L10 5800

feb 18/21 2022

shelat

Greedy is only good for certain problems

	start	end
sy3333	2	3.25
en1612	1	4
ma1231	3	4
Cs5800	3.5	4.75
cs4800	4	5.25
cs6051	4.5	6
sy3100	5	6.5
Cs1234	7	8

How many non-overlapping courses can you take?

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

(compatible)

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

(compatible)

For any two activities $a_i, a_j, i < j$ the start time of a_j is after the finish time of a_i.

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

(compatible)

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

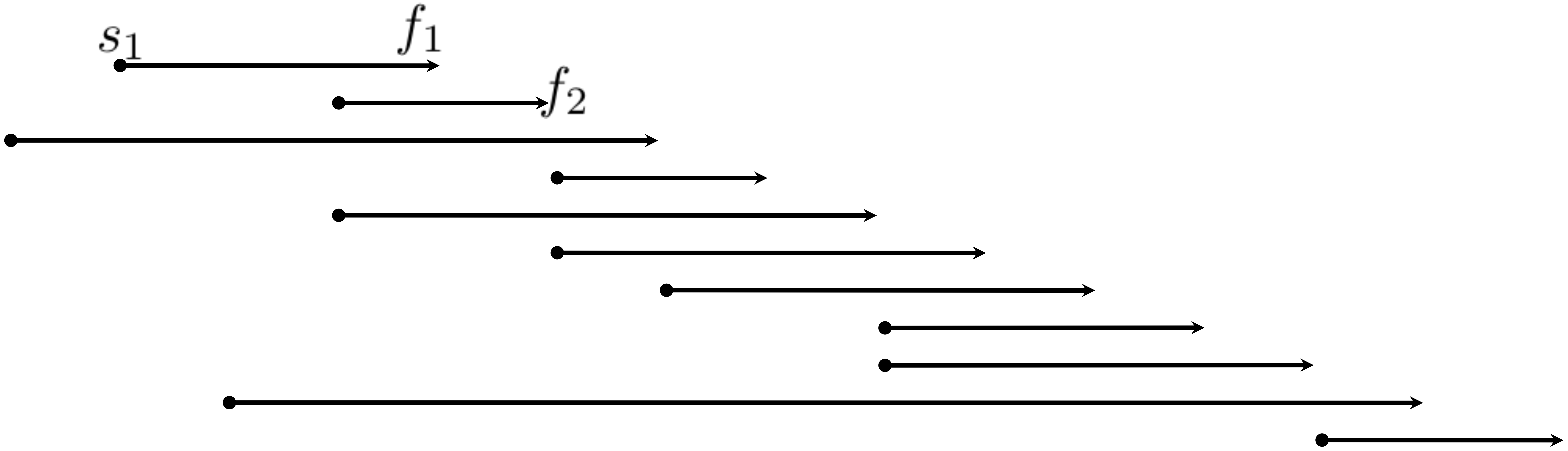
$$f_i \leq s_j$$

