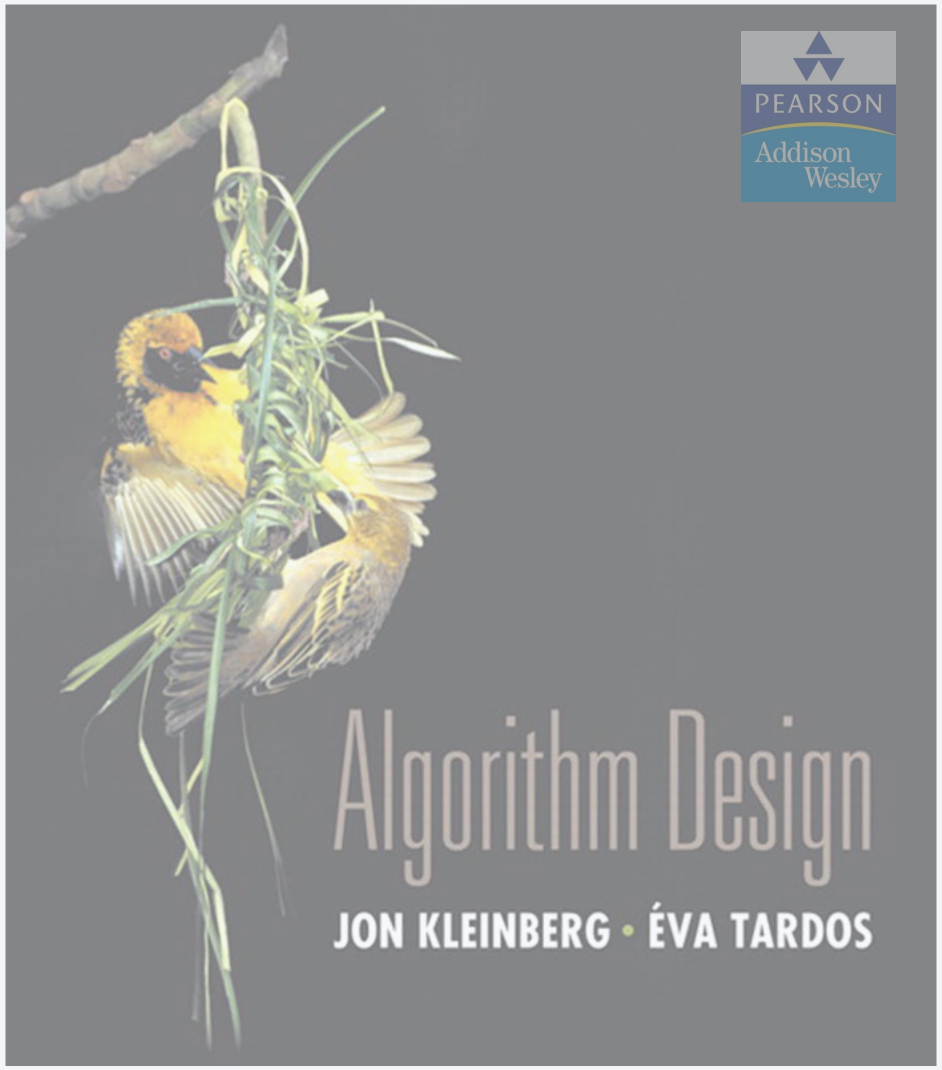


SECTION 4.3

4. GREEDY ALGORITHMS I

- *coin changing*
- *interval scheduling*
- *interval partitioning*
- *scheduling to minimize lateness*
- *optimal caching*

