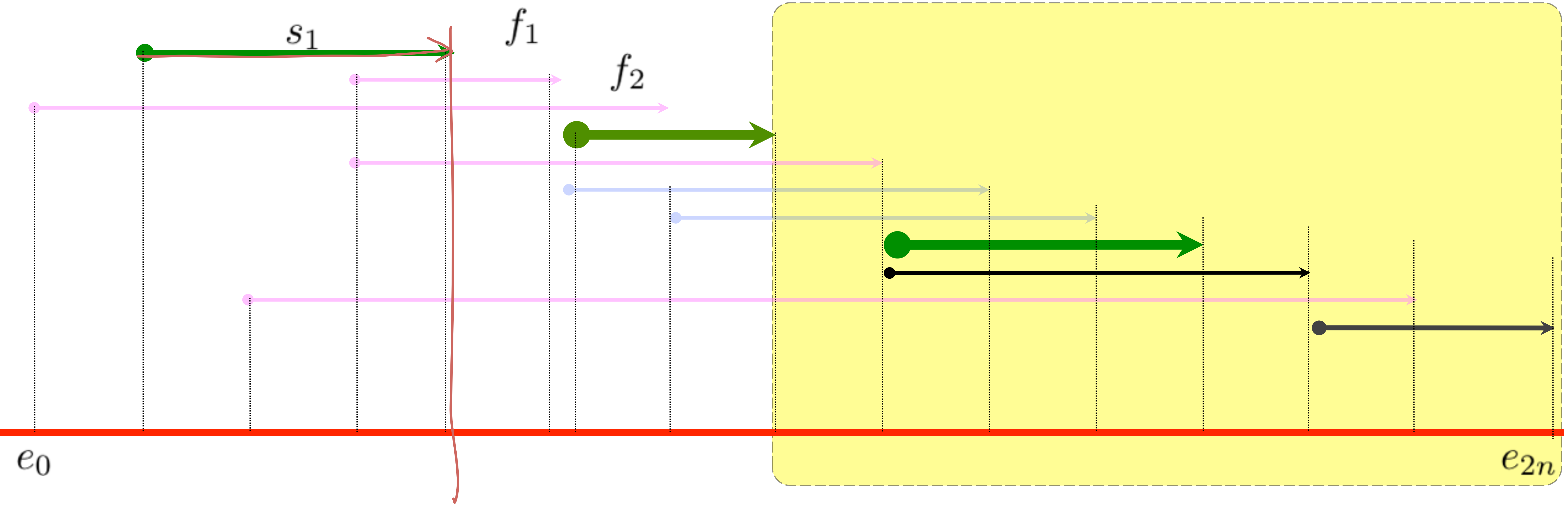


ALGORITHM: FIND FIRST EVENT TO FINISH. ADD TO SOLUTION.

REMOVE CONFLICTING EVENTS.

CONTINUE.

GREEDY SOLUTION:

