L10 5800

feb 18/21 2022

24/25

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)

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

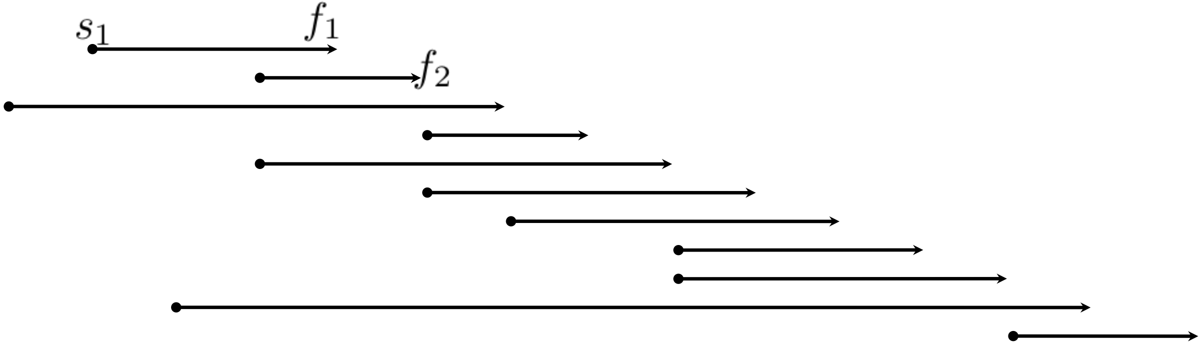
$$f_i \leq s_j$$

problem statement

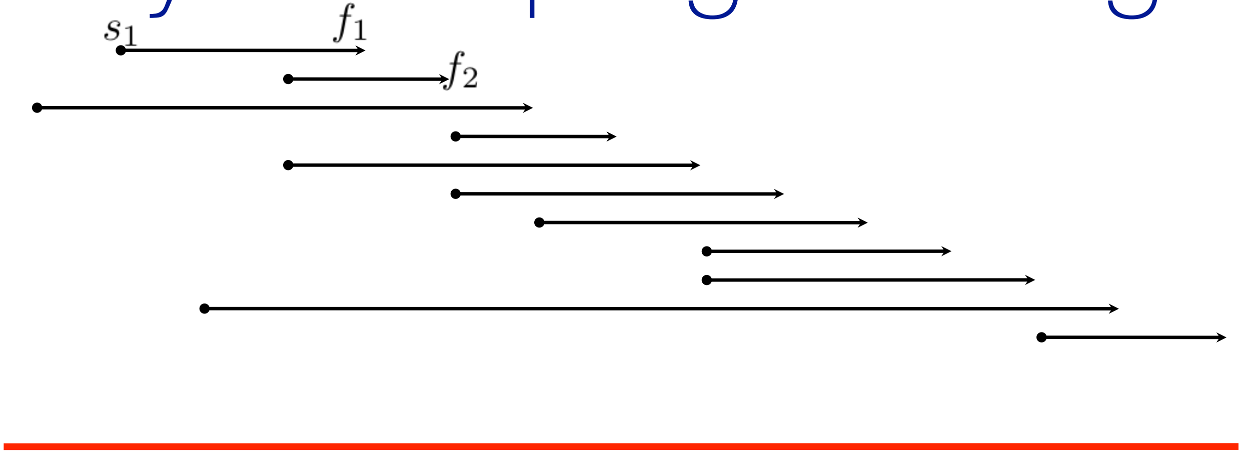
$(a_1, \ldots, a_n)$

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

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

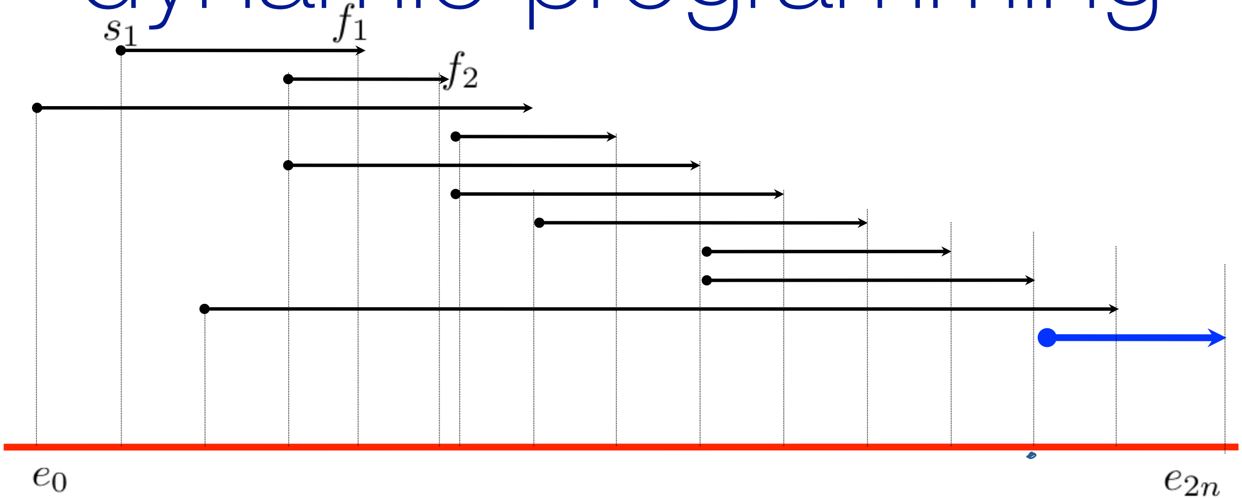