problem statement

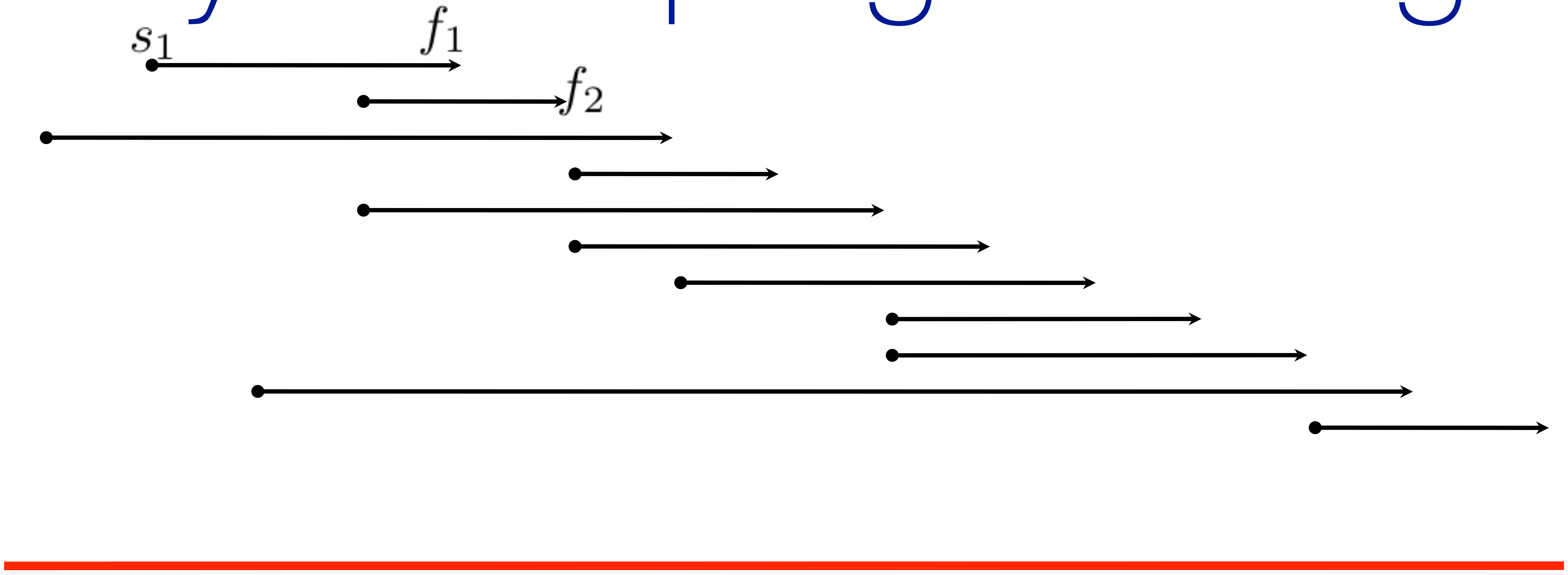
$$(a_1, \ldots, a_n)$$

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

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

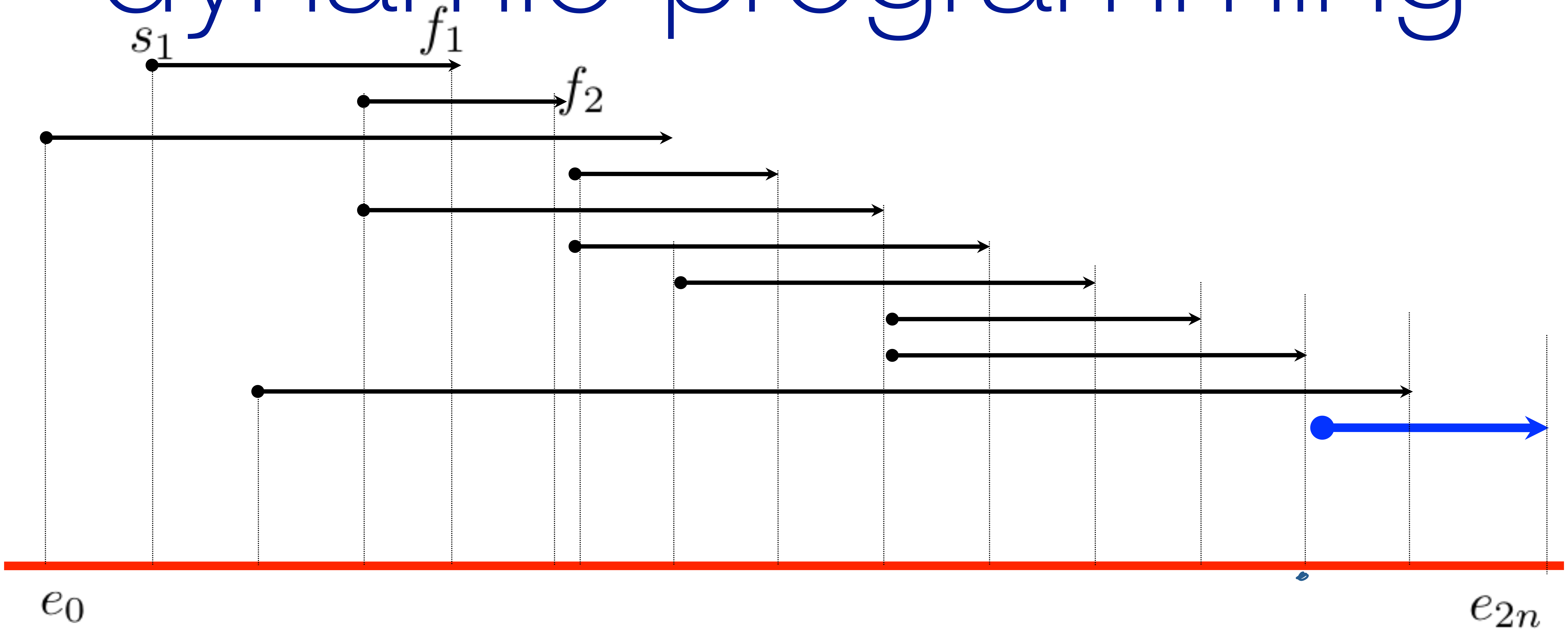


dynamic programming



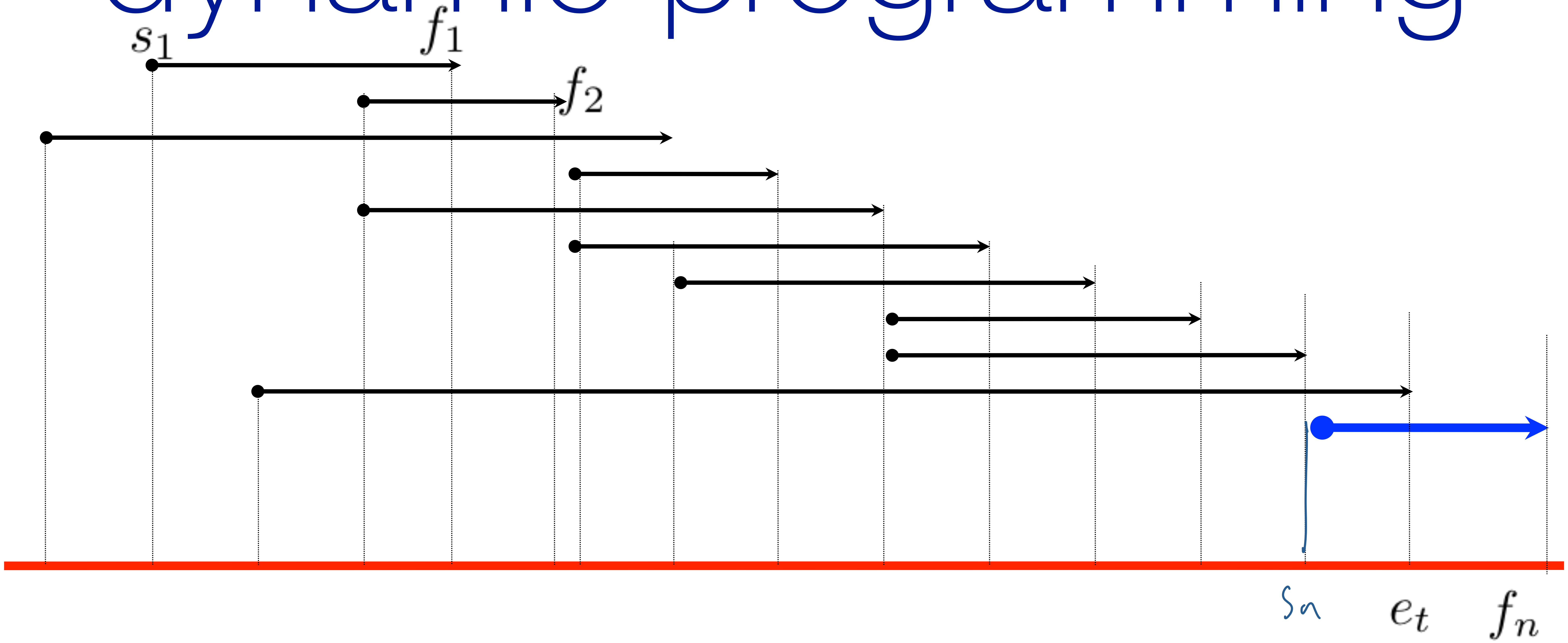
Lets draw all of the events on a timeline.

dynamic programming



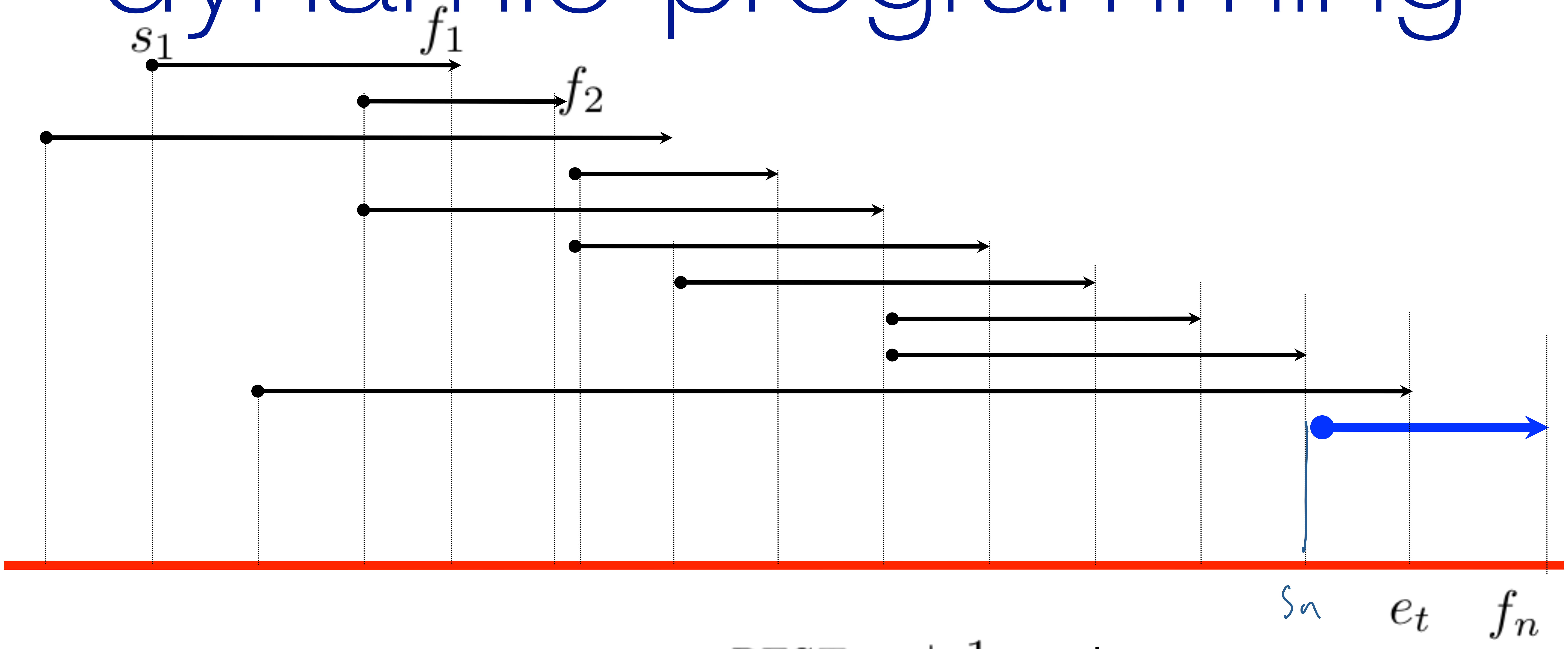
$Best_{2n} =$ Maximum number of non-overlapping activities possible among the first 2n events.

dynamic programming



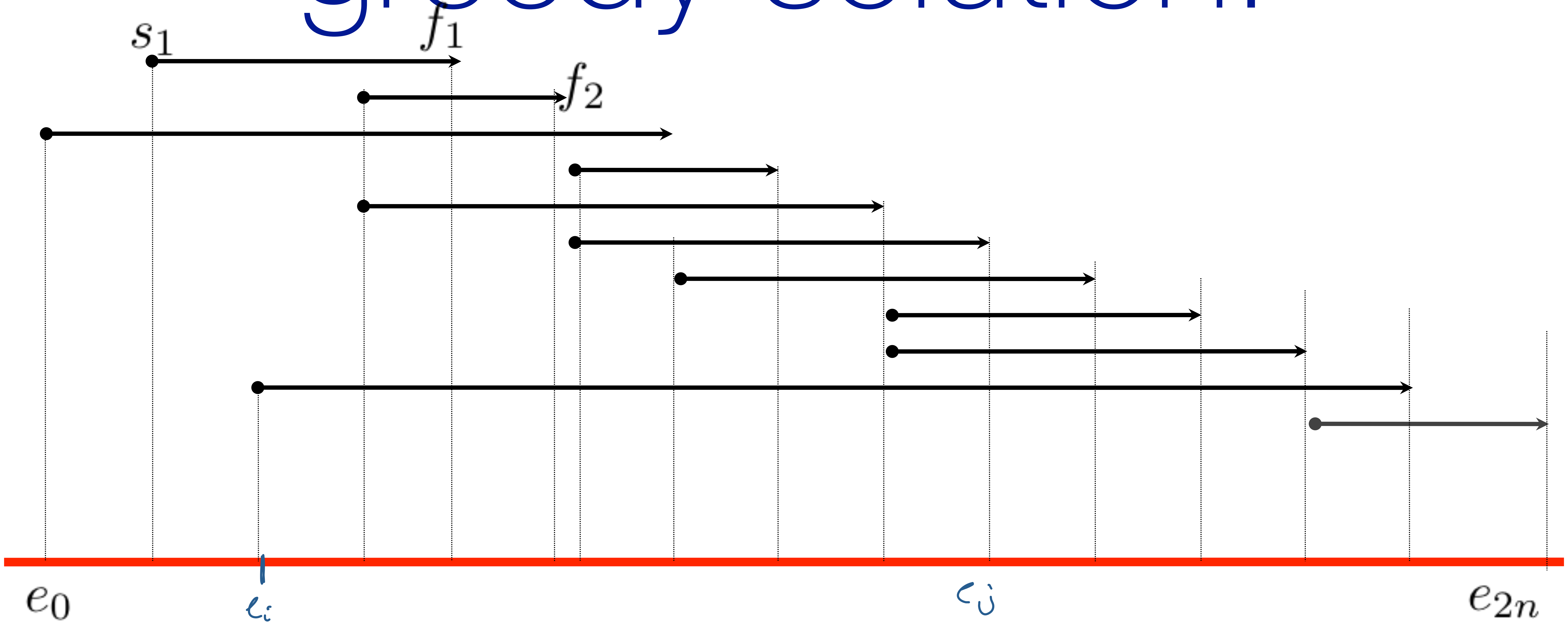
$\mathrm{BEST}_{f_n} =$

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:

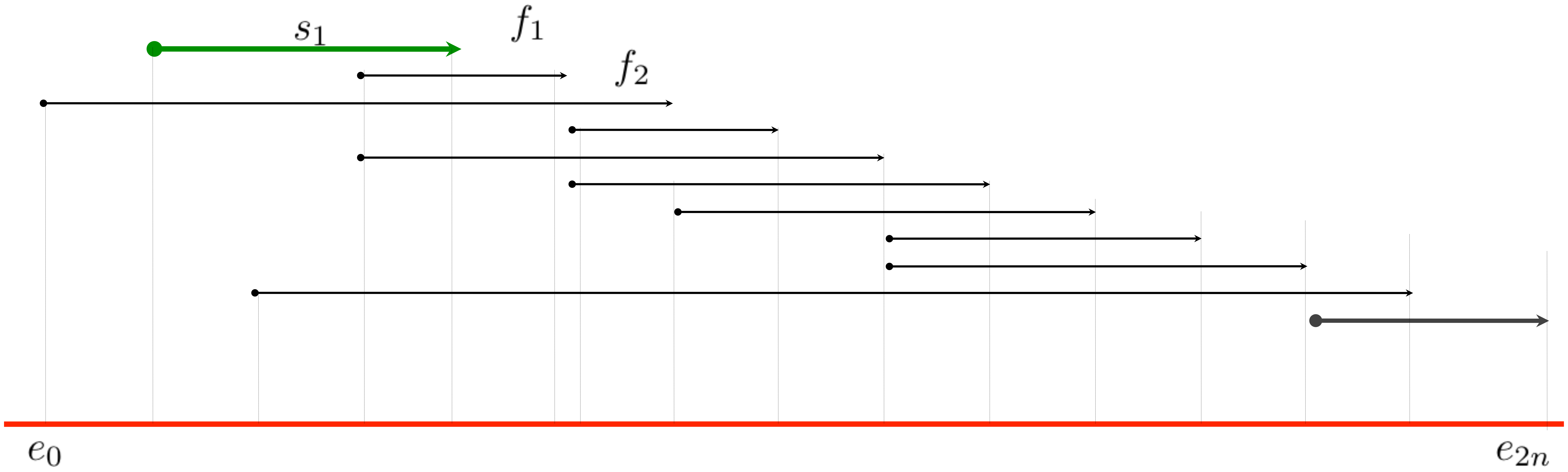


DEFINITION:

$$\text{soltn}_{i,j} =$$

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

greedy solution:



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

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

PROOF:

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

PROOF:

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

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

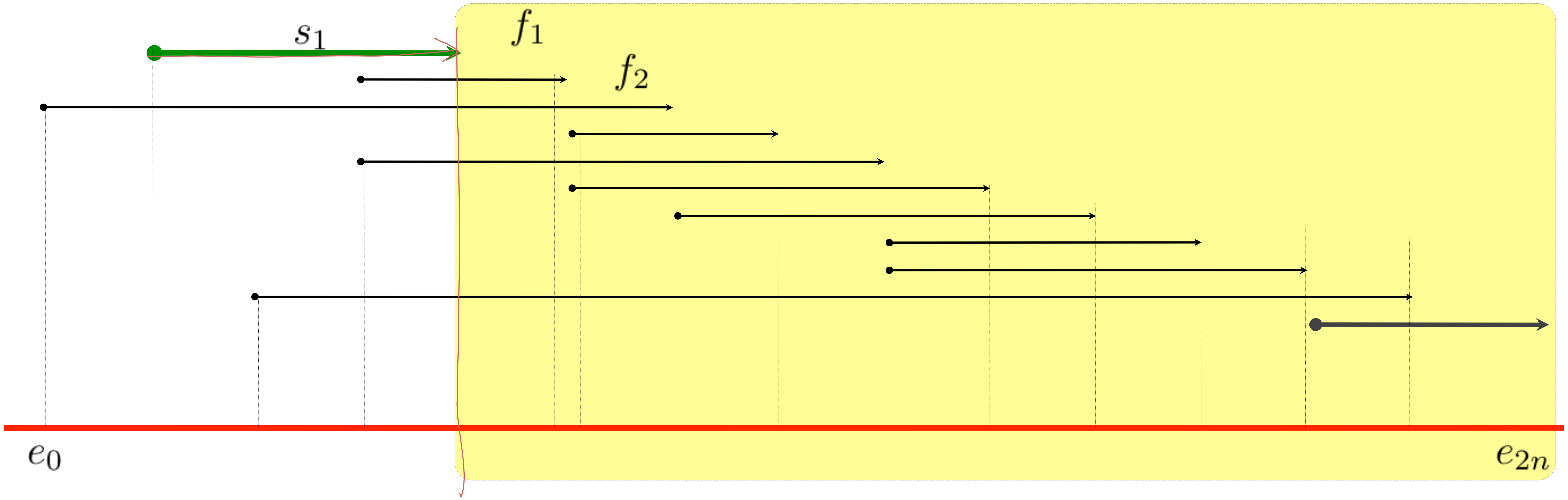
If not, let a be the activity that finishes first in $\text{soltn}_{i,j}$.

Consider a new solution that replaces a with a^*.

$$\text{soltn}^*_{i,j} = \text{soltn}_{i,j} - \{a\} \cup \{a^*\}$$

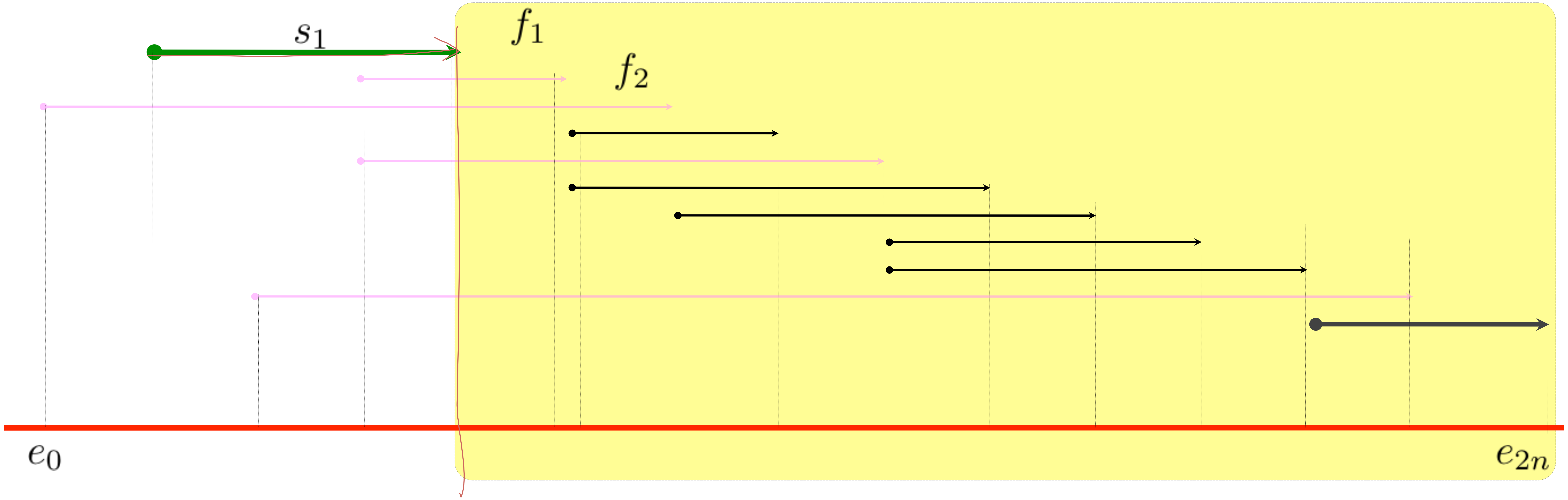
This new set is valid because a^* finishes before a and thus does not overlap with any activities. This new solution also has the same size and is therefore also optimal too.

greedy solution:



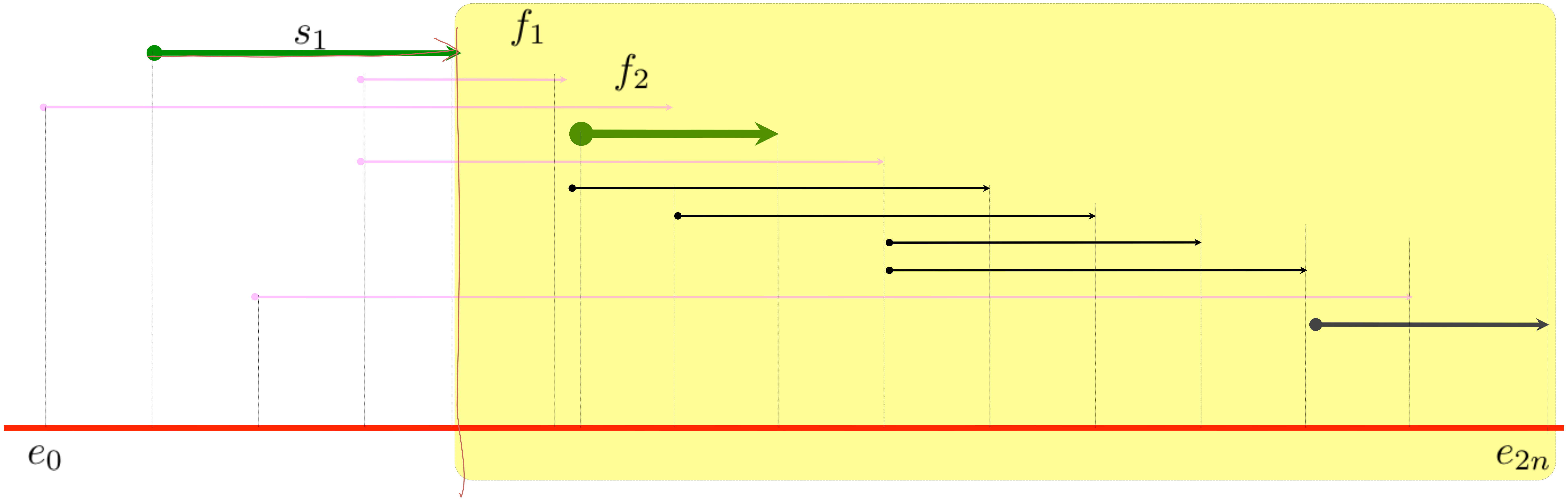
algorithm: find first event to finish. add to solution.
remove conflicting events.
continue.

greedy solution:



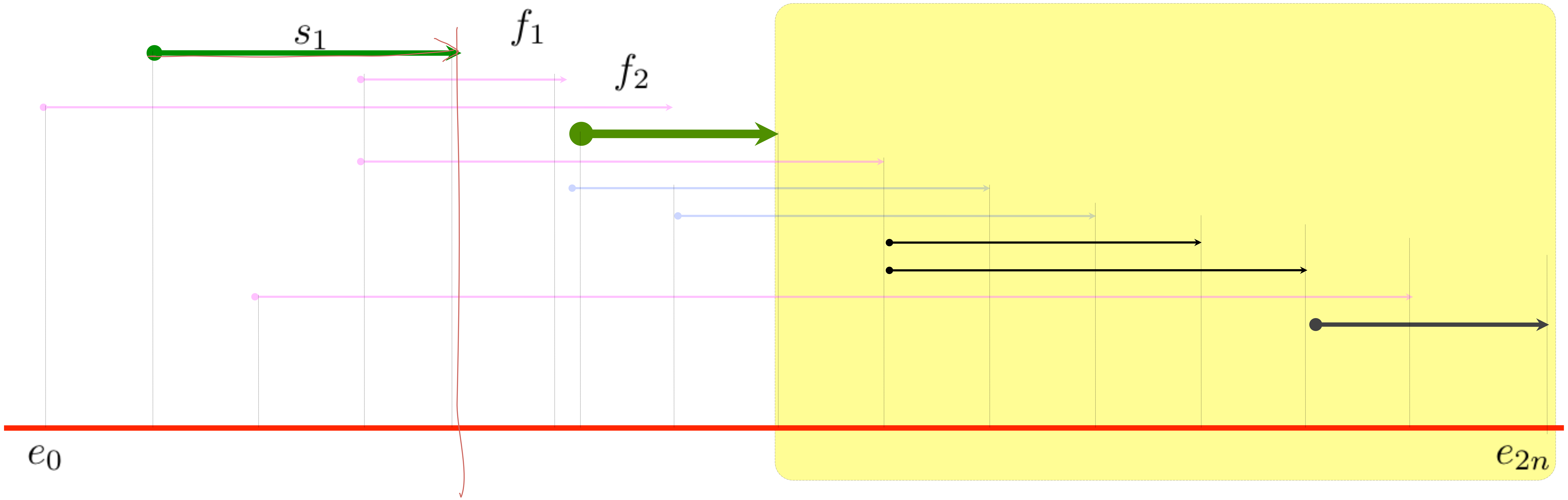
algorithm: find first event to finish. add to solution.
remove conflicting events.
continue.

greedy solution:



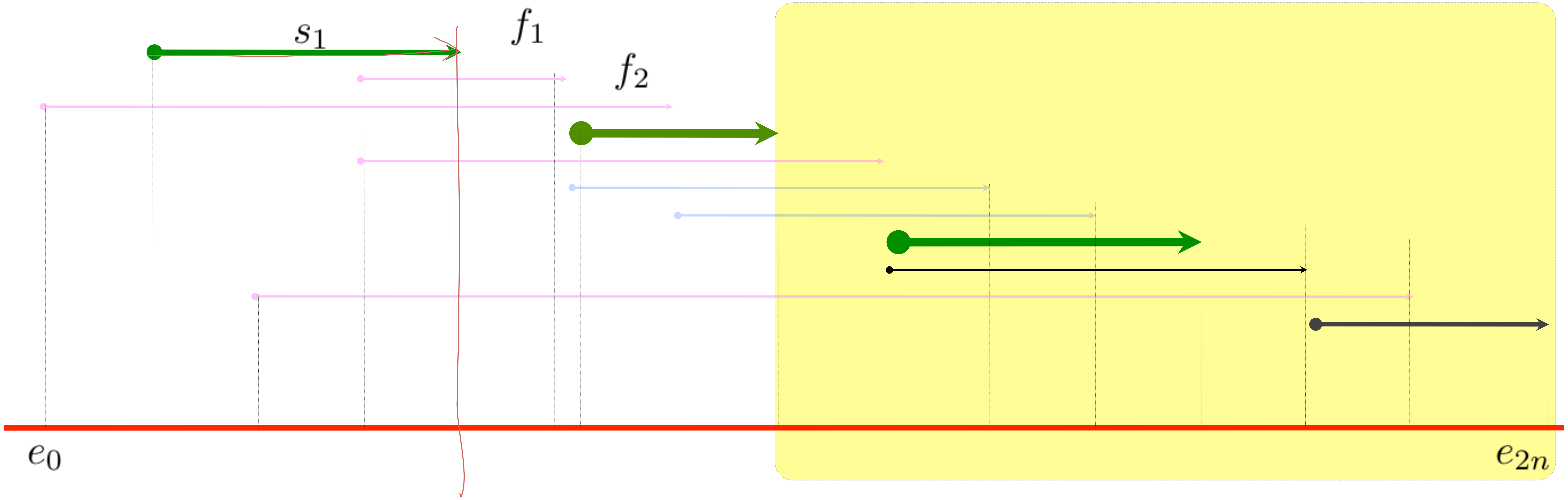
algorithm: find first event to finish. add to solution.
remove conflicting events.
continue.

greedy solution:



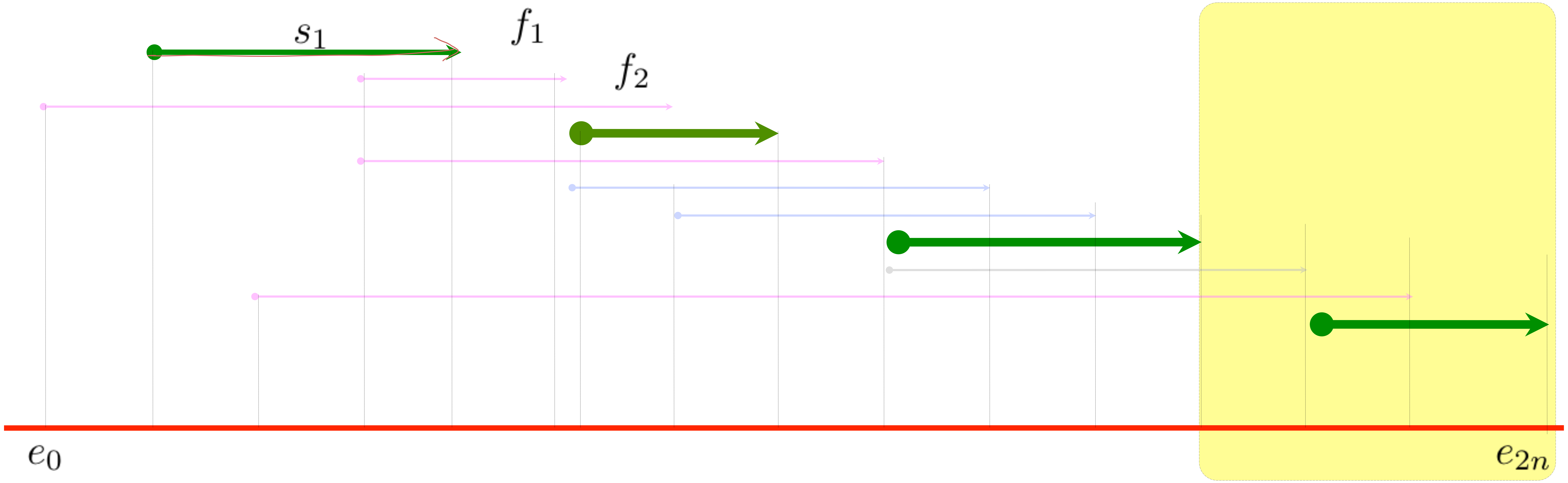
algorithm: find first event to finish. add to solution.
remove conflicting events.
continue.

greedy solution:



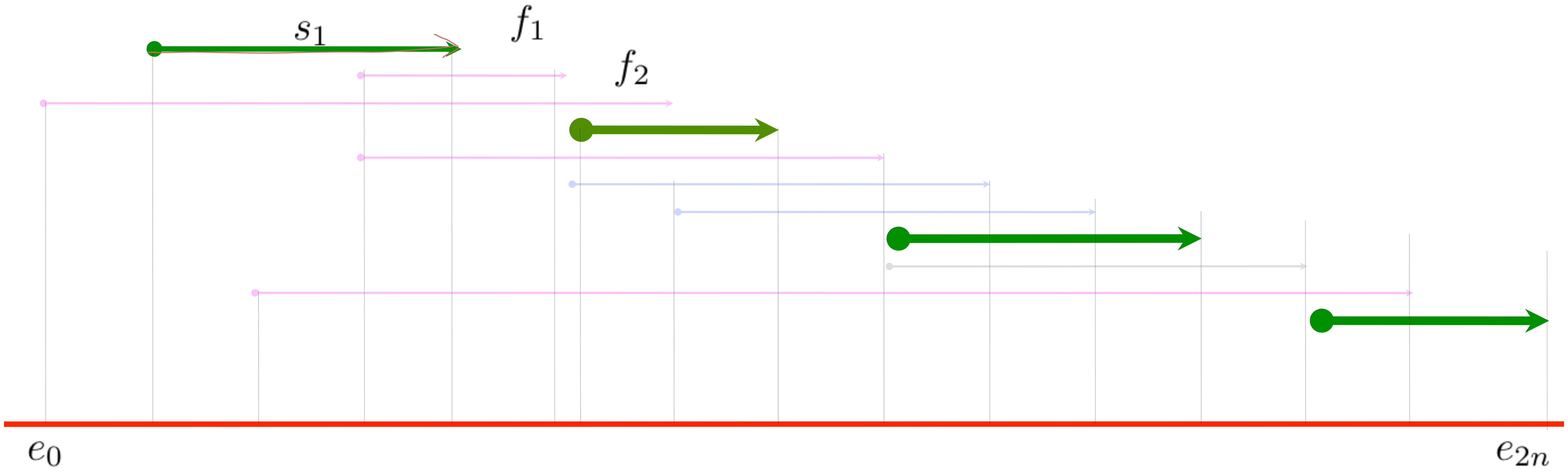
algorithm: find first event to finish. add to solution.
remove conflicting events.
continue.

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.