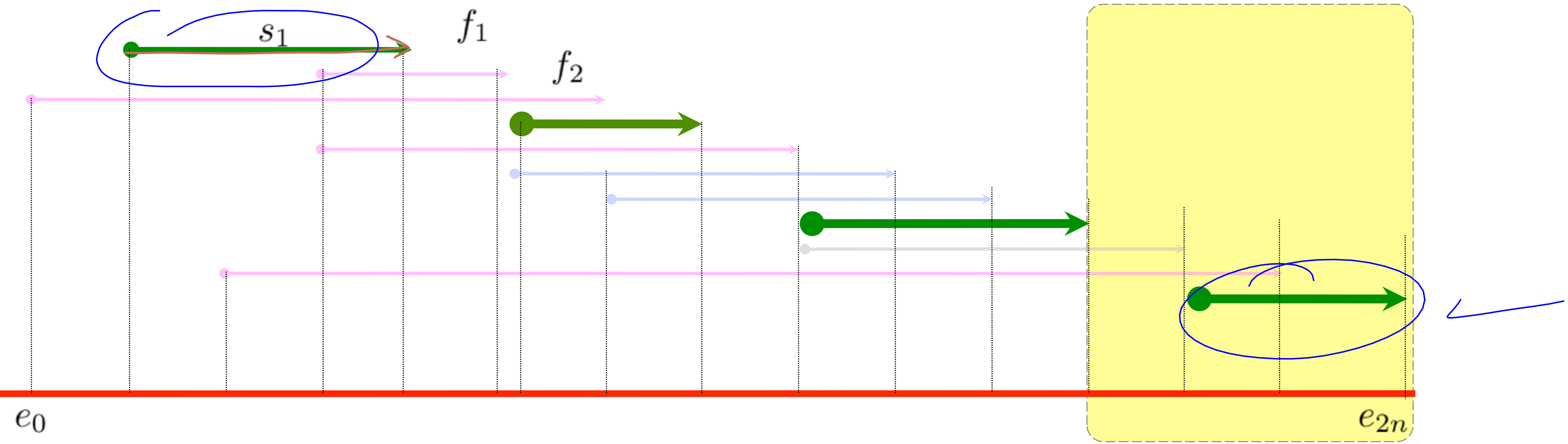


ALGORITHM: FIND FIRST EVENT TO FINISH. ADD TO SOLUTION.

REMOVE CONFLICTING EVENTS.

CONTINUE.

GREEDY SOLUTION:



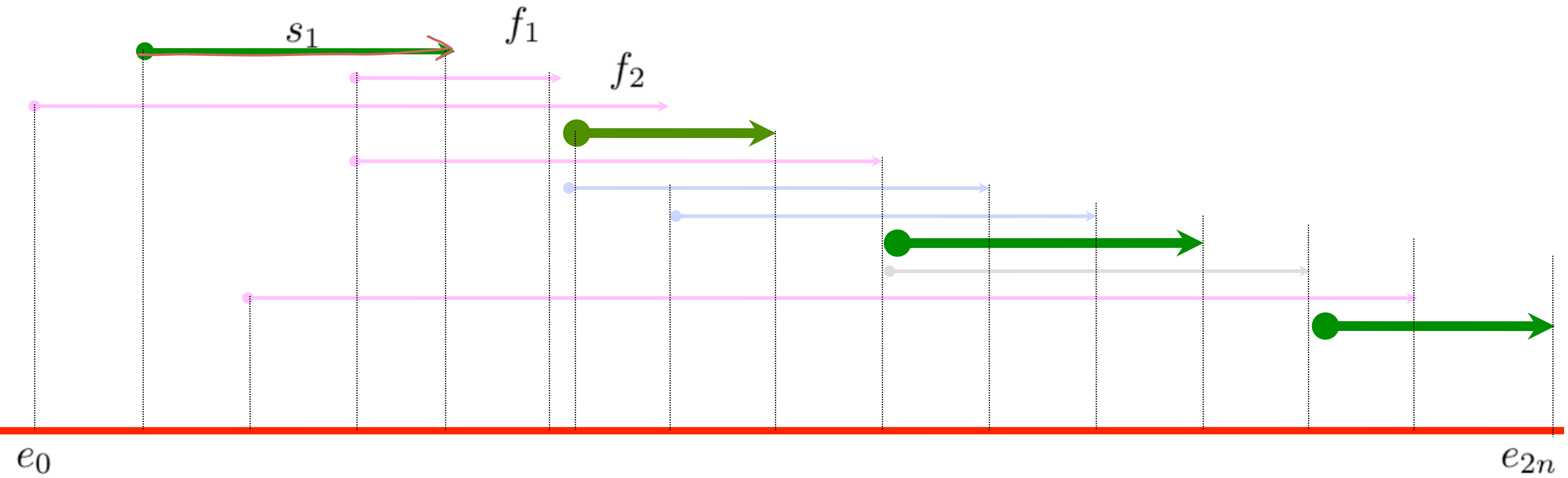
ALGORITHM: FIND FIRST EVENT TO FINISH. ADD TO SOLUTION.

REMOVE CONFLICTING EVENTS.

CONTINUE.

Much simpler algorithm. 1 pass thru the sorted list.

GREEDY SOLUTION:



ALGORITHM: FIND FIRST EVENT TO FINISH. ADD TO SOLUTION.

REMOVE CONFLICTING EVENTS.

CONTINUE.