dynamic programming



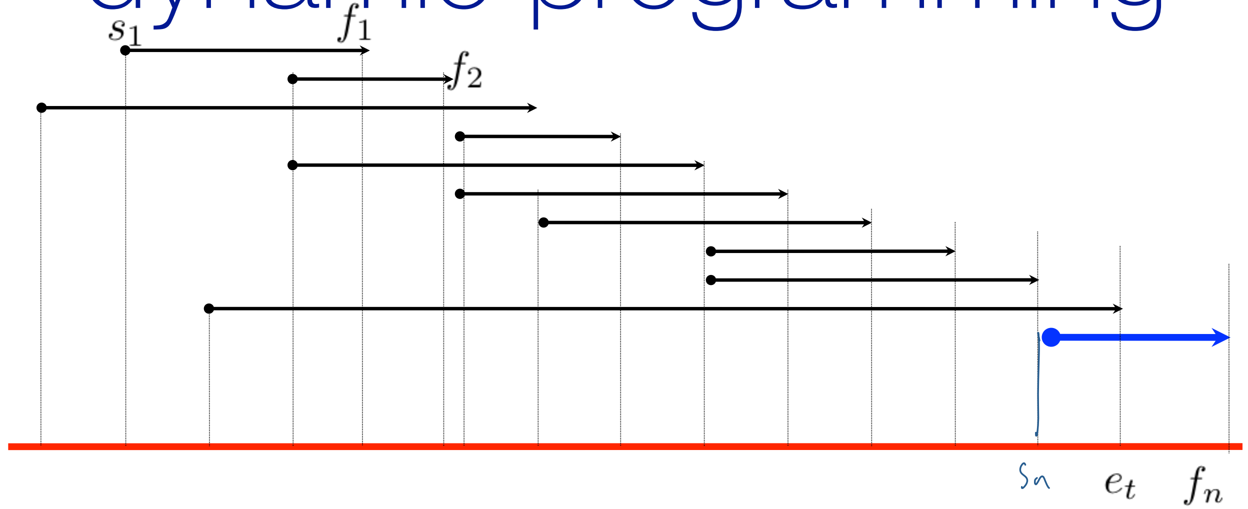
Lets draw all of the events on a timeline.

dynamic programming



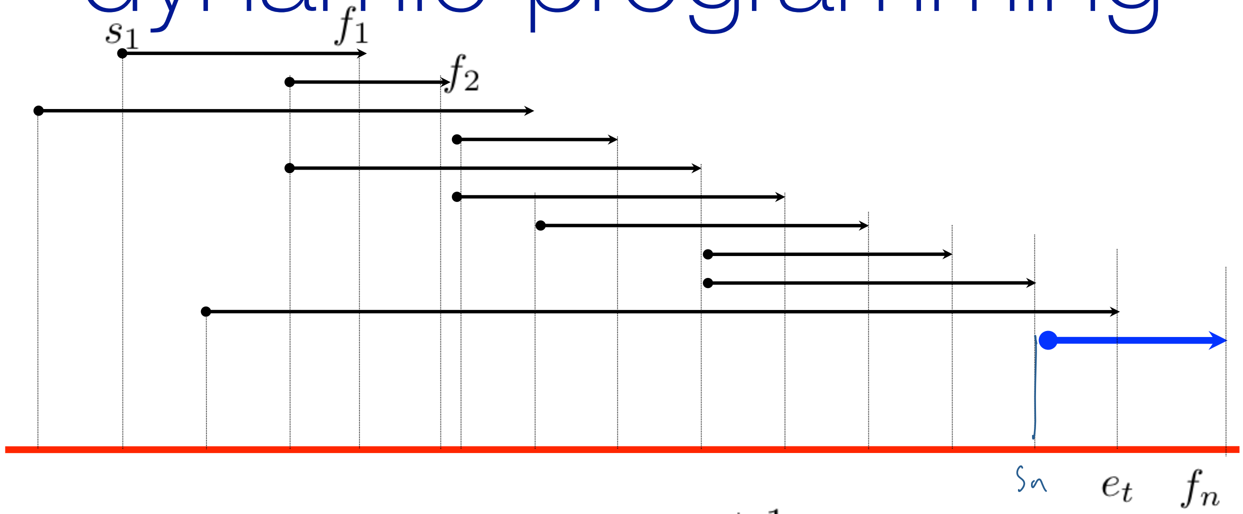
$Best_{2n} =$ Maximum number of non-overlapping activities possible among the first 2n events.

dynamic programming



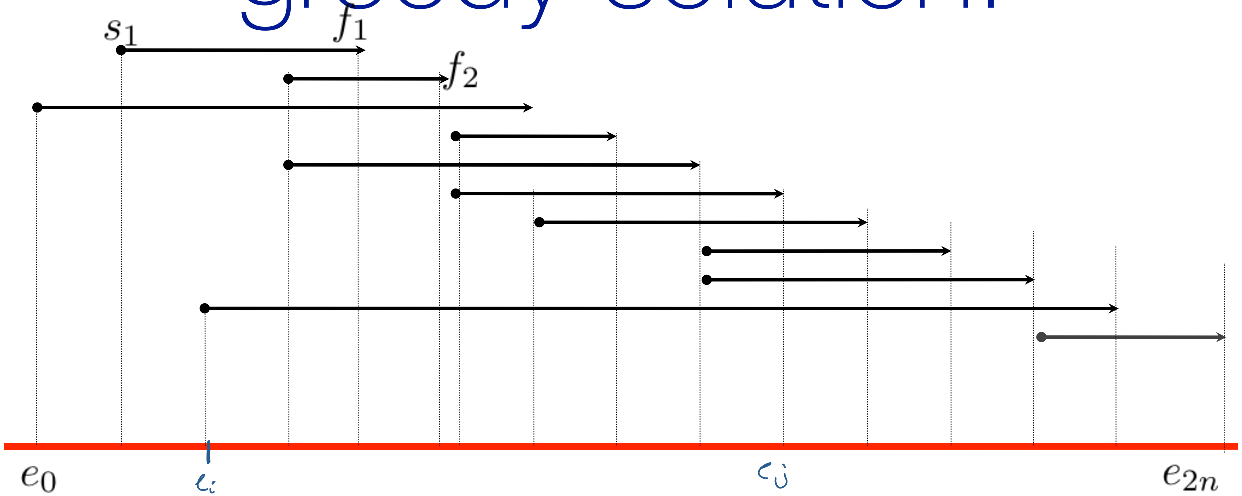
$$\mathrm{BEST}_{f_n} =$$

dynamic programming



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

greedy solution:

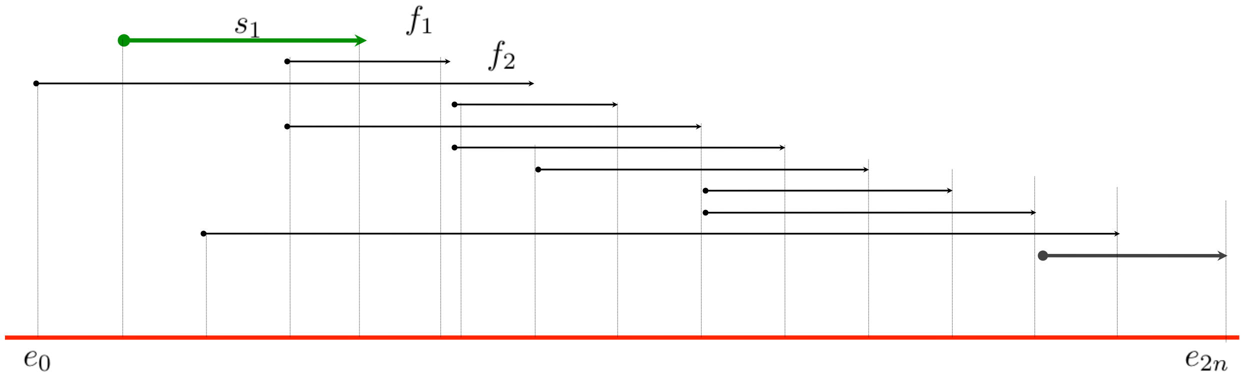


DEFINITION:

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

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

greedy solution:



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

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

PROOF:

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

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

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

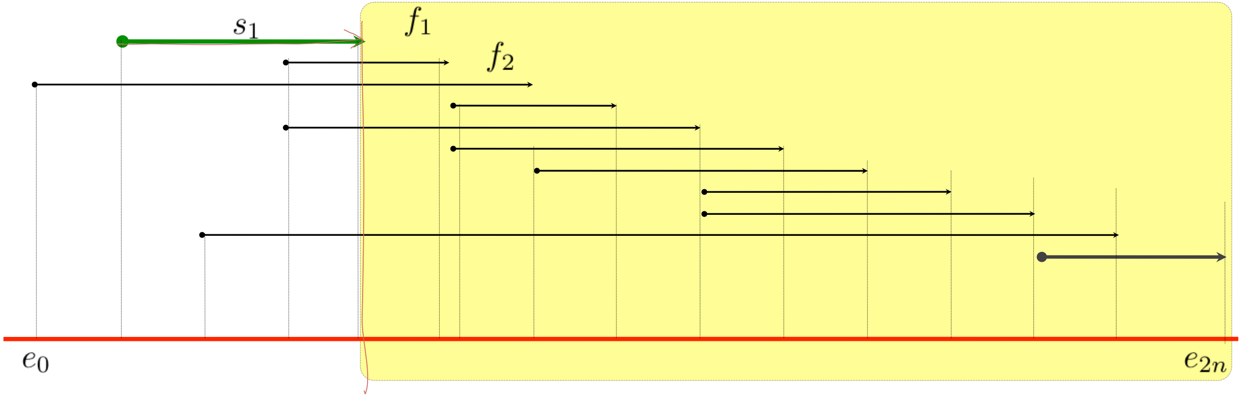
Consider a new solution that replaces a with a^*.

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

Exchange

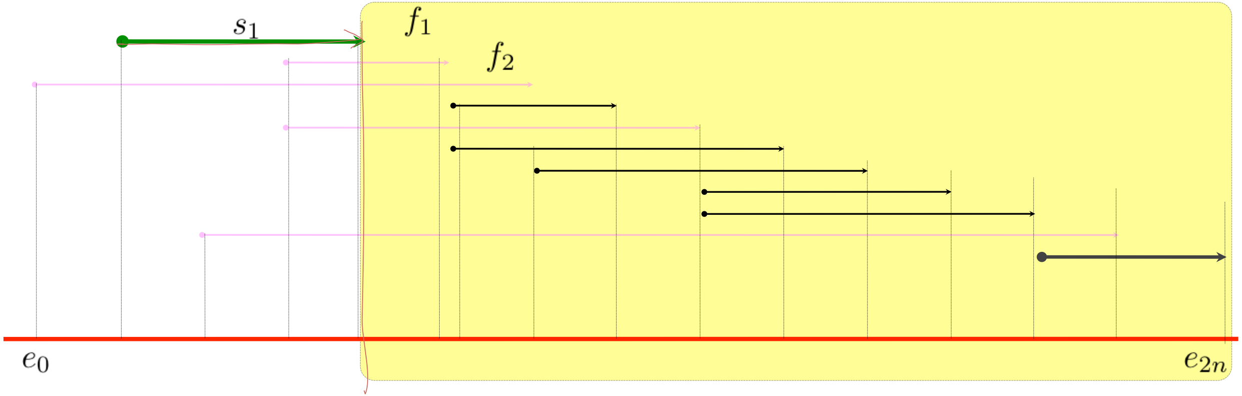
This new set is valid because a^* finishes before a and thus does not overlap with any activities. This new solution also has the same size and is therefore also optimal too.

greedy solution:



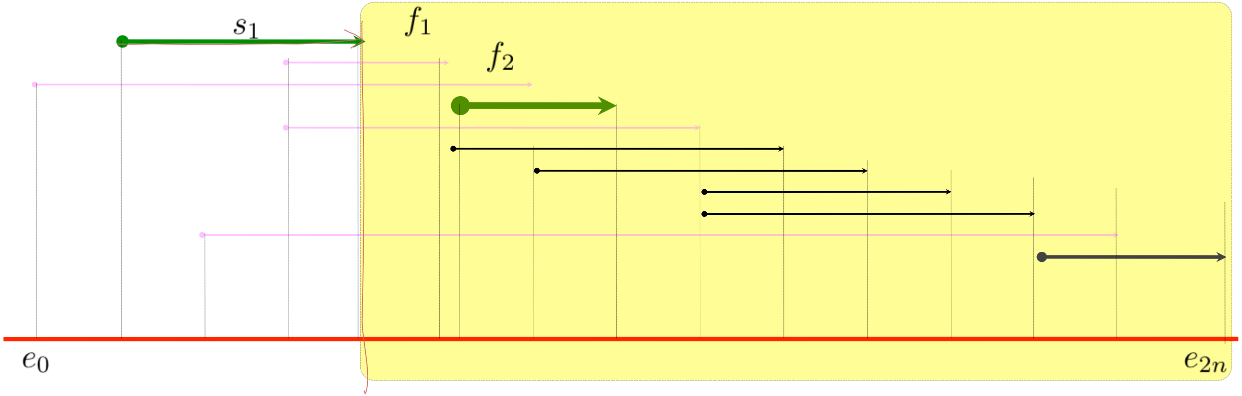
algorithm: find first event to finish. add to solution.
remove conflicting events.
continue.

greedy solution:



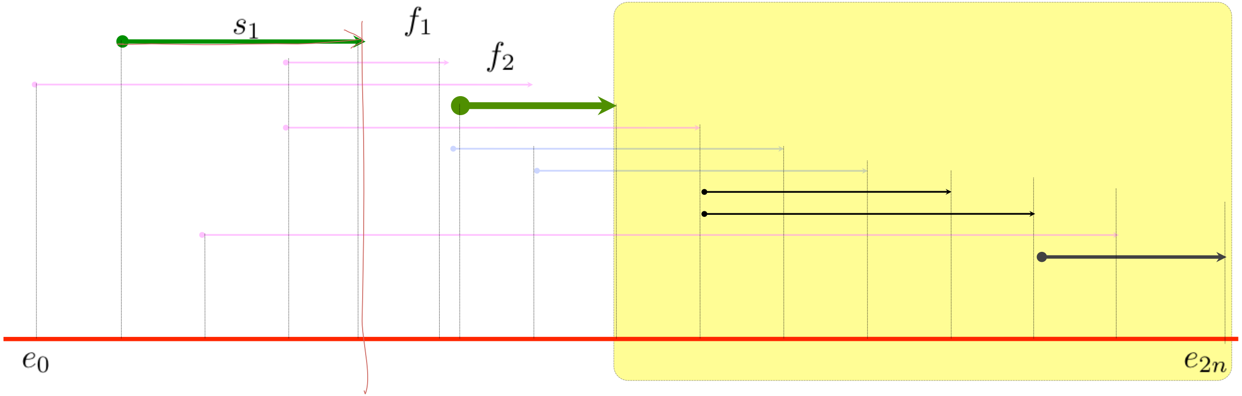
algorithm: find first event to finish. add to solution.
remove conflicting events.
continue.

greedy solution:



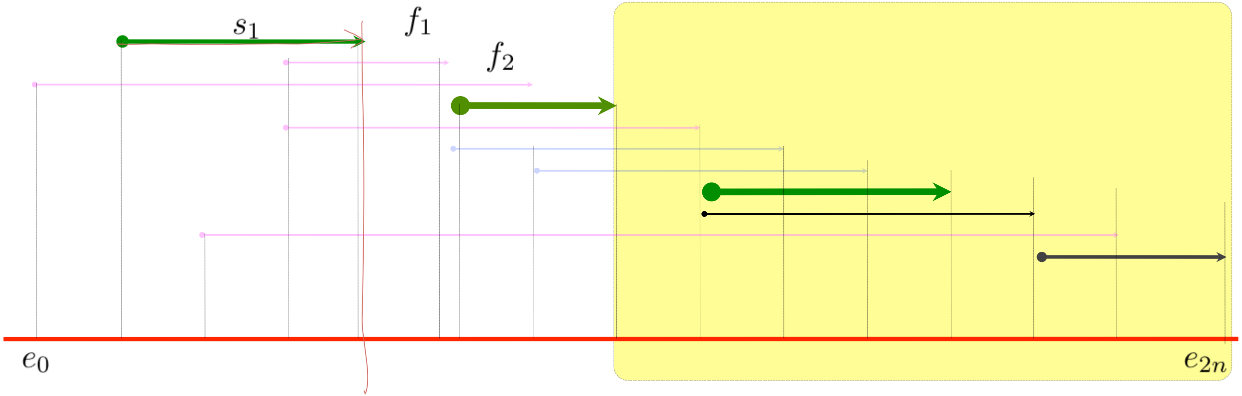
algorithm: find first event to finish. add to solution.
remove conflicting events.
continue.

greedy solution:



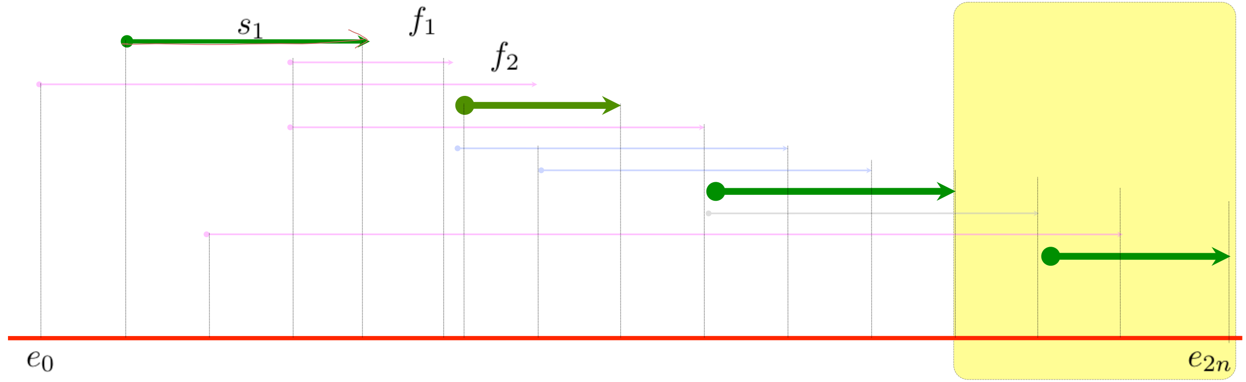
algorithm: find first event to finish. add to solution.
remove conflicting events.
continue.