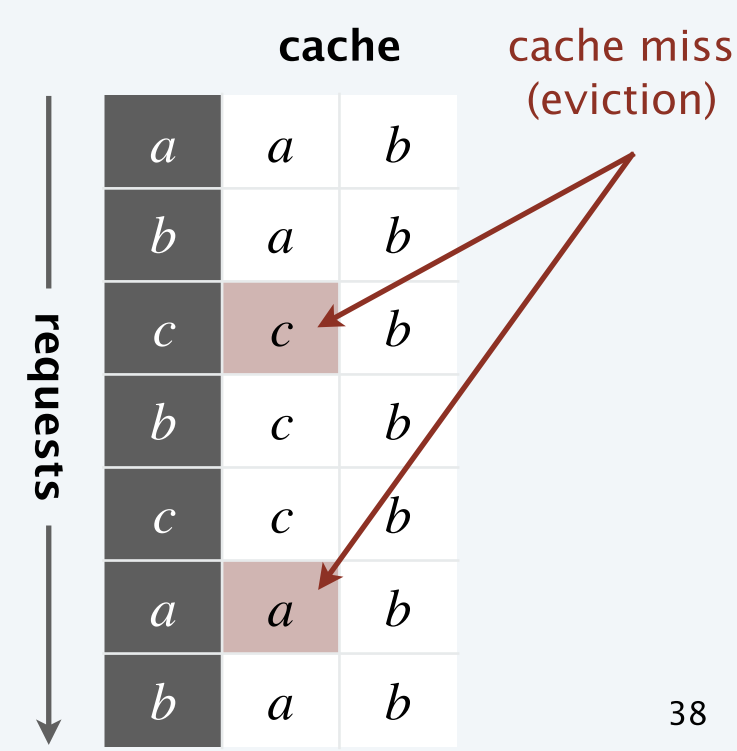
Optimal offline caching

Caching.
- Cache with capacity to store k items.
- Sequence of m item requests $d_1, d_2, \ldots, d_m$.
- Cache hit: item in cache when requested.
- Cache miss: item not in cache when requested.
  (must evict some item from cache and bring requested item into cache)

Applications. CPU, RAM, hard drive, web, browser, ….

Goal. Eviction schedule that minimizes the number of evictions.

Ex. $k = 2$, initial cache $= ab$, requests: a, b, c, b, c, a, b.
Optimal eviction schedule. 2 evictions.



Optimal offline caching: greedy algorithms

LIFO/FIFO. Evict item brought in least (most) recently.

LRU. Evict item whose most recent access was earliest.

LFU. Evict item that was least frequently requested.