RUNNING TIME

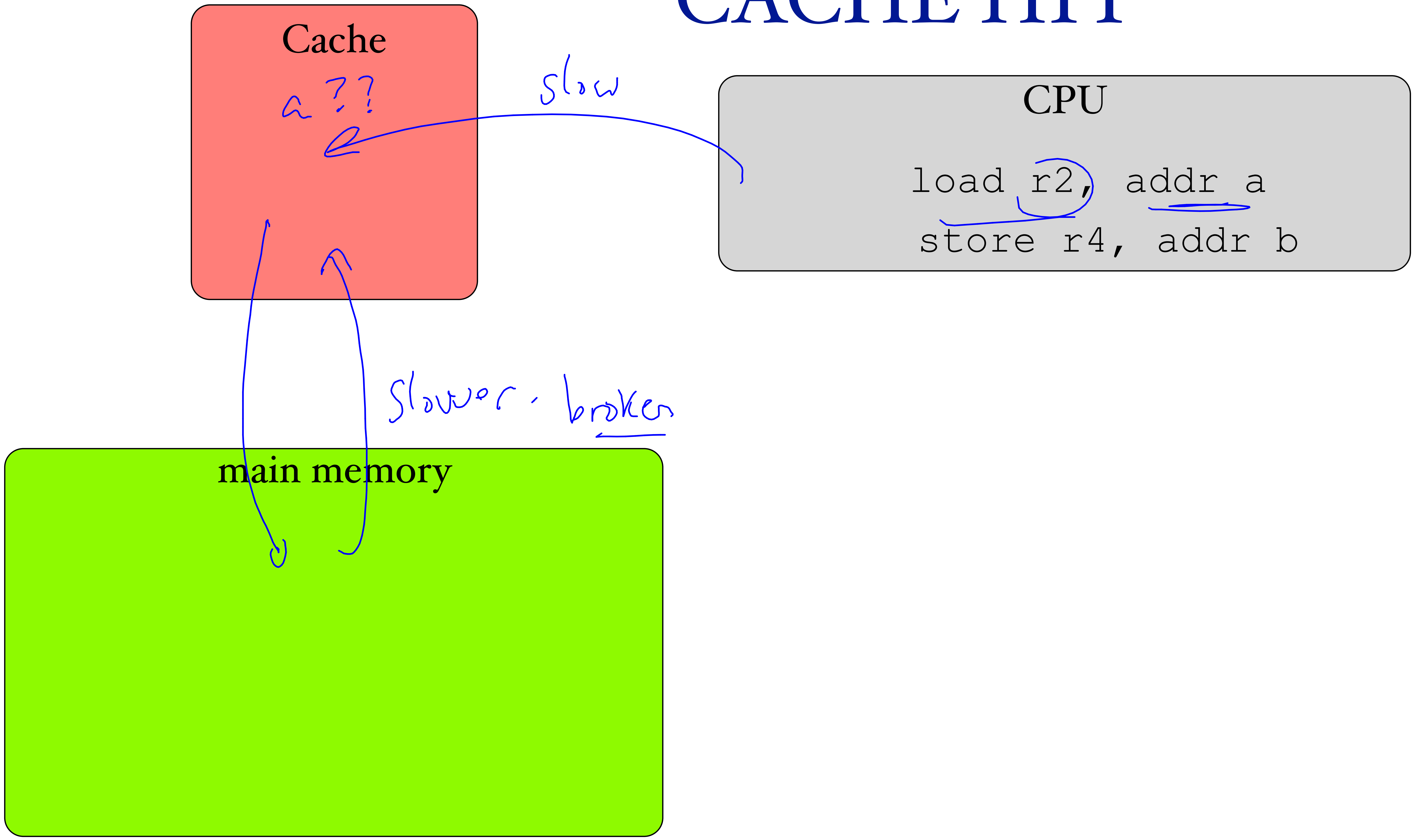
ALGORITHM: FIND FIRST EVENT TO FINISH. ADD TO SOLUTION.

REMOVE CONFLICTING EVENTS.

CONTINUE.

CACHING

CACHE HIT


Cache
a ??
slow
CPU
load r2, addr a
store r4, addr b
slower. broken
main memory

QUESTION:

How to manage the cache ??

① Assumption: Spse we know the entire access sequence ahead of time !!

