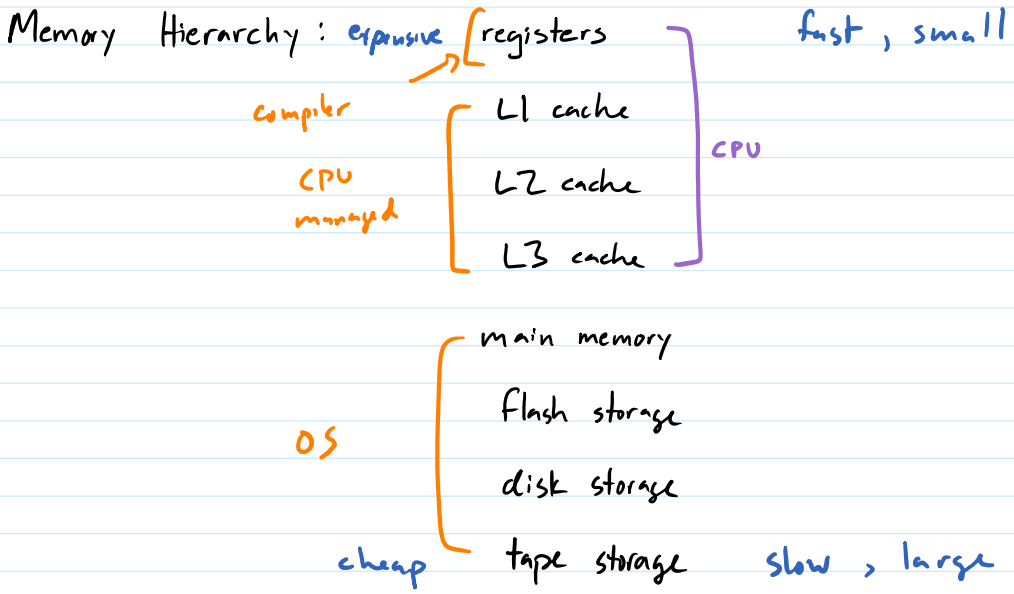
Optimal Caching



caching: keep soon-to-be-accessed data in fast memory

spatial locality - if use data, likely to use nearby data soon

temporal locality - things used recently likely to be used in near future

Optimal caching: given cache size k, sequence of requests $d_1, \ldots, d_m$ for data items, find eviction schedule to minimize cache loads

↳ collection of (x, y, t) meaning x loaded, y removed at time t

Ex: $k=2$, initial cache = 1, 2 requests 1, 2, 3, 2, 1, 3, 4, 2, 3, 1 ,6,7,6,7,1,3

Farthest in Future (optimal)

1 → 3
2 → 1 → 4 → 2 → 1

Least Recently Used

1 → 3 → 1
2 → 3 → ...

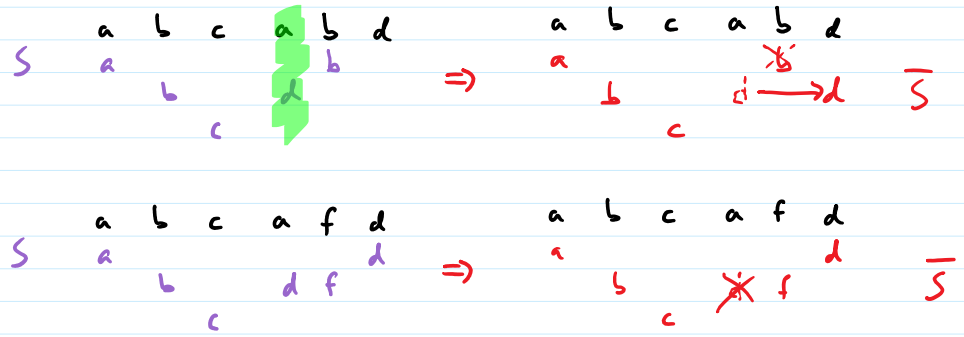
For each request for data d
  if not in cache
    evict item x in cache needed farthest in future, replace with d