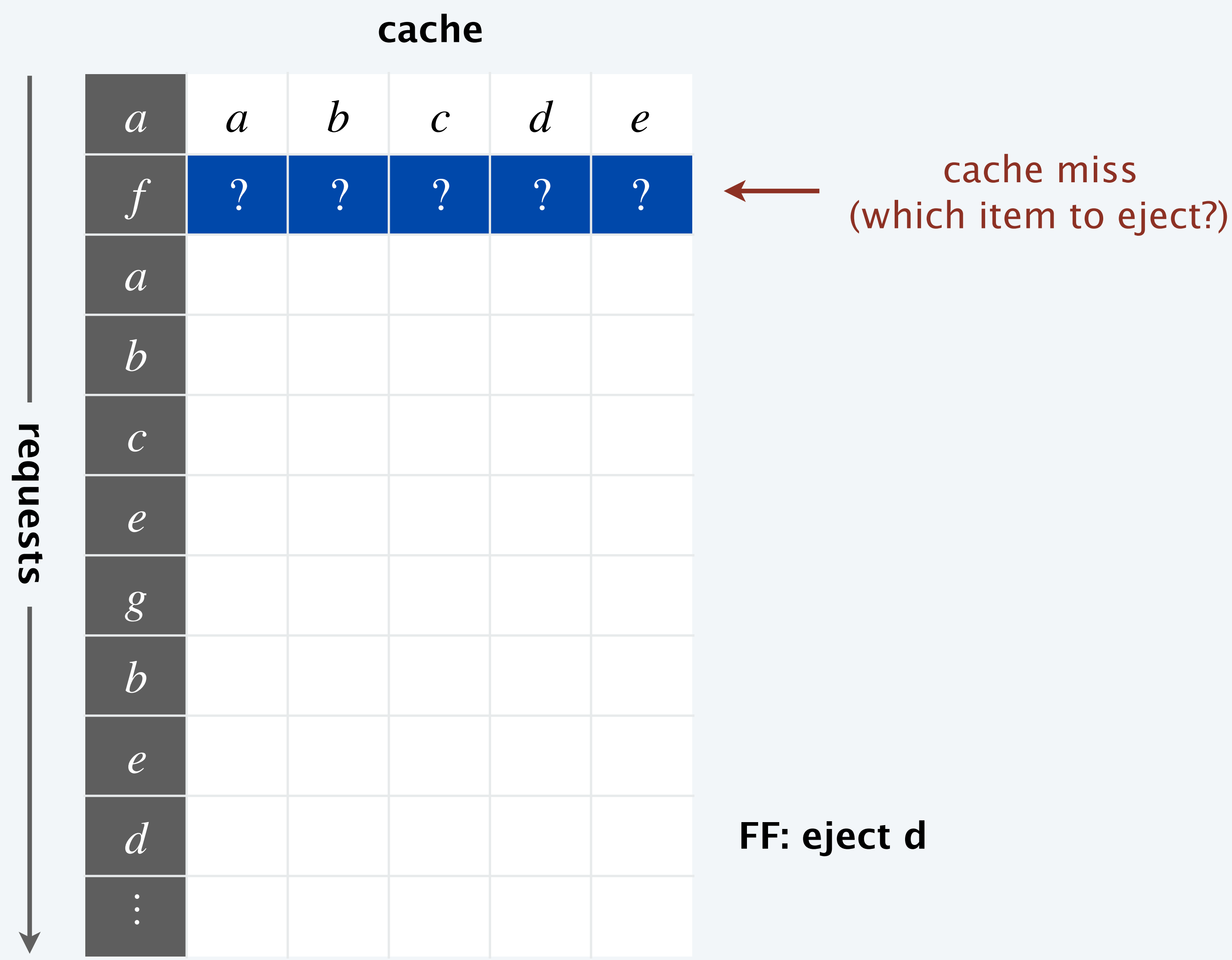
cache

requests						
⋮	.	.	.	.	.	
a	a	w	x	y	z	**FIFO: eject a**
d	a	w	x	d	z	**LRU: eject d**
a	a	w	x	d	z	
b	a	b	x	d	z	
c	a	b	c	d	z	
e	a	b	c	d	e	**LIFO: eject e**
g	?	?	?	?	?	
b						
e						
d						
⋮						

cache miss
(which item to eject?)

Optimal offline caching: farthest-in-future (clairvoyant algorithm)

Farthest-in-future. Evict item in the cache that is not requested until farthest in the future.



Theorem. [Bélády 1966] FF is optimal eviction schedule.

Pf. Algorithm and theorem are intuitive; proof is subtle.

Which item will be evicted next using farthest-in-future schedule?

A.

B.

C.

D.

E.

requests	cache			
⋮	.	.	.	.
B	D	B	Y	A
C	D	B	C	A
E	D	E	C	A
F	?	?	?	?
C				
D				
A				
E				
A				
C				
⋮				

cache miss
(which item to eject?)

Reduced eviction schedules

Def. A reduced schedule is a schedule that brings an item d into the cache in step j only if there is a request for d in step j and d is not already in the cache.

