# Optimal Cache Replacement

# Optimal Offline Caching

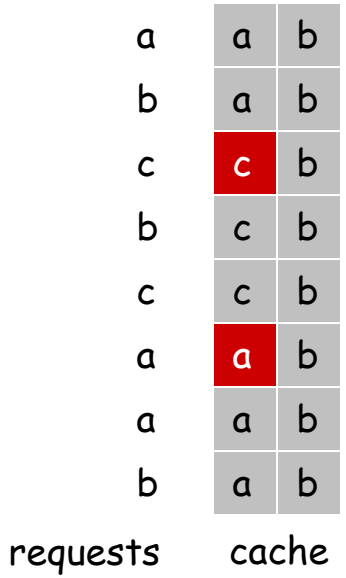
Caching.

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

Goal. Eviction schedule that minimizes number of cache misses.

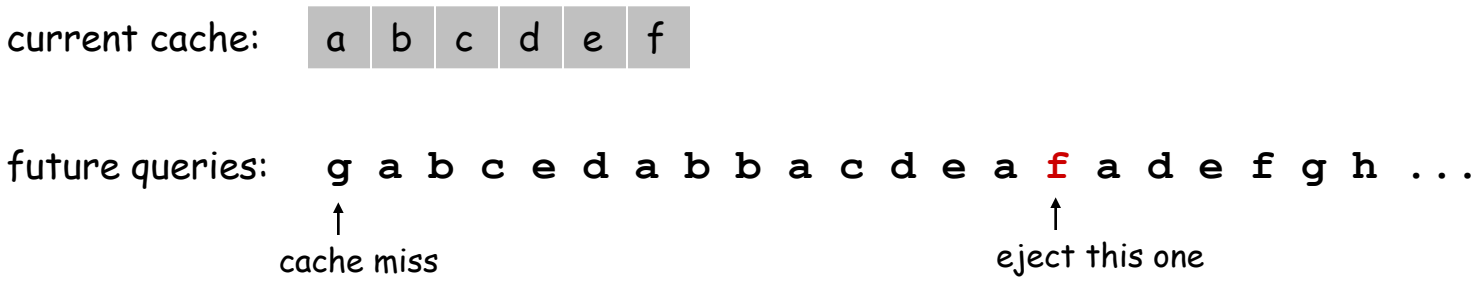
Ex: k = 2, initial cache = ab,
requests: a, b, c, b, c, a, a, b.

Optimal eviction schedule: 2 cache misses.



# Optimal Offline Caching: Farthest-In-Future

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




Theorem. [Bellady, 1960s] FF is optimal eviction schedule.

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

# Reduced Eviction Schedules

Def. A reduced schedule is a schedule that only inserts an item into the cache in a step in which that item is requested.

Intuition. Can transform an unreduced schedule into a reduced one with no more cache misses.

| | | | |
|---|---|---|---|
| a | a | b | c |
| a | a | x | c |
| c | a | d | c |
| d | a | d | b |
| a | a | c | b |
| b | a | x | b |
| c | a | c | b |
| a | a | b | c |
| a | a | b | c |

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 |
| a | a | c | b |
| a | a | c | b |

a reduced schedule

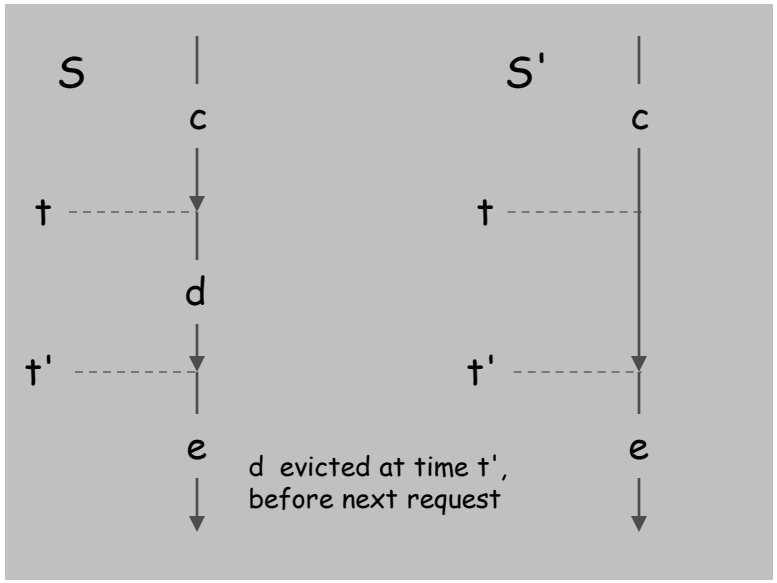
# Reduced Eviction Schedules

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

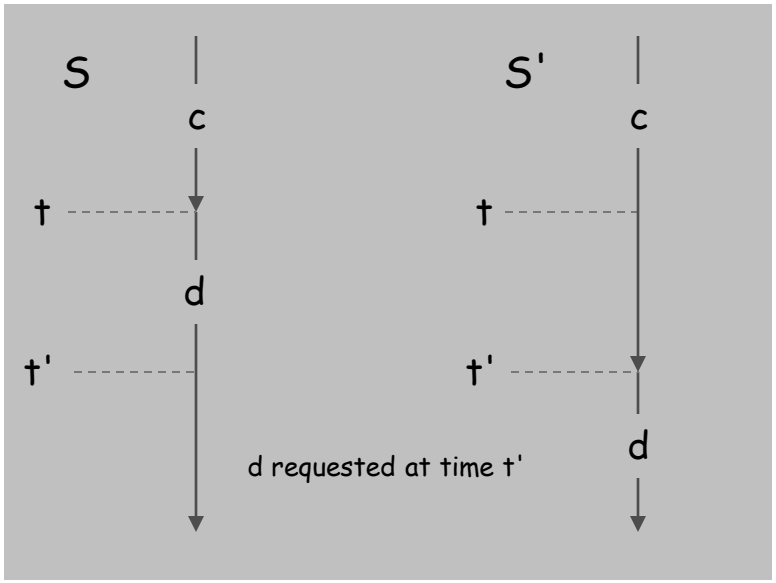
Pf. (by induction on number of unreduced items)

doesn't enter cache at requested time

- Suppose S brings d into the cache at time t, without a request.
- Let c be the item S evicts when it brings d into the cache.
- Case 1: d evicted at time t', before next request for d.
- Case 2: d requested at time t' before d is evicted. ▪