greedy solution:



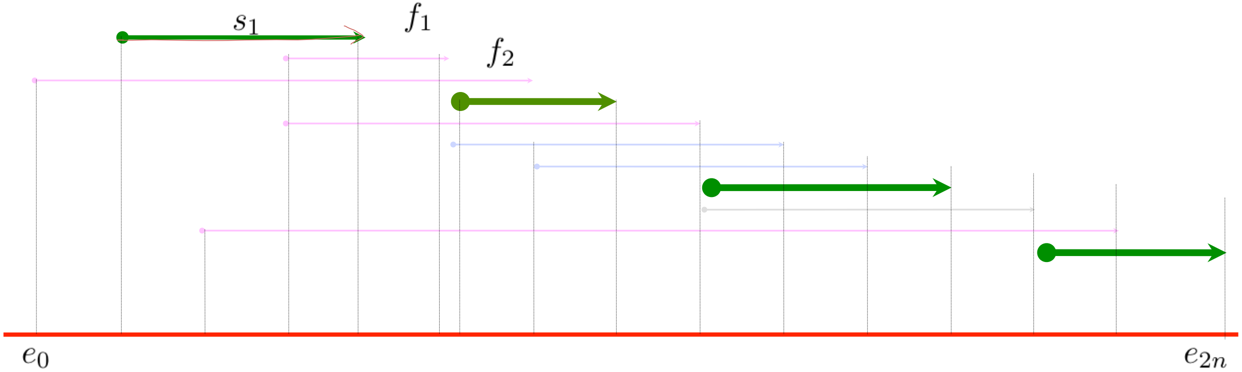
algorithm: find first event to finish. add to solution.
remove conflicting events.
continue.

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) $s_i < f_i$

$\Theta(n)$ time solution because each ~~event~~ activity is processed just once.

Recap

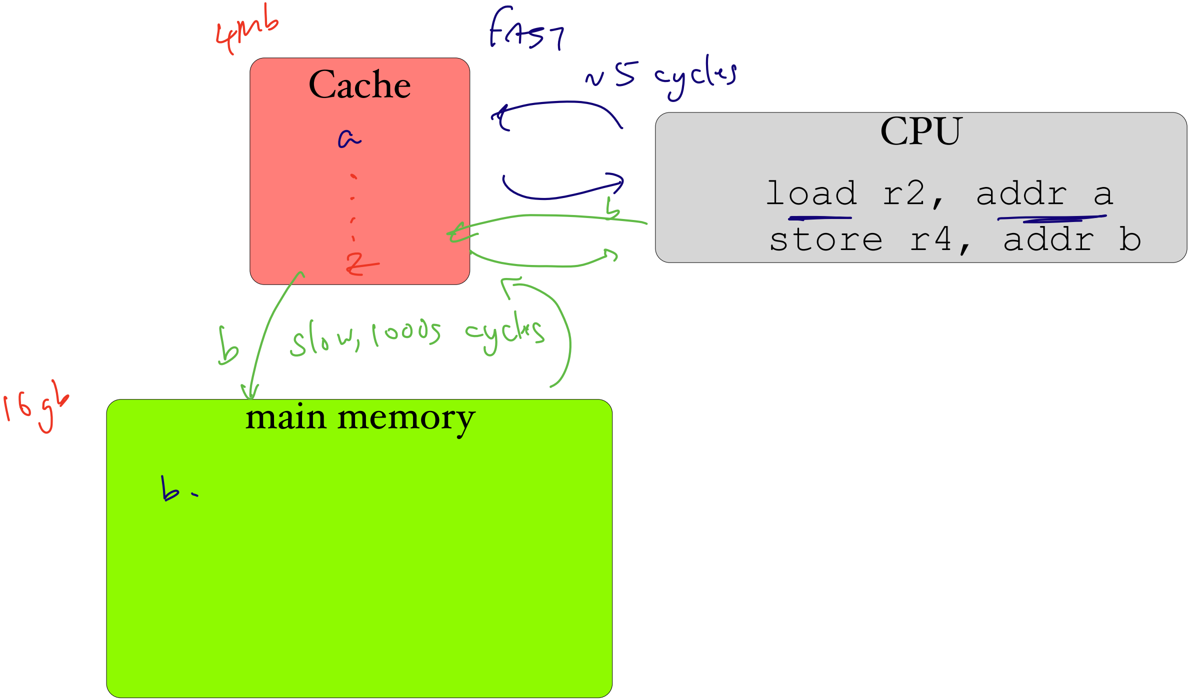
The main idea in this algorithm was the "exchange argument."

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



Cache

CPU

```
load r2, addr a
store r4, addr b
```

main memory

question:

How to manage the cache.

Which items should we evict when the cache is full.??

question:

How do we manage a fully-associate cache?

any empty slot in the cache can hold any address.

When it is full, which element do we replace?

problem statement

input: k, size of the cache, memory access pattern $d_1 d_2 \ldots d_m$.

output: schedule of operations on the cache that minimizes the number of cache misses.

cache is fully associative with line-size 1

problem statement

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

d_i is the address of the memory access at operation i.