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

d enters cache without a request

d enters cache even though already in cache

an unreduced schedule

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

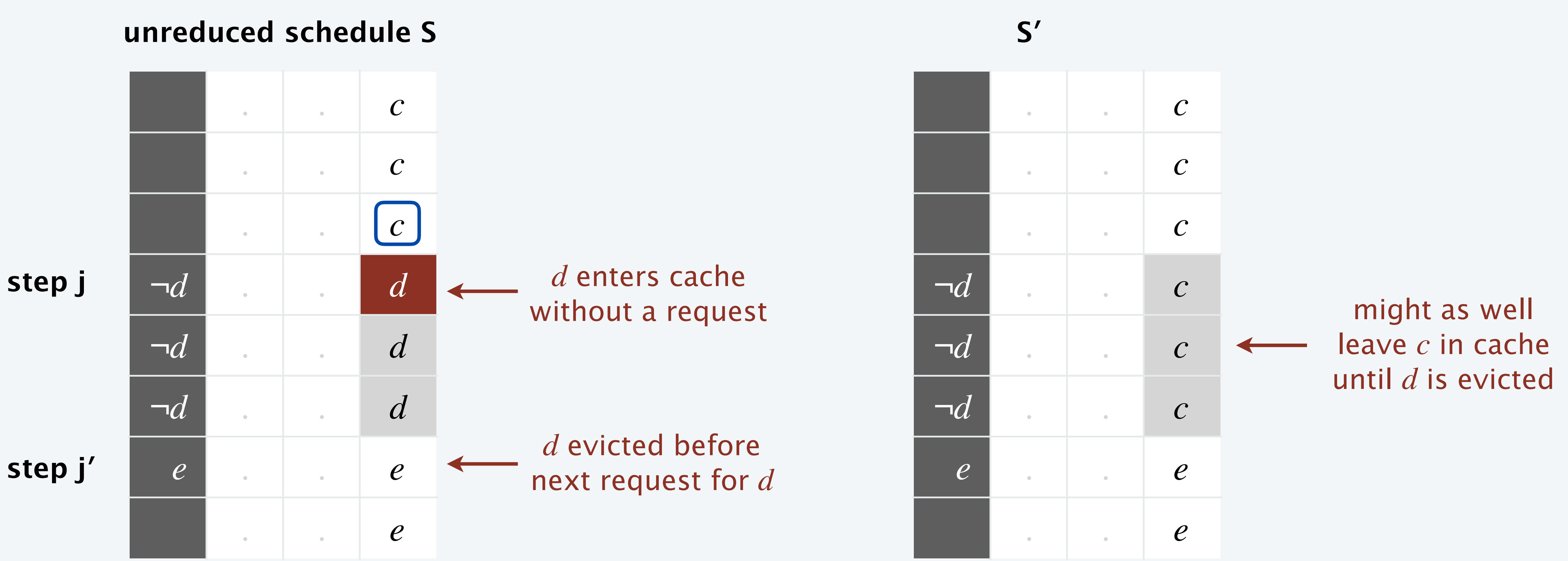
a reduced schedule

Reduced eviction schedules

Claim. Given any unreduced schedule S, can transform it into a reduced schedule S' with no more evictions.

Pf. [by induction on number of steps j]

- Suppose S brings d into the cache in step j without a request.
- Let c be the item S evicts when it brings d into the cache.
- Case 1a: d evicted before next request for d.

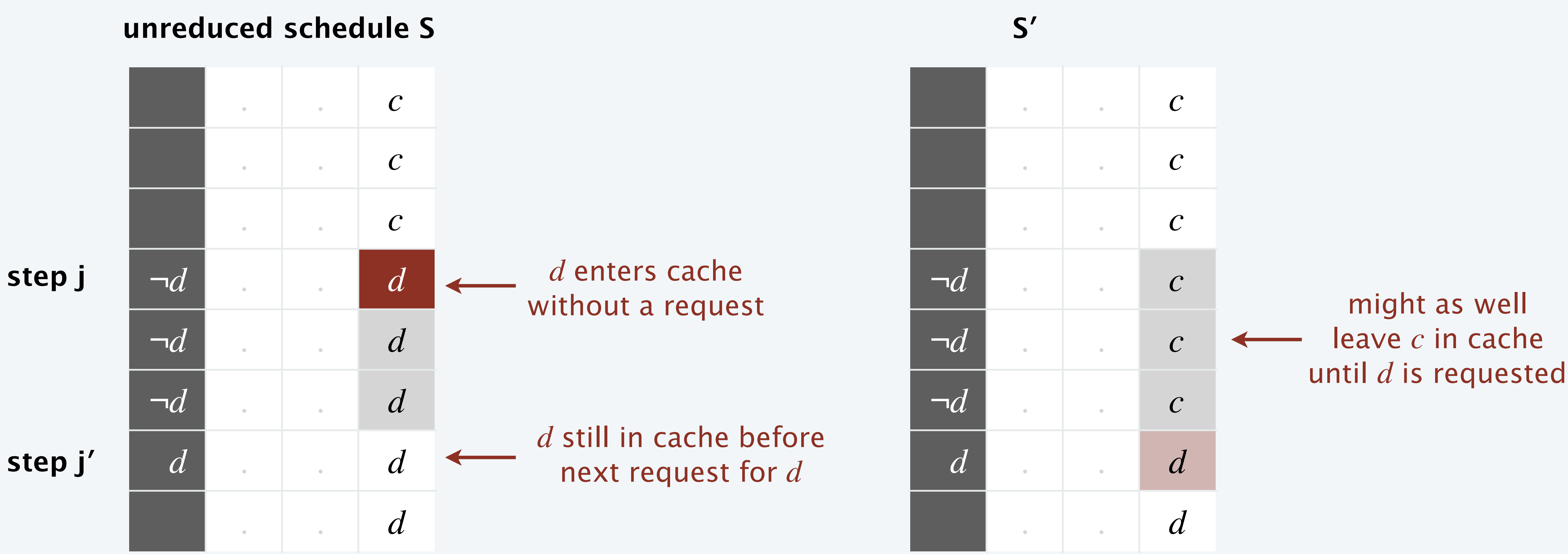


Reduced eviction schedules

Claim. Given any unreduced schedule S, can transform it into a reduced schedule S' with no more evictions.

Pf. [by induction on number of steps j]

- Suppose S brings d into the cache in step j without a request.
- Let c be the item S evicts when it brings d into the cache.
- Case 1a: d evicted before next request for d.
- Case 1b: next request for d occurs before d is evicted.



Reduced eviction schedules

Claim. Given any unreduced schedule S, can transform it into a reduced schedule S' with no more evictions.