Case 1




Case 2

# Farthest-In-Future: Analysis

Theorem. FF is optimal eviction algorithm.

Pf. (by induction on number or requests j)

> Invariant: There exists an optimal reduced schedule S that makes the same eviction schedule as $S_{FF}$ through the first j+1 requests.

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

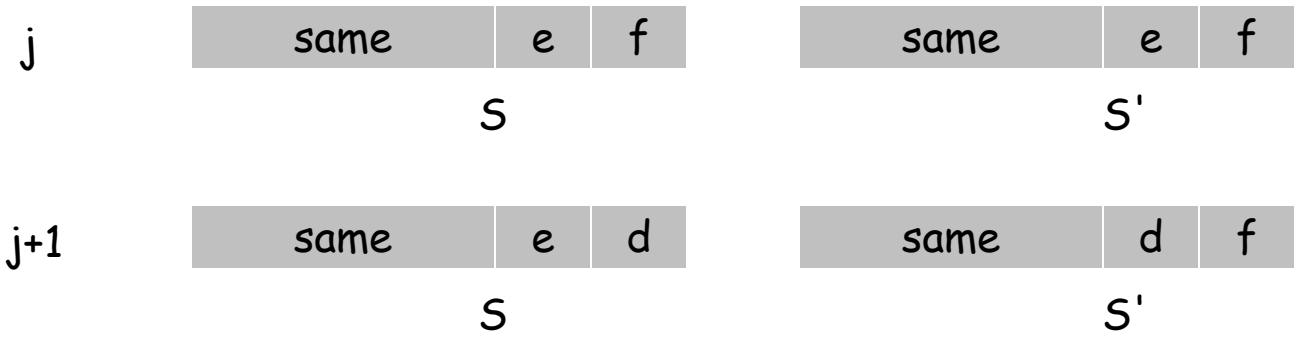
We produce S' that satisfies invariant after j+1 requests.

- Consider $(j+1)^{st}$ request $d = d_{j+1}$.
- Since S and $S_{FF}$ have agreed up until now, they have the same cache contents before request 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 element). 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 ≠ e).
  - begin construction of S' from S by evicting e instead of f

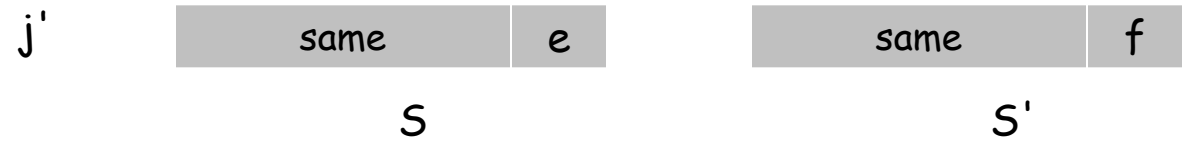



  - now S' agrees with $S_{FF}$ on first j+1 requests; we show that having element f in cache is no worse than having element e

# Farthest-In-Future: Analysis

Let j' be the first time after j+1 that S and S' take a different action, and let g be item requested at time j'.

must involve e or f (or both)




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

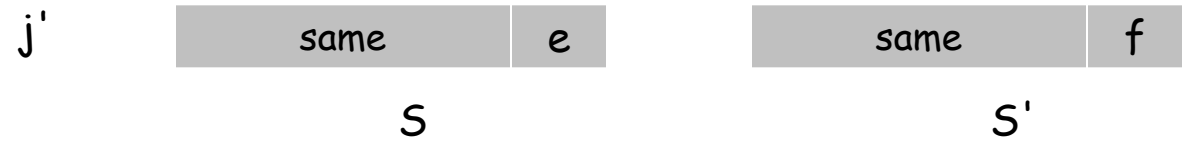
- Case 3b: g = f. Element f can't be in cache of S, so let e' be the element that S evicts.
  - if e' = e, S' accesses f from cache; now S and S' have same cache
  - if e' ≠ e, S' evicts e' and brings e into the cache; now S and S' have the same cache

Note: S' is no longer reduced, but can be transformed into a reduced schedule that agrees with $S_{FF}$ through step j+1

# Farthest-In-Future: Analysis

Let j' be the first time after j+1 that S and S' take a different action, and let g be item requested at time j'.

must involve e or f (or both)




otherwise S' would take the same action

- Case 3c: $g \neq e, f$. S must evict e.
  Make S' evict f; now S and S' have the same cache. ▪

# Caching Perspective

Online vs. offline algorithms.

- Offline: full sequence of requests is known a priori.
- Online (reality): requests are not known in advance.
- Caching is among most fundamental online problems in CS.

LIFO. Evict page brought in most recently.

LRU. Evict page whose most recent access was earliest.




Theorem. FF is optimal offline eviction algorithm.

- Provides basis for understanding and analyzing online algorithms.
- LRU is k-competitive.
- LIFO is arbitrarily bad.