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

Replace the element in the cache that is accessed "farthest in the future" (ff)

Belady eviction rule

Replace the element in the cache that is accessed “farthest into the future”

example

k=3

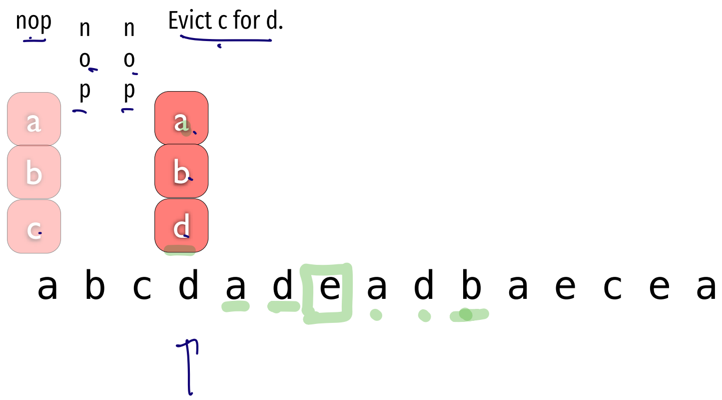
cache

a
b
c

memory access pattern. →

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

because at this point,
c is accessed farthest
in the future

example

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

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

↑

Replace b with e

because b is accessed

farthest in the future.

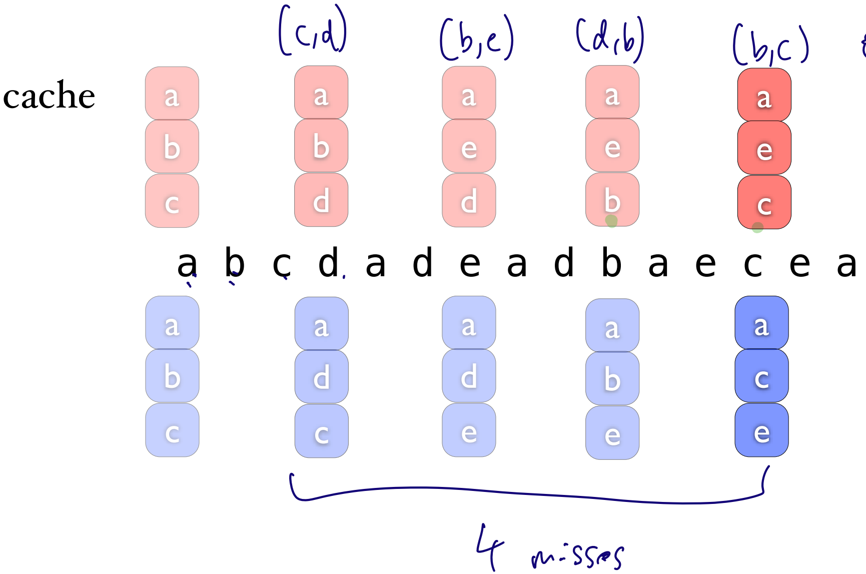
example

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

Replace d with b.

example



← Belady ff schedule, optimal w/ 4 cache misses.

Here is an alternate optimal set of cache operations.

Surprising theorem

The schedule S_{ff} produced by the Belady eviction rule is optimal, if it has the fewest cache misses that are possible while satisfying the memory access pattern.

if address d_i is accessed at operation i, then d_i must be in the cache by operation i.

Surprising theorem

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

Notation:

schedule

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

list of instructions to perform on the cache,
e.g. "Nop" or "evict b for d" evict(b,d).

Reduced schedule: A schedule in which evict(x,y) 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:

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:

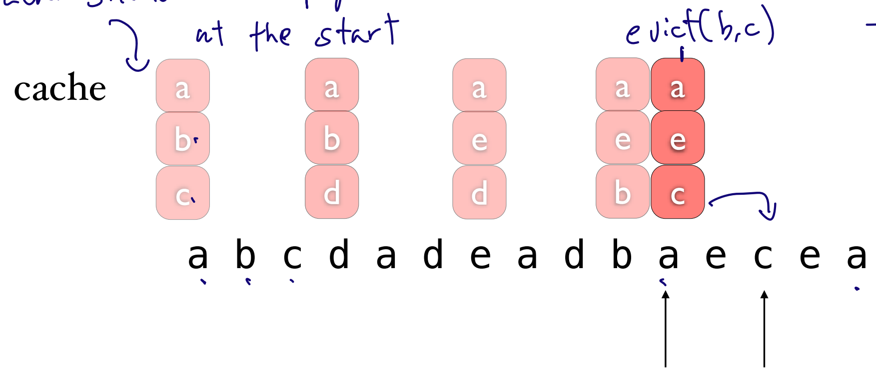
A schedule in which"evict x for y" instruction only occurs when y is accessed.

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

cache should be empty at the start



evict(b,c)

This schedule satisfies the memory access pattern.

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

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

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

e.g. $\text{miss}(S) \geq \text{miss}(S')$.