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

Recap

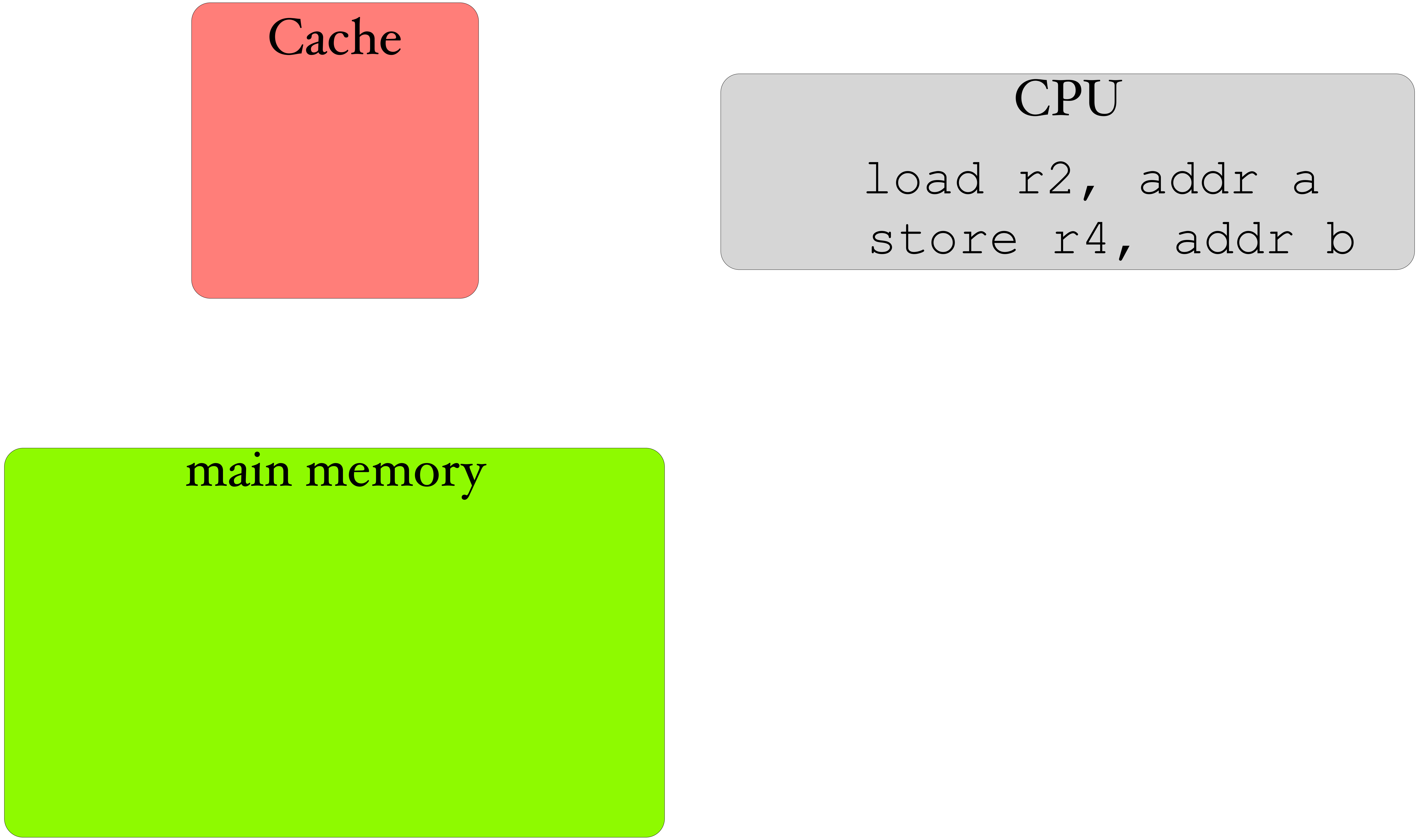
The main idea in this algorithm was the "exchange argument."

We were able to identify an item (first to finish) that must be part of *some* optimal solution by exchanging this element with one that we can identify in any optimal solution.

Since its easy to identify the item that is first to finish, our algorithm is conversely simple, "greedy."

caching

cache hit



question:

question:

How do we manage a fully-associate cache?

When it is full, which element do we replace?

problem statement

input:

output:

cache is

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

contrast with reality

contrast with reality

In a real situation, we may not know the future memory access patterns.

Some caches have additional restrictions, like line-size, associativity, etc.

However, this algorithm can still be used to compare a real-world algorithm against the optimum cache miss rate possible.

Belady eviction rule

Belady eviction rule

Replace the element in the cache that is accessed "farthest into the future"

example

cache

a

b

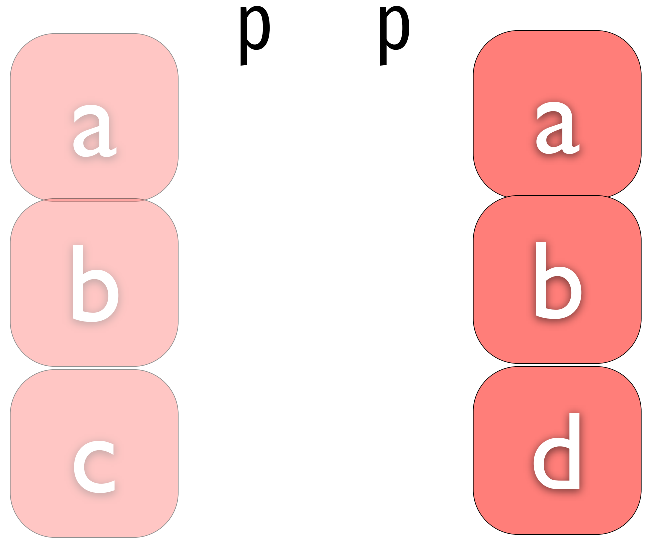
c

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

example

Cache operations: nop n o p n o p Evict c for d.

cache

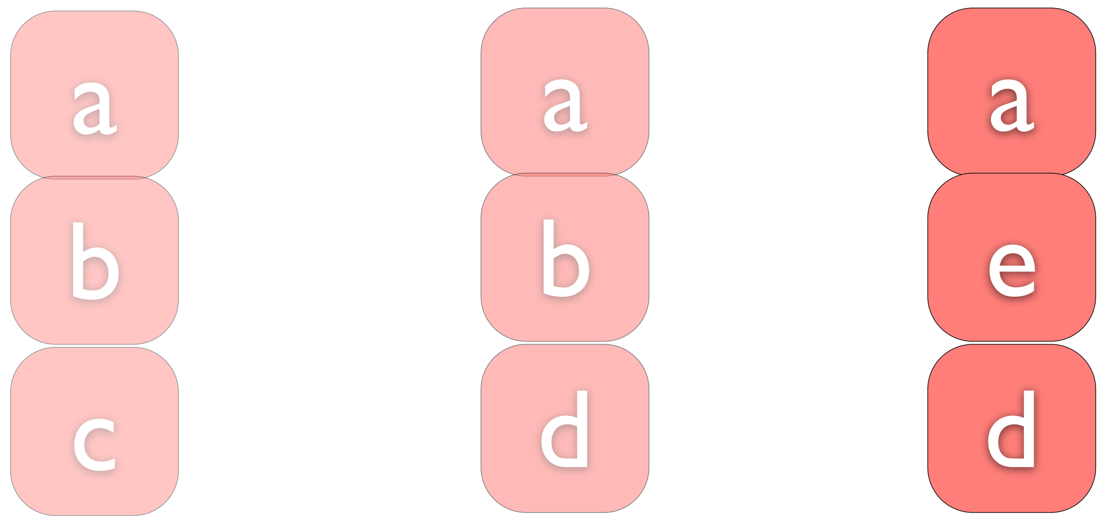


Memory accesses: a b c d a d e a d b a e c e a

example

Cache operations: nop Evict (c,d) Evict (b,e)

cache



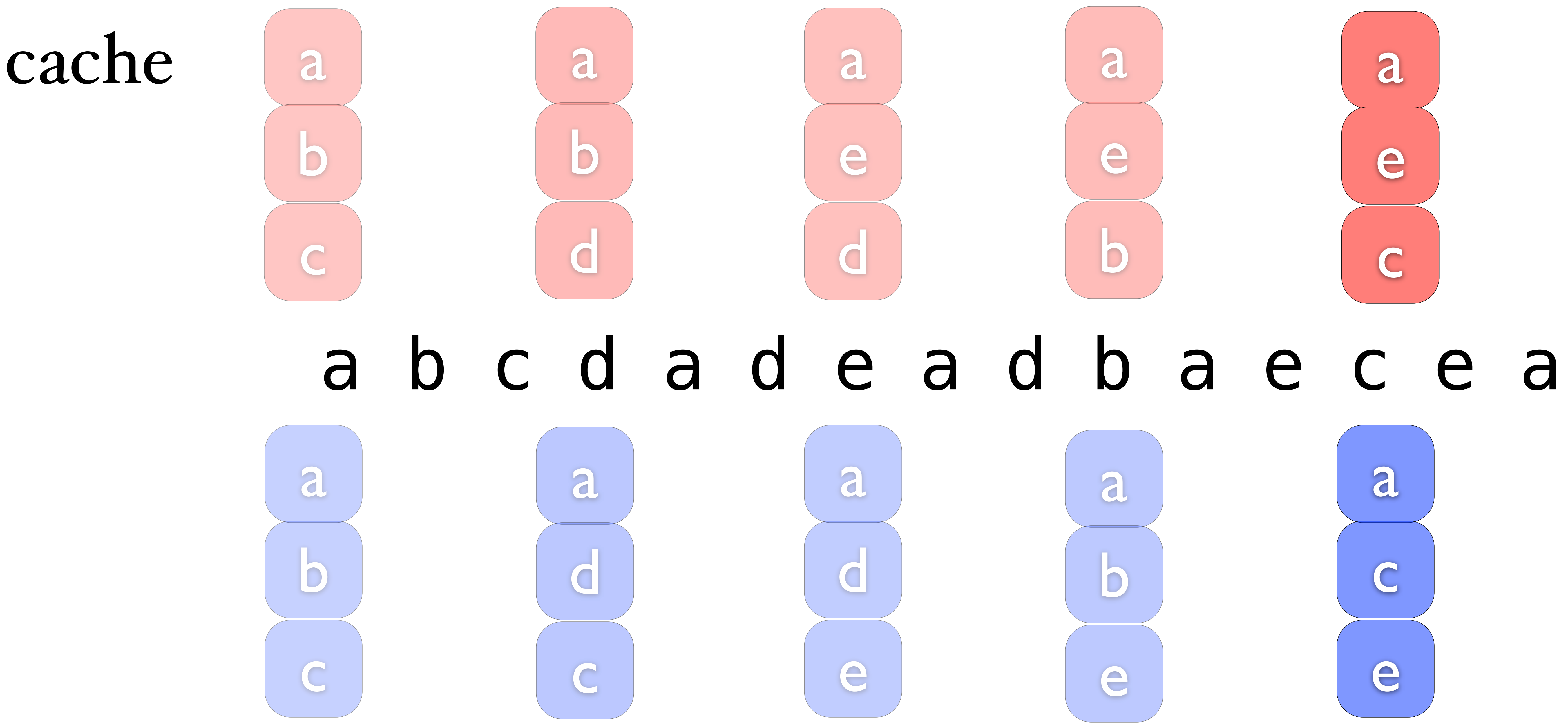
Memory accesses: a b c d a d e a d b a e c e a