Pf. [by induction on number of steps j]

- Suppose S brings d into the cache in step j even though d is in cache.
- Let c be the item S evicts when it brings d into the cache.
- Case 2a: d evicted before it is needed.

unreduced schedule S

		d_1	a	c	
		d_1	a	c	
		d_1	a	$\boxed{c}$	
step j	d	d_1	a	d_3	d_3 enters cache even though d_1 is already in cache
	d	d_1	a	d_3	d_3 not needed
	c	c	a	d_3	
step j'	b	c	a	b	d_3 evicted
	d	c	a	d_3	d_3 needed

S'

	d_1	a	c	
	d_1	a	c	
	d_1	a	c	
d	d_1	a	c	might as well leave c in cache until d_3 in evicted
d	d_1	a	c	
c	c	a	c	
b	c	a	b	
d	c	a	d_3	

Reduced eviction schedules

Claim. Given any unreduced schedule S, can transform it into a reduced schedule S' with no more evictions.

Pf. [by induction on number of steps j]

- Suppose S brings d into the cache in step j even though d is in cache.
- Let c be the item S evicts when it brings d into the cache.
- Case 2a: d evicted before it is needed.
- Case 2b: d needed before it is evicted.

unreduced schedule S

		d_1	a	c	
		d_1	a	c	
		d_1	a	$\boxed{c}$	
step j	d	d_1	a	d_3	d_3 enters cache even though d_1 is already in cache
	d	d_1	a	d_3	d_3 not needed
	c	c	a	d_3	
	a	c	a	d_3	
step j′	d	c	a	d_3	d_3 needed

S′

	d_1	a	c	
	d_1	a	c	
	d_1	a	c	
d	d_1	a	c	might as well leave c in cache until d_3 in needed
d	d_1	a	c	
c	c	a	c	
a	c	a	c	
d	c	a	d_3	

Reduced eviction schedules

Claim. Given any unreduced schedule S, can transform it into a reduced schedule S' with no more evictions.

Pf. [by induction on number of steps j]

- Case 1: S brings d into the cache in step j without a request. ✔
- Case 2: S brings d into the cache in step j even though d is in cache. ✔
- If multiple unreduced items in step j, apply each one in turn, dealing with Case 1 before Case 2. ▪

resolving Case 1 might trigger Case 2

Farthest-in-future: analysis

Theorem. FF is optimal eviction algorithm.

Pf. Follows directly from the following invariant.

Invariant. There exists an optimal reduced schedule S that has the same eviction schedule as S_{FF} through the first j steps.

Pf. [by induction on number of steps j]

Base case: $j = 0$.

Let S be reduced schedule that satisfies invariant through j steps.

We produce S' that satisfies invariant after $j + 1$ steps.

- Let d denote the item requested in step $j + 1$.
- Since S and S_{FF} have agreed up until now, they have the same cache contents before step $j + 1$.
- Case 1: d is already in the cache. $S' = S$ satisfies invariant.
- Case 2: d is not in the cache and S and S_{FF} evict the same item. $S' = S$ satisfies invariant.

Farthest-in-future: analysis

Pf. [continued]

- Case 3: d is not in the cache; S_{FF} evicts e; S evicts $f \neq e$.
  - begin construction of S' from S by evicting e instead of f

  - now S' agrees with S_{FF} for first $j+1$ steps; we show that having item f in cache is no worse than having item e in cache
  - let S' behave the same as S until S' is forced to take a different action (because either S evicts e; or because either e or f is requested)

Farthest-in-future: analysis

Let j' be the first step after $j+1$ that S' must take a different action from S; let g denote the item requested in step j'.

involves either e or f (or both)

same	e	**step j′**	*same*	f

S **S′**

- Case 3a: $g = e$.
  Can't happen with FF since there must be a request for f before e.

  S' agrees with S_{FF} through first $j+1$ steps

- Case 3b: $g = f$.
  Element f can't be in cache of S; let e' be the item that S evicts.
  - if $e' = e$, S' accesses f from cache; now S and S' have same cache
  - if $e' \neq e$, we make S' evict e' and bring e into the cache; now S and S' have the same cache

  We let S' behave exactly like S for remaining requests.

S' is no longer reduced, but can be transformed into a reduced schedule that agrees with FF through first $j+1$ steps

Farthest-in-future: analysis

Let j' be the first step after $j + 1$ that S' must take a different action from S;
let g denote the item requested in step j'.

involves wither e or f (or both)

same	e	**step j′**	*same*	f

S **S′**

otherwise S' could have taken the same action

- Case 3c: $g \neq e, f$. S evicts e.
  - make S' evict f.

same	g	**step j′**	*same*	g

S **S′**

  - now S and S' have the same cache
  - let S' behave exactly like S for the remaining requests ▪