Exchange lemma

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

0

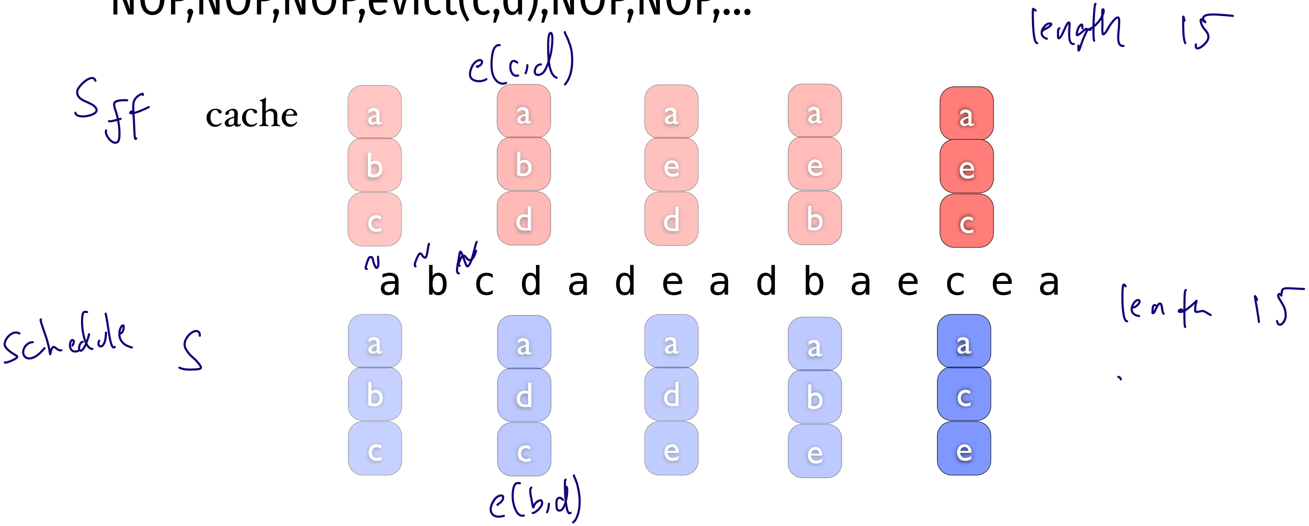
S_1

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

1

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 reduced schedule.

S^*

S_1

S_{ff}

S_{other}

greedy solution.

S^* agrees with S_{ff} on 0 operations.

The exchange lemma can be applied to S^* show the existance of a schedule S_1 that agrees with S_{ff} on 1 operation.

$$\text{miss}(S^*) \underset{=}{\geq} \text{miss}(S_1)$$

we want to argue that S_{ff} is optimal.

Some optimal schedule.

S^* S_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.

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.

15 operations

Some optimal schedule.

Exchange Lemma

$$\underline{S^*} \quad \overset{15}{\underline{S_1}} \curvearrowright \underline{S_2} \curvearrowright S_3 \curvearrowright \quad \curvearrowright \quad \underline{S_{n-1}} \curvearrowright \underline{S_{\text{ff}}}$$

15 operations.

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^*.

$\text{miss}(S_1) \geq \text{miss}(S_2)$

Repeatedly applying the exchange Lemma allows us to argue that $\underline{S_{ff}}$ is optimal.

Some optimal schedule. ↗ length n schedule

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

↗ length n schedule

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^*.

optimal

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

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^*

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~~ operations.
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: At operation j, both schedule S and S_{ff} have the same cache state. because they have both done the same cache operations starting from the same empty state.

Let d be the address accessed at operation j+1.

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 address d is in the cache.

$\Rightarrow$ S will do a NOP. S_{ff} will do a NOP

$\Rightarrow$ S and S_{ff} agree on the first $j+1$ operations. ✓

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 d is not in the cache, but both do evict(e, d).

Again, S and S_{ff} agree on $j+1$ operations ✓.

So $S' = S$ satisfies the lemma.

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.

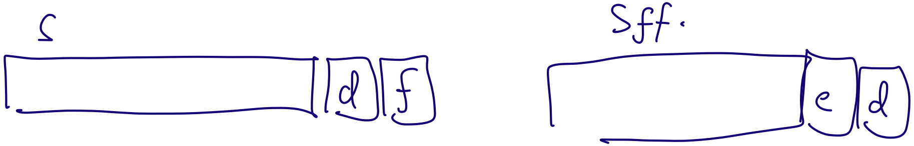
Proof of lemma

case 3. d is not in cache.

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

i.e., S and S_{ff} disagree on this j+1 operations.

The state of cache is



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:

Challenge: We need to construct an S' that agrees with S_{ff} on $j+1$ but doesn't incur any more misses than S.

Proof of lemma

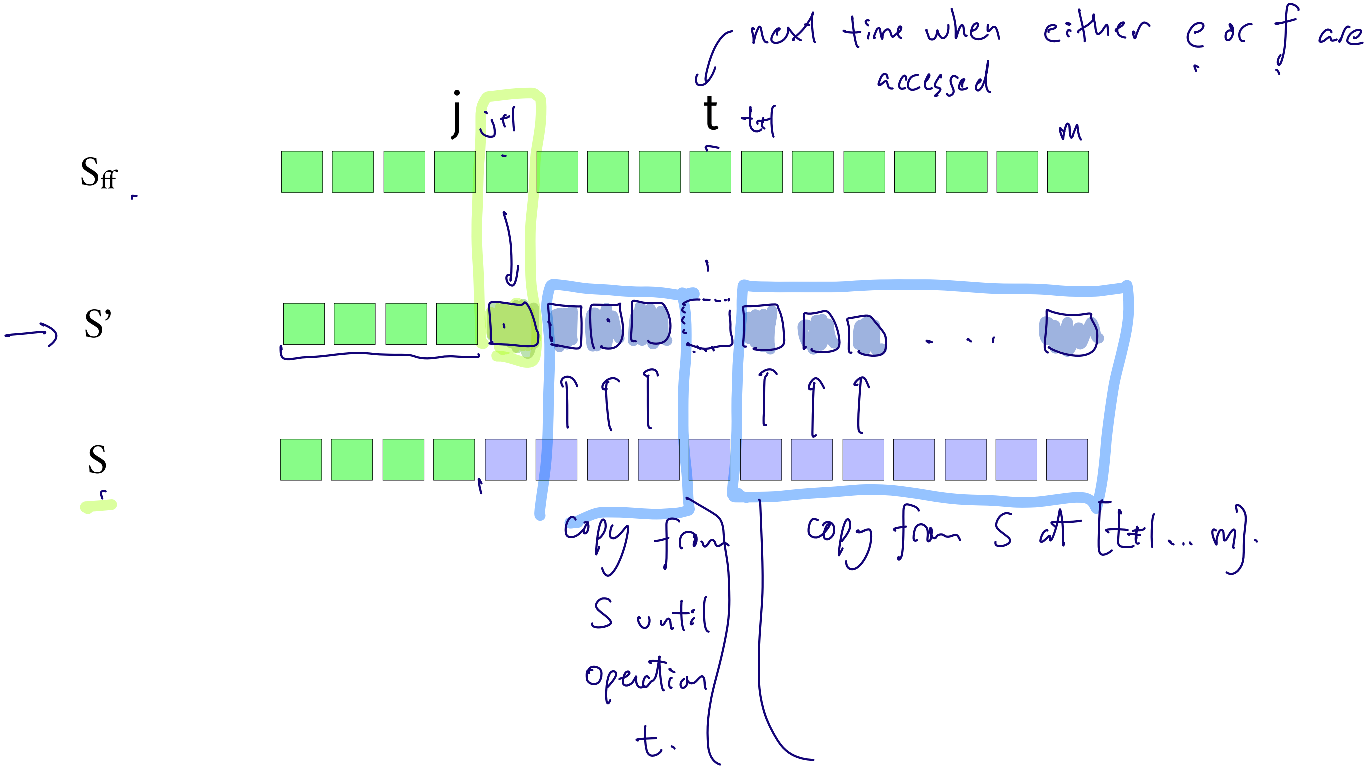
case 3 S does evict(~~d,e~~), and S_{ff} does evict(~~f,e~~)