② cache is fully associative

PROBLEM STATEMENT

input: K · cache size, $d_1, d_2 \dots d_n$ RAM access pattern

output: least # of cache misses. {must satisfy the memory requests}

cache is fully associative

PROBLEM STATEMENT

input: K, the size of the cache

$d_1, d_2, \ldots, d_m$ memory accesses

output: min # of cache misses

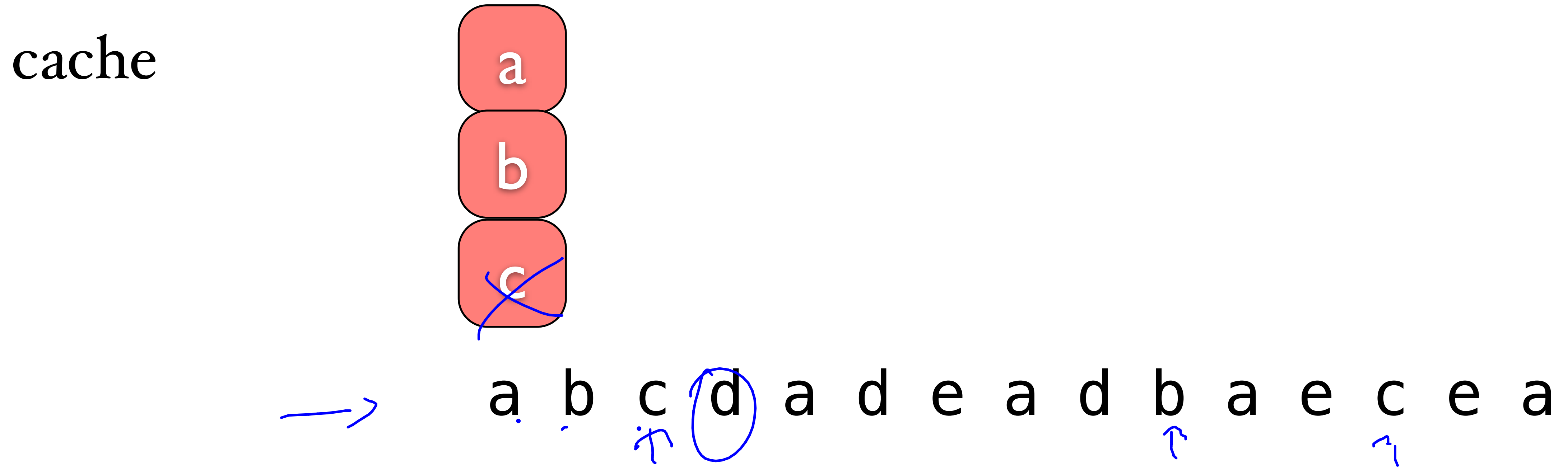
cache is fully associative, line size is 1

CONTRAST WITH REALITY

BELADY EVICT RULE

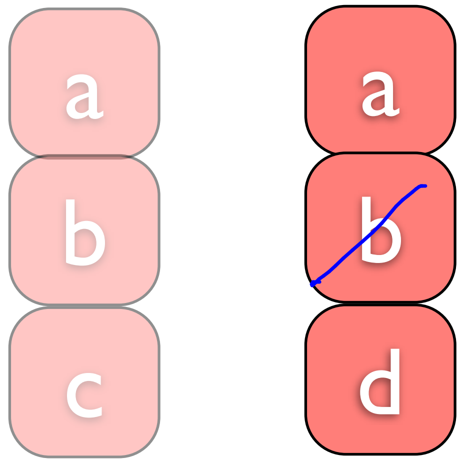
"if you must evict, evict the entry that is accessed farthest-in-the-future"