INVARIANT: after j requests, S_{FF} is same as some reduced optimal schedule

Key Lemma: if S_{FF} is same as reduced schedule S after j requests, then S_{FF} is same as some reduced schedule S' after $j+1$, where S' has no more cache loads than S

reduced schedule: y brought into cache at time $t \rightarrow y$ requested at time t

For any schedule S, there is a reduced schedule $\bar{S}$ with no more cache loads than S



Let S^* be some reduced optimal schedule

INVARIANT: Basis: after 0 requests, all schedules are same, so S_{FF} is same as all opt reduced schd

Induction: suppose after j req, S_{FF} same as S^* where S^* is opt, by Lemma, $\exists$ S' s.t. S_{FF} is same as S' on $j+1$ requests, and S' is as good as S^*, so S' is opt too.

Termination: if same as some opt on all requests, then S_{FF} = that opt soln, so is opt too

Key Lemma: if S_{FF} is same as reduced schedule S after j requests, then S_{FF} is same as some reduced schedule S' after $j+1$ requests, where S' has no more cache loads than S

Proof: Suppose S_{FF} is same as reduced schedule S after j requests

Case 1: $j+1$ is request for something in cache

S_{FF} does nothing $\quad S' = S$

Case 1: $j+1$ is request for something in cache

S_{FF} does nothing
S does nothing
$S' = S$

Case 2: $j+1$ is cache miss on request d, both S, S_{FF} evict same thing e

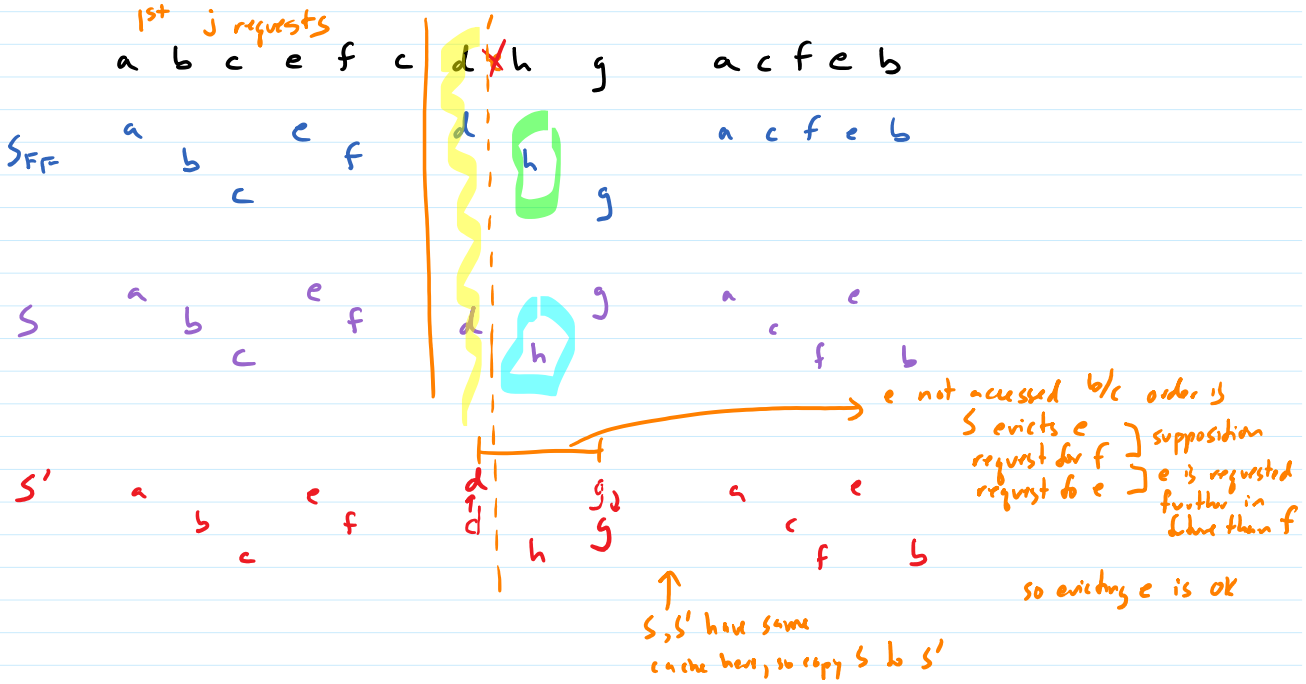
$S' = S$

Case 3: $j+1$ is cache miss on request d, S_{FF} evicts e, S evicts f, $e \neq f$