e,d f,d

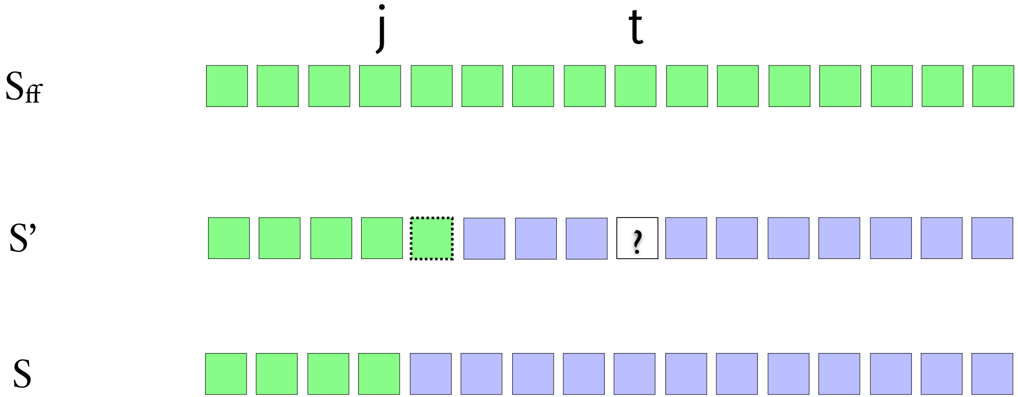
The state of the cache after this operation:

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

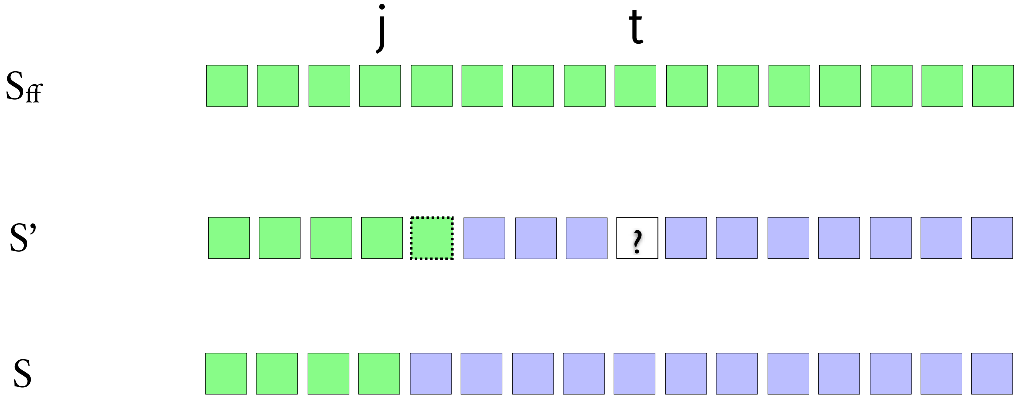

next time when either e or f are accessed
j
j+1
t
t+1
m
S_ff
S'
S
copy from S until operation t.
copy from S at [t+1 … m].

Timeline



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

operated meaningfully

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 accessed.

What if t=e: S must perform an eviction. because its cache does not contain e.

① if it performs evict(f,e), then S and S_{ff} have the same state. So S' can do a NOP.

h ≠ f.

② S does evict(h,e). In this case, S' can do evict(h,f)

miss(S) ≥ miss(S').

In either case, after this operation, both caches are the same.

Proof of lemma

what if t=e ?

Proof of lemma

what if t=f ?

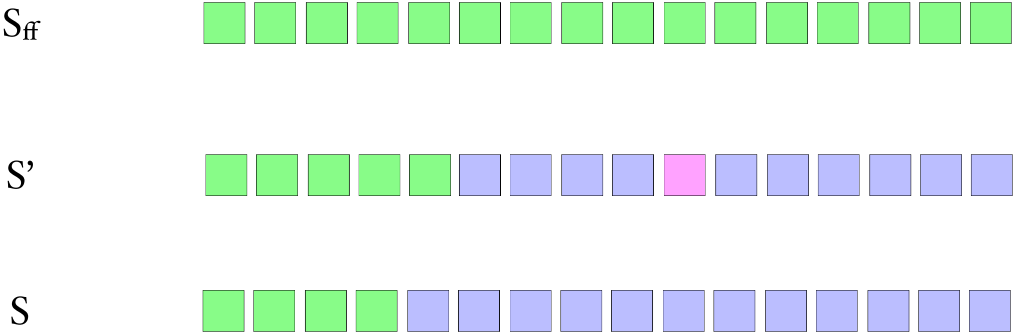
This is impossible case.

Because Sff always uses the farthest in the future rule. Earlier, Sff evicted f for d. So that means the access to f had to be farther in future than e. But t is defined as the first access to either e or f.

Proof of lemma

what if t is neither e nor f ?

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.

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

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.