EXAMPLE



EXAMPLE

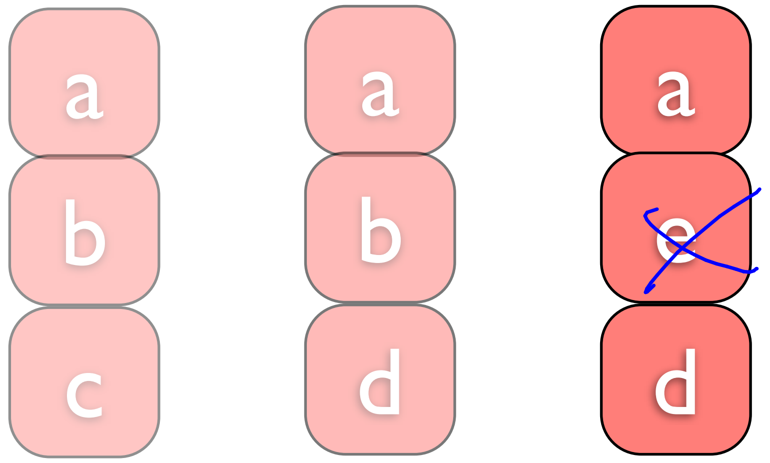
cache



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

EXAMPLE

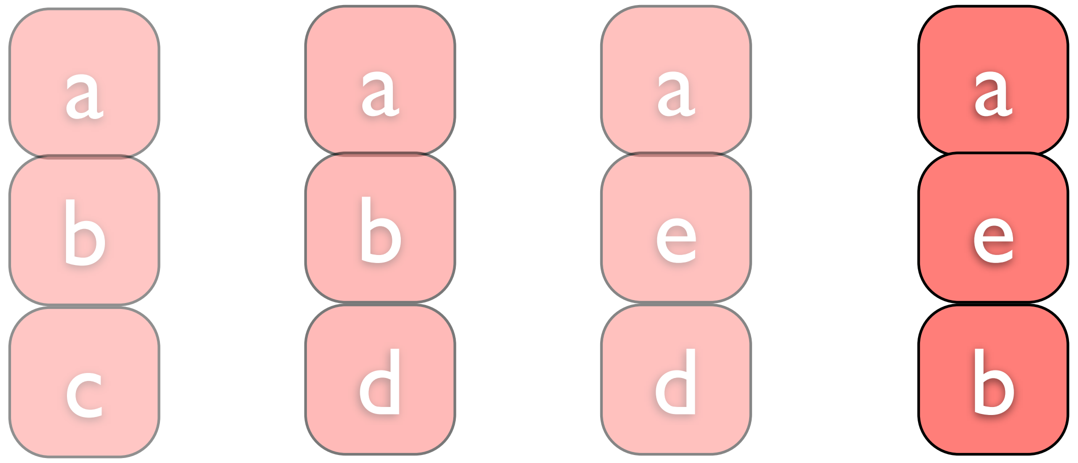
cache



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

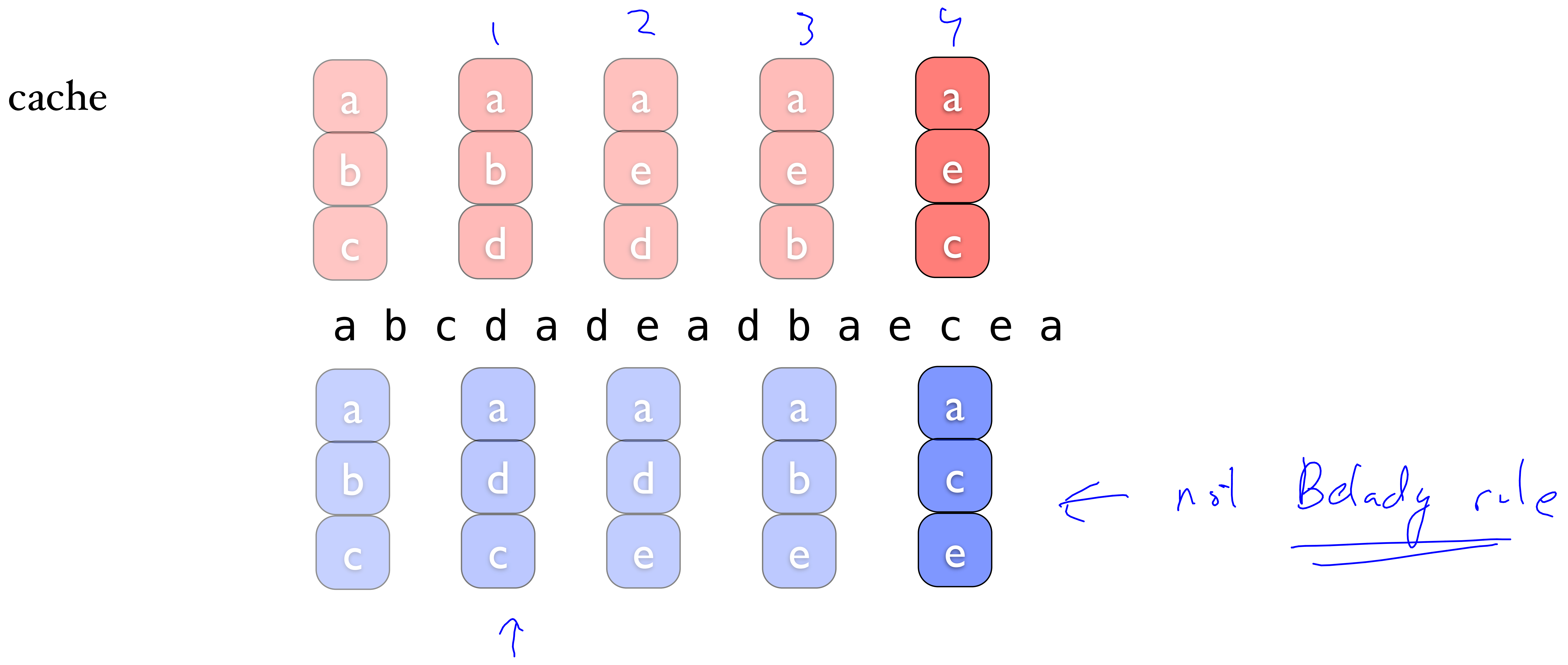
EXAMPLE

cache



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

EXAMPLE


cache
1
2
3
4
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
← not Belady rule

SURPRISING THEOREM

Thm: Belady rule is optimal.

SCHEDULE

Schedule for access pattern d1,d2,...,dn: is a sequence of "nop" or "evict x for y" for each operation

Reduced schedule: Schedule in which "evict x for y" occurs @ operation i only if $d_i = y$.

(lazy schedule)

$$\text{misses}(\text{schedule } S) \geq \text{misses}(\text{reduced}(S))$$

REDUCED SCHEDULE

Def:

EXCHANGE LEMMA

(schedule induced by the Belady)

- Spse some reduced schedule S agrees with S_{ff} for the first j operations.

Then $\exists$ a reduced schedule S' that agrees with S_{ff} on $j+1$ operations &

$$\text{misses}(S') \leq \text{misses}(S).$$

Exchange Lemma:

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

$$S^* \xrightarrow{} S_1 \xrightarrow{\text{lemma}} S_2 \xrightarrow{\text{lemma}} S_{3..n} \xrightarrow{\text{Lemma}} S_{ff}$$

↑ optimal schedule reduced

apply our lemma

misses(S^*) = misses(S_1)