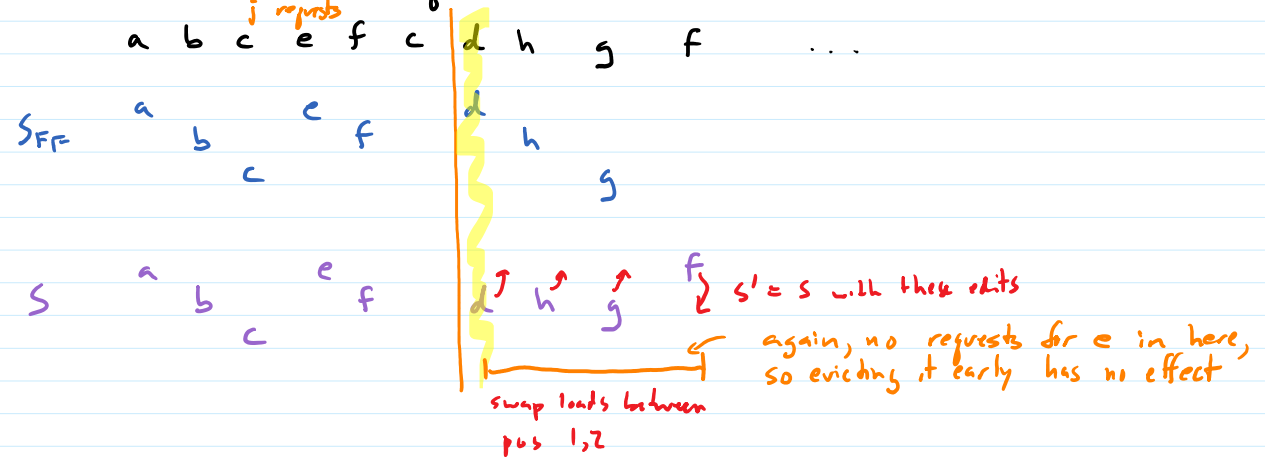
all possibilities for order of S evicting e and an access to f

a) S evicts e on request for $g \neq e, f$ before f requested again
b) S evicts e on a request for f and S evicted d before that
c) S evicts e on a request for f and d still in cache
d) request for f before S evicts e; S evicts e' and had evicted d before
e) request for f before S evicts e; S evicts e' and d still in cache