example

Cache operations:	nop	Evict (c,d)	Evict (b,e)	Evict (d,b)
cache	a	a	a	a
	b	b	e	e
	c	d	d	b

Memory accesses: a b c d a d e a d b a e c e a

example



Here is an alternate optimal set of cache operations.

Surprising theorem

Surprising theorem

The schedule S_{ff} produced by the Belady “farthest in the future” eviction rule is optimal.

schedule

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

Reduced schedule:

schedule

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

A list of instructions for each access that is either “NOP” or “evict x for y”

Reduced schedule:

schedule

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

A list of instructions for each access that is either "NOP" or "evict x for y"

Reduced schedule:

A schedule in which"evict x for y" instruction only occurs when y is accessed.

schedule

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

A list of instructions for each access that is either "NOP" or "evict x for y"

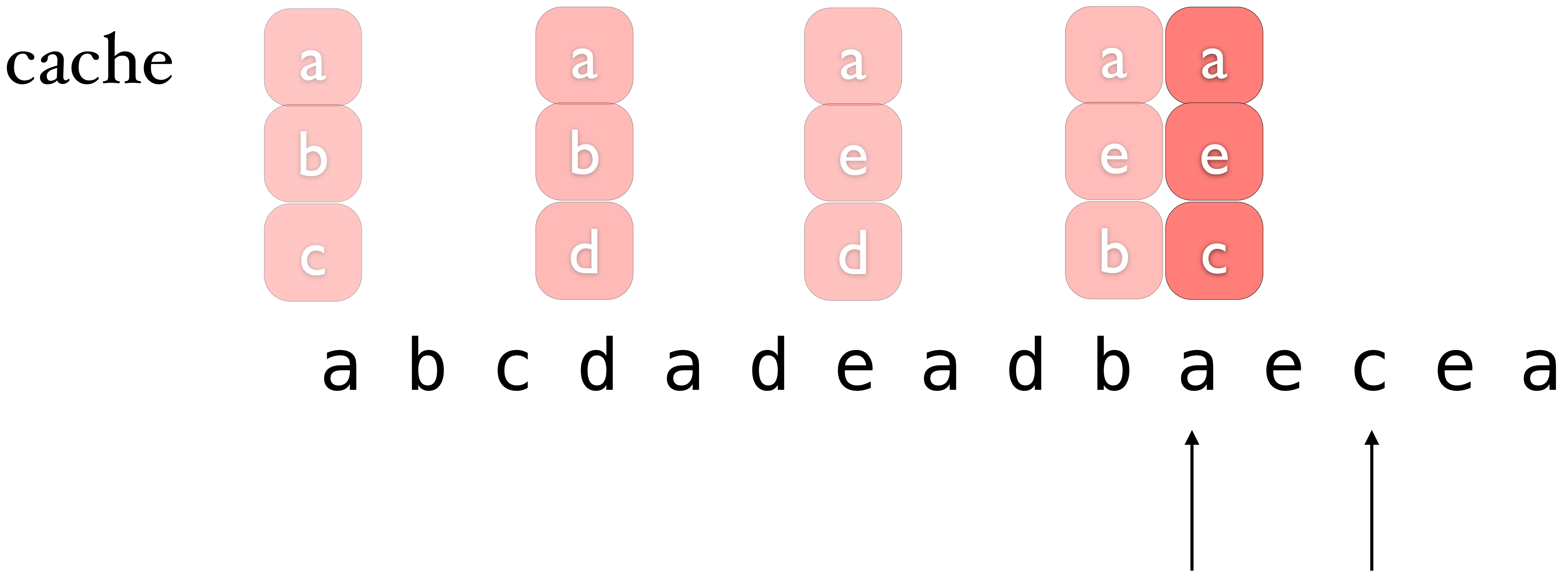
Reduced schedule:

A schedule in which"evict x for y" instruction only occurs when y is accessed.

Note: any schedule can be transformed into a reduced schedule with the same or fewer cache misses.

(Idea: starting at the end, defer "evict...t" until y is read)

Non-Reduced Schedule example



Example of a non-reduced schedule.
At this point, the cache evicts (b,c) when “a” is being accessed. It is possible to delay this eviction until “c” is accessed, thereby leading to a reduced schedule.

Exchange lemma

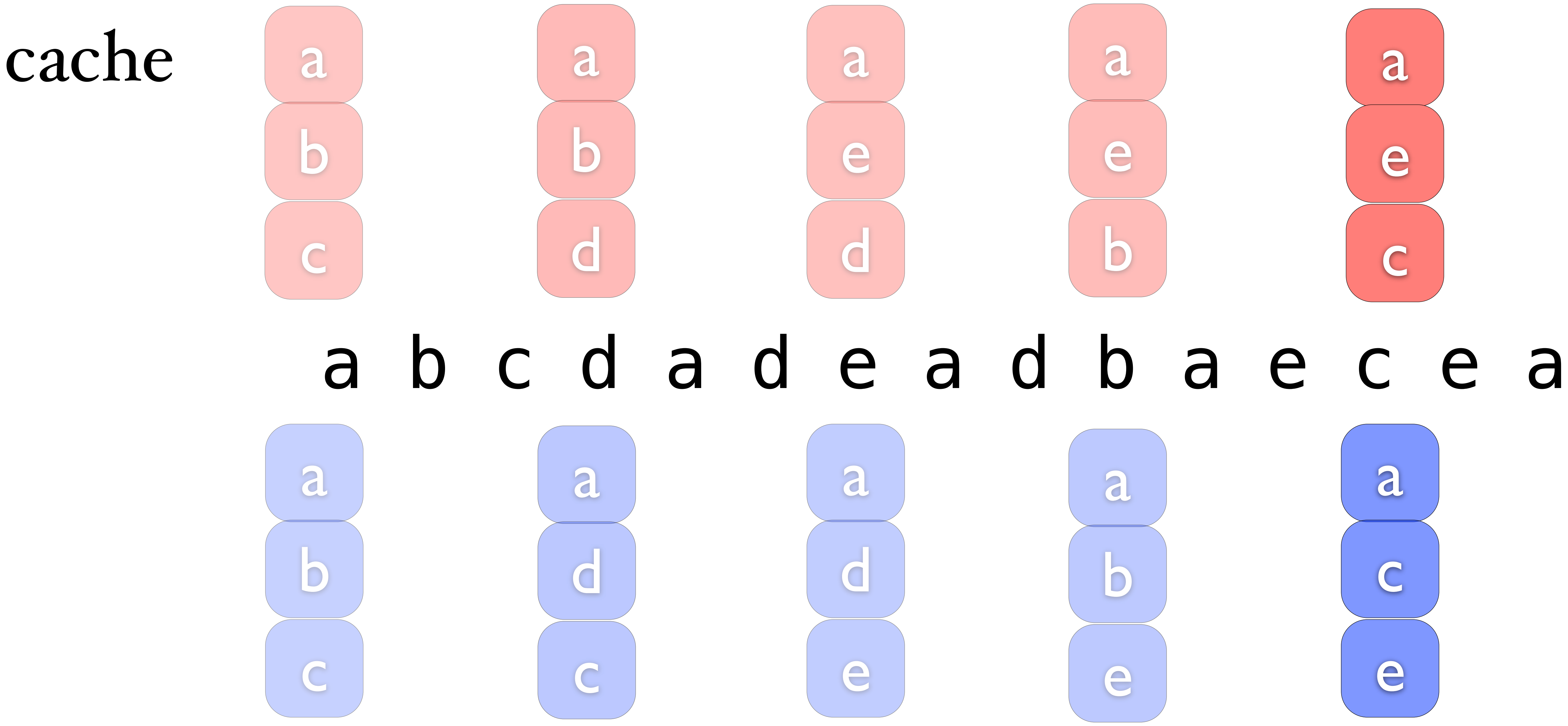
Exchange lemma

Let S be a reduced schedule that agrees with S_{ff} on the first j accesses.

Then there exists a schedule S' that agrees with S_{ff} on the first j+1 accesses and has the same or fewer misses.

What does it mean for 2 schedules to agree?

A schedule is a sequence of cache instructions:
NOP,NOP,NOP,evict(c,d),NOP,NOP,...



For example, these two schedules agree on the first three operations.

Some optimal
schedule.

S^*

S_{ff}

Some optimal schedule.

S^*

S_1

Agrees with S_{ff} on the first access. Can be constructed by applying the Lemma to S^* which agrees on 0 accesses.

S_{ff}

Some optimal schedule.

S^* S_1 S_2 S_{ff}

Agrees with S_{ff} on the first access. Can be constructed by applying the Lemma to S^* which agrees on 0 accesses.