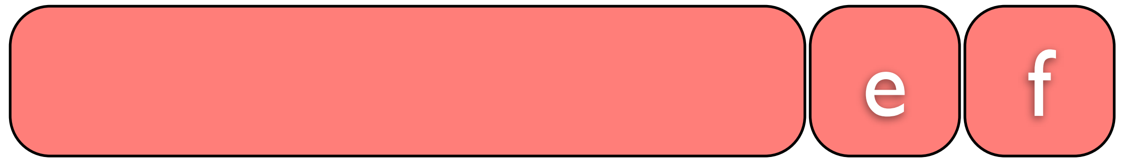
&

S_{ff} agrees w/ S_1 on 1 operation

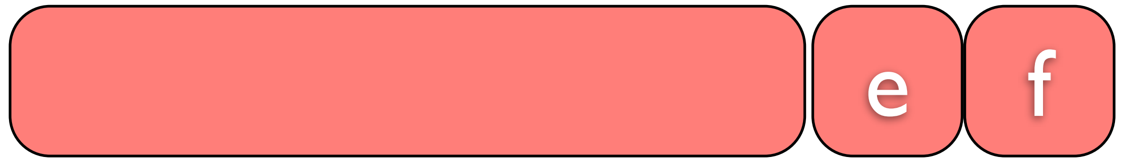
LEMMA

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

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



S



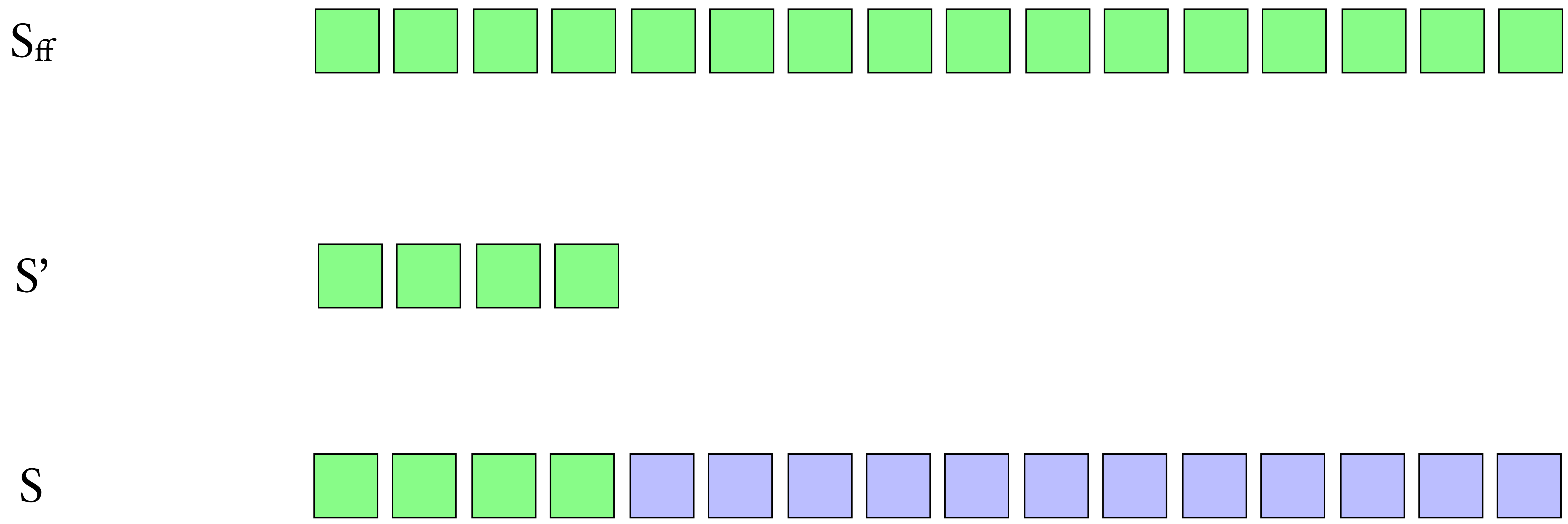
S_{ff}

easy case 1

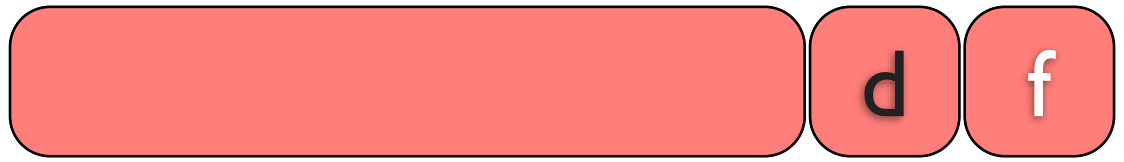
easy case 2

case 3

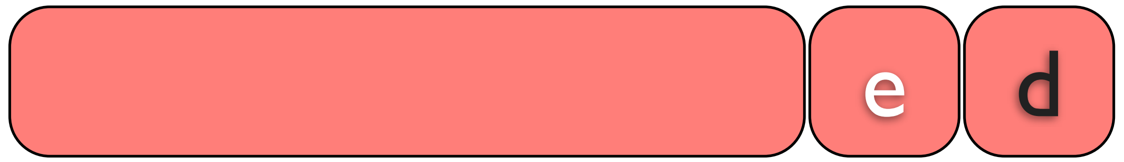
Timeline



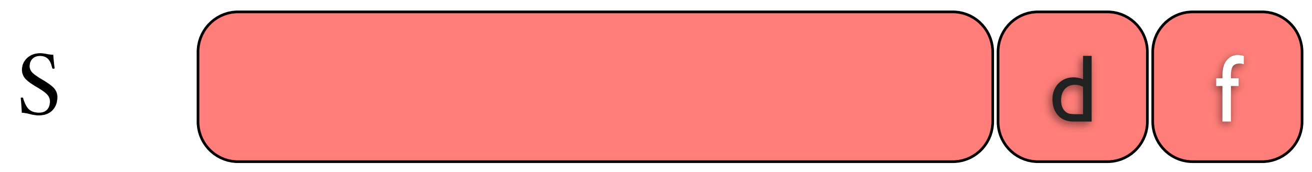
S

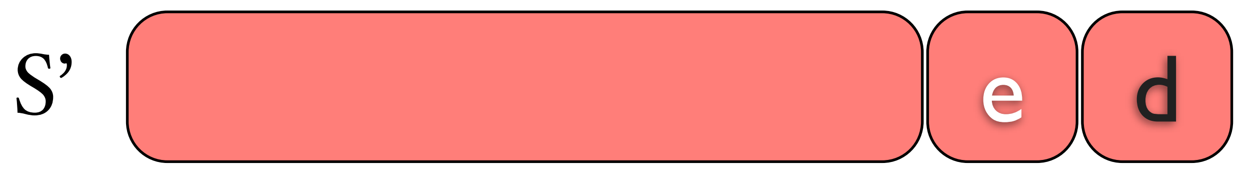


S’