a) S evicts e on request for $g \neq e, f$ before f requested again

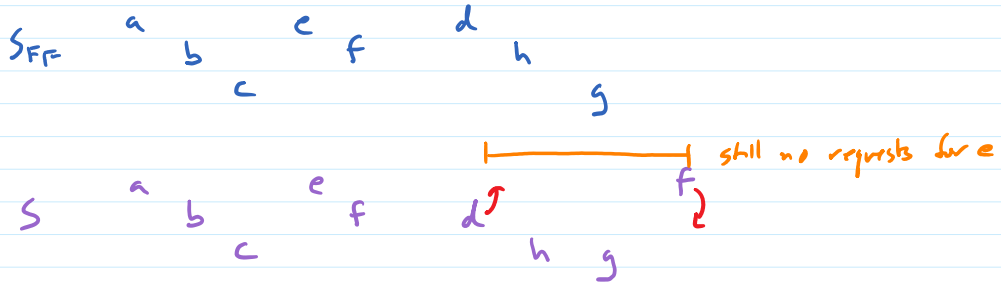


b) S evicts e on a request for f and S evicted d before that



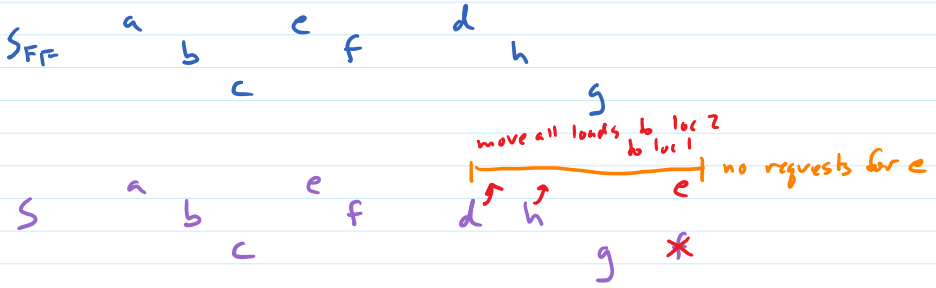
c) request for f when S evicts e and d still in cache

a b c e f c d h g . f . .



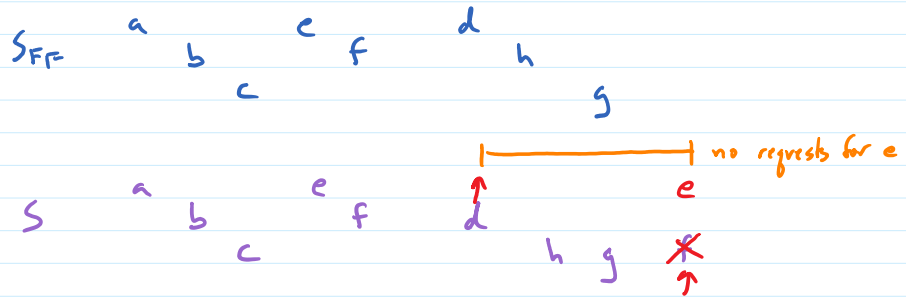
d) request for f before S evicts e, S evicts $e' \neq e$, d previously evicted

a b c e f c d h g f . . .



e) request for f before S evicts e, S evicts $e' \neq e$, d still in cache

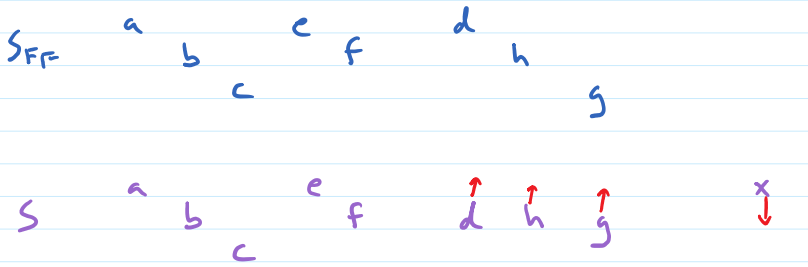
a b c e f c d h g f . . .



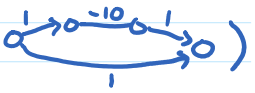
not reduced... so find reduced S' that has no more cache loads

f) f never requested again (so e never requested again)

a b c e f c d h g x

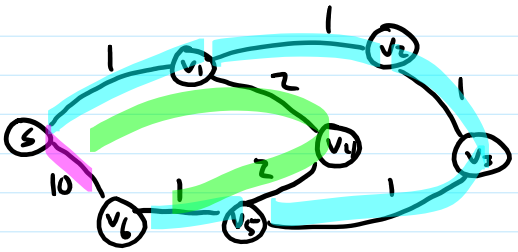


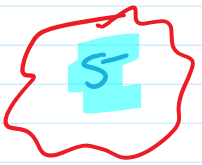
Shortest Paths

weights non-negative (can't just add constant to eliminate neg weights: 

)

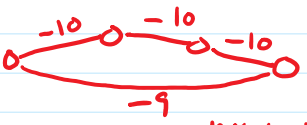
Given weighted G (directed or undirected), and a source vertex s, find min-weight path $s \rightsquigarrow v$ for all vertices v.





C

1δ



neg weight cycle (so no min-cost path)

c)