Agrees with S_{ff} on the first two accesses.

Some optimal schedule.

S^* S_1 S_2 S_3 S_{n-1} S_{ff}

Agrees with S_{ff} on the first access. Can be constructed by applying the Lemma to S^* which agrees on 0 accesses.

Agrees with S_{ff} on the first two accesses.

Agrees with S_{ff} on the first three accesses.

S_{ff} has the same number of cache misses as S^*.

Some optimal schedule.

$$S^* \quad S_1 \quad S_2 \quad S_3 \qquad\qquad S_{n-1} \quad S_{\text{ff}}$$

Agrees with S_{ff} on the first access. Can be constructed by applying the Lemma to S^* which agrees on 0 accesses.

Agrees with S_{ff} on the first two accesses.

Agrees with S_{ff} on the first three accesses.

S_{ff} has the same number of cache misses as S^*.

$$miss(S^*) \geq miss(S_1) \geq miss(S_2) \geq \cdots \geq miss(S_n)$$

Some optimal schedule.

S^*

S_{ff}

Since S^* is optimal, this means that all of these relations need to be equality.

This also means the S_{ff} is therefore optimal.

Some optimal schedule.

S^* $\qquad\qquad\qquad\qquad$ S_{ff}

$$miss(S^*) \geq miss(S_1) \geq miss(S_2) \geq \cdots \geq miss(S_n) = miss(S_{ff})$$

Since S^* is optimal, this means that all of these relations need to be equality.

This also means the S_{ff} is therefore optimal.

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 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.

At time j, both S and S_{ff} have the same state.
Let d be the element accessed at time j+1.

Proof of lemma

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

easy case 1

Proof of lemma

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

easy case 1 d is in the cache.

Proof of lemma

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

easy case 1 d is in the cache.

Both S and S_{ff} agree since both do NOPs at j+1.

Proof of lemma

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

easy case 2

Proof of lemma

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

easy case 2 d is not in the cache, but both schedules “evict e for d.”

Proof of lemma

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

easy case 2 d is not in the cache, but both schedules “evict e for d.”

Both S and S_{ff} agree at j+1.

Proof of lemma

case 3

Proof of lemma

case 3 S does evict(d,e), and S_{ff} does evict(f,e)

Proof of lemma

case 3 S does evict(d,e), and S_{ff} does evict(f,e)

The state of the cache after this operation:

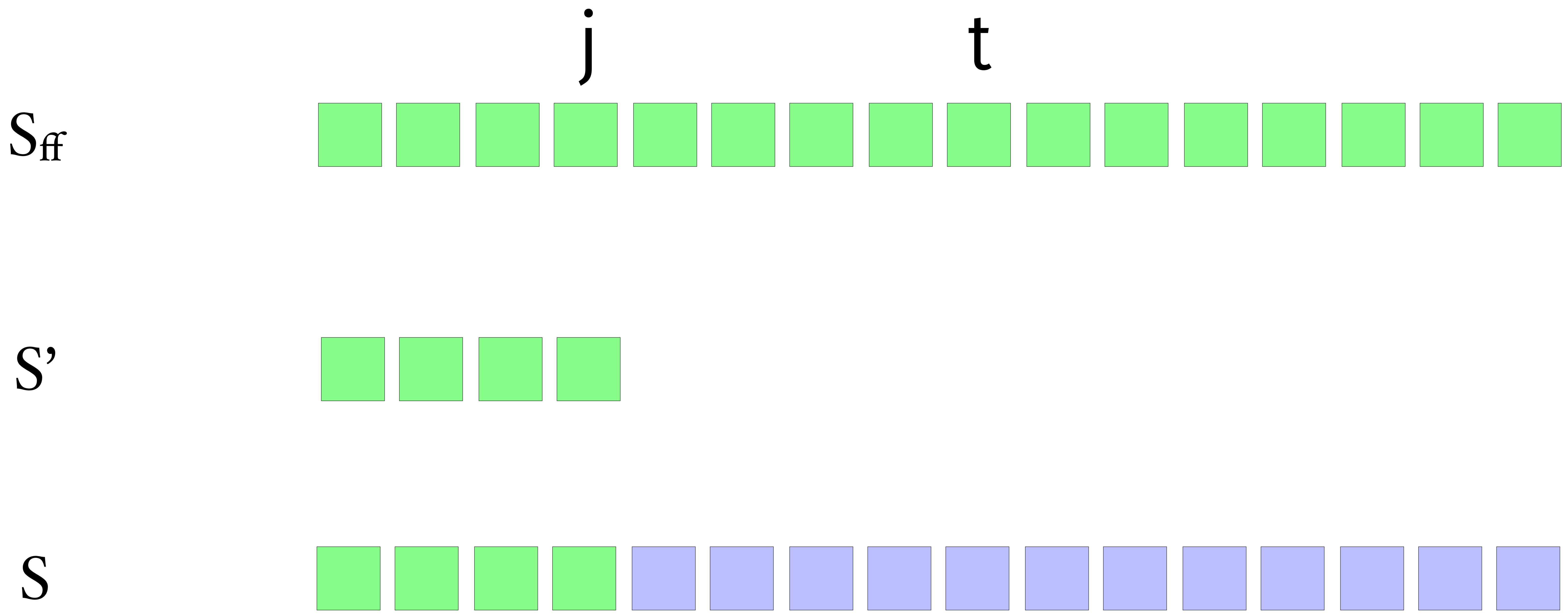
Proof of lemma

case 3 S does evict(d,e), and S_{ff} does evict(f,e)

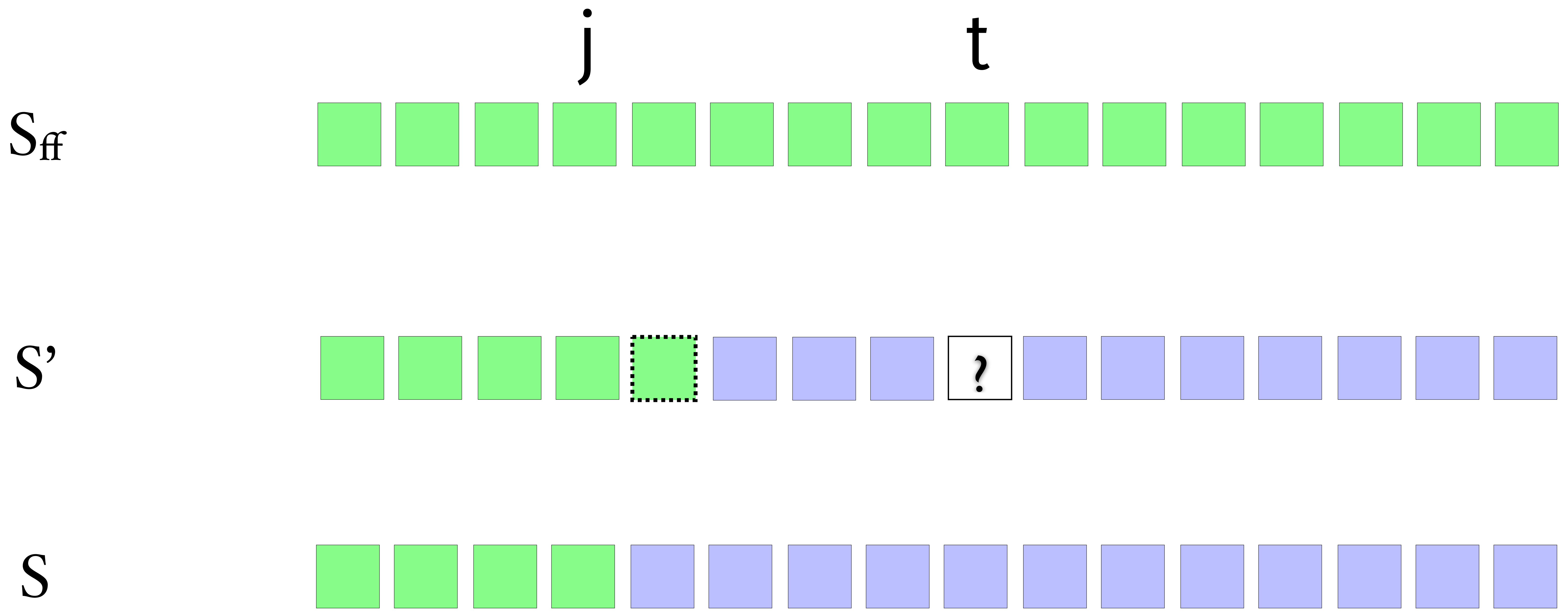
The state of the cache after this operation:

Challenge: the lemma requires us to find some schedule S' that agrees with S_{ff} and has the same or fewer misses as S.