Let access t

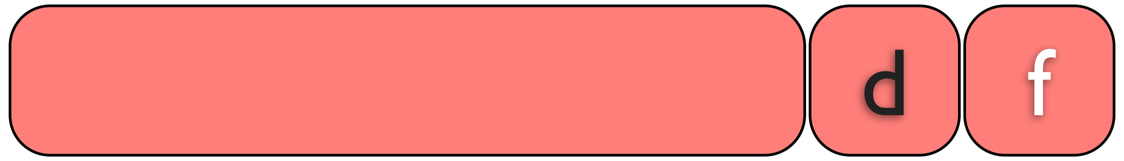




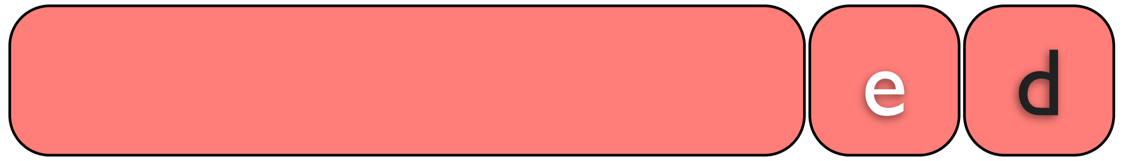
what if g=e ?

what if g=f ?

S

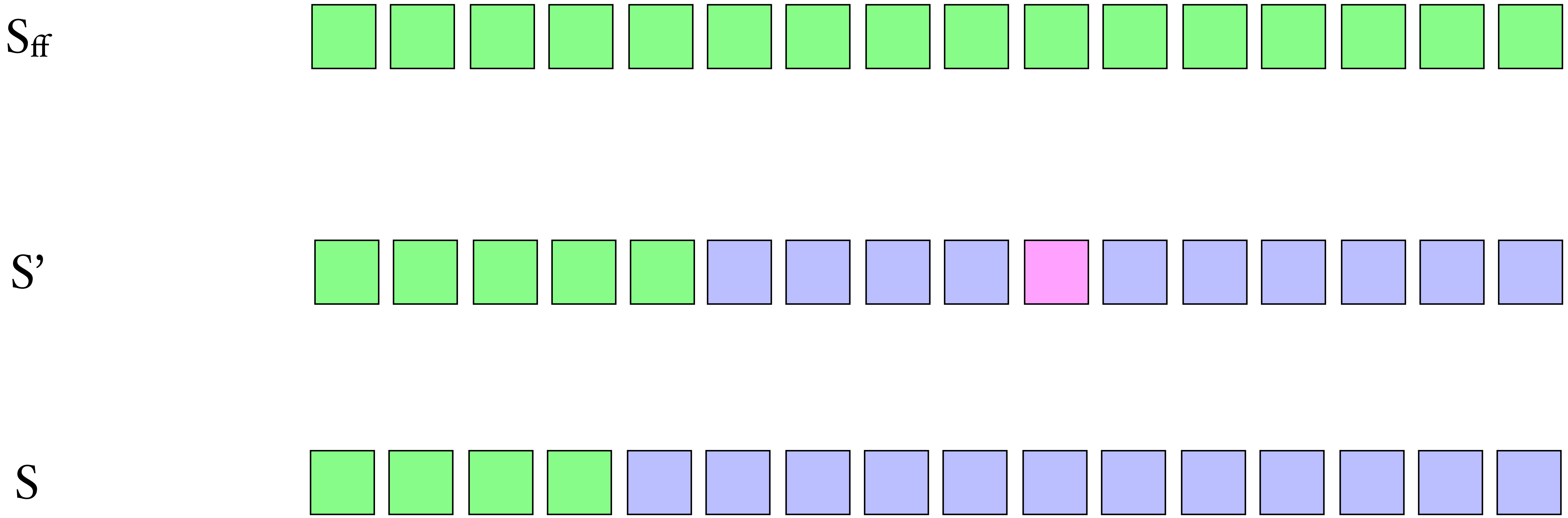


S’



what if g is neither e nor f ?

WHAT HAVE WE SHOWN



S^* S_{ff}