Timeline

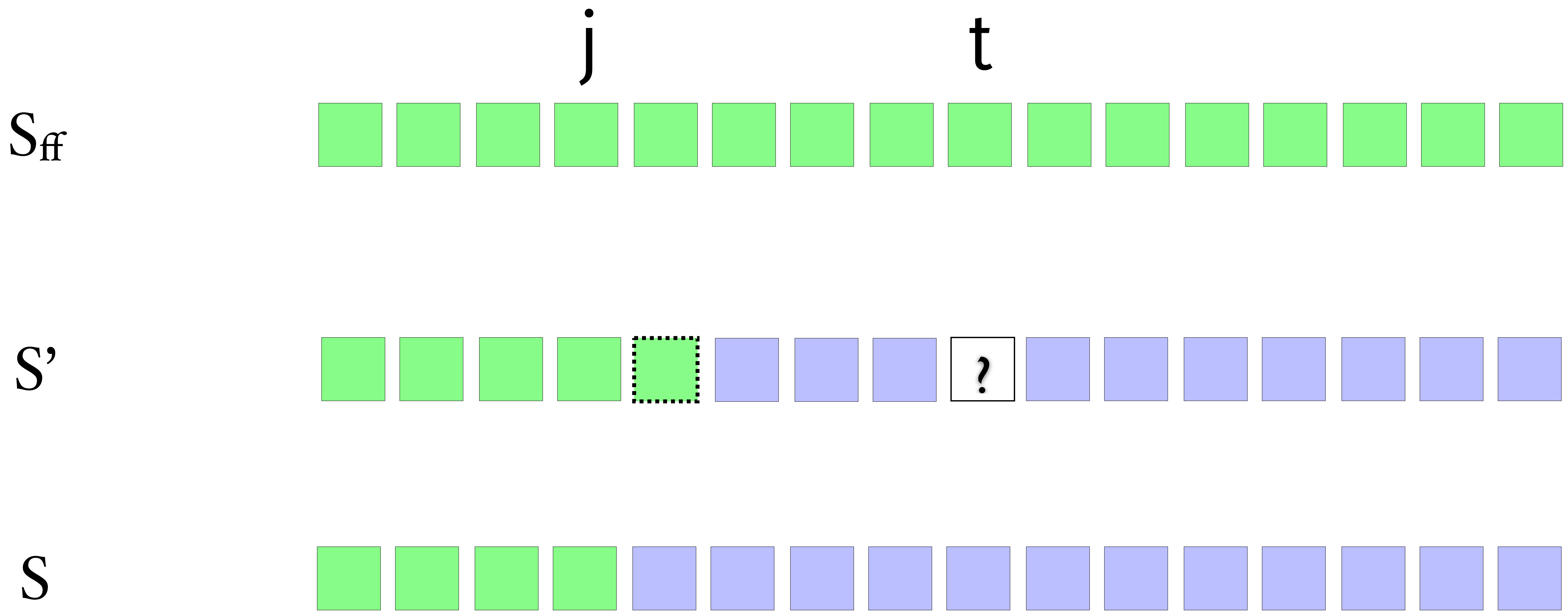


Timeline



Copy j+1 from S_{ff}. Then copy from S until t (the first time that either e or f are involved). Then copy from S until the end.

Timeline



Copy j+1 from S_{ff}. Then copy from S until t (the first time that either e or f are involved). Then copy from S until the end.

Challenge: Argue that S' has the same misses as S.

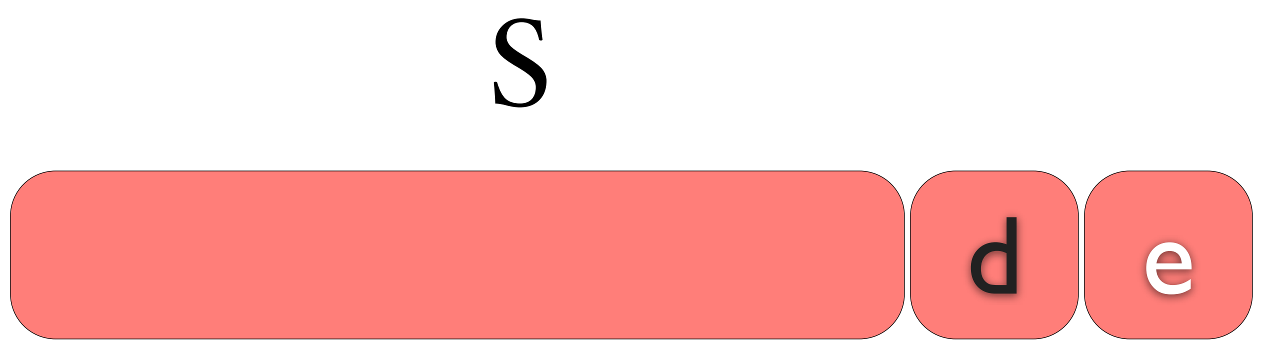
Proof of lemma

Let t be the first access that either e or f are involved.

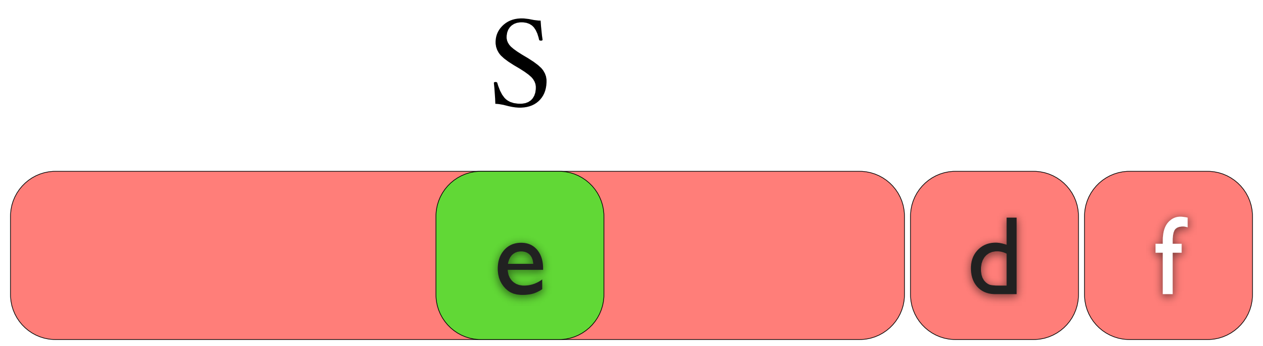
What if t is “access e”:

Proof of lemma

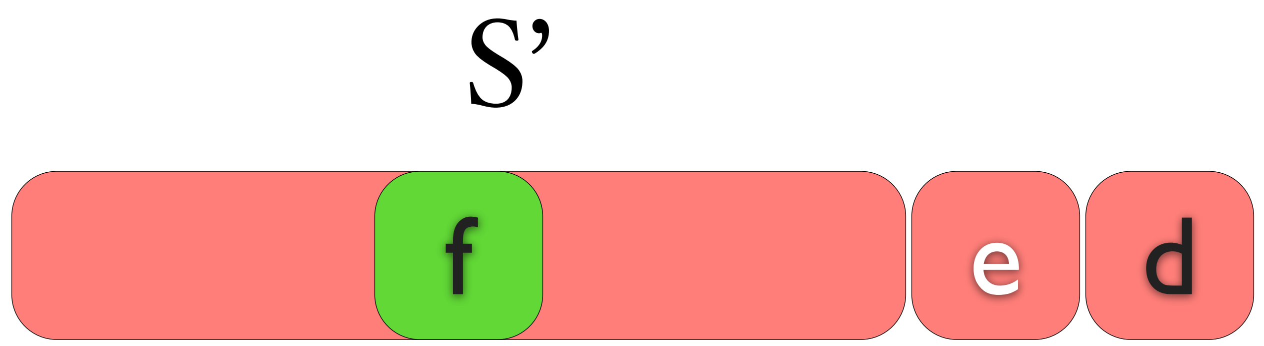
What if t = access e:



S needs to evict some element to load e.
If it evicts(f,e), then S’ can do a NOP.



If it evicts(h,e) $h \neq f$, S’ can evict(h,f) and maintain equality of the cache.



Proof of lemma

what if t=access f ?

Proof of lemma

what if t=access f ?

This case is impossible because f is accessed “farthest in the future.”

Proof of lemma

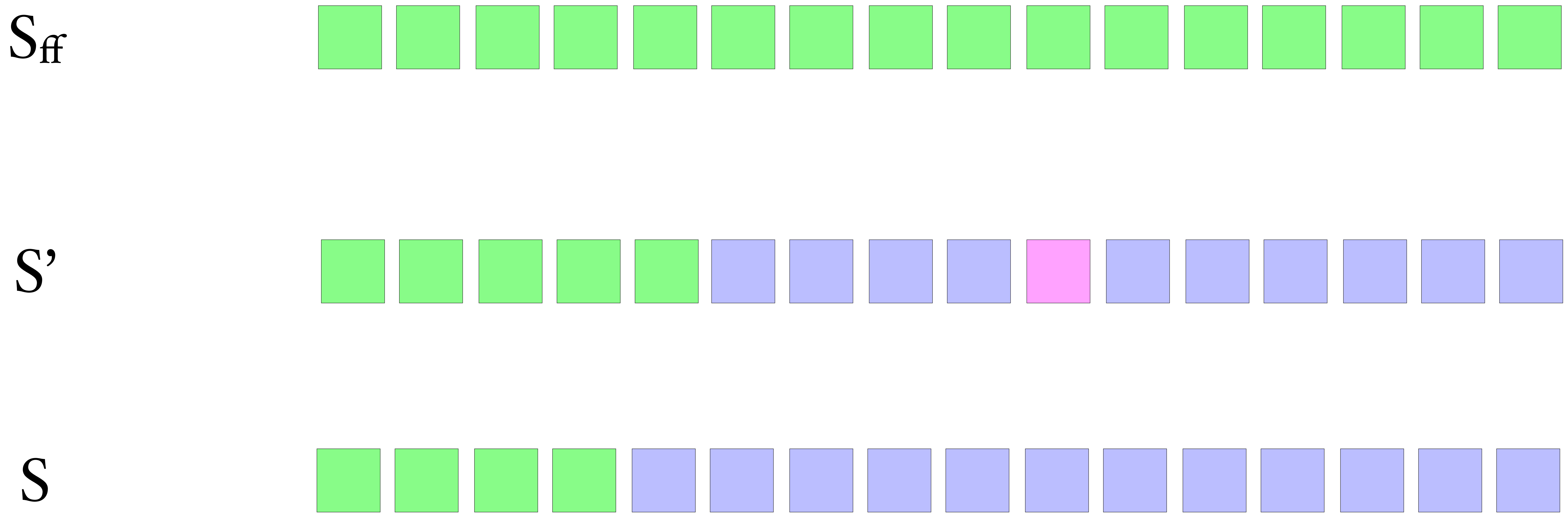
what if t is evict(f,x) ?

Proof of lemma

what if t is evict(f,x) ?

Then S’ can evict(e,x) and have the same cache state.

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}

Recap

The greedy algorithm is quite simple.

But the analysis for why the solution works is more subtle and complicated.

In this case, we had to apply the exchange lemma multiple times to prove optimality.